Speculative Versioning Cache: Unifying Speculation and Coherence

Sridhar Gopal

T.N. Vijaykumar†, Jim Smith, Guri Sohi

Multiscalar Project

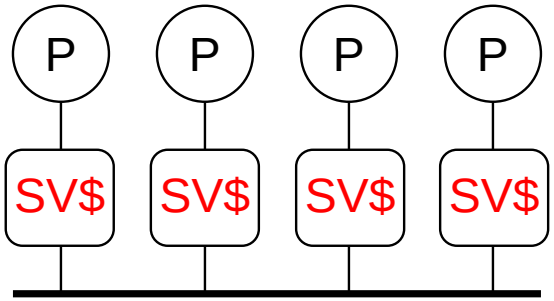
Computer Sciences Department

University of Wisconsin, Madison

†Electrical and Computer Engineering, Purdue University

Motivation

Enable a single hardware platform to support ...



1. SMP execution model - explicitly parallel programs

Private L1 coherent caches for high performance

2. Hierarchical execution model - sequential programs

- Multiscalar, Trace Processors, Agassiz, Hydra, Stampede, WarpEngine

Memory Renaming or Speculative Versioning required

SVC = CACHE COHERENCE + SPECULATIVE VERSIONING

Outline

Motivation

Hierarchical Execution Model

- Hierarchical Execution
- Multiscalar

Speculative Versioning

Unifying Speculation and Coherence

Speculative Versioning Cache (SVC)

Address Resolution Buffer (ARB)

Performance Evaluation

Conclusions

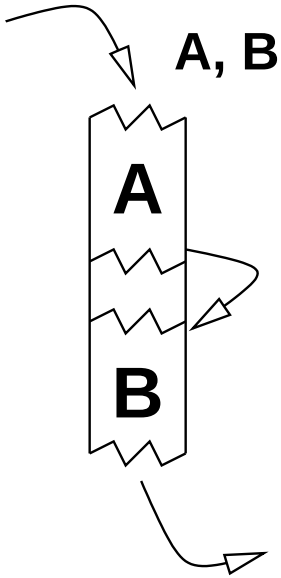
Hierarchical Execution Model

Extract Parallelism in Sequential Programs

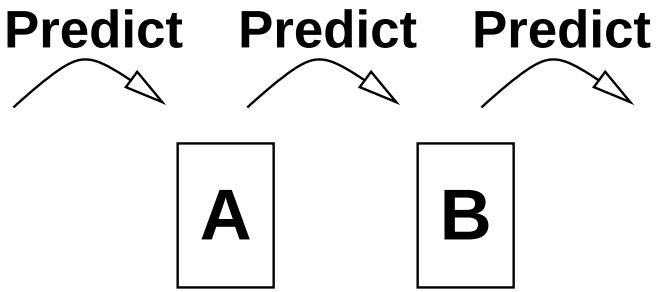
1. Using Speculation
2. Using Multiple Processors

- Group instructions into *tasks*, traces or speculation regions
- Employ task level speculation

Sequential Execution



Hierarchical Execution



Speculatively execute A and B in parallel

Dependences between A and B?

Hierarchical Execution: Multiscalar

Higher Level: Tasks

- Incorrect control prediction: **squash** task state and re-execute
- Task completed: **commit** task state **sequentially**

Lower Level: Instructions

Register dependences: Hardware + Software

Memory dependences: Hierarchical hardware

- Intra-task: Load Store Queue in each processor
- **Inter-task: SVC or ARB**

Outline

Motivation

Hierarchical Execution Model

Speculative Versioning

- Problem: Guarantee sequential program semantics
- Example execution orders
- Key requirements of a solution

Unifying Speculation and Coherence

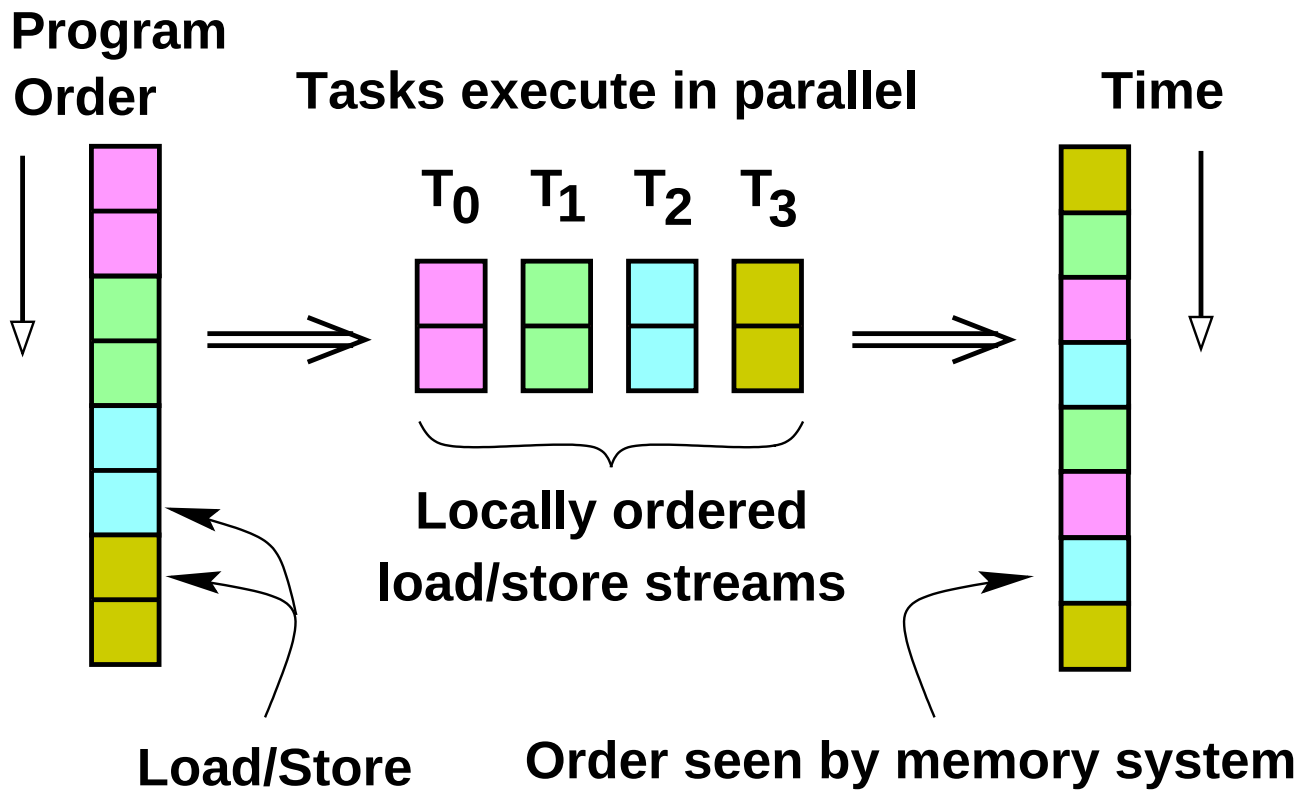
Speculative Versioning Cache (SVC)

Address Resolution Buffer (ARB)

Performance Evaluation

Conclusions

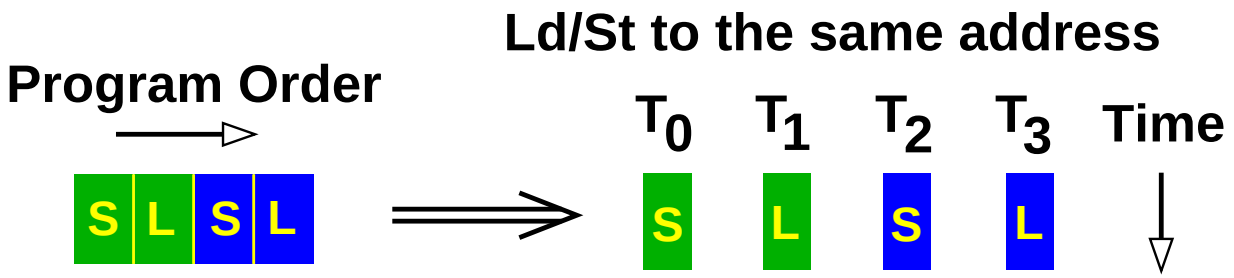
Speculative Versioning: Problem



DIFFERENT Address: Accesses can be reordered

SAME ADDRESS: WHICH ORDERS ARE ALLOWED?

