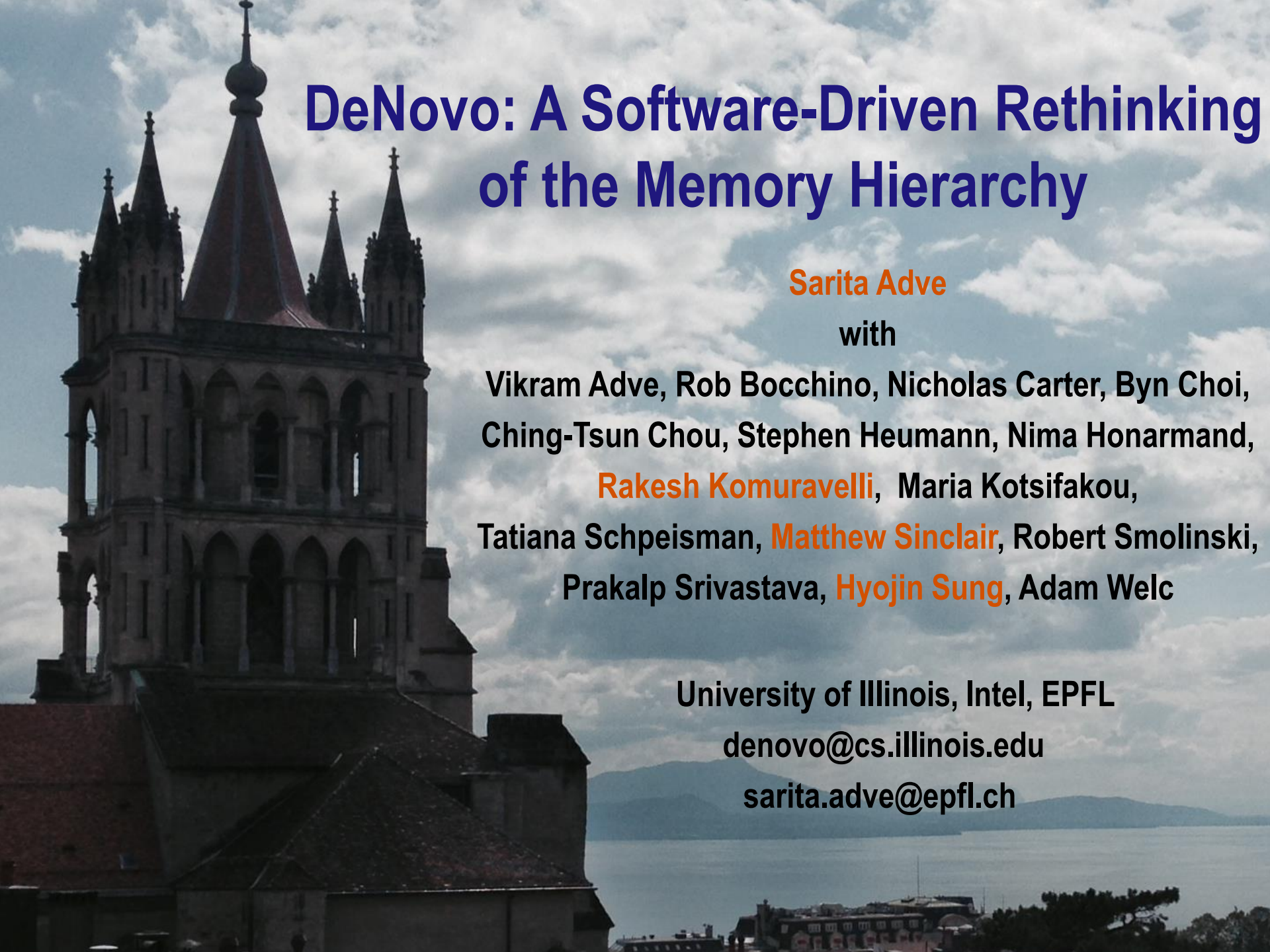


DeNovo: A Software-Driven Rethinking of the Memory Hierarchy

Sarita Adve

with

Vikram Adve, Rob Bocchino, Nicholas Carter, Byn Choi, Ching-Tsun Chou, Stephen Heumann, Nima Honarmand, Rakesh Komuravelli, Maria Kotsifakou, Tatiana Schpeisman, Matthew Sinclair, Robert Smolinski, Prakalp Srivastava, Hyojin Sung, Adam Welc

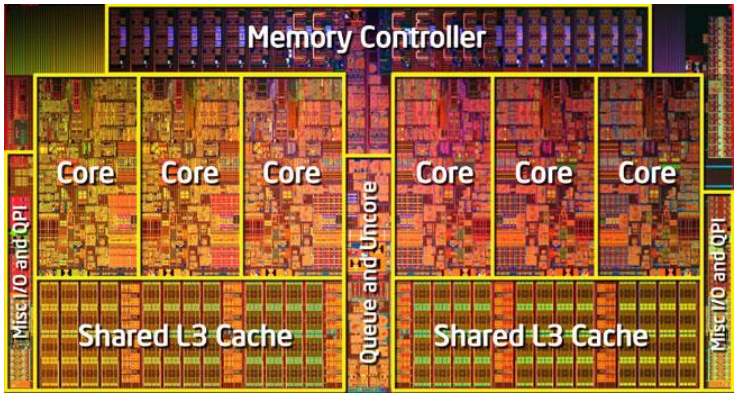
University of Illinois, Intel, EPFL

denovo@cs.illinois.edu

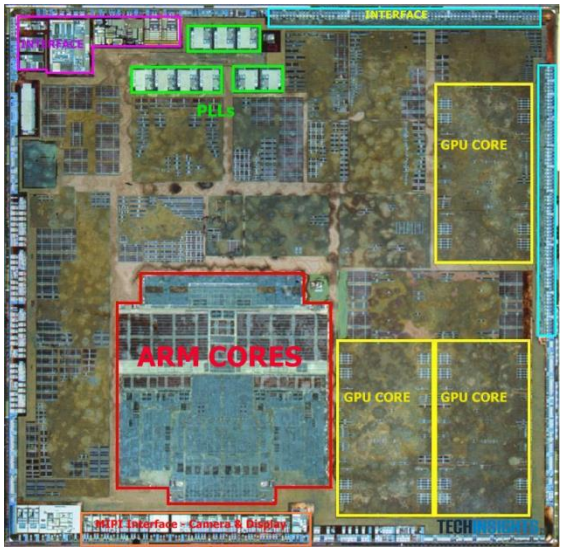
sarita.adve@epfl.ch

Silver Bullets for the Energy Crisis?

Parallelism



Specialization, heterogeneity, …



BUT large impact on

- Software
- Hardware
- Hardware-Software Interface

Multicore Parallelism: Current Practice

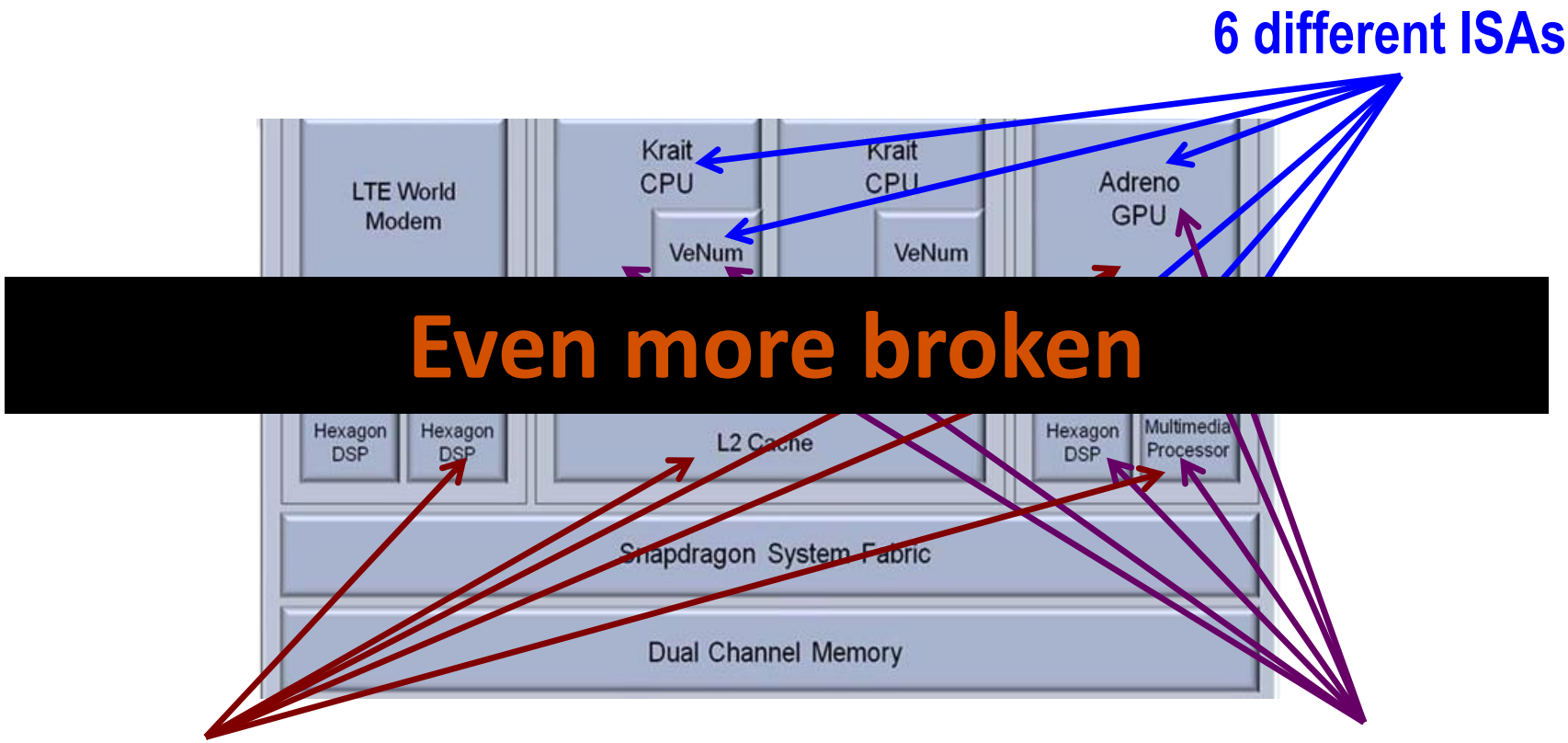
- Multicore parallelism today: shared-memory
  - Complexity-, power-, and performance-inefficient hardware
    - Complex directory coherence, unnecessary traffic, ...
  - Difficult programming model
    - Data races, non-determinism, composability?, testing?
  - Mismatched interface between HW and SW, a.k.a memory model
    - Can't specify "what value can read return"
    - Data races defy acceptable semantics

