# Lecture 20: Speculation

Papers:

- Is SC+ILP=RC?, Purdue, ISCA'99
- Coherence Decoupling: Making Use of Incoherence, Wisconsin, ASPLOS'04
- Selective, Accurate, and Timely Self-Invalidation using Last Touch Prediction, Purdue, ISCA'00

# Consistency: Summary

- To preserve sequential consistency:
  - hardware must preserve program order for all memory operations (including waiting for acks)
  - writes to a location must be serialized
  - the value of a write cannot be read unless all have seen the write (it is ok if writes to different locations are not seen in the same order as long as conflicting reads do not happen)

# Relaxations

| Relaxation | W → R Order | W → W Order | R →RW Order | Rd others' Wr early | Rd own Wr early |
|---|---|---|---|---|---|
| IBM 370 | X | | | | |
| TSO | X | | | | X |
| PC | X | | | X | X |

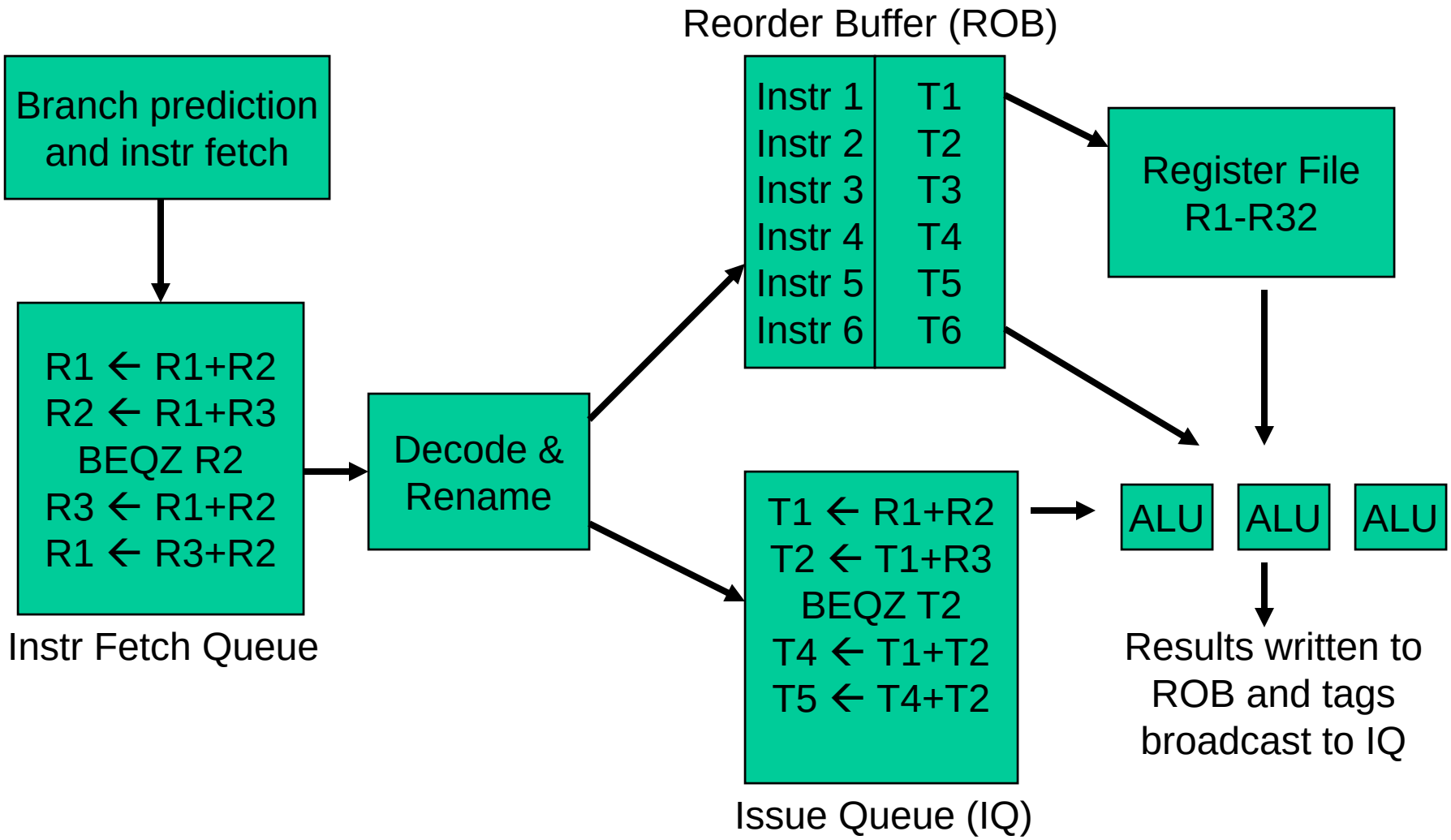
- IBM 370: a read can complete before an earlier write to a different address, but a read cannot return the value of a write unless all processors have seen the write
- SPARC V8 Total Store Ordering (TSO): a read can complete before an earlier write to a different address, but a read cannot return the value of a write by another processor unless all processors have seen the write (it returns the value of own write before others see it)
- Processor Consistency (PC): a read can complete before an earlier write (by any processor to any memory location) has been made visible to all

