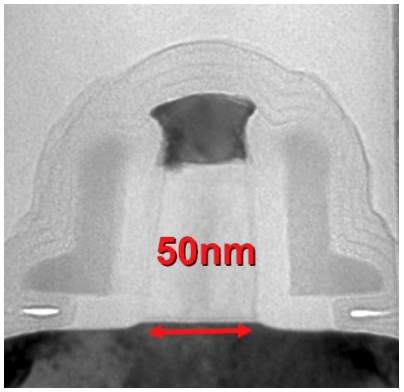
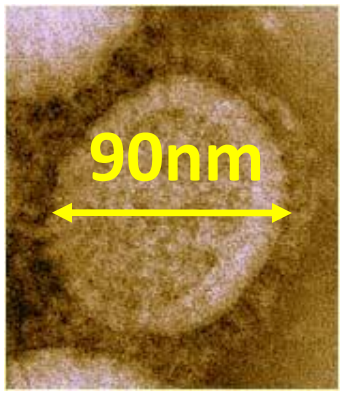
Many-Core Computing Era and New Challenges

Nikos Hardavellas, EECS

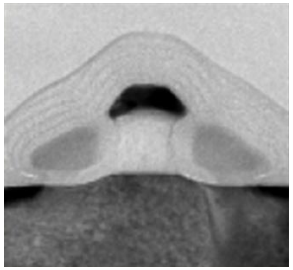
Moore's Law Is Alive And Well



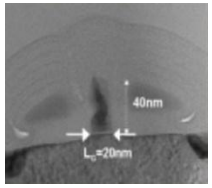
90nm transistor (Intel, 2005)



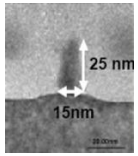
Swine Flu A/H1N1 (CDC)



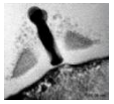
65nm
2007



45nm
2010

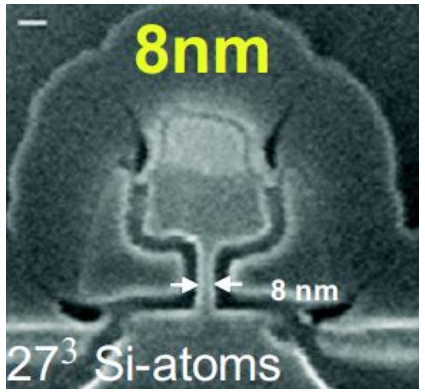


32nm
2013



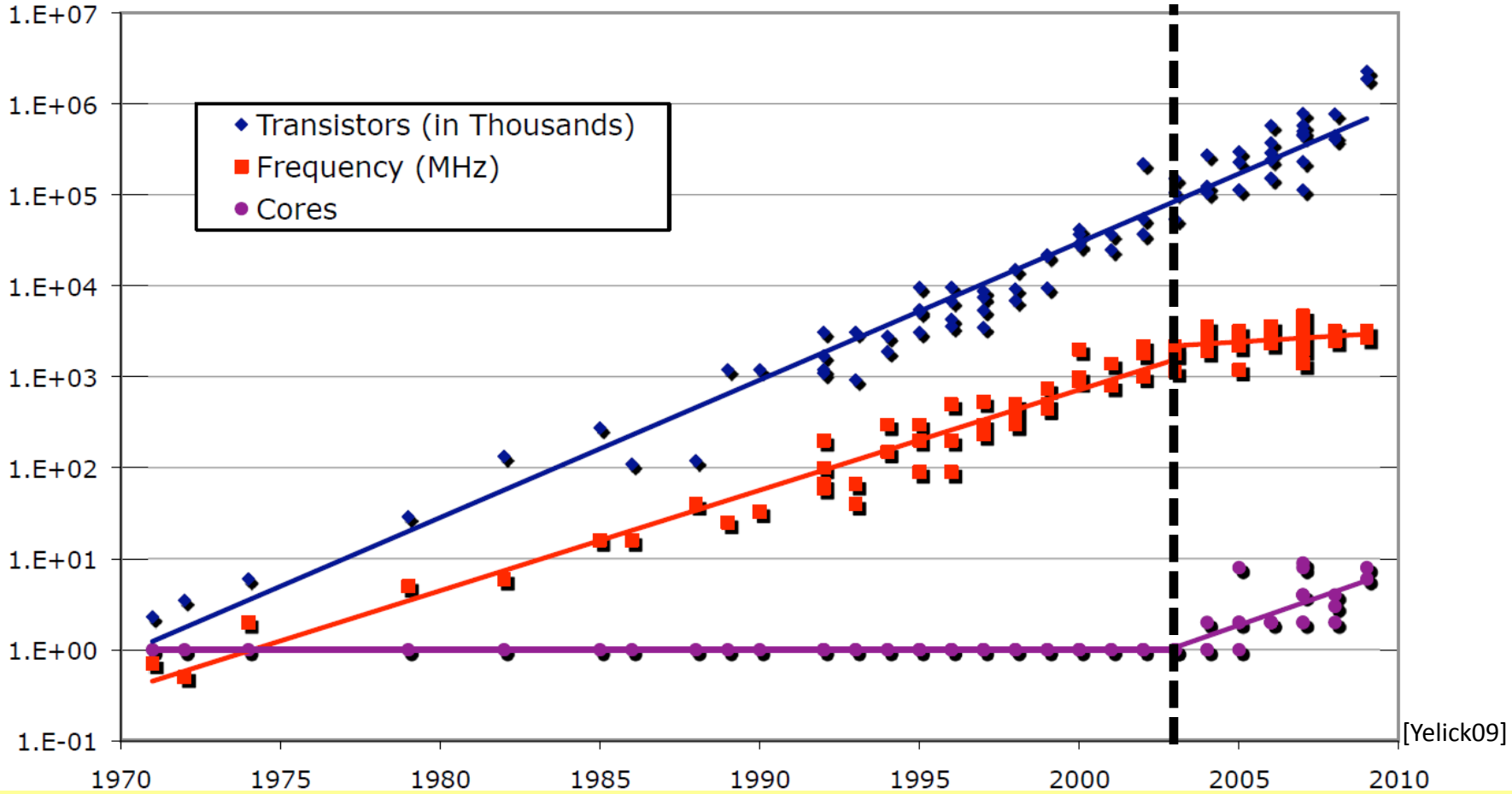
22nm
2016

16nm
2019



- Device scaling continues for at least another 10 years

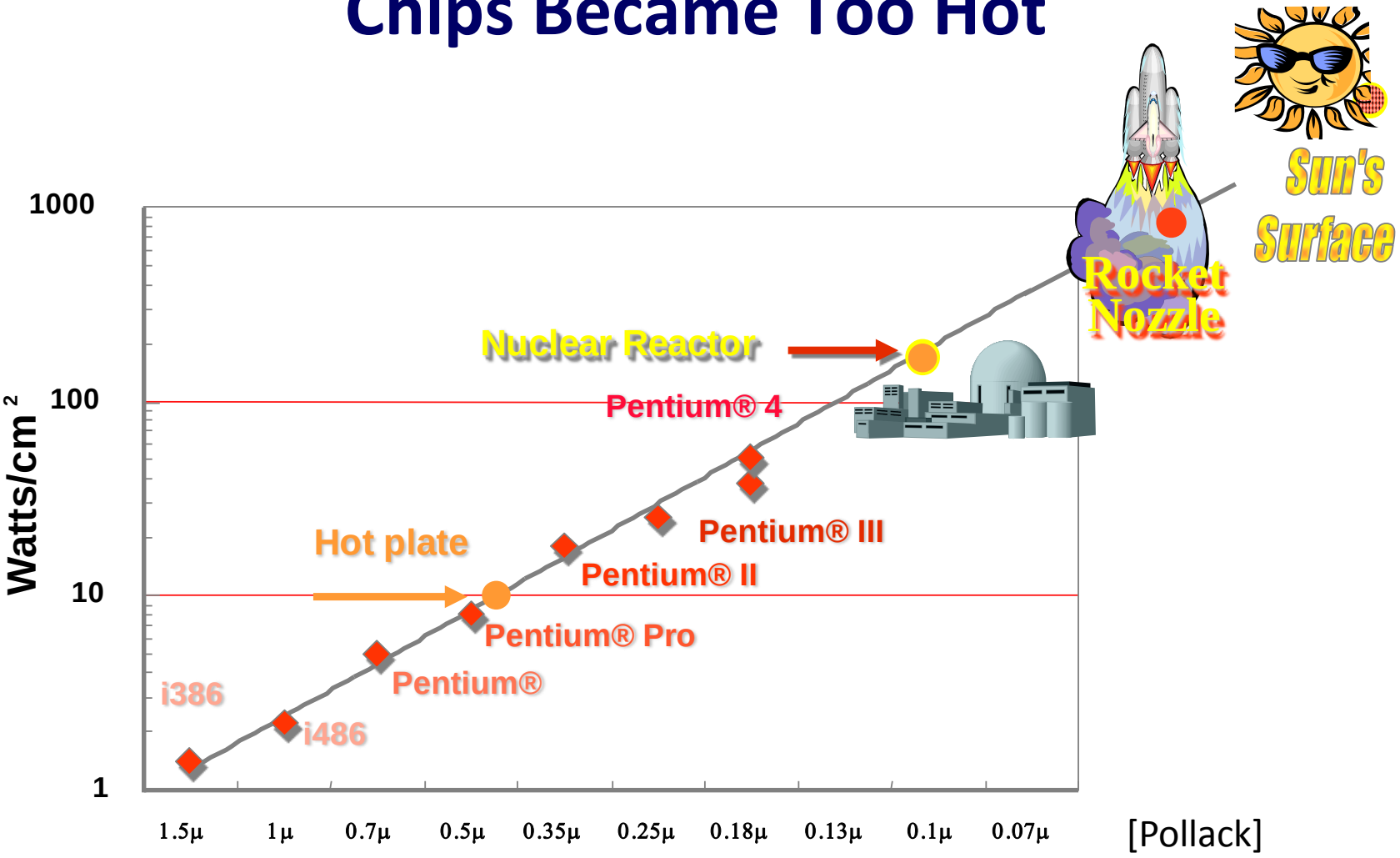
Good Days Ended Nov. 2002



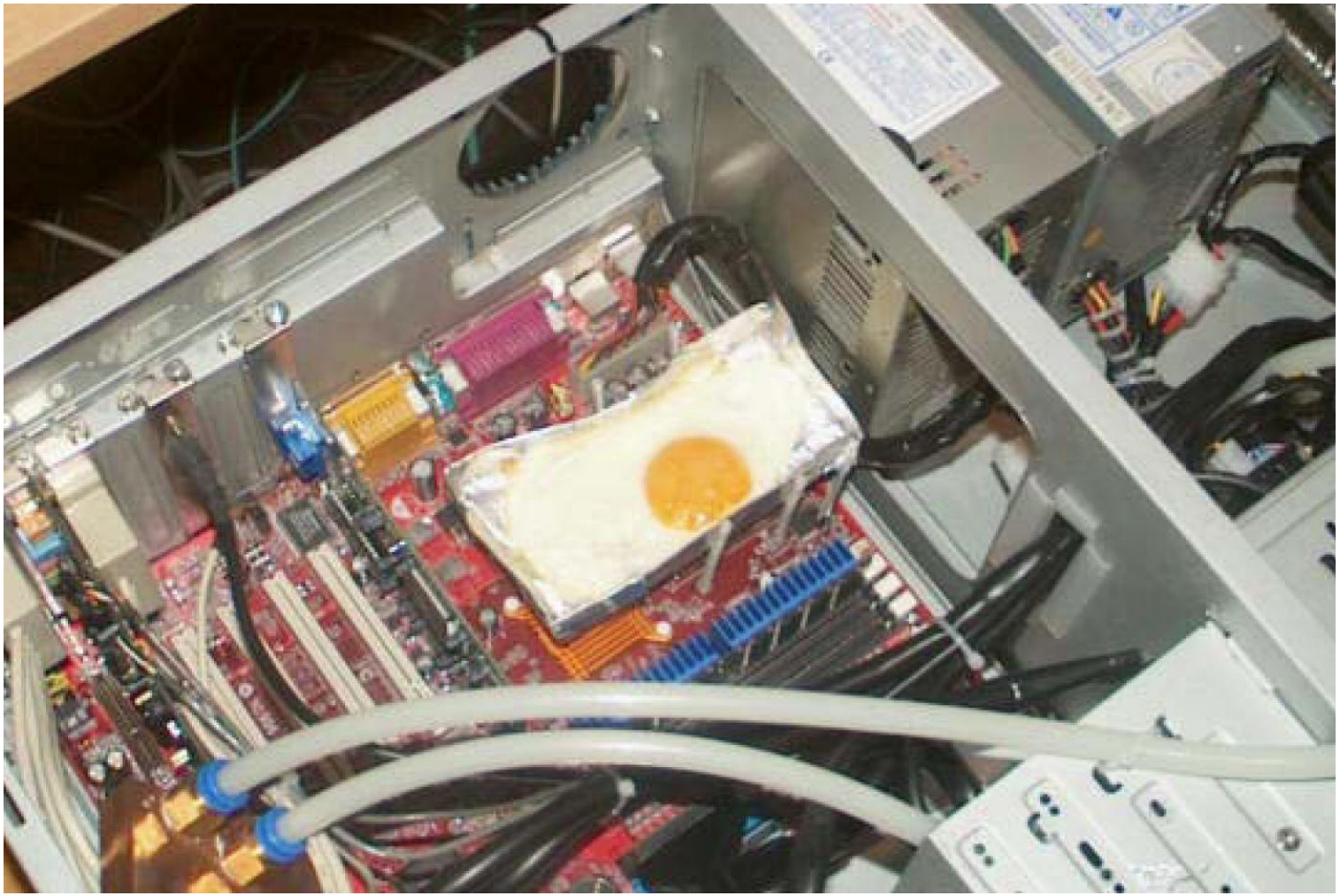
[Yelick09]

