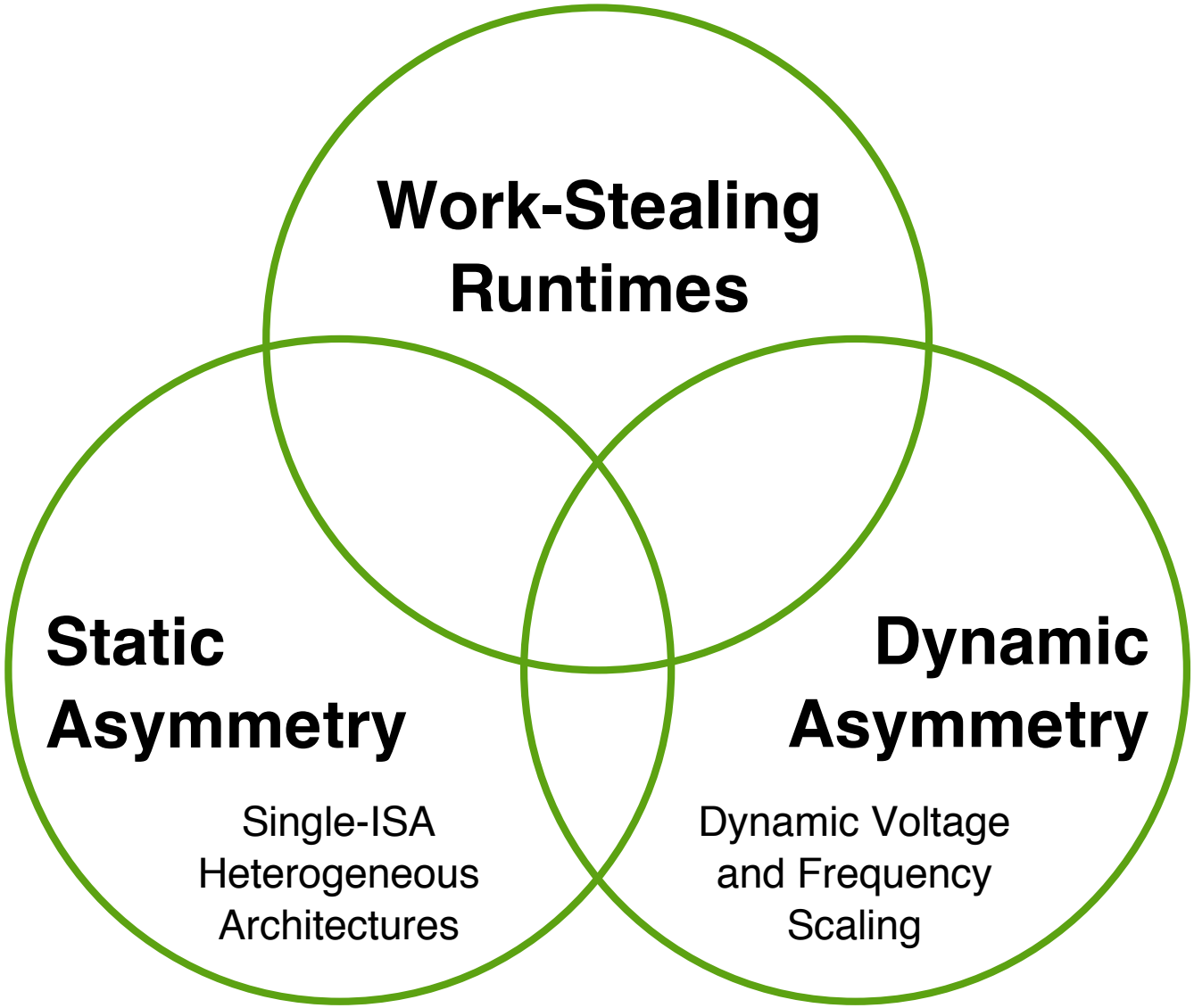
Asymmetry-Aware Work-Stealing Runtimes

Christopher Torng, Moyang Wang, and
Christopher Batten

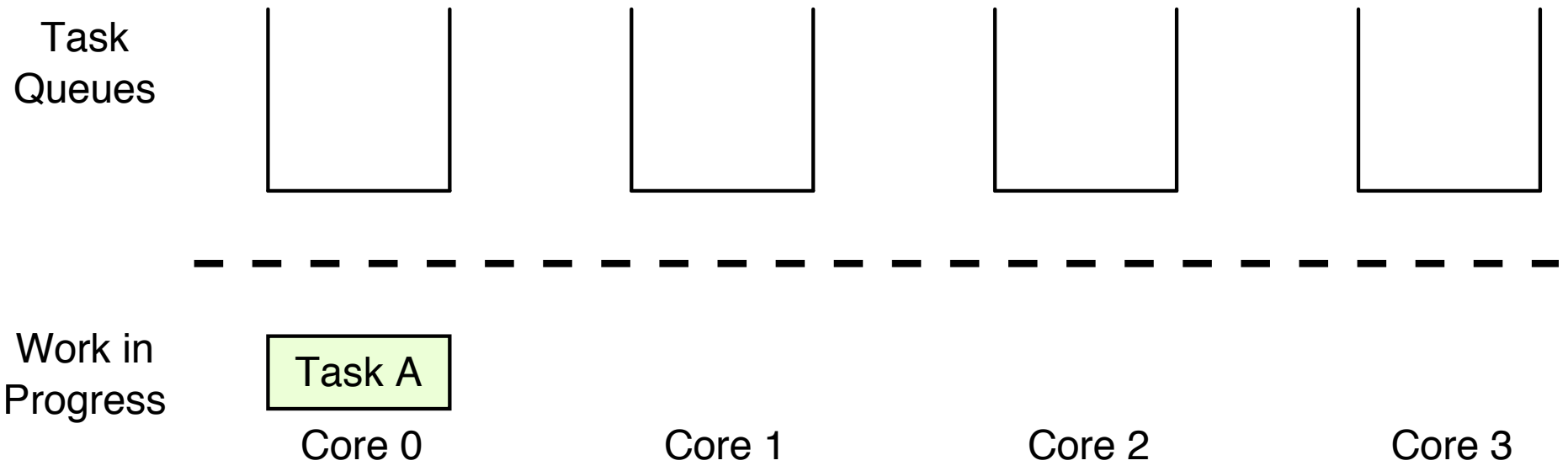
School of Electrical and Computer Engineering
Cornell University

