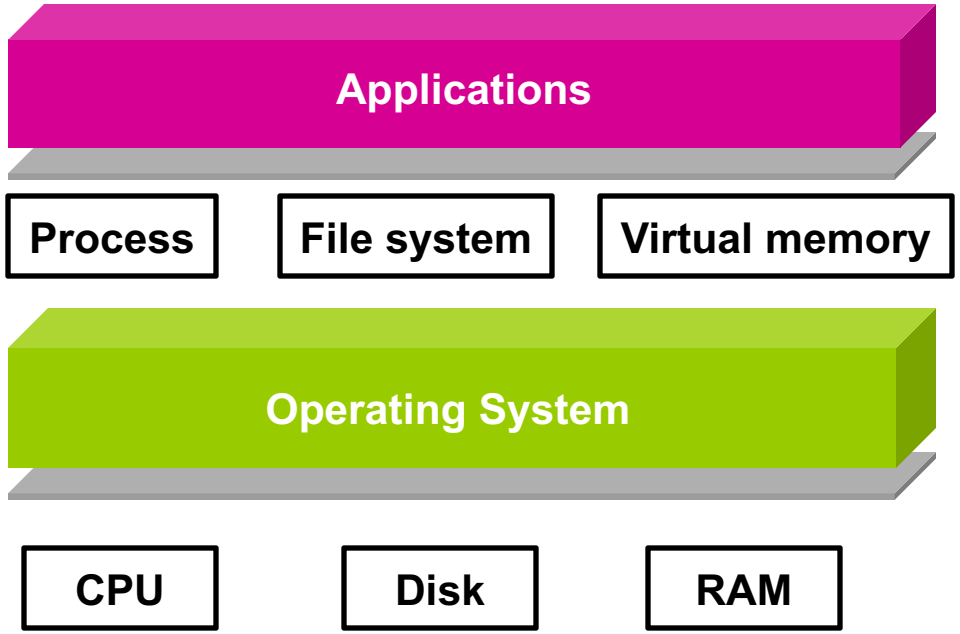
CSE 153
Design of Operating Systems

Winter 2023

Lecture 14/15: Memory Management (1)

Some slides from Dave O'Hallaron

OS Abstractions



Our plan of action

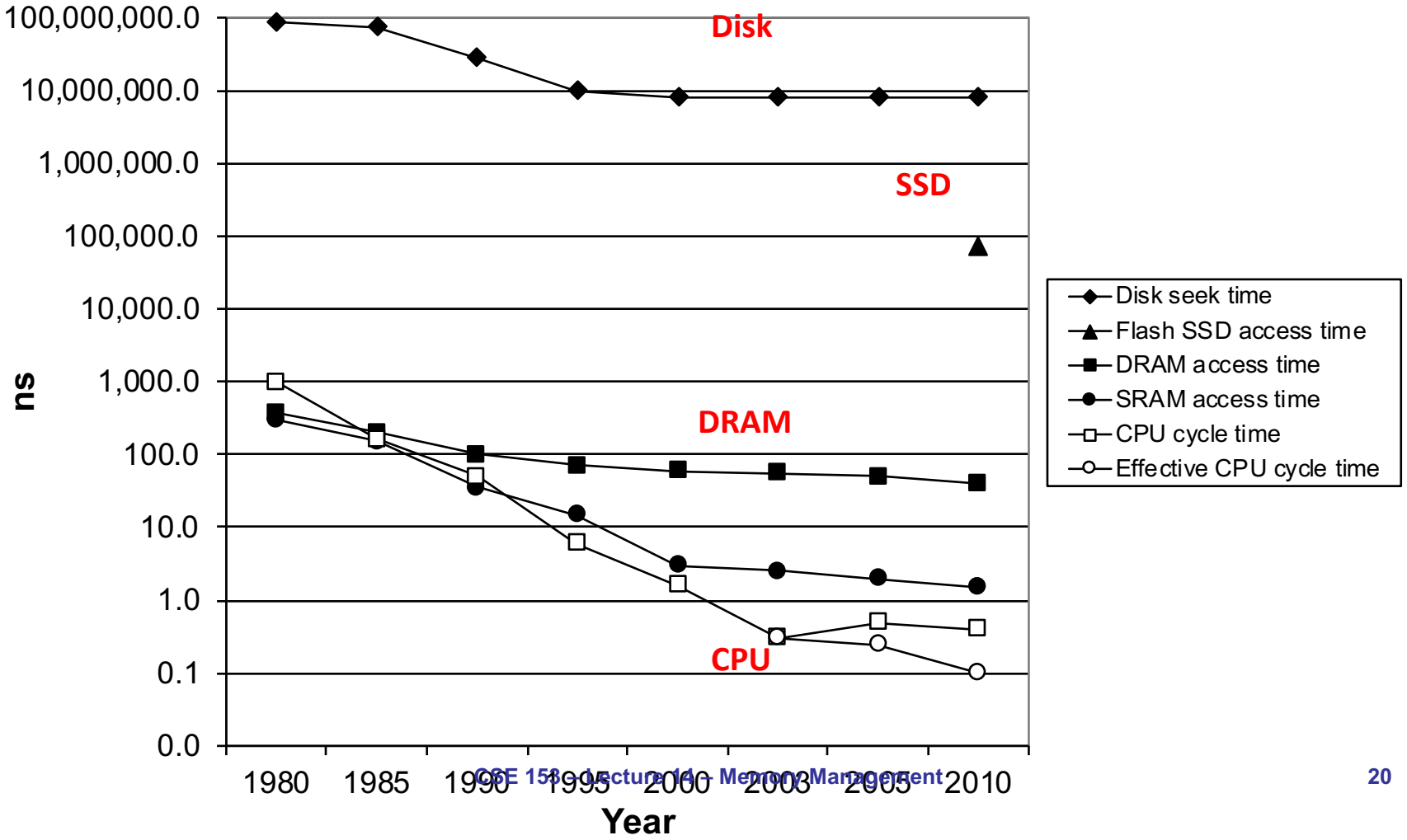
- Memory/storage technologies and trends
  - Memory wall!
- Locality of reference to the rescue
  - Caching in the memory hierarchy
- Abstraction: Address spaces and memory sharing
- Virtual memory
- Today: background and bird's eye view – more details to follow later