# Release Consistency

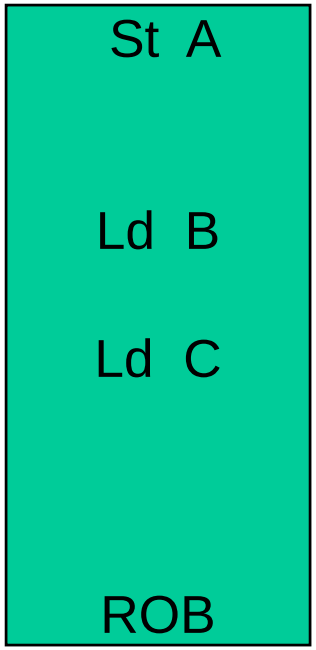
More distinctions among memory operations:

- Instructions are classified as non-special and special and special instructions are further classified as acquires and releases
- All instructions after an acquire can begin only after the acquire has completed
- A release can begin only after all previous instructions have completed

# An Out-of-Order Processor Implementation

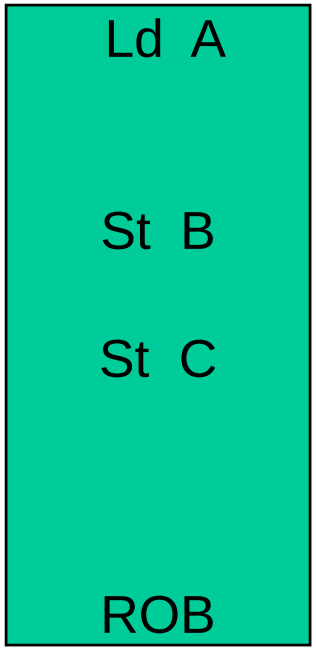


# SC Execution




- The world hasn't yet seen A
- Ld-B should really execute only when it reaches the head of the ROB
- Can execute early, but when it reaches the head of the ROB, it must make sure that the read value is still valid
- A table can keep track of speculatively read values and flag a violation if it detects a write (similar to LL-SC)
- An overflow will conservatively flag a violation
- No problem if Ld-C executes sooner than Ld-B

# SC Execution




- Ld-A hasn't yet executed
- St-B can acquire permissions early – if the permissions are lost before the store reaches the ROB head, permissions are re-acquired
- St-B can also write the result into the cache/memory early, but the old value must be preserved somewhere; if someone else asks for the data, a violation is flagged

