

# RUBIK: FAST ANALYTICAL POWER MANAGEMENT FOR LATENCY-CRITICAL SYSTEMS

**HARSHAD KASTURE**, DAVIDE BARTOLINI, NATHAN BECKMANN,  
DANIEL SANCHEZ

MICRO 2015

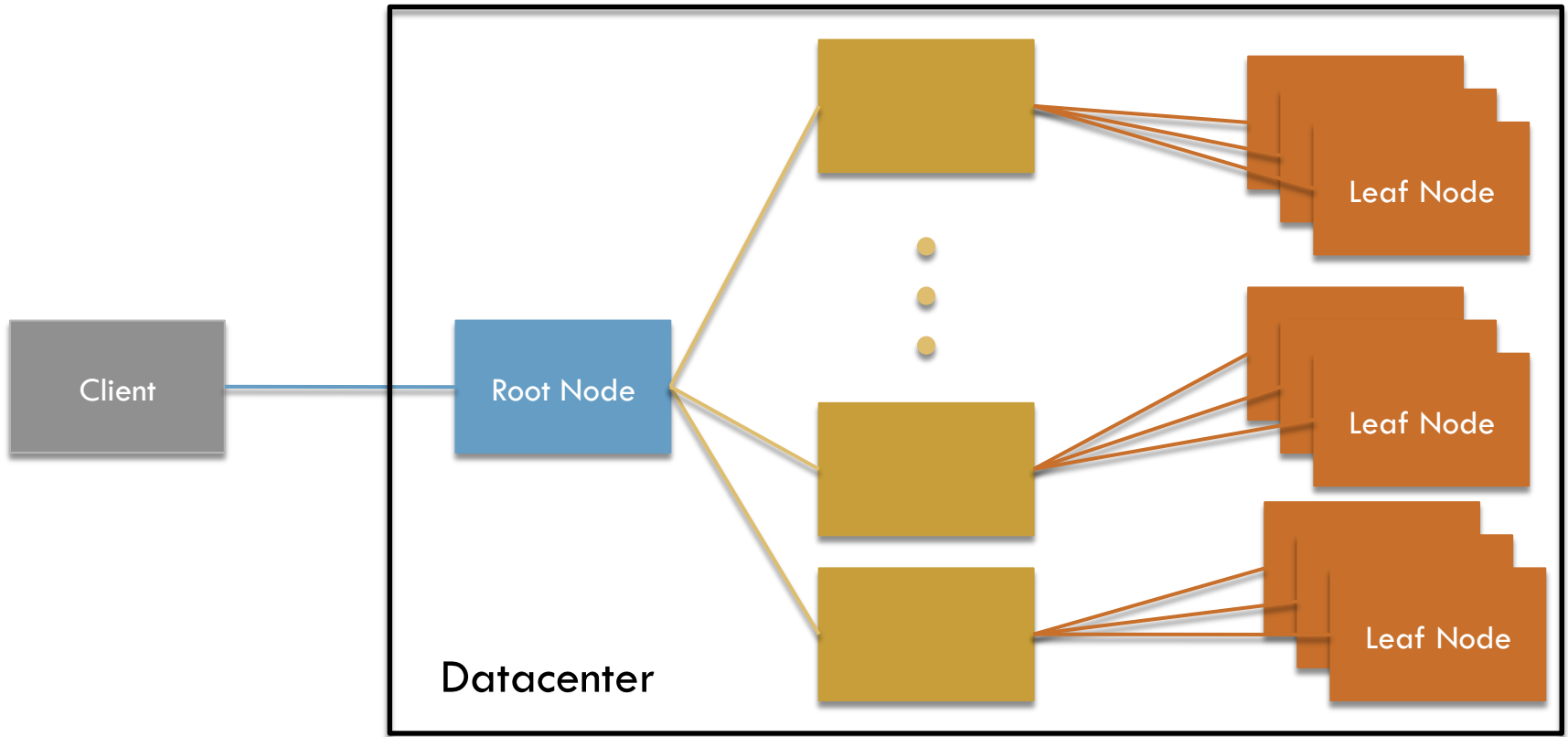


**Massachusetts  
Institute of  
Technology**

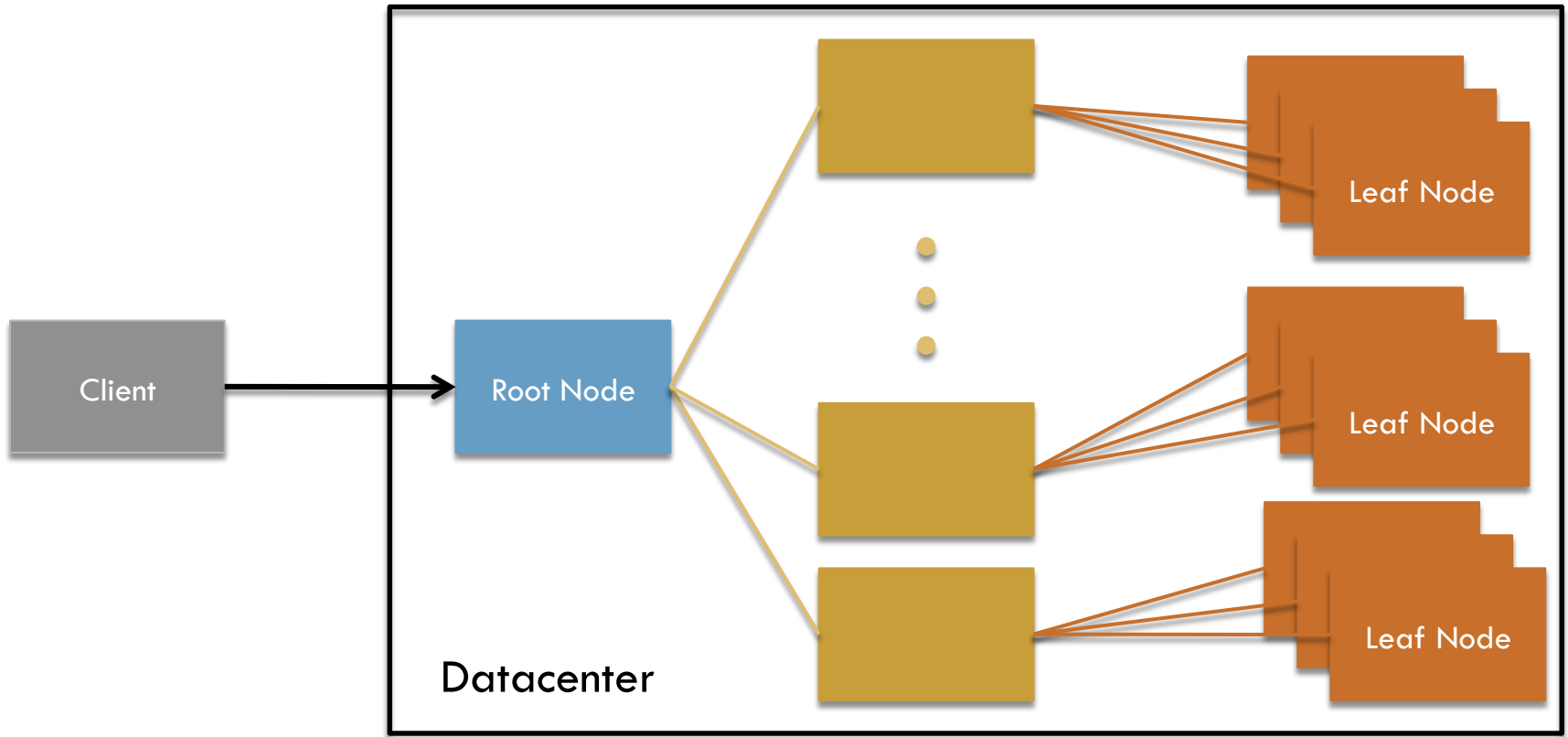


- ❑ Low server utilization in today's datacenters results in resource and energy inefficiency
- ❑ Stringent latency requirements of user-facing services is a major contributing factor
- ❑ Power management for these services is challenging
  - ▣ Strict requirements on tail latency
  - ▣ Inherent variability in request arrival and service times
- ❑ Rubik uses statistical modeling to adapt to short-term variations
  - ▣ Respond to abrupt load changes
  - ▣ Improve power efficiency
  - ▣ Allow colocation of latency-critical and batch applications

