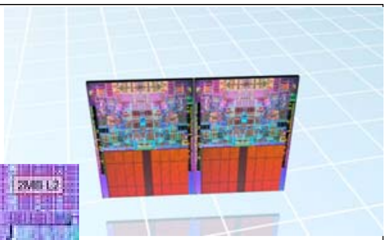
Uncorq:

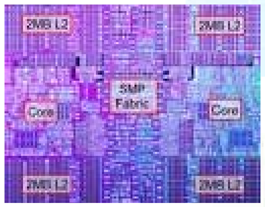
Unconstrained Snoop Request Delivery in Embedded-Ring Multiprocessors

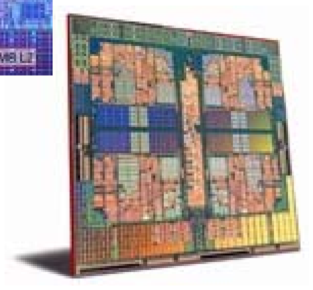
Karin Strauss	AMD Advanced Architecture and Technology Lab
Xiaowei Shen	IBM Research
Josep Torrellas	University of Illinois at Urbana-Champaign

http://iacoma.cs.uiuc.edu

Motivation







- CMPs are ubiquitous
- Shared memory + caches = cache coherence
- Traditional cache coherence solutions
  - shared bus-based: electrical, layout issues
  - directory-based: indirection, storage

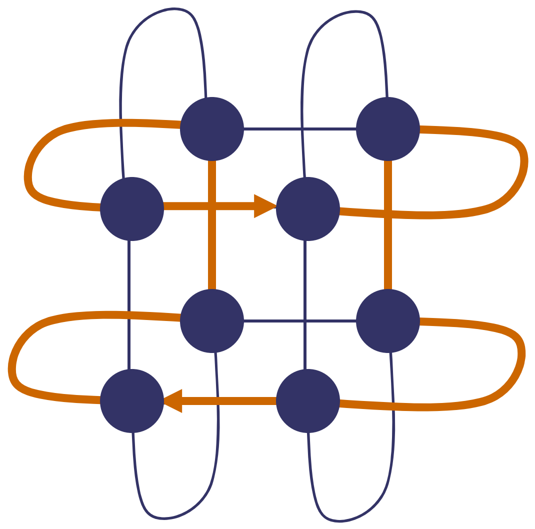
Contributions

- Novel cache coherence scheme (ISCA 2006)
  - Embedded-ring snoopy cache coherence
- Show protocol operation and invariant
  - Transaction serialization
  - Forward progress
- Improve performance
  - Uncorq
  - Optimization: Selective data prefetching
- Evaluate proposal and show it is competitive

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Embedded-ring cache coherence

Embedded-ring cache coherence



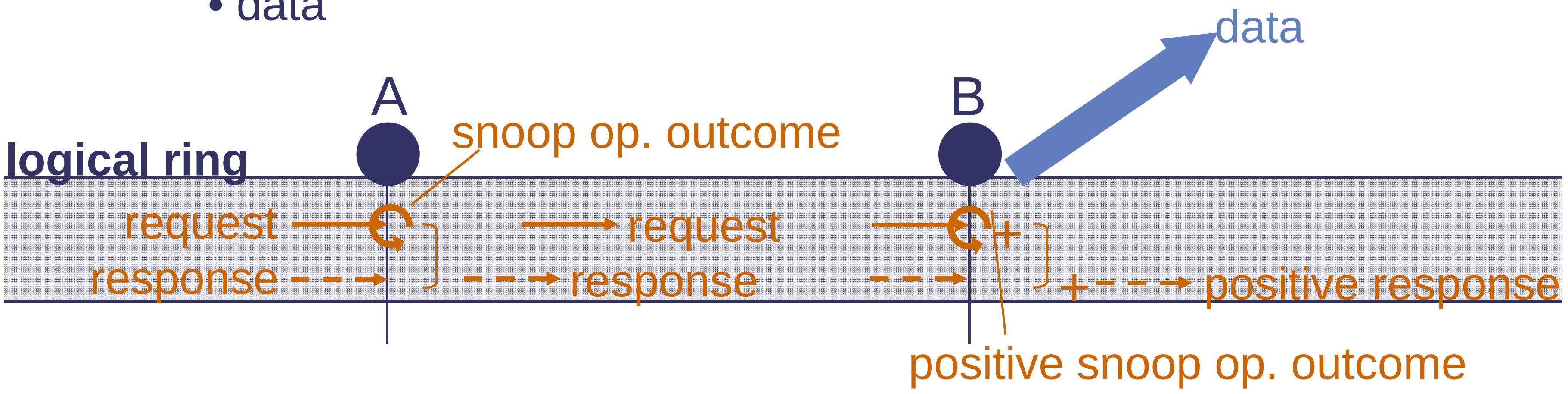
- Logical ring is embedded in network
- Control messages use ring
- Data messages use any path
- Easily reconfigurable
- Simple

