

BulkSC: Bulk Enforcement of Sequential Consistency

Luis Ceze, James Tuck, Pablo Montesinos, Josep Torrellas

University of Illinois at Urbana-Champaign

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



ISCA 2007, San Diego, CA



Memory Consistency Model



Memory Consistency Model

- Defines the values that a read can return

Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware

Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major implications on programmability

Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major implications on programmability

	Flag = Data = 0	
P1		P2
Data = 2000		while (Flag==0){ ; }
Flag = 1		... = Data

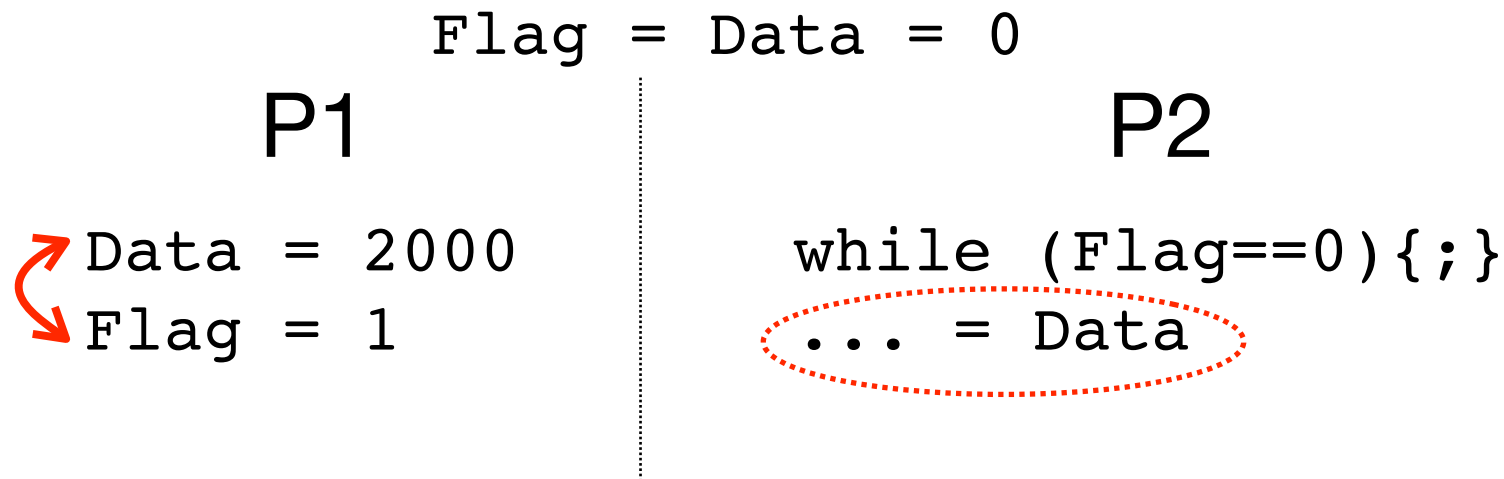
Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major implications on programmability

	Flag = Data = 0	
P1		P2
Data = 2000		while (Flag==0){;
Flag = 1		... = Data

Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major implications on programmability




Memory Consistency Model

- Defines the values that a read can return
- Supported by the hardware
- Has major impact on the software stack:

Affects the whole software stack:

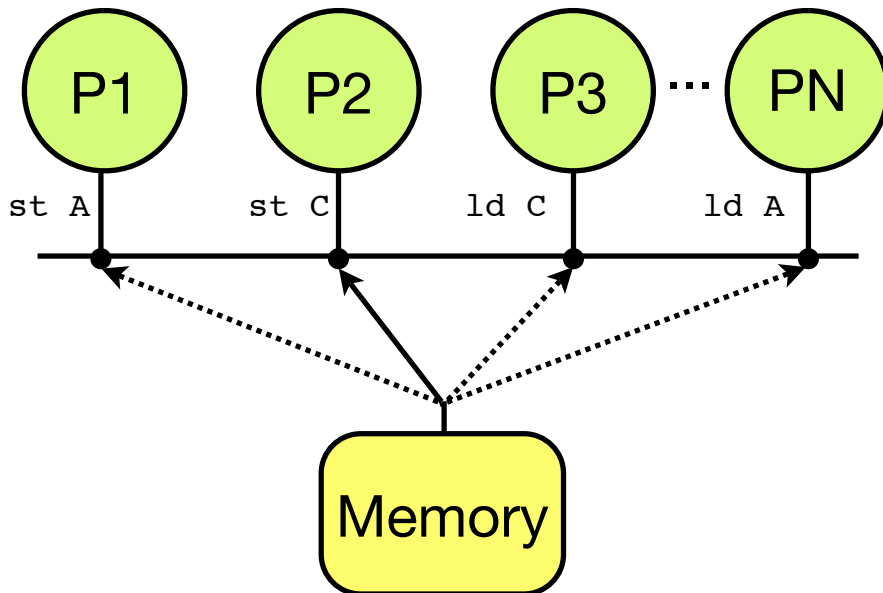
- applications
- OS, libraries, drivers
- compilers
- language semantics

 Data
Flag = 1 ... = Data ; }

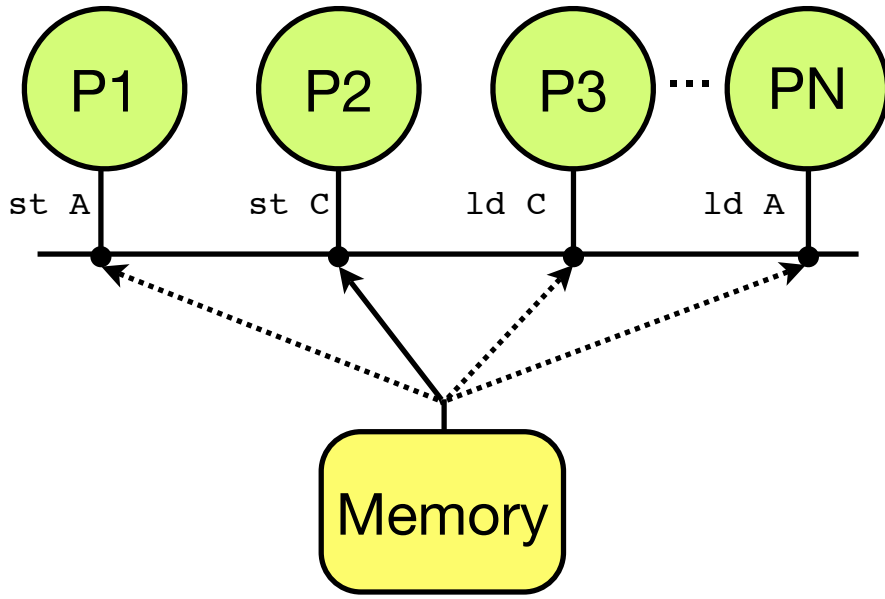
Sequential Consistency (SC)



Sequential Consistency (SC)

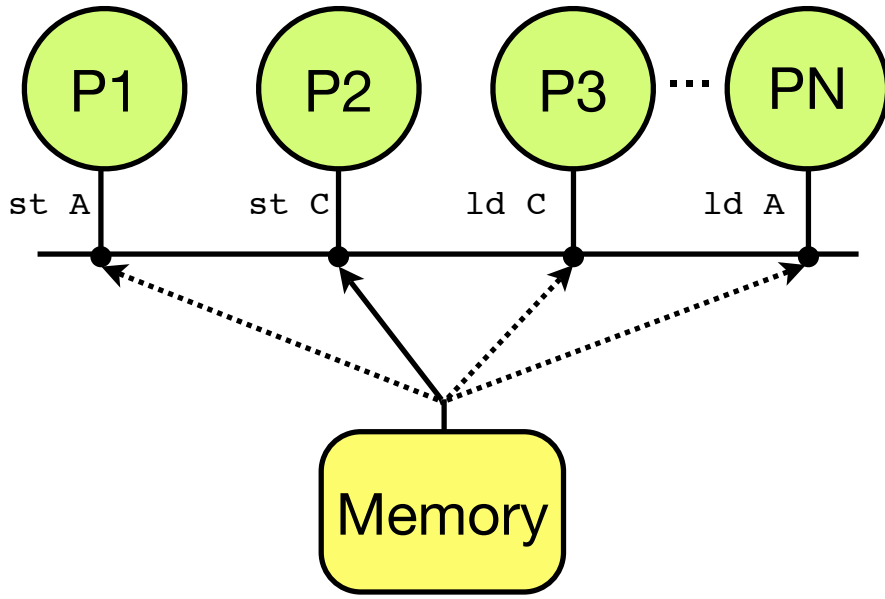


Sequential Consistency (SC)



[Lamport'79]

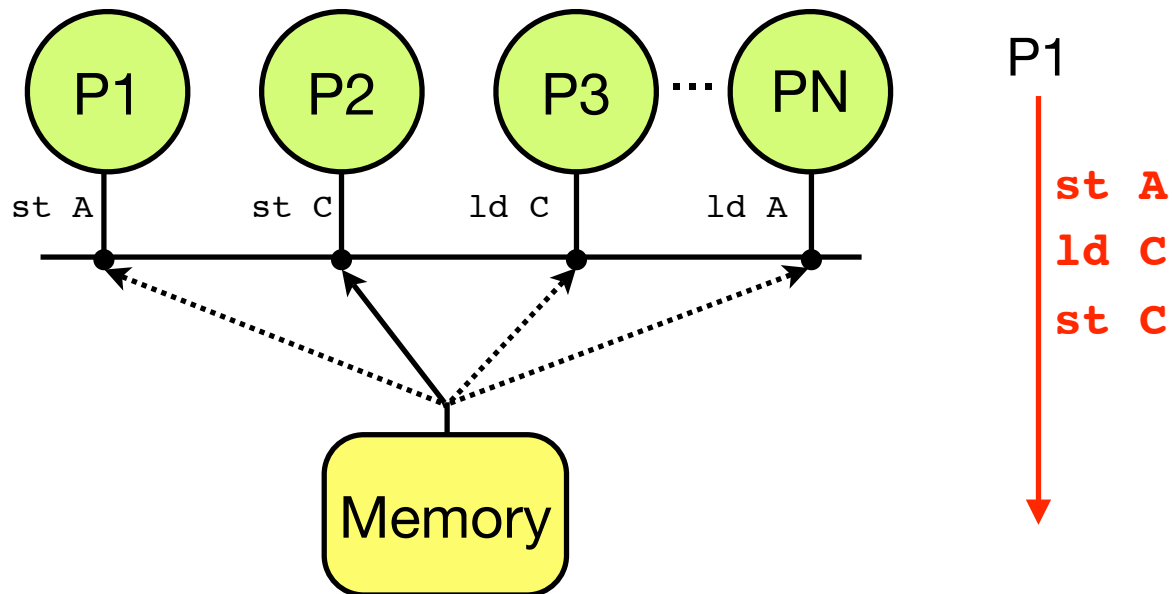
Sequential Consistency (SC)



Per-processor program order: memory operations from individual processors maintain program order

[Lamport'79]

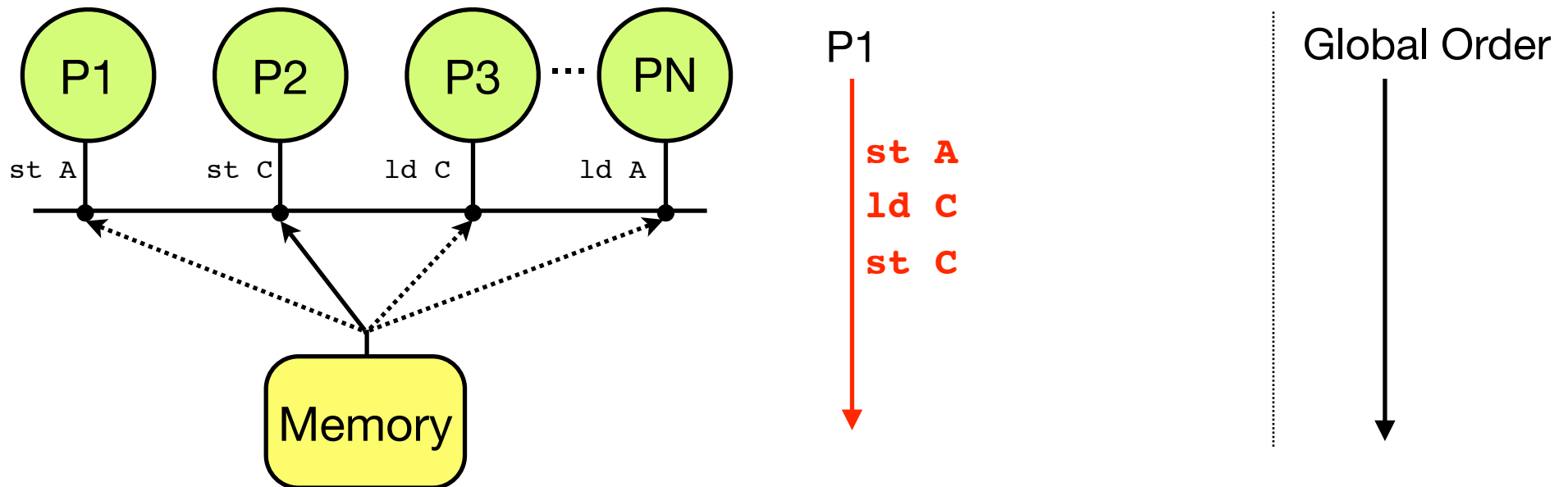
Sequential Consistency (SC)



Per-processor program order: memory operations from individual processors maintain program order

[Lamport'79]

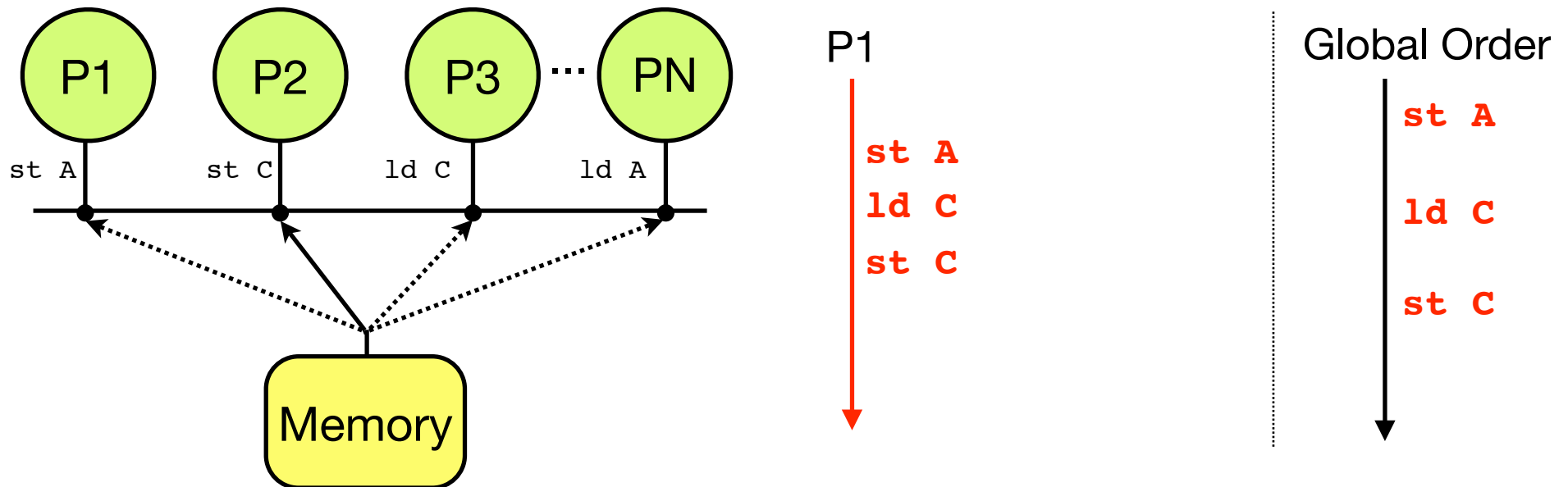
Sequential Consistency (SC)



Per-processor program order: memory operations from individual processors maintain program order

[Lamport'79]

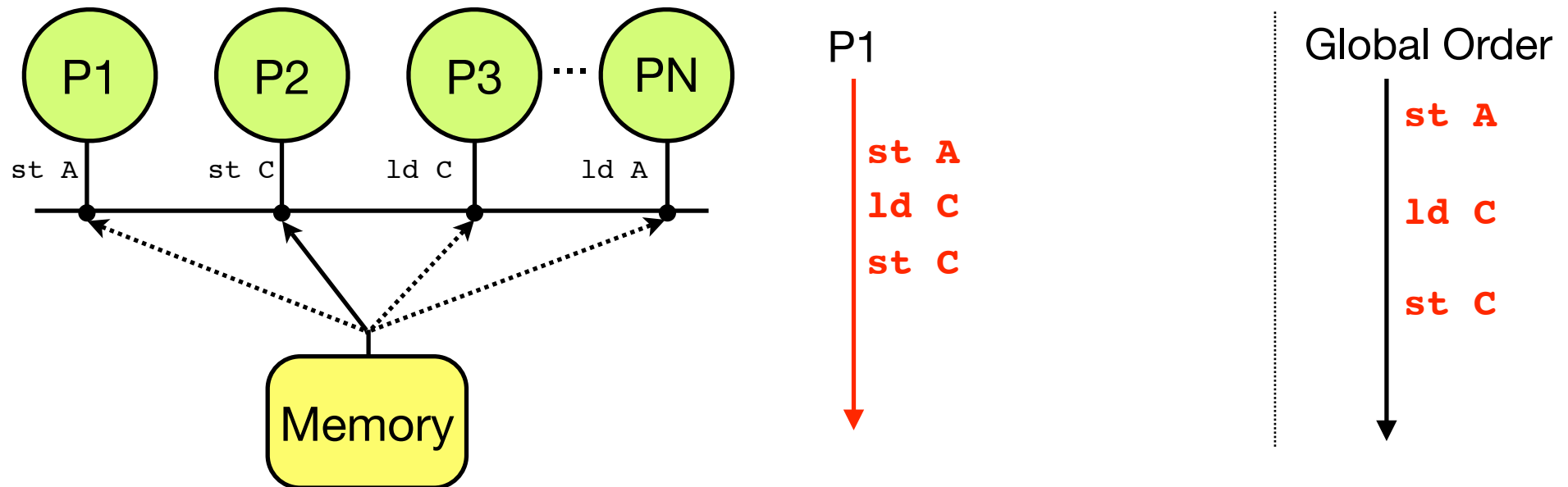
Sequential Consistency (SC)



Per-processor program order: memory operations from individual processors maintain program order

[Lamport'79]

Sequential Consistency (SC)

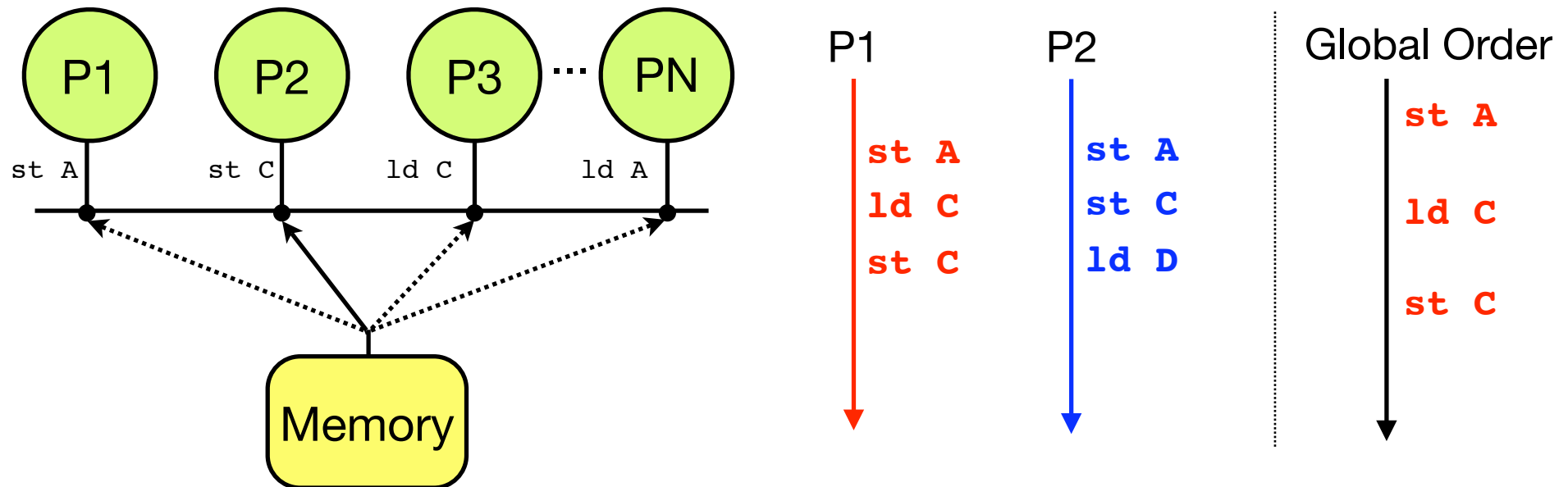


Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Lamport'79]

Sequential Consistency (SC)

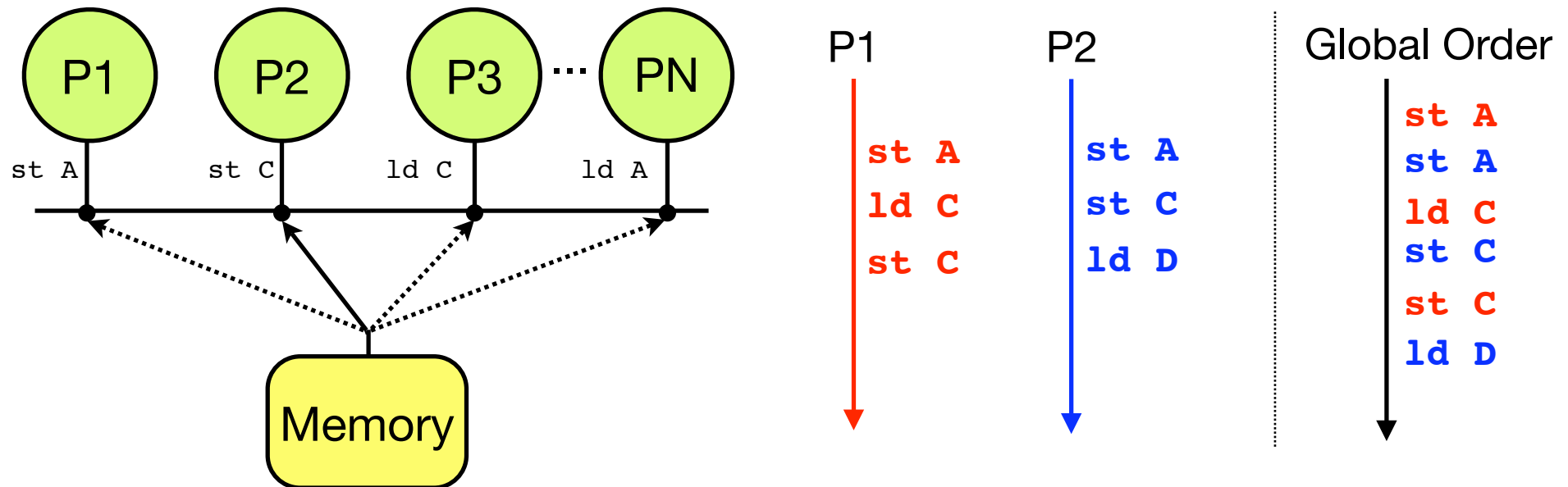


Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Lamport'79]

Sequential Consistency (SC)

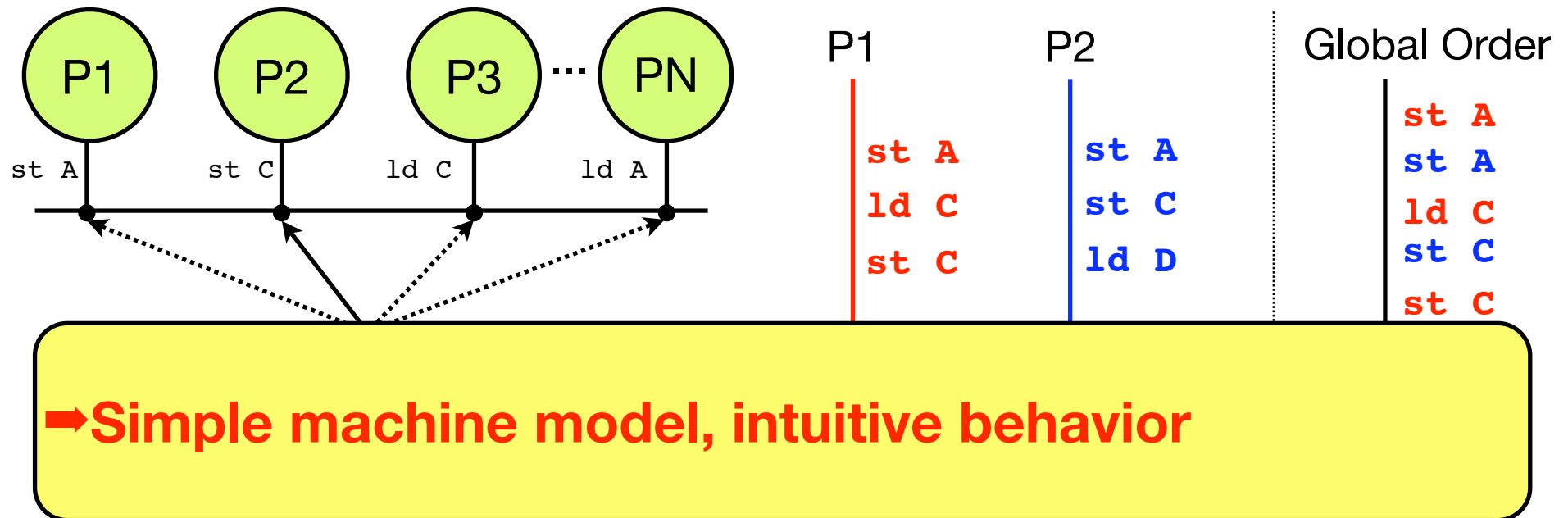


Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Lamport'79]

Sequential Consistency (SC)



Per-processor program order: memory operations from individual processors maintain program order

Single sequential order: the memory operations from all processors maintain a single sequential order

[Lamport'79]

Problems with SC Enforcement



Problems with SC Enforcement

- Low Performance

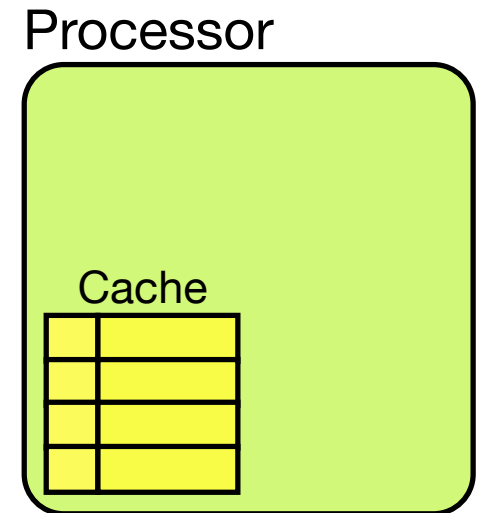
Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

Problems with SC Enforcement

- Low Performance
 - restrictions on performance-enhancing reordering of memory operations
- Or Complex Implementation



[Gharachorloo'91]
[Ranganathan'96]
[Gniady'97, SC++]

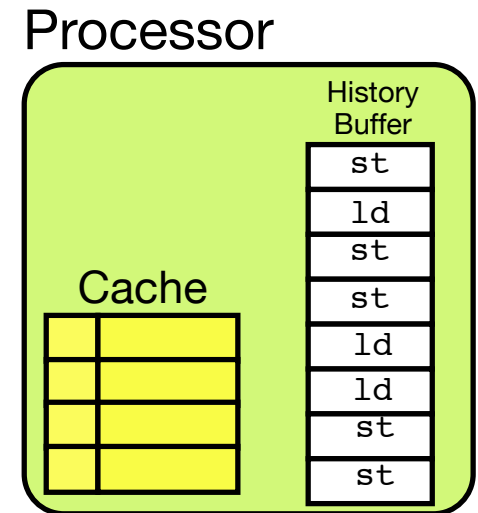
Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation

- buffer long history of speculative memory accesses



[Gharachorloo'91]

[Ranganathan'96]

[Gniady'97, SC++]

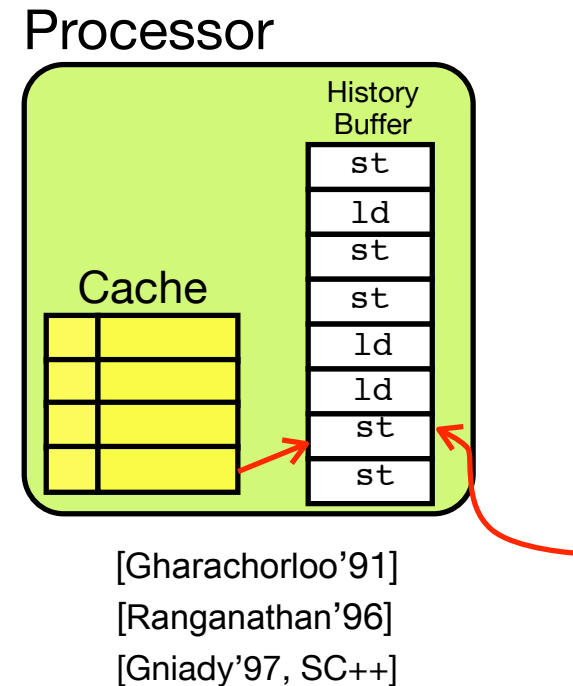
Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation

- buffer long history of speculative memory accesses
- check this history against coherence events and cache displacements



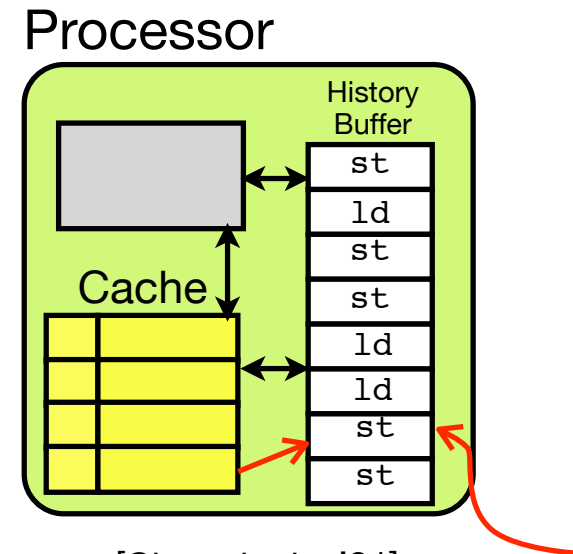
Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation

- buffer long history of speculative memory accesses
- check this history against coherence events and cache displacements
- coupled with key structures (LSQ, ROB, reg file, \$)



[Gharachorloo'91]

[Ranganathan'96]

[Gniady'97, SC++]

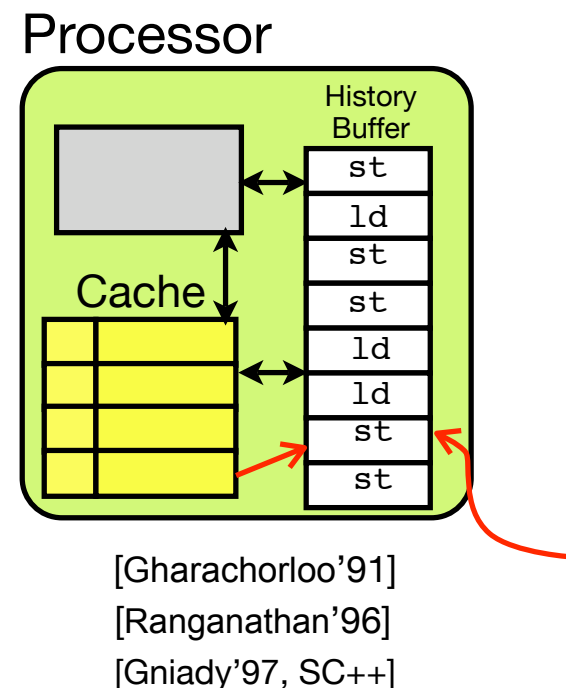
Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation

- buffer long history of speculative memory accesses
- check this history against coherence events and cache displacements
- coupled with key structures (LSQ, ROB, reg file, \$)
- typically fine-grain (instruction-level) undo



Problems with SC Enforcement

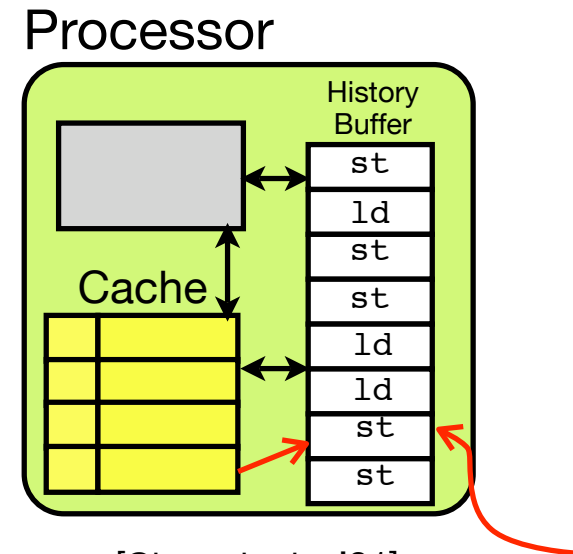
- Low Performance

- restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation

- buffer long history of speculative memory accesses
- check this history against coherence events and cache displacements
- coupled with key structures (LSQ, ROB, reg file, \$)
- typically fine-grain (instruction-level) undo

- Most current systems do not support SC



[Gharachorloo'91]

[Ranganathan'96]

[Gniady'97, SC++]

Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

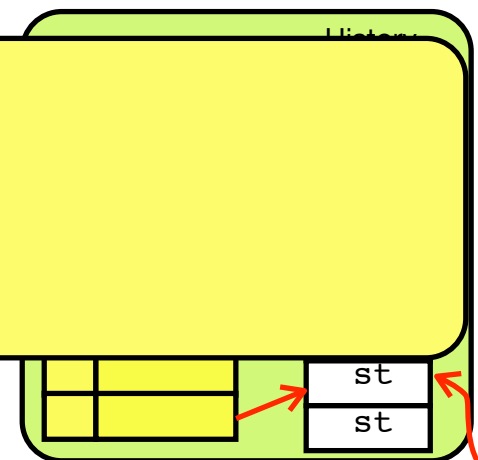
- Or Complex Implementation

→ We would like to change that!

- coupled with key structures (LSQ, ROB, reg file, \$)
- typically fine-grain (instruction-level) undo

- Most current systems do not support SC

Processor



[Gharachorloo'91]

[Ranganathan'96]

[Gniady'97, SC++]

Problems with SC Enforcement

- Low Performance

- restrictions on performance-enhancing reordering of memory operations

- Or Complex Implementation

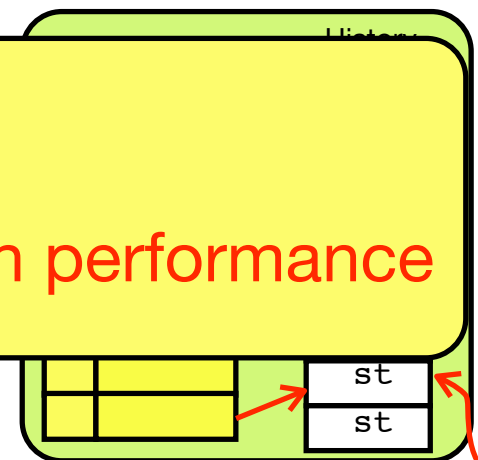
➡ We would like to change that!