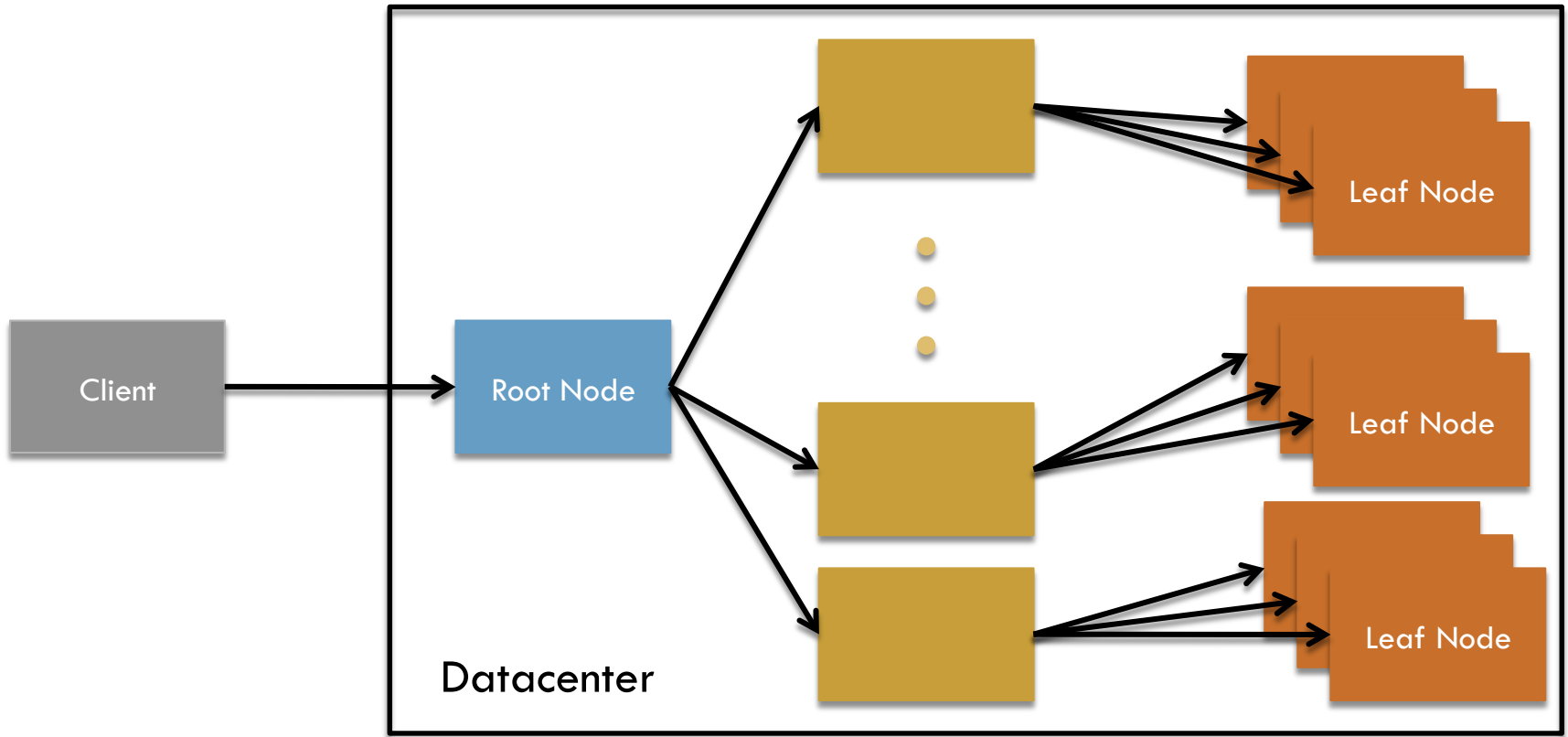
# Understanding Latency-Critical Applications<sub>3</sub>



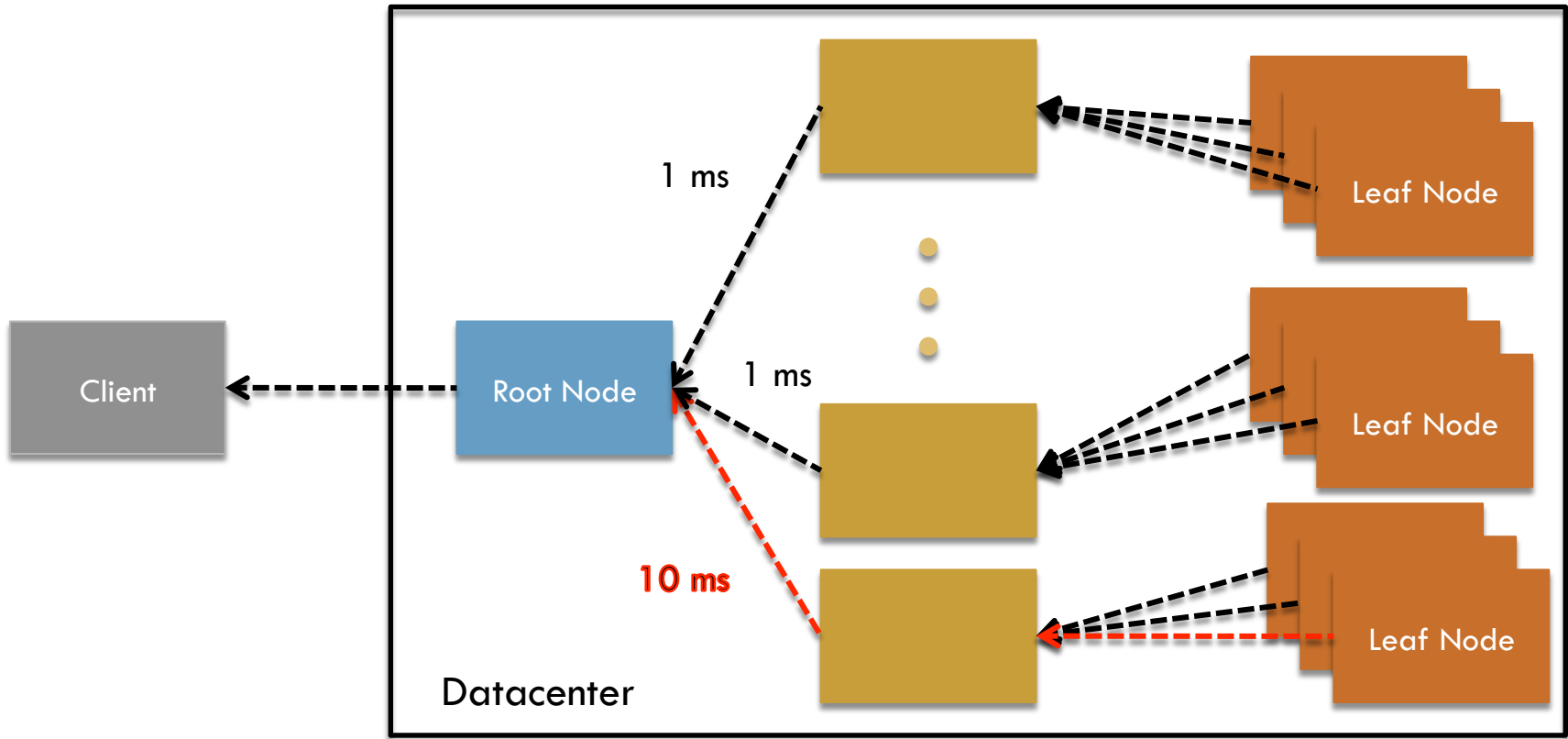
# Understanding Latency-Critical Applications<sub>4</sub>



