Memory Hierarchy Review

Outline

1. Why memory hierarchy?

2. How caches work

3. Cache performance

Part 1: Why Memory Hierarchy?

What is it?

Memory Hierarchy ⇔ A type of memory system design that uses multiple levels; as the distance from the CPU increases, the size of the memories and the access time also increase.

Why design memory in a hierarchy?

Technology/Economics – There exist a variety of ways to design program memory which span the price-performance spectrum

Demand – memory access is a performance and/or cost bottleneck

Capability – Programs behave in such way that make memory hierarchies advantageous.

Look at these in order →

Storage Trends

SRAM

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	19,200	2,900	320	256	100	75	60	320x
access (ns)	300	150	35	15	3	2	1.5	200x

DRAM

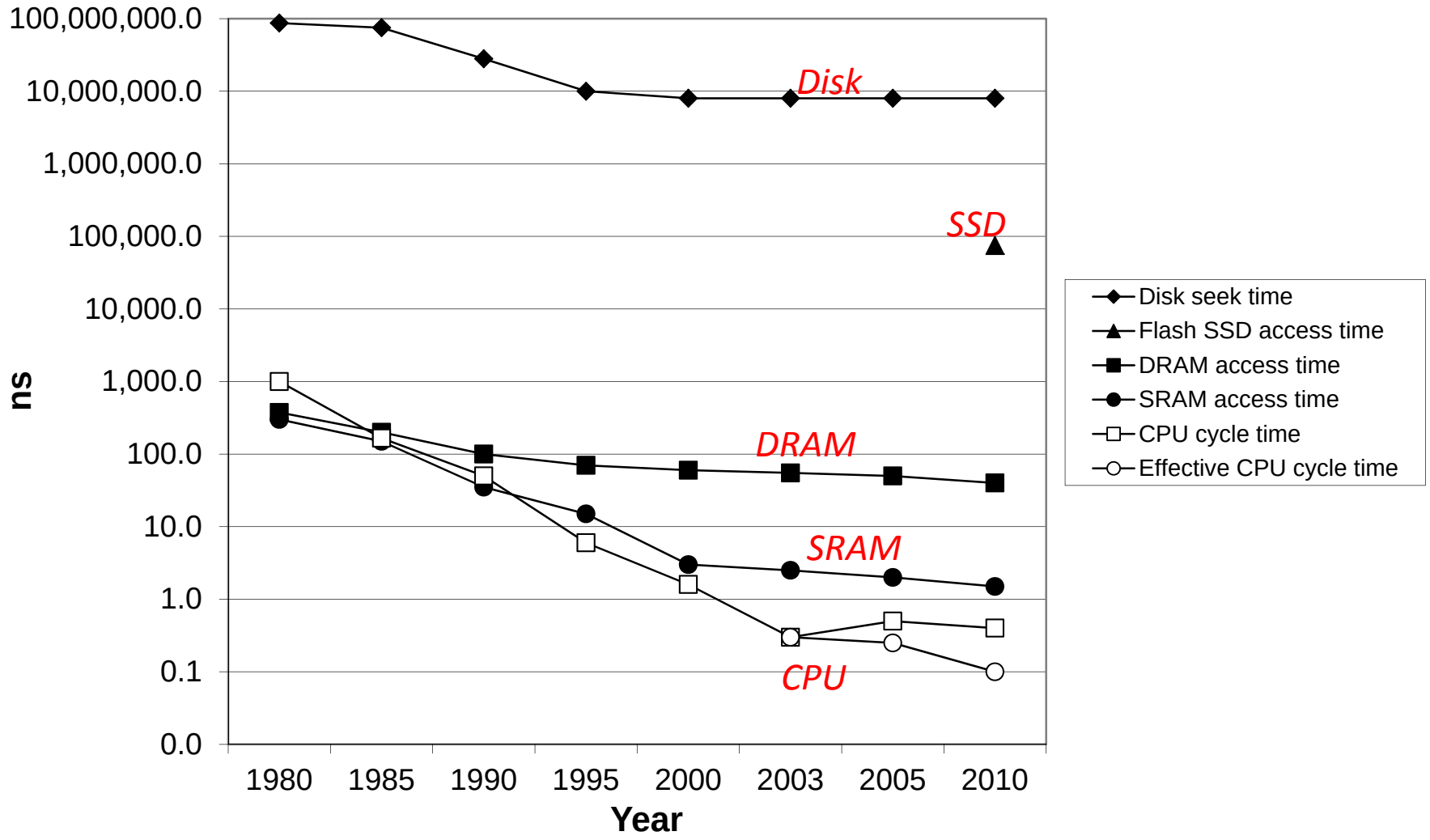
Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	8,000	880	100	30	1	0.1	0.06	130,000x
access (ns)	375	200	100	70	60	50	40	9x
typical size (MB)	0.064	0.256	4	16	64	2,000	8,000	125,000x

Disk

Metric	1980	1985	1990	1995	2000	2005	2010	2010:1980
$/MB	500	100	8	0.30	0.01	0.005	0.0003	1,600,000x
access (ms)	87	75	28	10	8	4	3	29x
typical size (MB)	1	10	160	1,000	20,000	160,000	1,500,000	1,500,000x

The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.

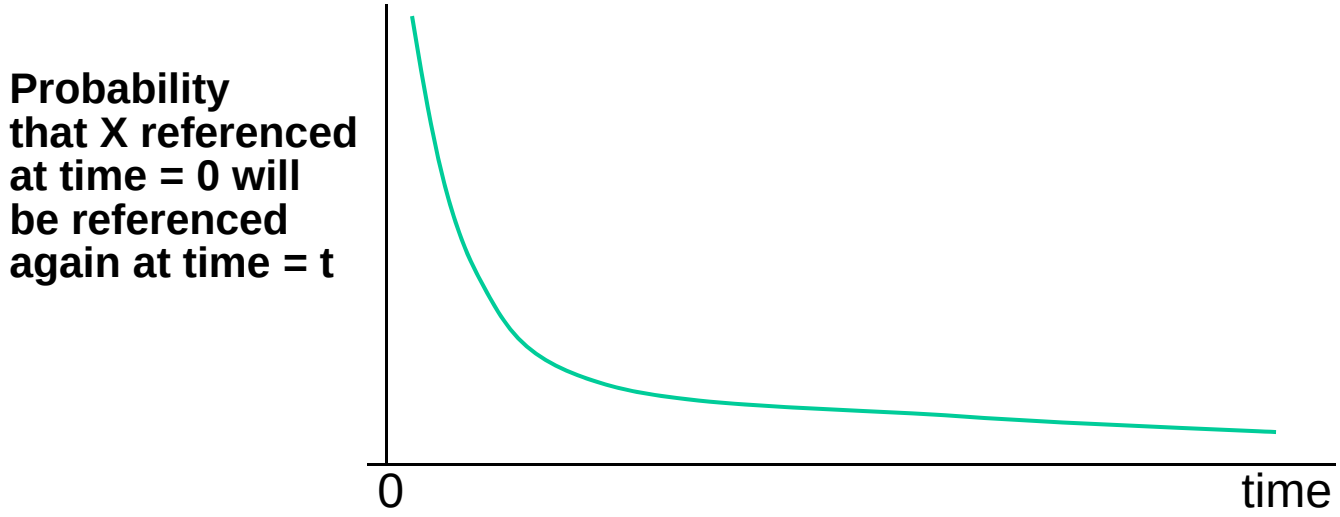


The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

Properties of Programs – Temporal Locality

Temporal Locality ⇔

If an item is referenced, it will tend to be referenced again soon.

