Asymmetric Memory Fences: Optimizing Performance & Programmability

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

ASPLOS, March 2015

Fence: a (Slow) Primitive for Parallelism

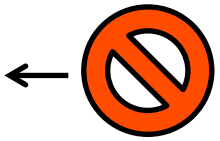
- Instruction inserted by programmers or compilers
- Prevents the compiler and HW from reordering memory accesses

rd x
wr y

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

fence

rd z



Cannot be observed by another processor

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

There are HW proposals to eliminate fence stall: WeeFence [ISCA13]…

… but they need complex HW

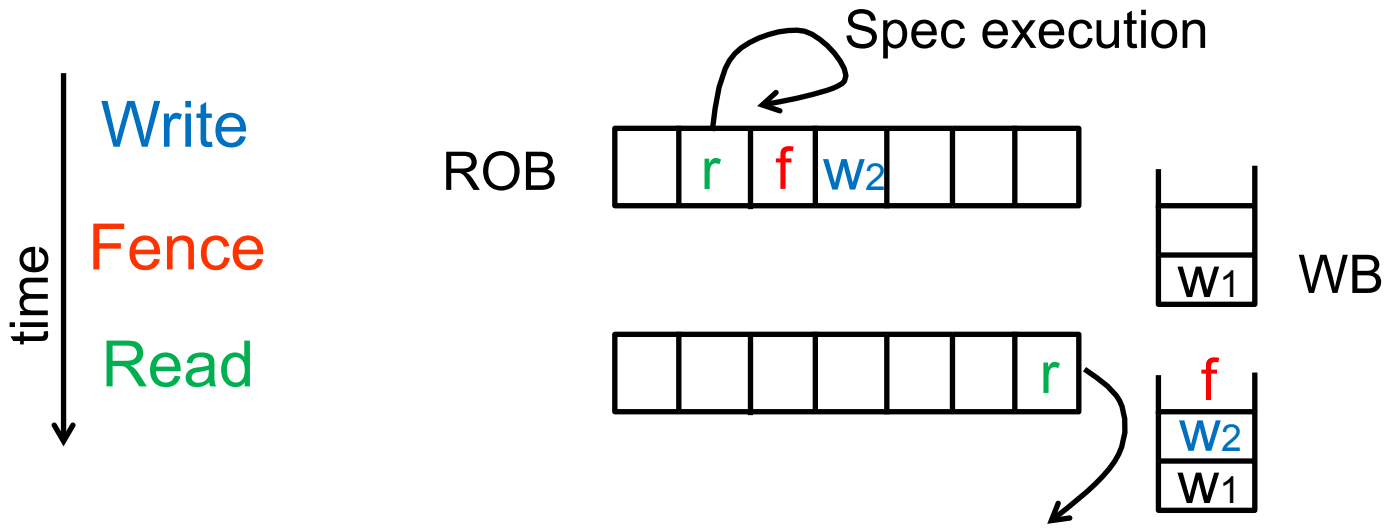
Contributions

- Asymmetric Fence Groups: eliminate fence stall with simple HW
  - Weak Fence (wF): aggressive access reordering
  - Strong Fence (sF): conventional
- Best fence performance-cost tradeoff yet
- Taxonomy of Asymmetric Fence Groups under TSO

Past Work: *WeeFence*

[ISCA 2013]

- Aggressive access reorder to eliminate pipeline stall
- Post-fence reads retire 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

