

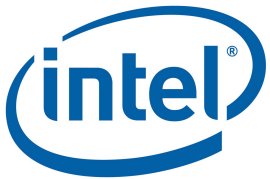
# CAMEO

## A CACHE-LIKE MEMORY ORGANIZATION FOR 3D MEMORY SYSTEMS

*Chiachen Chou, Georgia Tech, [cc.chou@ece.gatech.edu](mailto:cc.chou@ece.gatech.edu)*

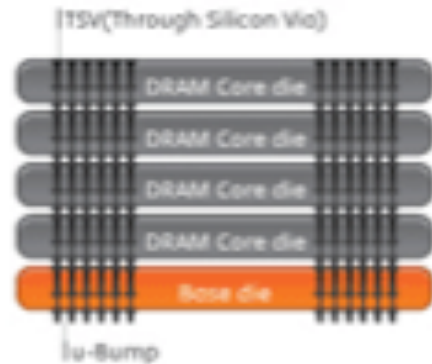
*Aamer Jaleel, Intel, [aamer.jaleel@intel.com](mailto:aamer.jaleel@intel.com)*

*Moinuddin K. Qureshi, Georgia Tech, [moin@ece.gatech.edu](mailto:moin@ece.gatech.edu)*

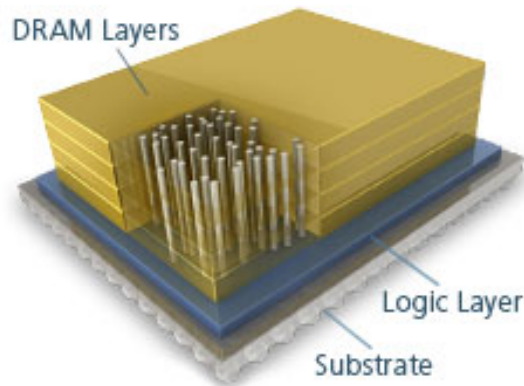


# ARCHITECTING 3D MEMORY SYSTEMS

3D-memory can overcome the bandwidth wall



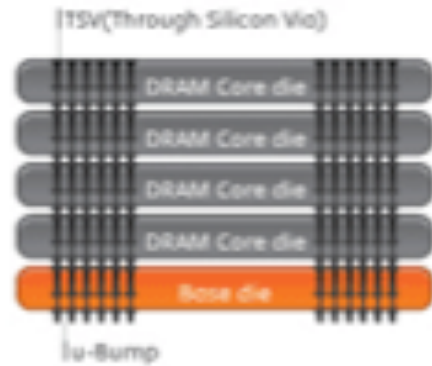
High Bandwidth Memory



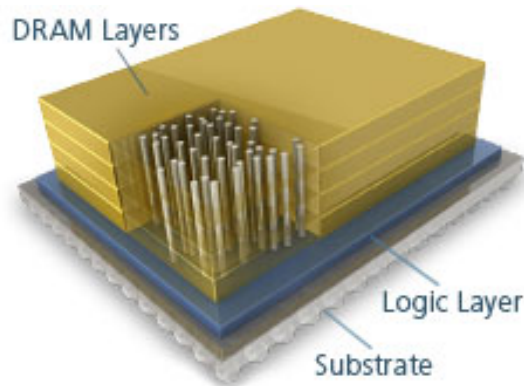
Hybrid Memory Cube

	<b>Stacked DRAM</b>
Bandwidth	2-8X
Latency	0.5-1X

# ARCHITECTING 3D MEMORY SYSTEMS

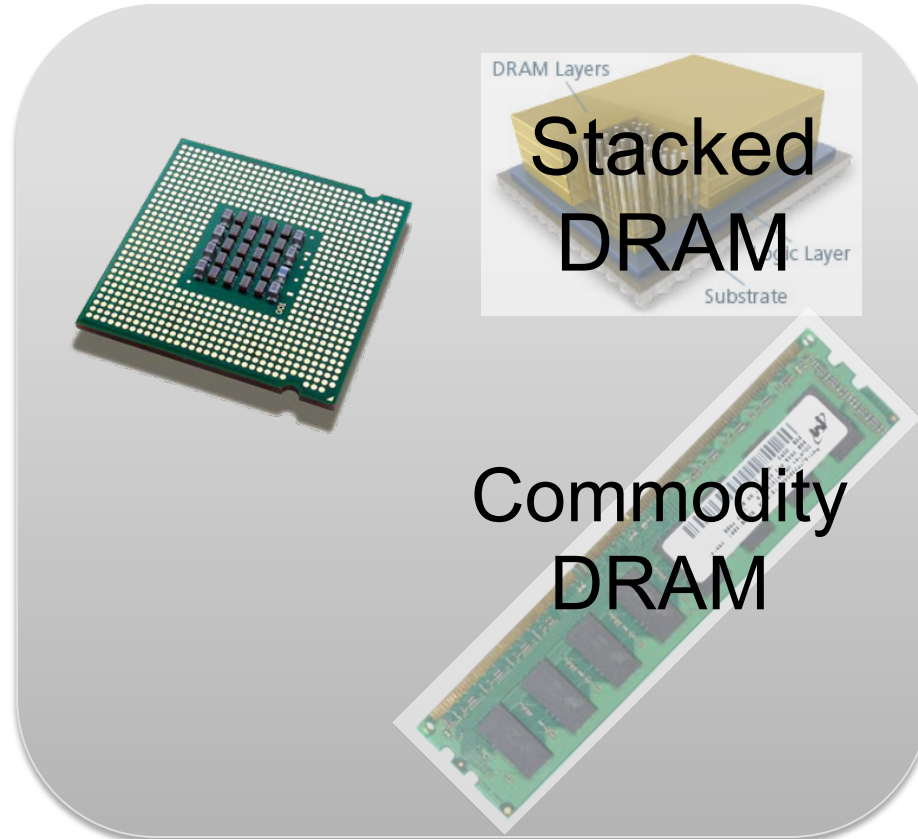


**Few GB**



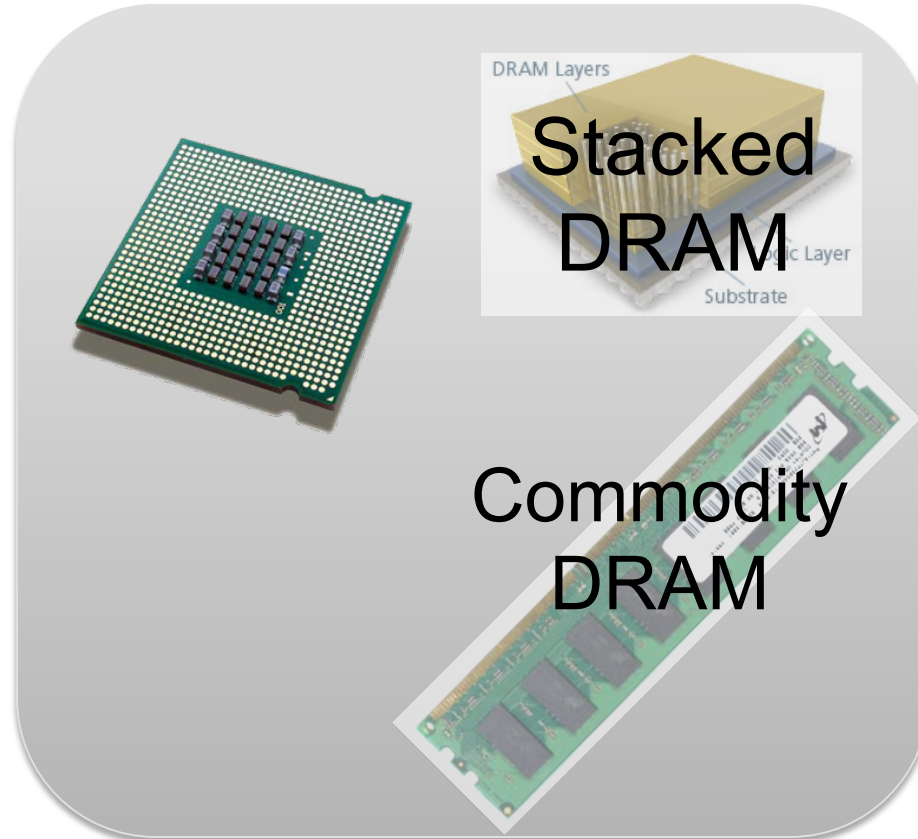
	<b>Stacked DRAM</b>
Bandwidth	2-8X
Latency	0.5-1X
Capacity	0.25X

# ARCHITECTING 3D MEMORY SYSTEMS



## Hybrid Memory System

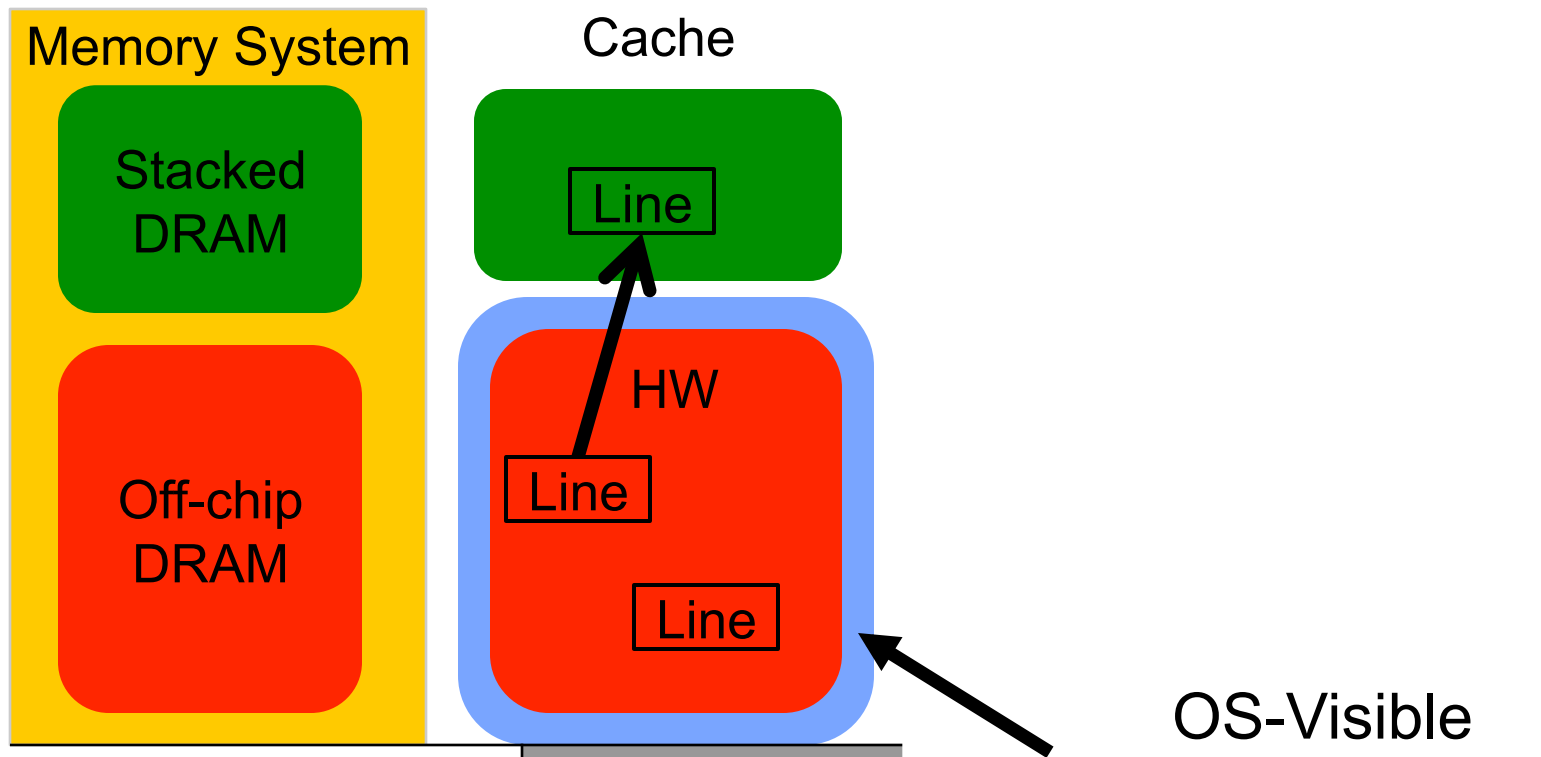
# ARCHITECTING 3D MEMORY SYSTEMS



## Hybrid Memory System

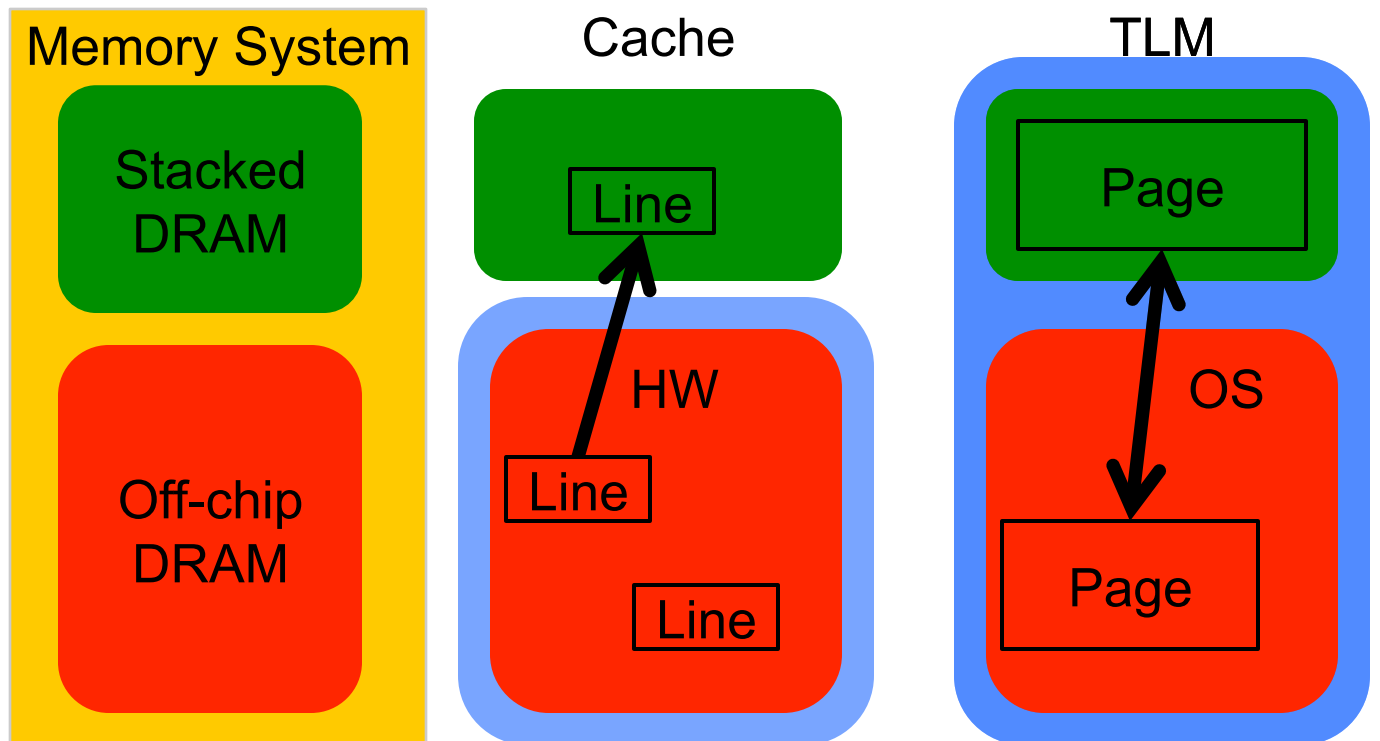
**How to use Stacked DRAM: Cache or Memory?**

# CAMEO FOR HYBRID MEMORY



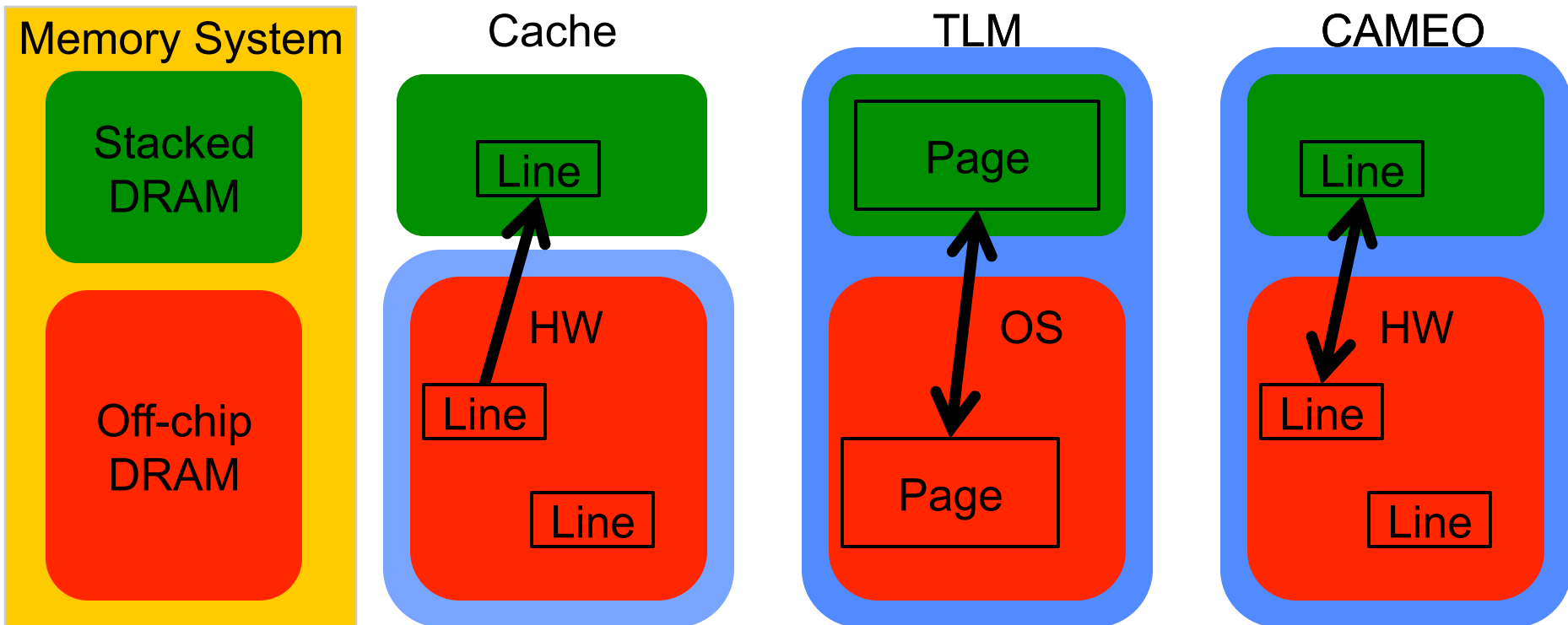
	Cache
Need OS Support	No ✓
Data Transfer @	64B ✓
Memory Capacity	No 3D ✗
Speedup	50%

# CAMEO FOR HYBRID MEMORY



	Cache	TLM
Need OS Support	No ✓	Yes ✗
Data Transfer @	64B ✓	4KB ✗
Memory Capacity	No 3D ✗	Plus 3D ✓
Speedup	50%	50%

# CAMEO FOR HYBRID MEMORY



	Cache	TLM	CAMEO
Need OS Support	No ✓	Yes ✗	No ✓
Data Transfer @	64B ✓	4KB ✗	64B ✓
Memory Capacity	No 3D ✗	Plus 3D ✓	Plus 3D ✓
Speedup	50%	50%	78%



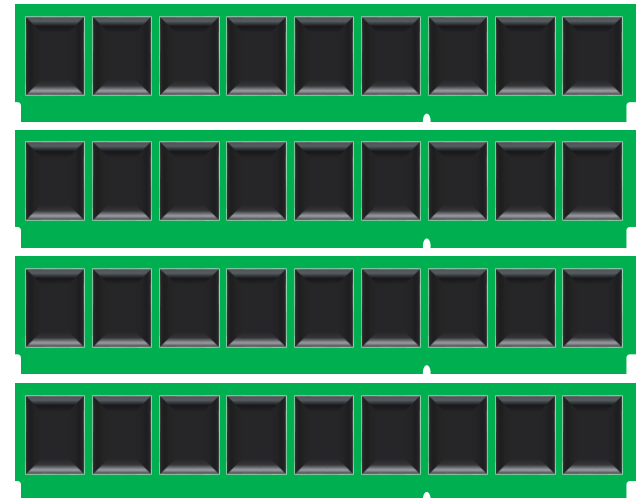
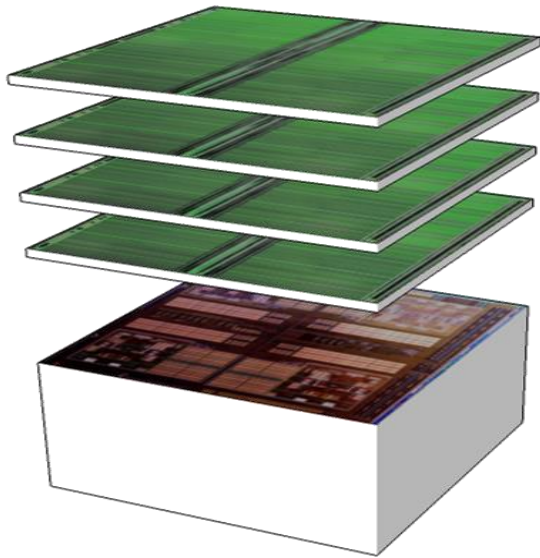
*next paper*

# Transparent Hardware Management of Stacked DRAM as Part of Memory

Jaewoong Sim<sup>1</sup> Alaa R. Alameldeen<sup>2</sup> Zeshan Chishti<sup>2</sup>  
Chris Wilkerson<sup>2</sup> Hyesoon Kim<sup>1</sup>

<sup>1</sup>Georgia Institute of Technology

<sup>2</sup>Intel Labs

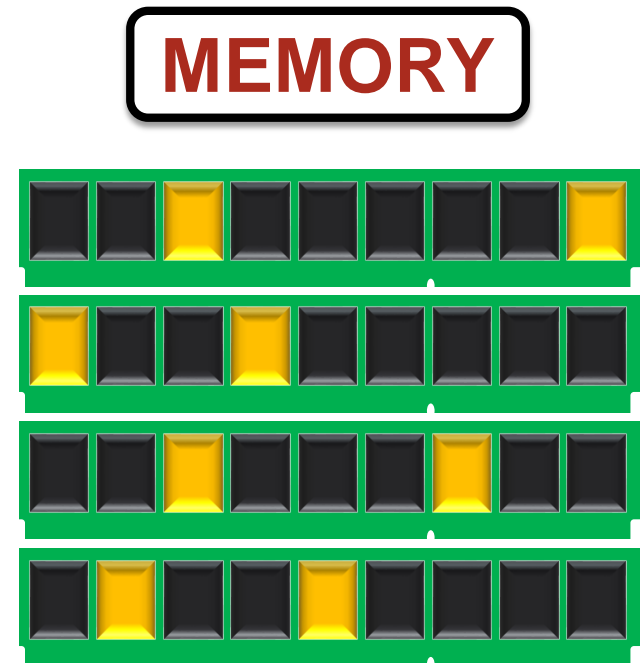
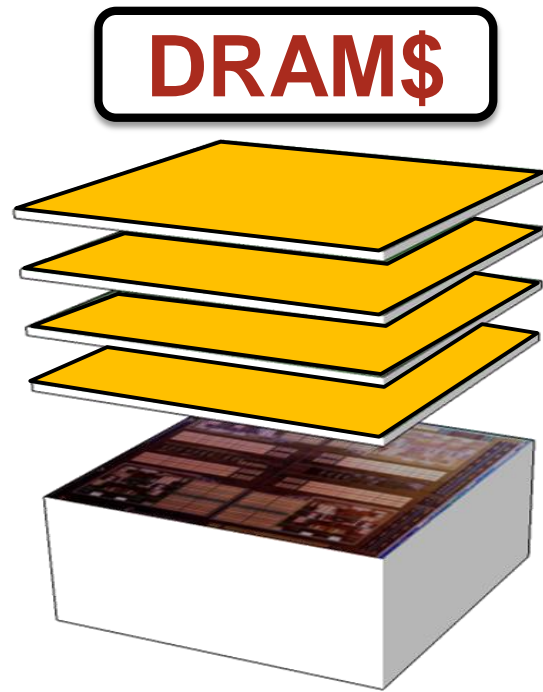


# Transparent Hardware Management of Stacked DRAM as Part of Memory

Jaewoong Sim<sup>1</sup> Alaa R. Alameldeen<sup>2</sup> Zeshan Chishti<sup>2</sup>  
Chris Wilkerson<sup>2</sup> Hyesoon Kim<sup>1</sup>

<sup>1</sup>Georgia Institute of Technology

<sup>2</sup>Intel Labs



# Transparent Hardware Management of Stacked DRAM as **Part of Memory**

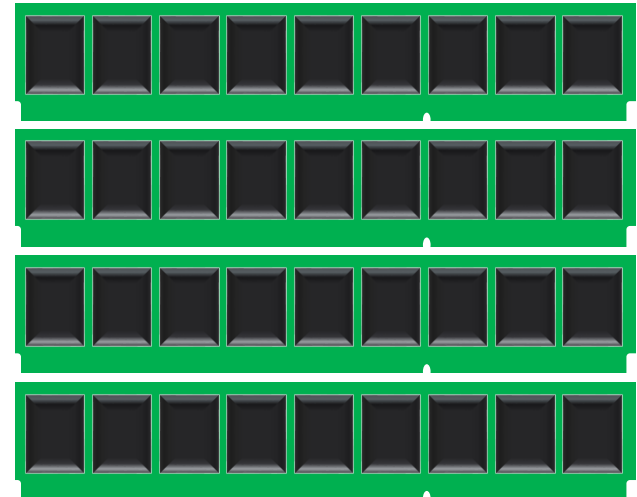
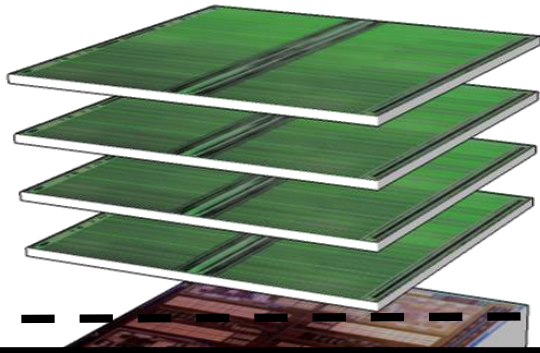
Jaewoong Sim<sup>1</sup> Alaa R. Alameldeen<sup>2</sup> Zeshan Chishti<sup>2</sup>  
Chris Wilkerson<sup>2</sup> Hyesoon Kim<sup>1</sup>

<sup>1</sup>Georgia Institute of Technology

<sup>2</sup>Intel Labs

**FAST**

**SLOW**



Not DRAM\$ 😊

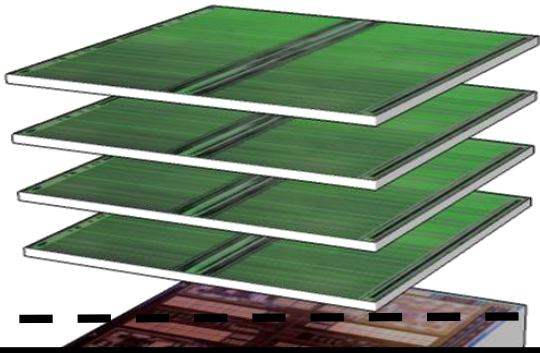
Now, **Part of Memory!!**

# Transparent Hardware Management of Stacked DRAM as **Part of Memory**

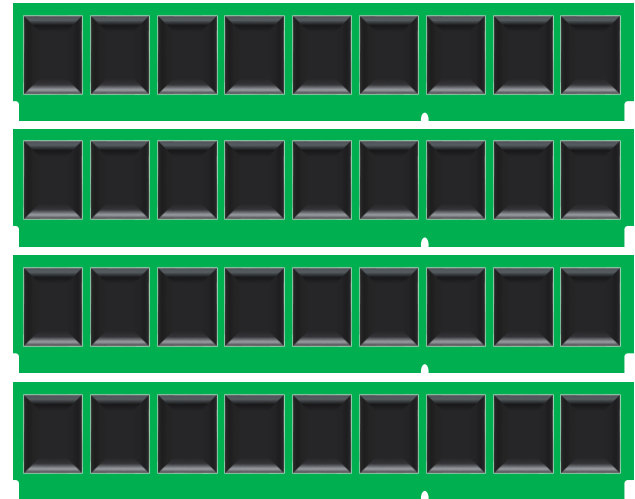
Goal: Enable **Hardware-Managed PoM!**

1. Data Migration
2. Memory Integrity

**FAST**



**SLOW**



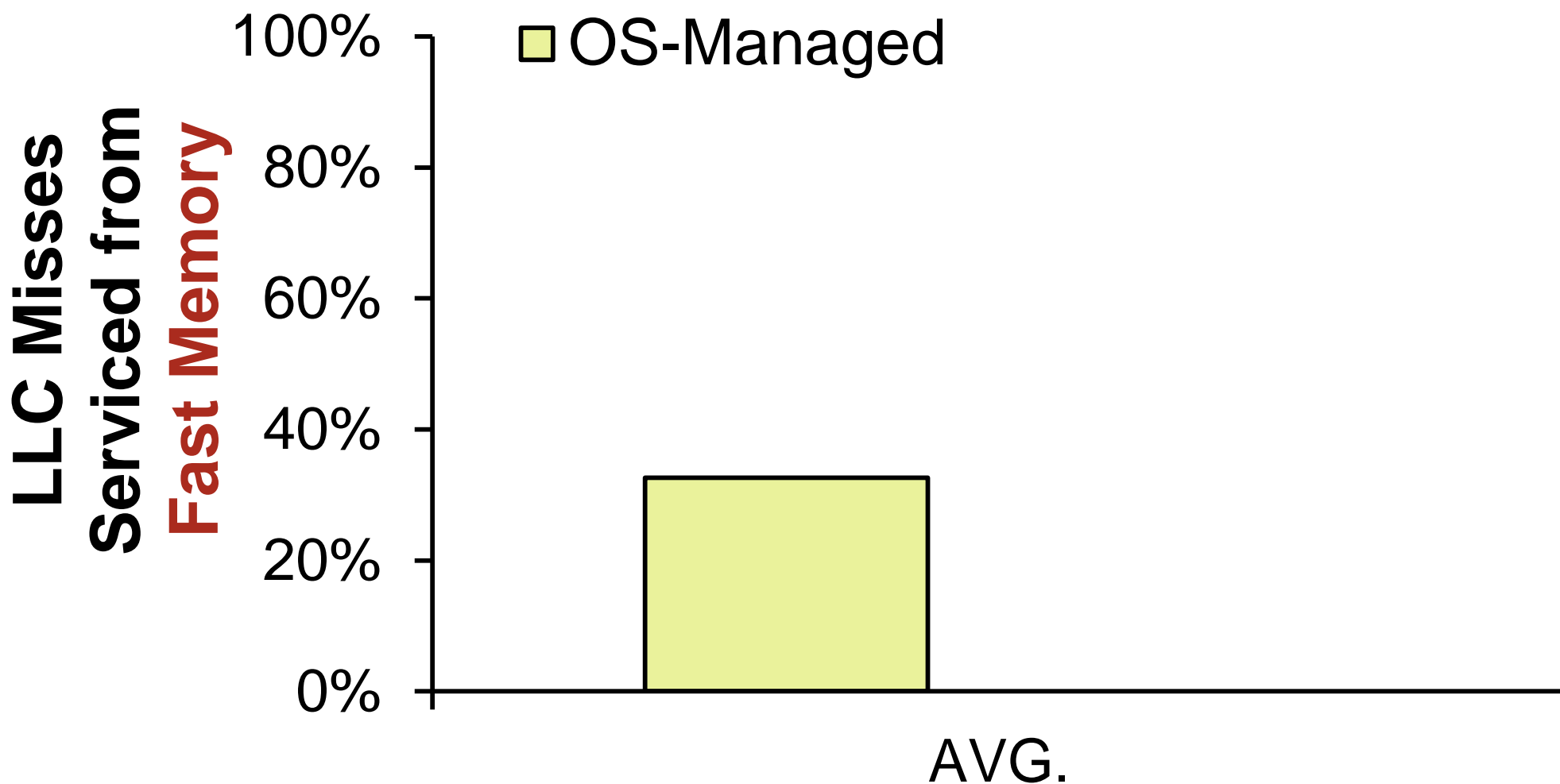
Not DRAM\$ 😊

Now, **Part of Memory!!**

# Stacked DRAM as PoM

---

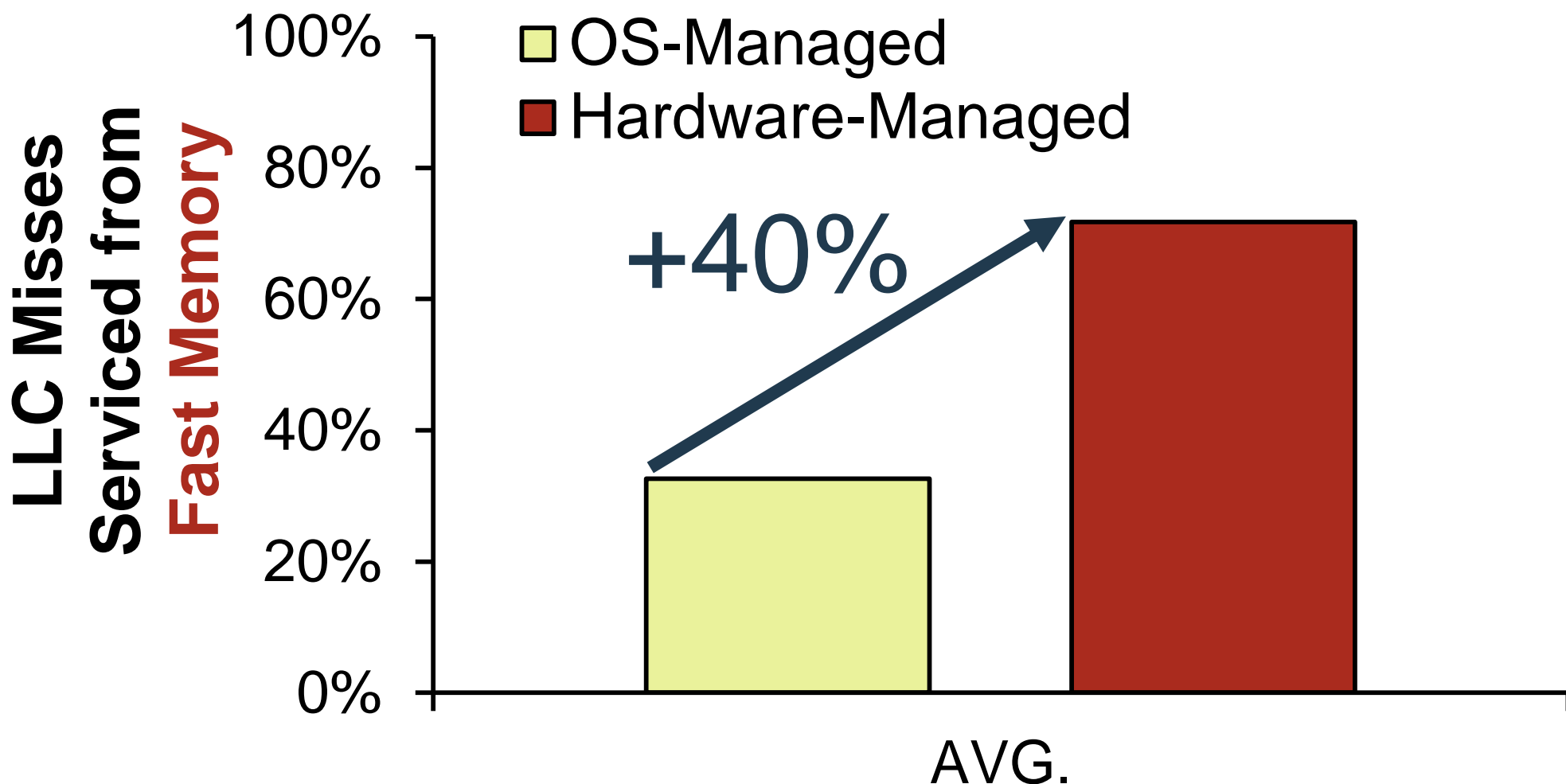
## Why Hardware?



# Stacked DRAM as PoM

## Why Hardware?

Adapt & Remap data  
at a fine granularity!



# Hardware-Managed PoM

---

What Challenges?

Metadata for GBs of Memory!

## Size & Latency

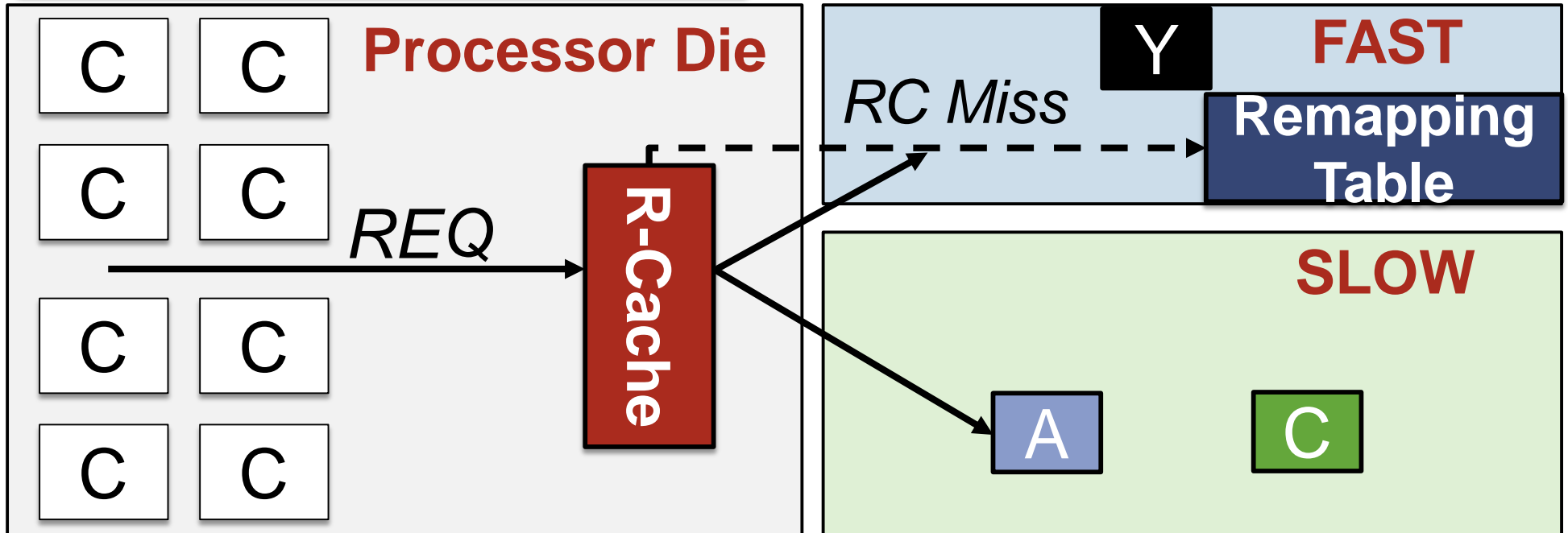
Remapping Table

Memory Utilization Tracking Structure

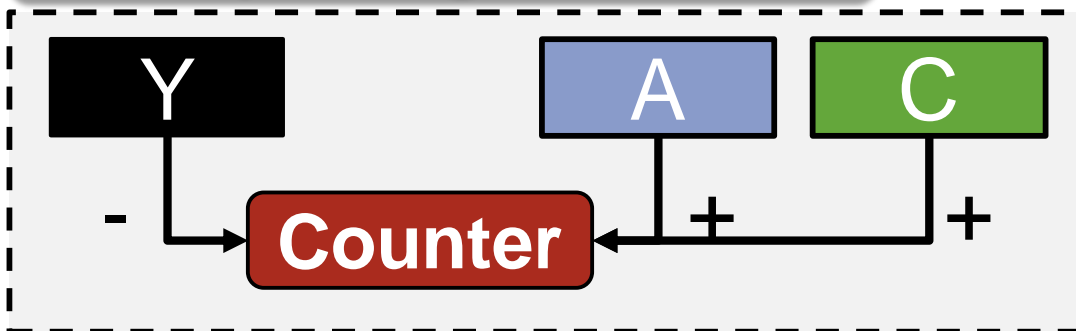


# A Practical PoM Architecture

## Two-Level Indirection



## Competing Counter



**Session 1A:**  
**Stacked DRAM**  
**Today at**  
**13:25PM!!**

*next paper*

**TODAY @2:15pm, Session 1A**

# Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache

Djordje Jevdjic

Cansu Kaynak

Gabriel H. Loh

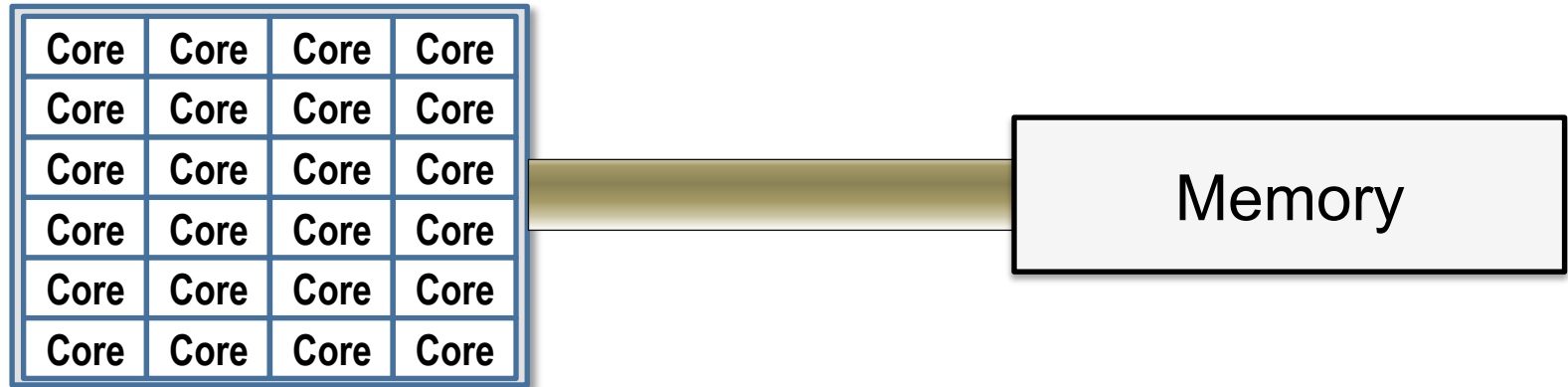
Babak Falsafi



# Server Trends



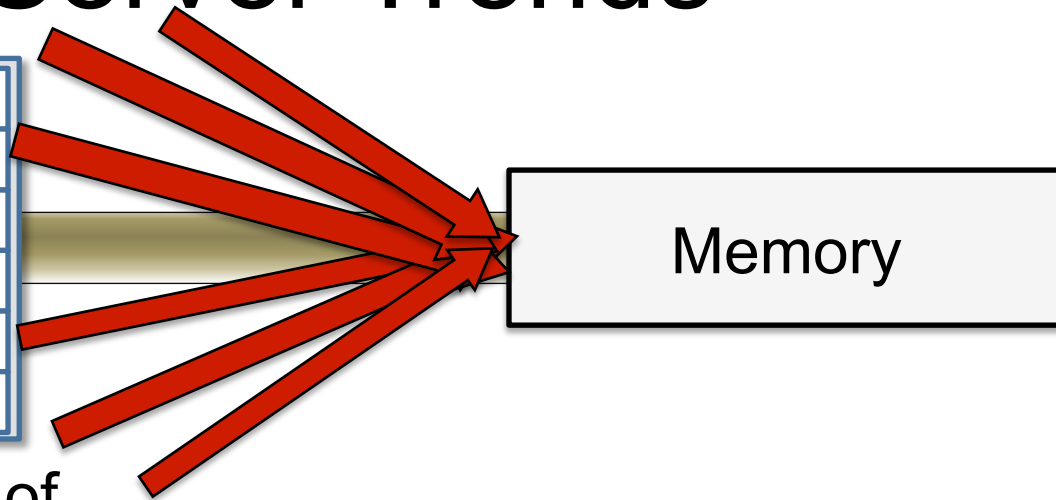
# Server Trends



Tens to hundreds of  
cores & accelerators!

# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



Tens to hundreds of  
cores & accelerators!

# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core

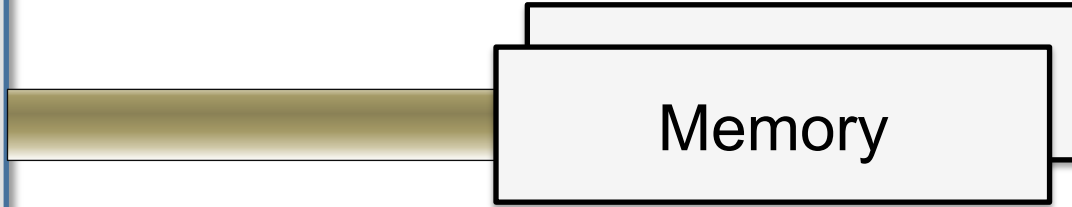


In-memory big data! 100s of GBs

Tens to hundreds of  
cores & accelerators!

# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



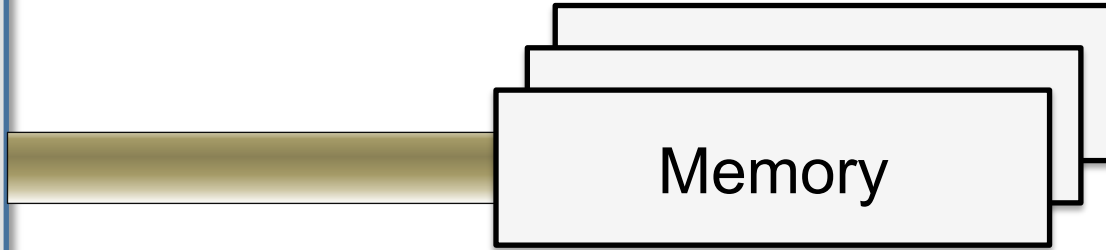
In-memory big data! 100s of GBs

Tens to hundreds of  
cores & accelerators!



# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



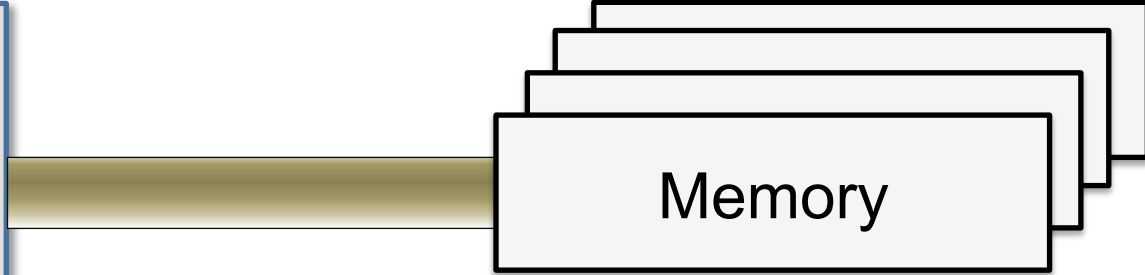
In-memory big data! 100s of GBs

Tens to hundreds of  
cores & accelerators!

# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core

Tens to hundreds of cores & accelerators!



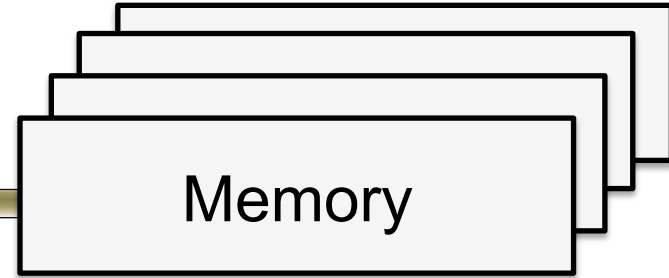
In-memory big data! 100s of GBs

Many DIMMs per channel  
Capacity/BW tradeoff

# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core

Tens to hundreds of cores & accelerators!

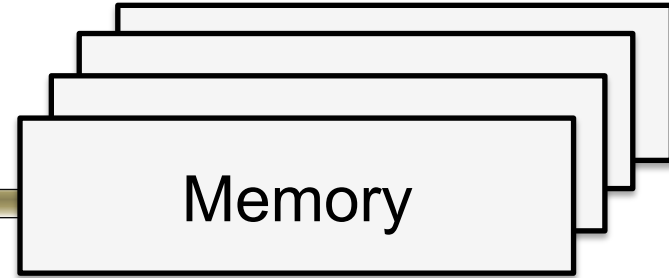


In-memory big data! 100s of GBs

Many DIMMs per channel  
Capacity/BW tradeoff

# Server Trends

Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



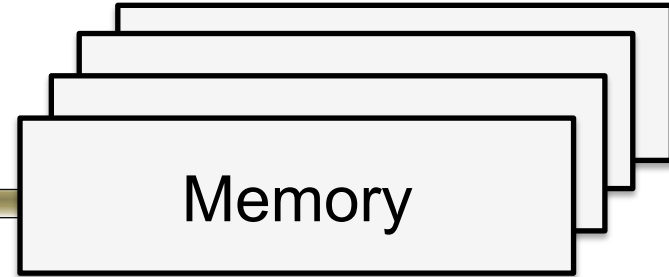
Tens to hundreds of cores & accelerators!

In-memory big data! 100s of GBs

Many DIMMs per channel  
Capacity/BW tradeoff

# Server Trends

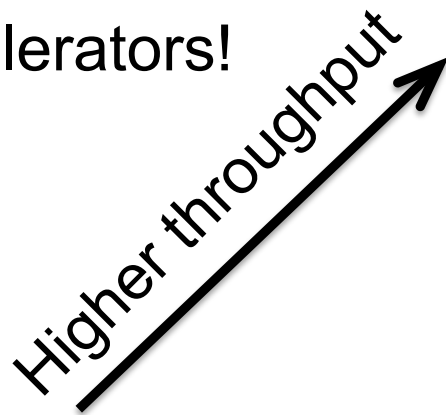
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



In-memory big data! 100s of GBs

Many DIMMs per channel  
Capacity/BW tradeoff

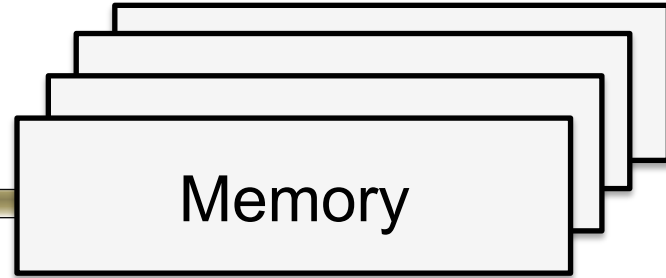
Tens to hundreds of  
cores & accelerators!



**Servers Drive Into Bandwidth Wall**

# Server Trends

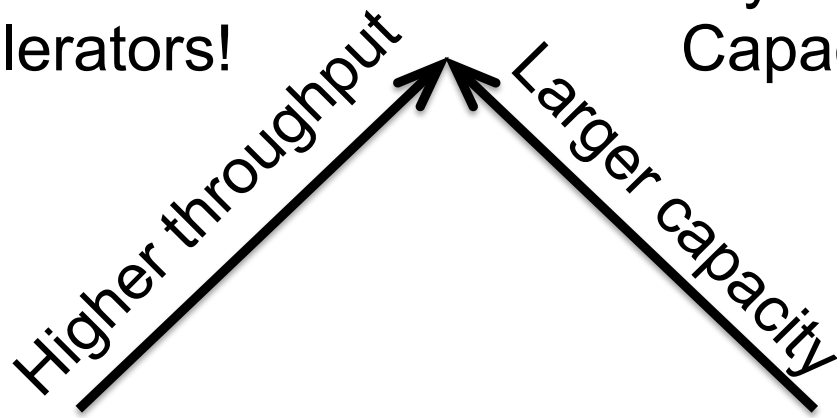
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



In-memory big data! 100s of GBs

Tens to hundreds of cores & accelerators!

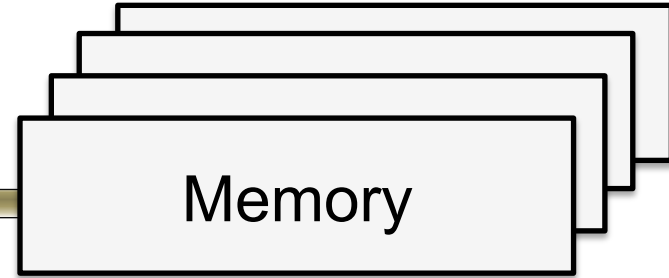
Many DIMMs per channel  
Capacity/BW tradeoff



**Servers Drive Into Bandwidth Wall**

# Server Trends

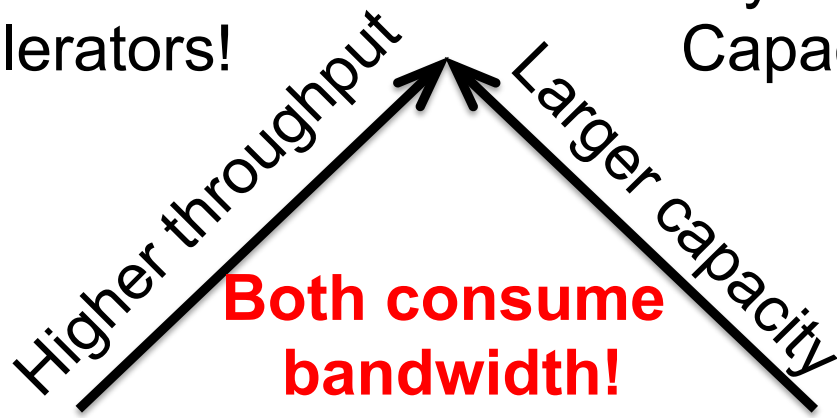
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



In-memory big data! 100s of GBs

Tens to hundreds of cores & accelerators!

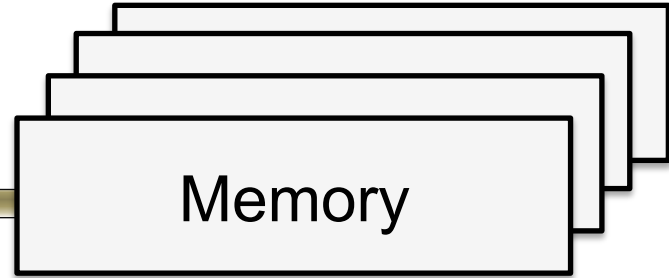
Many DIMMs per channel  
Capacity/BW tradeoff



**Servers Drive Into Bandwidth Wall**

# Server Trends

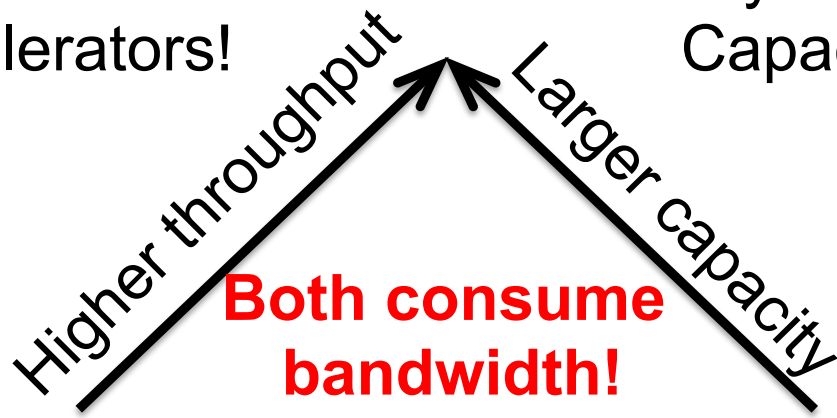
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core
Core	Core	Core	Core



In-memory big data! 100s of GBs

Tens to hundreds of cores & accelerators!

Many DIMMs per channel  
Capacity/BW tradeoff



**Corner!**

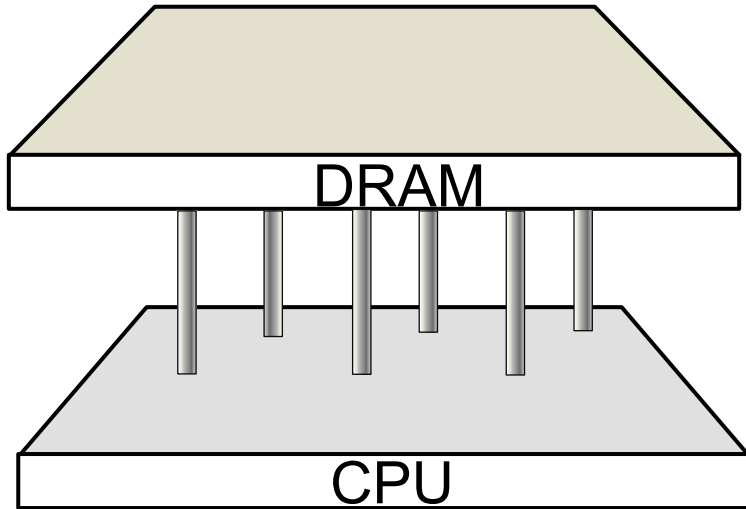
**Servers Drive Into Bandwidth Wall**



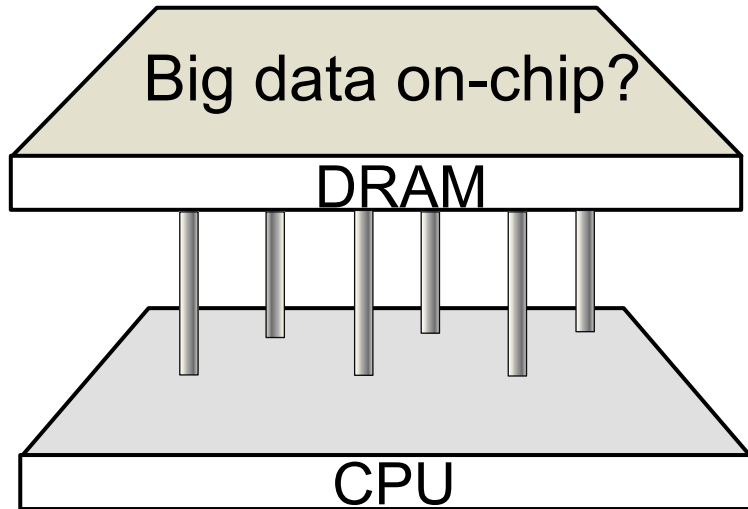




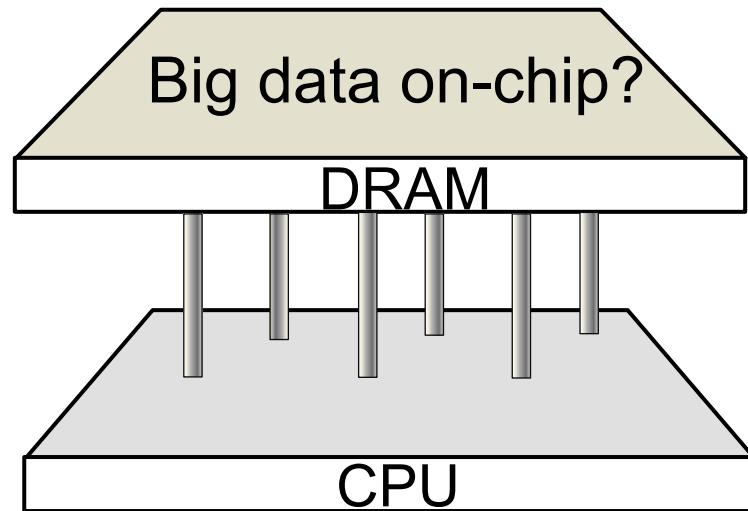
# Die-Stacked DRAM Caches



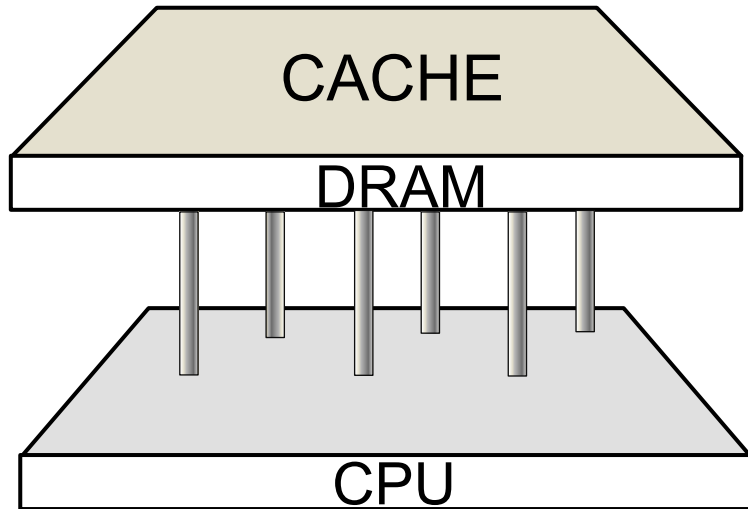
# Die-Stacked DRAM Caches



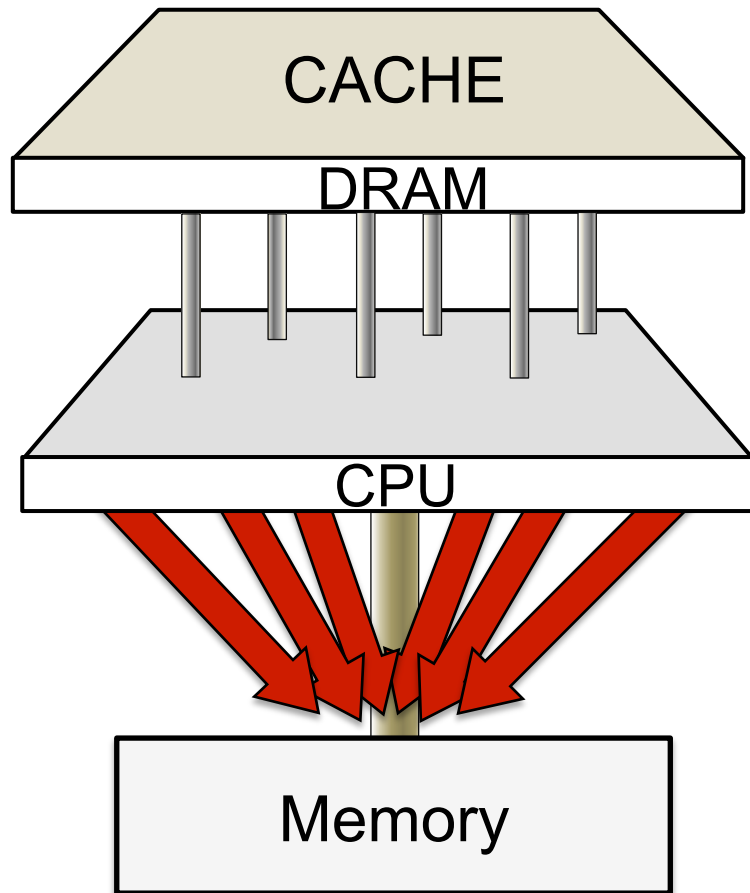
# Die-Stacked DRAM Caches



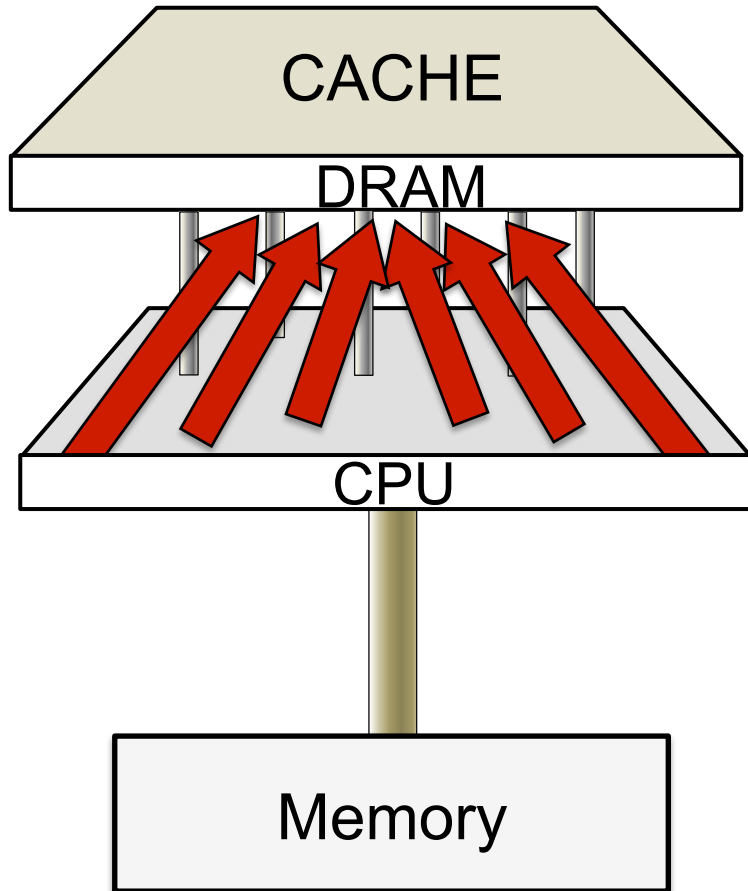
# Die-Stacked DRAM Caches



# Die-Stacked DRAM Caches

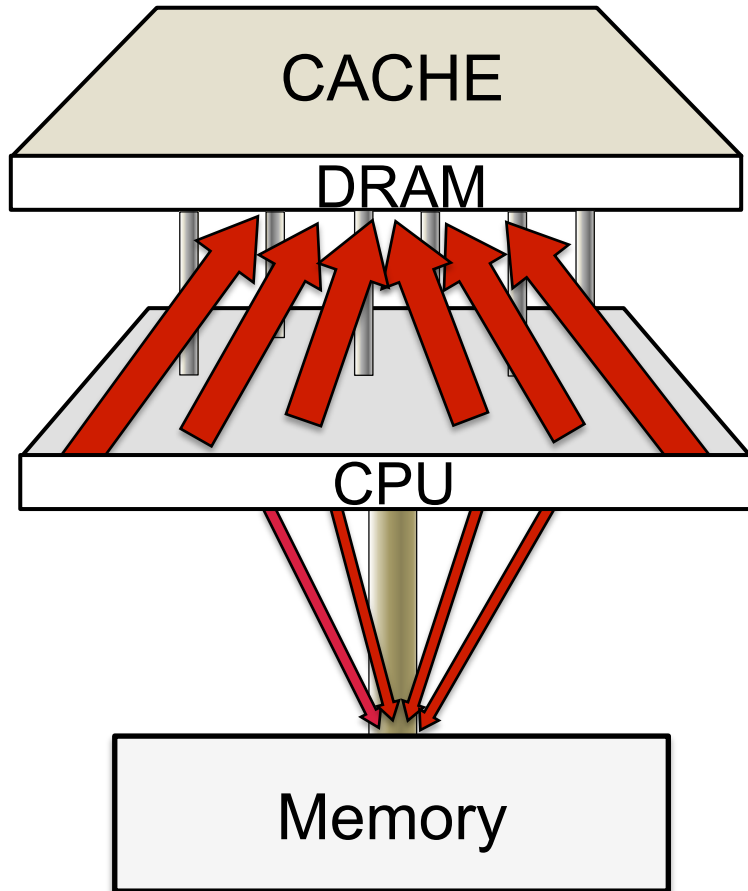


# Die-Stacked DRAM Caches

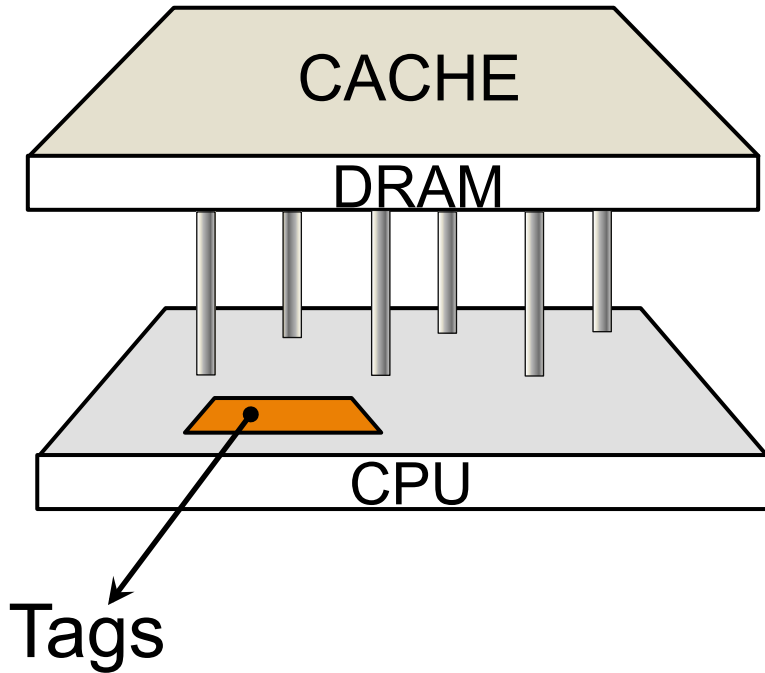




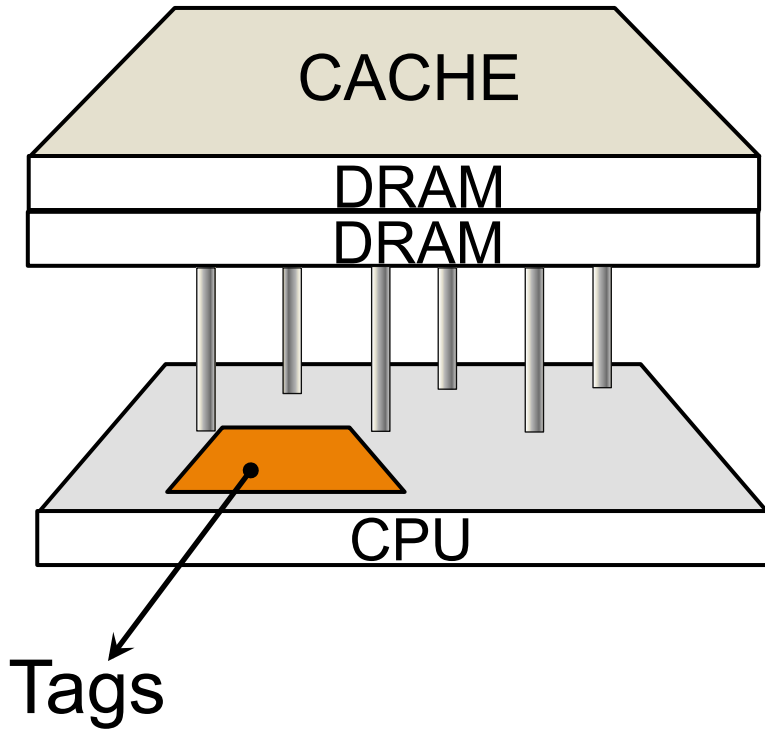
# Die-Stacked DRAM Caches



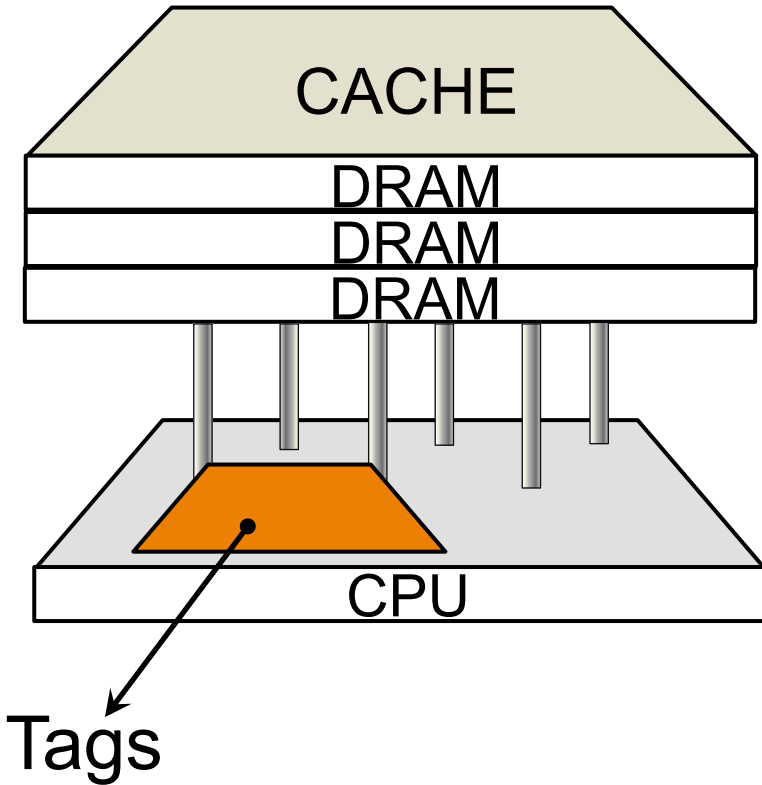
# Die-Stacked DRAM Caches



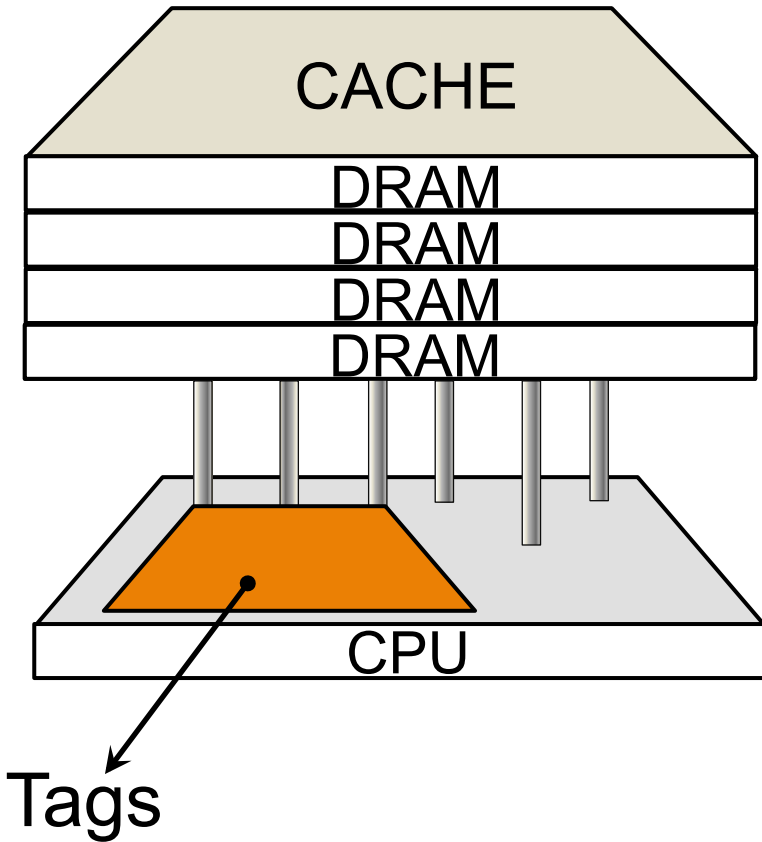
# Die-Stacked DRAM Caches



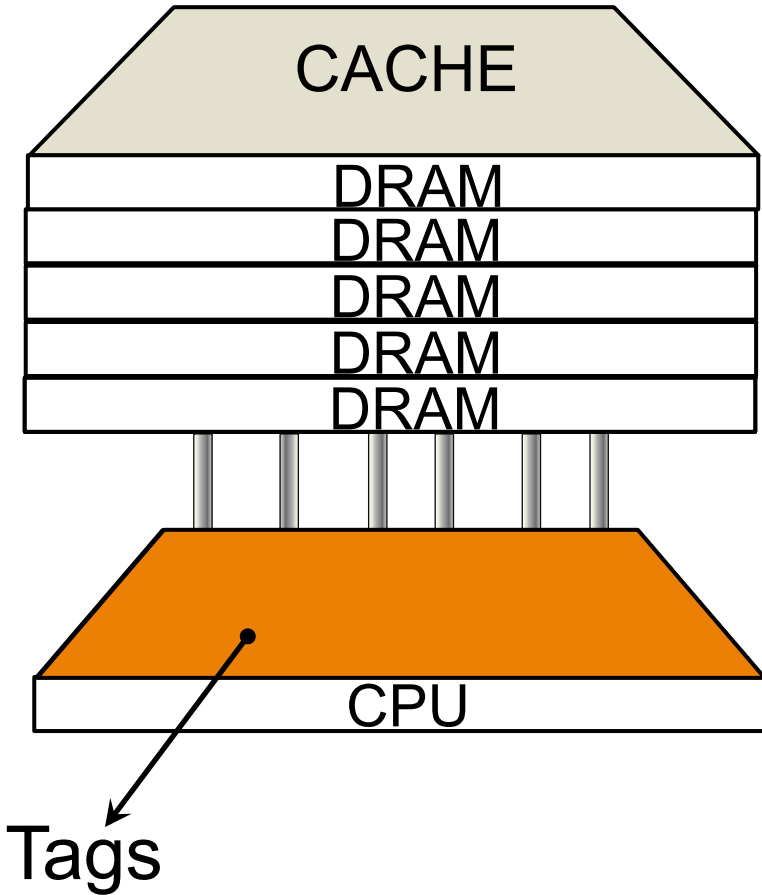
# Die-Stacked DRAM Caches



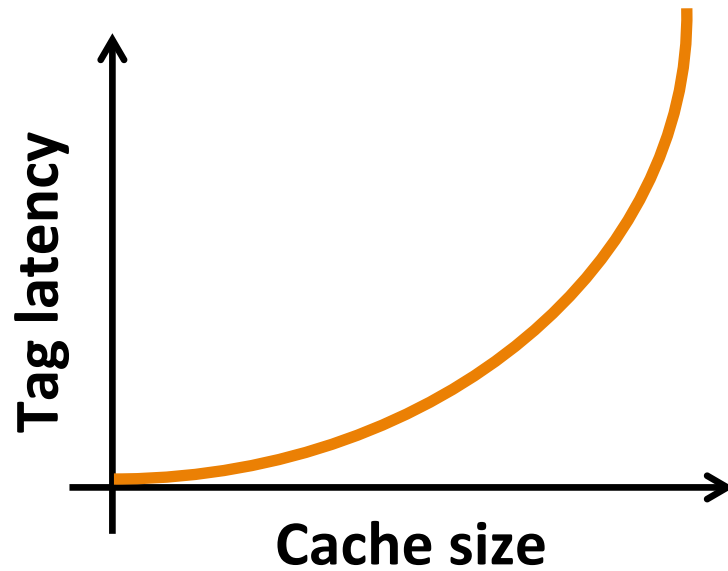
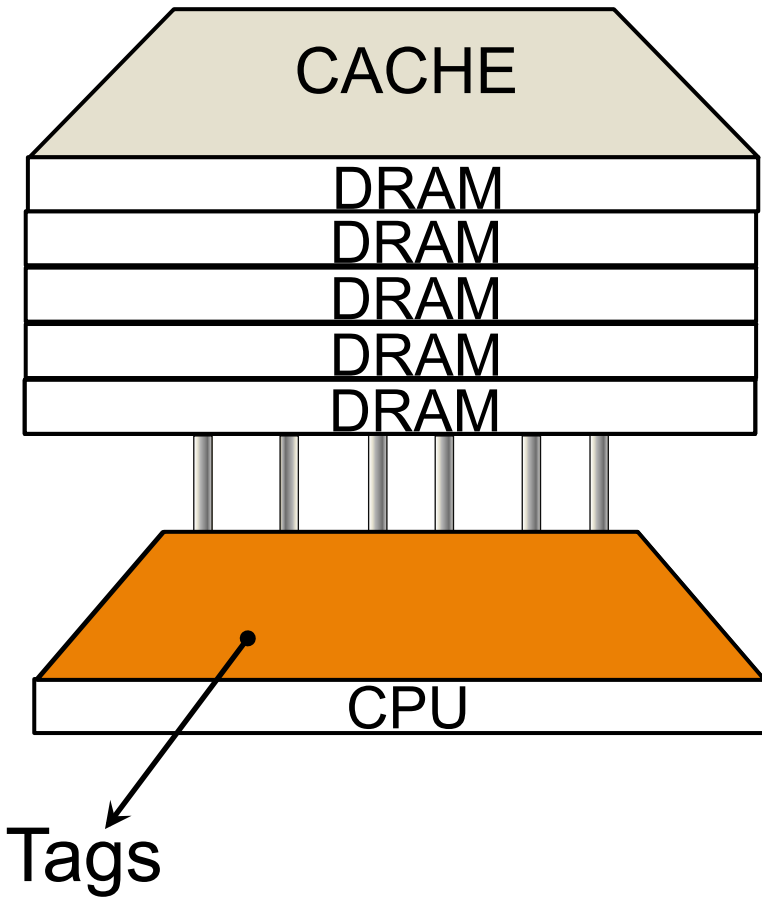
# Die-Stacked DRAM Caches



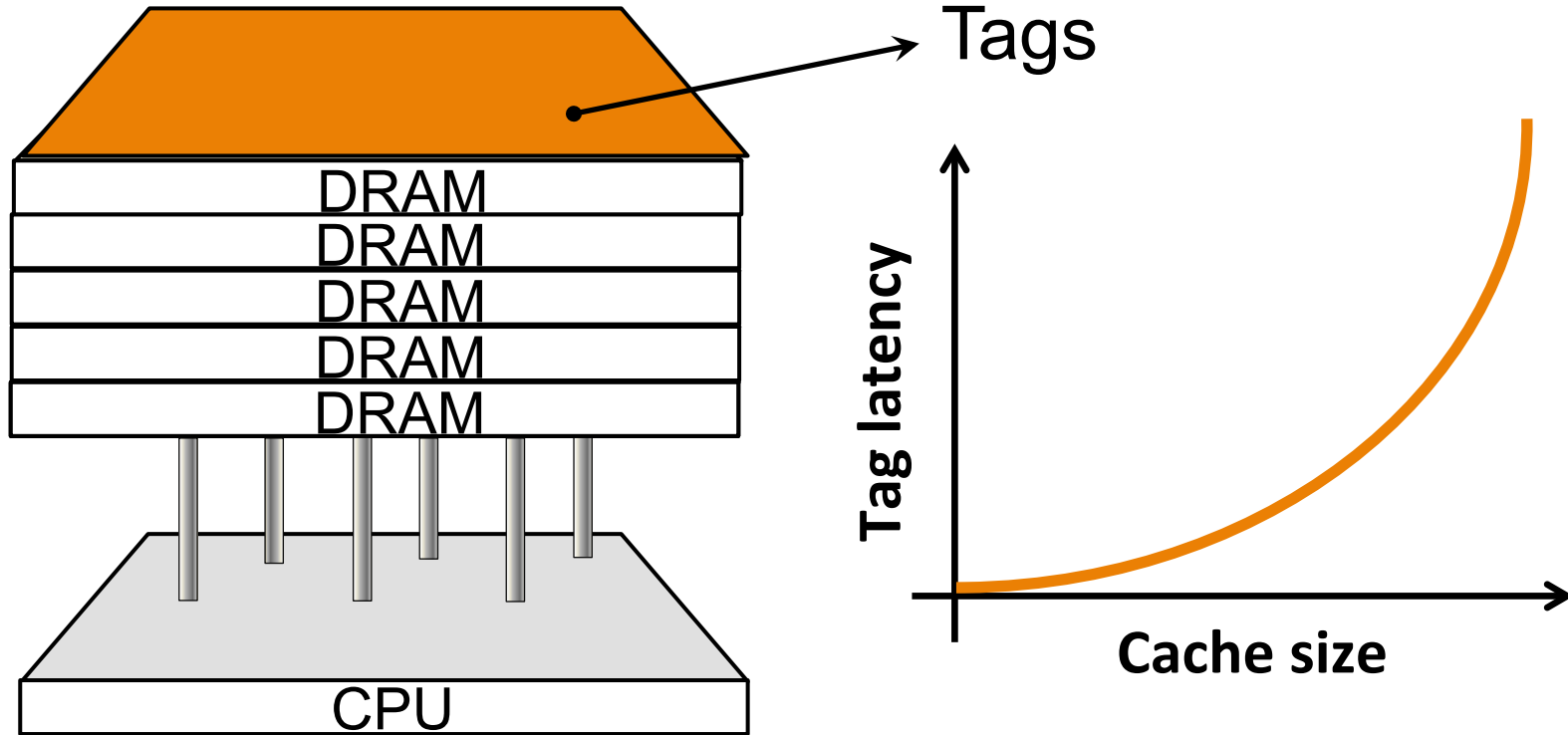
# Die-Stacked DRAM Caches



# Die-Stacked DRAM Caches

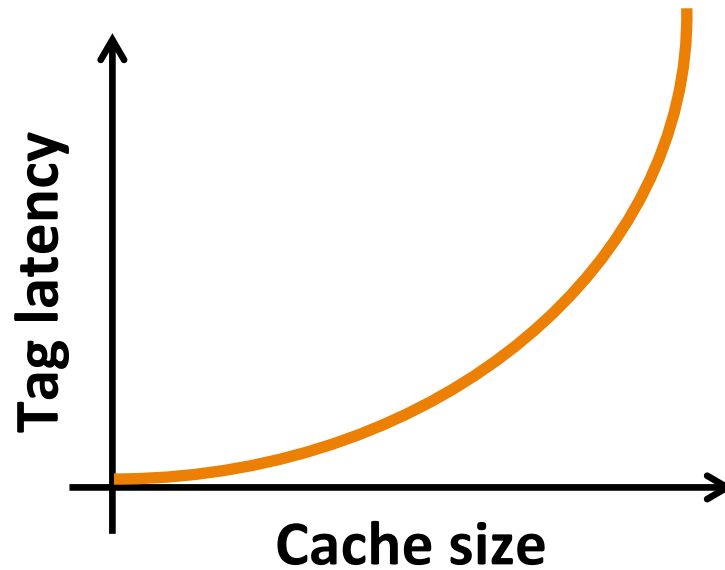
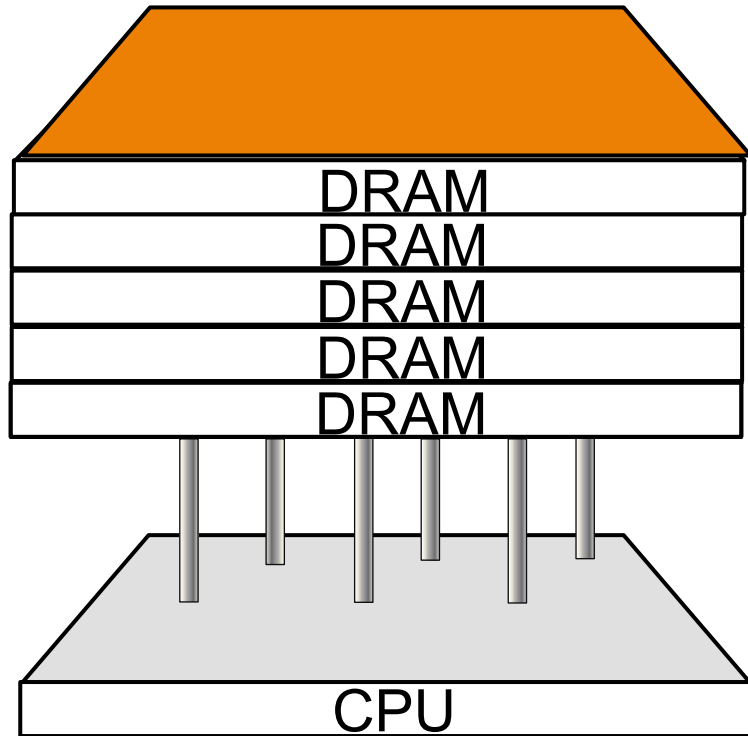


# Die-Stacked DRAM Caches



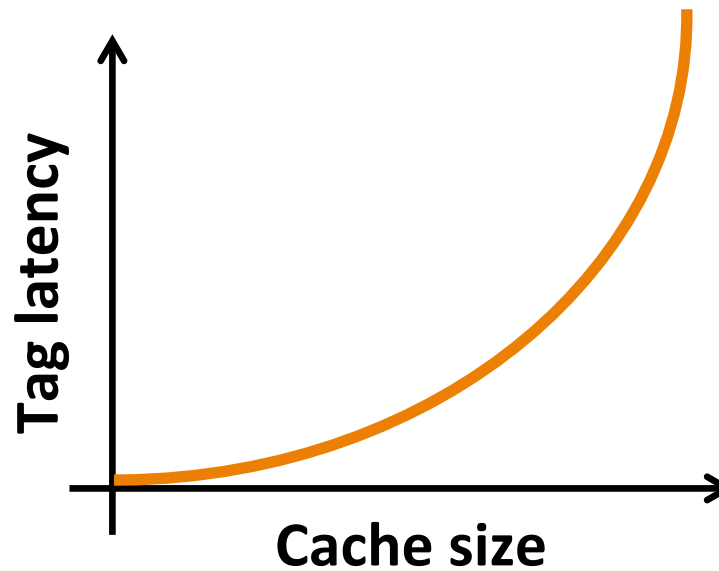
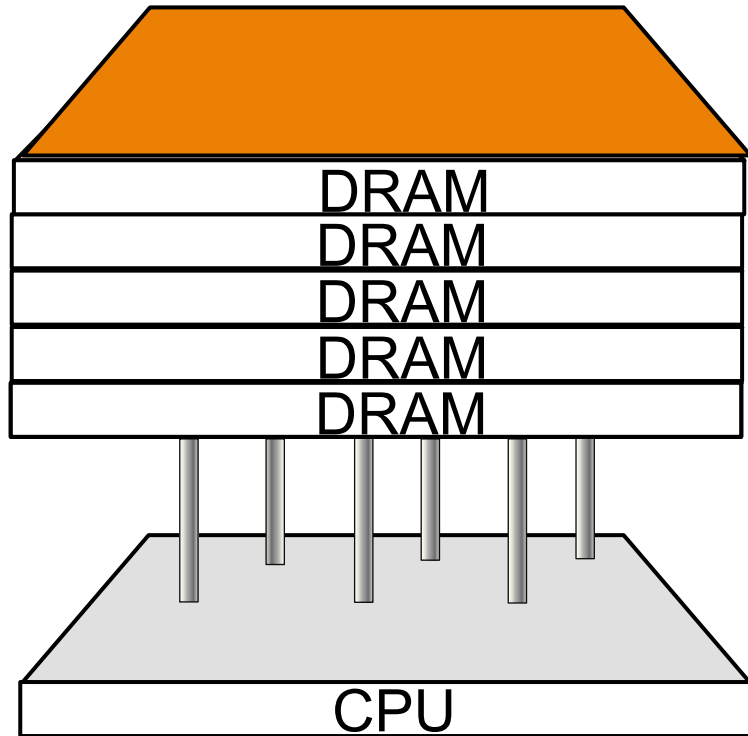


# Die-Stacked DRAM Caches



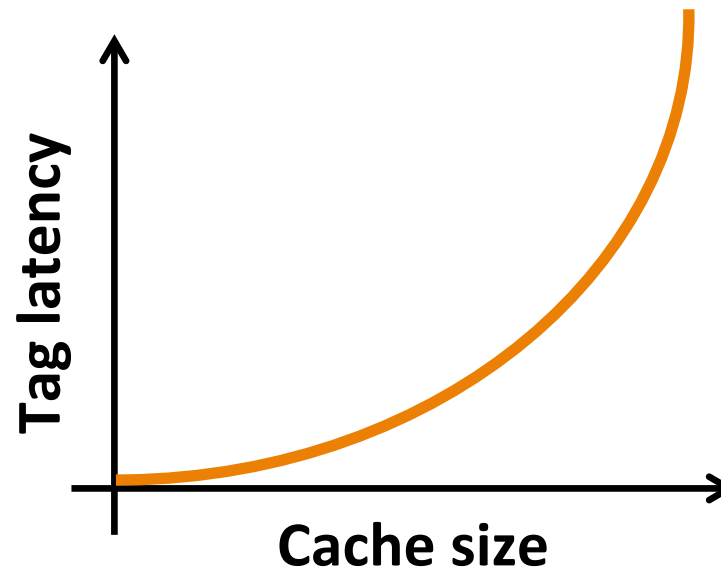
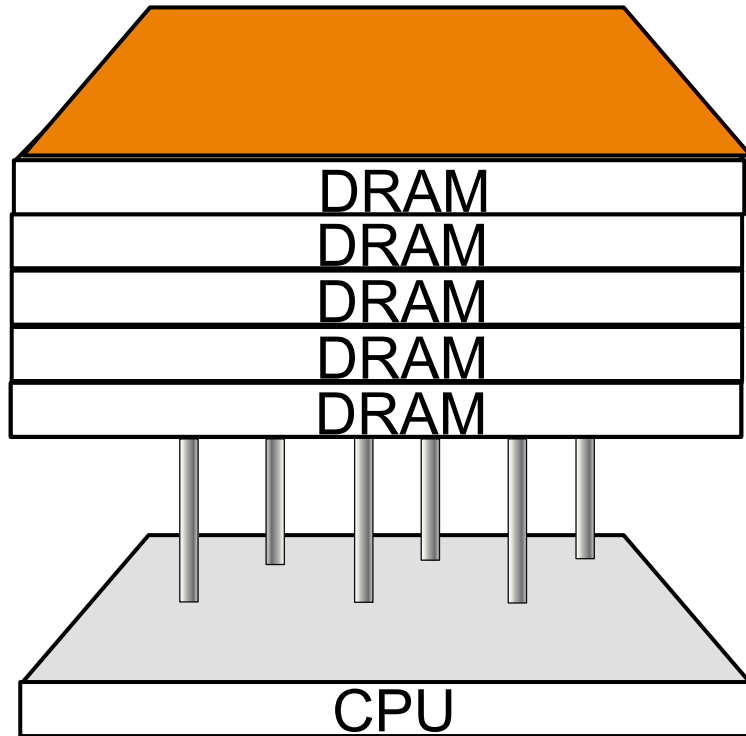
How to move tags to DRAM?

# Die-Stacked DRAM Caches



How to move tags to DRAM?  
 And improve performance?!

# Die-Stacked DRAM Caches

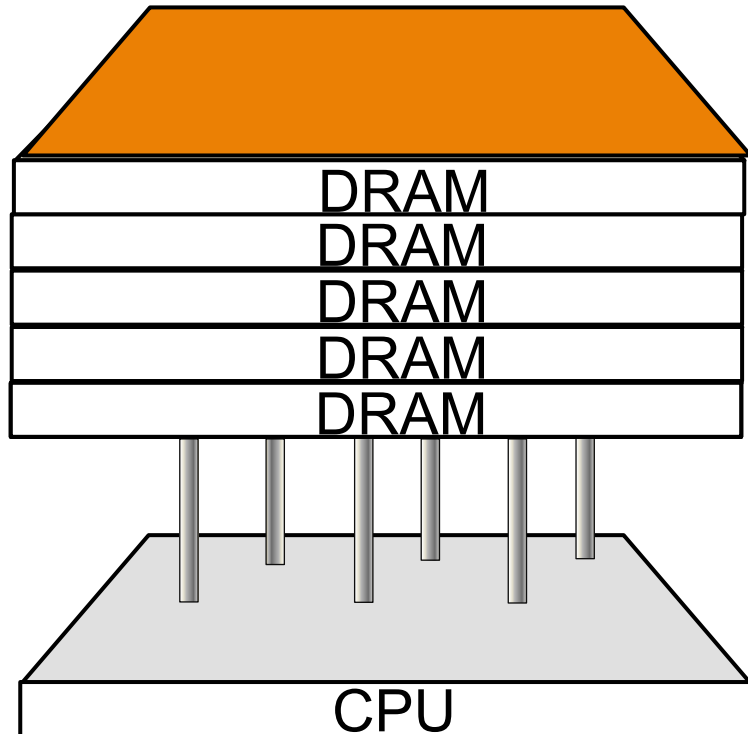


How to move tags to DRAM?

And improve performance?!

What is so special about servers?

# Die-Stacked DRAM Caches



Today at 2:15pm  
Session 1A  
Umney Theatre

How to move tags to DRAM?

And improve performance?!

What is so special about servers?

*next paper*

# Bi-Modal DRAM Cache: Improving Hit Rate, Hit Latency and Bandwidth

Nagendra Gulur\*

Collaborators: Mahesh Mehendale\*,  
R Manikantan§ and R Govindarajan+

\*Texas Instruments, Bangalore

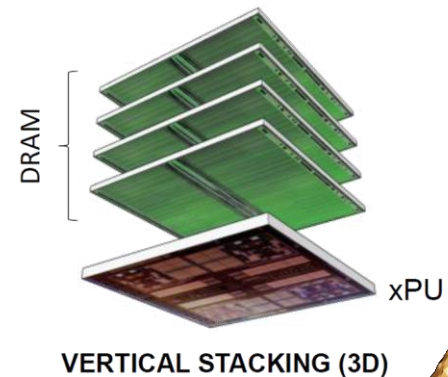
§ Intel Corporation, Bangalore

+Indian Institute of Science, Bangalore

## Problems with Stacked DRAM Caches

1. Large Tag Store
2. High Hit Latency
3. Wasted Off-Chip Bandwidth

*Picture courtesy Bryan Black*



# Large Metadata

SRAM

DRAM

- **Expensive**

- **Fast Access**

- **Mitigation:**

- Larger Blocks?



