Exploiting Commutativity to Reduce the Cost of Updates to Shared Data in Cache-Coherent Systems

Guowei Zhang, Webb Horn, Daniel Sanchez

MICRO 2015

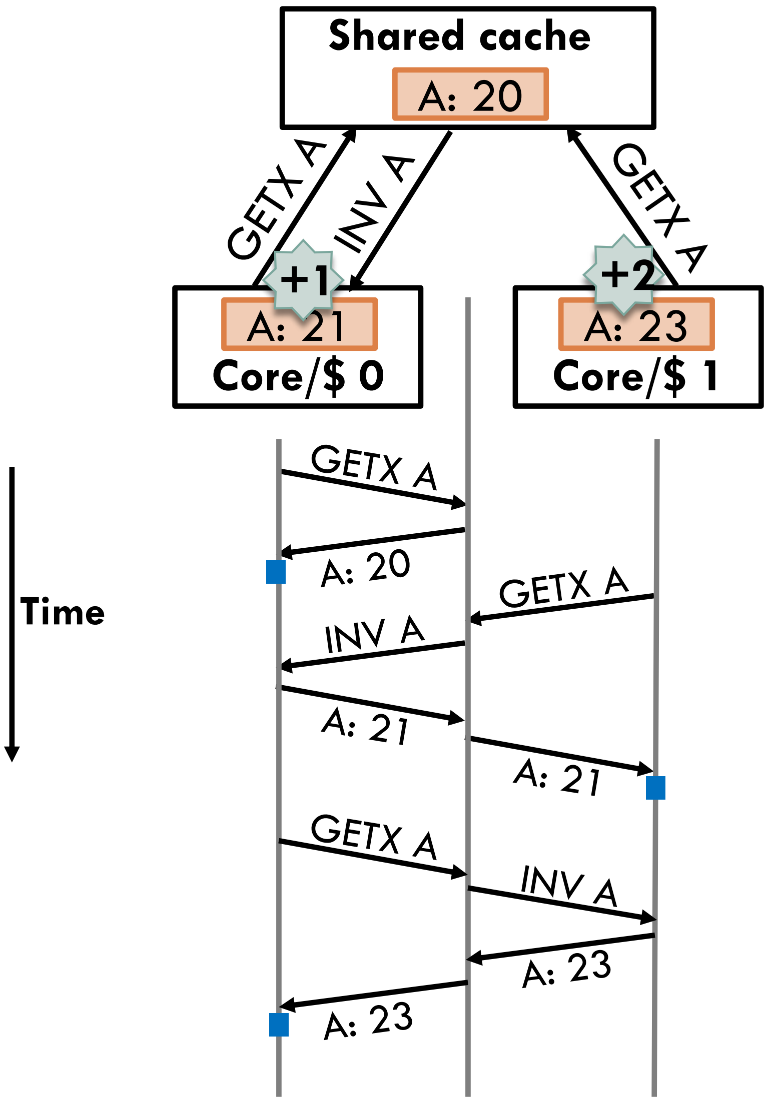
Massachusetts Institute of Technology

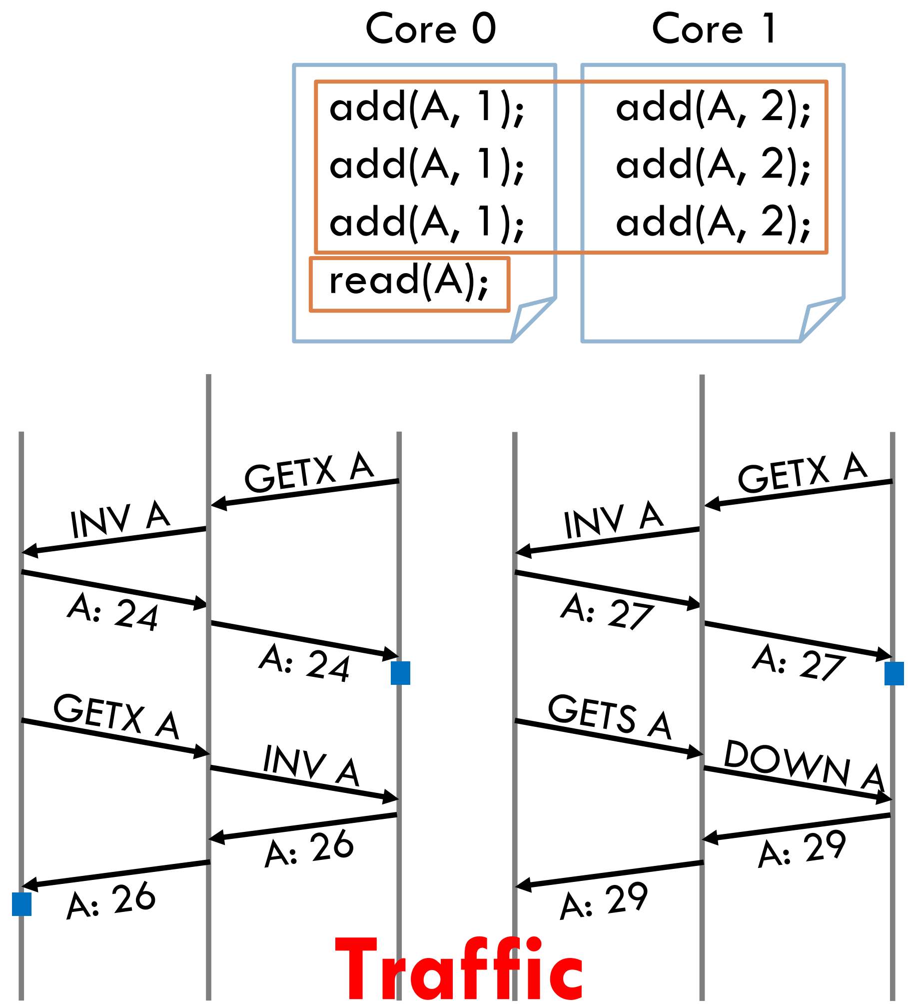
CSAIL

Executive summary

- Updates to shared data limit parallelism in current systems
- Insight: Many updates are commutative
- Coup extends cache coherence protocols to make commutative updates **as cheap as reads**
  - Maintains coherence and consistency
  - Accelerates update-heavy applications significantly

