Runtime Validation of Memory Ordering Using Constraint Graph Checking

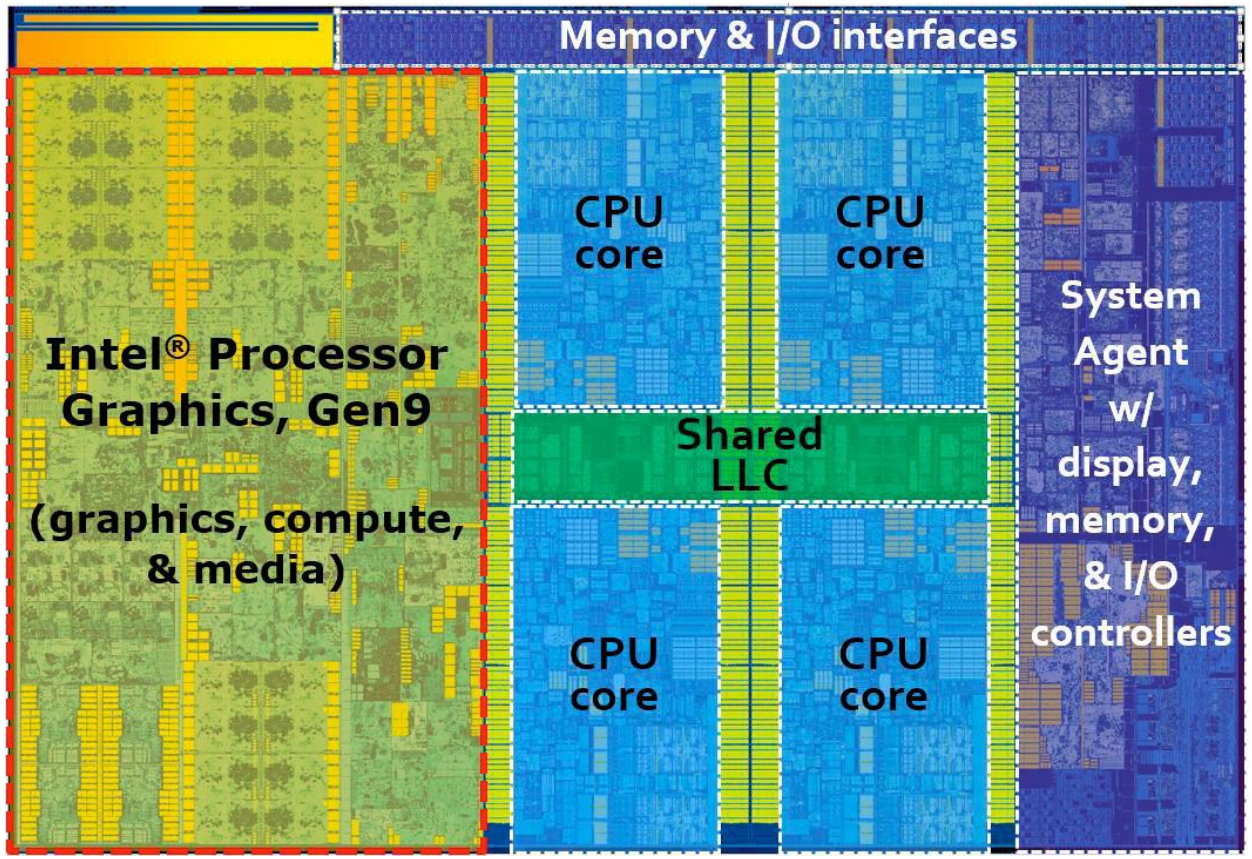
Kaiyu Chen, Sharad Malik and Priyadarsan Patra

Presented by:
Xiaoming Guo
Sijia He

Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

Cores Are Getting Complicated…

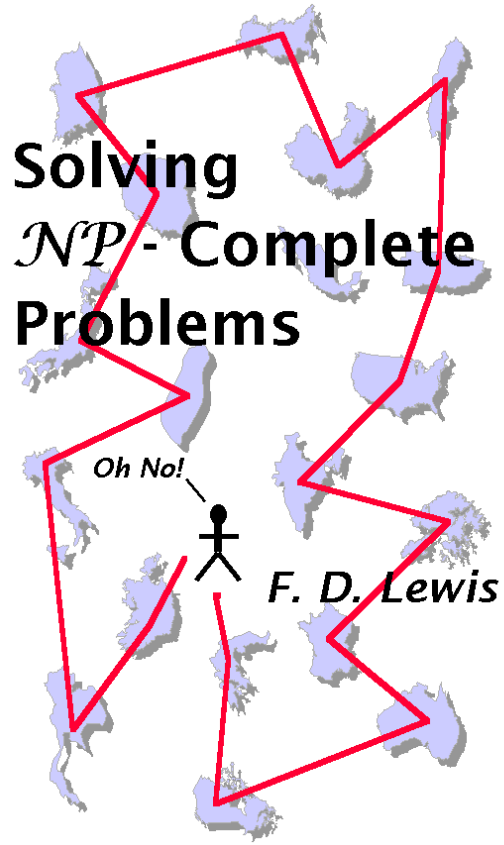


https://www.techpowerup.com/215333/intel-skylake-die-layout-detailed.html

- Three levels of $$$
- "Hyper-threading"
- Aggressive re-ordering
- Interacting via shared memory

Difficulty in Verifying Memory Orderings

- Verifying memory consistency is NP-Complete!
- Formal method cannot be applied to runtime environment
- Simulation based verification is limited by speed
- **Resort to runtime validation!**