SRAM vs DRAM Summary

	Trans. per bit	Access time	Needs refresh?	Needs	EDC?	Cost	Applications
SRAM	4 or 6	1X	No	Maybe	100x	Cache memories	
DRAM	1	10X	Yes	Yes	1X	Main memories, frame buffers	

The CPU-Memory Gap

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



Locality to the Rescue!

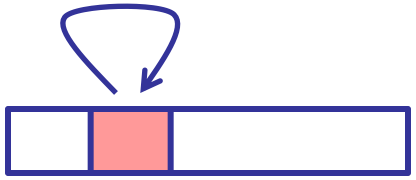
The key to bridging this CPU-Memory gap is a fundamental property of computer programs known as locality

Today

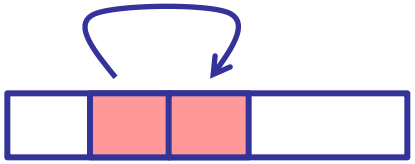
- Storage technologies and trends
- Locality of reference
- Caching in the memory hierarchy
- Virtual memory and memory sharing

Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to 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

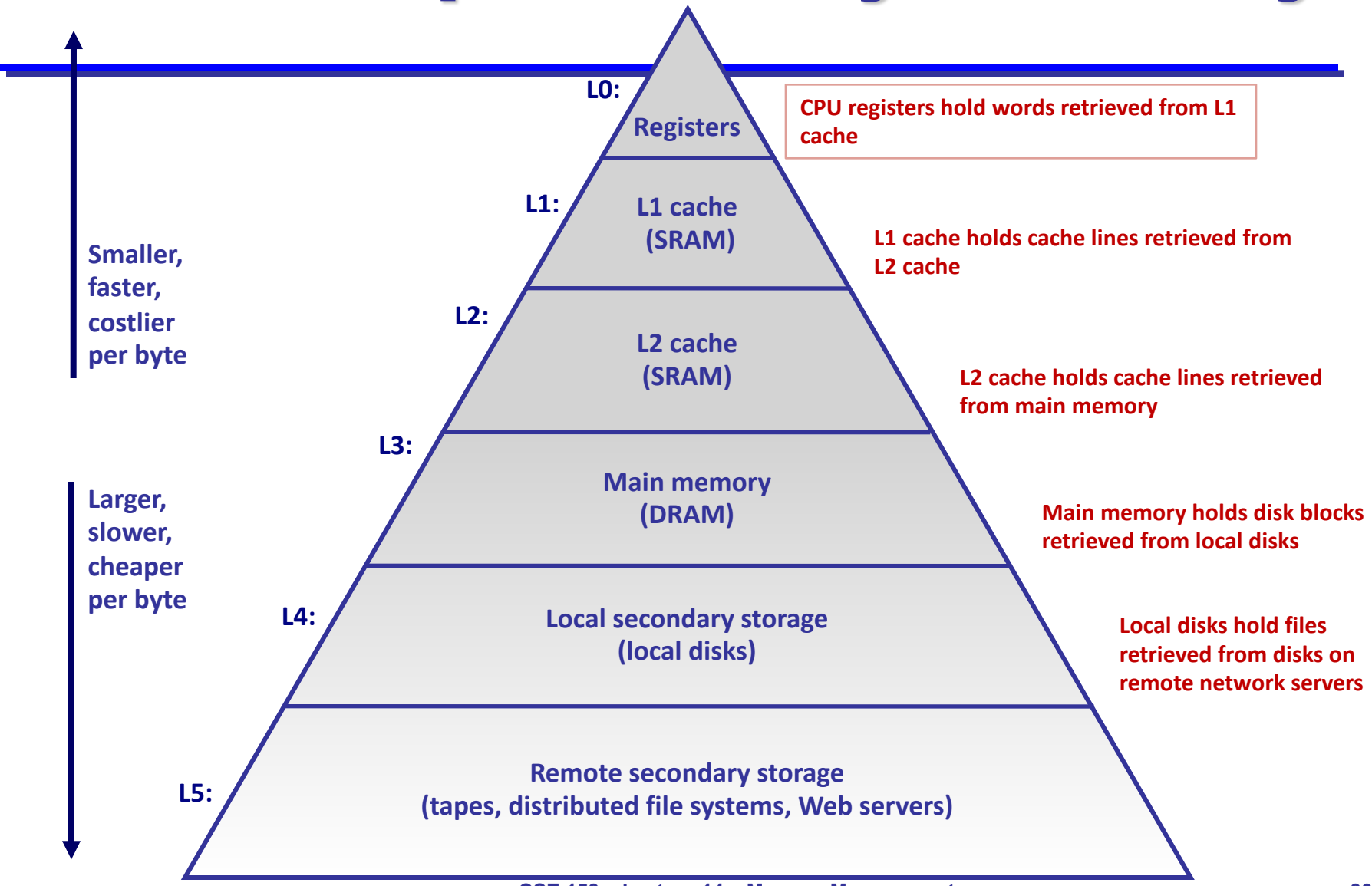
```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

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

Today

