

# QUALITY PROGRAMMABLE VECTOR PROCESSORS FOR APPROXIMATE COMPUTING

**Swagath Vekataramani**<sup>1</sup>, Vinay Chippa<sup>1</sup>, Srimat  
Chakradhar<sup>2</sup>, Kaushik Roy<sup>1</sup>, Anand Raghunathan<sup>1</sup>

**<sup>1</sup>Integrated Systems Laboratory  
School of ECE, Purdue University**

**<sup>2</sup>NEC Laboratories America**



**INTEGRATED SYSTEMS  
LABORATORY**

Computers viewed as *precise calculators*



Computers viewed as *precise calculators*

- ▶ Leads to inefficiency



But, I worked  
*harder* than  
needed

# Computers viewed as *precise calculators*

- ▶ Leads to inefficiency



But, I worked **harder** than needed



- ▶ Relaxed notion of correctness

# Computers viewed as *precise calculators*

- ▶ Leads to inefficiency



But, I worked **harder** than needed

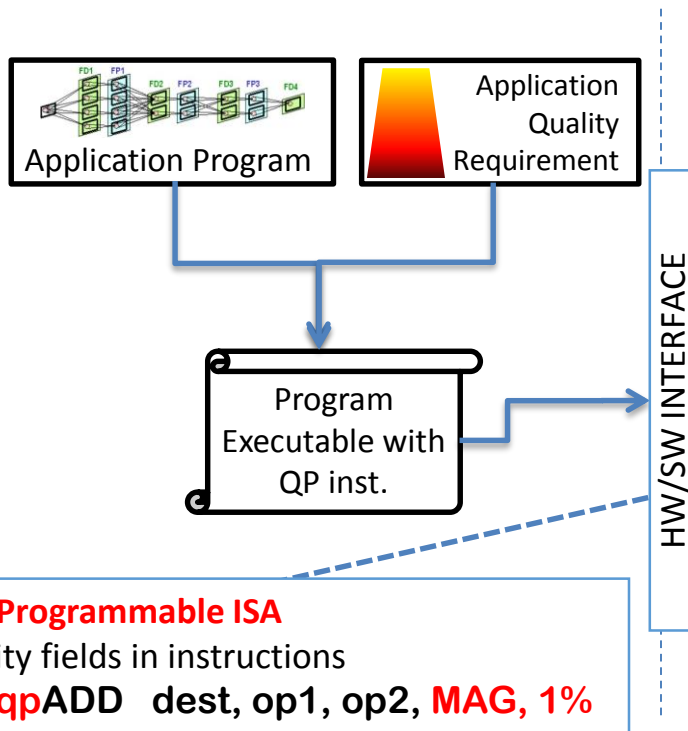


- ▶ Relaxed notion of correctness
  - ▶ Results cannot be arbitrary either

*Good enough* answers !!!

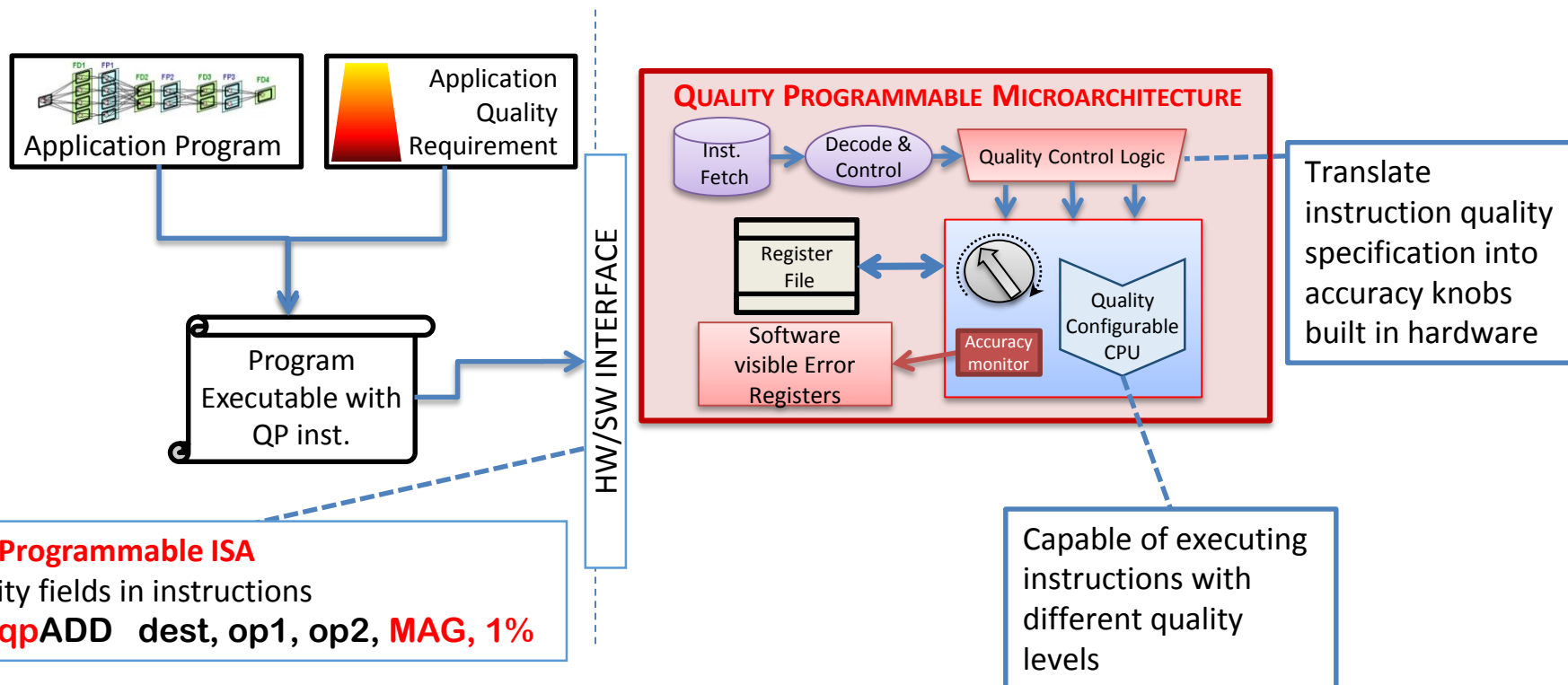
# QUALITY PROGRAMMABLE PROCESSORS

- ▶ Notion of *quality* built into the instruction set



# QUALITY PROGRAMMABLE PROCESSORS

- ▶ Notion of *quality* built into the instruction set



# QUALITY PROGRAMMABLE PROCESSORS

- ▶ Notion of *quality* built into the instruction set

## QUORA

Quality programmable 1D/2D vector processor

**1.7X** savings for **NO** loss in output quality

**>2X** savings for **modest (<2.5%)** quality loss

ality  
nto  
s  
are

### Quality Programmable ISA

Quality fields in instructions

e.g. **qpADD** dest, op1, op2, **MAG, 1%**

HV

Capable of executing instructions with different quality levels



# QUALITY PROGRAMMABLE PROCESSORS

- ▶ Notion of *quality* built into the instruction set

Hmm.. Does this really work?

*Lots* of design considerations!

*Session 1A: Approximate Computing*  
*MONDAY @ 1:30 PM*

ality  
nto  
s  
are

## Quality Programmable ISA

Quality fields in instructions

e.g. **qpADD** dest, op1, op2, **MAG**, 1%

HV

Capable of executing instructions with different quality levels

---

# SAGE: Self-Tuning Approximation for Graphics Engines

**Mehrzaad Samadi<sup>1</sup>, Janghaeng Lee<sup>1</sup>, D. Anoushe  
Jamshidi<sup>1</sup>, Amir Hormati<sup>2</sup>, and Scott Mahlke<sup>1</sup>**

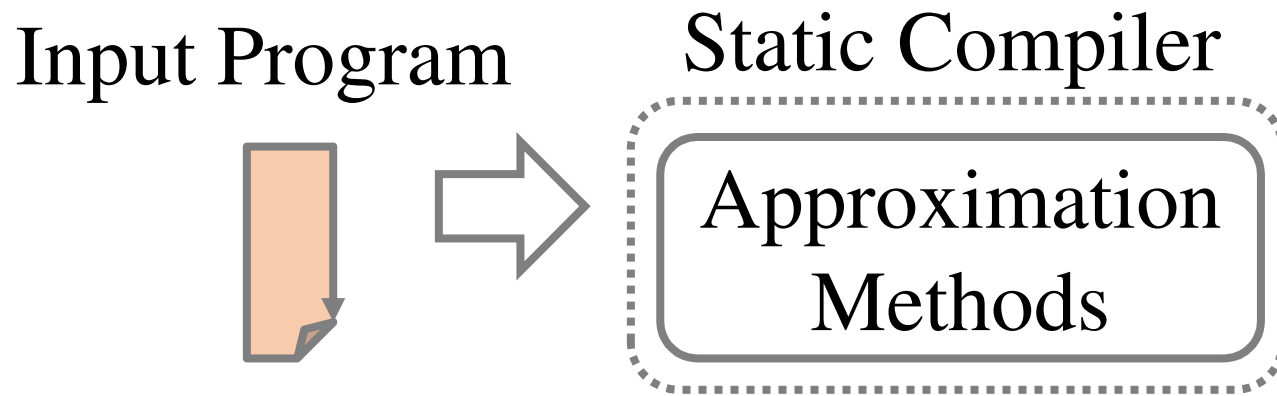
University of Michigan<sup>1</sup>, Google Inc.<sup>2</sup>



# GPU Specific Approximation

---

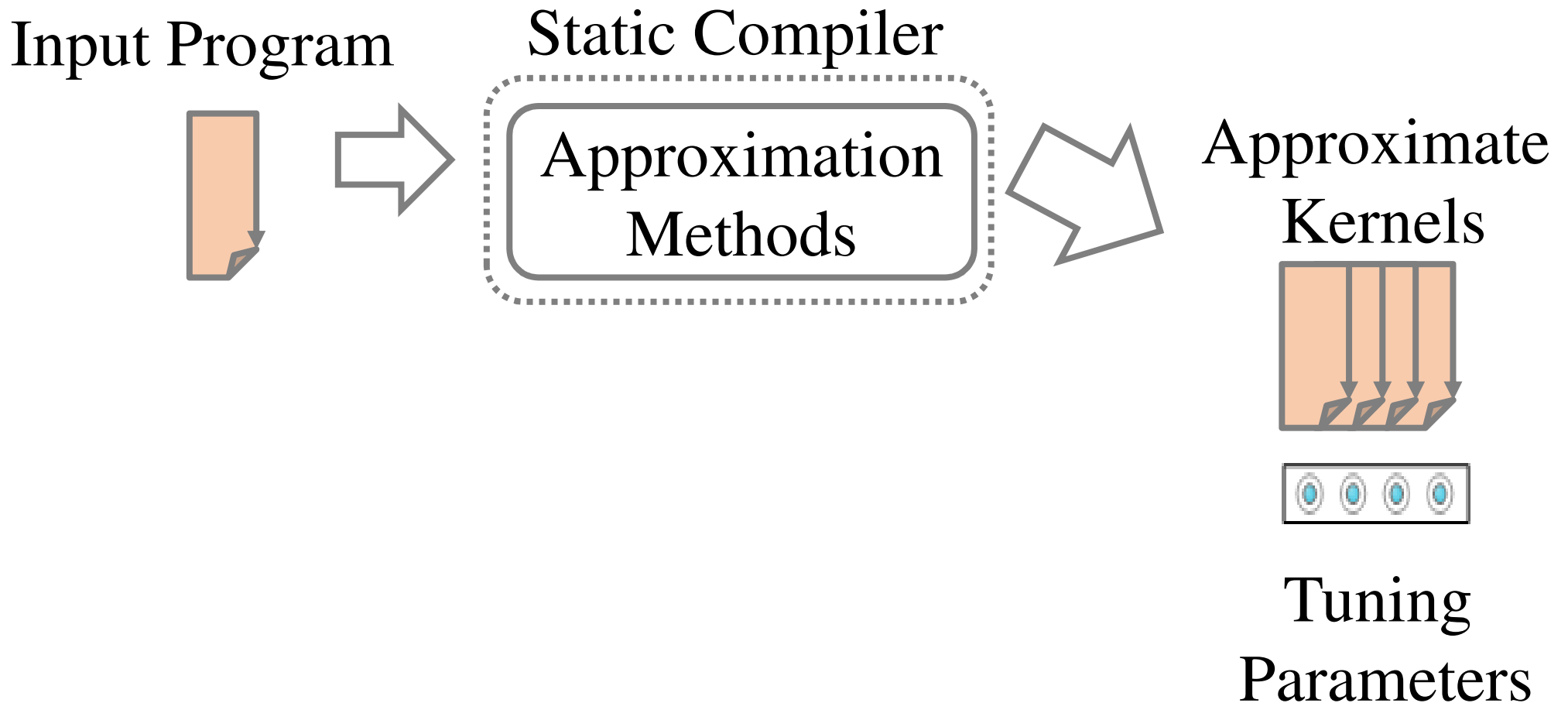
**Goal:** Hardware-aware approximation



# GPU Specific Approximation

---

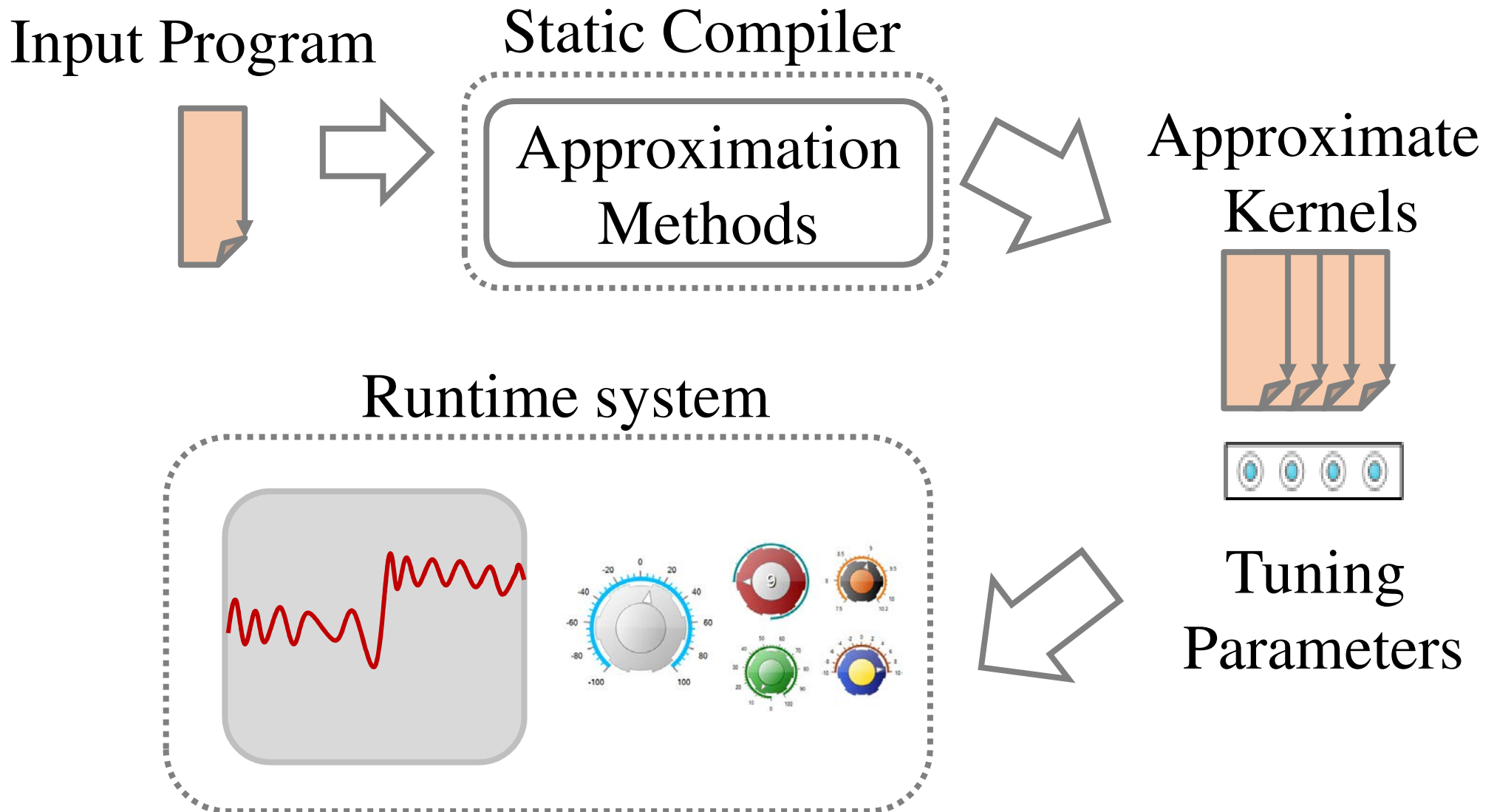
**Goal:** Hardware-aware approximation



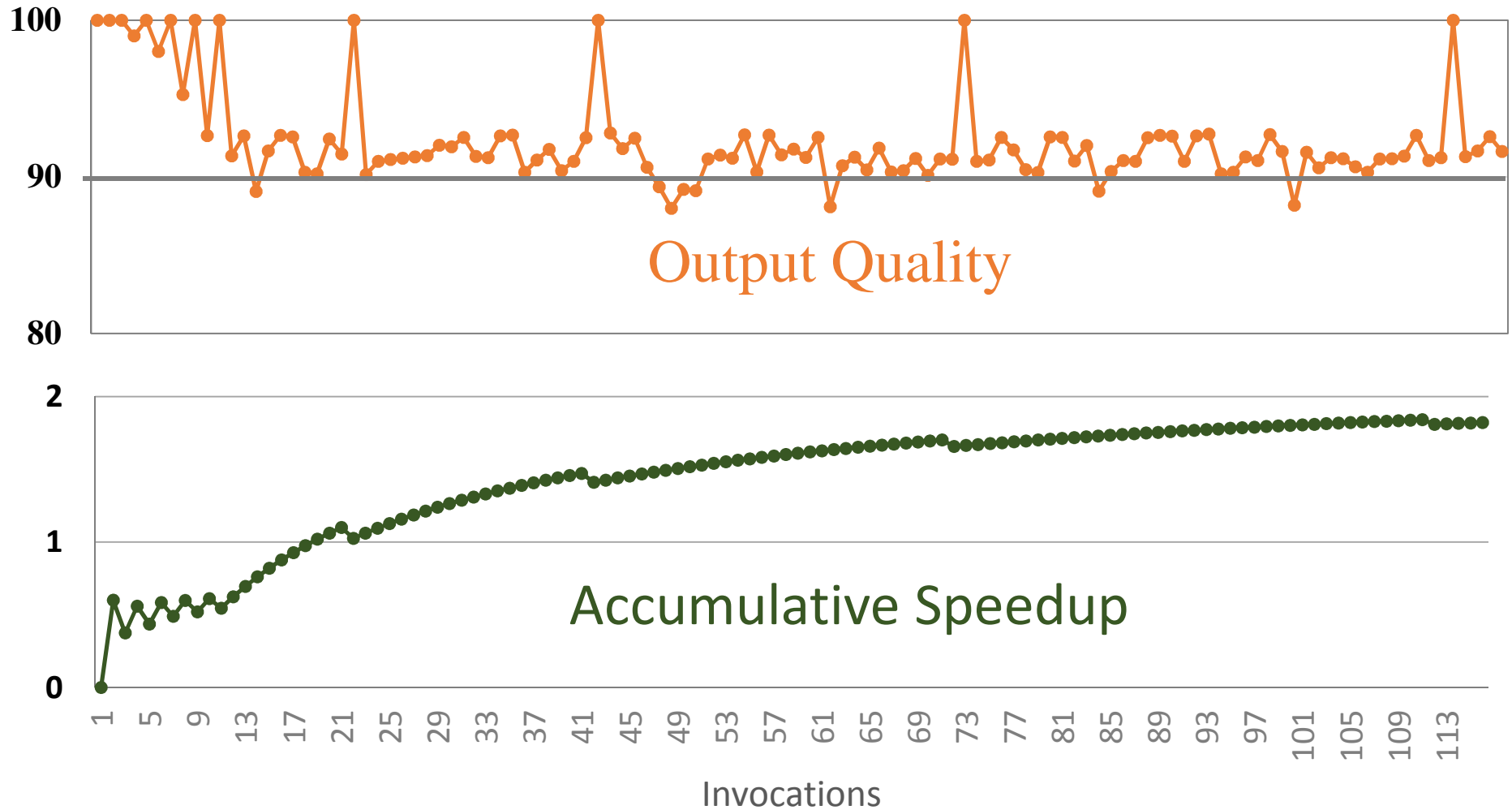
# GPU Specific Approximation

---

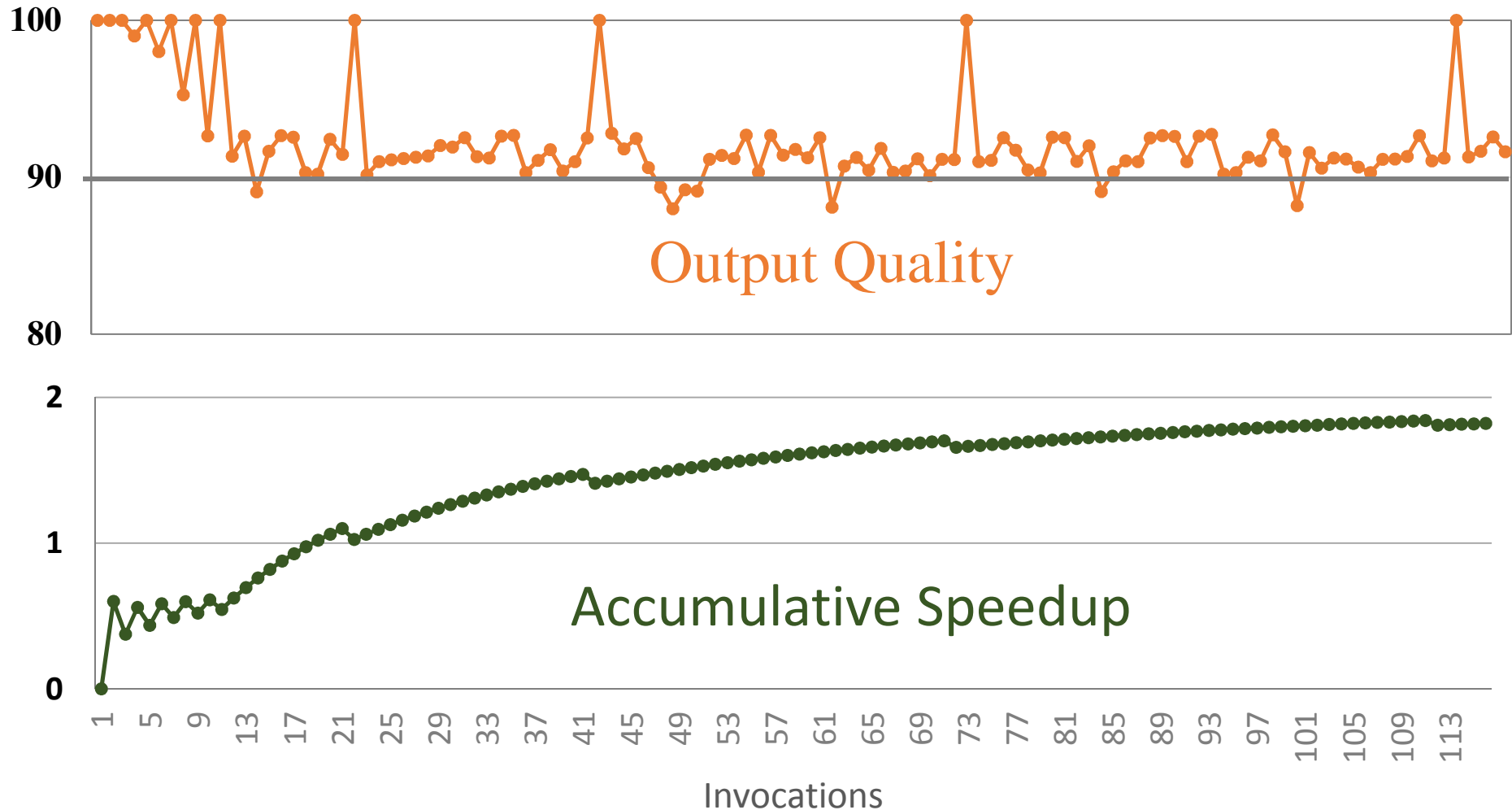
**Goal:** Hardware-aware approximation



# We Can Control Output Quality.



# We Can Control Output Quality.



Across 10 applications:

2.5x speedup with 90% output quality

2.0x speedup with 95% output quality

# Approximate Storage in Solid-State Memories

Adrian Sampson

Jacob Nelson

Karin Strauss

Luis Ceze

University of Washington &

Microsoft Research





# Approximate Storage in Solid-State Memories

Adrian Sampson  
Jacob Nelson  
Karin Strauss  
Luis Ceze

University of Washington &  
Microsoft Research



# Approximate Storage in Solid-State Memories

Adrian Sampson  
Jacob Nelson  
Karin Strauss  
Luis Ceze

University of Washington &  
Microsoft Research



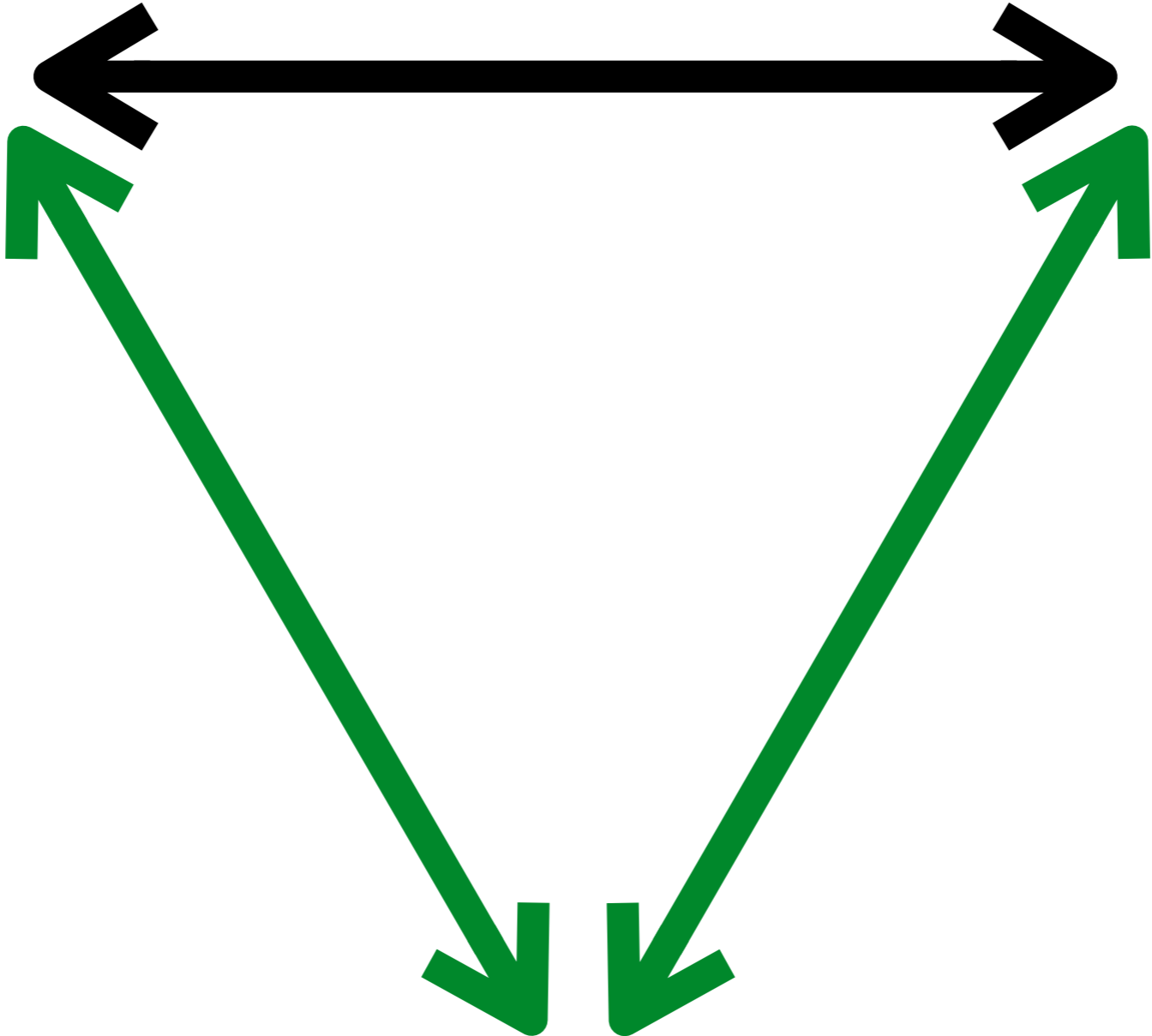
**Fast**



**Dense**

**Fast**

**Dense**



**Accurate**

**70%**

faster writes

**23%**

lifetime extension

**70%**

faster writes

**23%**

lifetime extension

Approximate Storage in Solid-State Memories

**right here, after lunch**

# MLP-Aware Dynamic Instruction Window Resizing for Adaptively Exploiting Both ILP and MLP

Yuya Kora

Kyohei Yamaguchi

Hideki Ando

*Nagoya University*

# Problem to Solve

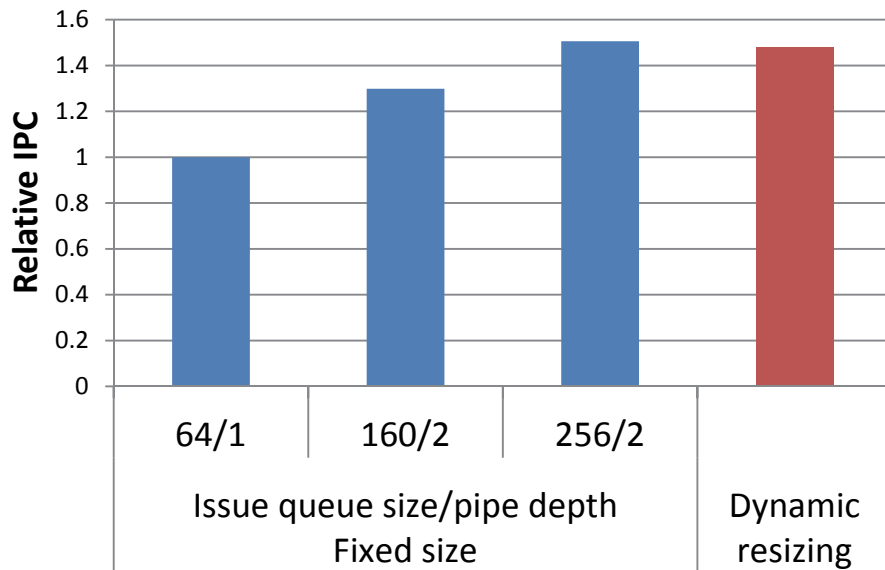
- Difficult to improve single-thread performance in memory-intensive programs
  - Memory wall
- Very large instruction window can overcome this problem by exploiting MLP
  - This degrades the clock cycle time
  - While pipelining can solve this, it instead prevents ILP exploitation, degrading IPC in compute-intensive programs



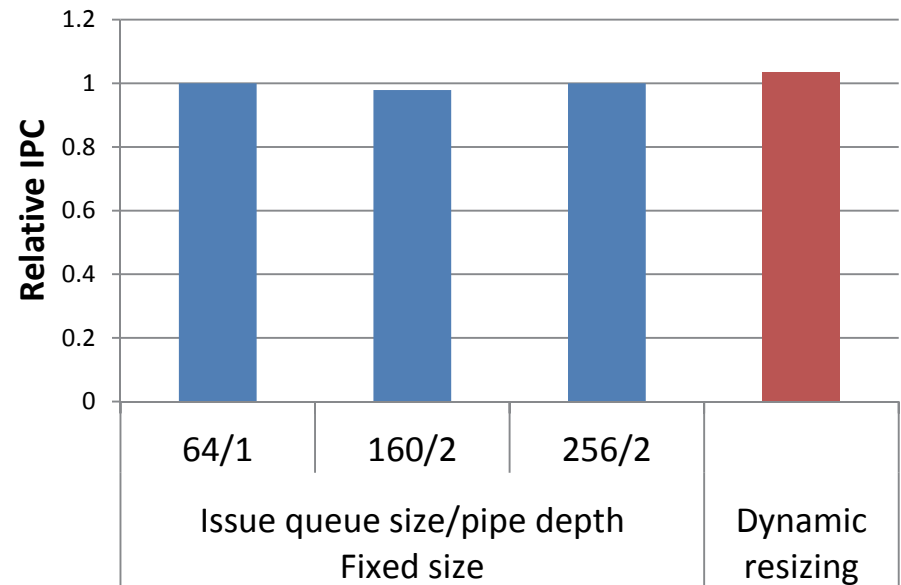
# Dynamic Instruction Window Resizing

- Adapt window size to available parallelism
  - ILP or MLP
  - Based on prediction

GM memory-intensive

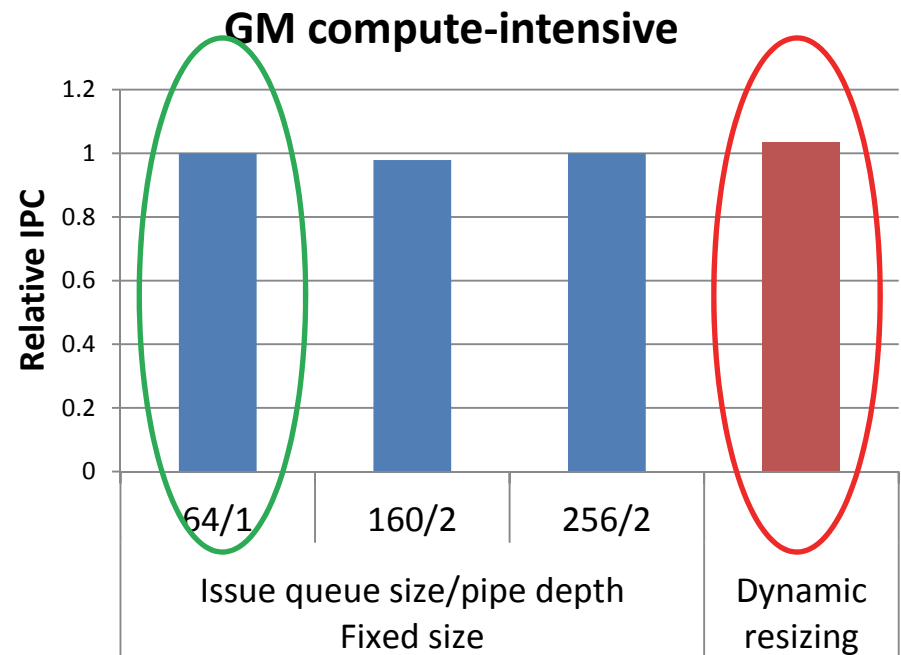
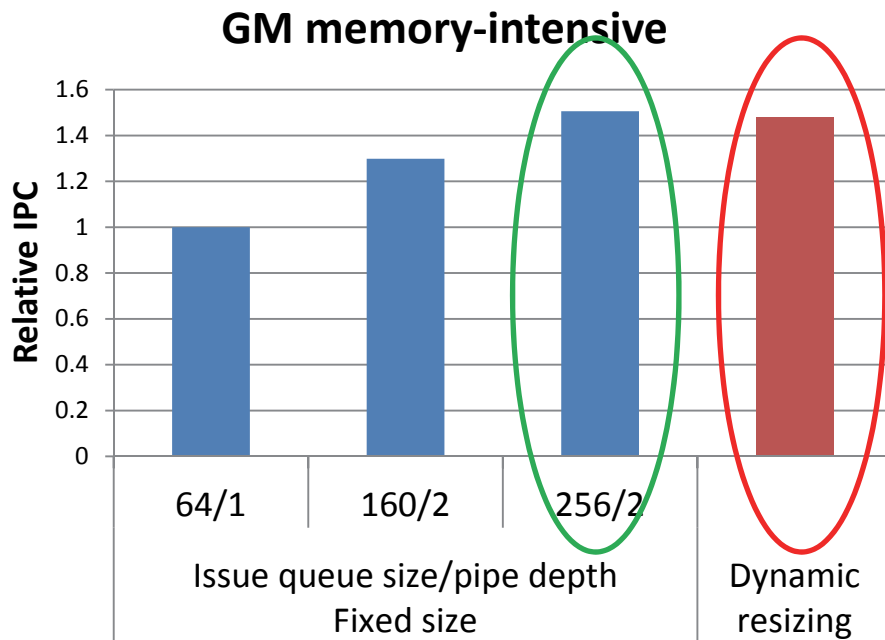


GM compute-intensive



# Dynamic Instruction Window Resizing

- Adapt window size to available parallelism
  - ILP or MLP
  - Based on prediction



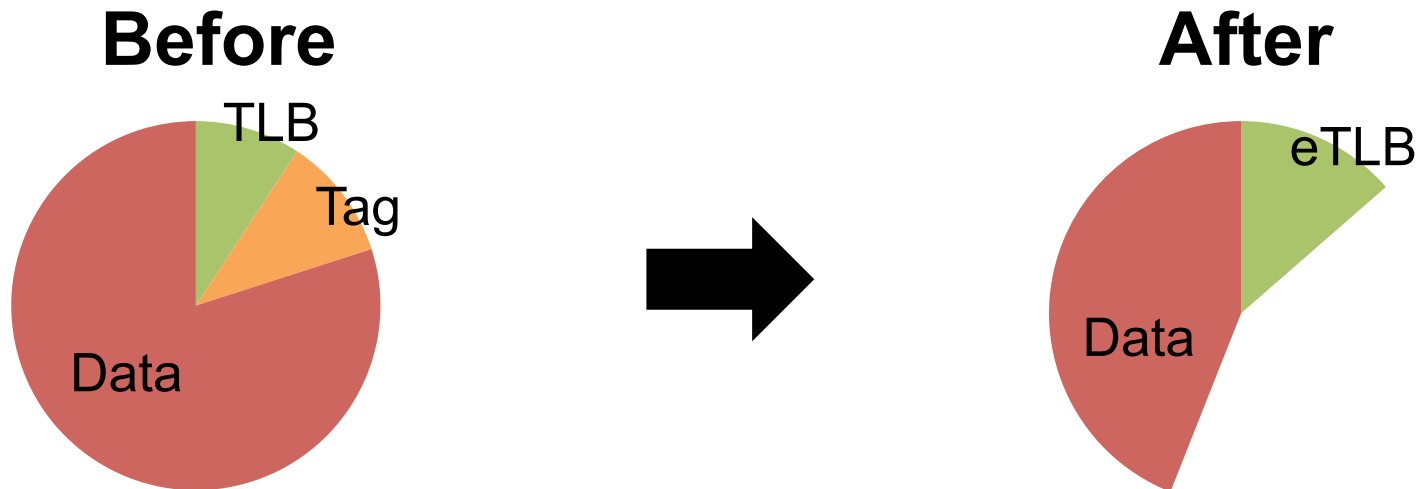
**21% speedup on average**

# TLC: A Tag-Less Cache for Reducing Dynamic First Level Cache Energy

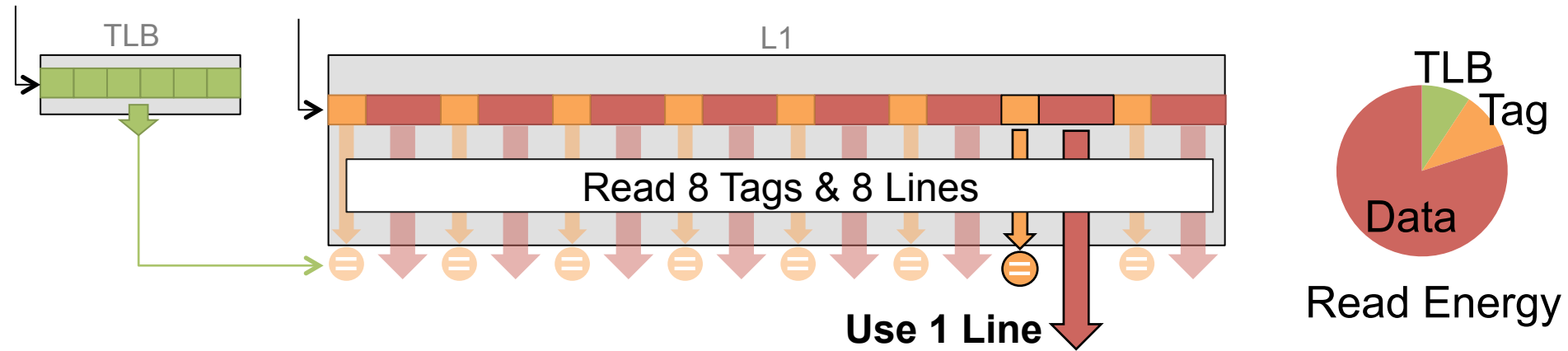
Andreas Sembrant, Erik Hagersten, David Black-Schaffer  
Uppsala University, Sweden

14:00 Session 1B - Energy Optimizations [Alpha Gamm Rho Room]

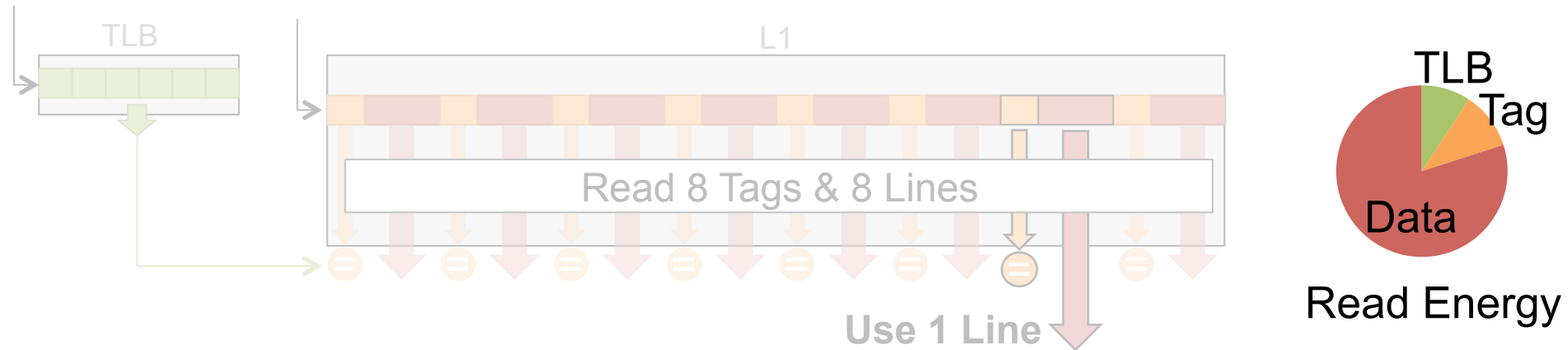
## L1D Dynamic Read Energy



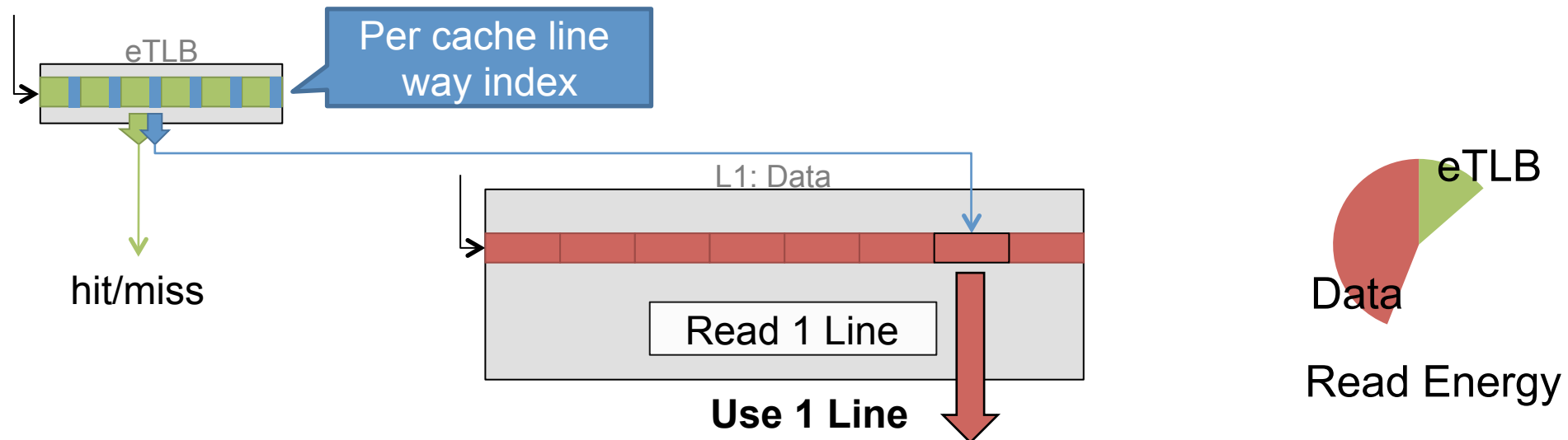
# Problem: L1D consumes energy due to tags and ways



## Problem: L1D consumes energy due to tags and ways



## Solution: extend the TLB to *eliminate tags* and *find the way*



# Results

## Reduce total L1D dynamic energy by 78%

### 1. Eliminate extra data-array reads

- *by determining the correct correct way from the TLB*

### 2. Eliminate the tag-array

- *by avoiding tag comparisons*

### 3. Filter out cache misses

- *by checking in the eTLB*

### 4. Amortize the TLB lookup energy

- *by integrating it with way information*

# Results

## Reduce total L1D dynamic energy by 78%

### 1. Eliminate extra data-array reads

- *by determining the correct correct way from the TLB*

### 2. Eliminate the tag-array

- *by avoiding tag comparisions*

### 3. Filter out cache misses

- *by checking in the eTLB*

### 4. Amortize the TLB lookup energy

- *by integrating it with way information*

### ▪ More cool stuff in the presentation:

- *$\mu$ Pages, synonyms, coherency, replacements, ...*

14:00 Session 1B - Energy Optimizations [Alpha Gamm Rho Room]

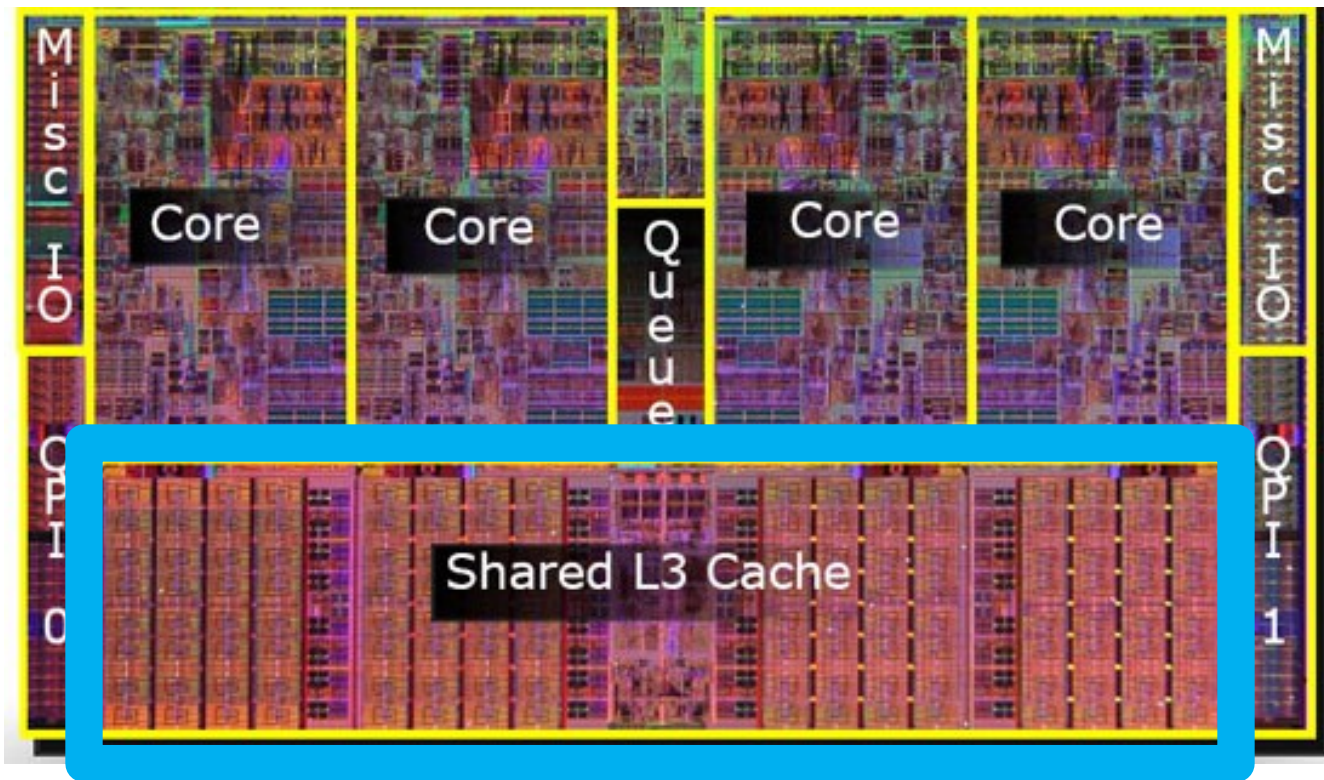


# Decoupled Compressed Cache:

Exploiting Spatial Locality for Energy-Optimized Compressed Caching

Somayeh Sardashti and David A. Wood

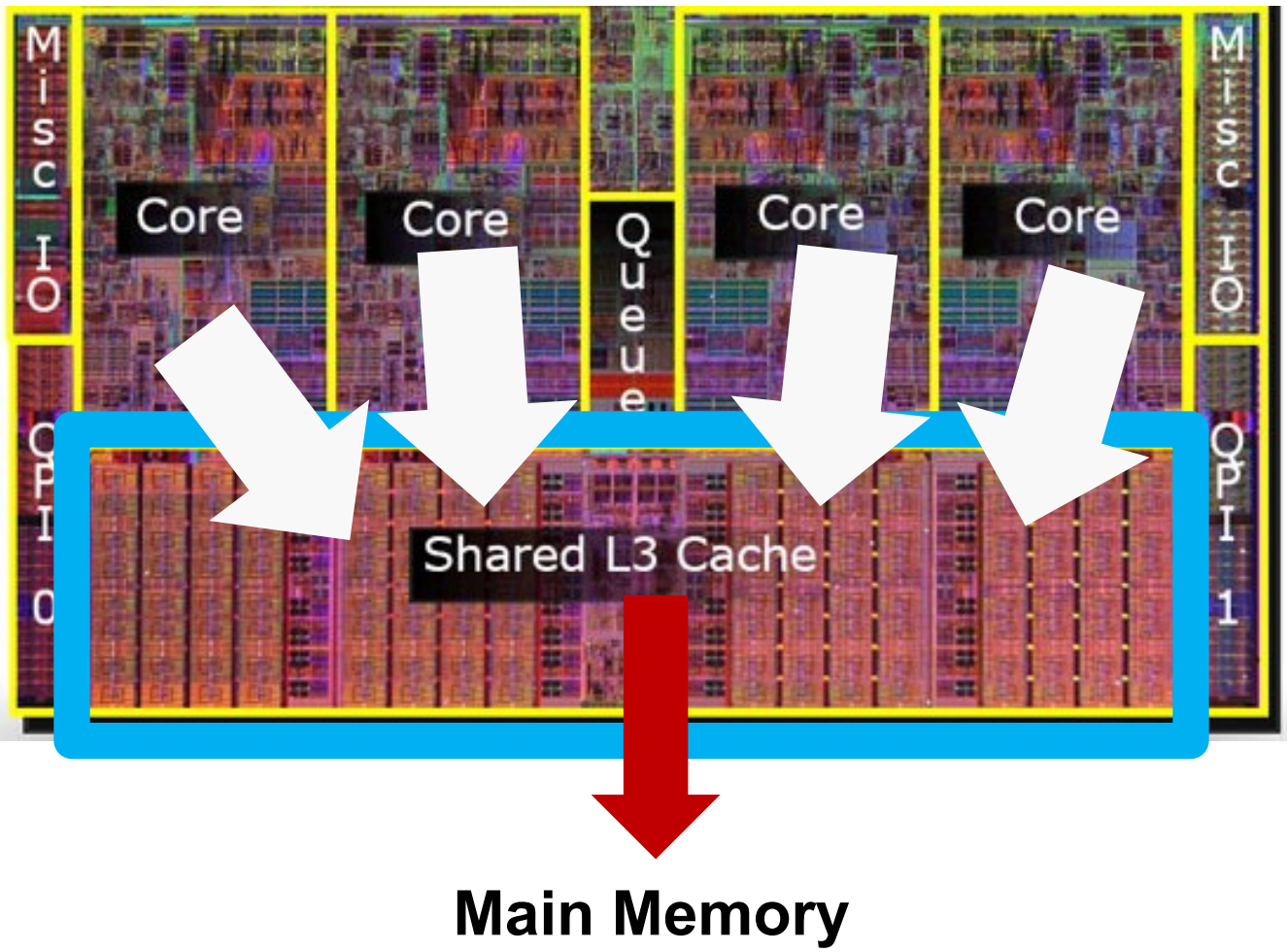
University of Wisconsin-Madison





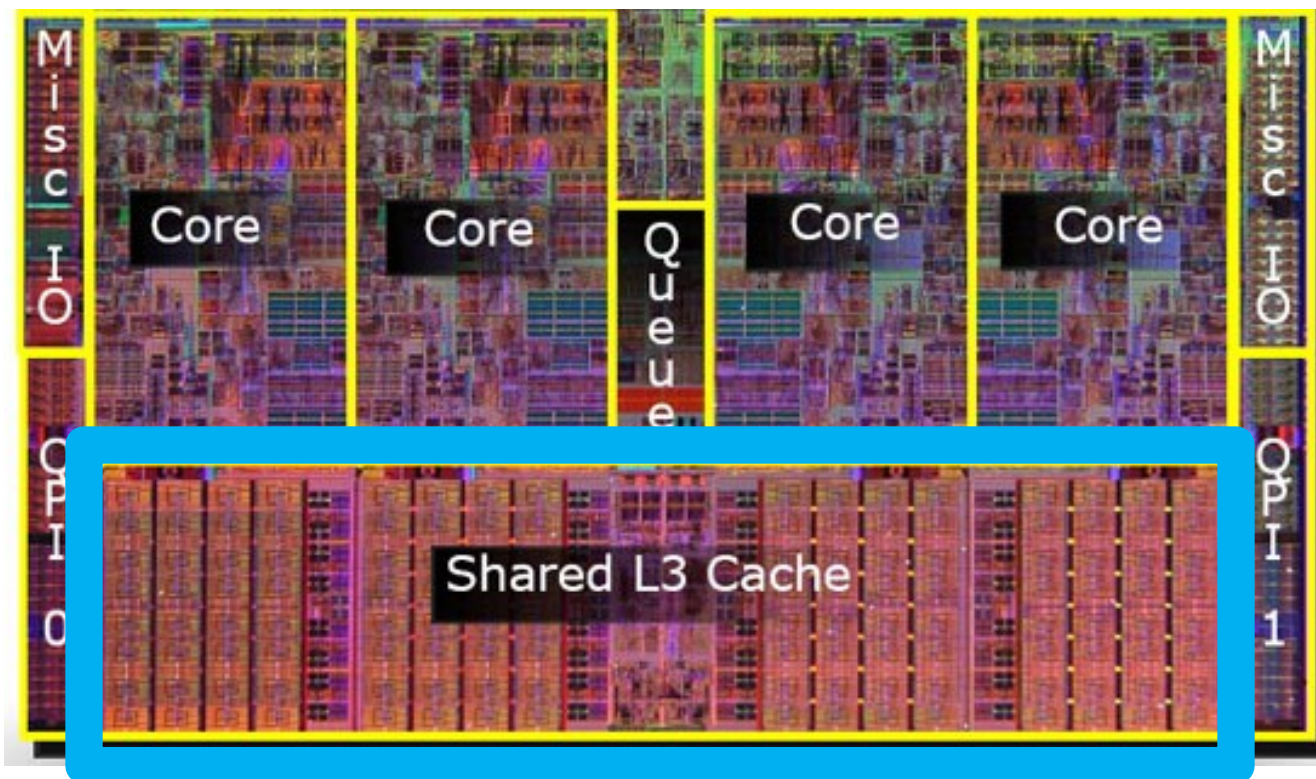


# Cache as Energy Filters





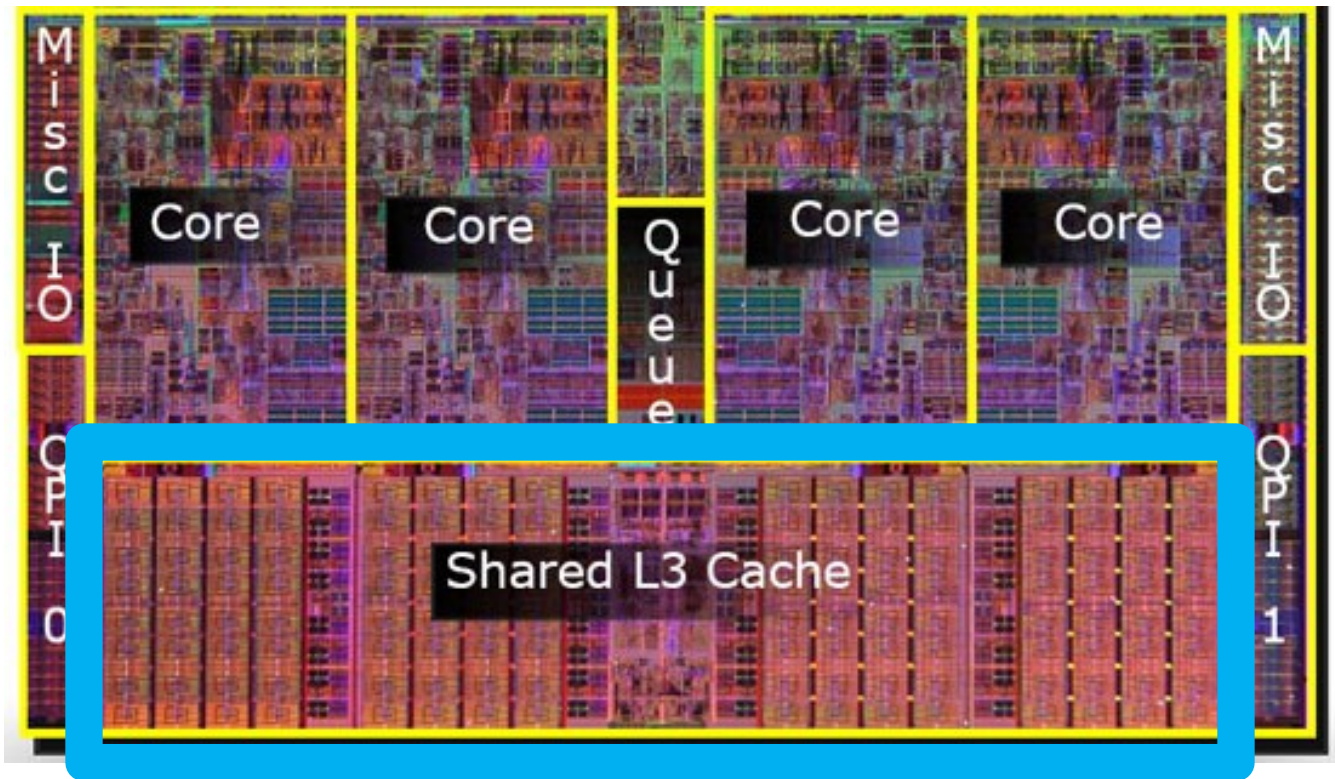
# Why not double the LLC?





