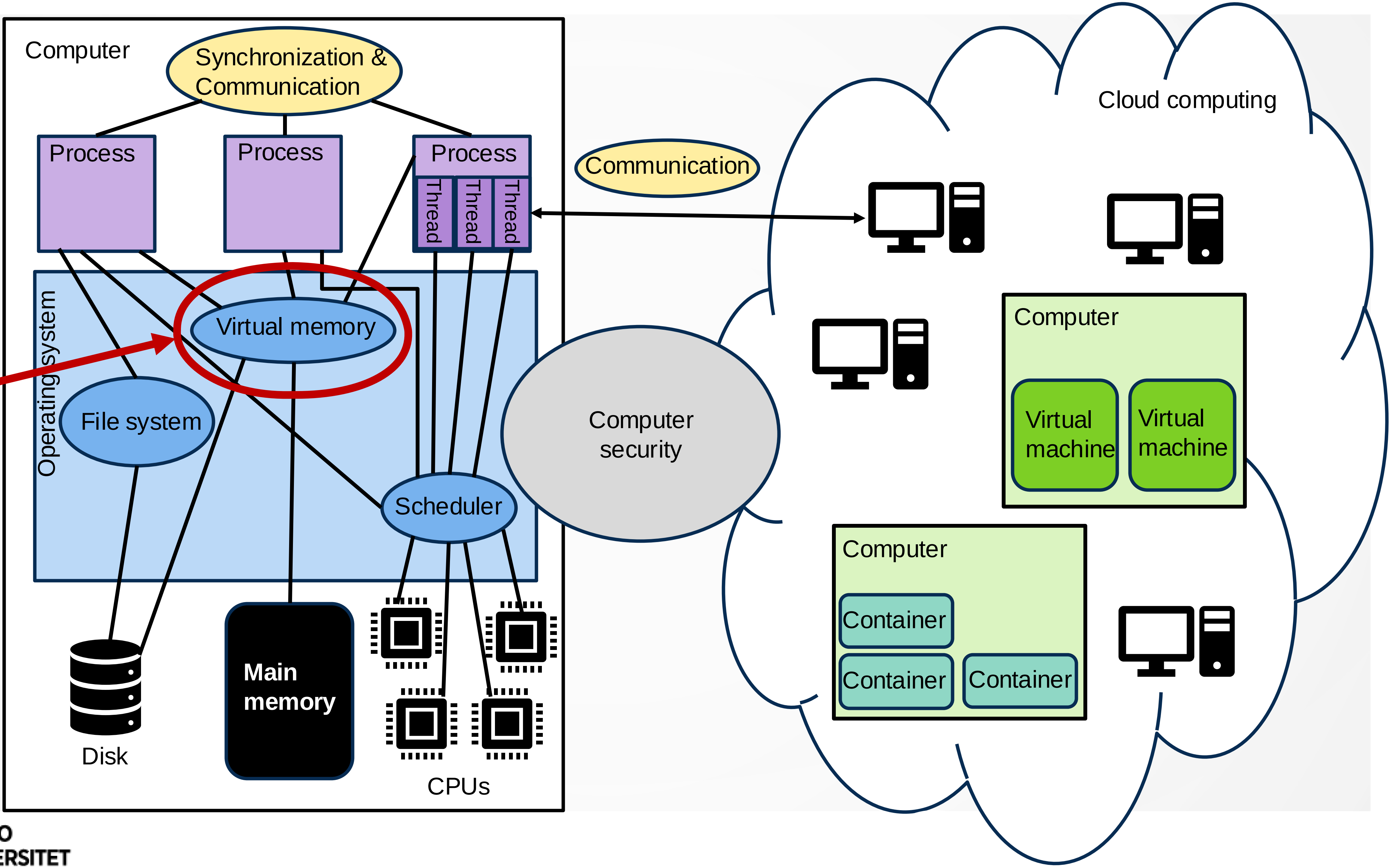
COMPUTING PLATFORMS

Virtual Memory:
Policies and algorithms



Today's topic

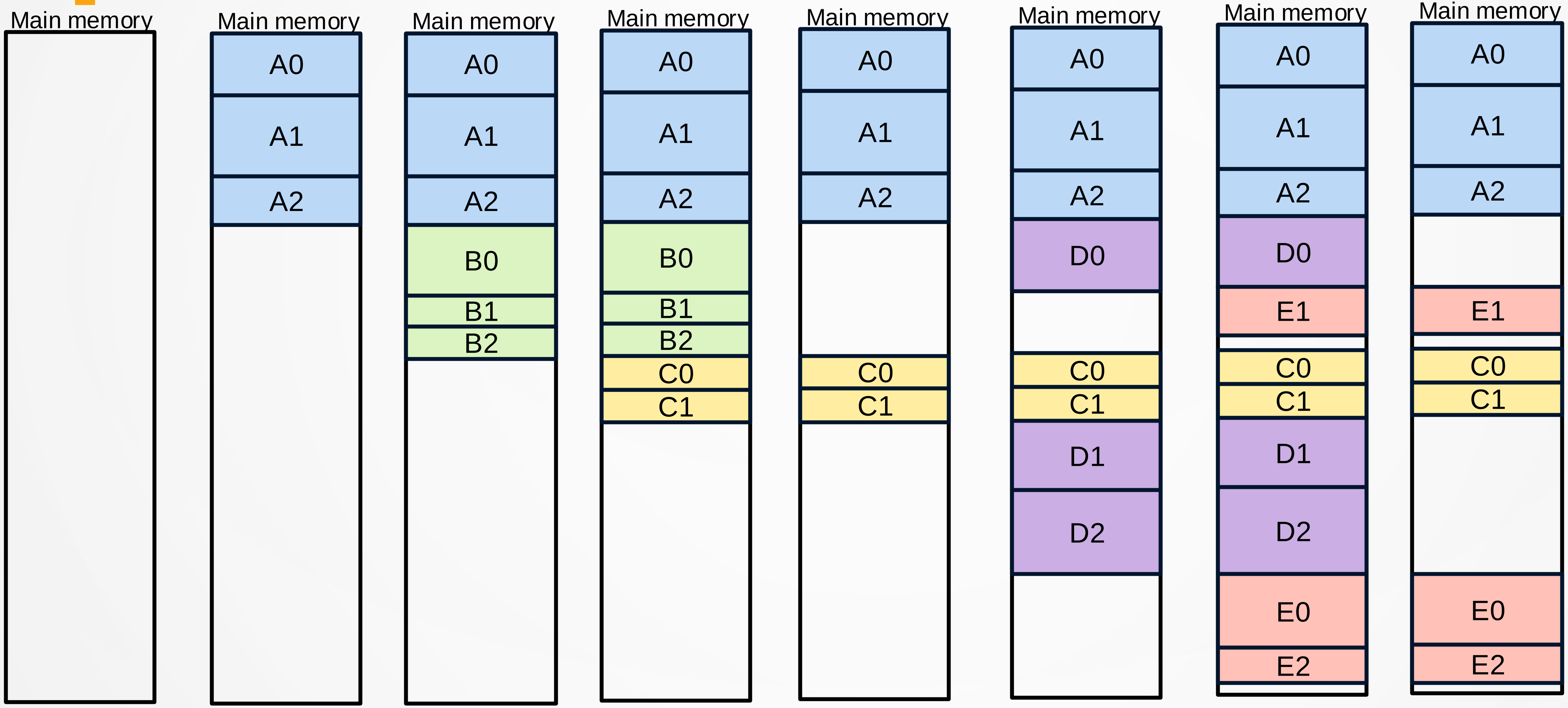
LEARNING OUTCOMES

- After today's lecture you
  - Can define the concept of virtual memory
  - Are able to describe how different OS mechanisms are used to implement memory management and virtual memory