# Three Implementations

- SC:
  - Loads can execute speculatively, but must check during commit
  - Relatively small ROB size can hold up commit
  - Stores can fetch permissions speculatively
- RC:
  - Software-assigned constraints
  - Orderings allowed among non-special ops
  - Loads can execute speculatively across fences (as in SC)
- SC++:
  - Stores are allowed to write to cache early (old values are maintained in a log)
  - Instructions are copied out of ROB into a Speculative History Queue SHiQ (effectively, provides a much larger ROB)
  - (Long latency loads can tie up the ROB, but long-latency stores can not)
  - To check for conflicts, a Block Look-up Table (BLT) maintains the relevant block addresses for instructions in the SHiQ

# Results

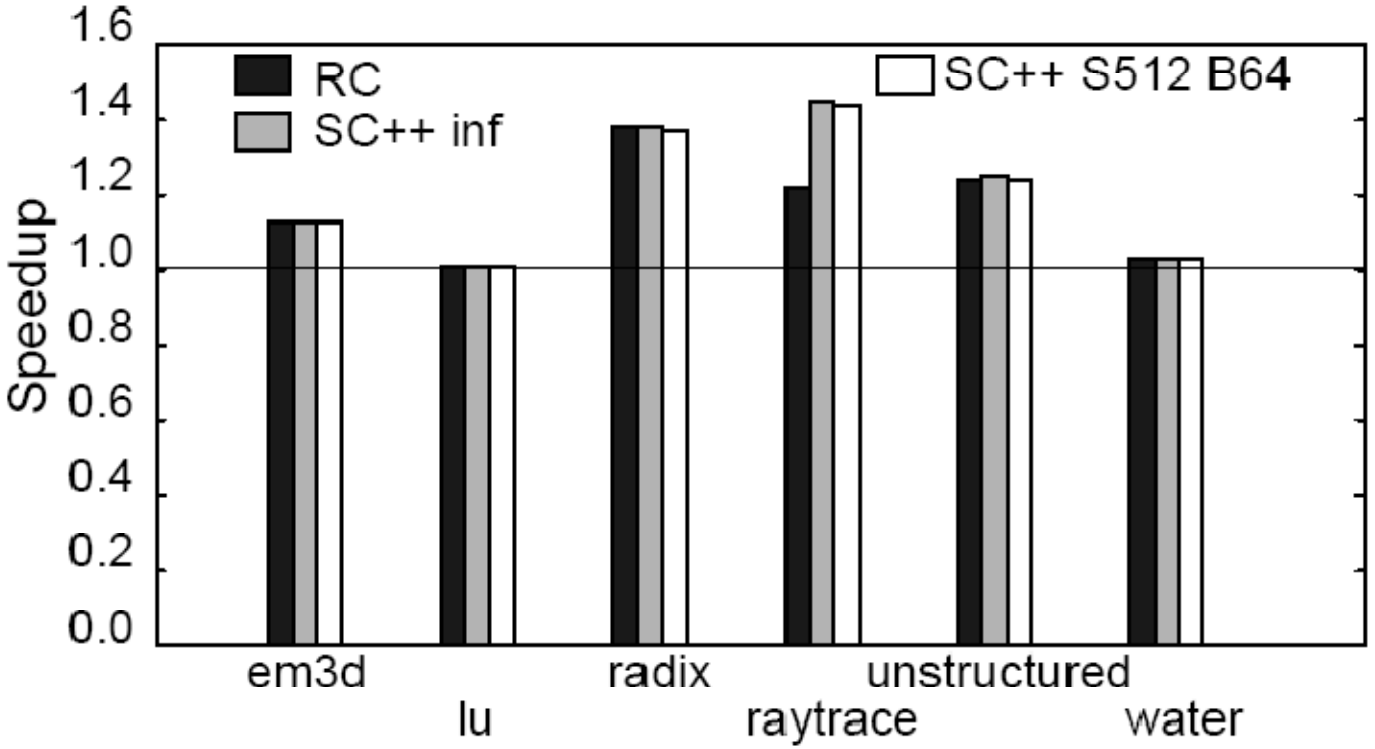



**FIGURE 3: Comparison of SC, RC, and SC++.**

# Violations

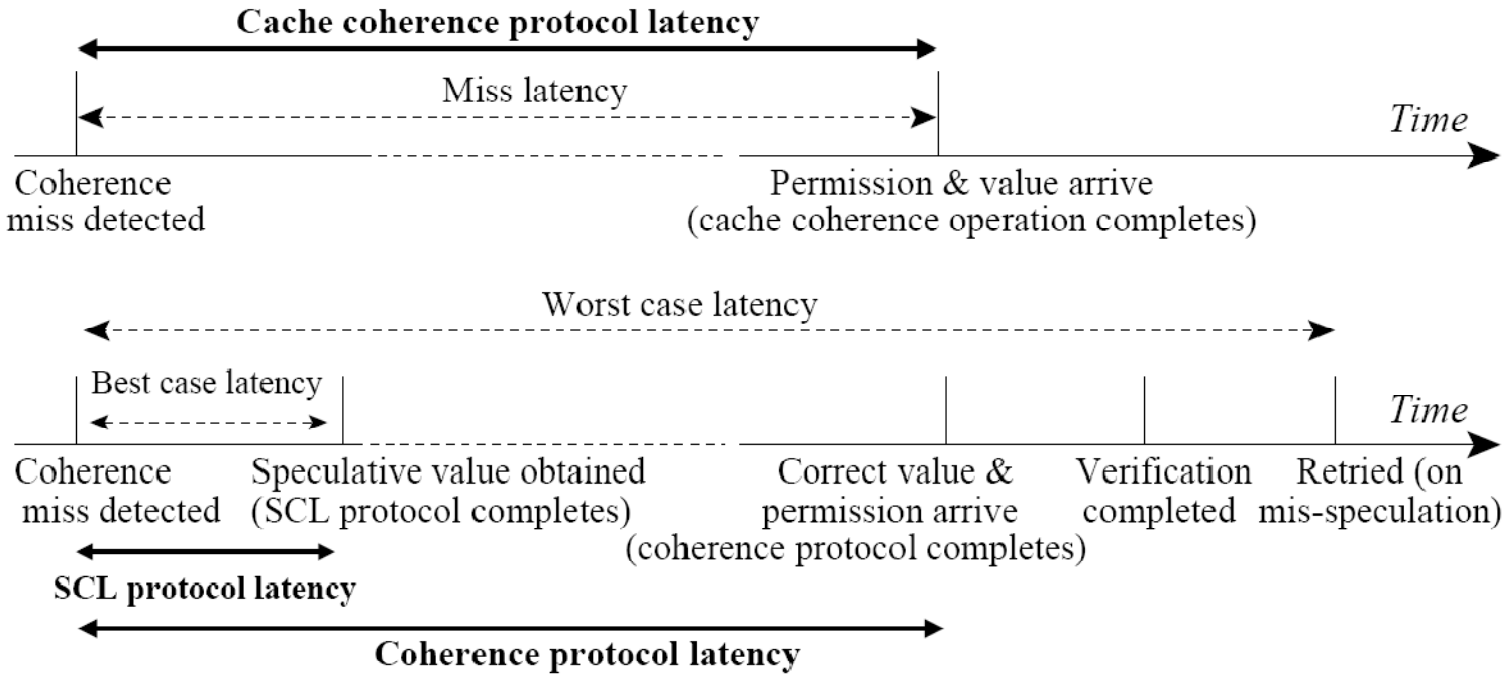
- Violations are caused by
  - true sharing: two threads that r/w the same word
  - false sharing: two threads that r/w the same block
  - conflict/capacity misses in the cache: if a block is evicted, a violation is conservatively flagged

- Violations are caused by
  - a speculative load that gets written to by another node
  - a speculative load and the block is evicted from cache
  - a speculative store that is accessed by another node before it commits (need not cause rollback; just have to re-acquire permissions when ready to commit)

