Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs

José A. Joao*
M. Aater Suleman*
Onur Mutlu‡
Yale N. Patt*

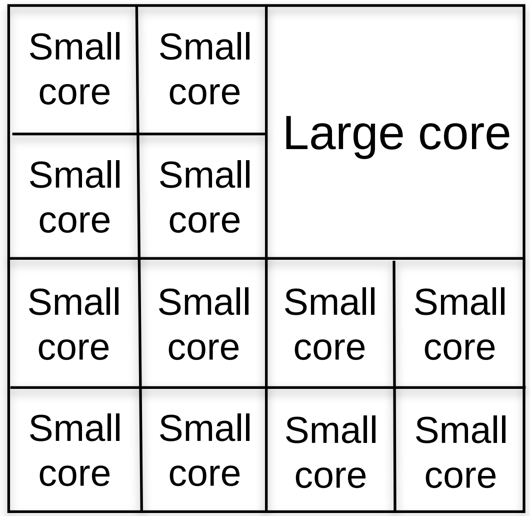
*** HPS Research Group**
University of Texas at Austin

‡ Computer Architecture Laboratory
Carnegie Mellon University

THE UNIVERSITY OF TEXAS AT AUSTIN
UT ECE
ELECTRICAL & COMPUTER ENGINEERING

Carnegie Mellon

Asymmetric CMP (ACMP)

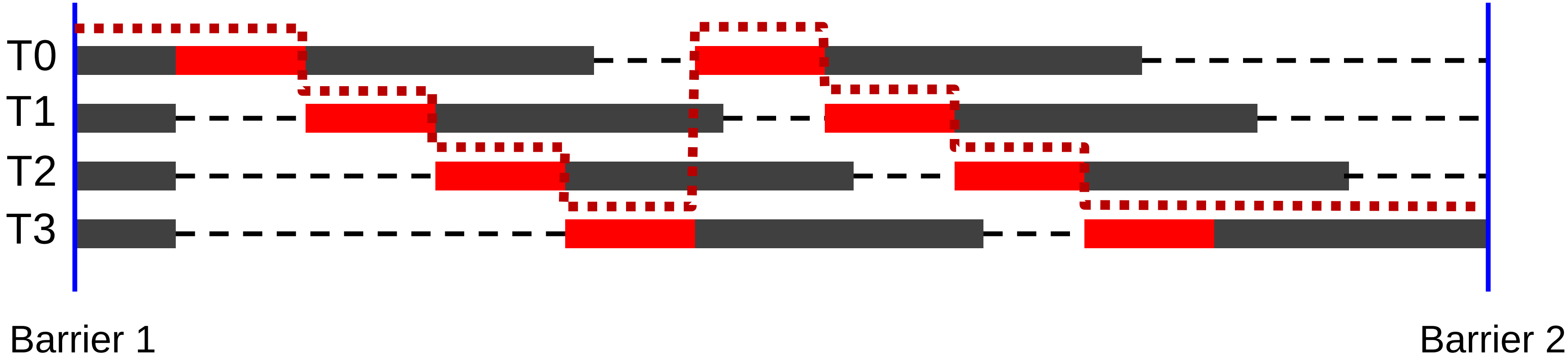


- One or a few large, out-of-order cores, fast
- Many small, in-order cores, power-efficient

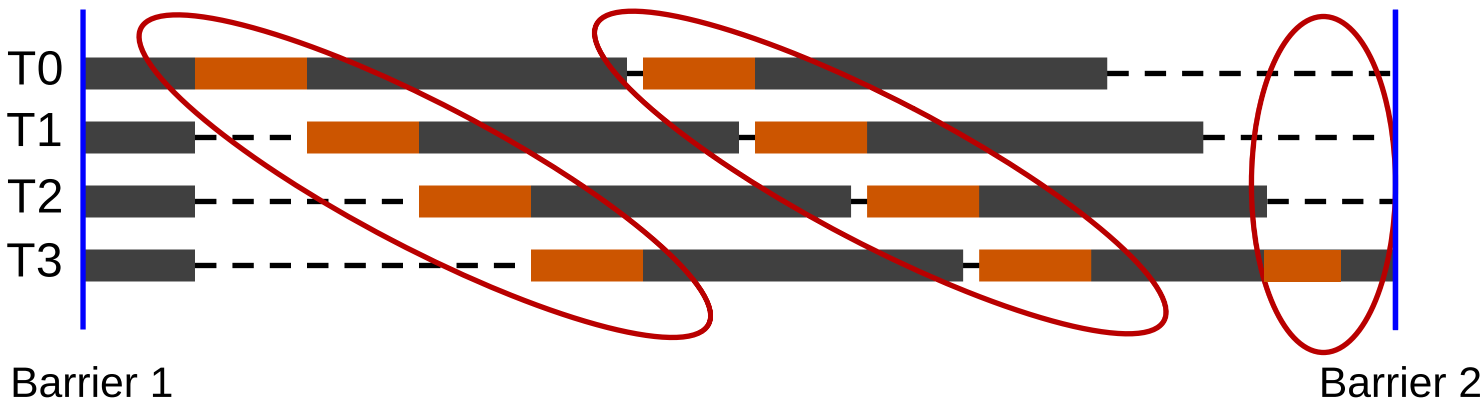
???

- Critical code segments run on large cores
- The rest of the code runs on small cores