FREE SPACE MANAGEMENT

- OS needs to keep track of free physical memory
- Paging: Easy!
  - All free frames are equal and memory always allocated one frame at a time
  - Just keep a list of free frames
- Segmentation: Difficult!
  - Free chunks of memory have different sizes
  - Fragmentation: The free chunks tend to get smaller and more numerous as time goes on
  - We will now concentrate on this
- Issues are the same as with any system managing free memory such as a programming language library handling memory allocation and deallocation

REMINDER: SEGMENTATION EXAMPLE



FREE SPACE MANAGEMENT: SEGMENTATION

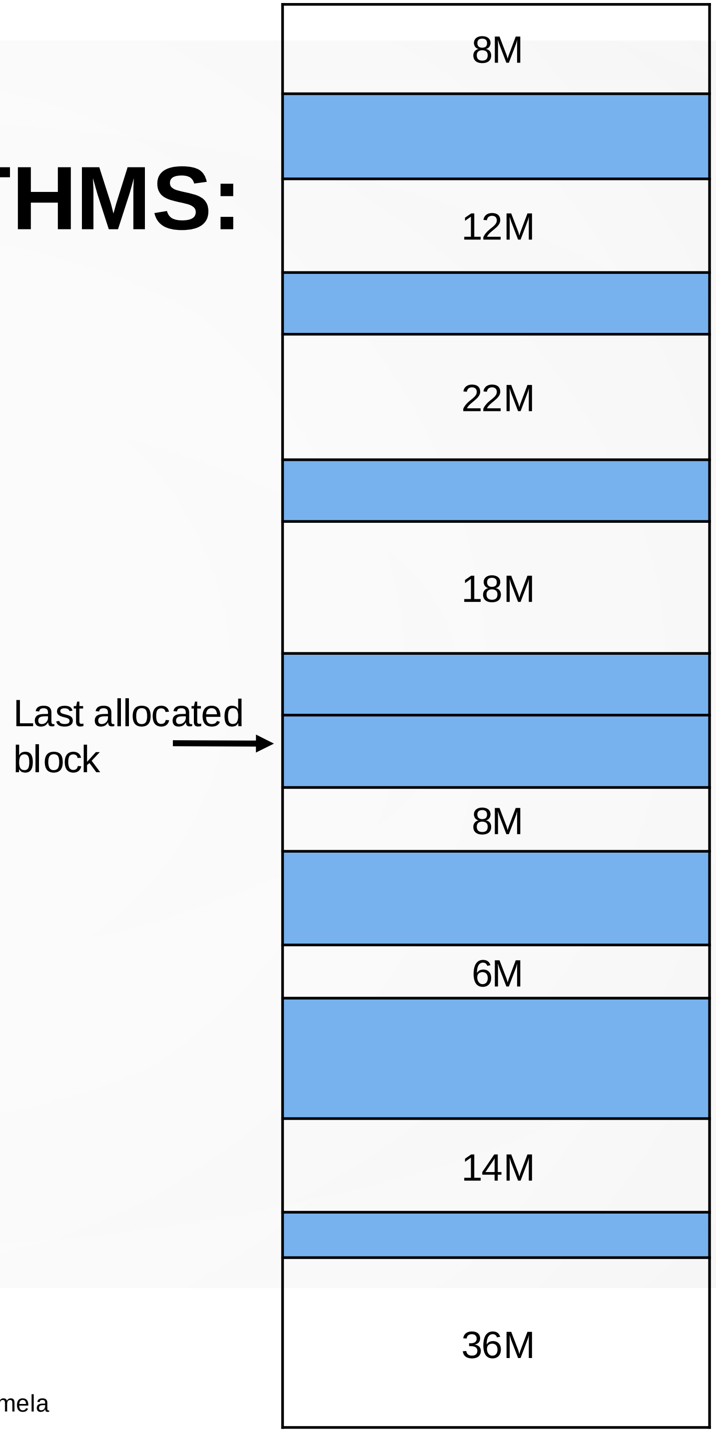
- Segments have varying size
- A segment is allocated exactly as much memory as it needs
  -> no internal fragmentation
- Disadvantage: **external fragmentation**
  - Memory becomes more and more fragmented, memory utilization declines
- Solution: **compaction**
  - OS shifts processes so that they are continuous
  - Free memory is together in one block
  - Time consuming and wastes CPU time

PLACEMENT ALGORITHMS

- **Best-fit**
  - Choose the block that is closest in size to the request
  - Attempts to leave large blocks free
  - Can be slow
- **First-fit**
  - Scan memory from the beginning and choose the first available block that is large enough
  - Can result in lots of small blocks in the beginning of the memory
  - Fast
- **Next-fit**
  - Scan memory from the location of the last placement and choose the next available block that is large enough
  - Fast
  - Attempts to avoid splintering the beginning of the free list

