DeNovo: Rethinking the Multicore Memory Hierarchy for Disciplined Parallelism

Byn Choi, Nima Honarmand, Rakesh Komuravelli,
Robert Smolinski, Hyojin Sung, Sarita V. Adve,
Vikram S. Adve, Nicholas P. Carter[□], Ching-Tsun Chou[□]
University of Illinois, Urbana-Champaign,
[□]Intel

UPCRC Illinois
Universal Parallel Computing Research Center

Motivation

- Goal: Power-, complexity-, performance-scalable hardware
- Today: shared-memory
  - Directory-based coherence
    - Complex and unscalable
  - Difficult programming model
    - Data races, non-determinism, no safety/composability/modularity
  - 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 and software

Solution?

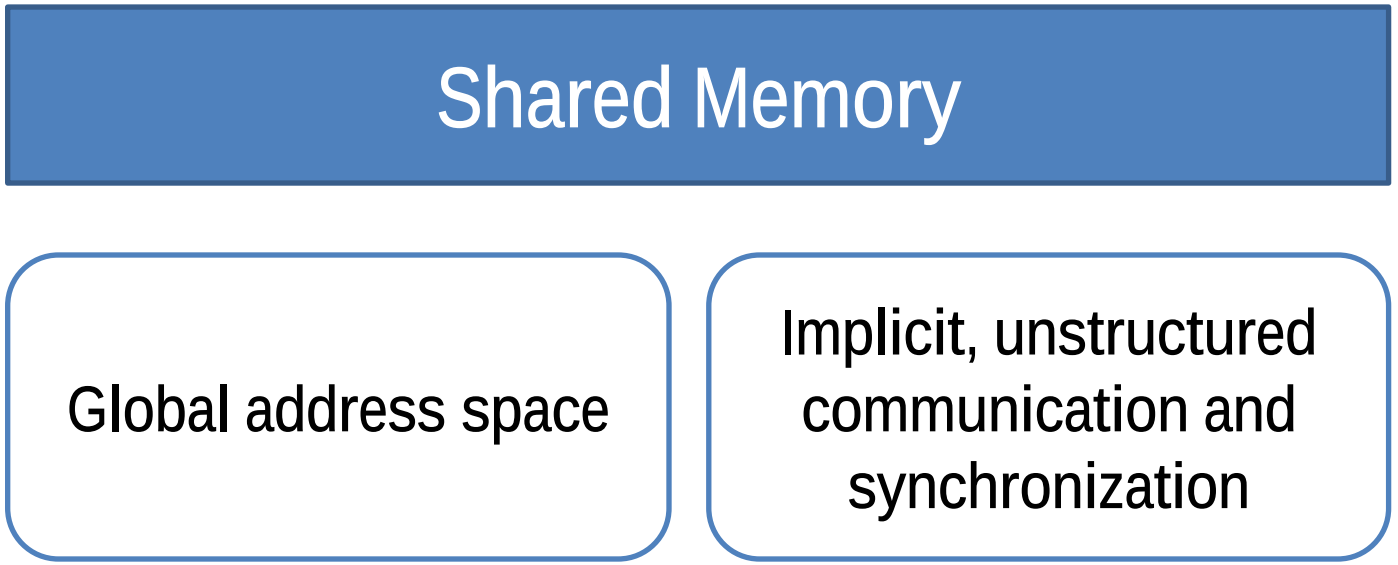
- Banish shared memory?

As you scale the number of cores on a cache coherent system (CC), “cost” in “time and memory” grows to a point beyond which the additional cores are not useful in a single parallel program. This is the coherency wall….

A case for message-passing for many-core computing,
Timothy Mattson et al

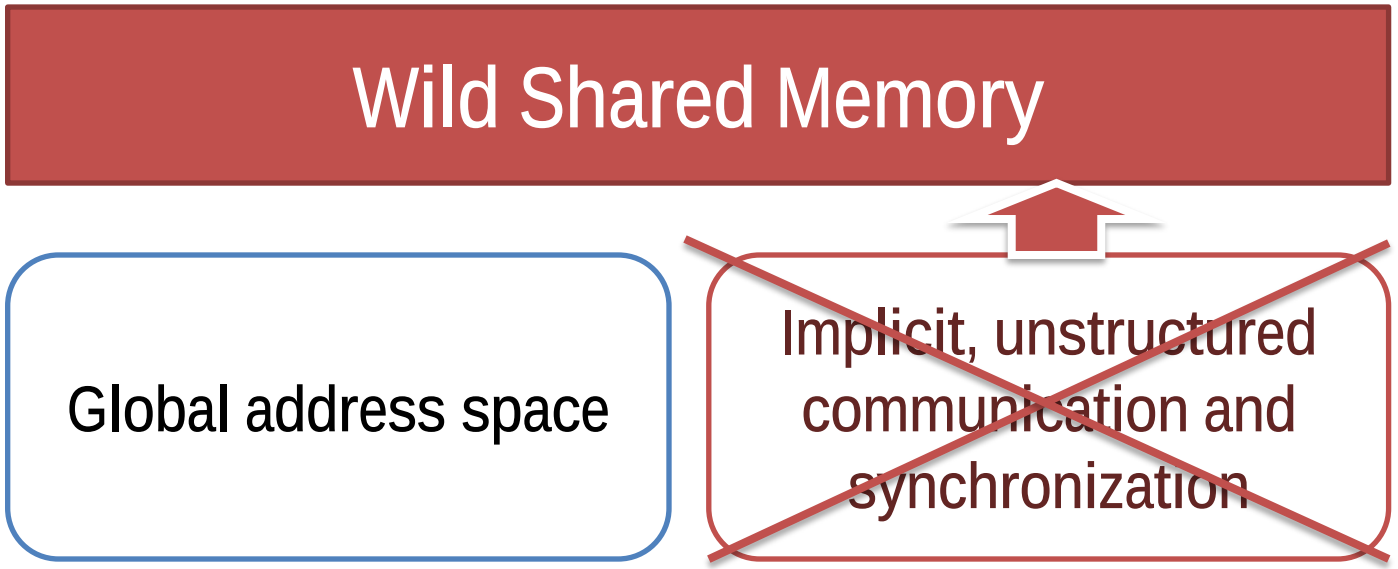
Solution

- The problems are not inherent to shared memory paradigm



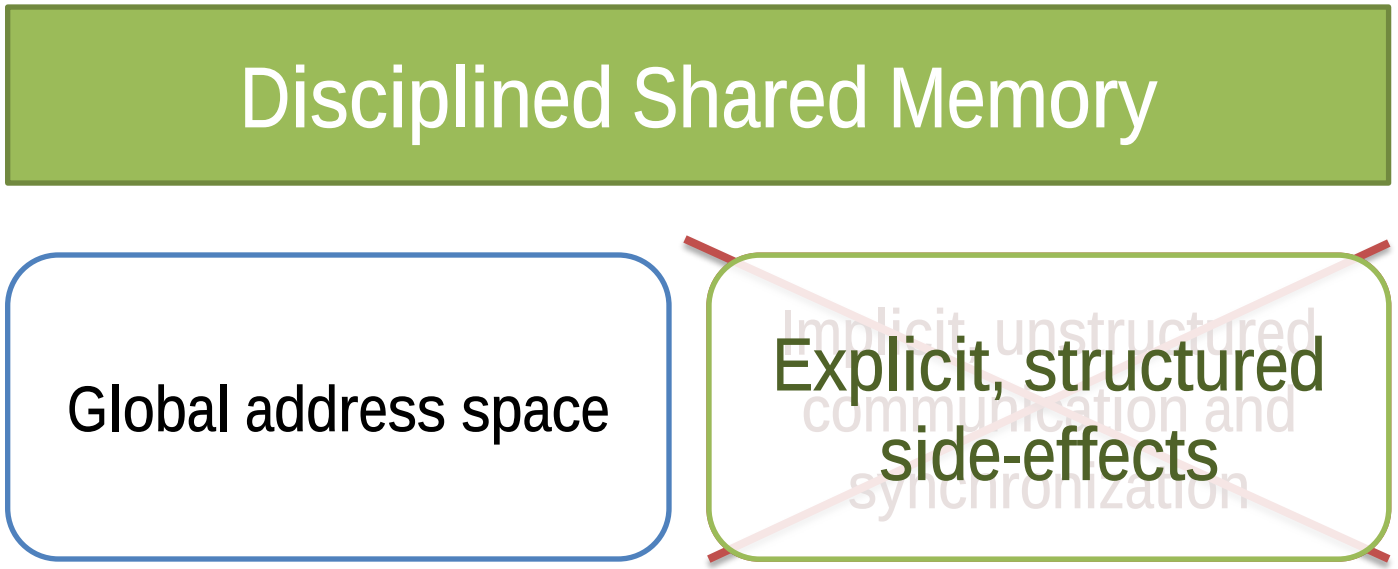
Solution

- Banish wild shared memory!



