BulkSMT:
Designing SMT Processors for Atomic-Block Execution

Xuehai Qian, Benjamin Sahelices and Josep Torrellas
University of Illinois

University
of Illinois

http://iacoma.cs.uiuc.edu/

i-acoma group

Motivation

- Architectures that continuously execute Atomic Blocks
  - Performance and programmability advantages [Hammond 04], [Ahn 10]
  - All proposals use single context cores

- What if we used Simultaneous Multithreading (SMT) cores?
  - Enables a better utilization of the hardware
  - Fast communication between local contexts
    - Enables higher-concurrency forms of atomic block execution

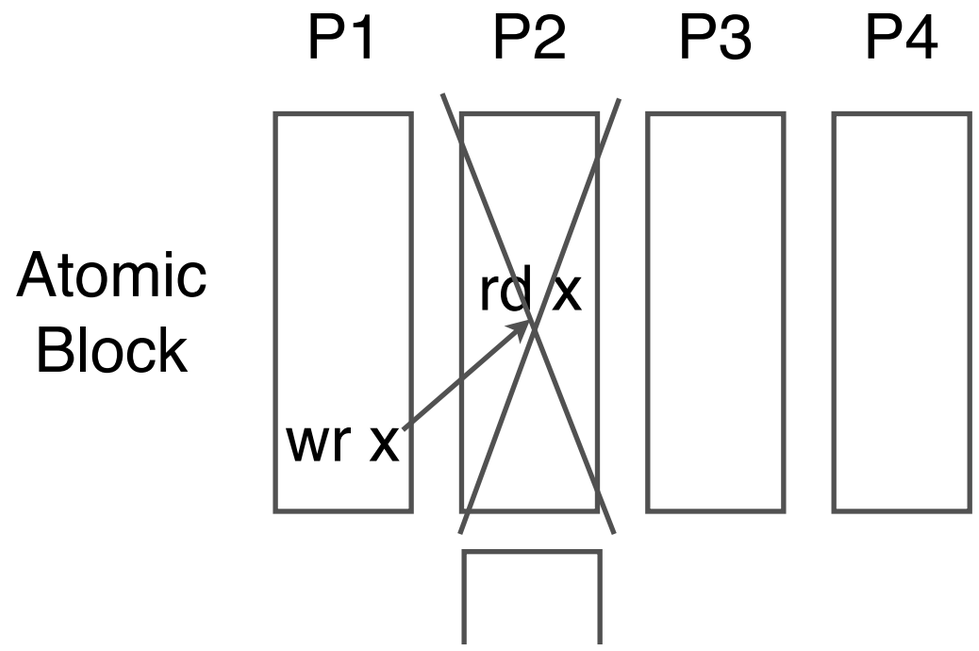
Contributions

- BulkSMT: first SMT design with atomic-block (transactional) execution
  - Enables concurrency between dependent blocks
- Analysis of design space:
  - SQUASH on conflict
  - STALL on conflict
  - ORDER on conflict
- Design of a multicore of BulkSMTs
- BulkSMT is cost effective
  - Higher performance for the same core count
  - Comparable performance for 1/4 of the cores

Outline

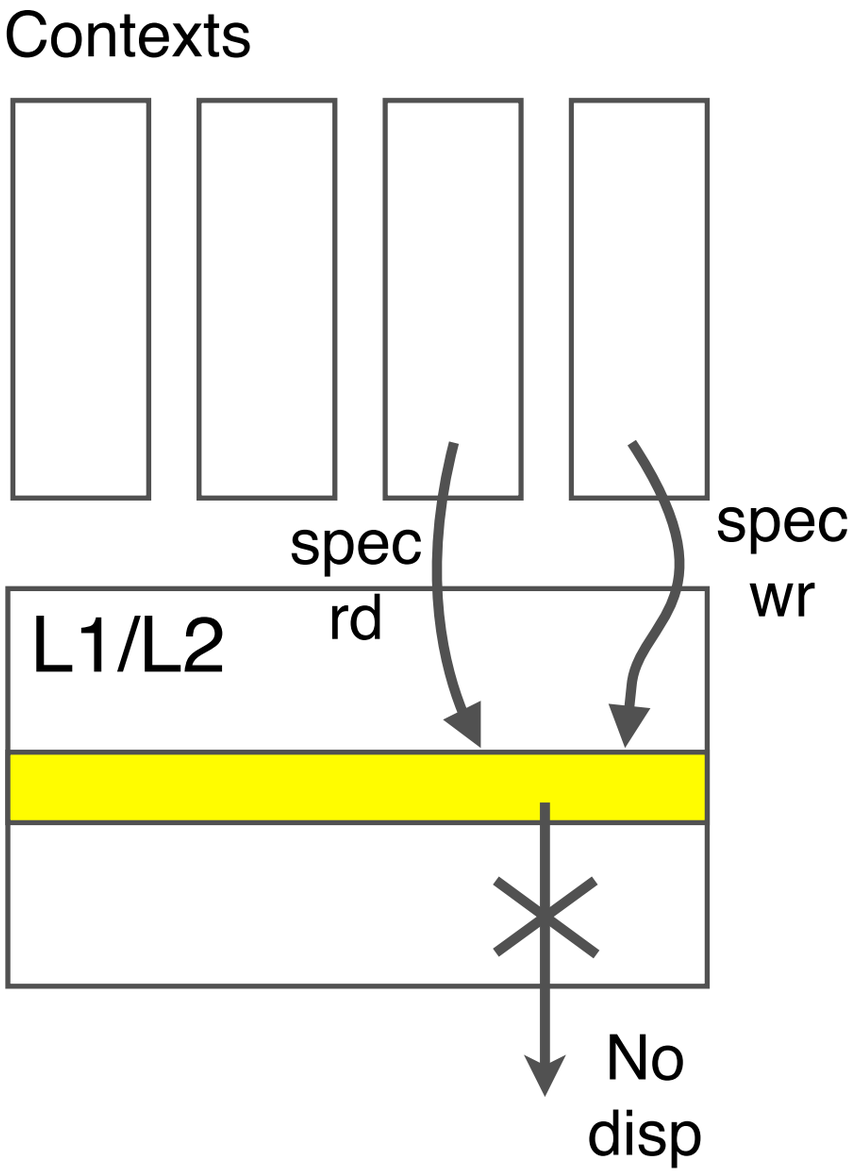
- Motivation
- Designing BulkSMT
  - Design Space
  - Hardware Mechanisms
- Multicore of BulkSMTs
- Evaluation

Blocked Execution



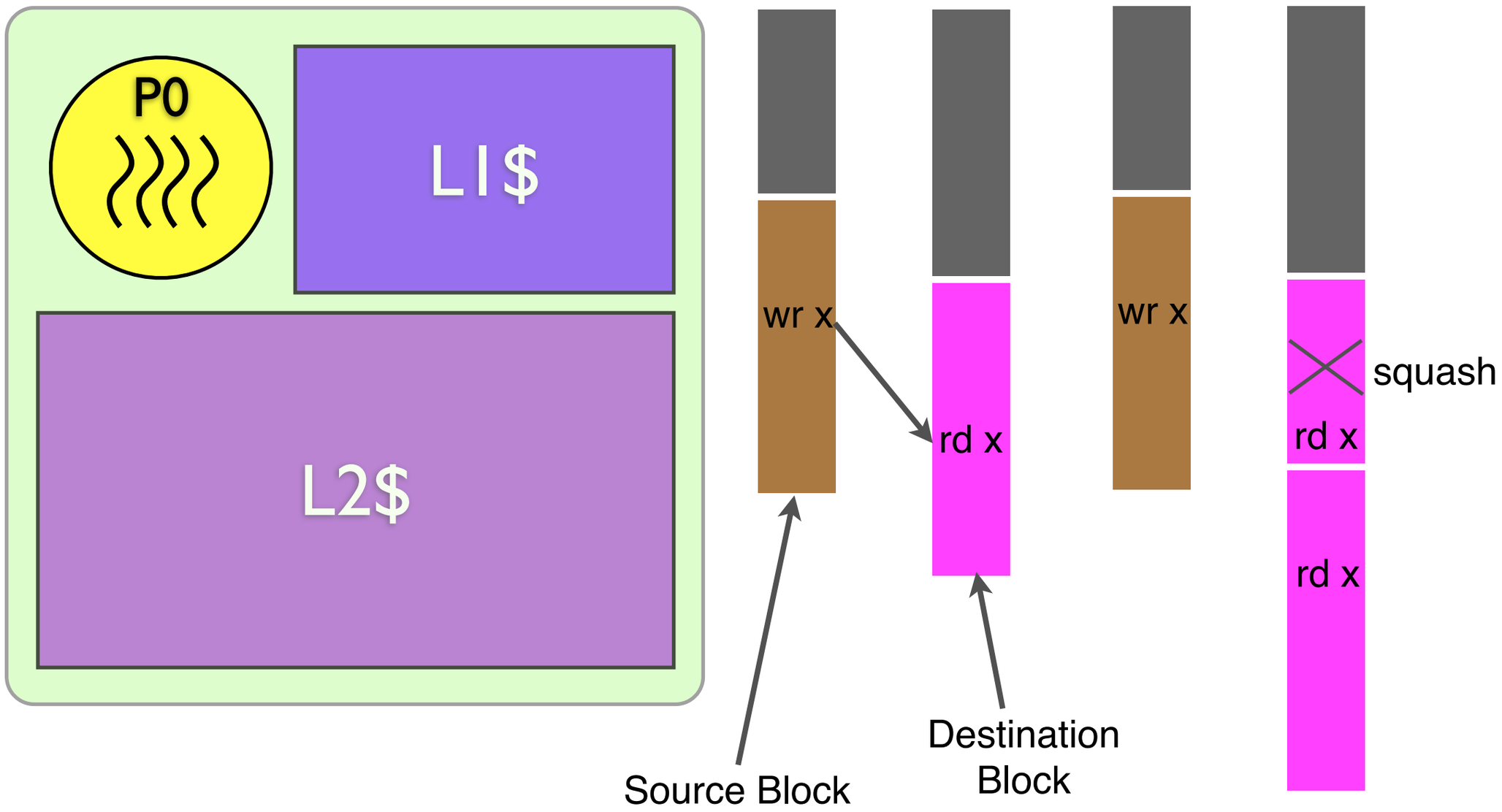
- Threads execute blocks of instructions atomically
- On inter-block dependence: typically squash and restart the block

