Basic Memory Management

Outline

- Memory management requirements
- Basic memory management
- Fixed Partitions
- Dynamic Partitions

In an ideal world…

- The ideal world has memory that is
  - Very large
  - Very fast
  - Non-volatile (doesn't go away when power is turned off)
- The real world has memory that is:
  - Very large
  - Very fast
  - Affordable!

  ⇒Pick any two…
- Memory management goal: make the real world look as much like the ideal world as possible.

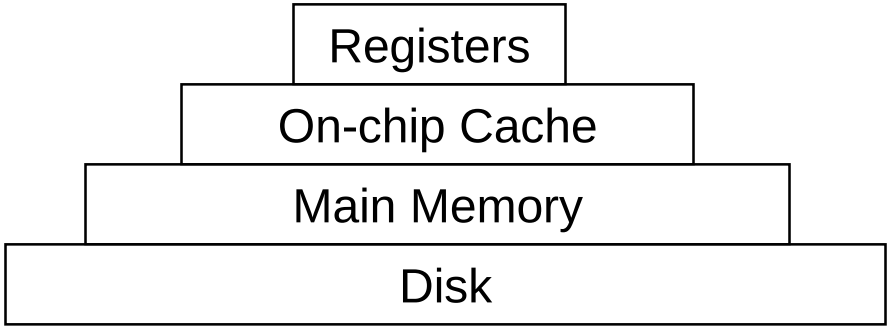
Introduction

- Our machines today have lots of more memory than the IBM 7094 – leading edge machine of the 1960's
- Cost of memory had dropped dramatically
- Bill Gates (former chair of Microsoft) once said "640K should be enough"

Introduction

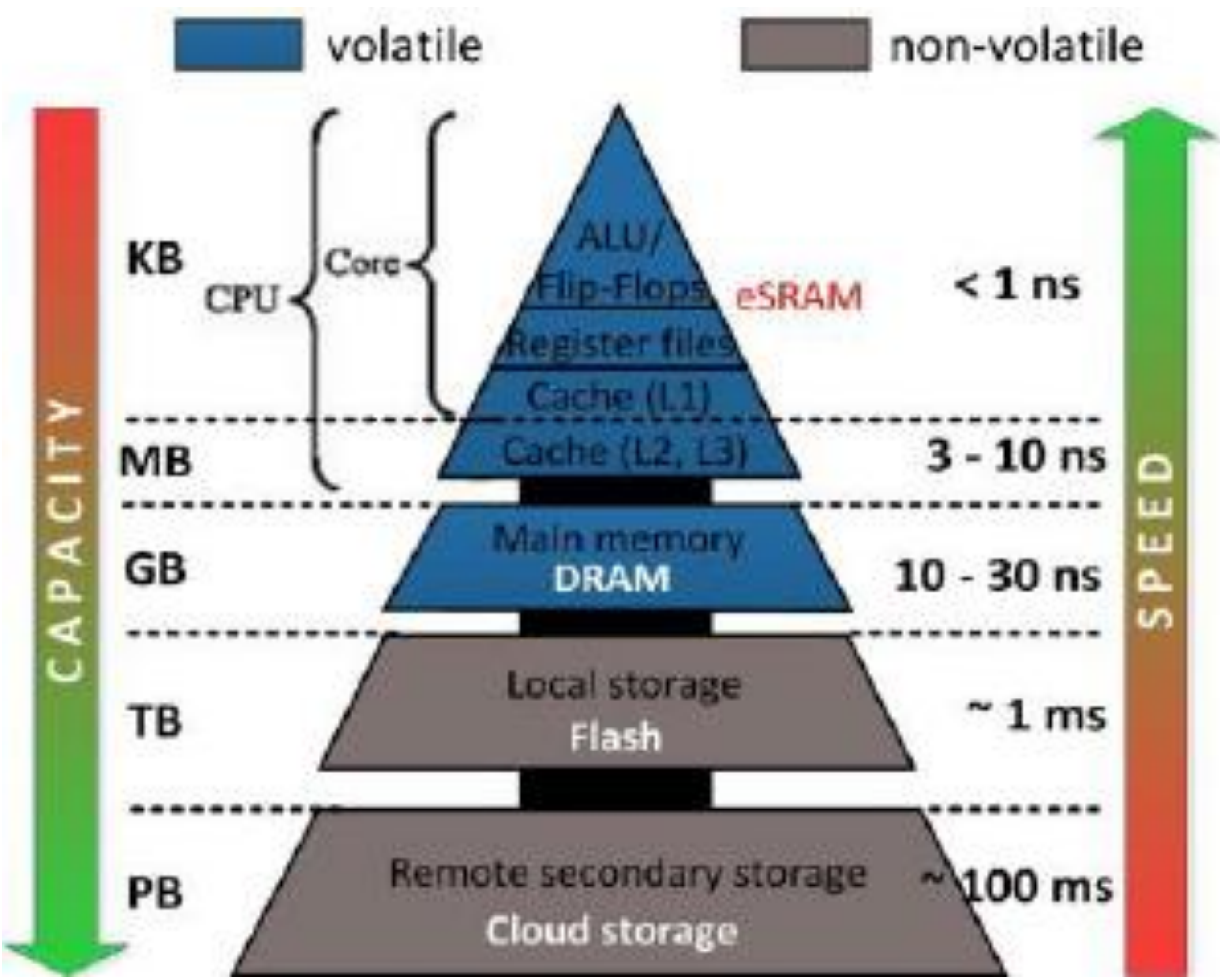
- Software tends to expand to fill the memory available
  - The 1 terabyte of memory – the researchers already want more.
- Operating systems must manage memory

Memory Hierarchy

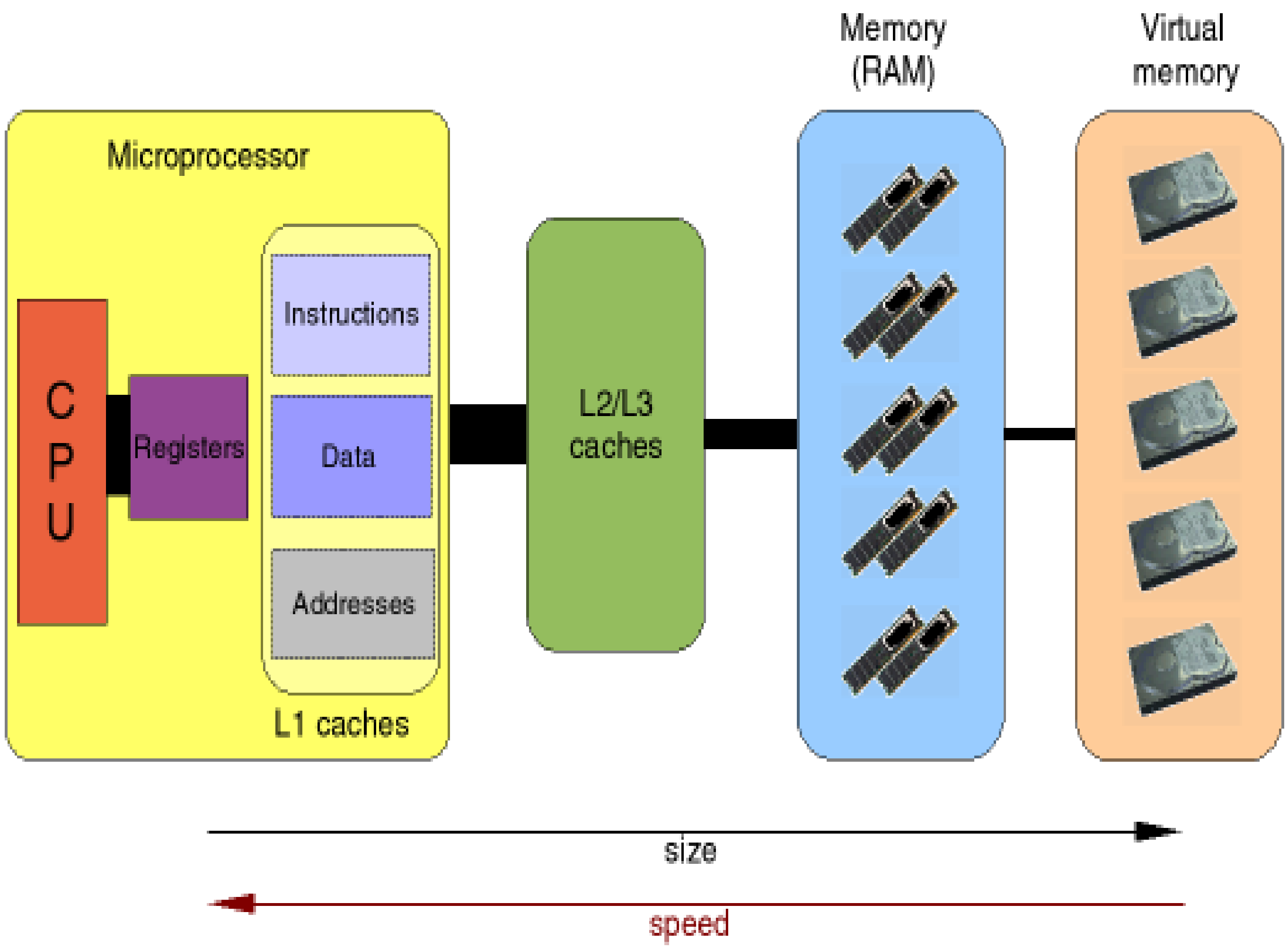


- A CPU waiting for data from main memory is not desired.
- Remedy: Add fast memory between the CPU and main memory called a cache.