Solution

- Build disciplined shared memory!



Disciplined Shared Memory

- No data races, determinism-by-default, safe non-determinism
- Simple semantics, safety and composability

explicit effects + structured parallel control

Disciplined Shared Memory

- Simple coherence and consistency
- Software-aware address/comm/coherence granularity
- Power-, complexity-, performance-scalable HW

Research Strategy: Software Scope

- Many systems provide disciplined shared-memory features
- Current driver is DPJ (Deterministic Parallel Java)
  - Determinism-by-default, safe non-determinism
- End goal is language-oblivious interface
- Current focus on deterministic codes
  - Common and best case
- Extend later to safe non-determinism, legacy codes

Research Strategy: Hardware Scope

- Today's focus: cache coherence
- Limitations of current directory-based protocols
  1. Complexity
     - Subtle races and numerous transient sates in the protocol
     - Hard to extend for optimizations
  2. Storage overhead
     - Directory overhead for sharer lists
  3. Performance and power inefficiencies
     - Invalidation and ack messages
     - False sharing
     - Indirection through the directory
     - Suboptimal comm. granularity of cache line …
- Ongoing: Rethink communication architecture and data layout

Contributions

- Simplicity
- Extensibility
- Storage overhead
- Performance/Power

Contributions

- Simplicity
  - Compared protocol complexity with MESI
  - 25x less reachable states with model checking
- Extensibility
- Storage overhead
- Performance/Power

Contributions

- Simplicity
  - Compared protocol complexity with MESI
  - 25x less reachable states with model checking
- Extensibility
  - Direct cache-to-cache transfer
  - Flexible communication granularity
- Storage overhead

- Performance/Power

Contributions

- Simplicity
  - Compared protocol complexity with MESI
  - 25x less reachable states with model checking
- Extensibility
  - Direct cache-to-cache transfer
  - Flexible communication granularity
- Storage overhead
  - No storage overhead for directory information
  - Storage overheads beat MESI after tens of cores and scale beyond
- Performance/Power

Contributions

- Simplicity
  - Compared protocol complexity with MESI
  - 25x less reachable states with model checking
- Extensibility
  - Direct cache-to-cache transfer
  - Flexible communication granularity
- Storage overhead
  - No storage overhead for directory information
  - Storage overheads beat MESI after tens of cores and scale beyond
- Performance/Power
  - Up to 73% reduction in memory stall time
  - Up to 70% reduction in network traffic

Outline

- Motivation
- Background: DPJ
- DeNovo Protocol
- DeNovo Extensions
- Evaluation
  - Protocol Verification
  - Performance
- Conclusion and Future Work

Background: DPJ

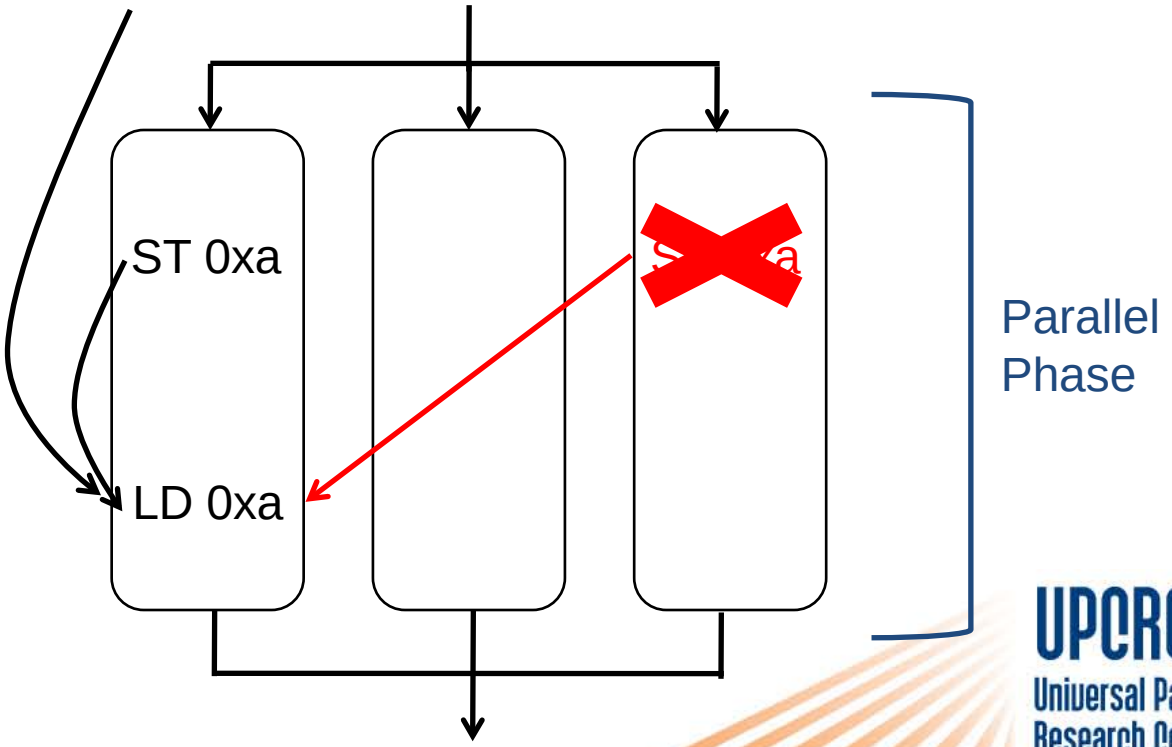
- Extension for modern OO languages
- Structured parallel control: nested fork-join style
  - *foreach*, *cobegin*
- Type and effect system ensures non-interference
  - Region names: partition heap
  - Effects for methods: which regions read/written in method
  - Type checking ensures race-freedom for parallel tasks