# Coherence Decoupling

- In addition to the correct cache coherence protocol, introduce a Speculative Cache Lookup (SCL) protocol
- In essence, value prediction for the coherence protocol
- Mechanisms incorporated for correct SC execution can be leveraged to handle violations
- SC++ allows us to overlap memory instructions; but for a load, cannot proceed with execution until data+perms arrive
- Coherence decoupling attempts to pass the results of the load to dependents before arrival of permissions/data

# Coherence Decoupling




Why it works:

- False sharing: the line is invalidated, but the relevant word is unchanged
- Speculative updates: updated values are selectively pushed to improve the chances of finding correct data

# Read and Update Components

| SCL Component | Policy | Description |
|---|---|---|
| Read | CD | Use the locally cached incoherent value for every L2 miss |
| Read | CD-F | Add a PC-indexed confidence predictor to filter speculations |
| Update | CD-IA | Use invalidation piggyback to update all invalid blocks |
| Update | CD-C | Use invalidation piggyback if the value is special (compressed) |
| Update | CD-N | Update all sharers after N writes to a block (N=5 in Section 4) |
| Update | CD-W | (Ideal): Update on every write if any sharers exist |

**Table 1: Coherence Decoupling Protocol Components**

- Filter: branch-pred-like confidence mechanism to avoid mis-speculations
- IA: send data with invalidation message (higher bw cost)
- C: send data only if it is {-1, 0, 1}

# Potential and Accuracy

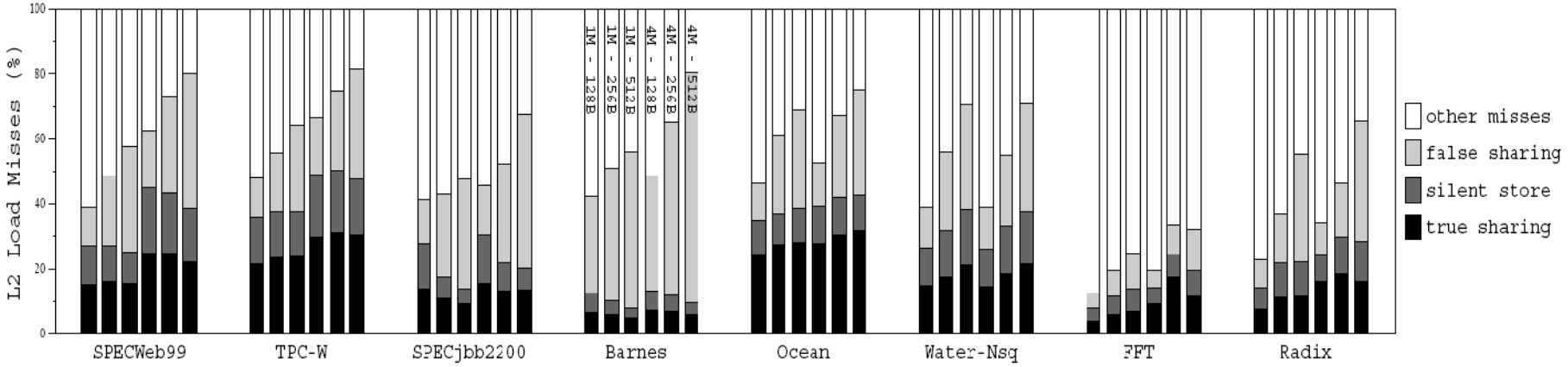



Figure 3: L2 Load Miss Breakdowns.

