Turning Centralized Coherence and Distributed Critical-Section Execution on their Head: A New Approach for Scalable Distributed Shared Memory

Stefanos Kaxiras

David Klaftenegger

Magnus Norgren

Alberto Ros

Kostis Sagonas

UppsalaUniversity

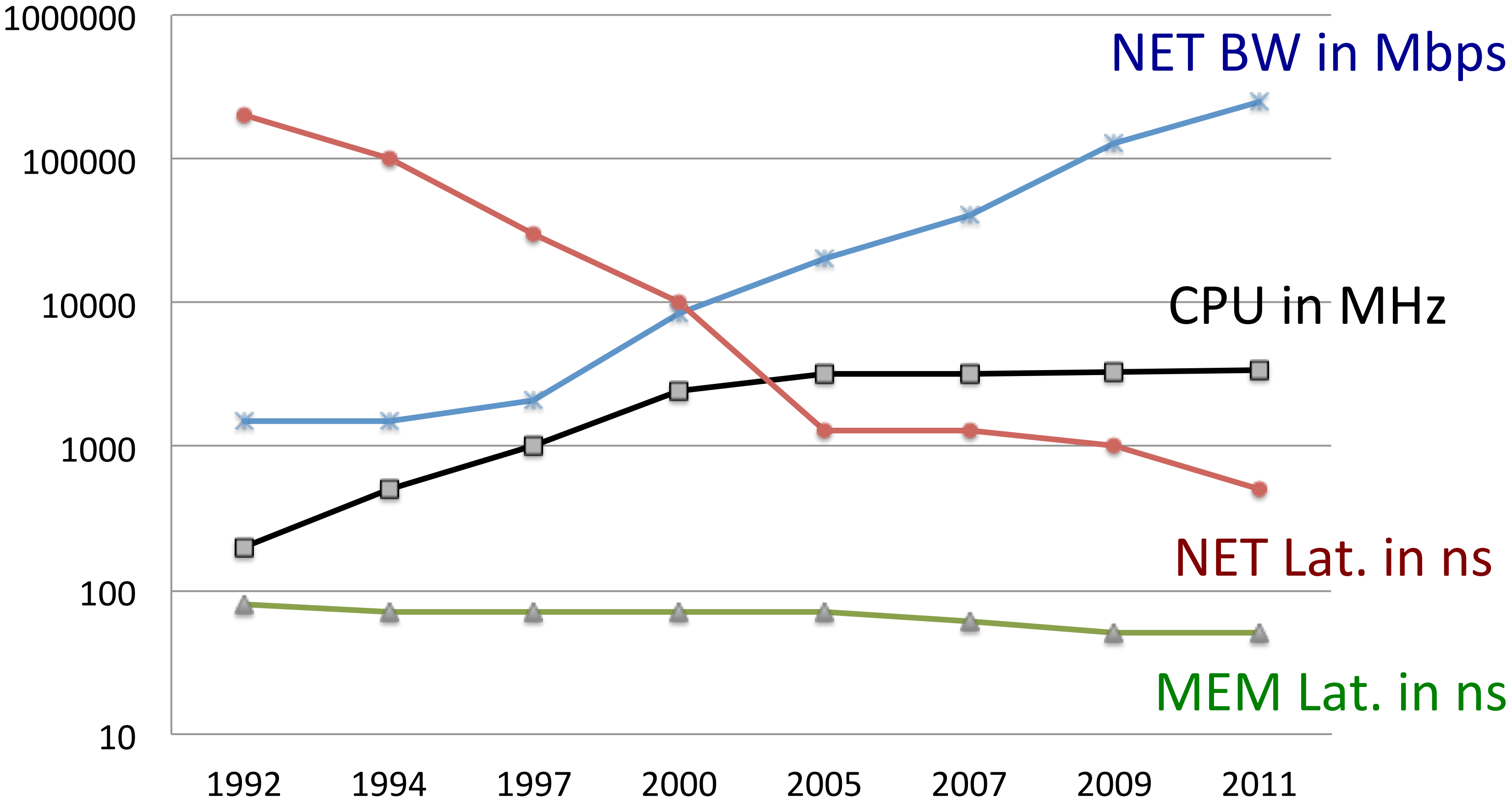
Quote

“Distributed shared memory is one of those holy grails for simplifying parallel programming. Sadly, these systems have never been shown to scale well.”

Anonymous Reviewer

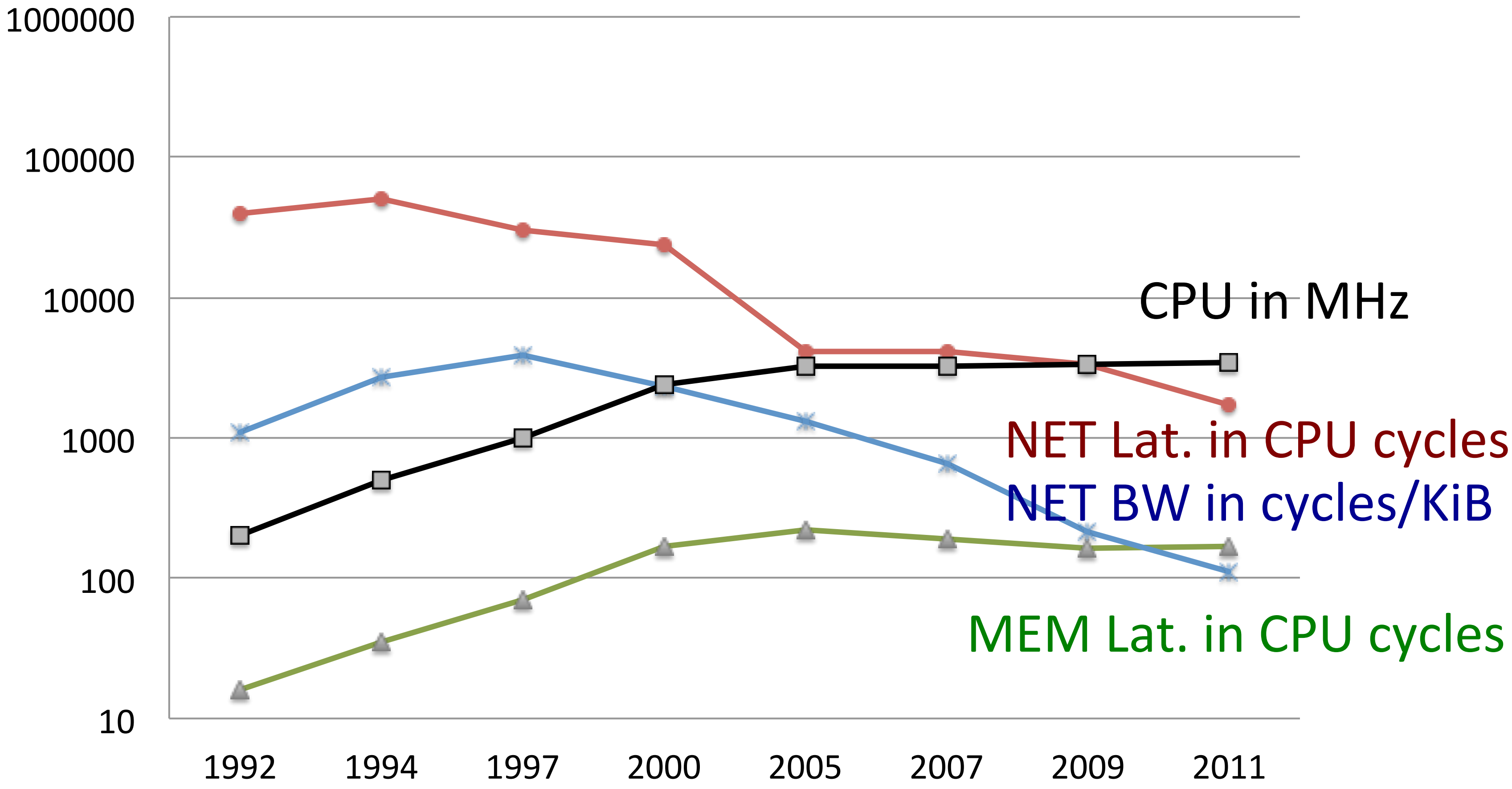
Why try again, why now?