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Embedded-ring terminology

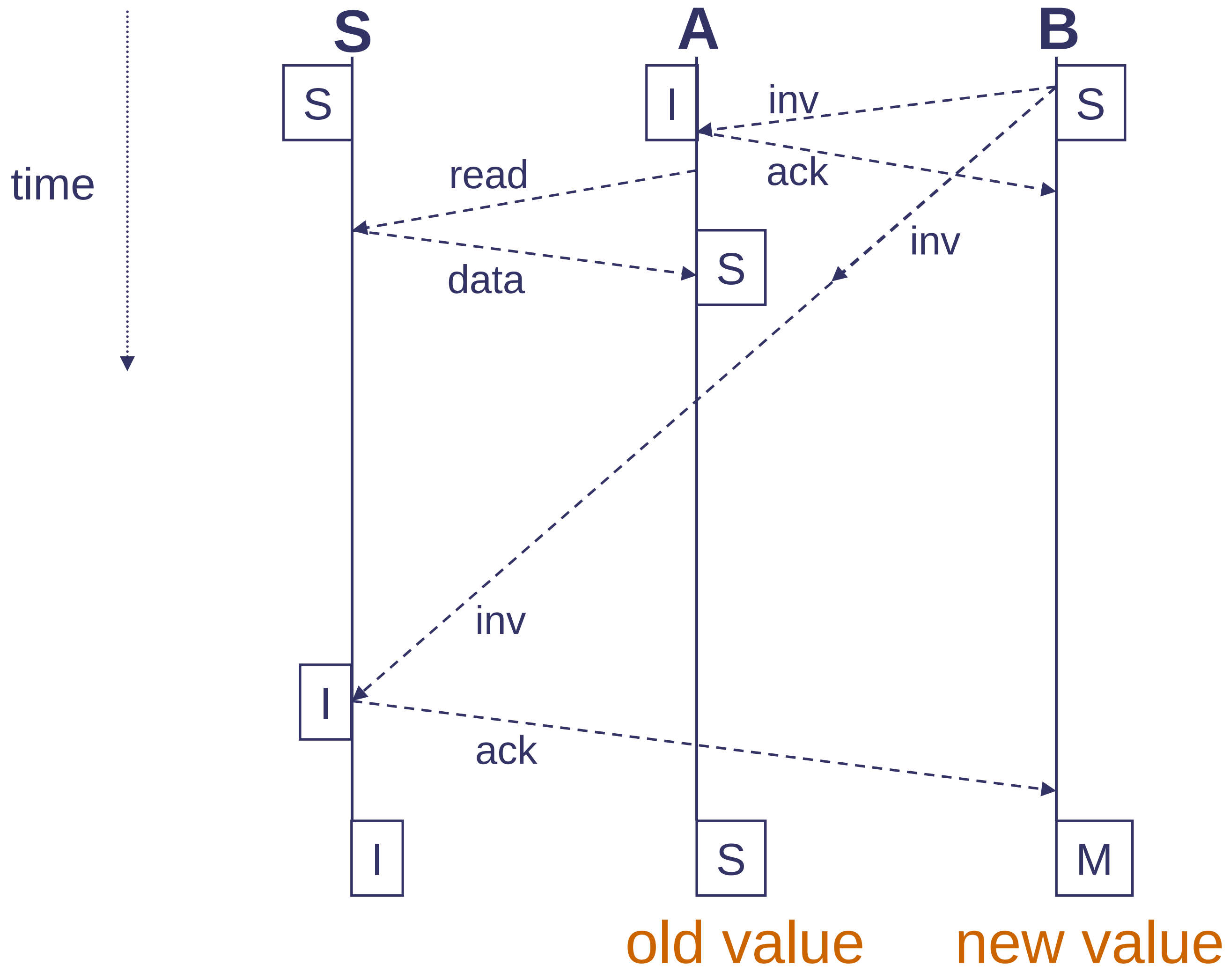
- Broadcast-based snoopy, invalidate protocol
- Single supplier protocol
- Types of messages:
  - request
  - response
  - request + response
  - data

control messages



Protocol operation and invariant

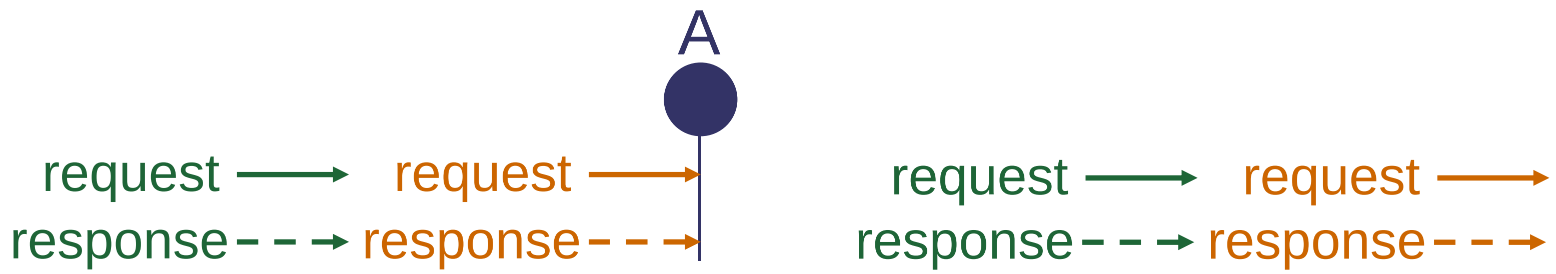
Transaction serialization



QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Transaction serialization on the ring

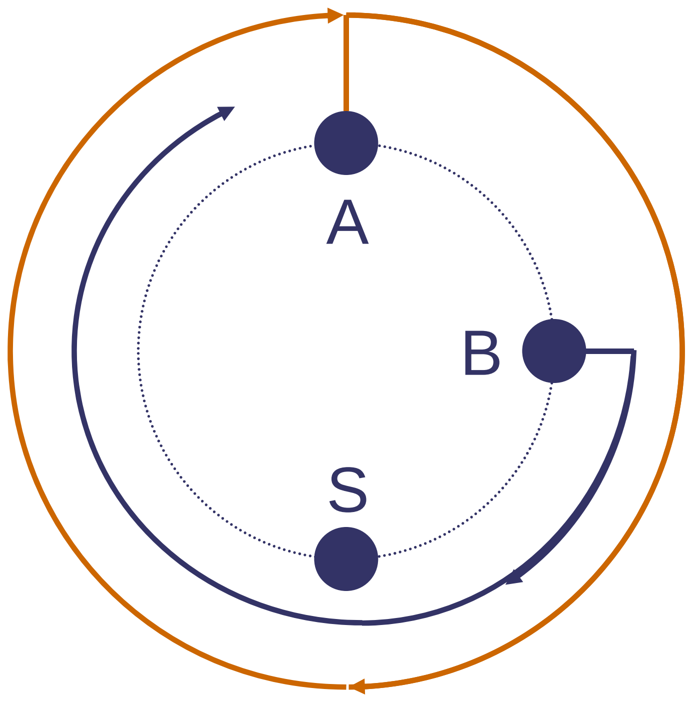
- Single supplier protocol
  - supplier status is transferred when next request is processed
- For same memory location:
  - network links do not reorder messages (requests or responses)
  - nodes process requests in the order they arrive
  - responses travel in the same relative order as their requests