PA

A0: x =1

Weefence

A1: t0 = y

PB

B0: y = 1

Weefence

B1: t1 = x

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

With WeeFences: A1

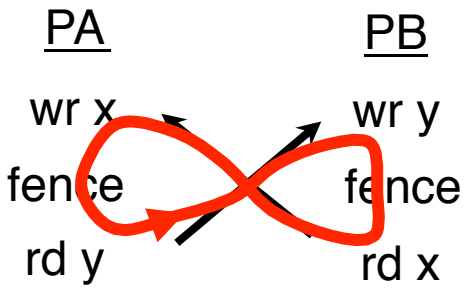
B0

B1

A0

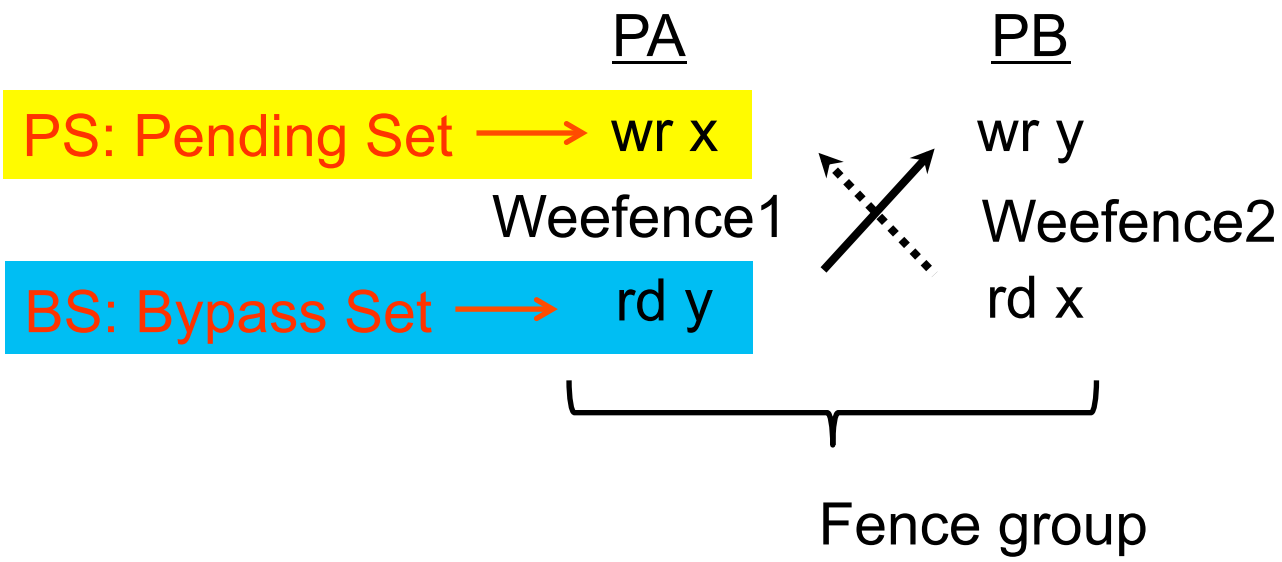
t0 = t1 = 0

Sequential Consistency(SC) Violation

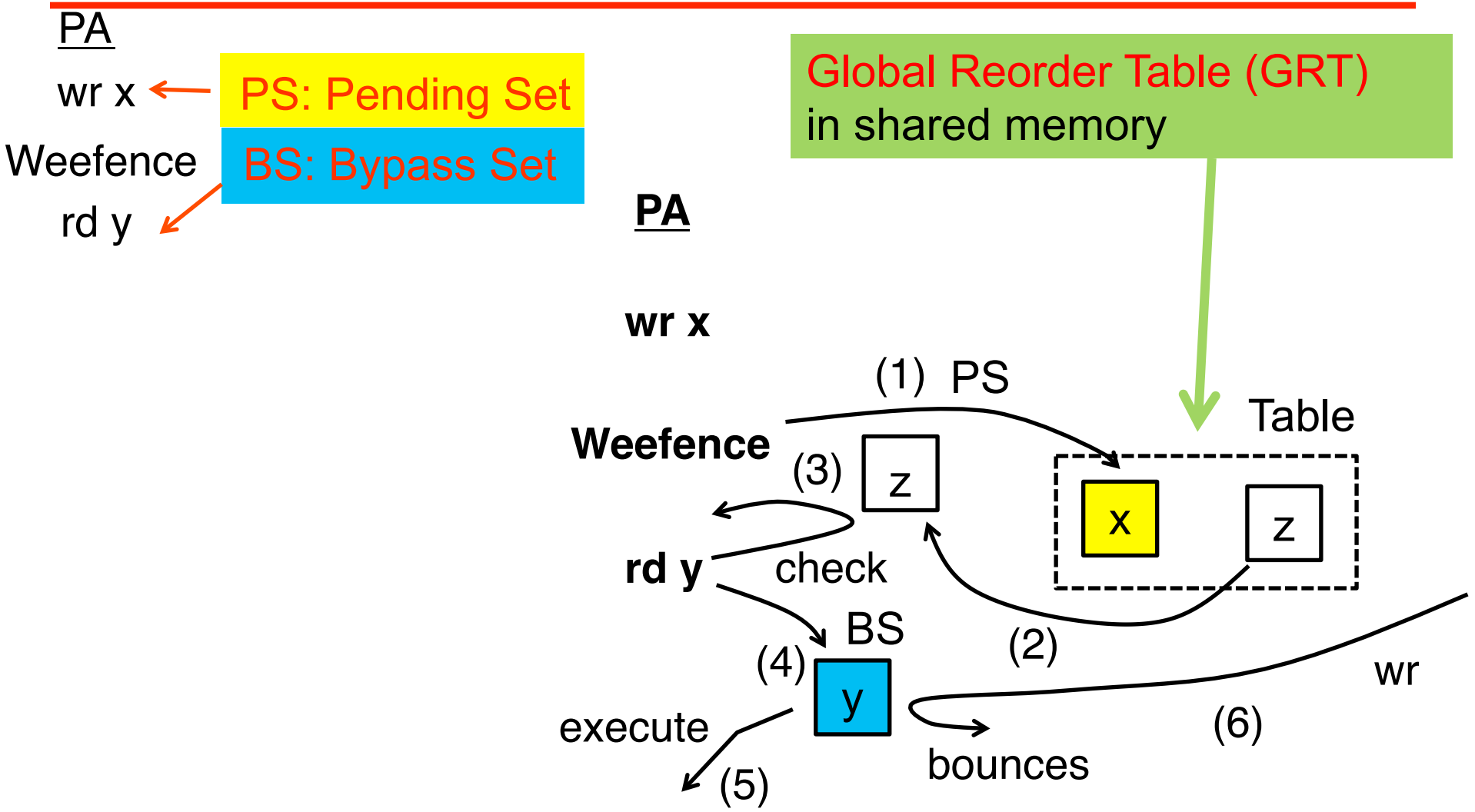


Solution: WeeFence stall rds/wrs if reordering may cause a cycle

How WeeFence Works



How WeeFence Works



WeeFence Hardware is Suboptimal

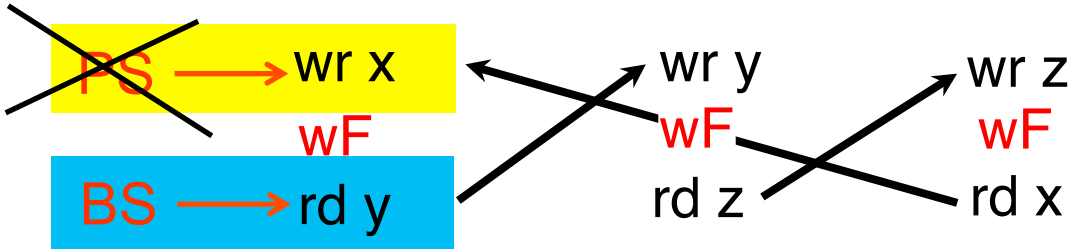
- Requires GRT global hardware and additional messages

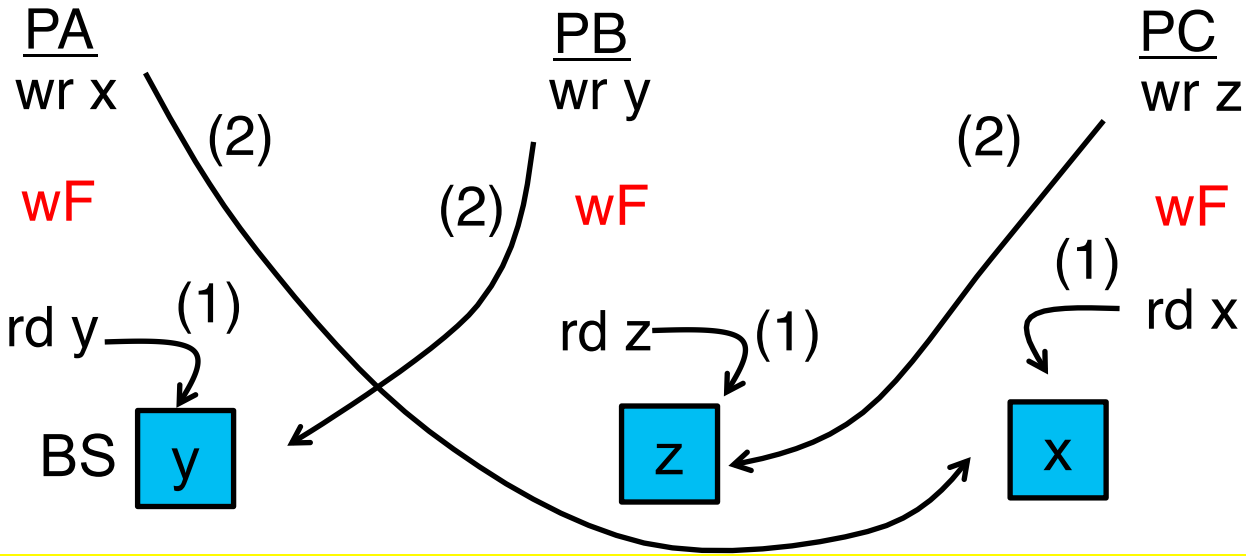
- GRT is hard to distribute like the directory
  - Creates coherence protocol races
  - If PS maps to multiple directory modules →turns into conventional fence