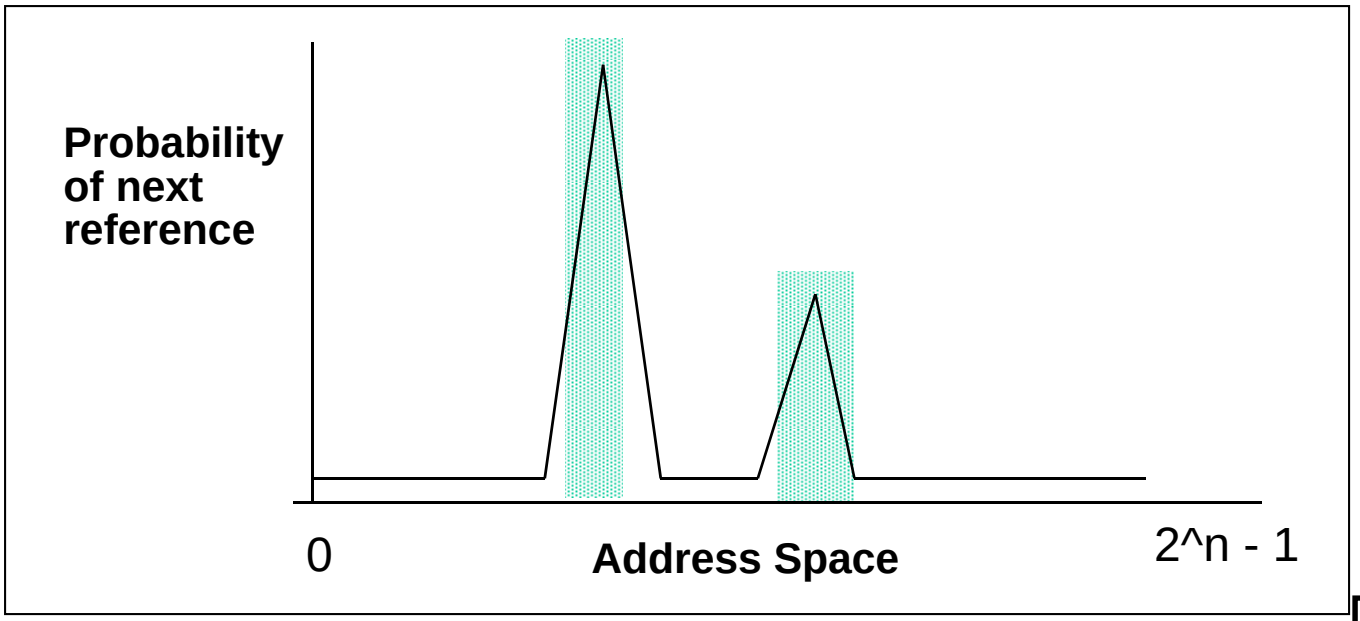


How to take advantage: *Keep most recently accessed data close to the processor*

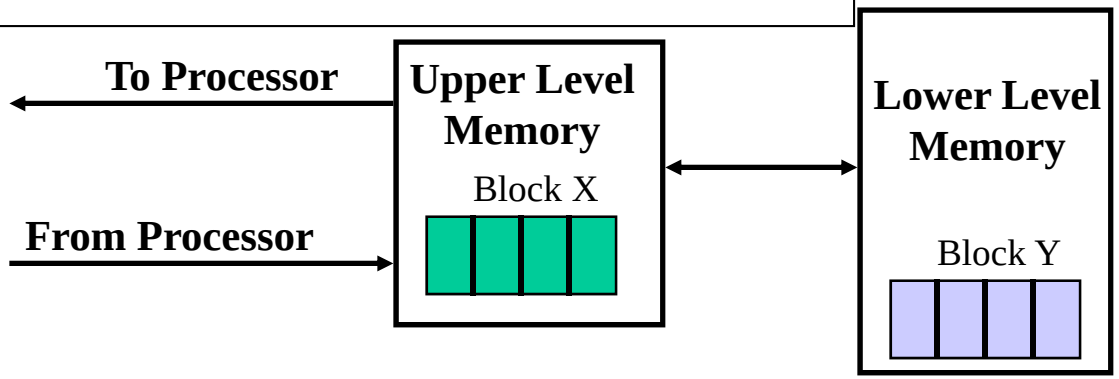
Properties of Programs – Spatial Locality

Spatial Locality ⇔

- If an item is referenced, items whose addresses are close by will tend to be referenced soon.
- At any instant, programs access a relatively small part of the address space.



How to take advantage:
When data are needed in a higher level of memory, move surrounding data as well (blocks)



Ex: → What are the locality properties of this code fragment?