Fundamentally broken for hardware & software

Specialization/Heterogeneity: Current Practice

A modern smartphone

CPU, GPU, DSP, Vector Units, Multimedia, Audio-Video accelerators



Energy Crisis Demands Rethinking HW, SW

How to (co-)design

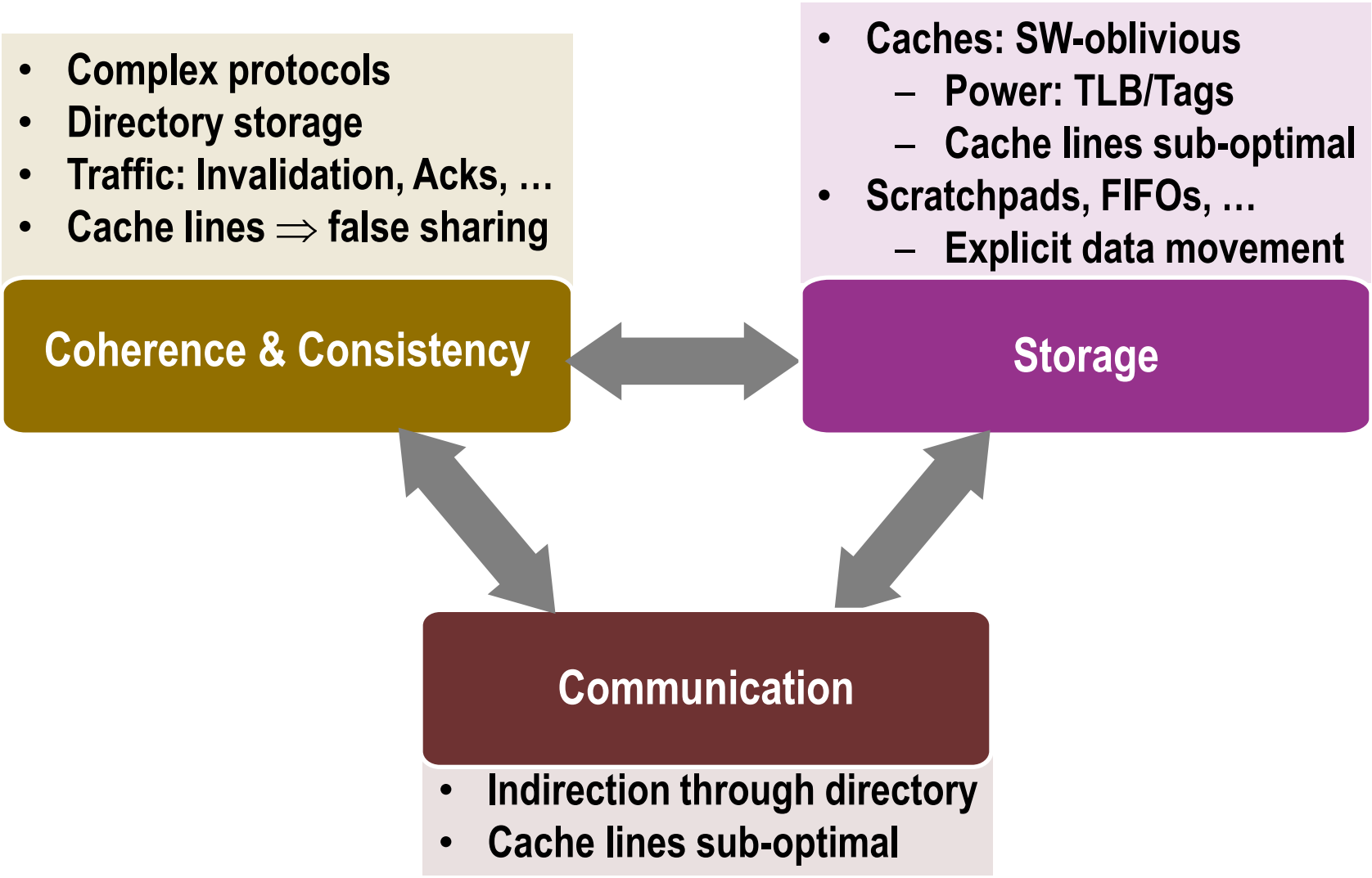
- **Software?** *Deterministic Parallel Java (DPJ)*
- **Hardware?** *DeNovo*
- **HW / SW Interface?** *Virtual Instruction Set Computing (VISC)*

Other implications of energy crisis (another talk)

SWAT: SoftWare Anomaly Treatment

A software-driven approach to hardware reliability

Memory Hierarchy Inefficiencies



Complexity-, power-, and performance-inefficient hardware

Memory Hierarchy Inefficiencies

- Complex protocols
- Directory storage
- Traffic: Invalidation, Acks, …
- Cache lines ⇒ false sharing

- Caches: SW-oblivious
  - Power: TLB/Tags
  - Cache lines sub-optimal
- Scratchpads, FIFOs, …
  - Explicit data movement

Banish shared memory?

Communication

- Indirection through directory
- Cache lines sub-optimal

Complexity-, power-, and performance-inefficient hardware

Memory Hierarchy Inefficiencies

- Complex protocols
- Directory storage
- Traffic: Invalidation, Acks, …
- Cache lines ⇒ false sharing

- Caches: SW-oblivious
  - Power: TLB/Tags
  - Cache lines sub-optimal
- Scratchpads, FIFOs, …
  - Explicit data movement

Banish wild shared memory!

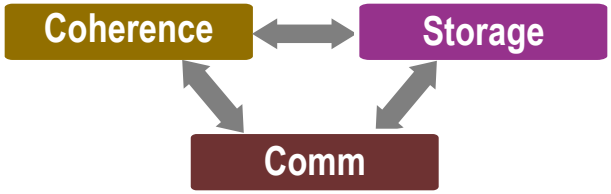
Need disciplined shared memory!

Communication

- Indirection through directory
- Cache lines sub-optimal

Complexity-, power-, and performance-inefficient hardware

What is Shared-Memory?



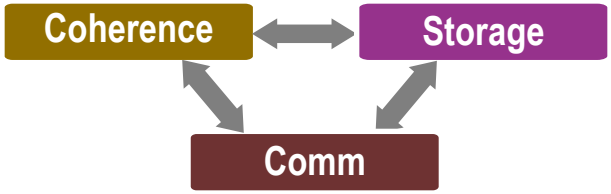
Shared-Memory =

Global address space

+

Implicit, anywhere communication, synchronization

What is Shared-Memory?



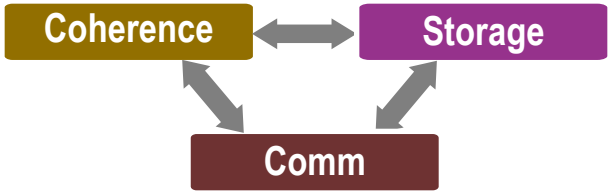
Shared-Memory =

Global address space

+

Implicit, anywhere communication, synchronization

What is Shared-Memory?