Our goal: High performance and simpler hardware

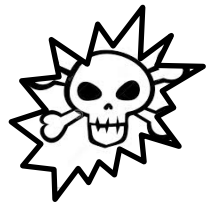
Weak Fence (wF)

- Reordering capabilities of WeeFence + no global state



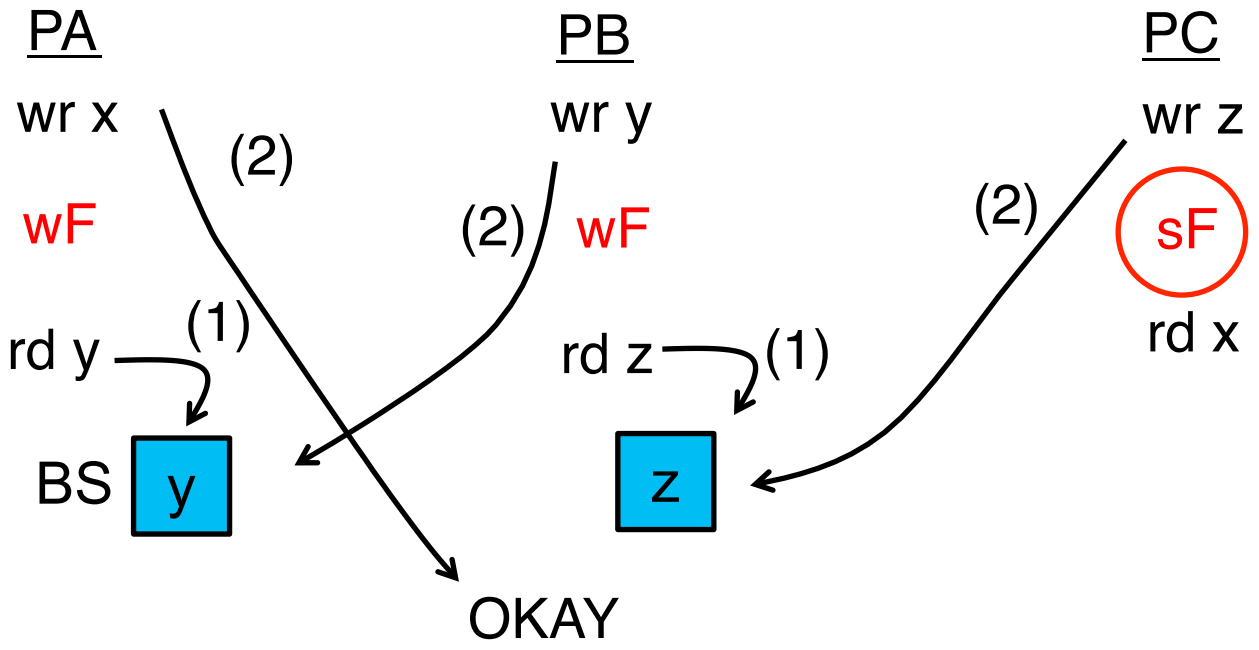


If all the fences in a Fence Group are wF → Deadlock



Insight: No Deadlock If At Least One Conventional Fence

- Strong Fence (sF): Conventional fence
- If the group has at least one sF ➔ no deadlock or SC violation



Asymmetric Fence Group

- Contains one or more wF and one or more sF
- Much simpler HW than WeeFence: no global state
- Similar performance (or higher) than WeeFence:
  - Insight: In a fence group, some threads more critical than others
  - Use wF for the critical threads and sF for others

Best fence performance-cost trade-off yet

Use of Asymmetric Fences: Work Stealing

take()	steal()
Tail =	Head =
wF	sF
= Head	= Tail

- Cilk, TBB and other runtime schedulers
- steal() < 5%

Use of Asymmetric Fences: Software Transactional Memory (STM)

read(M)	write(M)
Lock(M).readers =	Lock(M).writer =
wF	sF
= Lock(M).writer	= Lock(M).readers

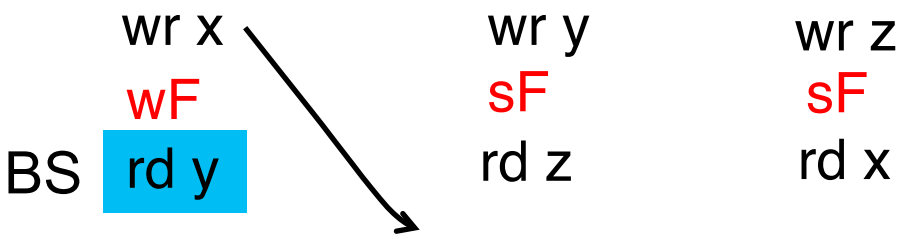
- Read and Write Barriers in STM:
  - Perform the requested rd/wr
  - Update STM metadata to ensure transaction serialization
- Reads are 3.5x more frequent than writes

Taxonomy of Asymmetric Fence Groups under TSO

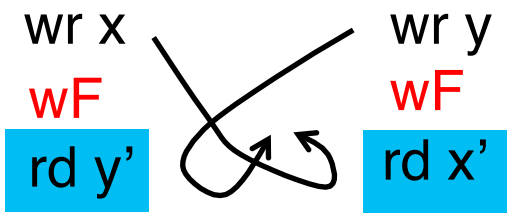
S+	Fence group only contains sFs	sF sF sF
WS+	Asymmetric group with at most one wF	wF sF sF
SW+	Any Asymmetric group	wF sF wF
W+	Group with only wFs	wF wF wF

