WeeFence: Toward Making Fences Free in TSO

Yuelu Duan, Abdullah Muzahid, Josep Torrellas
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

i-acoma group

ISCA, June 2013

ILLINOIS

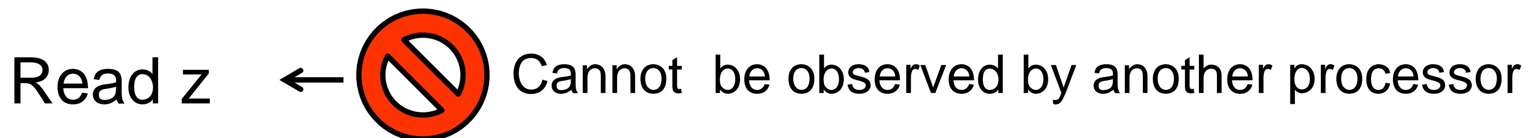
Fence: a Primitive for Parallelism

- Instruction inserted by programmers or compilers
- Prevents the compiler and HW from reordering memory accesses

Read x
Write y

Until these are finished
- loads retired
- writes retired + drained from write buffer

Fence



Use of Fences

Concurrency coordination with low overhead (much less than locks)

- Programmers insert fences in codes with fine-grain sharing
  - Work-stealing algorithm in Cilk
- Compilers insert fences in C++
  - Programmer uses intentional data race for performance → declares variable as atomic
  - Compiler inserts fence after the access, does not reorder
  - Hardware does not reorder across fence

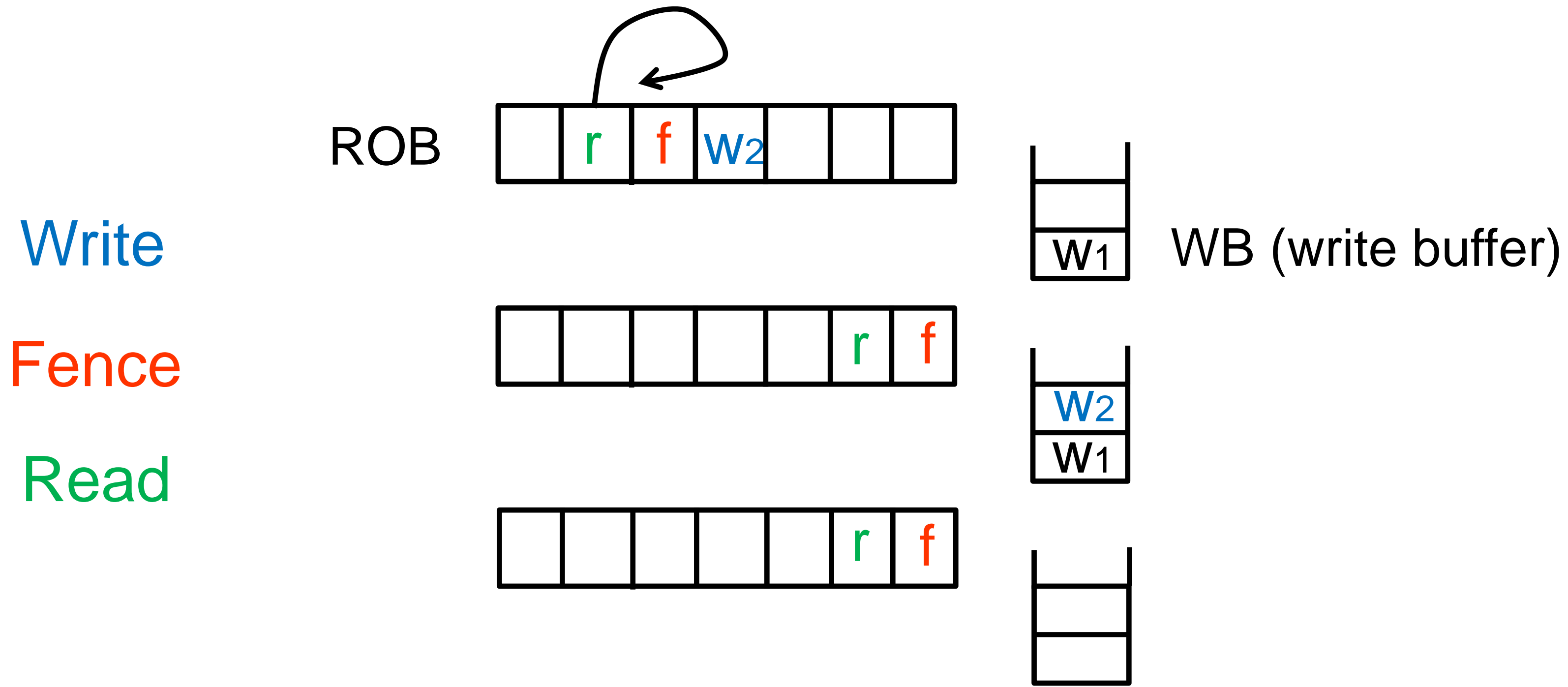
Expensive: cost of a fence in Xeon-based desktop is 20—200 cycles

What if Fences Were Free?