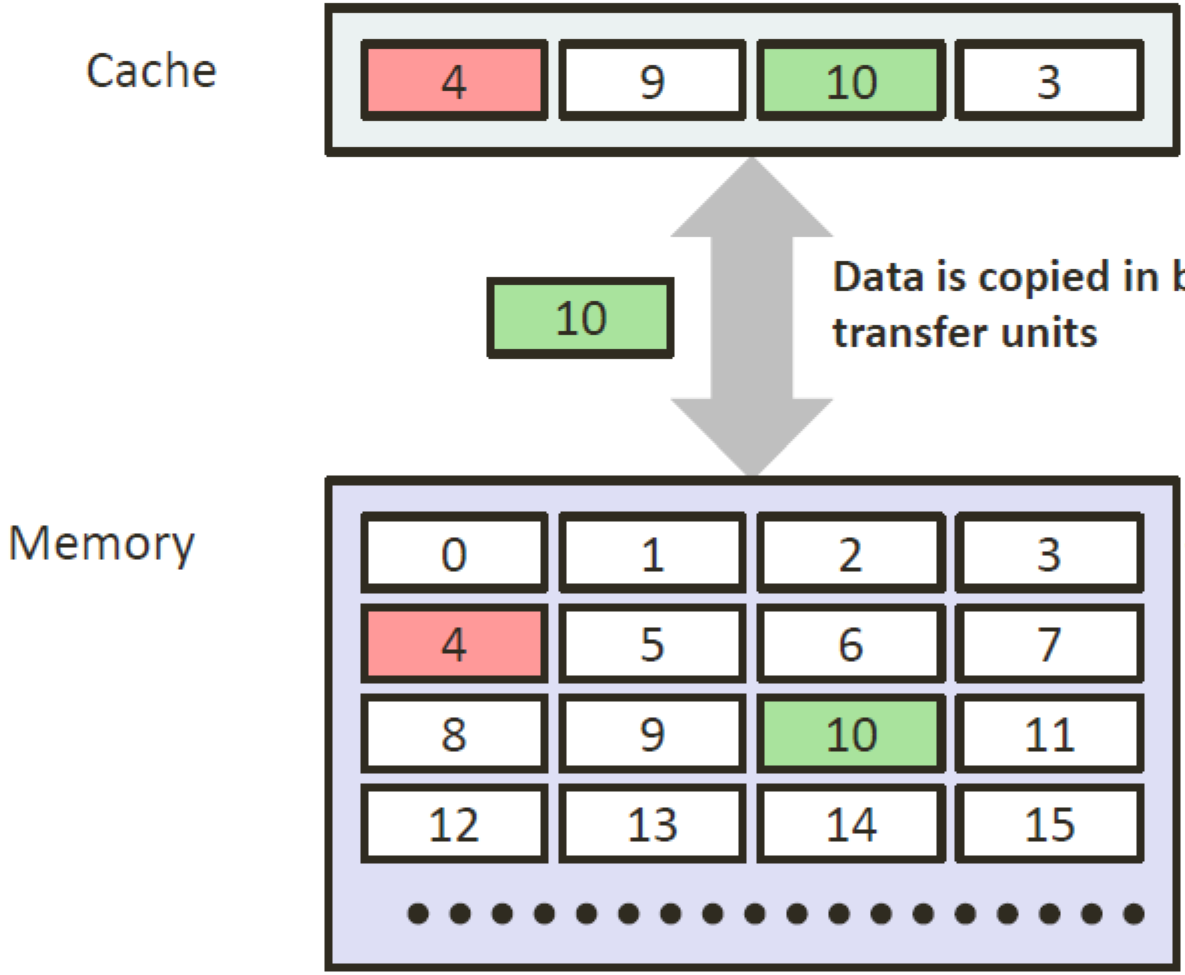
Memory Hierarchy



Memory Hierarchy



Memory Hierarchy

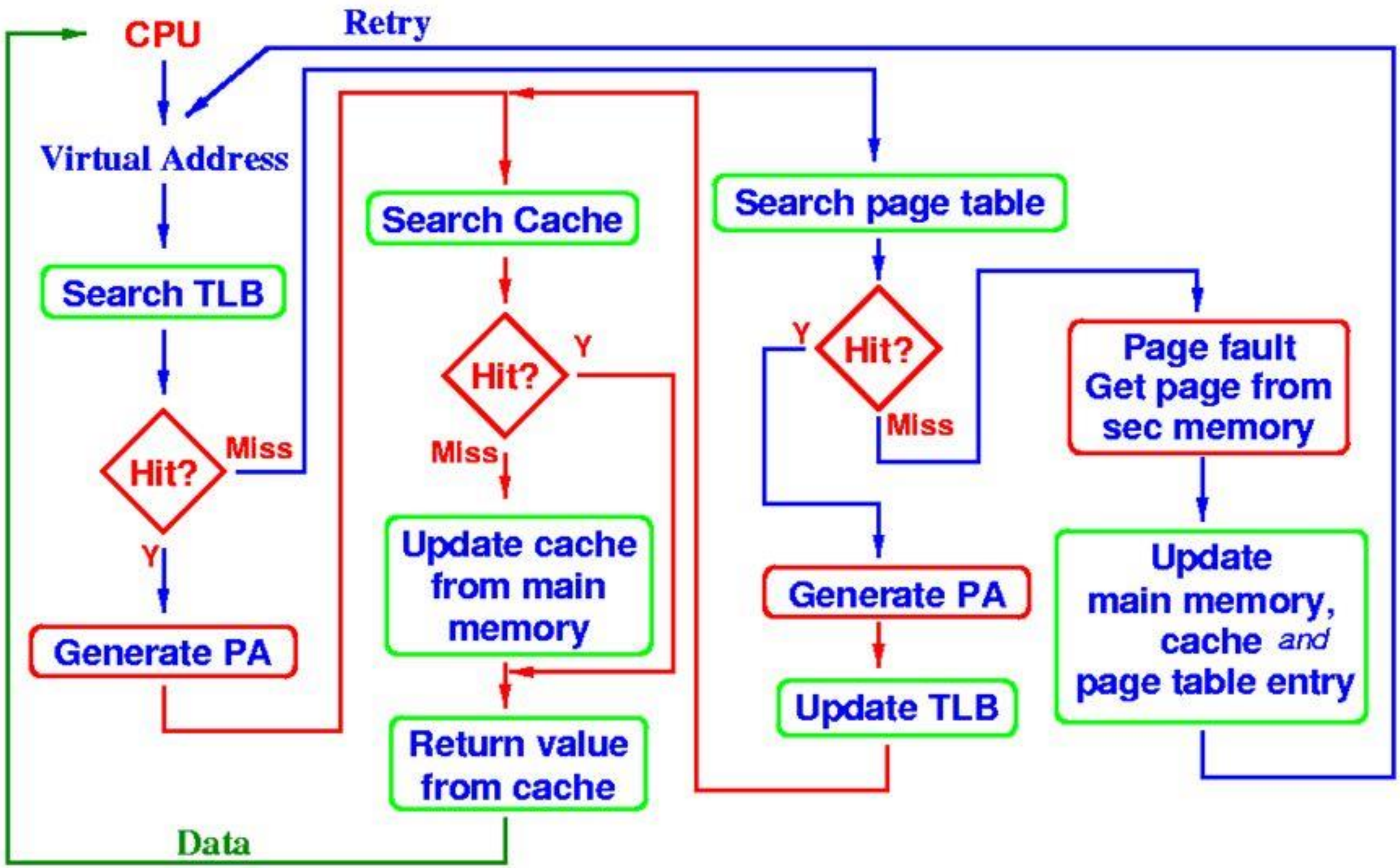


Smaller, faster, more expensive memory caches a subset of the blocks

Data is copied in block-sized transfer units

Larger, slower, cheaper memory viewed as partitioned into “blocks”