**High Wasted  
Bandwidth**

- **Slow Access**

- **Scales for very large  
cache sizes**

- **Mitigation:**

- Tags and Data in  
same rows?

- Give up Associativity?

## Our Proposal

Metadata : In DRAM

Block Size : Two Sizes (64B and 512B)

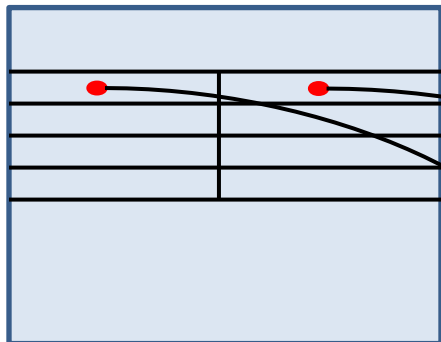
Tag Access : As good as SRAM



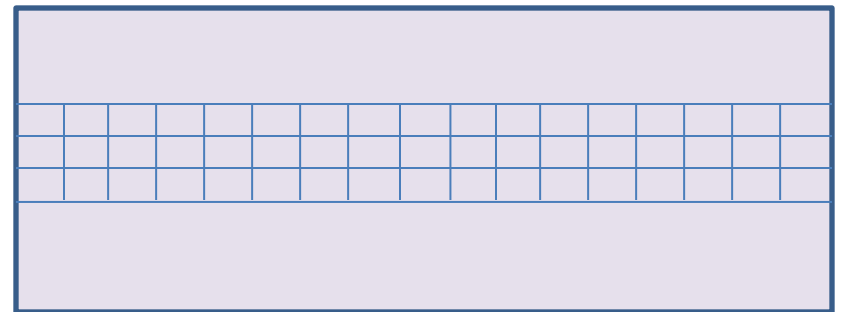
# Bi-Modal Cache Organization

*SRAM* | *DRAM*

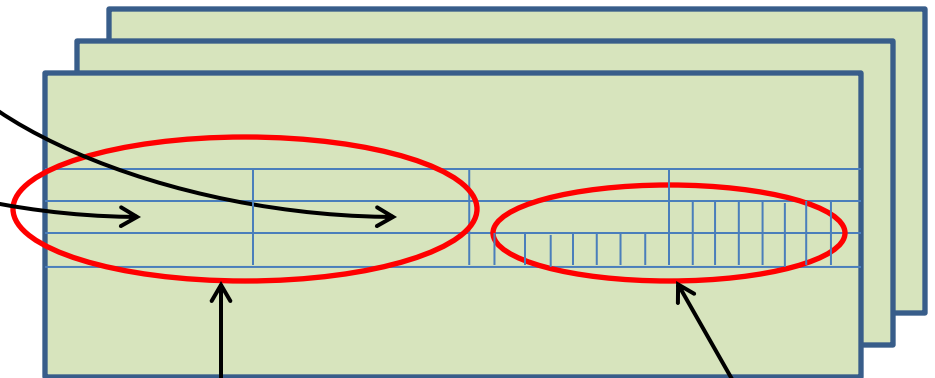
Way Locator



Tag Bank



Data Banks



Block Size Prediction



BIG

SMALL

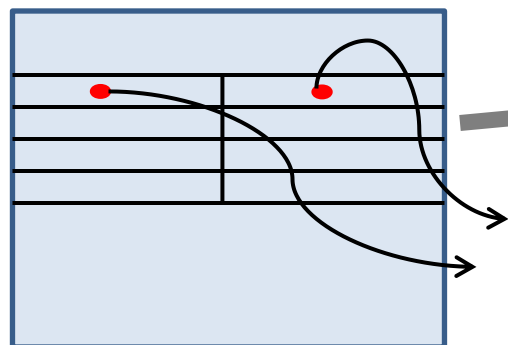
Some Big Blocks

Some Small Blocks



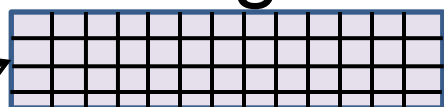


# Benefits and Results



Hit Rate	Hit Latency	Wasted Bandwidth
Green	Light Blue	Green
Light Blue	Green	Light Blue
Light Blue	Green	Light Blue

Tag



Data



Parallel Tag + Data

*Performance Improves by 10-14% in 4, 8 and 16-core workloads.*

*SRAM Overhead ~140KB*

*next paper*

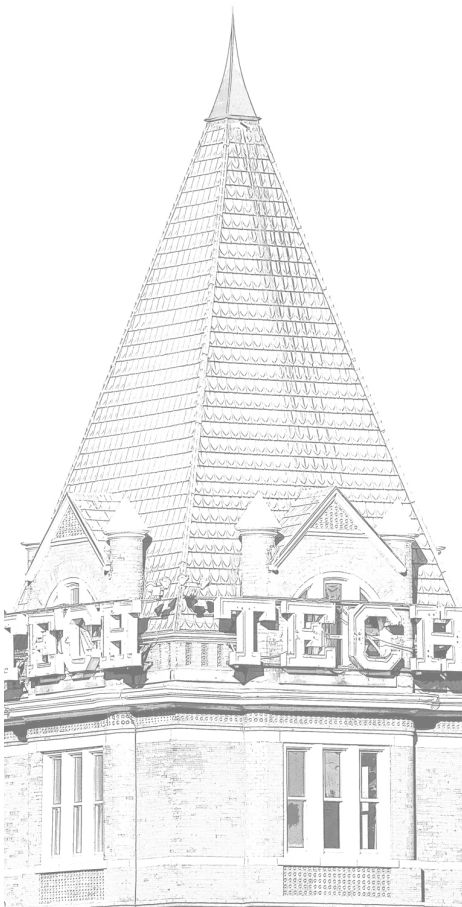
# Citadel: Efficiently Protecting Stacked Memory From Large Granularity Failures

Dec 15<sup>th</sup> 2014

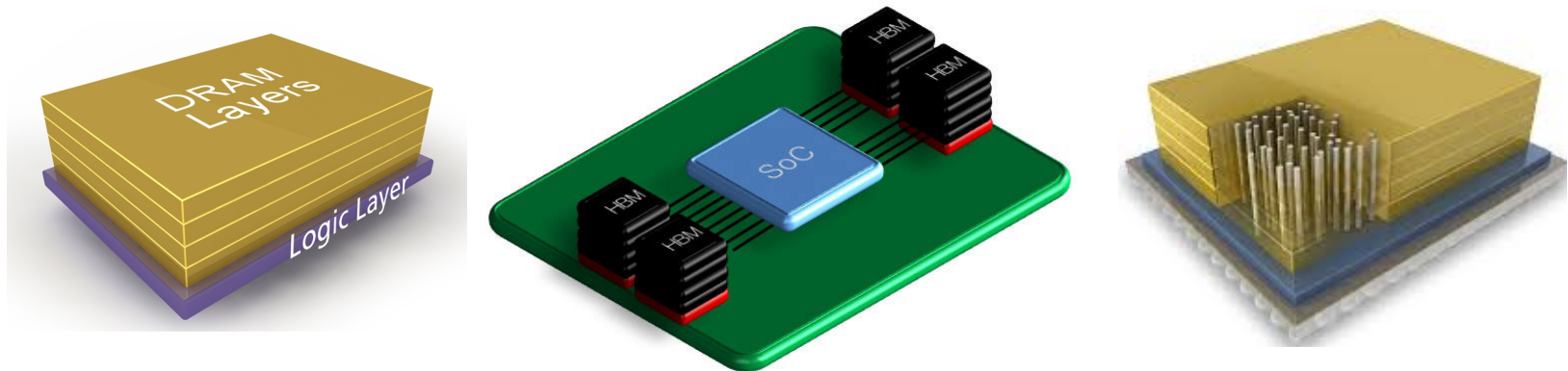
*Prashant Nair - Georgia Tech*

*David Roberts - AMD Research*

*Moinuddin Qureshi - Georgia Tech*



## 3D DRAM: Overcomes memory bandwidth wall

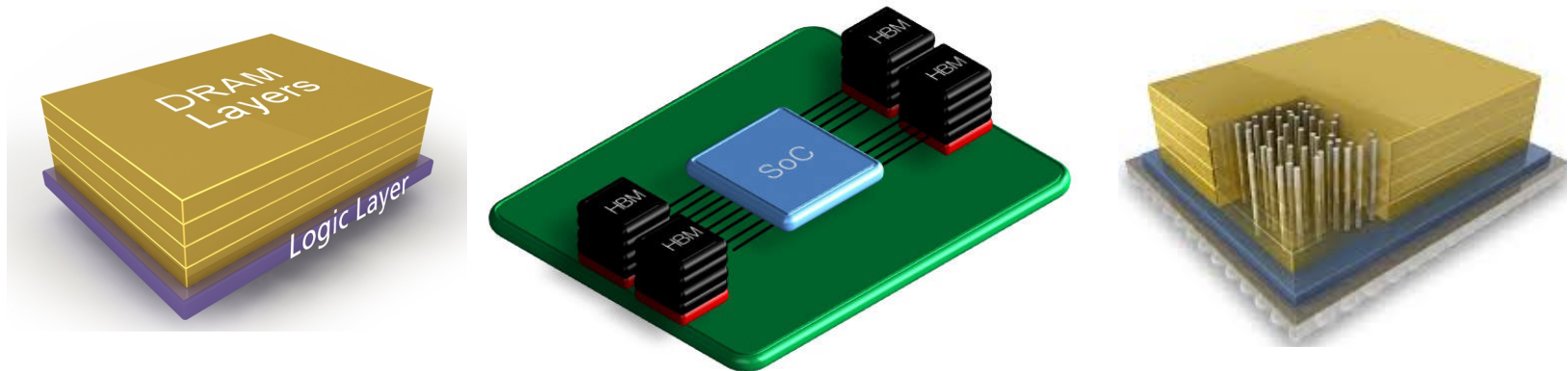


Susceptible to new failure modes: TSV faults

Causes large granularity failures (e.g. Faulty Bank)

Striping data across banks → high overheads

## 3D DRAM: Overcomes memory bandwidth wall



Susceptible to new failure modes: TSV faults

Causes large granularity failures (e.g. Faulty Bank)

Striping data across banks → high overheads

Goal: Tolerate TSV faults & Large faults at low cost

- Citadel protects against TSV and Large Faults, while retaining line in the same bank

# CITADEL FOR 3D DRAM

- Citadel protects against TSV and Large Faults, while retaining line in the same bank
- Citadel employs a three-pronged approach

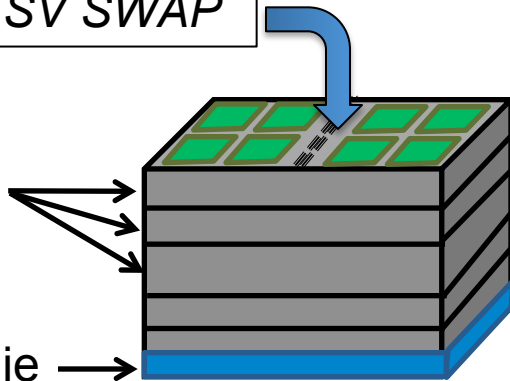
- Citadel protects against TSV and Large Faults, while retaining line in the same bank
- Citadel employs a three-pronged approach

1

*Dynamic TSV SWAP*

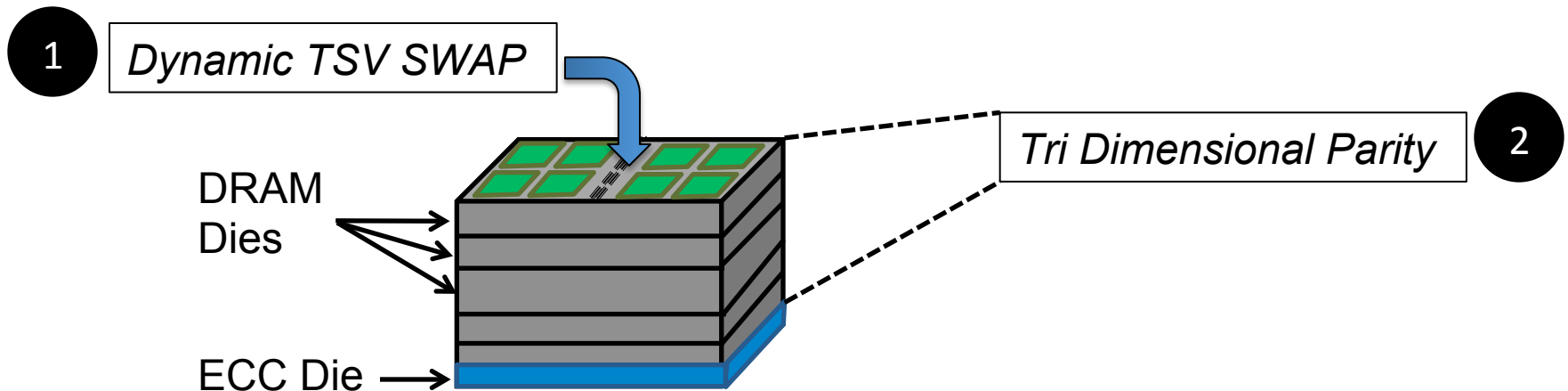
DRAM  
Dies

ECC Die



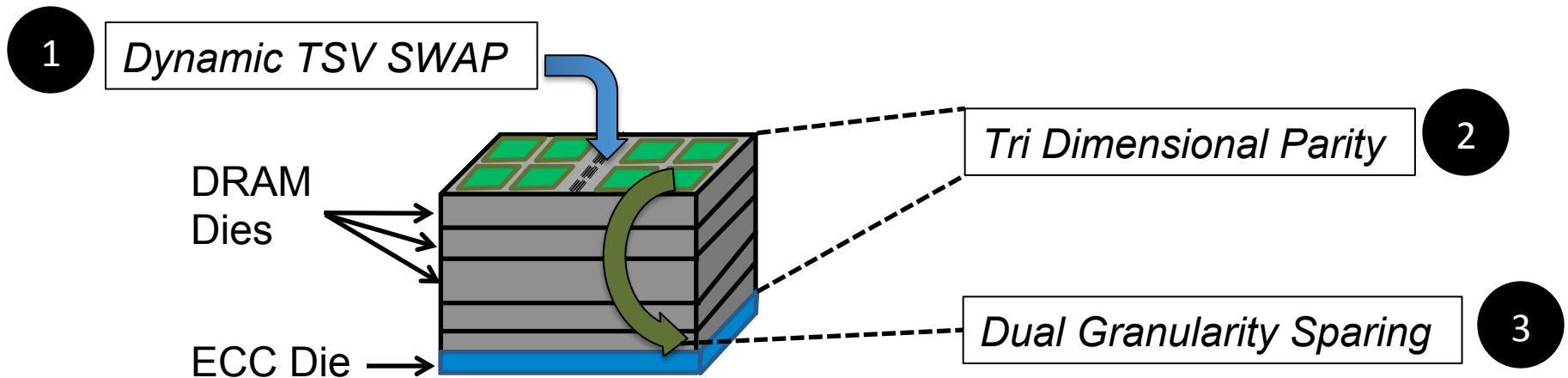


- Citadel protects against TSV and Large Faults, while retaining line in the same bank
- Citadel employs a three-pronged approach



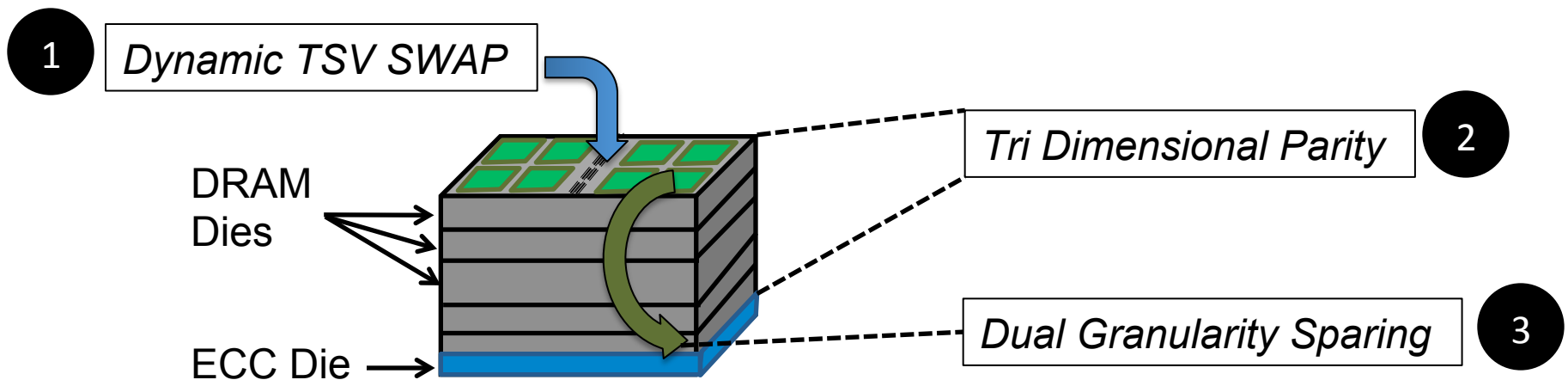
# CITADEL FOR 3D DRAM

- Citadel protects against TSV and Large Faults, while retaining line in the same bank
- Citadel employs a three-pronged approach



# CITADEL FOR 3D DRAM

- Citadel protects against TSV and Large Faults, while retaining line in the same bank
- Citadel employs a three-pronged approach



Citadel has negligible overheads and still provides **700x** higher resilience than the best ECC schemes

*next paper*



---

# Locality-Aware Mapping of Nested Parallel Patterns on GPUs

**HyoukJoong Lee**<sup>\*</sup>, Kevin Brown<sup>\*</sup>, Arvind Sujeeth<sup>\*</sup>,  
Tiark Rompf<sup>†‡</sup>, Kunle Olukotun<sup>\*</sup>

<sup>\*</sup>Pervasive Parallelism Laboratory, Stanford University  
<sup>†</sup>Purdue University, <sup>‡</sup>Oracle Labs



# Nested Parallel Patterns and Mapping Strategies

---



- High-level languages for GPUs
  - Provide higher productivity and portable performance
  - Using parallel patterns (e.g., map, reduce, groupby) is becoming popular



# Nested Parallel Patterns and Mapping Strategies

---



- High-level languages for GPUs
  - Provide higher productivity and portable performance
  - Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns are often nested, but difficult to map on GPUs
  - Many factors to consider together (e.g., coalescing, divergence, dynamic allocations)
  - Large space of possible mappings
  - Compilers often support only a fixed mapping strategy, but not always efficient



# Nested Parallel Patterns and Mapping Strategies



- High-level languages for GPUs
  - Provide higher productivity and portable performance
  - Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns are often nested, but difficult to map on GPUs
  - Many factors to consider together (e.g., coalescing, divergence, dynamic allocations)
  - Large space of possible mappings
  - Compilers often support only a fixed mapping strategy, but not always efficient

```
// Pagerank algorithm
Nodes map { n =>
  nbrsWeights = n.nbrs map { w =>
    getPrevPageRank(w) / w.degree
  }
  sumWeights = nbrsWeights reduce { (a,b) => a + b }
  ((1 - damp) / numNodes + damp * sumWeights
}
```

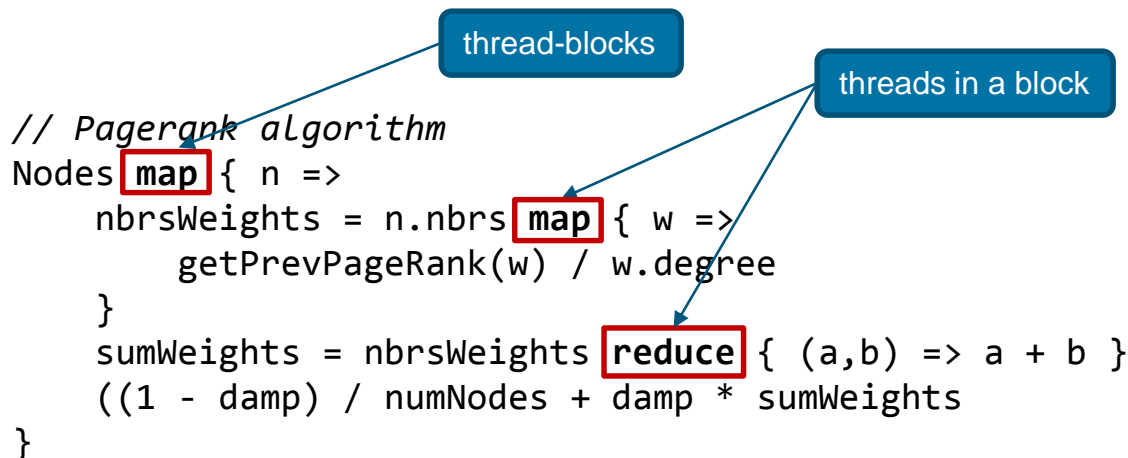




# Nested Parallel Patterns and Mapping Strategies



- High-level languages for GPUs
  - Provide higher productivity and portable performance
  - Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns are often nested, but difficult to map on GPUs
  - Many factors to consider together (e.g., coalescing, divergence, dynamic allocations)
  - Large space of possible mappings
  - Compilers often support only a fixed mapping strategy, but not always efficient

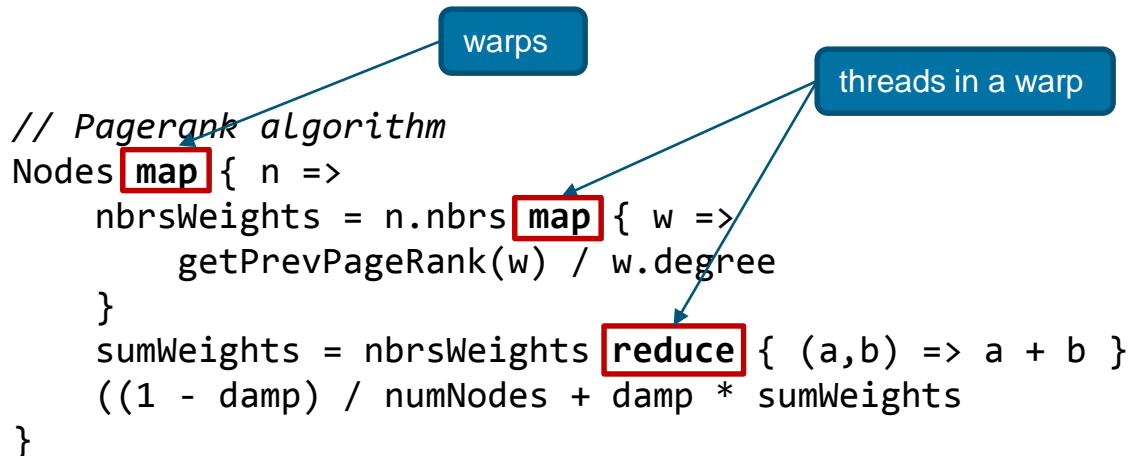




# Nested Parallel Patterns and Mapping Strategies



- High-level languages for GPUs
  - Provide higher productivity and portable performance
  - Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns are often nested, but difficult to map on GPUs
  - Many factors to consider together (e.g., coalescing, divergence, dynamic allocations)
  - Large space of possible mappings
  - Compilers often support only a fixed mapping strategy, but not always efficient

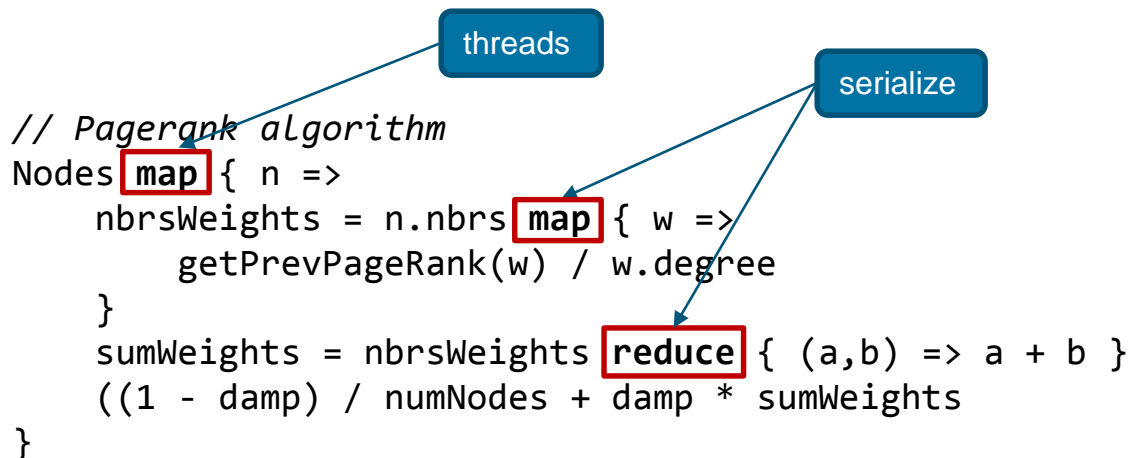




# Nested Parallel Patterns and Mapping Strategies



- High-level languages for GPUs
  - Provide higher productivity and portable performance
  - Using parallel patterns (e.g., map, reduce, groupby) is becoming popular
- Parallel patterns are often nested, but difficult to map on GPUs
  - Many factors to consider together (e.g., coalescing, divergence, dynamic allocations)
  - Large space of possible mappings
  - Compilers often support only a fixed mapping strategy, but not always efficient





# Compiler Framework for Multi-Dimensional Mapping

---



- Define Mapping Parameters

Logical Dimension:  $x, y, z, ..$

Block Size:  $N$

Degree of Parallelism (DOP):  $\text{Span}(n), \text{Split}(k)$

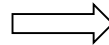


# Compiler Framework for Multi-Dimensional Mapping



## ■ Define Mapping Parameters

Logical Dimension:  $x, y, z, ..$   
Block Size:  $N$   
Degree of Parallelism (DOP):  $\text{Span}(n), \text{Split}(k)$



Pattern (I)	$\text{Dim}(y), \text{Size}(16), \text{Span}(1)$
Pattern (J)	$\text{Dim}(x), \text{Size}(32), \text{Span}(\text{all})$

Equivalent Parameters for Warp-Based Mapping

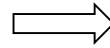


# Compiler Framework for Multi-Dimensional Mapping



## ■ Define Mapping Parameters

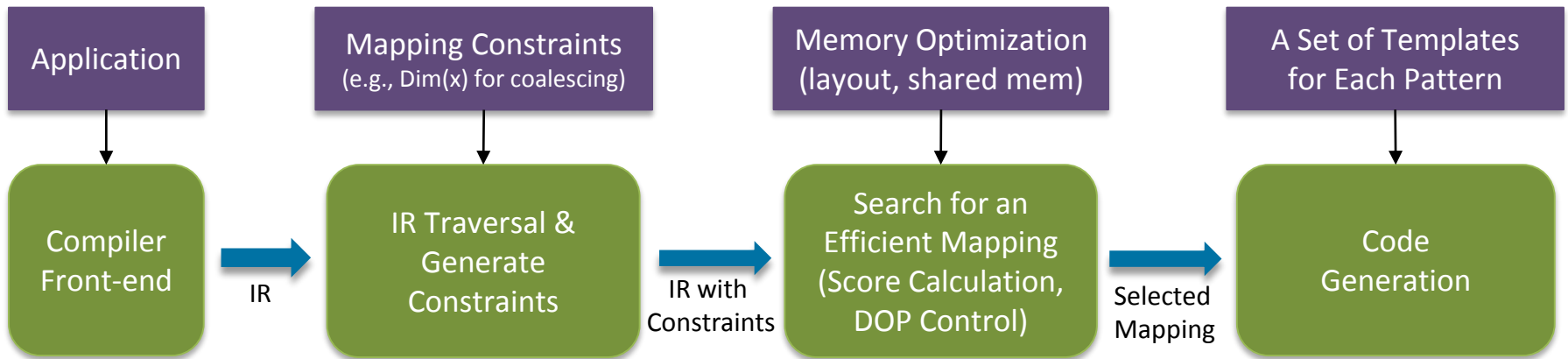
Logical Dimension:  $x, y, z, ..$   
Block Size:  $N$   
Degree of Parallelism (DOP):  $\text{Span}(n), \text{Split}(k)$



Pattern (I)     $\text{Dim}(y), \text{Size}(16), \text{Span}(1)$   
Pattern (J)     $\text{Dim}(x), \text{Size}(32), \text{Span}(\text{all})$

Equivalent Parameters for Warp-Based Mapping

## ■ Compiler Overview

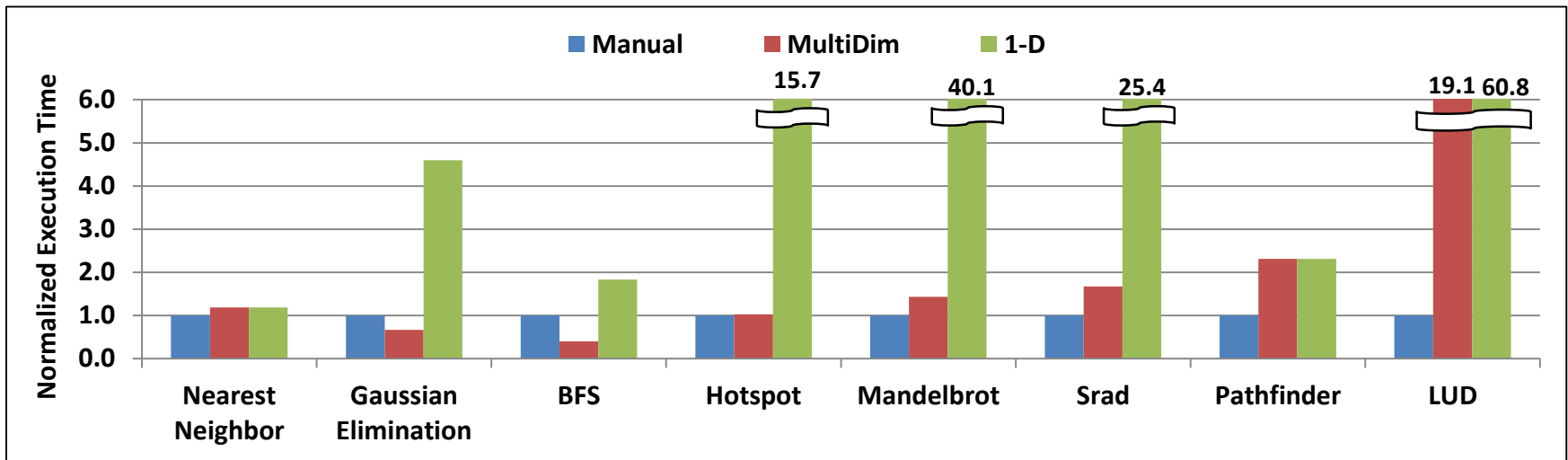




# Performance Results



## Rodinia benchmark suite



- 28.6x speedup over 1D mappings
- 9.6x speedup over existing 2D mappings
- 24% slower than manually optimized CUDA code (7 out of 8)
- Today 1:25 PM (Session IB, Room: Main Auditorium)

*next paper*



# Accelerating Irregular Algorithms on GPGPUs Using Fine-Grain Hardware Worklists

---

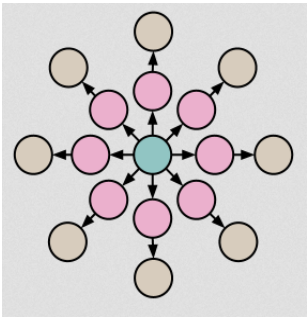
Ji Kim and Christopher Batten

Cornell University

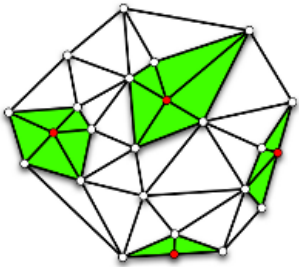
IEEE/ACM International Symposium  
on Microarchitecture 2014  
(MICRO-47)

# Amorphous Data Parallelism

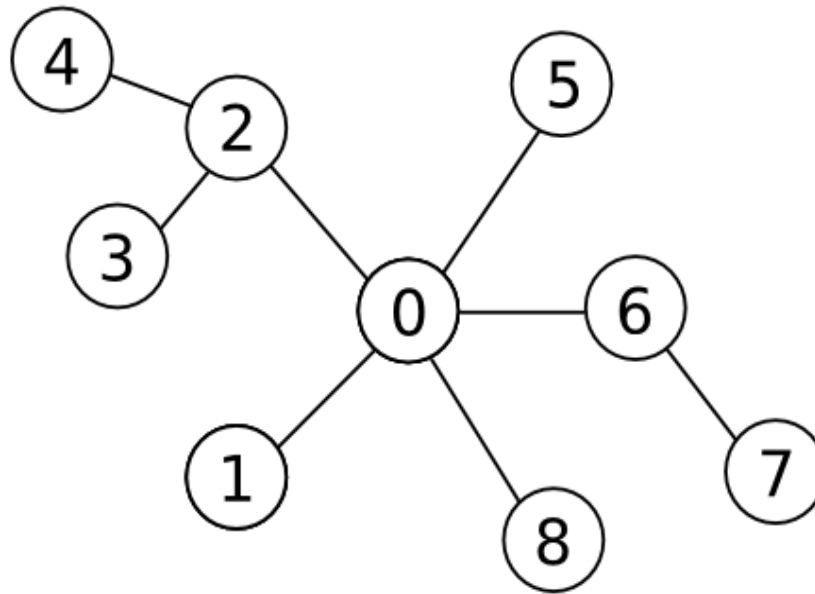
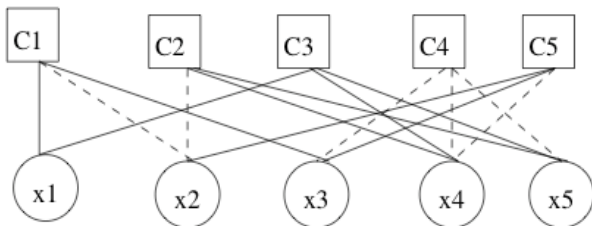
Breadth-First Search



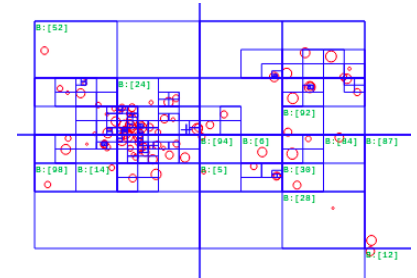
Delaunay Mesh Refinement



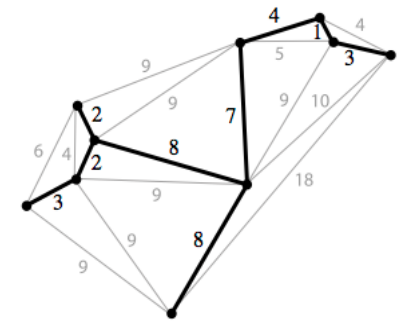
Survey Propagation



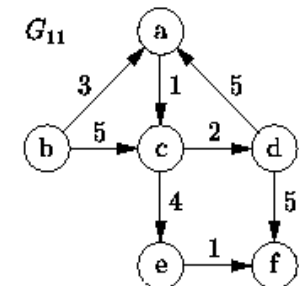
Barnes-Hut N-Body



Minimum Spanning Tree

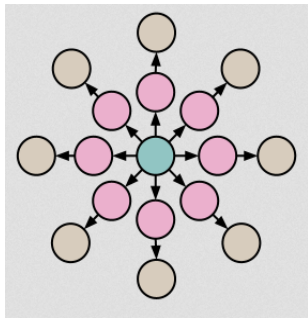


Single-Source Shortest-Path

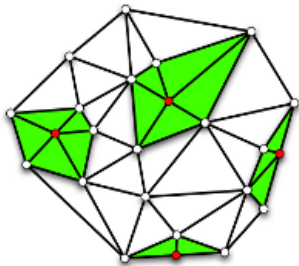


# Amorphous Data Parallelism

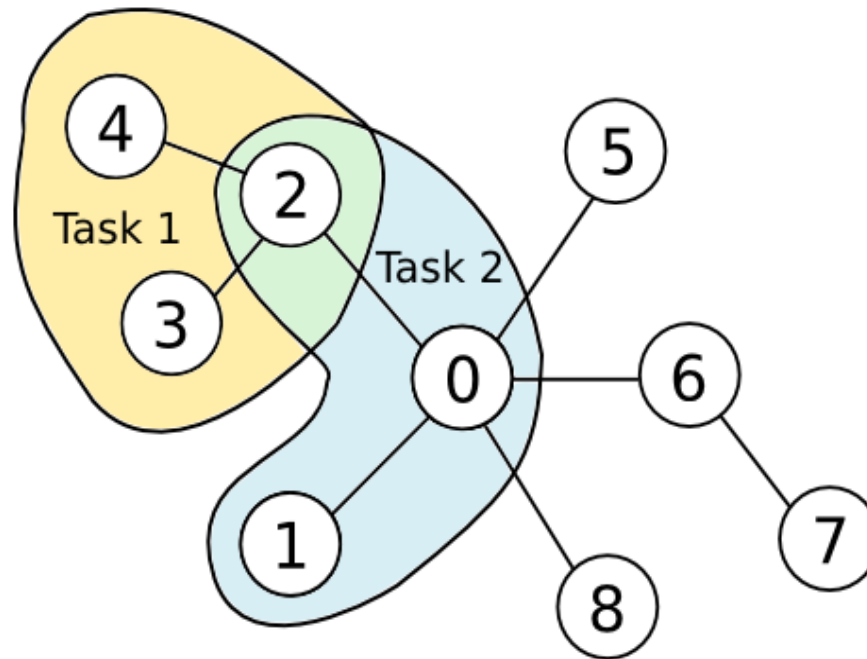
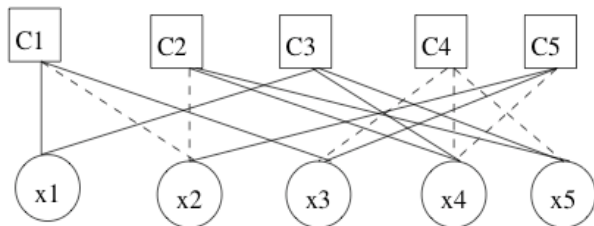
Breadth-First Search



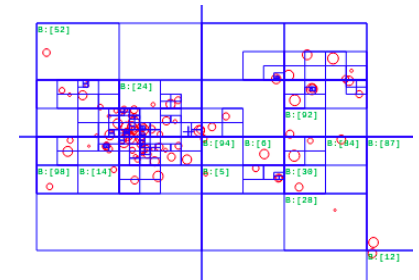
Delaunay Mesh Refinement



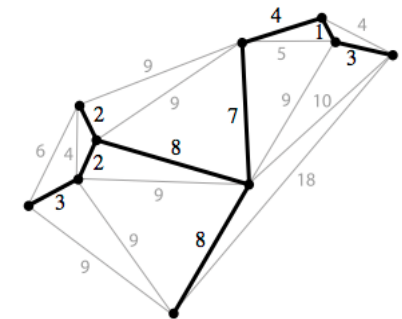
Survey Propagation



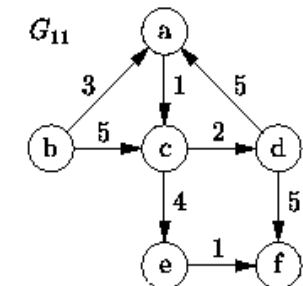
Barnes-Hut N-Body



Minimum Spanning Tree



Single-Source Shortest-Path

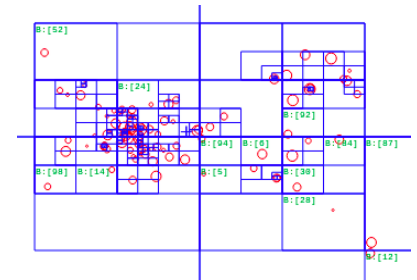




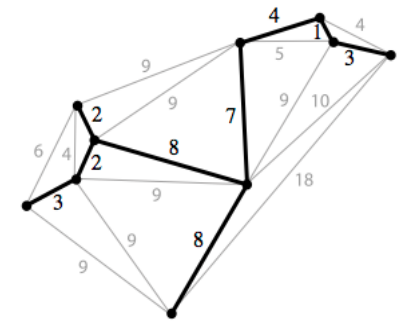


# Amorphous Data Parallelism

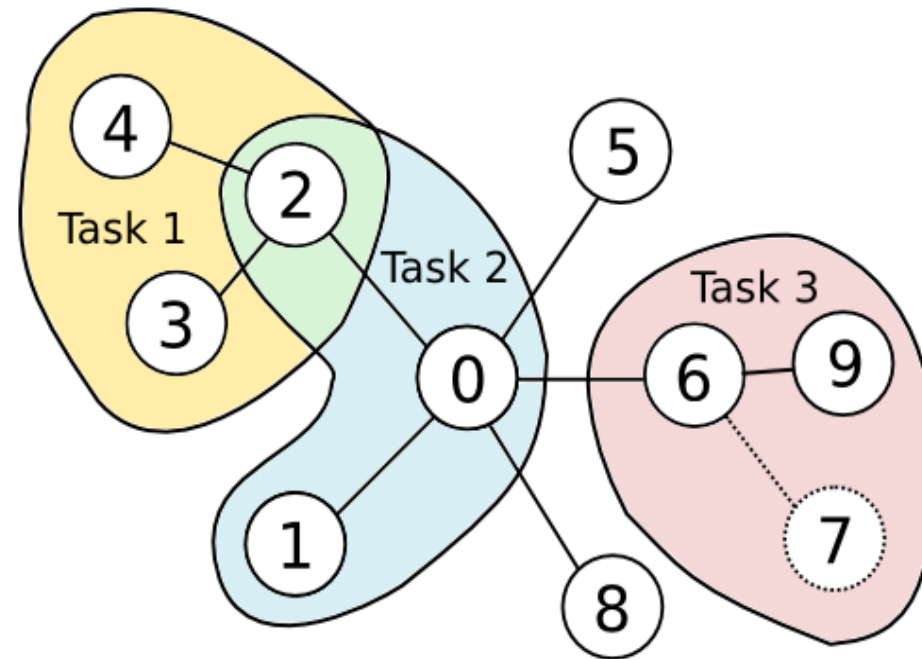
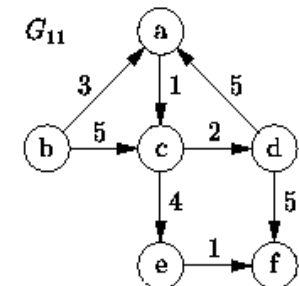
Barnes-Hut N-Body



Minimum Spanning Tree

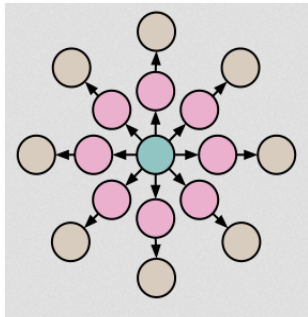


Single-Source Shortest-Path

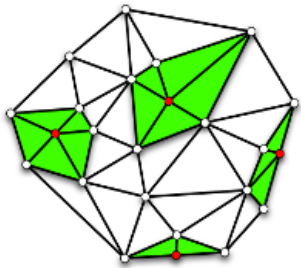


**Experiments on NVIDIA  
Tesla C2075 GPU using  
LonestarGPU  
benchmark suite**

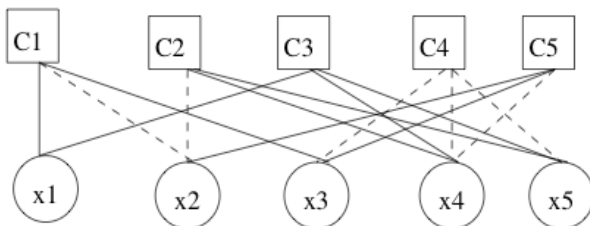
Breadth-First Search



Delaunay Mesh  
Refinement



Survey Propagation



# Existing SW Approaches

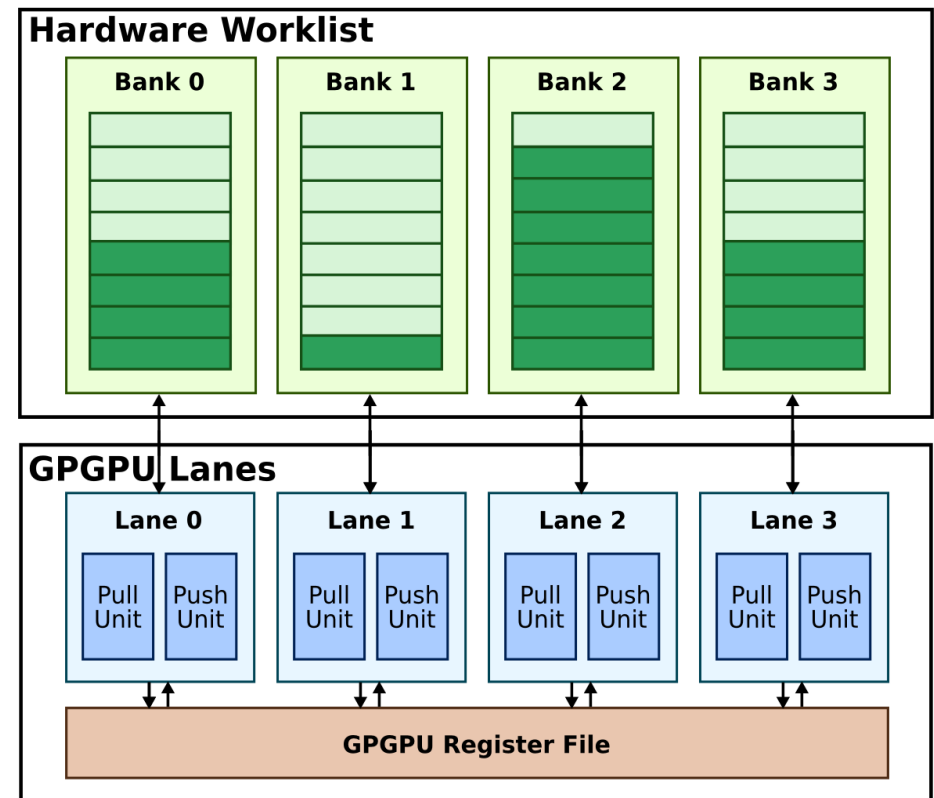
# Fine-Grain Hardware Worklist

- Memory contention
- Suboptimal load balancing
- SW overhead

# Existing SW Approaches

- Memory contention
  - **HWWL distributed banks**
- Suboptimal load balancing
- SW overhead
  - **HWWL distributed banks**

# Fine-Grain Hardware Worklist

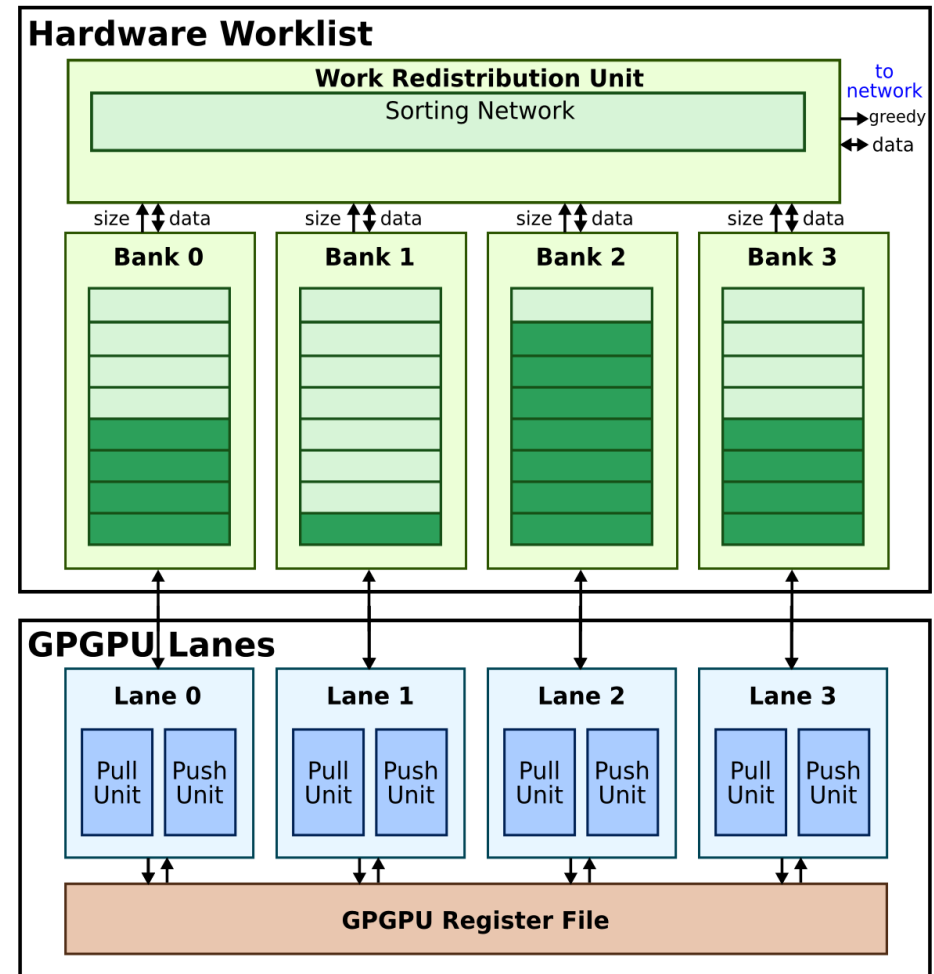




# Existing SW Approaches

- Memory contention
  - **HWWL distributed banks**
- Suboptimal load balancing
  - **HWWL work redistribution**
- SW overhead
  - **HWWL distributed banks**

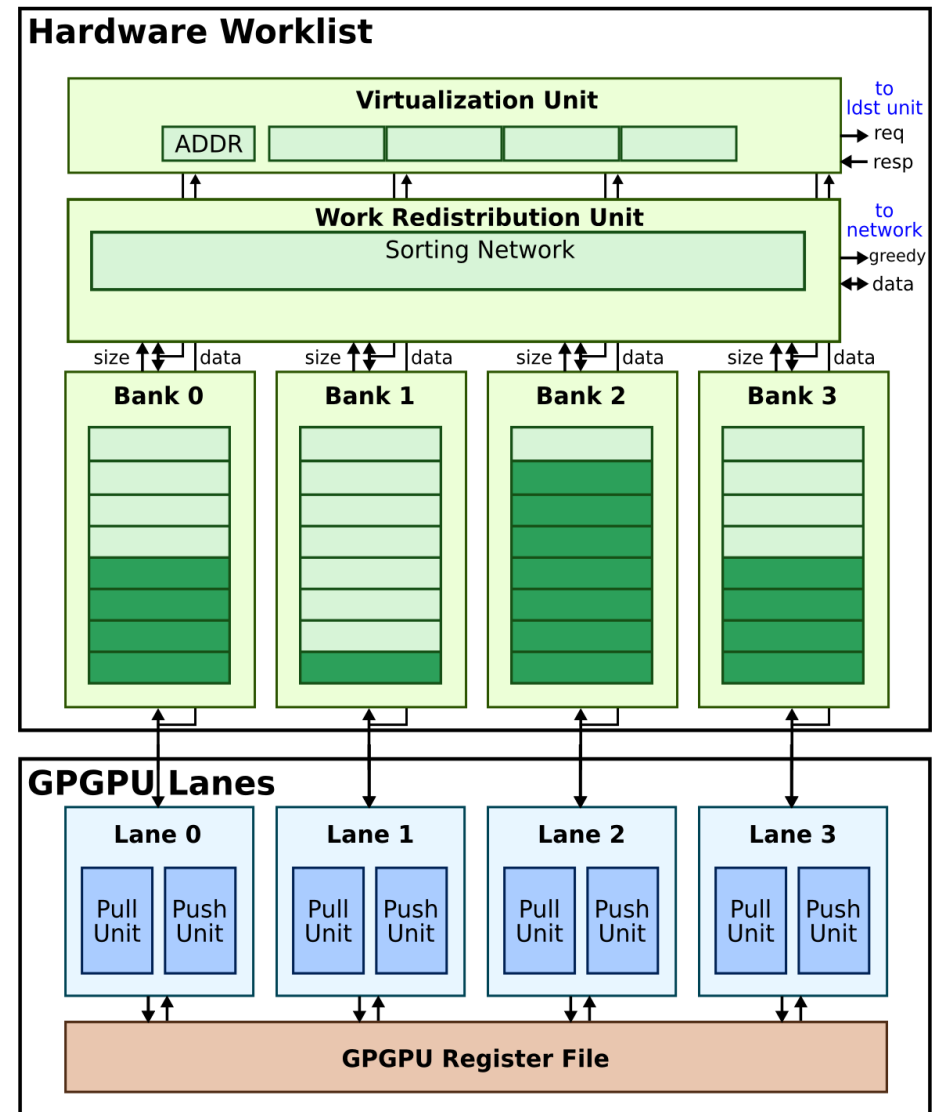
# Fine-Grain Hardware Worklist



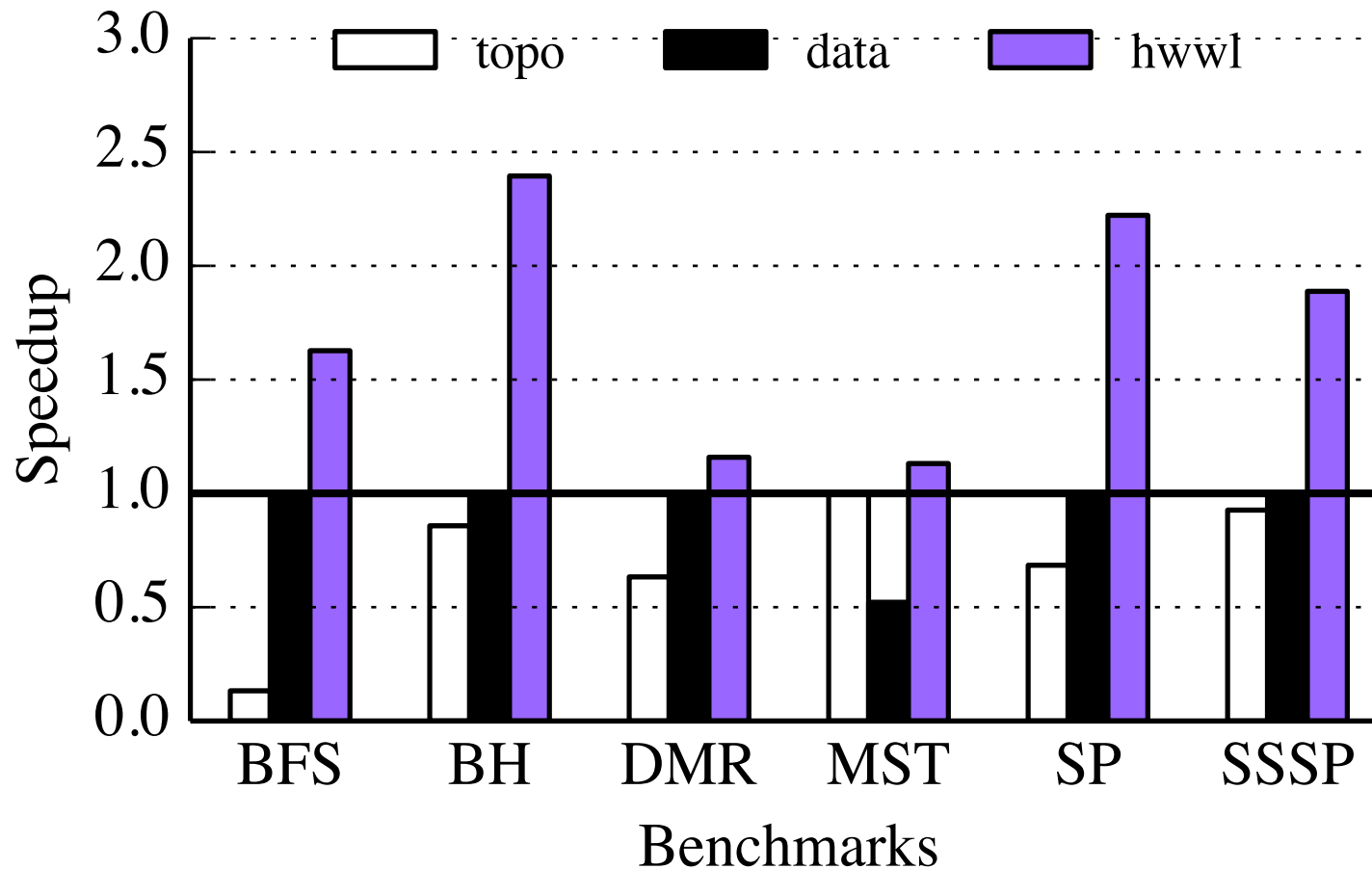
# Existing SW Approaches

- Memory contention
  - **HWWL distributed banks**
- Suboptimal load balancing
  - **HWWL work redistribution**
- SW overhead
  - **HWWL distributed banks**
- Seamless work spilling to and refilling from memory

# Fine-Grain Hardware Worklist



# Performance Results



**1.2—2.4X speedup over highly optimized SW implementation of challenging, irregular applications**

*next paper*

# PORPLE: An Extensible Optimizer for Portable Data Placement on GPU

Guoyang Chen

Xipeng Shen

North Carolina State University

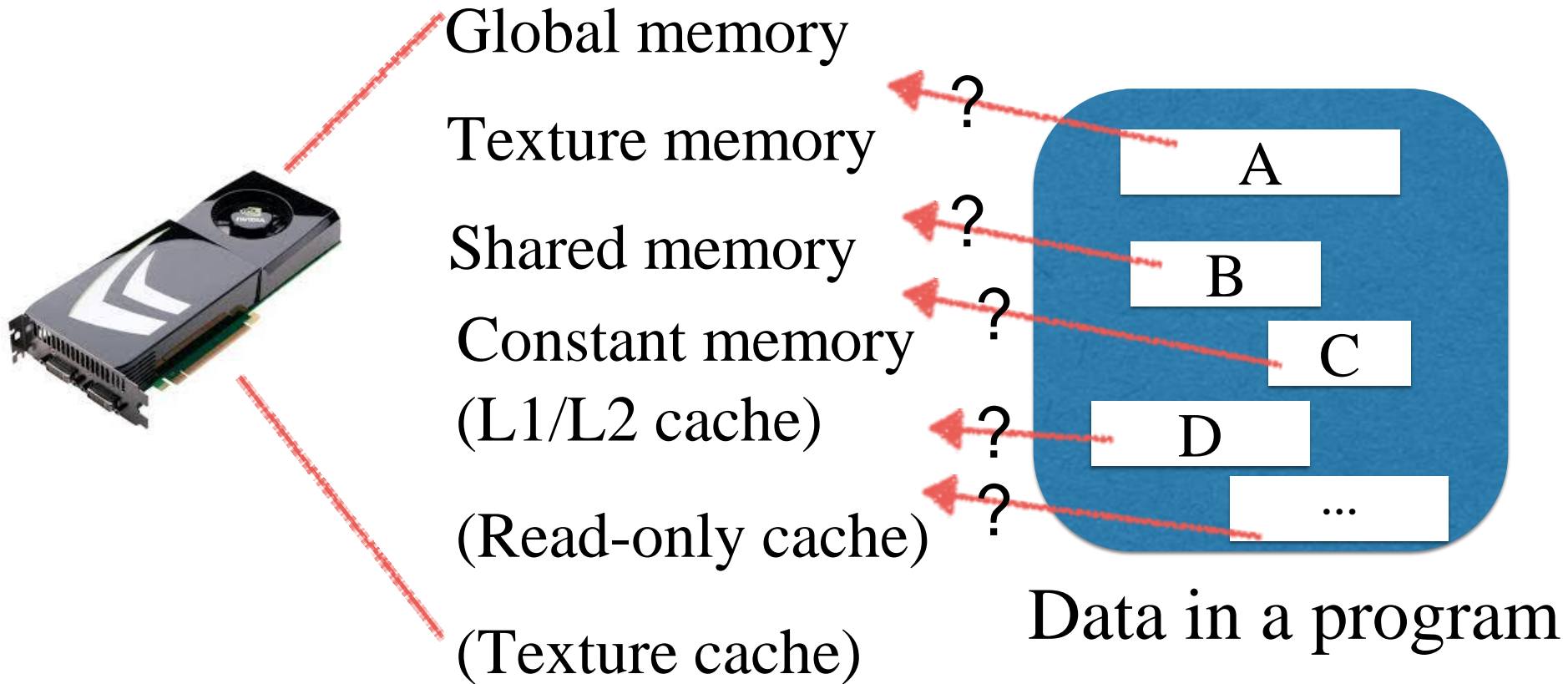
Bo Wu

Colorado School of Mines

Dong Li

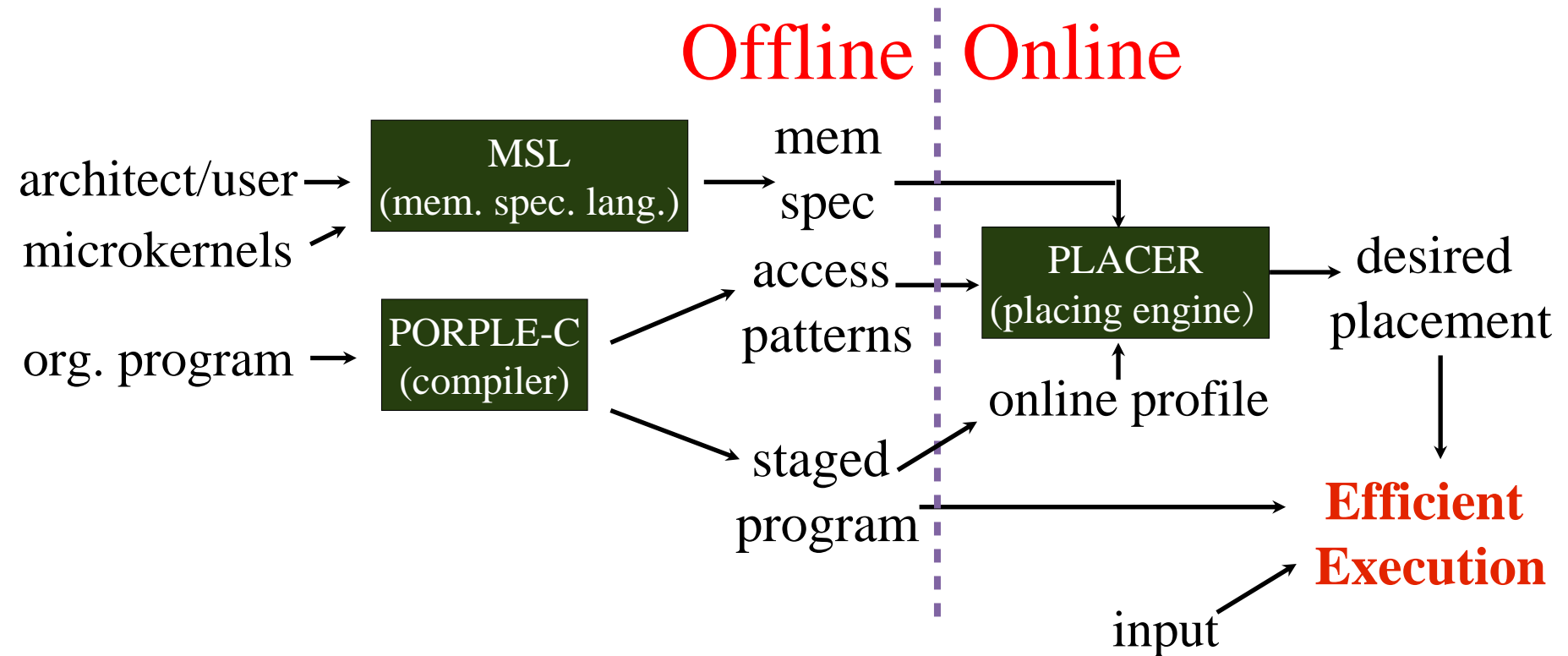
Oak Ridge National Lab

# Data Placement Problem on GPU



**3X performance difference**

# PORPLE: Portable Data Placement Engine



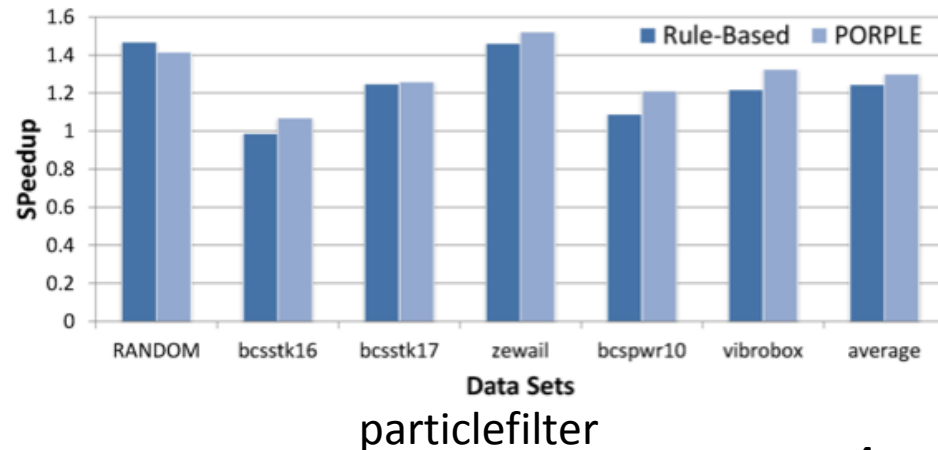
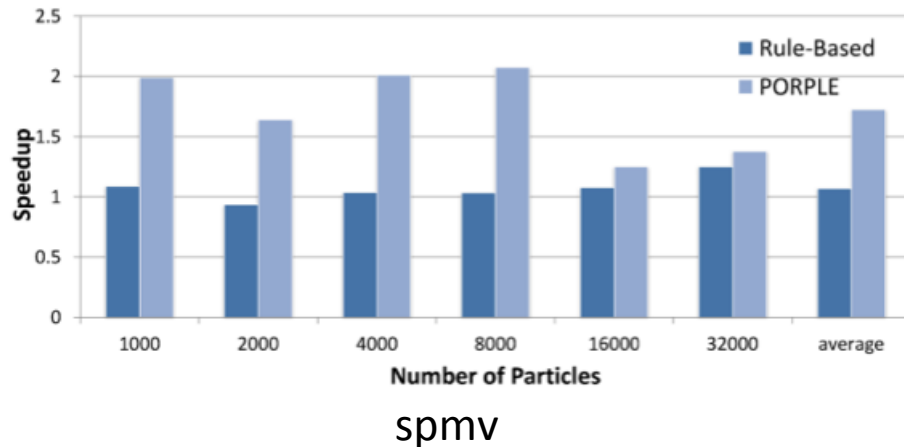
Goal: To determine the best data placement strategy cross different architectures and inputs during run-time.

# Portability and Input-adaptive

- Cross architectures

	spmv					particlefilter					
	A0	A1	A2	A3	A4	B0	B1	B2	B3	B4	B5
Rule-Based	T	T	T	T	G	G	S&G	G	G	G	G
PORPLE-C1060	C	T	T	T	G	C	S&G	G	G	G	G
PORPLE-M2075	C	T	G	T	G	C	S&G	G	G	G	G
PORPLE-K20c	C	R	T	R	G	C	S&R	G	T	G	G

- Cross inputs





*next paper*



# Exploring the Design Space of SPMD Divergence Management on Data-Parallel Architectures

**Yunsup Lee, UC Berkeley**

Vinod Grover, NVIDIA

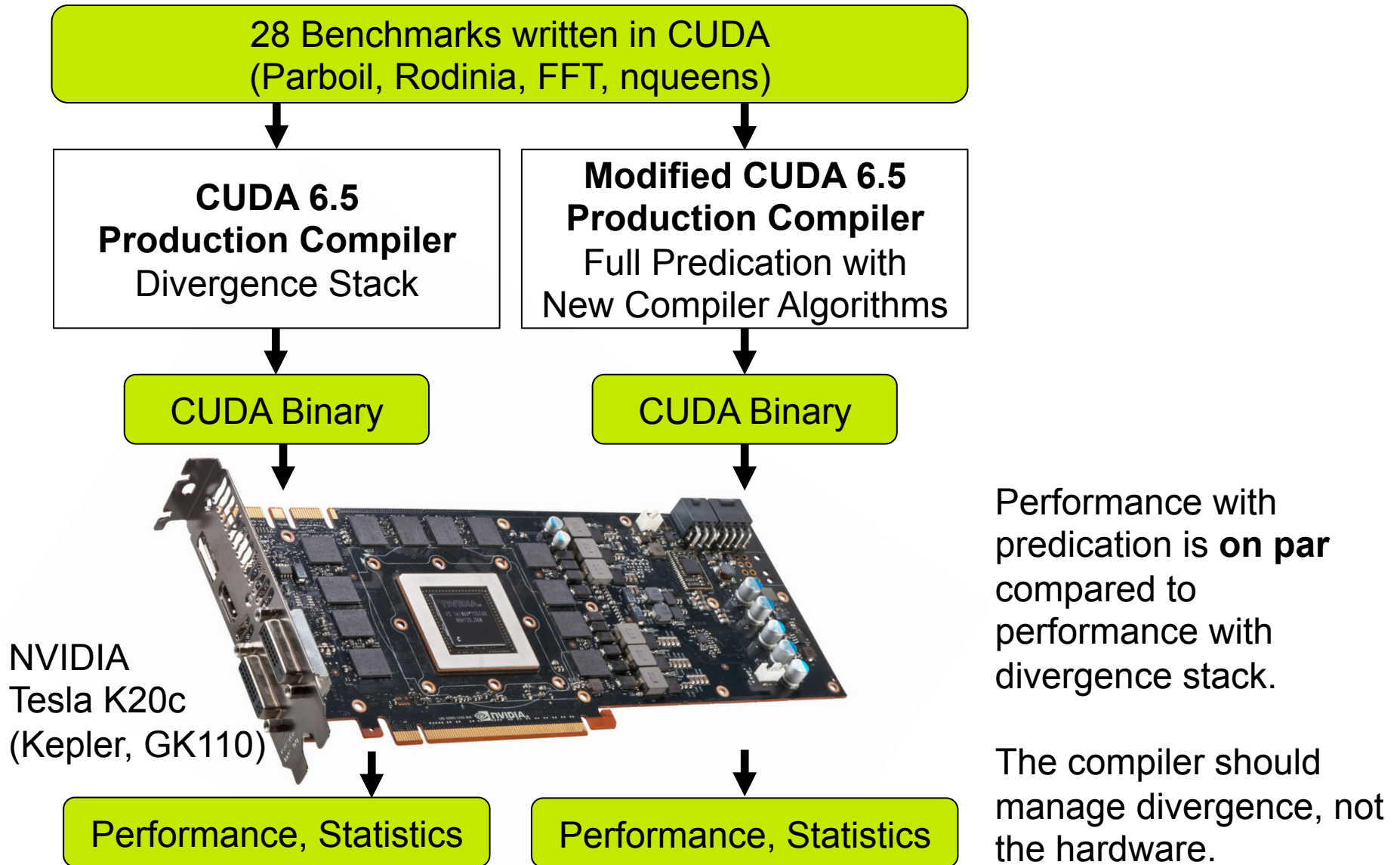
Ronny Krashinsky, NVIDIA

Mark Stephenson, NVIDIA

Stephen W. Keckler, NVIDIA

Krste Asanovic, UC Berkeley

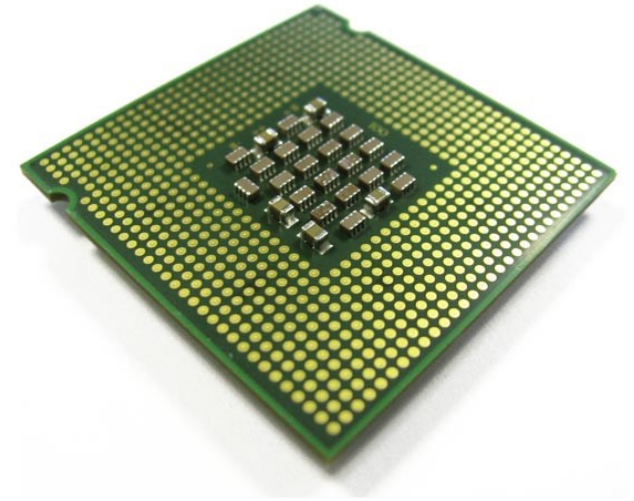
# Executive Summary



*GPUs do not need  
a divergence stack!*

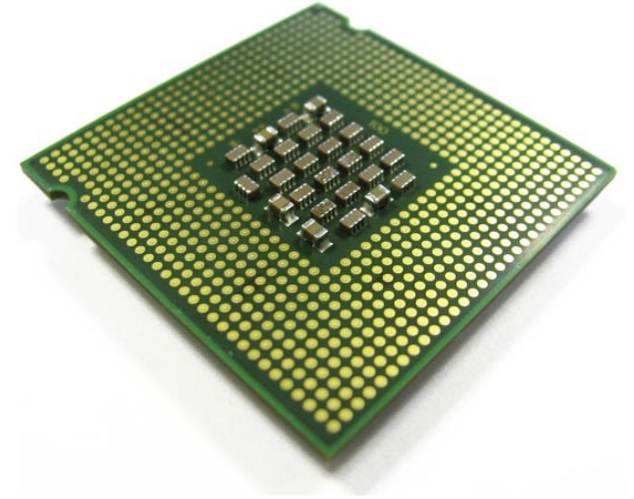
*next paper*

# Managing GPU Concurrency in Heterogeneous Architectures



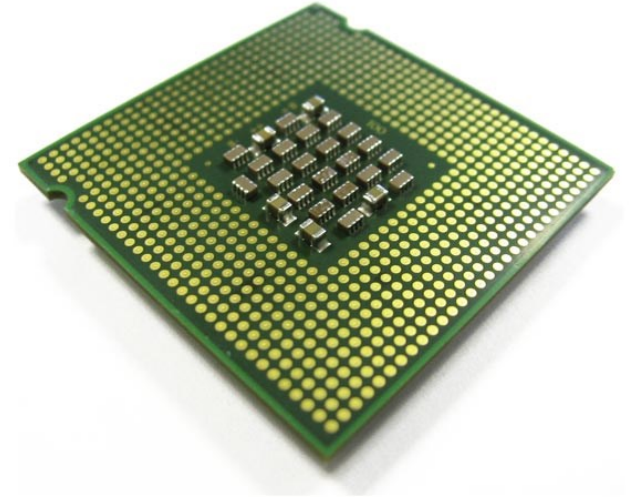
**LLC**  
**Memory** **Network**  
**Shared Resources**

# Managing GPU Concurrency in Heterogeneous Architectures



LLC  
Memory Network  
**Shared Resources**

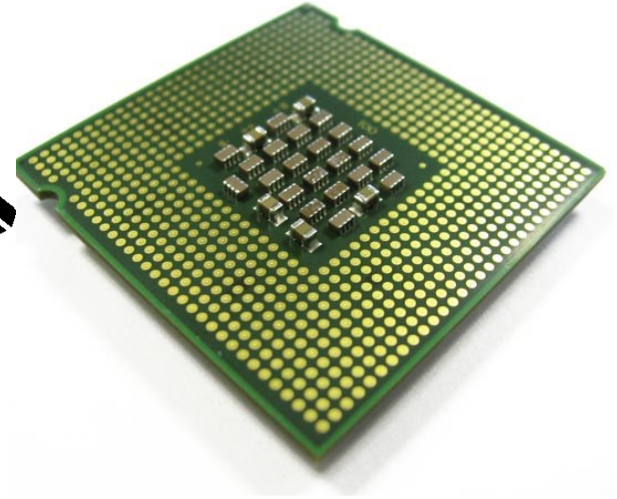
# Managing GPU Concurrency in Heterogeneous Architectures



LLC  
Memory Network  
Shared Resources



# Managing GPU Concurrency in Heterogeneous Architectures



LLC  
Memory Network  
**Shared Resources**

# Our Proposal

---

Warp Scheduler  
Controls GPU Thread-Level Parallelism

---

# Our Proposal

---

## Warp Scheduler Controls GPU Thread-Level Parallelism

	Improved <b>GPU</b> performance	Improved <b>CPU</b> performance
<b>CPU</b> -centric Strategy	×	☑

---

# Our Proposal

---

## Warp Scheduler Controls GPU Thread-Level Parallelism

	Improved <b>GPU</b> performance	Improved <b>CPU</b> performance
<b>CPU</b> -centric Strategy	×	☑
<b>CPU-GPU</b> Balanced Strategy	☑	☑

---

# Our Proposal

## Warp Scheduler Controls GPU Thread-Level Parallelism

	Improved <b>GPU</b> performance	Improved <b>CPU</b> performance
<b>CPU</b> -centric Strategy	×	☑
<b>CPU-GPU</b> Balanced Strategy	☑	☑

Control the trade-off

# Our Proposal

---

## **CPU-centric Strategy**

Memory Congestion 

CPU Performance 



# Our Proposal

---

## CPU-centric Strategy

Memory Congestion 

CPU Performance 

IF Memory Congestion 

 GPU TLP

---

# Our Proposal

---

## CPU-centric Strategy

Memory Congestion 

CPU Performance 

IF Memory Congestion   
 GPU TLP

Results Summary:

+24% CPU & -11% GPU

---



# Our Proposal

---

## CPU-centric Strategy

Memory Congestion 

CPU Performance 

## CPU-GPU Balanced Strategy

GPU TLP   

GPU Latency Tolerance 

IF Memory Congestion   
 GPU TLP

Results Summary:

+24% CPU & -11% GPU

---

# Our Proposal

---

## CPU-centric Strategy

Memory Congestion 

CPU Performance 

## CPU-GPU Balanced Strategy

GPU TLP   

GPU Latency Tolerance 

IF Memory Congestion   
 GPU TLP

IF Latency Tolerance   
 GPU TLP

Results Summary:

+24% CPU & -11% GPU

---

# Our Proposal

---

## CPU-centric Strategy

Memory Congestion 

CPU Performance 

## CPU-GPU Balanced Strategy

GPU TLP   

GPU Latency Tolerance 

IF Memory Congestion   
 GPU TLP

IF Latency Tolerance   
 GPU TLP

Results Summary:

+24% CPU & -11% GPU

Results Summary:

+7% both CPU & GPU

---

# Managing GPU Concurrency in Heterogeneous Architectures

Onur Kayiran<sup>1</sup>,

Nachiappan CN<sup>1</sup>, Adwait Jog<sup>1</sup>, Rachata Ausavarungnirun<sup>2</sup>,

Mahmut T. Kandemir<sup>1</sup>, Gabriel H. Loh<sup>3</sup>, Onur Mutlu<sup>2</sup>, Chita R. Das<sup>1</sup>



**Carnegie Mellon**



<sup>1</sup> Penn State

<sup>2</sup> Carnegie Mellon

<sup>3</sup> AMD Research

# Managing GPU Concurrency in Heterogeneous Architectures

Onur Kayiran<sup>1</sup>,

Nachiappan CN<sup>1</sup>, Adwait Jog<sup>1</sup>, Rachata Ausavarungnirun<sup>2</sup>,

Mahmut T. Kandemir<sup>1</sup>, Gabriel H. Loh<sup>3</sup>, Onur Mutlu<sup>2</sup>, Chita R. Das<sup>1</sup>



Carnegie Mellon



<sup>1</sup> Penn State

<sup>2</sup> Carnegie Mellon

<sup>3</sup> AMD Research

Today

Session 1B – Main Auditorium

@ 3 pm

*next paper*















**WIKIPEDIA**  
The Free Encyclopedia

- [Main page](#)
- [Contents](#)
- [Featured content](#)
- [Current events](#)
- [Random article](#)
- [Donate to Wikipedia](#)
- [Wikimedia Shop](#)

Interaction

- [Help](#)
- [About Wikipedia](#)
- [Community portal](#)
- [Recent changes](#)

[Article](#) [Talk](#)


## Microarchitecture

From Wikipedia, the free encyclopedia

*"Computer organization" redirects here. For another classification, see Flynn's taxonomy. For another classification, see Flynn's taxonomy.*

 This article is **Un sourced**.

In electronics engineering and computer science, **computer organization**, is the way a given ISA may be implemented with different hardware design or due to shifts in technology. **Computer architecture** is the combination



**WIKIPEDIA**  
The Free Encyclopedia

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikimedia Shop](#)


Interaction  
[Help](#)  
[About Wikipedia](#)  
[Community portal](#)  
[Recent changes](#)

[Article](#) [Talk](#)

## Microarchitecture

From Wikipedia, the free encyclopedia

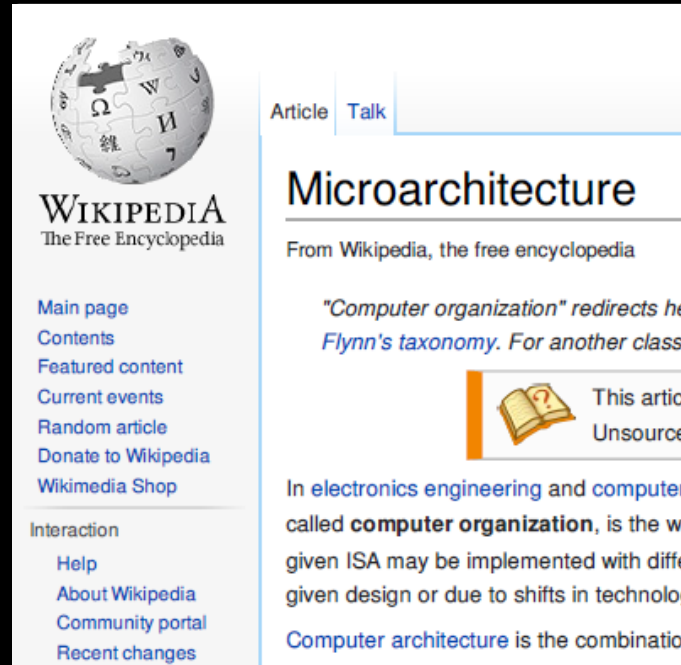
*"Computer organization" redirects here. For another classification, see Flynn's taxonomy. For another classification, see Flynn's taxonomy.*

 This article is **Unsources**

In electronics engineering and computer science, **computer organization**, is the way a given ISA may be implemented with different design or due to shifts in technology. **Computer architecture** is the combination



*“Wikipedia is the best thing ever...”*



*“...Anyone in the world can write anything they want about any subject. So you know you are getting the best possible information.” – Michael Scott, Dunder Mifflin*

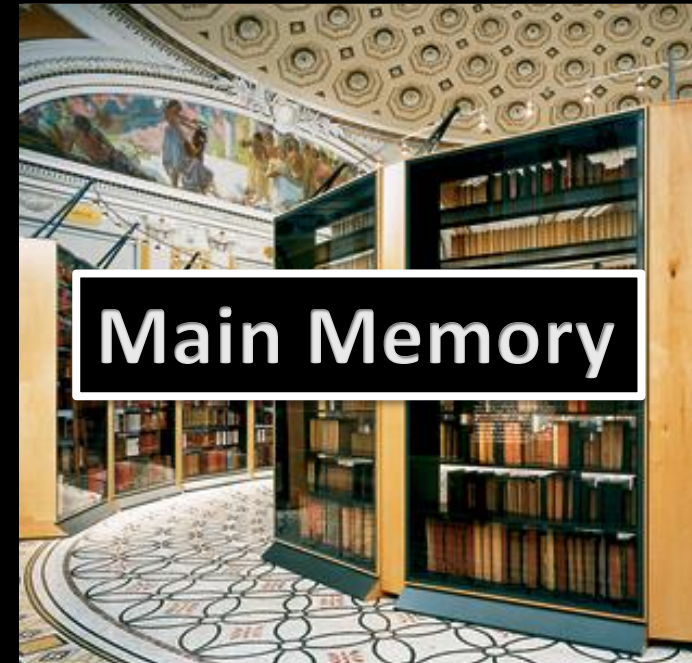
*“Wikipedia is the best thing ever...”*



**Private Cache**



**Approximator**



**Main Memory**

*“...Anyone in the world can write anything they want about any subject. So you know you are getting the best possible information.” – Michael Scott, Dunder Mifflin*

*“Load Value Approximation is the best thing ever”*

Joshua San Miguel  
Mario Badr  
Natalie Enright Jerger

3:55pm, Session 2A, Main Auditorium



*next paper*

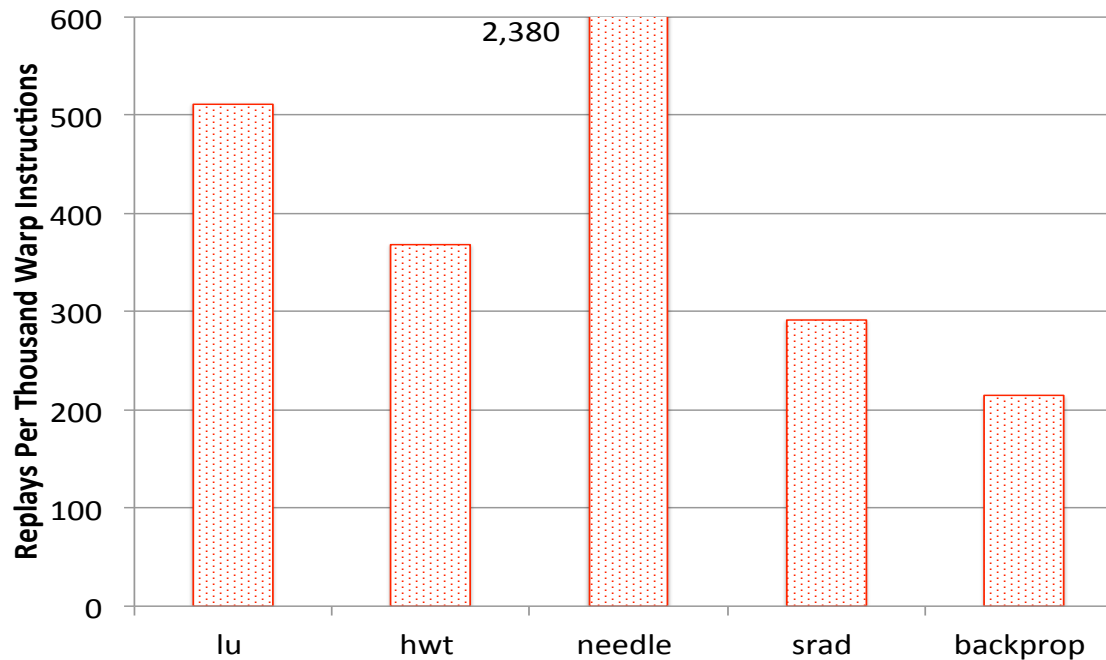


# Arbitrary Modulus Indexing

Jeffrey R. Diamond, Donald S. Fussell

(University of Texas at Austin)

Stephen W. Keckler (NVIDIA Corporation)



Power-of-2 (PO2) Indexing Leads To Conflicts 1





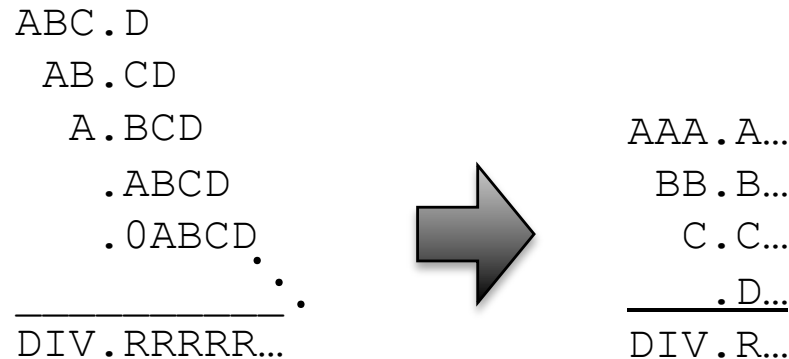
# Arbitrary Modulus Indexing (AMI)

1980s: Extensive NPO2 Indexing Research

- Few moduli implemented efficiently

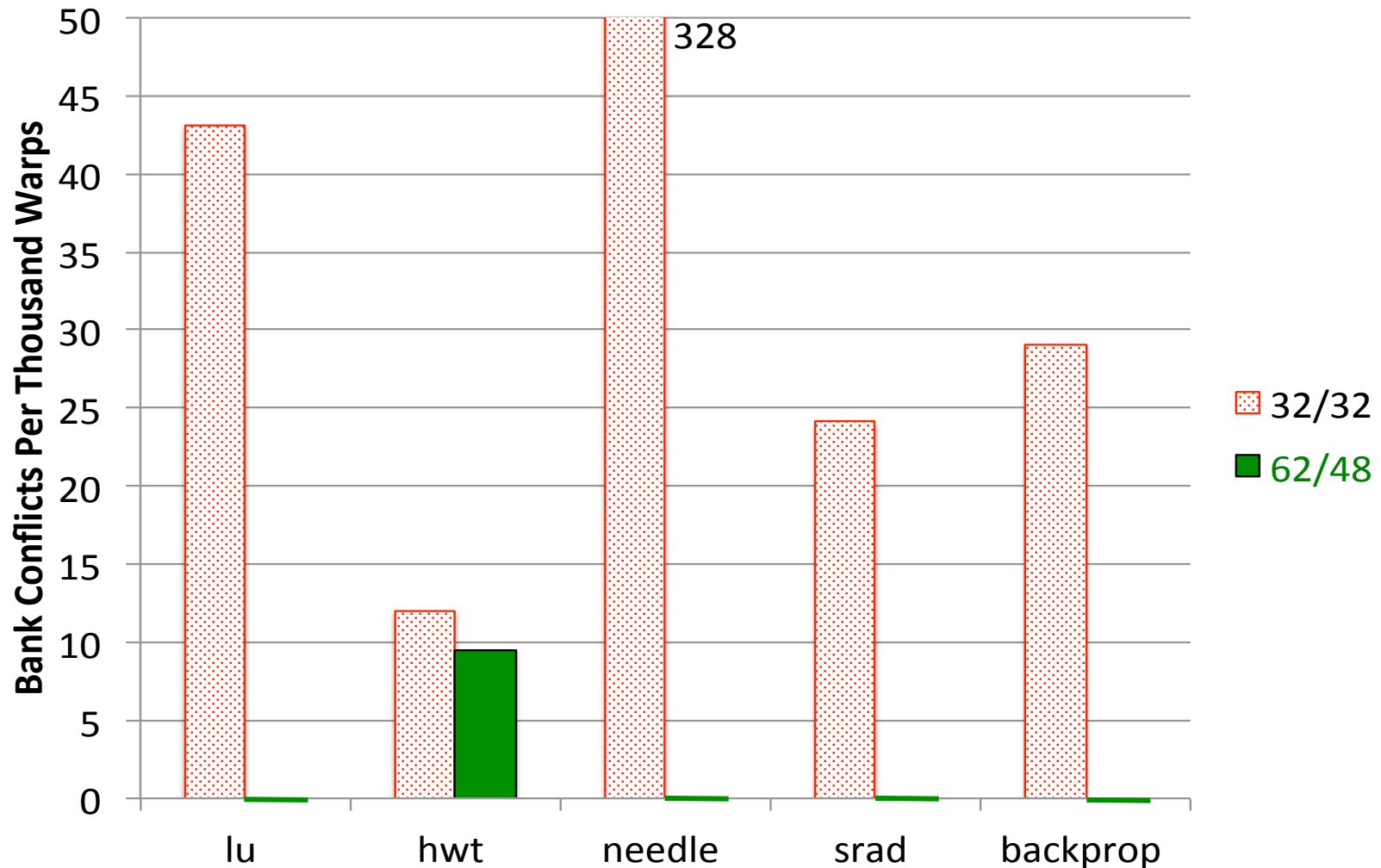
AMI can implement ANY moduli efficiently

- Found robust, novel moduli



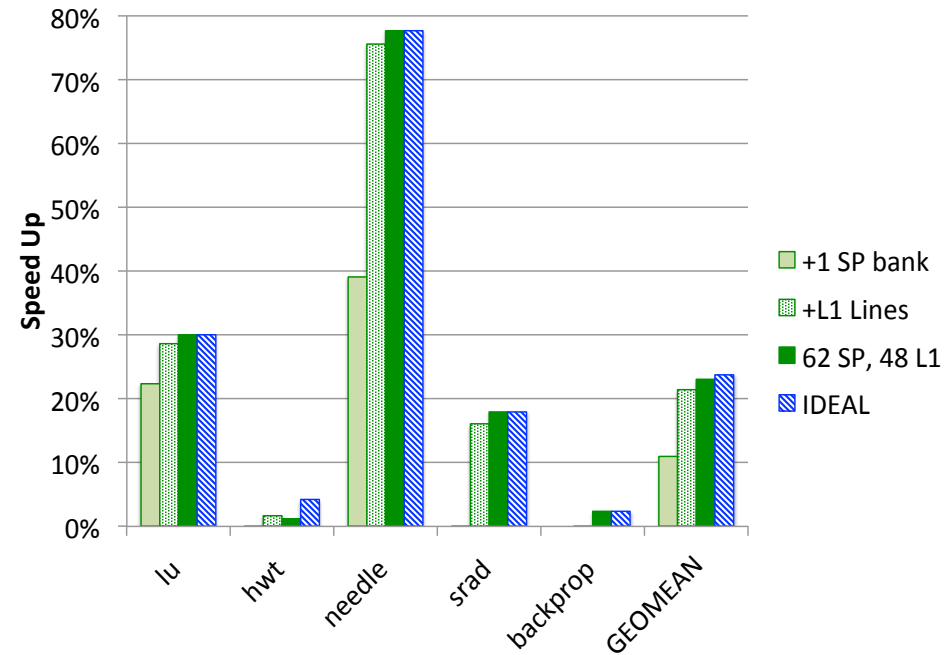
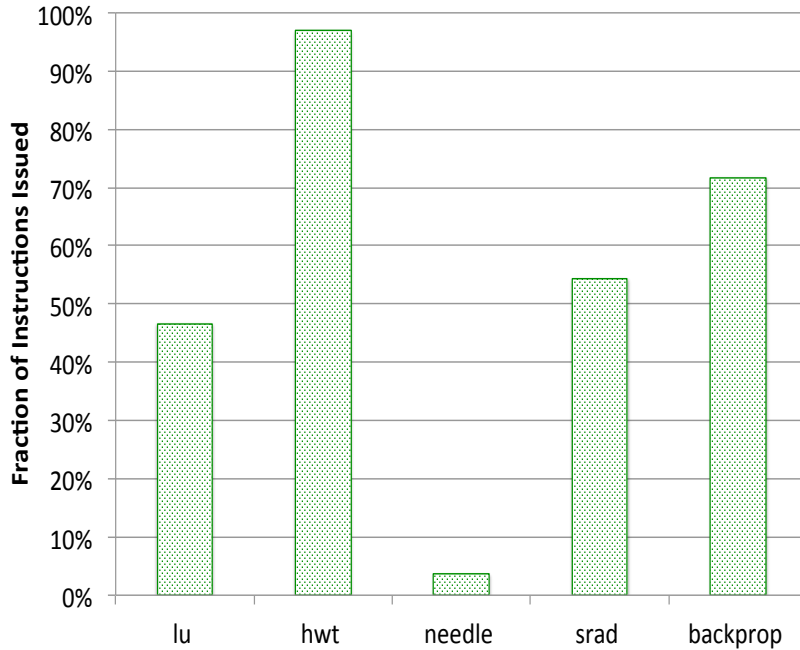


# AMI Eliminates Bank Conflicts





# AMI Improves Power And Performance



*next paper*

# Why Another Memory Control Scheme?



# Why Another Memory Control Scheme?

*FR-FCFS [ISCA'00]*





# Why Another Memory Control Scheme?

*PAR-BS [ISCA'08]*  
*STFM [MICRO'07]*  
*FR-FCFS [ISCA'00]*



# Why Another Memory Control Scheme?

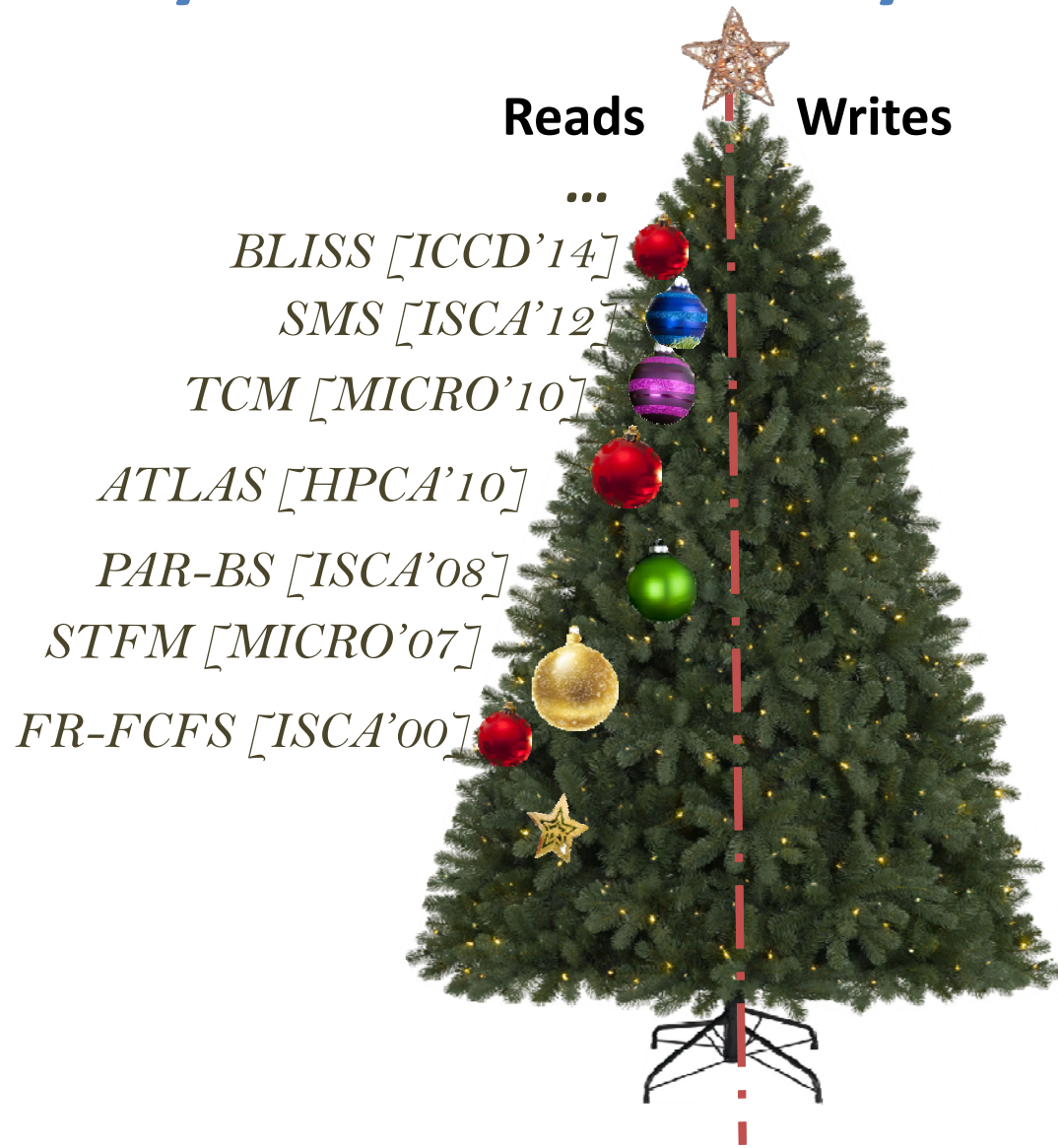
*TCM [MICRO'10]*  
*ATLAS [HPCA'10]*  
*PAR-BS [ISCA'08]*  
*STFM [MICRO'07]*  
*FR-FCFS [ISCA'00]*



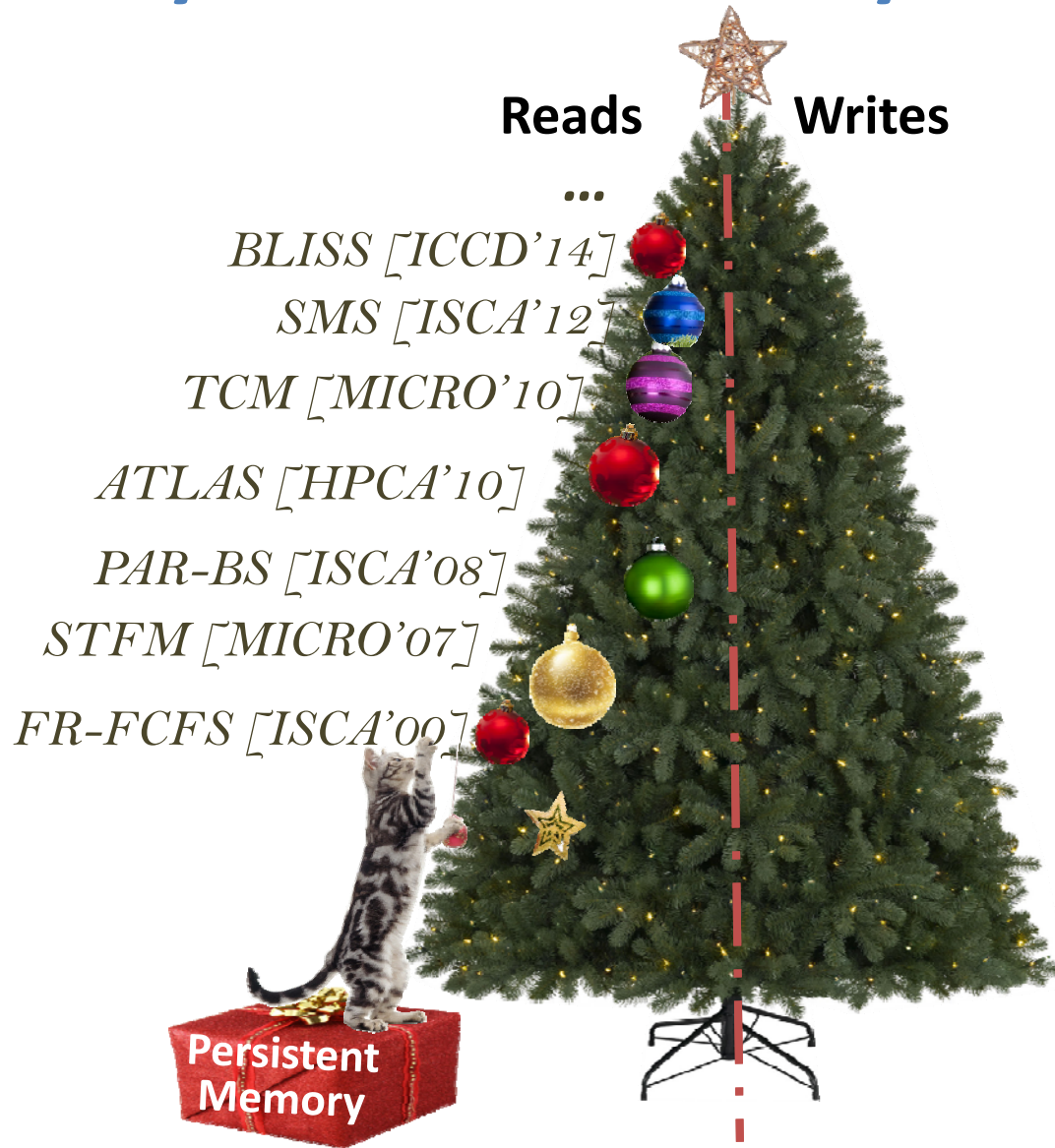
# Why Another Memory Control Scheme?