- Programmers could write faster fine-grained concurrent algorithms
- C++/Java programs could guarantee Sequential Consistency (SC) at little performance cost:
  - Programmers would declare all shared variables as atomic
  - Hardware would skip fences while retaining correctness

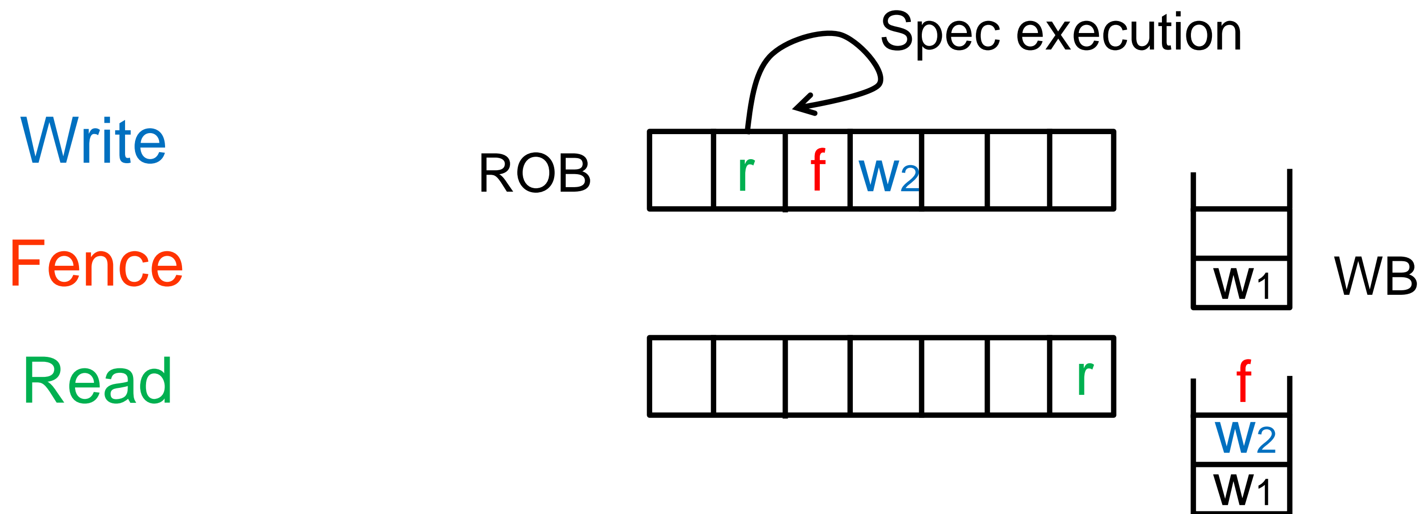
Current Fences Perform Speculation

- Reads following fences can load data speculatively
  - If no processor observes it, no problem
  - If coherence transaction received, squash and retry
- Still: speculative reads cannot retire until the WB is drained



Proposal: WeeFence (or WFence)

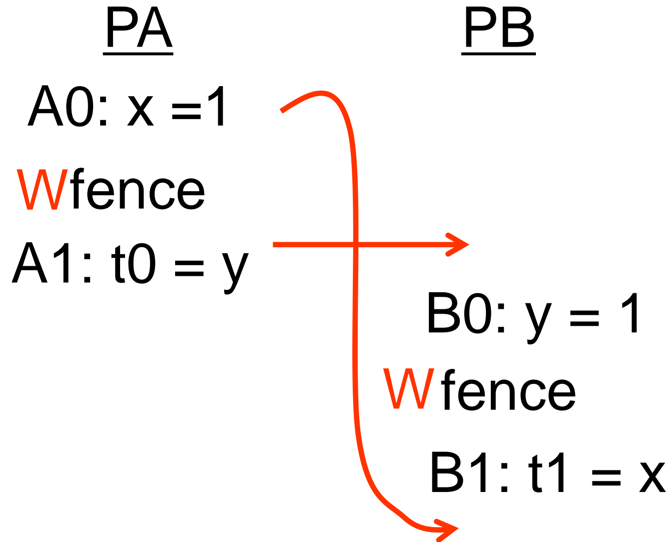
- Eliminate any stall in the pipeline
- Post-fence read retires before the pre-fence writes have drained
  - "Skip" the fence