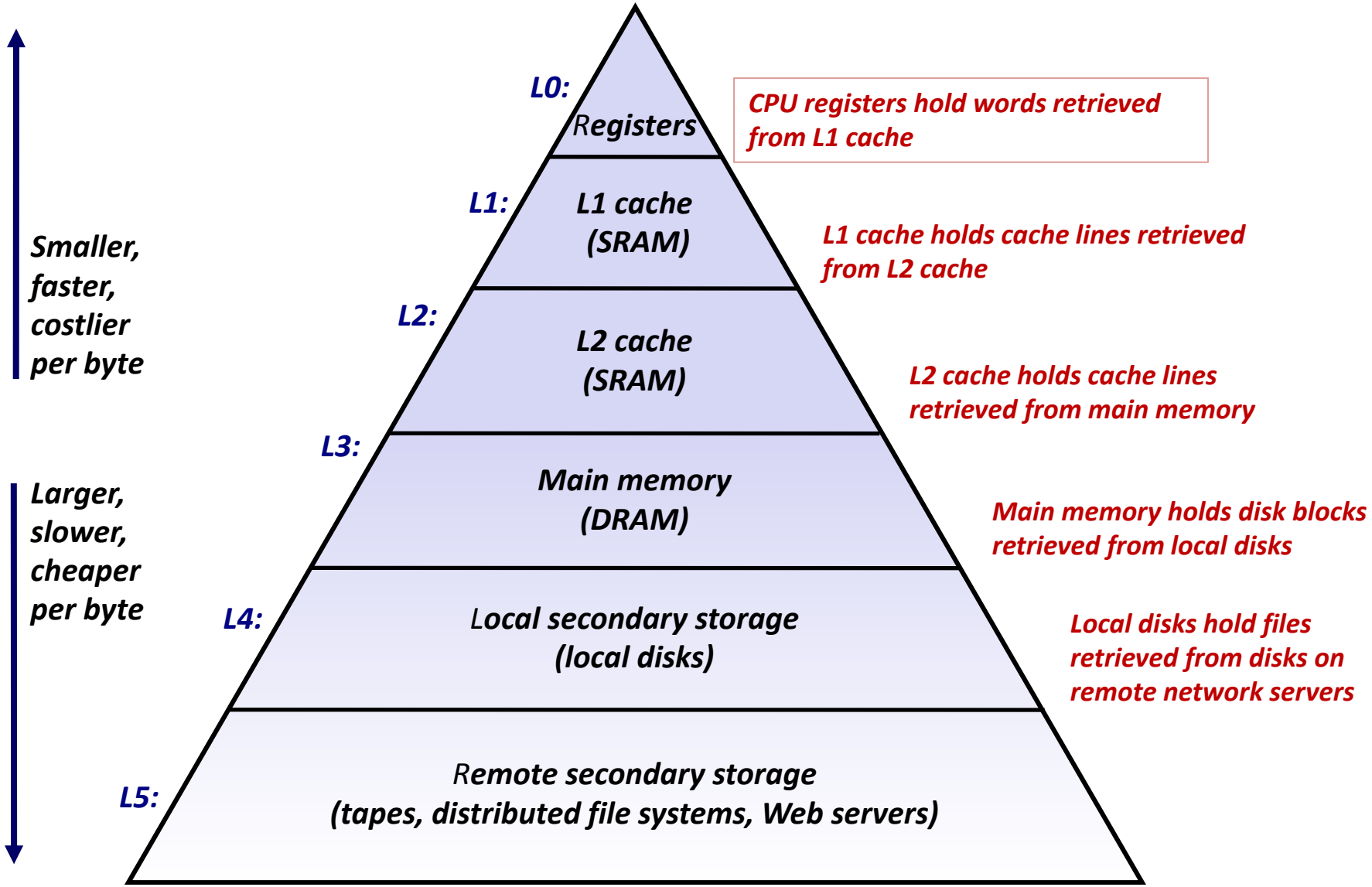
```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- Data references
  - Reference array elements in succession (stride-1 reference pattern). — *Spatial locality*
  - Reference variable **`sum`** each iteration. — *Temporal locality*
- Instruction references
  - Reference instructions in sequence. — *Spatial locality*
  - Cycle through loop repeatedly. — *Temporal locality*

Four kinds of locality (or lack of locality):

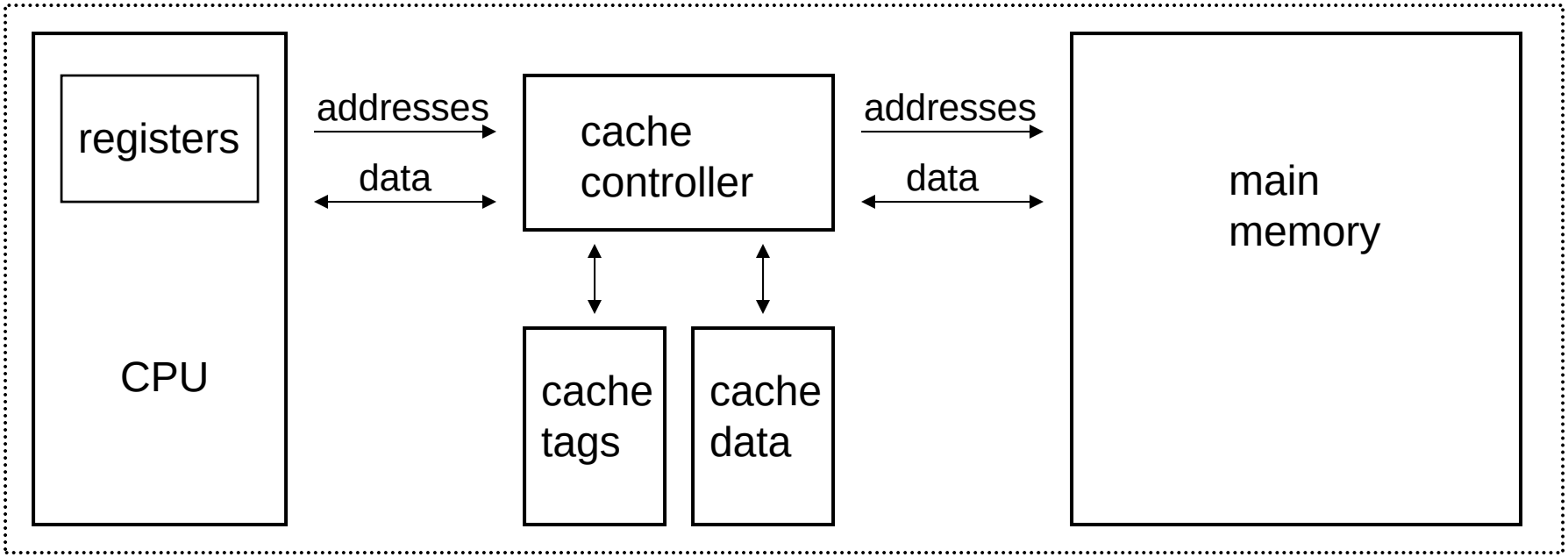
	Instruction Stream	Data Stream
Temporal Locality		
Spatial Locality		

- Some fundamental and enduring properties of hardware and software:
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - The gap between CPU and main memory speed is widening.
  - Well-written programs tend to exhibit good locality.
- These fundamental properties complement each other beautifully.
- They suggest an approach for organizing memory and storage systems known as a memory hierarchy.



Part 2: How Caches Work

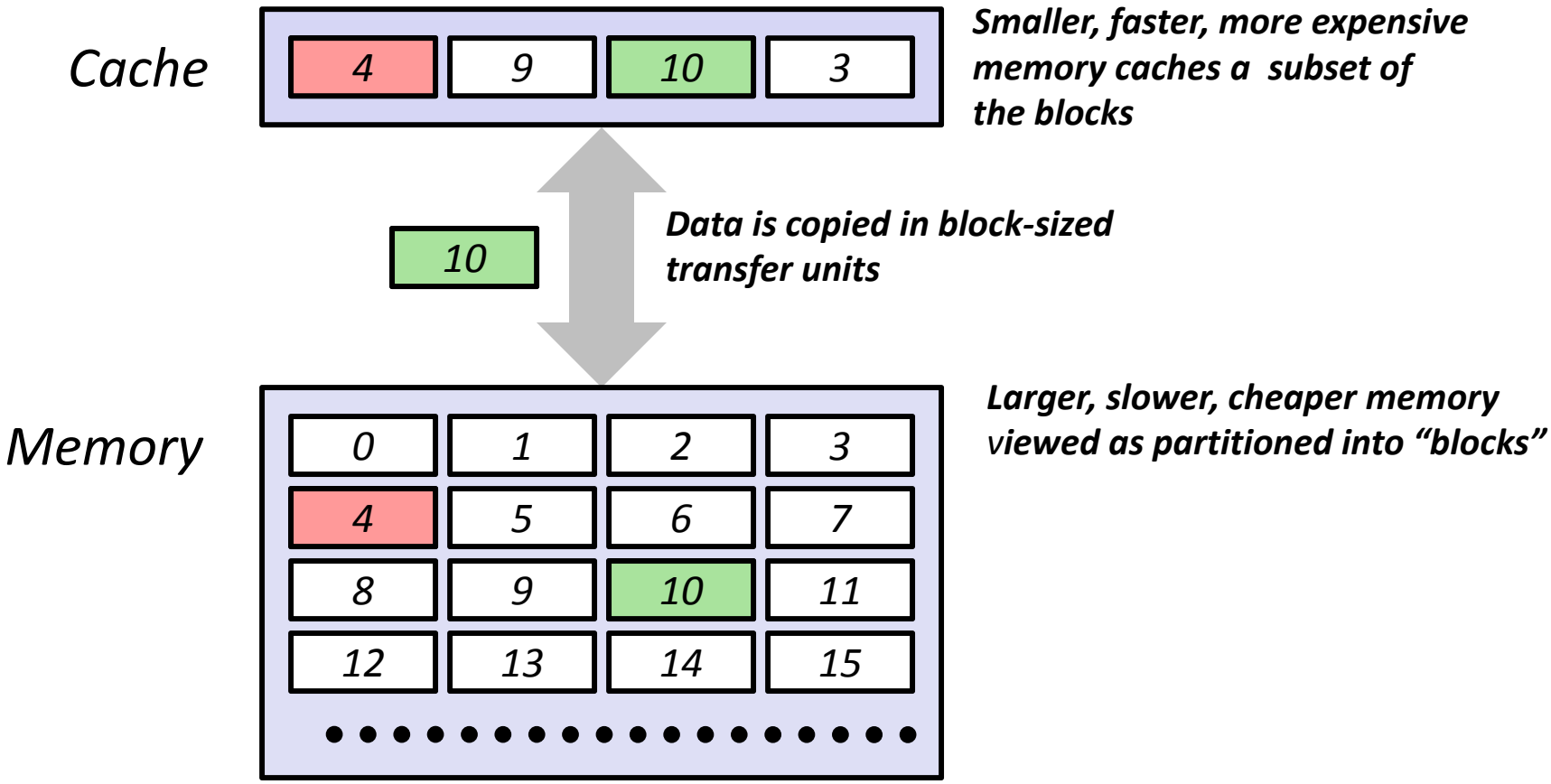
- CPU generates a memory access request
- Cache controller intercepts request, checks to see whether data is in cache
  - HIT (data is in cache): Cache is accessed and data is sent to CPU
  - MISS (data is not in cache): Memory is accessed, copy of data is put into cache, and data is sent to the CPU.
- Note: CPU is oblivious to all this!



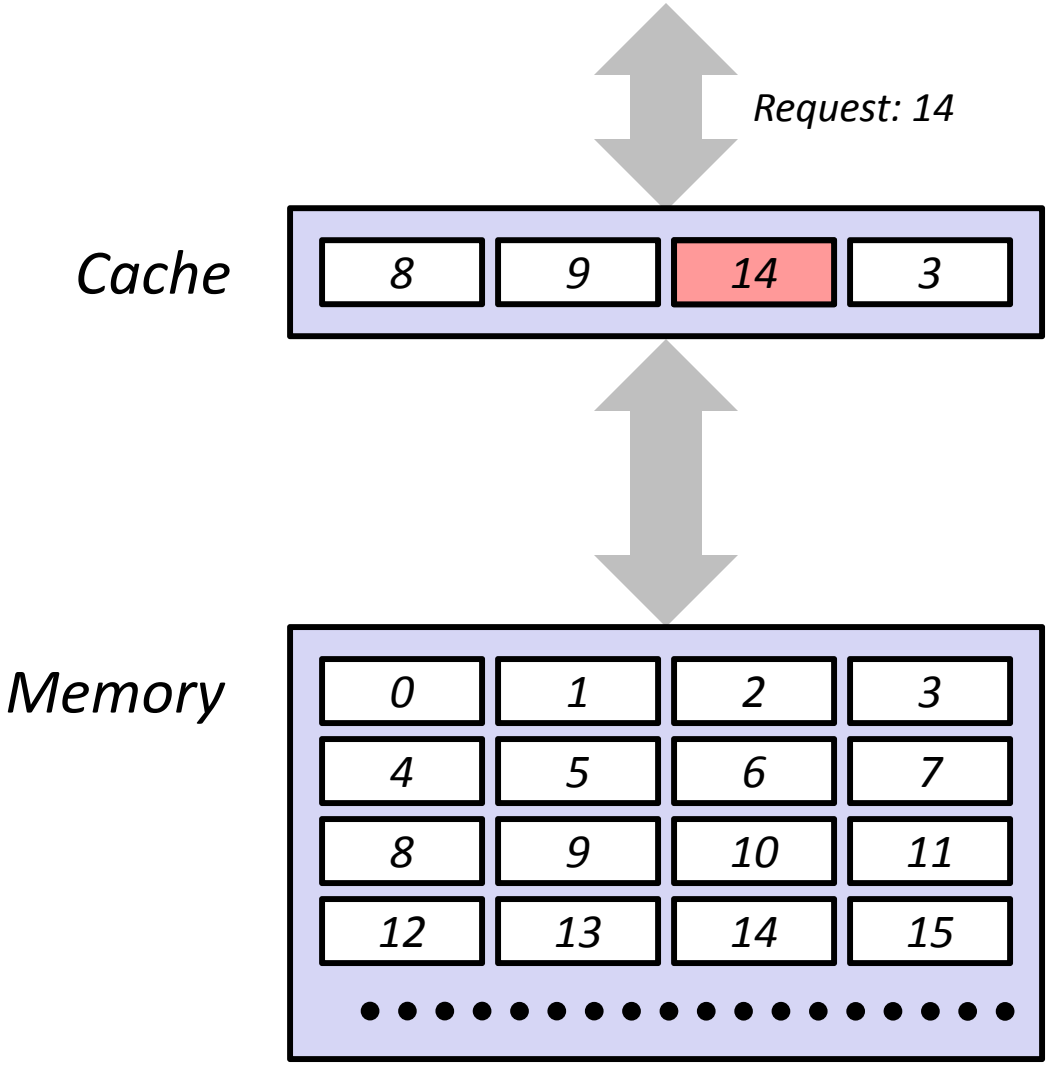