Bottlenecks

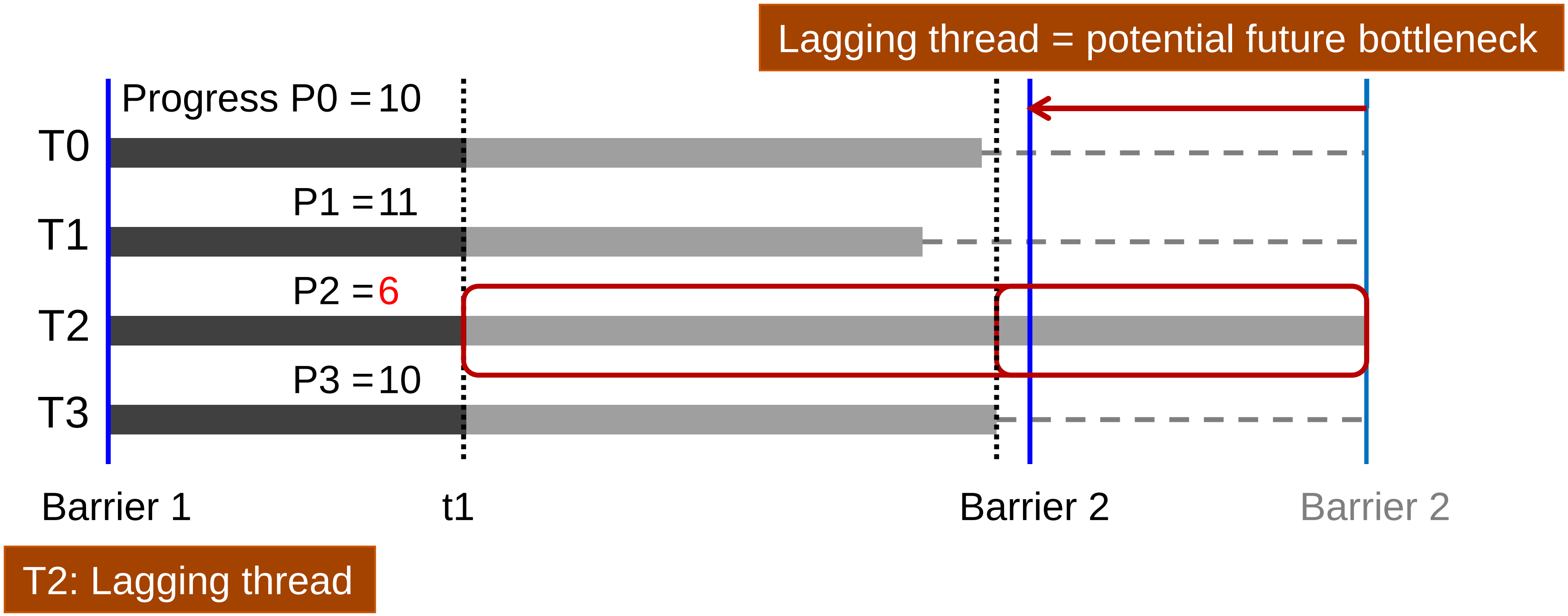


Accelerating Critical Sections (ACS), Suleman et al., ASPLOS'09



Bottleneck Identification and Scheduling (BIS), Joao et al., ASPLOS'12

Lagging Threads

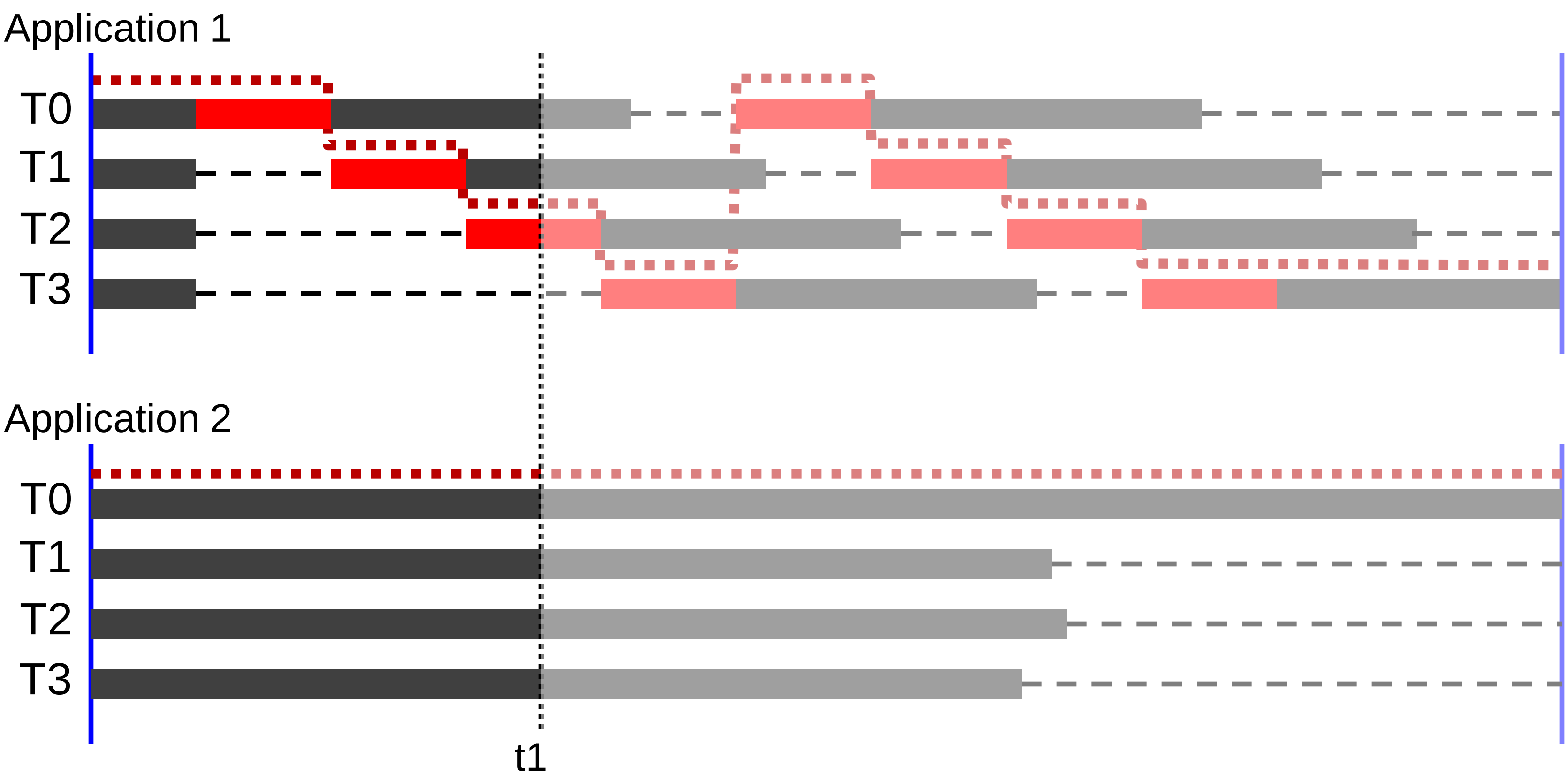


Previous work about progress of multithreaded applications:

- Meeting points, Cai et al., PACT'08
- Thread criticality predictors, Bhattacharjee and Martonosi, ISCA'09
- Age-based scheduling (AGETS), Lakshminarayana at al., SC'09

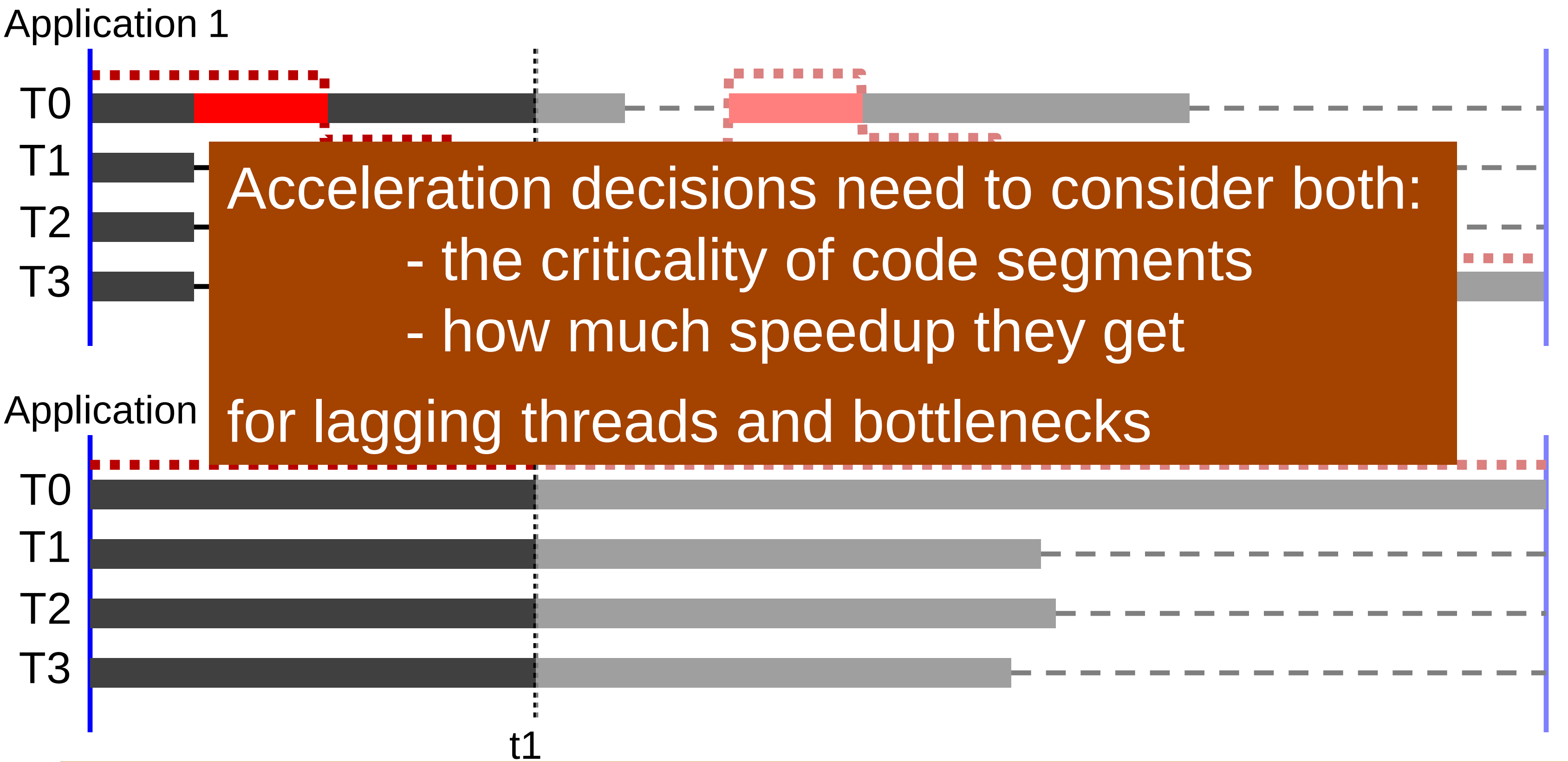
Two problems

- 1) Do we accelerate bottlenecks or lagging threads?
- 2) Multiple applications: which application do we accelerate?



Two problems

- 1) Do we accelerate bottlenecks or lagging threads?
- 2) Multiple applications: which application do we accelerate?



Acceleration decisions need to consider both:

- the criticality of code segments
- how much speedup they get

for lagging threads and bottlenecks

Utility-Based Acceleration (UBA)

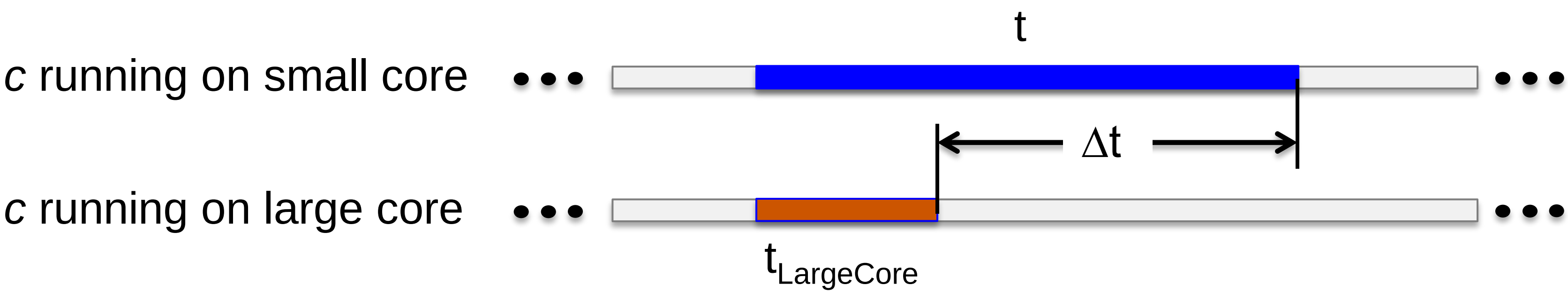
- *Goal*: identify performance-limiting bottlenecks or lagging threads from any running application and accelerate them on large cores of an ACMP
- *Key insight*: a Utility of Acceleration metric that combines speedup and criticality of each code segment
- Utility of accelerating code segment *c* of length t on an application of length T:

$$U_c = \frac{\Delta T}{T} = \underbrace{\boxed{\left(\frac{\Delta t}{t}\right)}}_{L} \times \underbrace{\boxed{\left(\frac{t}{T}\right)}}_{R} \times \underbrace{\boxed{\left(\frac{\Delta T}{\Delta t}\right)}}_{G}$$