- "Traditional" Moore's Law: 2x transistors with every generation
  - "New" Moore's Law: 2x cores with every generation

What Happened in Nov. 2002?
Chips Became Too Hot

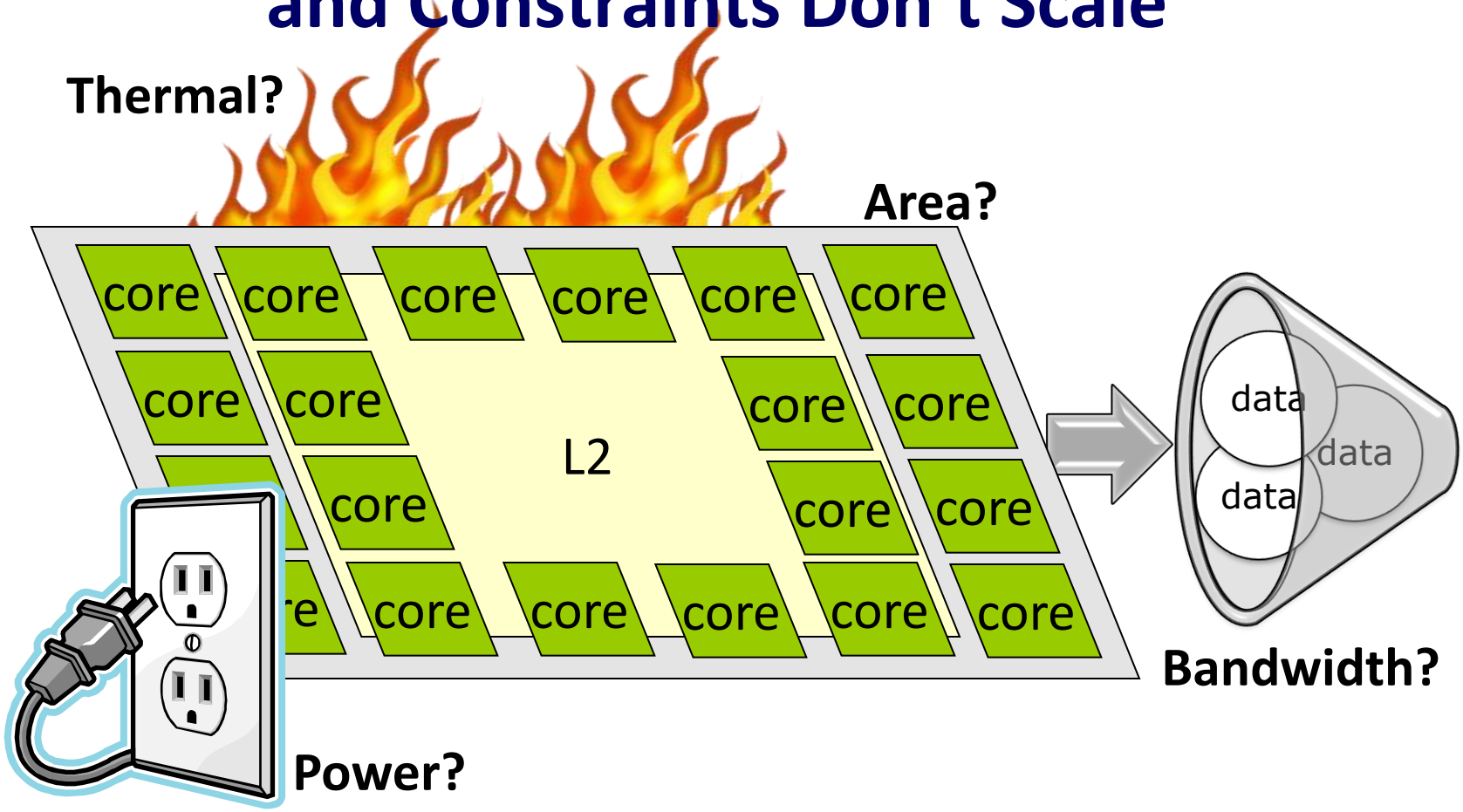


The New Cooking Sensation!



[Huang]

But, Chips Are Physically Constrained, and Constraints Don't Scale

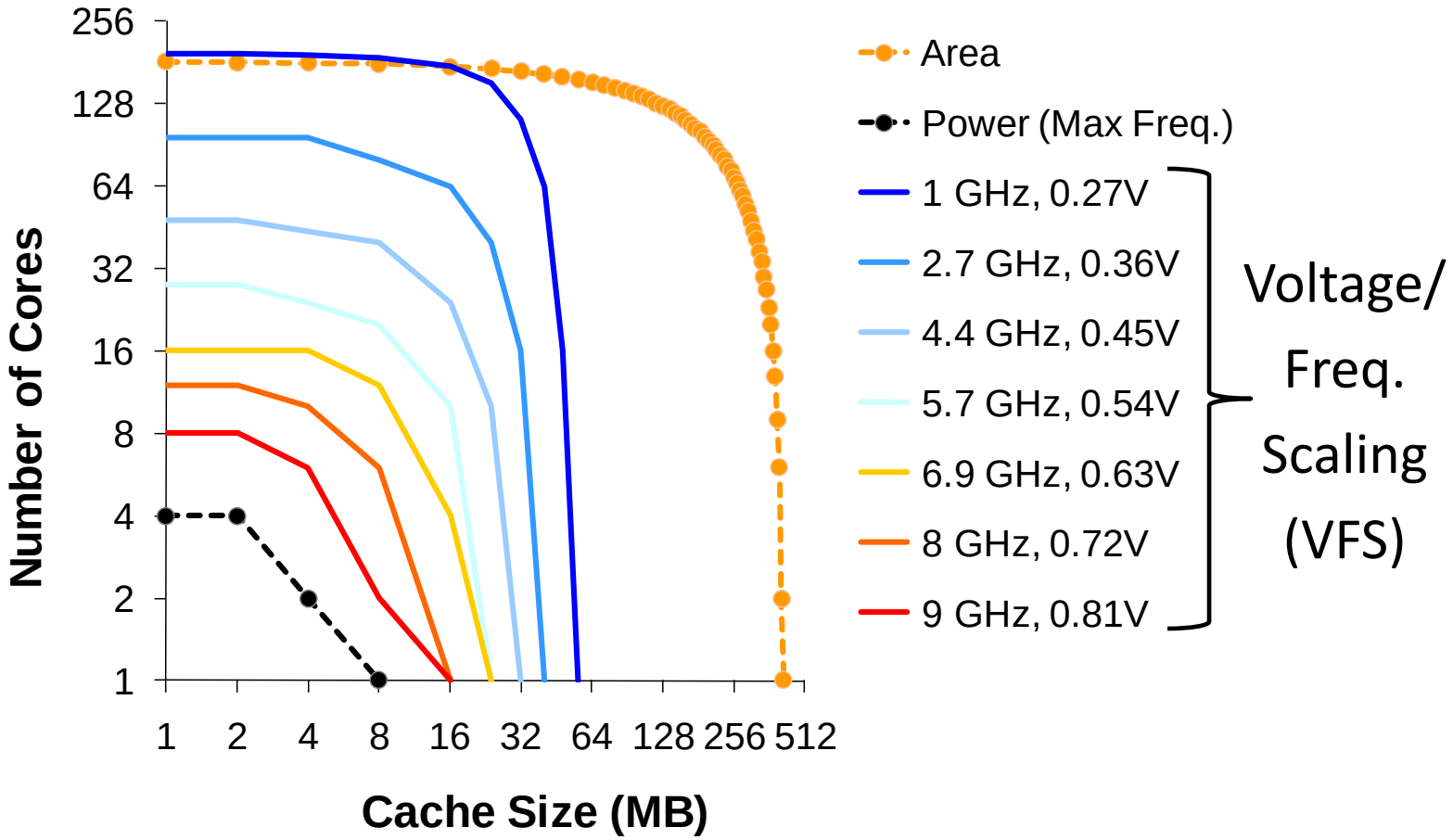


- How to balance constraints and achieve peak performance?

A Look at Technology Projections & Implications

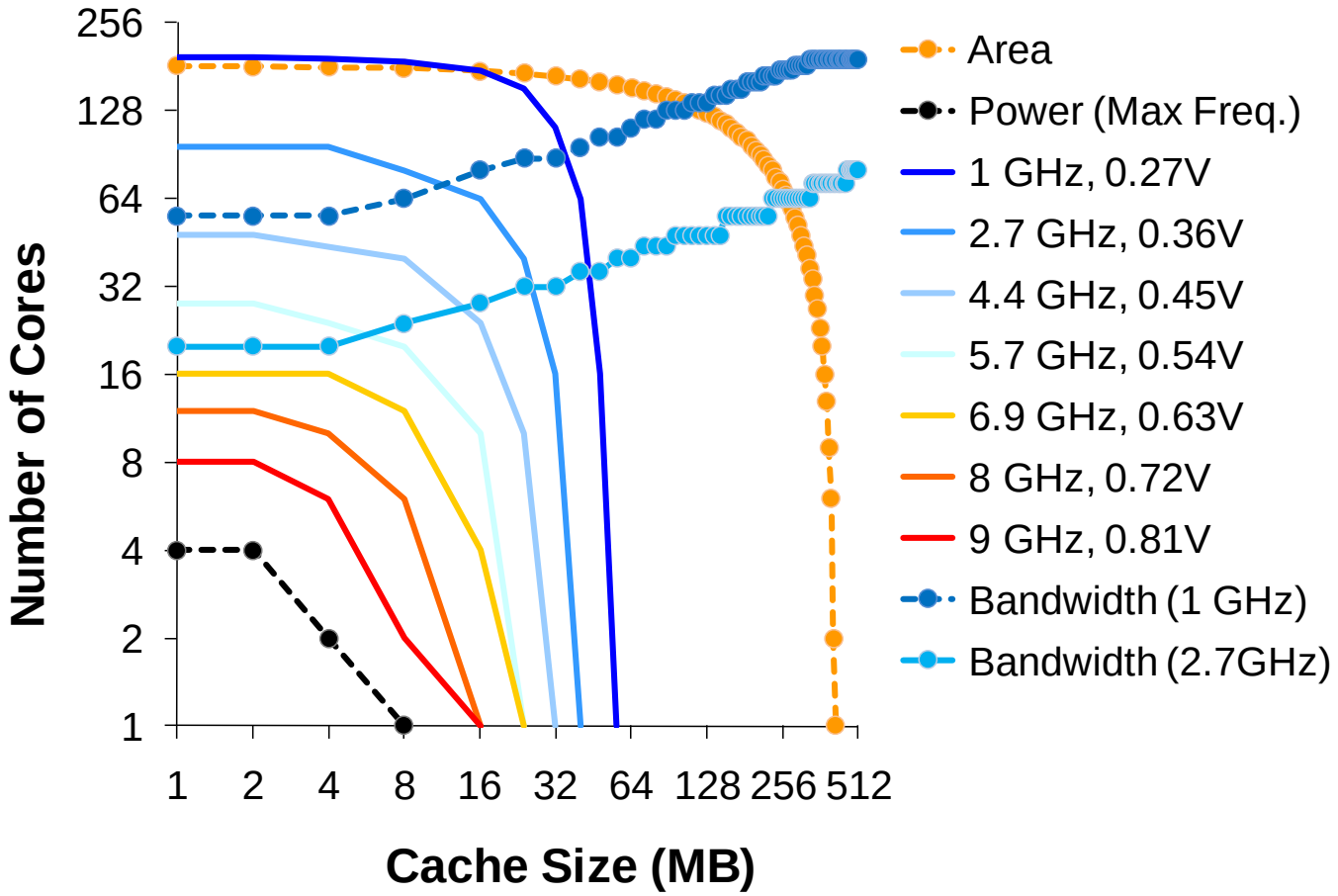
- Use ITRS projections + first-order models/analysis
  - Given a technology node, find highest-performance design
  - Respect physical constraints
  - Account for SW trends
    - Amdahl's Law
    - Exponentially-increasing datasets

How Many Cores/Cache Can We Power?



