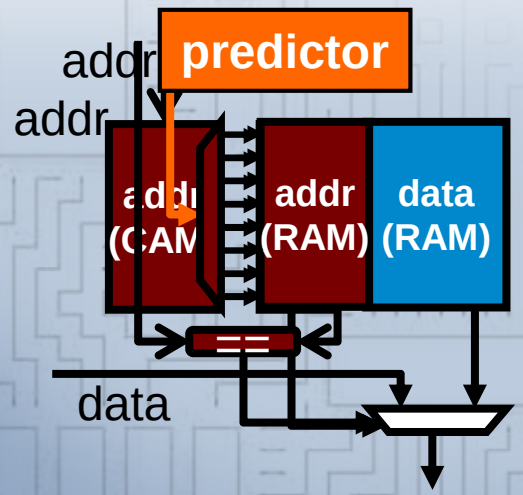
Scalable Store-Load Forwarding via Store Queue Index Prediction

Tingting Sha, Milo M.K. Martin, Amir Roth

University of Pennsylvania

{shatingt, milom, amir}@cis.upenn.edu



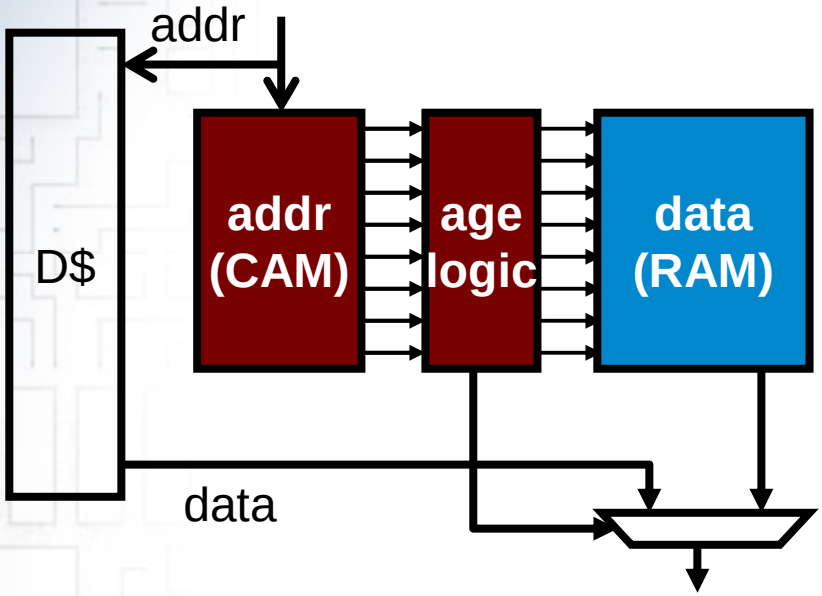
Four Aspects of OoO Load/Store Execution

- The four aspects…
  1. **Commit** *stores in order*
  2. **Detect** *memory-ordering violations*
  3. **Reduce** *memory-ordering violations*
  4. **Forward** *from stores to loads*

- … and state-of-the-art implementations
  1. Age ordered store queue (SQ)
  2. Age ordered load queue (LQ) + associative search
     - Proposed: in-order load re-execution [Cain+'04]
  3. Dependence prediction [Kessler+'94, Moshovos+'97, Chrysos+'98…]
  4. Associative store queue search

We consider the first three aspects "solved"

The Problem



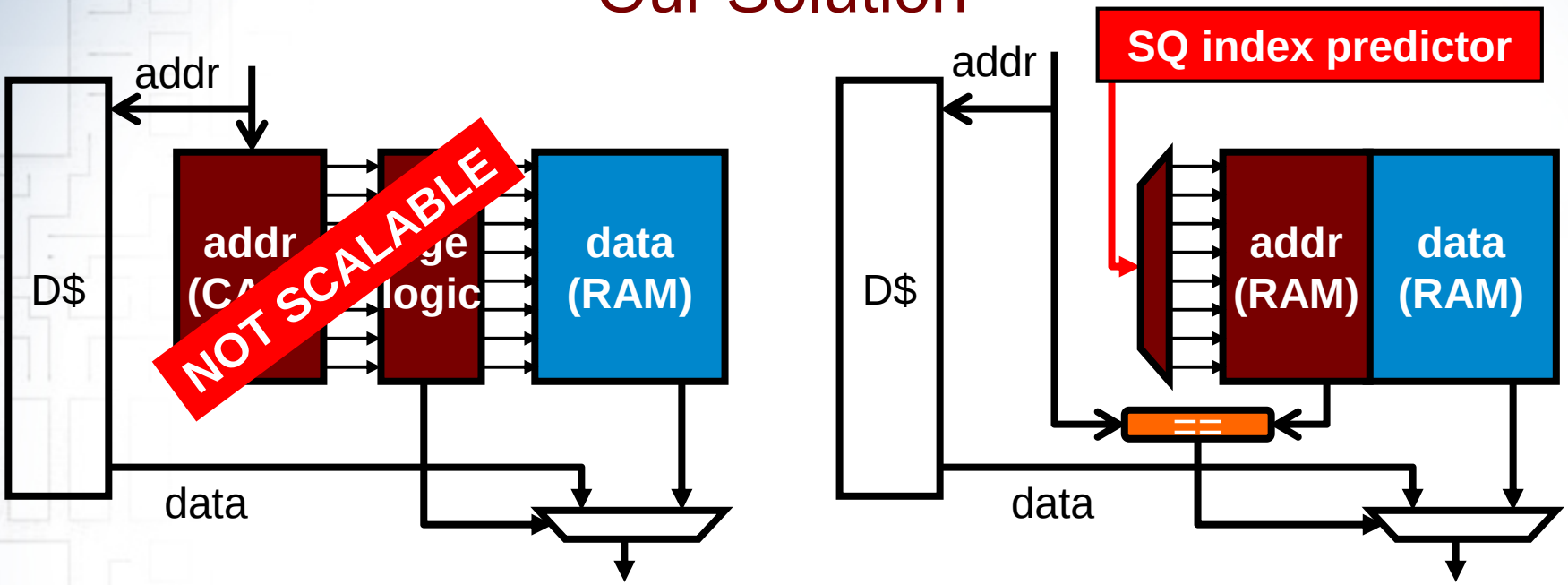
- **Age-ordered associative search is not scalable**
  - **Simple associative search**: CAM is slow, difficult to pipeline
    - But can be partitioned (to reduce wire delay)
  - **Age-ordered associative search**: CAM and age logic are slow
    - And age logic makes partitioning difficult
  - Load queue → load re-execution [Cain+'04]
  - Store queue → ?

Possible Solutions

- Engineer around associative search
  - Put your best designer on the store queue
  - Longer access latency than D$ → nasty scheduler, replays…
    - Put your best designer on the scheduler
- Proposed: reduce associative search
  - Reduce bandwidth
    - Bloom-filtered SQ [Sethumadhavan+'03]
  - Reduce number of stores searched
    - Pipelined/chained SQ [Park+'03]
    - Hierarchical/filtered SQ [Srinivasan+'04, Ghandi+'05, Torres+'05]
    - Decomposed SQ [Roth'05; Baugh+'04]

Why not just eliminate associative search?