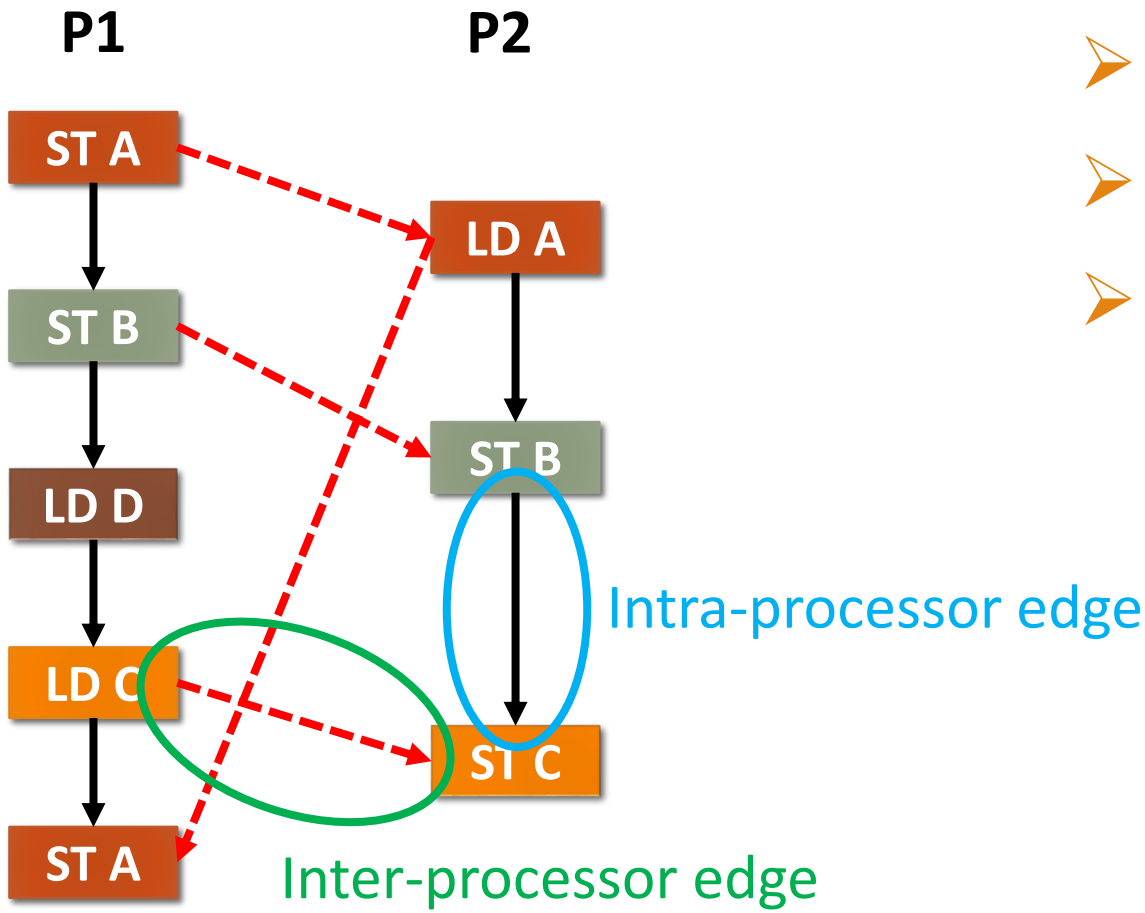


http://freecomputerbooks.com/Solving-NP-Complete-Problems.html

Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

Constrained Graphs 101



- A -> B: A happens before B
- Solid line: Consistency edges
- Dotted line: Dependence edges

Pop Quiz: Which consistency model best describes the ordering shown on the left?

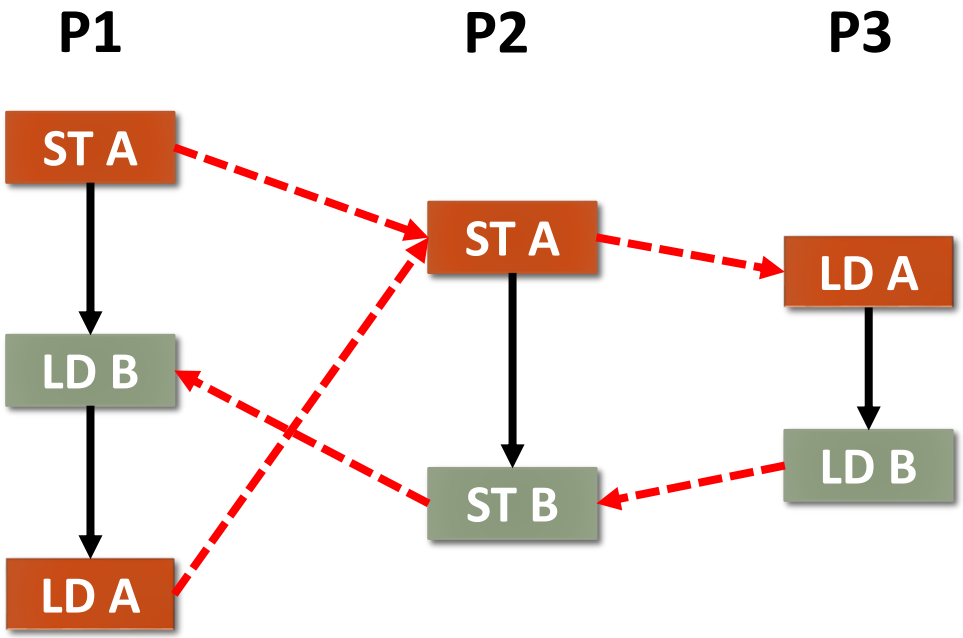
A: SC

B: TSO

C: RMO

D: UFO

Cycles in Constrained Graphs



- Assume somethings wrong with ordering
- A cycle is formed in the graph
- Cycles indicate consistency violations
- **Can be used to validate memory ordering**