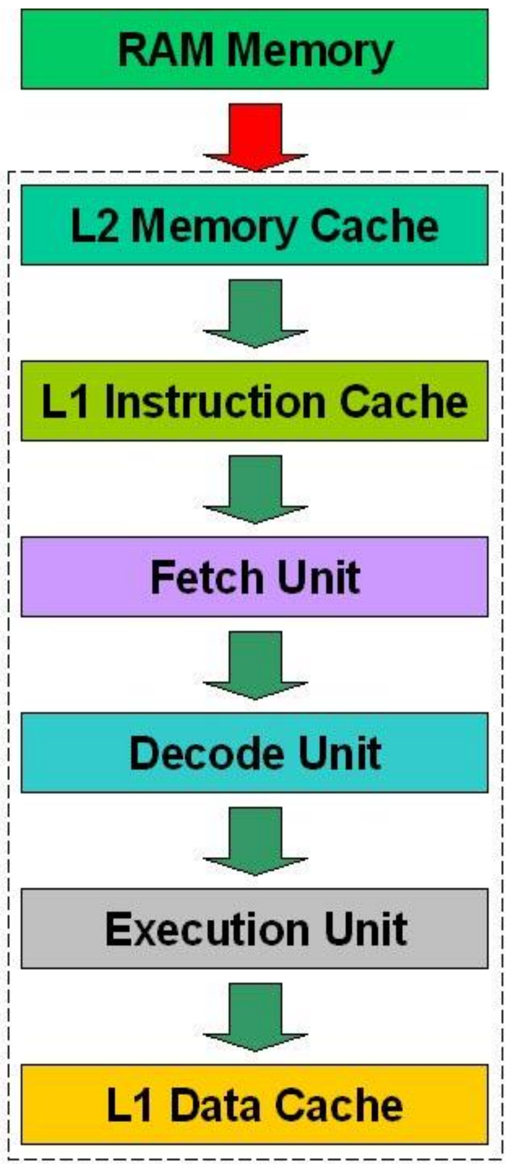
Memory Hierarchy - Operation

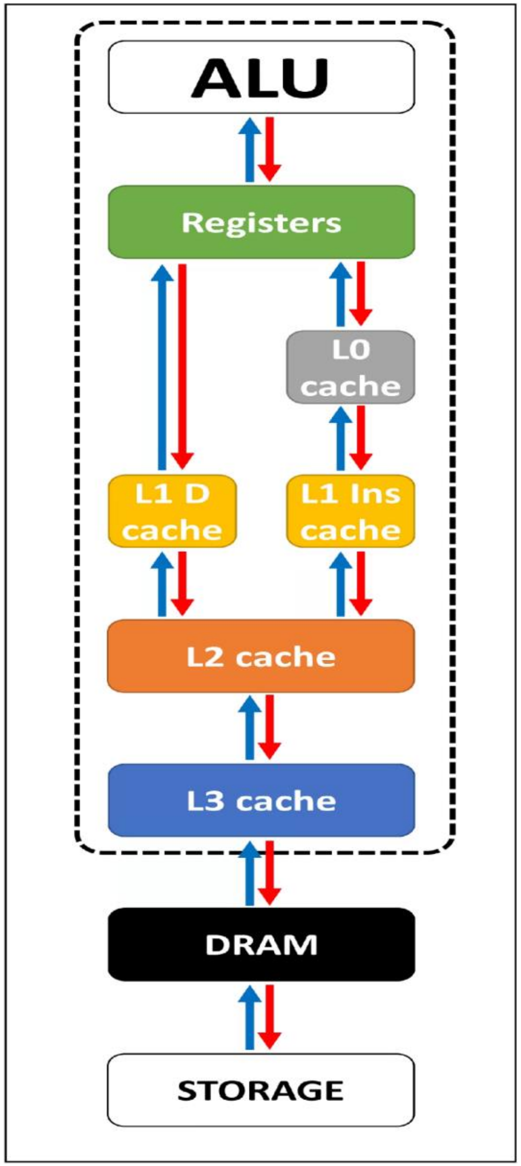


Memory hierarchy

- What is the memory hierarchy?
  - Different levels of memory
  - Some are small & fast
  - Others are large & slow
- What levels are usually included?
  - Cache: small amount of fast, expensive memory
    - L1 (level 1) cache: usually on the CPU chip
    - L2 & L3 cache: off-chip, made of SRAM in old machines
  - Main memory: medium-speed, medium price memory (DRAM)
  - Disk: many gigabytes of slow, cheap, non-volatile storage
- Memory manager handles the memory hierarchy

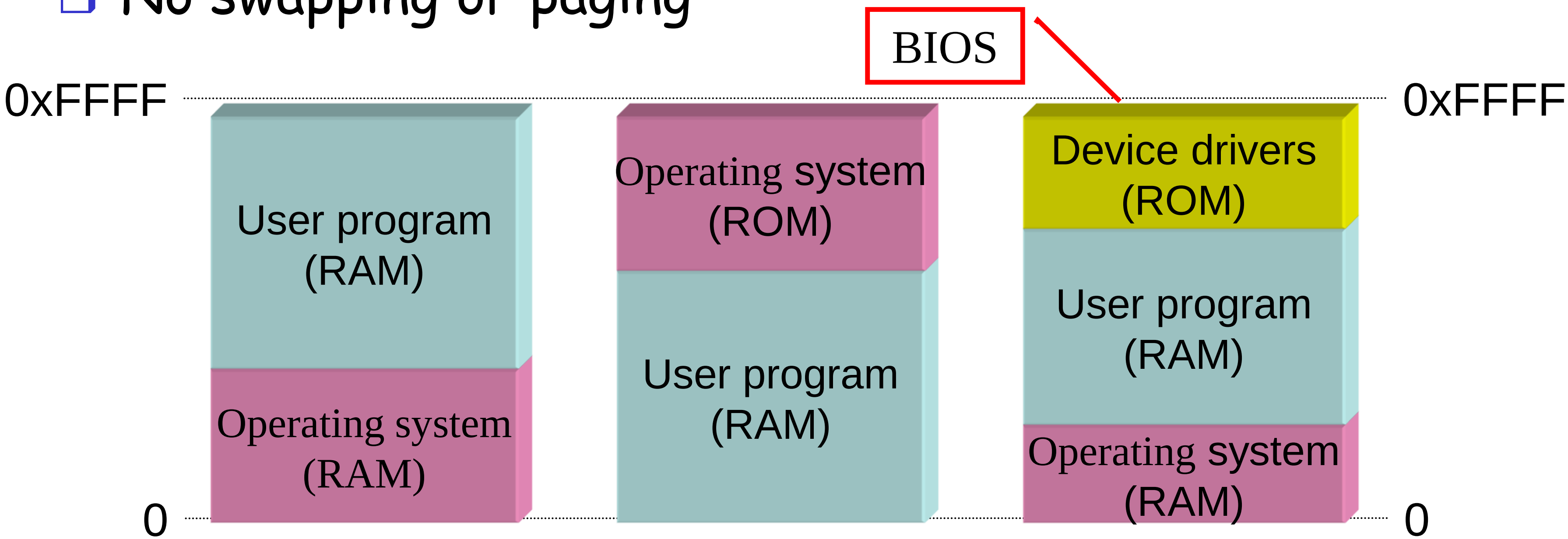
Memory hierarchy





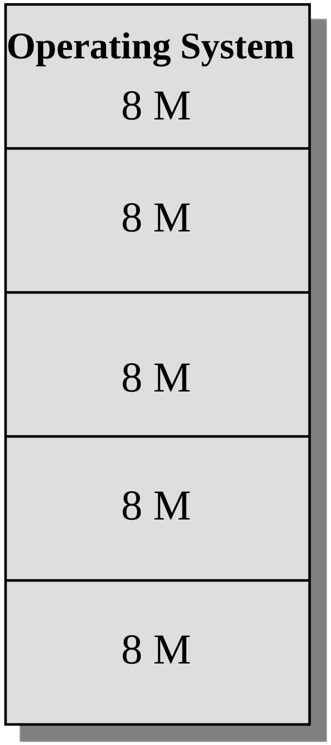
Basic memory management

- Components include
  - Operating system (perhaps with device drivers)
  - Single process
- Goal: lay these out in memory
  - Memory protection may not be an issue (only one program)
  - Flexibility may still be useful (allow OS changes, etc.)
- No swapping or paging