Designing SMT For Blocked Execution

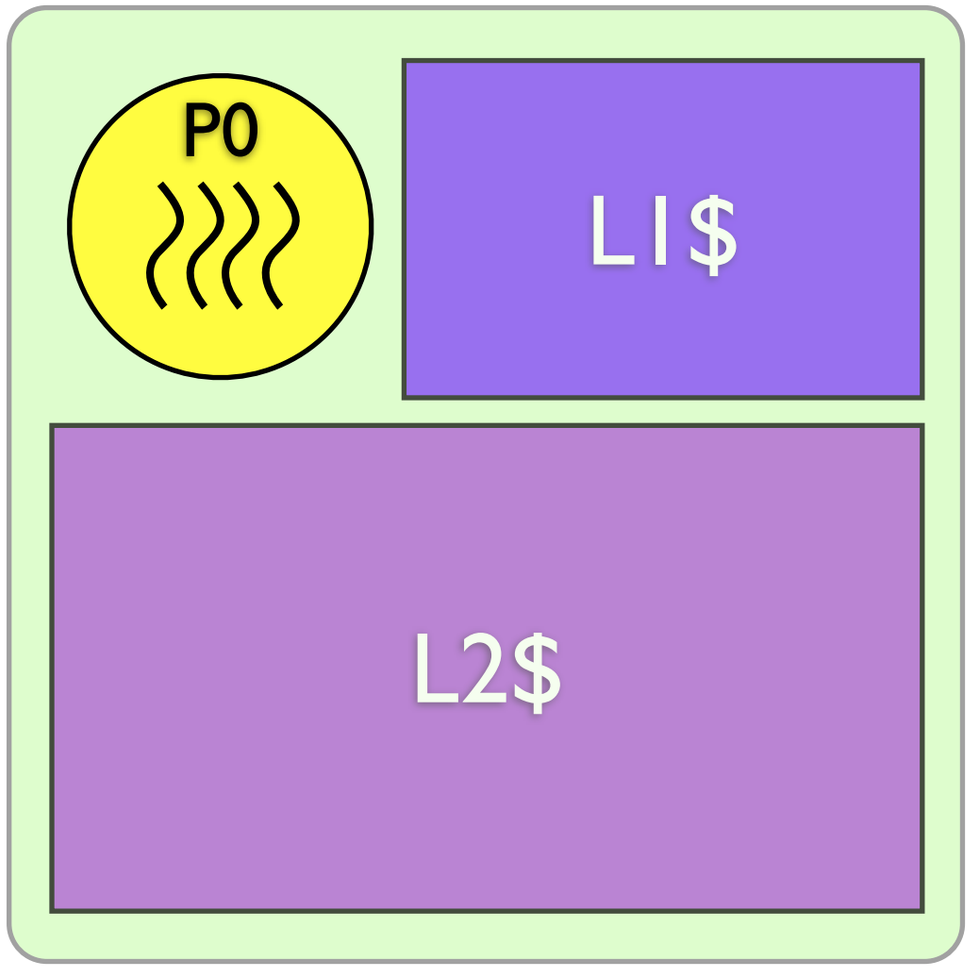


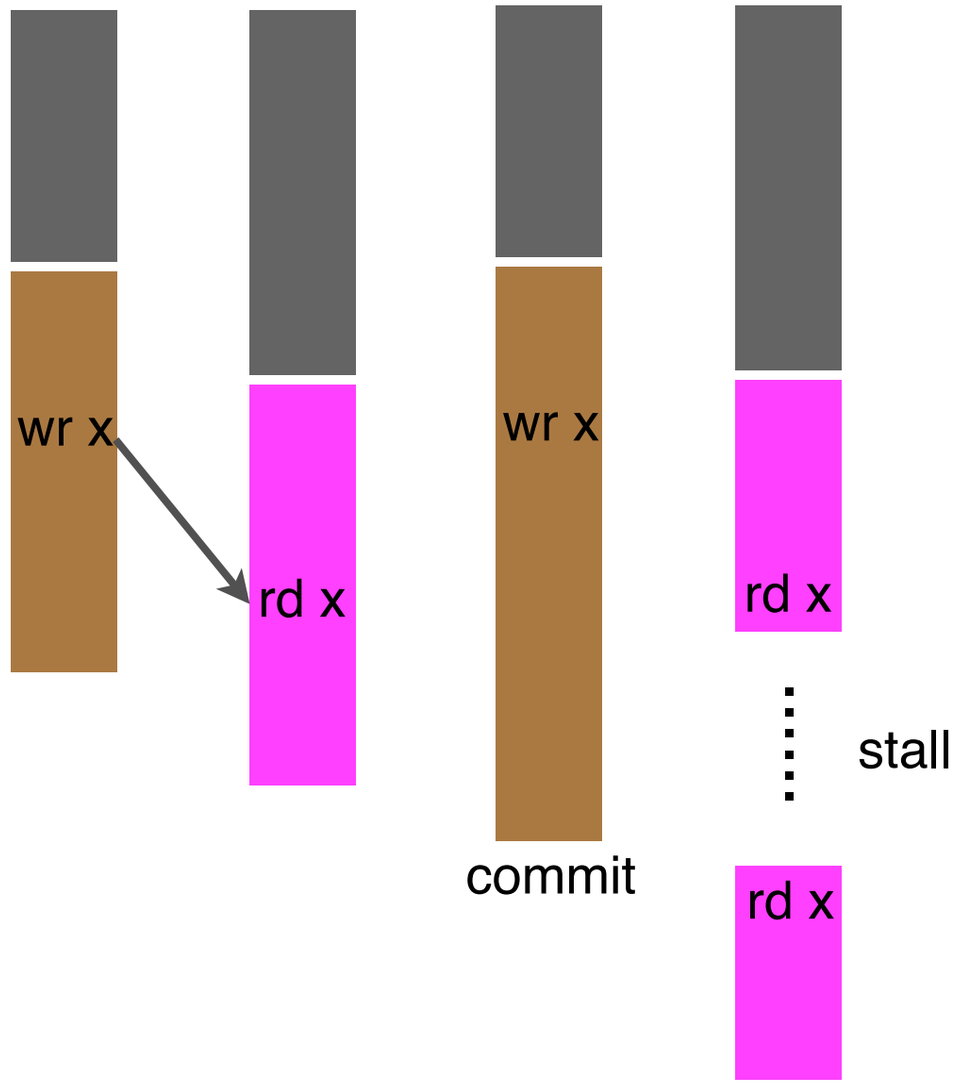
- Version Management: Eager
- Conflict Detection: Eager
- Conflict Resolution:
  - SQUASH
  - STALL
  - ORDER

