University of Nottingham
UK | CHINA | MALAYSIA

Operating Systems and Concurrency

Lecture 16: Memory Management 2

University of Nottingham,
Ningbo China 2024

Recall
Fixed Partitioning

- It is useful to have multiple processes in memory to **maximise CPU utilisation**: mono-programming ⇒ multi-programming
- Rather than allocating the full physical memory to one process, split it into **(non-)equal sized partitions** and **allocate a process to each partition**
- Fixed **equal sized partitions** result in **internal fragmentation**, **non-equal sized partitions** make **allocation** more difficult

Memory management has been evolved

- Approches
  - **Contiguous Memory**
    - ❖ **Mono-programming** [easy but low resource utilisation, cannot be used for modern multiprogramming machine]

      No relocation, physical address
    - ❖ **Multiprogramming** with **fixed partition** [multi-programming but internal fragmentation]

      Relocation (register), logical address
    - ❖ **Multiprogramming** with **dynamic partition** [low internal fragmentation but high external fragmentation]

      Relocation (register), logical address
  - Non-Contiguous Memory

    Relocation (table), logical address
    - ❖ **Paging** (fixed-partitioning/code re-location) [Internal fragmentation reduce to last block, no external fragmentation; Require page table to maintain page relocation]
    - ❖ **Virtual Memory** (locality →not all pages loaded) [more processes → CPU utilisation, no external fragmentation, more memory available; Need involve replacement algorithm, page table management, thrashing, variable/fixed resident set….]

Overview
Goals for Today

- **Dynamic partitioning**
- **Swapping**
- **Dynamic partitioning management** with Linked lists.

Dynamic Partitioning
context

- **Fixed partitioning** results in **internal fragmentation**:
  - An **exact match** between the requirements of the process and the available partitions **may not exist**
  - The partition may **not be used entirely**

- **Dynamic partitioning:**
  - A **variable number of partitions** of which the **size** and **starting address** **can change** over time
  - A process is allocated the **exact amount of contiguous memory** it requires, thereby preventing internal fragmentation

Dynamic Partitioning
Example

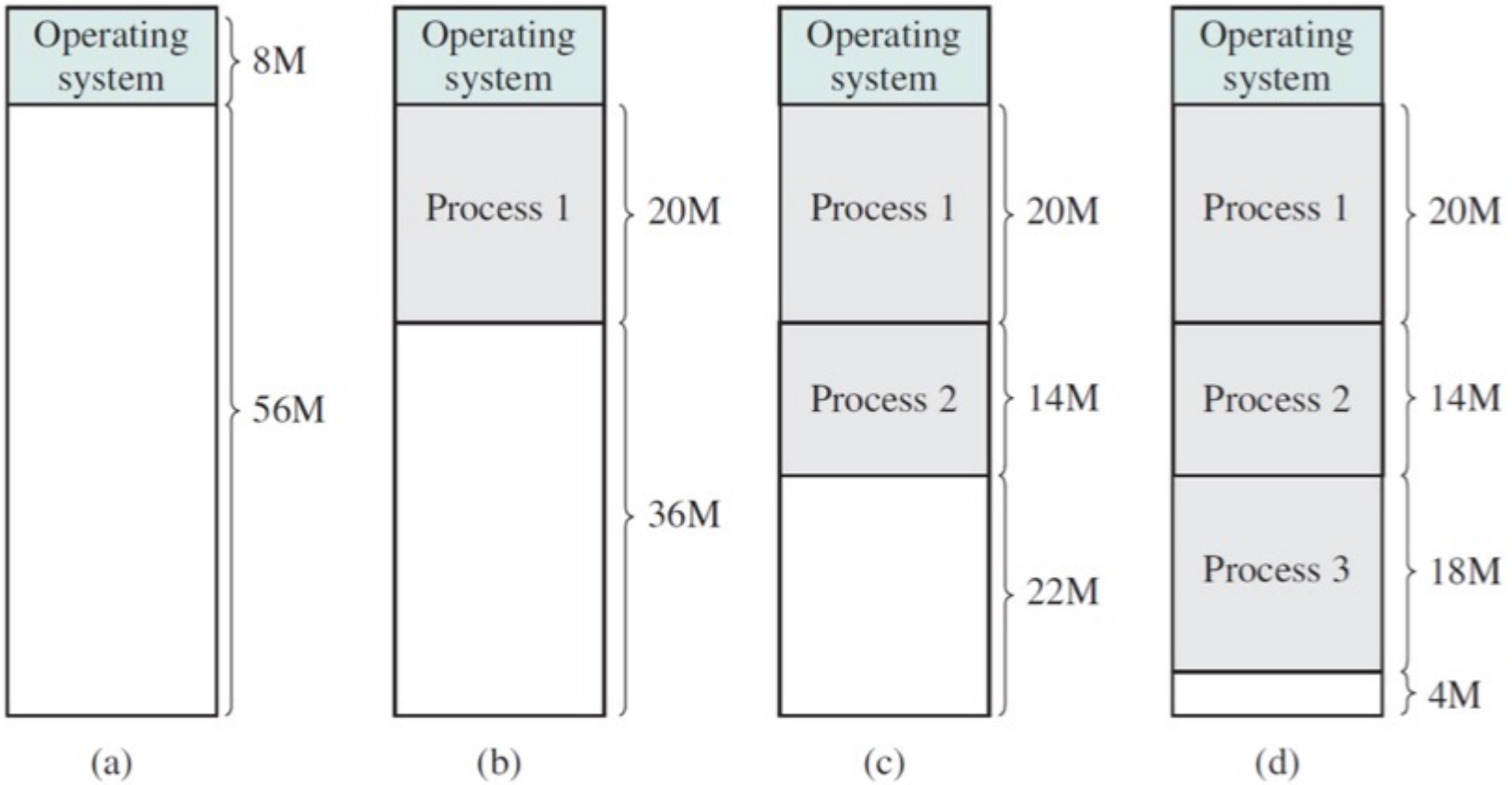


Figure: Dynamic partitioning (from Stallings)

Dynamic Partitioning
Swapping

- **Swapping** holds some of the **processes on the hard disk** and **shuttles processes** between the hard disk and main memory as necessary

- **Reasons** for swapping:
  - Some **processes** only **run occasionally**
  - We have **more processes** than **partitions** (assuming fixed partitions)
  - A process's **memory requirements** have **changed**, e.g. increased
  - The **total amount of memory that is required** for the processes **exceeds the available memory**

