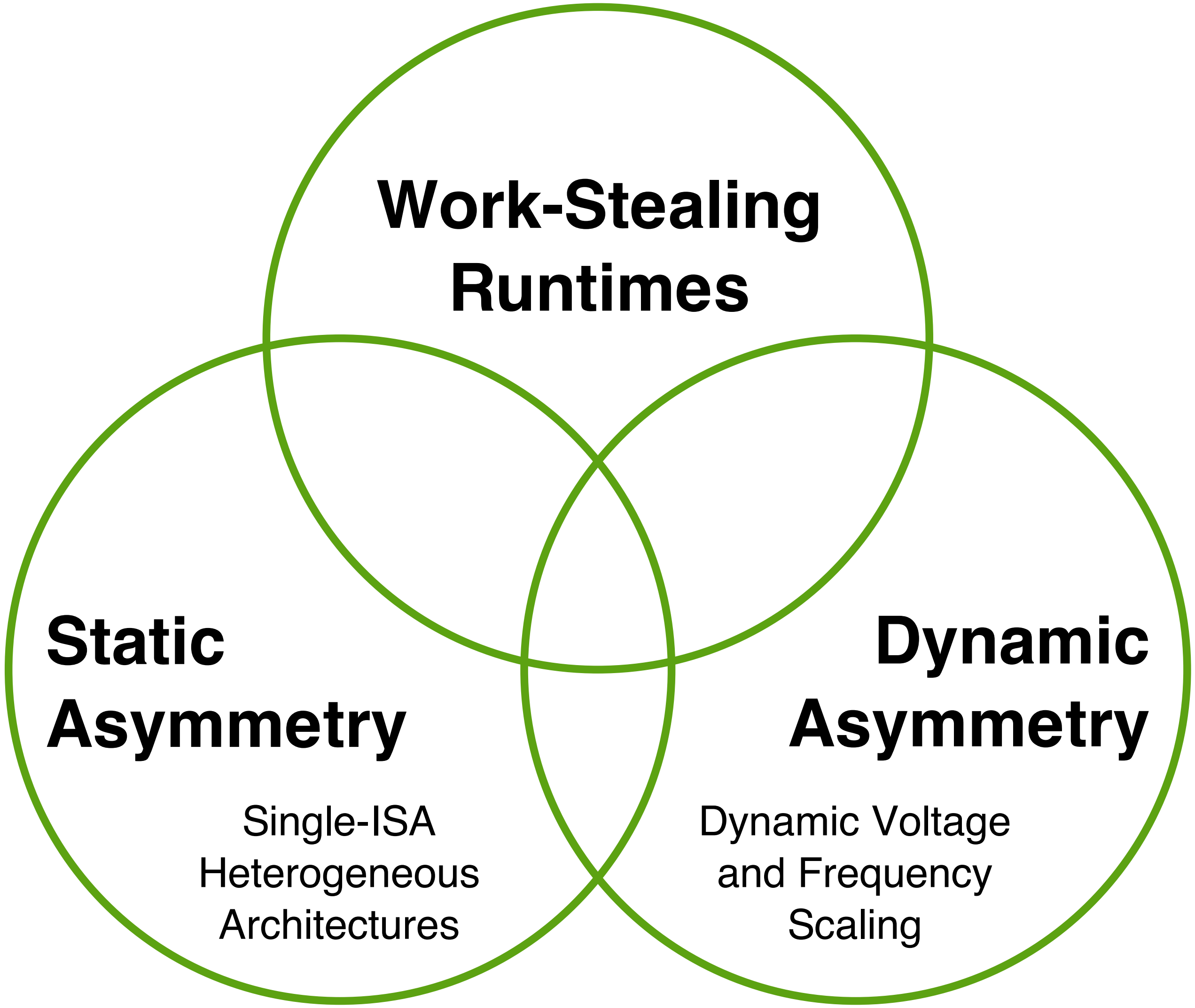
Asymmetry-Aware Work-Stealing Runtimes

Christopher Torng, Moyang Wang, and
Christopher Batten

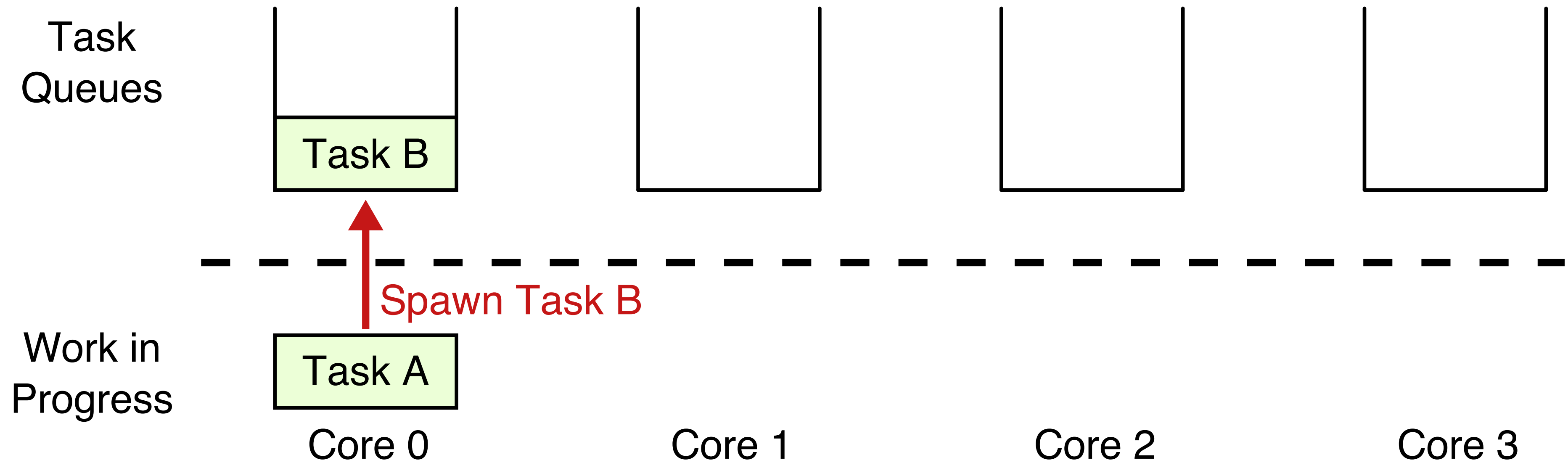
School of Electrical and Computer Engineering
Cornell University

43rd Int'l Symp. on Computer Architecture, June 2016

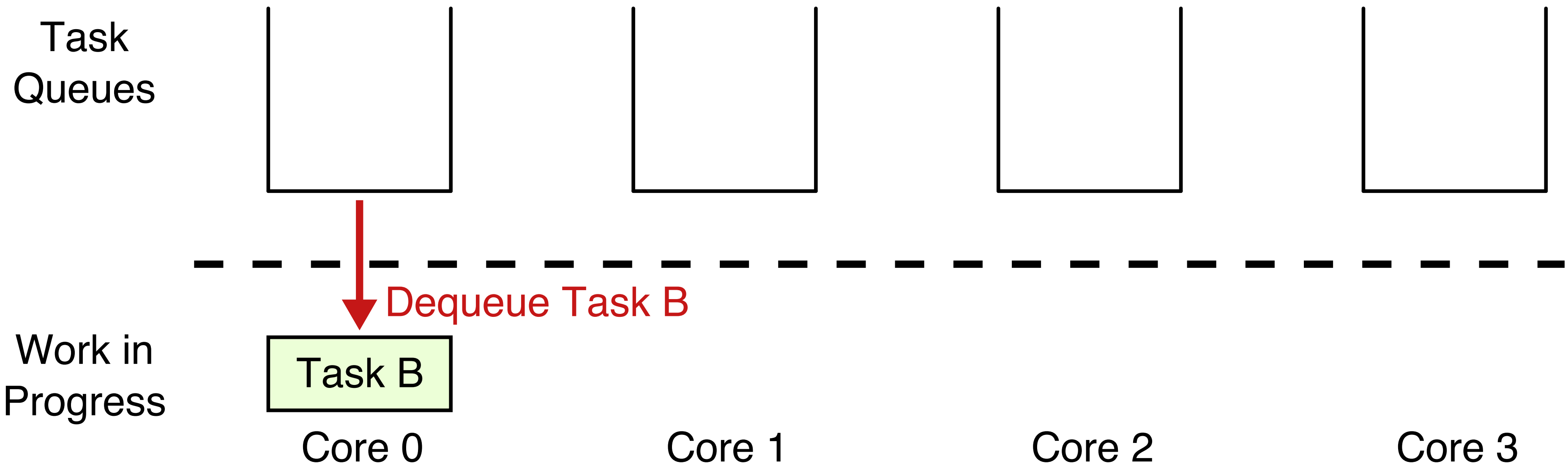


How can we use asymmetry awareness to improve the performance and energy efficiency of a work-stealing runtime?