Substantial gains when write misses pile-up before the fence

But… Reordering Can Cause Incorrect Execution

x = y = 0

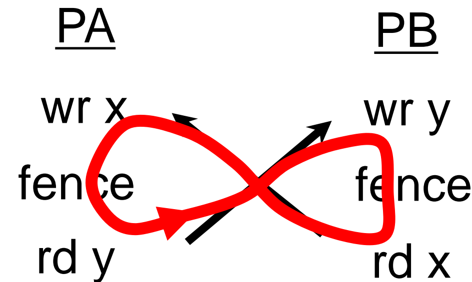


With fences: t0=1 or t1=1 or both=1

With WFences: A1 B0 B1 A0

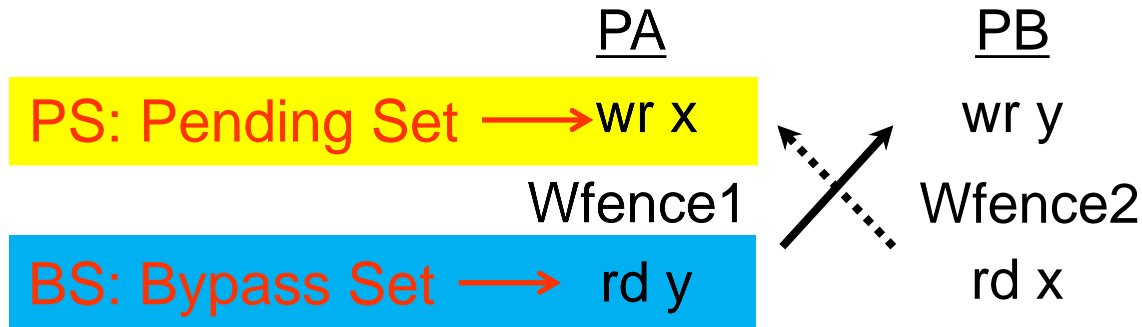
t0 = t1 = 0

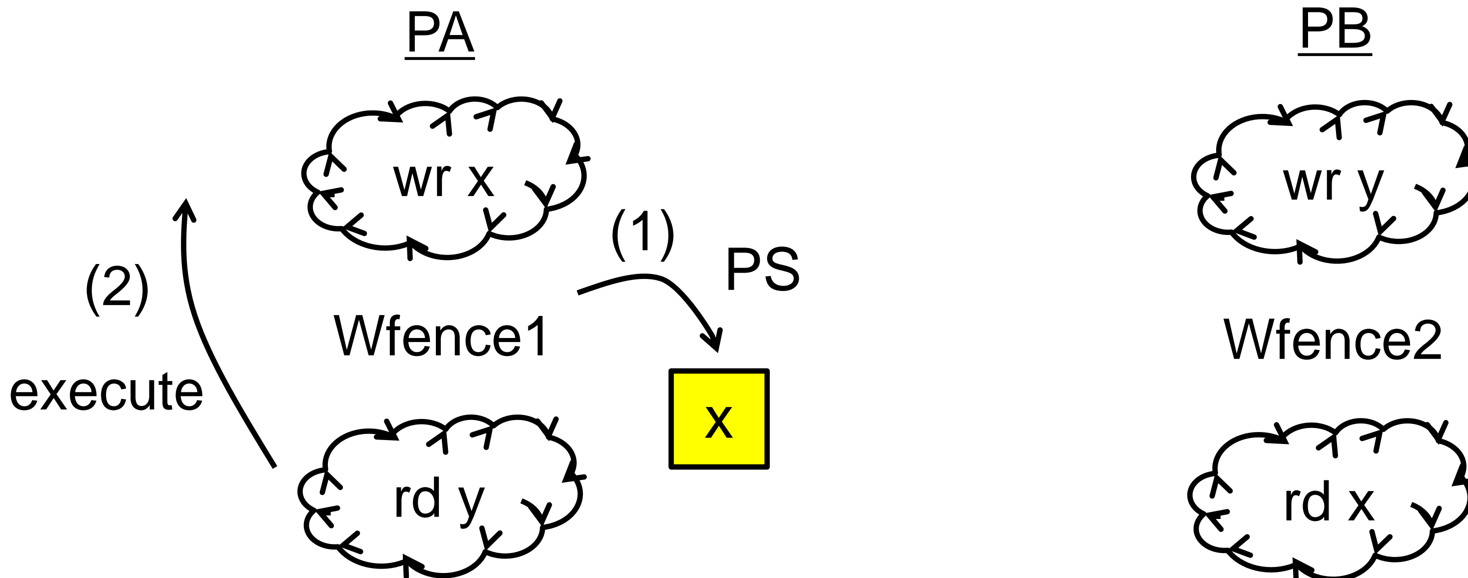
Unintuitive bug:
Sequential Consistency(SC) Violation



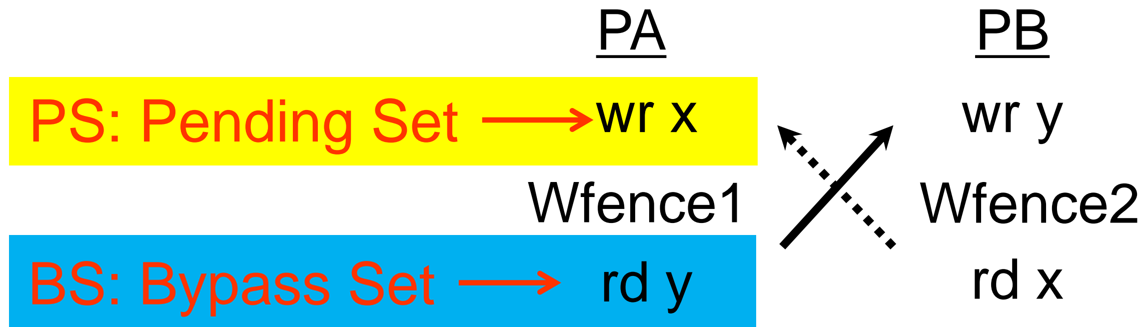
Solution: Stall reads if reordering can cause a dependence cycle

How WFence Works





How WFence Works



PA

PB

wr x

wr y

(1)

(2)

(3)

PS

PS

execute

Wfence1

Wfence2

x

y

x

(5)

local check

stall

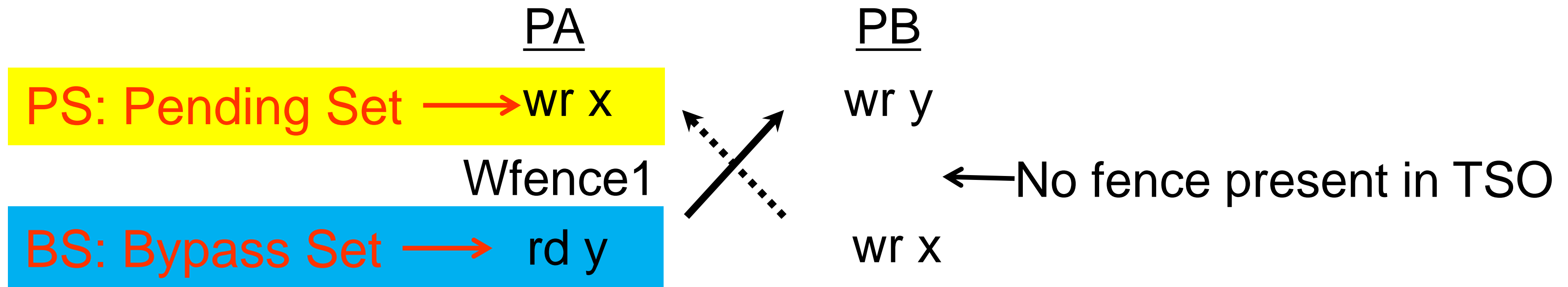
rd y

rd x

Table

(4)

How WFence Works (II)



PA

PB

wr x

wr y

(2)

execute

(1)