Cornell IAP Workshop
October 2015

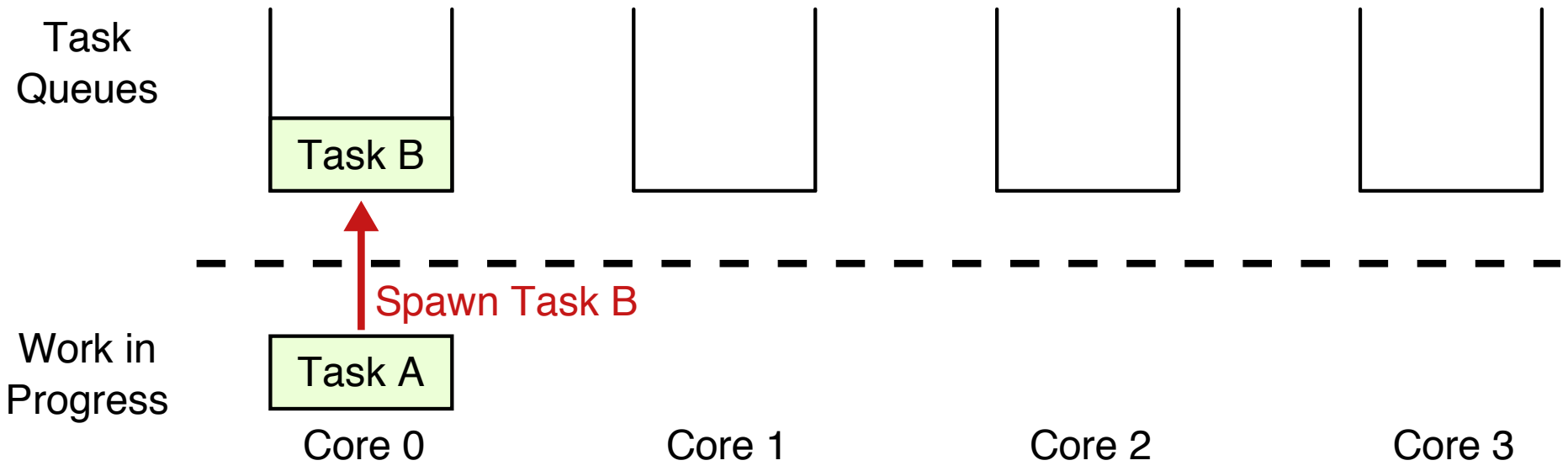
Asymmetry-Aware Work-Stealing Runtimes



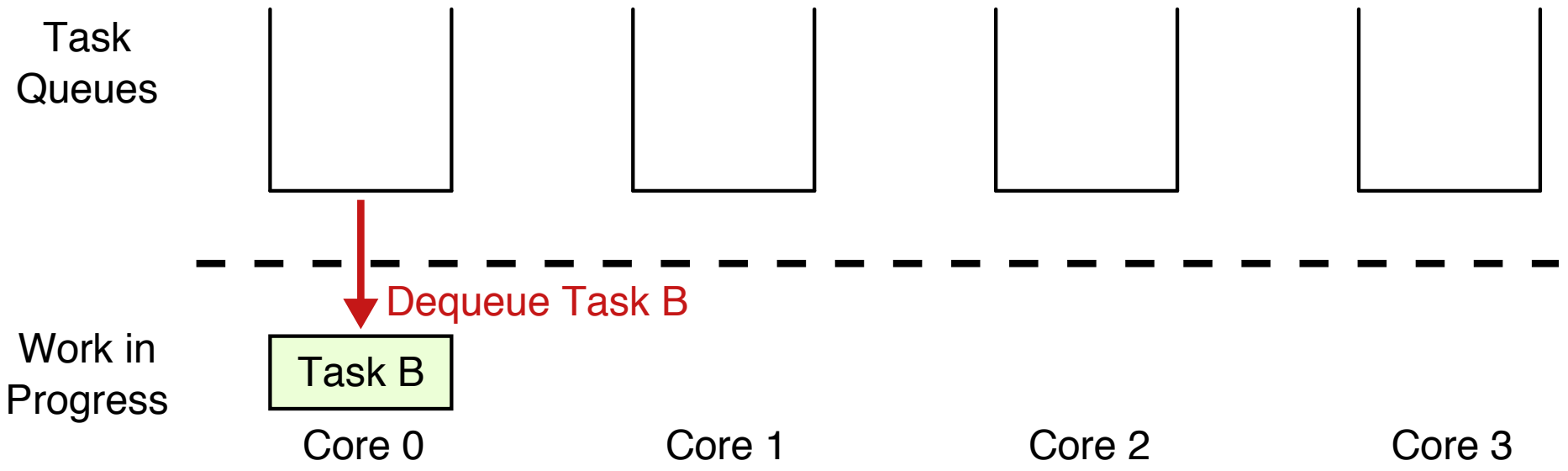
Work-Stealing Runtimes



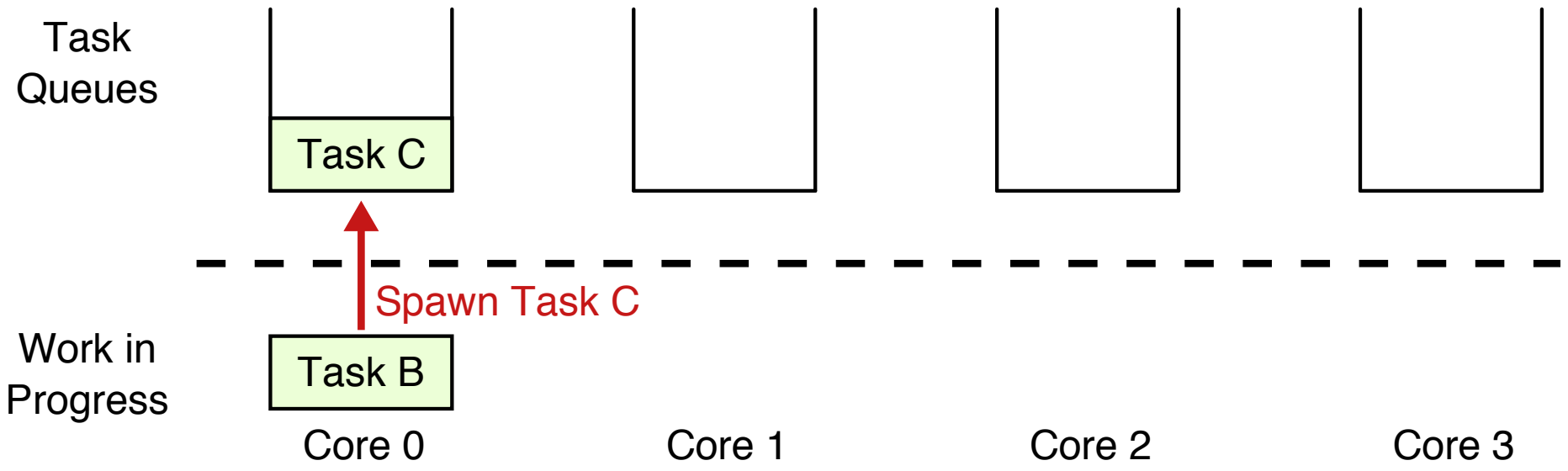
Work-Stealing Runtimes



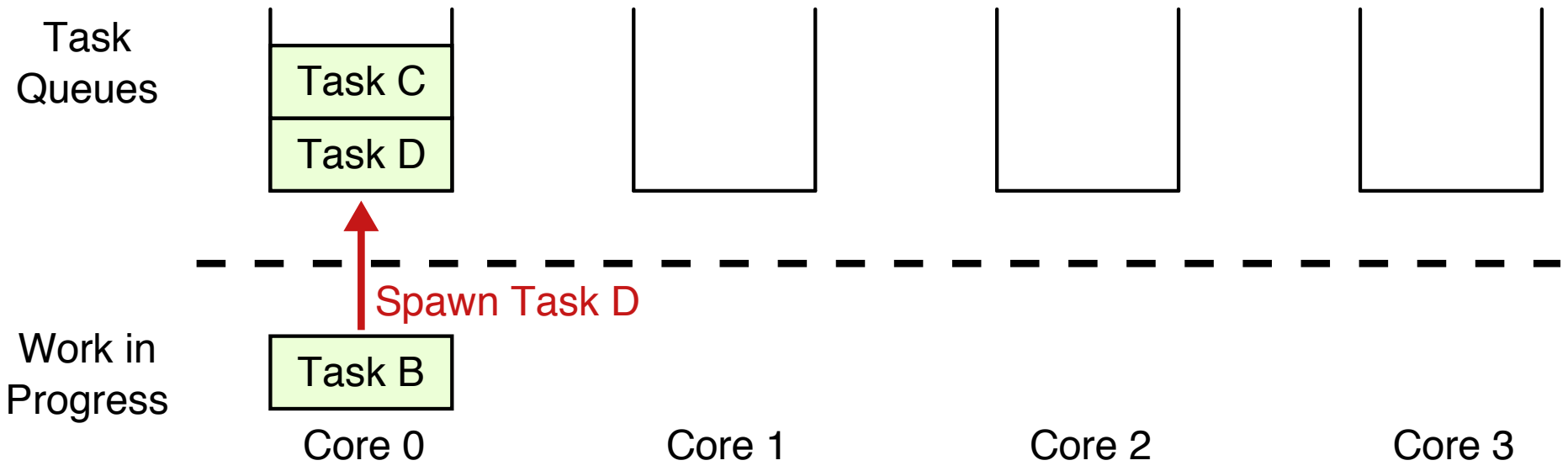
Work-Stealing Runtimes



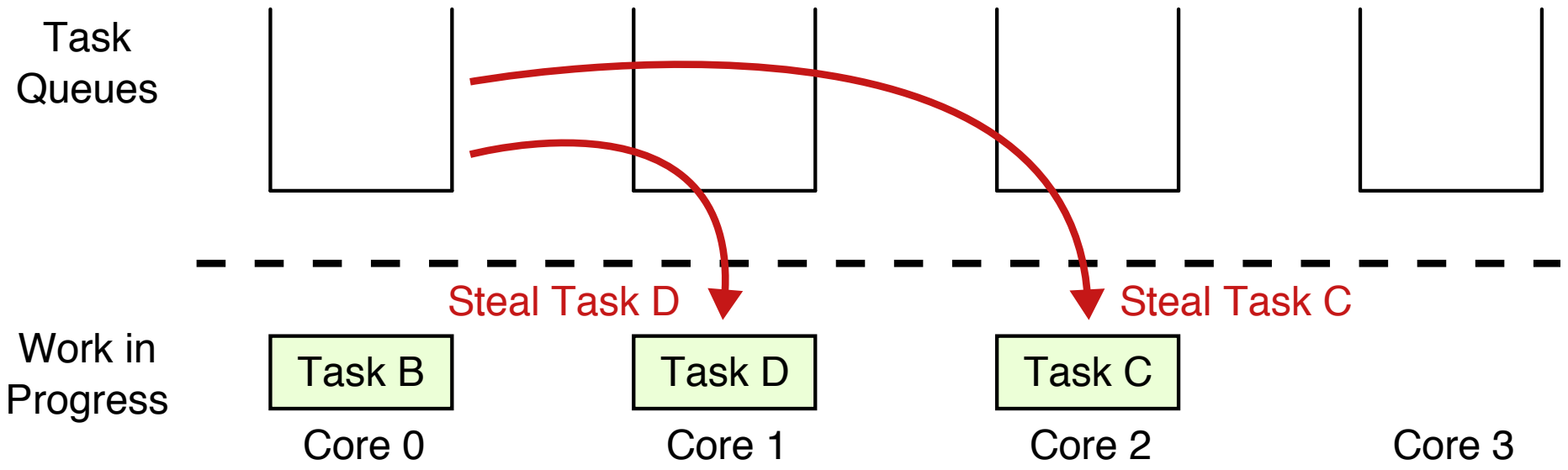
Work-Stealing Runtimes



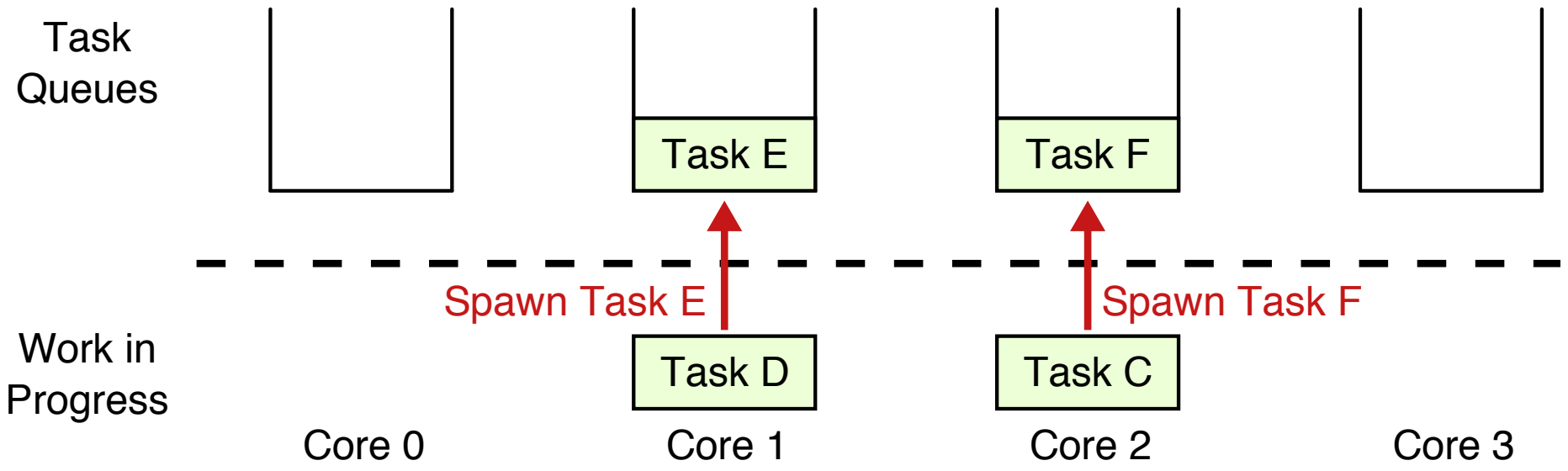
Work-Stealing Runtimes



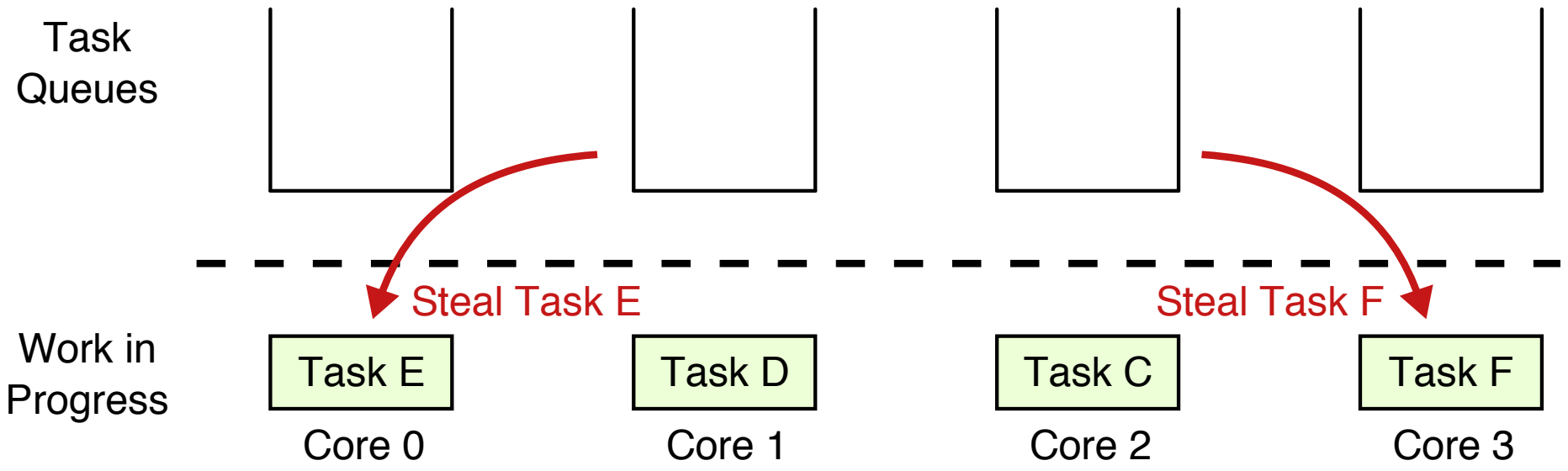