Updates are expensive

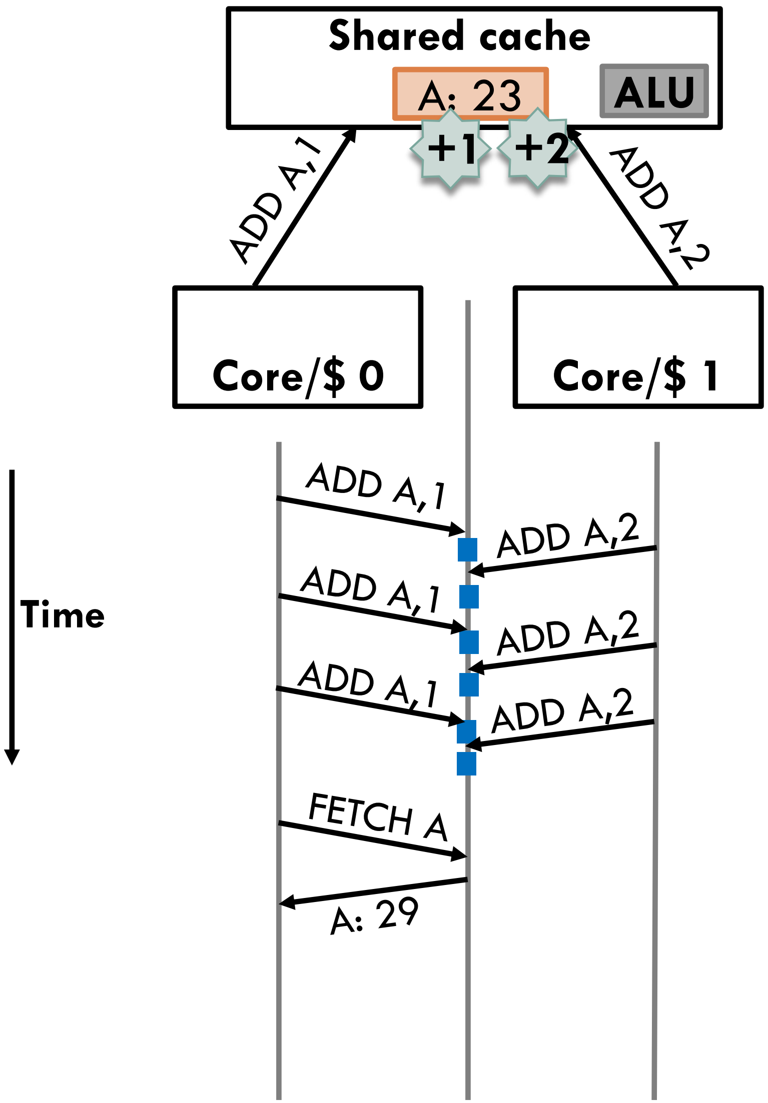




Traffic

Serialization

Updates are expensive, even with RMOs



Core 0

```
add(A, 1);
add(A, 1);
add(A, 1);
read(A);
```

Core 1

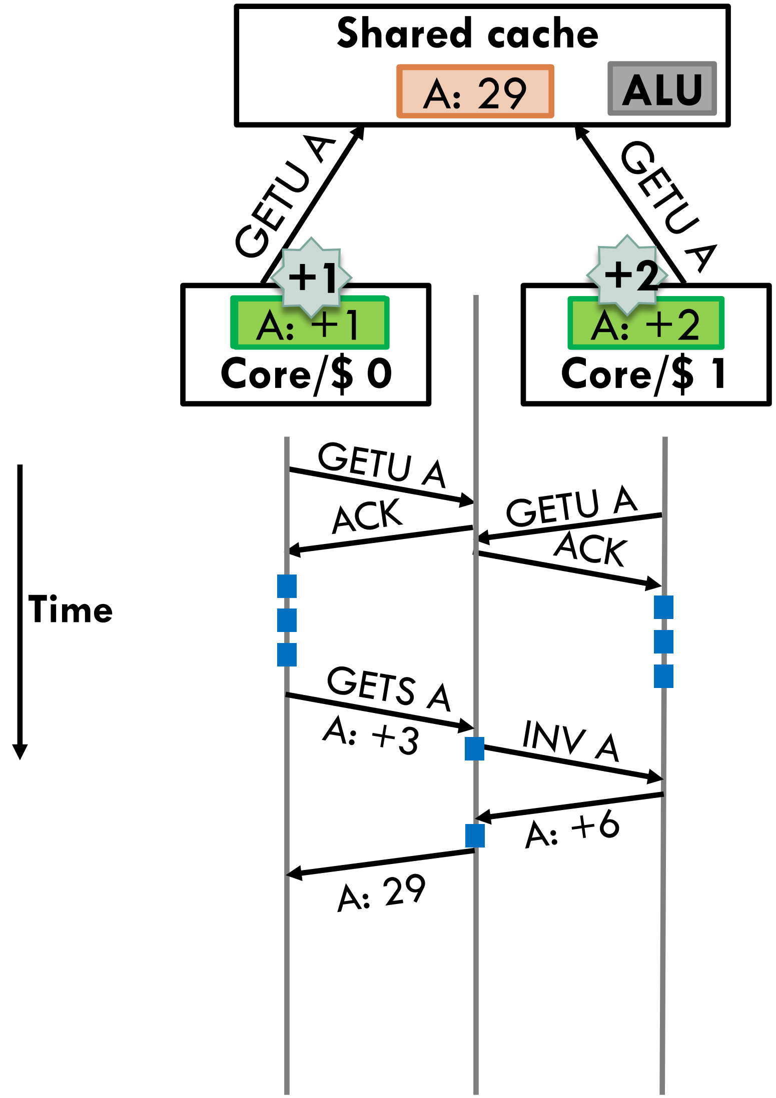
```
add(A, 2);
add(A, 2);
add(A, 2);
```

Traffic

Serialization

Complicates consistency

Coup: exploiting commutativity



Core 0	Core 1

add(A, 1);
add(A, 1);
add(A, 1);
read(A);
	add(A, 2);
	add(A, 2);
	add(A, 2);