➡ Support SC with simple hardware and high performance

- coupled with key structures (LSQ, ROB, reg file, \$)
- typically fine-grain (instruction-level) undo

- Most current systems do not support SC

Processor



[Gharachorloo'91]

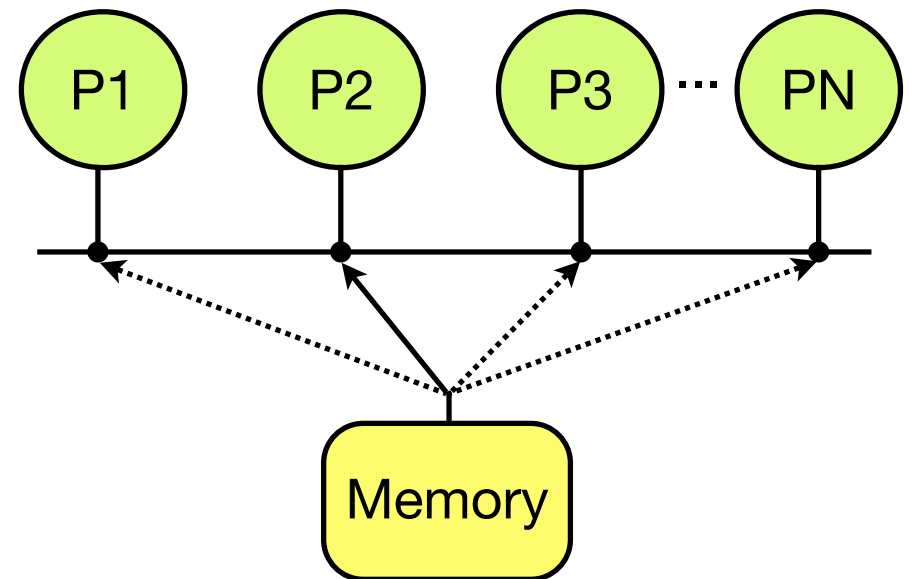
[Ranganathan'96]

[Gniady'97, SC++]

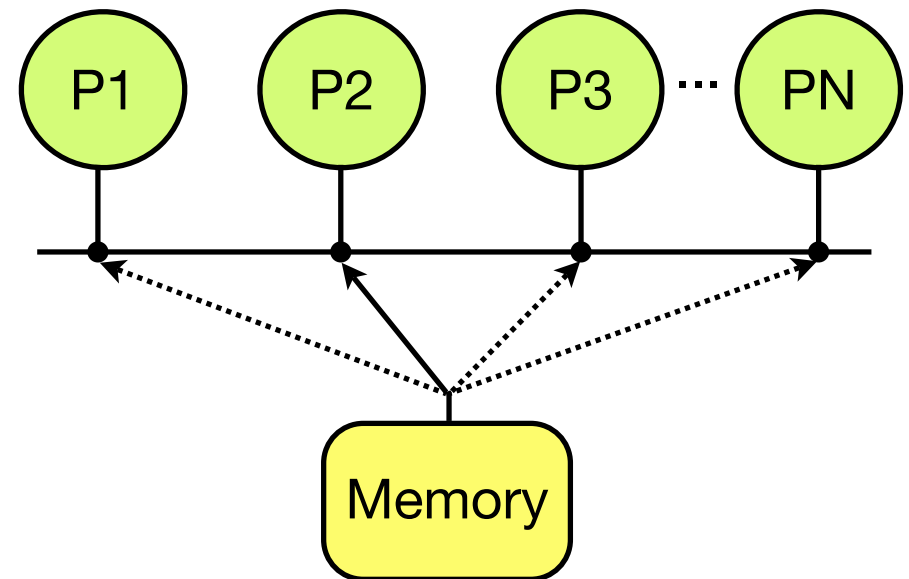
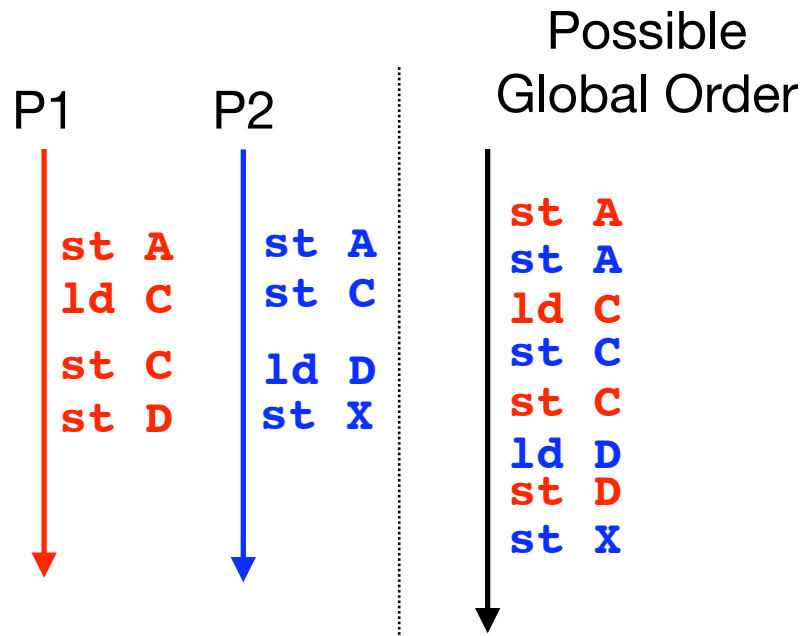
BulkSC: Bulk Enforcement of SC



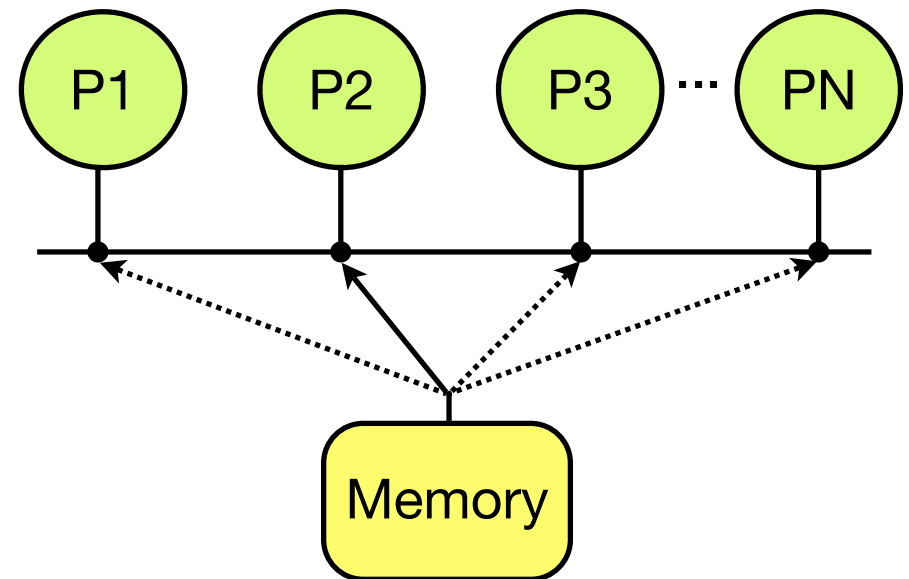
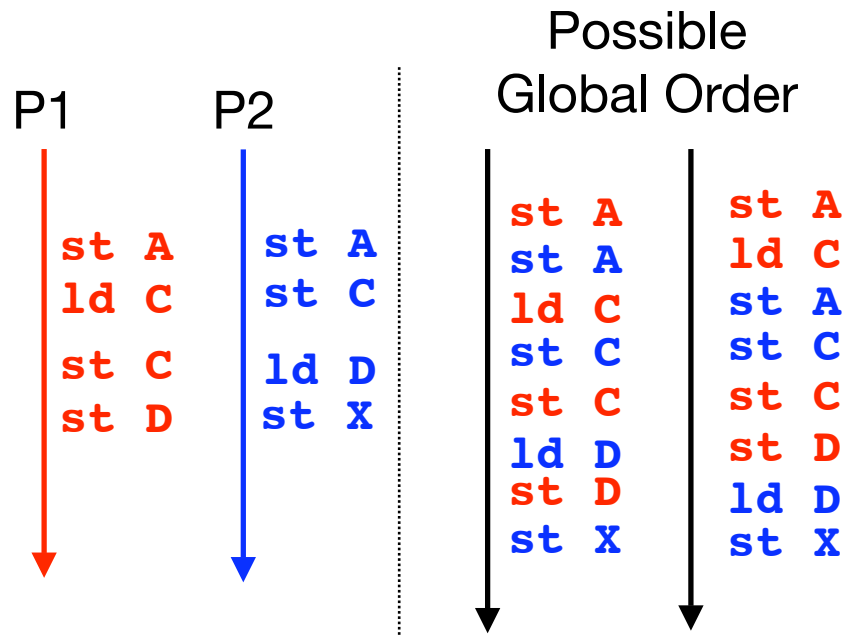
BulkSC: Bulk Enforcement of SC



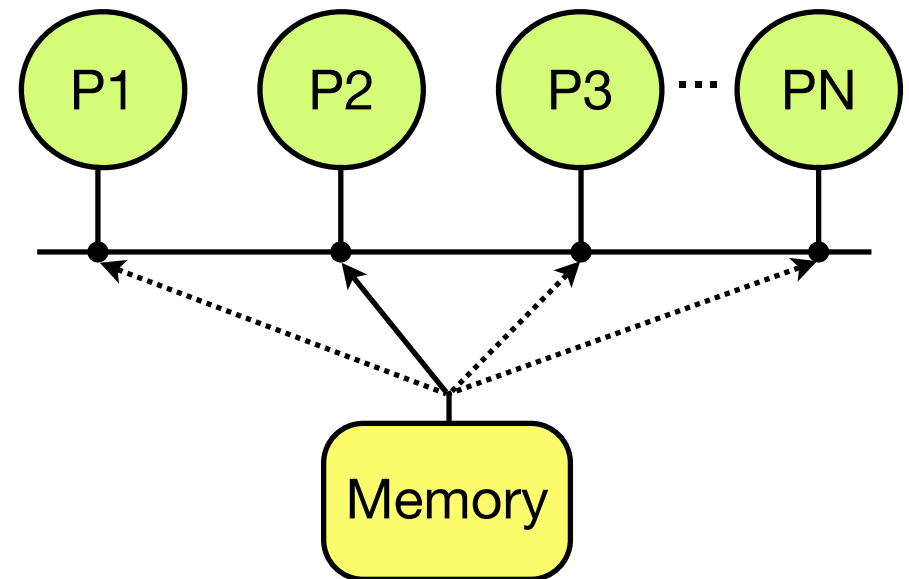
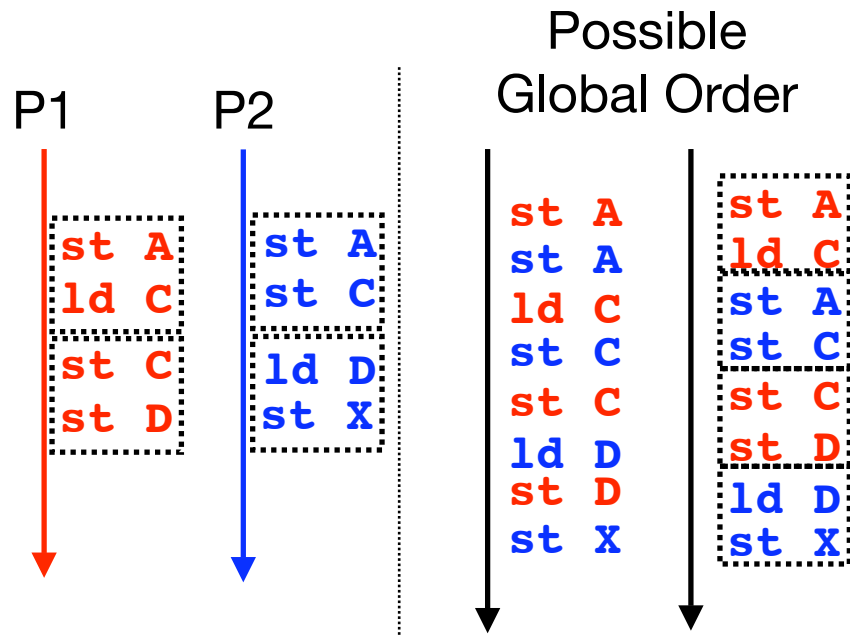
BulkSC: Bulk Enforcement of SC



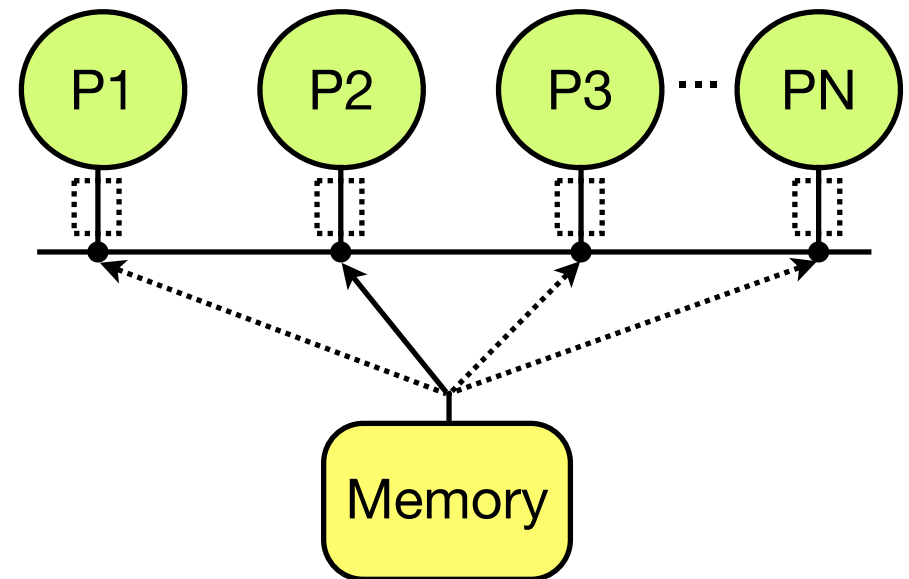
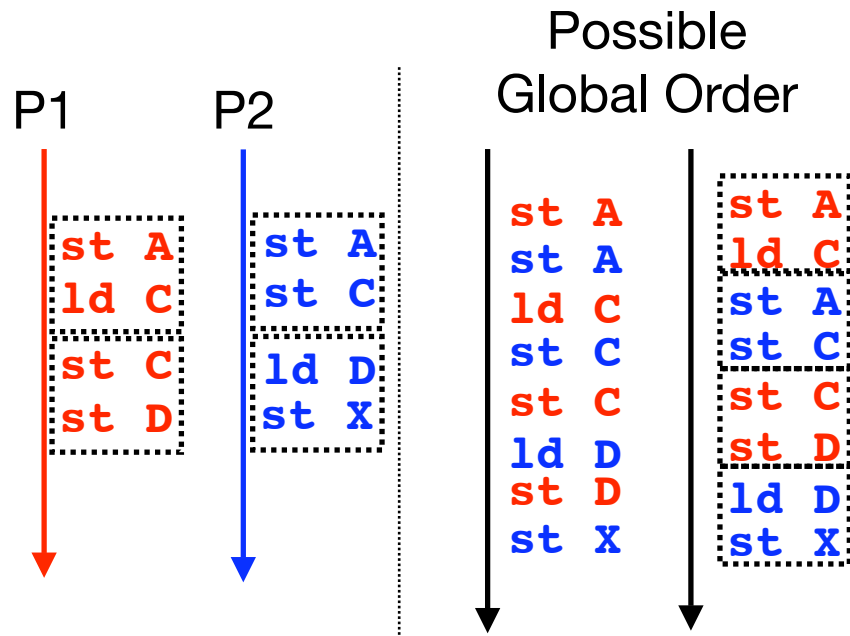
BulkSC: Bulk Enforcement of SC



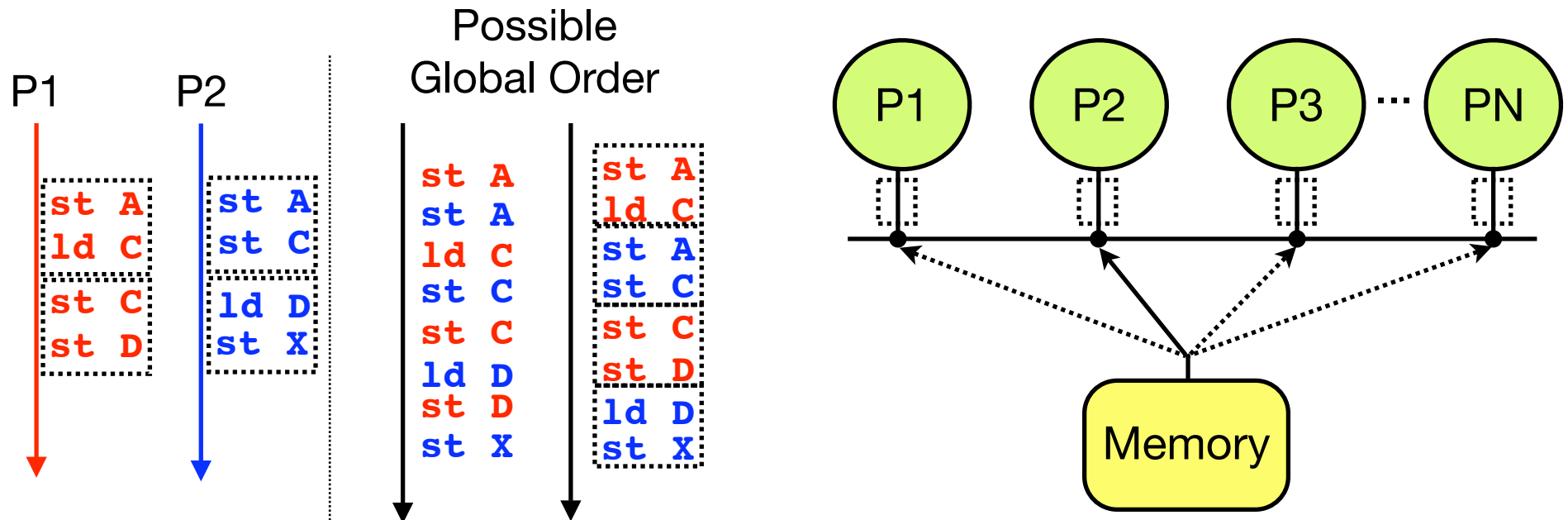
BulkSC: Bulk Enforcement of SC



BulkSC: Bulk Enforcement of SC

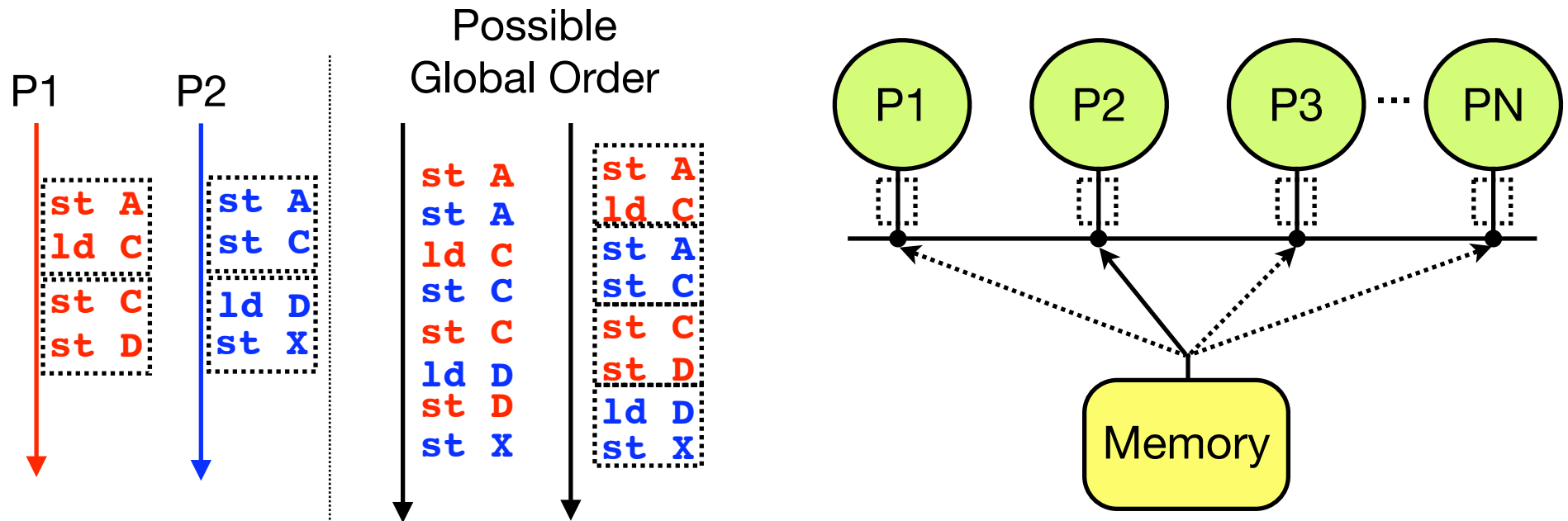


BulkSC: Bulk Enforcement of SC



➡ Group instructions into Chunks, enforce SC only at Chunk granularity

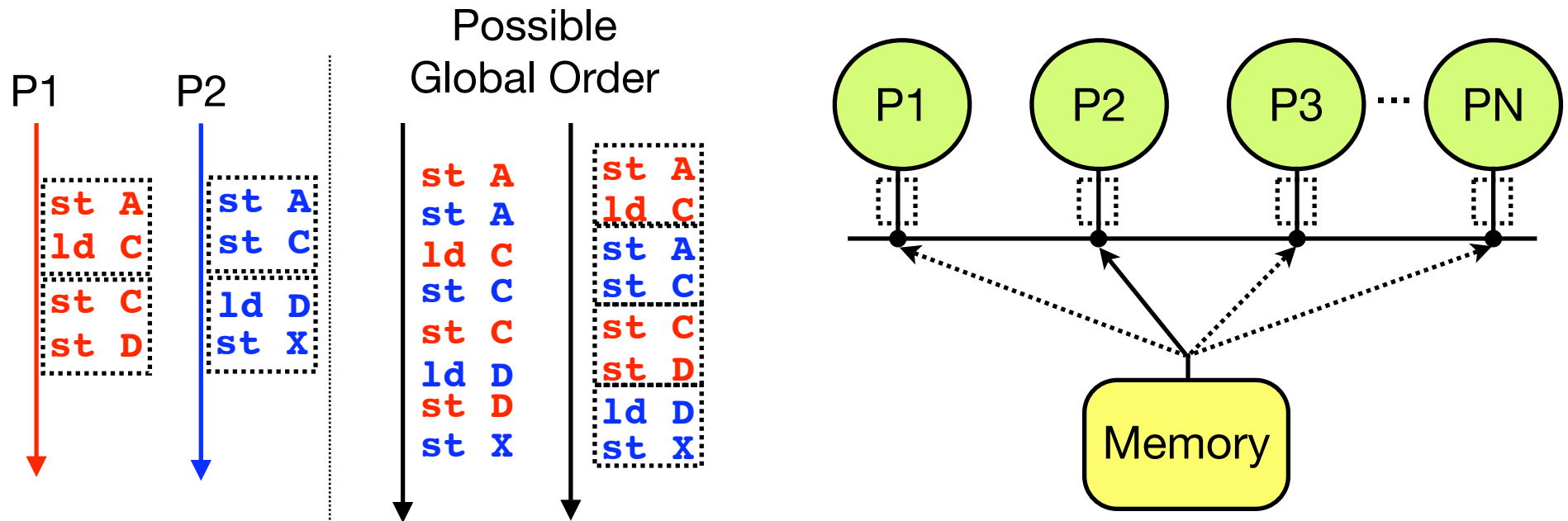
BulkSC: Bulk Enforcement of SC



➡ Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially **reduce hardware complexity**

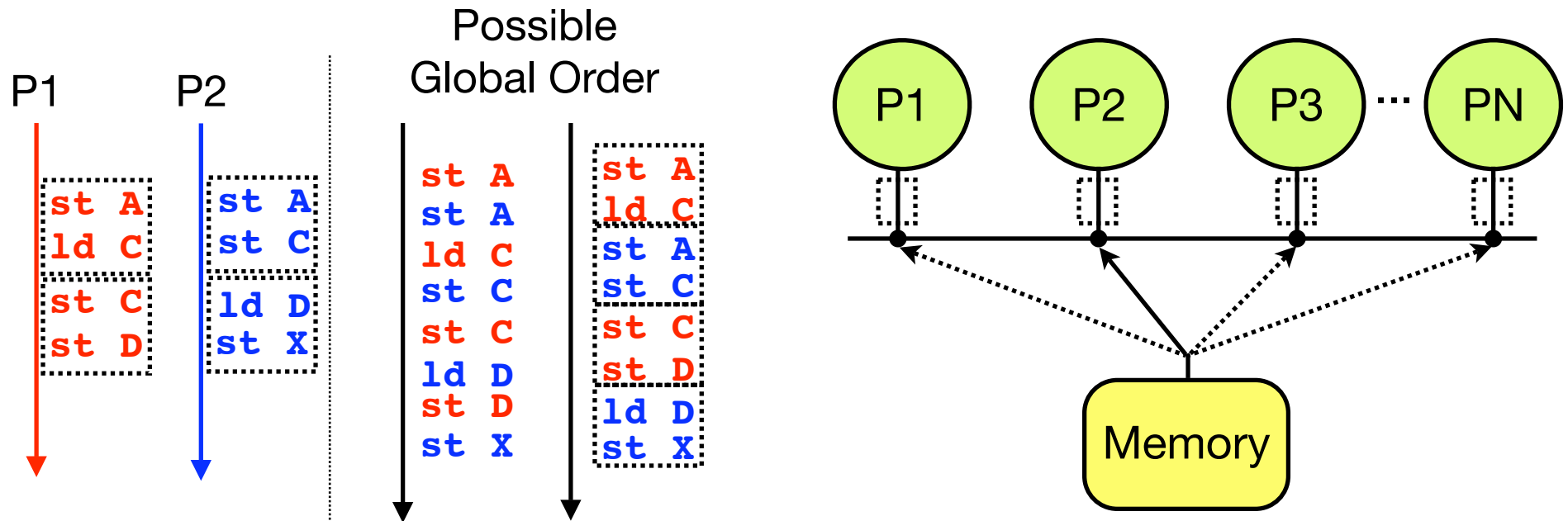
BulkSC: Bulk Enforcement of SC



➡ Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially **reduce hardware complexity**
2. enable **high performance**

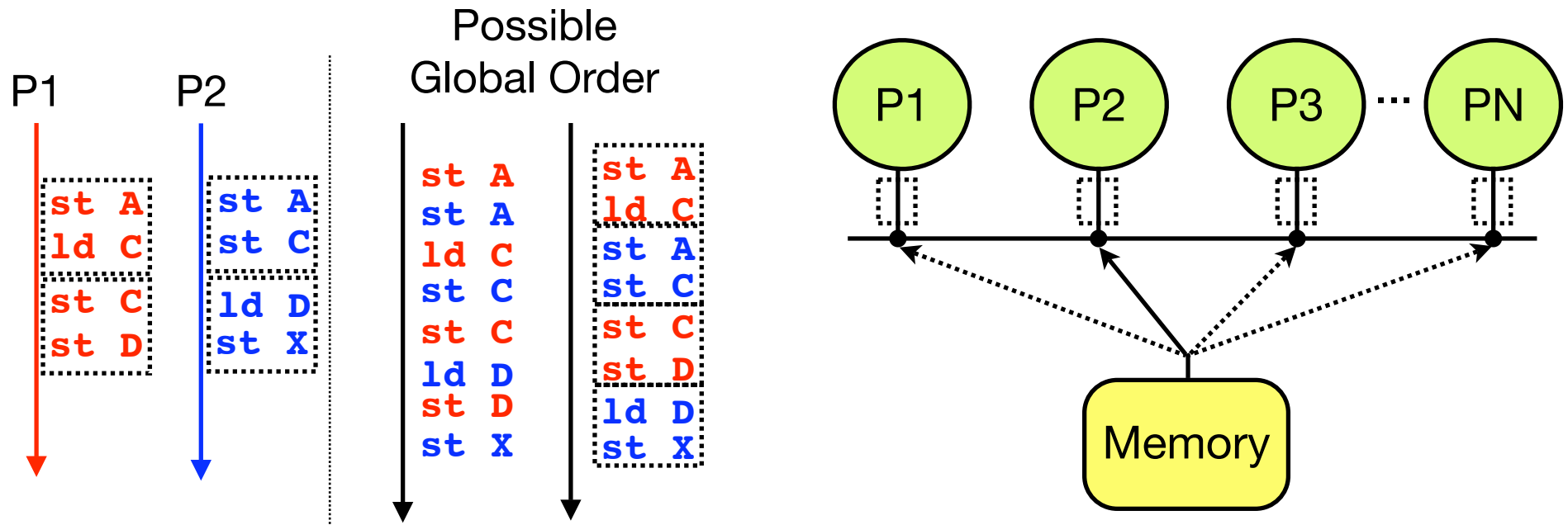
BulkSC: Bulk Enforcement of SC



➡ Group instructions into Chunks, enforce SC only at Chunk granularity

1. substantially **reduce hardware complexity**
2. enable **high performance**
3. retain **programmability**

BulkSC: Bulk Enforcement of SC

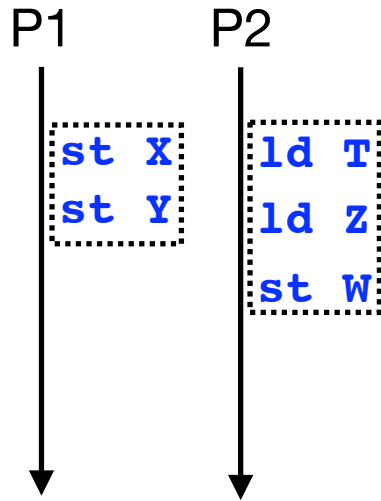


- ➡ Group instructions into Chunks, enforce SC only at Chunk granularity
 1. substantially **reduce hardware complexity**
 2. enable **high performance**
 3. retain **programmability**
- Execute a chunk **atomically** and in **isolation**, like a **single instruction**

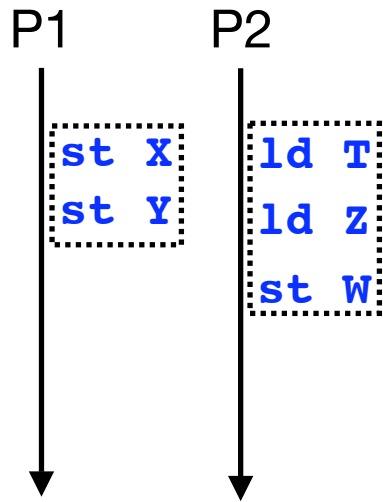
Chunk Execution: Atomicity and Isolation



Chunk Execution: Atomicity and Isolation

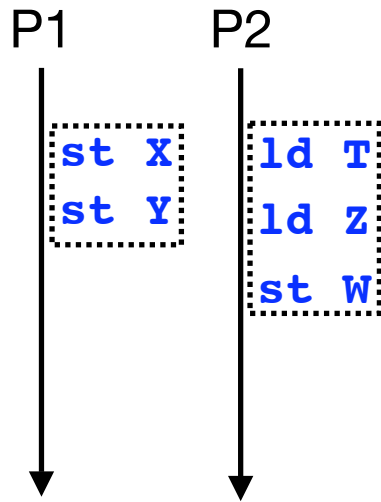


Chunk Execution: Atomicity and Isolation



Speculative
Execution

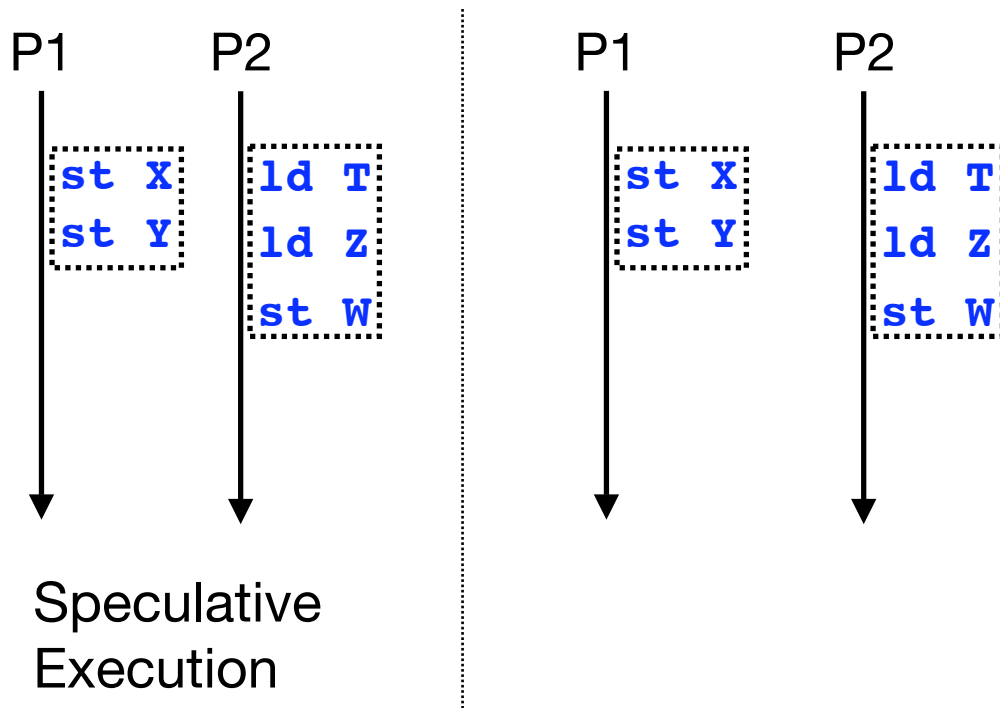
Chunk Execution: Atomicity and Isolation



Speculative
Execution

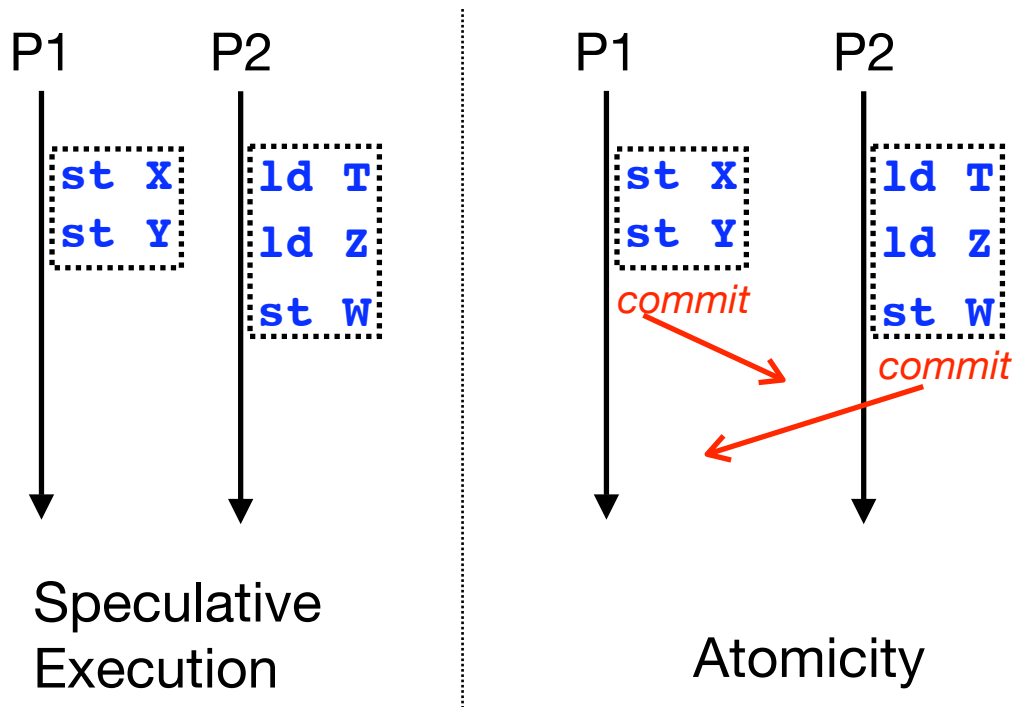
Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Chunk Execution: Atomicity and Isolation



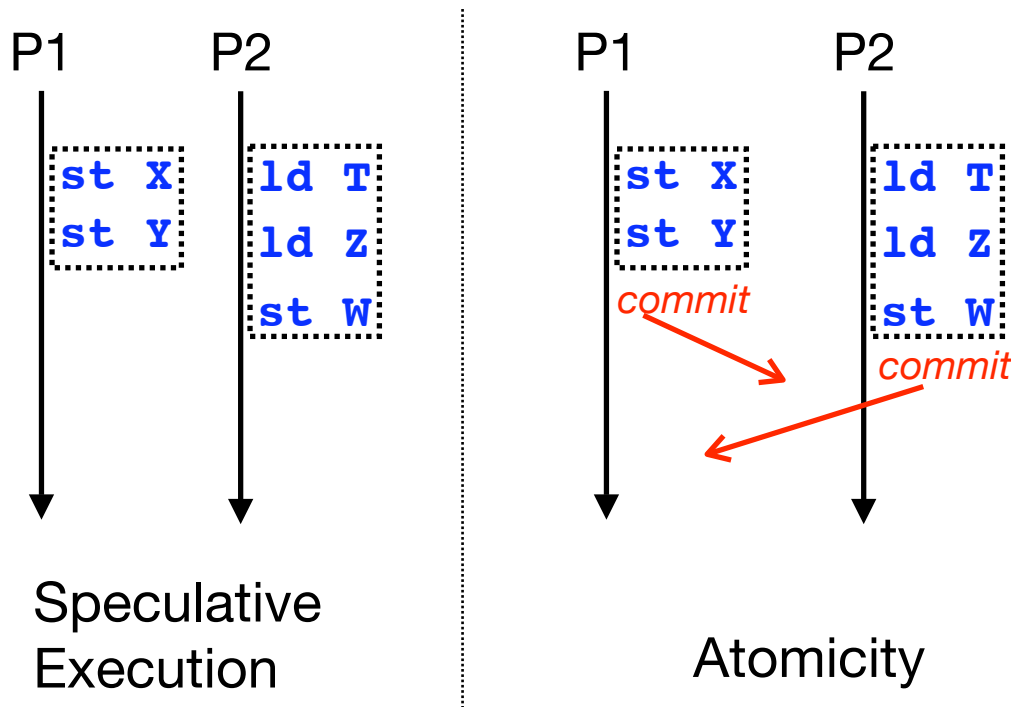
Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Chunk Execution: Atomicity and Isolation



Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

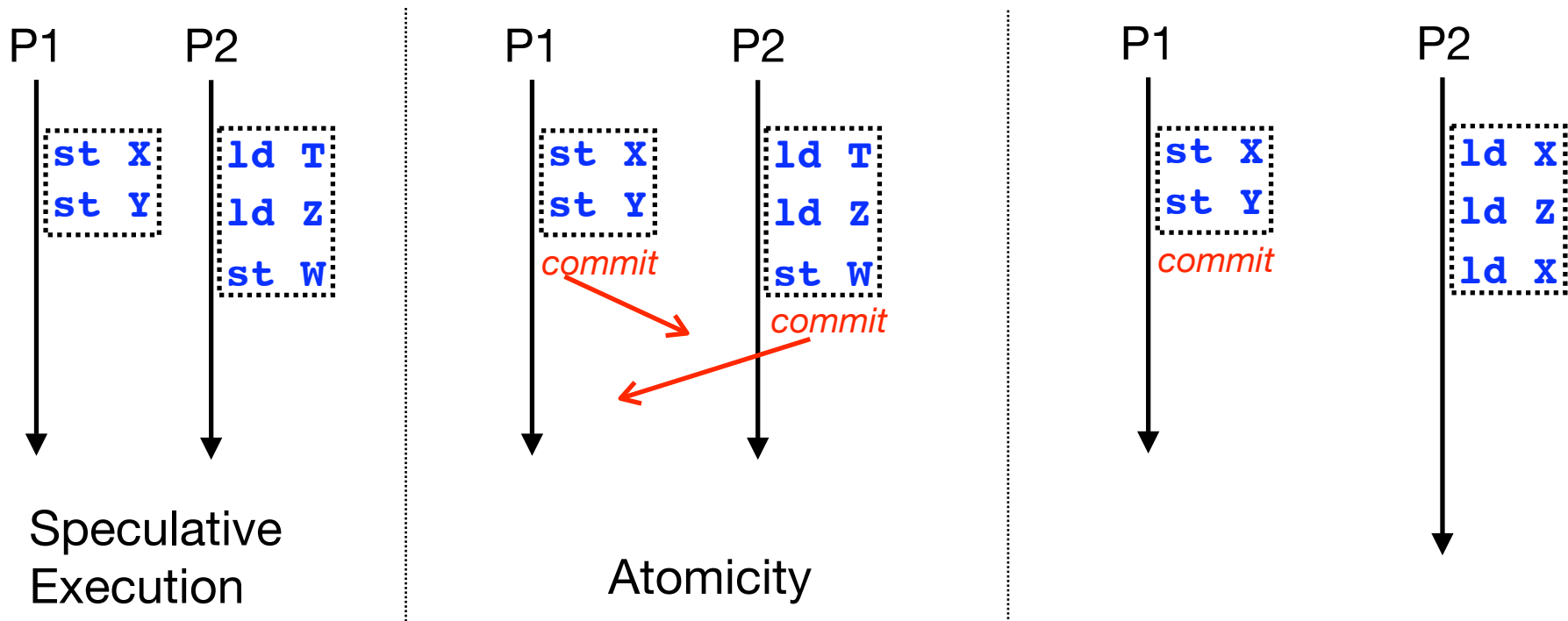
Chunk Execution: Atomicity and Isolation



Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Isolation: a chunk should not see “outside” state changing during its execution

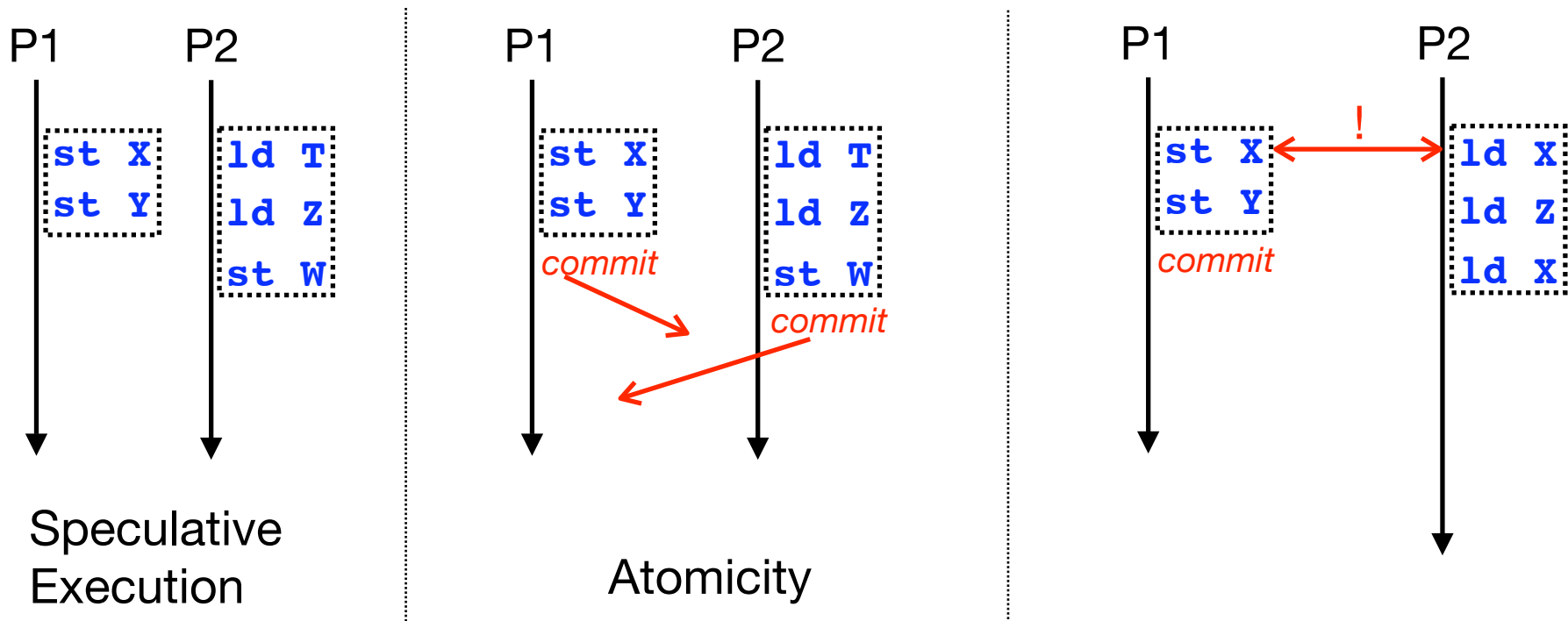
Chunk Execution: Atomicity and Isolation



Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Isolation: a chunk should not see “outside” state changing during its execution

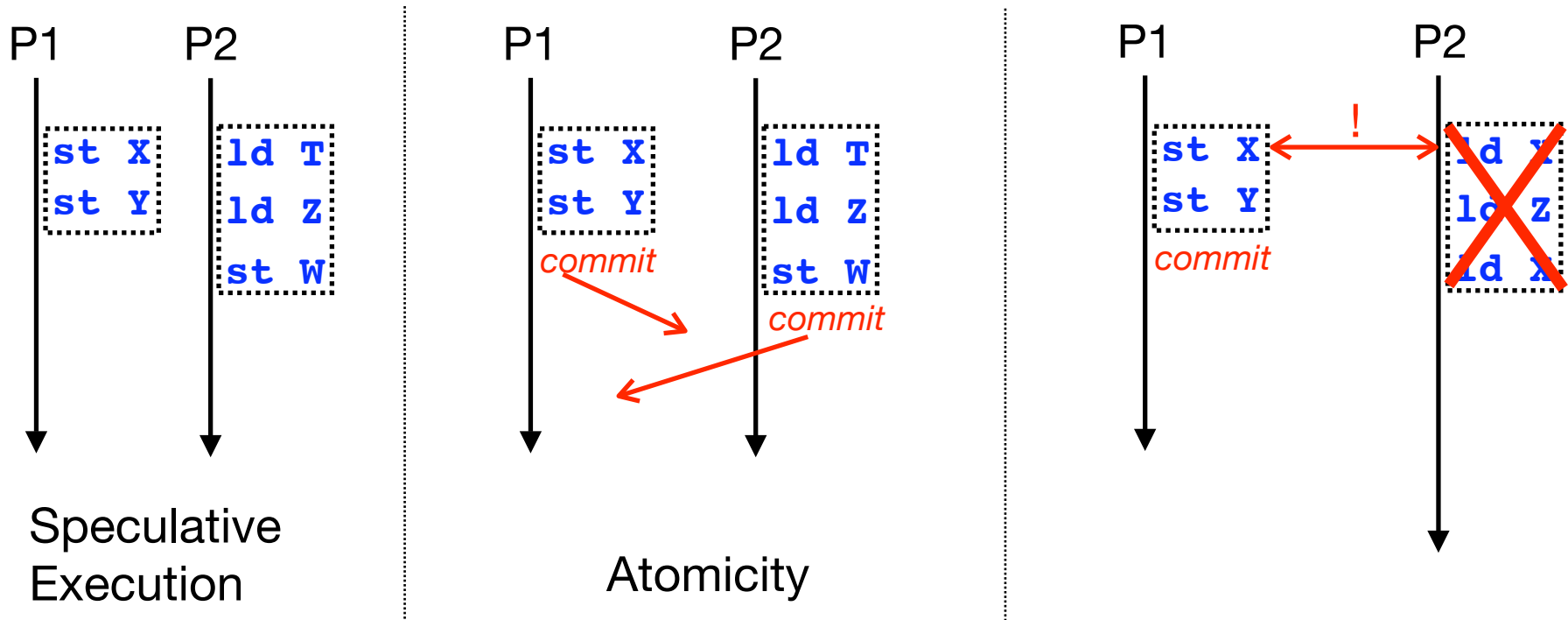
Chunk Execution: Atomicity and Isolation



Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Isolation: a chunk should not see “outside” state changing during its execution

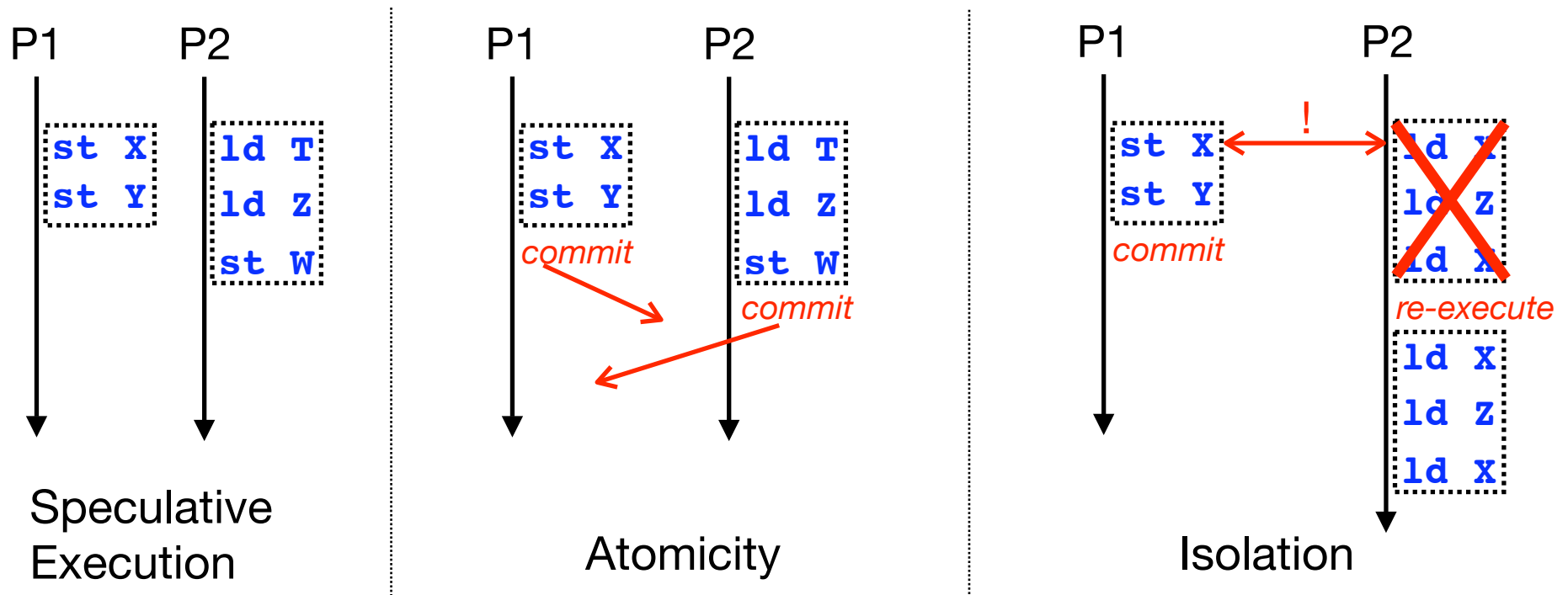
Chunk Execution: Atomicity and Isolation



Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Isolation: a chunk should not see “outside” state changing during its execution

Chunk Execution: Atomicity and Isolation



Atomicity: all updates in the chunk are made visible to other processors at once (all or nothing)

Isolation: a chunk should not see “outside” state changing during its execution

Requirements for Bulk Enforcement of SC



Requirements for Bulk Enforcement of SC

Per-processor program order:
chunks from individual processors
maintain program order

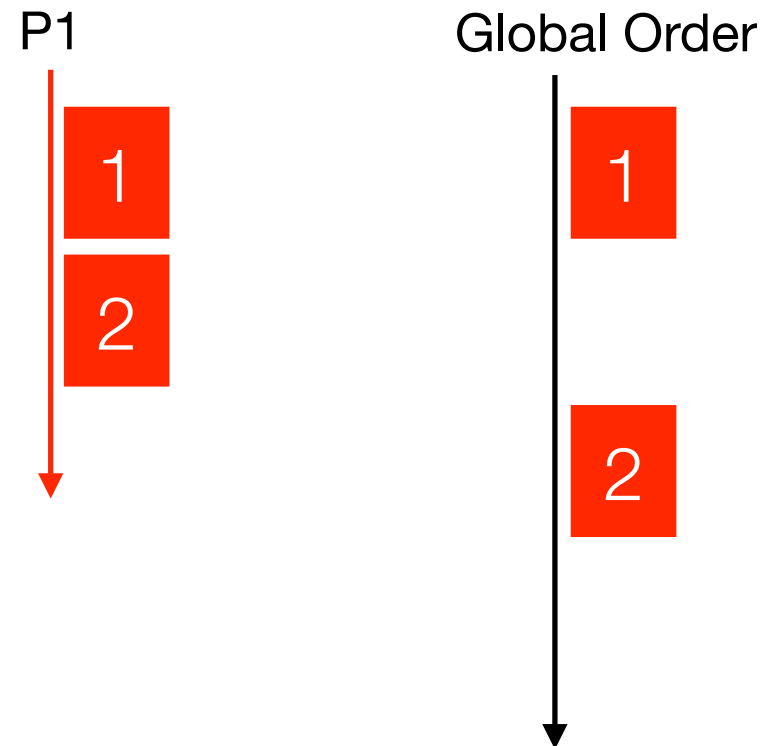
Requirements for Bulk Enforcement of SC

Per-processor program order:
chunks from individual processors
maintain program order



Requirements for Bulk Enforcement of SC

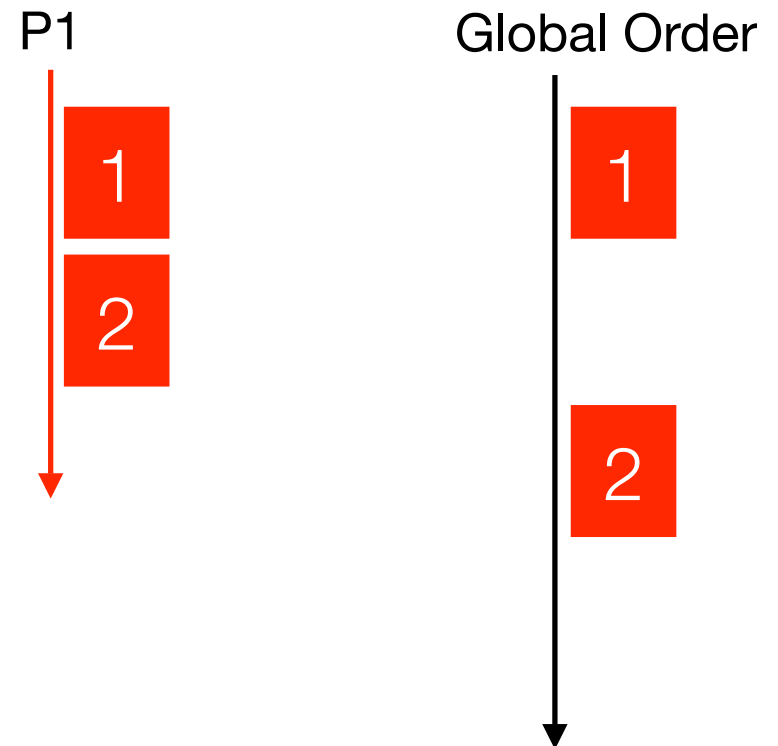
Per-processor program order:
chunks from individual processors
maintain program order



Requirements for Bulk Enforcement of SC

Per-processor program order:
chunks from individual processors
maintain program order

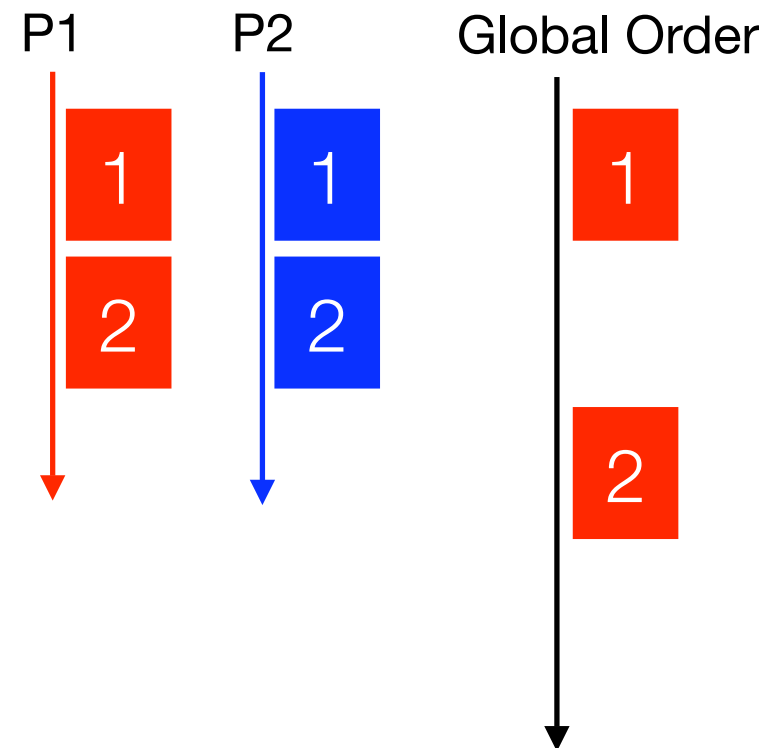
Single sequential order: *chunks*
from all processors maintain a
single sequential order



Requirements for Bulk Enforcement of SC

Per-processor program order:
chunks from individual processors
maintain program order

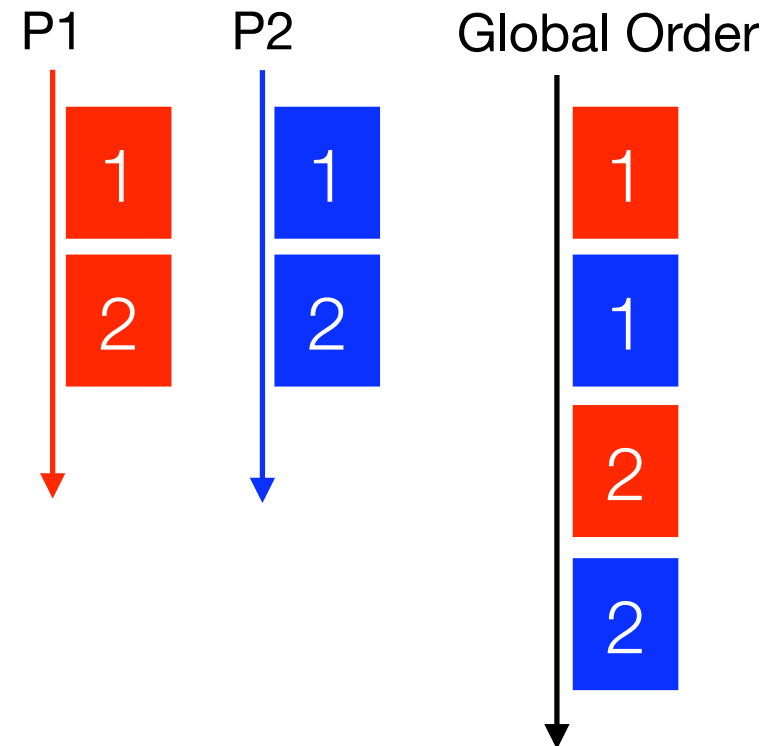
Single sequential order: *chunks*
from all processors maintain a
single sequential order



Requirements for Bulk Enforcement of SC

Per-processor program order:
chunks from individual processors
maintain program order

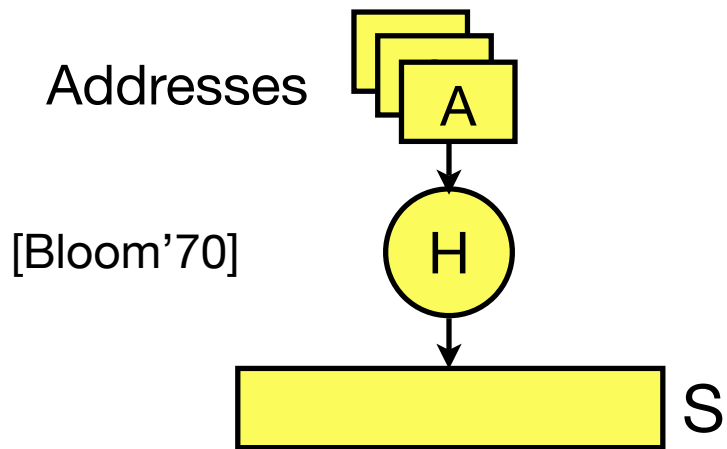
Single sequential order: *chunks*
from all processors maintain a
single sequential order



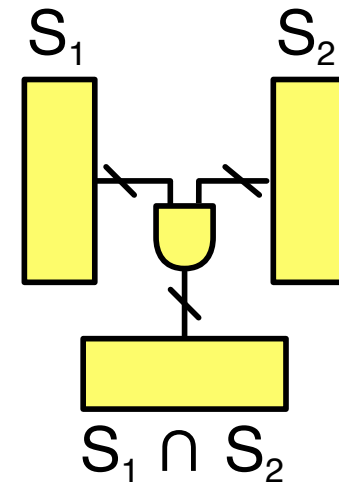
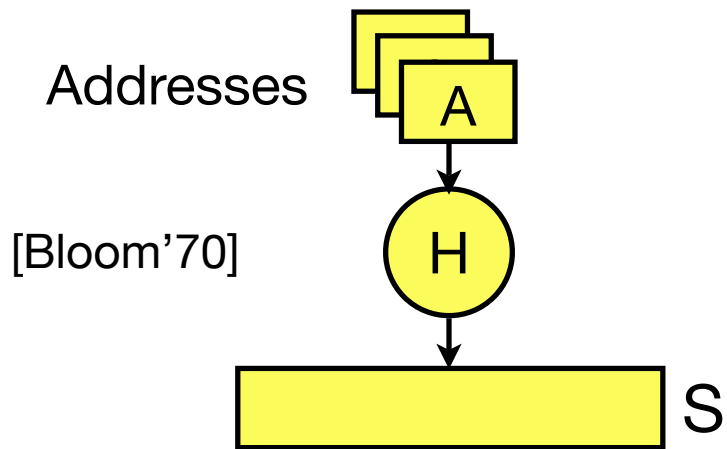
Bulk Operation [ISCA'06]



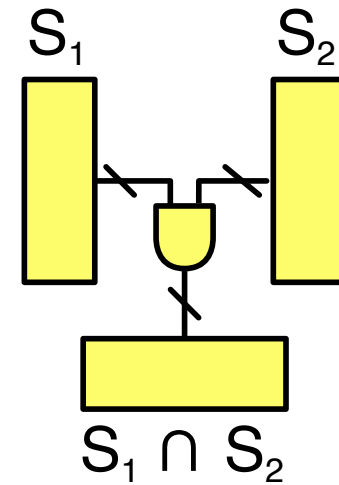
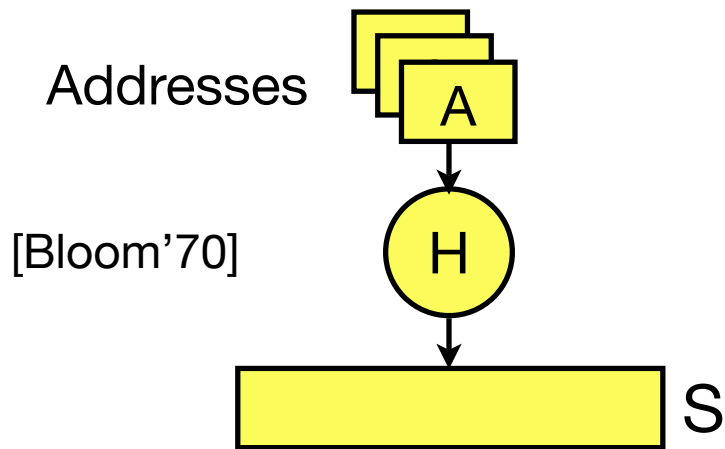
Bulk Operation [ISCA'06]



Bulk Operation [ISCA'06]

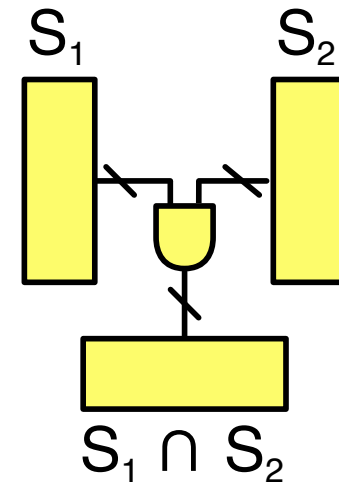
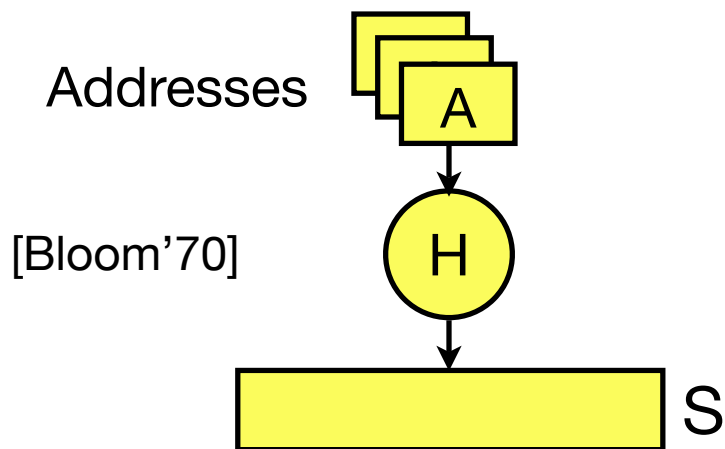


Bulk Operation [ISCA'06]



Signature Operations

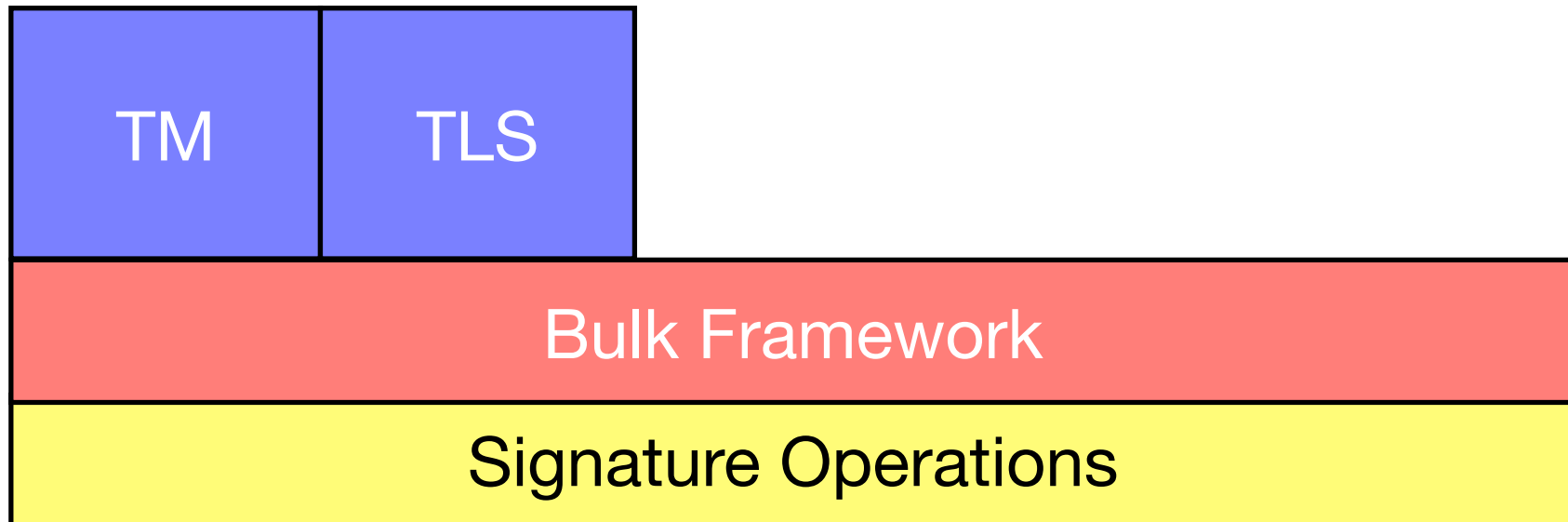
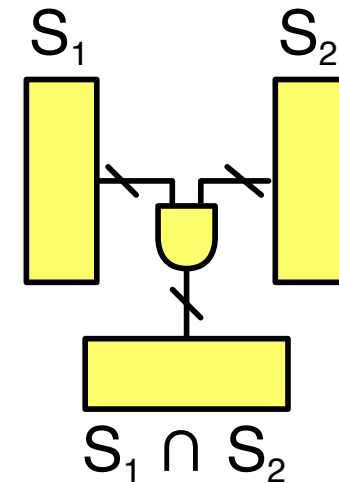
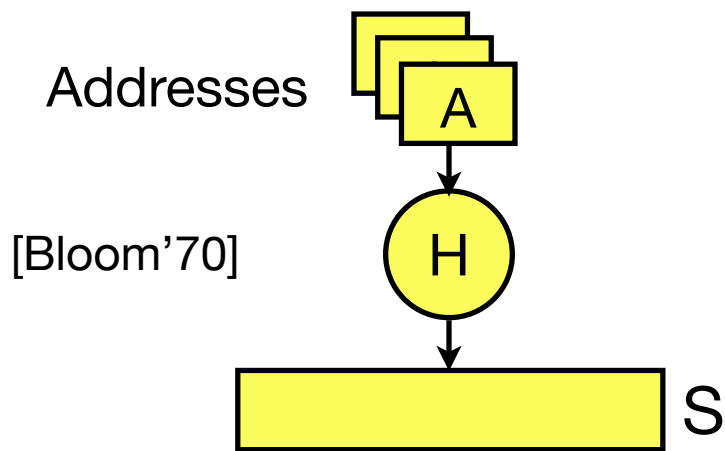
Bulk Operation [ISCA'06]



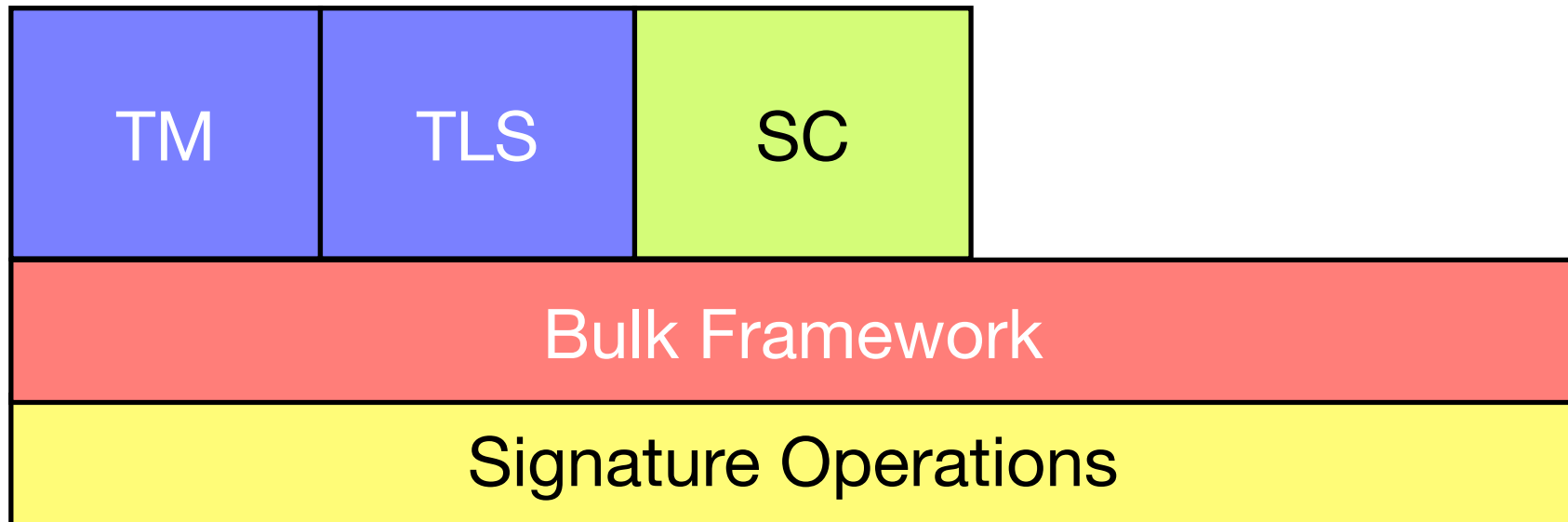
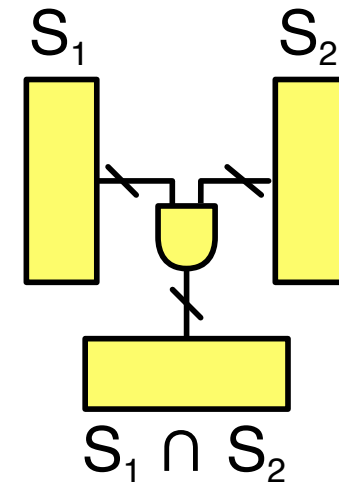
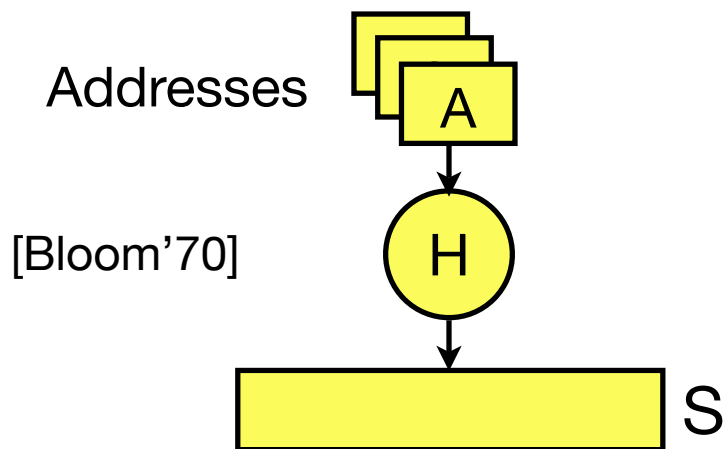
Bulk Framework

Signature Operations

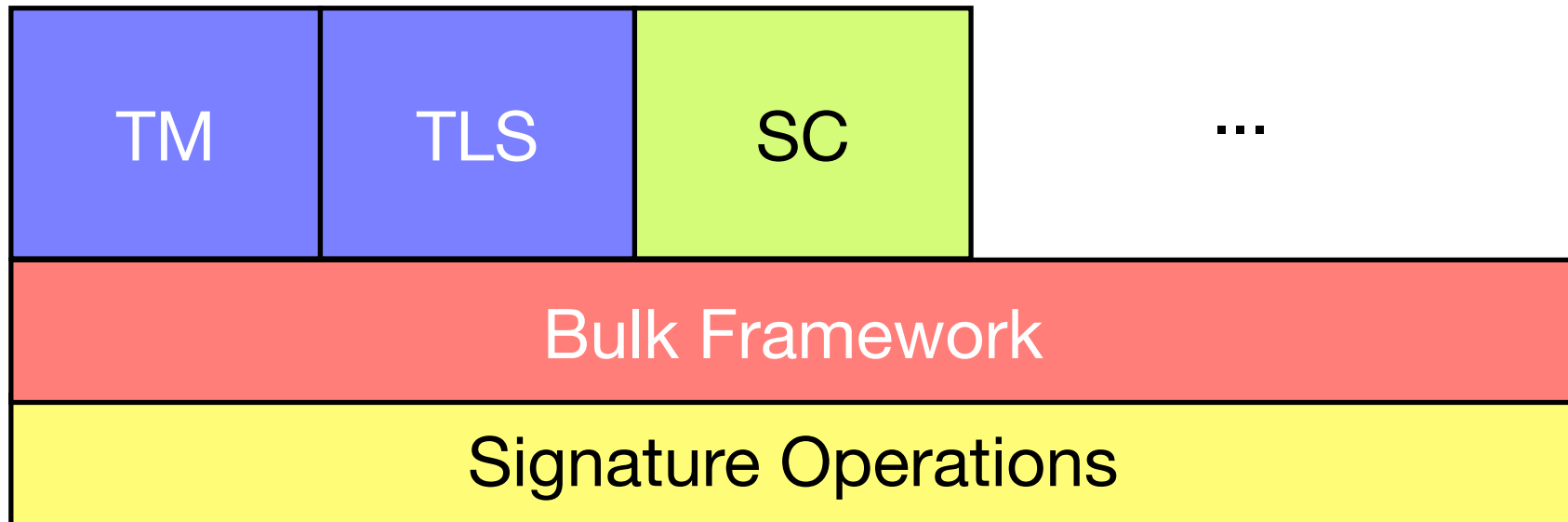
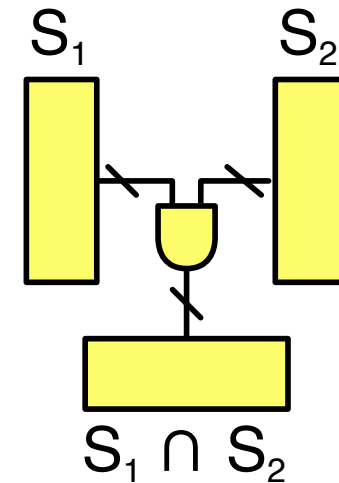
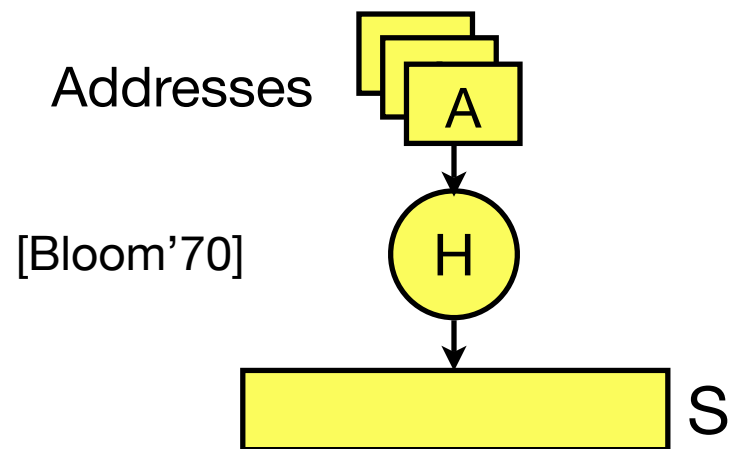
Bulk Operation [ISCA'06]



Bulk Operation [ISCA'06]



Bulk Operation [ISCA'06]



Efficiently Operating with Chunks



Efficiently Operating with Chunks

- Key idea: Summarize **in hardware** addresses accessed by a chunk into a pair of signatures

Efficiently Operating with Chunks

- Key idea: Summarize **in hardware** addresses accessed by a chunk into a pair of signatures

Read	
Write	

Efficiently Operating with Chunks

- Key idea: Summarize **in hardware** addresses accessed by a chunk into a pair of signatures



- Support signature operations with simple hardware
 - intersection, union, is-empty?, ...
 - much of the system operation is based on signatures

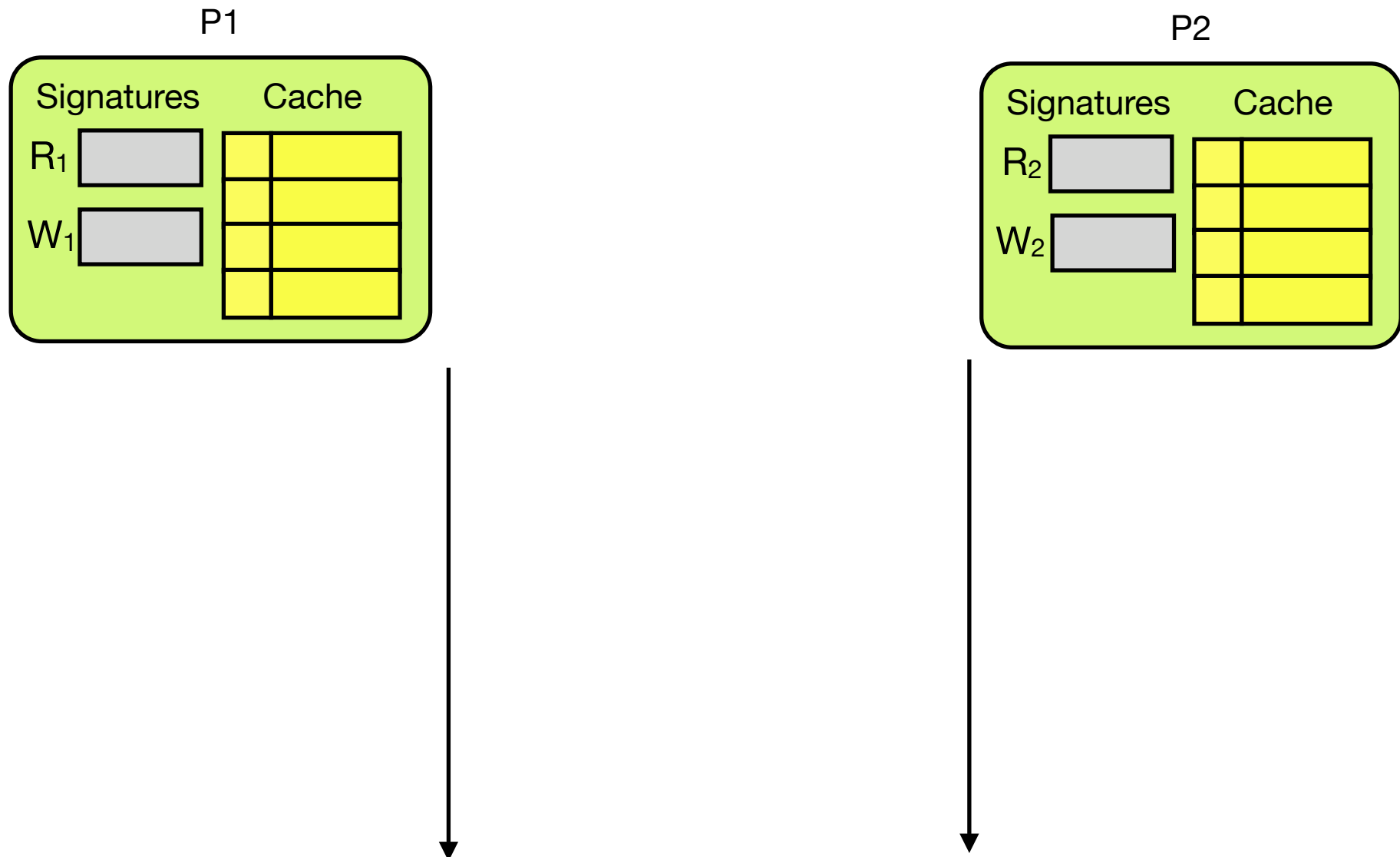
Efficiently Operating with Chunks

- Key idea: Summarize **in hardware** addresses accessed by a chunk into a pair of signatures