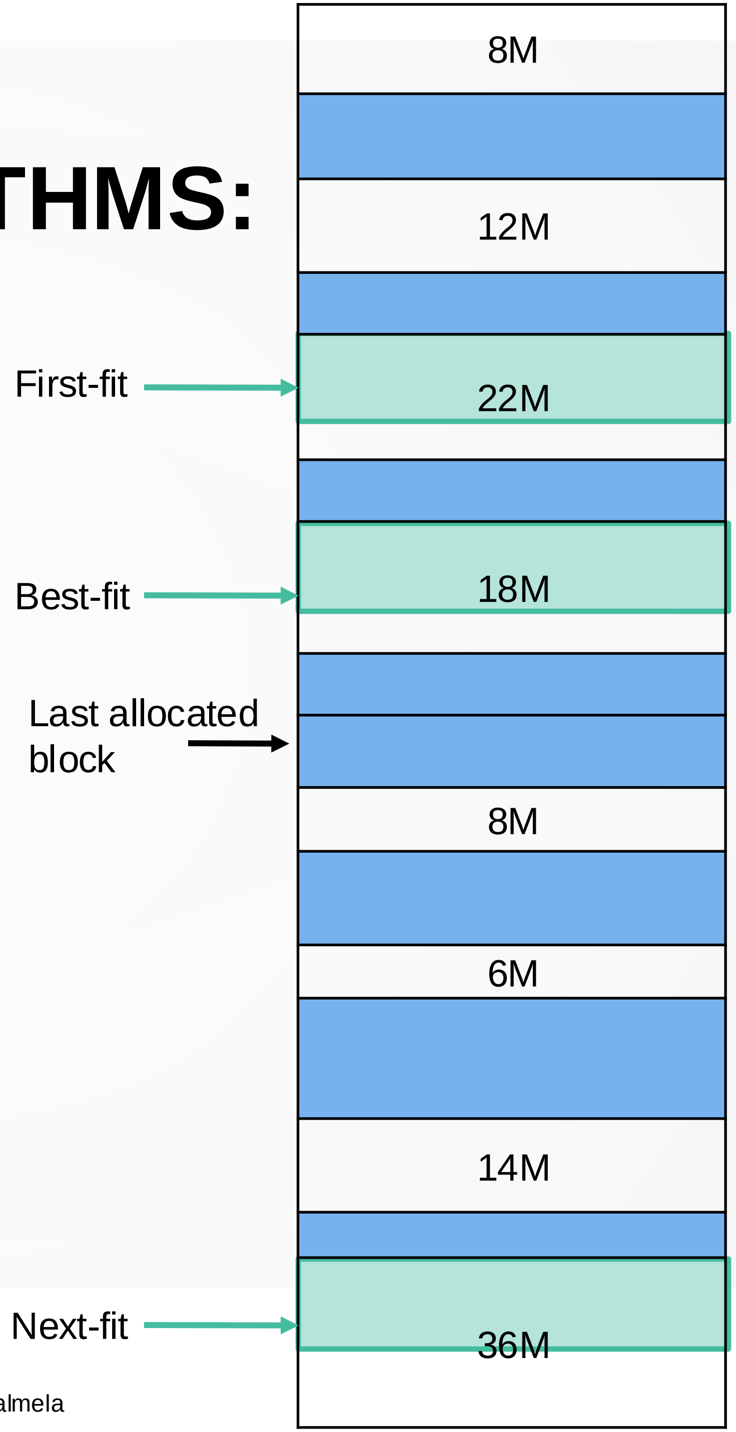
PLACEMENT ALGORITHMS: EXAMPLE

- Where is a 16M segment allocated?
  - Best-fit
  - First-fit
  - Next-fit



PLACEMENT ALGORITHMS: EXAMPLE

- Where is a 16M segment allocated?
  - Best-fit
  - First-fit
  - Next-fit



BUDDY SYSTEM

- Compromise between having fixed sized blocks available (e.g. paging) and having variable sized blocks available
- Memory available for allocation in blocks of size 2^K bytes, $L \leq K \leq U$, where
  - 2^L is the smallest size block that can be allocated
  - 2^U is the largest size block that can be allocated
  - Generally, 2^U is the size of the entire memory available for allocation

BUDDY SYSTEM

- Initially a single block of size 2^U available
- Keep a list of available blocks for each size 2^K
- When a request of size *s* is made:
  - Find the smallest **available** block that would fit *s*
  - If this is also the smallest block size where *s* fits, allocate it for *s*
  - If not, divide the block recursively into two equal size blocks (buddies) until smallest block size where *s* fits is reached.
- When a block is freed, combined it with its buddy recursively if the buddy is free