Tag ⇔ Info that uniquely characterizes a ***block*** of data, usually derived from the address of the block

Block ⇔ The unit of interaction (the chunk of data that gets transferred) between levels of a memory hierarchy

General Cache Concepts



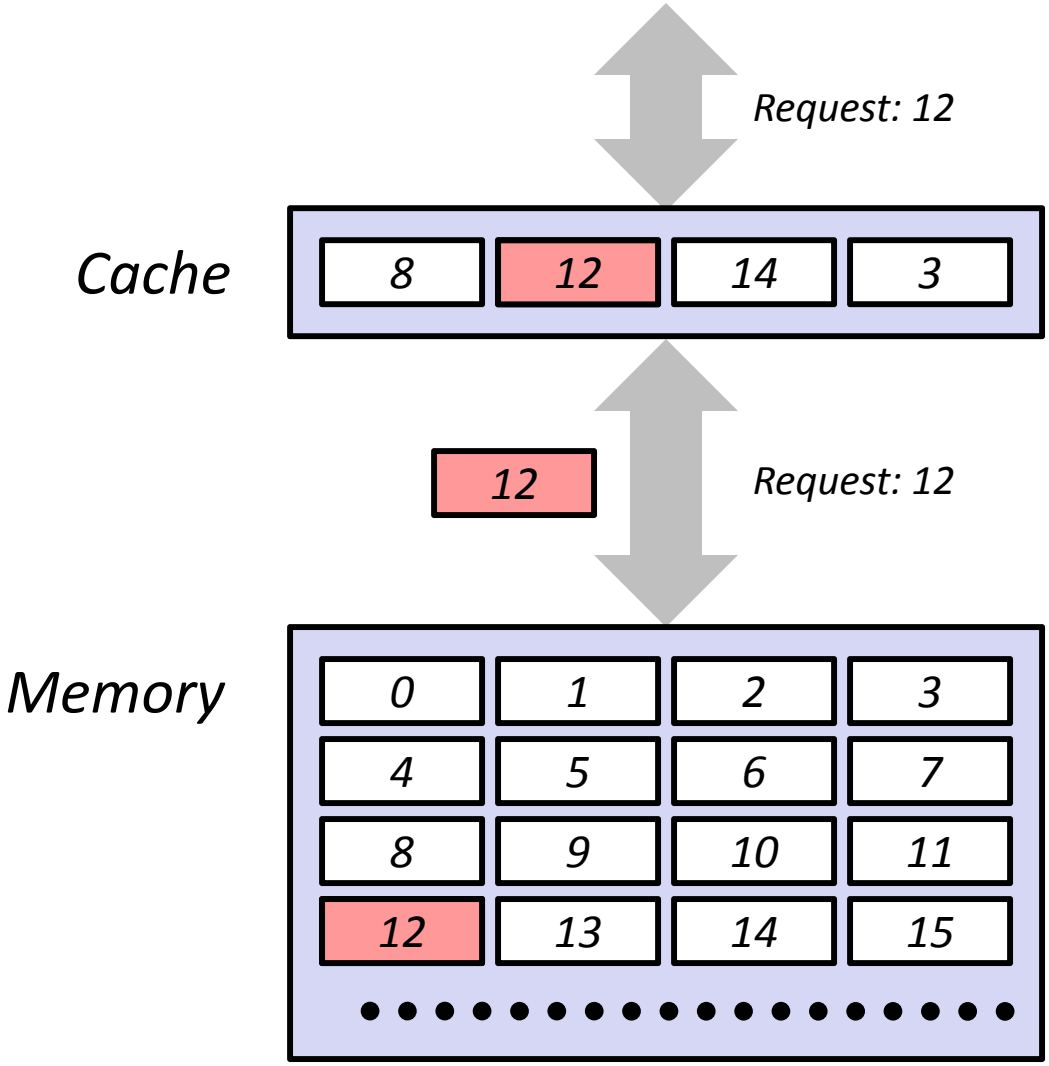
General Cache Concepts: Hit



Data in block b is needed

Block b is in cache:
Hit!

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

Block b is fetched from *memory*

Block b is stored in cache

- *Placement policy:* *determines where b goes*
- *Replacement policy:* *determines which block gets evicted (victim)*

Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware
L1 cache	64-bytes block	On-Chip L1	1	Hardware
L2 cache	64-bytes block	On/Off-Chip L2	10	Hardware
Virtual Memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Disk firmware
Network buffer cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server

Summary

- *The speed gap between CPU, memory and mass storage continues to widen.*
- *Well-written programs exhibit a property called locality.*
- *Memory hierarchies based on caching close the gap by exploiting locality.*

Part 3: Performance → Simple Performance Models

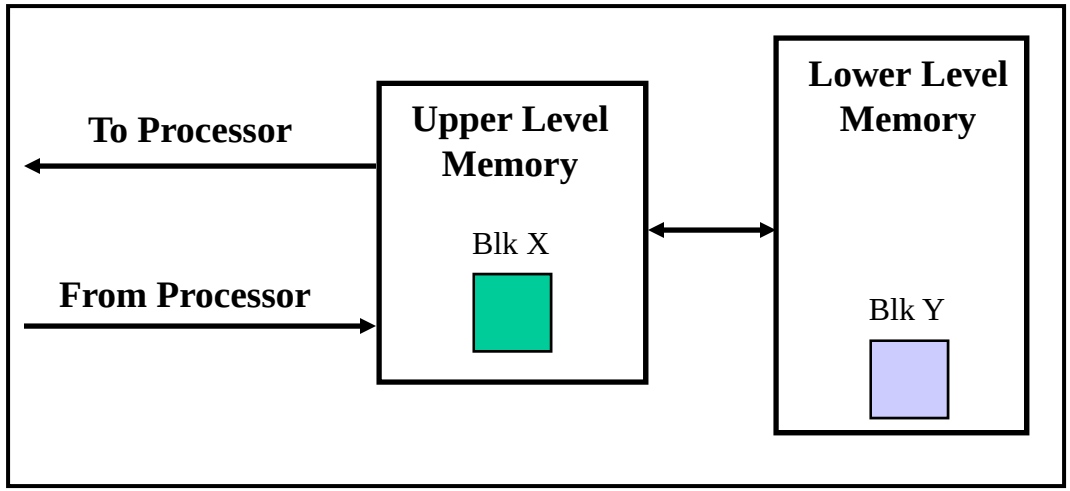
Hit ⇔ data appears in some block in the upper level (example: Block X)

- **Hit Rate** ⇔ the fraction of memory access found in the upper level
- **Hit Time** ⇔ Time to access the upper level which consists of
