Software-Defined Memory Hierarchies: Scalability and QoS in Thousand-Core Systems

Daniel Sanchez
MIT CSAIL

IAP Meeting
May 21, 2013

Massachusetts Institute of Technology

Research Agenda

- Lack of technology progress
  - Moore's Law still alive
  - Power & frequency walls

} **Architecture renaissance**
Parallelism → multicore
Specialization → heterogeneity

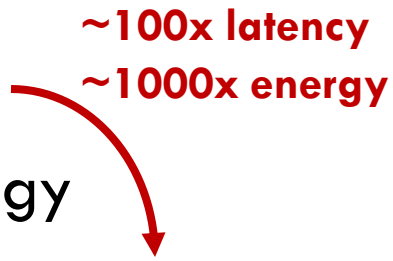
- Focus: General-purpose 1000-core heterogeneous systems
  - HW hard to scale, shared resources cause interference, no QoS
  - SW hard to scale, need pervasive parallelism
- Approach:
  - Consider full SW-HW stack – Most crucial issues span the SW/HW boundary
  - Combine analytical models and experimentation to design analyzable components that provide performance guarantees

Outline

- Introduction
- Software-Defined Memory Hierarchies
- Jigsaw: Software-Defined Caches
- Ubik: Strict QoS in Shared Datacenter Servers (brief)

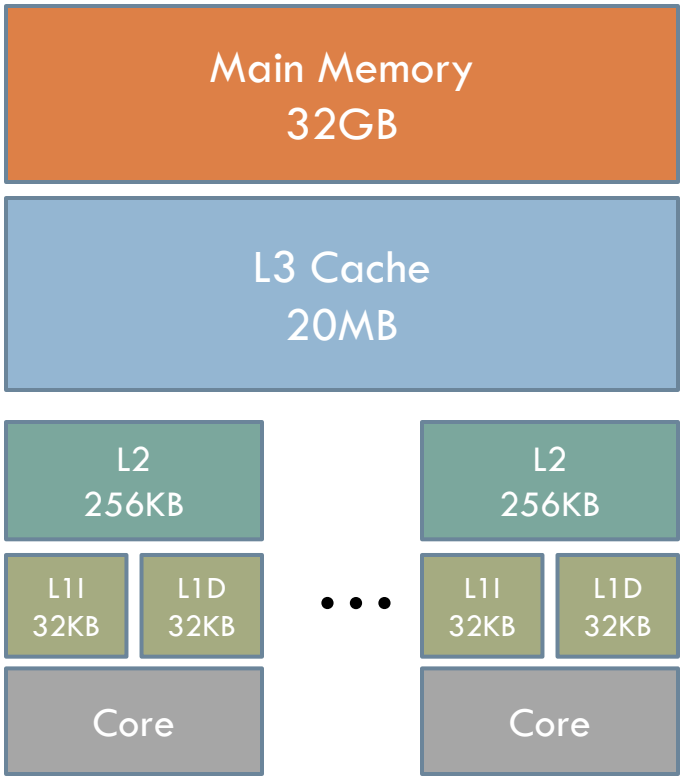
Motivation

- Current memory systems already limit system performance and energy efficiency
  - 64-bit FP op: ~1ns latency, ~20pJ energy
  - Shared cache access: ~10ns latency, ~1nJ energy
  - Off-chip DRAM access: ~100ns latency, ~30nJ energy



- Heterogeneous/simple cores improve compute performance & efficiency by 10-100x
  - Need similar improvement in the memory hierarchy!
  - Some promise: Emerging memory technologies (e.g., PCM)