Fixed Partitioning

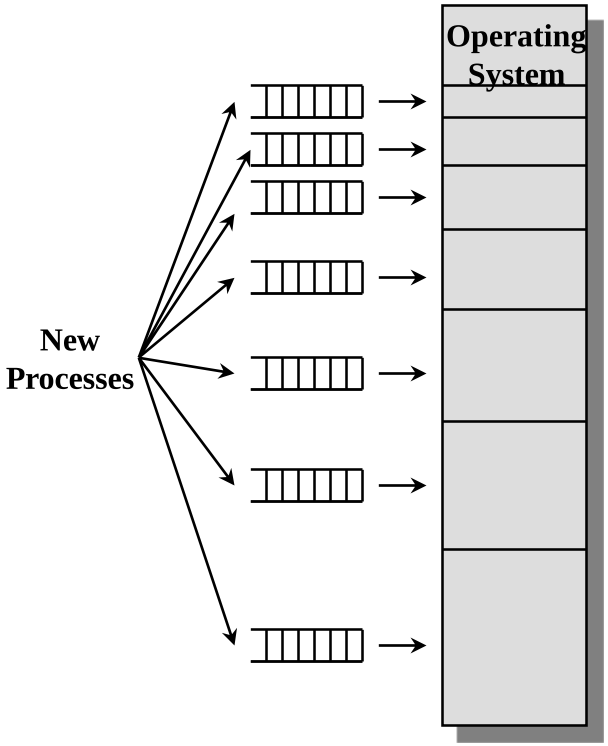
- Main memory use is inefficient.
- Any program, no matter how small, occupies an entire partition.
  - This is internal fragmentation.



Placement Algorithm with Partitions

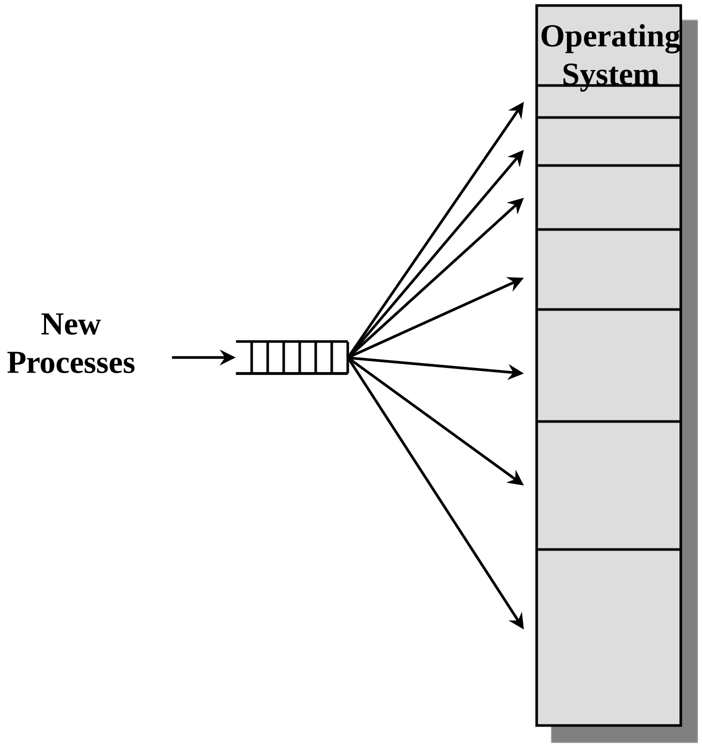
- Equal-size partitions
  - Since all partitions are of equal size, it does not matter which partition is used
- Unequal-size partitions
  - Each process can be assigned to the smallest partition within which it will fit
  - There is a queue for each partition
  - Processes are assigned in such a way as to minimize wasted memory within a partition

One Process Queue per Partition



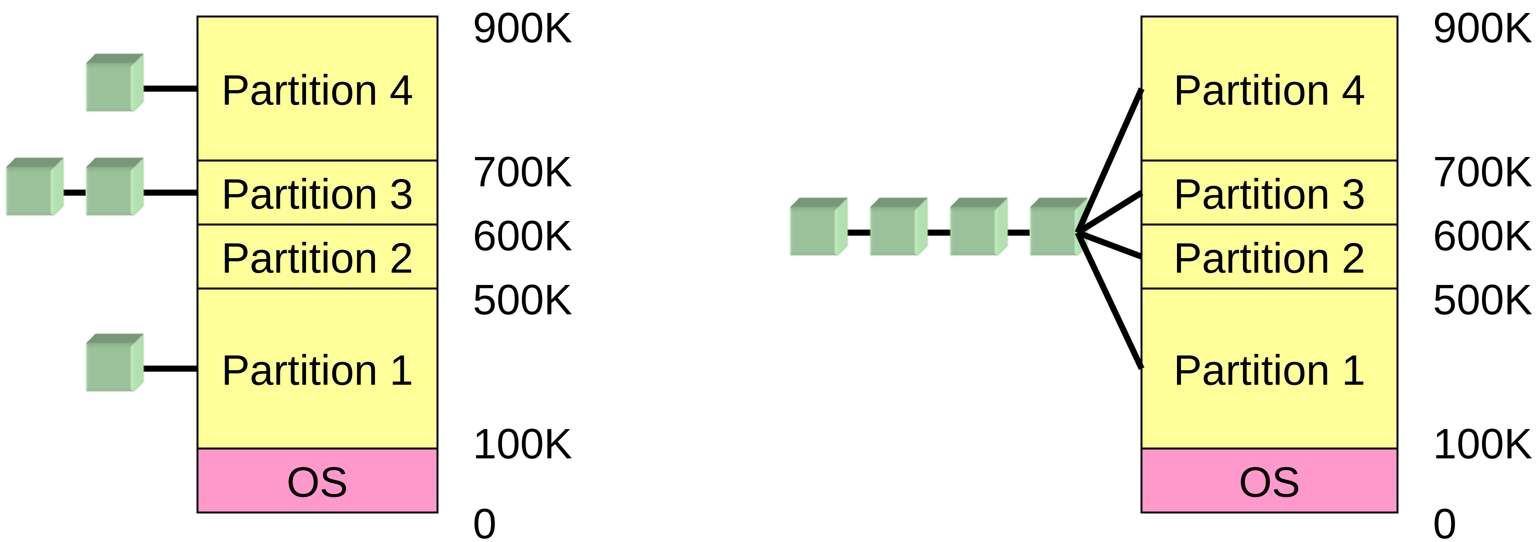
One Common Process Queue

- When its time to load a process into main memory the smallest available partition that will hold the process is selected



Fixed partitions: multiple programs

- Fixed memory partitions
  - Divide memory into fixed spaces
  - Assign a process to a space when it's free
- Mechanisms
  - Separate input queues for each partition
  - Single input queue: better ability to optimize CPU usage

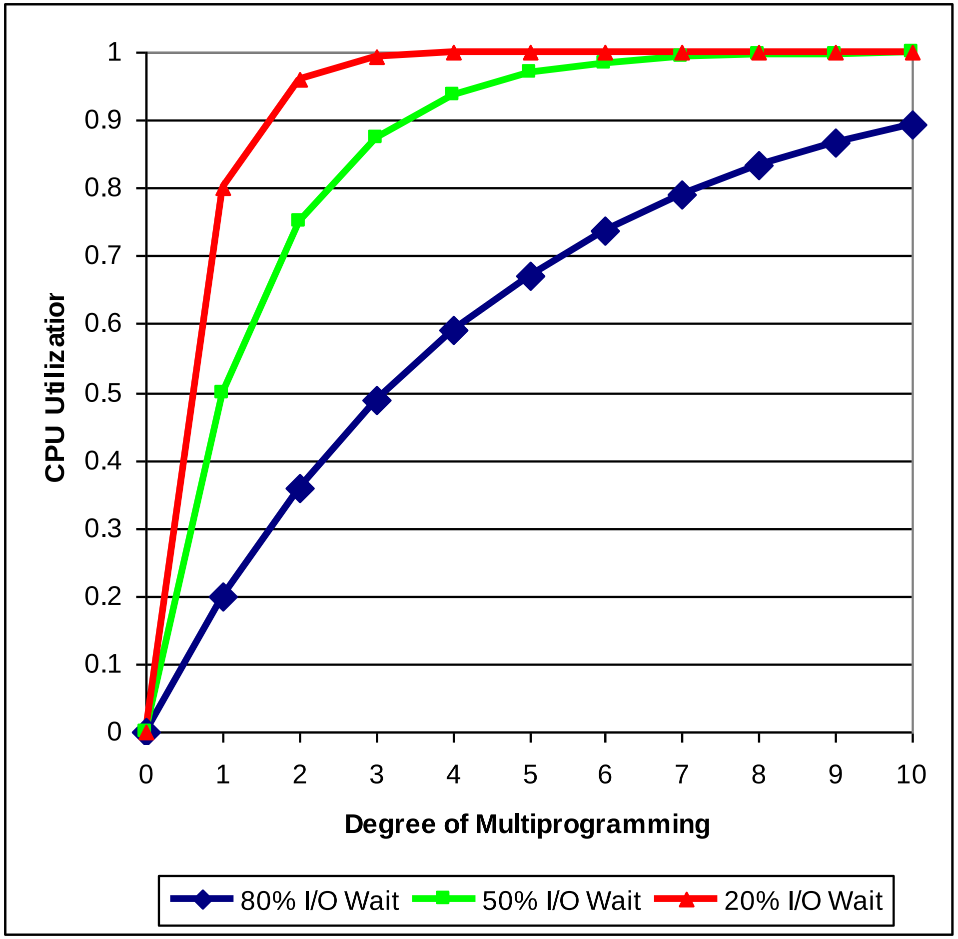


How many programs is enough?