  - → RAM access time + Time to determine hit/miss

Miss ⇔ data needs to be retrieved from a block in the lower level (Block Y)

- **Miss Rate** ⇔ 1 - (Hit Rate)
- **Miss Penalty** ⇔ Time to replace a block in the upper level + Time to deliver the block the processor

➔ Hit Time << Miss Penalty



How is the hierarchy managed?

- Registers ⇔ Memory
  - by compiler (programmer?)
- Cache ⇔ Memory
  - by the hardware
- Memory ⇔ Disks
  - by the hardware and operating system (virtual memory)
  - by the programmer (files)

Simplest Performance Model: Example

Average Memory Access Time =

Hit Rate * Hit Time + Miss Rate * Miss Penalty

Simple Example:

➔ Hit Rate = 90% Hit Time = 2ns Miss Penalty = 100ns

➔ Average Memory Access Time = .9 * 2 + .1 * 100 = 11.8ns

Look at some more Hit Rates

Hit Rate	Average Memory Access Time
50%	51 ns
90%	11.8 ns
95%	6.9 ns
98%	3.96 ns
99%	2.98 ns

Designing Cache for Performance: Taking Advantage of Spatial Locality

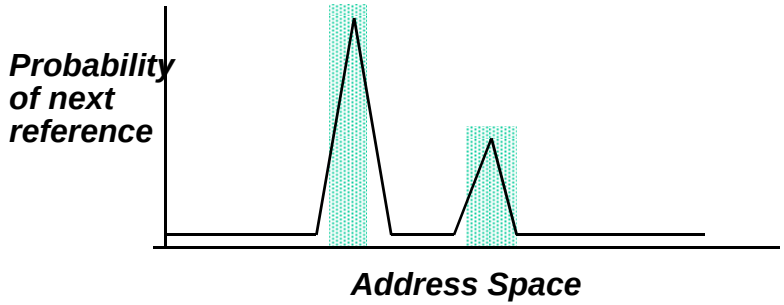
In the simple model of CPU/Register File – Cache – Main Memory there are two levels of interaction and two block sizes:

- CPU/Register File – Cache ➔ block size = 1 word
- Cache – Main memory ➔ block size has been 1 word so far

This scheme takes advantage of temporal, but not spatial locality.

Let cache – main memory block > 1 word. Then

- a cache miss will cause multiple adjacent words to be fetched from main memory.
- With high probability, these words will be used soon, resulting in fewer cache misses.



