NoC & NUCA

15-740 FALL'21

NATHAN BECKMANN

Topics

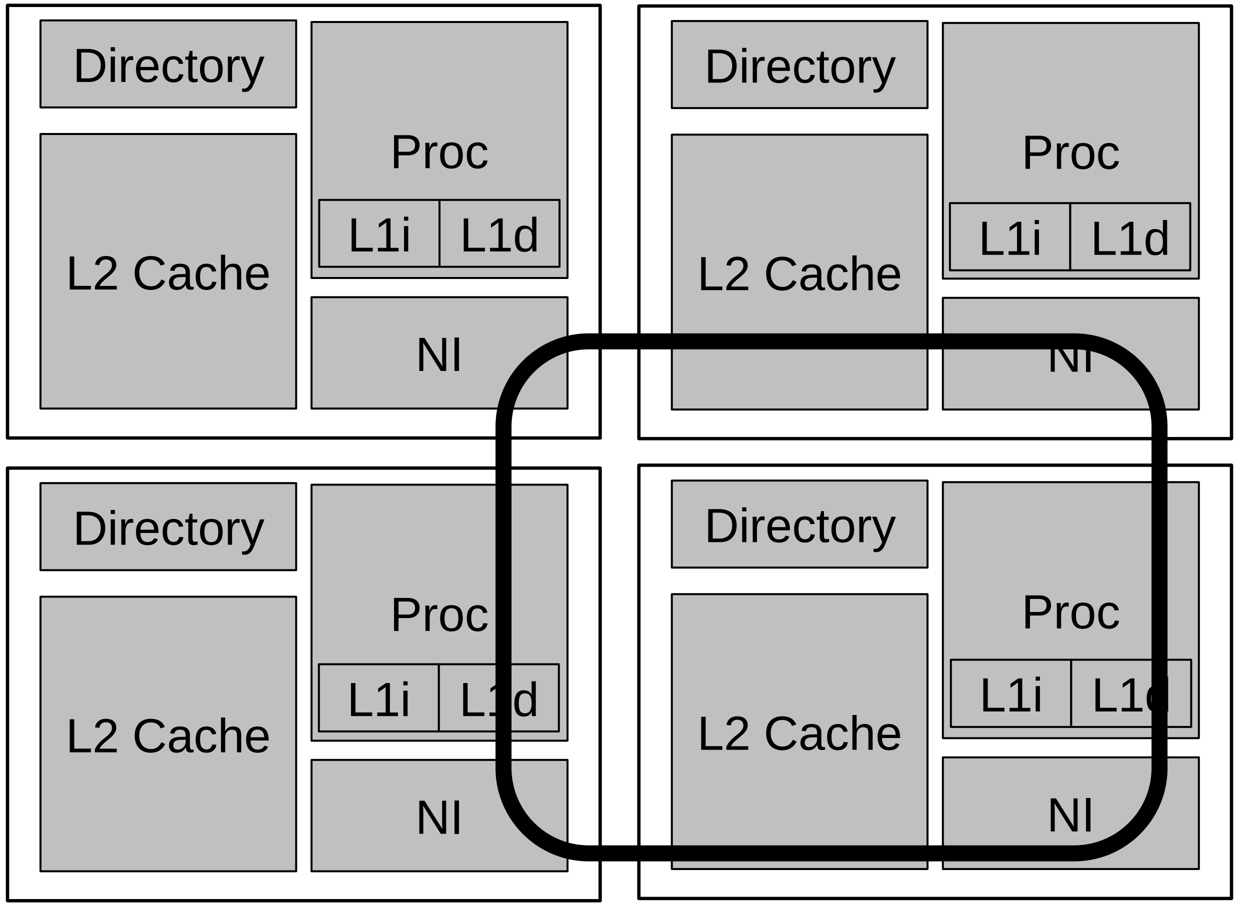
Network-on-chip (NoC)

Non-uniform cache access (NUCA)