Trends



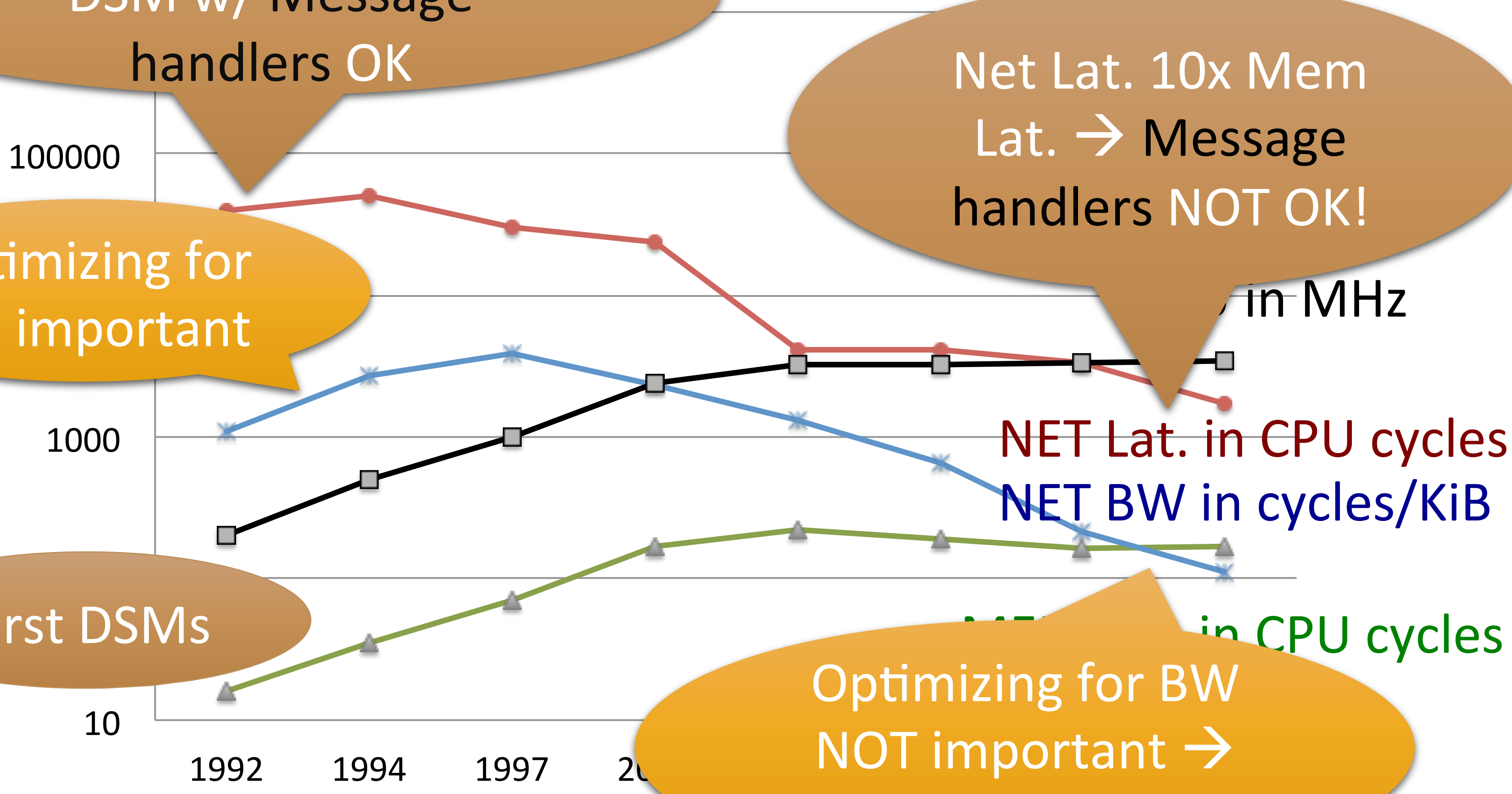
[Ramesh's thesis, 2013]

Trends in CPU cycles



[Ramesh’s thesis, 2013]

Trends in CPU cycles



HUGE Net Latency: DSM w/ Message handlers OK

Net Lat. 10x Mem Lat. → Message handlers NOT OK!

Optimizing for BW important

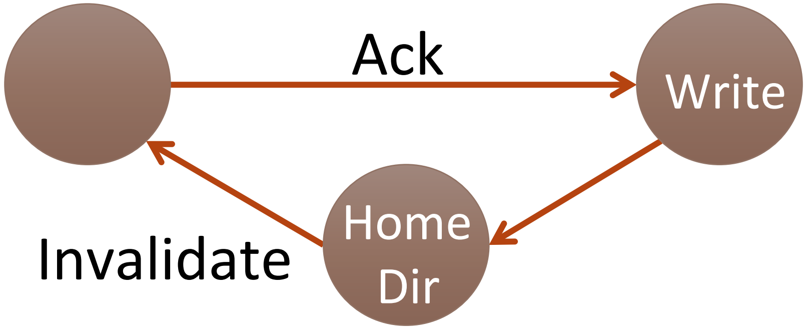
First DSMs

Optimizing for BW NOT important → Trade BW for Lat.!