# Understanding Latency-Critical Applications<sub>5</sub>



# Understanding Latency-Critical Applications <sub>6</sub>



- The few slowest responses determine user-perceived latency
  - ▣ Tail latency (e.g., 95<sup>th</sup> / 99<sup>th</sup> percentile), not mean latency, determines performance

# Prior Schemes Fall Short

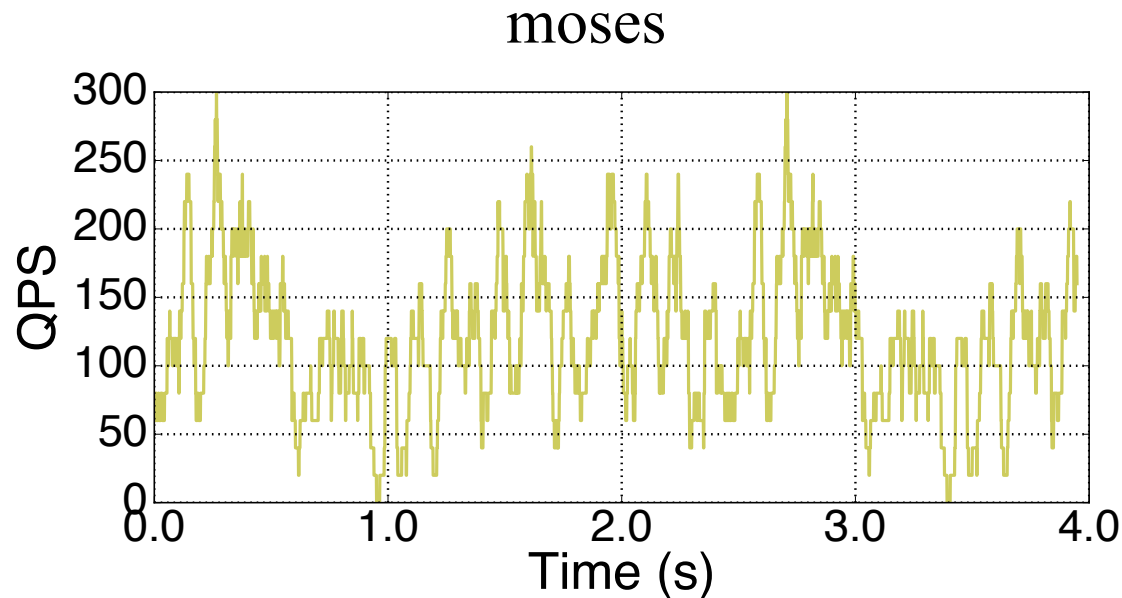
7

- Traditional DVFS schemes (cpufreq, TurboBoost...)
  - ▣ React to coarse grained metrics like processor utilization, oblivious to short-term performance requirements
- Power management for embedded systems (PACE, GRACE...)
  - ▣ Do not consider queuing
- Schemes designed specifically for latency-critical systems (PEGASUS [Lo ISCA'14], Adrenaline [Hsu HPCA'15])
  - ▣ Rely on application-specific heuristics
  - ▣ Too conservative

# Insight 1: Short-Term Load Variations

8

- Latency-critical applications have significant short-term load variations



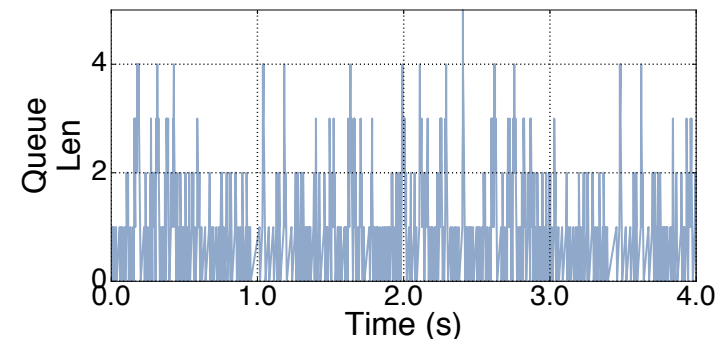
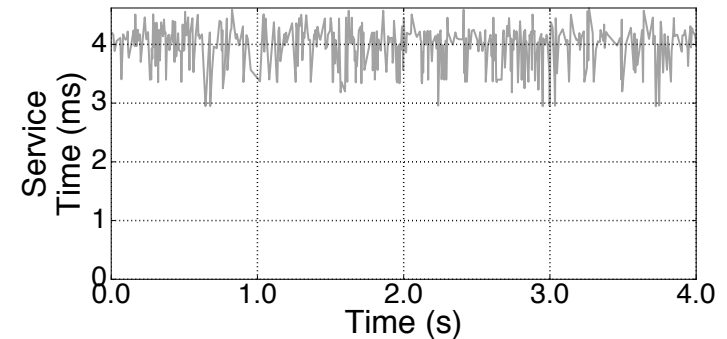
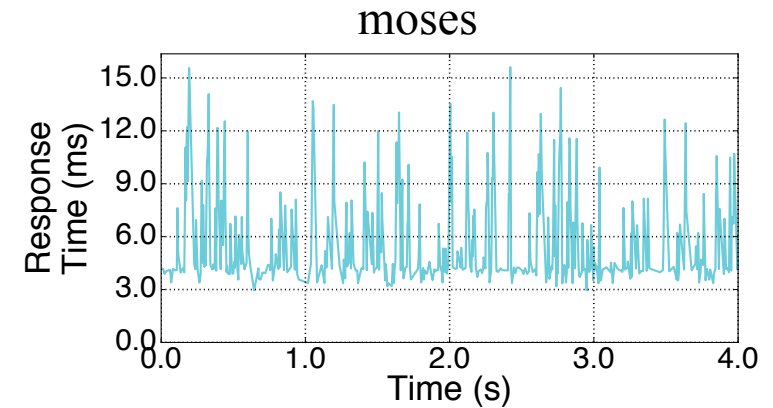
- PEGASUS [Lo ISCA'14] uses feedback control to adapt frequency setting to diurnal load variations
  - ▣ Deduce server load from observed request latency
  - ▣ Cannot adapt to short-term variations



# Insight 2: Queuing Matters!

9

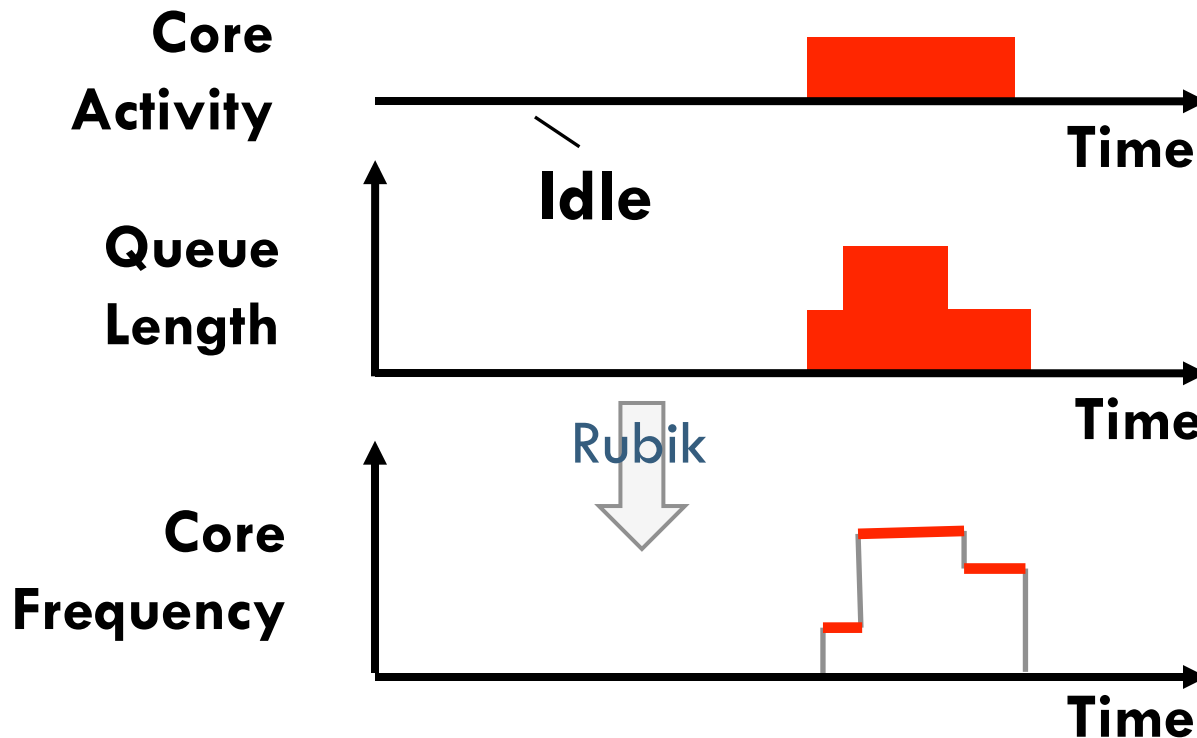
- Tail latency is often determined by queuing, not the length of individual requests
- Adrenaline [Hsu HPCA'15] uses application-level hints to distinguish long requests from short ones
  - ▣ Long requests *boosted* (sped up)
  - ▣ Frequency settings must be conservative to handle queuing