Our Solution



- **Replace associative search with indexed access**
  - Replace address CAM + age logic with address RAM
  - **Predict one SQ index per load**
    - Predictor (e.g., Store Sets) is not on load critical path
  - Keep **address match** → allow false positives → boost accuracy

Verification

- Speculative indexed access requires verification
  - In-order load re-execution prior to commit [Cain+'04]
  - **Store Vulnerability Window (SVW) re-execution filter** [Roth'05]
    - + On average 3% of loads re-execute → almost free
    - + Works unmodified for speculative indexed SQ (address check)

Re-execution + Indexed store queue = …

CAM-free load/store unit

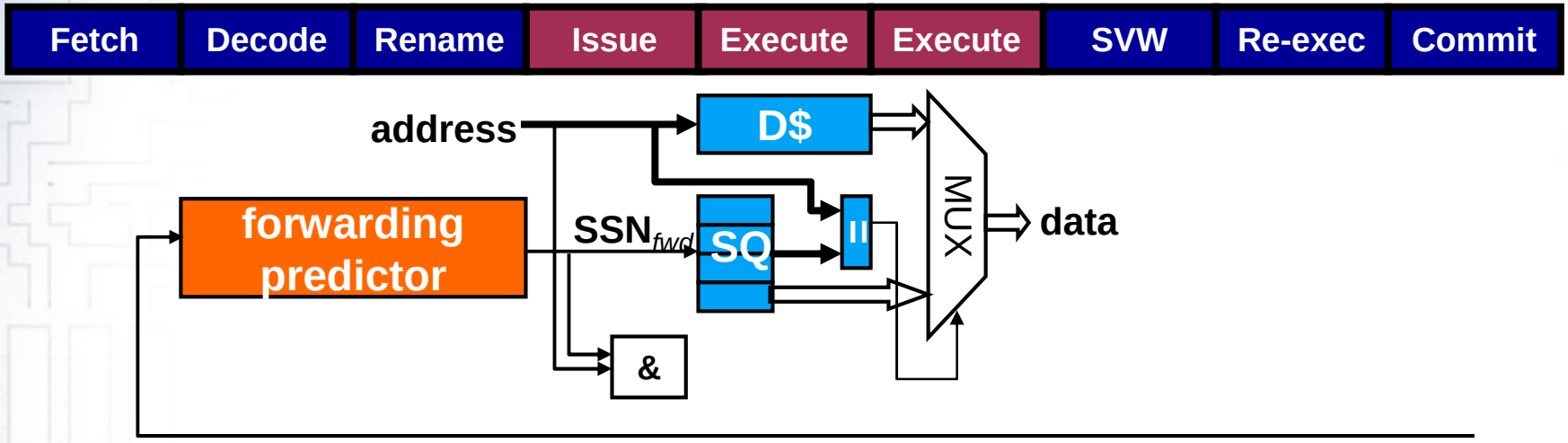
Outline

- ✓Introduction and background
- Forwarding index prediction
  - Mechanism
  - Evaluation
- Delay index prediction
  - Mechanism
  - Evaluation
- Conclusion

SSNs: A Naming System for Dynamic Stores

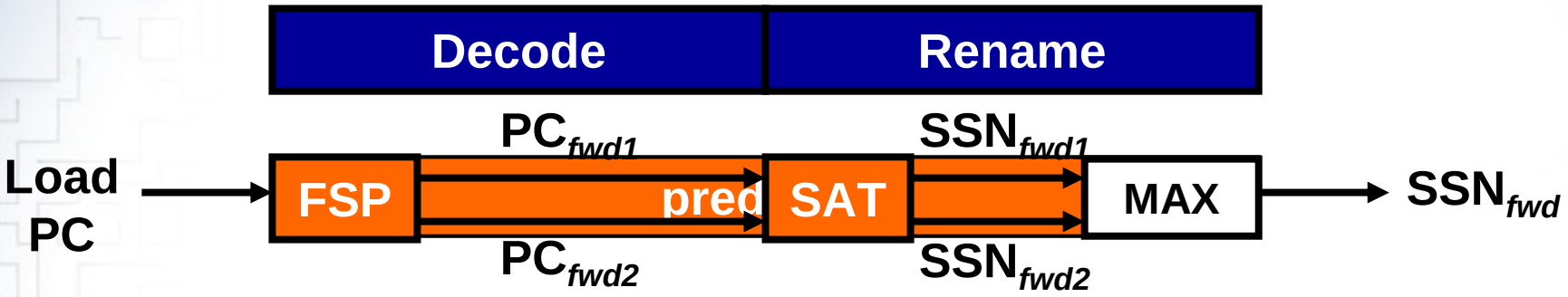
- **SSNs (Store Sequence Numbers)**
  - Required for SVW and more convenient than store queue indices
    - + Can name committed stores
    - + Fewer wrap-around issues
  - Monotonically increasing
  - **SSN_{commit}**: youngest committed store
  - **$SSN_{dispatch}$**: youngest dispatched store (**SSN_{commit}** + SQ.NUM)
  - From SSN to store queue index?
    - If st.SSN > SSN_{commit}, st.INDEX = (st.SSN % SQ.SIZE)

Indexed Forwarding Pipeline Actions



- Decode/rename: predict forwarding store (SSN_{fwd})
- Issue: wait for load address, **SSN_{fwd}** to execute
  - Unify: SSN_{fwd} used for both forwarding and scheduling
- Execute: **index SQ**, forward if address matches
- SVW/re-execute: verify forwarding
- Commit: train predictor

Forwarding Index Predictor



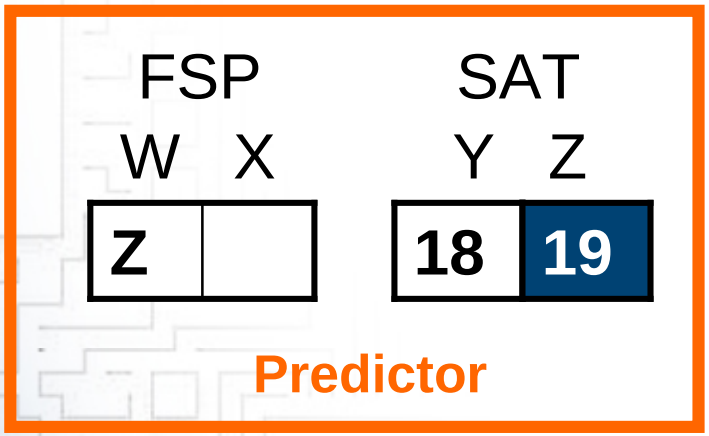
- **Forwarding Store Predictor (FSP)**
  - Maps load PC to small set of likely-to-forward store PCs
  - (Load) PC-indexed, set-associative, entry={tag, (2) store PCs}
- **Store Alias Table (SAT)**
  - Maps store PC to its most recent store instance (SSN)
  - (Store) PC-indexed, direct-mapped, entry={SSN}
- **SSN_{fwd}**: largest SSN (youngest in-flight store)
- Design inspired by Store Sets [Chrysos+'98]
  - Used for load scheduling … **and forwarding**

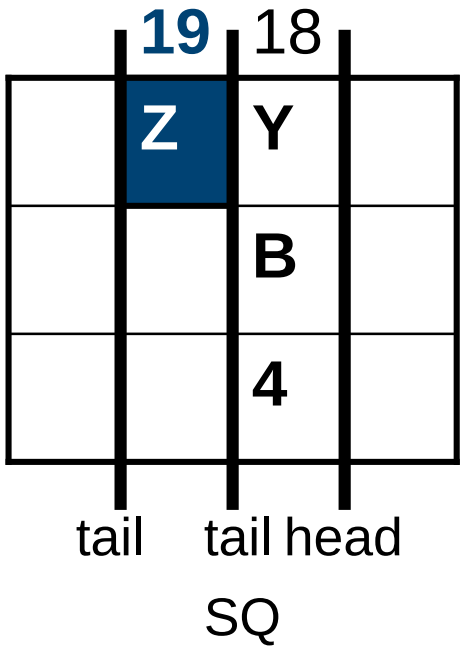
Working Example: Forwarding (Store Part)