Work-Stealing Runtimes



Work-Stealing Runtimes

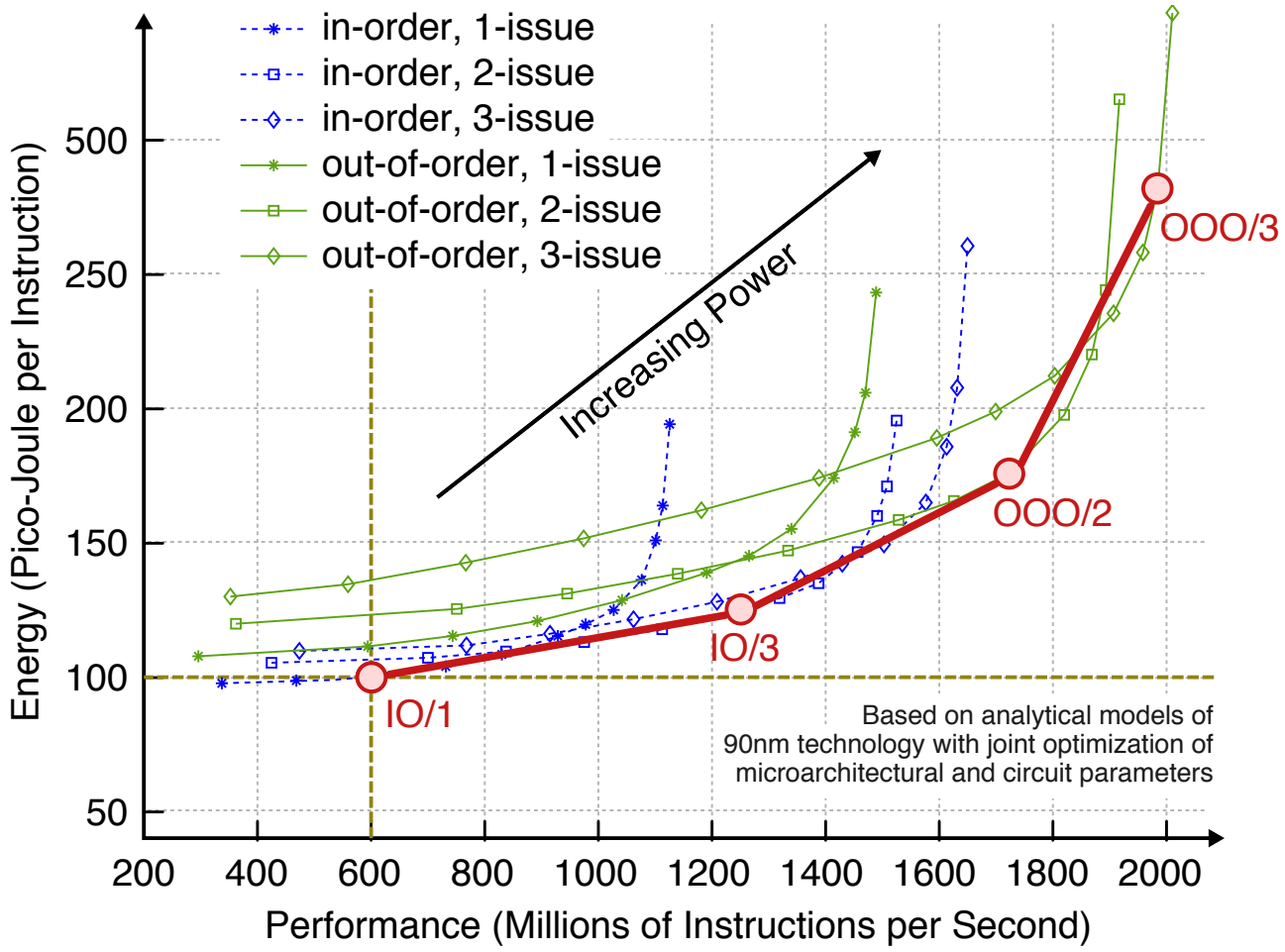


Work-Stealing Runtimes

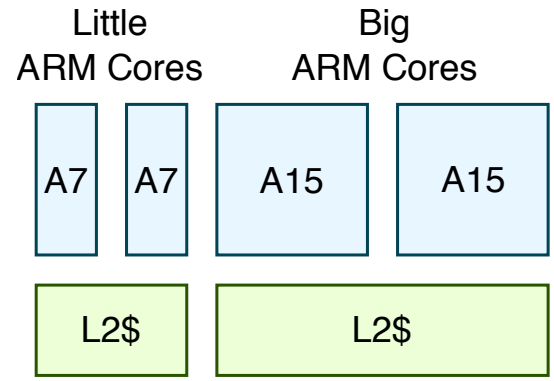


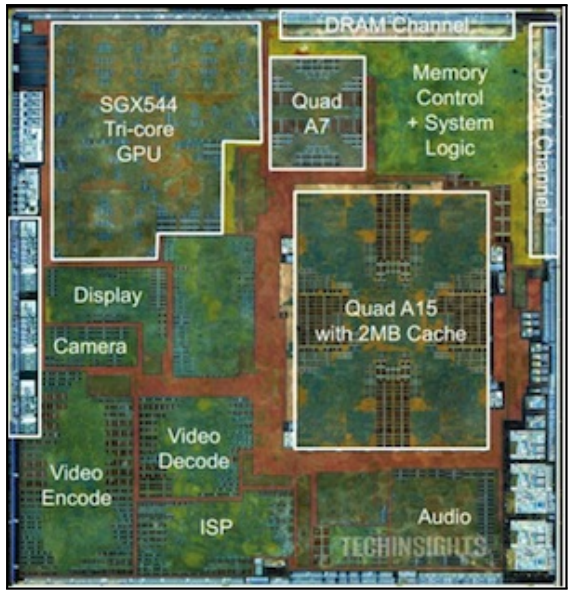
- Work stealing has good performance, space requirements, and communication overheads in both theory and practice
- Supported in many popular concurrency plaforms including: Intel's Cilk Plus, Intel's C++ TBB, Microsoft's .NET Task Parallel Library, Java's Fork/Join Framework, and OpenMP