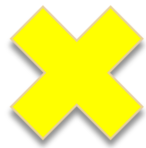
Low traffic

Concurrent updates

Simple consistency

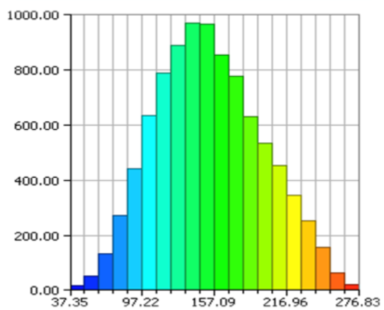
Less general than RMOs

Commutative updates are common

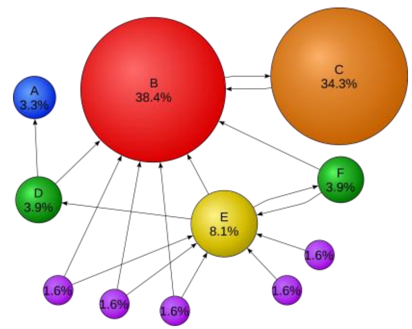
- Operations   MIN OR

- Applications

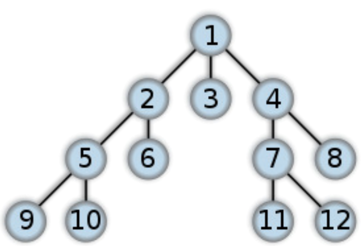
Reduction variables



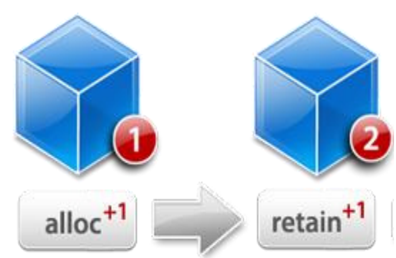