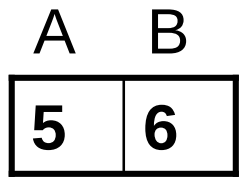
PC Z: ***store 8, A***

PC W: ***ld A***

PC Z: store 8, A







Data cache

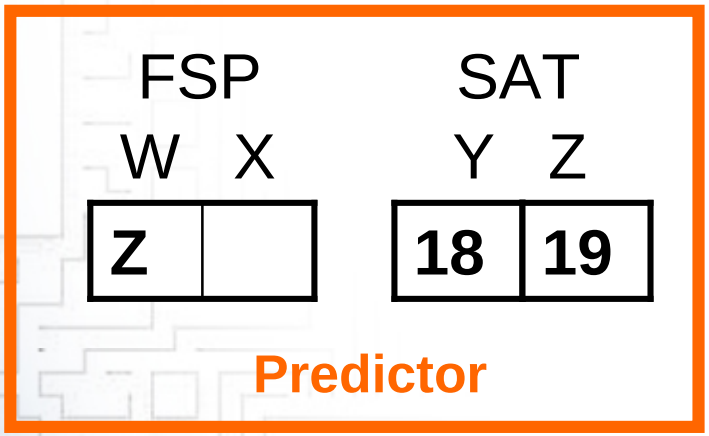
SSN_{commit} = 17

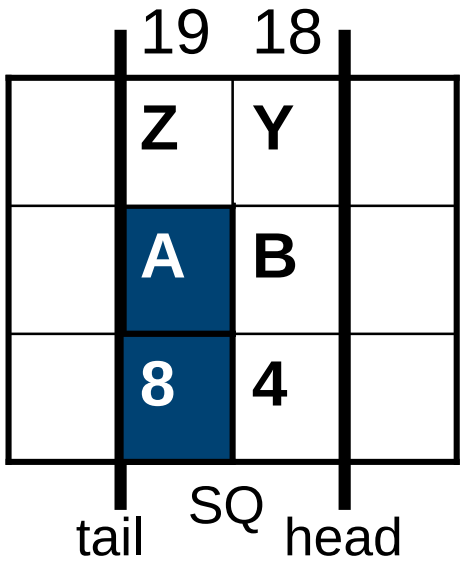
Working Example: Forwarding (Store Part)

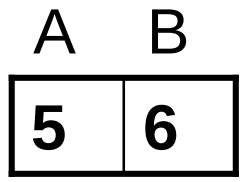
PC Z: ***store 8, A***

PC W: ***ld A***

PC Z: store 8, A







Data cache

SSN_{commit} = 17

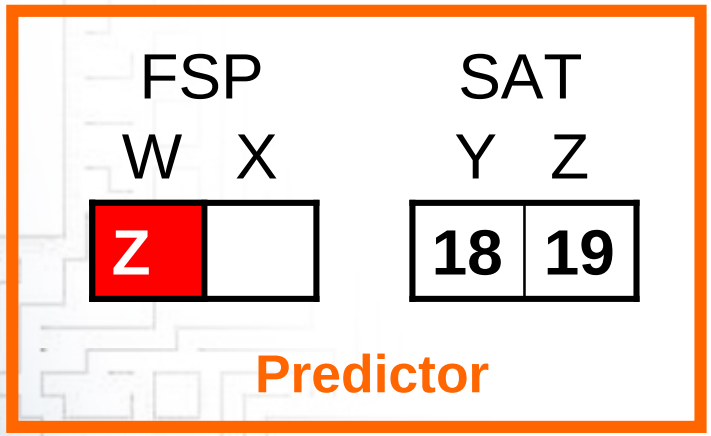
Working Example: Forwarding (Load Part)

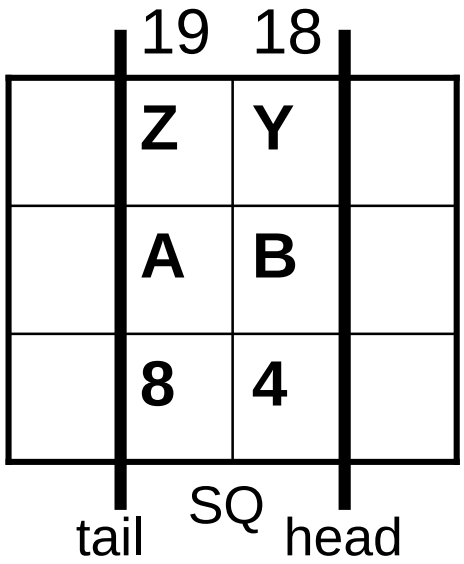
PC Z: ***store 8, A***

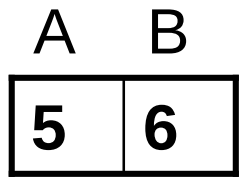
PC W: ***ld A***

PC_{fwd} = ?

PC W: ***ld A***







SSN_{commit} = 17

Working Example: Forwarding (Load Part)

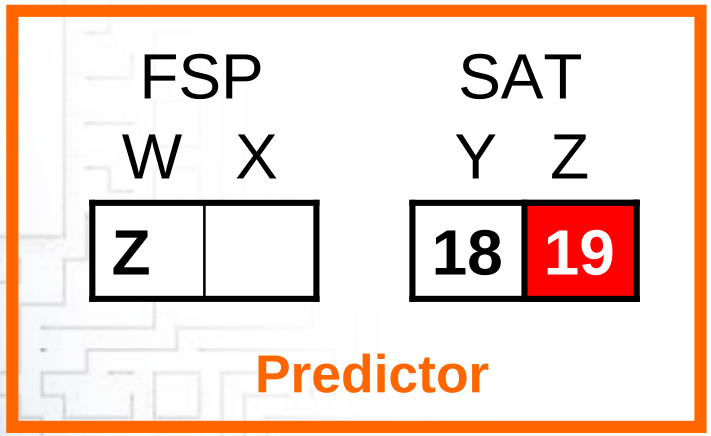
SSN_{fwd} = ?

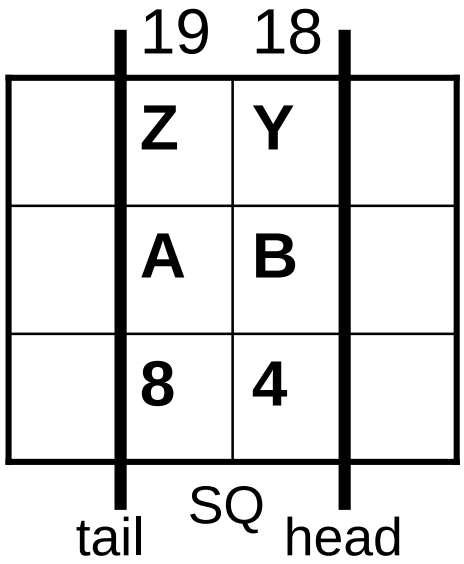
PC_{fwd} = Z

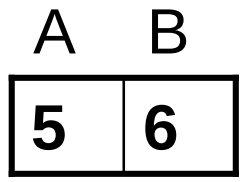
PC W: *ld A*

PC Z: *store 8, A*

PC W: *ld A*







Data cache

SSN_{commit} = 17

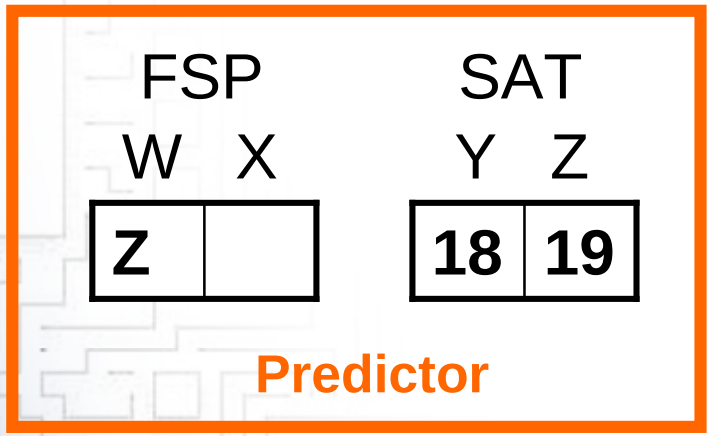
Working Example: Forwarding (Load Part)