Graph taken from EECS 578 lecture slides 10

Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

Key ideas

Solutions:

- Add hardware at each processor to capture share-memory operation orderings
- Perform online validation by checking for cycles in the constrained graphs

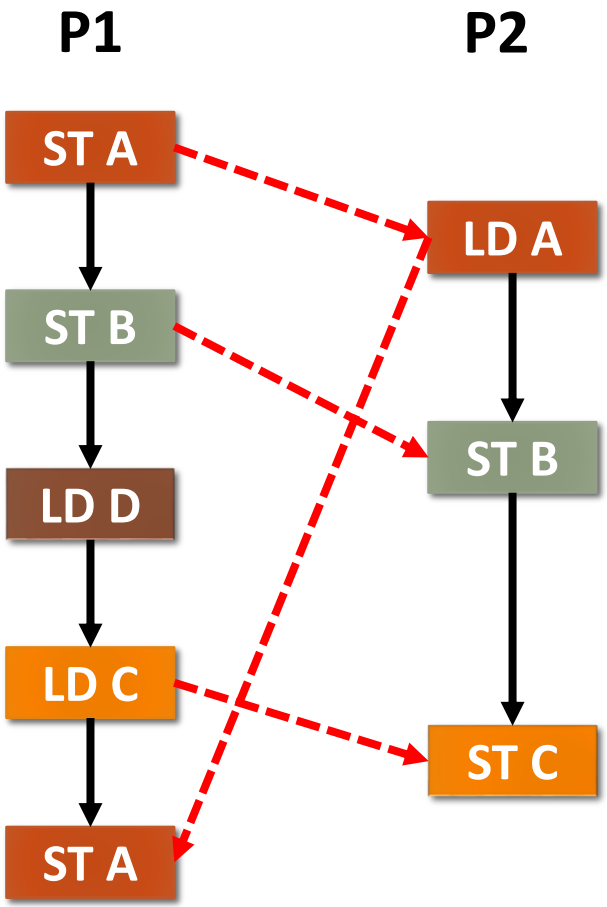
Problems:

- The size of a cycle may be unbounded
- Including all executed memory instructions is infeasible due to limited storage

Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

Constrained Graph Reduction



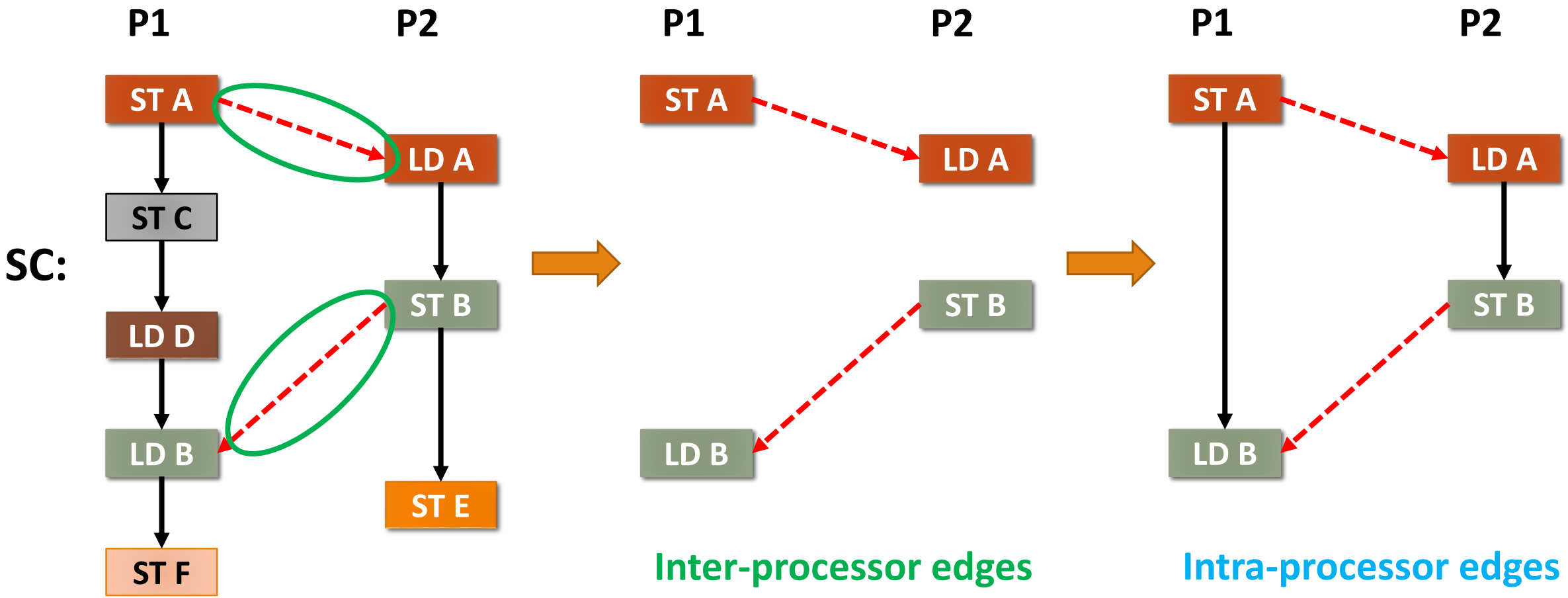
Cannot store every executed MEM OP?