Iterative algorithms



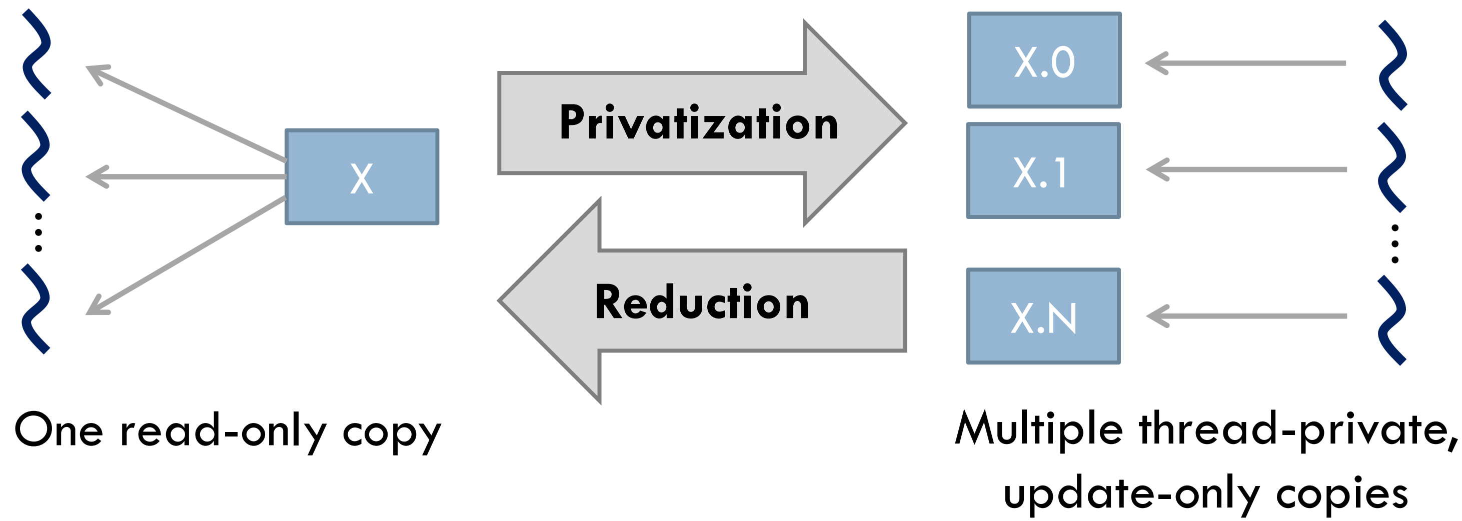
Graph traversal



Reference counting



Software privatization vs. Coup

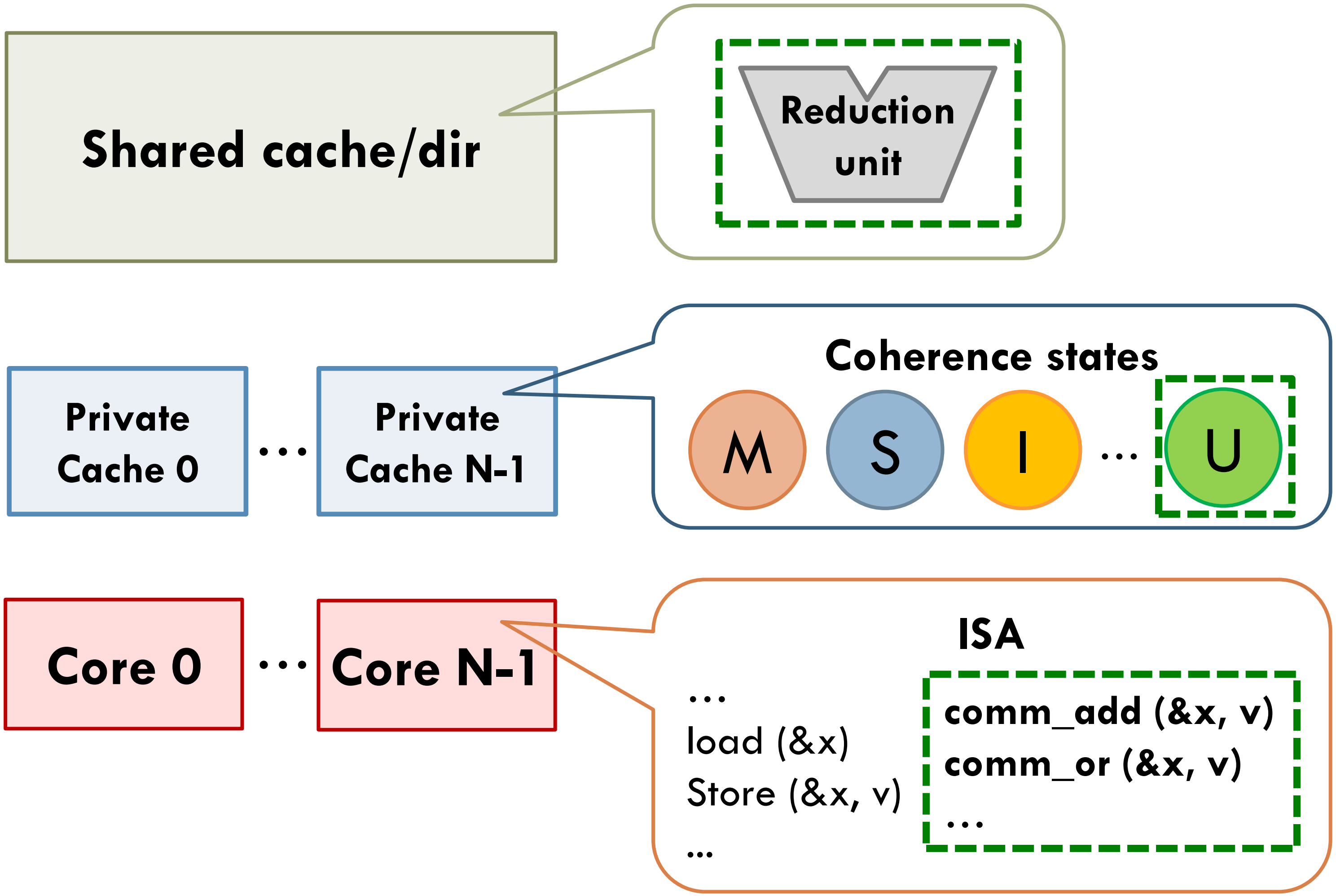


Software privatization	Coup
Needs to amortize privatization/reduction costs	No overheads
Wastes shared cache & memory capacity	No wasted capacity
Must apply selectively	Apply to any update that might commute