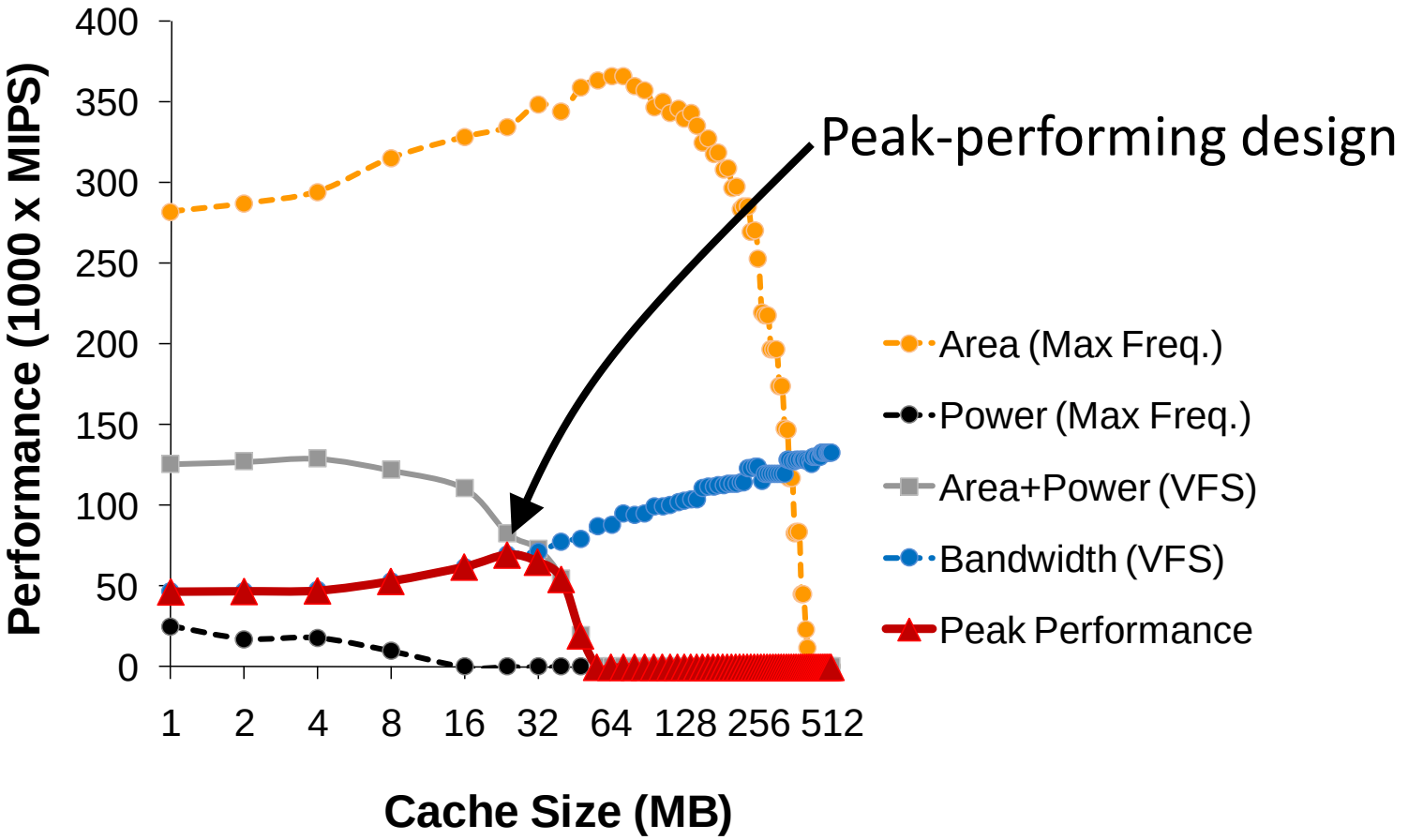
- The power wall: a power/performance trade-off

Pin Bandwidth: Fewer Cores, More Cache



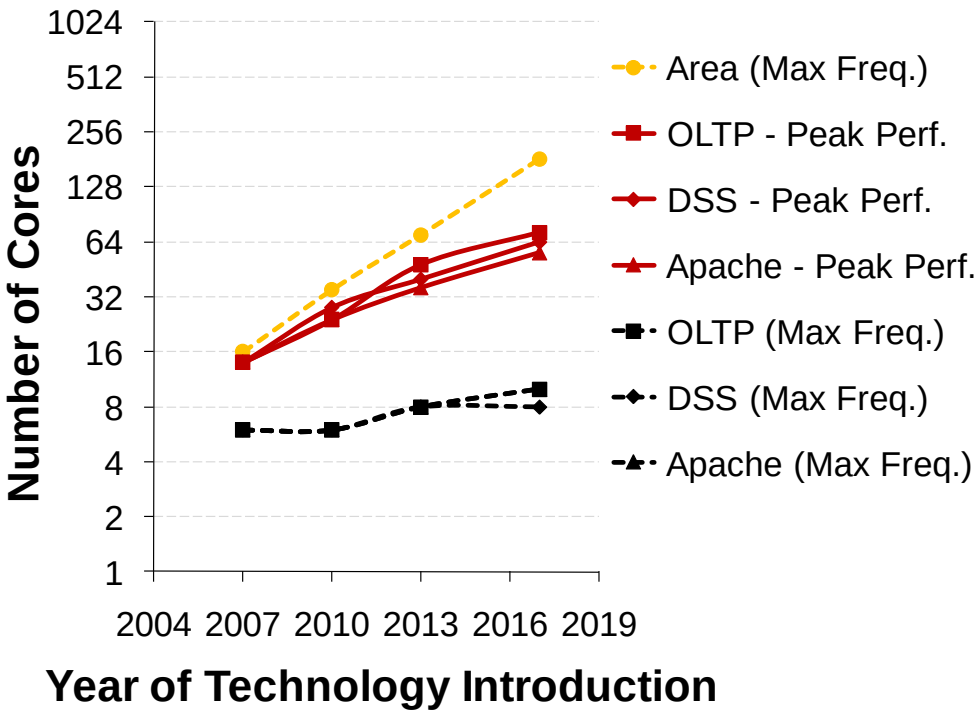
- Where would the best performance be?

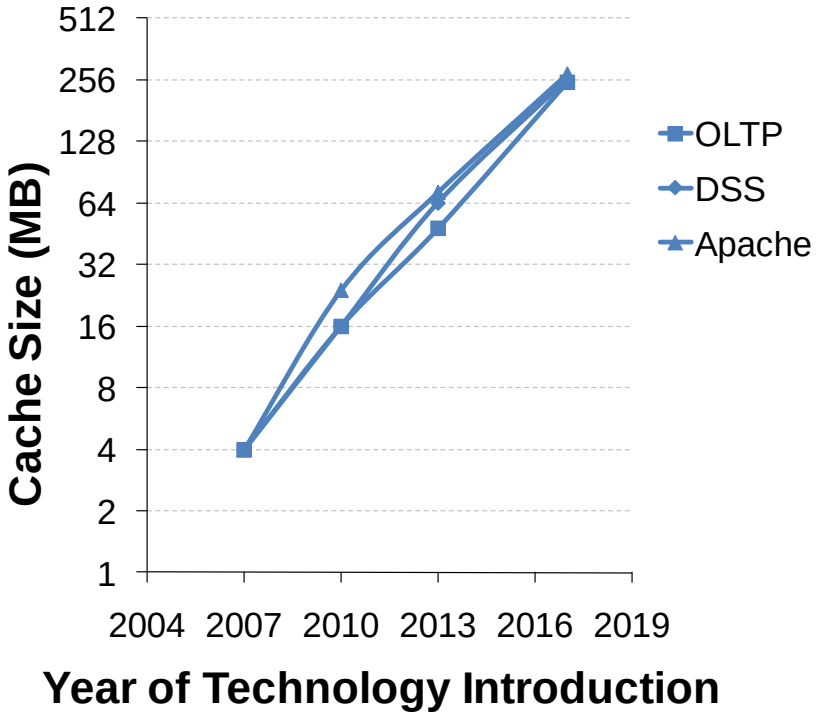
Peak-Performing Designs



- First, bandwidth constrained, then power constrained

Transistor Scaling: Many Cores, Huge Caches





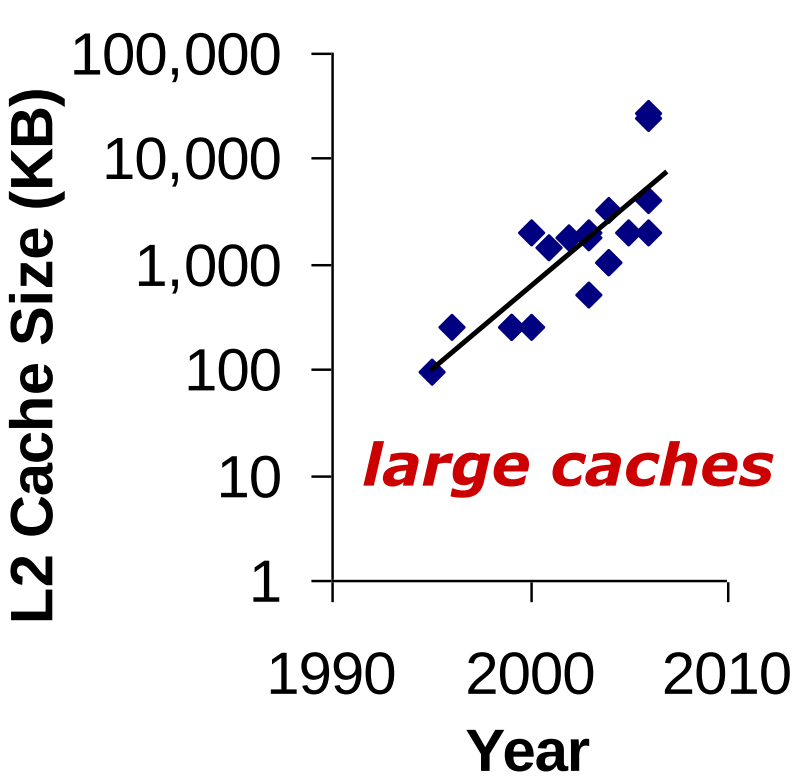
➧ Need cache architectures for >>MB

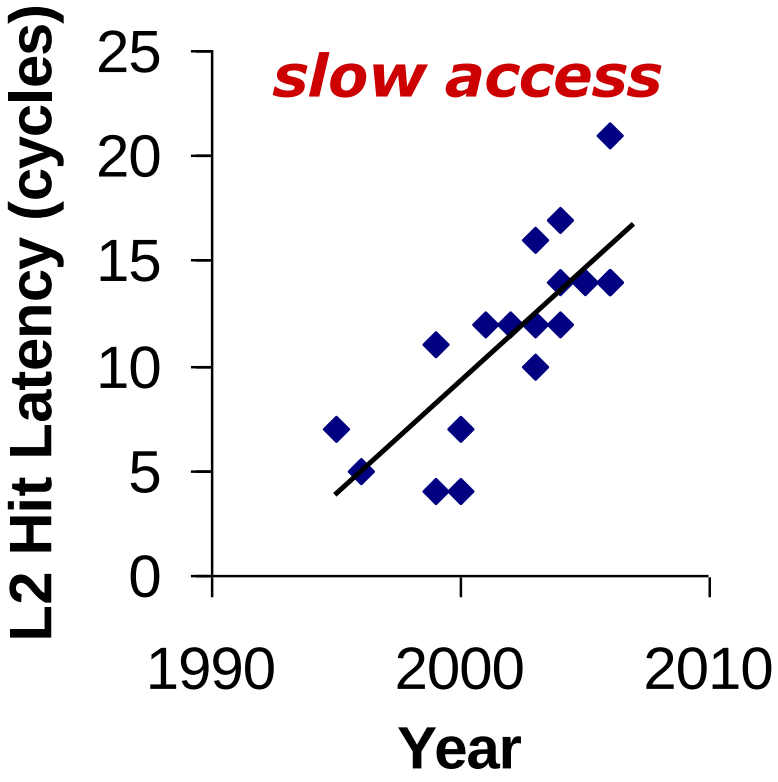
Why Are Caches Growing So Large?

- Increasing number of cores: cache grows commensurately
  - Fewer but faster cores have the same effect
- Increasing datasets: faster than Moore's Law!
- Power/thermal efficiency: caches are "cool", cores are "hot"
  - So, its easier to fit more cache in a power budget
- Limited bandwidth: large cache == more data on chip
  - Off-chip pins are used less frequently

➡ So, caches are getting huge. Great! What's the catch?

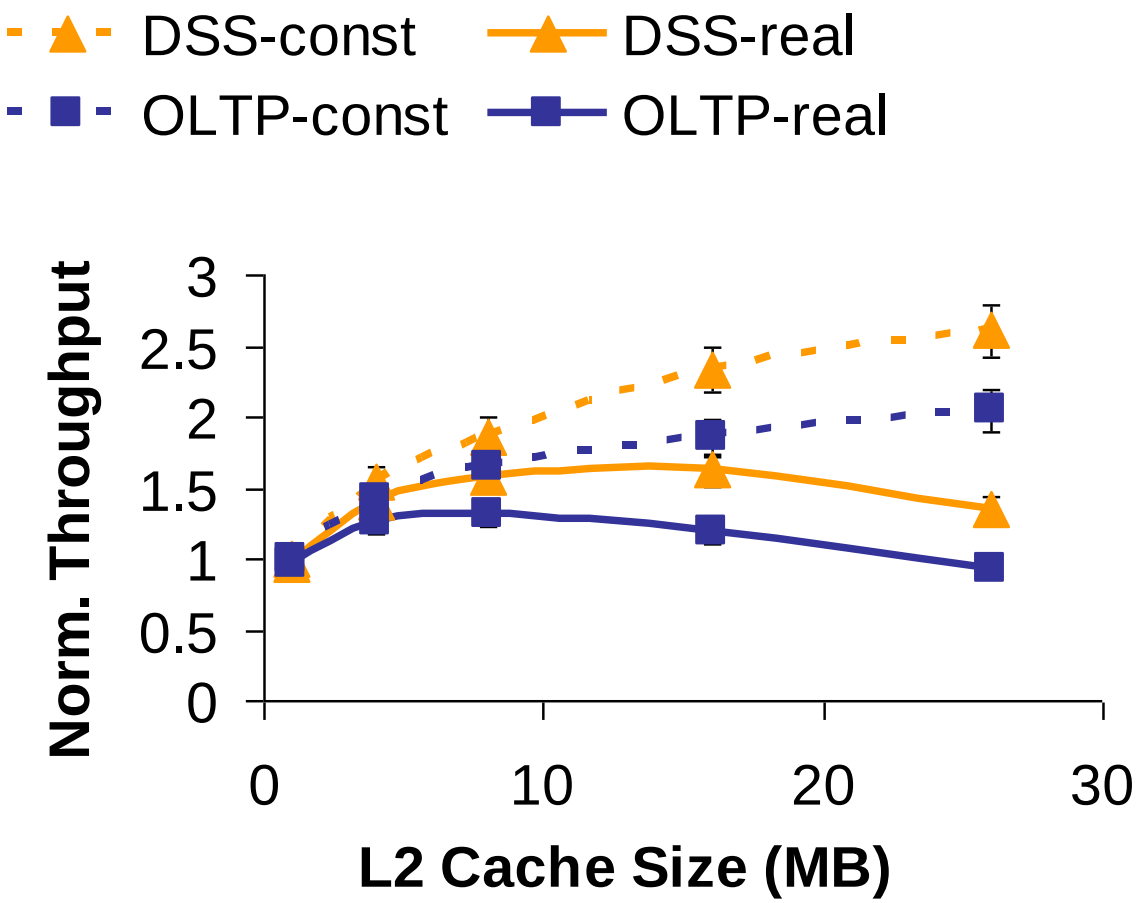
Larger Caches Are Slower Caches





➧ Does this affect the end performance?