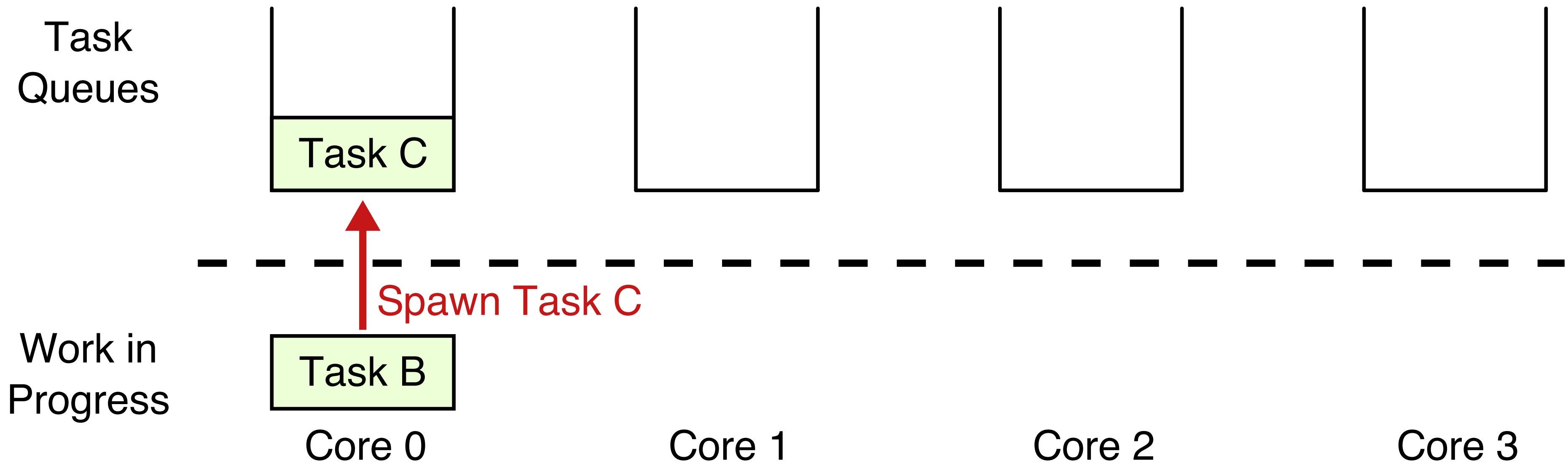
Work-Stealing Runtimes



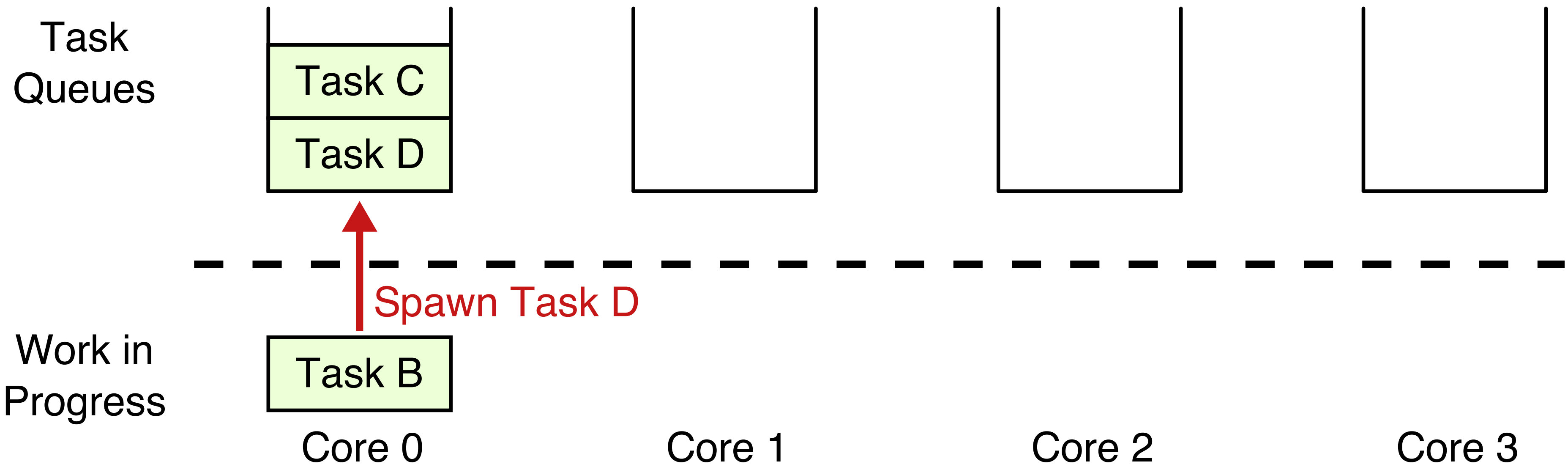
Work-Stealing Runtimes



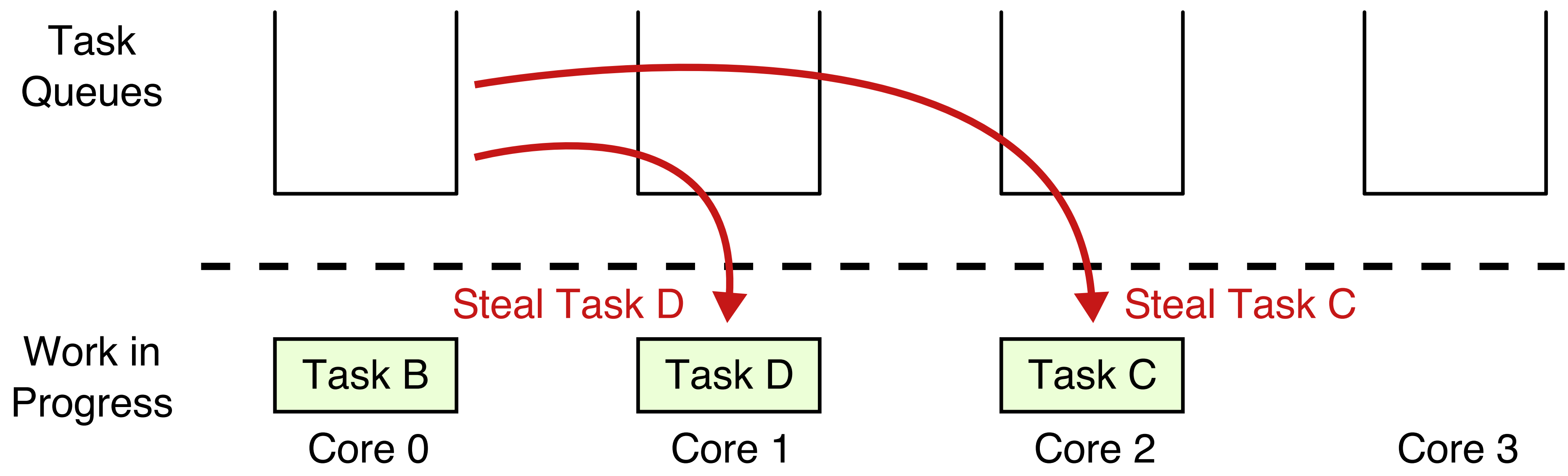
Work-Stealing Runtimes



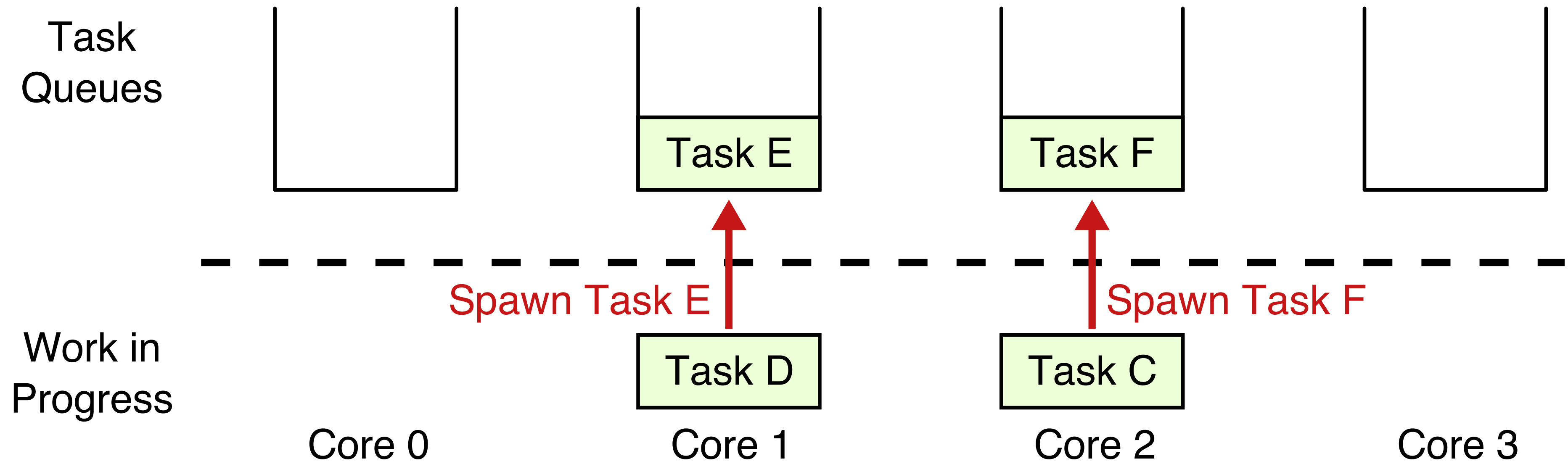
Work-Stealing Runtimes



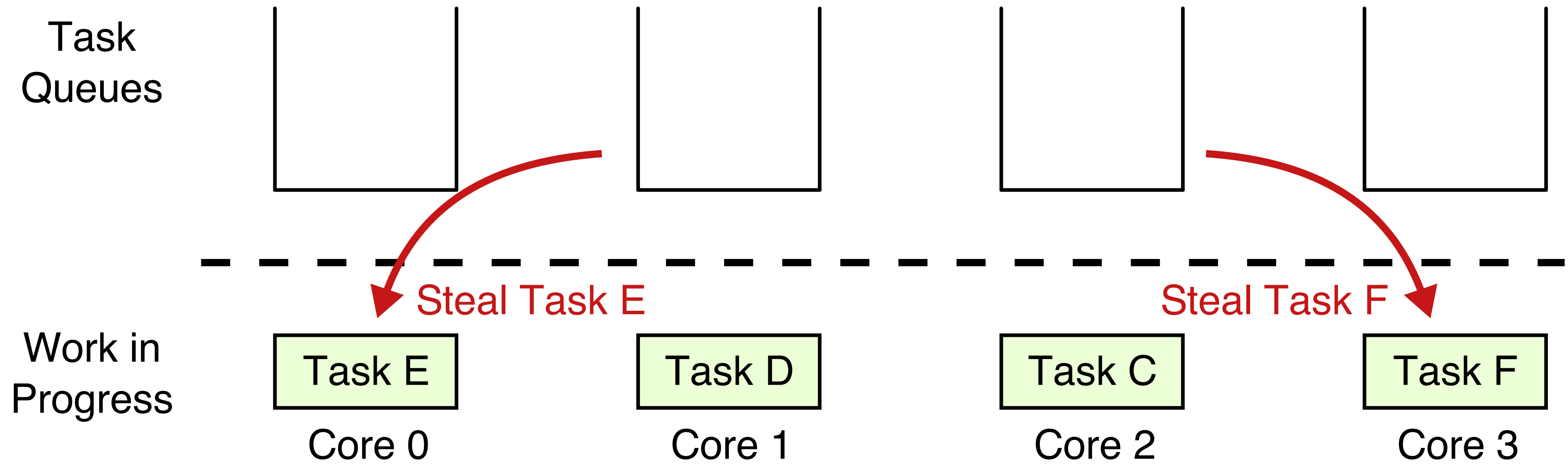
Work-Stealing Runtimes



Work-Stealing Runtimes

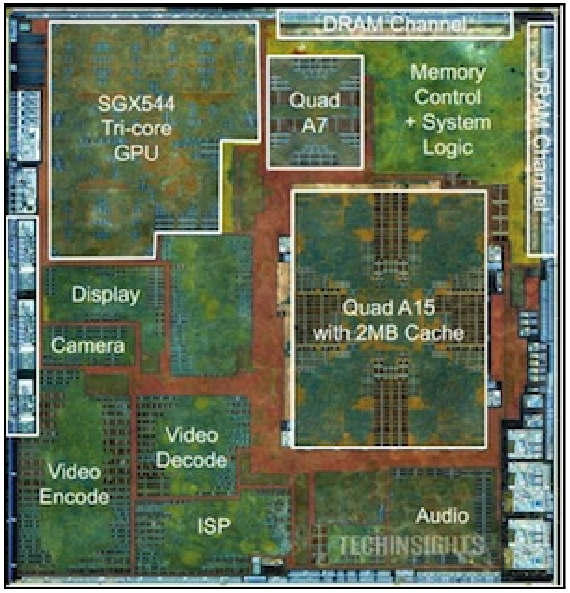


Work-Stealing Runtimes

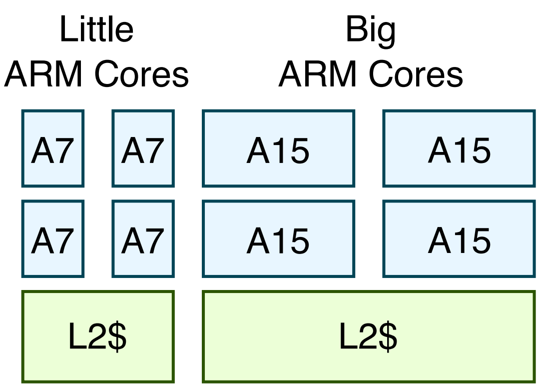


- Work stealing has good performance, space requirements, and communication overheads in both theory and practice
- Supported in many popular concurrency platforms including: Intel's Cilk Plus, Intel's C++ TBB, Microsoft's .NET Task Parallel Library, Java's Fork/Join Framework, and OpenMP

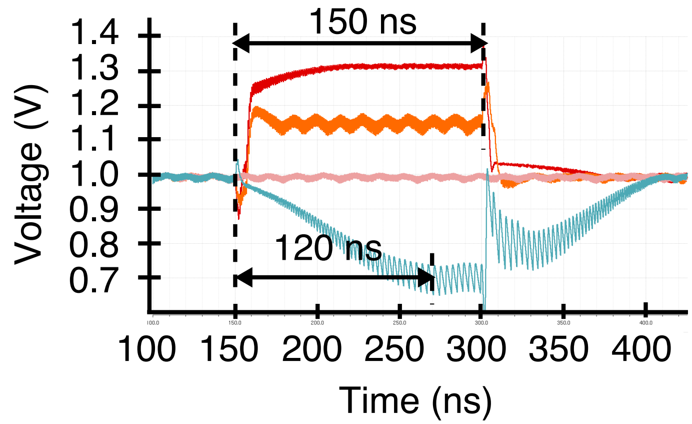
Static Asymmetry vs. Dynamic Asymmetry

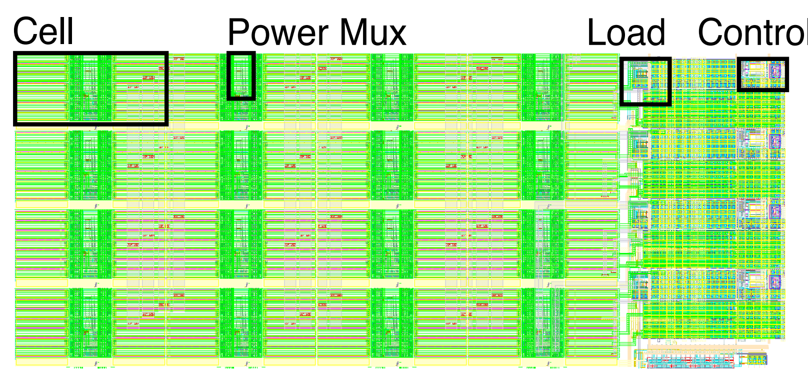


Samsung Exynos Octa
Mobile Processor



Integrated Voltage Regulation





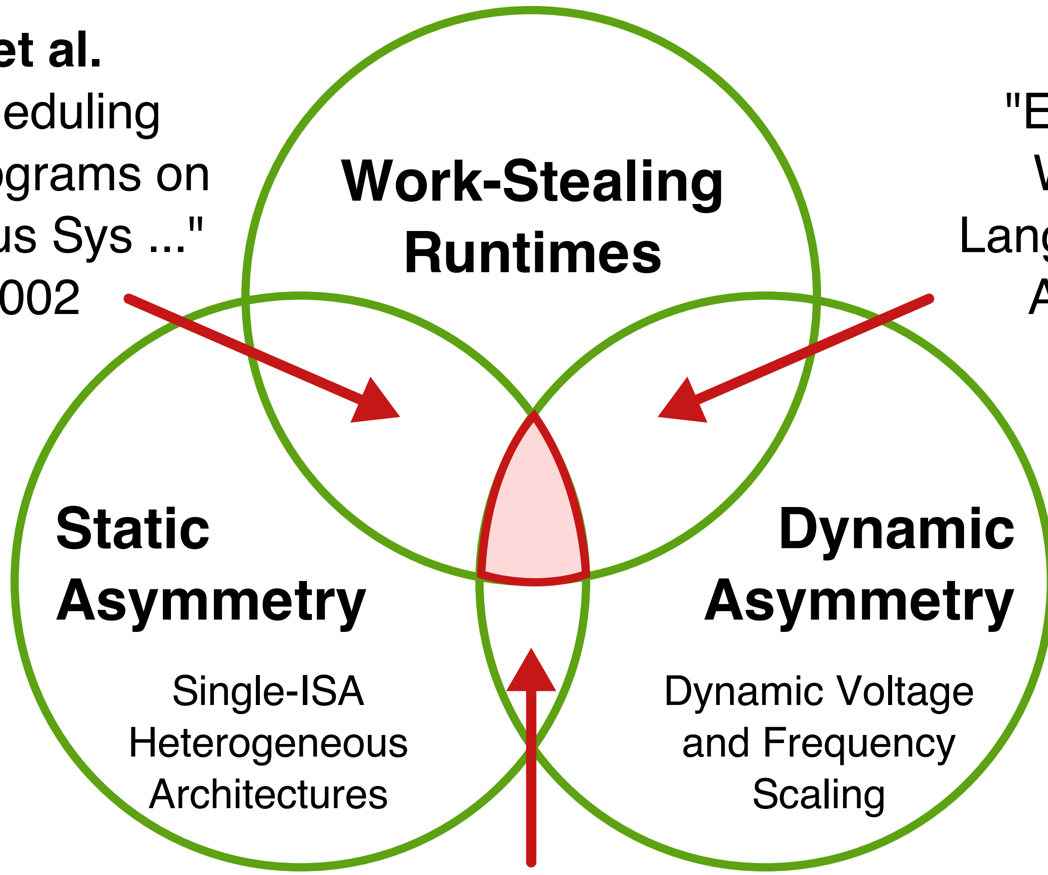
Test Chip with Four Integrated Voltage Regulators