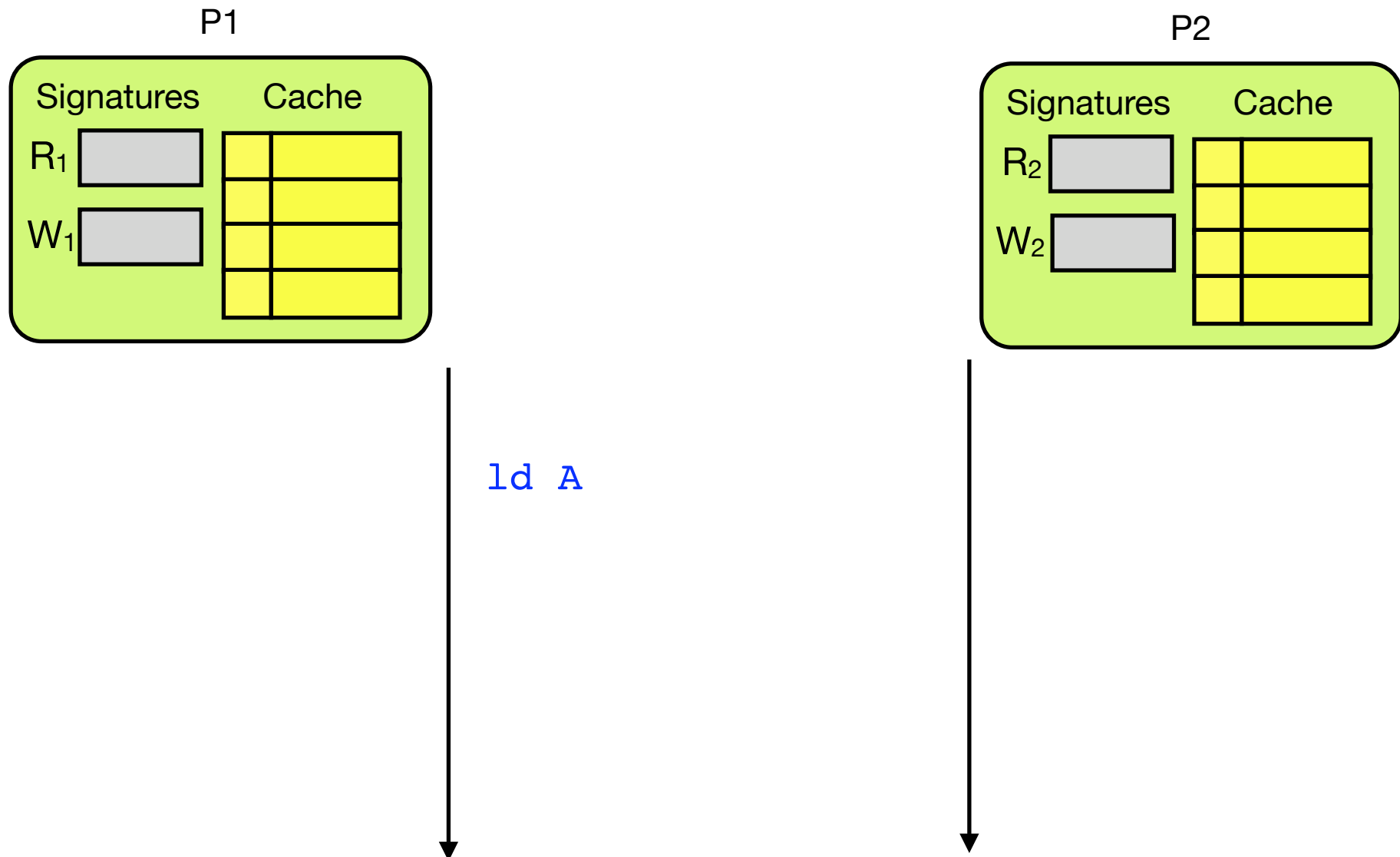


- Support signature operations with simple hardware
 - intersection, union, is-empty?, ...
 - much of the system operation is based on signatures
- At chunk commit
 - W signature is sent to other processors for disambiguation
 - R, W signatures are cleared