- Several memory partitions (fixed or variable size)
- Lots of processes wanting to use the CPU
- Tradeoff
  - More processes utilize the CPU better
  - Fewer processes use less memory (cheaper!)
- How many processes do we need to keep the CPU fully utilized?
  - This will help determine how much memory we need
  - Is this still relevant with memory costing $100/16GB?

Modeling multiprogramming

- More I/O wait means less processor utilization
  - At 20% I/O wait, 3–4 processes fully utilize CPU
  - At 80% I/O wait, even 10 processes aren't enough
- This means that the OS should have more processes if they're I/O bound
- More processes => memory management & protection more important!

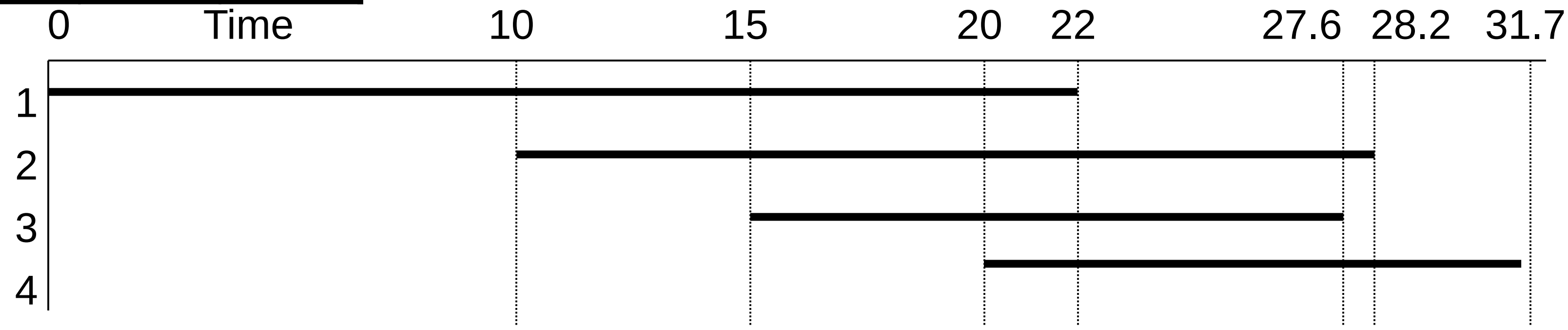


Multiprogrammed system performance

- Arrival and work requirements of 4 jobs
- CPU utilization for 1–4 jobs with 80% I/O wait
- Sequence of events as jobs arrive and finish
  - Numbers show amount of CPU time jobs get in each interval
  - More processes => better utilization, less time per process

Job	Arrival time	CPU needed
1	10:00	4
2	10:10	3
3	10:15	2
4	10:20	2

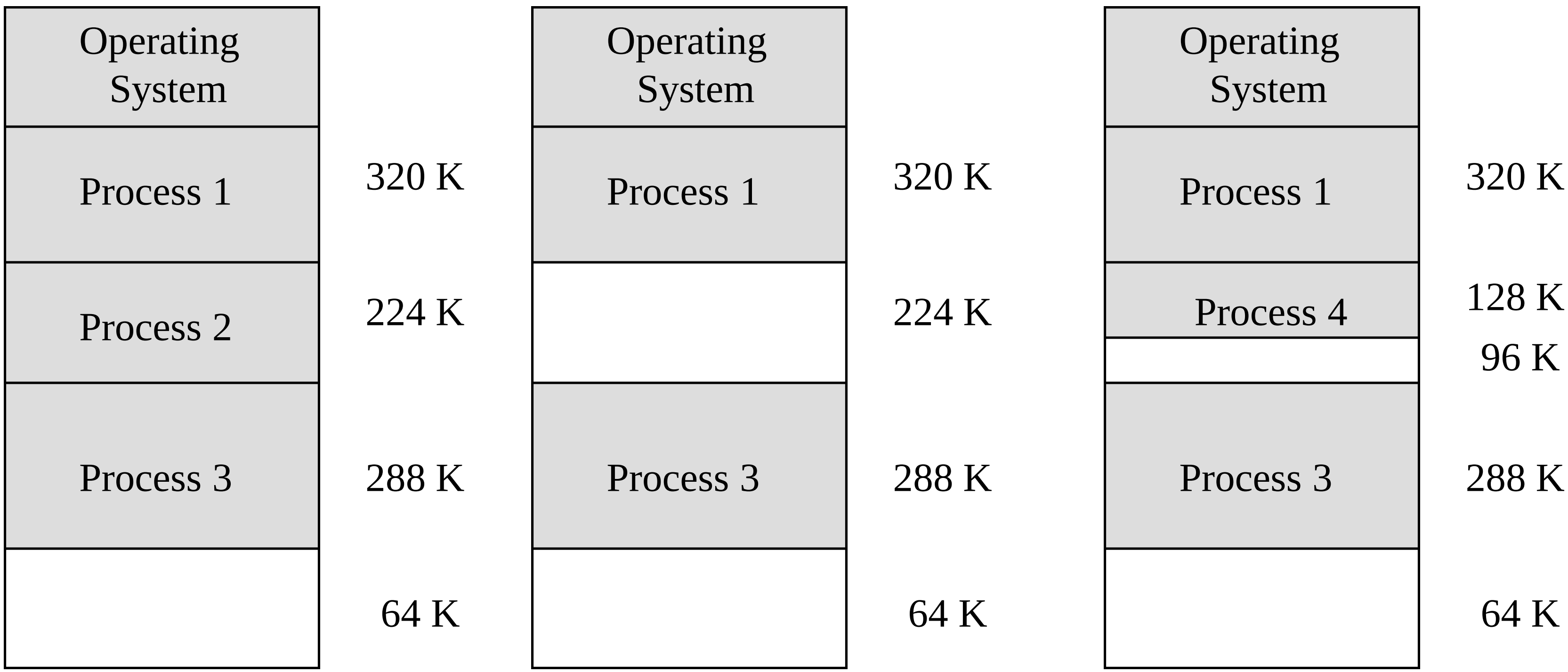
	1	2	3	4
CPU idle	0.80	0.64	0.51	0.41
CPU busy	0.20	0.36	0.49	0.59
CPU/process	0.20	0.18	0.16	0.15