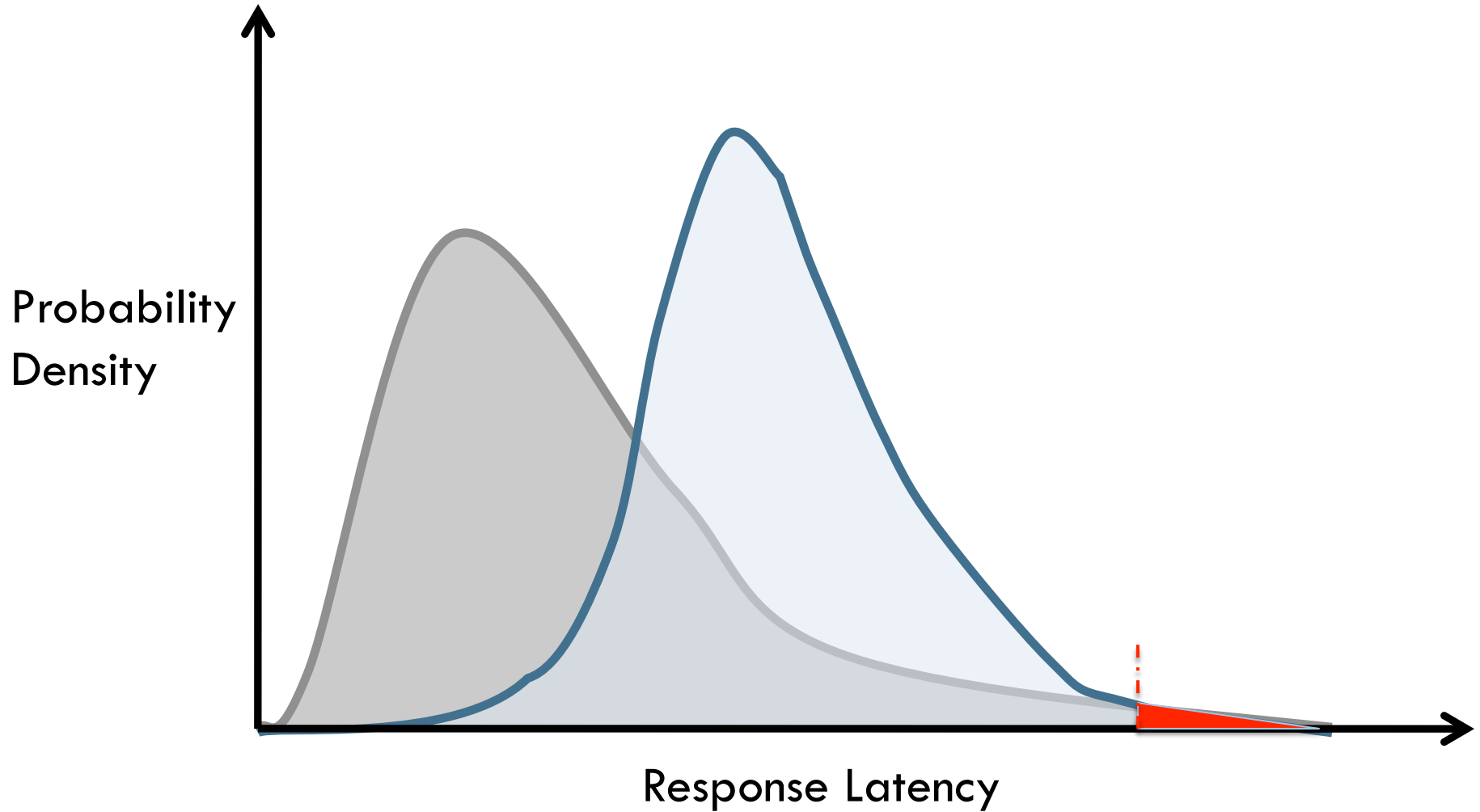
# Rubik Overview

10

- Use queue length as a measure of instantaneous system load
- Update frequency whenever queue length changes
  - ▣ Adapt to short-term load variations



# Goal: Reshaping Latency Distribution 11

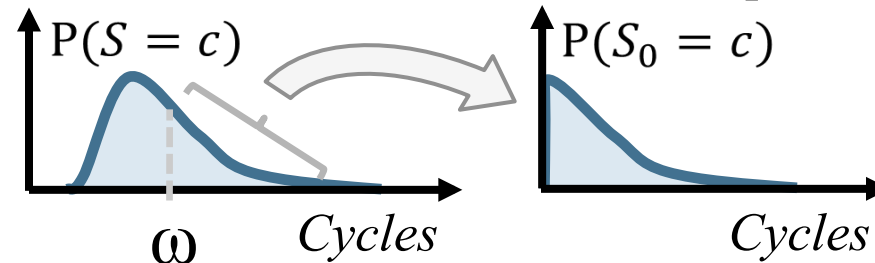


# Key Factors in Setting Frequencies

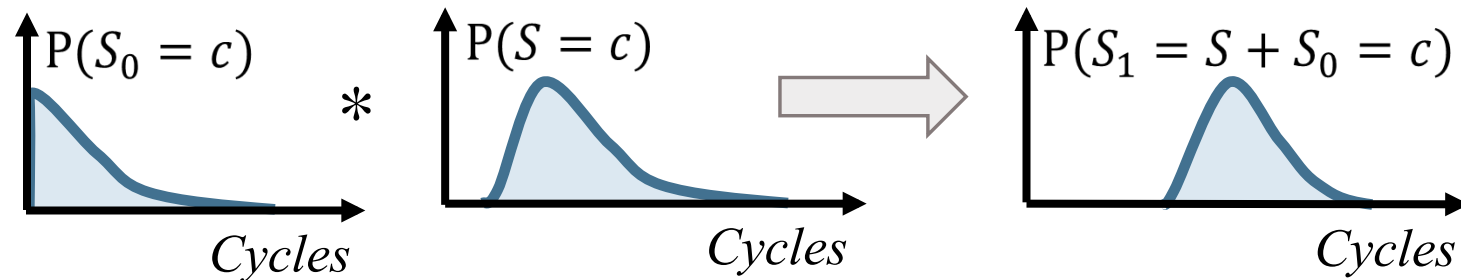
12

- Distribution of cycle requirements of individual requests
  - ▣ Larger variance → more conservative frequency setting
  
- How long has a request spent in the queue?
  - ▣ Longer wait times → higher frequency
  