From W, Godycki, C. Torng, I. Bukreyev, A. Apsel, C. Batten. "Enabling Realistic Fine-Grain Voltage Scaling with Reconfigurable Power Distribution Networks" MICRO, 2014

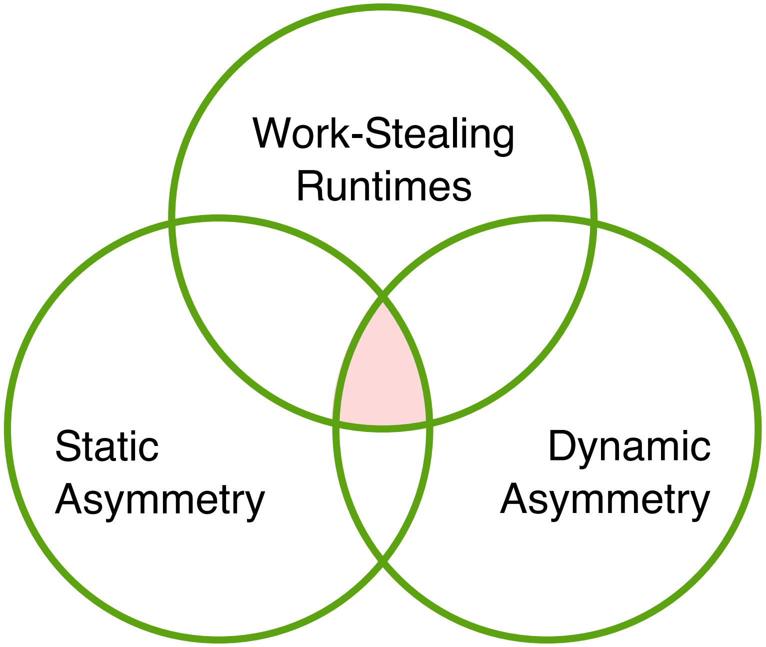
How can we use asymmetry awareness to improve the performance and energy efficiency of a work-stealing runtime?

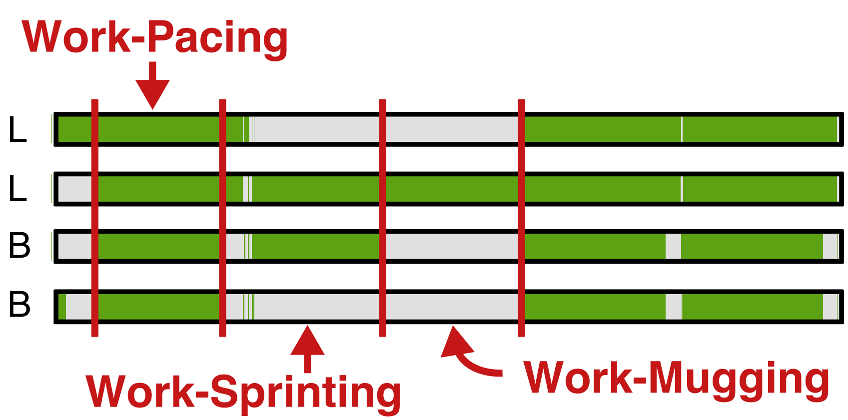
Bender et al.
"Online Scheduling of Parallel Programs on Heterogeneous Sys ..."
SPAA 2002

Ribic et al.
"Energy-Efficient Work-Stealing Language Runtimes"
ASPLOS 2014



Azizi et al.
"Energy-performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis" ISCA 2010





