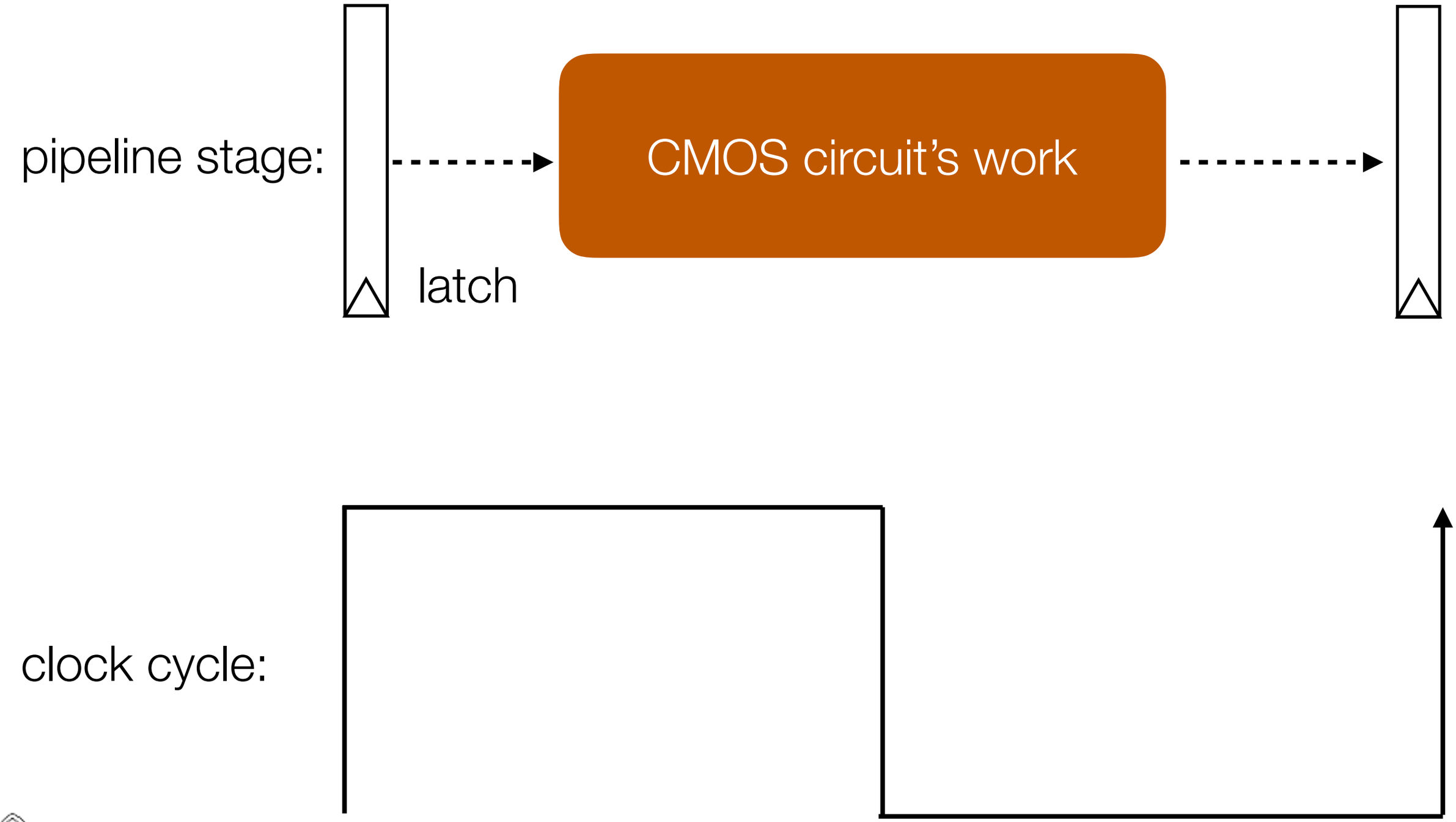
Fine-tuning the Active Timing Margin Control Loop for Maximizing Multicore Efficiency on an IBM POWER Server

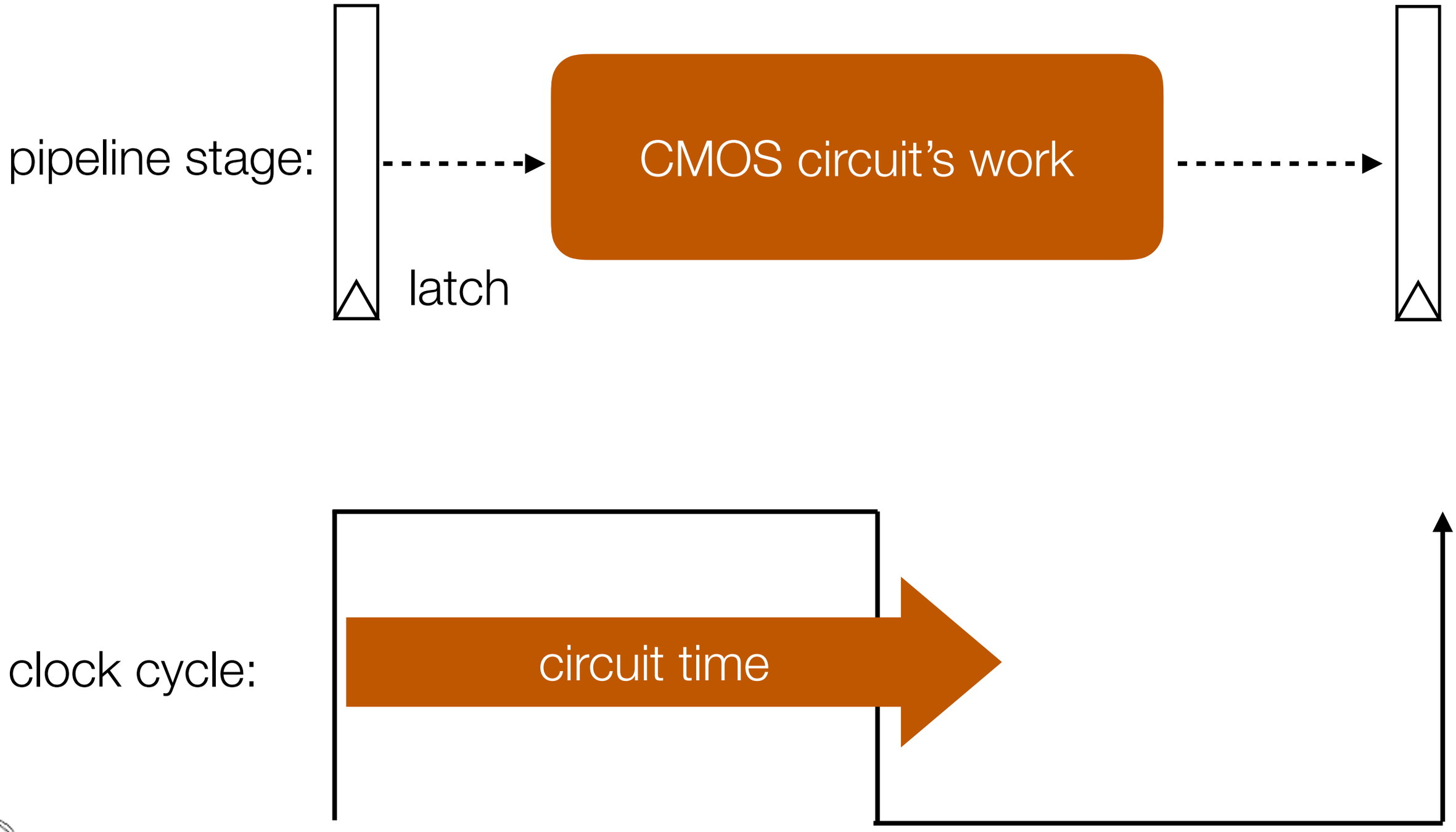
Yazhou Zu, Daniel Richins,

Charles R. Lefurgy, Vijay Janapa Reddi

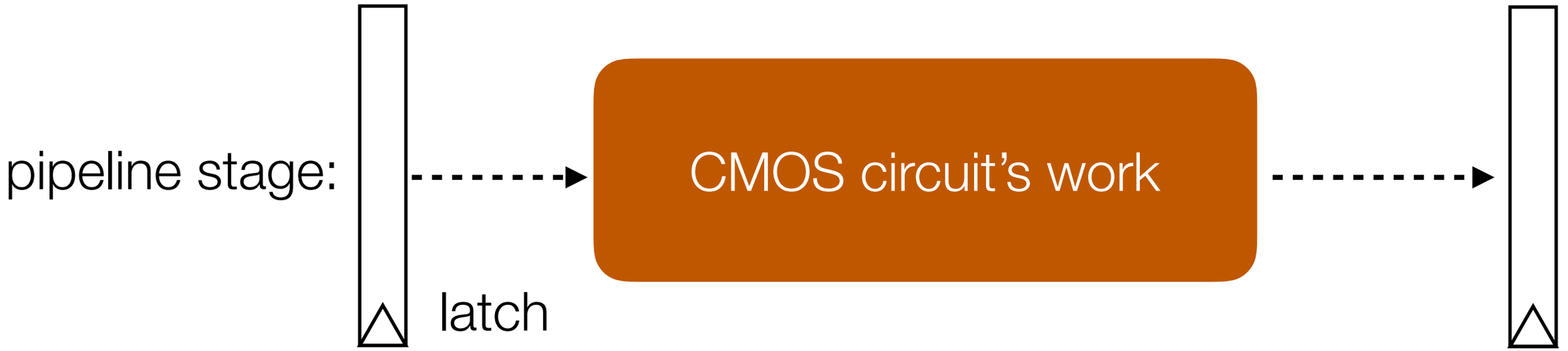
The Pipeline Timing Margin Problem

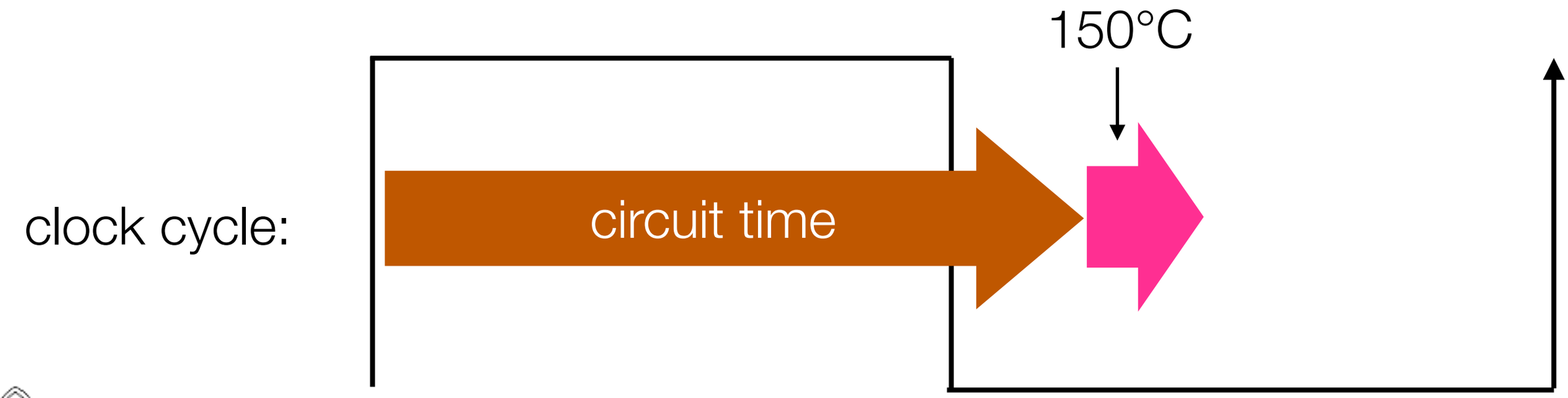


The Pipeline Timing Margin Problem

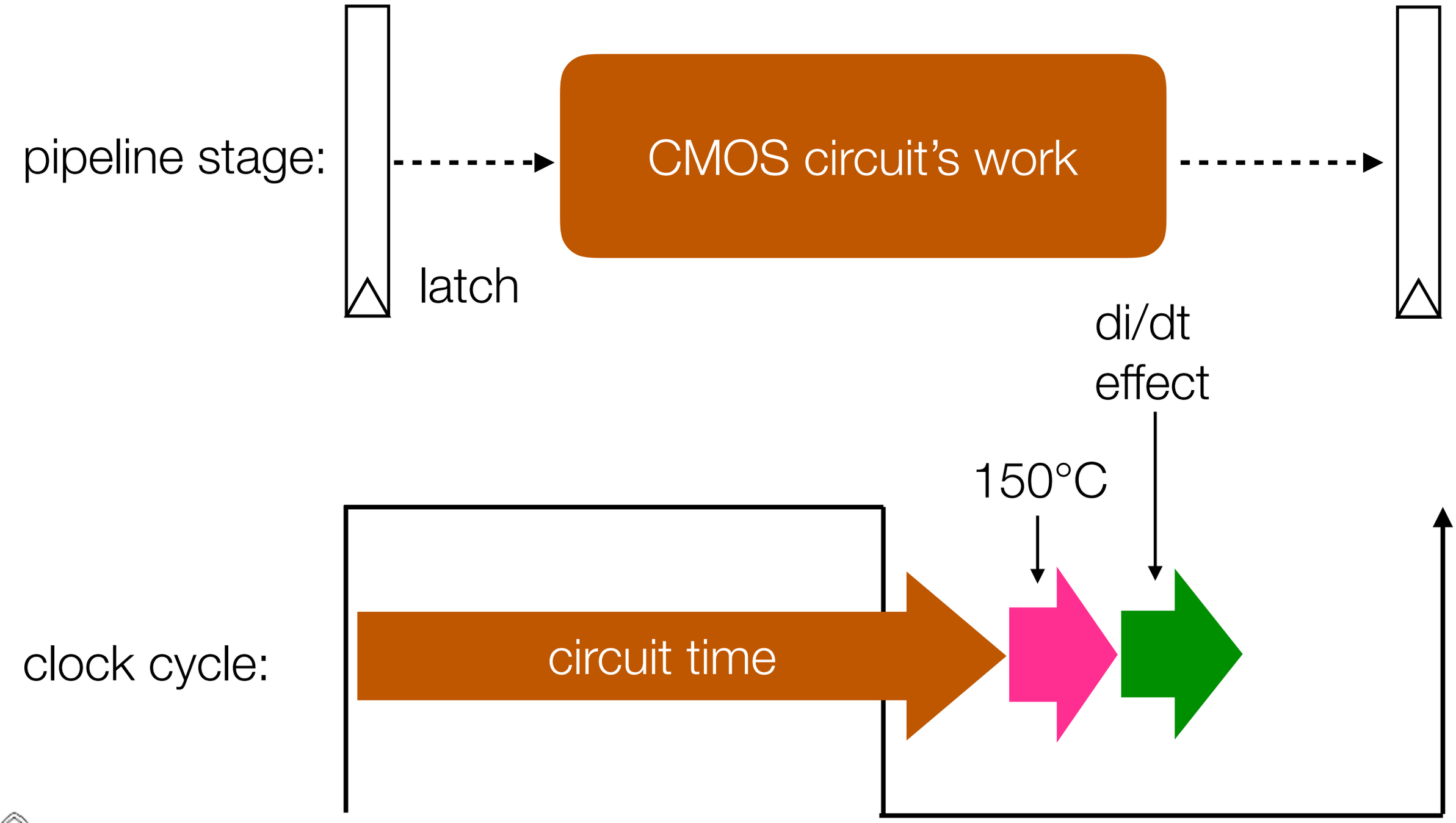


The Pipeline Timing Margin Problem

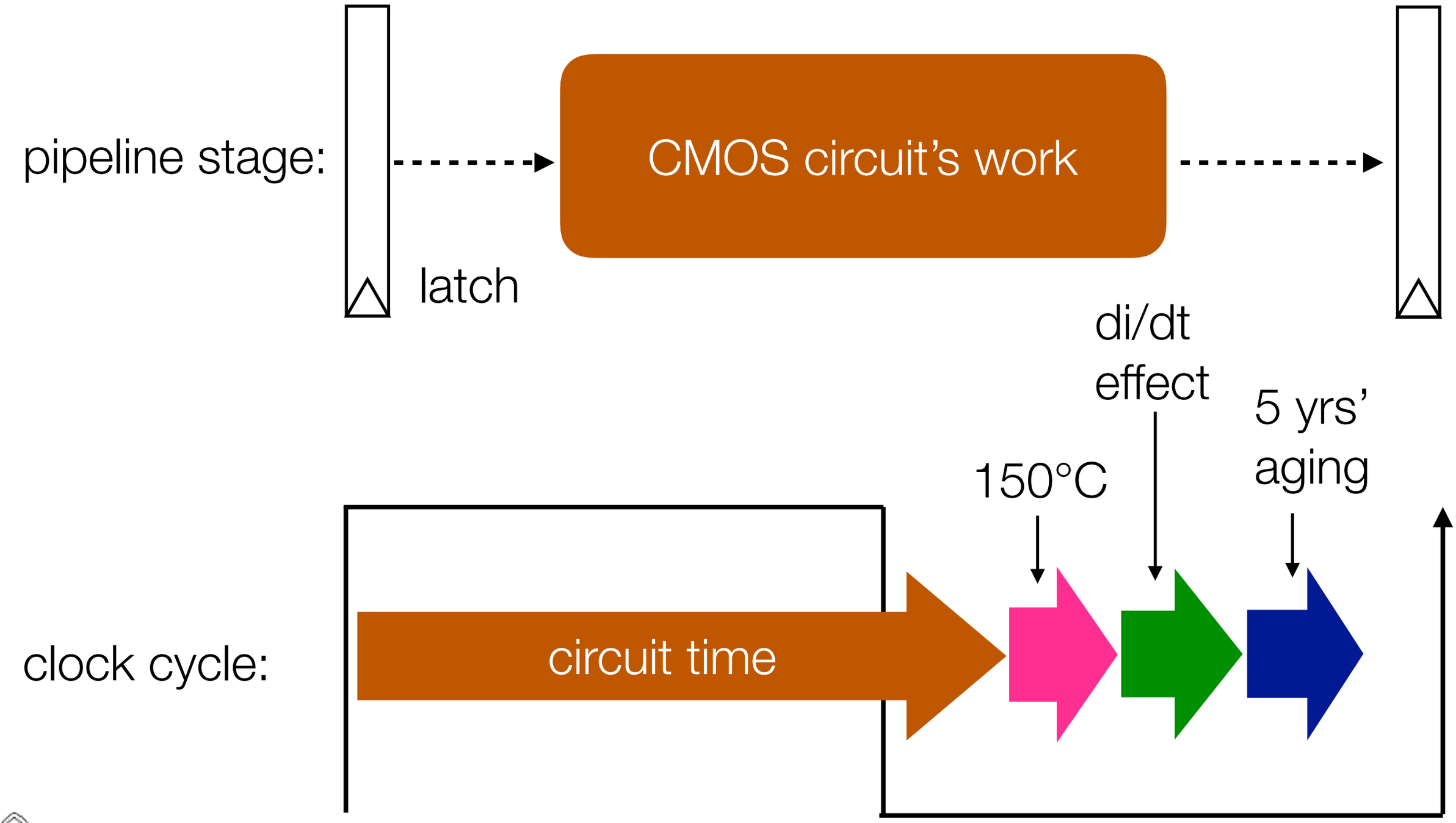




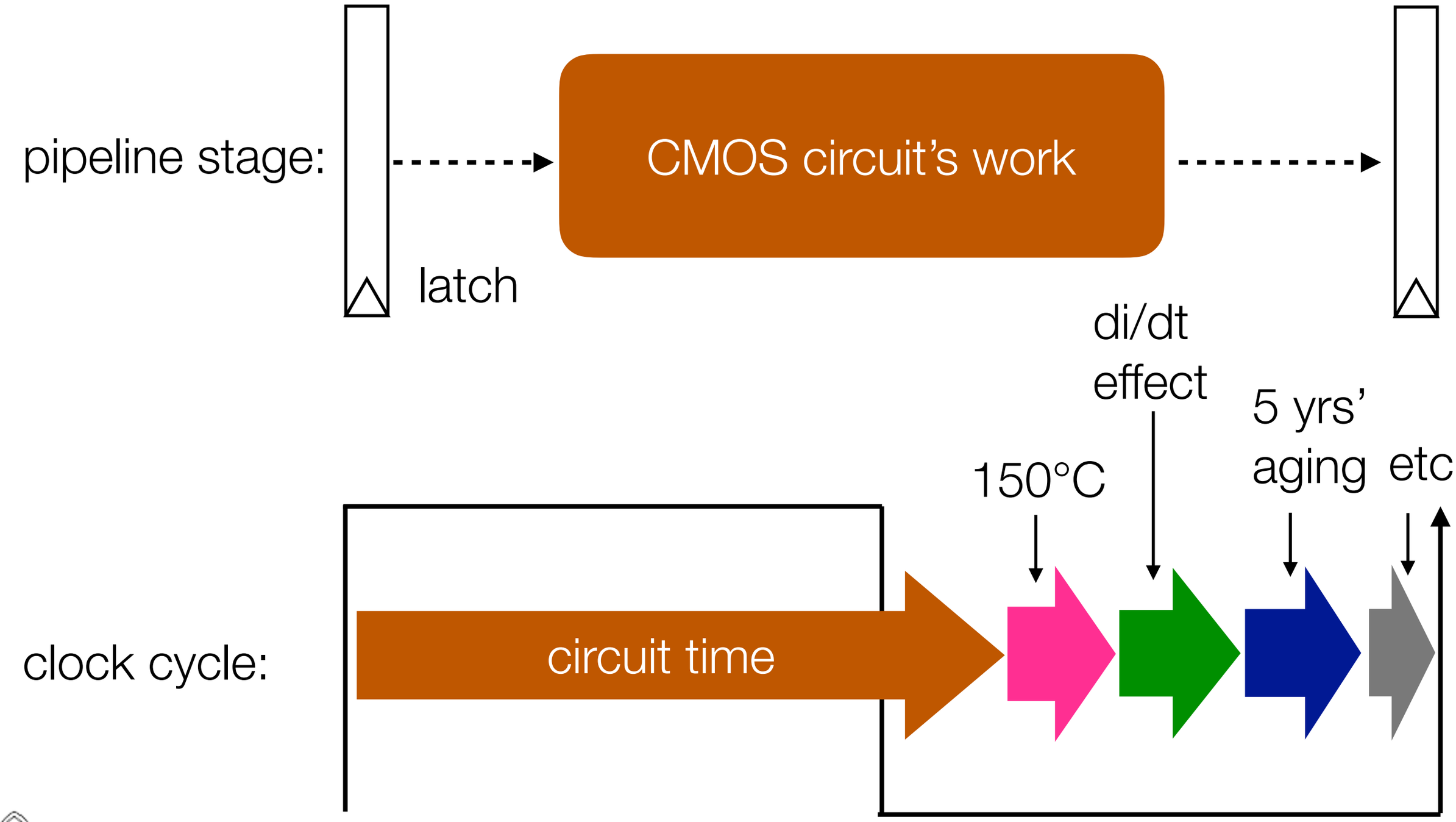
The Pipeline Timing Margin Problem



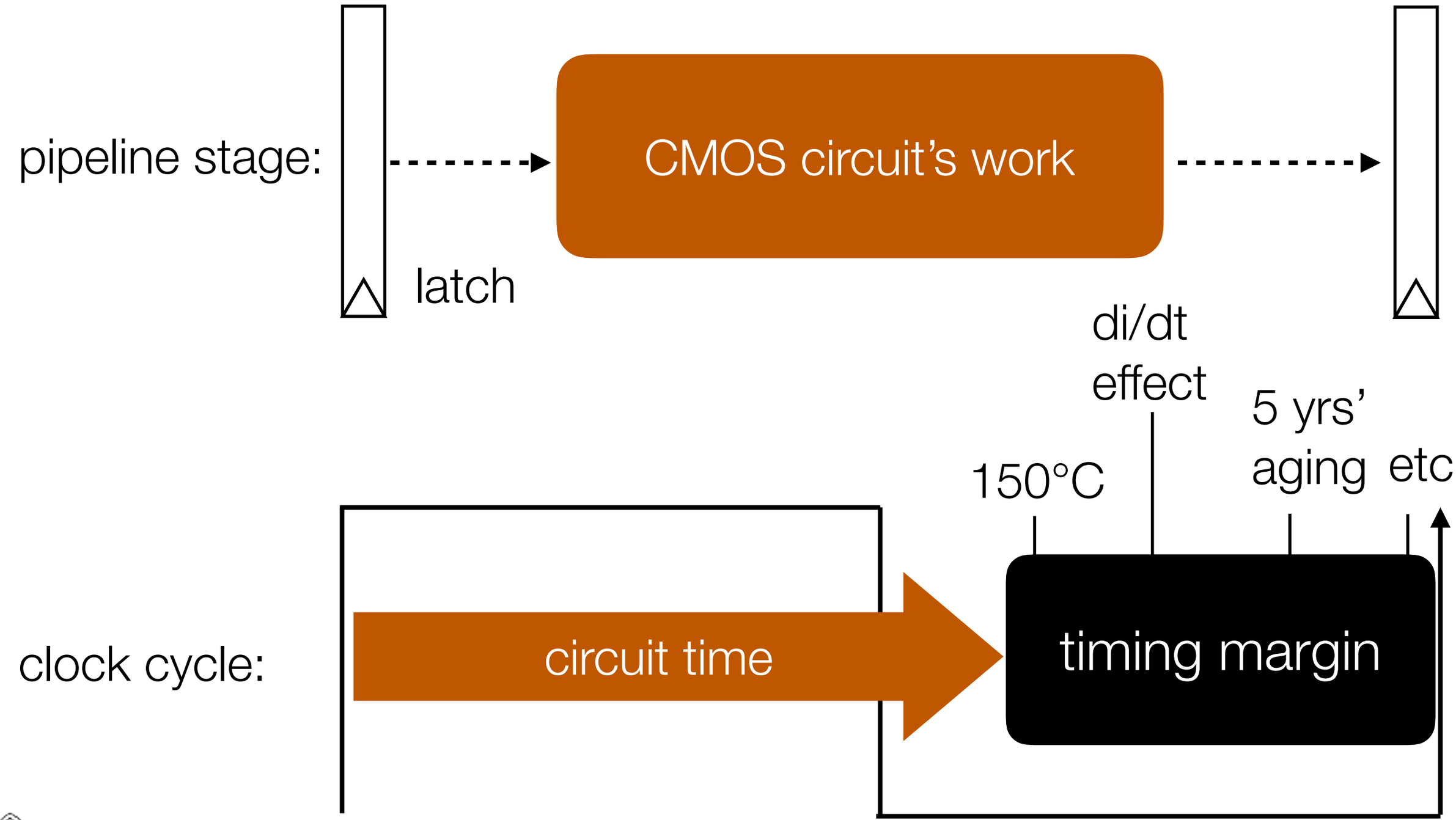
The Pipeline Timing Margin Problem



The Pipeline Timing Margin Problem

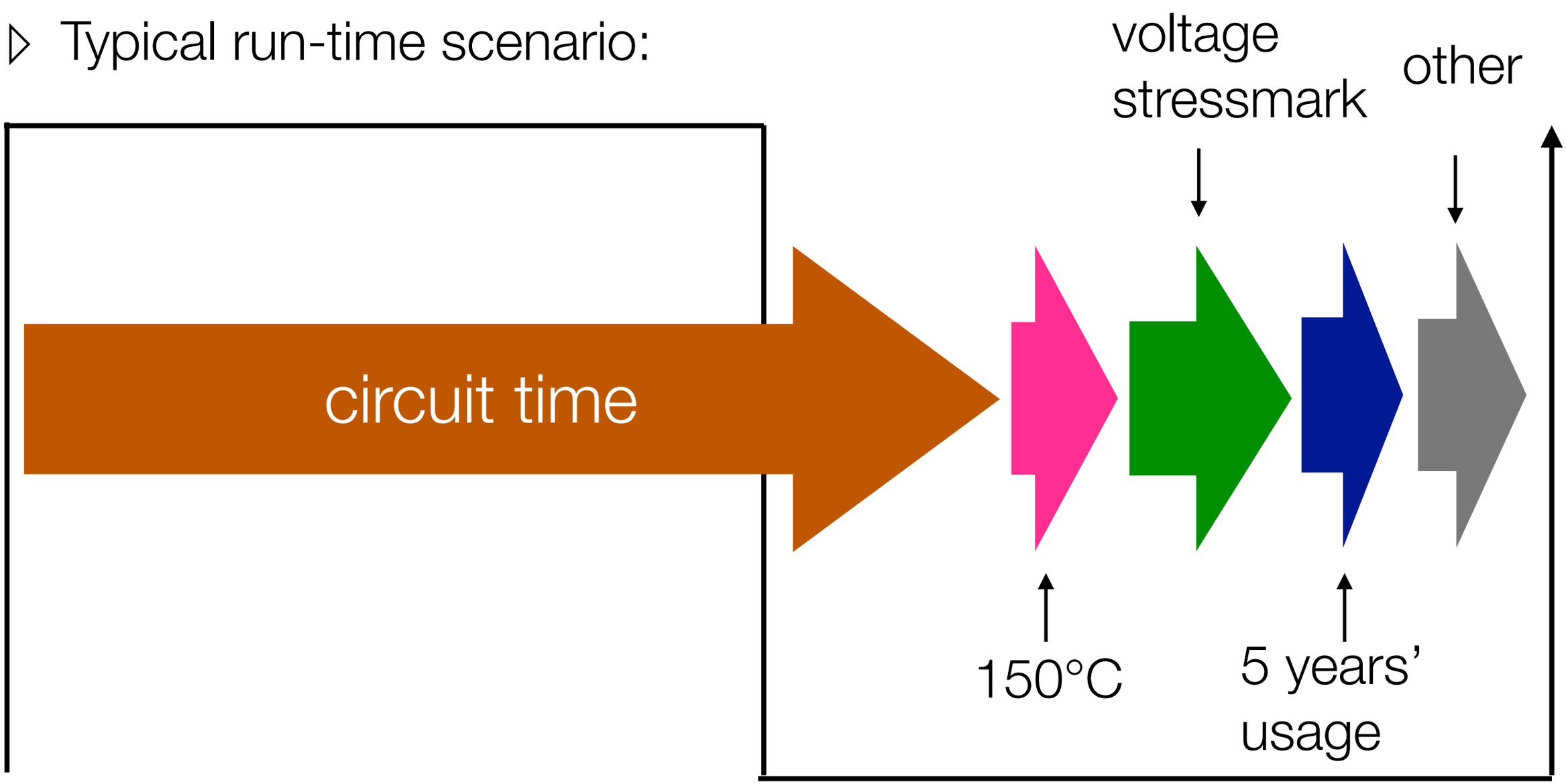


The Pipeline Timing Margin Problem



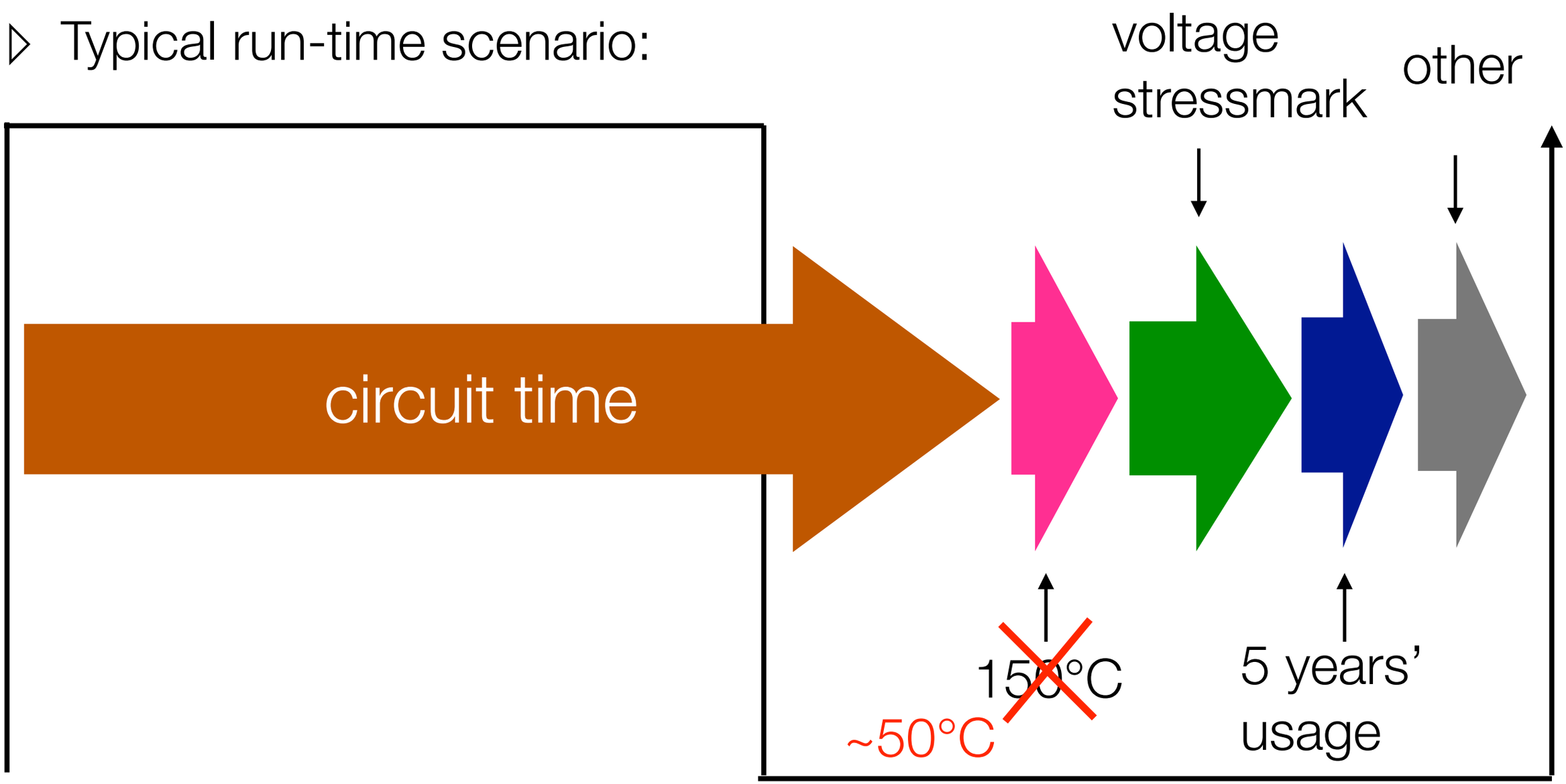
The Pipeline Timing Margin Problem

- Typical run-time scenario:



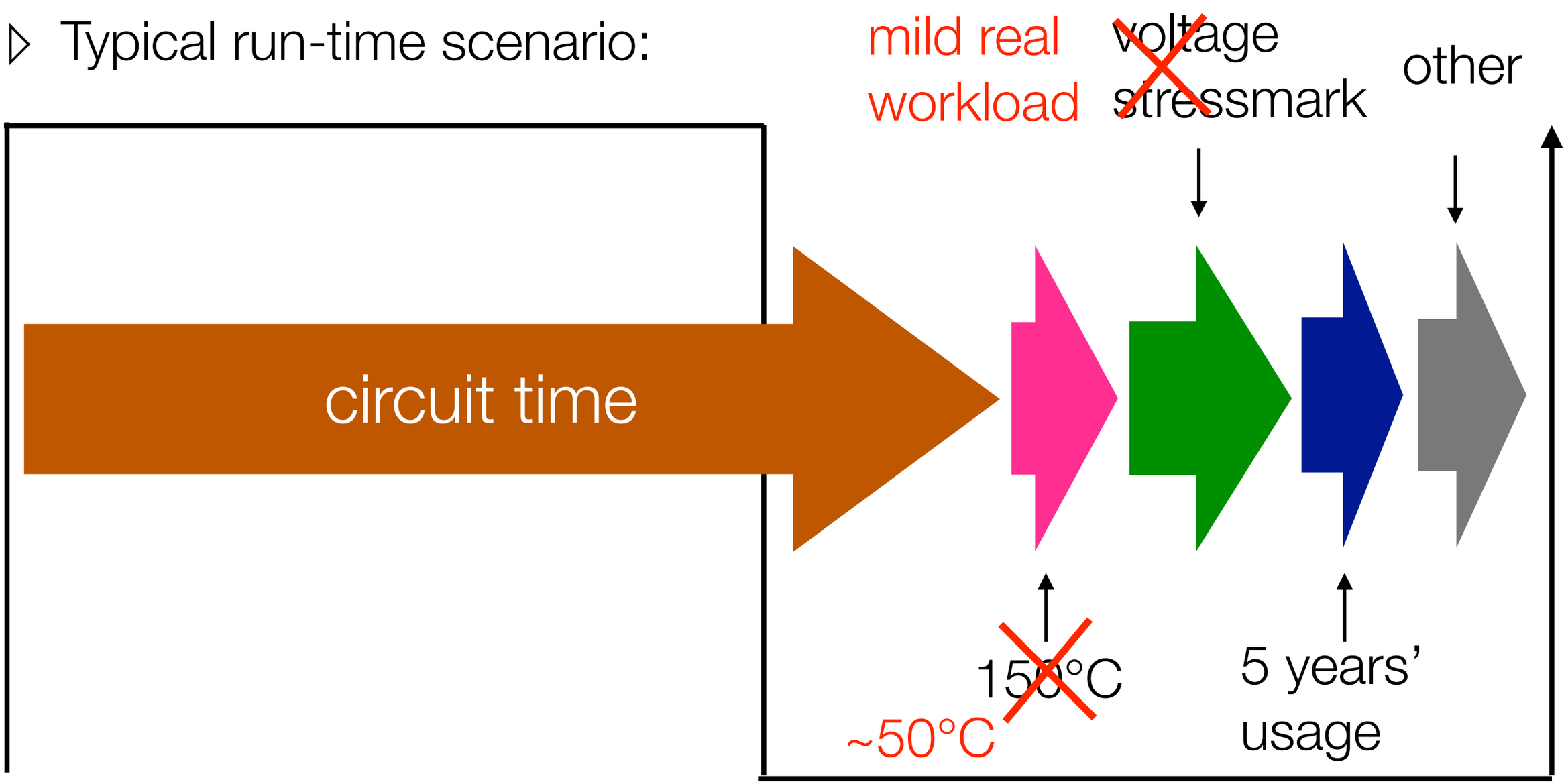
The Pipeline Timing Margin Problem

- Typical run-time scenario:



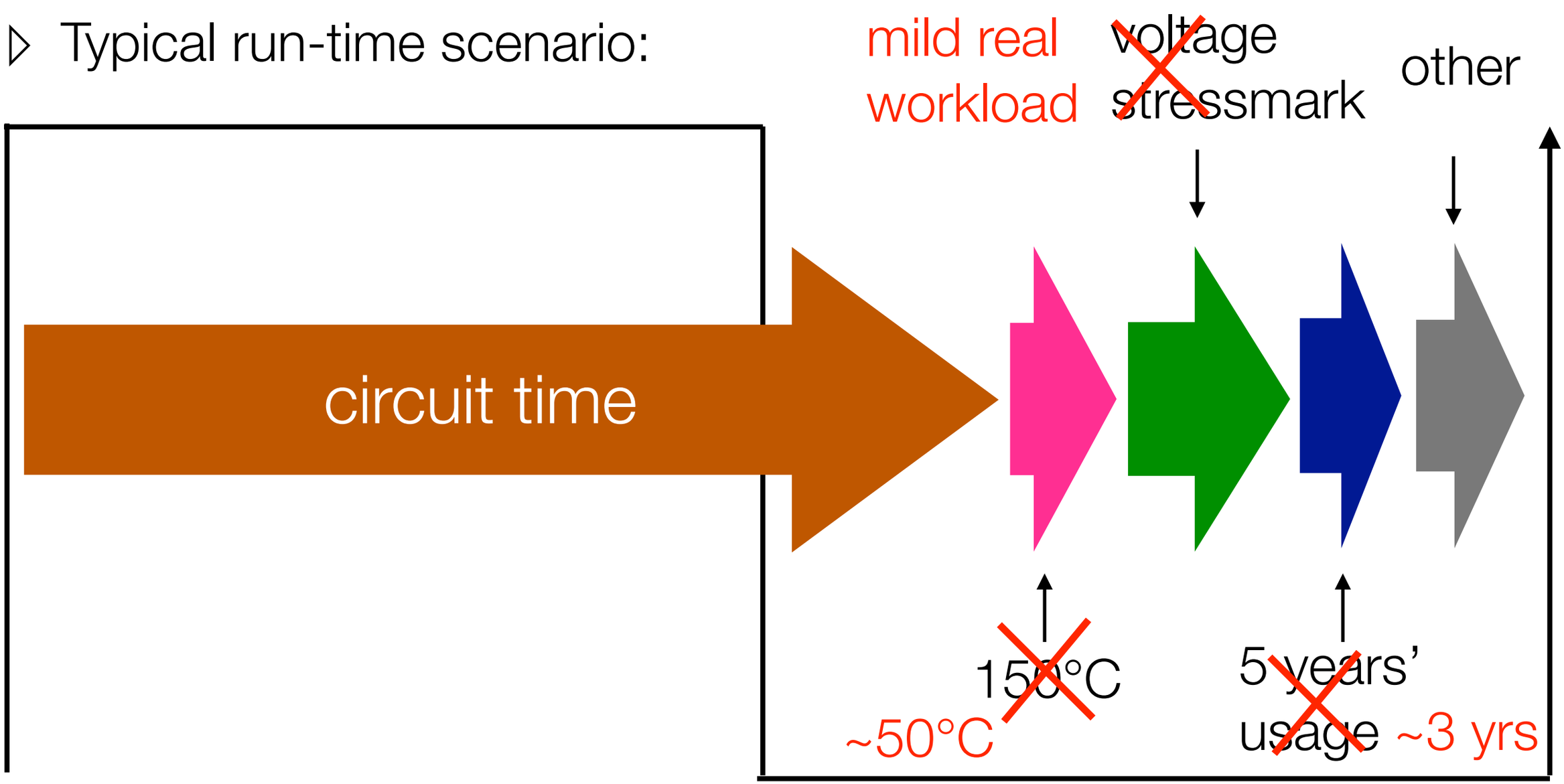
The Pipeline Timing Margin Problem

- Typical run-time scenario:



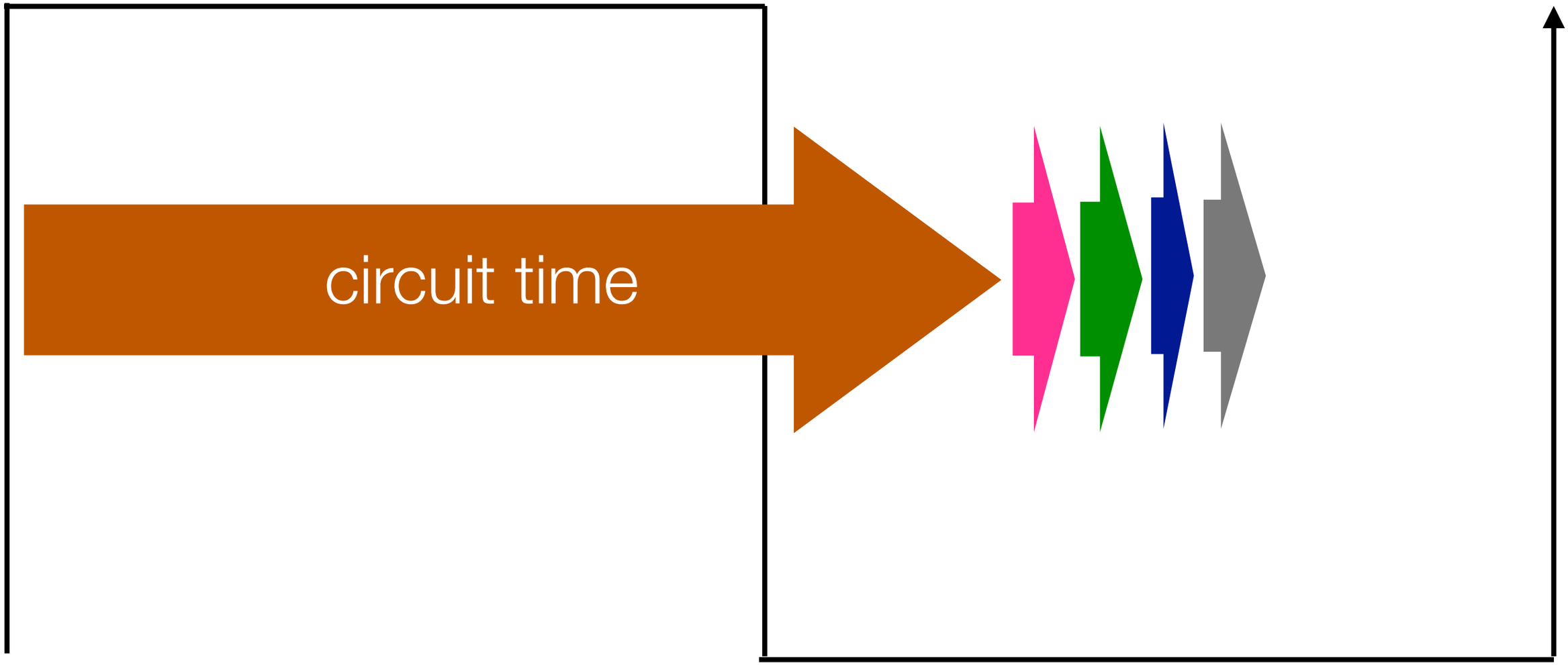
The Pipeline Timing Margin Problem

- Typical run-time scenario:



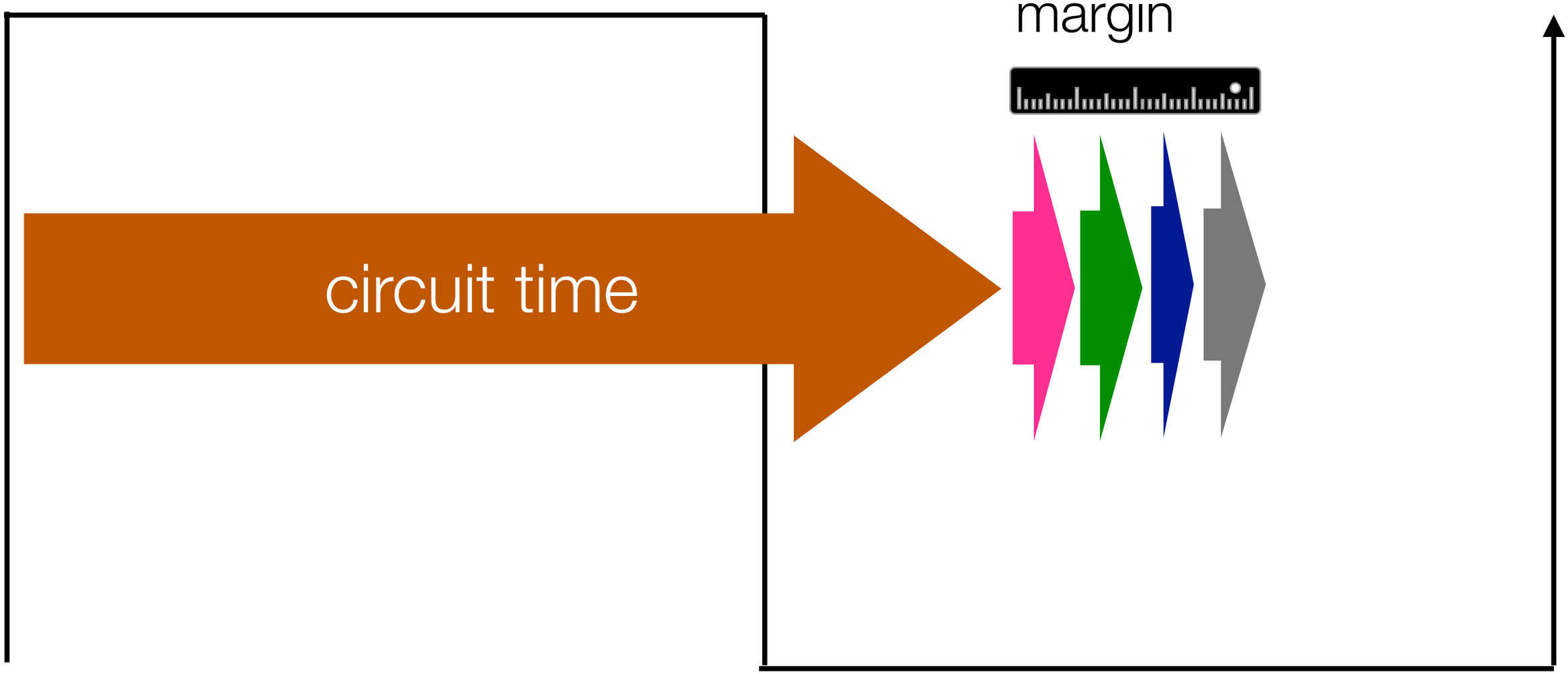
The Pipeline Timing Margin Problem

- Typical run-time scenario:



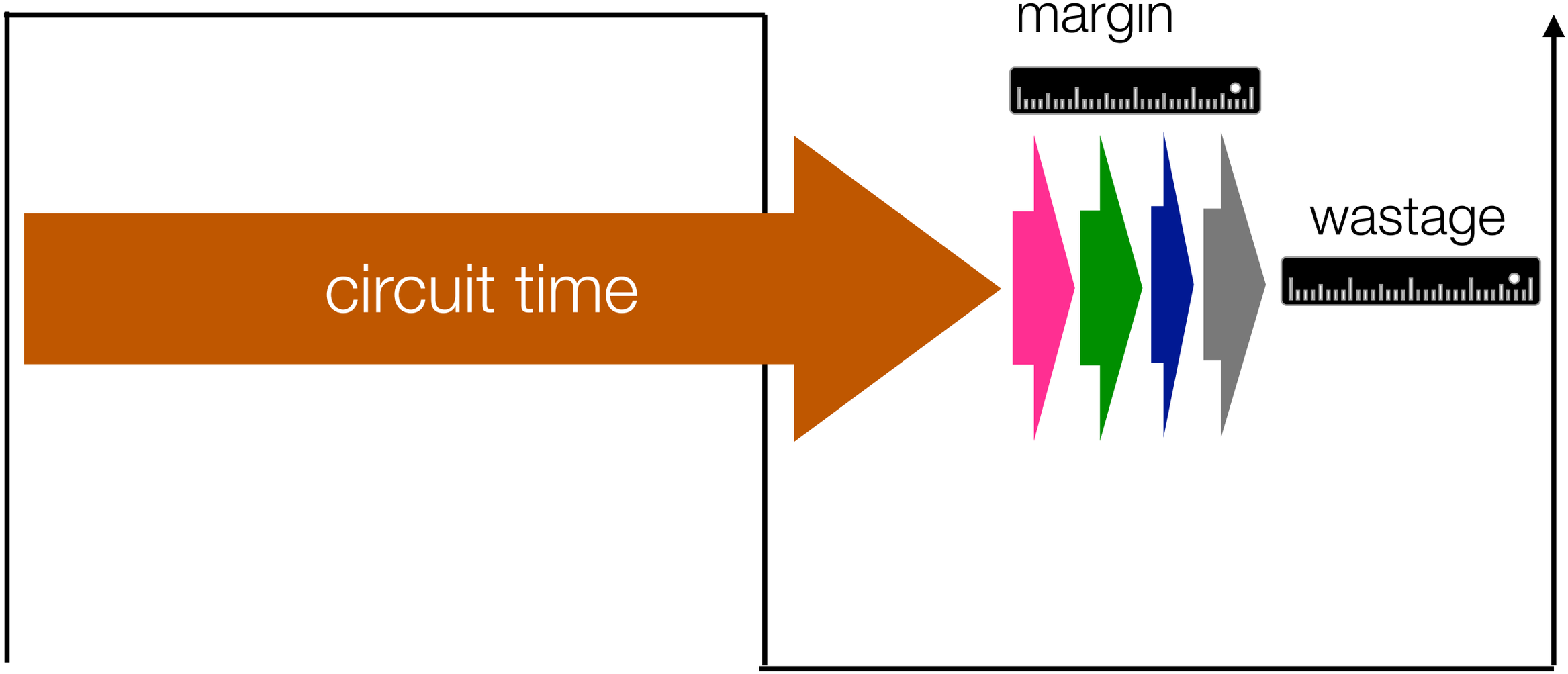
The Pipeline Timing Margin Problem

- Typical run-time scenario:

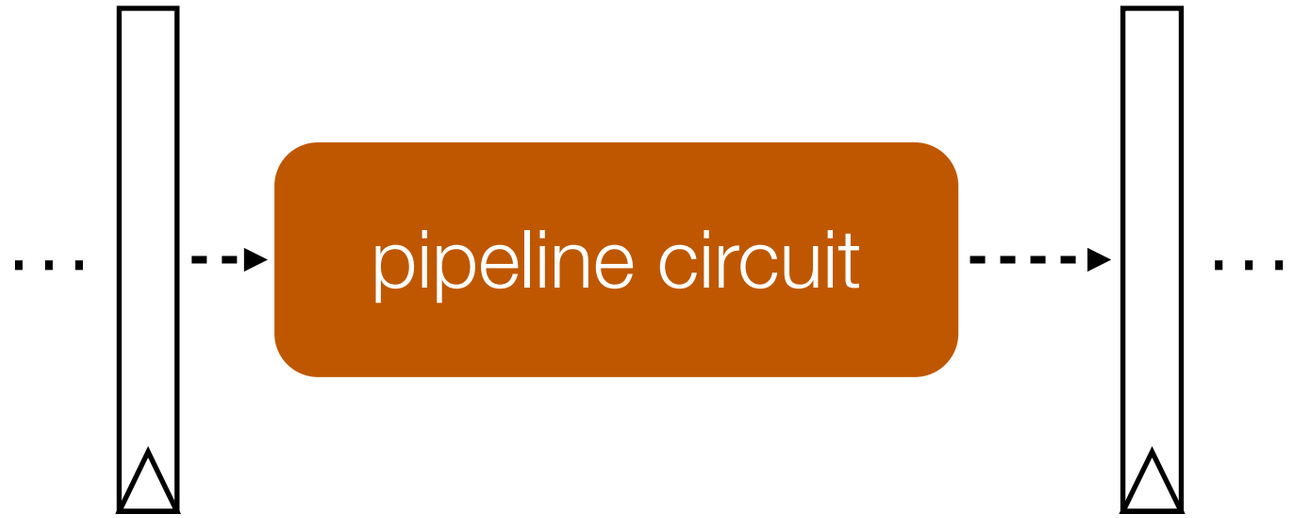


The Pipeline Timing Margin Problem

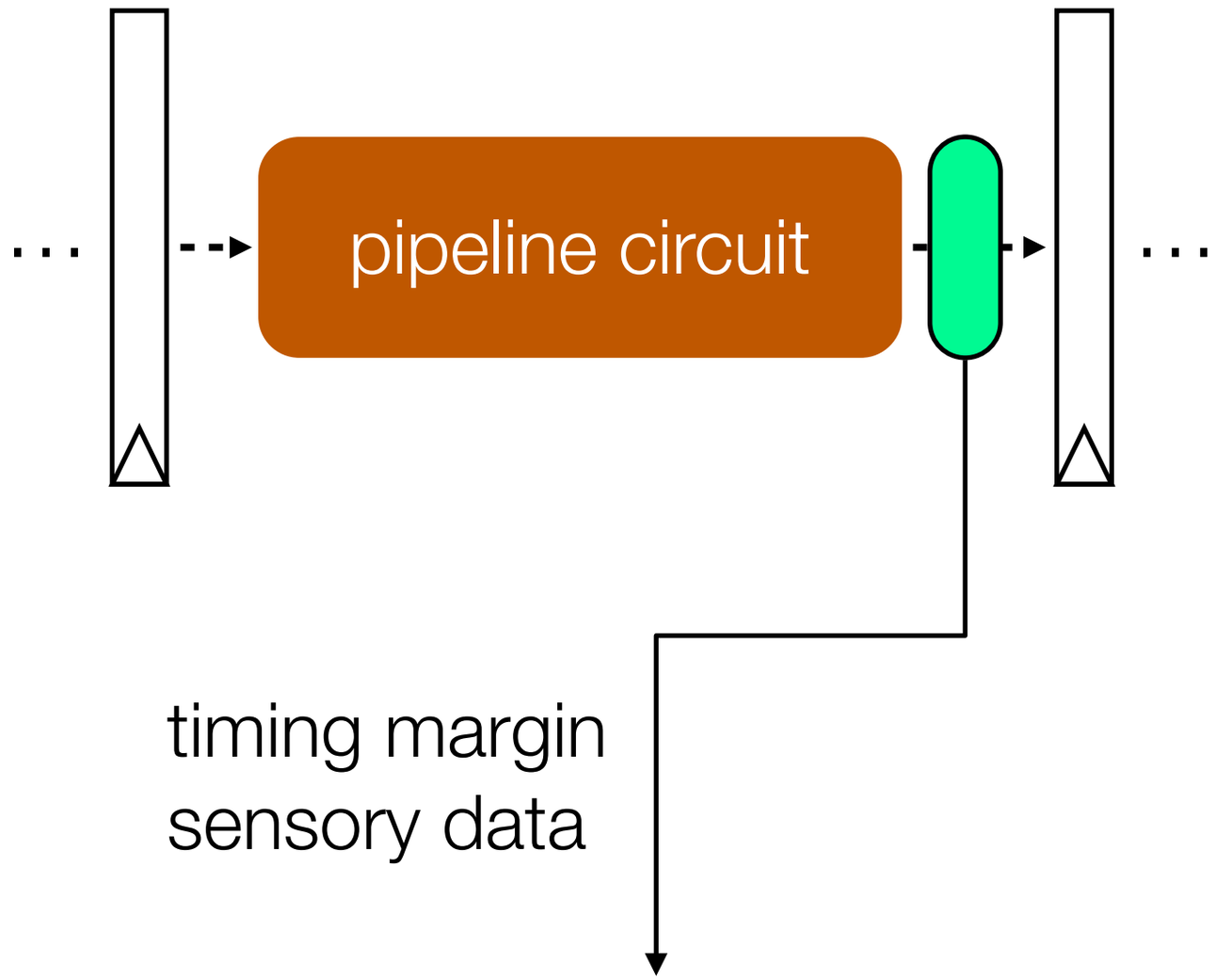
- Typical run-time scenario:



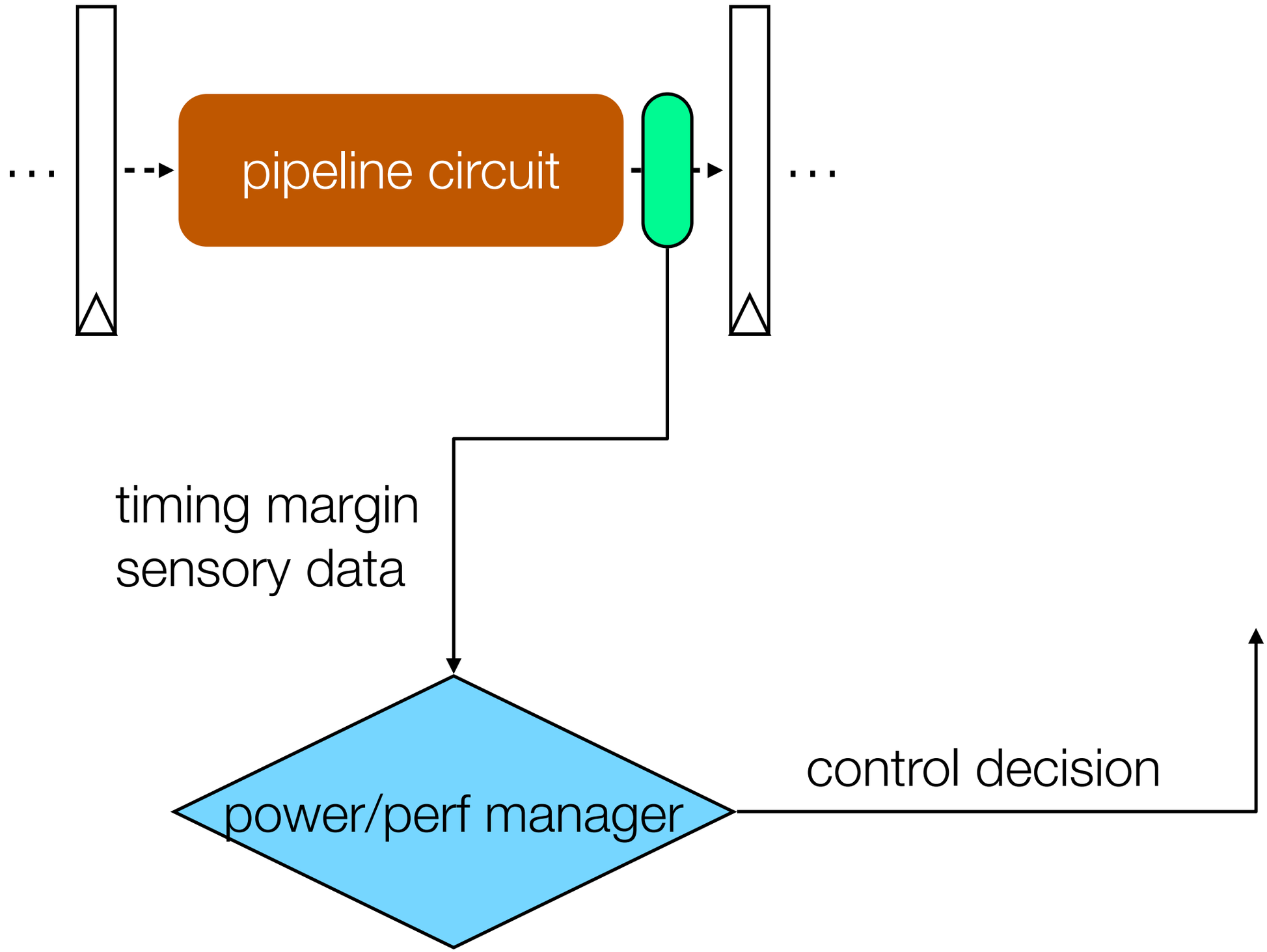
Active Timing Margin: Dynamically Provide Just Enough Margin



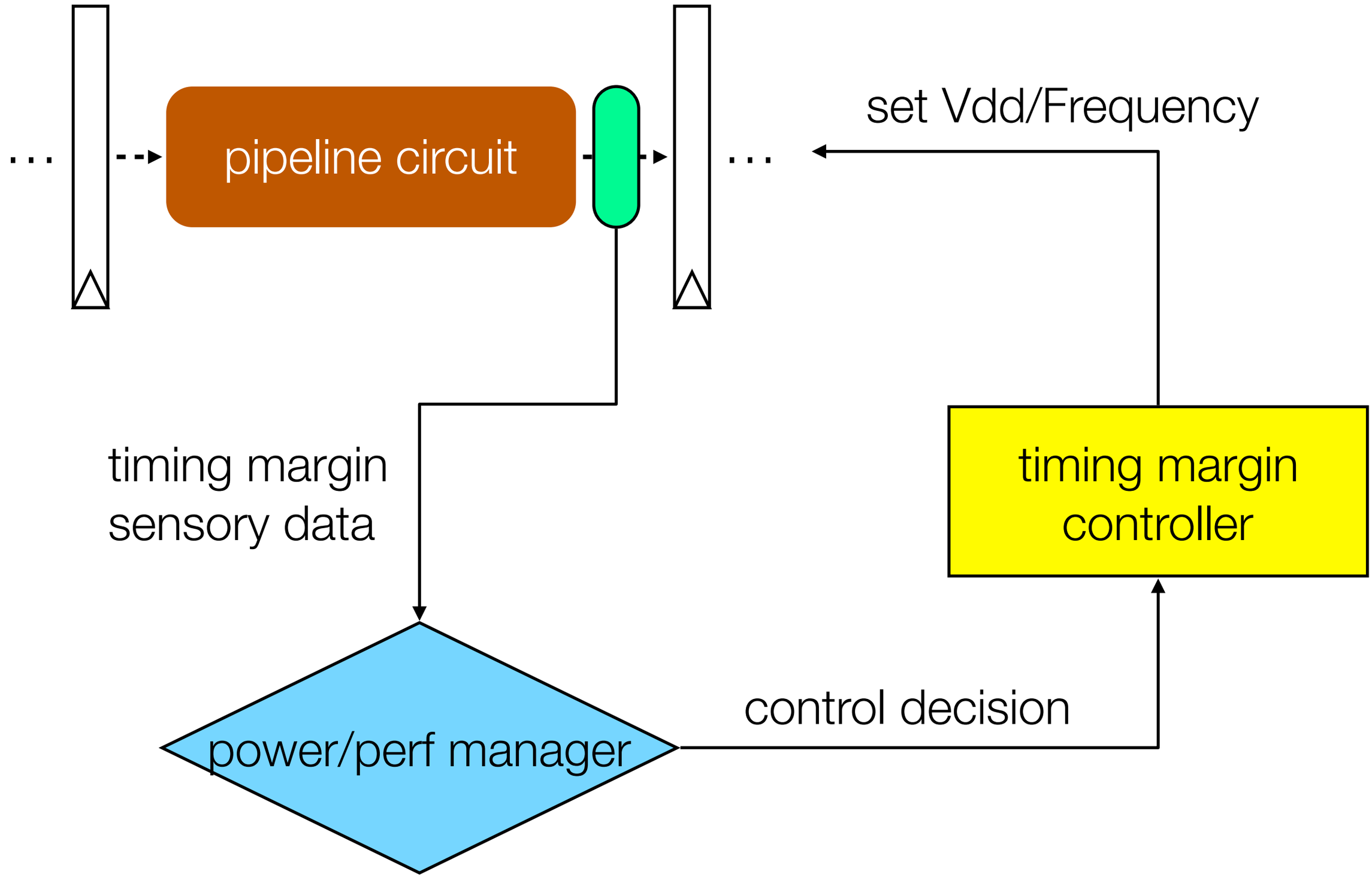
Active Timing Margin: Dynamically Provide Just Enough Margin



Active Timing Margin: Dynamically Provide Just Enough Margin



Active Timing Margin: Dynamically Provide Just Enough Margin



Active Timing Margin (ATM)

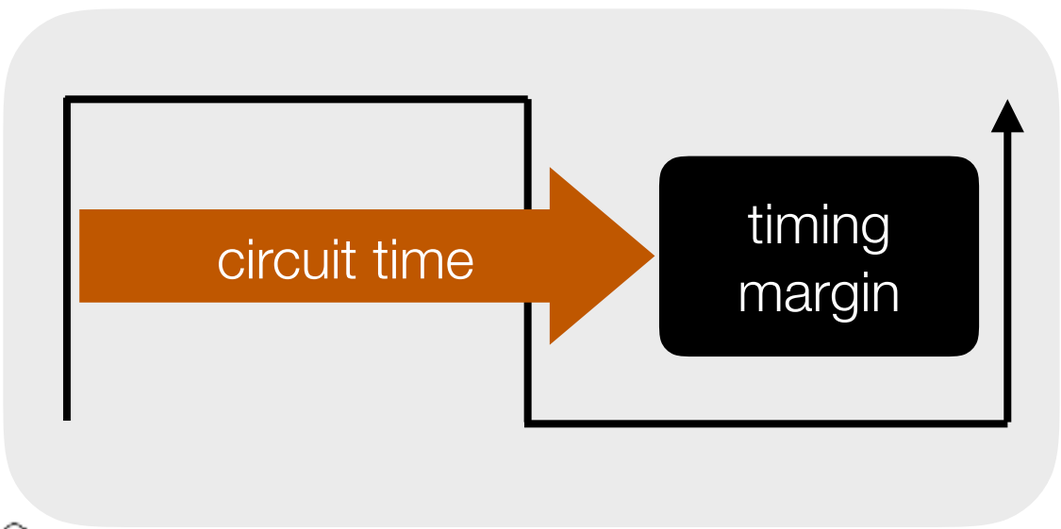
Software

Hardware

Active Timing Margin (ATM)

Software

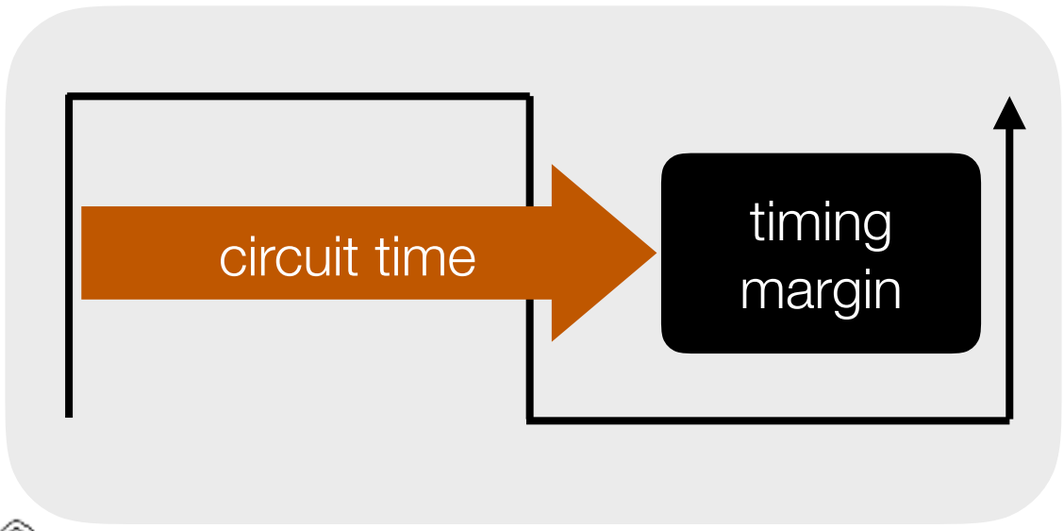
Hardware

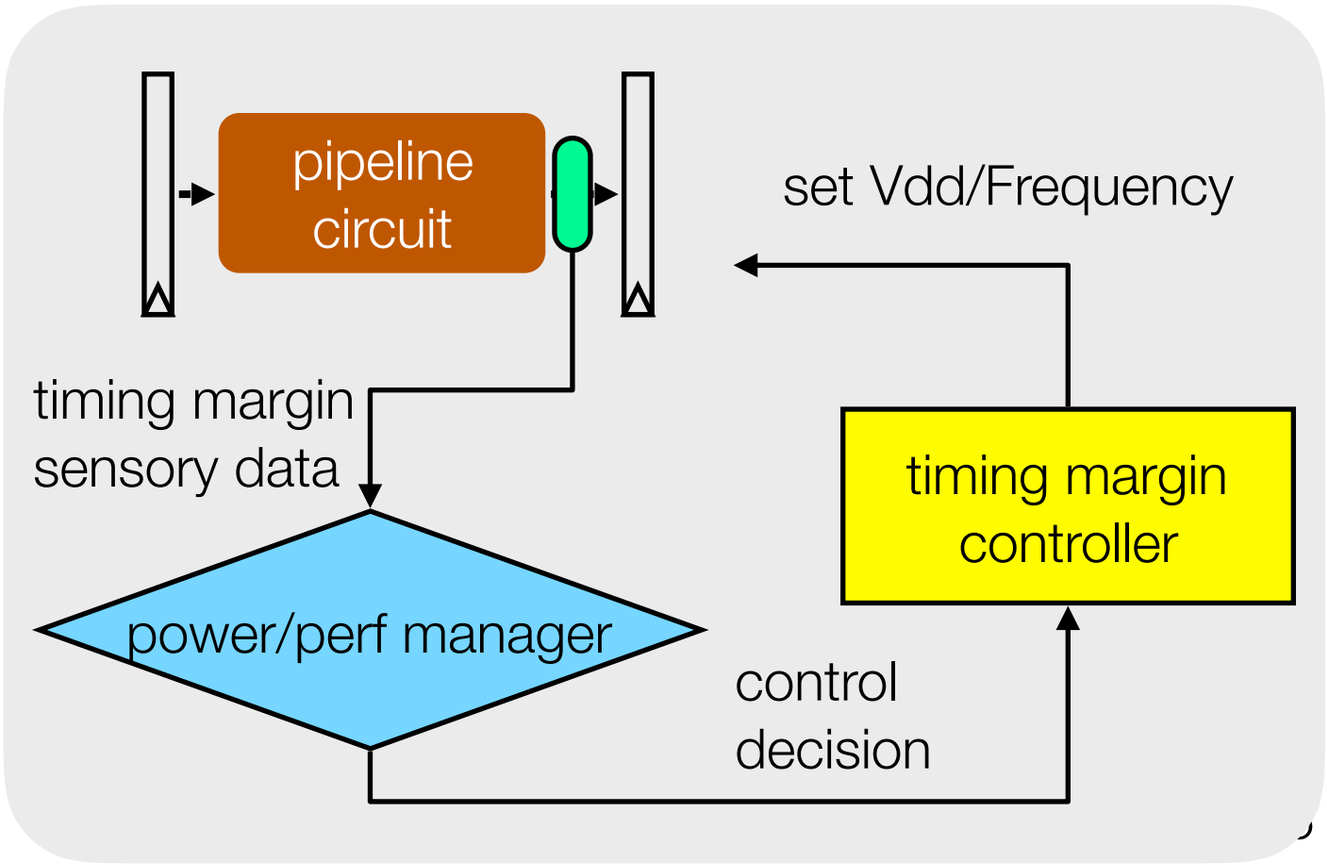


Active Timing Margin (ATM)

Software

Hardware



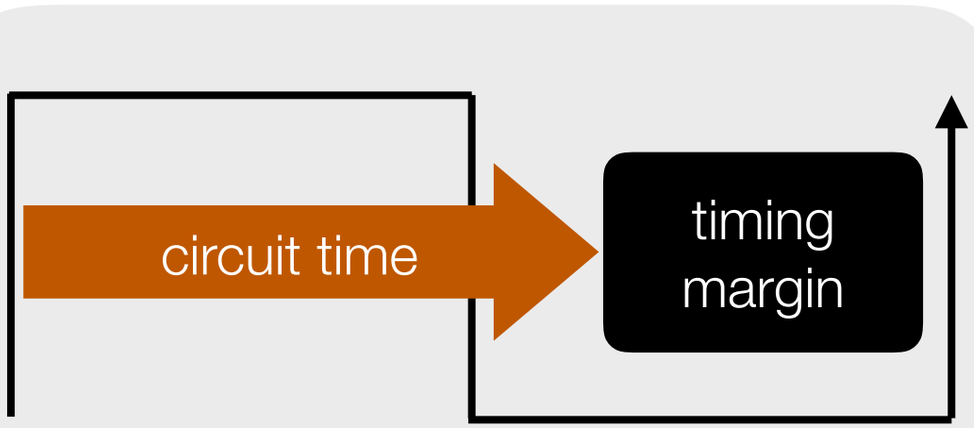


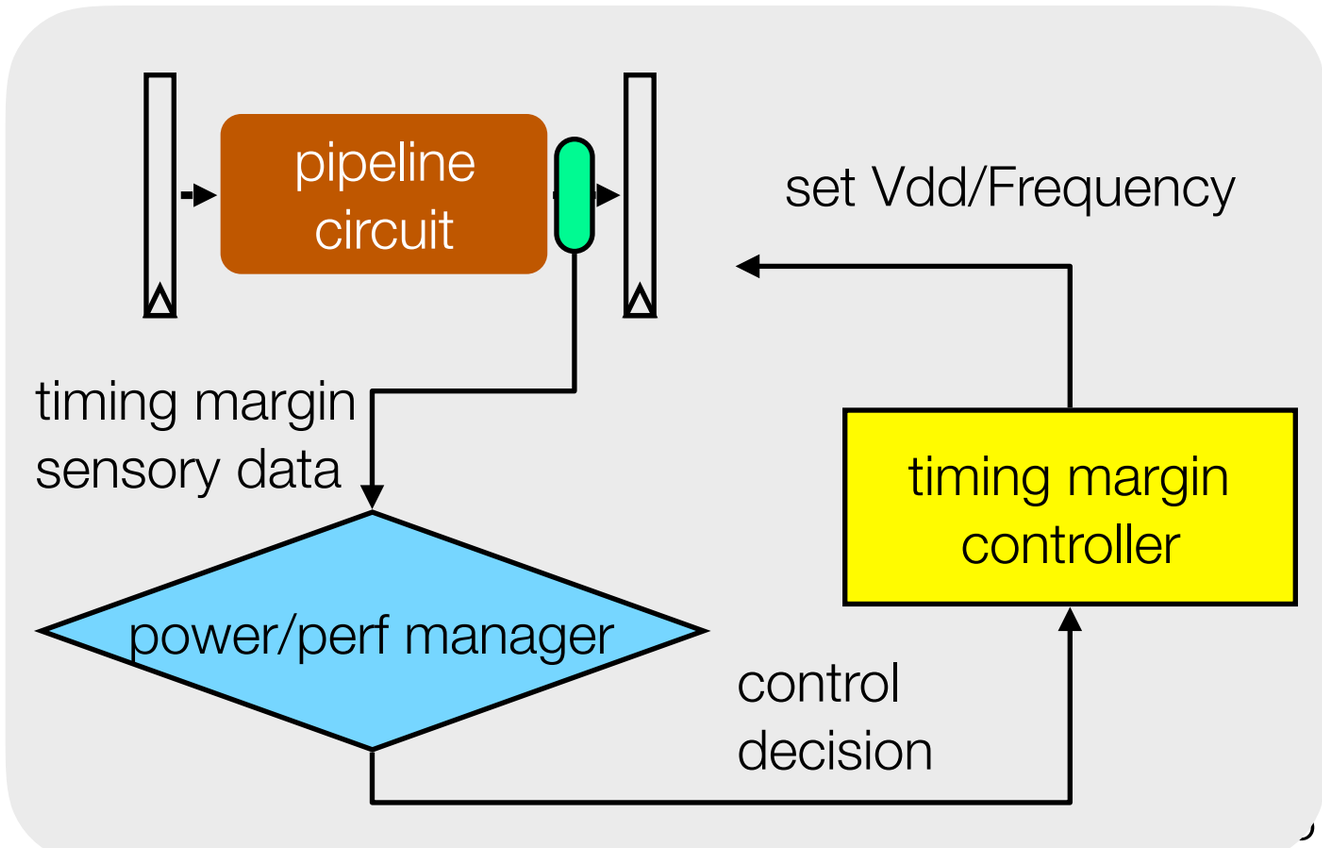
Active Timing Margin (ATM)

Software

Runs Anything!

Hardware



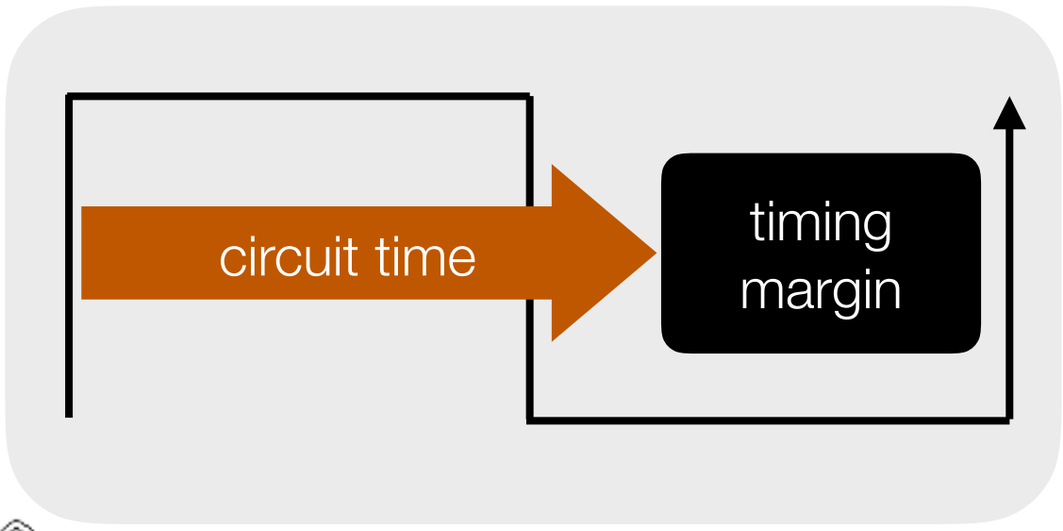


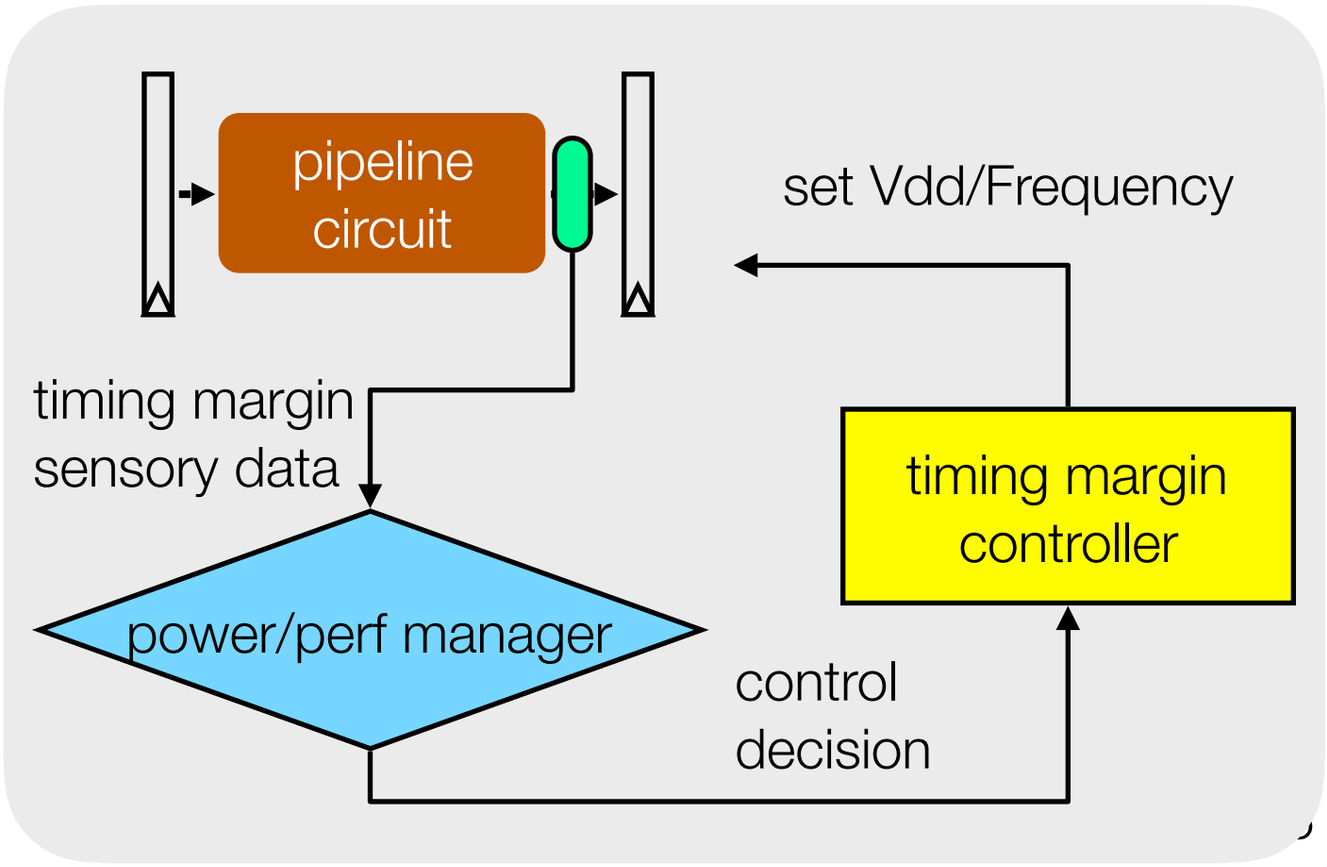
Active Timing Margin (ATM) — A Free Meal

Software

Runs Anything!

Hardware



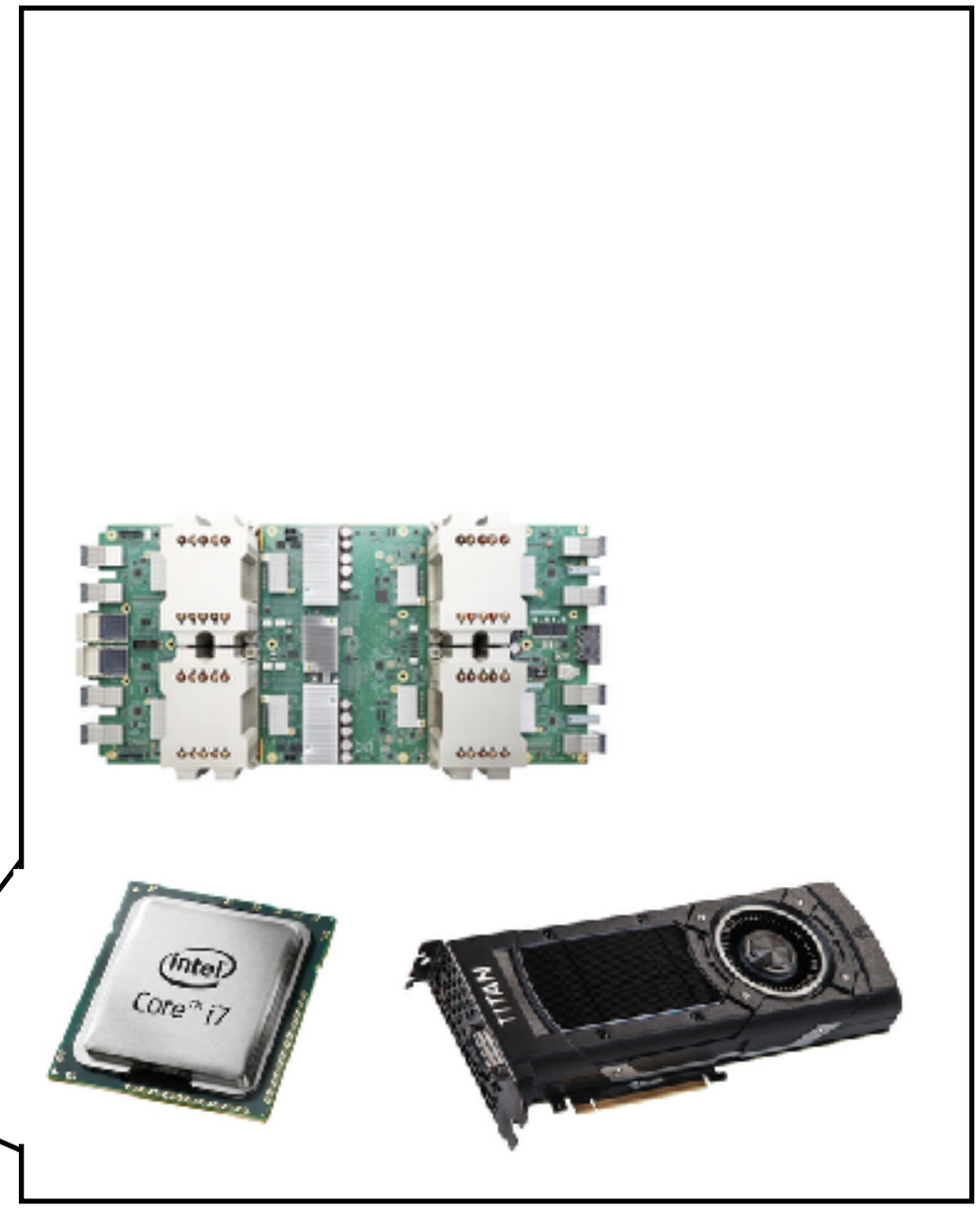


Active Timing Margin (ATM) — A Free Meal

Active Timing Margin (ATM) — A Free Meal

Active Timing Margin (ATM) — A Free Meal

Active Timing Margin (ATM) — A Free Meal



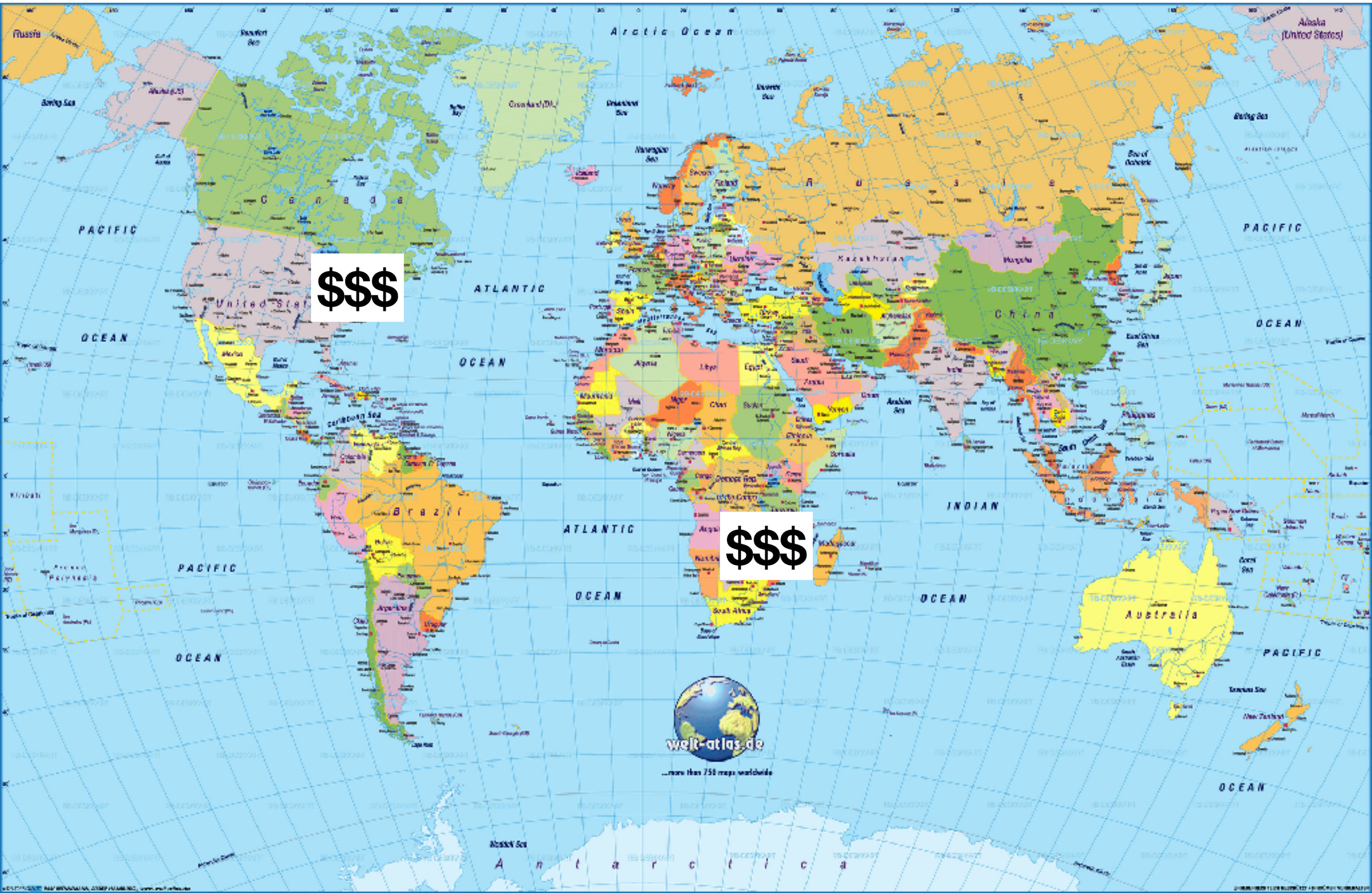
Active Timing Margin (ATM) — A Free Meal

Active Timing Margin (ATM) — A Free Meal

Problems of Existing ATMs - (Has to) Ignore Variation

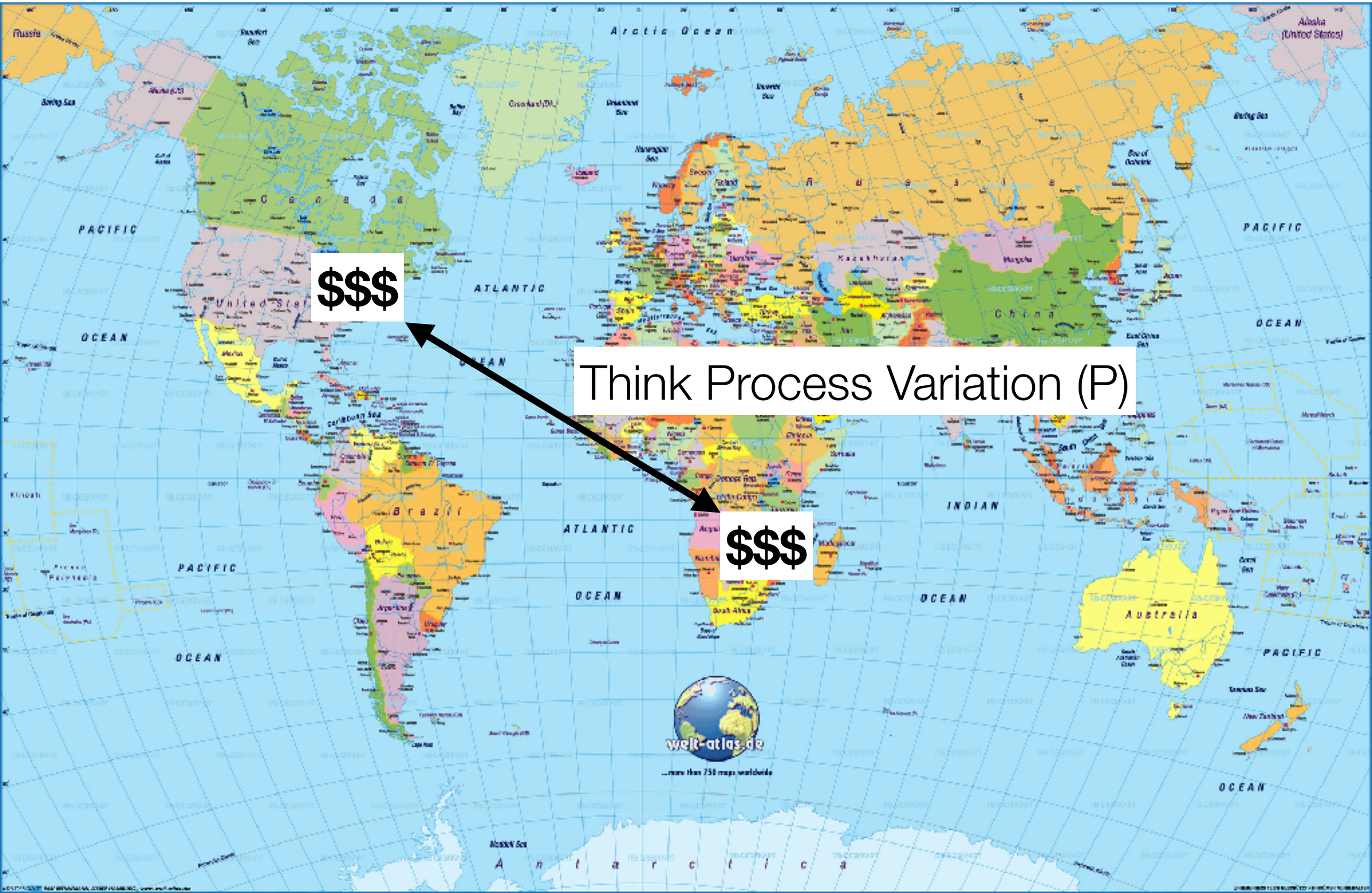
Problems of Existing ATMs - (Has to) Ignore Variation