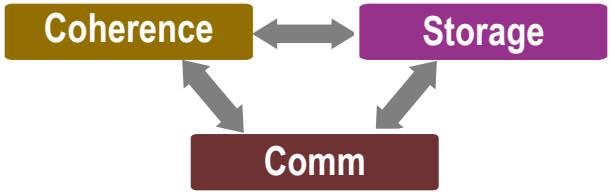
Wild Shared-Memory =

Global address space

+

Implicit, anywhere communication, synchronization

What is Shared-Memory?



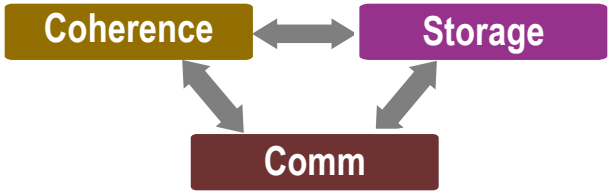
Wild Shared-Memory =

Global address space

+

~~**Implicit, anywhere communication, synchronization**~~

What is Shared-Memory?



Disciplined Shared-Memory =

Global address space

+

~~**Implicit, anywhere communication, synchronization**~~

Explicit, structured side-effects

How to build disciplined shared-memory software?

If software is more disciplined, can hardware be more efficient?

The DeNovo Approach

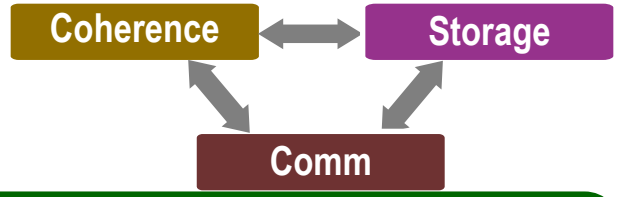
Software

DPJ: Determinism
OOPSLA'09

Disciplined non-determinism
POPL'11

Unstructured synchronization
⇒ OS, Legacy

Hardware



DeNovo: Coherence, Comm
PACT'11 best paper, TACO'14

DeNovoND
ASPLOS'13, Top picks'14

DeNovoSync (in review)

DeNovoH for heterogeneous systems: Coherence, Comm, Storage

Stash: Have your scratchpad and cache it too (in review)

Coherence/Communication Inefficiencies

- **Complexity**
  - Subtle races and numerous transient states in the protocol
  - Hard to verify and extend for optimizations
- **Storage overhead**
  - Directory overhead for sharer lists
- **Performance and power inefficiencies**
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Network traffic (cache-line based communication)

Results for Deterministic Codes

- **Complexity**
  - No transient states
  - Simple to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

- **Storage overhead**
  - Directory overhead for sharer lists

- **Performance and power inefficiencies**
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Network traffic (cache-line based communication)

Results for Deterministic Codes

- **Complexity**
  - No transient states
  - Simple to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

- **Storage overhead**
  - No storage overhead for directory information

- **Performance and power inefficiencies**
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Network traffic (cache-line based communication)

Results for Deterministic Codes

- **Complexity**
  - No transient states
  - Simple to extend for optimizations

Base DeNovo
20X faster to verify vs. MESI

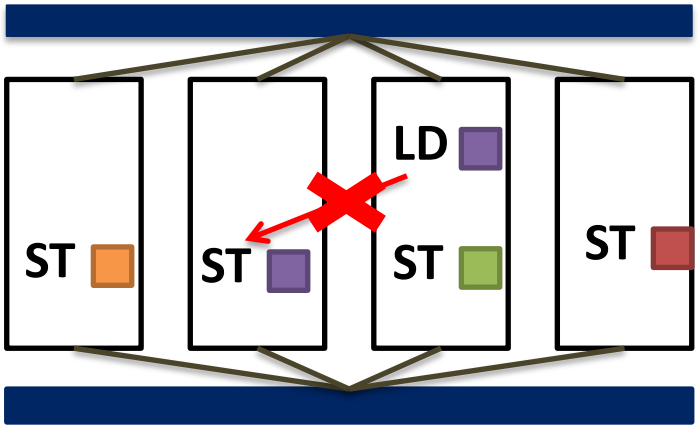
- **Storage overhead**
  - No storage overhead for directory information

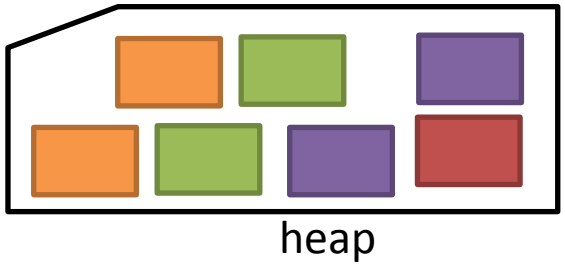
- **Performance and power ineffic**
  - No invalidation, ack messages
  - No indirection through director
  - No false sharing: region based coherence

Up to 77% lower memory stall time
Up to 71% lower traffic

Deterministic Parallel Java (DPJ) Overview

- **Structured parallel control**
  - Fork-join parallelism
- **Region: name for set of memory locations**
  - Assign region to each field, array cell
- **Effect: read or write on a region**
  - Summarize effects of method bodies
- **Compiler: simple type check**
  - Region types consistent
  - Effect summaries correct
  - Parallel tasks don't interfere (race-free)

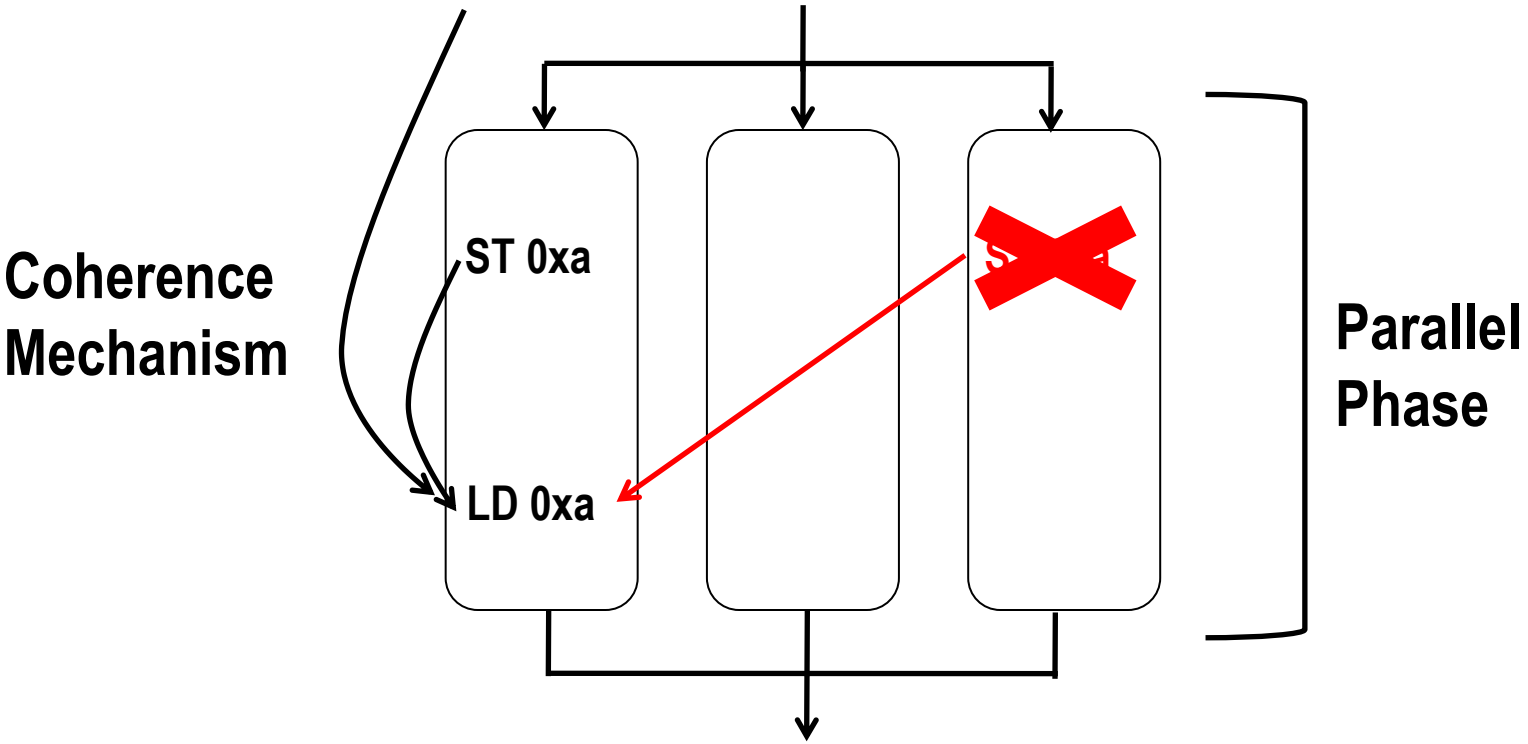




Type-checked programs guaranteed determinism (sequential semantics)