Recall: Distributed caches in multicore

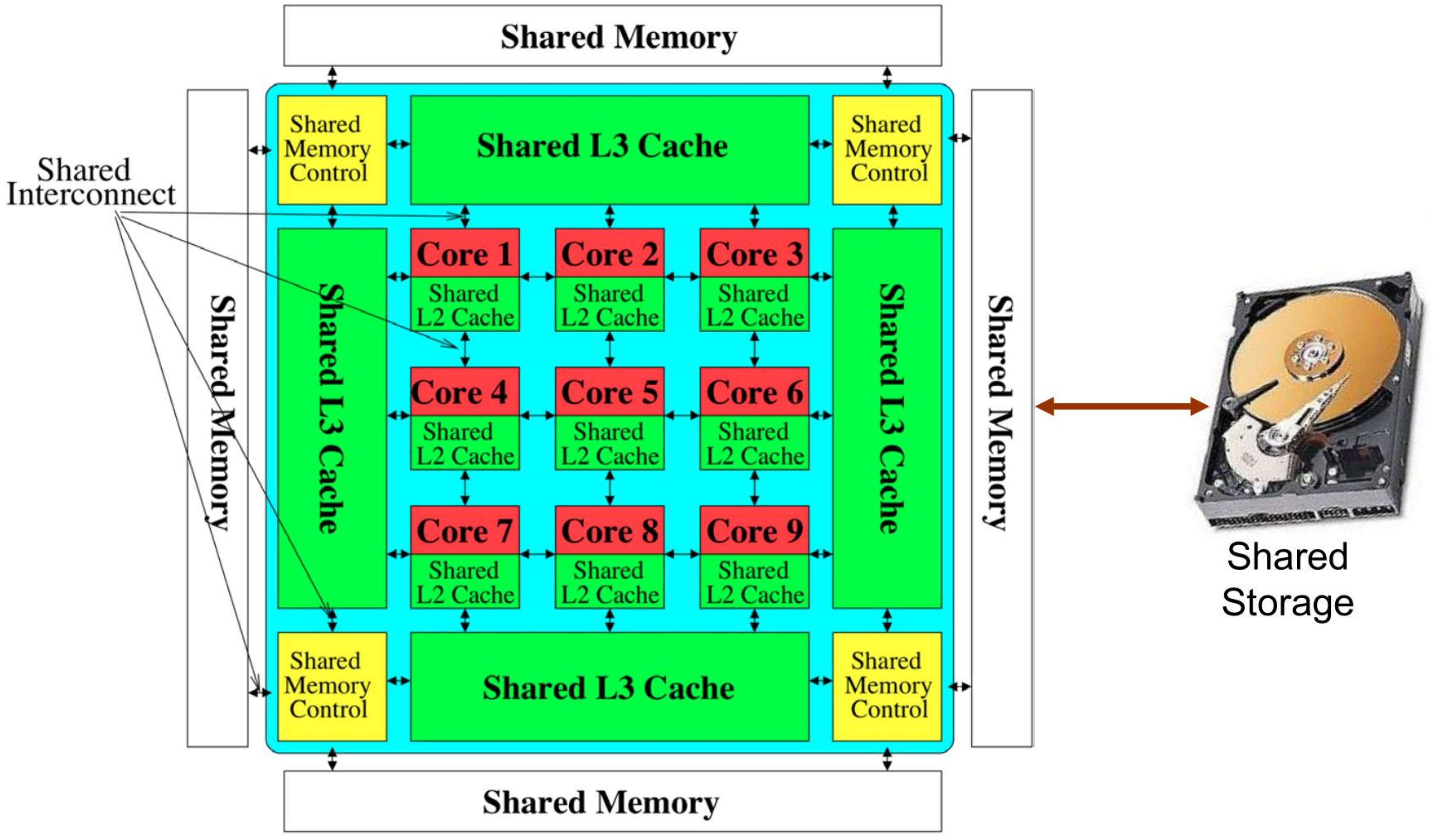
Cache banks distributed throughout chip, connected by an on-chip network

Communication arbitrated by coherence protocol



Interconnection Networks / Network-on-Chip (NoC)

On-chip interconnect in a multicore



Where is interconnect used?

To connect components

- Processor-to-processor
- Processor-to-cache
- Cache-to-cache
- Cache-to-memory
- I/O-to-memory
- Etc.

Typically, there are **multiple networks** for different communication

- Optimize NoC design for different communication patterns
- Avoid deadlock

Why is interconnect important?

Affects **scalability** of the system

- How large a system can you build?
- How easily can you add more processors/caches?

Affects performance & energy efficiency

- How fast can processors, caches, memories communicate? (longer than cache access)
- How much energy is spent on communication? (10-35%)

Interconnect basics

Topology
- How switches are wired to each other
- Affects routing, reliability, throughput, latency, cost

Routing (algorithm)
- How does a message get from source to destination?
- Static vs adaptive

Buffering and flow control
- What do we store within the network? (Packets, headers, …?)
- How do we throttle when oversubscribed?
- Tightly coupled with routing

Router & link microarchitecture
- Determines energy & delay of each network hop

Interconnect metrics

Latency (hops, cycles, nanoseconds)

Energy

Bandwidth (typically, “bisection” bw)

Cost (area)

Contention

End-to-end system performance

Best NoC depends on app & system

What matters?