Problems of Existing ATMs - (Has to) Ignore Variation



Problems of Existing ATMs - (Has to) Ignore Variation

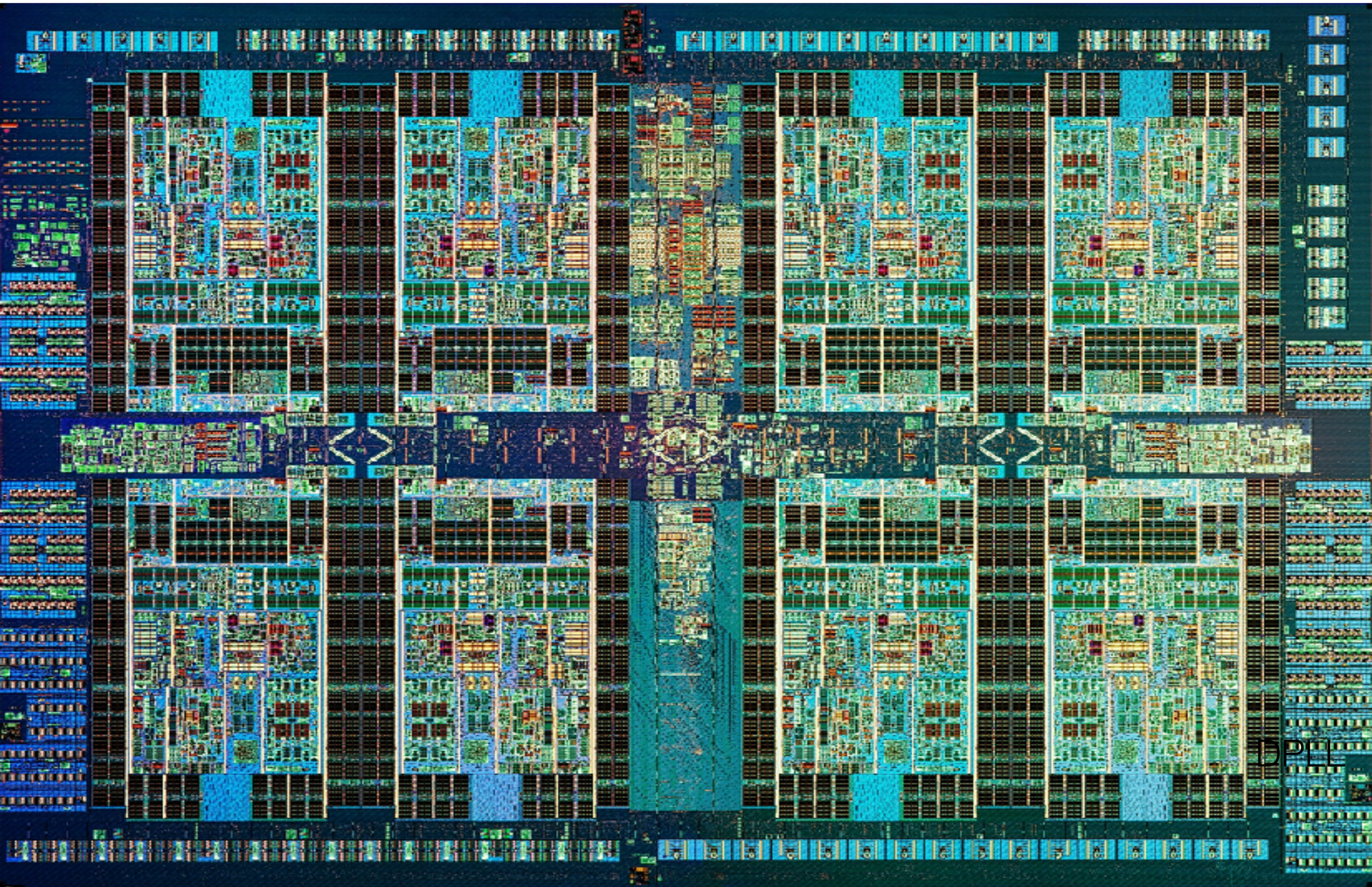
Problems of Existing ATMs - (Has to) Ignore Variation



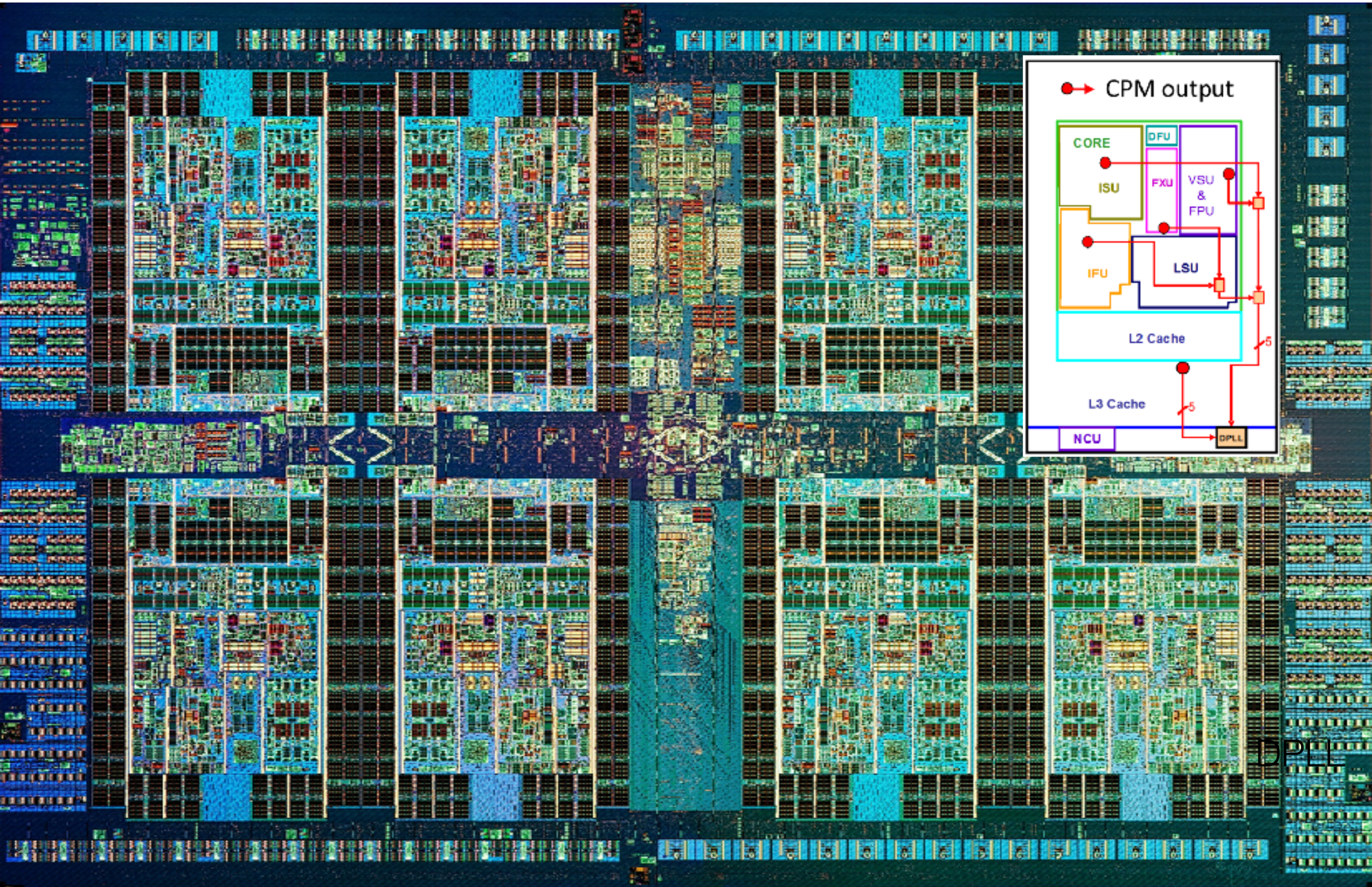
Executive Summary

- Fine-tune core-level ATM to **automatically** expose inter-core performance variation

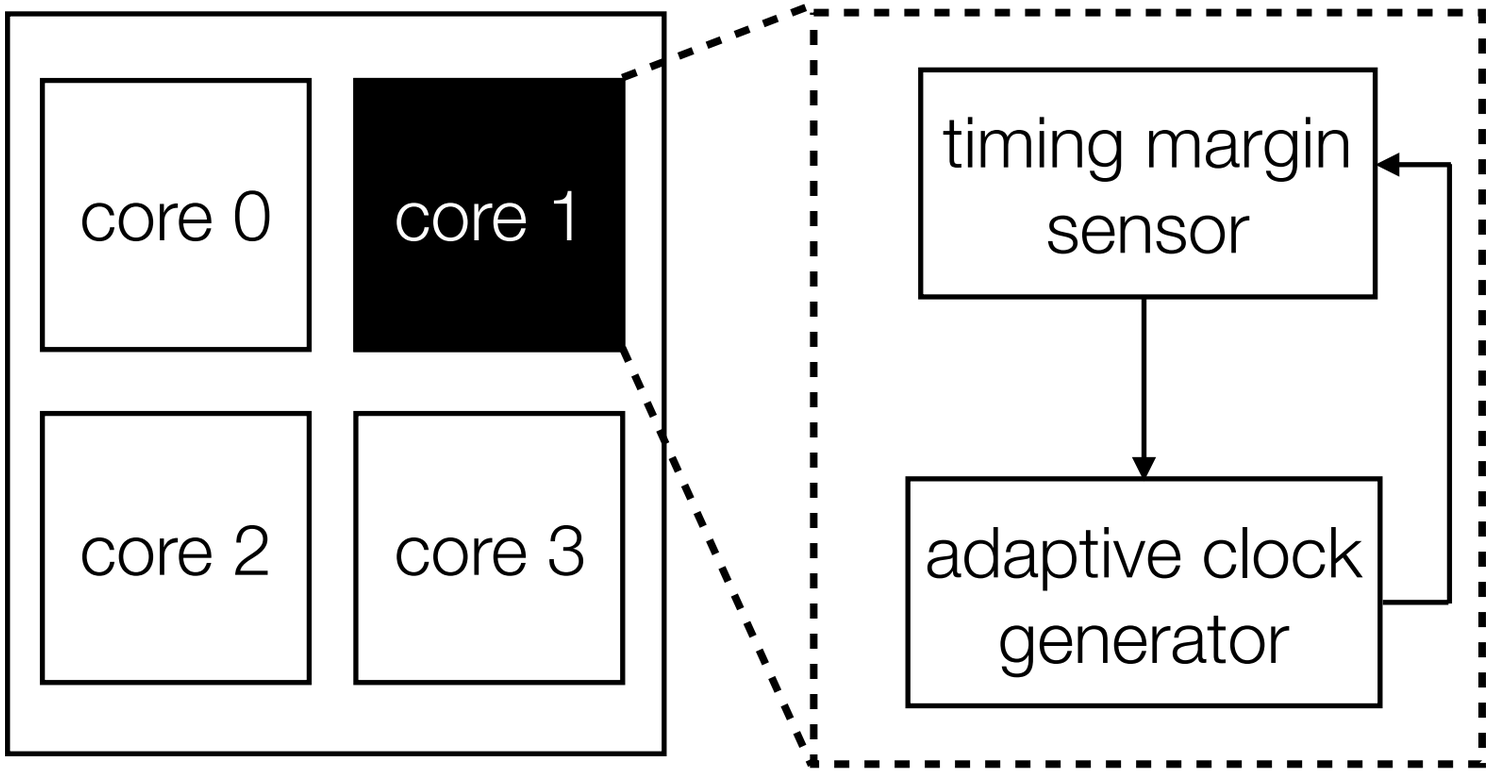
Experimental Setup: IBM® Eight-core POWER7+ x 2



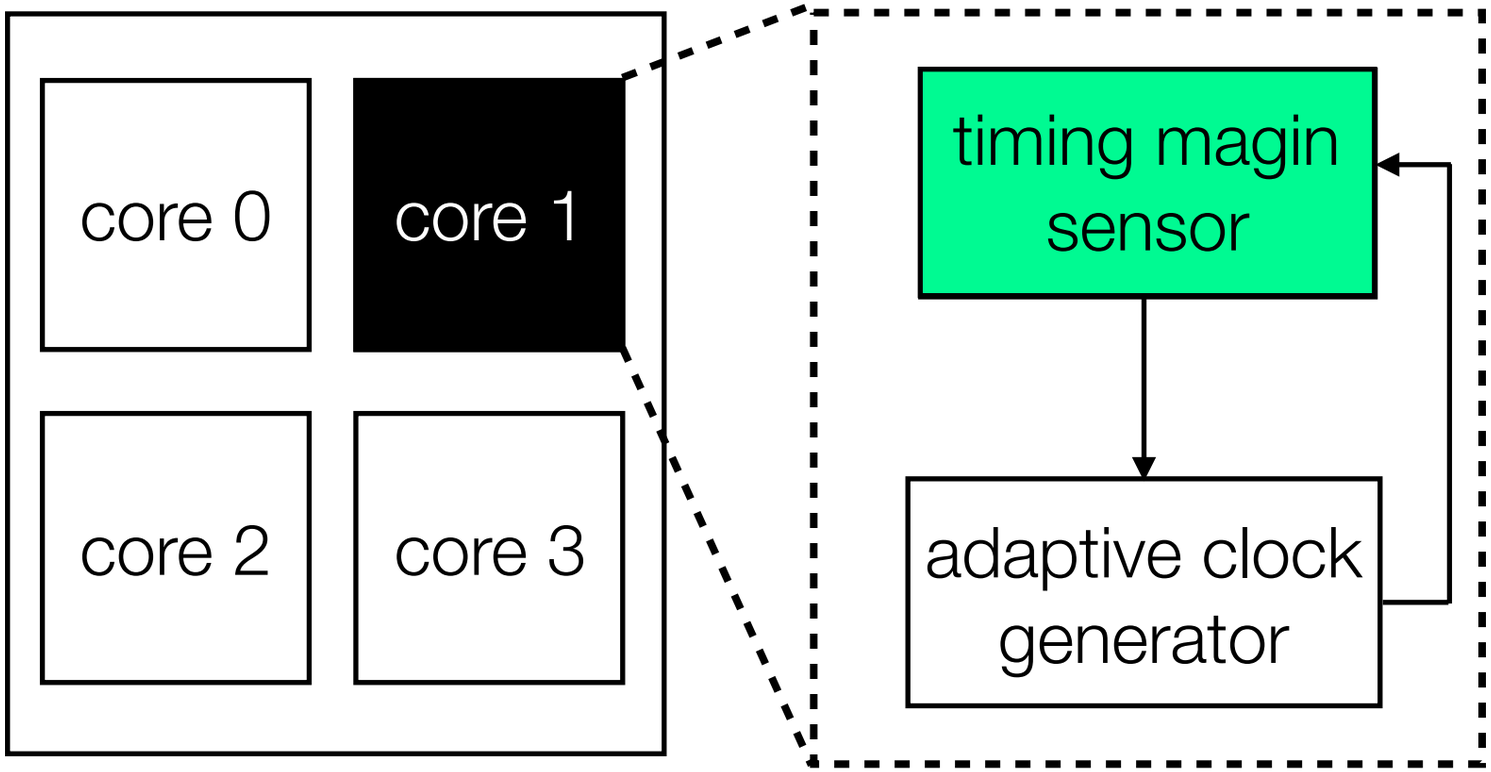
Experimental Setup: IBM[R] Eight-core POWER7+ x 2



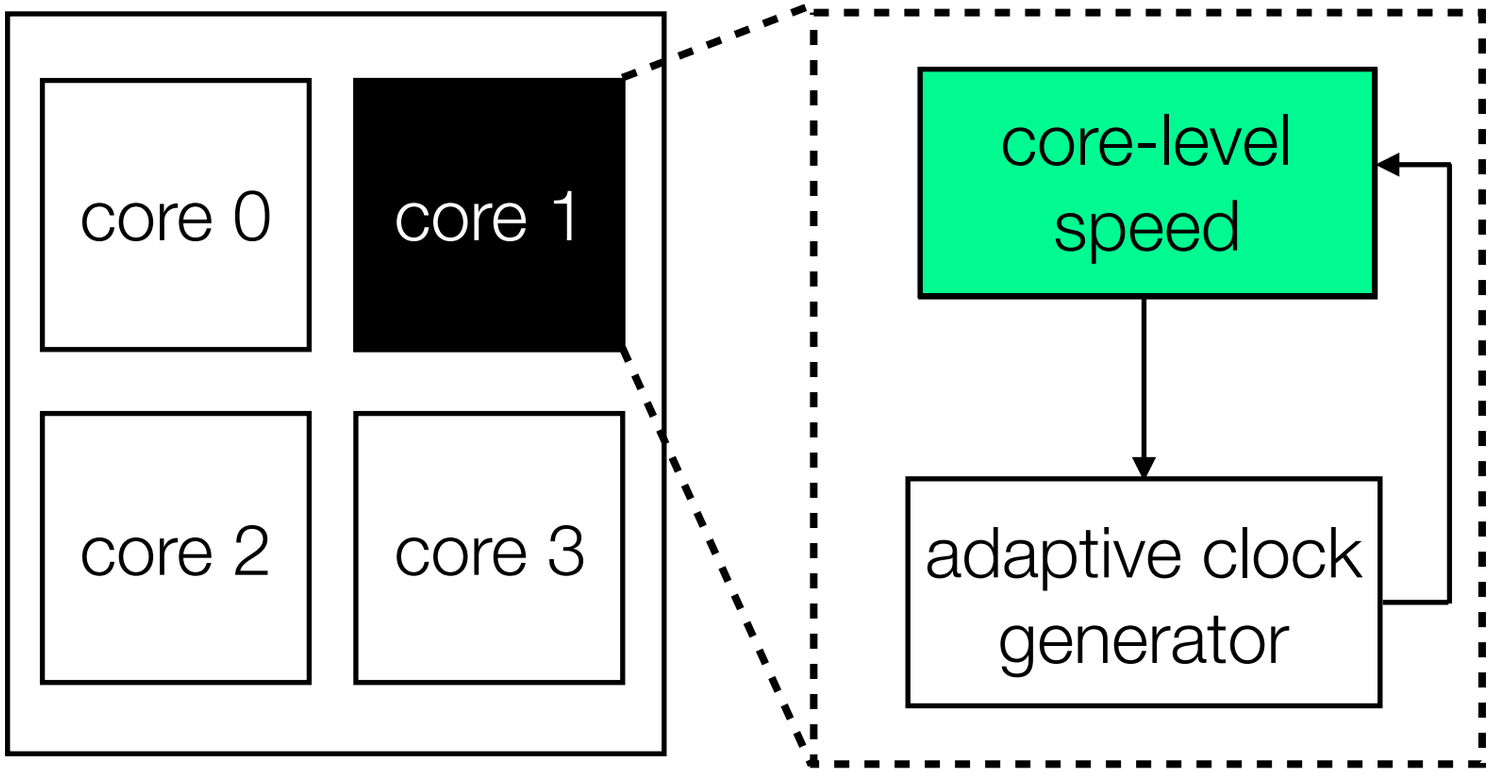
Expose Core-level Performance with ATM



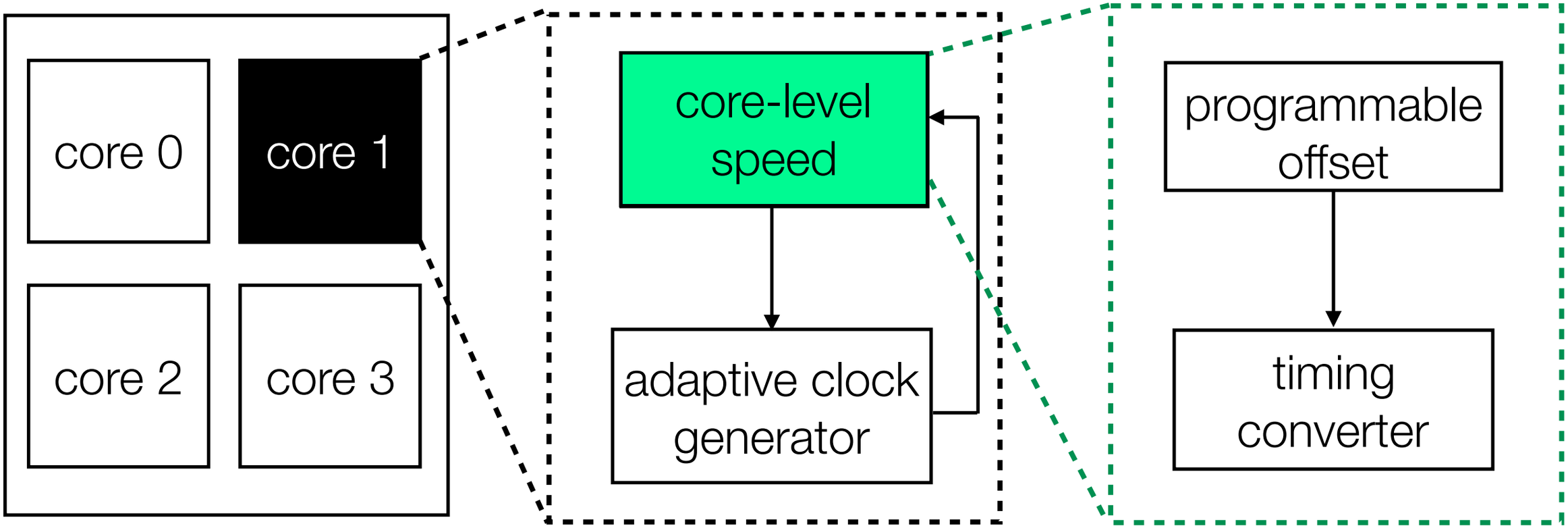
Expose Core-level Performance with ATM



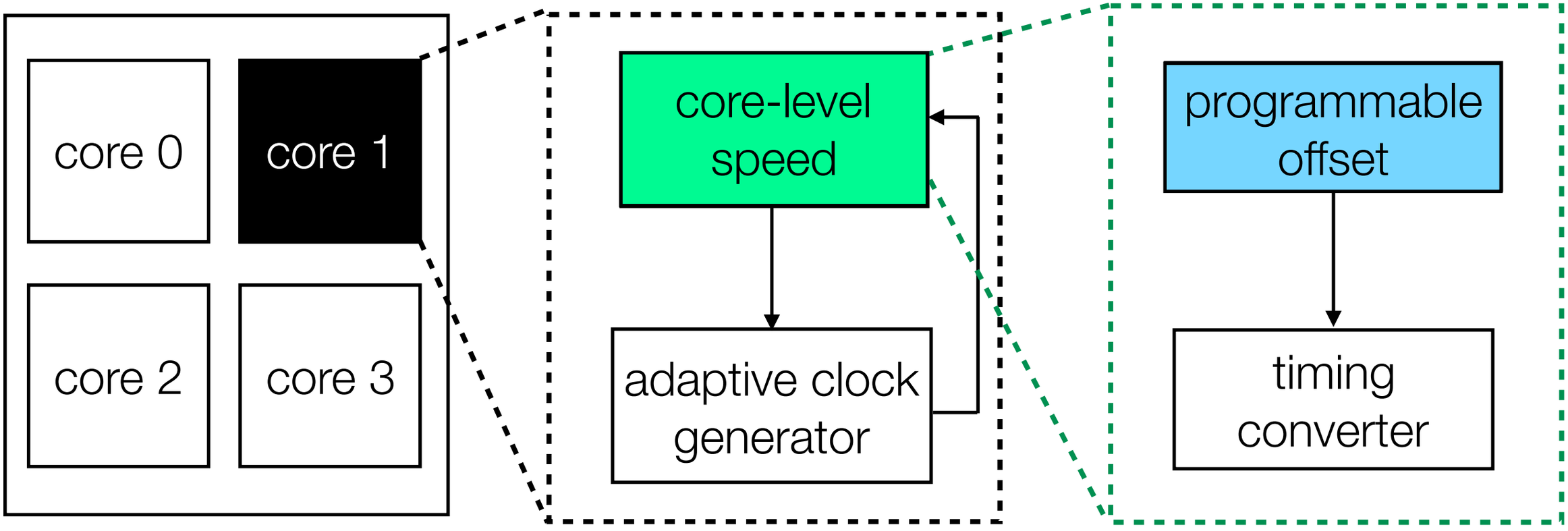
Expose Core-level Performance with ATM



Expose Core-level Performance with ATM

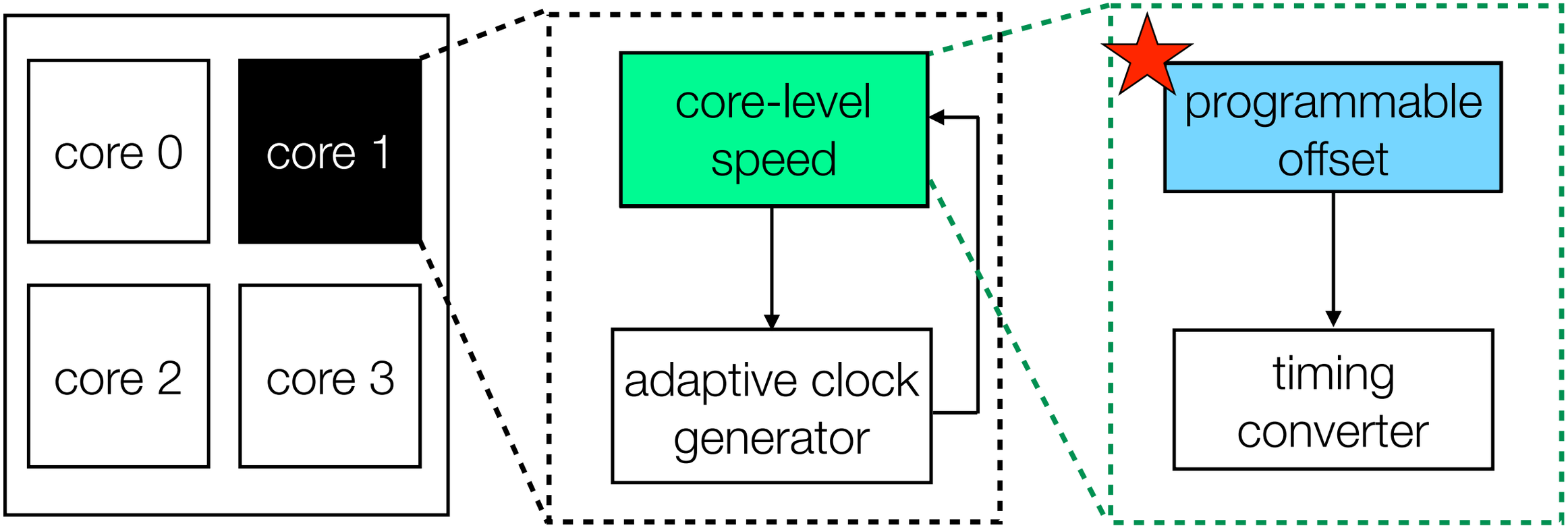


Expose Core-level Performance with ATM

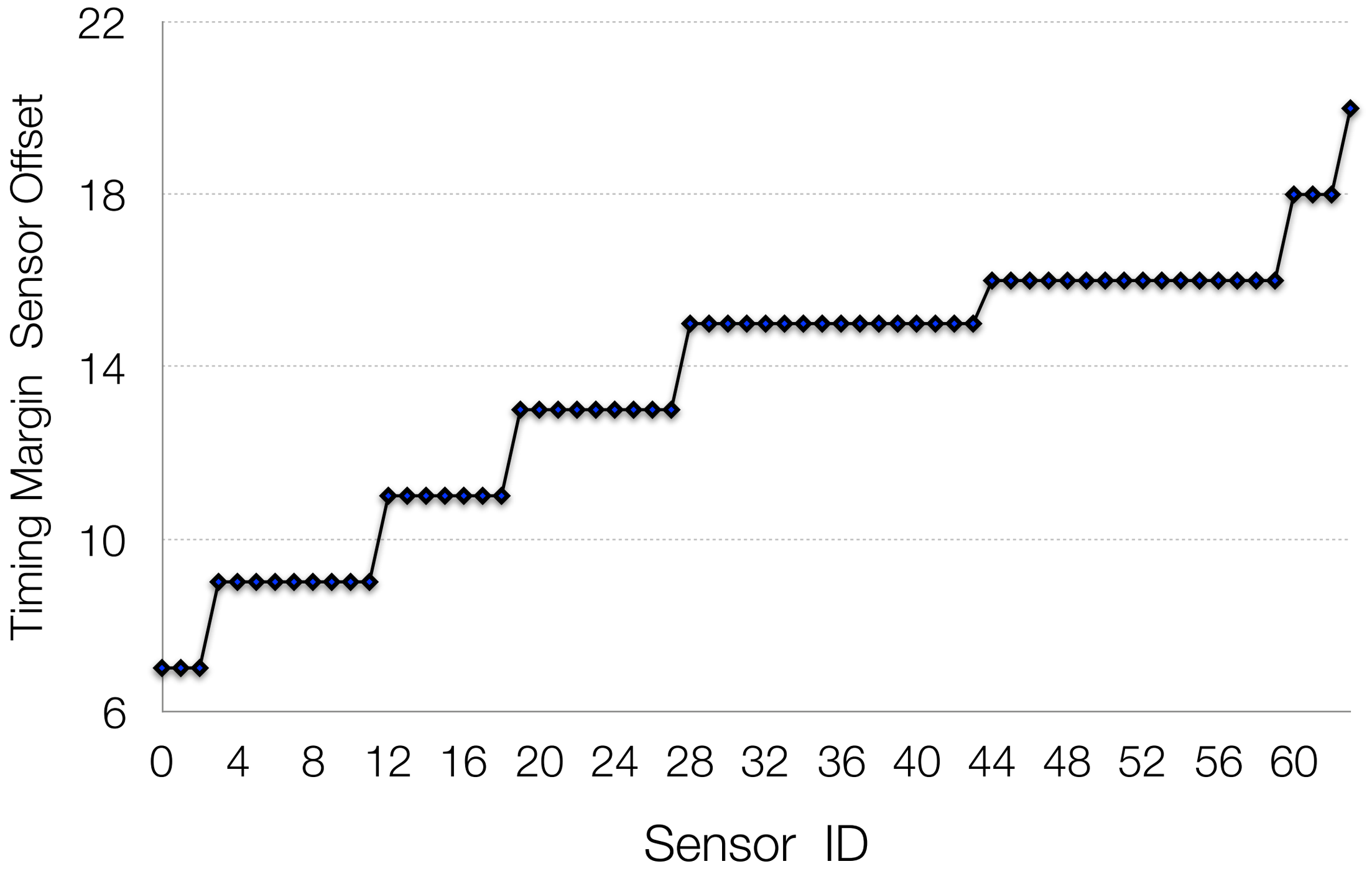


Expose Core-level Performance with ATM

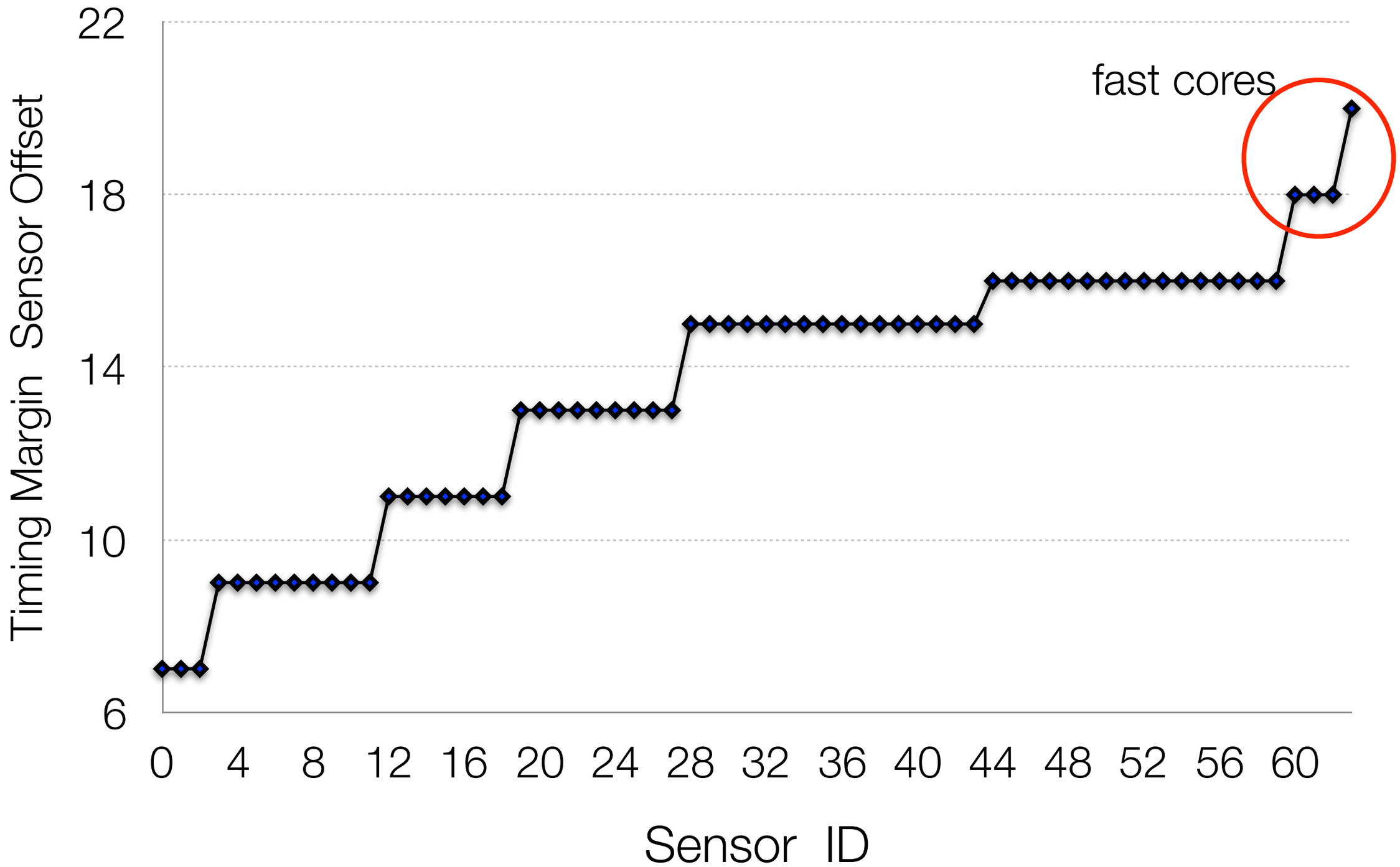
preset measurement offset
to hide process variation