Memory Consistency Model

- **Guaranteed determinism**
  ⇒ Read returns value of last write in sequential order
  1. **Same task in this parallel phase**
  2. **Or before this parallel phase**

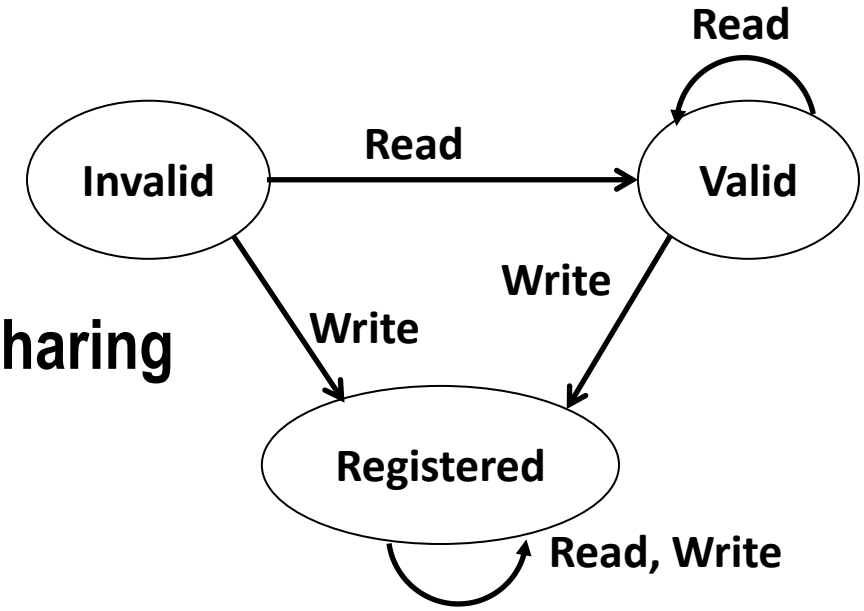


Cache Coherence

- **Coherence Enforcement**
  1. Invalidate stale copies in caches
  2. Track one up-to-date copy
- **Explicit effects**
  - Compiler knows all writeable regions in this parallel phase
  - Cache can self-invalidate before next parallel phase
    - Invalidates data in writeable regions not accessed by itself
- **Registration**
  - Directory keeps track of one up-to-date copy
  - Writer updates before next parallel phase

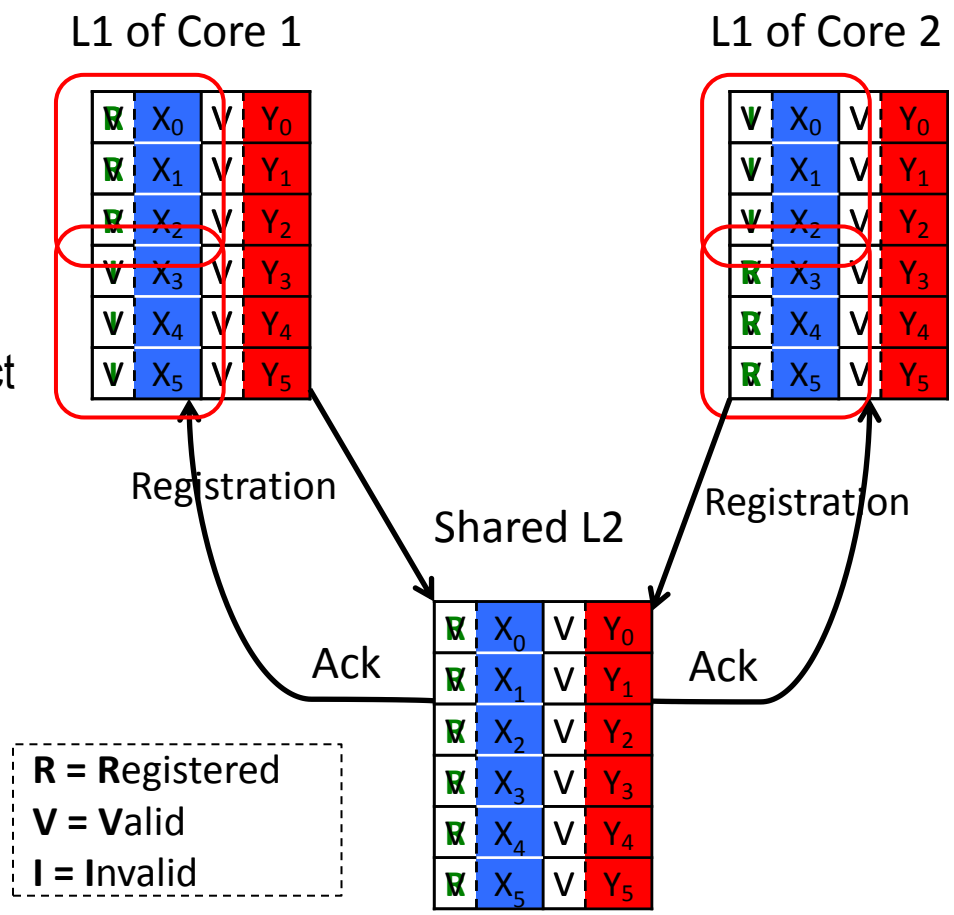
Basic DeNovo Coherence [PACT'11]

- Assume (for now): Private L1, shared L2; single word line
  - Data-race freedom at word granularity
- No transient states
- No invalidation traffic, no false sharing
- No directory storage overhead
  - L2 data arrays double as ~~directory~~ registry
  - Keep valid data or registered core id
- Touched bit: set if word read in the phase



Example Run

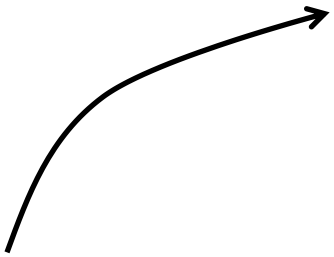
```
class S_type {
    X in DeNovo-region ■ ;
    Y in DeNovo-region ■;
}
S _type S[size];
...
Phase1 writes ■ { // DeNovo effect
    foreach i in 0, size {
        S[i].X = …;
    }
    self_invalidate( ■ );
}
```

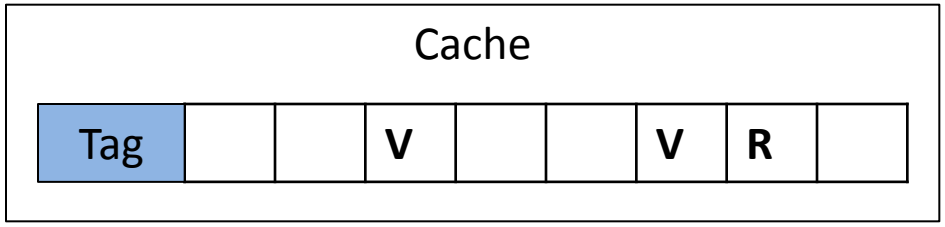


Decoupling Coherence and Tag Granularity

- Basic protocol has tag per word
- DeNovo Line-based protocol
  - Allocation/Transfer granularity > Coherence granularity
    - Allocate, transfer cache line at a time
    - Coherence granularity still at word
    - No word-level false-sharing

"Line Merging"





Current Hardware Limitations

- Complexity
  - ✔ – Subtle races and numerous transient sates in the protocol
  - – Hard to extend for optimizations

- Storage overhead
  - ✔ – Directory overhead for sharer lists (makes up for new bits at ~20 cores)

- Performance and power inefficiencies
  - ✔ – Invalidation, ack messages
  - – Indirection through directory
  - ✔ – False sharing (cache-line based coherence)
  - – Network traffic (cache-line based communication)

Flexible, Direct Communication

Insights

1. Traditional directory must be updated at every transfer

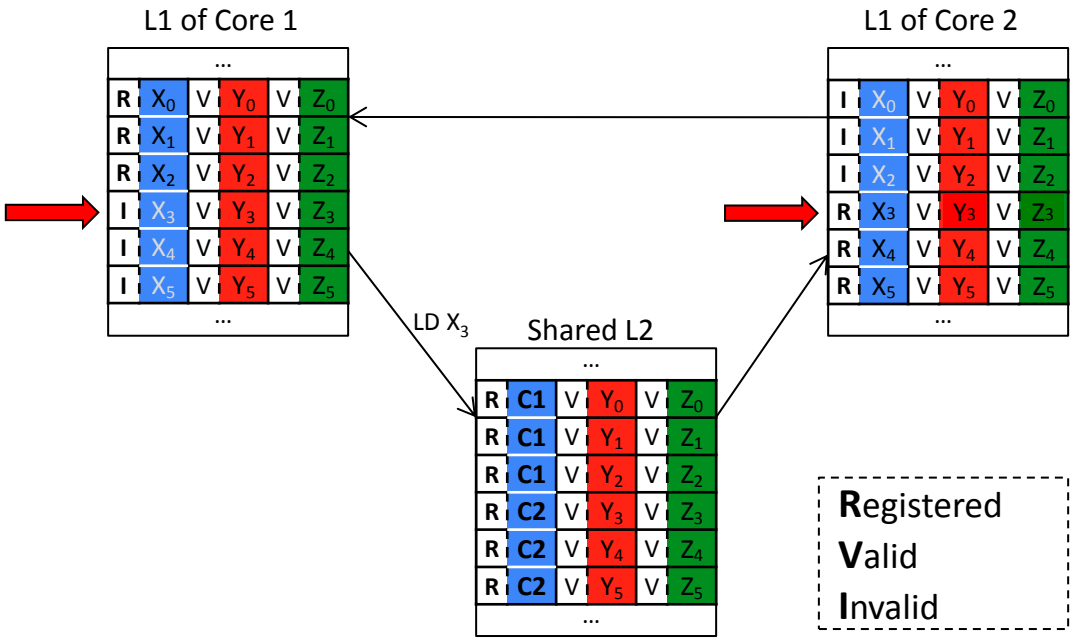
   DeNovo can copy valid data around freely