Static Asymmetry: Heterogeneous Multicore Systems



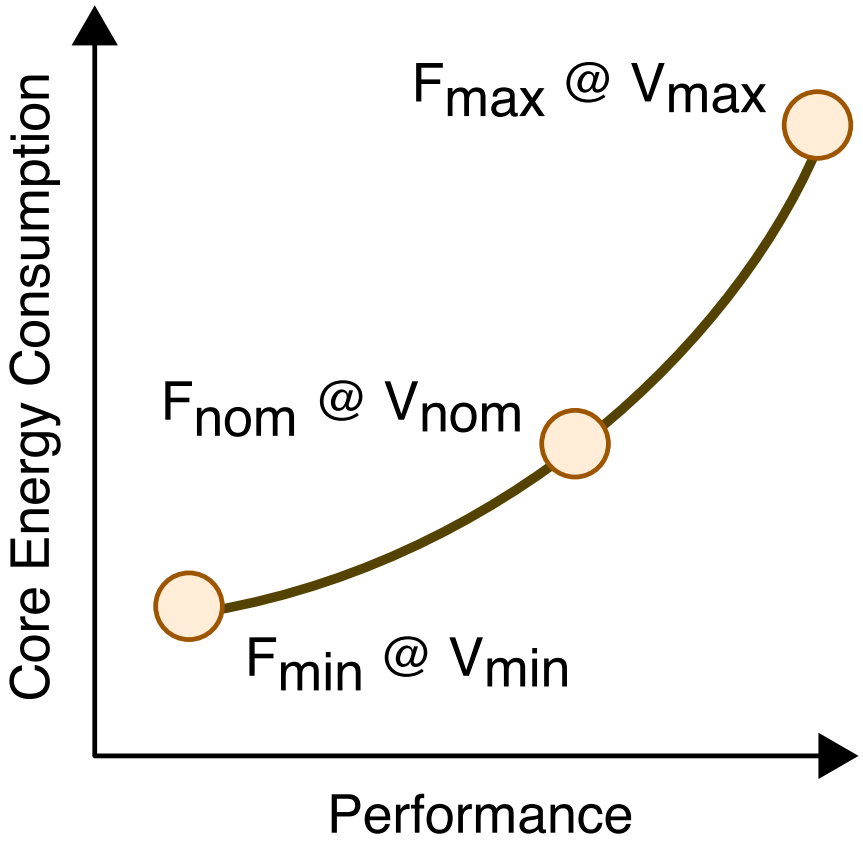
Adpated from O. Azizi et al. "Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Approach," ISCA, 2010.

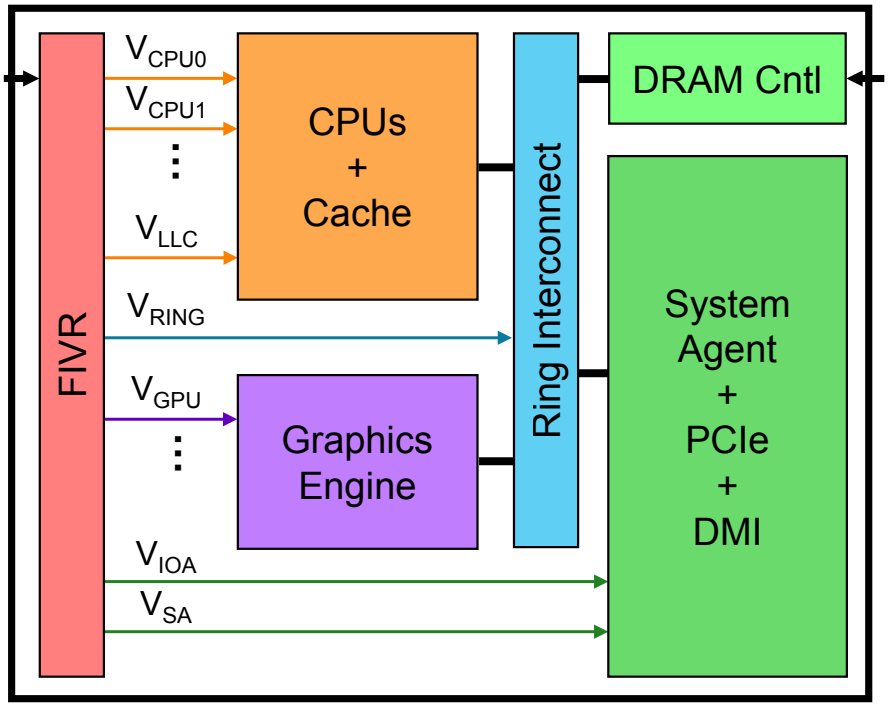




Samsung Exynos Octa Mobile Processor

Dynamic Asymmetry: Voltage/Frequency Scaling



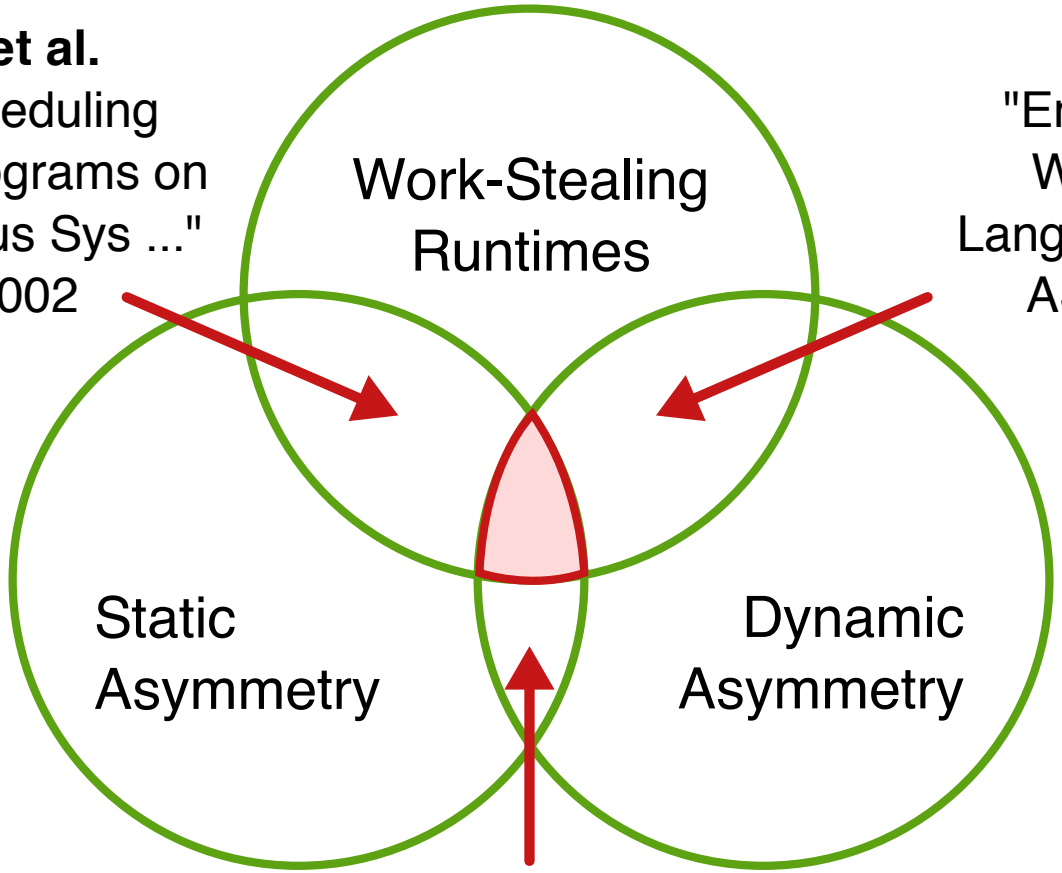


Intel Haswell integrates the voltage control loop circuitry *on-die* with inductors *in-package*.

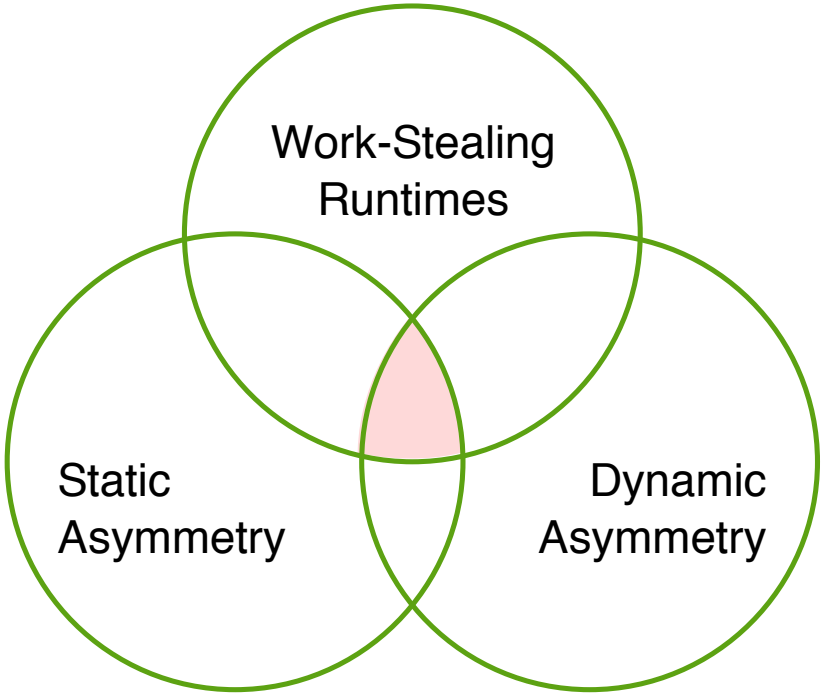
How can we use asymmetry awareness to improve the performance and energy efficiency of a work-stealing runtime?