ring provides partial order

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Natural serialization

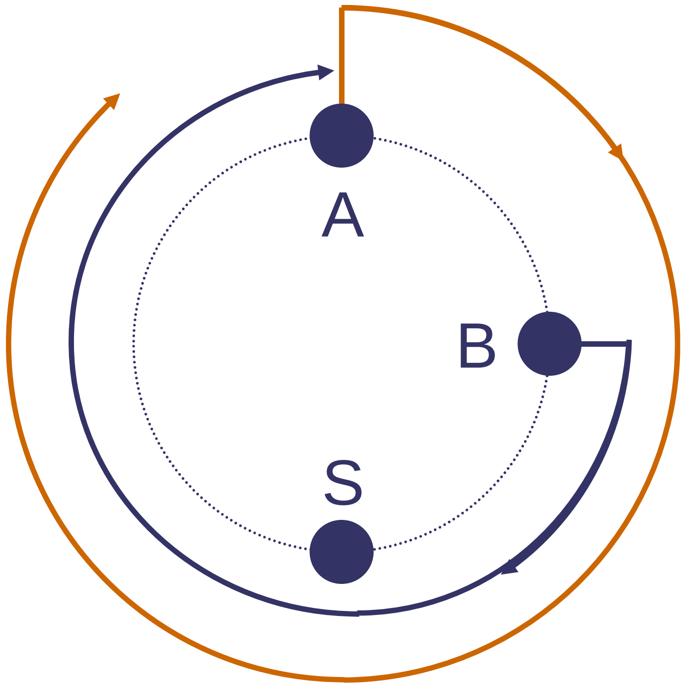


All nodes receive messages in the same order

Natural order is A → B

- A's request and response
- B's request and response

Forced serialization



A's request and response

B's request and response

No clear "first" transaction

B's request reaches S first

Ring guarantees responses are forwarded in the order S performed snoop operations

A receives B's positive response before its own

A retries: B → A

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

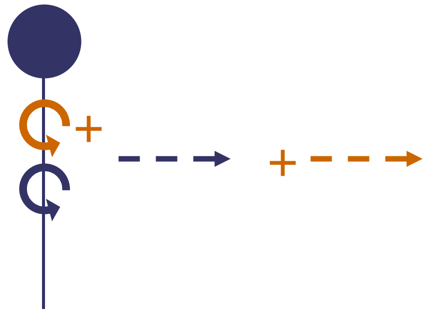
Ordering invariant

Predefined rule: transaction whose request arrives at the node with supplier status first is the "winner"

What we need to maintain to enforce the rule:

Ordering Invariant: the order in which responses travel the ring after leaving the supplier must be the same as the order in which the supplier received and processed their corresponding requests.

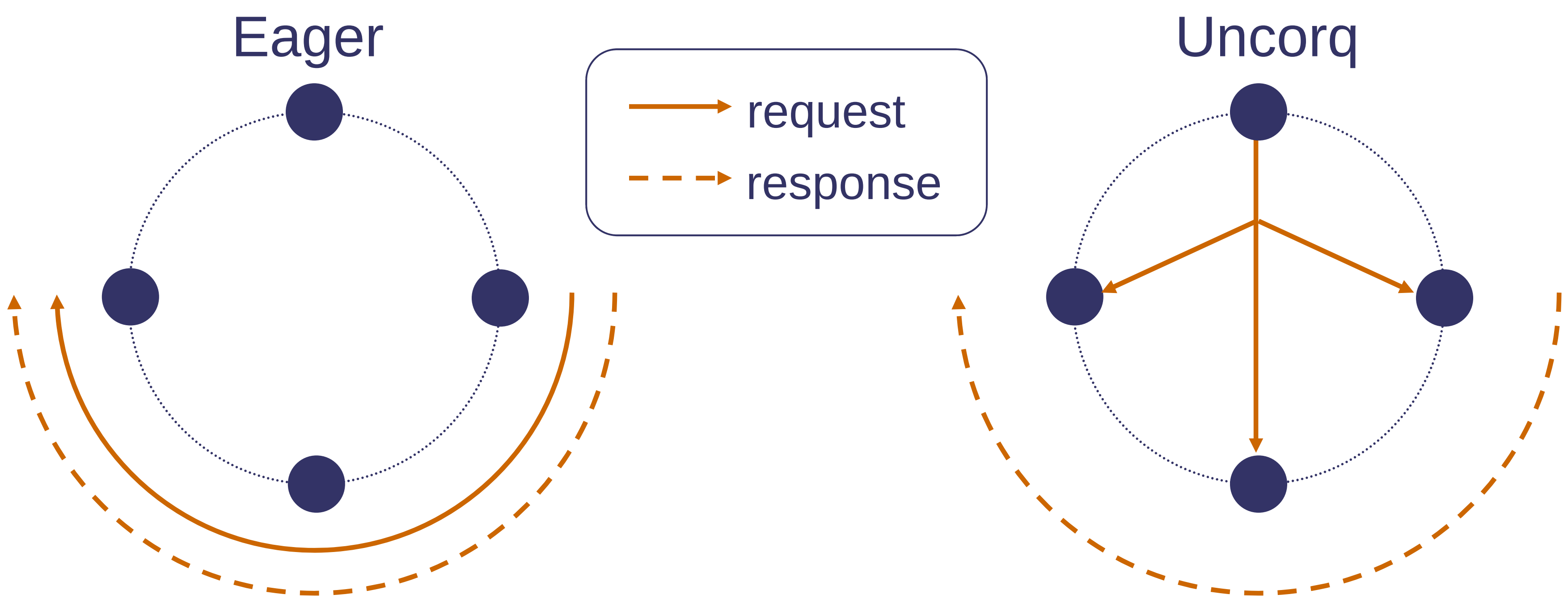
(such that the distributed algorithm has enough information to determine the "winner" throughout the ring)



QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Uncorq: Unconstrained snoop request delivery

Uncorq

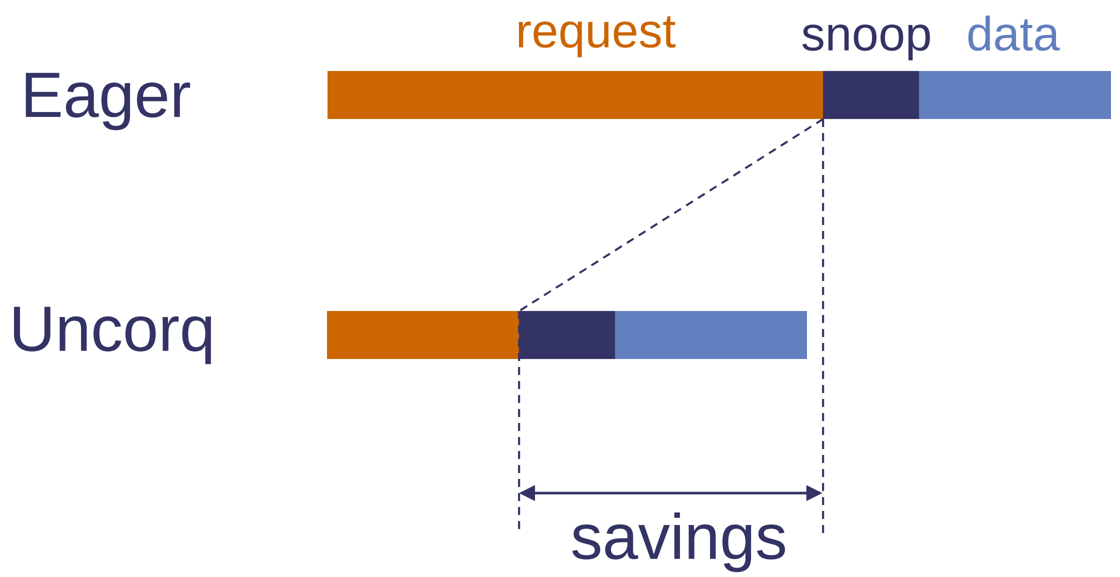


Idea: requests do not have to follow the ring
(but responses do)

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Benefit of Uncorq

Cache-to-cache transfer latency:



QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

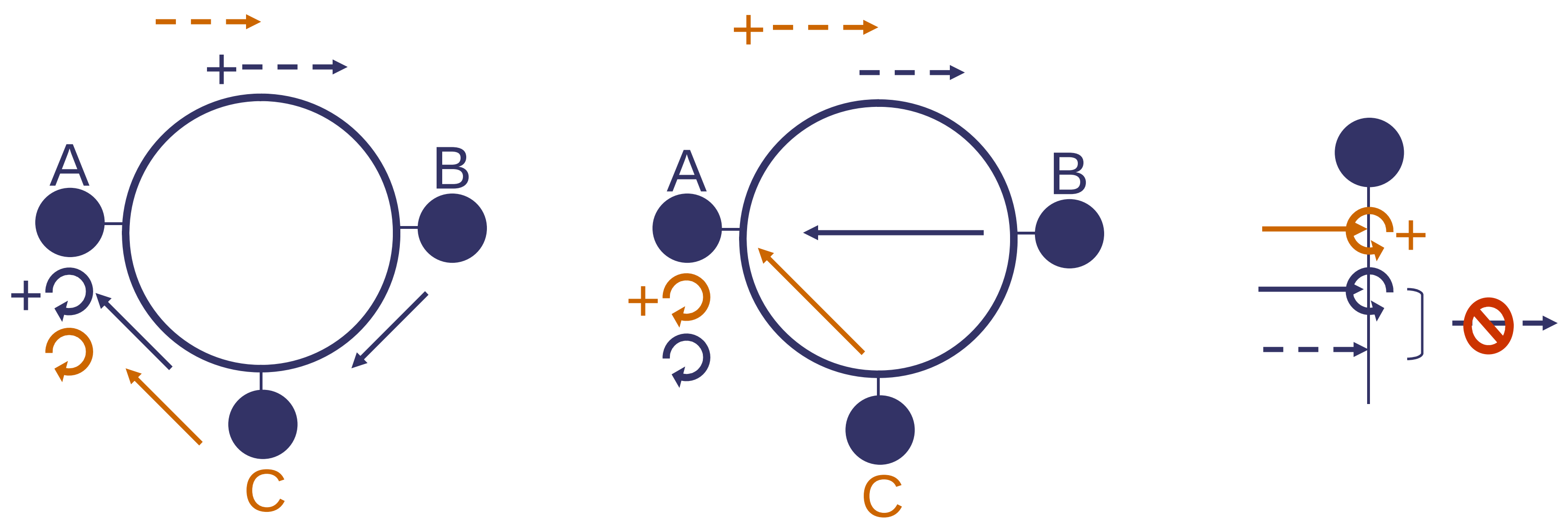
Implications of Uncorq

- Uncorq no longer restricts order of requests
- Nodes may receive and process requests in any order
- Responses may also get reordered

Problem: distributed algorithm relies on the fact that response order reflects order of snoops at supplier, if any

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Uncorq: request reordering



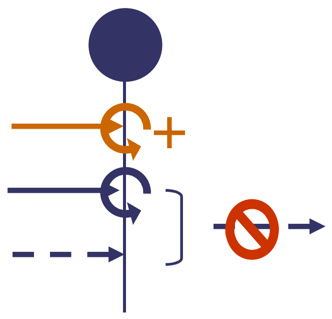
Solution: stall some responses to avoid reordering