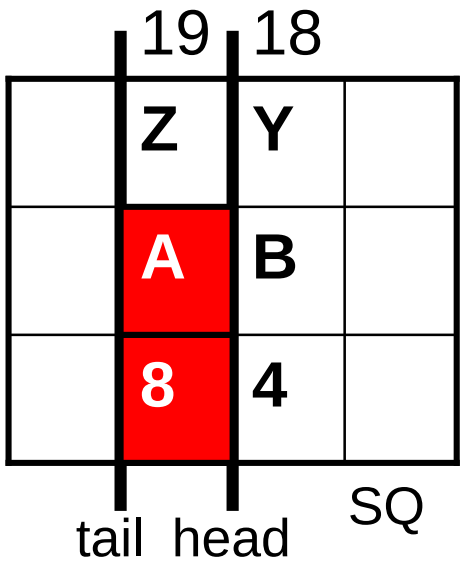
PC Z: ***store 8, A***

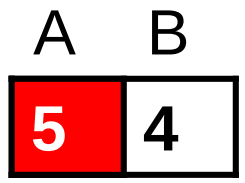
PC W: ***ld A***

SSN_{fwd} = 19

PC W: *ld A*







Data cache

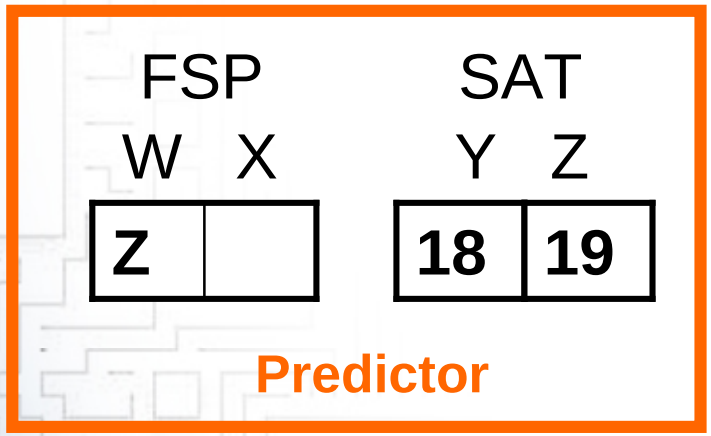
SSN_{commit} = 18

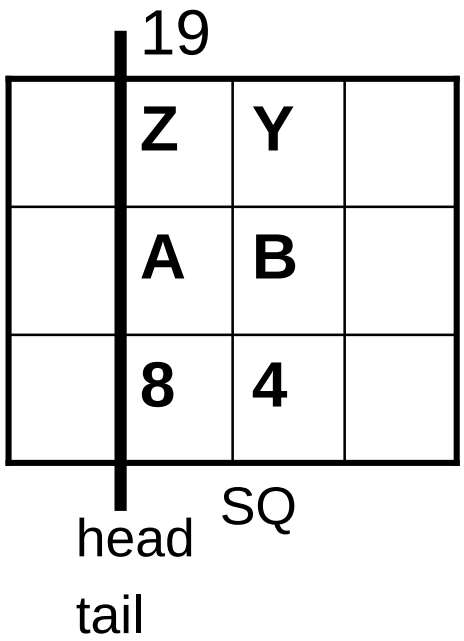
Working Example: Verification (Store Part)

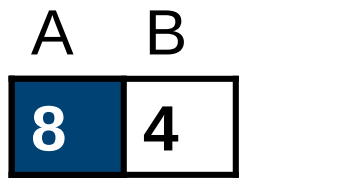
PC Z: ***store 8, A***

PC W: ***ld A, 8***

PC Z: store 8, A







SSN_{commit} = 19

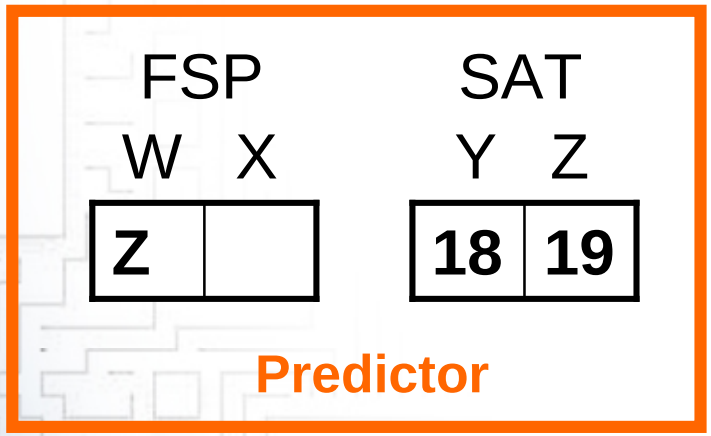
Working Example: Verification (Load Part)

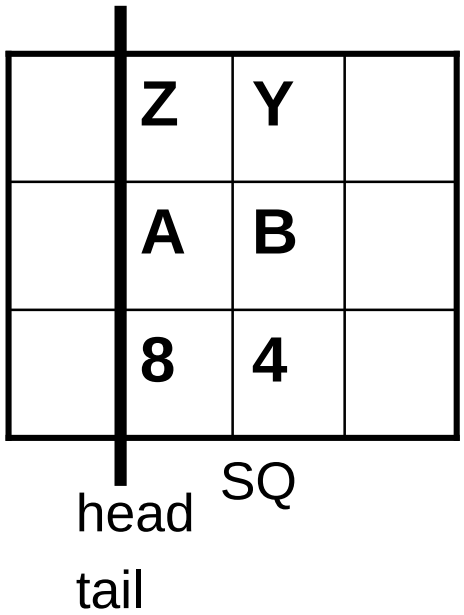
PC Z: ***store 8, A***

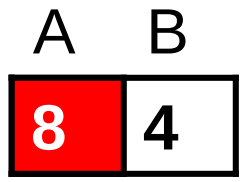
PC W: ***ld A***, **8**

Successful Forwarding

PC W: *ld A*







Data cache

SSN_{commit} = 19

Forwarding Index Predictor Training

- Forwarding Store Predictor (FSP)
  - **Train on every load at commit**
  - Address-indexed tables track PCs, SSNs of last committed stores