# Why Another Memory Control Scheme?



# Why Another Memory Control Scheme?



# Why Another Memory Control Scheme?



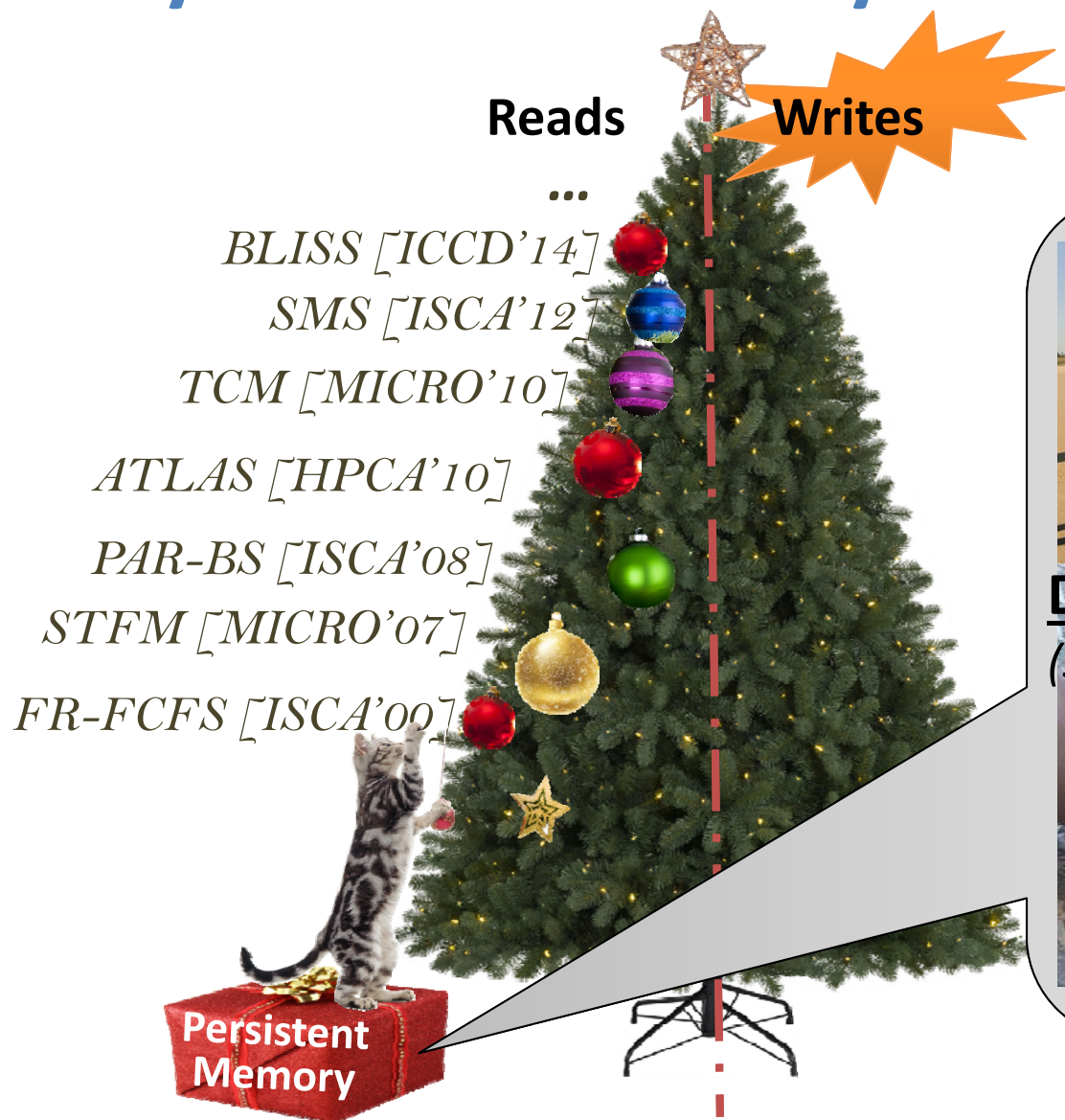
# Why Another Memory Control Scheme?



**Fast Load/Store**  
(Memory Attribute)

**Data Persistence**  
(Storage Attribute)

# Why Another Memory Control Scheme?





# Why Another Memory Control Scheme?



# Our Design: Towards A **FIRM** Tree





**Fairness**



**Performance**

**1.2x**

vs. the best of  
5 previous scheduler designs

# Details about “FIRM”

*Fair and High-Performance  
Memory Control for  
Persistent Memory Systems*

Jishen Zhao  
Onur Mutlu  
Yuan Xie

*HP Labs/Penn State/CMU  
CMU  
UCSB/Penn State/AMD Research*



**Presentation: Mon (Dec.15) 3:30 PM**  
**Session 2A**  
(Room: Dining Hall)

**Poster Session: Tue (Dec. 16) 12:00 PM**



*next paper*

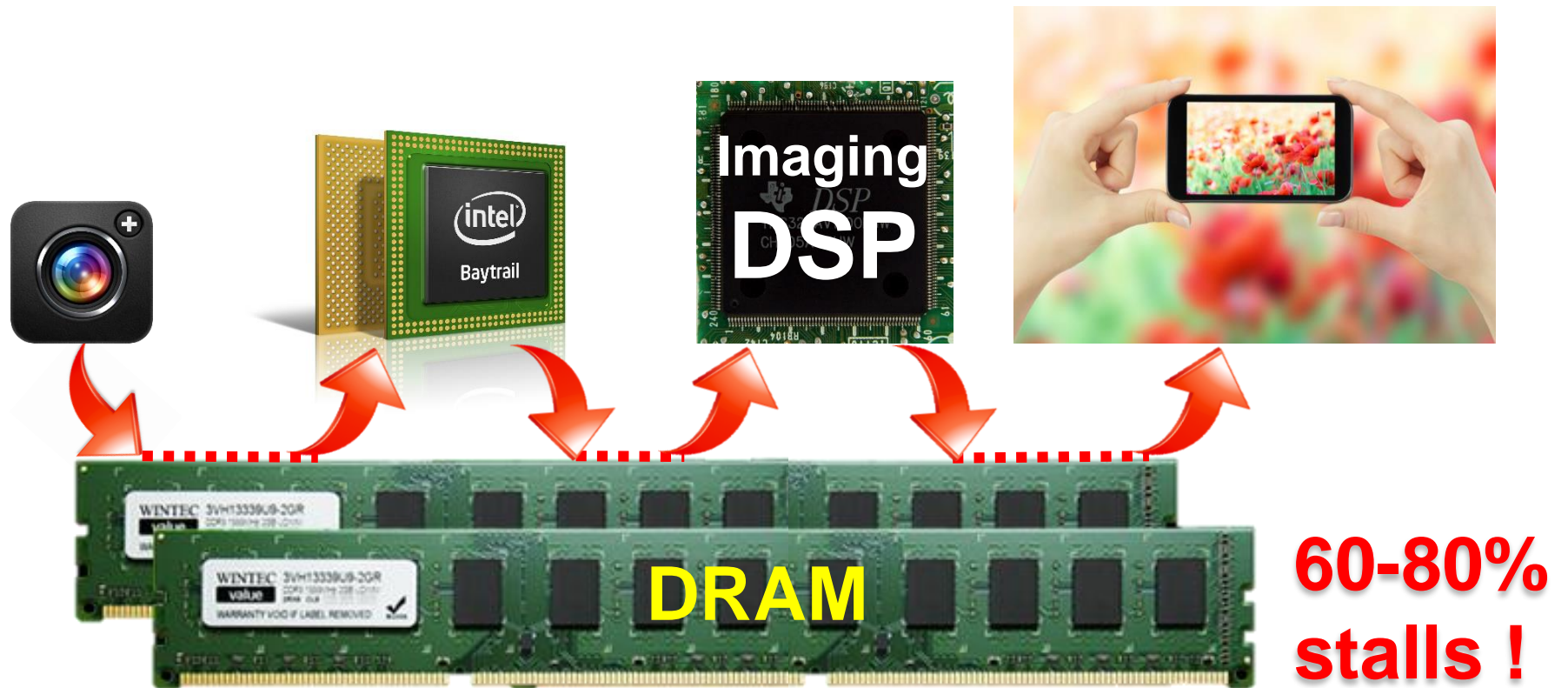
# Short-Circuiting Memory Traffic in Handheld Platforms

Praveen Yedlapalli, Nachi Chidambaram. N., \*Niranjan Soundararajan,

Anand Sivasubramaniam, Mahmut Kandemir, Chita R. Das

*The Pennsylvania State University*

*\*Intel Corp.*



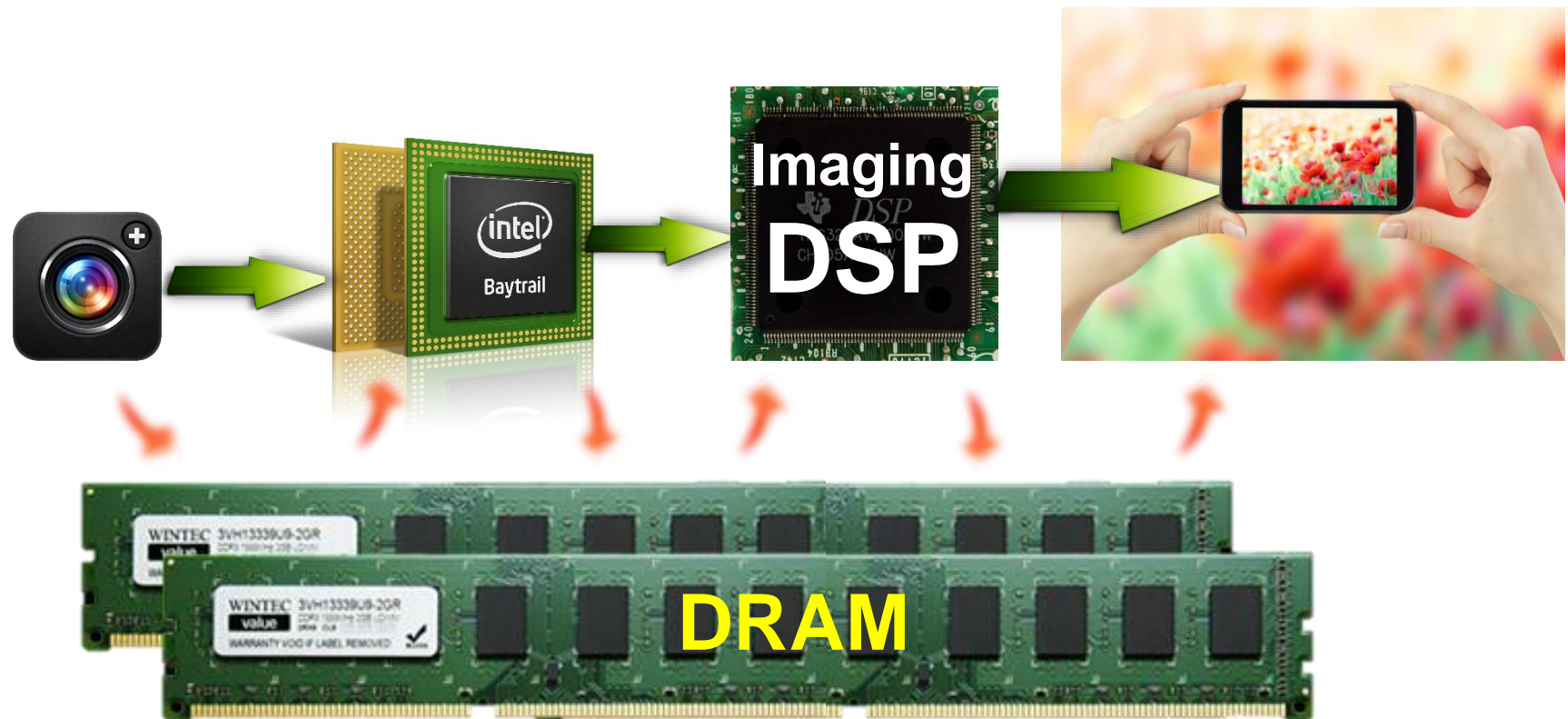
# Short-Circuiting Memory Traffic in Handheld Platforms

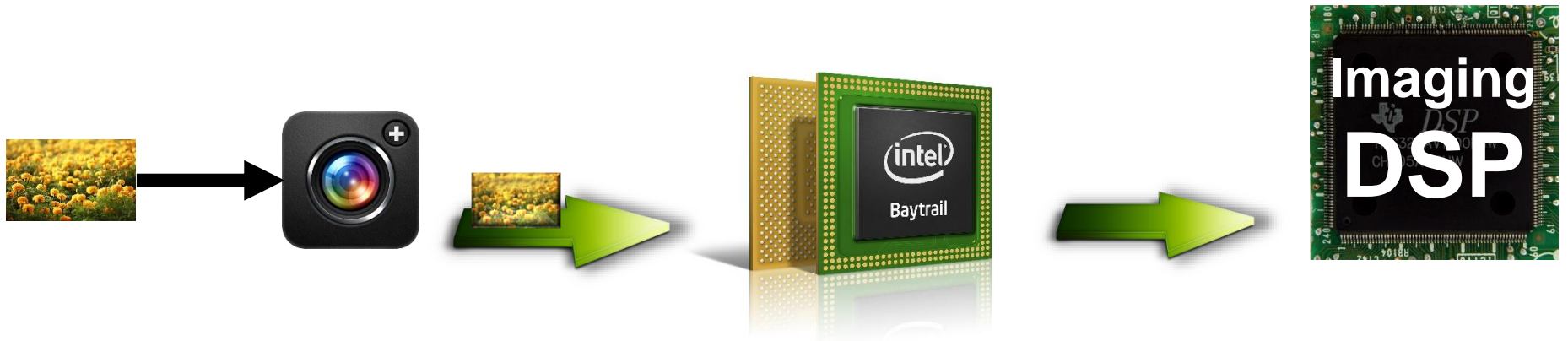
Praveen Yedlapalli, Nachi Chidambaram. N., \*Niranjan Soundararajan,

Anand Sivasubramaniam, Mahmut Kandemir, Chita R. Das

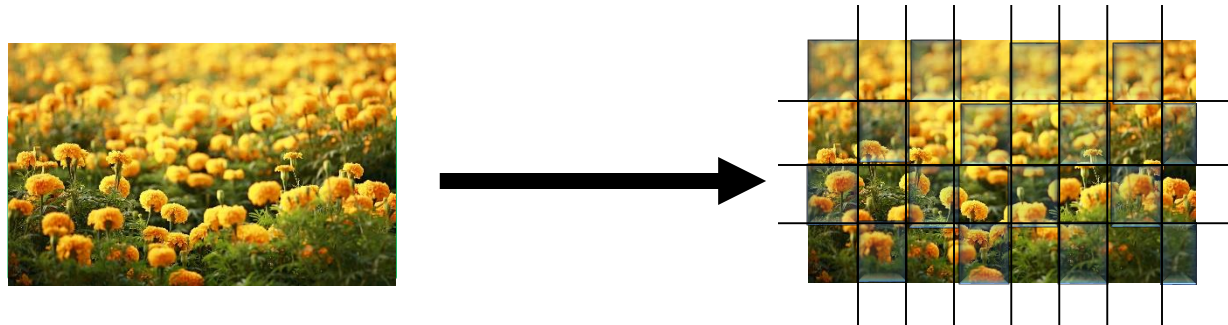
*The Pennsylvania State University*

*\*Intel Corp.*

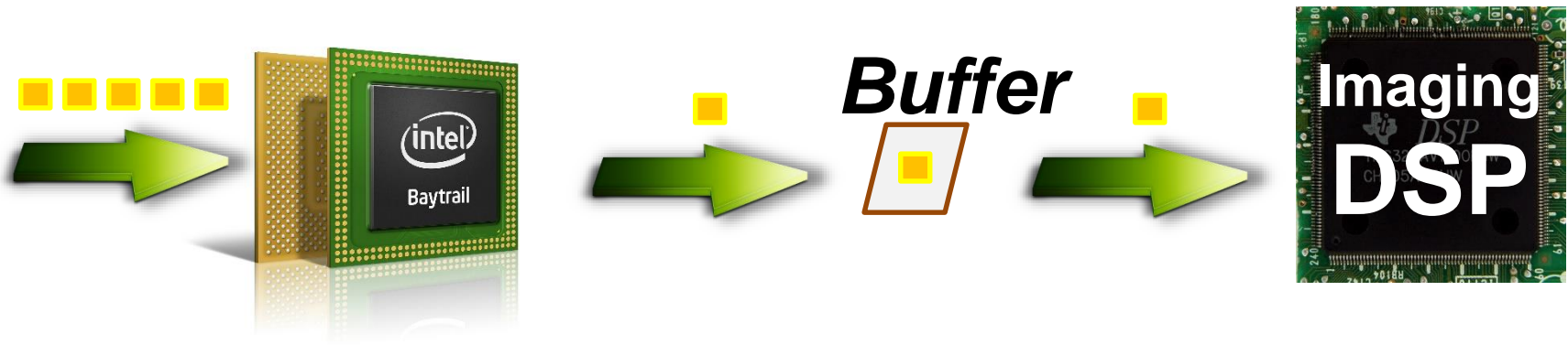




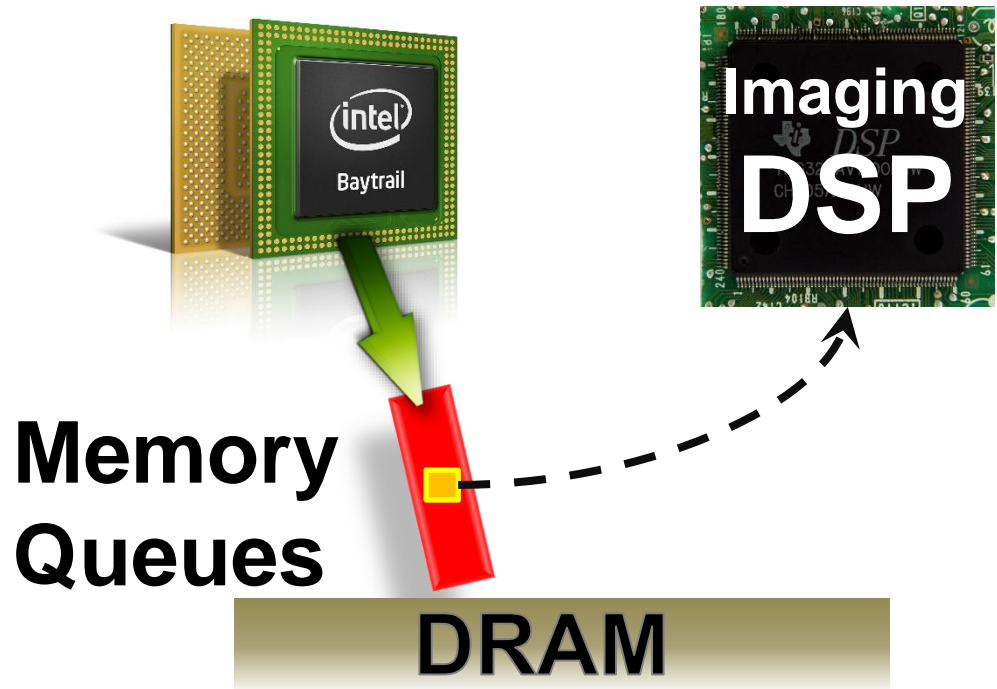
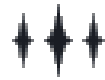
## “Sub-Frame”







## Sub-frame buffers



**Store** → **load forwarding**

- 33 %      Reduced DRAM energy**
- 35+%      Reduced active cycles for IP**
- 45 %      Reduced Cycles Per Frame**
- ~15%      Increase in FPS**

**Talk on Dec-15<sup>th</sup> - 15:55 - 18:00 Session 2A**

**See you all at the poster session!**

*next paper*

# Efficient Memory Virtualization

## Reducing Dimensionality of Nested Page Walks

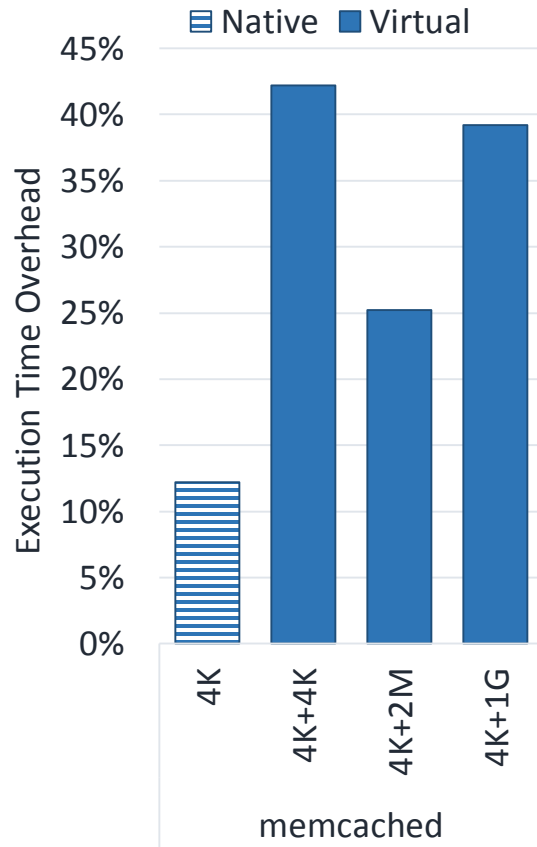
**Jayneel Gandhi**, Arkaprava Basu,  
Mark D. Hill, Michael M. Swift



*TLB misses are very costly in virtual servers.*

—Buell, et al. VMware Technical Journal 2013

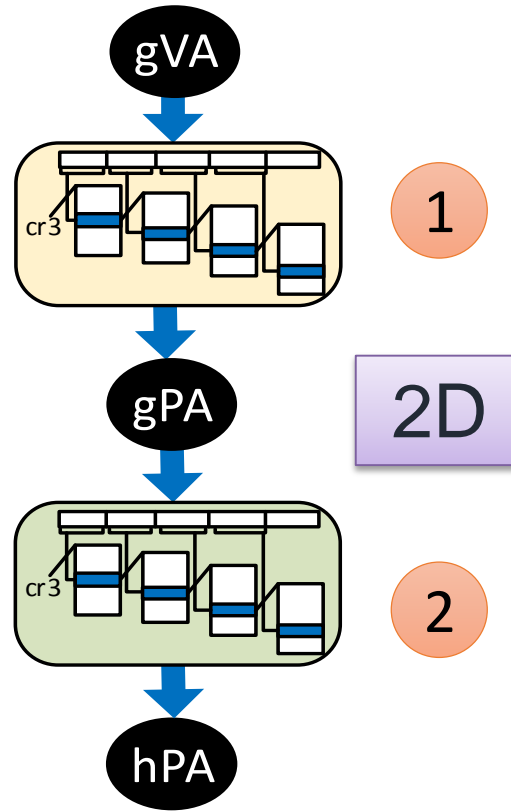
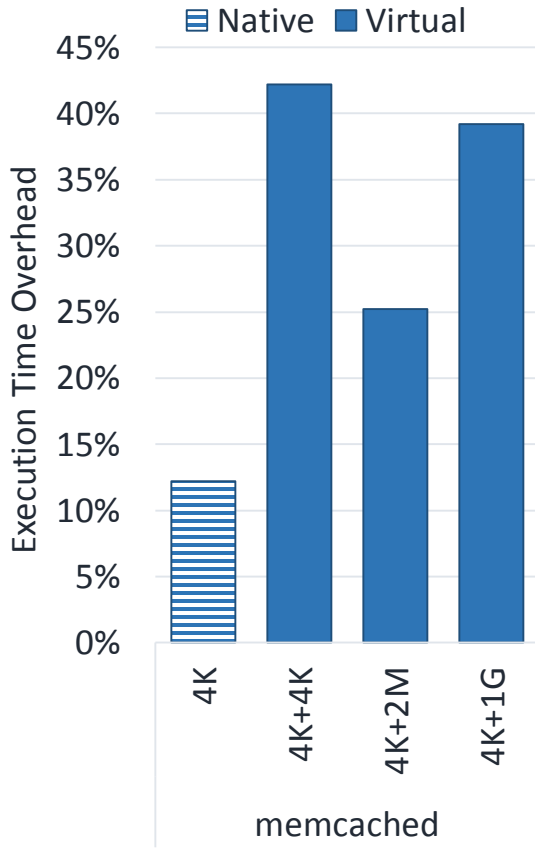
# Cost



3.6x increase in overheads

# Cost

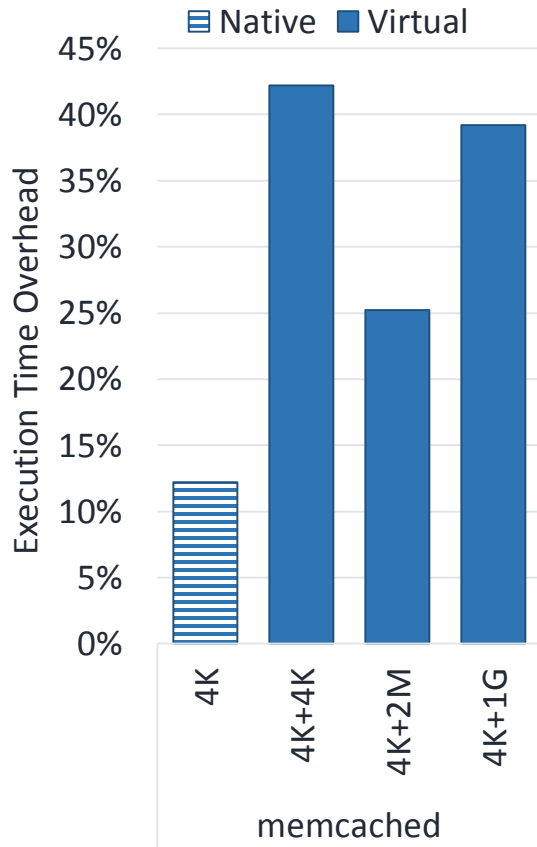
# Problem



3.6x increase in overheads

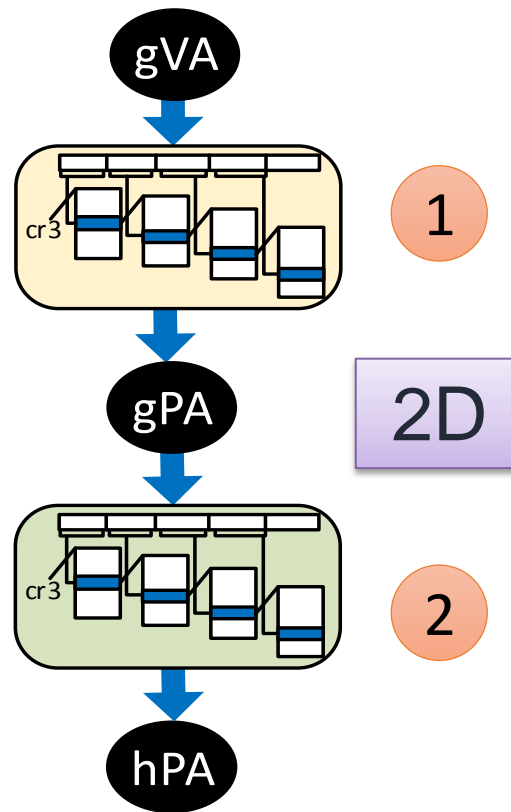
Up to 24 memory accesses

# Cost



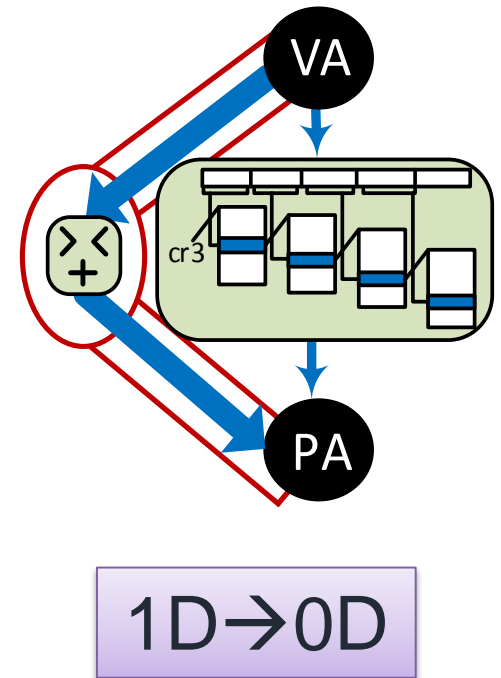
3.6x increase in overheads

# Problem



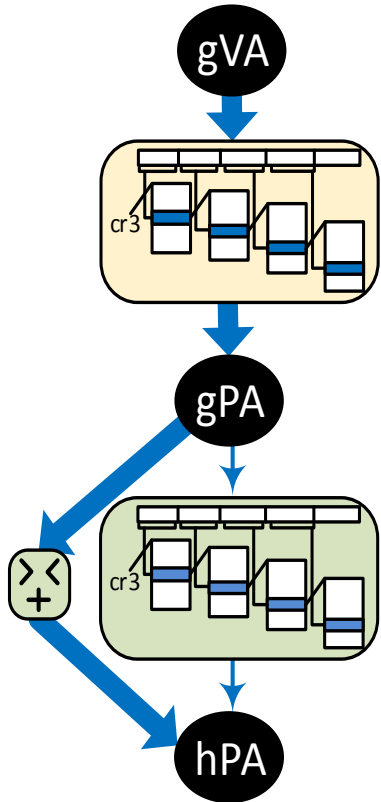
Up to 24 memory accesses

# Opportunity



Direct Segments

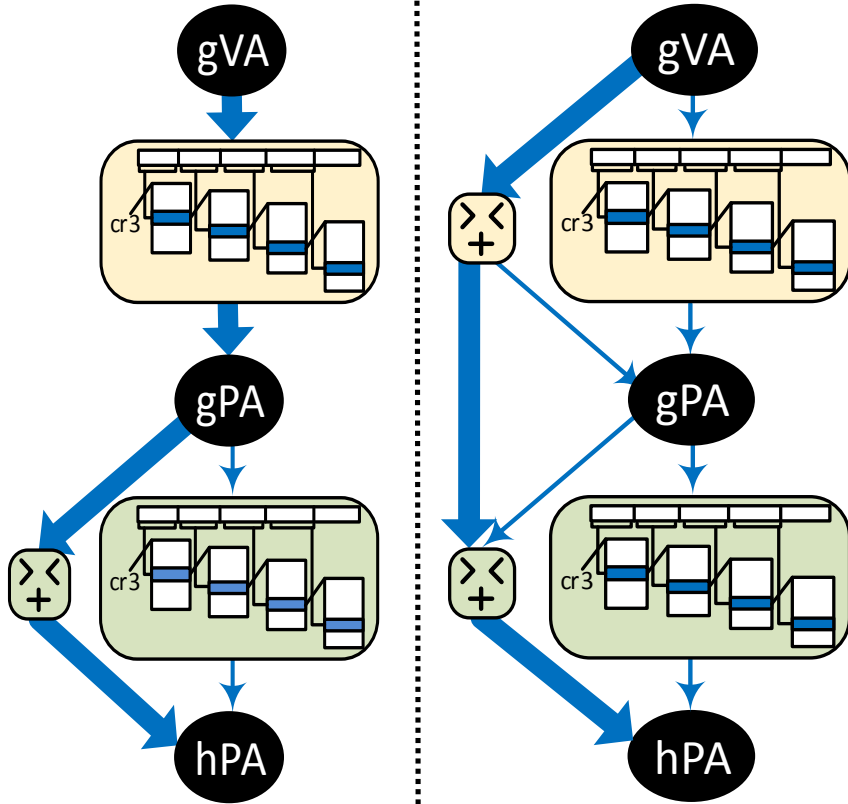
# Solution: Three Modes Extending Direct Segments



VMM Direct  
2D → 1D



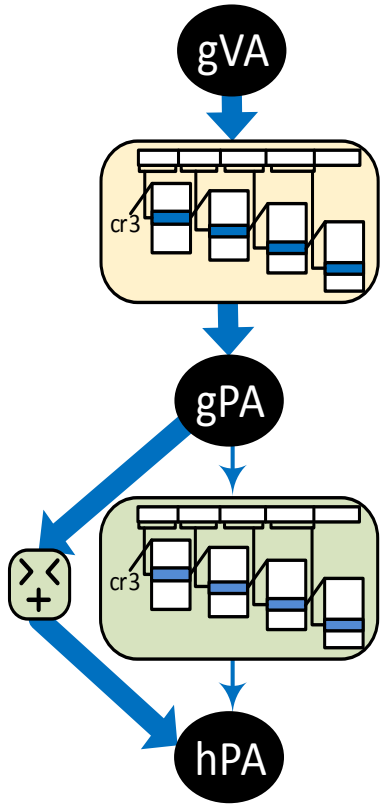
# Solution: Three Modes Extending Direct Segments



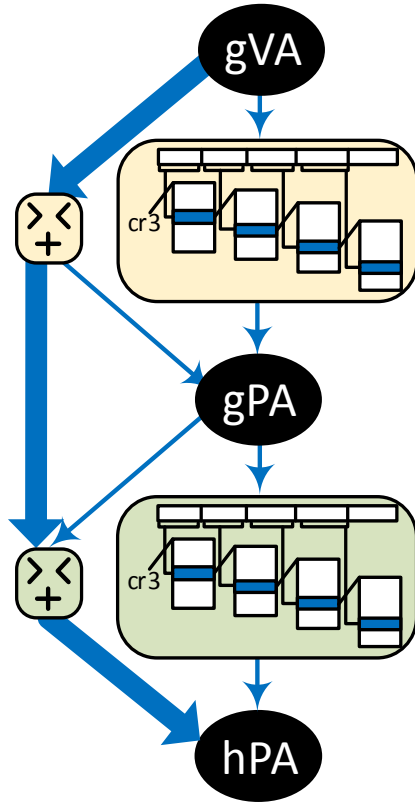
VMM Direct  
2D → 1D

Dual Direct  
2D → 0D

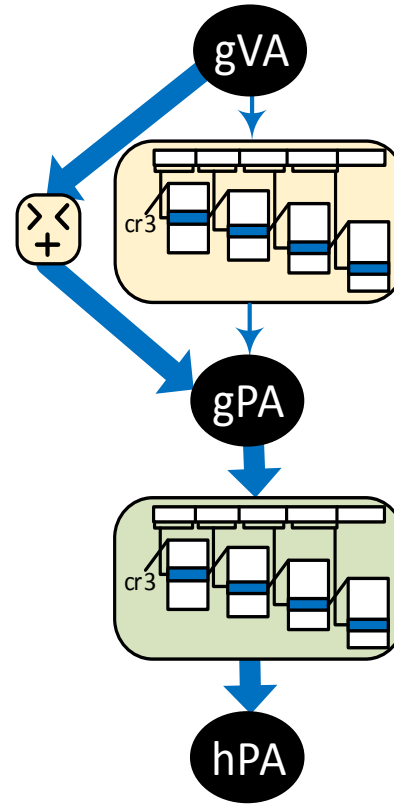
# Solution: Three Modes Extending Direct Segments



VMM Direct  
2D → 1D

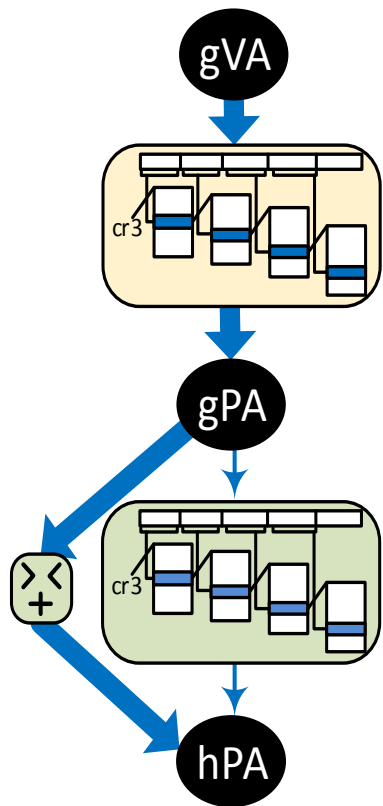


Dual Direct  
2D → 0D

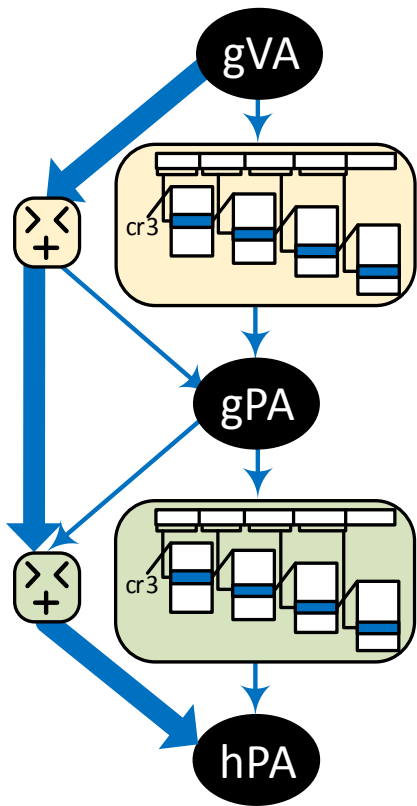


Guest Direct  
2D → 1D

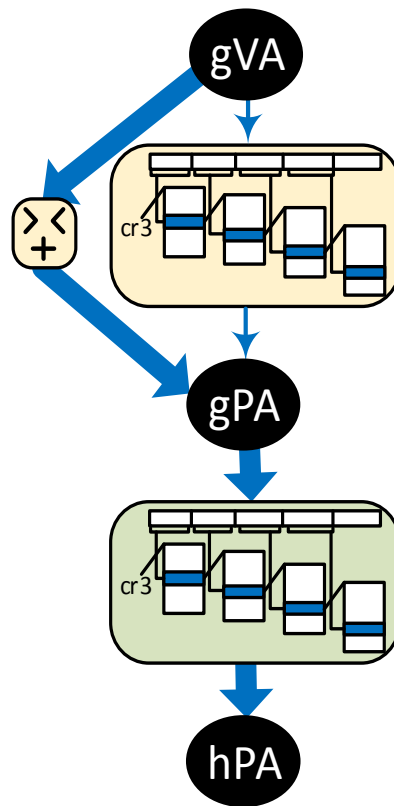
# Solution: Three Modes Extending Direct Segments



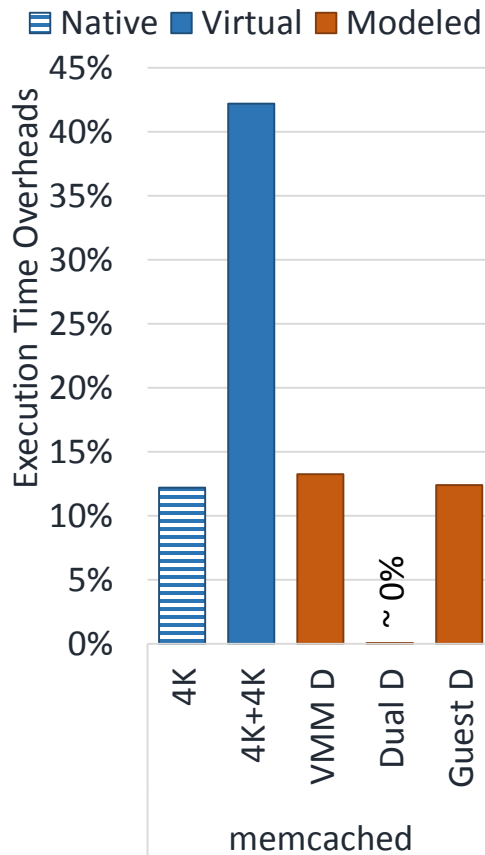
VMM Direct  
2D → 1D



Dual Direct  
2D → 0D



Guest Direct  
2D → 1D



Minimal  
Overheads

# Optimization

Escape Filter: Permanent “hard” memory faults

# Optimization

Escape Filter: Permanent “hard” memory faults

Please come to our talk

**Today, Session: 2A, Main Auditorium**

**Efficient Memory Virtualization**  
Reducing Dimensionality of Nested Page Walks

**Jayneel Gandhi, Arkaprava Basu,**  
Mark D. Hill, Michael M. Swift



*next paper*

# Iso-X: A Flexible Architecture for Hardware-Managed Isolated Execution

**Dmitry Evtushkin<sup>1</sup> Jesse Elwell<sup>1</sup> Meltem Ozsoy<sup>1</sup>**

**Dmitry Ponomarev<sup>1</sup> Nael Abu-Ghazaleh<sup>2</sup> Ryan Riley<sup>3</sup>**

<sup>1</sup>*State University of New York at  
Binghamton  
Department of Computer Science*

<sup>2</sup>*University of California at Riverside  
Department of Computer Science &  
Engineering*

<sup>3</sup>*Qatar University  
Department of Computer Science*

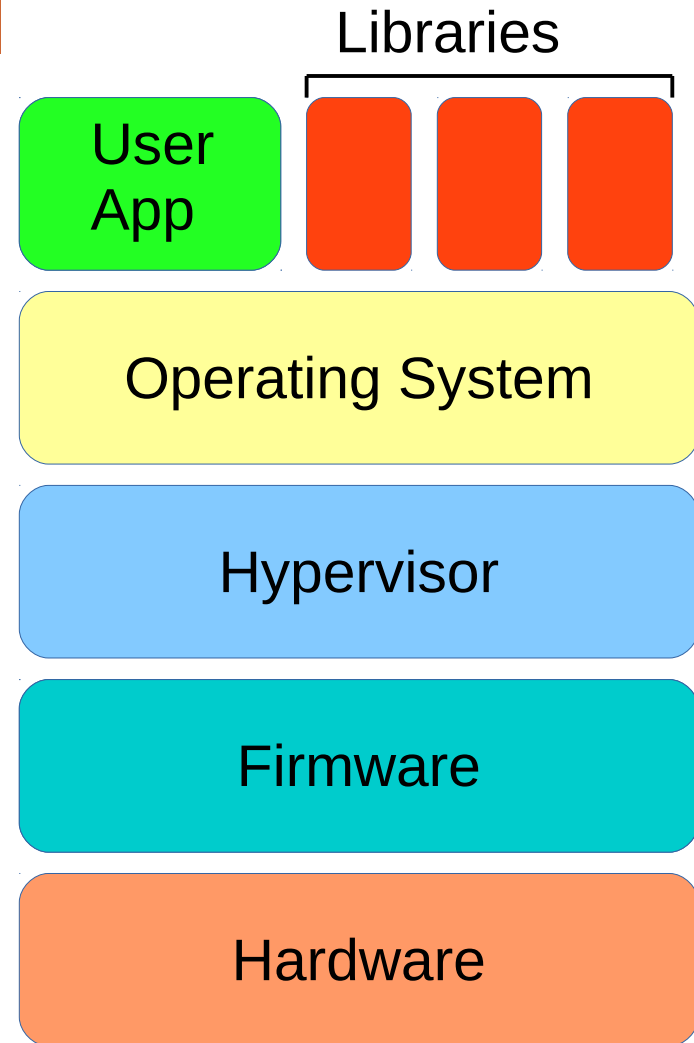
**The 47th Annual IEEE/ACM International Symposium on  
Microarchitecture**

December 14<sup>th</sup>, 2014



# Threat Model

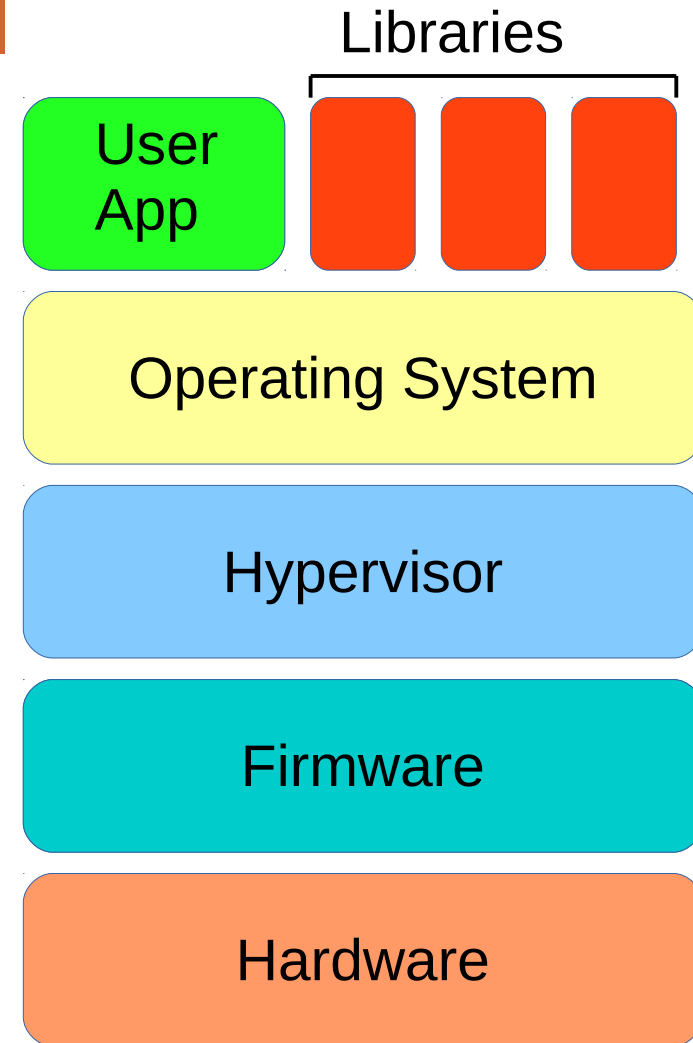
- Multi-layered TCB
- Large and complex software





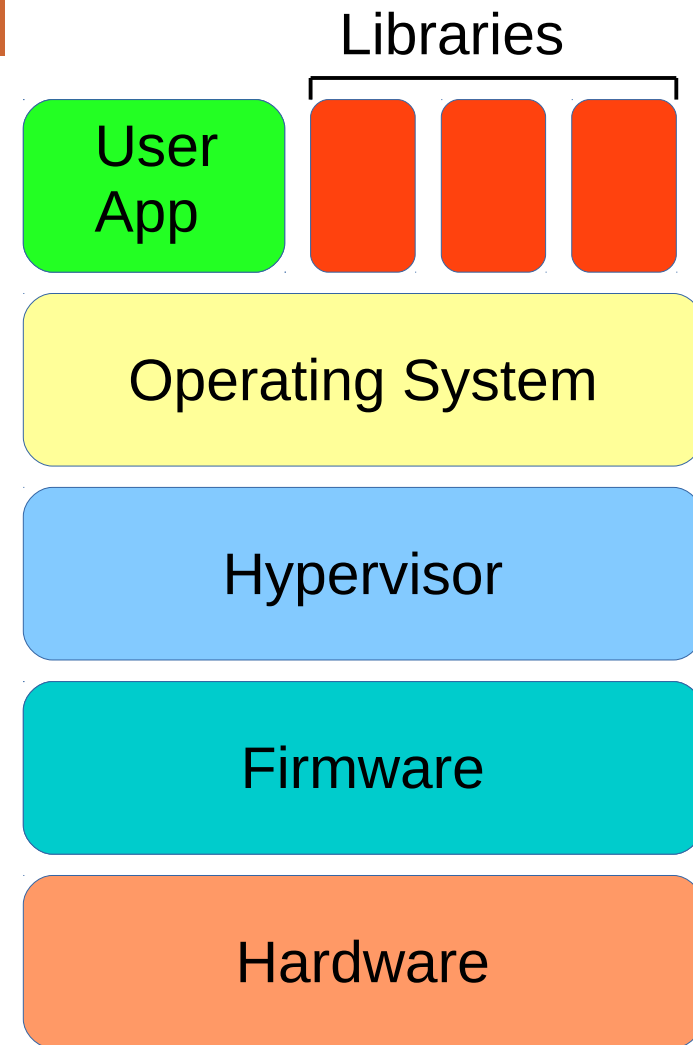
# Threat Model

- Multi-layered TCB
- Large and complex software



# Threat Model

- Multi-layered TCB
- Large and complex software

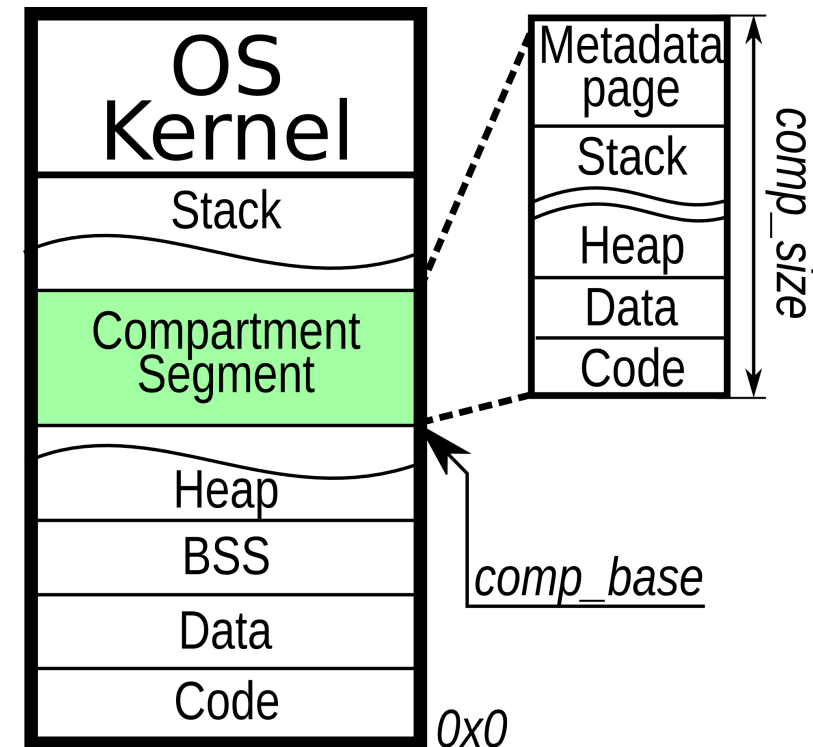


Hostile Software Environment

- **Iso-X** offers **isolated execution environment**
- Software modules can run in full isolation
- Only **Hardware** in Trusted Computing Base (TCB)

# Isolated Compartments

- Compartments reside in process' address space
  - Allows efficient interaction
  - Simple compartment booting process
- All code outside is untrusted
  - Used by compartment to communicate with outside world



# Iso-X Highlights

- Full featured execution environment for isolated compartments
- Relies only on a few simple data structures maintained in protected memory
- Attestation mechanism to protect against emulation
- Hardware explicitly controls every access to compartment memory
- Low performance impact, only **0.97%** on average

*next paper*

# Random Fill Cache Architecture

Fangfei Liu and Ruby B. Lee

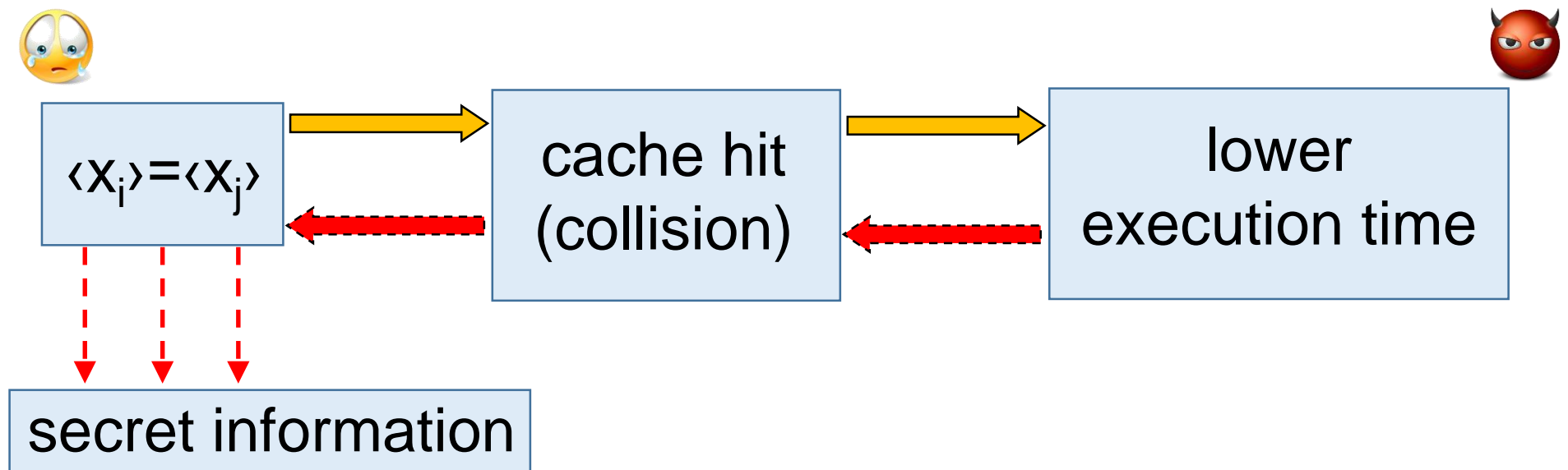
Princeton University

- Defends against challenging cache side-channel attacks, without impacting performance
- Unlike past work focused on cache contention-based attacks, we focus on “reuse-based attacks” – **strikes at heart of cache’s function!**

# Motivation

- Reuse of data may leak secret information

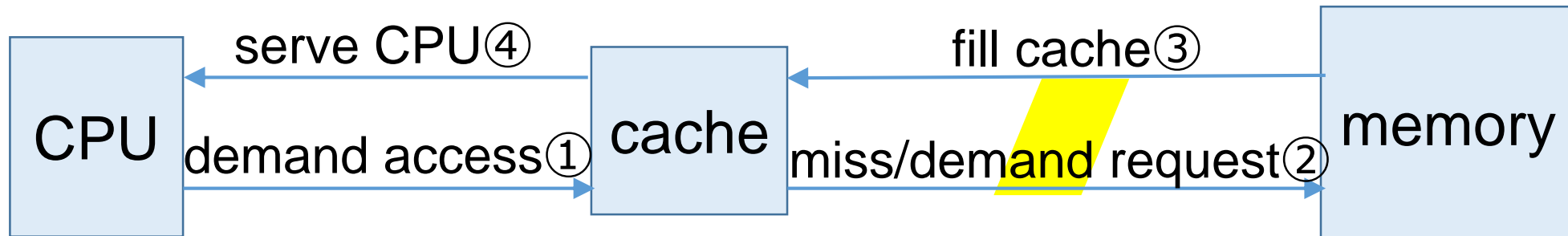
... $T[x]$ ,  $T[x_i]$ ,  $T[x]$ ,  $T[x]$ ,  $T[x_j]$ ,  $T[x]$ ...



- No resource contention (conflicting misses)
- Strike at cache's main purpose – cache hits

# Our discoveries and solutions

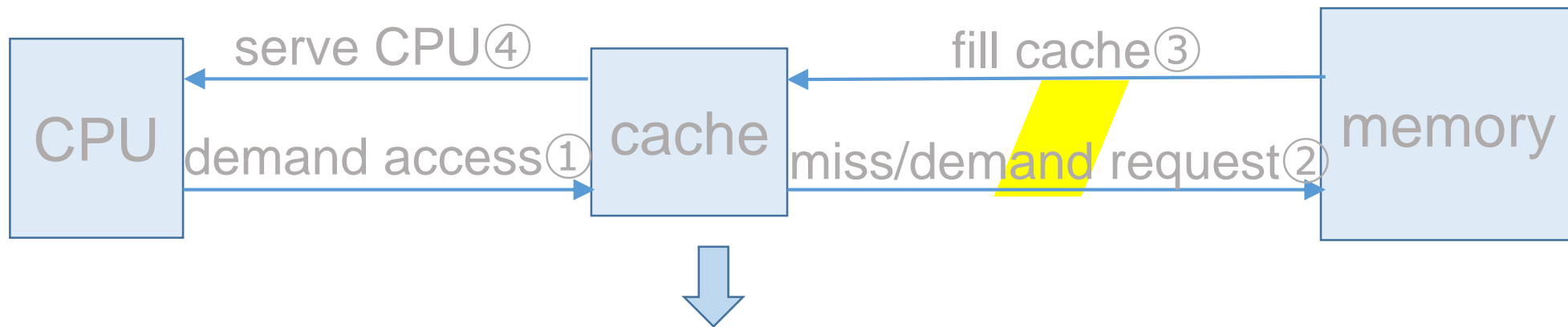
- Demand fetch policy is root cause of reuse-based attacks!!



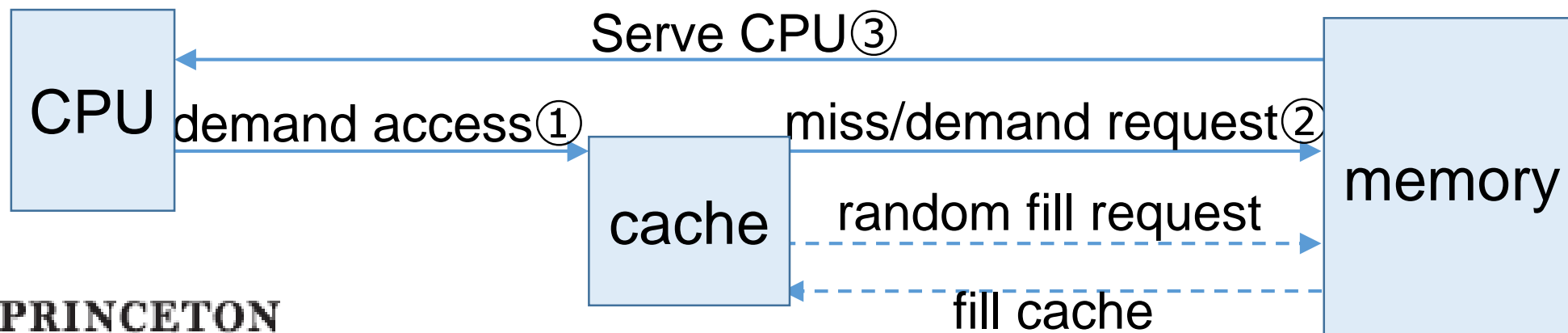


# Our discoveries and solutions

- Demand fetch policy is root cause of reuse-based attacks!!

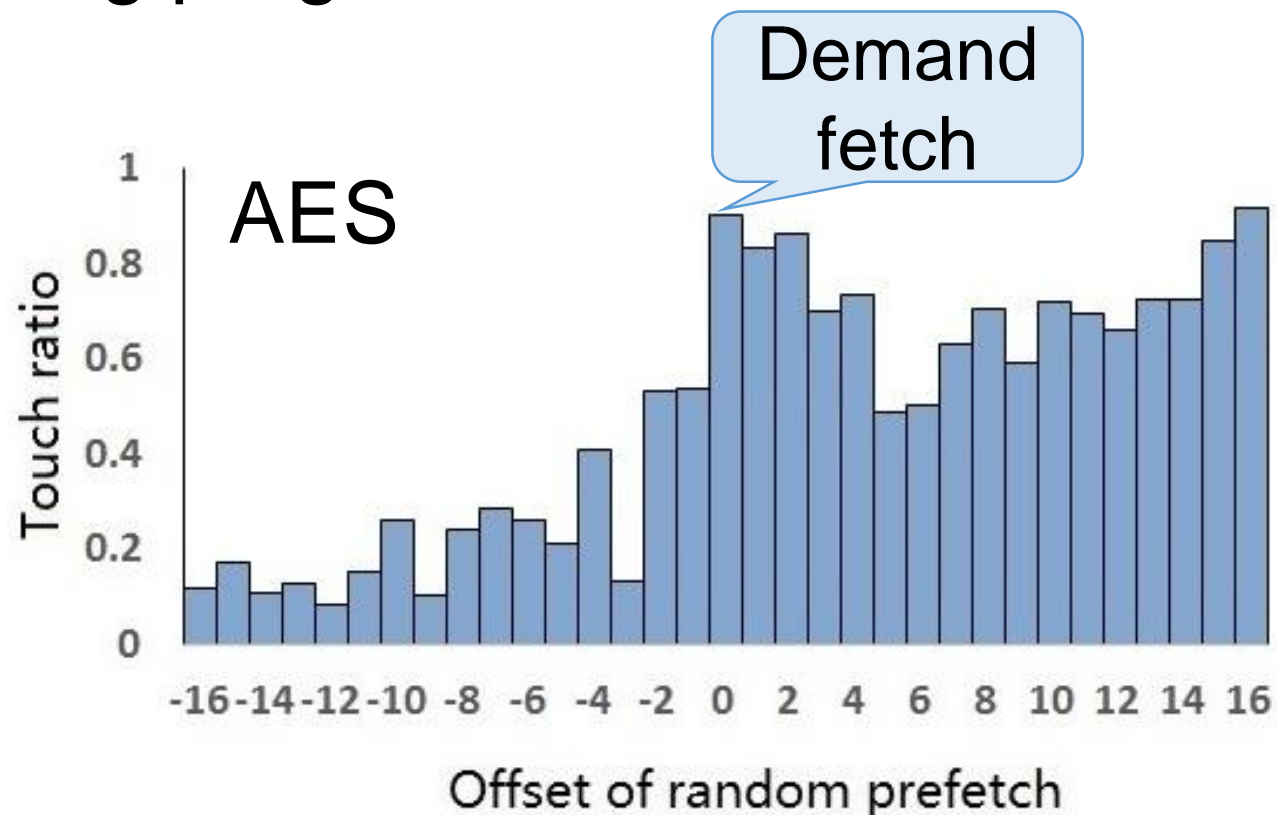


- Random fill policy: cache fill is **de-correlated** with a demand access



# Random fill within configurable neighborhood window

- Still take advantage of spatial locality – No performance degradation for crypto programs
- Even improves performance of some streaming programs



*next paper*

# CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware

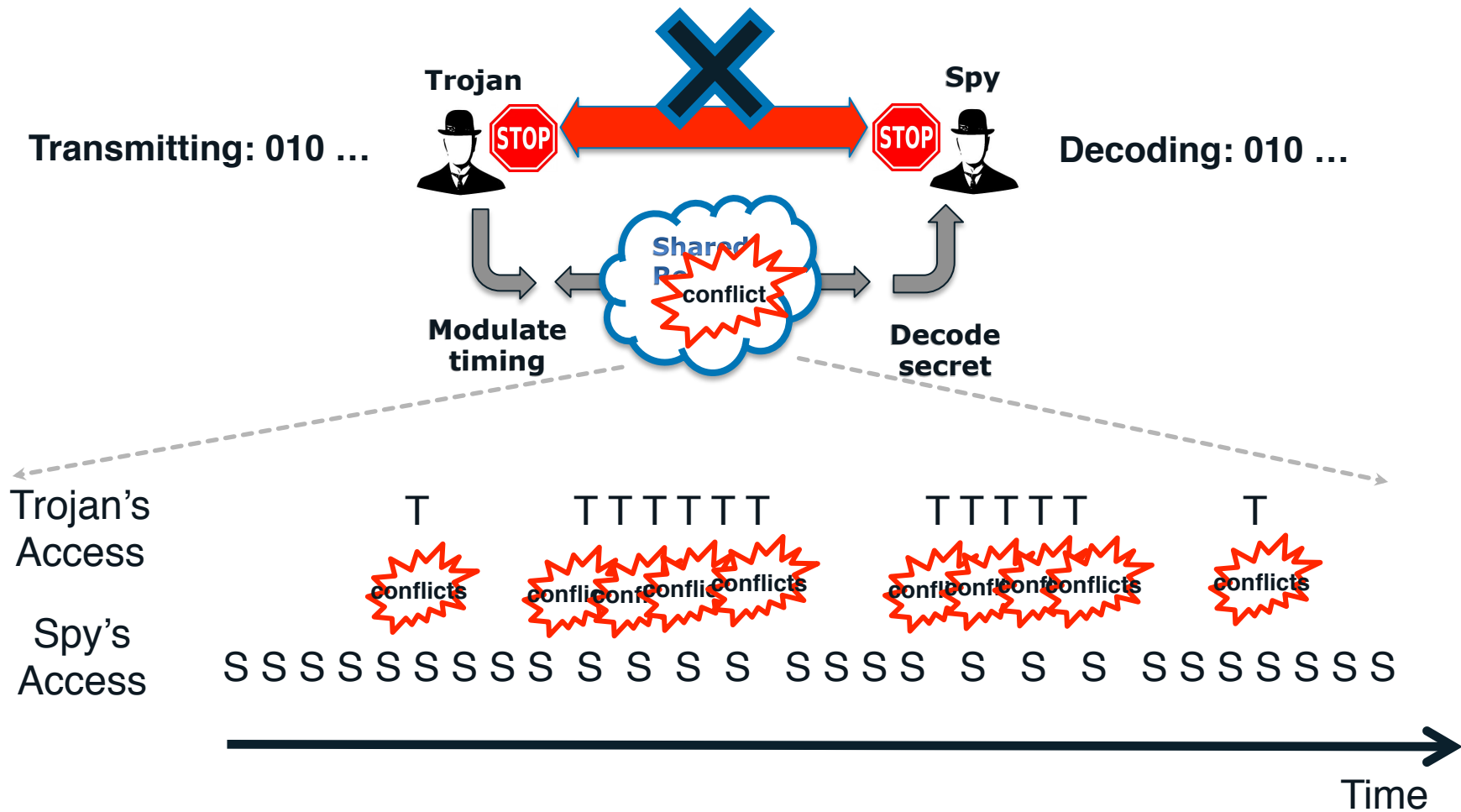
Jie Chen and Guru Venkataramani  
Department of Electrical and Computer Engineering  
The George Washington University  
Washington, DC

Today, 3:55 pm, Session 2B: Security, Room: Umney Theatre

## Information Leakage

- ★ **Unauthorized exposure of sensitive data**
  - Examples: Identify theft, credit card data breach
- ★ **Software confinement mechanisms continue to improve**
  - Attackers increasingly turn to hardware-based attacks

# Covert Timing Channel



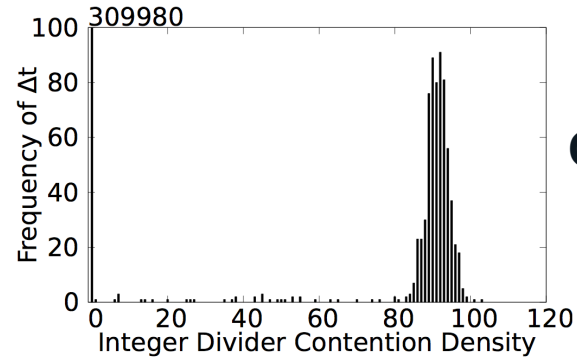
# Detecting Covert Timing Channels on Hardware

---

- ★ **We study conflicts on shared hardware resource and the associated events**
- ★ **We design algorithms that look for patterns of conflicts**
  - On combinational structures
    - Recurrent Burst Pattern
  - On memory structures
    - Oscillation Pattern
- ★ **Runtime detection capability**
  - Low-cost and efficient hardware-software support

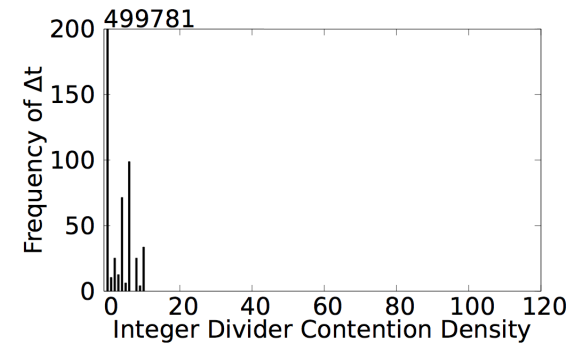
# Results

**Conflict Patterns  
for Covert Timing Channels**

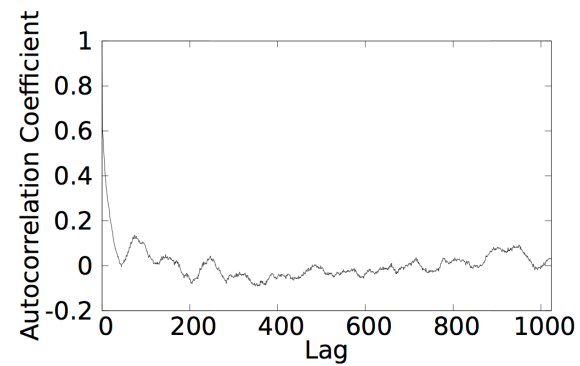
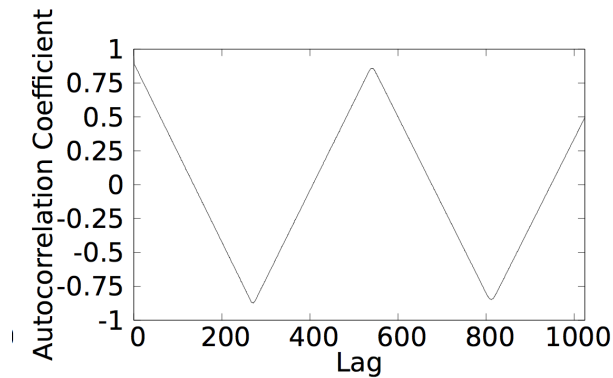


**On combinational  
structures**

**Conflict Patterns  
for regular benchmark applications**



**On memory  
structures**



★ **More details in the talk!**

Today, 3:55 pm, Session 2B: Security, Room: Umney Theatre

*next paper*



# Continuous, Low Overhead, Run-Time Validation of Program Executions

## The Problem and Existing Solutions

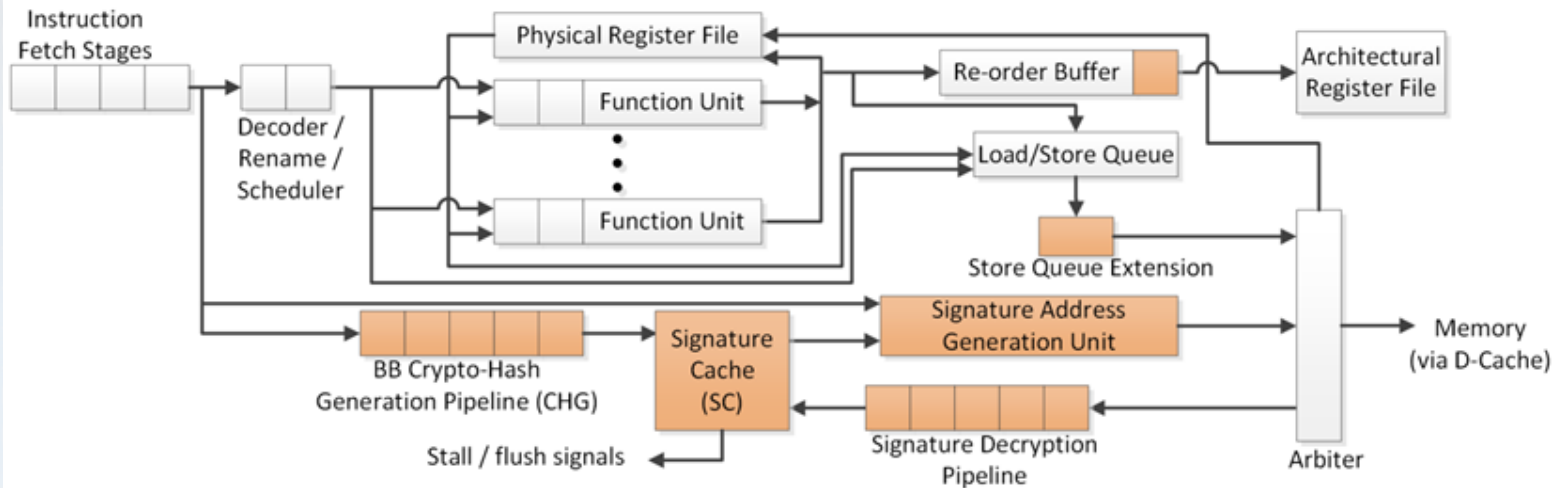
- **An execution is authenticated** if its control flow path and the instructions along that path were as intended: adversaries can exploit vulnerabilities to alter either of both at **run-time** or before the run starts
- **Validating the binaries *prior* to execution is thus not enough**
- **Most existing solutions that check for control flow integrity are geared towards specific types of attacks** (Vtable modification, ROP, JOP etc.)
- **Need an enduring and universal solution:**
  - Can handle all attacks against code and instruction integrity
  - Can handle future attacks that can compromise these

# Goal and Unique Aspects of this Work

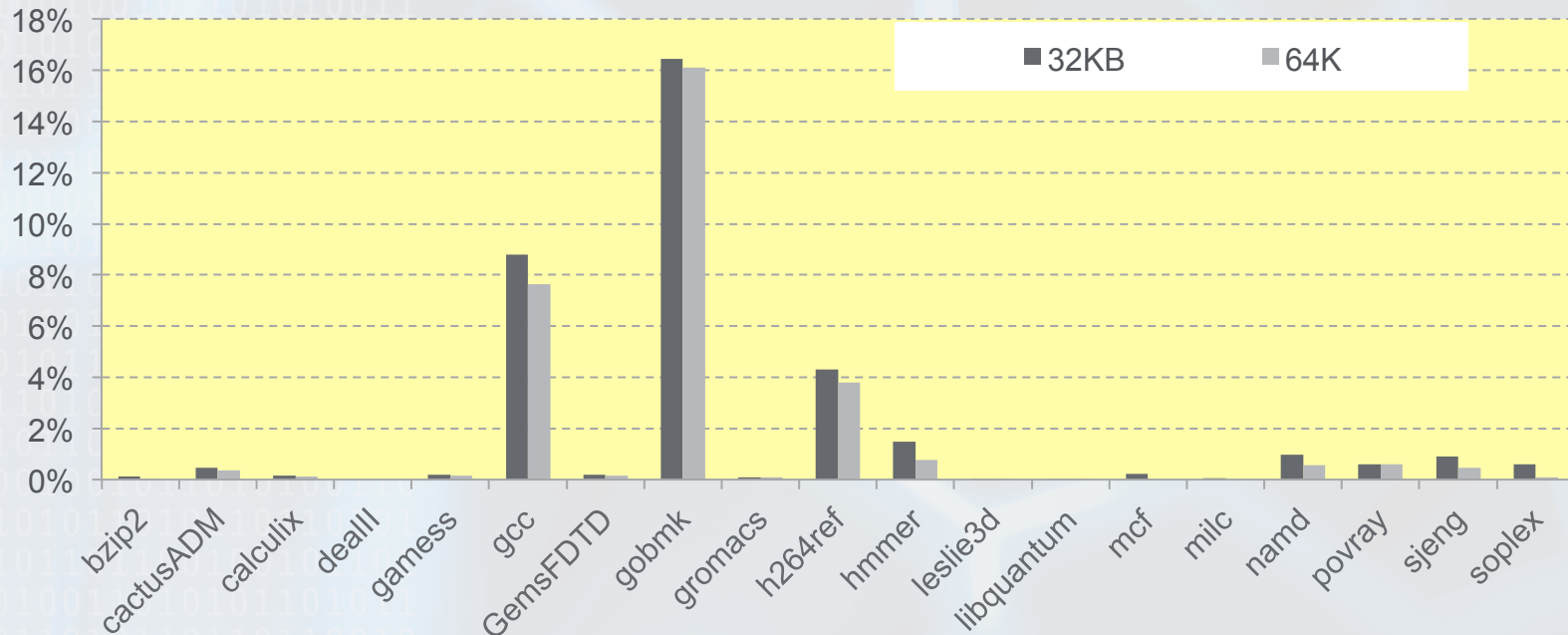
- **Go beyond the current point-in-time solutions** for dealing with specific types of attacks *that are known today*
- **Validate code integrity and and/or control flow violations at run-time: as the program executes**
- ***Provide a more universal and enduring solution, irrespective of the specific cause of the violations/attacks***
- **This solution:**
  - **Does not require binary modifications**
  - **Does not require ISA extensions**
  - **Works with out-of-order processors**
  - **Is scalable to any binary size**
  - **Has a low execution overhead**

# Approach

- **Authenticate crypto-hash signature of basic blocks and control flow path between consecutive basic blocks as program executes**
- **Store statically-derived reference information in an encrypted form in RAM**
- **Use a signature cache and overlap authentication and normal pipeline activity to reduce performance penalty**
- **Delay memory updates from a BB till its execution is authenticated**



# Performance Overhead: Two SC Sizes

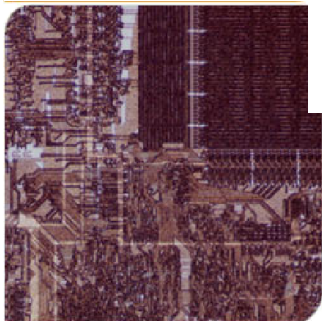
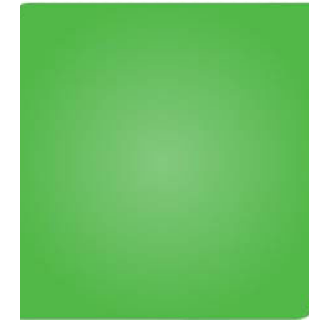
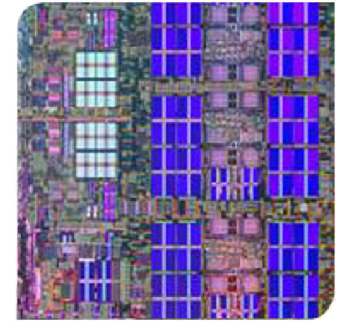


IPC(Instruction per Cycle) overhead in % of the benchmarks  
(Average of 1.87%)

\*REV was evaluated through simulation using the cycle-accurate MARSS full system microarchitectural simulator for X86-64 ISA

*next paper*

**SAVAT:**  
A Practical Methodology  
for  
Measuring  
the Side-Channel  
Signal Available to the  
Attacker  
for  
Instruction-Level Events



Rob Callan, Alenka Zajic, Milos Prvulovic

Georgia  
Tech

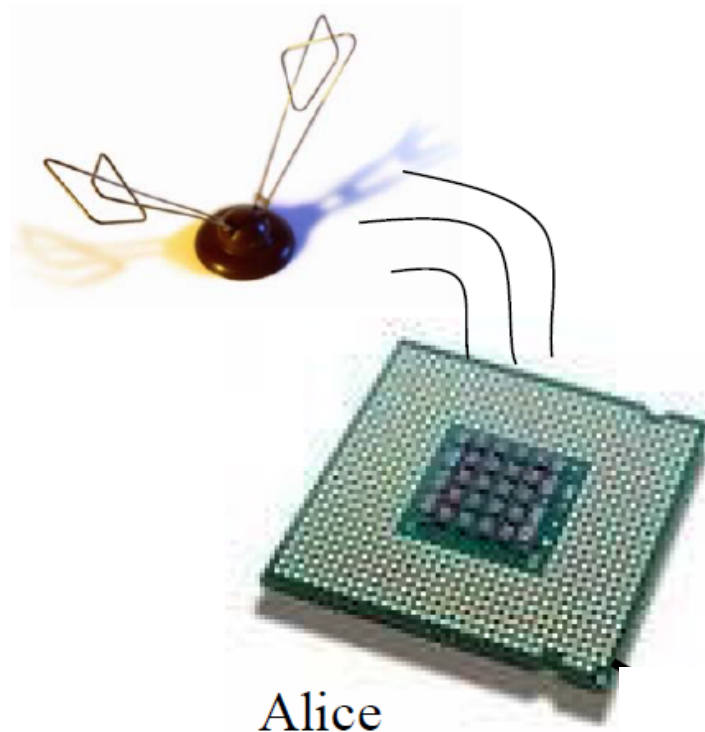


comparch

# Quantifying Side Channel Vulnerability



Eve  
EM emanations





# SAVAT

- Side-Channel **S**ignal **A**vailable to **A**ttackers
  - Entire-program
  - Circuit-Level
  - **Instruction-Level**
    - Useful for both HW and SW improvements





# SAVAT Results

$7.9 \times 10^{-21}$  joules per instruction

	LDM	STM	LDL2	STL2	LDL1	STL1	NOI	ADD	SUB	MUL	DIV
LDM	1.8	2.4	7.9	11.4	4.6	4.4	4.3	4.2	4.4	4.2	5.1
STM	2.3	2.4	8.8	11.8	4.3	4.2	3.8	3.9	3.9	4.3	4.2
LDL2	7.7	7.7	0.6	0.8	3.9	3.5	4.3	3.6	4.8	3.8	6.2
STL2	11.5	10.6	0.8	0.7	5.1	6.1	6.1	6.1	6.1	6.2	10.1
LDL1	4.4	4.2	3.3	5.8	0.7	0.6	0.7	0.7	0.7	0.7	1.3
STL1	4.5	4.2	3.8	4.9	0.7	0.6	0.7	0.6	0.6	0.6	1.2
NOI	4.1	3.8	4.1	6.4	0.7	0.7	0.6	0.6	0.7	0.6	1.0
ADD	4.2	4.1	4.1	7.0	0.7	0.7	0.6	0.7	0.6	0.6	1.0
SUB	4.4	4.0	3.8	7.3	0.7	0.6	0.7	0.6	0.6	0.6	1.1
MUL	4.4	3.9	3.7	5.7	0.7	0.7	0.6	0.6	0.6	0.6	1.1
DIV	5.0	4.6	6.9	9.3	1.3	1.2	1.0	1.1	1.1	1.1	0.8

*next paper*



# RpStacks:

Fast and Accurate

Processor Design Space Exploration

Using Representative Stall-Event Stacks

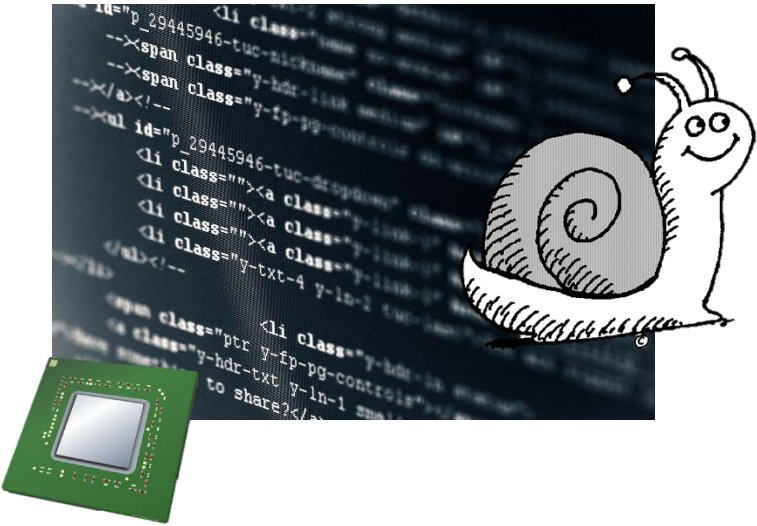
---

Jaewon Lee, Hanhwi Jang, and Jangwoo Kim

**POSTECH**

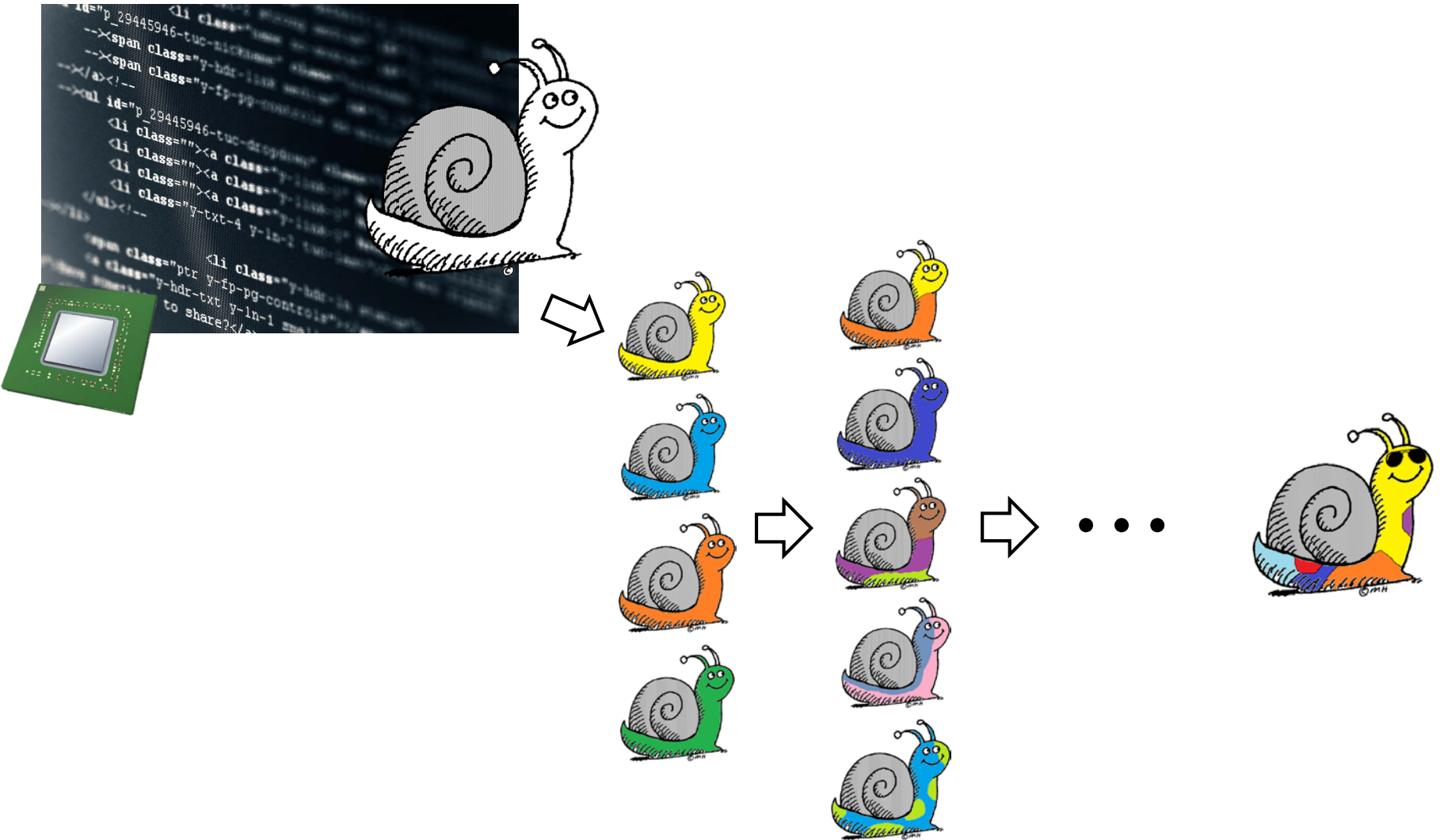
# Slow and expensive

---

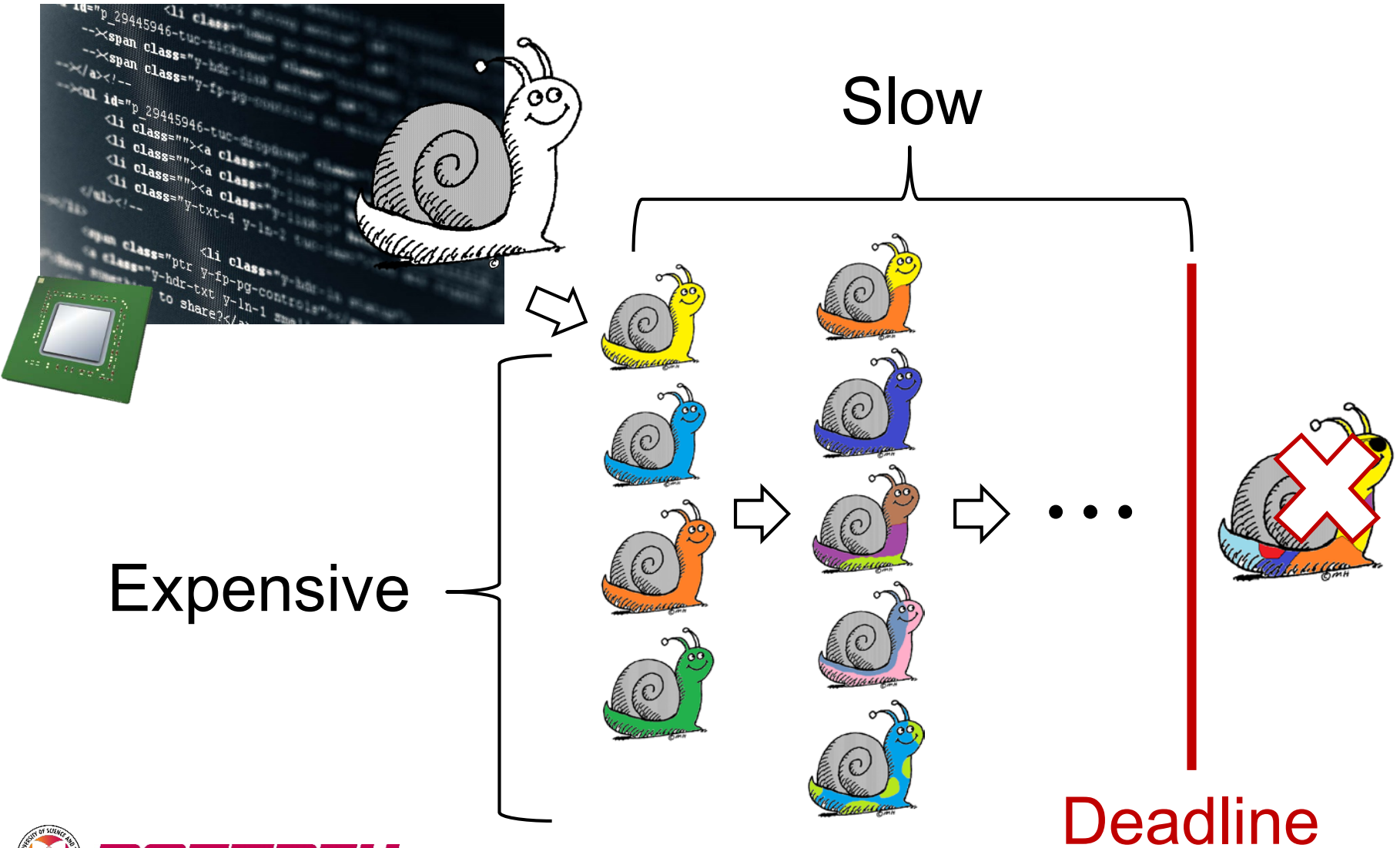


# Slow and expensive

---



# Slow and expensive

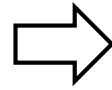


# Simultaneous representative-path analysis

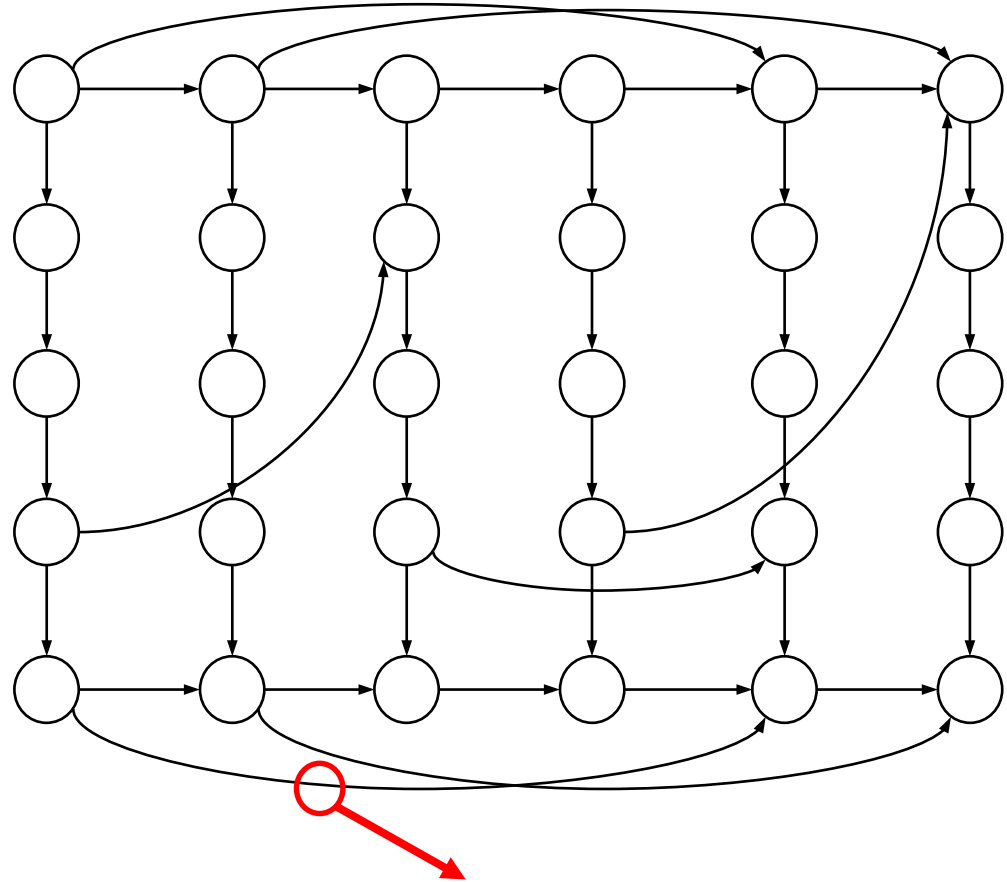
div, R2, R3, Timing: 1, 5, 6, 7, ...  
sub, R1, R0, Timing: 1, 6, 7, 7, ...  
ld, R5, 0x02, L1Miss, Timing: ...,  
beq, R7, R0, Ntaken, Timing: ...,  
add, R6, R1, Timing: 1, 5, 6, 6, ...  
...