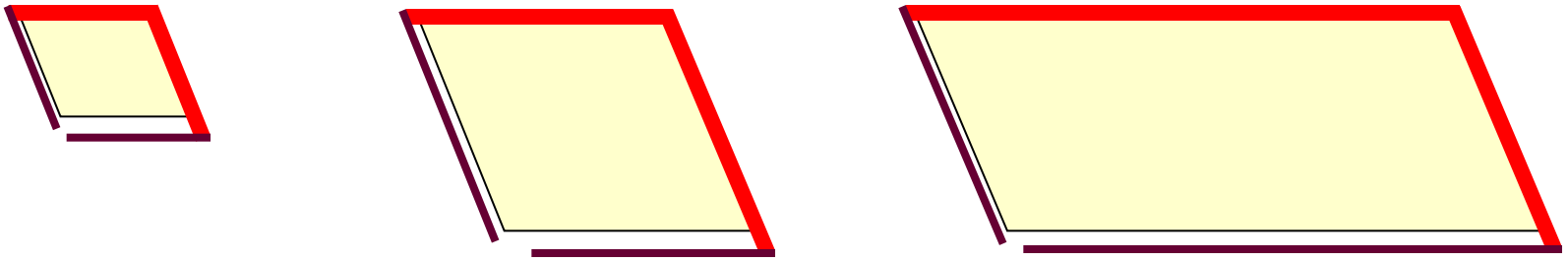
Impact of Slower Caches on Performance



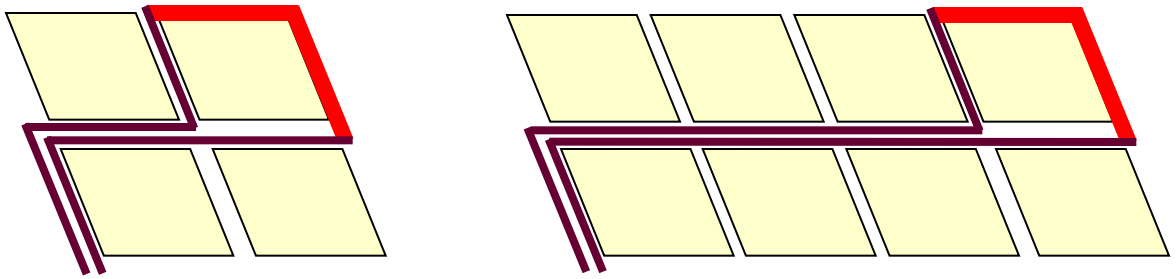
We lose half the potential throughput!

Cache Design Trends

As caches become bigger, they get slower:

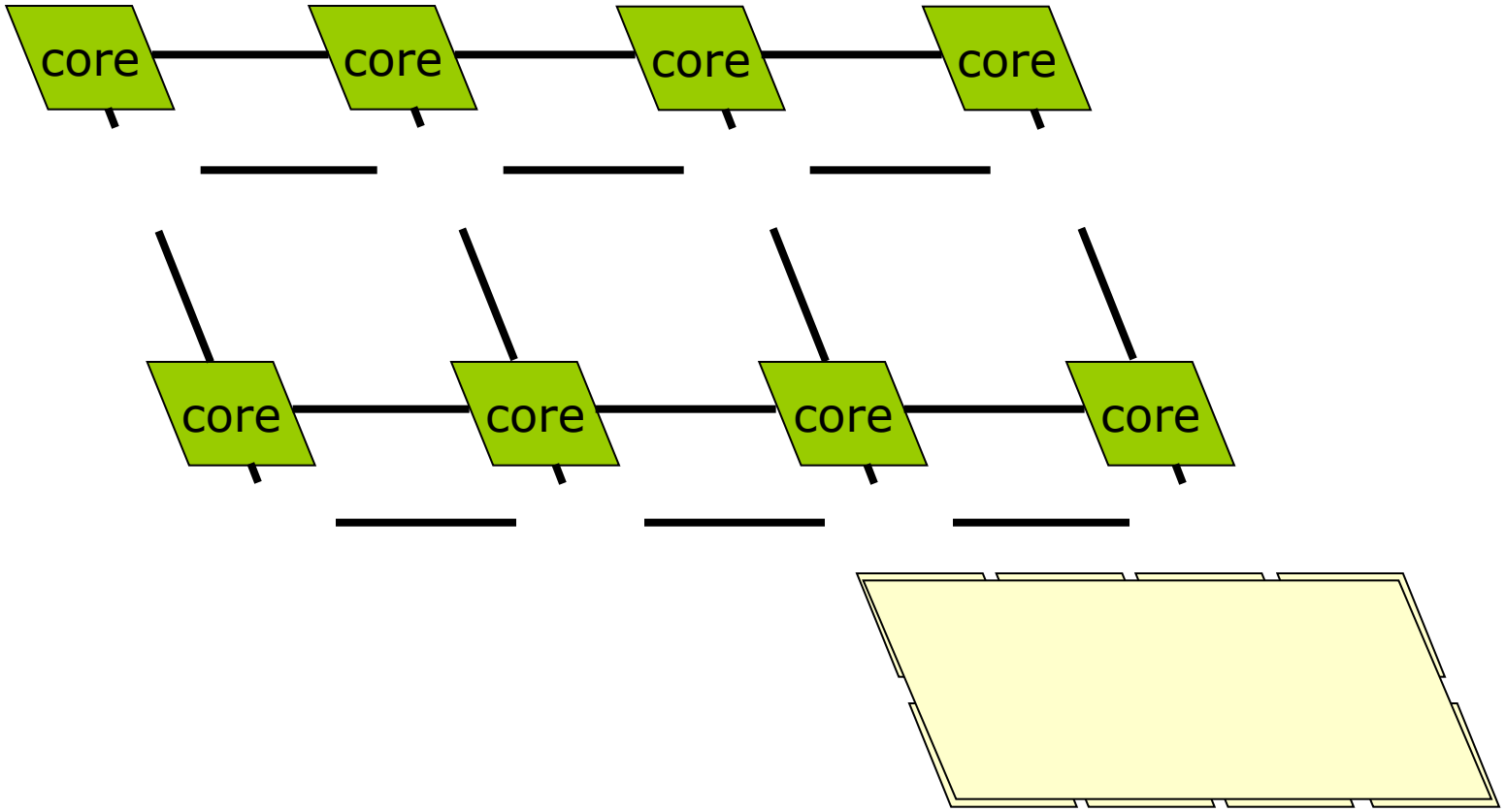


Divide the cache into smaller “slices”:



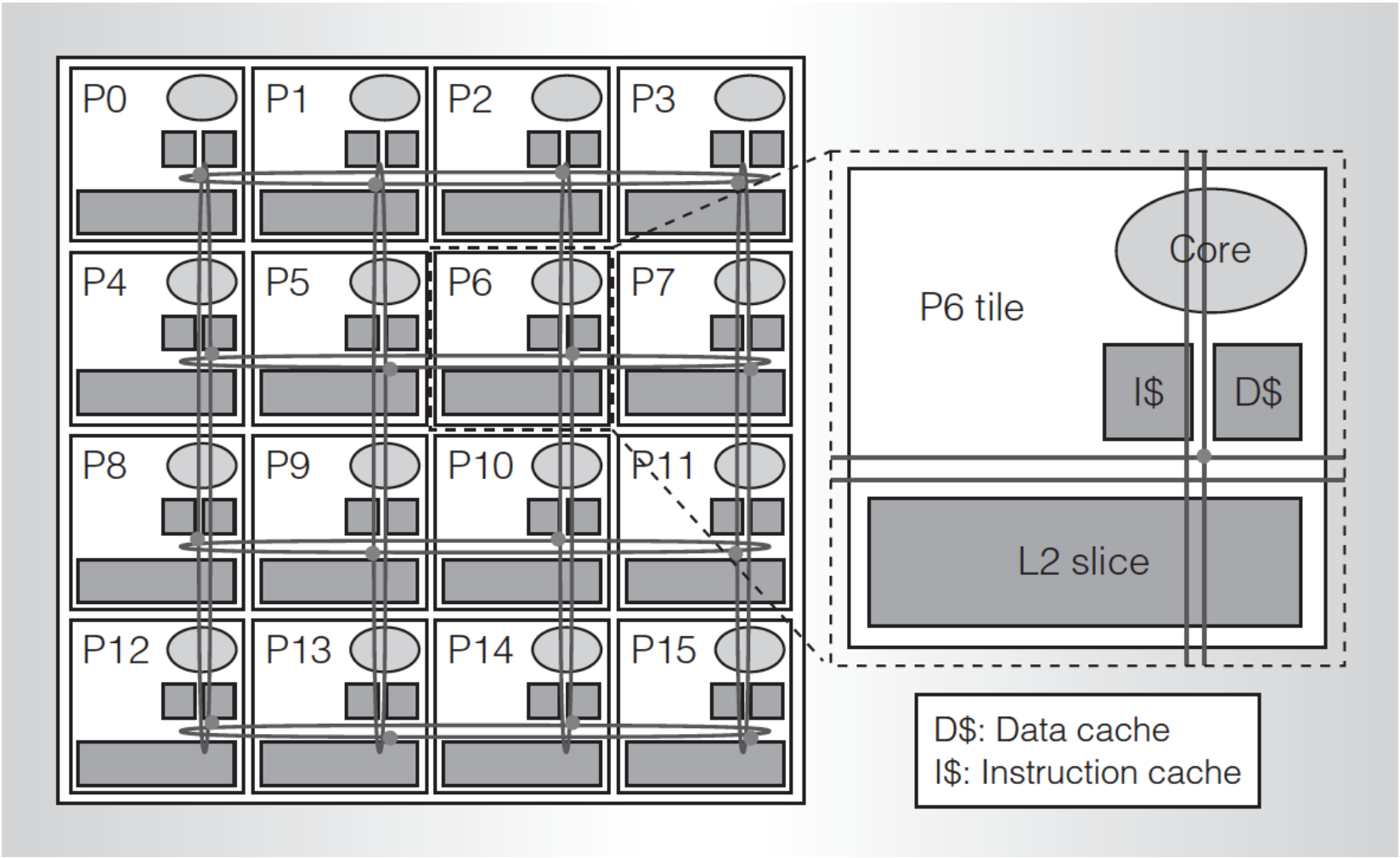
- Balance cache slice access with network latency