Store an equivalent but reduced graph!

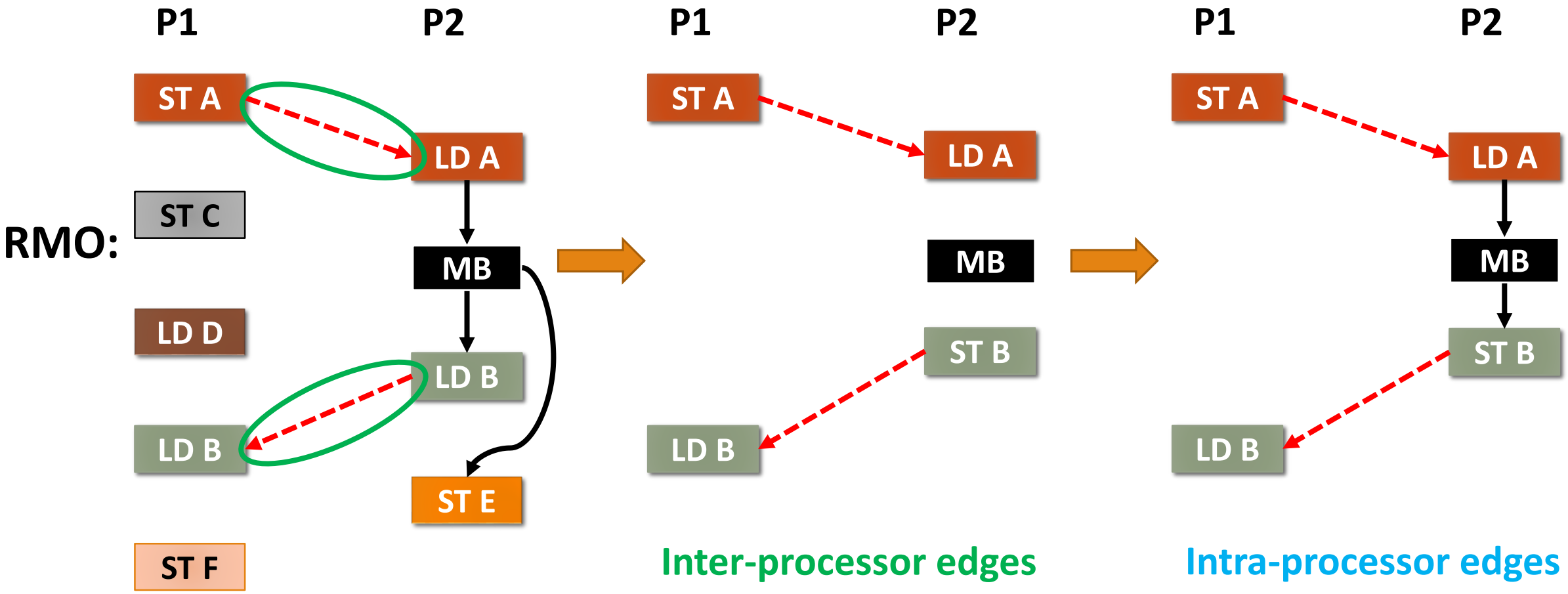
1. Only capture inter-processor dependence edges
2. Build intra-processor edges according to consistency models

The proof is omitted.

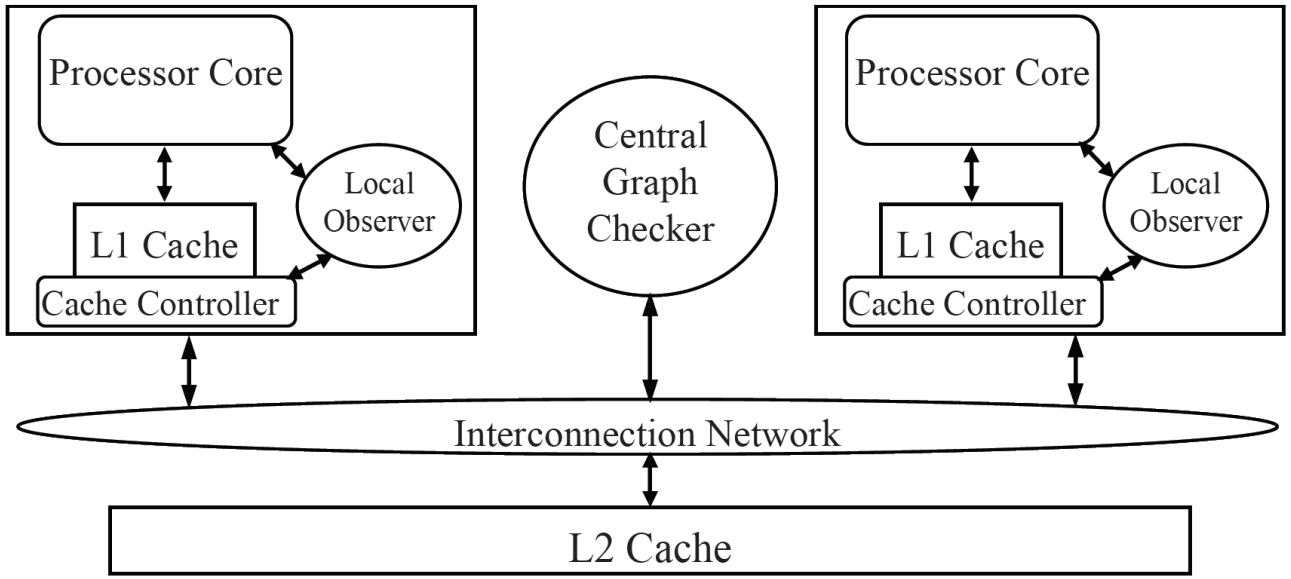
Constrained Graph Reduction: An Example for SC