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

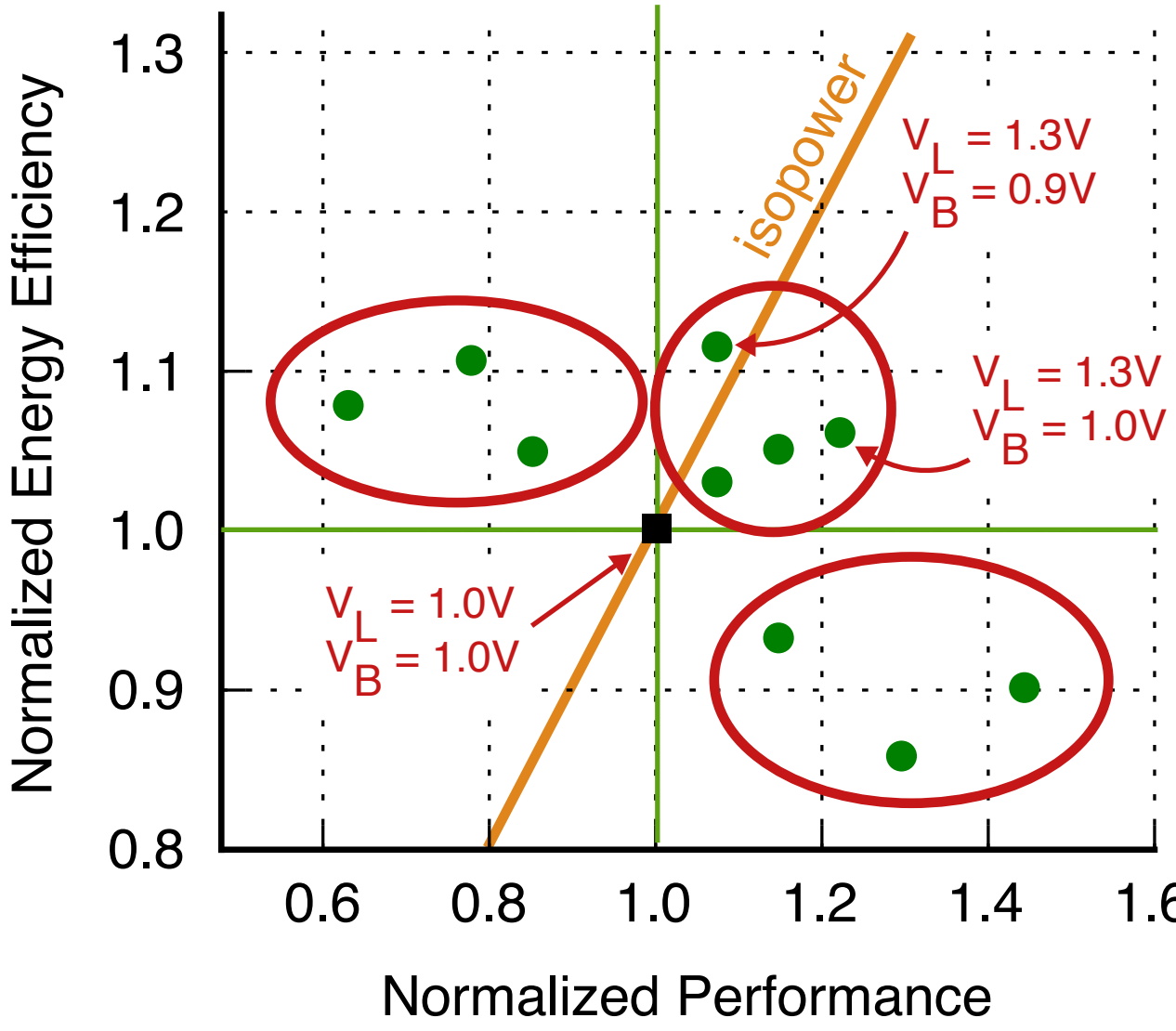
First-Order Modeling

Asymmetry-Aware
Work-Stealing Runtimes

Evaluation

Lower Performance
Higher Energy Efficiency

Higher Performance
Higher Energy Efficiency



Systematic way to adjust voltage and frequency to move into upper right quadrant?

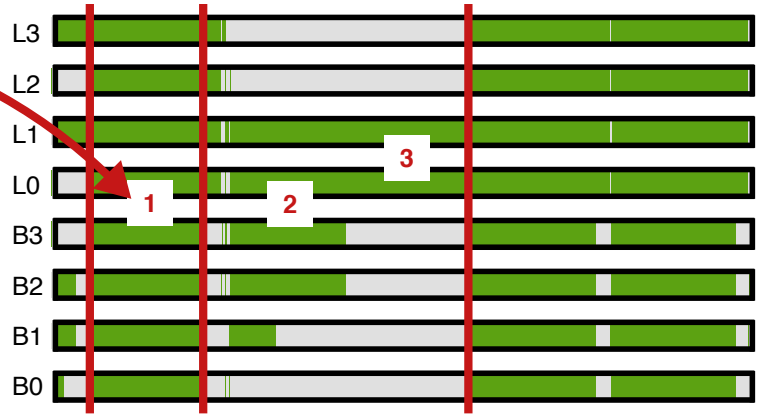
Use average power at V_{nom} as an optimization constraint

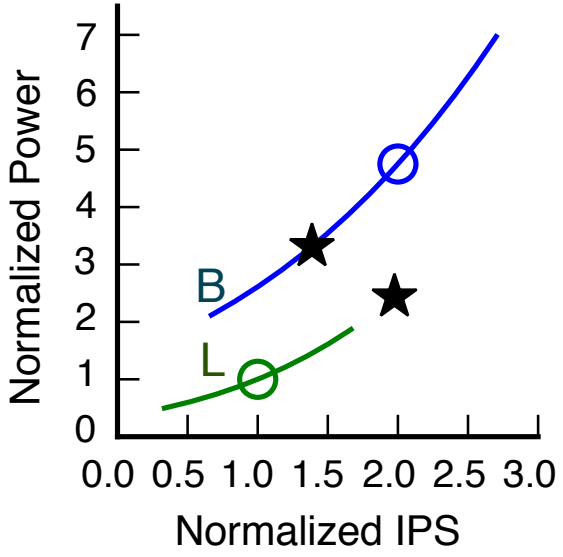
Higher Performance
Lower Energy Efficiency

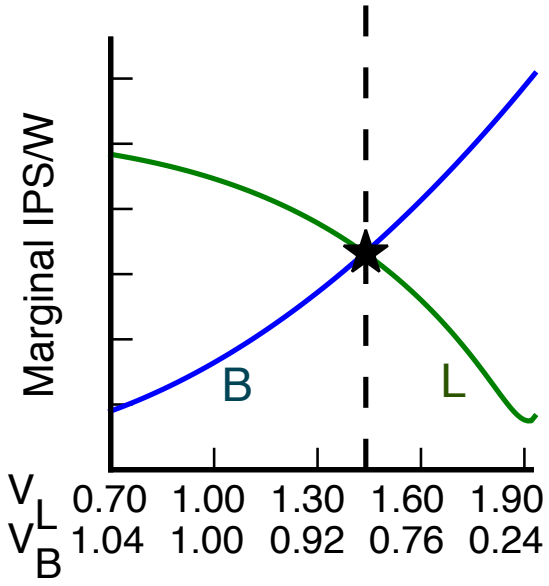
Opportunity #1

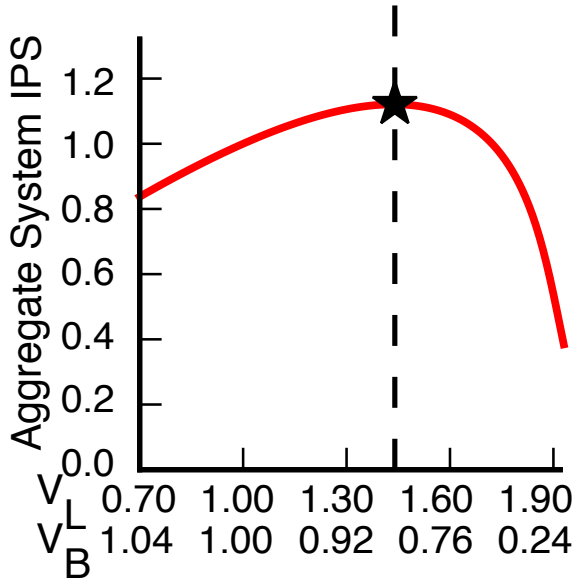
Balance performance/power
across cores when all cores are busy

Technique #1: Work-Pacing







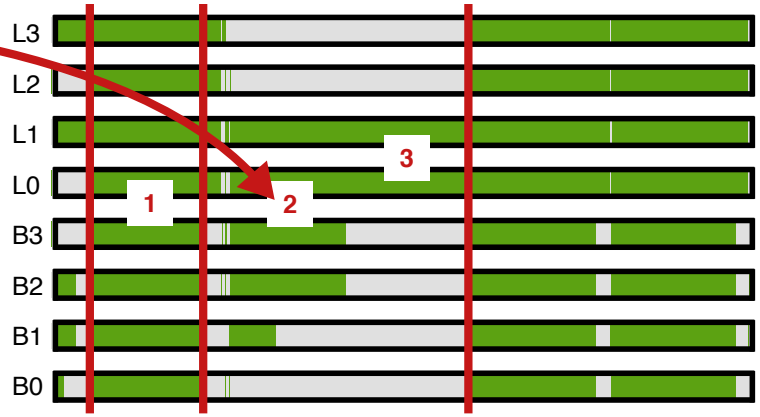


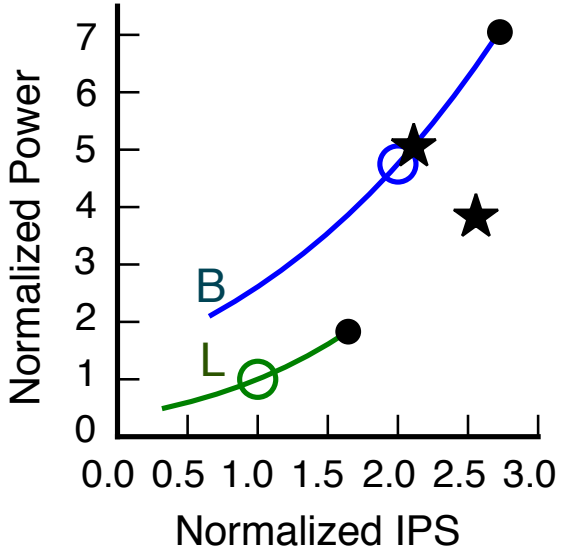
System with four active big cores and four active little cores