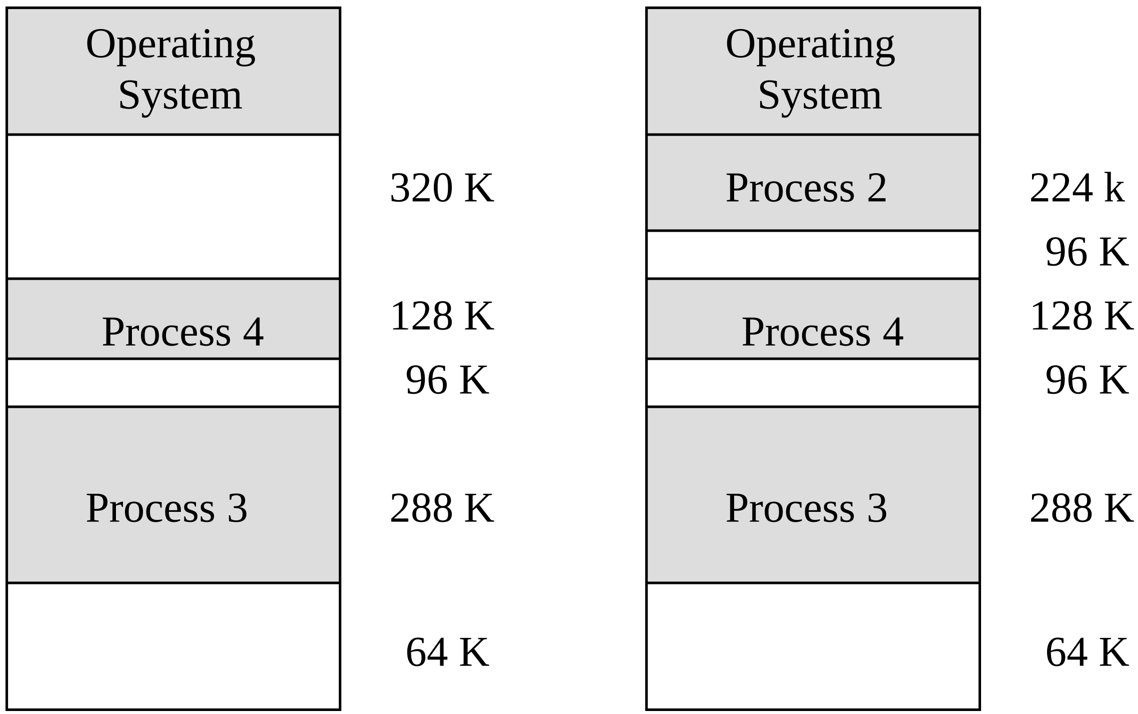
Dynamic Partitioning

- Partitions are of variable length and number
- Process is allocated exactly as much memory as required
- Eventually get holes in the memory. This is called external fragmentation
- Compaction is required to obtain a large block at the end of memory
  - Shift processes so they are contiguous and all free memory is in one block

Example Dynamic Partitioning



Example Dynamic Partitioning



Dynamic Partitioning Placement Algorithm

- Operating system must decide which free block to allocate to a process
- Best-fit algorithm
  - Choose the block that is closest in size to the request
  - This has the worst overall performance
  - The smallest block is found for a process
    - The smallest amount of fragmentation is left;
    - Memory compaction must be done more often

Dynamic Partitioning Placement Algorithm

- First-fit algorithm
  - Starts scanning memory from the beginning and chooses the first available block that is large enough.
- Next-fit
  - Starts scanning memory from the location of the last placement and chooses the next available block that is large enough

Memory and multiprogramming

- Memory needs two things for multiprogramming
  - Relocation
  - Protection
- The OS cannot be certain where a program will be loaded in memory
  - Variables and procedures can't use absolute locations in memory
  - Several ways to guarantee this
- The OS must keep processes' memory separate
  - Protect a process from other processes reading or modifying its own memory
  - Protect a process from modifying its own memory in undesirable ways (such as writing to program code)

Registers Used during Execution

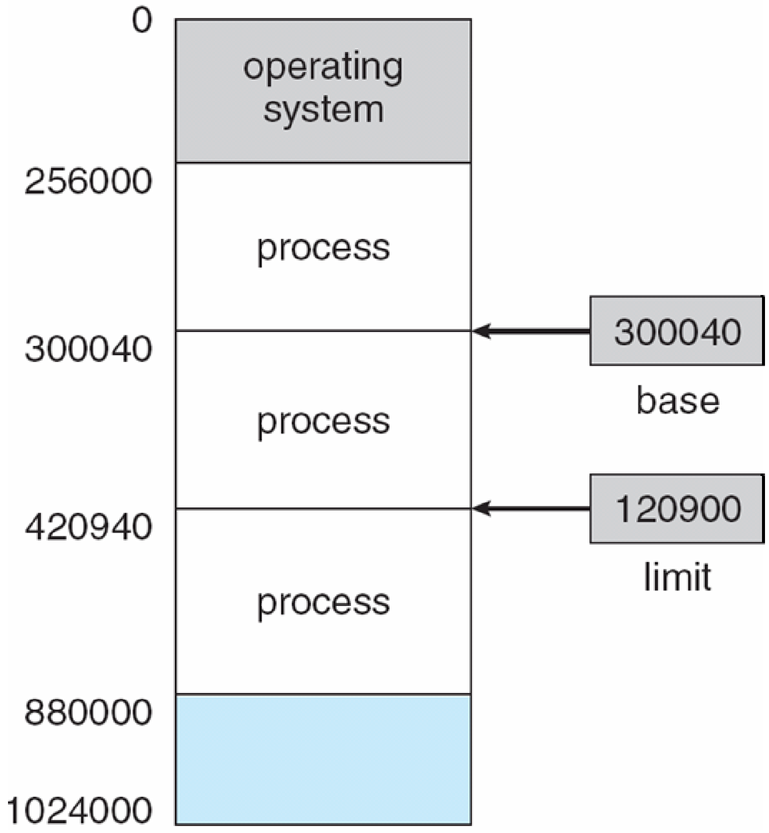
- Base register
  - Starting address for the process
- Bounds register
  - Ending location of the process
- These values are set when the process is loaded and when the process is swapped in

Registers Used during Execution

- The value of the base register is added to a relative address to produce an absolute address
- The resulting address is compared with the value in the bounds register
- If the address is not within bounds, an interrupt is generated to the operating system

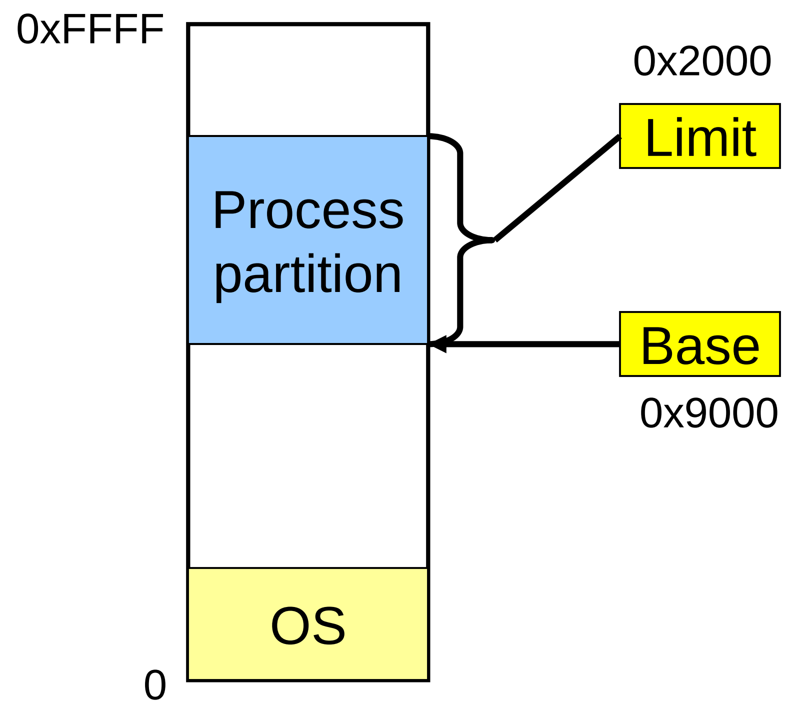
Base and Limit Registers

- A pair of base and limit registers define the logical address space



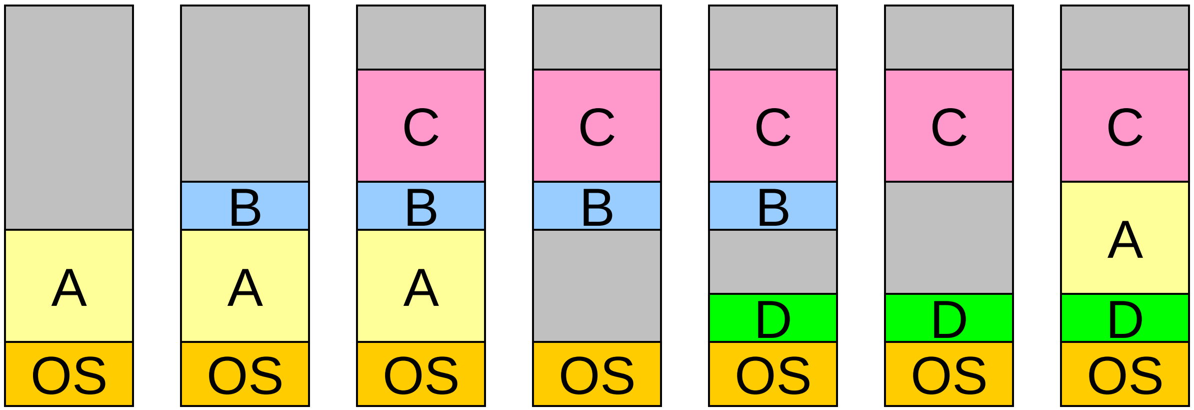
Base and limit registers

- Special CPU registers: base & limit
  - Access to the registers limited to system mode
  - Registers contain
    - Base: start of the process's memory partition
    - Limit: length of the process's memory partition
- Address generation
  - Physical address: location in actual memory
  - Logical address: location from the process's point of view
  - Physical address = base + logical address
  - Logical address larger than limit => error