⇒ *Deterministic execution*

- Recent support for safe non-determinism

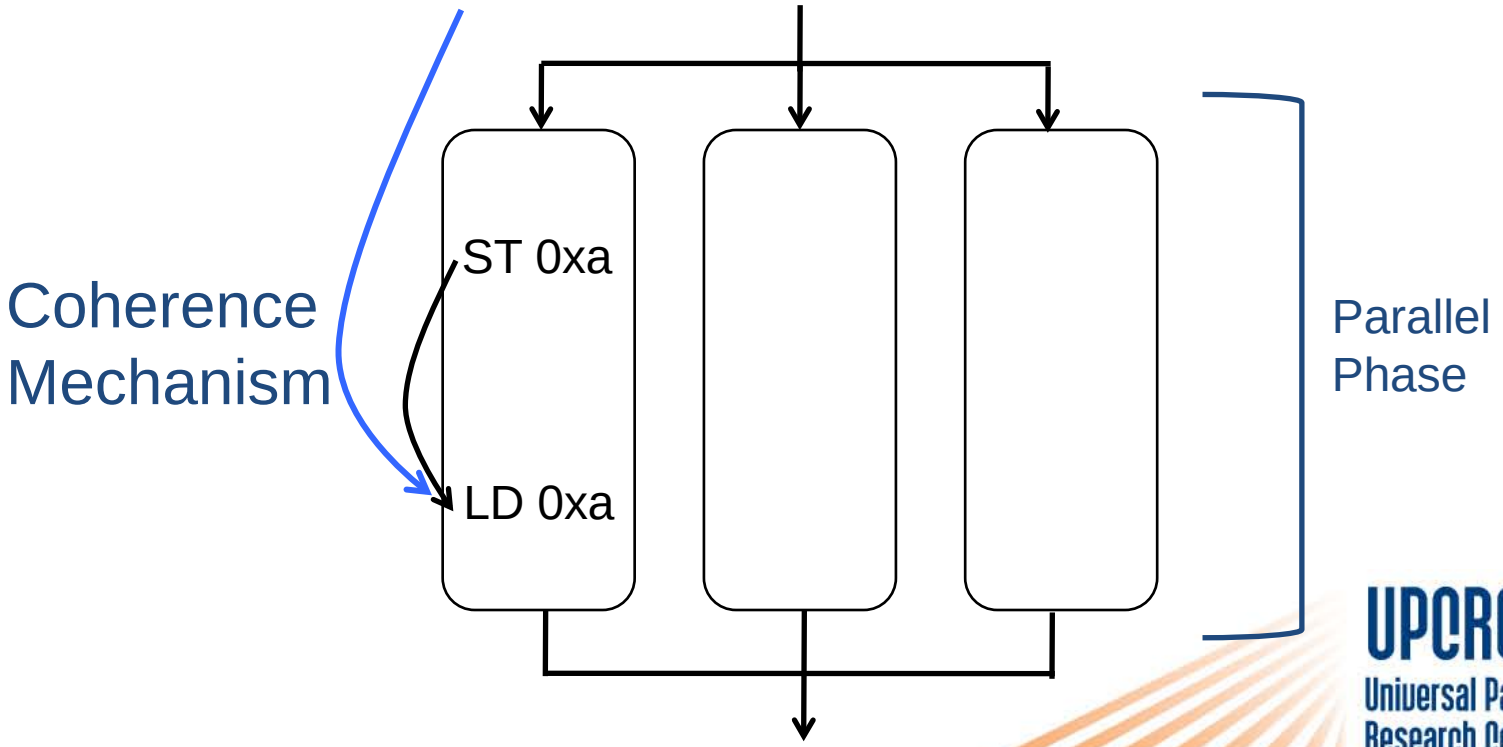
Memory Consistency Model

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



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 up-to-date copy

Cache Coherence

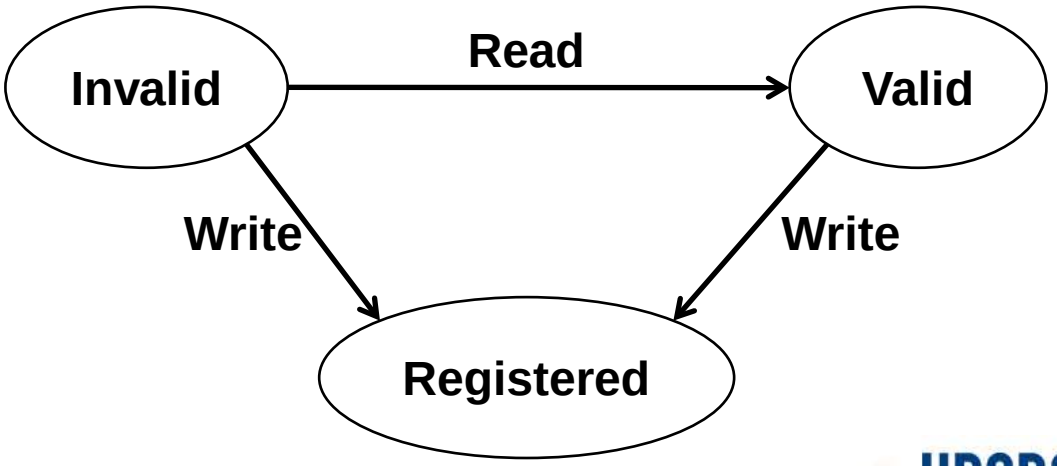
- Coherence Enforcement
  1. Invalidate stale copies in caches
  2. Track up-to-date copy
- Explicit effects
  - Compiler knows all regions written in this parallel phase
  - Cache can self-invalidate before next parallel phase
    - Invalidates data in writeable regions not accessed by itself

Cache Coherence

- Coherence Enforcement
  1. Invalidate stale copies in caches
  2. Track up-to-date copy
- Explicit effects
  - Compiler knows all regions written 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

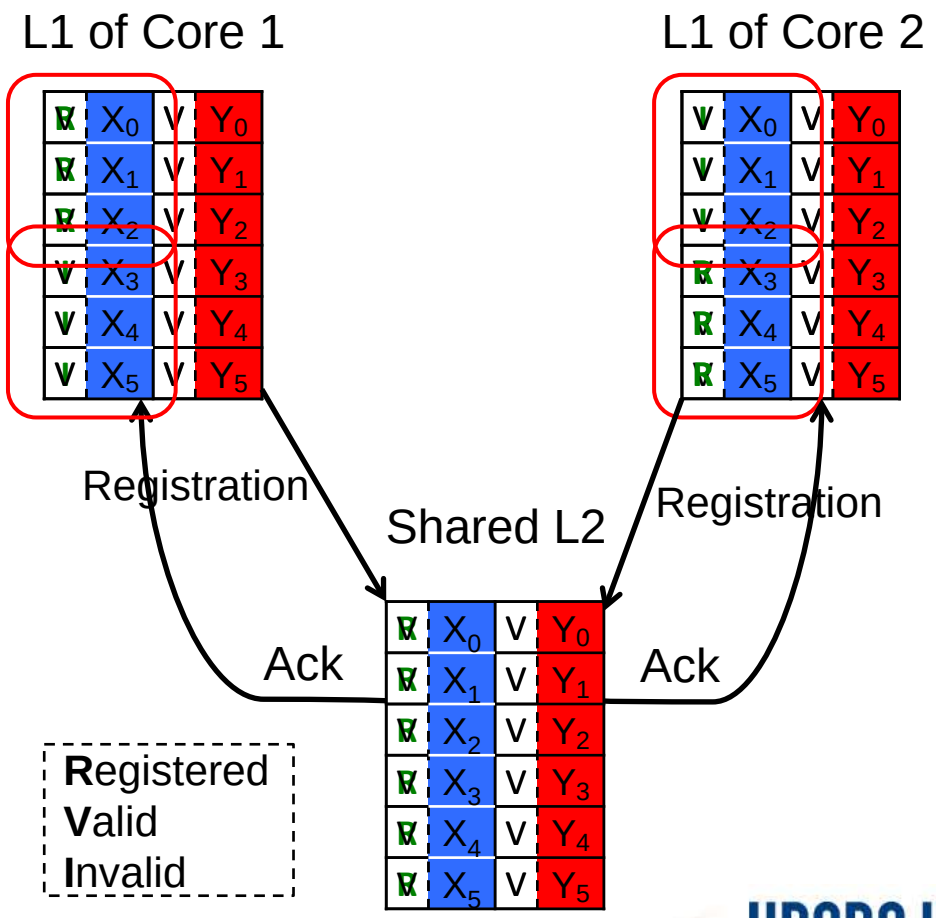
- Assume (for now): Private L1, shared L2; single word line
  - Data-race freedom at word granularity
- L2 data arrays double as ~~directory~~ registry
  - Keep valid data or registered core id, no space overhead
- L1/L2 states