Simulation  
Information  
(Baseline)

Instruction →



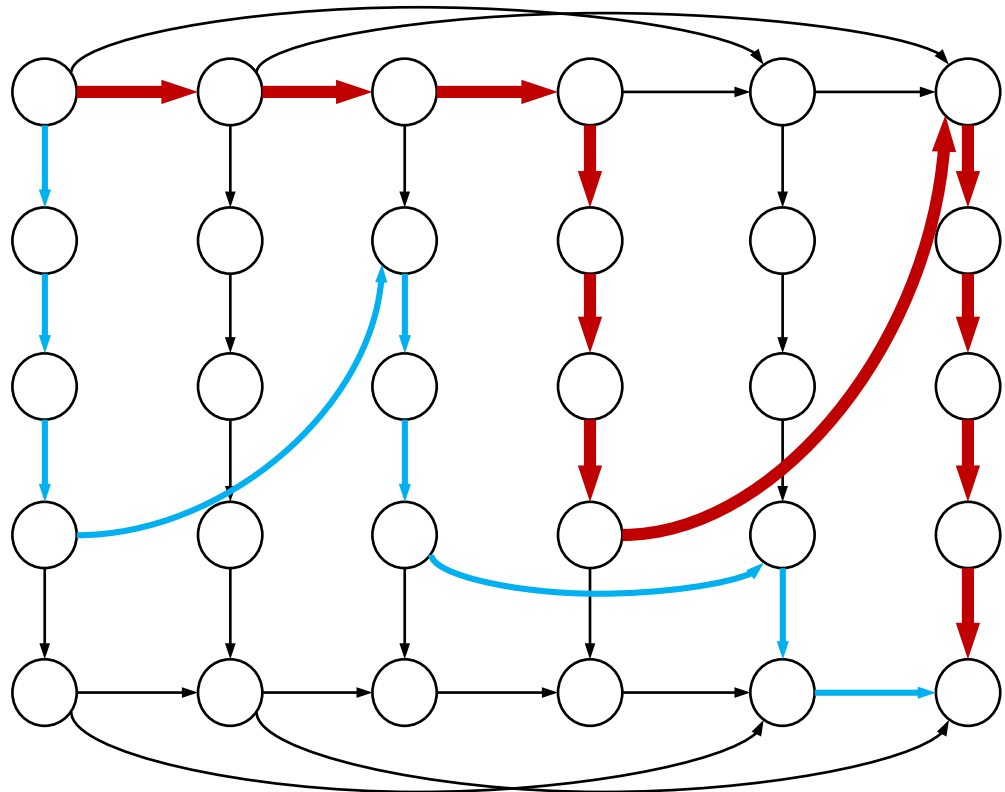
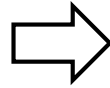
Pipeline ↓



Dependency & Stall cycles

# Simultaneous representative-path analysis

div, R2, R3, Timing: 1, 5, 6, 7, ...  
sub, R1, R0, Timing: 1, 6, 7, 7, ...  
ld, R5, 0x02, L1Miss, Timing: ...,  
beq, R7, R0, Ntaken, Timing: ...,  
add, R6, R1, Timing: 1, 5, 6, 6, ...  
...



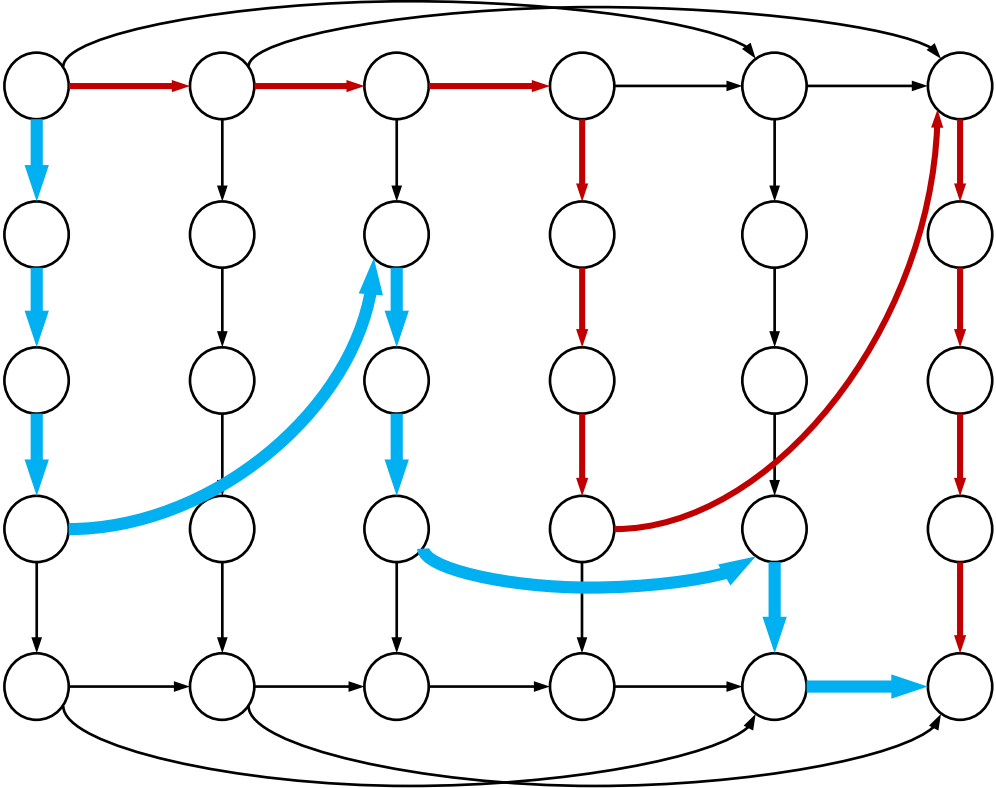
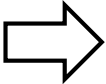
→ Cycles: 100

→ Cycles: 80



# Simultaneous representative-path analysis

```
div, R2, R3, Timing: 1, 5, 6, 7, ...
sub, R1, R0, Timing: 1, 6, 7, 7, ...
ld, R5, 0x02, L1Miss, Timing: ...,
beq, R7, R0, Ntaken, Timing: ...,
add, R6, R1, Timing: 1, 5, 6, 6, ...
...
```



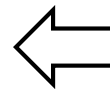
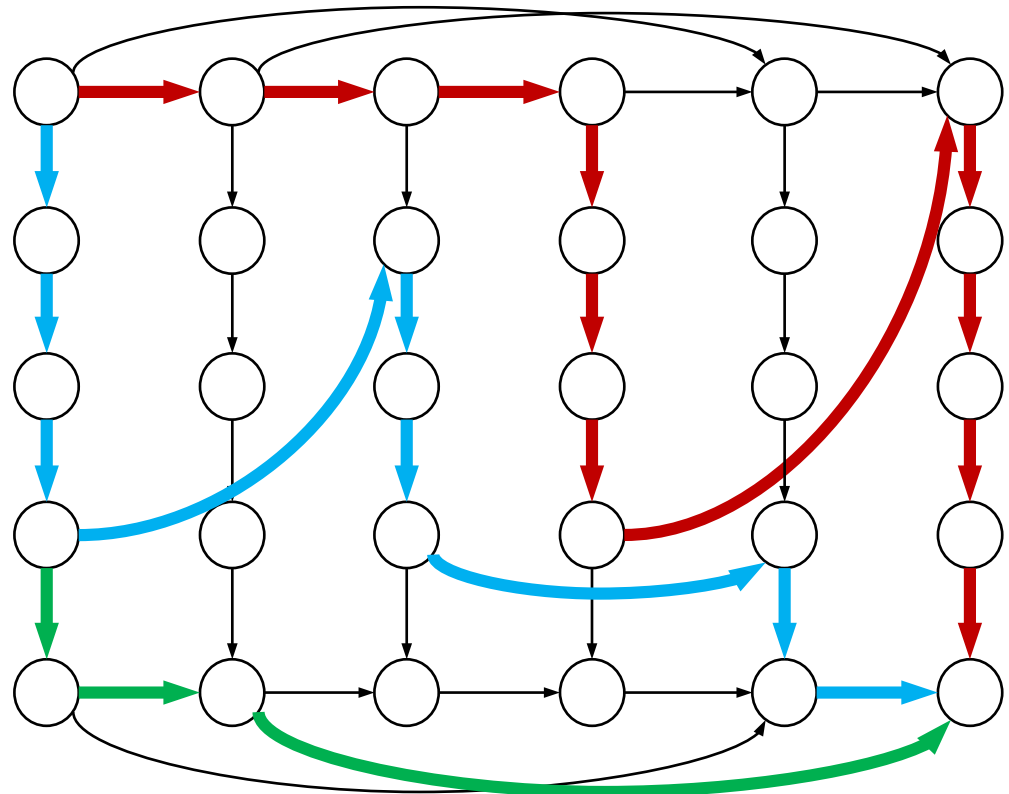
→ Cycles: 90



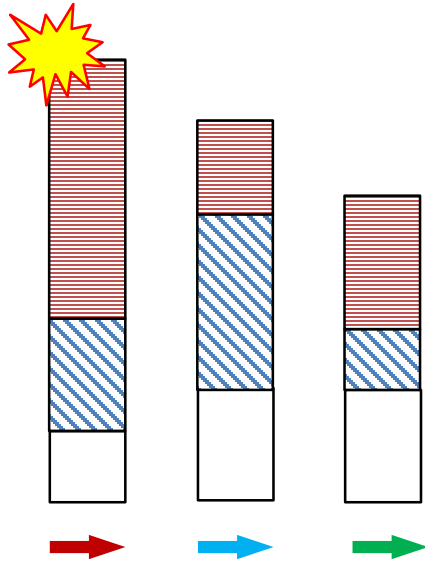
→ Cycles: 110

# Simultaneous representative-path analysis

div, R2, R3, Timing: 1, 5, 6, 7, ...  
sub, R1, R0, Timing: 1, 6, 7, 7, ...  
ld, R5, 0x02, L1Miss, Timing: ...,  
beq, R7, R0, Ntaken, Timing: ...,  
add, R6, R1, Timing: 1, 5, 6, 6, ...  
...



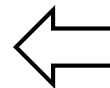
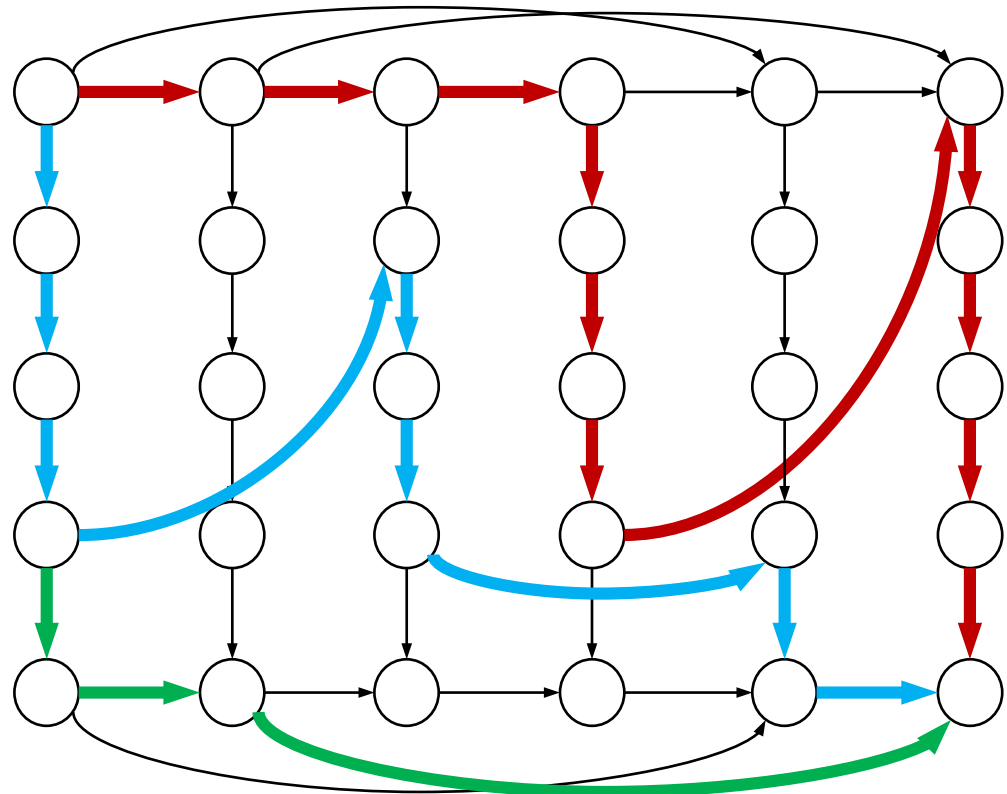
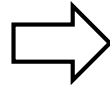
□ Event1    ▨ Event2    ▩ Event3



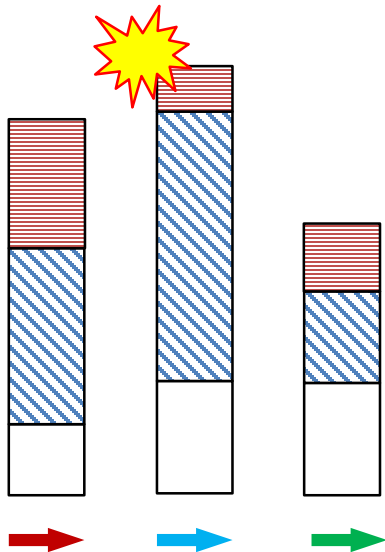
RpStacks

# Simultaneous representative-path analysis

div, R2, R3, Timing: 1, 5, 6, 7, ...  
sub, R1, R0, Timing: 1, 6, 7, 7, ...  
ld, R5, 0x02, L1Miss, Timing: ...,  
beq, R7, R0, Ntaken, Timing: ...,  
add, R6, R1, Timing: 1, 5, 6, 6, ...  
...



□ Event1   ▨ Event2   ▩ Event3



RpStacks

# Fast and accurate

---

- **Fast**

x26 speedup vs. simulator  
(1,000 design point testing)

- **Accurate**

vs. traditional simulation result analysis

Main auditorium

16<sup>th</sup> / 9:15 AM

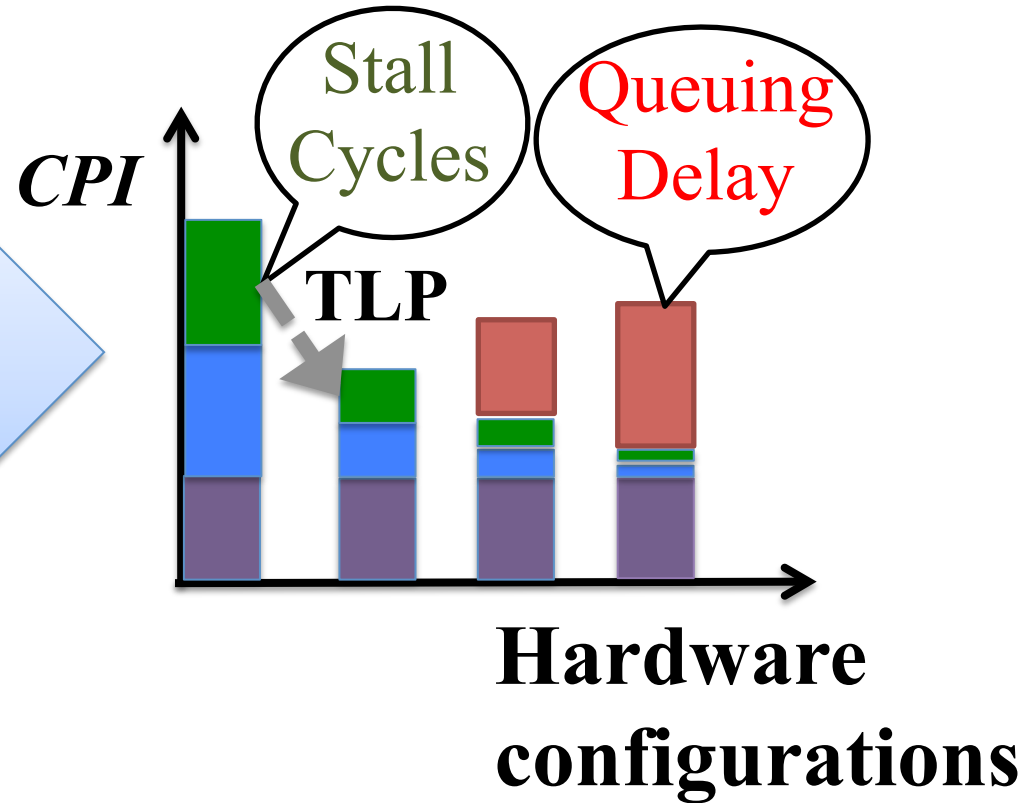
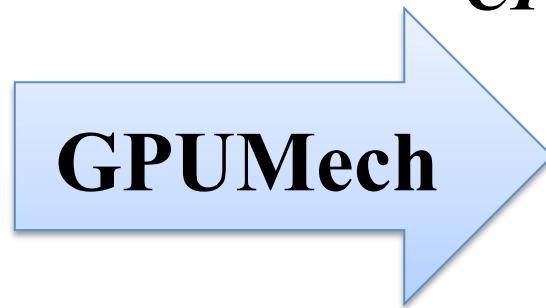
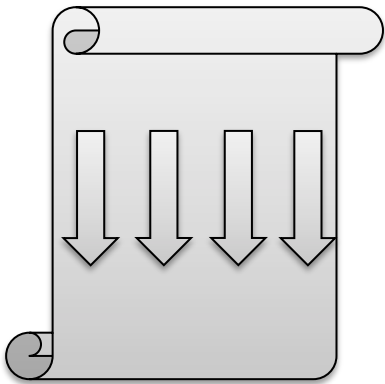


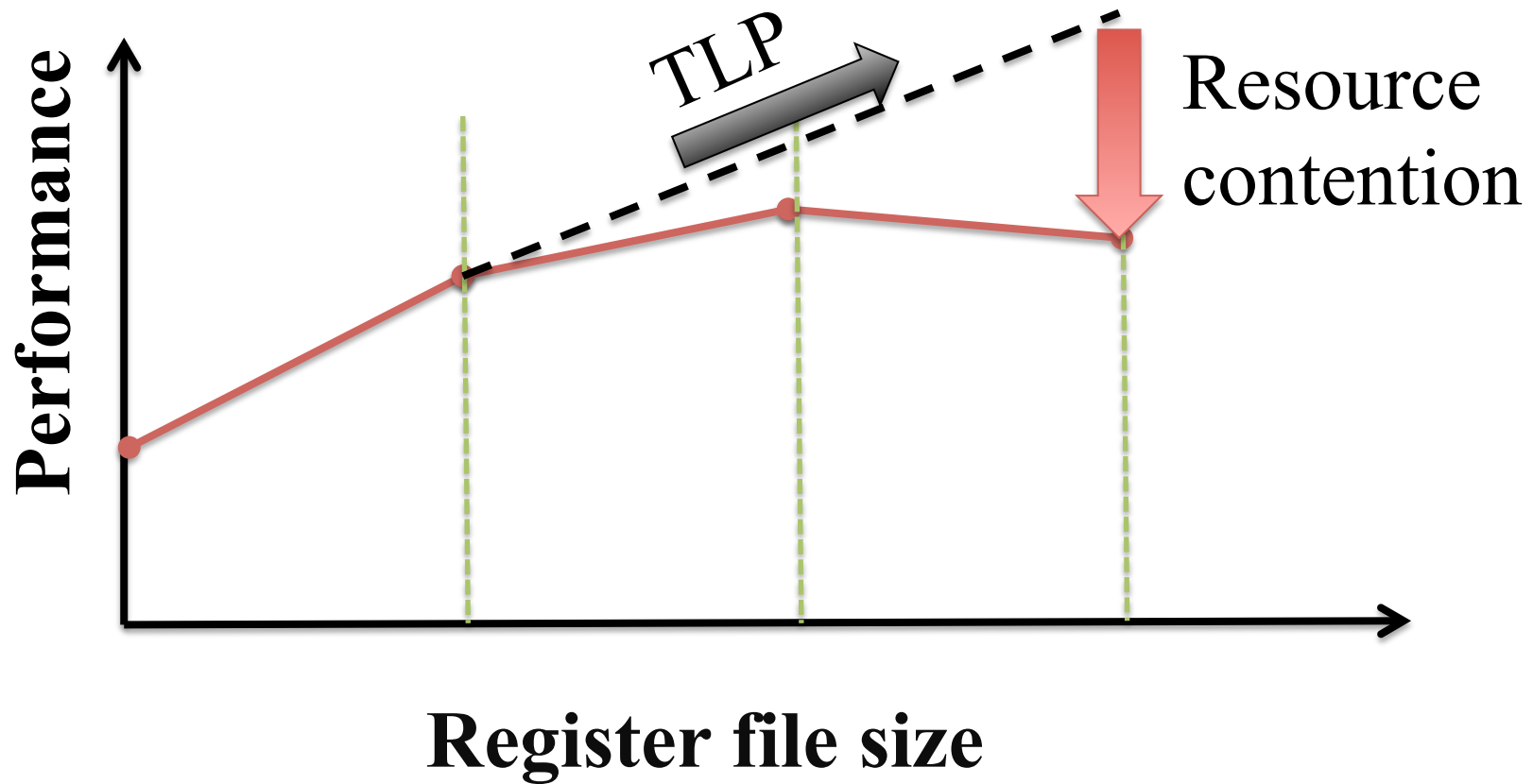
*next paper*

# GPUMech: GPU Performance Modeling Technique based on Interval Analysis

**Jen-Cheng Huang**, Joo Hwan Lee, Hyesoon Kim, Hsien-Hsin S. Lee  
*Georgia Institute of Technology*

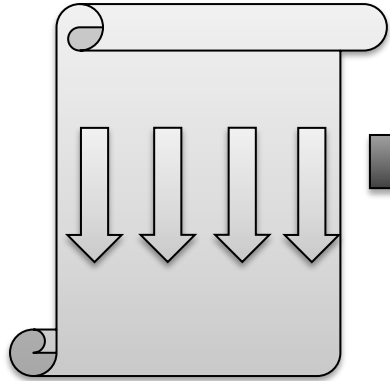
## GPU Kernel





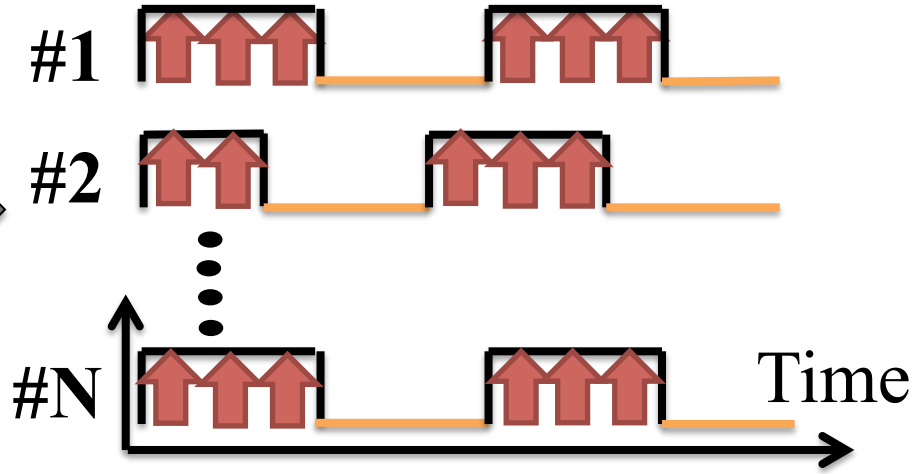
Can we **quickly** find **bottlenecks** of hardware configurations without time-consuming detailed timing simulation?

**GPU Kernel**



**Functional  
Simulation**

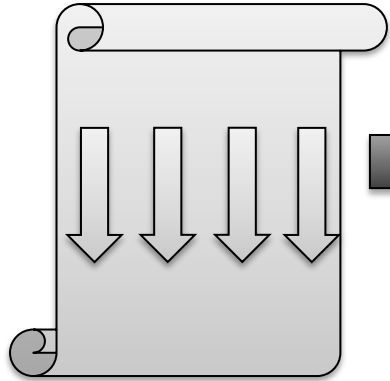
**Warp ID**



**Warp Profiling**

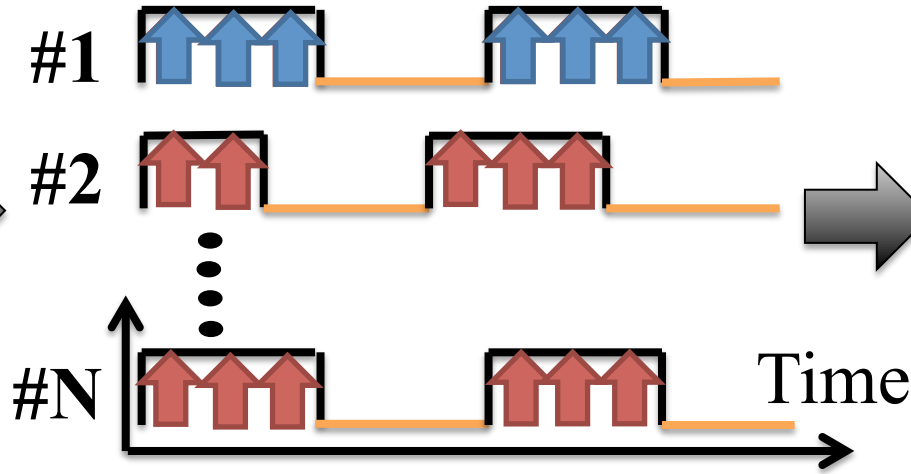


# GPU Kernel

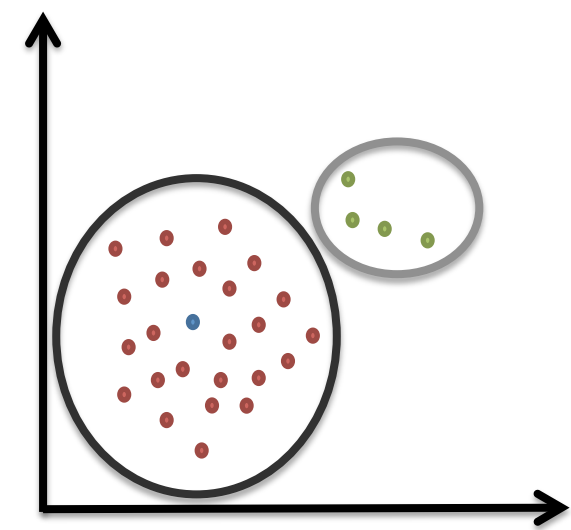


## Functional Simulation

### Warp ID

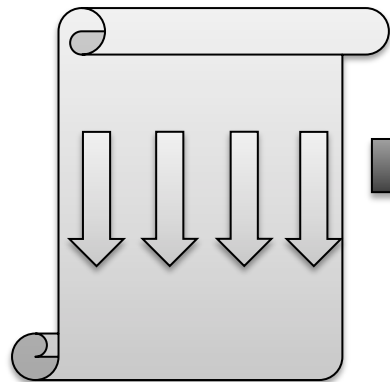


## Warp Profiling



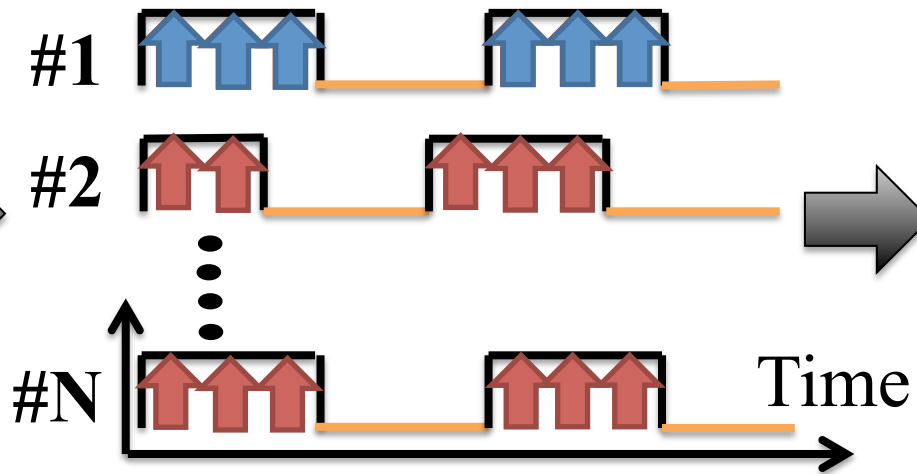
## Warp Selection

# GPU Kernel

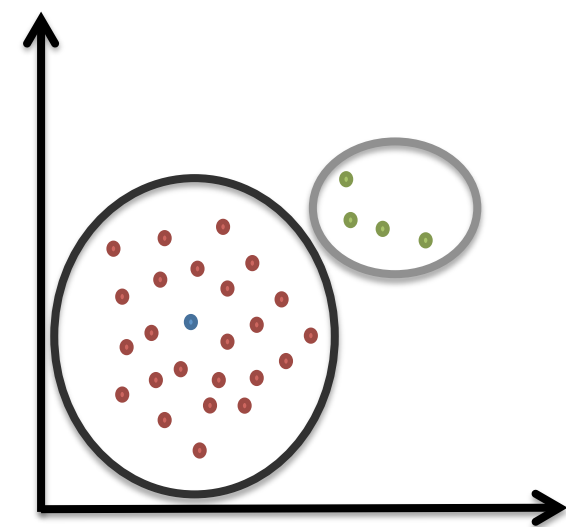


## Functional Simulation

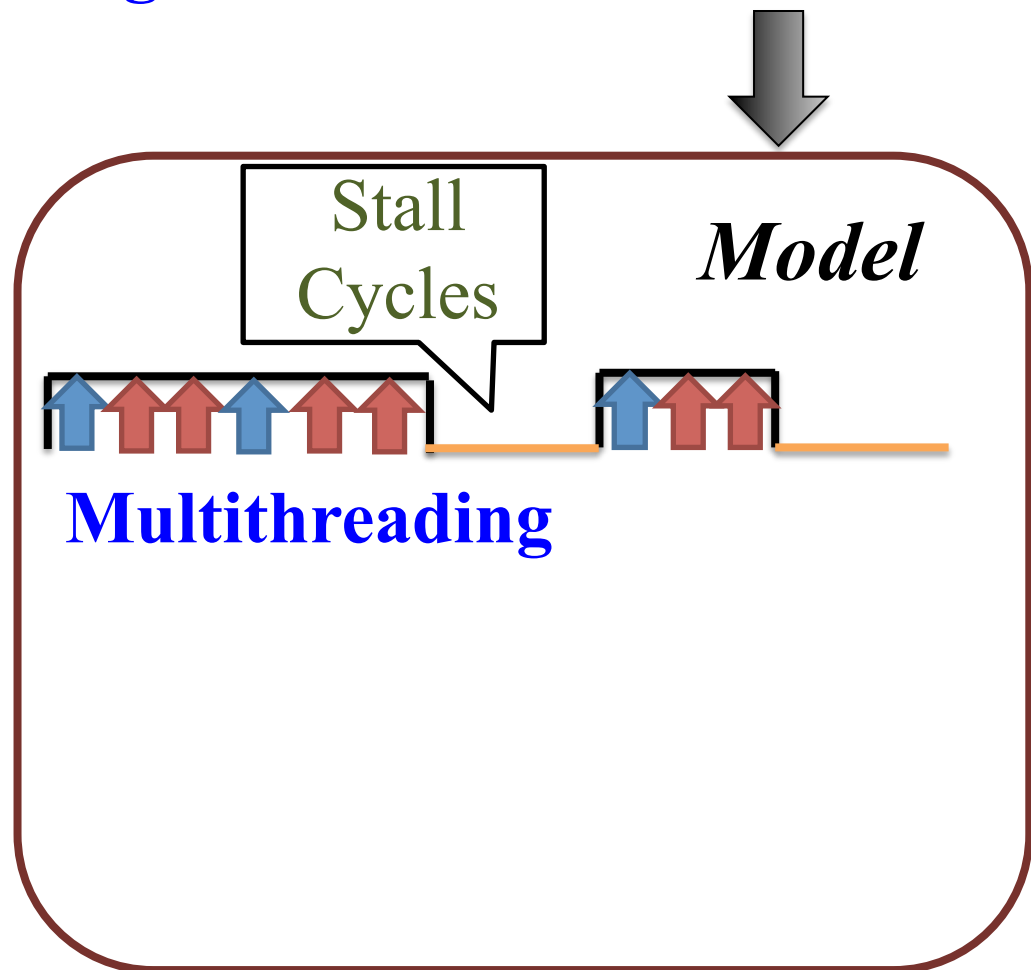
### Warp ID



## Warp Profiling

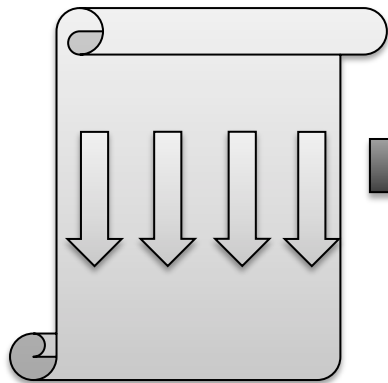


## Warp Selection



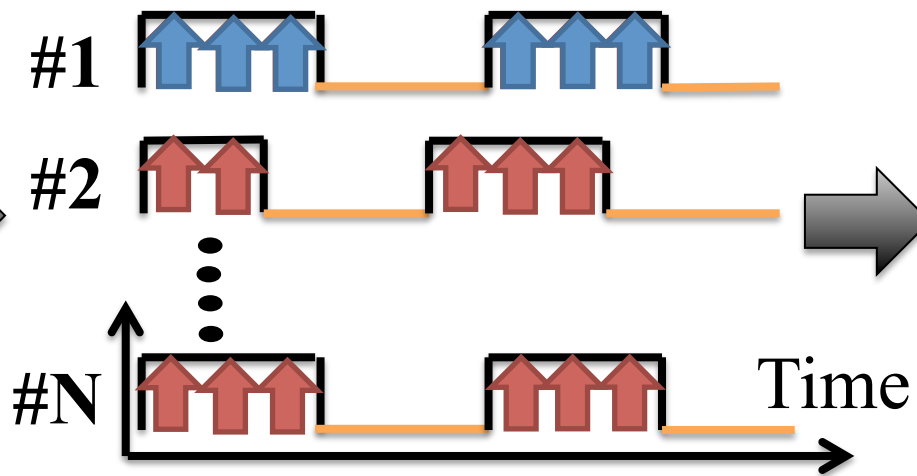
## Multithreading

# GPU Kernel

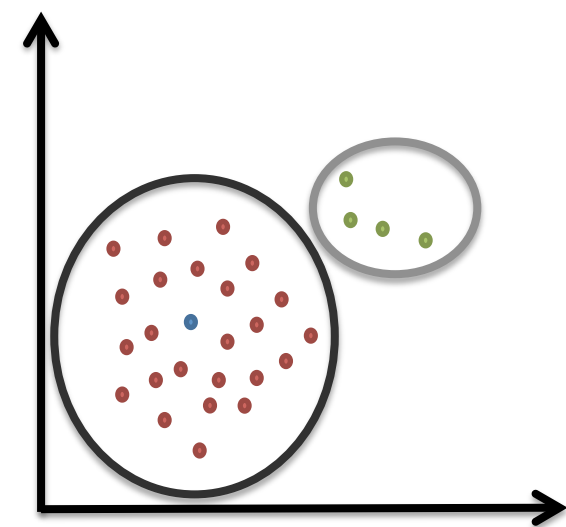


## Functional Simulation

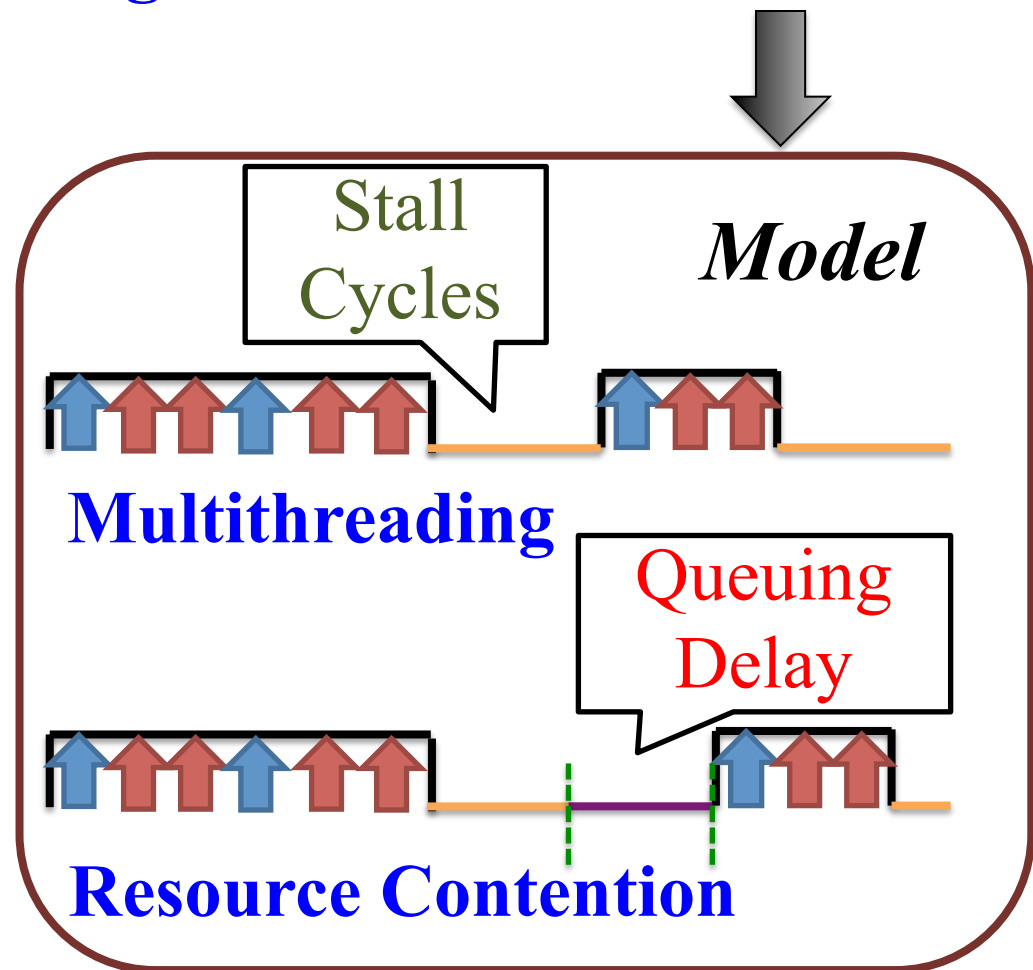
Warp ID



## Warp Profiling



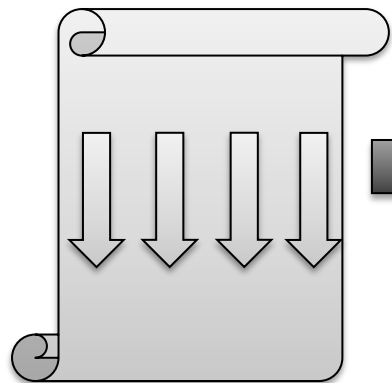
## Warp Selection



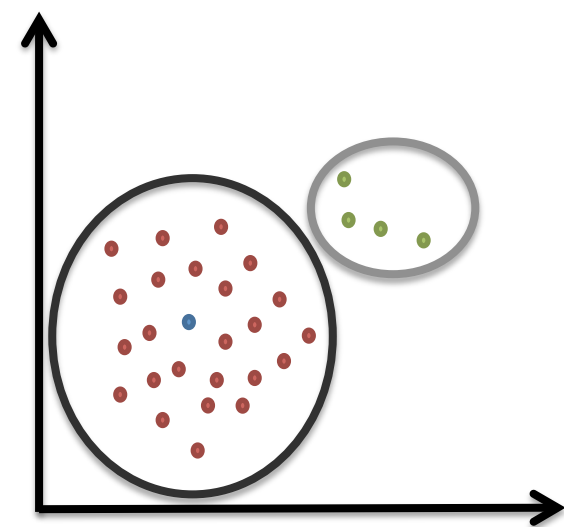
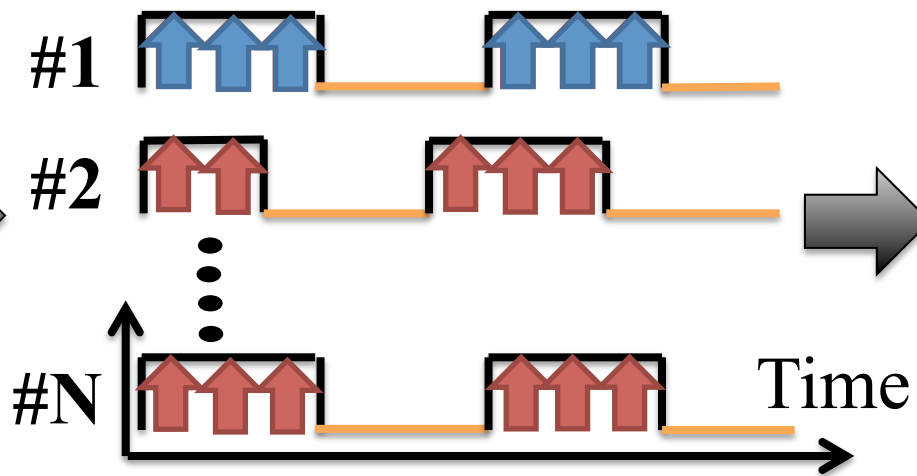
## Multithreading

## Resource Contention

# GPU Kernel



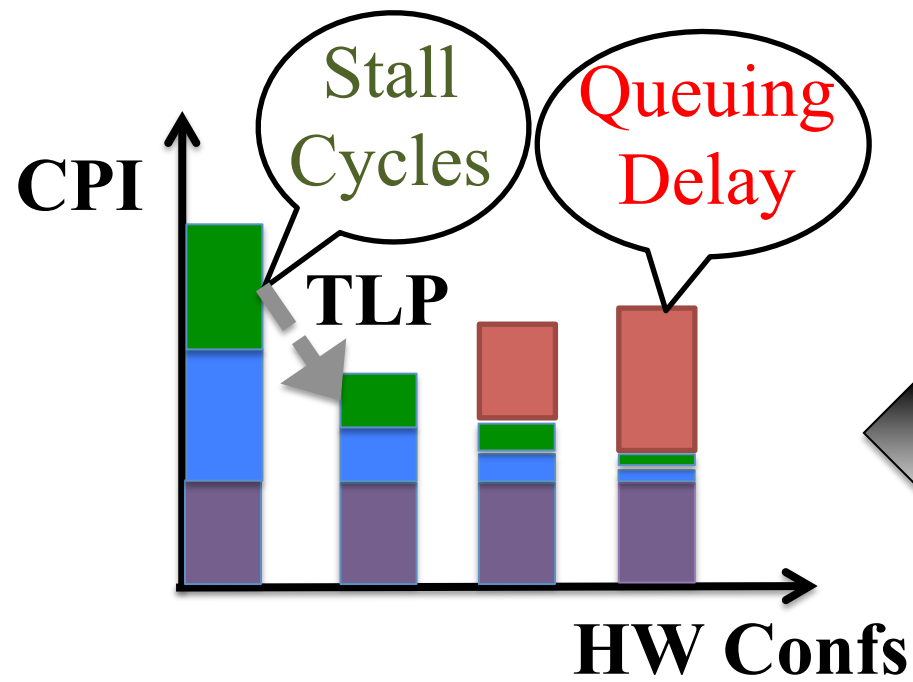
## Warp ID



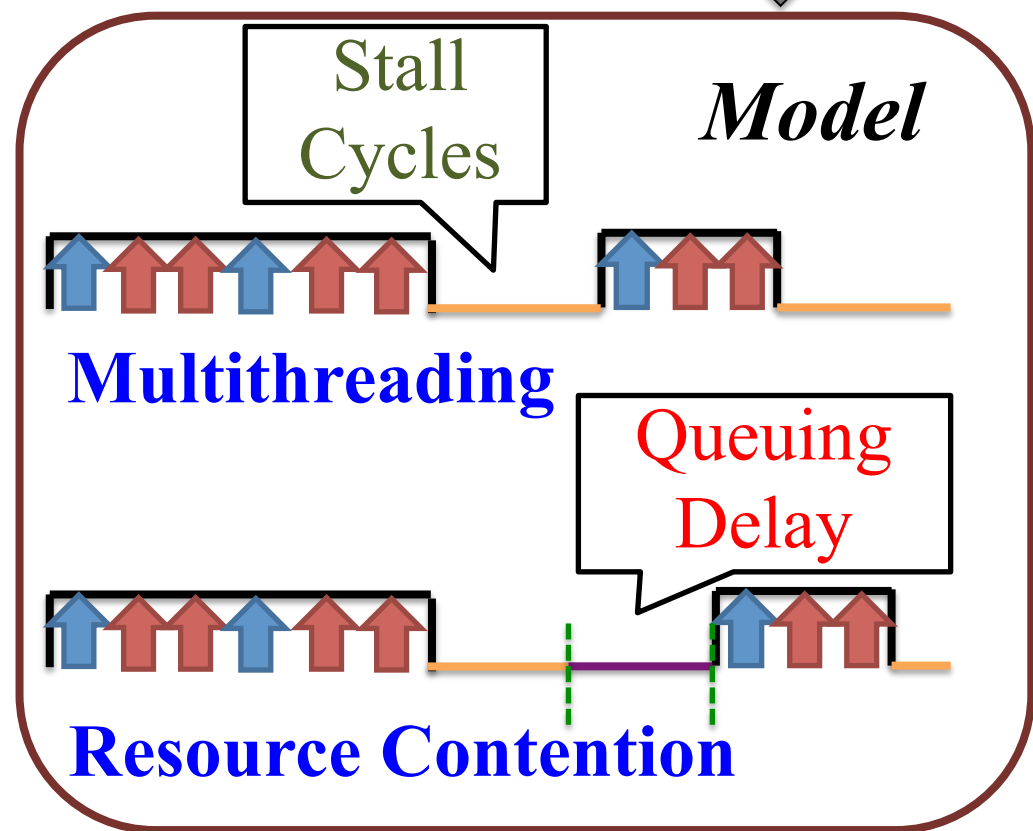
## Functional Simulation

## Warp Profiling

## Warp Selection



## Bottleneck Visualization



## Resource Contention

## Approach: **GPUMech**

- Use analytical modeling to model performance
- Use functional simulation to identify stall events
- Visualize **performance bottlenecks** using **CPI stack**
- **97x Speed Up** over detailed timing simulation  
(Error: 13.2%)

*Session 3A: Methodology, Modeling and Tools*

*Tue 09:15 – 10:30*

*next paper*

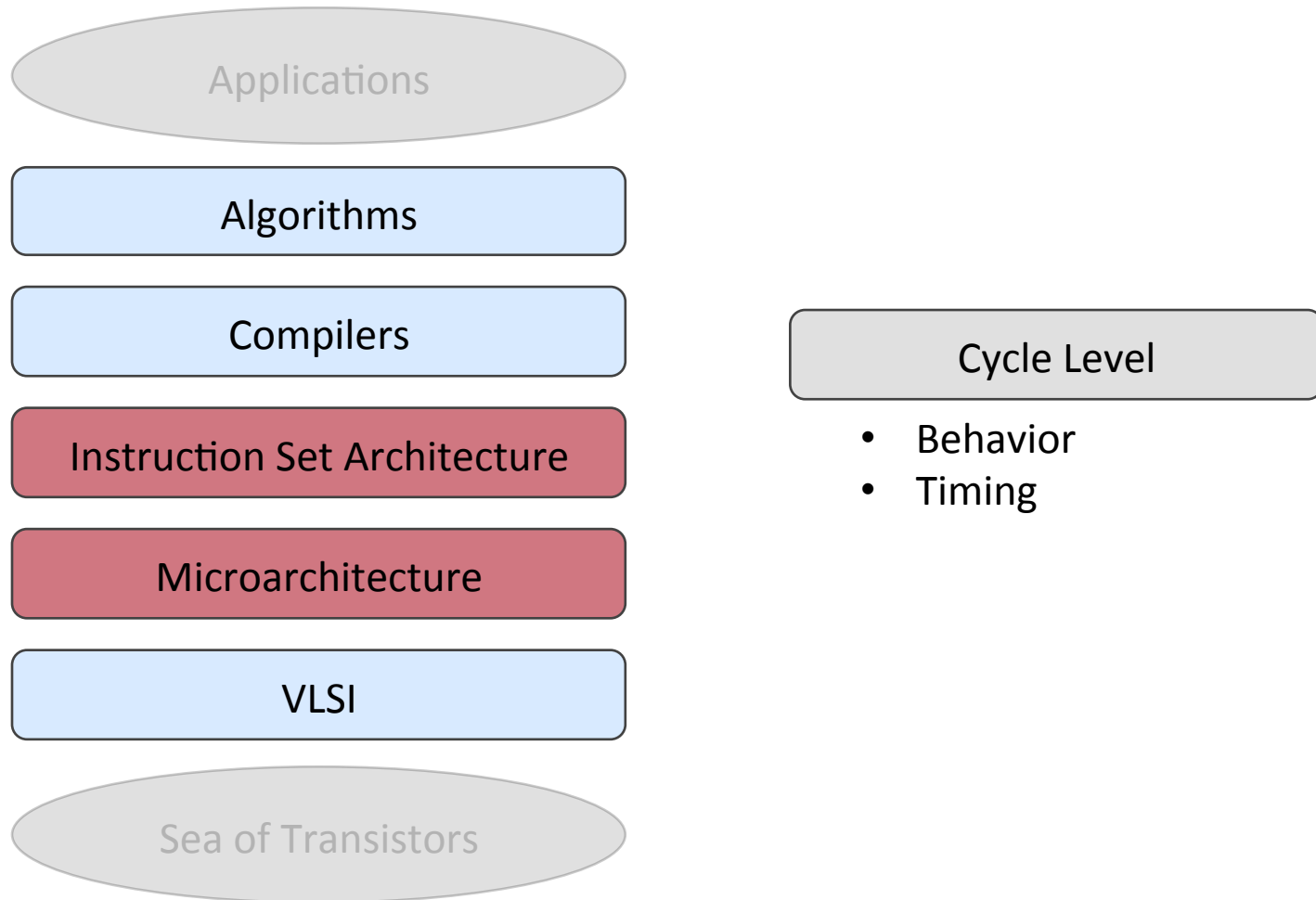
# **PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research**

Derek Lockhart, Gary Zibrat, and Christopher Batten



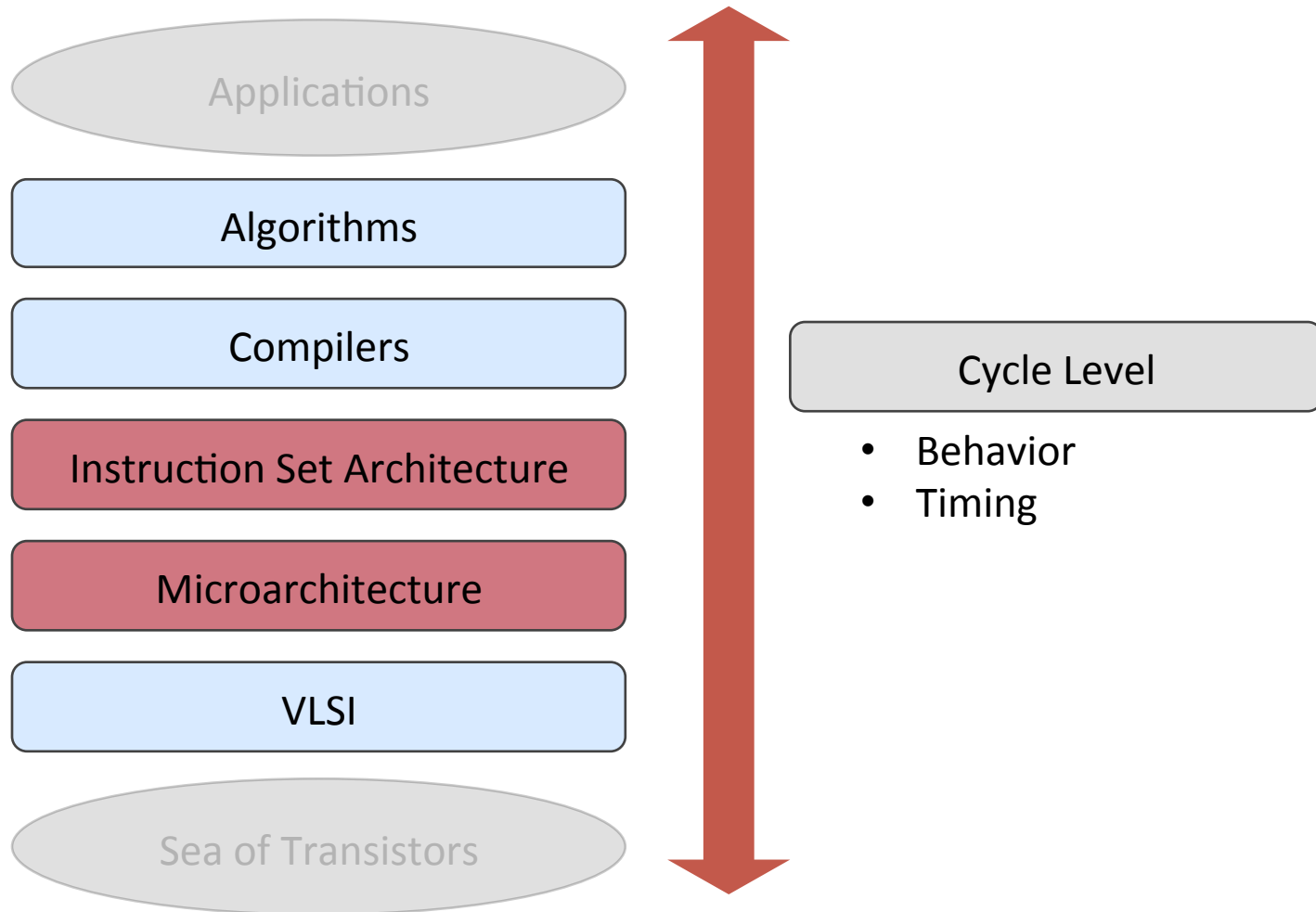
Cornell University  
Computer Systems Laboratory

# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

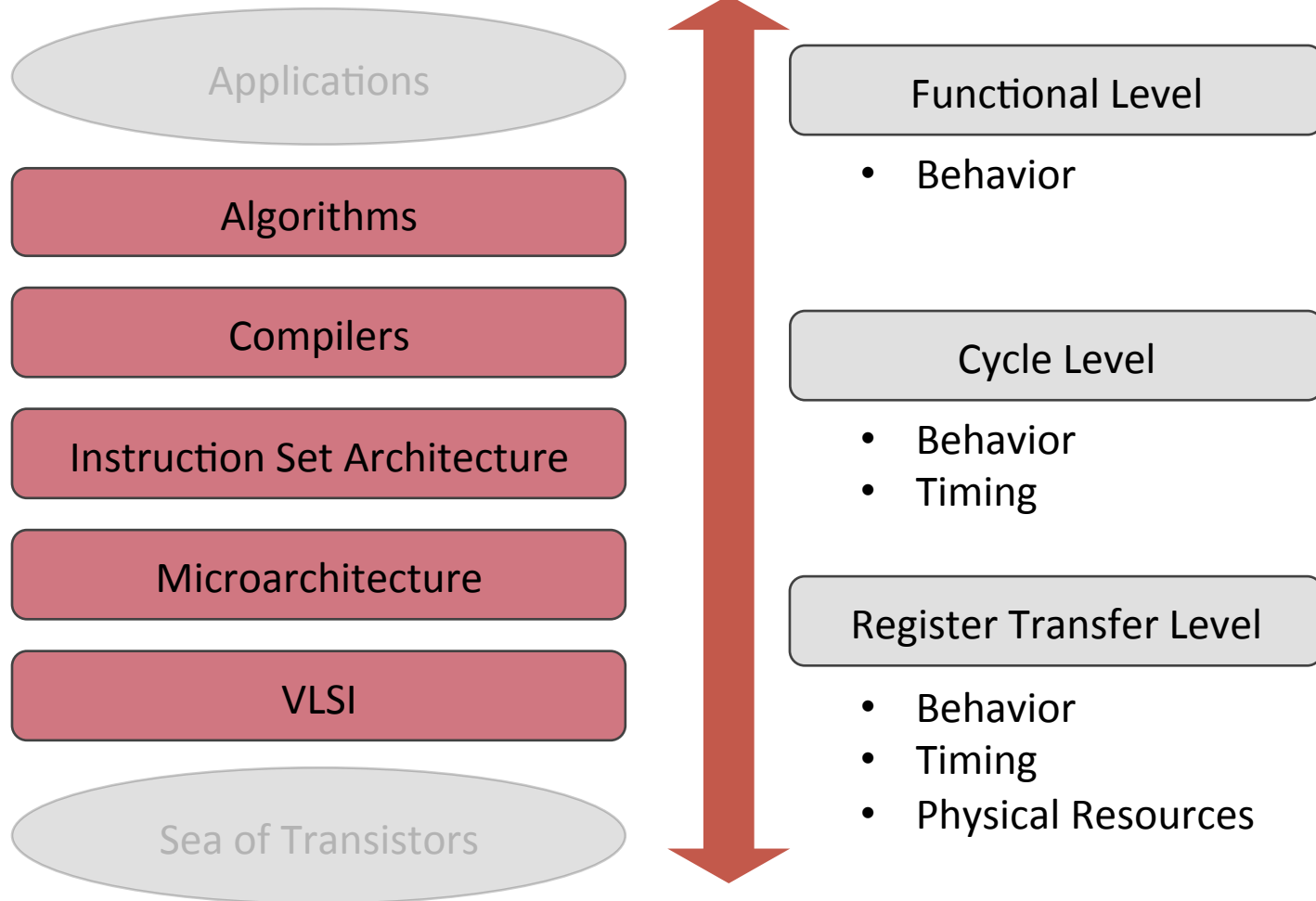




# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

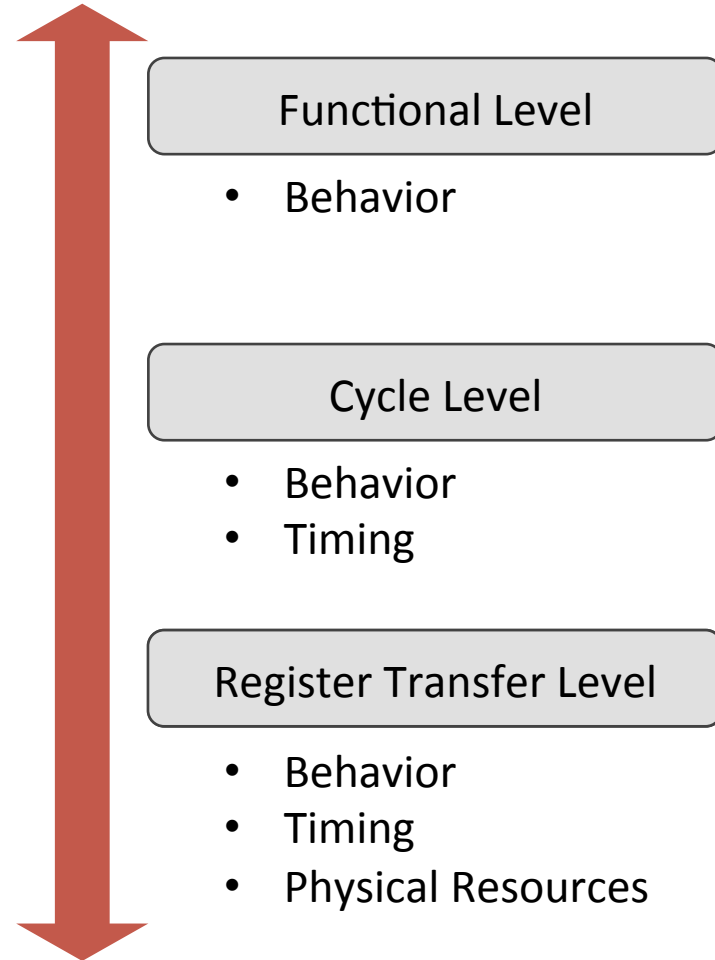


# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research



# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

## Modeling Towards Layout



# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

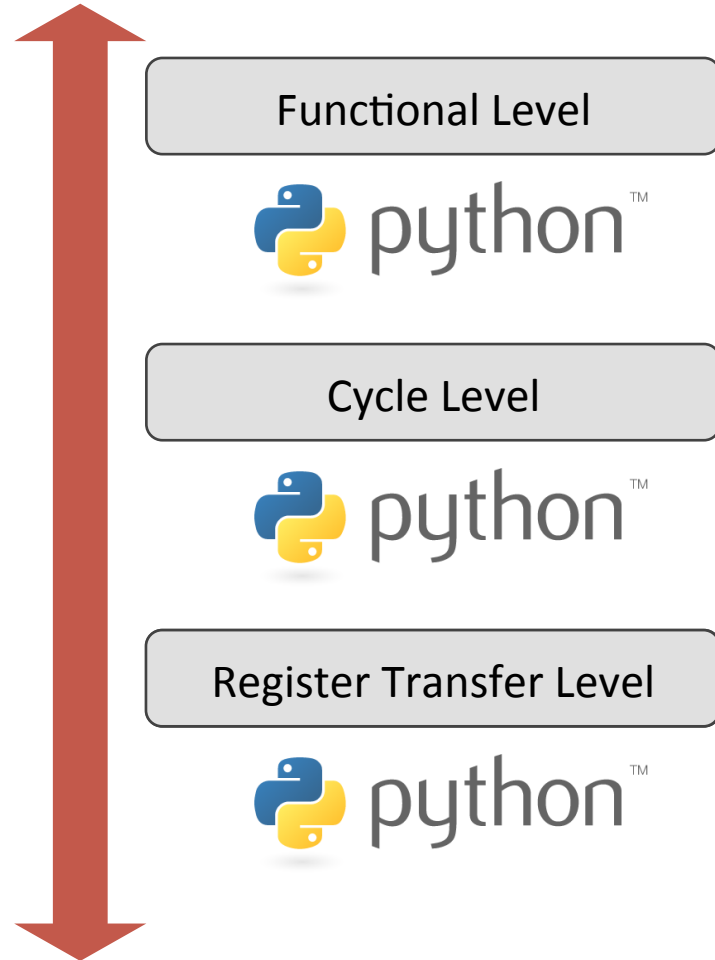
**Modeling Towards Layout**

Traditional



# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

Modeling Towards Layout

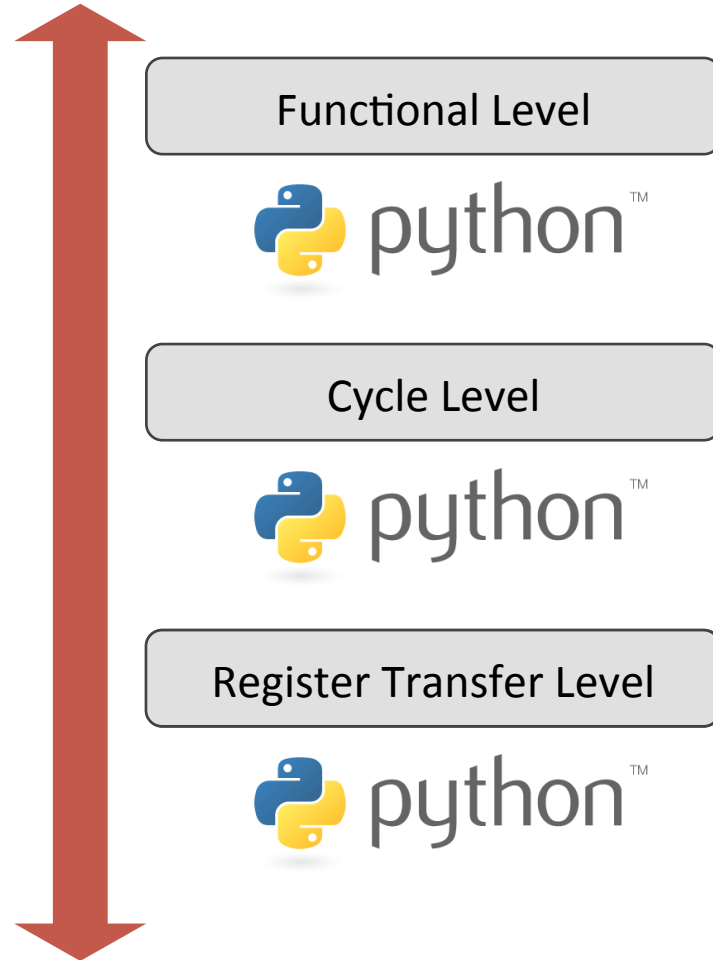


# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

## Modeling Towards Layout



 Programmer Productivity



# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

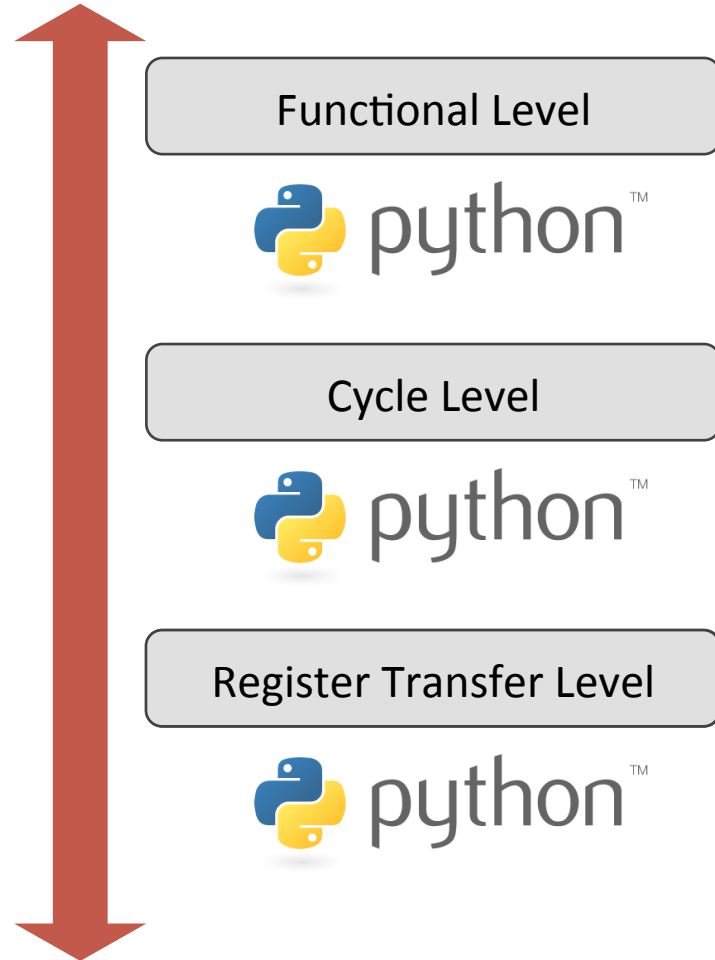
## Modeling Towards Layout



 Programmer Productivity

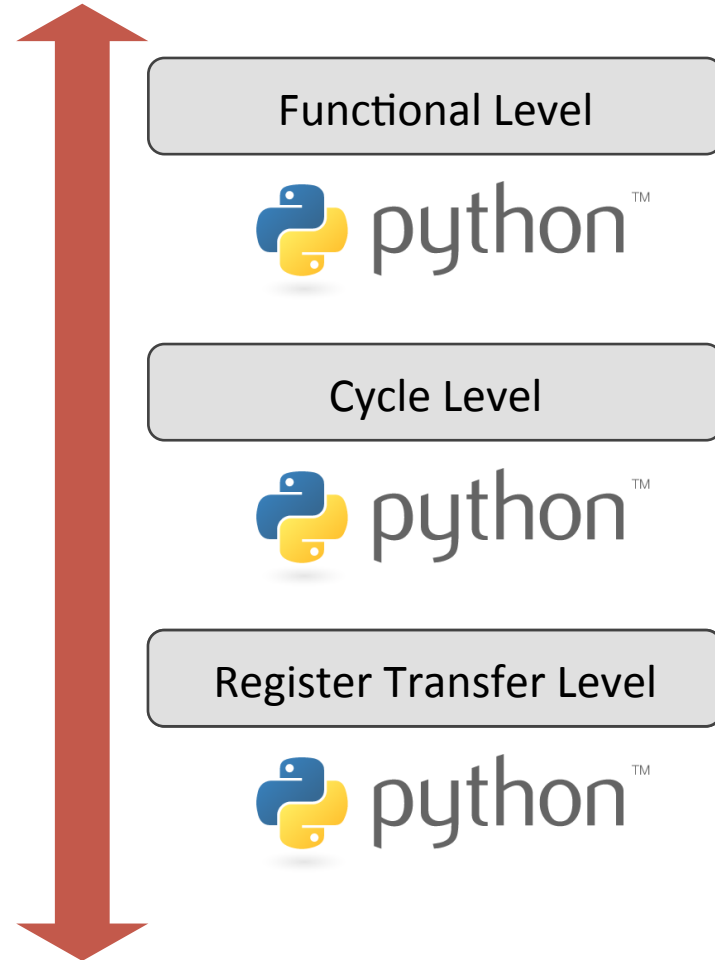


 Simulation Performance



# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

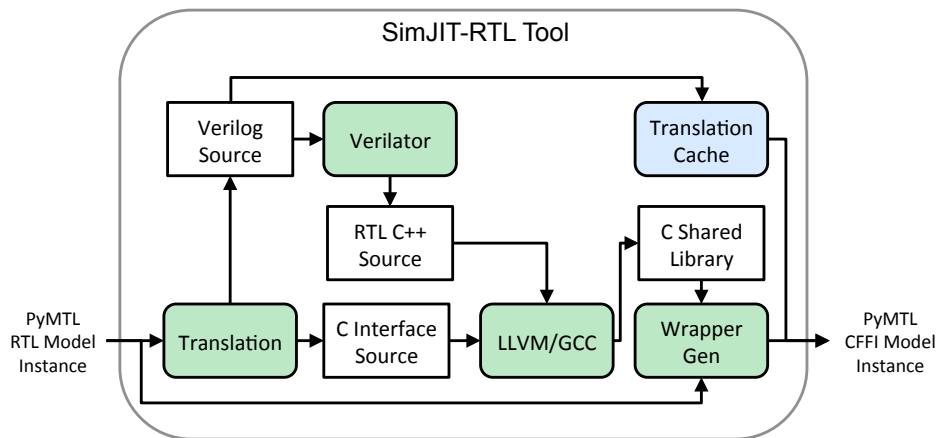
**Solution:**  
SimJIT





# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research

## Solution: SimJIT



Functional Level



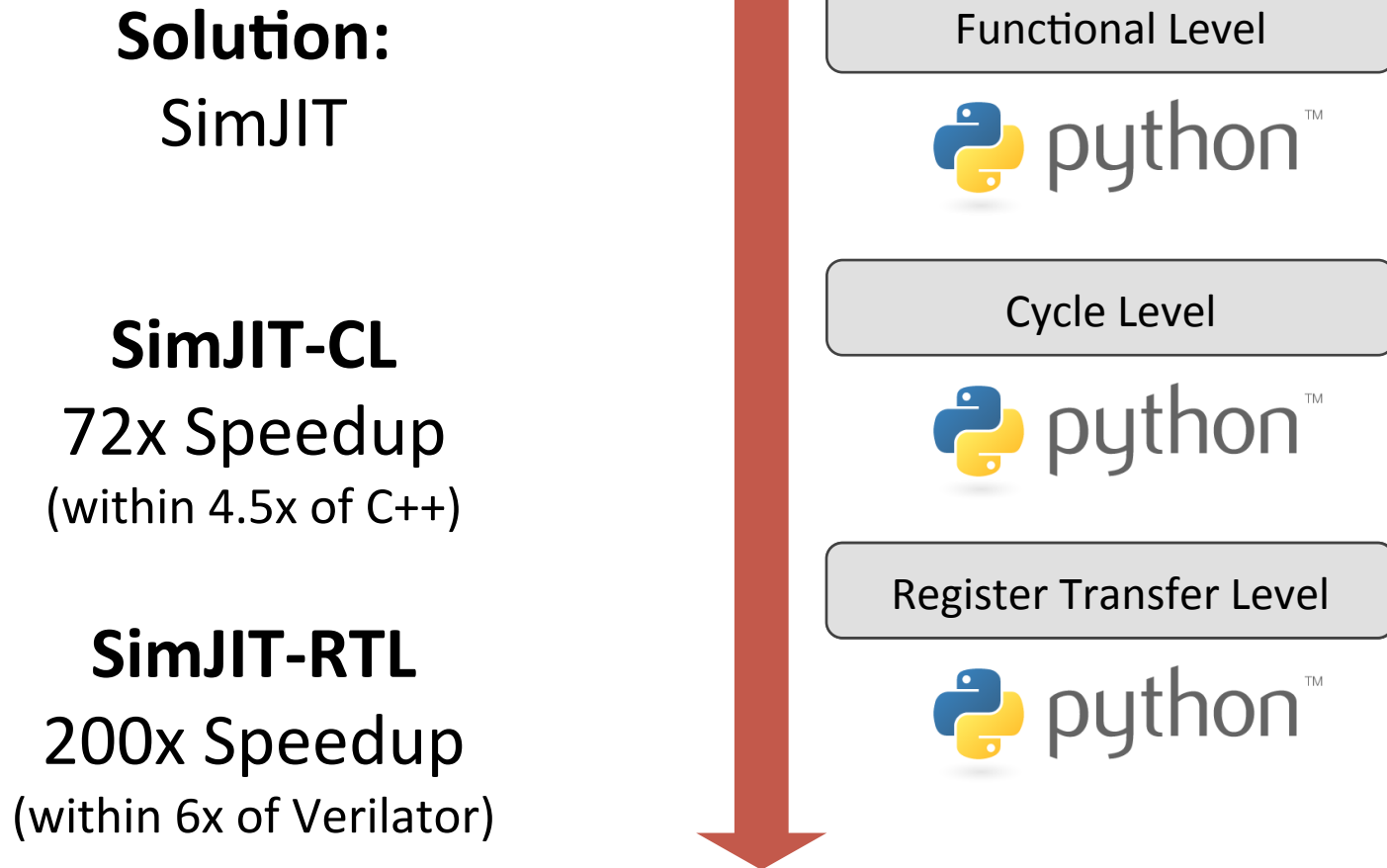
Cycle Level



Register Transfer Level



# PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research



*next paper*



# ***CALCULATING ARCHITECTURAL VULNERABILITY FACTORS FOR SPATIAL MULTI-BIT TRANSIENT FAULTS***

Mark Wilkening<sup>1</sup>, Vilas Sridharan<sup>2</sup>, Si Li<sup>3</sup>, Fritz Previlon<sup>1</sup>, Sudhanva Gurumurthi<sup>4</sup> and David R. Kaeli<sup>1</sup>

<sup>1</sup>ECE Department, Northeastern University, Boston, MA, USA

<sup>2</sup>RAS Architecture, Advanced Micro Devices, Inc., Boxborough, MA, USA

<sup>3</sup>ECE Department, Georgia Institute of Technology, Atlanta, GA, USA

<sup>4</sup>AMD Research, Advanced Micro Devices, Inc., Boxborough, MA, USA

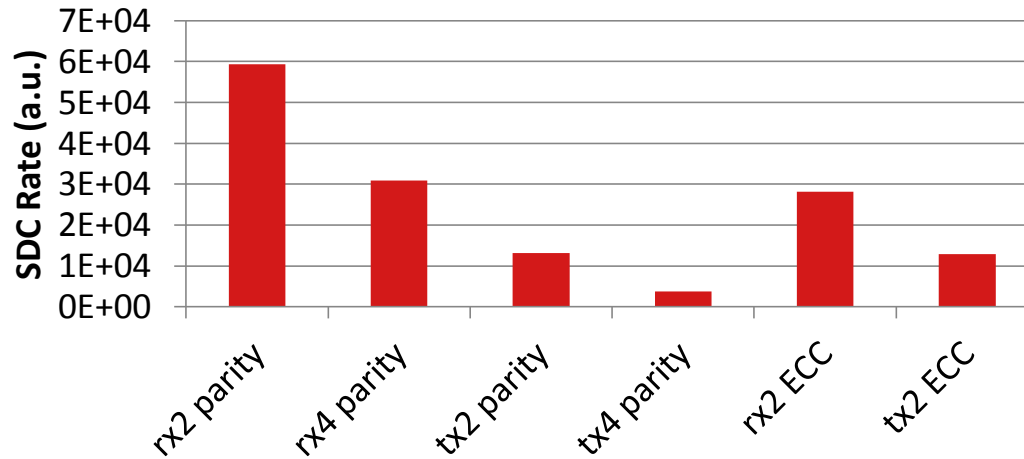


- **Particle-induced transient faults in SRAM are the dominant contributor to microprocessor faults [Baumann 2005]**
  - High energy particles deposit charge in silicon
  - This can invert the state of logic devices and cause one or more bit flips
  
- **Multi-bit transient faults have become more important as technology scales**
  - Number and size of multi-bit faults is increasing [Ibe 2010]
  - Trend will continue despite FinFETs [Seifert 2012]
  
- **Methods to characterize the impact of multi-bit faults are lacking**



# WHAT IS THIS PAPER ABOUT?

- We introduce architectural vulnerability factors for multi-bit transient faults (MB-AVFs)
- We measure MB-AVFs for detected, uncorrected errors (DUE MB-AVF)
- We approximate MB-AVFs for silent data corruption (SDC MB-AVF)
- We show how MB-AVFs can be used to make design trade-offs



*next paper*

# Using ECC Feedback to Guide Voltage Speculation in Low-Voltage Processors

Anys Bacha and Radu Teodorescu  
Department of Computer Science and Engineering

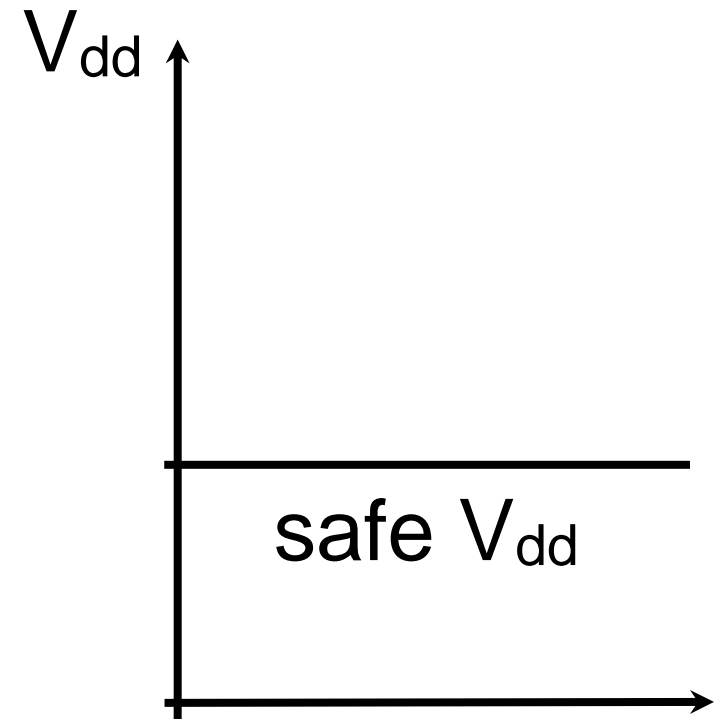
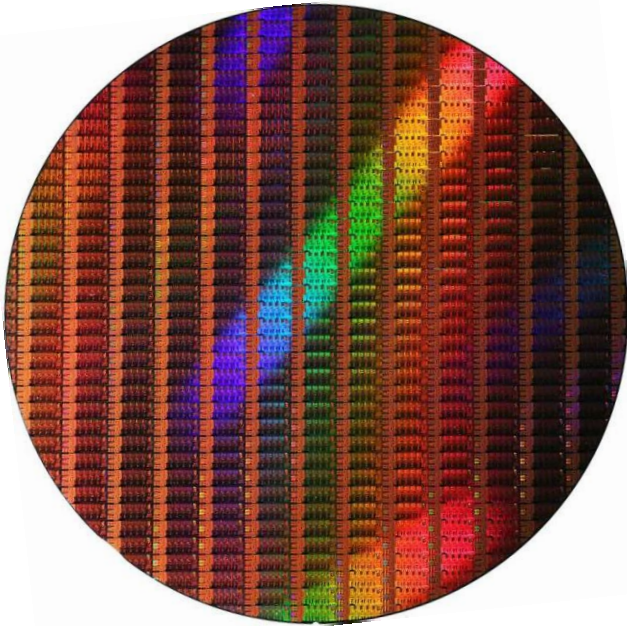


THE OHIO STATE UNIVERSITY



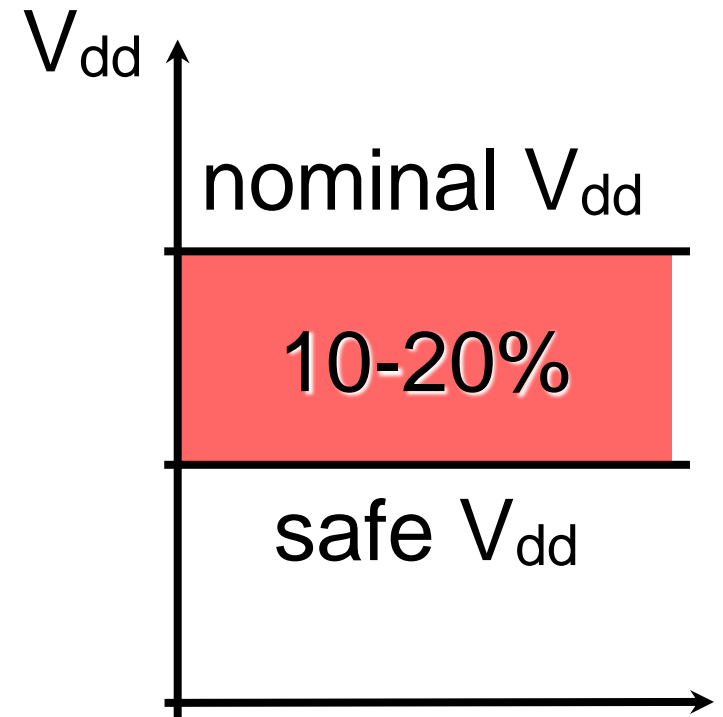
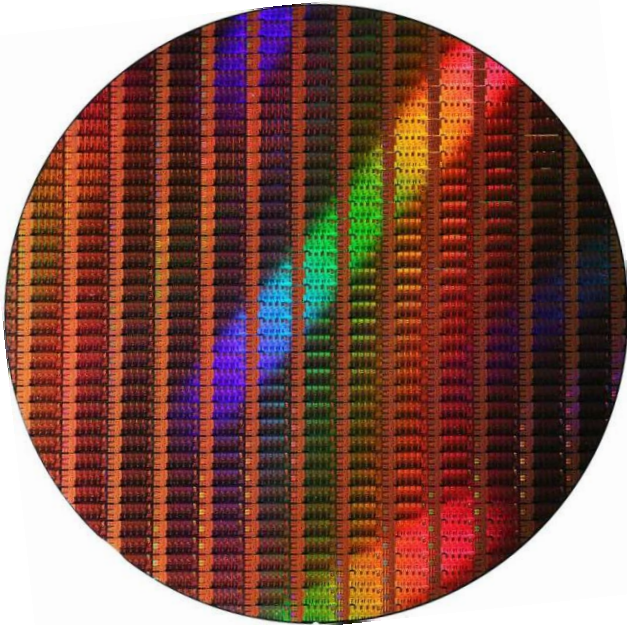


# High Guardbands in Modern Processors



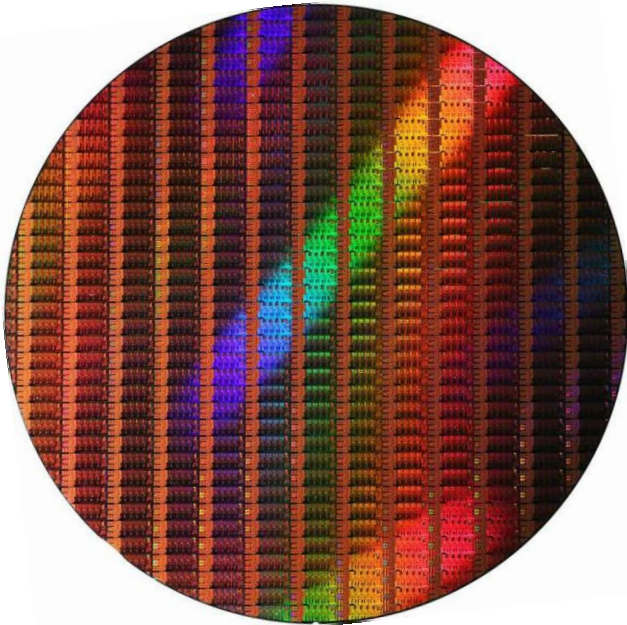


# High Guardbands in Modern Processors

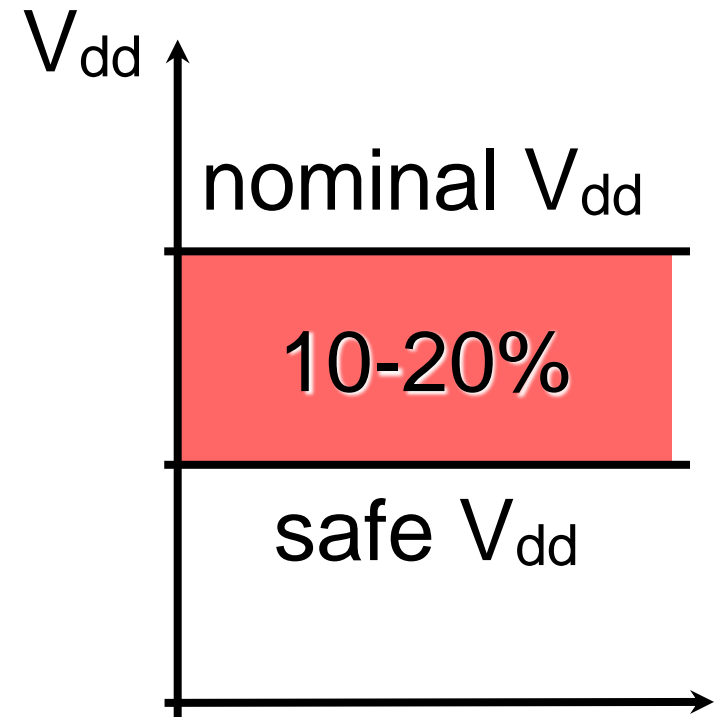




# High Guardbands in Modern Processors

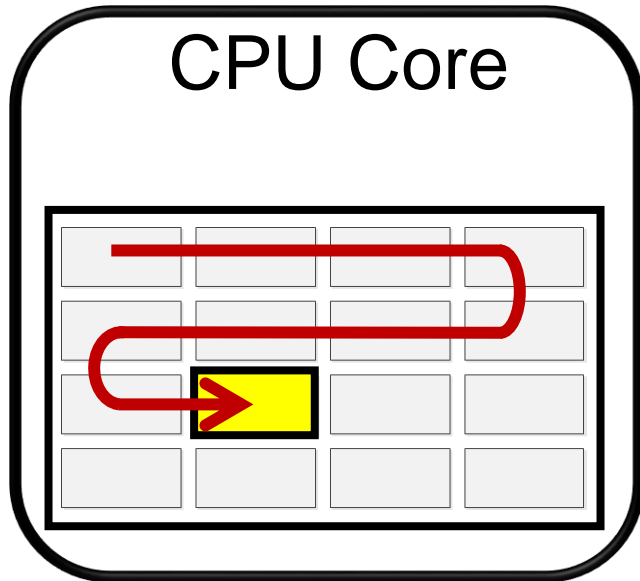


Energy inefficient





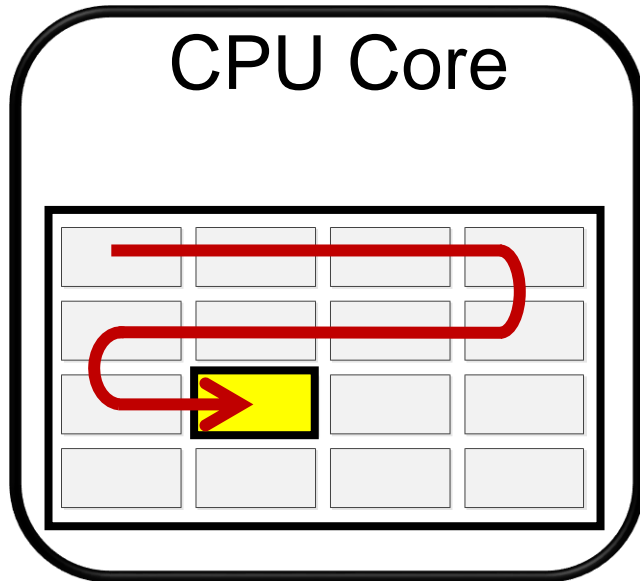
# ECC Guided Voltage Speculation



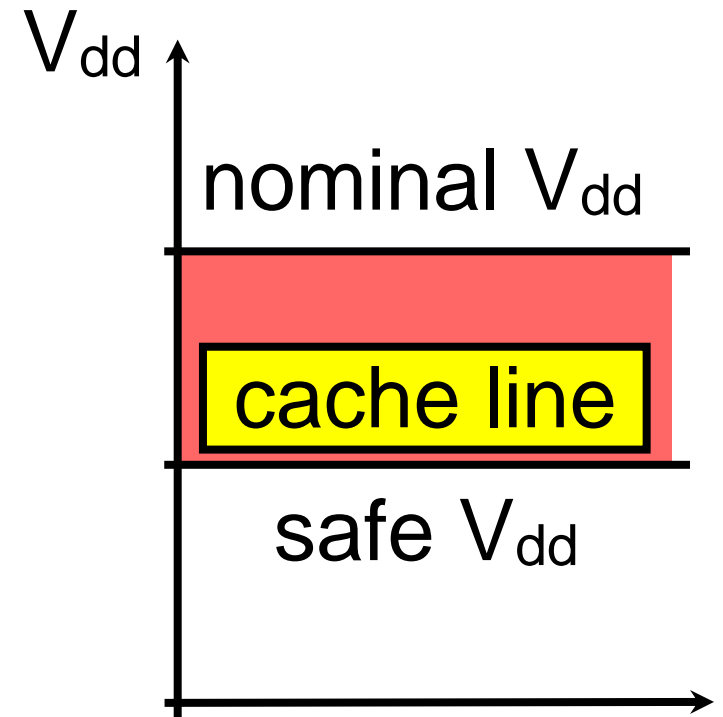
**Find weakest  
cache line**



# ECC Guided Voltage Speculation



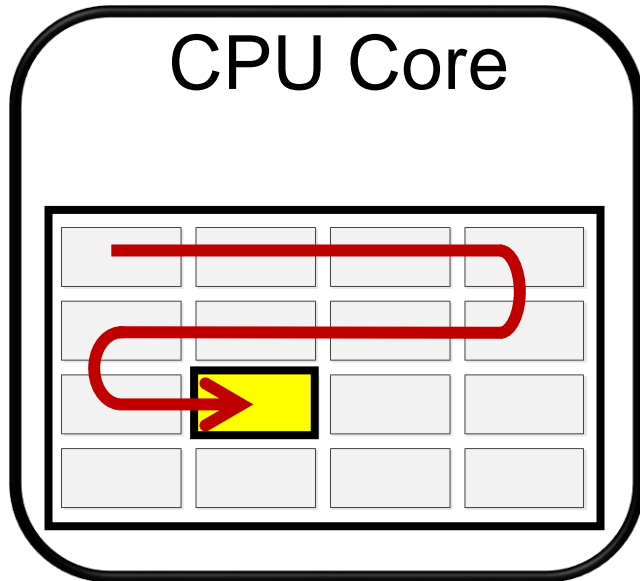
**Find weakest  
cache line**



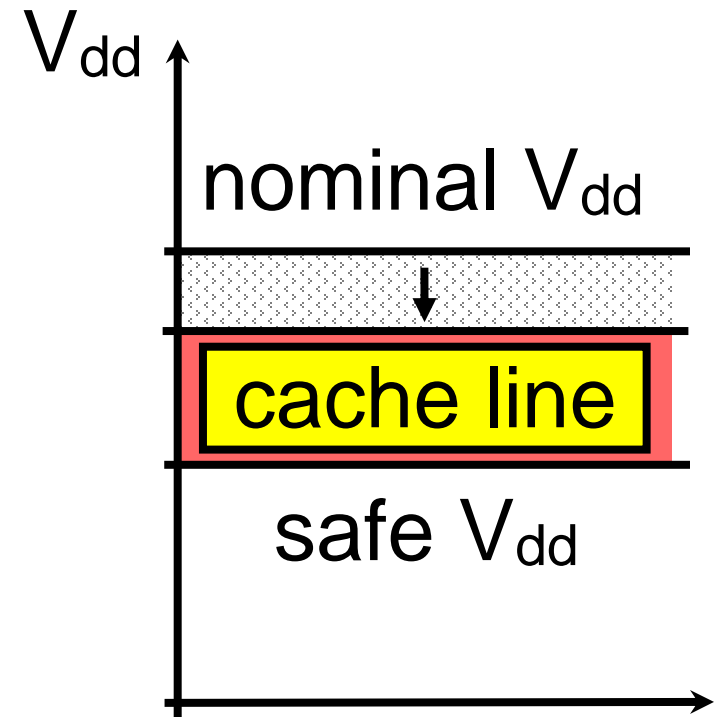
**Monitor weakest  
cache line**



# ECC Guided Voltage Speculation



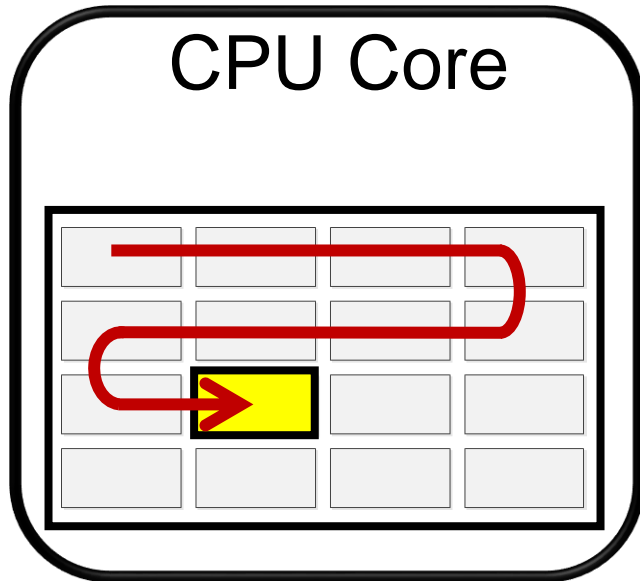
**Find weakest  
cache line**



**Monitor weakest  
cache line**



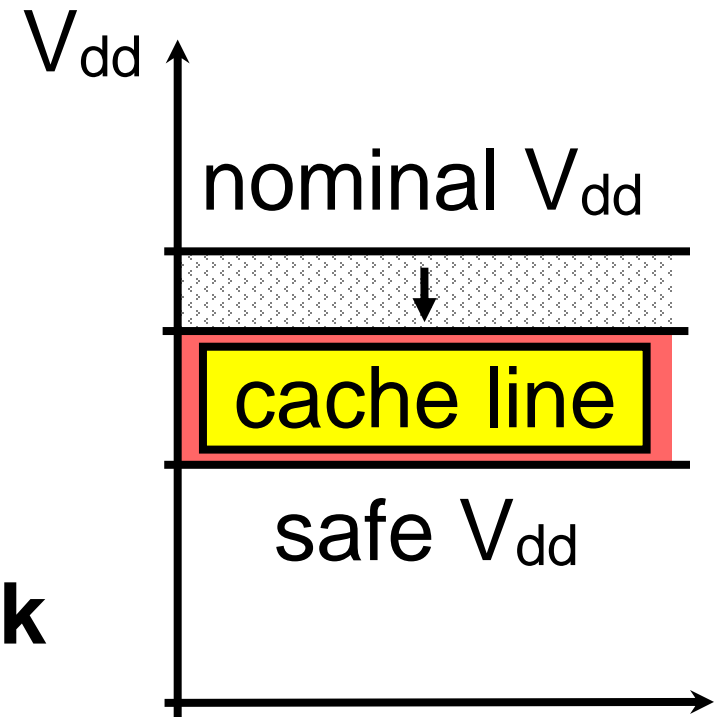
# ECC Guided Voltage Speculation



**Find weakest  
cache line**



**Caution  
ECC Feedback**



**Monitor weakest  
cache line**



# Real System





**Real  
System**

**Real  
Results**



**Real System      Real Results**

**33%**

**Energy Savings**



**Real Real**



**Coming soon to an  
Umney Theatre near you!**

**Energy Savings**

*next paper*

# Harnessing Soft Computation for Low-Budget Fault Tolerance

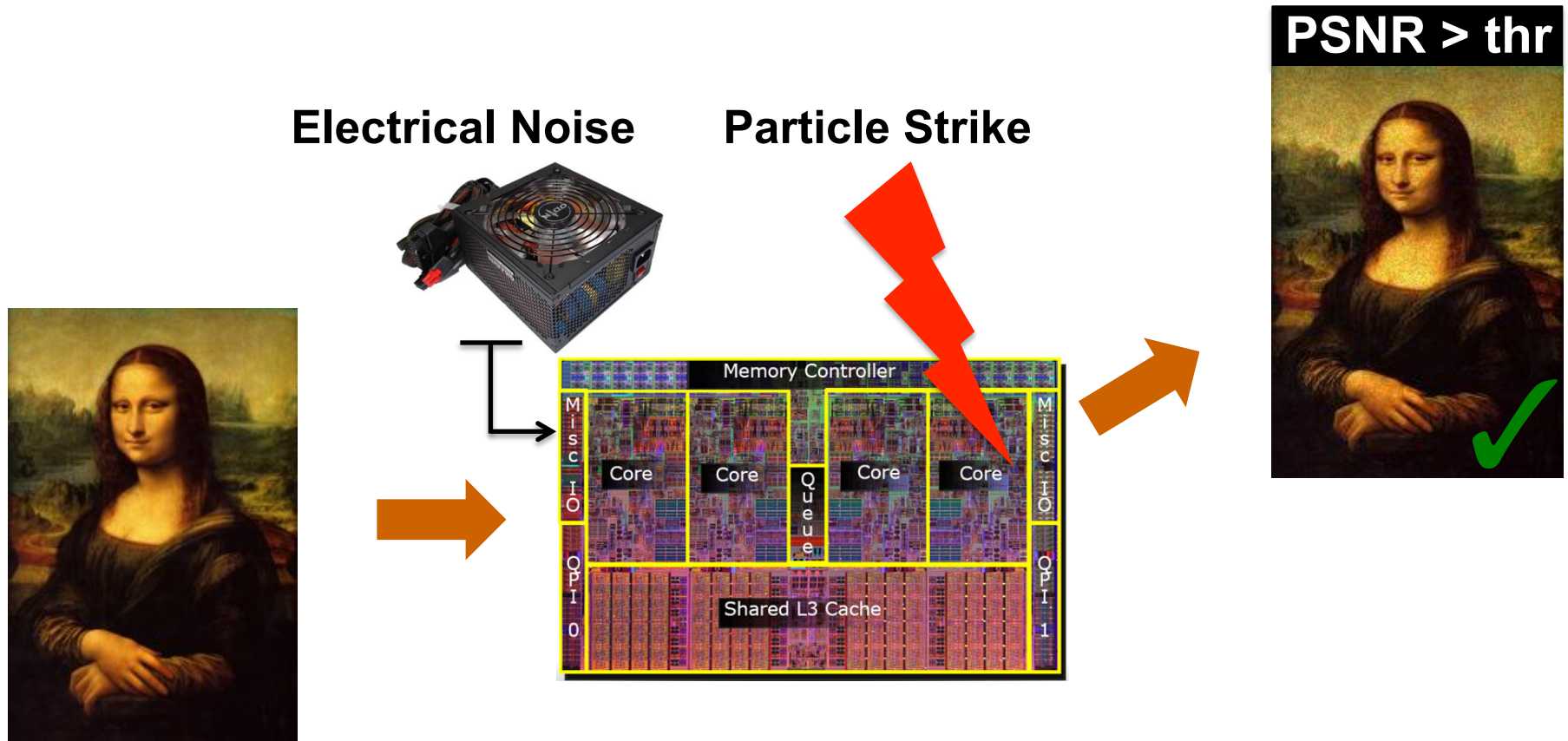
Daya S Khudia

Scott Mahlke

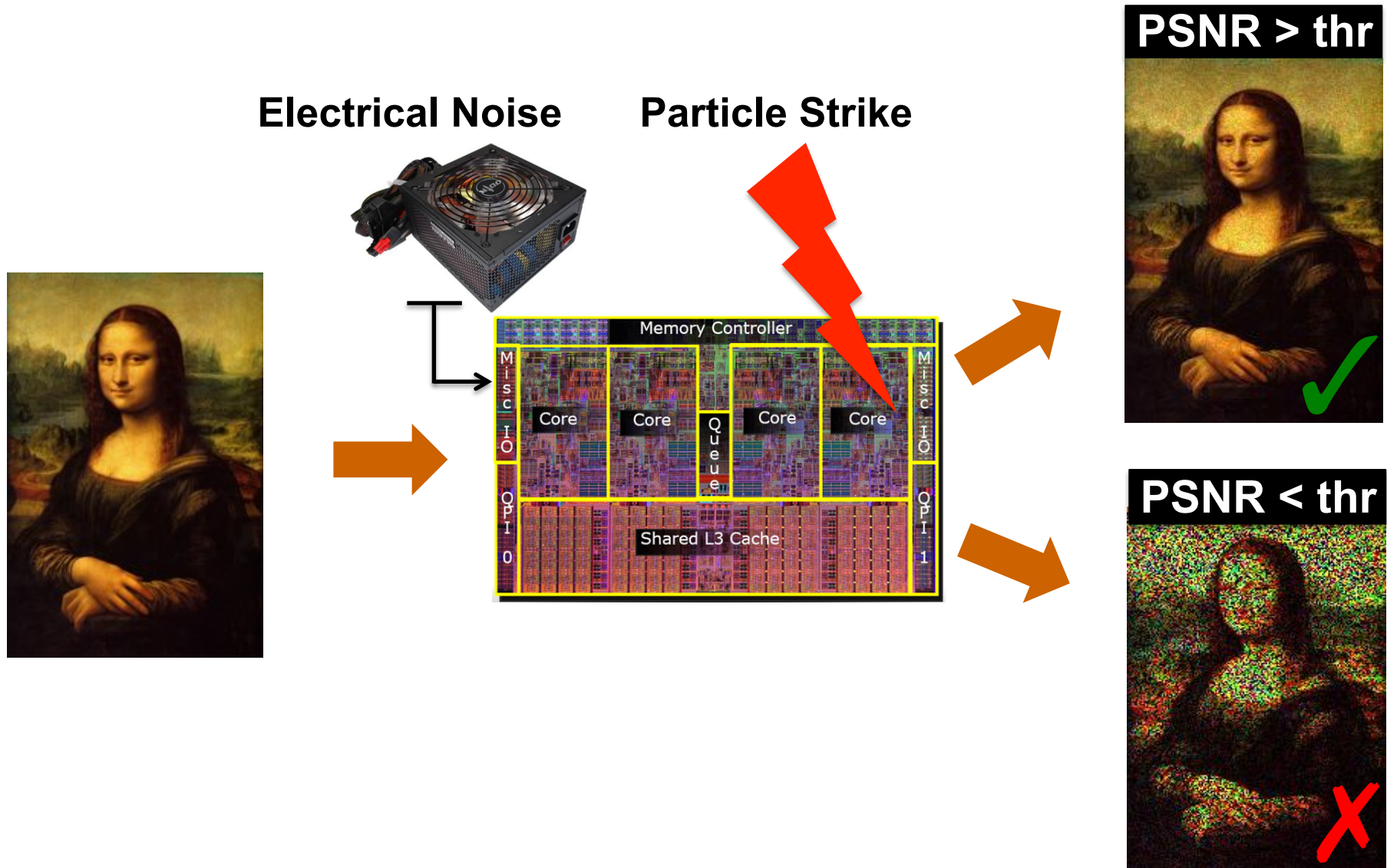
Advanced Computer Architecture Laboratory  
University of Michigan, Ann Arbor