L: Local acceleration of *c*

$$U_c = L \times R \times G$$

How much code segment *c* is accelerated



$$L = \frac{\Delta t}{t} = \frac{t - t_{LargeCore}}{t} = 1 - \frac{1}{S}$$

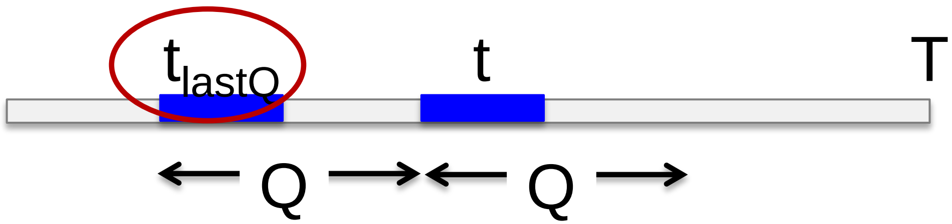
S: speedup of *c*

- Estimate S = estimate performance on a large core while running on a small core
- Performance Impact Estimation (PIE, Van Craeynest et al., ISCA'12) : considers both instruction-level parallelism (ILP) and memory-level parallelism (MLP) to estimate CPI

R: Relevance of code segment *c*

$$U_c = L \times R \times G$$

How relevant code segment *c* is for the application



Q: scheduling quantum

$$R = \frac{t}{T}$$

$$R_{estimated} = \frac{t_{lastQ}}{Q}$$

G: Global effect of accelerating *c*

How much accelerating *c* reduces total execution time

$$\boxed{U_c = L \times R \times G}$$

$$G = \frac{\Delta T}{\Delta t}$$

→ Acceleration of *application*

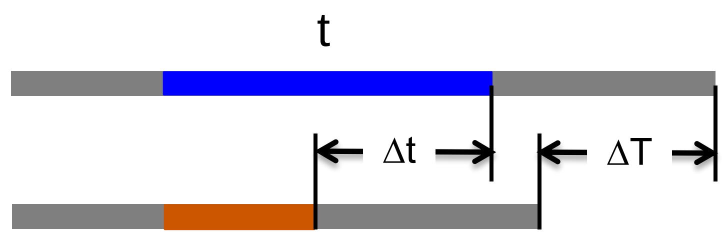
→ Acceleration of *c*

Criticality of *c*

Single thread

c running on small core •••

c running on large core •••



G=1

G: Global effect of accelerating *c*

How much accelerating *c* reduces total execution time

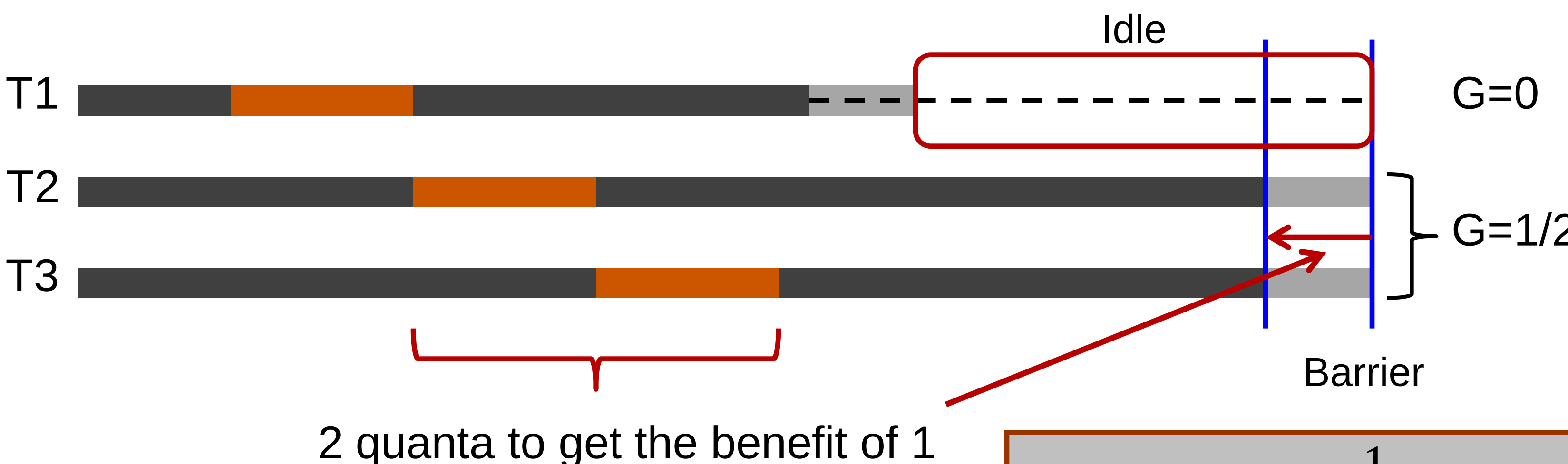
$$U_c = L \times R \times G$$

$$G = \frac{\Delta T}{\Delta t}$$

→ Acceleration of *application*

→ Acceleration of *c*

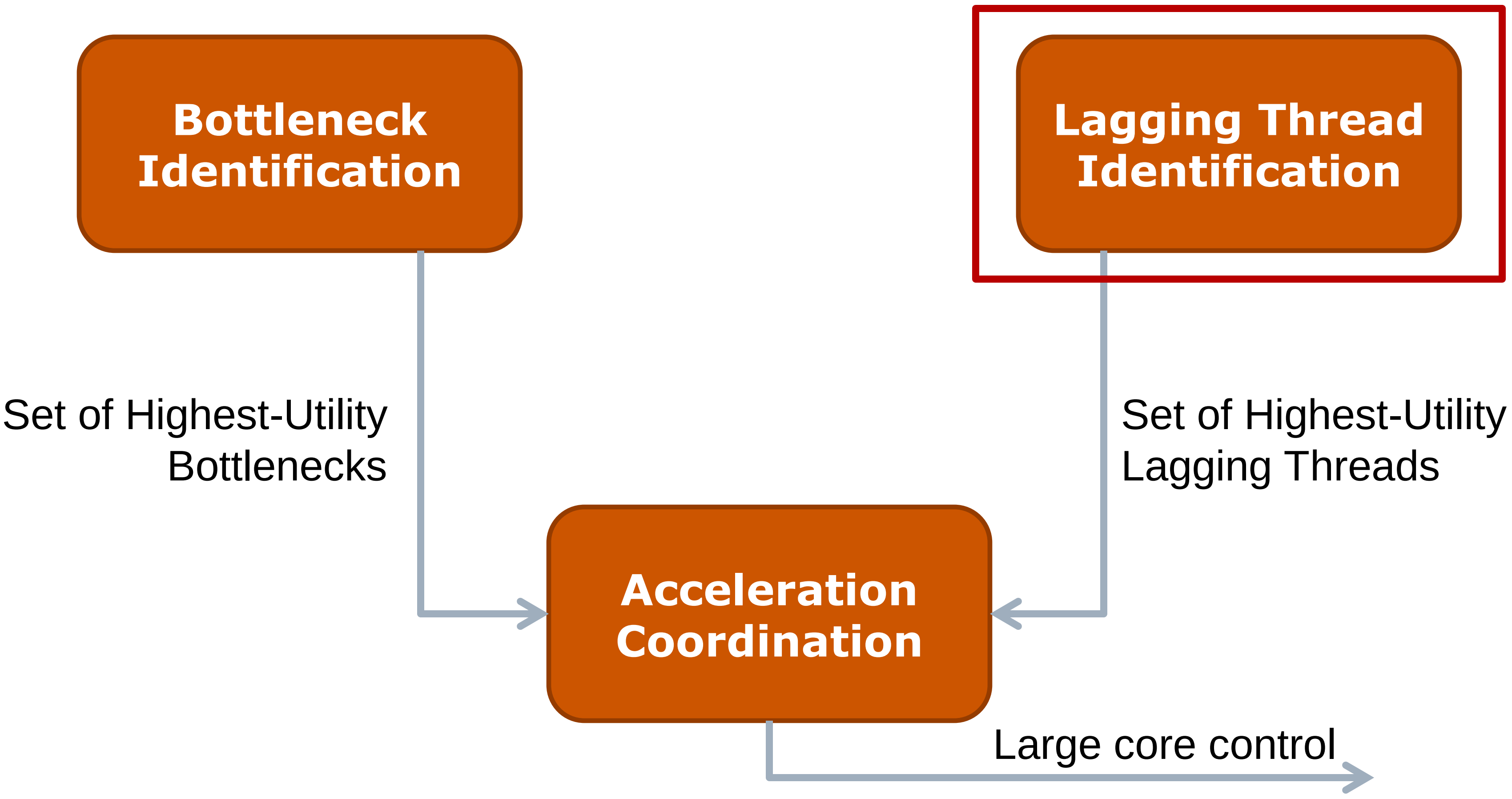
Criticality of *c*



$$G = \frac{1}{\text{Number of Lagging Threads}}$$

Critical sections: classify into *strongly-contended* and *weakly-contended* and estimate G differently (in the paper)