- "Touched" bit – set only if 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( ■ );
}
```



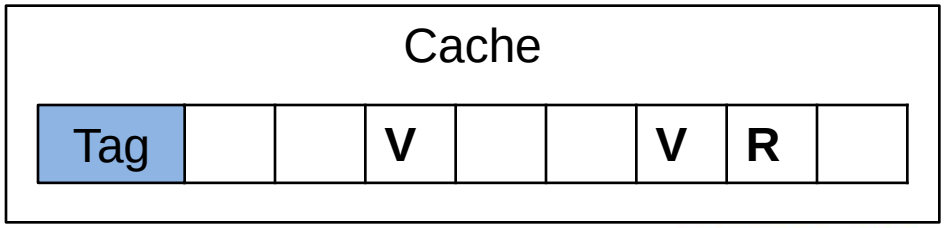
Addressing Limitations

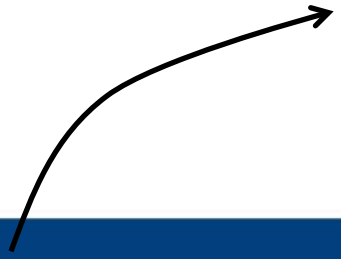
- Limitations of current directory-based protocols
  1. Complexity ✔
     - Subtle races and numerous transient sates in the protocol
     - Hard to extend for optimizations
  2. Storage overhead ✔
     - Directory overhead for sharer lists
  3. Performance and power overhead
     - Invalidation and ack messages ✔
     - False-sharing
     - Indirection through the directory

Practical DeNovo Coherence

- Basic protocol impractical
  - High tag storage overhead (a tag per word)
- Address/Transfer granularity > Coherence granularity
- DeNovo Line-based protocol
  - Traditional software-oblivious spatial locality
  - Coherence granularity still at word
    - no word-level false-sharing

“Line Merging”

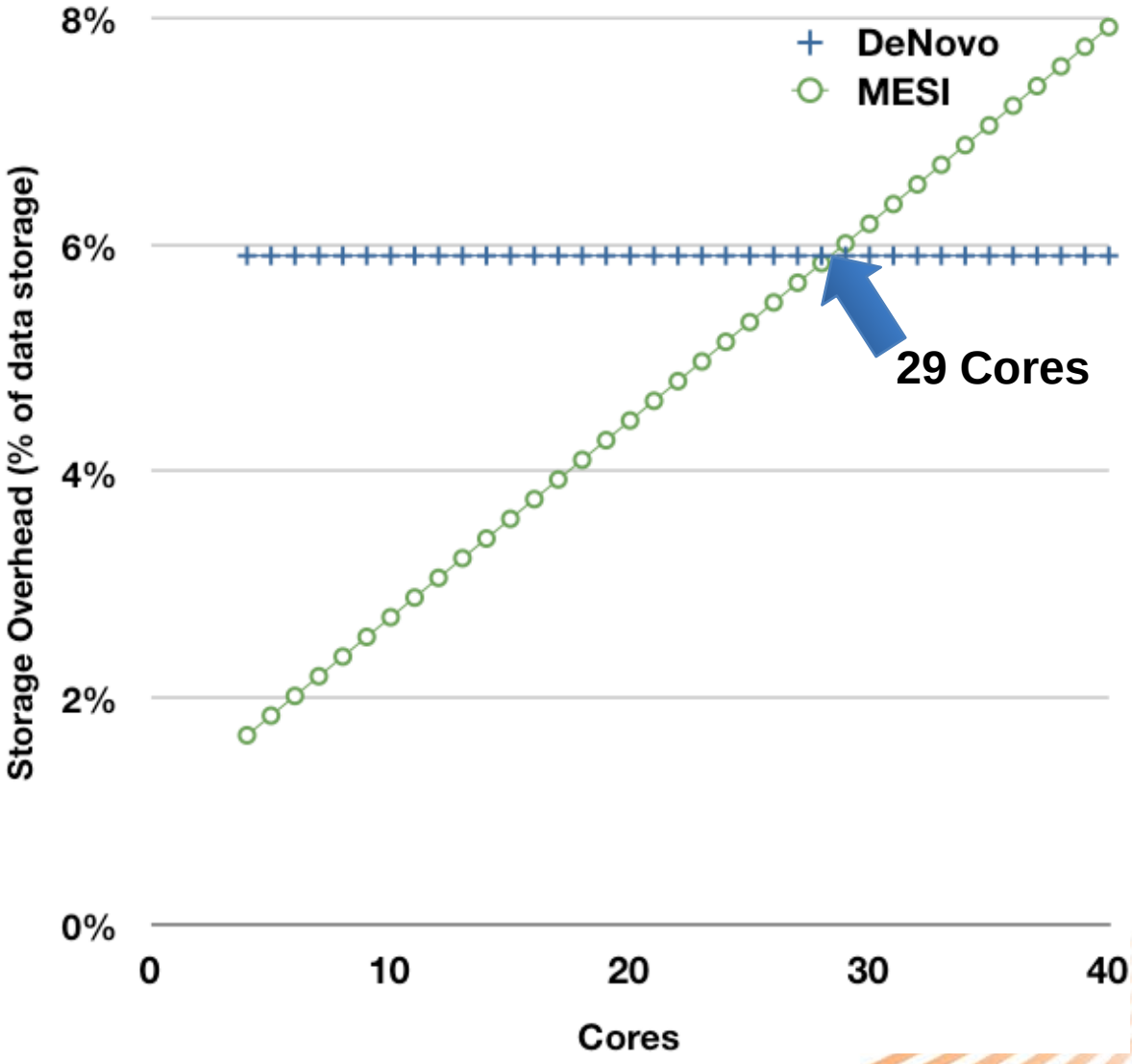




Storage Overhead

- DeNovo line (64 bytes/line)
  - L1: 2 state bits, 1 touched bit, 5 region bits (128 bits/line)
  - L2: 1 valid/line, 1 dirty/line, 1 state bit/word (18 bits/line)
- MESI (full-map, in-cache directory)
  - L1: 5 state bits (5 bits/line)
  - L2: 5 state bits, [# of cores for directory] bits (5+P bits/line)
- Size of L2 == 8 x [aggregate size of L1s]

Storage Overhead



Alternative Directory Designs

- Duplicate Tag Directory
  - L1 tags duplicated in directory