Miss Rate Versus Block Size

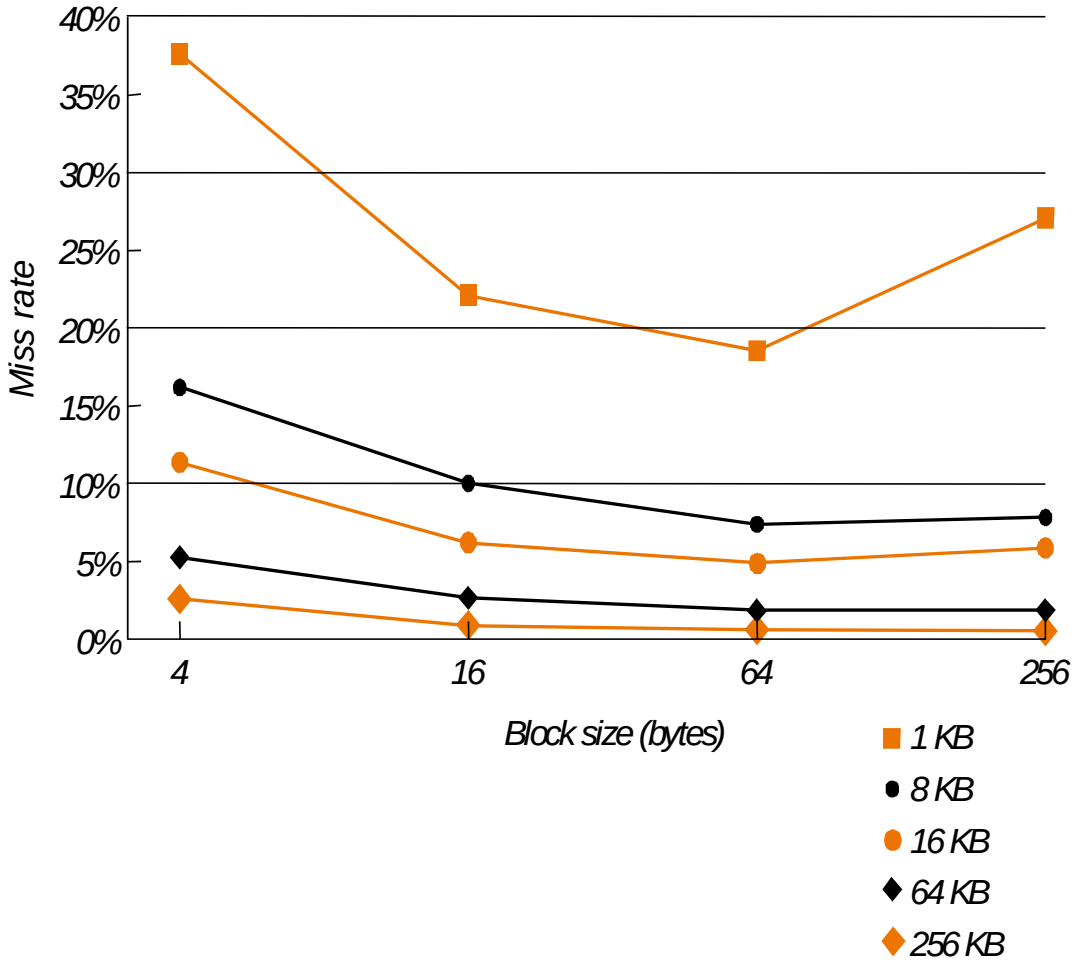
Performance depends on cache size as well as block size.

Increased block size takes advantage of spatial locality.

Increased block size increases miss penalty.

“Miss rate” as performance measure ignores increased cost of transfer of larger blocks.

Increasing block size too much can also be counter productive even ignoring transfer time. Why?



Design for performance: Multilevel Caches

How big should a cache be?

Two conflicting goals:

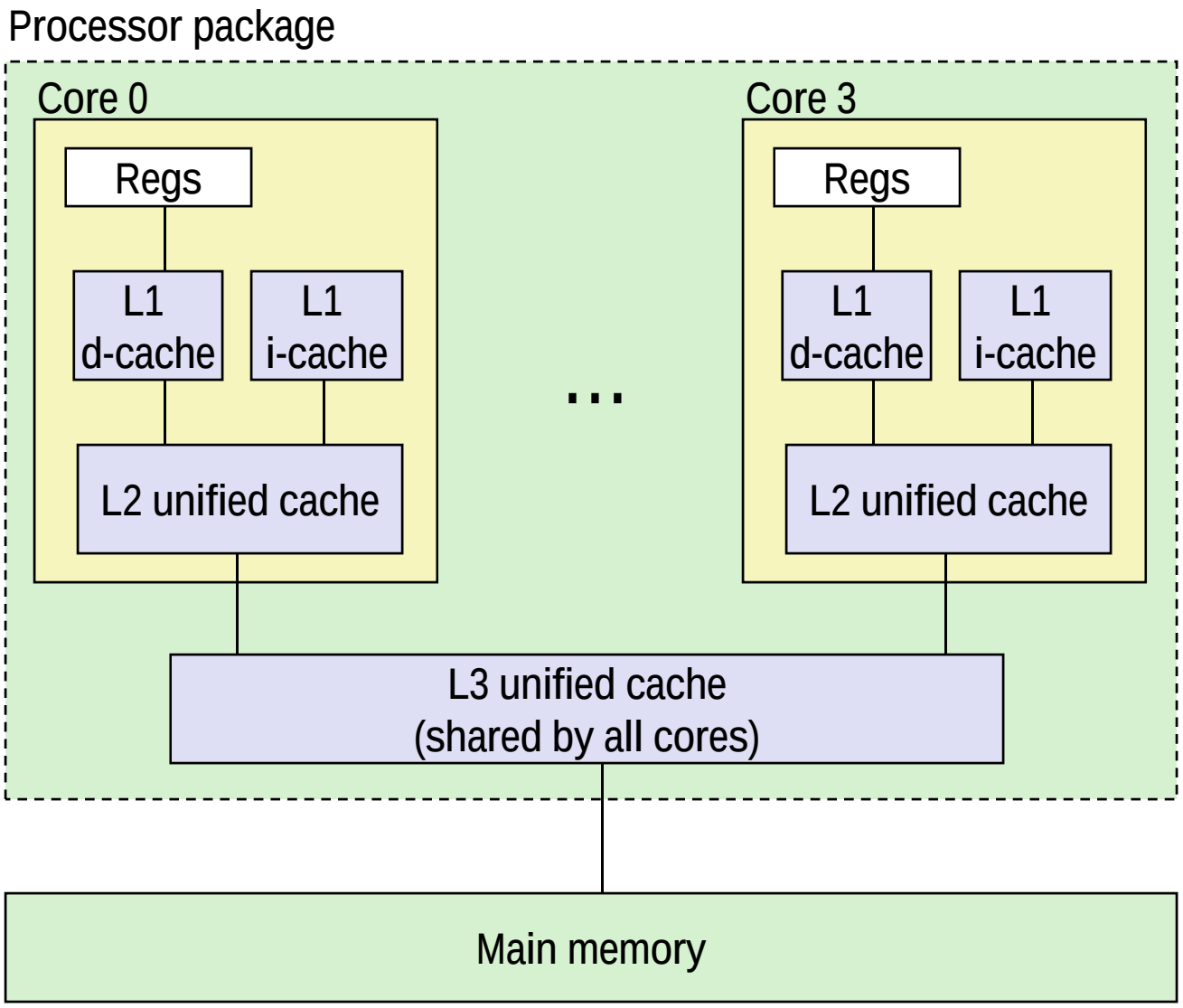
1. Feed data to CPU at the CPU's operating frequency
   - indicates very small cache, perhaps < 100KB *(or even 10KB!)*
2. Maximize hit rate
   - indicates a very large cache, perhaps > 10MB

Problems

1. Small cache may have
   - a hit rate < 90%
   - a miss penalty of 50-100 cycles or more
2. Large cache may have
   - a cycle time 10x slower than that of the CPU

Note: recall canonical performance equation – this is a classic tradeoff between operating frequency and CPI.

Intel Core i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

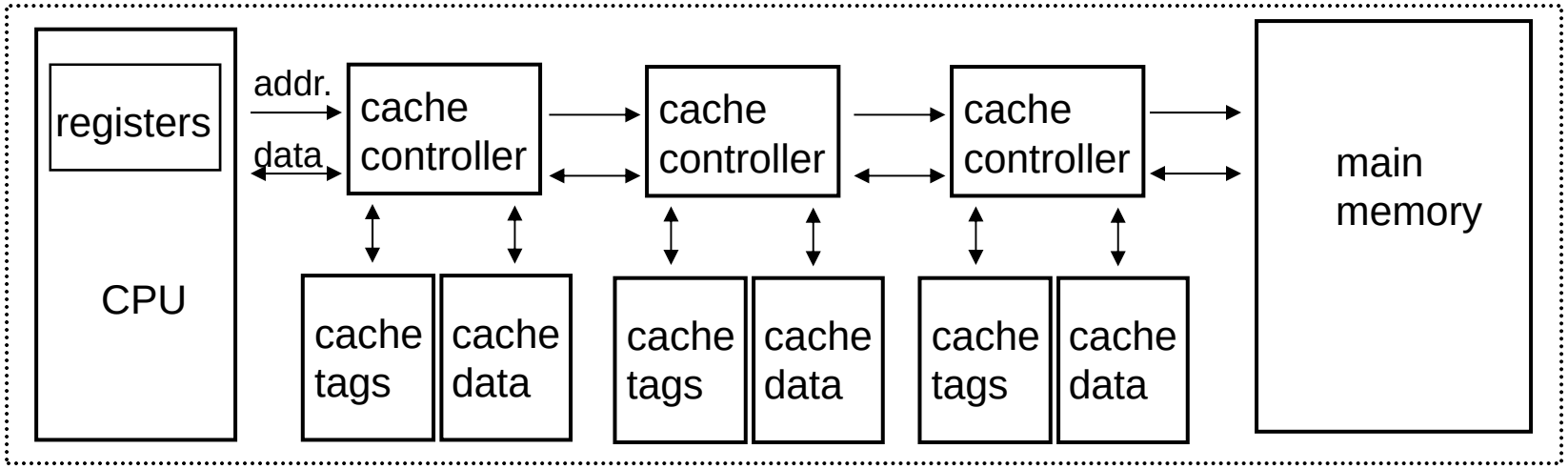
L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches.

Multilevel Cache Implementation

- Cache controllers generally only talk with cache controllers one level away
- Unit of interaction between levels remains the block, though it sometimes varies in size from level to level.