Dynamic Partitioning
Swapping: Example

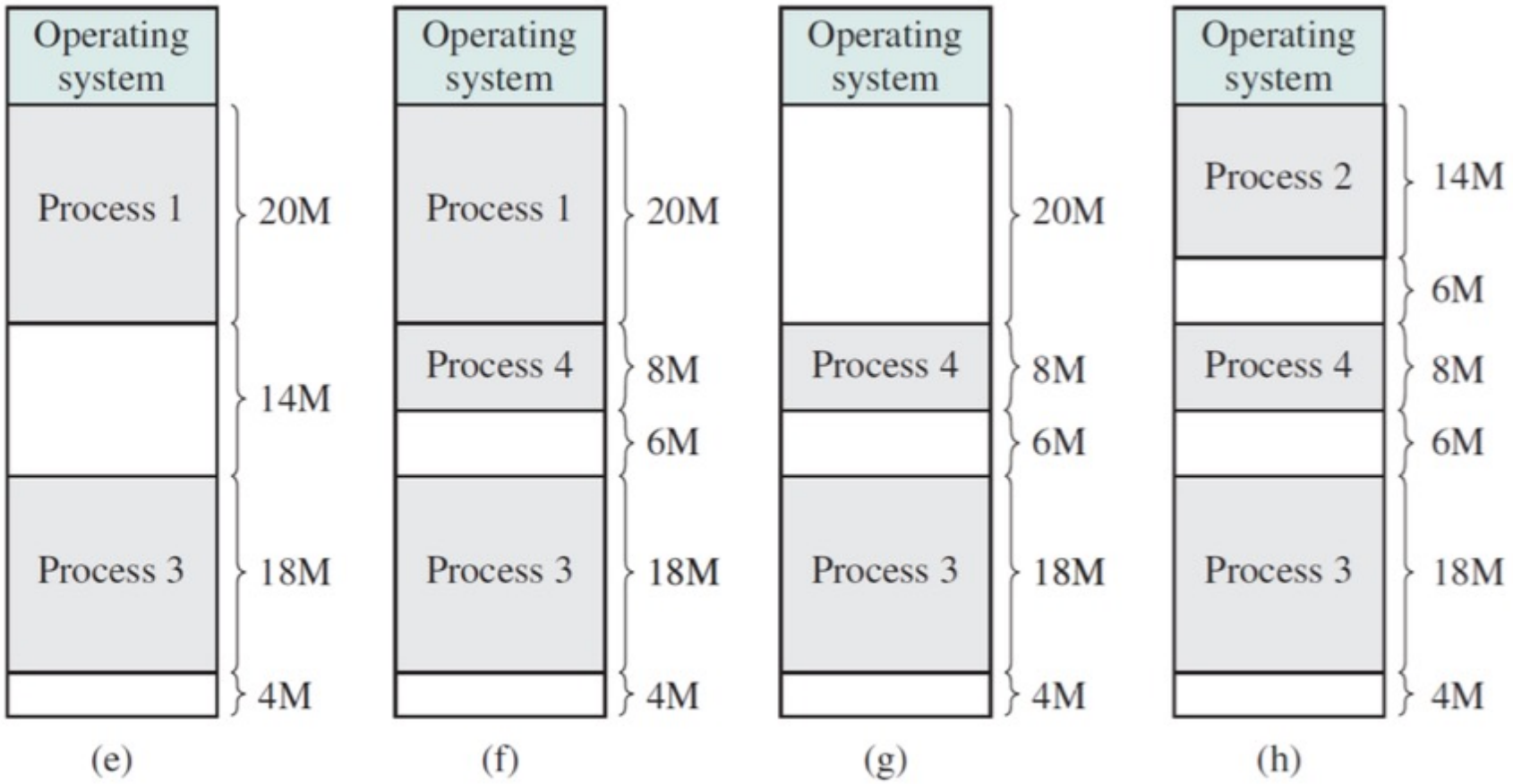


Figure: External fragmentation (from Stallings)

Dynamic Partitioning
Swapping: Questions

- **Memory management** becomes more complicated
- How to keep track of **available memory**
  - Bitmaps
  - Linked lists
- What strategies can I use to (quickly) **allocate** processes to available memory (“holes”)

Dynamic Partitioning
Allocation Structure: Bitmaps

- The simplest data structure to **keep track of free memory** is a form of **bitmap**
- **Memory** is **split into blocks** of say 4 kilobyte (4Kb) size
  - A bit map is set up so that each **bit is 0** if the **memory block is free** and **1** if the **block is used**, e.g.
    - 32Mb memory = (32 * 2^{20} bytes / 4K bytes) blocks => 8192 bitmap entries
    - 8192 bits occupy 8192 / 8 = 1K bytes of storage
  - The size of this bitmap will depend on the **size of the memory** and the **size of the allocation unit**.

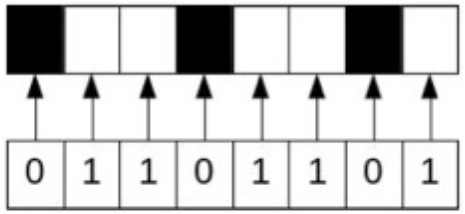


Figure: Memory management with bitmaps

Dynamic Partitioning
Allocation Structure: Bitmaps (Cont'ed)

- To find a hole of e.g. size 128K, then a group of **32 adjacent bits set to zero** must be found, typically a **long operation** (esp. with smaller blocks)