# Why not double the LLC?

## 2X LLC Area!





# State of the Art: Compressed Cache

**Compacting compressed blocks in the same data space**

- ✓ **High Effective Cache Capacity**
- ✓ **Small Area Overhead**

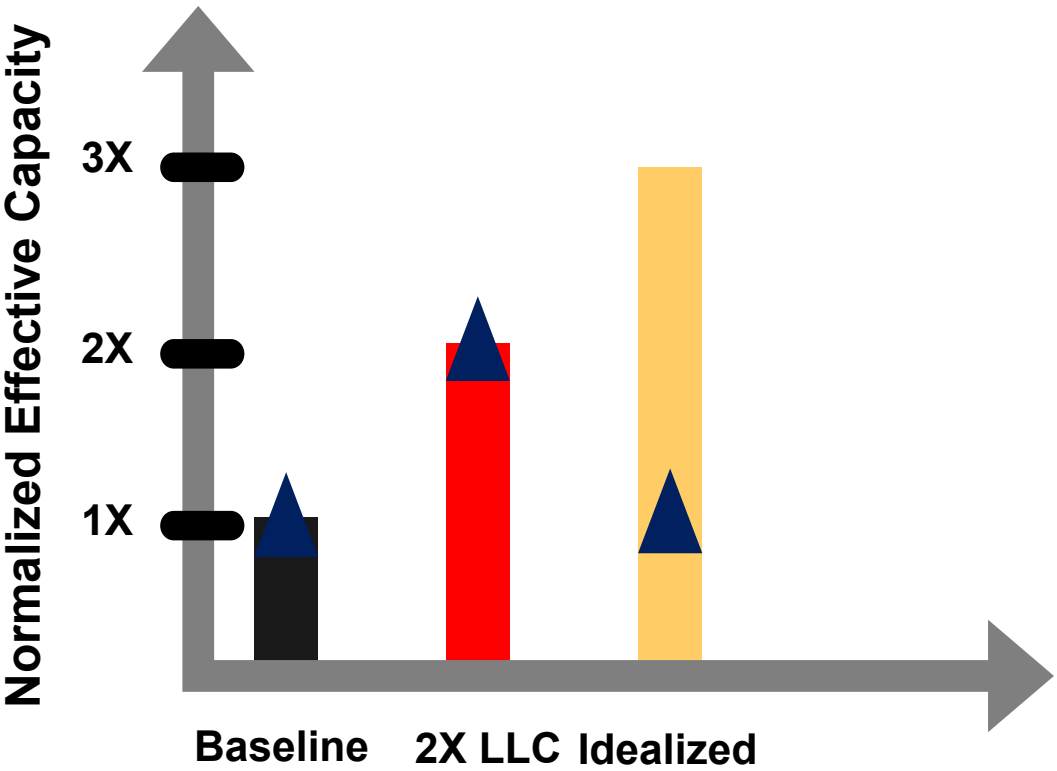




# State of the Art: Compressed Cache

Compacting compressed blocks in the same data space

- ✓ High Effective Cache Capacity
- ✓ Small Area Overhead

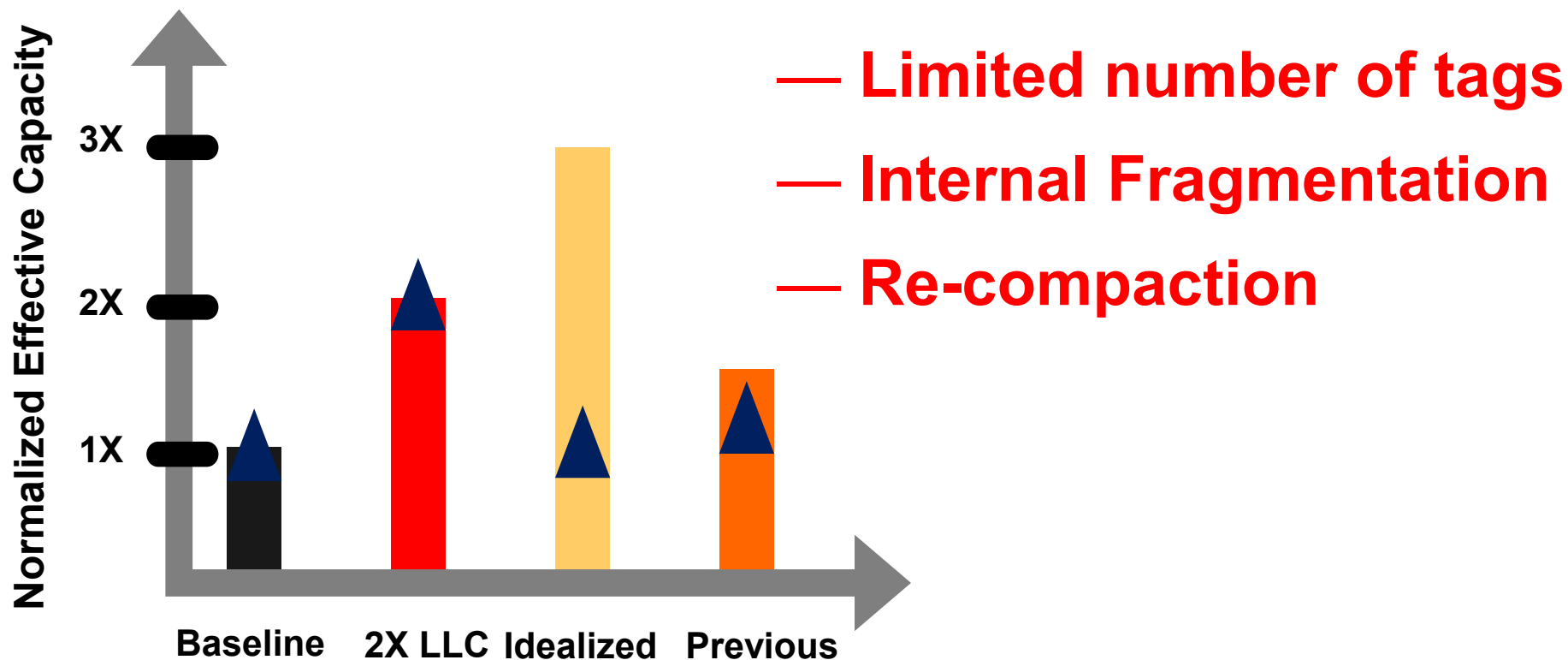




# State of the Art: Compressed Cache

Compacting compressed blocks in the same data space

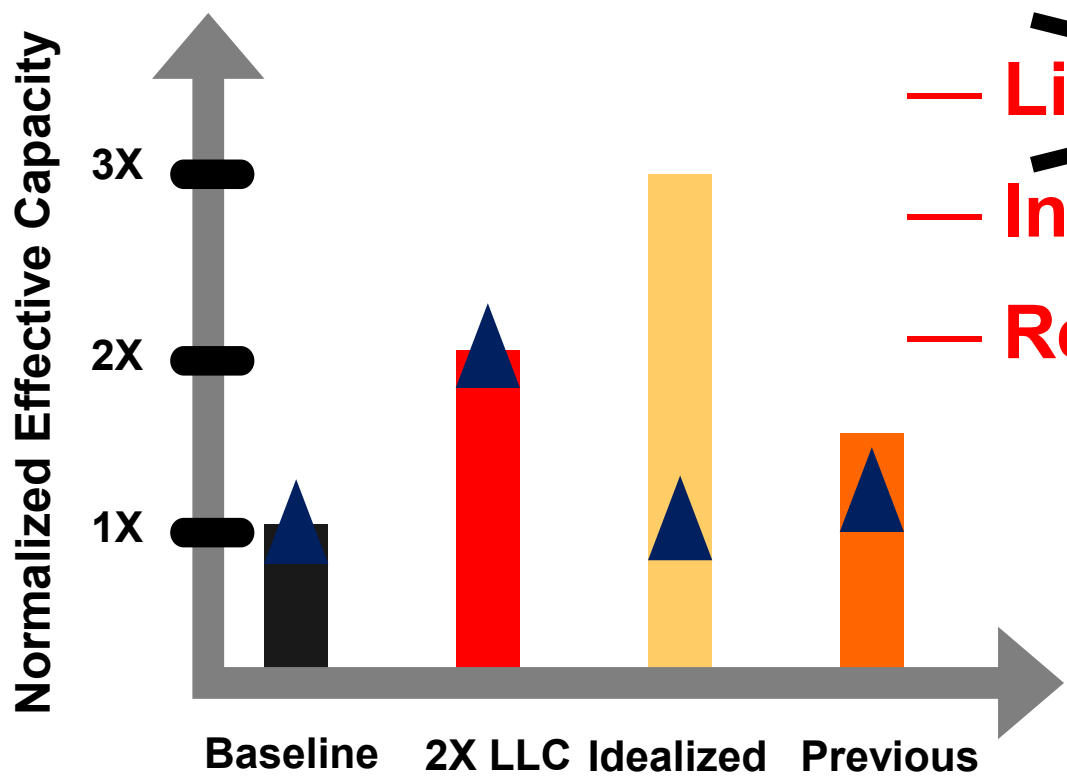
- ✓ High Effective Cache Capacity
- ✓ Small Area Overhead





# Our Proposal: Decoupled Compressed Cache (DCC)

## Decoupled Super-Blocks

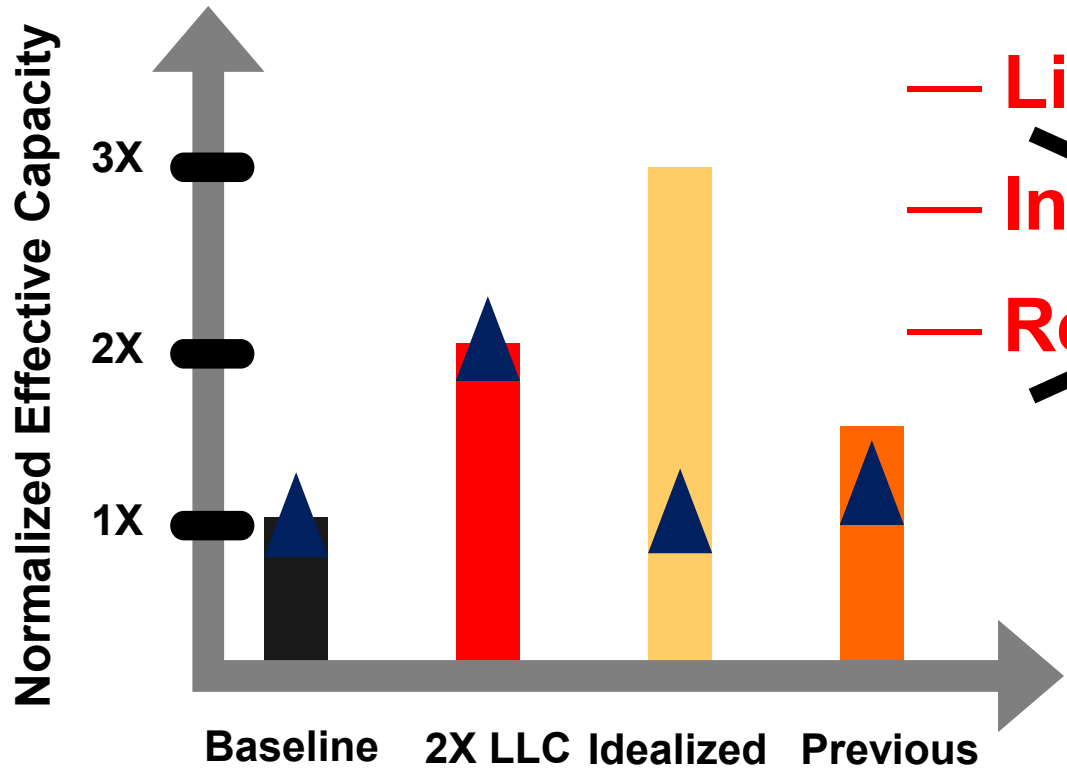


- ~~Limited number of tags~~
- ~~Internal Fragmentation~~
- ~~Re-compaction~~



# Our Proposal: Decoupled Compressed Cache (DCC)

## Non-Contiguous Sub-Blocks



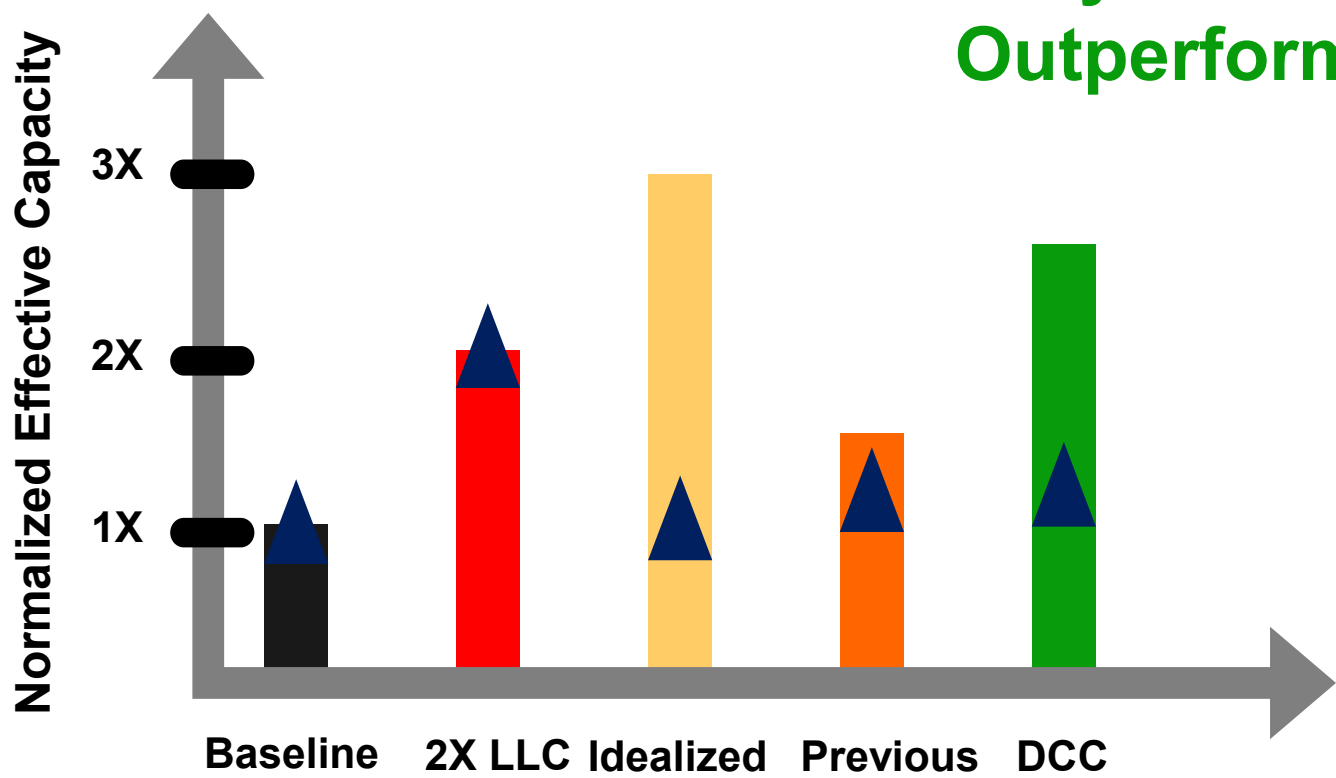
- Limited number of tags
- Internal Fragmentation
- Re-compaction





# Our Proposal: Decoupled Compressed Cache (DCC)

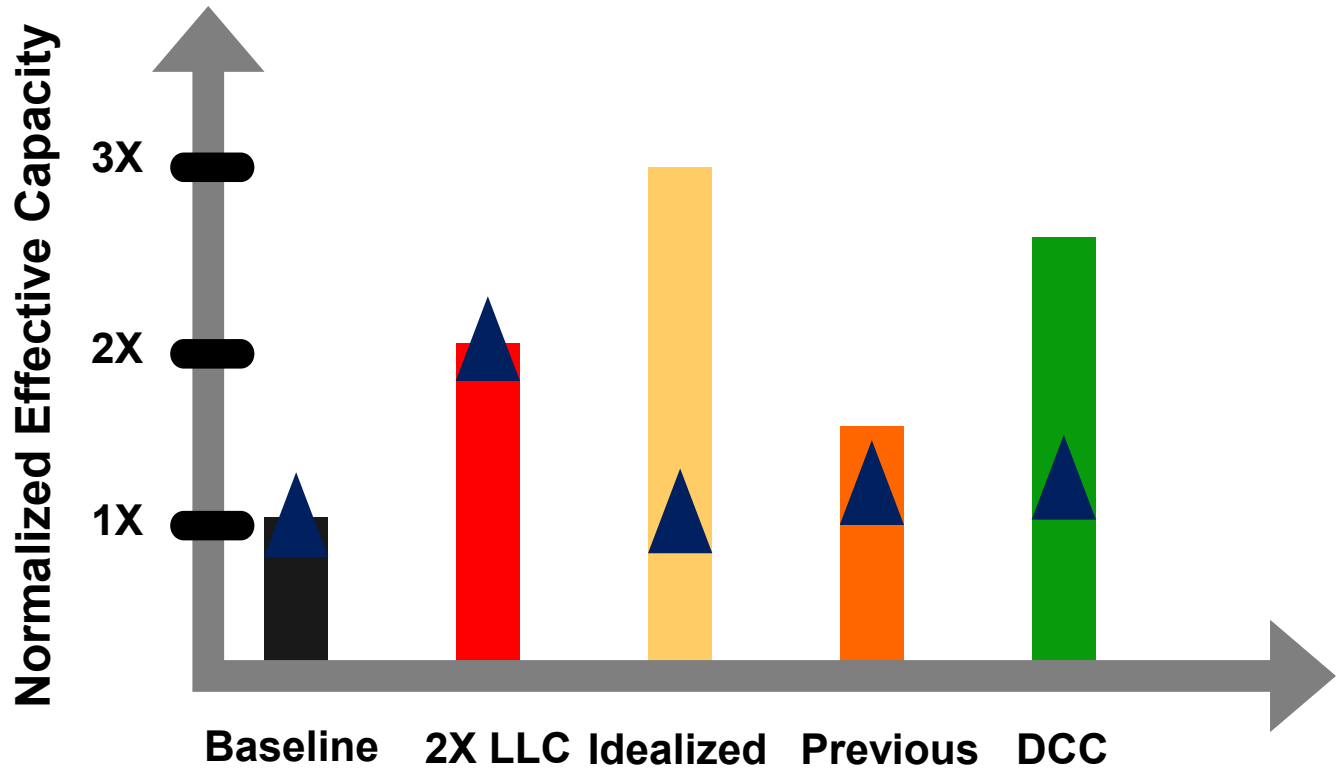
Only 8% Area Overhead  
Outperform 2X LLC





# Our Proposal: Decoupled Compressed Cache (DCC)

## Today at 1:30pm!



# Exploiting GPU Peak-power and Performance Tradeoffs through Reduced Effective Pipeline Latency

Syed Gilani (AMD)

Nam Sung Kim (UW-Madison)

Michael Schulte (AMD Research)

# Problem

- **Thread-level parallelism (TLP) limited by**
  - Thread synchronization patterns
  - Memory access patterns
  - Data dependencies
  - Limited hardware resources
- **Low TLP exposes pipeline latencies**
  - Data-forwarding networks are power hungry

# Contributions

- **Limited forwarding for a few recently executed instructions**
- **Reduce impact of pipeline latency on performance**
  - Low voltage pipelines with negligible impact on performance
- Mean speedups of **23%** (SP/Int) and **33%** (DP) within the same power-budget



# What are we solving?

- GPUs leverage massive multi-threading
  - Core of their latency-tolerance
- Per-thread cache capacity of modern CPUs/GPU

Intel Core i7-4960x	IBM Power7	Oracle UltraSparc T3	NVIDIA Kepler GK 110
32KB L1 2 threads/core <b>16KB/thread</b>	32KB L1 4 threads/core <b>8KB/thread</b>	8KB L1 8 threads/core <b>1KB/thread</b>	48KB L1 2K threads/core <b><u>24B/thread</u></b>

- Efficient caching becomes extremely challenging
  - Low cache hit rates
  - Low cache block reuse
  - Waste in off-chip bandwidth utilization



# What are we solving?

- GPUs leverage massive multi-threading
  - Core of their latency-tolerance
- Per-thread cache capacity of modern CPUs/GPU

Intel Core i7-4960x	IBM Power7	Oracle UltraSparc T3	NVIDIA Kepler GK 110
32KB L1 2 threads/core	32KB L1 4 threads/core	8KB L1 8 threads/core	48KB L1 2K threads/core
<b>16KB/thread</b>	<b>8KB/thread</b>	<b>1KB/thread</b>	<b><u>24B/thread</u></b>

- Efficient caching becomes extremely challenging
  - Low cache hit rates
  - Low cache block reuse
  - Waste in off-chip bandwidth utilization



# What are we solving?

- GPUs leverage massive multi-threading
  - Core of their latency-tolerance
- Per-thread cache capacity of modern CPUs/GPU

Intel Core i7-4960x	IBM Power7	Oracle UltraSparc T3	NVIDIA Kepler GK 110
32KB L1 2 threads/core <b>16KB/thread</b>	32KB L1 4 threads/core <b>8KB/thread</b>	8KB L1 8 threads/core <b>1KB/thread</b>	48KB L1 2K threads/core <b><u>24B/thread</u></b>

- Efficient caching becomes extremely challenging
  - Low cache hit rates
  - Low cache block reuse

**Key target of our paper!**

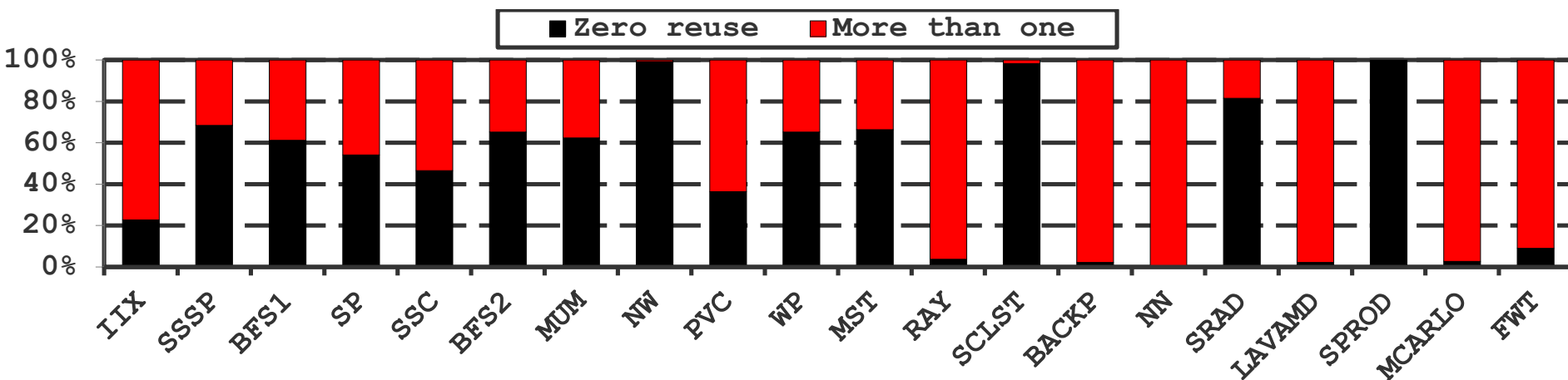
**– Waste in off-chip bandwidth utilization**



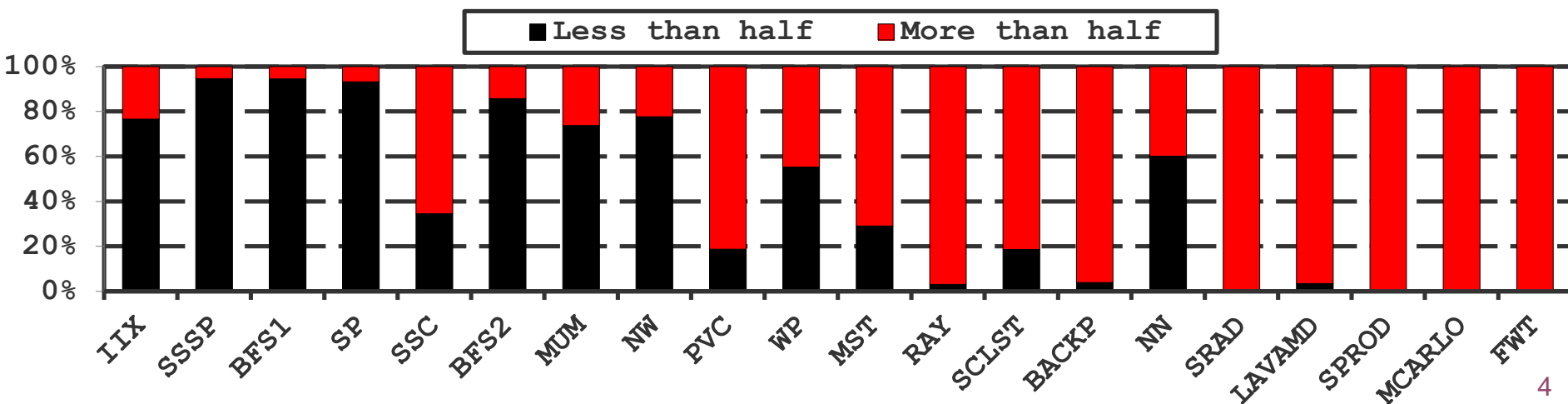


## Last level cache block reuse (**temporal** / **spatial**)

- Number of *repeated accesses* to cache blocks -- **temporal**



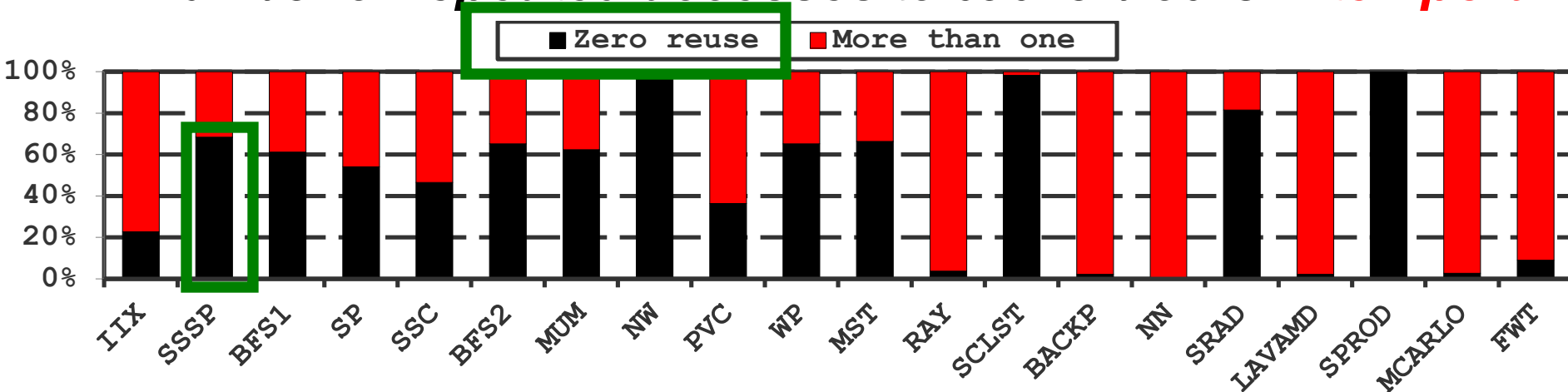
- *Fraction* of cache block data actually used -- **spatial**



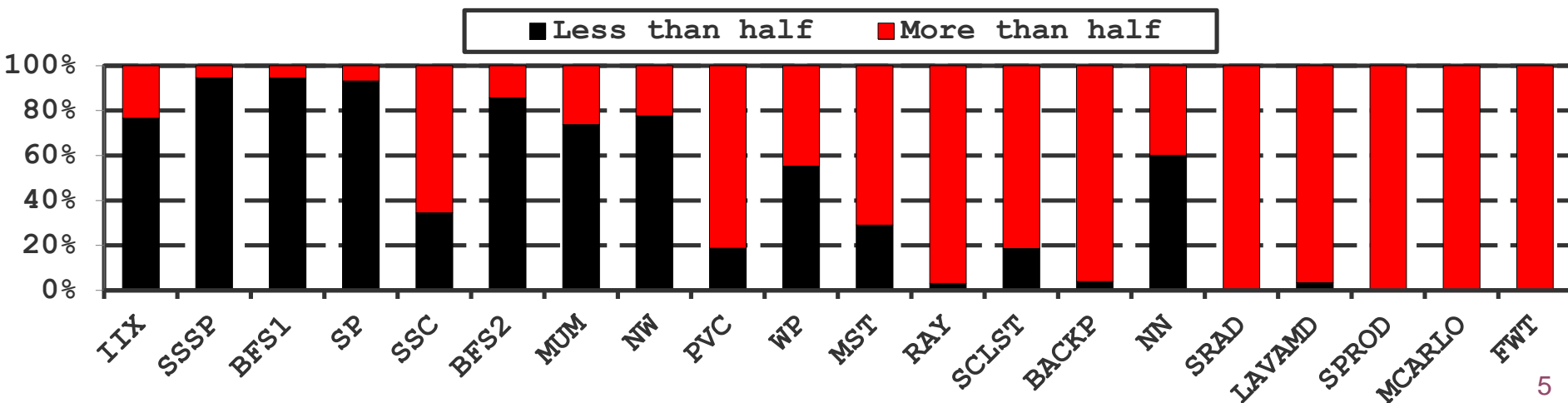


## Last level cache block reuse (**temporal** / **spatial**)

- Number of *repeated accesses* to cache blocks -- **temporal**



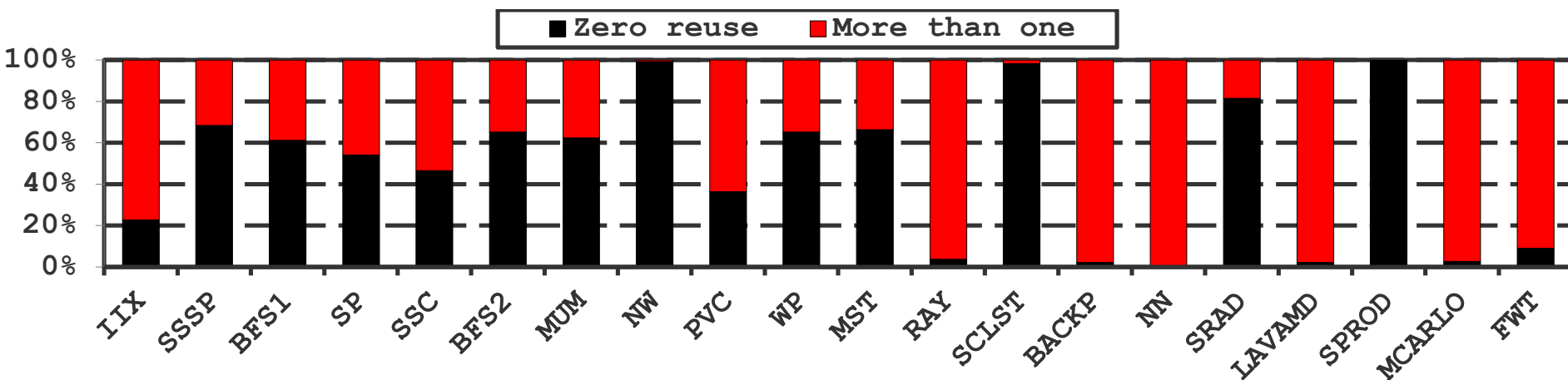
- Fraction* of cache block data actually used -- **spatial**



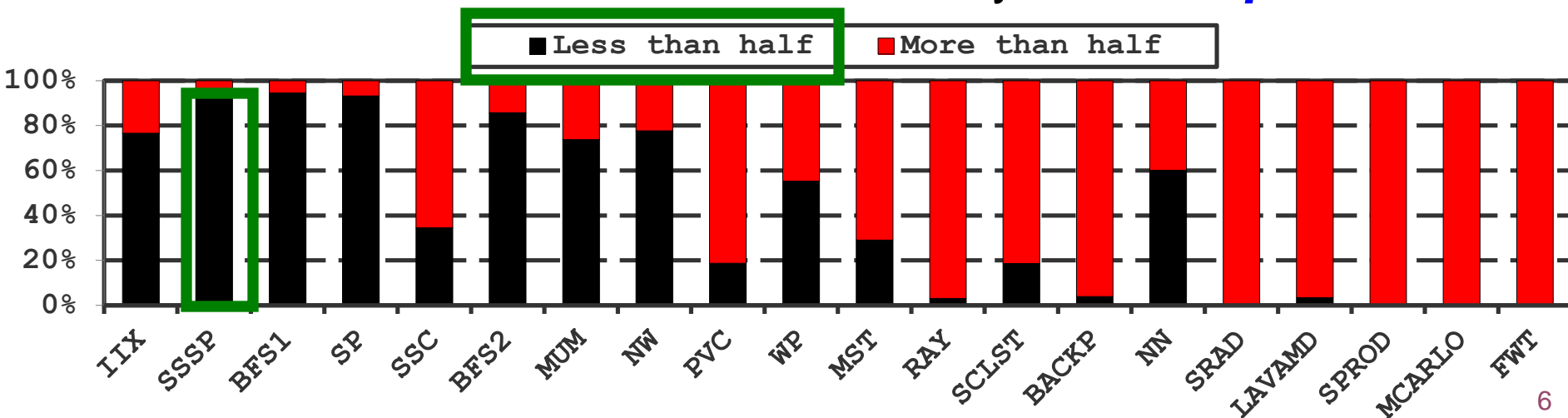


## Last level cache block reuse (**temporal** / **spatial**)

- Number of *repeated accesses* to cache blocks -- **temporal**



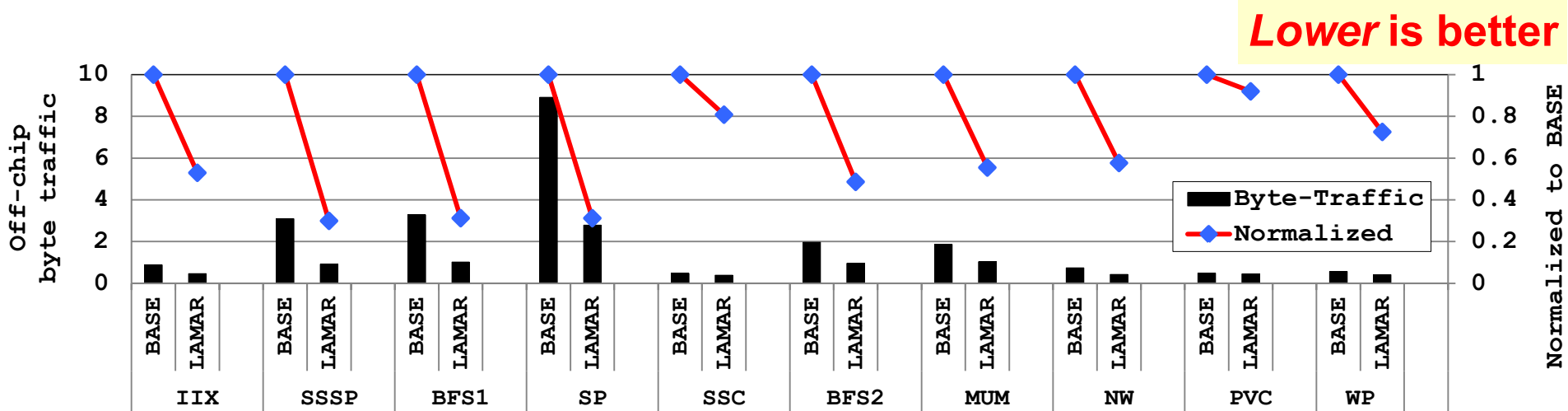
- Fraction* of cache block data actually used -- **spatial**





# How do we solve the problem?

- **Predict** optimal data fetching granularity
  - **Coarse granularity** (fetch *all* cache block-wide data)
  - **Fine granularity** (fetch *just enough* to service GPU core)
  - Reduce number of RD/WR commands to memory



(a) [Byte-traffic/num\_of\_instrs] (left-axis) and the normalization to baseline memory system (right-axis).

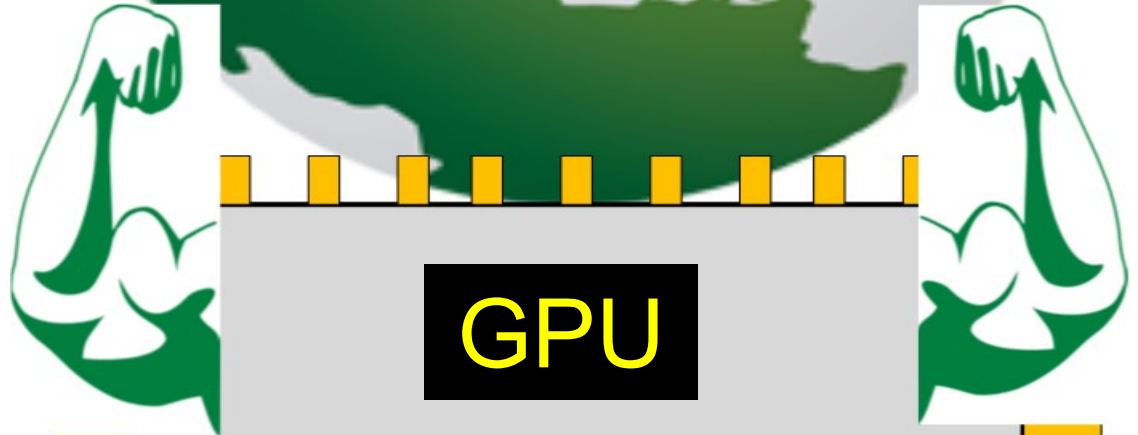
**BASE** : Baseline memory system  
**LAMAR** : Proposed solution



**GPU  
Programmer**



**Locality**



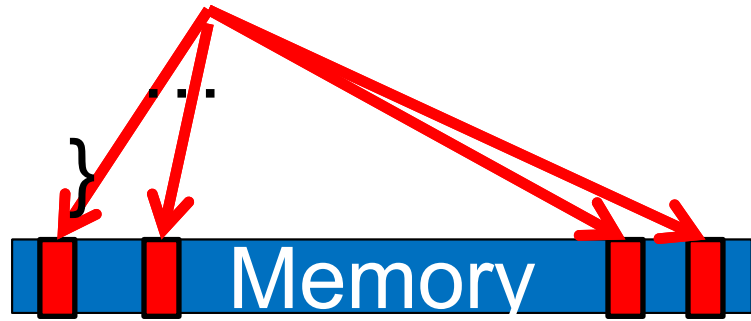
**GPU**



**Divergence-Aware  
Warp Scheduling**

```
while (x[tid]) {  
    load  
    ...  
}
```

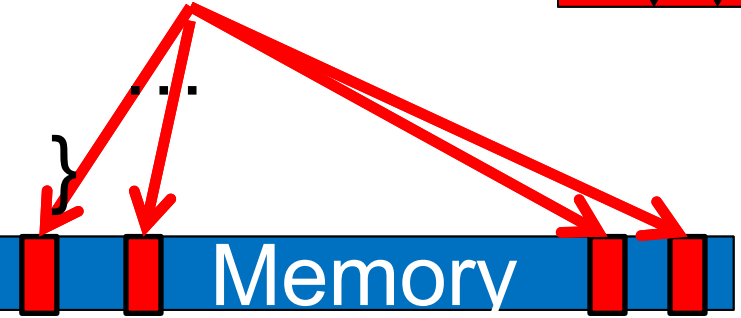
```
while (x[tid]) {  
  load
```



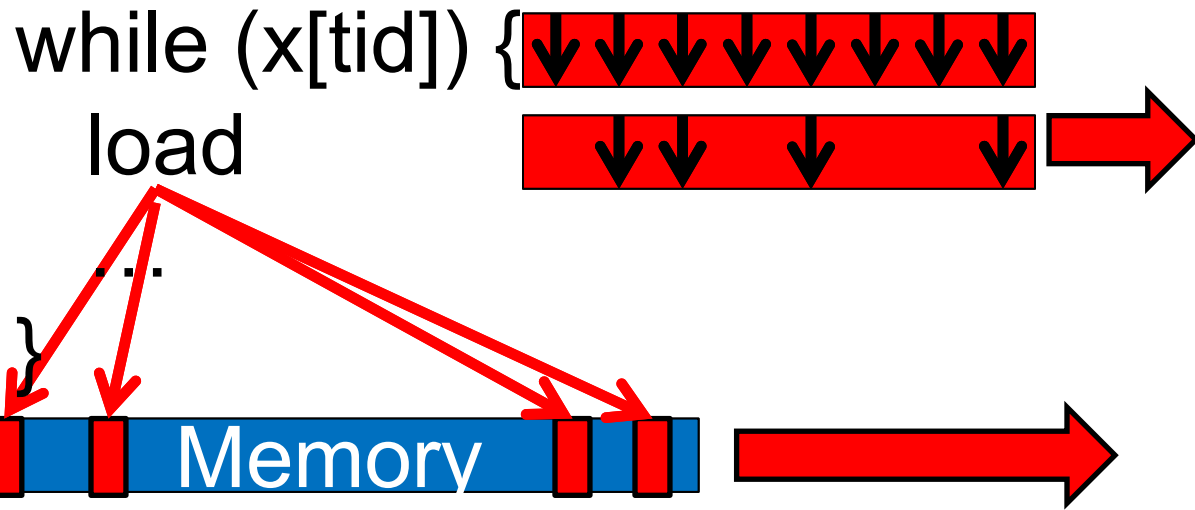
while (x[tid]) {



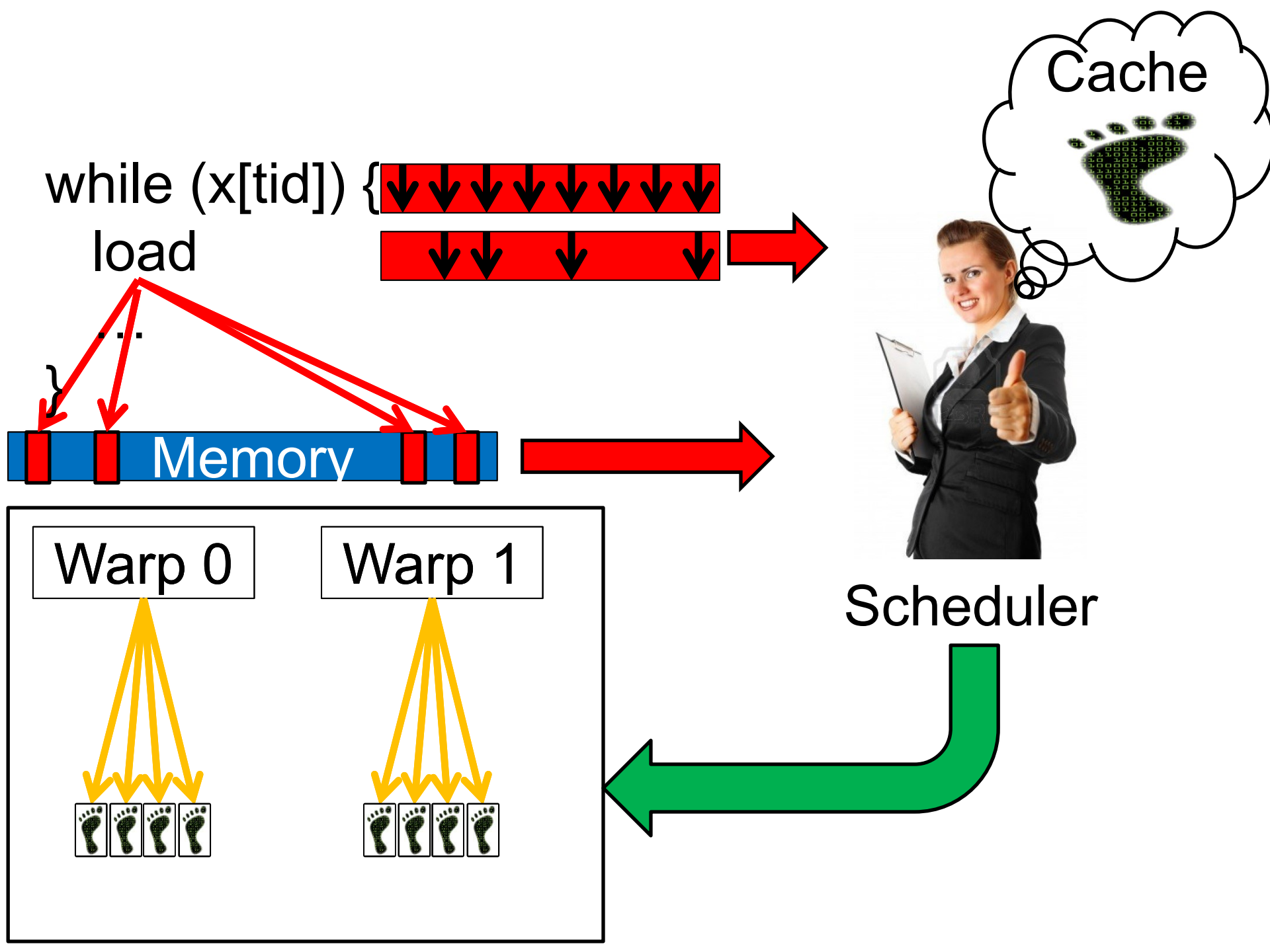
load

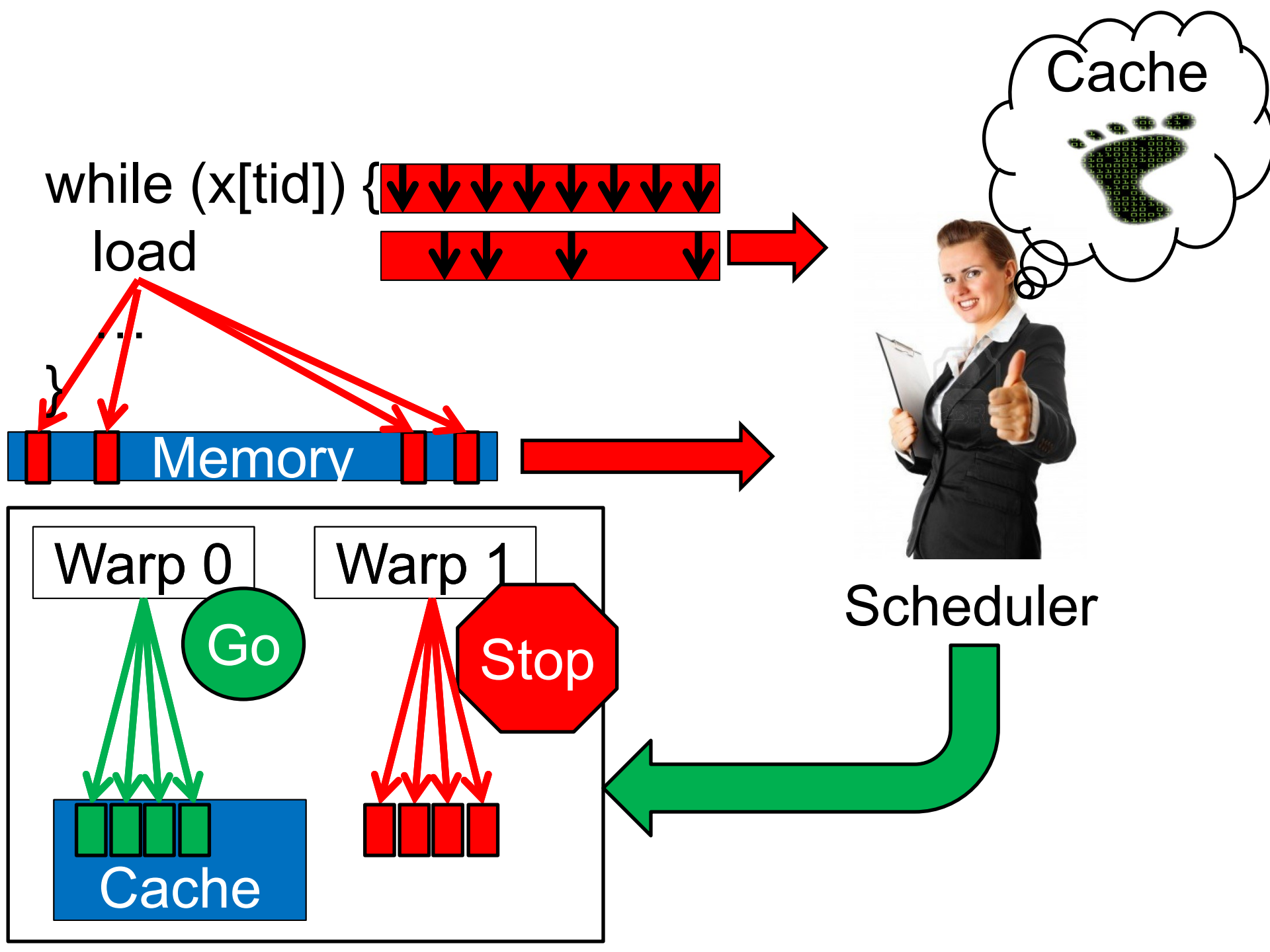






Scheduler





Timothy G. Rogers, Mike O'Connor, Tor M. Aamodt

26%

Speedup

Today 4:30pm Conference Center Ballroom

Timothy G. Rogers, Mike O'Connor, Tor M. Aamodt

**Within**

**4%**

**2.8x**

**less instr**

**Today 4:30pm Conference Center Ballroom**



# Warped-GATES

Gating Aware Scheduling and Power Gating for GPGPUs

*Mohammad Abdel-Majeed, Daniel Wong and Murali Annavaram*

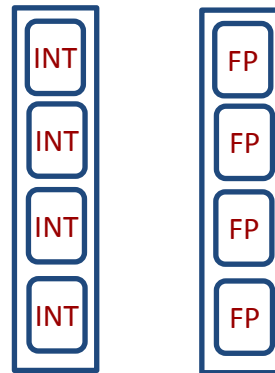
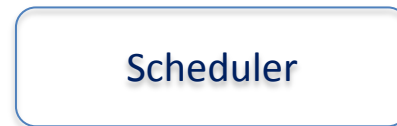
*University of Southern California*



# Power Gating Challenges in GPUs



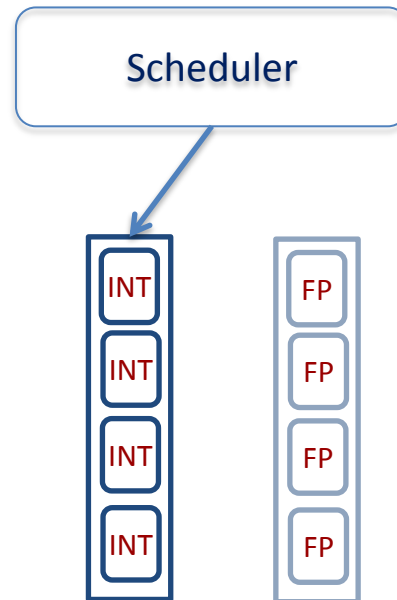
- Scheduler greedily issues ready instructions
  - Agnostic to instruction type.