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

An Example Memory Hierarchy



Memory hierarchy

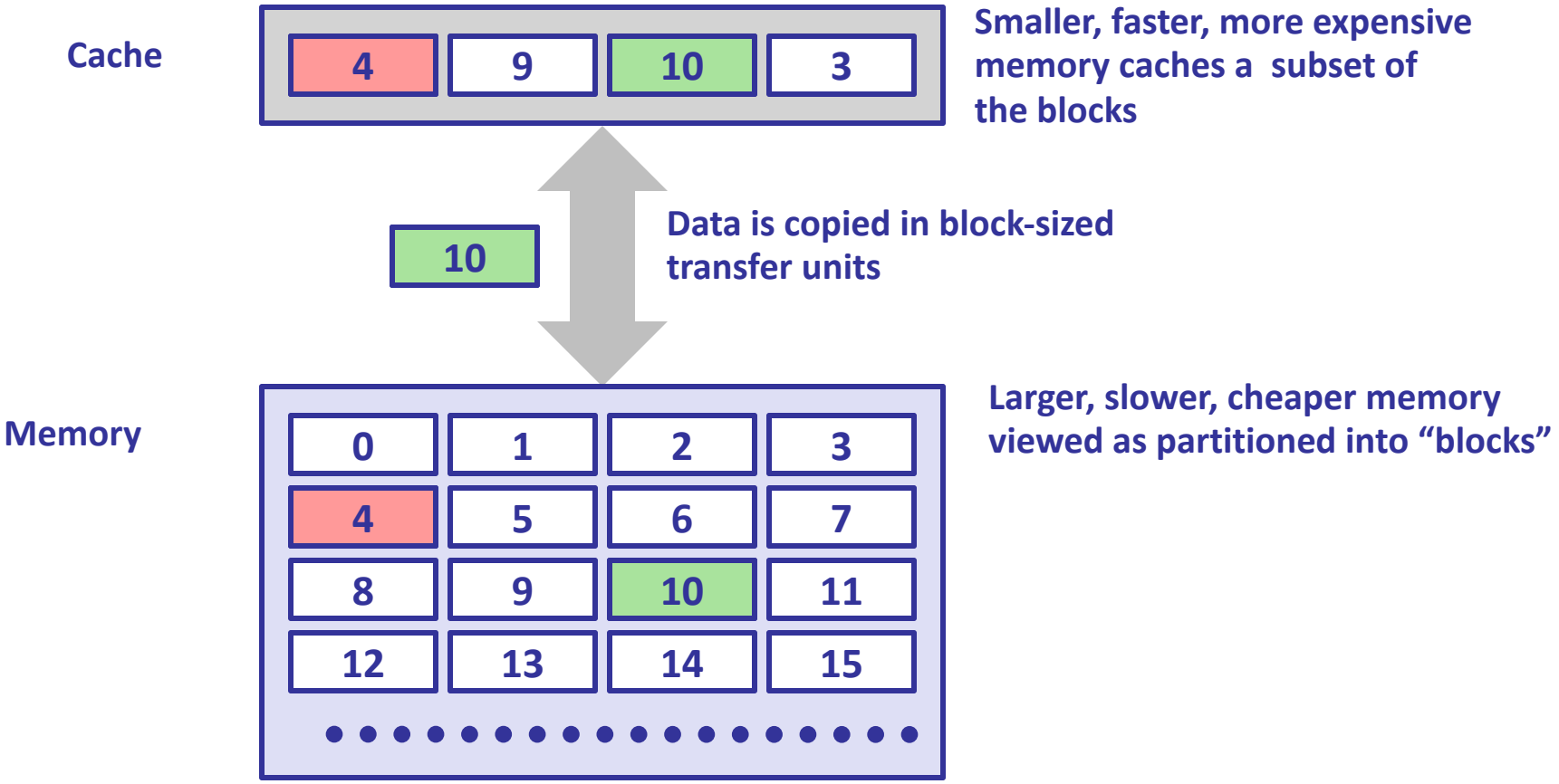
- *Cache:* A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.

- Fundamental idea of a memory hierarchy:
  - For each layer, faster, smaller device caches larger, slower device

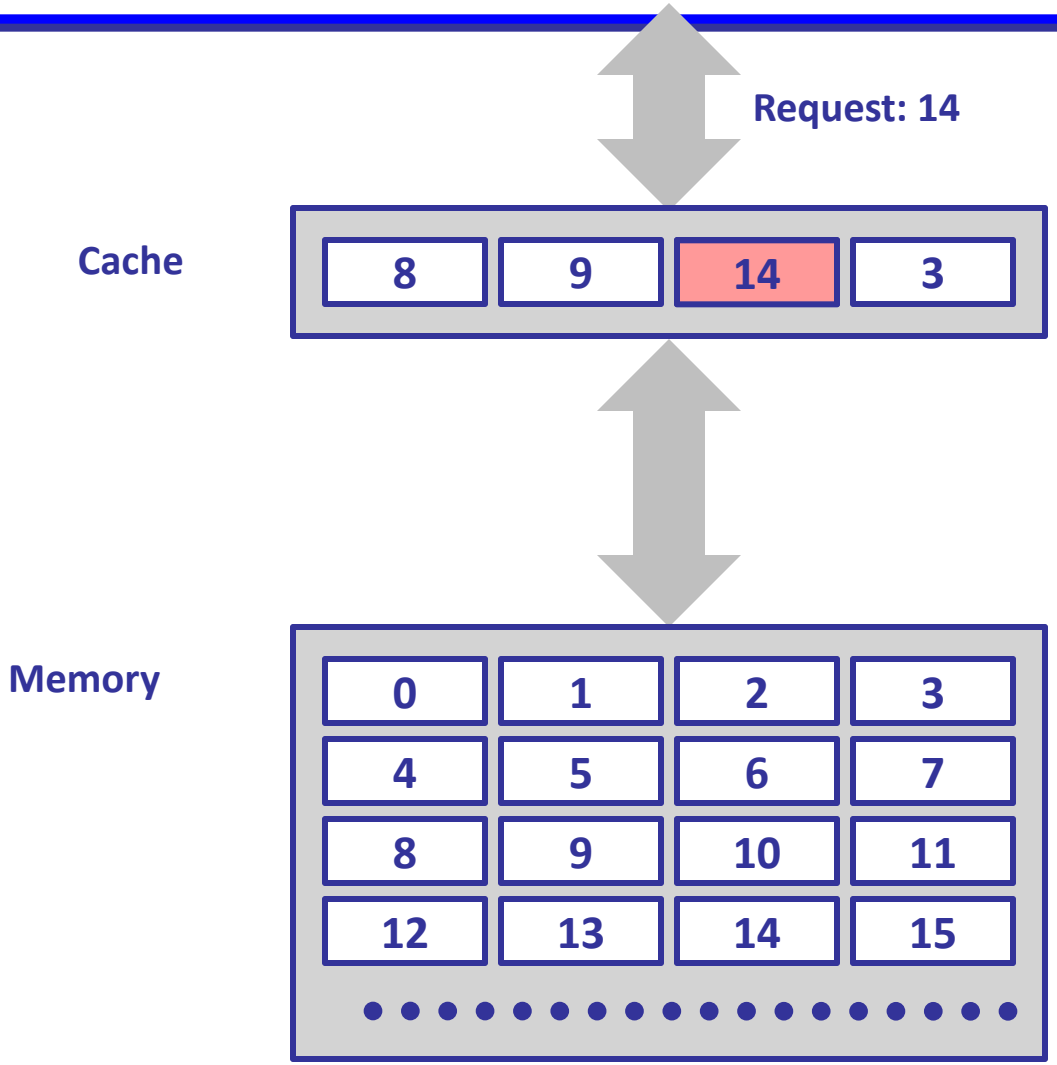
  .
- Why do memory hierarchies work?
  - Because of locality!
    - » Hit fast memory much more frequently even though its smaller
  - Thus, the storage at level k+1 can be slower (but larger and cheaper!)

- *Big Idea:* The memory hierarchy creates a large pool of storage that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

General Cache Concepts



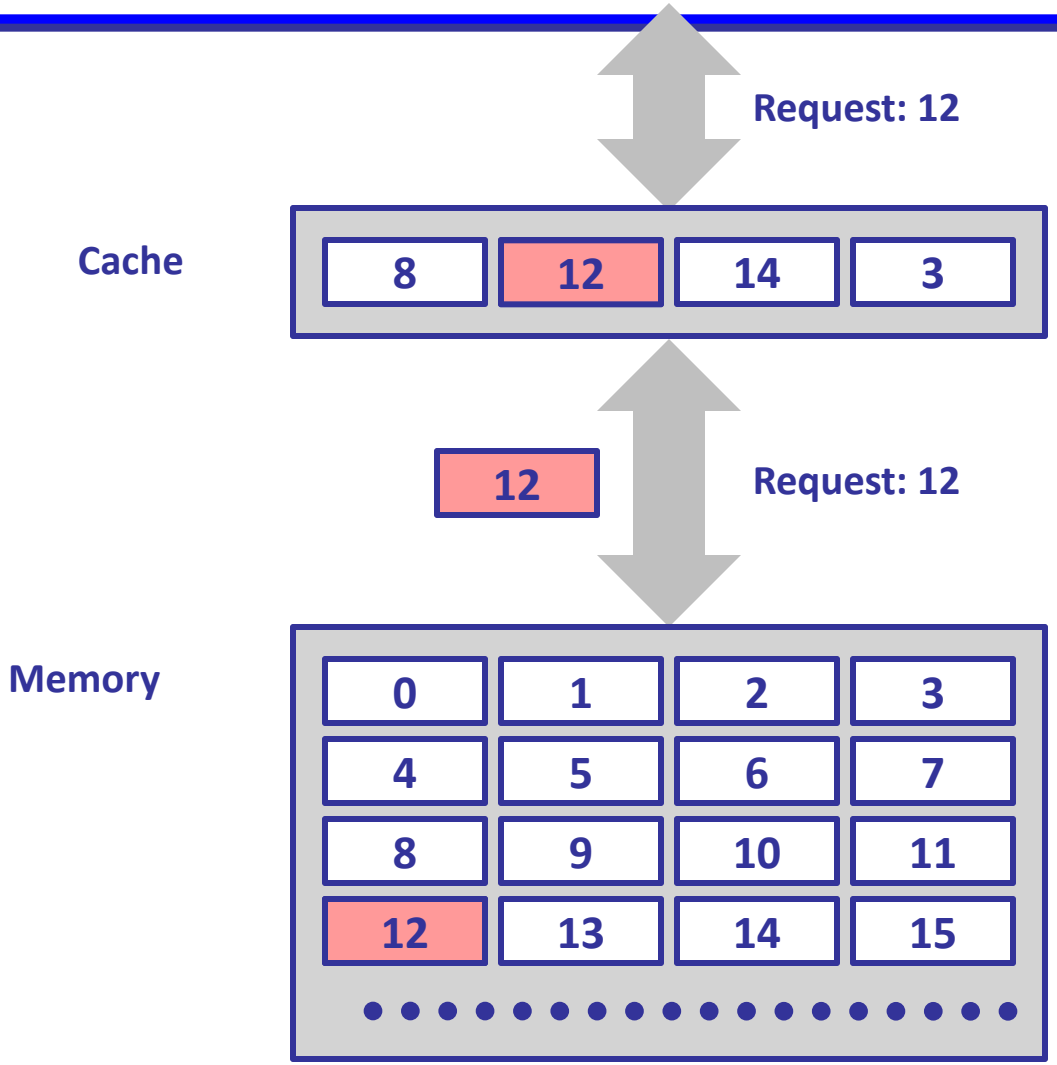
General Cache Concepts: Hit



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)

General Caching Concepts: 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.