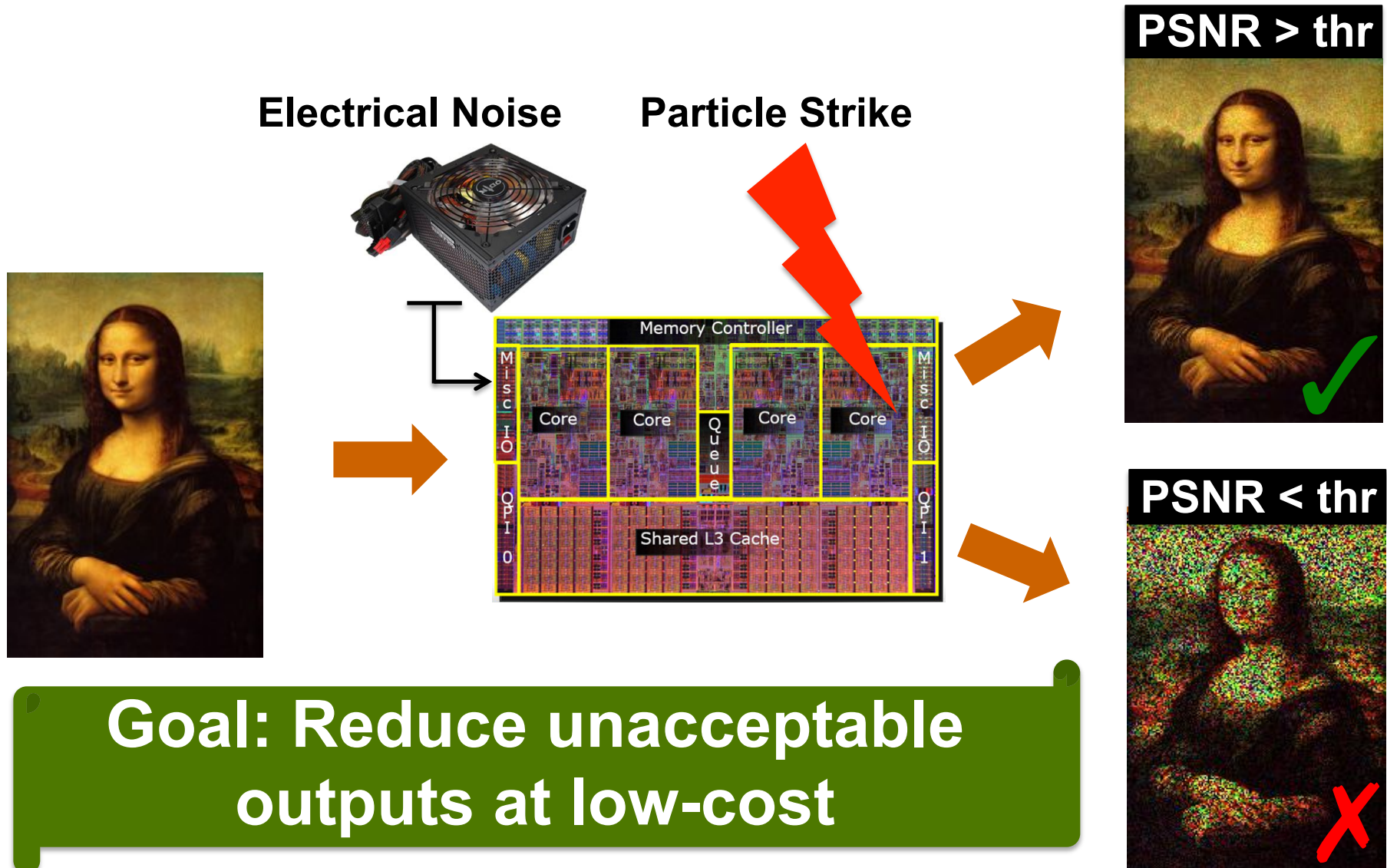
# Acceptable Vs. Unacceptable Outputs



# Acceptable Vs. Unacceptable Outputs



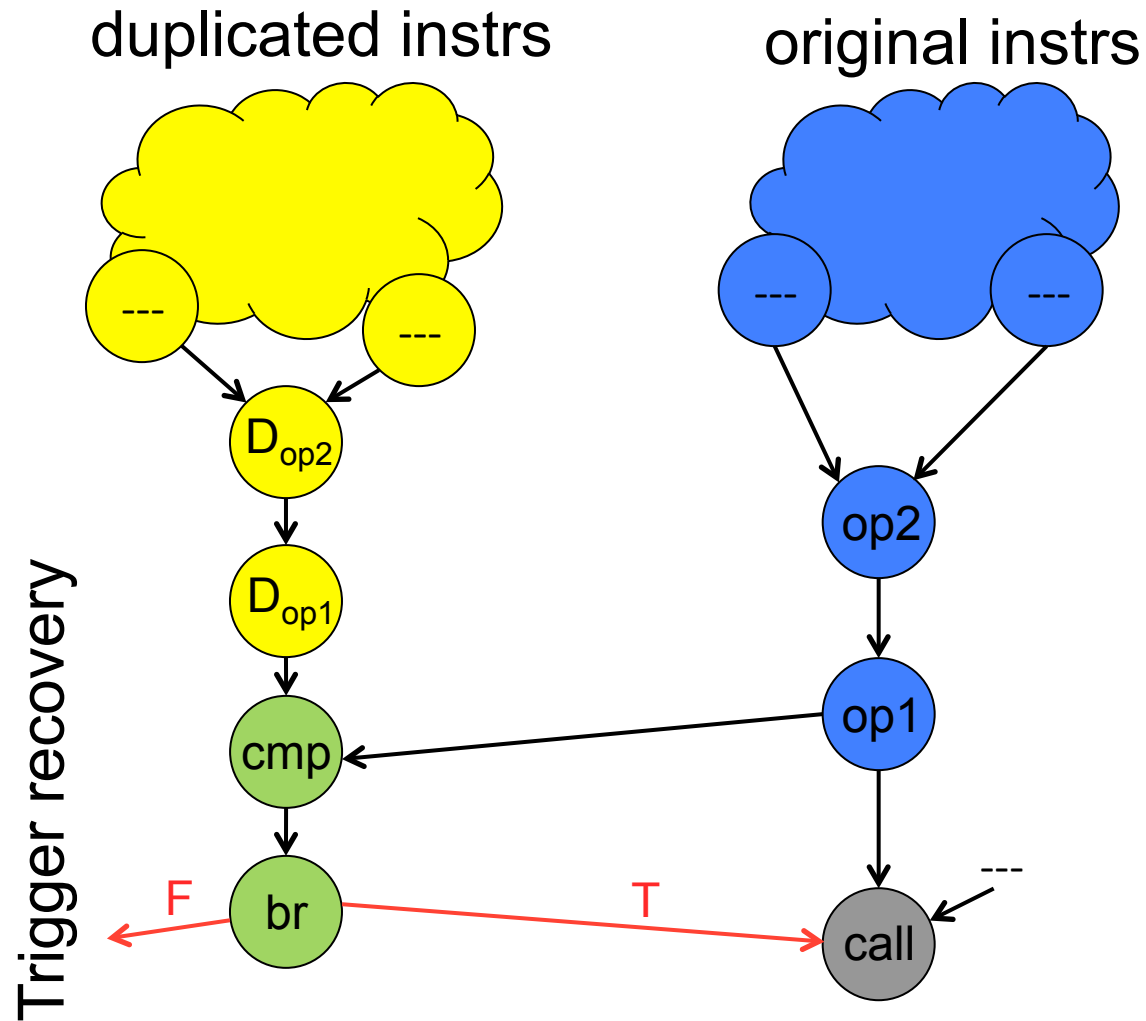
# Acceptable Vs. Unacceptable Outputs



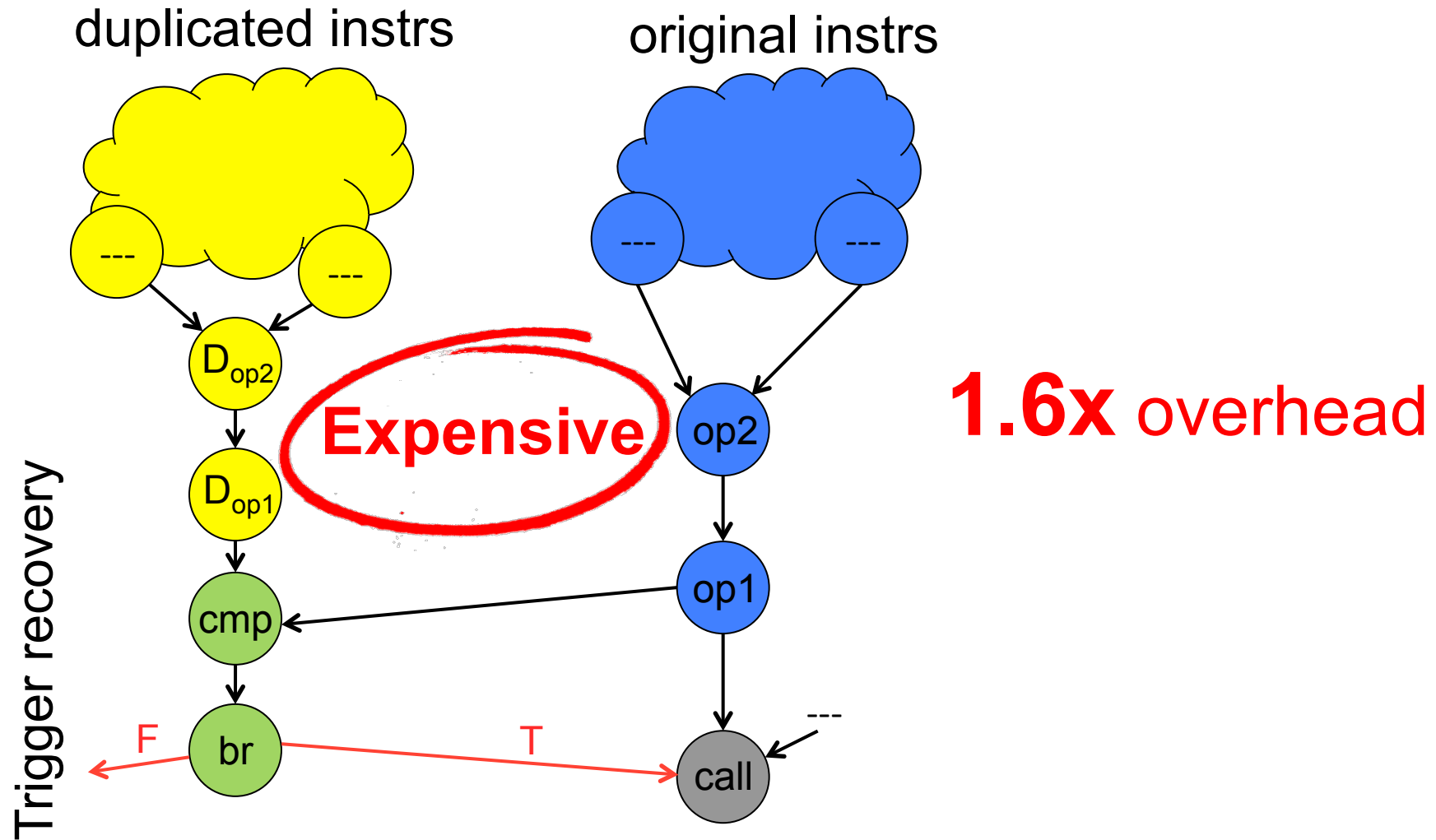
**Goal: Reduce unacceptable outputs at low-cost**



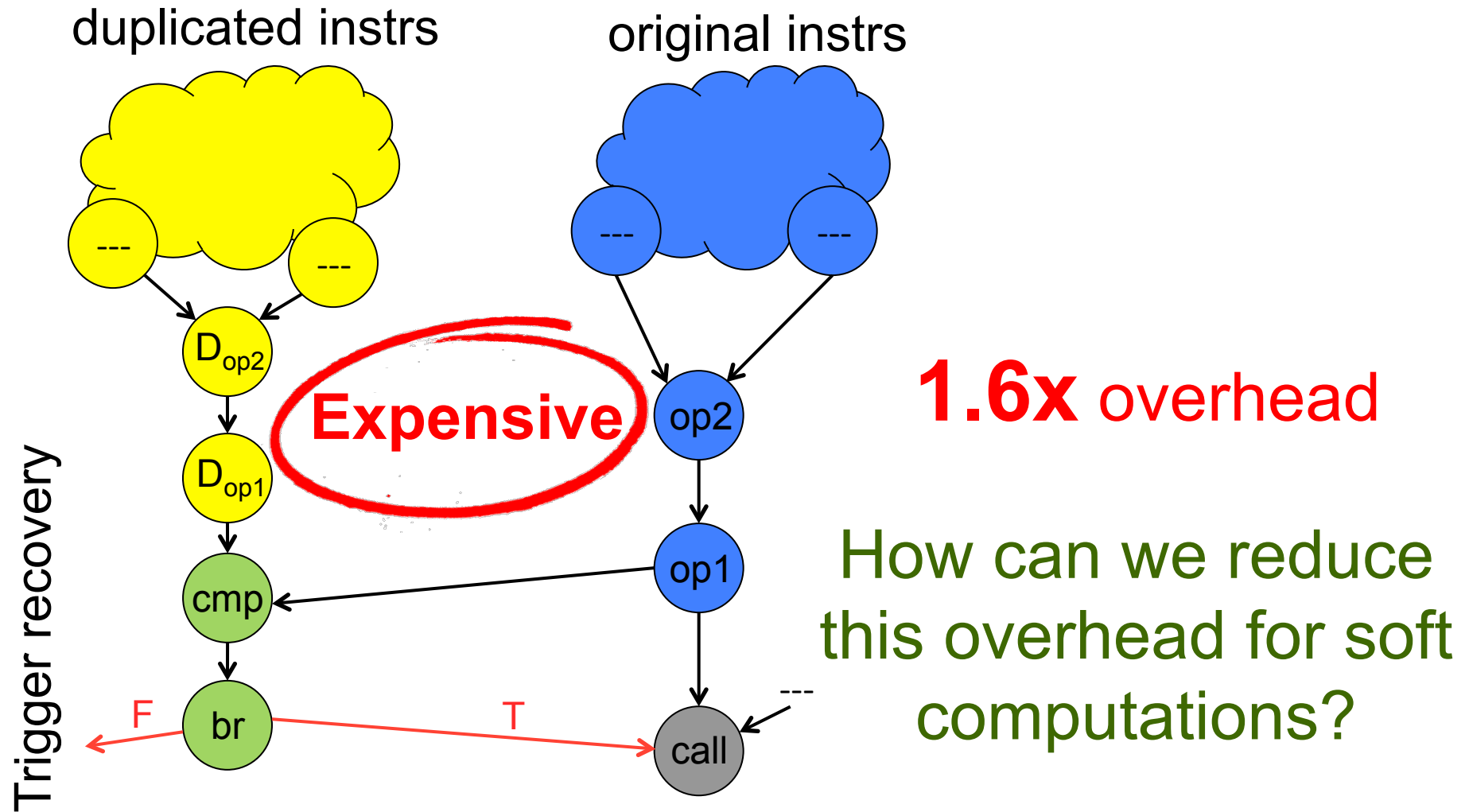
# Traditional Duplication



# Traditional Duplication



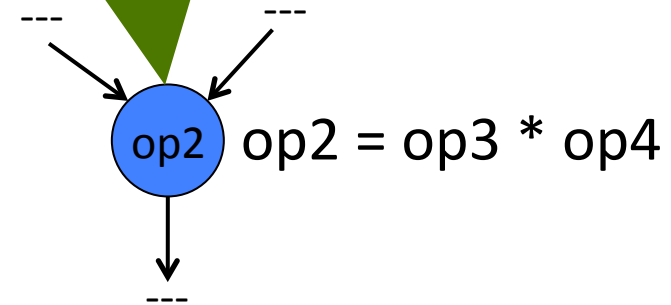
# Traditional Duplication



# Selective Duplication and Value Checks

- Duplication for critical variables
- Exploit value locality and check for deviations

-Produces 0 more than thr  
-Insert value comparison



**1.2x**

Performance Overhead

**2.8x**

Reduction in  
unacceptable outputs

*next paper*

Somayeh Sardashti

André Seznec

David A. Wood

UW-Madison

INRIA/IRISA

UW-Madison

~~Decoupled Compressed Cache~~

Micro 2013

Skewed ~~TLB~~

IEEE TC 2004

**Skewed Compressed Cache**

**Micro 2014**

## What did we borrow?

From Decoupled Compressed Cache:

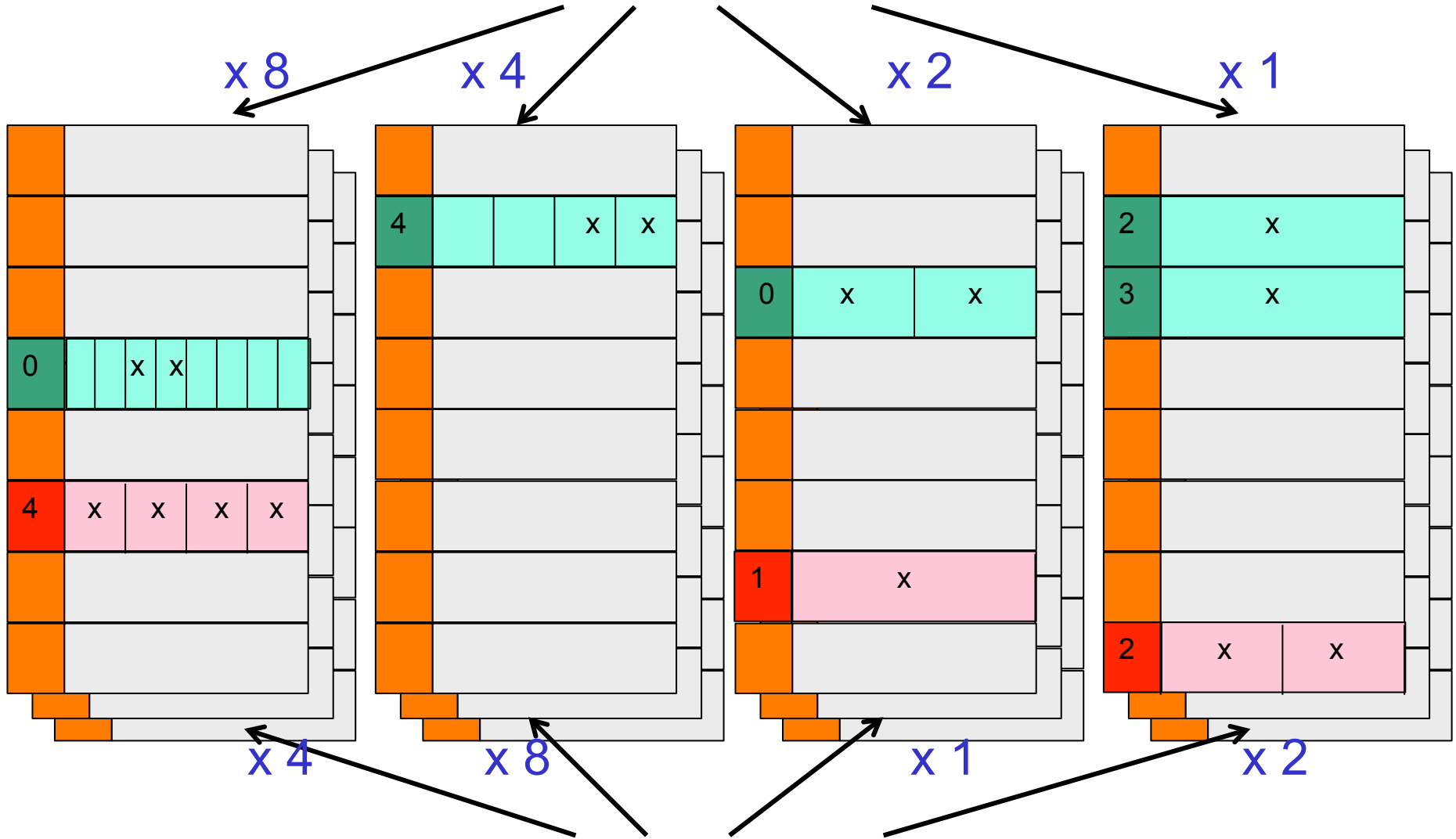
- Compression factor is a spatial property
- Leverage superblocks to limit tag overhead
- ~~Complex look-up and replacement management~~
- ~~Meta data overhead~~

From Skewed TLB:

- A multiple grain structure in a N-way (skewed) cache
- The analogy:

At read time, page size (compression factor) is unknown

# Superblock A000



# Superblock B000



## Skewed Compressed Cache

- Leverages spatial compression factor locality
  - « x 2 » the LLC size
- Direct tag-data mapping
  - Simple allocation/replacement automaton
  - No meta-data, only 1.6 % storage overhead



**Last step towards effective compressed caching?**

*next paper*



# Adaptive Cache Management for Energy-efficient GPU Computing

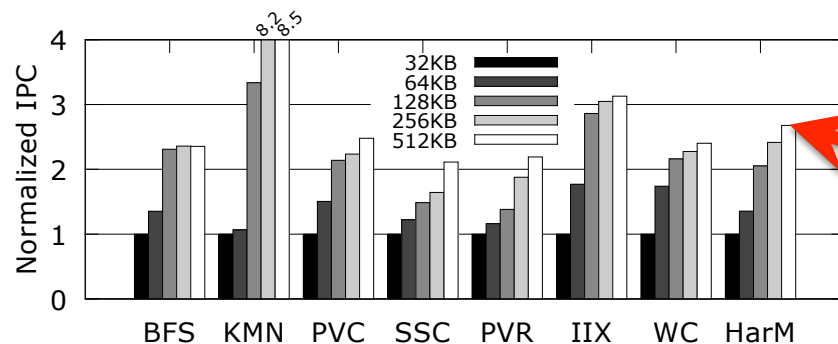
Xuhao Chen<sup>1,2,3</sup>, Li-Wen Chang<sup>3</sup>, Chris Rodrigues<sup>3</sup>, Jie Lv<sup>3</sup>, Zhiying Wang<sup>1,2</sup>, Wen-Mei Hwu<sup>3</sup>

[1] State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha, China

[2] School of Computer, National University of Defense Technology, Changsha, China

[3] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, USA

- Many cache sensitive GPU applications have severe cache contention → low cache efficiency → poor performance
  - Smaller L1 cache capacity per thread
- Existing management schemes have limitations
- We propose **Coordinated Bypassing and Warp Throttling (CBWT)** to improve GPU cache efficiency
  - Reduce cache contention rate and NoC latency



A **2.68x** speedup on average (harmonic mean) for highly cache sensitive (HCS) benchmarks

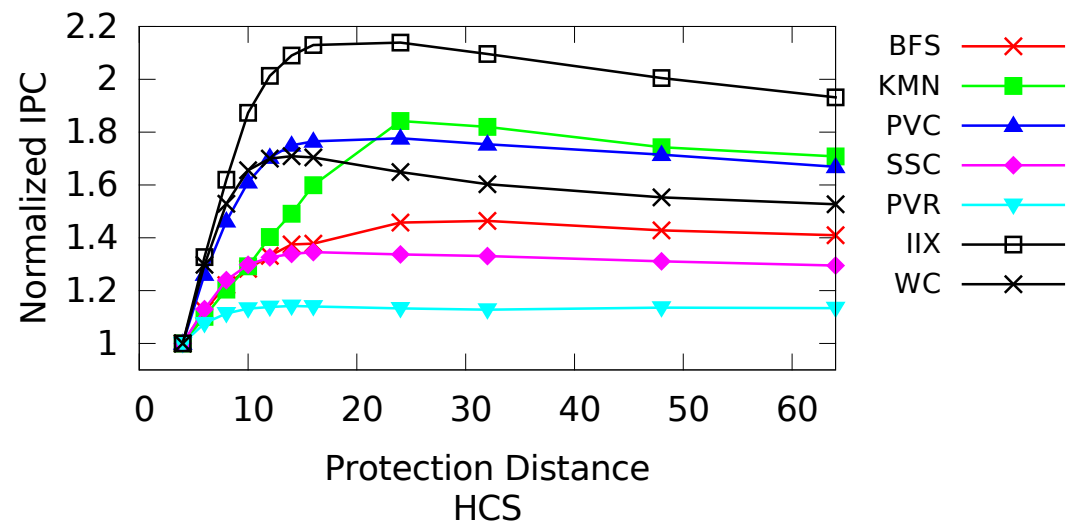
# Observations – Understanding the Limitations

- Cache bypassing retains useful cache lines instead of replacing upon miss

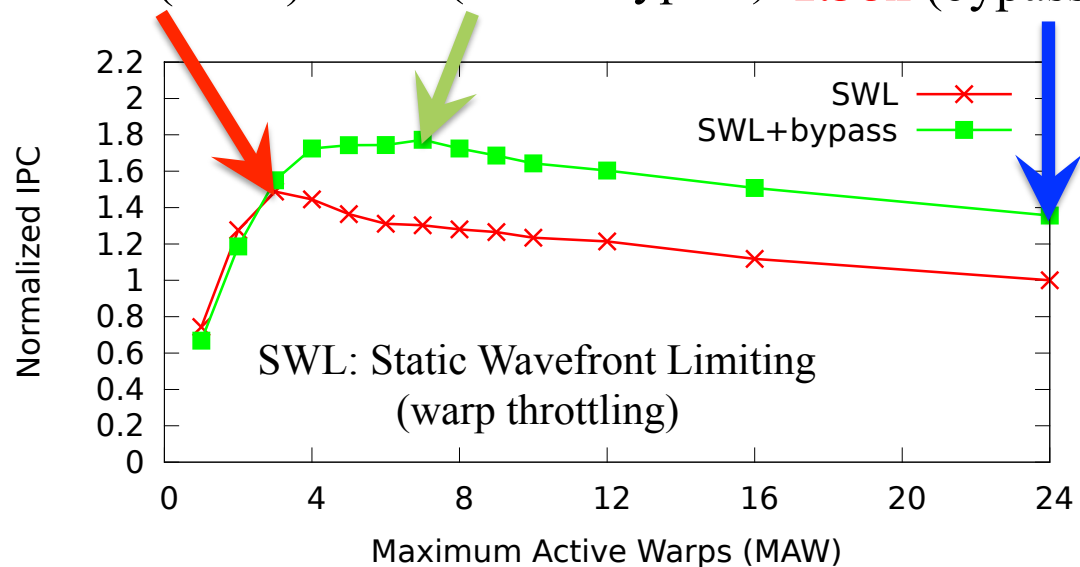
- ✓ Retain useful data → fewer cache misses per thread
  - ✧ Average HCS speedup **1.57x**
- ✗ High demand on NoC to serve misses, i.e. **congestion**
- ✗ Still cannot avoid **locality loss**

- Warp throttling temporarily deactivates some threads

- ✓ Fewer threads → more cache per thread → fewer misses
- ✗ Few threads → **cannot hide latency** through multithreading
- ✗ Resource **under-utilization**

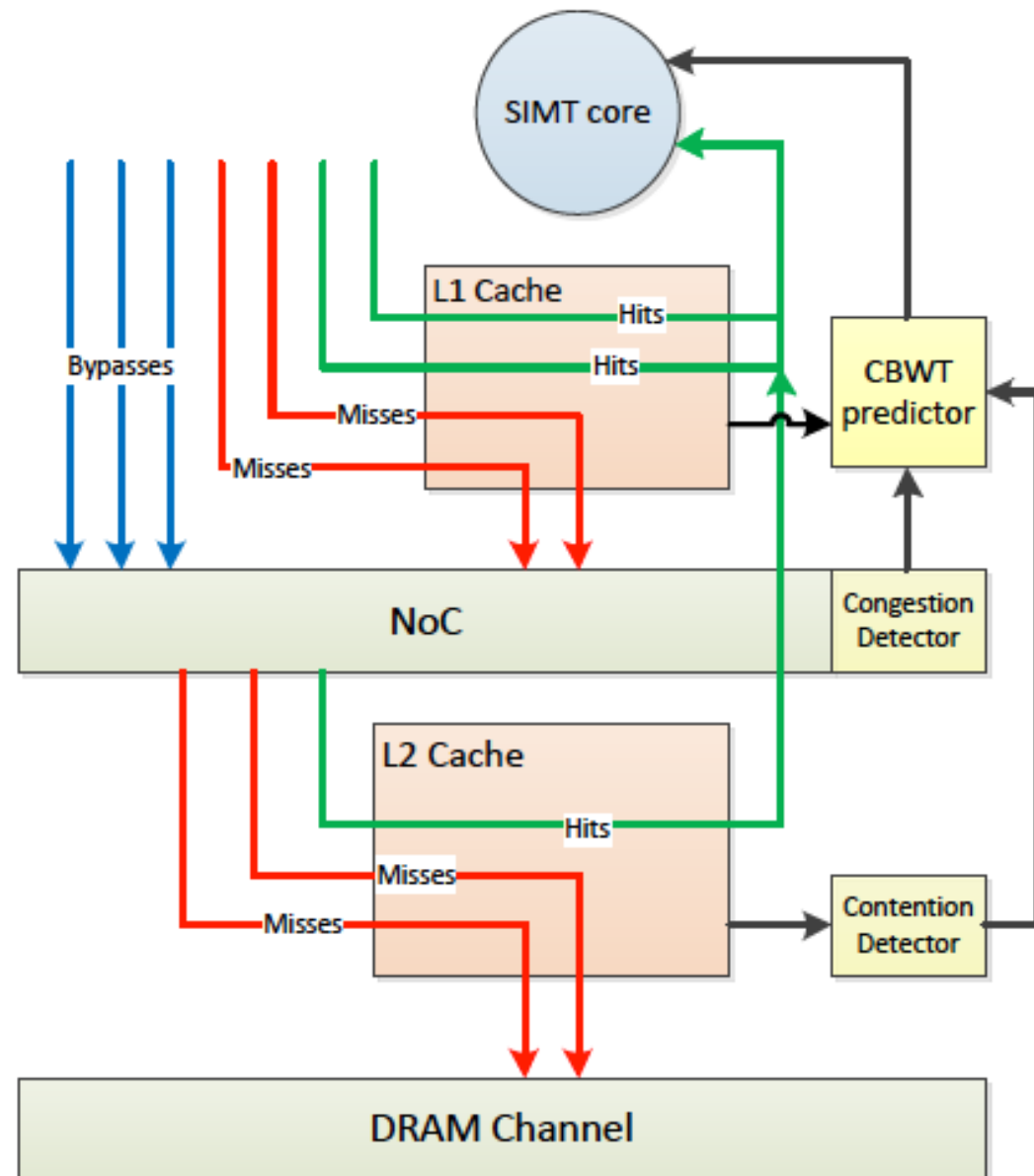
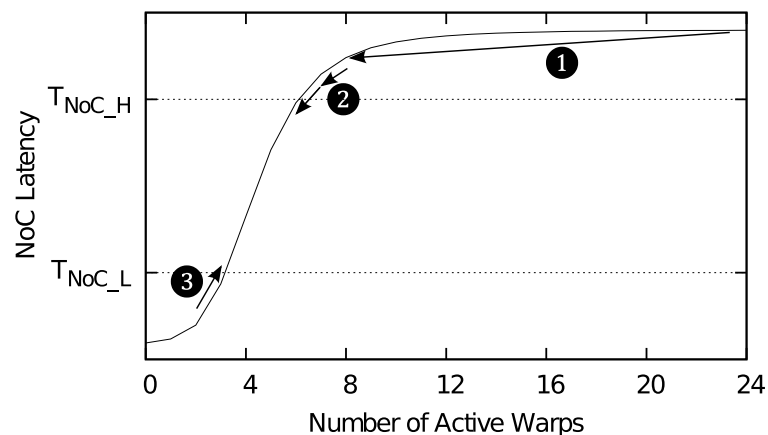


**1.46x (SWL) 1.84x (SWL+bypass) 1.38x (bypass)**



# CBWT Architecture Overview

- Extra sampling modules (yellow blocks) are added to monitor *contention* and *congestion*.
- Adjust the MAW to keep the network in a *busy* but *low-congestion* range.



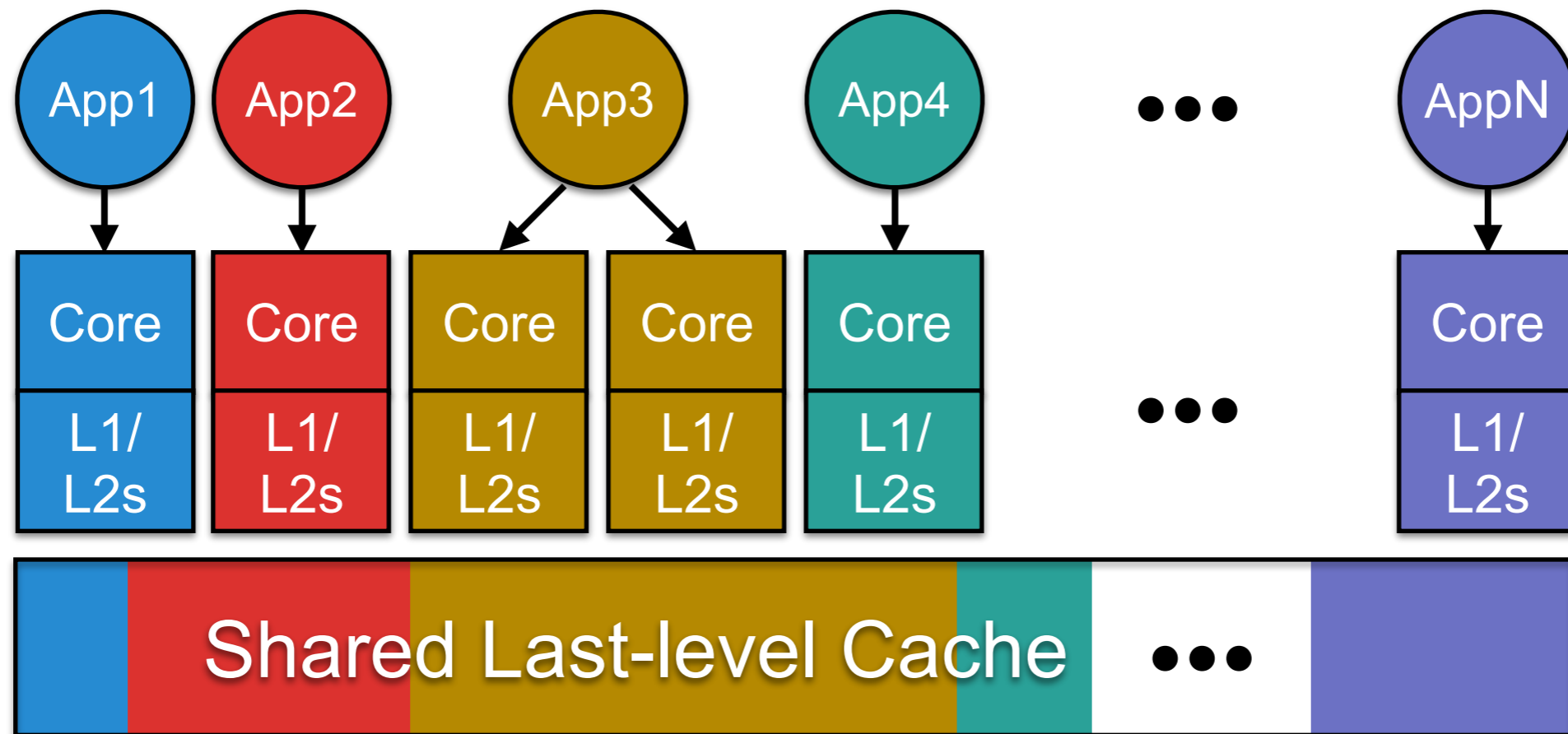


# Performance and Energy-efficiency

- CBWT achieves an average of **74%** (maximum **661%**) IPC improvement on HCS benchmarks over baseline, which significantly outperforms PDP bypassing (**42%**) and Best-SWL (**52%**).
  - PDP bypassing: pure cache bypassing
  - Best-SWL: pure warp throttling
- CBWT outperforms the baseline with an average of **58.6%** Perf/Watt improvement
  - On average, PDP bypassing can reduce **16.5%** of DRAM traffic,
  - CBWT reduces DRAM traffic by **54.9%**
- Welcome to **Session 4A** in **Main Auditorium** on **Dec. 16** (Tuesday) at **11:10 AM** for more details

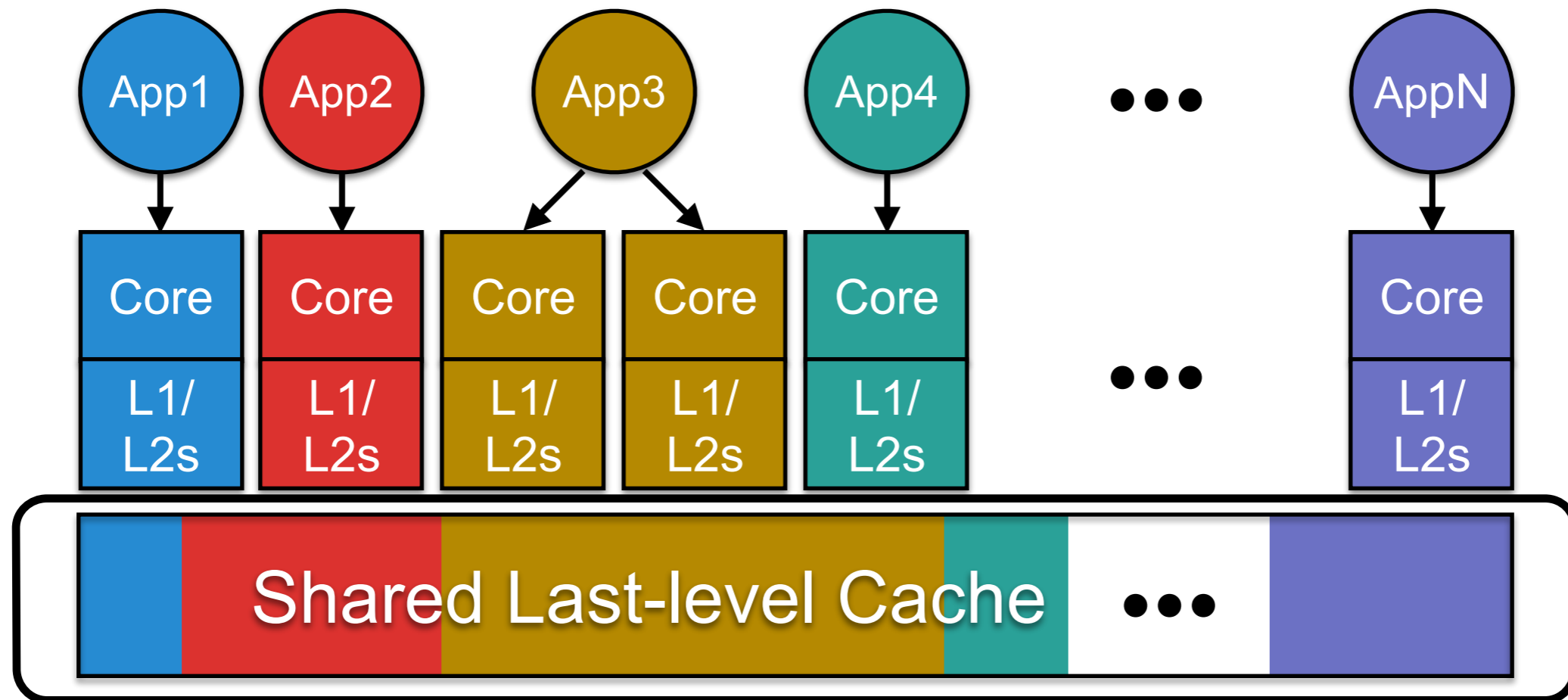
*next paper*

# Problem: How to efficiently divide a cache into hundreds of partitions?



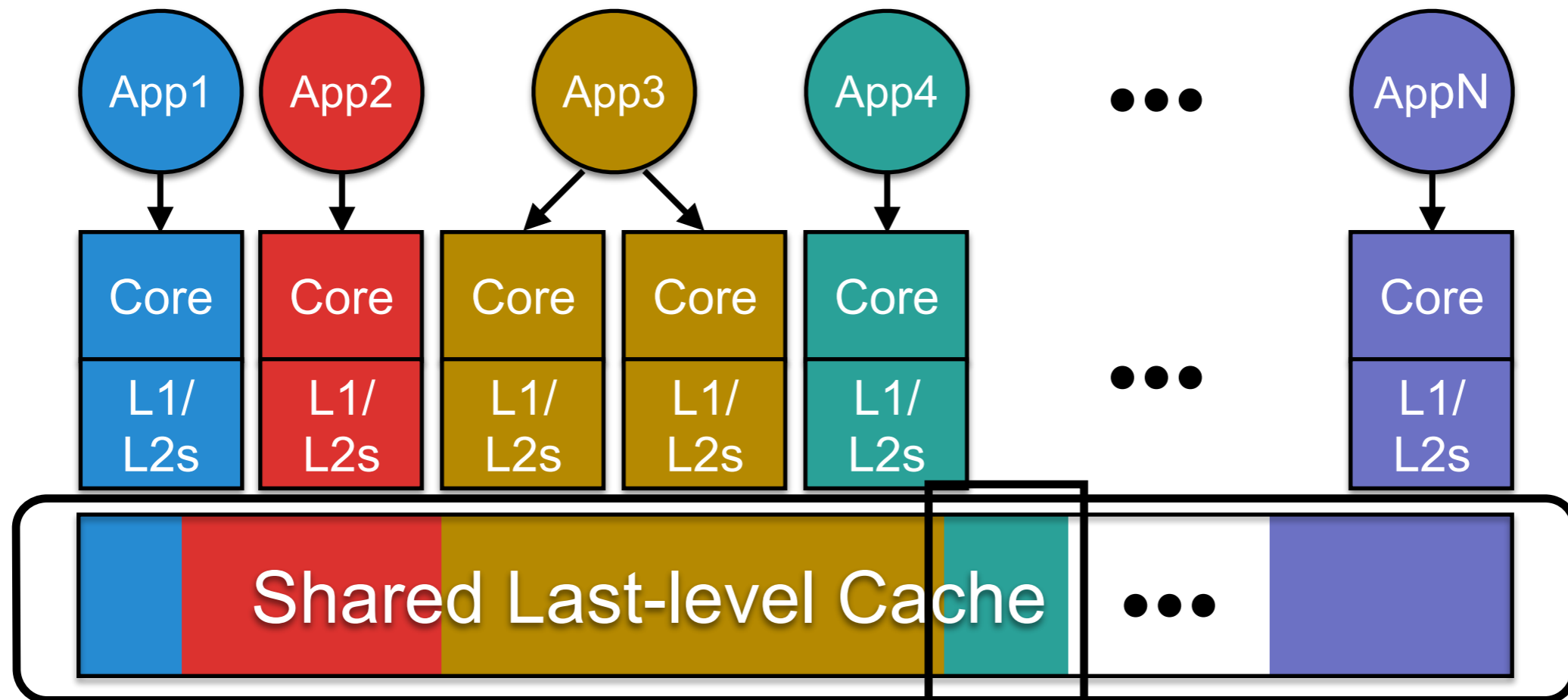


# Problem: How to efficiently divide a cache into hundreds of partitions?



Hundreds of partitions

# Problem: How to efficiently divide a cache into hundreds of partitions?







Hundreds of partitions

For each partition





- ◆ Precisely control its size (fine-grain)
- ◆ Maintain high associativity

 Partition 1  Partition 2

	Way1	Way2	Way3
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			

Way partitioning

 Partition 1  Partition 2



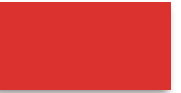














	Way1	Way2	Way3
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			

Way partitioning

-  Few coarse-grain partitions
-  Low associativity


 Partition 1  Partition 2

 Partition 3  Partition 4

	Way1	Way2	Way3
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			












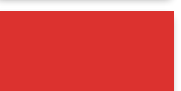




Way partitioning

-  Few coarse-grain partitions
-  Low associativity

	Way1	Way2	Way3
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			

Control the partition size by adjusting eviction rate at replacement


 Partition 1  Partition 2

	Way1	Way2	Way3
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			


Way partitioning

-  Few coarse-grain partitions
-  Low associativity

 Partition 3  Partition 4

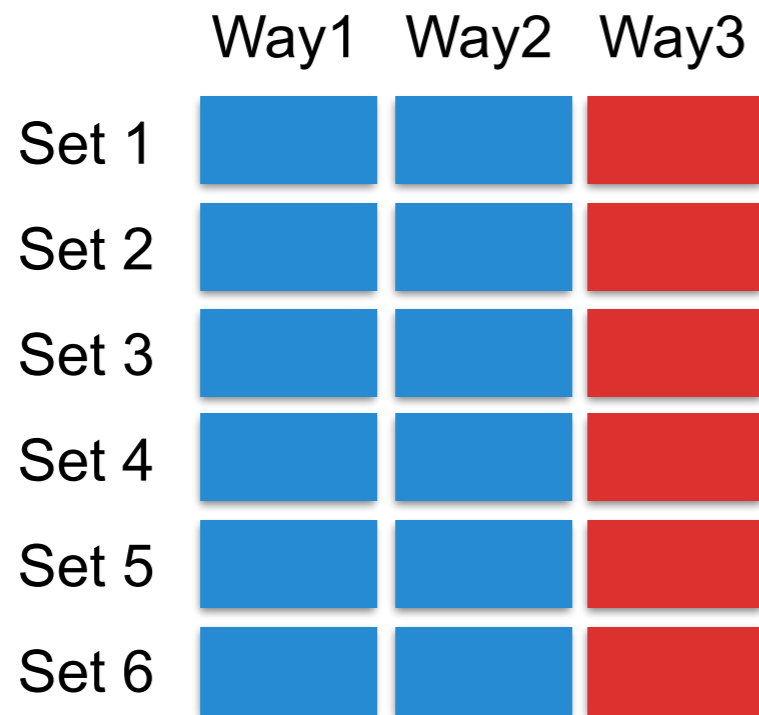
	Way1	Way2	Way3
Set 1			
Set 2			
Set 3			
Set 4			
Set 5			
Set 6			

Control the partition size by adjusting eviction rate at replacement

-  Able to support many fine-grain partitions

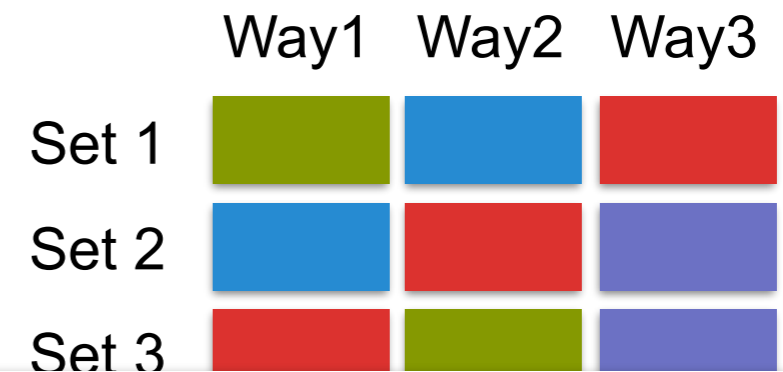
■ Partition 1 ■ Partition 2

■ Partition 3 ■ Partition 4



Way partitioning

- ✗ Few coarse-grain partitions
- ✗ Low associativity



How to **precisely partition** the cache while still maintaining **high associativity**?

Control the partition size by adjusting eviction rate at replacement

- ✓ Able to support many fine-grain partitions

# Futility Scaling



# Futility Scaling

Basic idea: control the size of each partition by properly scaling the futility of its cache lines

# Futility Scaling

Basic idea: control the size of each partition by properly scaling the futility of its cache lines

Intuition:

**Scaling up futility** → cache lines are evaluated as less useful → lower probability to be kept in the cache → **partition size is reduced**

# Futility Scaling

Basic idea: control the size of each partition by properly scaling the futility of its cache lines

Intuition:

**Scaling up futility** → cache lines are evaluated as less useful → lower probability to be kept in the cache → **partition size is reduced**

More details are in our tomorrow's talk:

- ❖ Futility estimation
- ❖ Scaling factor adjustment
- ❖ Feedback-based implementation

**Tomorrow at 10:45am in Session 4A  
“TLB and Cache Optimization”**

# **Futility Scaling: High- Associativity Cache Partitioning**

**Ruisheng Wang — University of Southern California**  
**Lizhong Chen — Oregon State University**

**USC**  
**Viterbi**  
School of Engineering

**OSU**  
**Oregon State**  
UNIVERSITY  

---

**College of Engineering**

*next paper*

# Voltage Noise in Multi-core Processors: Empirical Characterization and Optimization Opportunities

**Ramon Bertran**<sup>1</sup>, Alper Buyuktosunoglu<sup>1</sup>, Pradip Bose<sup>1</sup>,  
Timothy J. Slegel<sup>2</sup>, Gerard Salem<sup>2</sup>, Sean Carey<sup>2</sup>,  
Richard F. Rizzolo<sup>2</sup>, Thomas Strach<sup>2</sup>

***<sup>1</sup>IBM Research***

***<sup>2</sup>IBM Systems & Technology Group***

***Full presentation:***

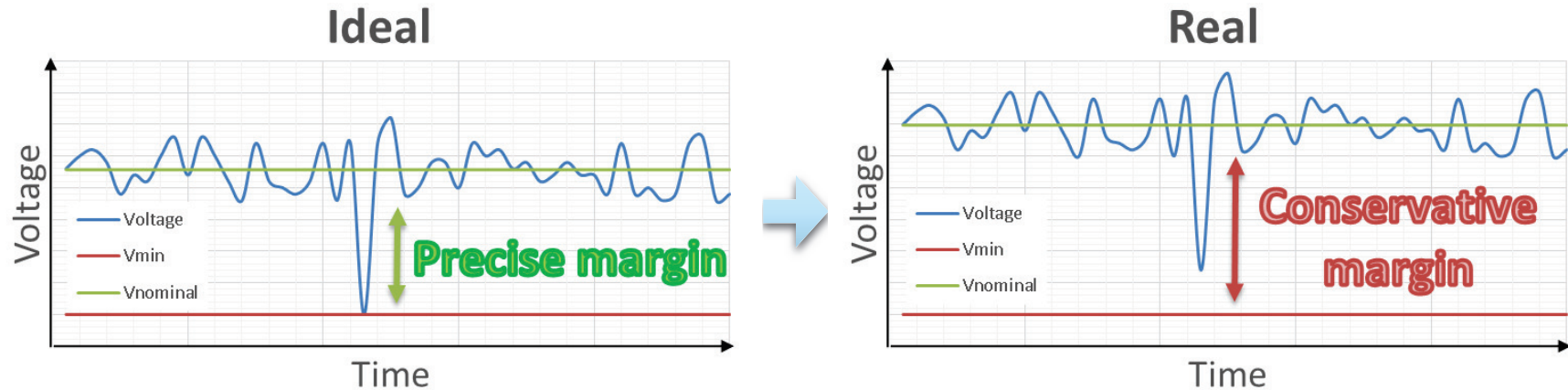
***Session 4B Managing Voltage and Time***

***Tomorrow at 10:45AM***

***Umney Theatre & Lounge room***



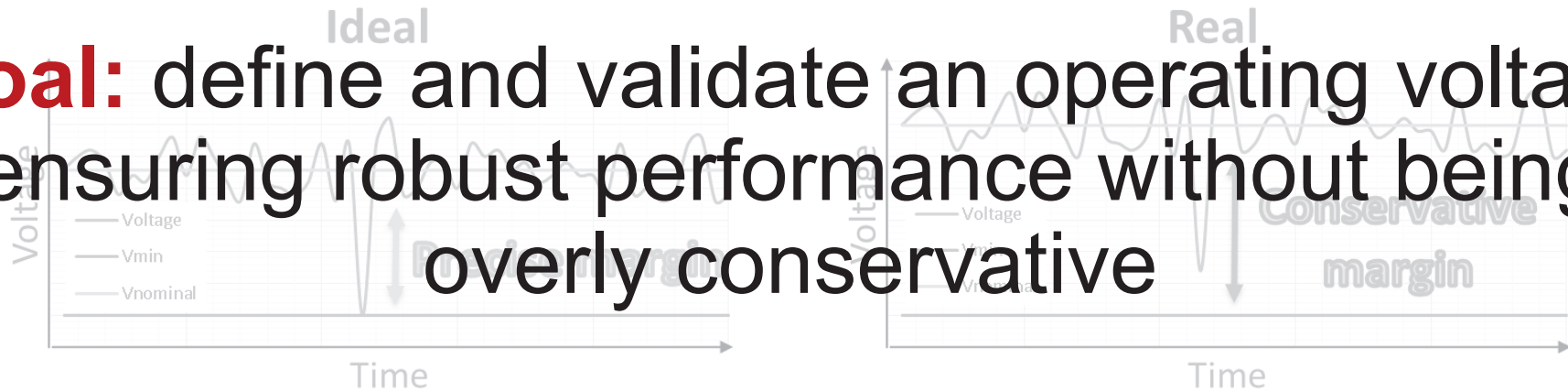
# Voltage noise: Transient variations in supply voltage



**Full presentation: Session 4B Managing Voltage and Time  
Tomorrow at 10:45AM - Umney Theatre & Lounge room**

## Voltage noise: Transient variations in supply voltage

**Goal:** define and validate an operating voltage ensuring robust performance without being overly conservative

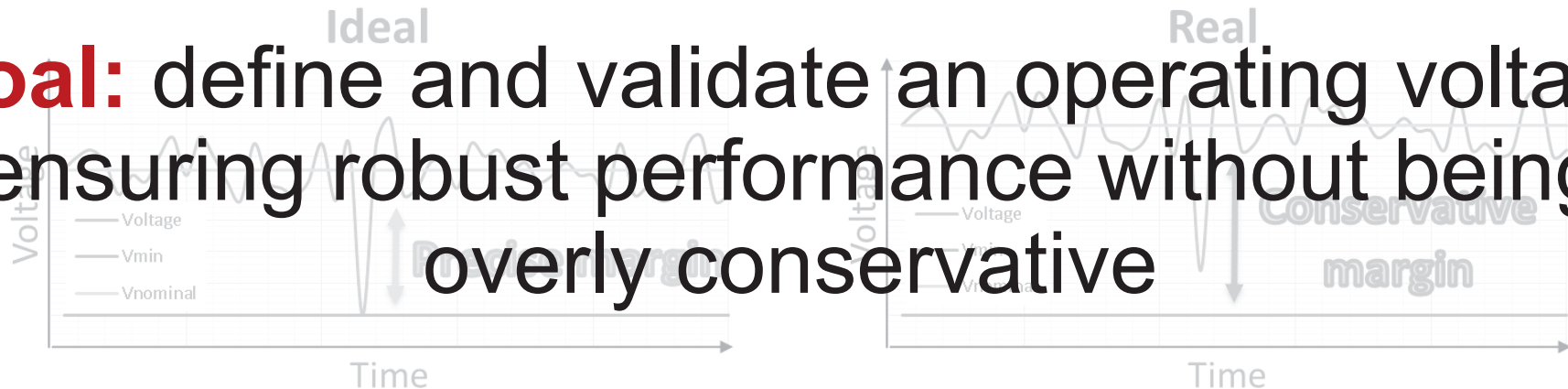


*Full presentation: Session 4B Managing Voltage and Time  
Tomorrow at 10:45AM - Umney Theatre & Lounge room*



## Voltage noise: Transient variations in supply voltage

**Goal:** define and validate an operating voltage ensuring robust performance without being overly conservative

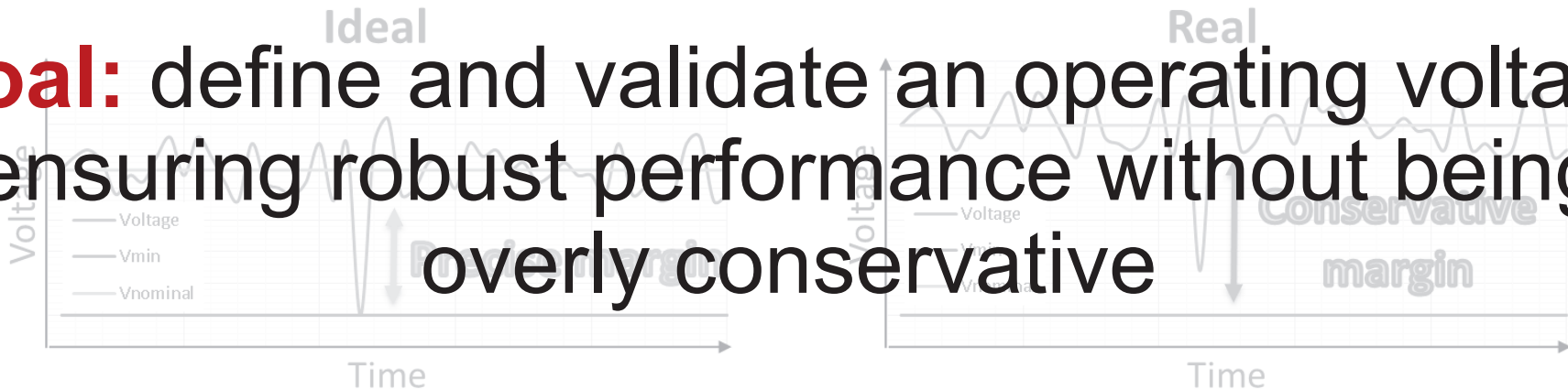


- Pre-silicon characterization is inadequate
- A systematic, direct measurement-based characterization is required

*Full presentation: Session 4B Managing Voltage and Time  
Tomorrow at 10:45AM - Umney Theatre & Lounge room*

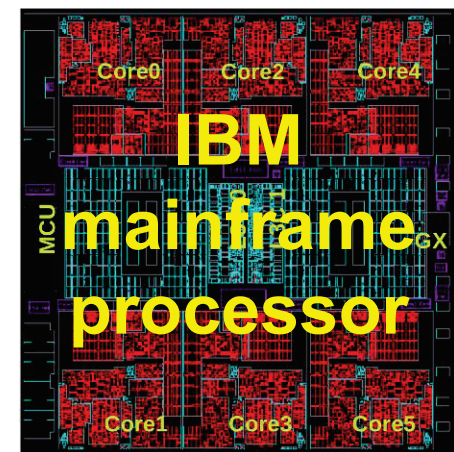
# Voltage noise: Transient variations in supply voltage

**Goal:** define and validate an operating voltage ensuring robust performance without being overly conservative



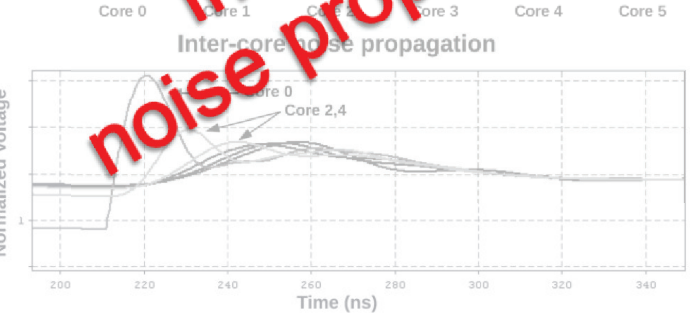
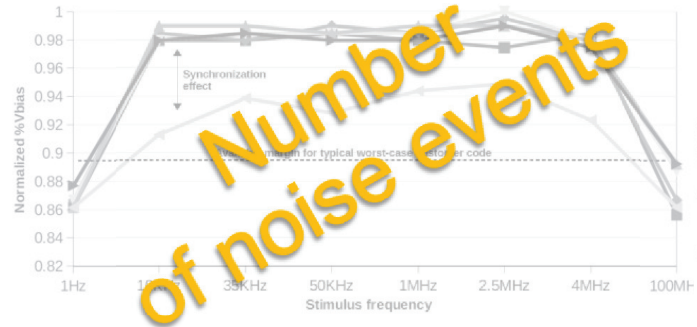
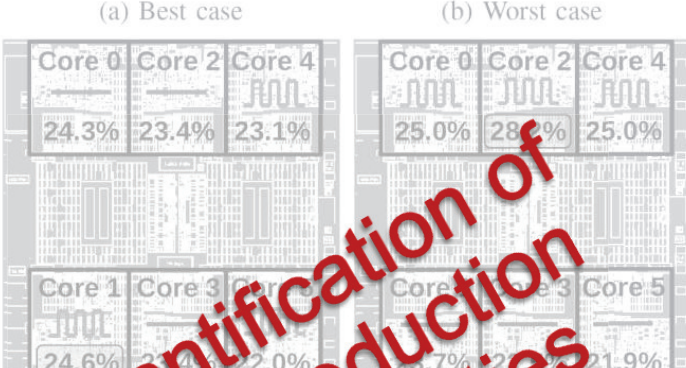
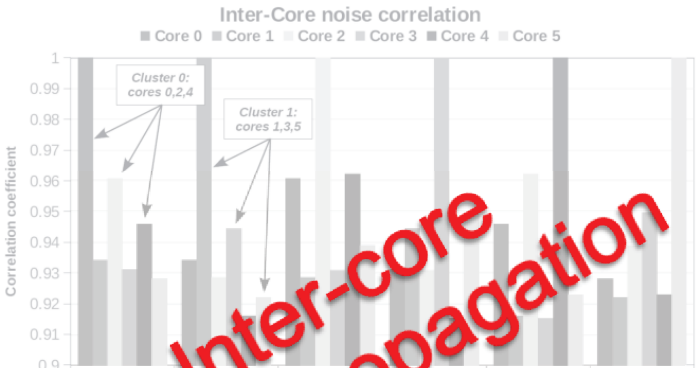
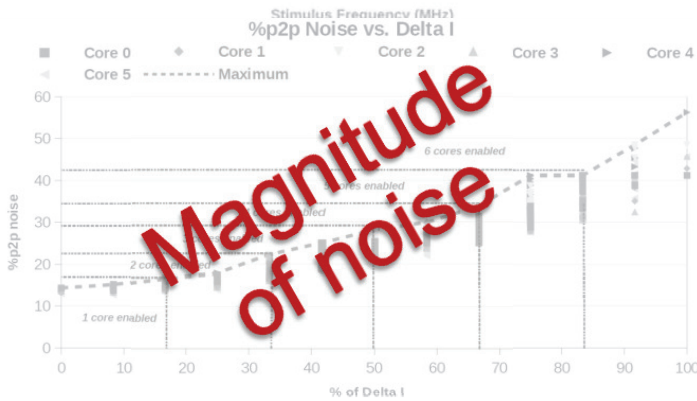
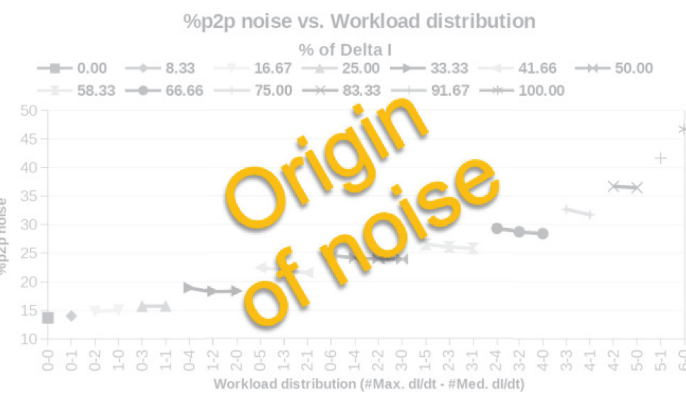
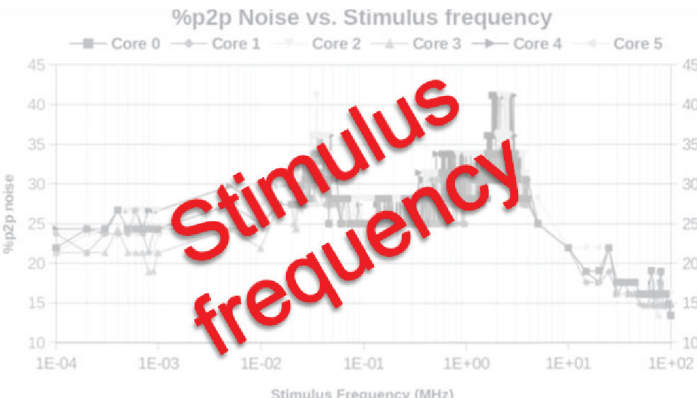
- Pre-silicon characterization is inadequate
- A systematic, direct measurement-based characterization is required

**Our work:** Systematic noise characterization and exploration of optimization opportunities on **zEC12**



*Full presentation: Session 4B Managing Voltage and Time  
Tomorrow at 10:45AM - Umney Theatre & Lounge room*

# Characterization of noise



**Full presentation: Session 4B Managing Voltage and Time Tomorrow at 10:45AM - Umney Theatre & Lounge room**

# Do you want to understand voltage noise?

When:

**Tomorrow**, Tuesday 16<sup>th</sup>, Dec 2014

at **10:45AM**

Where:

**Umney Theatre & Lounge**

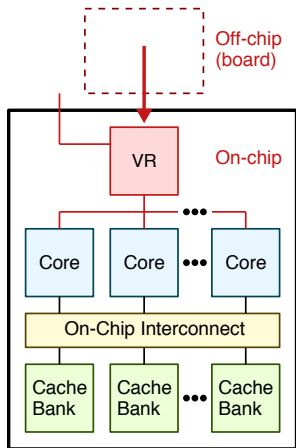
Session 4B - Managing Voltage and Time

*Please come to our poster session too!*

*next paper*

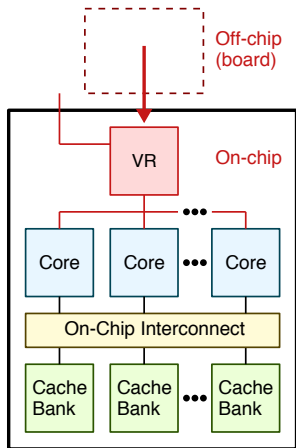
# Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks

Waclaw Godycki, **Christopher Torng**, Alyssa Apsel, Christopher Batten



# Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks

Waclaw Godycki, **Christopher Torng**, Alyssa Apsel, Christopher Batten



## Benefits of Integrated Voltage Regulation

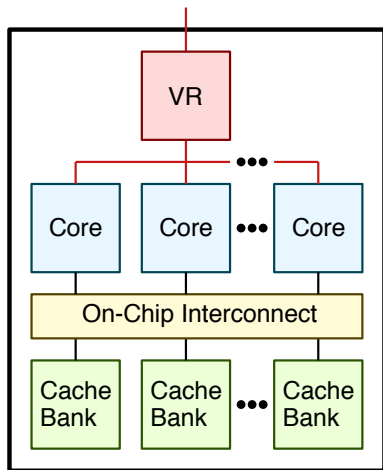
- ▶ Reduced system cost
- ▶ Potential for fine-grain voltage scaling (per-core,  $\mu\text{s}$ -scale voltage transitions)

## Architecture and Analog Circuit Co-Design Challenges

- ▶ Enabling per-core voltage regulation while mitigating large regulator area overhead
- ▶ Choosing useful voltage levels
- ▶ Providing fast voltage transition times

# Simple On-Chip Power Distribution Network

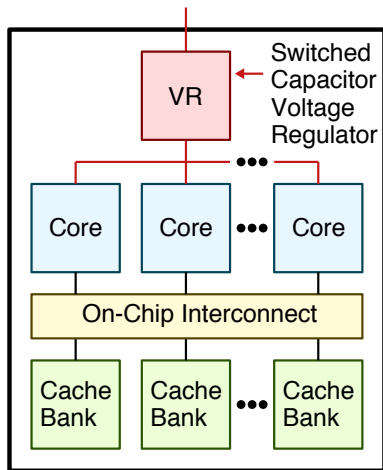
## Single Fixed Voltage Regulator





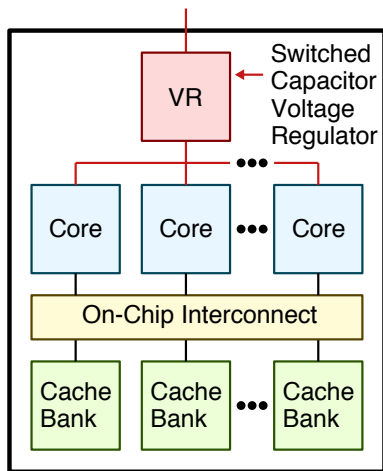
# Simple On-Chip Power Distribution Network

## Single Fixed Voltage Regulator

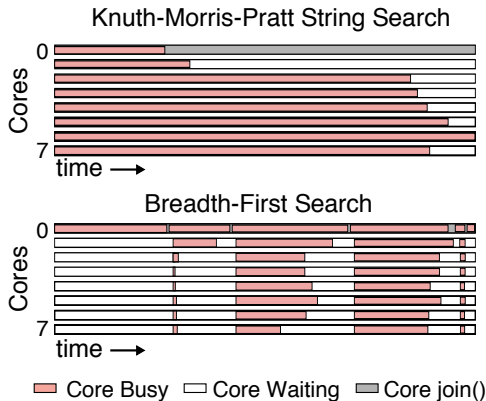


# Simple On-Chip Power Distribution Network

## Single Fixed Voltage Regulator

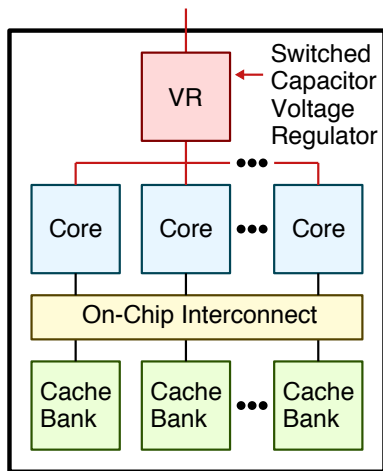


## Length of Fine-Grain Activity Intervals

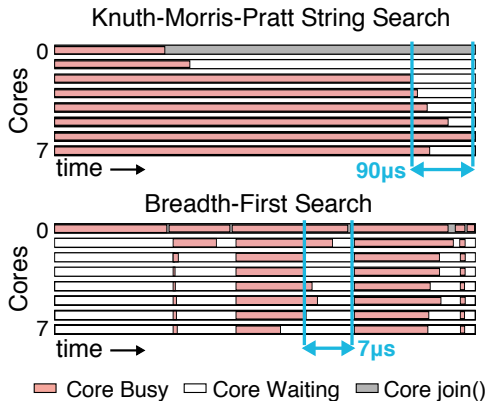


# Simple On-Chip Power Distribution Network

## Single Fixed Voltage Regulator

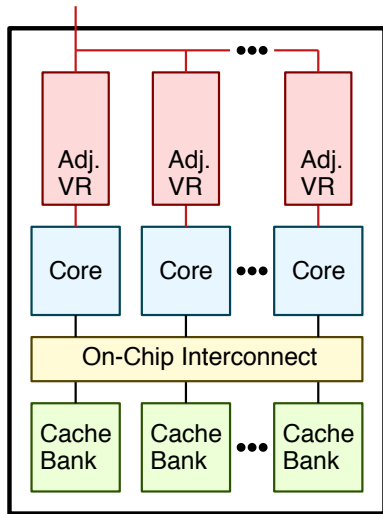


## Length of Fine-Grain Activity Intervals

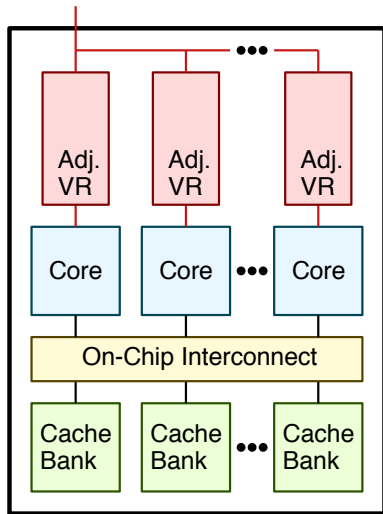


Off-chip transition times  $\sim$  ms  
 On-chip transition times  $\sim$   $\mu\text{s}$

# Multiple Adjustable Voltage Regulators Approach



# Multiple Adjustable Voltage Regulators Approach



Rest  
Mode  
(0.6V)



Nominal  
Mode  
(1.0V)



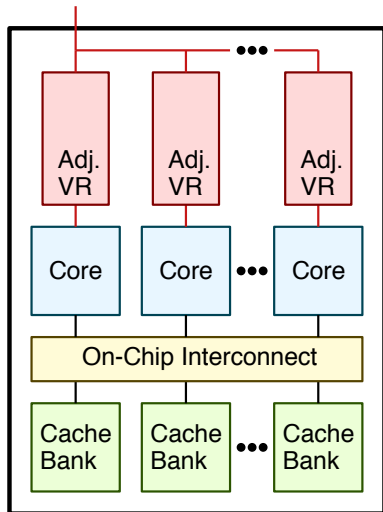
Sprint  
Mode  
(1.15V)



Super-Sprint  
Mode  
(1.33V)

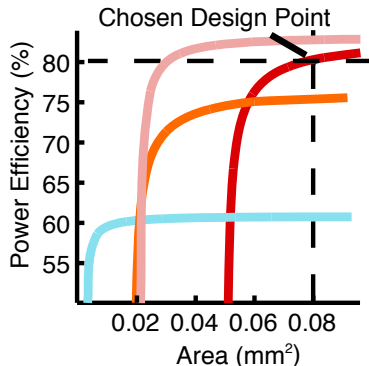


# Multiple Adjustable Voltage Regulators Approach

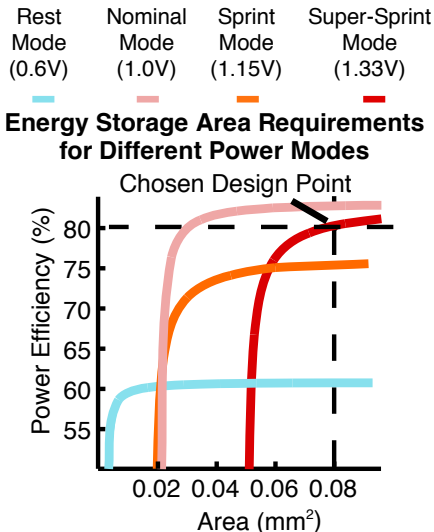
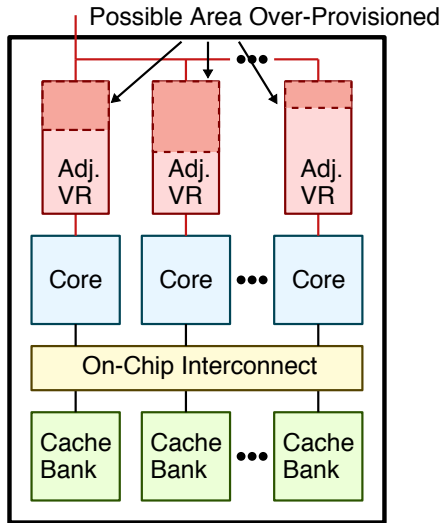


Rest Mode (0.6V)	Nominal Mode (1.0V)	Sprint Mode (1.15V)	Super-Sprint Mode (1.33V)
---------------------	------------------------	------------------------	------------------------------

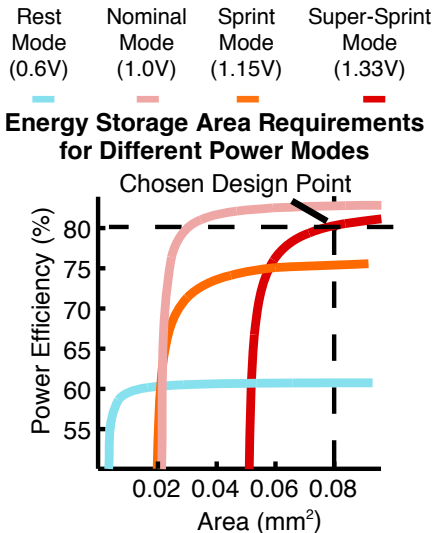
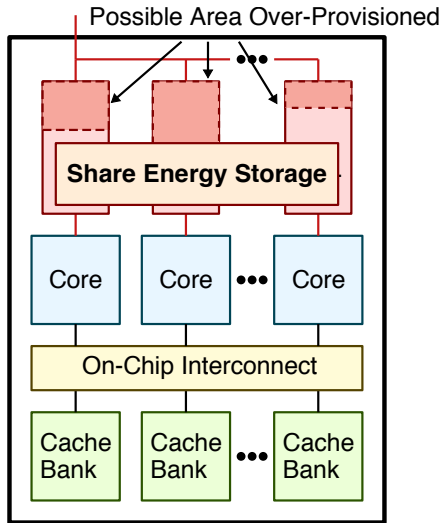
**Energy Storage Area Requirements for Different Power Modes**



# Multiple Adjustable Voltage Regulators Approach



# Multiple Adjustable Voltage Regulators Approach

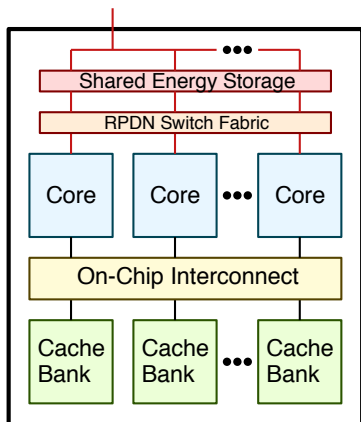




# Reconfigurable Power Distribution Networks

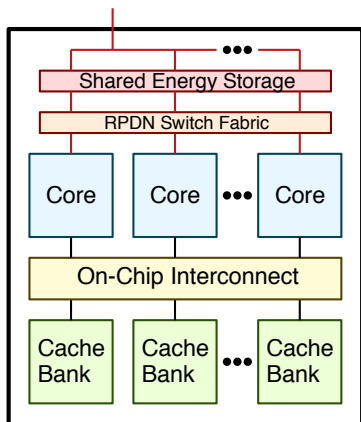
---

# Reconfigurable Power Distribution Networks



Unit cells of shared energy storage are flexibly reconfigured to effectively create multiple differently-sized SC regulators “on-demand”.

# Reconfigurable Power Distribution Networks

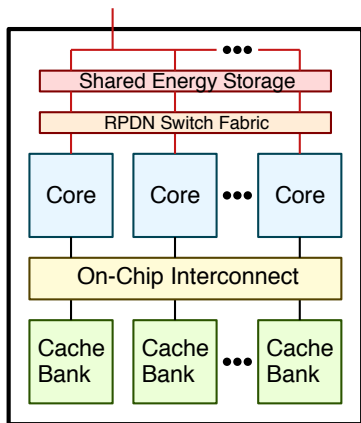


Unit cells of shared energy storage are flexibly reconfigured to effectively create multiple differently-sized SC regulators “on-demand”.

## Benefits of RPDN

- ▶ 10-50% performance and 10-70% energy-efficiency improvement compared to no FGVS
- ▶ 10× faster voltage transition times compared with MAVR ( $\mu\text{s}$  to 100 ns)
- ▶ 40% less area compared to a more traditional per-core regulation scheme

# Reconfigurable Power Distribution Networks



Unit cells of shared energy storage are flexibly reconfigured to effectively create multiple differently-sized SC regulators “on-demand”.