SQUASH Design

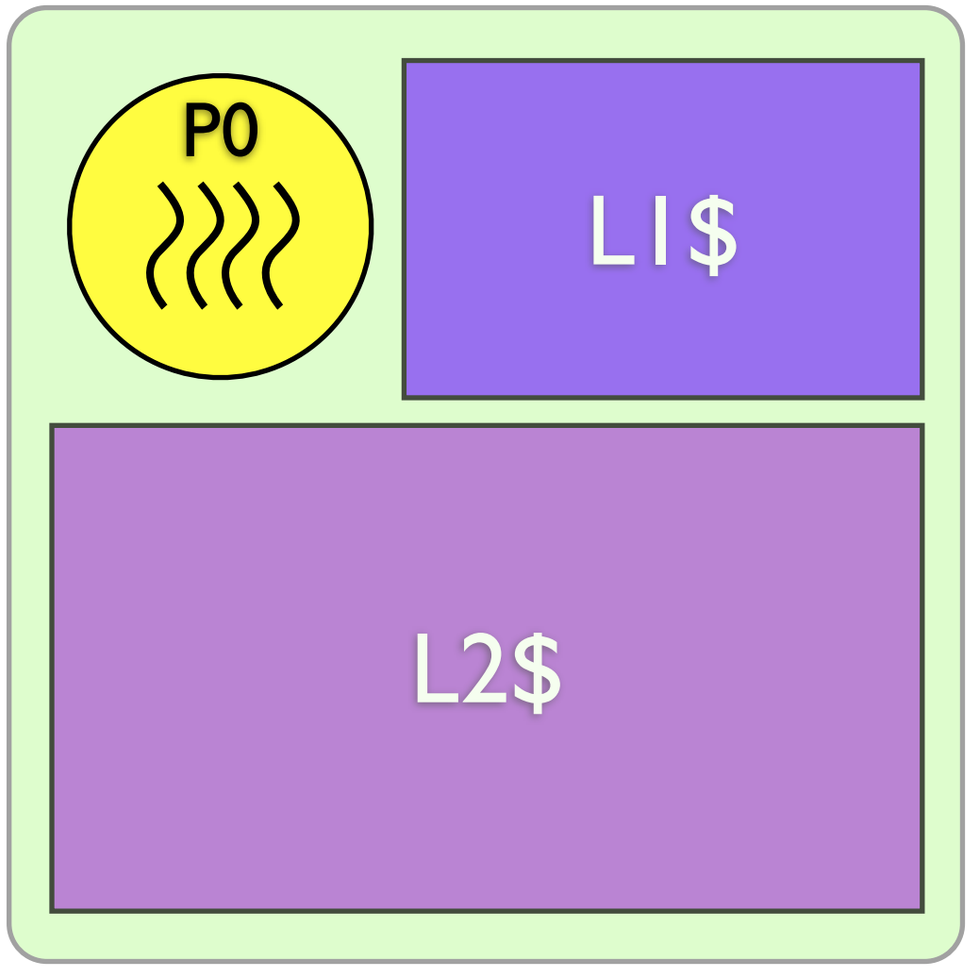


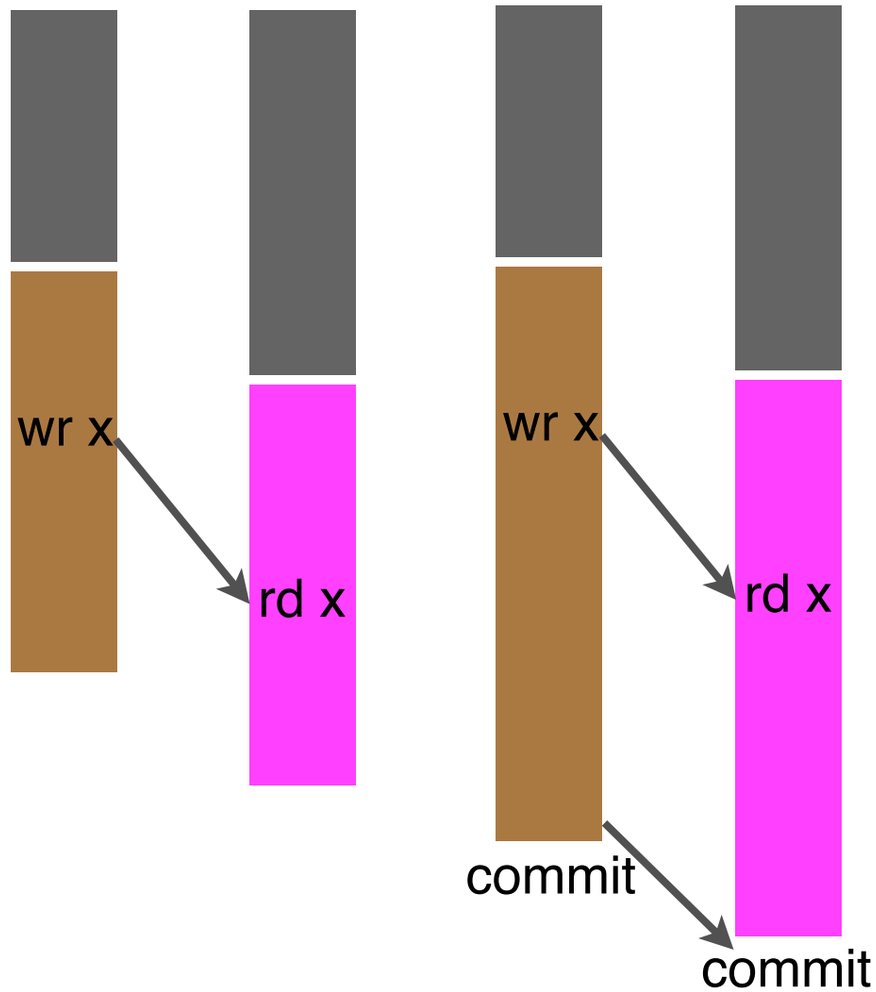
STALL Design





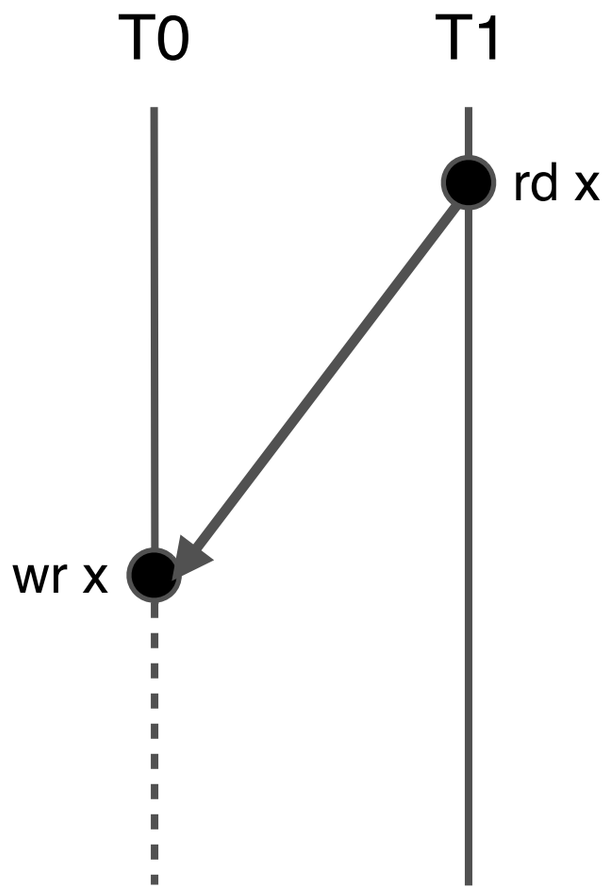
ORDER Design





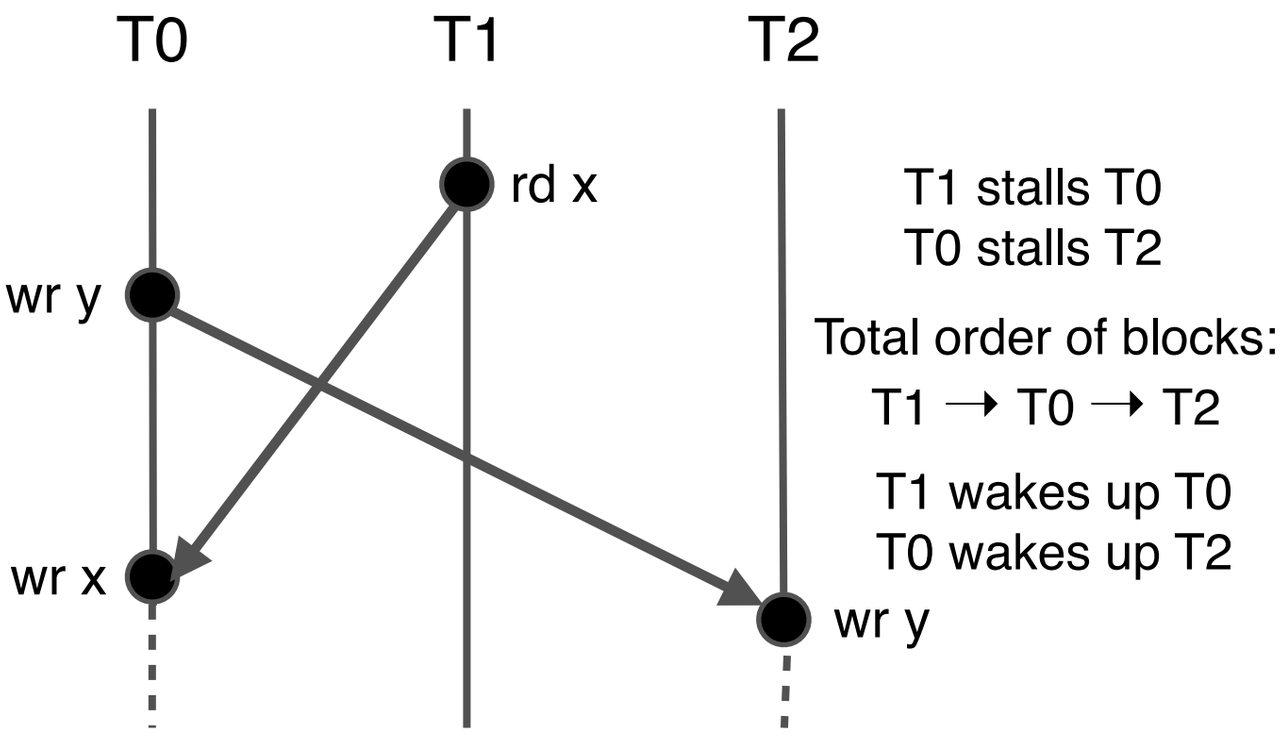
Dependences in STALL

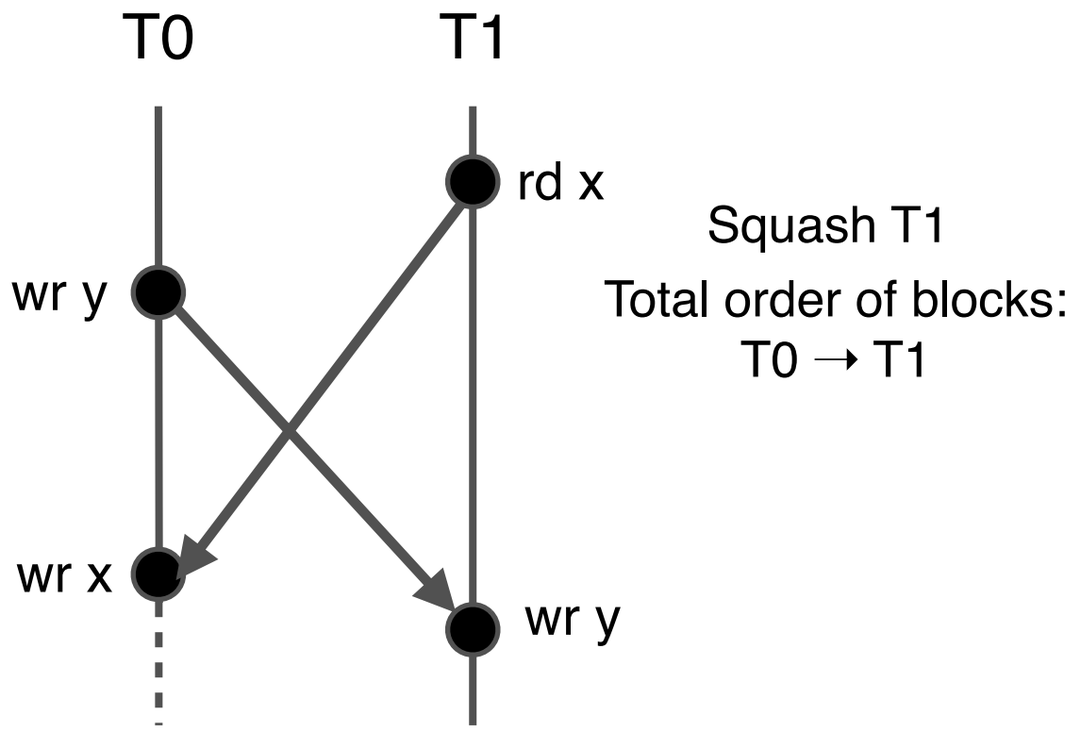
- On a dependence:
  - Stall the destination block
  - HW records source and destination blocks
- On commit/squash of source block
  - Wake up stalled destination block



Multiple Dependences in STALL

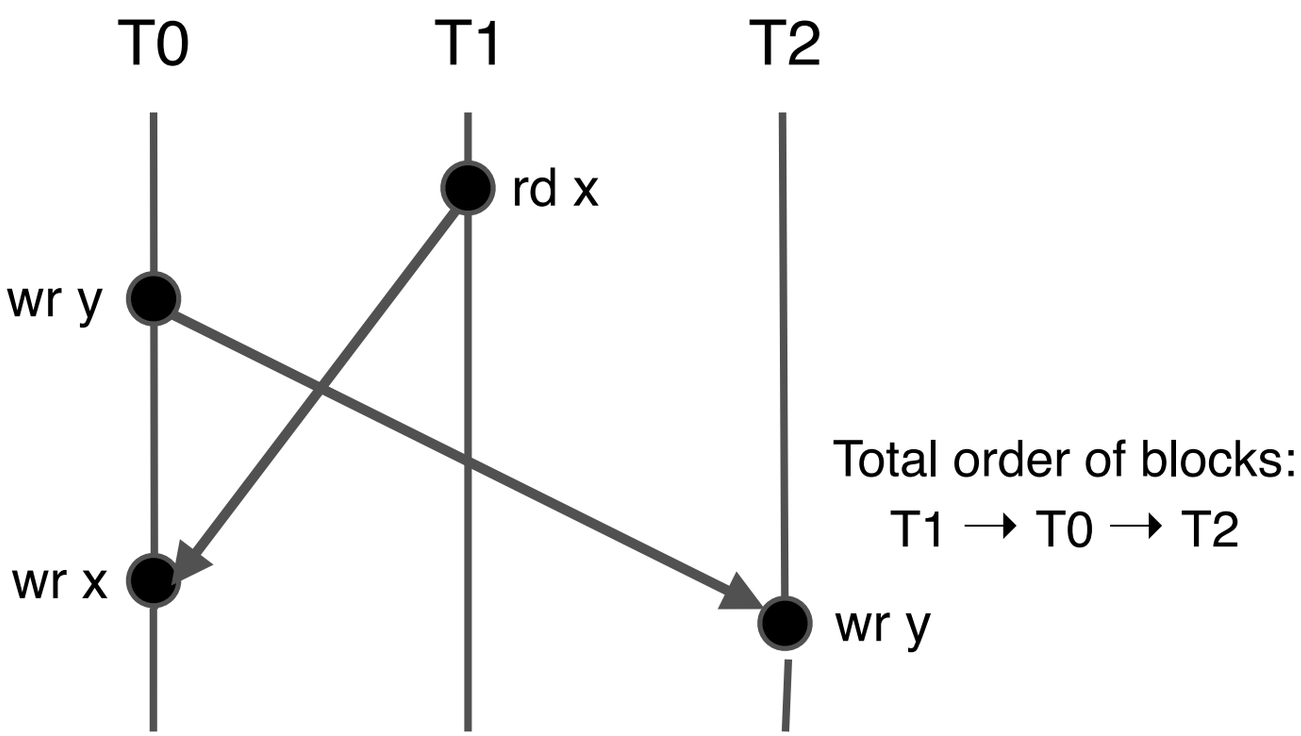