# Power Gating Challenges in GPUs



- Scheduler greedily issues ready instructions
  - Agnostic to instruction type.

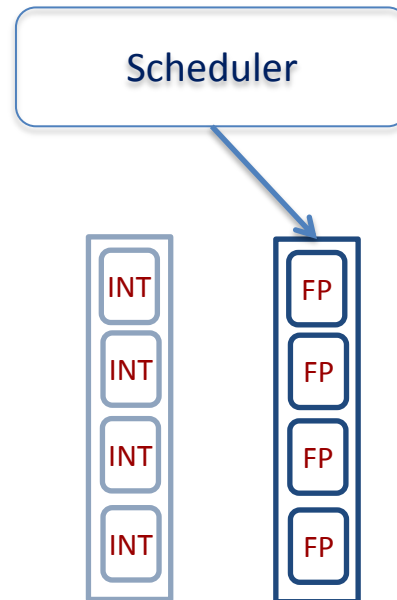




# Power Gating Challenges in GPUs



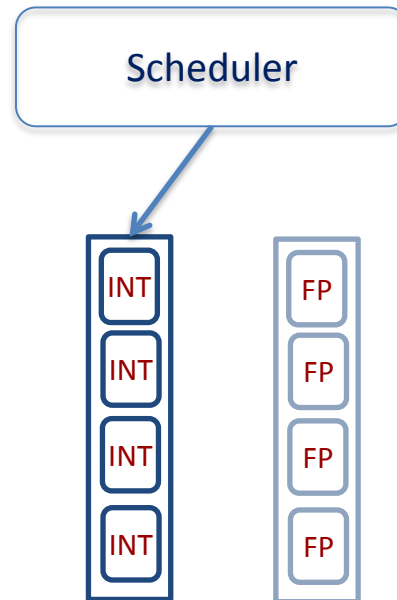
- Scheduler greedily issues ready instructions
  - Agnostic to instruction type.



# Power Gating Challenges in GPUs



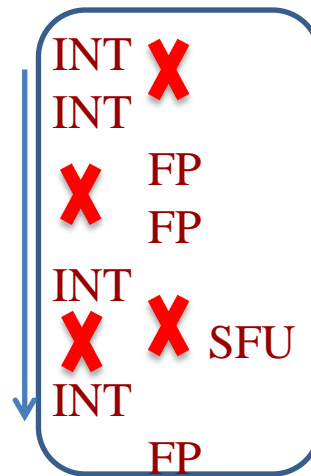
- Scheduler greedily issues ready instructions
  - Agnostic to instruction type.



# Power Gating Challenges in GPUs



- Scheduler greedily issues ready instructions
  - Agnostic to instruction type.



# Proposed Techniques



- Gating Aware Scheduler (GATES)
  - Gives priority to same instruction type during scheduling.
  - Is able to increase the length of the idle periods.
  - **Idle periods are not long enough to avoid negative savings!!**
- Blackout technique
  - Eliminates negative savings by forcing the unit to stay in power gating state.

1.5X

Static Power Savings

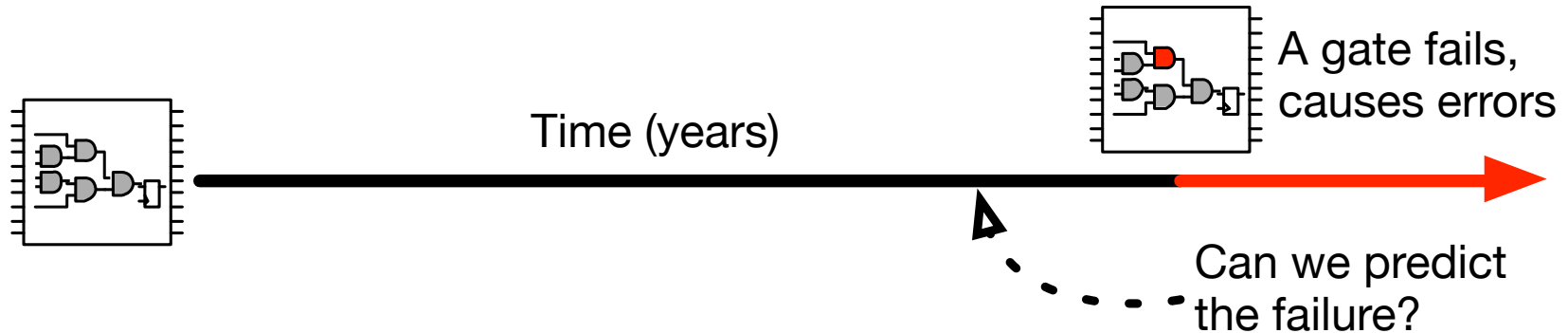
<1%

Performance Overhead

# Virtually Aged Sampling DMR

## Unifying Circuit Failure Prediction and Detection

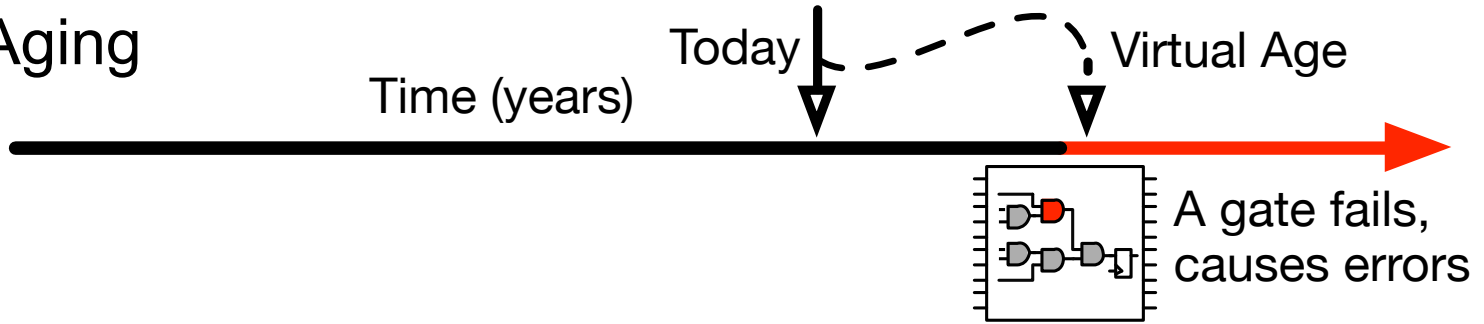
Raghuraman Balasubramanian  
Karthikeyan Sankaralingam





# Virtually Aged Sampling DMR

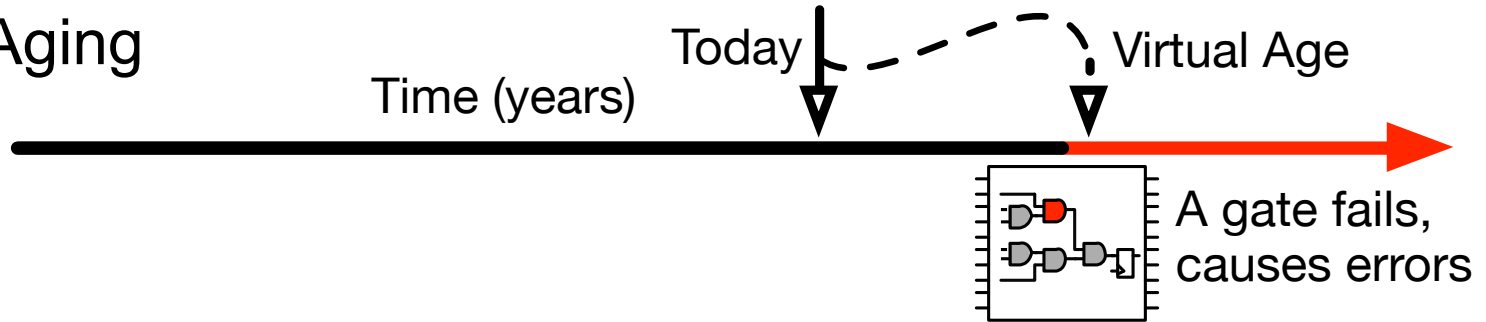
Virtual Aging



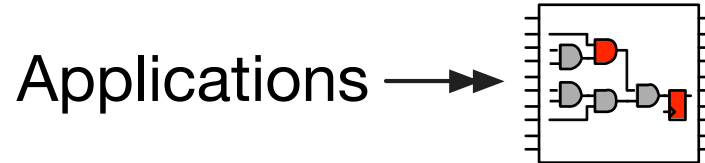


# Virtually Aged Sampling DMR

Virtual Aging



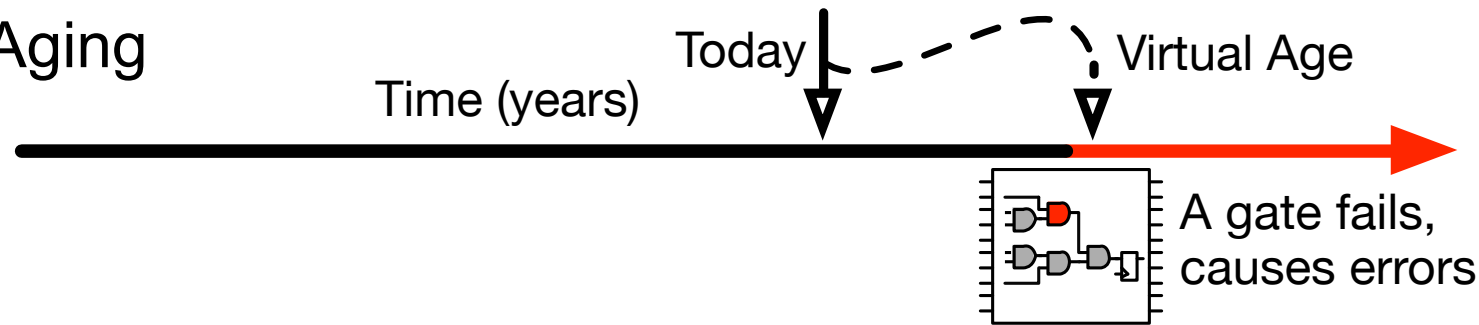
Fault Exposure



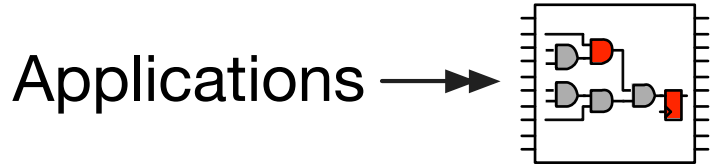


# Virtually Aged Sampling DMR

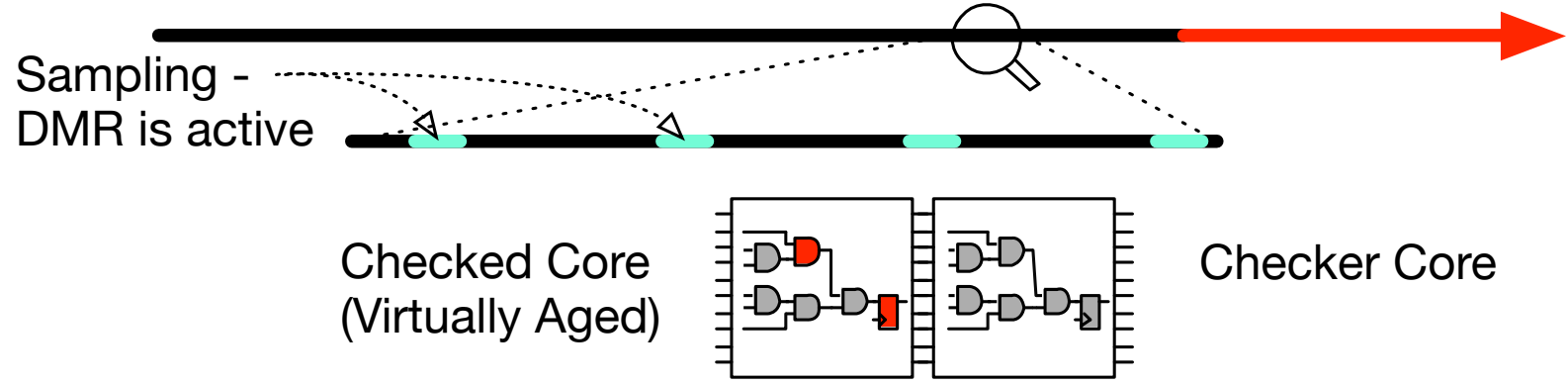
Virtual Aging



Fault Exposure



Detect Errors







# Virtually Aged Sampling DMR

**$\mu$ -Processor Failure Prediction Technique**



# Virtually Aged Sampling DMR

$\mu$ -Processor Failure Prediction Technique

Comprehensive Coverage



# Virtually Aged Sampling DMR

$\mu$ -Processor Failure Prediction Technique

Comprehensive Coverage

No Performance Overhead

Less than 0.7 % Energy Overhead



# Virtually Aged Sampling DMR

$\mu$ -Processor Failure Prediction Technique

Comprehensive Coverage

No Performance Overhead

Less than 0.7 % Energy Overhead

Today @ 3:30PM Alpha Gamma Rho



TEXAS A&M  
UNIVERSITY

# *Use-it or Lose-it: Wearout and Lifetime in Future Chip-Multiprocessors*

Hyungjun Kim,<sup>1</sup> Arseniy Vitkovsky,<sup>2</sup>

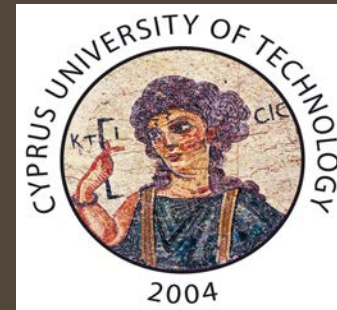
Paul V. Gratz,<sup>1</sup> Vassos Soteriou<sup>2</sup>

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

<sup>2</sup> Department of Electrical Engineering, Computer Engineering and  
Informatics, Cyprus University of Technology



TEXAS A&M  
UNIVERSITY



# Chip-multiprocessor Wearout



ITRS: Rates of wearout induced failure to increase 10X in 10 years

- HCI and NBTI: transistor slowdown with use

Wearout effects in CMPs:

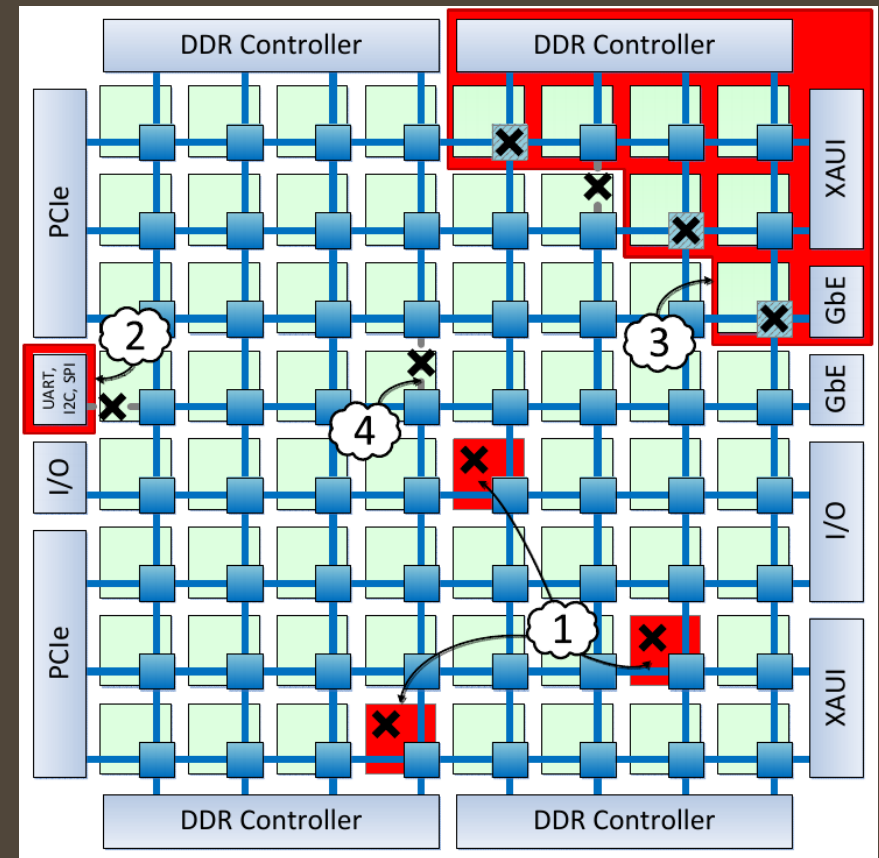
Recoverable failures:

- 1) Core failure
  - Failure detection and remapping

Non-recoverable failures:

- 2) I/O device disconnection
  - Device unreachable
- 3) Network partition
  - Disruption of communication between cores
- 4) Individual link breakage
  - Deadlock potential

Interconnect critical point of failure



A 64-core Chip-Multiprocessor (CMP) with various peripherals interconnected via a 2-D Mesh, all failure scenarios illustrated

# Use it or Lose it

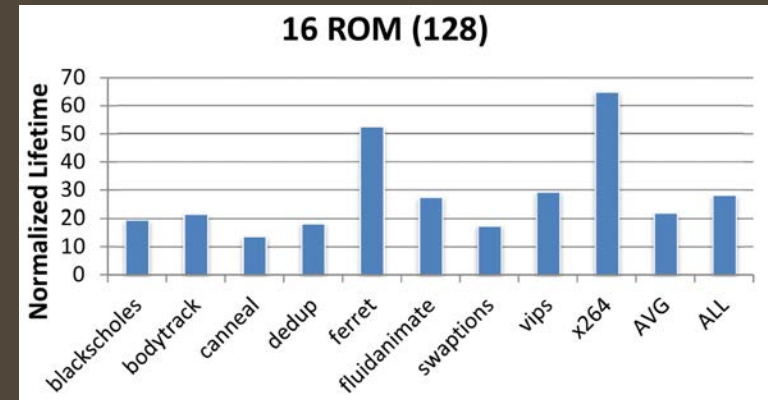


Analysis of real CMP workloads:

- Low loads in interconnect
- NBTI causes critical path slowdown
- *Lack* of load leads to interconnect breakdown and failure

The *Use it or Lose it*, wear-resistant router microarchitecture

- *Increases* utilization of router critical path
- *22x lifetime improvement!*



Lifetime improvement of 8x8 CMP executing applications from the PARSEC benchmark suite

Session 2B

(Alpha Gamma Rho Room)

4:00 PM today!

*MICRO-46, 9<sup>th</sup> December- 2013*  
*Davis, California*



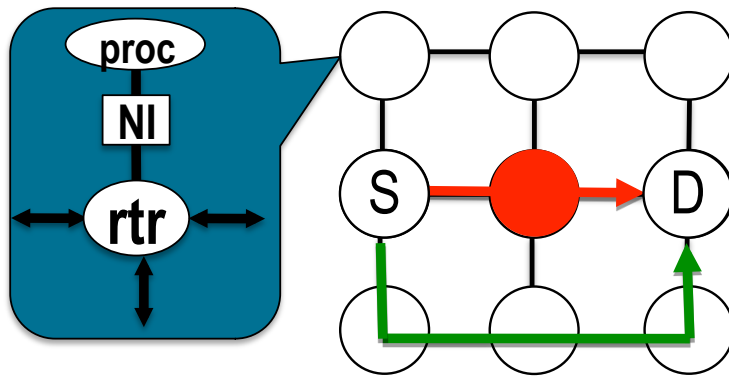
# uDIREC: Unified Diagnosis and Reconfiguration for Frugal Bypass of NoC Faults

Ritesh Parikh and Valeria Bertacco

*Electrical Engineering & Computer Science Department*  
*The University of Michigan, Ann Arbor*



# Unified Diagnosis and Reconfiguration



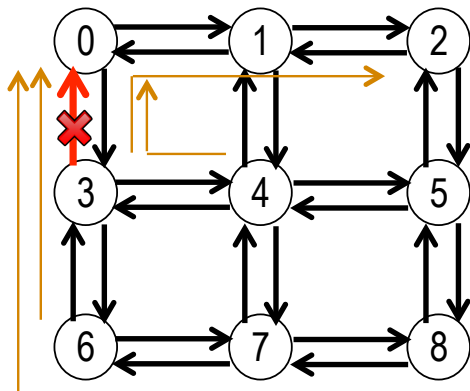
- ✗ cannot resend
- ✓ need to re-route around fault

Our contributions:

- **Fault Diagnosis** at fine granularity
- **Integrated Reconfiguration** to find new routes

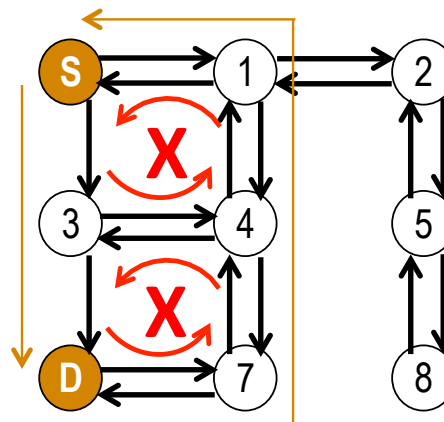
## Diagnosis

- End-to-end scheme in SW
- Based on analyzing faulty routes
- Passive and fine-grained



## Reconfiguration

- Based on a novel routing algorithm
- Tightly integrated with the diagnosis scheme
- Unconstrained by number and location of fault

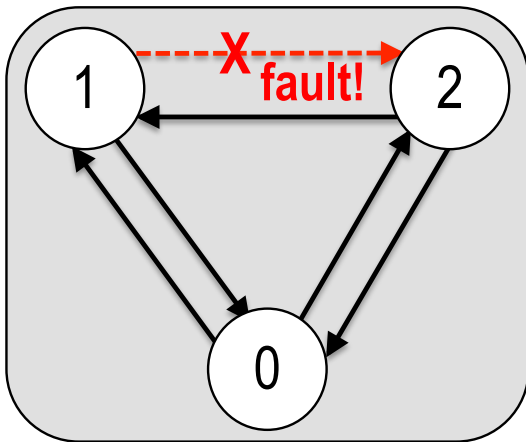


Faulty irregular network with deadlock-free routes

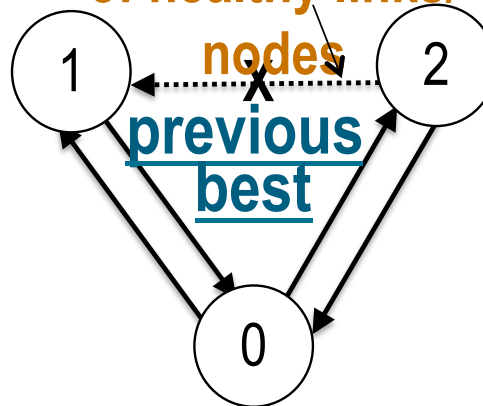
# Reliability and Performance Benefits

- Dedicated testing is not required → no overhead in absence of errors
- Unified implementation in software → low area overhead

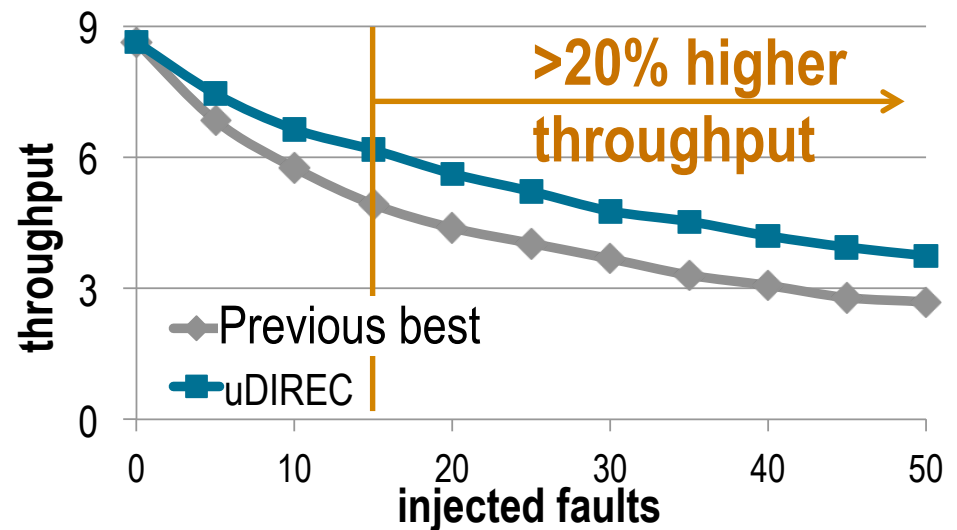
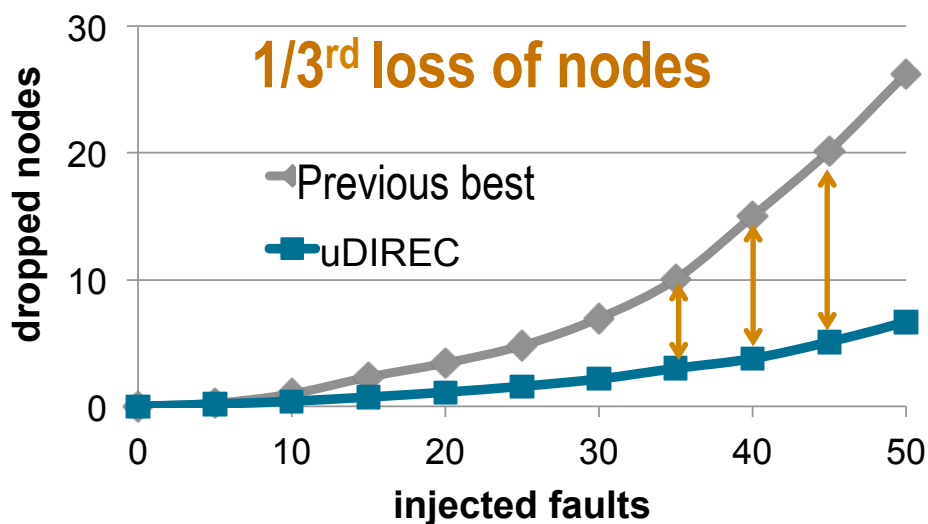
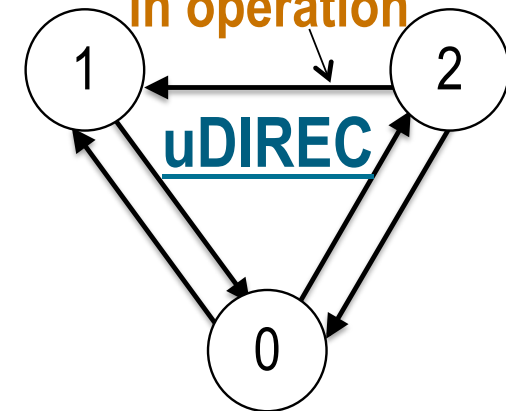
fault manifestation



unnecessary loss of healthy links/  
nodes



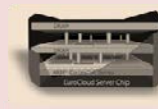
healthy links kept in operation





# Implicit-Storing and Redundant- Encoding-of-Attribute Information in Error-Correction-Codes

Yiannakis Sazeides<sup>1</sup>, Emre Ozer<sup>2</sup>, Danny Kershaw<sup>3</sup>, **Panagiota Nikolaou**<sup>1</sup>,  
Marios Kleanthous<sup>1</sup>, Jaume Abella<sup>4</sup>  
<sup>1</sup>University of Cyprus, <sup>2</sup>ARM, <sup>3</sup>NXP, <sup>4</sup>Barcelona Supercomputing Center





## Implicit Storing (IS)

Leverage error correction codes used for cache and memory data protection

- **encode** extra information
- **without storing** the information
- **infer** the information on reads

Based on **error** and **erasure** coding

Needs **shortened codes**: the number of protected data bits to be smaller than what can be protected by an error correction code

😊: **reduce area and energy** with **low performance overhead**

☹️: Hurts error correction **code strength**



## Redundant Encoding of Attribute Information (REA)

Exploit fine granularity of protection in caches and memory

- **encode** the same extra information in **multiple codewords**
- **decode** the extra information from **multiple codewords**

Needs the **multiple codewords** to be **correlated**

😊: improve **strength of the code** that **implicitly stores** or **tags** extra info

😐: not full strength recovery

😊: **minimal area, energy, and timing overheads**

Several **IS** & **REA** uses: **reliability, performance, security, energy**



## Redundant Encoding of Attribute Information (REA)

Exploit fine granularity of protection in caches and memory

- **encode** the same extra information in **multiple codewords**
- **decode** the extra information from **multiple codewords**

Needs the **multiple codewords** to be **correlated**

☺: improve **strength of the code** that **implicitly stores** or **tags** extra info

☹: not full strength recovery

☺: **minimal area, energy, and timing overheads**

Several **IS** & **REA** uses: **reliability, performance, security, energy**

**We would be glad to see you all at:**

***Session 2B - Resilience I (15:30-17:30)***

# Linearly Compressed Pages: A Main Memory Compression Framework with Low Complexity and Low Latency

---

**Gennady Pekhimenko,**

Vivek Seshadri , Yoongu Kim,

Hongyi Xin, Onur Mutlu,

Todd C. Mowry

Phillip B. Gibbons,

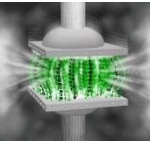
Michael A. Kozuch

**Carnegie Mellon University**



# Summary

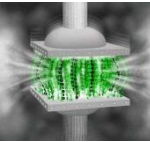
- Main memory is a limited shared resource
- **Observation**: Significant data redundancy
- **Old Idea**: Compress data in main memory





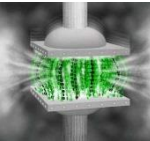
# Summary

- Main memory is a limited shared resource
- **Observation**: Significant data redundancy
- **Old Idea**: Compress data in main memory
- **Problem**: How to avoid inefficiency in address computation?

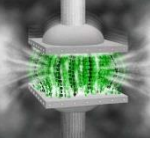


# Summary

- Main memory is a limited shared resource
- **Observation**: Significant data redundancy
- **Old Idea**: Compress data in main memory
- **Problem**: How to avoid **inefficiency in address computation?**
- **Solution**: Linearly Compressed Pages (LCP): fixed-size cache line granularity compression



# Summary

- Main memory is a limited shared resource
- **Observation**: Significant data redundancy
- **Old Idea**: Compress data in main memory 
- **Problem**: How to avoid **inefficiency in address computation**?
- **Solution**: Linearly Compressed Pages (LCP): fixed-size cache line granularity compression
  1. Increases capacity (**62%** on average)
  2. Decreases bandwidth consumption (**24%**)
  3. Improves overall performance (**13.9%**)

# Linearly Compressed Pages (LCP)

Uncompressed Page (4KB: 64\***64B**)



# Linearly Compressed Pages (LCP)

Uncompressed Page (4KB:  $64 \times 64B$ )



4:1 Compression



Compressed  
Data (1KB)

# Linearly Compressed Pages (LCP)

Uncompressed Page (4KB:  $64 \times 64B$ )



4:1 Compression

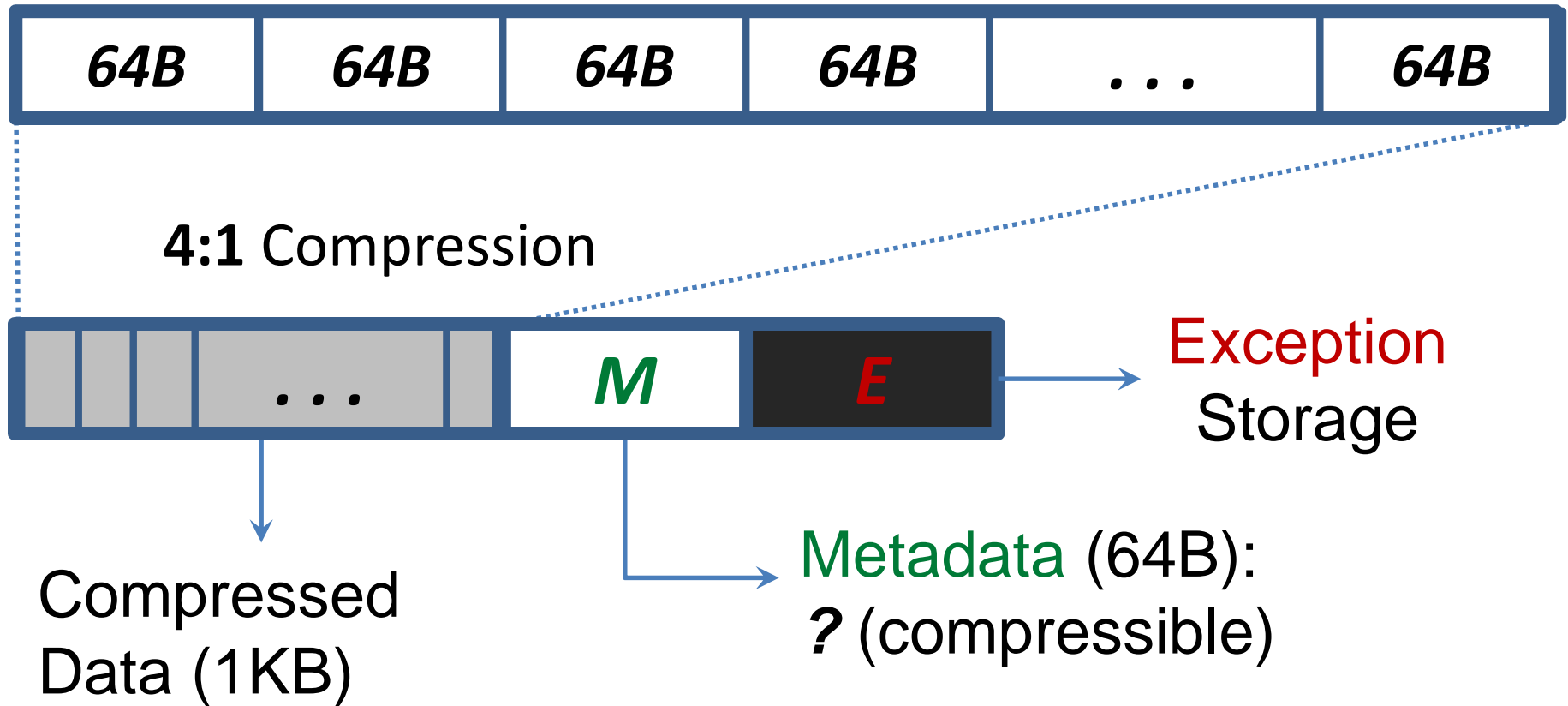


Compressed  
Data (1KB)

Metadata (64B):  
? (compressible)

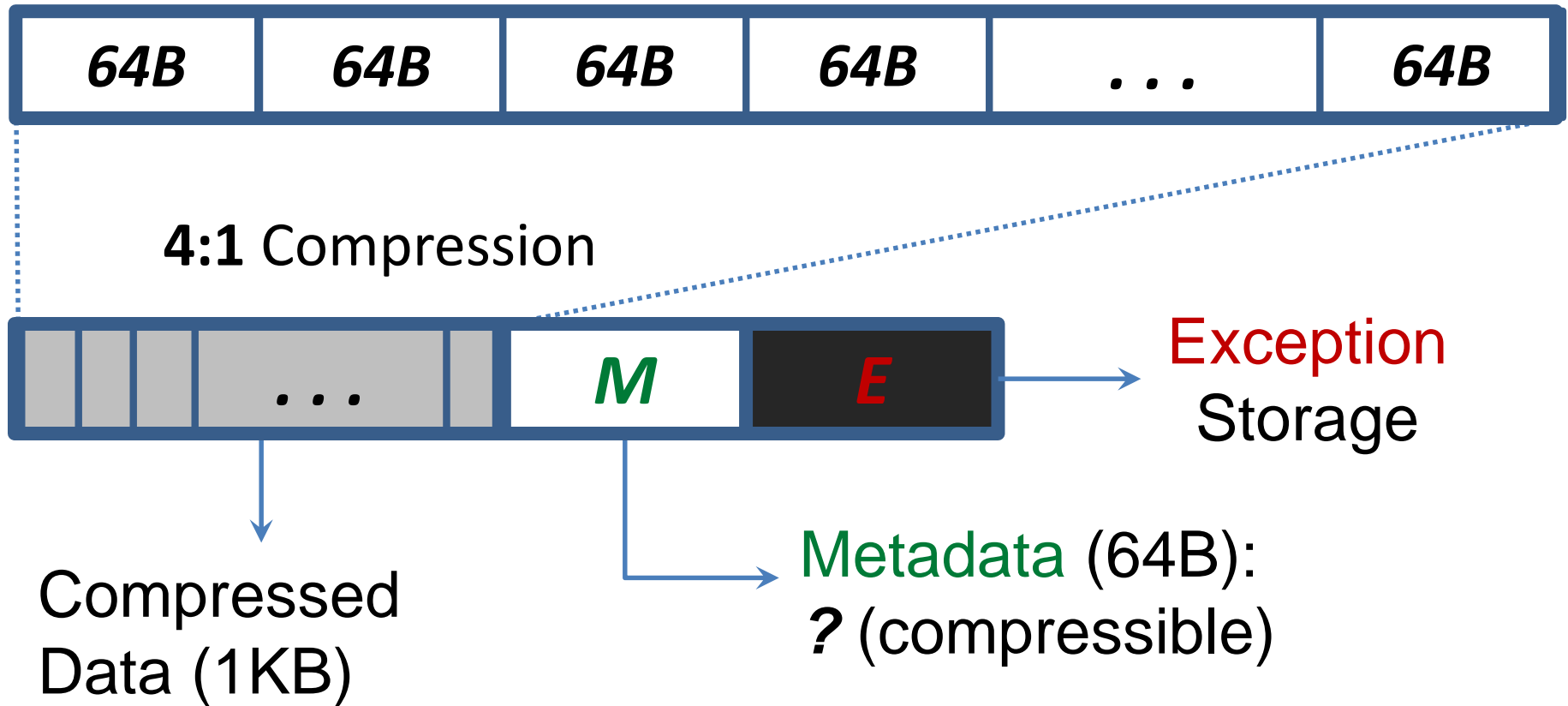
# Linearly Compressed Pages (LCP)

Uncompressed Page (4KB:  $64 \times 64\text{B}$ )



# Linearly Compressed Pages (LCP)

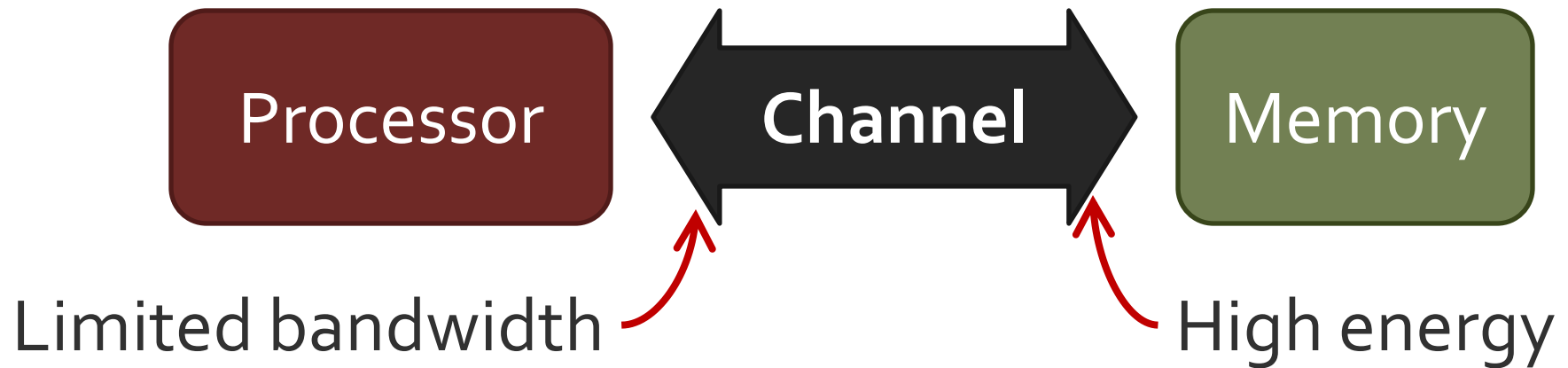
Uncompressed Page (4KB:  $64 \times 64\text{B}$ )



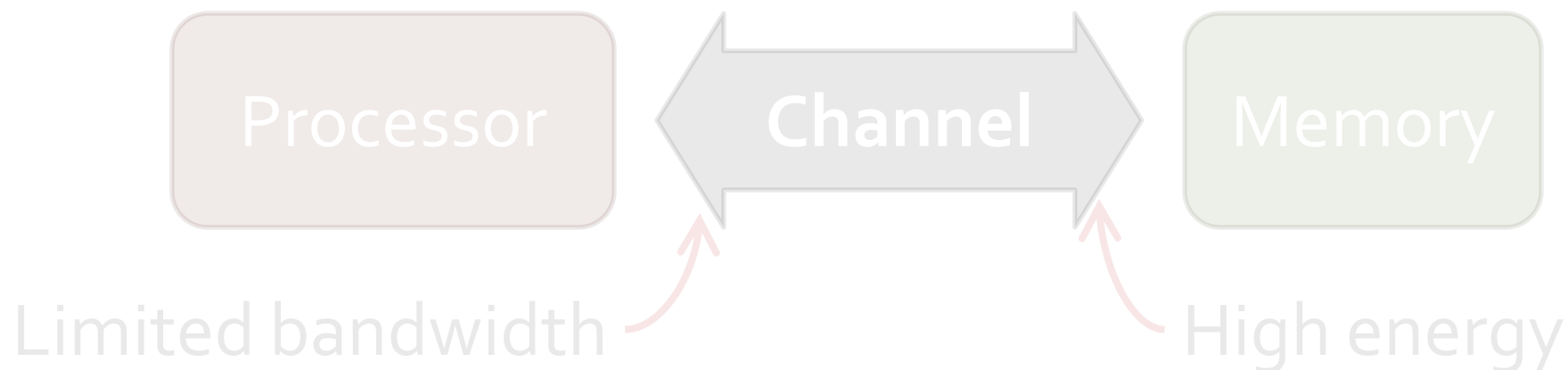
Tomorrow, **8:30am**, Session 3A



# RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

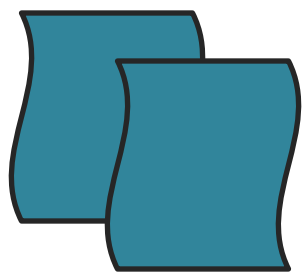


# RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

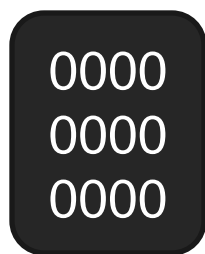


## Bulk Data Copy

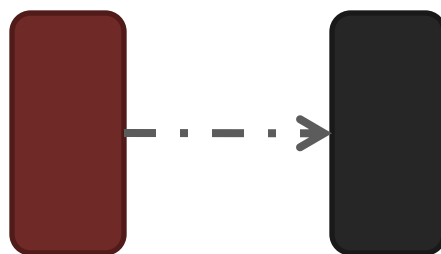
## Data Initialization



Forking



Zeroing

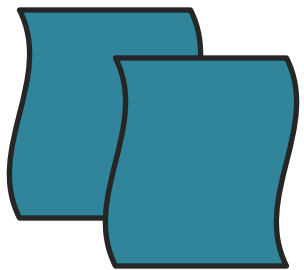
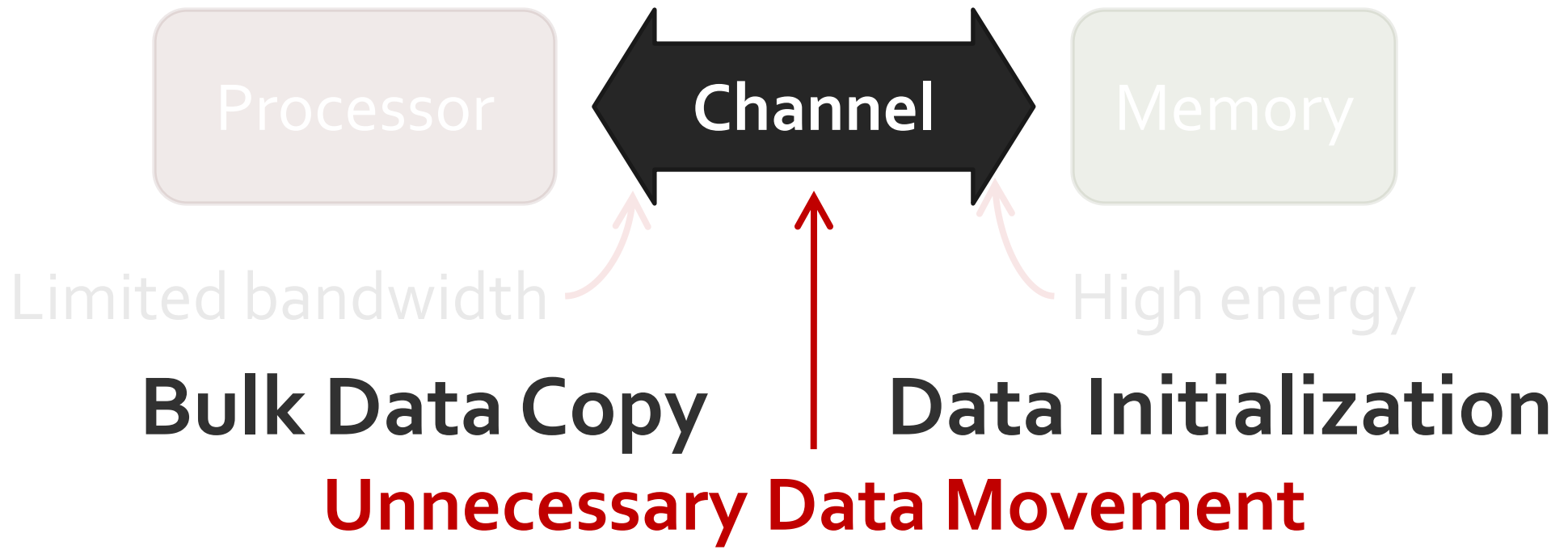


Checkpointing

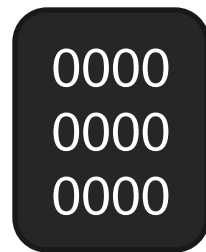


VM Cloning

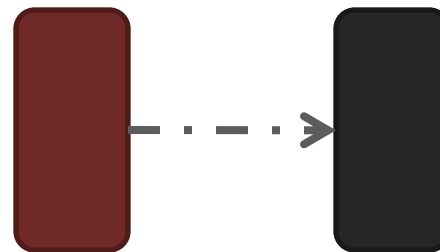
# RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization



Forking



Zeroing

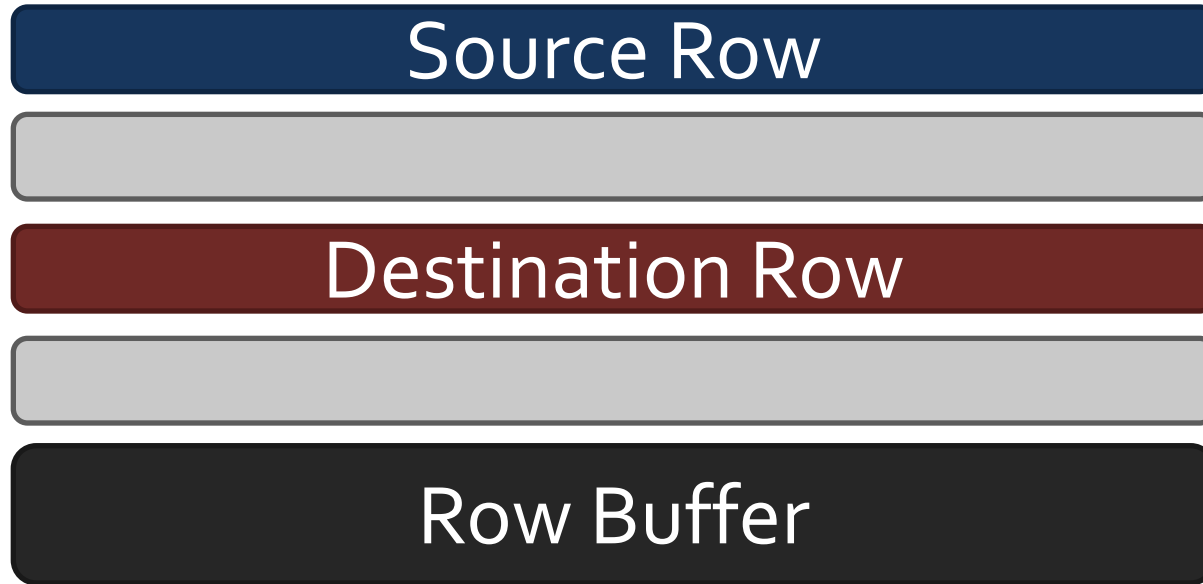


Checkpointing

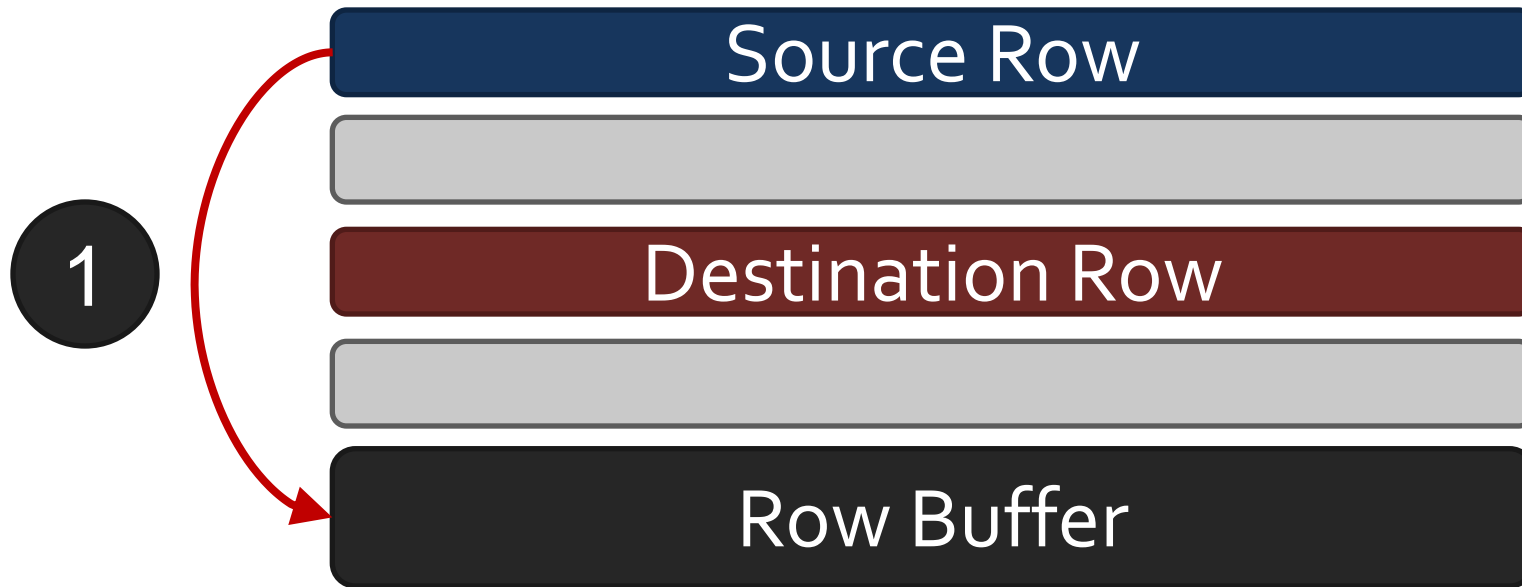


VM Cloning

# RowClone: In-DRAM Bulk Copy & Initialization

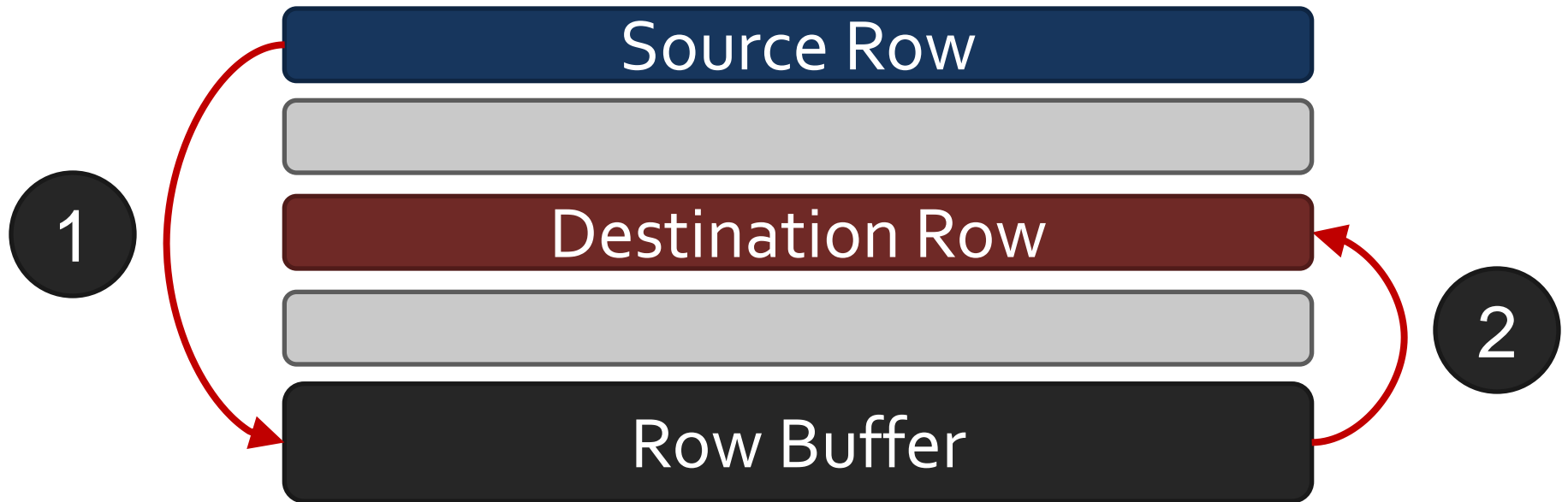


# RowClone: In-DRAM Bulk Copy & Initialization



Copy from source  
row to row buffer

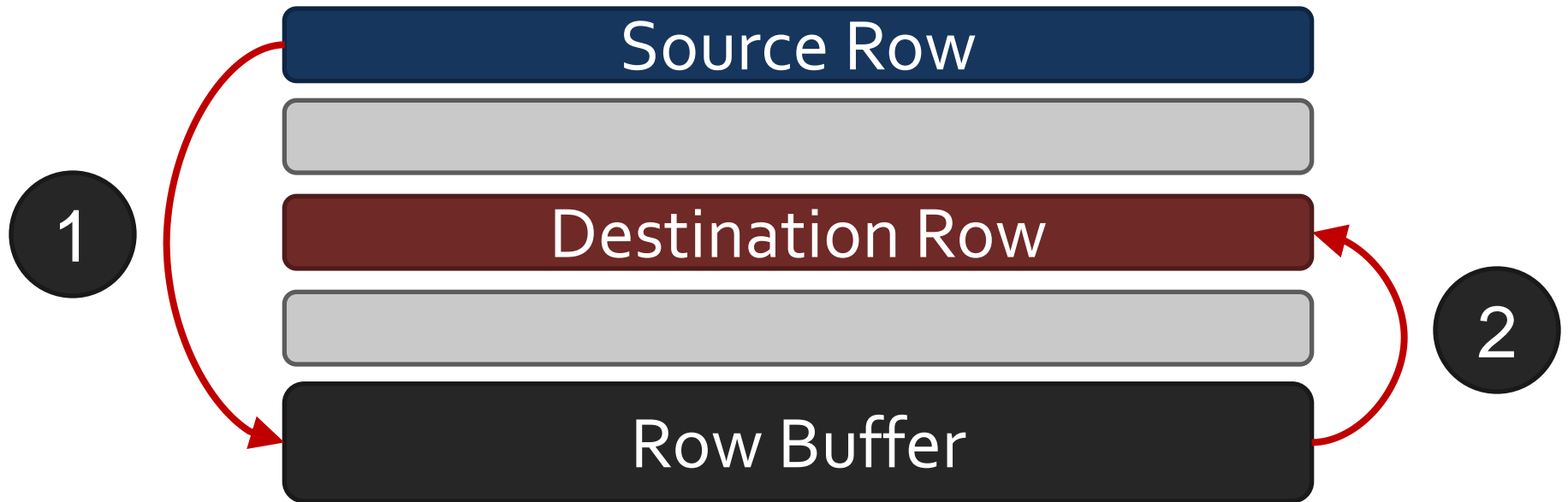
# RowClone: In-DRAM Bulk Copy & Initialization



Copy from source row to row buffer

Copy from row buffer to destination row

# RowClone: In-DRAM Bulk Copy & Initialization



Copy from source row to row buffer

Latency

**11x**



Copy from row buffer to destination row

Energy

**74x**



Very few changes to DRAM  
(0.01% increase in die area)

# RowClone: In-DRAM Bulk Copy & Initialization

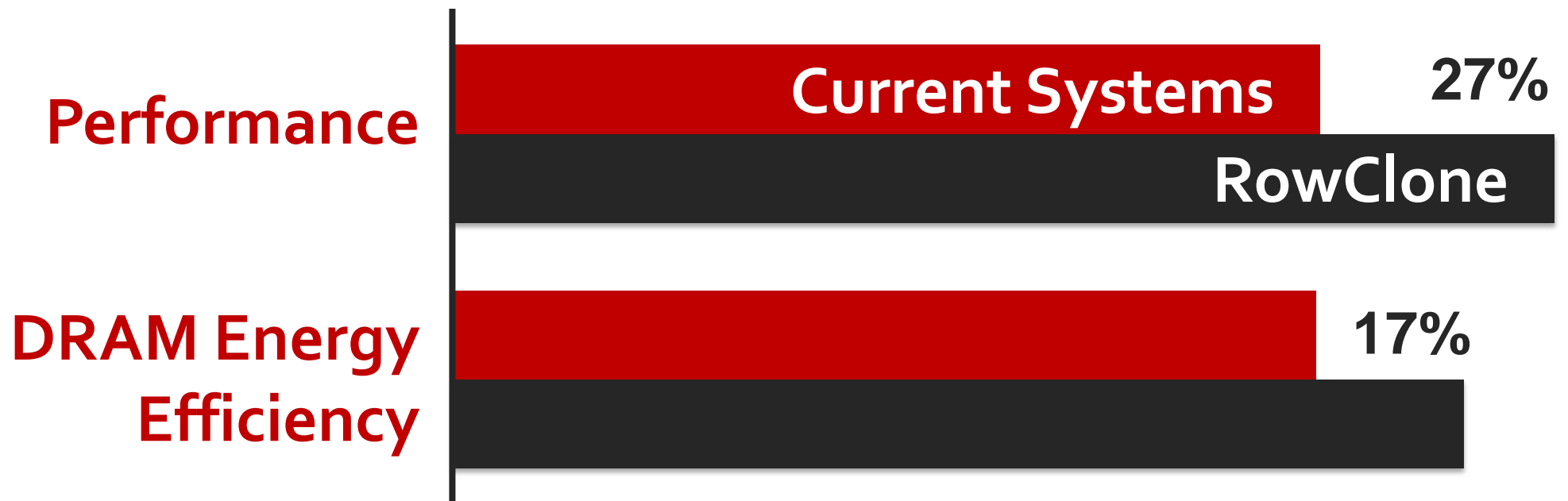
- End-to-end system design to exploit DRAM substrate
- Several applications that benefit from RowClone



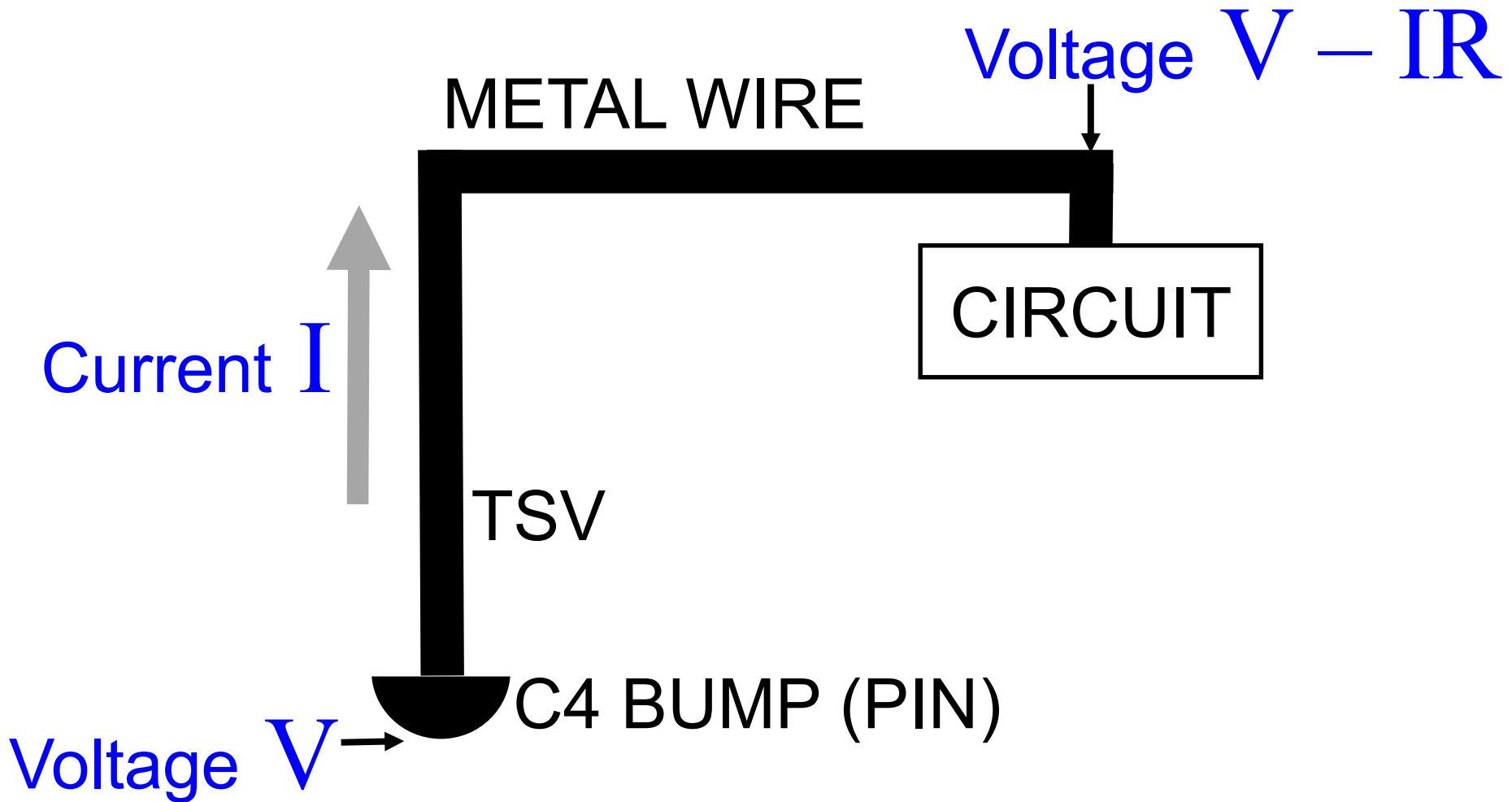
# RowClone: In-DRAM Bulk Copy & Initialization

- End-to-end system design to exploit DRAM substrate
- Several applications that benefit from RowClone

## 8-Core System

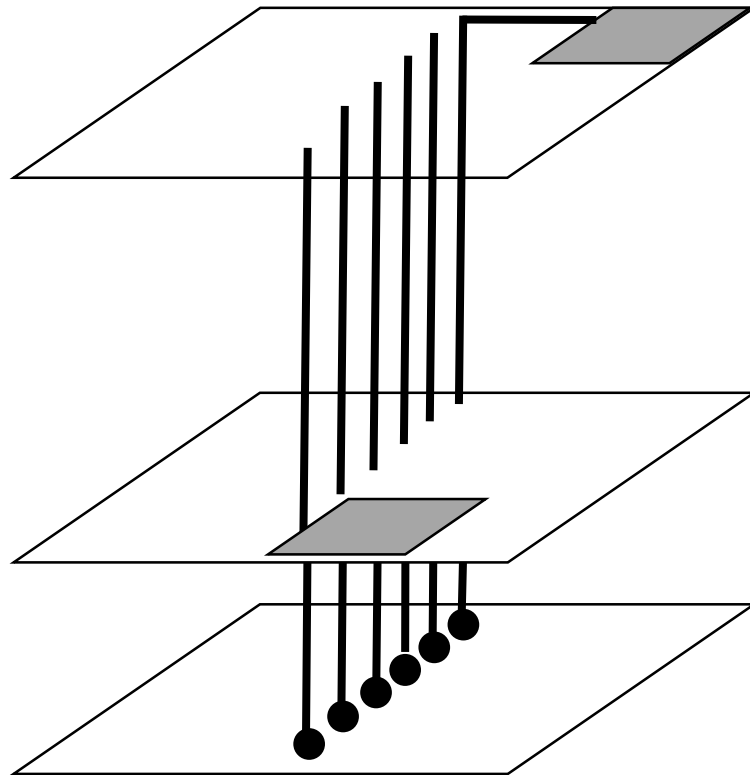


# WHAT IS IR DROP ?



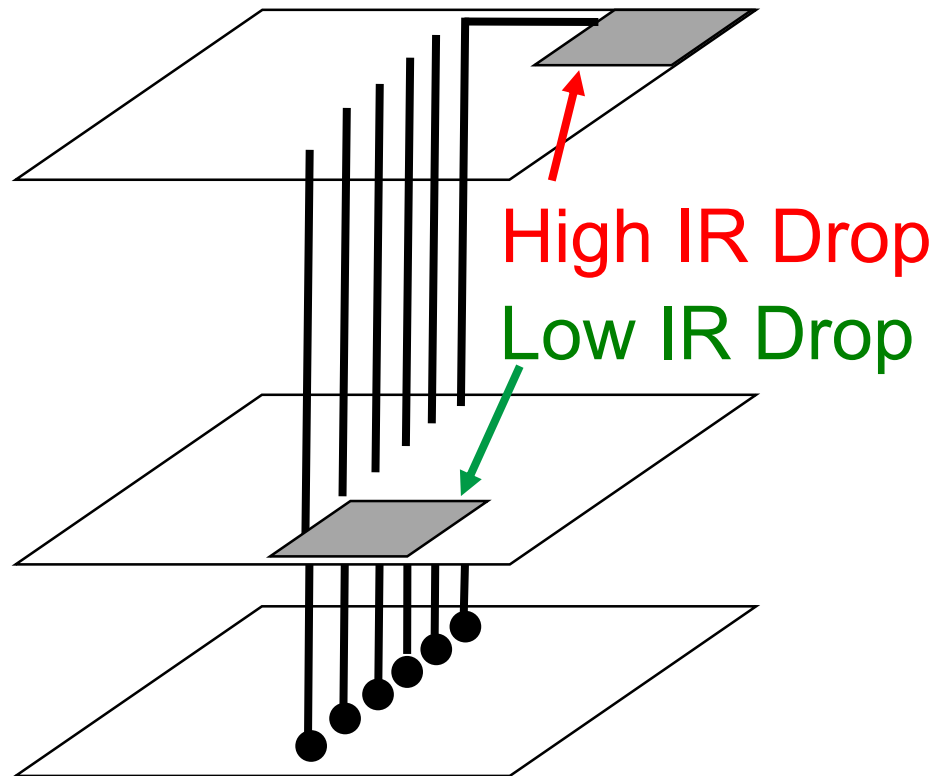
# KEY CONTRIBUTIONS

- ① A 3D memory package with few pins & TSVs



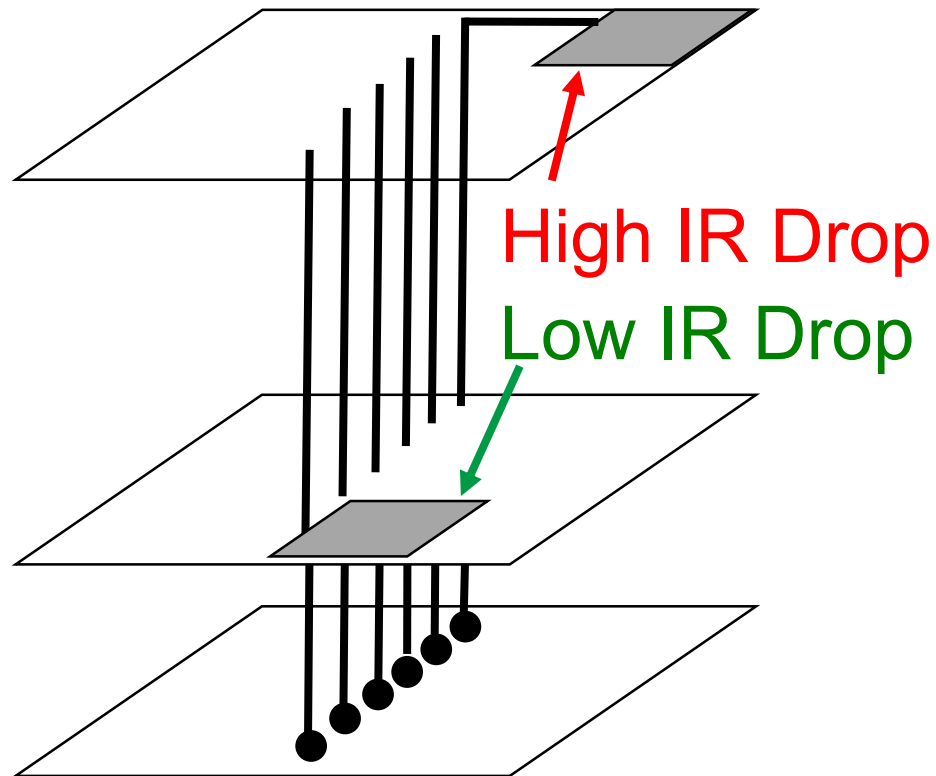
# KEY CONTRIBUTIONS

- ② Spice analysis to show voltage maps



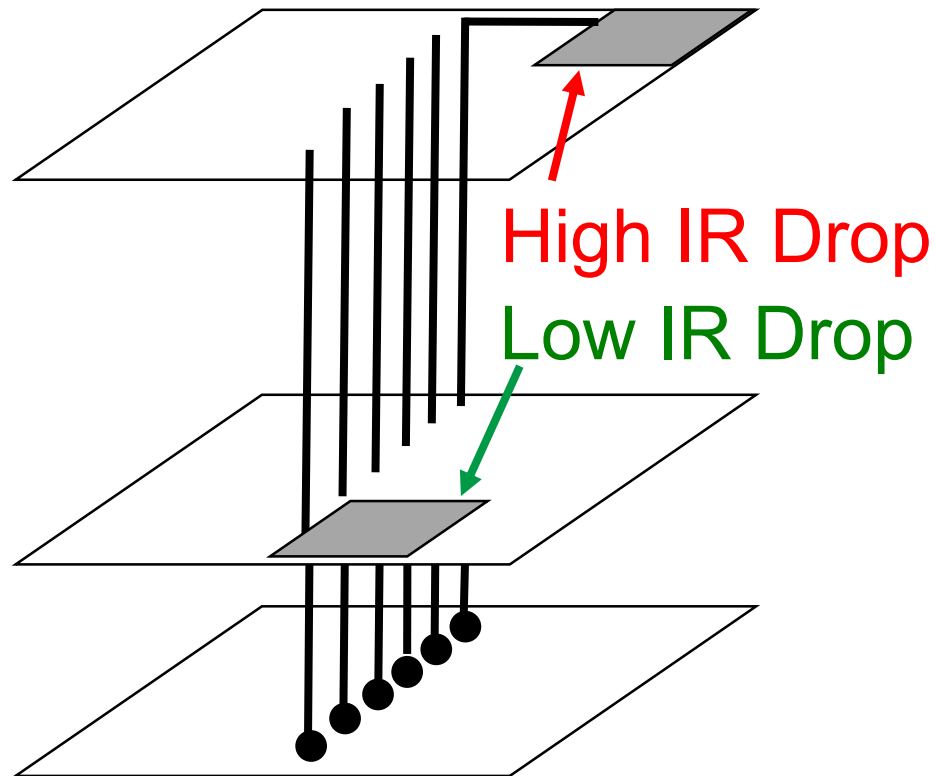
# KEY CONTRIBUTIONS

③ Memory controller: what, when, **where**



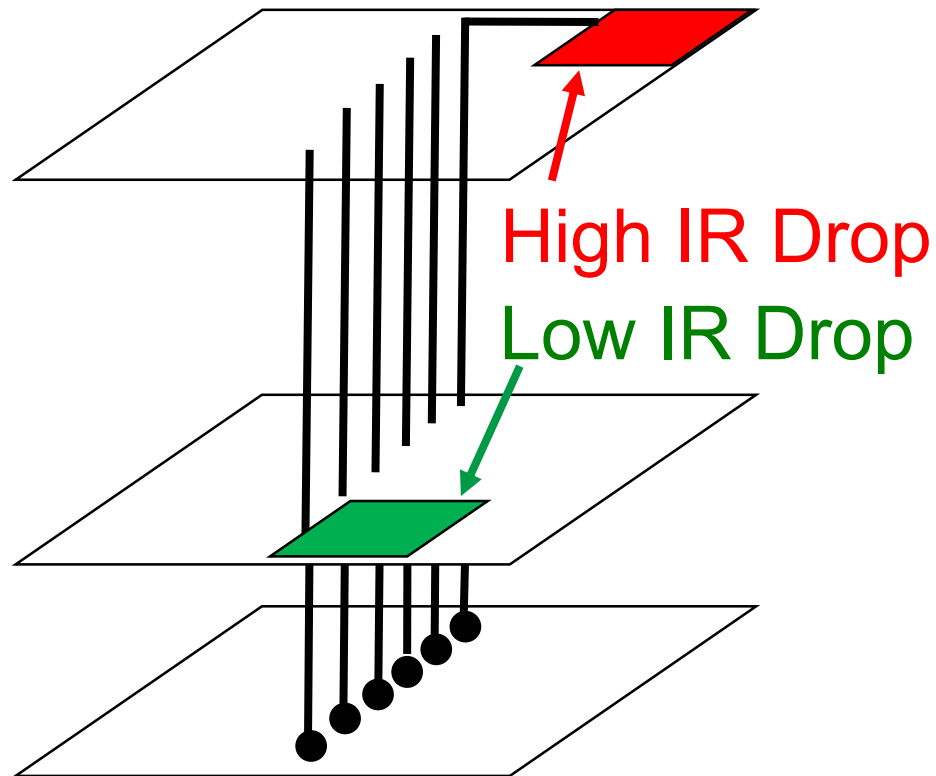
# KEY CONTRIBUTIONS

## ④ Handle starvation



# KEY CONTRIBUTIONS

- ⑤ Place pages in favored regions



**NOW PLAYING**

A PAPER ABOUT

**COST &  
VOLTAGE  
NOISE**

UTAH, SAMSUNG, ARM

**NOW PLAYING**

WHERE ARE THE  
#&\$@ PERFORMANCE  
NUMBERS?  
TUESDAY 9:30am,  
SESSION 3A ?!  
I'LL BE THERE !



**INTRIGUED TOM CONTE**



Augusto Vega, Alper Buyuktosunoglu, Heather Hanson, Pradip Bose, Srinivasan Ramani  
IBM T. J. Watson Research Center, IBM Systems & Technology Group

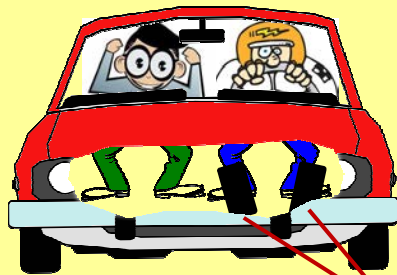
## Executive Summary

- Modern multi-core systems incorporate support for **dynamic power management** with **multiple actuators**
- Algorithms that control these actuators have evolved independently
  - Their independent operation can result in **suboptimal decisions**



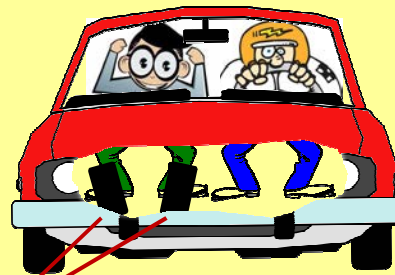
## What is more appropriate?

This?



Coordinated  
Control

Or this?



Decoupled  
Control

Pedals

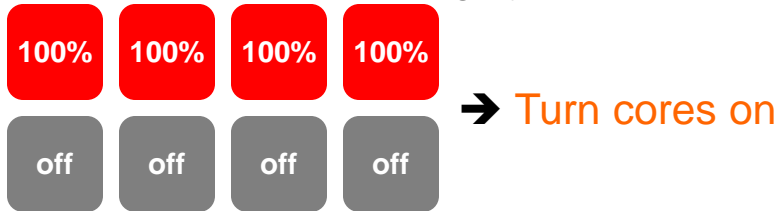
We argue in favor of a **coordinated** control of these actuators to avoid potential conflicts in dynamic power management

# Performance *And* Throughput Awareness

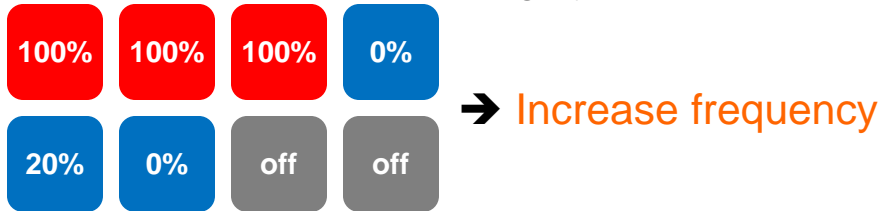
**OUR  
IDEA**

Operate power management *knobs* depending on if an application's current execution phase is **single-thread performance** or **throughput bound**

All turned-on cores are highly utilized



Some turned-on cores are highly utilized



All turned-on cores are low utilized or idle

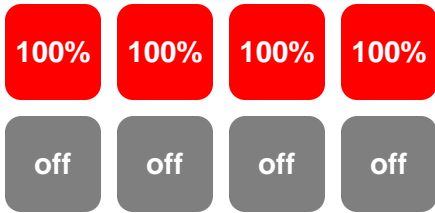


# Performance *And* Throughput Awareness

**OUR  
IDEA**

Operate power management *knobs* depending on if an application's current execution phase is **single-thread performance** or **throughput bound**

All turned-on cores are highly utilized



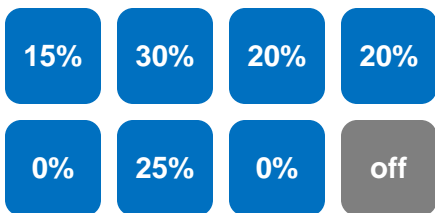
→ Turn cores on

Some turned-on cores are highly utilized



→ Increase frequency

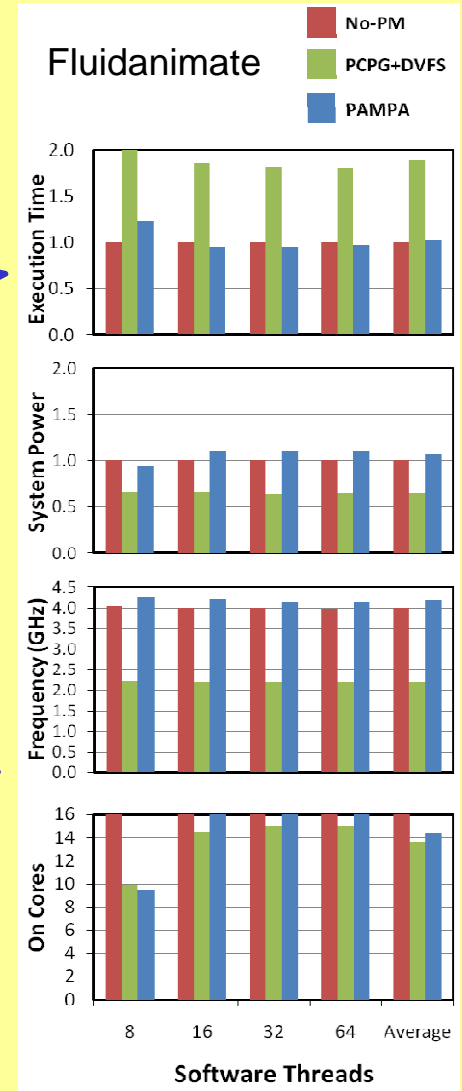
All turned-on cores are low utilized or idle



→ Decrease frequency, turn cores off

PAMPA preserves performance

PAMPA properly actuates DVFS and PCPG



# Wavelength Stealing: An Opportunistic Approach to Channel Sharing in Multi-chip Photonic Interconnects

**Arslan Zulfiqar** (UW-Madison)

**Pranay Koka** (Oracle Labs)

**Herb Schwetman** (Oracle Labs)

**Mikko Lipasti** (UW-Madison)

**Xuezhe Zheng** (Oracle Labs)

**Ashok Krishnamoorthy** (Oracle Labs)

# Problem: What is the “best” topology design for photonic substrates?



Point-to-Point  
or  
Channel sharing

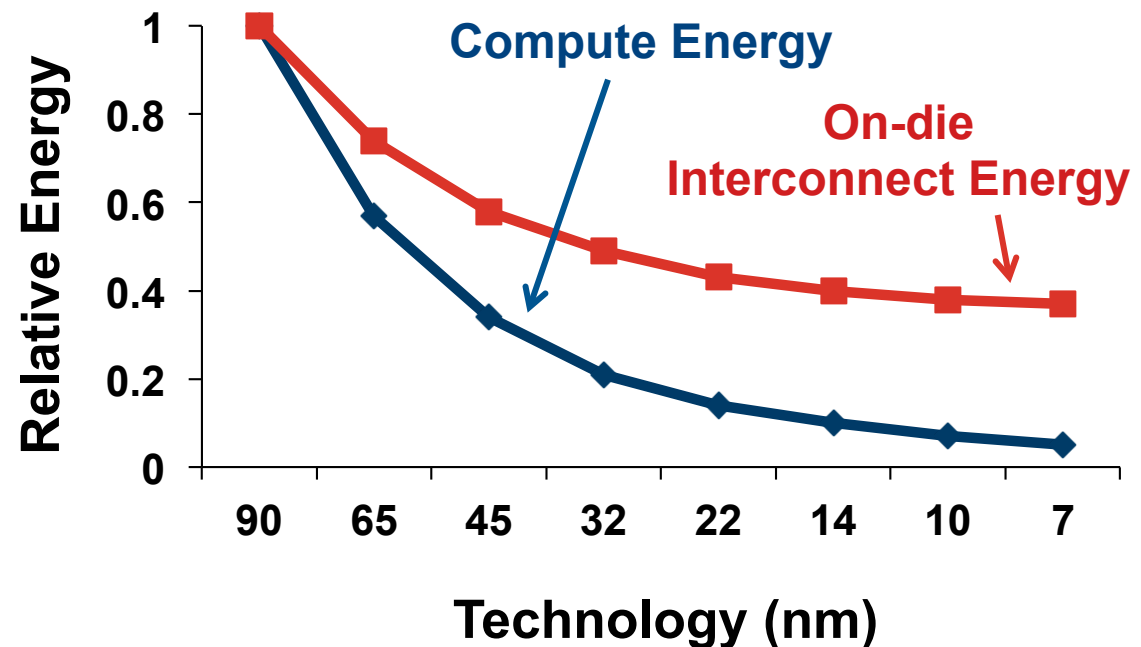
# Our Contributions

- **Analytical model to quantify the limits and gains of channel sharing**
  - # of senders per channel  $\leq 3$
  - Performance speedup  $\leq 1.70x$
  
- **“Wavelength Stealing” architecture**
  - Arbitration-free accesses
  - Strong fairness guarantees
  - Up to **28%** EDP improvement over baseline

# The High Cost of Data Movement

A significant and growing fraction of on-die energy is spent in data movement.

Long, capacitive interconnects consume most of the LLC access energy.

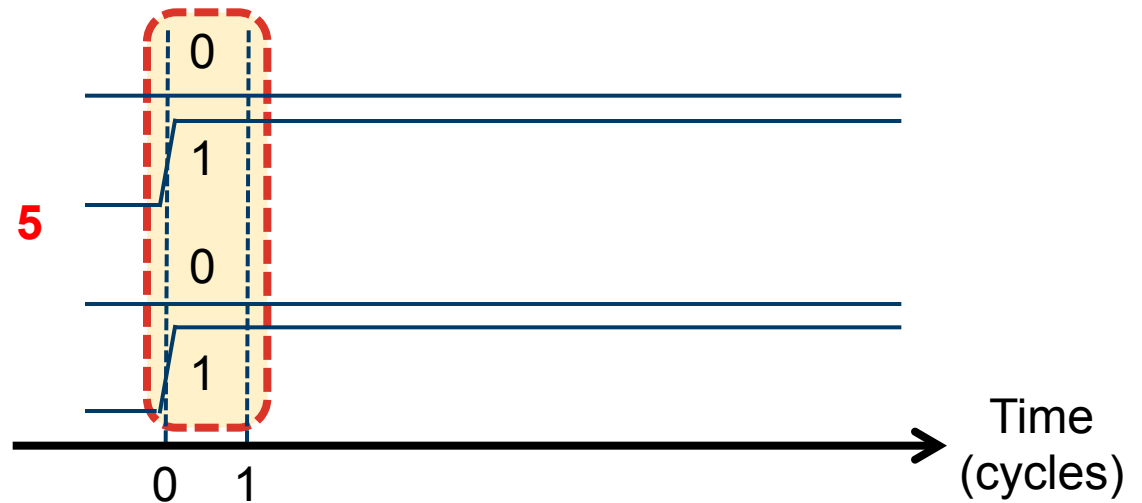


Shekhar Borkar, *Journal of Lightwave Technology*, 2013

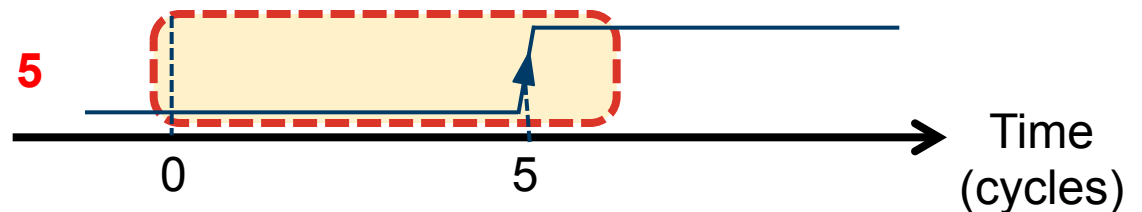
# Proposal: Time Based Data Transfer

**Key idea:**  
represent  
information by  
the number of  
clock cycles  
between two  
consecutive  
pulses to  
reduce the  
interconnect  
activity factor.

## Parallel Communication



## DESC Communication

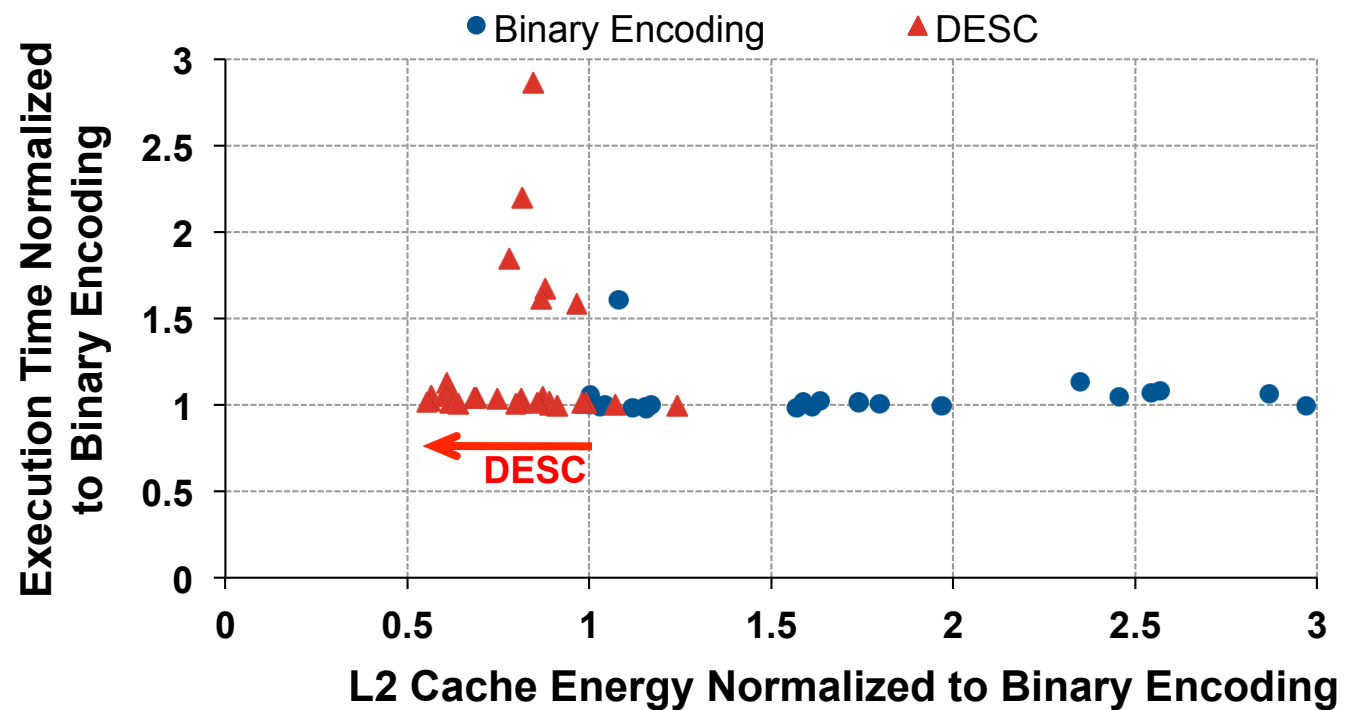




# Summary of Results

DESC reduces LLC energy by 1.8x at the cost of a 2% increase in execution time.

DESC expands the Pareto frontier in energy-efficient cache design.



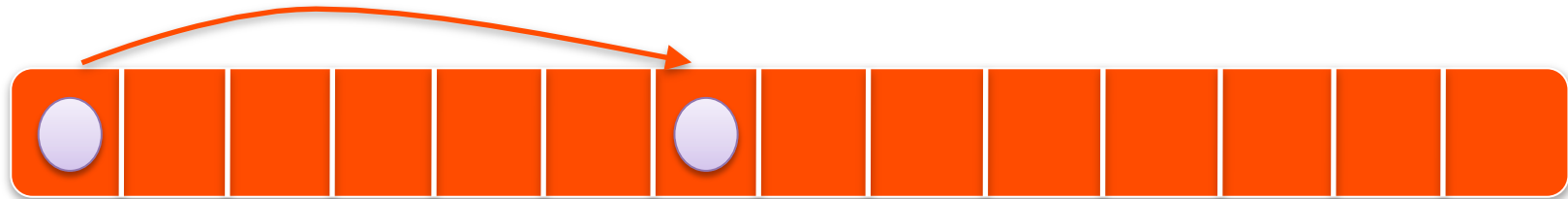
# Linearizing Irregular Memory Accesses for Improved Correlated Prefetching