  - Requires highly-associative lookups
    - 256-way associative lookup for 64-core setup with 4-way L1s
- Sparse Directory
  - Significant performance overhead with higher core-counts
- Tagless Directory (MICRO '09)
  - Imprecise sharer-list encoding using bloom-filters
  - Even more complexity in the protocol (false-positives)
  - Trade performance for less storage overhead/power

Addressing Limitations

- Limitations of current directory-based protocols
  1. Complexity ✔
     - Subtle races and numerous transient sates in the protocol
     - Hard to extend for optimizations
  2. Storage overhead ✔
     - Directory overhead for sharer lists
  3. Performance and power overhead
     - Invalidation and ack messages ✔
     - False-sharing ✔
     - Indirection through the directory

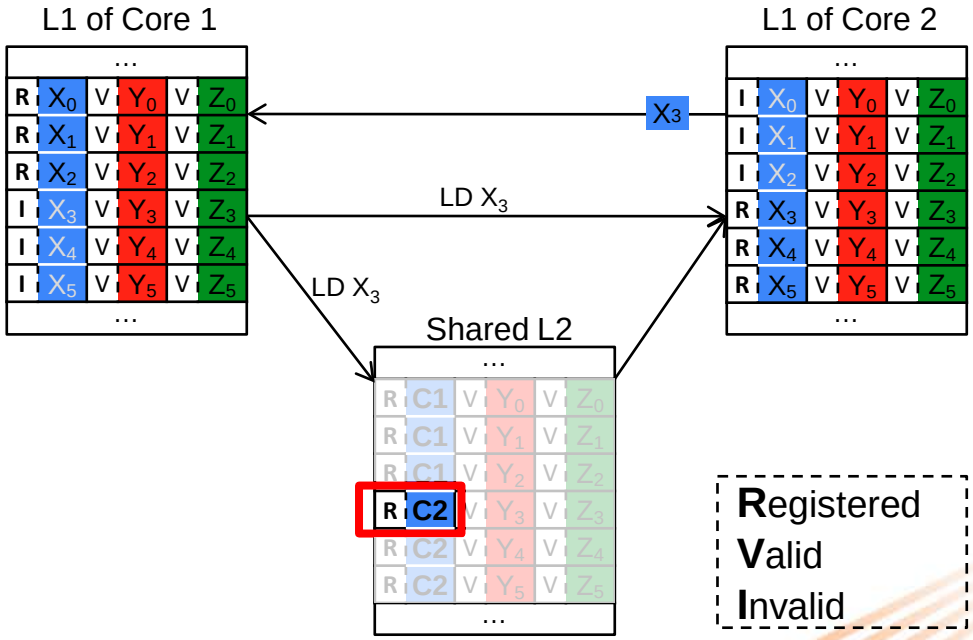
Extensions

- Traditional directory-based protocols
  ⇒ Sharer-lists always contain all the true sharers
- DeNovo protocol
  ⇒ Registry points to latest copy at end of phase

➔ **Valid data can be copied around freely**

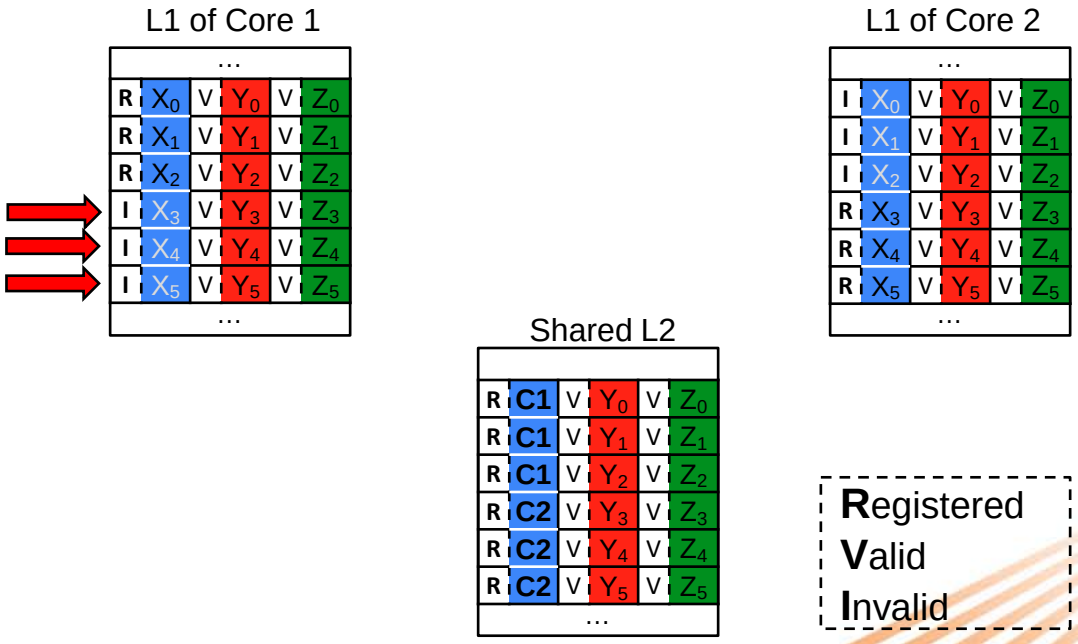
Extensions (1 of 2)

- Basic with Direct cache-to-cache transfer
  - Get data directly from producer
  - Through prediction and/or software-assistance
  - Convert 3-hop misses to 2-hop misses



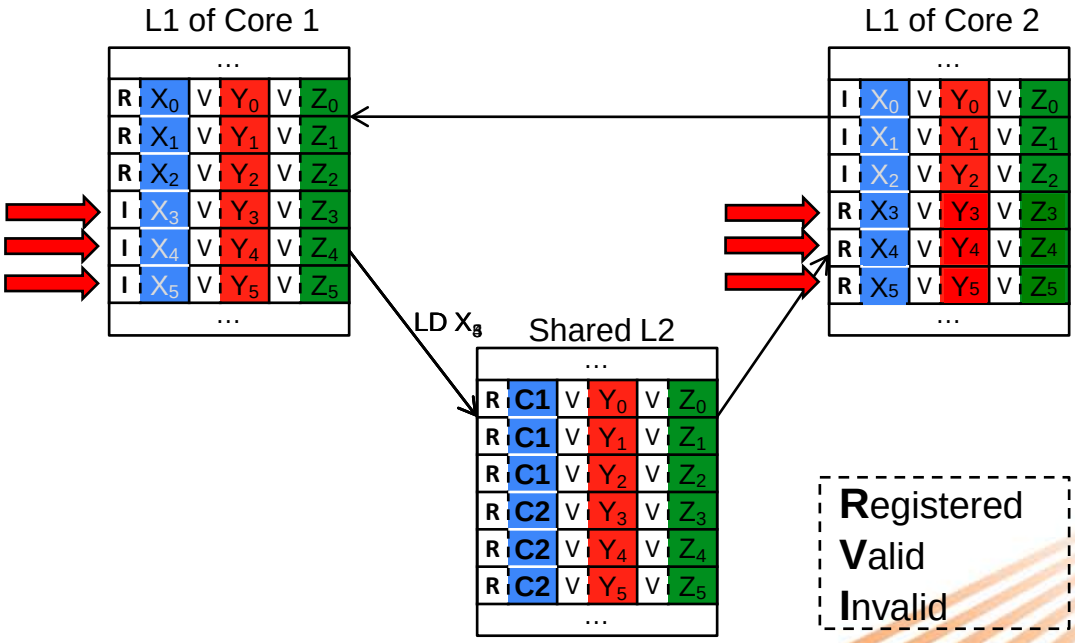
Extensions (2 of 2)

- Basic with Flexible communication
  - Software-directed data transfer
  - Transfer “relevant” data together