2. Traditional systems send cache line at a time

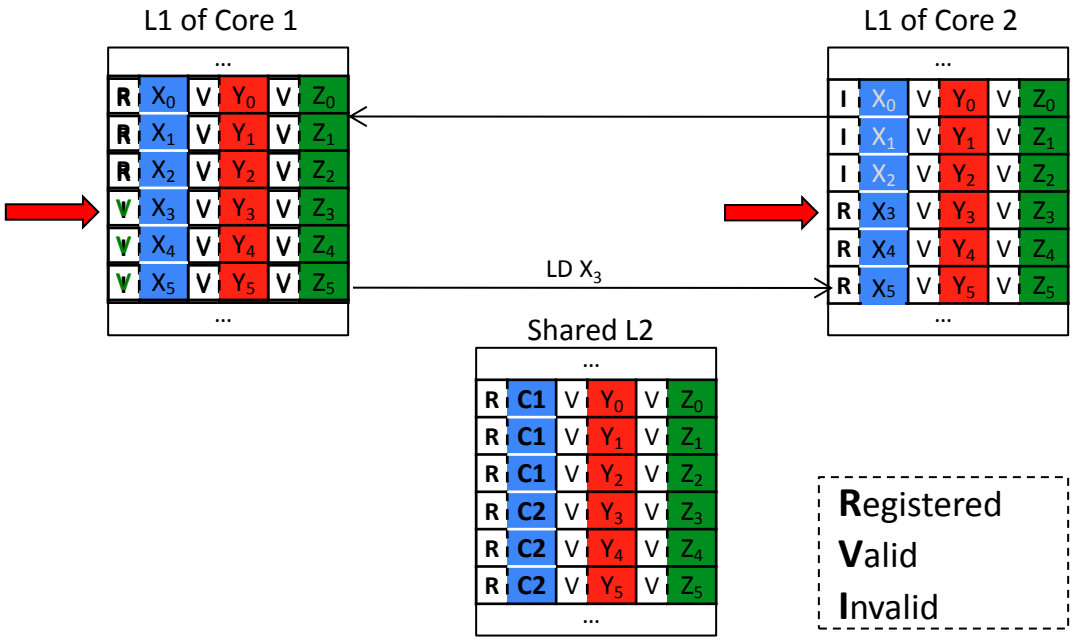
   DeNovo uses regions to transfer only relevant data

   Effect of AoS-to-SoA transformation w/o programmer/compiler

Flexible, Direct Communication



Flexible, Direct Communication



Current Hardware Limitations

- Complexity
  - Subtle races and numerous transient sates in the protocol
  - Hard to extend for optimizations

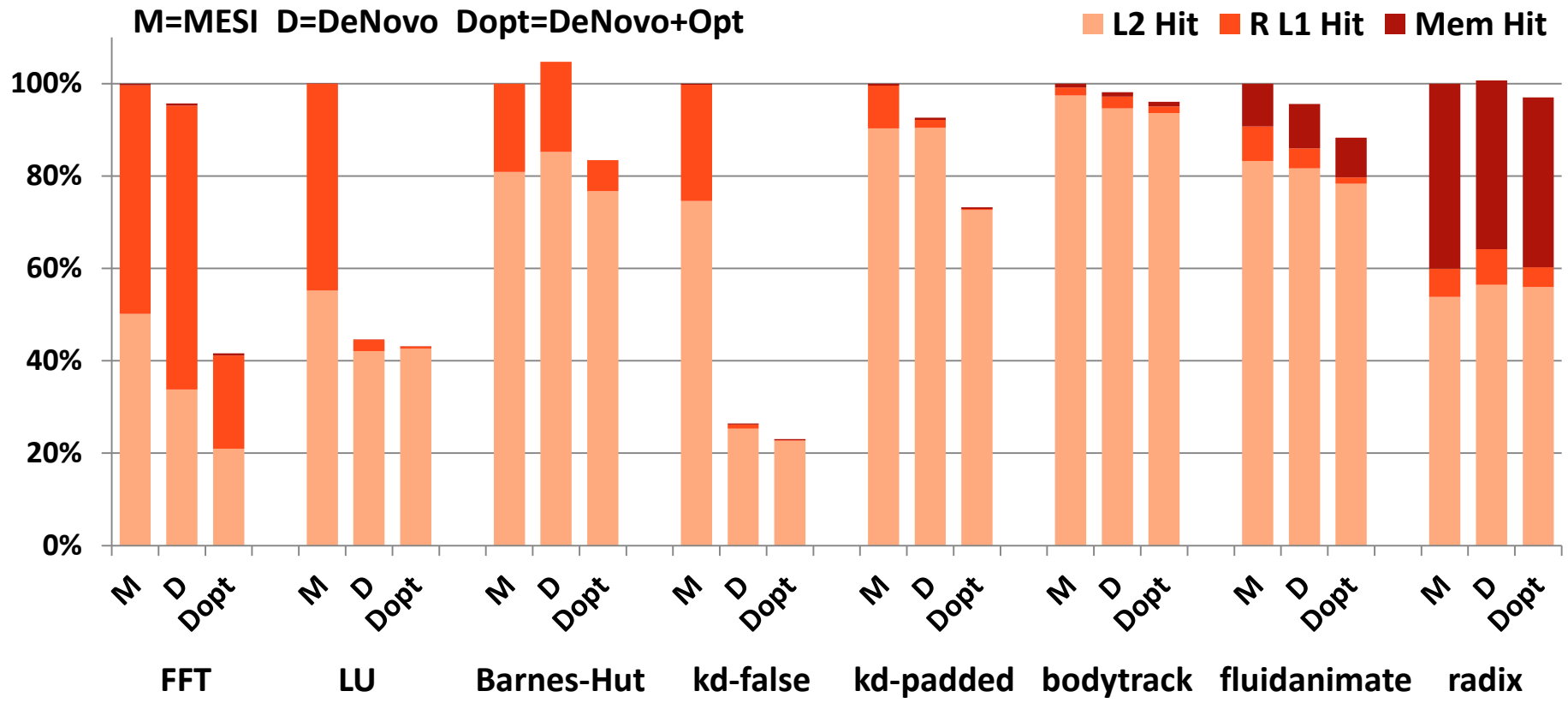
- Storage overhead
  - Directory overhead for sharer lists (makes up for new bits at ~20 cores)

- Performance and power inefficiencies
  - Invalidation, ack messages
  - Indirection through directory
  - False sharing (cache-line based coherence)
  - Network traffic (cache-line based communication)