Akanksha Jain, Calvin Lin  
University of Texas at Austin

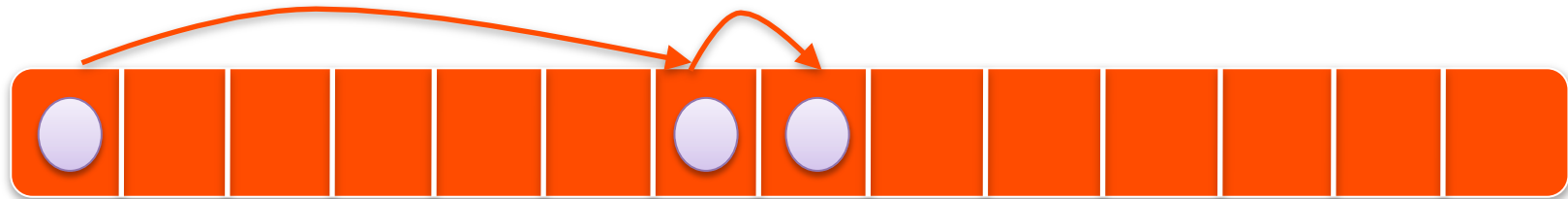
# The Problem : Irregular Prefetching



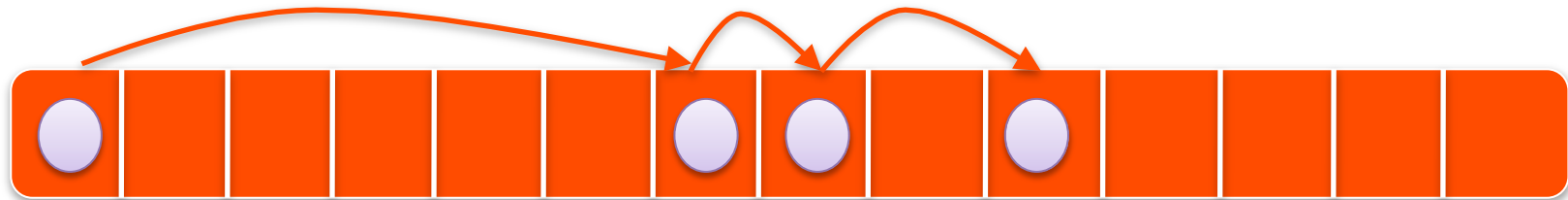
# The Problem : Irregular Prefetching



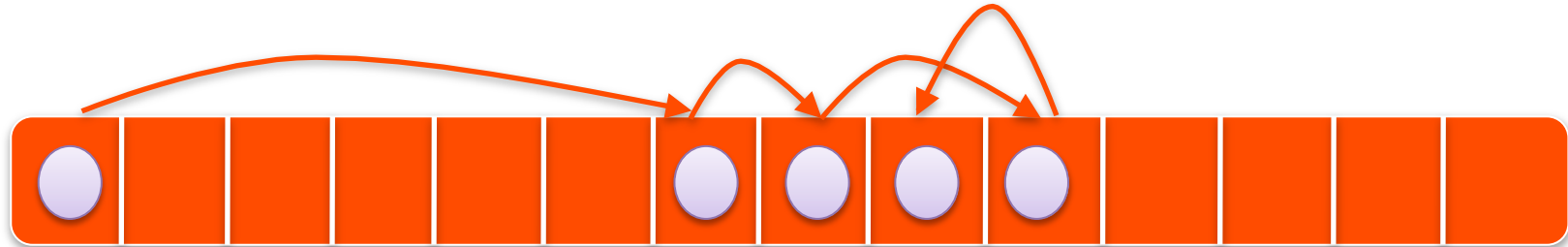
# The Problem : Irregular Prefetching



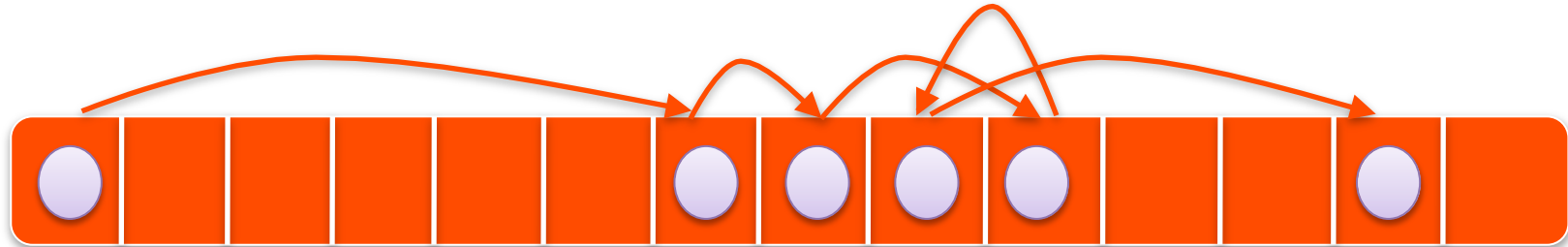
# The Problem : Irregular Prefetching



# The Problem : Irregular Prefetching



# The Problem : Irregular Prefetching

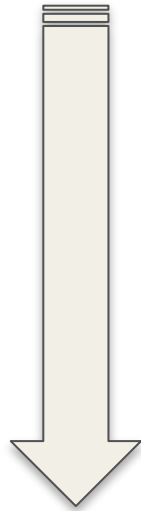




# Our Solution : Transformation



Irregular  
Prefetching



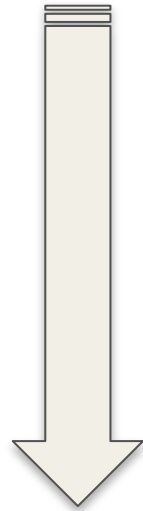
Regular  
Prefetching



# Our Solution : Transformation



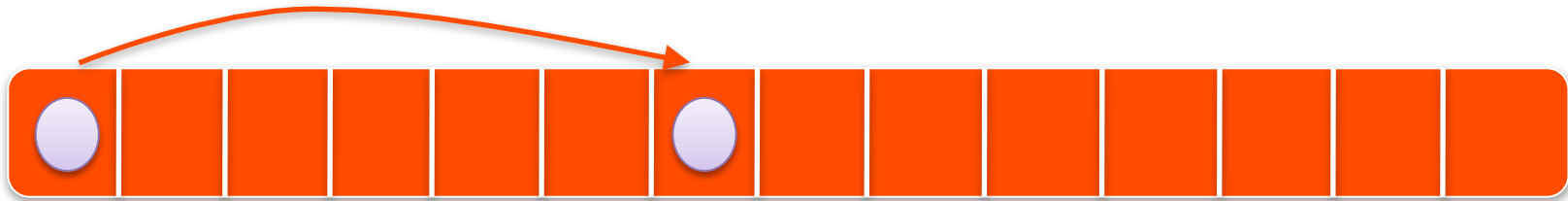
Irregular  
Prefetching



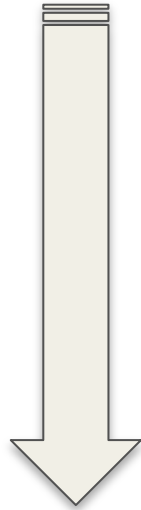
Regular  
Prefetching



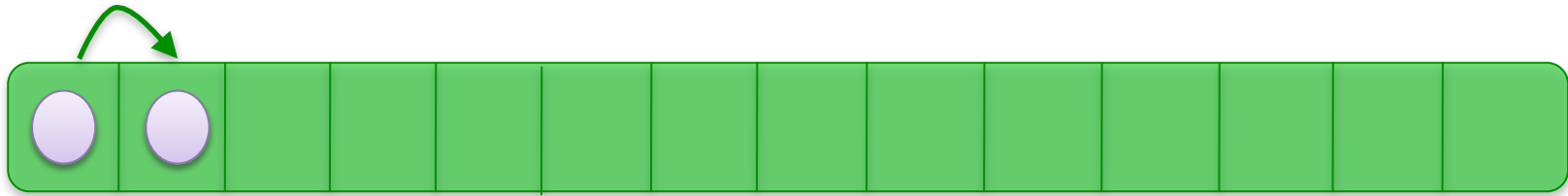
# Our Solution : Transformation



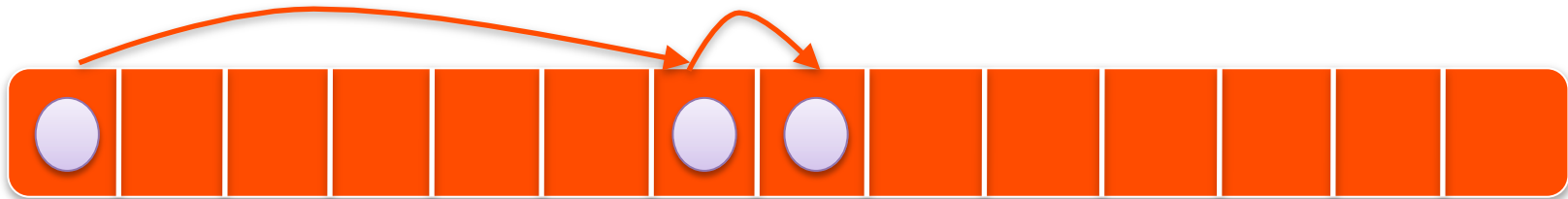
Irregular  
Prefetching



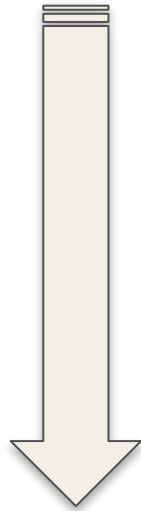
Regular  
Prefetching



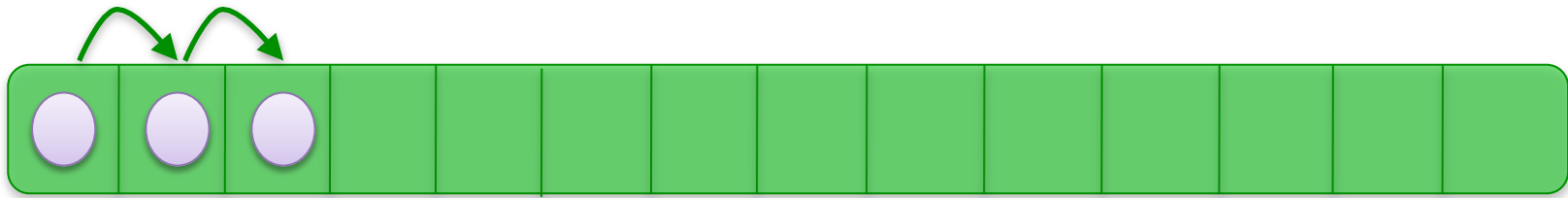
# Our Solution : Transformation



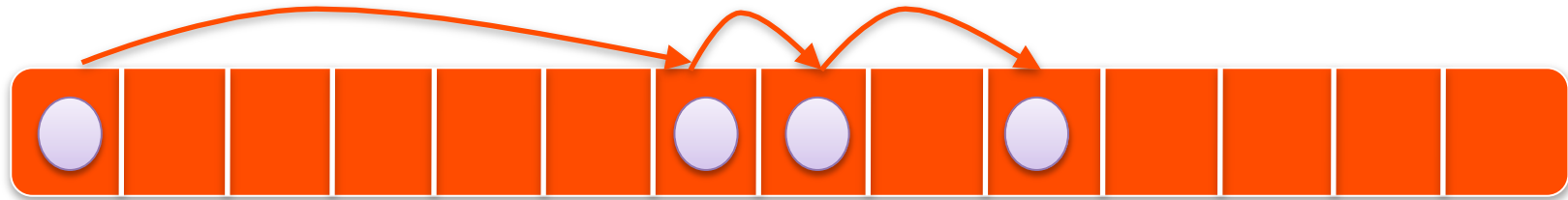
Irregular  
Prefetching



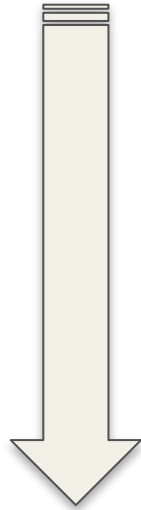
Regular  
Prefetching



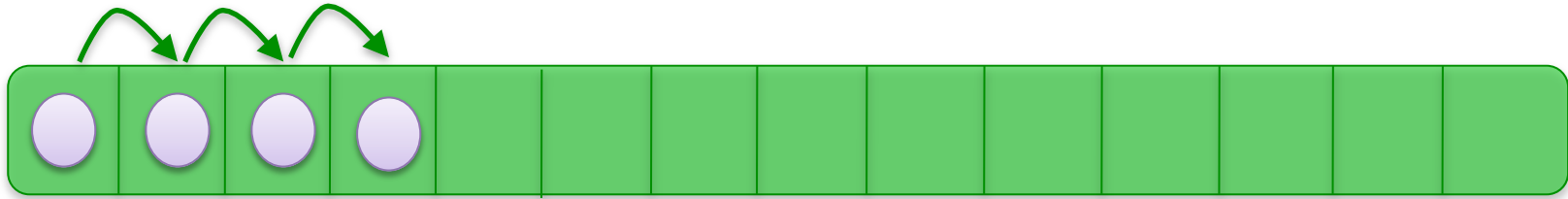
# Our Solution : Transformation



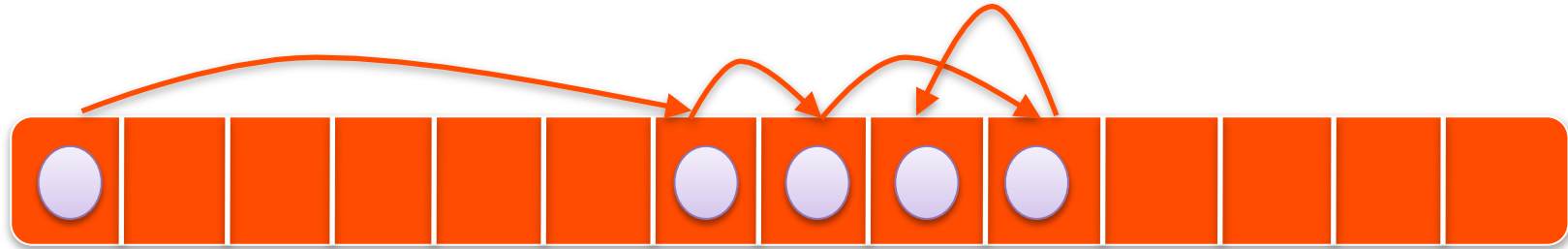
Irregular  
Prefetching



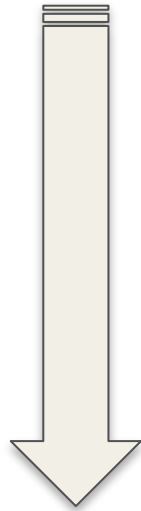
Regular  
Prefetching



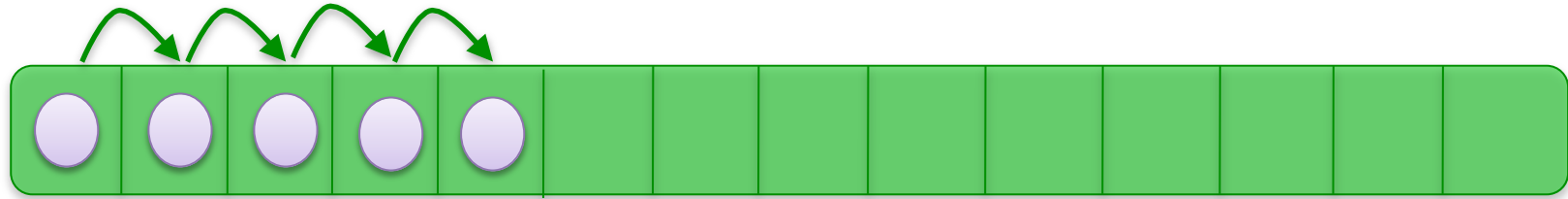
# Our Solution : Transformation



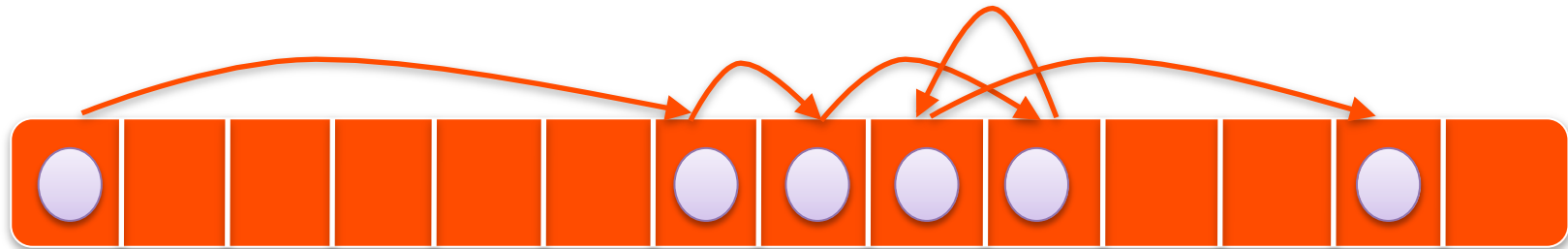
Irregular  
Prefetching



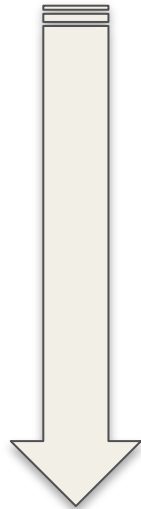
Regular  
Prefetching



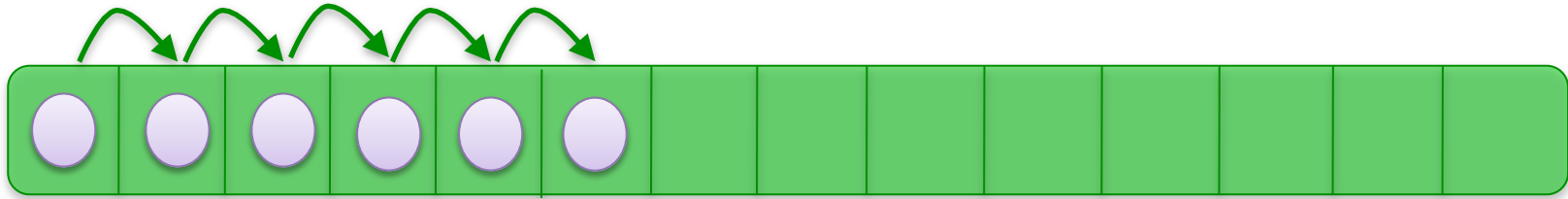
# Our Solution : Transformation



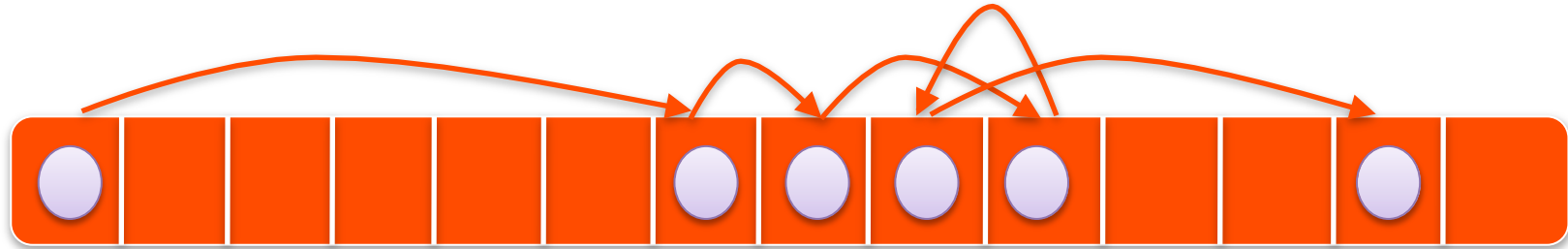
Irregular  
Prefetching



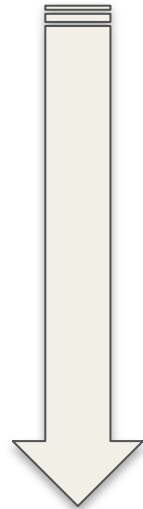
Regular  
Prefetching



# Our Solution : Transformation



Irregular  
Prefetching



Regular  
Prefetching



	<b>Previous Best</b>	<b>Our Solution</b>
Speedup	8.3%	23.1%
Accuracy	58.6%	93.7%



# RAS-Directed Instruction Prefetching (RDIP)

**Aasheesh Kolli\***

**Ali Saidi<sup>†</sup>**

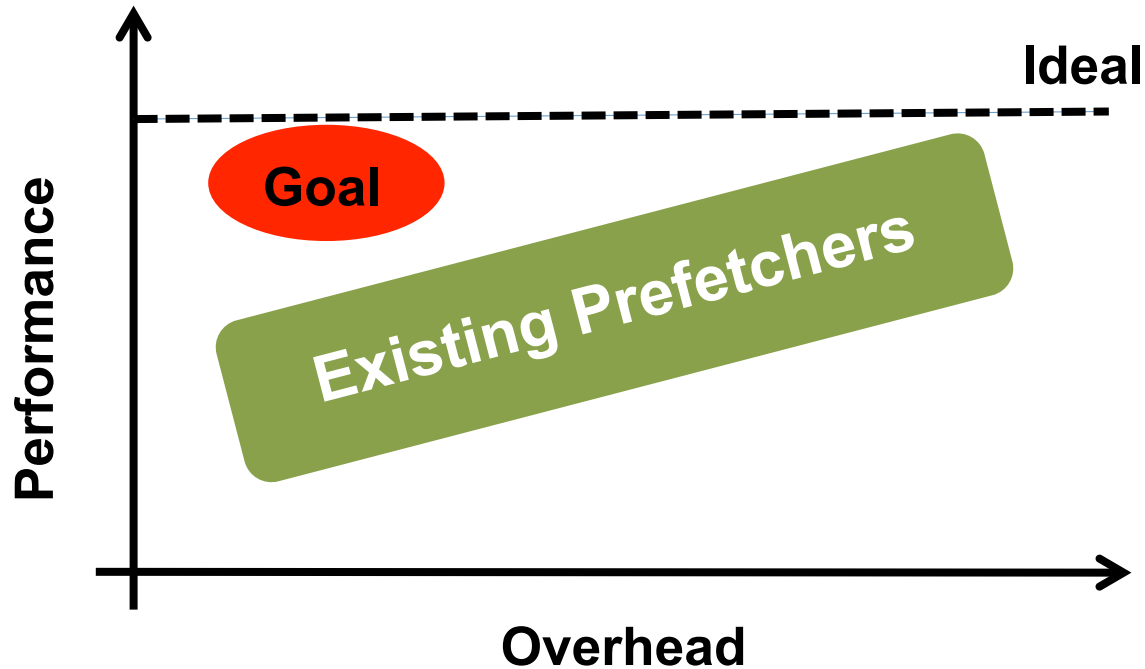
**Thomas F. Wenisch\***

\* University of Michigan

<sup>†</sup> ARM

# Why another instruction prefetcher?

- Poor I\$ behavior affects modern workloads
- Cache size constraints → Prefetching



**Our Goal: Low overhead, high accuracy instruction prefetcher**

# Contributions

- I\$ misses correlate to program context
- Program contexts are repetitive → predictable
- RAS succinctly captures program context

# Contributions

- I\$ misses correlate to program context
- Program contexts are repetitive → predictable
- RAS succinctly captures program context

## **RAS-Directed Instruction Prefetching (RDIP)**

# Contributions

- I\$ misses correlate to program context
- Program contexts are repetitive → predictable
- RAS succinctly captures program context

## **RAS-Directed Instruction Prefetching (RDIP)**

**Performance improvement of 11.5%**  
**Overhead of 64kB (3X ↓)**

# SHIFT

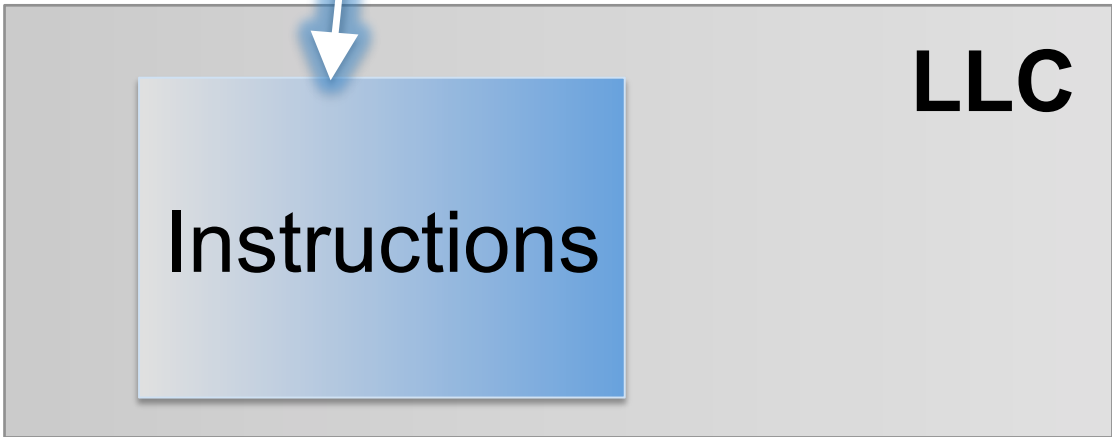
## Shared History Instruction Fetch for Lean-Core Server Processors

Cansu Kaynak, Boris Grot, Babak Falsafi





Core



**Instruction fetch stalls:**  
Major performance bottleneck  
Up to 60% of server exec. time

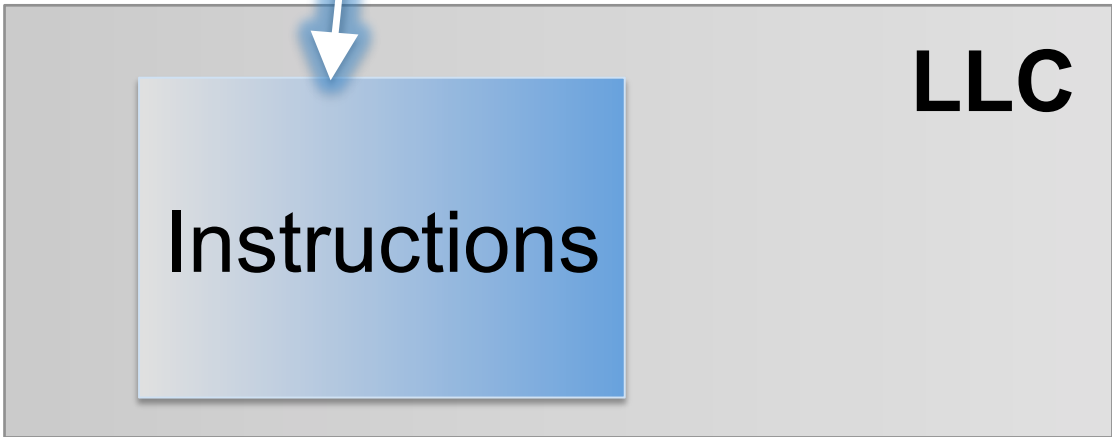


Core

Temporal Stream



I-History





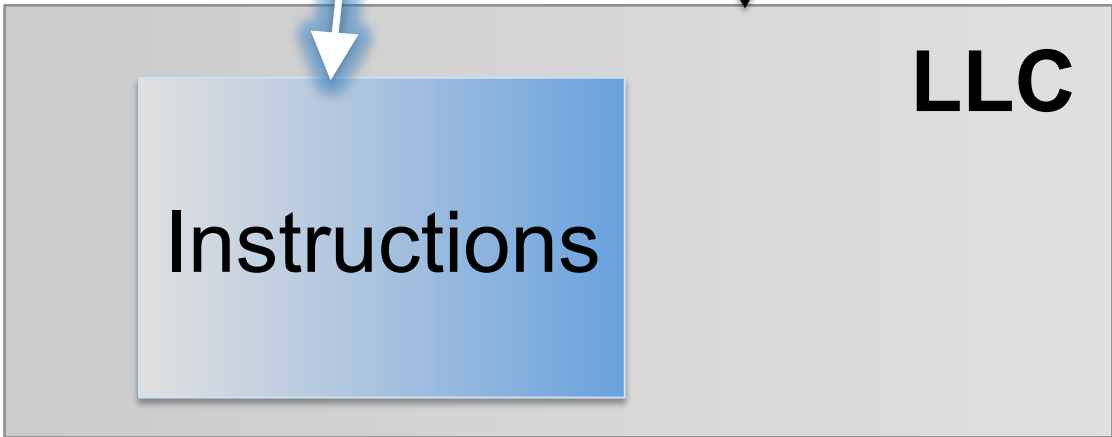


Core



I-History

Prefetch

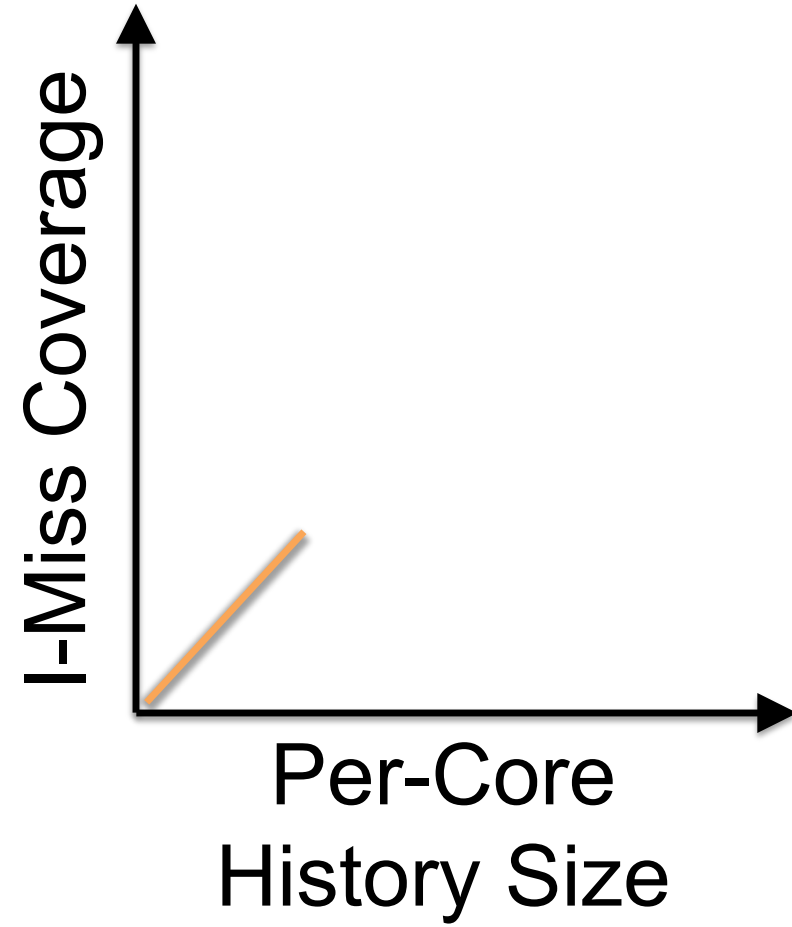
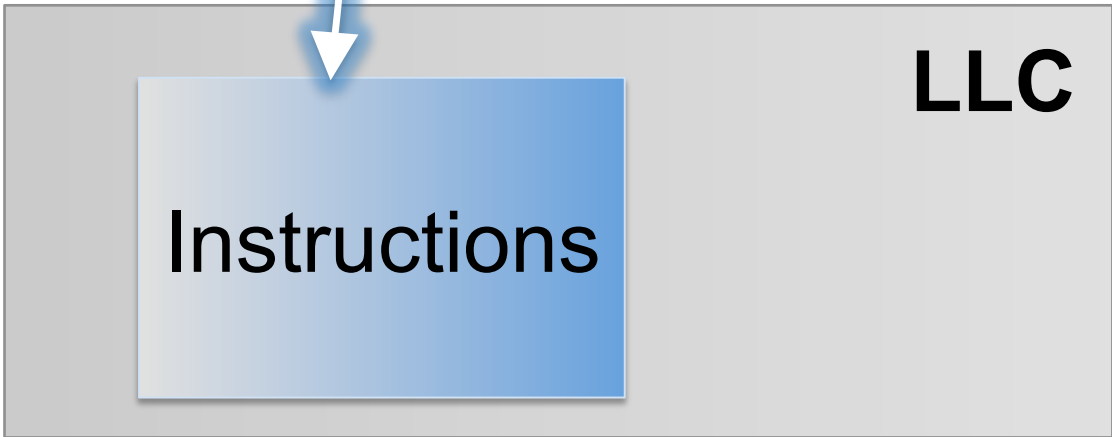




Core



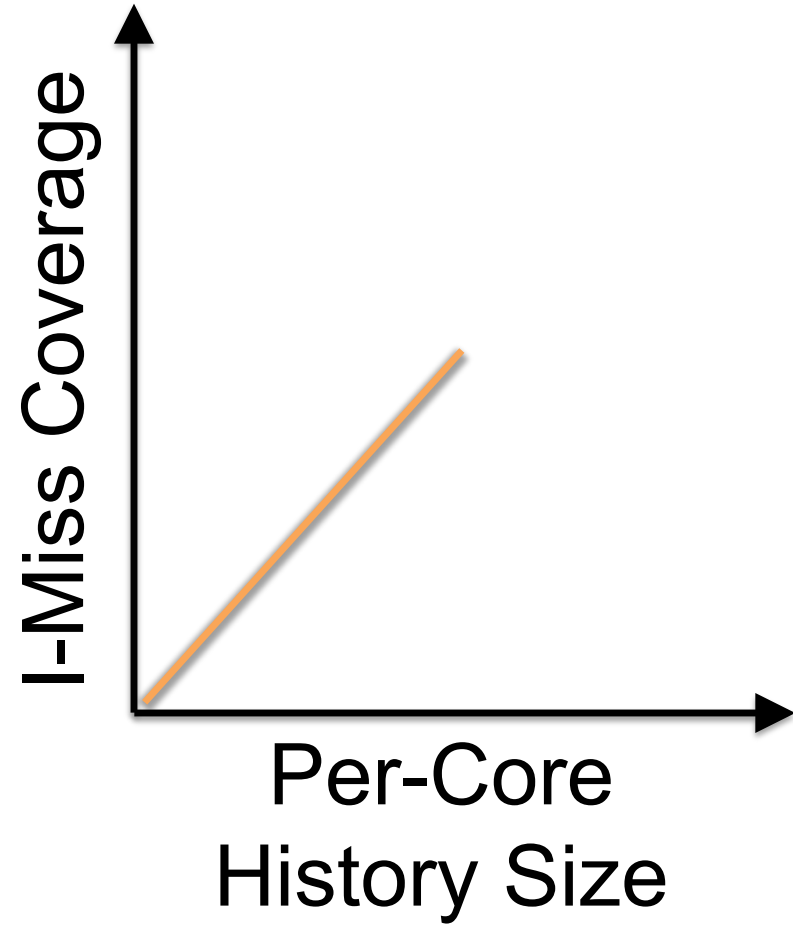
I-History

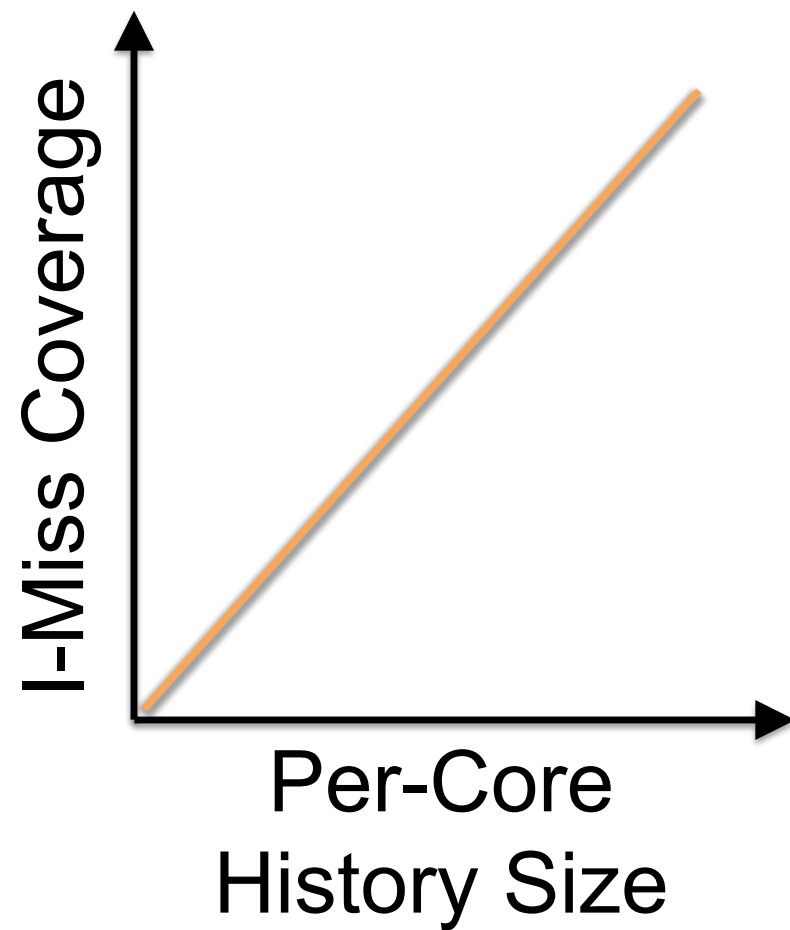
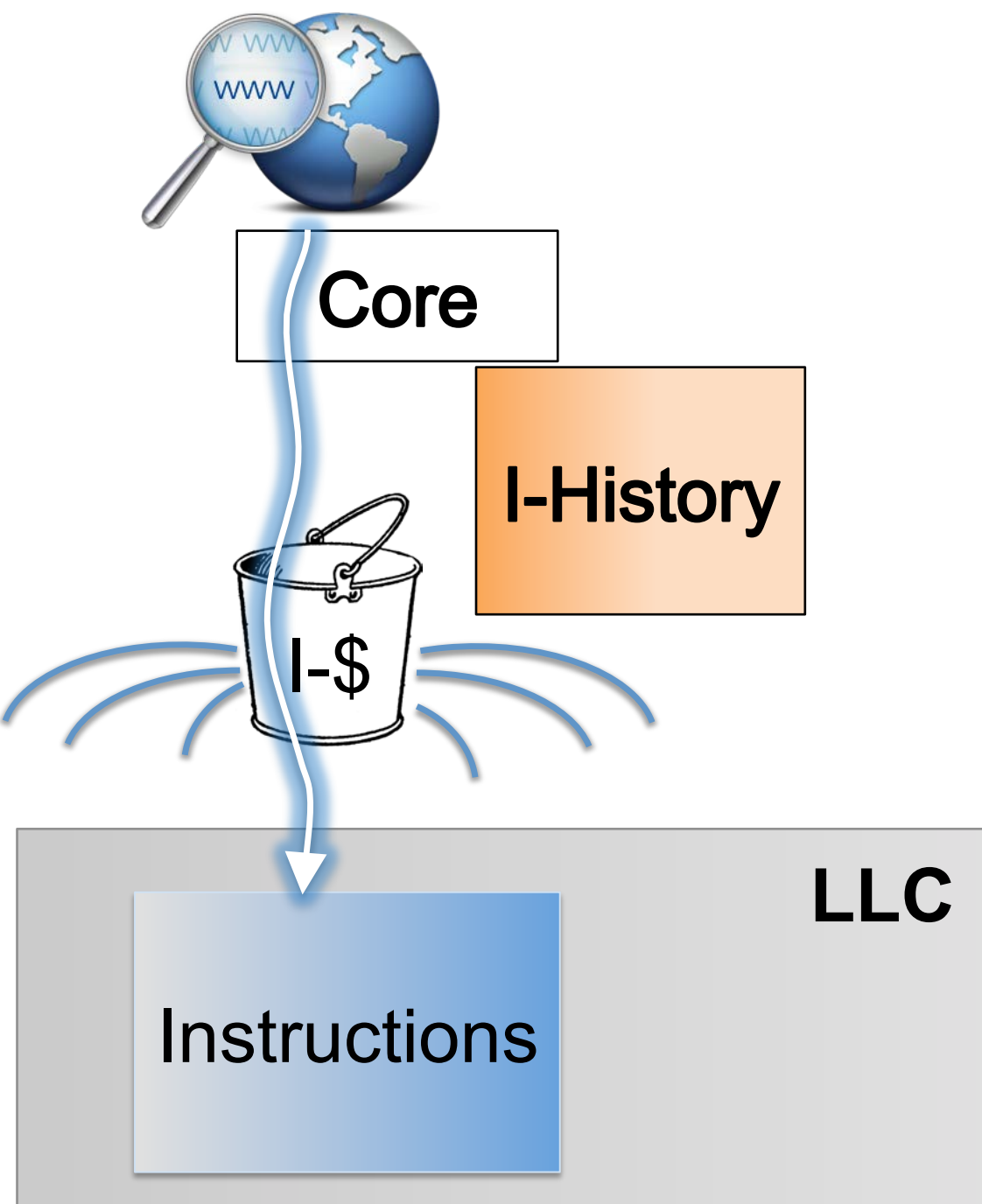


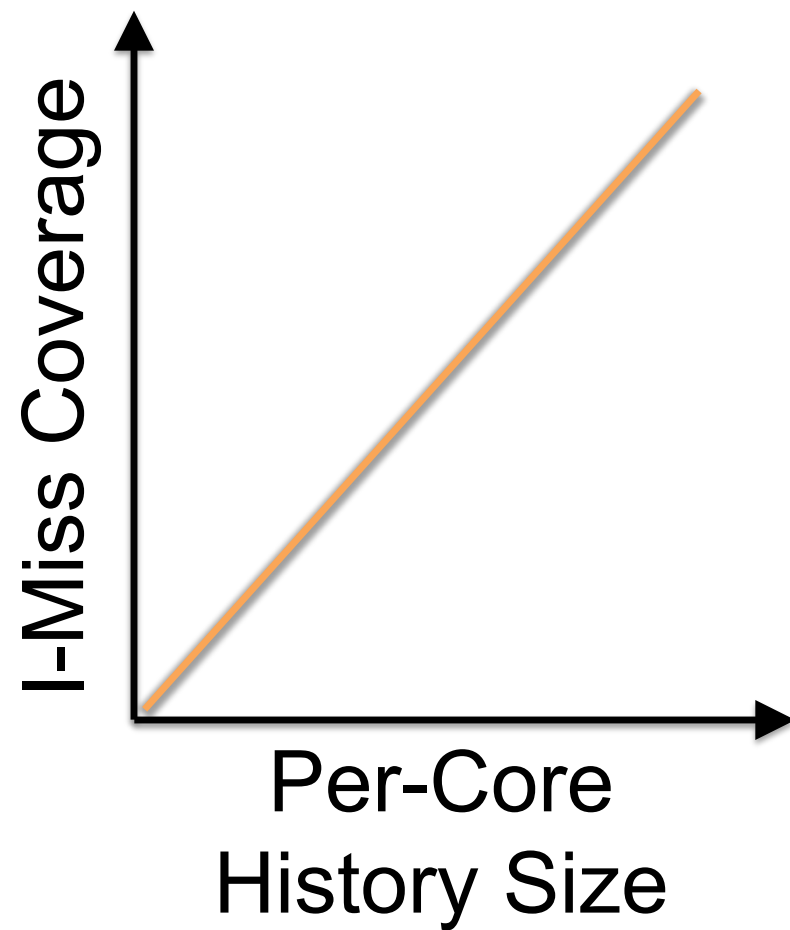
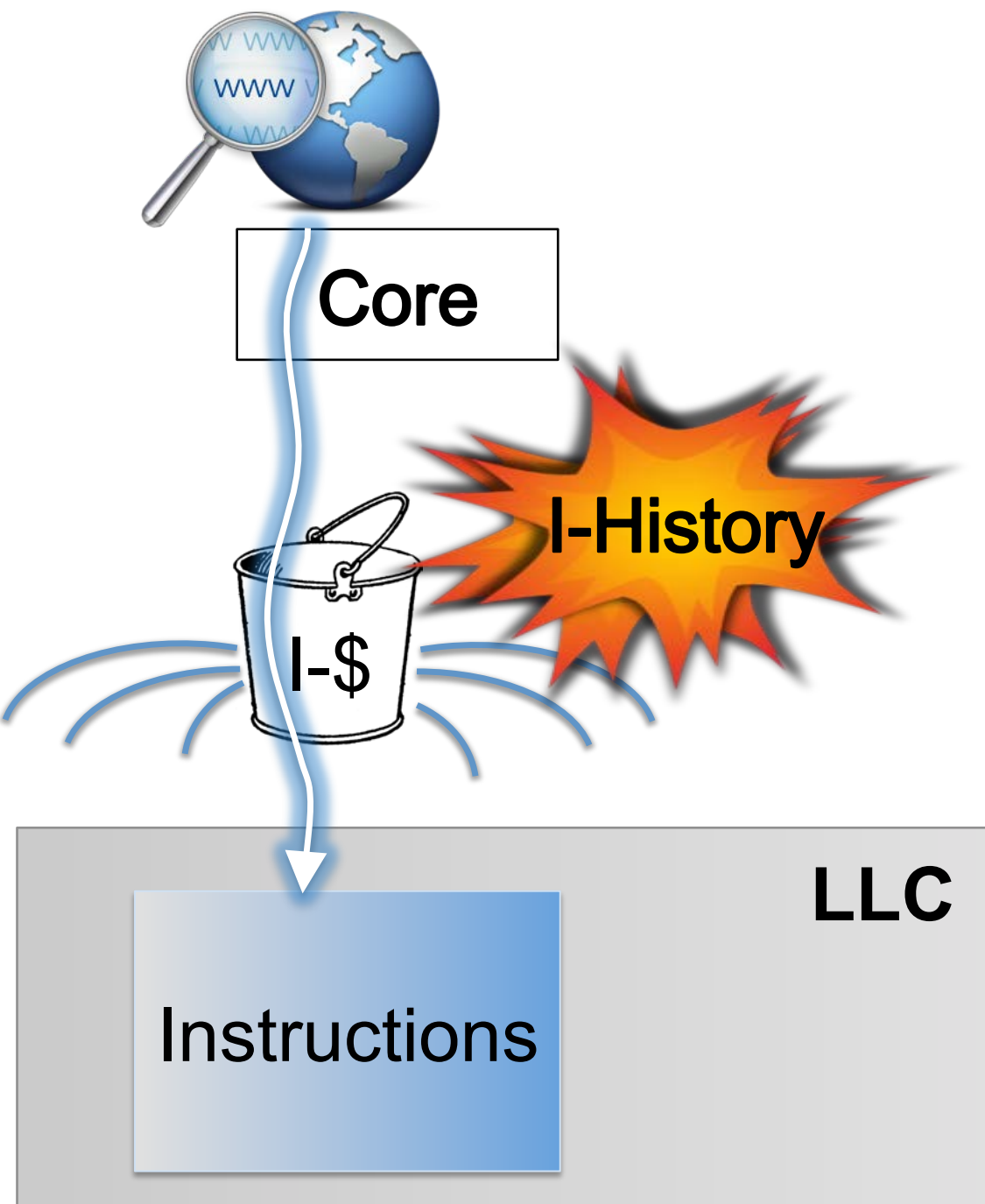


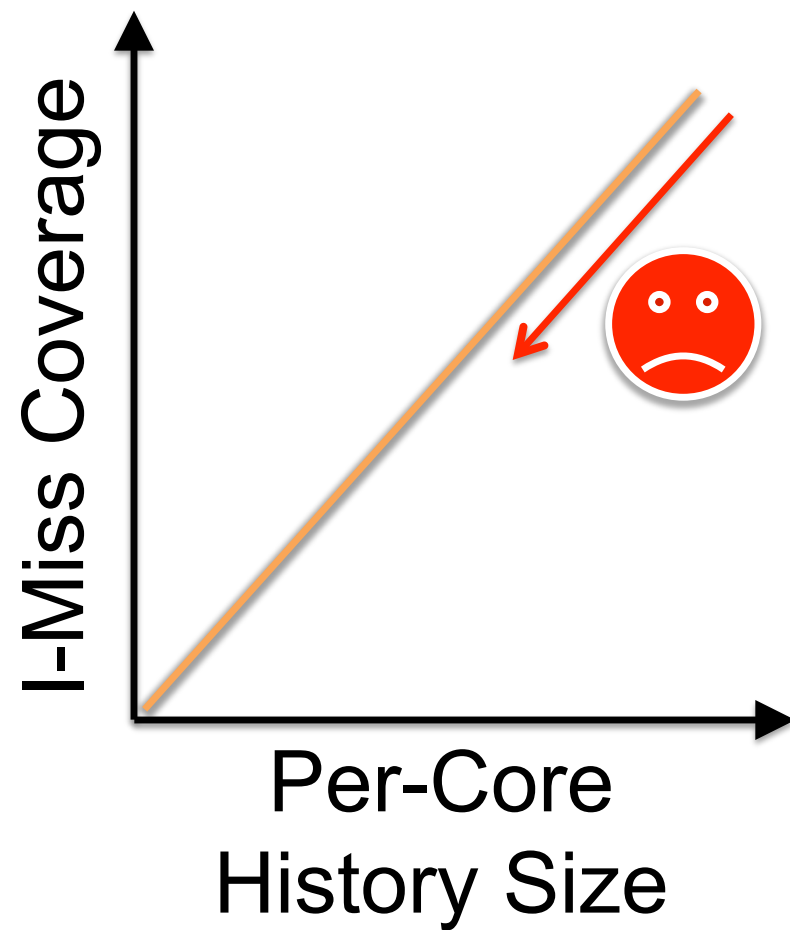
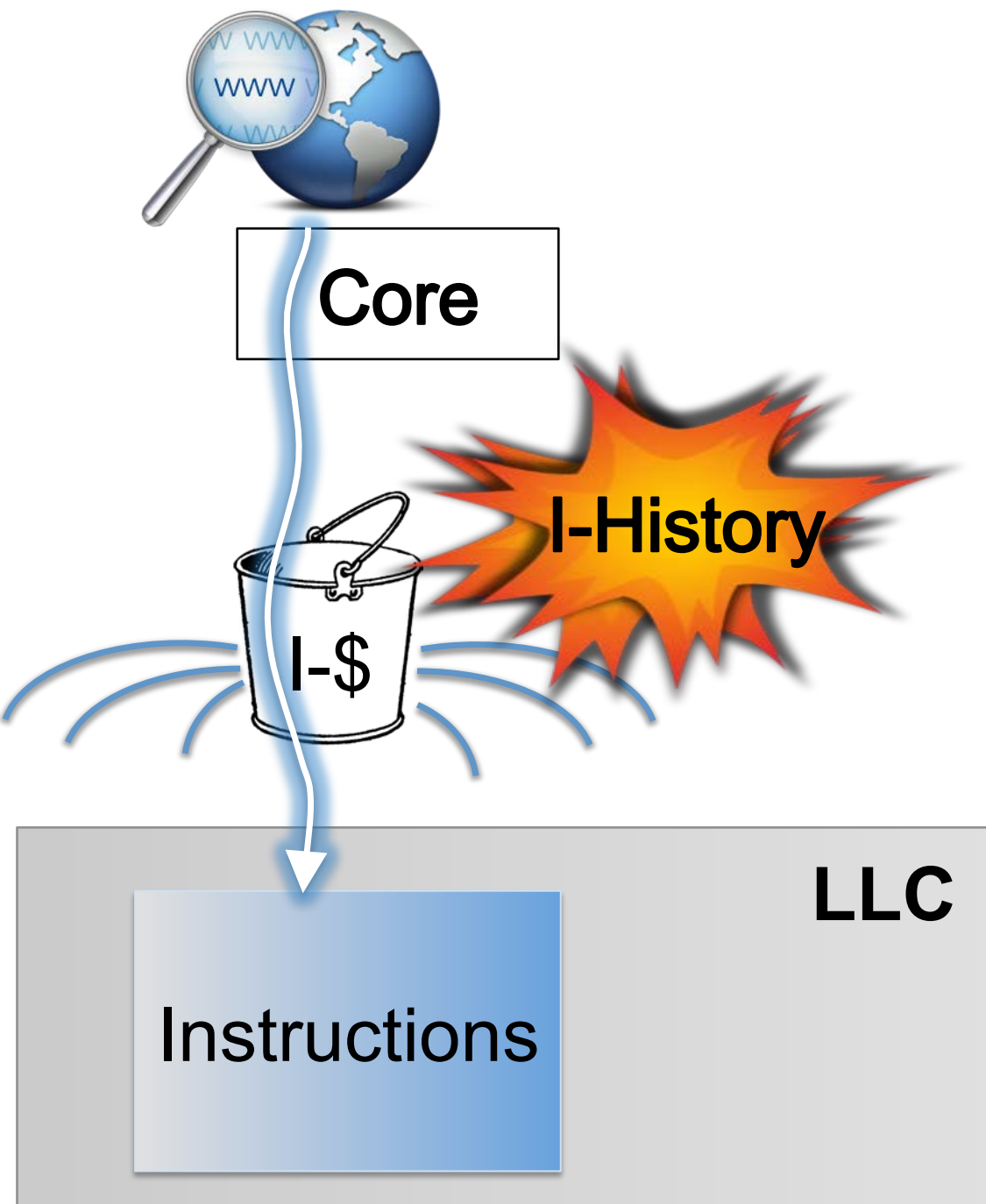
Core

I-History



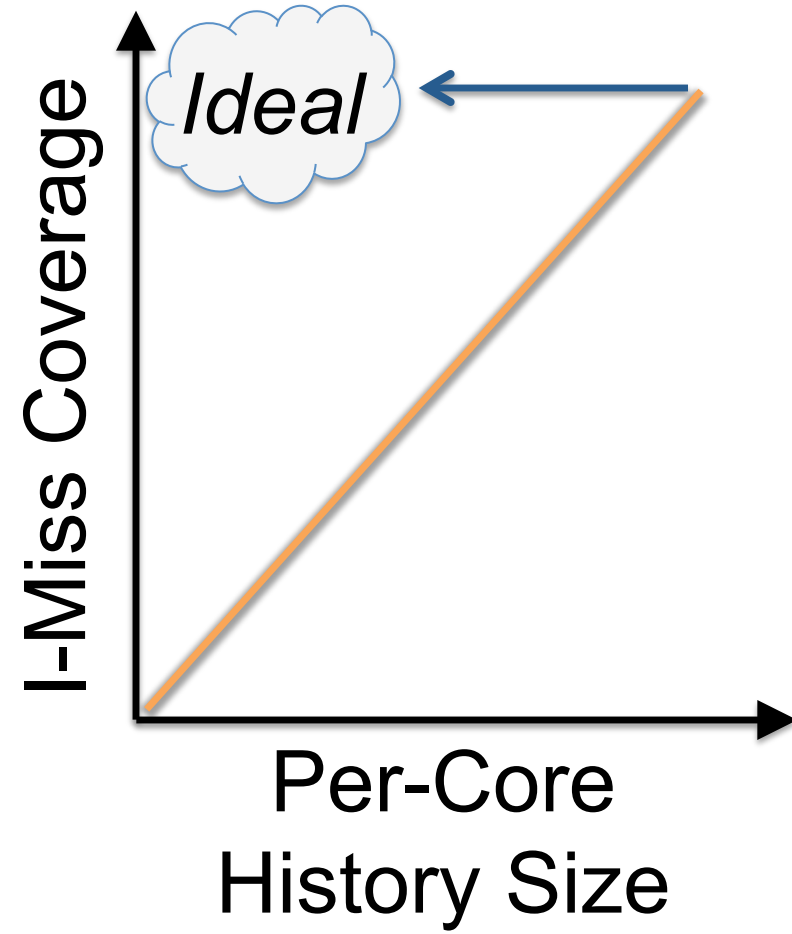




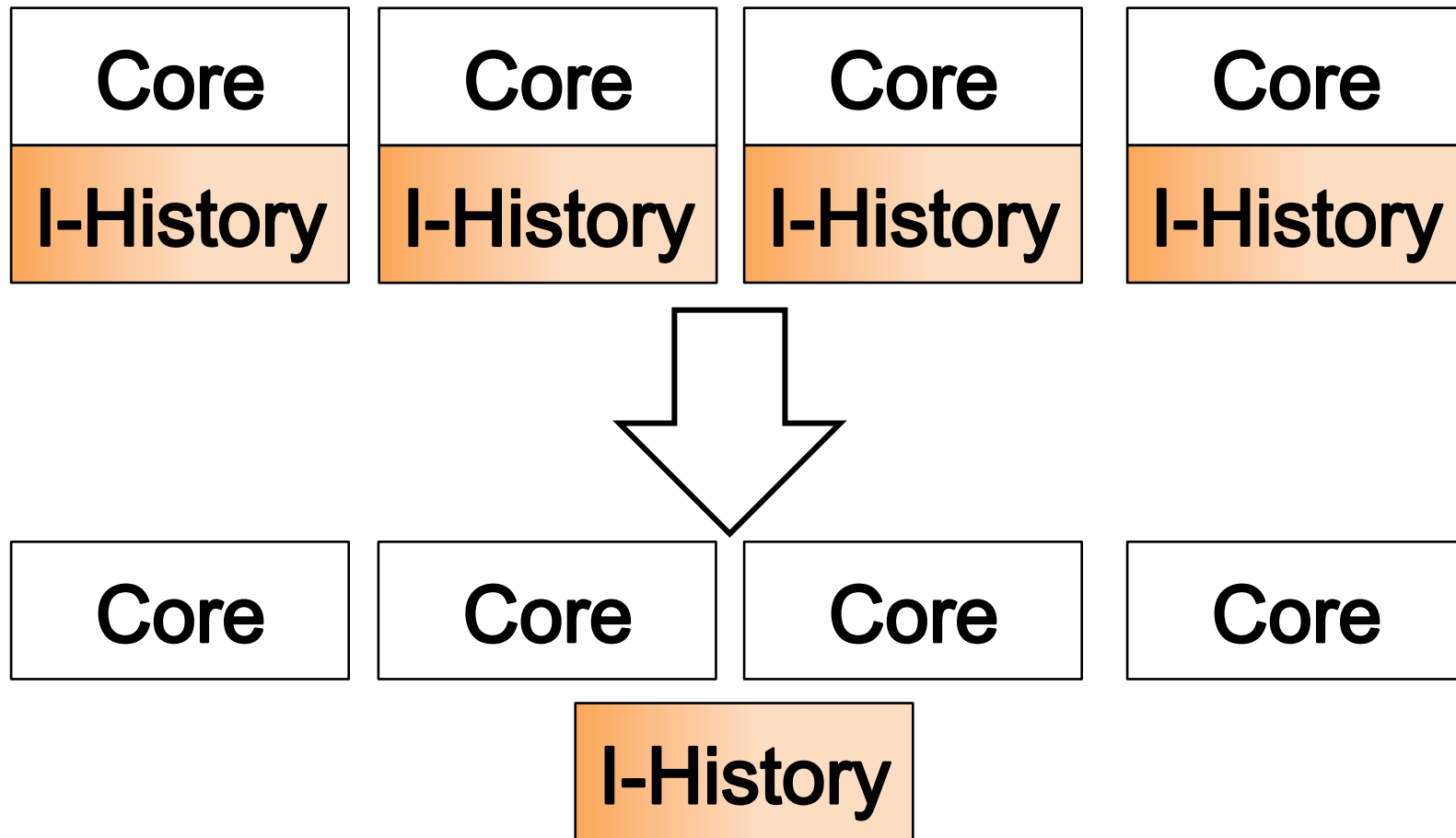




Core



# Shared History Instruction Fetch



- ✓ Preserves performance of per-core history
- ✓ At 1/14<sup>th</sup> of history size



# Insertion and Promotion for Tree-Based PseudoLRU Last-Level Caches

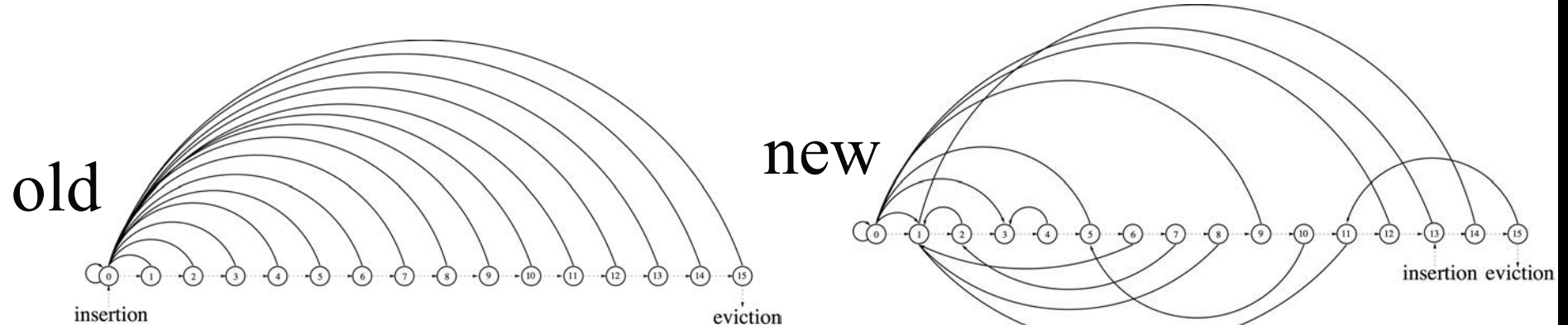
---

- ◆ Daniel A. Jiménez, Texas A&M University
- ◆ LRU keeps blocks in a recency stack
  - ◆  $n$ -way cache, 0 is MRU,  $n-1$  is LRU
- ◆ When a block is inserted or promoted (used) it goes to the MRU position
  - ◆ Not always the best choice
- ◆ Instead, let's use the blocks' former position to indicate its new position