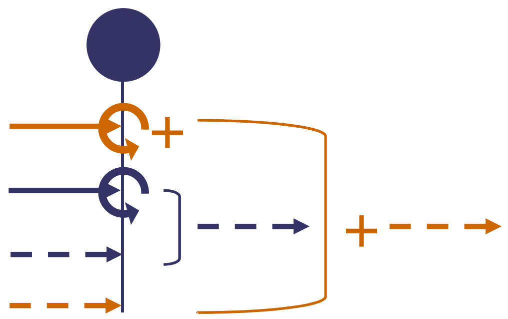
QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Stalling responses

When to stall:

- outstanding positive response waiting in node
- another response (same address) is ready to leave node



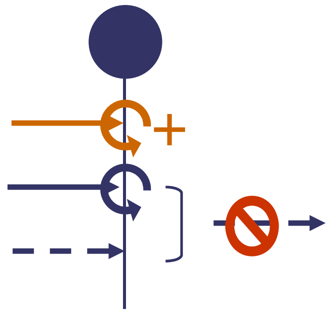


	+	A	B	C	…	
addr	C		✓	✓		snoops
			✓	✓		responses

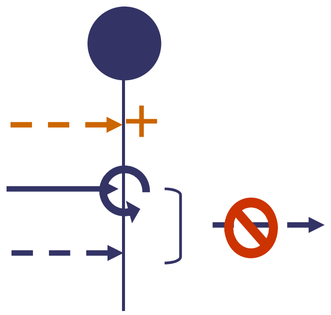
QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Preserving the ordering invariant

- A node cannot forward a response if:
  - it has an outstanding positive snoop outcome



  - it has an outstanding positive response



Optimization: Selective data prefetching

- Accessing memory while snooping may be faster…

  … but wastes energy if done for every miss

- Solution: predict when no node is able to supply data and access memory prematurely only in that case

- Two predictors:
  - node-side predictor
    - records addresses for which it has received requests recently
    - sends requests to memory-side predictor if address is not present
  - memory-side predictor
    - records which lines have been brought on-chip recently
    - sends request to memory if line has not been recently touched

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Evaluation

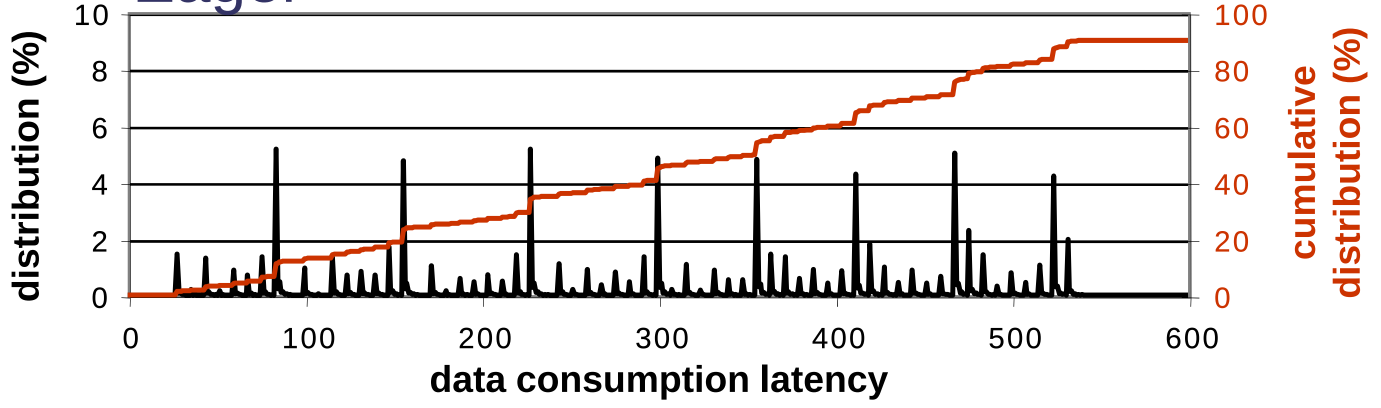
Experimental setup

- SPLASH-2, SPECjbb and SPECweb workloads
- SESC simulator (sesc.sourceforge.net)
- Single-CMP, 64 nodes, each node with DL1, IL1 and L2
- Interconnection network: 2D torus with embedded-ring