They trade off hardware cost for performance

WS+: Asymmetric Groups with at Most one wF



- Pre-wF accesses never bounce-off from another processor's BS
- Addresses and checks always at line level (as conventional protocol)
- False sharing may create bouncing



Write proceeds (no SC violation possible)

Other processor kept as sharer so that it sees future coherence activity

Supported with the Order bit (see paper)

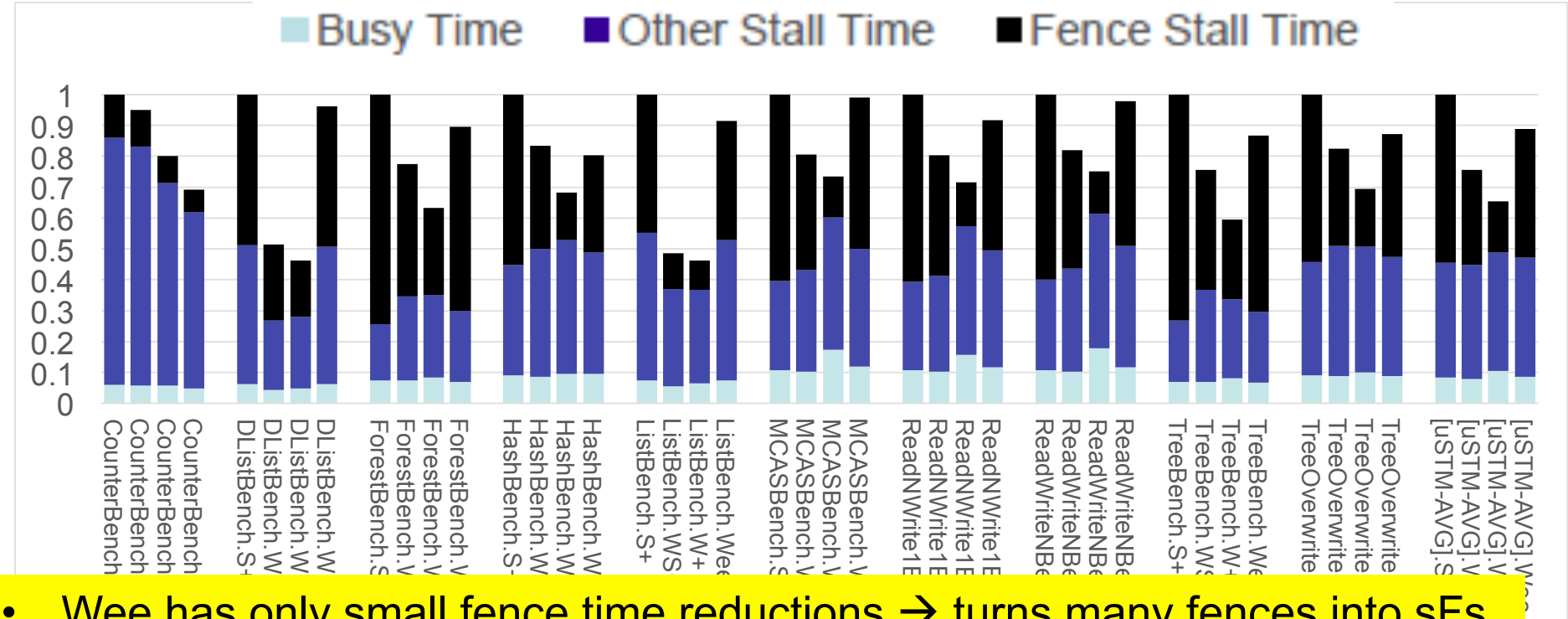
W+: Groups with only wFs

- Can deadlock under unfavorable timing
  - True cycle or due to false sharing
- Insight: Under TSO only, recovery is not too costly
  - Completed accesses are only reads
  - No speculative writes
- When wF reaches ROB head → checkpoint and proceed
- If HW detects bouncing and being bounced → timeout
  - Rollback
  - Wait for write buffer to drain (no deadlock again)

Evaluation

- Simulations of 8-core multicore
- Workloads:
  - 10 Cilk apps using the THE work-stealing algorithm
  - 10 Software Transactional Memory (STM) kernels
  - 6 STAMP apps that use STM
- Goal: Speed-up execution
- Compare execution time of WS+ and W+ to:
  - S+: conventional fences
  - WeeFence

Per-Transaction Execution Time of STM Kernels



- Wee has only small fence time reductions → turns many fences into sFs
- WS+ and W+:
  - Eliminate avg. 1/2 and 2/3 of the fence stall time
  - Avg. transaction takes 24% and 35% fewer cycles
- Choice WS+ vs W+: Order bit for false sharing vs hardware timeouts

Conclusions

- Asymmetric Fences for performance and simple HW:
  - Weak Fence (wF)
  - Strong Fence (sF)
- Best fence performance-cost trade-off yet
  - Simpler HW than WeeFence: no global hardware
  - Higher perf. than WeeFence: 13% (WS+) and 21% (W+) avg.
- Taxonomy of Asymmetric fence groups under TSO
- Outlined uses
- Future Work: programmability issues