Execution Order: Examples



Execution Orders →

✔	S L S L	**No action!**
✔	S S L L	**Out of order loads okay!**
?	S S L L	**Buffer green store, Supply correct value**
✘	S L L S	**Blue load is incorrect!**
?	S S L L	**Stores out of order, Write back in order**

CANNOT REORDER DEPENDENT LOADS AND STORES

Speculative Versioning: Solution

Definition: Version of a location

- Each store creates a new version

What should a solution provide?

- Buffer multiple versions and track order
- Supply the correct version for a load
- Detect incorrect loads
- Write back versions in order

Definition: Version Ordering List (VOL) of a location

- Execution order of loads and stores

DIRECTORY OF VERSION ORDERING LISTS

Unification

Speculative Versioning

- Multiple copies of **multiple speculative** versions
- Execution order tracked using an **ordered** list

Multiple Reader **Multiple** Writer Protocol

Cache Coherence

- Multiple copies of a **single** version
- Sharers tracked using an **unordered** list

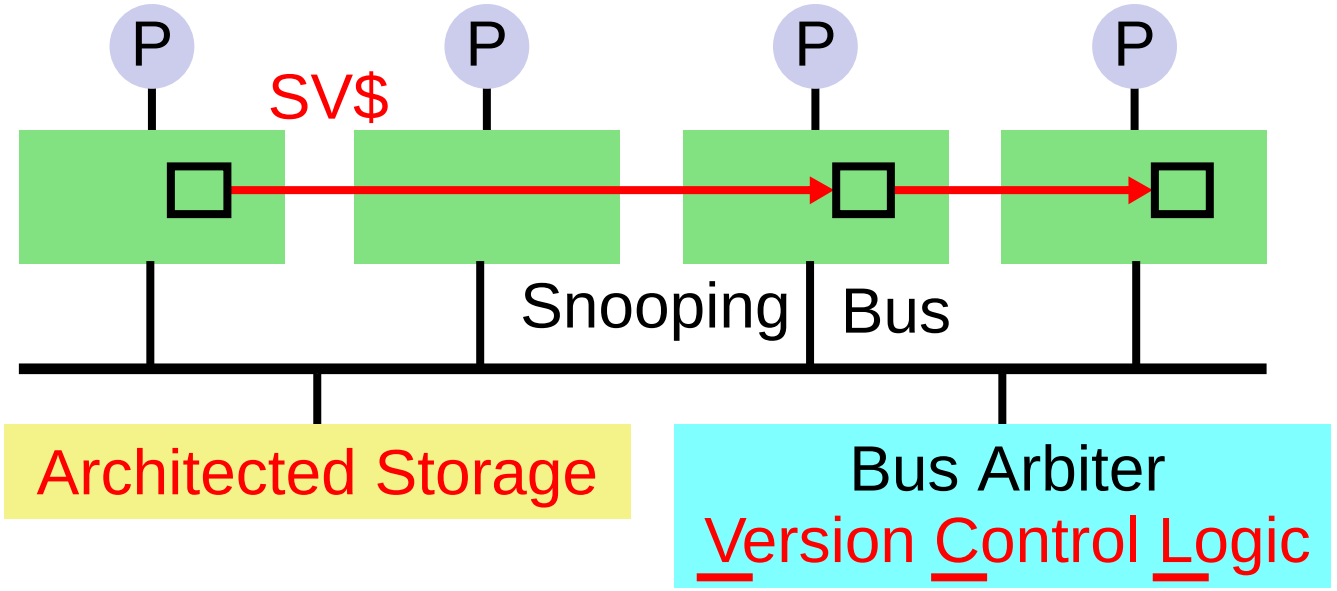
Multiple Reader **Single** Writer Protocol

SVC = CACHE COHERENCE + SPECULATIVE VERSIONING

Outline

- Motivation
- Hierarchical Execution Model
- Speculative Versioning
- Speculative Versioning Cache (SVC)
  - Simple Cache Coherence Protocol
  - Base SVC Protocol
  - Advanced SVC Protocol
  - Examples
- Address Resolution Buffer (ARB)
- Performance Evaluation
- Conclusions

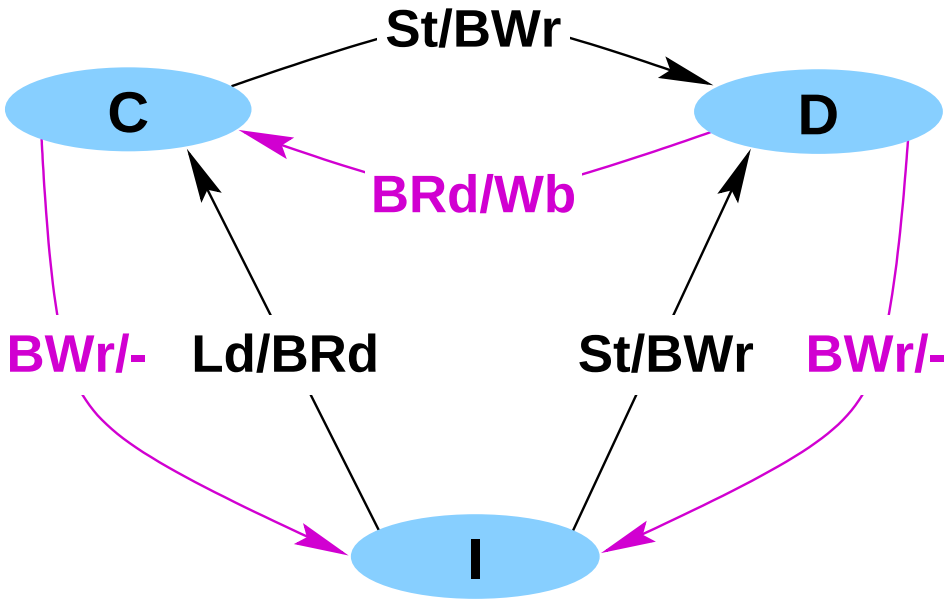
Speculative Versioning Cache



Extensions to SMP coherent cache

- Both speculative *and* committed data
- More state information
- VOL: a pointer in each line
- VCL: *maintain* VOL on bus requests