- Block can stall on an already stalled block: Transitive Stall
- Cycle not OK
  - When a stall cycle is formed, block that closes the cycle is squashed





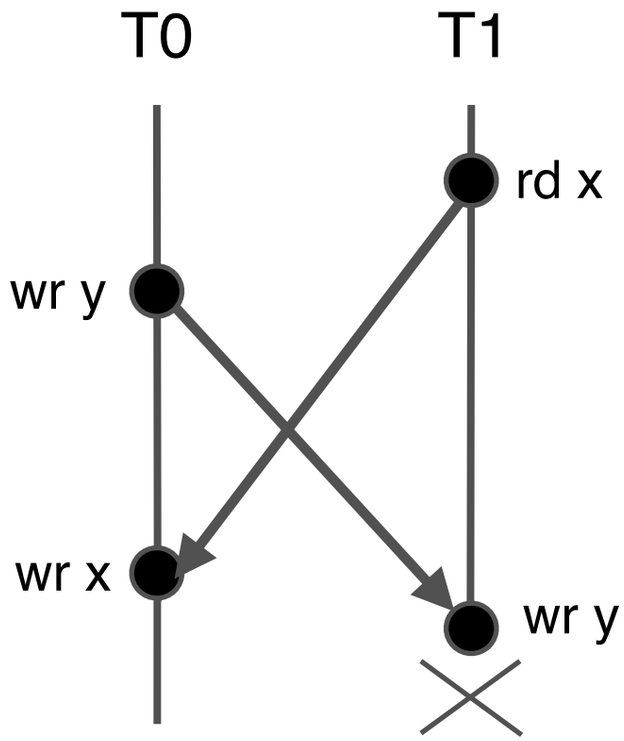
Dependences in ORDER

- On a dependence:
  - HW records source and destination block; Both blocks proceed
  - HW will enforce the same order in commit
- Transitive order is OK
- Cycle is not OK
  - On cycle, HW breaks it by squashing one or more blocks involved in the cycle



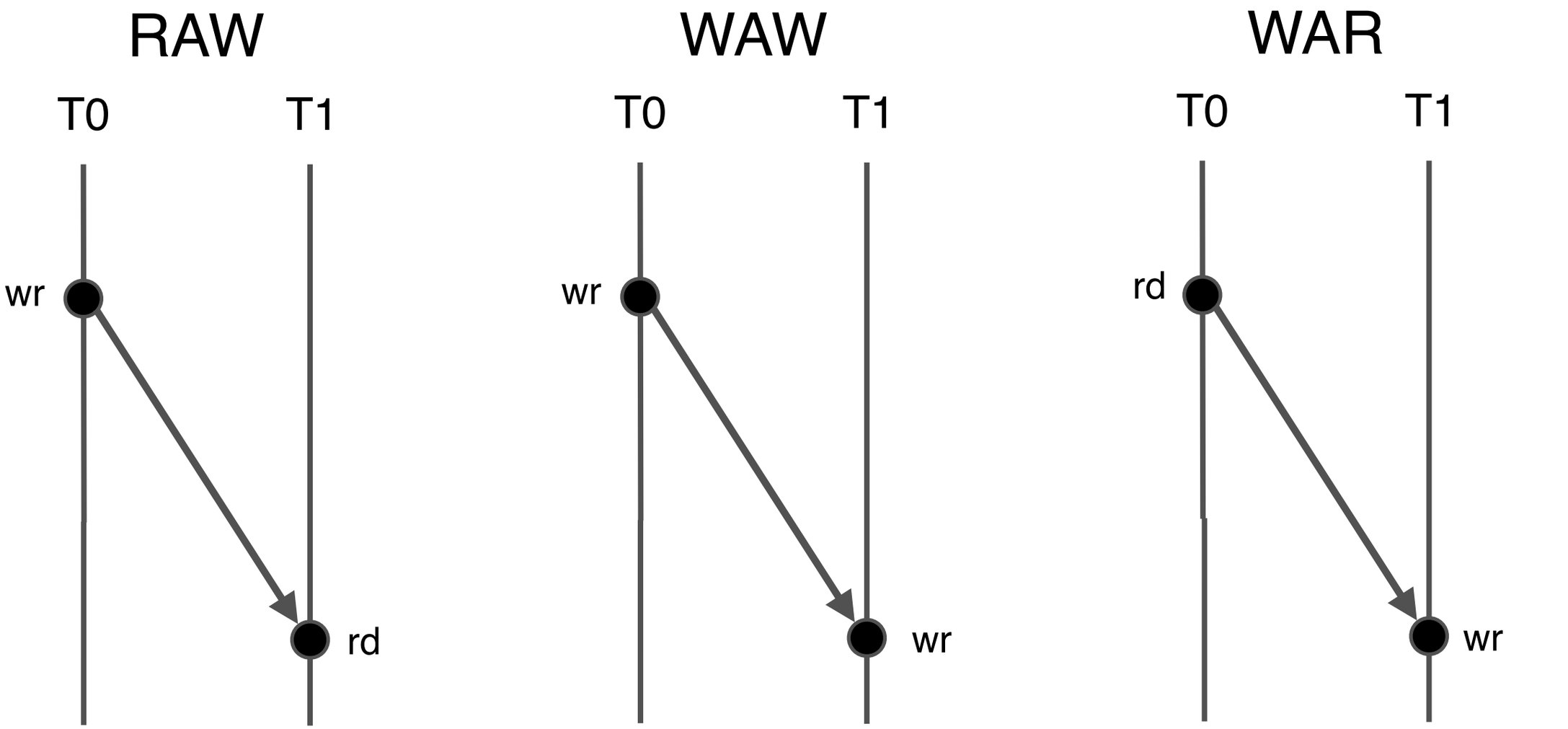
Total order of blocks:

T1 → T0 → T2



Squash Policy in ORDER

- On a cycle: different dependences have different squash requirements



Squash T0 ⇒ Squash T1
Squash T1 ⇏ Squash T0

Squash T0 ⇒ Squash T1
Squash T1 ⇒ Squash T0

Squash T0 ⇏ Squash T1
Squash T1 ⇏ Squash T0

Outline

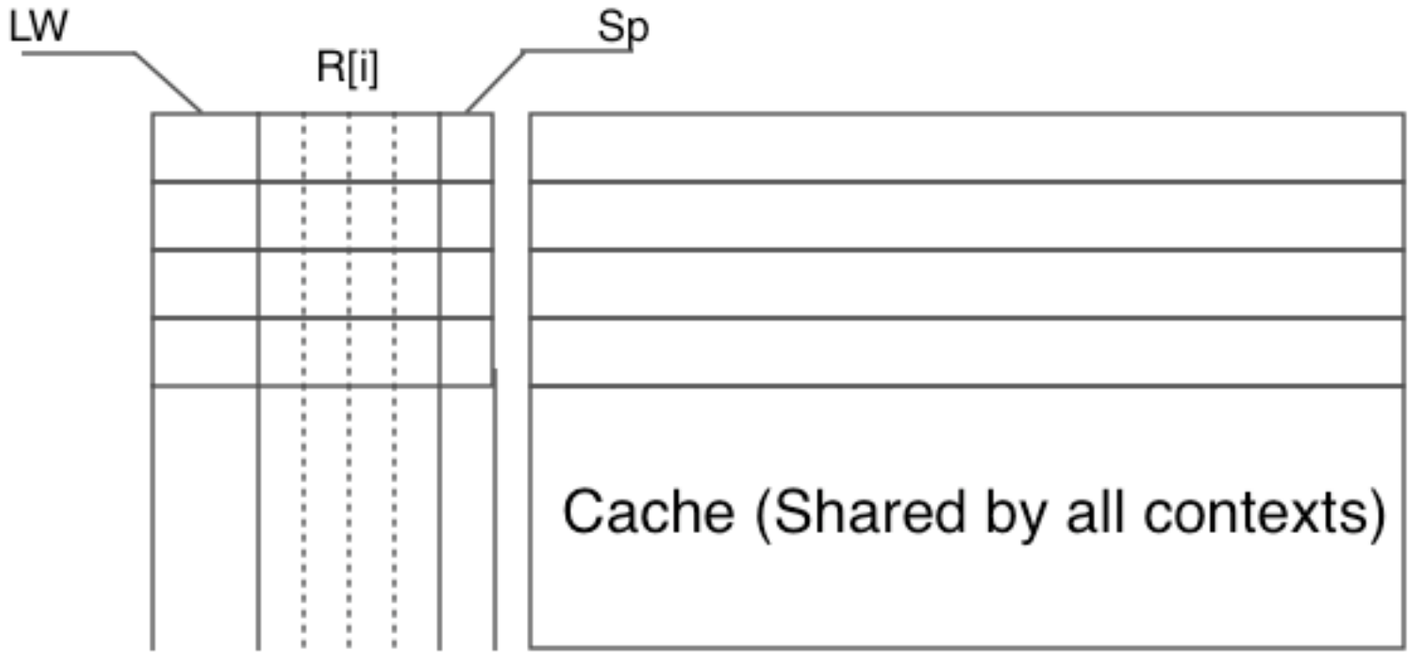
- Motivation
- Designing BulkSMT
  - Design Space
  - Hardware Mechanisms
- Multicore of BulkSMTs
- Evaluation

Hardware Mechanisms

Mechanism	Function	Impl.	Designs
Access Recording and Conflict Detection	Record the addresses accessed by each thread and detect when two threads have a data conflict	Access Bits in cache and related logic	All
Cycle Detection	Record data conflicts and their ordering, and detect conflict cycles	Dependence Table and Cycle Table	STALL ORDER
Advanced Conflict Recording	Represent the type of conflict between different blocks compactly	Enhanced Dependence Table	ORDER
Squash Set Generation	On a cycle of conflicts, decide the set of blocks to squash	Logic that operates on the Dependence Table	ORDER

Access Recording

- LW: Last Writer context-ID
- R[i]: Read bit-mask (one bit per context)
- Sp: Speculative bit



- Read by context *k*: set R[k]
- Write by context *k*: Sp ← 1, LW ← k

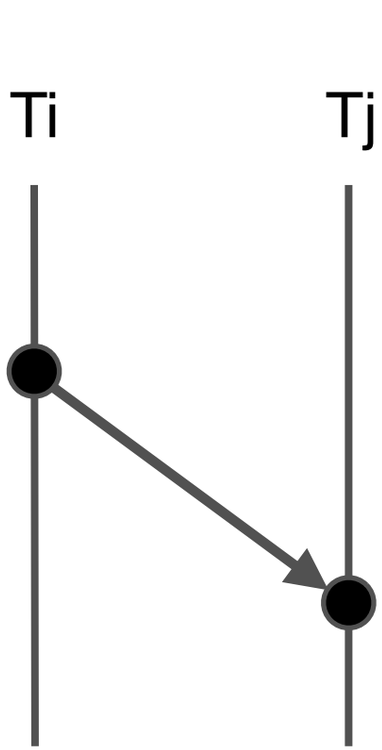
Conflict Detection

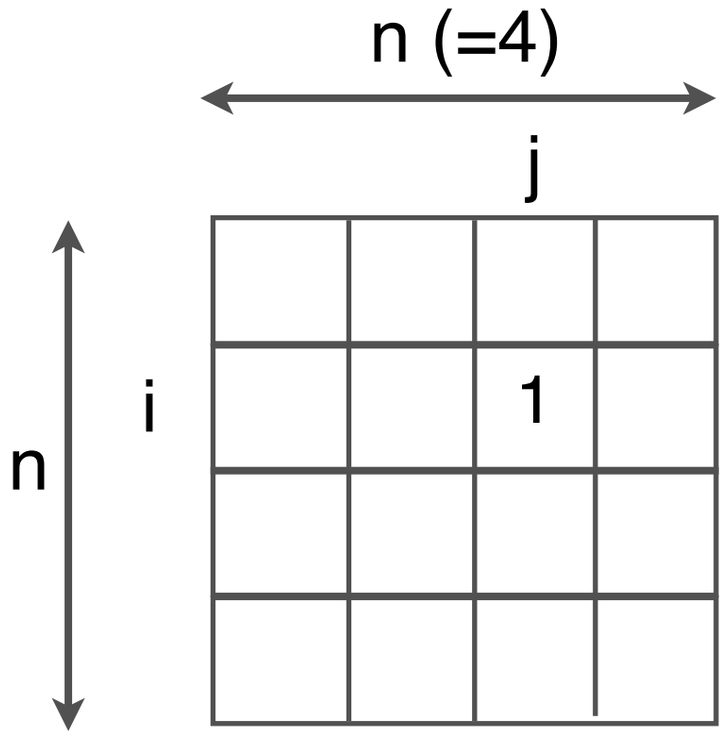
- Read After Write (RAW)
  - Load from context *k* reads cache line and (Sp == 1) && (LW != k)
- Write After Read (WAR)
  - ...
- Write After Write (WAW)
  - ...

Hardware Mechanisms

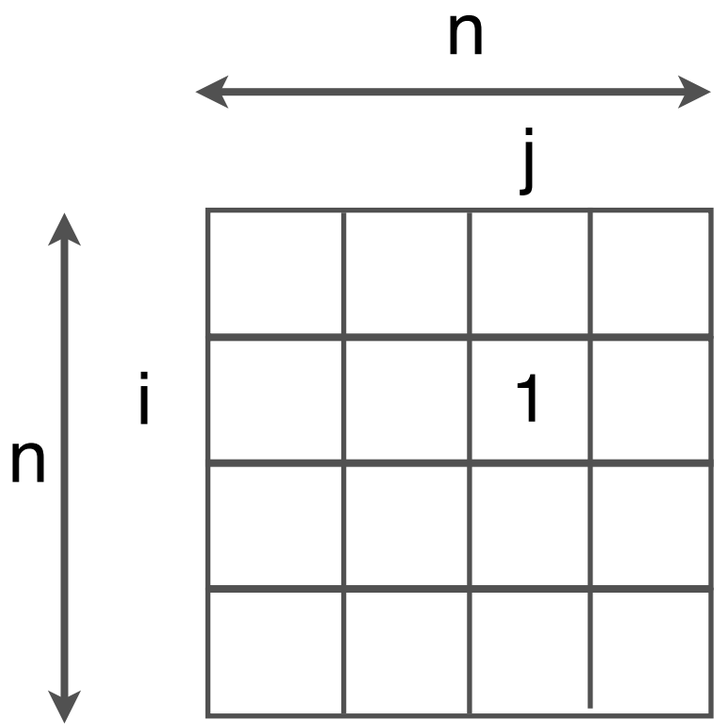
Mechanism	Function	Impl.	Designs
Access Recording and Conflict Detection	Record the addresses accessed by each thread and detect when two threads have a data conflict	Access Bits in cache and related logic	All
Cycle Detection			
Advanced Conflict Recording			
Squash Set Generation			