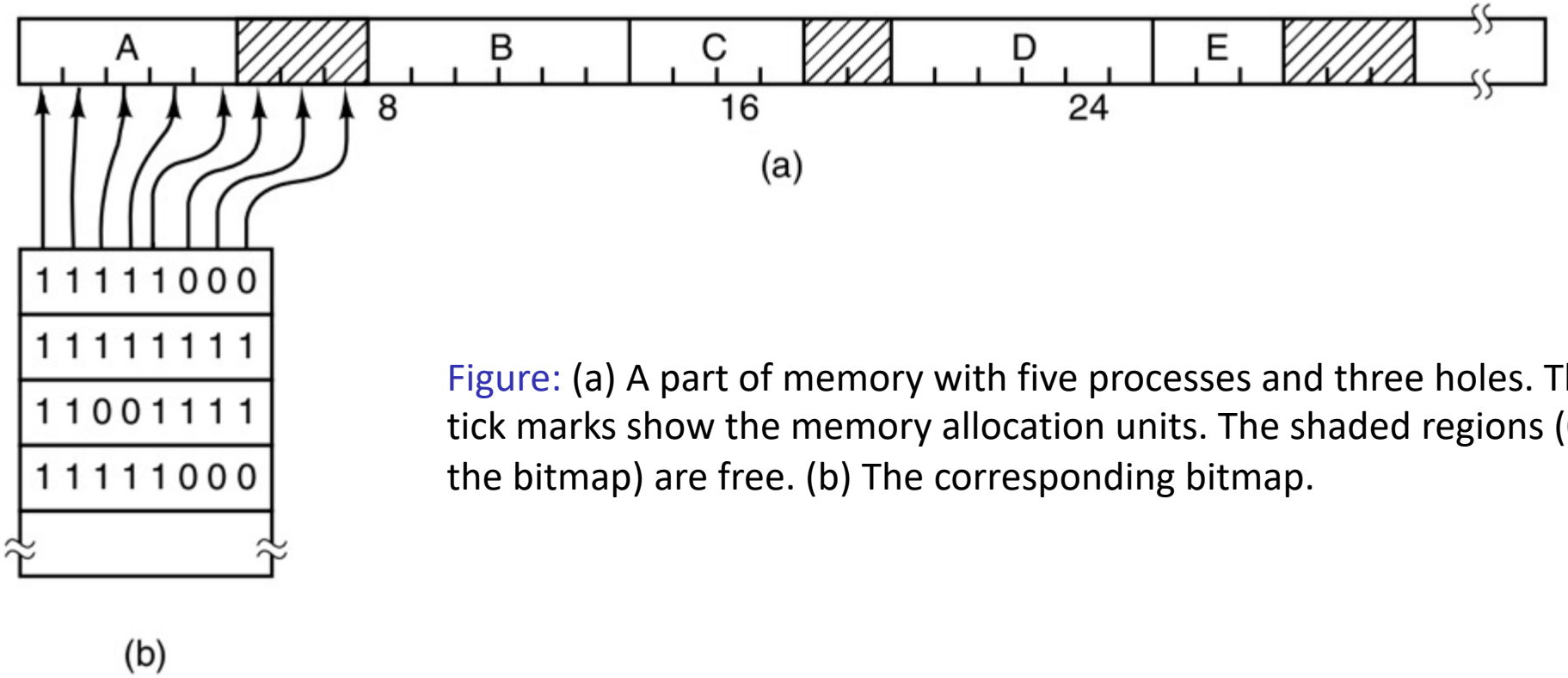


Figure: (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap.

Dynamic Partitioning
Allocation Structure: Bitmaps (Cont'ed)

- A **trade-off exists** between the **size of the bitmap** and the **size of allocation unit (block)** exists
  - The **size of bitmaps** can become prohibitive for small blocks and may **make searching** the bitmap **slower**
  - Larger blocks may increase **internal fragmentation**
- **Bitmaps** are **rarely used** for this reason

Dynamic Partitioning
Allocation Structures : Linked List

- A more **sophisticated data structure** is required to deal with **a variable number** of **free and used partitions**
- A **linked list** is one such possible data structure
  - A linked list consists of a **number of entries** (“links”!)
  - Each link **contains data items**, e.g. **start of memory block**, **size**, free/allocated **flag**
  - Each link also contains a **pointer to the next** in the chain

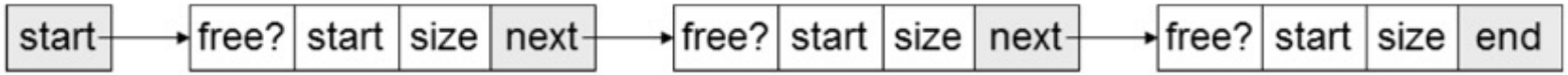
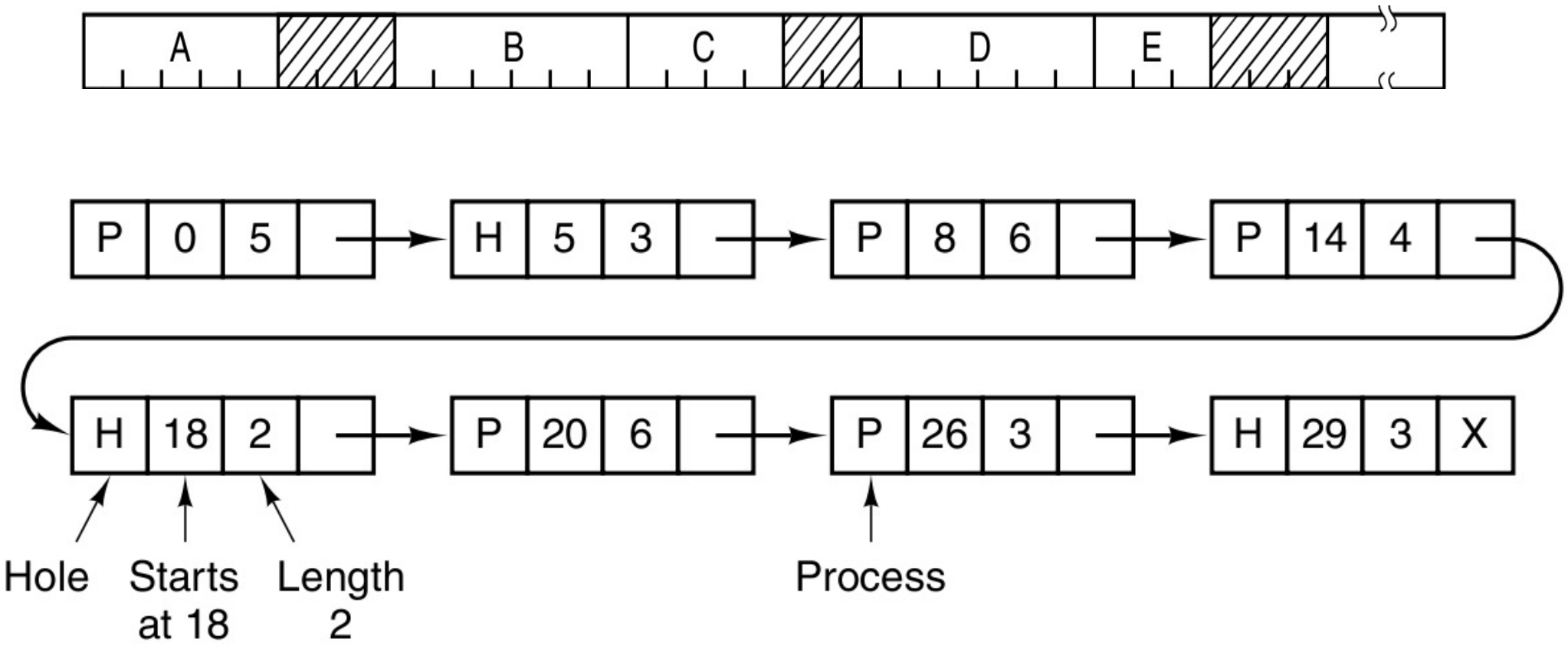
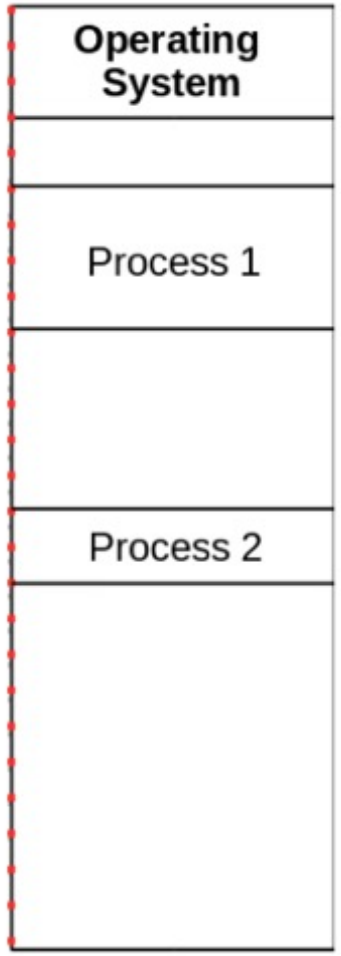