Simplified SMP Cache Coherence

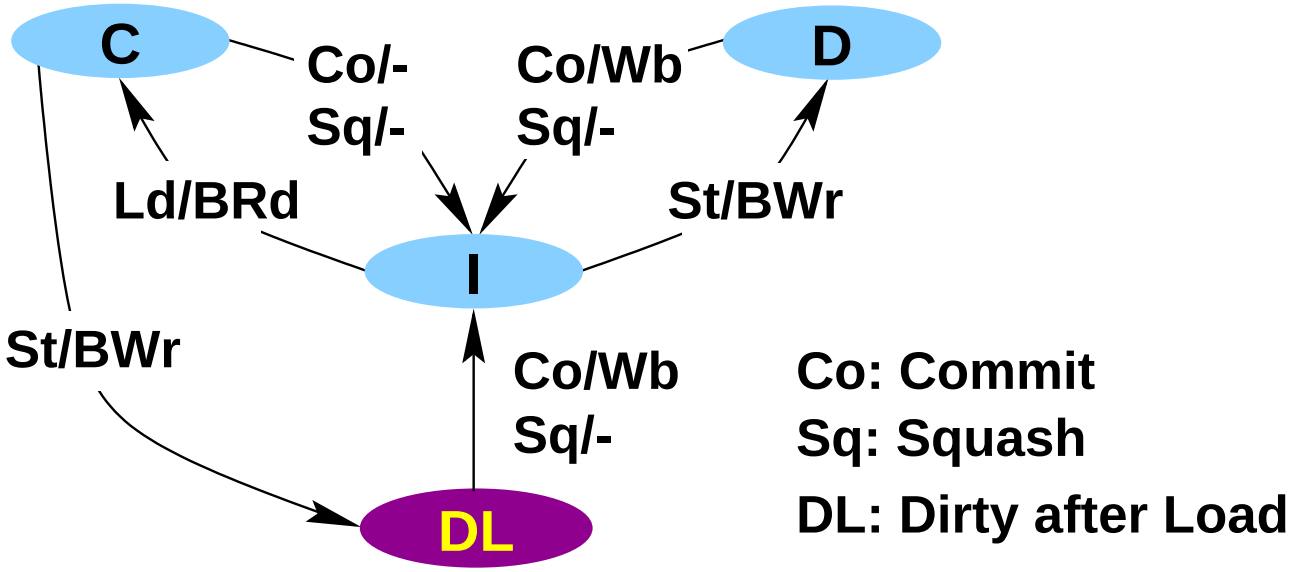


I	Invalid
C	Clean
D	Dirty
Ld	Load
St	Store
BRd	BusRead
BWr	BusWrite
Wb	Writeback

TALK Cache block size = Load/Store granularity = One word

PAPER Realistic linesize, Cast-outs

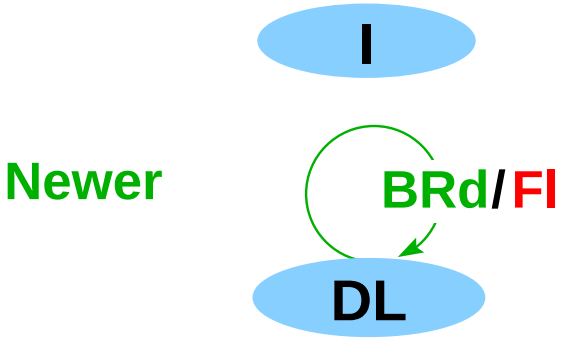
Base SVC Design: Processor Requests



- **Load** bit or **DL**: detect incorrect loads
- DL remembers use before definition
- Write back **all** dirty lines on commit
- Invalidate **all** clean lines on squash

BusRead: Supplying the Correct Version

Fl: Flush



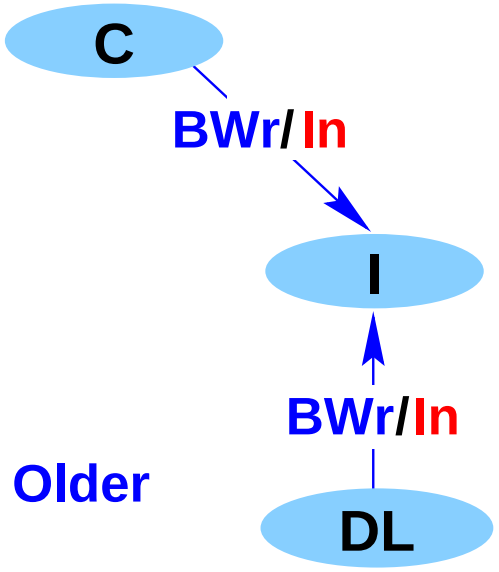
Examples

Program Order →

	T_0	T_1	T_2	T_3	T_4
1.	D	D	C	(I)	D
2.	C	I	C	(I)	D

- VCL: Closest older version is **Flushed**
- Self Loops: Cannot write back speculative versions

BusWrite: Detecting Incorrect Loads



In: Invalidate

Examples

Program Order →

	T_0	T_1	T_2	T_3	T_4
1.	C	I	C	C	D
2.	C	I	C	C	DL

- VCL: Copies used by newer tasks **Invalidated**
- DL is also invalidated because of incorrect use

Base Design Problems and Solutions

Write back dirty lines on commit

- New tasks begin after **all** write backs
- **Bottleneck**: Tasks commit sequentially

→ Commit (C) bit

Invalidate clean lines on commit/squash

- **Every** task begins with a **cold** cache

→ sTale (T) and Architectural (A) bits

Reference Spreading

- Private caches: **Multiple** misses to the **same** data

→ Snarfing

Outline

Motivation

Hierarchical Execution Model

Speculative Versioning

Unifying Speculation and Coherence

Speculative Versioning Cache (SVC)

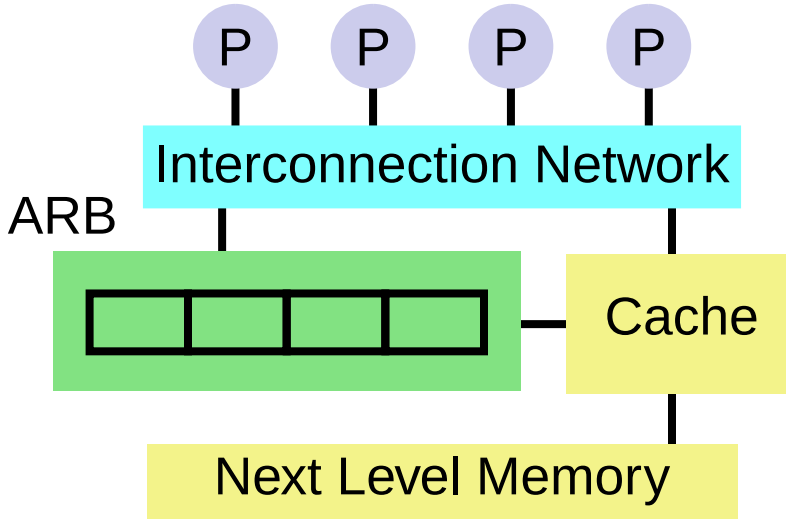
Address Resolution Buffer (ARB)