Conventional Memory Hierarchies



Off-chip, DRAM
150 cycles, 36GB/s, 30 nJ

On-chip, dense SRAM, distributed, 26-34 cycles, 192GB/s (24GB/s/core), 1 nJ

Fast SRAM, 8 cycles, 48GB/s, 120 pJ

Fast SRAM, 4 cycles, 72GB/s, 40 pJ

8 OOO cores @ 2.6 GHz

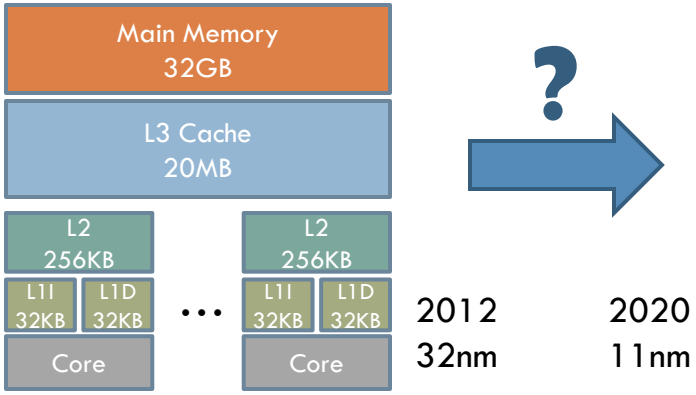
Slow, high energy, low BW
Dense, cheap

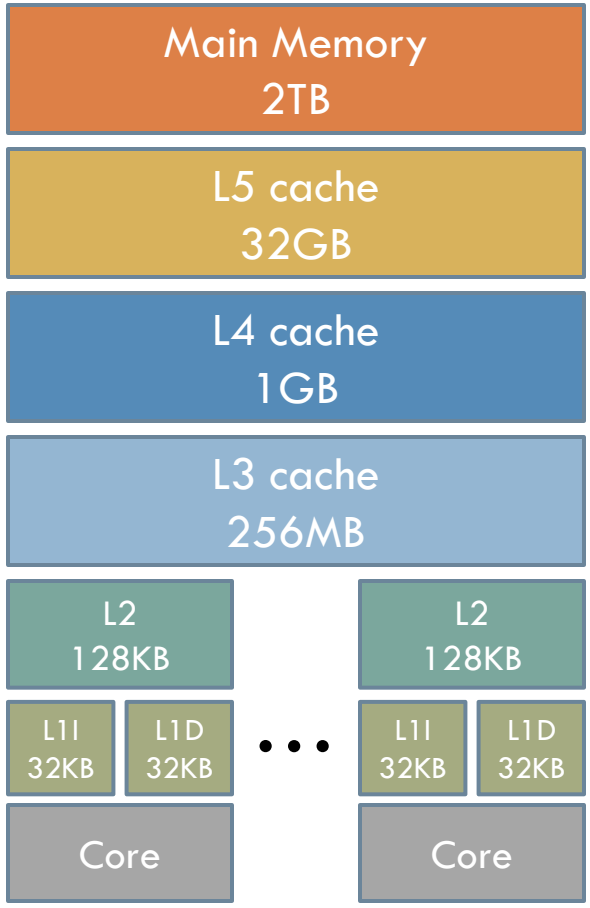
Fast, low energy, high BW
Small, expensive

- Cache-based hierarchies combine multiple memory technologies with different tradeoffs to *emulate* a large, fast, cheap, energy-efficient memory

Few levels
Simple tradeoffs

Future: Deeper Hierarchies?





Off-chip, PCM
500 cycles, 100GB/s

Off-chip, DRAM, page cache
130 cycles, 200GB/s

Die-stacked, eDRAM
50 cycles, 500GB/s

On-chip, dense SRAM, distrib,
20-96 cycles, 2.4TB/s

Fast SRAM, 6 cycles, 48GB/s

Fast SRAM, 3 cycles, 72GB/s

1024 simple/heterogeneous
cores @ 2 GHz

- Complex tradeoffs (lat, BW, energy, reads vs writes)
- Distributed caches blur the notion of a level
- Adding levels often hurts performance and efficiency
- Uncontrolled interference in shared levels → No QoS

Deep, rigid hierarchies do not scale

Future: Deeper Hierarchies?

- Deep, rigid hierarchies do not scale
- Can we fix this in HW?

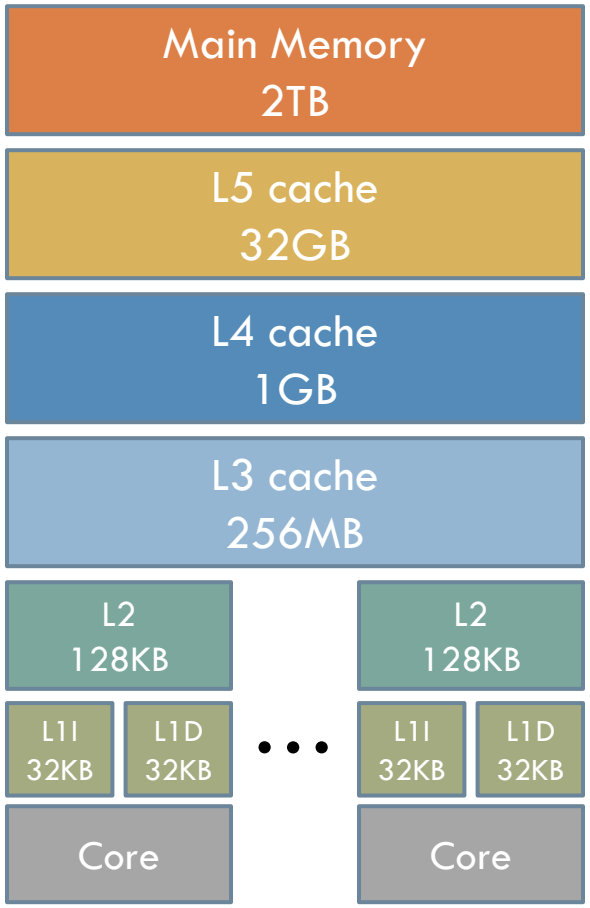
Speculative next-level accesses

Hit/miss prediction

Non-Uniform Cache Access techniques

- ✘ Complex, trade energy & bandwidth for latency
- ✘ Often require prediction, pred state ~ O(size)
- ✘ **Best-effort, unpredictable**