Figure: Memory management with linked lists

Dynamic Partitioning
Linked list

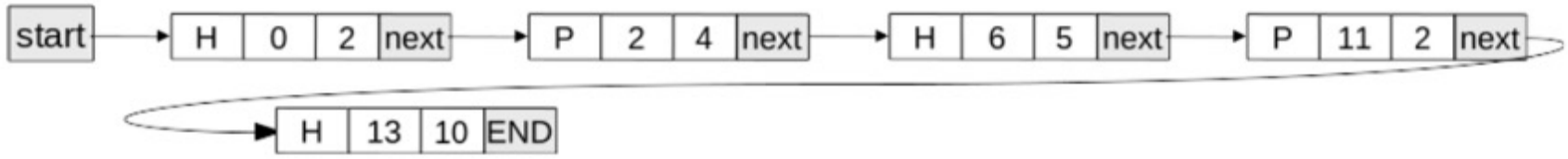


Dynamic Partitioning
Memory Management

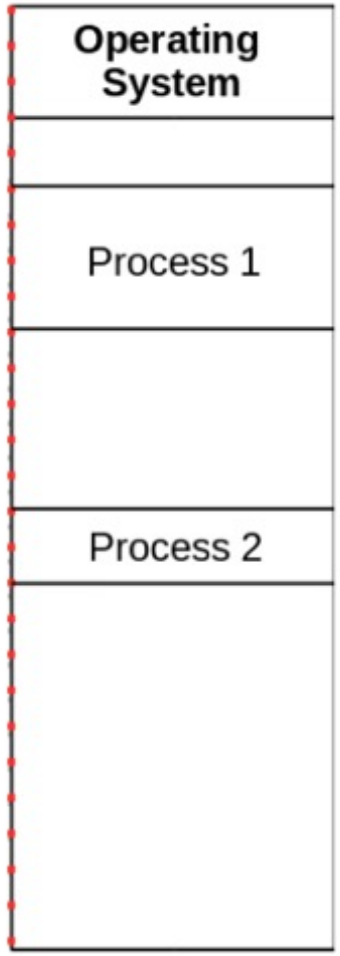


- How to keep track of **available memory**
  - Bitmaps
  - **Linked lists**

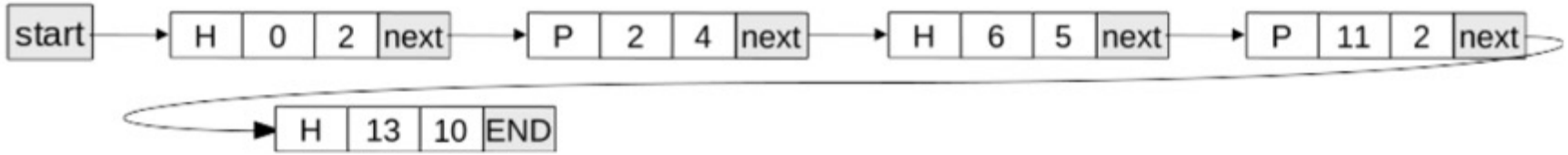
- The operating system is responsible for:
  - Applying strategies to (quickly) **allocate processes** to available memory (“holes”)
  - Managing **free space**



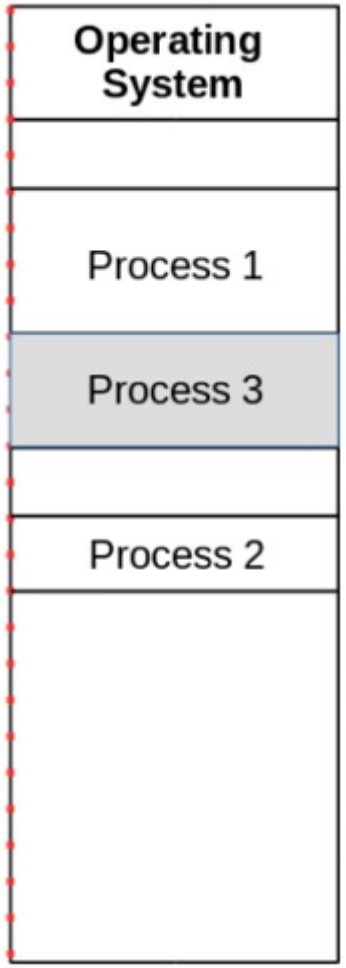
Dynamic Partitioning
Allocating Available Memory: Algorithms



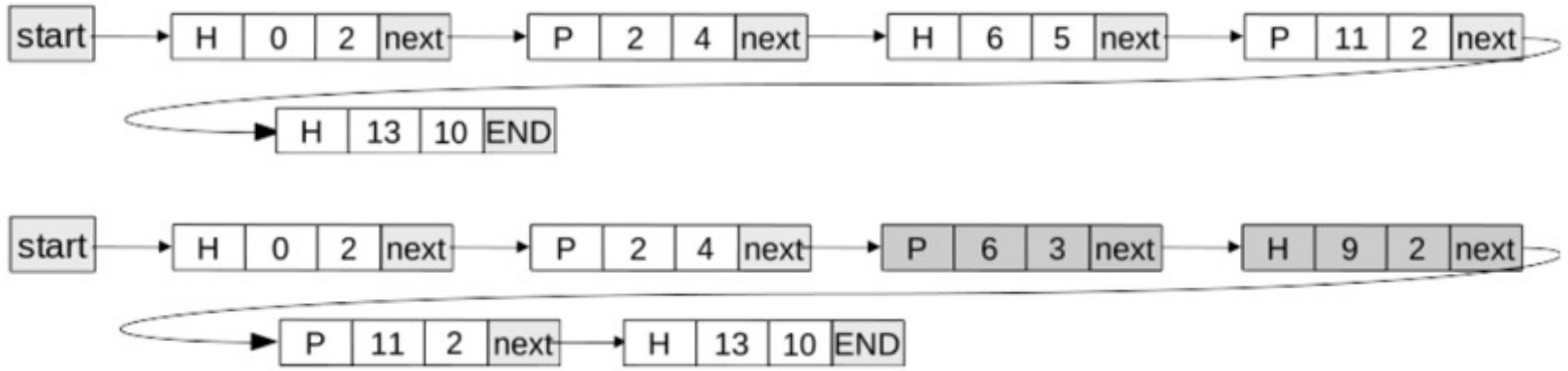
- Algorithms to (quickly) **allocate processes** to available memory ("holes")
  - First Fit
  - Next Fit
  - Best Fit
  - Worst Fit
  - Quick Fit