Broadcast
- E.g., snoopy coherence

Point-to-point communication
- E.g., directory-based coherence

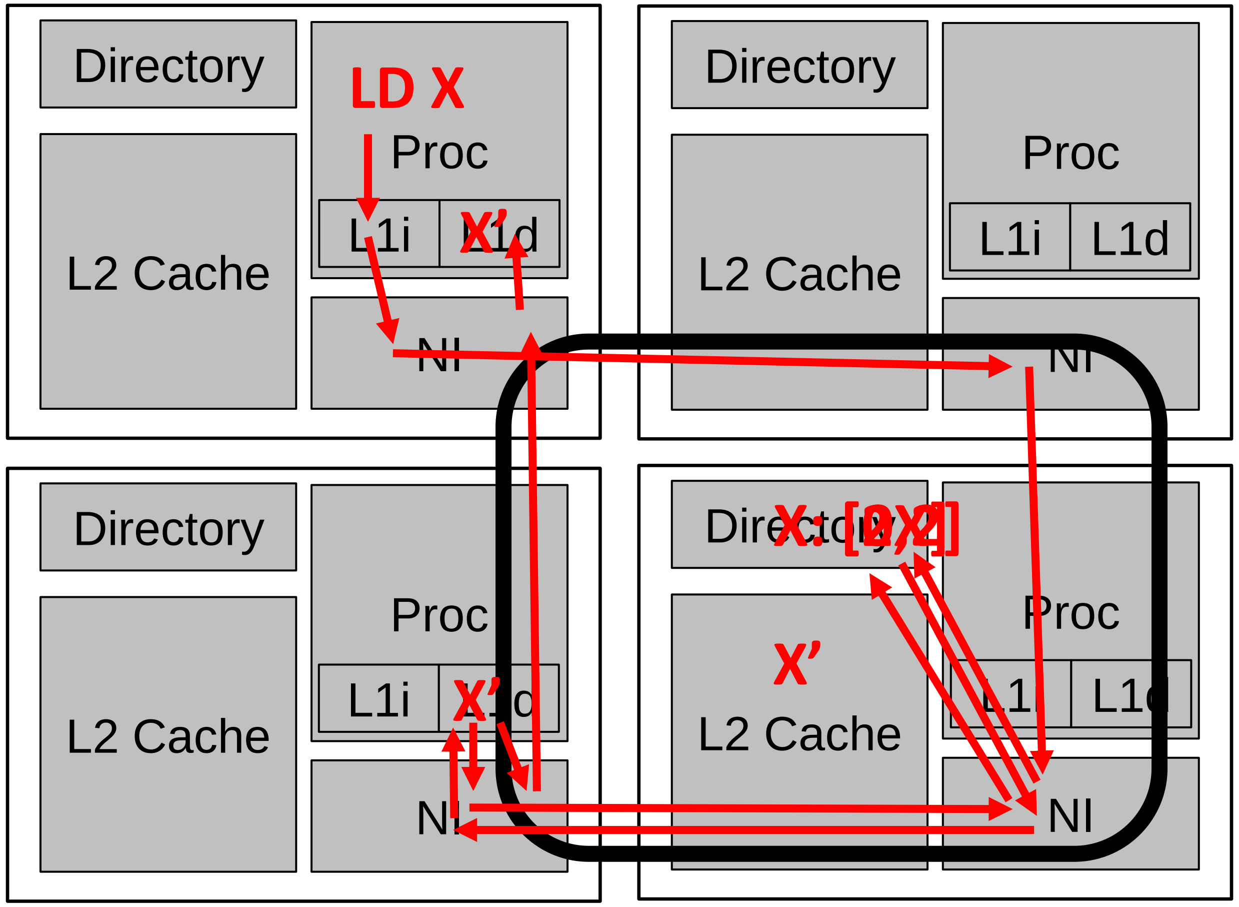
Latency vs bandwidth
- E.g., CPU vs GPU

Scalability
- E.g., 4-core vs 64-core system

An example

Processor 2 has cached a modified value of X

Processor 0 reads X



NoC Topologies

Interconnect topologies

Bus (simplest)

Point-to-point (ideal and most costly)

Crossbar (less costly)

Ring

Mesh

Tree

Omega

Hypercube

Torus

Butterfly

…

Bus

+ Simple

+ Cost-effective for small number of nodes

+ Easy to implement coherence (global broadcast)

- Poor scalability (electrical limitations)

- High contention

Point-to-point

Every node connected directly to every other

+ Lowest contention

+ Lowest latency (maybe—wire length, wasted area)

+ Ideal except for cost

- Highest cost
  - $O(N^2)$ links

- Not scalable

- Physical layout??

Crossbar

Every node connected to every other, but only one at a time

Concurrent communication to different destinations

Good with few nodes

+ Low latency & high throughput

- Expensive

- Doesn’t scale -- $O(N^2)$ switches

- Difficult to arbitrate with many nodes

Used in many designs (e.g., Sun UltraSPARC T1)

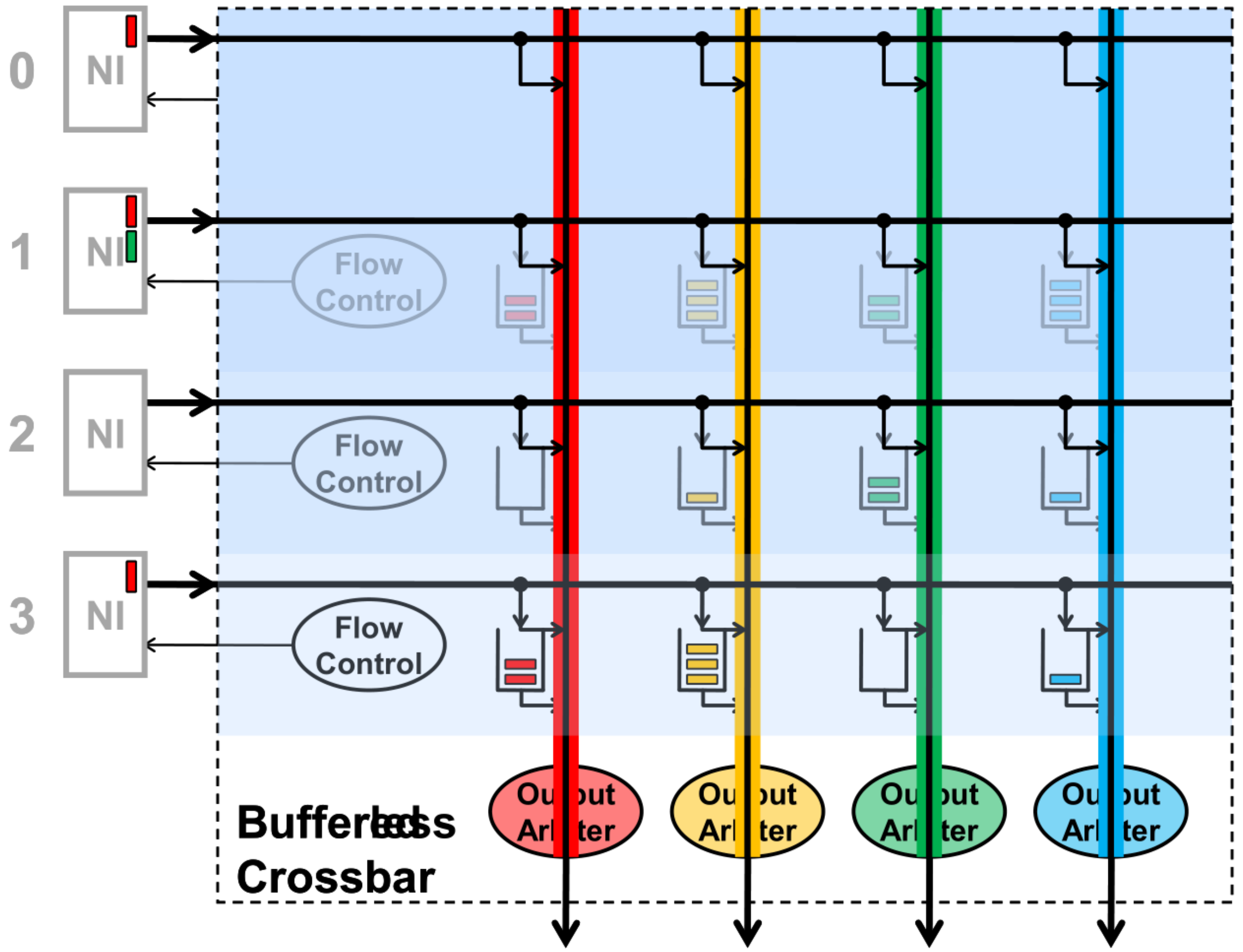
Buffered crossbar

+ Simpler arbitration & scheduling

+ Efficient support for variable sized packets

- Requires $O(N^2)$ buffers

Can we scale the interconnect without contention?