PS

Wfence1

x

(3)

rd y

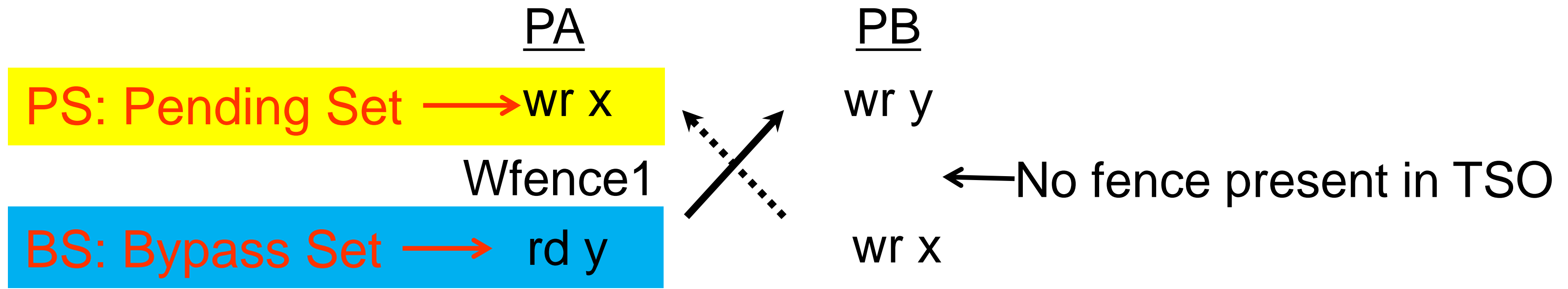
wr x

BS

y

Table

How WFence Works (II)



PA

PB

wr x

wr y

(1)

PS

(2)

execute

Wfence1

(4)

coherence

x

rd y

wr x

(3)