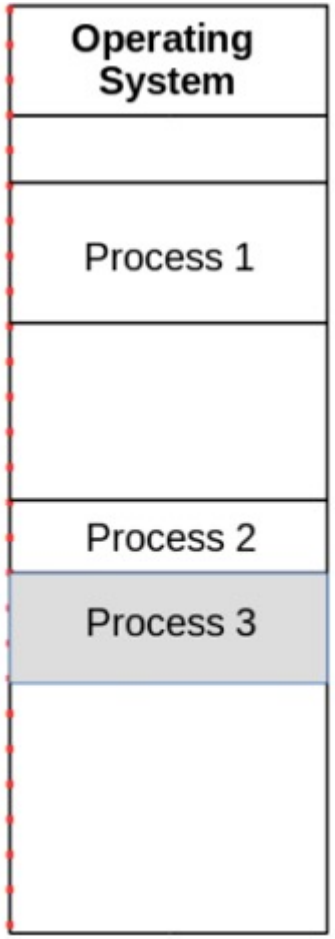
Dynamic Partitioning
Allocating Available Memory: First Fit



- **First fit** starts scanning **from the start** of the linked list until a link of **sufficient free size** is found
  - If requested space is **the exact same size** as the "hole", all the space is allocated (i.e., no **internal** fragmentation)
  - Else, the free link is **split into two**:
    - The first entry is set to the **size requested** and marked "**used**"
    - The second entry is set to **remaining size** and marked "**free**"

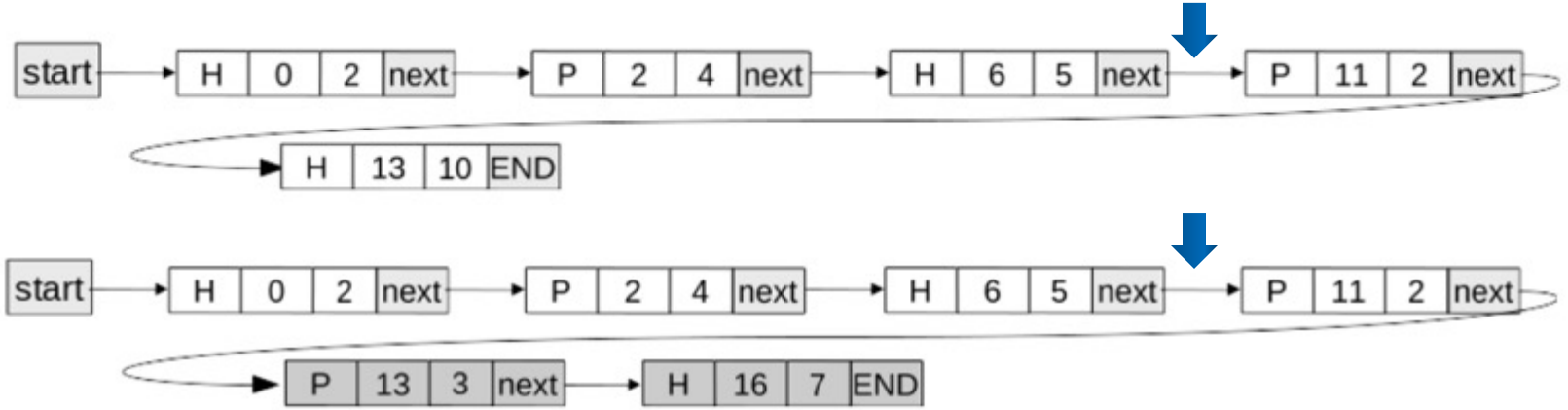


Dynamic Partitioning
Allocating Available Memory: Next Fit

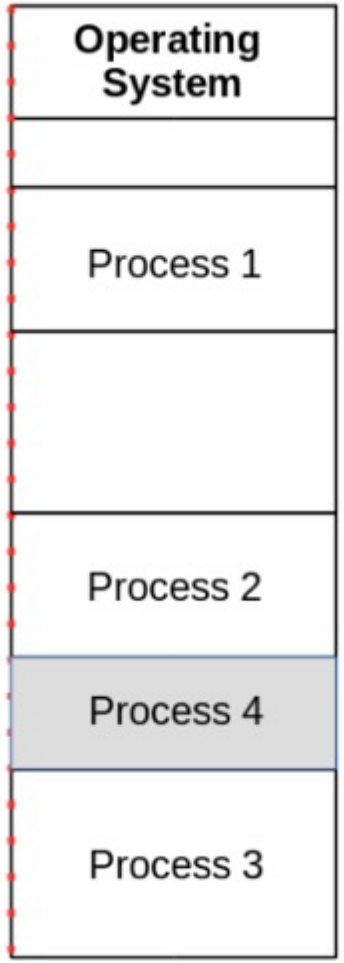


- The **next fit algorithm** maintains a **record** of where it got to:
  - It **restarts** **its search** from **where it stopped last time**
  - It gives an **even chance to all memory to get allocated** (first fit concentrates on the start of the list)

- However, simulations have shown that next fit actually gives **worse performance** than first fit!