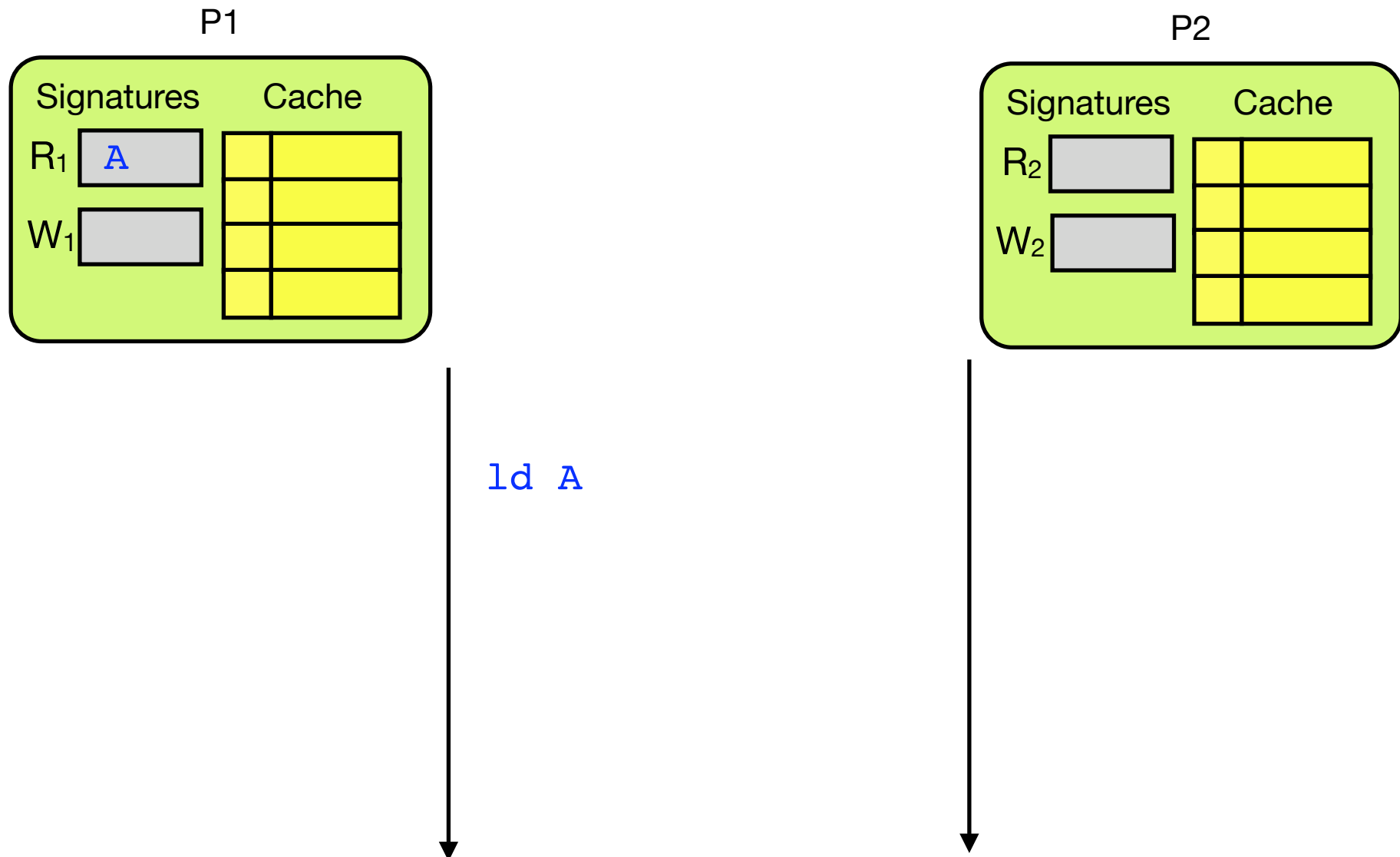
Chunk Atomicity and Isolation in Bulk



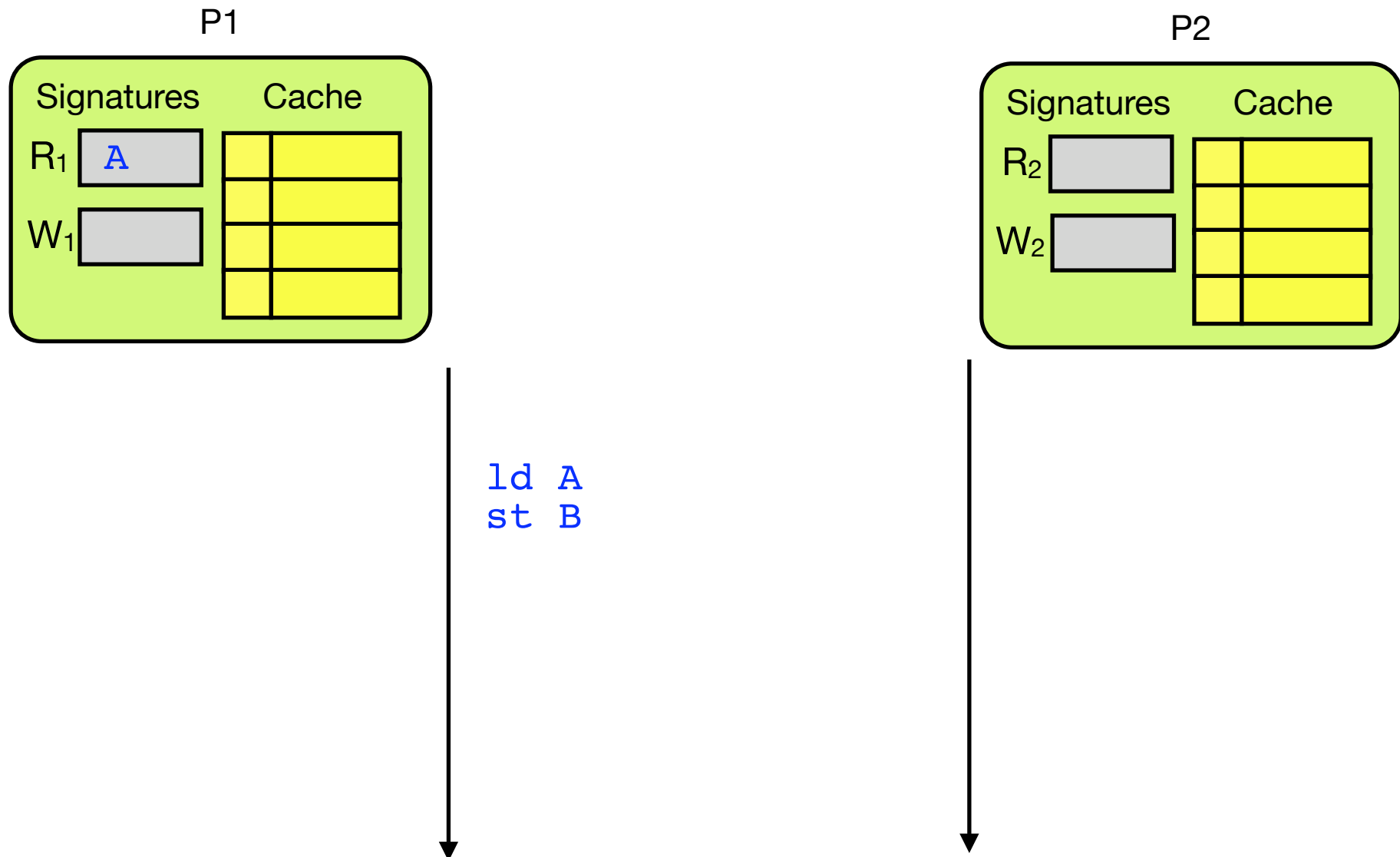
Chunk Atomicity and Isolation in Bulk



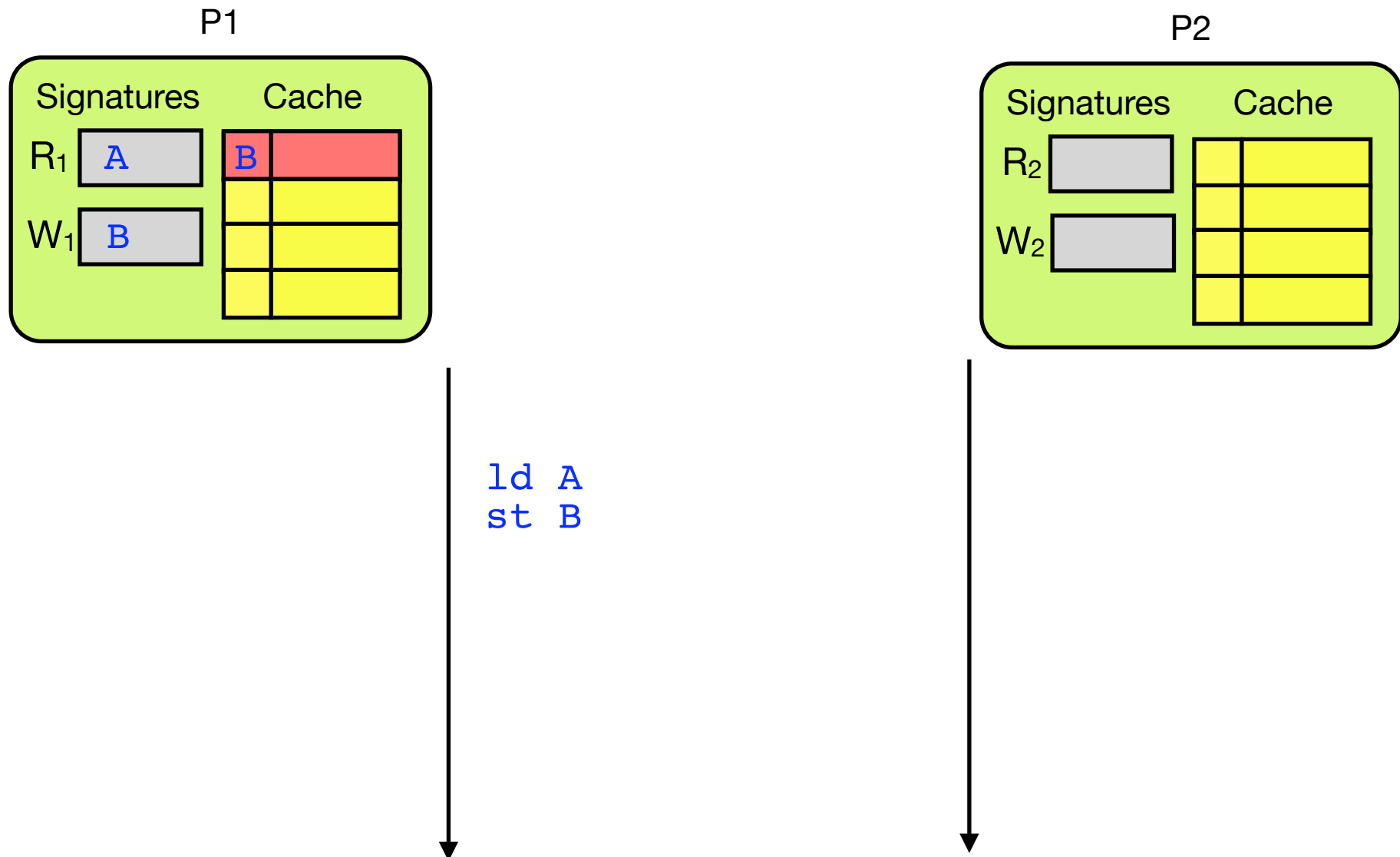
Chunk Atomicity and Isolation in Bulk



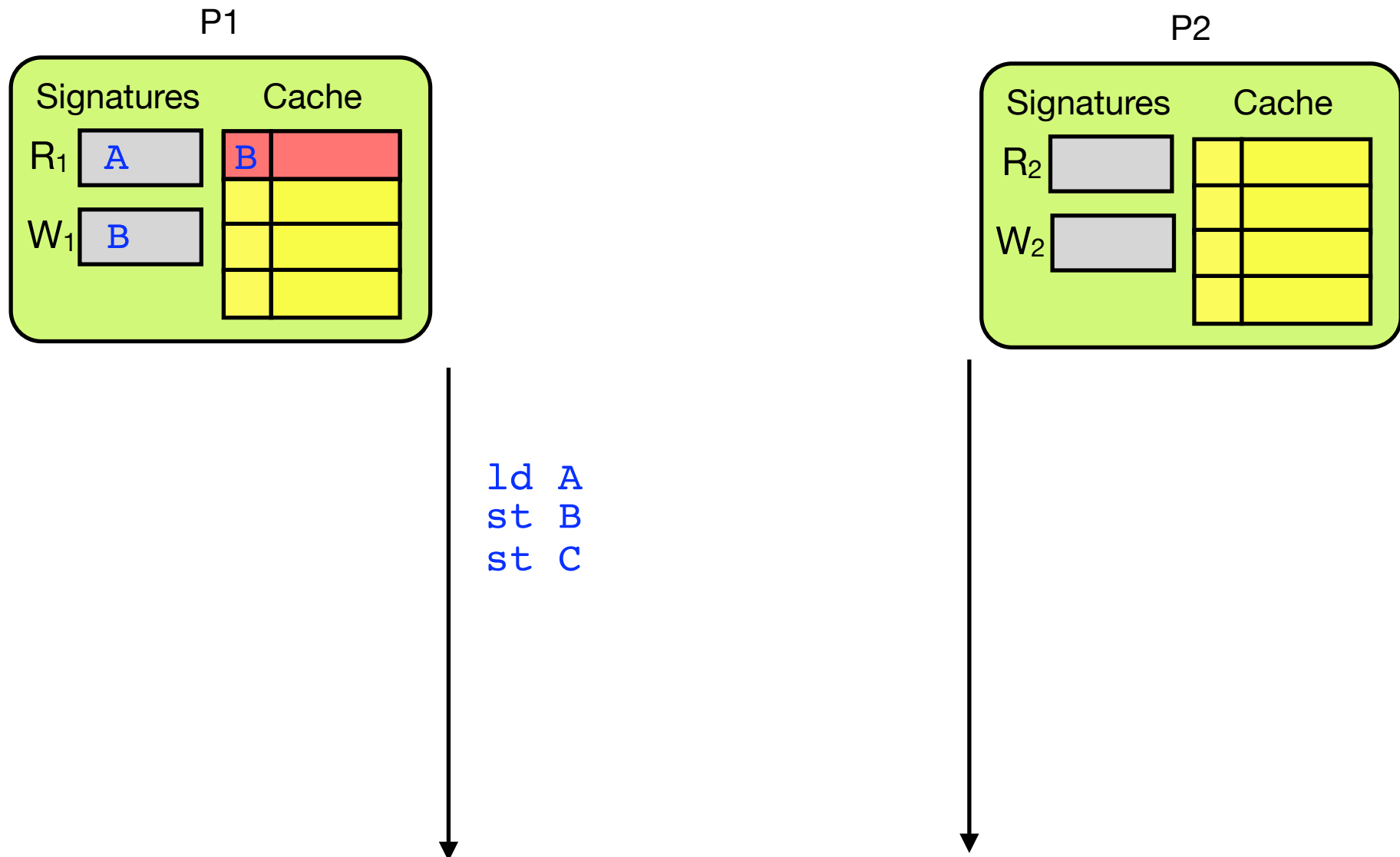
Chunk Atomicity and Isolation in Bulk



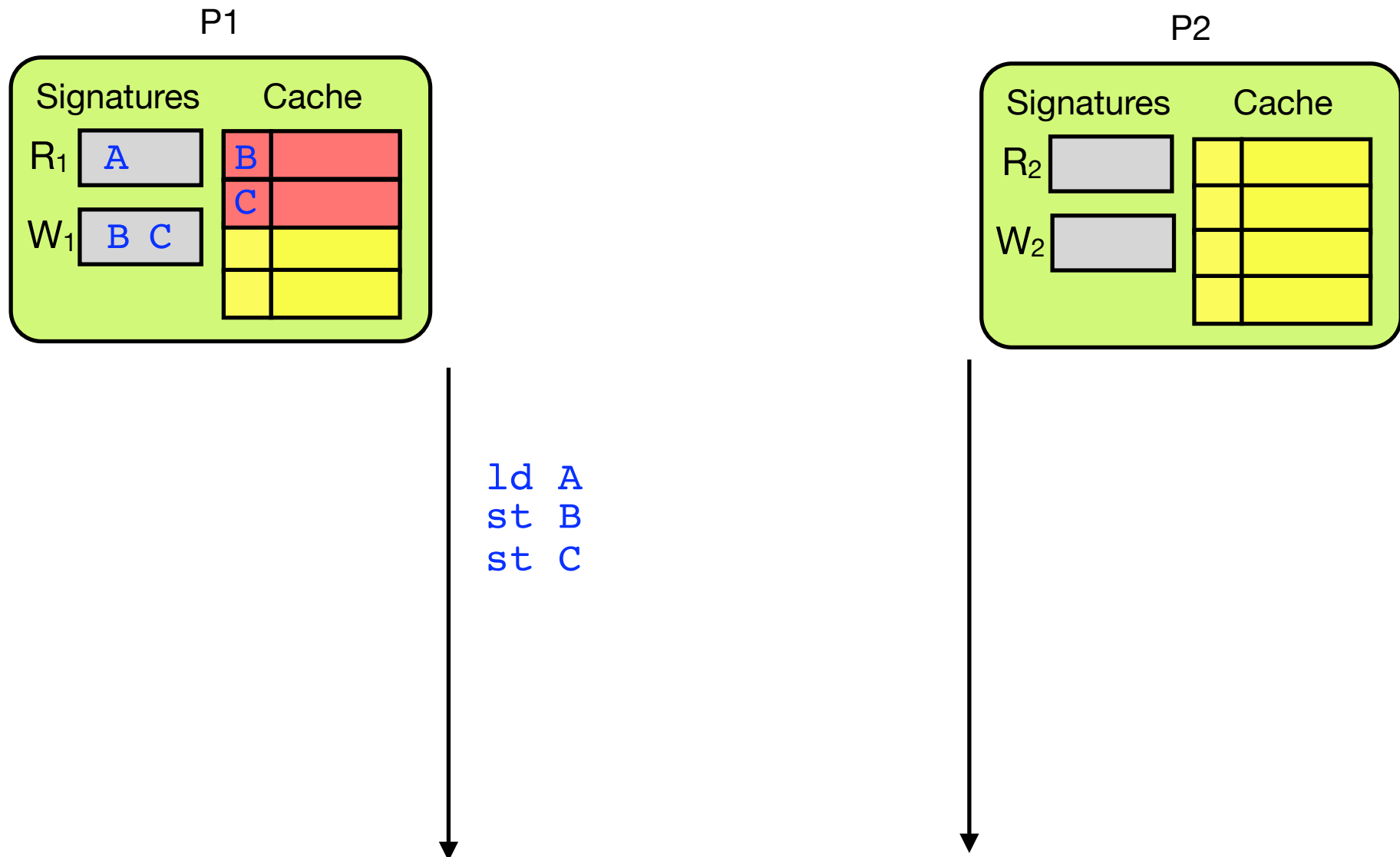
Chunk Atomicity and Isolation in Bulk



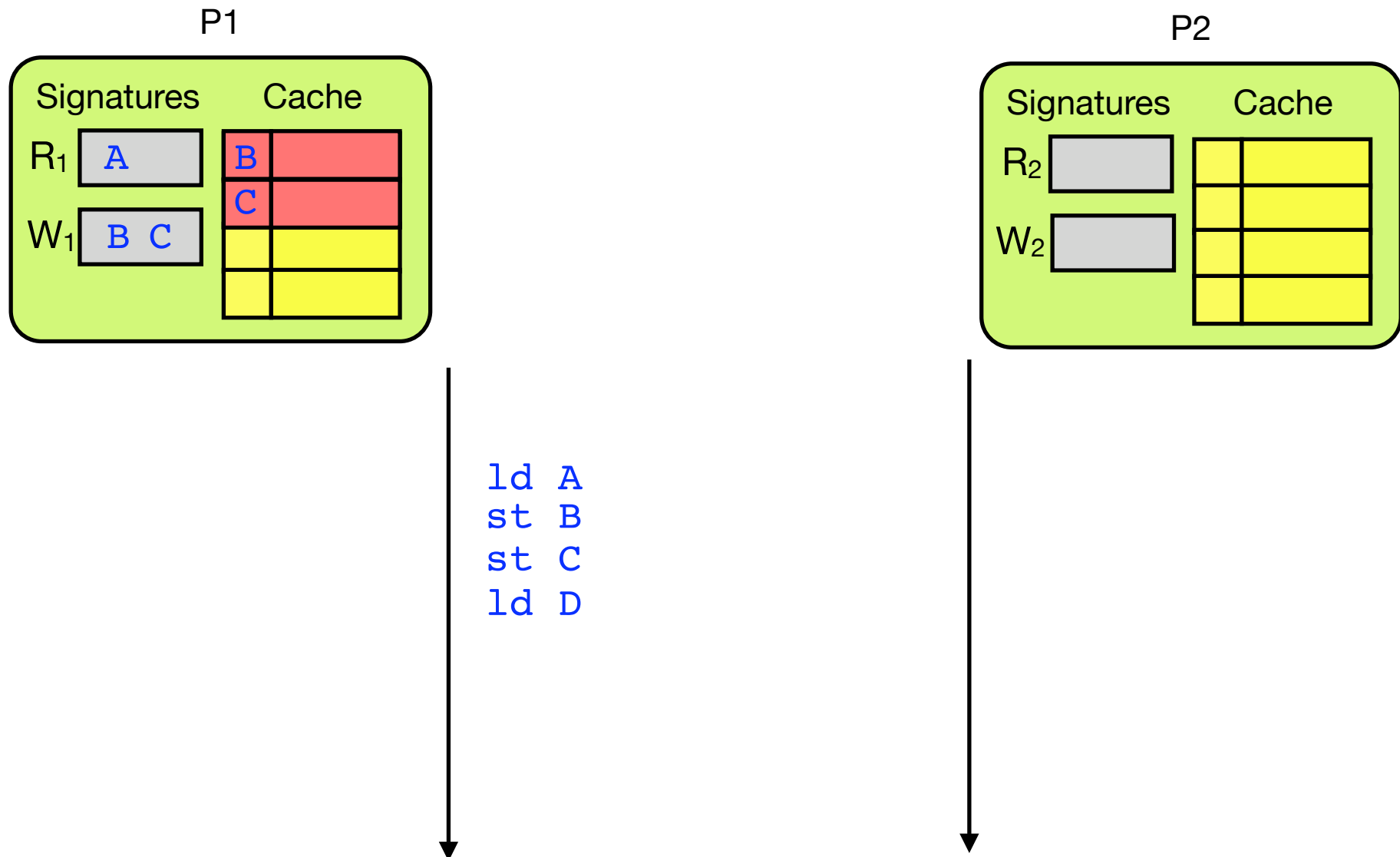
Chunk Atomicity and Isolation in Bulk



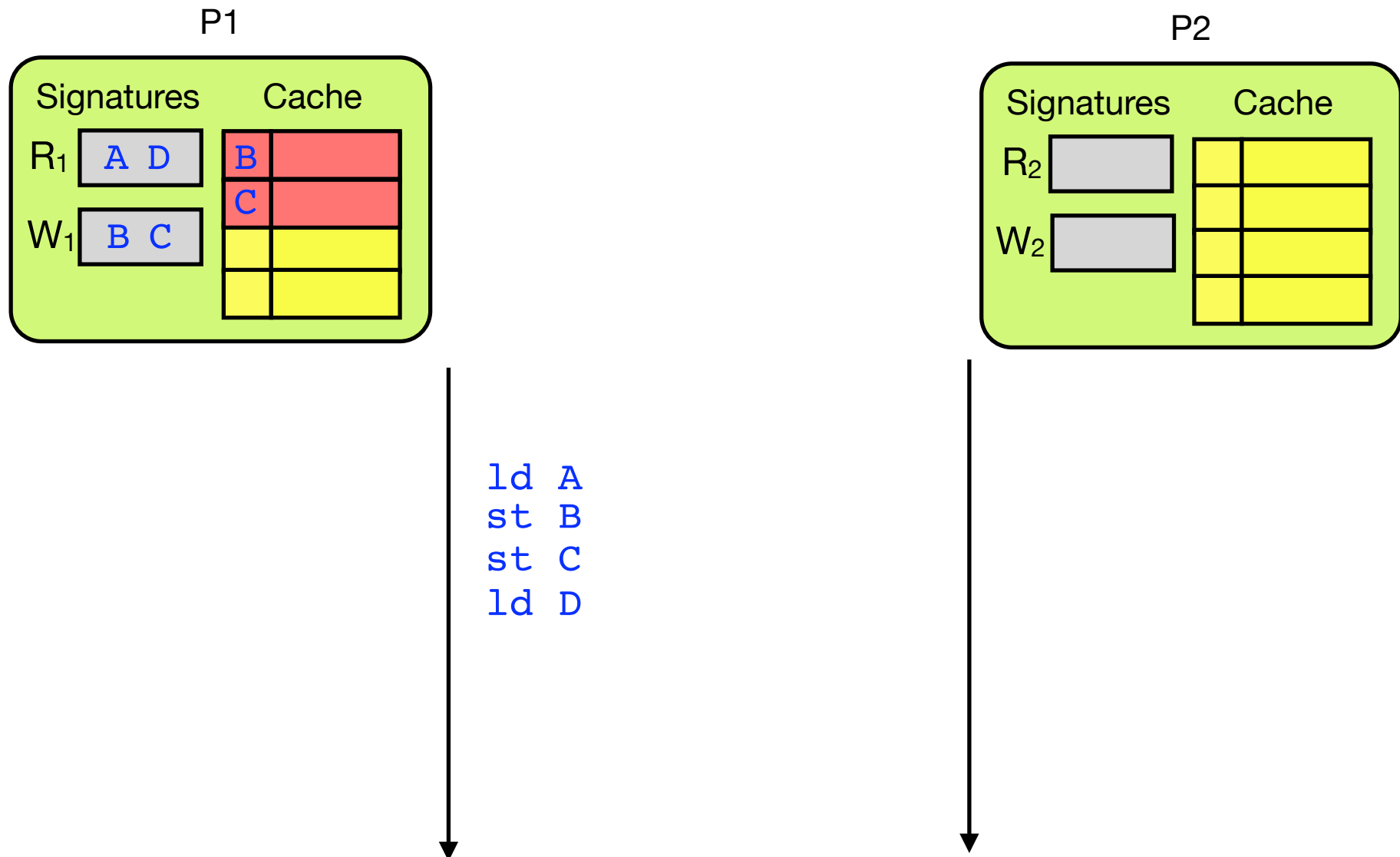
Chunk Atomicity and Isolation in Bulk



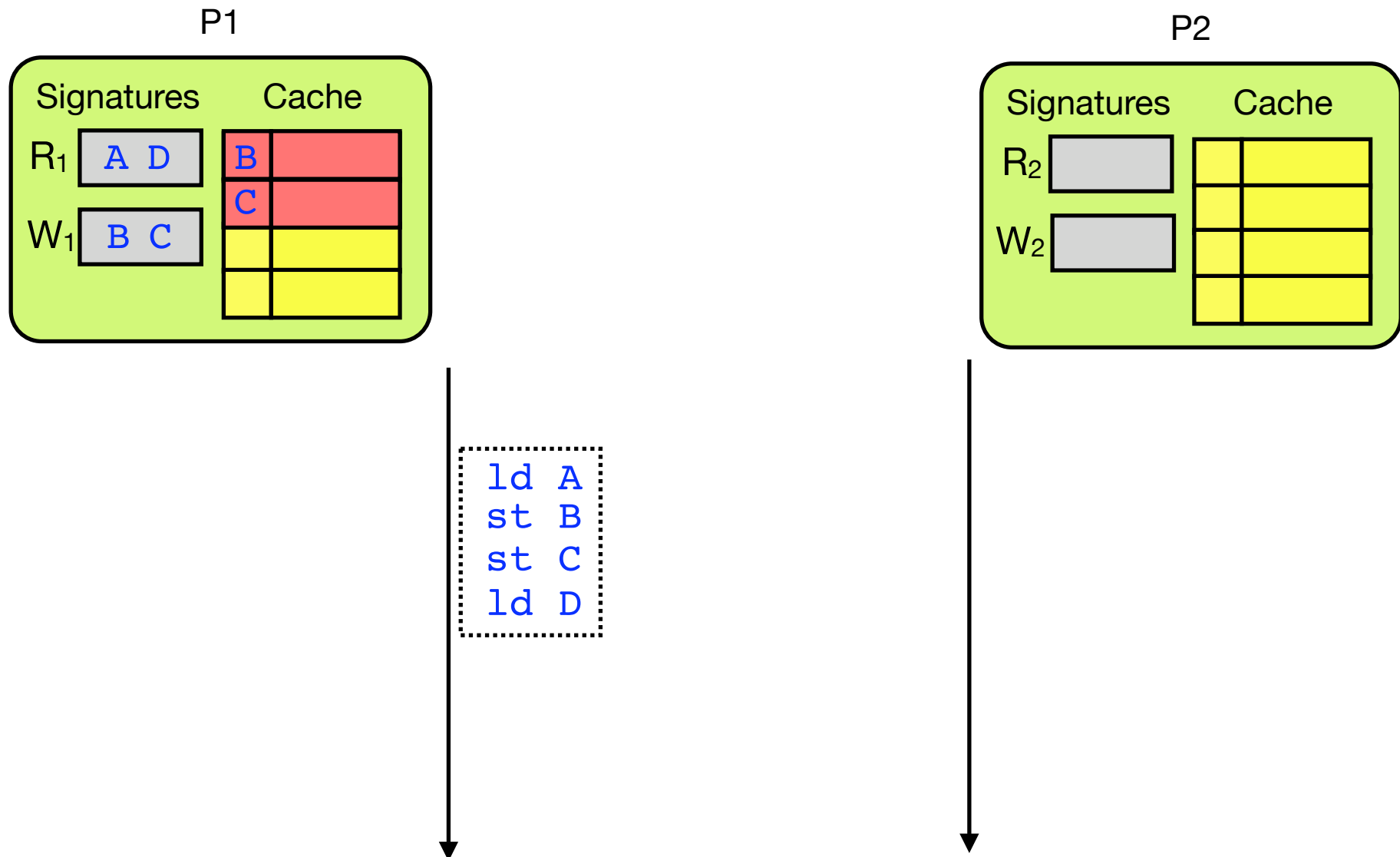
Chunk Atomicity and Isolation in Bulk



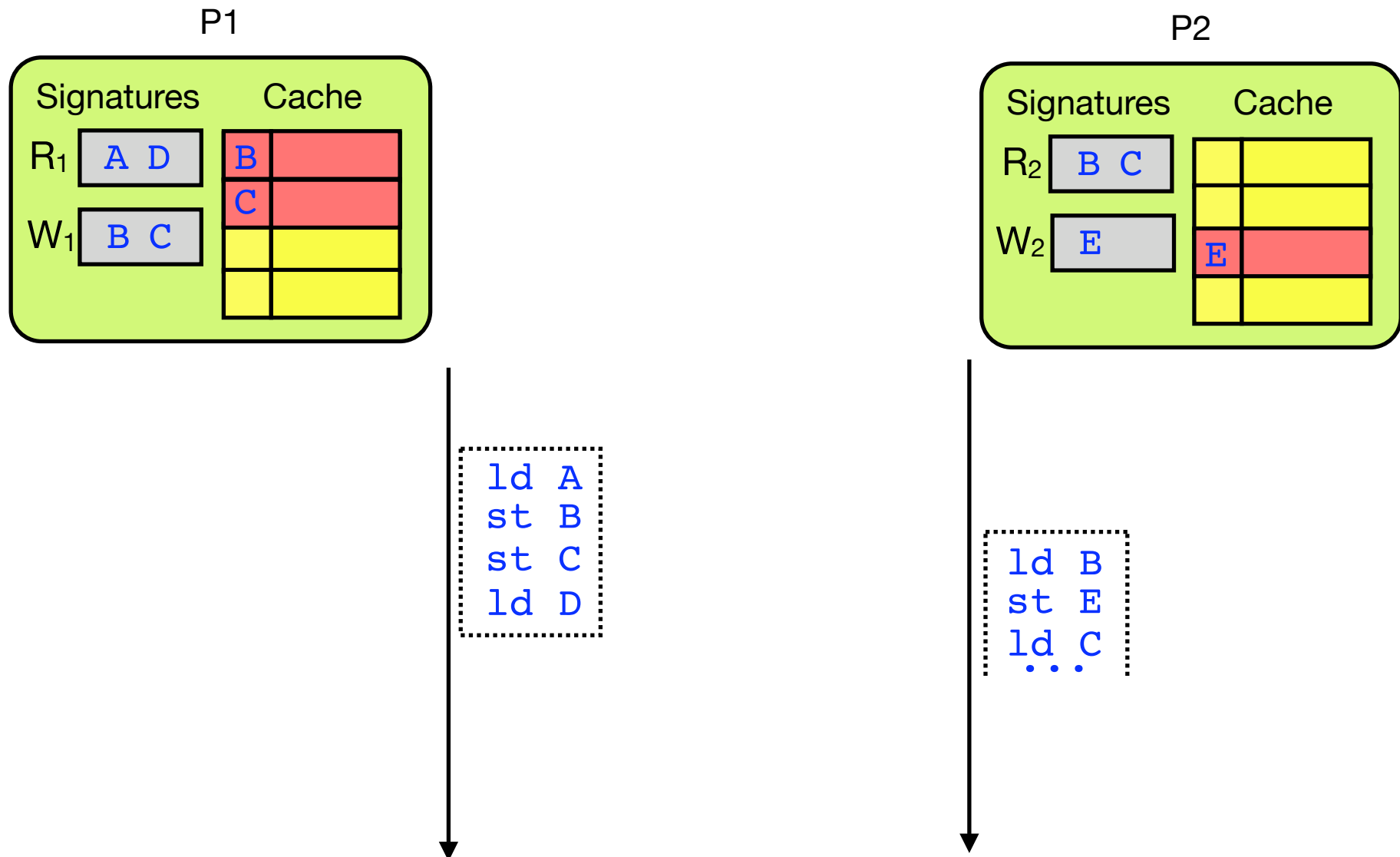
Chunk Atomicity and Isolation in Bulk



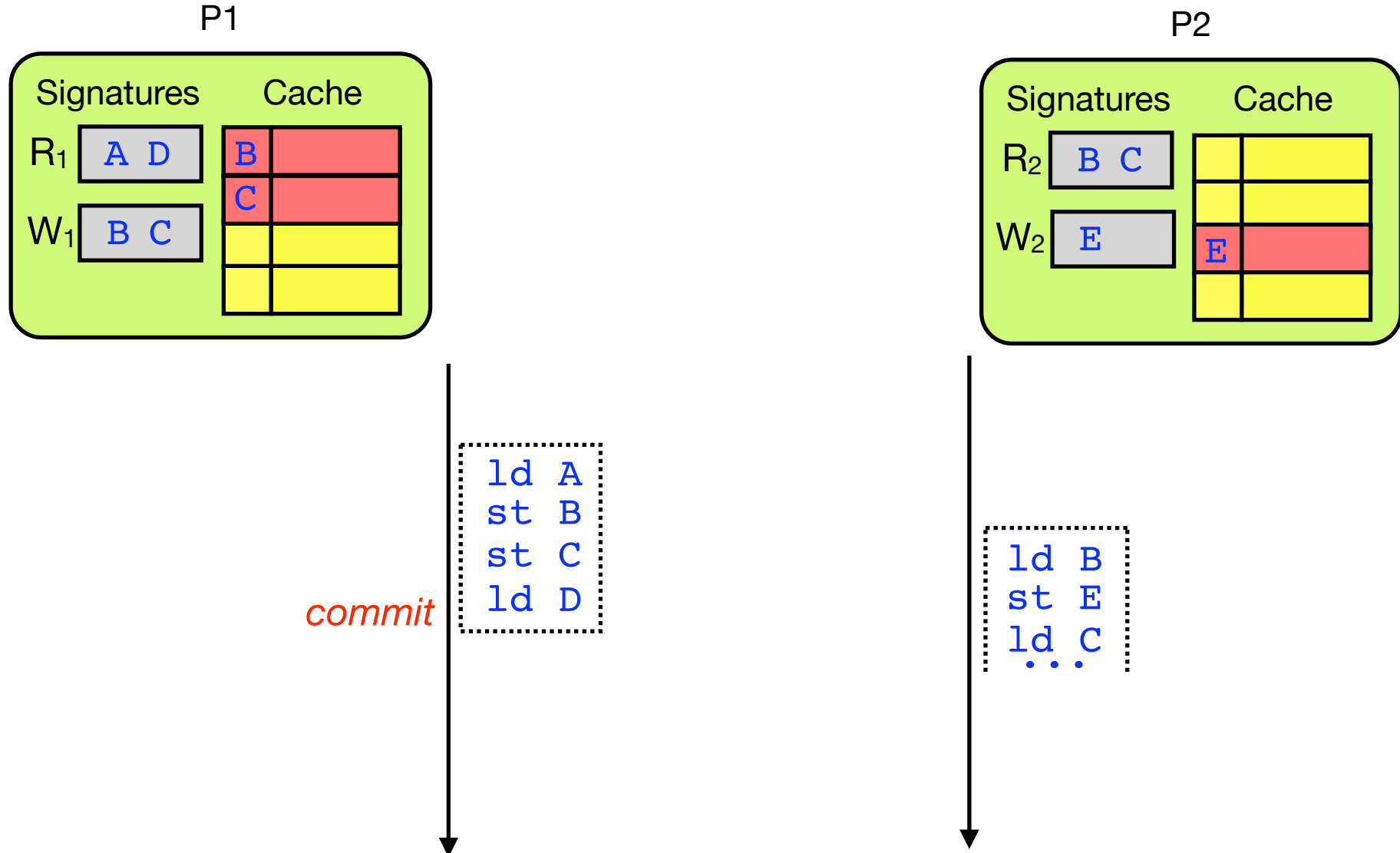
Chunk Atomicity and Isolation in Bulk



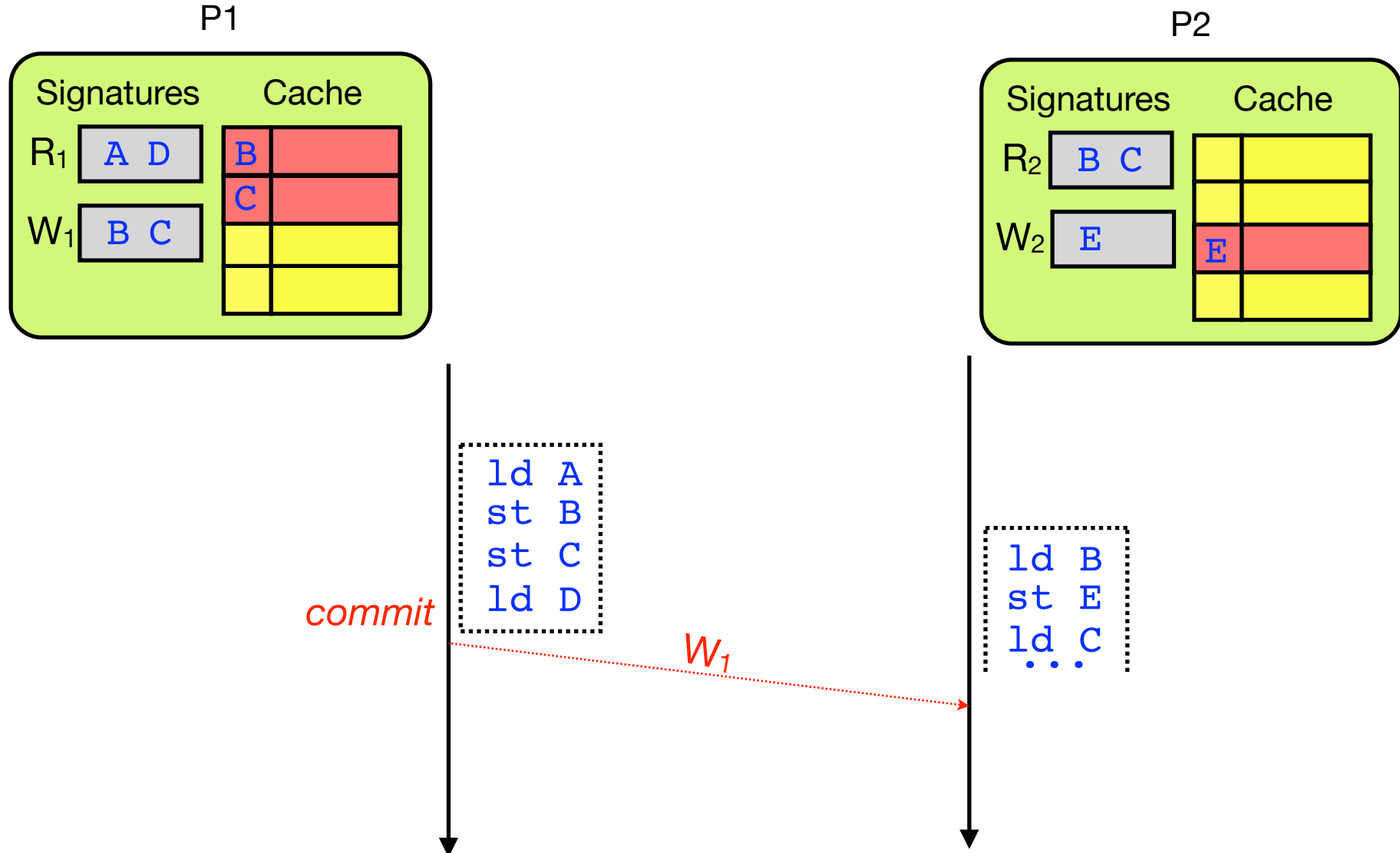
Chunk Atomicity and Isolation in Bulk



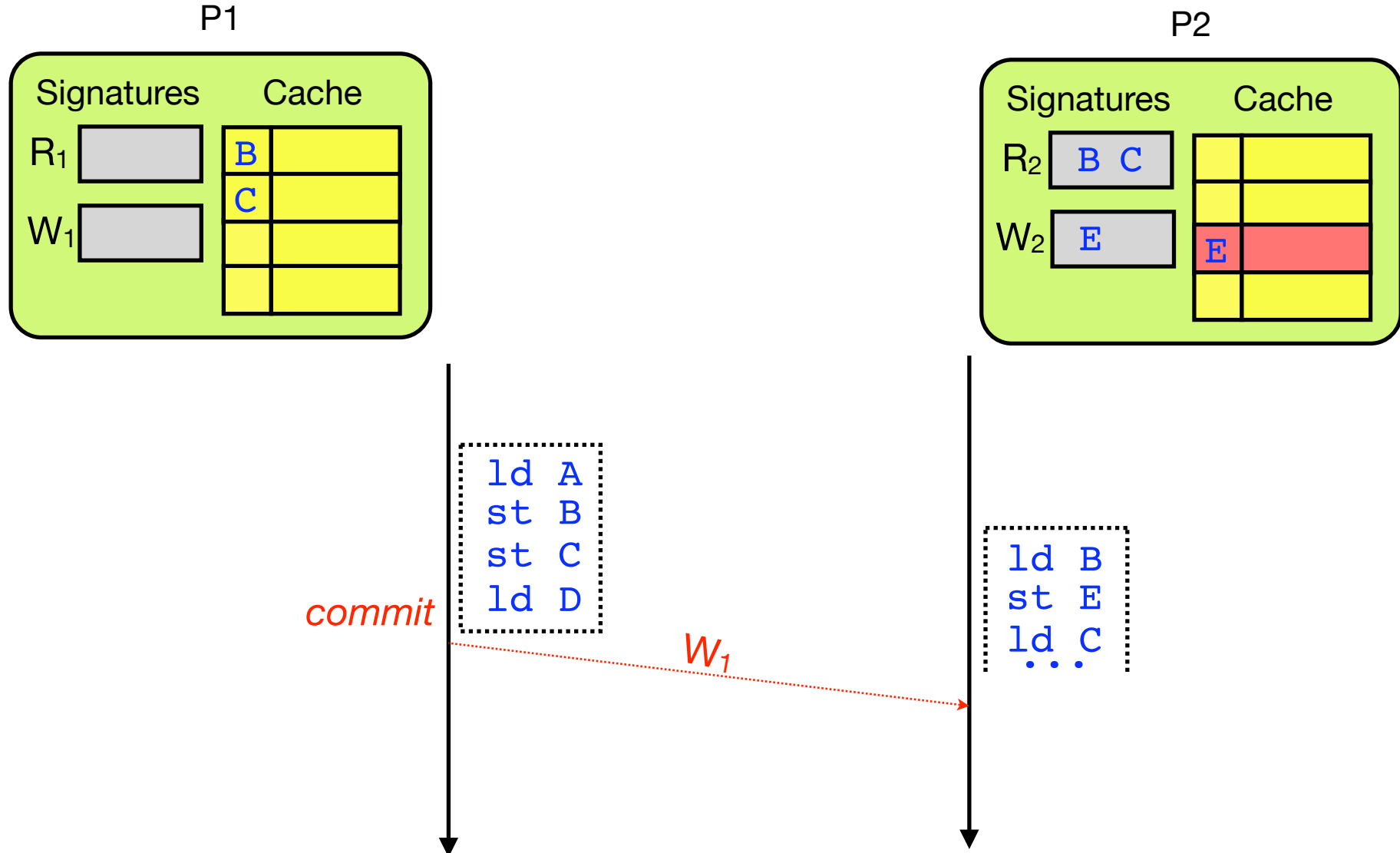
Chunk Atomicity and Isolation in Bulk



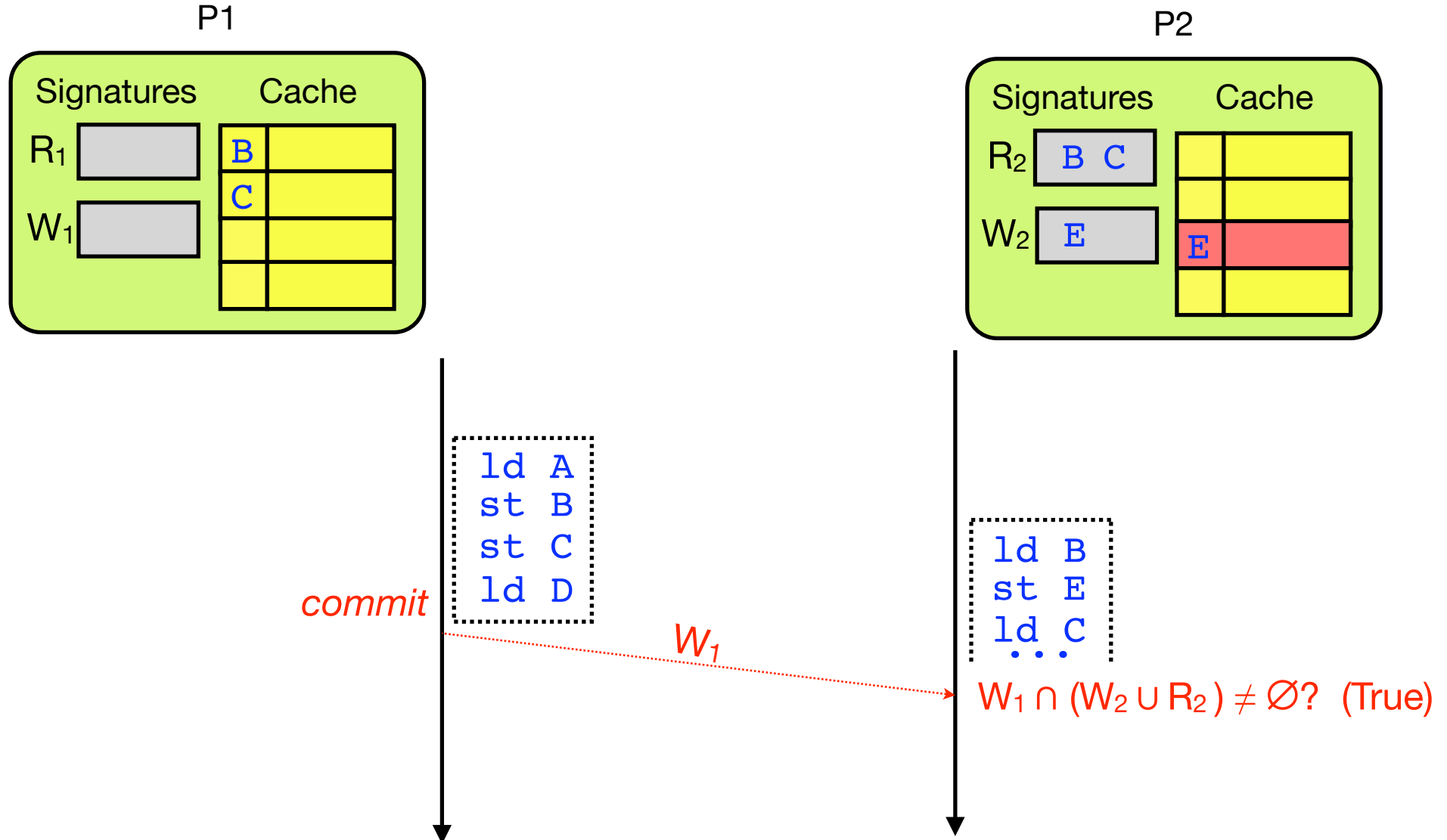
Chunk Atomicity and Isolation in Bulk



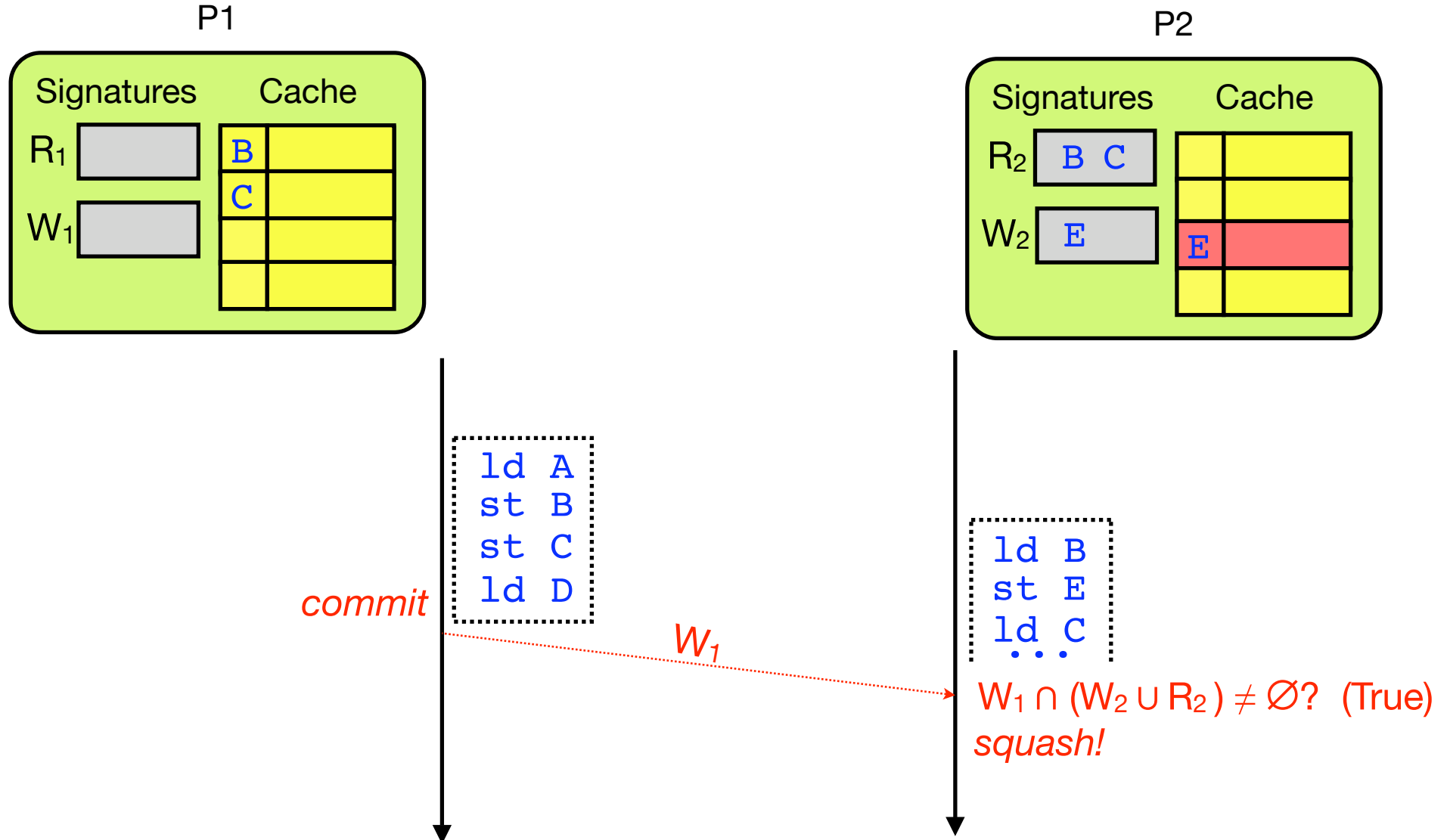
Chunk Atomicity and Isolation in Bulk



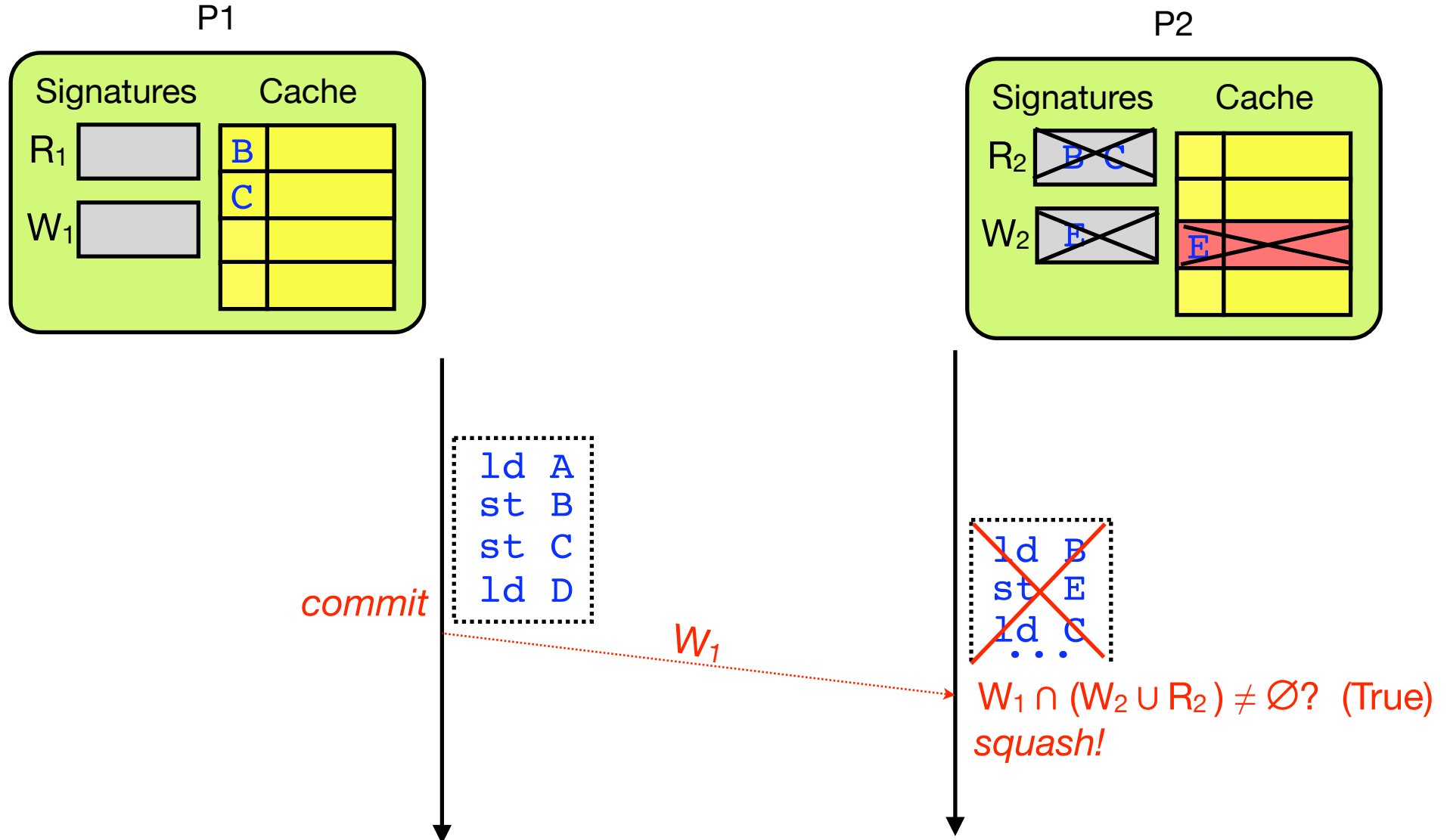
Chunk Atomicity and Isolation in Bulk



Chunk Atomicity and Isolation in Bulk



Chunk Atomicity and Isolation in Bulk



Program Chunk Order



Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware

Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions

Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*

Program Chunk Order

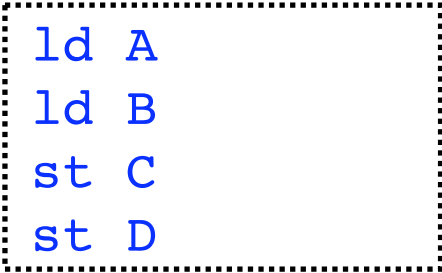
- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:

Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
 - Instructions inside a chunk arbitrarily reordered

Program Chunk Order

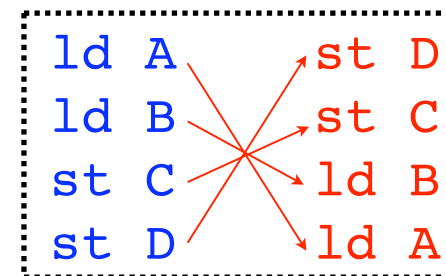
- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
 - Instructions inside a chunk arbitrarily reordered



```
ld A  
ld B  
st C  
st D
```

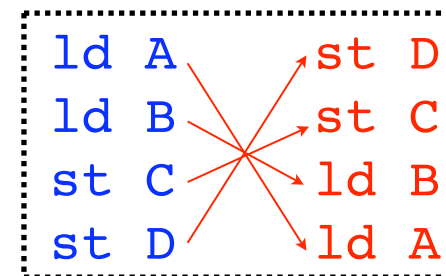
Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
 - Instructions inside a chunk arbitrarily reordered



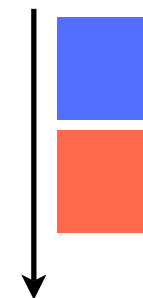
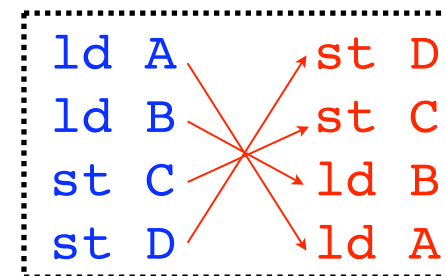
Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
 - Instructions inside a chunk arbitrarily reordered
 - A processor can interleave the execution of instructions from 2 in-flight chunks



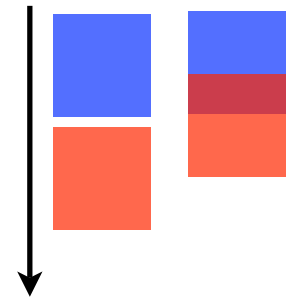
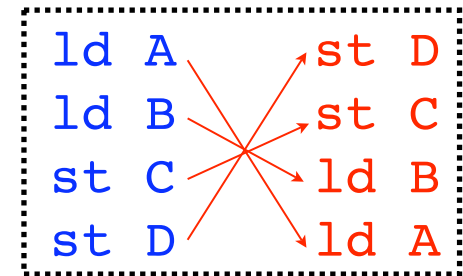
Program Chunk Order

- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
 - Instructions inside a chunk arbitrarily reordered
 - A processor can interleave the execution of instructions from 2 in-flight chunks



Program Chunk Order

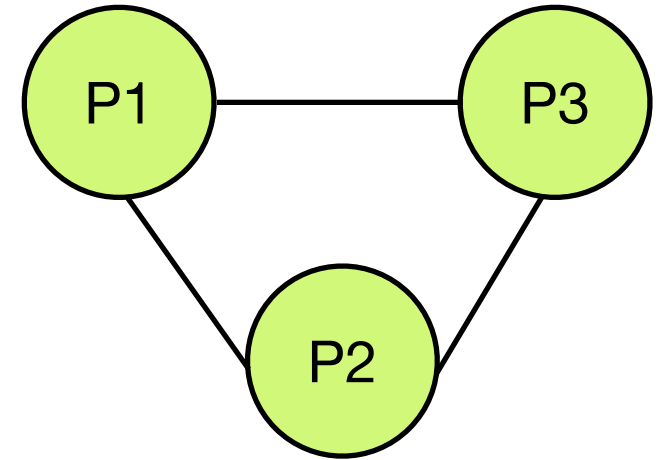
- Chunks are **arbitrarily** defined by the hardware
 - e.g. every 2000 dynamic instructions
- A processor commits chunks in *program order*
- High-performance:
 - Instructions inside a chunk arbitrarily reordered
 - A processor can interleave the execution of instructions from 2 in-flight chunks



Single Sequential Order

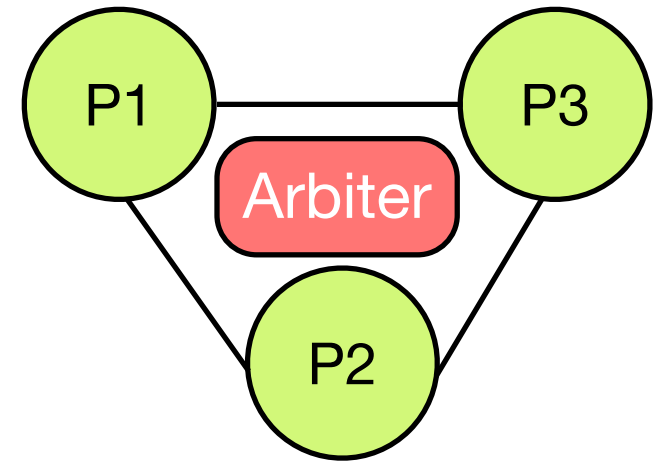


Single Sequential Order



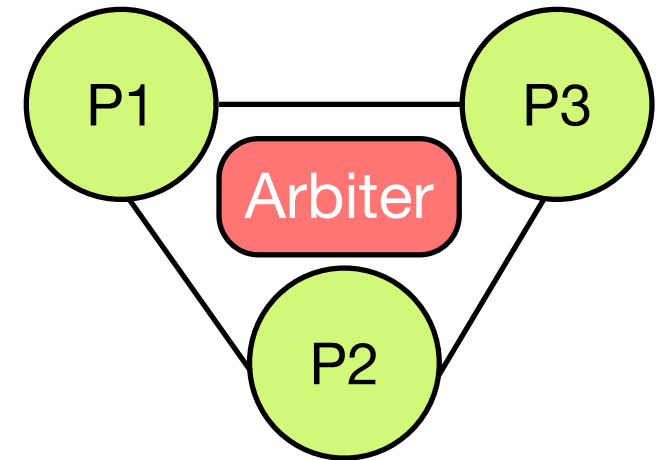
Single Sequential Order

- Need an Arbiter for chunk commits



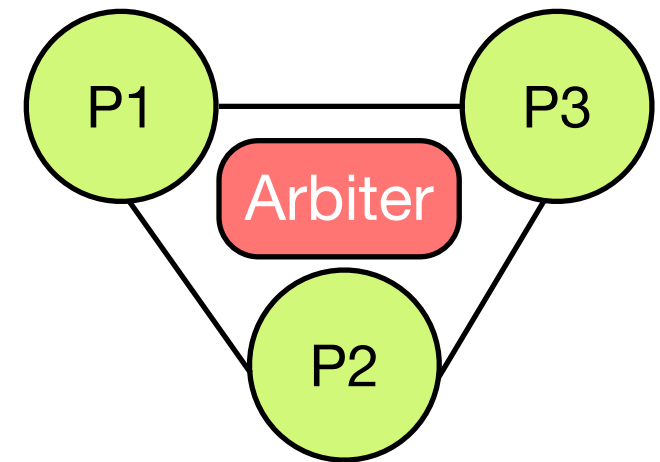
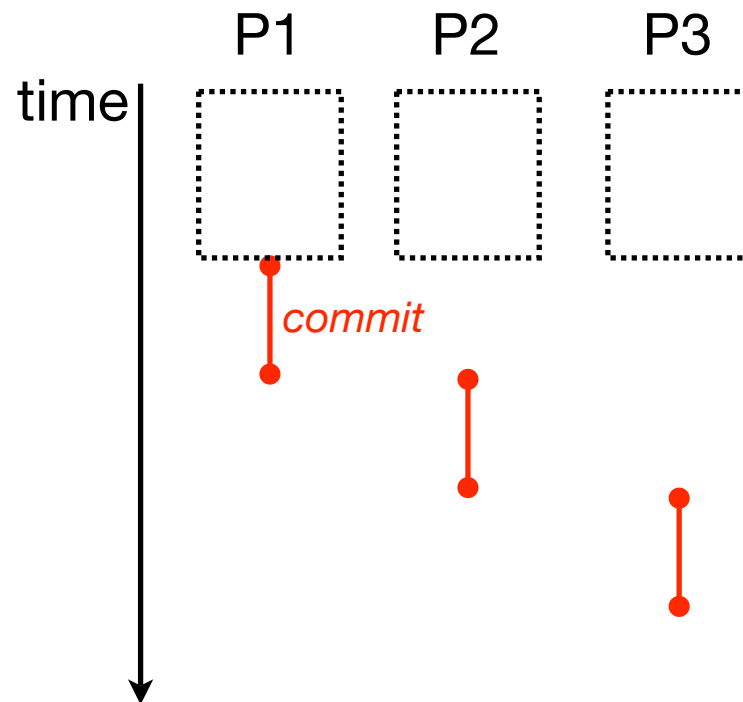
Single Sequential Order

- Need an Arbiter for chunk commits
- Naive:
 - allow a single commit at a time



Single Sequential Order

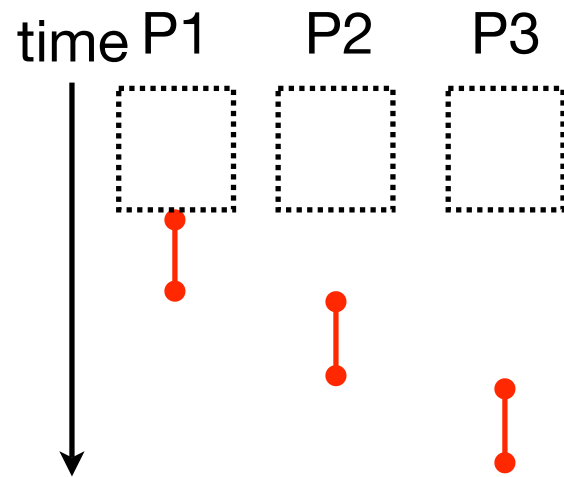
- Need an Arbiter for chunk commits
- Naive:
 - allow a single commit at a time



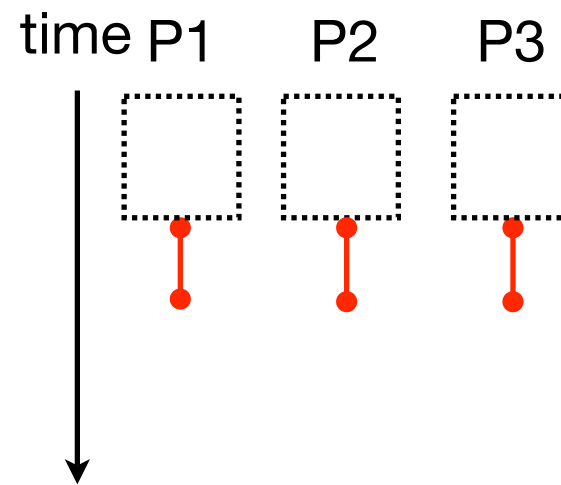
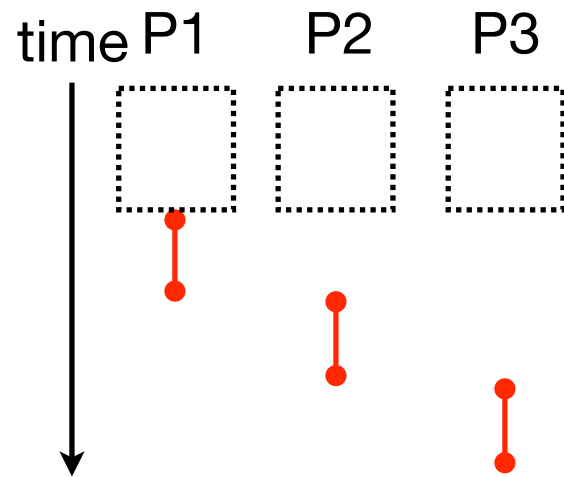
Allowing Simultaneous Chunk Commits



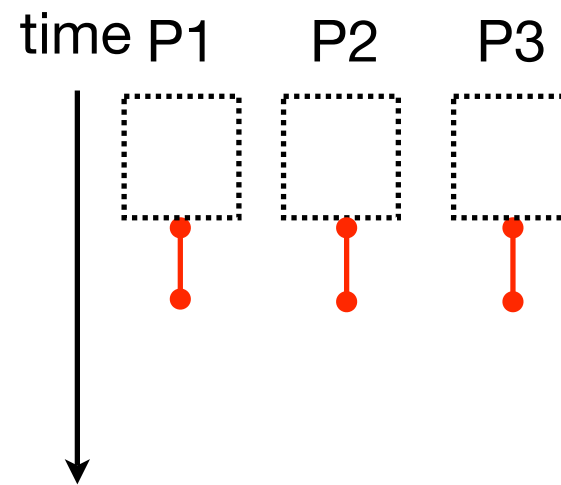
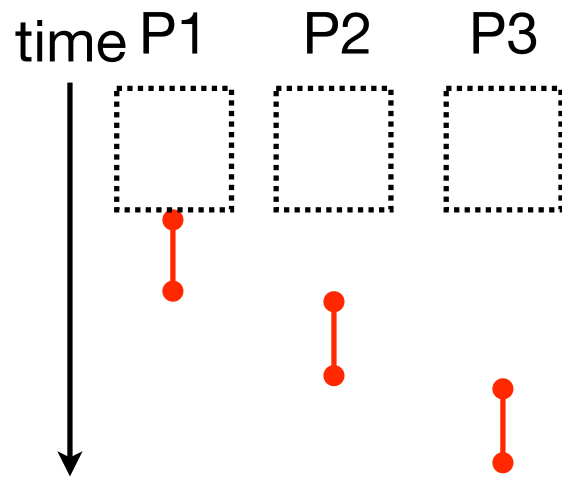
Allowing Simultaneous Chunk Commits



Allowing Simultaneous Chunk Commits

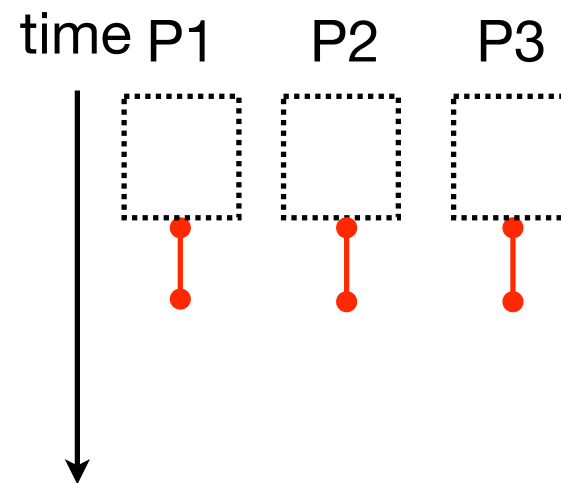
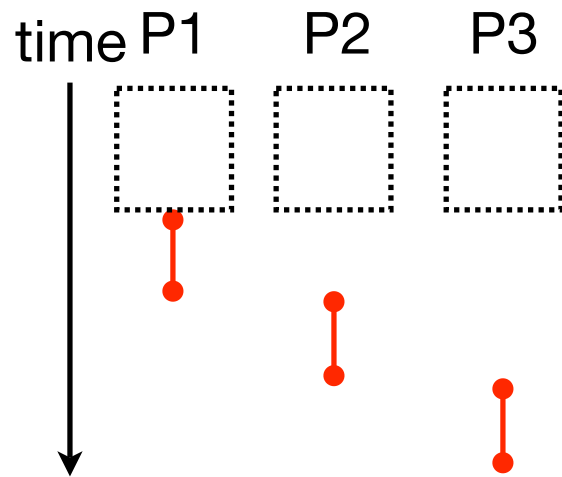


Allowing Simultaneous Chunk Commits



- Conditions:

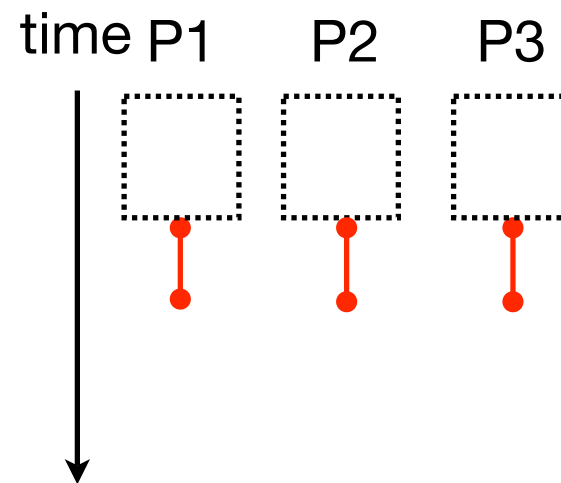
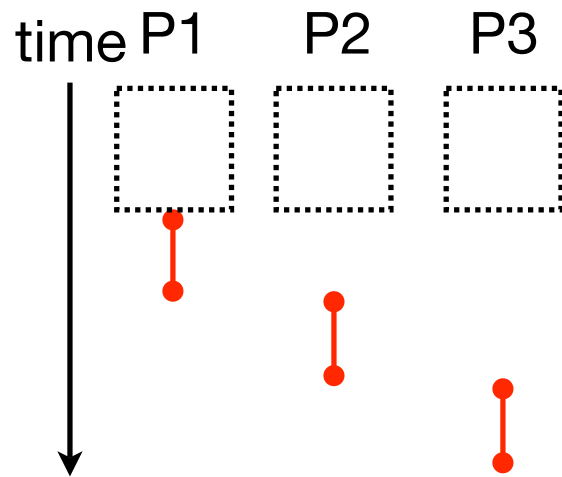
Allowing Simultaneous Chunk Commits



- Conditions:

- committing chunks' memory operations should not intersect

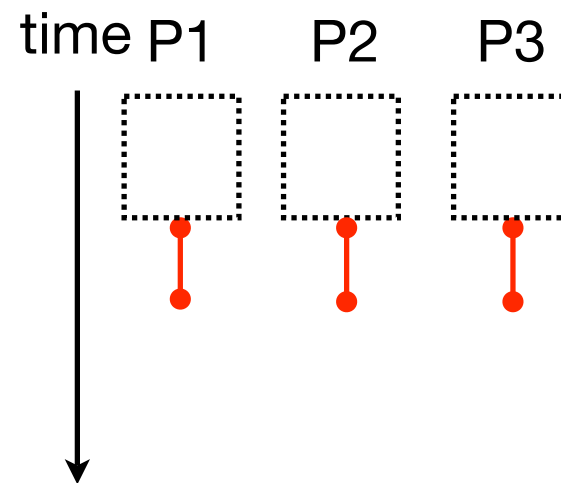
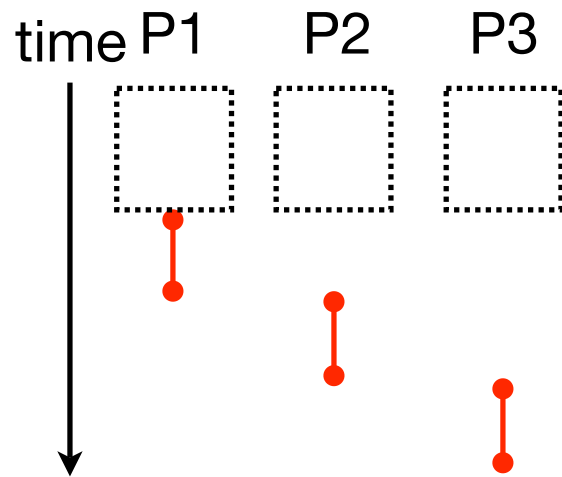
Allowing Simultaneous Chunk Commits



- Conditions:

- committing chunks' memory operations should not intersect
 - both reads and writes

Allowing Simultaneous Chunk Commits



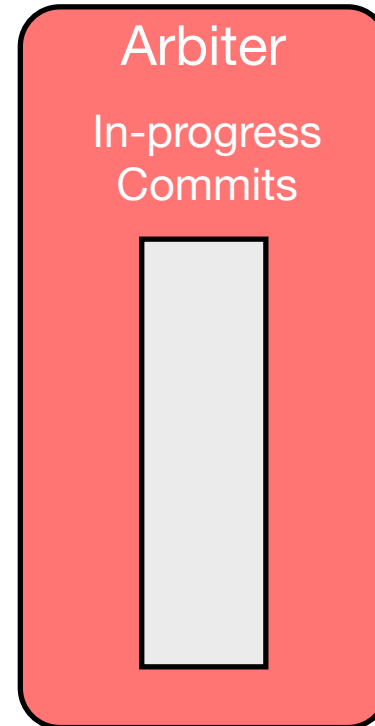
- Conditions:

- committing chunks' memory operations should not intersect
 - both reads and writes
- this can be efficiently enforced with chunk signatures

