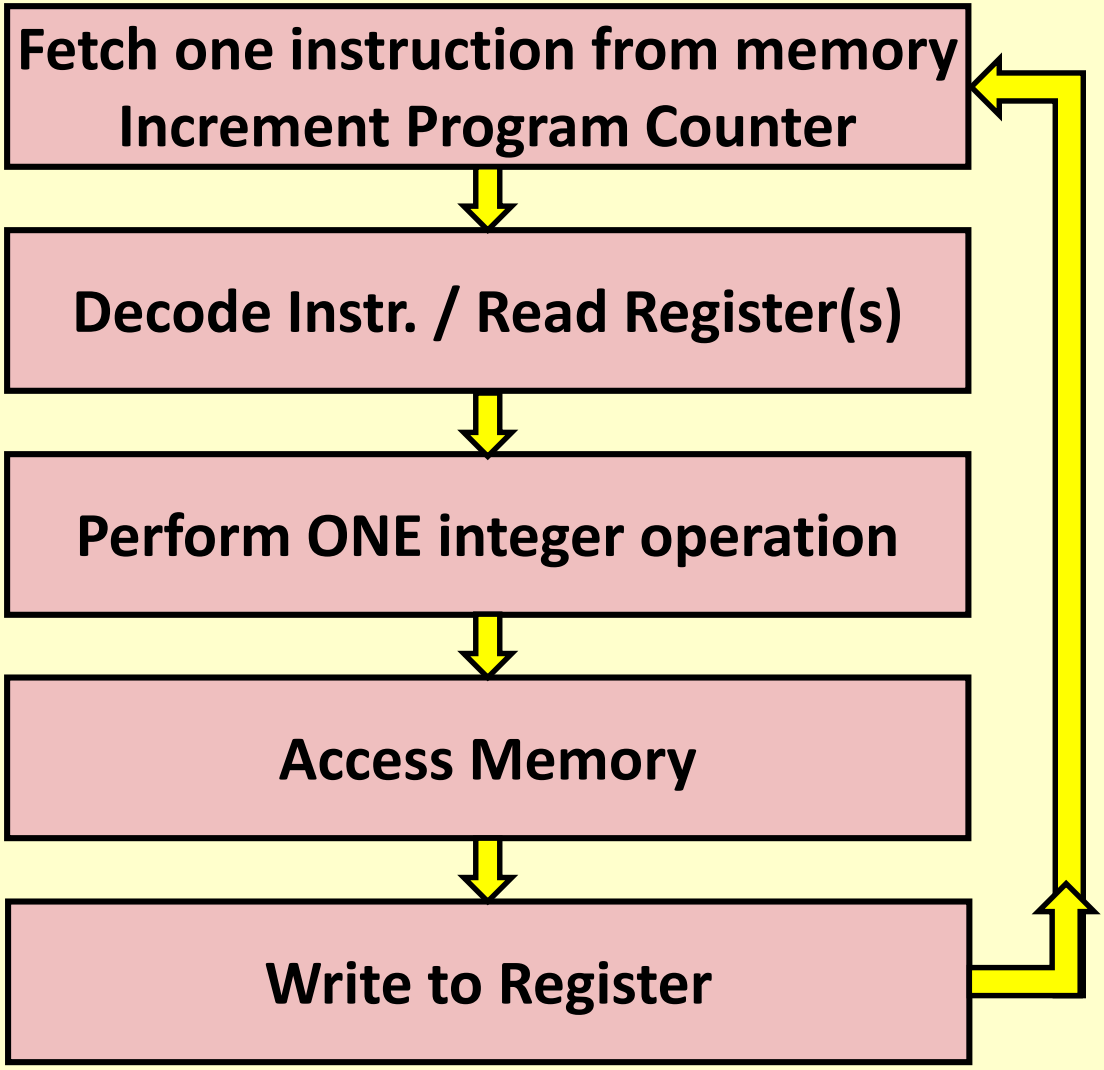
The Memory Hierarchy

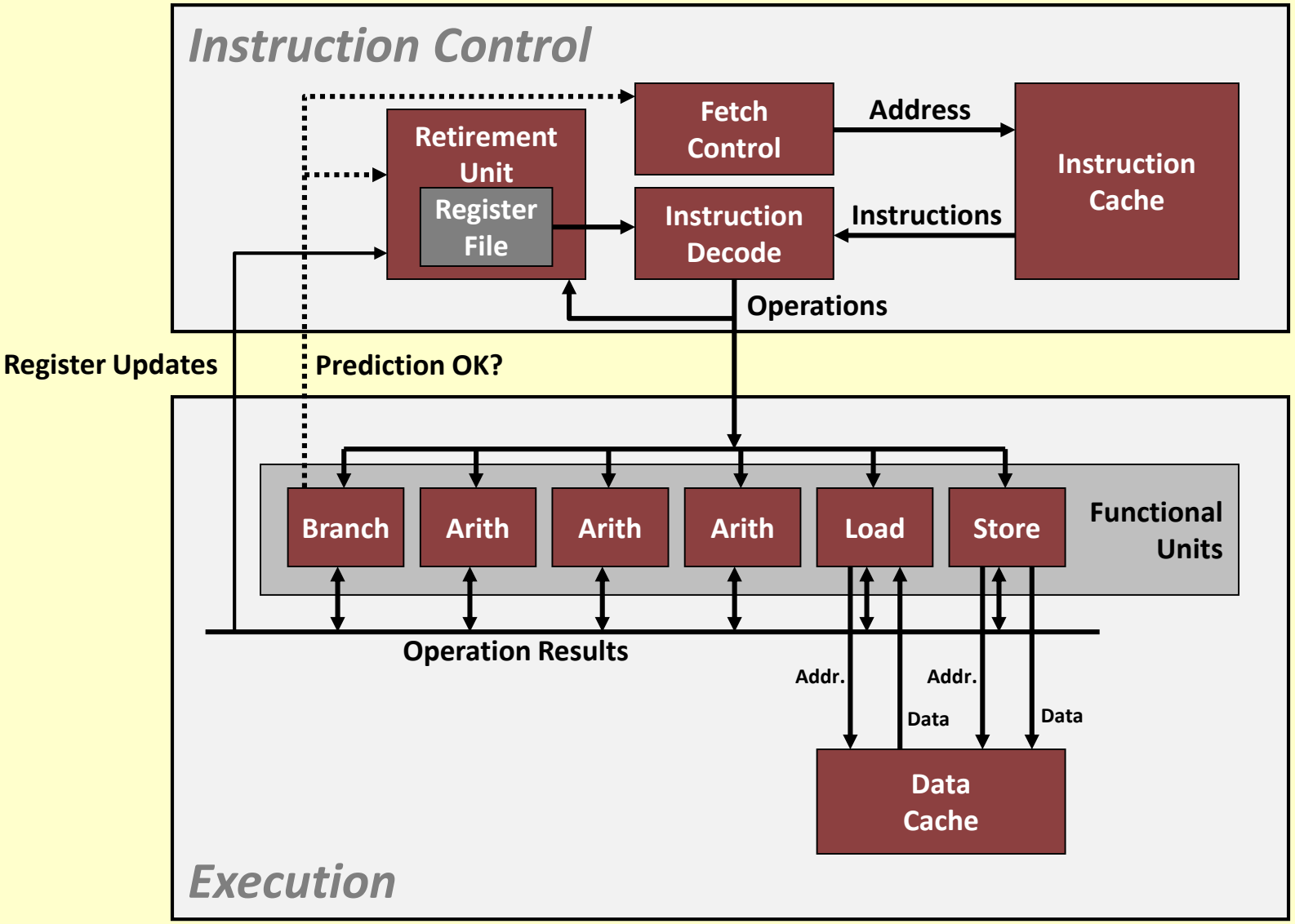
Professor Hugh C. Lauer
CS-2011, Machine Organization and Assembly Language

Execution Model for Modern Computers

A highly idealized view



Modern Processor Design



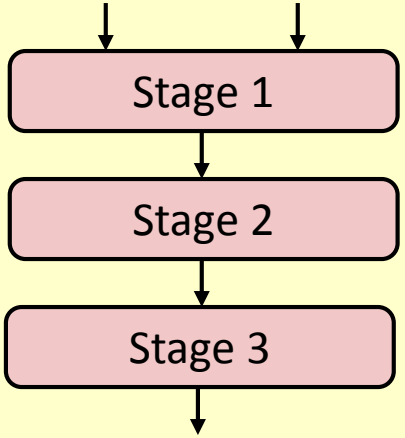
Superscalar Processor

- **Definition:** A superscalar processor can issue and execute *multiple instructions in one cycle*. The instructions are retrieved from a sequential instruction stream and are usually scheduled dynamically.

- Benefit: without programming effort, superscalar processor can take advantage of the *instruction level parallelism* that most programs have

- Most modern processors are superscalar.
- Intel: since Pentium (1993)

Pipelined Functional Units




```
long mult_eg(long a, long b, long c) {
    long p1 = a*b;
    long p2 = a*c;
    long p3 = p1 * p2;
    return p3;
}
```

	Time						
	1	2	3	4	5	6	7
Stage 1	a*b	a*c			p1*p2		
Stage 2		a*b	a*c			p1*p2	
Stage 3			a*b	a*c			p1*p2

- Divide computation into stages
- Pass partial computations from stage to stage
- Stage i can start on new computation once values passed to i+1
- E.g., complete 3 multiplications in 7 cycles, even though each requires 3 cycles

More to say about this later in course!

Much more in CS-4515, Computer Architecture
Spring 2019

Problem

- **`addq %rdx, %rax`**
  - Requires *one* cycle (register to register)
- **`addq 8(%rsp), %rax`**
  - Requires *many* cycles
  - Depending upon type of memory!
- **On-chip**
  - 2–10 cycles to local SRAM
- **Off-chip**
  - 100+ cycles to DRAM

Problem (continued)

- **All that compute power is wasted without quick access to memory**

- **Large on-chip memories impractical**
  - On-chip DRAMs technologically challenging