[Ramesh’s thesis, 2013]

Lesson: We’ve been doing it wrong

- Centralized Coherence
  for *distributed* data

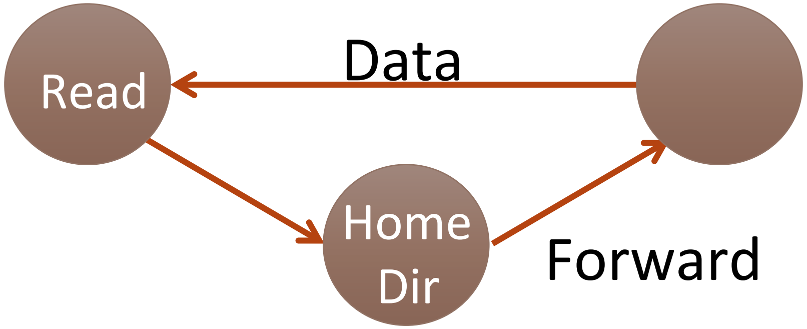


- Distributed CS execution
  for inherently *serial* execution

Lesson: We've been doing it wrong

- Centralized Coherence

  for *distributed* data



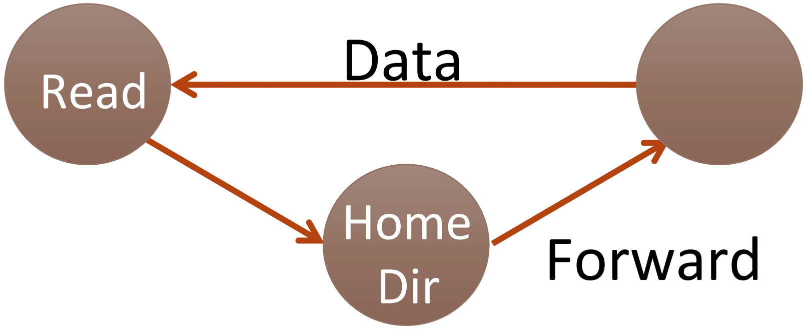
- Distributed CS execution

  for inherently *serial* execution

Lesson: We've been doing it wrong

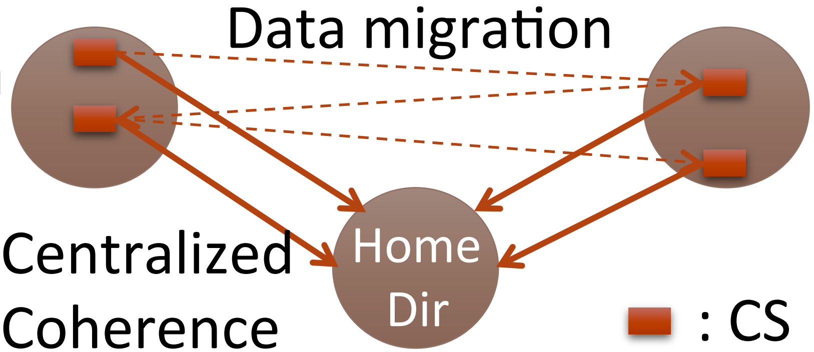
- Centralized Coherence

  for *distributed* data



- Distributed CS execution

  for inherently *serial* execution



🟥 : CS

Centralized
Coherence
Control!

Instead …

- Distributed Coherence
  - Self-downgrade
  - Self-invalidation
- Data-Classification Directories

} Trade BW for Latency

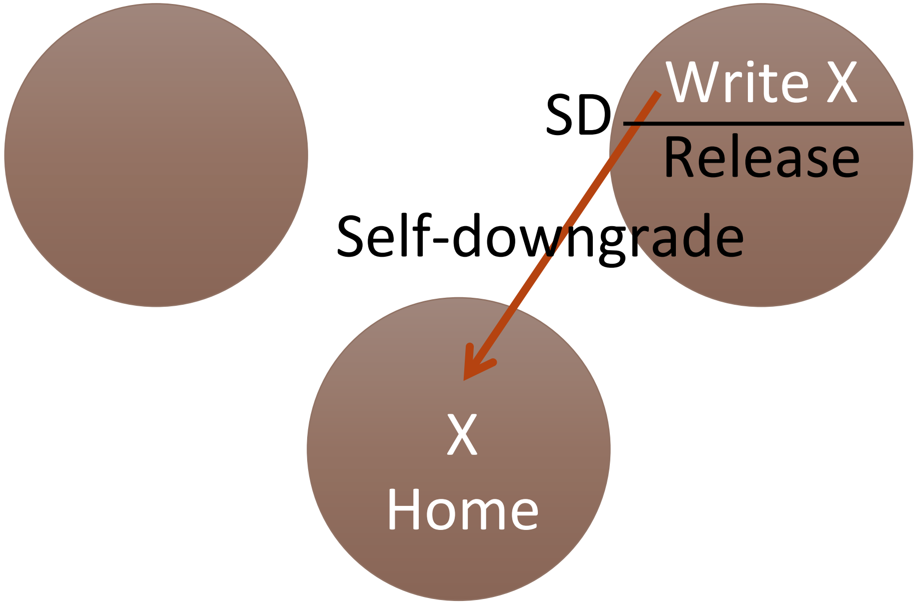
} No message handlers

- Centralized CS execution
  - Queue delegation locking
  - Delegate the CS code to the lock holder

Distributed Coherence

- Self-Downgrade on release (unlock, signal, barrier)
  - No Invalidation on writes

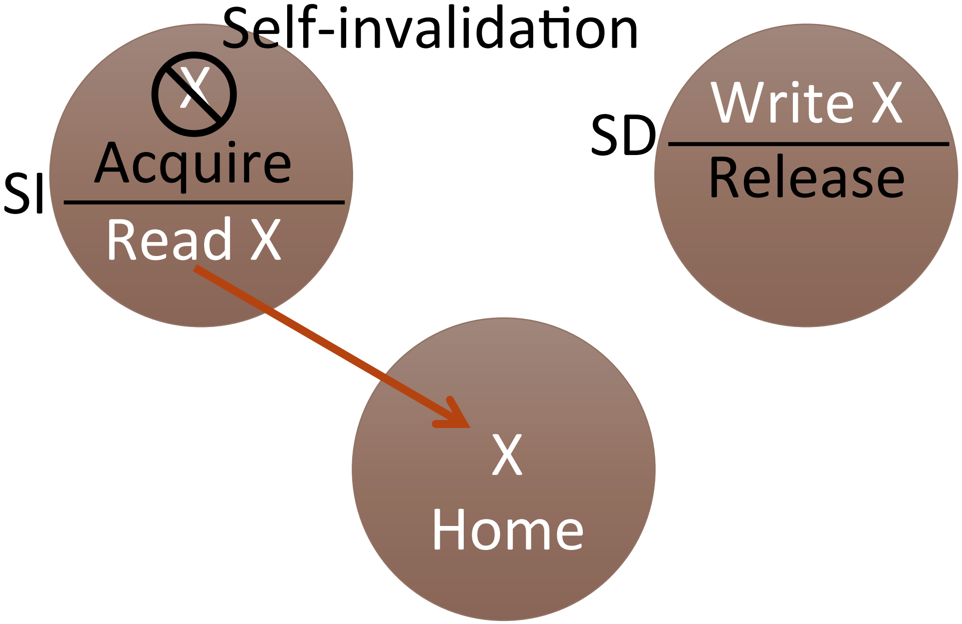
- Write-through implemented with a Software Write Buffer
- Eagerly propagates writes to home nodes (BW)
- Only need to empty WB on release (Lat.)