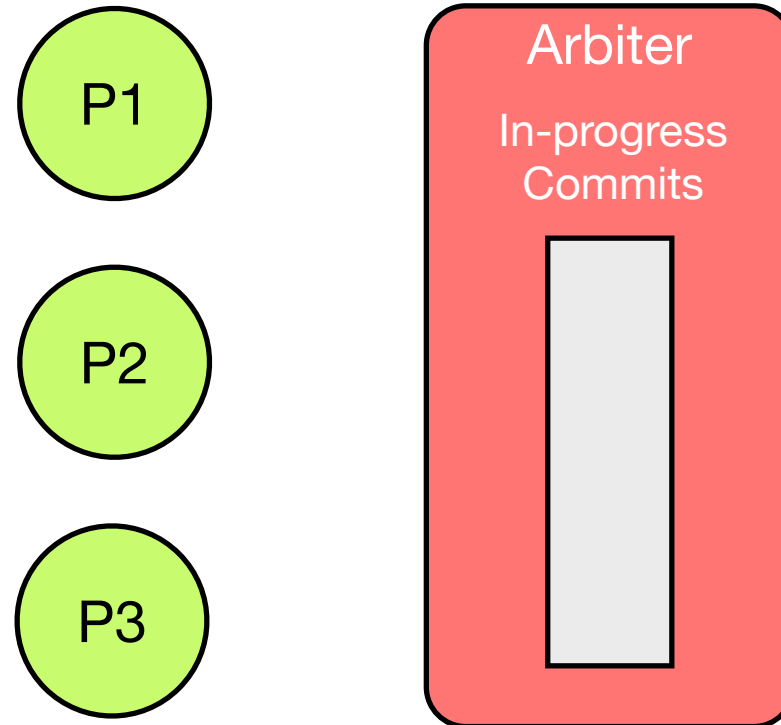
Arbitrating Simultaneous Chunk Commits



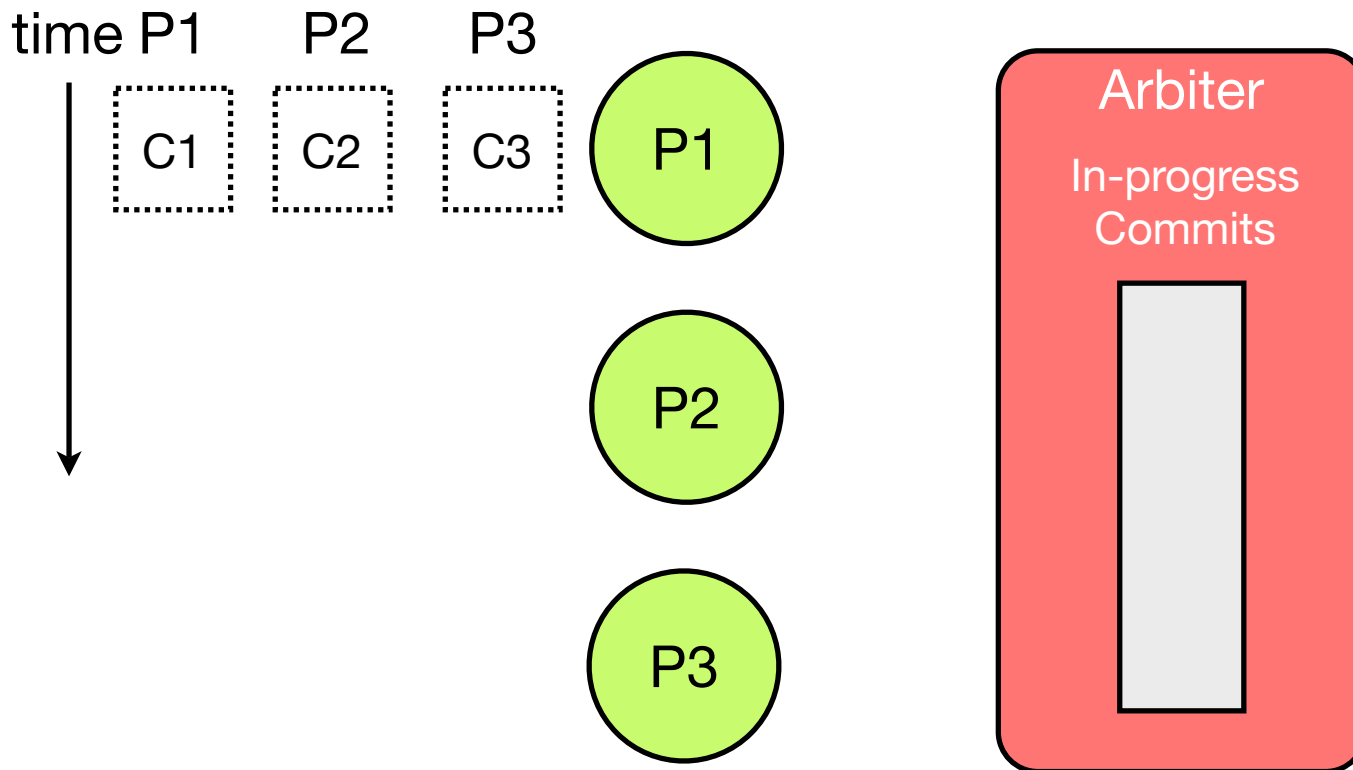
Arbitrating Simultaneous Chunk Commits



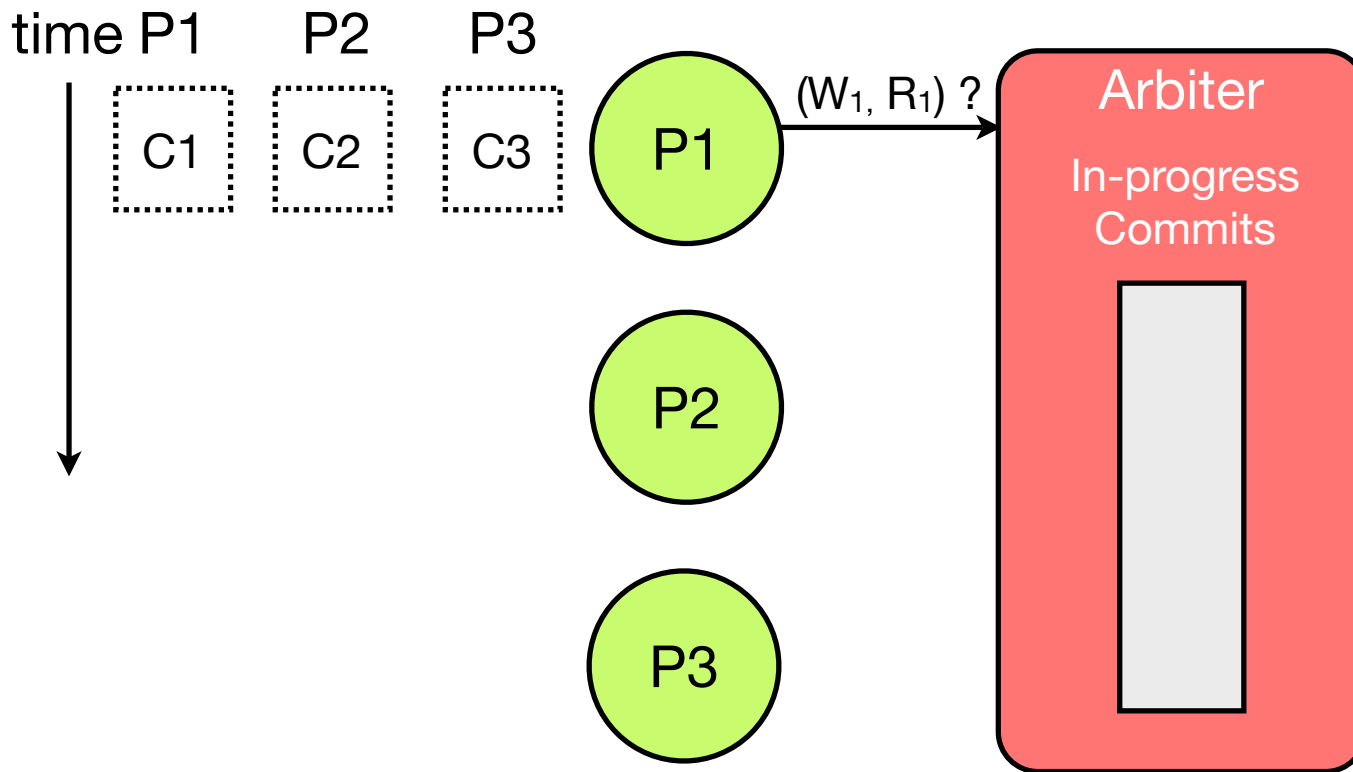
Arbitrating Simultaneous Chunk Commits



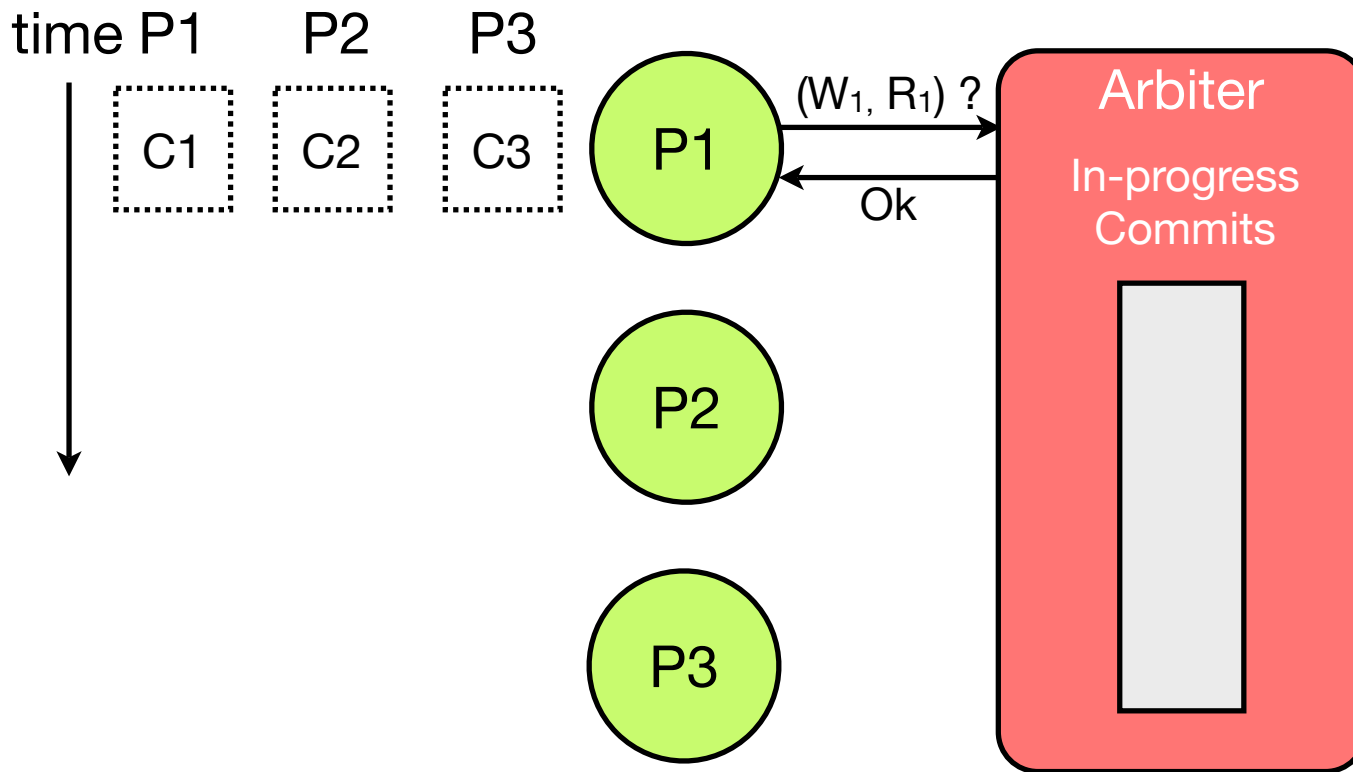
Arbitrating Simultaneous Chunk Commits



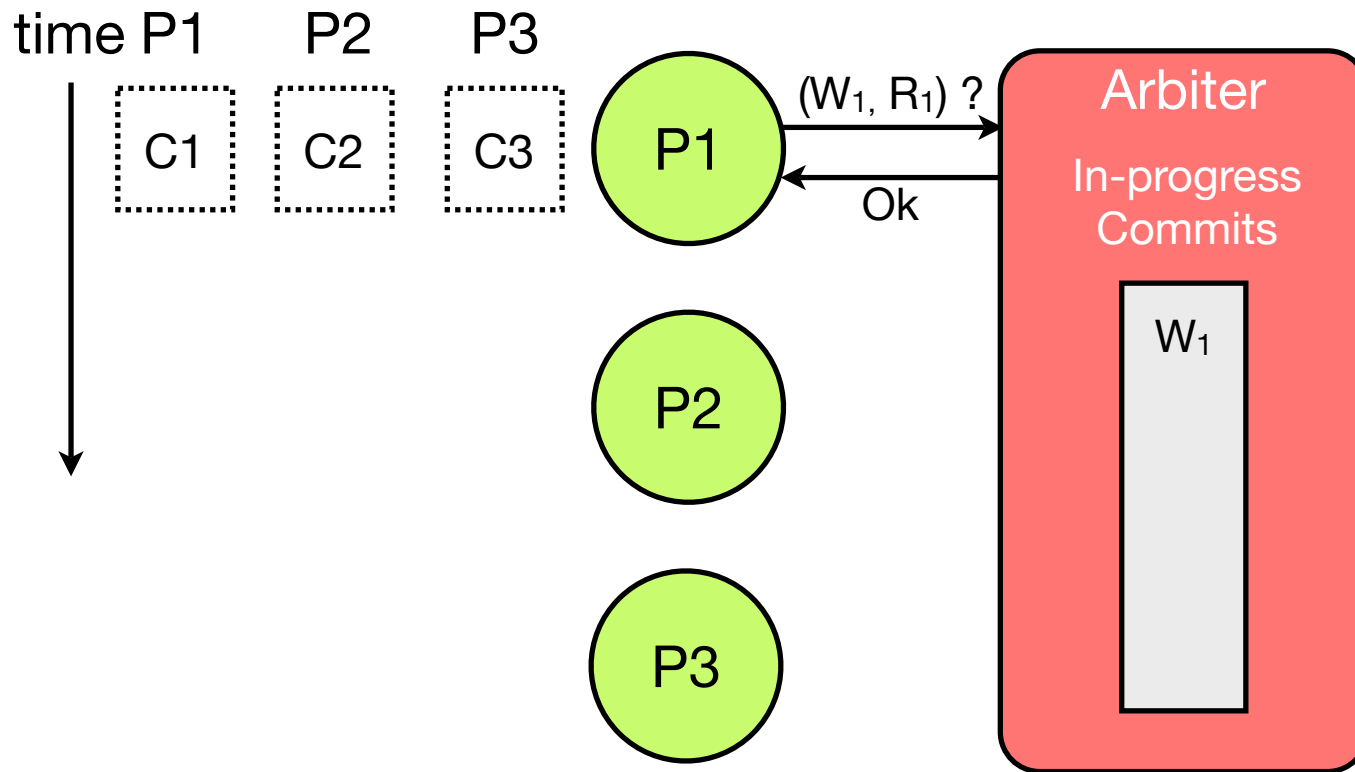
Arbitrating Simultaneous Chunk Commits



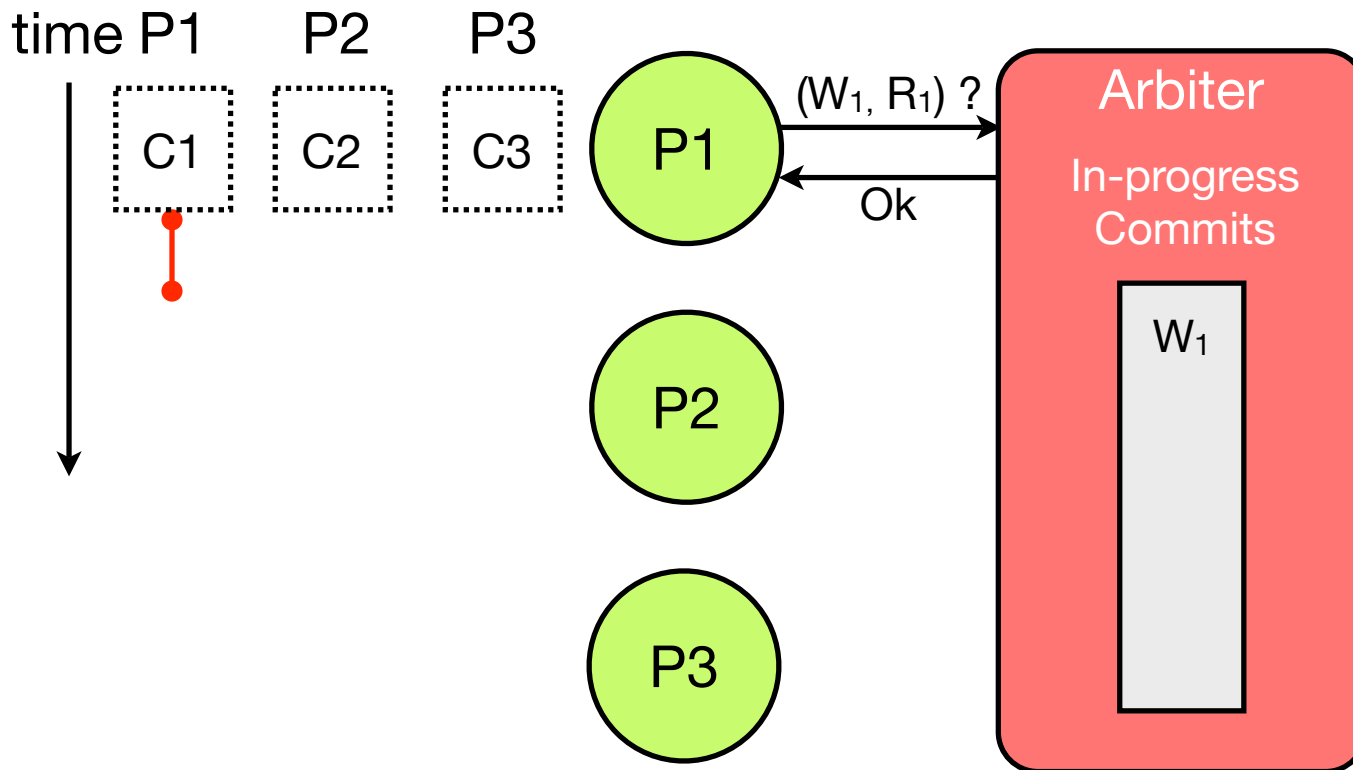
Arbitrating Simultaneous Chunk Commits



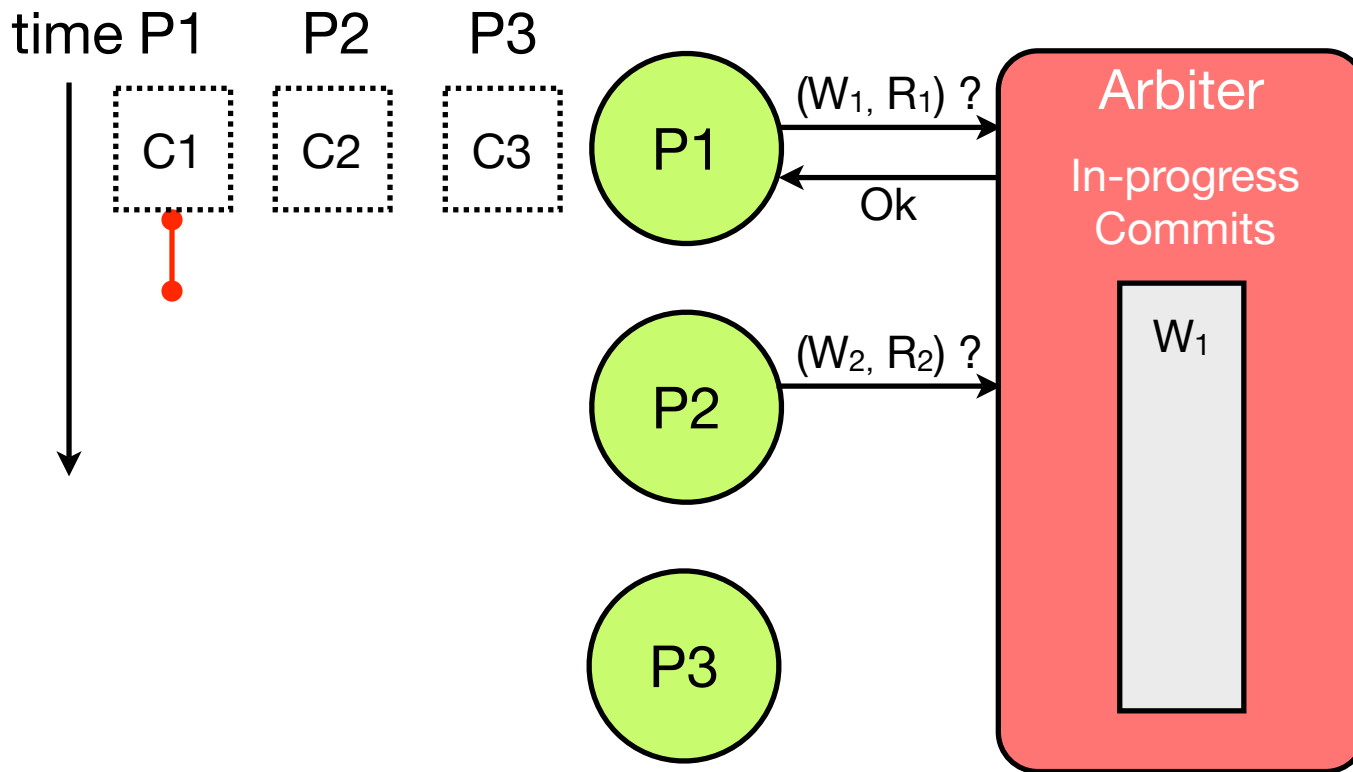
Arbitrating Simultaneous Chunk Commits



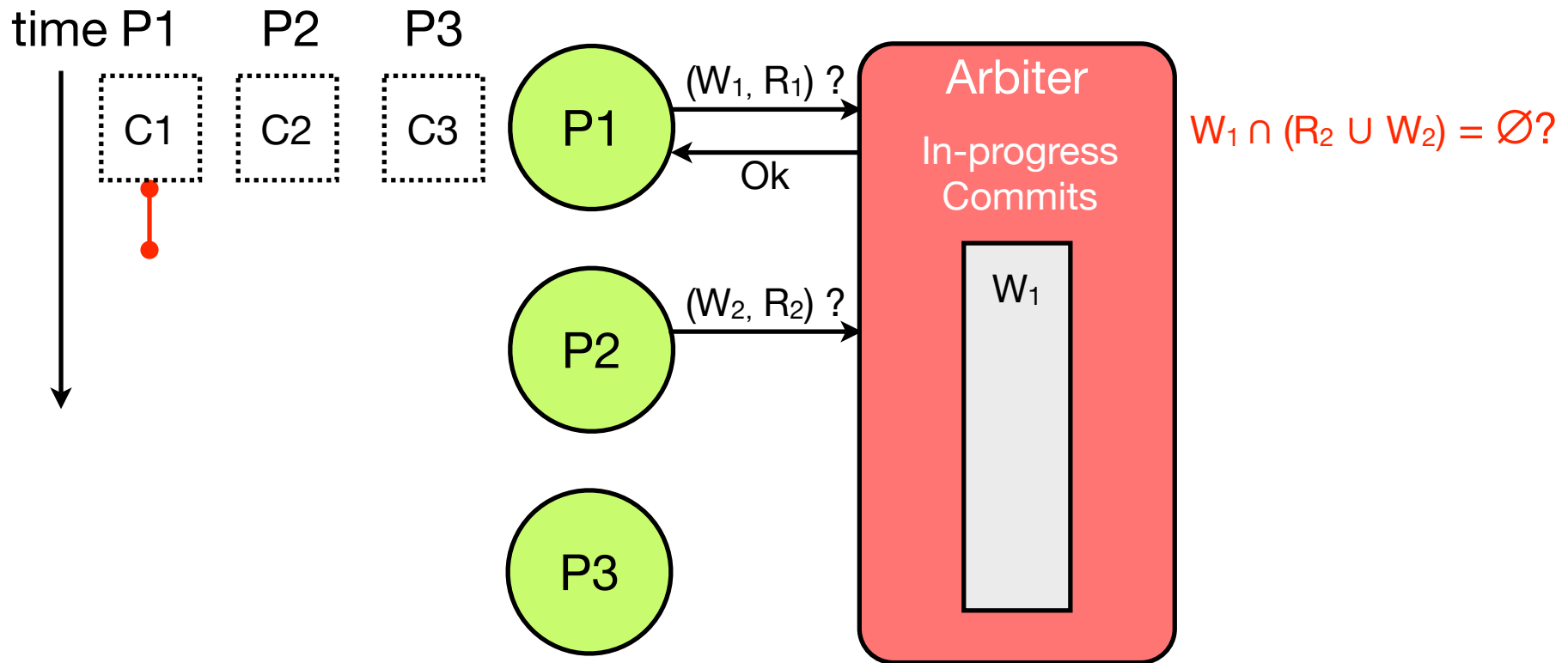
Arbitrating Simultaneous Chunk Commits



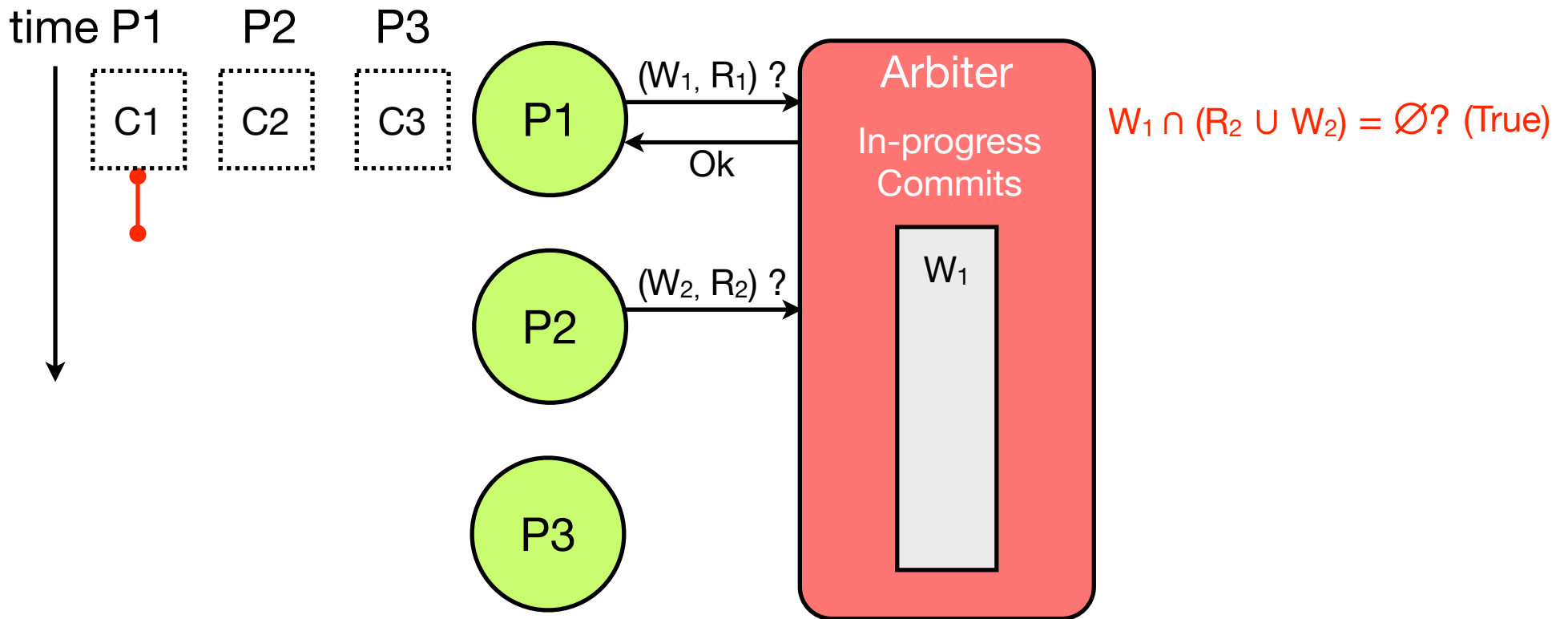
Arbitrating Simultaneous Chunk Commits



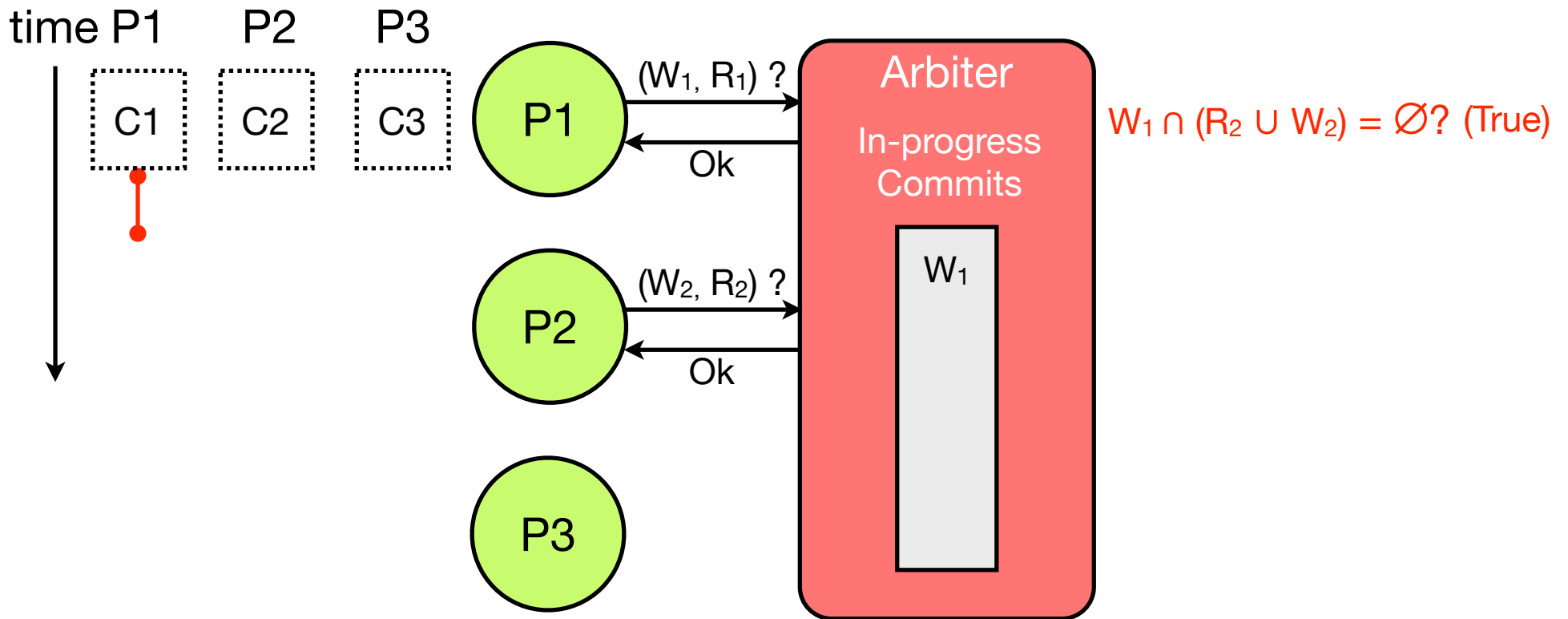
Arbitrating Simultaneous Chunk Commits



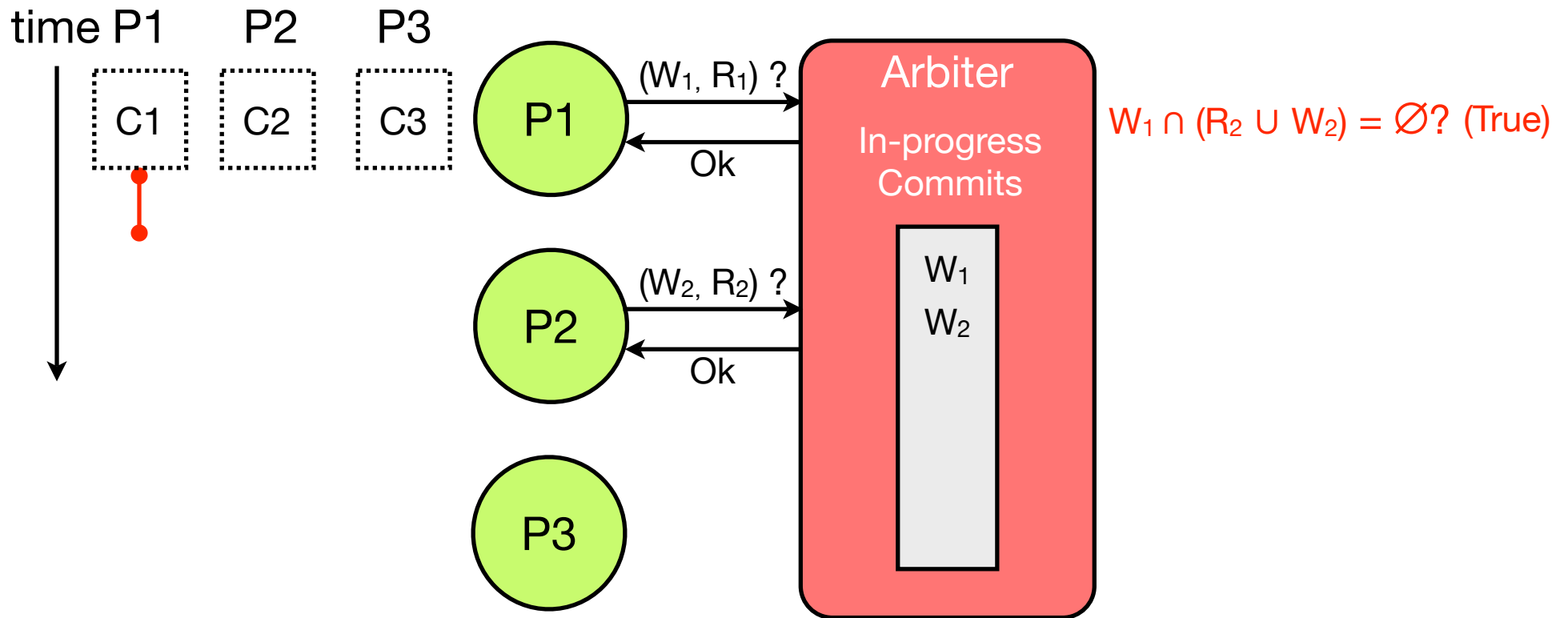
Arbitrating Simultaneous Chunk Commits



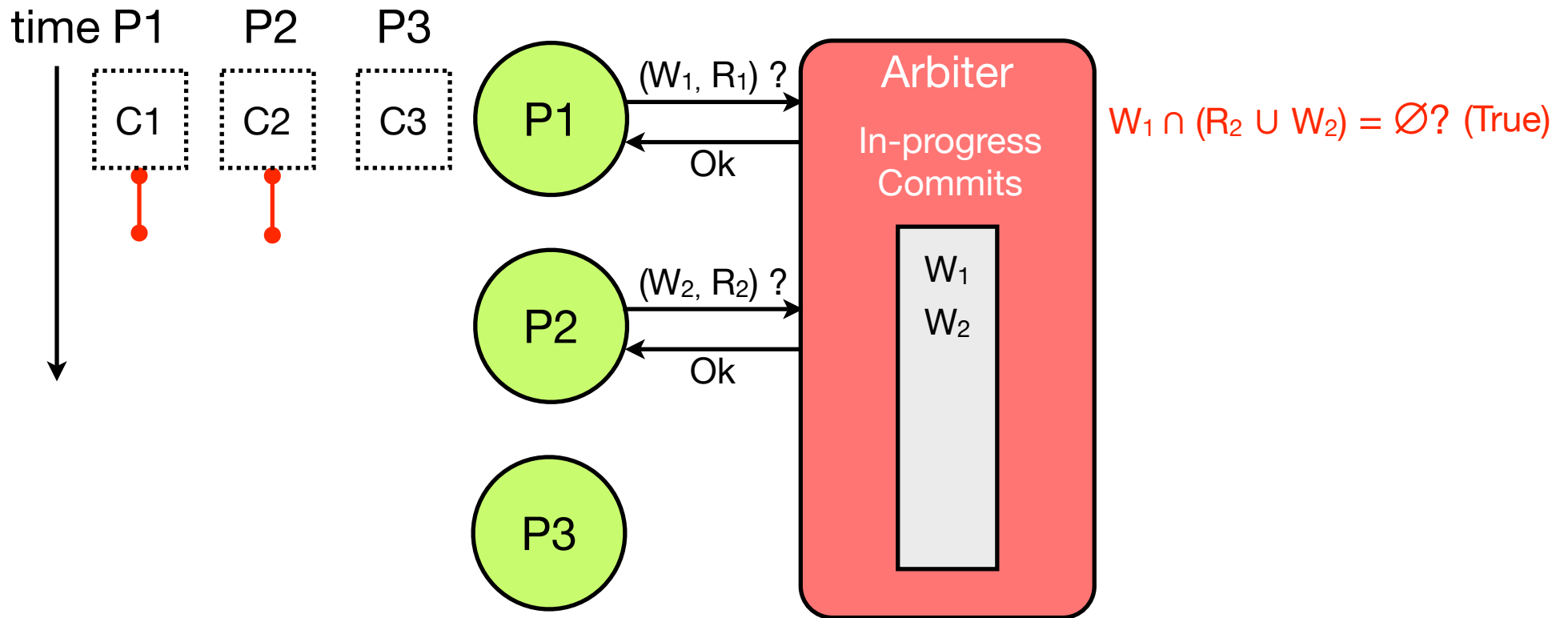
Arbitrating Simultaneous Chunk Commits



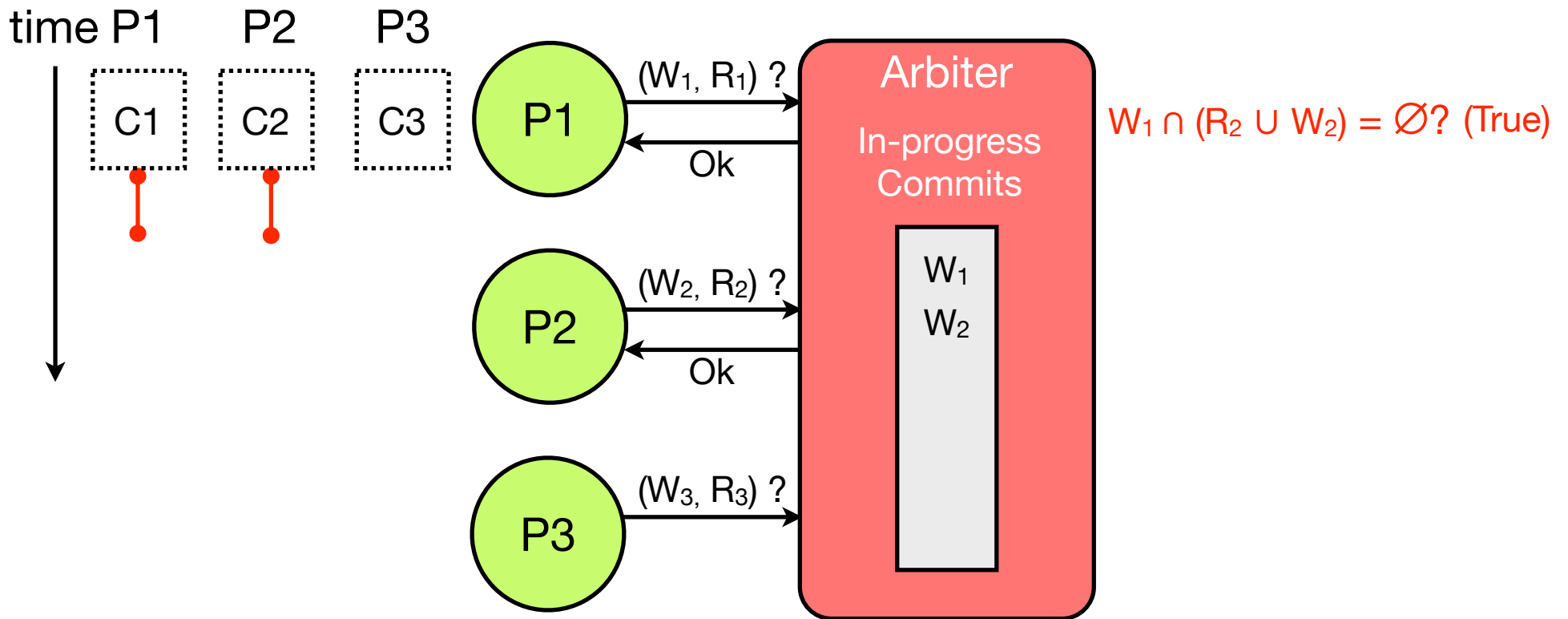
Arbitrating Simultaneous Chunk Commits



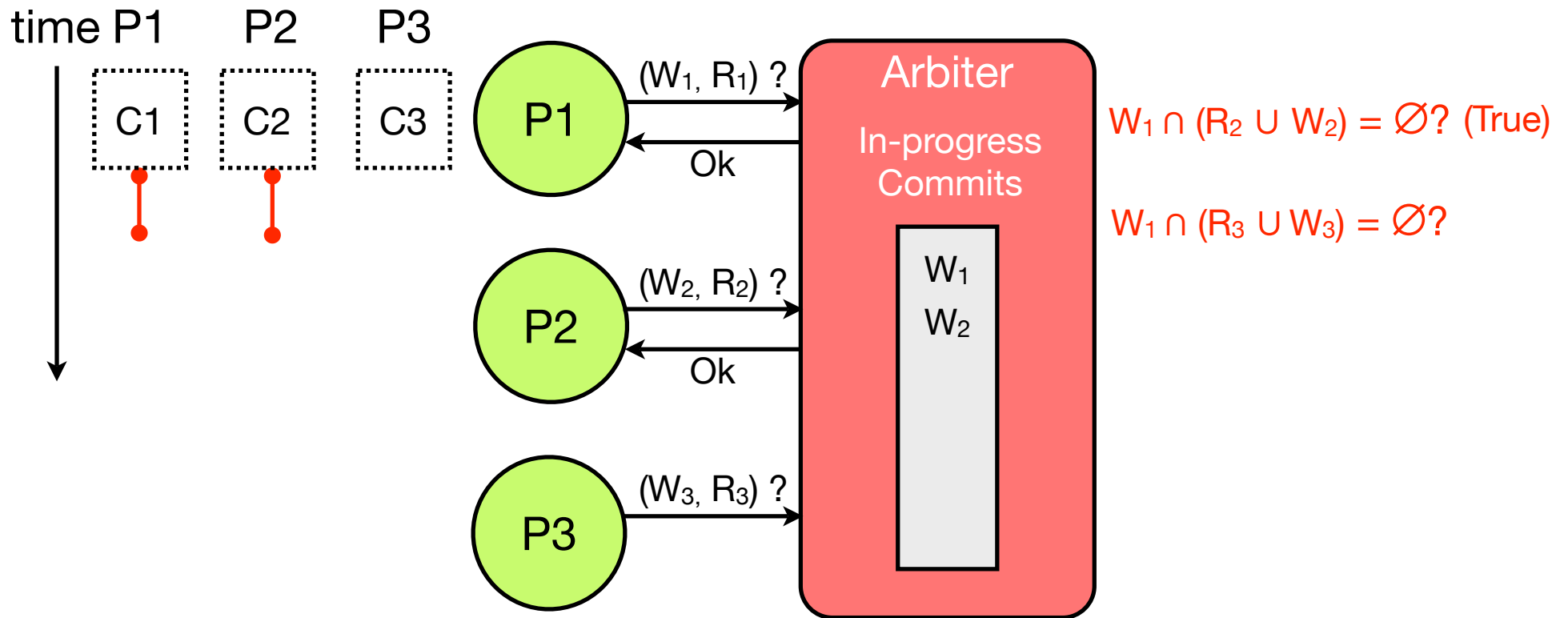
Arbitrating Simultaneous Chunk Commits



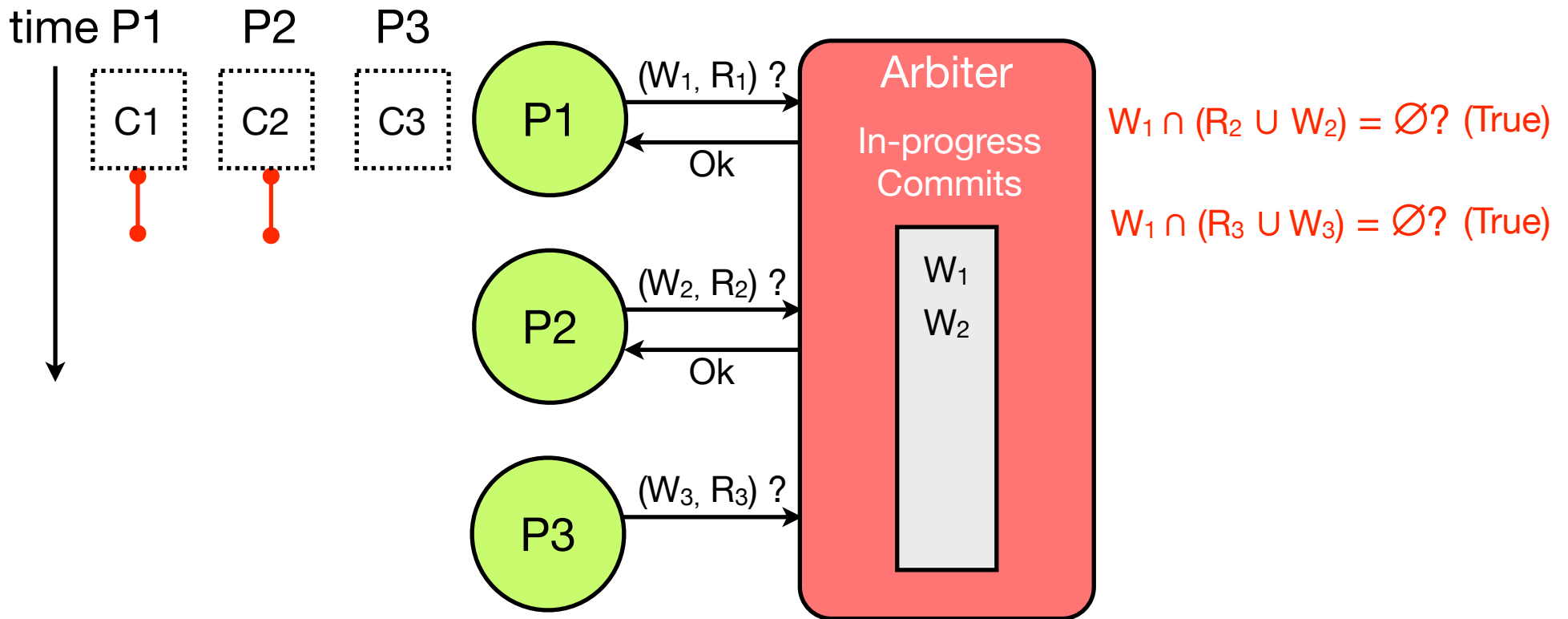
Arbitrating Simultaneous Chunk Commits



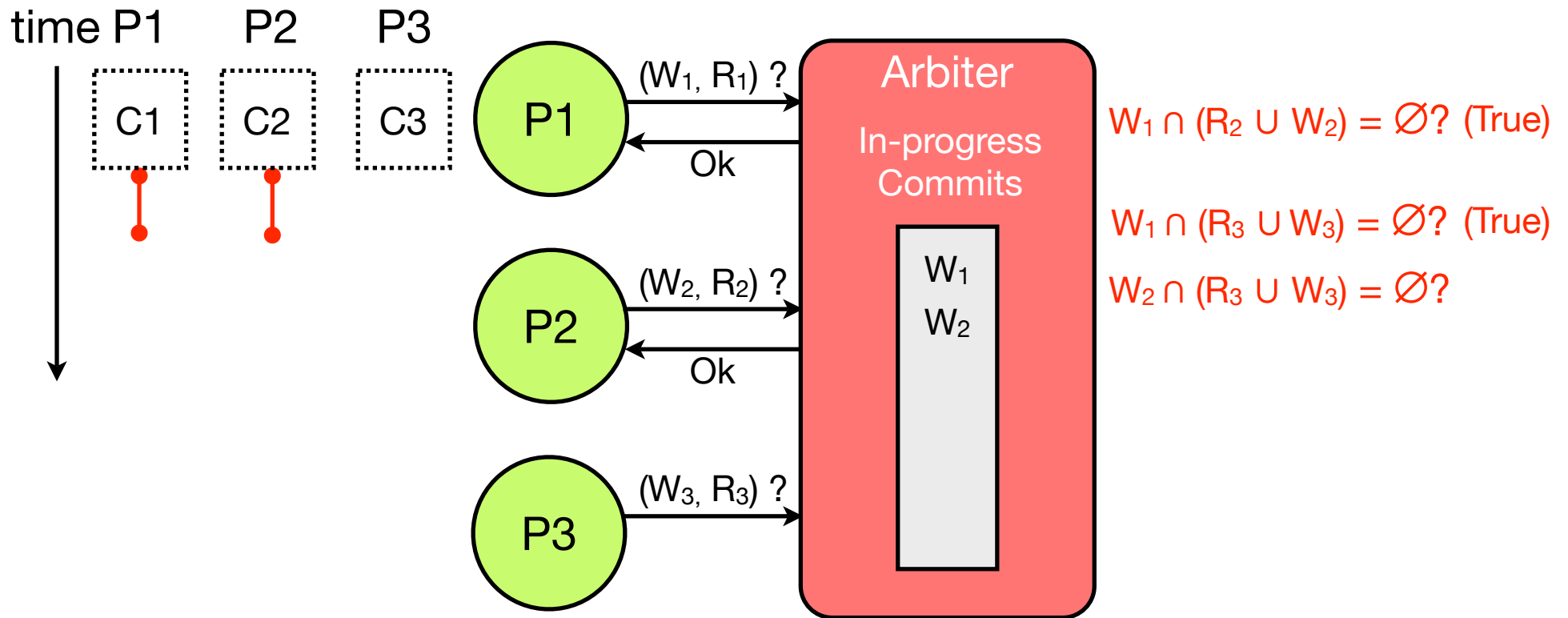
Arbitrating Simultaneous Chunk Commits



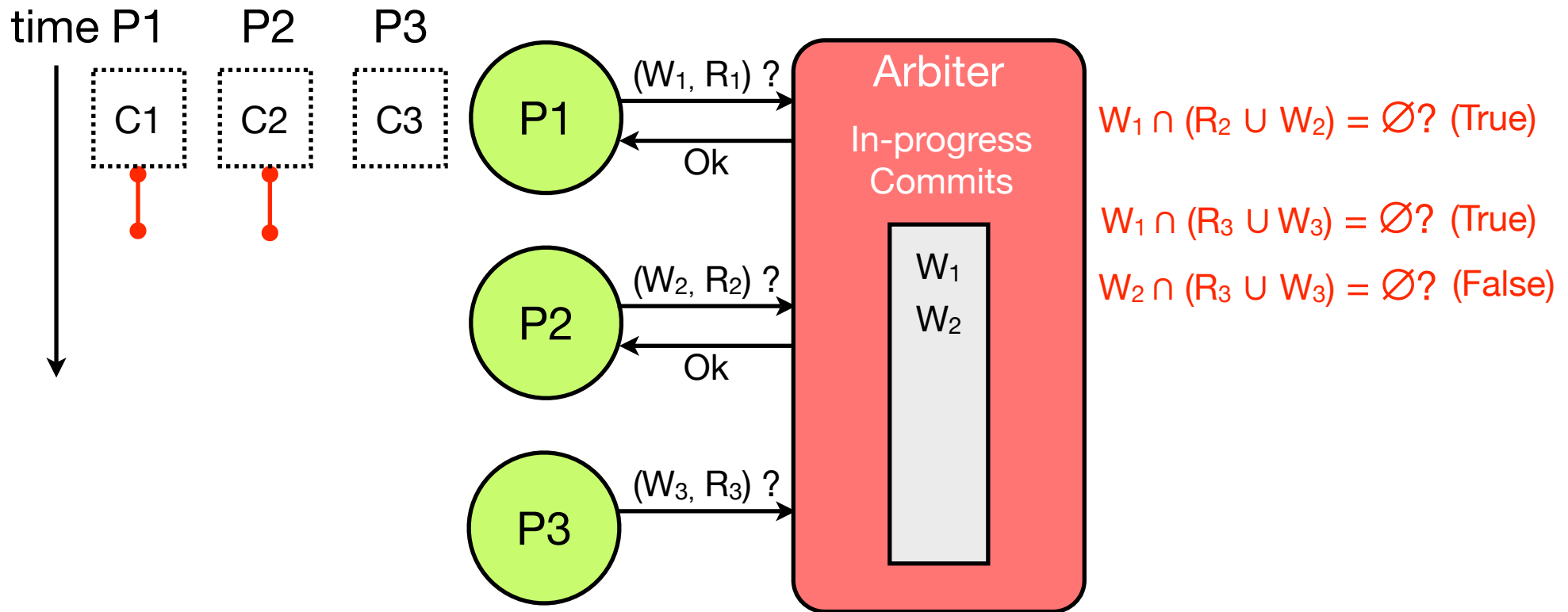
Arbitrating Simultaneous Chunk Commits



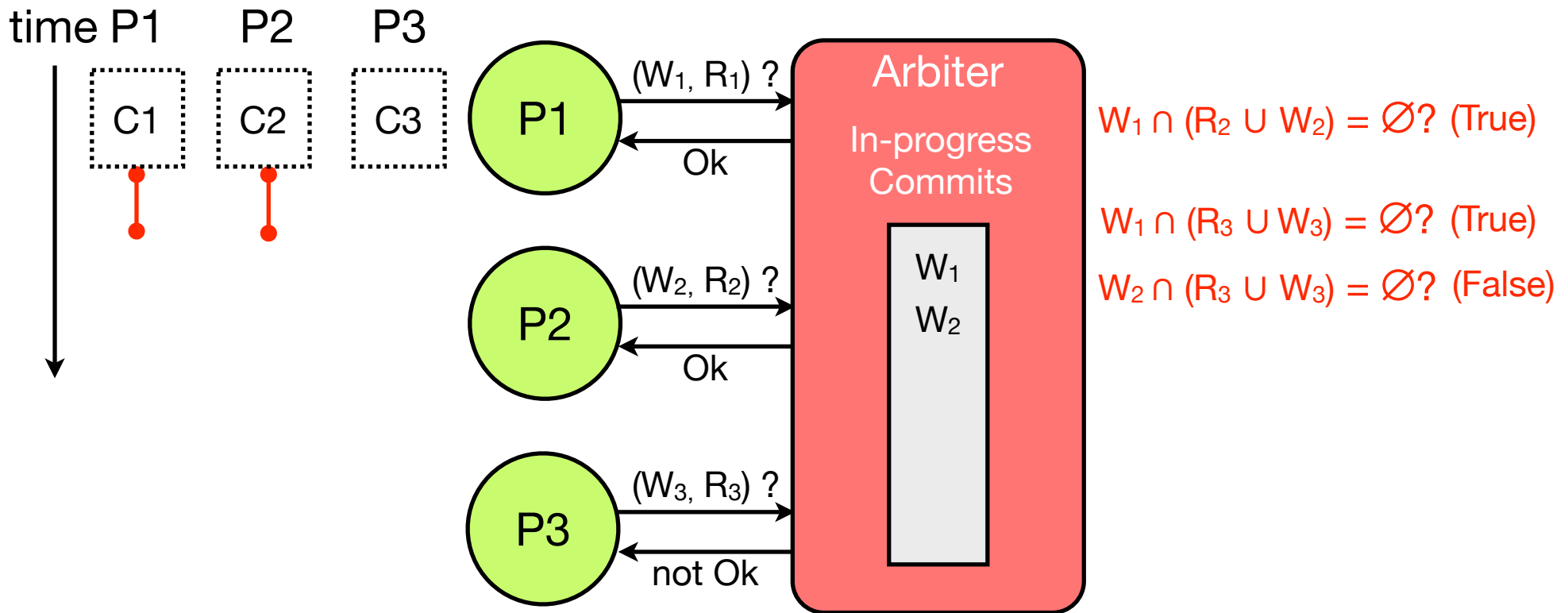
Arbitrating Simultaneous Chunk Commits



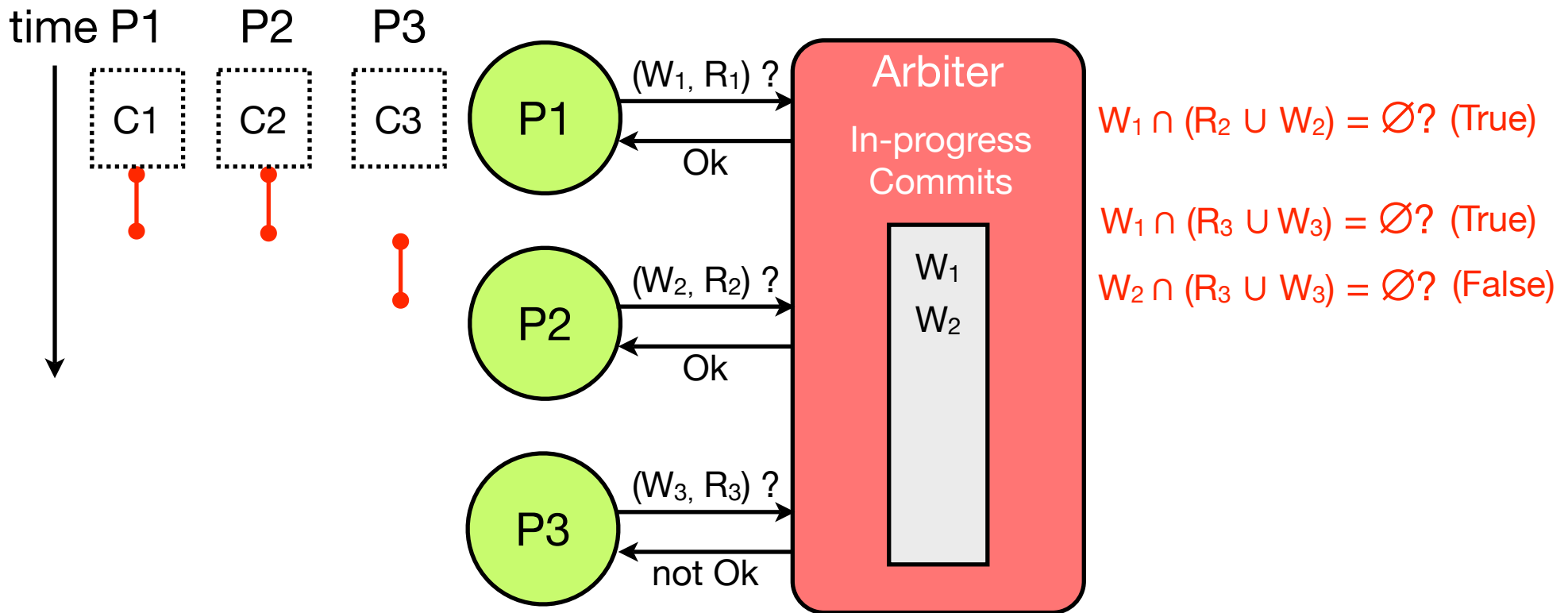
Arbitrating Simultaneous Chunk Commits



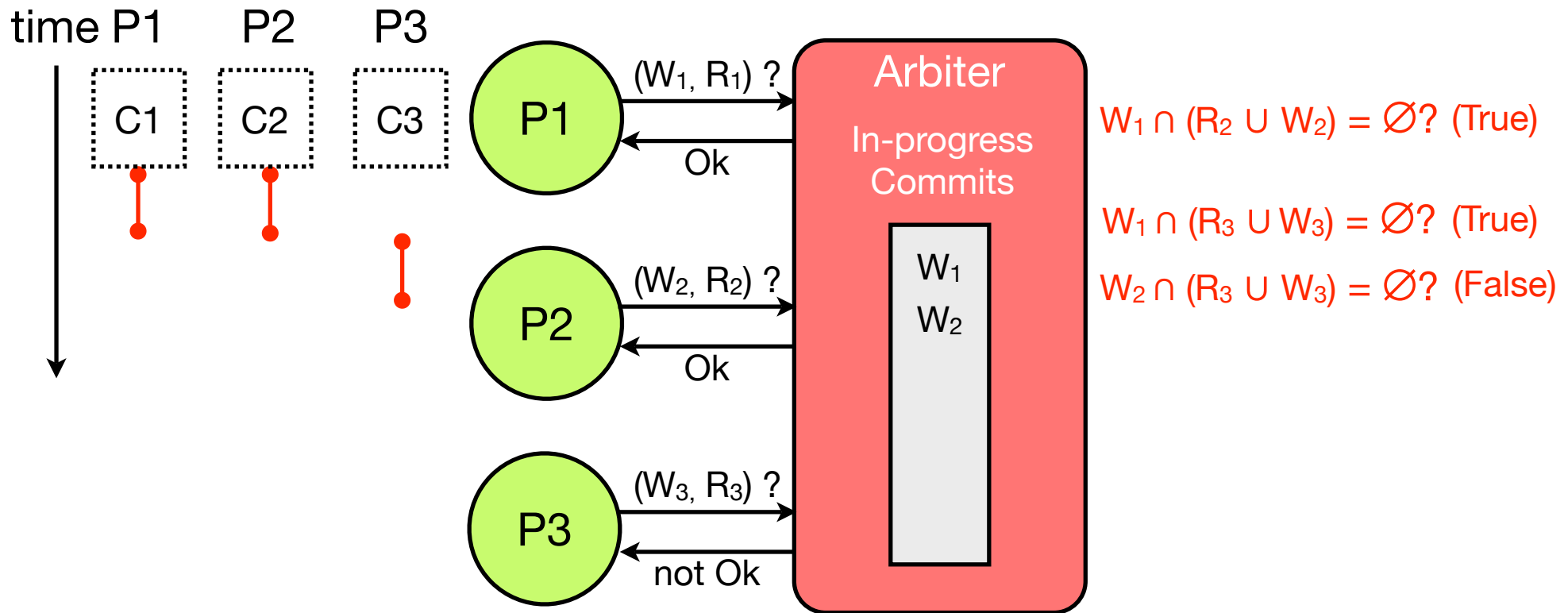
Arbitrating Simultaneous Chunk Commits



Arbitrating Simultaneous Chunk Commits



Arbitrating Simultaneous Chunk Commits

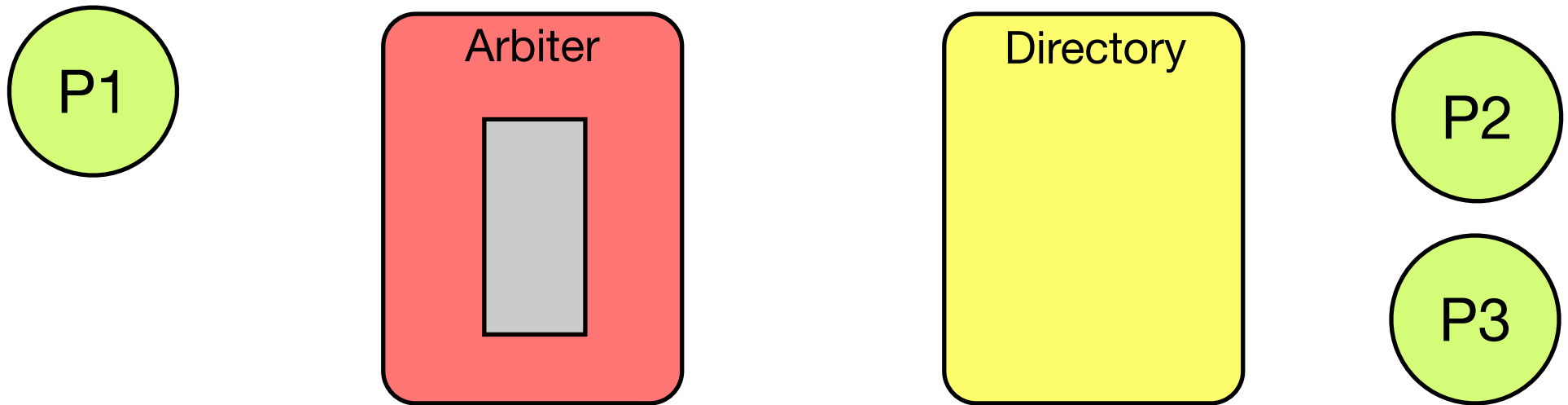


Write signatures stay in the arbiter until commit completes

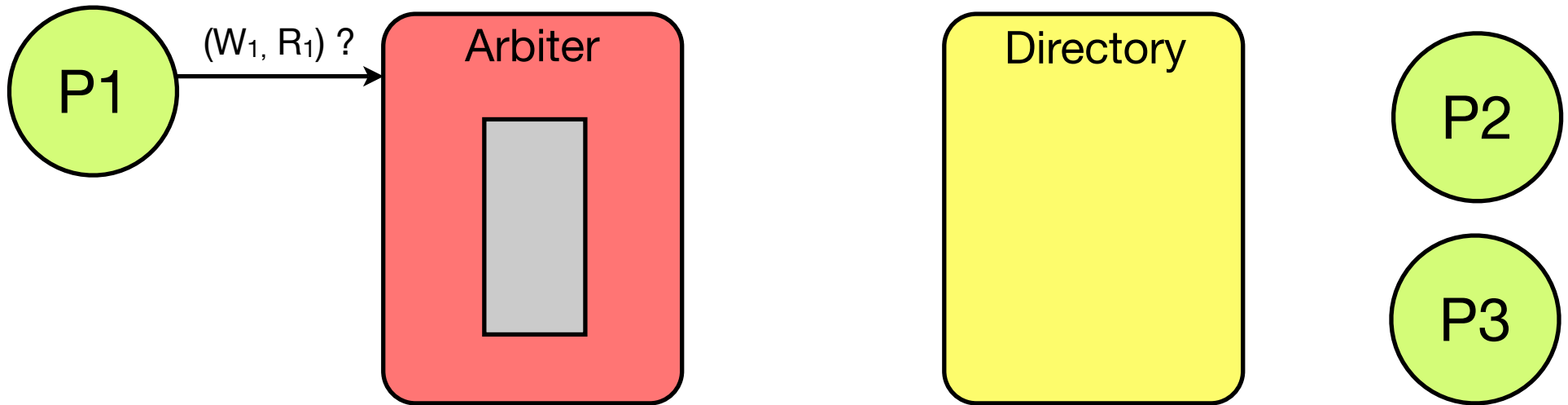
Complete Commit Process



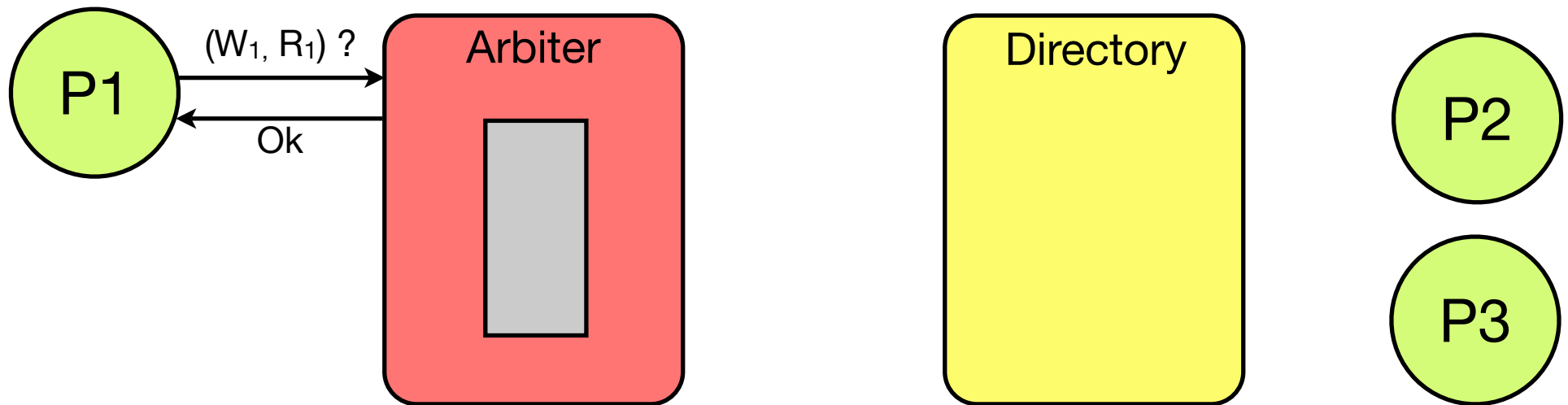
Complete Commit Process



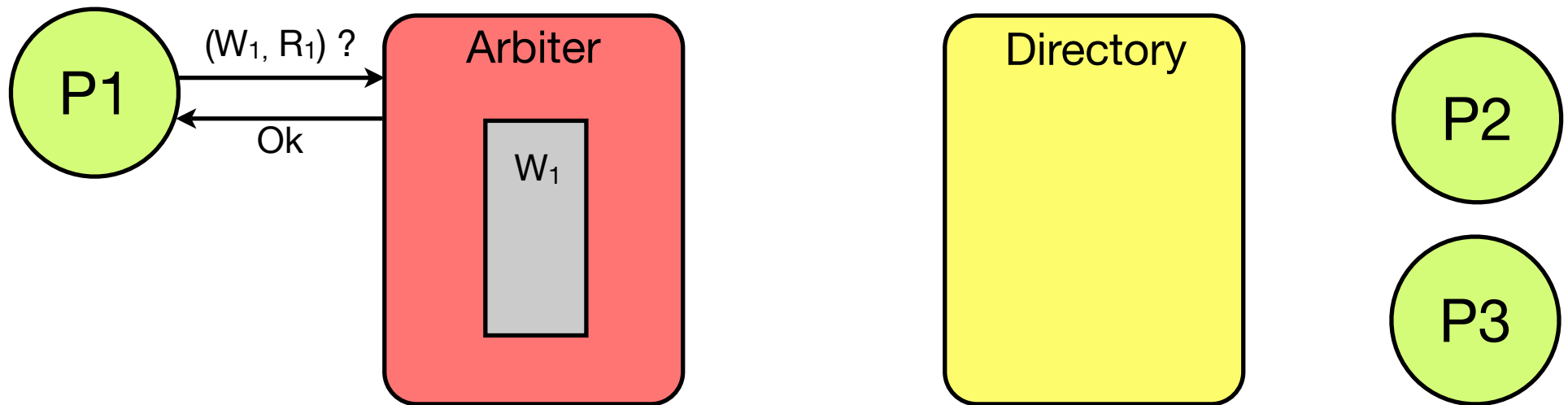
Complete Commit Process



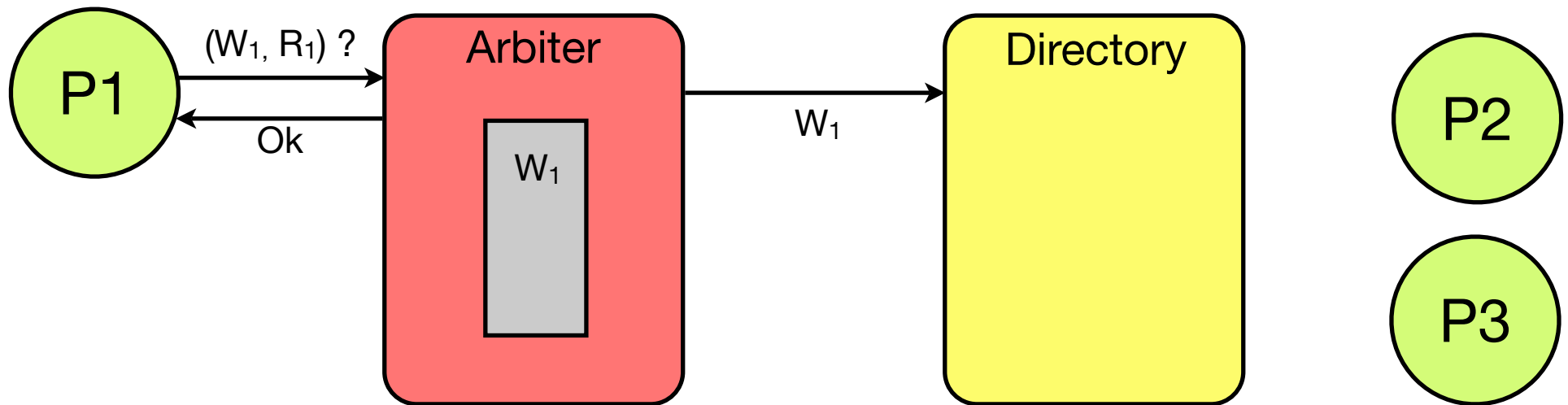
Complete Commit Process



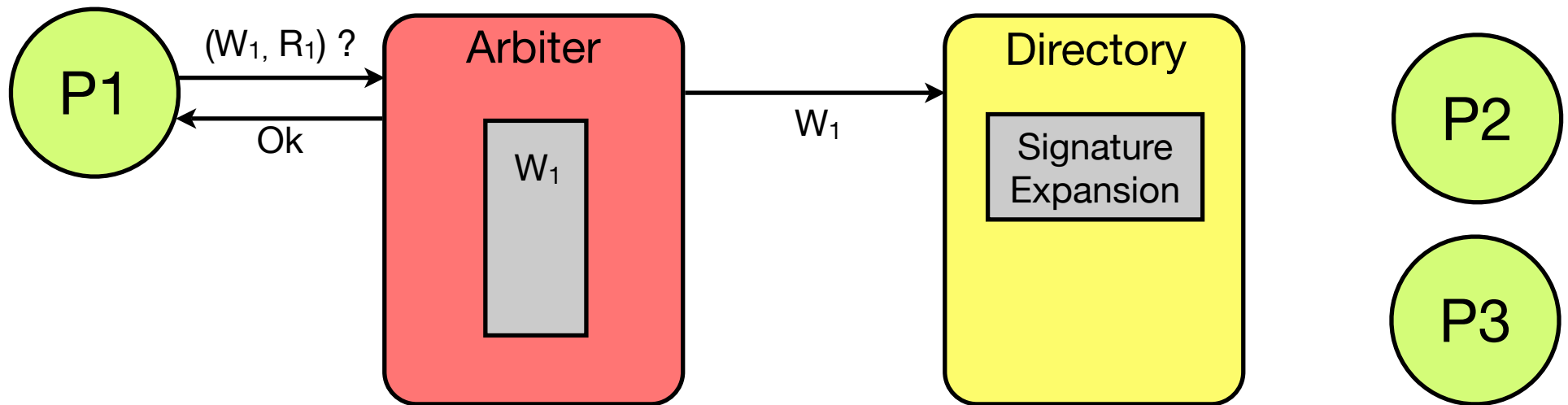
Complete Commit Process



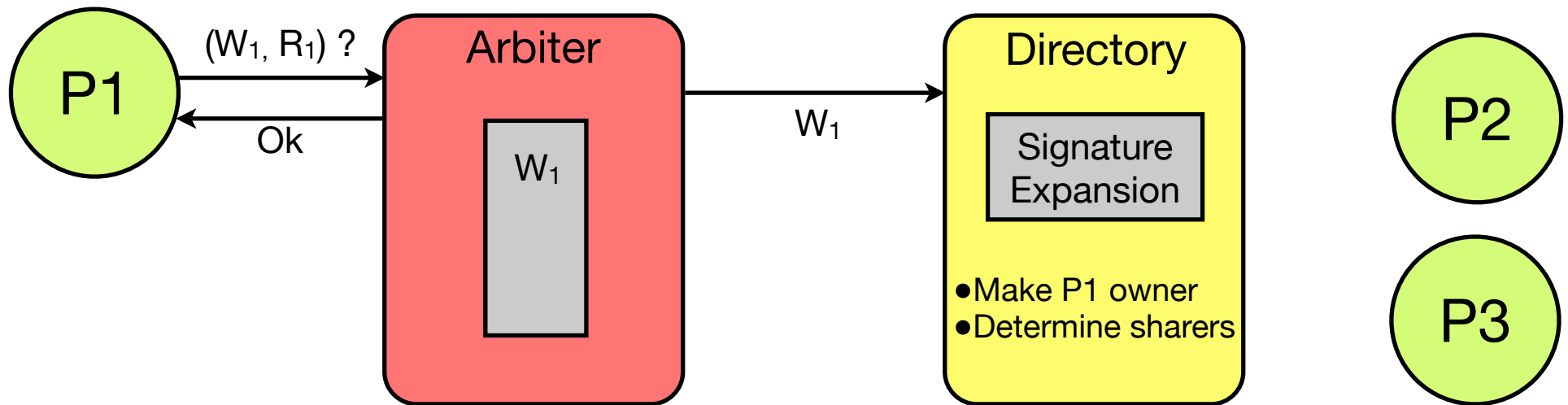
Complete Commit Process



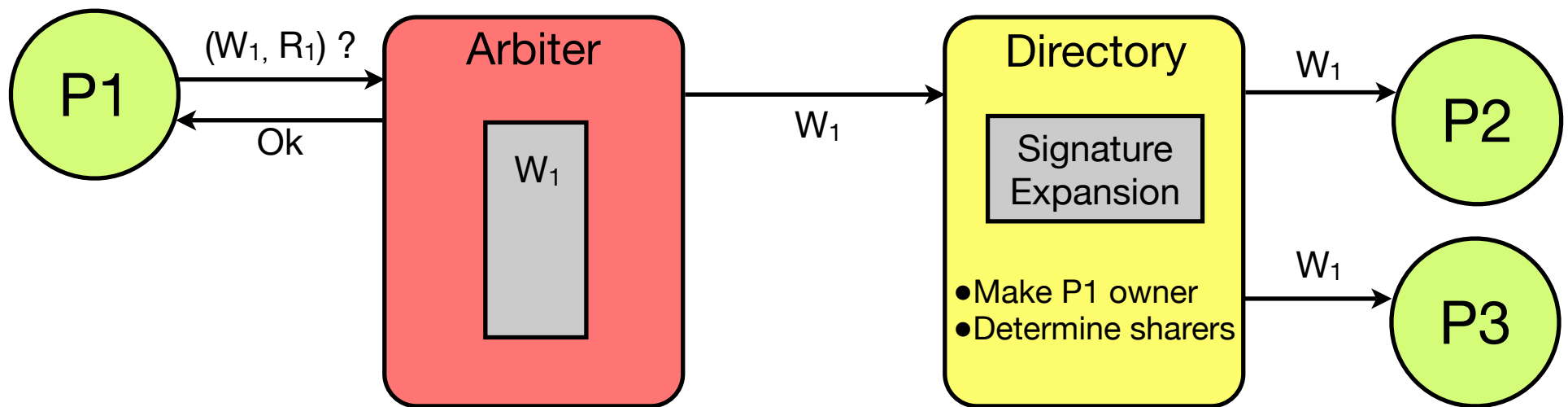
Complete Commit Process



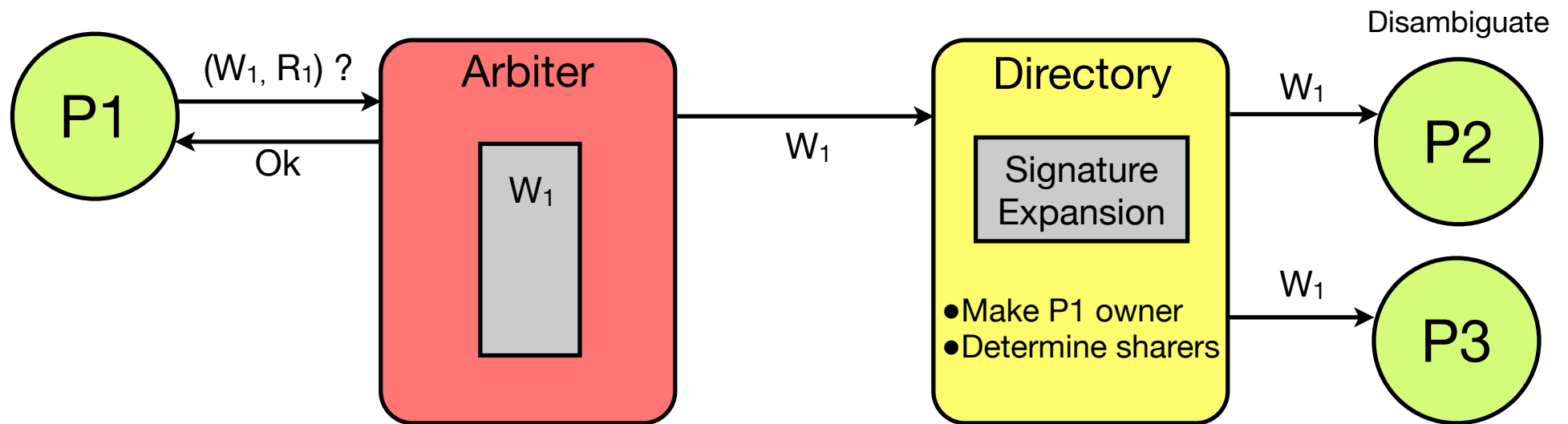
Complete Commit Process



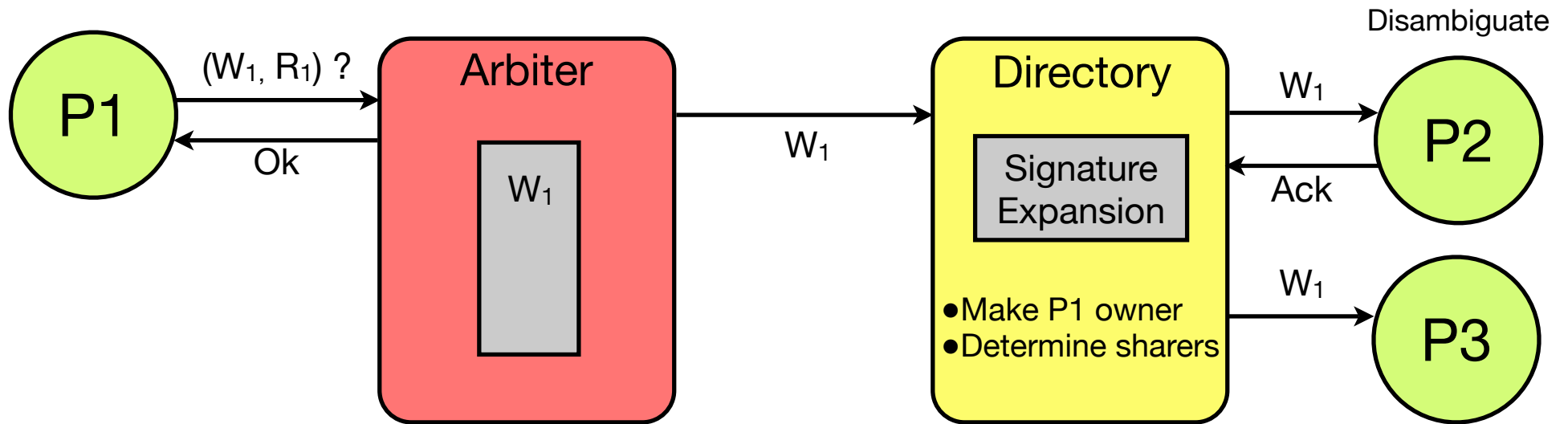
Complete Commit Process



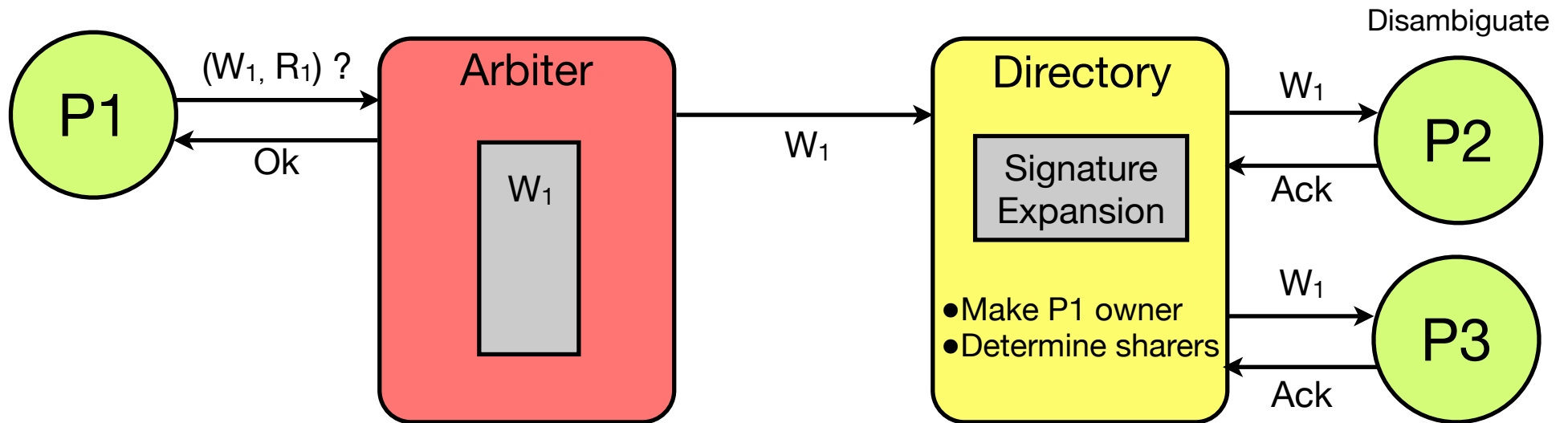
Complete Commit Process



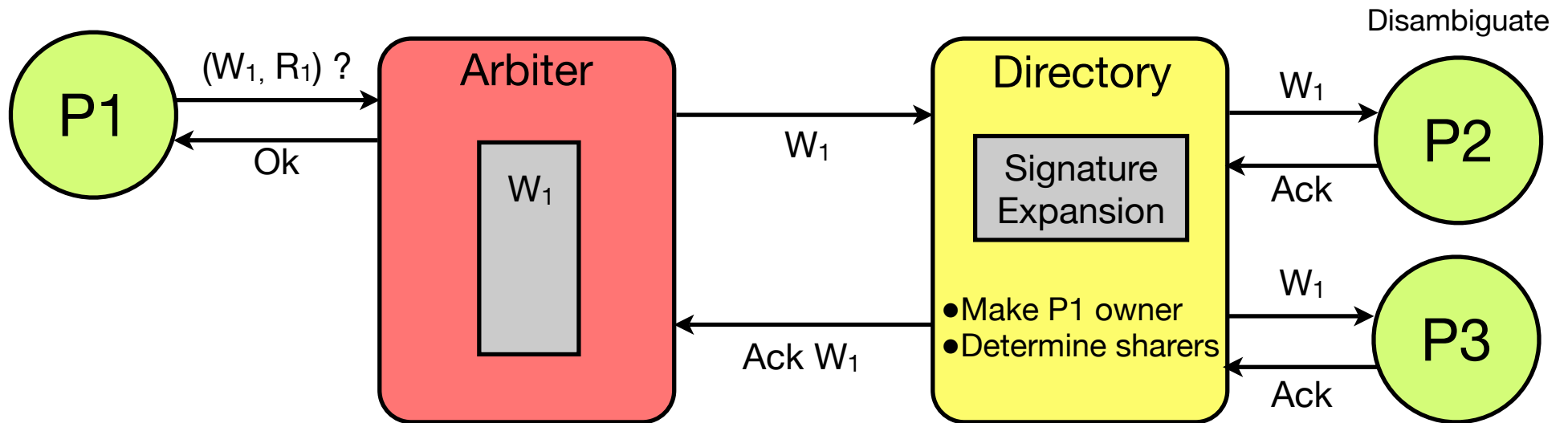
Complete Commit Process



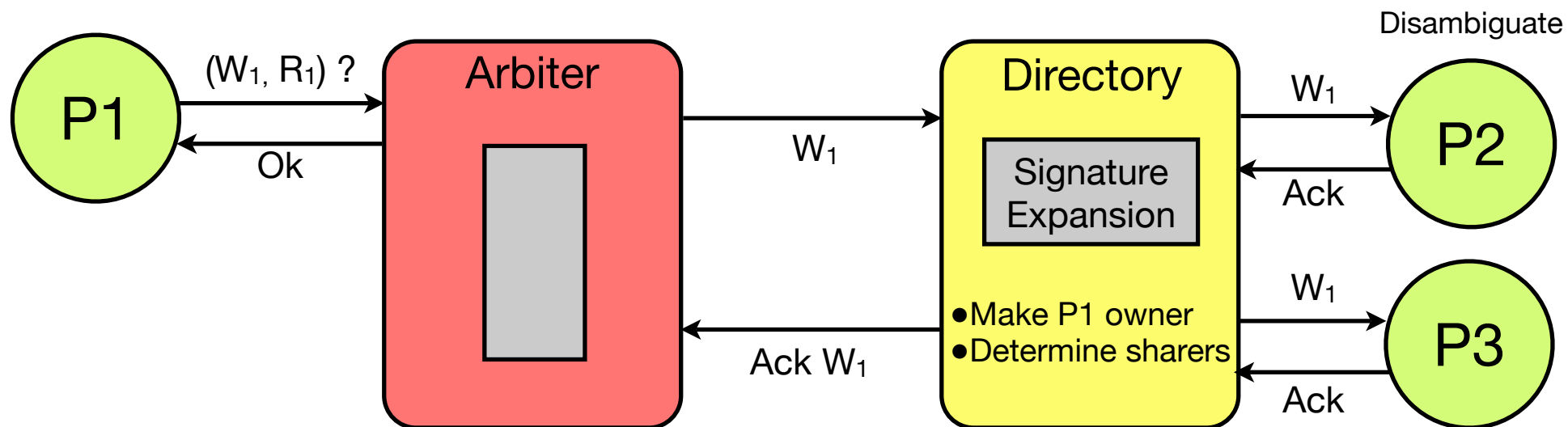
Complete Commit Process



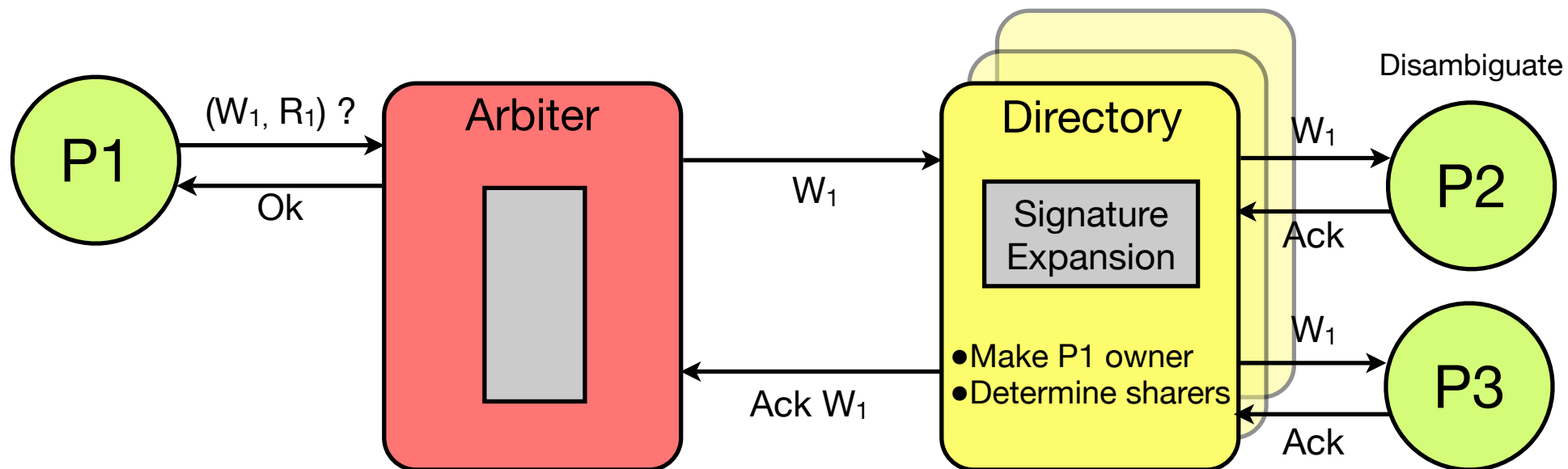
Complete Commit Process



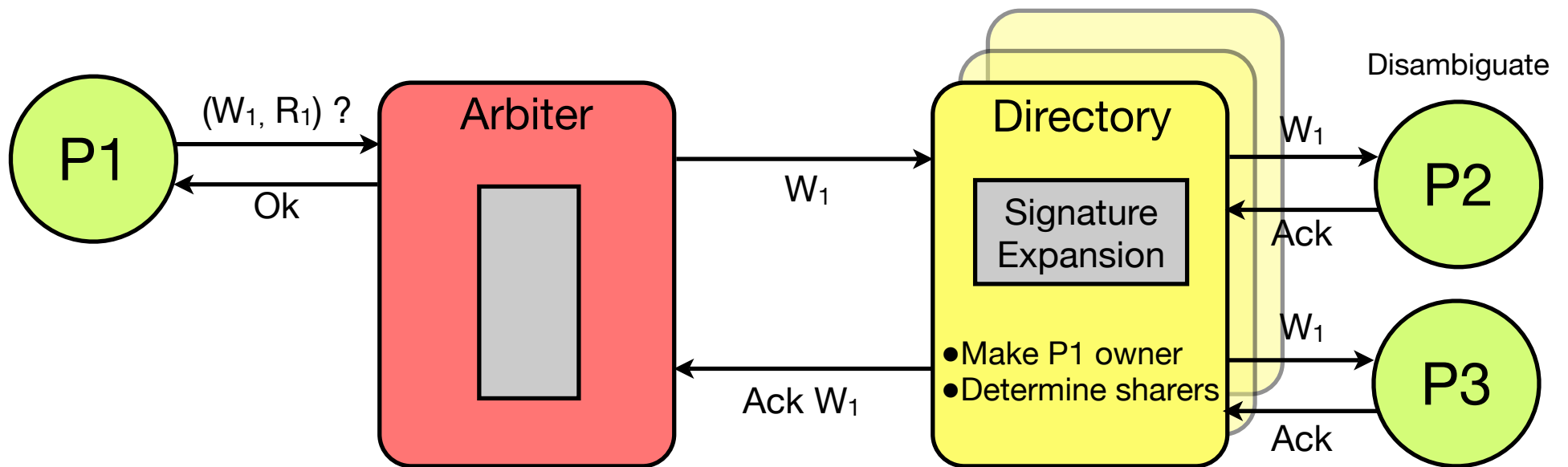
Complete Commit Process



Complete Commit Process

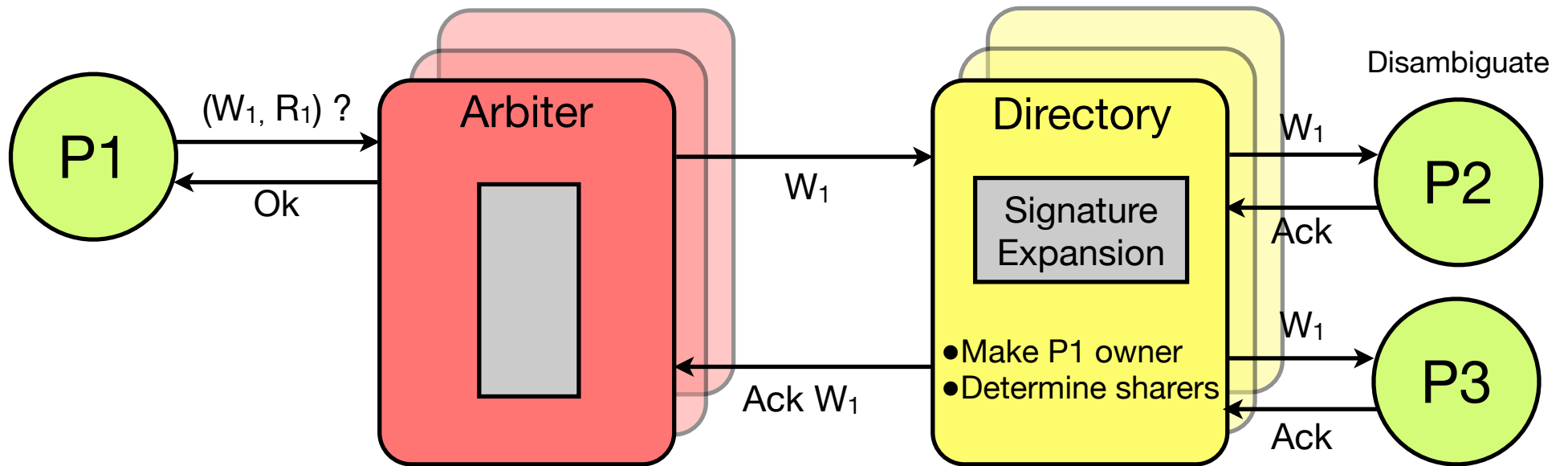


Complete Commit Process



The whole process only uses signatures

Complete Commit Process



The whole process only uses signatures
Can support a distributed arbiter

Advantages of BulkSC



Advantages of BulkSC

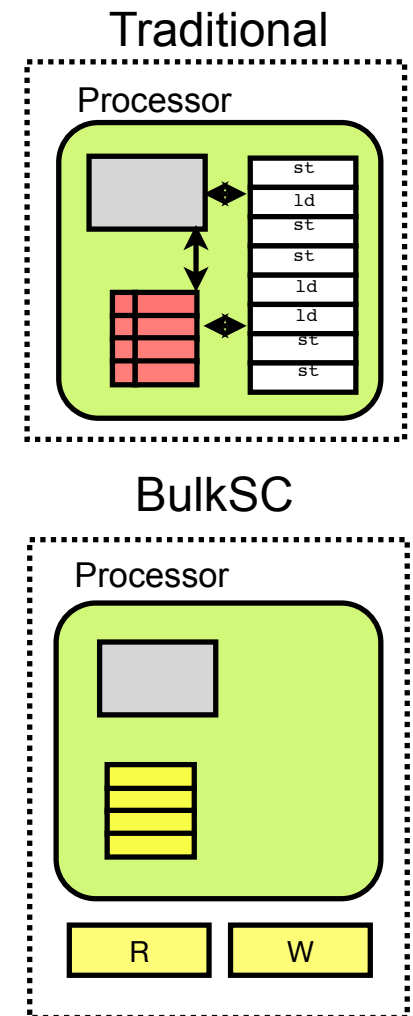
- Simplifies the HW complexity of high-performance SC

Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches

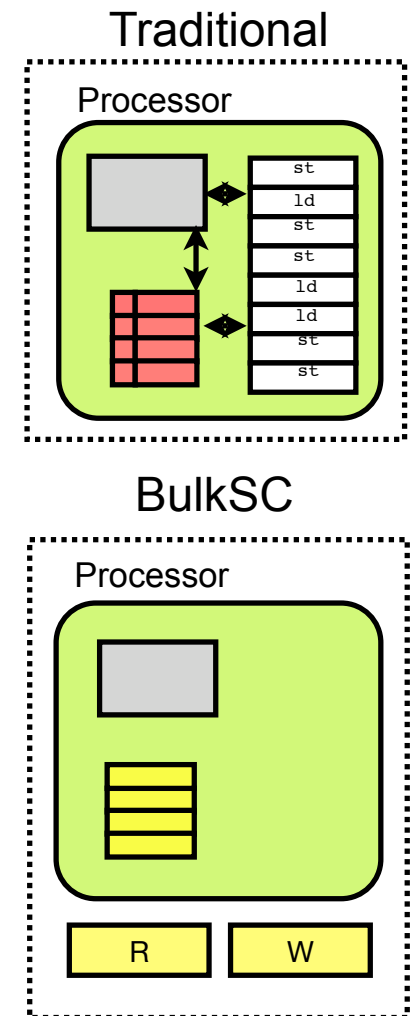
Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches



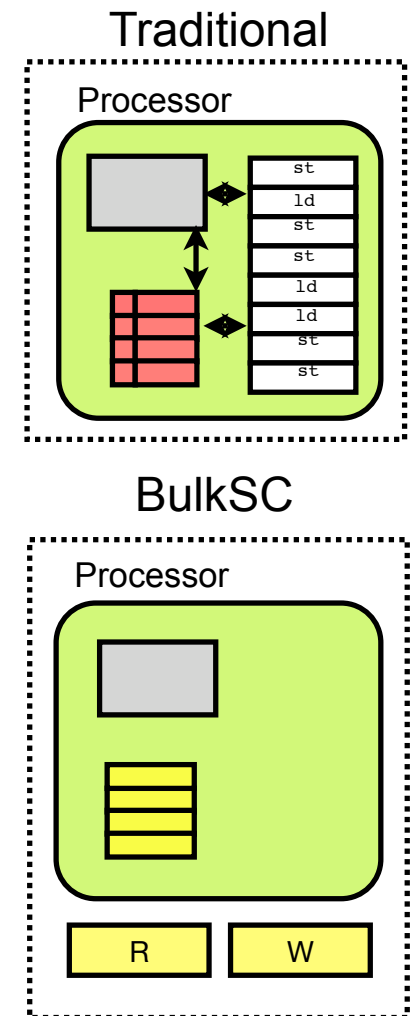
Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches
 - single-thread optimizations without worrying about multiprocessor issues



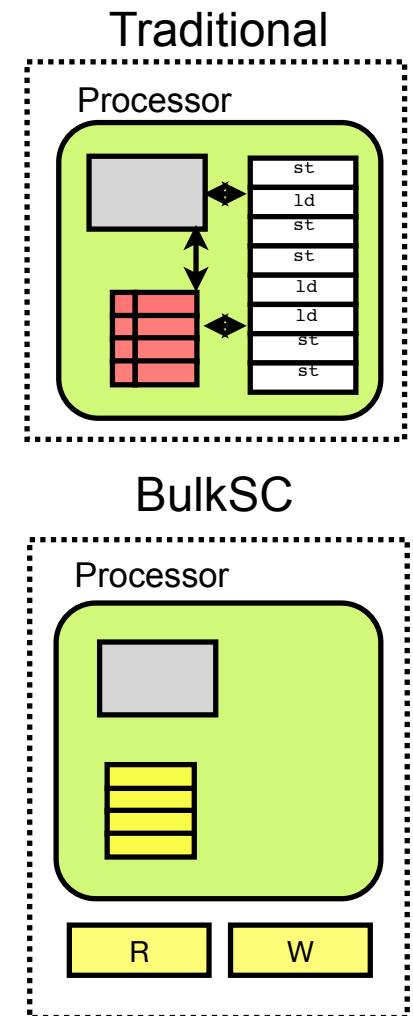
Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches
 - single-thread optimizations without worrying about multiprocessor issues