Must put SW in the loop



Off-chip, PCM
500 cycles, 100GB/s

Off-chip, DRAM, page cache
130 cycles, 200GB/s

Die-stacked, eDRAM
50 cycles, 500GB/s

On-chip, dense SRAM, distrib,
20-96 cycles, 2.4TB/s

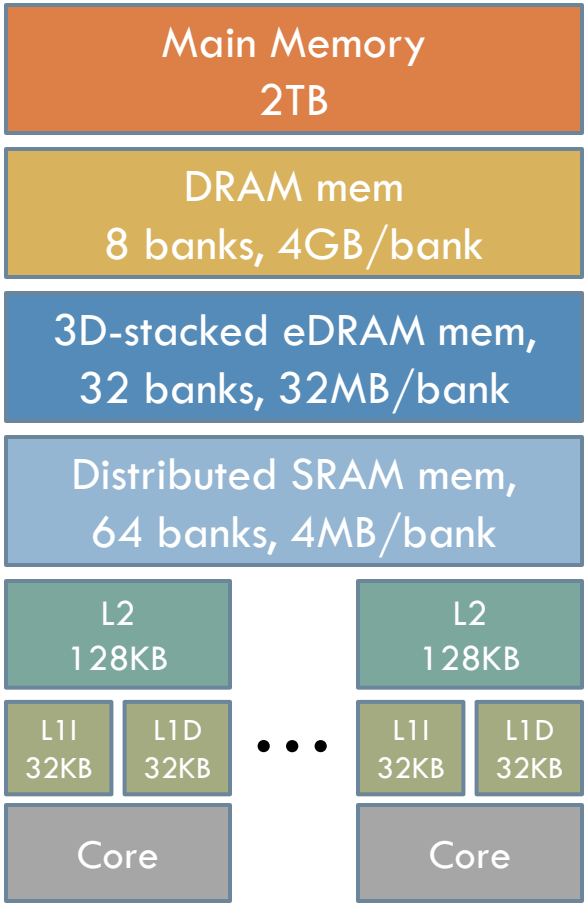
Fast SRAM, 6 cycles, 48GB/s

Fast SRAM, 3 cycles, 72GB/s

1024 simple/heterogeneous cores @ 2 GHz

Future: Back to Scratchpads?

- We've tried this before…
- Excessive programmer complexity
- Hard to virtualize
- Similar performance to (shallow) cache-based hierarchies [Leverich et al, ISCA07]