Talk Outline

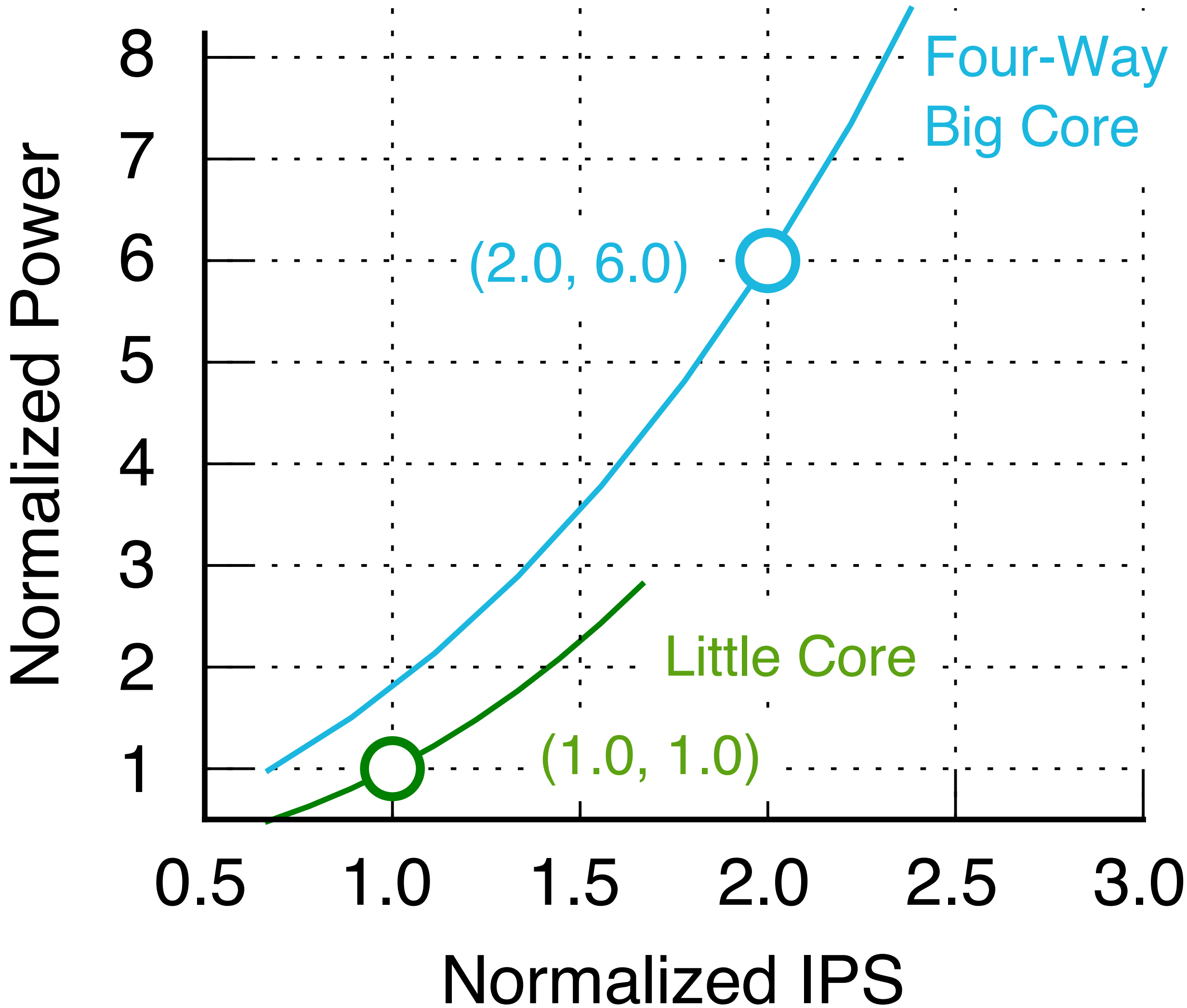
Motivation

First-Order Modeling

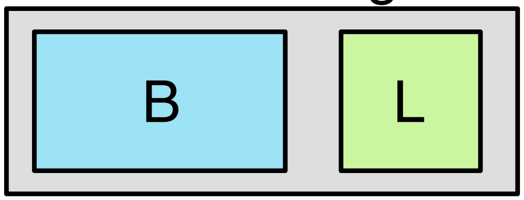
Asymmetry-Aware Work-Stealing Runtimes

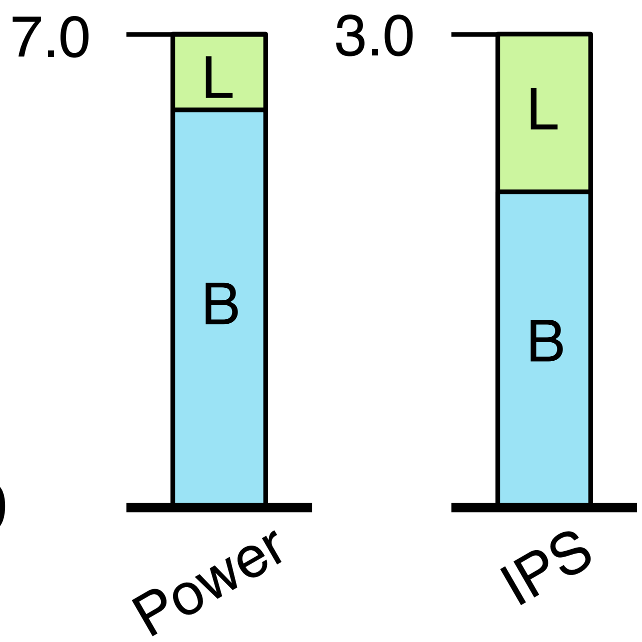
Evaluation

Building Intuition by Exploring a 1 Big 1 Little System

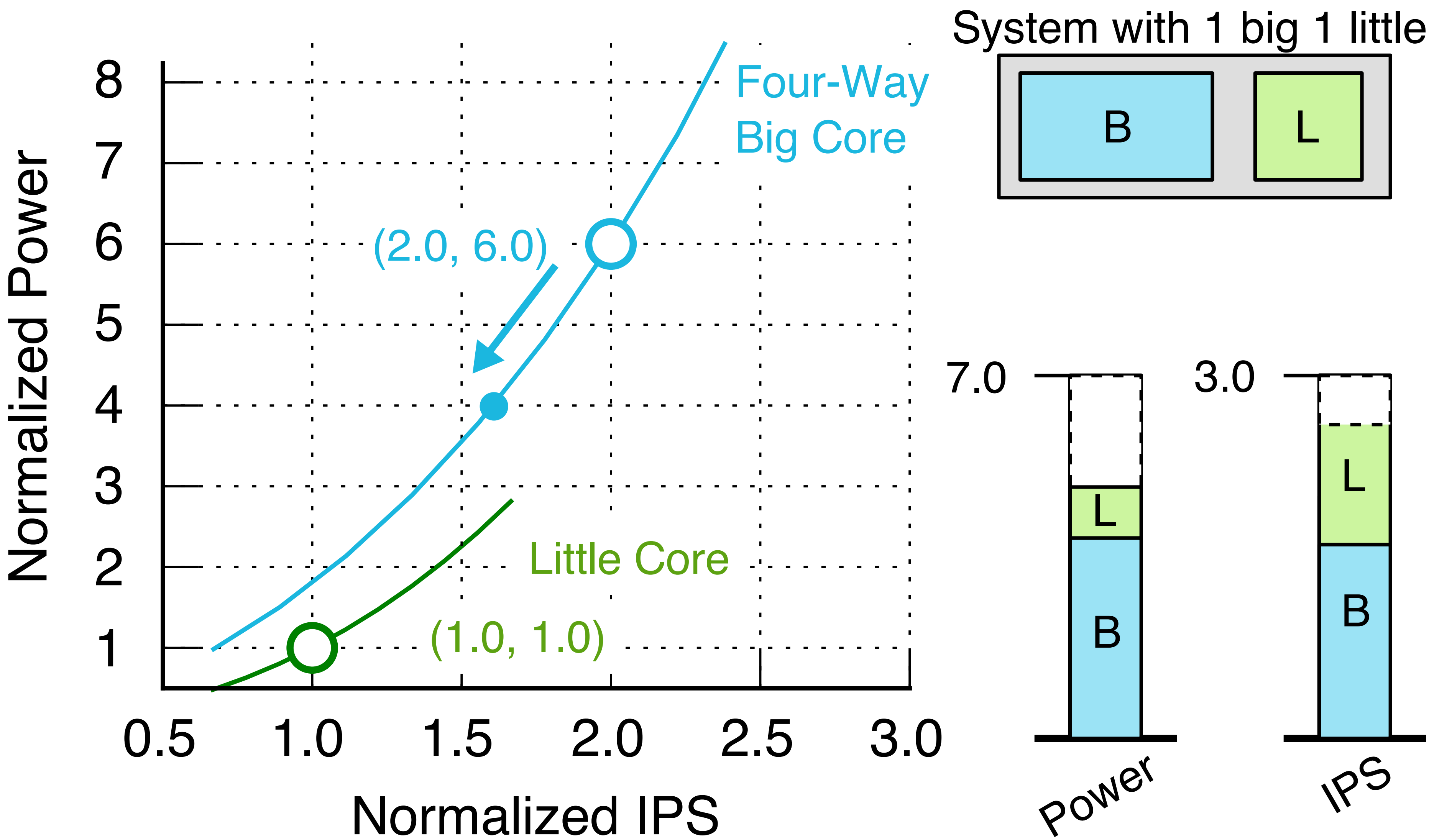


System with 1 big 1 little

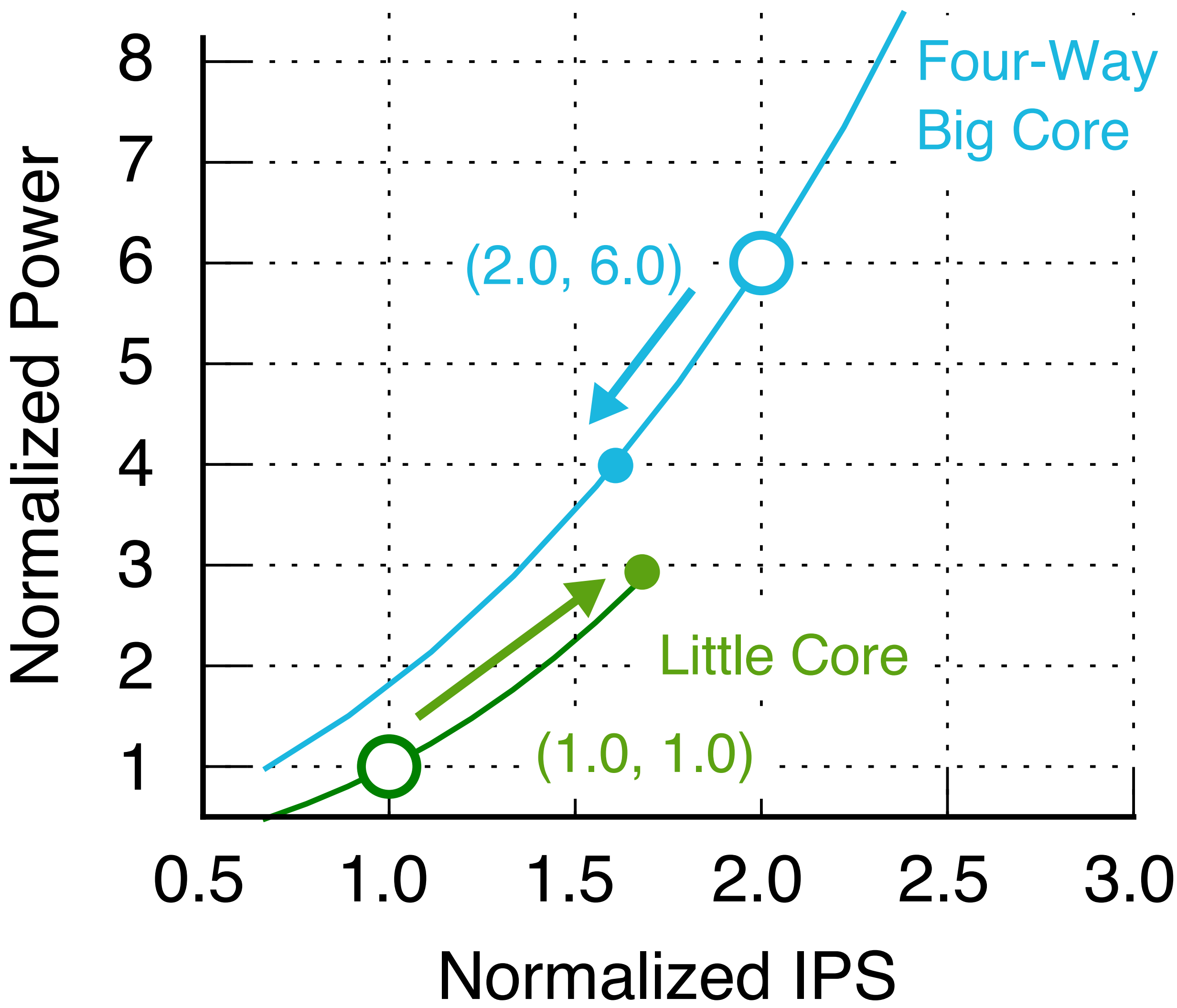


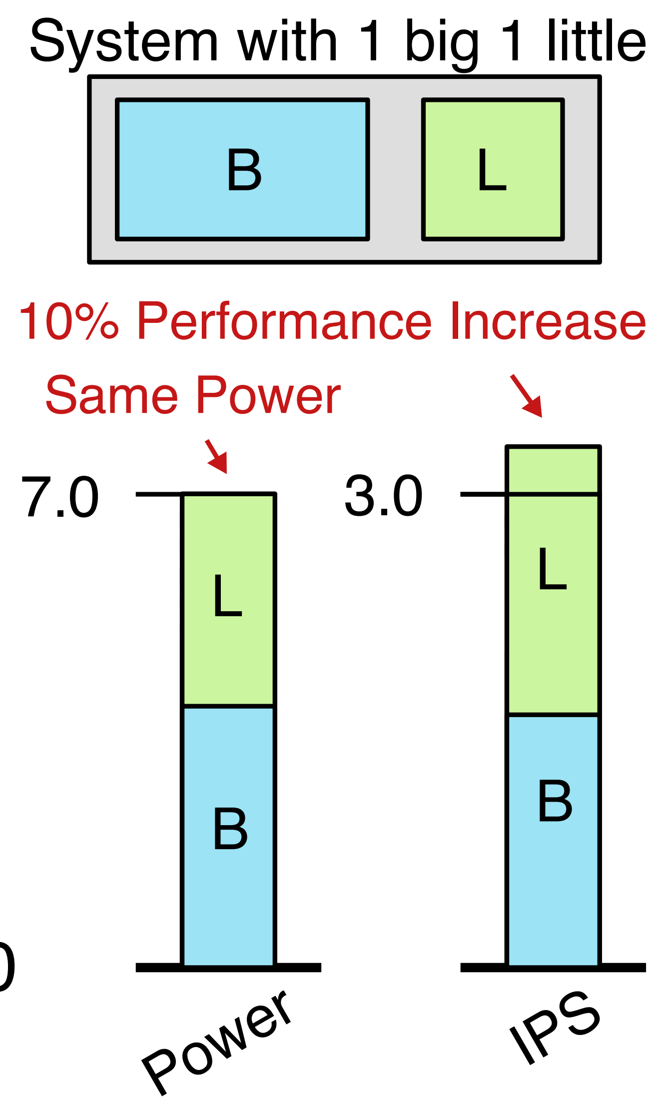


Building Intuition by Exploring a 1 Big 1 Little System

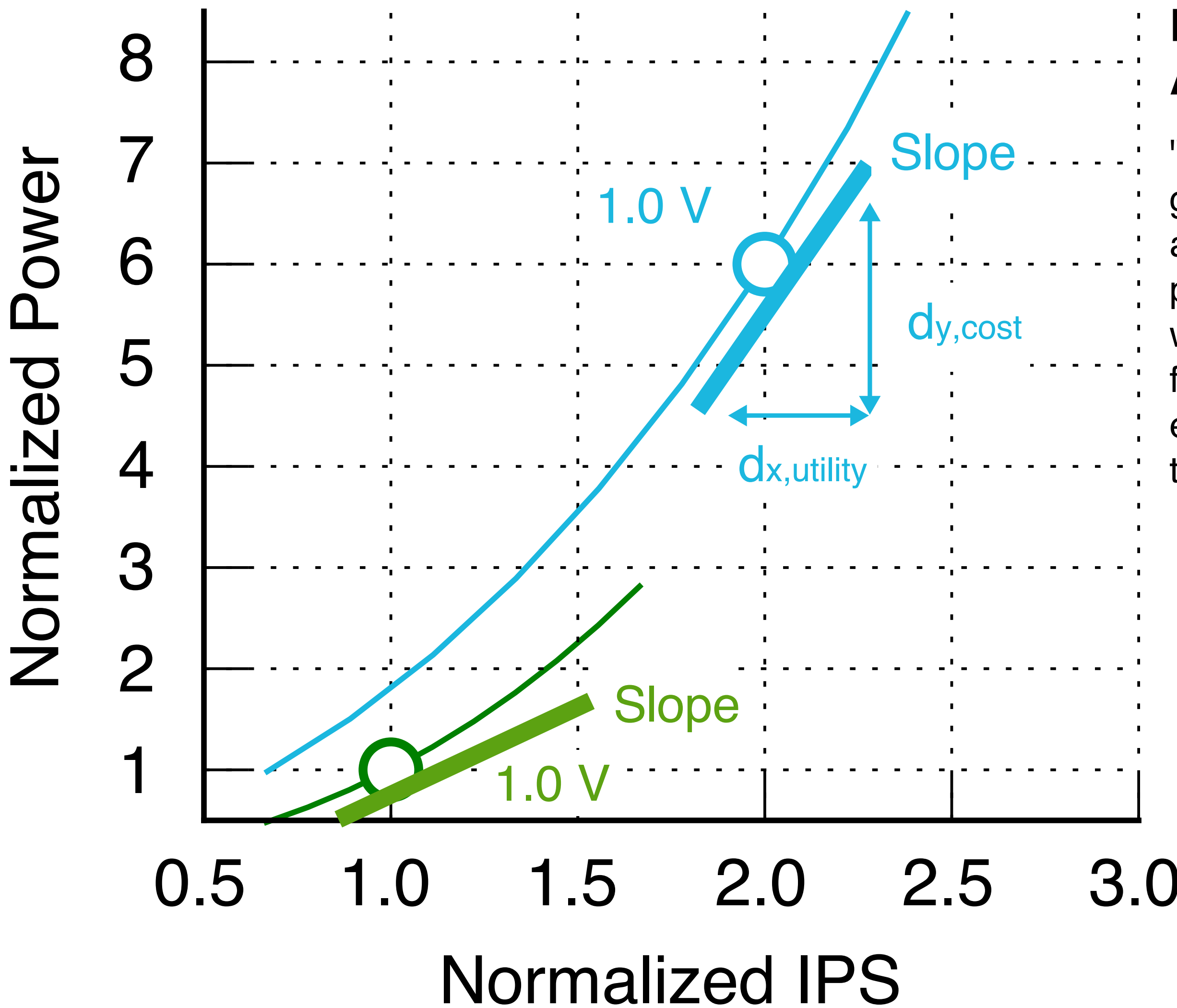


Building Intuition by Exploring a 1 Big 1 Little System





The Law of Equi-Marginal Utility

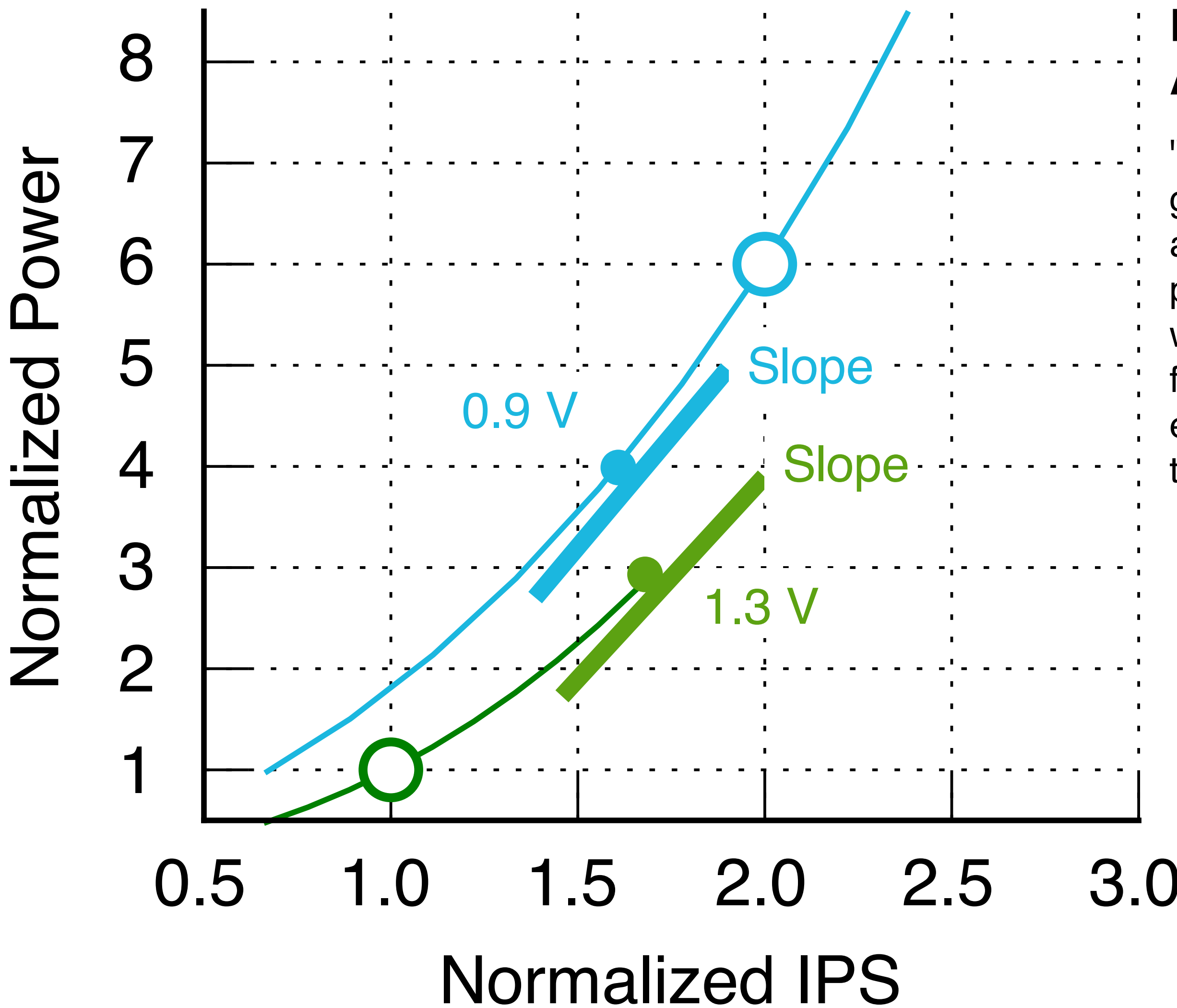


British Economist
Alfred Marshall (1824 - 1924)

"Other things being equal, a consumer gets **maximum satisfaction** when he allocates his **limited income** to the purchase of different goods in such a way that the **Marginal Utility** derived from the last unit of money spent on each item of expenditure tend **to be equal**."