Summary so far

- 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.

Sharing Memory

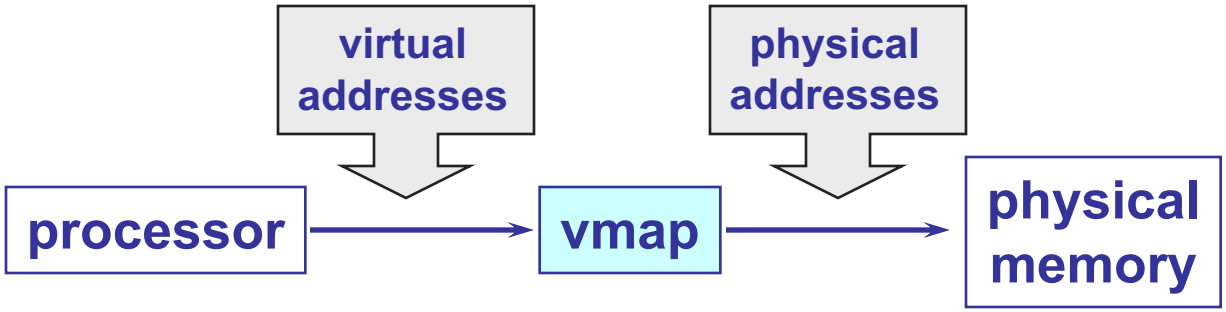
- Rewind to the days of "second-generation" computers
  - Programs use physical addresses directly
  - OS loads job, runs it, unloads it

- Multiprogramming changes all of this
  - Want multiple processes in memory at once
    - » Overlap I/O and CPU of multiple jobs
  - How to share physical memory across multiple processes?
    - » Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it
    - » A program can run on machine with less memory than it "needs"

Virtual Addresses

- To make it easier to manage the memory of processes running in the system, we're going to make them use virtual addresses (logical addresses)
  - Virtual addresses are independent of the actual physical location of the data referenced
  - OS determines location of data in physical memory