Utility-Based Acceleration (UBA)

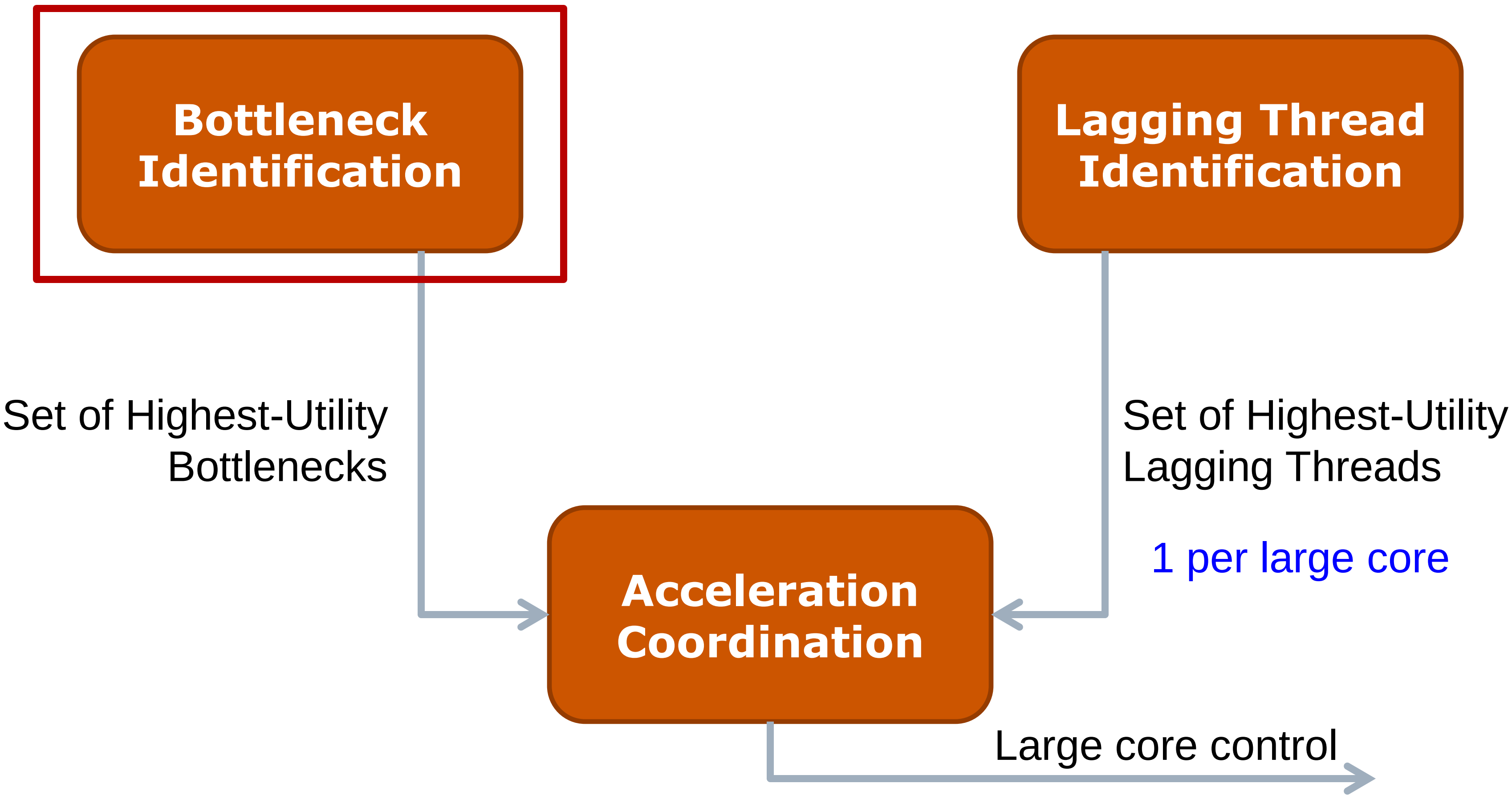


Lagging thread identification

- Lagging threads are those that are making the least progress
- How to define and measure progress? → Application-specific problem
  - We borrow from Age-Based Scheduling (SC'09)
    - Progress metric (committed instructions)
    - Assumption: same number of committed instructions between barriers
  - But we could easily use any other progress metric…

- Minimum progress = minP
- Set of lagging threads = { any thread with progress < minP + ΔP }
- Compute Utility for each lagging thread

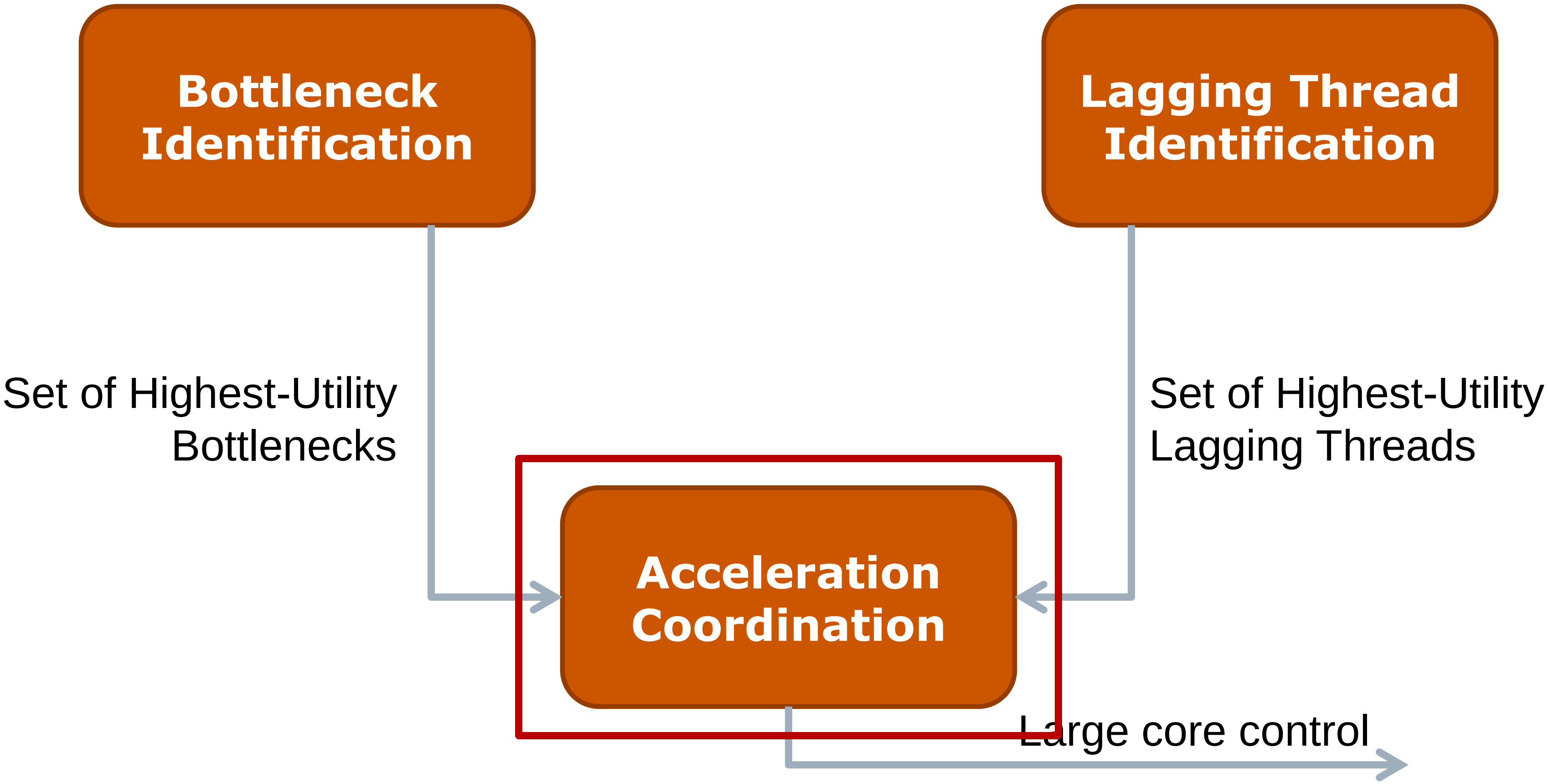
Utility-Based Acceleration (UBA)