Cycle Detection: HW Structures





Dependence Table (DT)

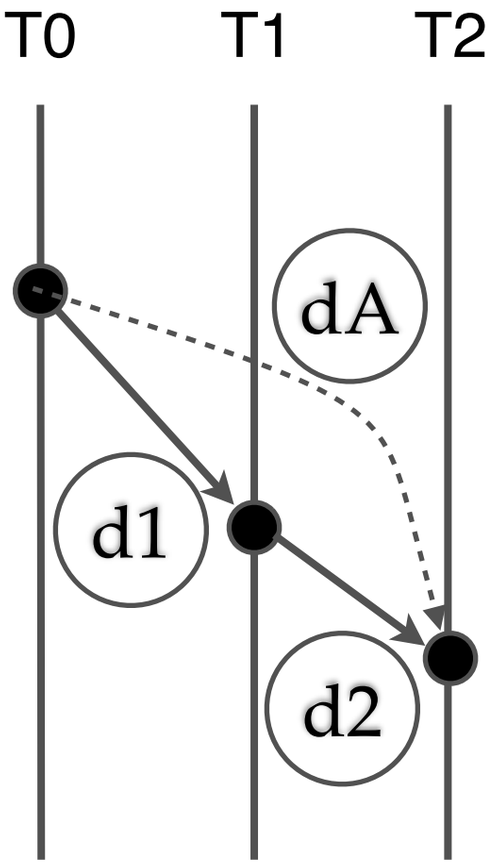


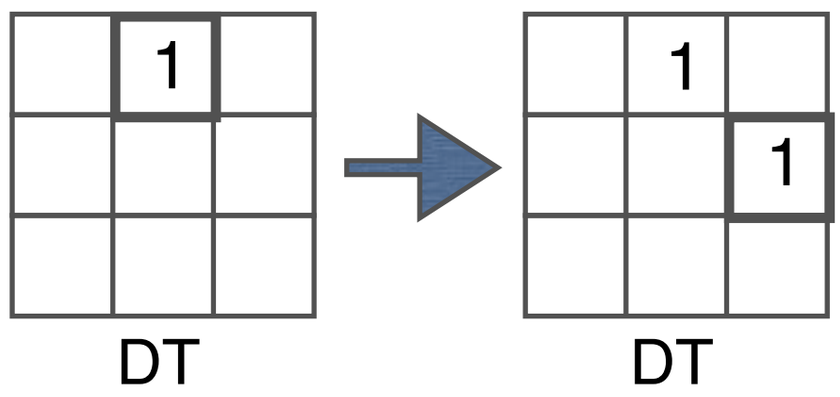
Cycle Table (CT)

Records ordered dependences
Ti → Tj

In background:
Finds cycles of dependences
Cycle = bit in diagonal

Dependence Cycle Detection

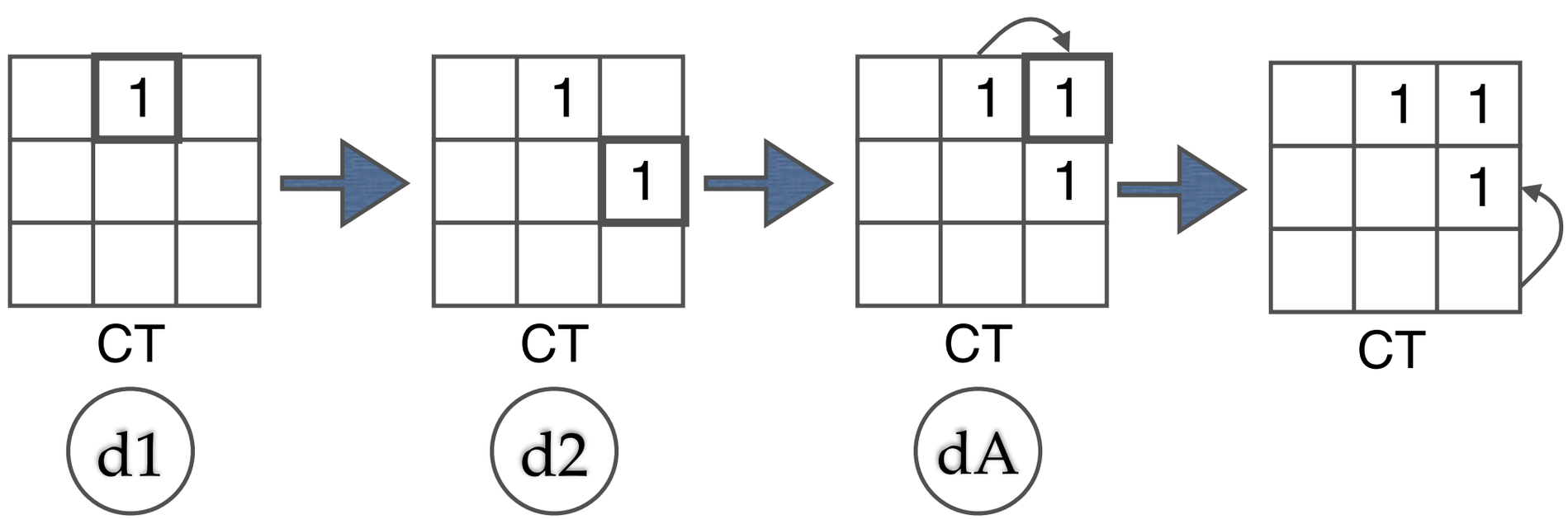




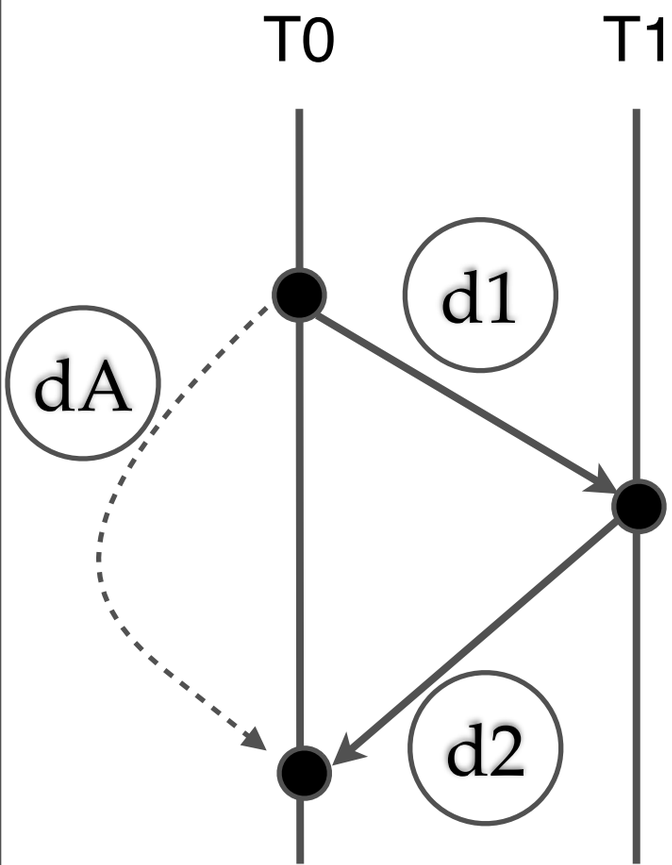
Propagation of d2 dependence:

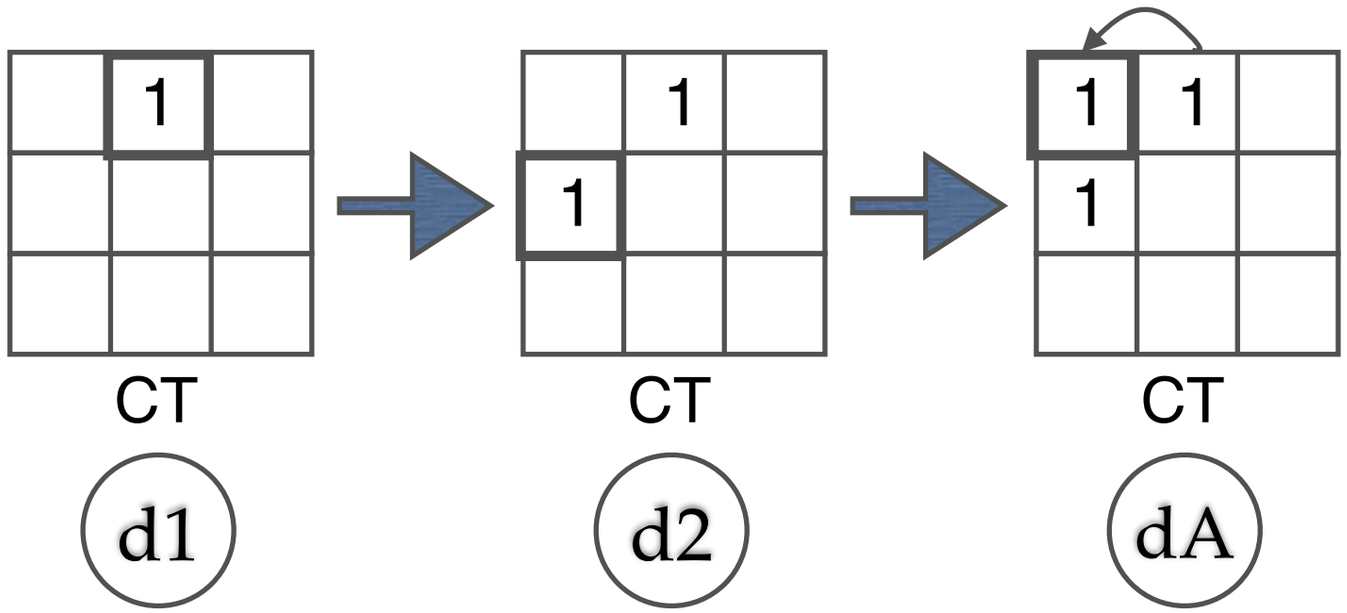
Col2 I= Col1

Row1 I= Row2



Cycle Detection





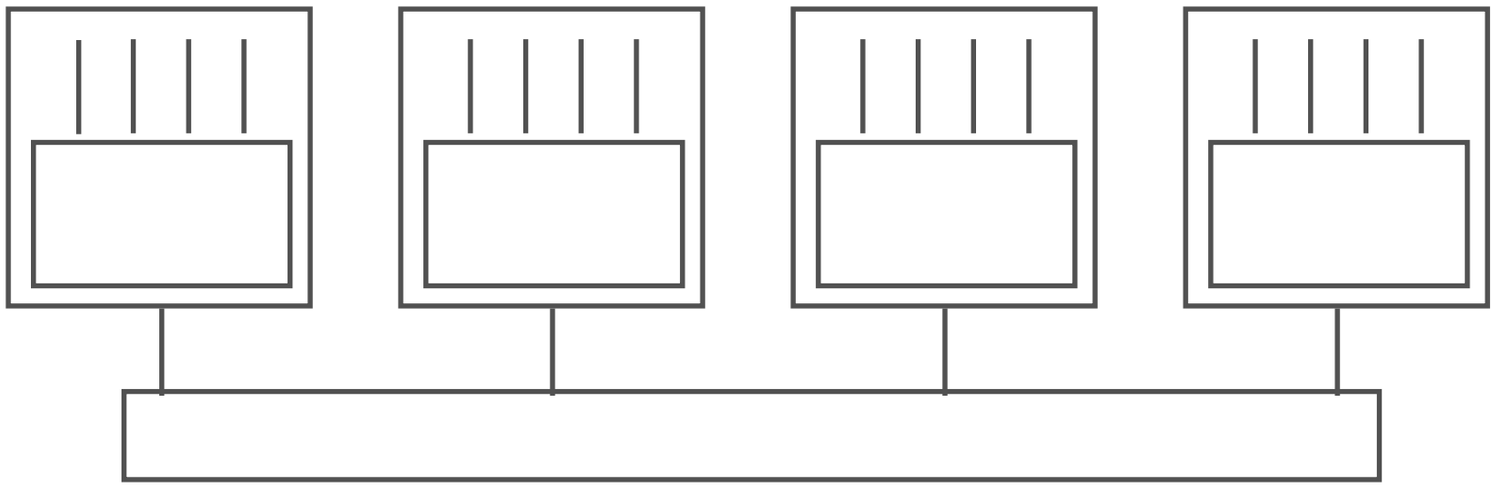
Hardware Mechanisms

Mechanism	Function	Impl.	Designs
Access Recording and Conflict Detection	Record the addresses accessed by each thread and detect when two threads have a data conflict	Access Bits in cache and related logic	All
Cycle Detection	Record data conflicts and their ordering, and detect conflict cycles	Dependence Table and Cycle Table	STALL ORDER
Advanced Conflict Recording			
Squash Set Generation			

Outline

- Motivation
- Designing BulkSMT
  - Design Space
  - Hardware Mechanisms
- Multicore of BulkSMTs
- Evaluation

Multicore of BulkSMTs



- Global Eager Scheme (EE)
  - Correct block commit:
    - Commit globally first, then commit locally
  - Correct operation on reception of cache invalidation:
    - Check the cache access bits; may squash multiple blocks
- Global Lazy Scheme (LL): see paper

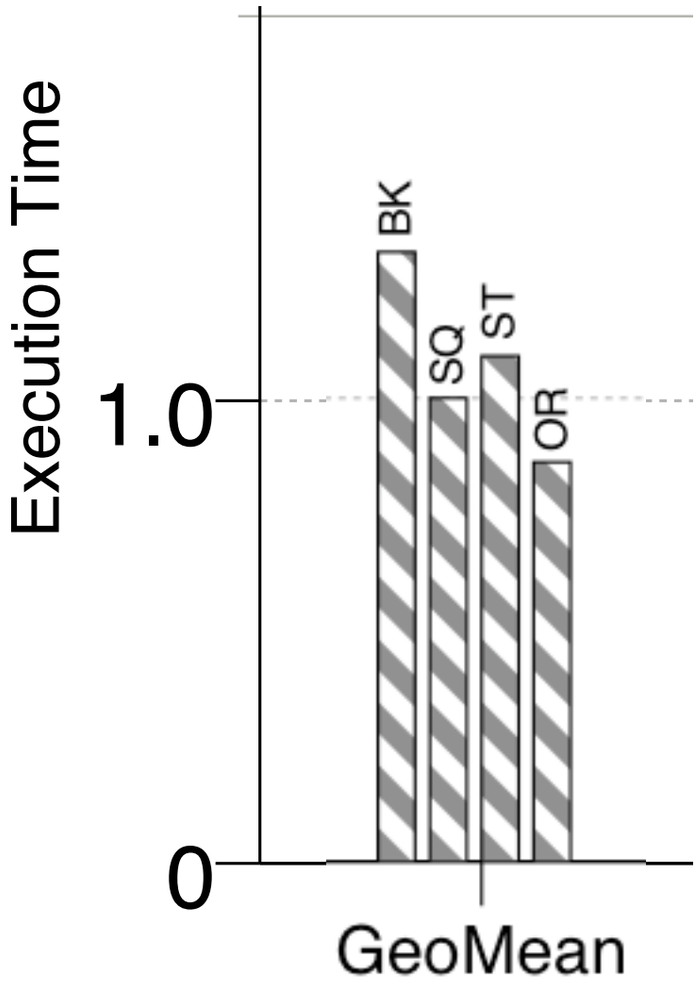
Outline

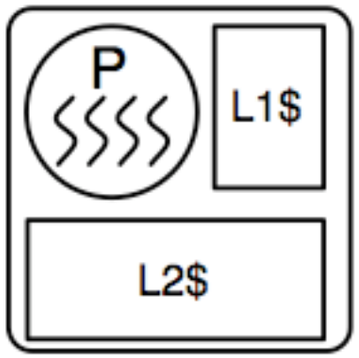
- Motivation
- Designing BulkSMT
  - Design Space
  - Hardware Mechanisms
- Multicore of BulkSMTs
- Evaluation

Evaluation

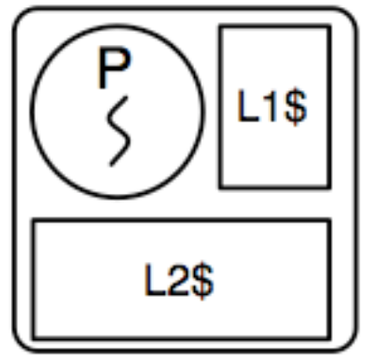
- Cycle-accurate execution-driven simulator based on SESC
- 6 SPLASH-2 and 2 PARSEC applications
- Applications run with 1, 4, or 16 threads
- Compare performance between BulkSMT and conventional block:
  - Using same number of cores, diff number of threads
  - Using same number of threads, diff number of cores

Execution Time (Same Core Count, 1)