BUDDY SYSTEM: EXAMPLE

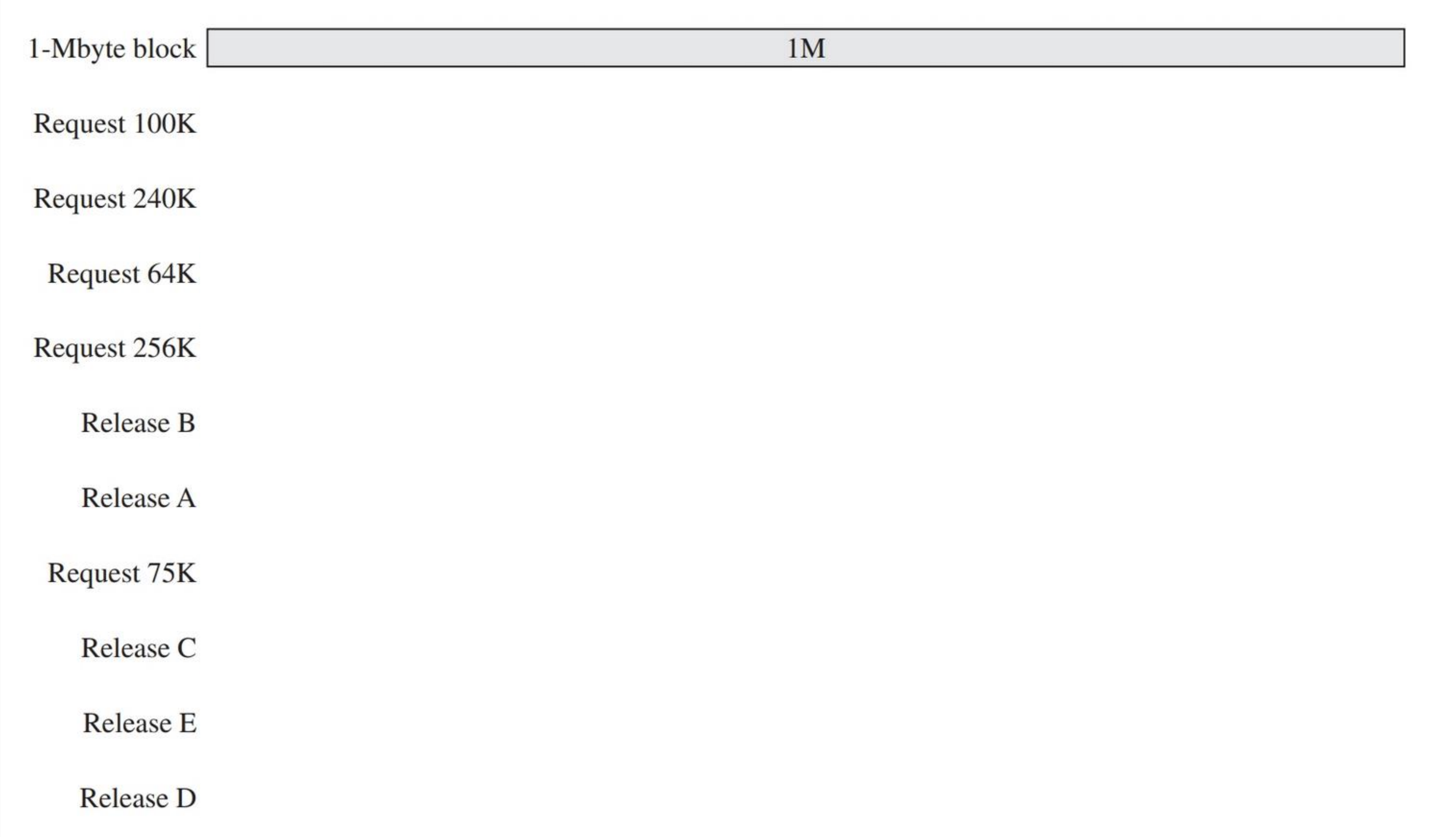


Figure 7.6 Example of the Buddy System

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

BUDDY SYSTEM: EXAMPLE

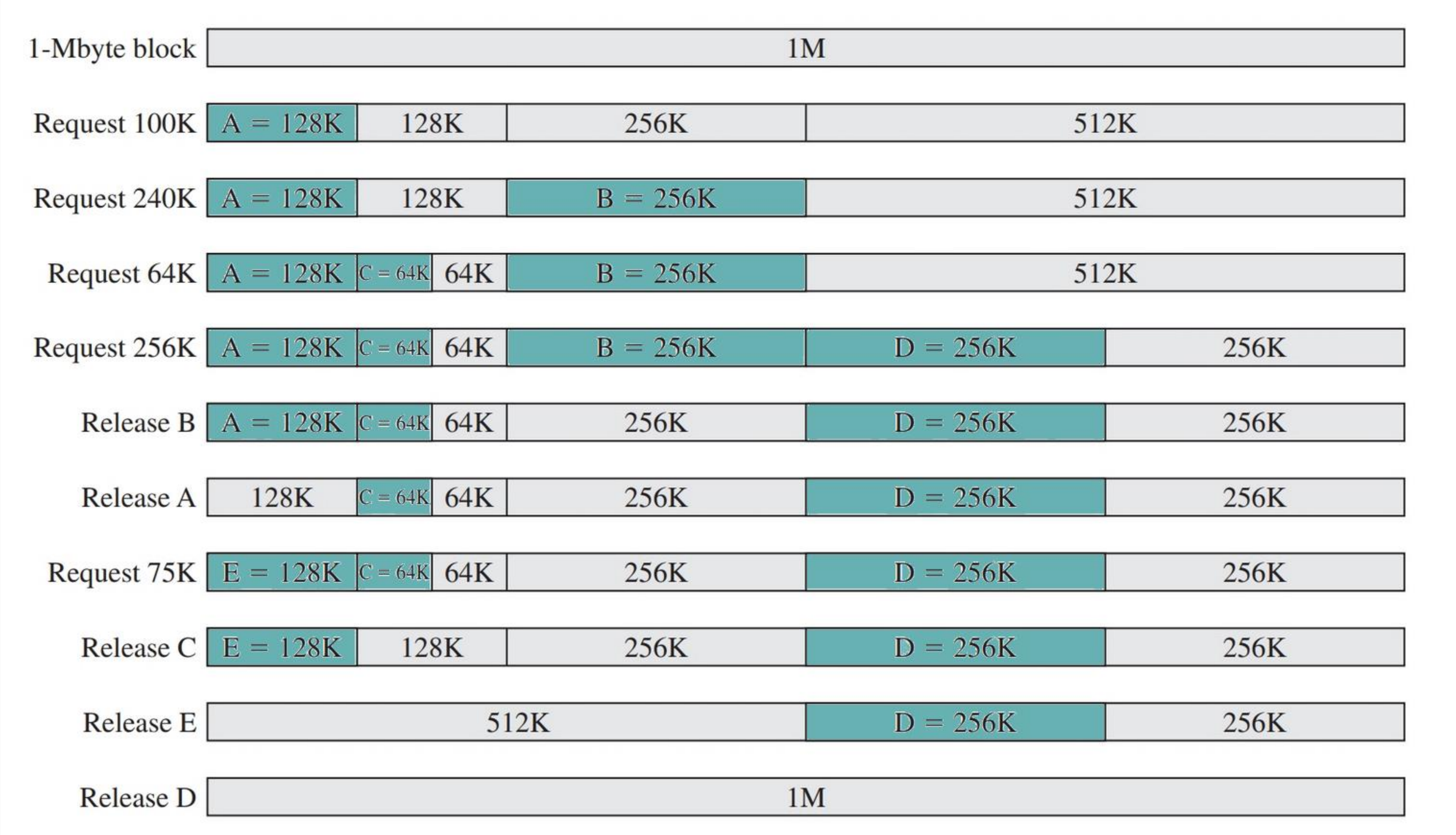


Figure 7.6 Example of the Buddy System

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

MEMORY MANAGEMENT

- Two characteristics fundamental to memory management
  - **All memory references are virtual addresses** dynamically translated to physical addresses at run time
  - **Process images are broken up into pieces** that don't need to be contiguously located in memory
- With these characteristics, not all pieces (pages or segments) of a process need to be in main memory during execution
  - If the piece holding the next instruction to be fetched and the piece holding the next data location to be referenced are in main memory, then execution may proceed

VIRTUAL MEMORY

- **Storage allocation scheme** in which **secondary memory** can be **addressed as though it were part of main memory**
- The addresses the program uses to reference memory are distinguished from the addresses that the memory system uses to identify physical storage sites: **program generated virtual addresses are translated automatically to the corresponding physical addresses**
- The size of the virtual storage is limited by the **addressing scheme of the computer system**, and by **the secondary memory available**

EXECUTION OF A PROCESS

- OS brings into main memory a **few pieces of a program**
  - **Resident set:** portion of program residing in main memory
- Interrupt (**page fault** interrupt) is generated when an address is needed that is not in main memory
  - OS places the process in **blocking state**
- Piece of process that contains the referenced logical address is brought into main memory
  - OS issues a disk IO Read request
  - Another process is dispatched to run while the disk IO takes place
  - IO interrupt is issued when disk IO complete, which causes OS to place the affected process in **Ready** state

IMPLICATIONS OF THIS STRATEGY

- **More processes may be maintained in main memory**
  - Only load in some of the pieces of each process
  - Which pieces? How many pieces? - Critical questions!
  - With many processes in main memory, it is likely that some process will be in Ready state at any particular time (good for performance)
- **Process may be larger than main memory**
  - Virtual memory vs real memory

SUPPORT NEEDED FOR VIRTUAL MEMORY

- Virtual memory usually employed with paging: We will focus on that here.
- Hardware must support paging
- OS must include software for managing the movement of pages between secondary memory and main memory
- All pages of the process will be in secondary memory
- Some pages are also loaded to main memory
- Page tables need to be augmented to include
  - **Present bit**: Is the page in main memory?
  - **Modify bit**: Has the page been modified? If not, it is not necessary to write it to secondary memory when it is replaced in main memory
  - Other control bits may also be present (protection, sharing)