Distributed Coherence

- Self-invalidation on acquire (lock, wait, barrier)
  - No indirection to find latest value

- No Invalidations → No SC!
- SC for DRF
- Synchronization drives coherence

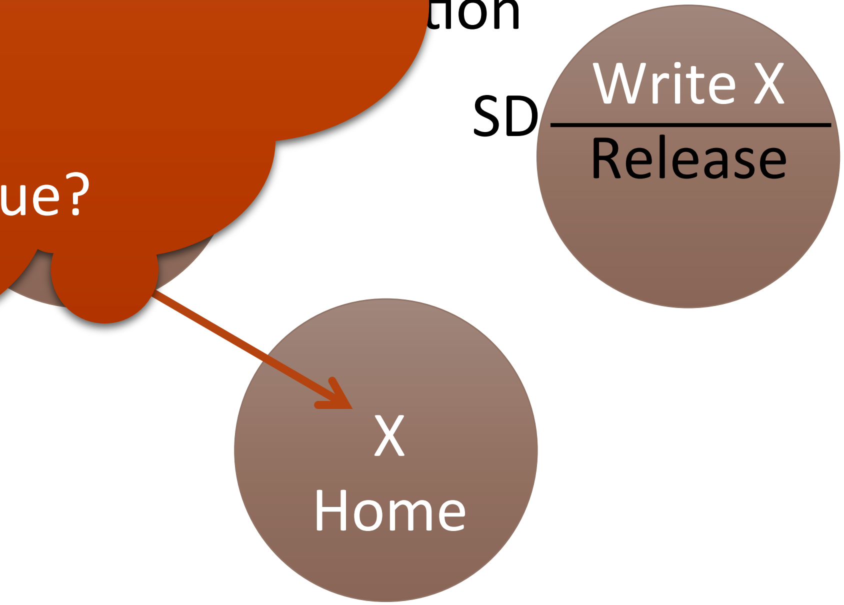


Distributed Coherence

- Self-invalidati o barrier)
  - No ind
- No Inv SC!
- SC for DR
- Synchronization drives coherence

tion

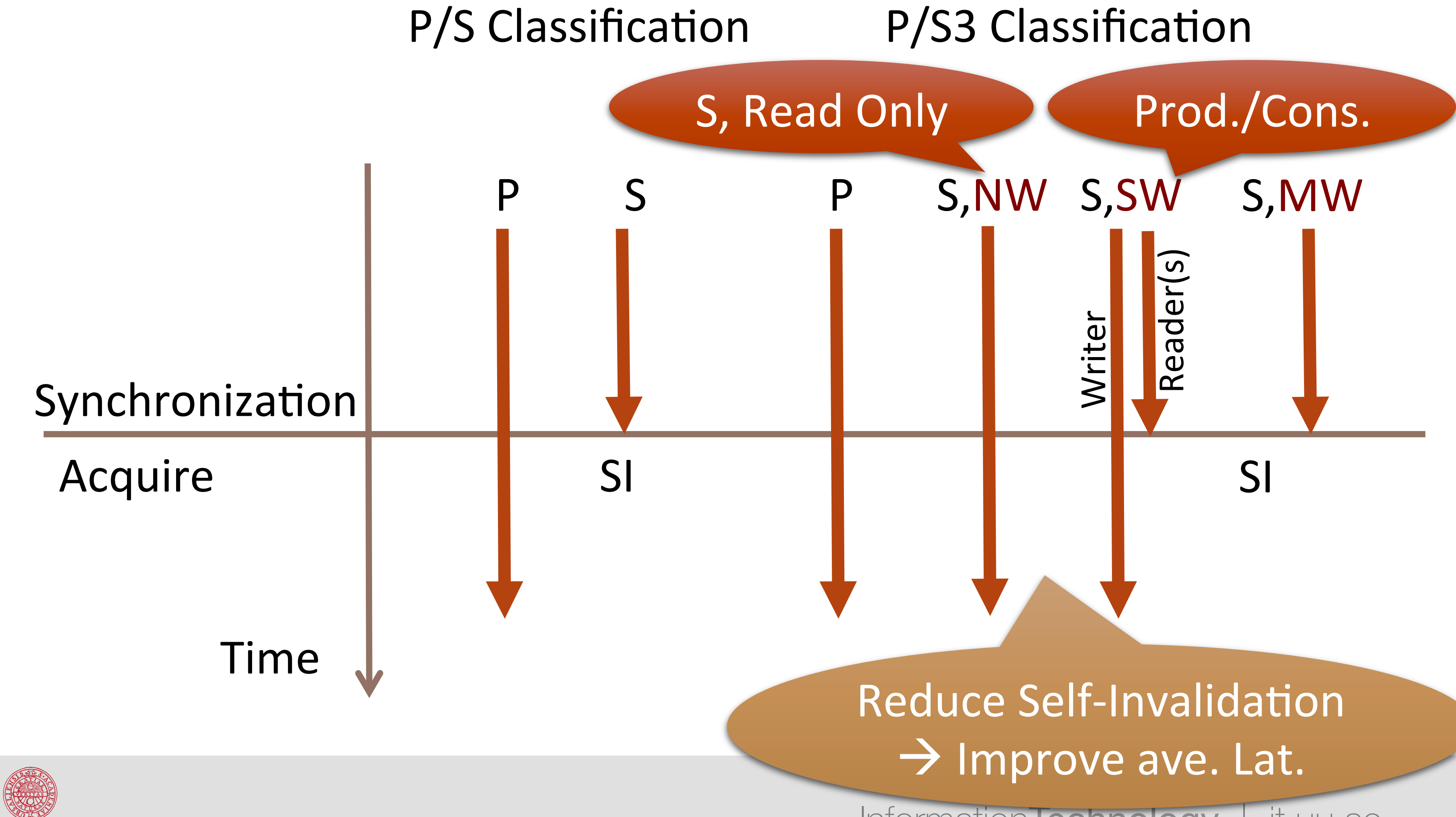
Great!
There are home nodes but NO directories at the home nodes!
…
Too good to be true?



Data-Classification Directories

- Yes and No …
  - No coherence directories …
  - But still some “*sharing*” information would be useful
  - Self-invalidation bad for performance: flush cached data on sync! → Need to be selective
- Passive Classification Directories
  - Info exchanged through directory on reads/writes
- Data Classification:
  - Private / Shared [PACT 2012]
  - Here: a better classification: P/S3

Data-Classification Directories



S, Read Only

Prod./Cons.

Reduce Self-Invalidation → Improve ave. Lat.