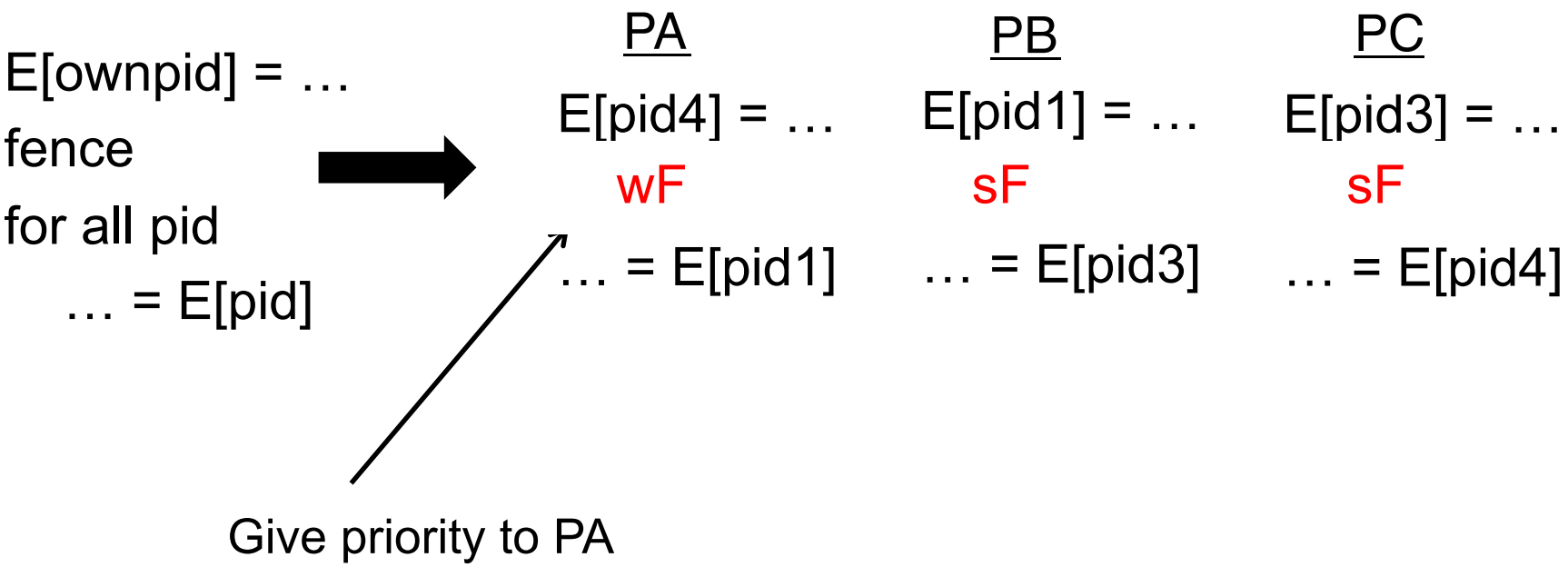
Asymmetric Memory Fences: Optimizing Performance & Programmability