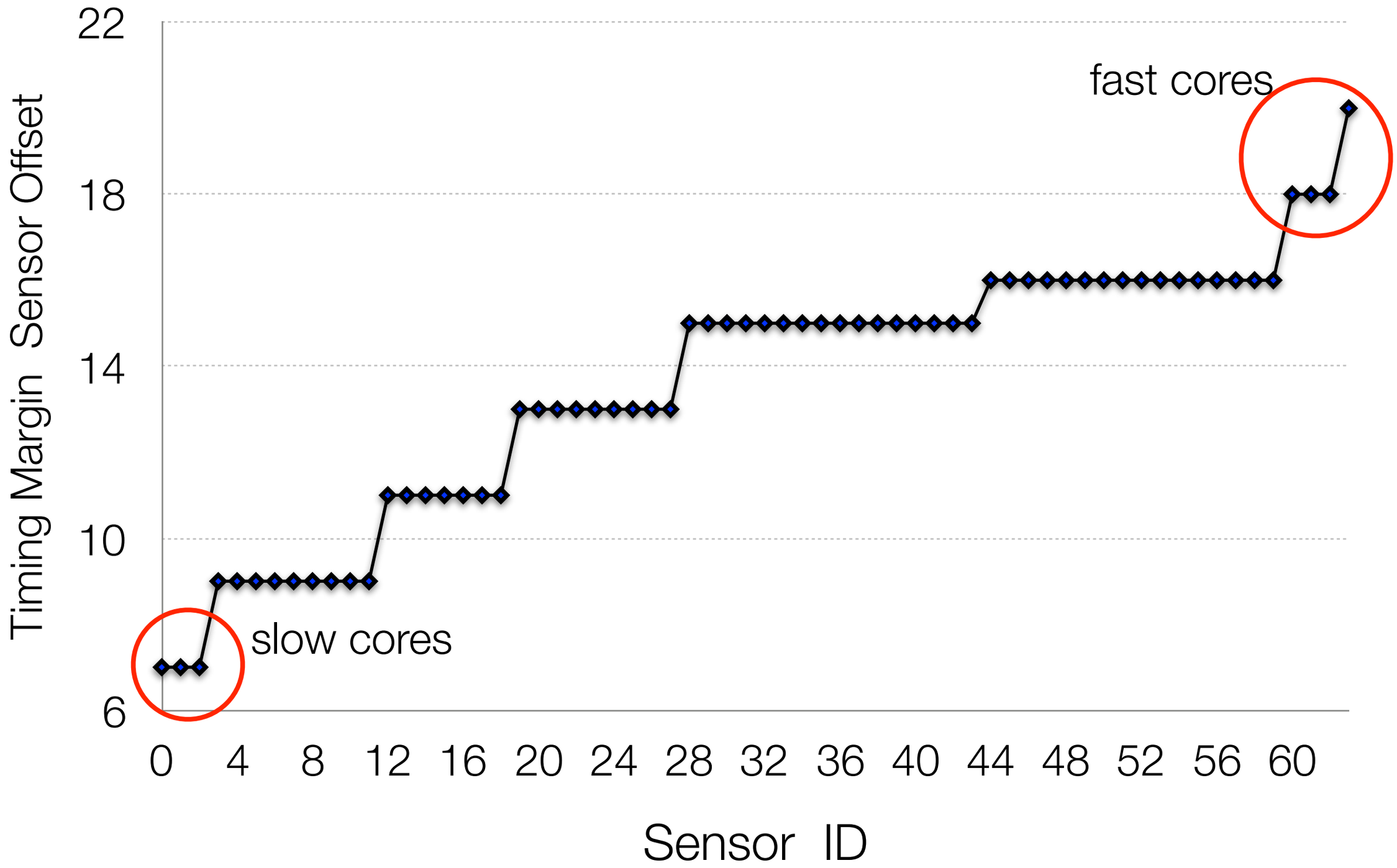
Preset Timing Margin Measurement Offset



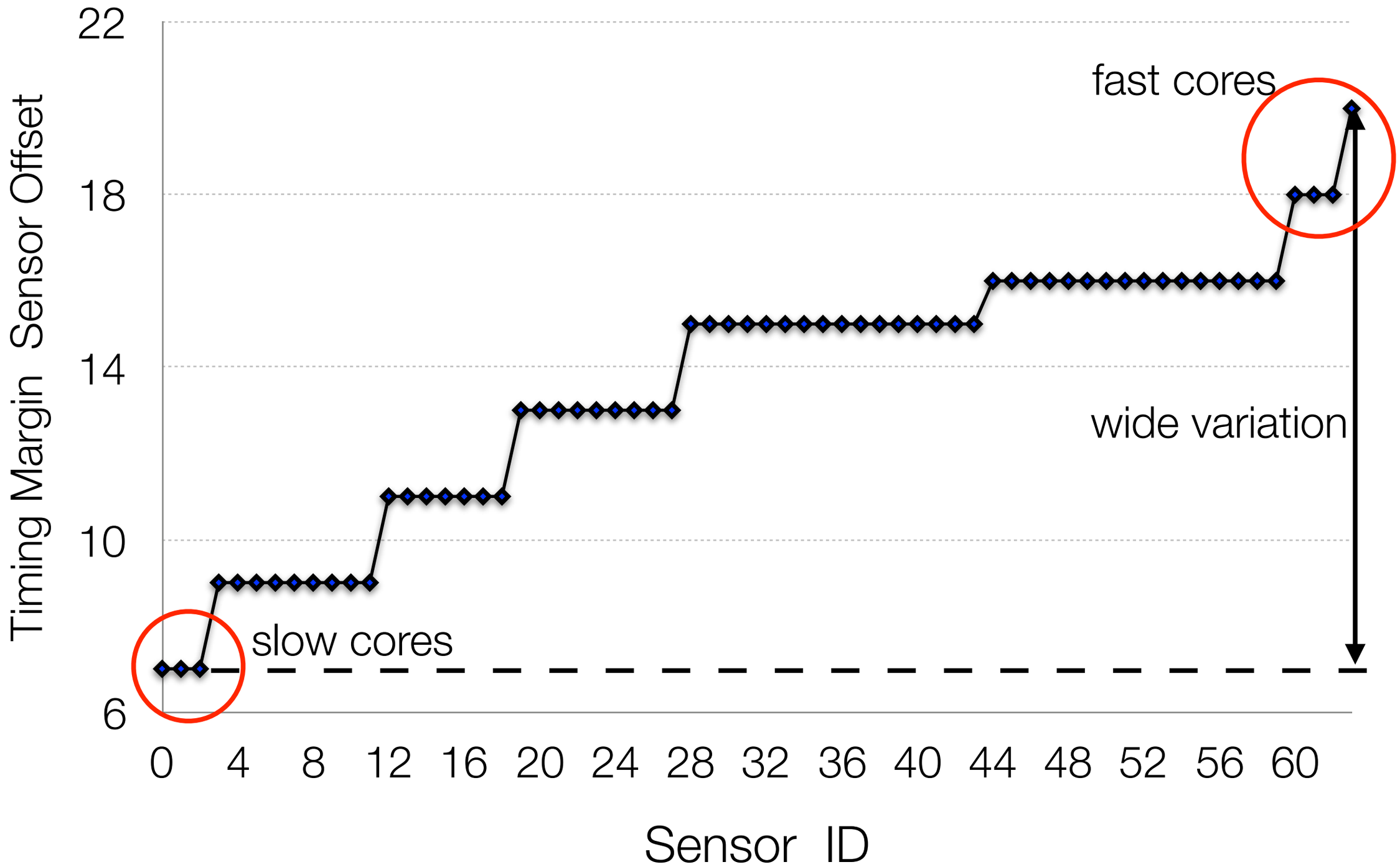
Preset Timing Margin Measurement Offset



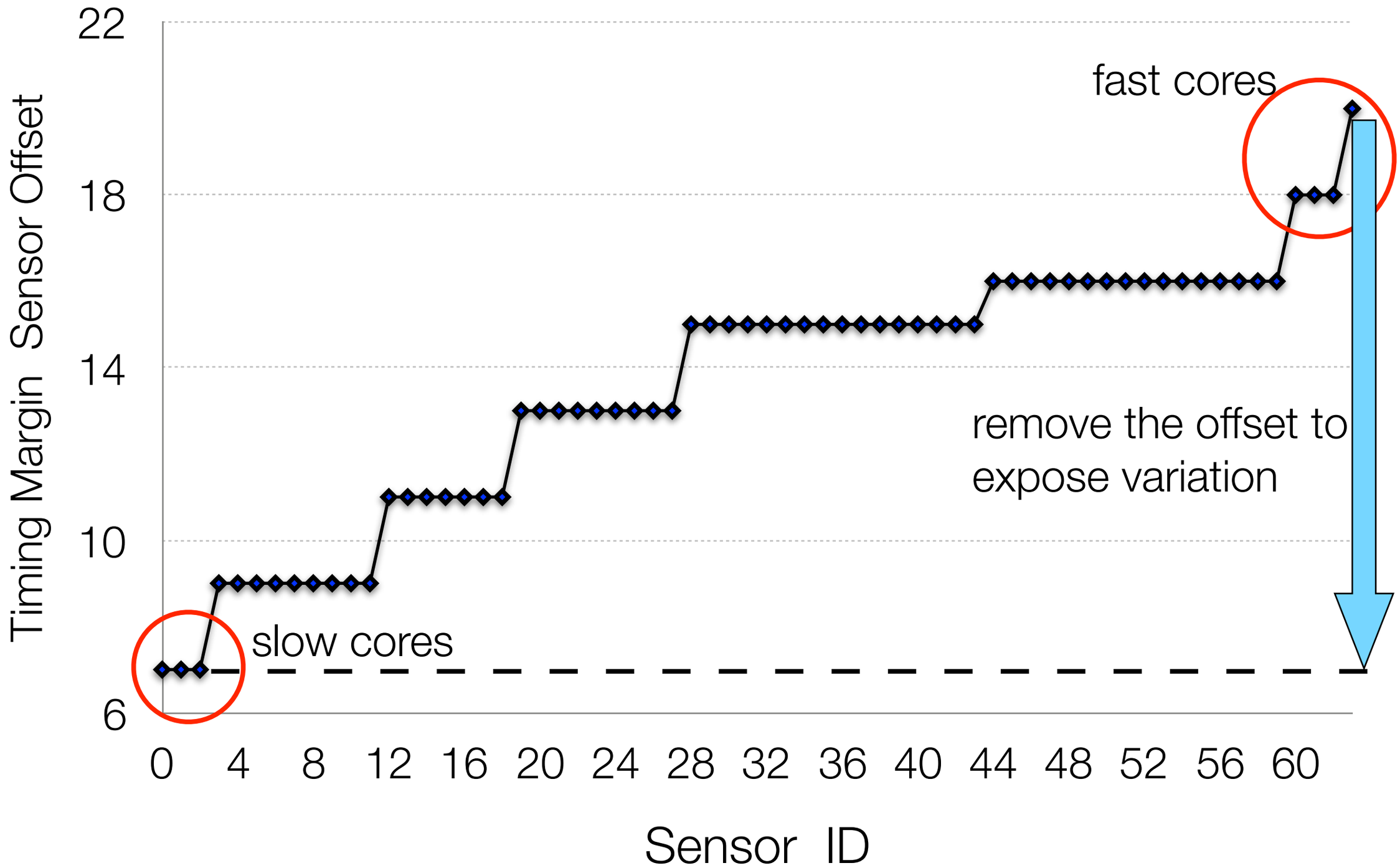
Preset Timing Margin Measurement Offset



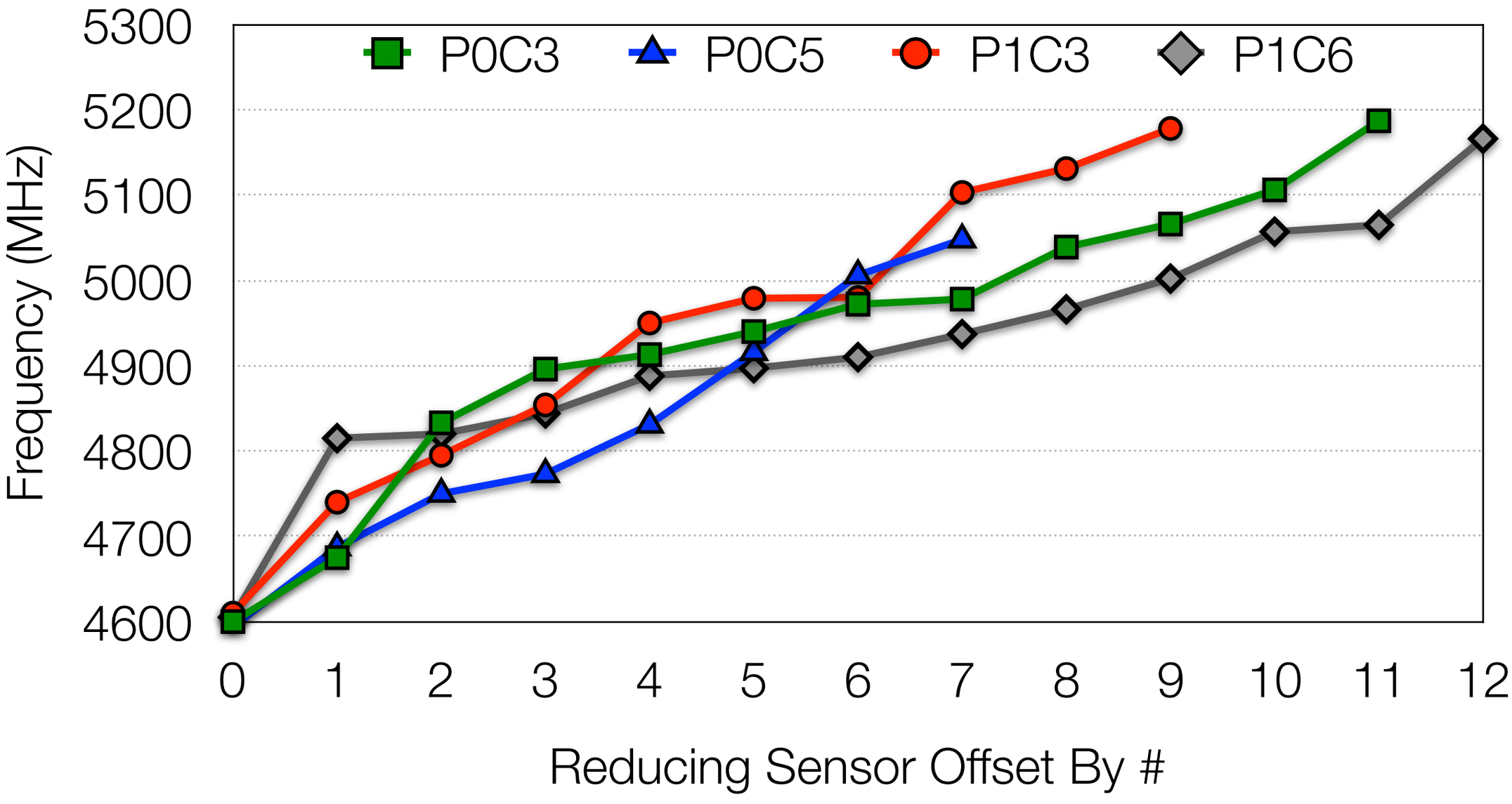
Preset Timing Margin Measurement Offset



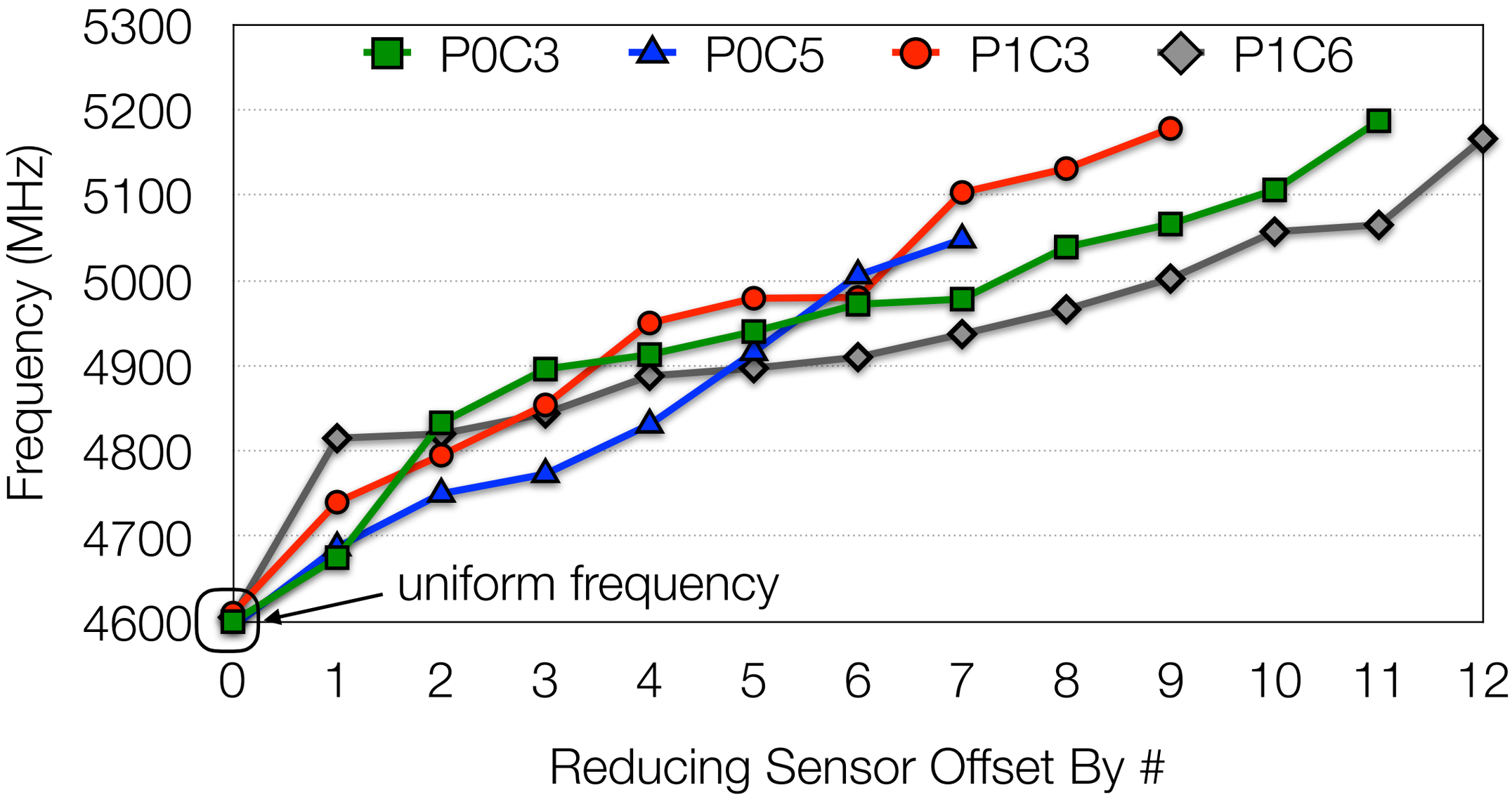
Preset Timing Margin Measurement Offset



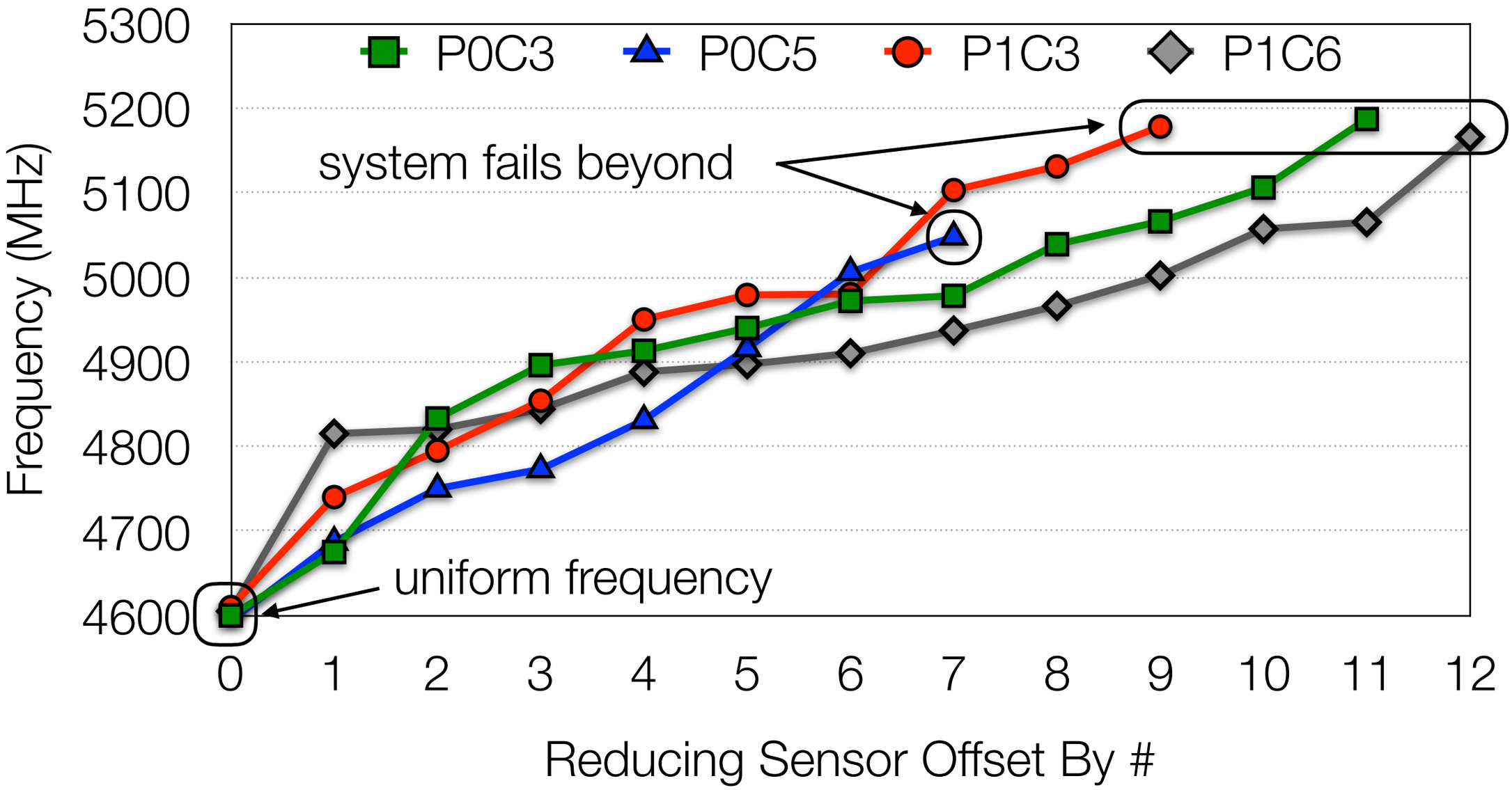
Re-Program Timing Margin Sensor to Fine-tune Core-level Performance



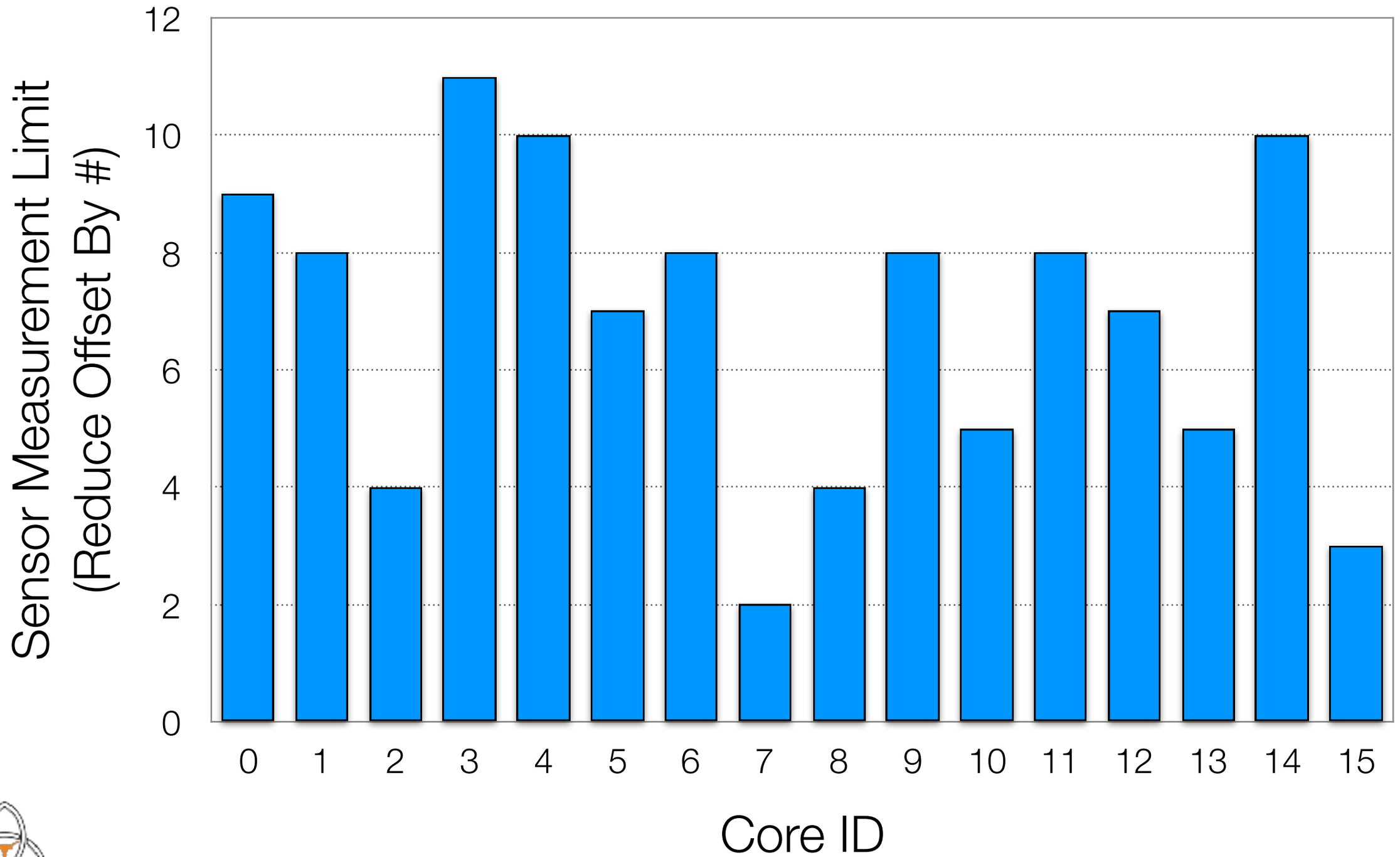
Re-Program Timing Margin Sensor to Fine-tune Core-level Performance



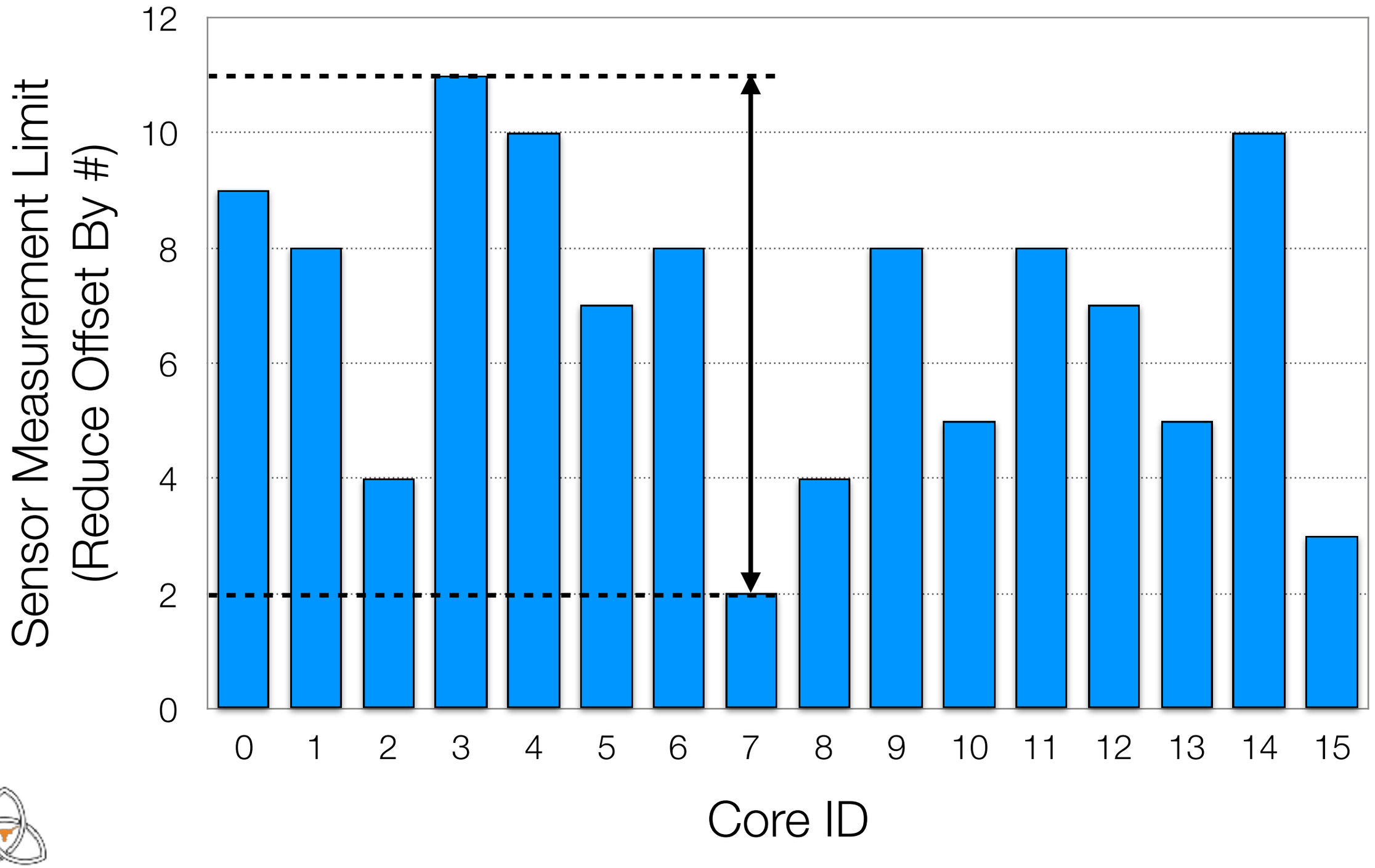
Re-Program Timing Margin Sensor to Fine-tune Core-level Performance



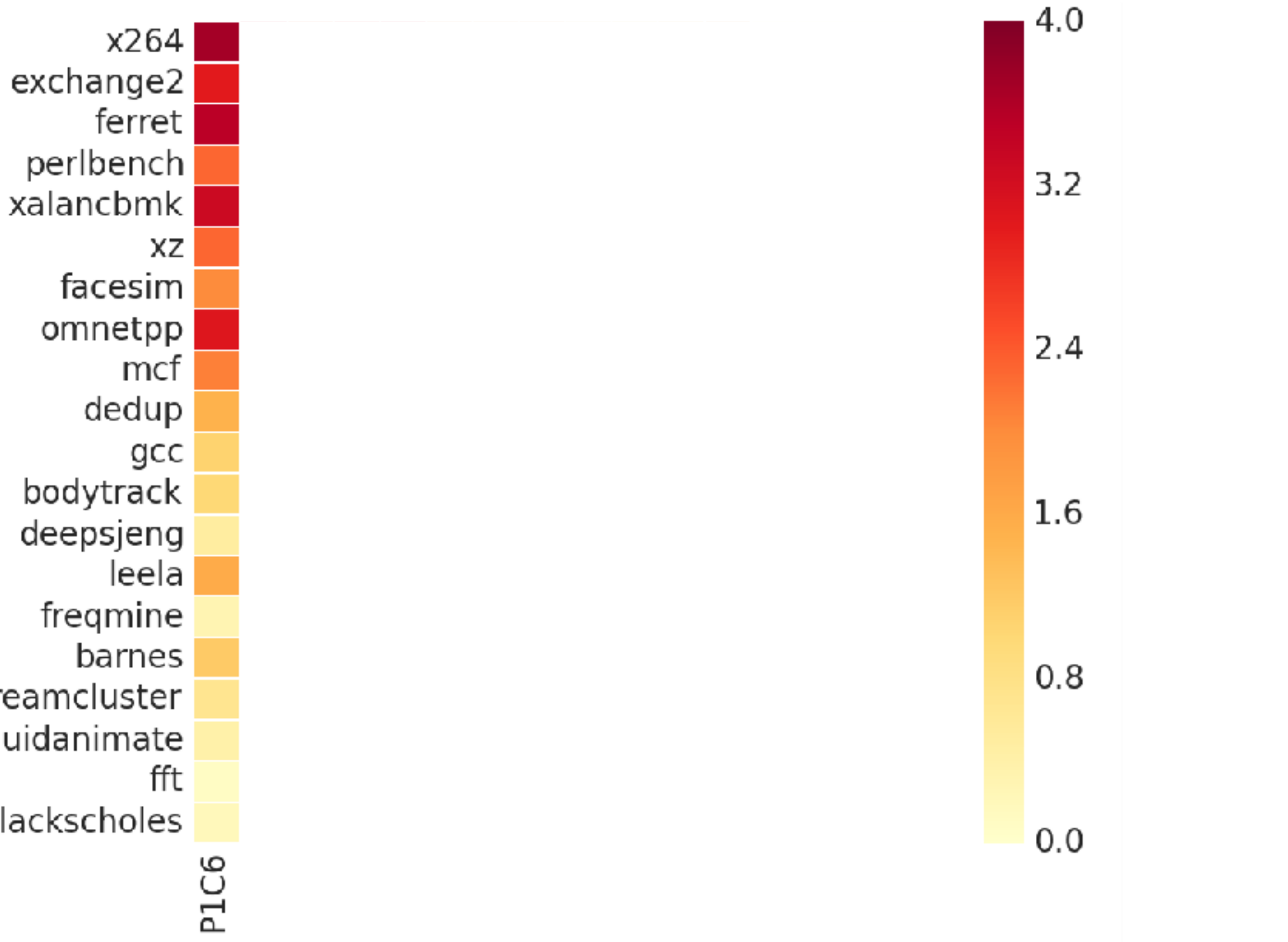
Core-to-core Sensor Configuration variation



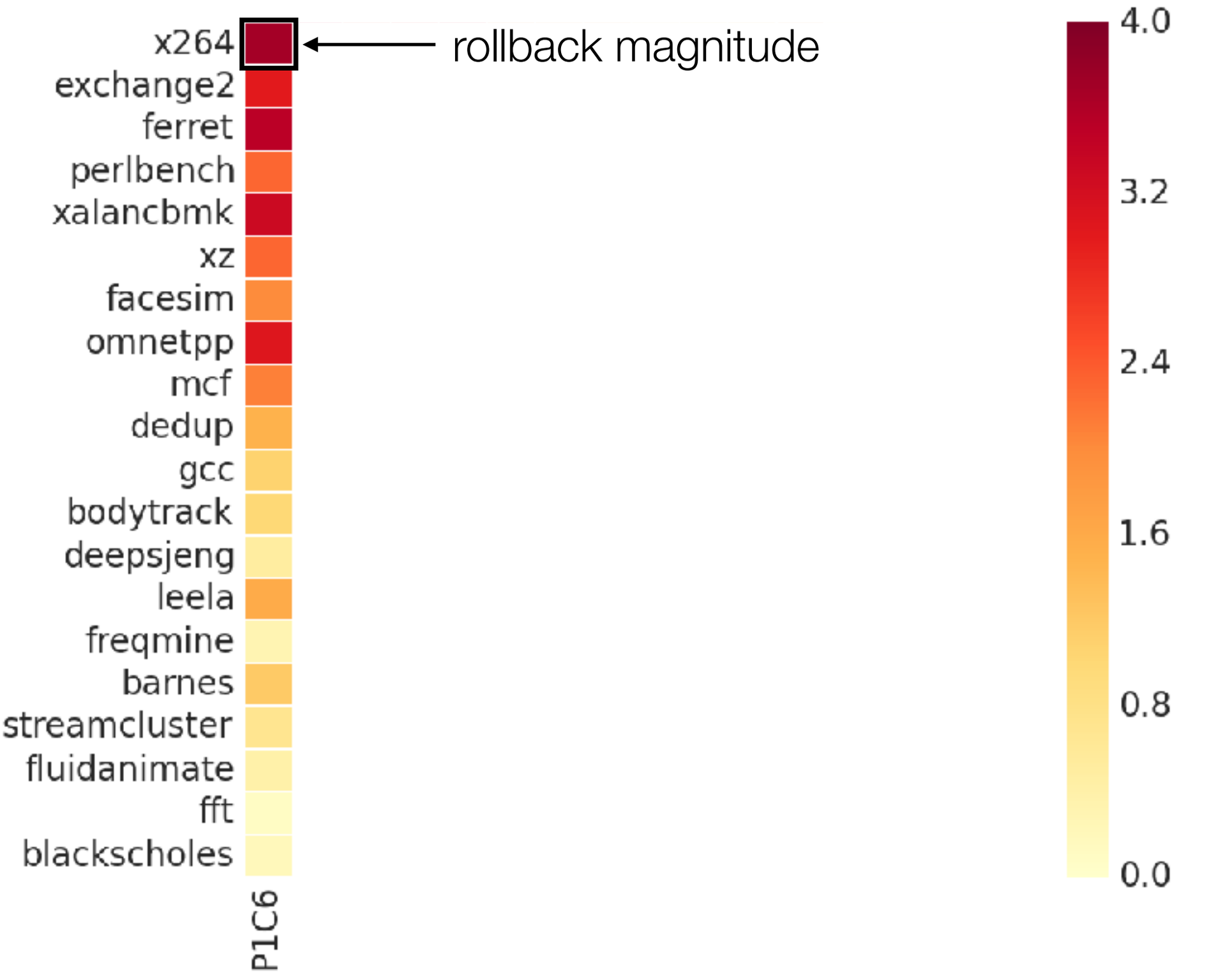
Core-to-core Sensor Configuration variation



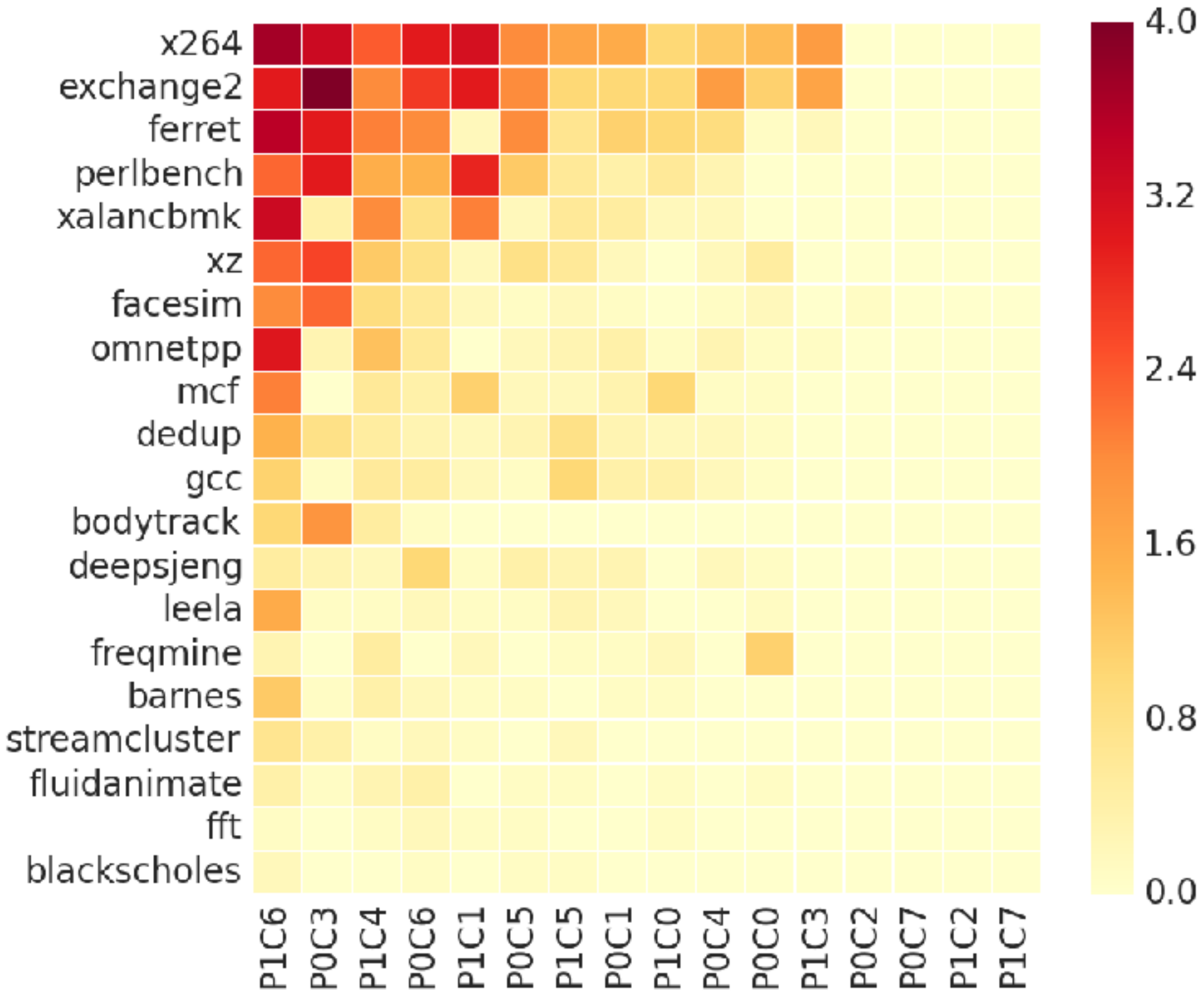
Running applications: Rollback from System Idle Limits



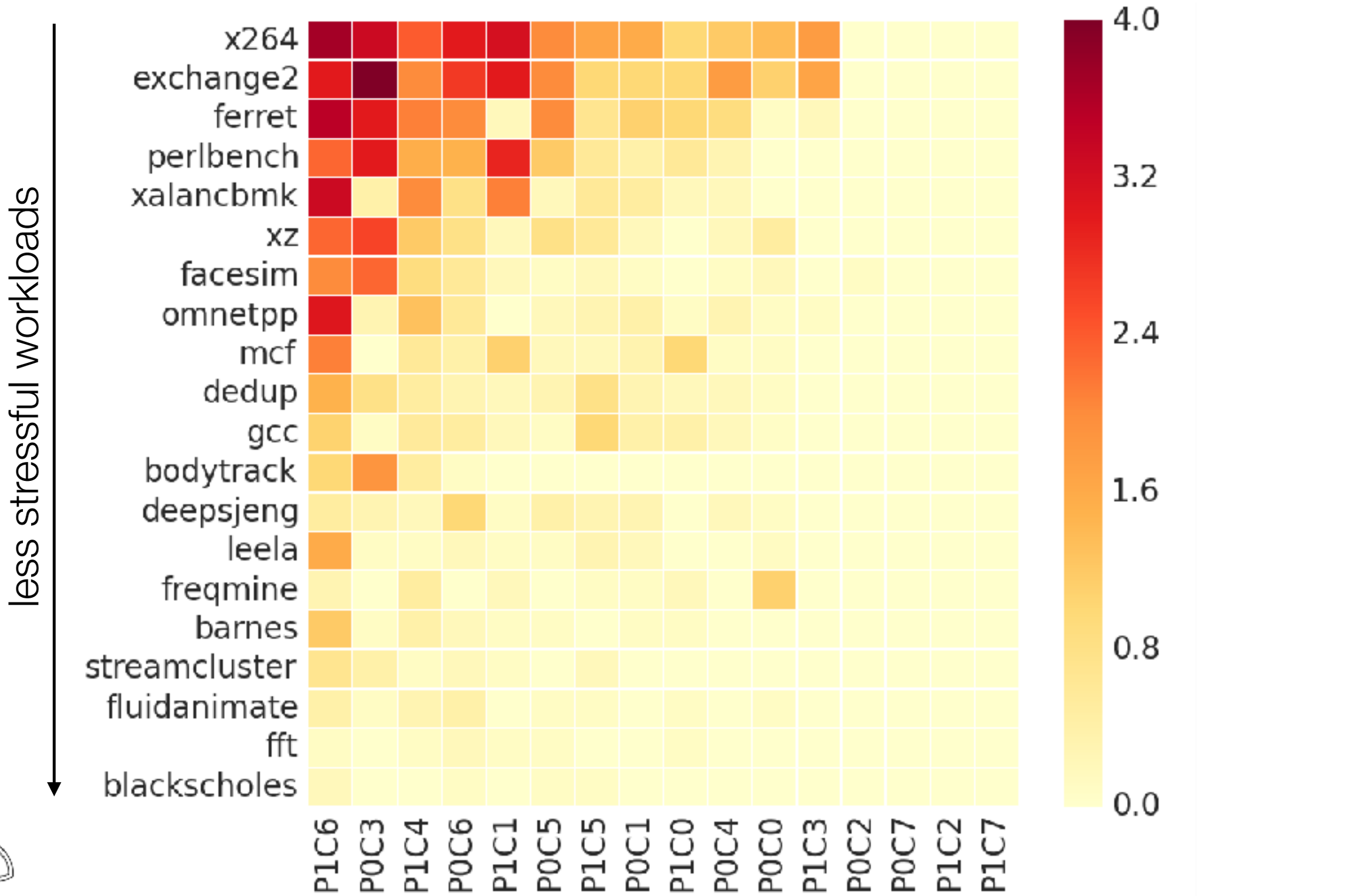
Running applications: Rollback from System Idle Limits



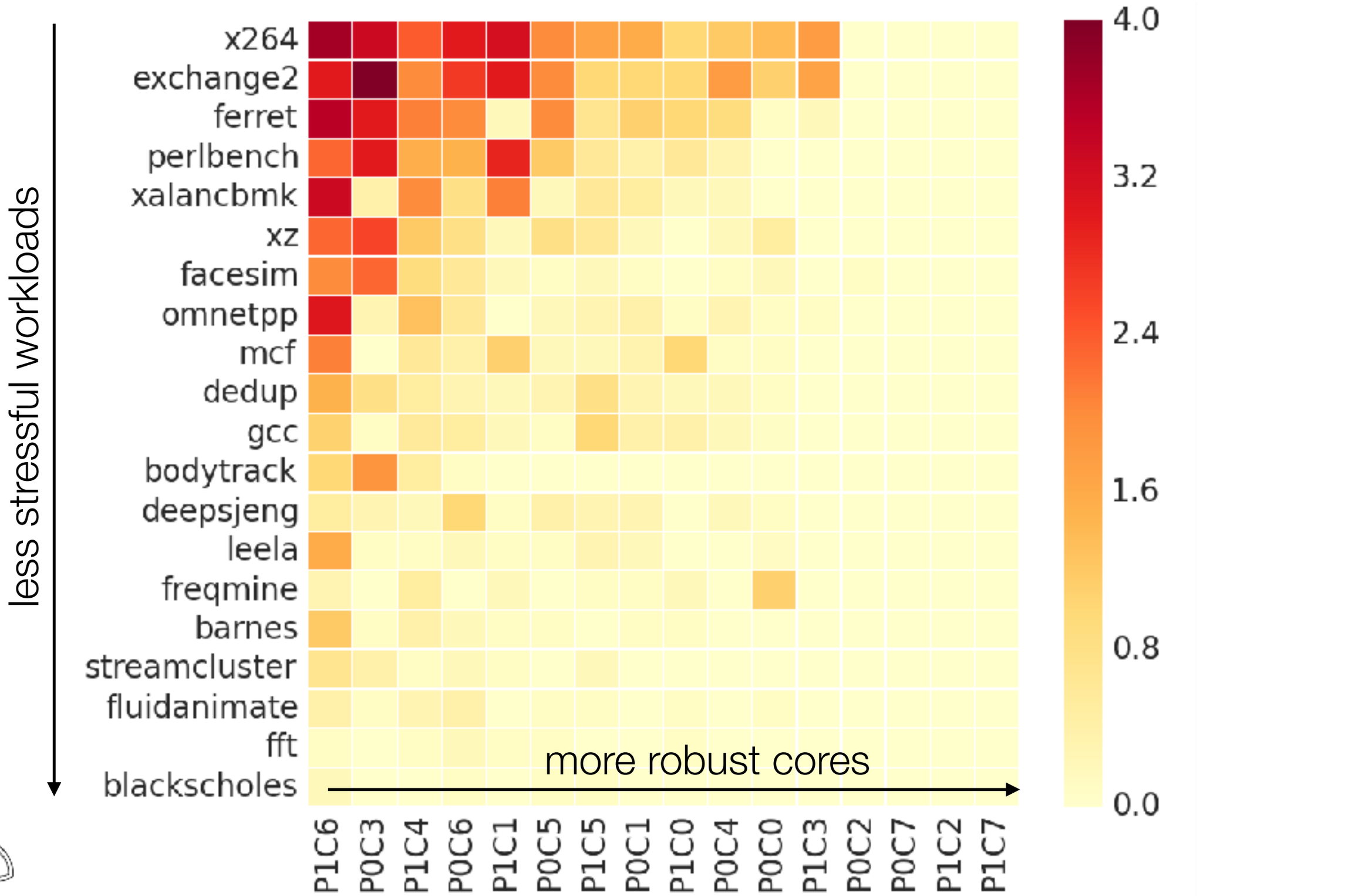
Running applications: Rollback from System Idle Limits



Running applications: Rollback from System Idle Limits



Running applications: Rollback from System Idle Limits



Fine-tuning Active Timing Margin

- Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity		
improvemnt		

Fine-tuning Active Timing Margin

- Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity	high	
improvemnt		

Fine-tuning Active Timing Margin

- Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity	high	di/dt effect, input data, instruction coverage, etc.
improvemnt		

Fine-tuning Active Timing Margin

- Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity	high	di/dt effect, input data, instruction coverage, etc.
improvemnt	low	

Fine-tuning Active Timing Margin

- Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity	high	di/dt effect, input data, instruction coverage, etc.
improvemnt	low	~50 MHz average increase

Fine-tuning Active Timing Margin

✗ Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity	high	di/dt effect, input data, instruction coverage, etc.
improvemnt	low	~50 MHz average increase

Fine-tuning Active Timing Margin

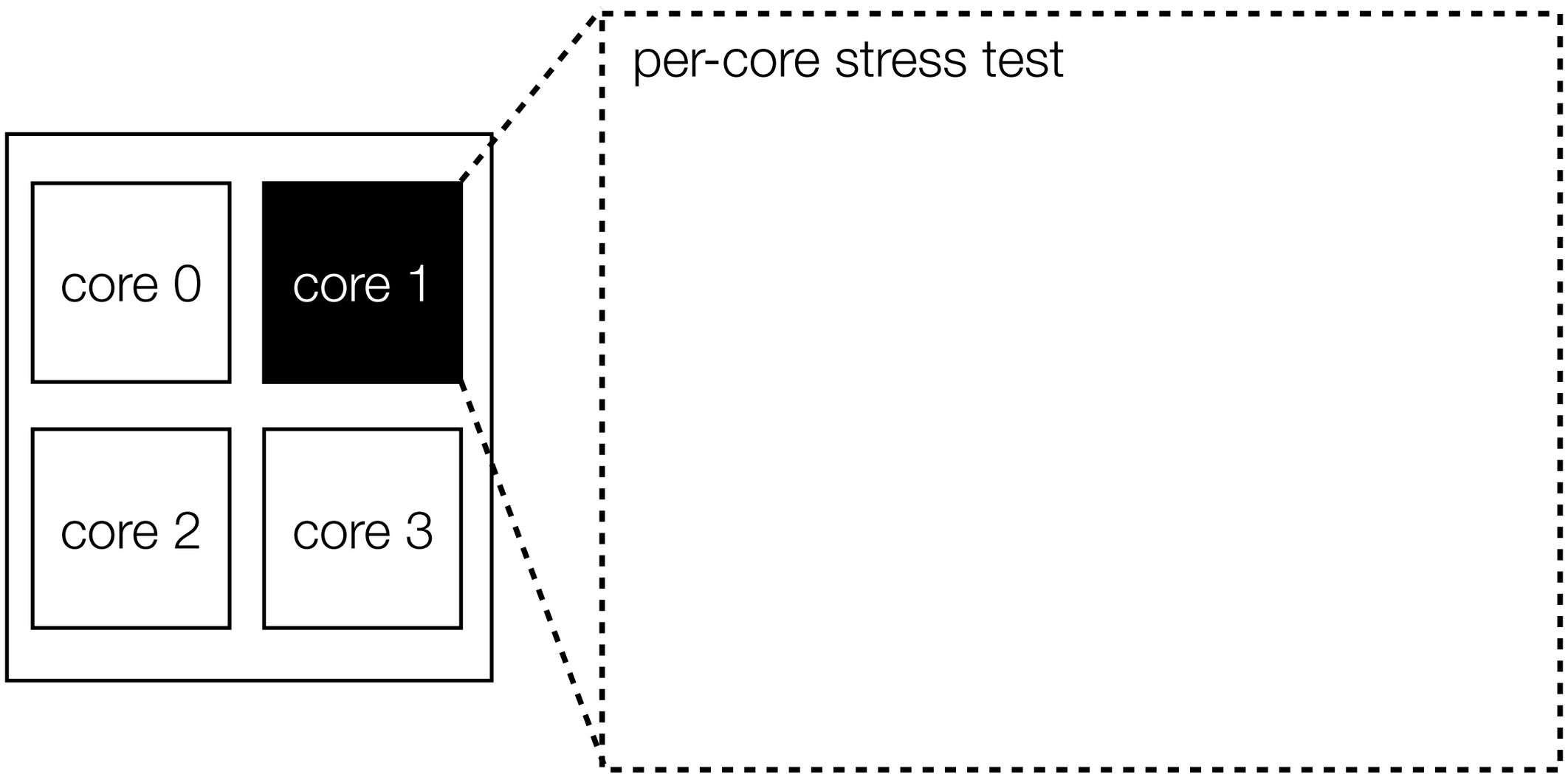
✗ Option 1: predict ATM config for each <App, Core>

	Evaluation	Reason
complexity	high	di/dt effect, input data, instruction coverage, etc.
improvemnt	low	~50 MHz average increase

✓ Option 2: apply one configuration for all apps

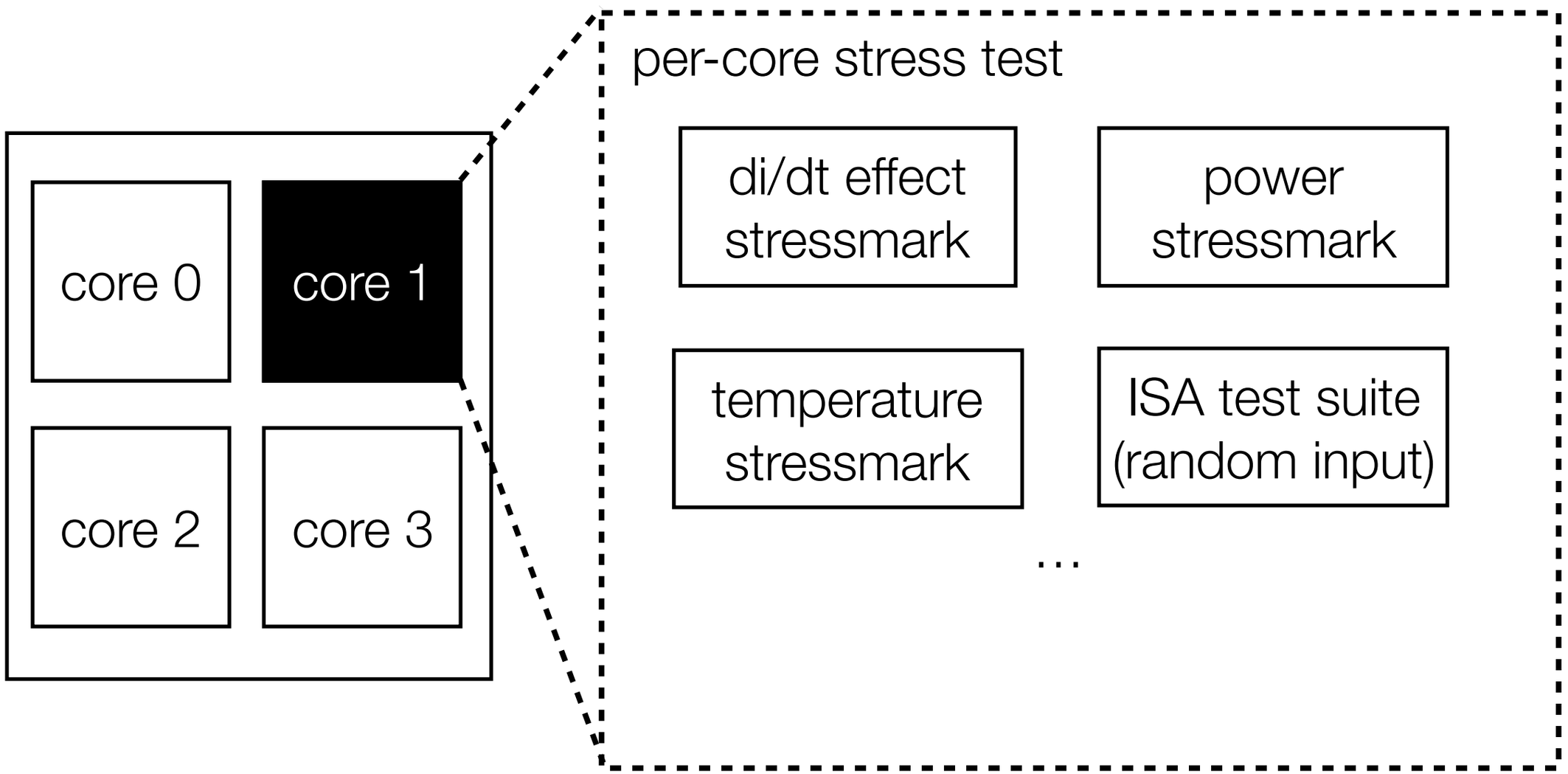
Fine-tuning at Test-time

- Set timing margin sensor offset with core-level stress test



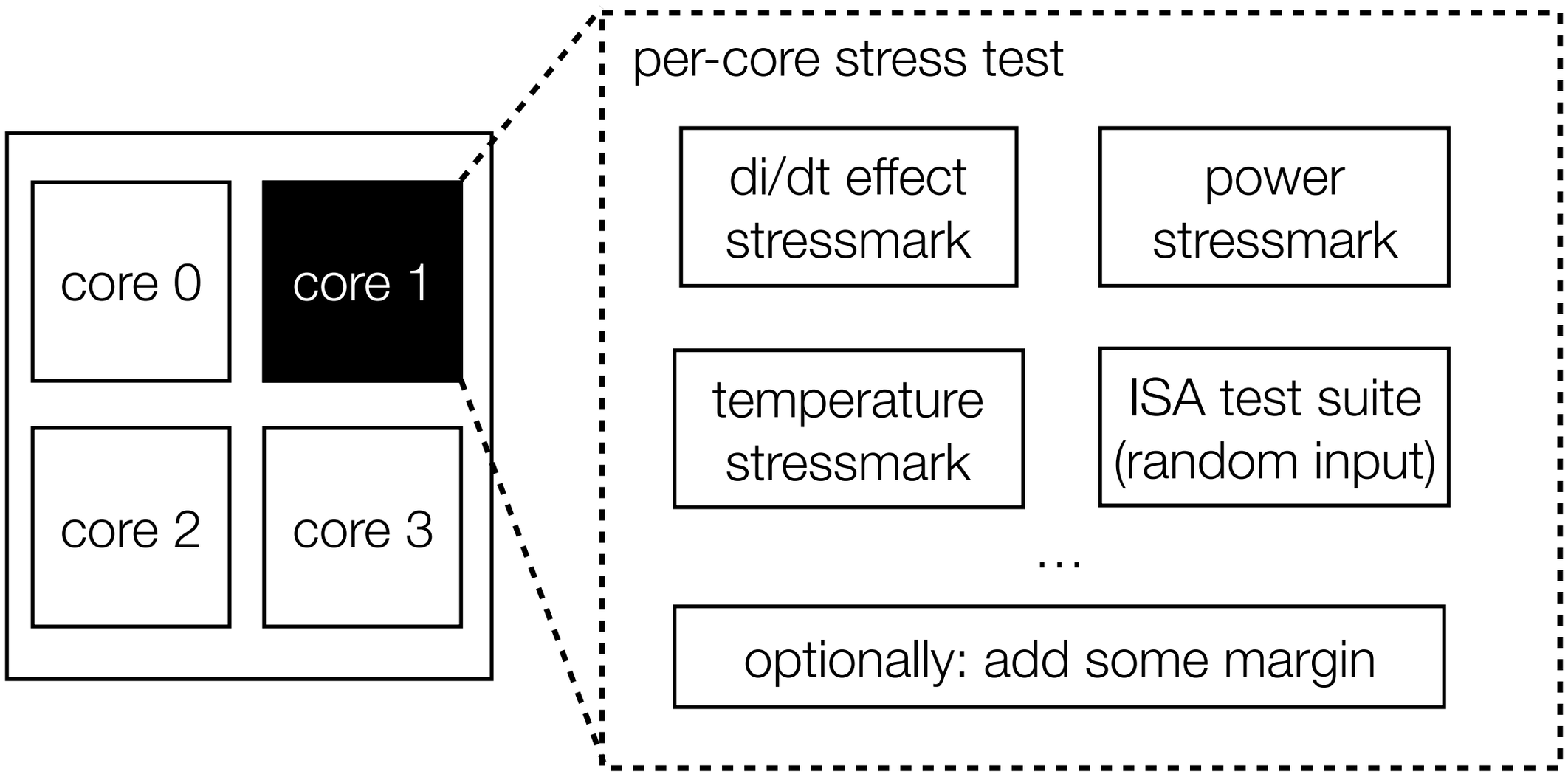
Fine-tuning at Test-time

- Set timing margin sensor offset with core-level stress test

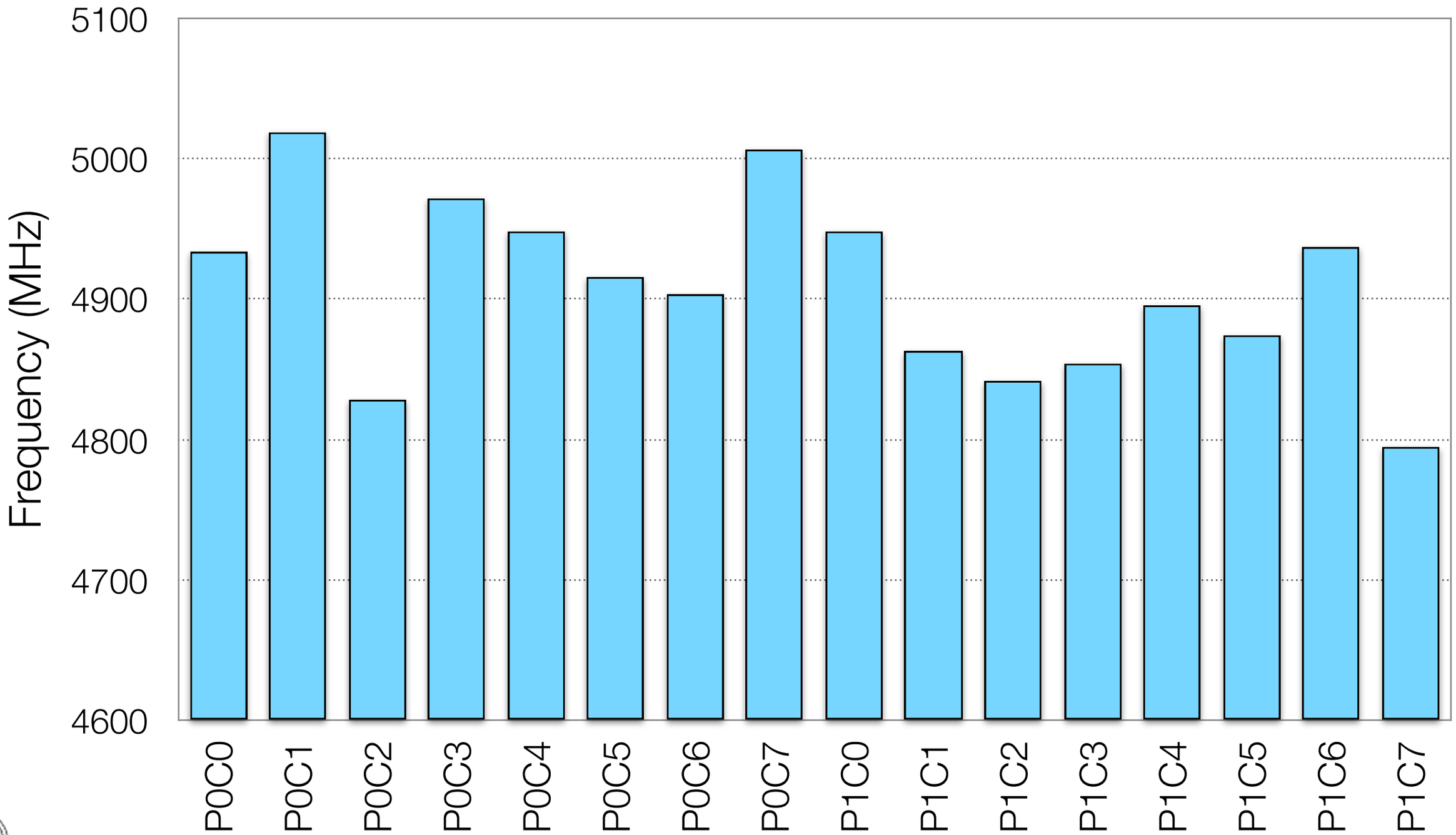


Fine-tuning at Test-time

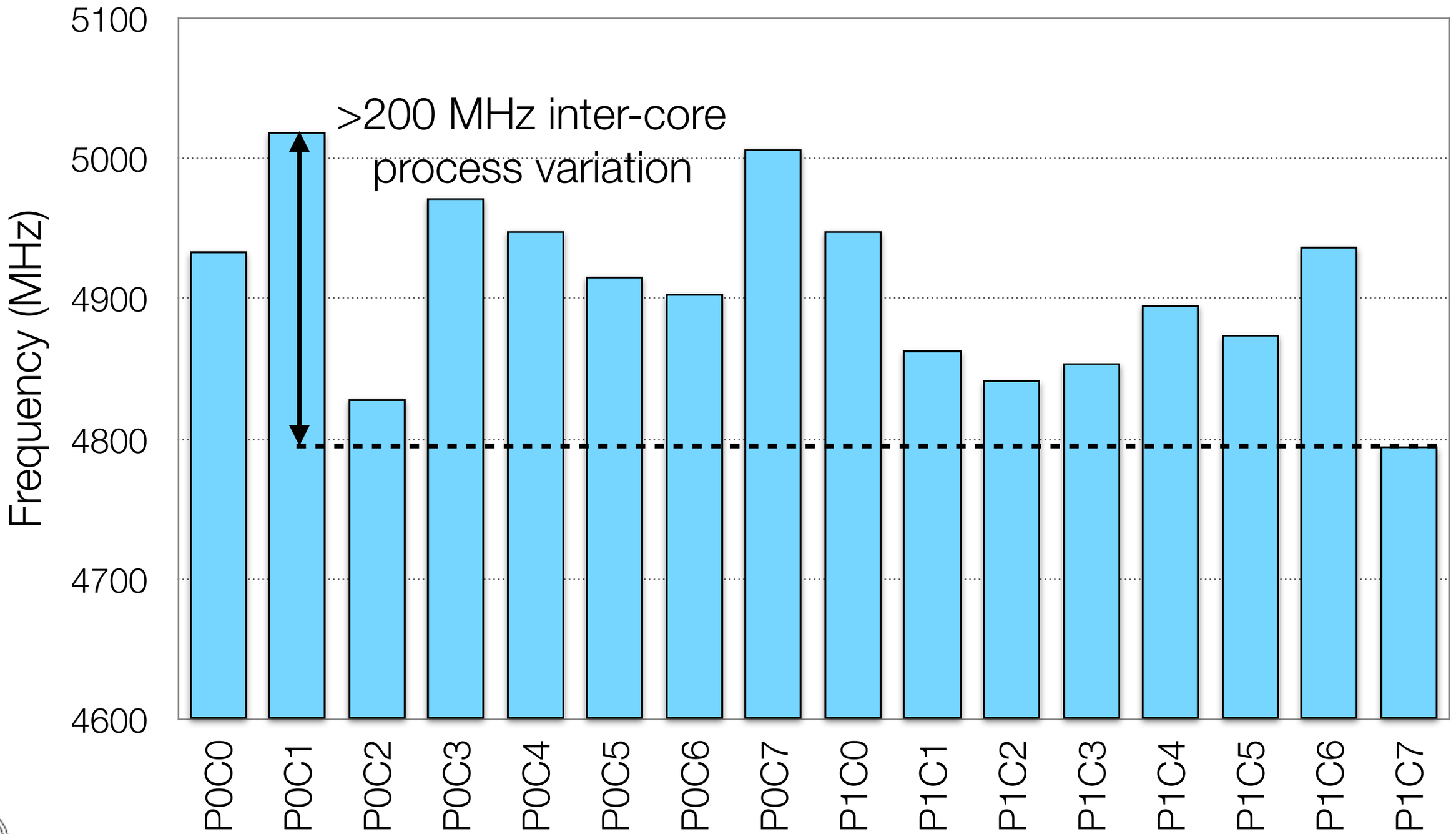
- Set timing margin sensor offset with core-level stress test