## Benefits of RPDN

- ▶ 10-50% performance and 10-70% energy-efficiency improvement compared to no FGVS
- ▶ 10× faster voltage transition times compared with MAVR ( $\mu\text{s}$  to 100 ns)
- ▶ 40% less area compared to a more traditional per-core regulation scheme

*To hear more, come to the talk on Tuesday!*

*next paper*

# Micro-sliced Virtual Processors

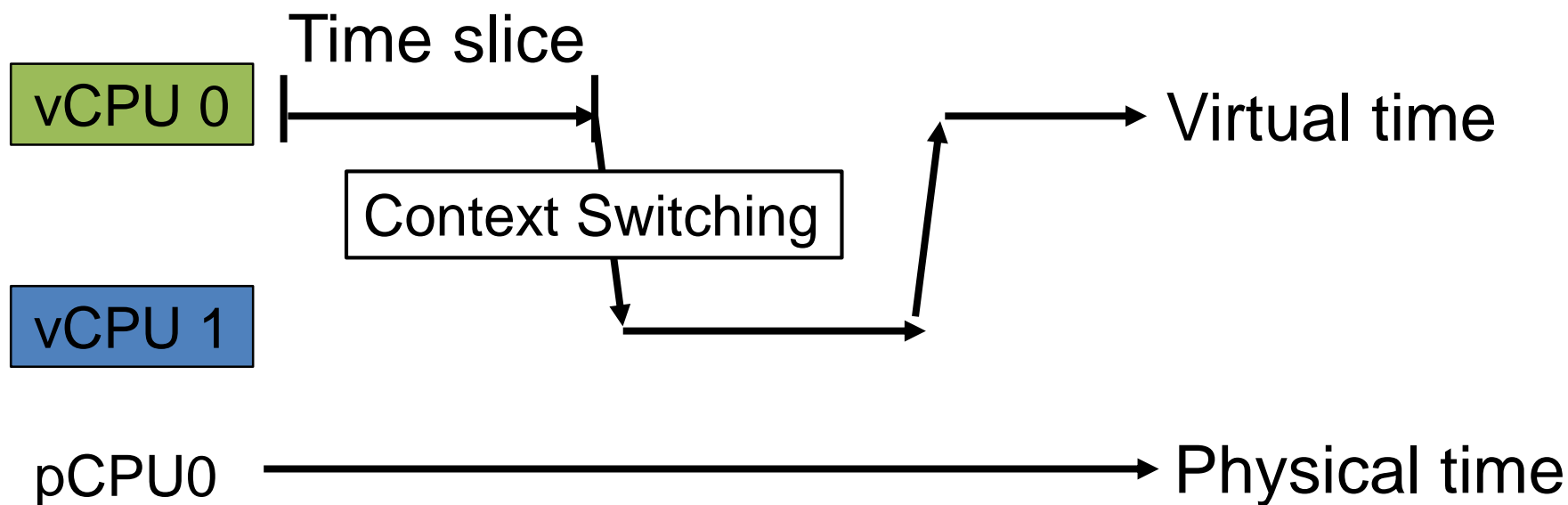
to Hide the Effect of Discontinuous CPU Availability  
for Consolidated Systems

**Jeongseob Ahn**, Chang Hyun Park, and Jaehyuk Huh

Computer Science Department  
KAIST

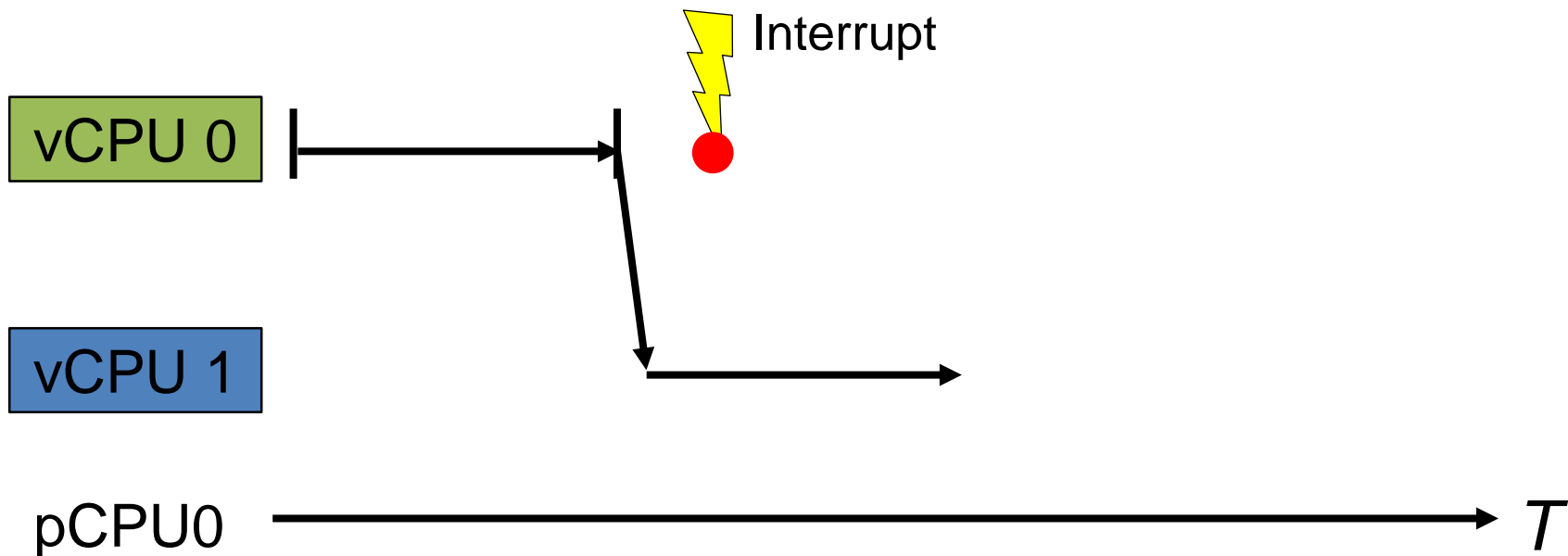
Tomorrow, Session 4B, paper3 @ Umney Theatre

# Virtual Time Discontinuity



Virtual CPUs are not always running

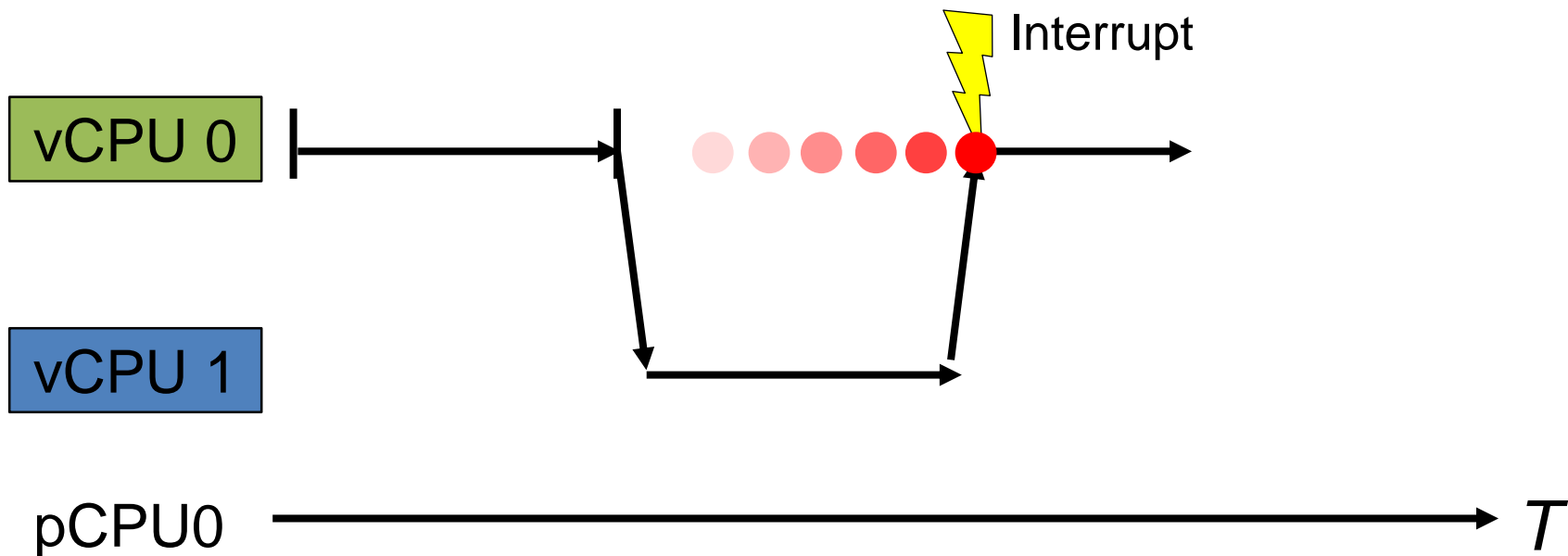
# Interrupt with Virtualization



Interrupt occurs on vCPU0

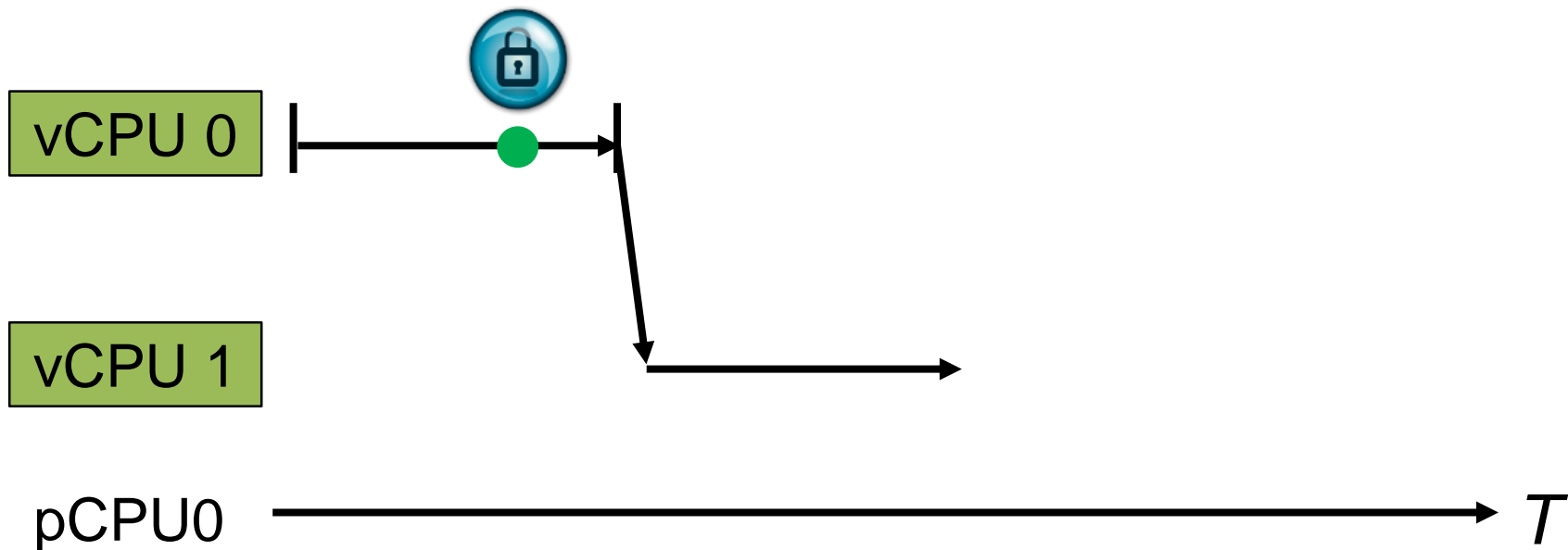


# Interrupt with Virtualization



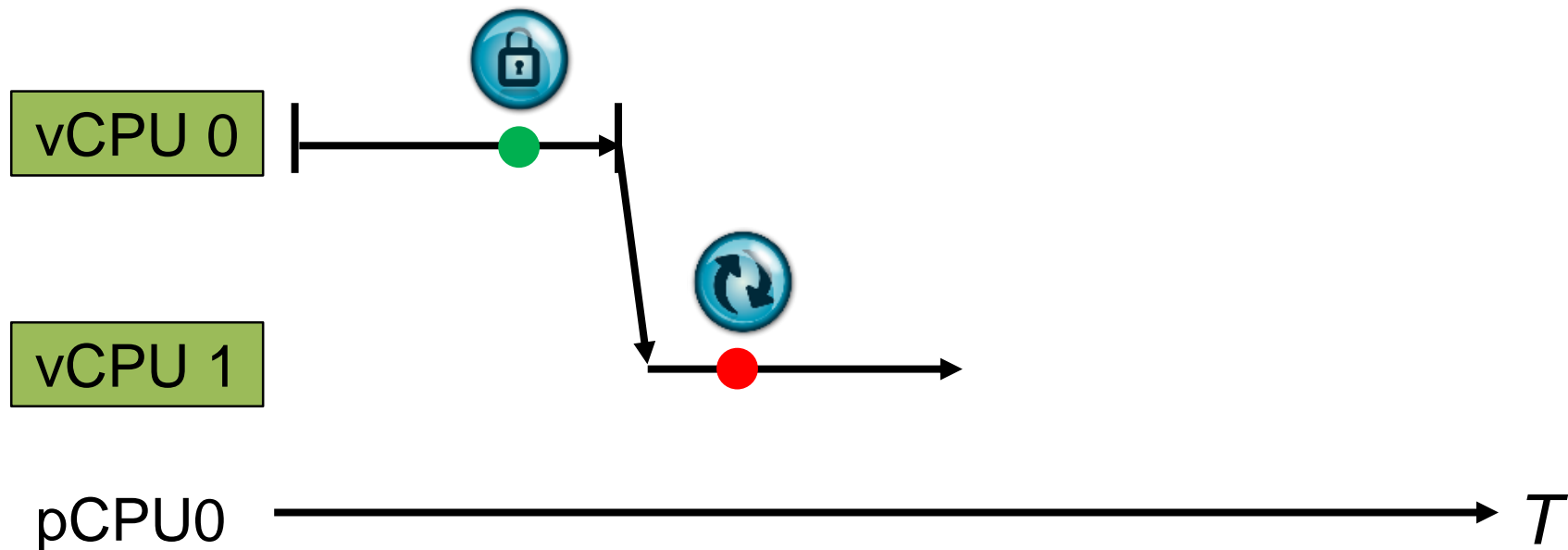
Interrupt processing is delayed

# Spinlock with Virtualization



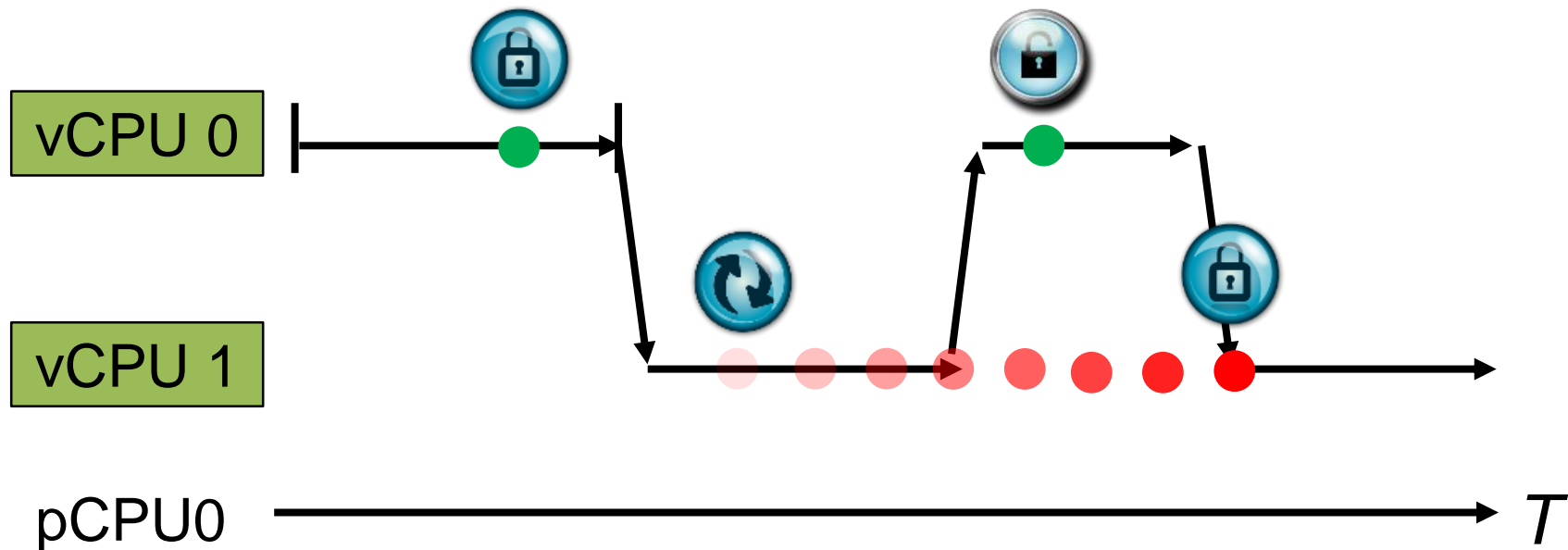
vCPU0 holding a lock is preempted

# Spinlock with Virtualization



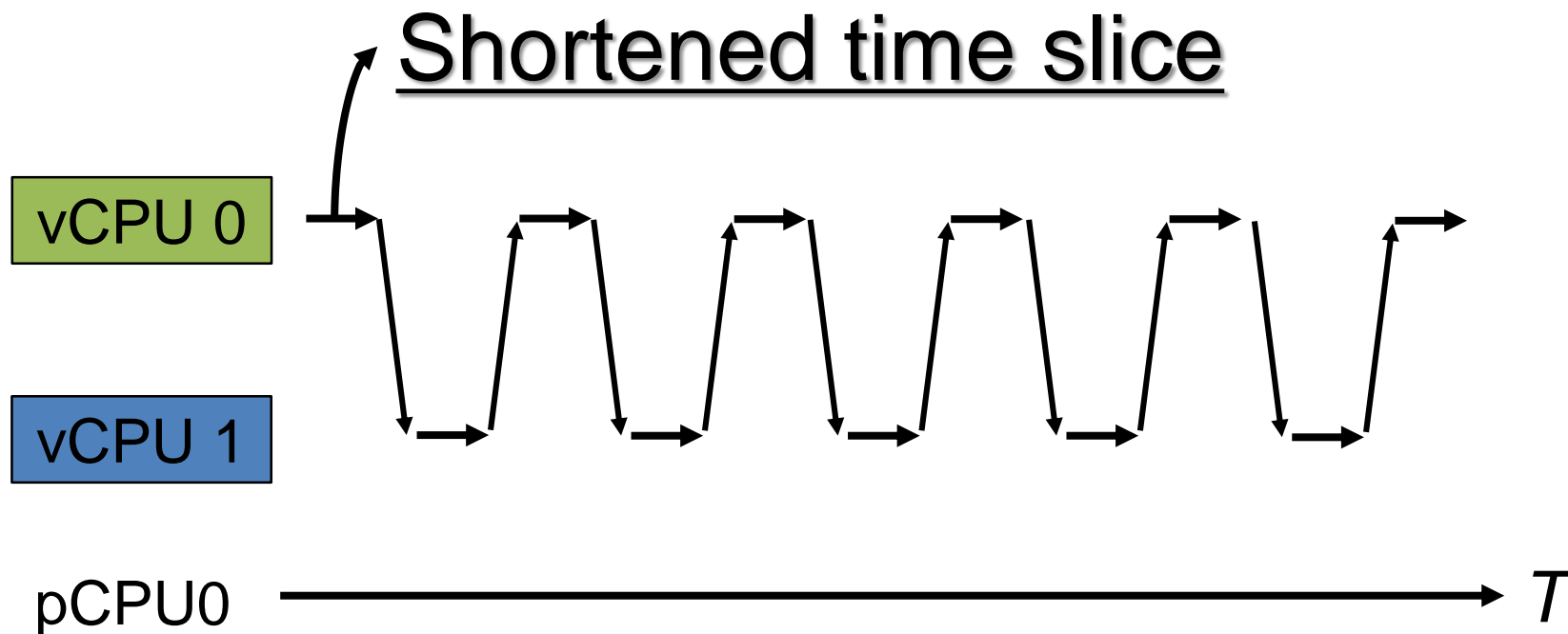
vCPU1 starts spinning to acquire the lock

# Spinlock with Virtualization



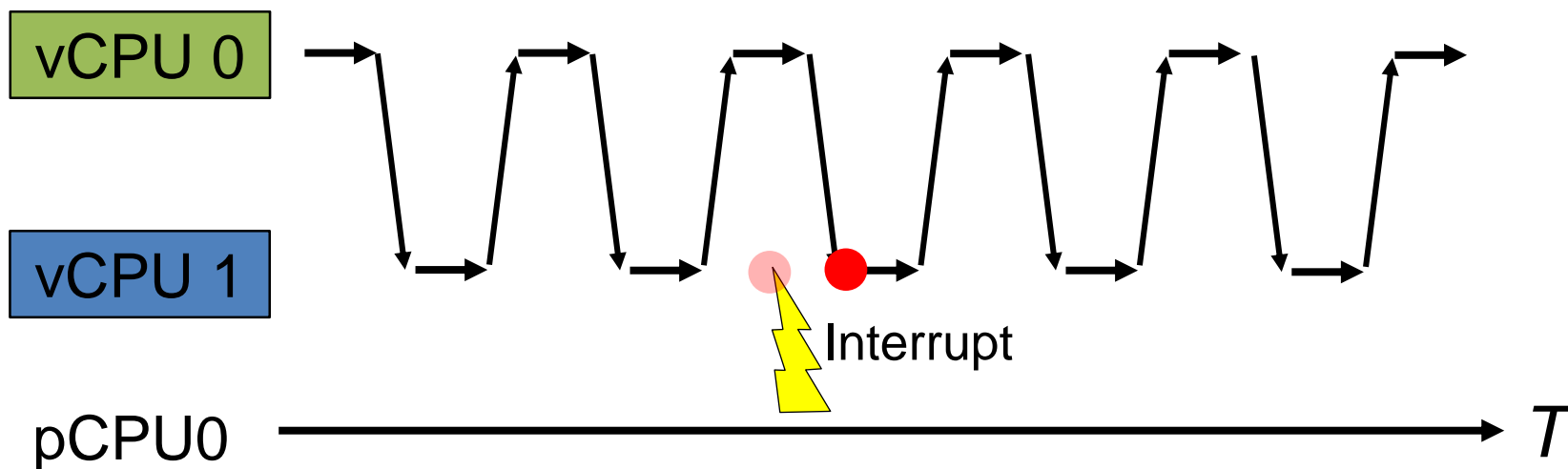
Lock acquiring is delayed

# Toward Virtual Time Continuity



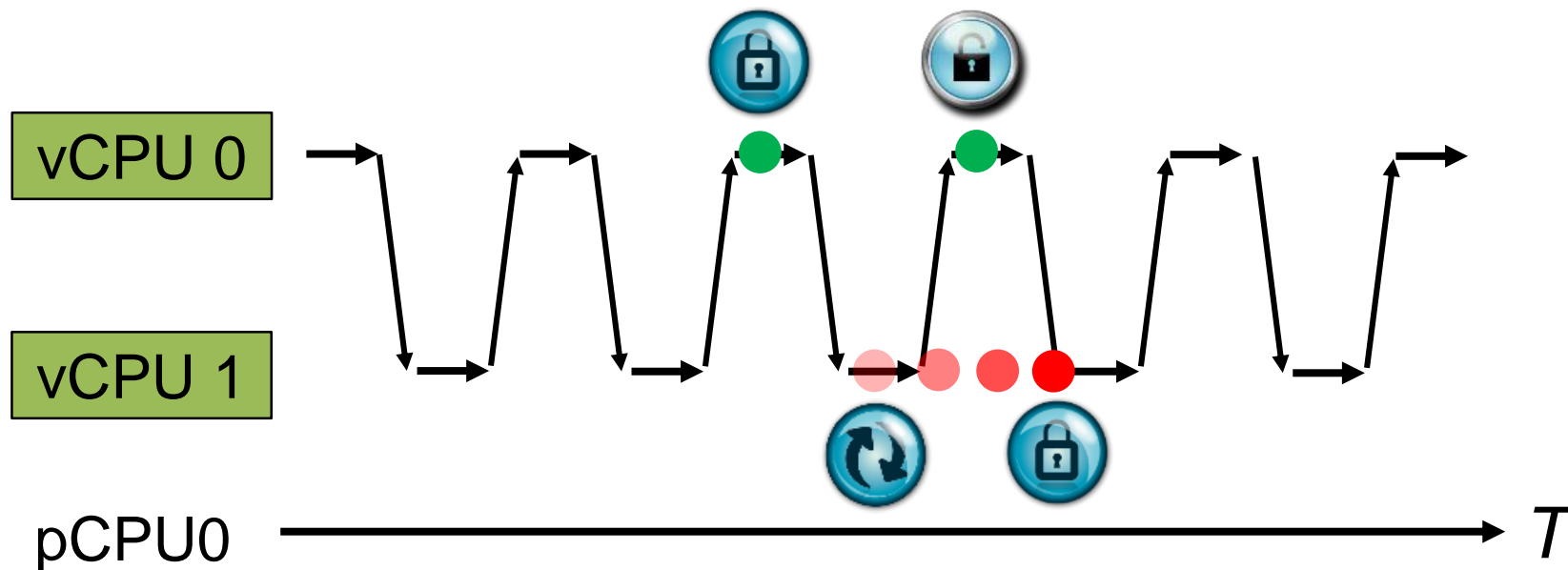
Short, but more frequent runs

# Toward Virtual Time Continuity



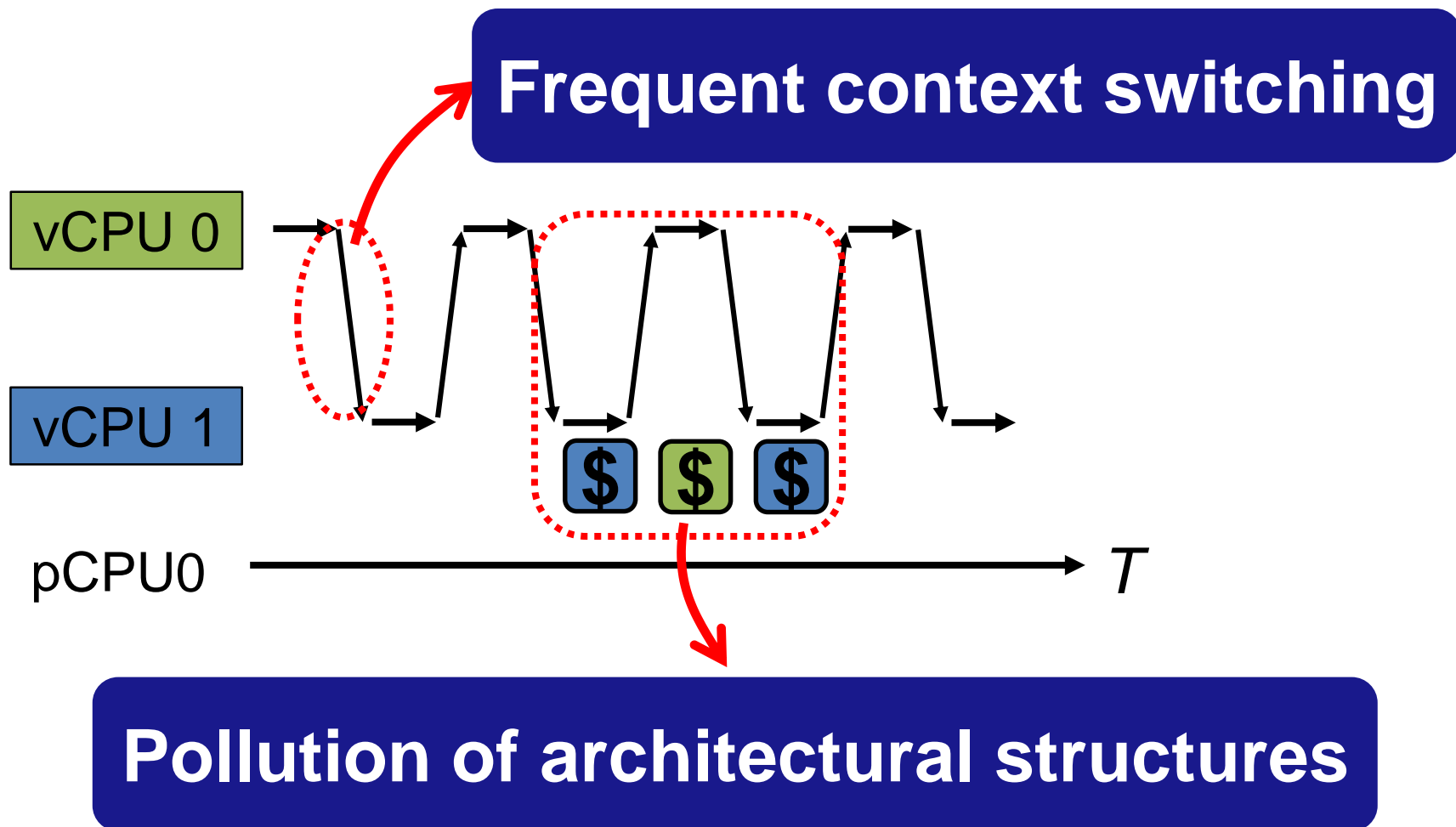
Interrupt is handled sooner

# Toward Virtual Time Continuity



Lock is acquired sooner

# Overheads of Short Time Slice





*next paper*

# SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers

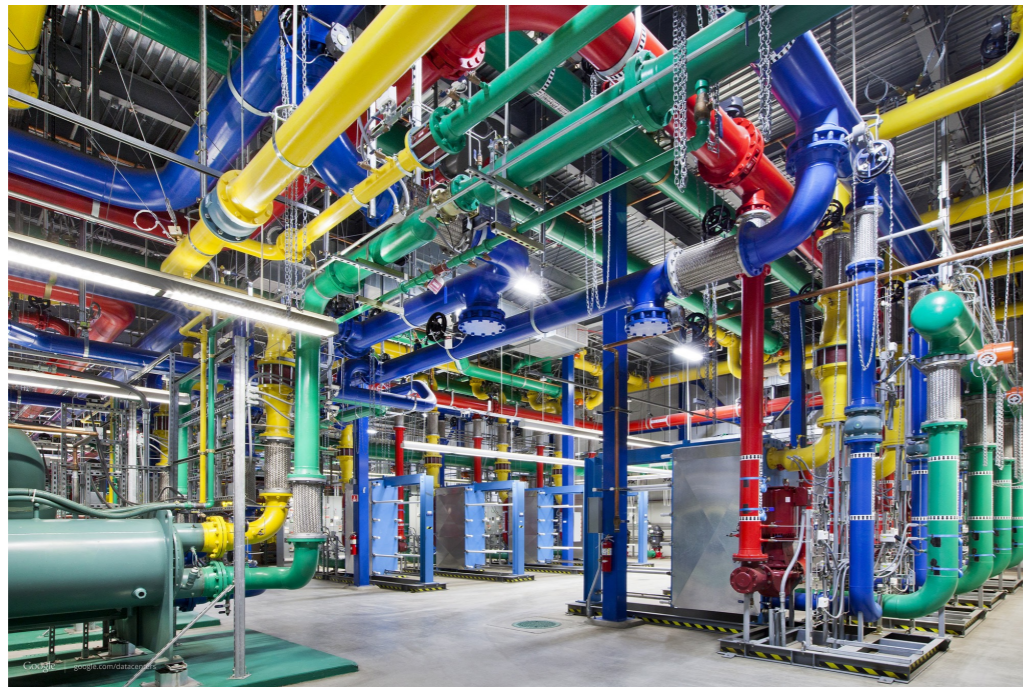
---

Yunqi Zhang, Michael A. Laurenzano, Jason Mars, Lingjia Tang

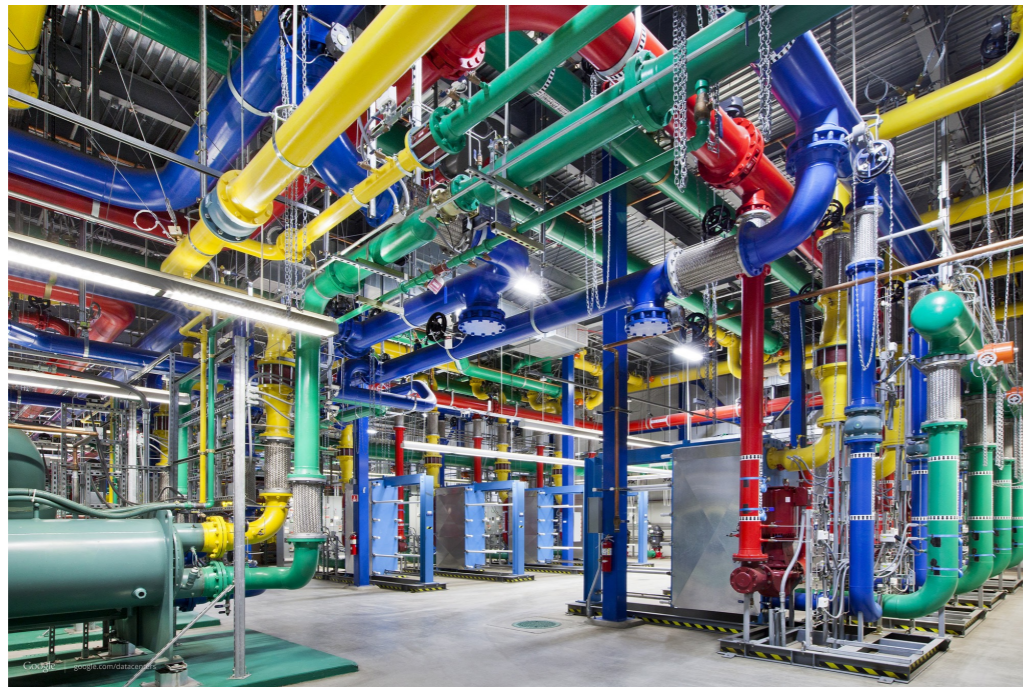


Clarity-Lab  
Electrical Engineering and Computer Science  
University of Michigan

# Data centers are expensive

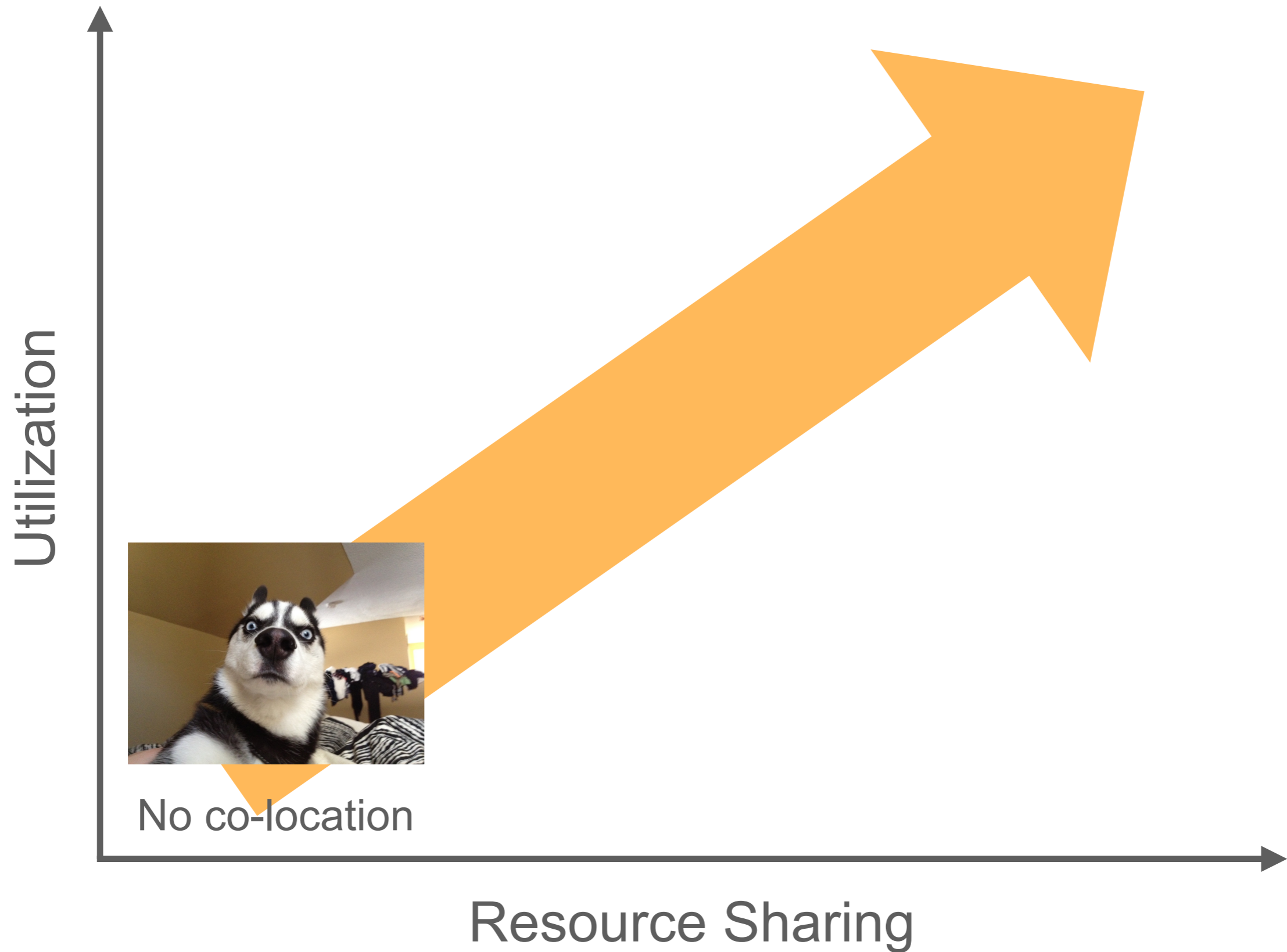


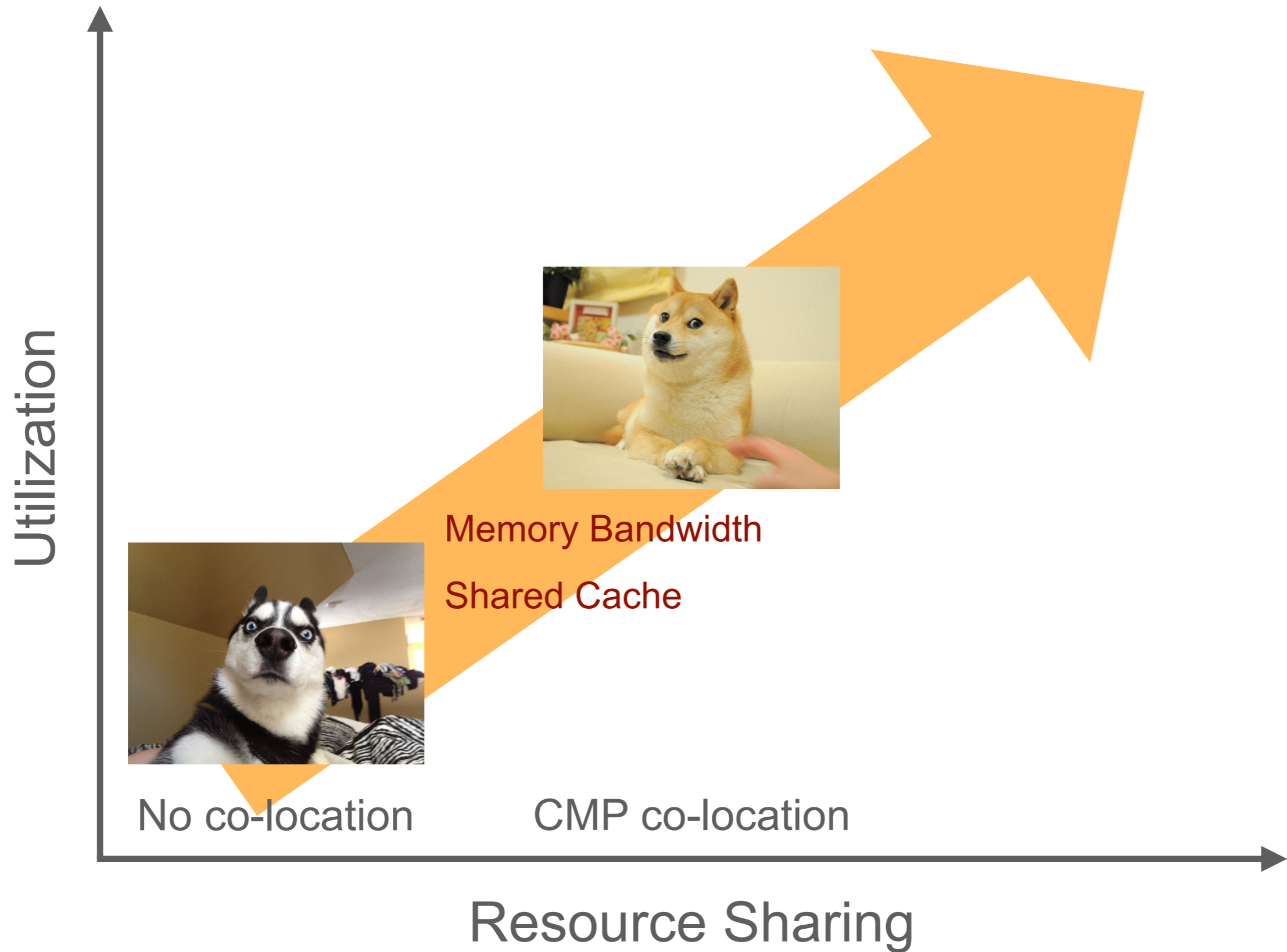
Data centers are expensive

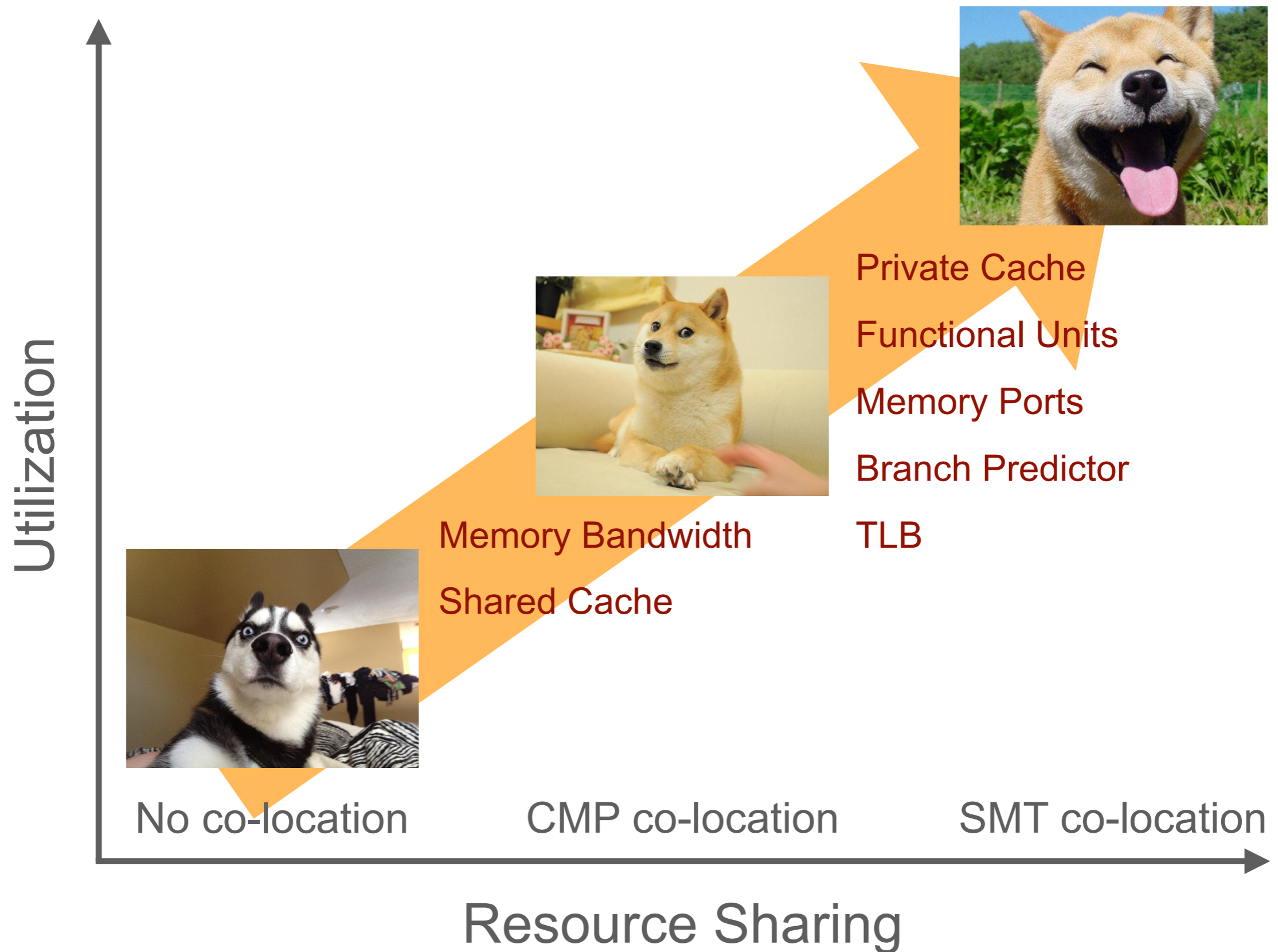


**30%**

Low utilization leads to inefficiency









**< 2%**

**Prediction Error**

**42%**

**Utilization Improvement**

**Section 5A**  
**Wednesday at 9:50 A.M.**



*next paper*



---

# A Front-end Execution Architecture for High Energy Efficiency

---



Ryota Shioya<sup>\*</sup>, Masahiro Goshima<sup>+</sup>, and Hideki Ando<sup>\*</sup>

<sup>\*</sup> Nagoya University

<sup>+</sup> National Institute of Informatics

---

# Front-end Execution Architecture (FXA)

## ■ Background:

- ◆ OoO superscalar processors are fast, but their energy efficiency is low
- ◆ Even in heterogeneous architecture, big cores still consume a large amount of energy

## ■ Goal:

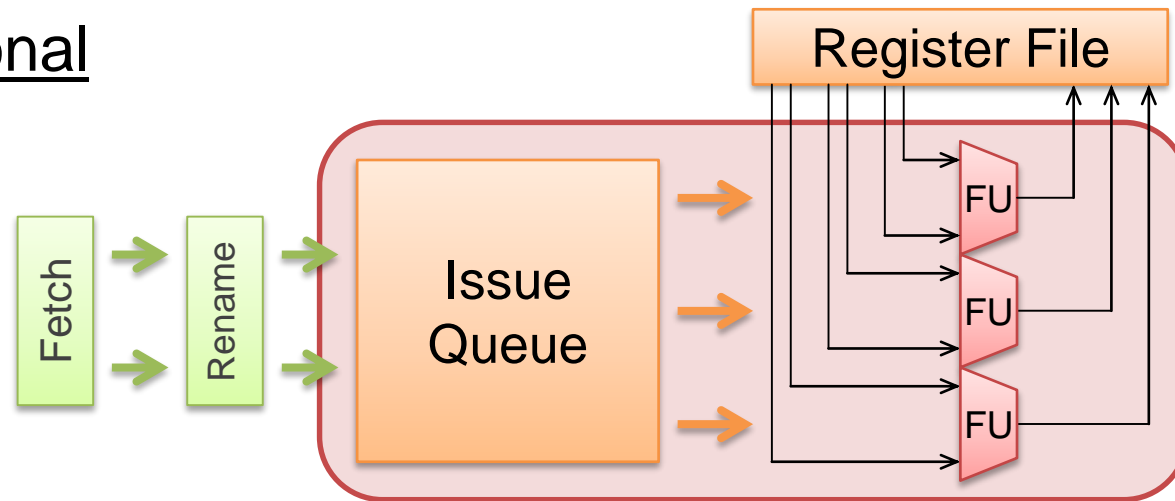
- ◆ Improving the energy efficiency of OoO SSPs

## ■ Approach:

- ◆ Execute instructions in-order in a front-end

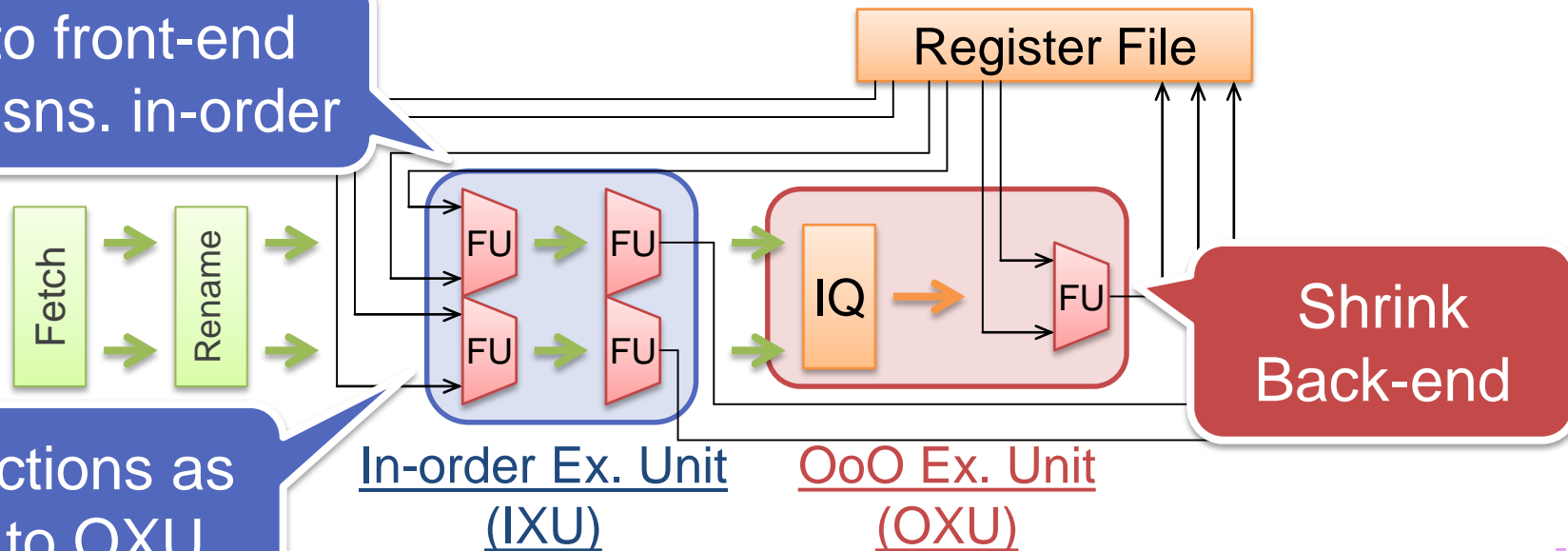
# FXA has two execution units: IXU and OXU

## Conventional



## FXA

Add FUs to front-end  
Executes insns. in-order



IXU functions as  
a filter to OXU

Shrink  
Back-end

# FXA's merit: improving energy efficiency

## ■ Merits:

- ◆ FUs can be added to the IXU with low overhead, and it improves performance
- ◆ The IXU can execute over 50% insns in-order
  - The energy-consuming OXU is shrunk

## ■ Compared to Cortex A-57 (big)

- 5.7% higher IPC and 17% lower energy consumption
- 25% higher performance energy ratio (=the inverse of EDP)

*next paper*

# Execution Drafting

Energy Efficiency Through Computation Deduplication

**Michael McKeown**, Jonathan Balkind, David Wentzlaff  
Princeton University

MICRO-47

**Wednesday, December 17, 2014 – Session 5A – 9:50am**



**PRINCETON**  
UNIVERSITY

**PRINCETON**  
School of Engineering and Applied Science

# Motivation

- Cloud Computing
- Data Center Energy Efficiency



Source:[http://www.google.com/about/datacenters/gallery/images/\\_2000/CBF\\_009.jpg](http://www.google.com/about/datacenters/gallery/images/_2000/CBF_009.jpg)

- Computation Deduplication
  - Commonality in Applications

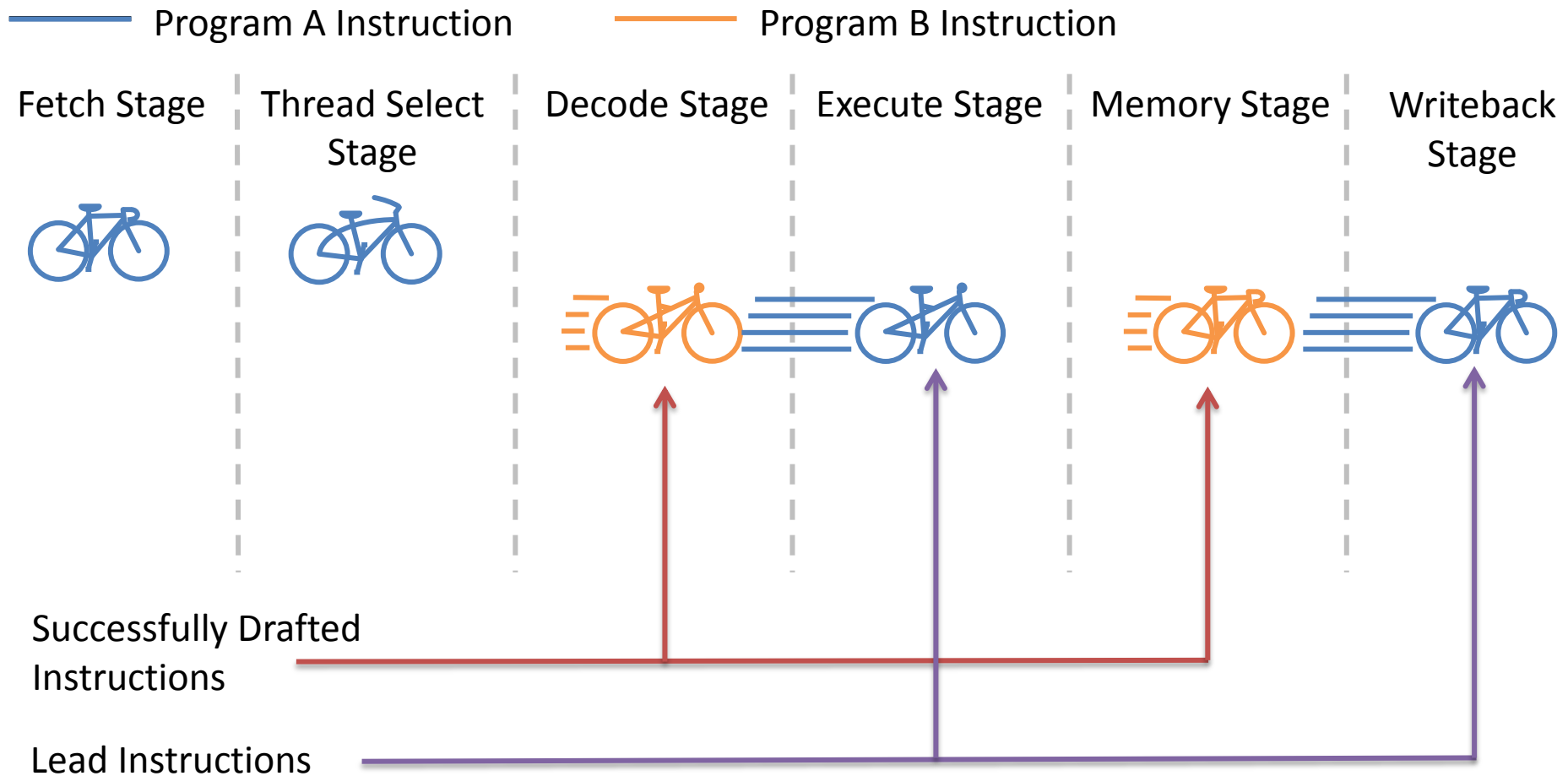




# Execution Drafting

Microarchitectural mechanism to improve energy efficiency in a multithreaded core

**Goal: Maximize**  $\frac{\textit{Performance}}{\textit{Energy}}$



# Main Results

- Up to **20% performance/energy gain**

- Minuscule performance degradation

- Small area overhead

- Potential in drafting more threads

2.2.9_cnn	100	93	100	100	100	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2.2.9_simple	93	100	93	94	94	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2.2.9_ted	100	93	100	100	100	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2.2.9_top500	100	94	100	100	100	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2.2.9_youtube	100	94	100	100	100	7	7	7	7	7	7	7	7	7	7	7	7	7	7
2.4.2_cnn	7	7	7	7	7	100	92	94	95	93	53	53	53	53	53	53	53	53	53
2.4.2_simple	7	7	7	7	7	92	100	93	93	94	53	53	53	53	53	56	56	56	56
2.4.2_ted	7	7	7	7	7	94	93	100	94	93	57	54	54	54	54	57	57	57	57
2.4.2_top500	7	7	7	7	7	95	93	94	100	94	56	54	58	53	56	56	56	56	56
2.4.2_youtube	7	7	7	7	7	93	94	93	94	100	53	54	54	53	54	54	54	54	54
2.4.6_cnn	7	7	7	7	7	53	53	57	56	53	100	94	93	95	93	93	93	93	93
2.4.6_simple	7	7	7	7	7	53	53	54	54	54	94	100	94	95	94	94	94	94	94
2.4.6_ted	7	7	7	7	7	53	53	54	58	54	93	94	100	93	94	94	94	94	94
2.4.6_top500	7	7	7	7	7	53	53	54	53	53	95	95	93	100	93	93	93	93	93
2.4.6_youtube	7	7	7	7	7	53	56	57	56	54	93	94	94	93	100	100	100	100	100

Wednesday, December 17, 2014 – Session 5A – 9:50am

*next paper*

# PPEP: ONLINE PERFORMANCE, POWER, AND ENERGY PREDICTION FRAMEWORK and DVFS Space Exploration

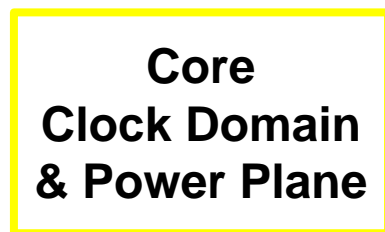


Bo Su<sup>†</sup> Junli Gu<sup>‡</sup> Li Shen<sup>†</sup> Wei Huang<sup>‡</sup> Joseph L. Greathouse<sup>‡</sup> Zhiying Wang<sup>†</sup>

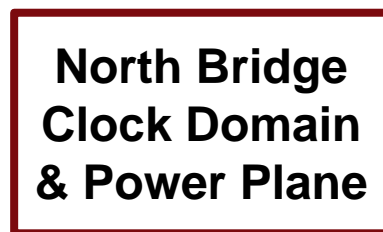
<sup>†</sup>National University of Defense Technology

<sup>‡</sup>AMD Research

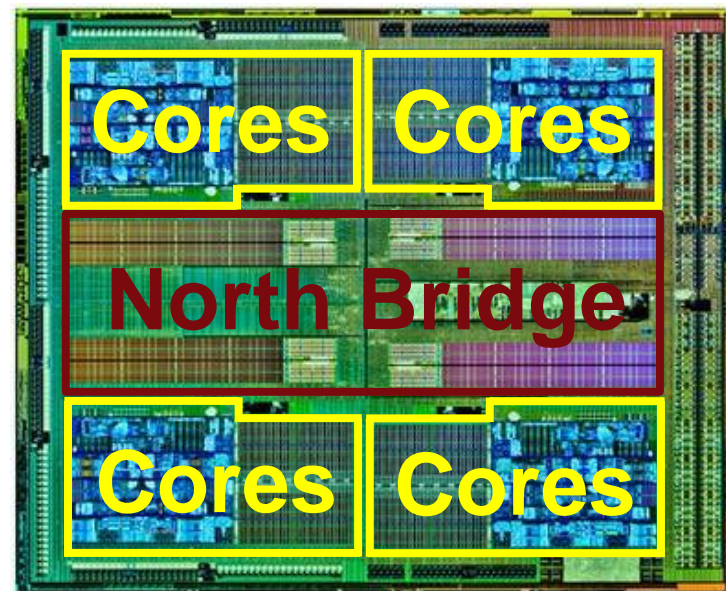
- ▲ Dynamic Voltage & Frequency Scaling Challenge
  - How to predict performance & power across VF states
- ▲ Difficulties on modern processor
  - Multiple clock domains
  - Multiple power planes



Scalable



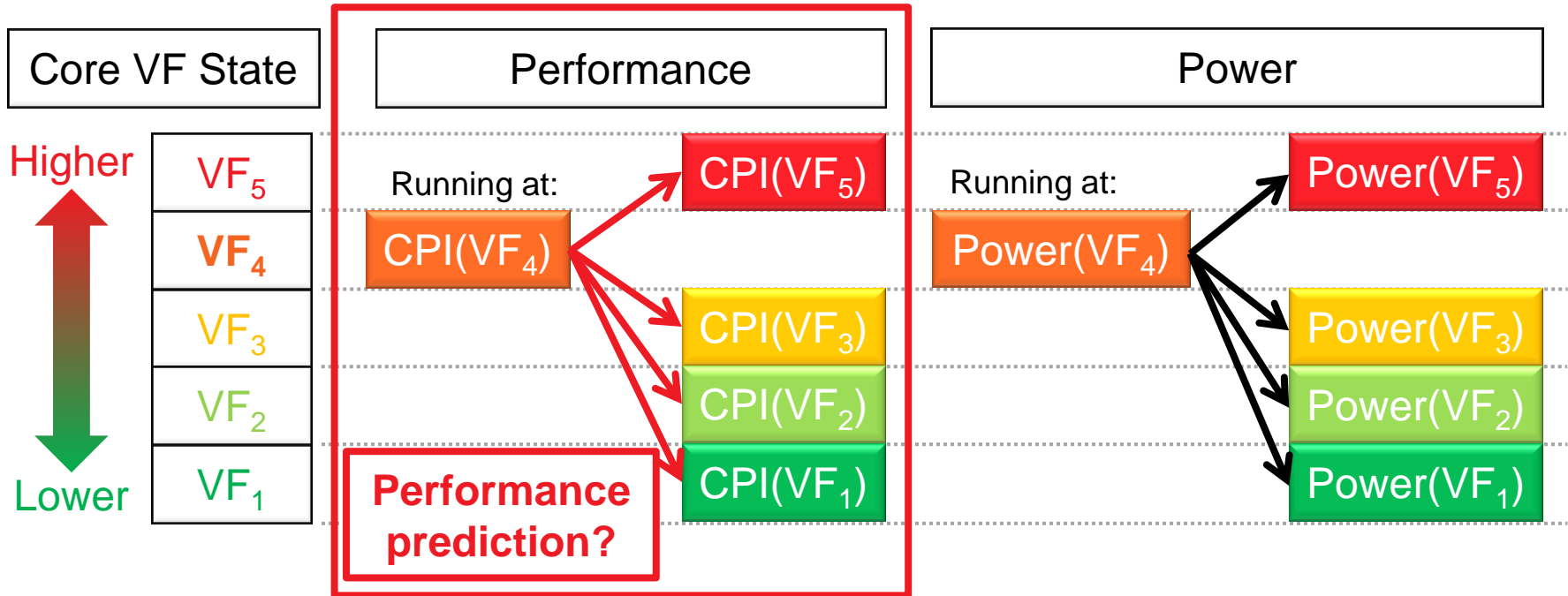
Not scalable



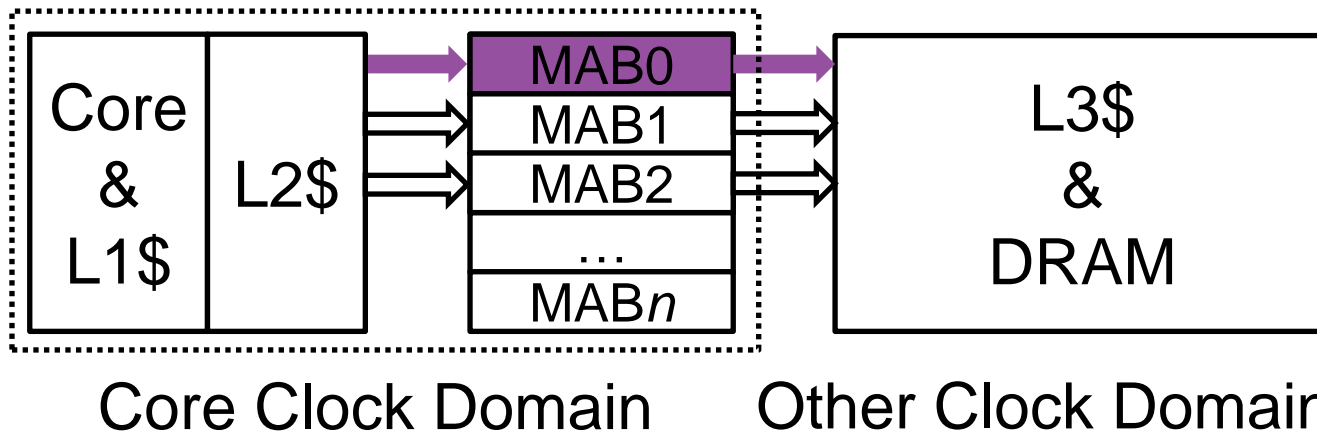
AMD FX-8320



# Q1: PERFORMANCE PREDICTION



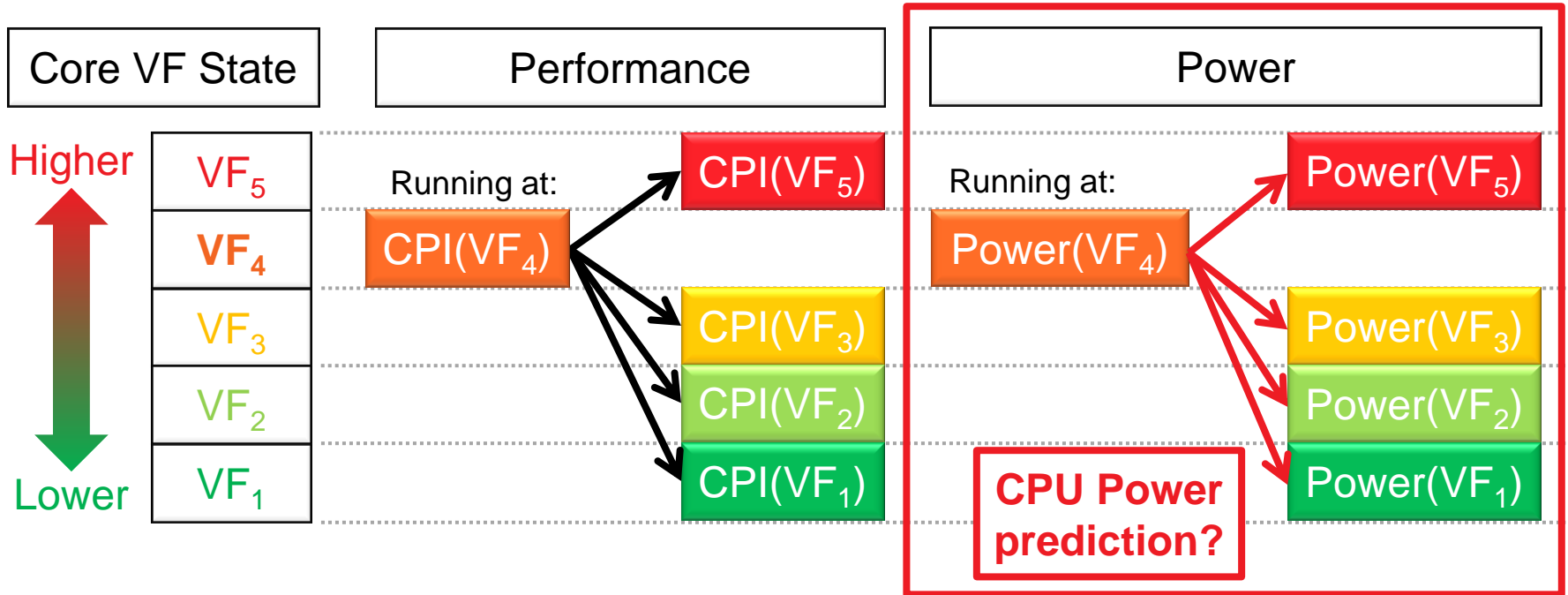
▲ LL-MAB: Miss Address Buffer (MAB) based Leading Loads (LL) CPI Predictor;



3.2% error  
>2X freq. diff.



## Q2: POWER PREDICTION



- ▲ Power model: CPU Events + Temperature
- ▲ Power prediction: LL-MAB + 2 observations of CPU events.
- ▲ 4.2% error across 5 VF states.