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

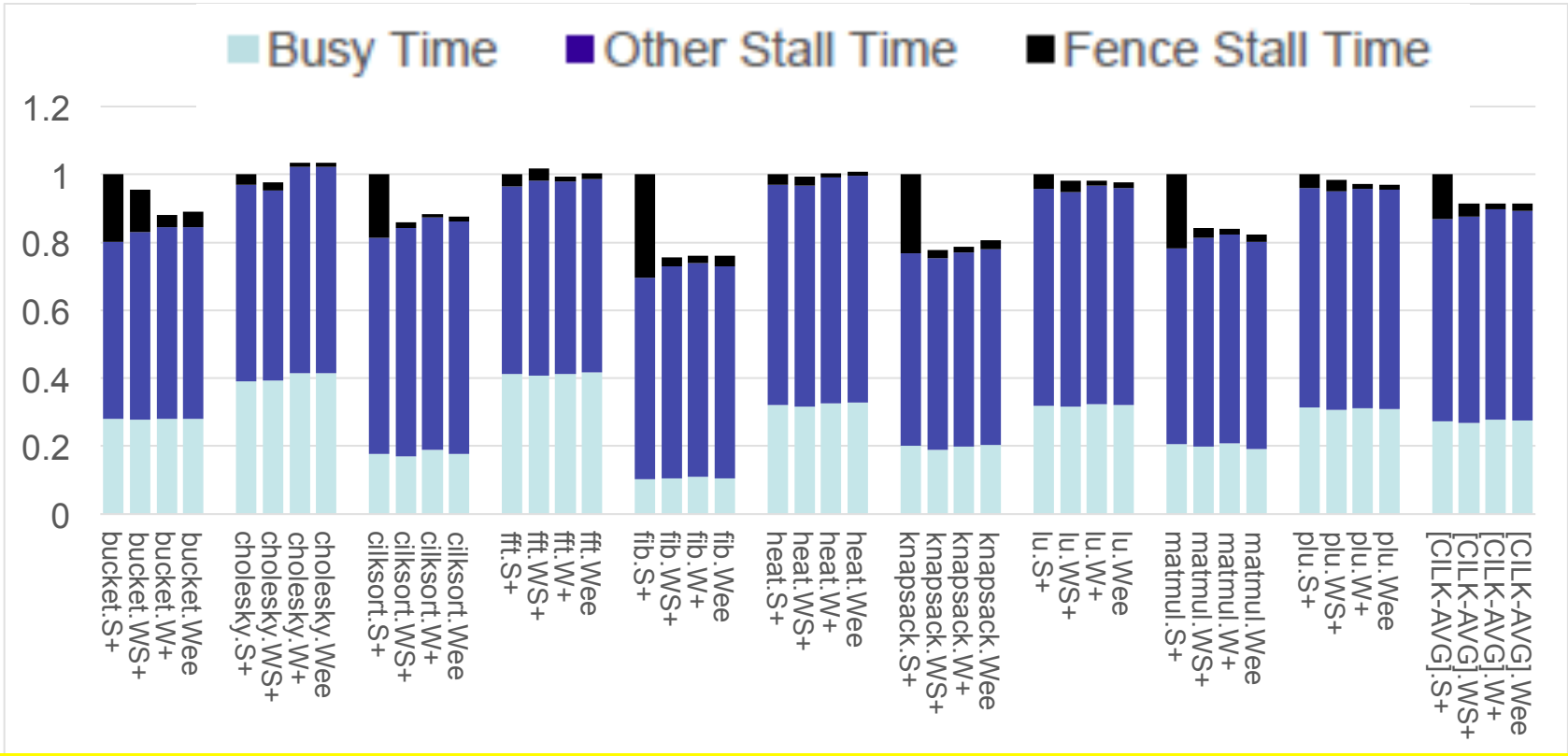
ASPLOS, March 2014

Use of Asymmetric Fences: Bakery Algorithm

Fence groups of many different sizes



Execution Time of Cilk Apps

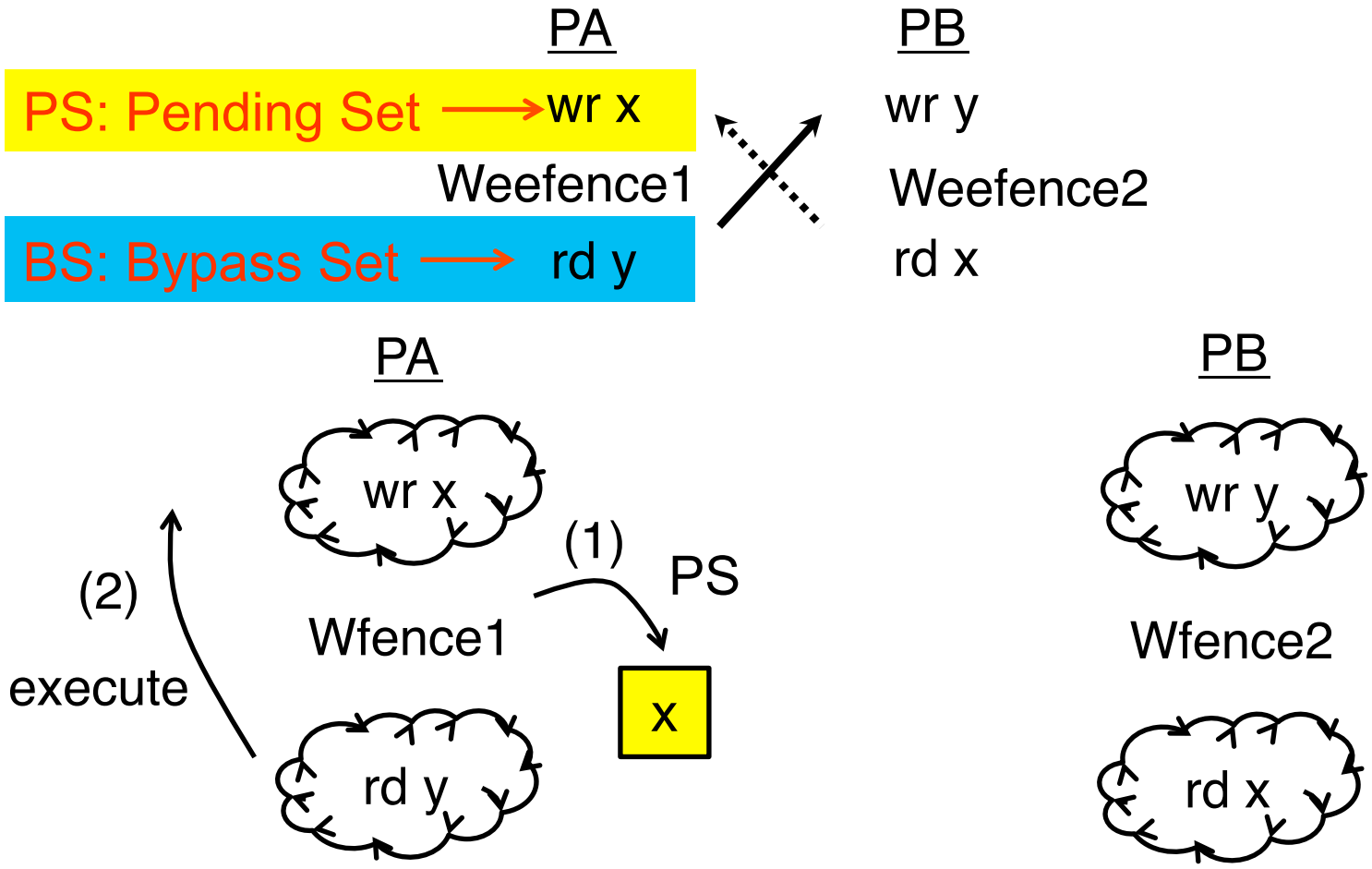


- WS+ and W+:
  - Eliminate most of the fence stall time
  - Similar impact as Wee, which is more expensive
- Overall execution time reduced by avg. 9%