Modern Caches: Distributed



- Split cache into “slices”, distribute across die

Modern Multi-Core Chips



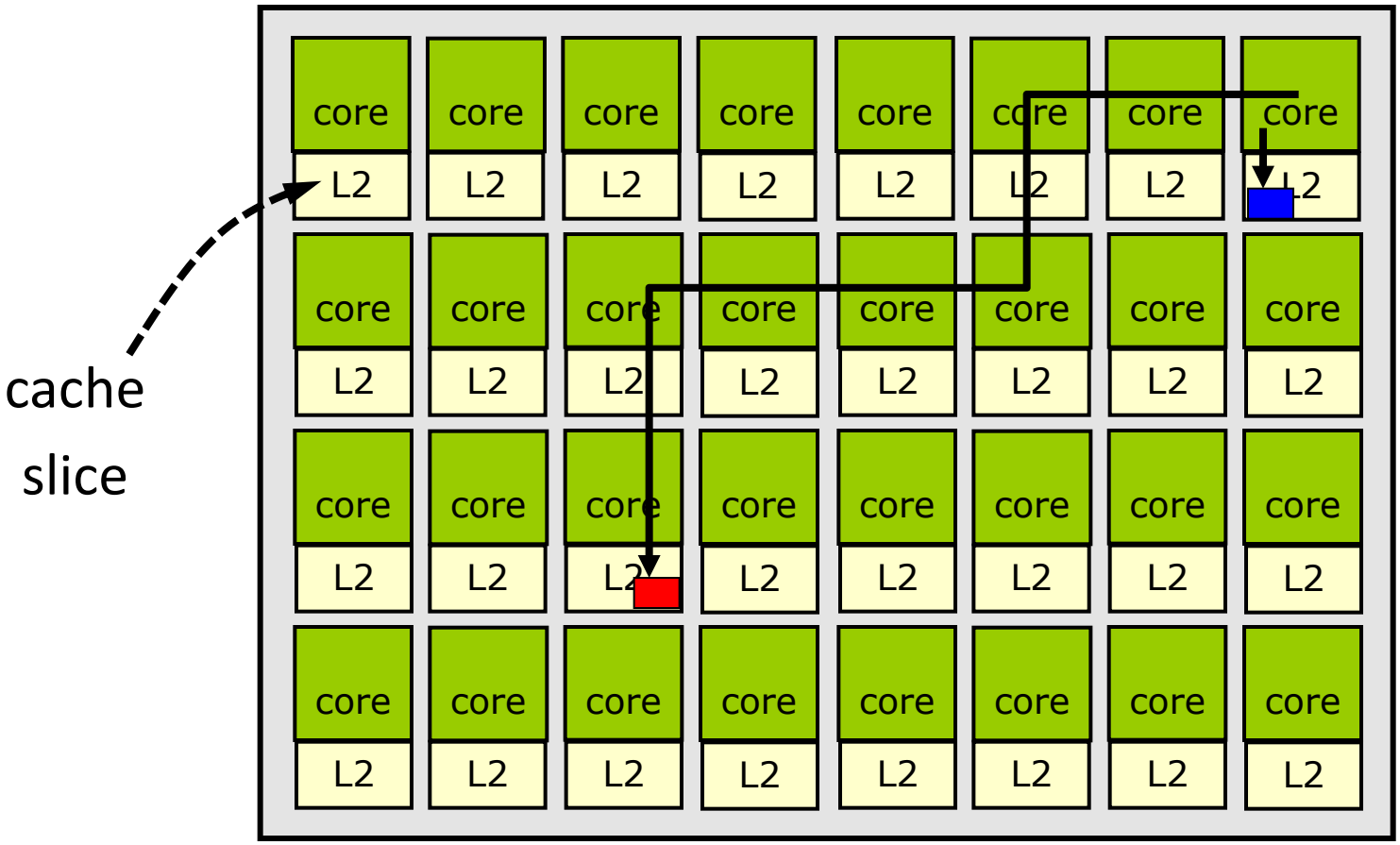
Multi-Core: Distributed System On Chip

- Similar structure
  - Nodes in a cluster or a multiprocessor -> cores on a chip
  - Physically distributed memory -> distributed cache
  - Interconnect -> ditto, but on chip

- Similar challenges, albeit with different constants:
  - Power, thermal, reliability, performance

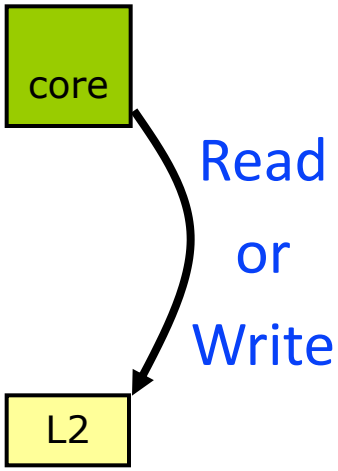
- Also, some new challenges:
  - Off-chip bandwidth: limited pins

Data Placement Determines Performance

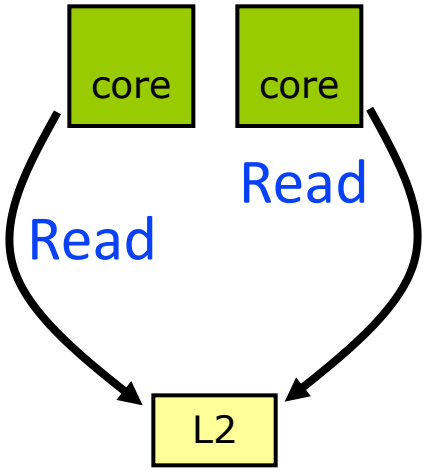


- Goal: place data on chip close to where they are used

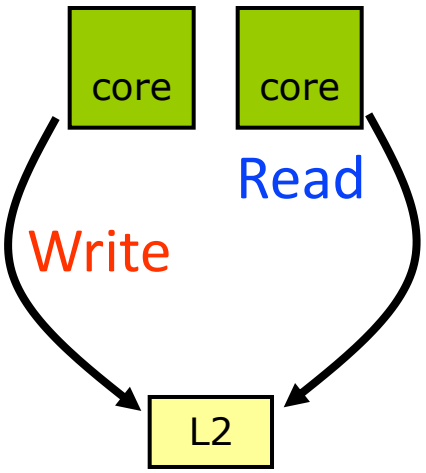
Terminology: Data Types



Private

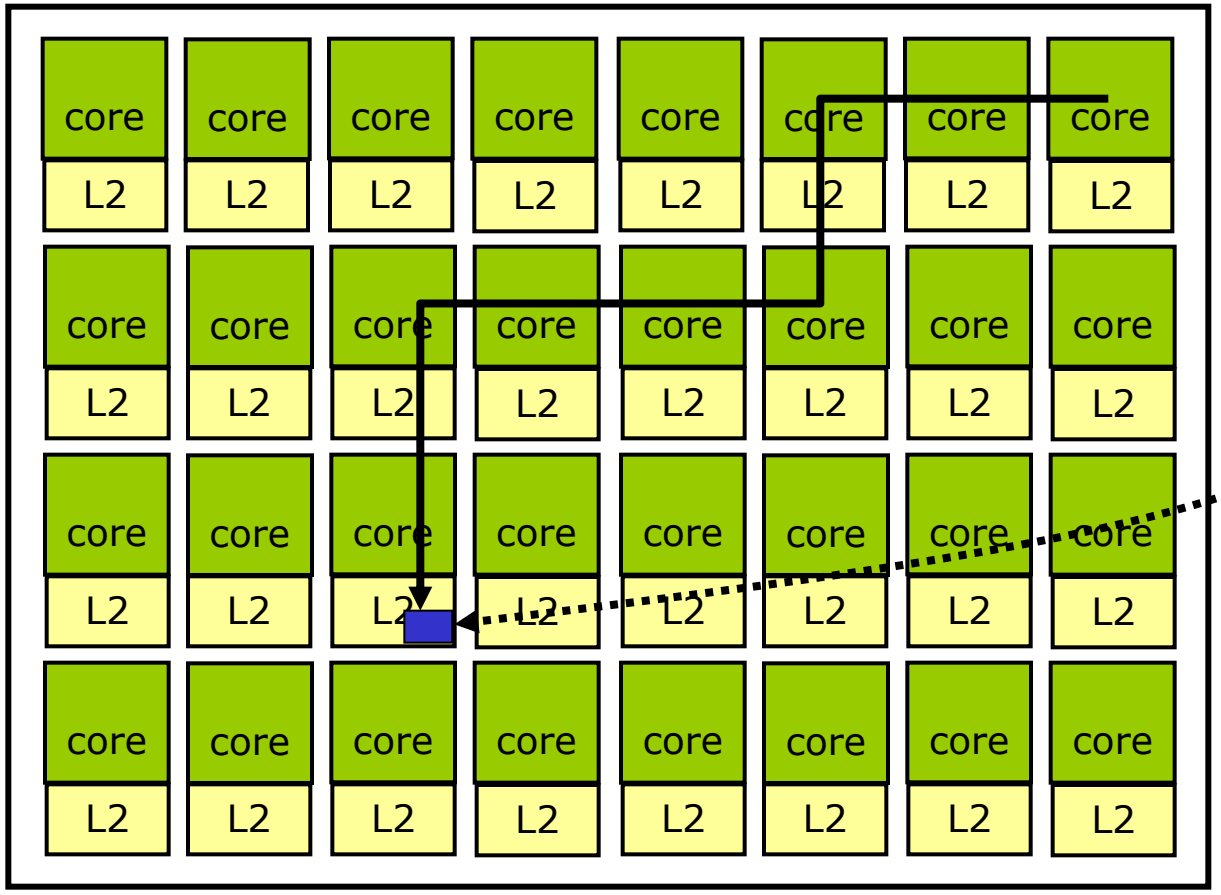


Shared
Read-Only



Shared
Read-Write

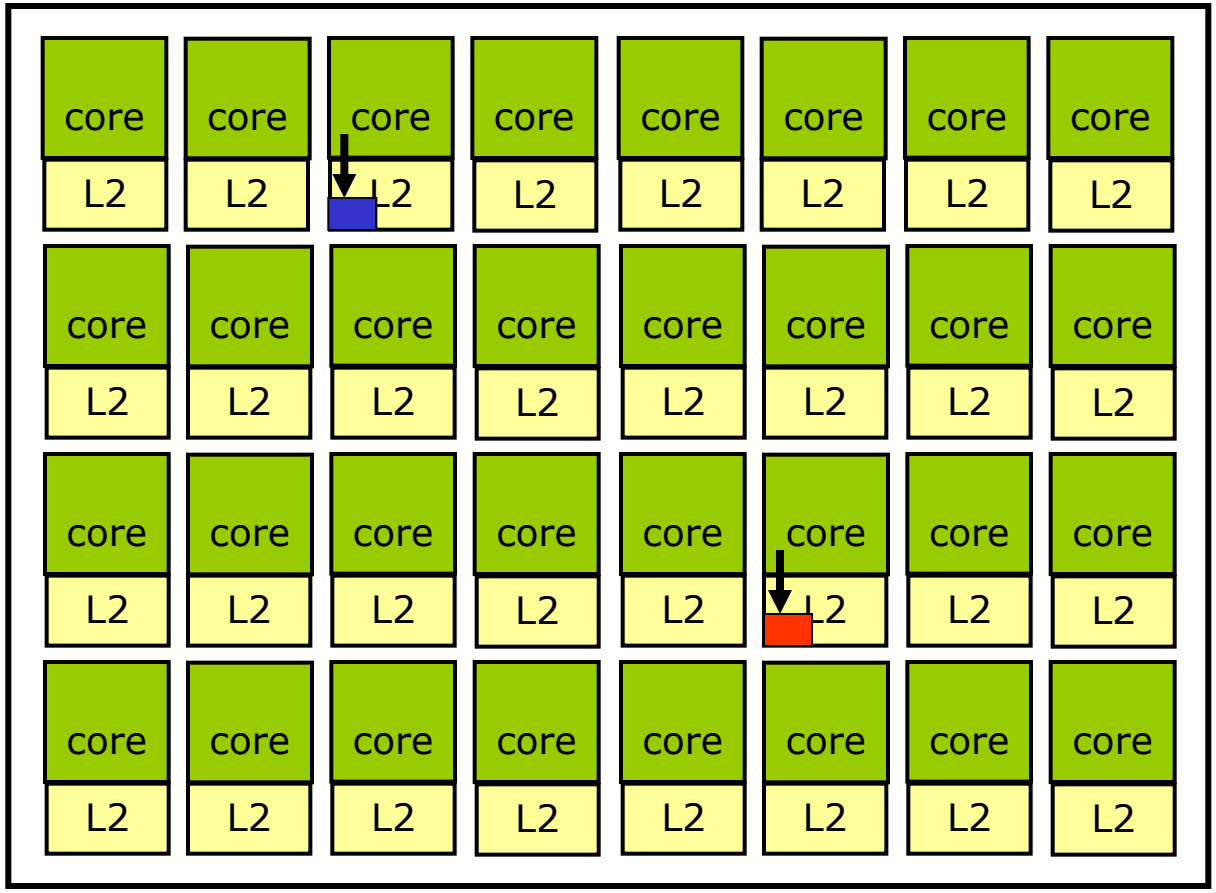
Distributed shared L2



- Maximum capacity, but slow access (40+ cycles)

Can we do better?

Distributed private L2: private data access



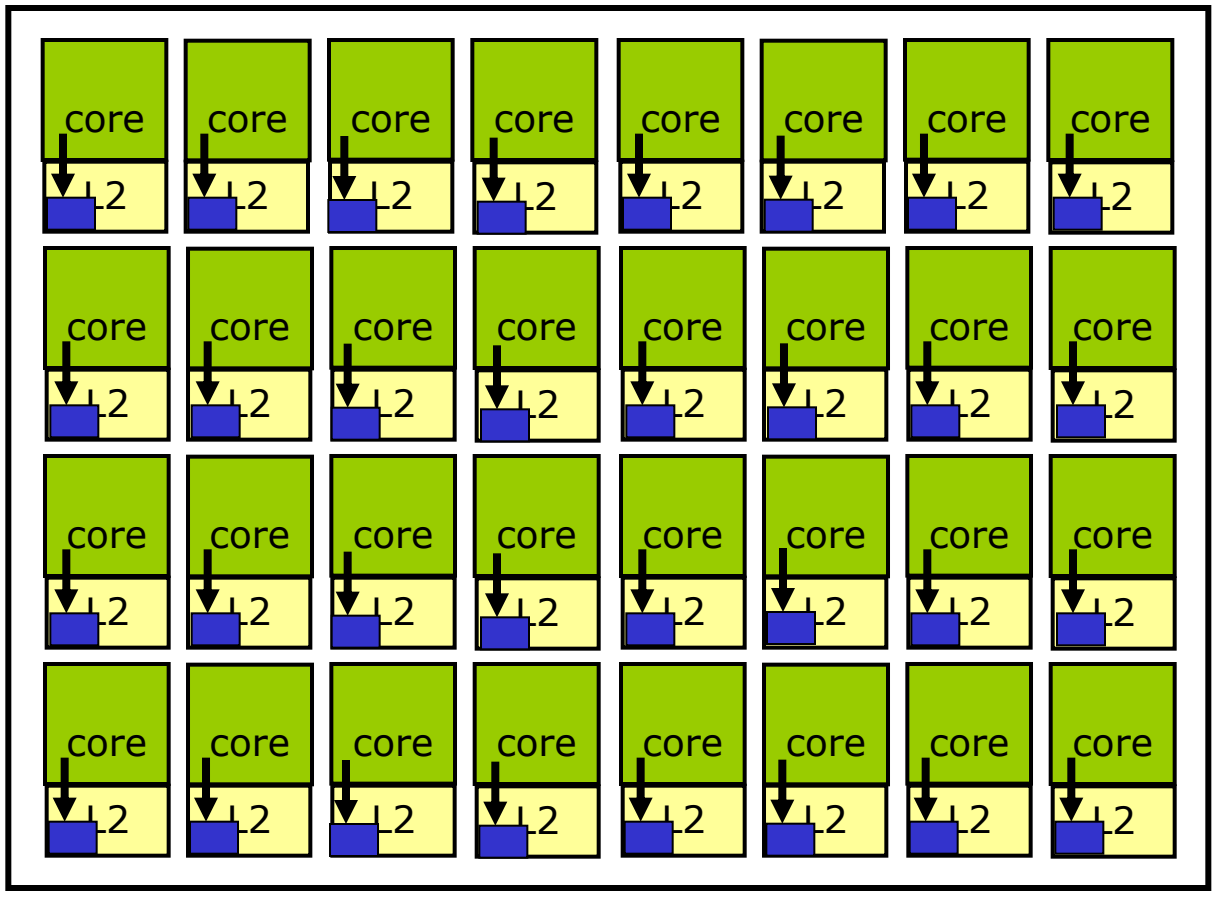
On every access allocate data at local L2 slice

Private data: allocate at local L2 slice

- Fast access to core-private data

What about accesses to shared data?