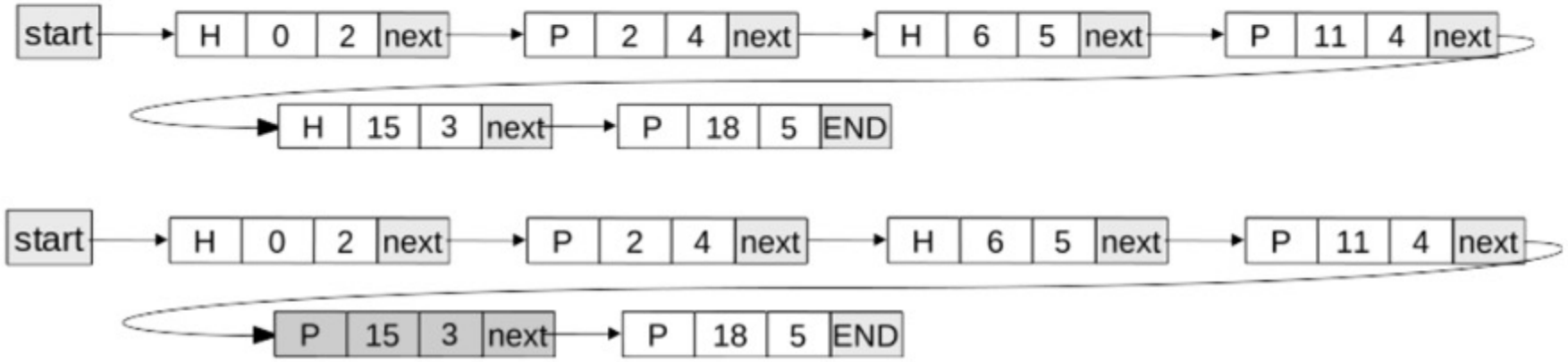


Dynamic Partitioning
Allocating Available Memory: Best Fit

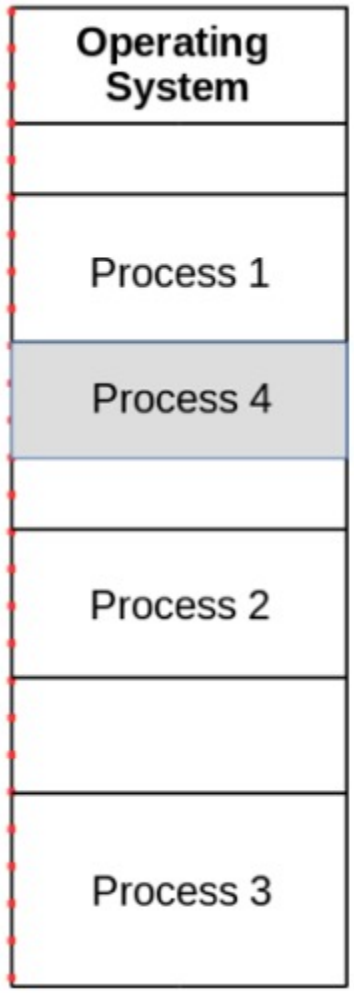


- **First fit** just looks for the **first available hole**
  - It doesn't take into account that there may be **a hole later** in the list that **exactly(-ish)** fits the requested size
  - First fit **may break up a big hole** when the right size hole exists later on
- **Next Fit** doesn't improve that model. What else can we do?

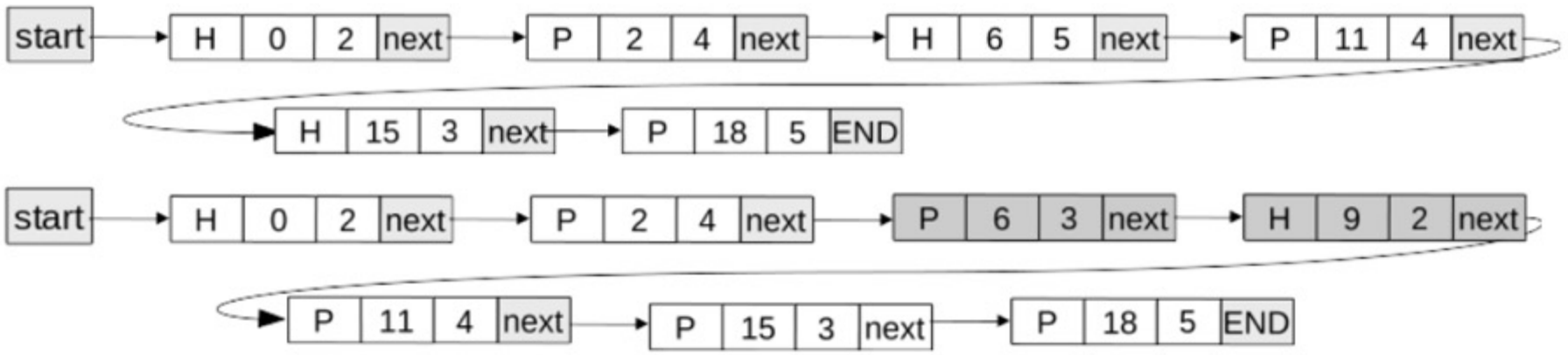
- The **best fit algorithm** always **searches the entire** linked list to find **the smallest hole big enough** to satisfy the memory request
  - It is **slower** than first fit
  - It also results in **more wasted memory** (memory is **more likely to fill up with tiny - useless – holes**, first fit generates larger hoes on the average)



Dynamic Partitioning
Allocating Available Memory: Worst Fit



- **Tiny holes** are created when **best fit** split an empty partition
- The **worst fit algorithm** finds the **largest available empty partition** and splits it
  - The **left over part will still be large** (and **potentially more useful**)
  - Simulations have also shown that worst fit is **not very good either**!



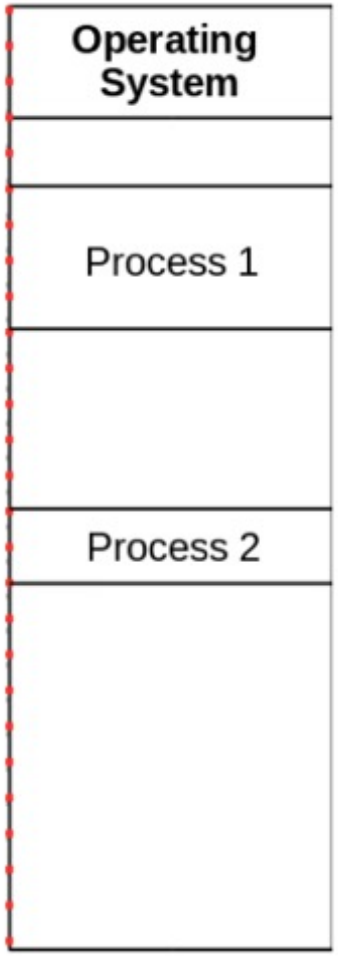
Dynamic Partitioning
Allocating Available Memory: Summary

- **First fit**: allocate **first block** that is **large enough**
- **Next fit**: allocate **next block** that is large enough, i.e. **starting from the current location**
- **Best fit**: choose block that **matches** required size **closest** - O(N) complexity
- **Worst fit**: choose the **largest possible block** - O(N) complexity

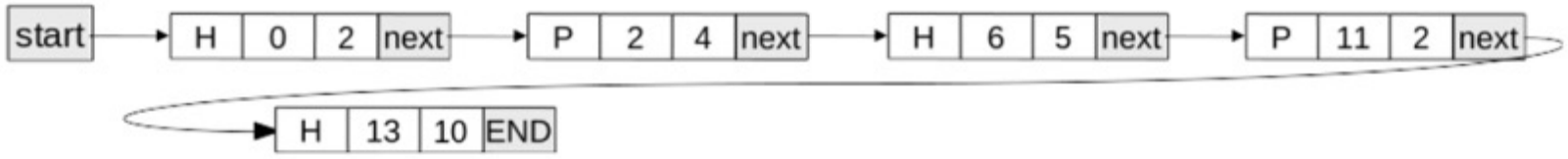
Dynamic Partitioning
Allocating Available Memory: Quick Fit and Others