Balance the ratio of utility (IPS) to cost (power)

The Law of Equi-Marginal Utility



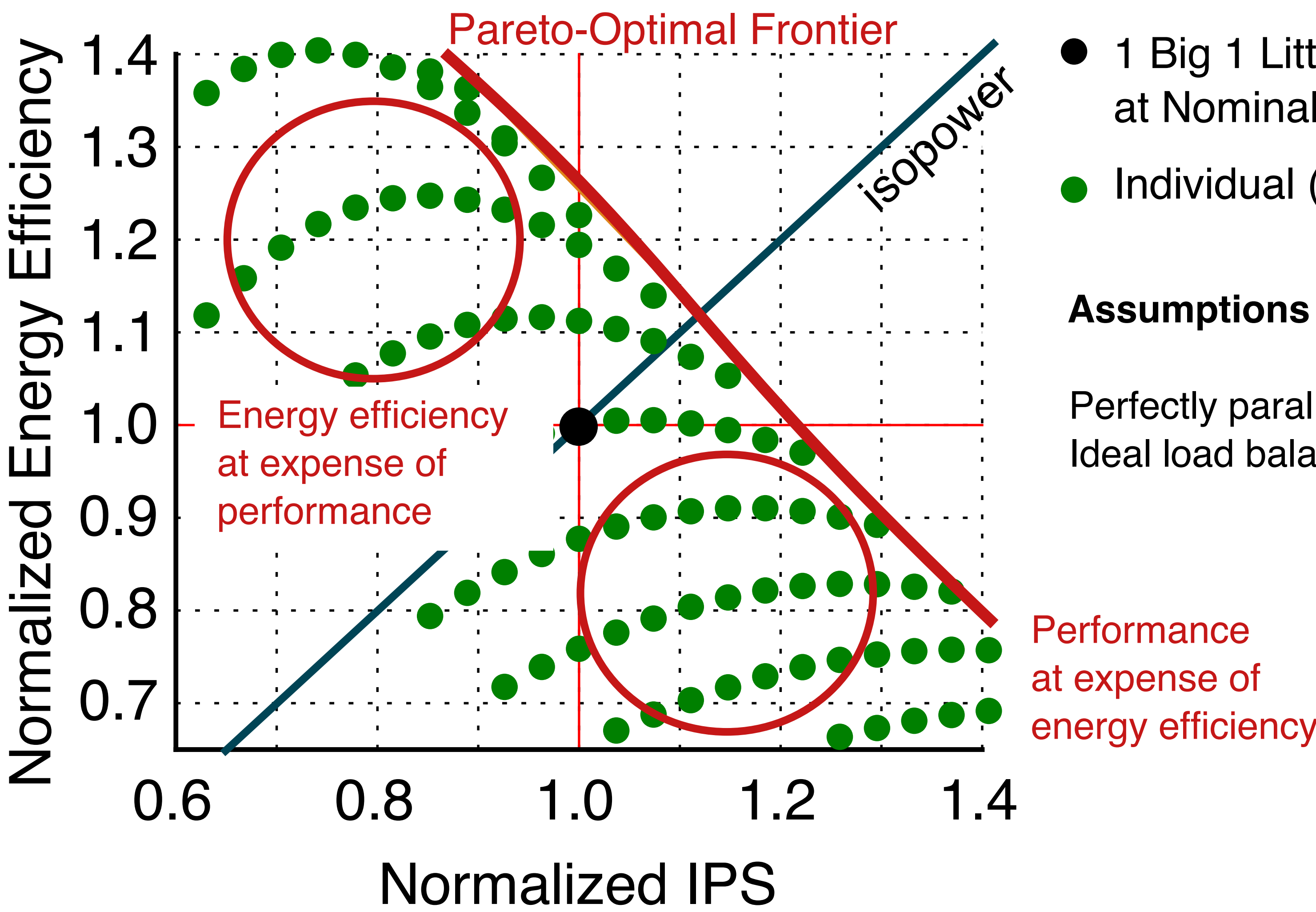
British Economist
Alfred Marshall (1824 - 1924)

"Other things being equal, a consumer gets **maximum satisfaction** when he allocates his **limited income** to the purchase of different goods in such a way that the **Marginal Utility** derived from the last unit of money spent on each item of expenditure tend **to be equal**."

Balance the ratio of utility (IPS) to cost (power)

Arbitrage
"Buy Low, Sell High"

Systematic Approach for Balancing Marginal Utility



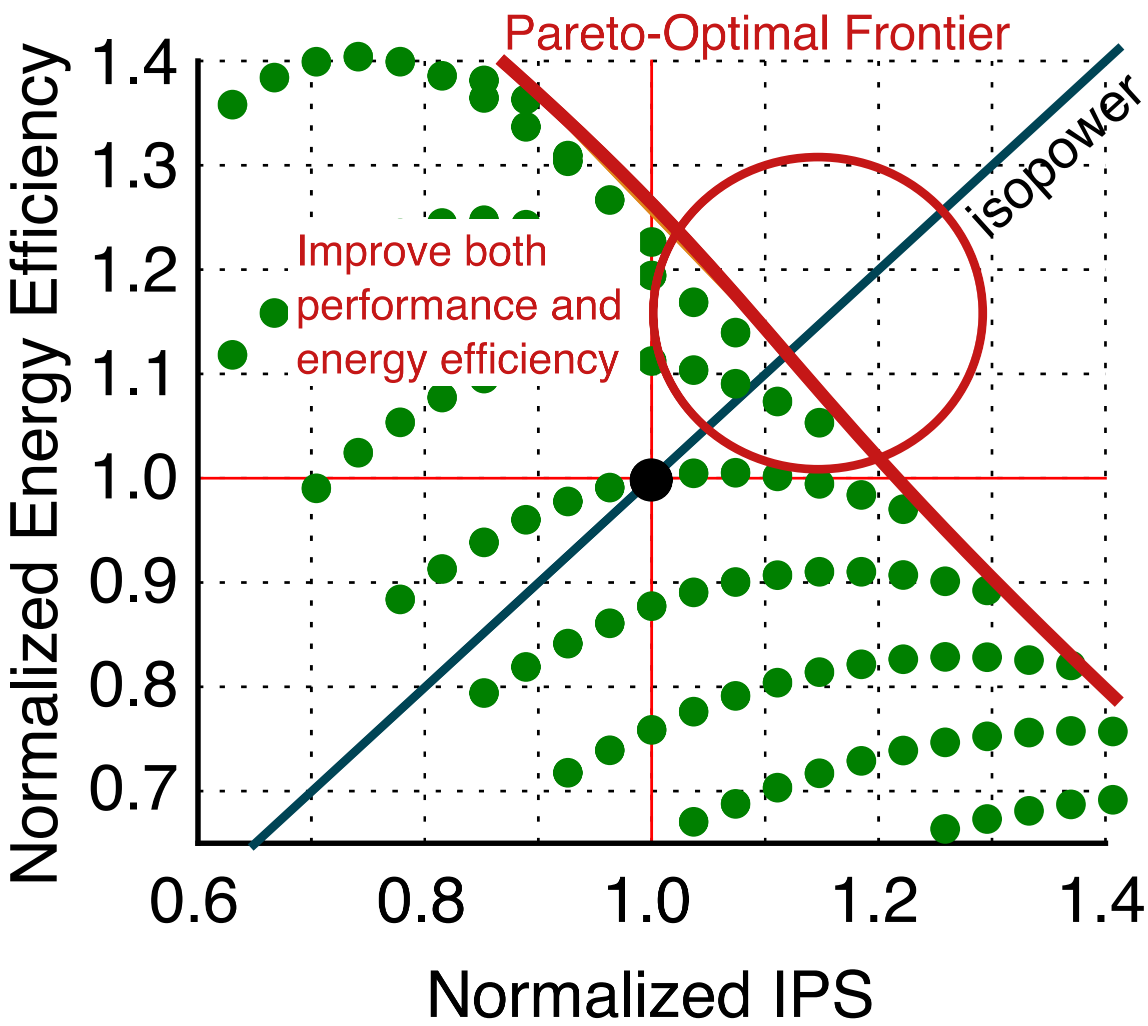
● 1 Big 1 Little System at Nominal voltage

● Individual (V_B, V_L) pair

Assumptions

Perfectly parallel application
Ideal load balancing

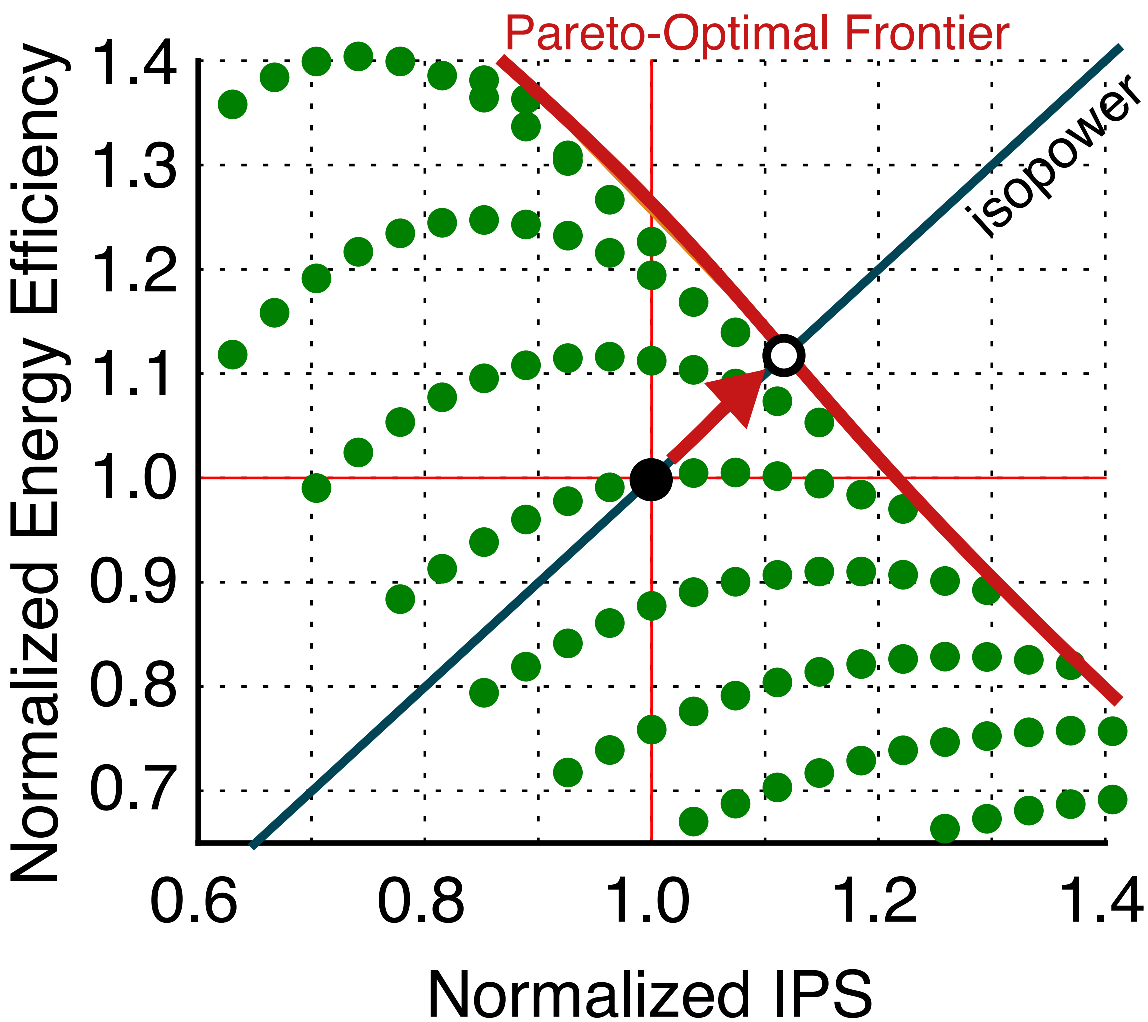
Systematic Approach for Balancing Marginal Utility