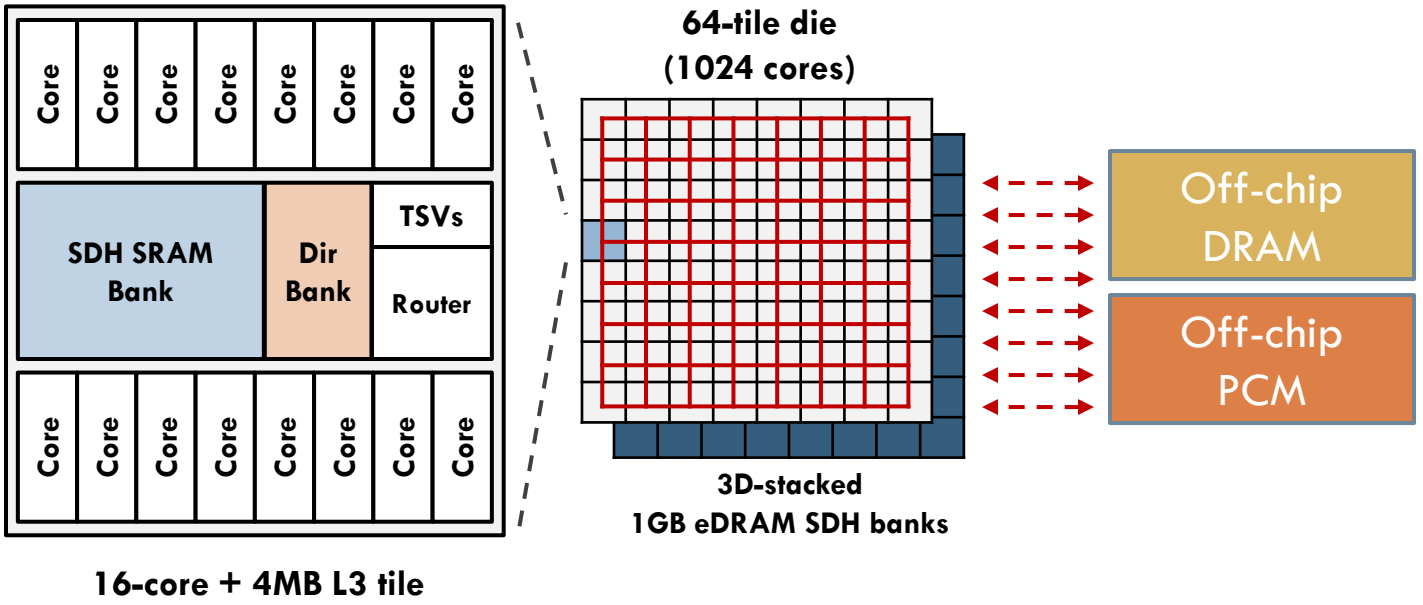
Outline

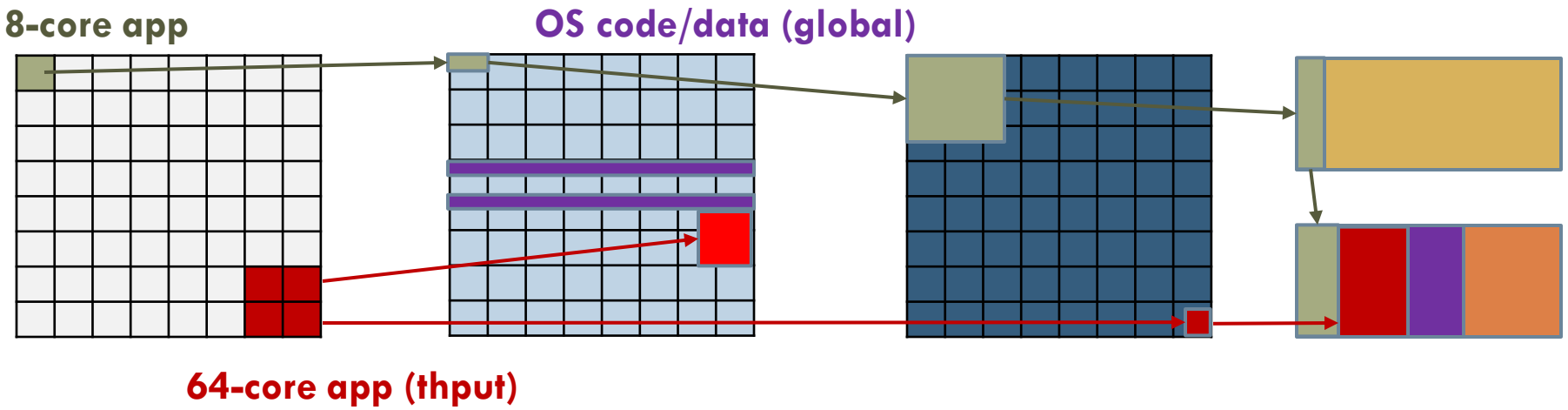
- Introduction
- Software-Defined Memory Hierarchies
- Jigsaw: Software-Defined Caches
- Ubik: Strict QoS in Shared Datacenter Servers

Software-Defined Memory Hierarchies

- Goal: 10-100x memory system performance & efficiency
- Insight: Software has much more semantic information about how it uses memory than hardware
- Approach: Instead of hiding performance and energy tradeoffs, expose them to software *efficiently*
- Idea: Expose a *flat* collection of on-chip and off-chip caches and memories, and allow software to define multiple logical cache and memory hierarchies over them
  - Explicit tradeoffs (latency, energy, bandwidth, capacity)
  - Implicit data placement and movement → no programmer burden

Software-Defined Hierarchy Example





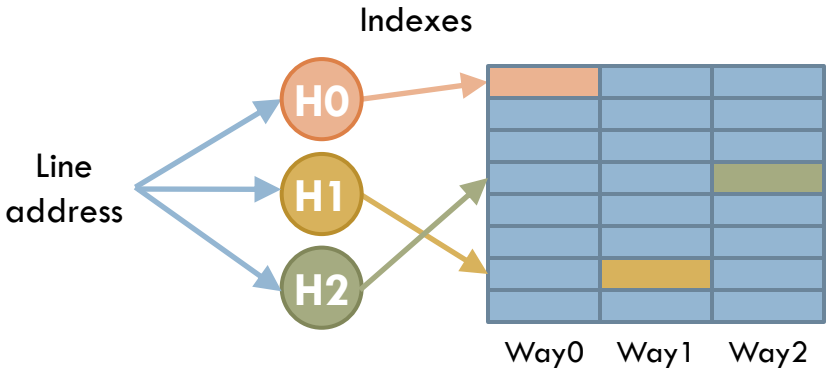
SDH: Necessary Components

- HW: Efficient memory components
  - Efficient caches (as close to scratchpads as possible)
  - Efficient cache virtualization
  - Scalable cache coherence (flat → cannot leverage hierarchy!)
- HW/SW: Efficient monitoring and reconfiguration
  - Converge to optimal configuration quickly, no trial-and-error
- SW: Efficient runtimes
  - Scalable management algorithms
  - OS-level runtime: Global arbitration, no user-level changes
  - Application-level runtimes (e.g., Cilk, Streamlt, GRAMPS): Leverage high-level semantic information (e.g., producer-consumer) to provide further gains

Efficient Caches: ZCache

- Conventional set-associative caches are far from ideal:
  - Reducing conflicts (higher associativity) → more ways
    - High energy, latency, area overheads
  - Conflicts depend on workload's access patterns

- ZCache: A highly-associative cache with a small number of ways
  - Hits take a single lookup
  - In a miss, replacement process provides many replacement candidates



  - Provides cheap high associativity (e.g., associativity equivalent to 64 ways with a 4-way cache)
  - Achieves analytical guarantees on associativity