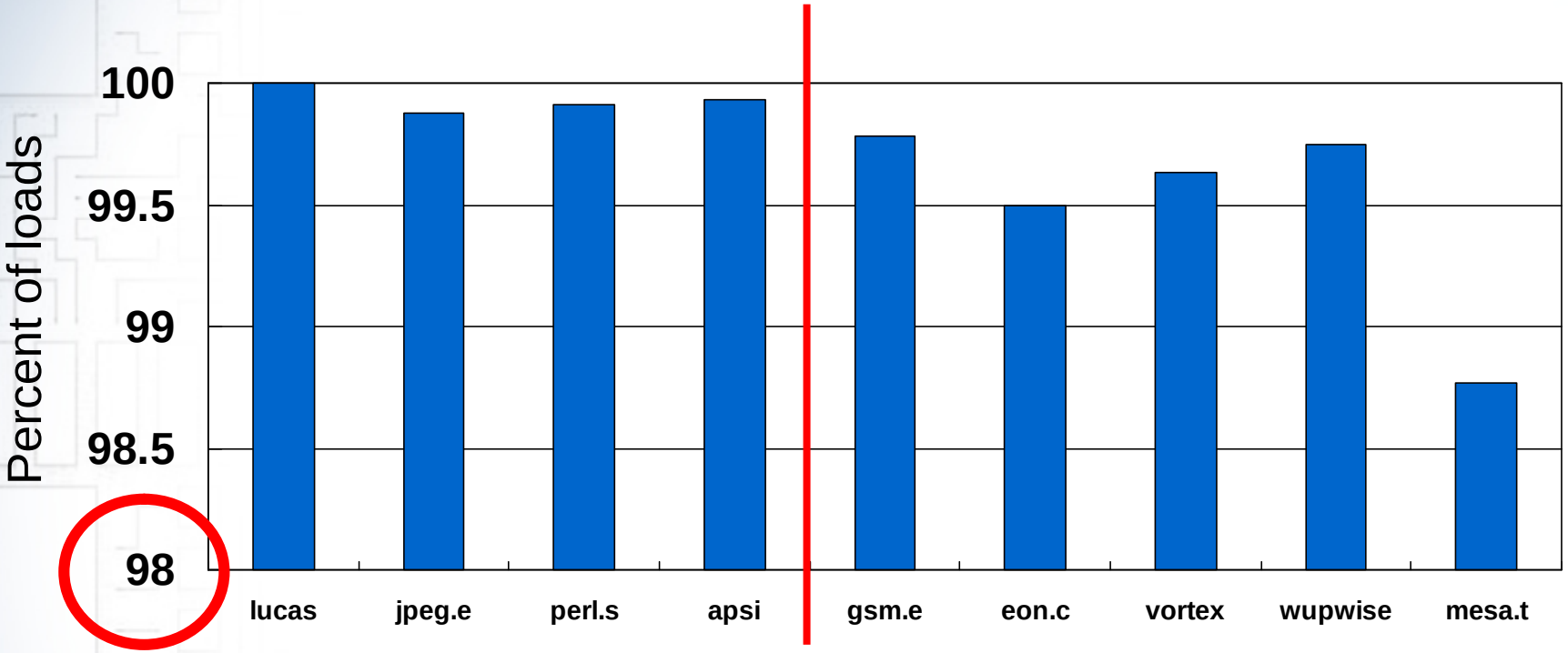
```
if (SSN_fwd == committed_store_SSN[load.addr])
   correct -> re-inforce
else
   if (load.PC_fwd != committed_store_PC[load.addr])
      wrong -> learn new load-store pair
   else
      wrong -> unlearn existing store-load pair (later)
```

- Store Alias Table (SAT)
  - Not a “predictor”, analogous to RAT (register alias table)

So Far, How Well Are We Doing?

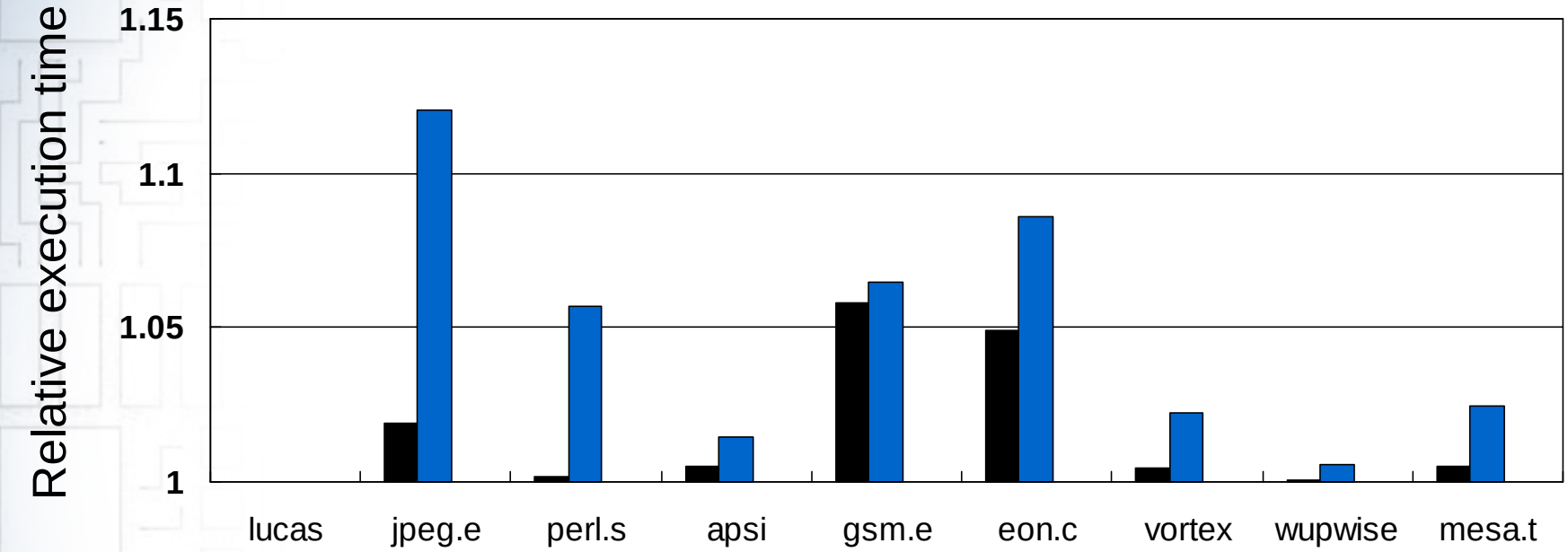
- What we care about
  - Forwarding prediction accuracy
  - Performance vs. associative SQ
- Basic setup
  - Simulator: simplescalar Alpha
    - 8-way superscalar, 19 stage pipe, 512-entry ROB
  - Benchmarks: SPEC2000, Mediabench (only show 9 benchmarks)
- Important Parameters
  - **64KB D$: 3 cycles**
  - **64-entry SQ: associative → 3 & 5 cycles, indexed → 3 cycles**
    - CACTI 3.2: 90nm, 1.1V, 3GHz
  - **FSP: 4K-entry, 2-way**
    - Bigger than Store Sets: all dependences, not just violations
    - Probably OK: PC-indexed, in-order front-end, only 8KB

Forwarding Index Prediction Accuracy



- 99.5%+ accuracy (better than branch prediction), why?
  - ~88% loads don't forward
    - Address check → these ~88% are right "by design"
  - ~12% loads forward
    - Forwarding patterns are stable [Moshovos+, '97]