Constrained Graph Reduction: An Example for RMO



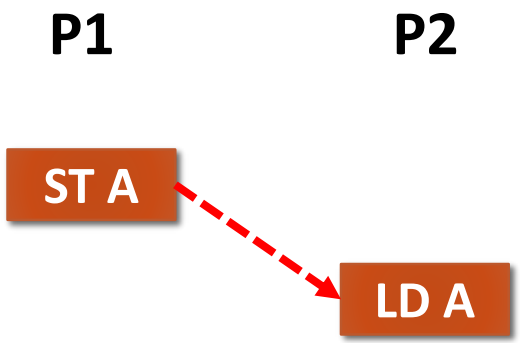
Microarchitecture



- Augment pipeline to assign each memory operation its own ID, called MID
- Augment L1 $ to store local access history
- Local Observer captures inter-processer edges and stores them in cache controller
- Central Graph Checker builds intra-processor edges and performs checking
- Augment L2 $ to store evicted memory access info from L1 $ (like victim cache)

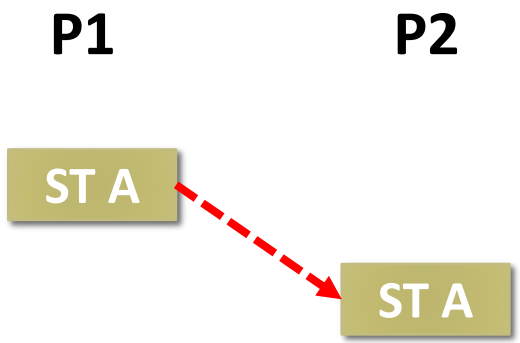
Constrained Graph Edge Construction

Each inter-processor edge corresponds to different cache coherence events



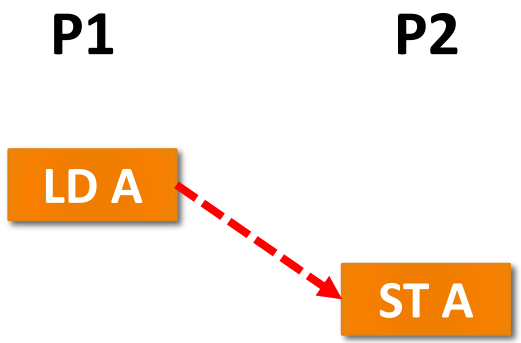
RAW edge:

- Read miss
- Transfer modified data from P1 to P2



WAW edge:

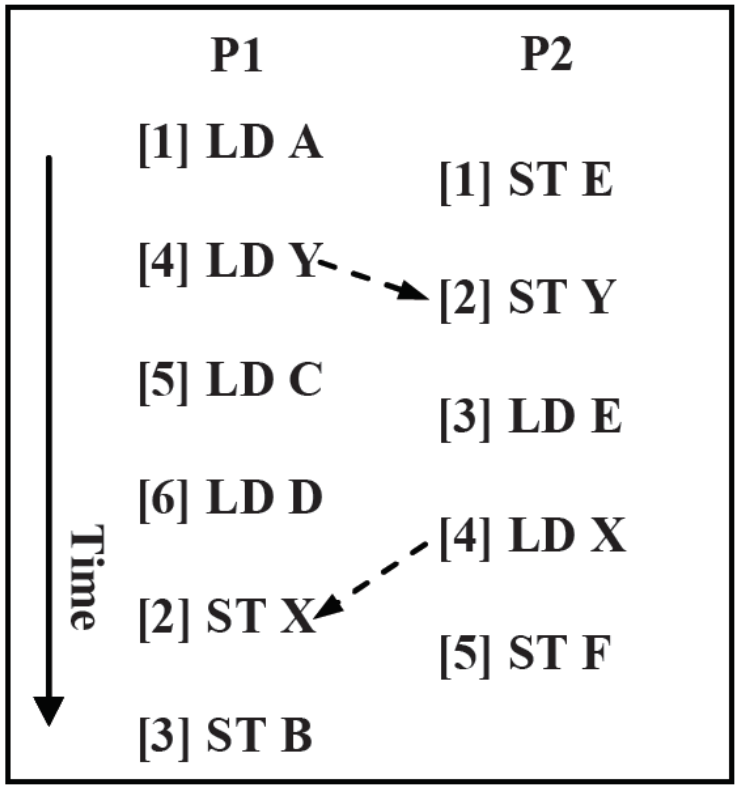
- Write miss
- Transfer modified data from P1 to P2
- P2 upgrades to M state