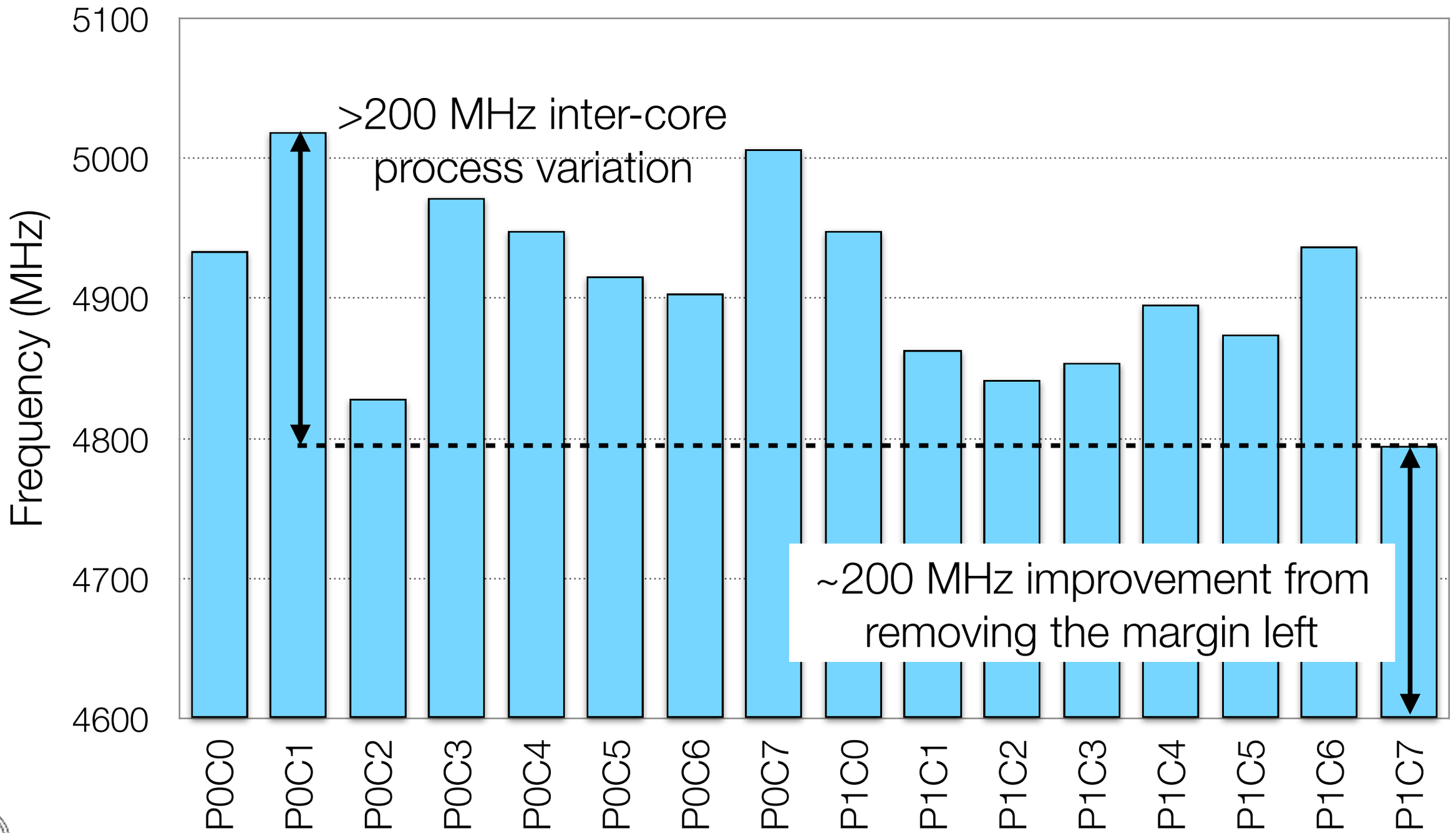
Fine-tuning ATM to Expose Variation



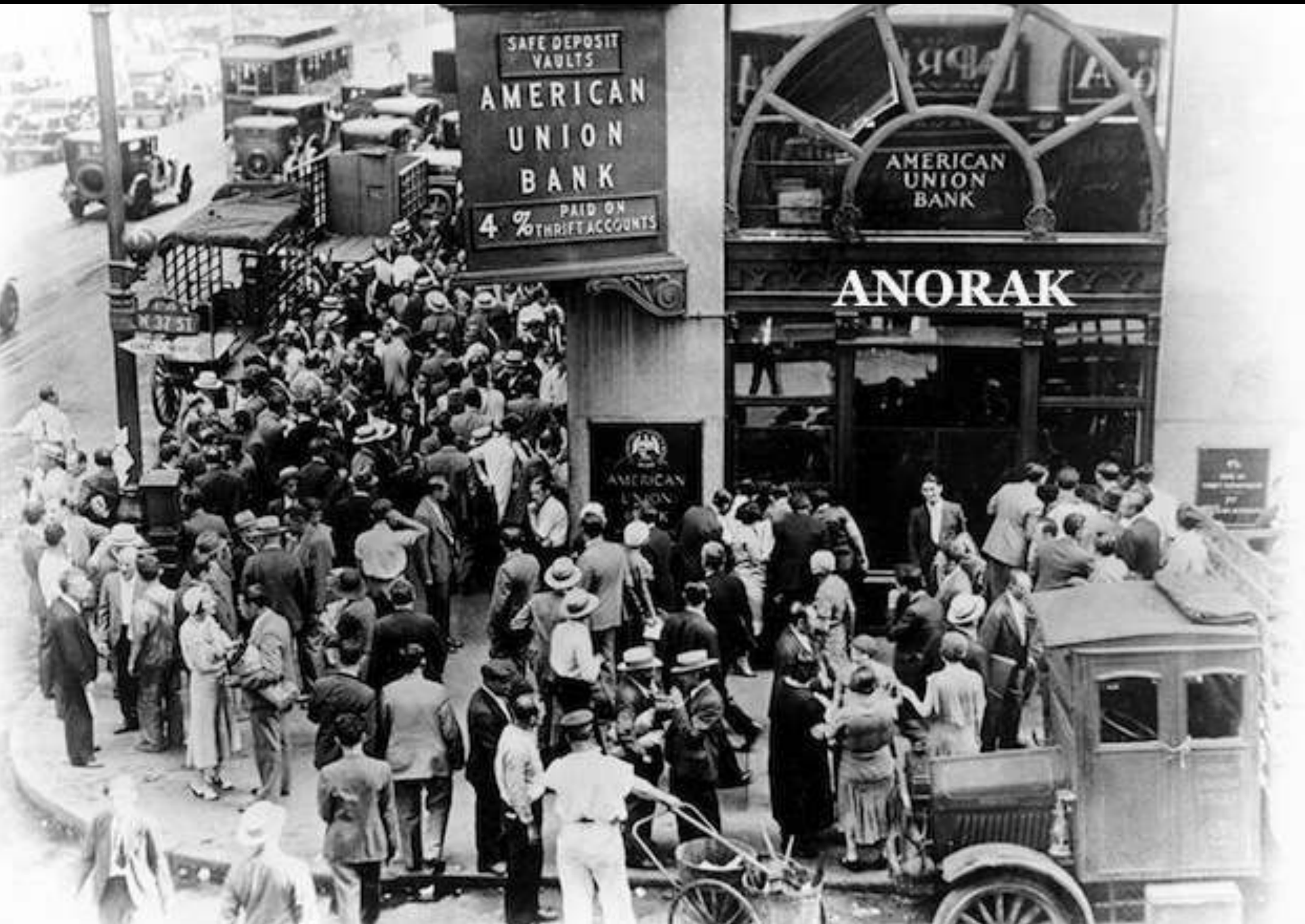
Fine-tuning ATM to Expose Variation



Fine-tuning ATM to Expose Variation



“Bank Run” on Active Timing Margin

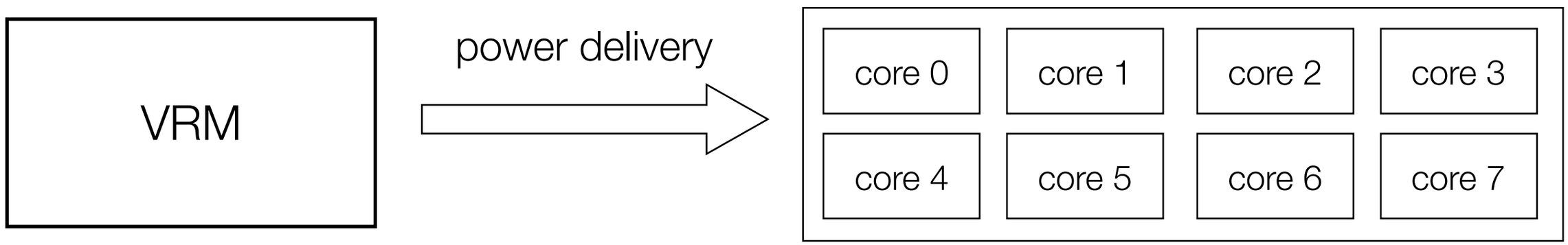


Executive Summary

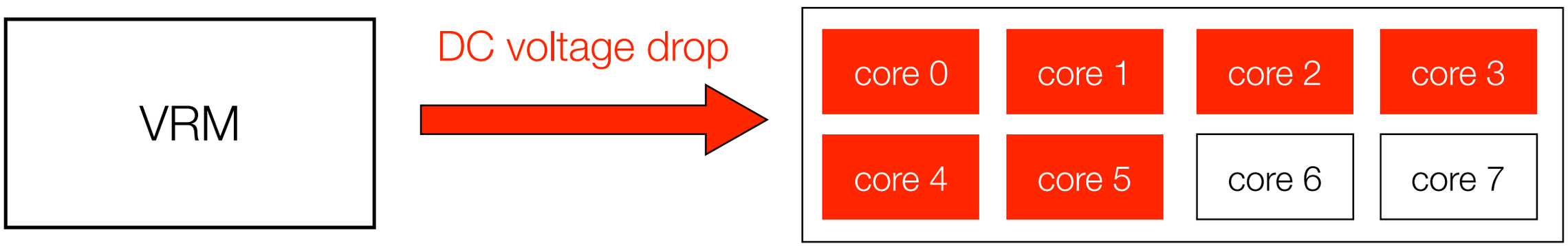
- Fine-tune core-level ATM to **automatically** expose inter-core performance variation

- Expose chip-wide performance variation caused by heterogenous workload intensity

“Bank Run” on Active Timing Margin

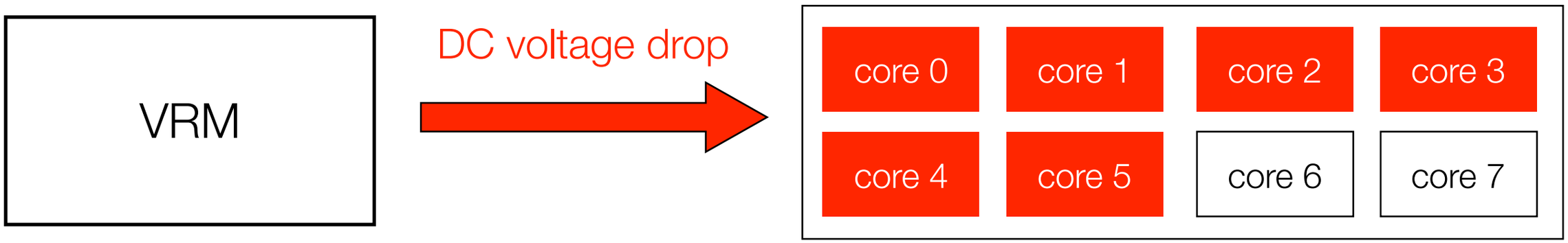


“Bank Run” on Active Timing Margin



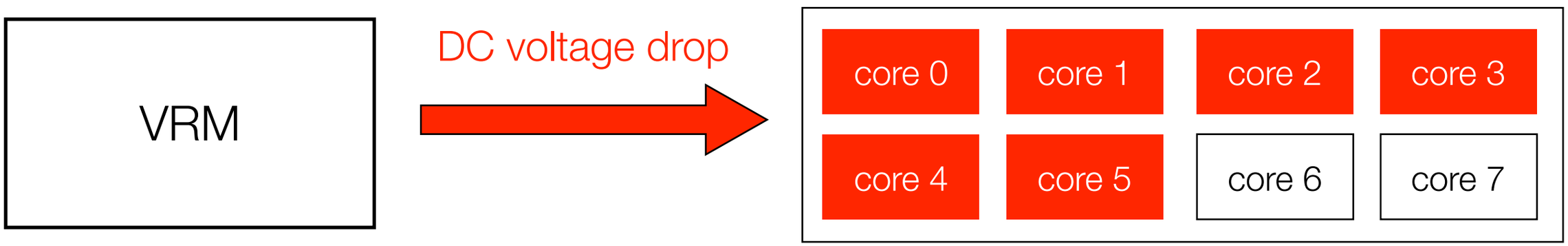
“Bank Run” on Active Timing Margin

- High-power workload induces DC voltage drop



“Bank Run” on Active Timing Margin

- High-power workload induces DC voltage drop
- Limit ATM’s timing margin reclamation potential



Adaptive Guardband Scheduling to Improve System-Level Efficiency of the POWER7+

Yazhou Zu[1], Charles R. Lefurgy[2], Jingwen Leng[1], Matthew Halpern[1],
Michael S. Floyd[2], Vijay Janapa Reddi[1]

[1] The University of Texas at Austin
{yazhou.zu, jingwen, matthalp}@utexas.edu, vj@ece.utexas.edu

[2] IBM
{lefurgy, mfloyd}@us.ibm.com

ABSTRACT

The traditional guardbanding approach to ensure processor reliability is becoming obsolete because it always over-provisions voltage and wastes a lot of energy. As a next-generation alternative, adaptive guardbanding dynamically adjusts chip clock frequency and voltage based on timing margin measured at runtime. With adaptive guardbanding, voltage guardband is only provided when needed, thereby promising significant energy efficiency improvement.

In this paper, we provide the first full-system analysis of adaptive guardbanding's implications using

Categories and Subject Descriptors

B.8 [**Hardware**]: Performance and Reliability; C.1.0 [**Processor Architectures**]: General

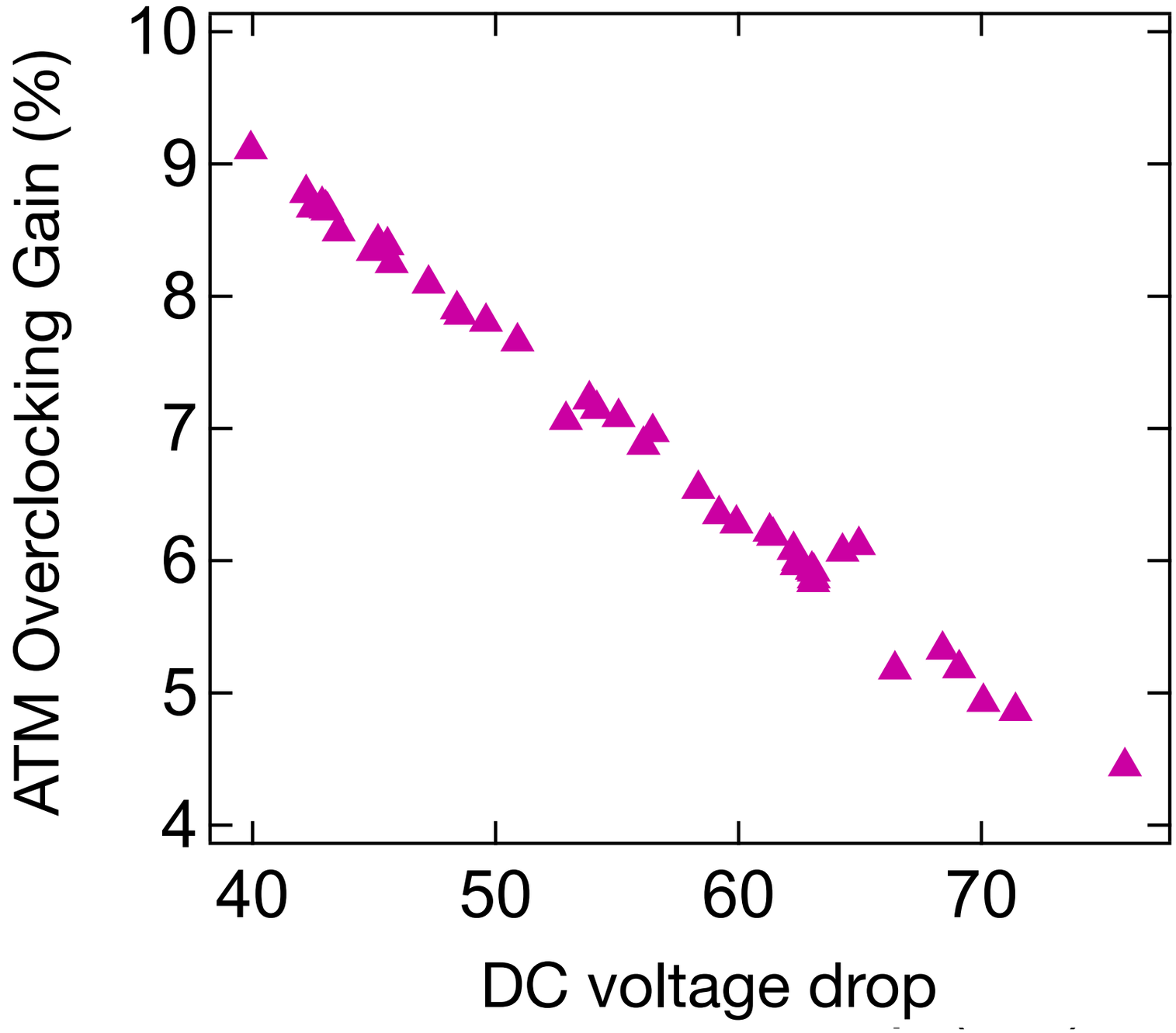
Keywords

operating margin; di/dt effect; voltage drop; energy efficiency; scheduling

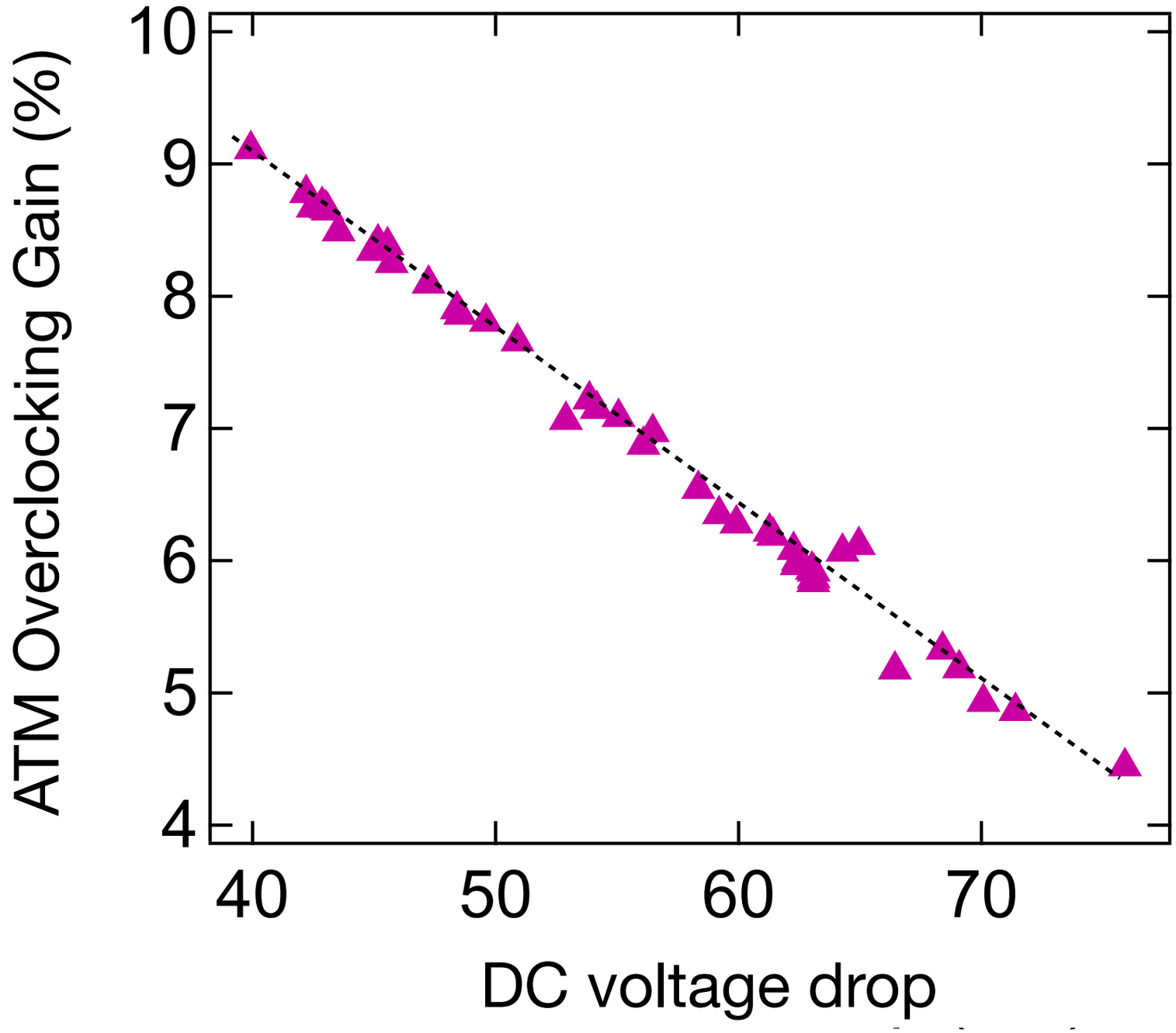
1. INTRODUCTION

Processor manufacturers commonly apply operating guardband to ensure that microprocessors operate reli-

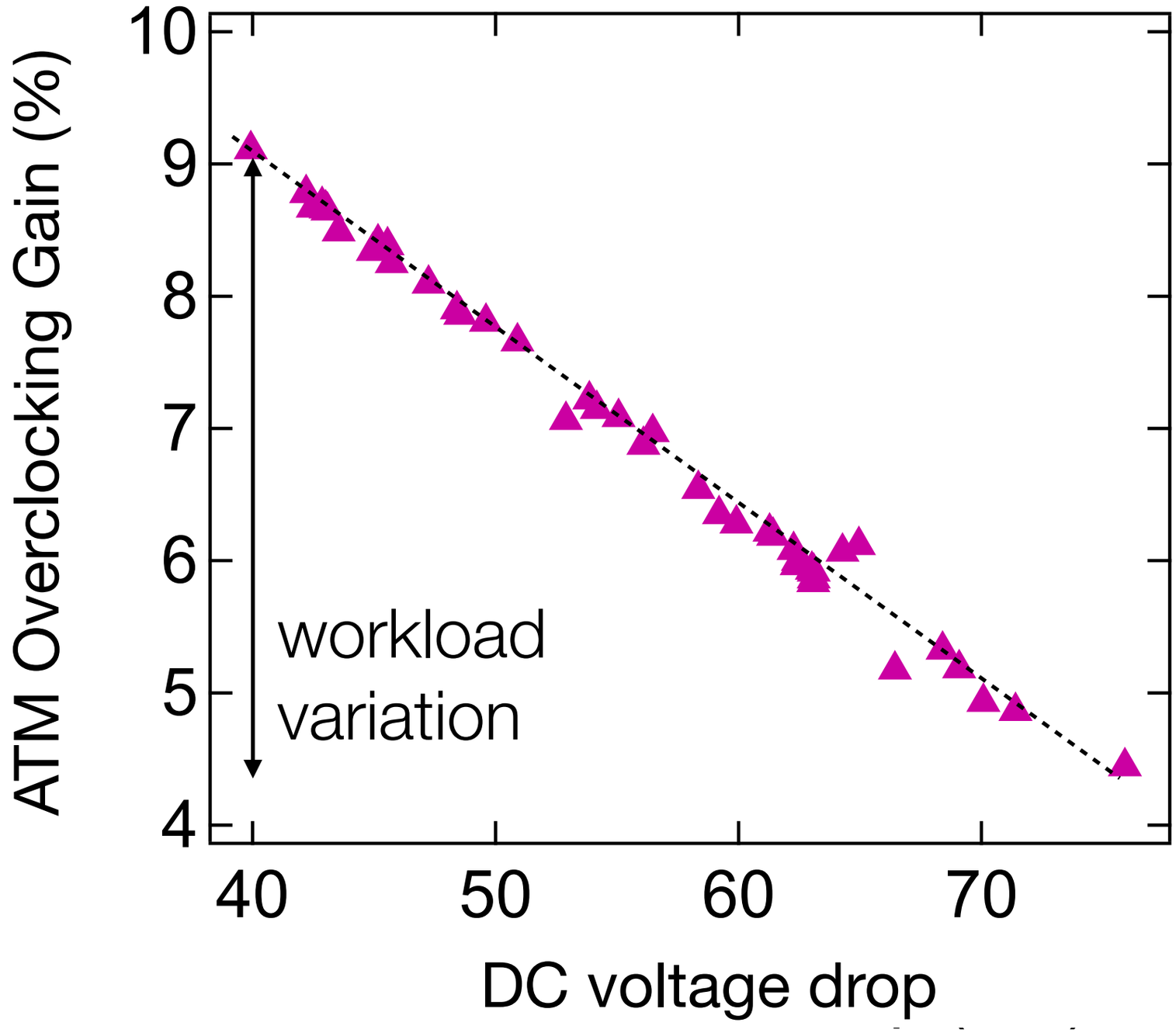
Application Power Consumption Limits ATM's Gain



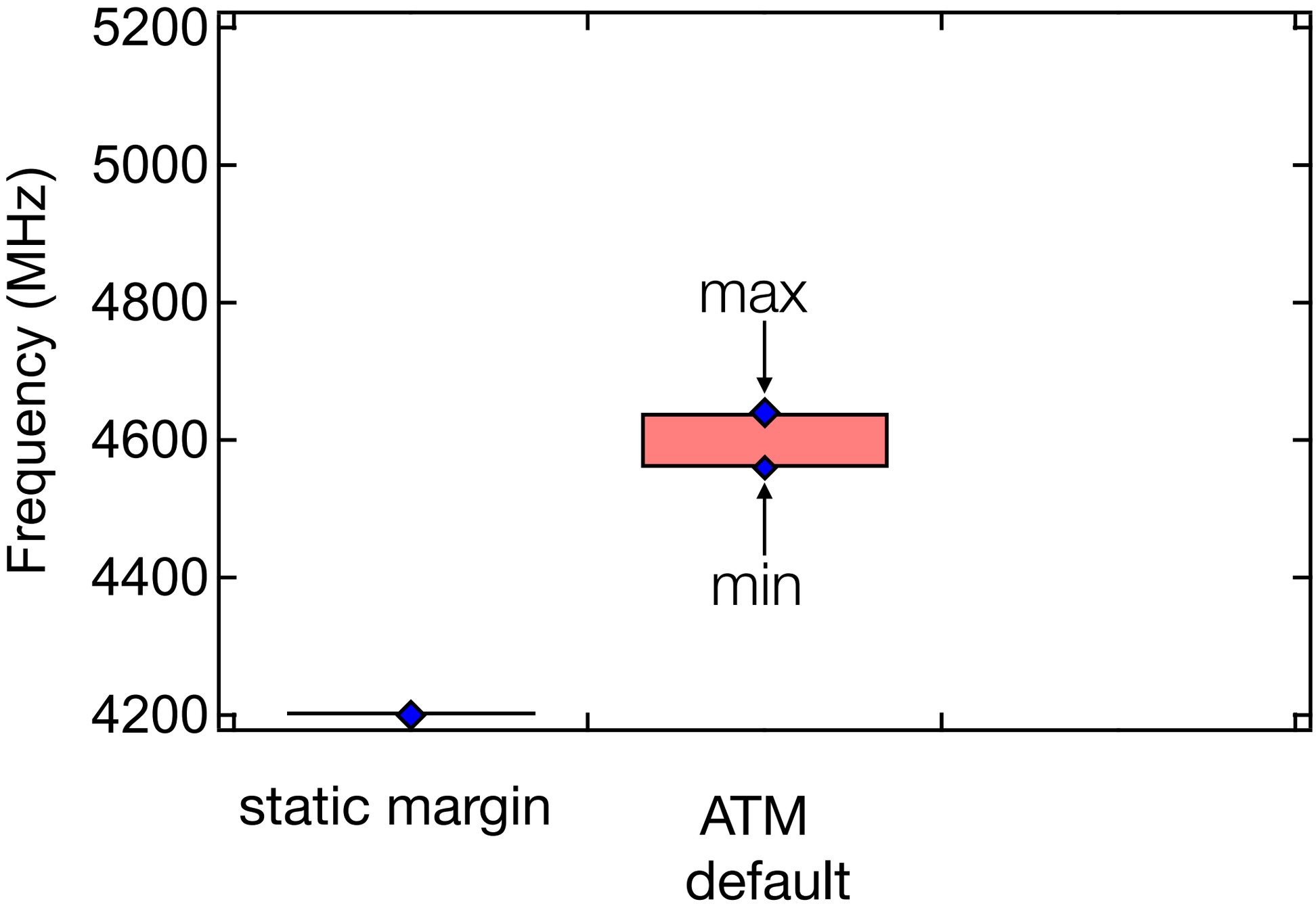
Application Power Consumption Limits ATM's Gain



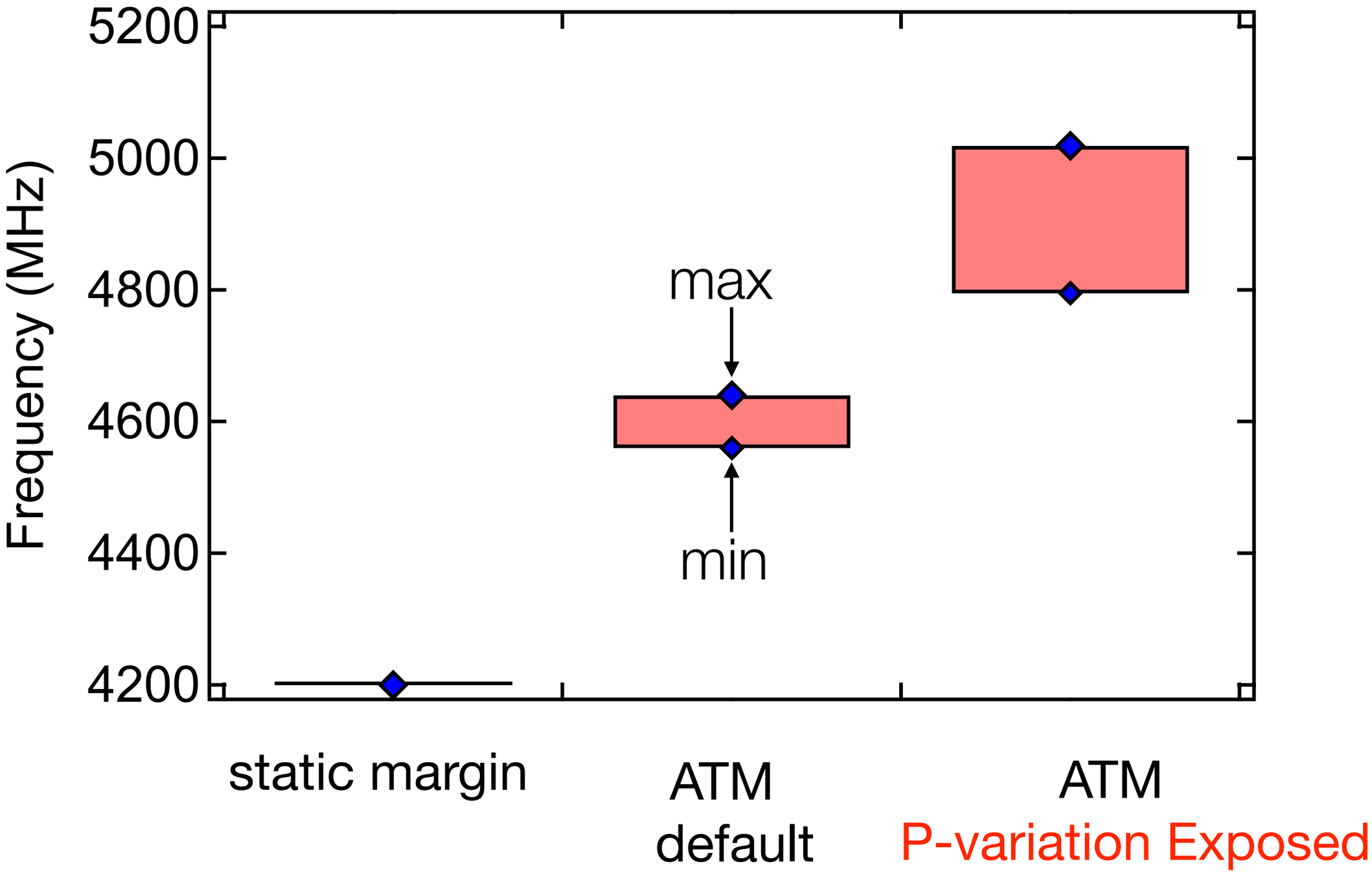
Application Power Consumption Limits ATM's Gain