  - Achieve effects of AoS-to-SoA transformation without programmer/compiler intervention



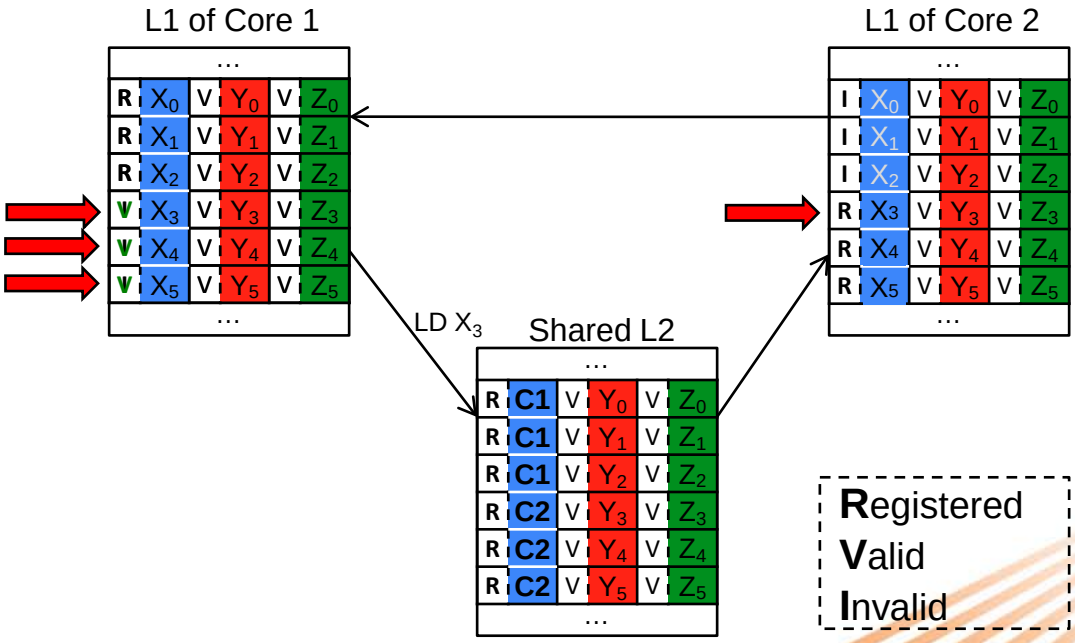
Extensions (2 of 2)

- Basic with Flexible communication
  - Software-directed data transfer
  - Transfer “relevant” data together
  - Achieve effects of AoS-to-SoA transformation without programmer/compiler intervention



Extensions (2 of 2)

- Basic with Flexible communication
  - Software-directed data transfer
  - Transfer “relevant” data together
  - Achieve effects of AoS-to-SoA transformation without programmer/compiler intervention



Outline

- Motivation
- Background: DPJ
- DeNovo Protocol
- DeNovo Extensions
- Evaluation
  - Protocol Verification
  - Performance
- Conclusion and Future Work

Evaluation

- Simplicity
  - Formal verification of the cache coherence protocol
  - Comparing reachable states
- Performance/Power
  - Simulation experiments
- Extensibility
  - DeNovo extensions

Protocol Verification Methodology

- Murphi model checking tool
- Verified DeNovo word and MESI word protocols
  - State-of-the art GEMS implementation
- Abstract model
  - Single address / region, two data values
  - Two cores with private L1 and unified L2, unordered n/w
  - Data race free guarantee for DeNovo
  - Cross phase interactions

Protocol Verification: Results

- Correctness
  - Three bugs in DeNovo protocol
    - Mistakes in translation from the high level specification