Wednesday 11:05AM



*next paper*

# NoC Architectures for Silicon Interposer Systems

Wed: Session 5B

Natalie Enright Jerger\*

Ajaykumar Kannan\*

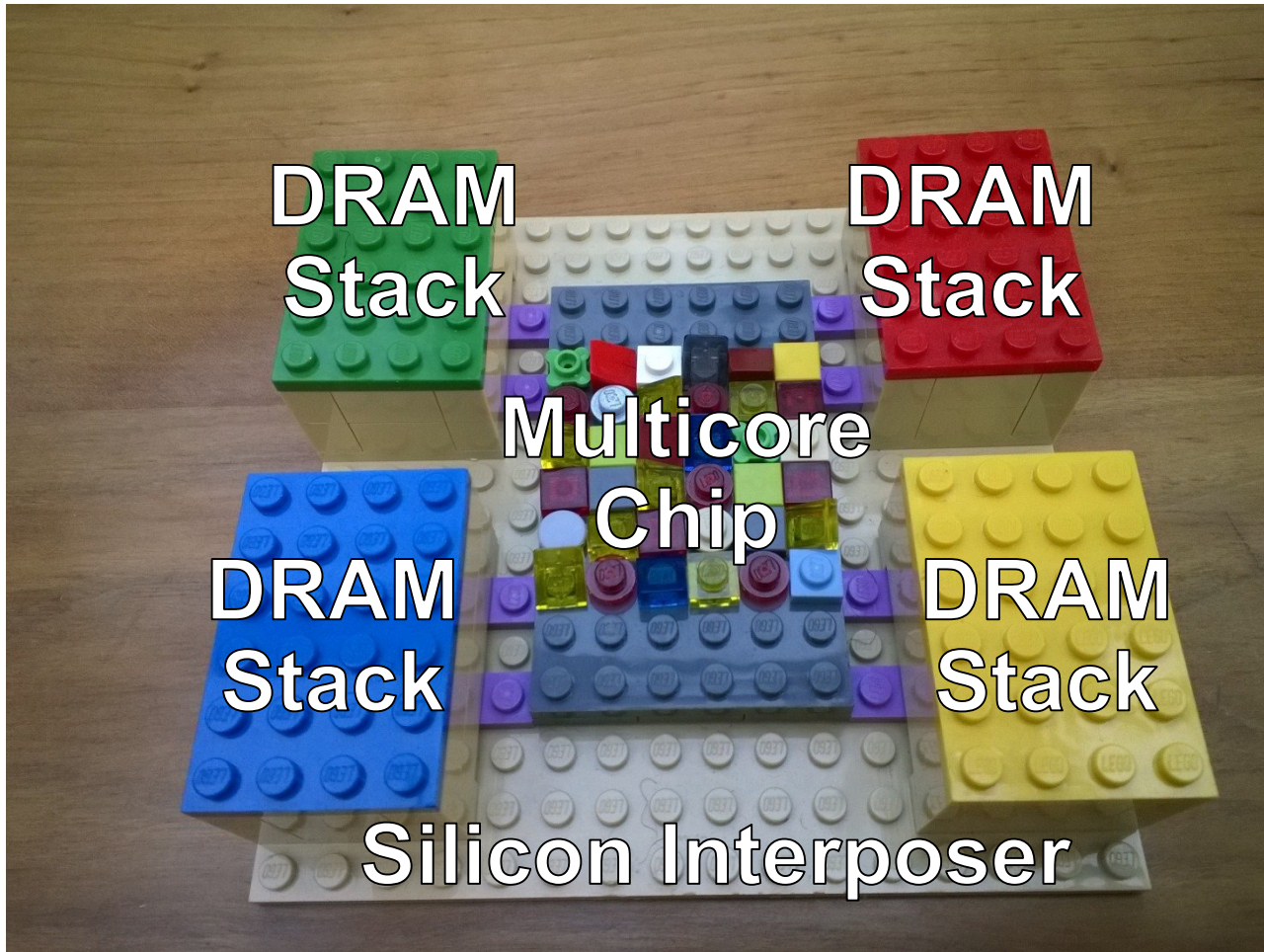
Zimo Li\*

Gabriel H. Loh<sup>+</sup>

\* University of Toronto

<sup>+</sup> AMD Research





DRAM  
Stack

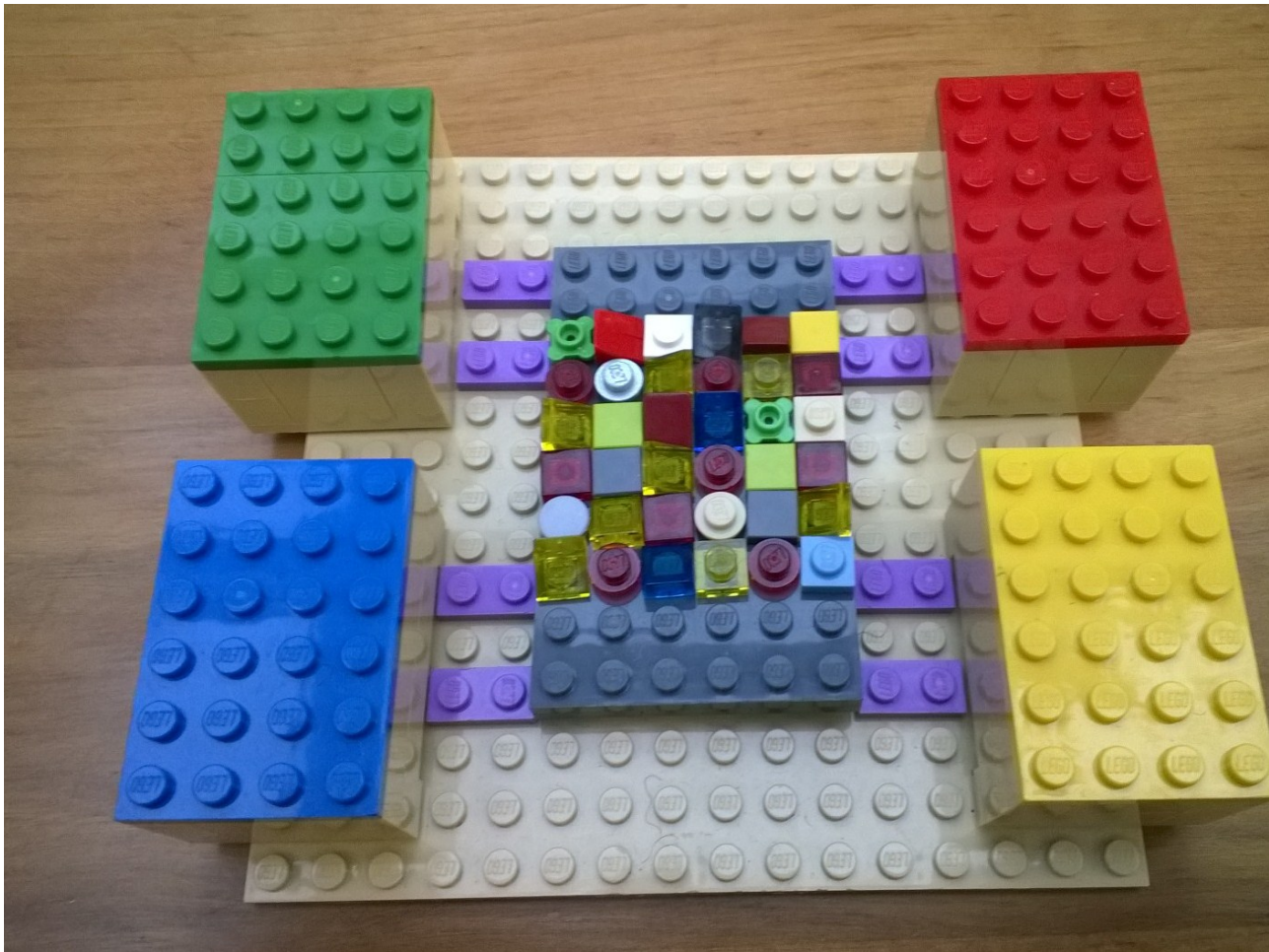
DRAM  
Stack

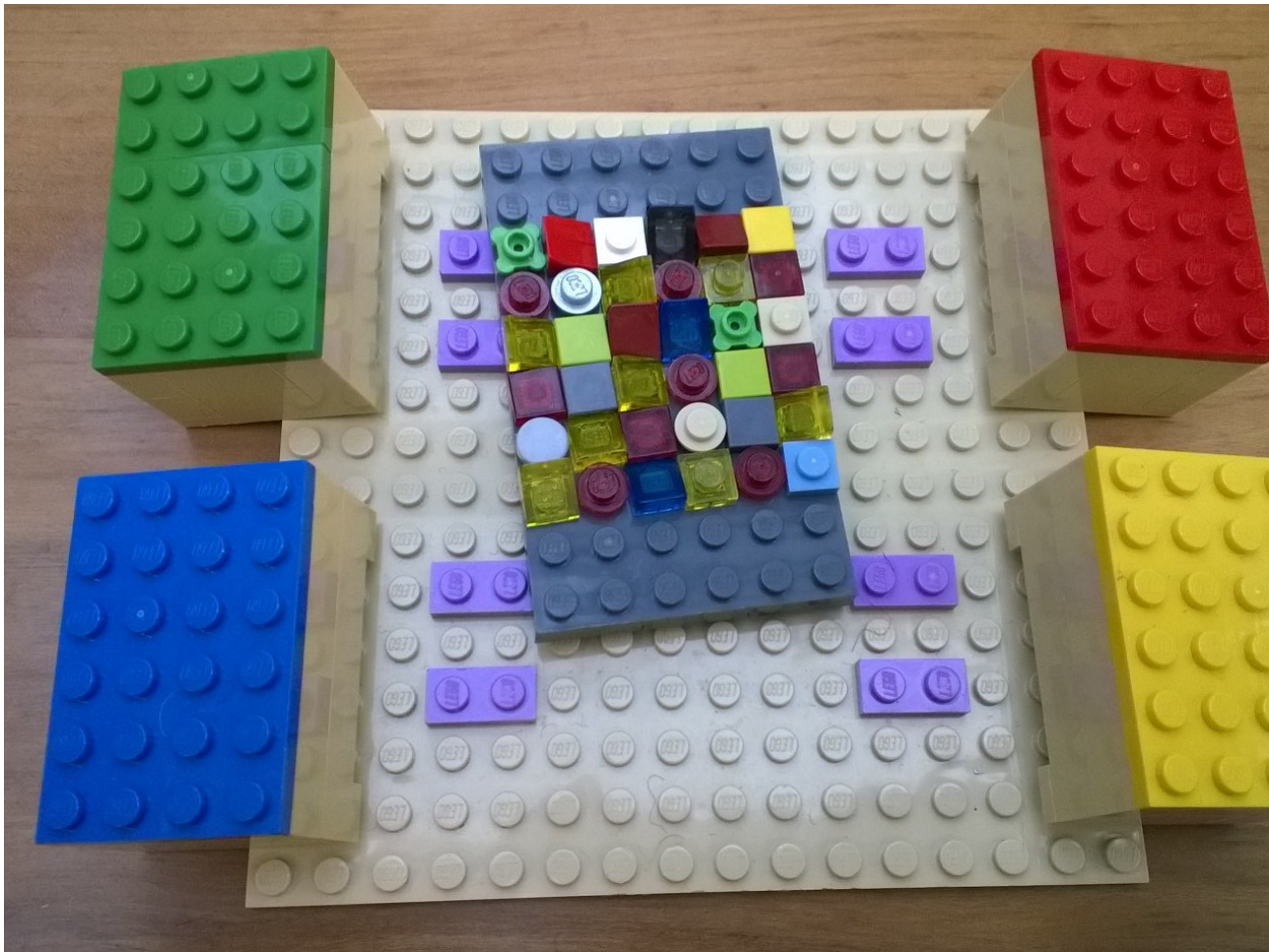
Multicore  
Chip

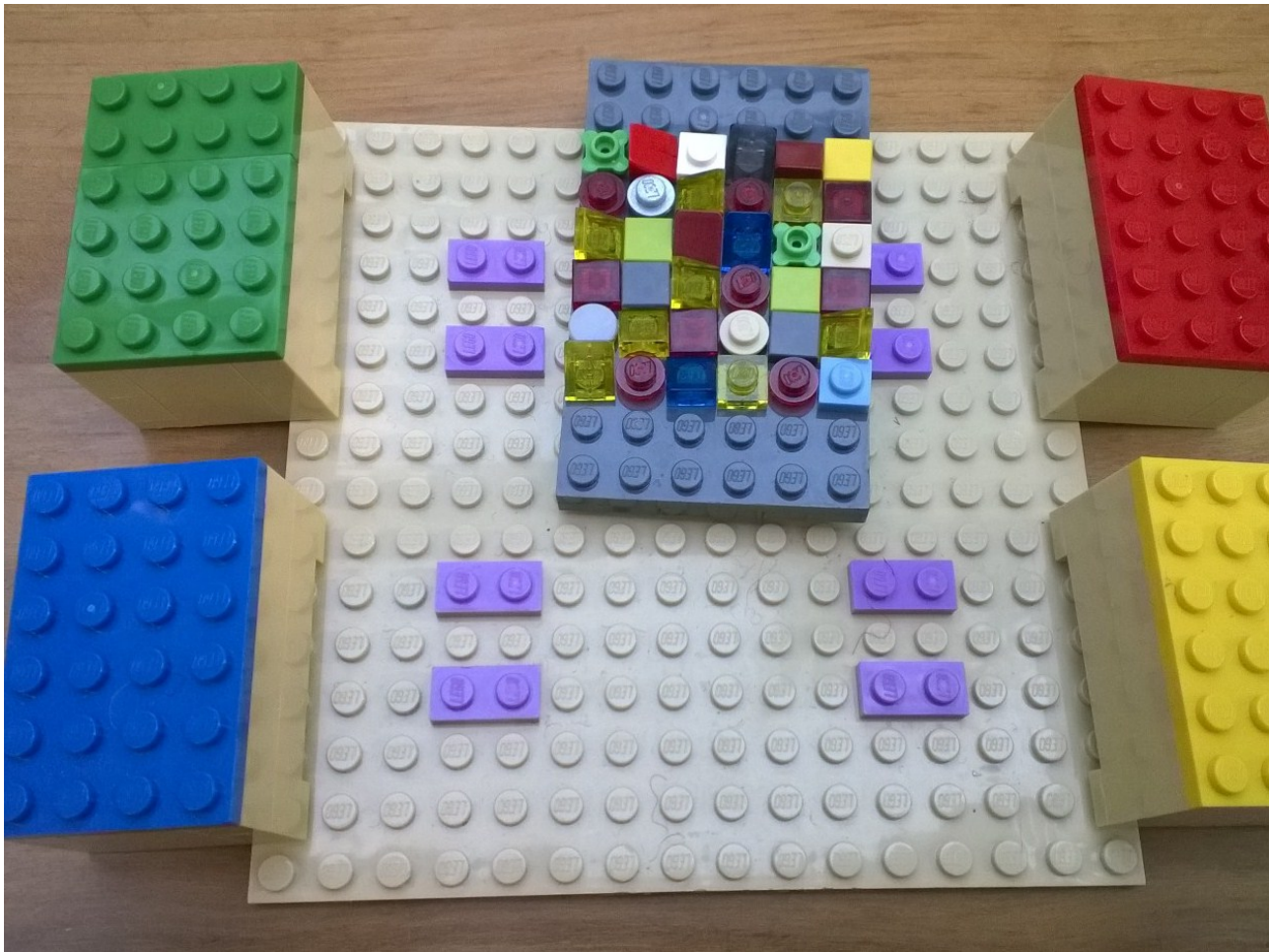
DRAM  
Stack

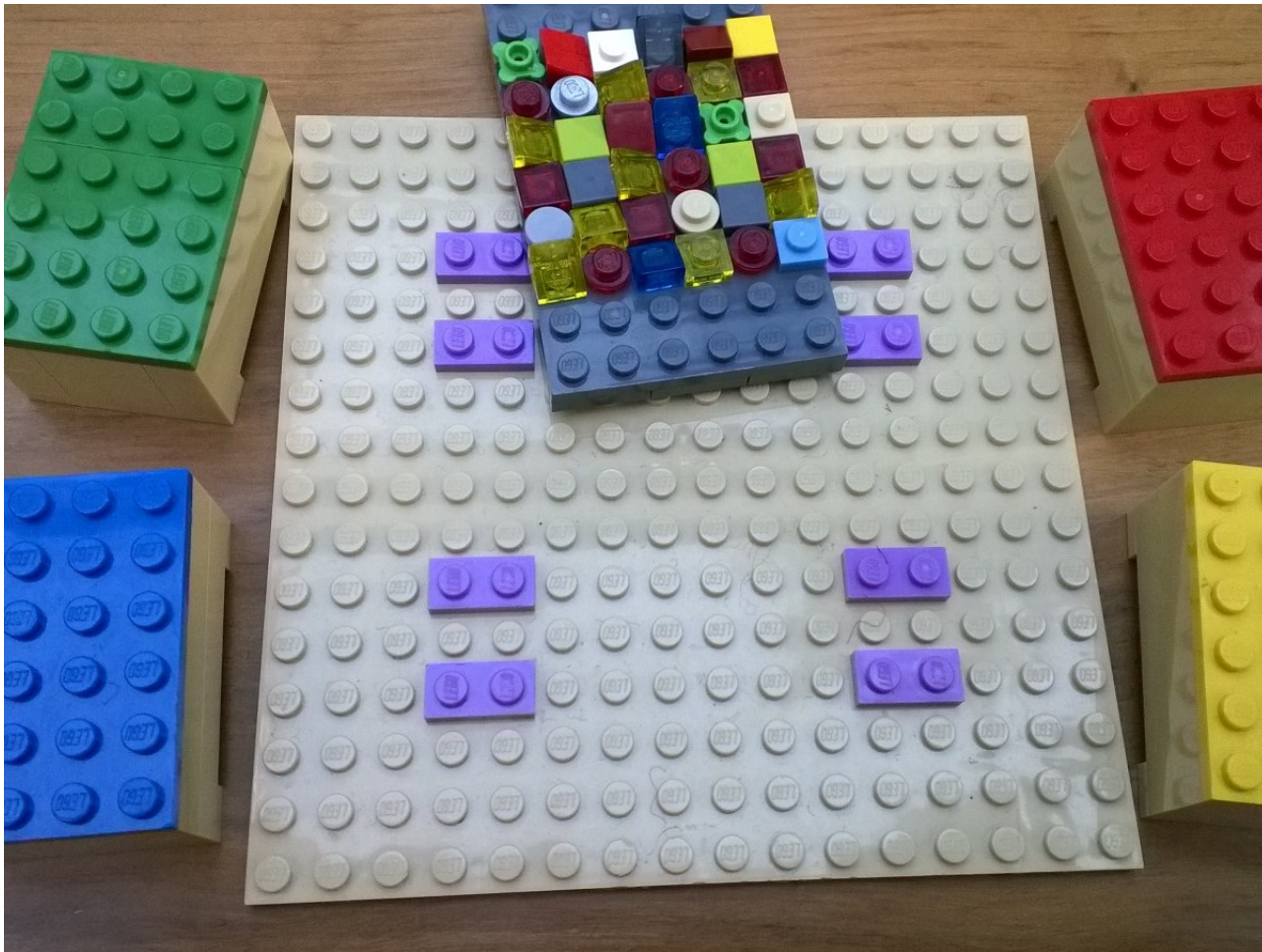
DRAM  
Stack

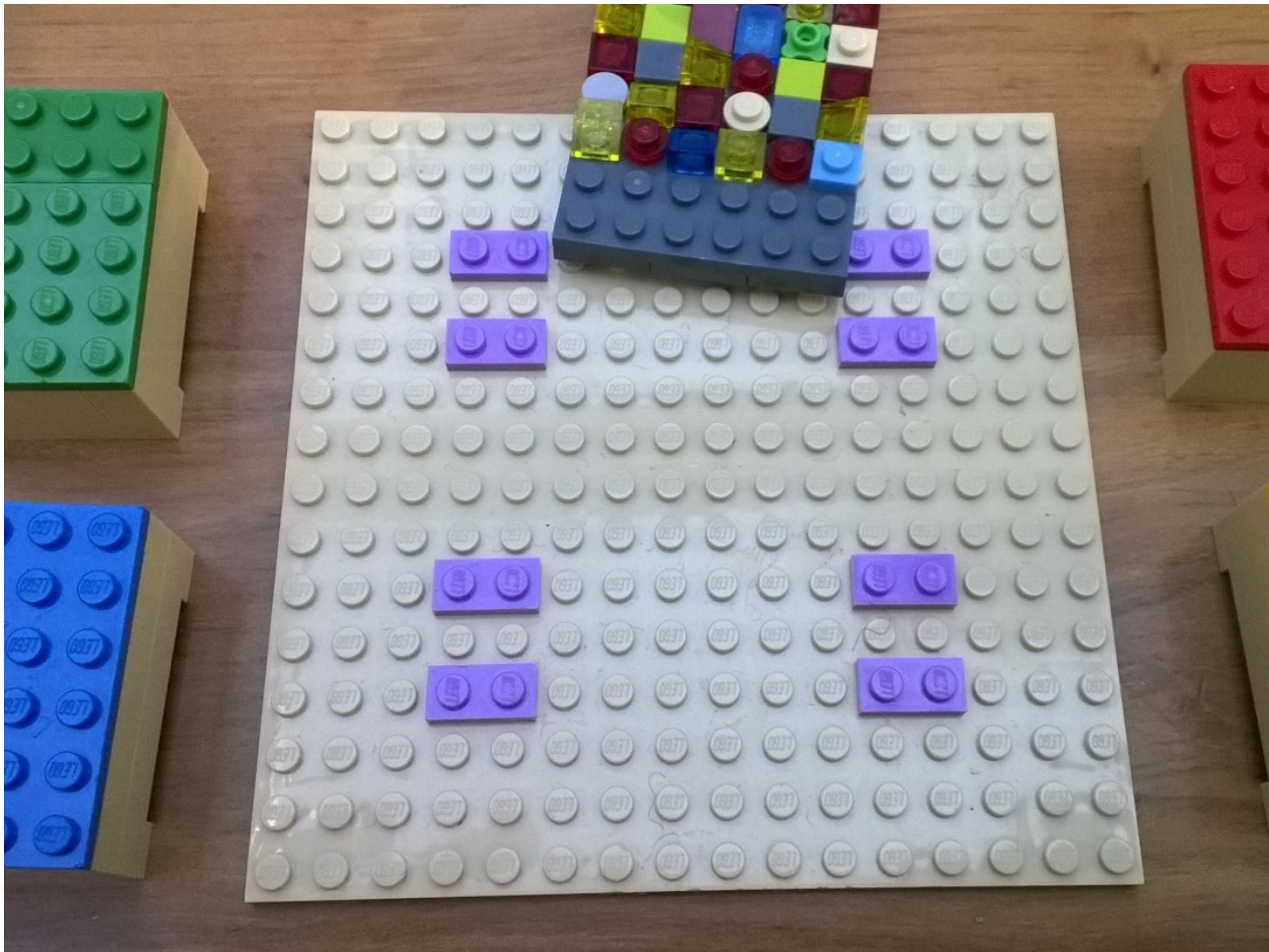
Silicon Interposer

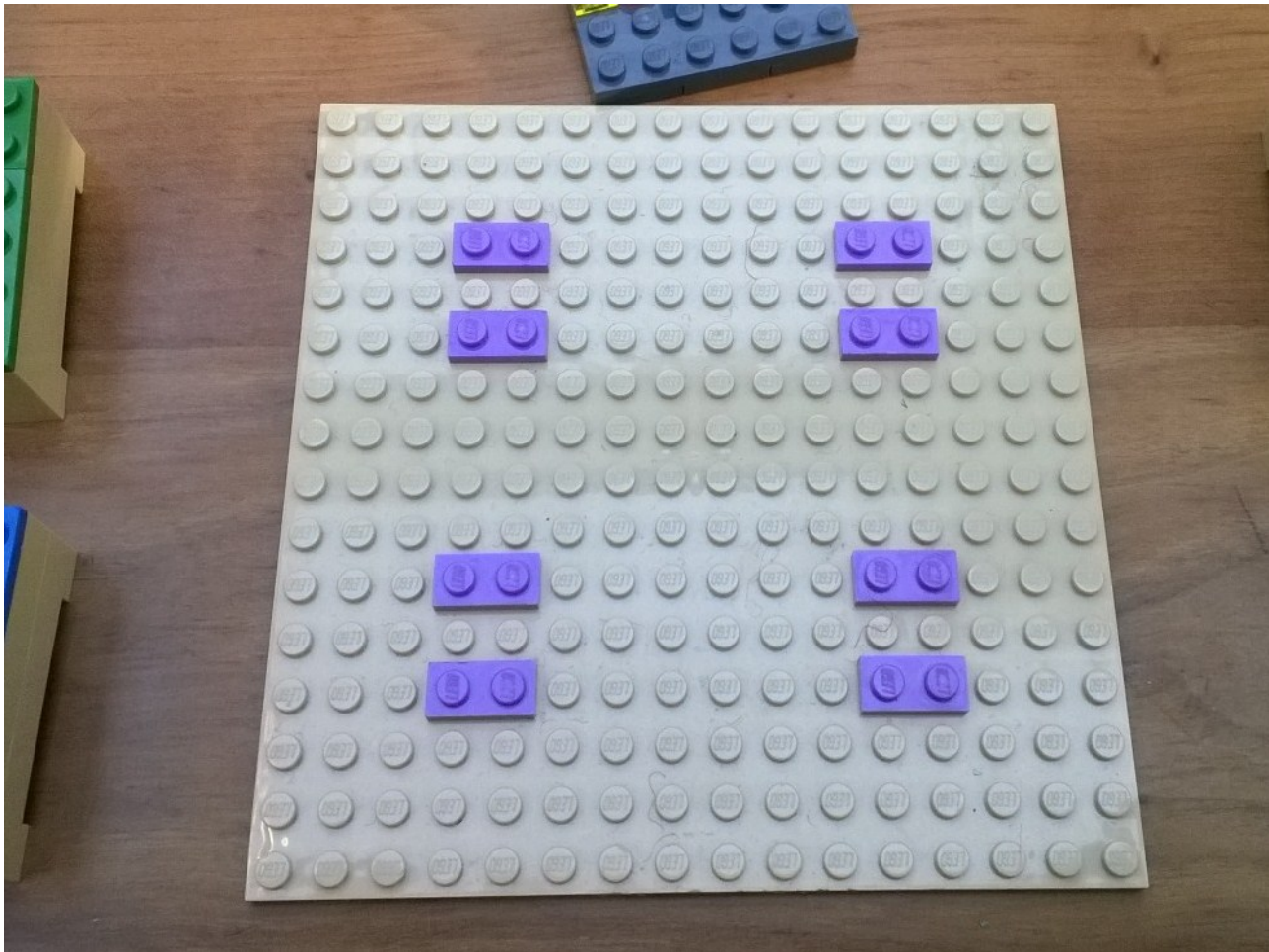


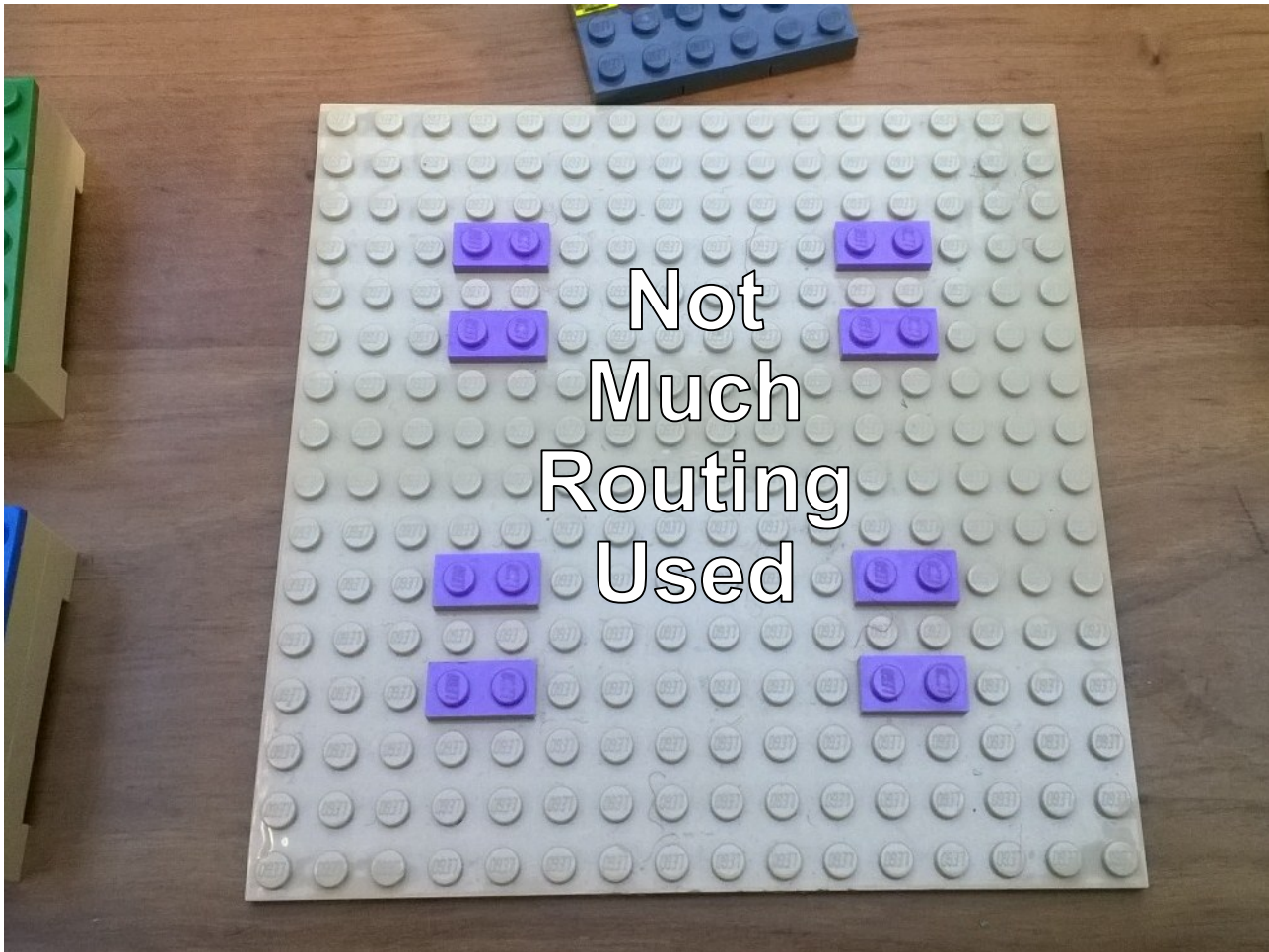








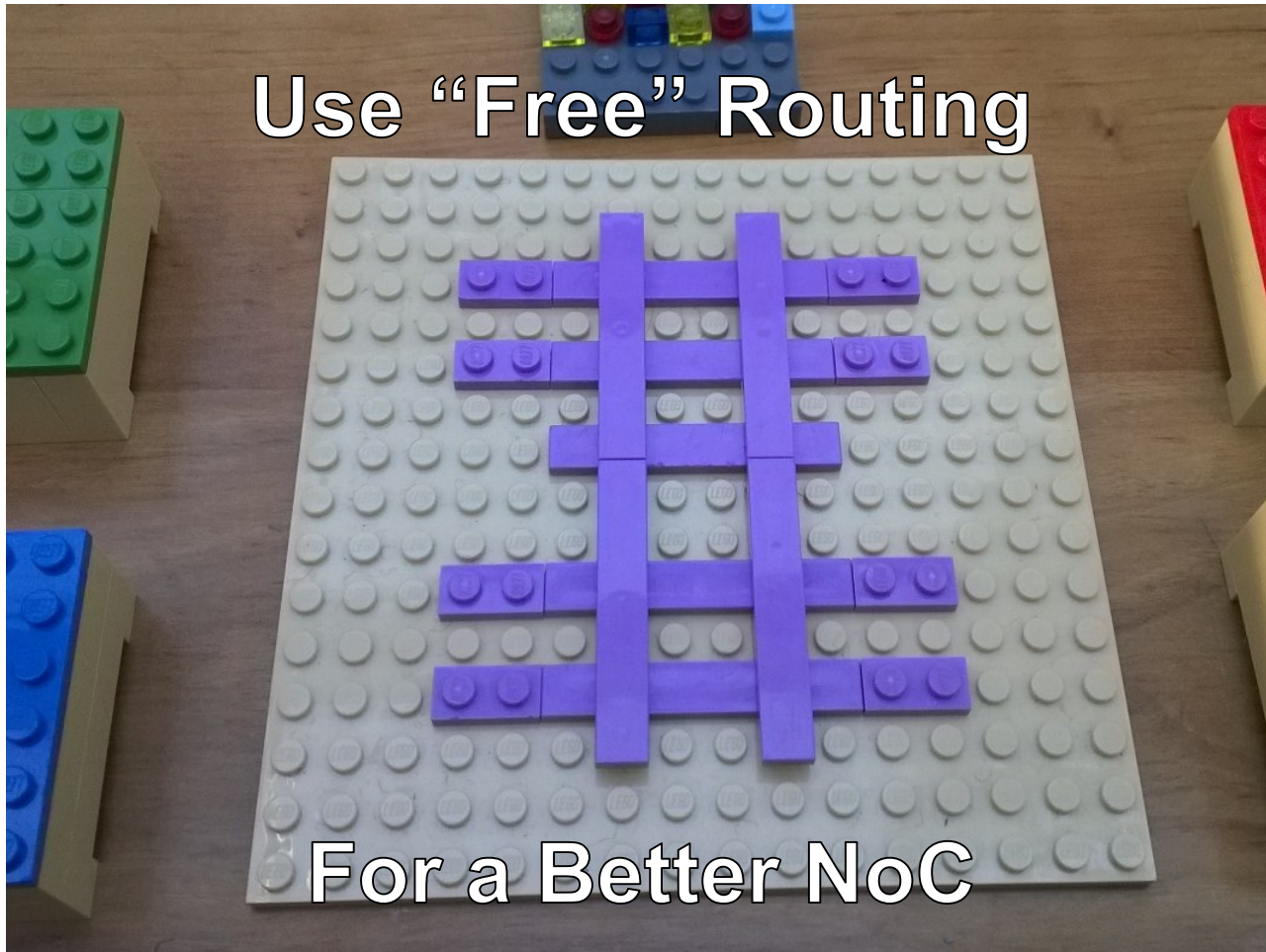




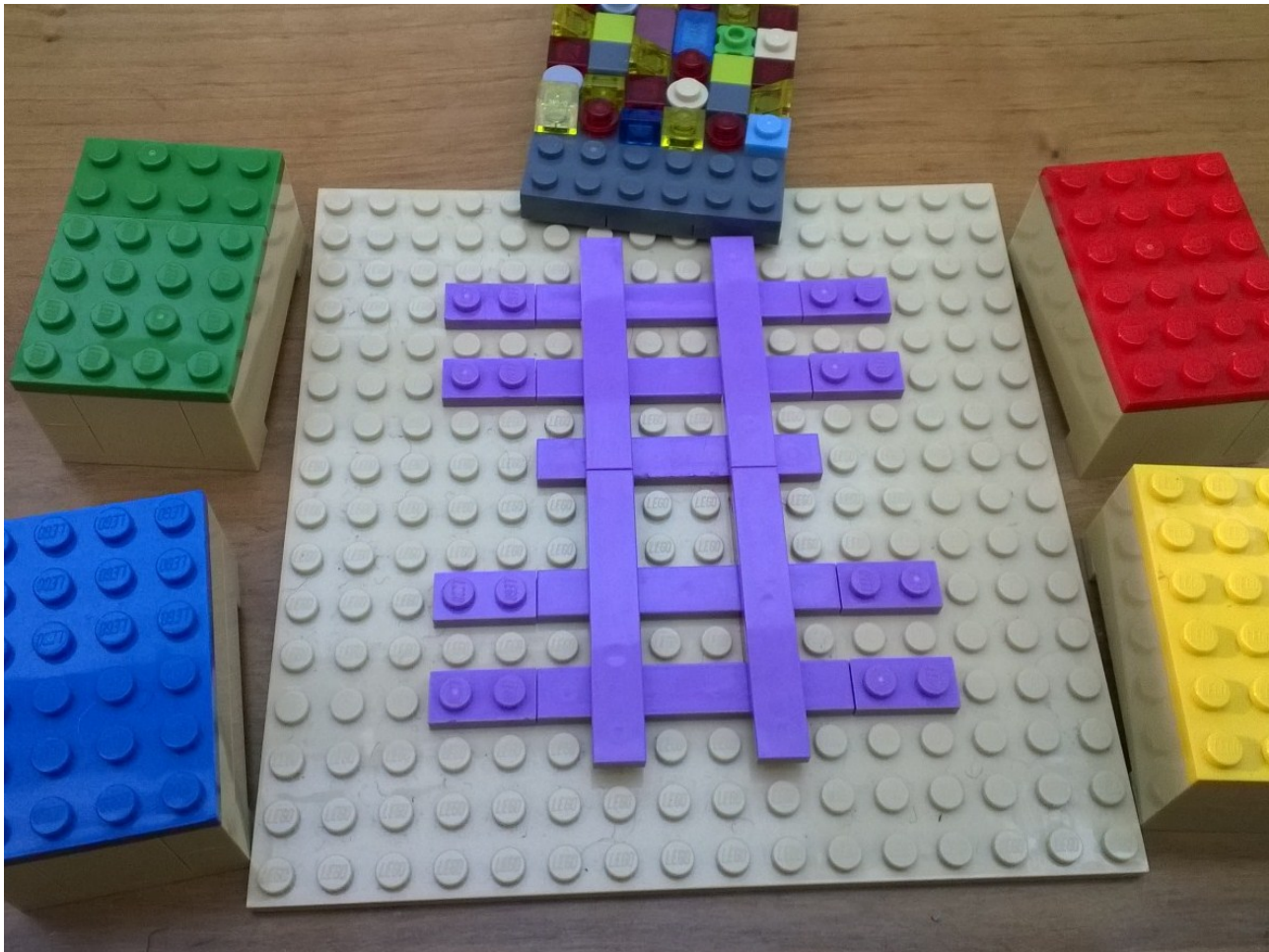
Not  
Much  
Routing  
Used

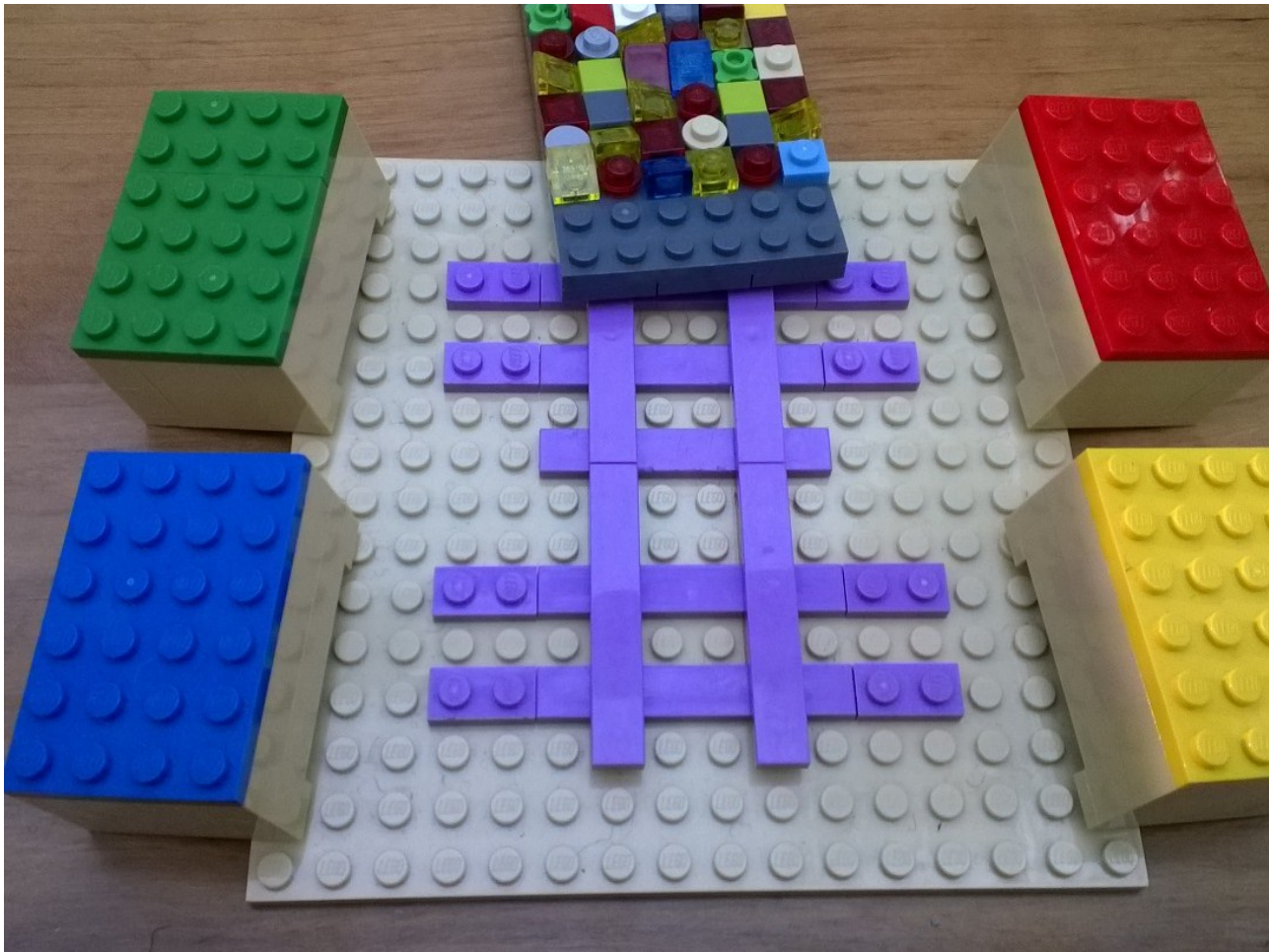


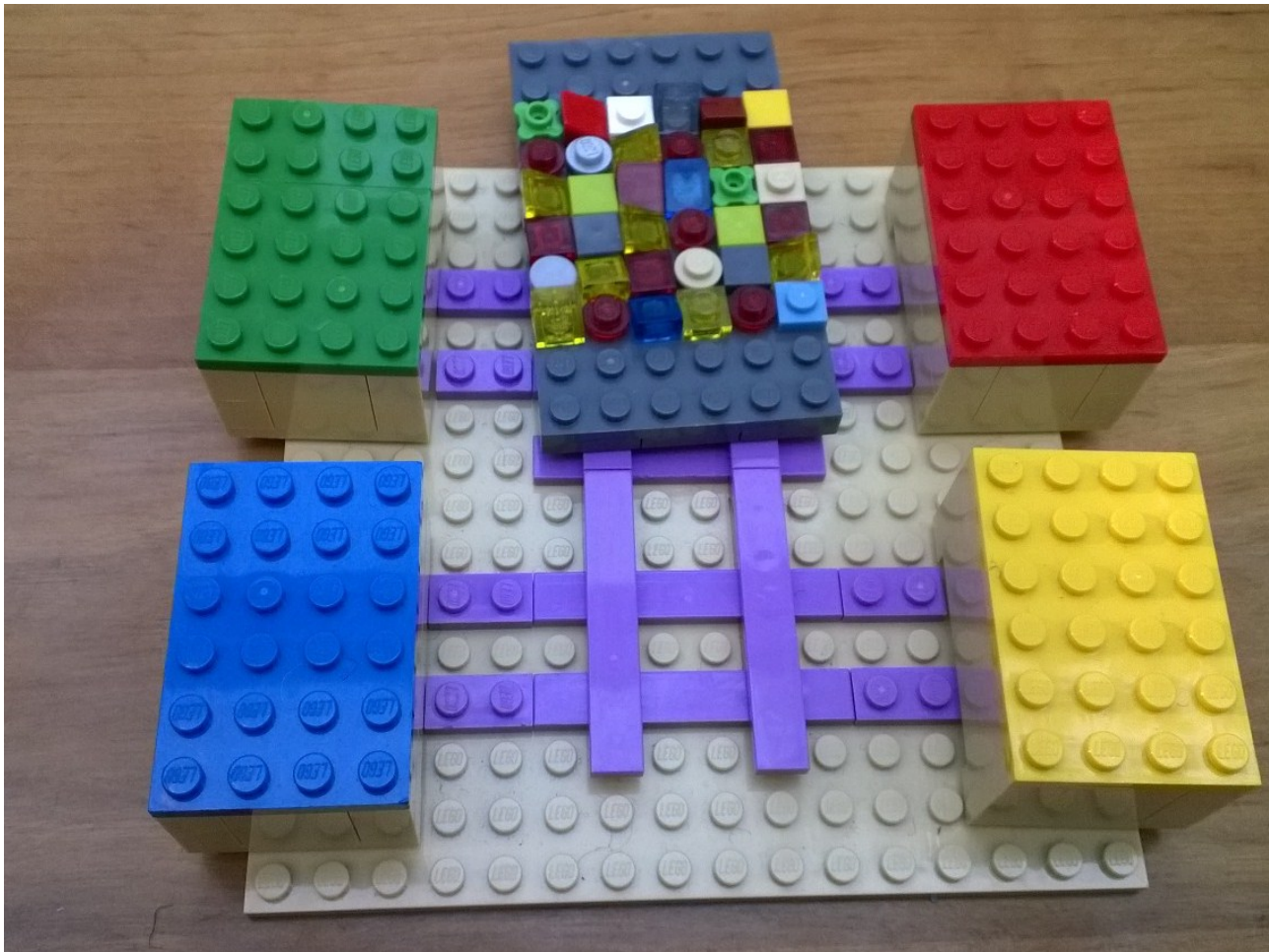
Use “Free” Routing

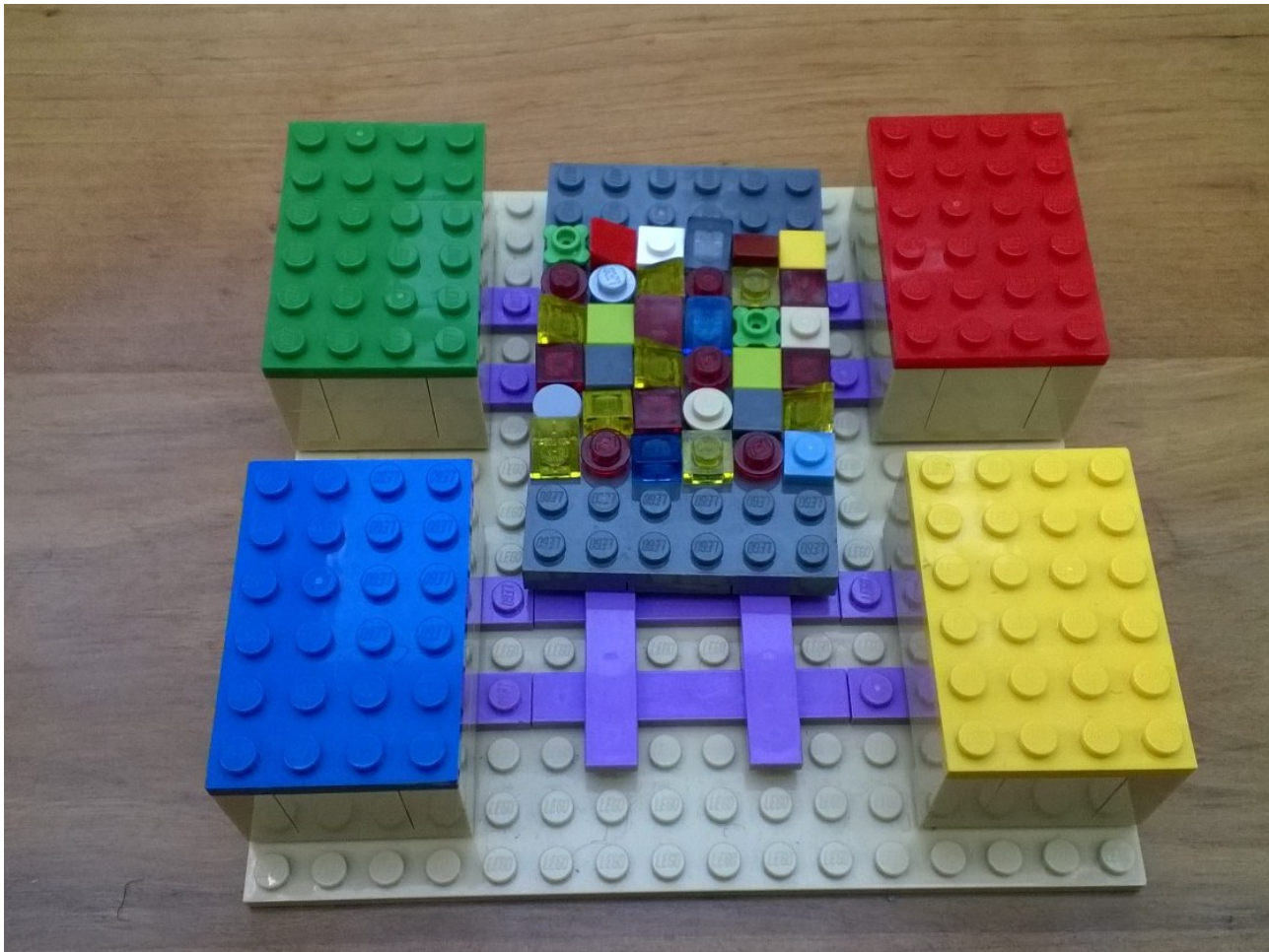


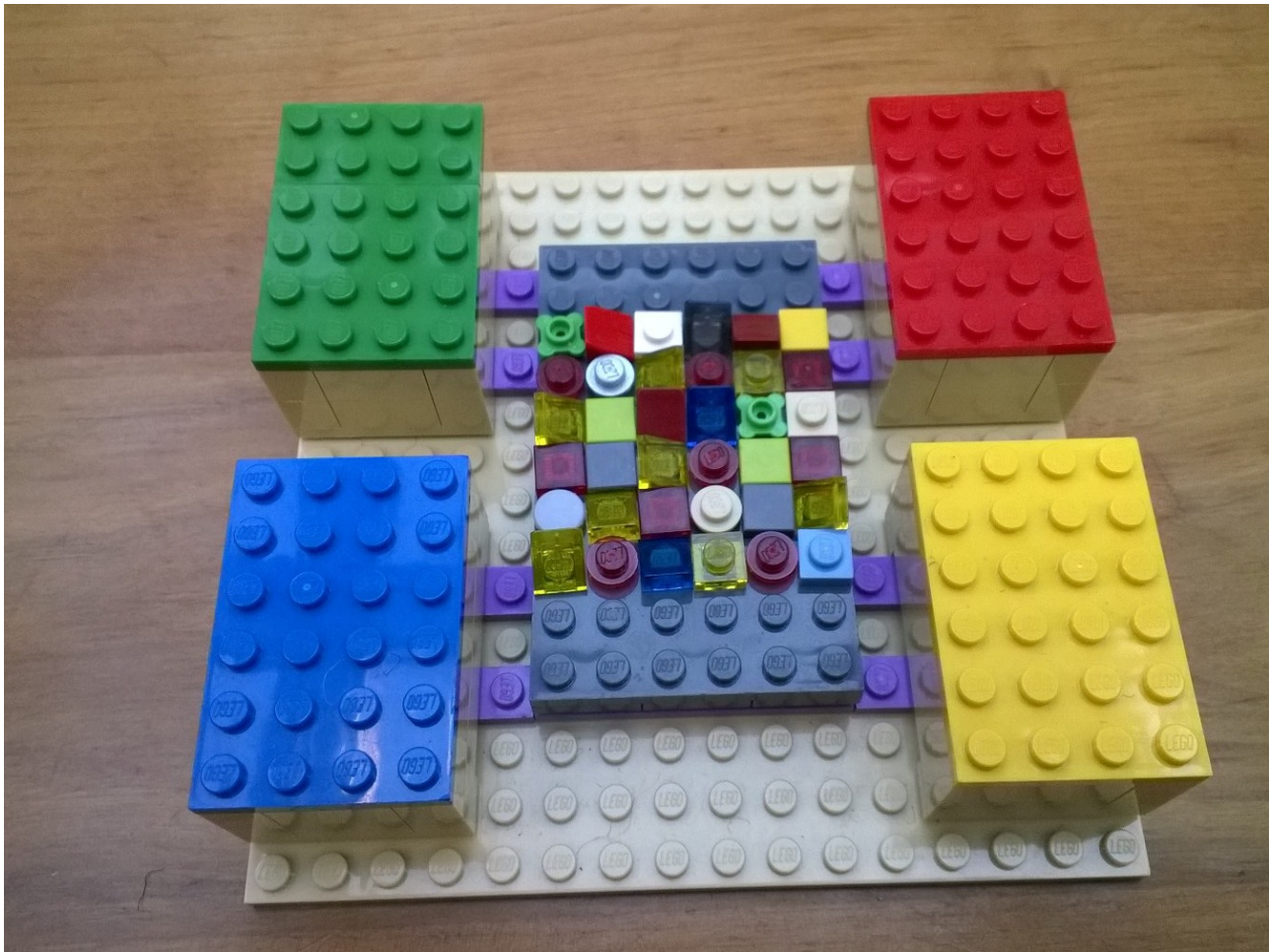
For a Better NoC



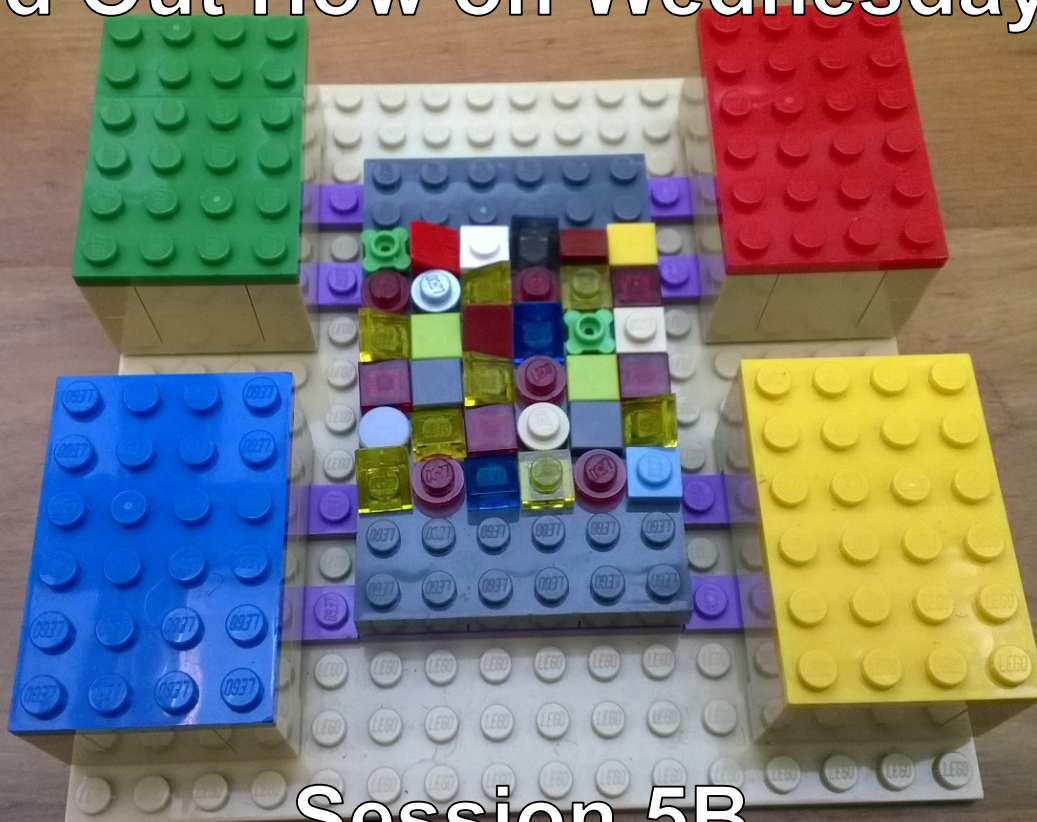








Find Out How on Wednesday



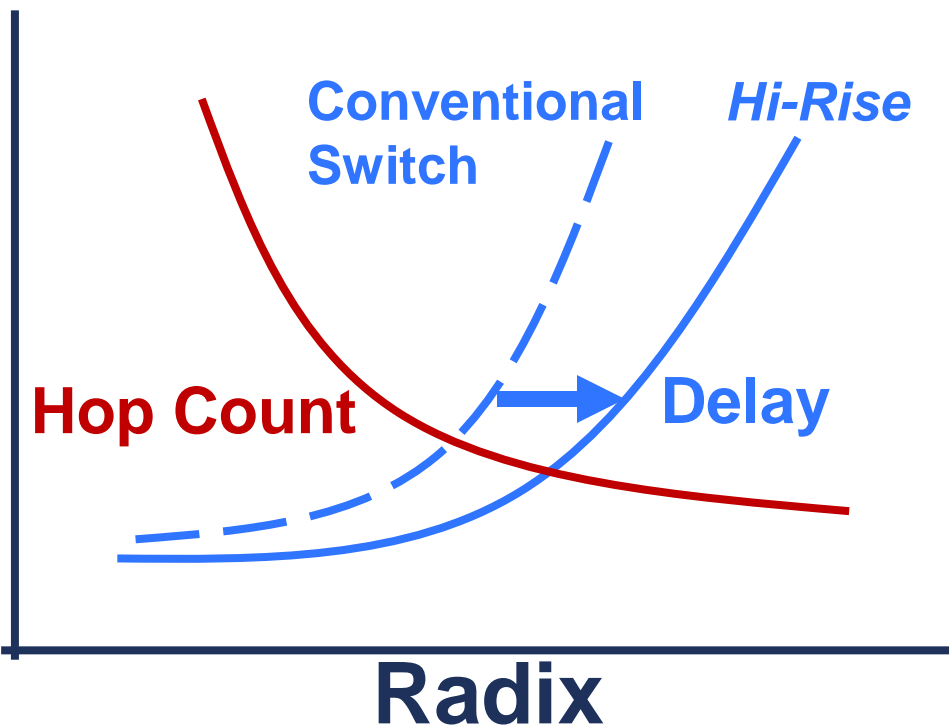
Session 5B

*next paper*



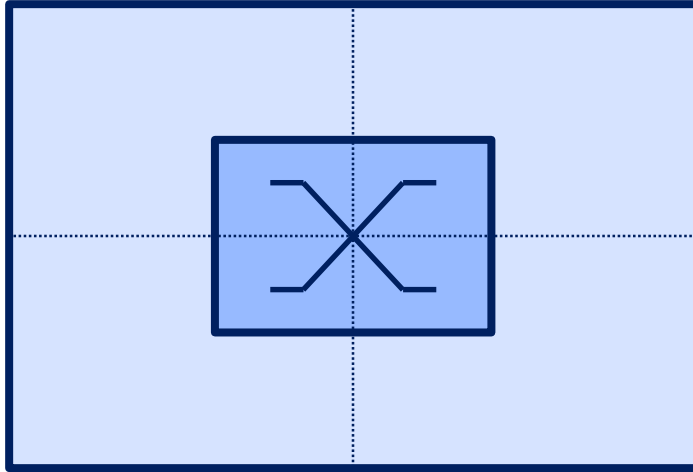
# Hi-Rise: A High-Radix Switch for 3D Integration with Single-cycle Arbitration

**Supreet Jeloka**, Reetuparna Das, Ronald G. Dreslinski,  
Trevor Mudge, David Blaauw  
**University of Michigan, Ann Arbor**

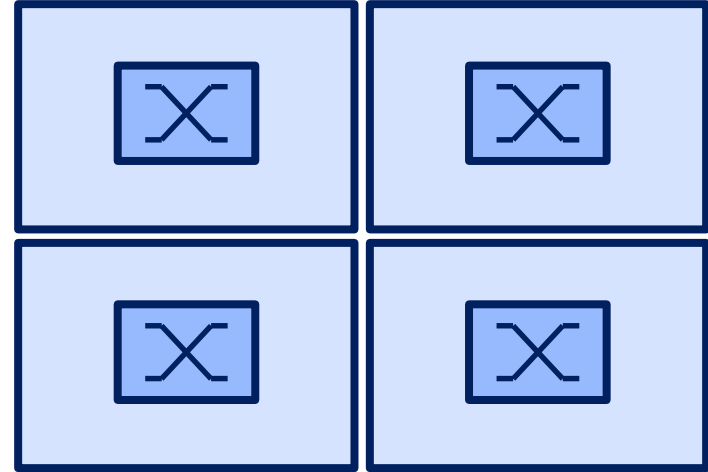


- Many-core systems
- Low-radix not scalable
- Goal: 3D High-radix switch

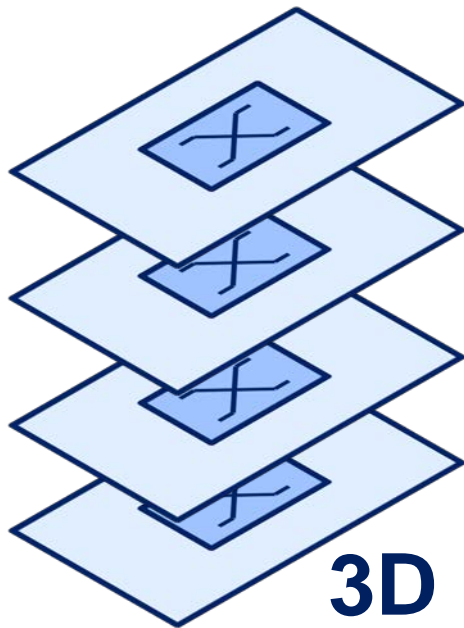
# 2D & 3D Switch Designs



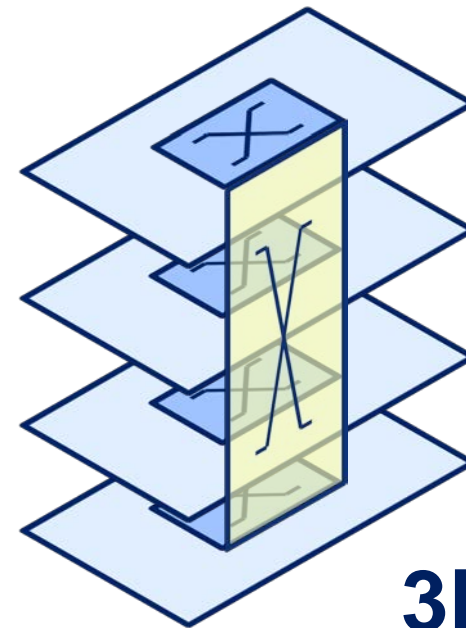
**2D Design**



**2D Partitioned**

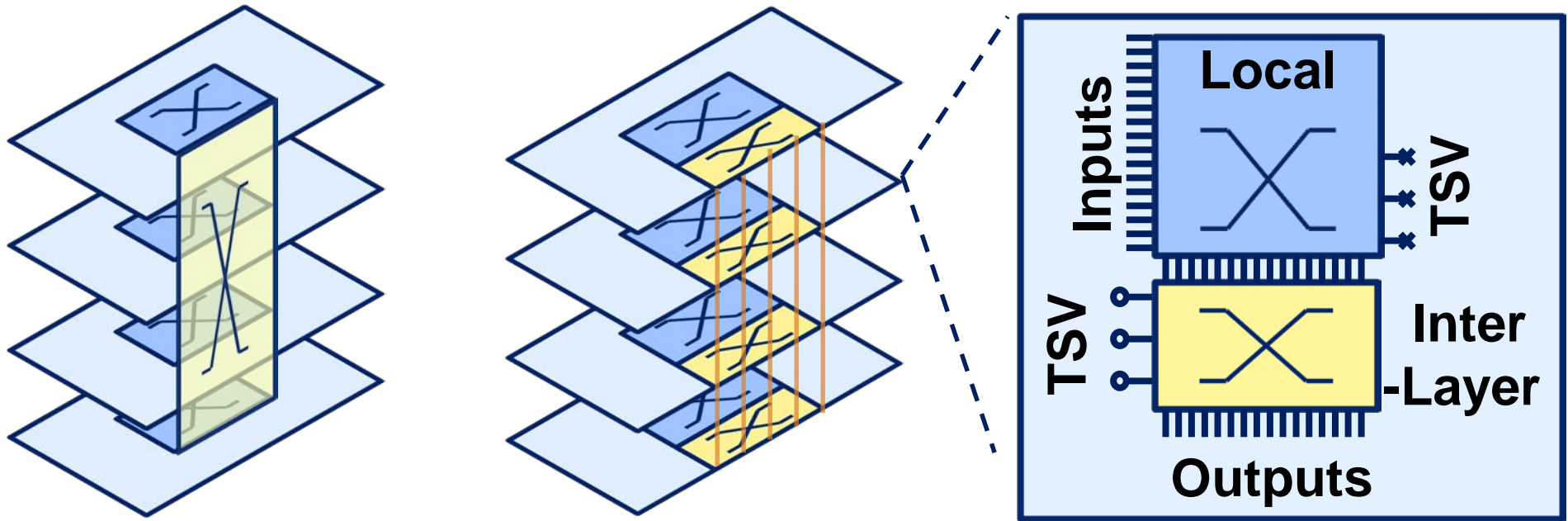


**3D Stacking**



**3D Switch**

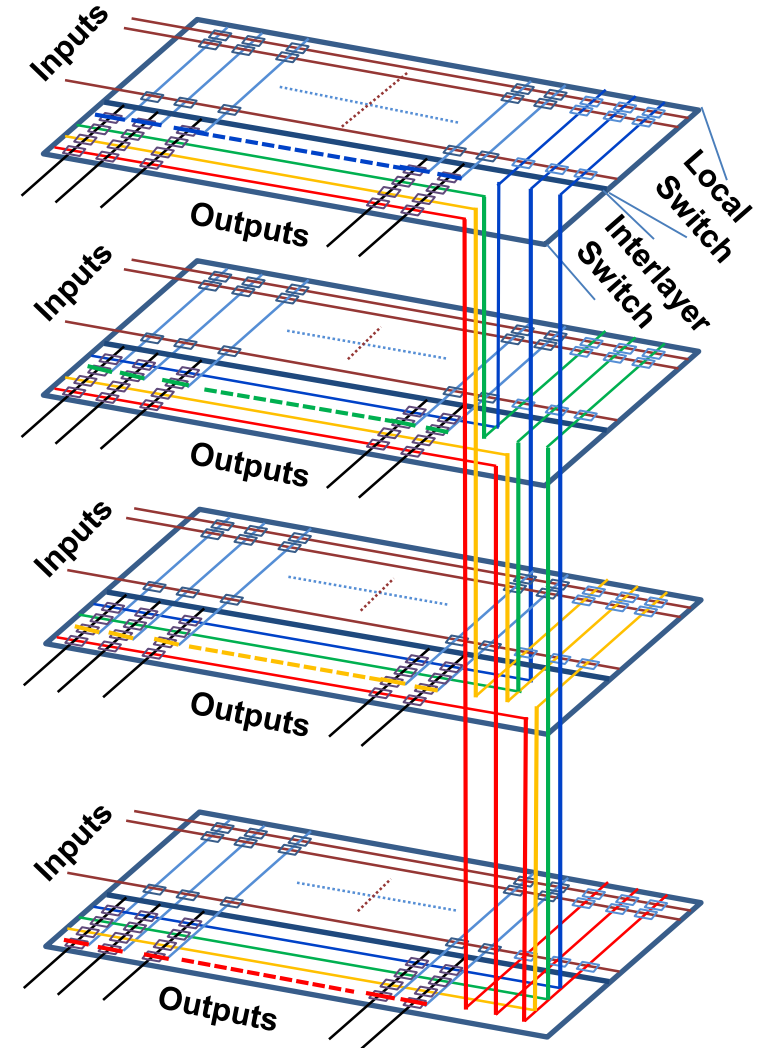
# 3D Switch Design Challenges



- Efficient High Radix 3D switch requires
  - Optimized **datapath** for connection heterogeneity
  - Fair low-cost **arbitration** for multi-stage switch

# Proposed 3D-Switch: *Hi-Rise*

- True 3D Switch
- Hierarchical datapath
  - Reduced TSVs
- Class-based arbitration
  - Composable
  - Single cycle & Built-In
- 64-Radix 4-Layer Hi-Rise
  - 2.2 GHz, 10.65 Tbps
  - 44pJ / 128-bit transaction
  - 13% System speedup over FBFly

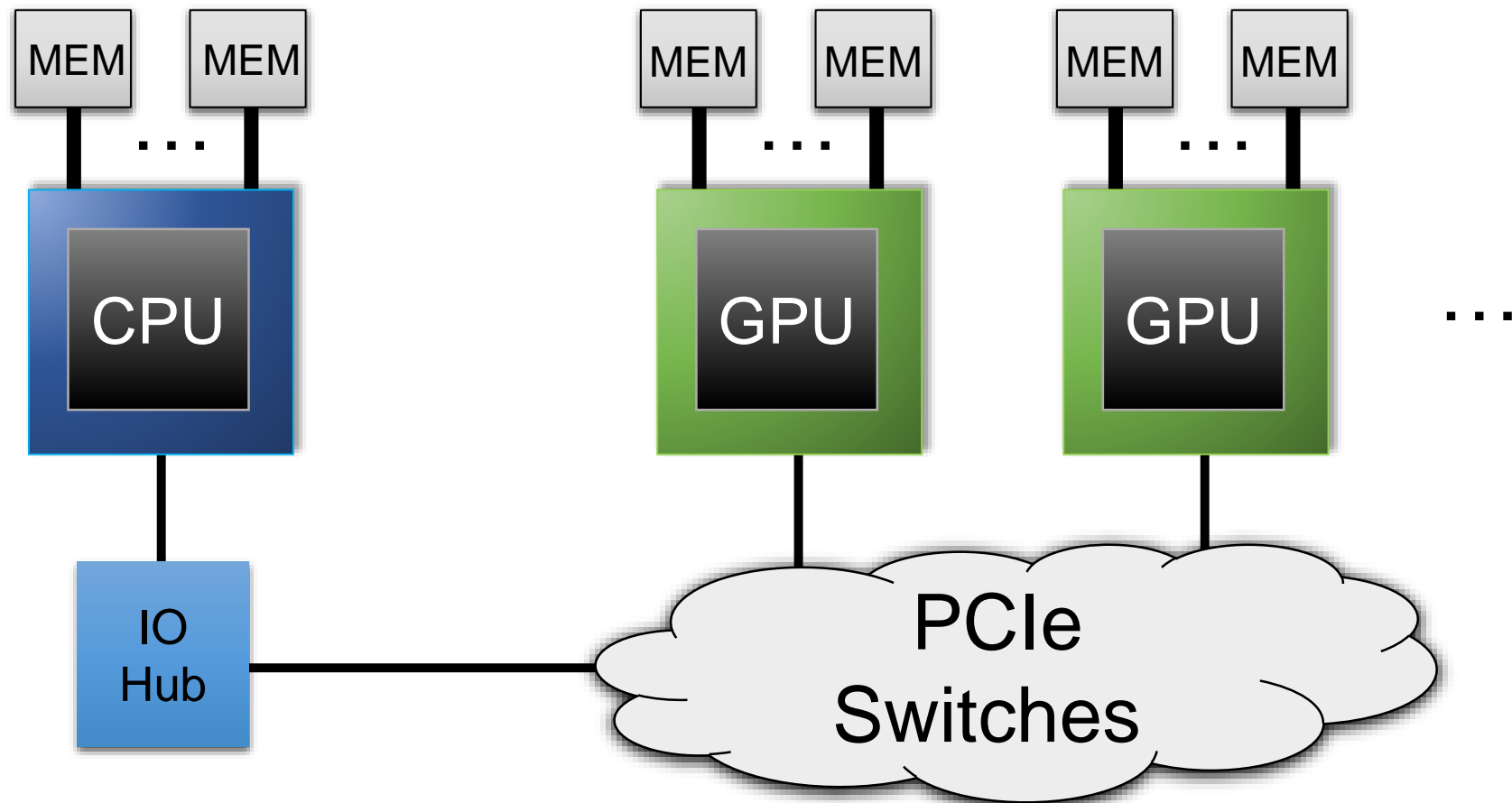


*next paper*

# Multi-GPU System Design with Memory Networks

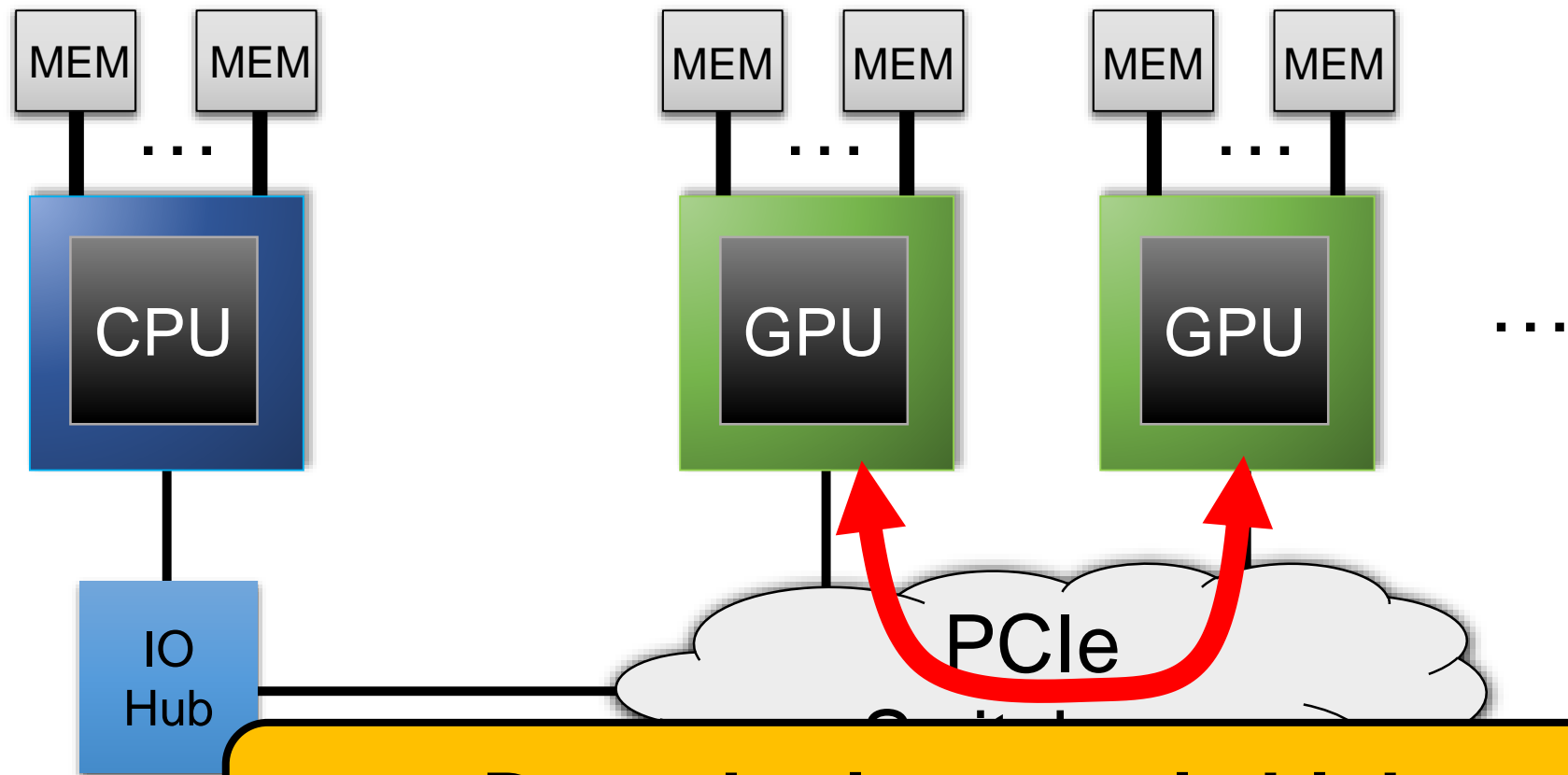
Gwangsun Kim, Minseok Lee, Jiyun Jeong, John Kim  
Department of Computer Science, KAIST

---



# Multi-GPU System Design with Memory Networks

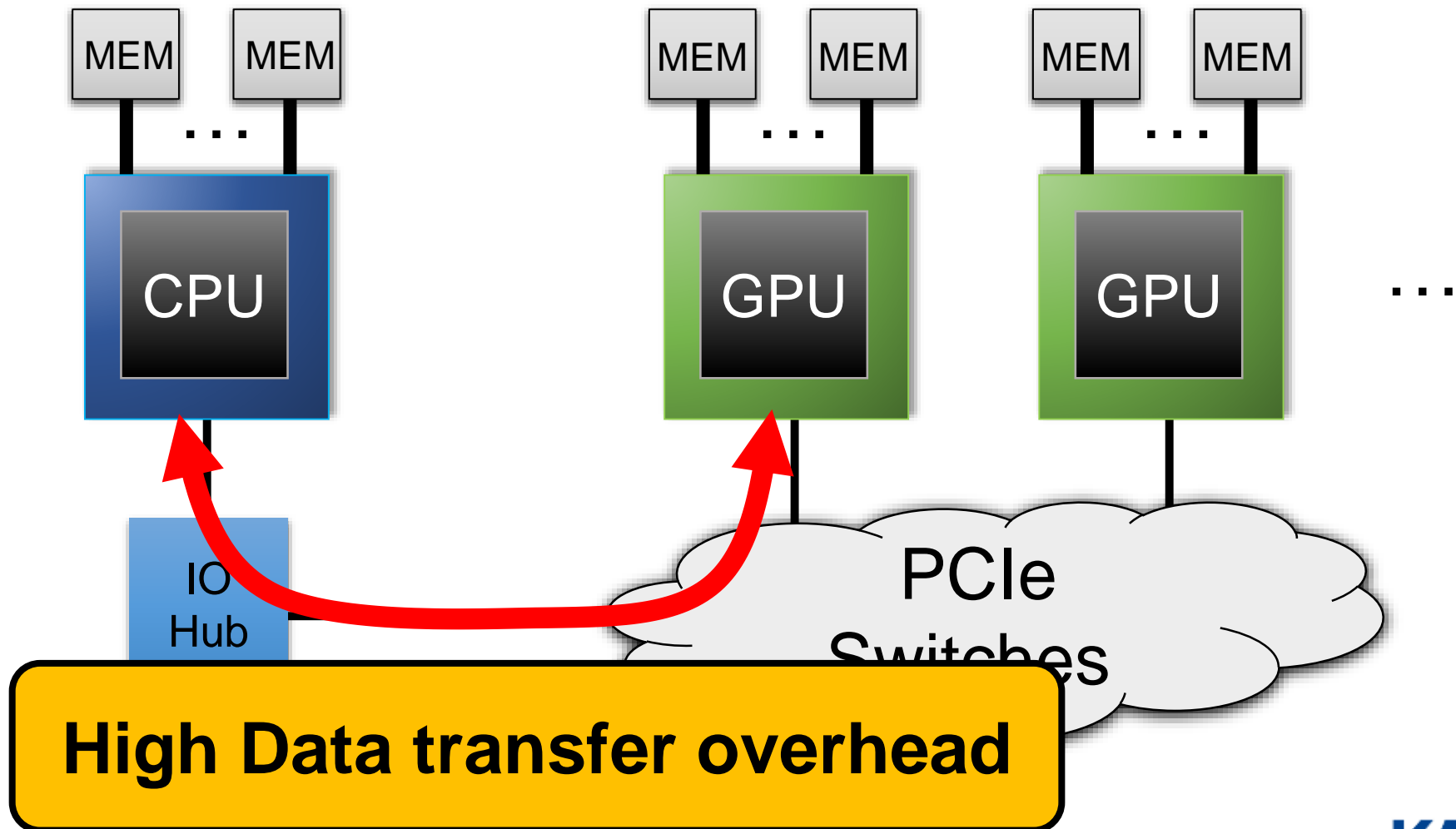
Gwangsun Kim, Minseok Lee, Jiyun Jeong, John Kim  
Department of Computer Science, KAIST



**Data sharing cost is high.  
Programming can be challenging.**

# Multi-GPU System Design with Memory Networks

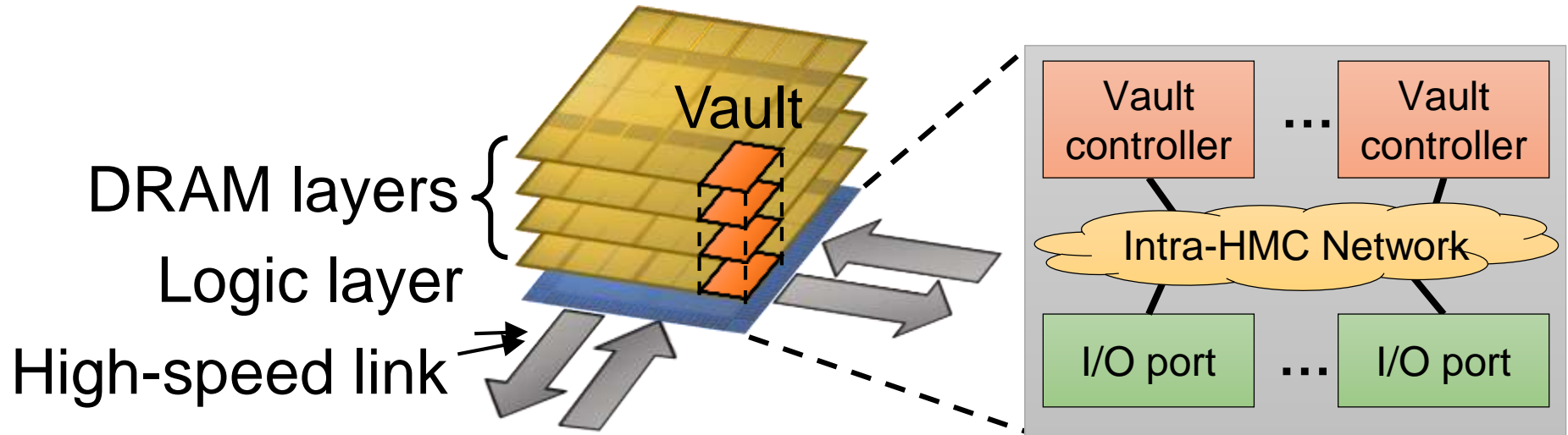
Gwangsun Kim, Minseok Lee, Jiyun Jeong, John Kim  
Department of Computer Science, KAIST





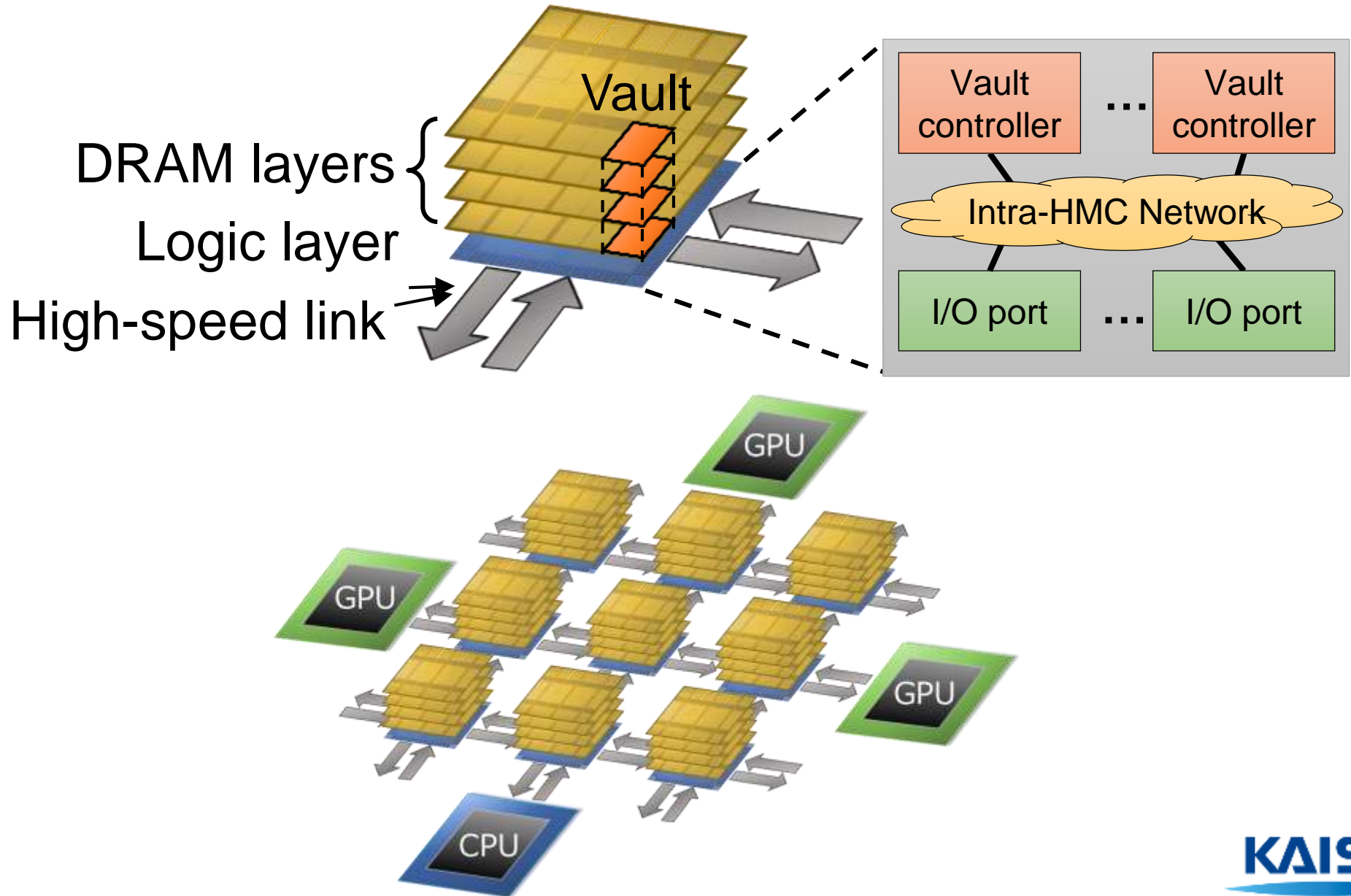
# Memory Network

## Hybrid Memory Cube (HMC)



# Memory Network

## Hybrid Memory Cube (HMC)



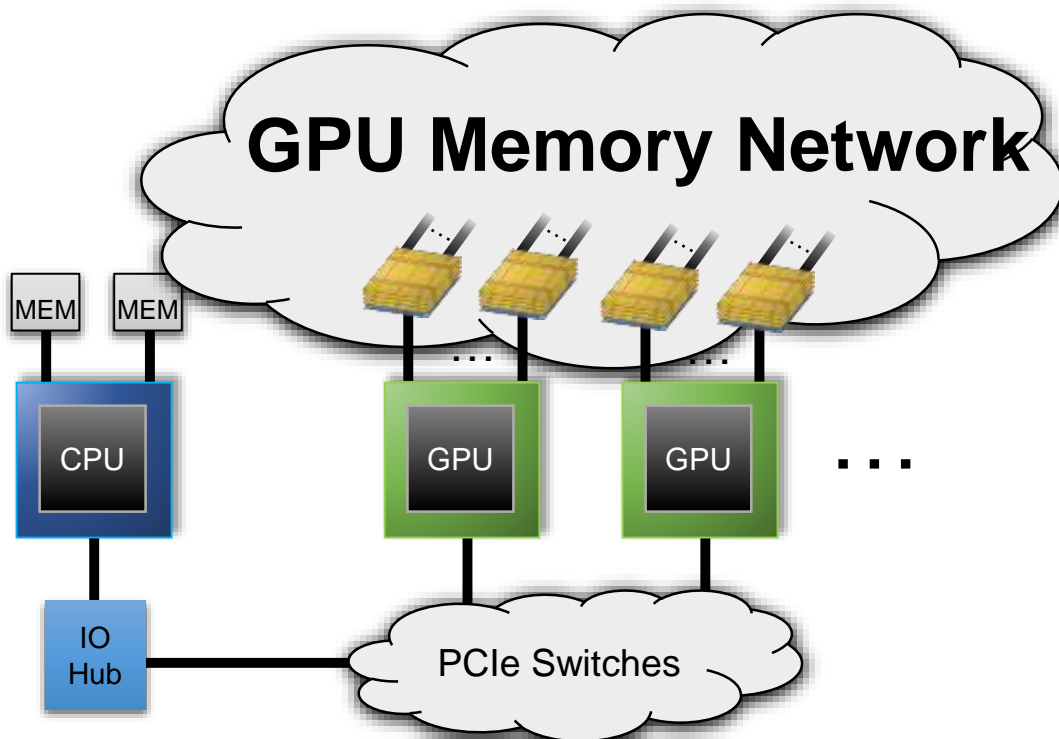
# Multi-GPU System Design Space

---

**How to design the different networks?**

# Multi-GPU System Design Space

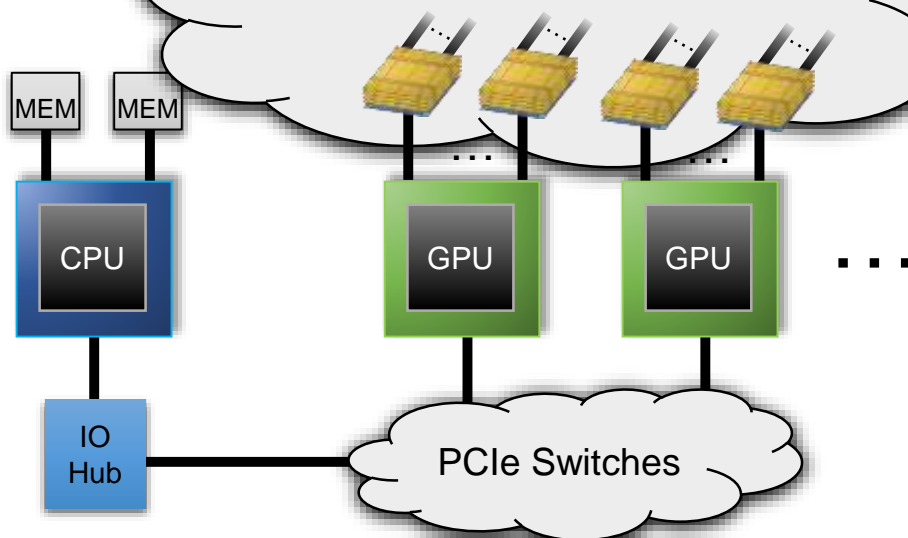
How to design the different networks?



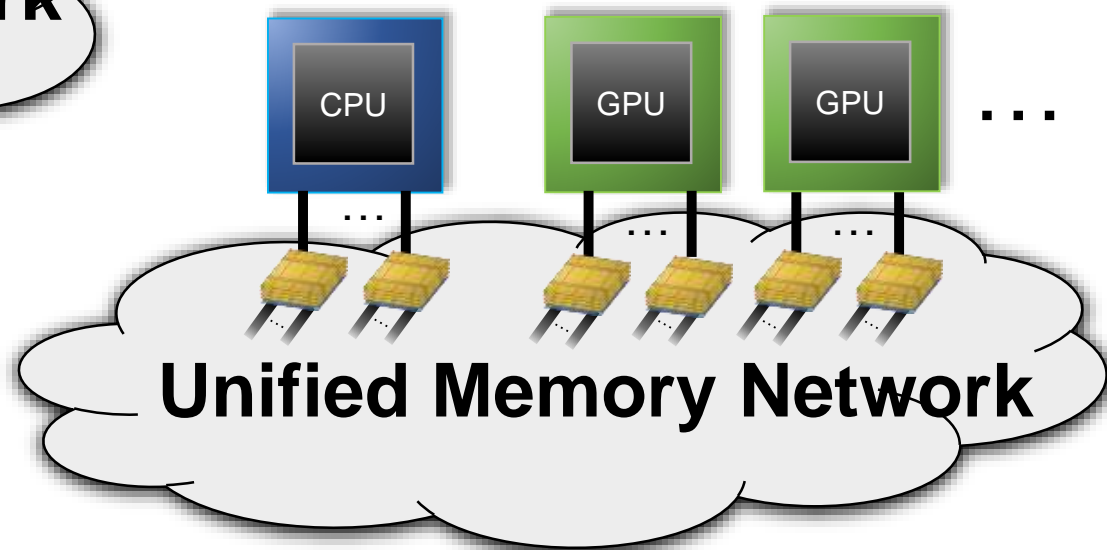
# Multi-GPU System Design Space

How to design the different networks?

**GPU Memory Network**

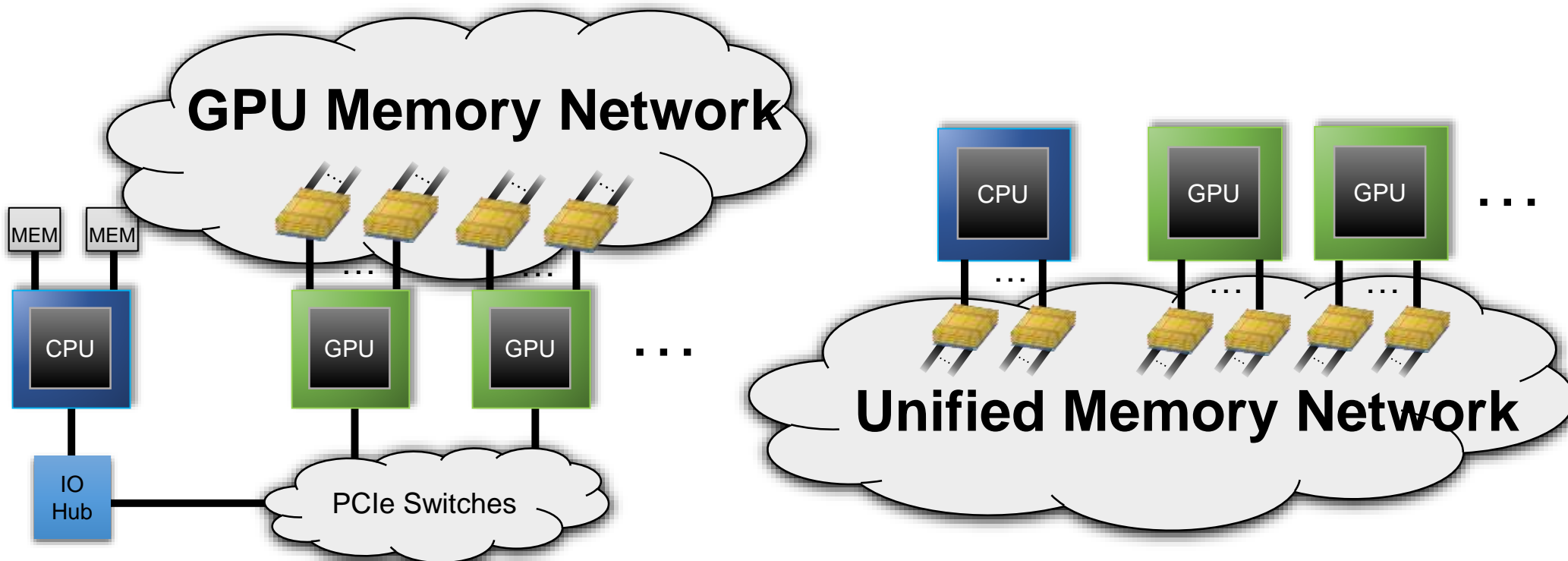


**Unified Memory Network**



# Multi-GPU System Design Space

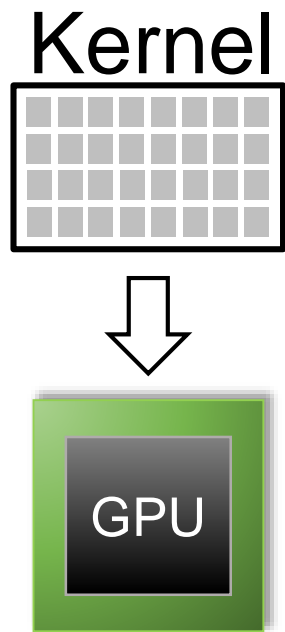
How to design the different networks?



Impact on multi-GPU programming?

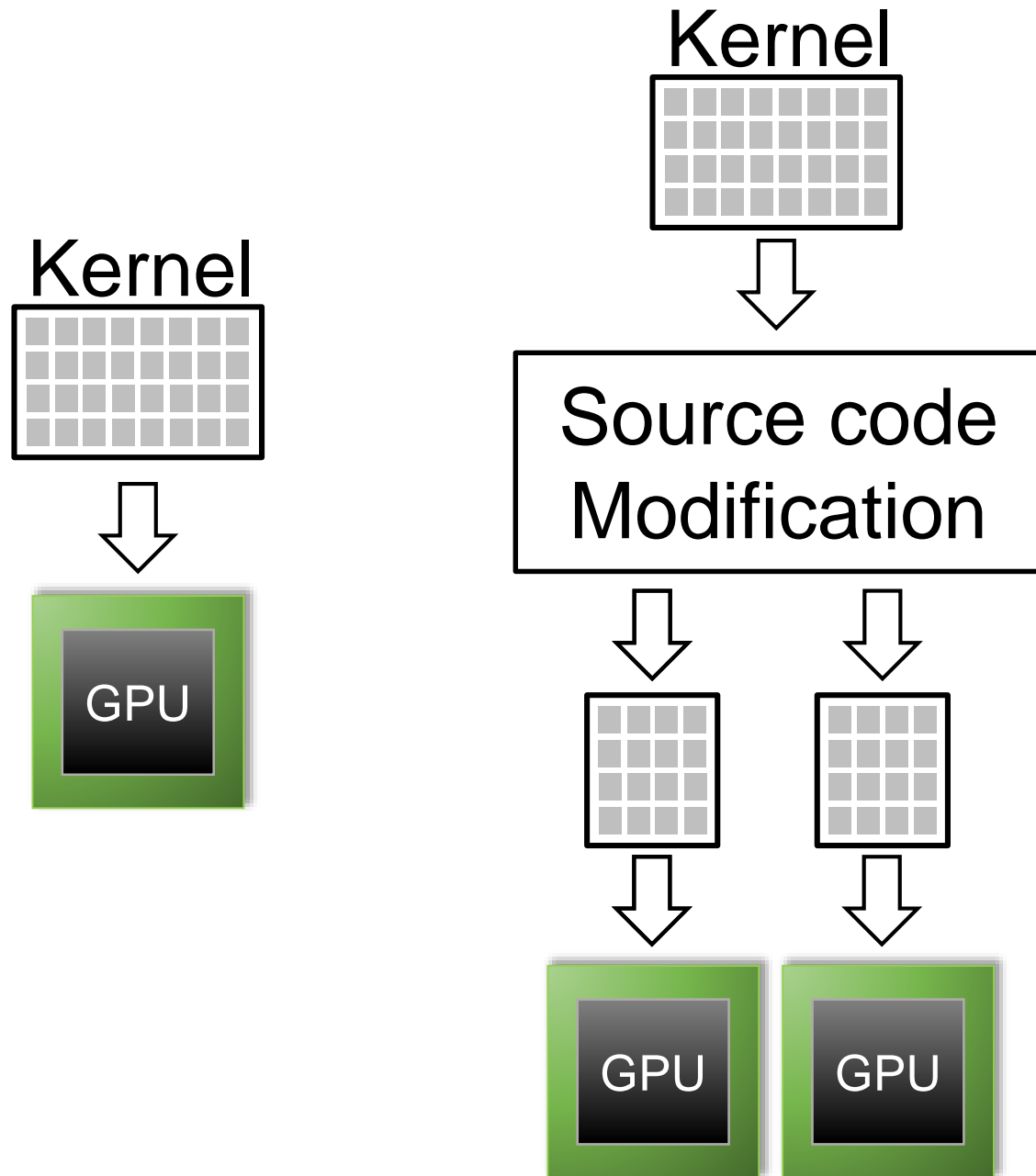
# Scalable Kernel Execution

---



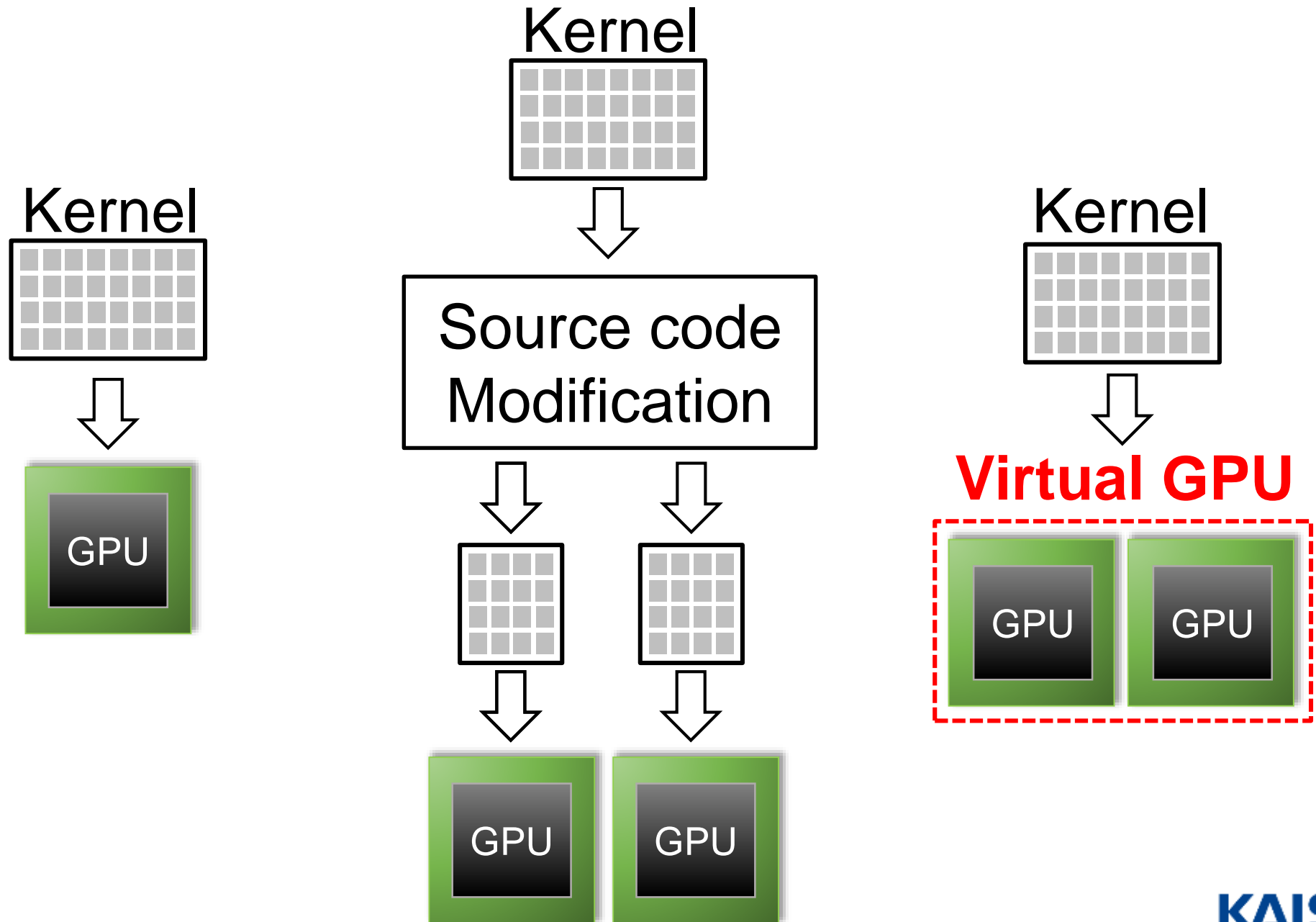
# Scalable Kernel Execution

---

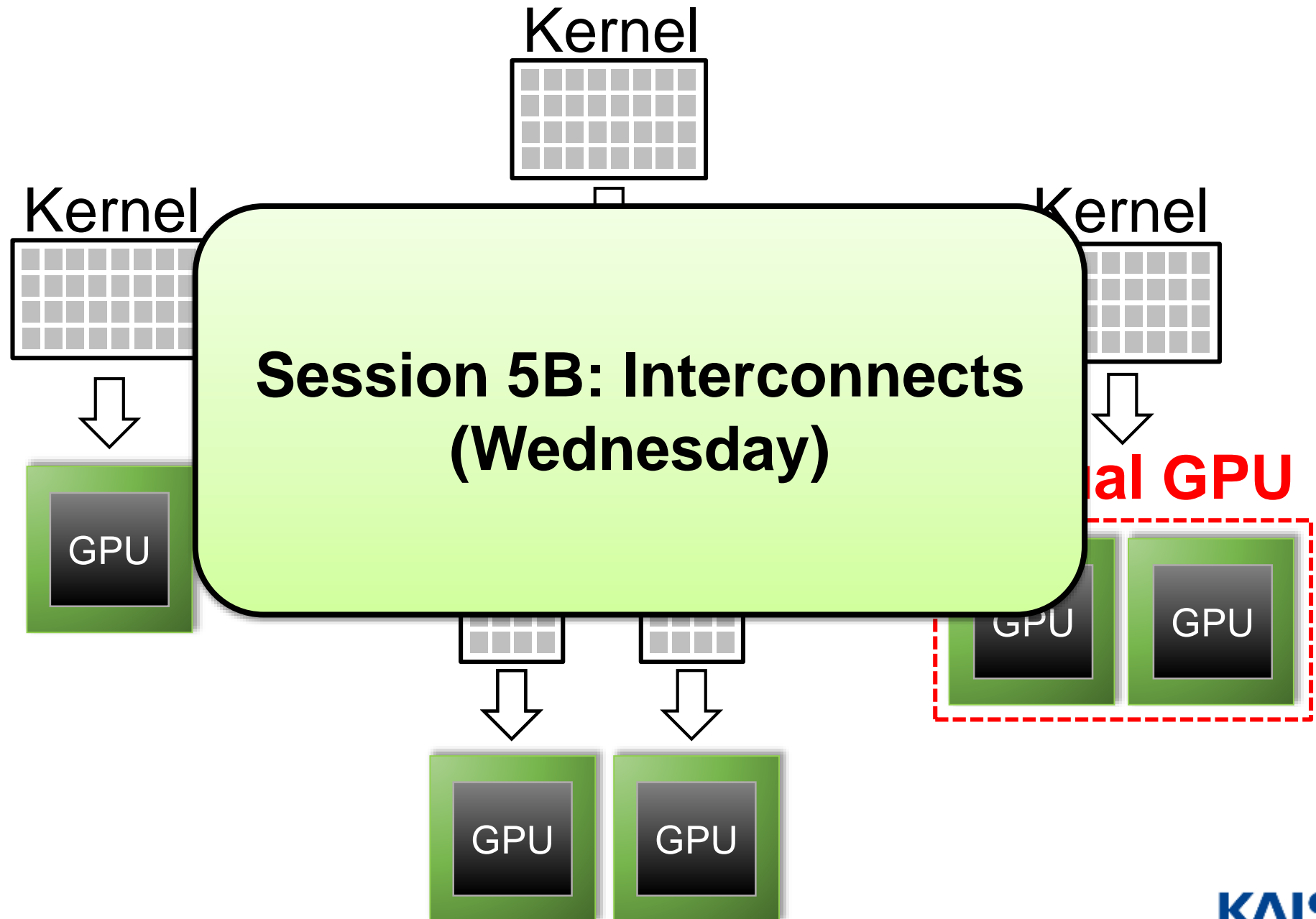




# Scalable Kernel Execution



# Scalable Kernel Execution

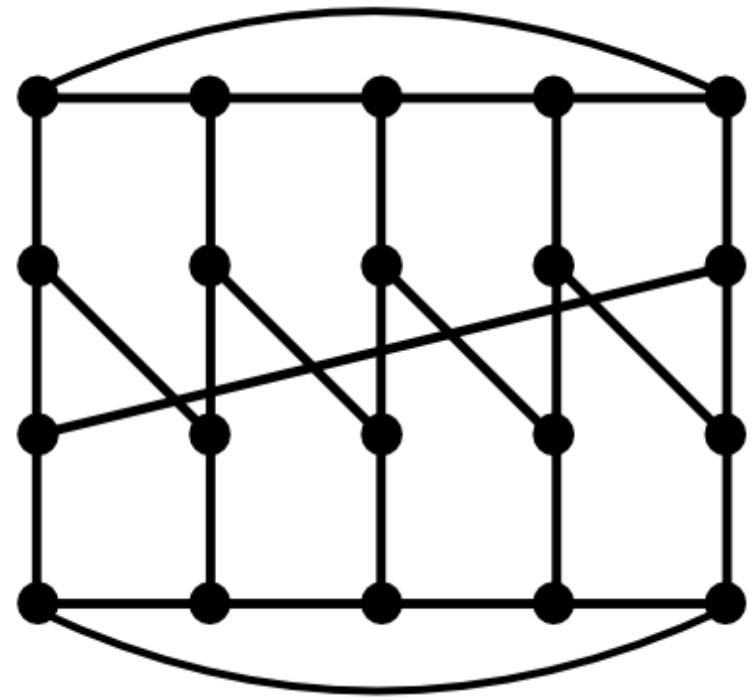
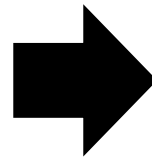
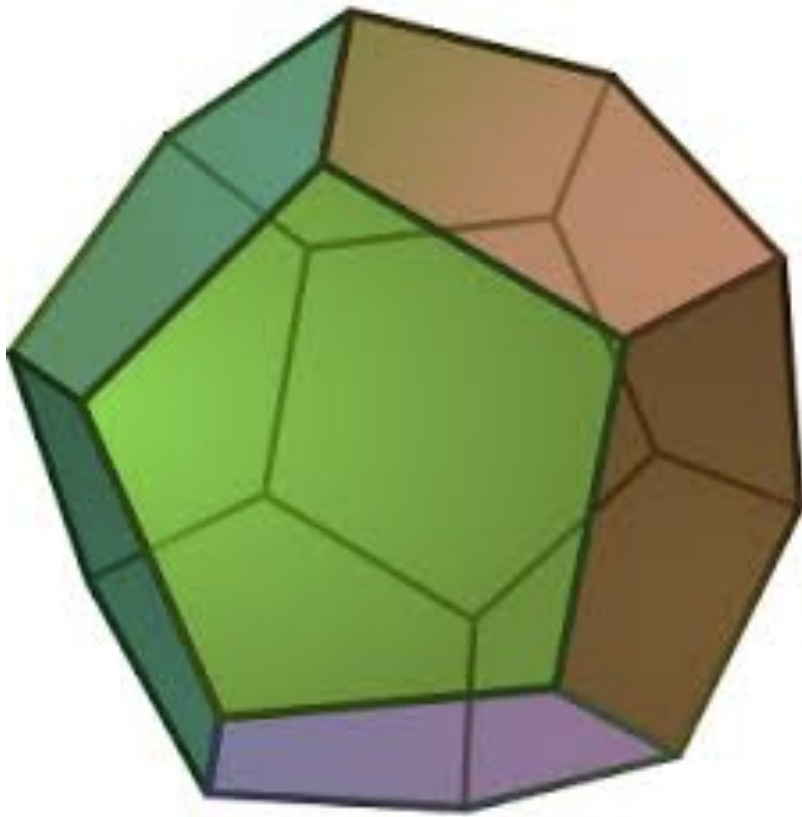


*next paper*

# Dodec: Random-Link, Low-Radix On-Chip Networks

Haofan Yang, Jyoti Tripathi, **Natalie Enright Jerger**, Dan Gibson

Session 5B: Wednesday @ 11:05



# On-Chip Network Routers



© umnet.com

4-ported router

# On-Chip Network Routers



4-ported router



Many-ported router

# On-Chip Network Routers



4-ported router



Many-ported router



© muppet.wikia.com

Simple 3-ported router

# Topological Choices



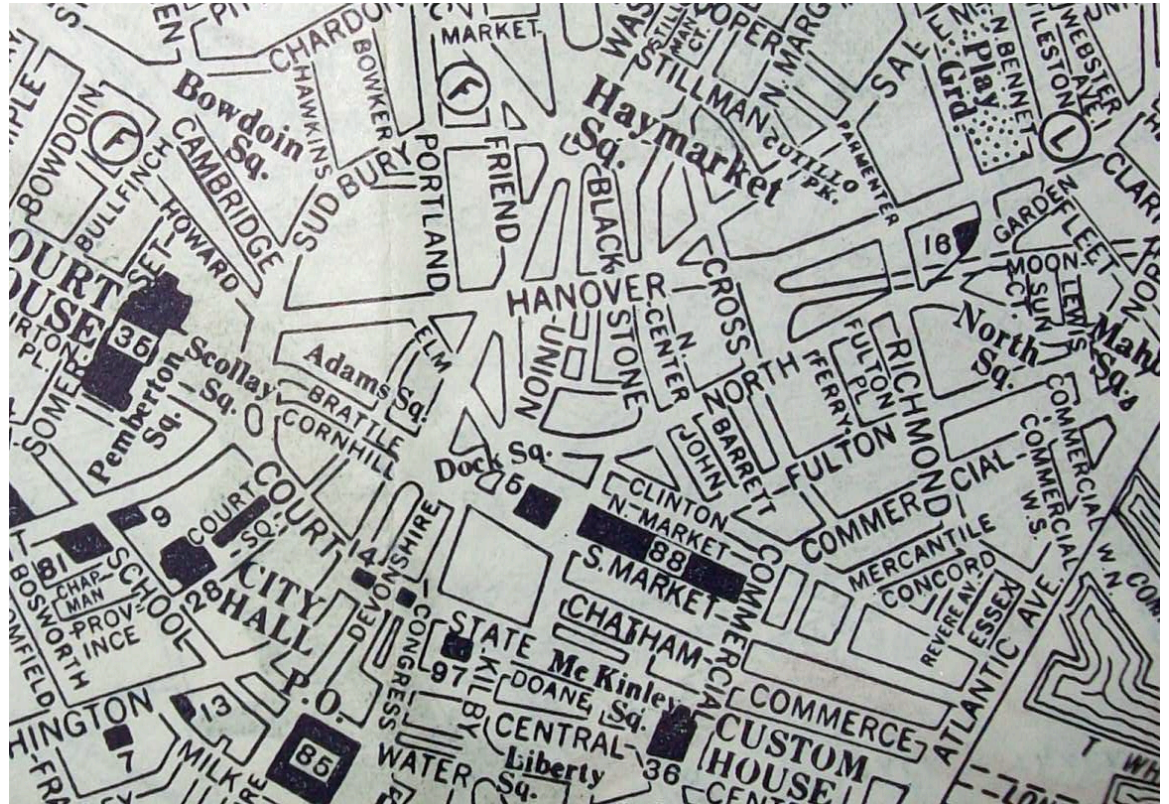
© wxs.ca

2D Mesh: regular, grid-like

- Grid cities: government designed



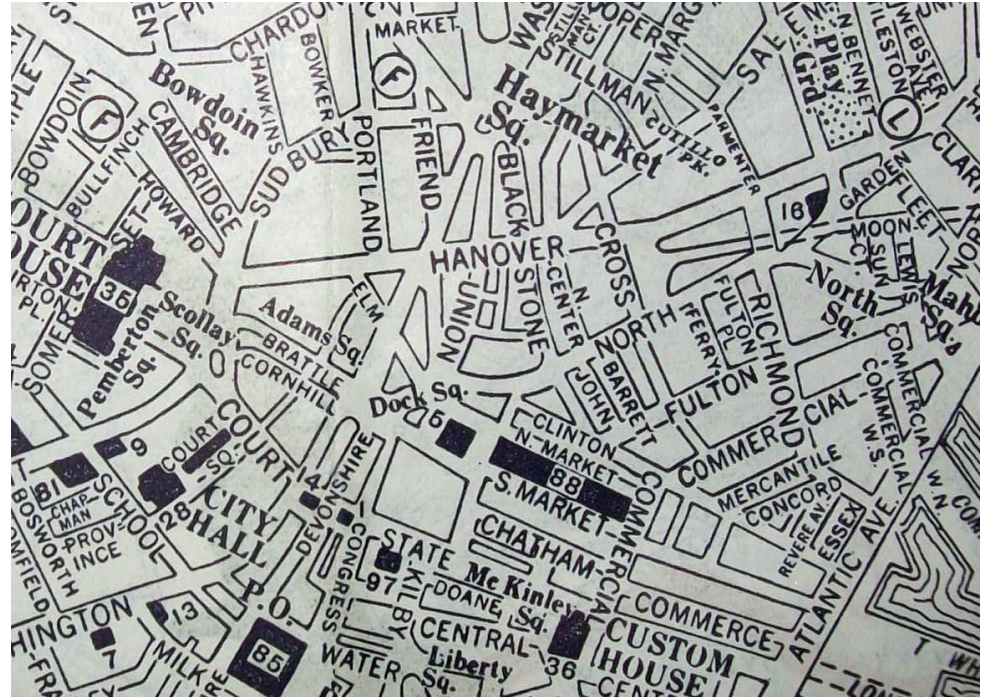
# Topological Choices



© city-data.com

- Grid cities: government designed
- Non-grid cities: grow organically based on how people use them
- Similarly, we consider irregular network structures that might be more **useful** on-chip

# Simple Routers + Irregular Networks!



Dodec: Random-Link, Low-Radix On-Chip Networks

Session 5B: Wednesday @ 11:05

*next paper*









	10:00	12:00	15:00	17:00	21:00
Monday	L	R	L	R	R
Wednesday	L	R	L	R	R
Friday	L	R	L	R	R

# Wormhole: Wisely Predicting Multidimensional Branches

**Jorge Albericio,**  
Joshua San Miguel,  
Natalie Enright Jerger, and Andreas Moshovos

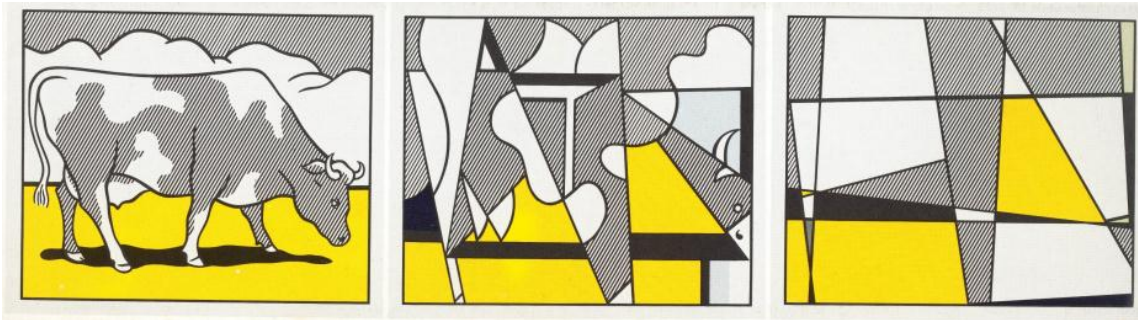


**Wednesday. 13:00**  
**Session 6A**



*next paper*

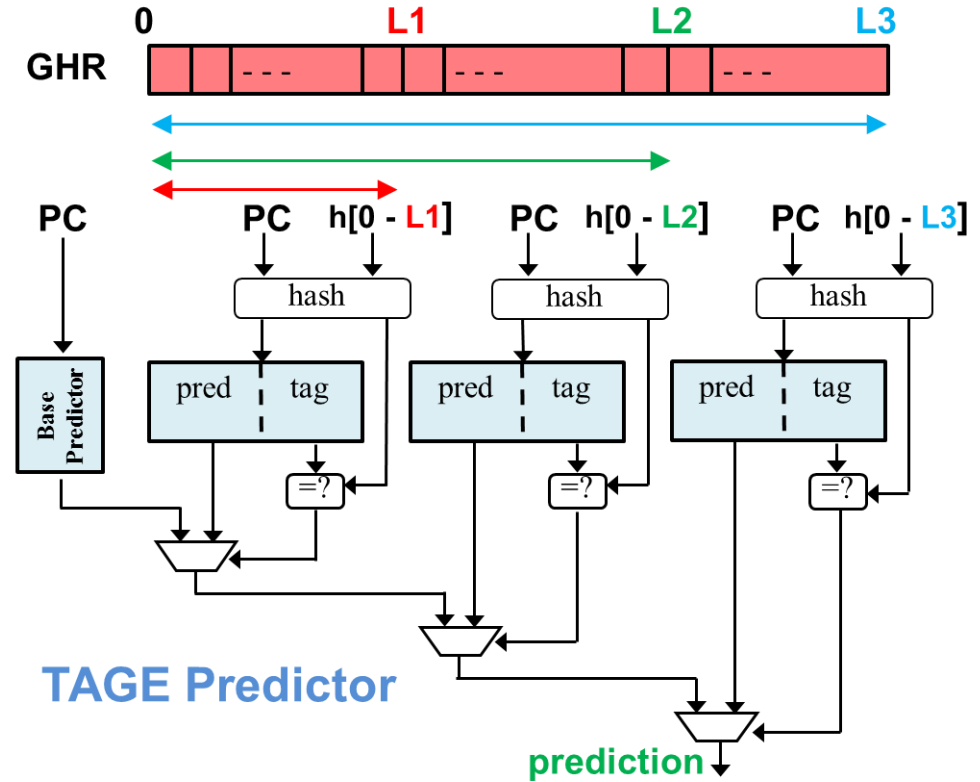
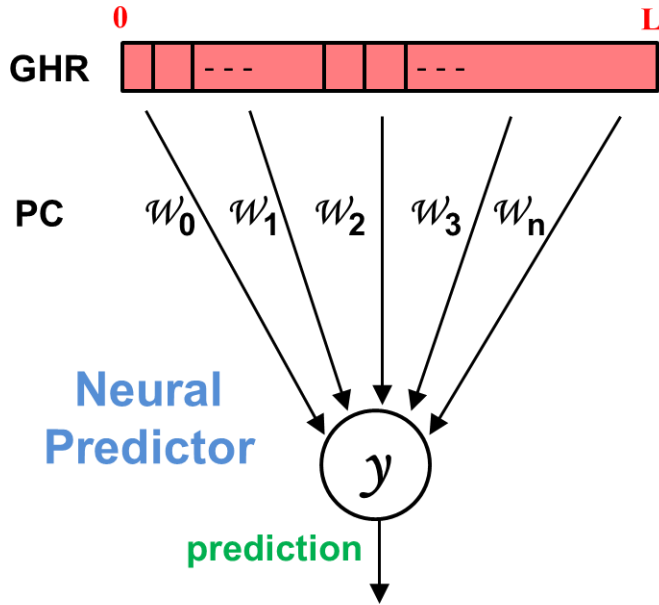
# Bias-Free Branch Predictor



**Dibakar Gope & Mikko H. Lipasti**  
**University of Wisconsin – Madison**

**MICRO 2014**

# Why Another Branch Predictor?



Correlations to **~256** branches

Access to **several** tables

Predictor consumes **~15%** of core energy (ARM's Cortex A15)

# Our Solution: Filter **Useless** Context Out

GHR:



Bias-Free Pred:

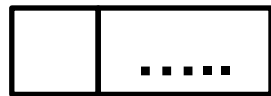


# Our Solution: Filter **Useless** Context Out

GHR:



BF-GHR:



Bias-Free Pred:



compressed

smaller

Expand the effective reach of fixed length GHR

# Our Solution: Bias-Free Branch Predictor



**5-6%**  
Improvement  
over OH-SNAP

Neural



TAGE-like  
accuracy w/  
**fewer** tables

TAGE

# Our Solution: Bias-Free Branch Predictor



**5-6%**  
Improvement  
over OH-SNAP

Neural



TAGE-like  
accuracy w/  
**fewer** tables

TAGE

- What is **useless** information?
- How prevalent is it?
- How to make it work

**Session 6A**

**WEDNESDAY**  
**@ 1:25 PM**

*next paper*



# Loop-Aware Memory Prefetching Using Code-Block Working Sets

*\*Prefetch in Bulk and Listen to Your Mother*

Adi Fuchs

Shie Mannor

Uri Weiser

**Yoav Etsion**

*Electrical Engineering*

*Computer Science*

*Technion – Israel Institute of Technology*



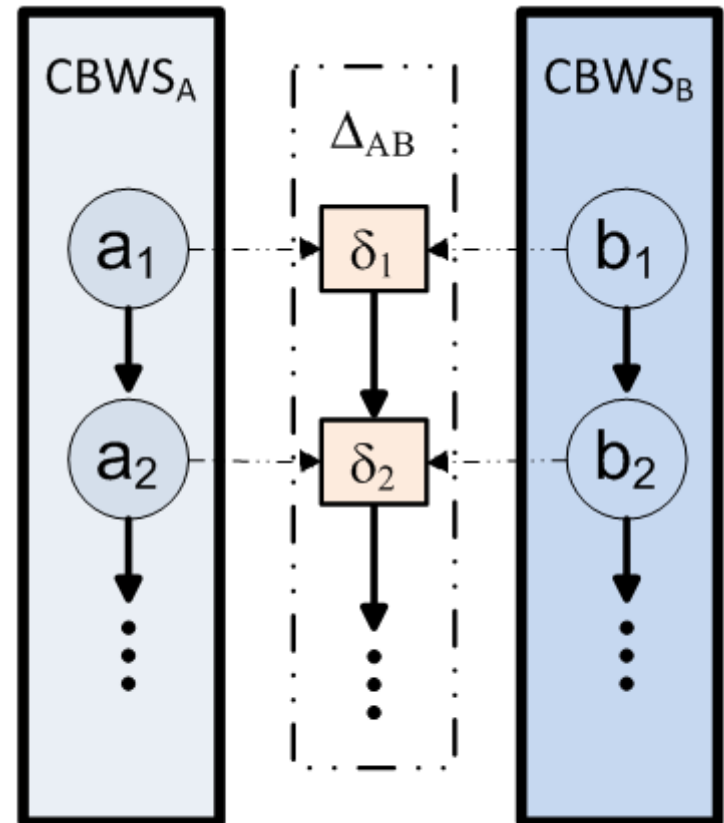
**TECHNION**  
Israel Institute of Technology



# Prefetch Aggressively on Tight Loops

*(i.e., Prefetch in Bulk)*

- **Observation:**  
Working sets of tight loop iterations (CBWS) are highly interdependent
- **Means:**  
Vector arithmetic to compute CBWS differential vectors
- **Objective:**  
Prefetch complete CBWS when possible



$$\Delta_{AB} = \{\delta_i \mid \delta_i = b_i - a_i \text{ for each } i\}$$

# The Compiler Identifies Tight Loops

*(i.e., Listen to Your Mother)*

“The compiler is your mother”

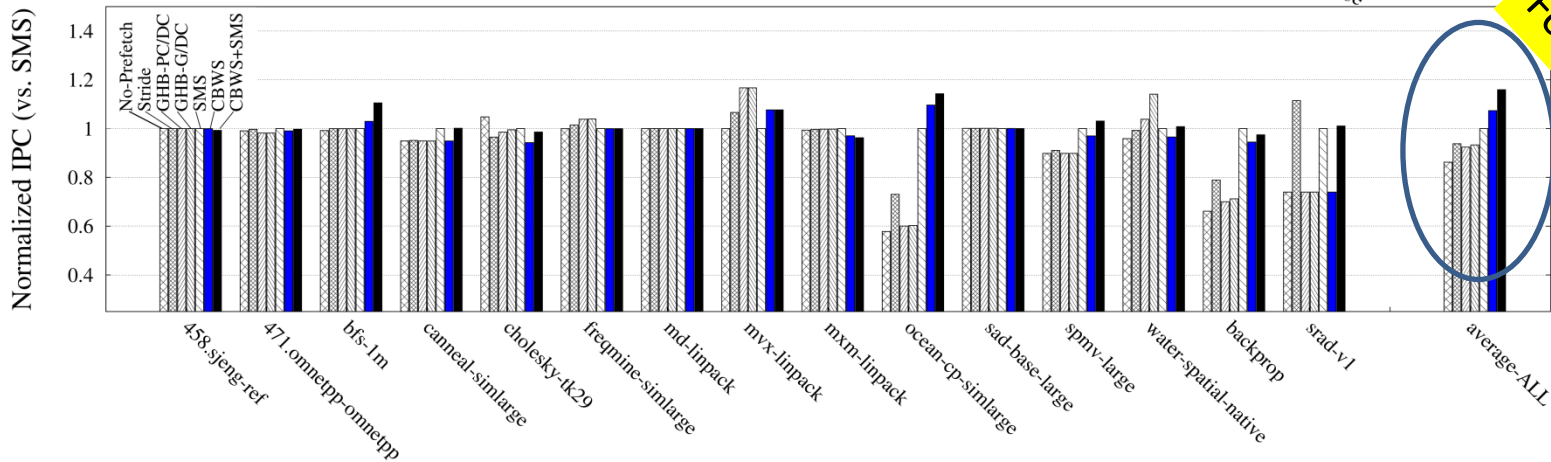
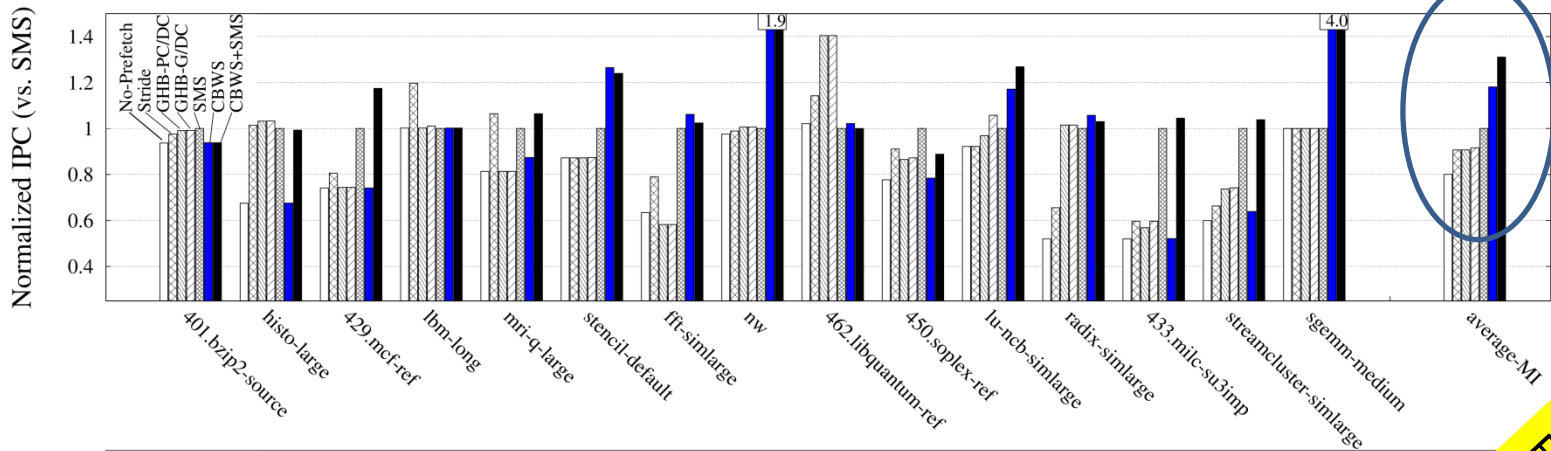
Y. Patt

- The compiler tells us when to use tight loop prefetcher
  - Otherwise, fallback to high-performance SMS prefetcher

```
for( ... ) {  
    for( ... ) {  
        BLOCK_BEGIN(0);  
  
        ...  
  
        BLOCK_END(0);  
    }  
}
```

# Speedup

Over 1.3x over SMS  
for memory intensive  
benchmarks

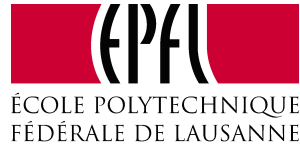


Almost 1.2x  
For SPEC2006

*next paper*

# BuMP: Bulk Memory Access Prediction and Streaming

Stavros Volos, Javier Picorel, Babak Falsafi, Boris Grot



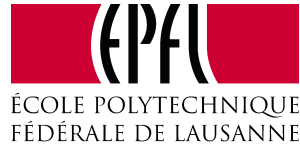
# BuMP: Bulk Memory Access Prediction and Streaming

Stavros Volos, Javier Picorel, Babak Falsafi, Boris Grot



# BuMP: Bulk Memory Access Prediction and Streaming

Stavros Volos, Javier Picorel, Babak Falsafi, Boris Grot



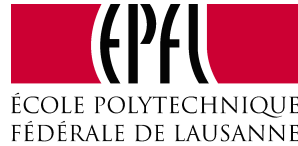
## Datacenters: The Workhorses of Information Age





# BuMP: Bulk Memory Access Prediction and Streaming

Stavros Volos, Javier Picorel, Babak Falsafi, Boris Grot

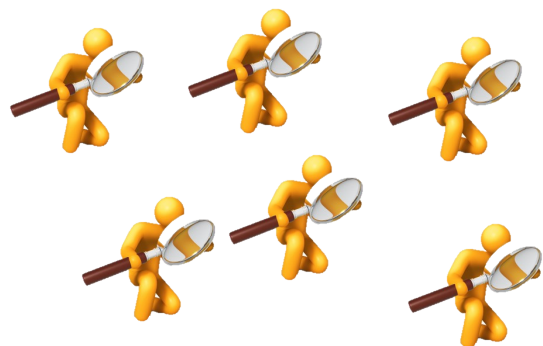


## Datacenters: The Workhorses of Information Age

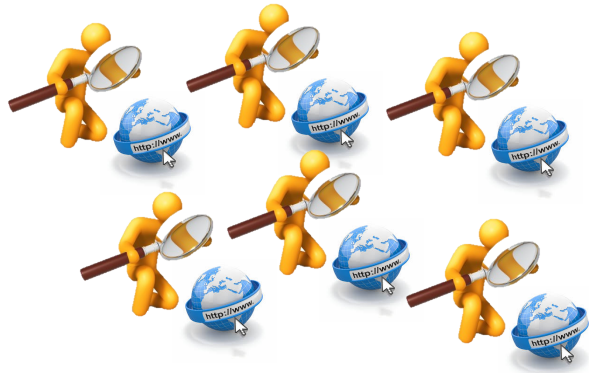


Electricity bills of \$25 Billion per year

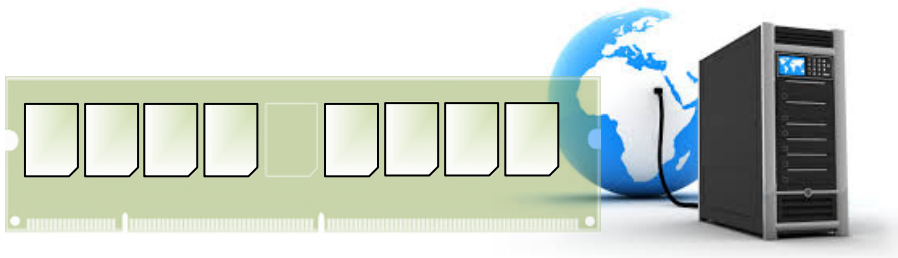
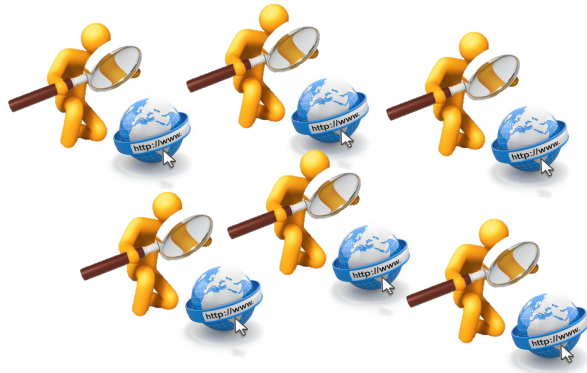
# User Requests