Logical address: 0x1204
Physical address:
0x1204+0x9000 = 0xA204

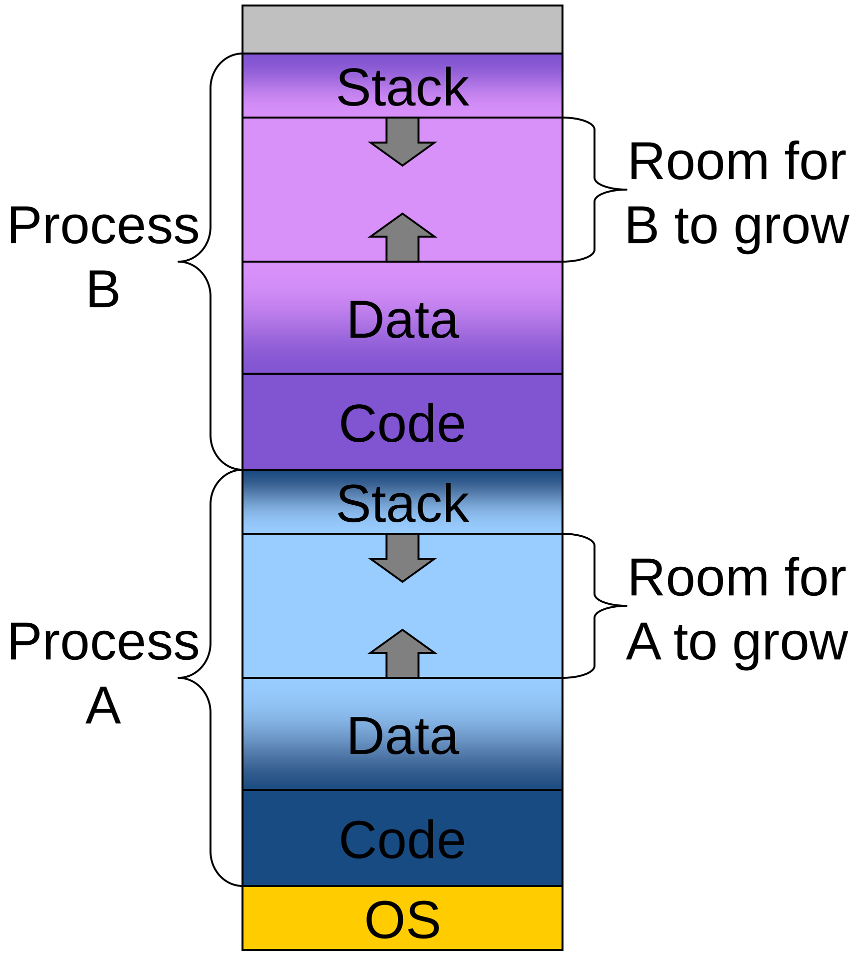
Swapping



- Memory allocation changes as
  - Processes come into memory
  - Processes leave memory
    - Swapped to disk
    - Complete execution
- Gray regions are unused memory

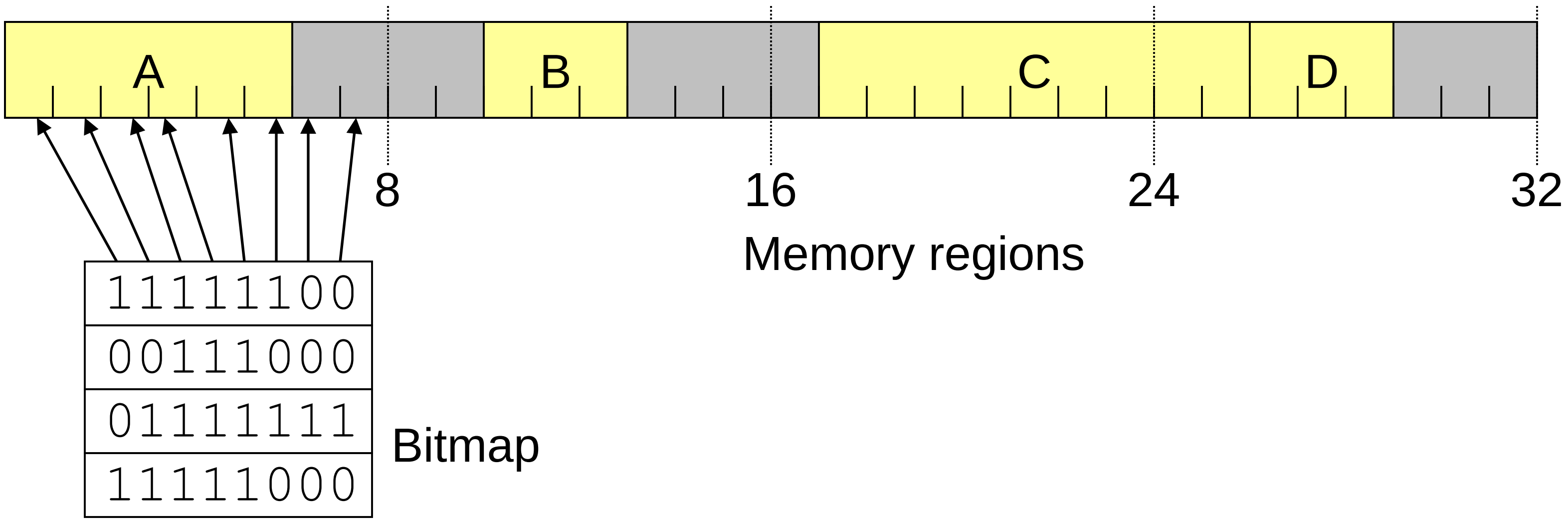
Swapping: leaving room to grow

- Need to allow for programs to grow
  - Allocate more memory for data
  - Larger stack
- Handled by allocating more space than is necessary at the start
  - Inefficient: wastes memory that's not currently in use
  - What if the process requests too much memory?



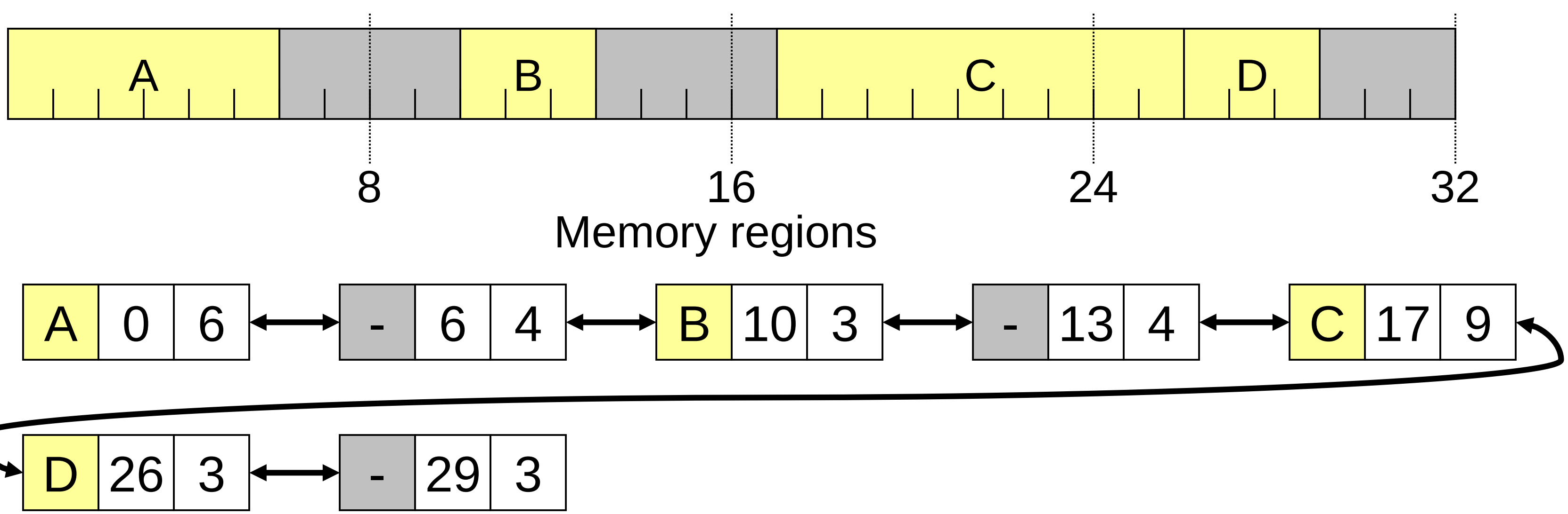
Tracking memory usage: bitmaps

- Keep track of free / allocated memory regions with a bitmap
  - One bit in map corresponds to a fixed-size region of memory
  - Bitmap is a constant size for a given amount of memory regardless of how much is allocated at a particular time
- Chunk size determines efficiency
  - At 1 bit per 4KB chunk, we need just 256 bits (32 bytes) per MB of memory
  - For smaller chunks, we need more memory for the bitmap
  - Can be difficult to find large contiguous free areas in bitmap