- How many requests are queued waiting for service
  - ▣ Longer queues → higher frequency

$$P[S_0 = c] = P[S = c + \omega \mid S > \omega] = \frac{P[S = c + \omega]}{P[S > \omega]}$$



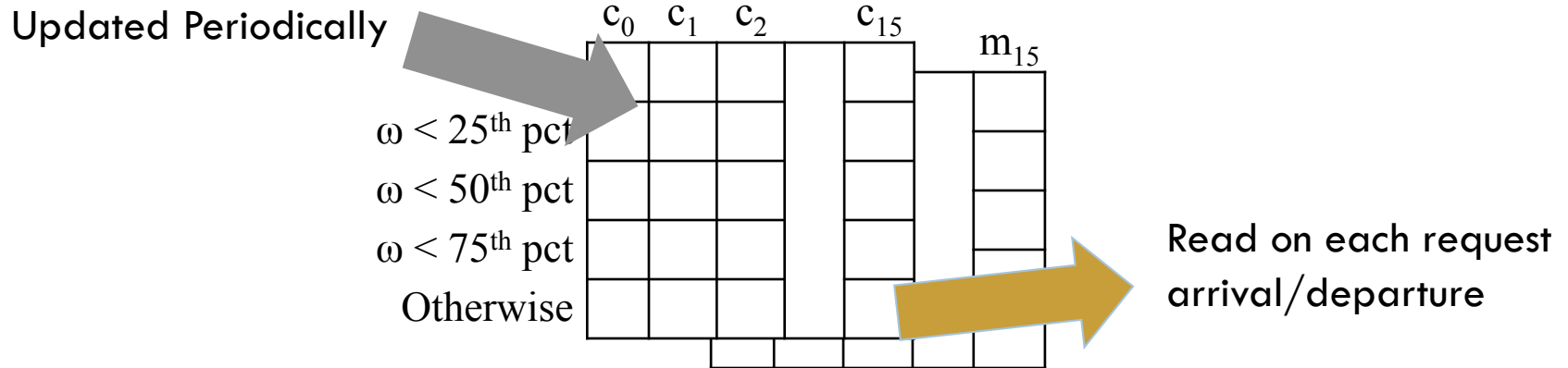
$$P_{S_i} = P_{S_{i-1}} * P_S = P_{S_0} * \overbrace{P_S * P_S * \dots * P_S}^{i \text{ times}}$$



$$f \geq \max_{i=0 \dots N} \frac{c_i}{L - (t_i + m_i)}$$

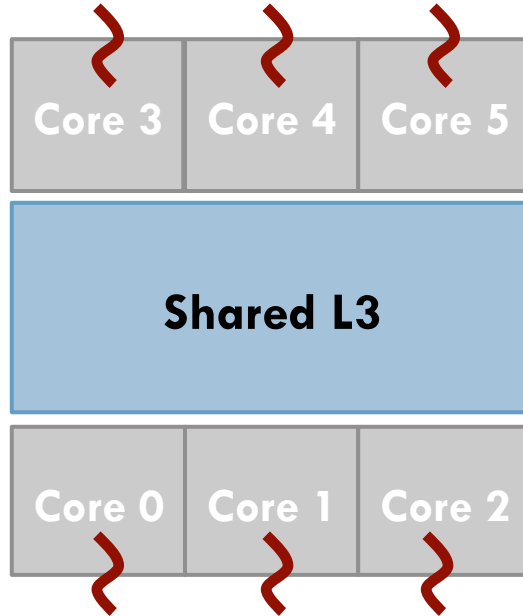
- Pre-computed tables store most of the required quantities

## Target Tail Tables



- Table contents are independent of system load!
- Implemented as a software runtime
  - ▣ Hardware support: fast, per-core DVFS, performance counters for CPI stacks

- Microarchitectural simulations using zsim
  - ▣ Power model tuned to a real system



- Westmere-like OOO cores
- Fast per-core DVFS
- CPI stack counters
- Pin threads to cores

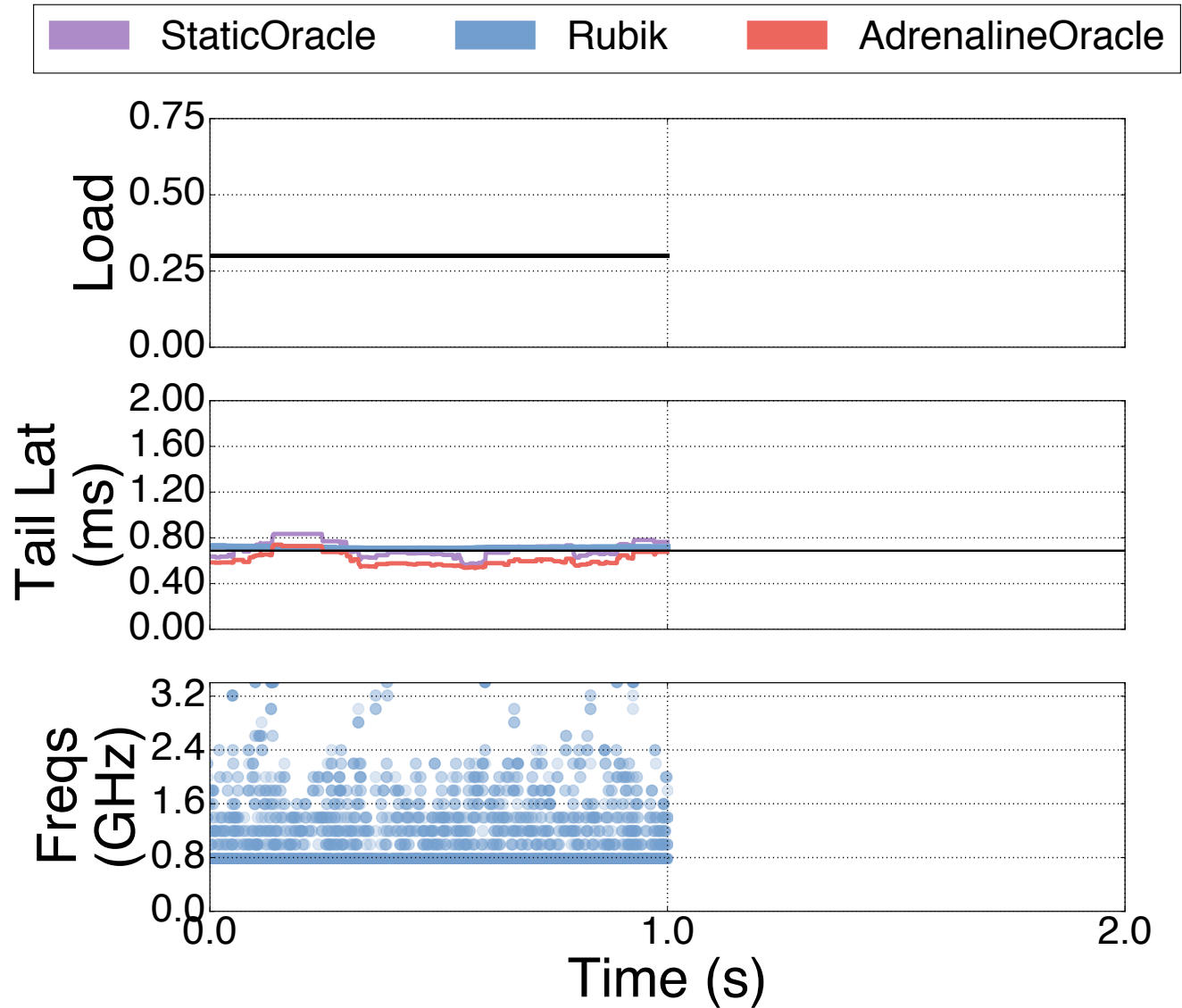
- Compare Rubik against two oracular schemes:
  - ▣ StaticOracle: Pick the lowest static frequency that meets latency targets for a given request trace
  - ▣ AdrenalineOracle: Assume oracular knowledge of long and short requests, use offline training to pick frequencies for each