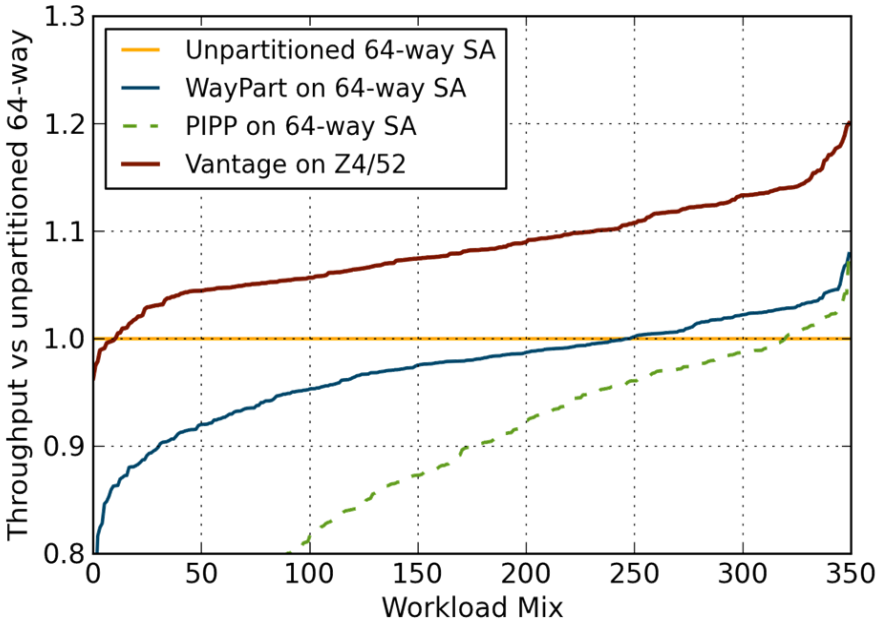
[MICRO 2010] "The ZCache: Decoupling Ways and Associativity"

Efficient Cache Virtualization: Vantage

- Leverage statistical guarantees of ZCache to design scalable partitioning
  - Hundreds of partitions per cache, defined at fine granularity (cache lines)
  - Strict guarantees on partition sizes and isolation
  - Extremely fast reconfigurations
  - Minimal overheads: 1.5% state for 64 partitions, negligible additional logic

Vantage improves throughput, uses an efficient 4-way cache

Conventional techniques hurt throughput, require inefficient (64-way) caches



[ISCA 2011] "Vantage: Scalable and Efficient Fine-Grain Cache Partitioning"
[Top Picks 2012] "Scalable and Efficient Fine-Grained Cache Partitioning with Vantage"

Scalable Coherence: SCD

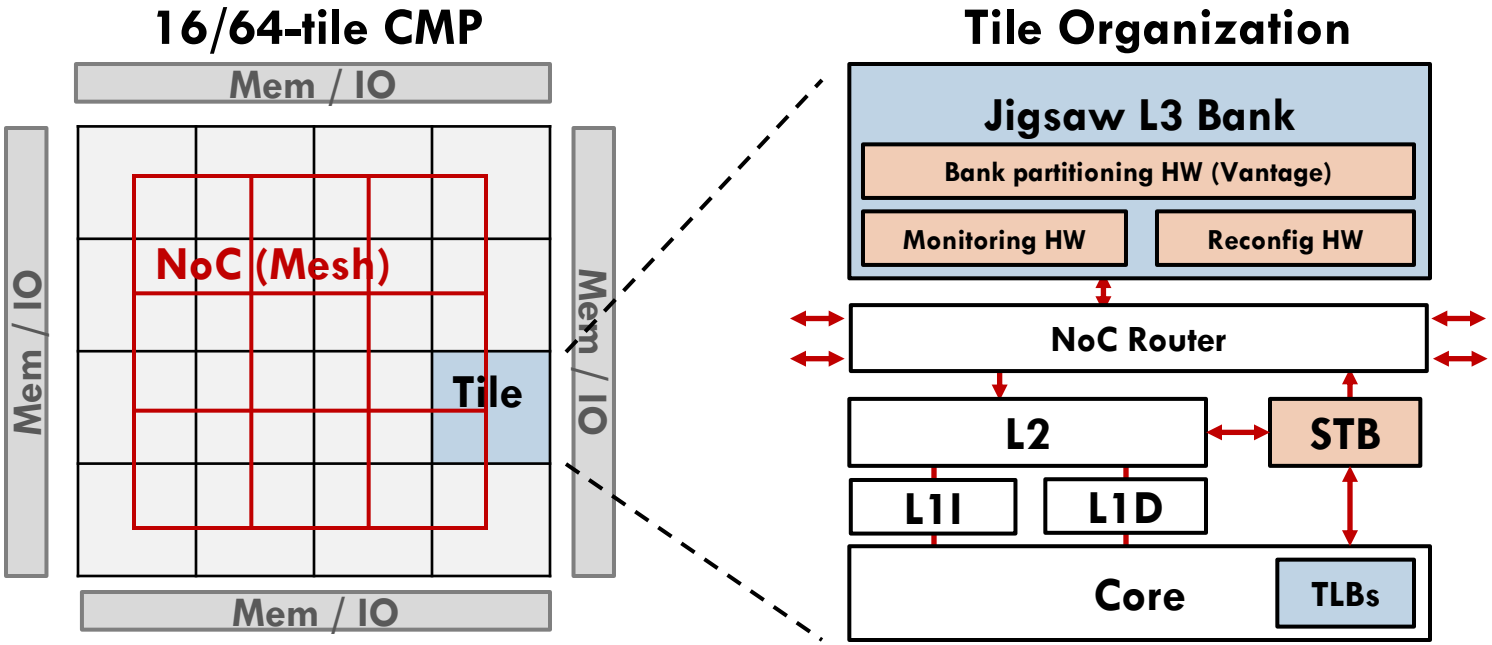
- Scaling conventional directories is hard:
  - Excessive latency, energy, area overheads, or too complex
  - Introduce invalidations → Interference
- Insights:
  - Flexible sharer set encoding: Lines with few sharers use one entry, widely shared lines use multiple entries → Scalability
  - Use ZCache → Efficient high associativity, analytical models
    - Negligible invalidations with minimal overprovisioning (~10%)
- SCD achieves scalability and performance guarantees
  - Area, energy grow with log(cores), constant latency
  - Simple: No modifications to coherence protocol
  - At 1024 cores, SCD is 13x smaller than a sparse directory, 2x smaller, faster and simpler than a hierarchical directory