- **Quick fit** maintains **lists of commonly used sizes**
  - For example a separate list for each of 4K, 8K, 12K, 16K, etc., holes
  - **Odd sizes** can either go into the **nearest size** or into a **special separate list**
- It is **much faster** to find the required size hole using **quick fit** (table of entries, entries point to head of holes)
- Similar to next fit, it has the problem of creating **many tiny holes**
- Finding neighbours for **coalescing** (combining empty partitions) becomes more difficult/time consuming

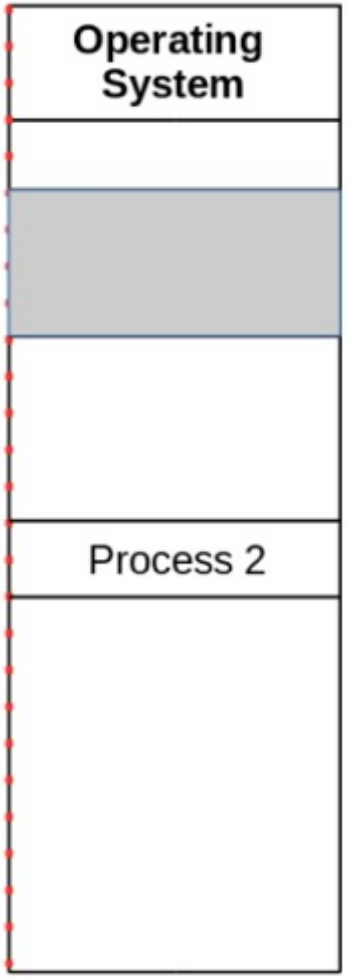
Dynamic Partitioning
Managing Available Memory: Coalescing



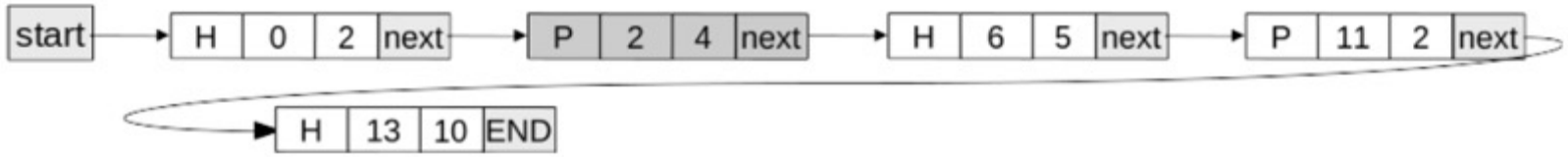
- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
  - There may be three adjacent free entries if a used block in-between two free blocks is freed
- Both **neighbours** are examined when a **block is freed**
  - If either (or both) are **also free**
  - Then the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted



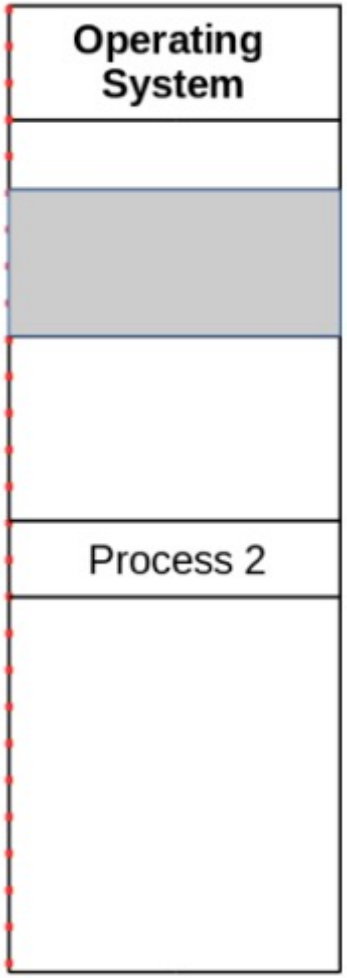
Dynamic Partitioning
Managing Available Memory: Coalescing



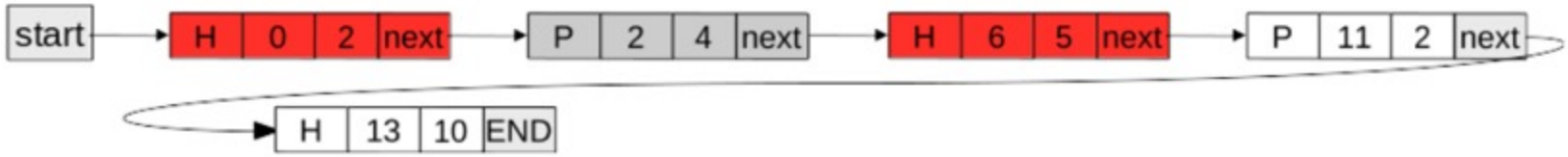
- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
  - There may be three adjacent free entries if a used block in-between two free blocks is freed
- Both **neighbours** are examined when a **block is freed**
  - If either (or both) are **also free**
  - Then the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted



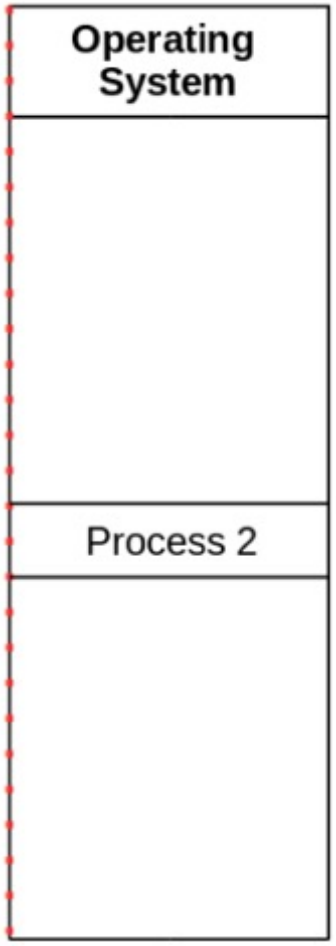
Dynamic Partitioning
Managing Available Memory: Coalescing