The Processor-Memory Gap

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

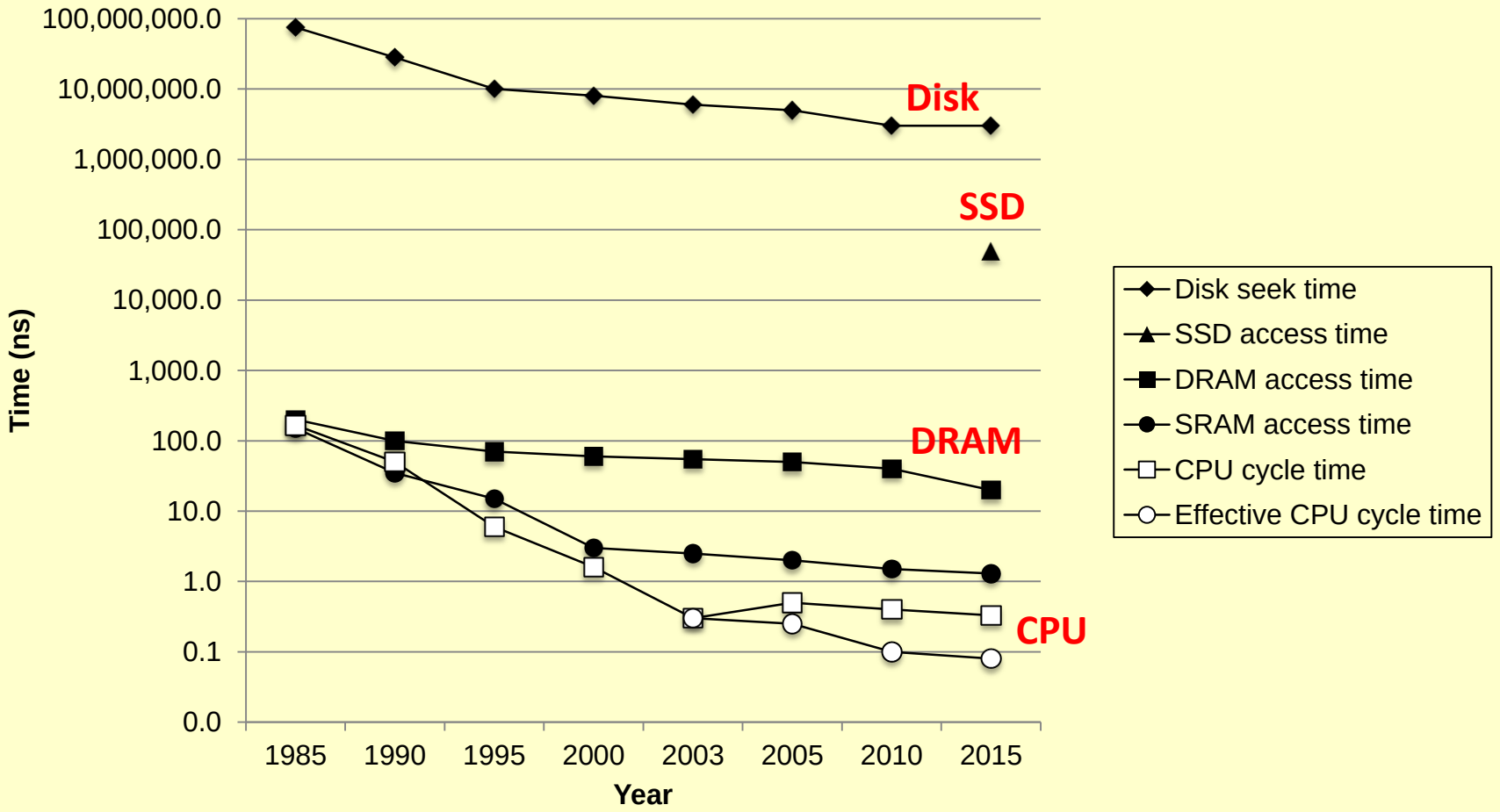


Fig. 6.16

Solution

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

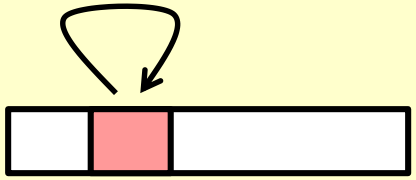
Exploit locality in the form of caches

Today

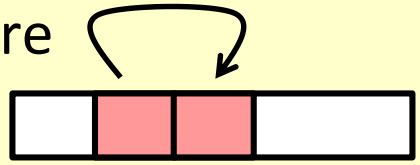
- Storage technologies and trends
- **Locality of reference**
- Caching in the memory hierarchy

Locality

- **Principle of Locality: Programs tend to use data and instructions with addresses near those they have used recently**



- **Temporal locality:**
  - Recently referenced items are likely to be referenced again in the near future



- **Spatial locality:**
  - Items with nearby addresses tend to be referenced close together in time

Locality Example

```
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**

Qualitative Estimates of Locality

- **Claim:** Being able to look at code and get a qualitative sense of its locality is a key skill for a professional programmer.

- **Question:** Does this function have good locality with respect to array a?

```
int sum_array_rows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

I.e., hold row number constant while looping thru columns

Locality Example

- **Question:** What about this function?