- Previously proposed shared cache solution

Performance Evaluation

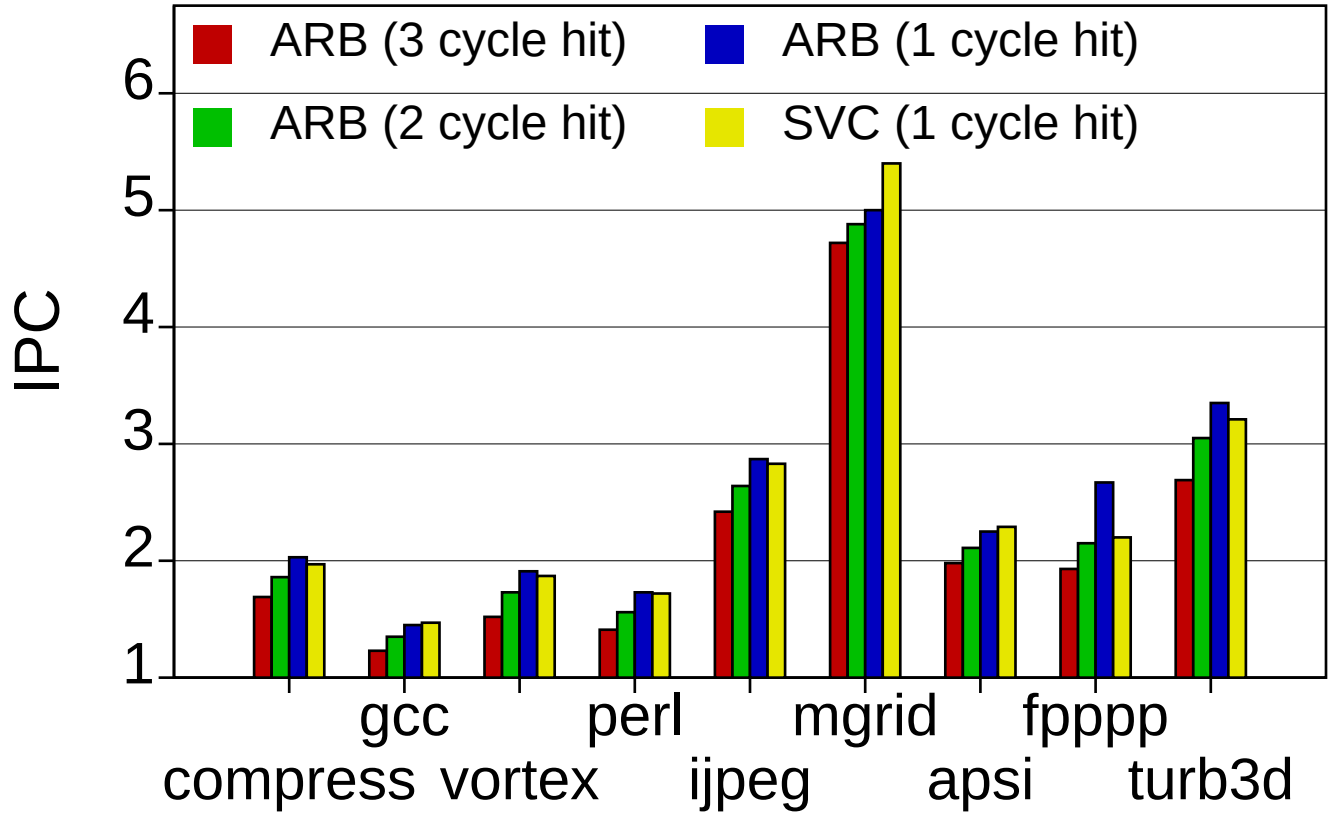
Conclusions

Shared Cache Solution: ARB



- Shared speculative ARB + Shared Cache
- Every load and store incurs interconnect latency
- On commits, speculative versions **must** be written back
- Allocates space for non-existent versions

Performance Evaluation: SPEC95



4-way Multiscalar 2-way ooo superscalar Ps

64KB data cache with 8KB ARB 4x16KB SVC

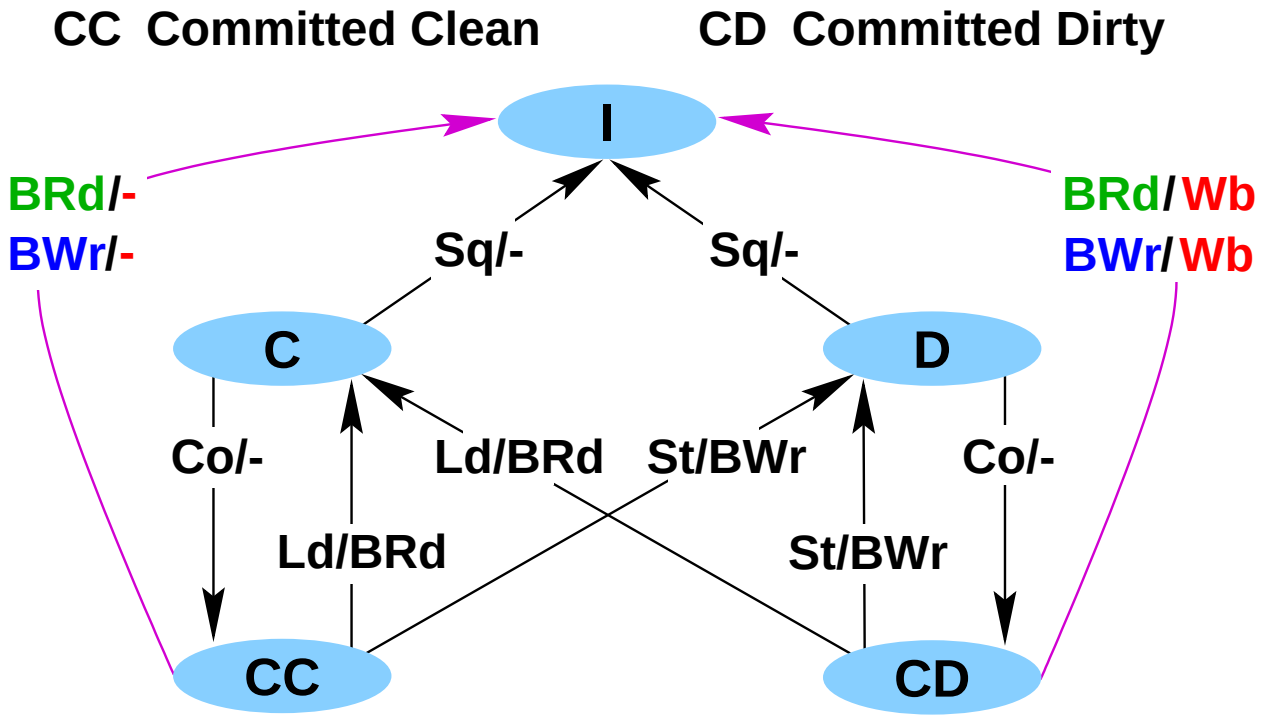
TRADES-OFF HIT-RATE FOR HIT-LATENCY

Speculative Versioning Cache

- Unifies speculative versioning and cache coherence
- Private cache solution for speculative versioning
- Hold both speculative and committed data in one cache
- Decouples write backs from commits
- Eliminates unnecessary write backs
- Trades-off hit-rate for hit-latency