Bottleneck identification

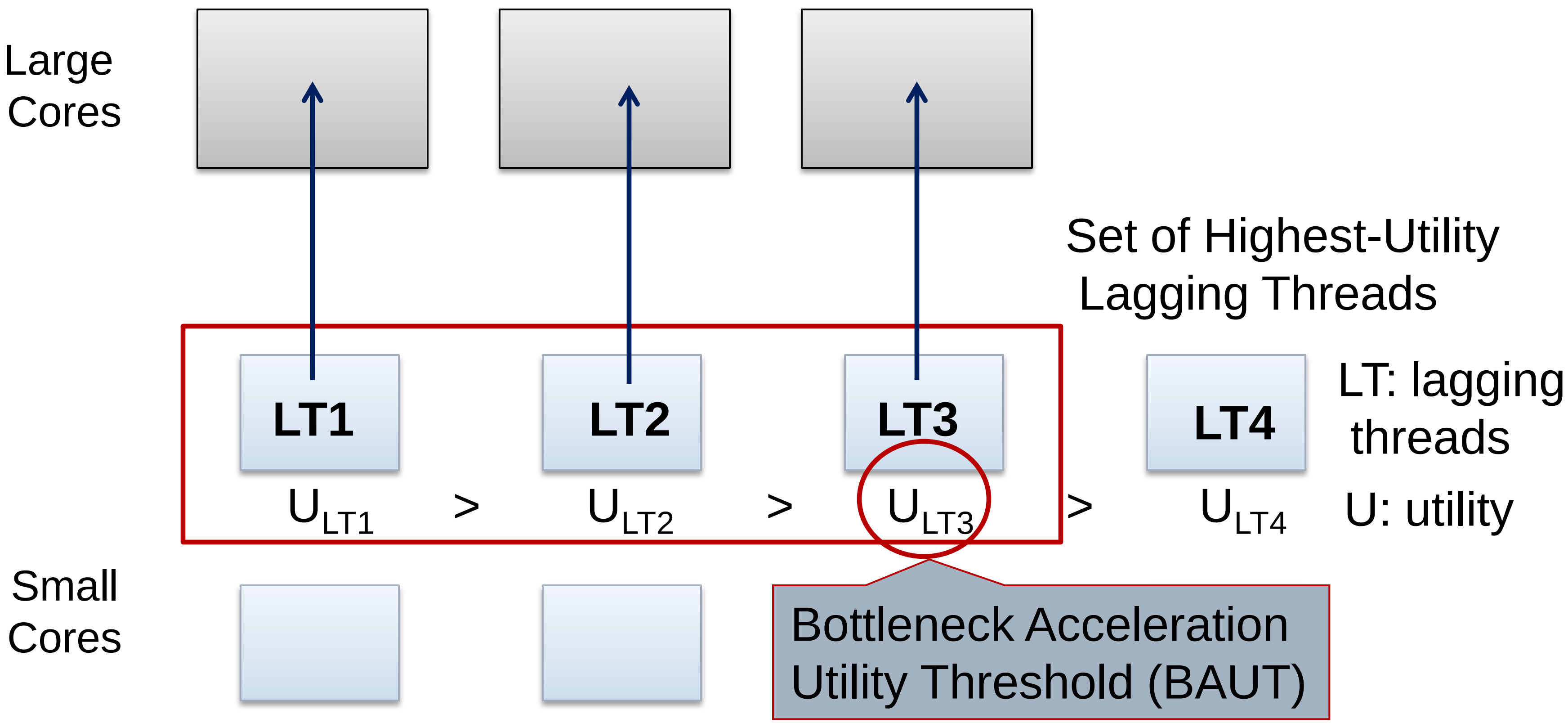
- Software: programmer, compiler or library
  - Delimit potential bottlenecks with BottleneckCall and BottleneckReturn instructions
  - Replace code that waits with a BottleneckWait instruction
- Hardware: Bottleneck Table
  - Keep track of threads executing or waiting for bottlenecks
  - Compute Utility for each bottleneck
  - Determine set of Highest-Utility Bottlenecks
- Similar to our previous work BIS, ASPLOS'12
  - BIS uses *thread waiting cycles* instead of Utility

Utility-Based Acceleration (UBA)