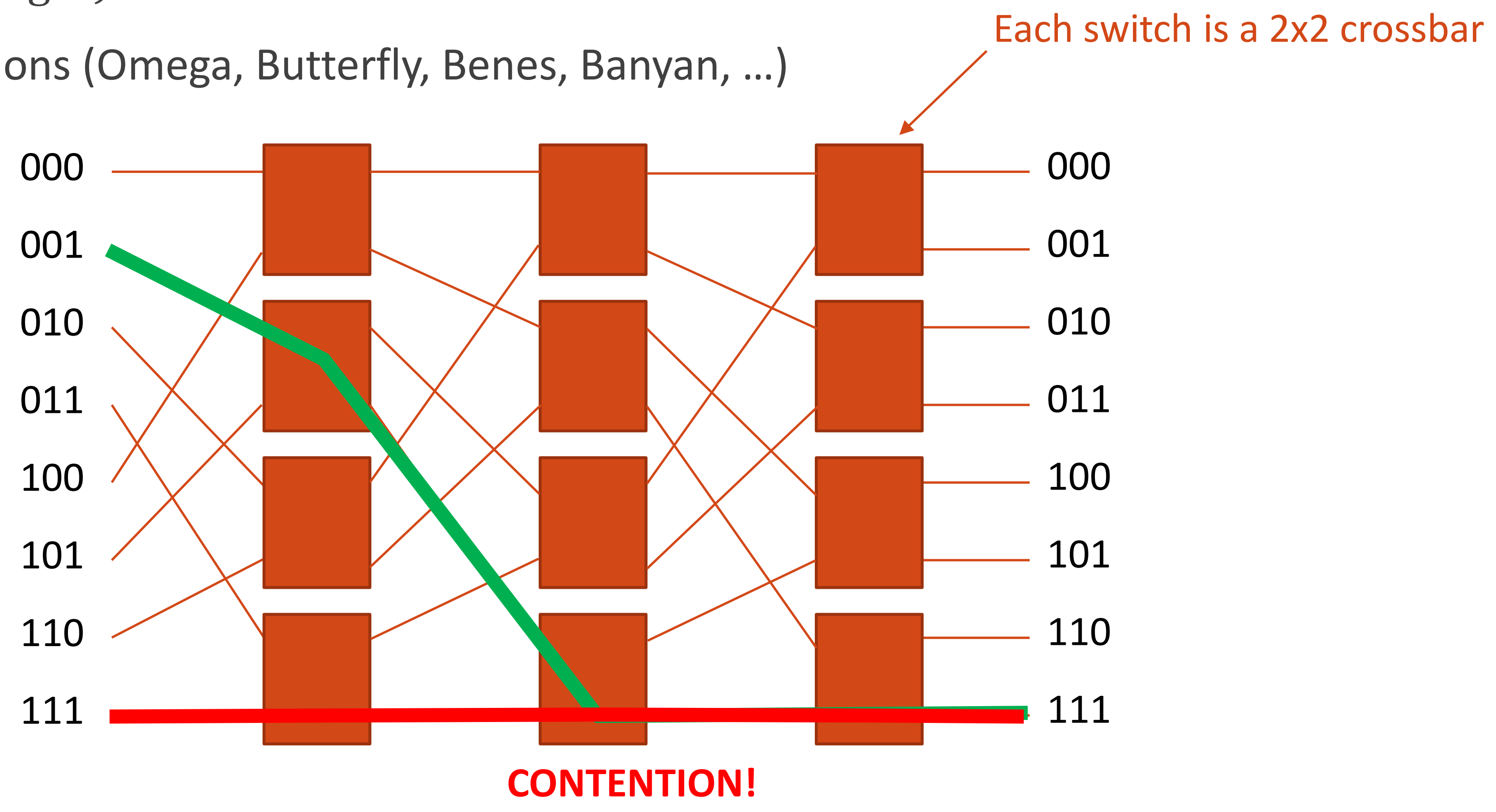


Multistage networks

Idea: $\log N$ switches between nodes

+ Cost $O(N \log N)$

Many variations (Omega, Butterfly, Benes, Banyan, …)



Handling contention

Two packets try to use same link at the same time

What do you do?

- Buffer one
- Drop one
- Misroute one (deflection)

Tradeoffs?