Evaluation

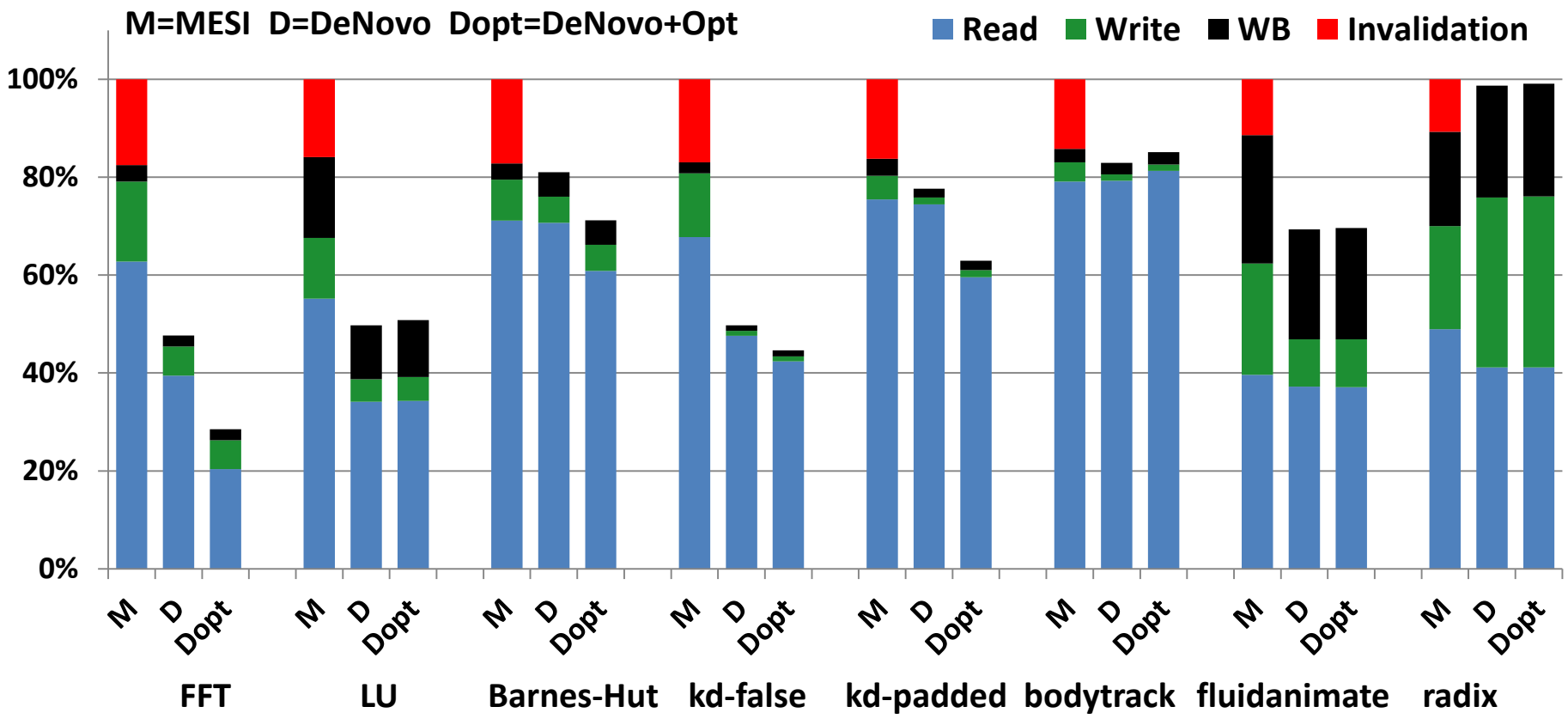
- Verification: DeNovo vs. MESI word w/ Murphi model checker
  - Correctness
    - Six bugs in MESI protocol: Difficult to find and fix
    - Three bugs in DeNovo protocol: Simple to fix
  - Complexity
    - 15x fewer reachable states for DeNovo
    - 20x difference in the runtime
- Performance: Simics + GEMS + Garnet
  - 64 cores, simple in-order core model
  - Workloads
    - FFT, LU, Barnes-Hut, and radix from SPLASH-2
    - bodytrack and fluidanimate from PARSEC 2.1
    - kd-Tree (two versions) [HPG 09]

Memory Stall Time for MESI vs. DeNovo



- DeNovo is comparable to or better than MESI
- DeNovo + opts shows 32% lower memory stalls vs. MESI (max 77%)

Network Traffic for MESI vs. DeNovo



- DeNovo has 36% less traffic than MESI (max 71%)

The DeNovo Approach

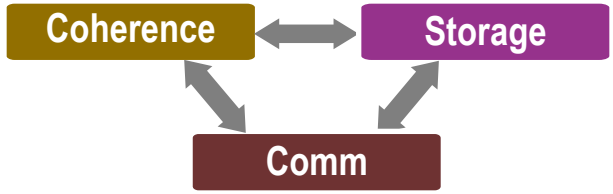
Software

DPJ: Determinism
OOPSLA'09

Disciplined non-determinism
POPL'11

Unstructured synchronization
⇒ OS, Legacy

Hardware



DeNovo: Coherence, Comm
PACT'11 best paper, TACO'14

DeNovoND
ASPLOS'13, Top picks'14

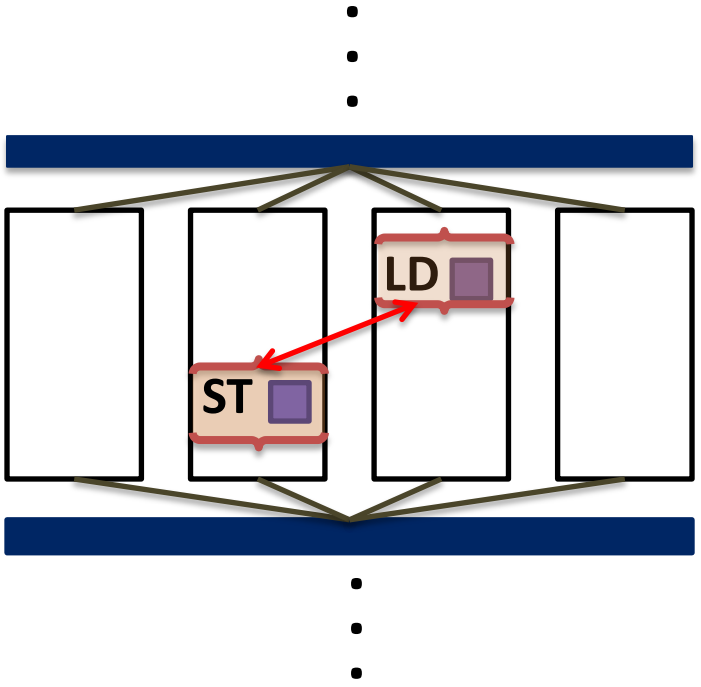
DeNovoSync (in review)

DeNovoH for heterogeneous systems: Coherence, Comm, Storage

Stash: Have your scratchpad and cache it too (in review)

DPJ Support for Disciplined Non-Determinism

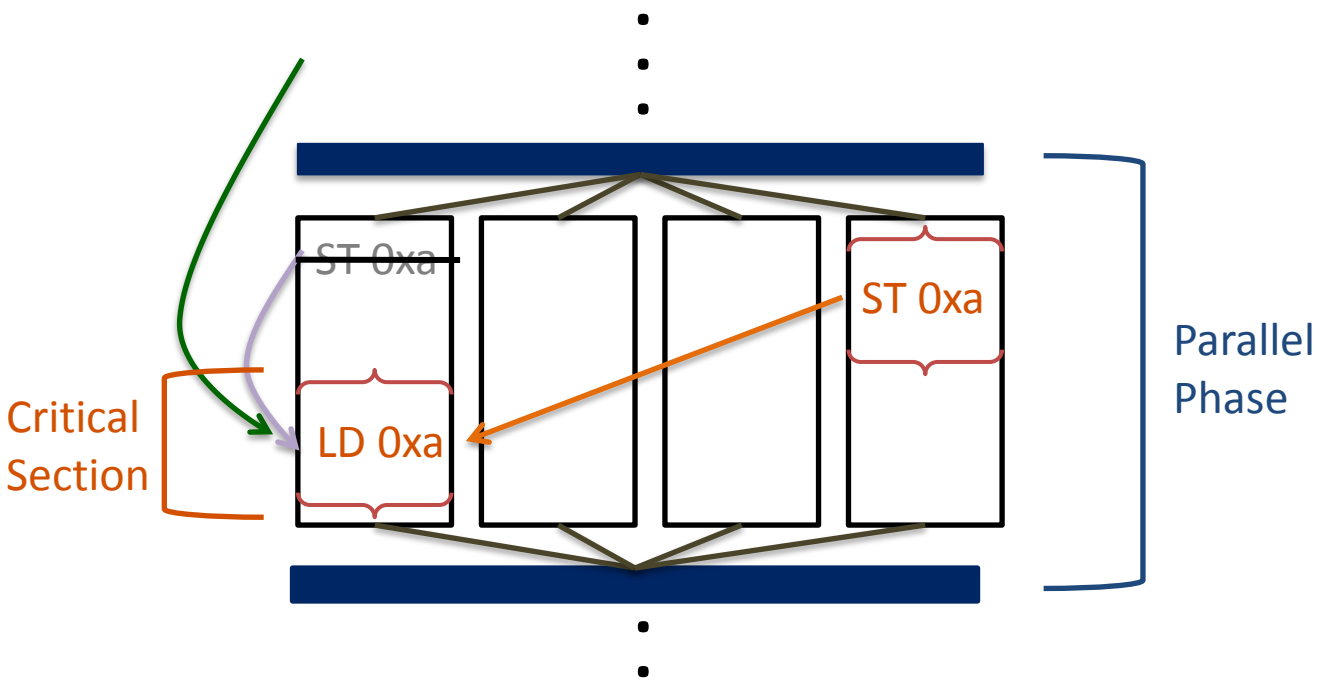
- Non-determinism comes from conflicting concurrent accesses
- Isolate interfering accesses as atomic
  - Enclosed in atomic sections
  - Atomic regions and effects
- Disciplined non-determinism
  - Race freedom, strong isolation
  - Determinism-by-default semantics
- DeNovoND converts atomic statements into locks