- Does it have good locality with respect to array a?

```
int sum_array_cols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

I.e., hold column number constant while looping thru rows

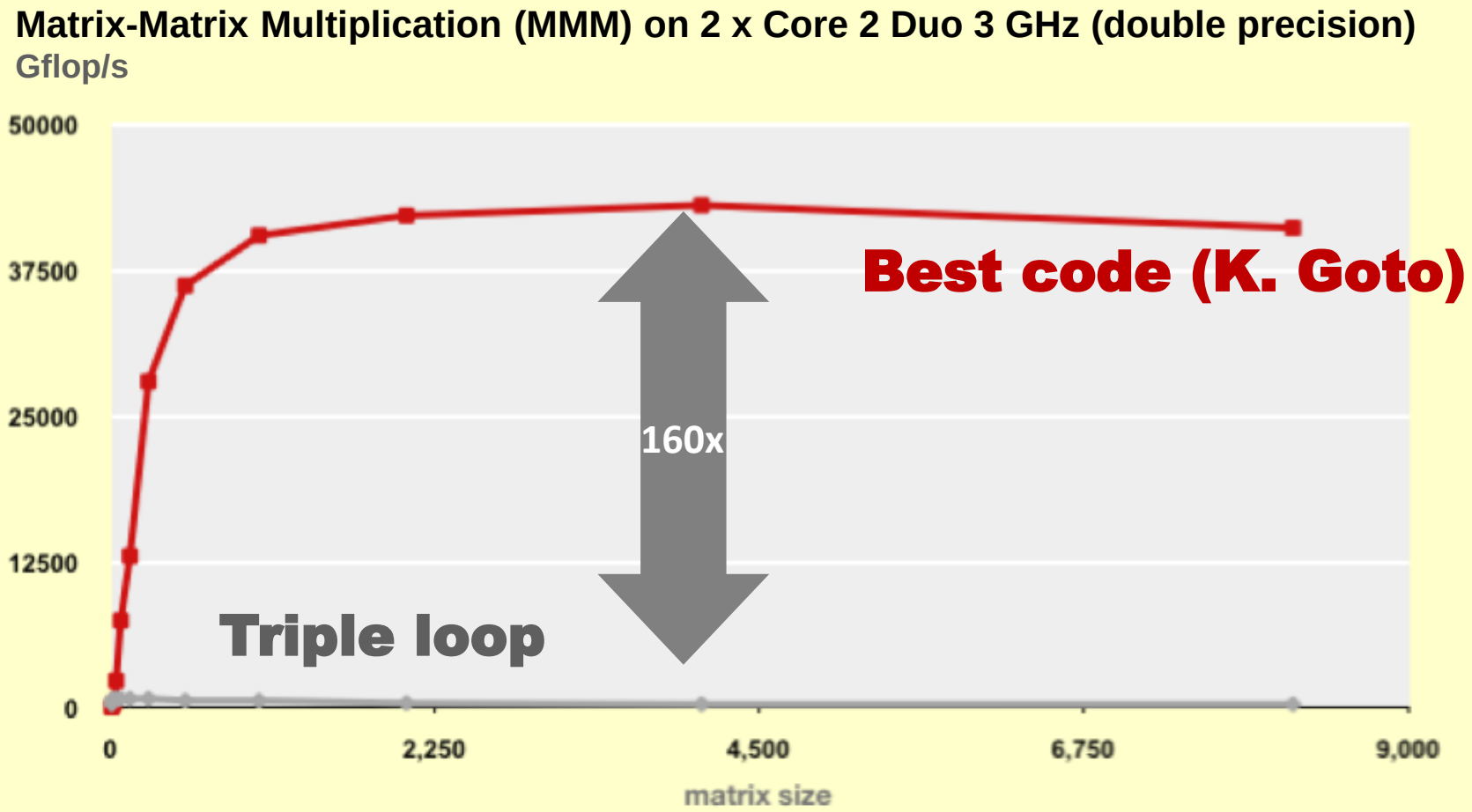
Locality Example

- **Question: Can you permute the loops so that the function scans the 3-d array a with a stride-1 reference pattern (and thus has good spatial locality)?**

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum;
}
```

Example Matrix Multiplication



- Standard desktop computer, vendor compiler, using optimization flags
- Both implementations have exactly the same operations count ($2n^3$)
- What is going on?

Locality Example

- **Question: Can you permute the loops so that the function scans the 3-d array a with a stride-1 reference pattern (and thus has good spatial locality)?**

```
int sum_array_3d(int a[M][N][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < N; k++)
                sum += a[k][i][j];
    return sum;
}
```

Memory Hierarchies

- **Some fundamental and enduring properties of hardware and software:**
  - Fast storage technologies cost more per byte, have less capacity, and require more power (heat!).
  - Widening gap between processor and main memory speeds.
  - 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.**

Today

- Storage technologies and trends
- Locality of reference
- ~~Caching in the memory hierarchy~~
- Caches and caching principles

Definition:– Cache

- A small fast memory that holds a (frequently accessed) subset of items from a much larger, slower memory

- Reason:–

  To approximate the performance of the fast memory while retaining the size and economic benefits of the larger memory

Basic Idea

- Design a cache system so that …
- … most of the accesses go to the small, fast memory, …
- … and those that don't go to the small fast memory occur with sufficiently low probability that the extra cost of access does not hurt, …
- … at least, not very much

Caches occur *everywhere* in computing

- **Transaction processing**
  - Keep records of today's departures in RAM or local storage while records of future flights are on remote database
- **Program execution**
  - Keep the bytes near the current program counter in on-chip SRAM memory while rest of program is in DRAM
- **File management**
  - Keep disk maps of open files in RAM while retaining maps of *all* files on disk
- **Game design**
  - Keep details of nearby environment in cache of each character
- **…**

In fact, …

Caching is, by far, THE MOST IMPORTANT TOPIC pertaining to the performance of *ANY* hardware or software or distributed system!