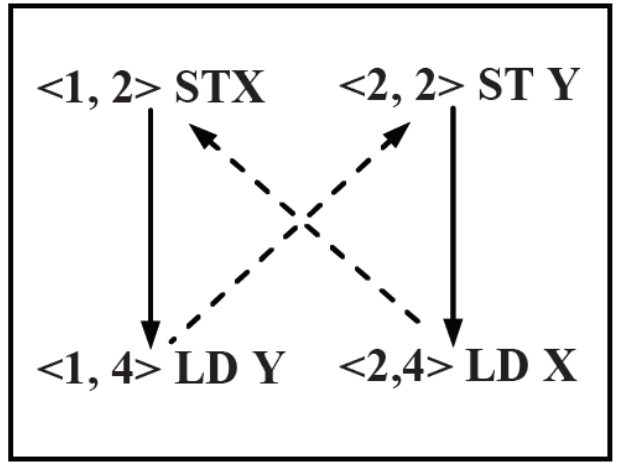
WAR edge:

- Upgrade P2 to M
- Transfer clean data from P1 to P2 (if P2 is a write miss)

Constrained Graph Edge Construction: An Example



1. <P2, 2> performs ST Y
2. <P2, 2> generates invalidation message and send to P1
3. P1 receives the info and construct dependence edge <P1, 4> -> <P2, 2>
4. <P2, 4> transfers data and "pass dirty" to <P1, 2>
5. Edge <P2,4> -> <P1, 2> can be constructed

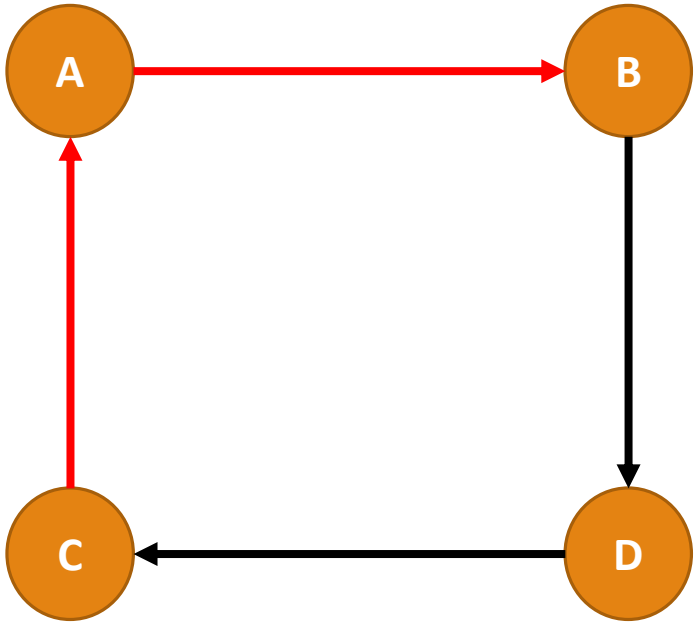


Reduced graph is constructed by building intra-processor edges. A cycle is found.

Do We Really Have Unbounded Window?

- Actually, the cycle is not unbounded in practice.
- A cycle comes from re-ordering
- Only a limited window of re-ordering in hardware
- Can prune the sub-graph if it is not possible to contribute to a cycle
- Simplifies the hardware and reduce storage needed

Subgraph Pruning

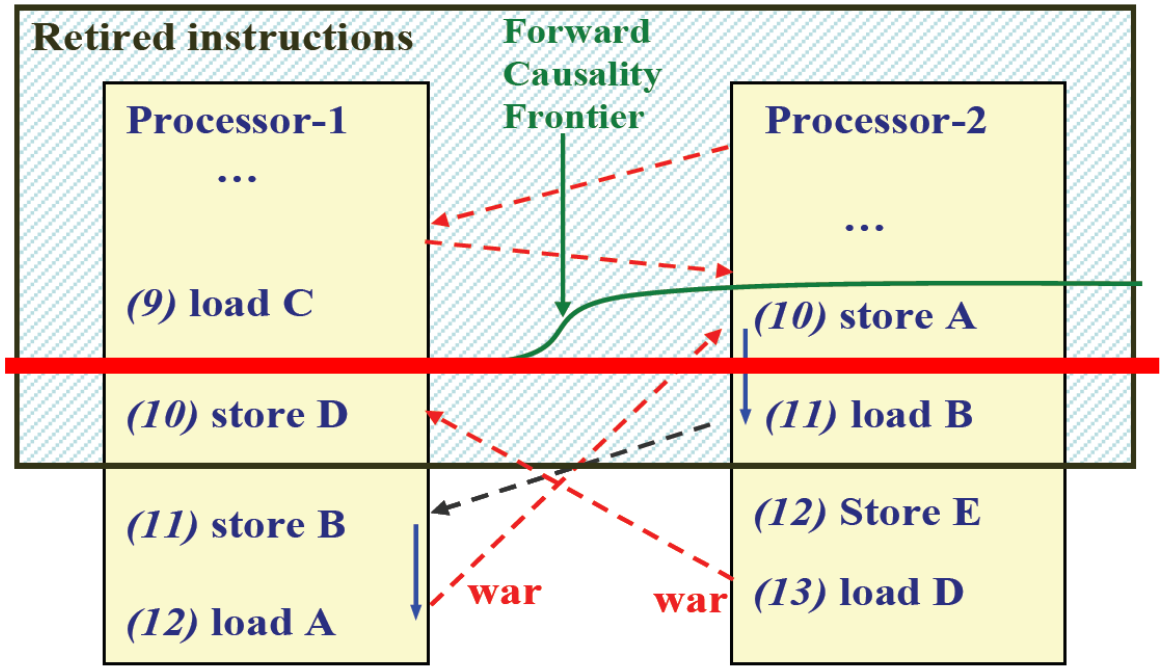


To form a cycle:

Both incoming & outgoing edges

If we can ensure that A will not have incoming edge, we can take it away from checking the cycle.