SVC = CACHE COHERENCE + SPECULATIVE VERSIONING

Decoupling Commit from Write backs

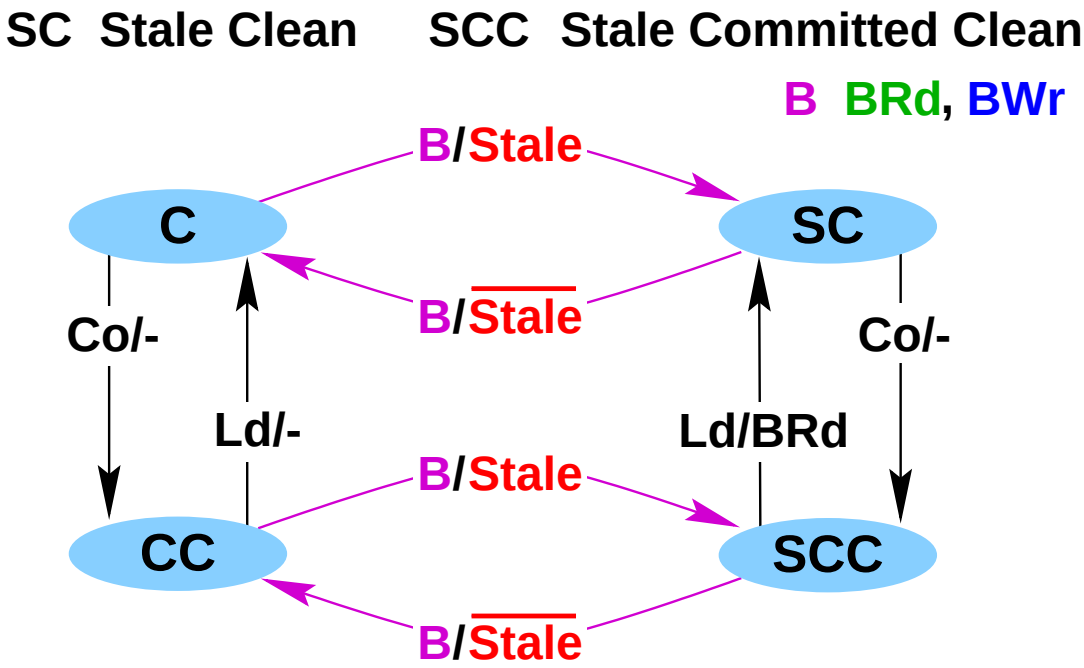


Explicit pointers necessary to track order among versions

VCL determines the CD line to be written back;

all other CD lines are purged

Keeping Cache Warm: After Commits



Problem

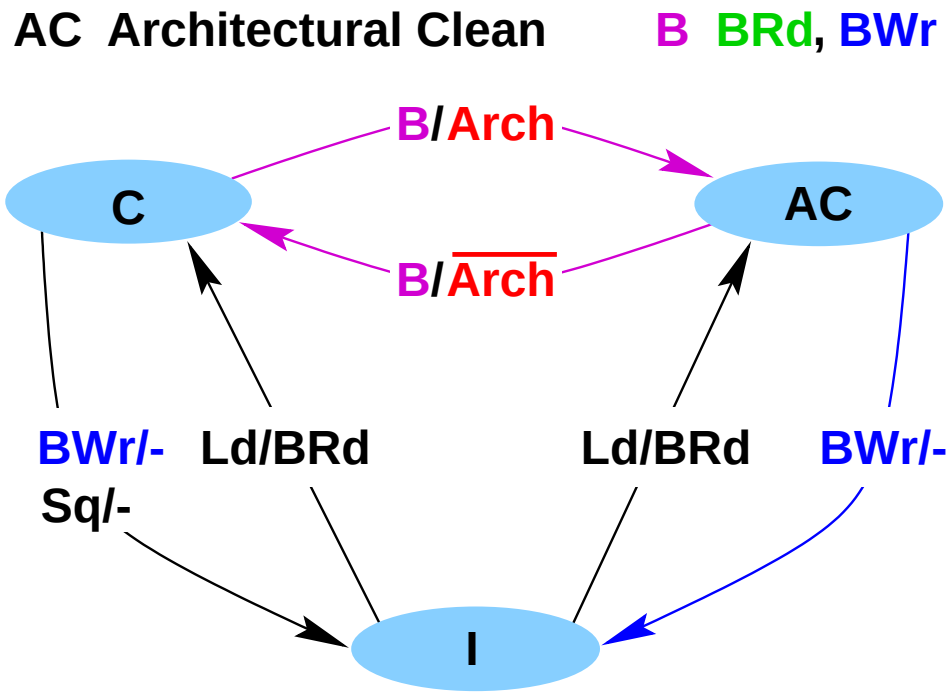
- All C lines are invalidated when a task commits

Solution

- sTale bit: to distinguish between sTale and clean copies

CC LINES ARE CACHE HITS ON LOADS

Keeping Cache Warm: After Squashes



Problem

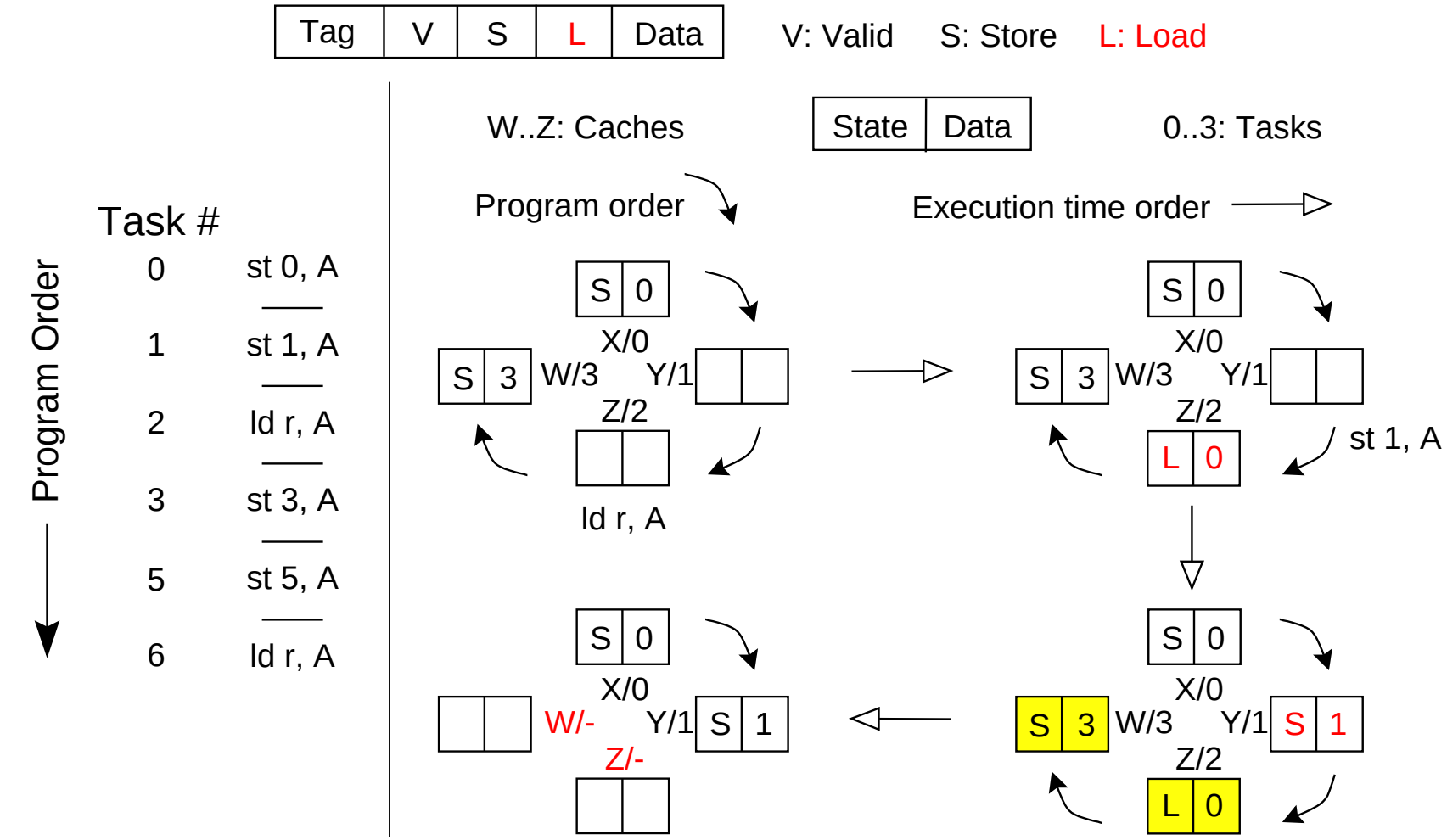
- A squashed task could fetch the same data multiple times