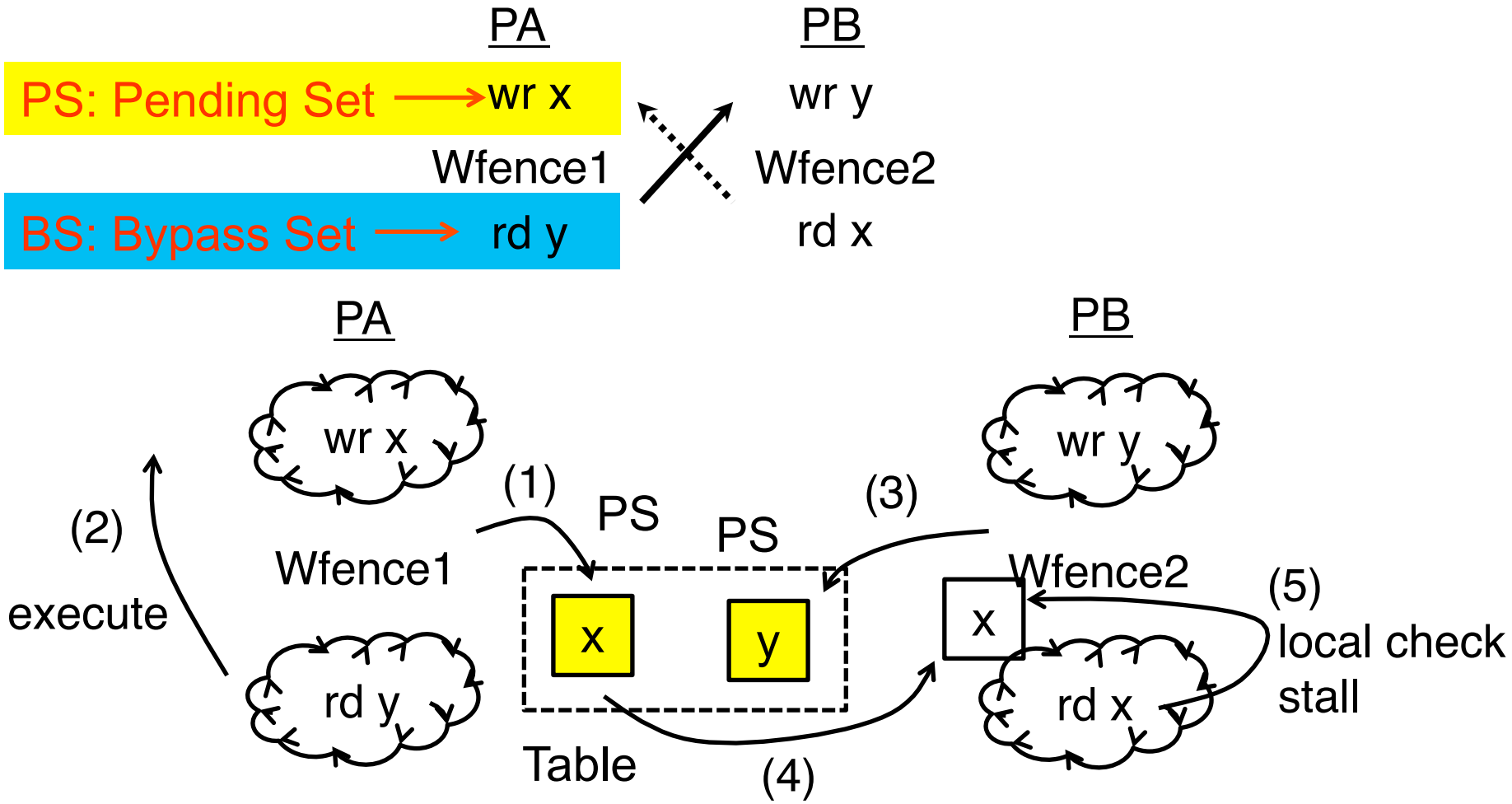
Also in the Paper

- Description of taxonomy in detail
- Hardware implementation issues
  - Line displacements, RC…
- Outline interesting programming issues
- Evaluation of STAMP benchmarks
- Scalability to 32 threads