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

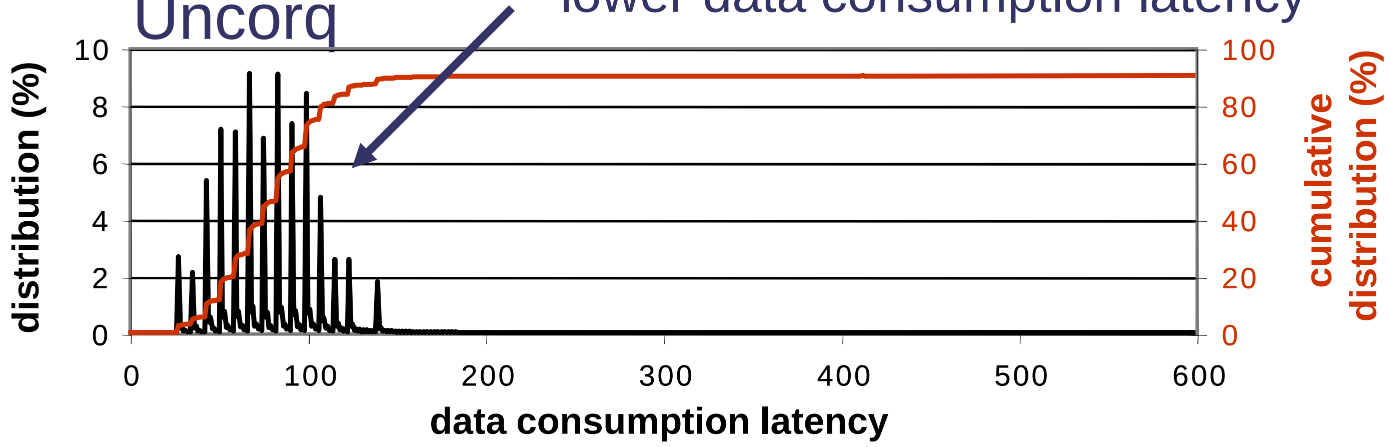
UncoRq: data consumption latency

Eager



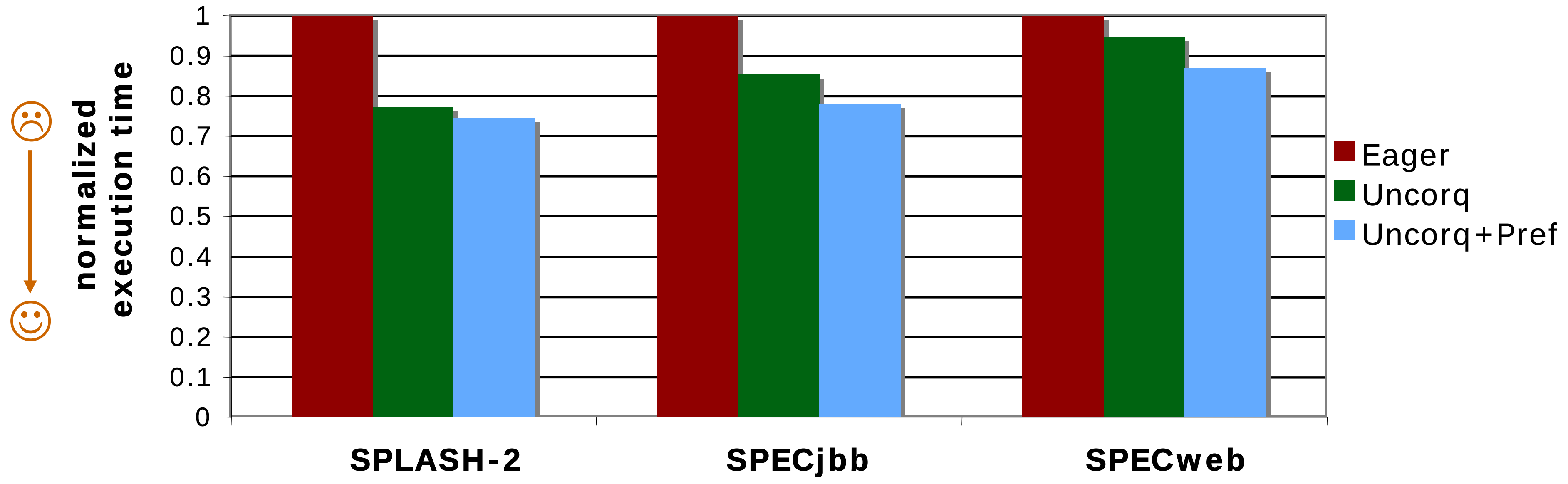
Uncorq

lower data consumption latency



QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Execution Time



- Uncorq significantly reduces execution time
- Uncorq + Pref performs the best

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Also in the paper

- Serialization mechanism for case with no supplier
- System and node forward progress
- Fences and memory consistency issues
- Characterization of prefetching mechanism
- Comparison against ccHyperTransport

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Conclusion

- Show protocol operation and invariants
  - Transaction serialization
  - Forward progress
- Improve performance
  - Uncorq
  - Optimization: Selective data prefetching
- Evaluate proposal and show it is competitive

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Uncorq:

Unconstrained Snoop Request Delivery in Embedded-Ring Multiprocessors

Karin Strauss, Xiaowei Shen*, Josep Torrellas

University of Illinois at Urbana-Champaign

*IBM Research

http://iacoma.cs.uiuc.edu

Question: single supplier?

- Simple!
- Destination-set prediction [Martin'03] can be seamlessly used to avoid performance penalty
- Proper thread scheduling can also help

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Question: why c2c transfers?

- On-chip caches are growing
- Parallel programs are growing

- More on-chip cache sharing

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Question: slow m2c transfers

- Latency depends on ring size
- Can be attenuated with:
  - private data predictors
  - prefetching
  - speculation
  - smaller, partitioned rings

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Question: Latency of writes

- Problem for stricter consistency models
- We assume a weaker consistency model (PowerPC)
  - requires large write buffers
  - does not prevent instructions from retiring
    - unless fences are present

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Question: Latency of fences

- Problem: fences need to wait until all previous writes complete
- Data can be provided before write completes (lock changes hands quickly)
- Speculation across fences solves the problem (Wait-free multiprocessors, SC++, BulkSC)

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

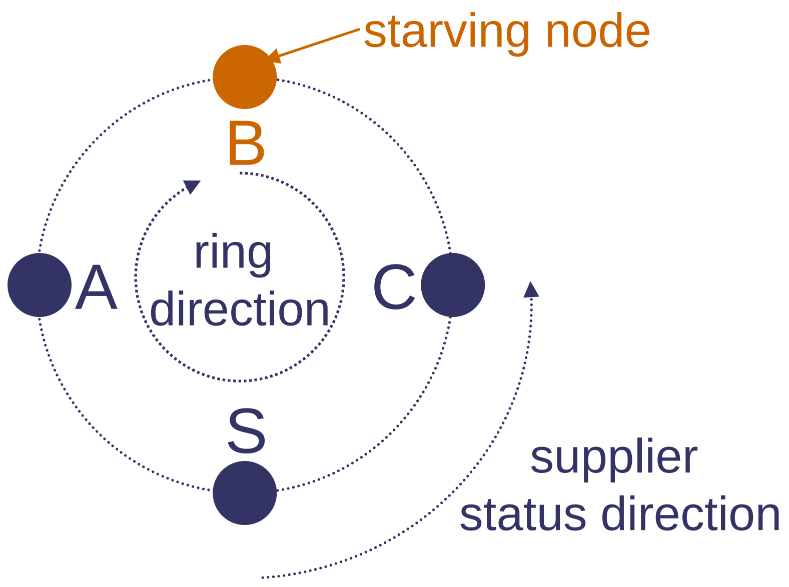
Question: Token coherence?