Constrained Graph Edge Slicing



Can we draw a FCF like that? No!!!

Key observation:

A retired instruction cannot have incoming edges from subsequent instructions

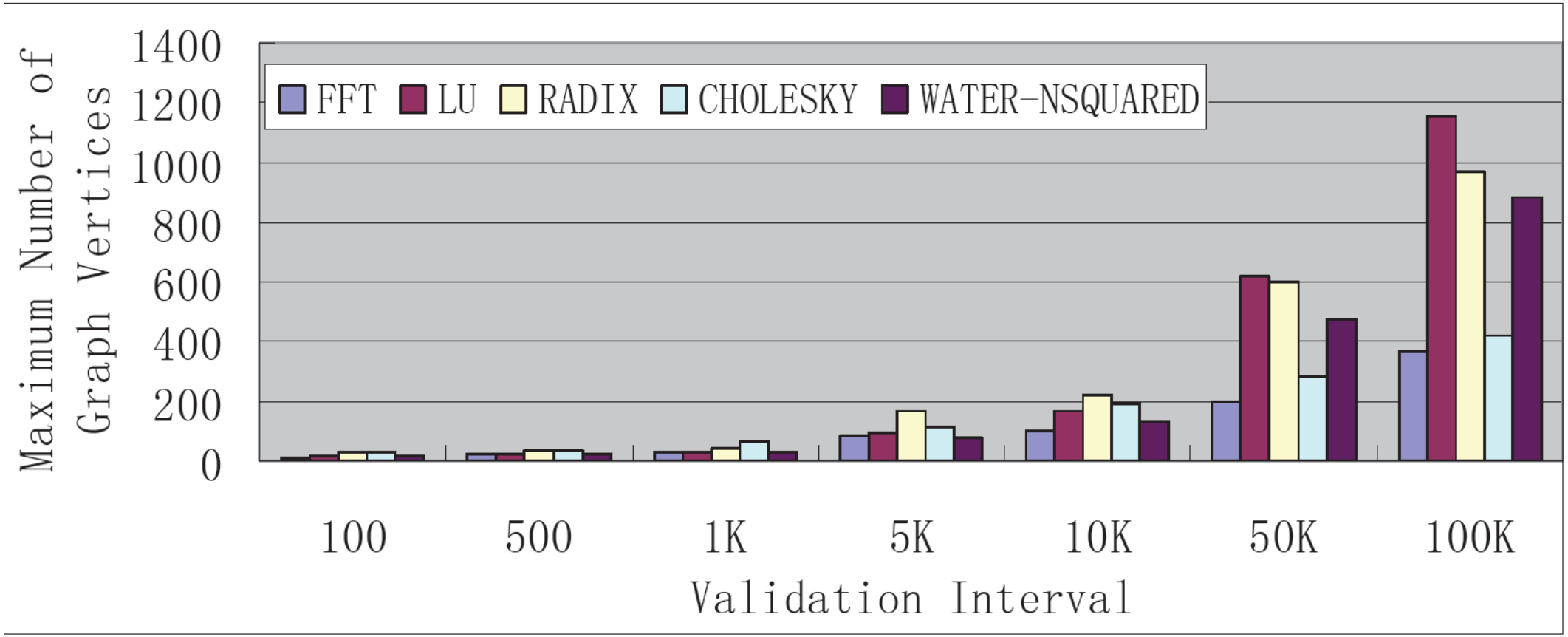
Forward Causality Frontier (FCF):

- No back-edge across the boundary
- Defined as the oldest retired instruction that is reachable from any unretired instruction
- Deallocate records for everything above FCF

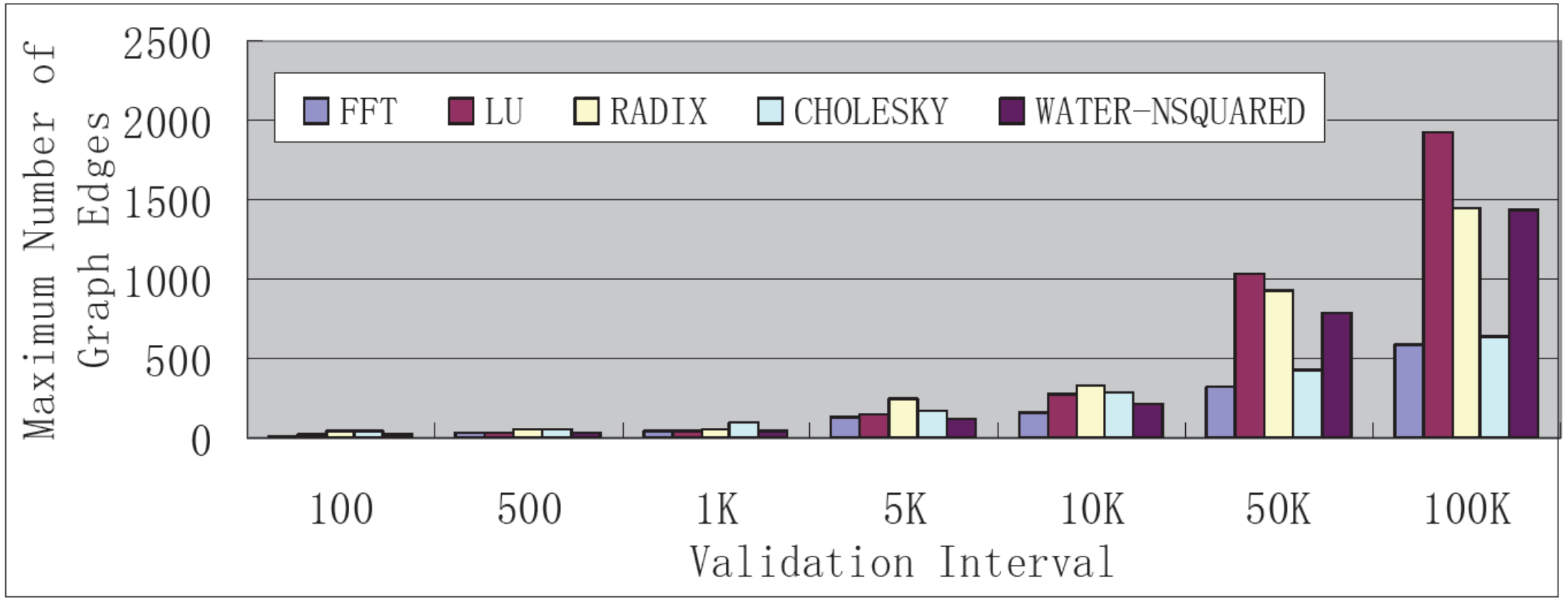
Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