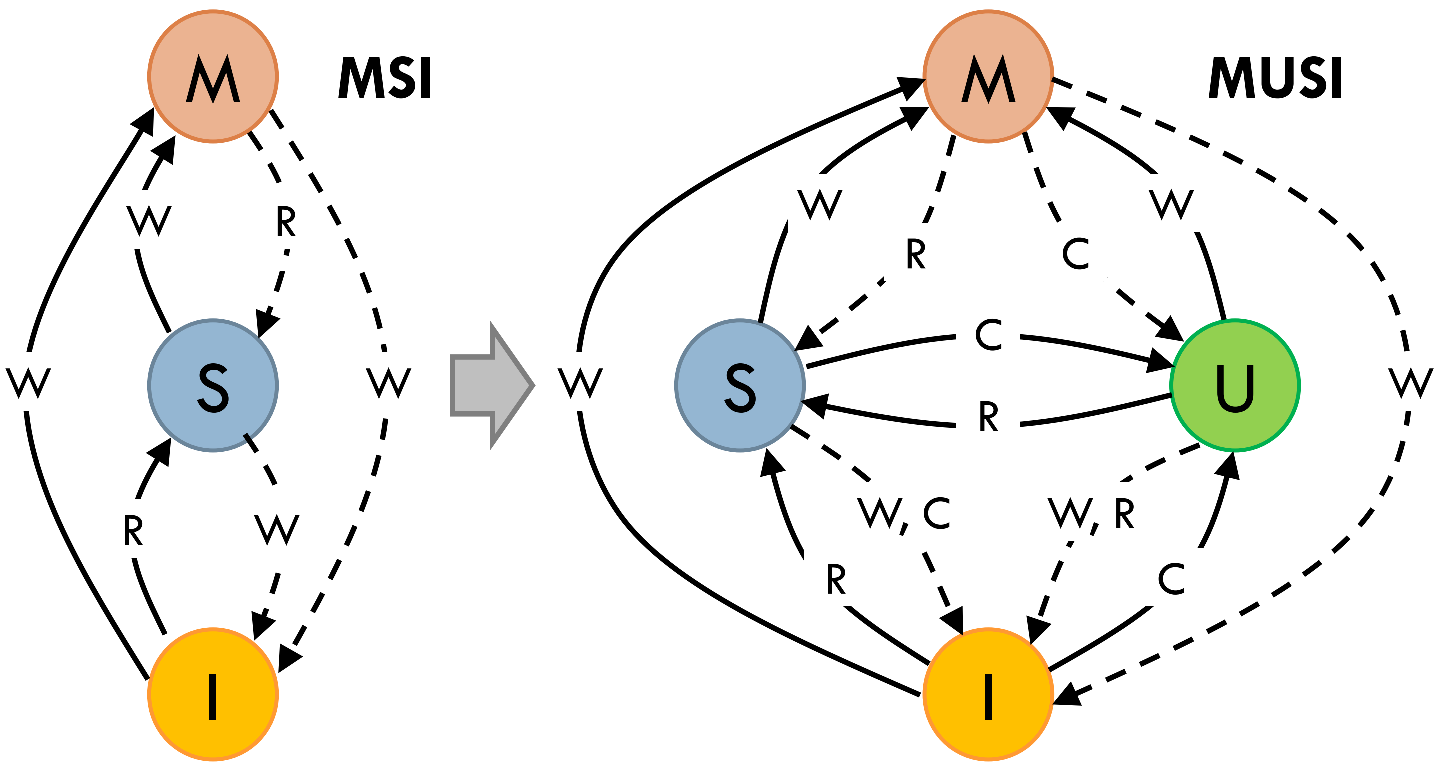
Outline

- Introduction
- Coup
- Evaluation

Structural changes



Example: extending MSI



Legend

Transitions