    - Simple to fix
  - Six bugs in MESI protocol
    - Two deadlock scenarios
    - Unhandled races due to L1 writebacks
    - Several days to fix
- Complexity
  - 25x fewer reachable states for DeNovo
  - 30x difference in the runtime

Performance Evaluation Methodology

- Simulation Environment
  - Wisconsin GEMS + Simics + Princeton Garnet n/w
- System Parameters
  - 64 cores
  - Private L1(128KB) and Unified L2(32MB)
- Simple core model
  - 5-stage, one-issue, in-order core
  - Results for only memory stall time

Benchmarks

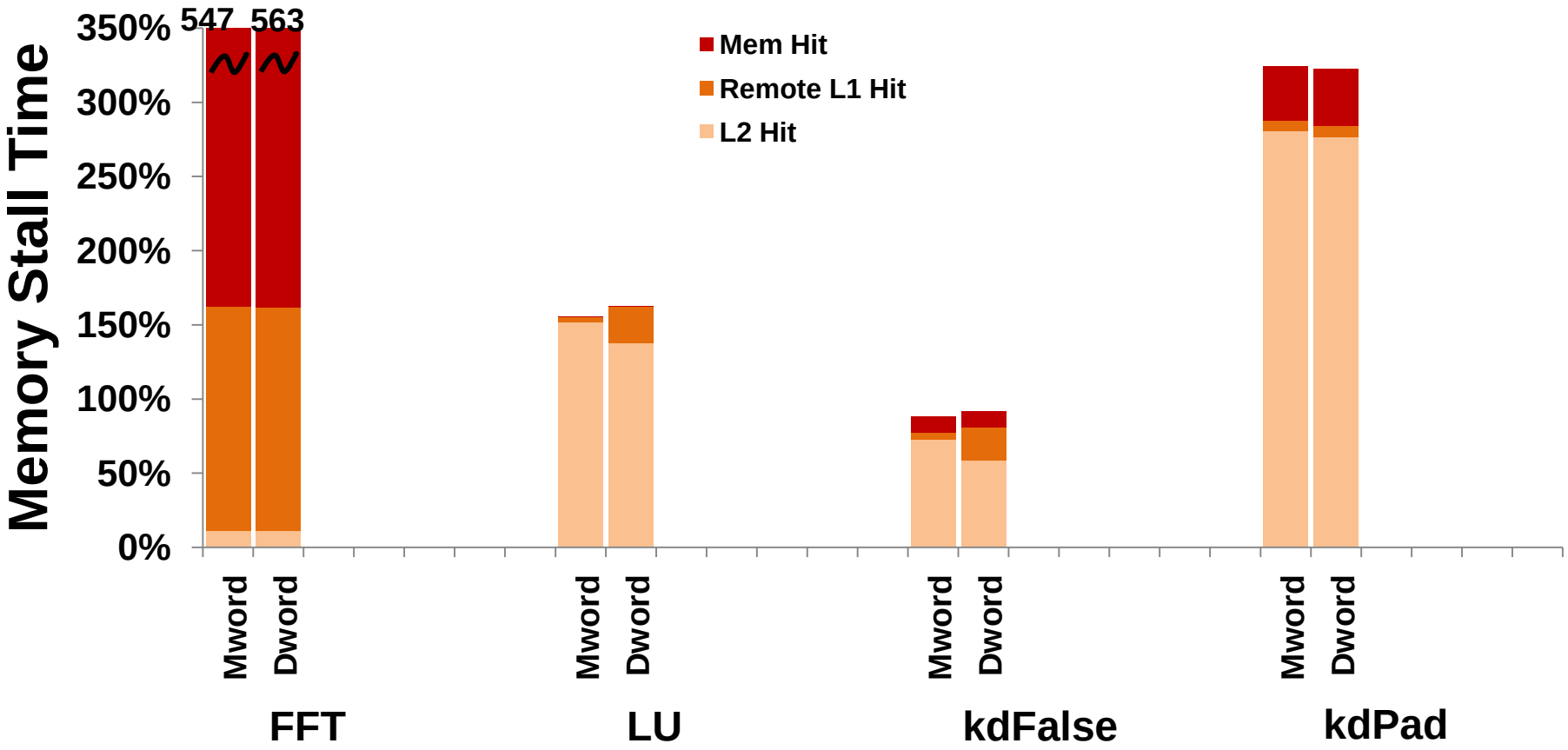
- FFT and LU from SPLASH-2

- kdTree – two versions
  - Construction of k-D trees for ray tracing
  - Developed within UPCRC, presented at HPG 2010
  - Two versions
    - kdFalse: false sharing in an auxiliary structure
    - kdPad: padding to eliminated false sharing

Simulated Protocols

- Word-based: 4B cache line
  - MESI and DeNovo
- Line-based: 64B cache line
  - MESI and DeNovo
- DeNovo extensions (proof of concept, word based)
  - DeNovo direct
  - DeNovo flex

Word Protocols

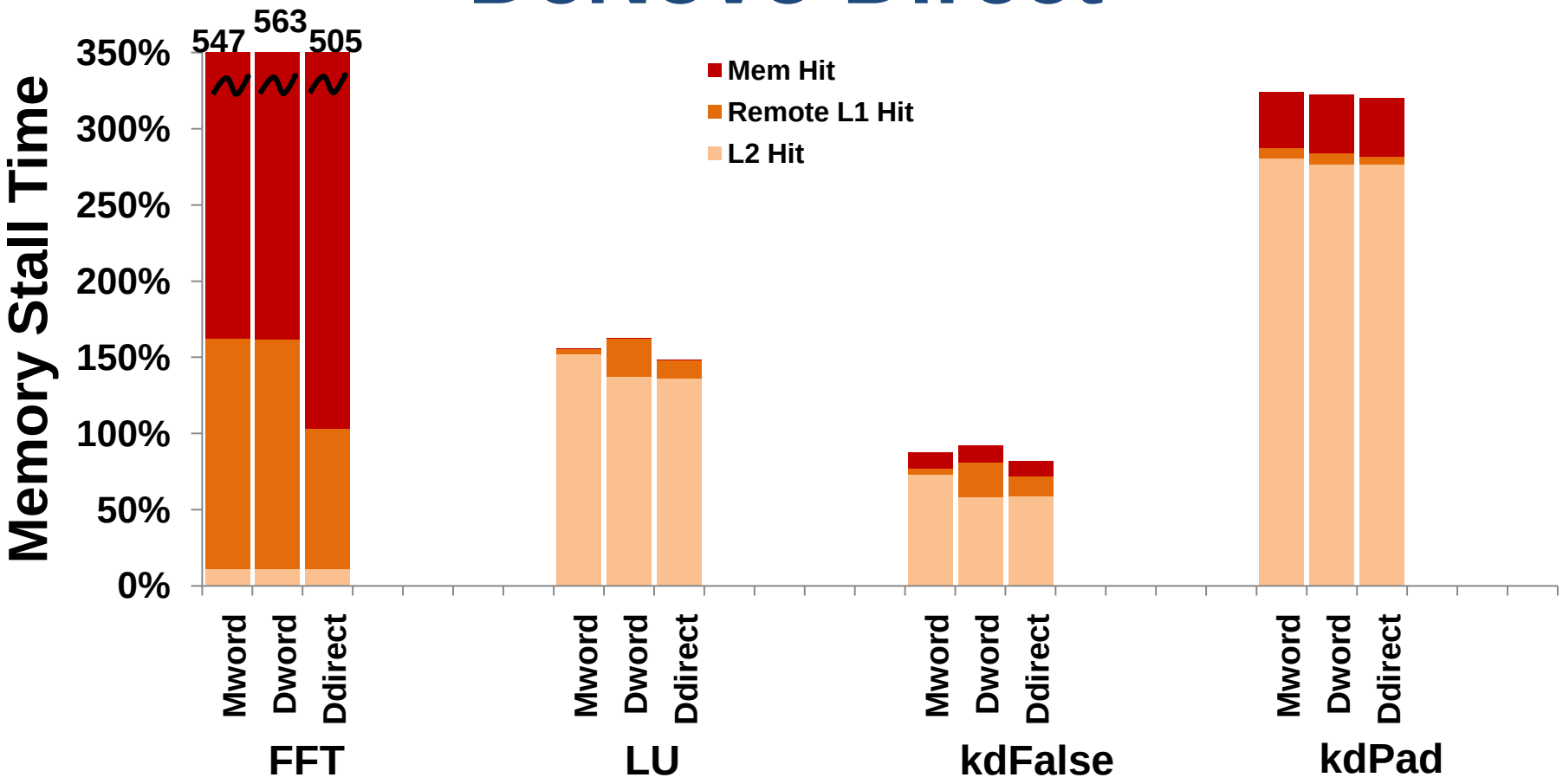


Dword comparable to Mword

Simplicity doesn't compromise performance

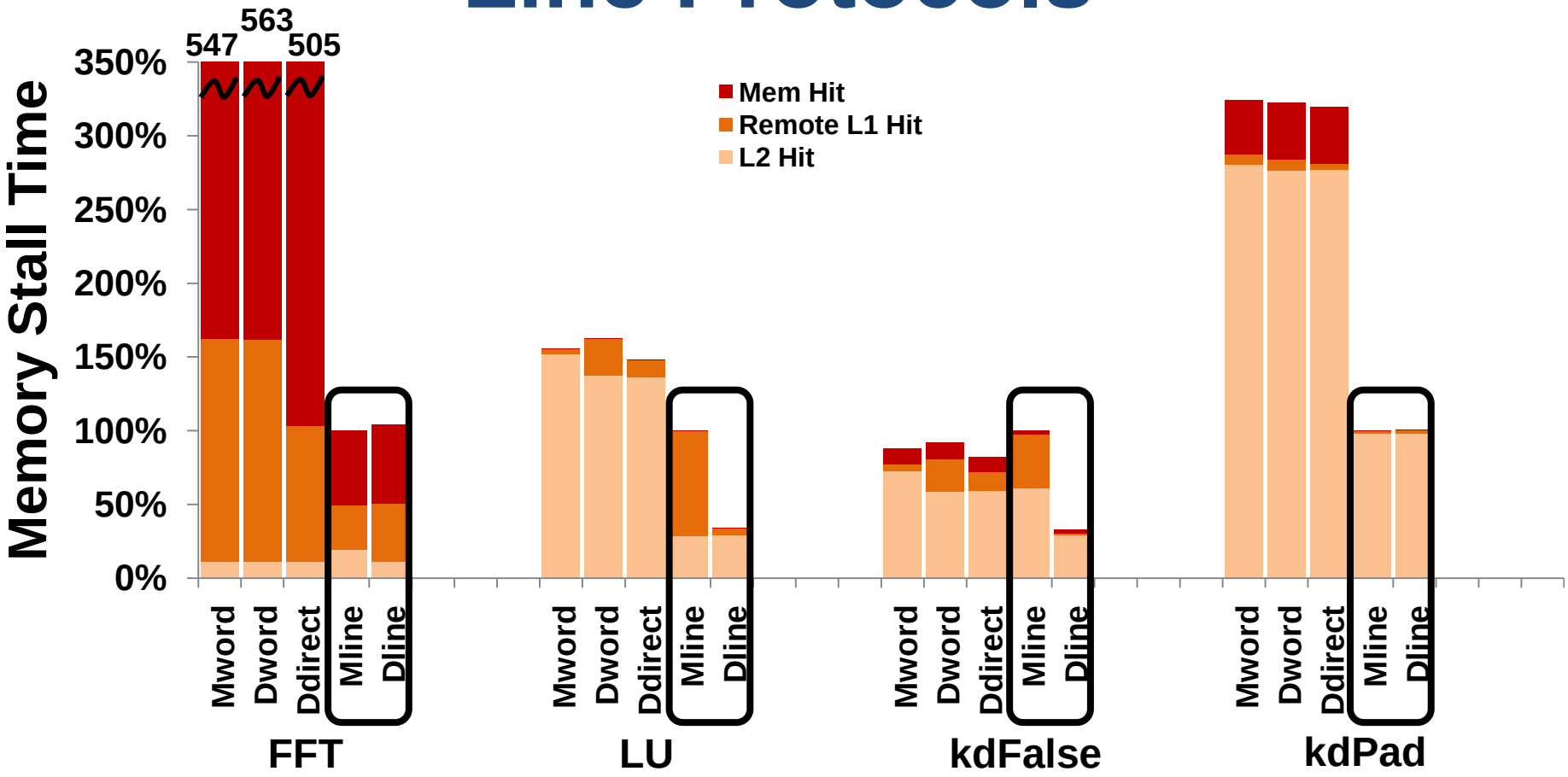
UPCRC Illinois
Universal Parallel Computing Research Center

DeNovo Direct



Ddirect reduces remote L1 hit time

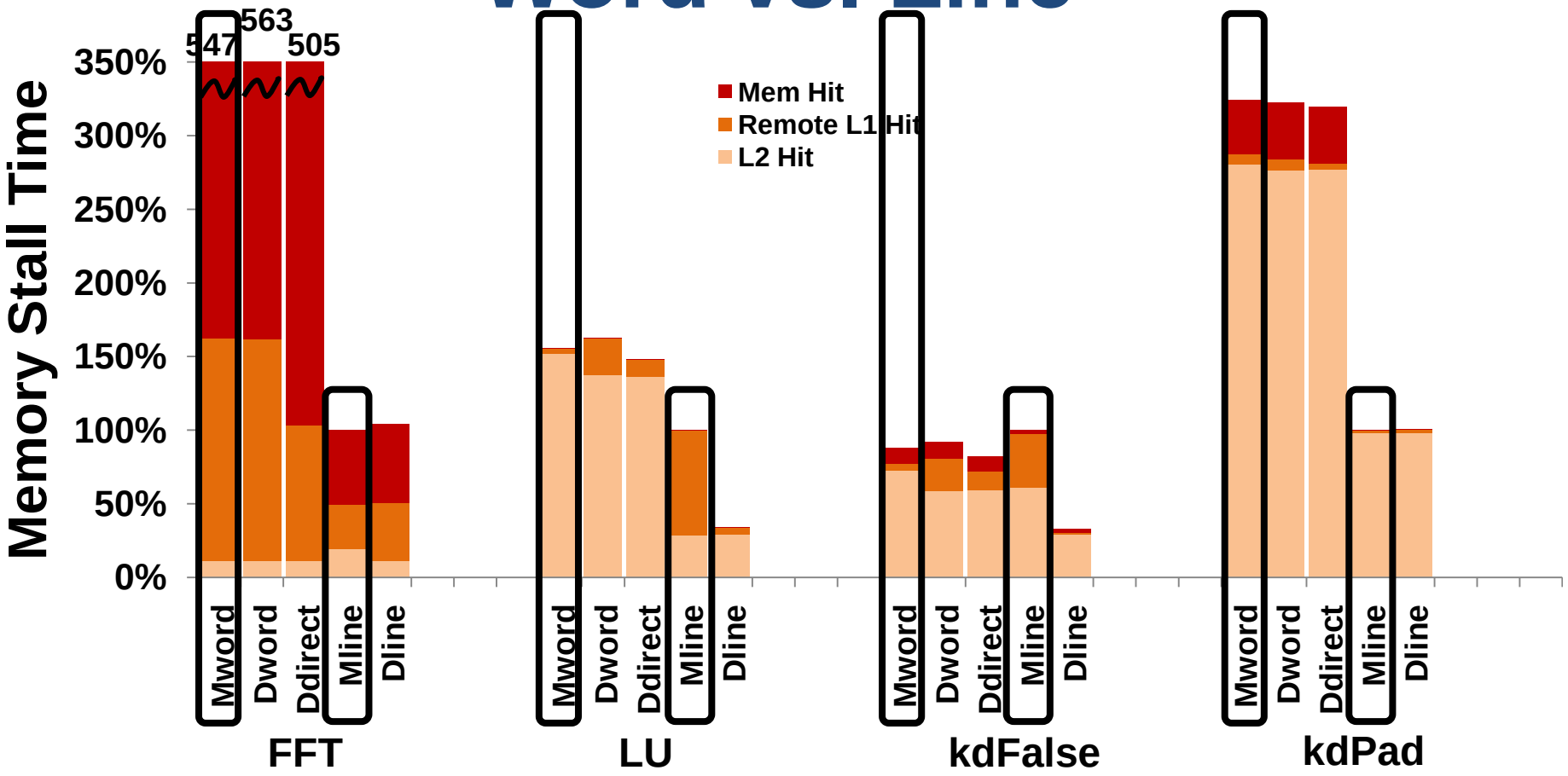
Line Protocols



Dline not susceptible to false-sharing

Up to 66% reduction in total time