Distributed private L2: shared-RO access



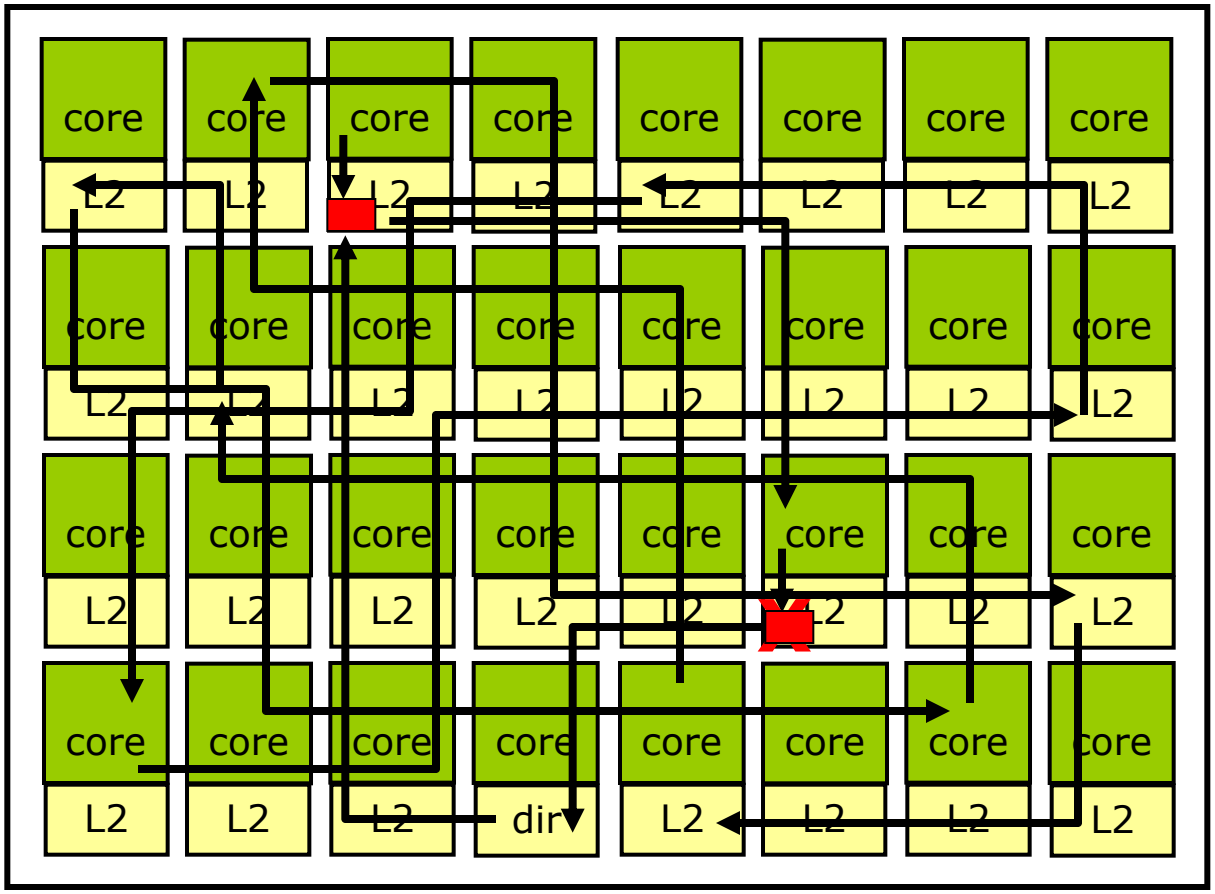
On every access allocate data at local L2 slice

Shared read-only data: replicate across L2 slices

- Wastes capacity due to replication

What about accesses to shared read-write data?

Distributed private L2: shared-RW access



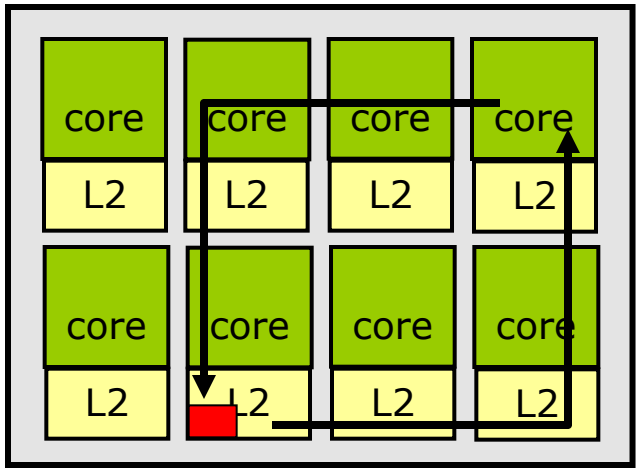
On every access allocate data at local L2 slice

Shared read-write data: maintain **coherence** via indirection (dir)

- Slow for shared read-write
- Wastes capacity (dir overhead) and bandwidth

Conventional Multi-Core Caches

Shared



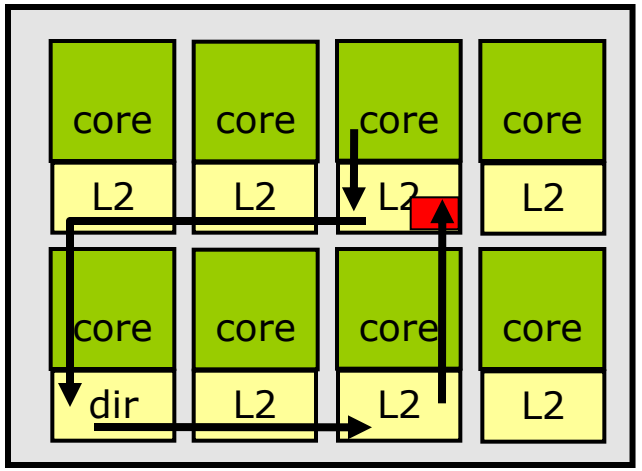
Address-interleave blocks

(sliceID = addr mod #slices)

+ High effective capacity

– Slow access

Private



Each block cached locally

+ Fast access (local)

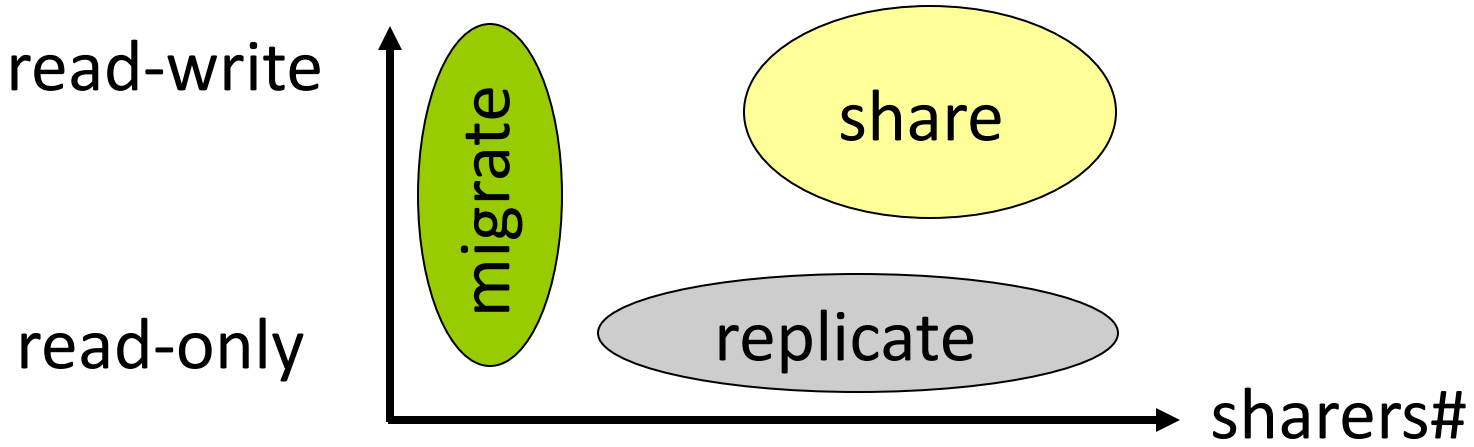
– Low capacity (replicas)

– Coherence: via indirection (distributed directory)

We want: high capacity (shared) + fast access (priv.)

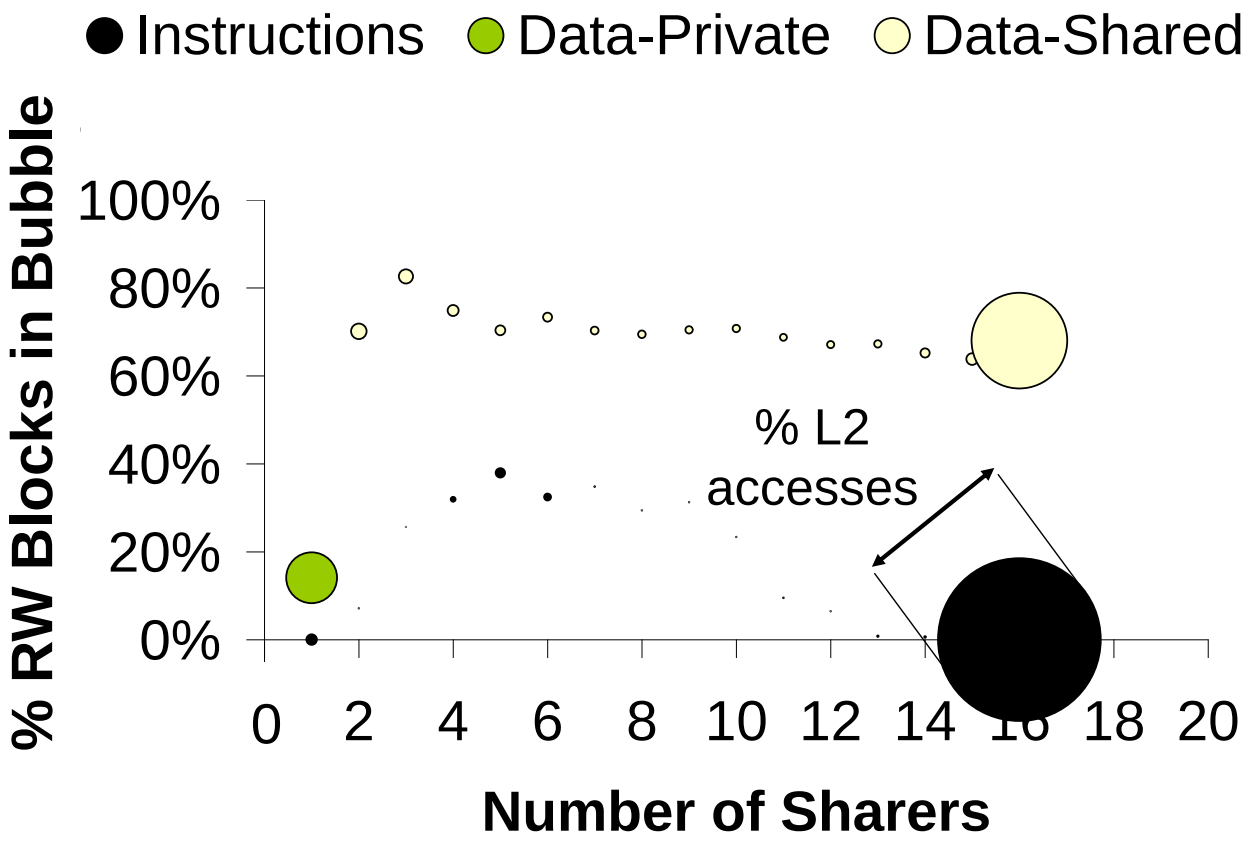
Where to Place the Data?