Assumptions

Perfectly parallel application
Ideal load balancing

Systematic Approach for Balancing Marginal Utility



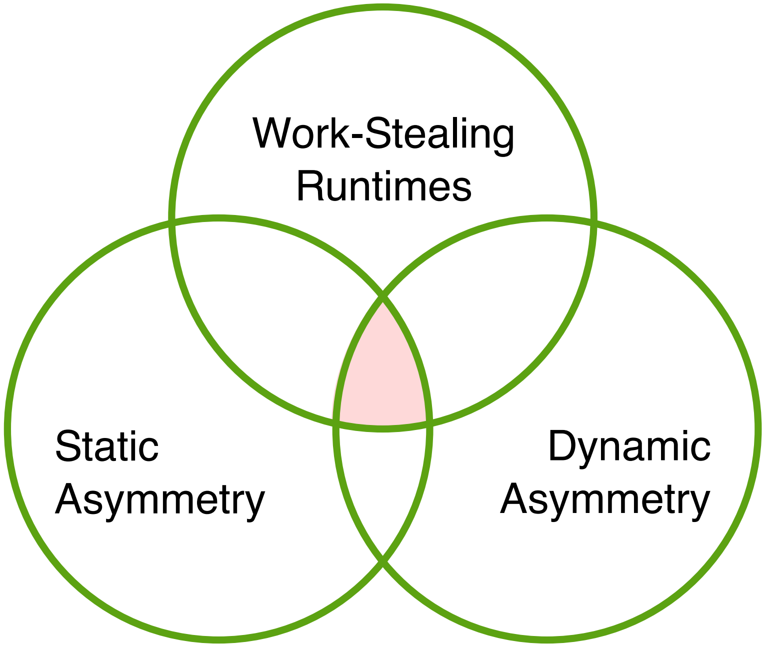
- 1 Big 1 Little System at Nominal voltage
- Individual (V_B, V_L) pair

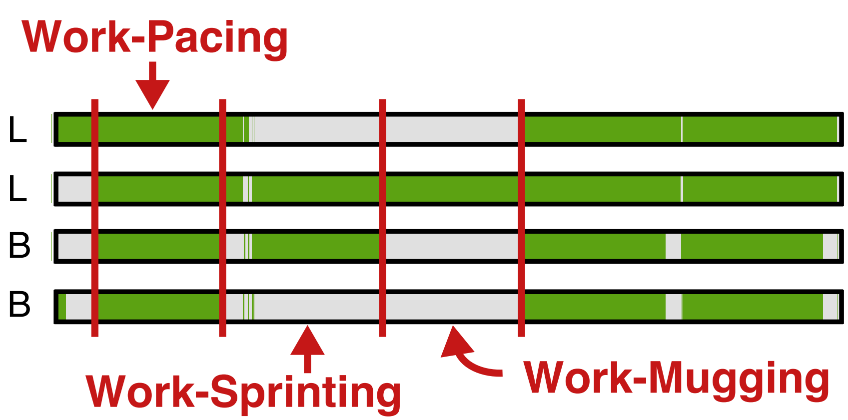
Assumptions

Perfectly parallel application
Ideal load balancing

Marginal Utility-Based Optimization Problem

Constraint: isopower line
Objective: maximize performance
Solved numerically





Talk Outline

Motivation

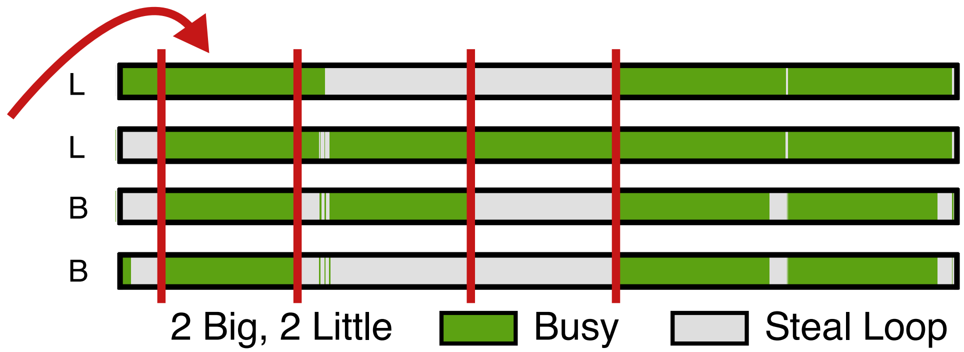
First-Order Modeling

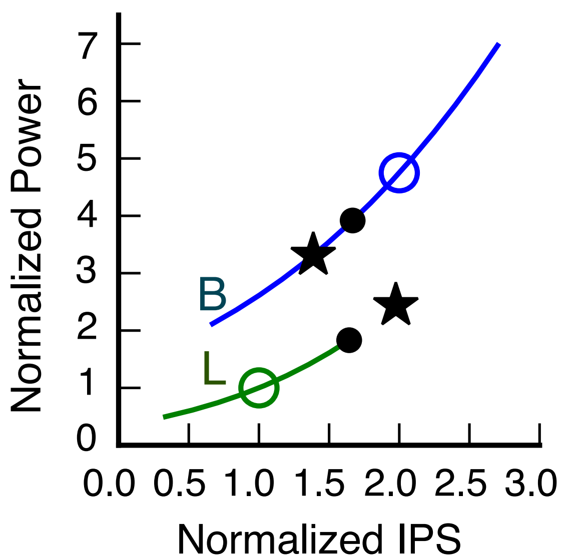
Asymmetry-Aware Work-Stealing Runtimes

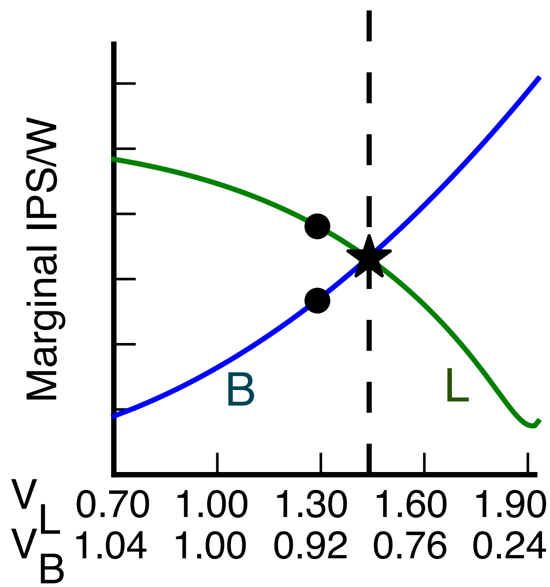
Evaluation

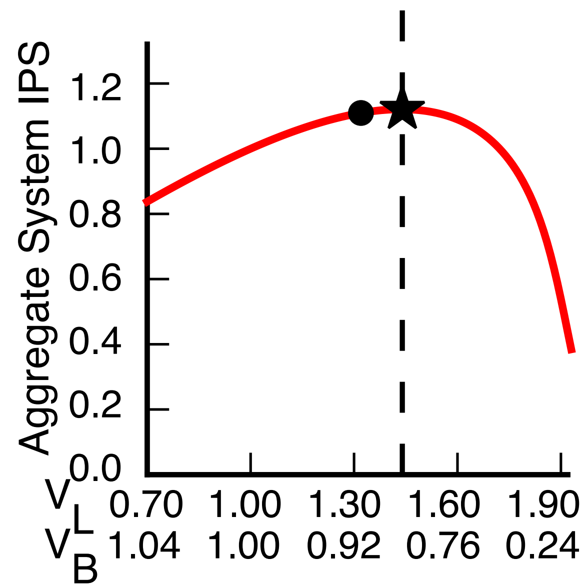
Work-Pacing: Building Intuition

Balance performance/power across cores in the high-parallel (HP) region



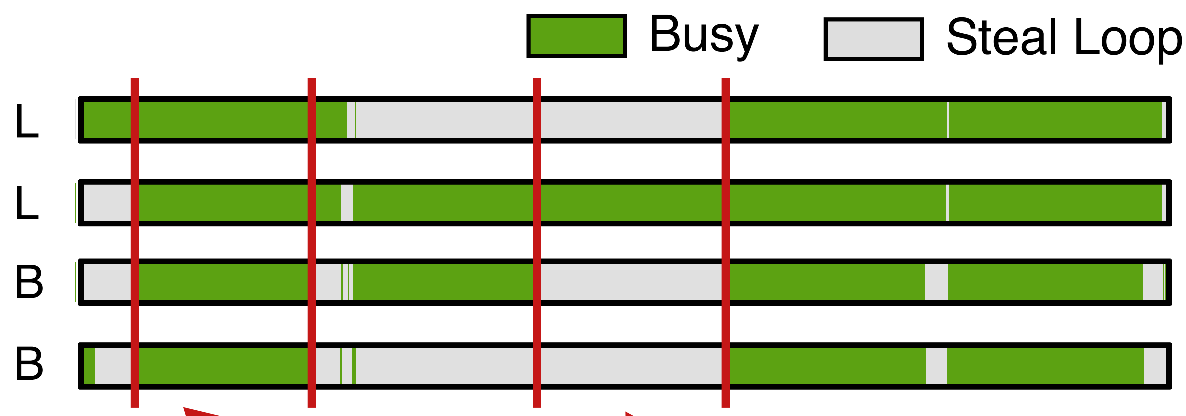






System with both big cores active and both little cores active

Work-Pacing, Work-Sprinting, and Work-Mugging



Work-Pacing

Balance performance/power across cores in the high-parallel (HP) region

Work-Sprinting

Rest cores in the steal loop to the lowest voltage

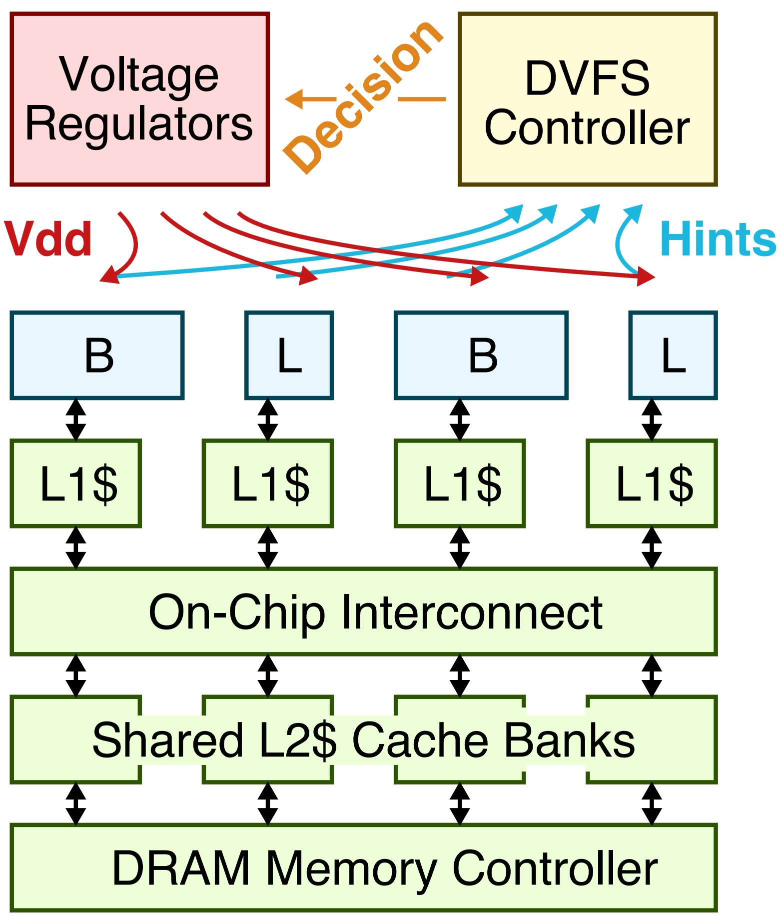
With additional power slack, balance performance/power across busy cores in the low-parallel (LP) region

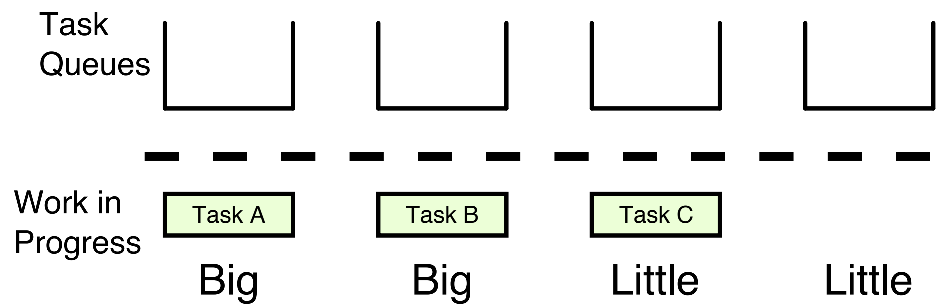
Work-Mugging

Move work from slow little cores to fast big cores in the low-parallel (LP) region

Work-Pacing and Work-Sprinting Mechanisms

Which cores are stealing? Big or little?