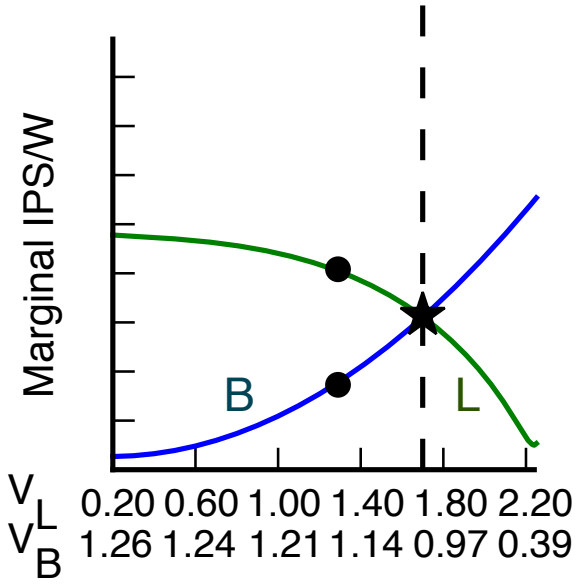
Opportunity #2

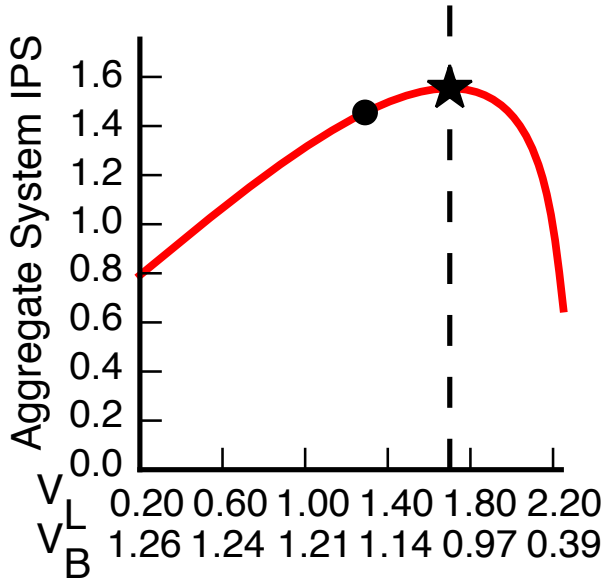
Rest waiting cores then balance performance and power acrossy busy cores

Technique #2: Work-Sprinting







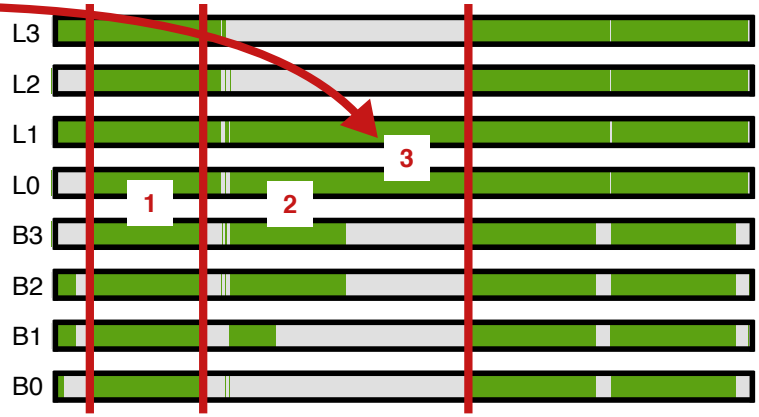


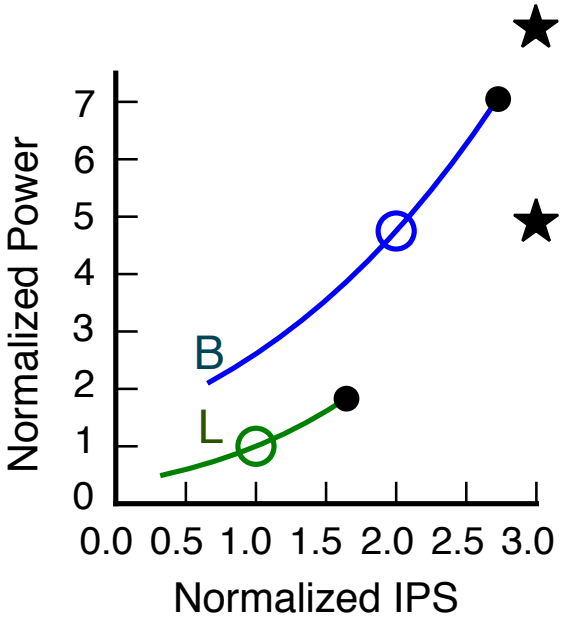
4B4L system with two active big cores and two active little cores

Opportunity #3

Move work from slow little cores to fast big cores

Technique #3: Work-Mugging





On Little Core

Optimal V_L = 2.59V

Feasible V_L = 1.3V

Speedup of 1.6x

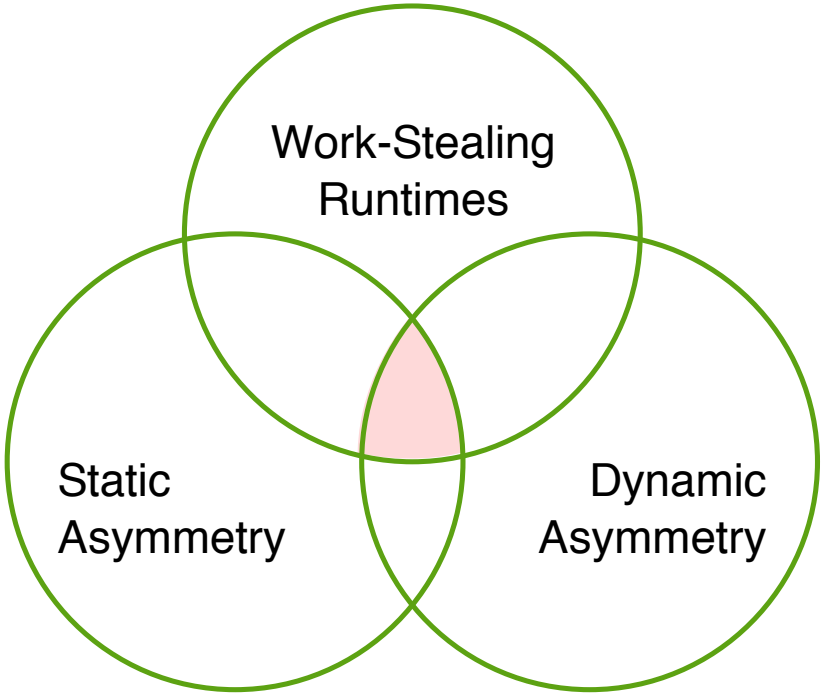
On Big Core

Optimal V_B = 1.51V

Feasible V_B = 1.3V

Speedup of 3.3x

4B4L system with either one active big core OR one active little core



Talk Outline

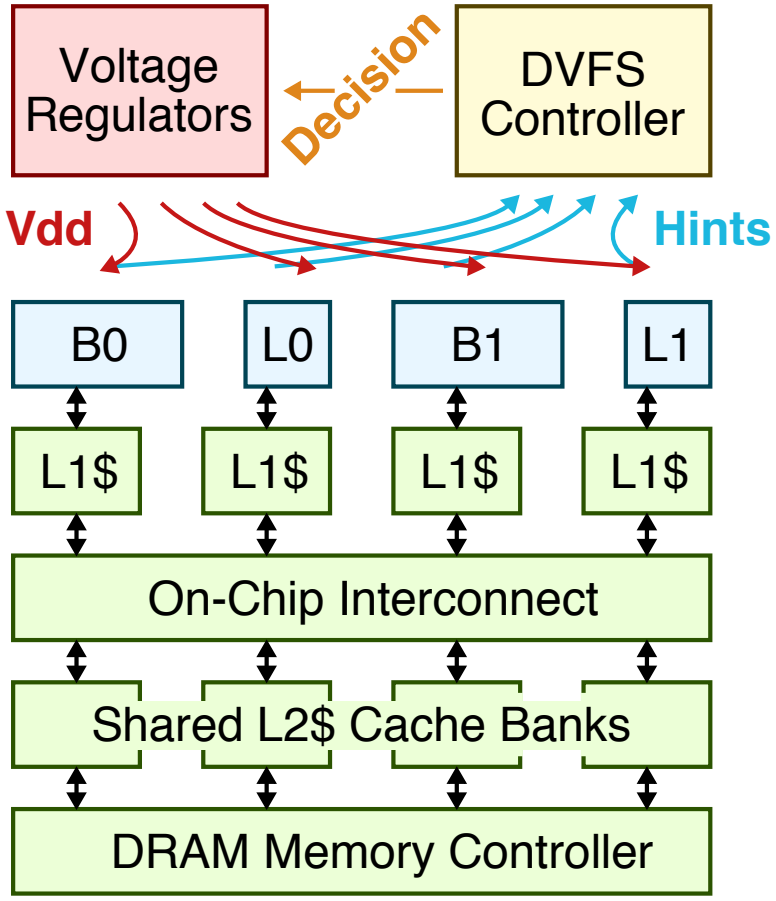
Motivation

First-Order Modeling

Asymmetry-Aware
Work-Stealing Runtimes

Evaluation

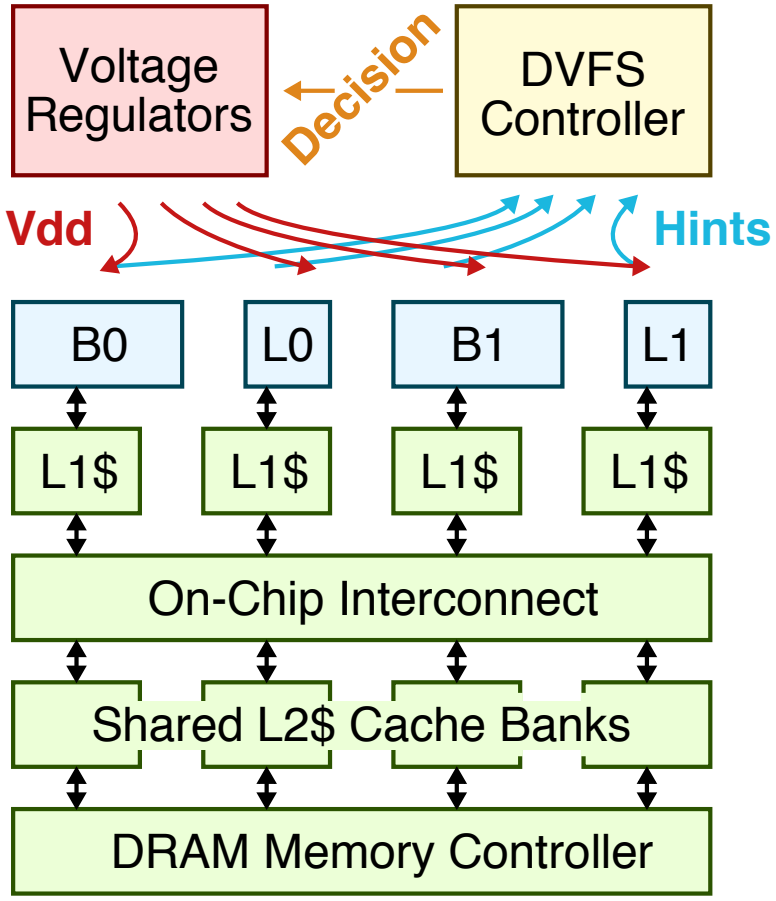
AAWS Runtime Techniques



Work-Pacing

When all cores are busy, tune V&F of little and big cores (based on offline marginal utility analysis) to increase performance and energy efficiency