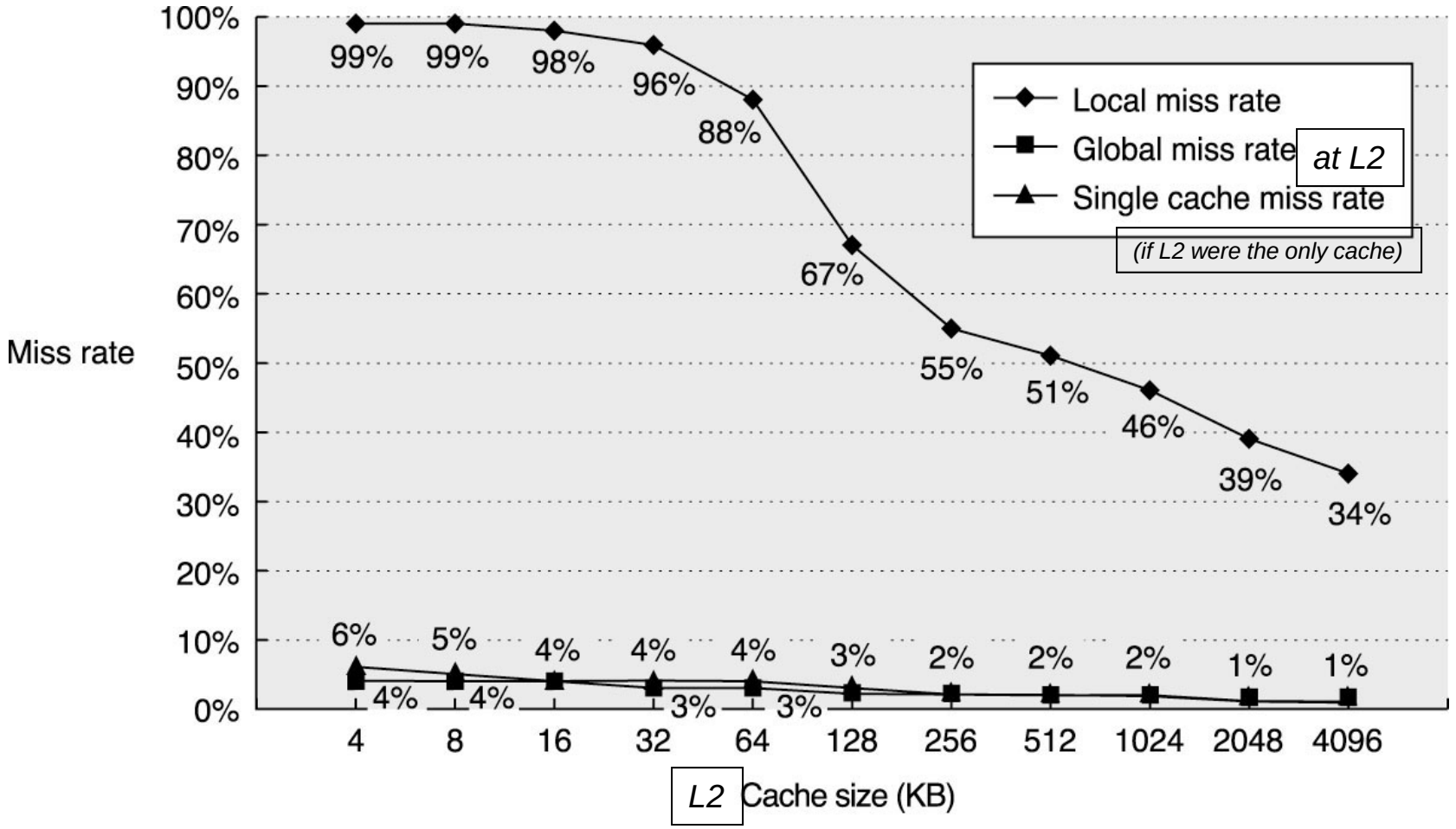
Tag ⇔ Info that uniquely characterizes a ***block*** of data, usually derived from the address of the block

Block ⇔ The unit of interaction (the chunk of data that gets transferred) between levels of a memory hierarchy

Parameters:
L1 : split instruction/data, 64KB each, 2-way set associative, LRU
L2 : unified, 2-way set associative, LRU

Notes:
→ single cache miss rate for L2 ~=
global miss rate for large L2 cache
→ high L2 miss rate still leads to excellent overall performance

Miss rates versus cache size



Review → Sources of Cache Misses

- **Compulsory misses** → aka ***cold start misses***
  - First access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If running billions of instruction, compulsory misses are insignificant

- **Capacity misses**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.
  - A replaced block is later accessed again

- **Conflict misses** → aka ***collision misses***
  - Multiple memory locations mapped to the same cache location
    - Due to competition for entries in a set
    - Would not occur in a fully associative cache of the same total size
  - Most caches limit blocks at level k+1 to a small subset (sometimes a singleton) of the block positions at level k.
    - E.g. Block i at level k+1 must be placed in block (i mod 4) at level k.
  - Conflict misses occur when the level k cache is large enough, but multiple data objects all map to the same level k block.