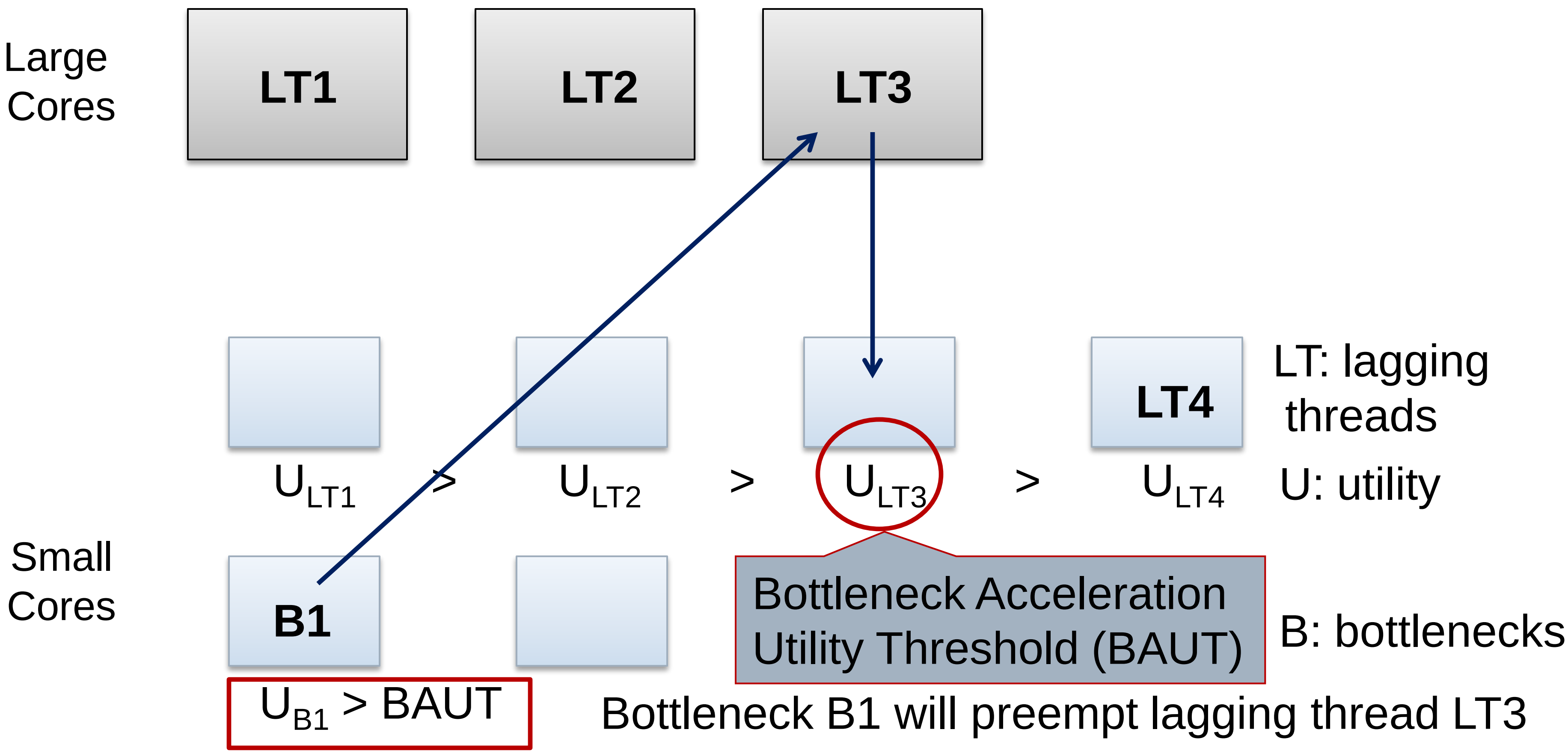


Acceleration coordination

LT assigned to each large core every quantum

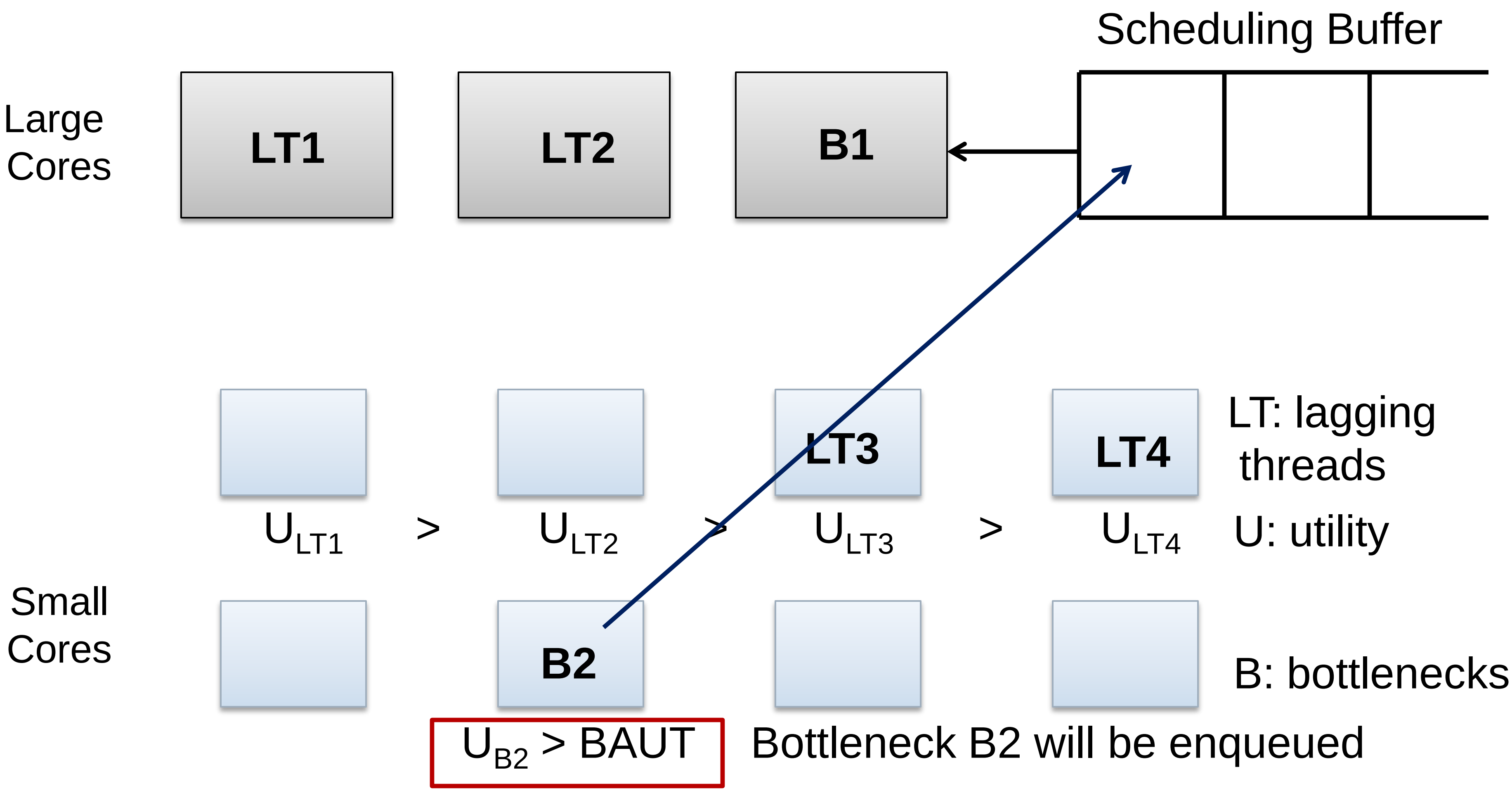


Acceleration coordination

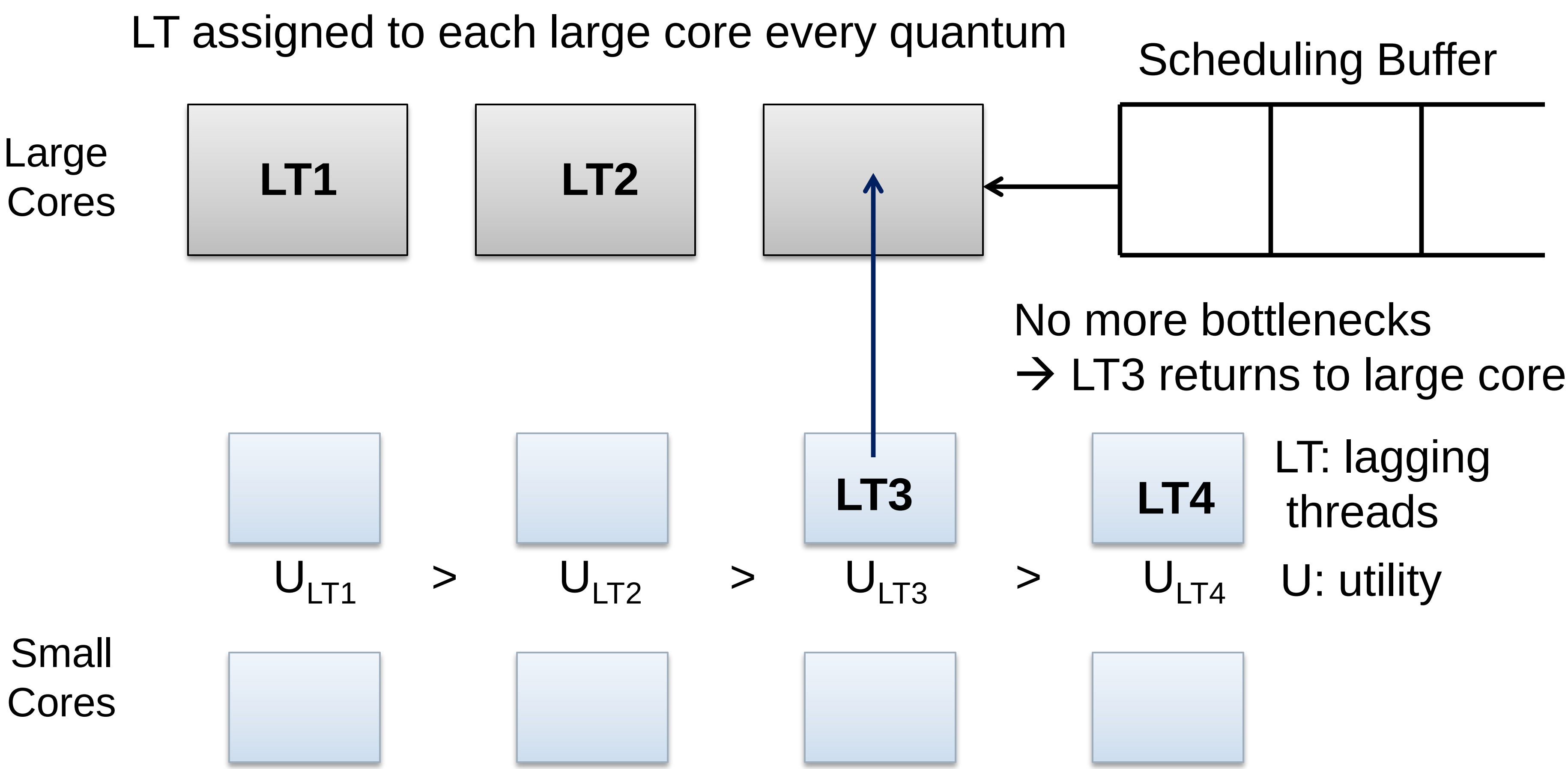


Bottleneck B1 will preempt lagging thread LT3

Acceleration coordination



Acceleration coordination



Methodology

- Workloads
  - single-application: 9 multithreaded applications
    with different impact from bottlenecks
  - 2-application: all 55 combinations of (9 MT + 1 ST)
  - 4-application: 50 random combinations of (9 MT + 1 ST)
- Processor configuration
  - x86 ISA
  - Area of large core = 4 x Area of small core
  - Large core: 4GHz, out-of-order, 128-entry ROB, 4-wide, 12-stage
  - Small core: 4GHz, in-order, 2-wide, 5-stage
  - Private 32KB L1, private 256KB L2, shared 8MB L3
  - On-chip interconnect: Bi-directional ring, 2-cycle hop latency