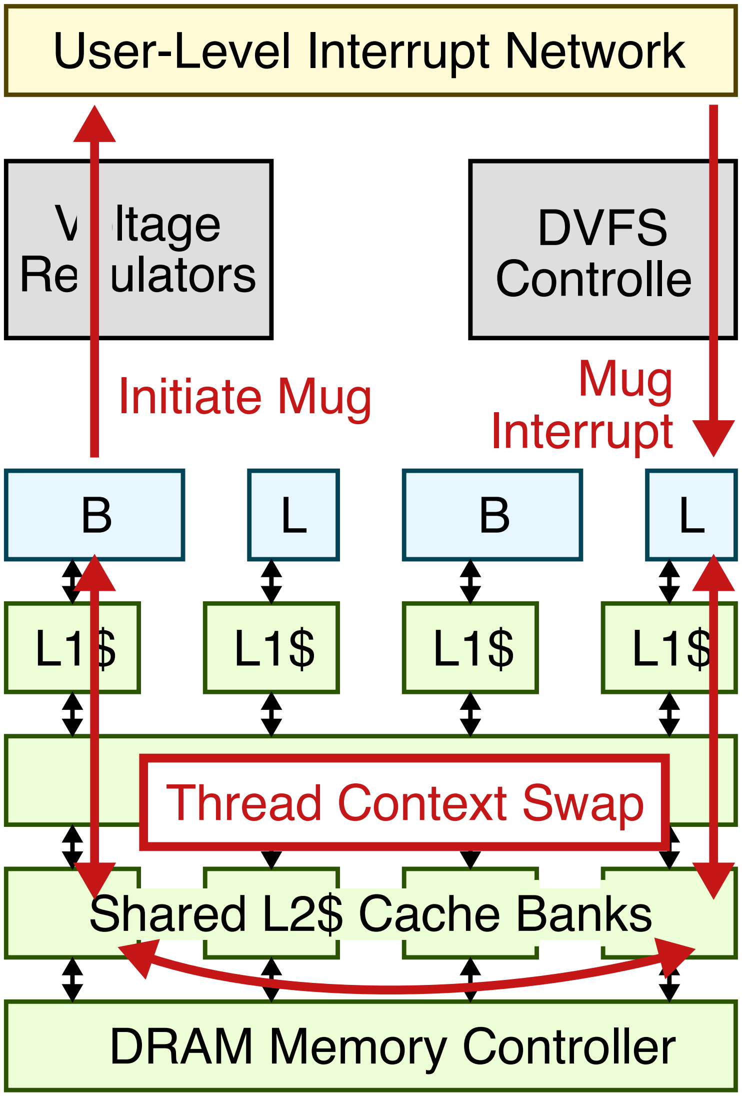




Activity Pattern					Voltages		
B	B	L	L		V_B	V_L	Stealing
A	A	A	A	→	0.91V	1.30V	
A	A	A	S	→	0.98V	1.30V	0.70V
A	A	S	S	→	1.03V	1.30V	0.70V
A	S	A	A	→	1.04V	1.30V	0.70V
A	S	A	S	→	1.13V	1.30V	0.70V
A	S	S	S	→	1.21V	0.70V	0.70V
S	S	A	A	→	0.70V	1.30V	0.70V
S	S	A	S	→	0.70V	1.30V	0.70V
S	S	S	S	→	0.70V	0.70V	0.70V

A = Active, **S** = Stealing

Work-Mugging Mechanisms



Mug Instruction

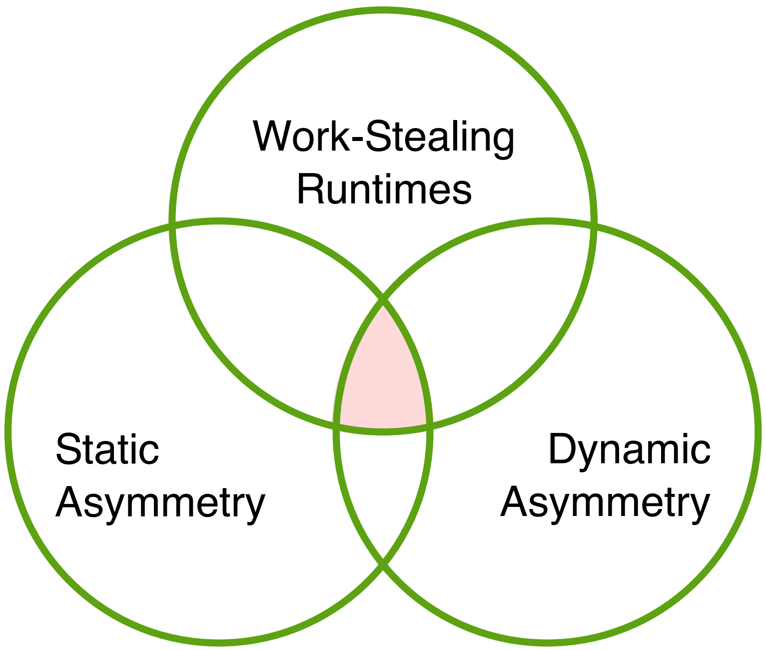
- Thread ID to mug
- Address of thread-swapping handler

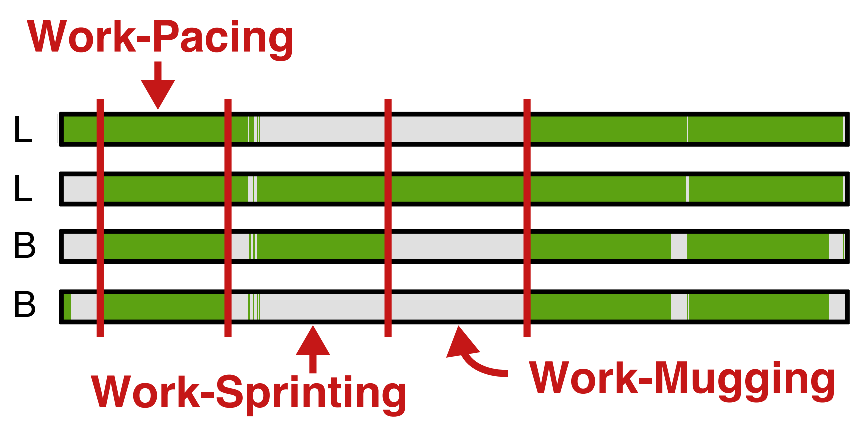
User-Level Interrupt Network

- Simple, low-bandwidth inter-core network
- Latency on order of 20 cycles

Thread Context Swap

- Threads store architectural state to separate locations in shared memory
- Both threads sync
- Threads load architectural state from other location





Talk Outline

Motivation

First-Order Modeling

Asymmetry-Aware Work-Stealing Runtimes

Evaluation

Evaluation Methodology: Modeling

Work-Stealing Runtime

- State-of-the-art Intel TBB-inspired work-stealing scheduler
- Chase-Lev task queues with occupancy-based victim selection
- Instrumented with activity hints

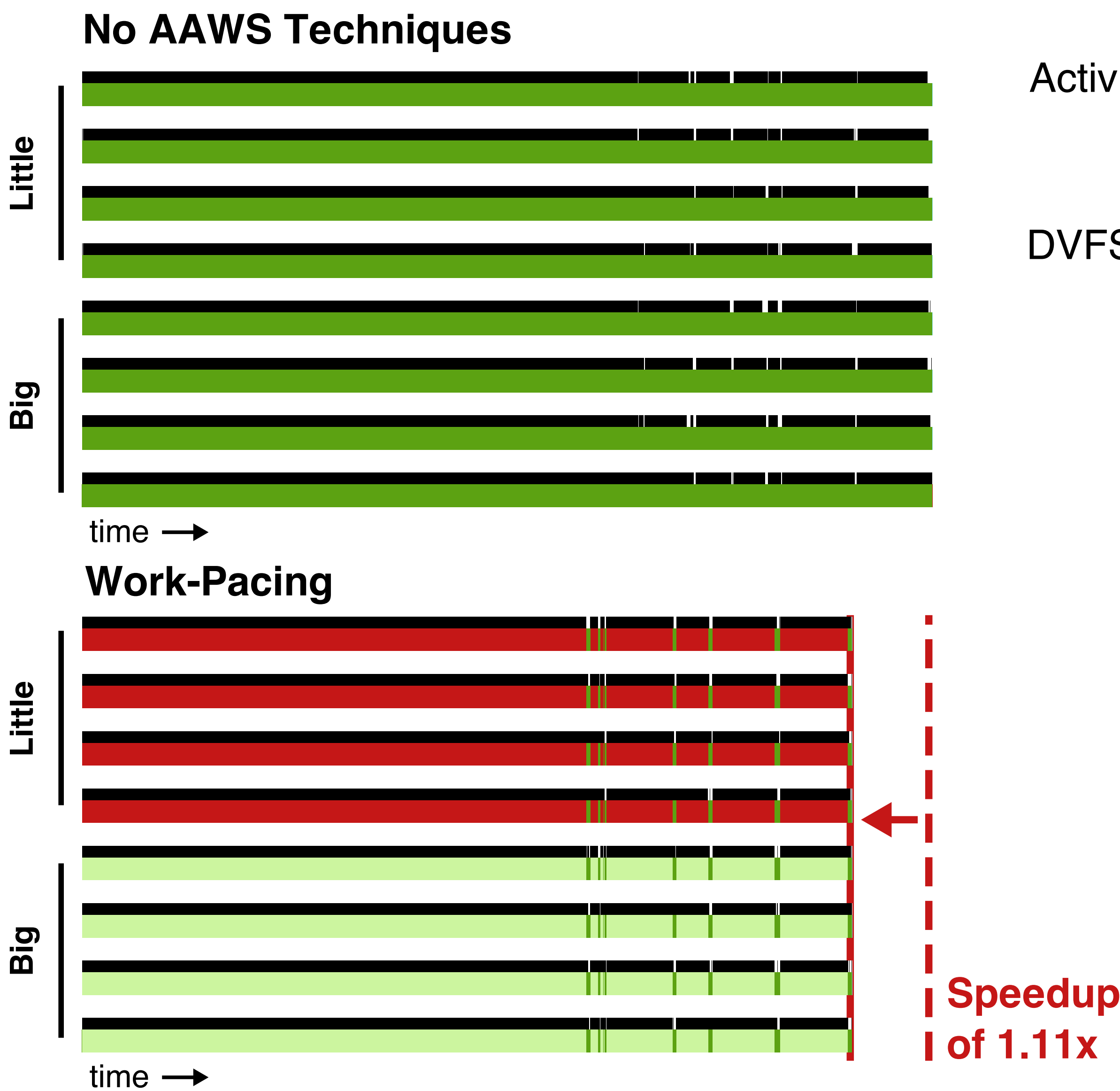
Cycle-Level Modeling

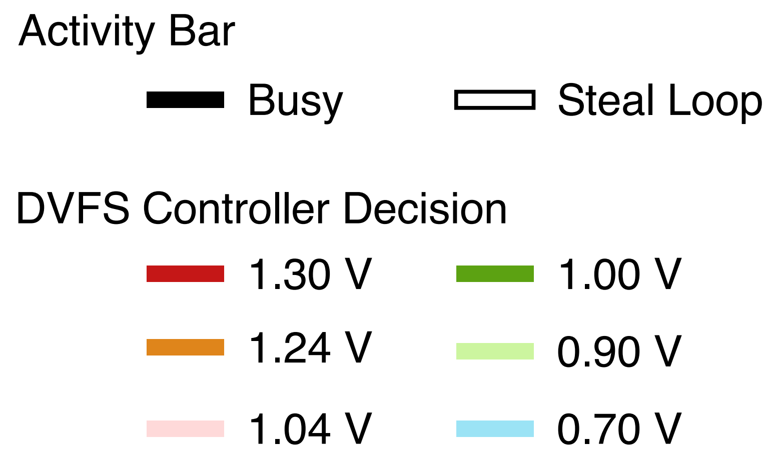
- Heterogeneous system modeled in gem5 cycle-approximate simulator
- Support for scaling per-core frequencies + central DVFS Controller

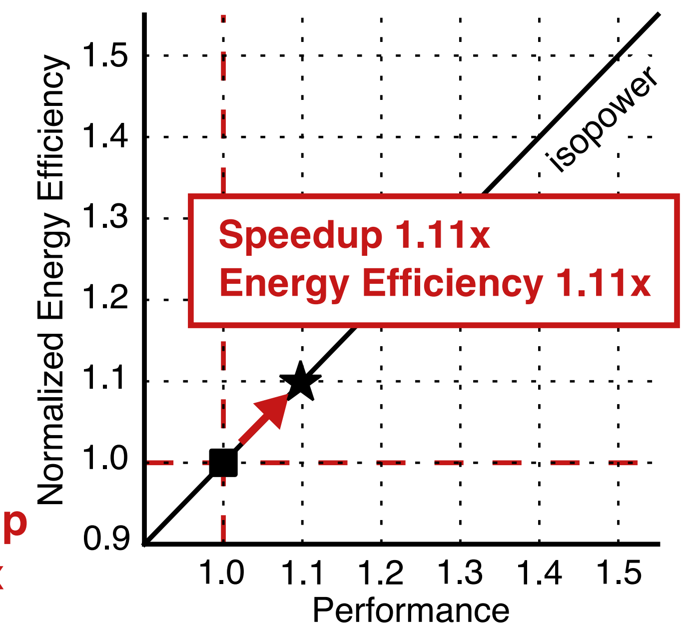
Energy Modeling

- Event-based energy modeling based on detailed RTL/gate-level sims (Synopsys ASIC toolflow, TSMC LP, 65 nm 1.0 V)
- Carefully selected subset of McPAT results tuned to our µarchitecture