    - E.g. Referencing blocks 0, 8, 0, 8, 0, 8, ... would miss every time.

- **Coherence misses** → aka ***invalidation misses***
  - ***Other*** process (e.g., I/O or another core) updates memory so cache no longer valid
  - Solutions → later in the course!

Summary: Cache Design for Performance

Methods generally work by affecting some combination of

- Hit Time
- Miss Rate
- Miss Penalty

Technique	Miss Penalty	Miss Rate	Hit Time	Hardware Complexity	Comment
Larger Cache		+	-	1	Widely used, especially for L2 and L3 caches. Decreases capacity misses.
Larger Block Size	-	+		0	Trivial. Decreases compulsory misses.
Higher Associativity		+	-	1	Widely used. Decreases conflict misses.
Multilevel Caches	+			2	Costly hardware; harder if block size L1 ~= L2; widely used
Small and Simple Cache		-	+	0	Trivial; widely used

3-Level Cache Organization

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU replacement, hit time 4 cycles L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU replacement, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU replacement, write-back/allocate, hit time 11 cycles	512KB, 64-byte blocks, 16-way, approx LRU replacement, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, replacement n/a, write-back/allocate, hit time 30-40 cycles	2MB, 64-byte blocks, 32-way, replace block shared by fewest cores, write-back/allocate, hit time 32 cycles

n/a: data not available