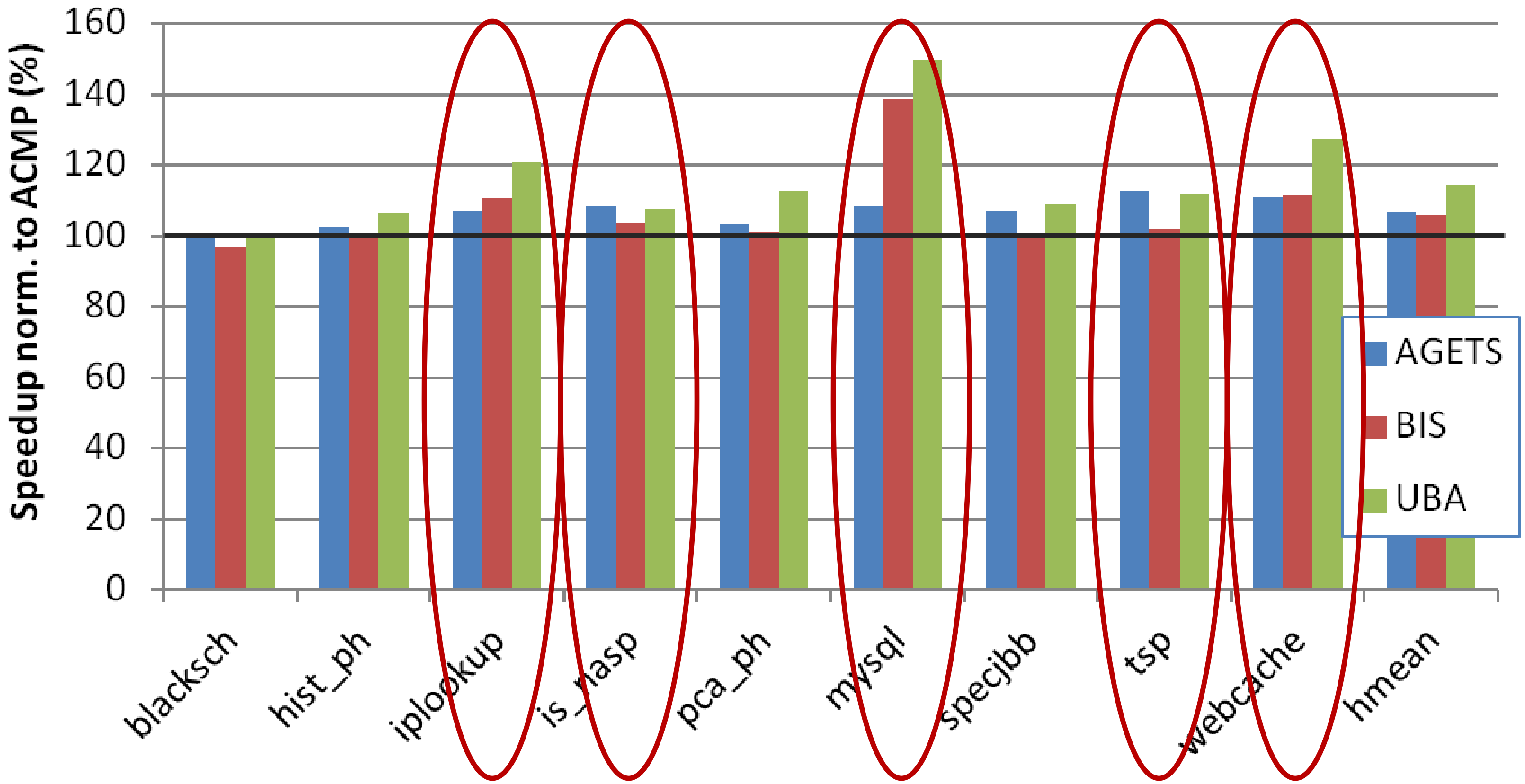
Comparison points

- Single application
  - ACMP (Morad et al., Comp. Arch. Letters'06)
    - only accelerates Amdahl's serial bottleneck
  - Age-based scheduling (AGETS, Lakshminarayana et al., SC'09)
    - only accelerates lagging threads
  - Bottleneck Identification and Scheduling (BIS, Joao et al., ASPLOS'12)
    - only accelerates bottlenecks

- Multiple applications
  - AGETS+PIE: select most lagging thread with AGETS and use PIE across applications
    - only accelerates lagging threads
  - MA-BIS: BIS with shared large cores across applications
    - only accelerates bottlenecks

Single application, 1 large core

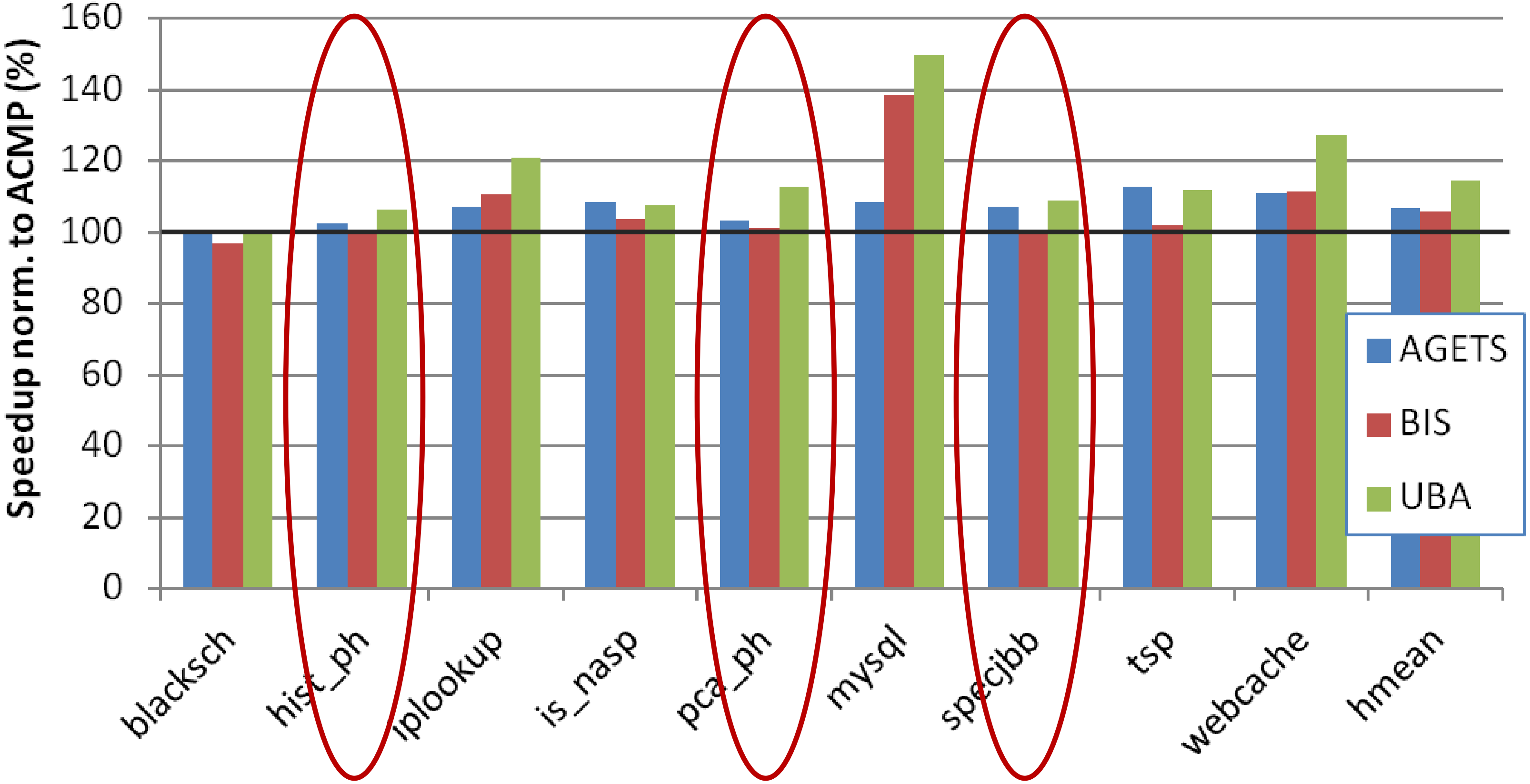
Optimal number of threads, 28 small cores, 1 large core



Limiting critical sections: benefit from BIS and UBA

Single application, 1 large core

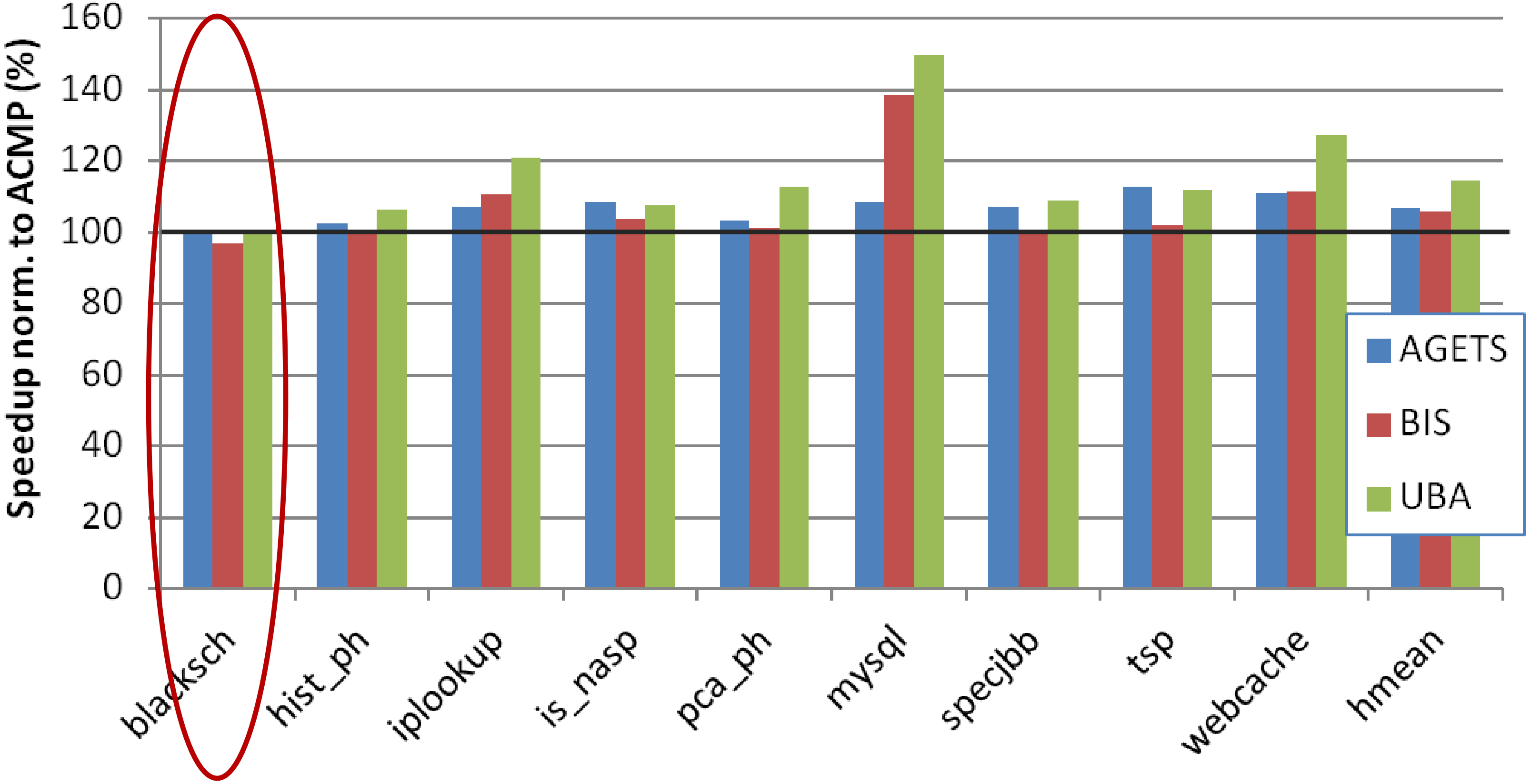
Optimal number of threads, 28 small cores, 1 large core