SQ,ST,OR



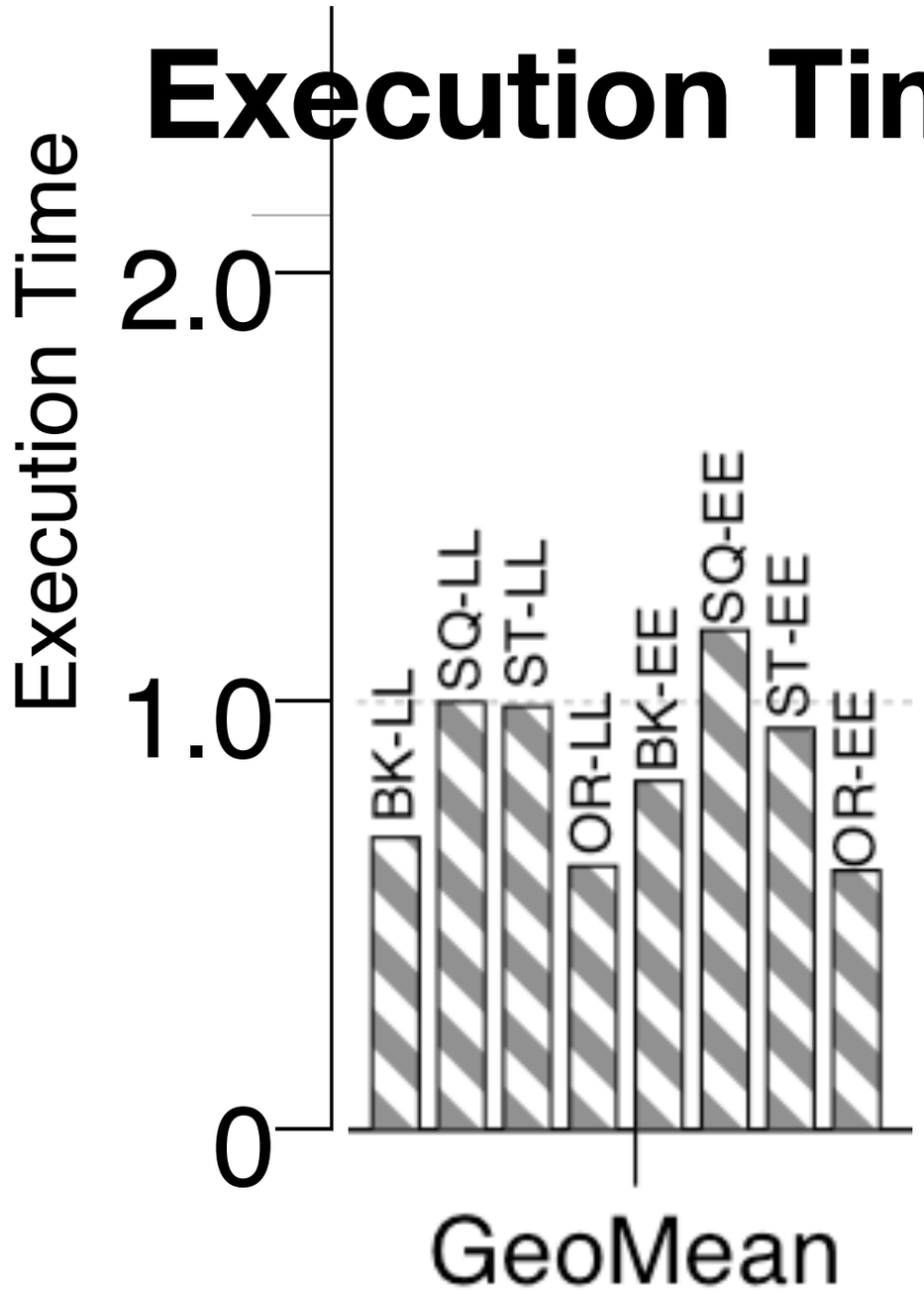
BK

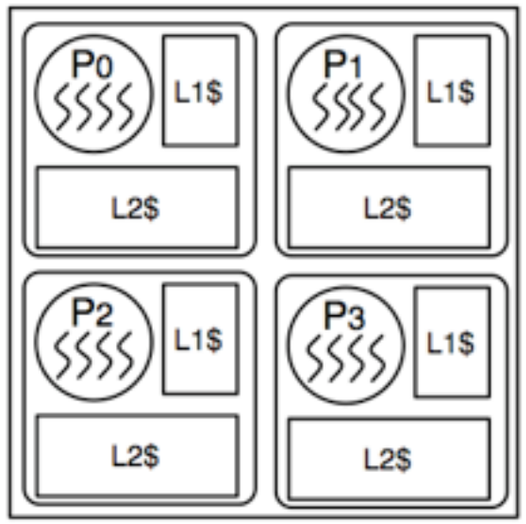
SQ: SQUASH
ST: STALL
OR: ORDER

BK:Conventional blocked

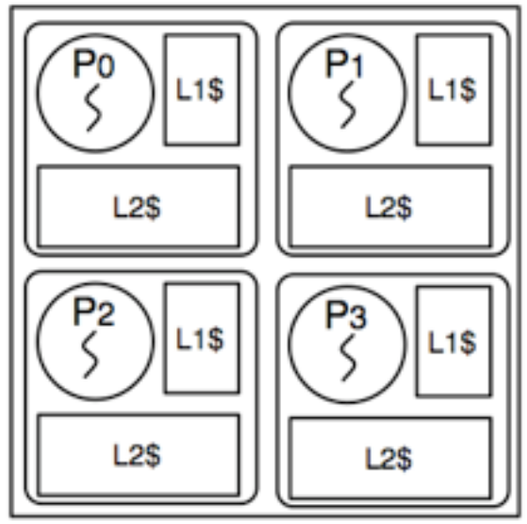
- BulkSMT more cost effective: faster for the same core count
- Among the BulkSMT designs, ORDER is the best

Execution Time (Same Core Count, 4)





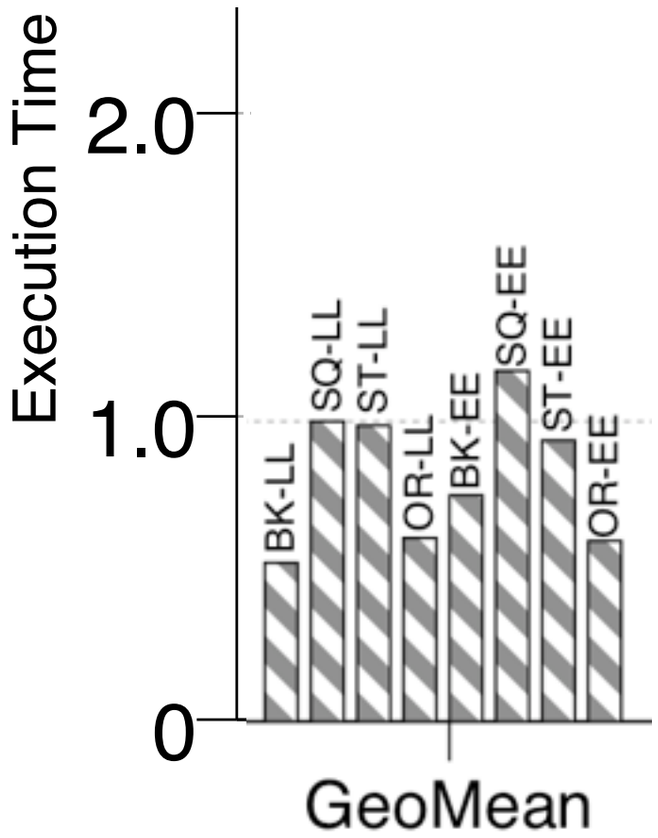
{SQ,ST,OR}-{LL,EE}

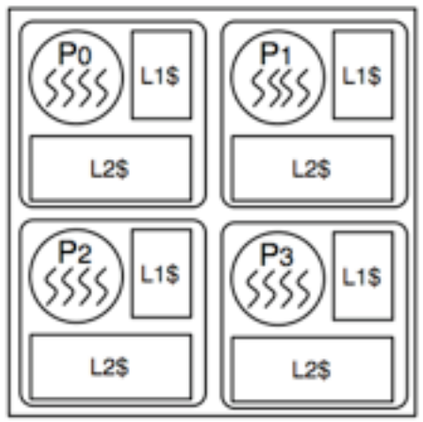


BK-{LL,EE}

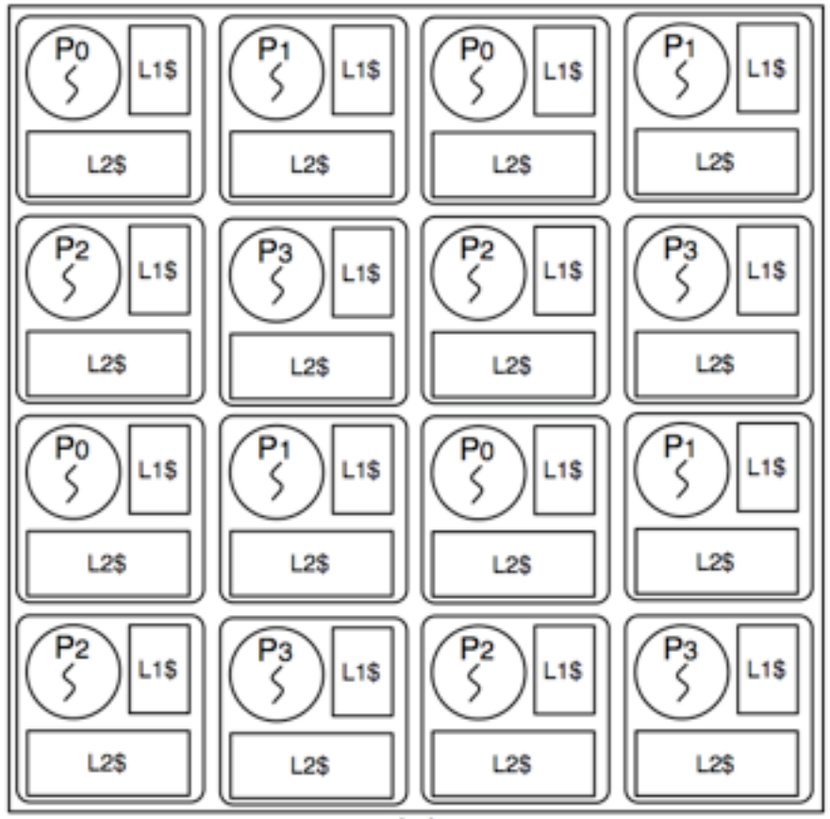
- BulkSMT ORDER is best
- BulkSMT limited by application scalability

Execution Time (Same Thread Count, 16)





{SQ,ST,OR}-{LL,EE}



BK-{LL,EE}

- BulkSMT ORDER multicore:
  - Uses 1/4 of BK hardware and achieves comparable performance
  - Reason: ability to avoid squashes

Also in the paper

- Other hardware mechanisms
- Implementation issues
- Handling high-contention synchronization
- Other characteristics of execution behavior
- Related work

Conclusion

- Proposed BulkSMT: first SMT design that supports atomic-block (transactional) execution
  - Enables concurrency between dependent blocks
- Proposed designs of different concurrency vs cost:
  - SQUASH on conflict
  - STALL on conflict
  - ORDER on conflict
- Designed a multicore of BulkSMTs
- BulkSMT ORDER is cost effective
  - Higher performance for the same core count
  - Competitive performance for 1/4 of the cores

BulkSMT:
Designing SMT Processors for Atomic-Block Execution

Xuehai Qian, Benjamin Sahelices and Josep Torrellas
University of Illinois

University
of Illinois

http://iacoma.cs.uiuc.edu/

i-acoma group