Data-Classification Directories

- NO message handlers to classify data and propagate classification changes
- Requestors are responsible to update classification at remote Private or S,SW nodes
- No need to interrupt anyone!
  - Classification changes are discovered at the next request or synchronization point!
- How is this possible?
  - DRF semantics!
- … or even correct?
  - Correct data are always found valid at home: everything self-downgrades (Private & Shared)

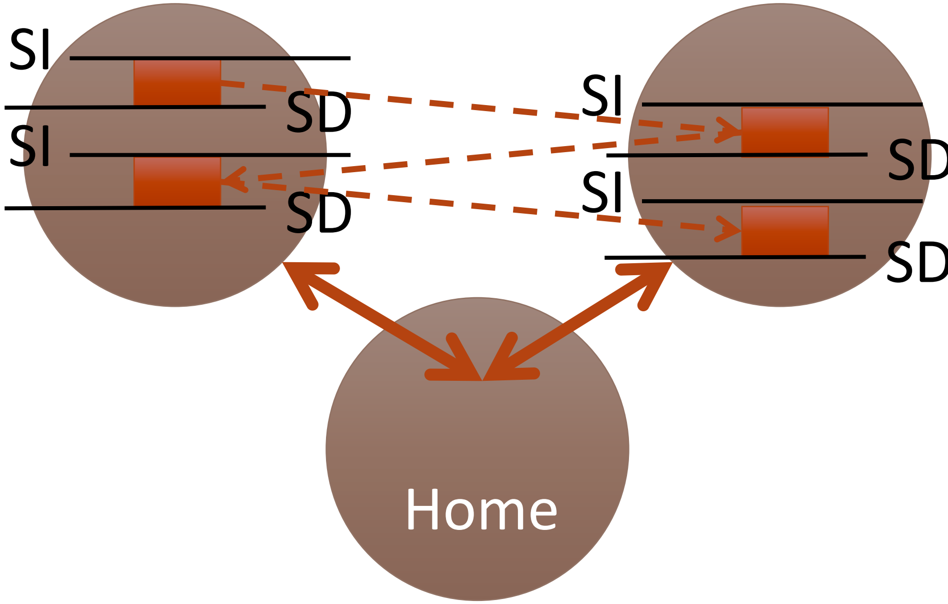
Trade BW (downgrade) for Latency (No message handlers)

Queue-Delegation Locks

- The trouble with distributed CS execution: Inherently *serial* execution that migrates from node to node!

- Worse, we must:
    - SD on every Unlock
    - SI on every Lock

  … causing traffic w/ remote node(s) and havoc in the shared caches



Queue-Delegation Locks

- Queue-Delegation Locking [SPAA'14, EuroPar'14]:
  - Delegate the execution of the CS to the current holder of the lock (up to a point)
  - First thread that takes lock opens a delegation queue
    - Becomes *helper*
  - Subsequent threads delegate their CSs to the lock holder
    - Option to *detach* CS execution and continue
  - Helper executes all CSs in its queue
    - no lock handoff!
    - no migration of CS data
  - After a number of CSs, helper closes delegation queue and releases the lock

Queue-Delegation Locks

- Hierarchical Queue-Delegation Locking
  - Delegate only locally

While on the same node (core):

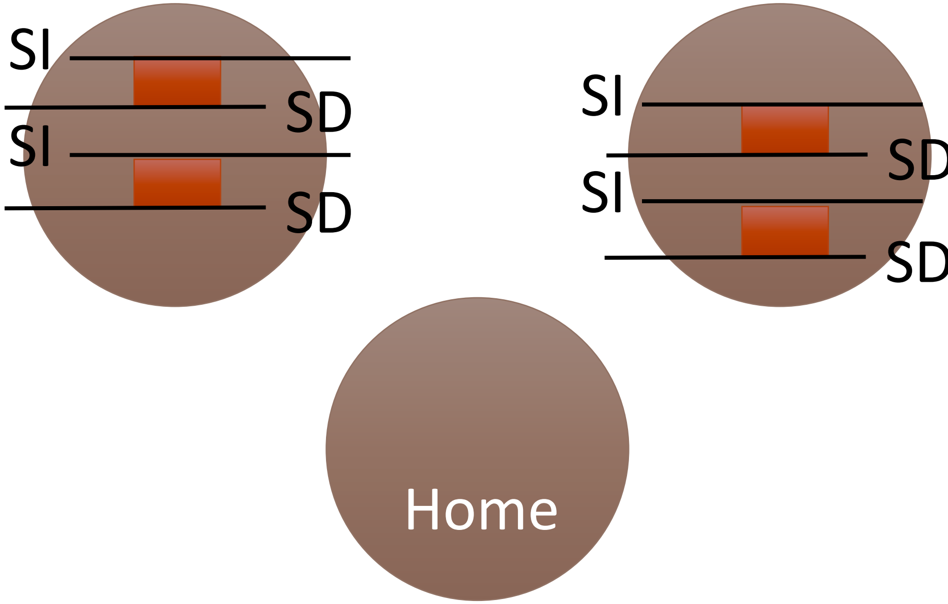
→ No latency between CSs

→ No migration of critical data

→ No SI/SD between CSs

On lock hand-off:

→ Distributed Coherence (SI/SD) needed



Queue-Delegation Locks

- Hierarchical Queue-Delegation Locking
  - Delegate only locally

While on the same node (core):

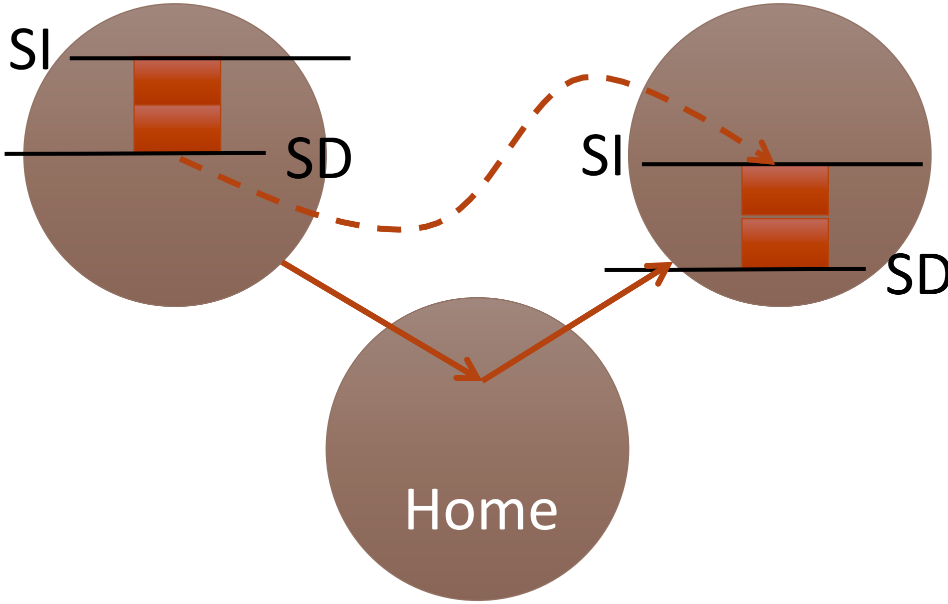
→ No latency between CSs

→ No migration of critical data

→ No SI/SD between CSs

On lock hand-off:

→ Distributed Coherence (SI/SD) needed



ARGO DSM

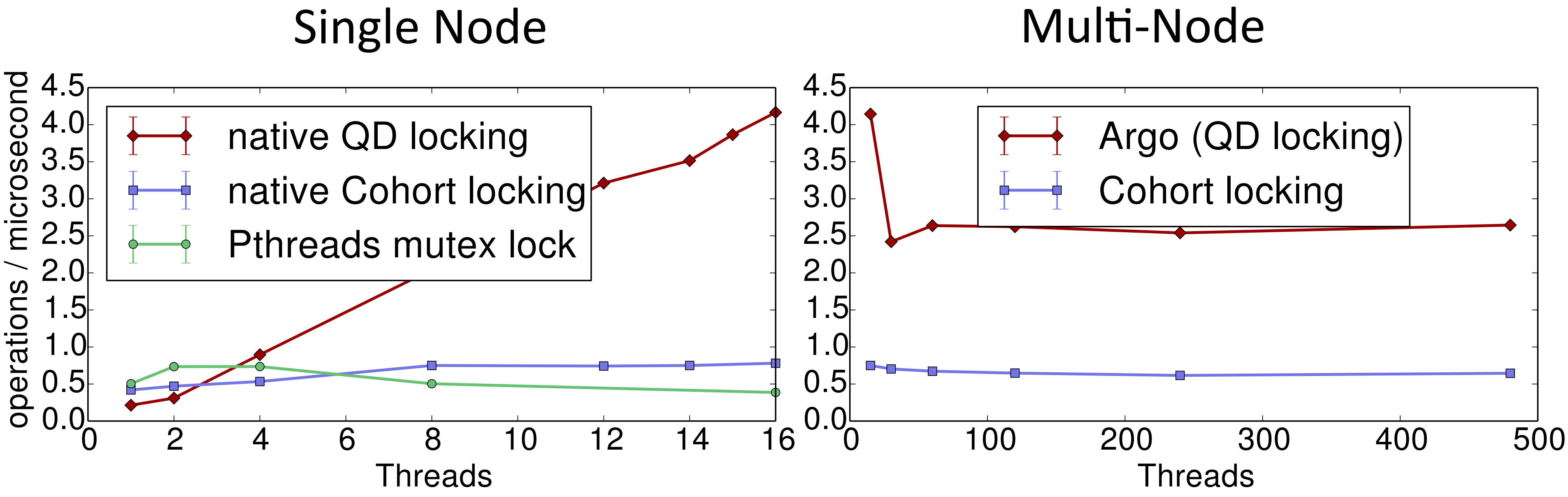
- On the surface a traditional DSM:
  - User-space implementation
  - Page-based DSM (uses virtual memory faults for misses)
  - Pages have a home node (for now: naïve distribution)
  - MPI is the “network layer” (but only need RDMA)
  - Runs Pthreads (DRF programs); Compile and link with Argo library → MPI program that implements DSM (Coming soon: OpenMP)
- Underneath:
  - Carina: Distributed Coherence
  - Pyxis: Classification directories
  - Vela: Hierarchical Queue Delegation Locking system

Evaluation

- 6 Pthreads programs compared to MPI or UPC versions, synchronization microbenchmarks
- University cluster:
  - 32—128 nodes
    - 2x AMD Opteron 6220 (16 cores total) per node
  - 500—2000 threads
  - QDR Infiniband
  - 64GB per node
  - Scientific Linux 6.4

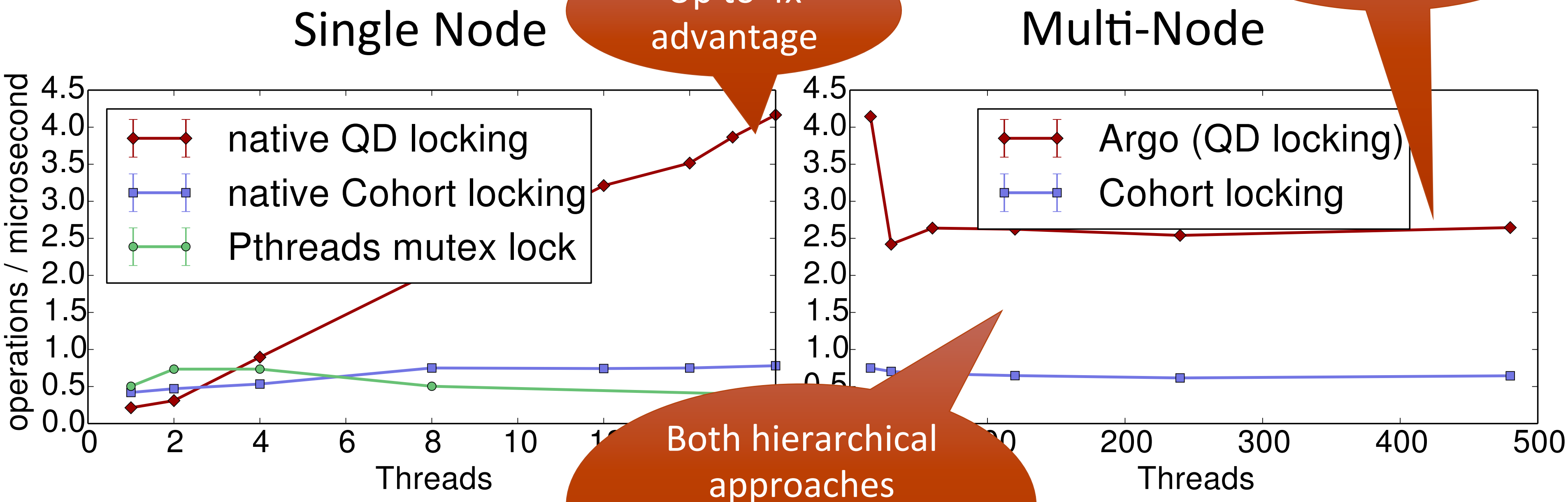
Synchronization performance

- Argo QDL vs. Cohort vs. Pthreads mutex
- Intensive-synchronization microbenchmark: Concurrent Priority Queue (not supposed to scale)



Synchronization performance

- Argo QDL vs. Cohort vs. Pthreads mutex
- Intensive-synchronization microbenchmark: Concurrent Priority Queue (not supposed