ADDRESS TRANSLATION WITH PAGING VIRTUAL MEMORY

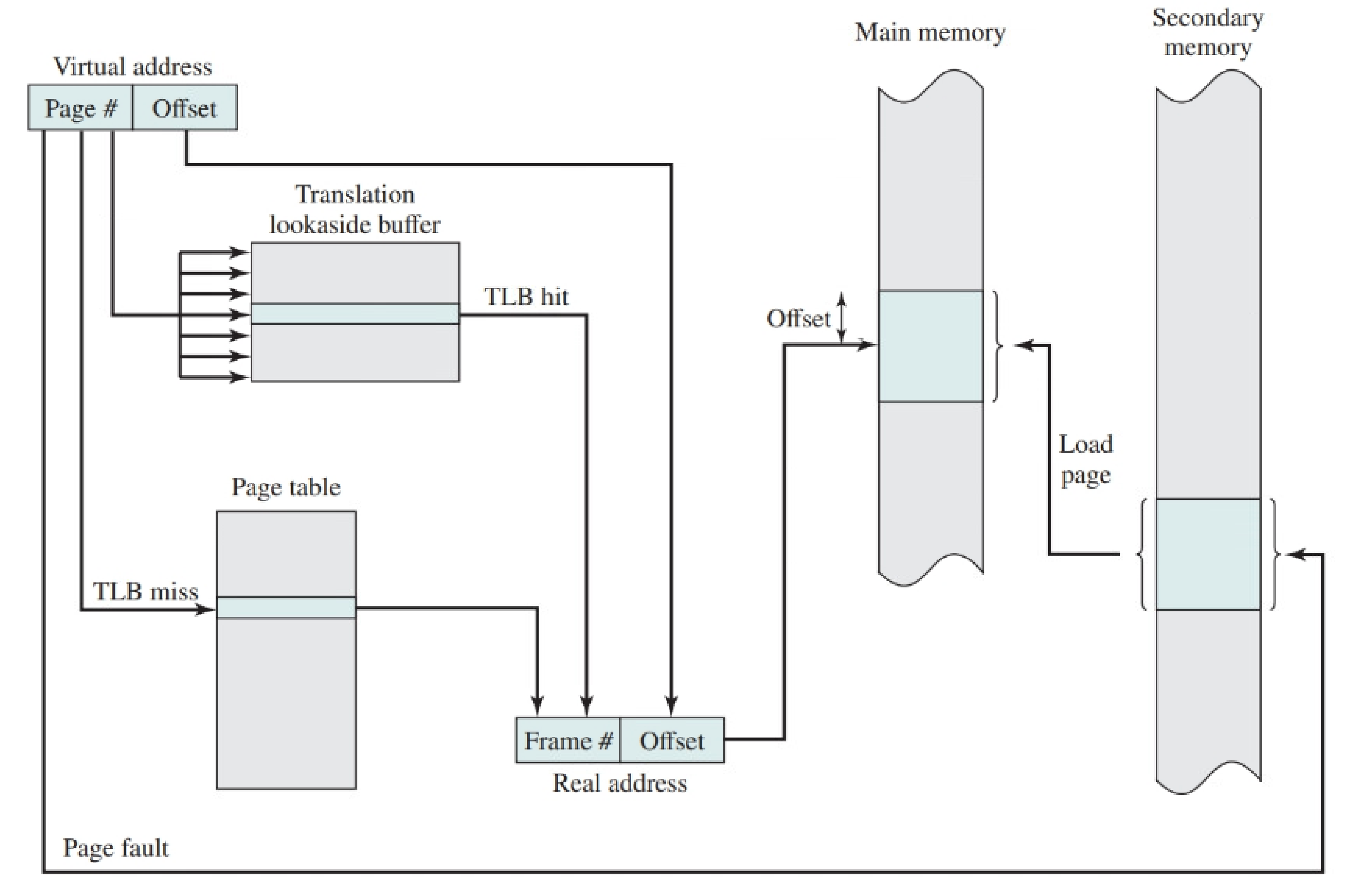


Figure 8.6 **Use of a Translation Lookaside Buffer**

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

THRASHING AND PRINCIPLE OF LOCALITY

- **Thrashing**: system spends most of its time swapping process pieces rather than executing instructions
  - To avoid this, OS needs to guess (based on recent history) which pieces of virtual memory are least likely to be used in the near future and how much memory a process needs
- Key idea: **Principle of locality** (both **temporal** and **spatial**)
  - Code and data references within a process tend to cluster
  - Only a few pieces of a process will be needed over a short period of time
  - Possible to make intelligent guesses about which pieces will be needed in the future

POLICIES FOR VIRTUAL MEMORY

- Key issue performance: minimize page faults

Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy	**Resident Set Management**
Demand paging	Resident set size
Prepaging	Fixed
	Variable
Placement Policy	Replacement Scope
	Global
Replacement Policy	Local
Basic Algorithms	
Optimal	**Cleaning Policy**
Least recently used (LRU)	Demand
First-in-first-out (FIFO)	Precleaning
Clock	
Page Buffering	**Load Control**
	Degree of multiprogramming

Table from [Stallings, Operating systems: Internals and design principles, 9th ed]

FETCH POLICY

- Decides when a page is brought into main memory
- **Demand paging**:
  - Only bring pages into main memory when a reference is made to a location on the page
  - Lots of page faults when a process is started
  - Principle of locality suggests that after some pages have been brought into main memory, most future references are to pages already in main memory – **page faults should drop to a very low level**
- **Prepaging**:
  - Also pages other that the one causing a page fault are brought into main memory
  - Exploits characteristics of most secondary storage devices: If pages of a process are stored contiguously in secondary memory, efficient to bring in several pages at a time
  - Inefficient if extra pages are not referenced

REPLACEMENT POLICY

- **Selection of a page in main memory to be replaced when a new page must be brought in**
- **Page** that is removed should be the one that is **least likely to be referenced in the near future**
- The more elaborate the replacement policy, the greater the hardware and software overhead to implement it
- **Frame locking**:
  - When a frame is locked, the page currently stored in that frame cannot be replaced
  - Kernel and key control structures are held in locked frames
  - IO buffers and time-critical areas may be locked

REPLACEMENT POLICY: ALGORITHMS

- Optimal
- Least recently used (LRU)
- First-in-first-out (FIFO)
- Clock
- Random

- **Optimal** policy selects the page for which the time to next reference is the longest
- This policy leads to the fewest number of page faults
- **Impossible to implement** (we do not know the future!)
- Serves as a baseline against which to judge real-world algorithms

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT												

Figure 8.14 from [Stallings, Operating systems: Internals and design principles, 9th ed]