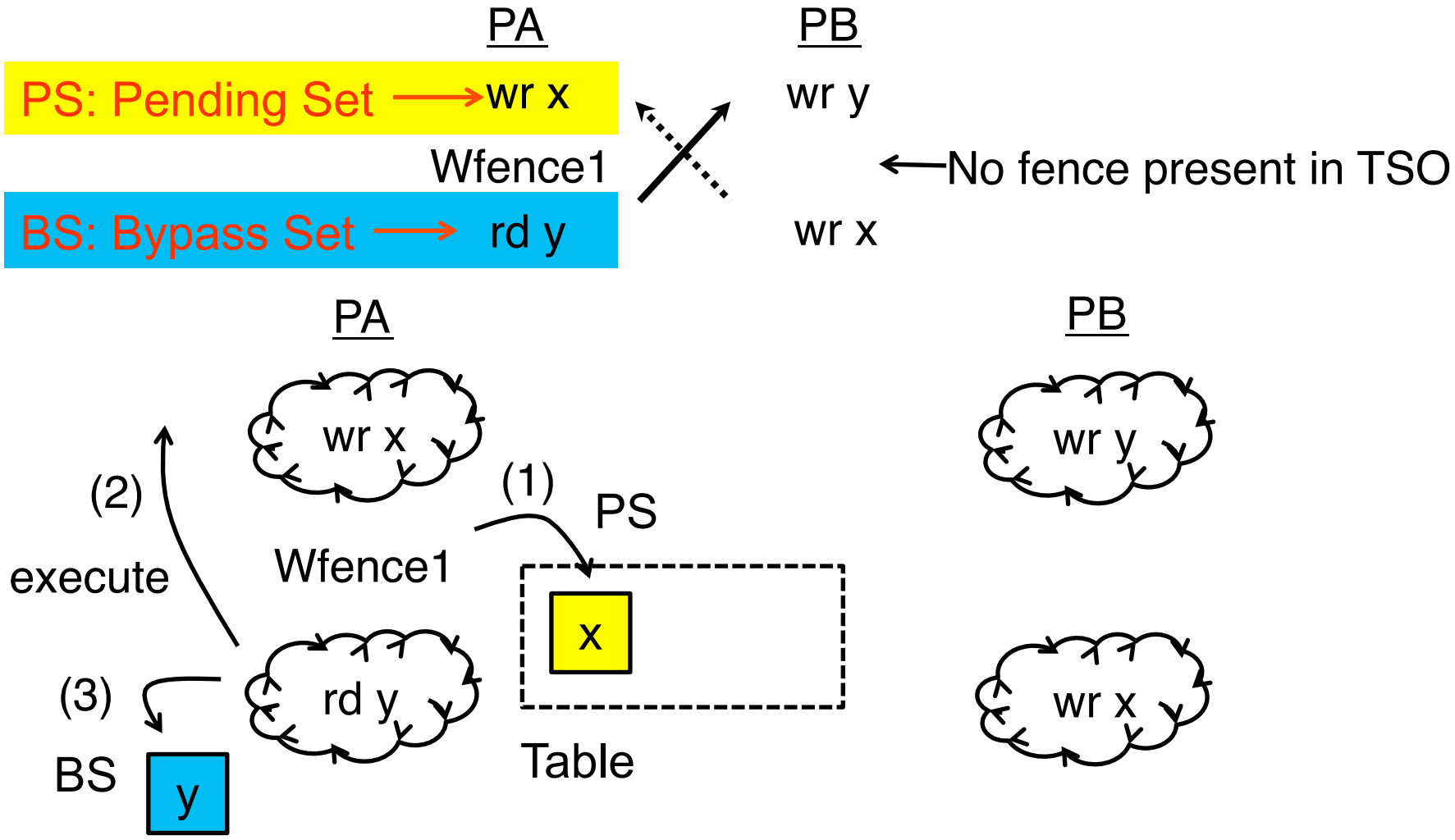
How WeeFence Works



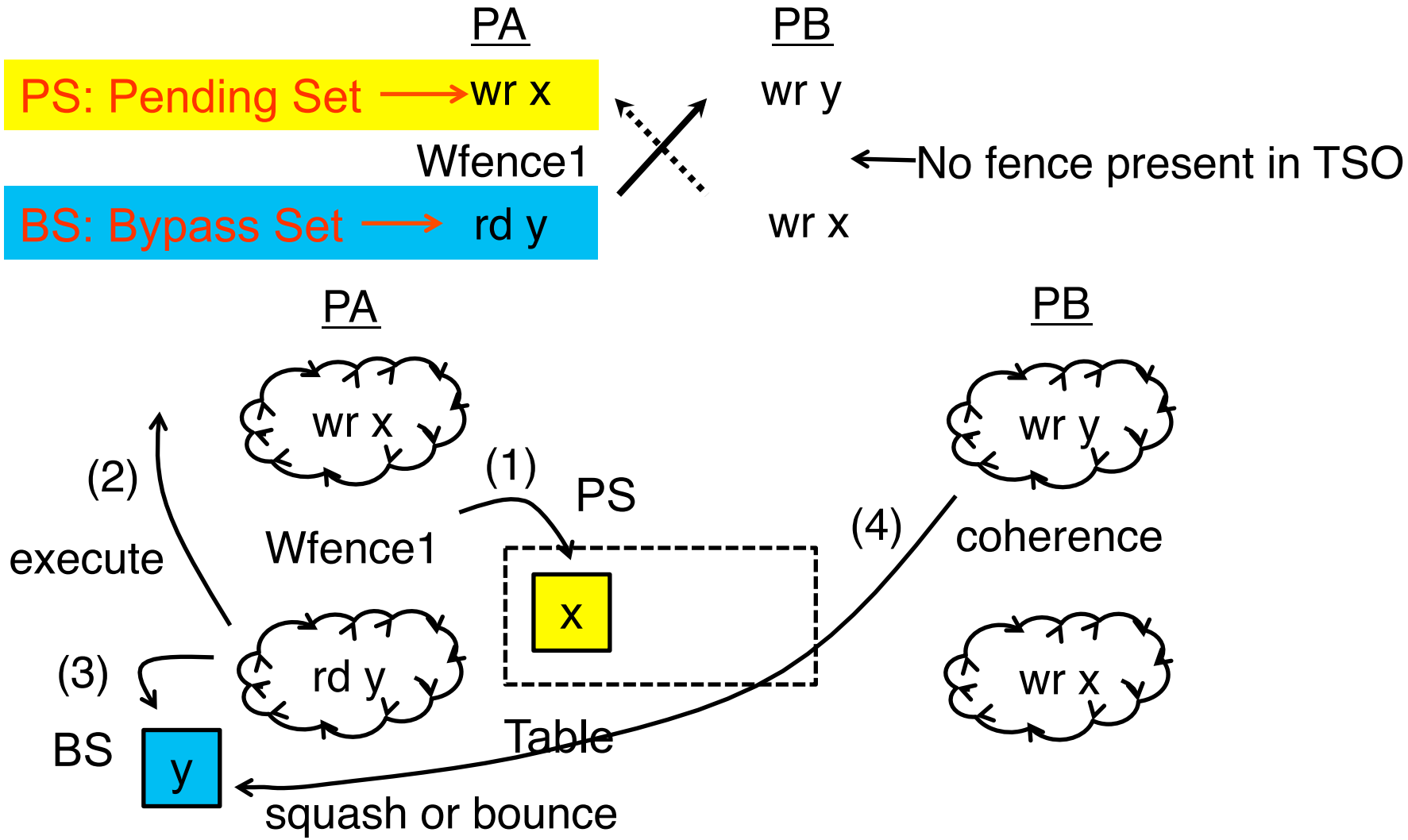
How WFence Works



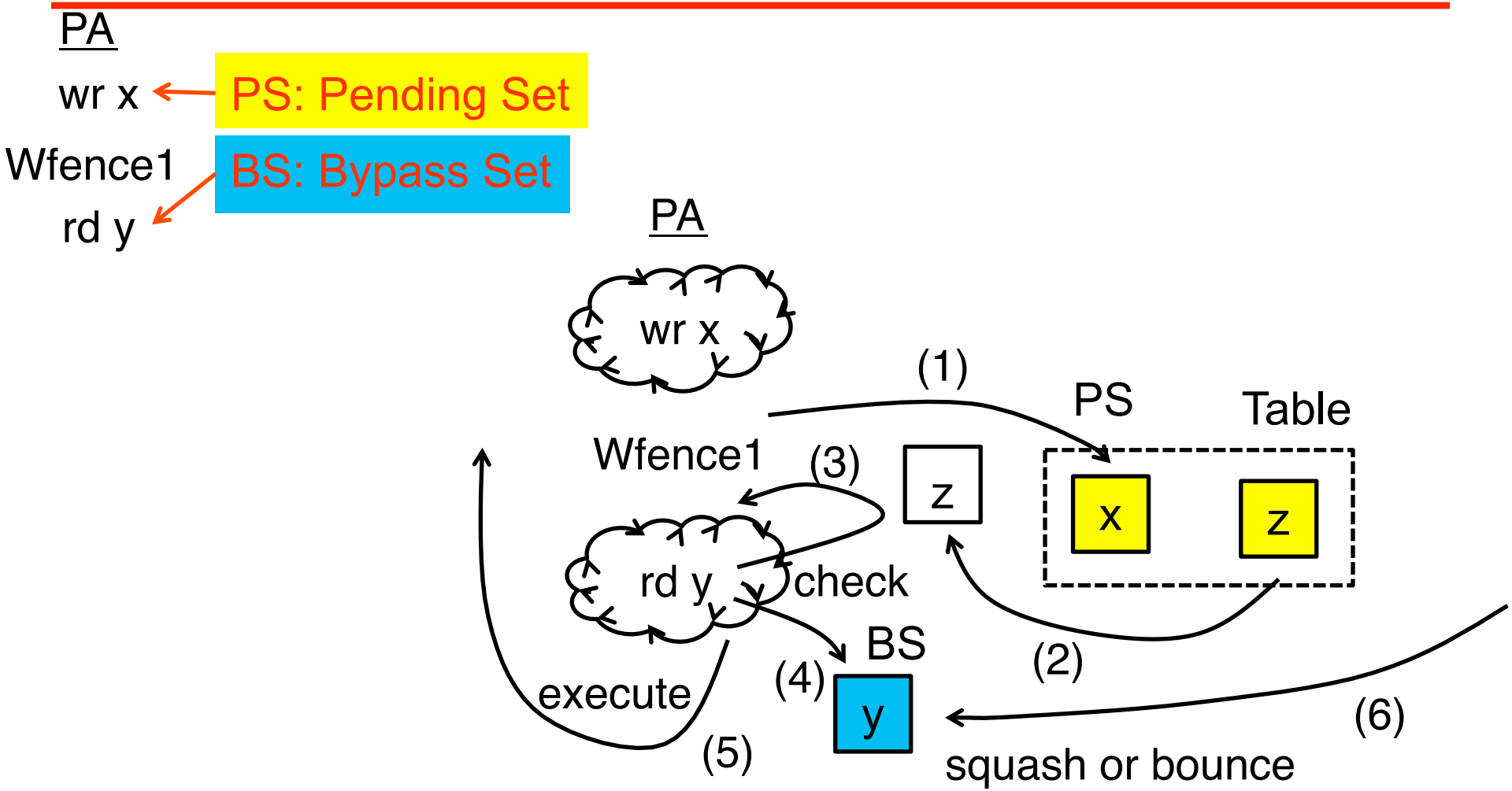
How WFence Works (II)



How WFence Works (II)



Summary: How WFence Works



Summary: How WFence Works