[HPCA 2012] "SCD: A Scalable Coherence Directory with Flexible Sharer Set Encoding"

Outline

- Introduction
- Software-Defined Memory Hierarchies
- Jigsaw: Software-Defined Caches
- Ubik: Strict QoS in Shared Datacenter Servers

First Prototype: Jigsaw



- Apply basic SDH concepts to manage a last-level cache
- 16/64 1MB LLC banks, 64 partitions/bank
- Software combines individual partitions to form shares (virtual caches)
  - e.g., a 512KB share for private data taking 50% of the local bank
  - e.g., a 2MB share for shared data spanning 25% of 8 banks
  - Latency vs capacity tradeoff fully exposed to SW

[PACT 2013] "Jigsaw: Scalable Software-Defined Caches", Beckmann and Sanchez (to appear)

Jigsaw Runtime

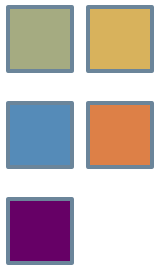
- Can software manage distributed shared caches better than state-of-the-art HW schemes?

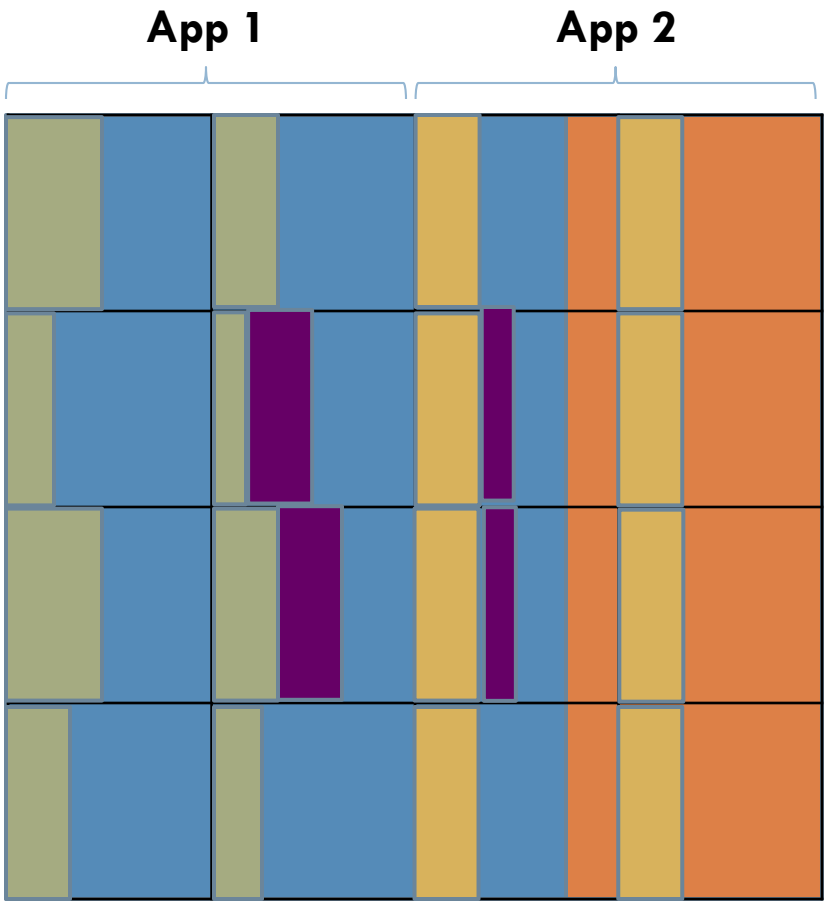
- Prototype OS-level runtime:
  - Maps data (pages) to shares: Per-thread, per-process, global
  - Places & sizes shares across physical banks to co-optimize latency and miss rate
  - Monitors & reconfigures periodically (with HW support)

- Required HW support:
  - Monitoring (based on distributed utility monitors)
  - Fast reconfiguration: Bulk page/selective bank invalidations