First Things First: Baseline Performance



- Baseline: 3 cycle associative SQ and perfect scheduling

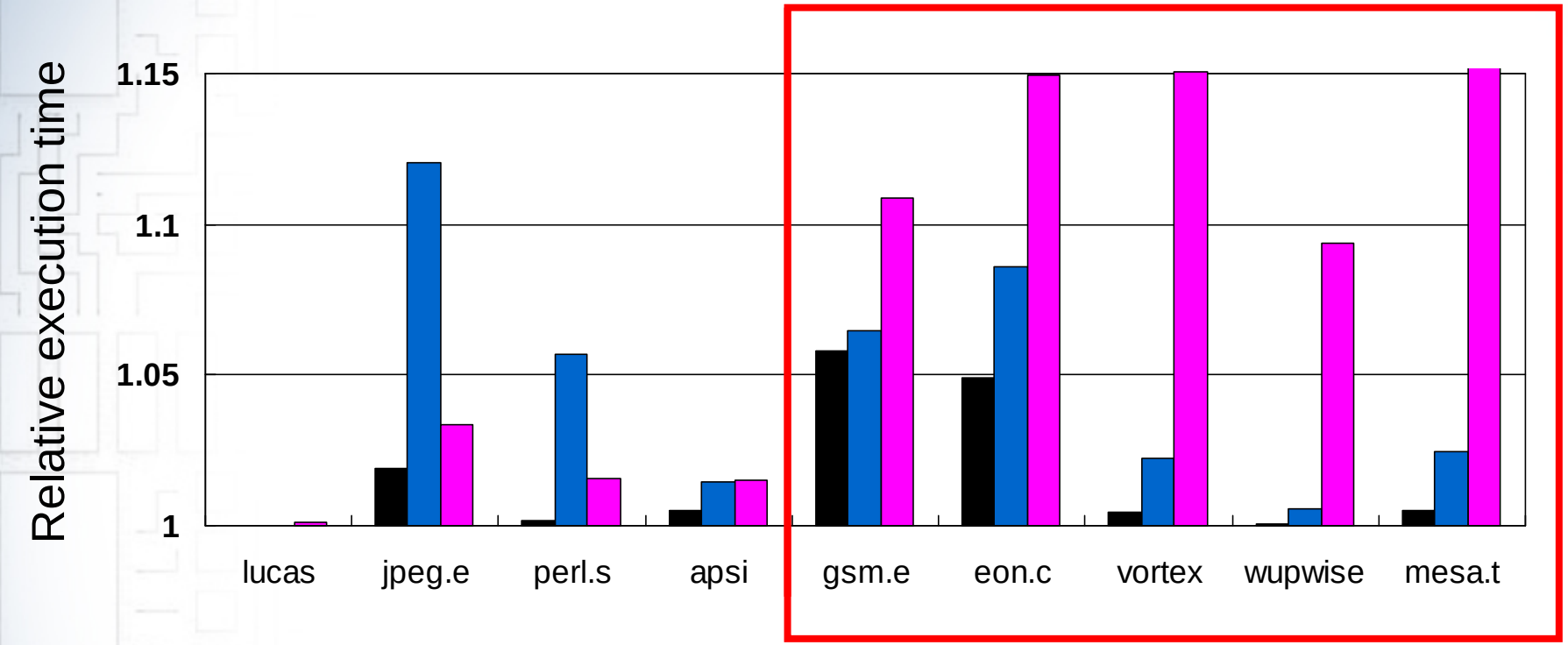
■ + (slightly modified) Store Sets scheduling

- Store sets is basically a “perfect” scheduler

■ + 5 cycle associative SQ (forwarding triggers replays)

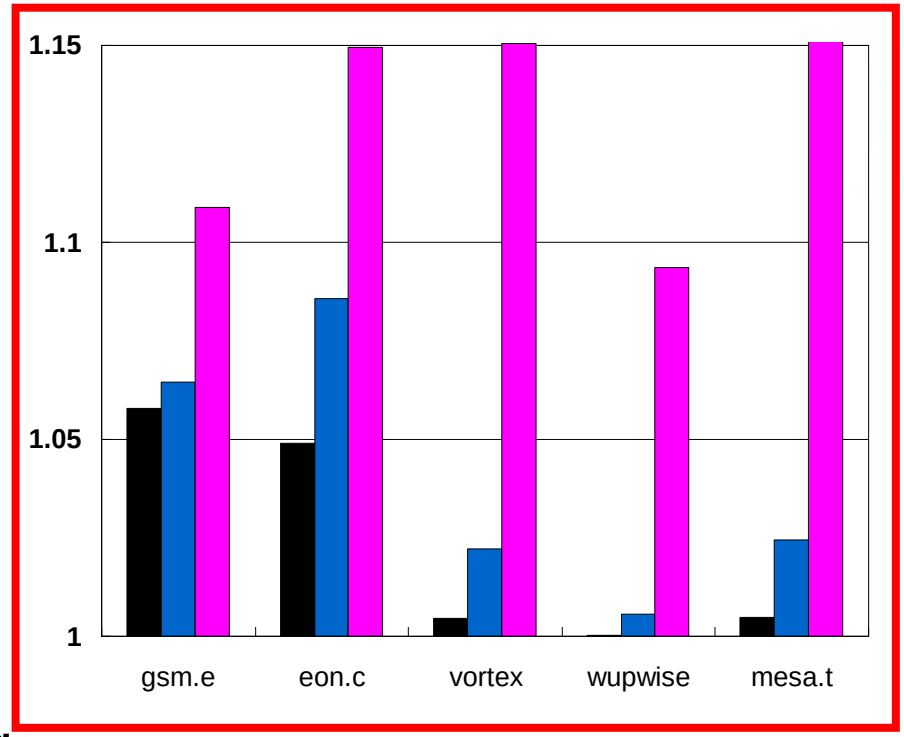
- Latency/replays cause extra 1-10% slowdown

Indexed Forwarding Performance



- 3 cycle indexed SQ (with unified scheduling)
  - \+ Forwarding accuracy high → outperforms 5-cycle associative
  - – Forwarding accuracy low → slowdowns

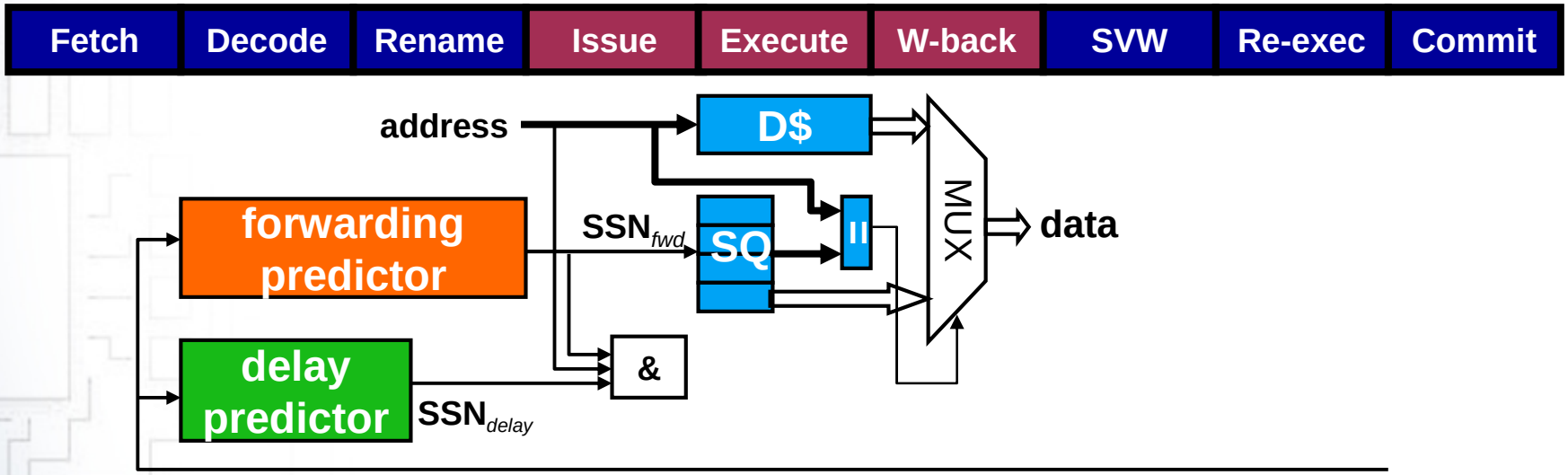
Sources of Low Forwarding Prediction Accuracy