Lagging threads: benefit from AGETS and UBA

Single application, 1 large core

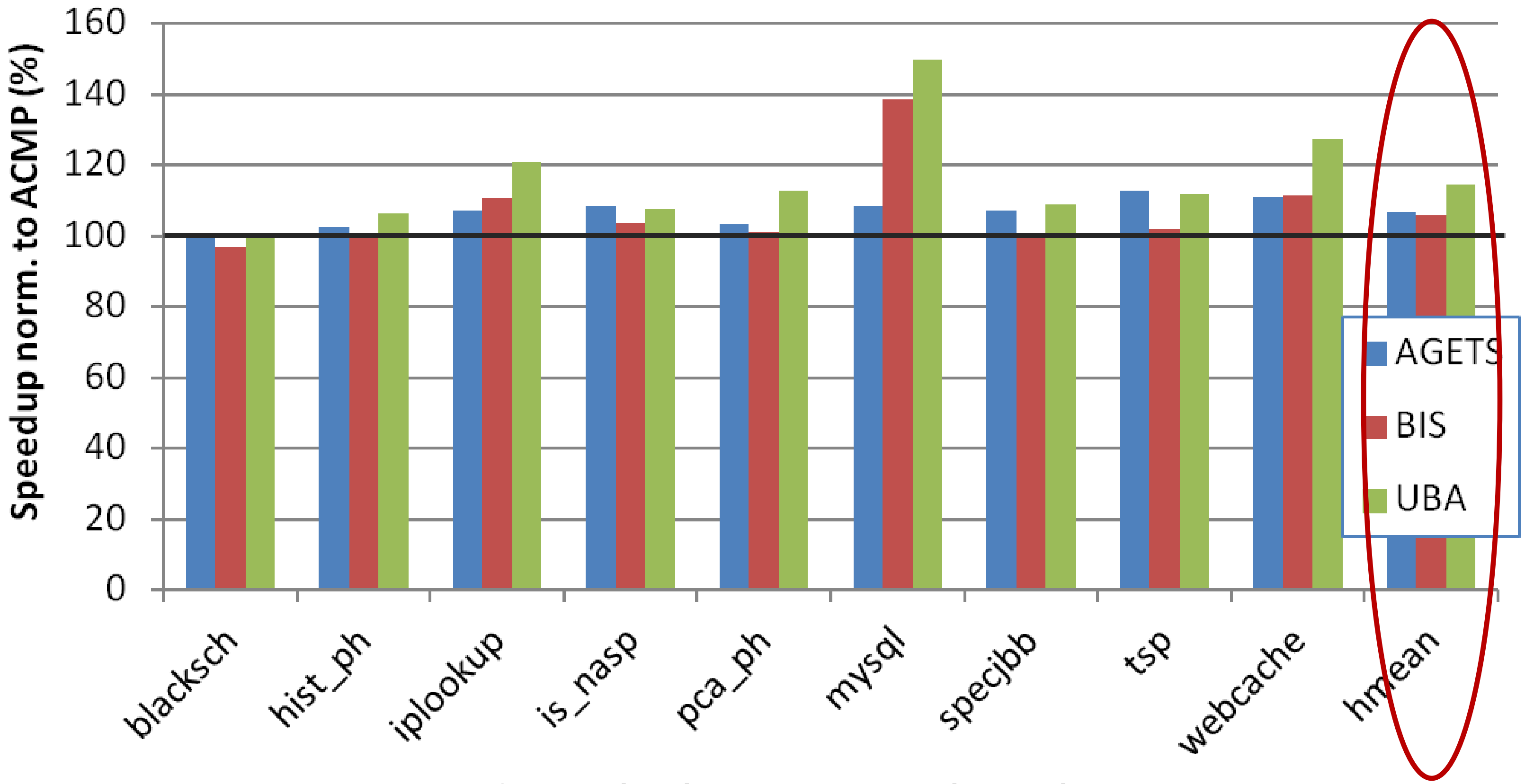
Optimal number of threads, 28 small cores, 1 large core



Neither bottlenecks
nor lagging threads

Single application, 1 large core

Optimal number of threads, 28 small cores, 1 large core

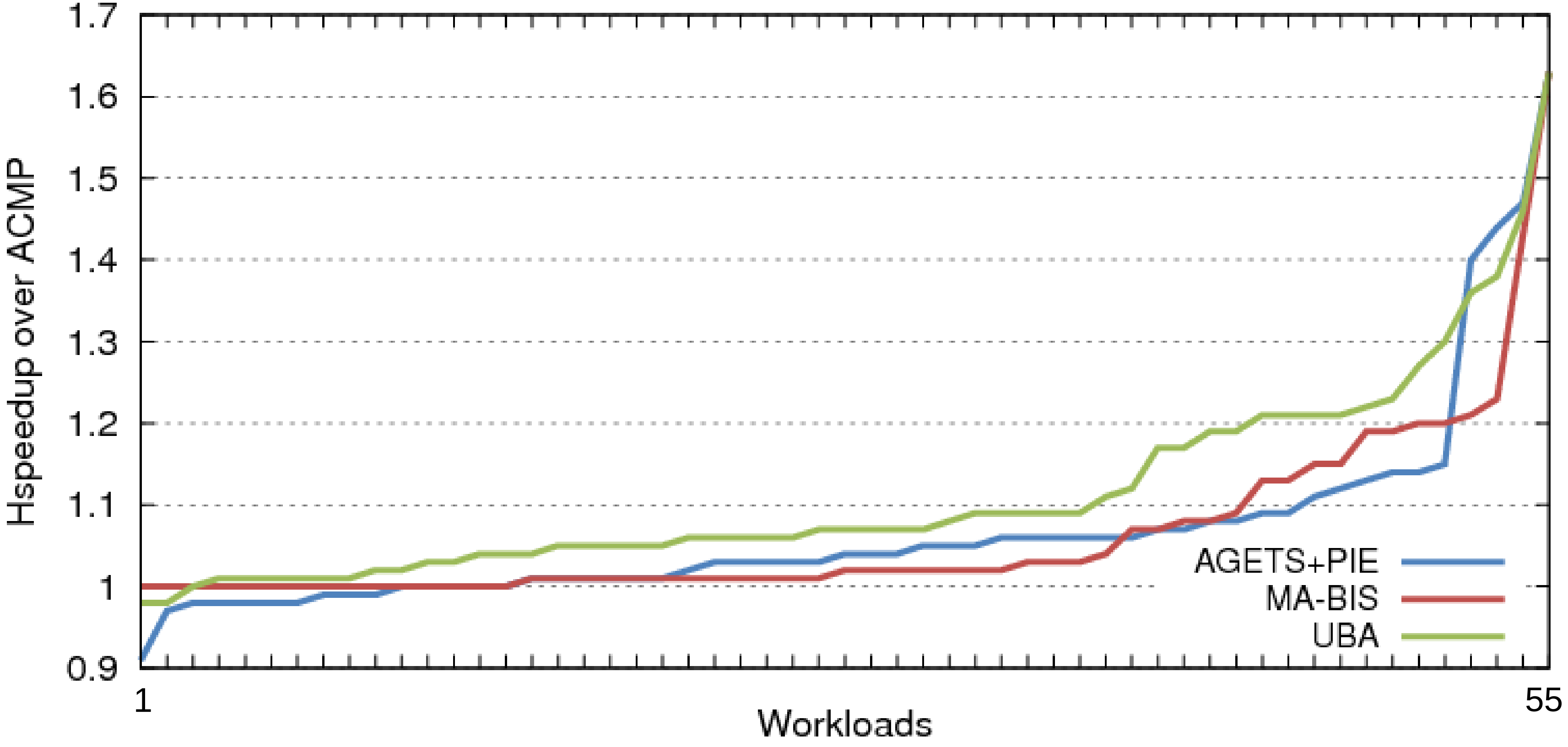


UBA outperforms both AGETS and BIS by 8%

UBA's benefit increases with area budget and number of large cores

Multiple applications

2-application workloads, 60 small cores, 1 large core



UBA improves Hspeedup over AGETS+PIE and MA-BIS by 2 to 9%

Summary

- To effectively use ACMPs:
  - Accelerate both fine-grained bottlenecks and lagging threads
  - Accelerate single and multiple applications
- Utility-Based Acceleration (UBA) is a cooperative software-hardware solution to both problems
- Our Utility of Acceleration metric combines a measure of acceleration and a measure of criticality to allow meaningful comparisons between code segments
- Utility is implemented for an ACMP but is general enough to be extended to other acceleration mechanisms
- UBA outperforms previous proposals for single applications and their aggressive extensions for multiple-application workloads
- UBA is a comprehensive fine-grained acceleration proposal for parallel applications without programmer effort

Thank You!

Questions?

Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs

José A. Joao*
M. Aater Suleman*
Onur Mutlu‡
Yale N. Patt*

* HPS Research Group
University of Texas at Austin

‡ Computer Architecture Laboratory
Carnegie Mellon University