Example — Virtual Memory

- **Definition:– Virtual memory is the *illusion* that each running program has its own memory space**
  - Separate from those of all other running programs …
  - … on the same computer or different computers …
  - … belonging to the same user or different users

- **In reality, virtual memories are stored on disk**
  - Yes, really slow disks!
  - RAM is a *cache* of the frequently used "pages" of virtual memory

Properties of all caches

- **A mechanism to recognize when something is already in the cache …**
  - … and use it
  - Called a *Cache Hit*
- **A mechanism to recognize when something needed is *not* in the cache …**
  - … and to fetch it from the underlying large memory into the cache
  - Called a *Cache Miss*
- **A policy for throwing something out of a cache …**
  - … to make space for cache misses
- **A mechanism for updating the underlying memory when cached objects change …**
  - … or (especially) when they are thrown out!

Caches can be layered

Processor designer's view

- **L1 is a *cache* of L2 memory**
  - 32 k-bytes, on-chip, 4 cycle access (Core i7)
- **L2 is a *cache* of L3 memory**
  - 256 k-bytes, on-chip, 11 cycle access
- **L3 is a *cache* of DRAM**
  - 8 megabytes, shared among 4 cores; 30-40 cycle access
- **DRAM is a *cache* of virtual memory**
  - *N* gigabytes, shared among all processes; 100's of cycles

Caches can be layered

Programmer's view

- **L1 is a *cache* of L2 memory**
  - 32 k-bytes, on-chip, 4 cycle access (Core i7)
- **L2 is a *cache* of L3 memory**
  - 256 k-bytes, on-chip, 11 cycle access
- **L3 is a *cache* of DRAM**
  - 8 megabytes, shared among 4 cores; 30-40 cycle access
- **DRAM is a *cache* of virtual memory**
  - *N* gigabytes, shared among all processes; 100's of cycles