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

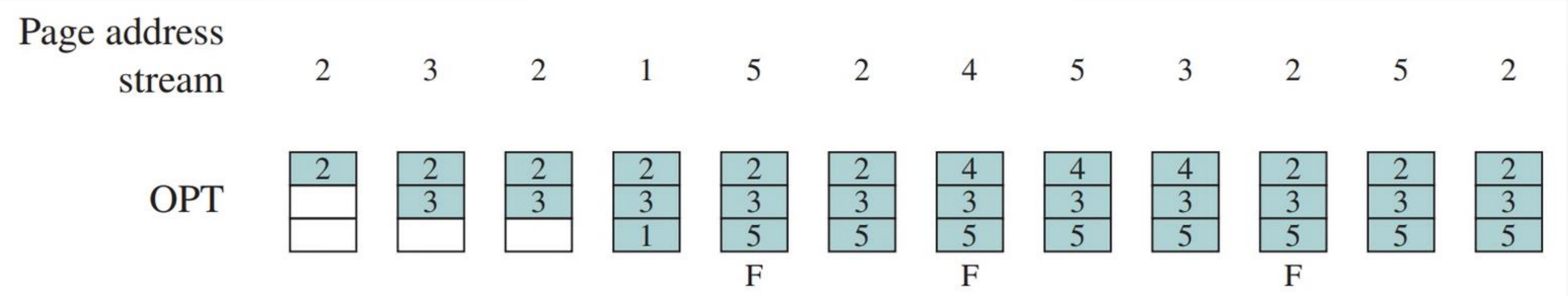


Figure 8.14 from [Stallings, Operating systems: Internals and design principles, 9th ed]

LEAST RECENTLY USED (LRU)

- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page that is least likely to be referenced in the near future
- Advantage: Close to the performance of optimal policy
- Disadvantage: Difficult to implement
  - Tag each page with the time of last reference (lots of overhead)

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2 3	2 3	2 3 1	2 3 5 F	2 3 5	4 3 5 F	4 3 5	4 3 5	2 3 5 F	2 3 5	2 3 5
LRU												

Figure 8.14 from [Stallings, Operating systems: Internals and design principles, 9th ed]

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					F		F		F	F		

Figure 8.14 from [Stallings, Operating systems: Internals and design principles, 9th ed]

FIRST-IN-FIRST-OUT (FIFO)

- Treats allocated page frames as a circular buffer
- Pages are replaced in a round-robin style
- Page that has been the longest in main memory is replaced
- **Advantage**: Simple to implement
- **Disadvantage**: Performs poorly – often there are regions of a code or data that are referenced frequently throughout the lifetime of a process, these will be repeatedly paged in and out of main memory

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

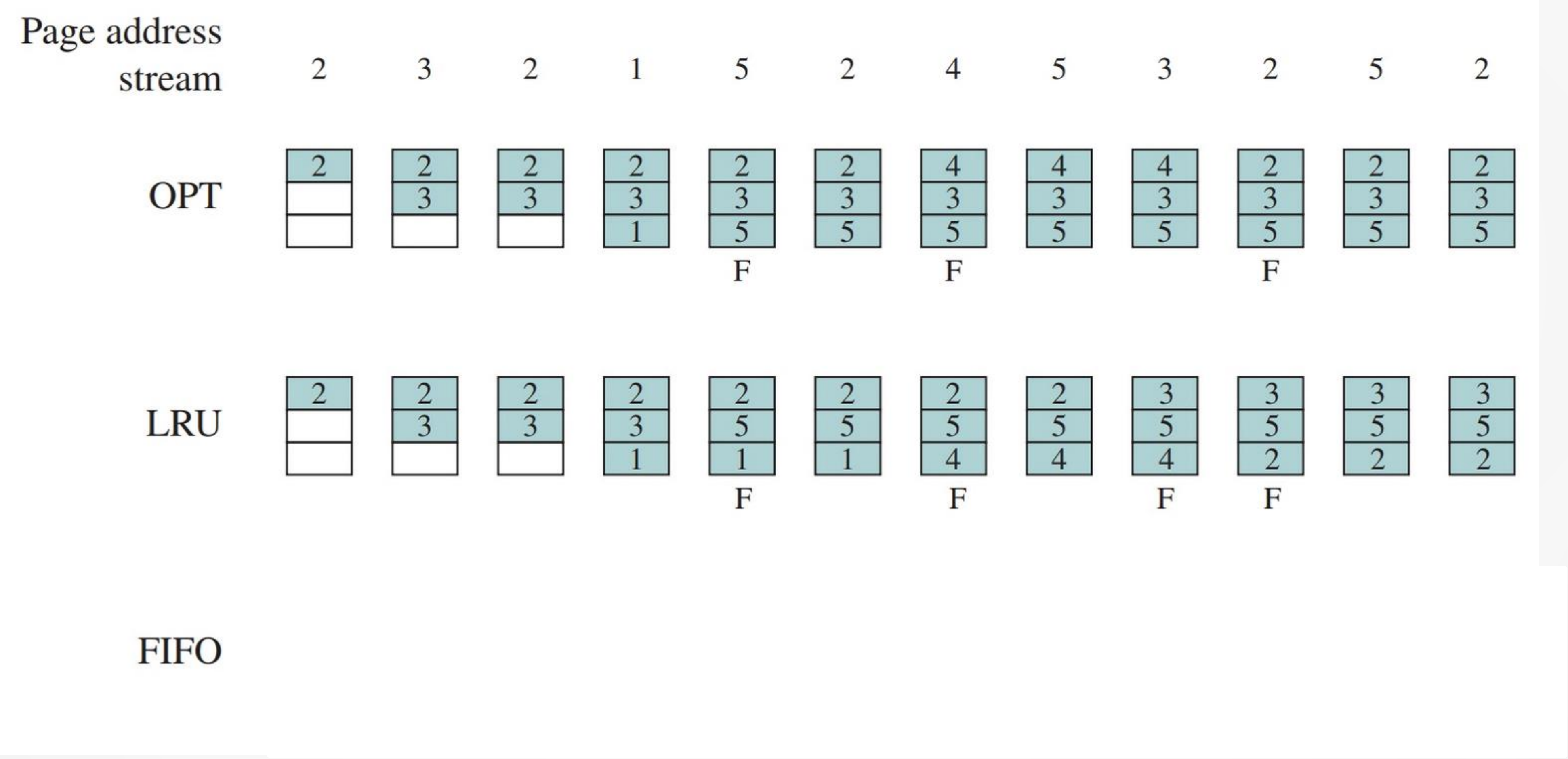


Figure 8.14 from [Stallings, Operating systems: Internals and design principles, 9th ed]