- Buffering increases cost (latency, area, energy)
- Dropping complicates protocols
- Deflection loses ordering

Ring

Unidirectional or bidirectional

+ Cheap $O(N)$ switches

+ Simple switches ➔ Low hop latency

- High latency $O(N)$

- Not scalable; **bisection bandwidth** is constant

Used in many commercial systems today; until recently Intel used “ring of rings” topology

2D Mesh

+ $O(N)$ cost

+ $O(\sqrt{N})$ average latency

+ Natural physical layout

+ Path diversity: Many routes between most sources & destinations
  - Potentially lower contention

+ Decent bisection bandwidth

- More complex routers than ring ➔ Higher hop latency

Used in Tilera 100-core chip & many research papers/prototypes

Trees

Planar, hierarchical topology

+ $O(\log N)$ latency

+ $O(N)$ cost

+ Easy to layout

- Root is bottleneck; constant bisection bandwidth

Trees common for local communication; e.g., banks of single cache

Bisection bandwidth mitigated by “fat trees”, at add’l cost

- Replicate root node, randomize routing
- Used in Thinking Machines CM-5 (1992)

Flow-control methods

Circuit switching

Packet switching

- Store and forward
- Virtual cut-through
- Wormhole

Circuit switching

Pre-allocate resources across multiple switches

Requires “probe” ahead of message

+ No need for buffering

+ No contention (after circuit established)

+ Handles arbitrary message sizes

- Low link utilization

- Delay to set up circuit

Store and forward

Copy entire packet between switches

+ Simple

- High per-packet latency

- Requires big buffers / small messages

Virtual cut-through

Start forwarding as soon as header is received

+ Dramatic reduction in latency vs store and forward

- Still buffers entire message in worst case: requires large buffers / small messages

Wormhole

Break packets into much smaller “flits”

Pipeline delivery: Each flit follows its predecessor through network

If head is blocked, rest of packet waits in earlier switches

+ No large buffering in network

+ Latency independent of distance for large messages

- Head-of-line blocking

Routing algorithms

Deterministic: Simplest, high contention
- Dimension-order (e.g., XY)
- Deadlock-free

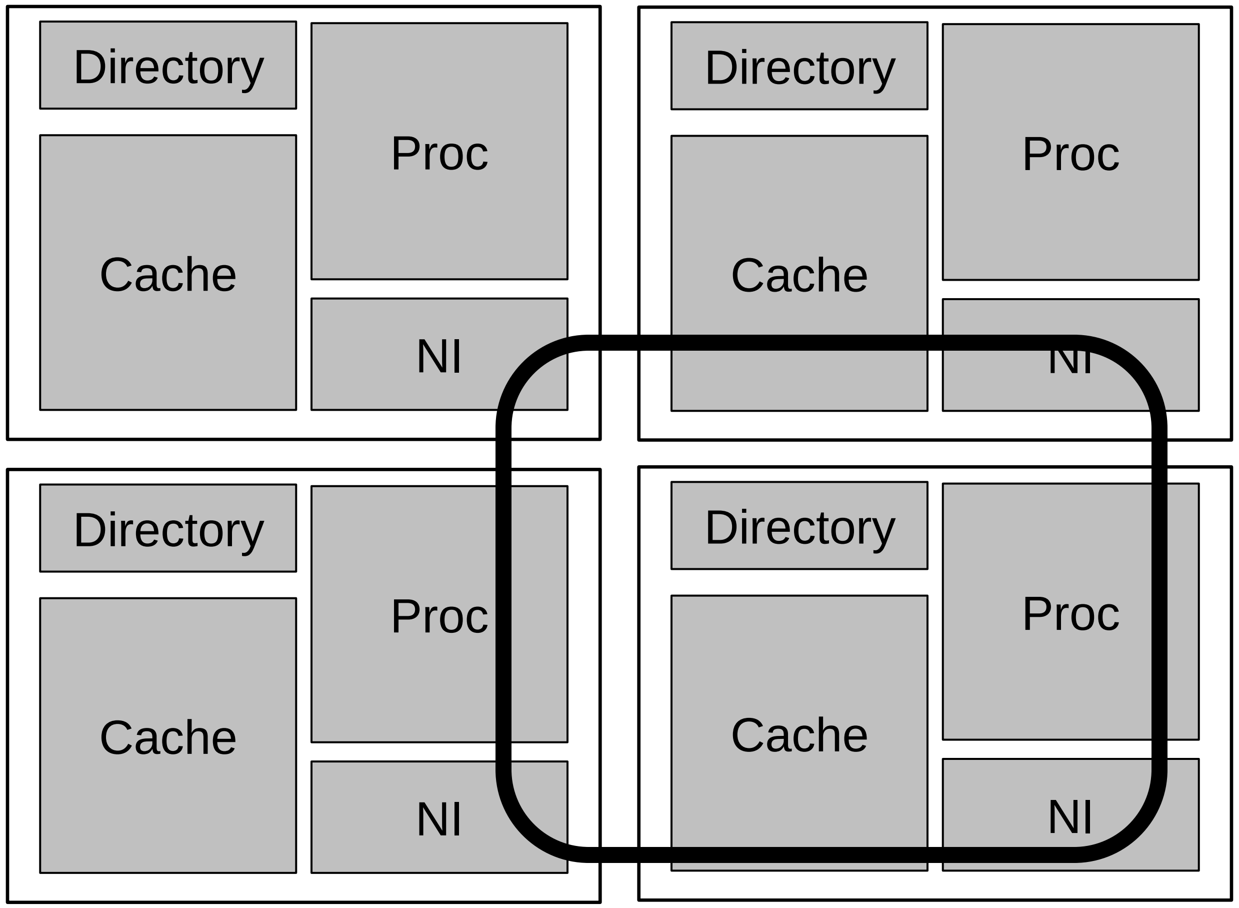
Oblivious: Simple, mitigates contention
- Valiant’s algorithm: Route deterministically via a random node
- Balances network load, adds latency
- Optimization: Use only at high load

Adaptive: Complex, most efficient
- Minimal adaptive: Always route closer to destination on least-contended port
- Fully adaptive: “Misroute” packets to optimize overall network load
  - Must guard against livelock
  - How to coordinate overall network state?