- Five diverse latency-critical applications
  - xapian (search engine)
  - masstree (in-memory key-value store)
  - moses (statistical machine translation)
  - shore-mt (OLTP)
  - specjbb (java middleware)
  
- For each application, latency target set at the tail latency achieved at nominal frequency (2.4 GHz) at 50% utilization



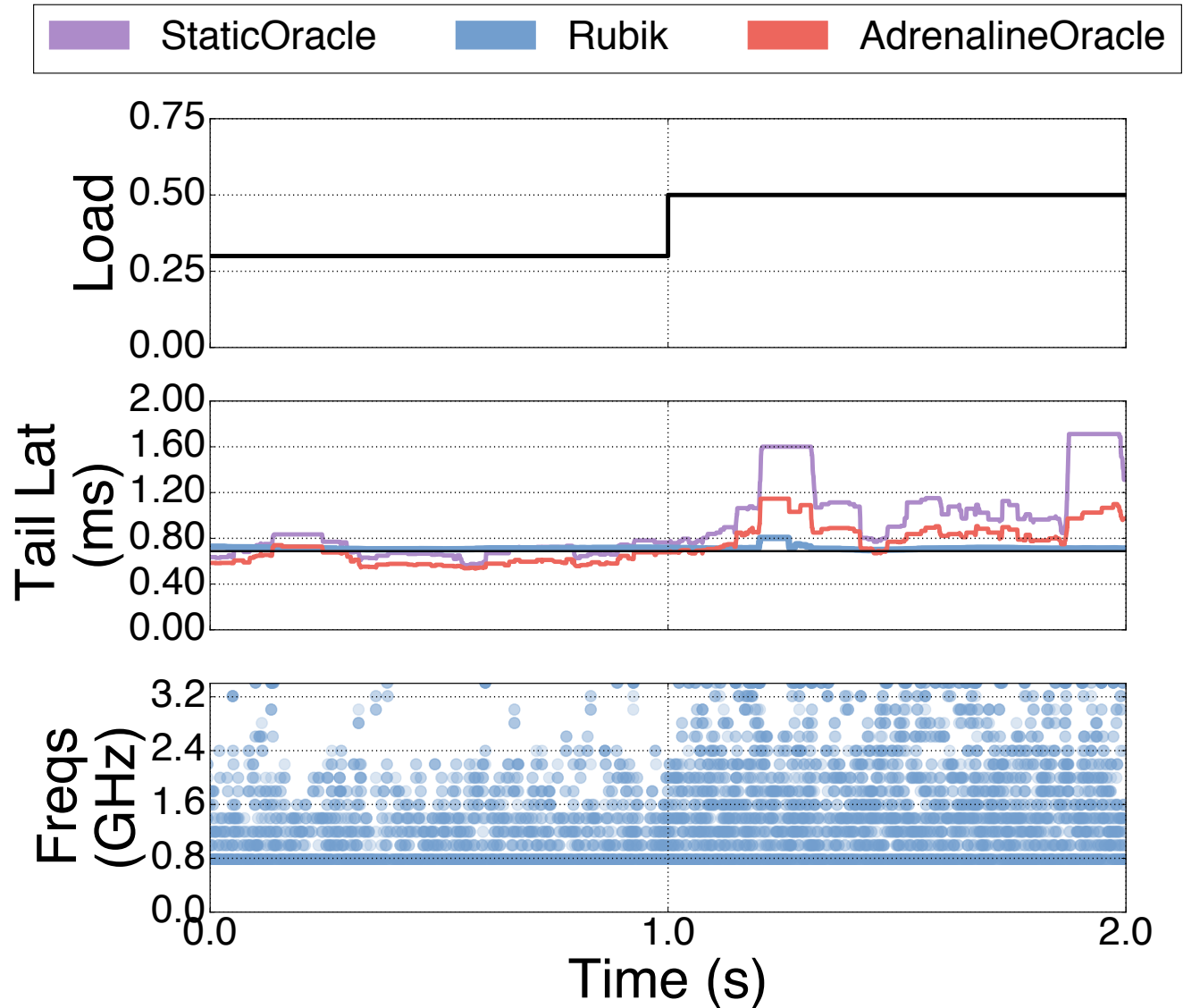
# Tail Latency

17



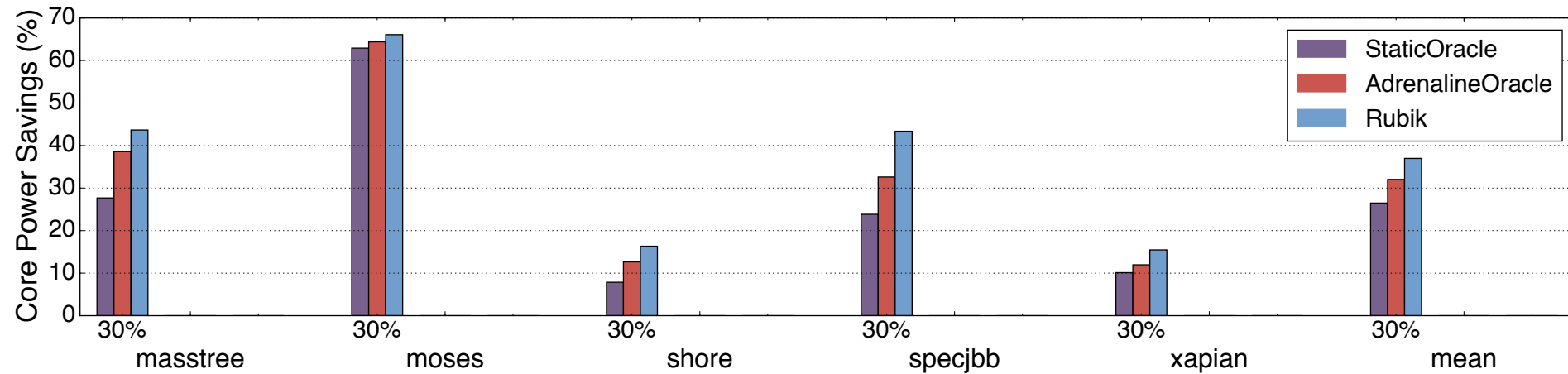
# Tail Latency

18



# Core Power Savings

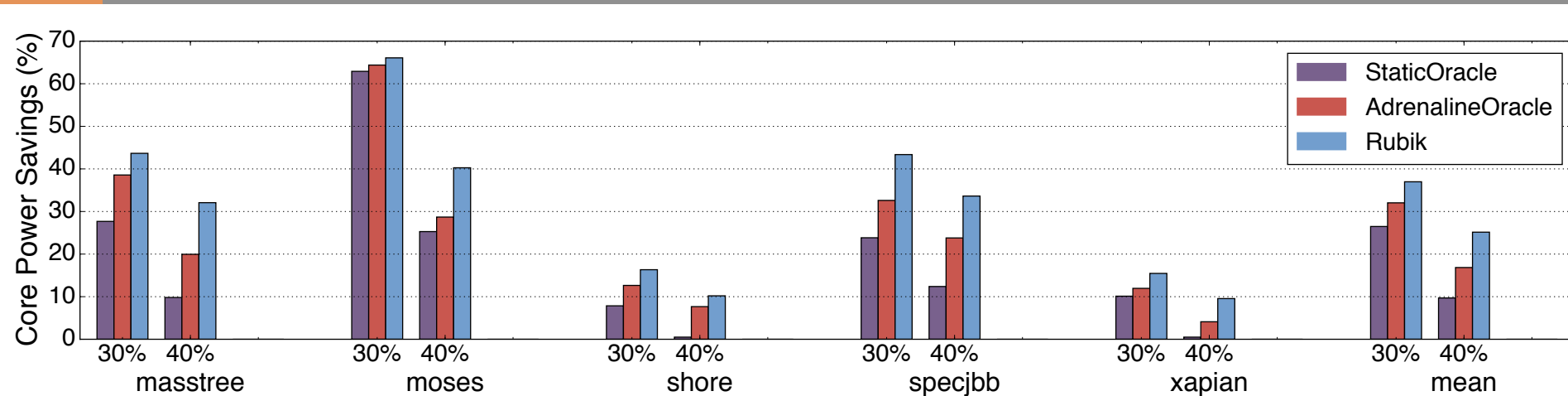
19



- All three schemes save significant power at low utilization
  - ▣ Rubik performs best, reducing core power by up to 66%