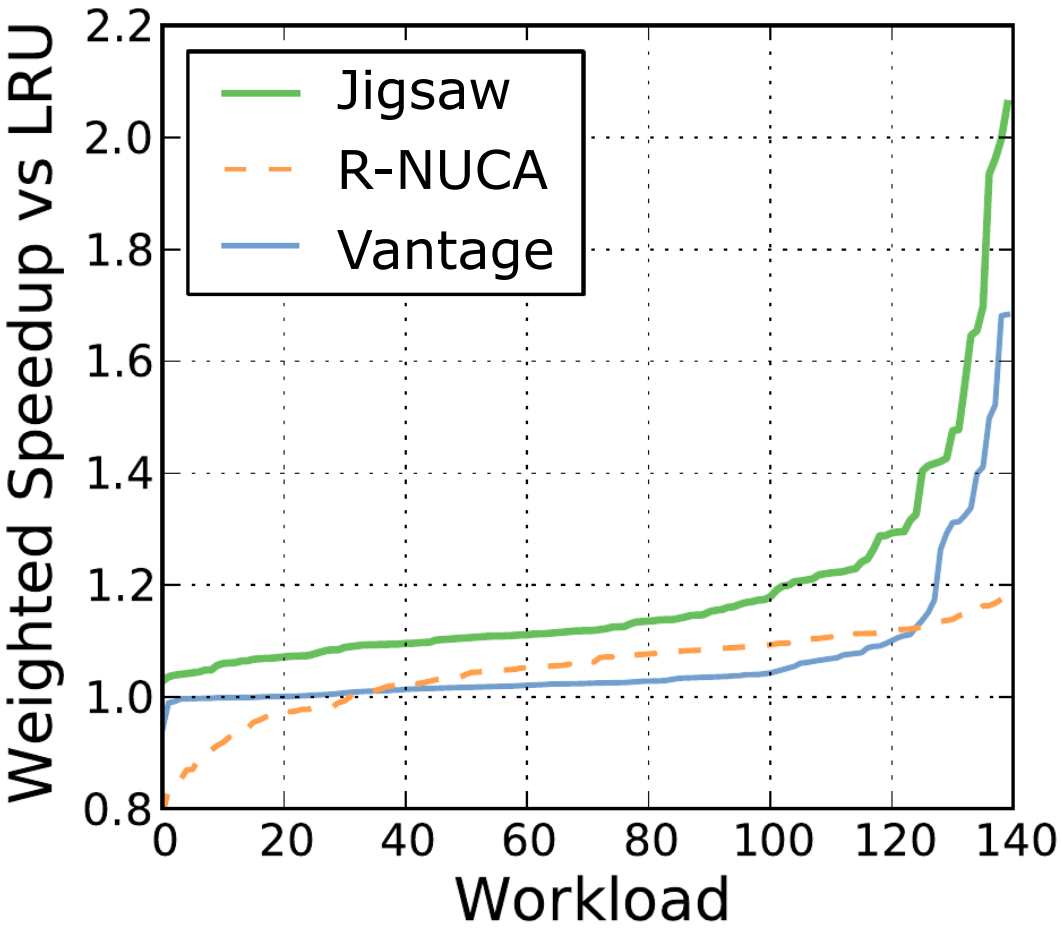
Jigsaw Example

- 2 8-thread apps
- 9 shares/app:
  - 8 per-thread shares
  - 1 per-process share
- 1 global share
- 19 shares total