- Close to where they are used!
- Accessed by single core: migrate locally
- Accessed by many cores: replicate (?)
  - If read-only, replication is OK
  - If read-write, coherence a problem
    - Low reuse: evenly distribute across sharers

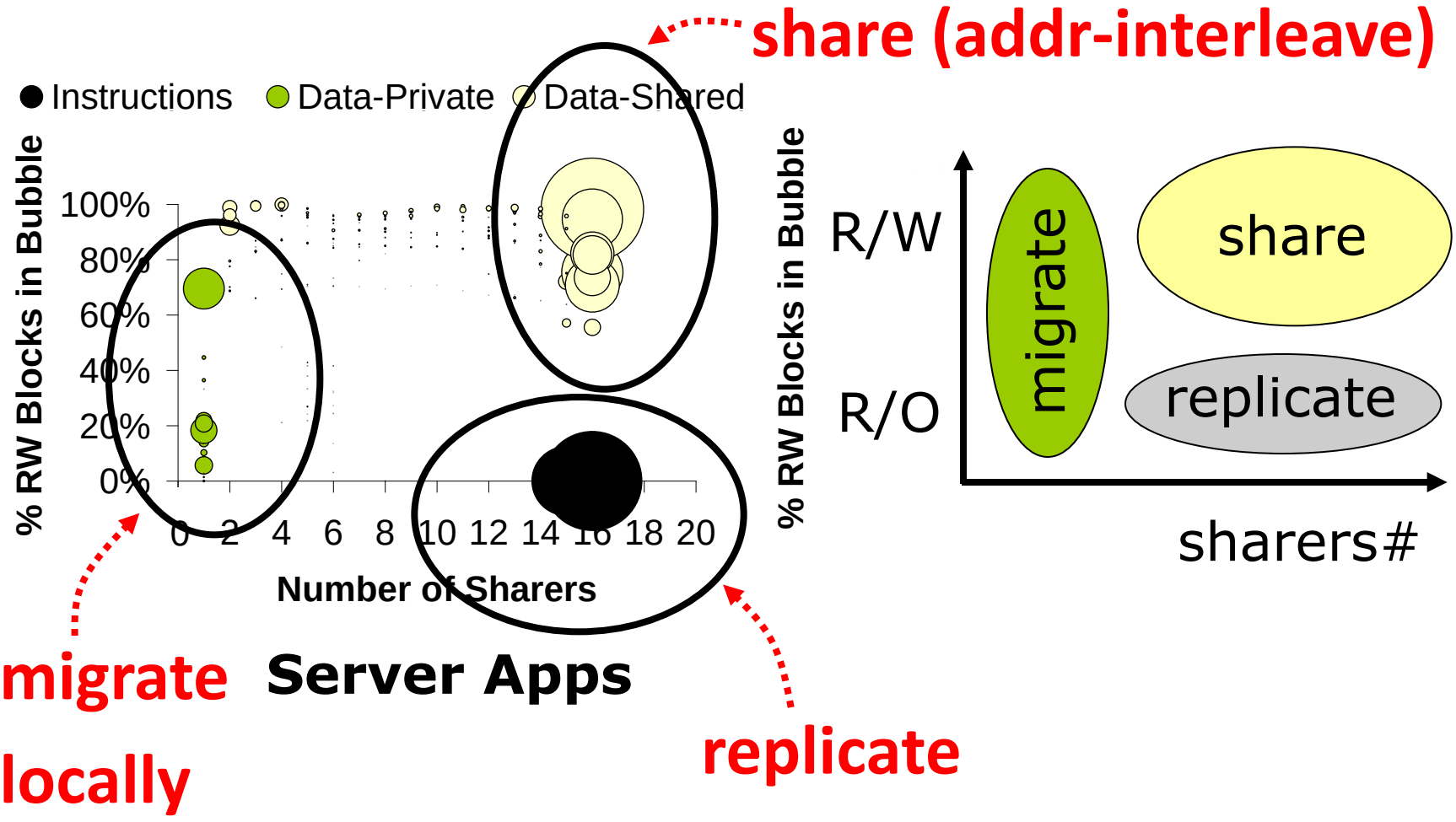


Cache Access Classification Example

- Each bubble: cache blocks shared by x cores
- Size of bubble proportional to % L2 accesses
- y axis: % blocks in bubble that are read-write



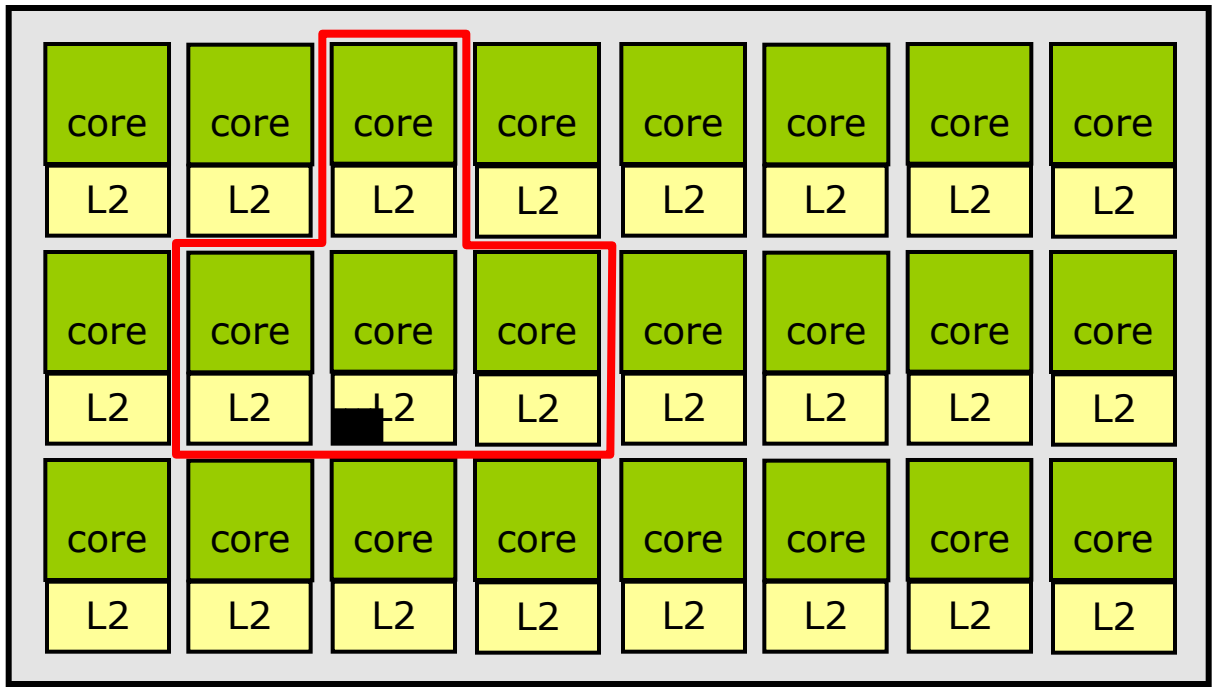
Cache Access Clustering



Accesses naturally form 3 clusters

Instruction Replication

- Instruction working set too large for one cache slice



- Distribute in cluster of neighbors, replicate across

Reactive NUCA in a nutshell

- Classify accesses
  - private data: like private scheme (migrate)
  - shared data: like shared scheme (interleave)
  - instructions: controlled replication (middle ground)

➡ To place cache blocks, we first need to classify them

Classification Granularity

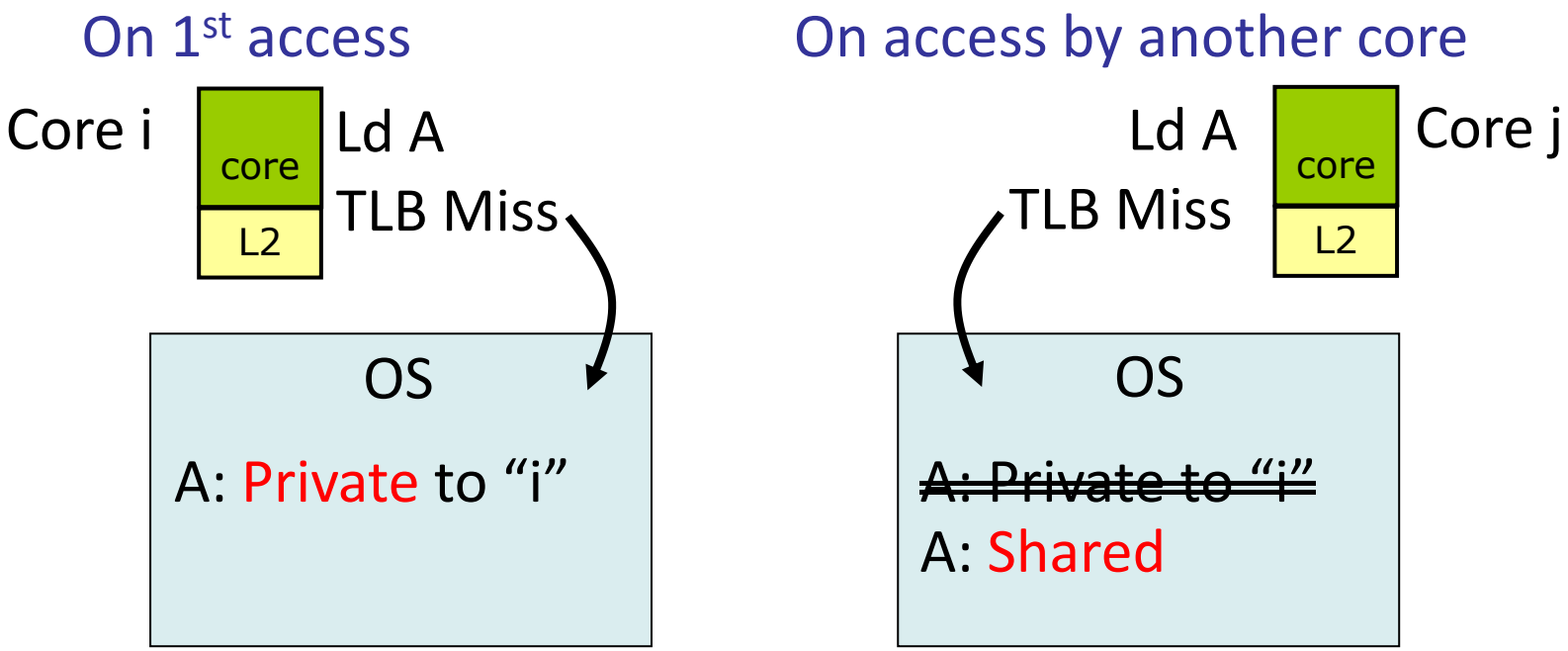
- Per-block classification
  - High area/power overhead (cut L2 size by half)
  - High latency (indirection through directory)

- Per-page classification (utilize OS page table)
  - Persistent structure
  - Core accesses the page table for every access anyway (TLB)
  - Utilize already existing SW/HW structures and events
  - Page classification is accurate (<0.5% error)

➡ Classify entire data pages, page table/TLB for bookkeeping

Classification Mechanisms

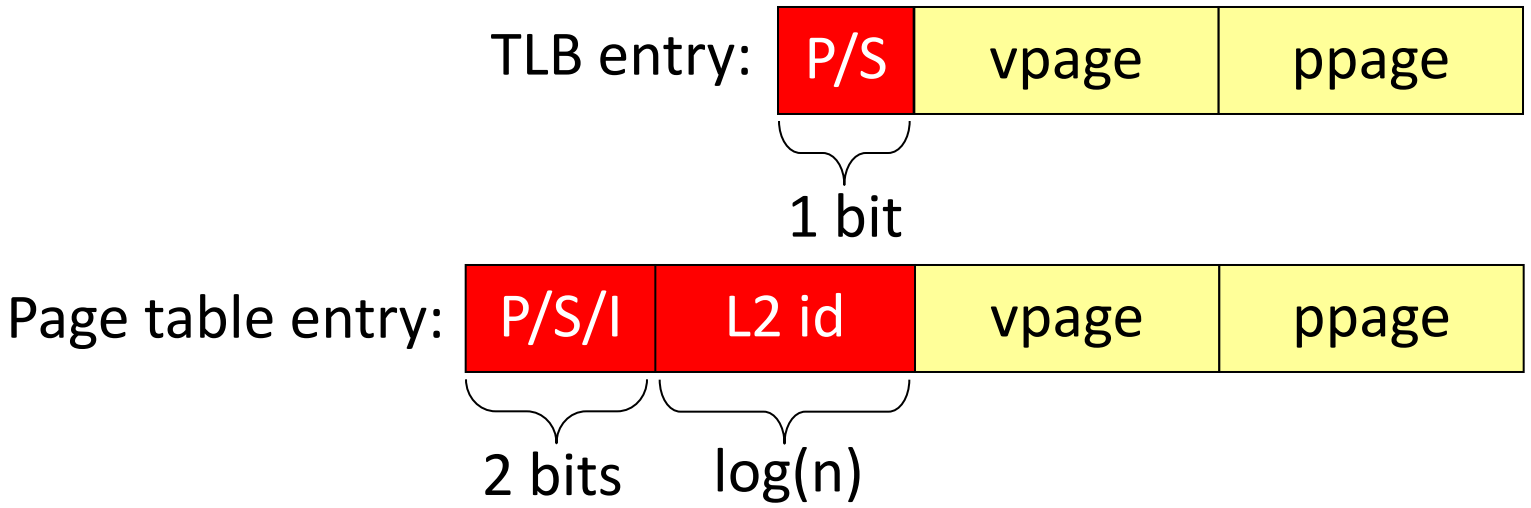
- Instructions classification: all accesses from L1-I (per-block)
- Data classification: private/shared per-page at TLB miss
  - Page classification is accurate (<0.5% error)