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

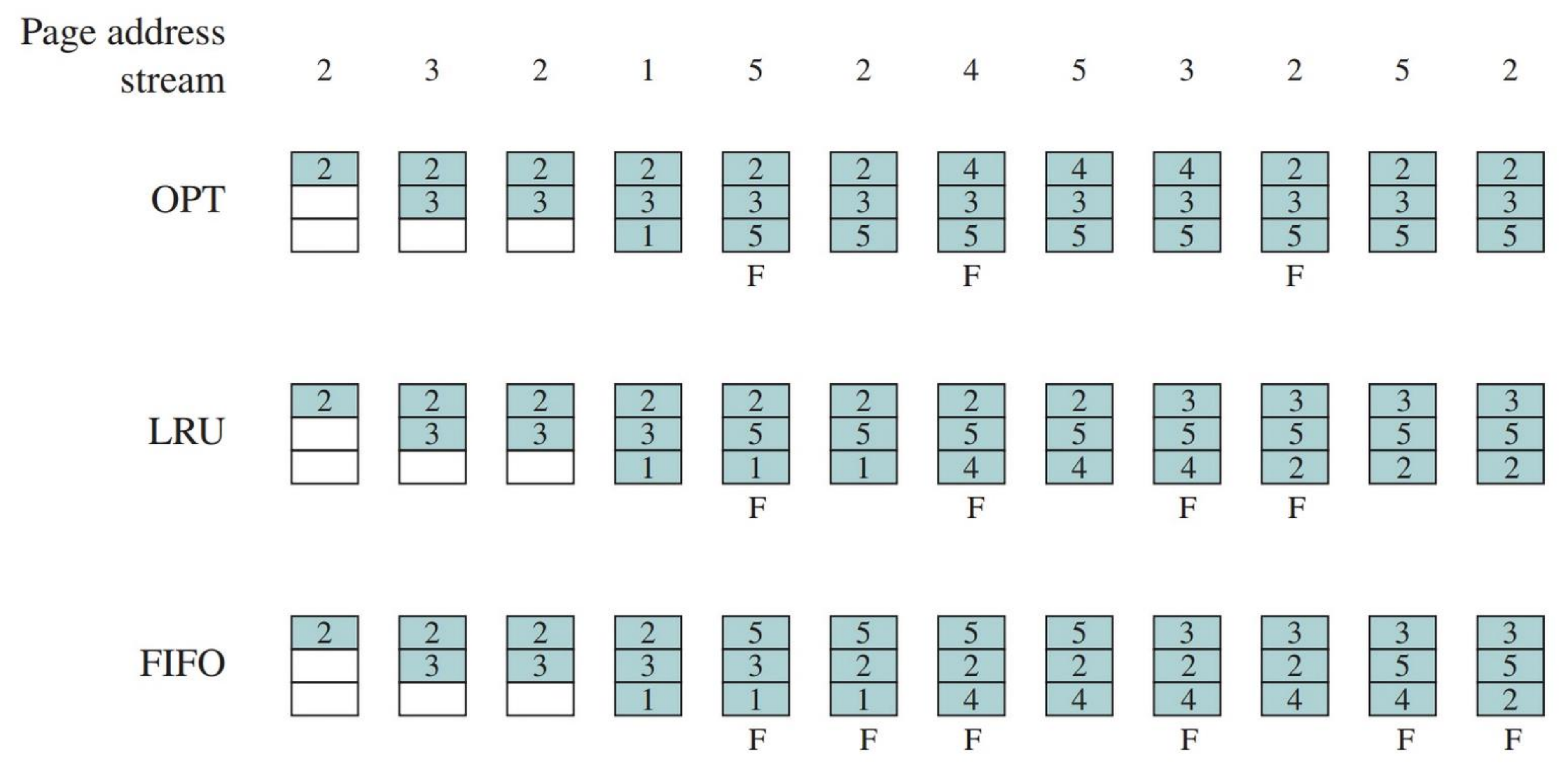
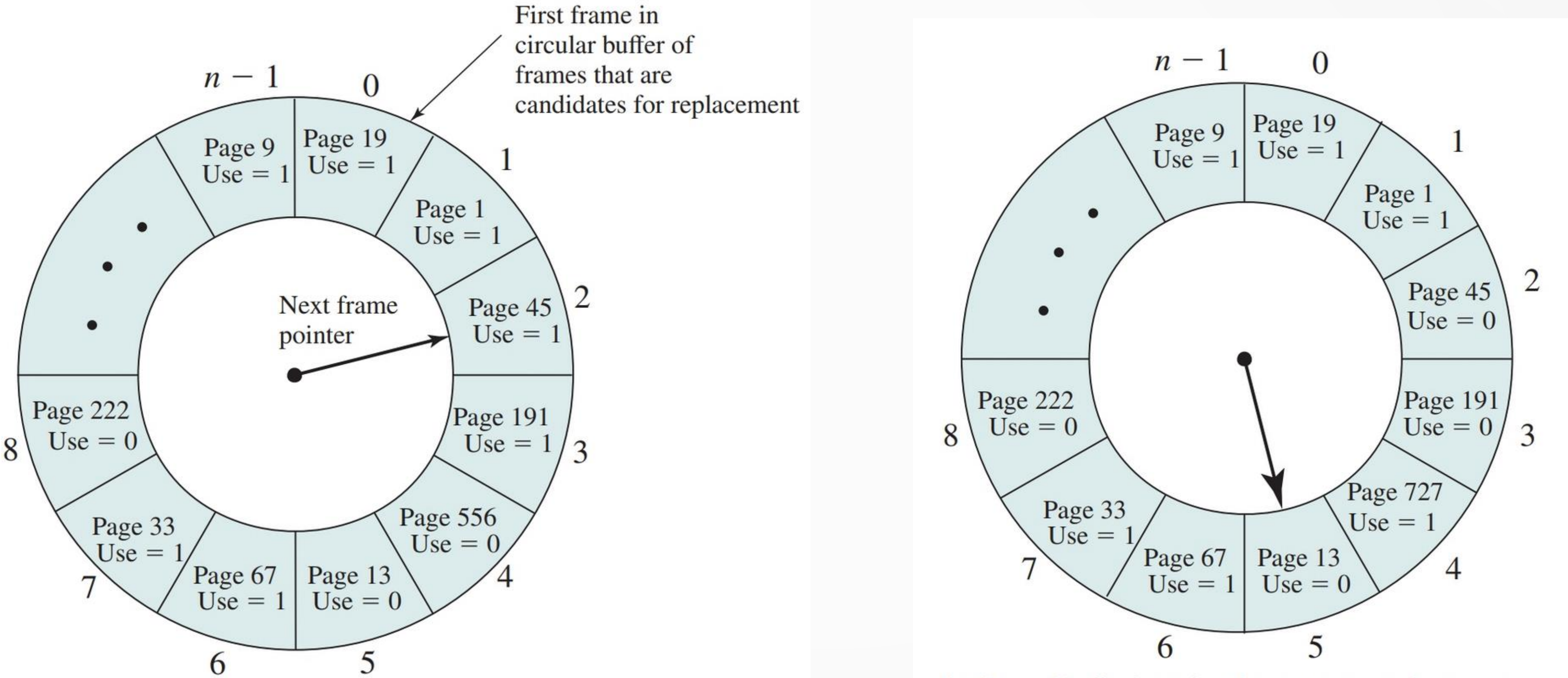