Single Node

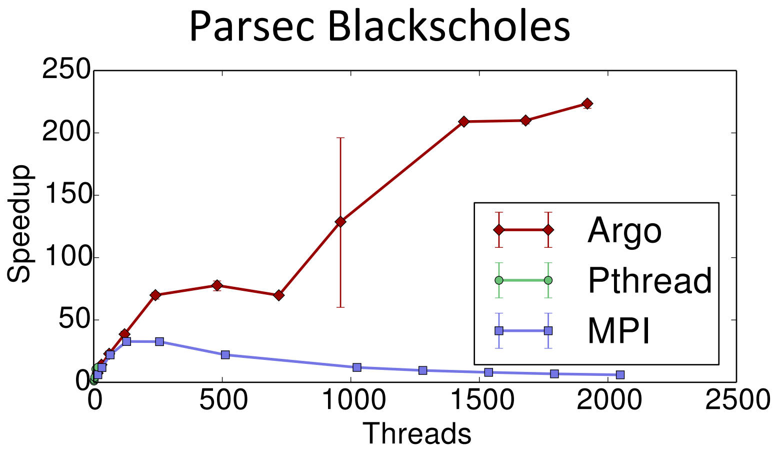
Multi-Node

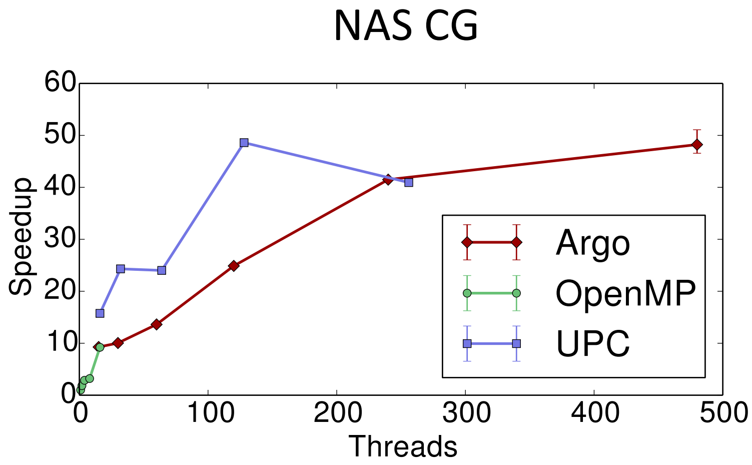
Up to 4x advantage

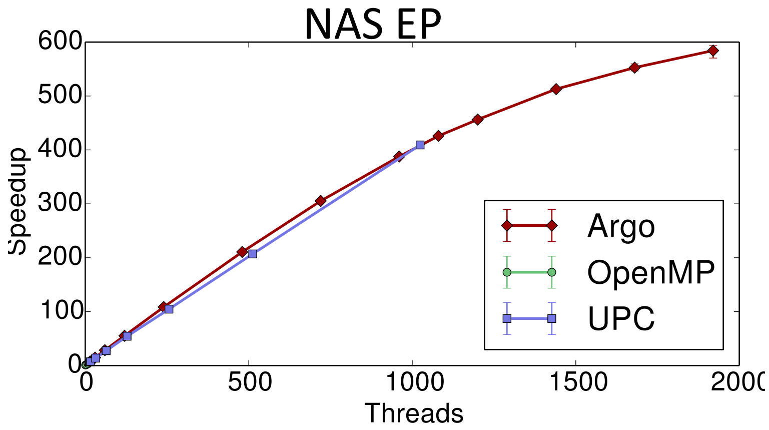
2.5x advantage

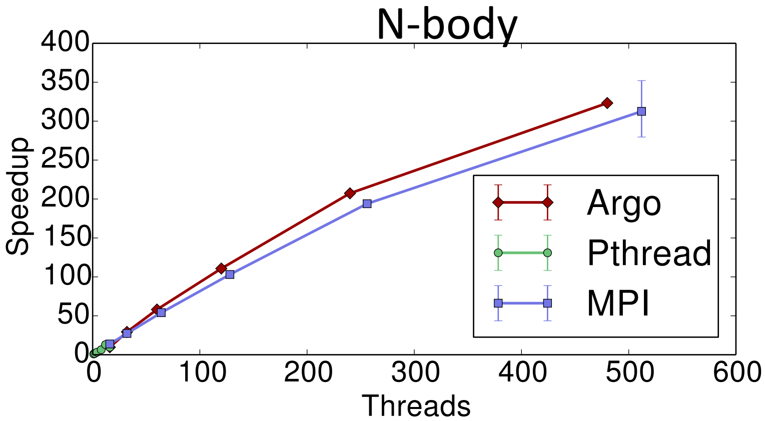
Both hierarchical approaches maintain high throughput