Tracking memory usage: linked lists

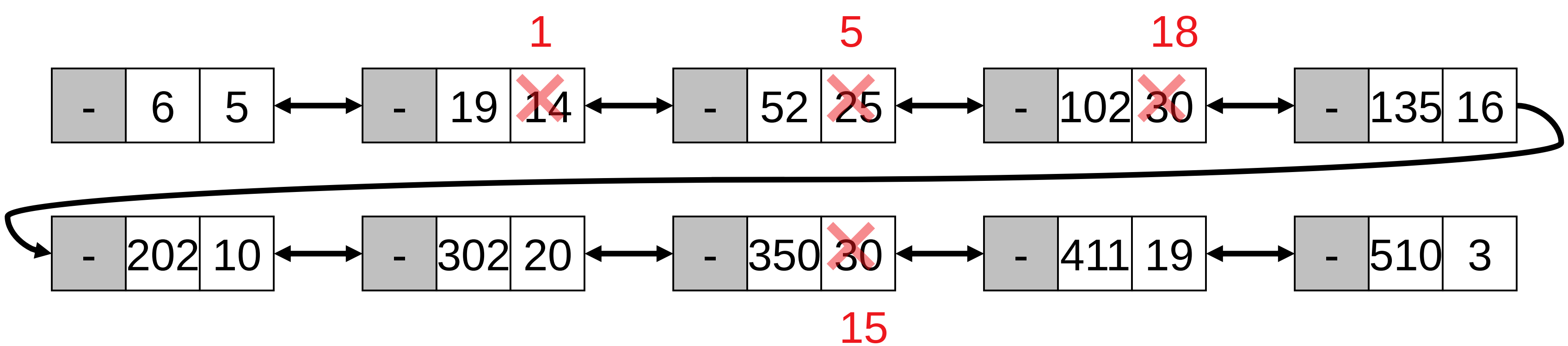
- Keep track of free / allocated memory regions with a linked list
  - Each entry in the list corresponds to a contiguous region of memory
  - Entry can indicate either allocated or free (and, optionally, owning process)
  - May have separate lists for free and allocated areas
- Efficient if chunks are large
  - Fixed-size representation for each region
  - More regions => more space needed for free lists



Allocating memory

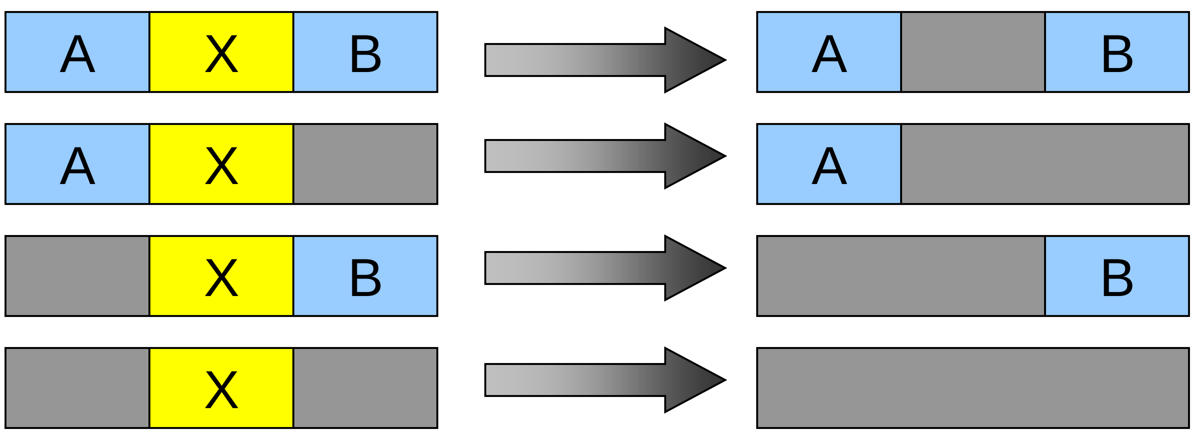
- Search through region list to find a large enough space
- Suppose there are several choices: which one to use?
  - First fit: the first suitable hole on the list
  - Next fit: the first suitable after the previously allocated hole
  - Best fit: the smallest hole that is larger than the desired region (wastes least space?)
  - Worst fit: the largest available hole (leaves largest fragment)
- Option: maintain separate queues for different-size holes

Allocate 20 blocks first fit
Allocate 12 blocks next fit
Allocate 13 blocks best fit
Allocate 15 blocks worst fit



Freeing memory

- Allocation structures must be updated when memory is freed
- Easy with bitmaps: just set the appropriate bits in the bitmap
- Linked lists: modify adjacent elements as needed
  - Merge adjacent free regions into a single region
  - May involve merging two regions with the just-freed area



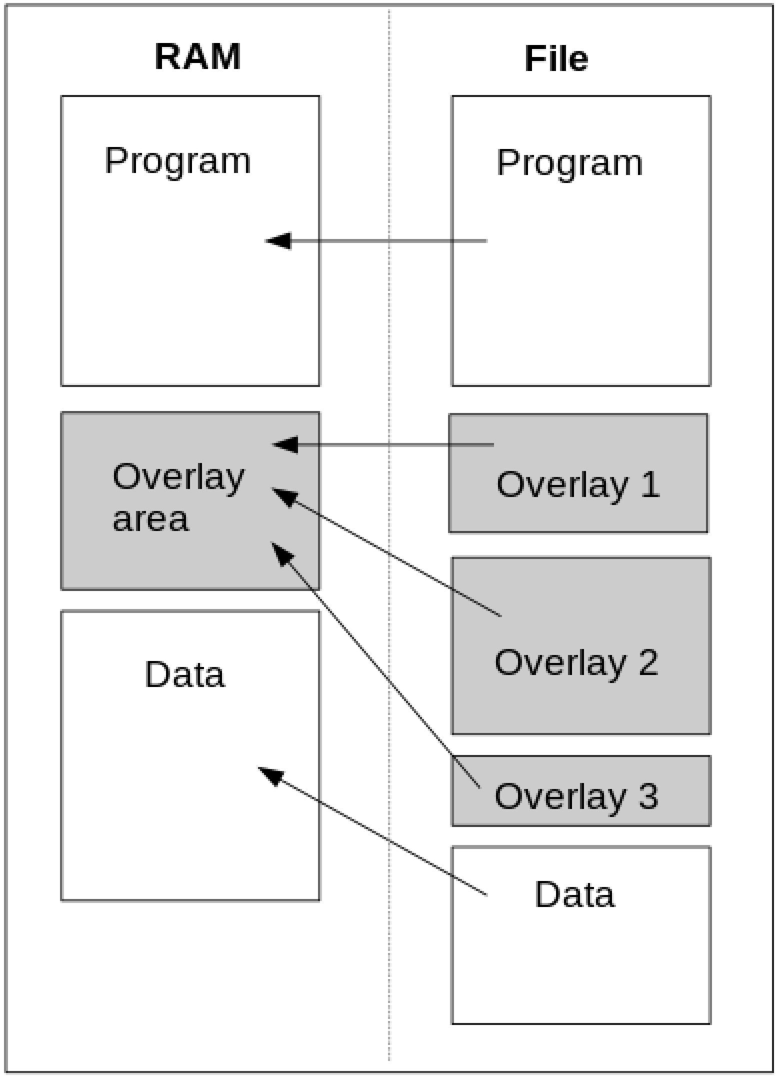
Limitations of swapping

- Problems with swapping
  - Process must fit into physical memory (impossible to run larger processes)
  - Memory becomes fragmented
    - External fragmentation: lots of small free areas
    - Compaction needed to reassemble larger free areas
  - Processes are either in memory or on disk: half and half doesn't do any good
- Overlays solved the first problem
  - Bring in pieces of the process over time (typically data)
  - Still doesn't solve the problem of fragmentation or partially resident processes

Overlays

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support from the operating system.
- Programming design of overlay structure is complex.

Overlays


RAM
File
Program
Program
Overlay area
Overlay 1
Overlay 2
Overlay 3
Data
Data

Summary

- This section studied basic memory allocation techniques