 - no associative structures in the processor



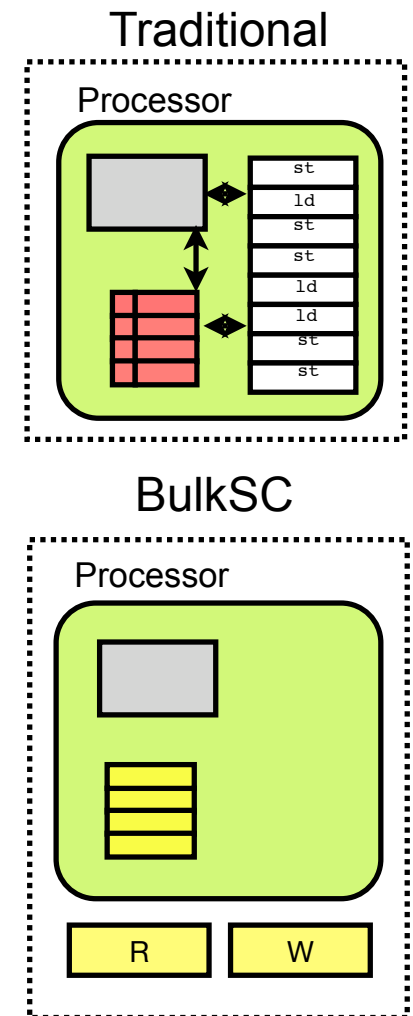
Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches
 - single-thread optimizations without worrying about multiprocessor issues
 - no associative structures in the processor
 - no extra bits in the cache for versioning [ISCA'06]



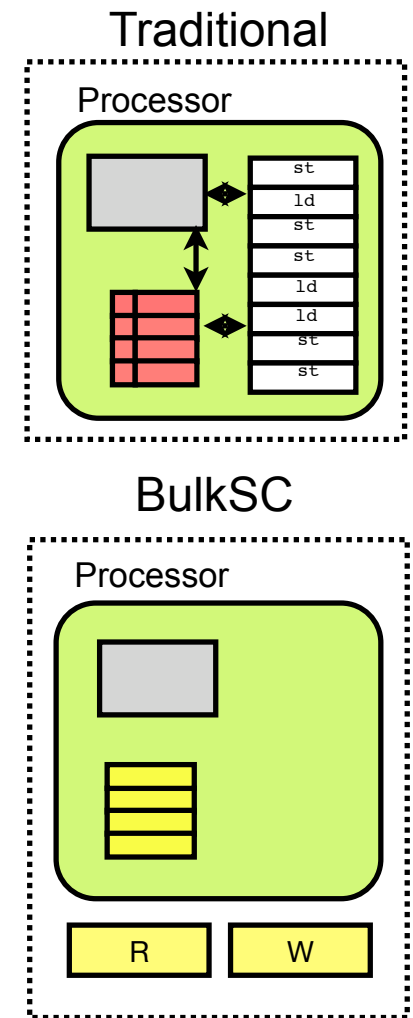
Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches
 - single-thread optimizations without worrying about multiprocessor issues
 - no associative structures in the processor
 - no extra bits in the cache for versioning [ISCA'06]
- Memory ordering framework for MPs



Advantages of BulkSC

- **Simplifies the HW complexity** of high-performance SC
 - decouples consistency enforcement from core micro-architecture and caches
 - single-thread optimizations without worrying about multiprocessor issues
 - no associative structures in the processor
 - no extra bits in the cache for versioning [ISCA'06]
- **Memory ordering framework for MPs**
 - small extension to support TM, TLS, ...



Summary of Evaluation

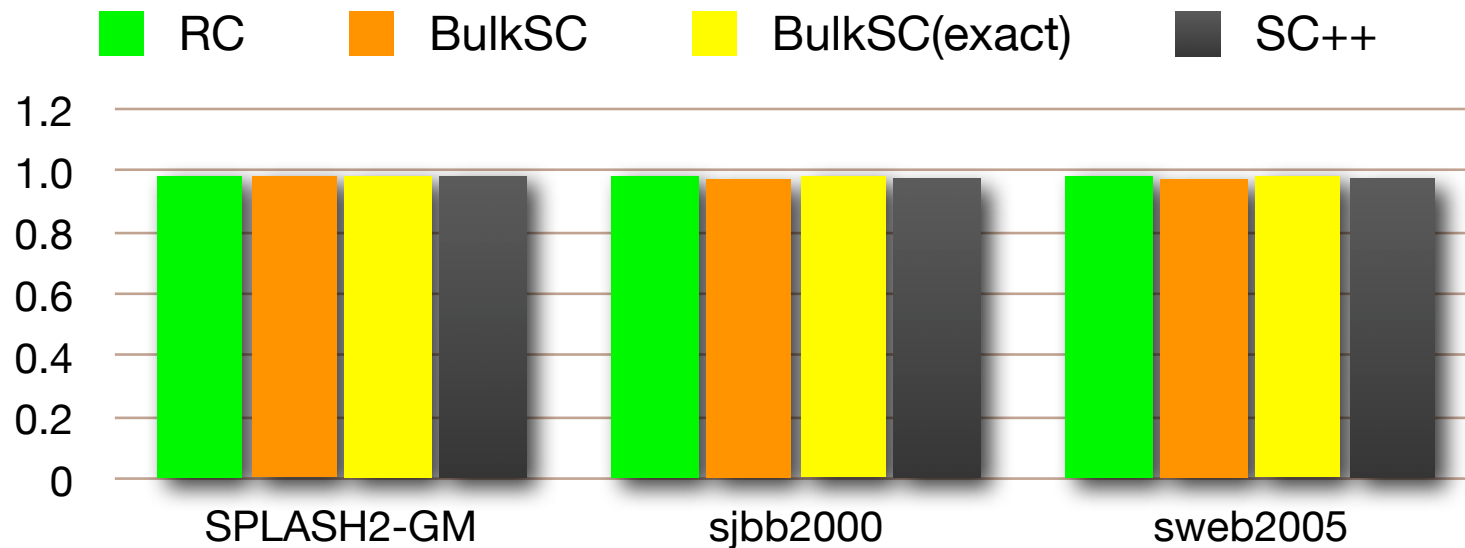


Summary of Evaluation

- Cycle-accurate simulations [SESC]

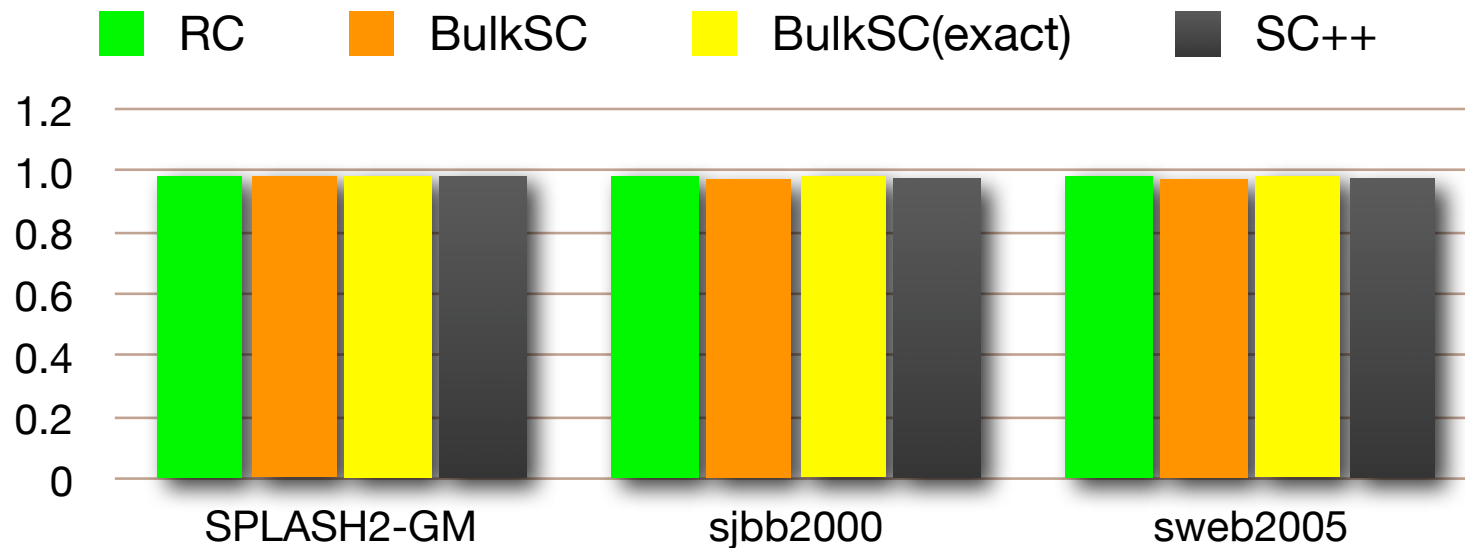
Summary of Evaluation

- Cycle-accurate simulations [SESC]



Summary of Evaluation

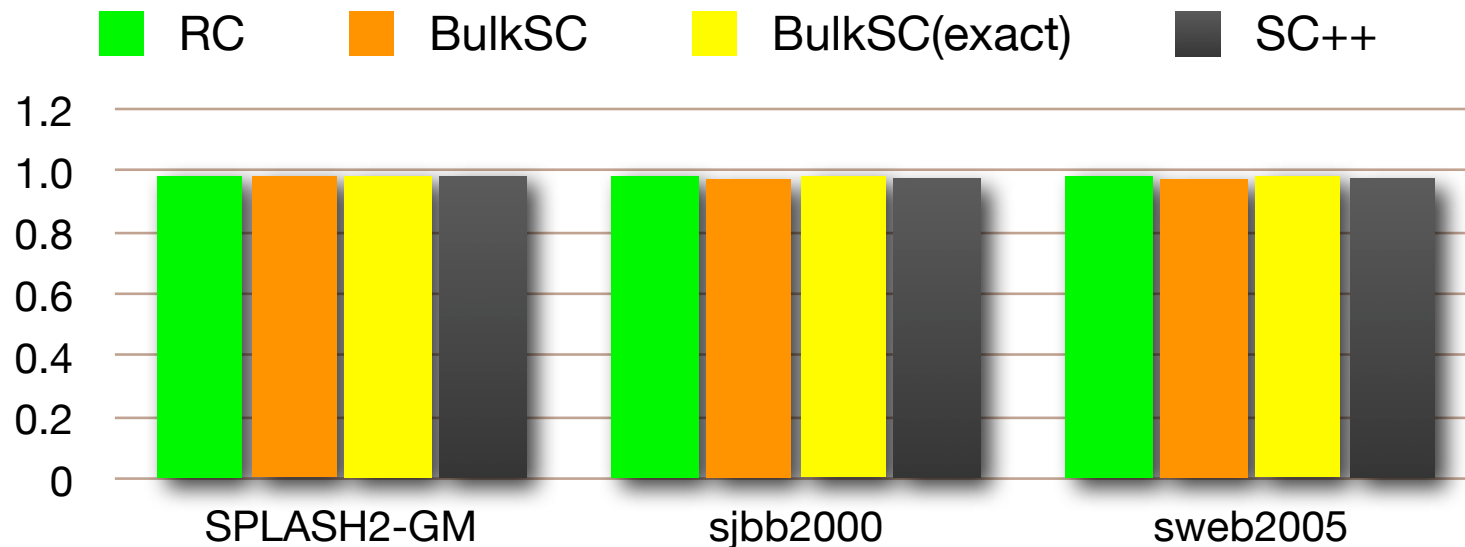
- Cycle-accurate simulations [SESC]



- **Result:** SC with performance comparable to RC (1%) with little BW cost (5-13%)

Summary of Evaluation

- Cycle-accurate simulations [SESC]



- **Result:** SC with performance comparable to RC (1%) with little BW cost (5-13%)
- Much simpler hardware than SC++

Also on the paper...



Also on the paper...

- Interaction with explicit synchronization, TM

Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress

Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O

Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O
- Distributed arbiter

Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O
- Distributed arbiter
- Directory design for signatures

Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O
- Distributed arbiter
- Directory design for signatures
- Two optimizations for private data

Also on the paper...

- Interaction with explicit synchronization, TM
- Forward progress
- I/O
- Distributed arbiter
- Directory design for signatures
- Two optimizations for private data
- Discussion on scalability

Conclusions



Conclusions

- Presented BulkSC

Conclusions

- Presented BulkSC
 - coarse-grain, signature-based enforcement of SC

Conclusions

- Presented BulkSC
 - coarse-grain, signature-based enforcement of SC
- Performance comparable to RC

Conclusions

- Presented BulkSC
 - coarse-grain, signature-based enforcement of SC
- Performance comparable to RC
- Simplicity: decouple consistency enforcement from processor design

Conclusions

- Presented BulkSC
 - coarse-grain, signature-based enforcement of SC
- Performance comparable to RC
- Simplicity: decouple consistency enforcement from processor design
- Independent of network ordering properties

Conclusions

- Presented BulkSC
 - coarse-grain, signature-based enforcement of SC
- Performance comparable to RC
- Simplicity: decouple consistency enforcement from processor design
- Independent of network ordering properties
- Generic ordering framework for speculative multiprocessors



BulkSC: Bulk Enforcement of Sequential Consistency

Luis Ceze, James Tuck, Pablo Montesinos, Josep Torrellas

University of Illinois at Urbana-Champaign

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



ISCA 2007, San Diego, CA