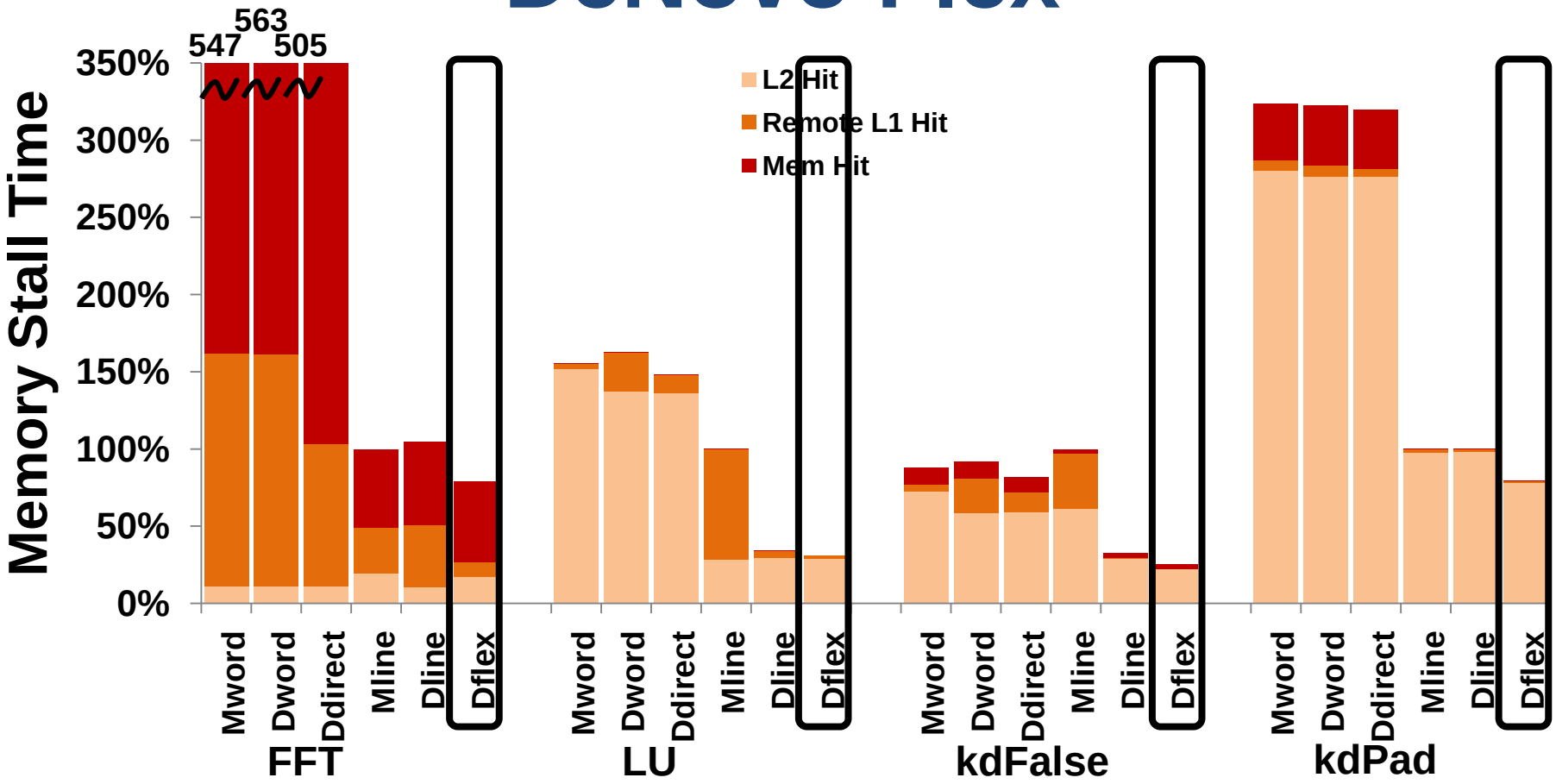
Word vs. Line



Application dependent benefit

kdFalse – Mline worse by 12%

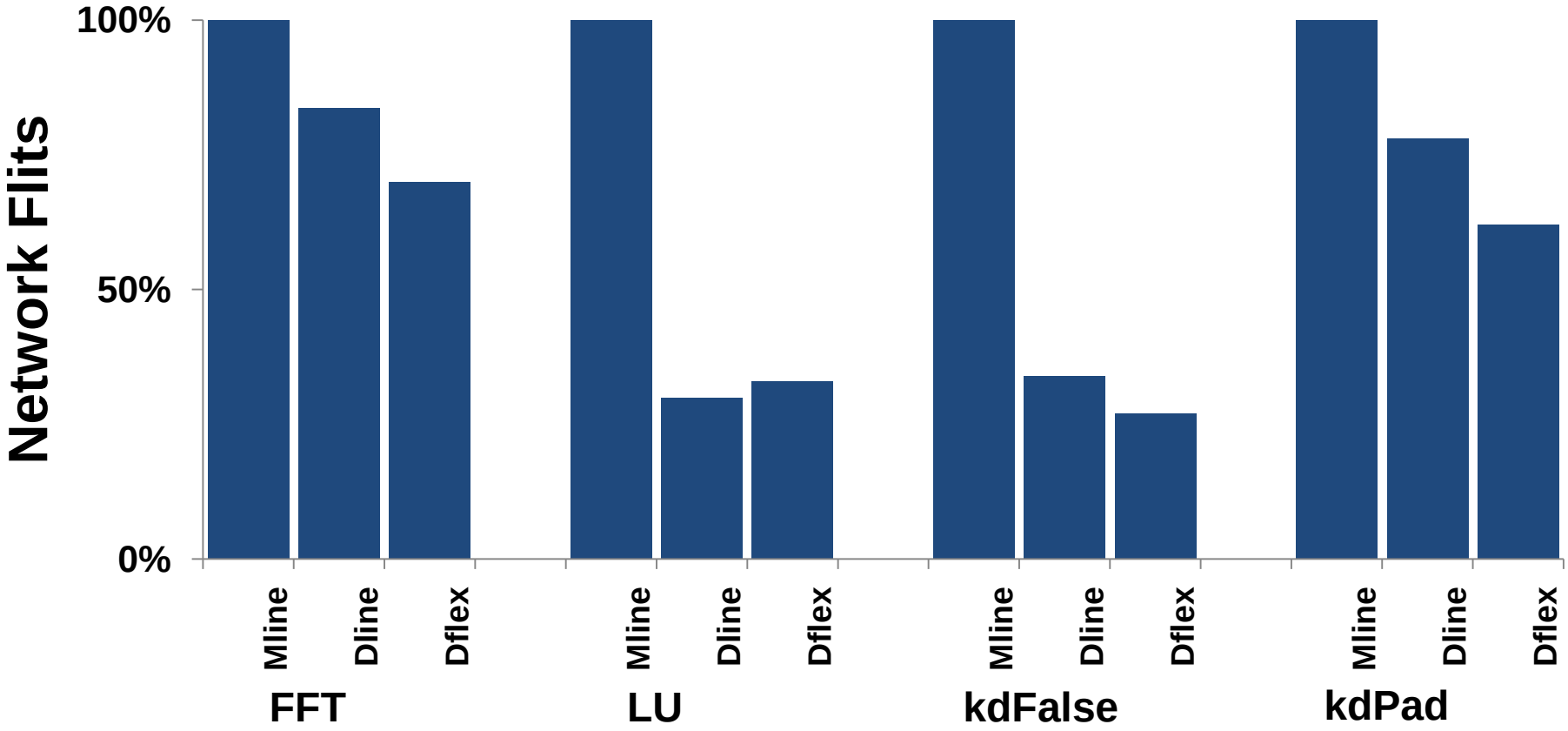
DeNovo Flex



Dflex outperforms all systems

Up to 73% reduction over Mline

Network Traffic



Dline and Dflex less n/w traffic than Mline

Up to 70% reduction

Conclusion

- Disciplined programming models key for software
  - DeNovo rethinks hardware for disciplined models
- Simplicity
  - 25x fewer reachable states with model checking
  - 30x runtime difference
- Extensibility
  - Direct cache-to-cache transfer
  - Flexible communication granularity
- Storage overhead
  - No storage overhead for directory information
  - Storage overheads beat MESI after tens of cores
- Performance/Power
  - 73% less time spent in memory requests
  - 70% reduction in n/w traffic

Future Work

- Rethinking cache data layout
- Extend to
  - Disciplined non-deterministic codes
  - Synchronization
  - Legacy codes
- Extend to off-chip memory
- Automate generation of hardware regions
- More extensive evaluations

Thank You!

Backup Slides

Flexible Coherence Granularity

- Byte-level sharing is uncommon
  - None of apps we studied so far
- Handle it correctly but not necessarily efficiently
  - Compiler aligns byte-granularity regions at word boundaries
  - If fails, H/W "clone" the line into 4 cache frames
    - With at least 4-way associativity, all reside in the same set