- Embedded-ring and token coherence can be combined
  - Multicast request
  - Data is sent directly
  - Collect tokens with ring

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Forward progress

- System forward progress: there is always one winner
- Node forward progress: also known as starvation-freedom
  - starving nodes can intercept many requests, but not all



eventually, B will be right next to node with supplier status

when this happens, B can intercept all requests and finally complete its transaction

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Starvation-freedom with UncoRq

Problem: UncoRq no longer restricts requests to the ring

⇩

Starving node can no longer intercept requests

⇩

Solution: intercept responses and rely on LTT mechanism

- starving node records its own id on any response it receives
- when “winner” node receives its response back,
  it sets its DRN to the starving node’s ID
- starving node can then safely complete its transaction

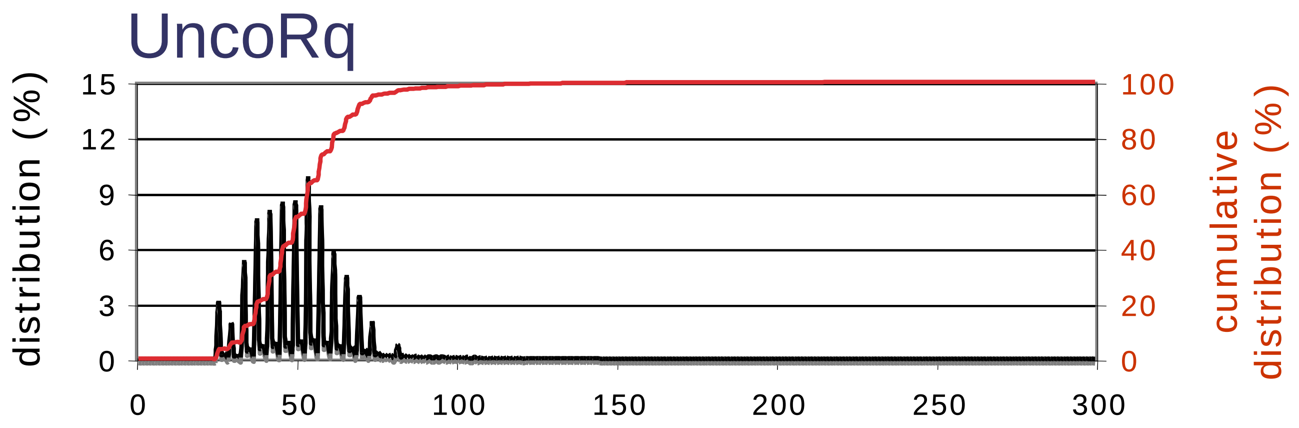
QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

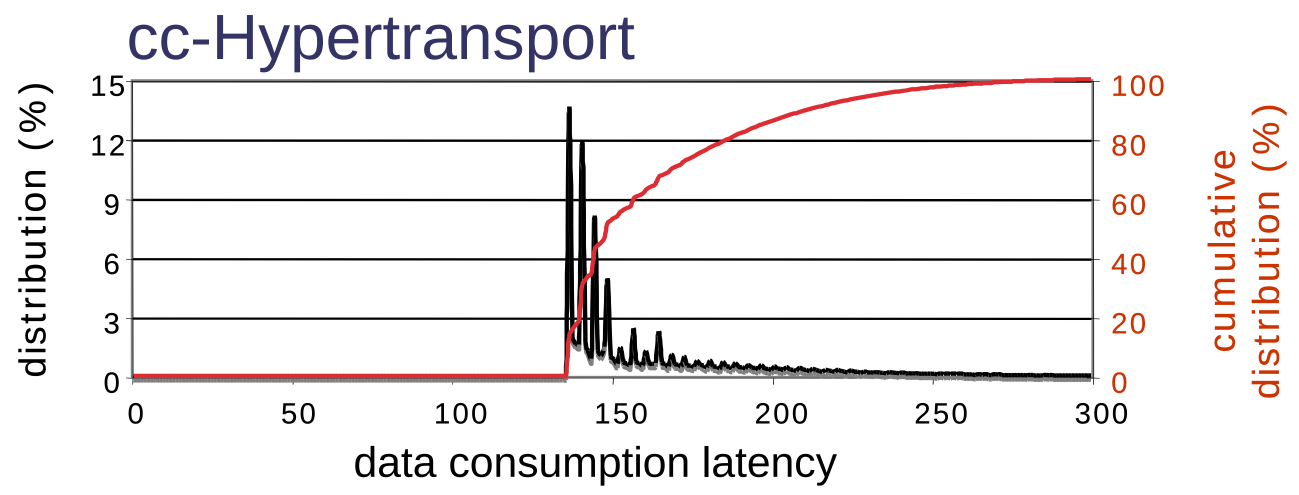
Distributed arbitration algorithm

- Two different situations:
  - there is a node with supplier status
    - node whose request gets to supplier first is the “winner”
  - there is no node with supplier status
    1. if one of them is invalidate transaction, it is the “winner”
    2. if one of them is read transaction, it is the “loser”
    3. node whose ID is the lowest is the “winner”