Benchmark scaling

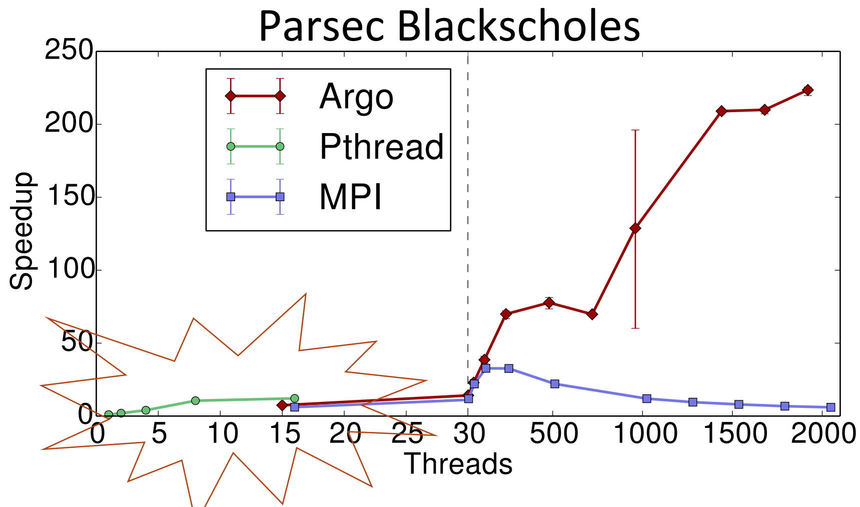

Parsec Blackscholes

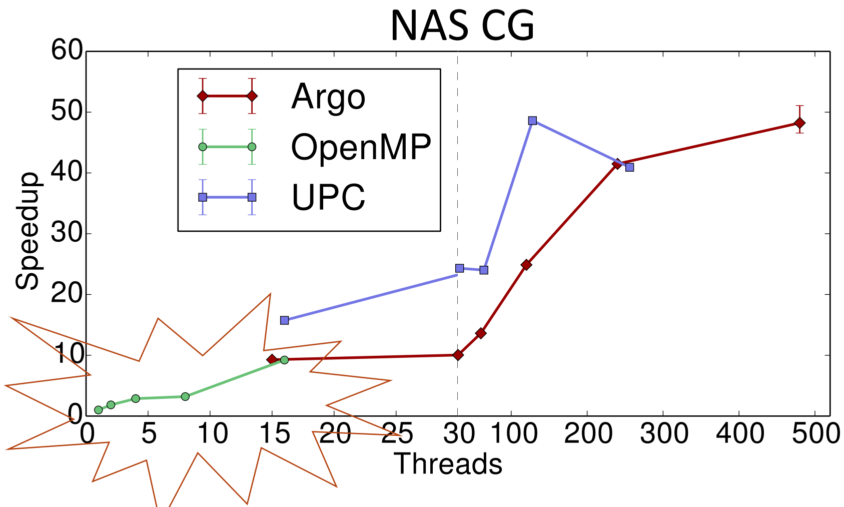

NAS CG

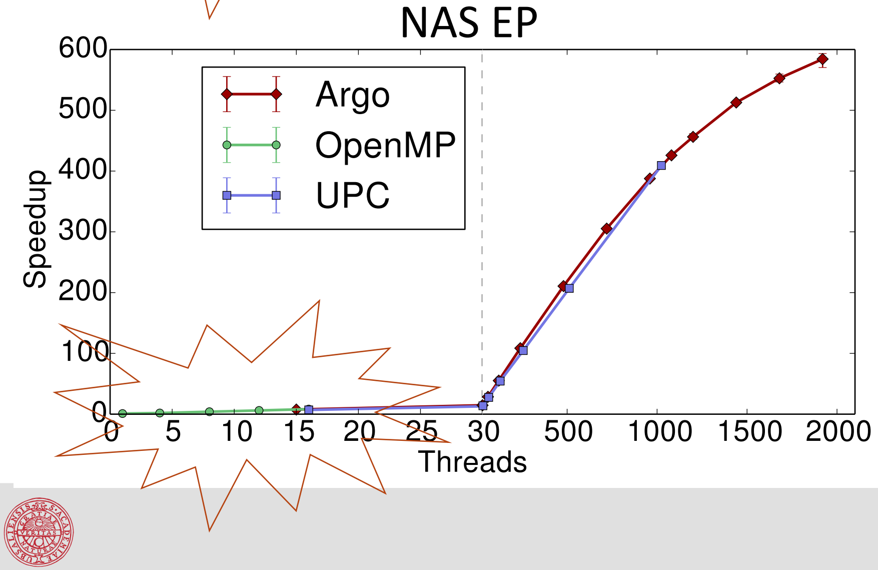

NAS EP

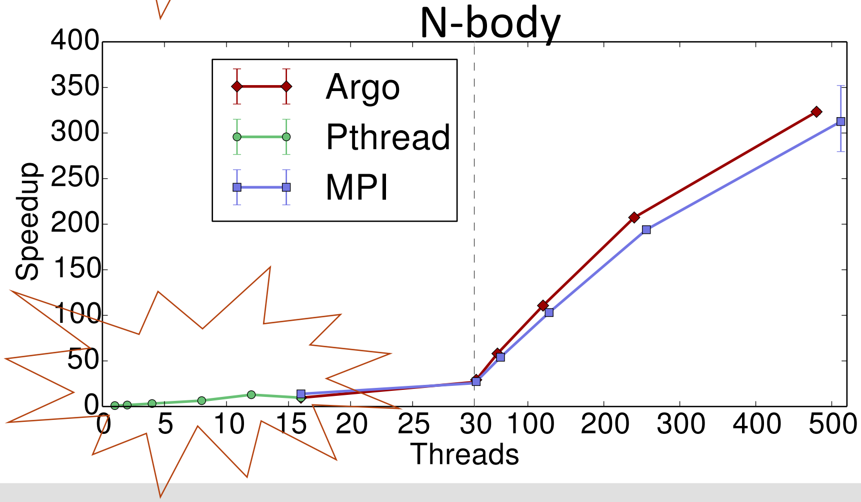

N-body

Benchmark scaling

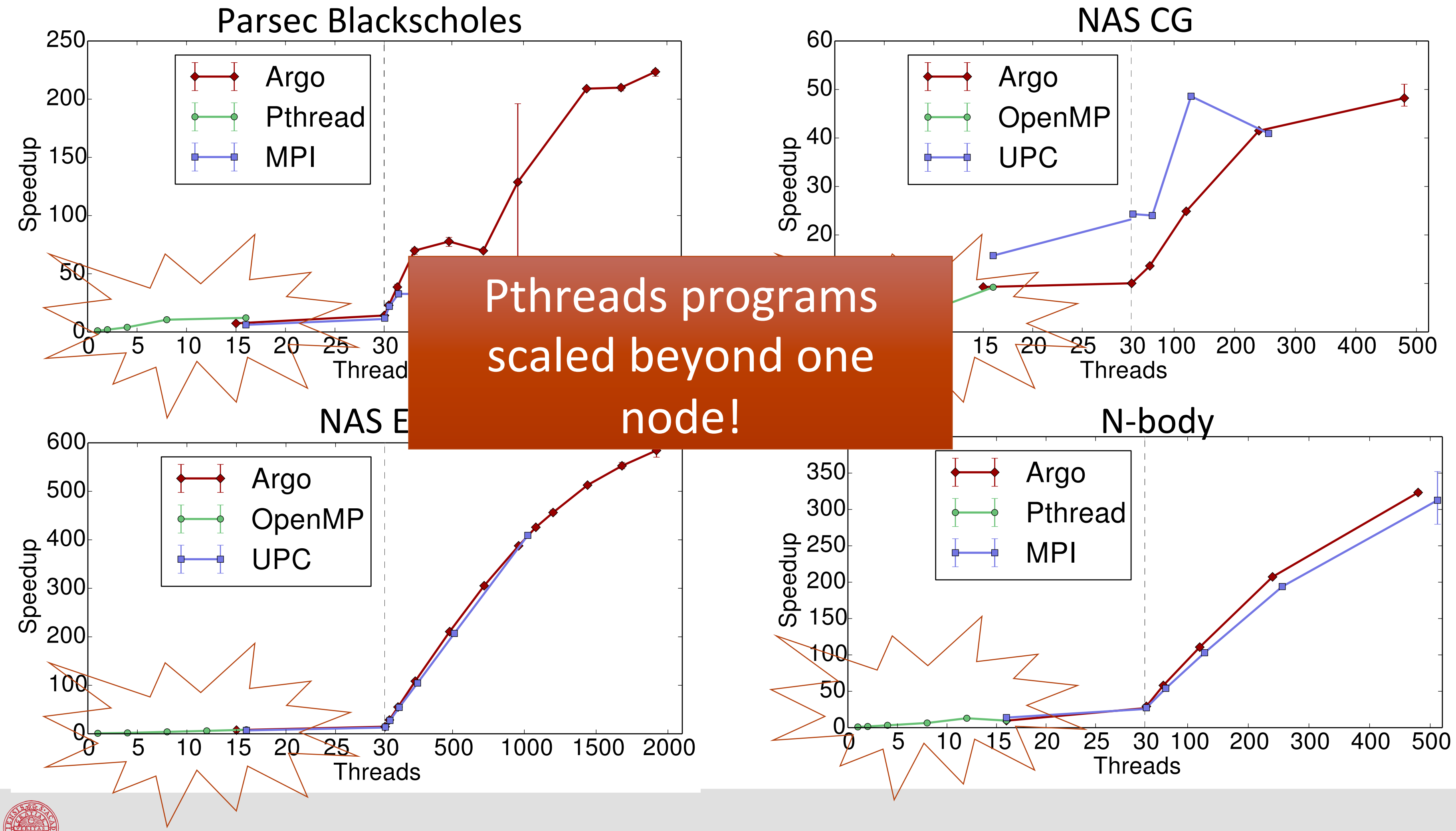

Parsec Blackscholes


NAS CG


NAS EP


N-body

Benchmark scaling



Pthreads programs scaled beyond one node!

Conclusions

Why try again, why now?

- Trade-offs changed in last 2 decades
- New approach to DSM that fits TRENDS
  - Distributed Coherence
  - Centralized CS execution
  - Trade Increasing BW for Reducing Latency
  - Implemented with NO message handlers
- Aim to rekindle interest in DSM for both users (run Pthreads on clusters!) and researchers (new oppotunities!)
- Soon to be released widely (contact us now for prototype versions)!

argo@it.uu.se