- Instructions executed by the CPU issue virtual addresses
  - Virtual addresses are translated by hardware into physical addresses (with help from OS)
  - The set of virtual addresses that can be used by a process comprises its virtual address space

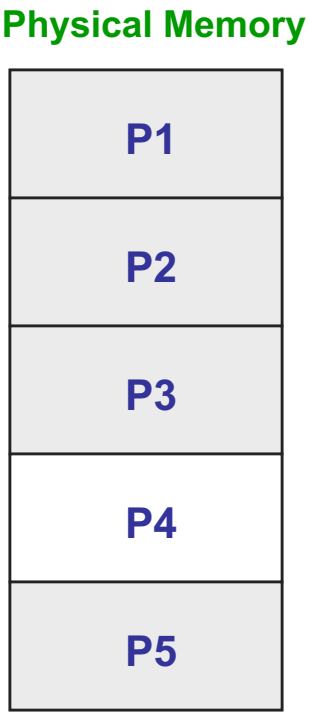
Virtual Addresses



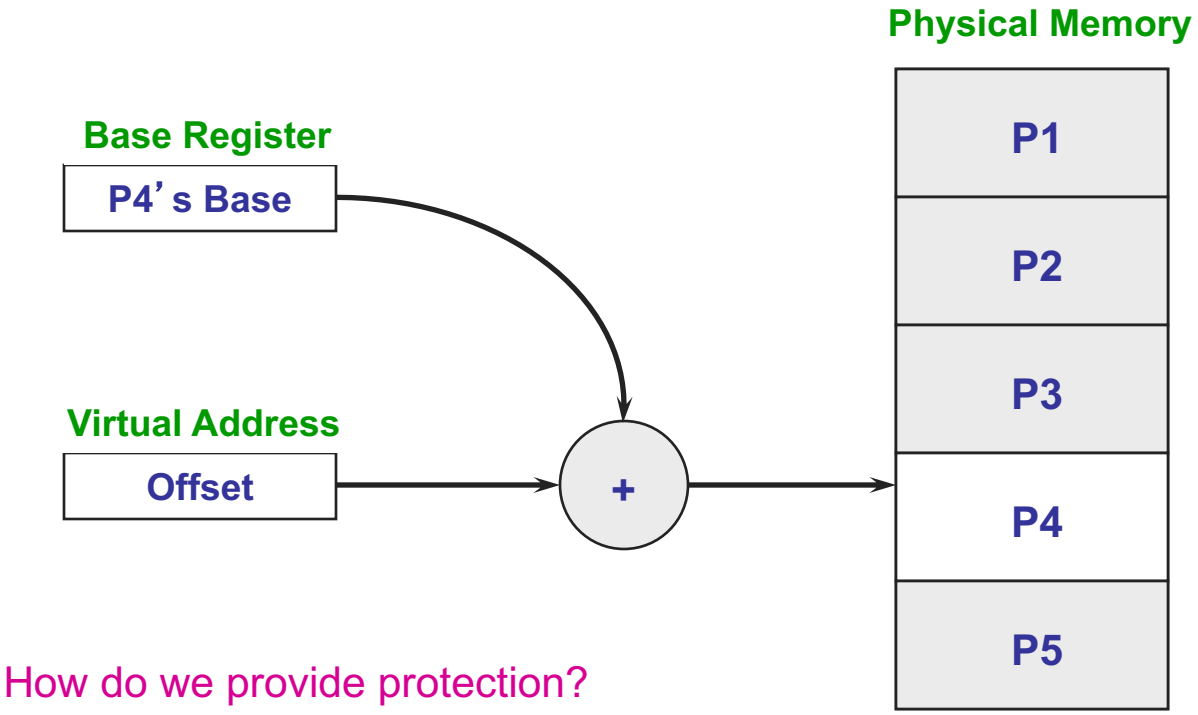
- Many ways to do this translation…
  - Need hardware support and OS management algorithms
- Requirements
  - Need protection – restrict which addresses jobs can use
  - Fast translation – lookups need to be fast
  - Fast change – updating memory hardware on context switch

Fixed Partitions

- Physical memory is broken up into fixed partitions
  - Size of each partition is the same and fixed
  - Hardware requirements: base register
  - Physical address = virtual address + base register
  - Base register loaded by OS when it switches to a process



Fixed Partitions