Memory Hierarchy — layered caches

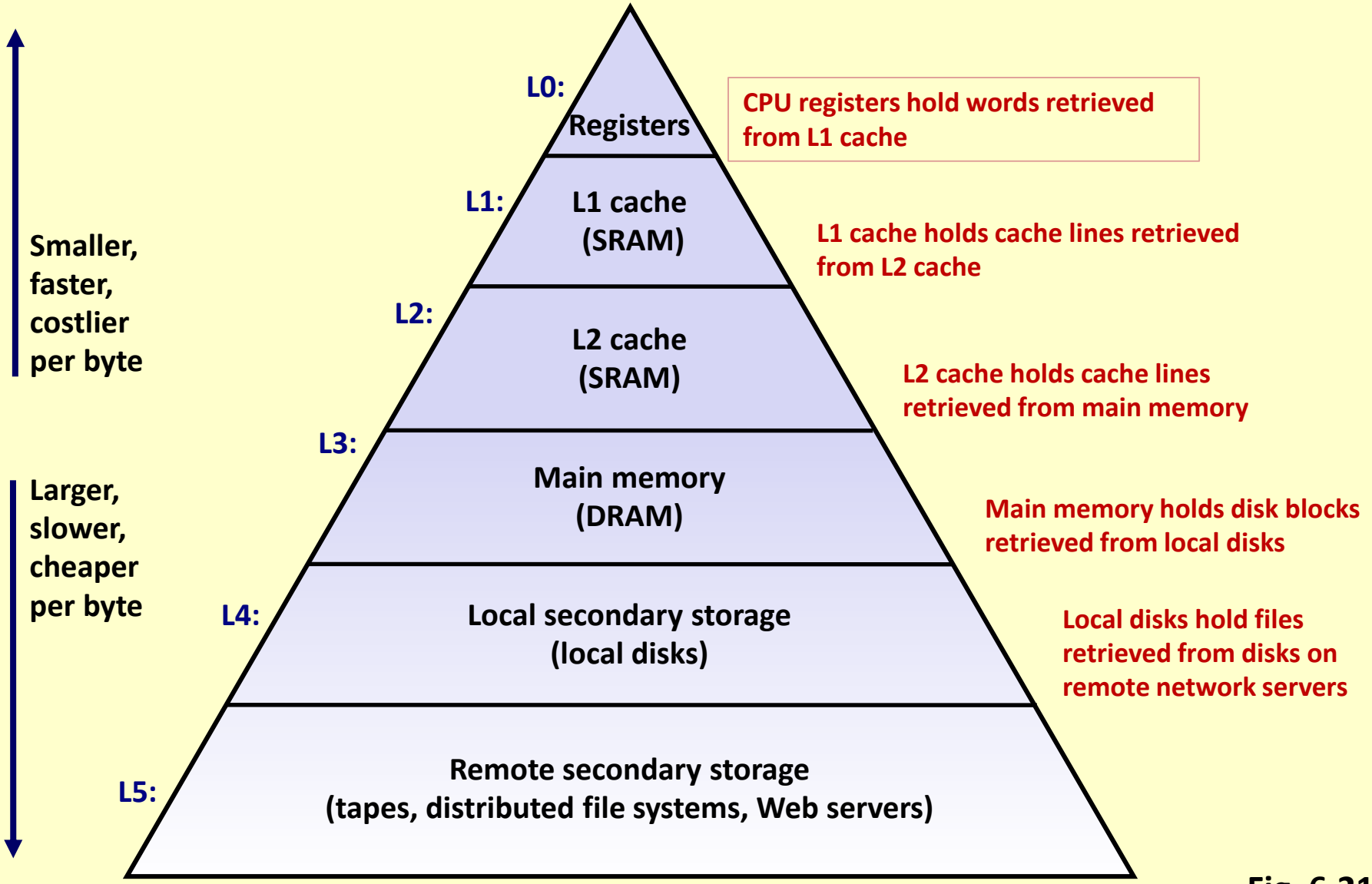


Fig. 6.21

Caches and Memory Hierarchies

- **Fundamental idea of a memory hierarchy**
  - For each *k*, the faster, smaller device at level *k* serves as a *cache* for the larger, slower device at level *k+1*
- **Why do memory hierarchies work?**
  - Because of ***locality***, programs tend to access the data at level *k* more often than they access the data at level *k+1*
  - Thus, the storage at level *k+1* can be slower, and thus larger and cheaper per bit
- ***Big Idea:*** **The memory hierarchy creates illusion of a large pool of storage …**
  - **… as big and (nearly) as cheap as the bottom layer**
  - **… as (nearly) fast as the top layer**

Caching is all about performance …

… and probabilities

Cache Performance

- **Miss Rate**
  - Fraction of memory references not found in cache (misses / accesses) = 1 – hit rate
  - Typical numbers (in percentages):
    - 3-10% for L1
    - can be quite small (e.g., < 1%) for L2, depending on size, etc.
- **Hit Time**
  - Time to deliver a block in the cache to the processor
    - includes time to determine whether the block is in the cache
  - Typical numbers:
    - 1-4 clock cycle for L1
    - 10-20 clock cycles for L2
- **Miss Penalty**
  - Additional time required because of a miss
    - typically 50-200 cycles for main memory (Trend: increasing!)

Cache Performance (continued)

- **Average access time =**
  - Hit time + miss_rate × miss penalty
- **Example**
  - Hit time for L1 cache = 1 cycle
  - Miss penalty for L1 cache = 10 cycles
  - Miss rate = 10%
  - $\Rightarrow$ Average access time = 1 + 0.1 * 10 = 2
- **Example 2**
  - Miss rate = 1%
  - $\Rightarrow$ Average access time = 1 + 0.01 * 10 = 1.1

Think about those numbers

- **Huge difference between a hit and a miss**
  - Could be 100x, if just L1 and main memory

- **Would you believe 99% hits is twice as good as 97%?**
  - Consider:
    cache hit time of 1 cycle
    miss penalty of 100 cycles

- **Average access time:**

  97% hits: 1 cycle + 0.03 * 100 cycles = 4 cycles

  99% hits: 1 cycle + 0.01 * 100 cycles = 2 cycles

- **This is why "miss rate" is used instead of "hit rate"**

Writing Cache Friendly Code

See especially: §6.5

- **Make the common case go fast**
  - Focus on the inner loops of the core functions

- **Minimize the misses in the inner loops**
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