- Initiated by own core (gain permissions)
- Initiated by others (lose permissions)

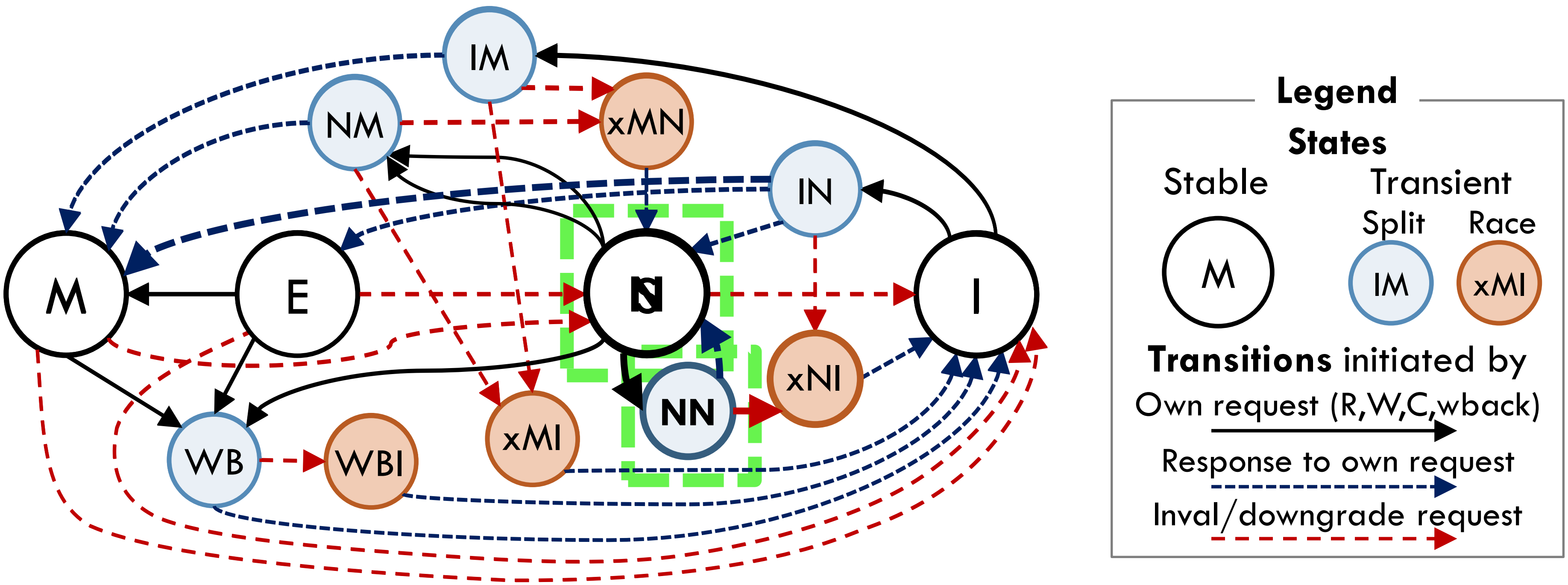
States Modified Shared (read-only) Invalid Update-only