Non-Uniform Cache Access (NUCA)

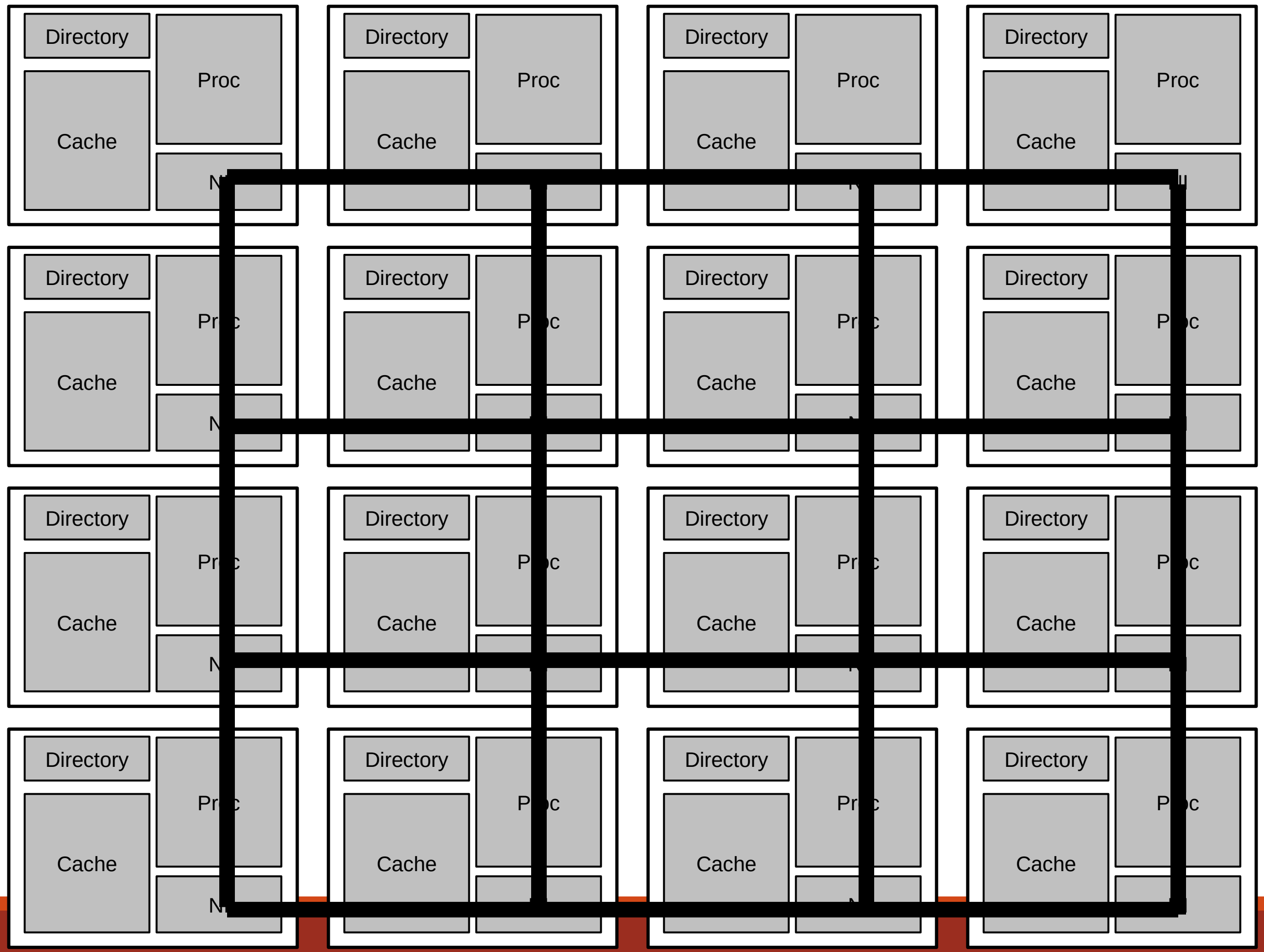
Distributed caches today

4-core system



Distributed caches today

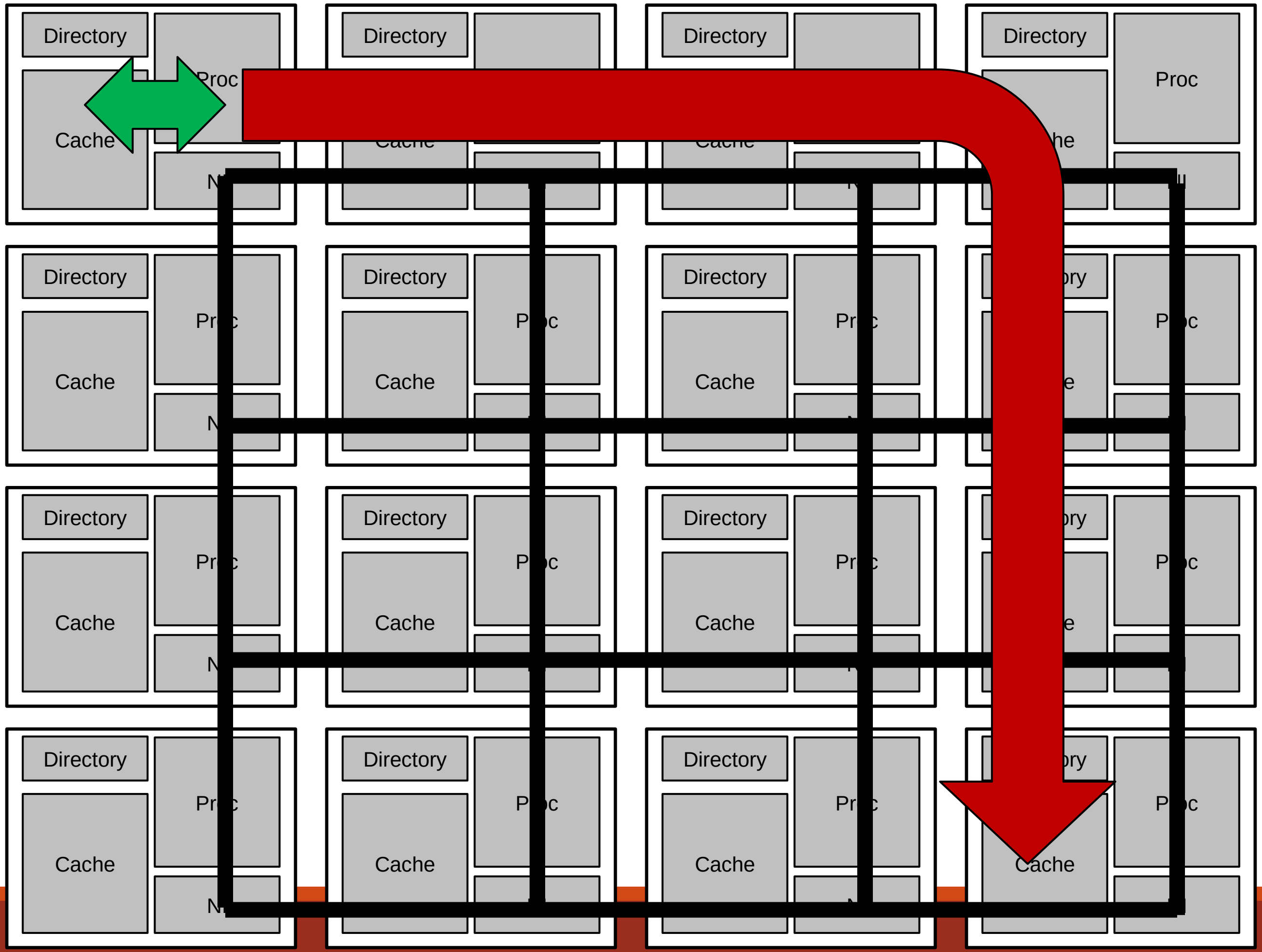
16-core system w/ mesh interconnect



Distributed caches today

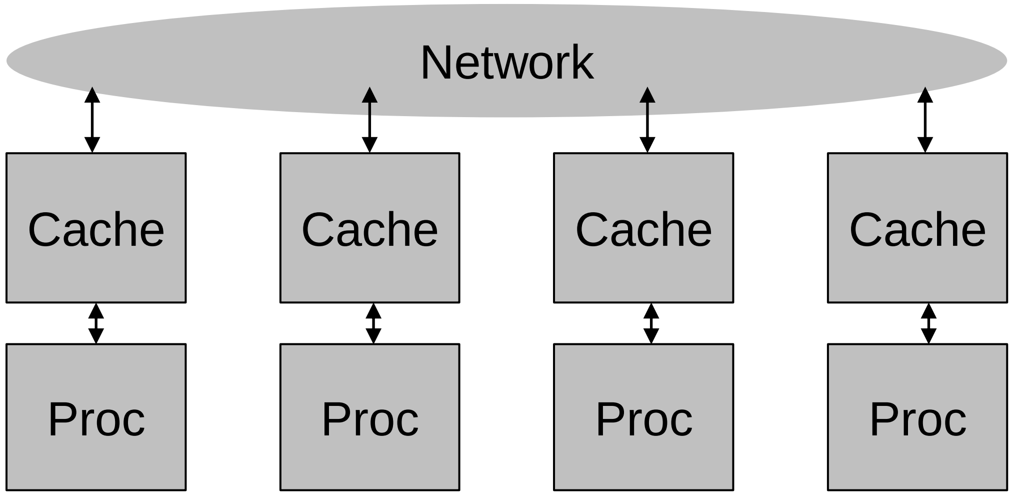
Non-uniform cache access (NUCA): e.g., 7 – 40 cycles for LLC

Problem worsening with larger systems



Recall: Shared vs private caches

Private caches

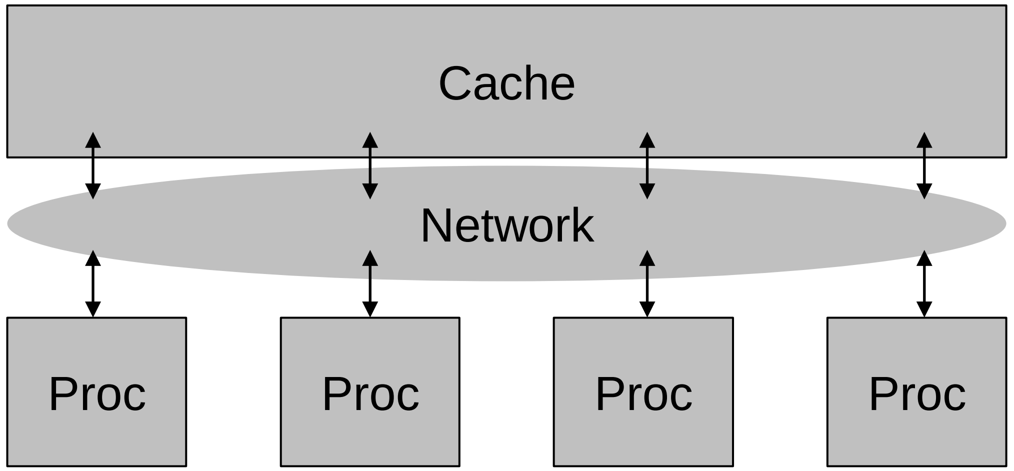


+ Data is nearby (low latency, high bw)

– Limited cache capacity

– Need coherence

Shared cache



+ Lots of capacity

– Data is far away (high latency, low bw)

+ Don't need coherence?

Can we get the best of both worlds?

Victim Replication

[Zhang & Asanovic, ISCA'05]

A representative *private-shared* organization

Starting point: private L1s + shared L2

Idea: treat local L2 bank as a "victim cache"
- L1 misses check local L2 bank, then global L2 bank
- Data is brought into local L2 bank
- Thus all L2 banks are *simultaneously private and shared*

Tradeoffs:

+ Lower latency than shared
+ Higher capacity than private
– Higher latency than pure private (why?)
– Lower capacity than shared (why?)
– Add'l coherence overheads (why?)