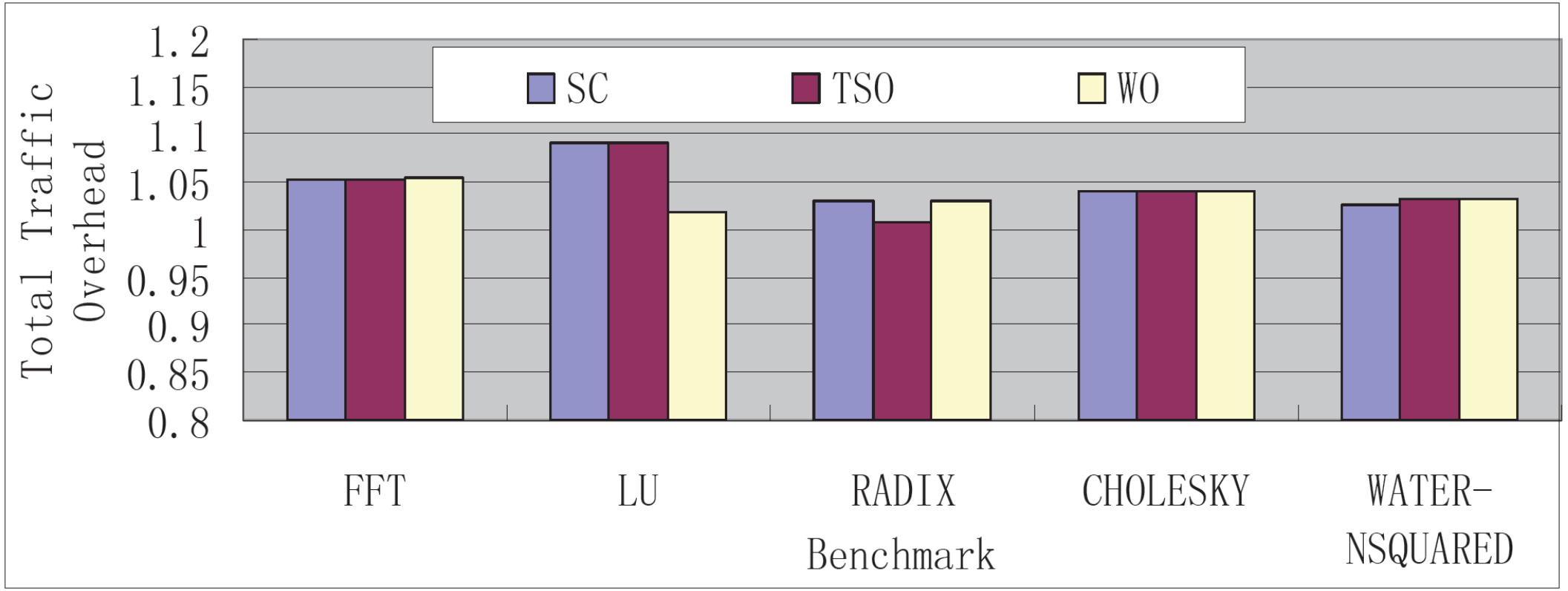
Max # of Vertices vs. Validation Interval



Max # of Edges vs. Validation Interval



Traffic Overhead



Hardware Overhead

Local access record at L1 cache	4 bytes/block * 1000 blocks	4 KB
Locally recorded edge list	8 bytes/entry * 128 entries	1 KB
Evicted access record at L2 cache	12 bytes/entry * 256 entries	3 KB
Central graph checker		4 KB

Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

Related Work

Deterministic replay and race detection:

- Records dependency edges utilizing coherence hardware
- Does not address issues such as storage overhead, unbounded window, etc.

Validating SC using indirect verification of system invariants:

- Only applies to SC
- Introduces false positive

Outline

- Motivation
- Background
- Key ideas
- Implementations
- Evaluation
- Related work
- Conclusion
- Discussion

Conclusion

- Validation of memory ordering is challenging
- Propose a runtime validation approach
- Use efficient hardware to construct constraint graph and perform cycle checking
- Use constraint graph reduction and constraint graph slicing to reduce overhead

Discussion

1. This paper only simulates a dual-core system. Does this approach have good scalability with increasing number of cores?
2. Facing the false positive problem caused by false sharing, is it better to augment the coherence message and cache line for finer granularity, or rely on the rollback mechanism and accept the performance penalty?