Requests Read Write Commutative update

Coherence and consistency

- Coherence is maintained
- Consistency is not affected
- See paper for proofs

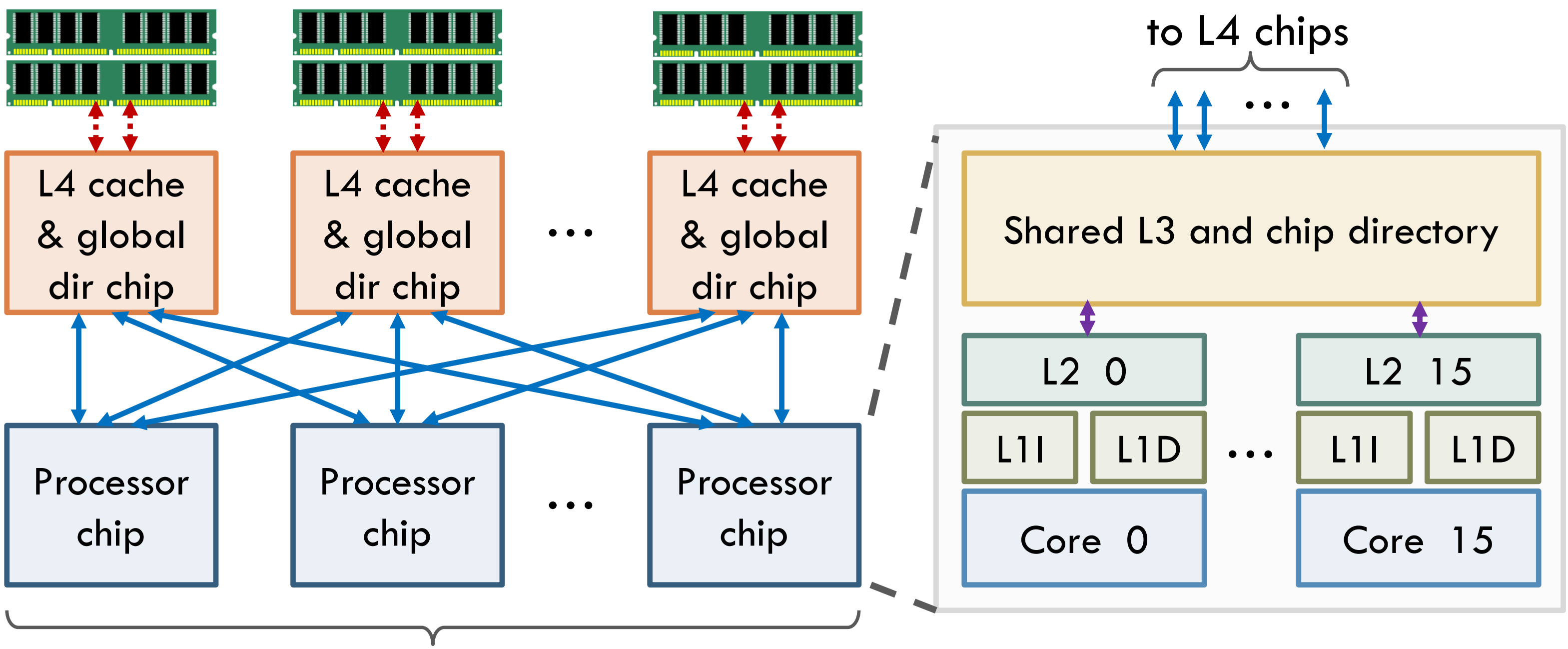
Implementation and verification



No extra stable states

Easy to verify

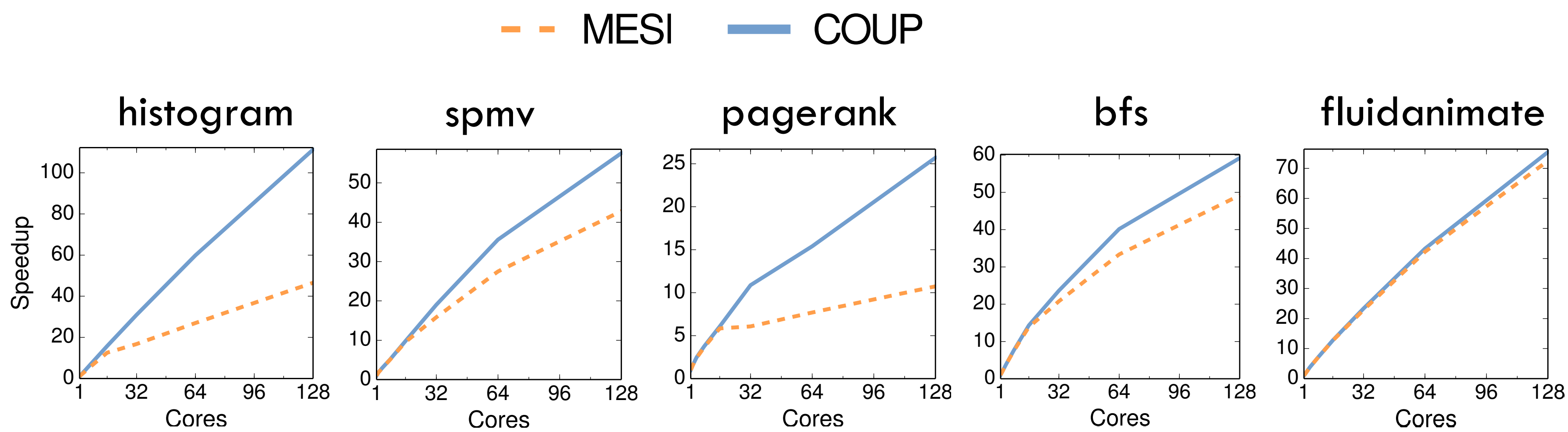
Evaluation Methodology



1-8 processor and L4 chips **Processor chip organization**

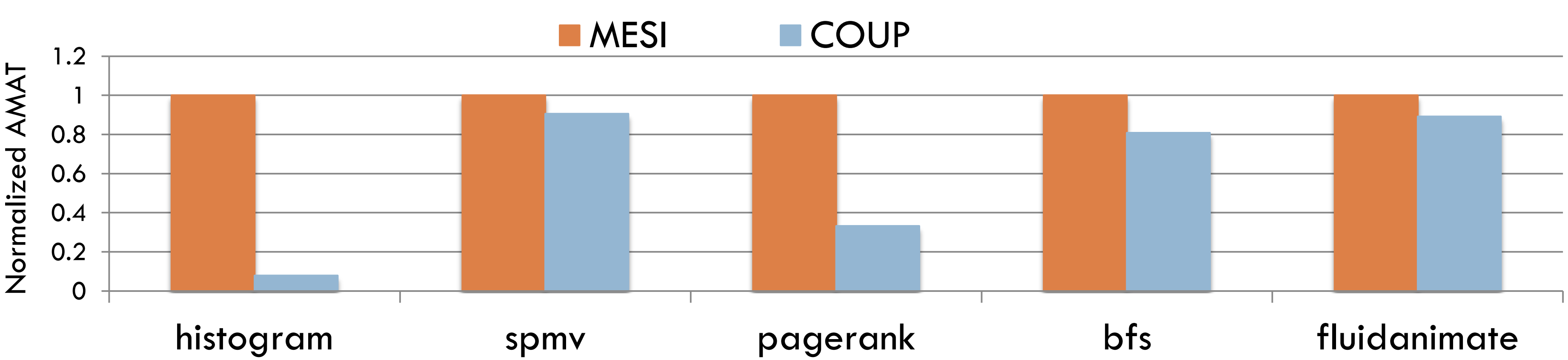
8 sockets × 16 cores/socket = 128 cores

Coup vs. Atomic Operations



Fraction of commutative instructions

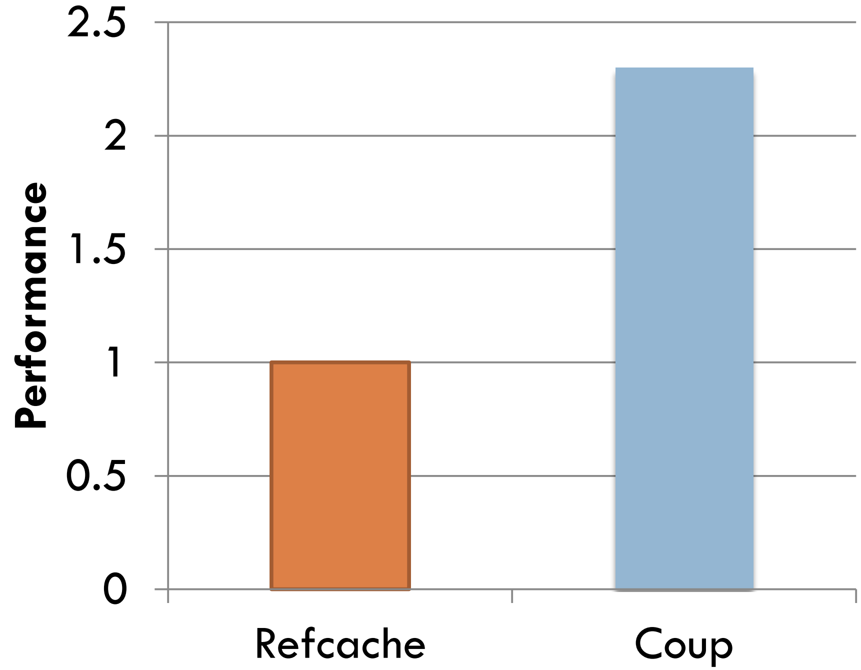
1.0%	2.4%	4.9%	0.40%	0.96%



Modifying algorithms to exploit Coup

Delayed deallocation reference counting

Scheme	Data structure
Refcache[1]	Hash table
Coup implementation	Hierarchical bit vectors + comm add/or



[1] Clements et al, EuroSys 2013

Conclusions

- Coup allows concurrent commutative updates
  - Maintains coherence and consistency
- Coup implementation accelerates single-word updates
  - Minor hardware overhead
  - Accelerates update-heavy applications by up to 2.4x
- Coup opens exciting research avenues
  - Commutativity-aware hardware transactional memory
  - Support arbitrary update functions, semantic commutativity

Thanks For Your Attention!

Questions are welcome!

Massachusetts Institute of Technology

CSAIL