AAWS Runtime Techniques



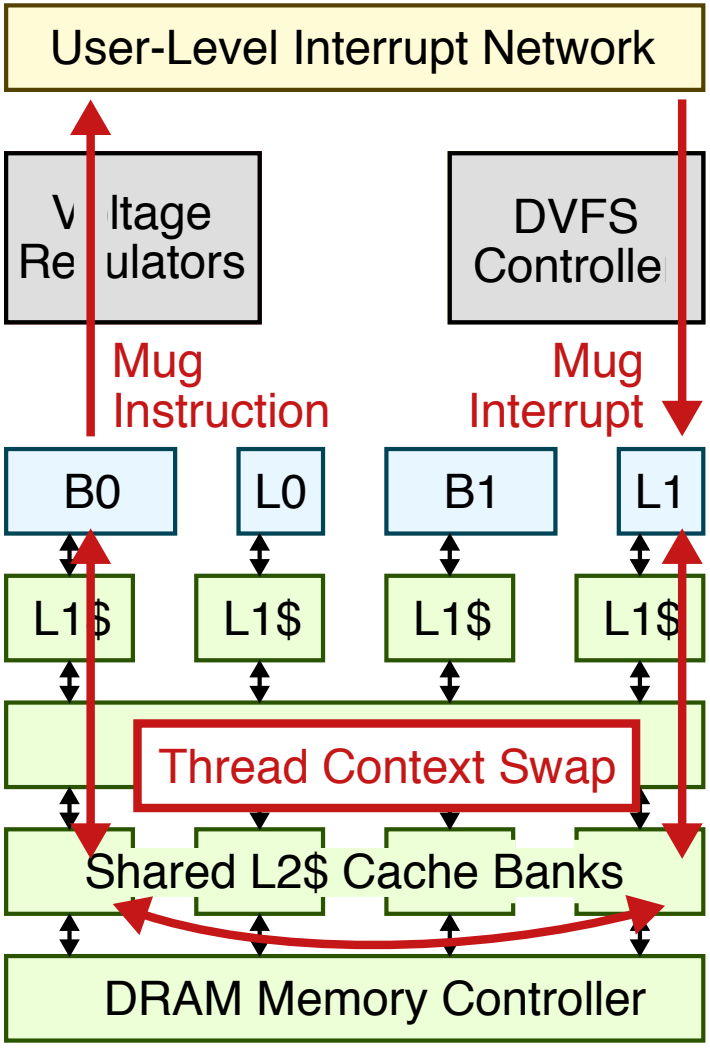
Work-Pacing

When all cores are busy, tune V&F of little and big cores (based on offline marginal utility analysis) to increase performance and energy efficiency

Work-Sprinting

Rest waiting cores to generate power slack, tune V&F of busy cores (based on offline marginal utility analysis)

AAWS Runtime Techniques



Work-Pacing

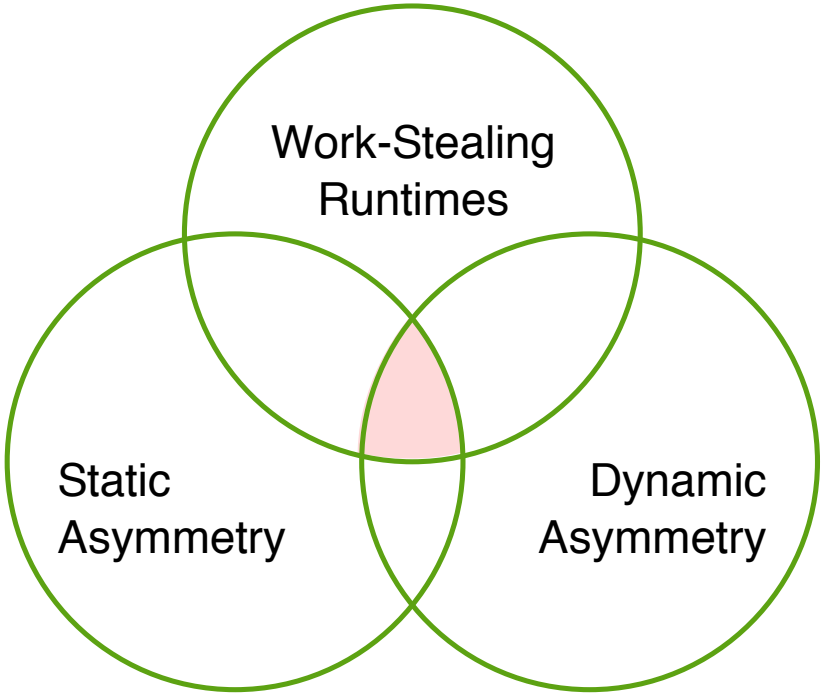
When all cores are busy, tune V&F of little and big cores (based on offline marginal utility analysis) to increase performance and energy efficiency

Work-Sprinting

Rest waiting cores to generate power slack, tune V&F of busy cores (based on offline marginal utility analysis)

Work-Mugging

Move work from little to big cores by allowing big cores to preemptively mug tasks from little cores



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
- Automatic recursive decomposition of parallel loop task chunking
- Instrumented with activity hints

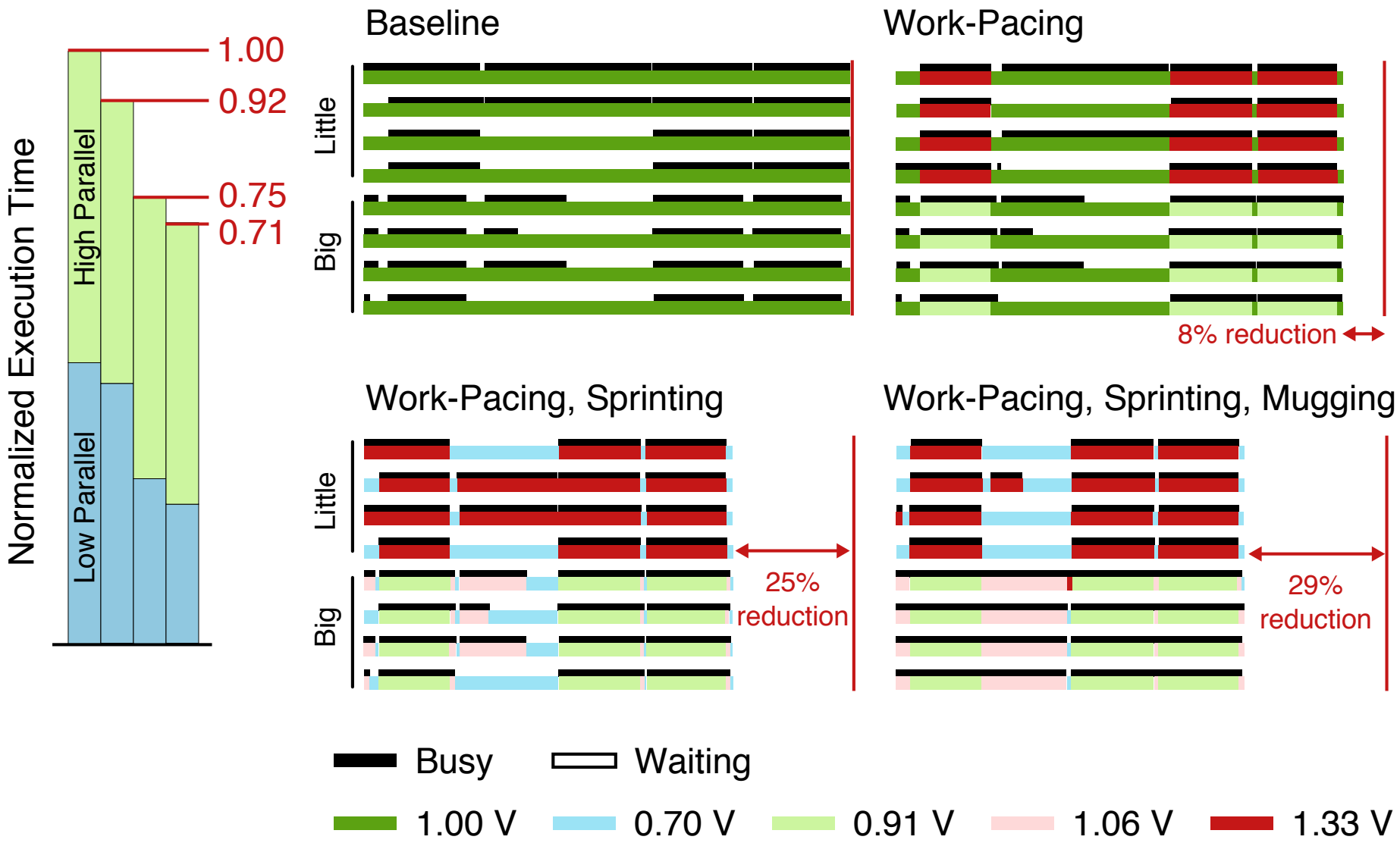
Cycle-Level Modeling

- Heterogeneous system modeled in gem5 cycle-approximate simulator
- Support for scaling per-core frequencies + central DVFS Controller
- Accounting for intercore interrupt latency during mugging