# Core Power Savings

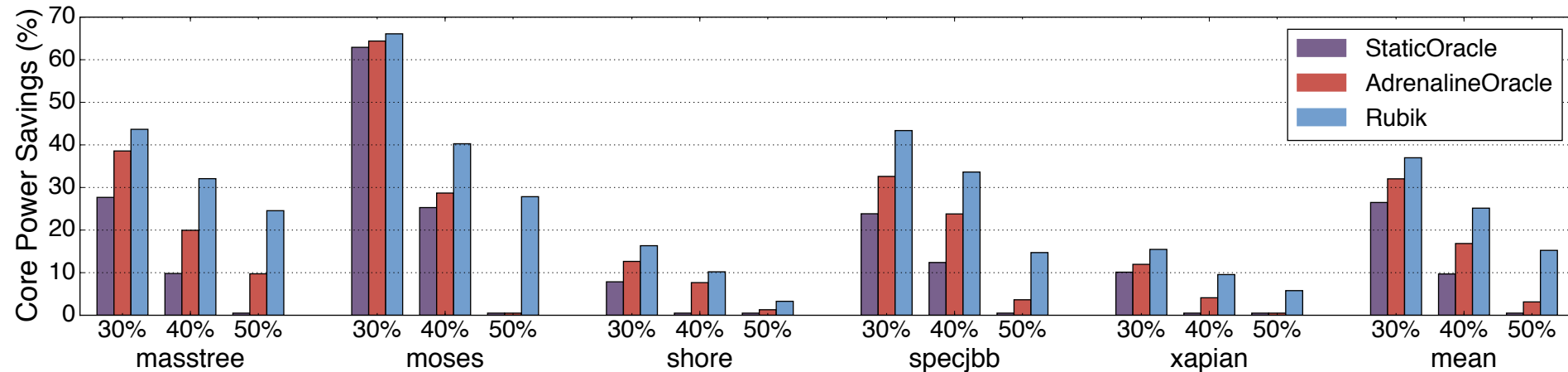
20



- All three schemes save significant power at low utilization
  - ▣ Rubik performs best, reducing core power by up to 66%
- Rubik's relative savings *increase* as short-term adaptation becomes more important

# Core Power Savings

21

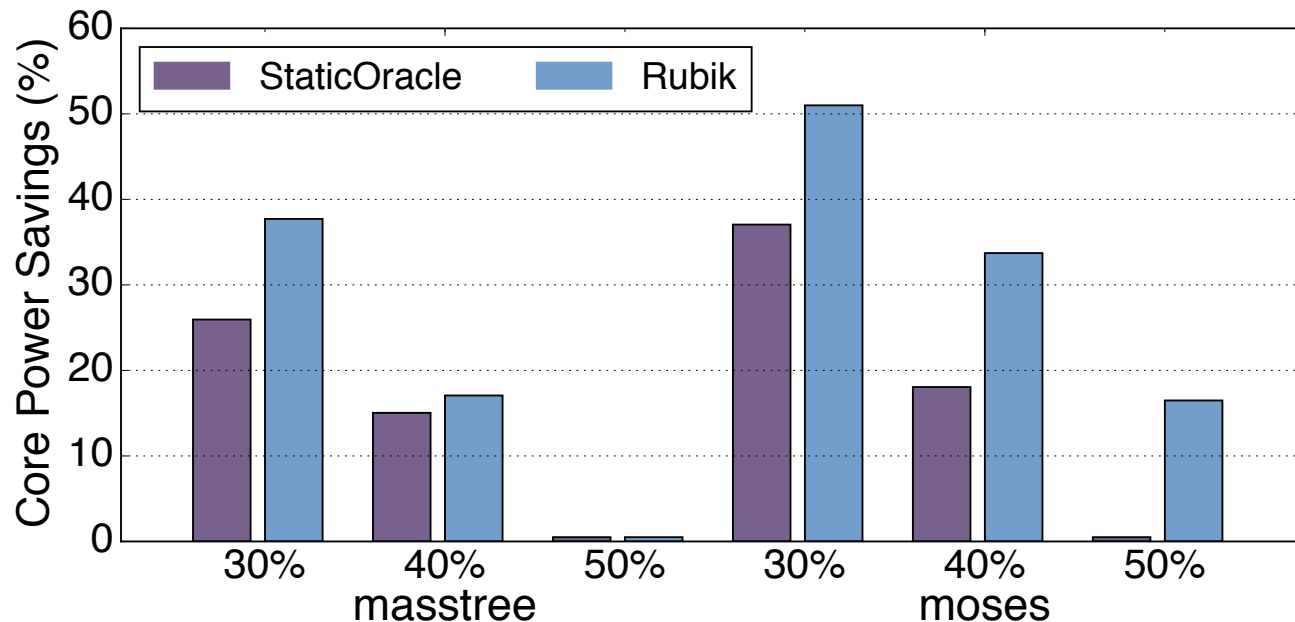


- All three schemes save significant power at low utilization
  - ▣ Rubik performs best, reducing core power by up to 66%
- Rubik's relative savings *increase* as short-term adaptation becomes more important
- Rubik saves significant power even at high utilization
  - ▣ 17% on average, and up to 34%