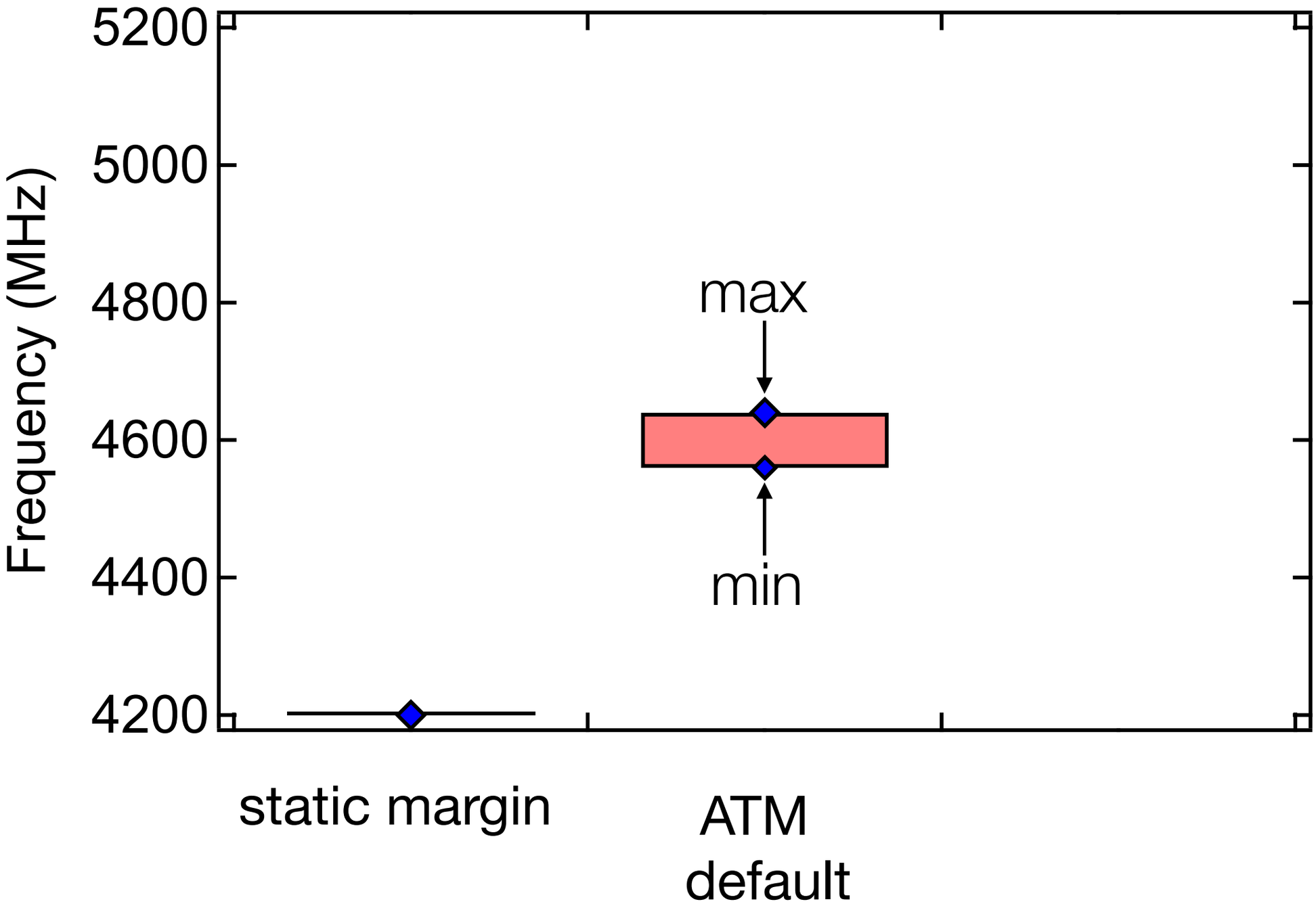
Executive Summary



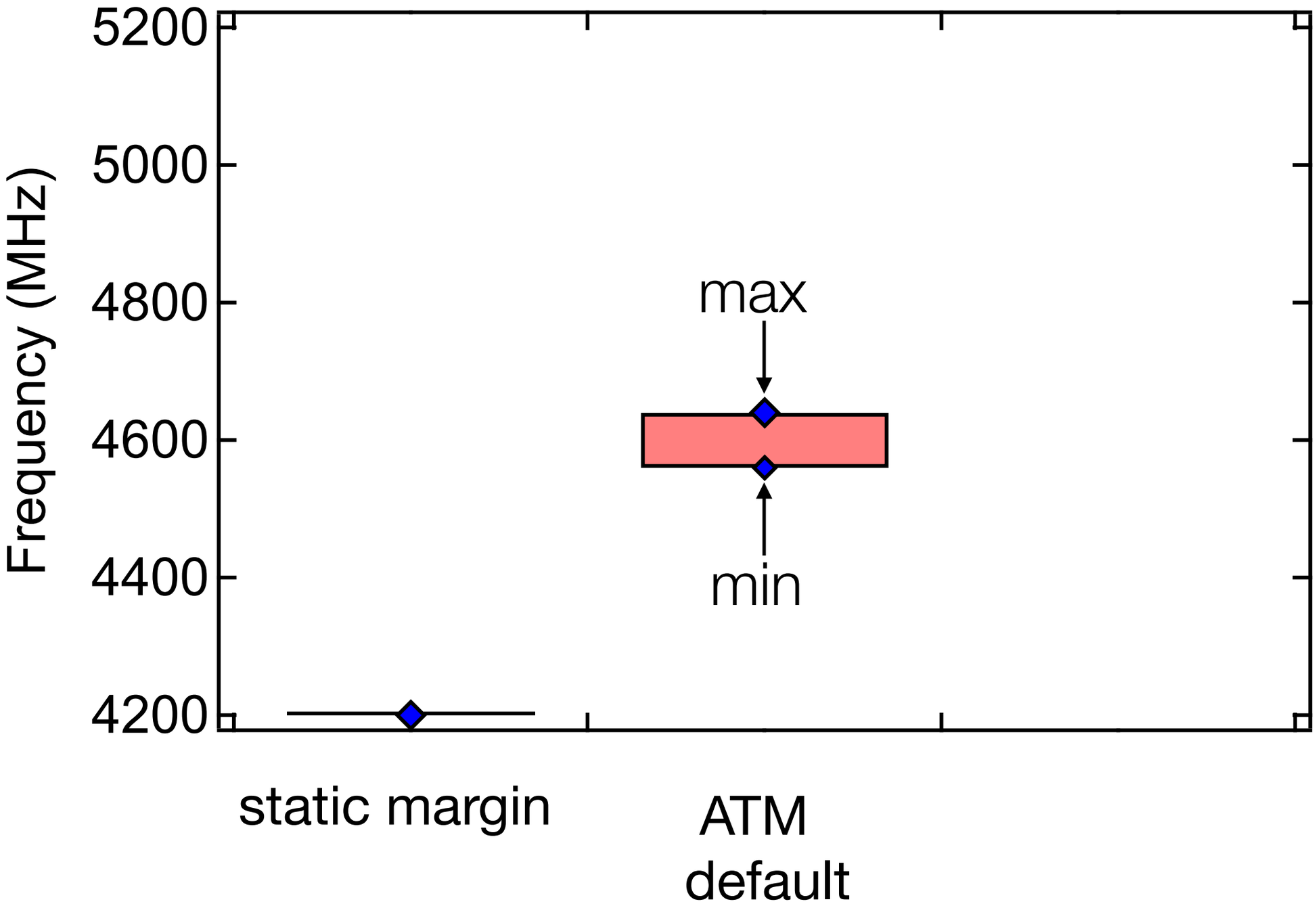
Executive Summary



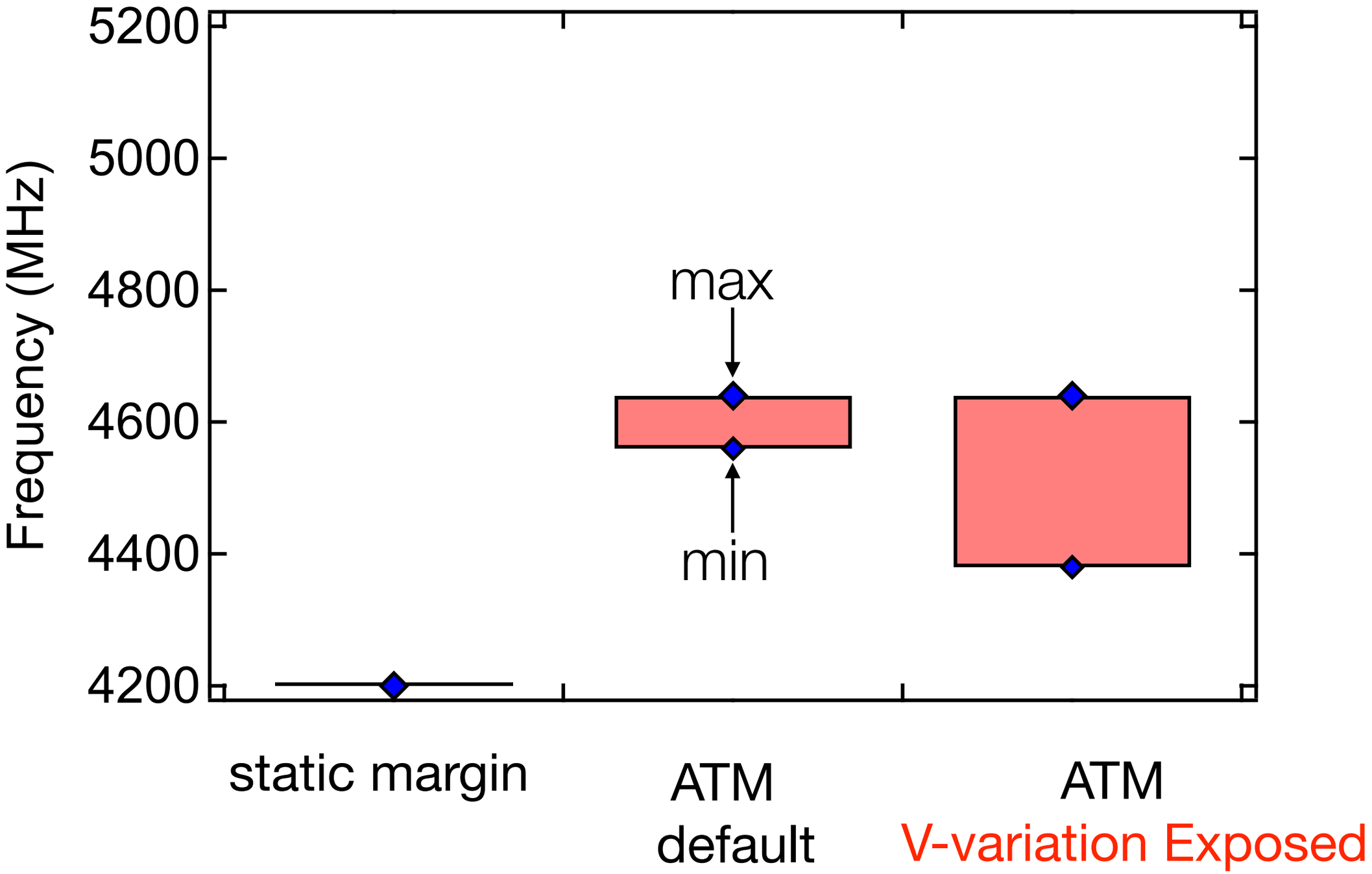
Executive Summary



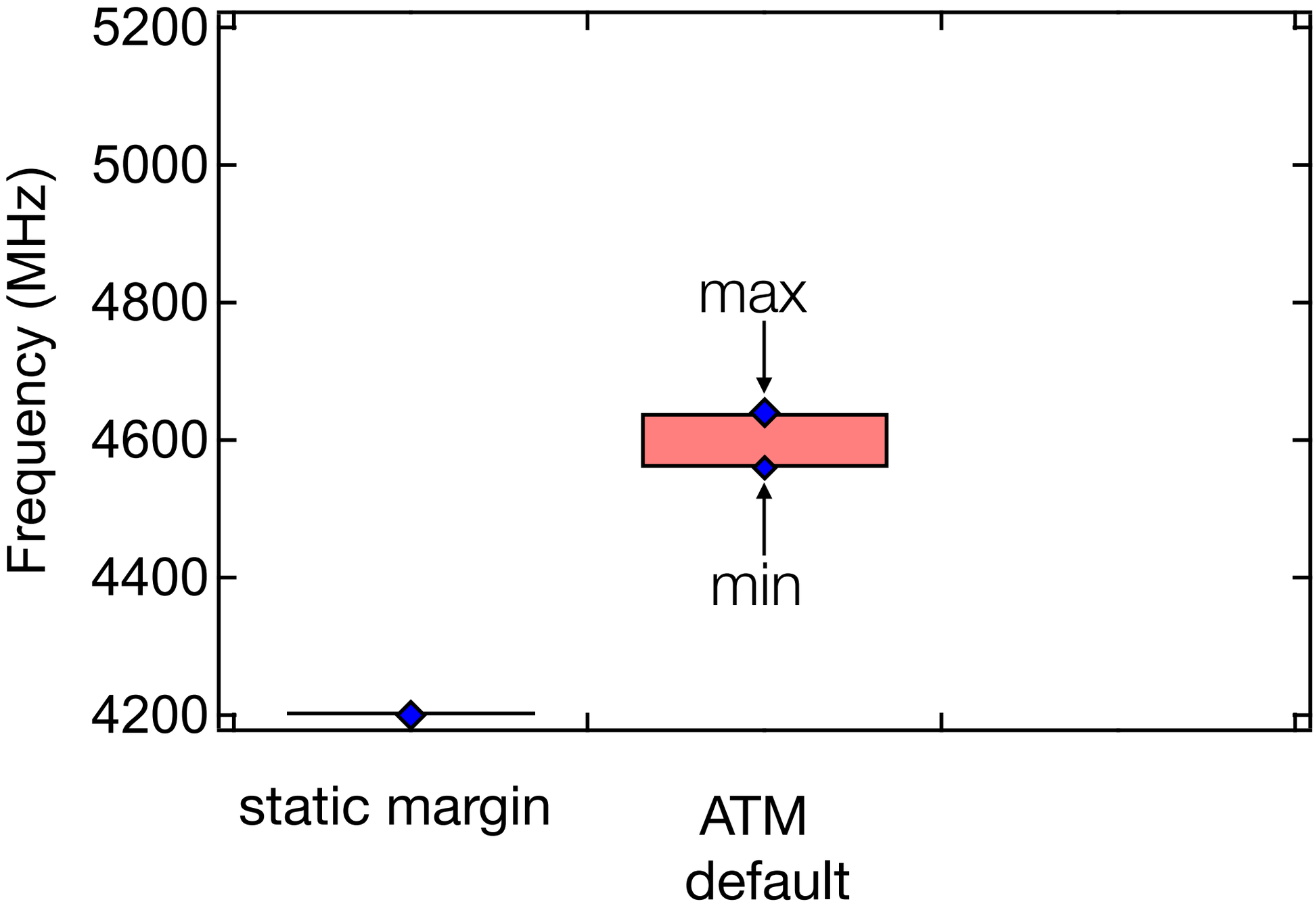
Executive Summary



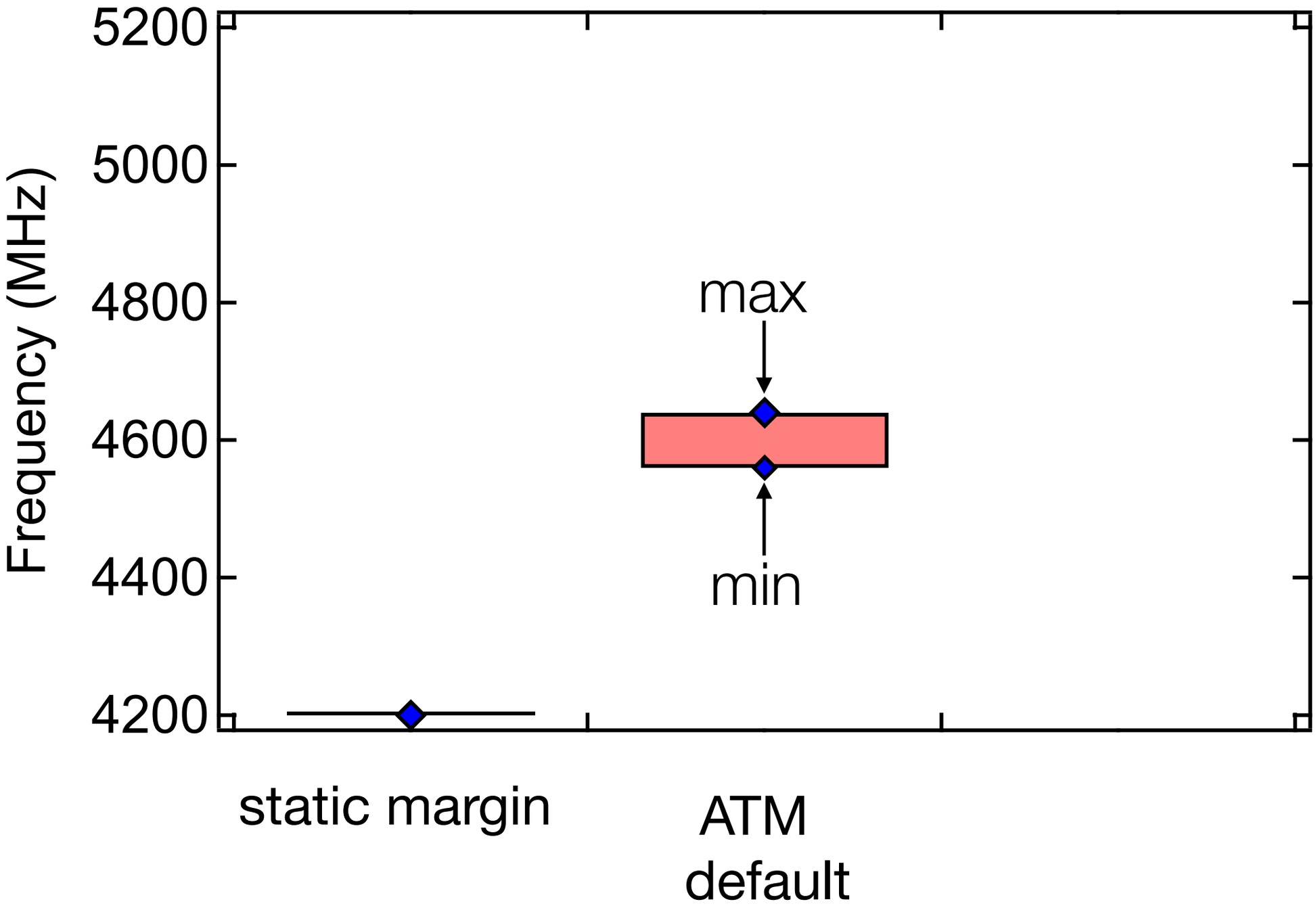
Executive Summary



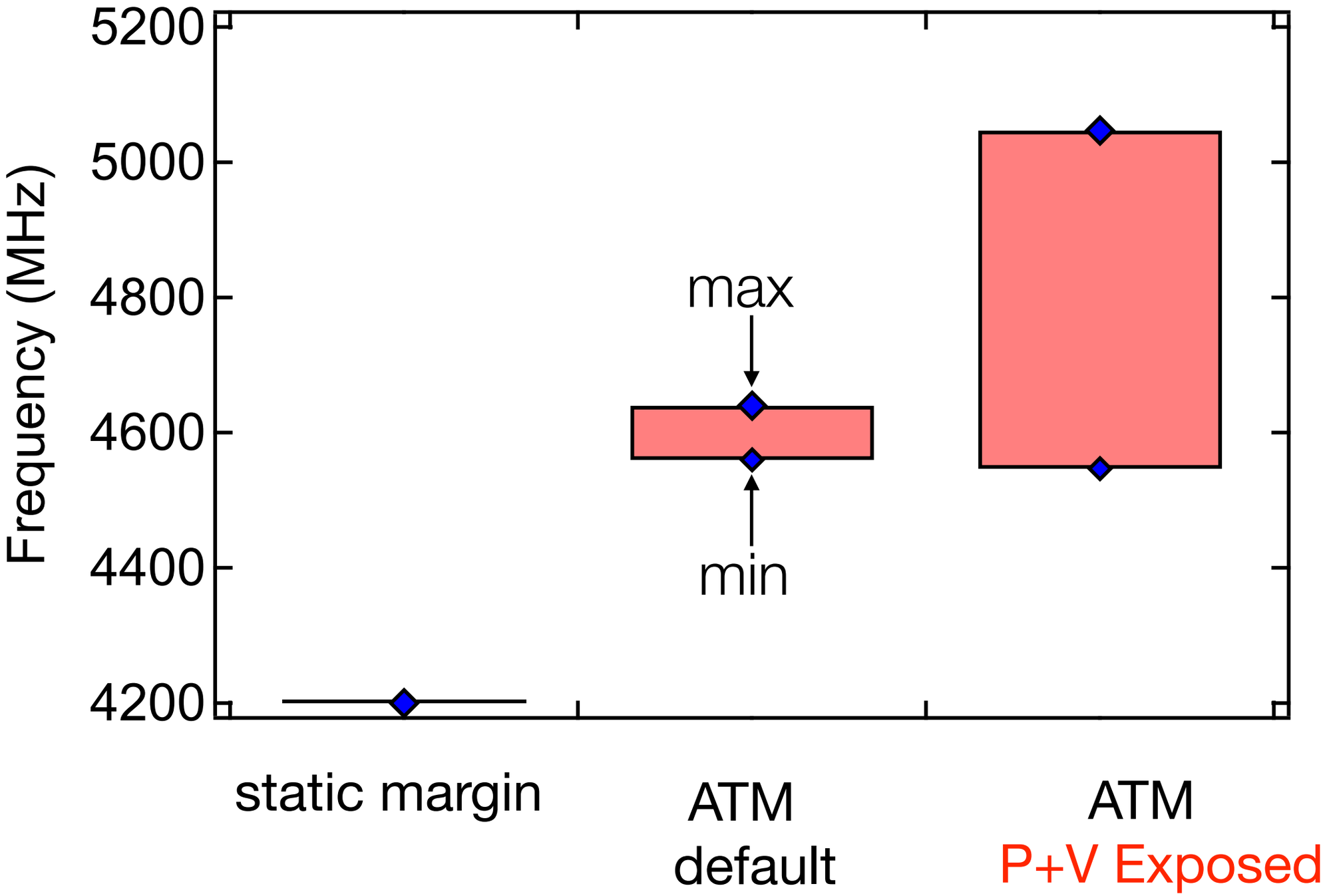
Executive Summary



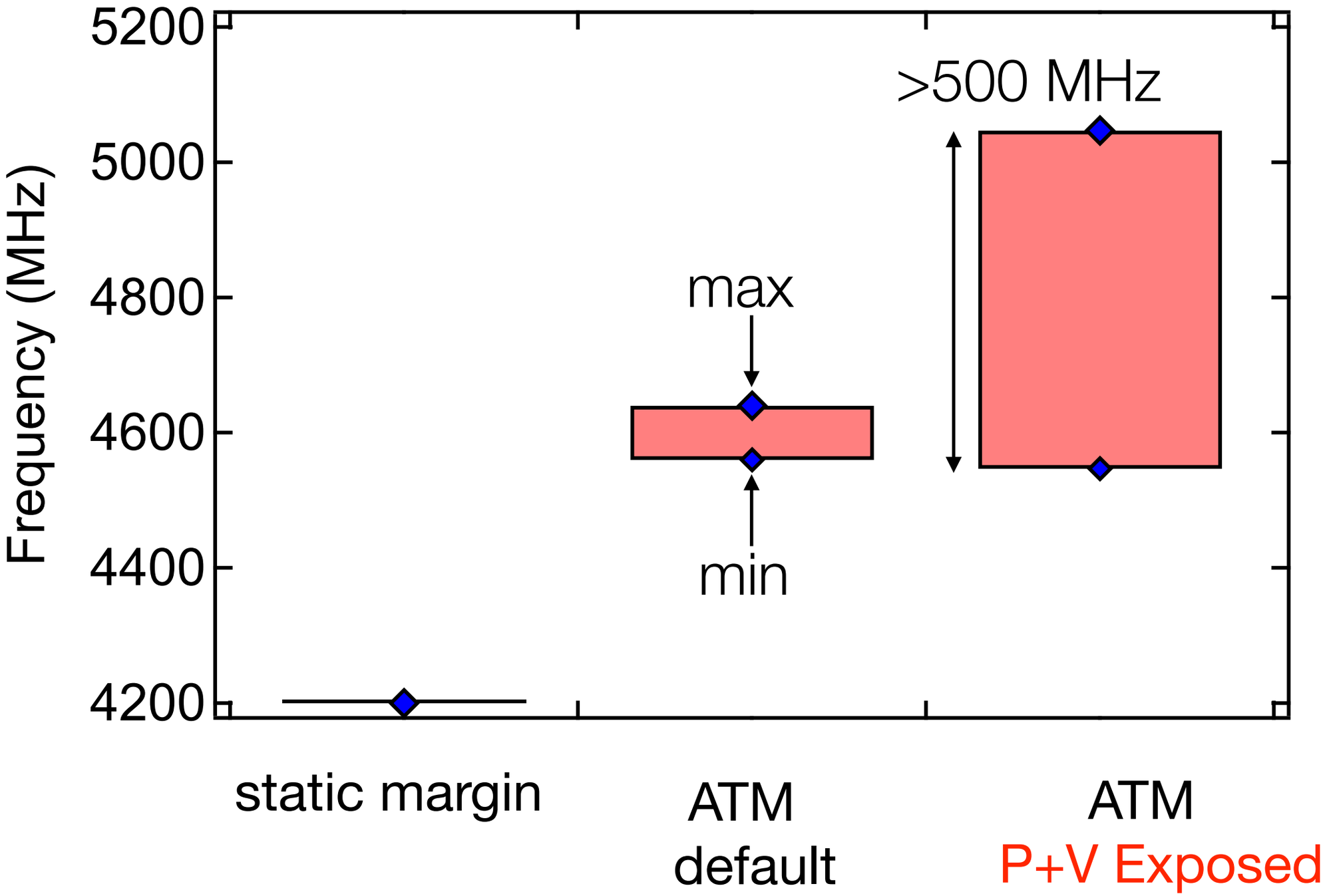
Executive Summary



Executive Summary



Executive Summary



Executive Summary

- Fine-tune core-level ATM to **automatically** expose inter-core performance variation

- Expose chip-wide performance variation caused by heterogenous workload intensity

- Propose application management to demonstrate >10% predictable performance gain

Managing Cloud Workload Performance

- Latency, not frequency, matters for cloud workloads

Managing Cloud Workload Performance

- Latency, not frequency, matters for cloud workloads

Foreground (fg)	Background (bg)
resnet, vgg, ferret, fluidanimate	mlp-train, gcc, facesim, lu_cb, streamcluster
squeezenet, seq2seq, babi-rnn, bodytrack, vips	blackscholes, x264 swaptions, raytrace

Managing Cloud Workload Performance

- Latency, not frequency, matters for cloud workloads

latency-sensitive, user-facing services

Foreground (fg)	Background (bg)
resnet, vgg, ferret, fluidanimate	mlp-train, gcc, facesim, lu_cb, streamcluster
squeezenet, seq2seq, babi-rnn, bodytrack, vips	blackscholes, x264 swaptions, raytrace

Managing Cloud Workload Performance

- Latency, not frequency, matters for cloud workloads

Foreground (fg)	Background (bg)
latency-sensitive, user-facing services	offline batch-processing data analytics jobs
resnet, vgg, ferret, fluidanimate	mlp-train, gcc, facesim, lu_cb, streamcluster
squeezenet, seq2seq, babi-rnn, bodytrack, vips	blackscholes, x264 swaptions, raytrace

Managing Cloud Workload Performance

- Latency, not frequency, matters for cloud workloads
- Improve foreground critical workload performance controllably

Foreground (fg)	Background (bg)
latency-sensitive, user-facing services	offline batch-processing data analytics jobs
resnet, vgg, ferret, fluidanimate	mlp-train, gcc, facesim, lu_cb, streamcluster
squeezenet, seq2seq, babi-rnn, bodytrack, vips	blackscholes, x264 swaptions, raytrace

Managing Cloud Workload Performance

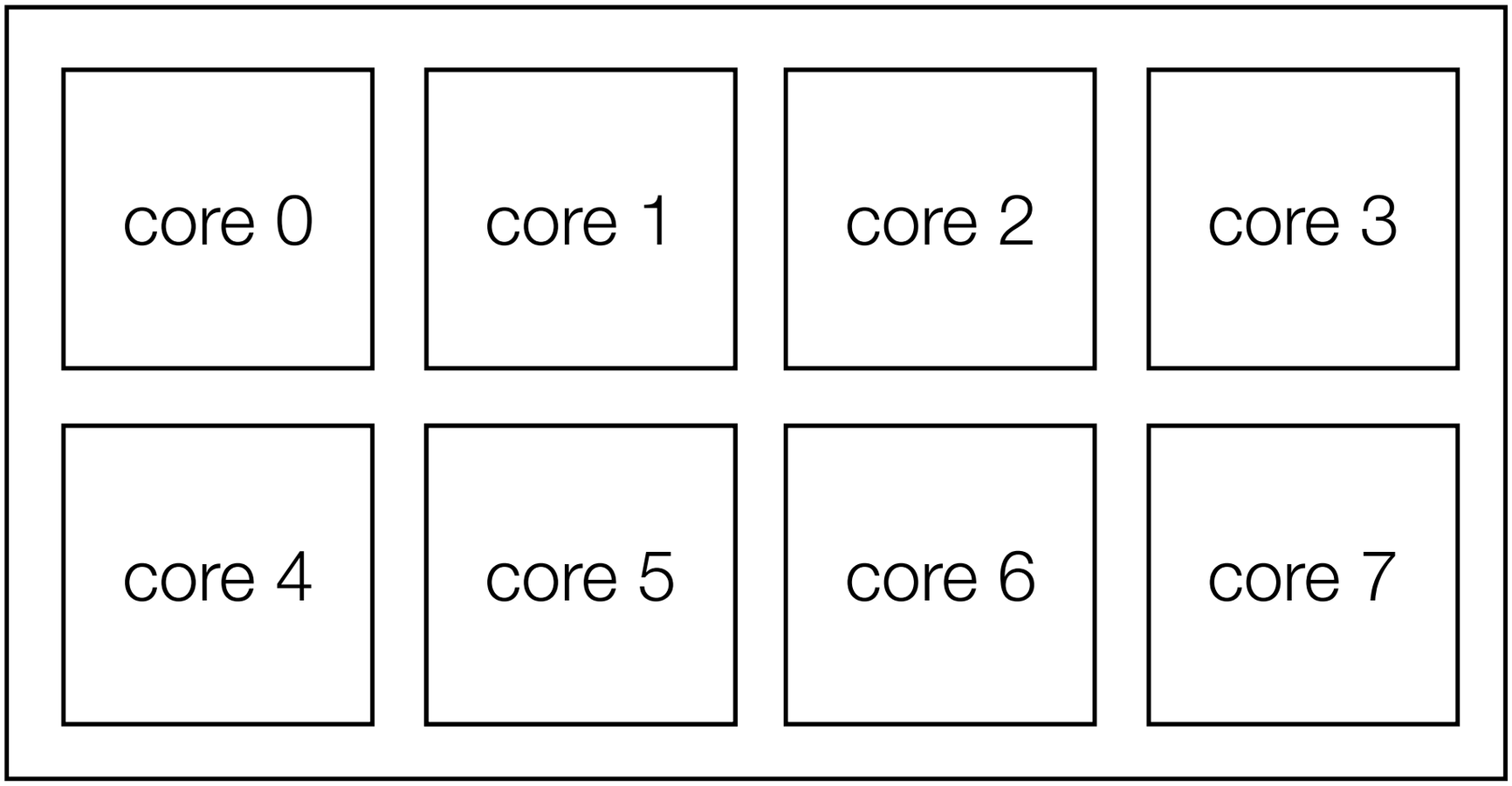
- Latency, not frequency, matters for cloud workloads
- Improve foreground critical workload performance controllably

latency-sensitive, user-facing services

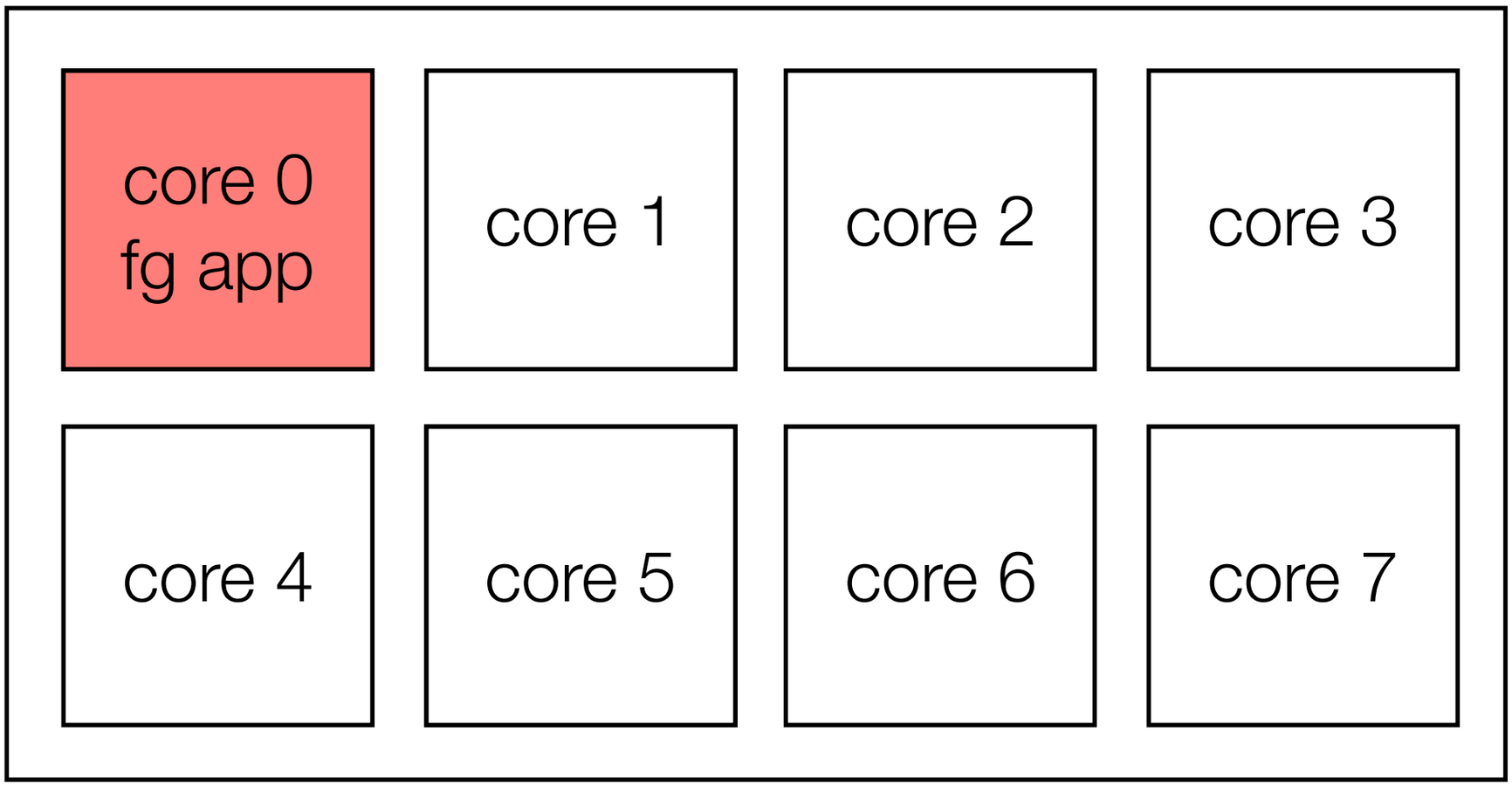
offline batch-processing data analytics jobs

	Foreground (fg)	Background (bg)
memory intensive	resnet, vgg, ferret, fluidanimate	mlp-train, gcc, facesim, lu_cb, streamcluster
memory non-intensive	squeezenet, seq2seq, babi-rnn, bodytrack, vips	blackscholes, x264 swaptions, raytrace

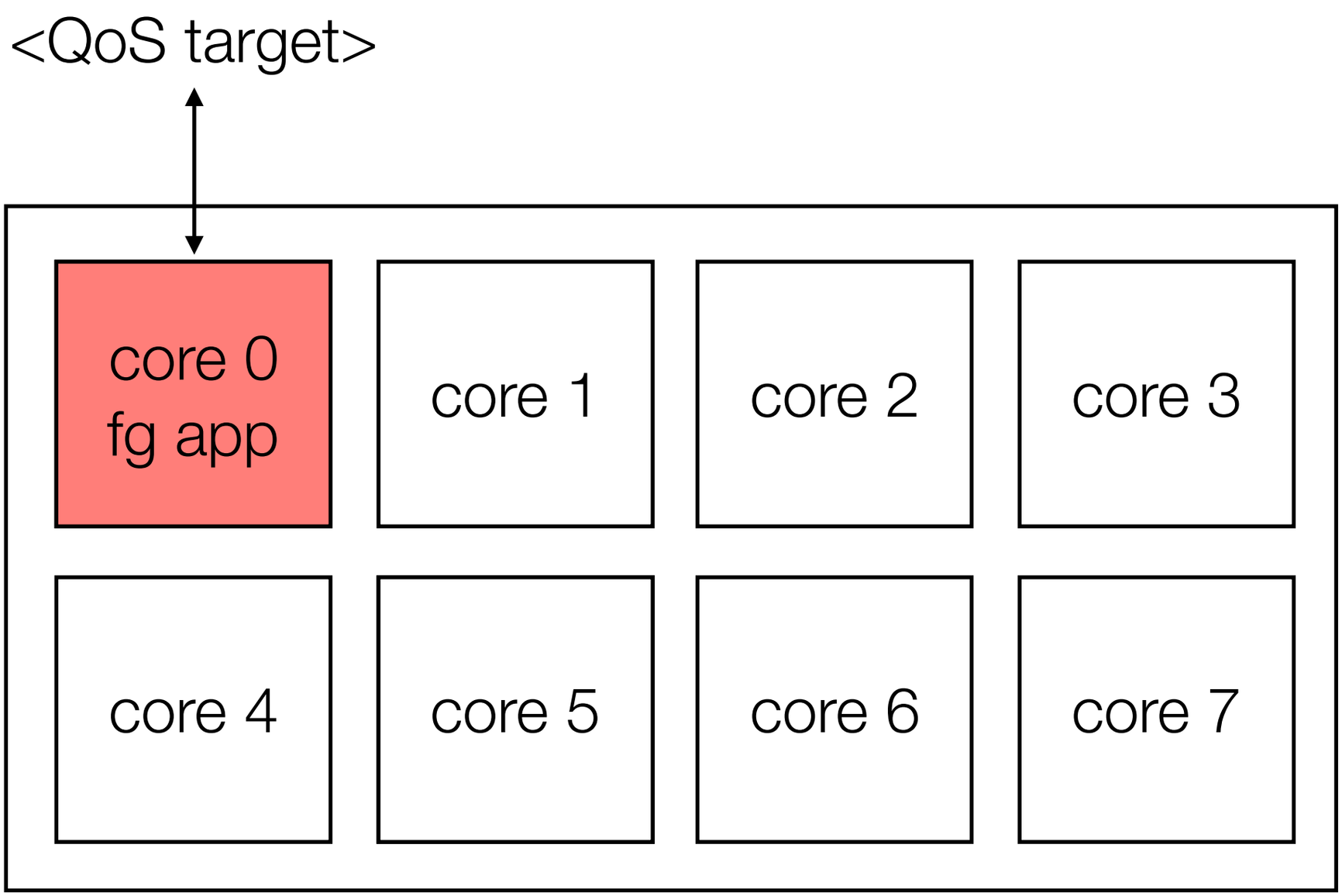
Managing Critical Application Performance



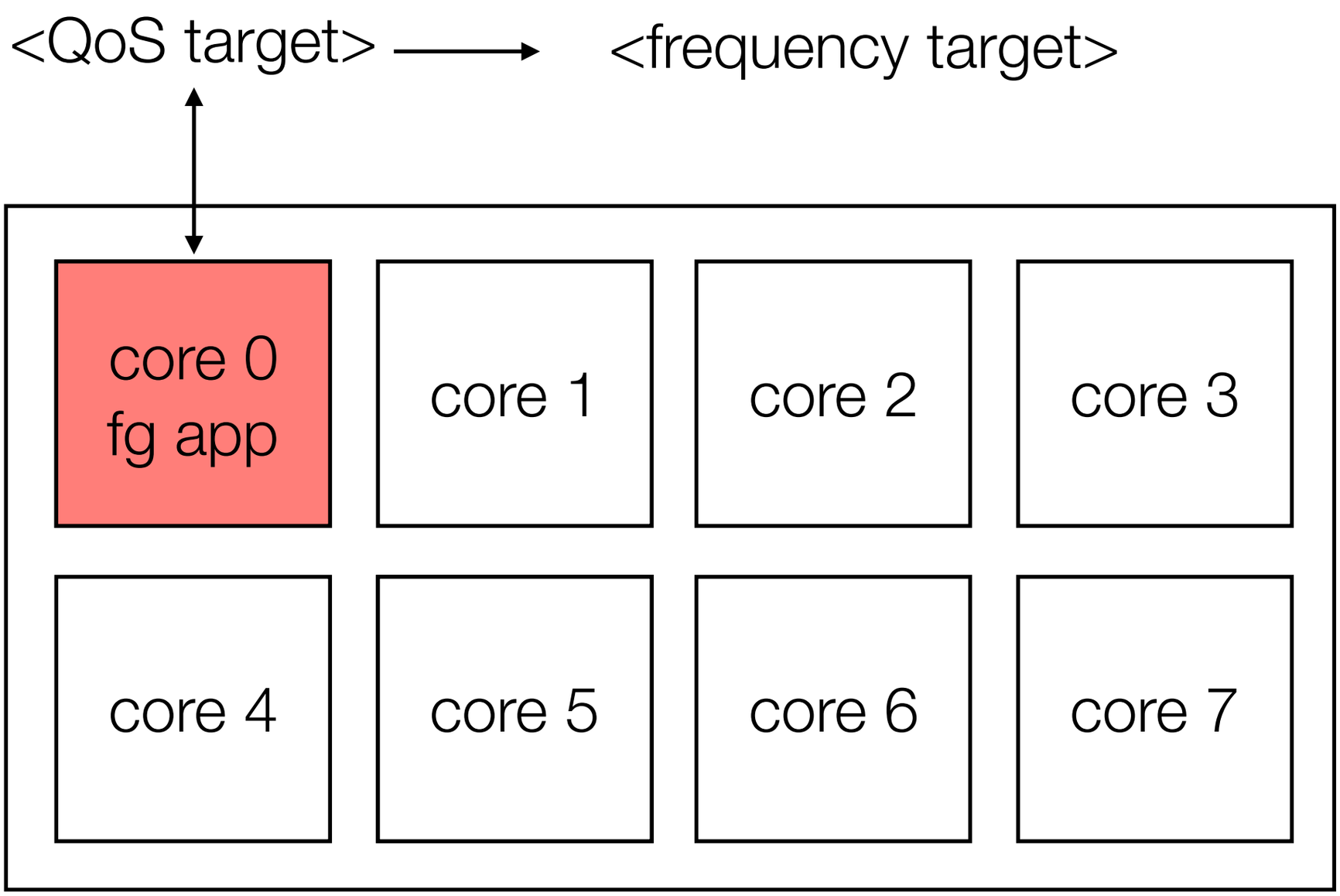
Managing Critical Application Performance



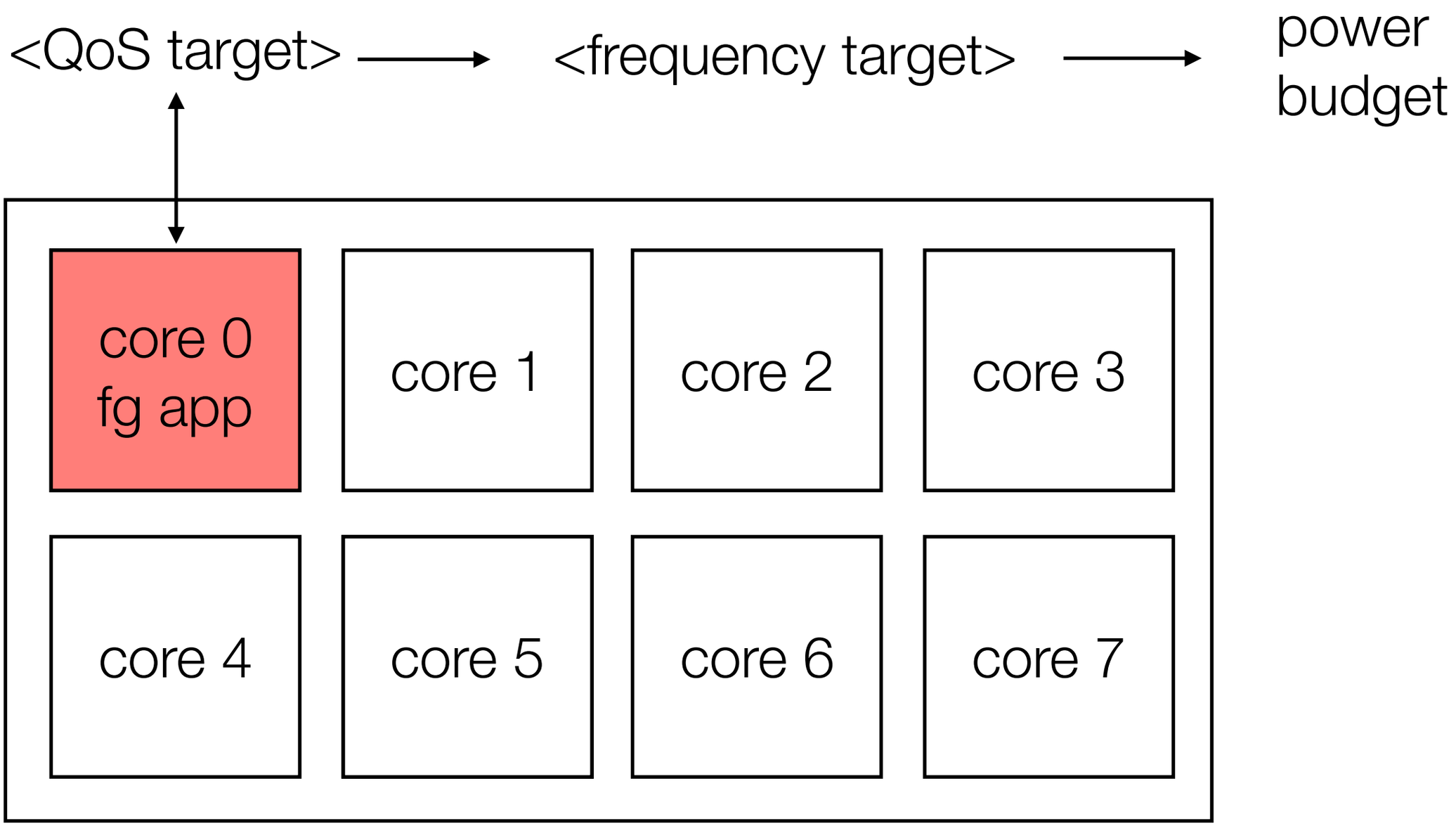
Managing Critical Application Performance



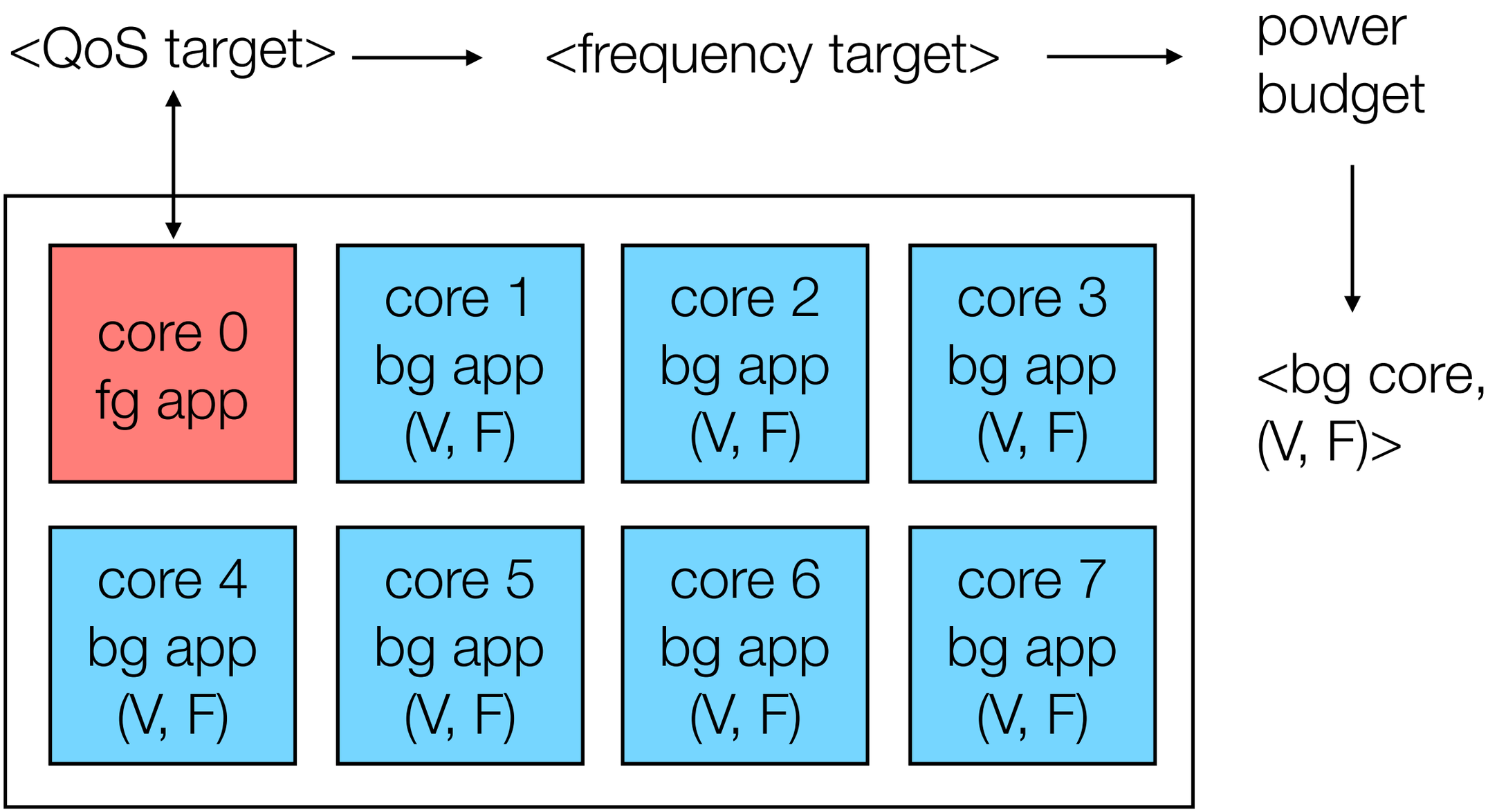
Managing Critical Application Performance



Managing Critical Application Performance

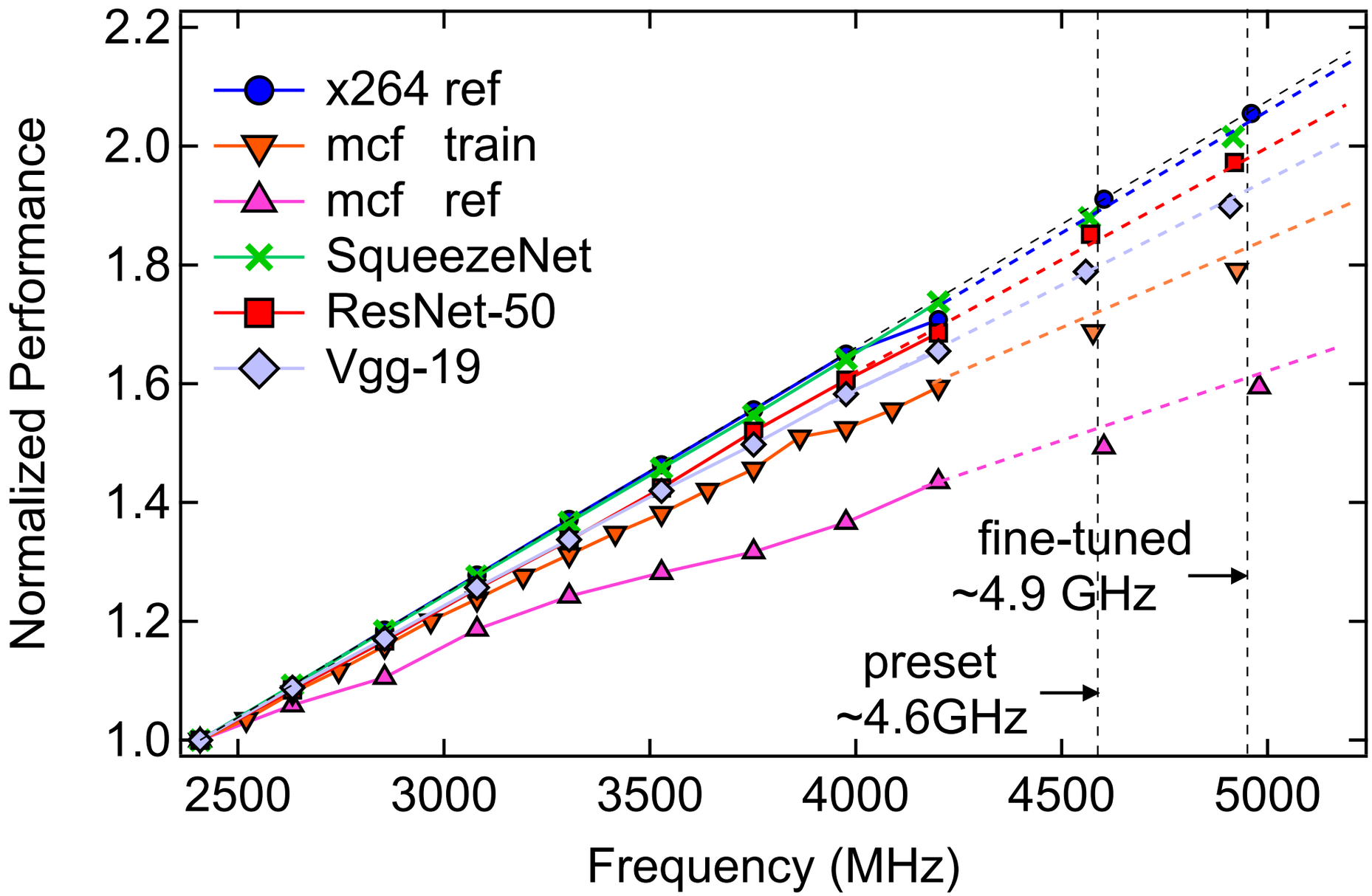


Managing Critical Application Performance



Frequency-based Performance Predictor

- Linear predictor profiled by application and input size



Power-based ATM Frequency Predictor

$$\overline{f} = k \cdot \overline{V_{chip}}$$

Power-based ATM Frequency Predictor

$$\overline{f} = k \cdot \overline{V_{chip}} \Big| = k \cdot (V_{vrm} - R \cdot \overline{I})$$