Adaptive selective replication

[Beckmann+, MICRO'06]
(not me)

Be smart about which lines are replicated in L2s

Focus on shared, read-only blocks
- 40% of L2 requests and only 10% of L2 space

Most frequently requested L2 blocks == most frequently evicted L1 blocks

Monitor L1 evictions to choose what to replicate in L2

Use probabilistic policy to get the hot data eventually replicated
- Hardware dynamically adapts the replication probability based on observed access patterns, weighing benefit (local hits) vs cost (off-chip misses) of replication

[Qureshi, HPCA'09]

Dynamic spill-receive

Starting point: Private organization w/ snoopy bus

Caches can *spill* data to other caches [Chang, ISCA'06]

- Alternative method to share capacity

Problem: All caches shouldn't be spilling at once!

- Caches should *either* spill or receive, not both!

DSR uses set dueling (again) within each cache bank to choose whether it should spill/receive

[Hardavellas+, ISCA'09]

Reactive NUCA

Goal: Get a good data placement w/out adding lookups

Starting point: Shared organization

Split accesses into three categories and customize **data placement** (i.e,. home node) for each:

- Instructions: local 4-way replication
- Read-write data: stripe data across banks
- Private: local bank always

Classify data on a page-level and use TLB to determine placement

[Beckmann & Sanchez, PACT’13]
[Tsai+, ISCA’17]

Software-defined caches

Prior NUCAs do not correctly tradeoff capacity + latency!

- Capacity often more important than latency
- (Remember AMAT equation)

Idea: Dynamically size + place data

- Use ~UCP to choose how much space
- Place data near the threads that use it

Hardware: Control placement via TLB, like R-NUCA

Software: Complex optimization problem to find best placement

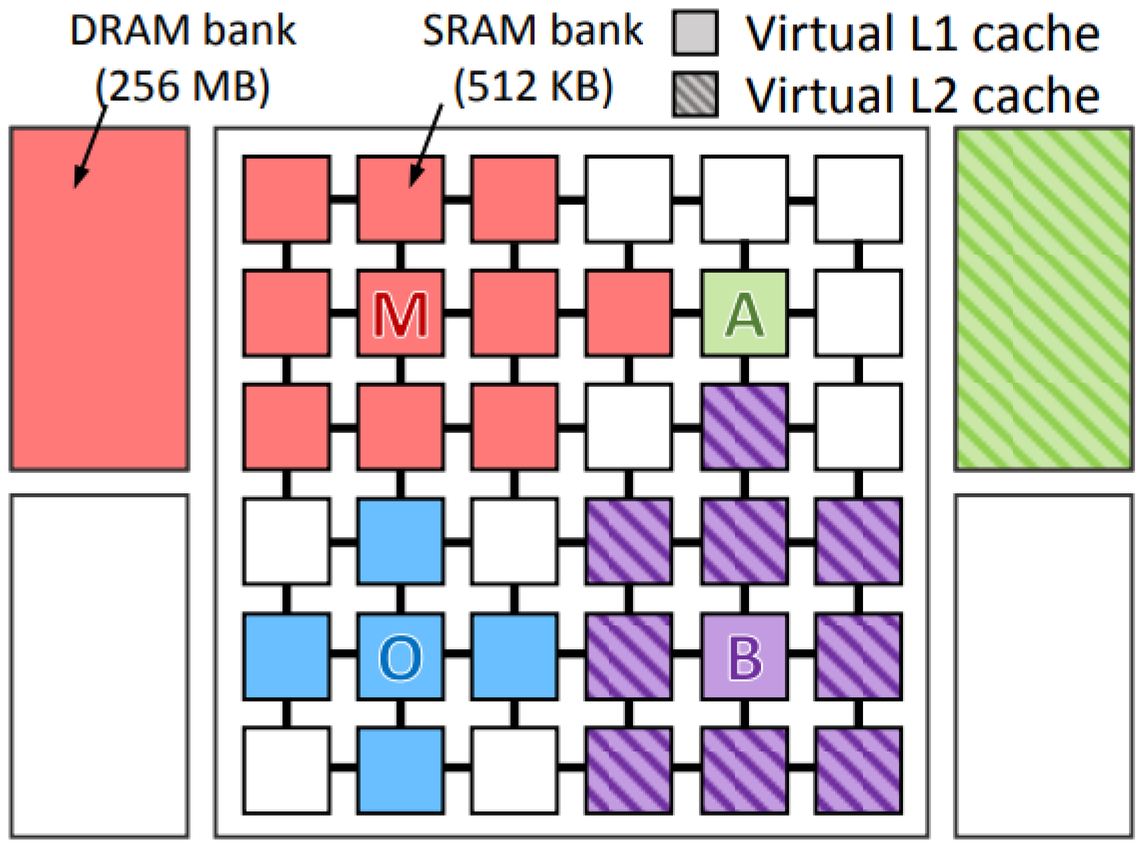


Figure 6: A 36-tile Jenga system running four applications. Jenga gives each a custom *virtual cache hierarchy*.

NoC & NUCA Summary

Communication between cores is the big scalability problem for multicore

NoC architecture determines cost & scaling trends

- No free lunch; hard tradeoffs at every level

NUCA is new challenge for shared multicore caches

- Need to build-in support to “schedule data” onto cache banks just like we schedule threads on cores

Self-check questions

Why did multicores move from buses to rings to meshes?

How does NoC architecture affect the design of cache coherence, and vice versa?

A “concentrated mesh” shares a single NoC router among multiple cores / cache banks. What are the tradeoffs in such a design?

(For theory folks) The PRAM (parallel random access memory) model is often used to analyze the performance of parallel algorithms, assuming that all threads can access a shared memory. How would NUCA effects change this model?