How do we provide protection?

Fixed Partitions

- Advantages
  - Easy to implement
    - » Need base register
    - » Verify that offset is less than fixed partition size
  - Fast context switch

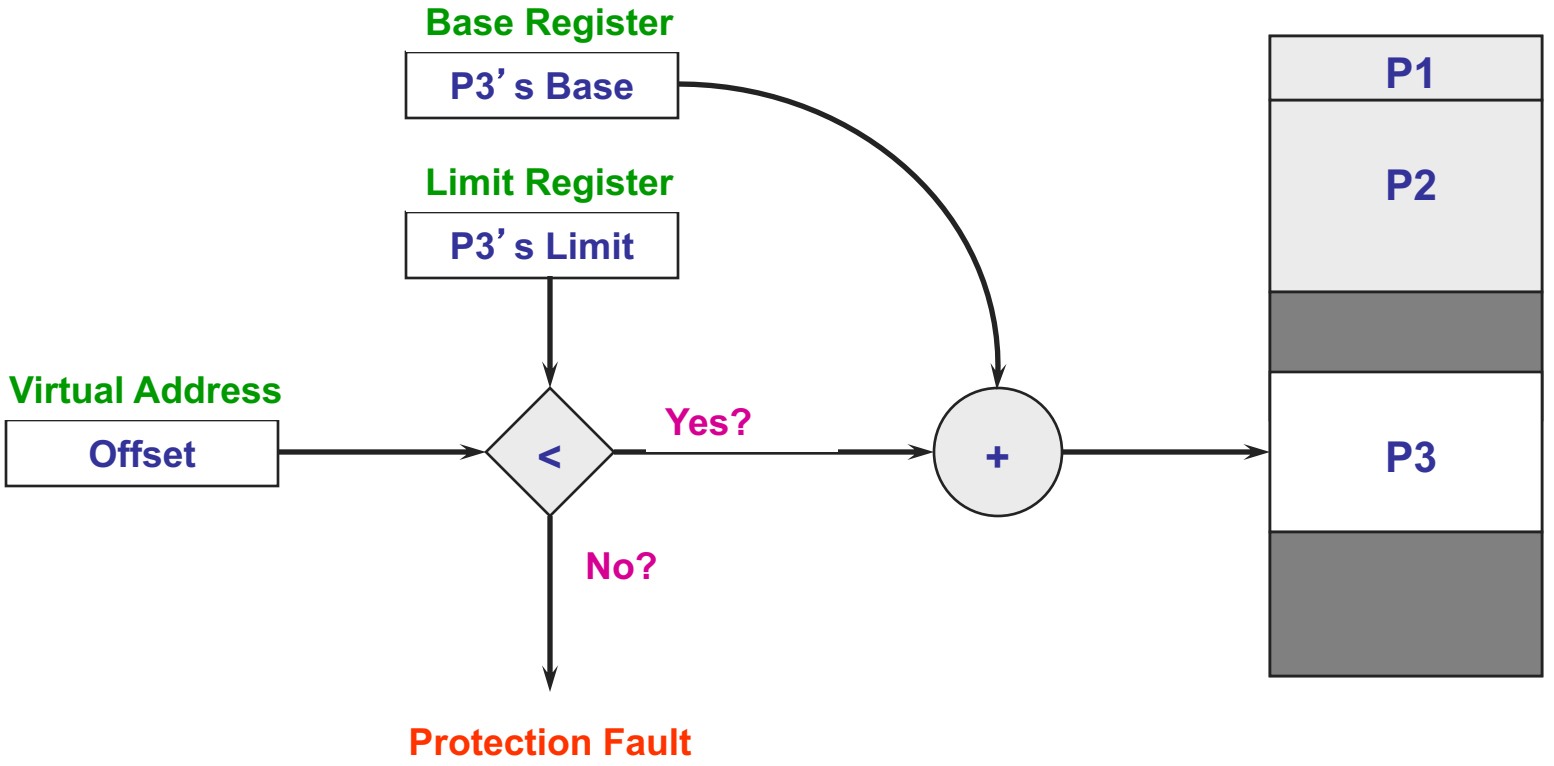
- Problems?
  - Internal fragmentation: memory in a partition not used by a process is not available to other processes
  - Partition size: one size does not fit all (very large processes?)

Variable Partitions

- Natural extension – physical memory is broken up into variable sized partitions
  - Hardware requirements: base register and limit register
  - Physical address = virtual address + base register

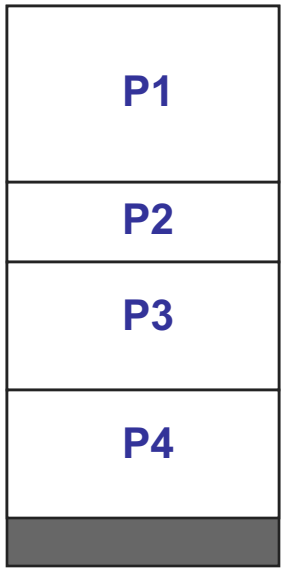
- Why do we need the limit register?
  - Protection: if (virtual address > limit) then fault

Variable Partitions



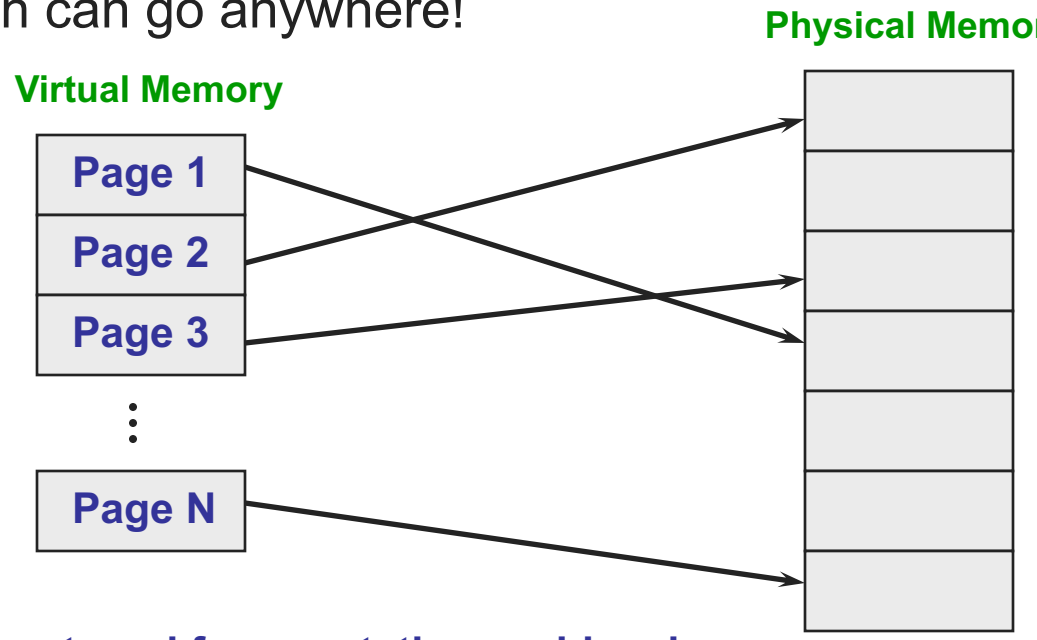
Variable Partitions

- Advantages