Solution

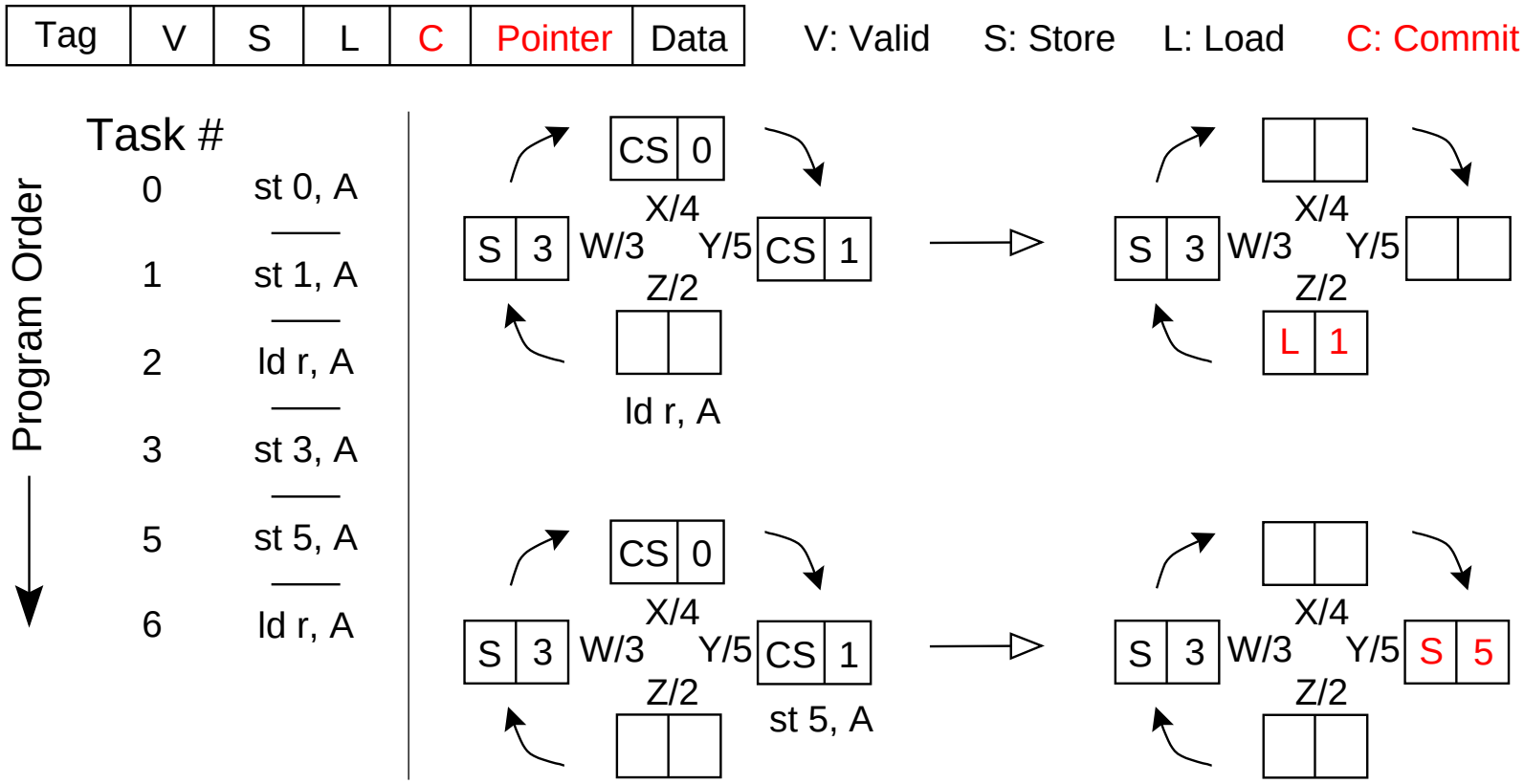
- Architectural bit: to retain architectural versions after a squash