Power-based ATM Frequency Predictor

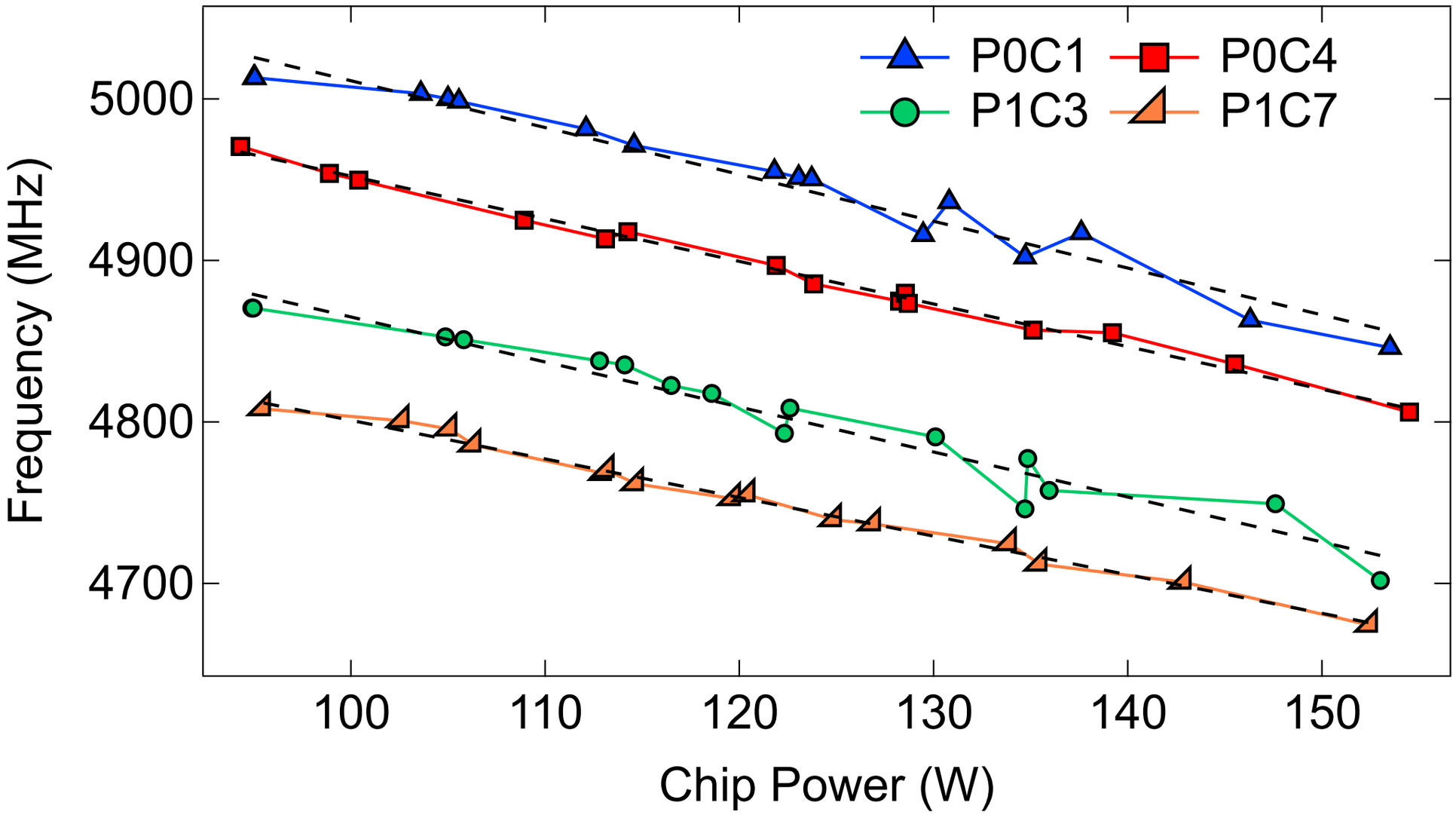
$$\overline{f} = k \cdot \overline{V_{chip}} \Big| = k \cdot (V_{vrm} - R \cdot \overline{I}) \; = k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}})$$

Power-based ATM Frequency Predictor

$$\overline{f} = k \cdot \overline{V_{chip}} \Big| = k \cdot (V_{vrm} - R \cdot \overline{I}) \; = k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}}) \Big|$$
$$= -k' \cdot \overline{P} + b$$

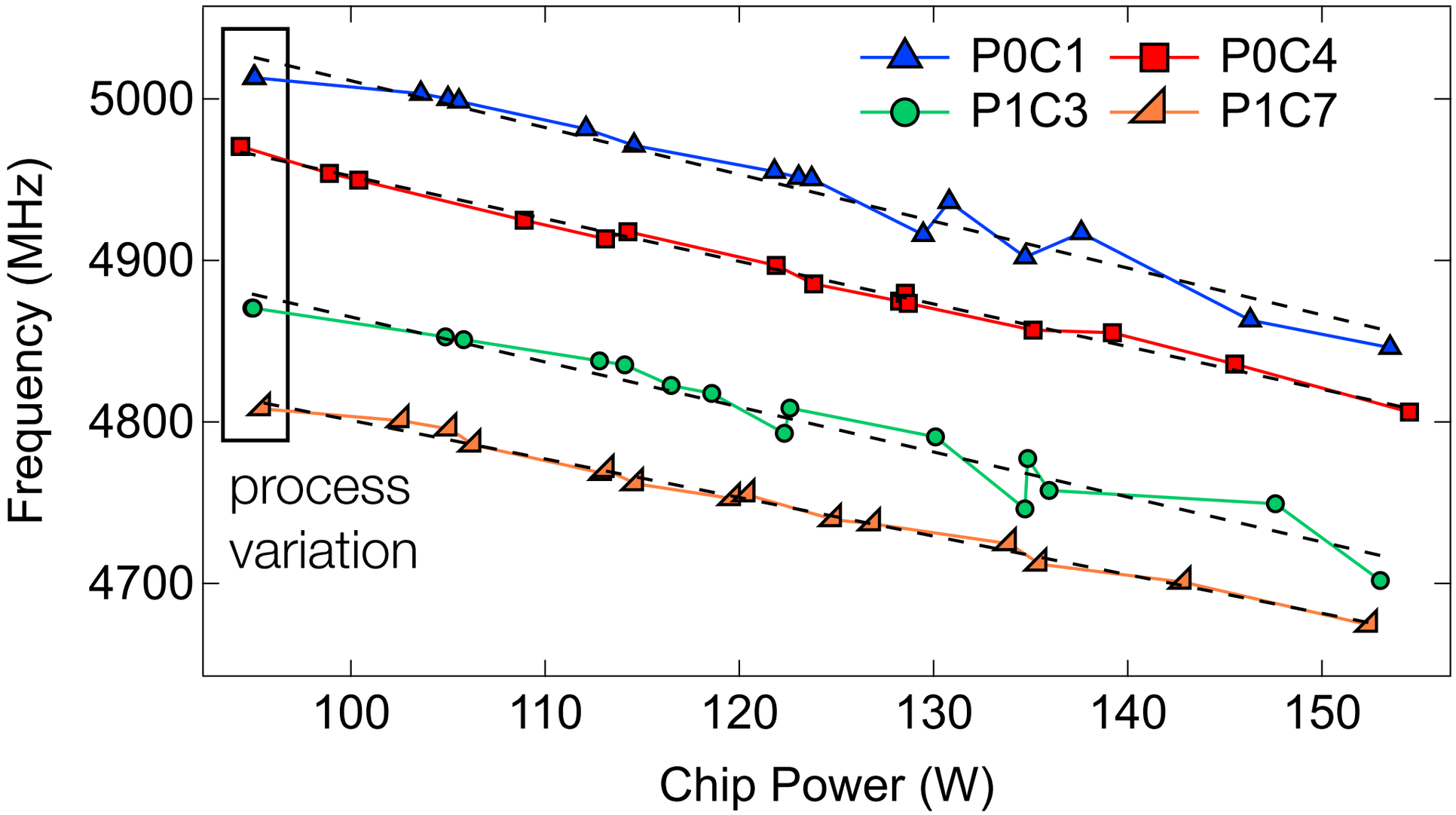
Power-based ATM Frequency Predictor

$$\overline{f} = k \cdot \overline{V_{chip}} = k \cdot (V_{vrm} - R \cdot \overline{I}) = k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}})$$
$$= -k' \cdot \overline{P} + b$$



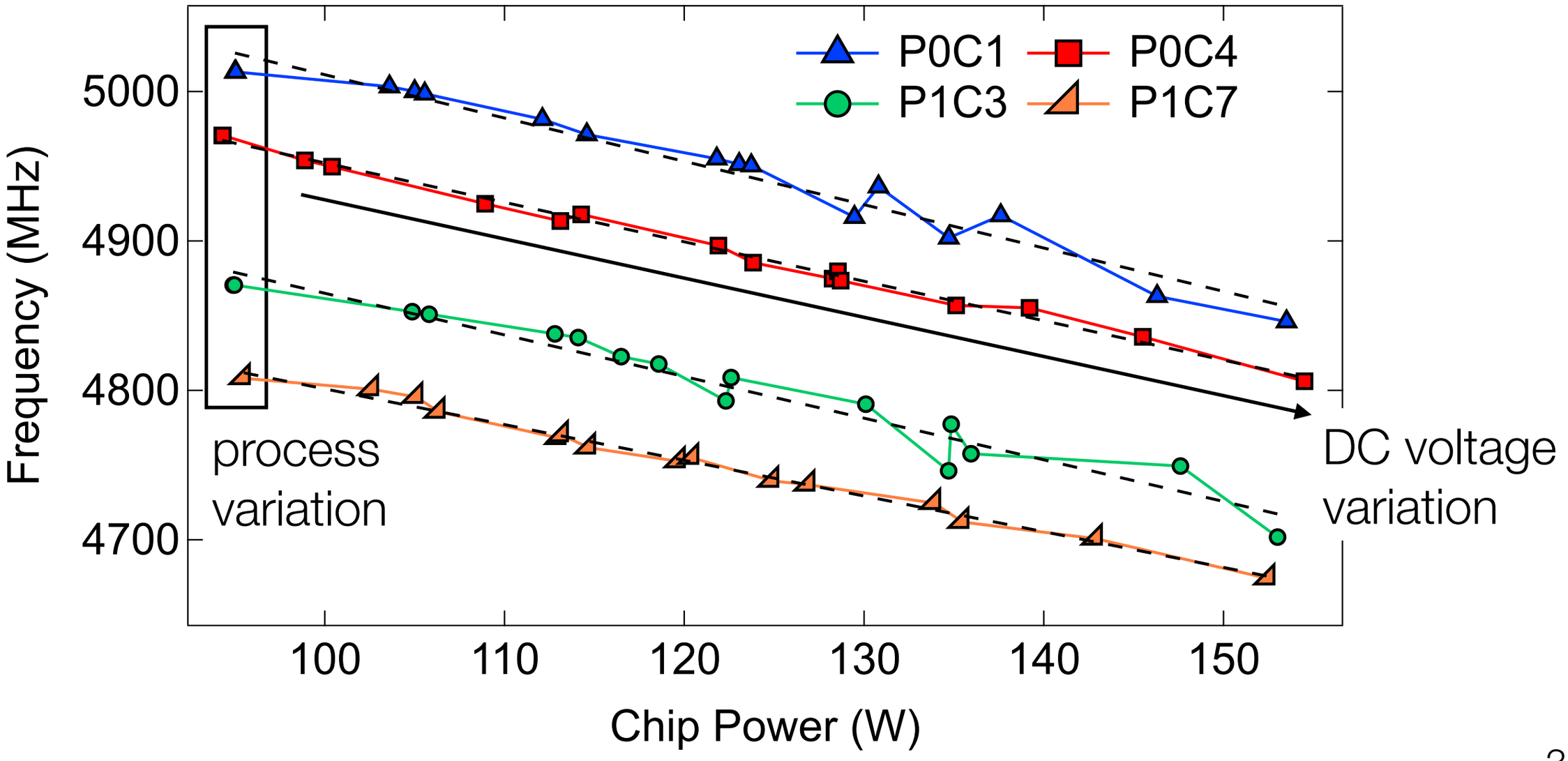
Power-based ATM Frequency Predictor

$$\overline{f} = k \cdot \overline{V_{chip}} = k \cdot (V_{vrm} - R \cdot \overline{I}) = k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}})$$
$$= -k' \cdot \overline{P} + b$$

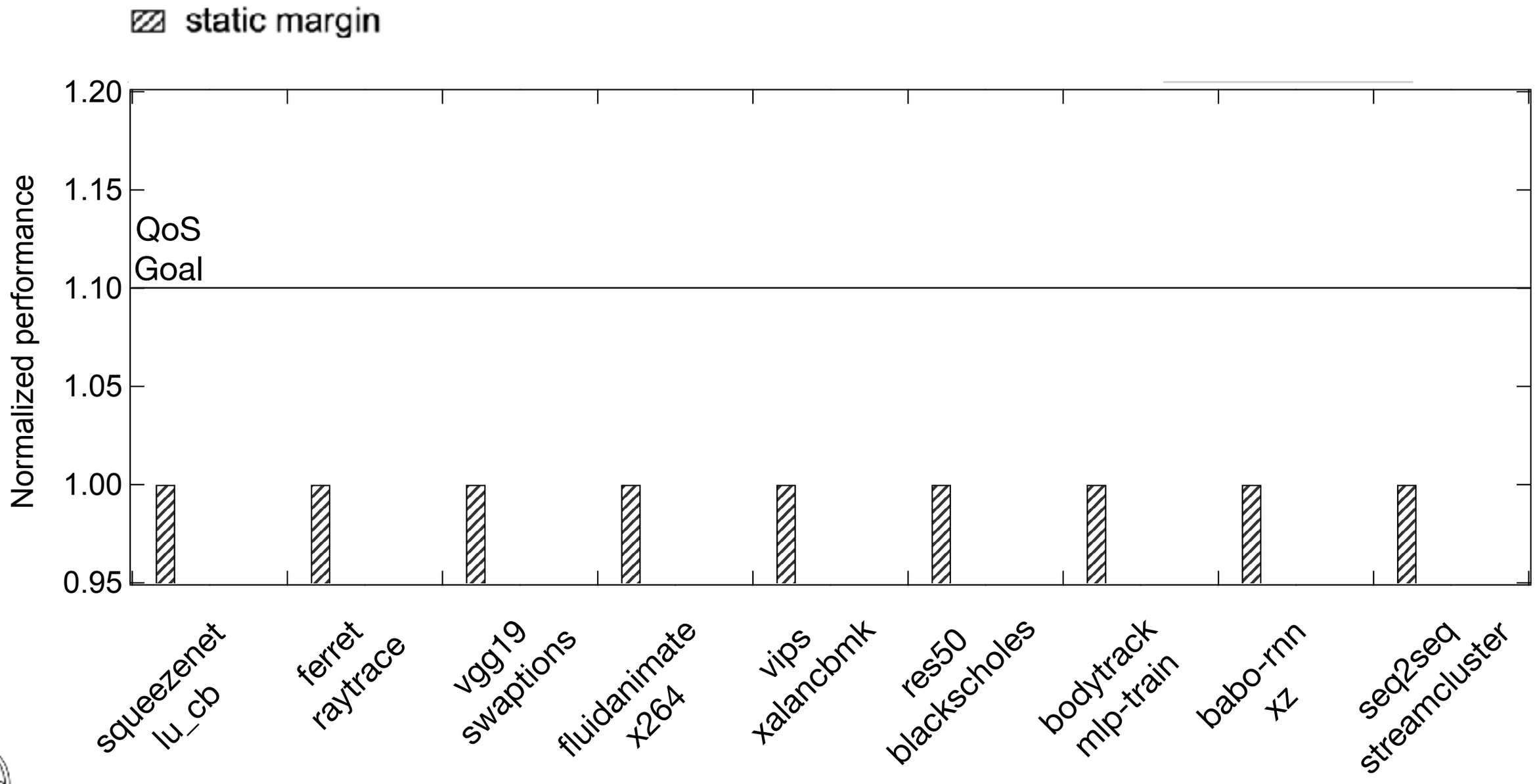


Power-based ATM Frequency Predictor

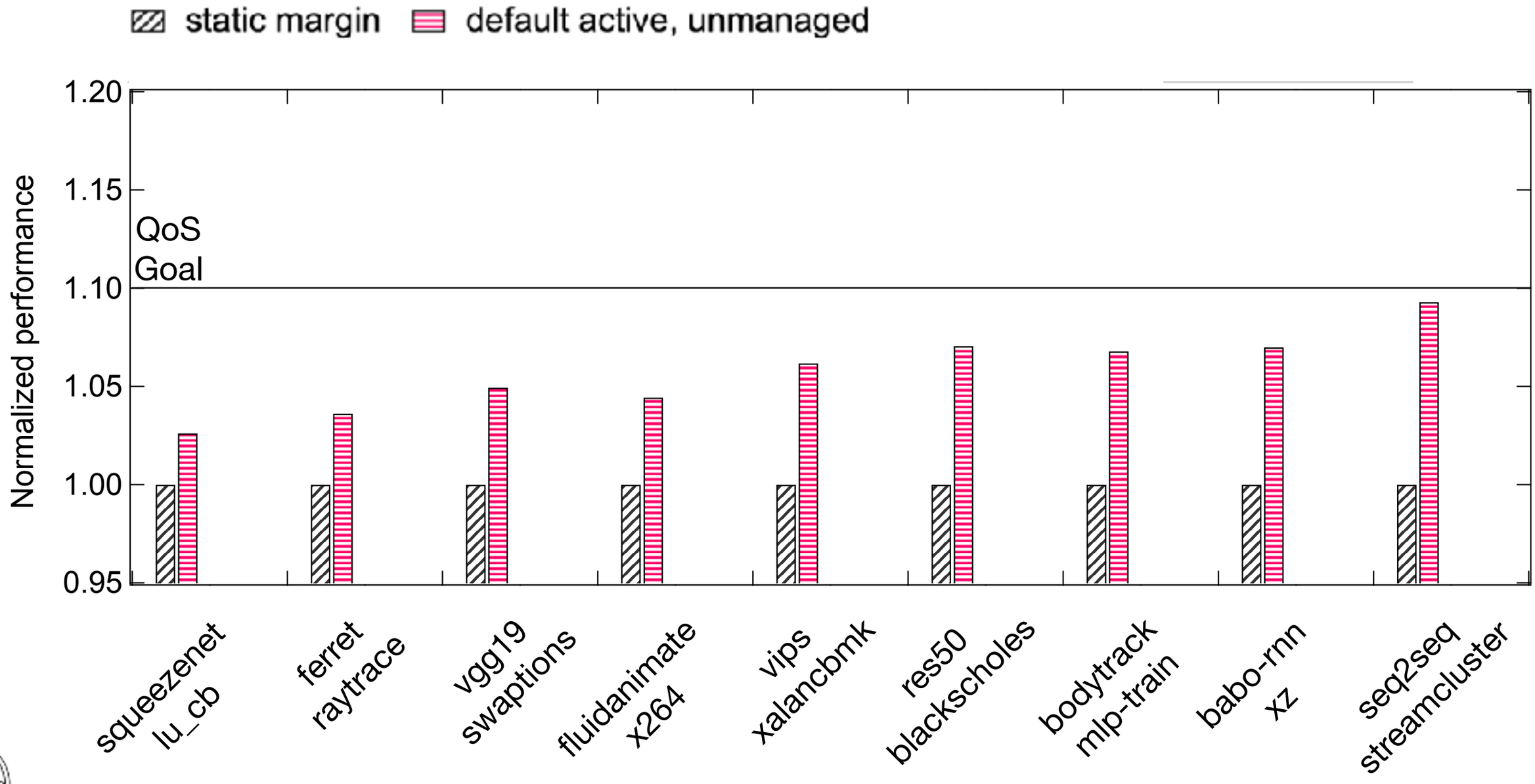
$$\overline{f} = k \cdot \overline{V_{chip}} = k \cdot (V_{vrm} - R \cdot \overline{I}) = k \cdot (V_{vrm} - R \cdot \frac{\overline{P}}{V_{vrm}})$$
$$= -k' \cdot \overline{P} + b$$



Performance Improvement

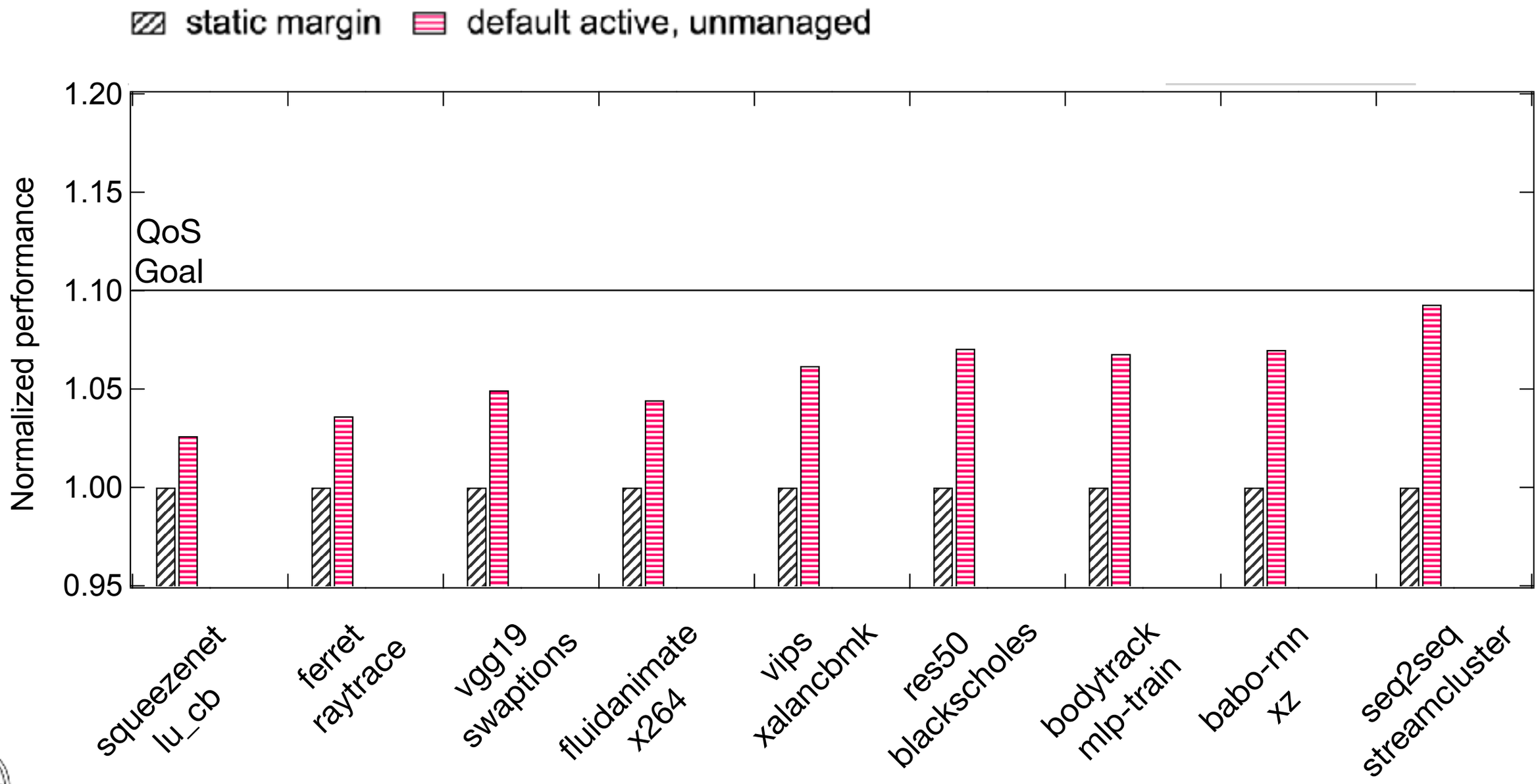


Performance Improvement



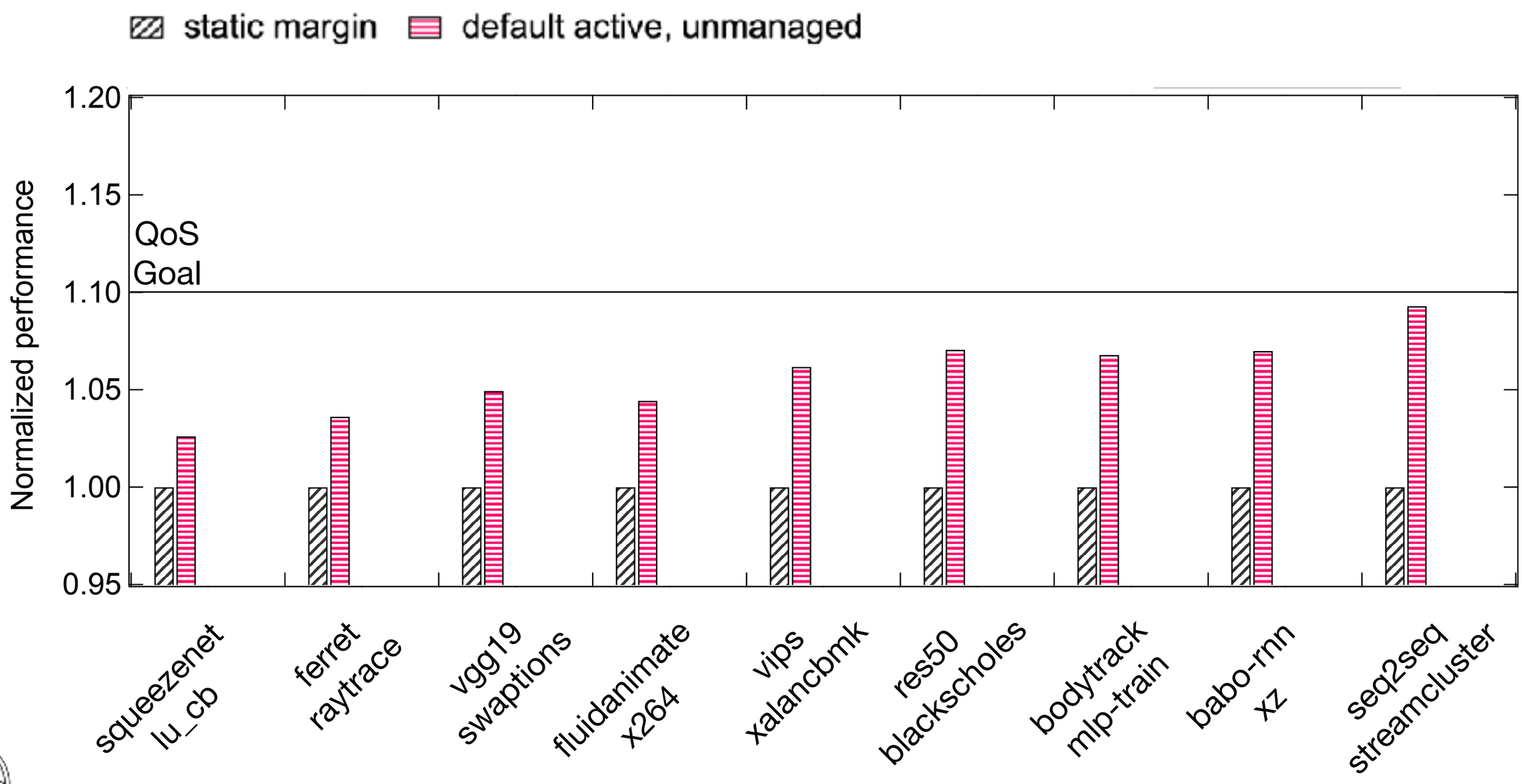
Performance Improvement

- Unmanaged DC voltage drop from co-runner apps

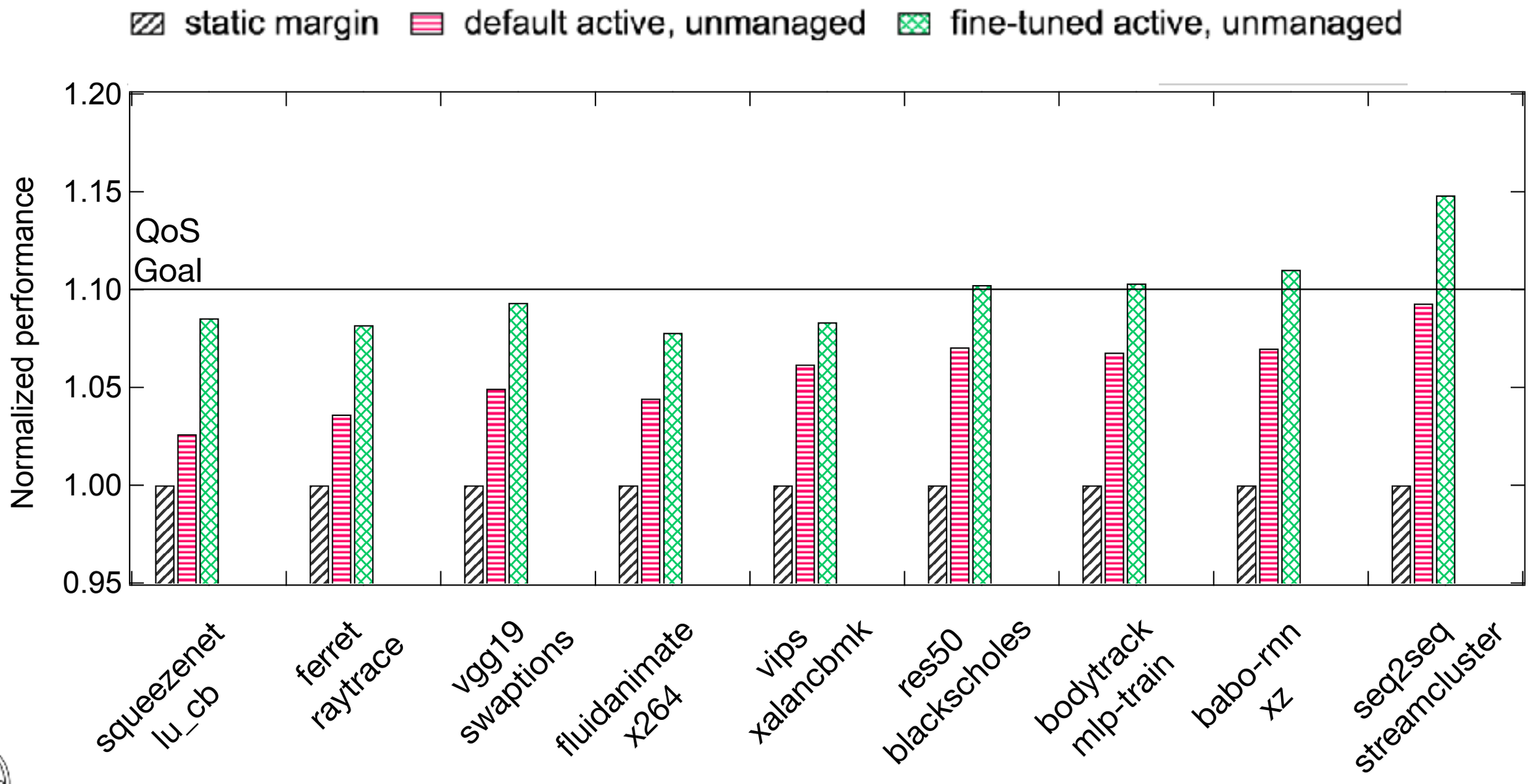


Performance Improvement

- Unmanaged DC voltage drop from co-runner apps
- 6.1% average performance increase.

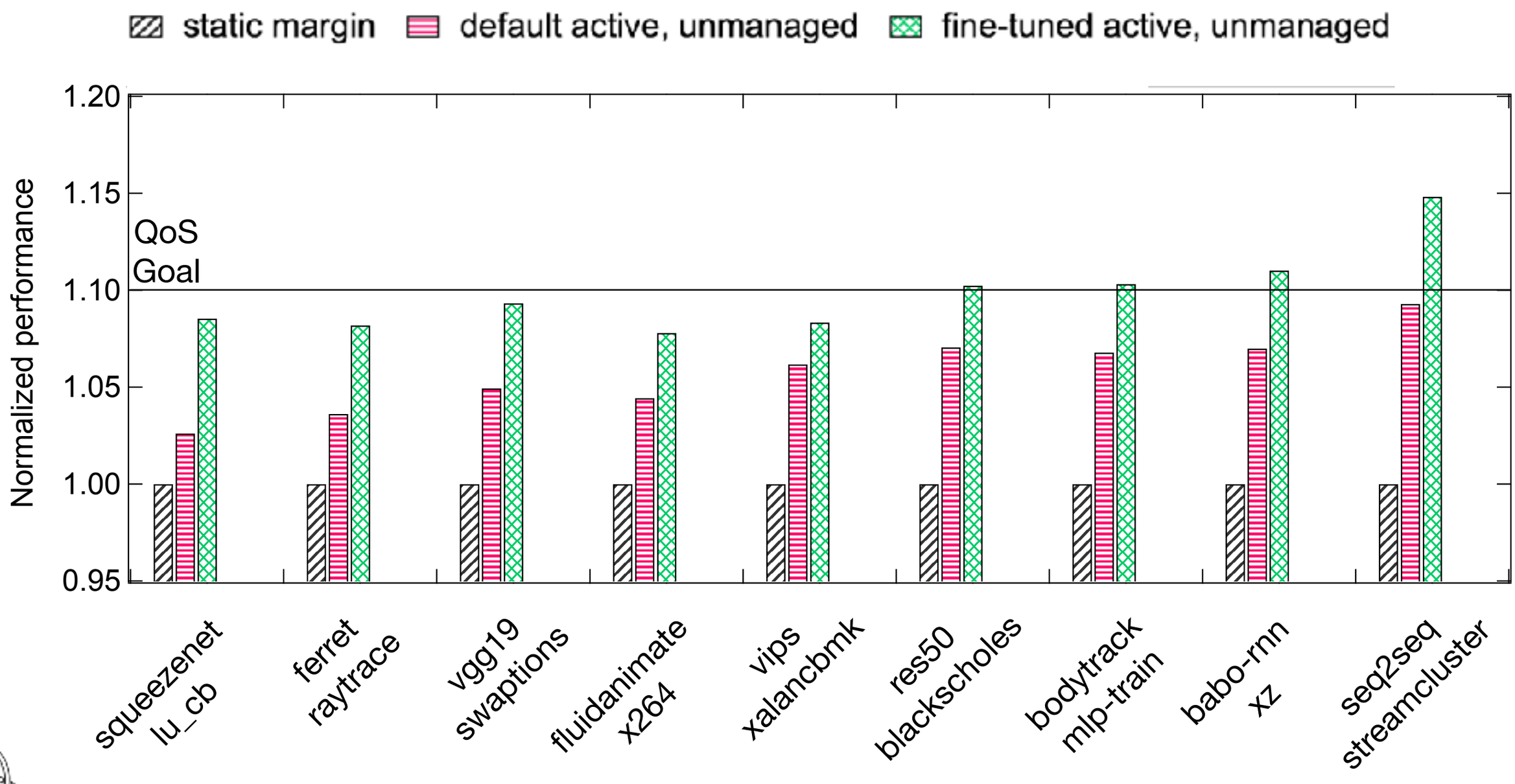


Performance Improvement



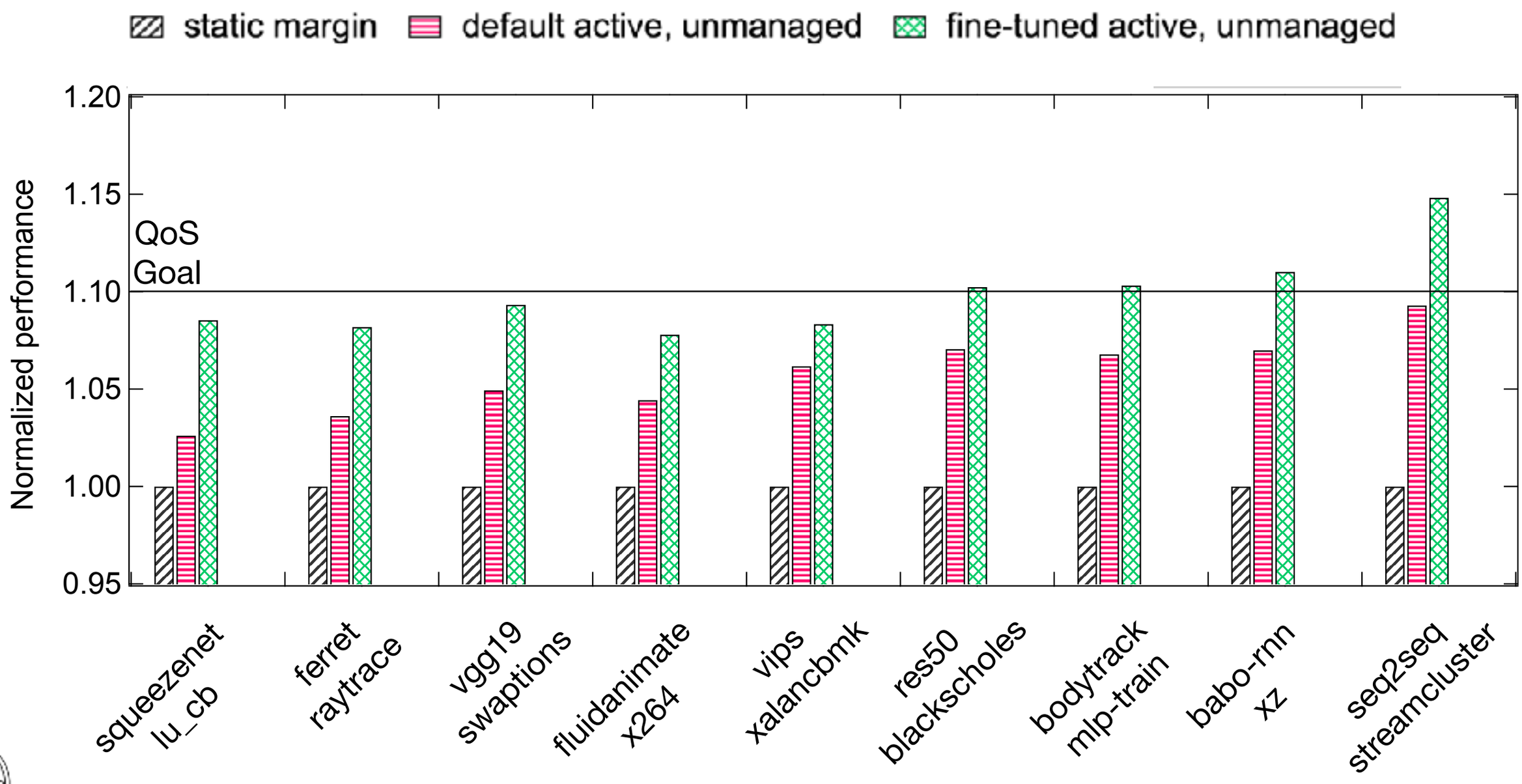
Performance Improvement

- Schedule to slow cores, with high power co-runner



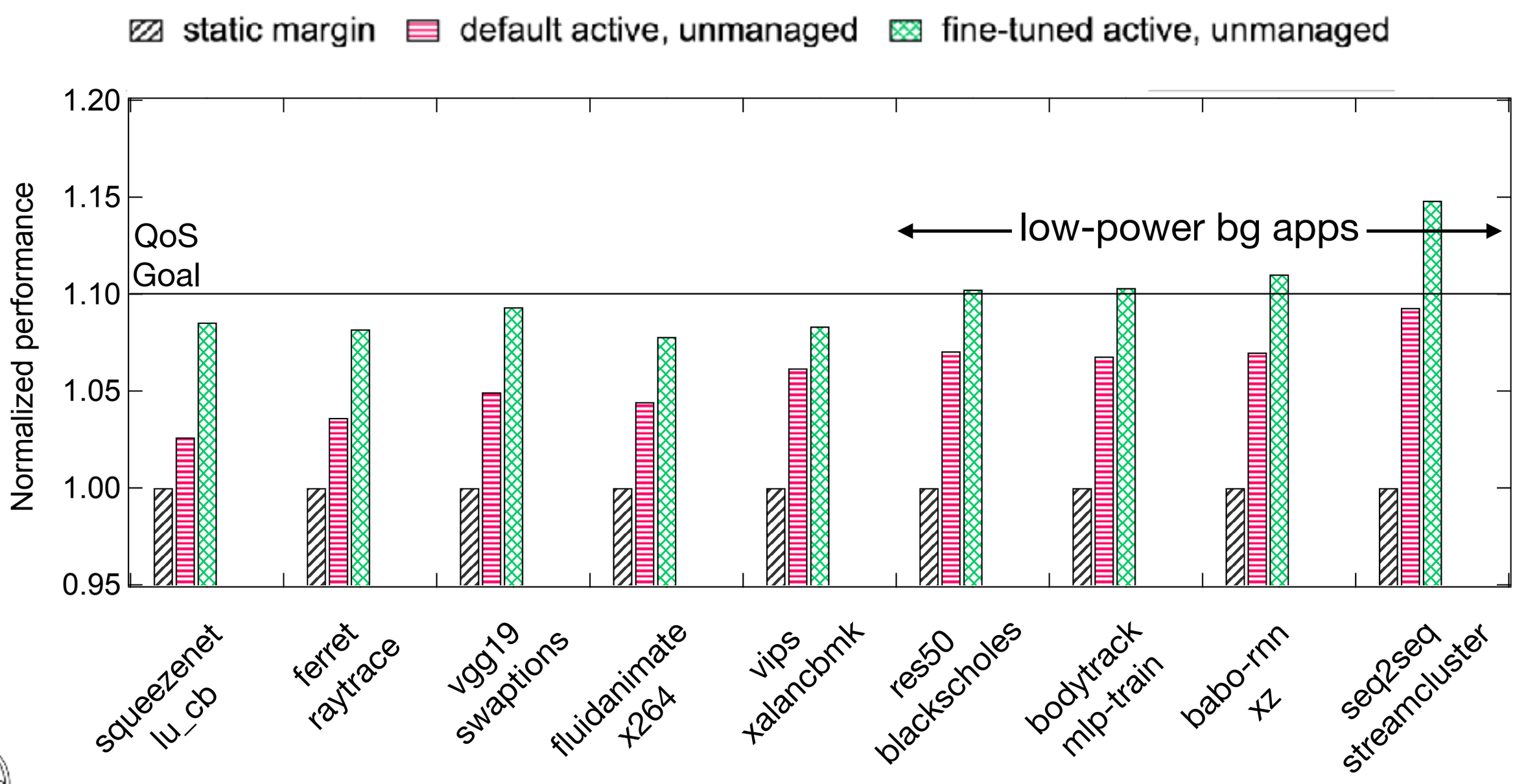
Performance Improvement

- Schedule to slow cores, with high power co-runner
- 10.2% average performance increase

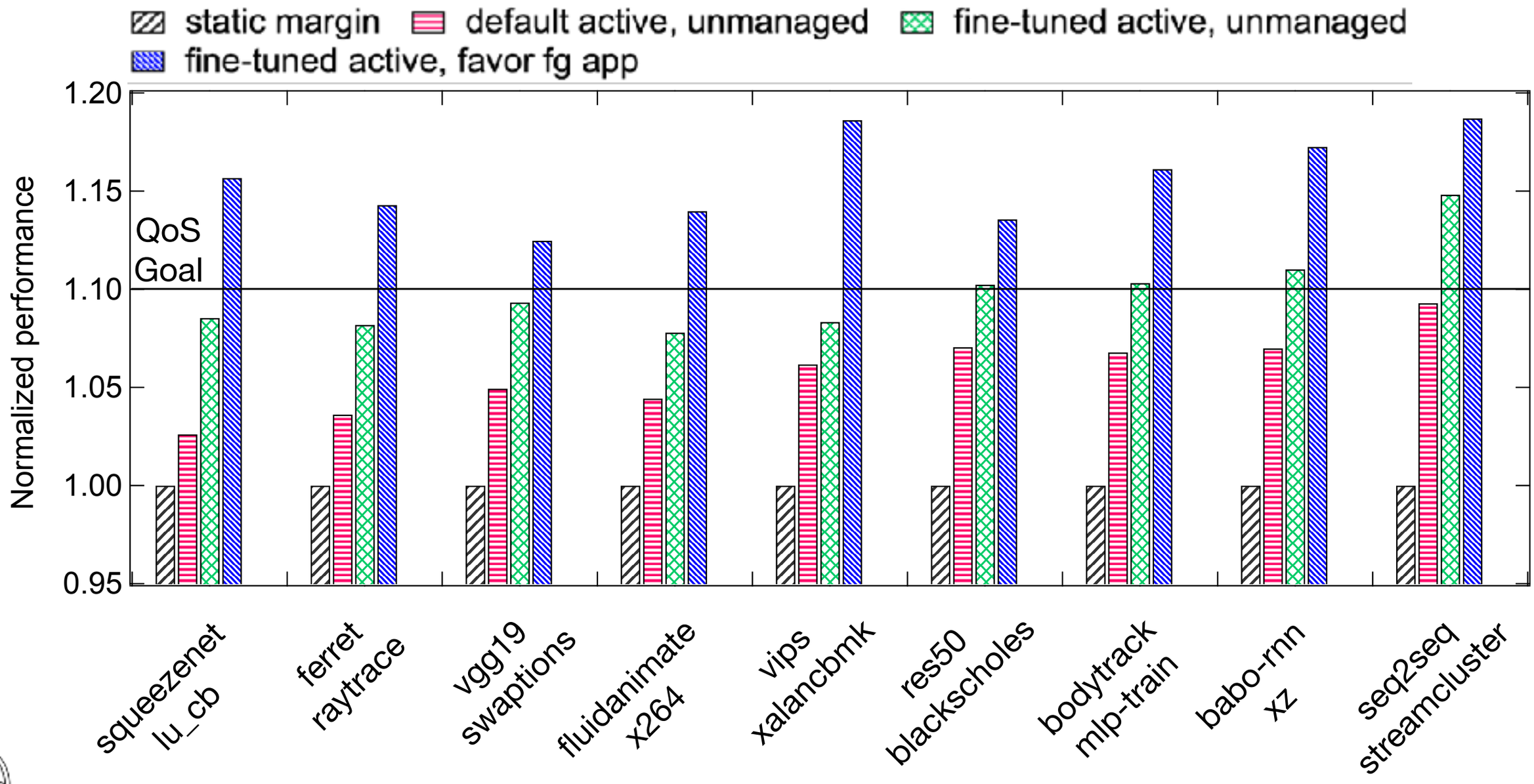


Performance Improvement

- Schedule to slow cores, with high power co-runner
- 10.2% average performance increase