# Searching a Large Design Space

---

- ◆ We want to develop a new transition graph



- ◆ For 16-way, there are  $> 300$  trillion possibilities
- ◆ So we use a genetic algorithm to search them
  - ◆ Fitness function is estimate of speedup

## PseudoLRU instead of LRU

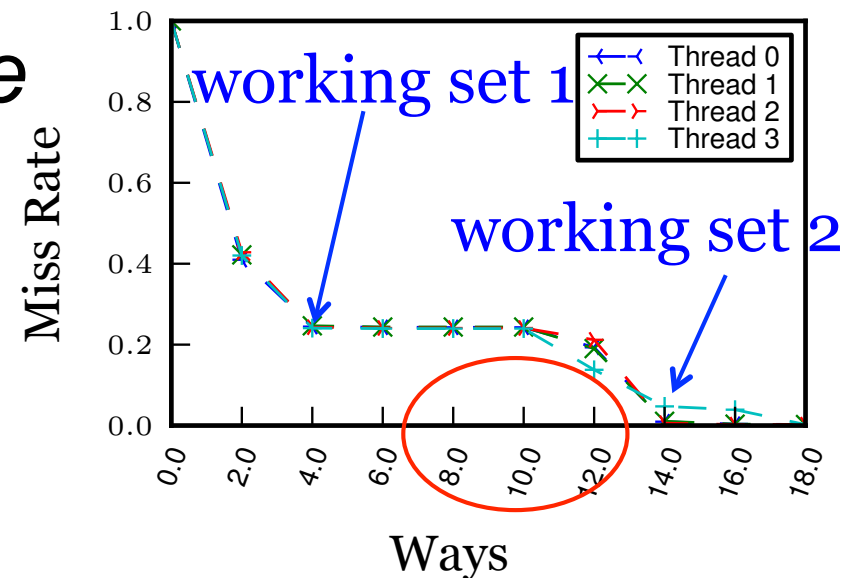
---

- ◆ This idea works just as well for tree-based PseudoLRU
- ◆ Use set-dueling to dynamically choose between policies
- ◆ Replacement policy consumes  $< 1$  bit per block
- ◆ Performance comparable to state-of-the-art
  - ◆ 5.6% speedup over LRU on SPEC CPU 2006
  - ◆ 15.6% on a memory-intensive subset

# Imbalanced Cache Partitioning for Balanced Data-Parallel Programs

Abhisek Pan & Vijay S. Pai, Purdue University

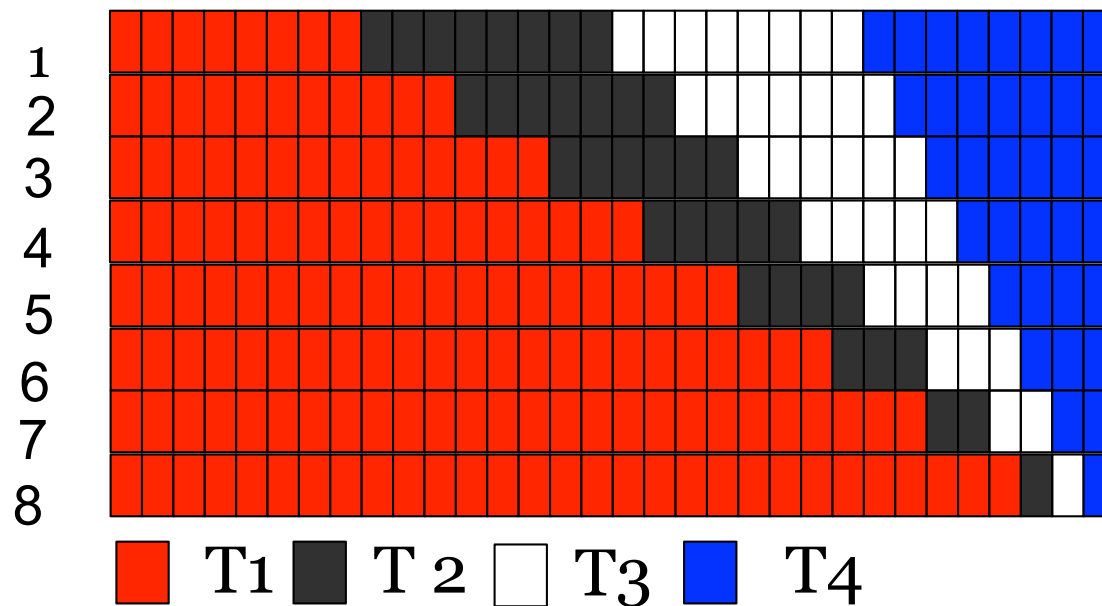
- *Balanced* data parallel programs need *imbalance* in allocation
- High imbalance helps both rewarded and penalized threads
- Prioritizing each thread in turn at a time ensures balanced progress



**Inefficient  
Allocation!**

# Two-Stage Partitioning Method

- **Evaluation Stage**



- Divide cache sets into segments with different levels of imbalance
- Choose segment with lowest # of misses

- **Stable Stage**

- Use chosen partition for the entire cache
- Choose preferred thread in round-robin manner

# Evaluation

- Partitioning beneficial only when per-thread working set between the default allocation and the cache capacity
- Improves upon the state-of-the-art runtime partitioning method in most such cases
  - 6% drop in execution time, 17 % drop in misses for 8 MB cache with 4 cores
- Limited overheads in space (way-partitioning, phase detection) and time (evaluation stage)

# The Reuse Cache

## Downsizing the Shared Last-Level Cache

**Jorge Albericio**<sup>1</sup>, Pablo Ibáñez<sup>2</sup>, Víctor Viñals<sup>2</sup>,  
and José M. Llabería<sup>3</sup>

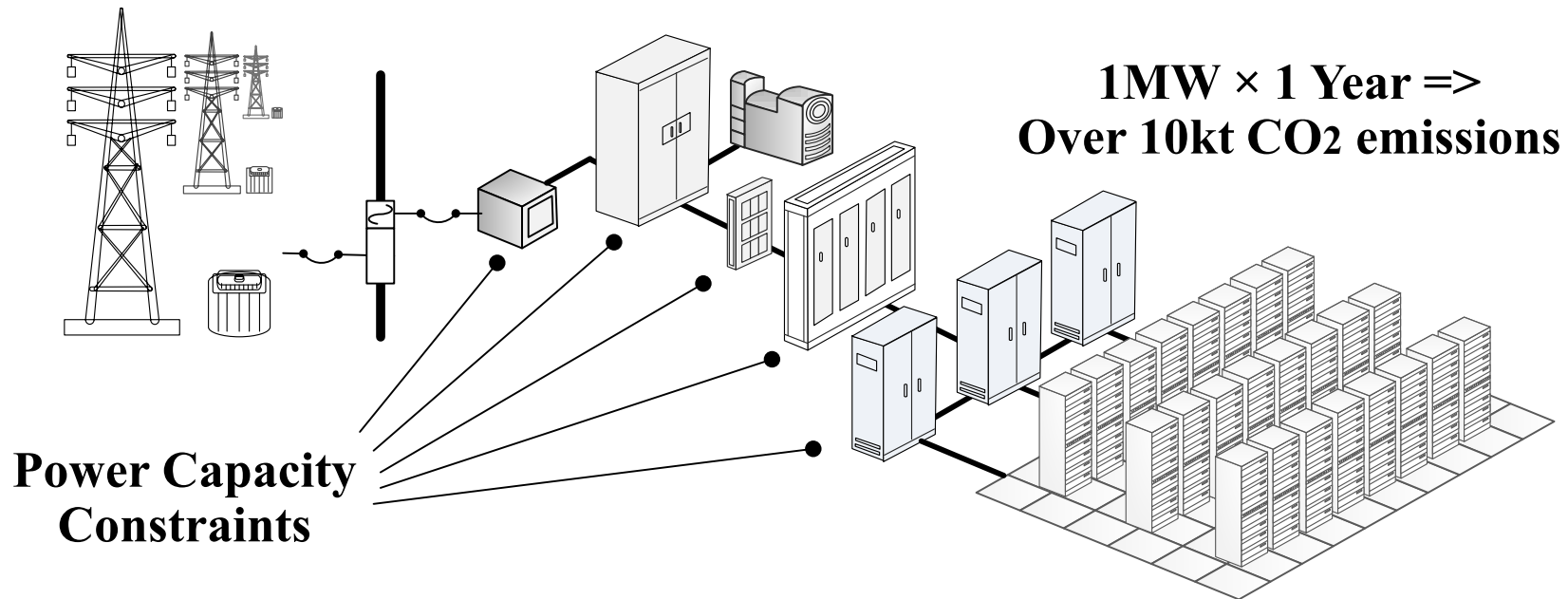
**Tuesday, Session 4B (10:30-12:00)**



# Enabling Datacenter Servers to Scale Out Economically and Sustainably

Chao Li, Yang Hu, Ruijin Zhou, Ming Liu, Longjun Liu, Jingling Yuan, Tao Li  
University of Florida

As data sets become **big**, servers must **scale out**

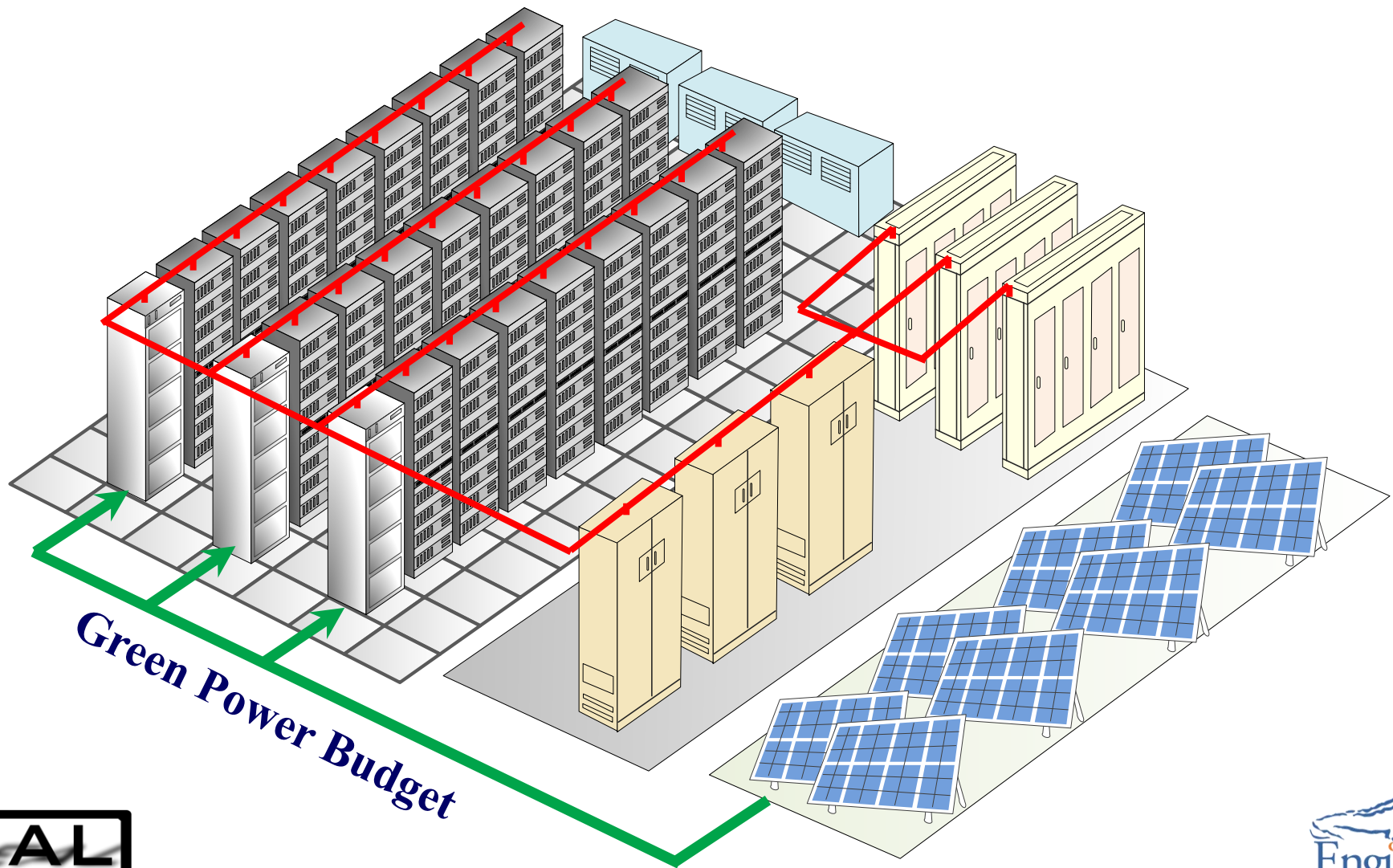


Unfortunately, modern datacenter servers are both **power-constrained** and **carbon-constrained**



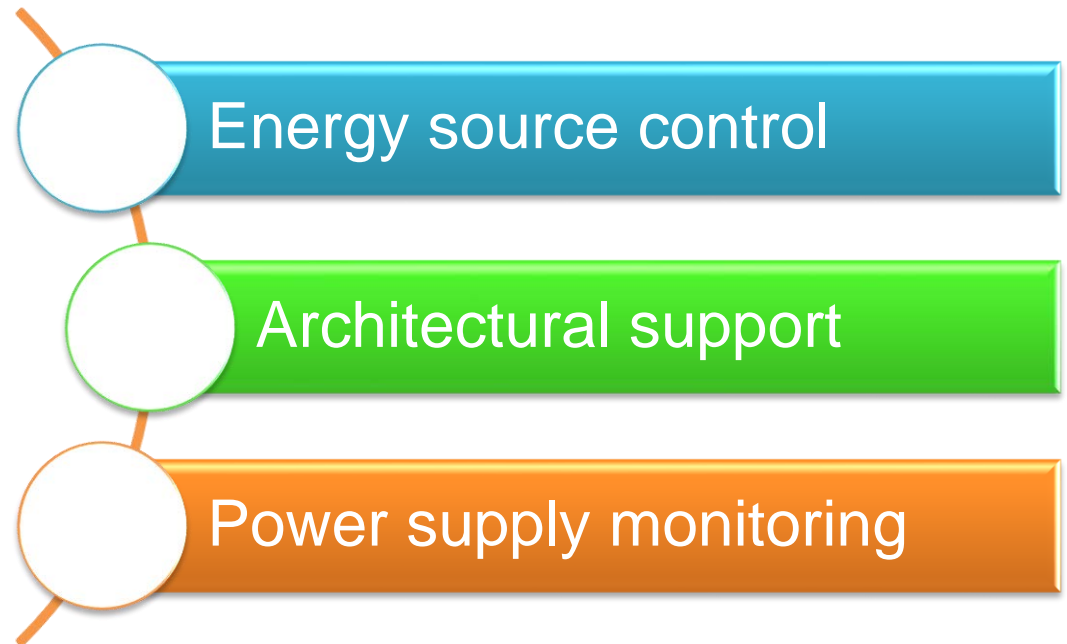
**Distributed, incremental green energy integration** allows a datacenter to double its power capacity with zero emissions and 25% cost reduction

---



# Oasis & Ozone: A unified power provisioning framework for scale-out green datacenters

---



**Welcome! (15:30-17:00 Session 5A - #1)**

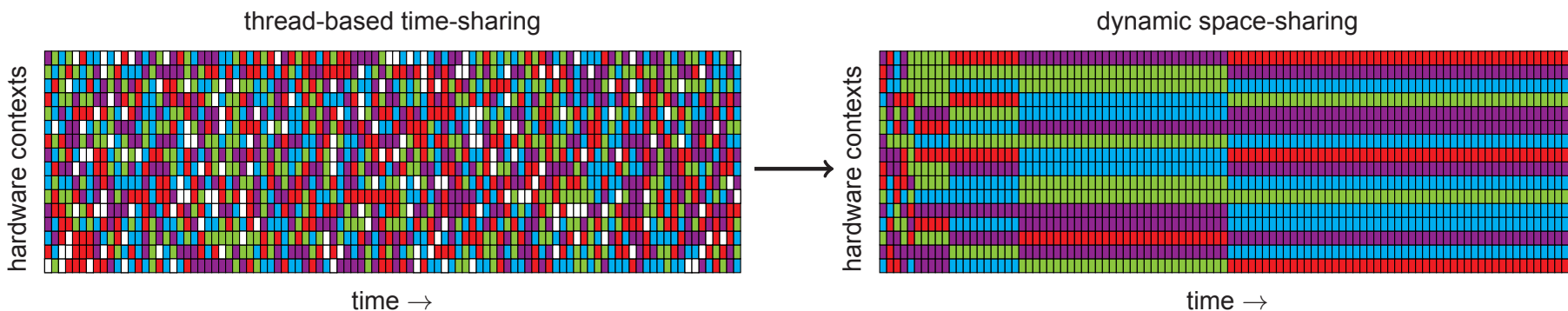
# Efficient Multiprogramming for Multicores with SCAF

Timothy Creech, Aparna Kotha, Rajeev Barua  
University of Maryland, College Park, MD

- **S**cheduling and **A**llocation with **F**eedback
  - Runtime system for multiprogramming parallel processes
  - ~15% gains over equipartitioning
  - Targeting shared-memory systems

# Efficient Multiprogramming for Multicores with SCAF

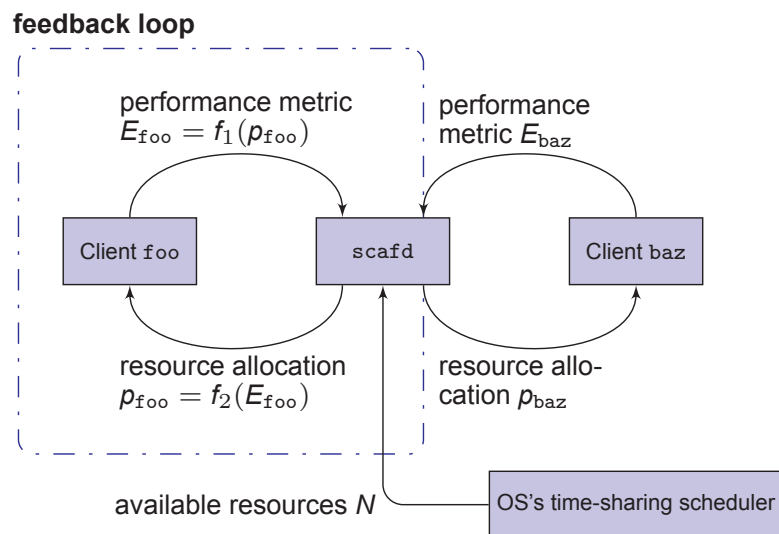
- Problem?
  - Parallel runtimes try to let the OS handle parallel multiprogramming
- **SCAF** runtime: automatic space-sharing
  - No porting, modification, recompilation
  - Policy: maximize sum of speedups



# Efficient Multiprogramming for Multicores with SCAF

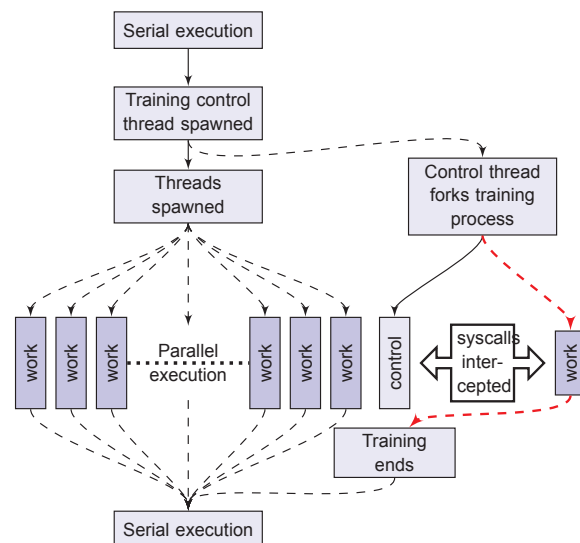
## Dynamic Allocation

- Realtime feedback
- Reward efficient processes



## Serial “Experiments”

- Estimate serial performance to reason about efficiency



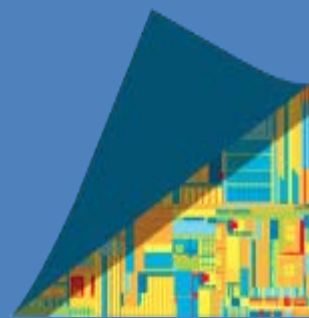


# Allocating Rotating Registers by Scheduling

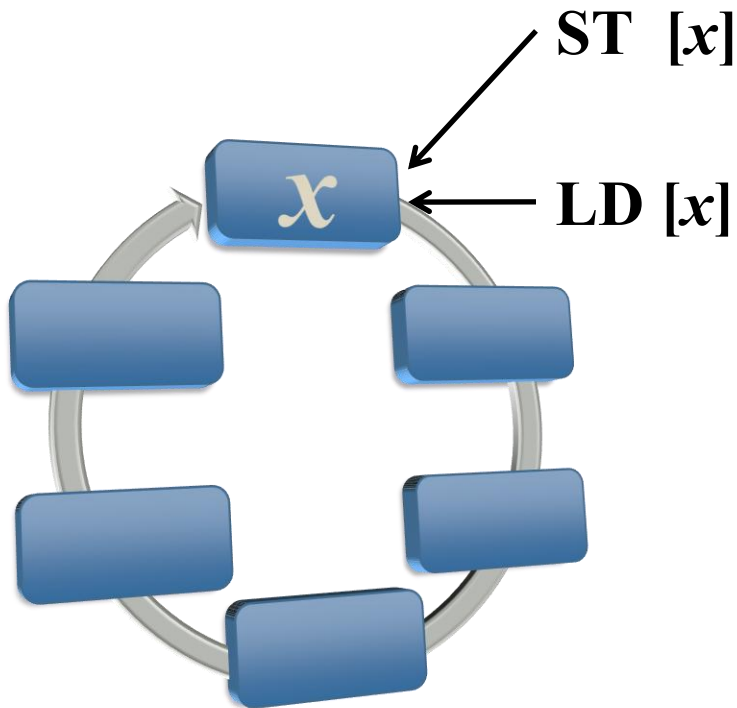
Hongbo Rong  
Cheng Wang

Hyunchul Park  
Youfeng Wu

Intel Labs



# Rotating Registers for Alias Detection



- Given a software pipelined schedule of a loop, how to allocate registers?
  - Detect ALL aliases
  - No false positive
  - Minimal spilling
  - Minimal registers

# Rotating Register Allocation = Scheduling!

- It is a software pipelining problem
  - A modulo schedule of lifetimes
- Contributions
  - Framework
  - A simple algorithm
  - Near-optimal results



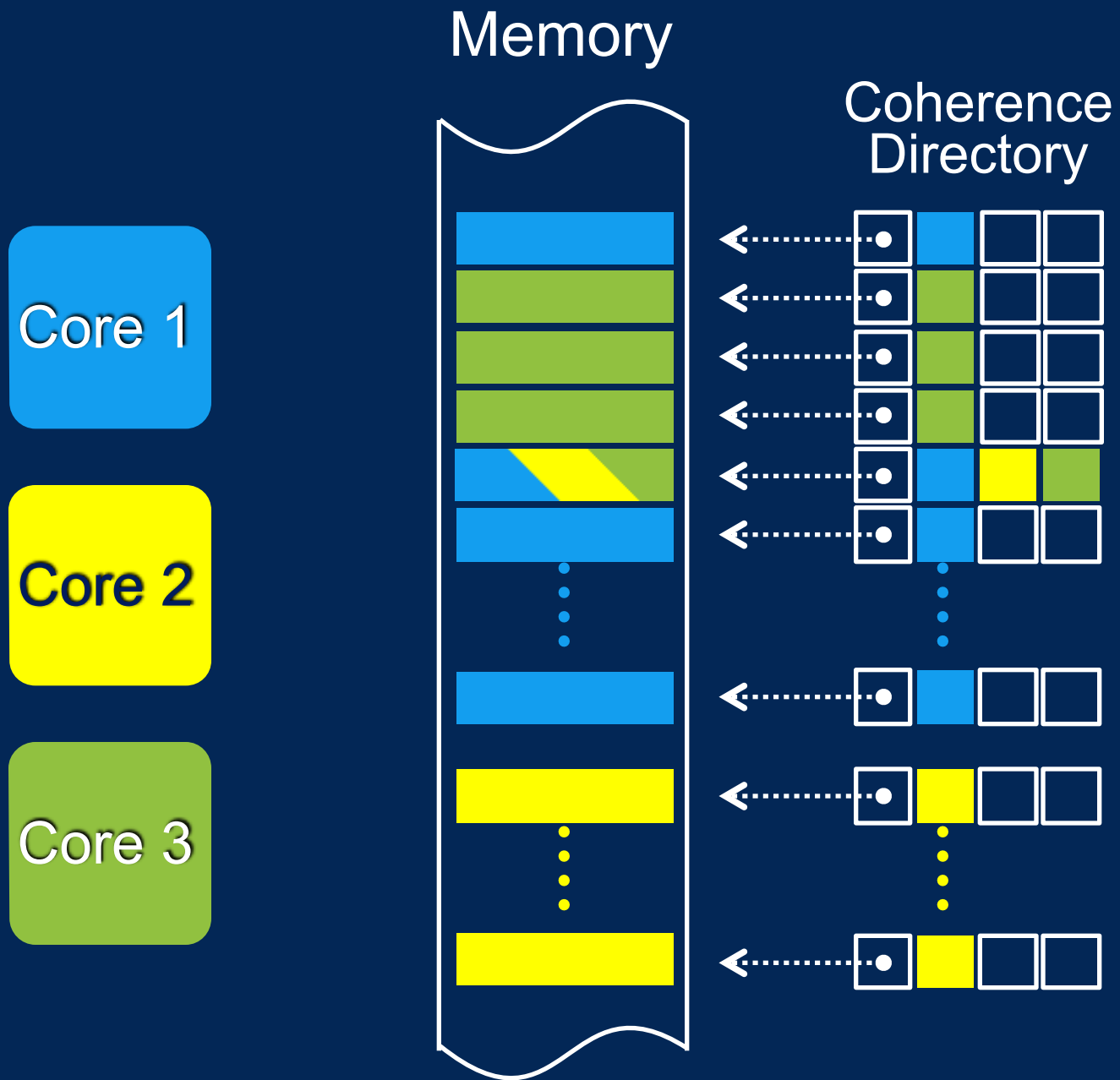
# Multi-Grain Coherence Directories

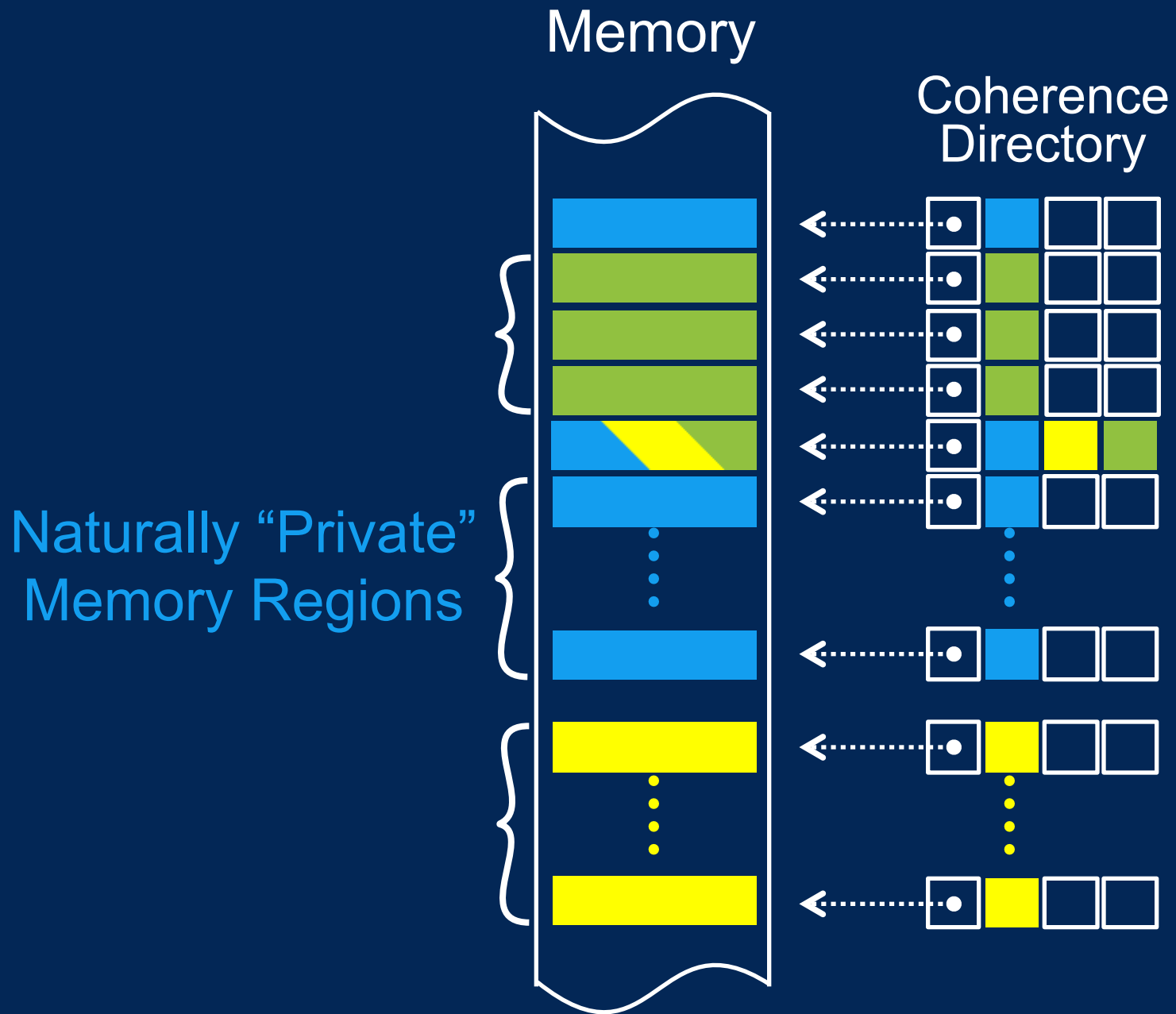
**Dr. Jason Zebchuk**

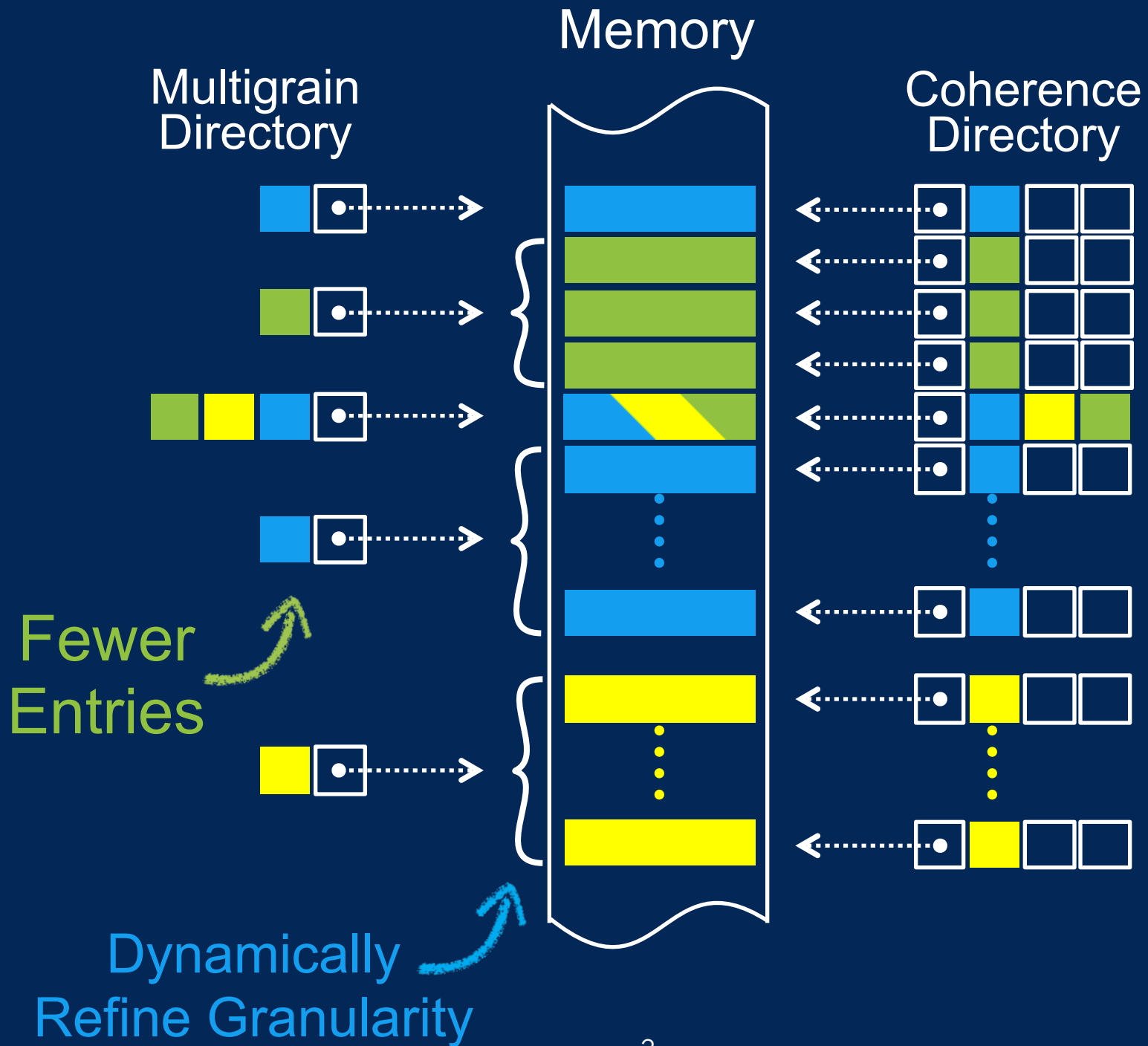
*Principal Engineer, Cavium Inc.*

Prof. Andreas Moshovos, *University of Toronto*

Prof. Babak Falsafi, *EcoCloud, EPFL*







# Multi-Grain Coherence Directory (MGD)

## Conceptual MGD Directory:

- ✓ Dynamically refine granularity of entries
- ✓ 78% fewer directory entries (on average)

## Practical MGD Directory:

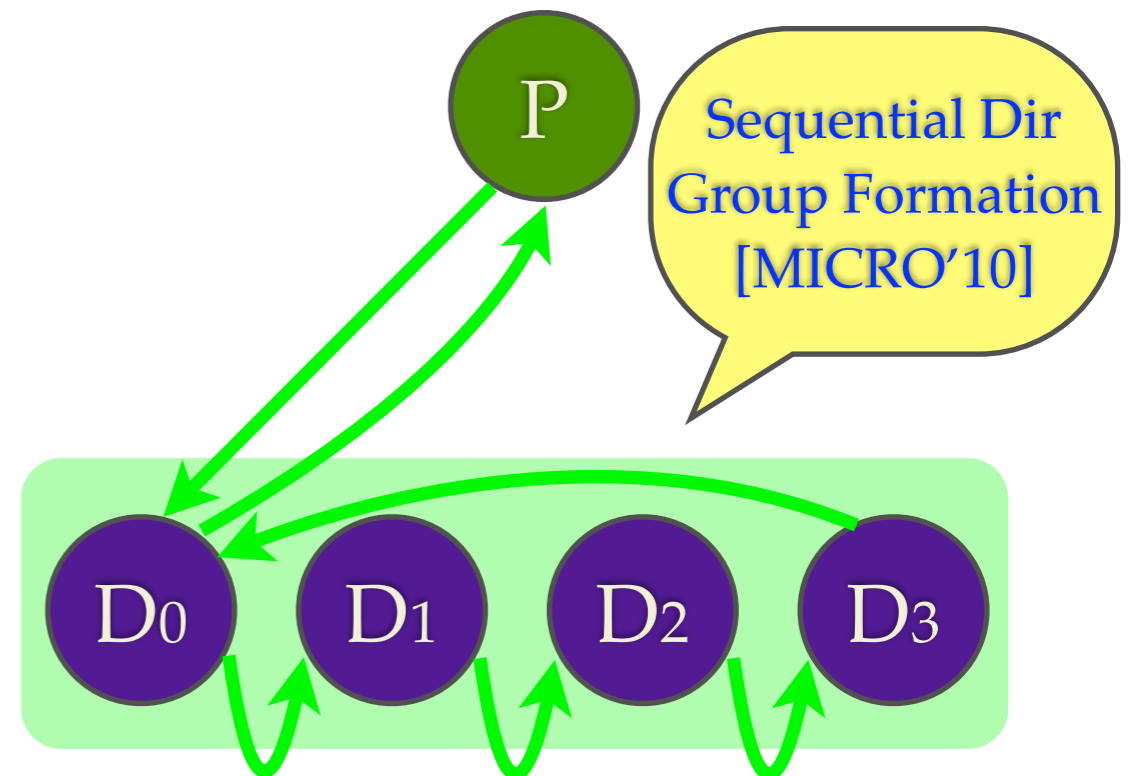
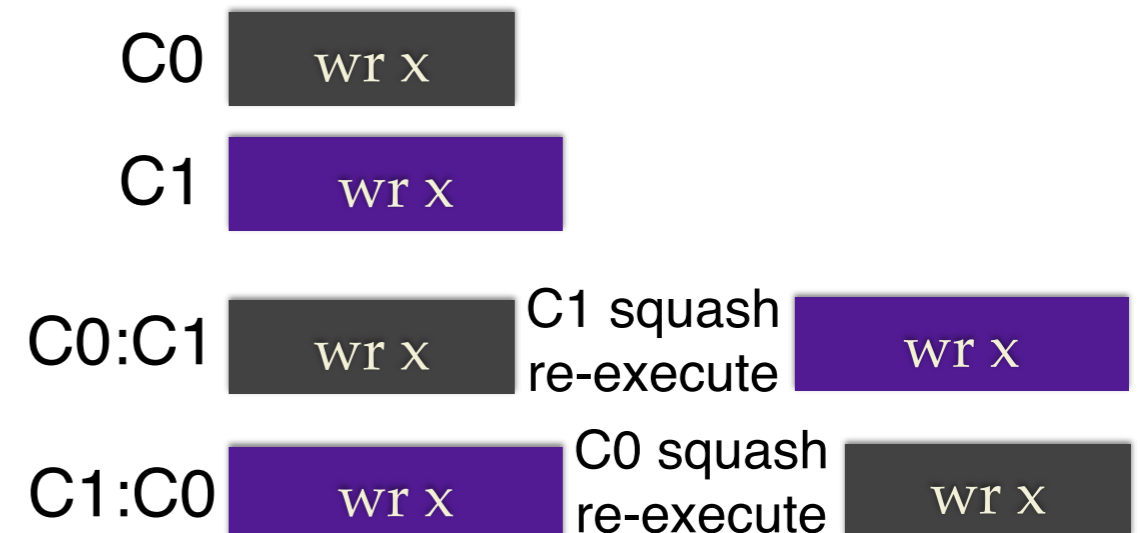
- ✓ Limited number of fixed granularities
- ✓ 41% less area
- ✓ Robust performance
- ✓ No coherence protocol changes

# BulkCommit: Scalable and Fast Commit of Atomic Blocks in a Lazy Multiprocessor Environment

Xuehai Qian, Josep Torrellas (UIUC)

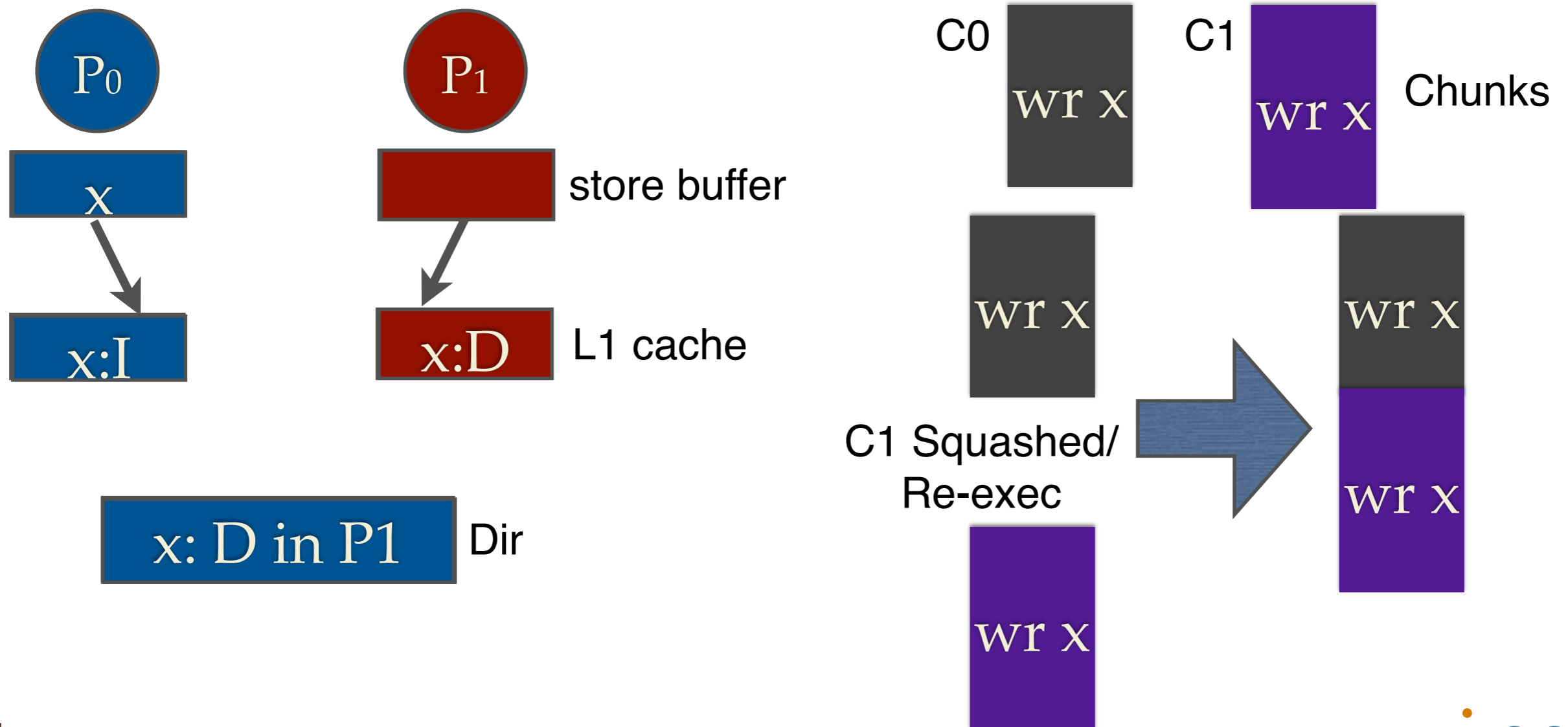
Benjamin Sahelices (Univ of Valladolid) and Depei Qian (Beihang Univ.)

- Problem:
  - Current atomic block (chunk) execution incurs unnecessary squashes
  - Atomic block commit operation in a lazy environment has sequential bottlenecks
- Our solution:
  - IntelliSquash: no squash on WAW-only conflict
  - IntelliCommit: parallel directory group formation

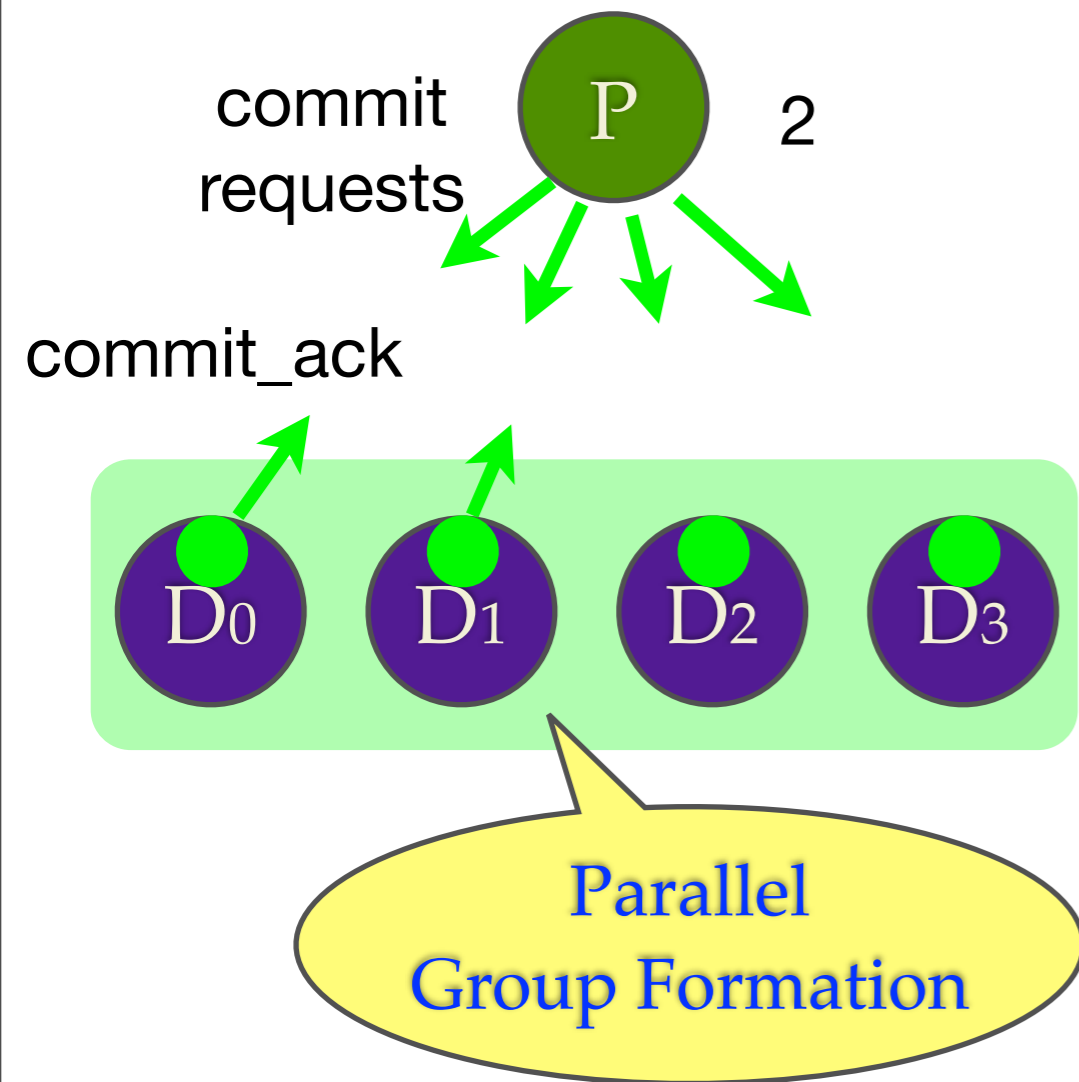


# IntelliSquash: No Squash on WAW-only Conflict

- Insight: WAW is a name dependence. It does not break semantic atomicity
- Similarity with two conflict stores from two processors
- If two chunks only have WAW conflicts, IntelliSquash serializes them without squash



# IntelliCommit: Parallel Directory Group Formation



- On chunk commit:
  - Processor sends commit requests to all the relevant directory modules
  - Directory module receives commit request:
    - Locks the memory lines
    - Responds with commit\_ack
  - Processor counts the number of commit\_acks received
  - Processor sends commit\_confirm when it receives the expected number of commit\_acks
- Challenge: resolving conflicts from two processors
- ChunkSort: ordering all the conflicting chunks in the same order in all relevant directories by **preemption**





# Large-Reach Memory Management Unit Caches

Abhishek Bhattacharjee  
Rutgers University

*International Symposium on Microarchitecture-46*

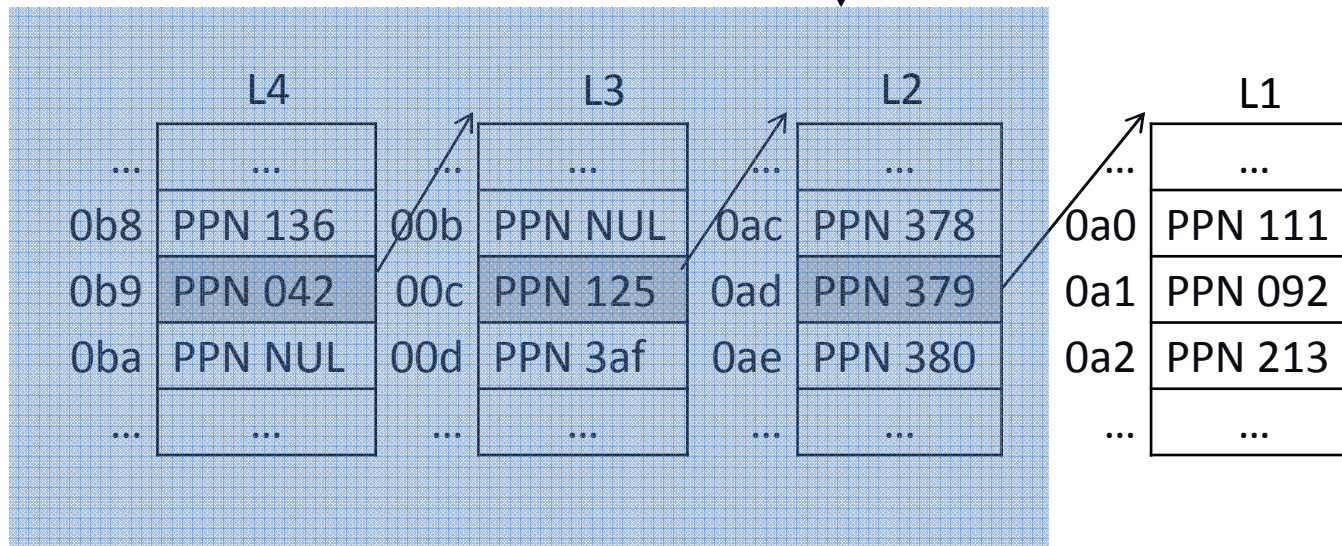
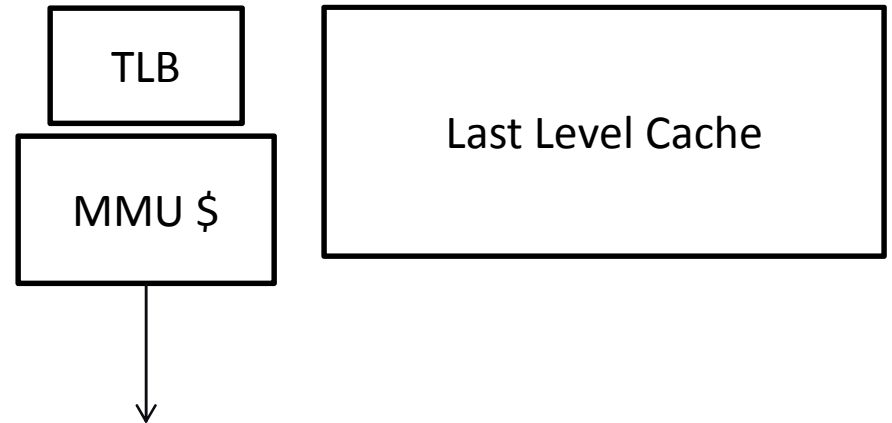
December 2013



# Address Translation and MMU Caches

On a typical TLB miss:

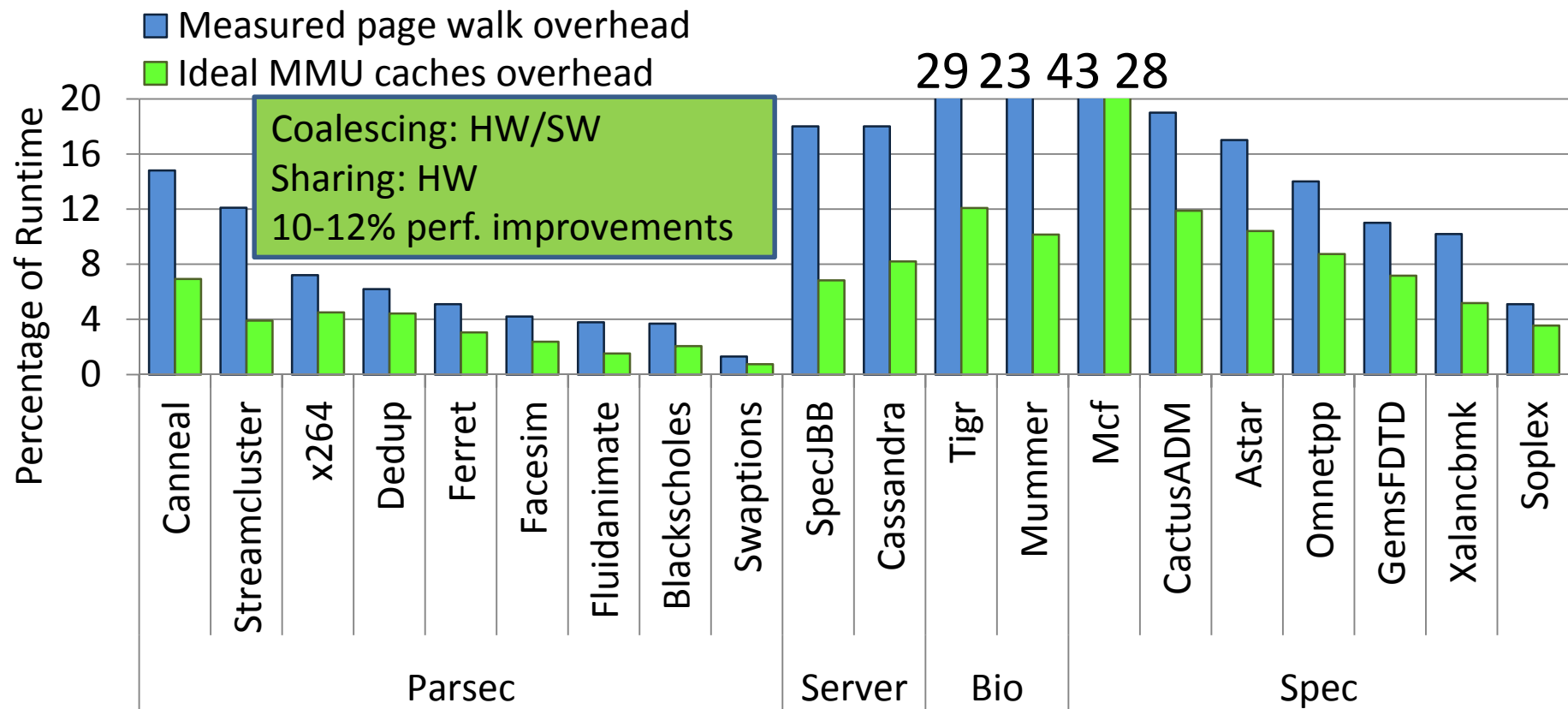
- x86: 1-4 memory references
- ARM: 1-2 memory references
- Sparc: 1-2 memory references





# Approaching an Ideal MMU Cache

- Intel i7: 8 cores, 8GB memory, 512-entry L2 TLB, and 8MB LLC



# GPU LLC Management for 3D Scene Rendering

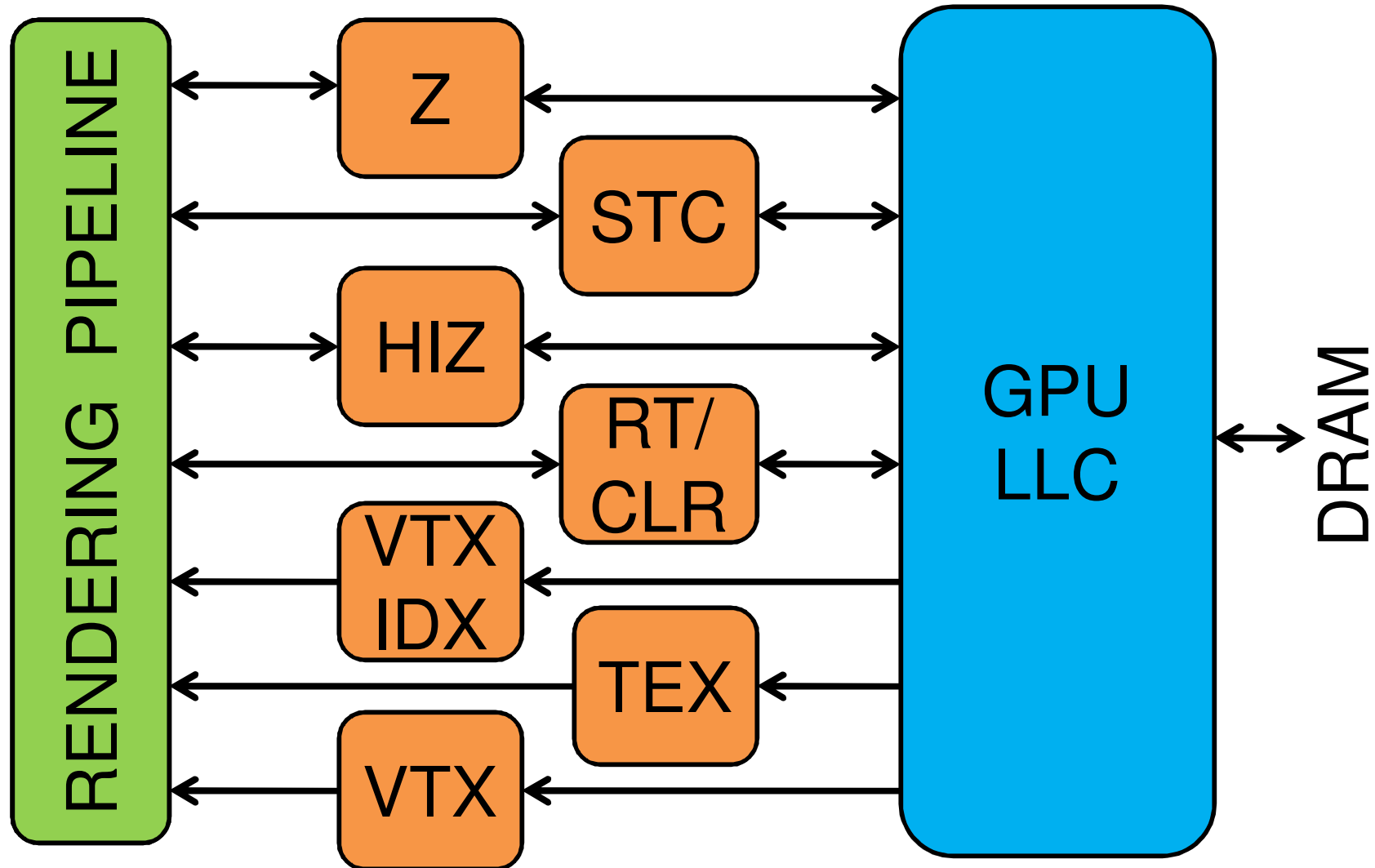
Jayesh Gaur, Intel

Raghuram Srinivasan, Ohio State

Sreenivas Subramoney, Intel

Mainak Chaudhuri, IIT Kanpur

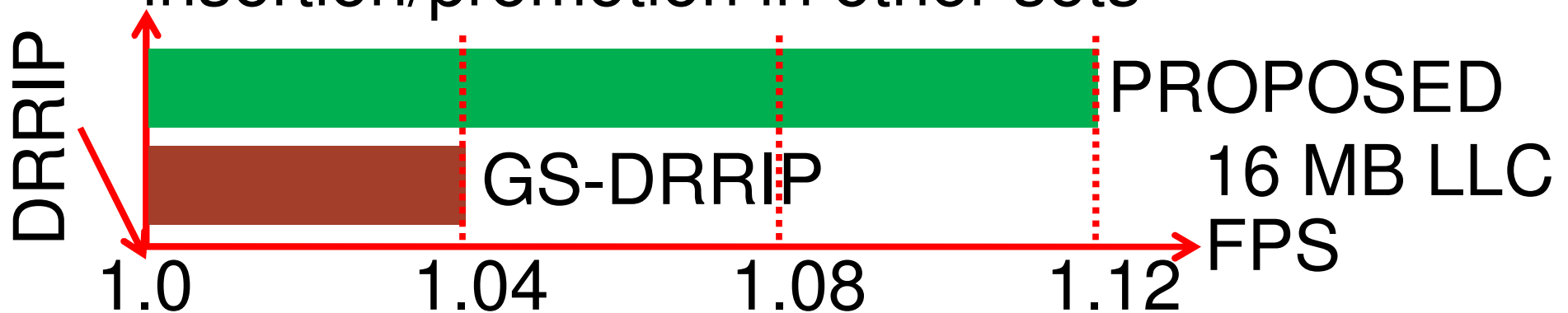
# GPU last-level cache interface



**Efficient management of the LLC shared between different 3D rendering streams**

# Solution approach and results

- Inter- and intra-stream reuses in LLC
  - RT, TEX, Z are dominant in the LLC traffic
  - Significant reuse from RT production to TEX consumption (render to texture)
  - Intra-stream reuses vary across streams
- Learn intra- and inter-stream dynamic reuse probabilities from sample sets and modulate insertion/promotion in other sets



# GPU Transactional Memory



# GPU Transactional Memory



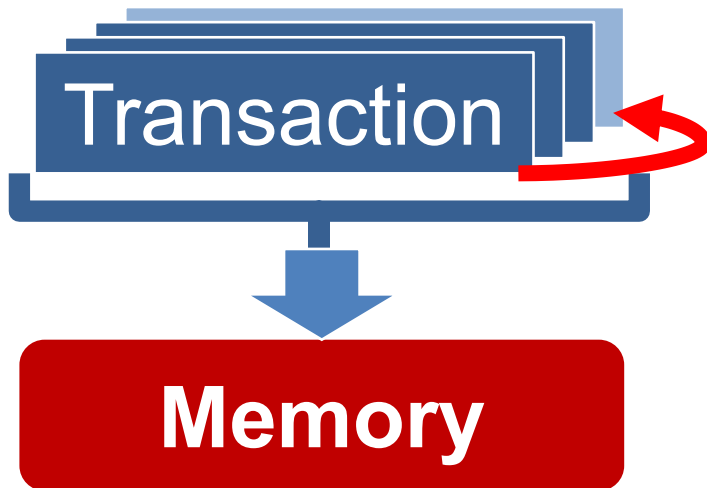


# Energy Efficient GPU Transactional Memory via Space-Time Optimizations

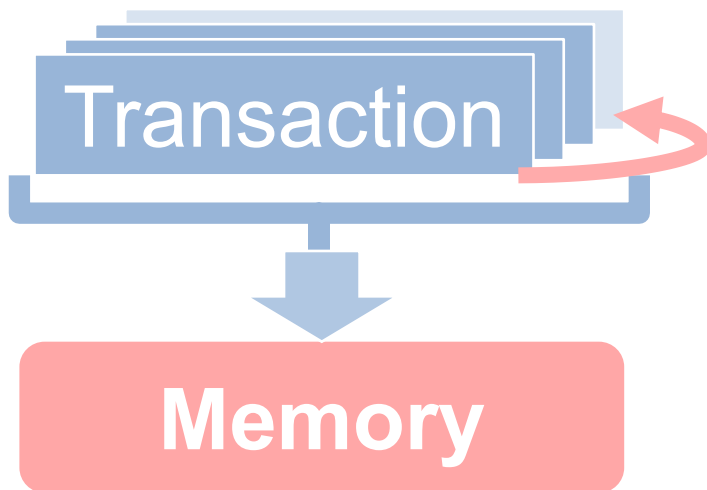
Wilson Fung, Tor Aamodt (UBC)



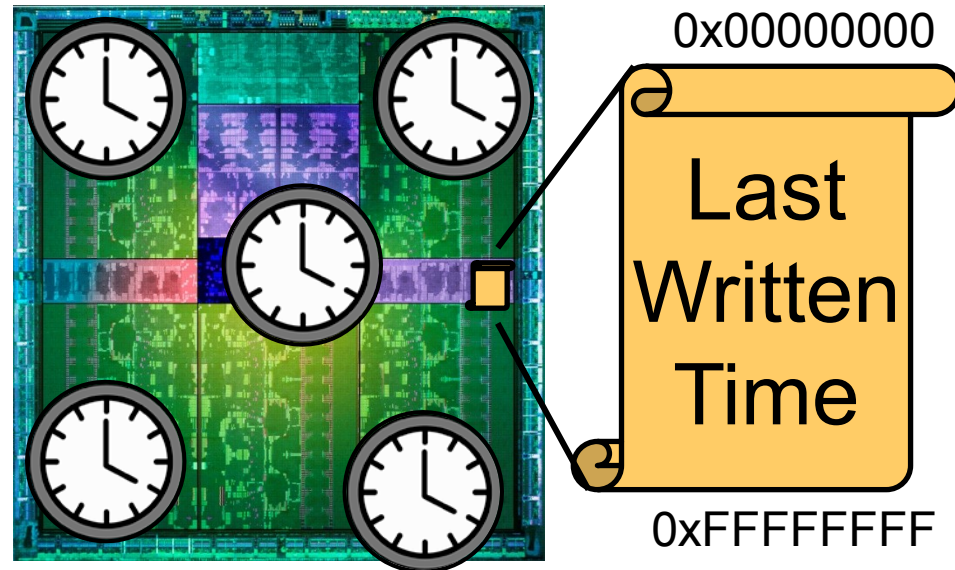
# Warp-Level Transaction Management



# Warp-Level Transaction Management



# Temporal Conflict Detection



Low conflict workloads:  
**TM  $\approx$  Fine-Grained Locks**



**65%**  
**Speedup**

**2X  $\rightarrow$  1.3X**  
**Energy Usage**

# Energy Efficient GPU Transactional Memory

via Space-Time Optimizations

**Wednesday 10:15am, Section 6A**

---

Low conflict workloads:  
**TM  $\approx$  Fine-Grained Locks**



**65%**  
Speedup

**2X  $\rightarrow$  1.3X**  
Energy Usage

# Memory

<http://depositphotos.com/3735495/stock-photo-Kids-writing-in-sand.html>





## Memory

<http://depositphotos.com/3735495/stock-photo-Kids-writing-in-sand.html>



## Storage

<http://www.bbc.co.uk/news/uk-england-leeds-15157345>



## Persistent Memory

<http://www.lookandlearn.com/2012/11/5/sumerian-civilisation-and-the-birth-of-writing/>





## Memory

<http://depositphotos.com/3735495/stock-photo-Kids-writing-in-sand.html>



## Storage

<http://www.bbc.co.uk/news/uk-england-leeds-15157345>



## Persistent Memory

<http://www.lookandlearn.com/2015/2115/sumerian-civilisation-and-the-birth-of-writing/>



*Dry it*



<http://historicconnections.webs.com/mesopotamia.htm>

# How to Turn Data Updates Into Permanent Records?

# How to Turn Data Updates Into Permanent Records?

*Previous: With “Log”*



**Leave original data intact**

# How to Turn Data Updates Into Permanent Records?

*Previous: With “Log”*



**Leave original data intact**

*Our Design: “Kiln”*



**Directly overwrite original data**

# How to Turn Data Updates Into Permanent Records?

*Previous: With “Log”*

*Our Design: “Kiln”*

 **3x**

speedup across 6 benchmarks

Leave original data intact

Directly overwrite original data

# Details about “Kiln”

*Closing the Performance Gap Between  
Systems With and Without  
Persistence Support*

Jishen Zhao

*Penn State*

Sheng Li

*HP Labs*

Doe Hyun Yoon

*IBM Research*

Yuan Xie

*Penn State/AMD Research*

Norm Jouppi

*Google*

▶ **Poster Session: Tue (Dec.10) 2 – 3PM**

**Presentation: Wed (Dec. 11) 9:45AM  
Session 6B**

**(Alpha Gamma Rho Room)**

# Aegis: Partitioning Data Block for Efficient Recovery of Stuck-at-Faults in Phase Change Memory

Jie Fan, Jiwu Shu,  
Youhui Zhang, and  
Weimin Zhen

Song Jiang



清華大學  
Tsinghua University



WAYNE STATE  
UNIVERSITY

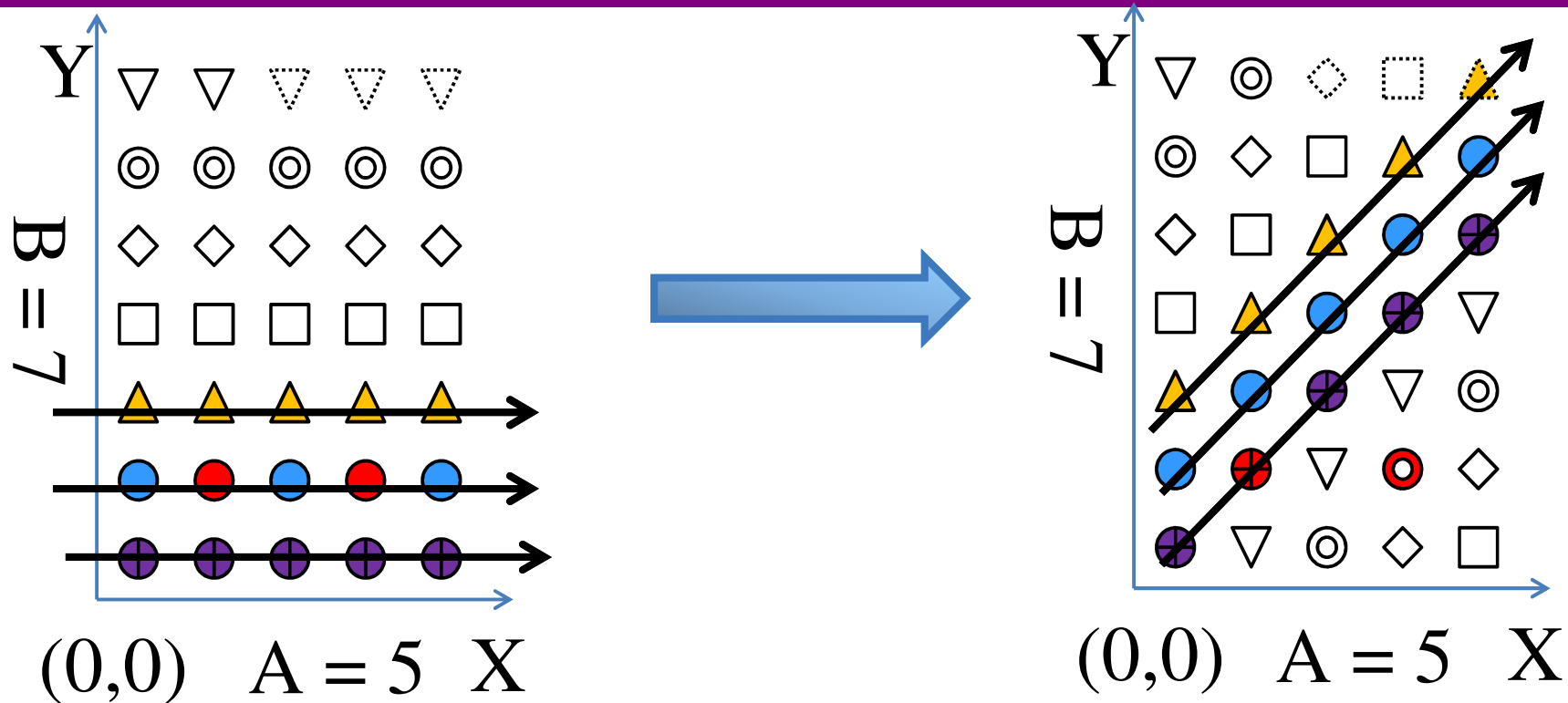
# Stuck-at Faults in PCM

---

- PCM has limited endurance.
- Stuck-at faults occur when memory cell fails to change its value.
  - It is a major type of faults in PCM.
  - This type of faults is permanent and accumulates.
  - **Values in such faulty cells can still be read.**
- Inversion-based correction
  - Partition data block into a number of groups and exploit the fact that stuck-at values are still readable (e.g., SAFER).
  - **Each group can tolerate only one fault.**
- **Proposal of an efficient partition scheme separating faults into different groups.**



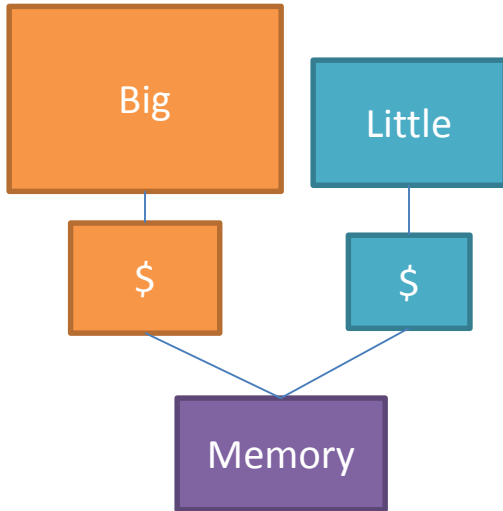
# Illustration of Aegis Partition



- PCM bits are laid out on an  $A \times B$  Cartesian plane.
- Aegis considers all points on a line as a group.
- **Any two bits in the same line will not be in the same line** after Aegis changes slope of the lines.
- Aegis distributes faults more evenly to tolerate more faults with lower overhead.

# Fine-grained Heterogeneity

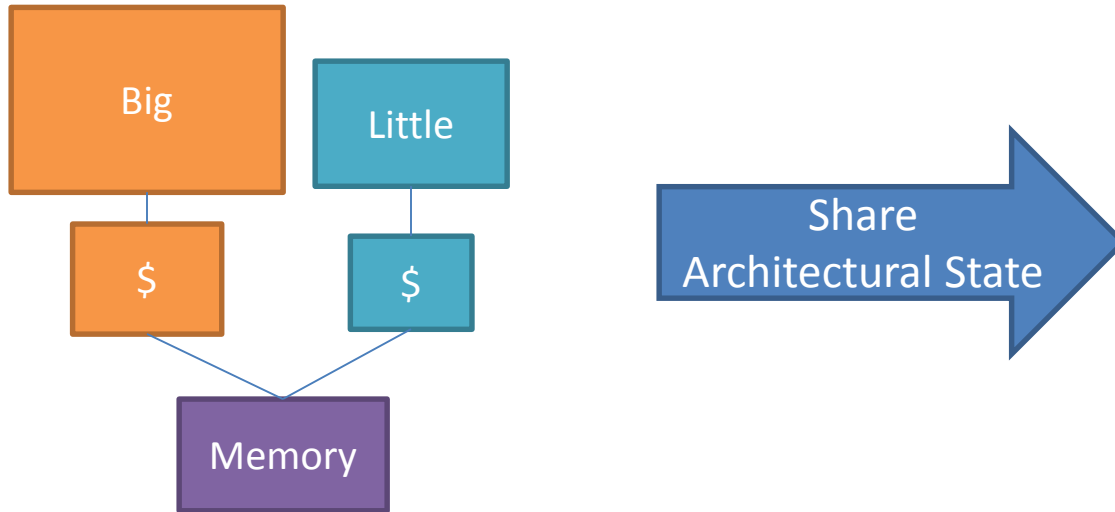
Traditional big.LITTLE Architecture



Transfer Overhead: ~**20K** cycles

# Fine-grained Heterogeneity

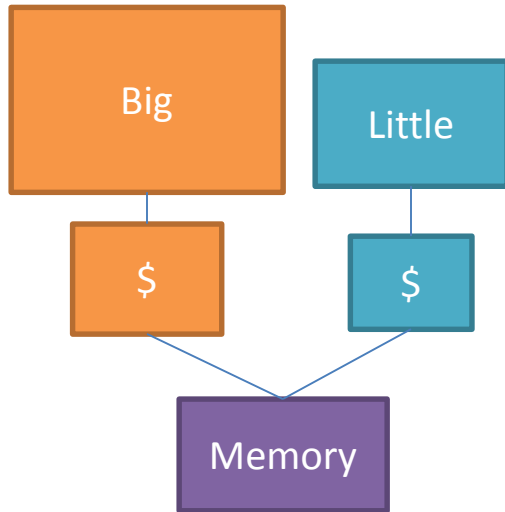
Traditional big.LITTLE Architecture



Transfer Overhead: ~**20K** cycles

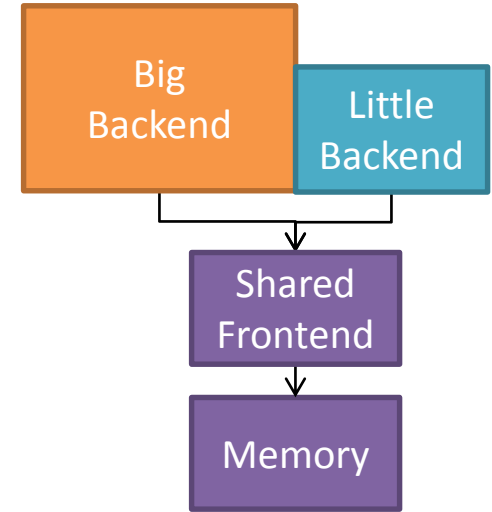
# Fine-grained Heterogeneity

Traditional big.LITTLE Architecture



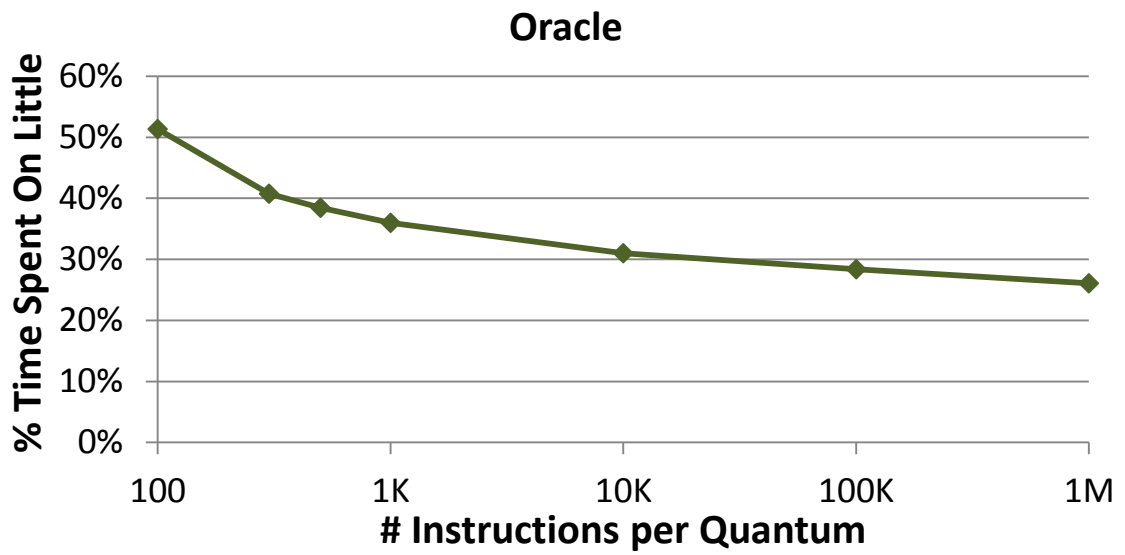
Transfer Overhead: ~**20K** cycles

Composite Cores Architecture

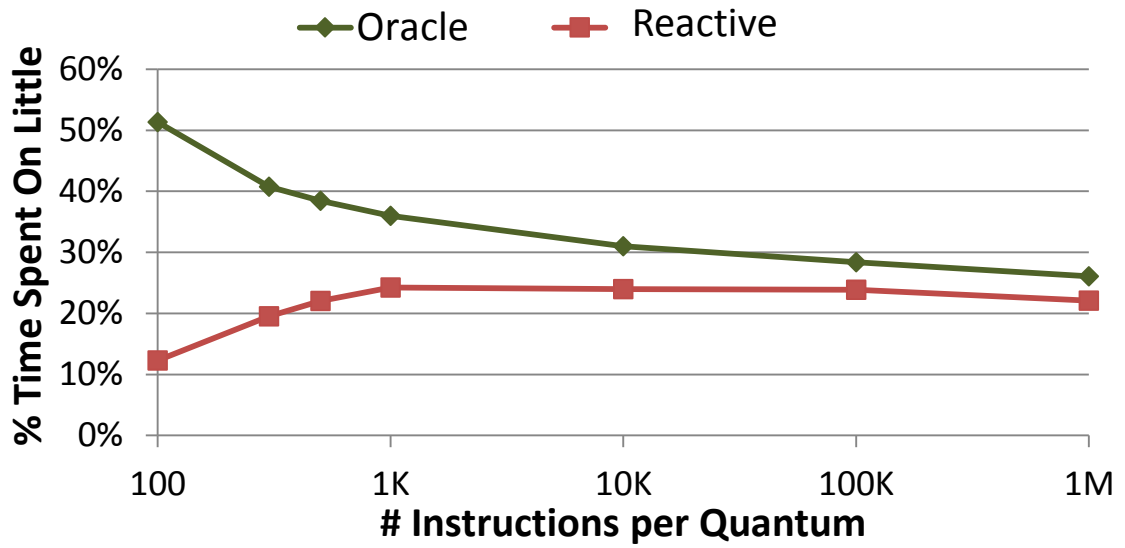


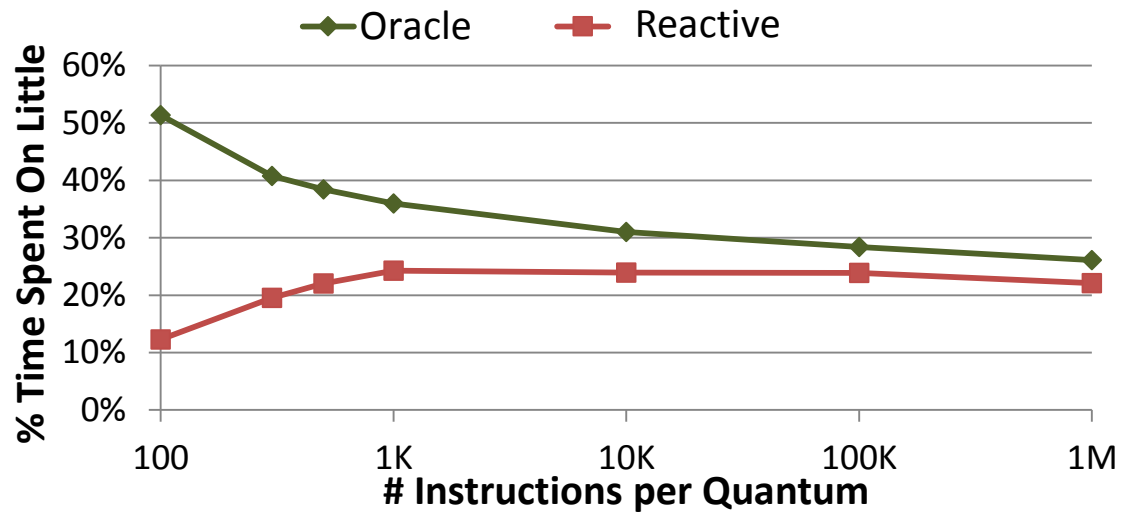
Transfer Overhead: ~**35** cycles

Traditional Reactive  
Controllers



Traditional Reactive  
Controllers

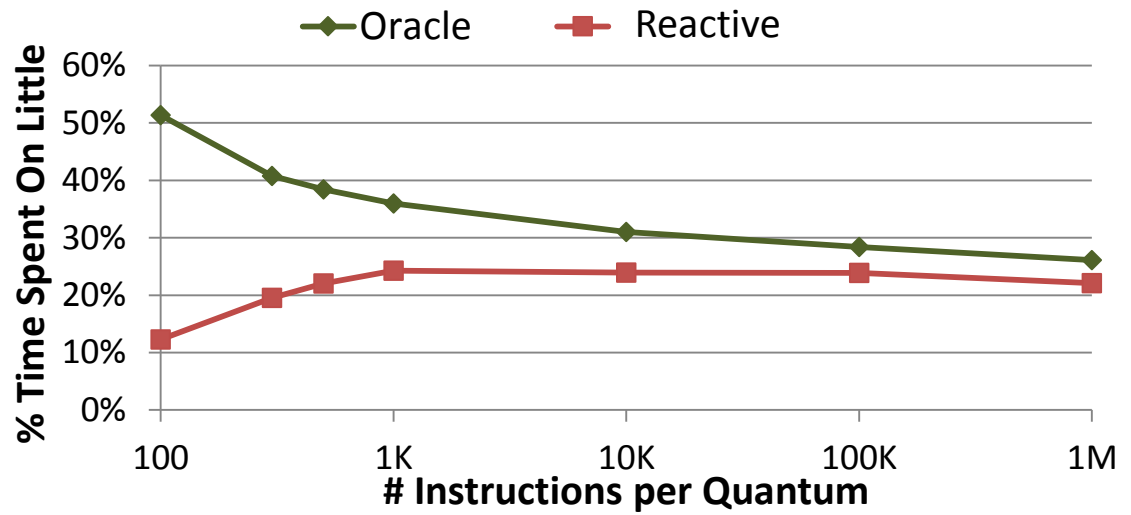




Traditional Reactive  
Controllers

Don't React – Predict!

Traditional Reactive  
Controllers

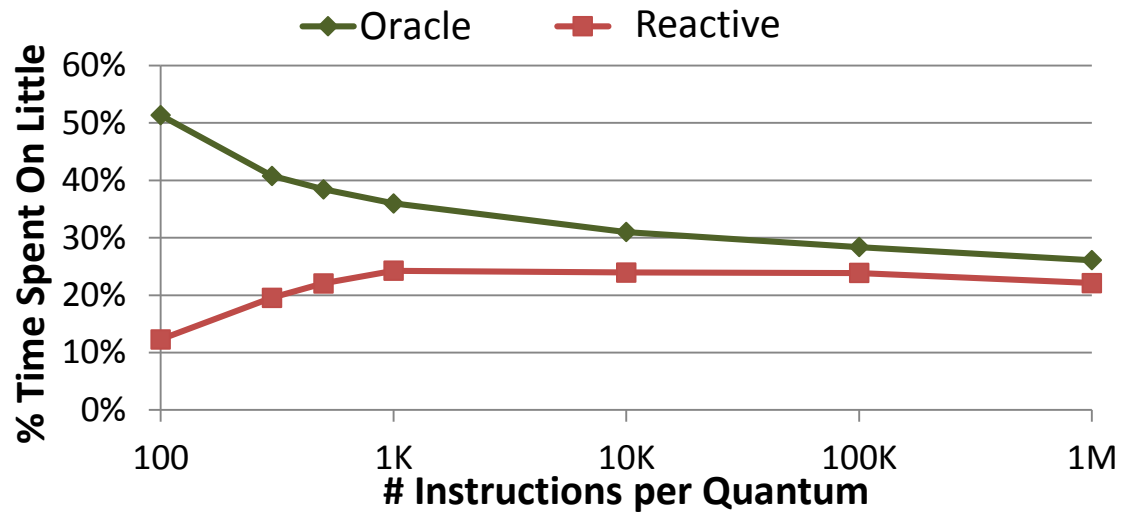


Don't React – Predict!

**Code repeats** (loops, functions)

**Behavior repeats** in the same program context



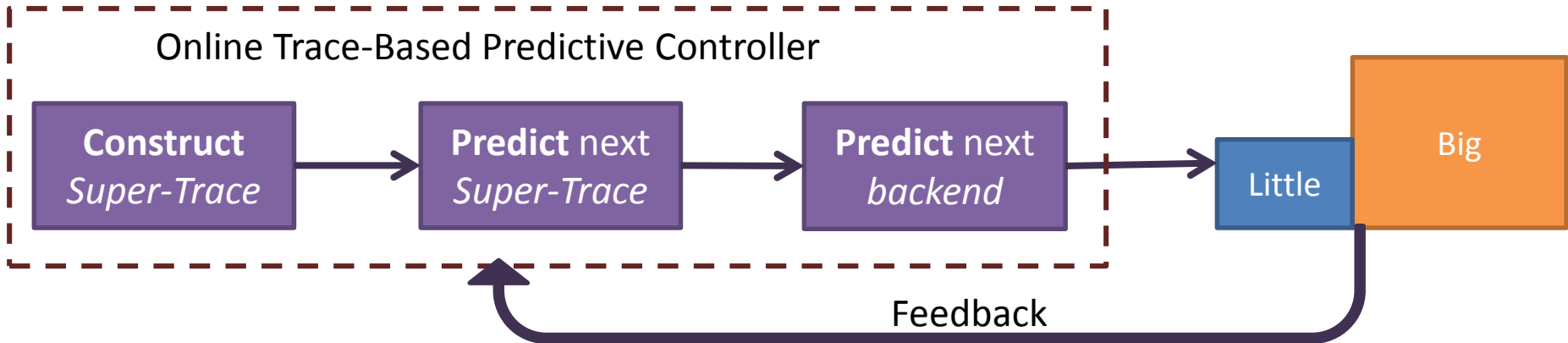


Traditional Reactive Controllers

# Don't React – Predict!

Code repeats (loops, functions)

Behavior repeats in the same program context



# Trace Based Switching For A Tightly Coupled Heterogeneous Core

Reduce energy consumption by **43%** more than state of art

Shruti Padmanabha, Andrew Lukefahr, Reetuparna Das, Scott Mahlke

Micro-46

December 2013

**Full Talk: Session 7**  
**Wednesday, 11<sup>th</sup> December 2013: 11 am**

# HETEROGENEOUS SYSTEM COHERENCE

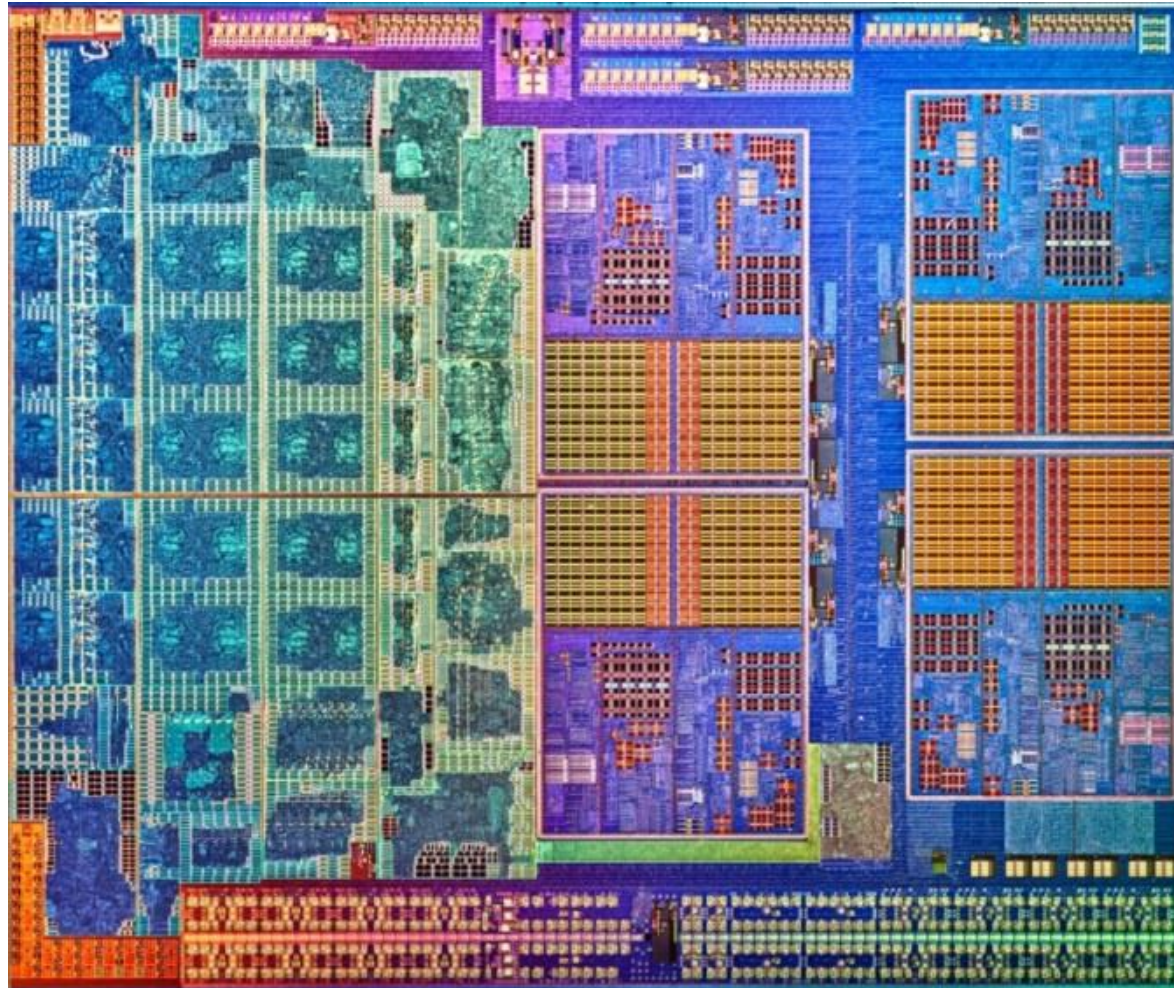
## for Integrated CPU-GPU Systems



**Jason Power\***, Arkaprava Basu\*, Junli Gu<sup>†</sup>, Sooraj Puthoor<sup>†</sup>,  
Bradford M Beckmann<sup>†</sup>, Mark D Hill<sup>\*†</sup>, Steven K Reinhardt<sup>†</sup>, David A Wood<sup>\*†</sup>

\*University of Wisconsin-Madison

<sup>†</sup>Advanced Micro Devices, Inc.



# HETEROGENEOUS SYSTEM COHERENCE

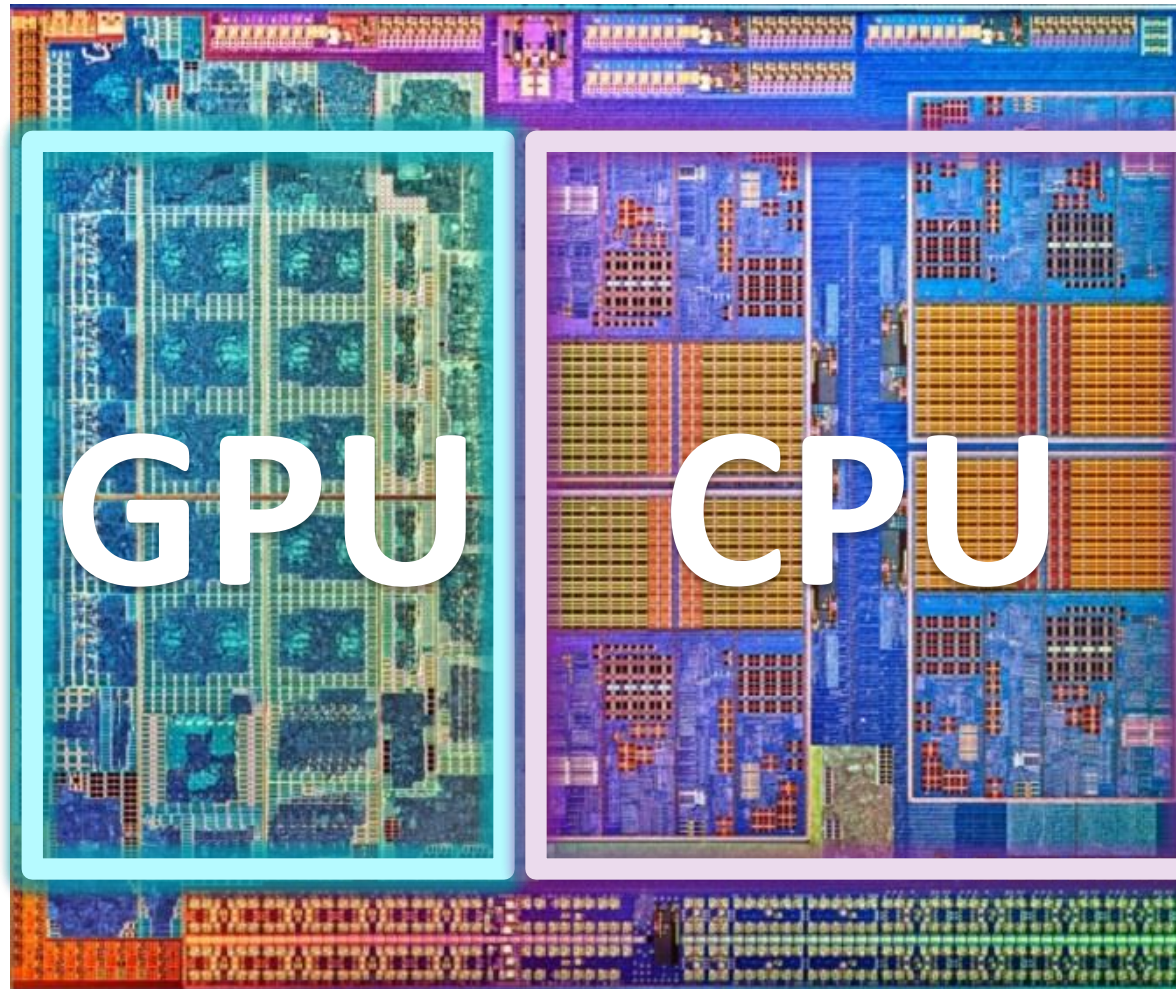
## for Integrated CPU-GPU Systems



**Jason Power\***, Arkaprava Basu\*, Junli Gu<sup>†</sup>, Sooraj Puthoor<sup>†</sup>,  
Bradford M Beckmann<sup>†</sup>, Mark D Hill<sup>\*†</sup>, Steven K Reinhardt<sup>†</sup>, David A Wood<sup>\*†</sup>

\*University of Wisconsin-Madison

<sup>†</sup>Advanced Micro Devices, Inc.



# HETEROGENEOUS SYSTEM COHERENCE

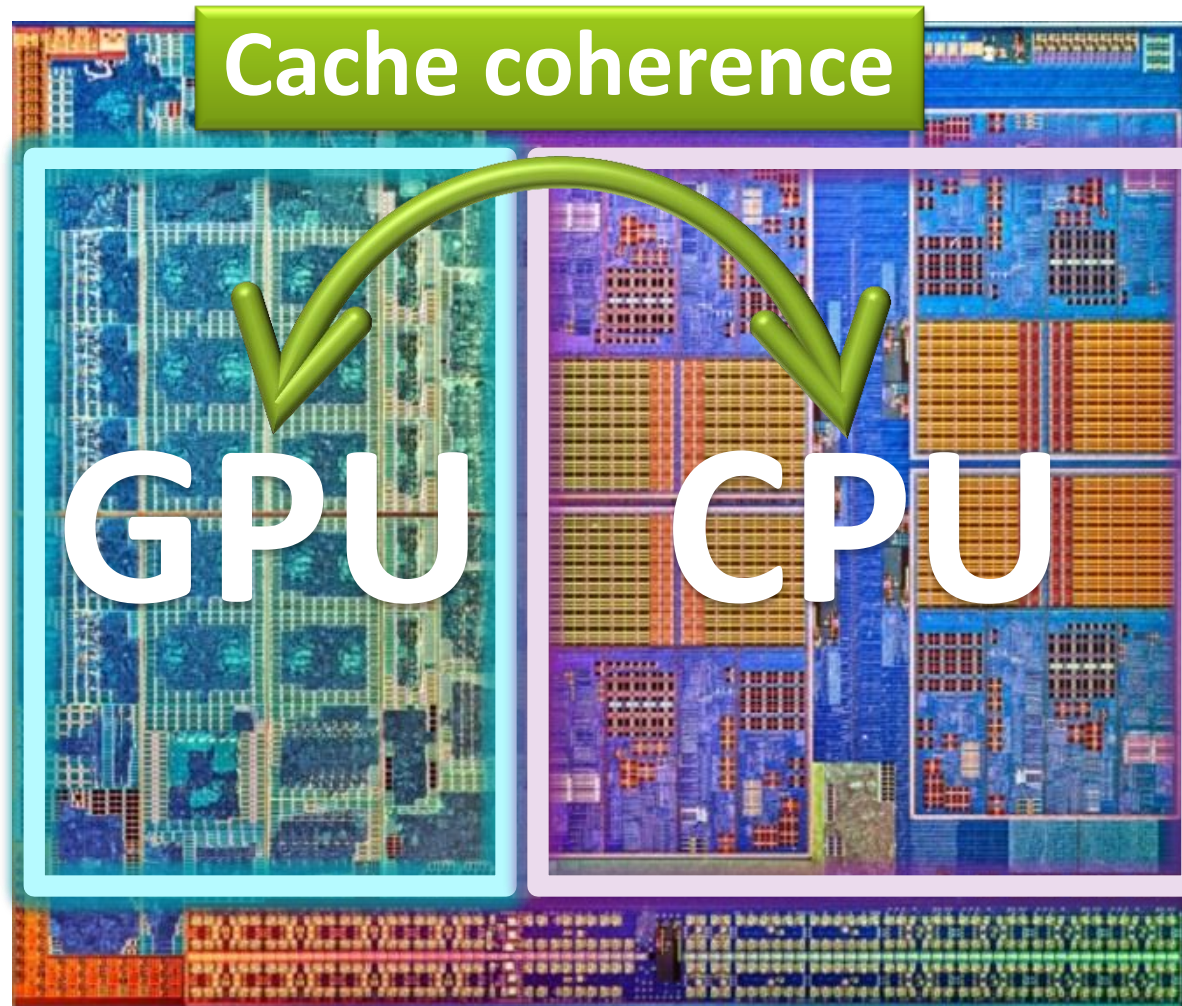
## for Integrated CPU-GPU Systems



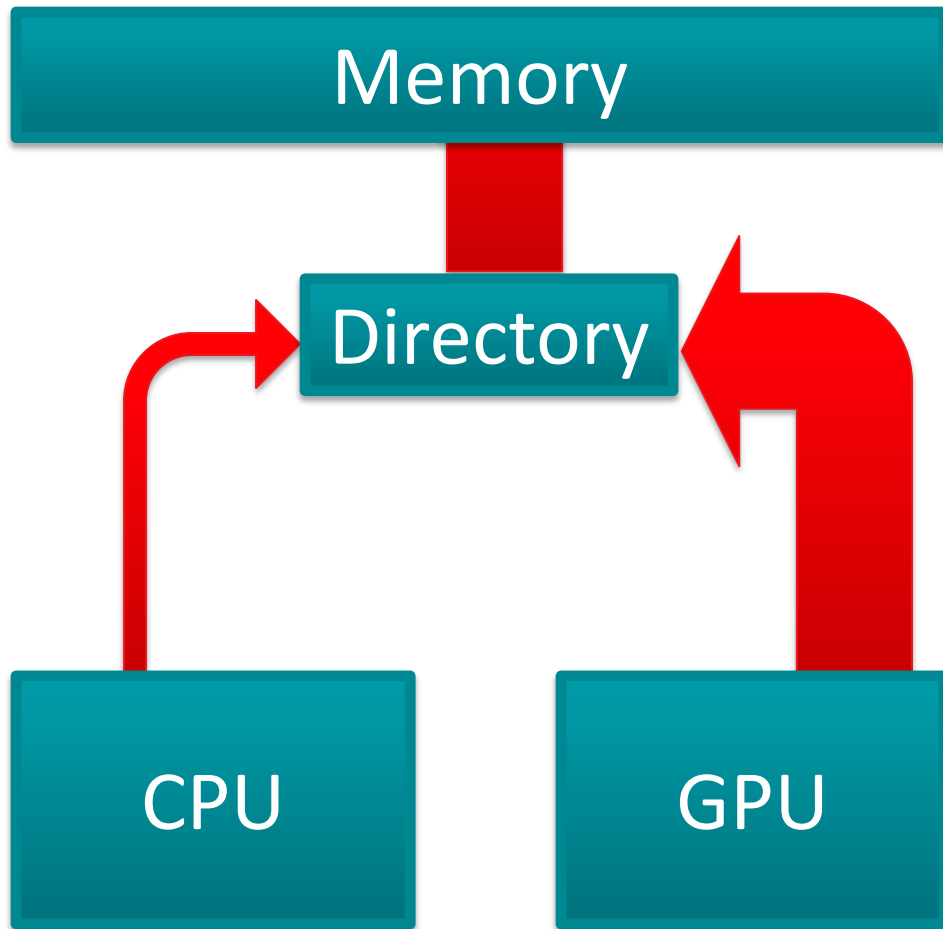
**Jason Power\***, Arkaprava Basu\*, Junli Gu<sup>†</sup>, Sooraj Puthoor<sup>†</sup>,  
Bradford M Beckmann<sup>†</sup>, Mark D Hill<sup>\*†</sup>, Steven K Reinhardt<sup>†</sup>, David A Wood<sup>\*†</sup>

\*University of Wisconsin-Madison

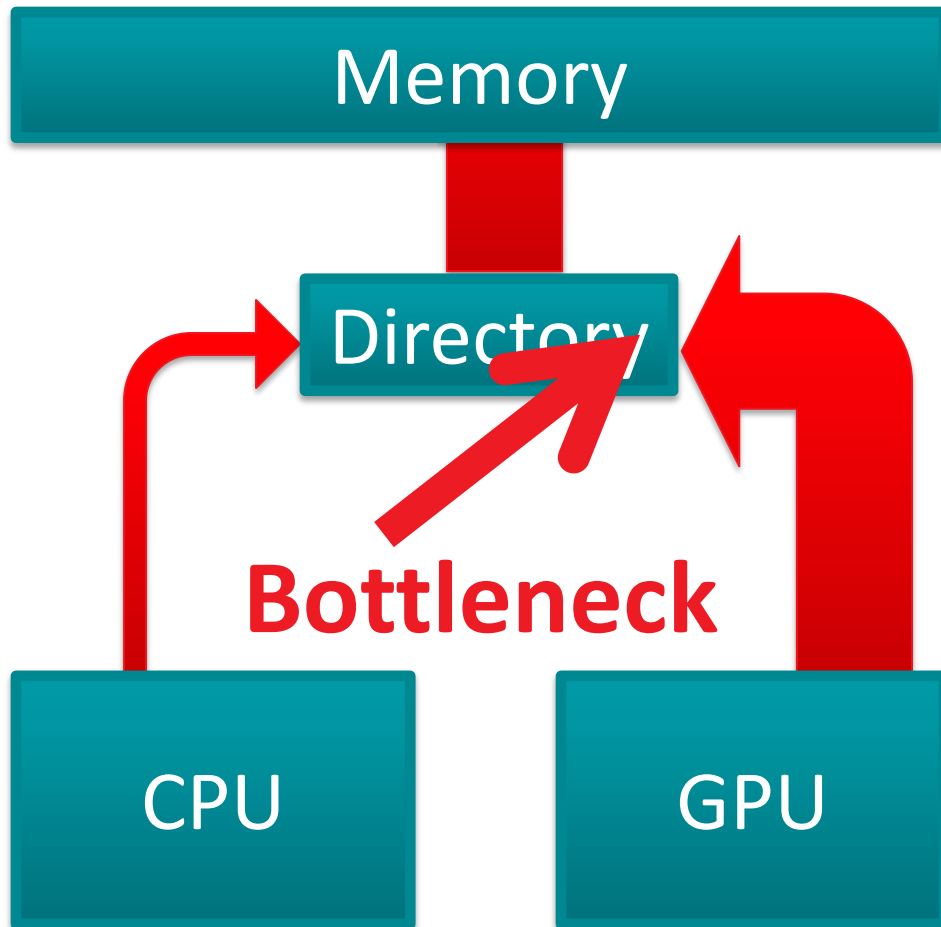
<sup>†</sup>Advanced Micro Devices, Inc.