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

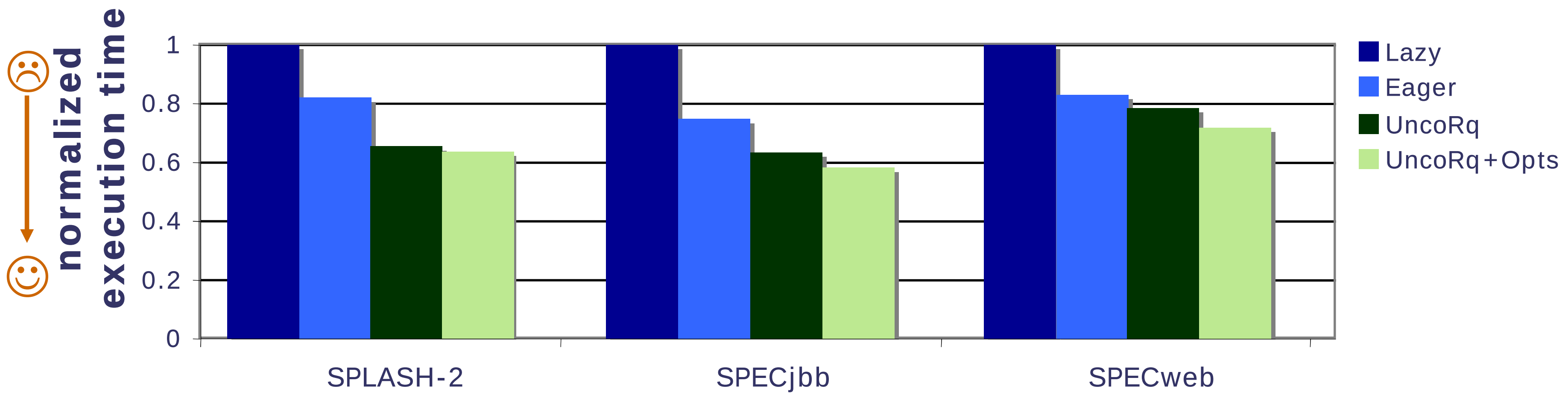
Uncorq vs cc-Hypertransport





QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

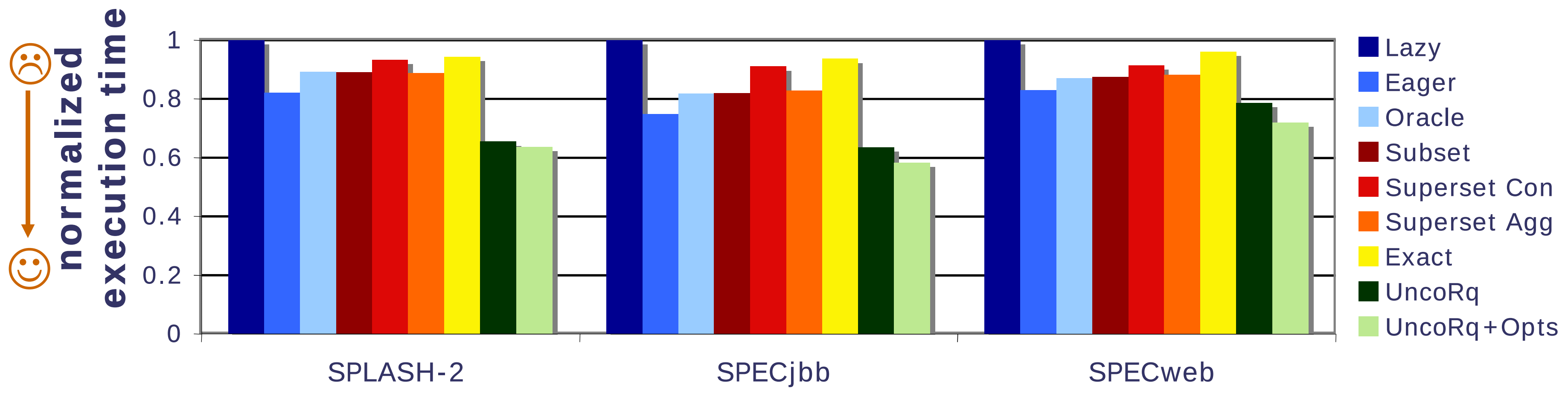
Execution time



- UncoRq + Opts performs the best

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

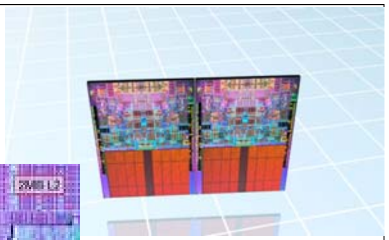
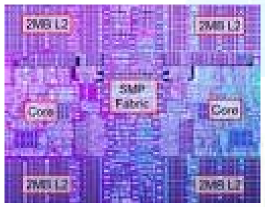
Execution time

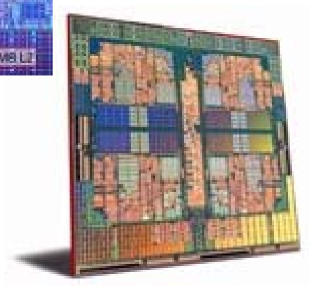


- UncoRq + Opts performs the best
- Flexible Snooping algorithms do not perform as well

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Motivation

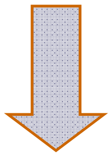





CMPs are ubiquitous

cheaper to build medium-size machines

shared memory + caches



cache coherence

shared bus-based

electrical, layout issues

directory-based

indirection, storage

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Conclusion

Show embedded-ring operation and invariants

- ➢ Transaction serialization
- ➢ Forward progress

Improve performance

- ➢ Uncorq
- ➢ Memory access prediction

Evaluate proposal and show it is competitive

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Implications of Uncorq

- Uncorq no longer restricts order of requests
- Nodes may receive and process requests in any order
- Responses may also get reordered

Problem: distributed algorithm relies on the fact that response order reflects order of snoops at supplier, if any

Solution: stall some responses to avoid reordering

QuickTime™ and a TIFF (Uncompressed) decompressor are needed to see this picture.

Transaction serialization on the ring II

- Ring partial order may avoid conflicts: natural serialization
- Some conflicts cannot be avoided: forced serialization
- Distributed algorithm uses partial order to resolve conflicts
  - one transaction is determined to be the “winner”
  - other transactions may have to retry
- Two different situations:
  - there is a node with supplier status
    - node whose request reaches the supplier first is the “winner”
  - there is no node with supplier status
    - need another strategy to pick a “winner” (in the paper)

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.