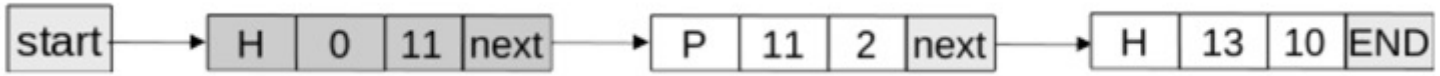
- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
  - There may be three adjacent free entries if a used block in-between two free blocks is freed
- Both **neighbours** are examined when a **block is freed**
  - If either (or both) are **also free**
  - Then the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted



Dynamic Partitioning
Managing Available Memory: Coalescing

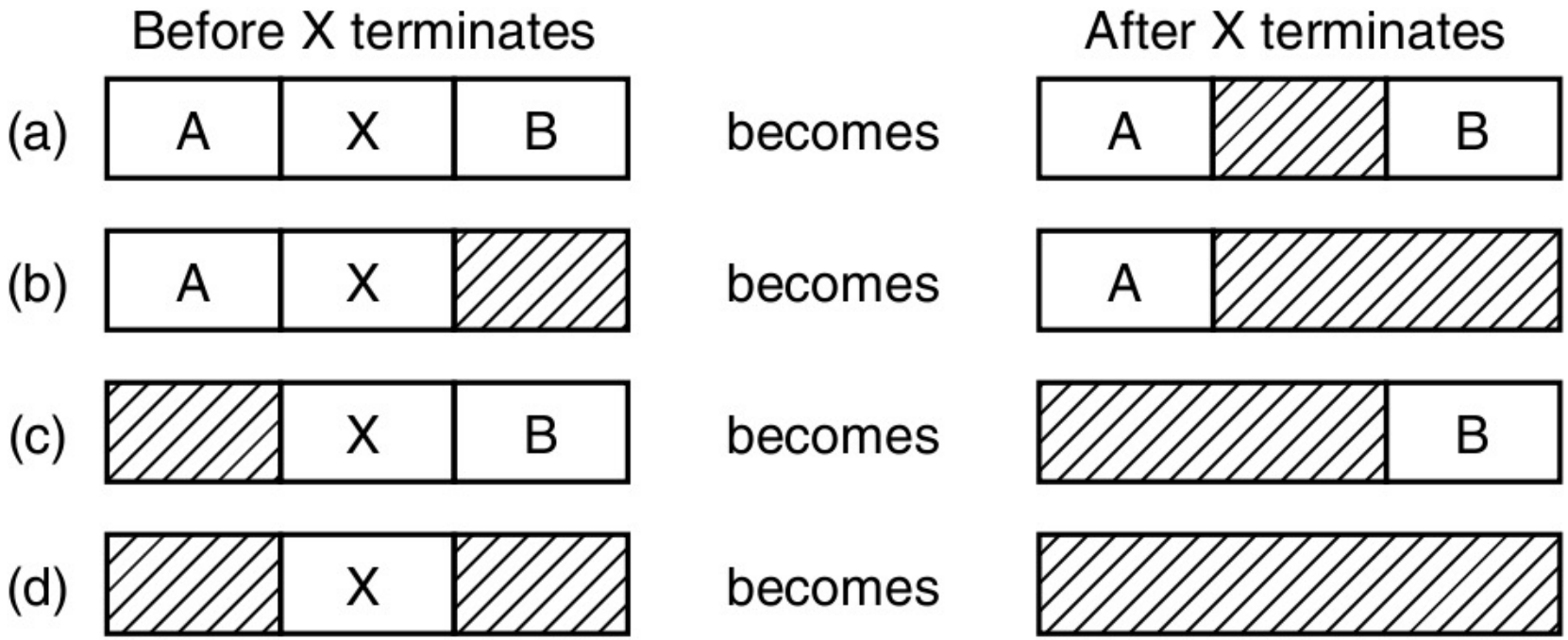


- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
  - There may be three adjacent free entries if a used block in-between two free blocks is freed
- Both **neighbours** are examined when a **block is freed**
  - If either (or both) are **also free**
  - Then the two (or three) **entries are combined** into one larger block by adding up the sizes
    - The earlier block in the linked list gives the **start point**
    - The **separate links are deleted** and a single link inserted



Dynamic Partitioning

Managing Available Memory: Coalescing



Dynamic Partitioning
Managing Available Memory: Compacting

- Even with coalescing happening automatically, **free blocks** may still **distributed across memory**
  - ⇒ **Compacting** can be used to join free and used memory (but is **time consuming**)

- **Compacting is more difficult and time consuming** to implement than coalescing (processes have to be moved)
  - Each process is **swapped out** & **free space coalesced**
  - Process swapped back in at lowest available location

Dynamic Partitioning
Difficulties

- The exact **memory requirements** may **not be known** in advance (**heap** and **stack** grow dynamically)
  - => Allocate the current requirements + “a bit extra”?

Figure: Memory organisation of a process

Dynamic Partitioning
Difficulties

- **External fragmentation**:
  - **Swapping** a process out of memory will create "**a hole**"
  - A new process may not use the entire "hole", leaving a small **unused block**
  - A new process may be **too large** for a given a "hole"

- The **overhead** of memory **compaction** to **recover holes** can be **prohibitive** and requires **dynamic relocation**
  - Requires a lot of CPU time

Contiguous Allocation Schemes
Overview and Shortcomings

- Different contiguous memory allocation schemes have different advantages/disadvantages
  - **Mono-programming** is easy but does result in **low resource utilisation**
  - **Fixed partitioning** facilitates **multi-programming** but results in **internal fragmentation**
  - **Dynamic partitioning** facilitates **multi-programming**, reduces **internal fragmentation**, but results in **external fragmentation** (allocation methods, coalescing, and compacting help)
- Can we design a memory management scheme that **resolves the shortcomings** of contiguous memory schemes?

Memory management has been evolved

- Approches
  - Contiguous Memory
    - ❖ **Mono-programming** [easy but low resource utilisation, cannot be used for modern multiprogramming machine]

    No relocation, physical address
    - ❖ **Multiprogramming** with **fixed partition** [multi-programming but internal fragmentation]

    Relocation (register), logical address
    - ❖ **Multiprogramming** with **dynamic partition** [low internal fragmentation but high external fragmentation]

    Relocation (register), logical address
  - **Non-Contiguous Memory**

    Relocation (table), logical address
    - ❖ **Paging** (fixed-partitioning/code re-location) [Internal fragmentation reduce to last block, no external fragmentation; Require page table to maintain page relocation]
    - ❖ **Virtual Memory** (locality →not all pages loaded) [more processes → CPU utilisation, no external fragmentation, more memory available; Need involve replacement algorithm, page table management, thrashing, variable/fixed resident set….]

Recap
Take-Home Message

- **Contiguous** memory schemes: mono-programming, static and **dynamic partitioning**
- Internal and external **fragmentation**
- Memory allocation**, coalescing** and **compacting** in dynamic partitioning