- FSP limitation
  - Our FSP: 2 stores per load
  - eon.c and vortex
- "Not-most-recent" forwarding
  - Example: `for (;;) { X[J] = A * X[J-2]; }`
  - When `J = 5`, load `X[5-2]` depends on store `X[3]`…
  - … but SAT tracks the most recent store instance (`X[4]`)
  - mesa.t

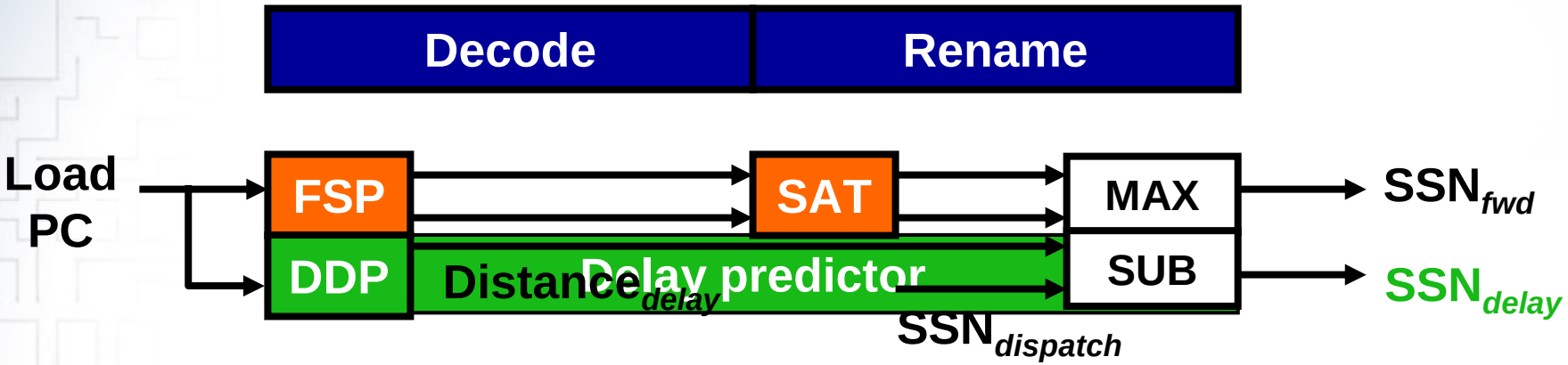
Delay Index Prediction for Difficult Loads

- Uniform solution to low forwarding prediction accuracy
  - Convert flush (really bad) to scheduling delay (less bad)
  - Delay load until uncertain stores **commit,** get value from cache



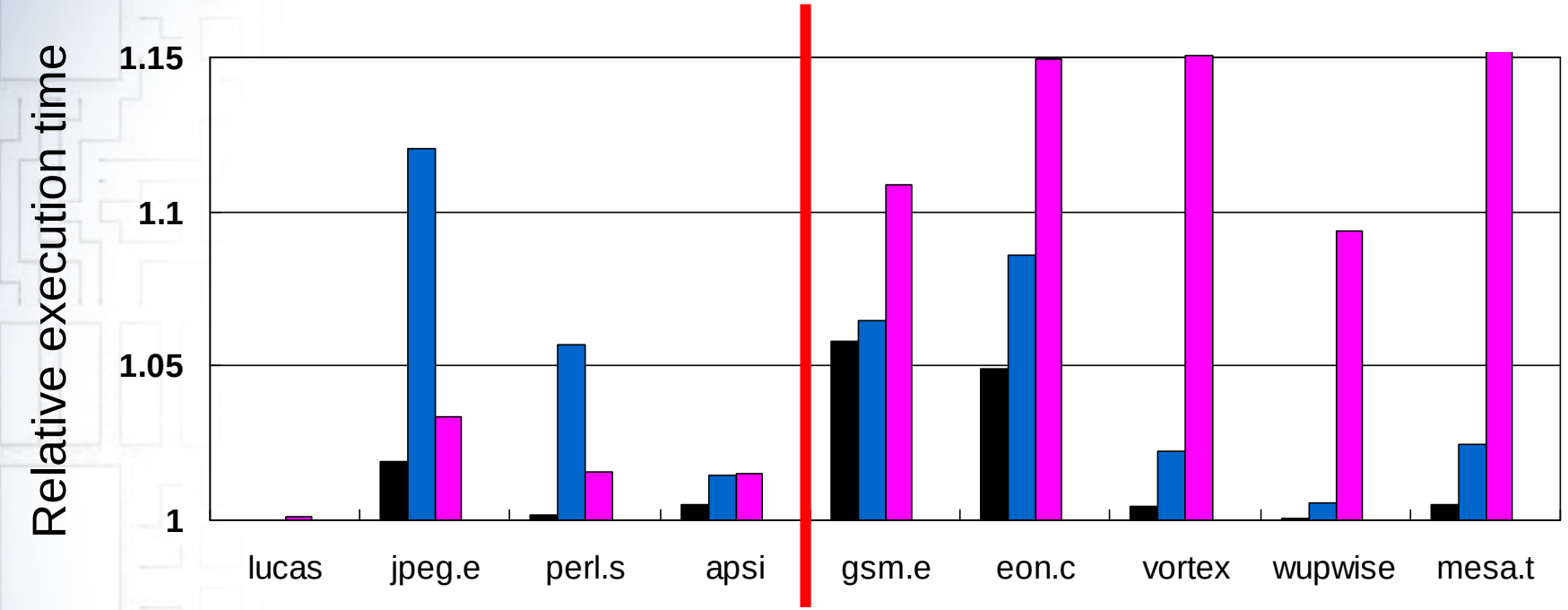
- Decode/rename: predict … + **delay store (SSN$_{delay}$)**
- Issue: wait for … + **SSN$_{delay}$ to commit**