Bookkeeping through OS page table and TLB

Page Table and TLB Extensions

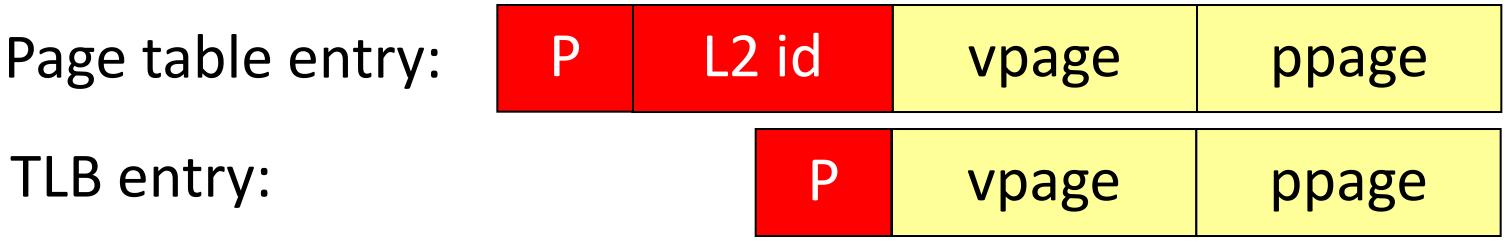
- Persistent structure, ideal for “directory”
- Core accesses the page table for every access anyway (TLB)
  - Pass information from the “directory” to the core
- Utilize already existing SW/HW structures and events



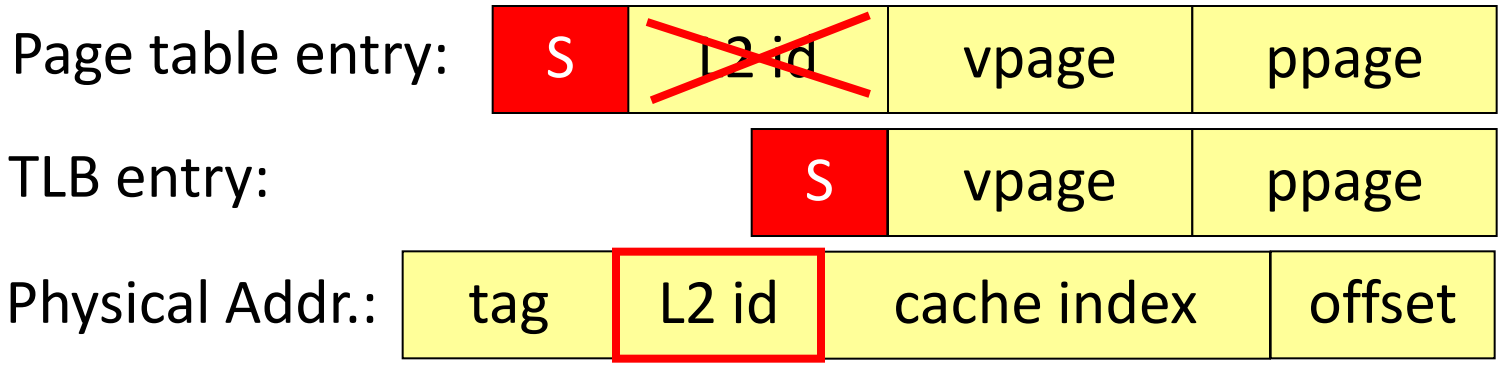
Page granularity allows simple + practical HW

Data Class Bookkeeping and Lookup

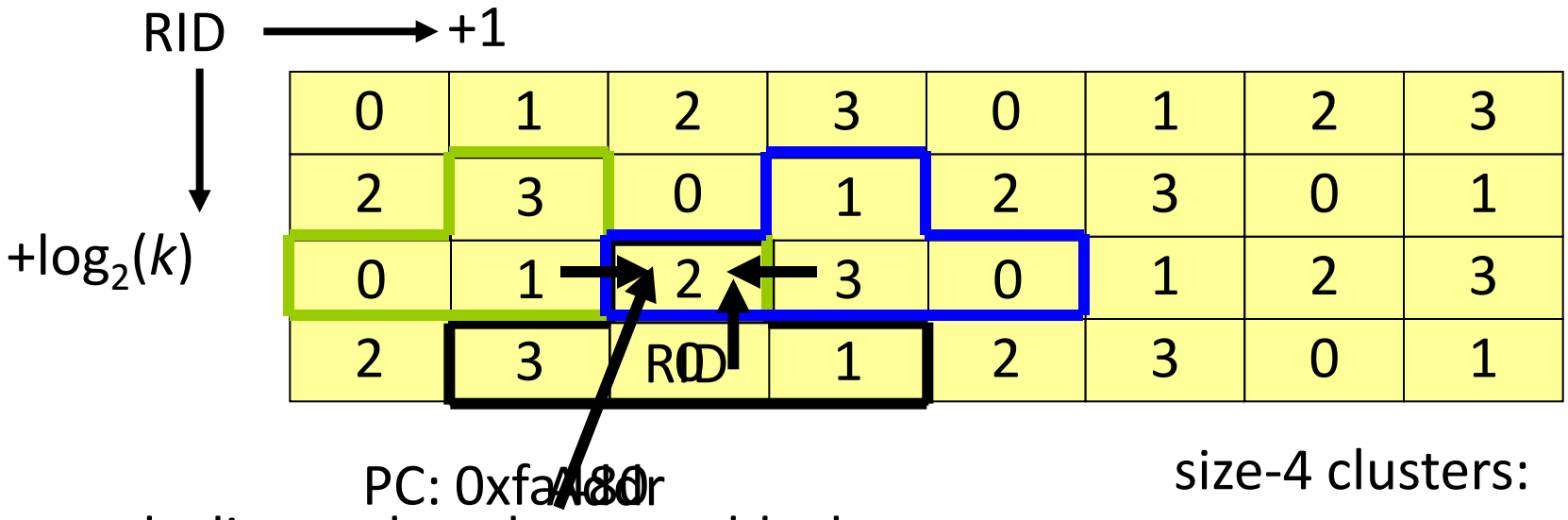
- **private data**: place in local L2 slice



- **shared data**: place in aggregate L2 (addr interleave)



Instructions Lookup: Rotational Interleaving



each slice caches the same blocks

on behalf of any cluster

size-4 clusters:

local slice + 3 neighbors

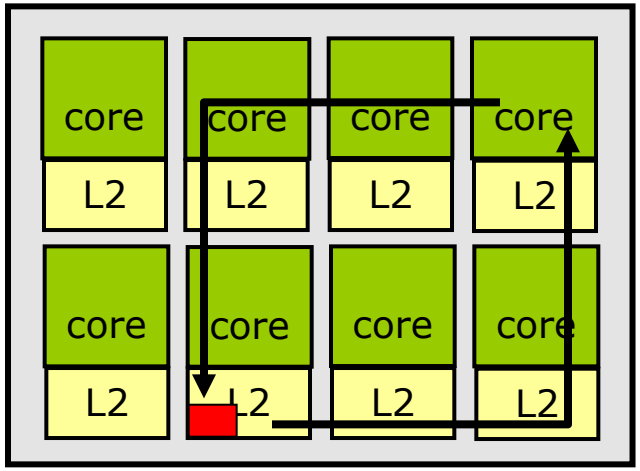
$$Destination = \left(Addr + \overline{RID} + 1\right) \& (n-1)$$

- Fast access (nearest-neighbor, simple lookup)
- Balance access latency with capacity constraints
- Equal capacity pressure at overlapped slices

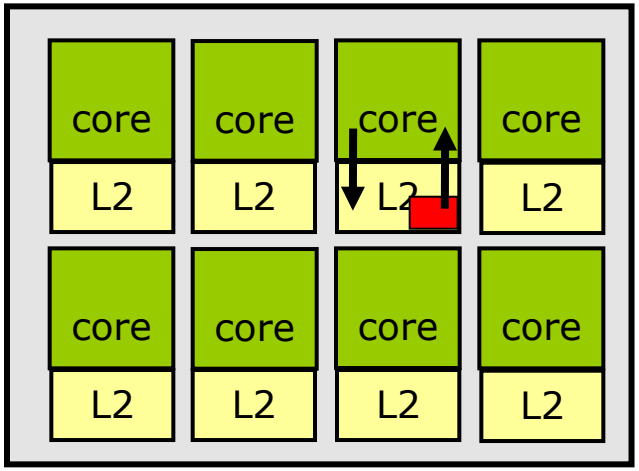
Coherence: No Need for HW Mechanisms at L2

- Reactive NUCA placement guarantee
  - Each R/W datum in unique & known location

Shared data: addr-interleave

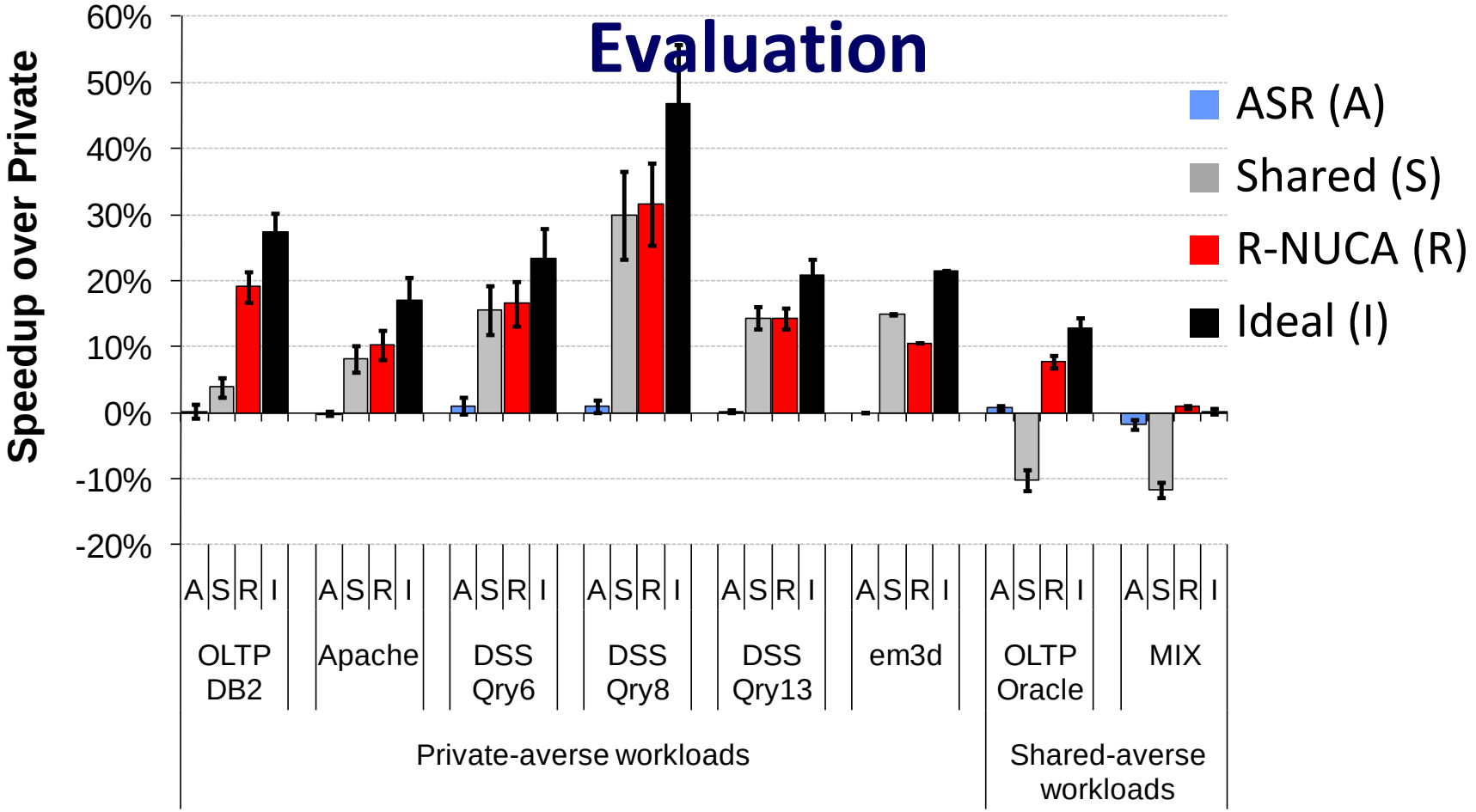


Private data: local slice



Fast access, eliminates HW overhead, SIMPLE

Evaluation



- **Delivers robust performance across workloads**
  - Shared: same for Web, DSS; **17%** for OLTP, MIX
  - Private: **17%** for OLTP, Web, DSS; same for MIX

Reactive NUCA: Fast >>MB Caches

- Data may exhibit arbitrarily complex behaviors
  - ...but few that matter!
- Learn the behaviors at run time for placement
  - Make the common case fast
- Fast!
  - Near-optimal placement (within 5% of ideal)
  - Robust (matches best alternative, or 17% better; up to 32%)
  - Nearest-neighbor communication → scalable
- Transparent to the user, simple design, negligible overhead
- **BONUS**: simplify hardware
  - Eliminate HW coherence at L2

Concluding Remarks

- The old multiprocessor/cluster is now within a single chip
- Still a distributed system
- Similar challenges, with new constants
  - Lecture focused on performance via data placement
  - Similar challenge: HPC systems strive to privatize data
    - R-NUCA strives to use “private caches”
  - New constants: latency, bandwidth,
    - Single OS image allows for many simplifications

- Research opportunity: map ideas from old domain to new

Multicore Research Brings Opportunities and Challenges

"Multicore: This is the one which will have the biggest impact on us. We have never had a problem to solve like this. A breakthrough is needed in how applications are done on multicore devices."

– *Bill Gates*

"It's time we rethink some of the basics of computing. It's scary and lots of fun at the same time."

– *Burton Smith*