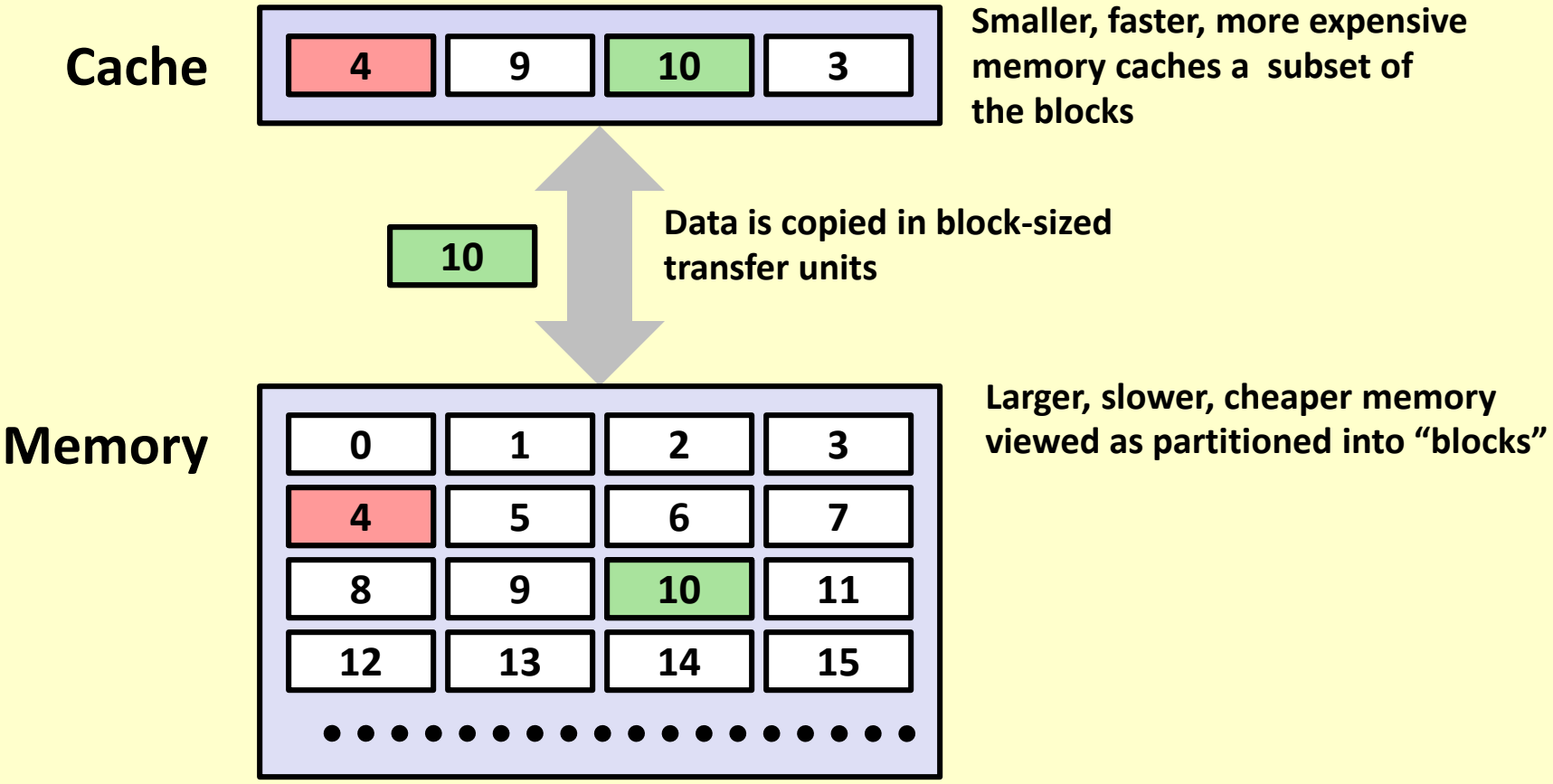
Key idea: Our qualitative notion of locality is quantified through our understanding of cache memories.

Note about hit-miss probabilities

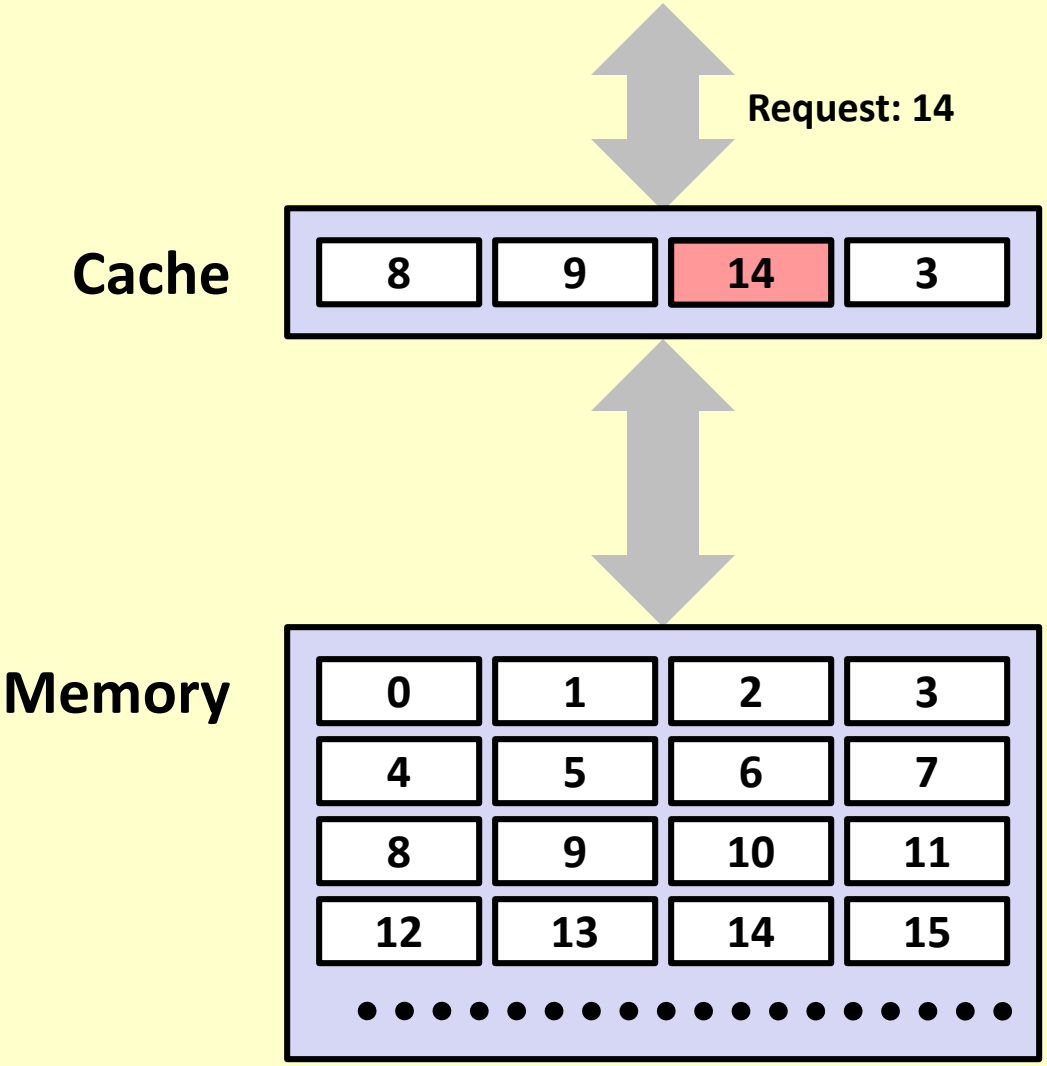
- **No *a priori* methods to predict probabilities of caches hits and misses**
- **I.e., no way to tell in advance how well a particular caching strategy will work**
  - Must determine experimentally!
- ***Benchmarks***
  - Standardized suites of real programs/applications for measuring performance
- **Used in**
  - Hardware design, OS design, Database design, etc.