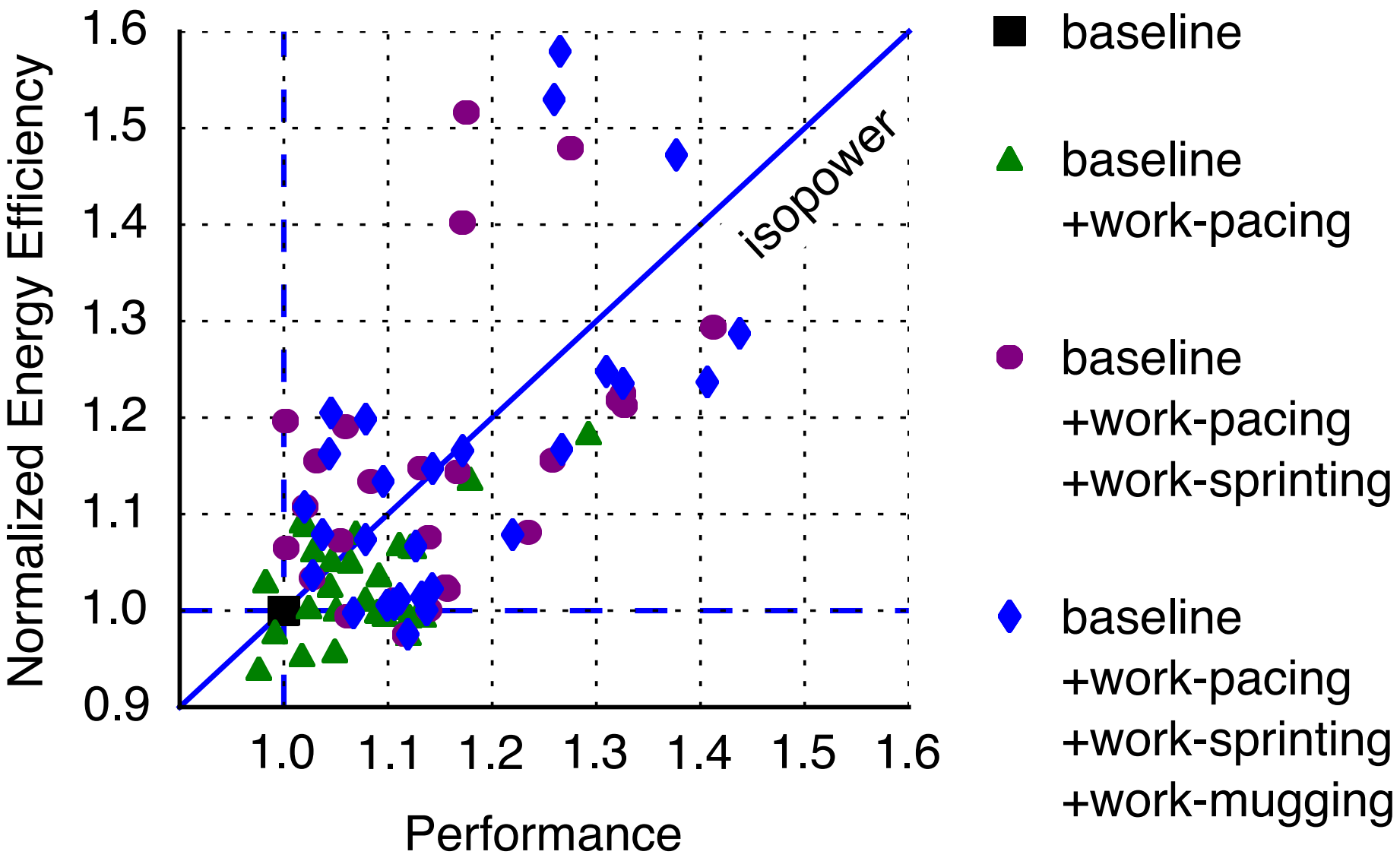
Energy Modeling

- Event-based energy modeling based on McPAT models and detailed RTL/gate-level sims (Synopsys ASIC toolflow, TSMC LP, 65 nm 1.0 V)
- Per-inst energy benchmarks to isolate event energy (e.g., rfile reads)

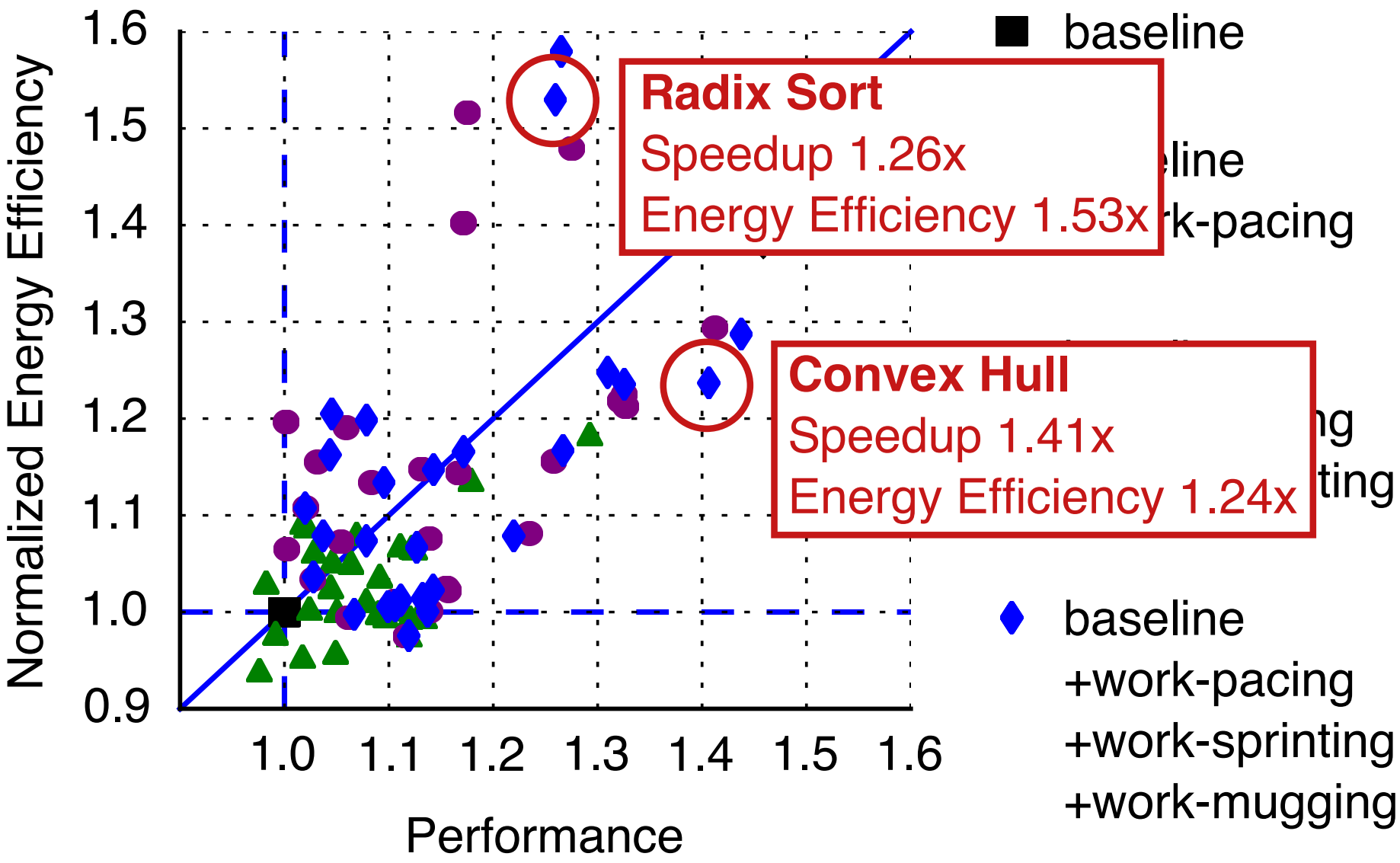
Performance of Convex Hull Application

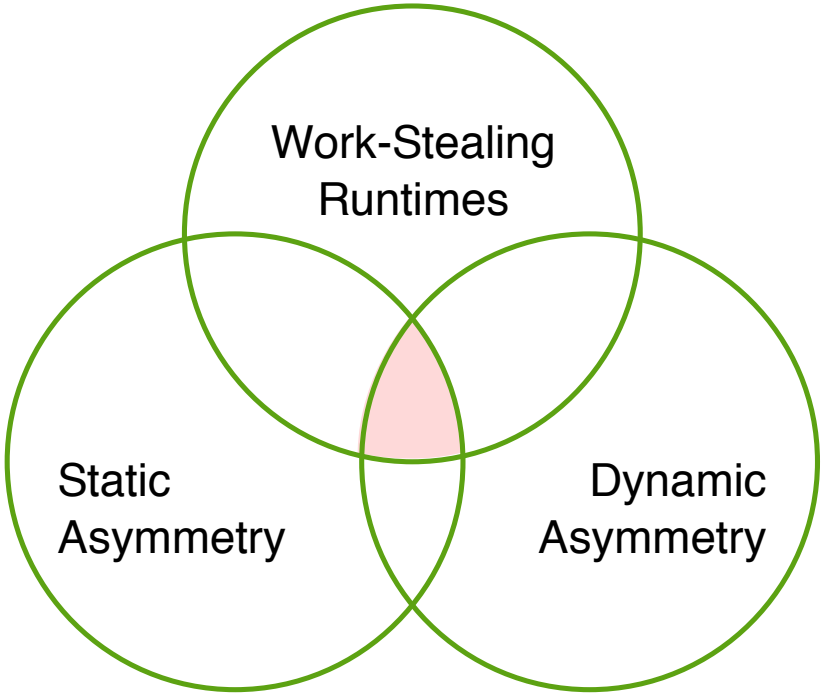


Energy-Efficiency and Performance Results



Energy-Efficiency and Performance Results





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