Memory Consistency Model

- **Non-deterministic read returns value of last write from**
  1. **Before this parallel phase**
  2. **Or same task in this phase**
  3. **Or in preceding critical section of same lock**

self-invalidations as before single core

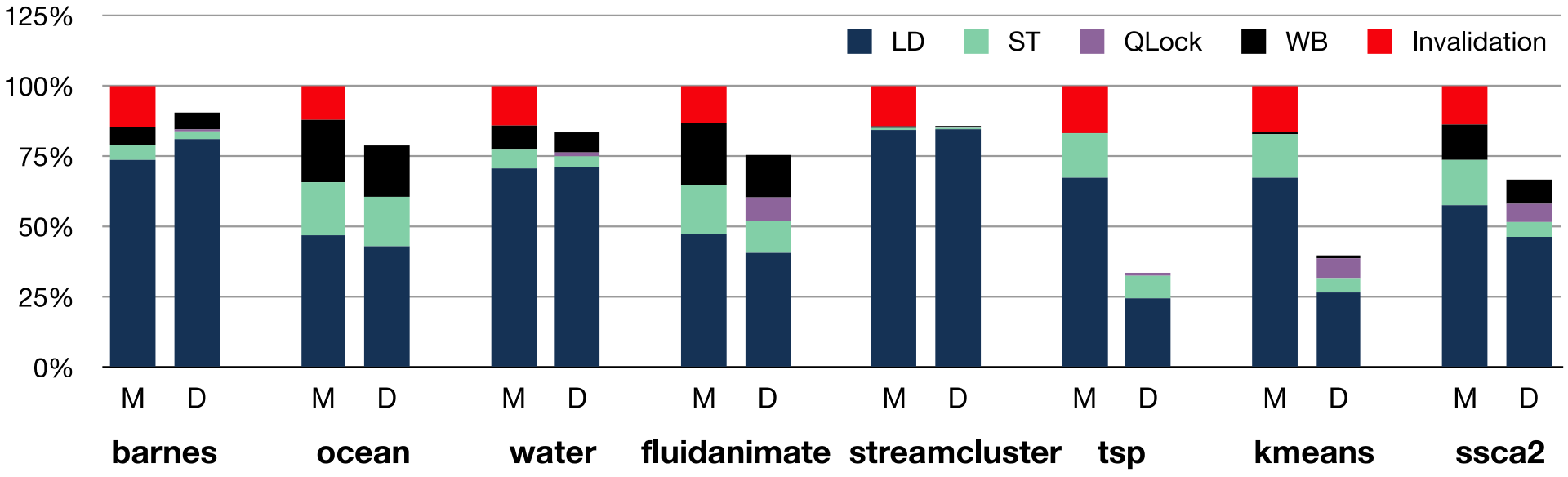


Coherence for Non-Deterministic Data

- Coherence Enforcement
  1. Invalidate stale copies in private cache
  2. Track up-to-date copy
- When to invalidate?
  - Between start of critical section and read
- What to invalidate?
  - Regions with “atomic” effects written in preceding critical sections
  - Track writes w/ small (256b) Bloom filter signature, Xfer with lock
- Registration
  - Writer updates registry before next critical section

Evaluation of MESI vs. DeNovoND (16 cores)

- DeNovoND execution time comparable or better than MESI
- DeNovoND has 33% less traffic than MESI (67% max)
  - No invalidation traffic
  - Reduced load misses due to lack of false sharing



The DeNovo Approach

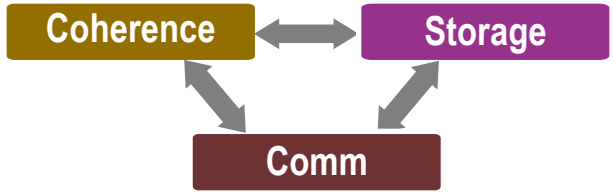
Software

DPJ: Determinism
OOPSLA'09

Disciplined non-determinism
POPL'11

Unstructured synchronization
⇒ OS, Legacy

Hardware



DeNovo: Coherence, Comm
PACT'11 best paper, TACO'14

DeNovoND
ASPLOS'13, Top picks'14

DeNovoSync (in review)

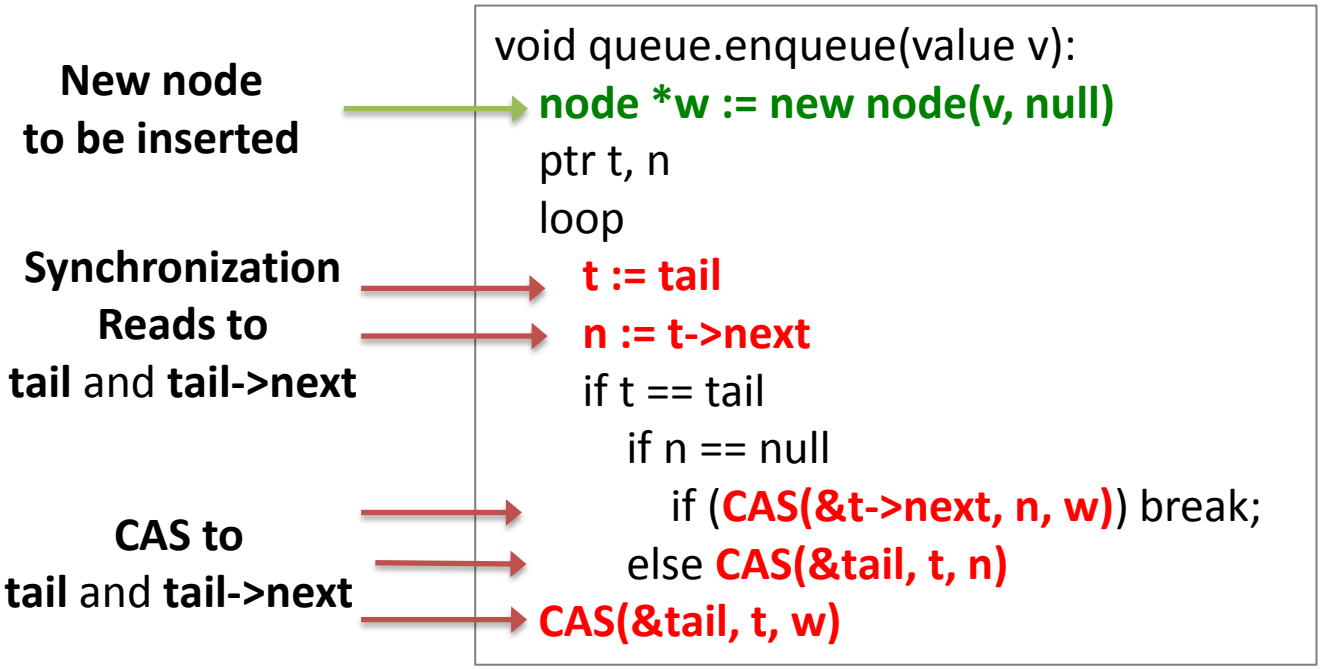
DeNovoH for heterogeneous systems: Coherence, Comm, Storage

Stash: Have your scratchpad and cache it too (in review)

Unstructured Synchronization

- **Many programs use arbitrary, unstructured synchronization**

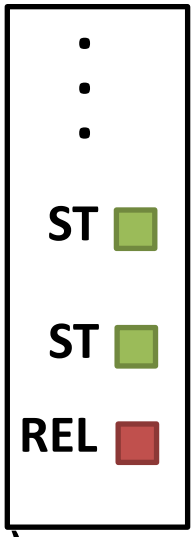
Michael-Scott non-blocking queue

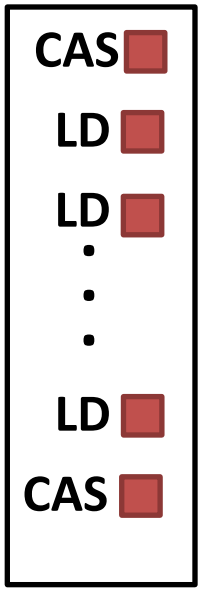


- **Accesses to data still ordered by synchronization (data-race-free)**
  - **Can use static self-invalidation or signatures**
- **But what about synchronization accesses?**