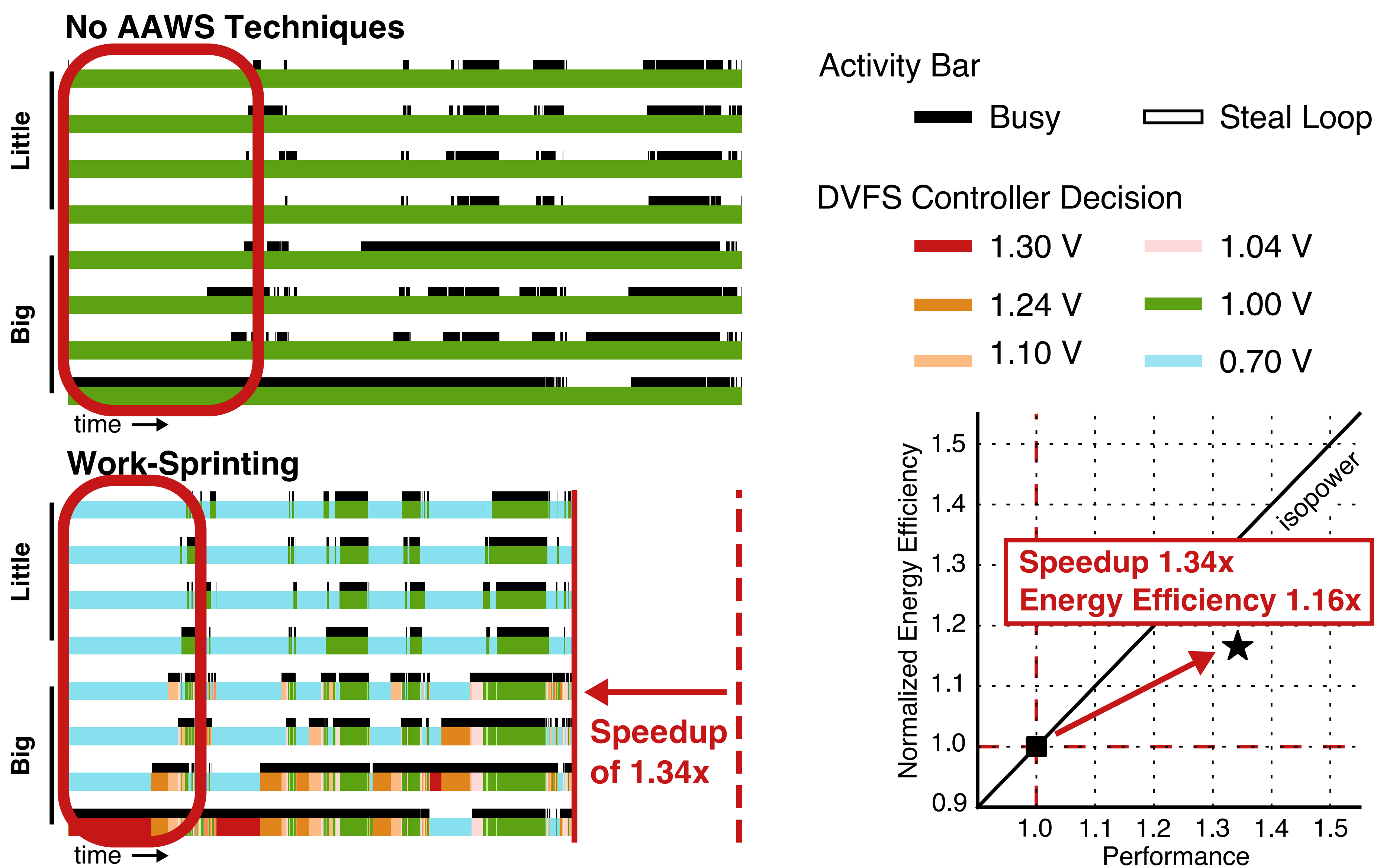
Work-Pacing in *cilk-sort*



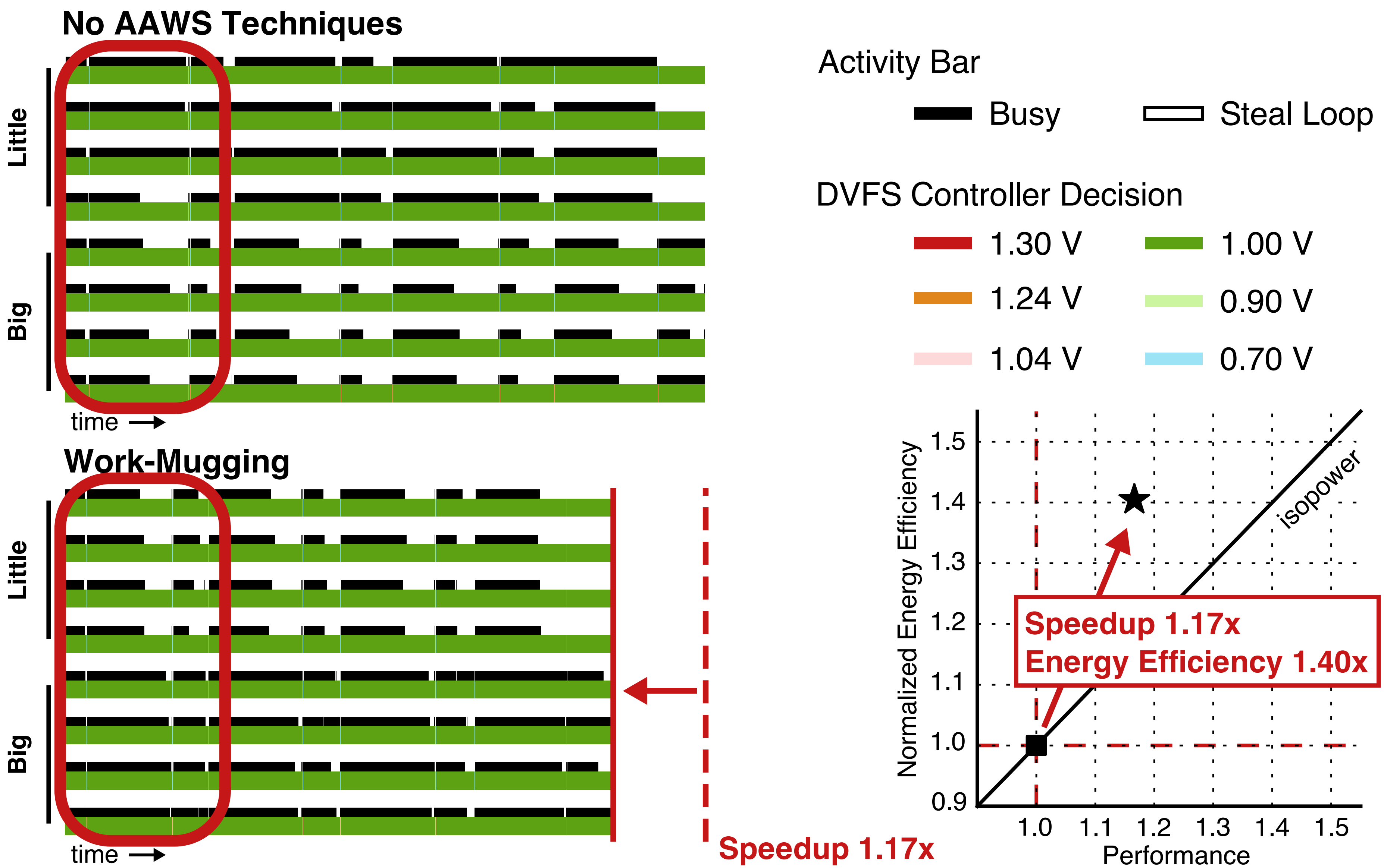




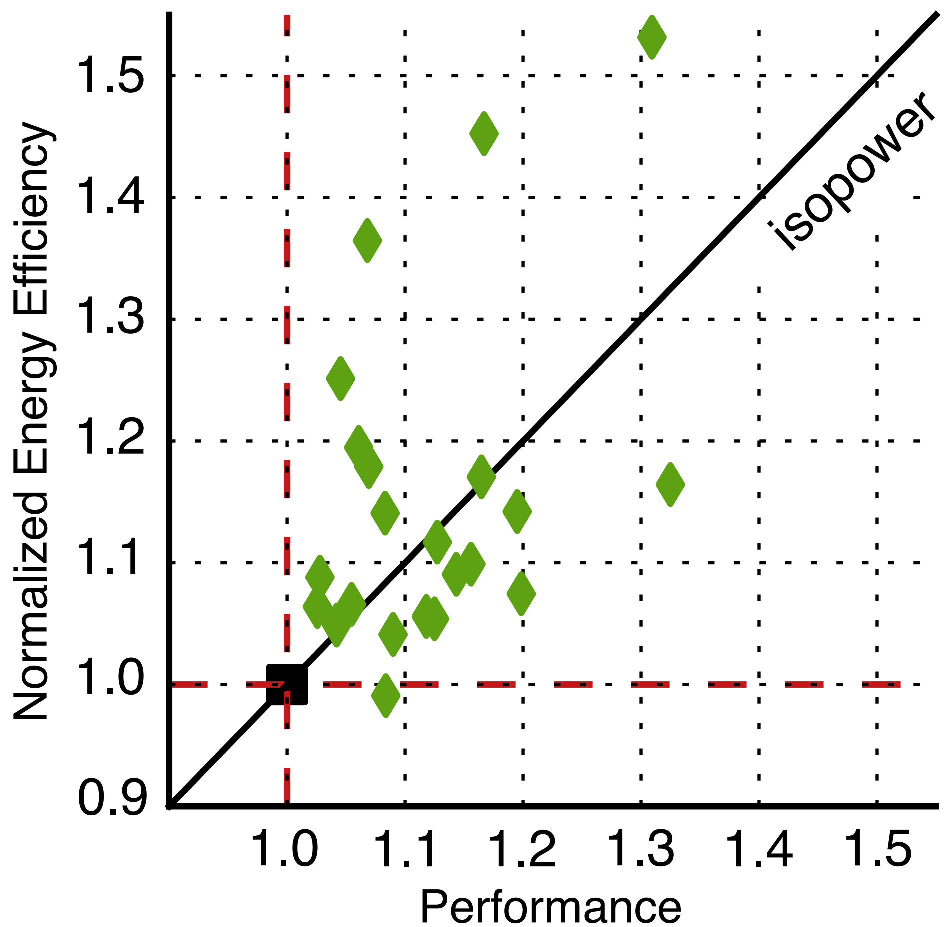
Work-Sprinting in *quicksort*



Work-Mugging in *radix sort*



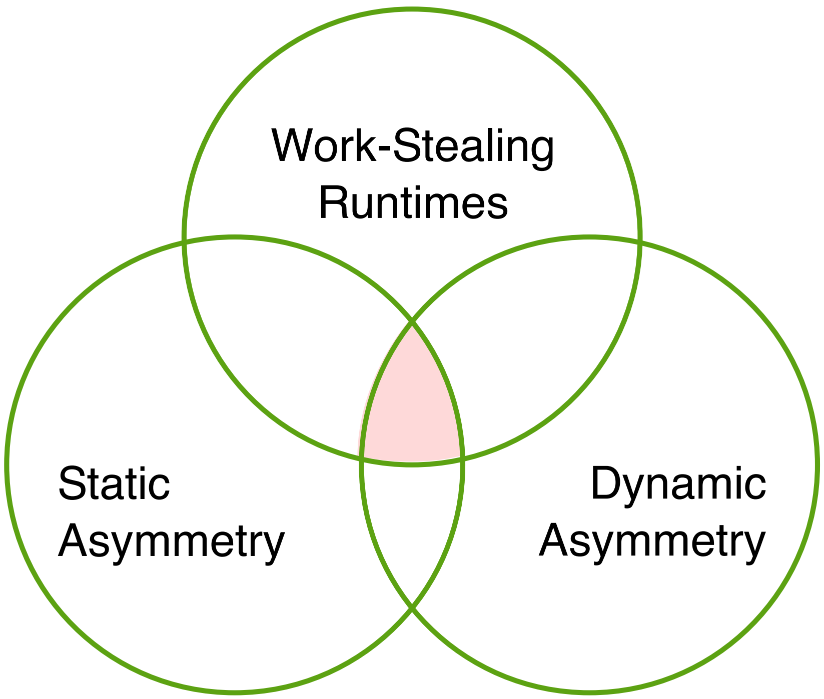
Evaluation of Complete AAWS Runtime



Application Kernels

- pbbs-bfs
- pbbs-quicksort
- pbbs-samplesort
- pbbs-dictionary
- pbbs-convex-hull
- pbbs-radix-sort
- pbbs-knn
- pbbs-max-independent-set
- pbbs-nbody
- pbbs-remove-duplicates
- pbbs-suffix-array
- pbbs-spanning-tree
- cilk-cholesky
- cilk-cilksort
- cilk-heat
- cilk-knapsack
- cilk-matrix-multiply
- parsec-blackscholes
- unbalanced-tree-search

Performance	Median: 1.10 x	Max: 1.32 x
Energy Efficiency	Median: 1.11 x	Max: 1.53 x



Take-Away Point

Holistically combining

- work-stealing runtimes
- static asymmetry
- dynamic asymmetry

through the use of

- work-pacing
- work-sprinting
- work-mugging

can improve both performance and energy efficiency in future multicore systems

This work was partially supported by the NSF, AFOSR, and donations from Intel and Synopsys