  - No internal fragmentation: allocate just enough for process
- Problems?
  - External fragmentation: job loading and unloading produces empty holes scattered throughout memory



Paging

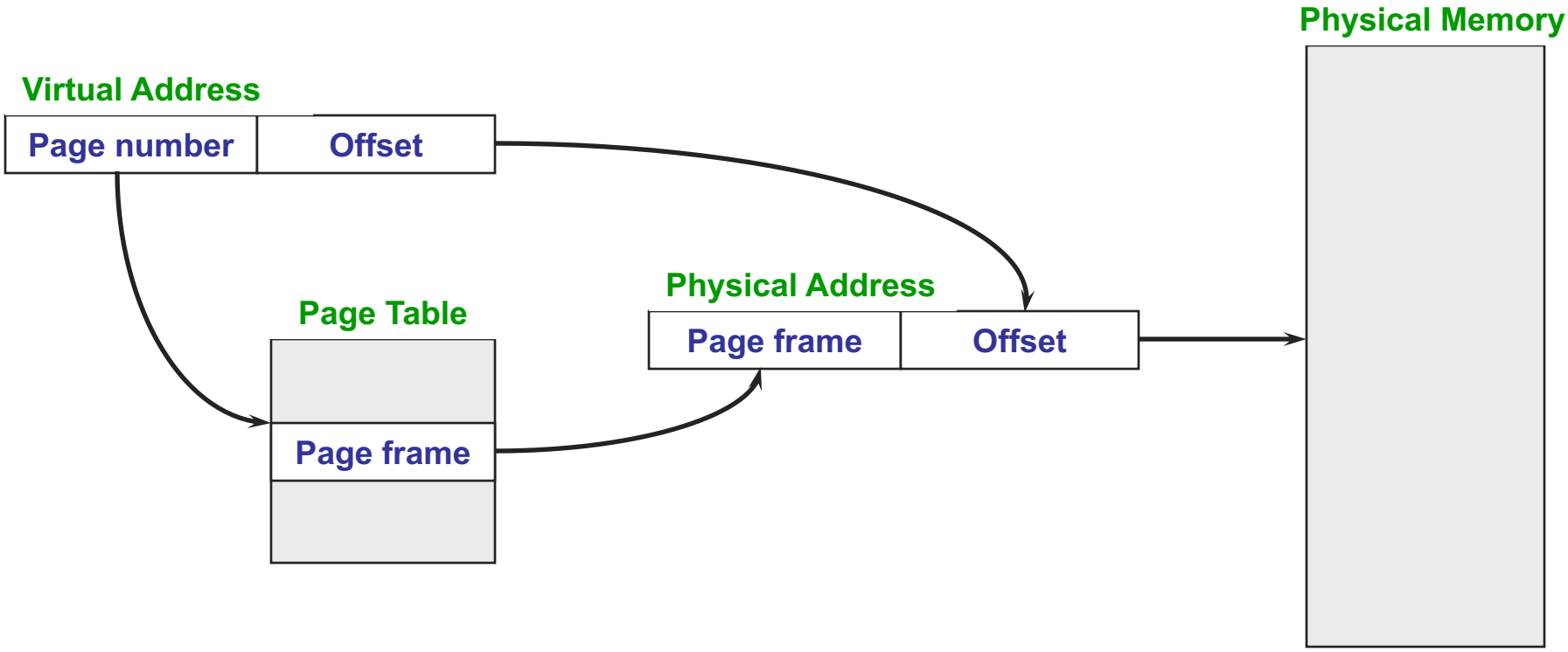
- New Idea: split virtual address space into multiple partitions
  - Each can go anywhere!



Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory

But need to keep track of where things are!

Page Lookups



Paging Advantages

- Easy to allocate memory
  - Memory comes from a free list of fixed size chunks
  - Allocating a page is just removing it from the list
  - External fragmentation not a problem
    - » All pages of the same size
- Simplifies protection
  - All chunks are the same size
  - Like fixed partitions, don't need a limit register
- Simplifies virtual memory – later

Paging Limitations

- Can still have internal fragmentation
  - Process may not use memory in multiples of a page
- Memory reference overhead
  - 2 references per address lookup (page table, then memory)
  - What can we do?
- Memory required to hold page table can be significant
  - Need one PTE per page
  - 32 bit address space w/ 4KB pages = 2^{20} PTEs
  - 4 bytes/PTE = 4MB/page table
  - 25 processes = 100MB just for page tables!
  - What can we do?

Segmentation

- Segmentation: partition memory into logically related units
  - Module, procedure, stack, data, file, etc.
  - Units of memory from user's perspective
- Natural extension of variable-sized partitions
  - Variable-sized partitions = 1 segment/process
  - Segmentation = many segments/process
  - Fixed partition : Paging :: Variable partition : Segmentation
- Hardware support
  - Multiple base/limit pairs, one per segment (segment table)
  - Segments named by #, used to index into table
  - Virtual addresses become <segment #, offset>

Segment Lookups