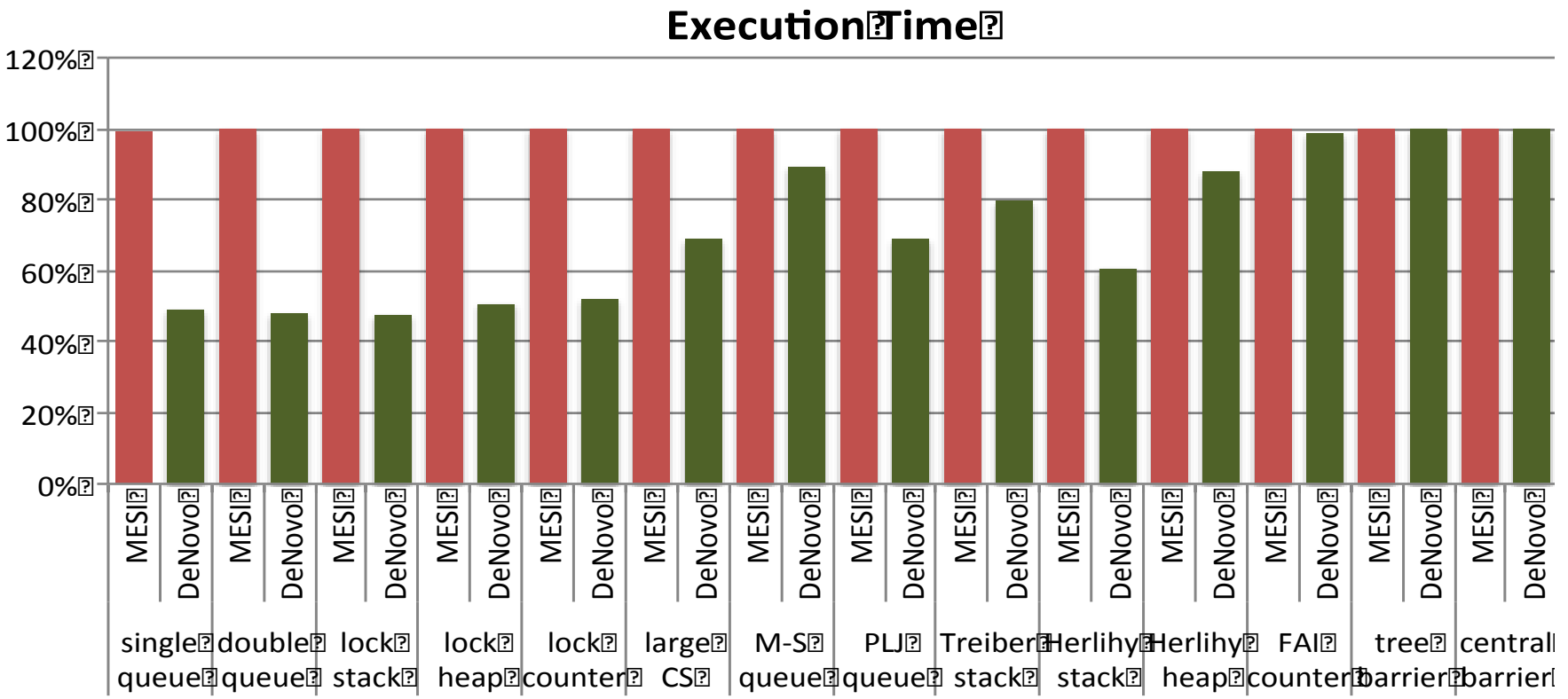
Unstructured Synchronization

- Problem: Synchronization accesses are inherently racy
  - Rely on writer-initiated invalidations
- Reader-initiated invalidations
  - What to invalidate, when to invalidate?
    - ~~Every read?~~
    - Every read to non-registered state
    - Register all sync reads (to enable future hits)
  - Concurrent readers?
    - Back off (delay) read registration





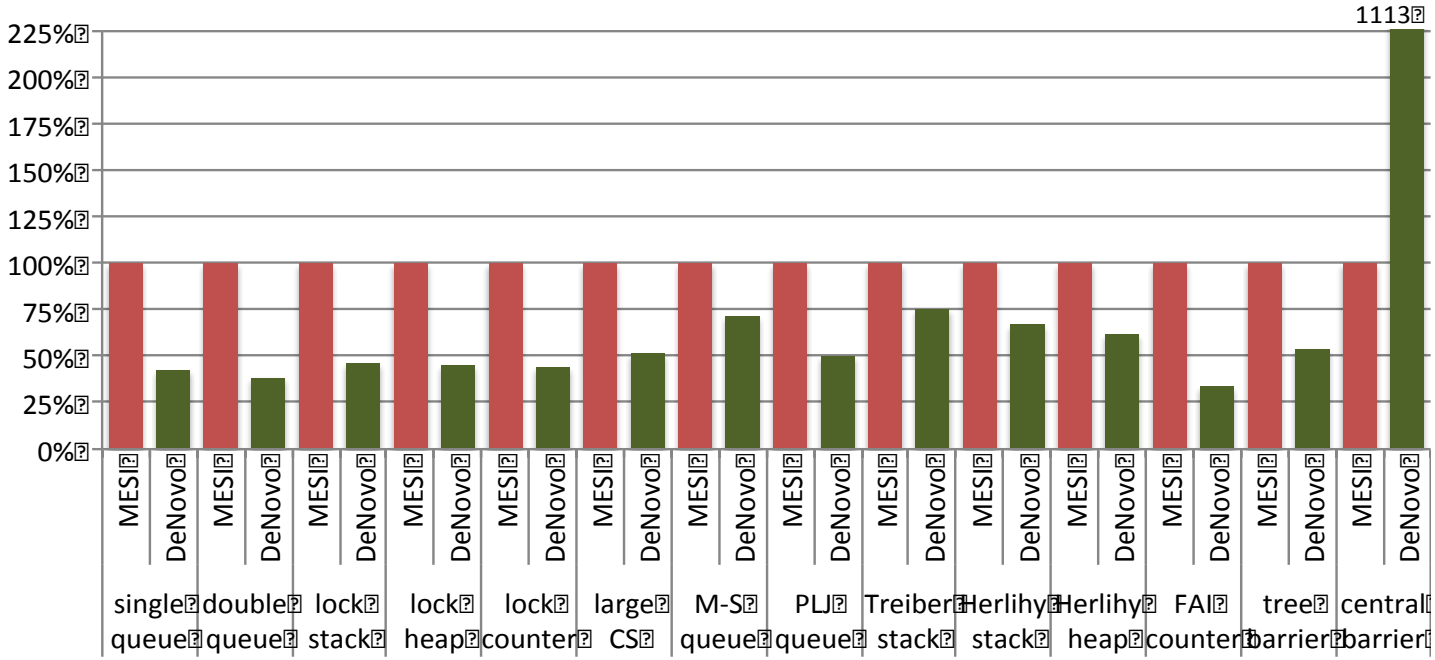
Unstructured Synch: Execution Time on 64 Cores



DeNovoSync reduces execution time by 28% over MESI (max 49%)

- **MESI's high invalidation overhead vs.**
- **DeNovo's fast point-to-point registration transfer**

Unstructured Synch: Network Traffic on 64 Cores



DeNovo reduces traffic by 44% vs. MESI (max 61%) for 11 of 12 cases

Centralized barrier

- **Many concurrent readers hurt DeNovo (and MESI)**
- **Should use tree barrier even with MESI**

The DeNovo Approach

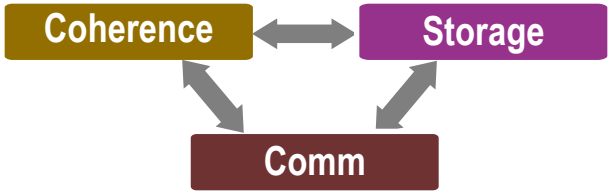
Software

DPJ: Determinism
OOPSLA'09

Disciplined non-determinism
POPL'11

Unstructured synchronization
⇒ OS, Legacy

Hardware



DeNovo: Coherence, Comm
PACT'11 best paper

DeNovoND
ASPLOS'13, Top picks'14

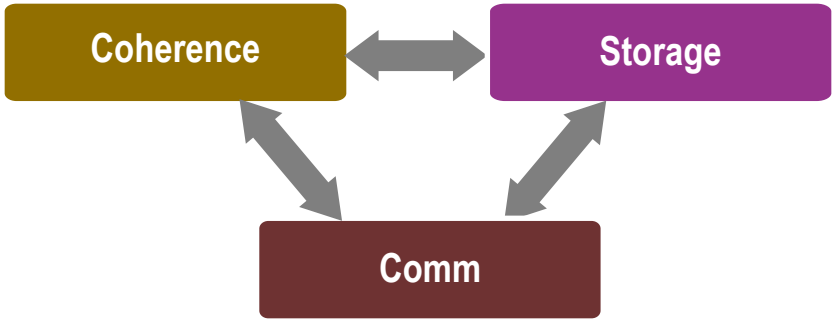
DeNovoSync (in review)

DeNovoH for heterogeneous systems: Coherence, Comm, Storage

Stash: Have your scratchpad and cache it too (in review)

Conclusions and Future Work

DeNovo rethinks memory hierarchy for disciplined models



- For deterministic codes
  - Complexity: no transients, 20X faster to verify, extensible
  - Storage overhead: no directory overhead
  - Performance, power: No inv/acks, false sharing, indirection, …

    Up to 77% lower memory stall time, up to 71% lower traffic
- Benefits even for non-determinism and unstructured synchs

Future

- Run full OS, legacy codes
- Heterogeneous memory structures and consistency models
- Virtual ISA for heterogeneous systems