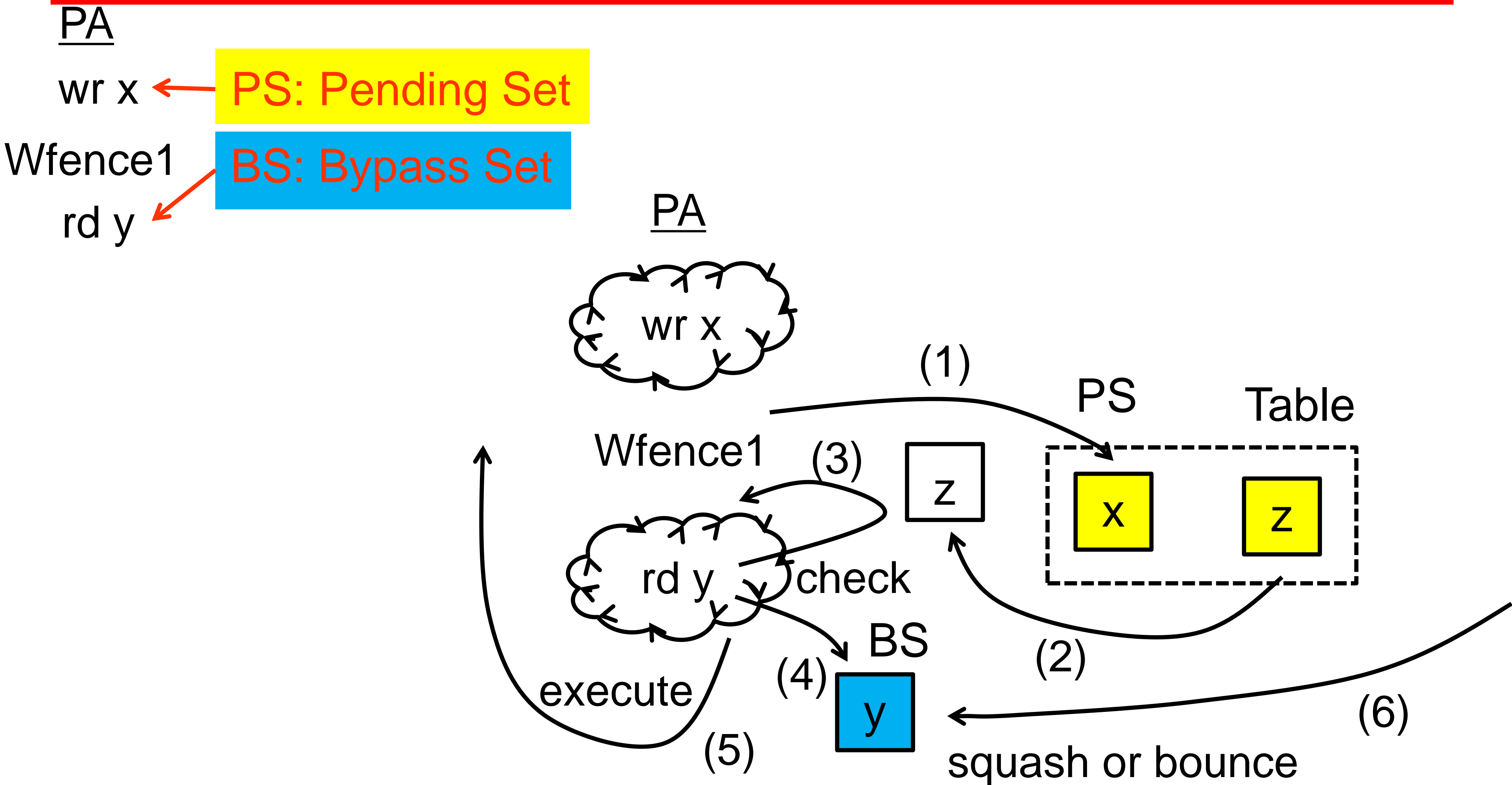
BS

y

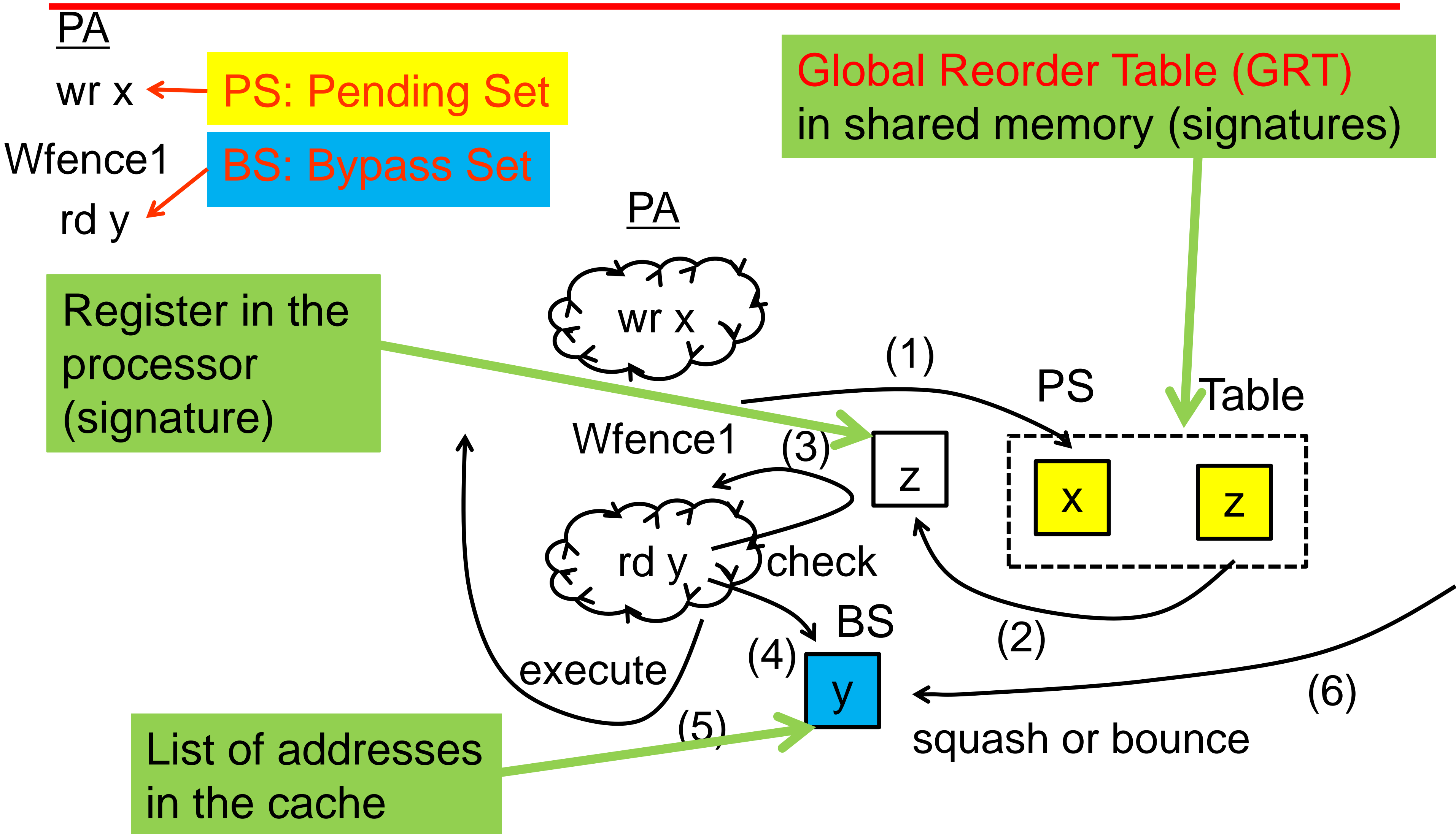
Table

squash or bounce

Summary: How WFence Works

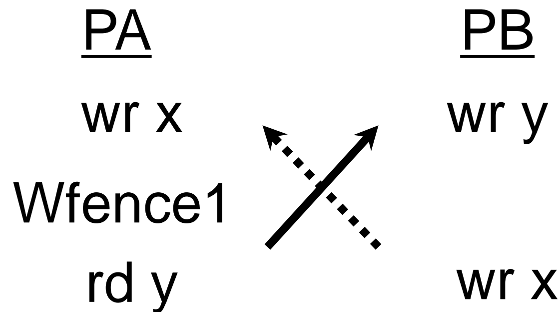


Summary: How WFence Works



WFence

- Cycles are rare: Wfence typically executes without stalling the processor
  - No reordering constraints
- Compatible with conventional fences



- Works with cycles with any number of processors
- No compiler support needed: Off-the shelf executable

Distributed Global Reorder Table (GRT)