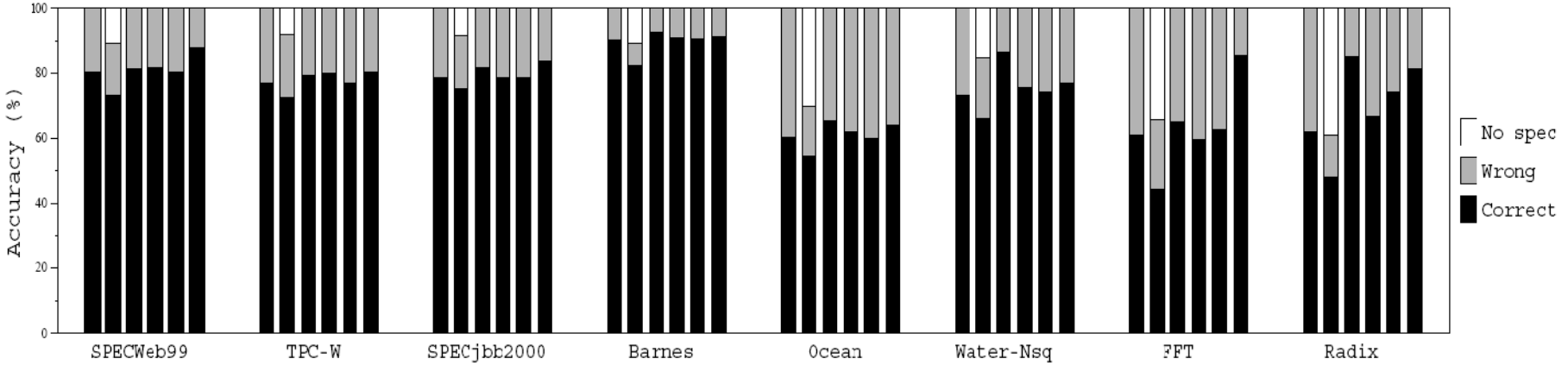



Figure 4: Accuracy of Coherence Decoupling (from left to right: CD, CD-F, CD-IA, CD-C, CD-N, CD-W)

# Performance

| Benchmark | CD | CD-F | CD-IA | CD-C | CD-N5 | CD-W | Optimal |
|---|---|---|---|---|---|---|---|
| SPECWeb99 | 13.8% | 11.0% | 13.2% | 13.1% | 14.9% | 18.0% | 34.6% |
| TPC-W | 1.2% | 2.6% | 2.3% | 1.7% | 1.4% | 2.4% | 17.8% |
| SPECjbb2000 | 16.6% | 15.8% | 13.5% | 13.0% | 17.1% | 16.5% | 26.3% |
| Barnes | 0.6% | 0.4% | 0.7% | 0.7% | 0.8% | 0.6% | 1.4% |
| Ocean | 6.9% | 4.7% | 8.2% | 7.4% | 6.0% | 7.5% | 34.5% |
| Water-Nsq | 2.1% | 1.7% | 2.8% | 3.5% | 0.7% | 5.4% | 17.4% |
| FFT | 5.1% | 4.2% | 6.1% | 7.2% | 4.6% | 10.8% | 21.4% |
| Radix | 6.8% | 3.6% | 7.6% | 8.8% | 6.3% | 12.0% | 42.4% |
| Mean | 6.6% | 5.5% | 6.8% | 6.9% | 6.5% | 9.1% | 24.5% |

**Table 3: Speedups for Coherence Decoupling**

| Benchmarks | CD-IA | CD-N5 |
|---|---|---|
| SPECWeb99 | 3.6% | 7.9% |
| TPC-W | 3.9% | 18.5% |
| SPECjbb2000 | 2.5% | 2.5% |
| Barnes | 2.7% | 95.3% |
| Ocean | 3.4% | 6.1% |
| Water-Nsq | 2.0% | 28.1% |
| FFT | 2.8% | 10.5% |
| Radix | 3.2% | 8.2% |

**Table 4: Data Traffic Increase**

# Self Invalidation

- To reduce the cost of invalidating shared blocks, blocks can "prefetch" this effect by invalidating themselves in advance
- Can perform self invalidation when leaving a critical section
- Can perform self invalidation by observing the sequence of instructions that eventually lead to the last touch (truncated addition is used to generate a signature) (may require complex branch predictor-like structures)
- Cache decay predictors have good accuracy in a single-threaded setting… may have similar benefits in multi-threaded settings as well

# Summary

- May need large windows to hide the cost of long memory operations
- Can overlap operations, can speculate on values
- Will need checks to make sure SC is not violated and values are speculated correctly

# Title

- Bullet