# CPU-GPU COHERENCE



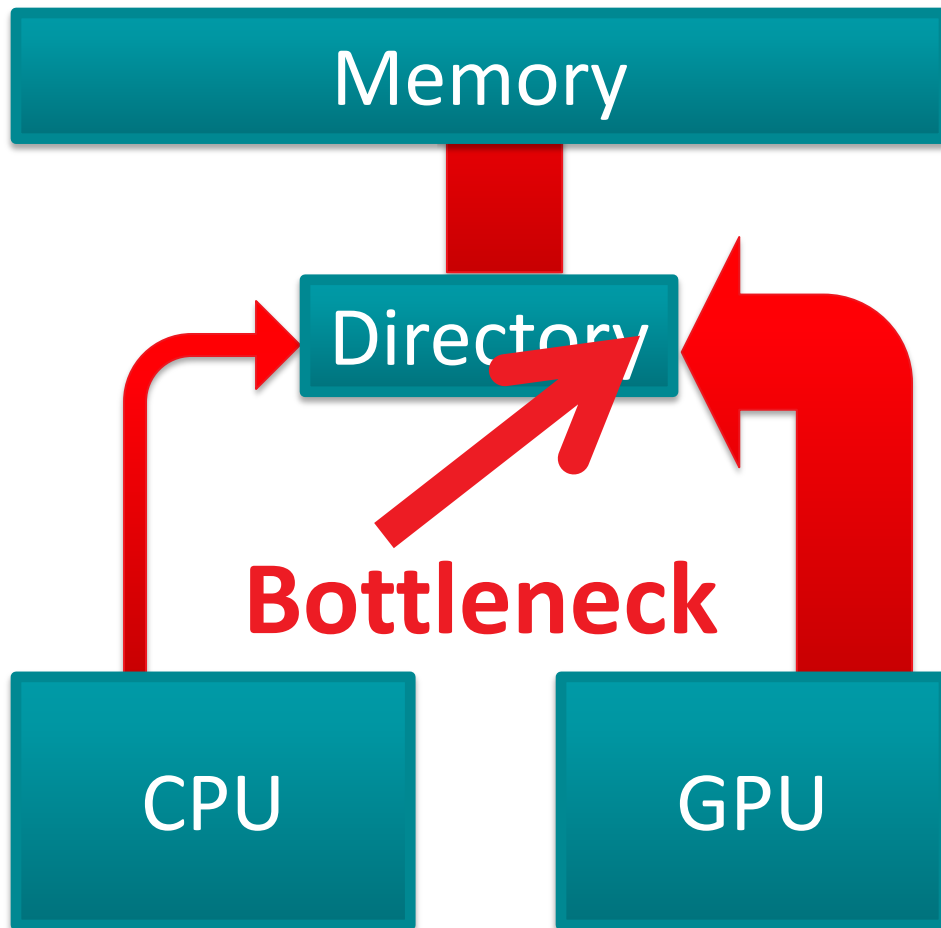
# CPU-GPU COHERENCE



**4x slowdown**

**Bottleneck**

# CPU-GPU COHERENCE

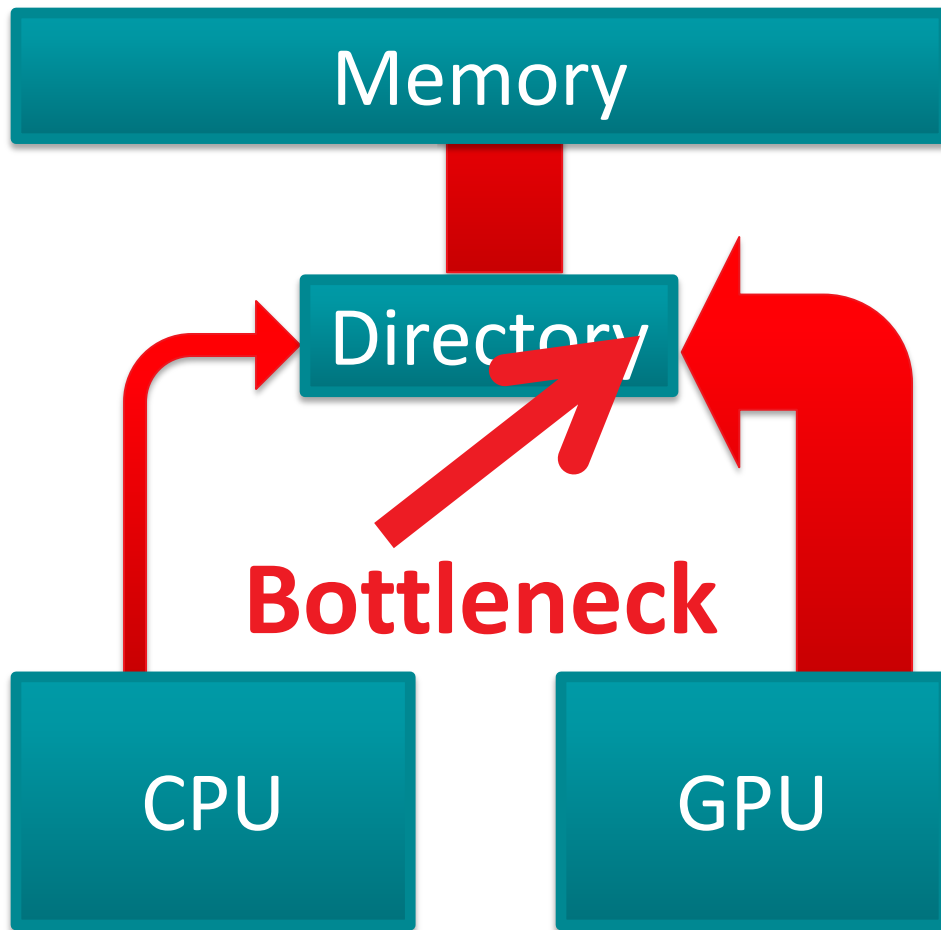


**4x slowdown**

~~1. More than 2 requests per cycle~~



# CPU-GPU COHERENCE

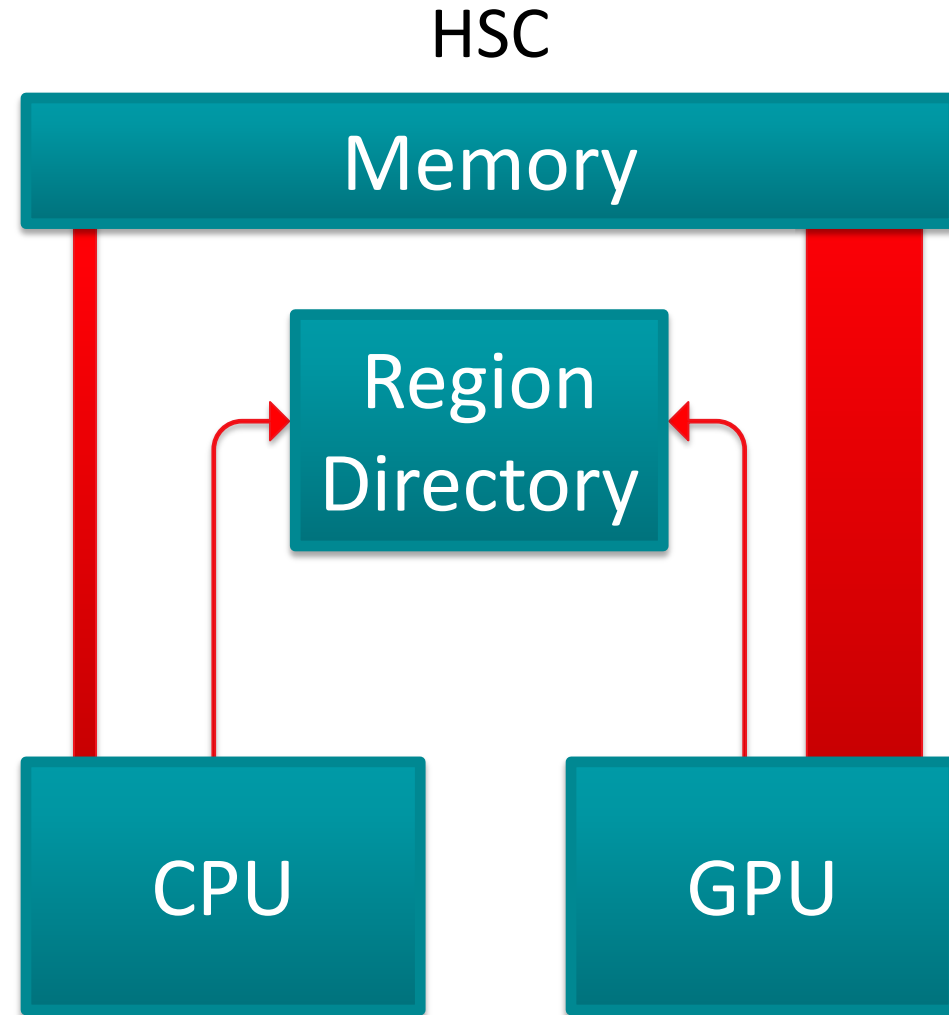


**4x slowdown**

~~1. More than 2 requests per cycle~~

~~2. Needs more than 10,000 MSHRs~~

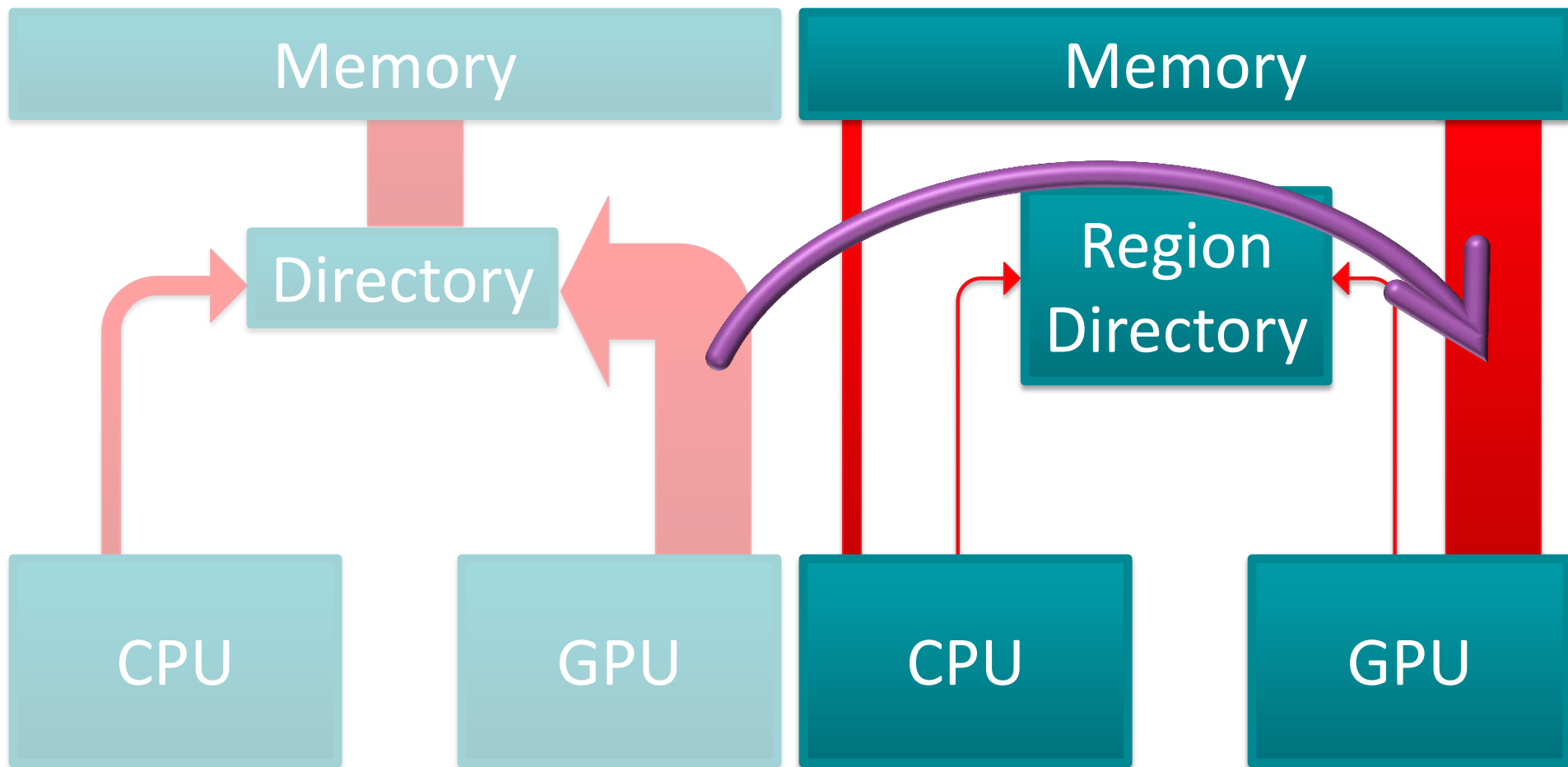
# HETEROGENEOUS SYSTEM COHERENCE AMD



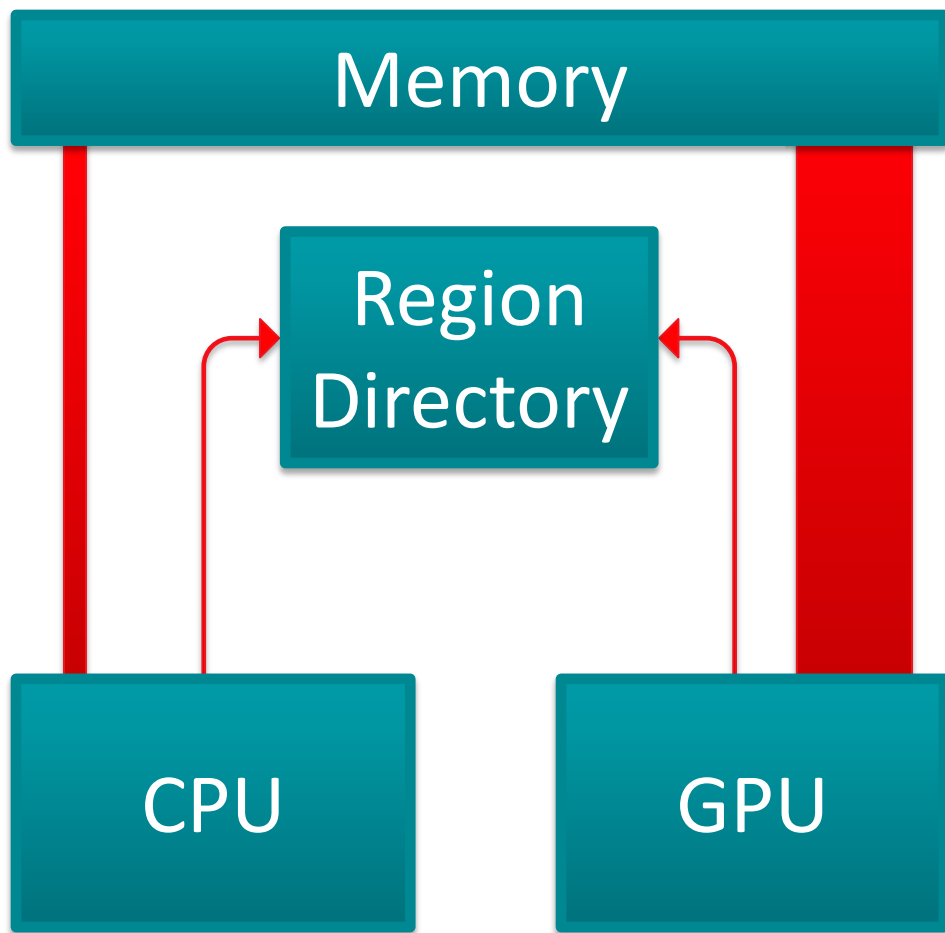
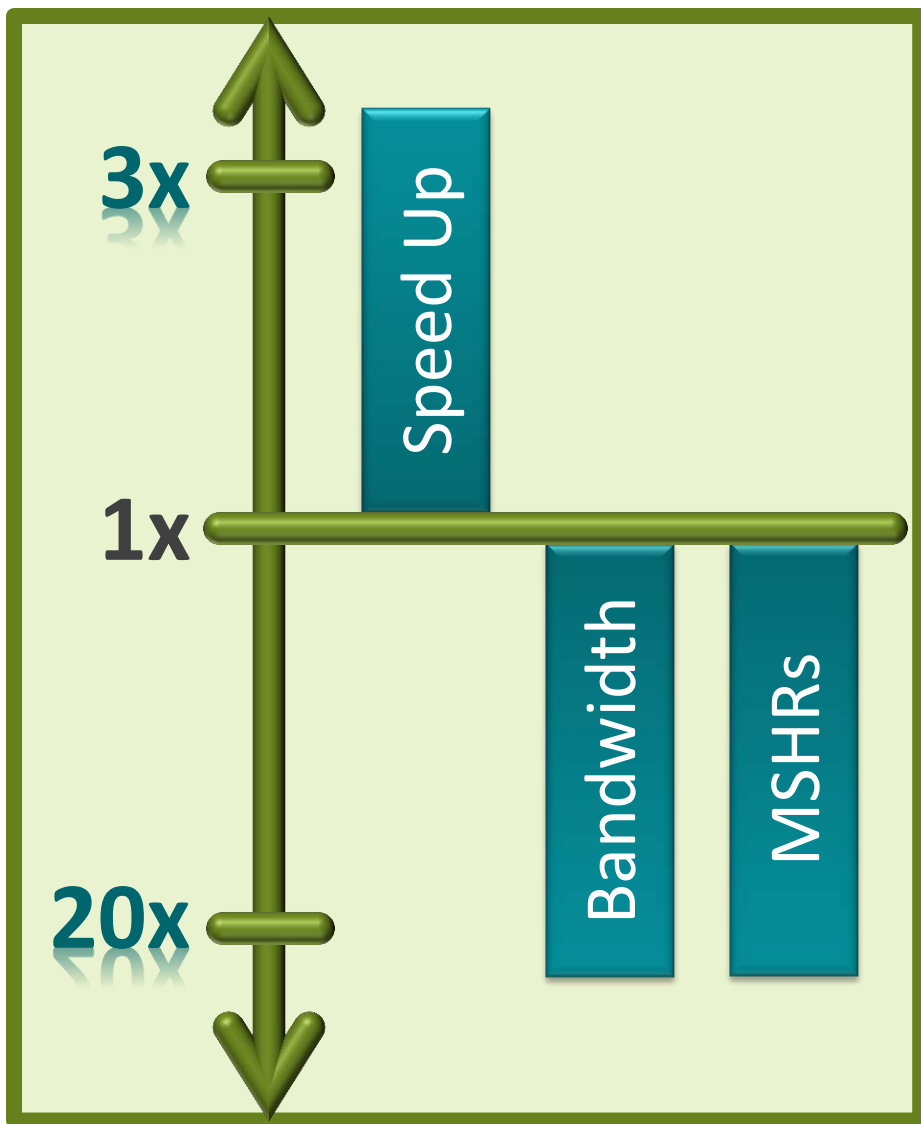
# HETEROGENEOUS SYSTEM COHERENCE AMD

Baseline Directory

HSC



# HETEROGENEOUS SYSTEM COHERENCE AMD



# Meet the Walkers

## Accelerating Index Traversals for In-Memory Databases

Onur Kocberber

Boris Grot, Javier Picorel, Babak Falsafi,  
Kevin Lim, Parthasarathy Ranganathan



Index  
Lookup



Index  
Lookup

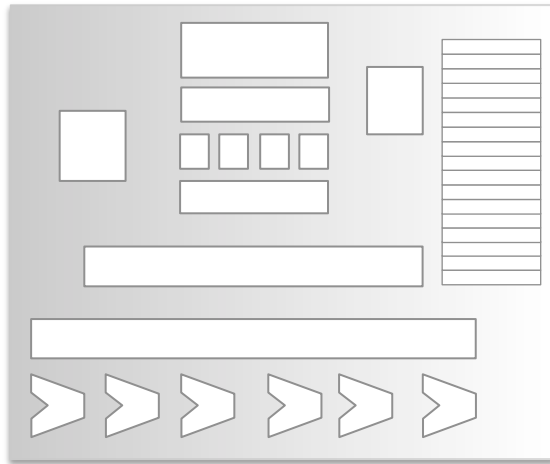


large data

pointer-  
intensive



Index  
Lookup

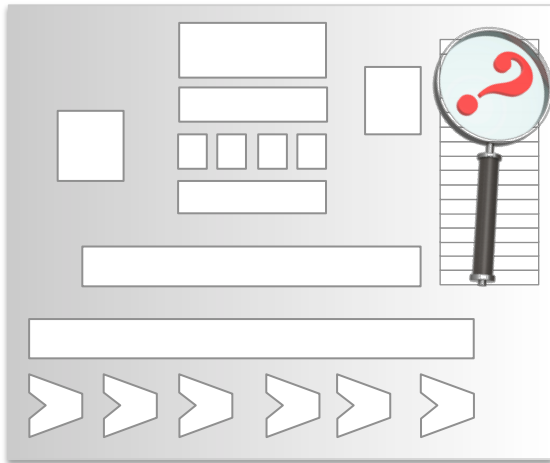


complex

General-purpose  
OoO

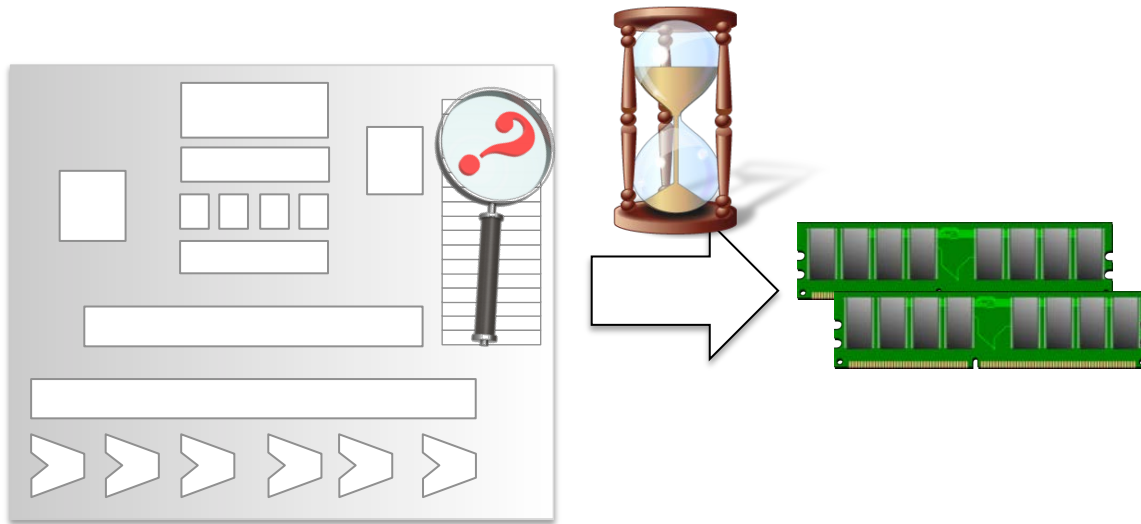


# Index Lookups on General-Purpose OoO



Index  
Lookup

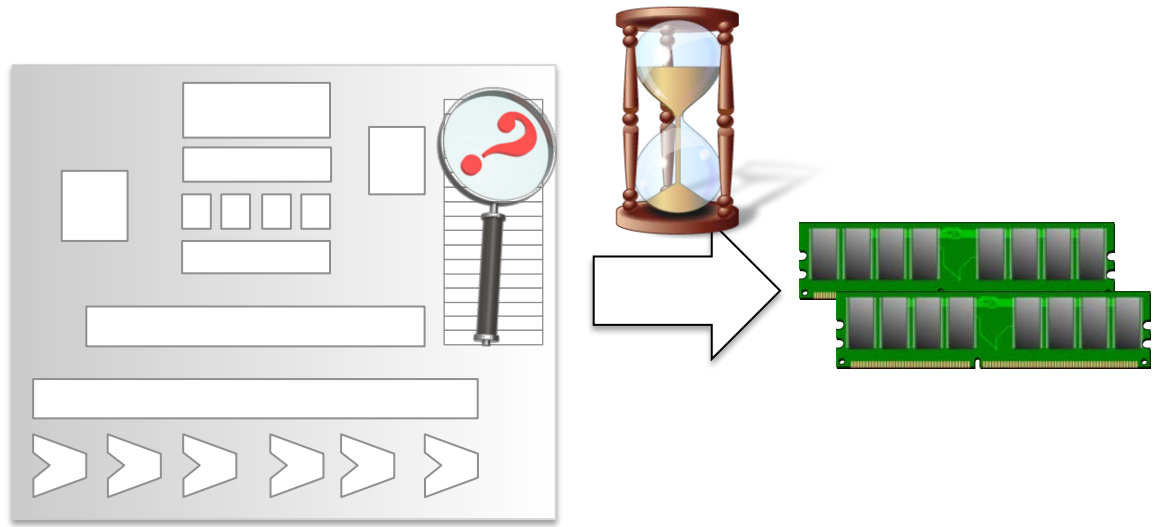
General-purpose  
OoO



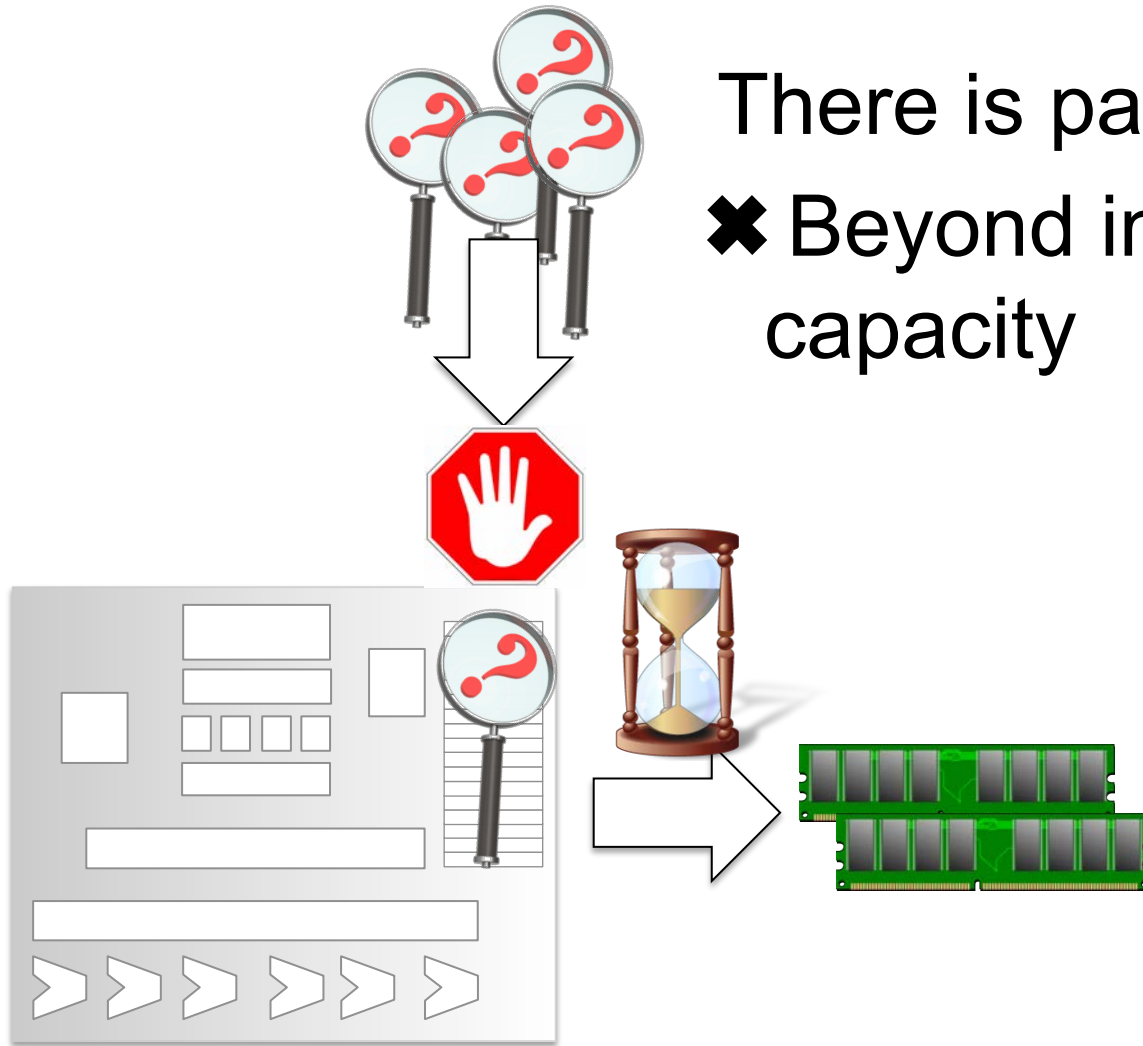
General-purpose  
OoO



There is parallelism!



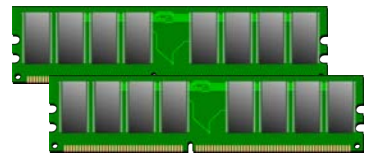
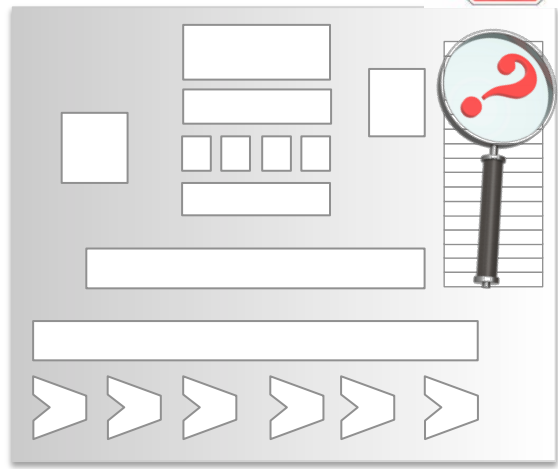
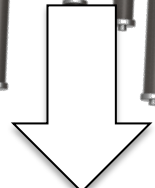
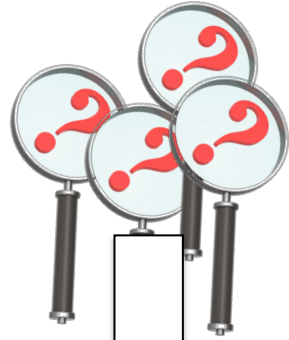
General-purpose  
OoO



There is parallelism!  
✘ Beyond inst. window capacity

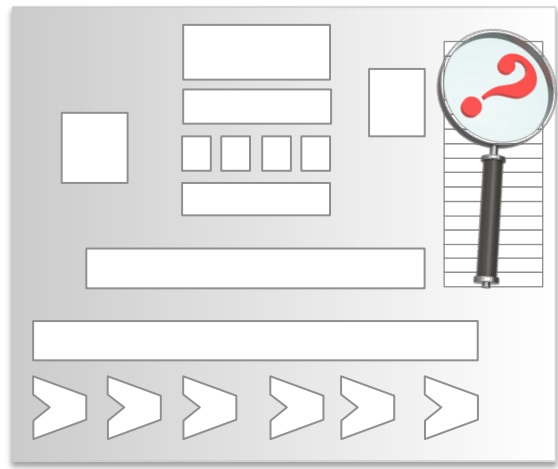
General-purpose  
OoO

Throughput

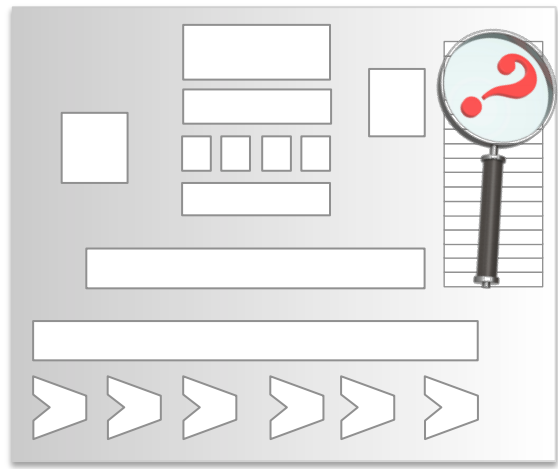


✘ Low throughput

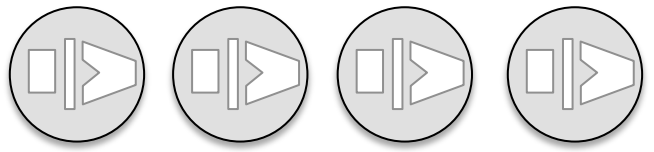
Throughput

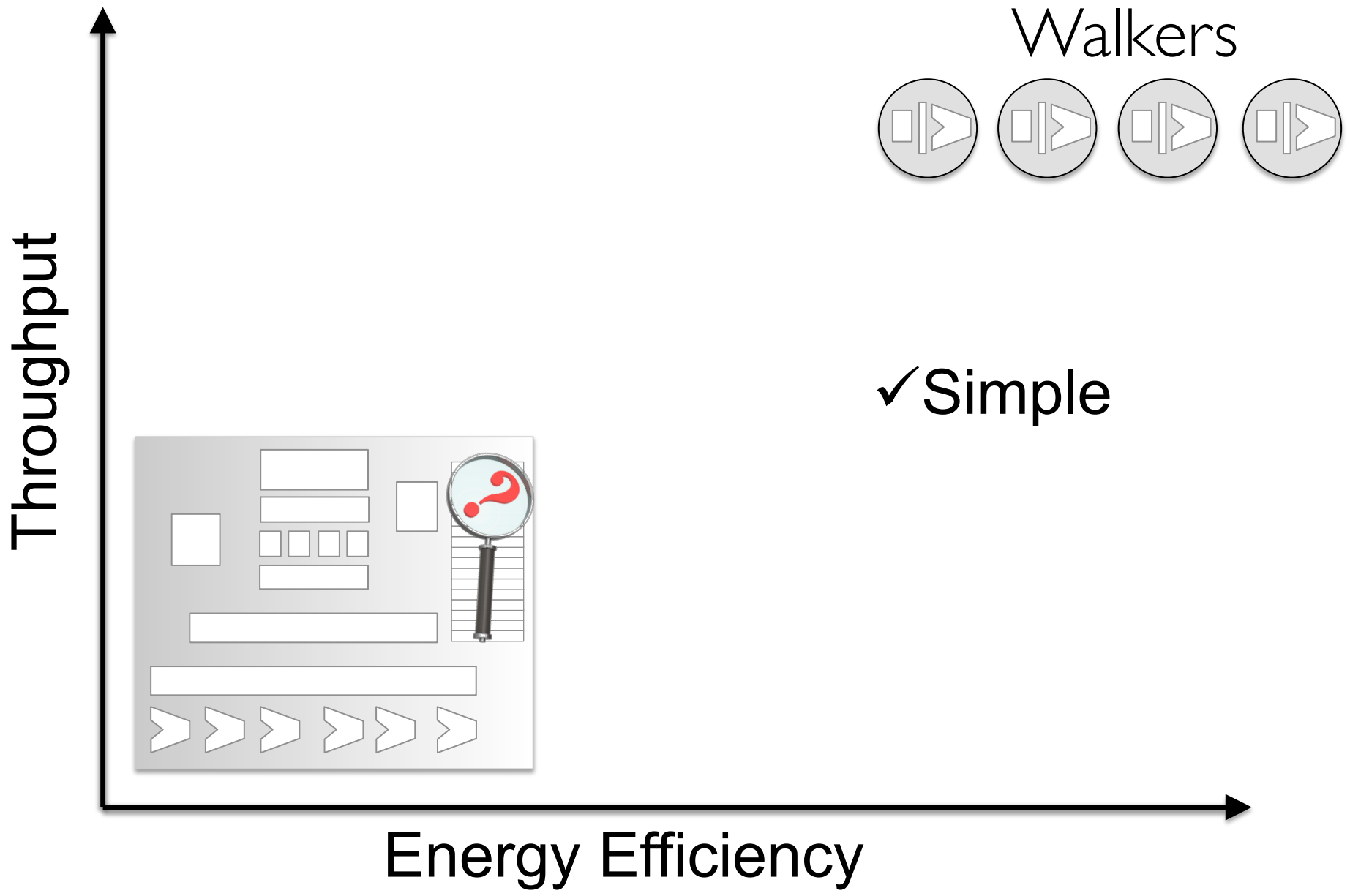


Throughput

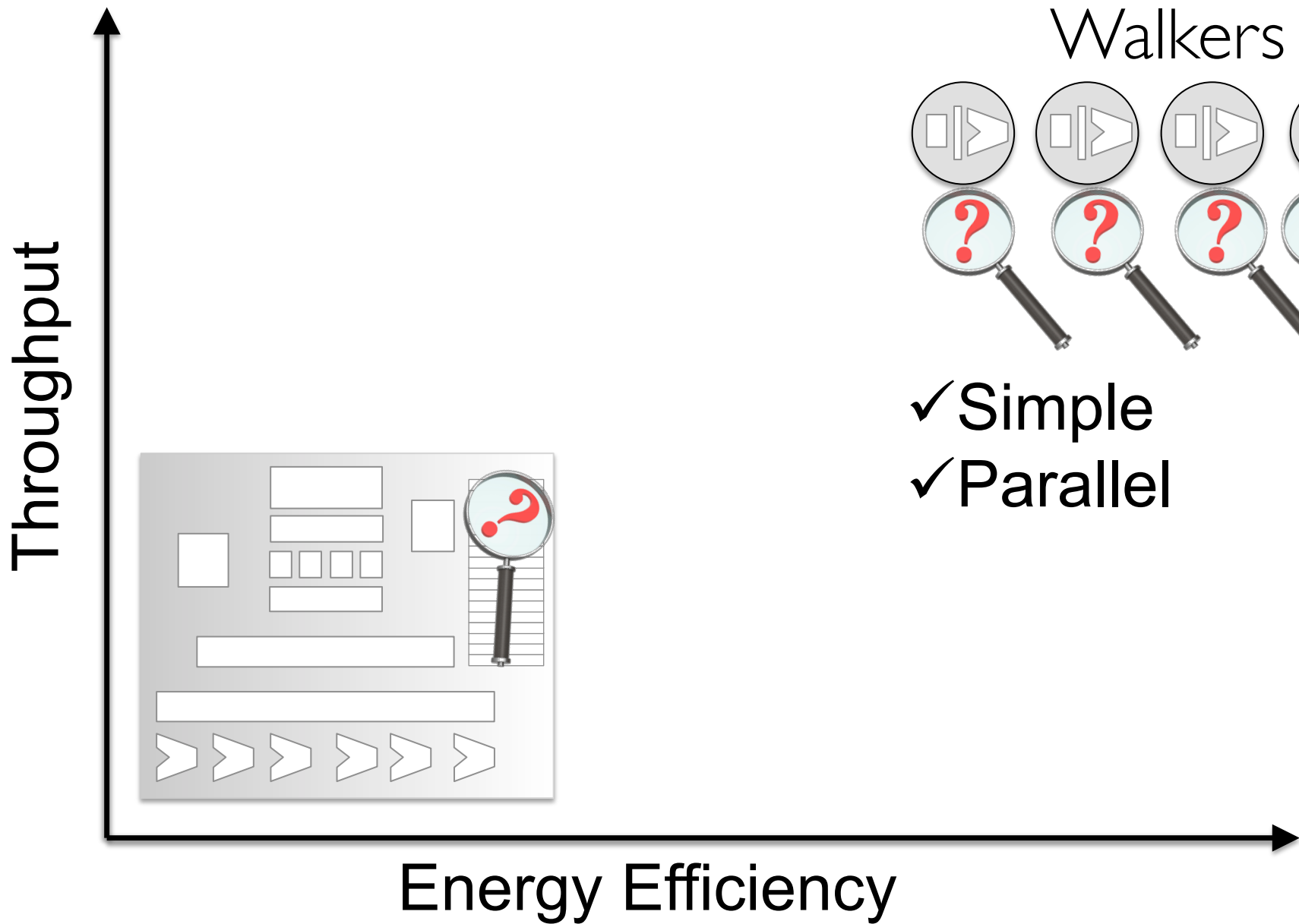


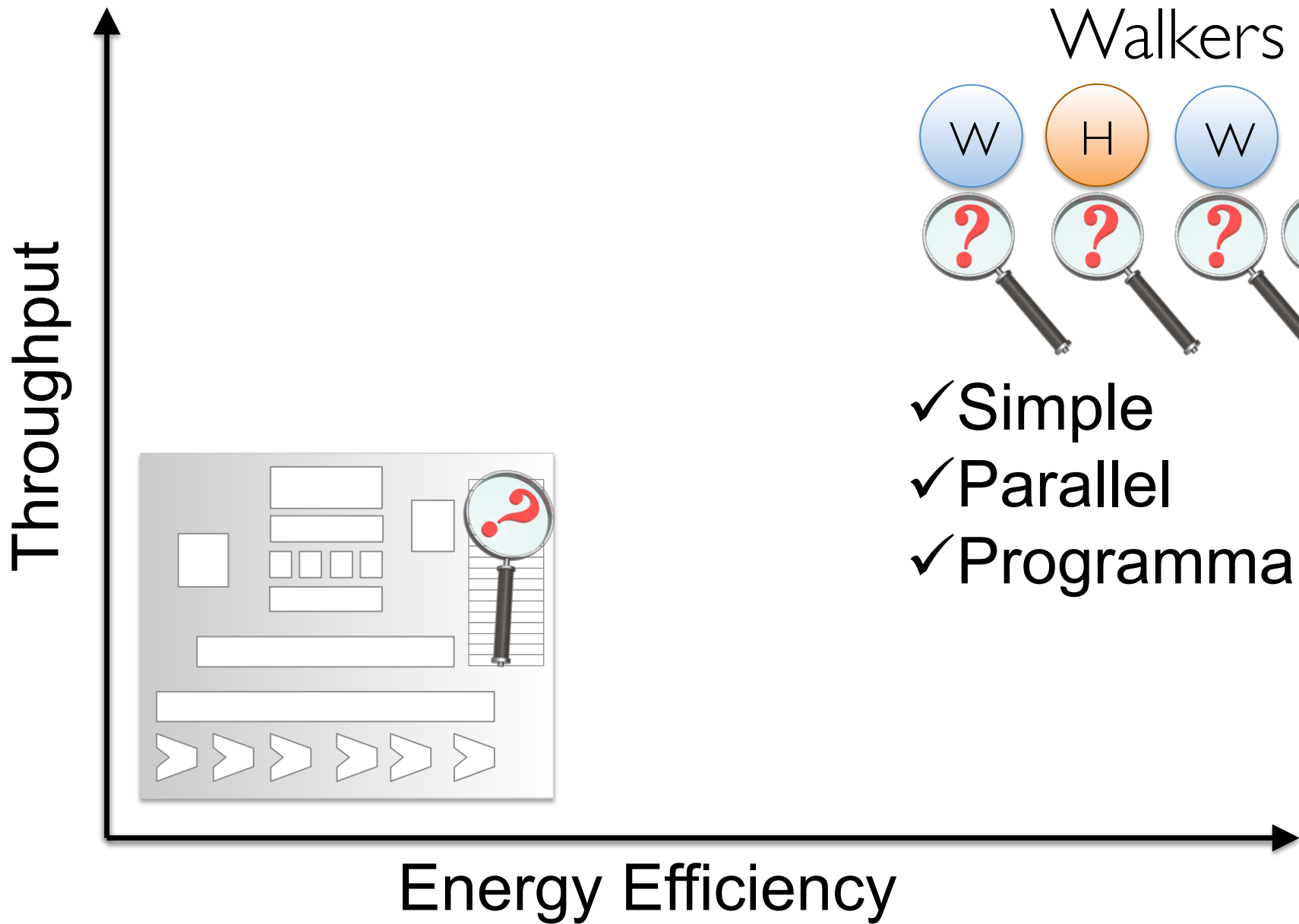
Walkers

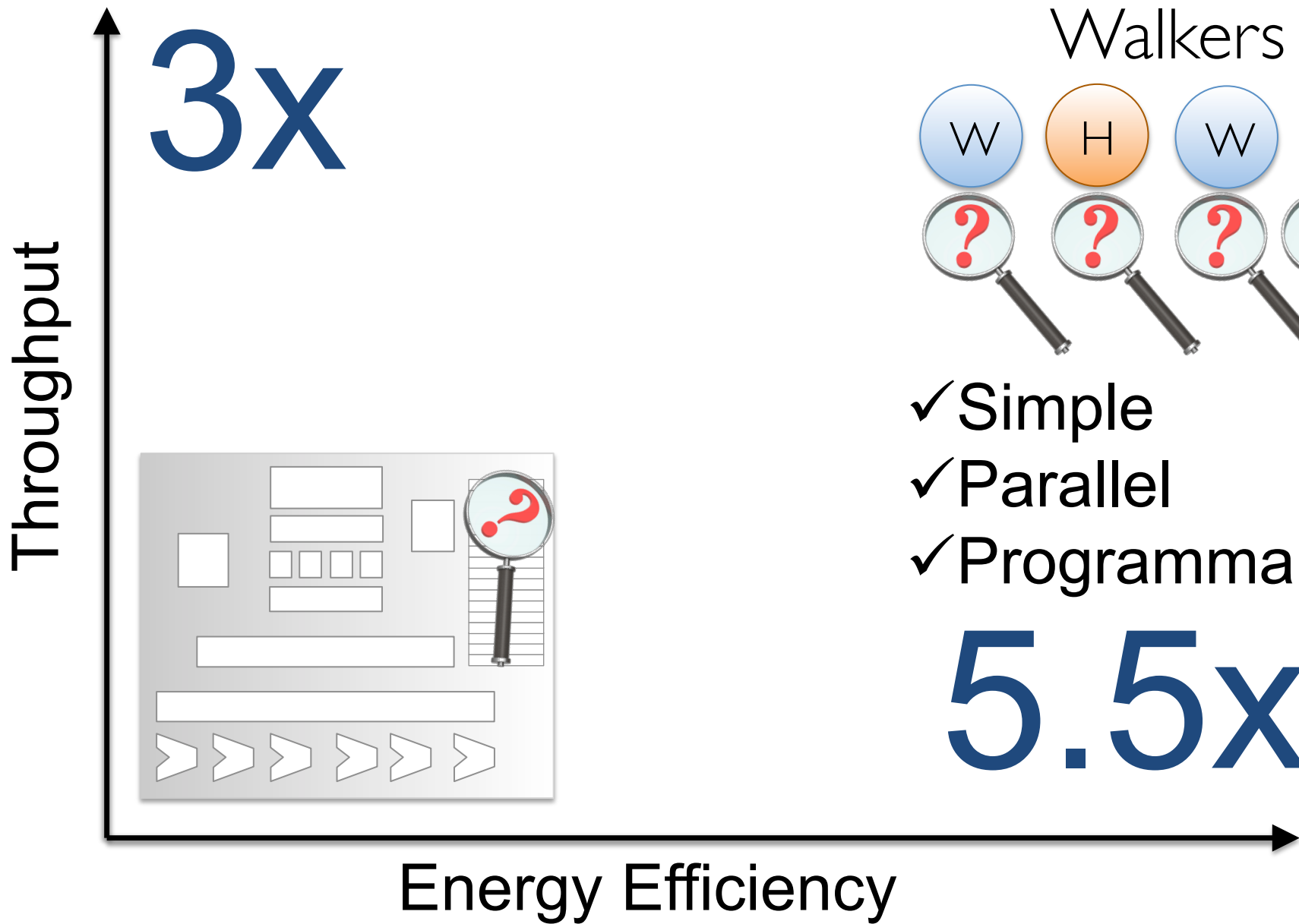




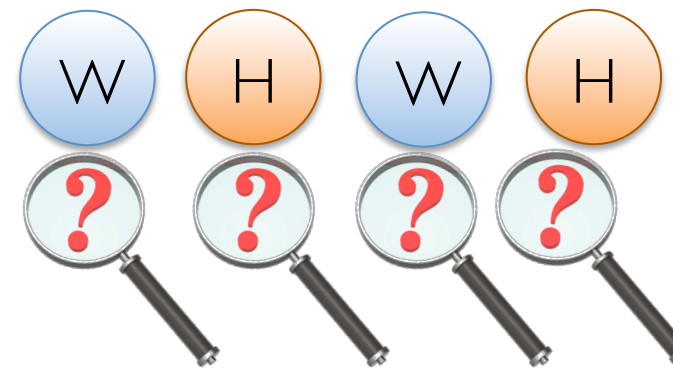








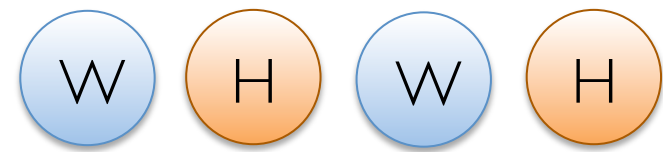
Walkers



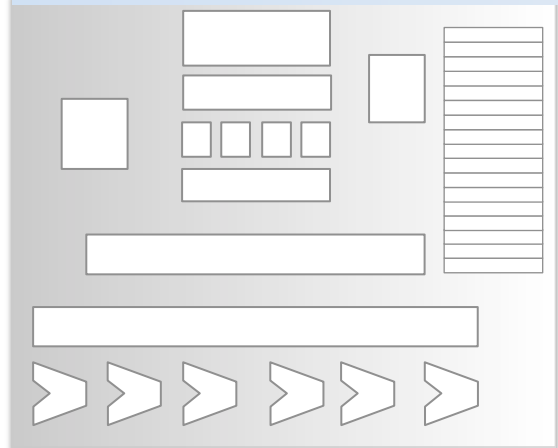
- ✓ Simple
- ✓ Parallel
- ✓ Programmable

3x

Walkers



LAST TALK of the conference  
Wednesday, 12pm



5.5x

Throughput

Energy Efficiency