AC LINES ARE CACHE HITS ON LOADS

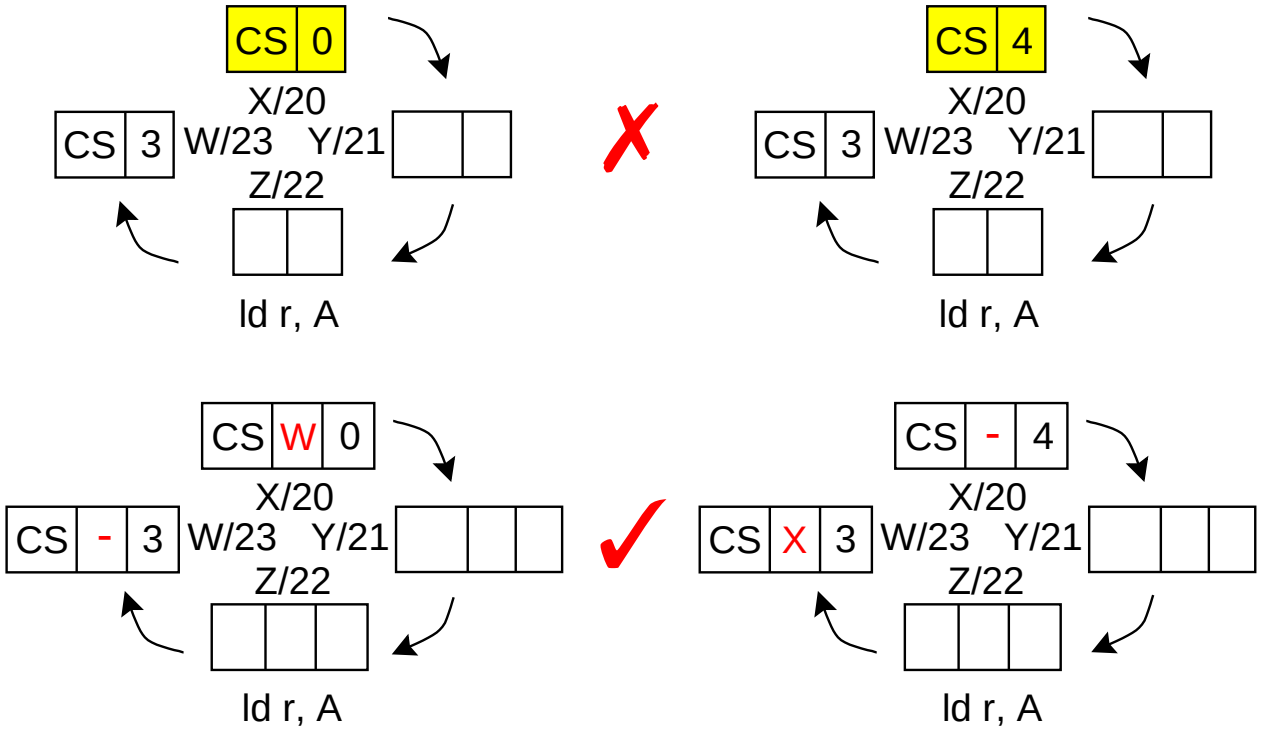
Base SVC Design: Examples



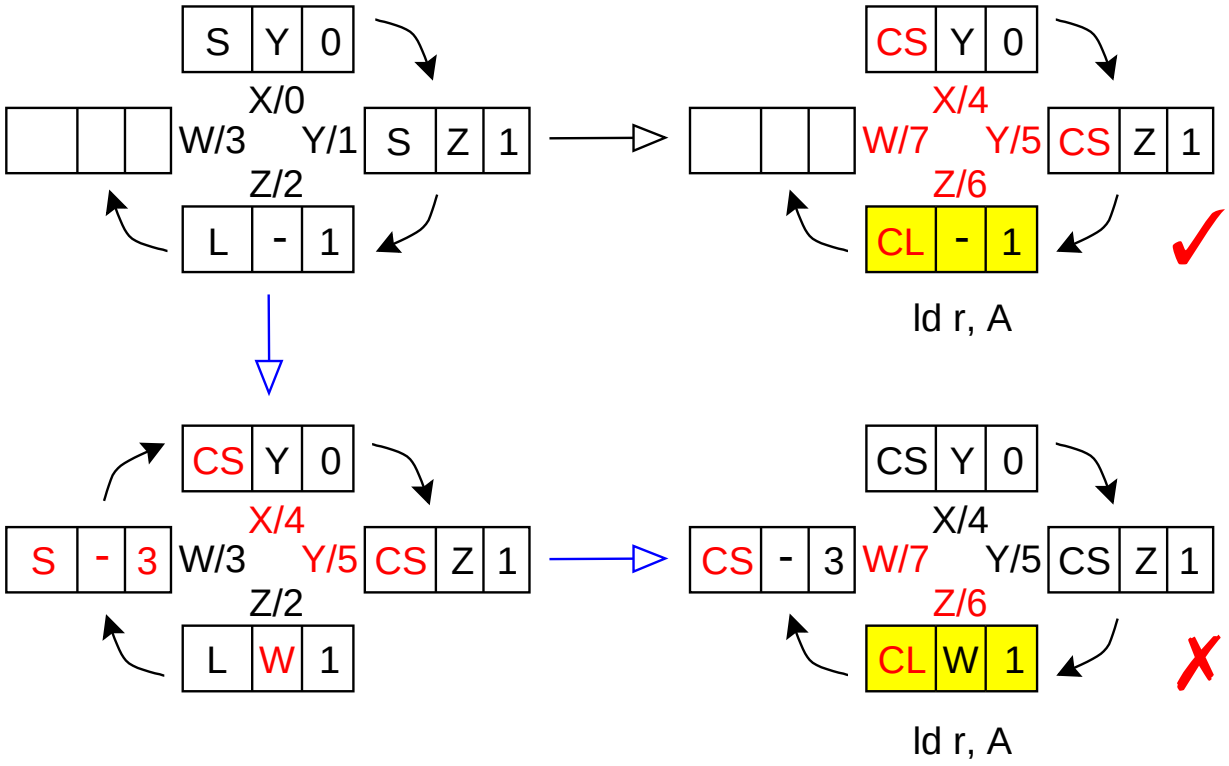
Adv. SVC Design I: Efficient Commits



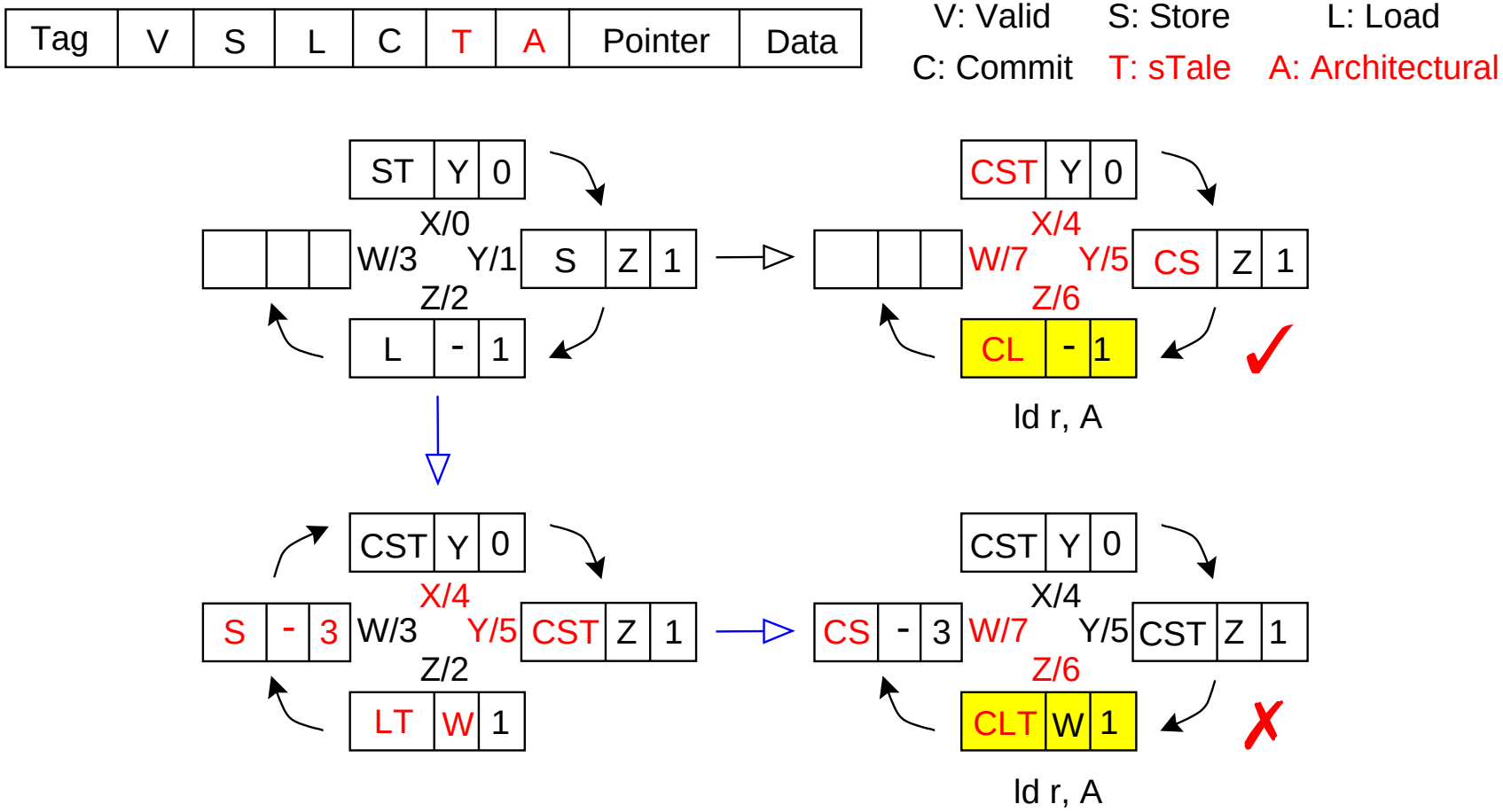
Multiple Committed Versions



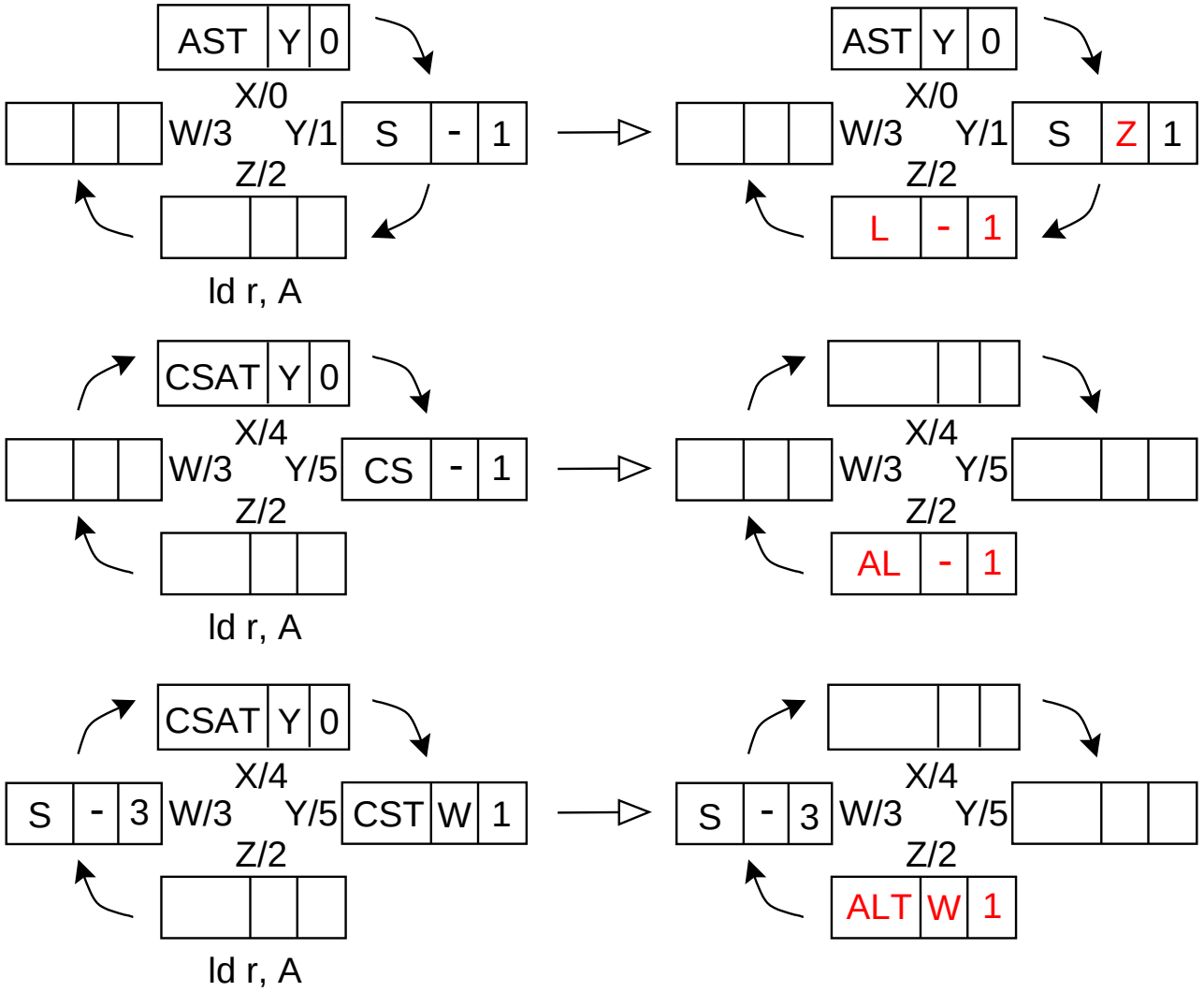
Adv. SVC Design II: Stale Copies



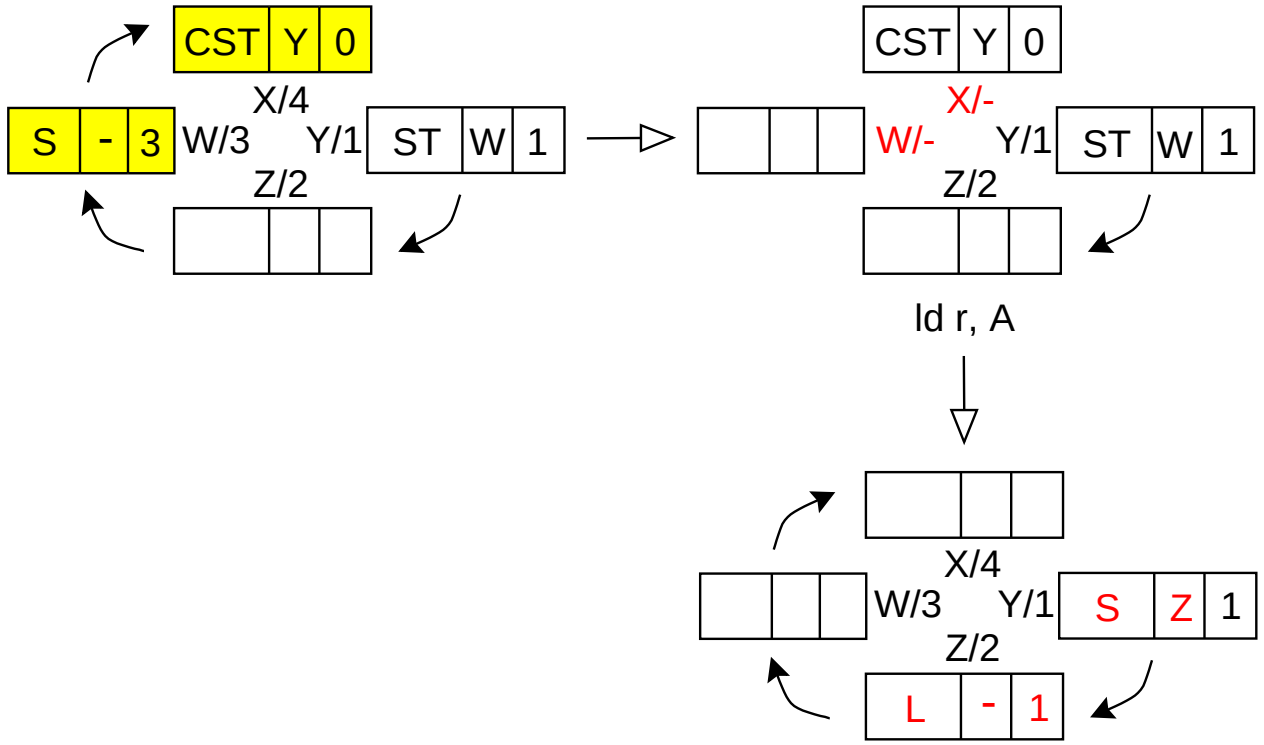
Adv. SVC Design II: sTale bit



Adv. SVC Design II: Architectural Copies



Adv. SVC Designs: Task Squashes



SVC Design: Realistic Linesize

False sharing

- Unnecessary invalidates similar to that for parallel programs
- Worse, these invalidates lead to **false squashes**

Versioning Block

- Similar to Sub-blocking (Transfer and Address Blocks)
- Definition: Granularity at which speculative versioning is provided
- Replicate only the *L* and *S* bits for every versioning block