# Real Machine Power Savings

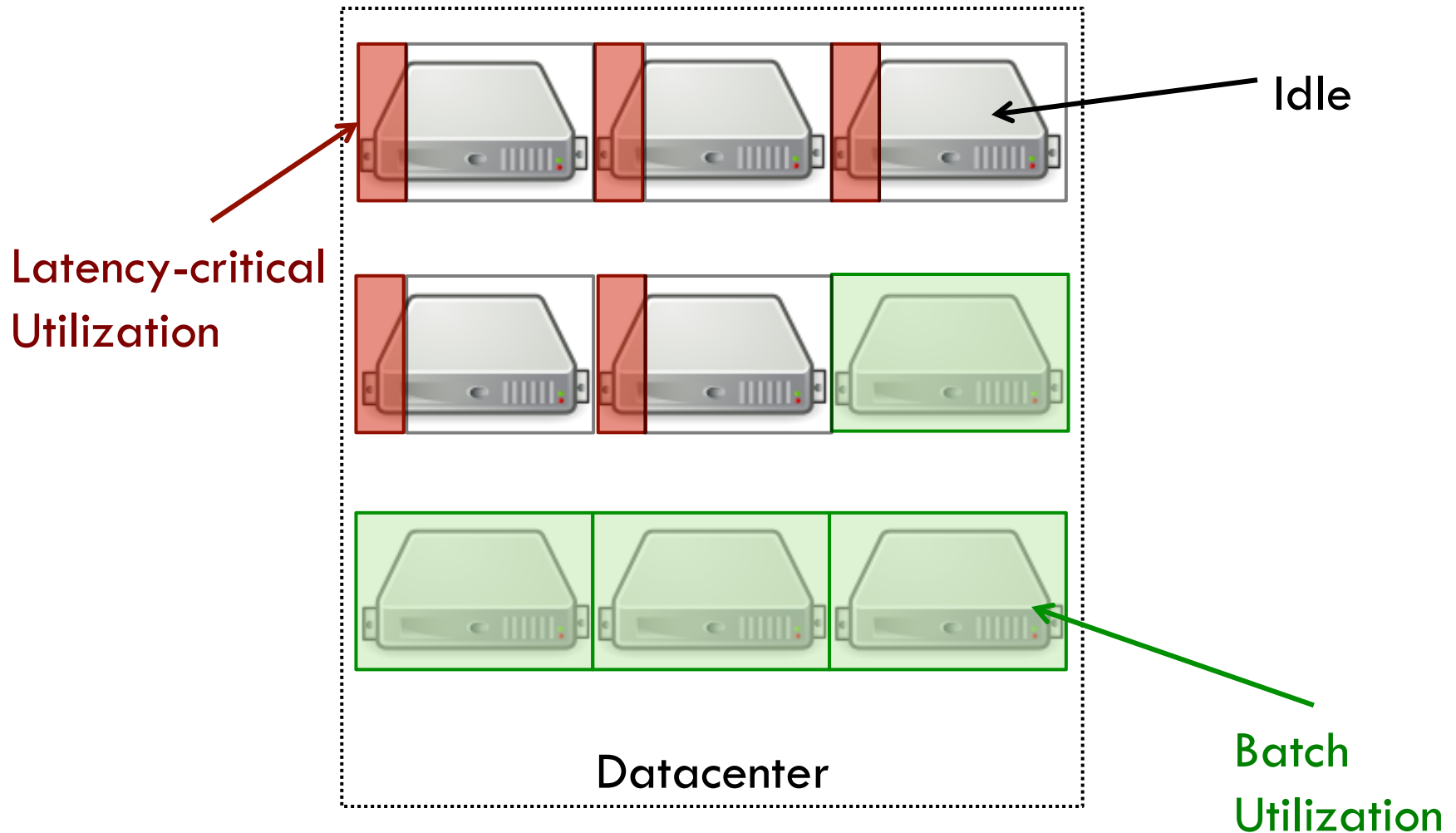
22

- V/F transition latencies of  $>100\ \mu\text{s}$  even with integrated voltage controllers
  - ▣ Likely due to inefficiencies in firmware
- Rubik successfully adapts to higher V/F transition latencies



# Static Power Limits Efficiency

23

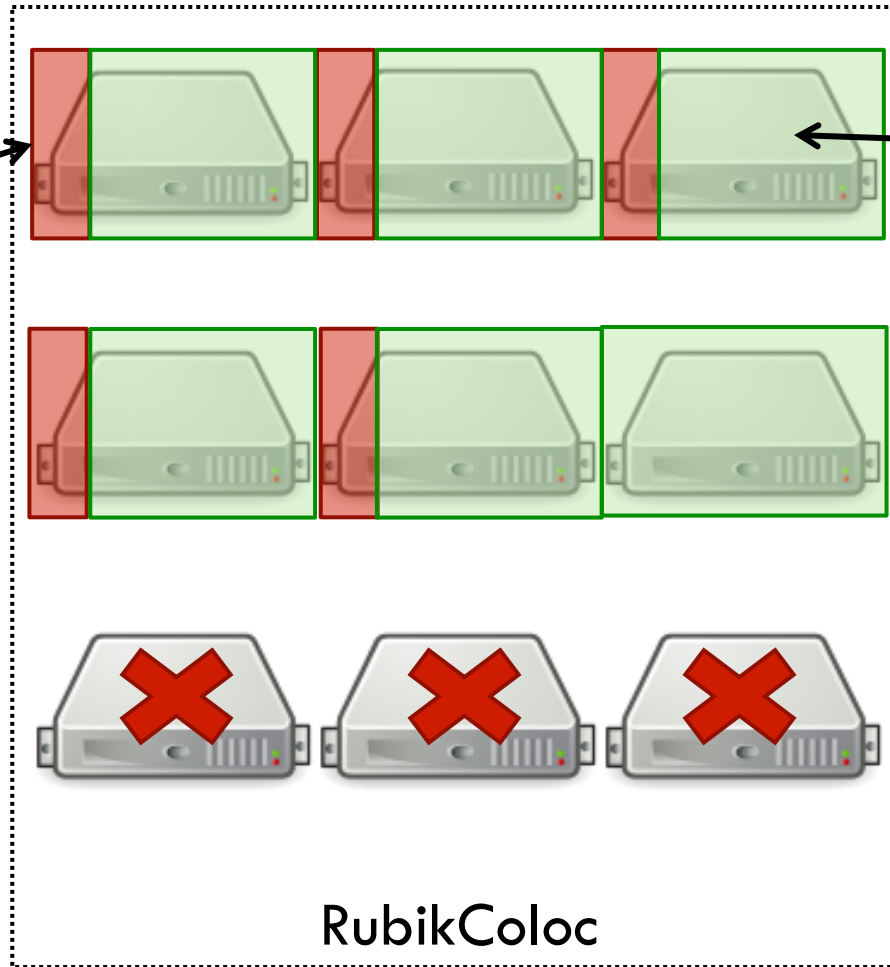


# RubikColoc: Colocation Using Rubik

24

Rubik sets  
Latency-Critical  
Frequencies

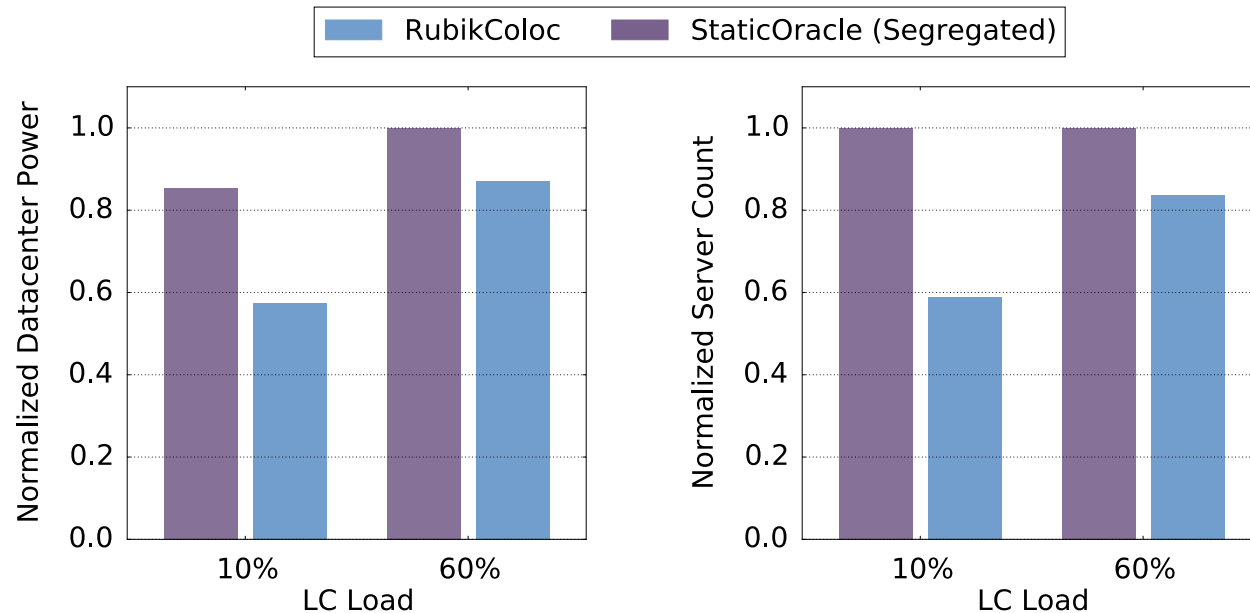
Statically  
Partitioned LLC





# RubikColoc Savings

25



- ❑ RubikColoc saves significant power and resources over a segregated datacenter baseline
  - ▣ 17% reduction in datacenter power consumption; 19% fewer machines at high load
  - ▣ 31% reduction in datacenter power consumption, 41% fewer machines at high load

- ❑ Rubik uses fine-grained power management to reduce active core power consumption by up to 66%
- ❑ Rubik uses statistical modeling to account for various sources of uncertainty, and avoids application-specific heuristics
- ❑ RubikColoc uses Rubik to colocate latency-critical and batch applications, reducing datacenter power consumption by up to 31% while using up to 41% fewer machines

**THANKS FOR YOUR ATTENTION!**

**QUESTIONS?**



**Massachusetts  
Institute of  
Technology**