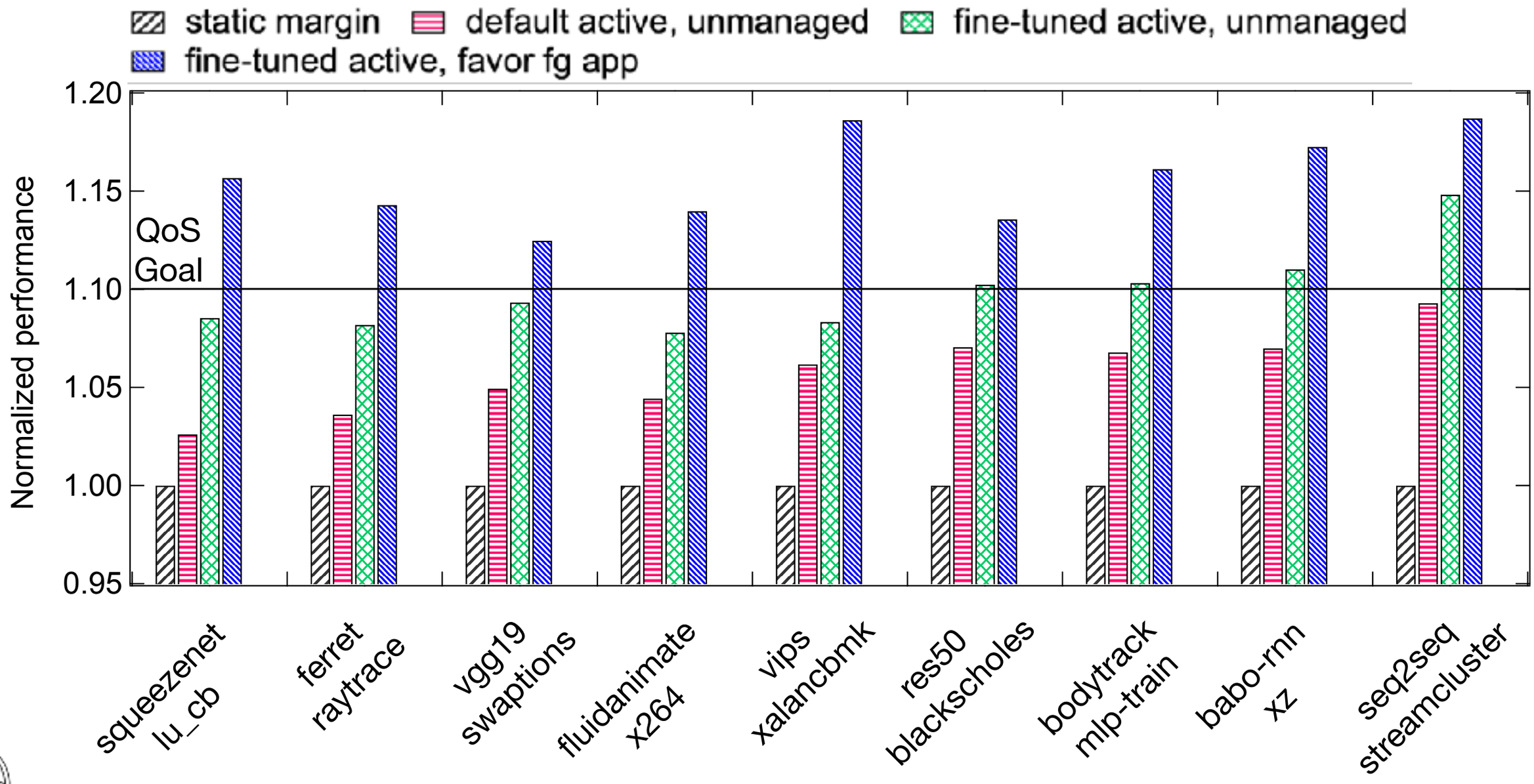


Performance Improvement



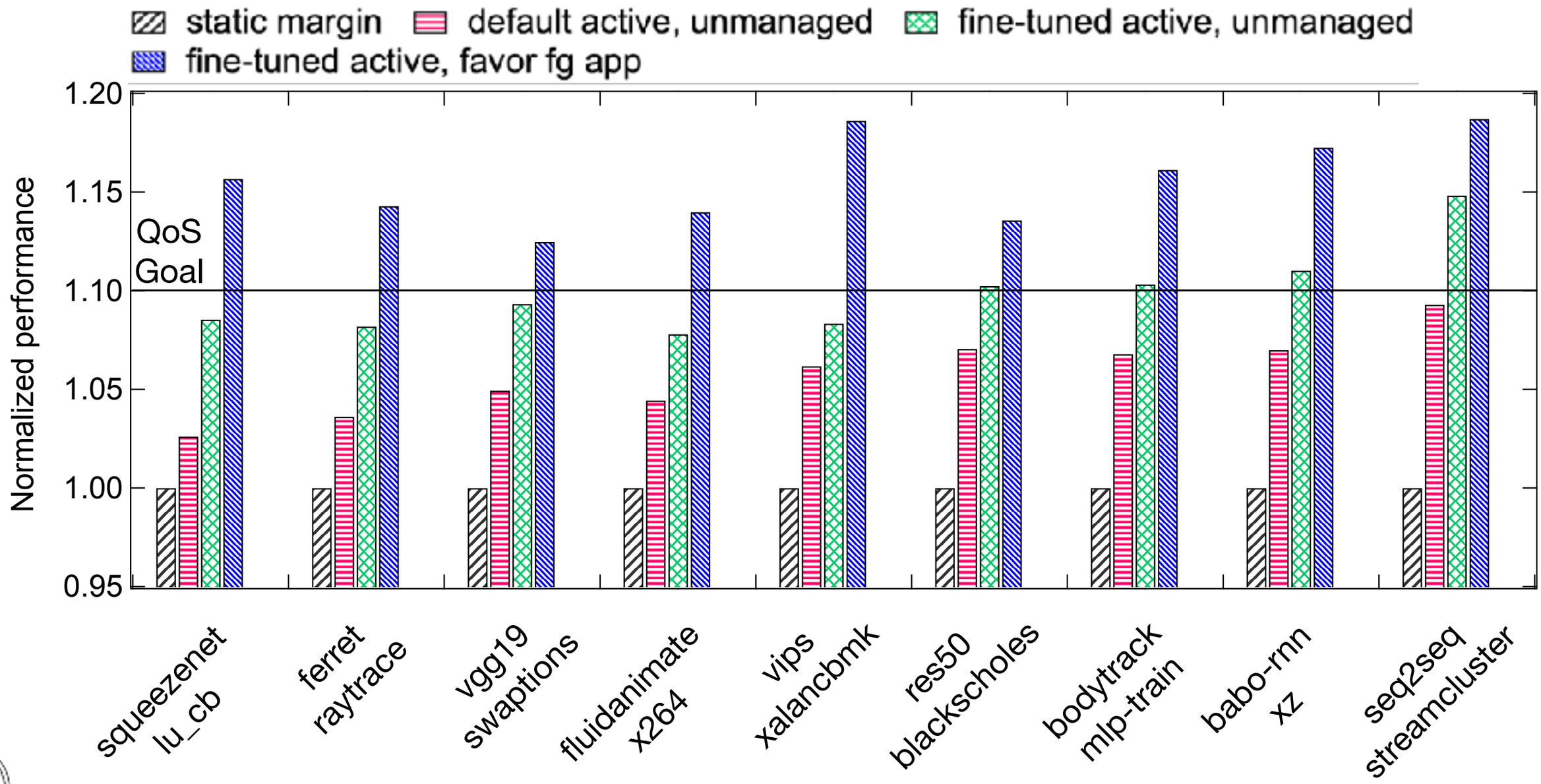
Performance Improvement

- Schedule to fast cores, and throttle co-runners to low p-states

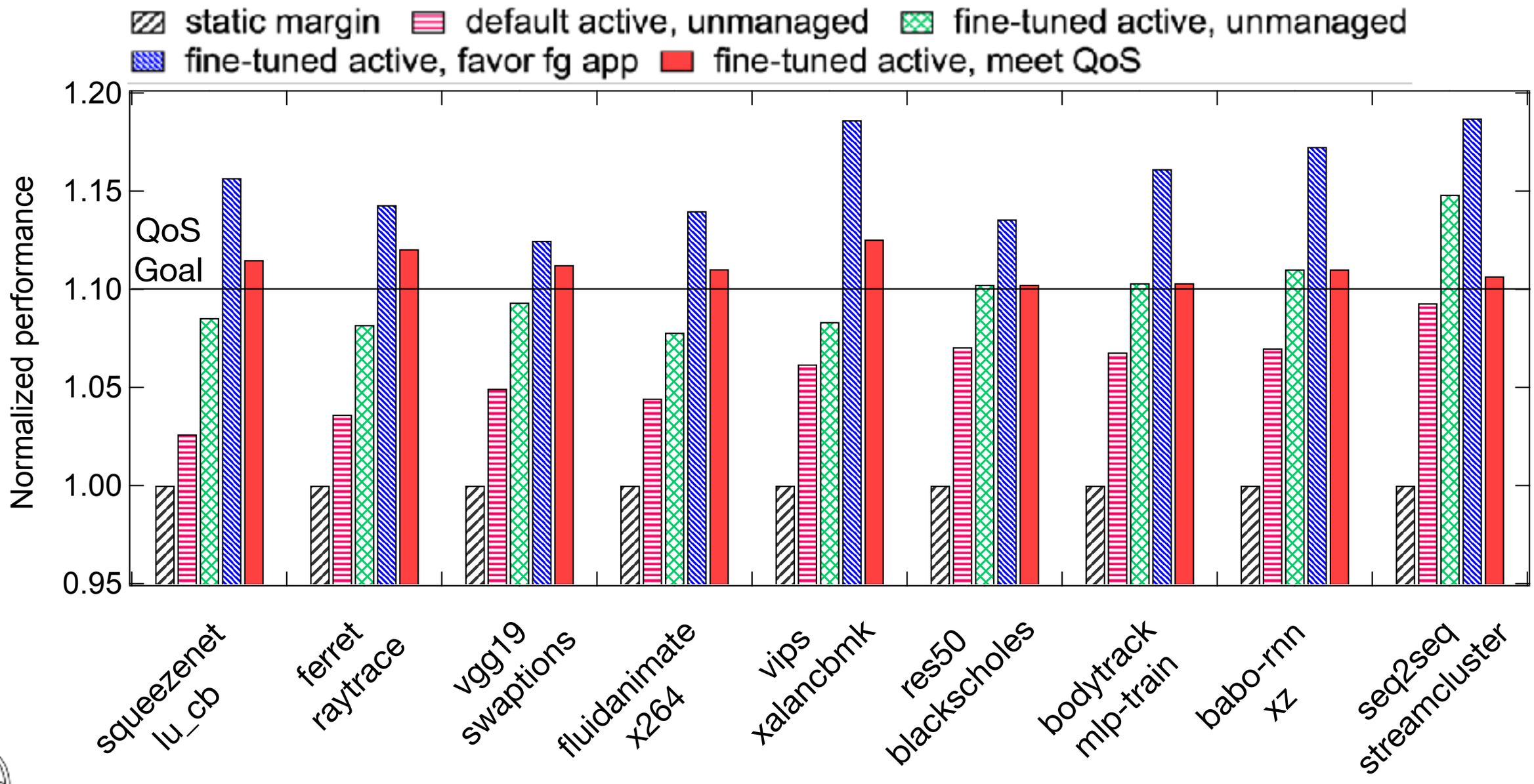


Performance Improvement

- Schedule to fast cores, and throttle co-runners to low p-states
- 15.2% average performance increase.

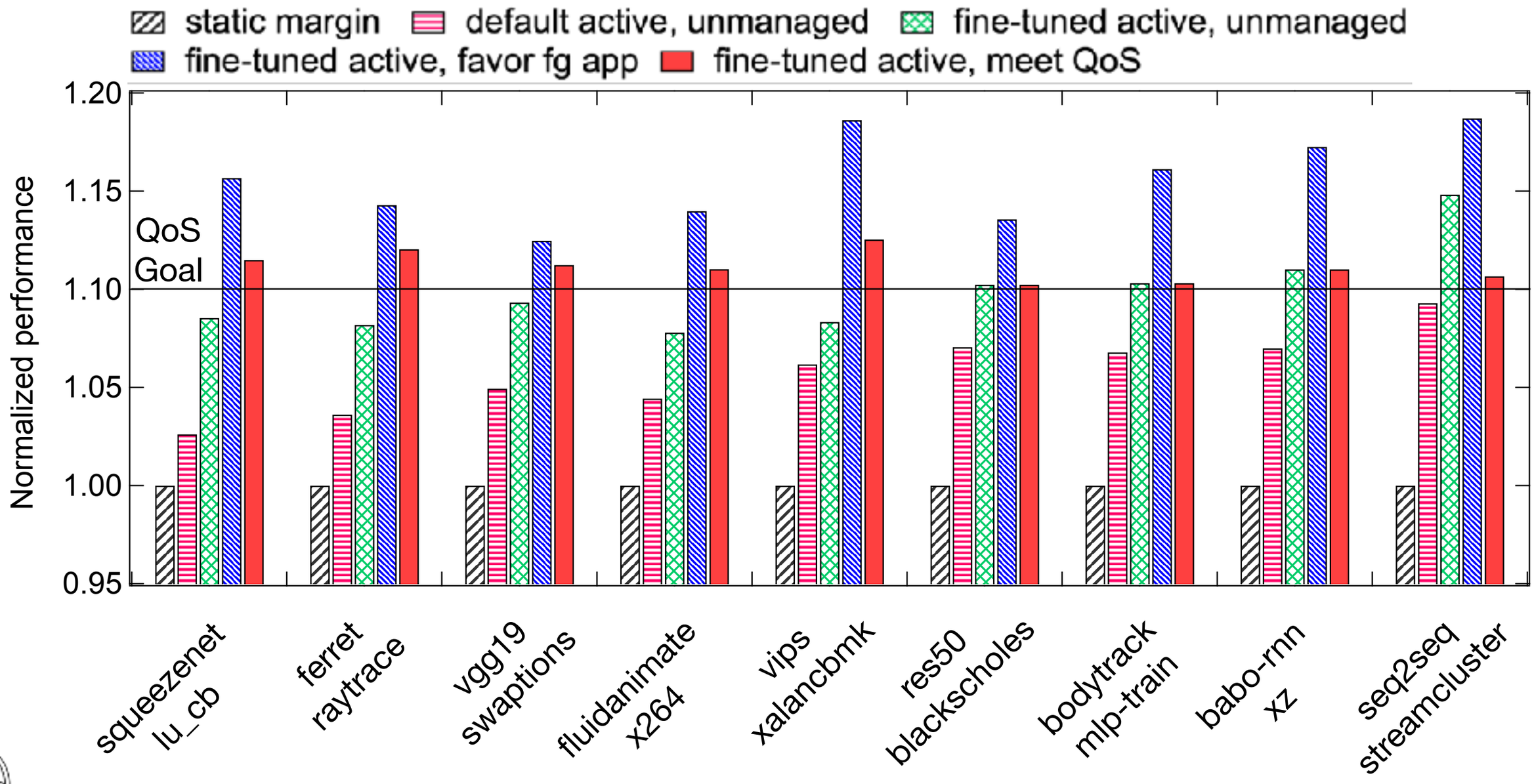


Performance Improvement



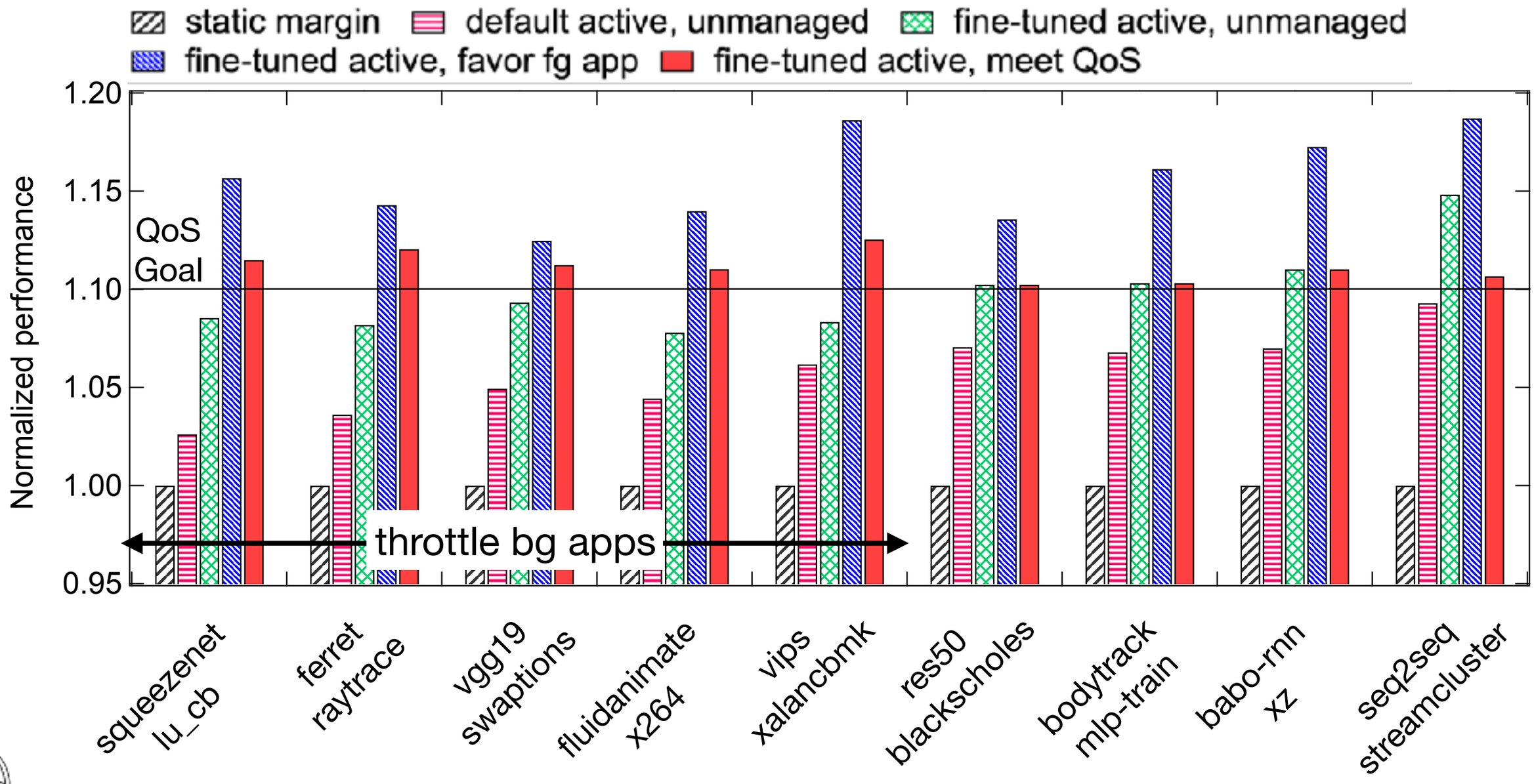
Performance Improvement

- Schedule to fast cores, and control co-runners' power



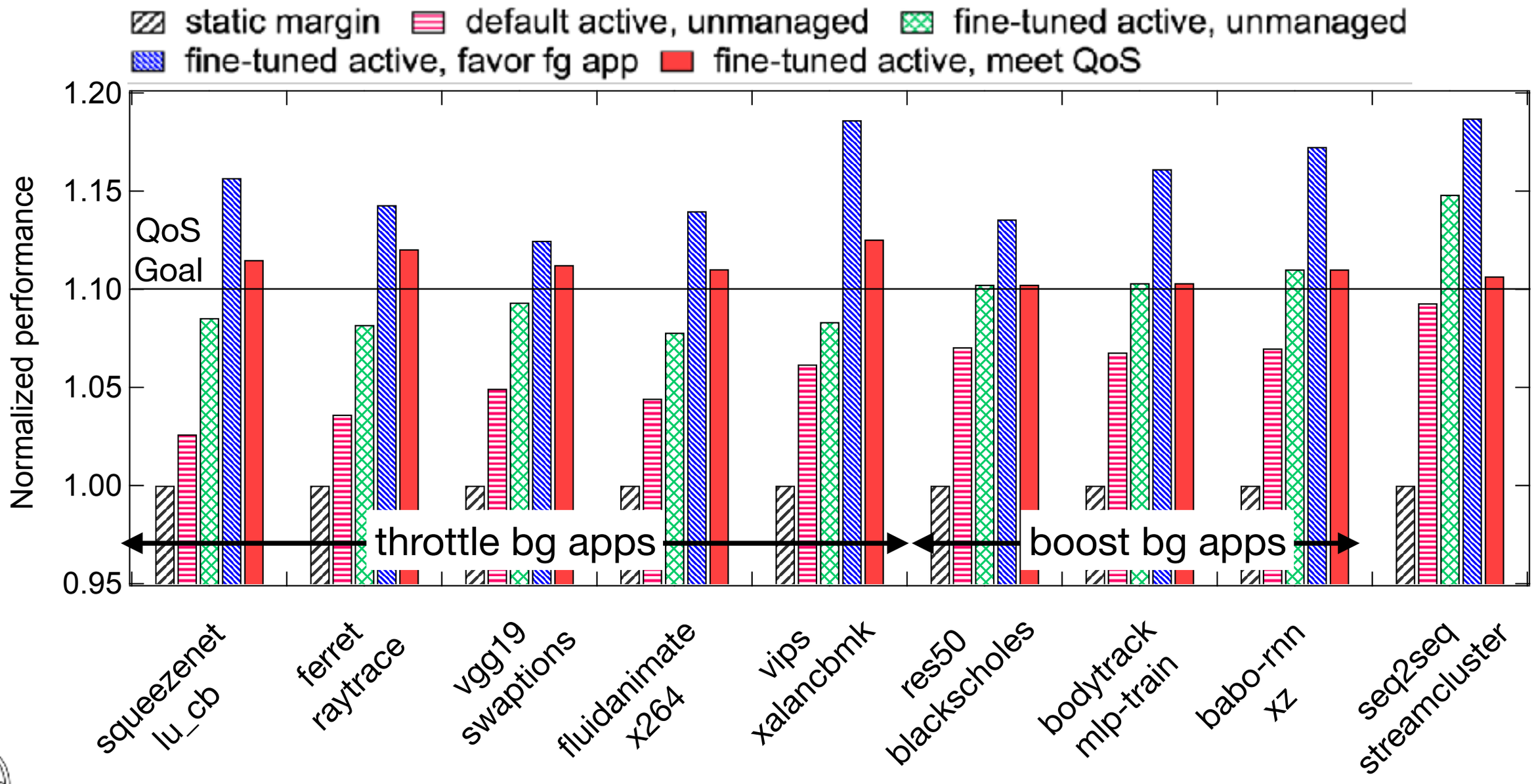
Performance Improvement

- Schedule to fast cores, and control co-runners' power



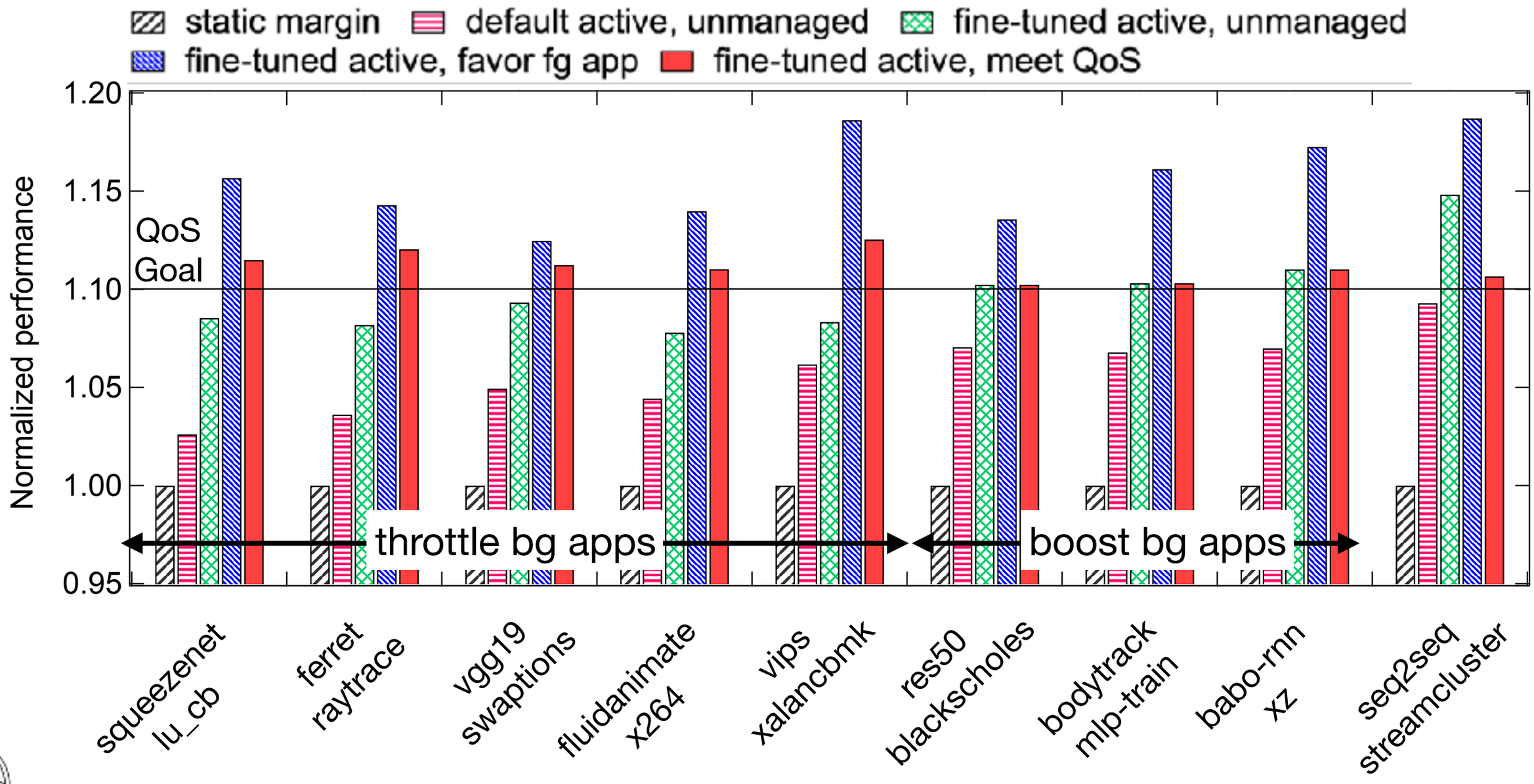
Performance Improvement

- Schedule to fast cores, and control co-runners' power



Performance Improvement

- Schedule to fast cores, and control co-runners' power
- Guaranteed 10% performance improvement.

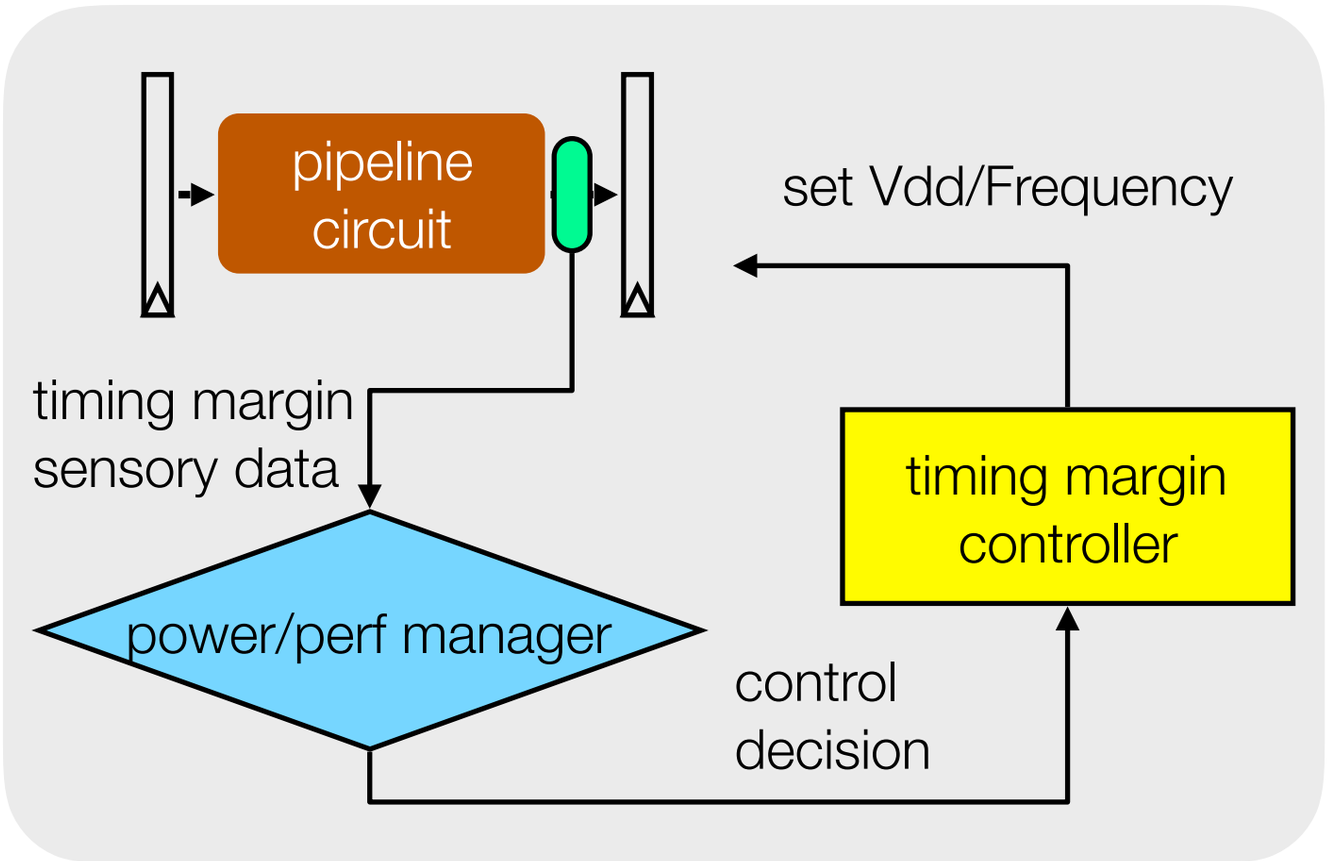


Managing Active Timing Margin

Applications

Software

Hardware

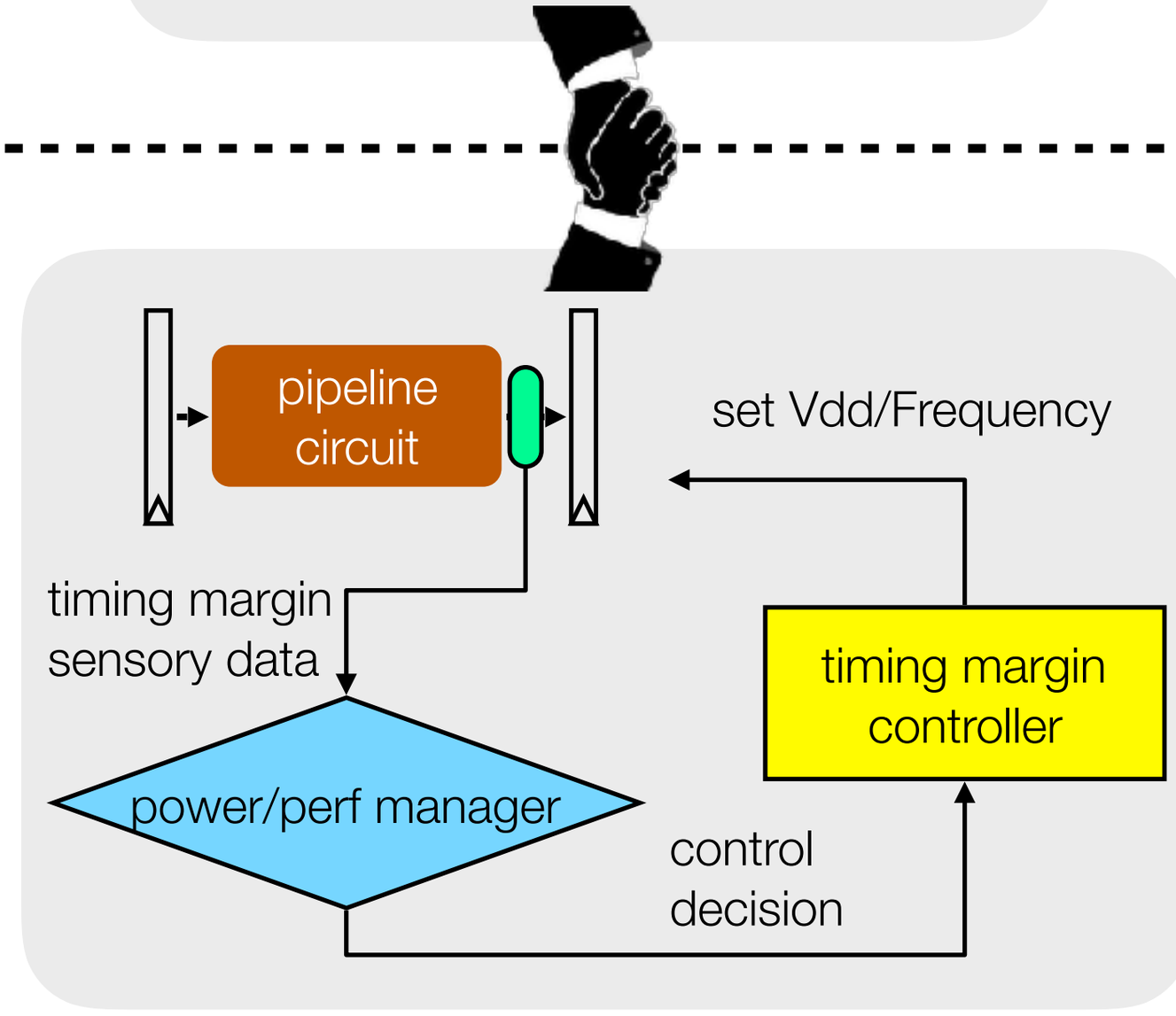


Managing Active Timing Margin - Bridging HW/SW

Applications

Software

Hardware



Backup

Active Management of Timing Guardband to Save Energy in POWER7*

Browse Journals & Magazines > IBM Journal of Research and D... > Volume: 59 Issue: 4/5

Robust power management in the IBM z13

A 22nm Dynamically Adaptive Clock Distribution for Voltage Droop Tolerance

8.5 **A 16nm Auto-Calibrating Dynamically Adaptive Clock Distribution for Maximizing Supply-Voltage-Droop Tolerance Across a Wide Operating Range**

margin for rising and falling input transitions to a positive value within a small TDE delay. The 3rd state inserts a positive timing margin into the DVM_{ROOT} by removing a programmable number of large TDEs so the ACD does not react to frequent small-magnitude V_{DD} variations. Finally, the FSM transitions back to the

16.4 **Adaptive Frequency and Biasing Techniques for Tolerance to Dynamic Temperature-Voltage Variations and Aging**

tion of the core when the TCP/IP header has finished processing and the core is idle. Both NMOS and PMOS body bias generators are implemented on the die and each includes a central bias generator (CBG) which is controlled by the DAB control, and many local bias

5.7 **A Graphics Execution Core in 22nm CMOS Featuring Adaptive Clocking, Selective Boosting and State-Retentive Sleep**

GRF bitcells to be connected to V_{BOOST} which acts as an always-on supply (Fig. 5.7.2). The bitcells are also disconnected from the WBLs. For storage of critical state distributed in the execution core, state-retention sequentials (Fig. 5.7.3) isolate the slave storage node during sleep and connect to an always-on "$V_{...}$"

5.6 **Adaptive Clocking System for Improved Power Efficiency in a 28nm x86-64 Microprocessor**

To create an instantly reduced clock frequency, a phase generator with a DLL is used to generate 20 phases of the PLL output clock. The DLL operates at a regulated voltage. To support finer stretch amounts, the 20 clock phases are

18.1 **Droop Mitigation using Critical-Path Sensors and an On-Chip Distributed Power Supply Estimation Engine in the z14™ Enterprise Processor**

in a single power-management unit and a global per-core throttling signal was sent to the targeted units for instruction throttling. In this design, each CPM still sends its output to the per-core power-management unit, but is also involved in a local loop. The local loop only uses a single CPM and a local copy of the power-management logic; it uses a process similar to an original loop, but with different filter, threshold, and slope parameters. Each local loop throttles a single unit in the core, but has a much faster response time. The local throttling signals are merged appropriately with the full core throttling loop signals to avoid excessive throttling.

The third new technique is to use CPM information from neighboring cores. Measurements of the previous generation modules revealed that power supply noise generated in one core propagates to neighboring cores approximately 5ns later [4]. CPM droop information can be sent to neighboring cores faster than

Christos Vezyrtzis[1], Thomas Strach[2], Pierce I-Jen Chuang[1], Preetham Lobo[3], Richard Rizzolo[4], Tobias Webel[2], Pawel Owczarczyk[4], Alper Buyuktosunoglu[1], Ramon Bertran[1], David Hui[4], Susan M. Eickhoff[4], Michael Floyd[5], Gerard Salem[6], Sean Carey[4], Stelios G. Tsapepas[4], Phillip J. Restle[1]

[1]IBM Research, Yorktown Heights, NY; [2]IBM STG, Boeblingen, Germany
[3]IBM STG, Bangalore, India; [4]IBM STG, Poughkeepsie, NY; [5]IBM STG, Austin, TX
[6]IBM STG, Essex Junction, VT

Active Management of Timing Guardband to Save Energy in POWER7*

Browse Journals & Magazines > IBM Journal of Research and D... > Volume: 59 Issue: 4/5

Robust power management in the IBM z13

A 22nm Dynamically Adaptive Clock Distribution for Voltage Droop Tolerance

8.5 **A 16nm Auto-Calibrating Dynamically Adaptive Clock Distribution for Maximizing Supply-Voltage-Droop Tolerance Across a Wide Operating Range**

margin for rising and falling input transitions to a positive value within a small TDE delay. The 3rd state inserts a positive timing margin into the DVM_{ROOT} by removing a programmable number of large TDEs so the ACD does not react to frequent small-magnitude V_{DD} variations. Finally, the FSM transitions back to the

16.4 **Adaptive Frequency and Biasing Techniques for**

tion of the core when the TCP/IP header has finished processing and

▷ Active timing margin saves 10%~20% processor power

5.6 **Adaptive Clocking System for Improved Power Efficiency in a 28nm x86-64 Microprocessor**

To create an instantly reduced clock frequency, a phase generator with a DLL is used to generate 20 phases of the PLL output clock. The DLL operates at a

18.1 **Droop Mitigation using Critical-Path Sensors and an On-Chip Distributed Power Supply Estimation Engine in the z14™ Enterprise Processor**

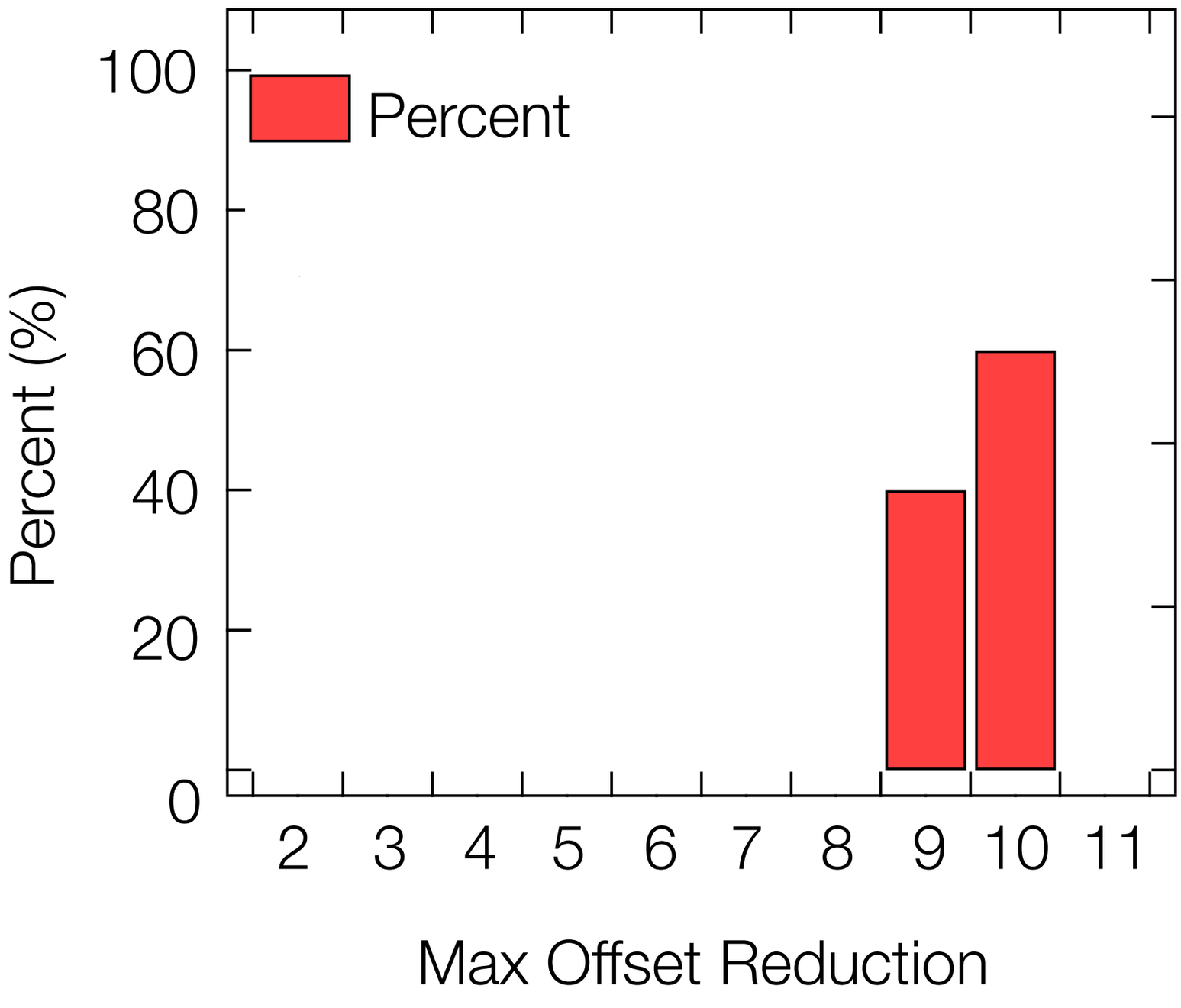
Christos Vezyrtzis[1], Thomas Strach[2], Pierce I-Jen Chuang[1], Preetham Lobo[3], Richard Rizzolo[4], Tobias Webel[2], Pawel Owczarczyk[4], Alper Buyuktosunoglu[1], Ramon Bertran[1], David Hui[4], Susan M. Eickhoff[4], Michael Floyd[5], Gerard Salem[6], Sean Carey[4], Stelios G. Tsapepas[4], Phillip J. Restle[1]

[1]IBM Research, Yorktown Heights, NY; [2]IBM STG, Boeblingen, Germany
[3]IBM STG, Bangalore, India; [4]IBM STG, Poughkeepsie, NY; [5]IBM STG, Austin, TX
[6]IBM STG, Essex Junction, VT

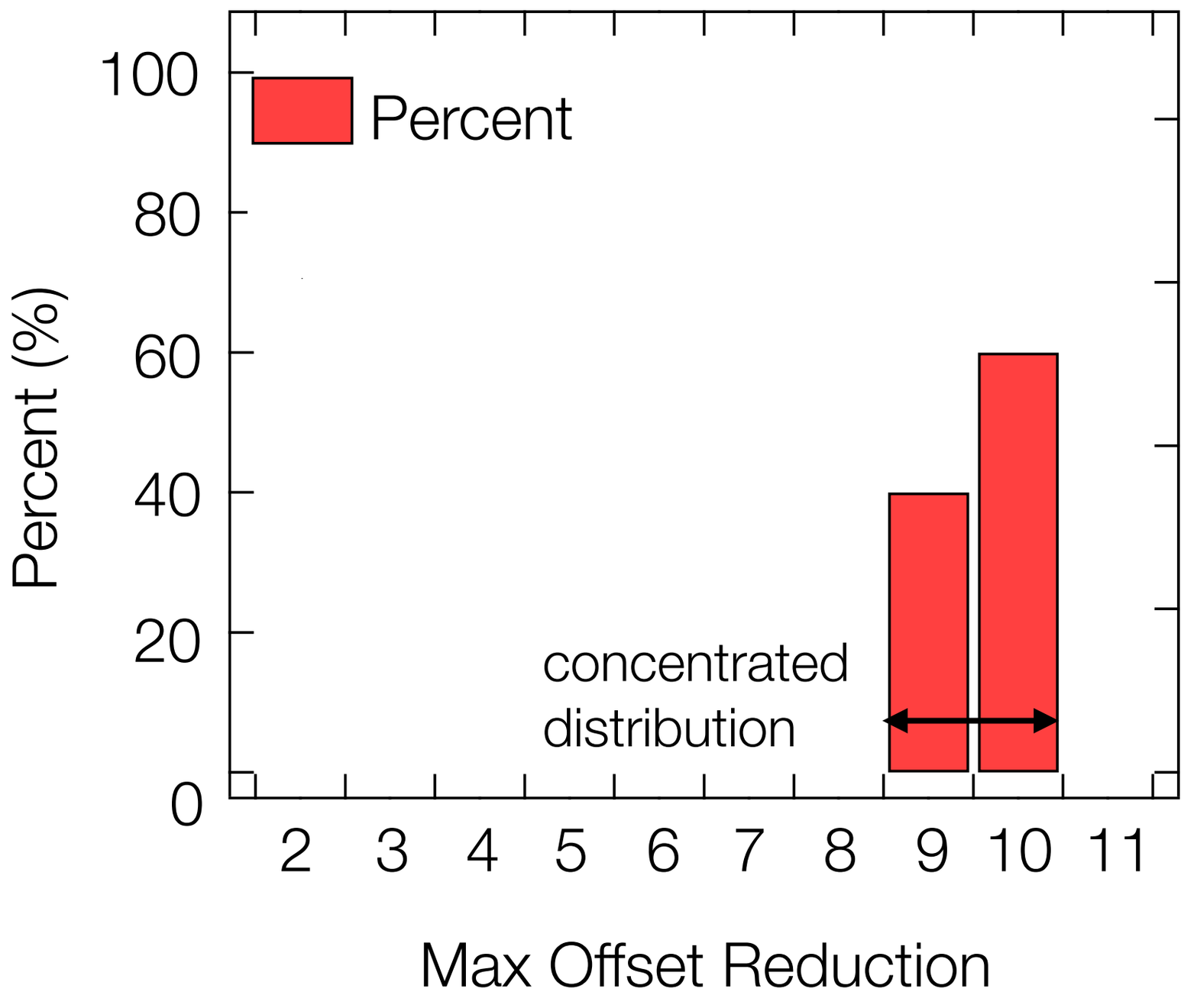
in a single power-management unit and a global per-core throttling signal was sent to the targeted units for instruction throttling. In this design, each CPM still sends its output to the per-core power-management unit, but is also involved in a local loop. The local loop only uses a single CPM and a local copy of the power-management logic; it uses a process similar to an original loop, but with different filter, threshold, and slope parameters. Each local loop throttles a single unit in the core, but has a much faster response time. The local throttling signals are merged appropriately with the full core throttling loop signals to avoid excessive throttling.

The third new technique is to use CPM information from neighboring cores. Measurements of the previous generation modules revealed that power supply noise generated in one core propagates to neighboring cores approximately 5ns later [4]. CPM droop information can be sent to neighboring cores faster than

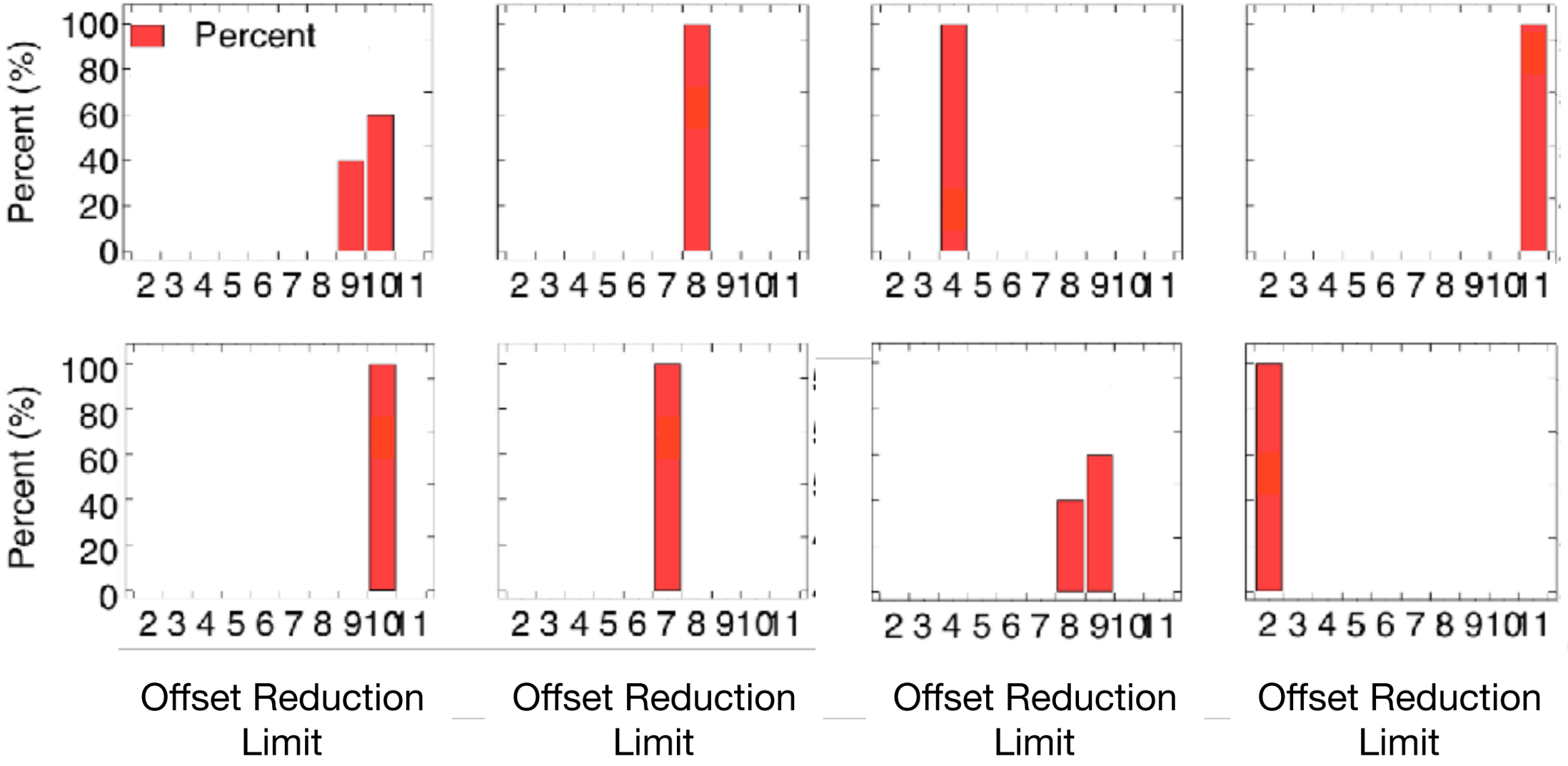
Fine-tuning Active Timing Margin: The Limits



Fine-tuning Active Timing Margin: The Limits



Fine-tuning Active Timing Margin: The Limits



Fine-tuning Active Timing Margin: The Limits