Questions?

Caches in Microprocessors



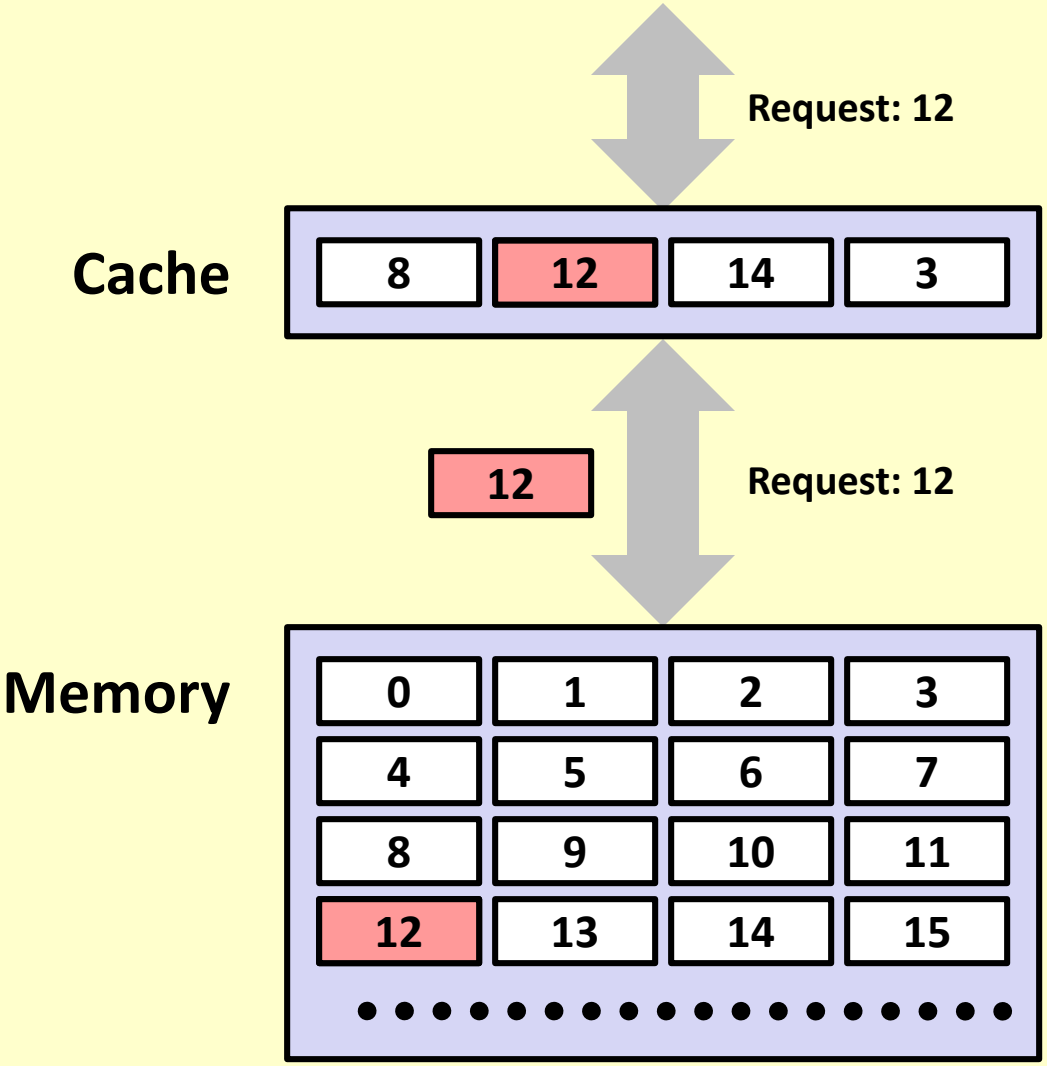
General Cache Concepts: Hit



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)

Types of Cache Misses

- **Cold (compulsory) miss**
  - Cold misses occur because the cache is empty.
- **Conflict miss**
  - 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.
- **Capacity miss**
  - Occurs when the set of active cache blocks (working set) is larger than the cache.

Examples of Caching in the Hierarchy

Cache Type	What is Cached?	Where is it Cached?	Latency (cycles)	Managed By
Registers	4-8 bytes words	Processor 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.

Questions?