Delay Index Predictor

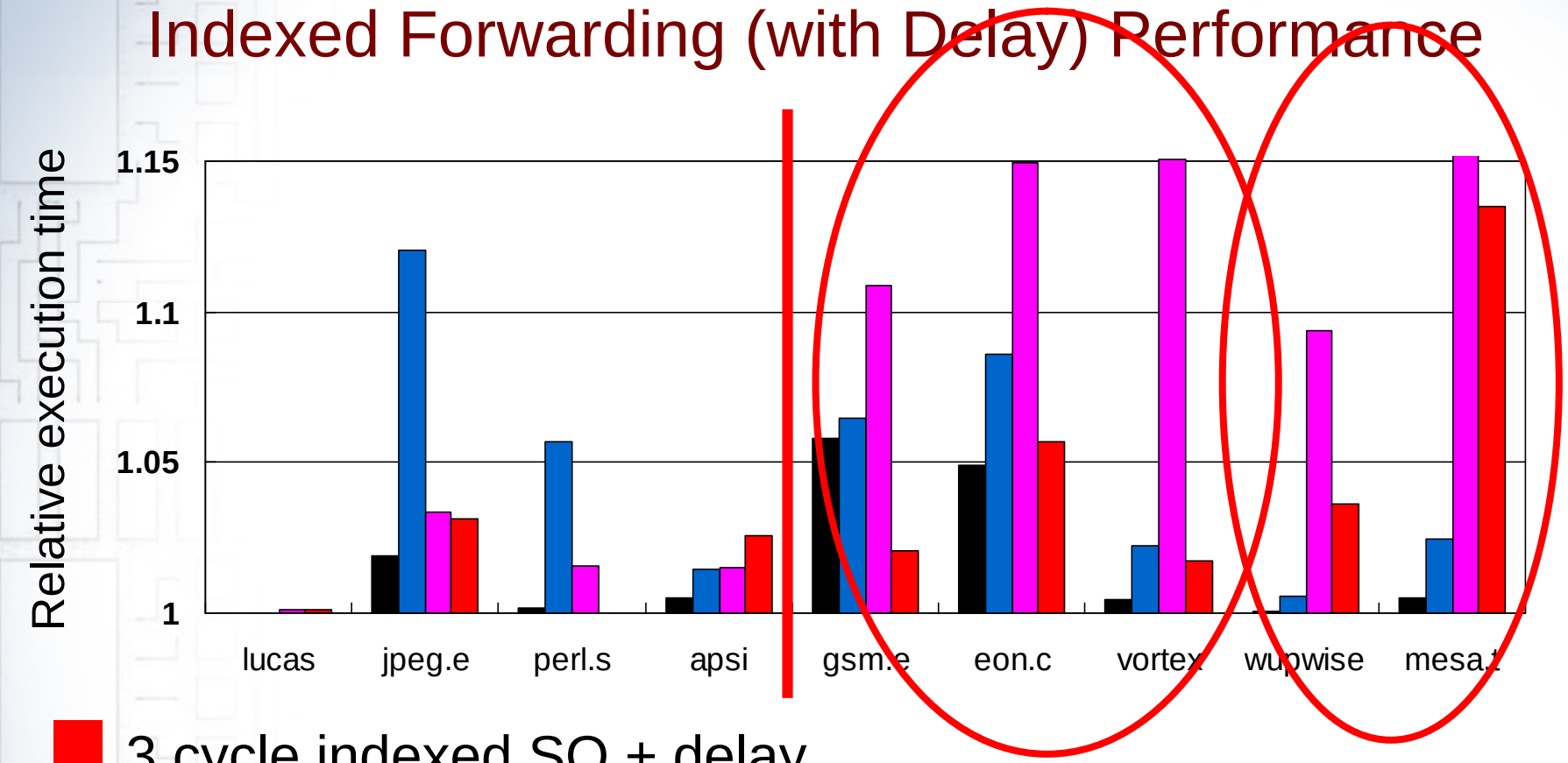


- **Delay Distance Predictor (DDP)**
  - Maps load PC to distance (in stores) to forwarding store
  - PC-indexed, set-associative table, entry = {tag, distance}
- $SSN_{delay} = SSN_{dispatch} - Distance_{delay}$
- Example: not-most-recent forwarding (for **J=5**)
  - $SSN_{dispatch}$ (X[4]) – $Distance_{delay}$ (1) → SSN_{delay} (X[3])
- Inspired by Exclusive Collision Predictor [Yoaz+'99]
- See our paper for training

Indexed Forwarding (with Delay) Performance

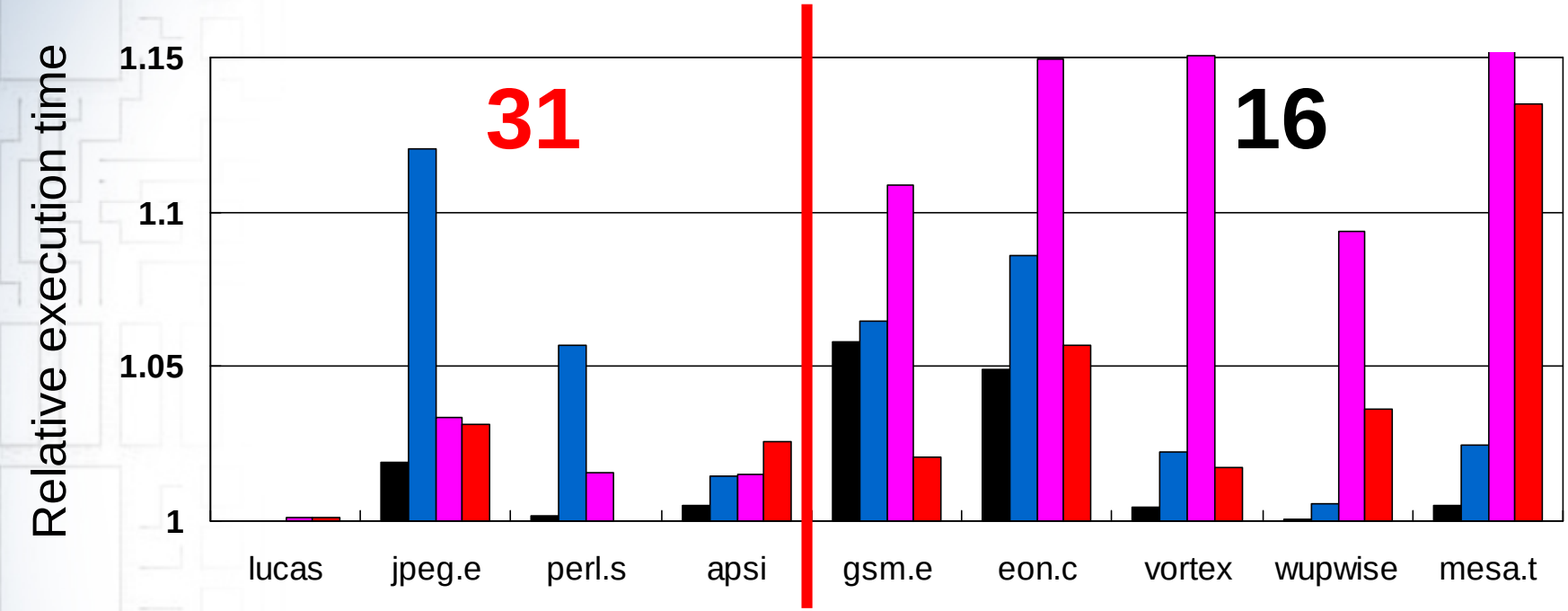


Indexed Forwarding (with Delay) Performance



- 3 cycle indexed SQ + delay
  - Forwarding prediction is good: not much impact (sometimes ±1%)
  - \+ FSP 2-store limitation: a few short delays
  - ± Not-most-recent forwarding: many long delays
    - Better than flushing … but not much

Performance Summary



- 3 cycle indexed SQ + delay
  - + On average 3% slower than 3 cycle associative SQ (black)
  - + On average as fast as 5-cycle associative SQ (blue)

Conclusion

- Problem
  - Scalability of store-load forwarding
- Approach
  - Keep age-ordered store queue
  - Replace associative search with indexed access
  - Adapted Store-Sets predictor predicts forwarding index
  - Adapted Exclusive Collision predictor delays difficult loads
- Effectiveness
  - + **As fast as realistic 5 cycle associative SQ**
  - + **But simpler: unified scheduler, no forwarding replays,…**

Indexed store queue + load re-execution =
CAM free in-flight load/store unit

The End

- Thank you!