Figure 8.14 from [Stallings, Operating systems: Internals and design principles, 9th ed]

CLOCK POLICY

- Requires the association of an additional **use bit** with each frame
- When page is loaded into main memory, the use bit is set to 1
- When a page is referenced the use bit is set (if not already set)
- The set of main memory frames is considered as a circular buffer, frames visualized as laid out in a circle
- The algorithm keeps track of the last replaced frame
- When a page for replacement needs to be found
    - Start looking for a frame from the frame following the last replaced frame
    - If a frame with use bit 0 is found, select that frame for replacement
    - Any frame with use bit 1 is passed over by the algorithm and the use bit of that frame is set to 0

CLOCK POLICY: EXAMPLE



(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

Figure 8.15 Example of Clock Policy Operation

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

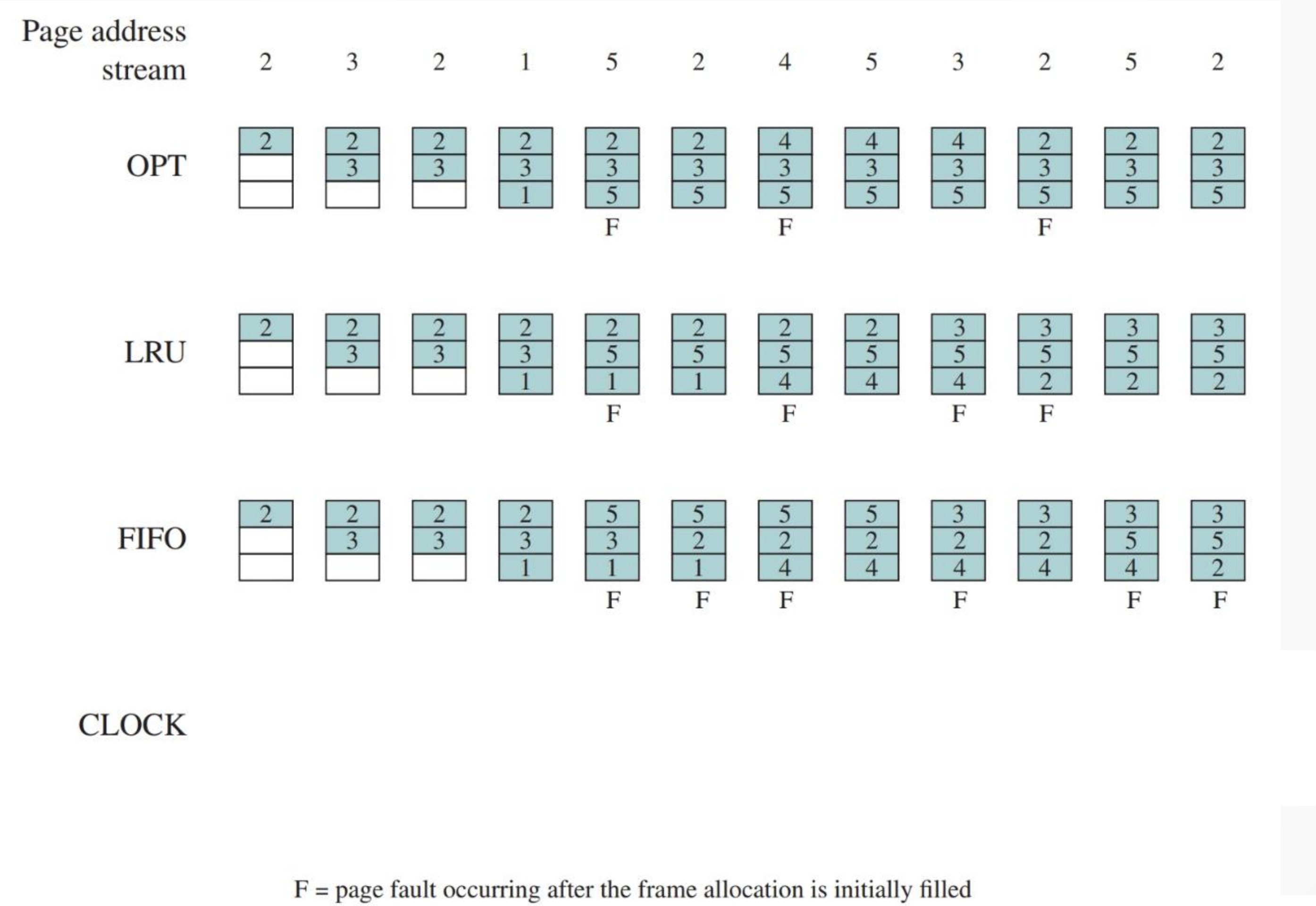


Figure 8.14 **Behavior of Four Page Replacement Algorithms**

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

EXAMPLE: COMPARISON OF REPLACEMENT ALGORITHMS

Figure 8.14 **Behavior of Four Page Replacement Algorithms**

Figure from [Stallings, Operating systems: Internals and design principles, 9th ed]

RESIDENT SET MANAGEMENT

- **OS must decide how many pages to bring into main memory**
  - The smaller the amount of memory allocated to each process, the more processes can reside in memory
  - Small number of pages loaded increases page faults
  - Beyond a certain size, further allocated pages will not affect the page fault rate
- **Fixed allocation**: gives a process a fixed number of frames in main memory
  - When a page fault occurs, one of the pages of that process will be replaced
- **Variable allocation**: number of pages allocated to a process can vary over the lifetime of a process

REPLACEMENT SCOPE

- Replacement activated by a page fault when there are no free frames left
- Scope of replacement:
  - **Global**: consider all unlocked pages in main memory
  - **Local**: choose only among resident pages of the process that generated the page fault
- While local policies are easier to analyze, there is no convincing evidence that they perform better than global ones
- Global policies are attractive because of simplicity of implementation and minimal overhead

CLEANING POLICY

- **When should a modified page be written out to secondary memory?**
- **Demand cleaning**
  - Page written out to secondary memory only when it has been selected for replacement
- **Precleaning**
  - Write out modified pages before their page frames are needed so that pages can be written out in batches

SUMMARY

- Memory management and virtual memory is one of the most important and complex tasks of the OS
- Virtual memory:
  - All address references are virtual addresses that are translated at run time to physical addresses
  - Part of the processes may reside in main memory, part in secondary memory
- Implementation requires both hardware (e.g. TLB, address translation) and software support (different virtual memory policies)