Small machine: GRT associated with the bus controller

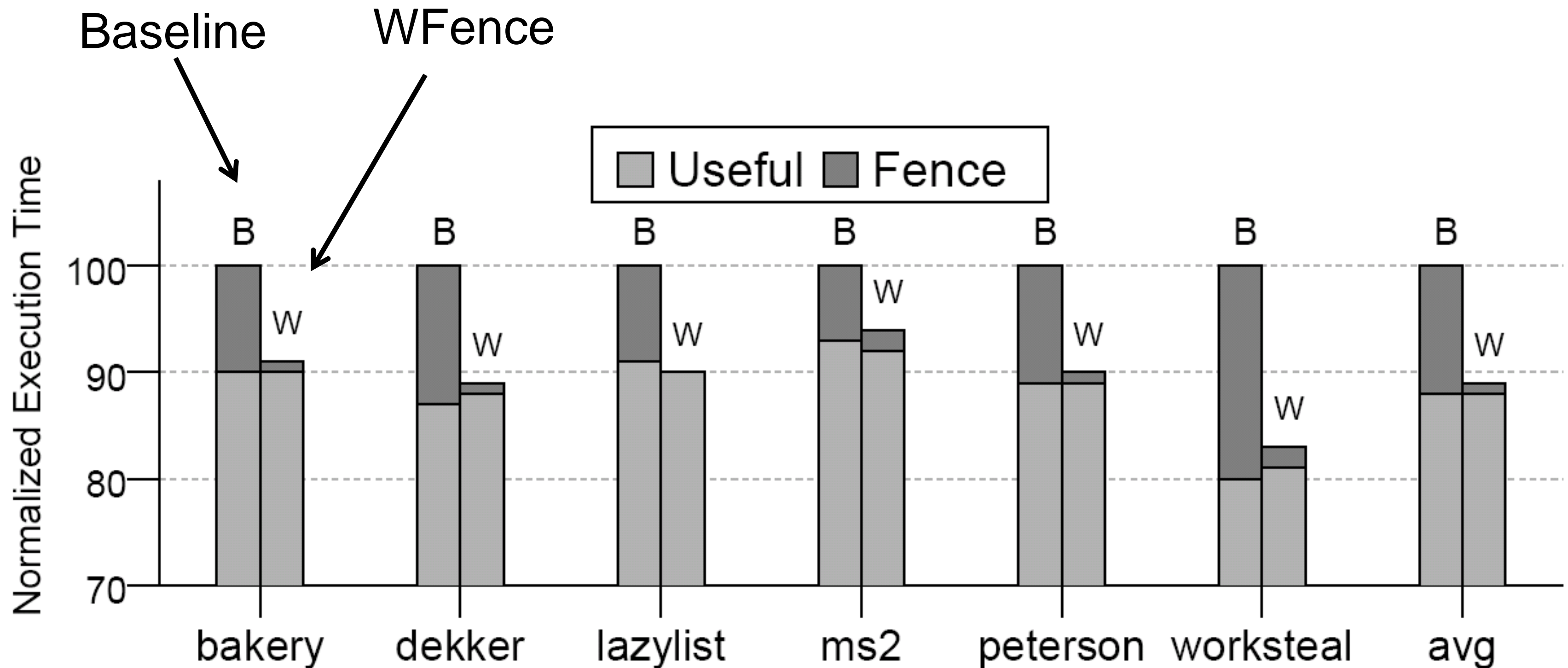
Larger machine: Distributed GRT

- Distribute the GRT like the directory, into modules with address ranges
- WFence works as usual if its PS communicates with single GRT module
  - Most common case due to locality (first-touch page allocation)
- Otherwise, it reverts to a conventional fence
  - Eliminates potential protocol races

Evaluation

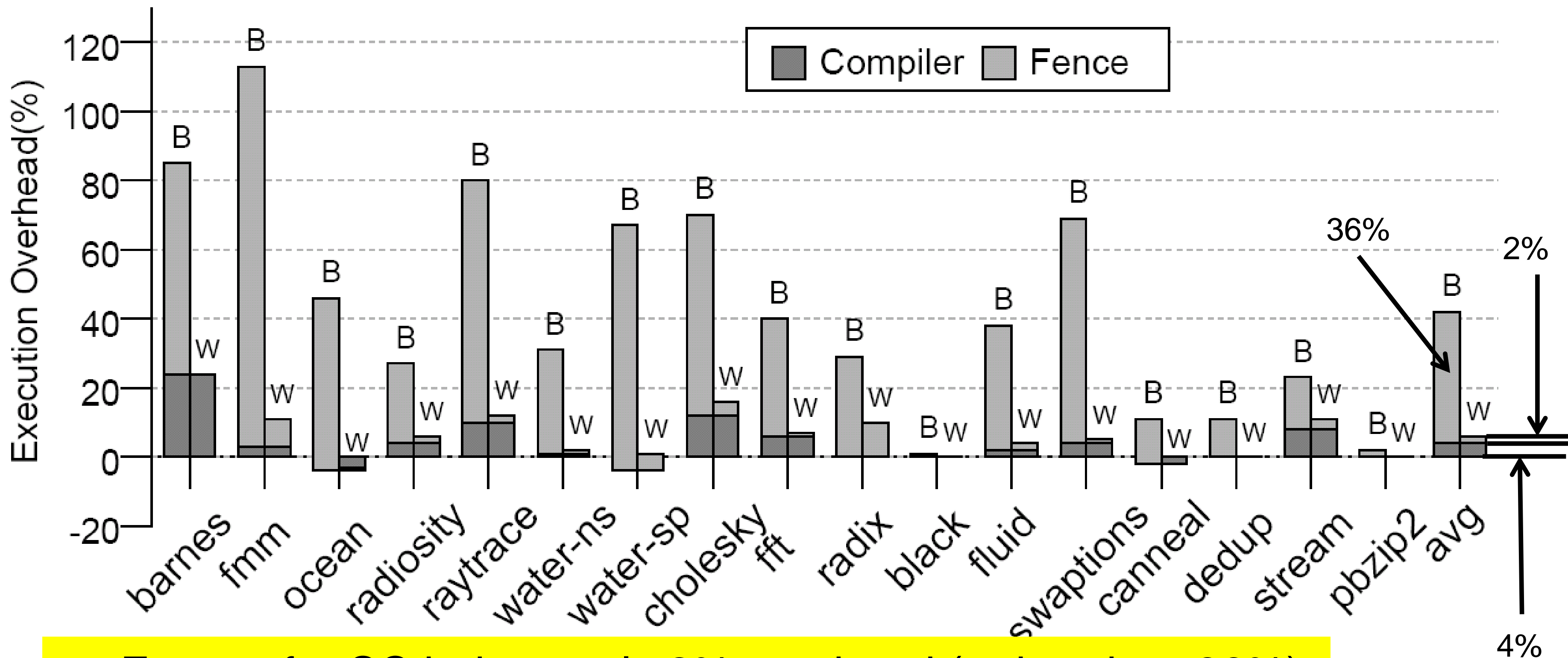
- Simulations of 8-core multicore (centralized and distributed GRT)
- Experiment with kernels (Peterson, Worksteal…)
  - Kernels have explicit fences
  - Goal: Remove all the fence stall time with Wfence
- Experiment with applications (SPLASH-2 and Parsec)
  - A compiler pass conservatively inserts fences to guarantee SC
  - Goal: The resulting fences, if implemented with WFence, induce negligible overhead