# User Requests are Data-Intensive

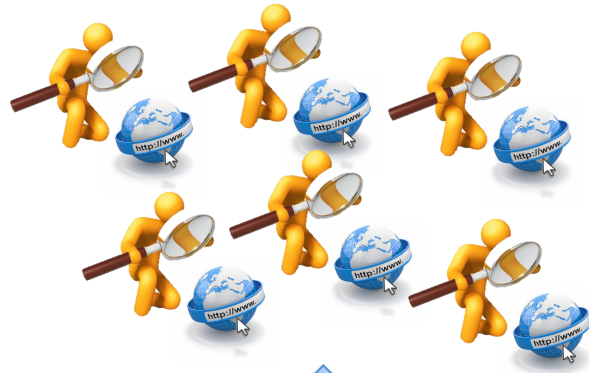


# User Requests are Data-Intensive

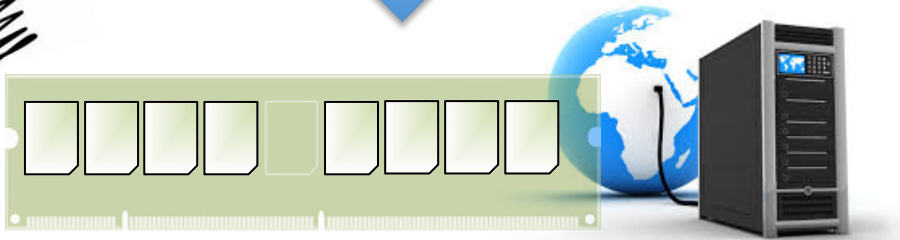
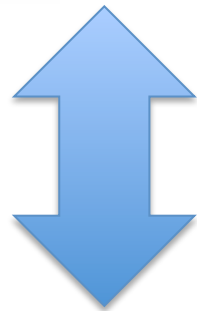


Vast DRAM-resident datasets

# User Requests are Data-Intensive



DRAM serves many requests



Vast DRAM-resident datasets

**DRAM: Major energy hog in datacenters**



# Datasets Have Different Flavors

*videos*



*twitter*

*posts*



*images*



*database*

*index  
pages*



*rows*



# Datasets Have Different Flavors

*videos*



*twitter*

*posts*



*images*



*database*

*index  
pages*



*rows*  
amazon.com





# Datasets Have Different Flavors

**MBS**

*videos*

You **Tube**™

*images*



*index  
pages*

bing™

Google™

**KBS**



twitter

*posts*



**KBS**

*database*

*rows*

amazon.com

ebay™

**KBS**

# Datasets Have Different Flavors

**MBS**

*videos*

You **Tube**™

twitter

*posts*



**KBS**

*images*



*index*

bing

*pages*

*database*

*rows*

ebay

**KBS**

Google™

amazon.com

**KBS**

Common: DRAM is accessed in bulk

# BuMP

## Bulk Memory Access Prediction and Streaming

*Prediction:* Identify bulk accesses

# BuMP

## Bulk Memory Access Prediction and Streaming

*Prediction:* Identify bulk accesses

*Streaming:* Trigger bulk transfers to exploit **locality**

# BuMP

## Bulk Memory Access Prediction and Streaming

For a 16-core server running datacenter applications

**23% lower DRAM energy**  
**11% higher throughput**

# BuMP

## Bulk Memory Access Prediction and Streaming

For a 16-core server running datacenter applications

**23% lower DRAM energy**  
**11% higher throughput**

**Session 6A, Wednesday at 2:15 PM**

*next paper*

# Protean Code: Achieving Near-free Online Code Transformations for Warehouse Scale Computers

**Michael A. Laurenzano**

Yunqi Zhang

Lingjia Tang

Jason Mars



Datacenter



Mobile



Desktop

# Dynamism is everywhere

Apps begin and end

Program phases

User behavior varies

Unreliable hardware

Datacenter



Mobile



Desktop

# Dynamism is everywhere

Apps begin and end

Program phases

User behavior varies

Unreliable hardware

Native code should change with the environment

Datacenter



Mobile



Desktop

# Dynamism is everywhere

Apps begin and end

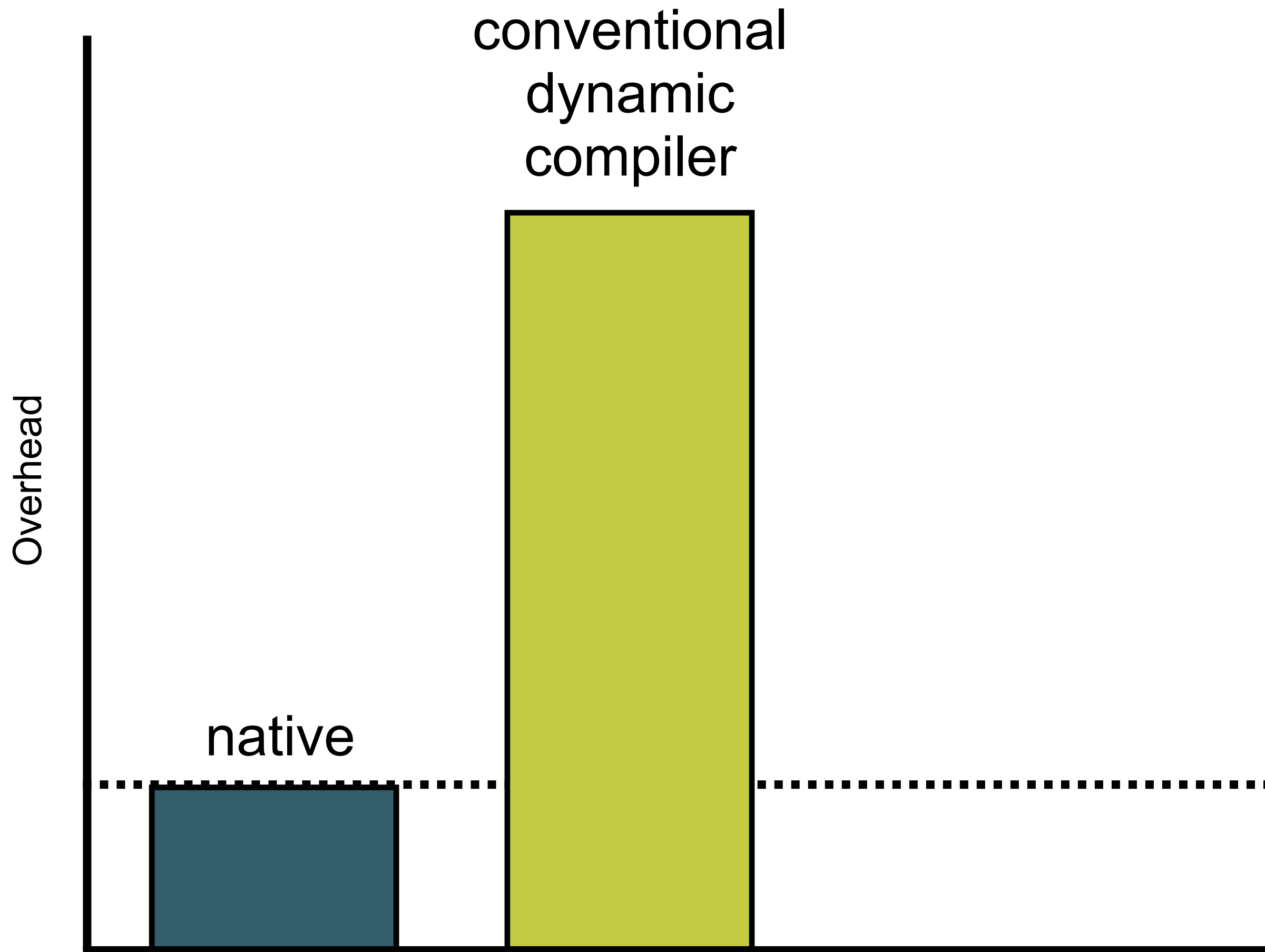
Program phases

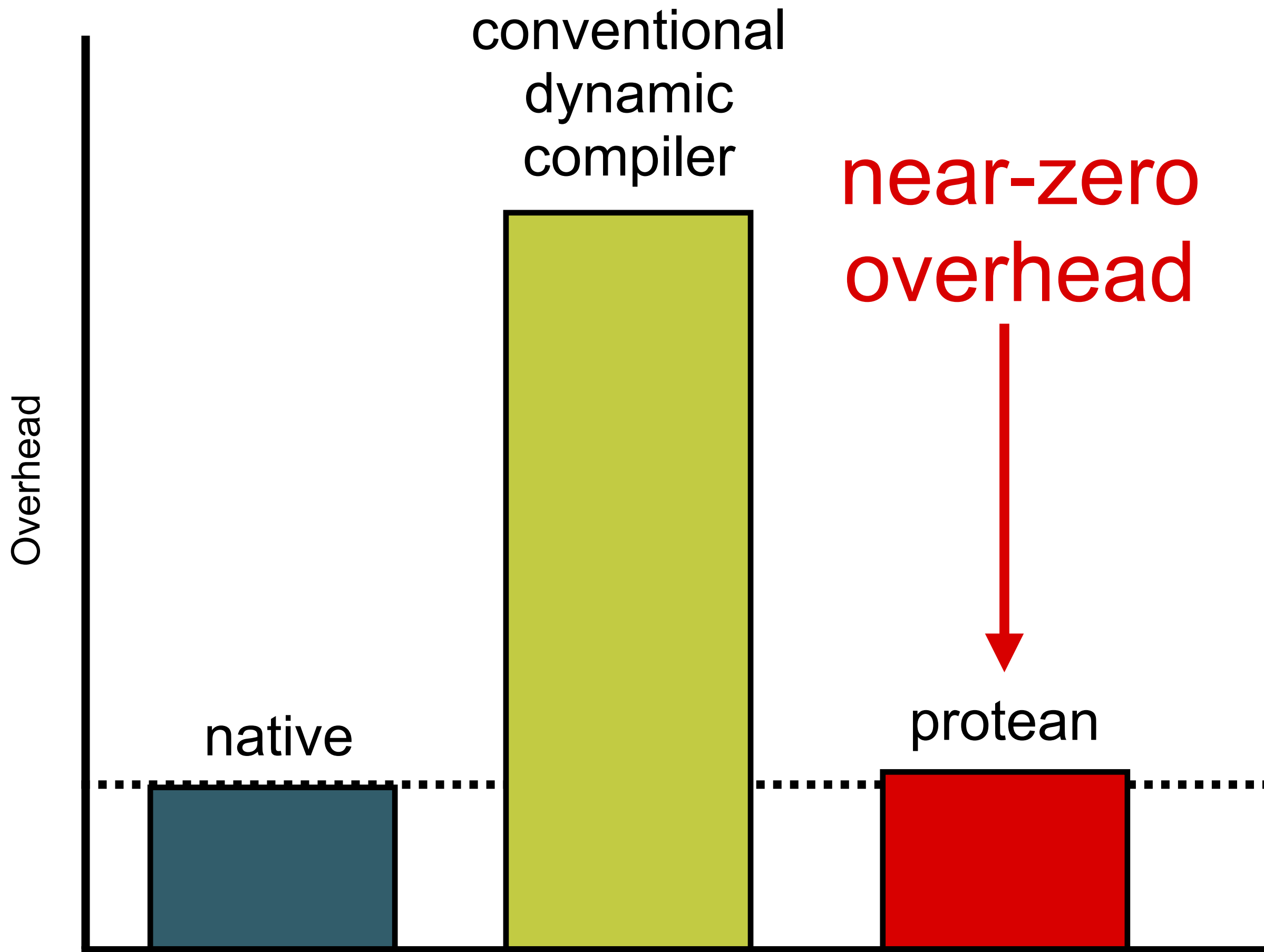
User behavior varies

Unreliable hardware

Native code should change with the environment

Not possible today in production environments





# Protean code is a breakthrough

- Compilation is asynchronous with near-zero overhead
- Dynamic code optimization is always available
- Static compilation choices do not have to be permanent

## Come to my talk to hear about

- A new paradigm for thinking about compilation
- A fully functional, open source dynamic compiler infrastructure implemented on top of LLVM
- A novel dynamic optimization that reduces the # of servers in the datacenter by > 25%

# Protean code is a breakthrough

- Compilation is asynchronous with near-zero overhead
- Dynamic code optimization is always available
- Static compilation choices do not have to be permanent

## Come to my talk to hear about

- A new paradigm for thinking about compilation
- A fully functional, open source dynamic compiler infrastructure implemented on top of LLVM
- A novel dynamic optimization that reduces the # of servers in the datacenter by > 25%

**Session 6B, Wednesday 1:00pm**

*next paper*



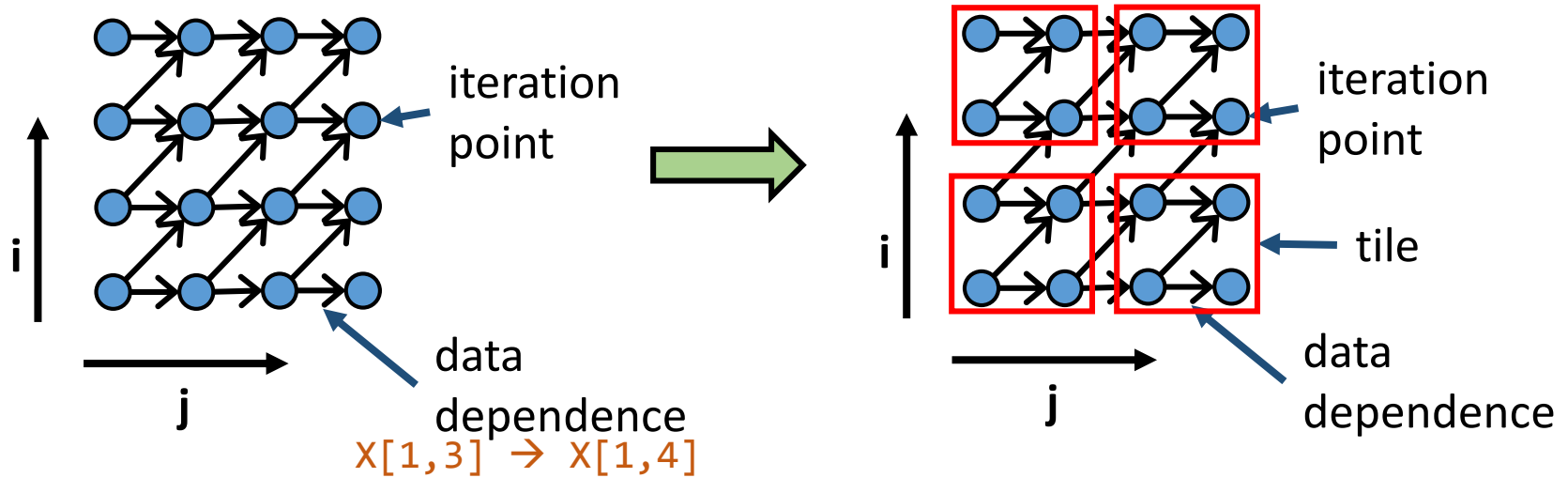
# Compiler Support for Optimizing Memory Bank Level-Parallelism

Wei Ding, Diana Guttman, and Mahmut Kandemir  
The Pennsylvania State University

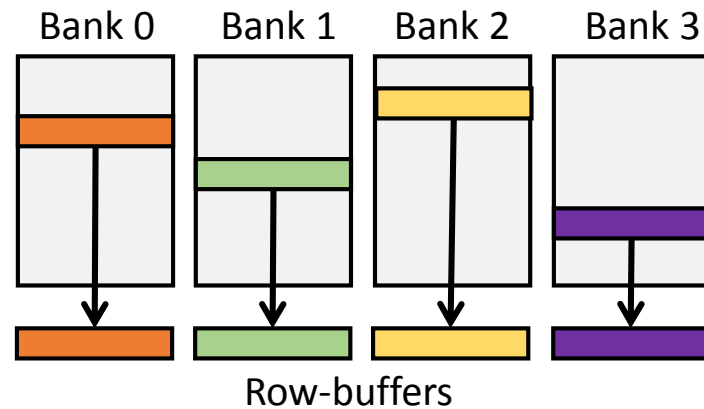
- Last-level cache misses (memory accesses) are important for good performance, but current compilers focus on cache locality only
  - Bank-level parallelism of LLC misses is critical to performance
- How can a compiler optimize for **bank-level parallelism** on a multicore processor?

# Loop Tiling

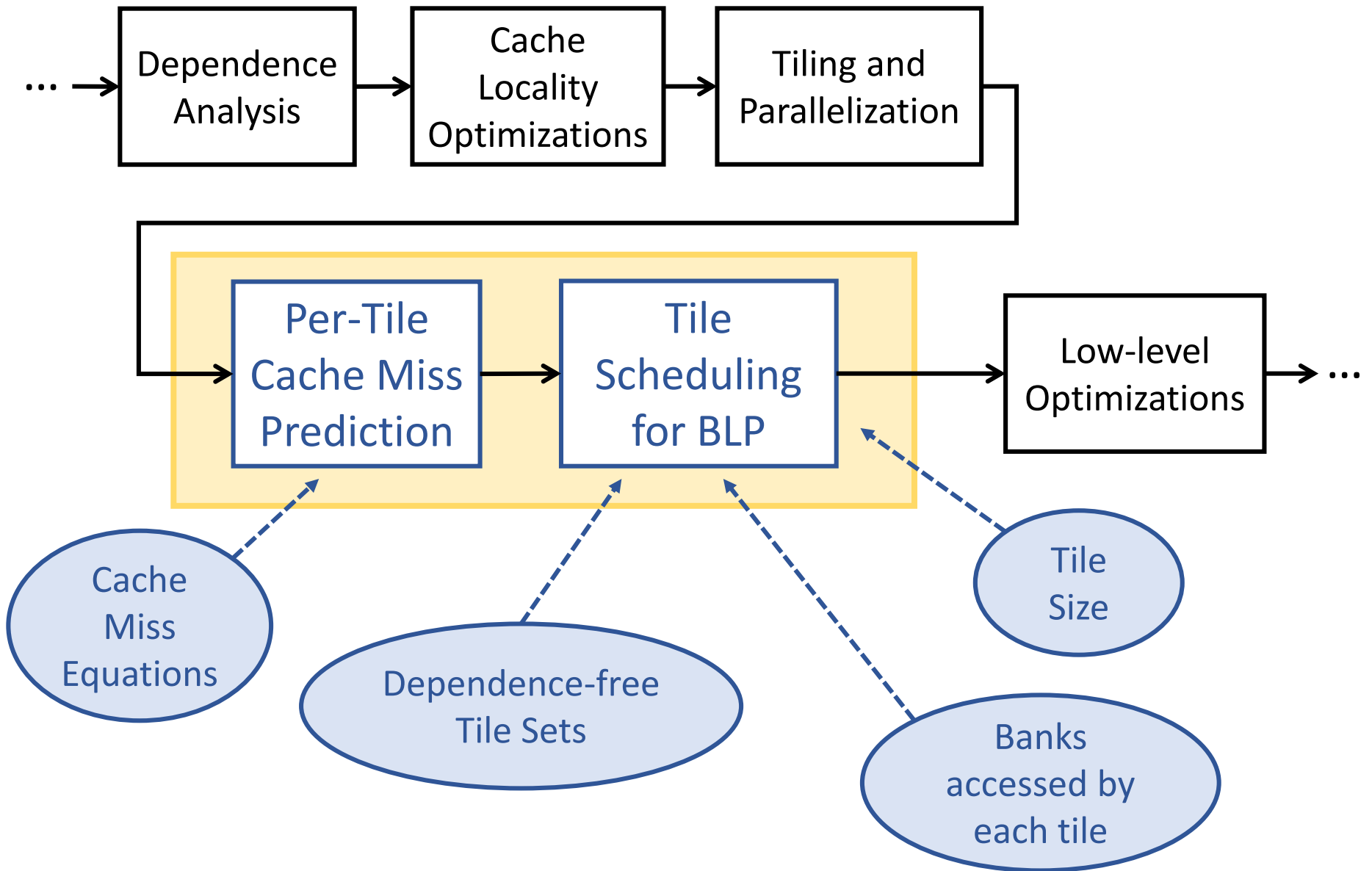
```
for i = 1..N  
  for j = 1..N  
    X[i,j] = X[i,j-1] + X[i-1,j-1]
```



# Bank-Level Parallelism



Compiler Support for Optimizing Memory Bank Level-Parallelism



Compiler Support for Optimizing Memory Bank Level-Parallelism

# Compiler Support for Optimizing Memory Bank Level-Parallelism

Average **bank-level parallelism** improvement of **17.1%**

Average **memory access latency** reduction of **9.2%**

Please come to the full-length presentation!

- Extensions to consider **memory controller-level parallelism** and **row-buffer locality**
- Algorithm details
- Detailed results

Session 6B: Compilation  
and Code Generation  
Wednesday, December 17  
13:00 - 14:40  
Room: Umney Theatre

*next paper*

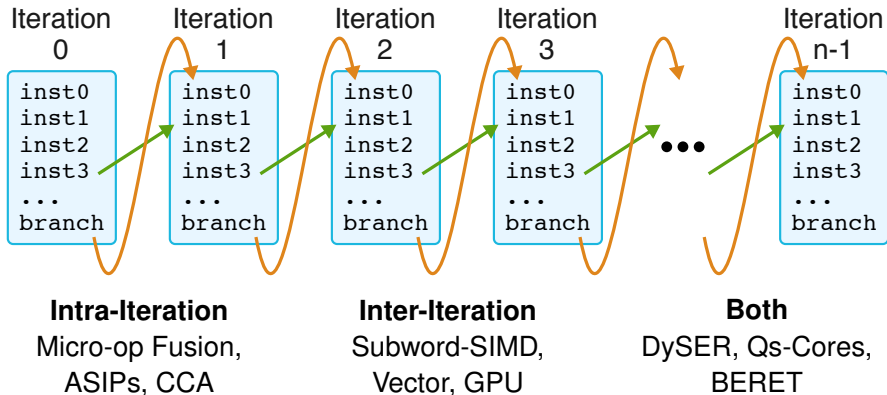
# Architectural Specialization for Inter-Iteration Loop Dependence Patterns

Shreeshha Srinath, Berkin Ilbeyi, Mingxing Tan, Gai Liu  
Zhiru Zhang, Christopher Batten

Computer Systems Laboratory  
School of Electrical and Computer Engineering  
Cornell University

47th Int'l Symp. on Microarchitecture, Dec 2014  
**Session 6B: Compilation and Code Generation**

# Loop Dependence Pattern Specialization



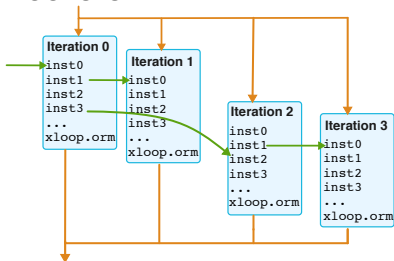
Key Challenge: Creating HW/SW abstractions that are flexible and enable performance-portable execution

```

loop:
  lw      r4, 0(r3)
  lw      r5, 0(rA)
  mul     r6, r4, r5
  addu   rX, r6, rX
  sw      rX, 0(r3)
  addiu.xi r3, 4
  addiu.xi rA, 4
  addiu   r1, r1, 1
  xloop.orm r1, rN, loop

```

## XLOOPS ISA



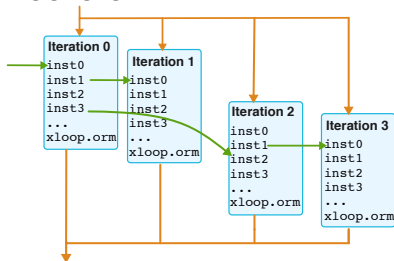


```

loop:
  lw      r4, 0(r3)
  lw      r5, 0(rA)
  mul     r6, r4, r5
  addu   rX, r6, rX
  sw     rX, 0(r3)
  addiu.xi r3, 4
  addiu.xi rA, 4
  addiu  r1, r1, 1
  xloop.orm r1, rN, loop

```

## XLOOPS ISA



## XLOOPS Compiler

```

#pragma xloop ordered
for ( X=0, i=K; i<N; i++ )
{
  A[i] = A[i] * A[i-K];
  X += A[i];
}

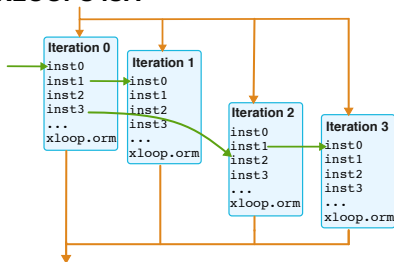
```

```

loop:
  lw      r4, 0(r3)
  lw      r5, 0(rA)
  mul     r6, r4, r5
  addu   rX, r6, rX
  sw      rX, 0(r3)
  addiu.xi r3, 4
  addiu.xi rA, 4
  addiu   r1, r1, 1
  xloop.orm r1, rN, loop

```

## XLOOPS ISA



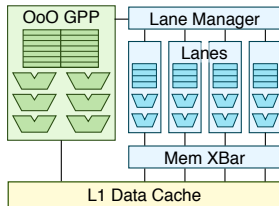
## XLOOPS Compiler

```

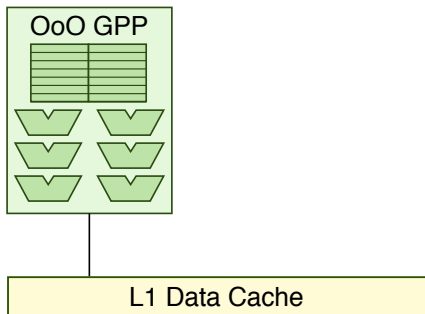
#pragma xloop ordered
for ( X=0, i=K; i<N; i++ )
{
  A[i] = A[i] * A[i-K];
  X += A[i];
}

```

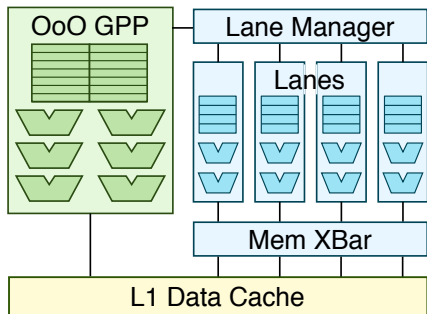
## XLOOPS Microarchitecture



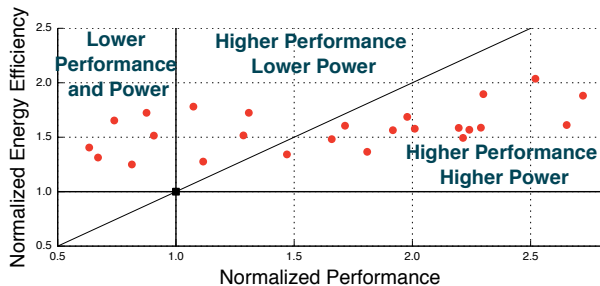
**Single-ISA heterogenous architecture** that transparently integrates traditional processors and specialized loop-accelerators

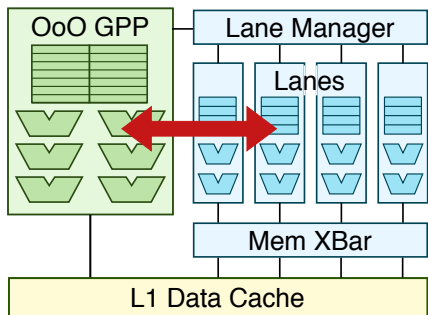


- ▶ **Traditional Execution**  
Speedups close to  $1\times$

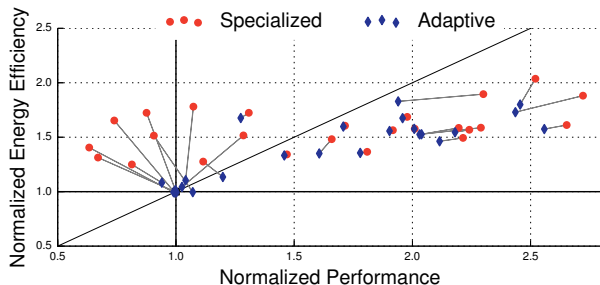


- ▶ **Traditional Execution**  
Speedups close to  $1\times$
- ▶ **Specialized Execution**  
Speedups  $1.25\text{--}2.5\times$   
Energy Efficiency  $1.5\text{--}3\times$





- ▶ **Traditional Execution**  
Speedups close to  $1\times$
- ▶ **Specialized Execution**  
Speedups  $1.25\text{--}2.5\times$   
Energy Efficiency  $1.5\text{--}3\times$
- ▶ **Adaptive Execution**  
Dynamically Trade  
Performance vs. Energy Efficiency



*next paper*

# Specializing Compiler Optimizations Through Programmable Composition For Dense Matrix Computations

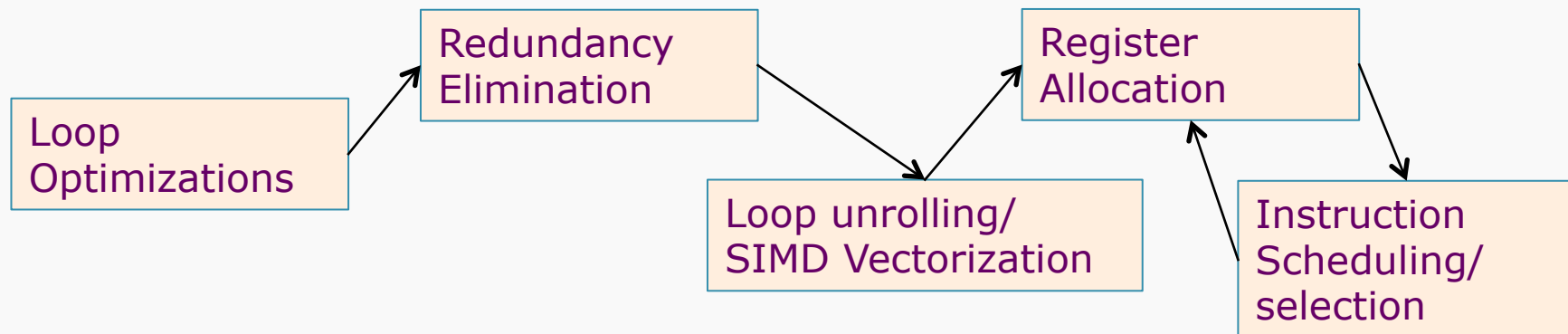
Qing Yi, Qian Wang, Huimin Cui

University of Colorado, Colorado Springs, USA

Institute Of Software & Institute of Computing, Chinese  
Academy of Science

# Motivation

- **What's wrong with general purpose compilers?**
  - Target all possible user applications
    - Try to attain the best average performance
    - Inferior to manual specialized optimizations
- **Independent optimization passes**

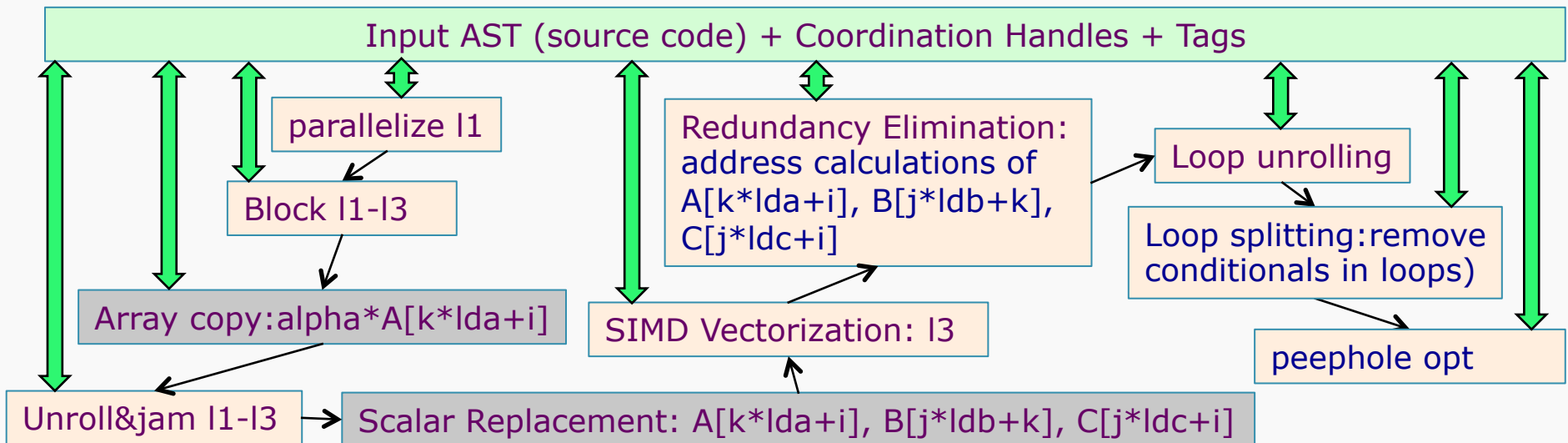


- Unpredictable Interferences among optimizations
  - Information loss from re-analyzing optimized code
  - Best optimization order is NP-complete

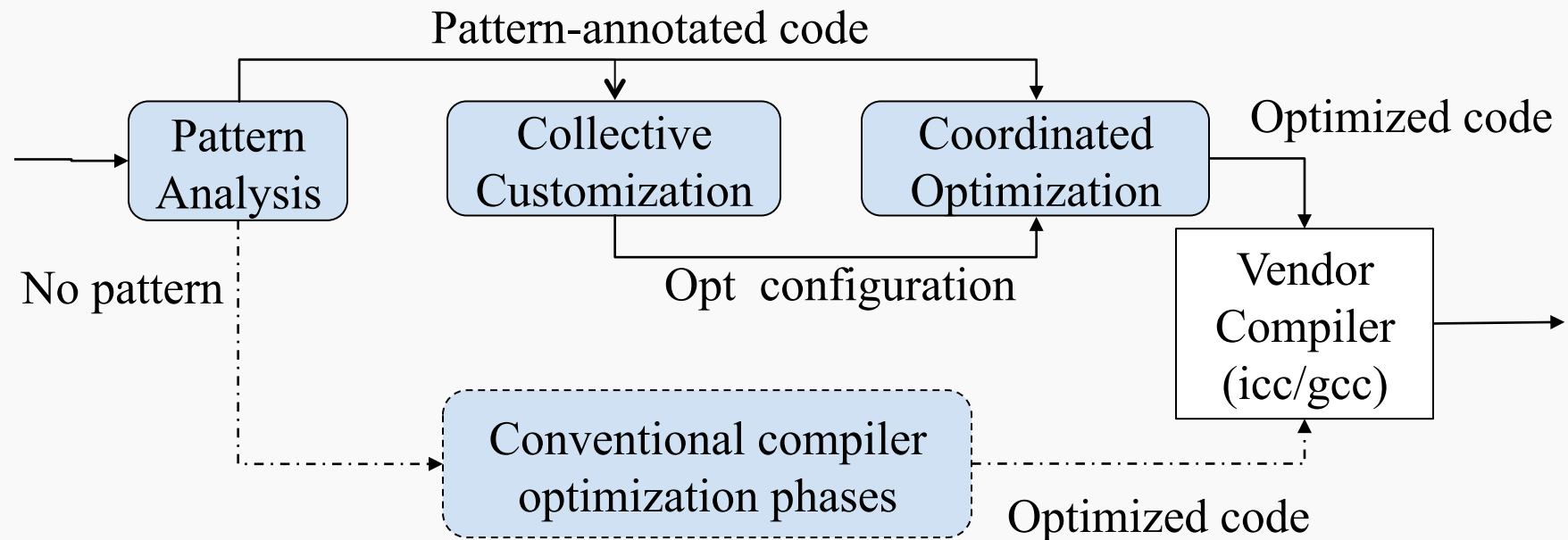


# Overcoming The Uncertainties

- **Programmable composition of compiler optimizations**
  - Eliminate optimization interferences
    - Analyze the original input source code only once
    - Enable fine-grained coordination among optimization passes
  - Pattern-based Specialization
    - Recognize known computational patterns
    - Specialize optimization customization and ordering



# Optimization Workflow

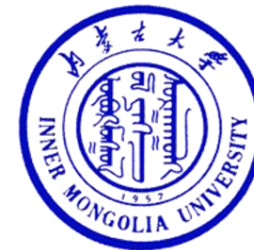


- **Specialized optimization for dense-matrix kernels**
  - Applied to 15 BLAS kernels and 15 applications in SPLASH-2
  - Kernel performance comparable to manual assembly programming

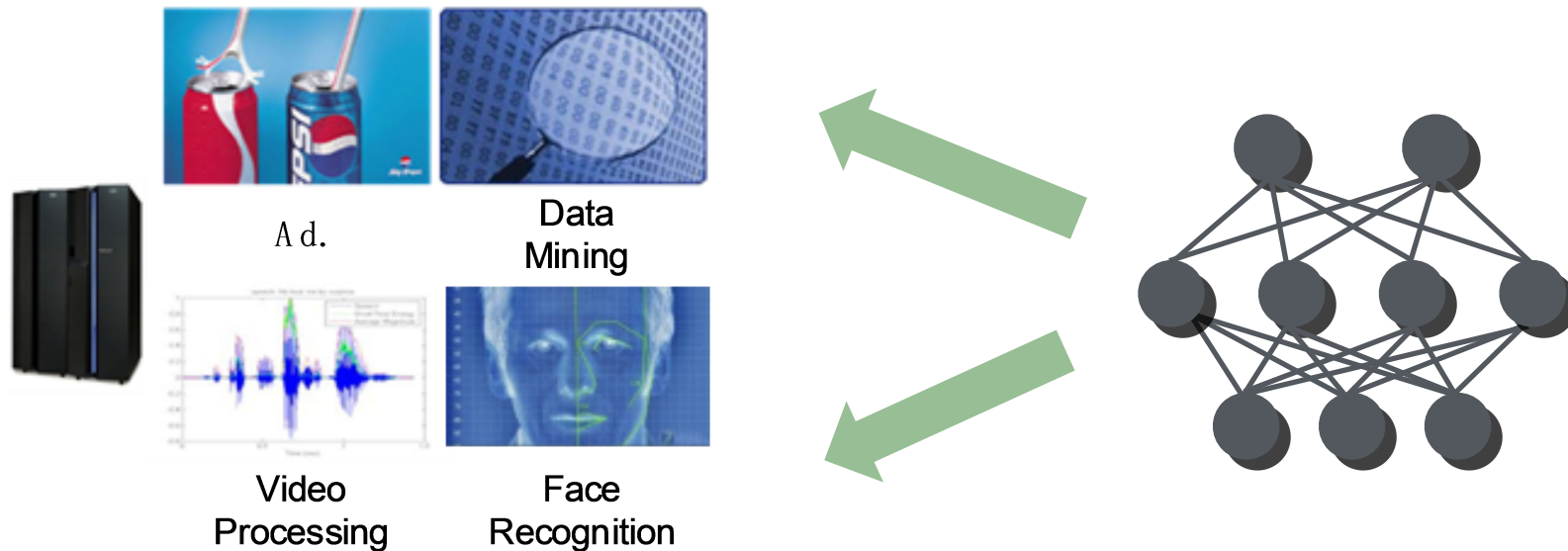
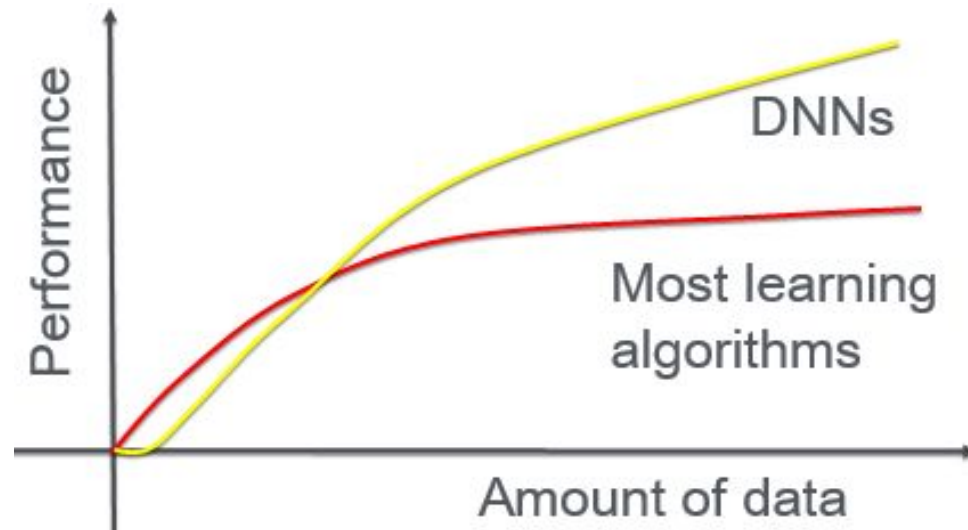
*next paper*

# A Machine-Learning Supercomputer

Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang,  
Liqiang He, Jia Wang, Ling Li, Tianshi Chen,  
Zhiwei Xu, Ninghui Sun, Olivier Temam



# DNN: State-of-the-art Machine-Learning Algorithm



# Hardware for DNNs

# Hardware for DNNs



# Hardware for DNNs





# Hardware for DNNs



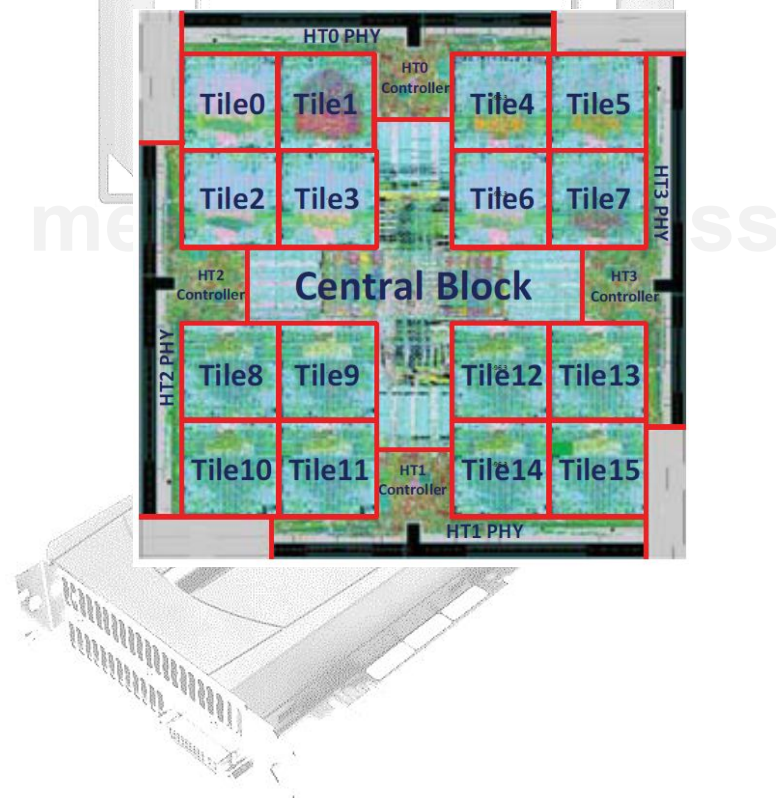
**memory X access**



**GPU**

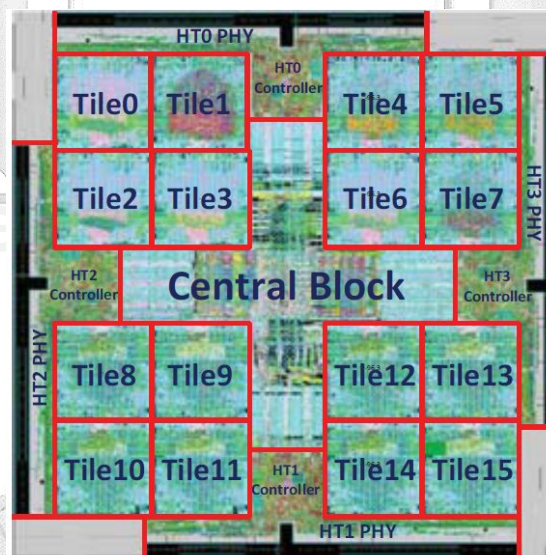
# Hardware for DNNs

# 大电脑



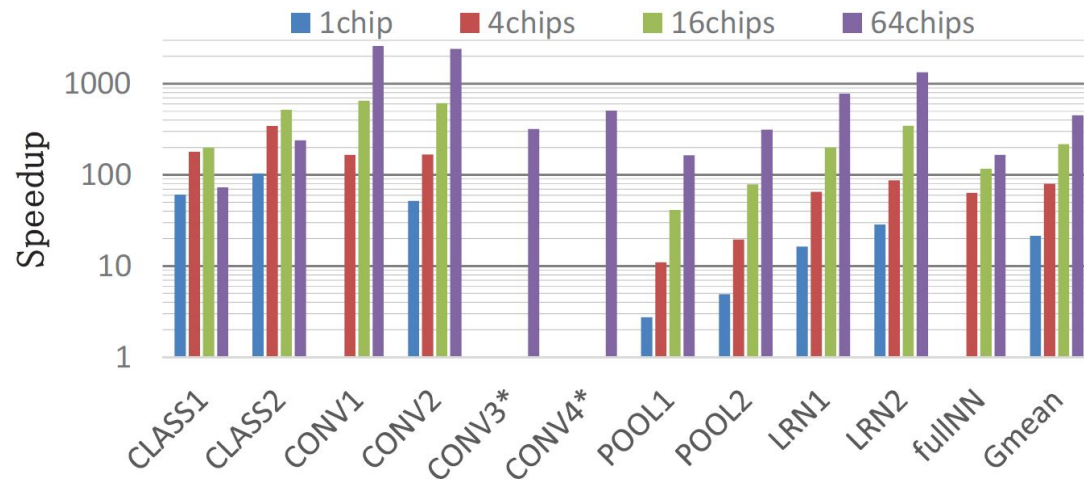
# Hardware for DNNs

# 大电脑

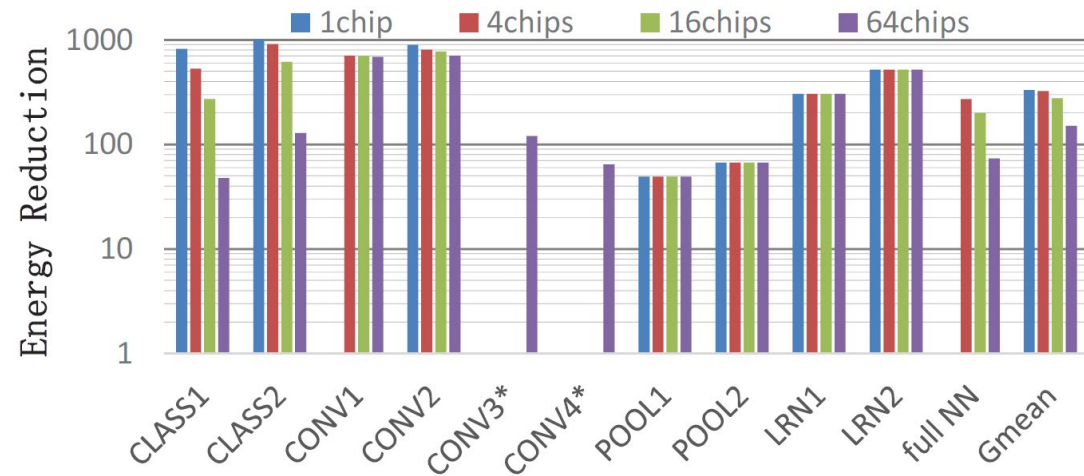


**A Supercomputer for DNNs  
w/o memory access!**

# Experimental Results

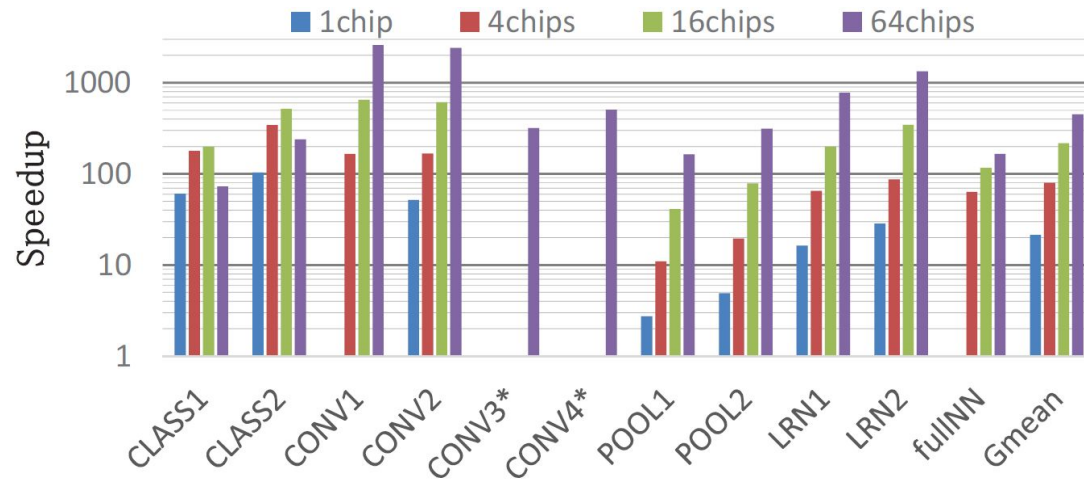


large speedups

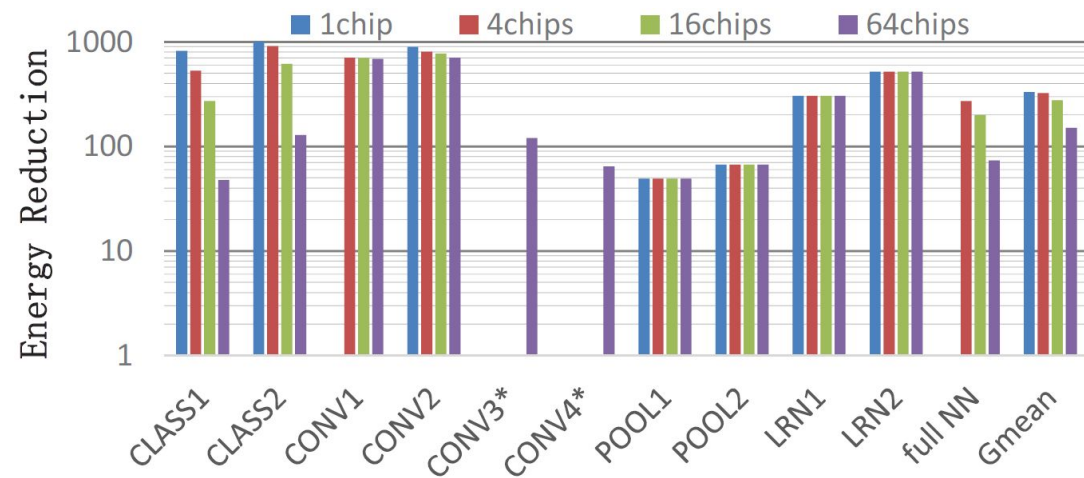


energy saving

# Experimental Results



large speedups



energy saving

See you in session 7

*next paper*

# *B-Fetch*: Branch Prediction Directed Prefetching for Chip-Multiprocessors

David Kadjo<sup>1</sup>, Jinchun Kim<sup>1</sup>, Prabal Sharma<sup>2</sup>, Reena Panda<sup>3</sup>,  
Paul V. Gratz<sup>1</sup>, Daniel Jiménez<sup>4</sup>

<sup>1</sup> Department of Electrical and Computer Engineering, Texas A&M University

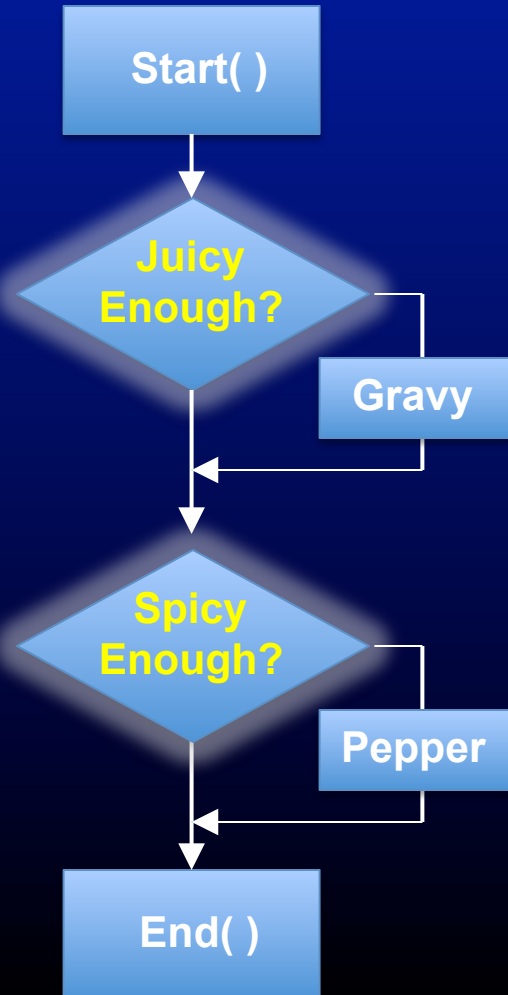
<sup>2</sup> Samsung R&D, Austin <sup>3</sup> Department of Electrical and Computer Engineering, University of Texas, Austin

<sup>4</sup> Department of Computer Science & Engineering, Texas A&M University

# Program: ./make Christmas\_turkey



## Turkey Recipe

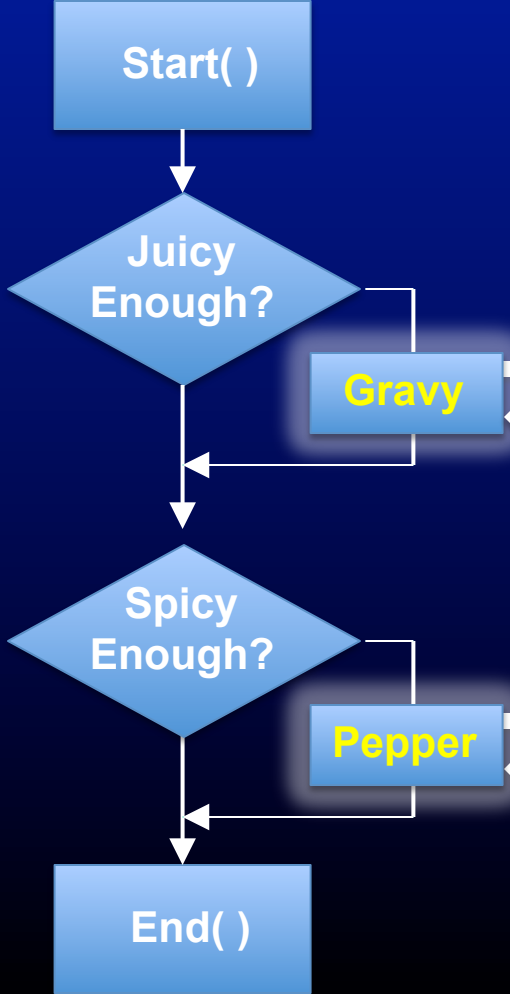


Decisions while cooking change the flavor of the turkey



Program: ./make Christmas\_turkey

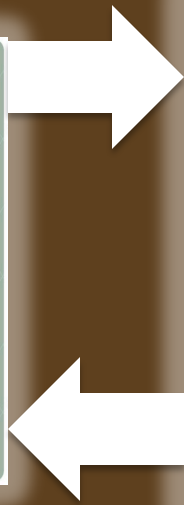
# Turkey Recipe



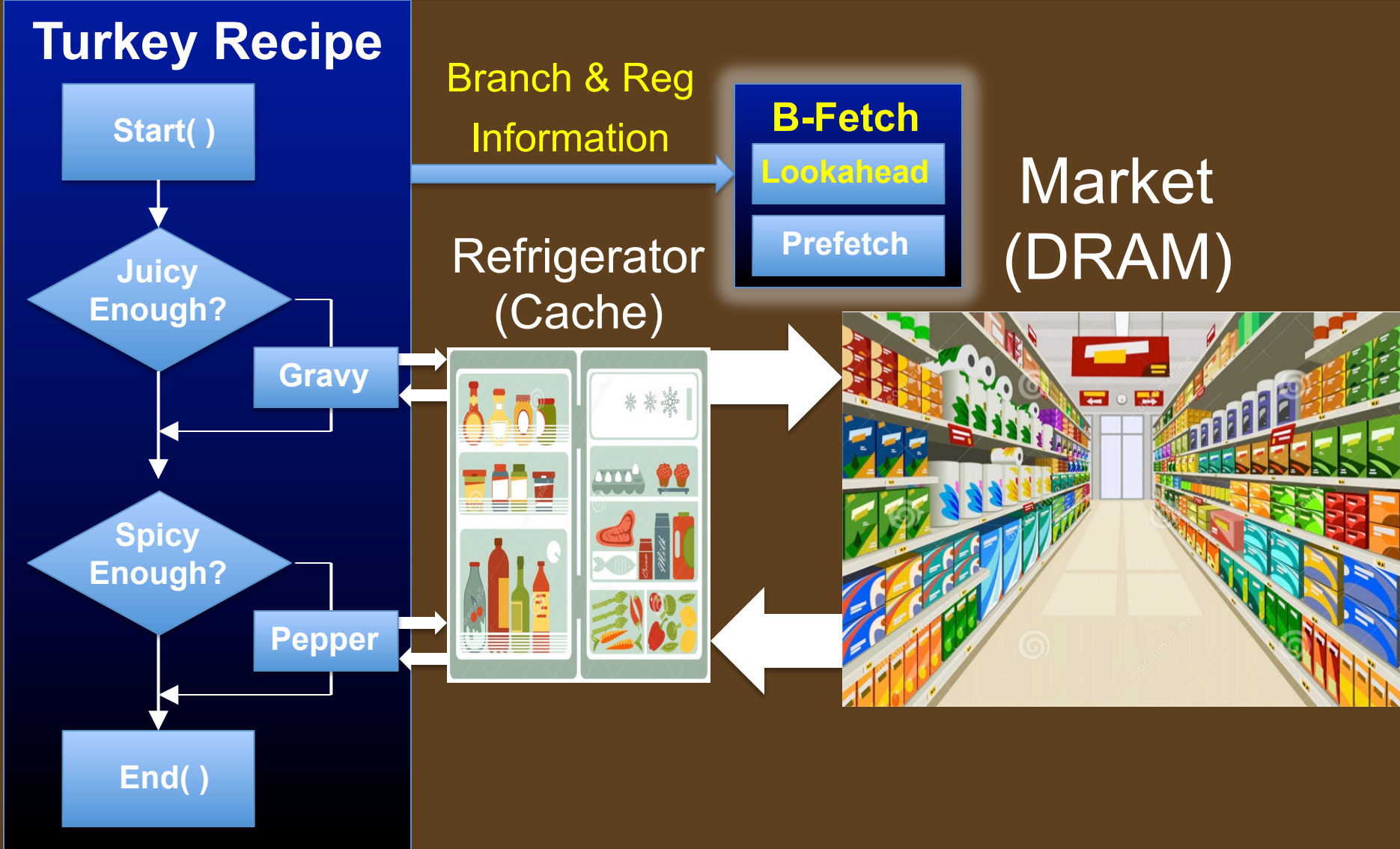
# TOO FAR AWAY

## Market (DRAM)

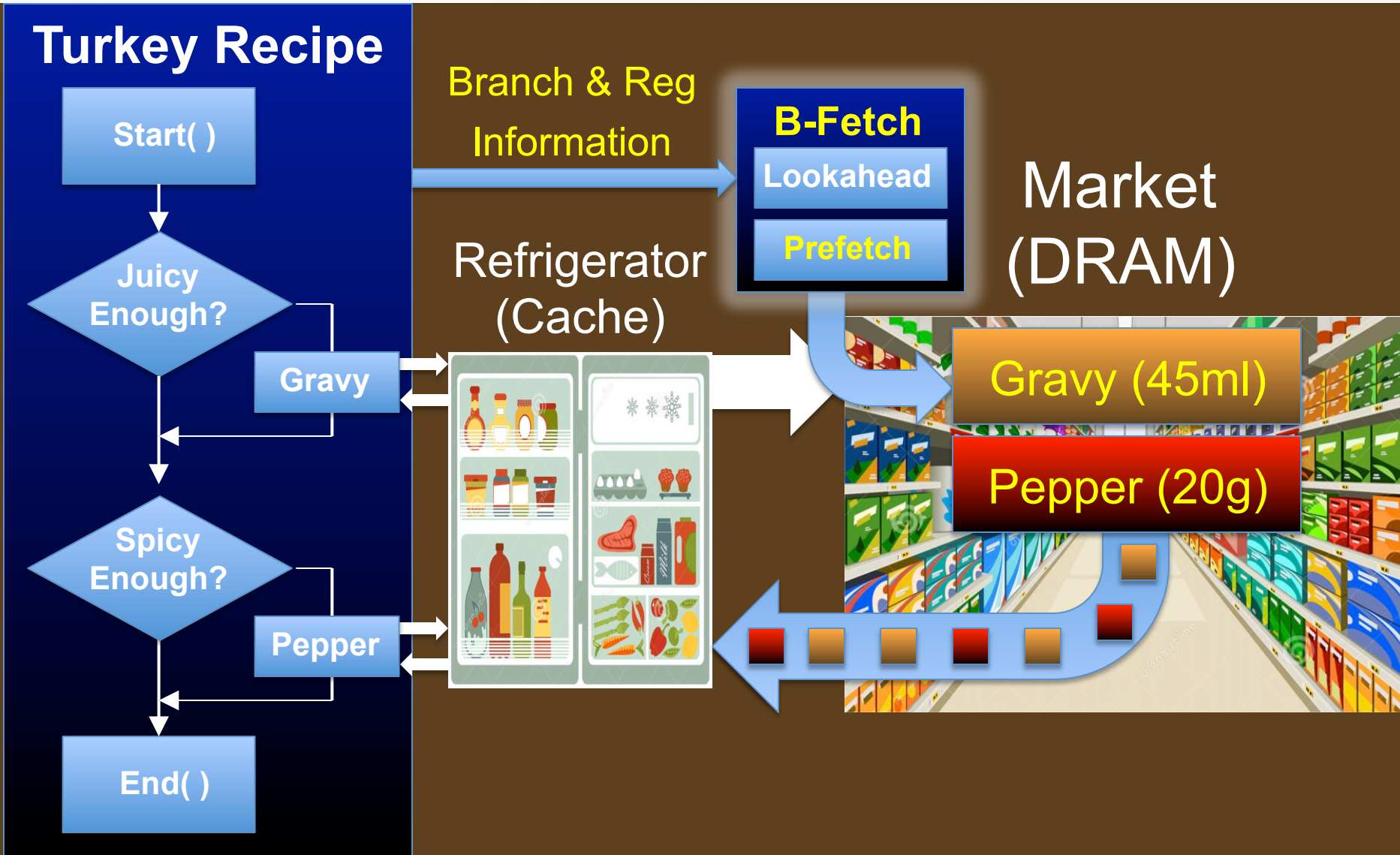
### Refrigerator (Cache)



# Program: ./make Christmas\_turkey



# Program: ./make Christmas\_turkey



# *B-Fetch*: Prefetching based on Branch Prediction



- Path Speculation and Effective Address Speculation
  - Path speculation based on branch lookahead
  - Effective address speculation based on architectural register files
- Light weight prefetcher leverages branch prediction
  - Provide **28% speed-up** compared to the baseline without data prefetching
  - **9% speed-up** and **65% less storage** than SMS [Somogyi 2006]
  - Join discussion at the Best Paper Section!  
**Dec. 17<sup>th</sup> (Wednesday) 3PM at Main Auditorium**

*next paper*

# PipeCheck: Specifying and Verifying Microarchitectural Enforcement of Memory Consistency Models

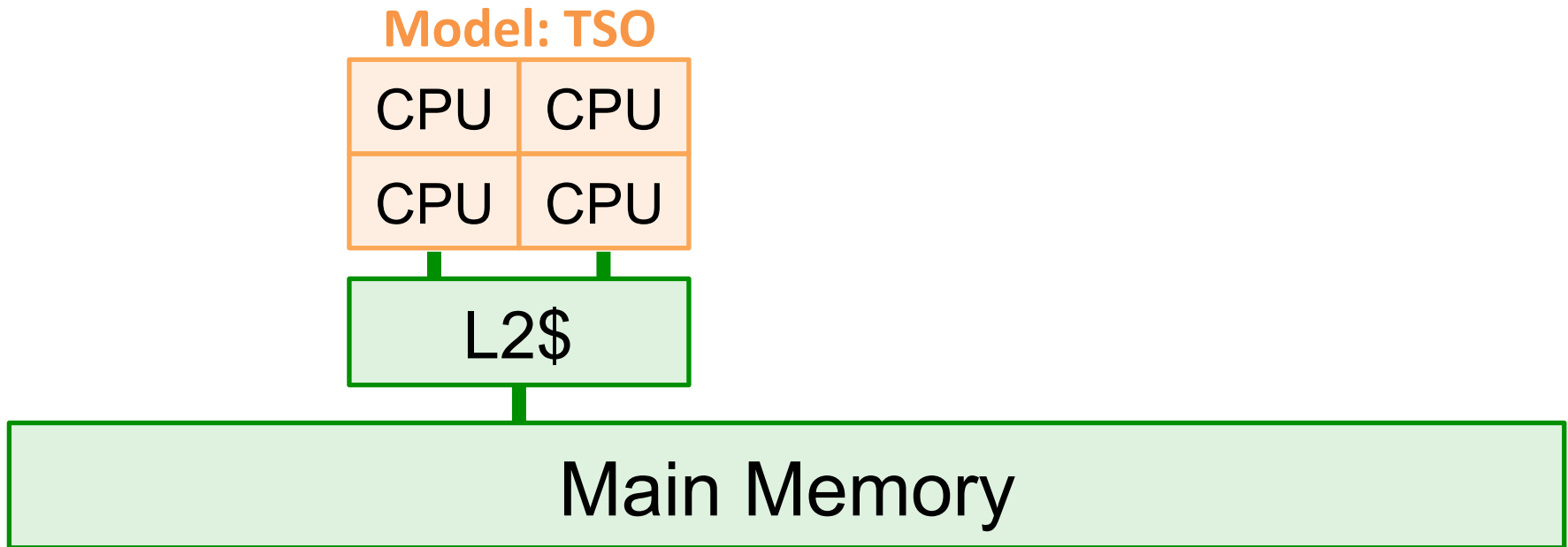
**Daniel Lustig**, Michael Pellauer, Margaret Martonosi  
Princeton University Intel VSSAD

Session 7 (Best Paper Nominees), Paper 3, Wed @ 4pm

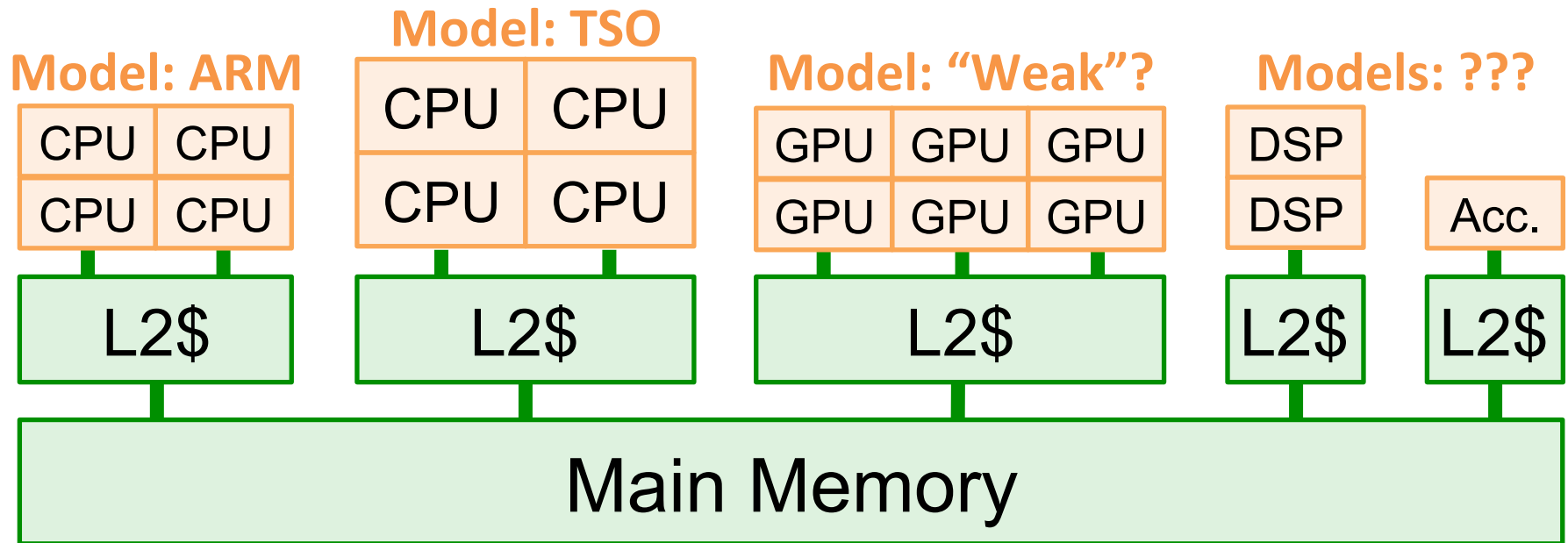


PRINCETON

# Motivation: Verify correctness of memory consistency model **implementation**



# Motivation: Verify correctness of memory consistency model **implementation**



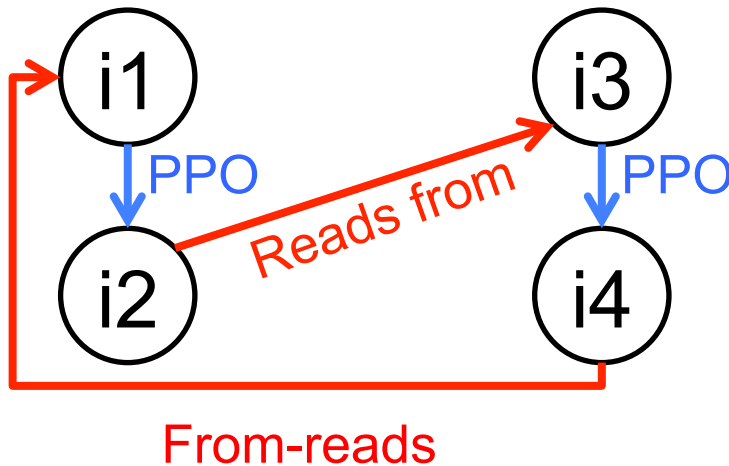
- Heterogeneity → even harder to verify!





# Architecture-Level Analyses Cannot Distinguish Between Implementations

Initially: $[x]=[y]=0$	
Core 0	Core 1
(i1) st $[x]$ , 1	(i3) ld $[y] \rightarrow r1$
(i2) st $[y]$ , 1	(i4) ld $[x] \rightarrow r2$
TSO: <b>Forbid</b> : $r1=1, r2=0$	

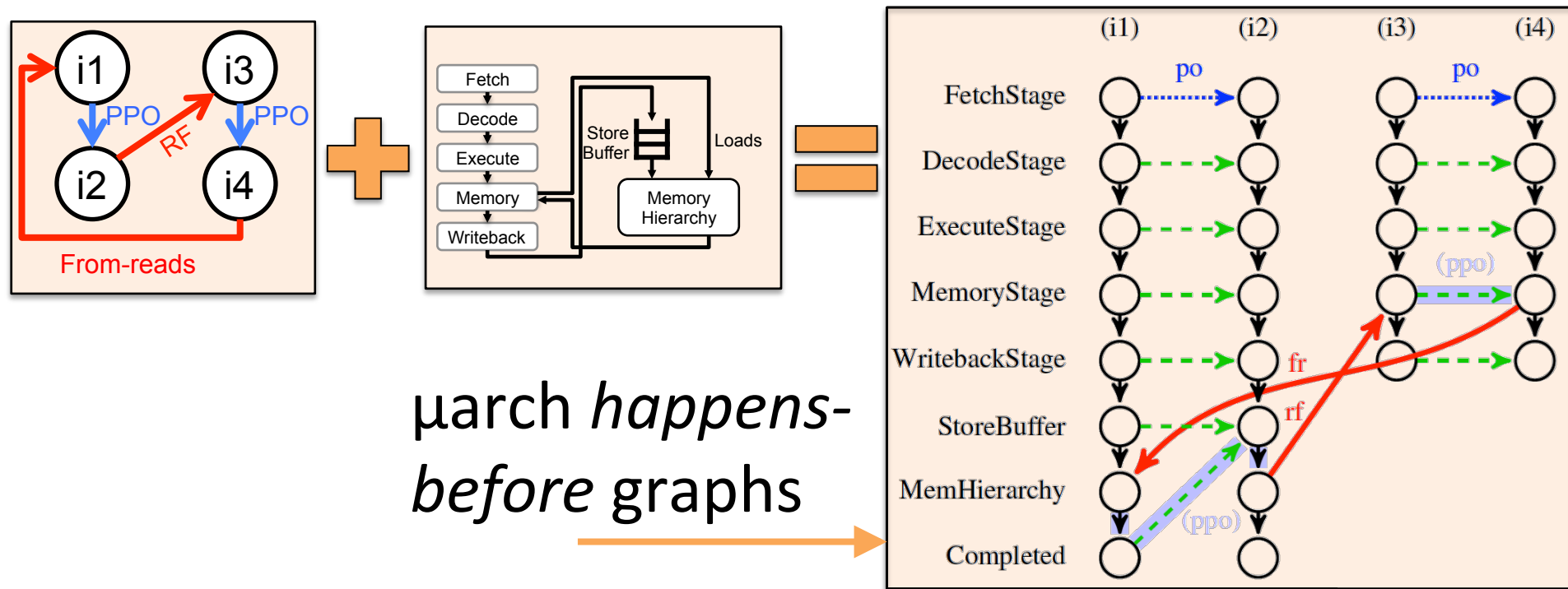


- Arch.-level models cannot analyze/verify the behavior of:
  - Out-of-Order Execution
  - Speculative load reordering
  - Other  $\mu$ arch. optimizations

[Alglave, FMSD '12, Owens et al., TPHOLs '09]



# PipeCheck: Verifying Microarchitectural Enforcement of Consistency Models



- **Verify a given  $\mu$ arch** w.r.t. architectural spec.
- **Successes:** fast automated verification; bugs found



*next paper*

# Equalizer: Dynamically Tuning GPU Resources for Efficient Execution

**Ankit Sethia\***

**Scott Mahlke**

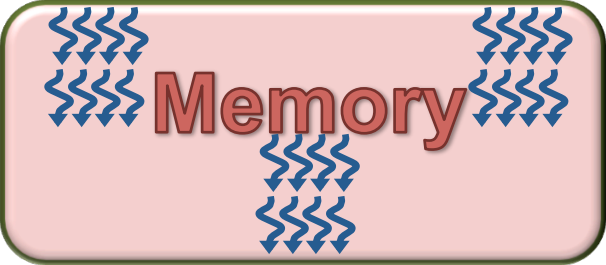
University of Michigan



# Imbalanced GPU resource utilization

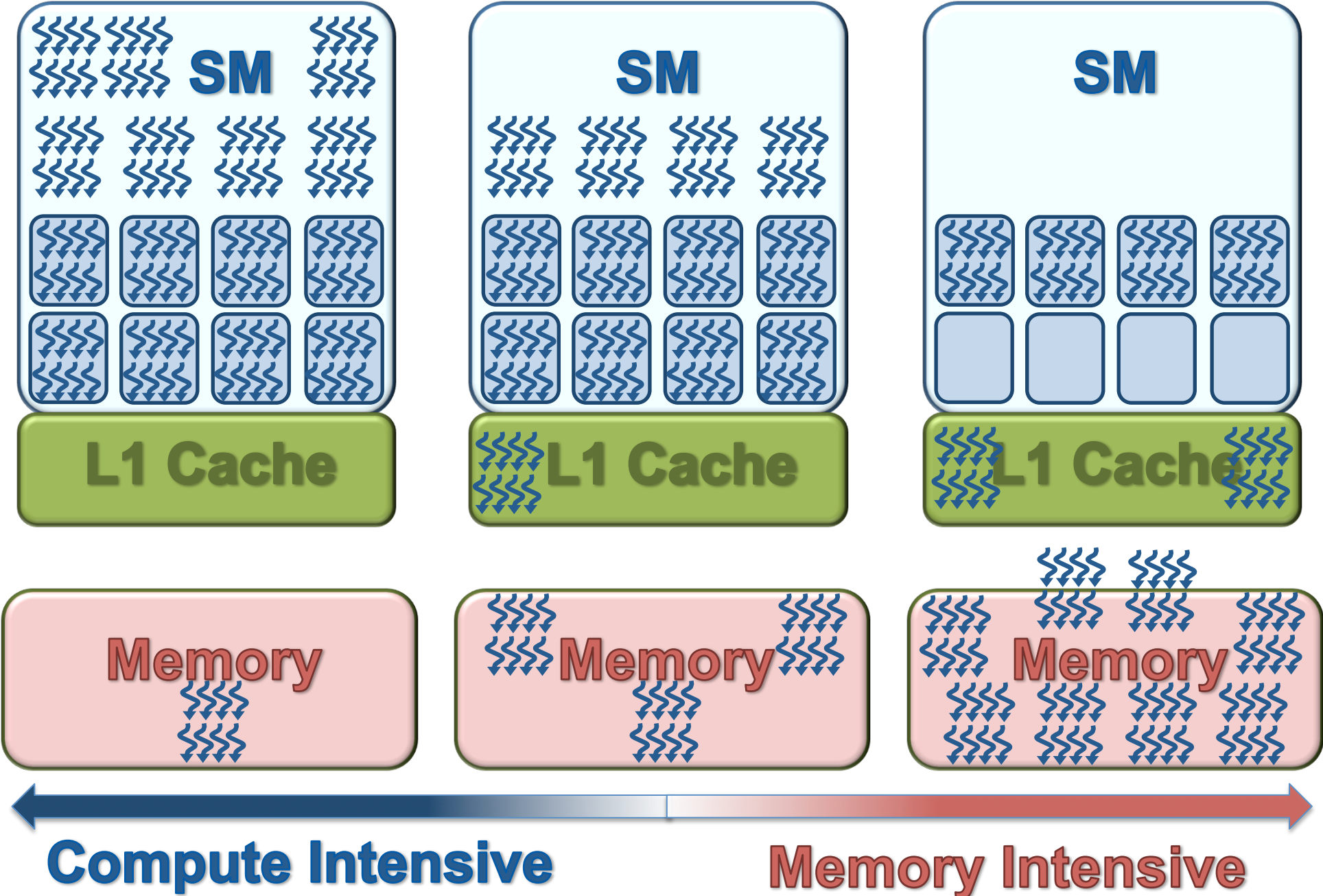


# Imbalanced GPU resource utilization



**Compute Intensive**

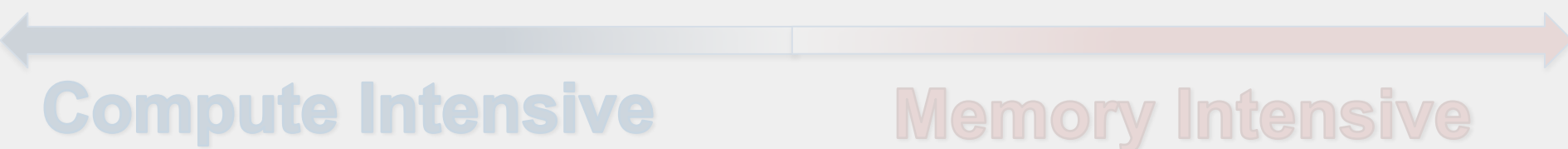
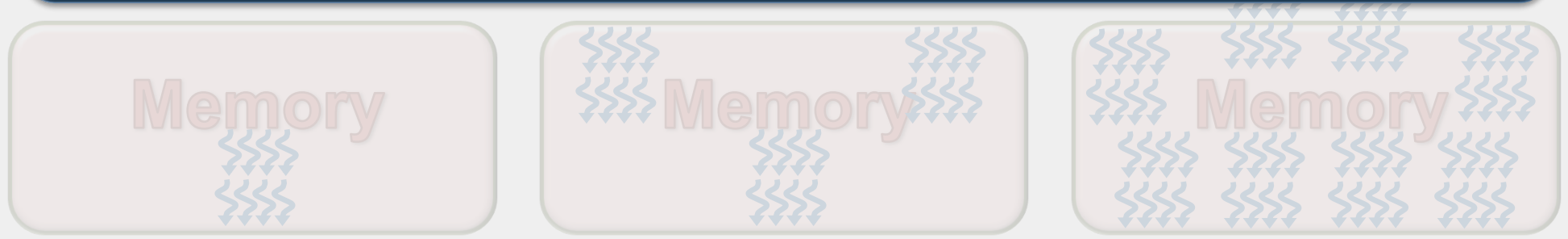
# Imbalanced GPU resource utilization



# Imbalanced GPU resource utilization



**Large number of threads cause early saturation of some resources and under-utilization of others**





# **Kernels saturate one resource much faster than others**

## *Opportunity 1:*

**Boost bottleneck resource for  
performance improvement**

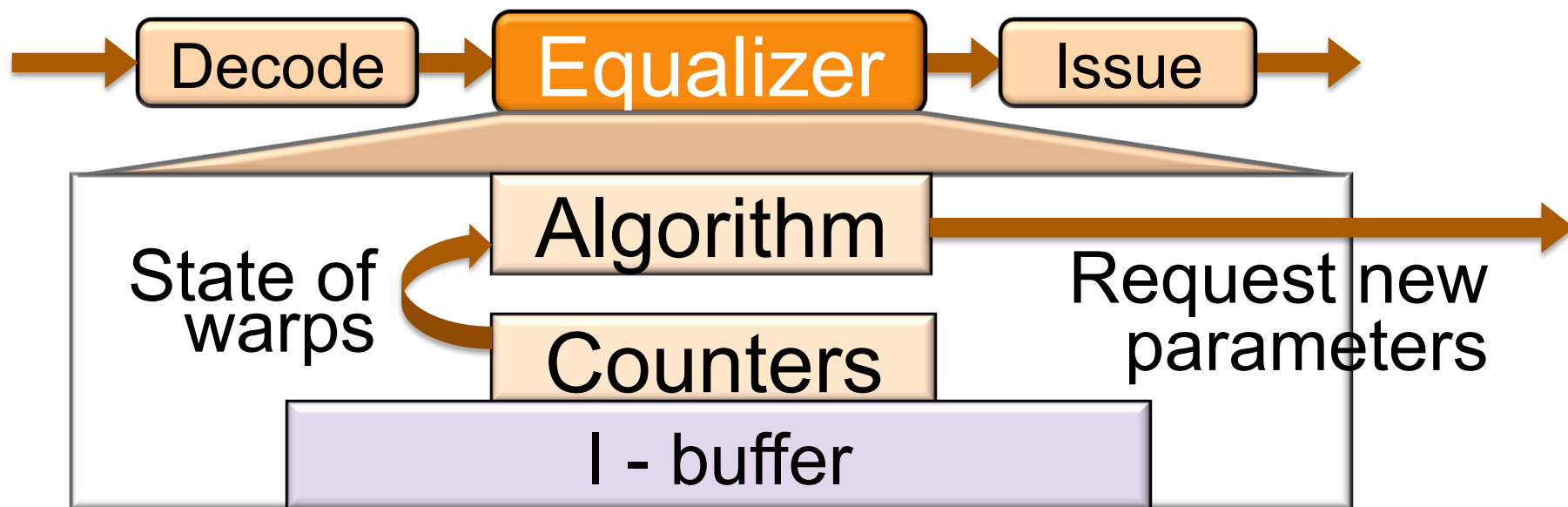
## *Opportunity 2:*

**Throttle under-utilized  
resources for energy savings**

# **Kernels saturate one resource much faster than others**

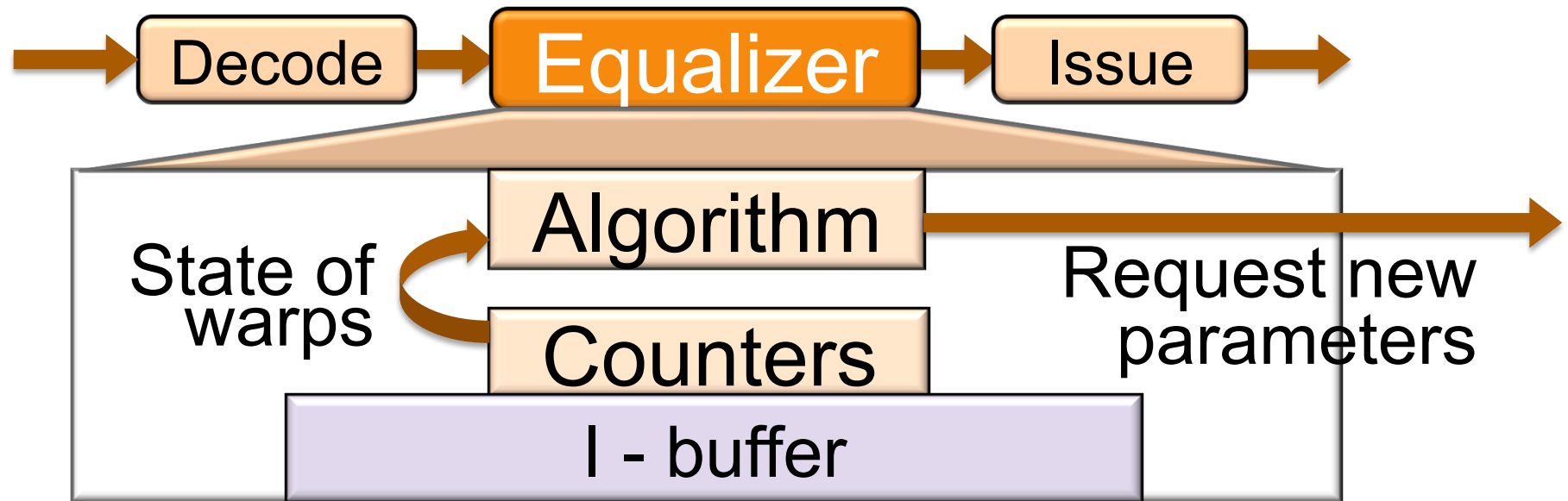
- **Observe kernel's hardware requirements**
- **Modulate hardware through:**
  - **Core frequency**
  - **Memory frequency**
  - **Number of threads**

# Equalizer



- Calculate state of warps over window of cycles
- Request new hardware parameters

# Equalizer



- Calculate state of warps over window of cycles
- Request new hardware parameters

**Boosting bottleneck resources:  
22% speedup, 6% energy overhead**

**Throttling under-utilized resources:  
15% energy savings, 5% speedup**

*next paper*

# COMP: Compiler Optimization for Manycore Processors

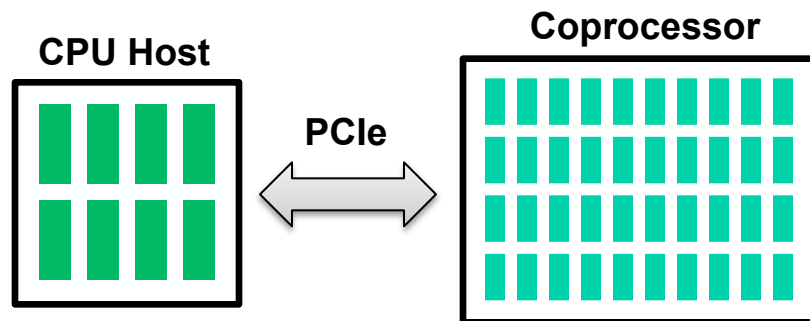
Linhai Song<sup>1</sup>, Min Feng<sup>2</sup>, Nishkam Ravi<sup>3</sup>, Yi Yang<sup>2</sup> and Srimat Chakradhar<sup>2</sup>

<sup>1</sup>University of Wisconsin-Madison

<sup>2</sup>NEC Laboratories America

<sup>3</sup>Cloudera Inc.

- We are entering manycore era
  - Intel Xeon Phi, Tiler processors, etc.
  - Used as **coprocessors**

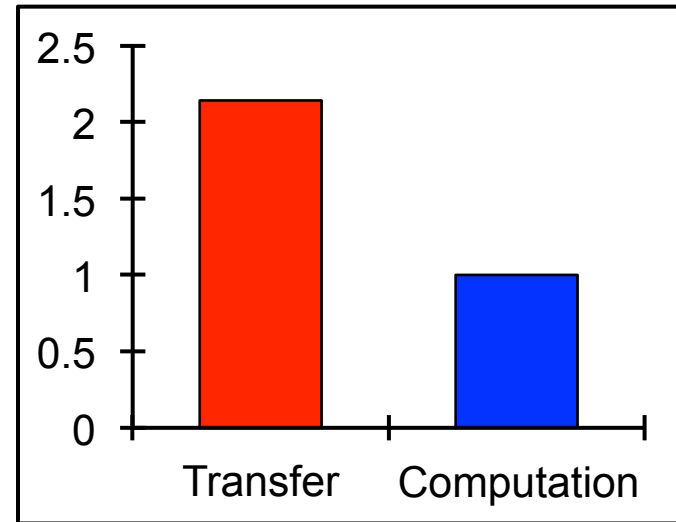


**Intel Xeon Phi**

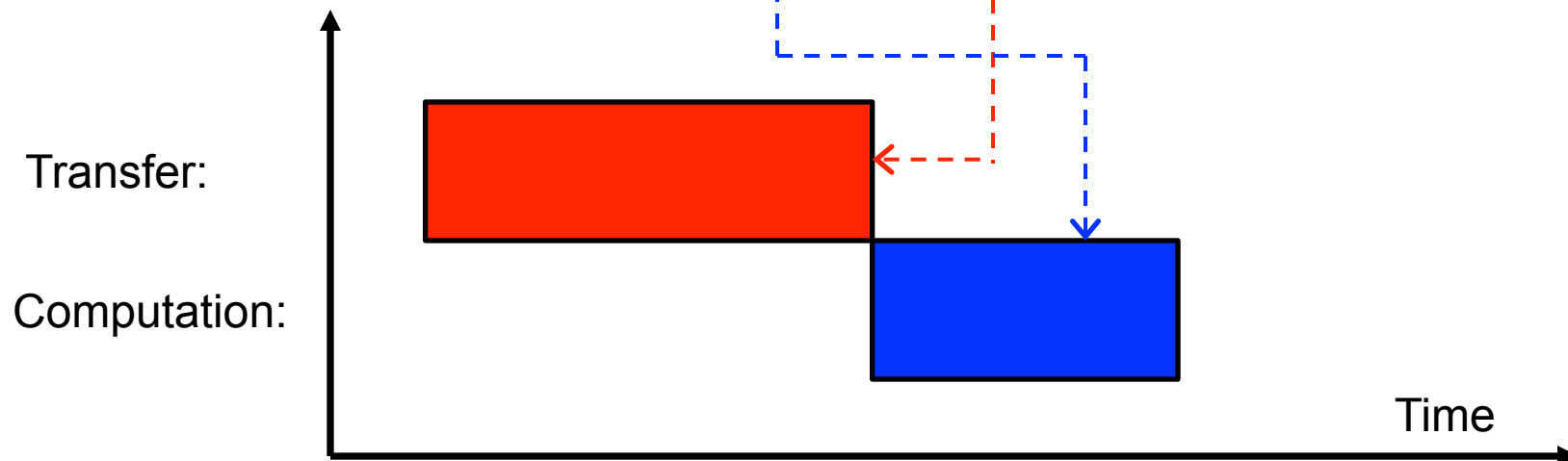
- 61 in-order cores
- 512-bit wide SIMD instructions and registers
- $4 \times 61 = 244$  hardware threads

# Performance Bottleneck: Data Transfer Overhead

```
#pragma offload target(mic:0)\  
in(sptprice, ...:length(numOptions))\  
out(prices:length(numOptions))  
#pragma omp parallel for private(i, price)  
for (i=0; i<numOptions; i++) {  
    price = BlkSchlsEqEuroNoDiv(  
        sptprice[i], ..., 0);  
    prices[i] = price;  
}
```



Time breakdown for Blacksholes

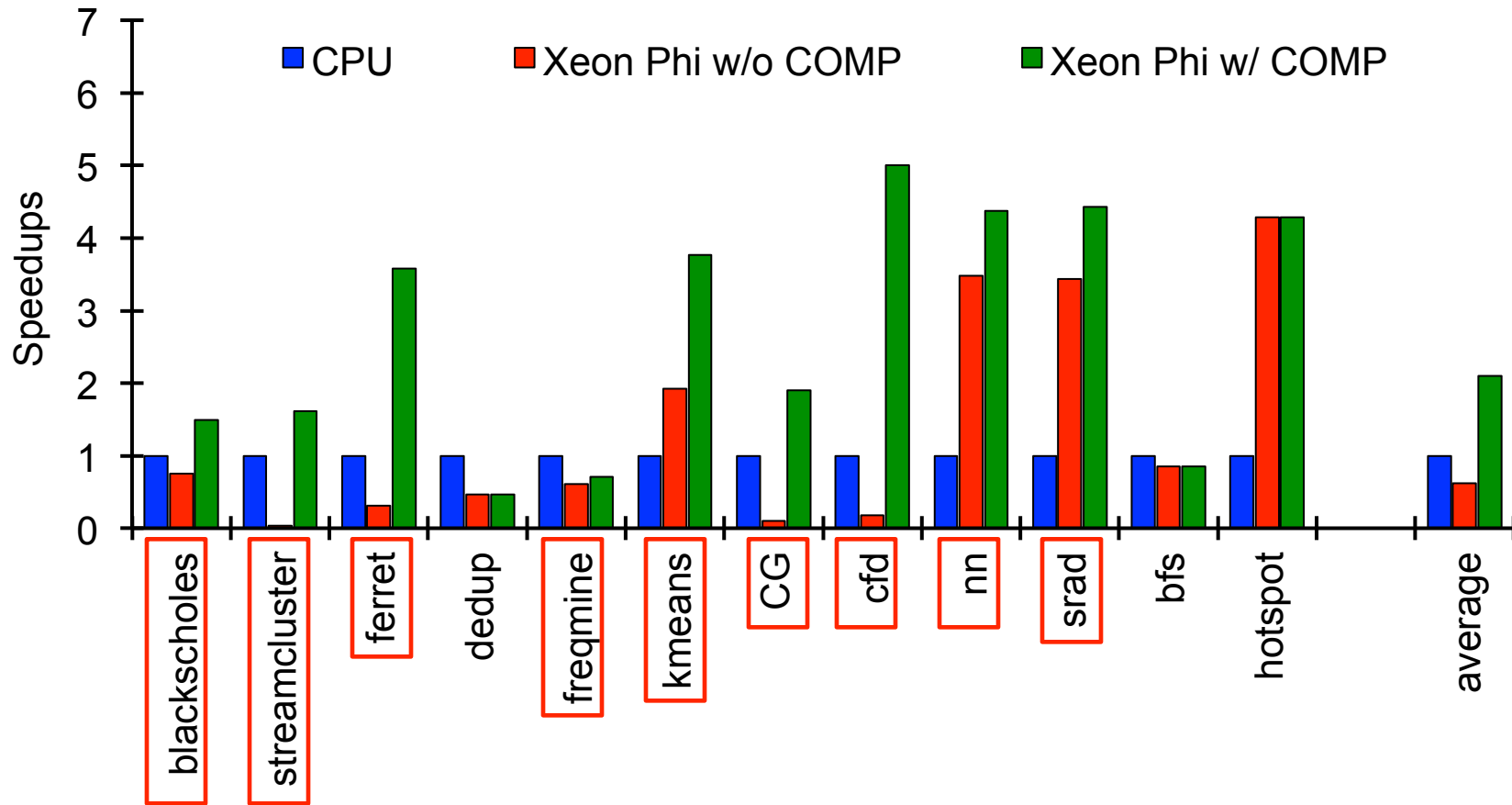


# Our Compiler Optimizations

- **Data streaming**
  - Automatically overlap data transfers and computations to reduce data transfer overhead
  - Designed to minimize the device memory usage while maximizing the performance
  - Avoid the overhead of launching the same kernel for multiple times
- **Regularization**
  - Enable data streaming in the presence of irregular accesses
  - Eliminate unnecessary data transfer
  - Improve vectorization and locality
- **New shared memory mechanism**
  - Designed to quickly transfer pointer-based data structures between host and device



# Evaluation



Our optimizations benefit 9 out of 12 benchmarks.  
(1.16x ~ 52.21x speedups)

*end*

*end*