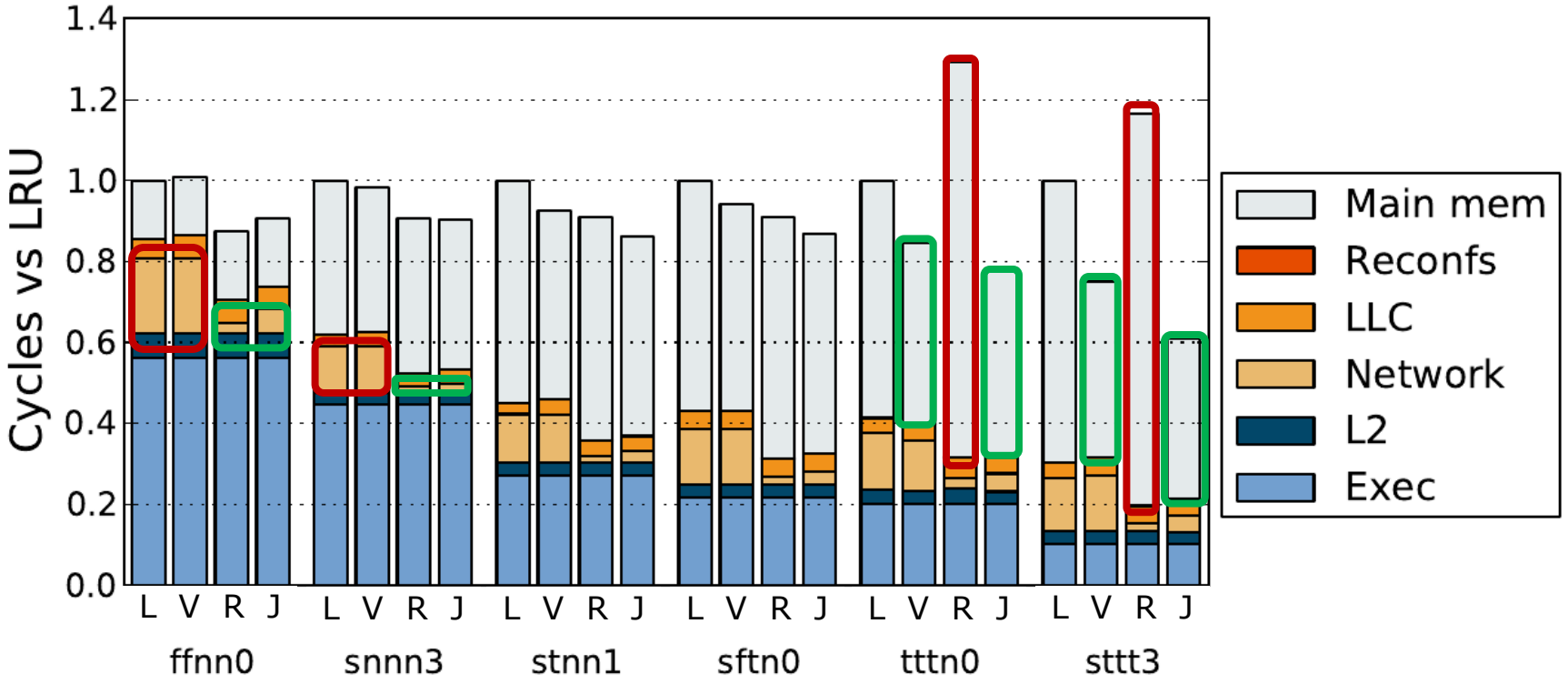


Jigsaw Performance



- 140 16-app mixes on 16-core CMP
- Jigsaw outperforms:
  - LRU by 17% (max 2.1x)
  - R-NUCA by 12% (max 2.5x)
  - Vantage by 8% (max 42%)
- Larger benefits with more cores

Jigsaw Performance Analysis



- Runtime successfully co-optimizes latency and MPKI
- Runtime overheads: 0.2% of system cycles
  - Required algorithmic innovation (non-convex opt, prior techniques 150x slower)
- Large energy savings: Network traffic, off-chip accesses

Jigsaw Summary & Future Work

- Jigsaw: Preliminary small-scale prototype
  - Single-level software-define caches over distributed LLC
  - HW+OS-level runtime shows:
    - Can achieve large gains *by exploiting SW information*
    - Can achieve small overheads *if HW&SW correctly co-designed*

- Now exploring:
  - Multi-level virtual hierarchies to exploit heterogeneous technologies & scale to1K-core systems
  - Application-level parallel runtimes that exploit SDH
  - HW and SW support for strict quality of service and isolation (e.g., co-schedule best-effort + hard real-time apps)

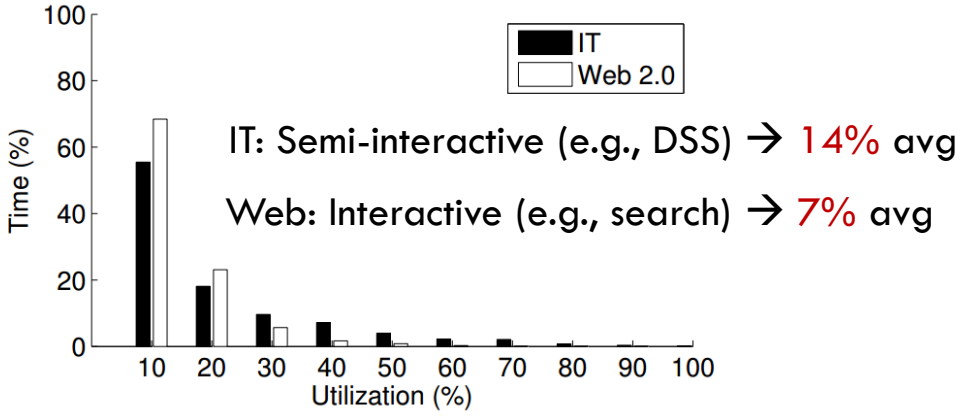
Outline

- Introduction
- Software-Defined Memory Hierarchies
- Jigsaw: Software-Defined Caches
- Ubik: Strict QoS in Shared Datacenter Servers

Motivation: QoS in Shared Multicores

- Fundamental disconnect in systems community:
  - Apps increasingly interactive, with soft real-time reqs
  - Systems still designed using long-term average metrics (IPC, QPS, fairness)
- Utilization wall in cloud computing

Server utilization histogram



[Meisner et al, ASPLOS 09 / anon HP client's traces]

Cloud apps need QoS

Current "solution"
Use a single app and one or few cores per server to avoid interference, or ignore QoS

Conventional wisdom
Low utilization in multicores is a fact of life, design servers for 10% utilization

Ubik: Efficient Strict QoS In Shared Systems

- Goal: High utilization under a mixed workload:
  - Latency-critical request-driven apps with probabilistic latency bounds (e.g., 99th pct latency < 10ms)
  - Best-effort batch apps that need good average performance

- Ubik: Rely on HW components with analytical guarantees to allow safe fine-grained sharing between latency-critical and best-effort apps
  - Tight lower bounds on performance degradation
  - Coordinated capacity and bandwidth management

Conclusions

- Need 10-100x mem performance and efficiency gains
- Hiding tradeoffs from SW has run out of steam

- Software-Defined Memory Hierarchies: Integrated hardware-software memory management
  - Efficiently expose complex memory system tradeoffs to SW
  - Develop the right HW mechanisms and SW runtimes to manage memory efficiently
  - Enable strict QoS in shared systems
  - Key enabling components developed, promising preliminary results

Thanks For Your Attention!

Questions?

Massachusetts Institute of Technology