Kernels with Fences: Execution Time



WFence eliminates most of the fence stall time (11% execution reduction)

Applications with Fences for SC: Execution Overhead



- Fences for SC induce only 2% overhead (rather than 36%)
- Disabled compiler optimizations add an additional 4%
- With distributed GRT, 2% additional overhead

Conclusions

- Today's fences are expensive. If they were free
  - Programmers could write faster fine-grained algorithms
  - C++/Java compilers could guarantee SC at little cost
- WFence:
  - Executes without stalling the processor (cycles are rare)
  - Compatible with conventional fences
  - No compiler support needed: off-the-shelf executable
  - Effective:
    - Eliminates fence stall from kernels (11% execution reduction)
    - Supports SC in applications with only 2% overhead
- Our Future Work: Show how WFence can help parallel programming

WeeFence: Toward Making Fences Free in TSO

Yuelu Duan, Abdullah Muzahid, Josep Torrellas
Department of Computer Science
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

ISCA, June 2013

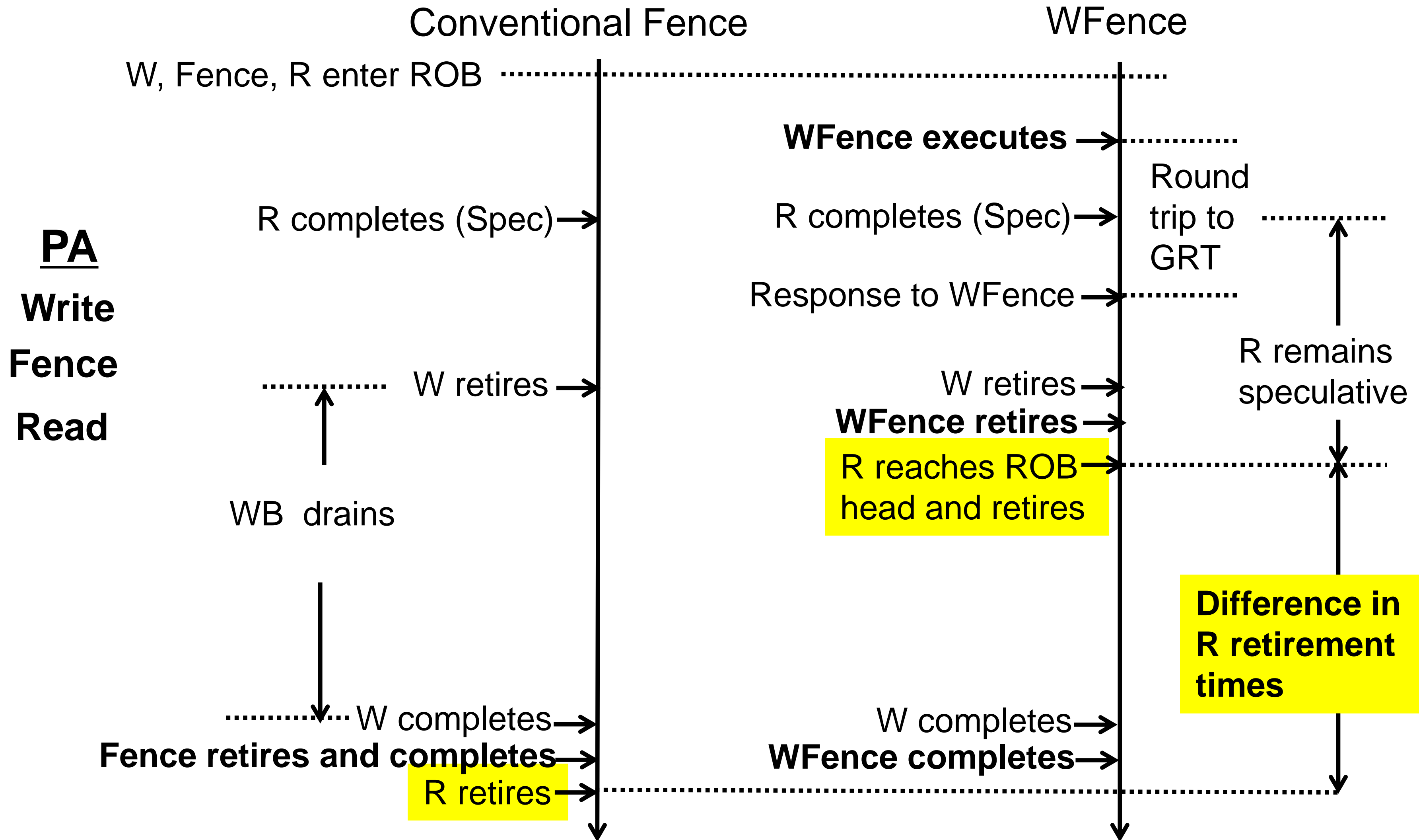
i-acoma group

ILLINOIS

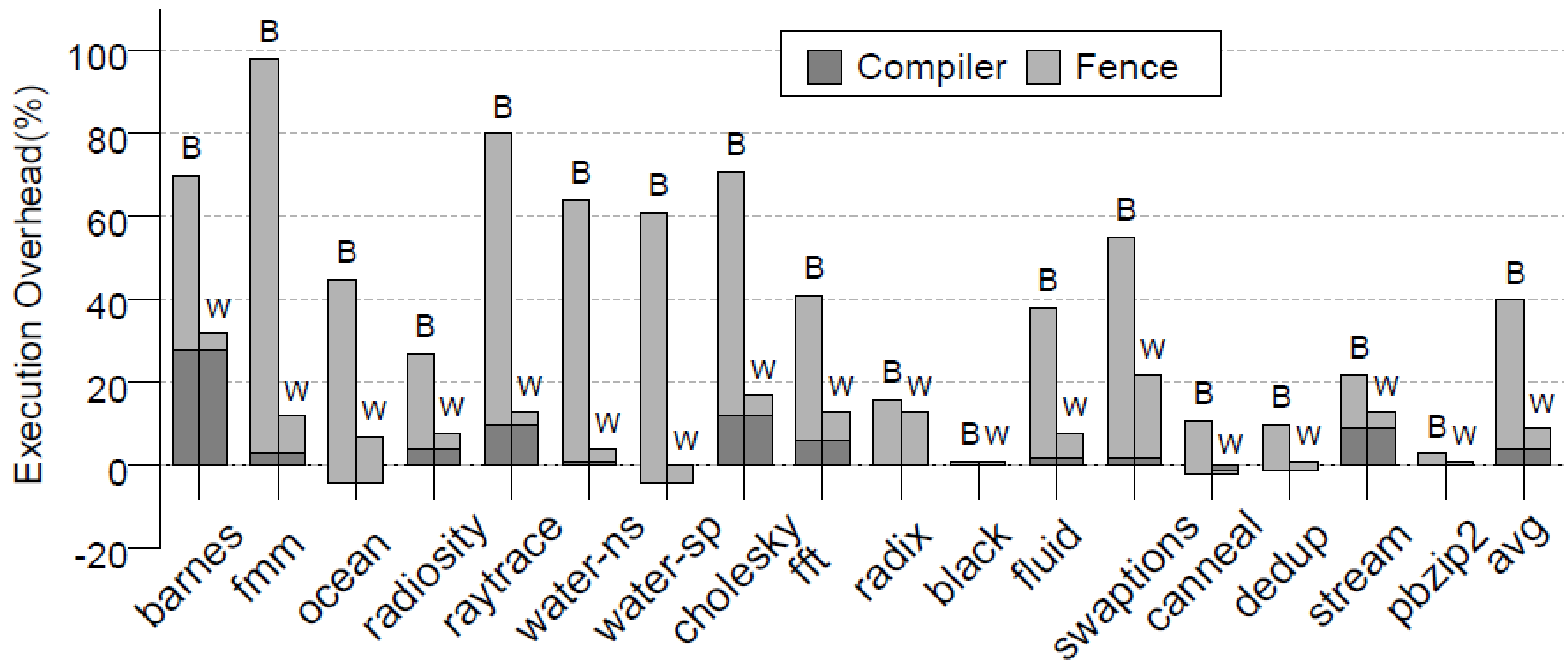
Also in the Paper

- Show that deadlock is not possible
- Value forwarding
- Multiple WFences per thread
- Application to release consistency
- Detailed characterization of WFences
- Scalability with the number of processors

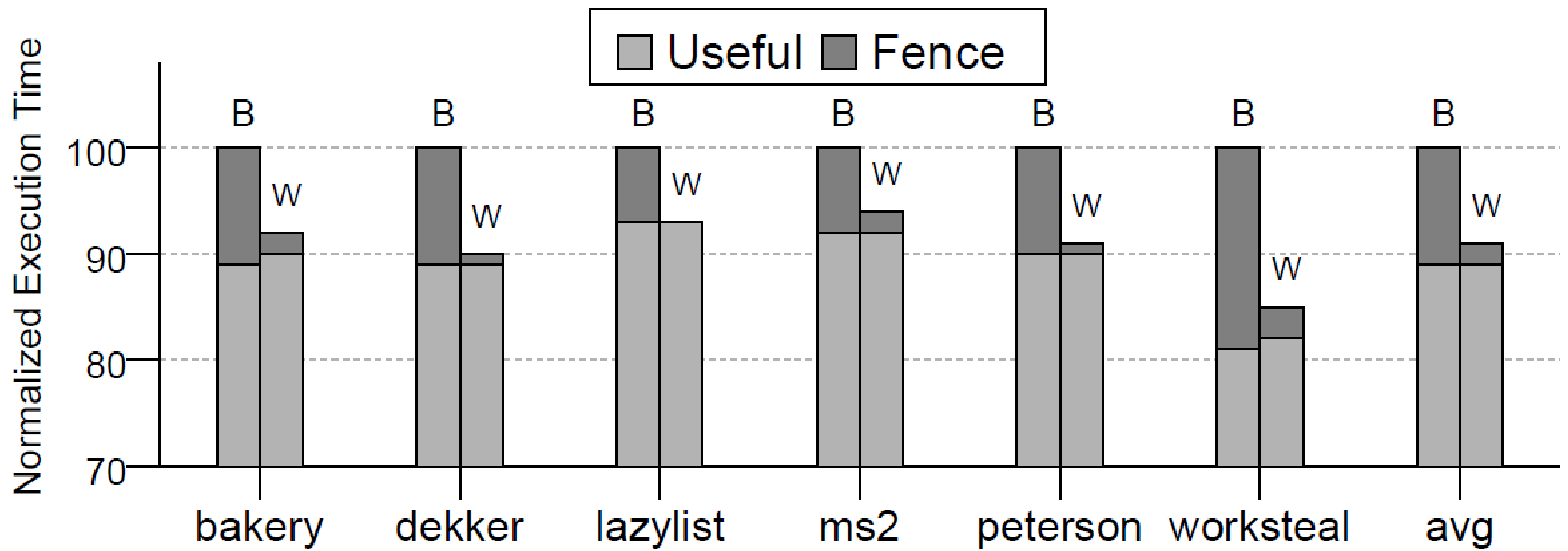
Timeline



Execution Overhead: Distributed GRT



Distributed GRT



Intel Processor