Operating Systems and Concurrency

Lecture 14: Memory Management III

G52OSC

Geert De Maere and Isaac Triguero
{Geert.DeMaere,Isaac.Triguero}@Nottingham.ac.uk

University Of Nottingham
United Kingdom

2018

Recall

Last Lecture

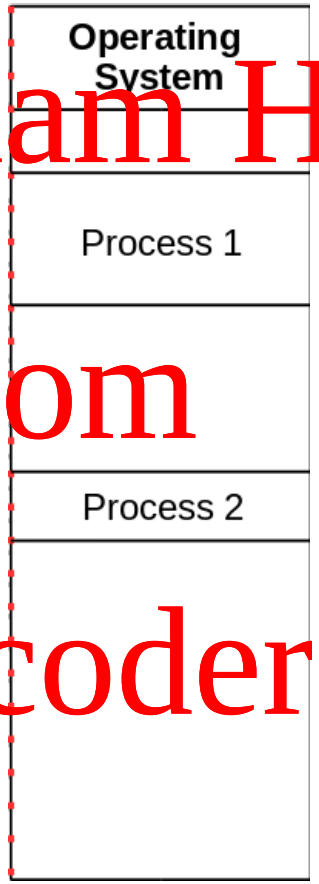
- **Dynamic relocation** and **protection** $\Rightarrow$ **base** (offset) and **limit** registers (logical/physical address)
- **Dynamic partitioning** $\Rightarrow$ **internal** $\Rightarrow$ **external fragmentation**
- **Dynamic Memory management** using **linked lists** and bitmaps

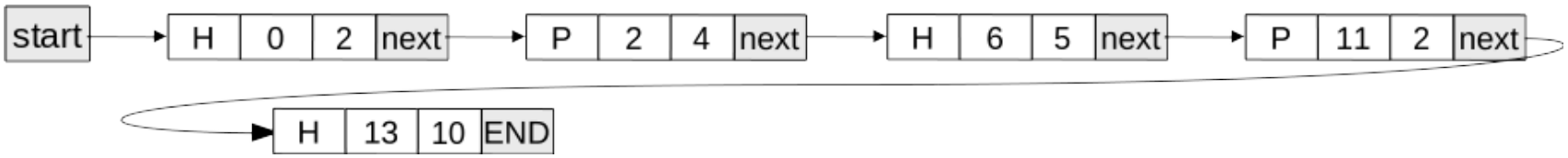
- **Dynamic partitioning management** with Linked lists.
- **Non-contiguous approaches:**
  - **Paging**, page tables, address translation

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**

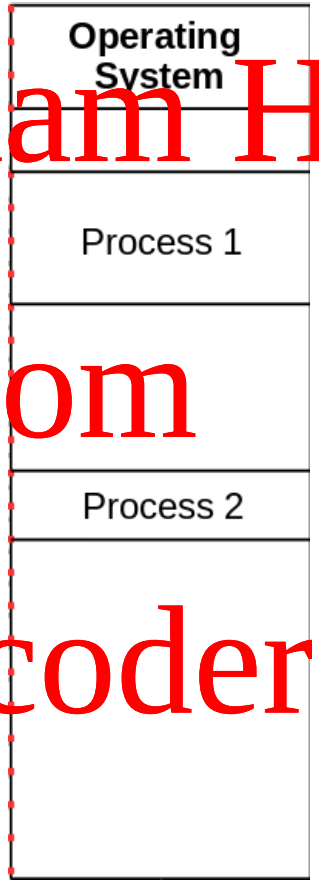


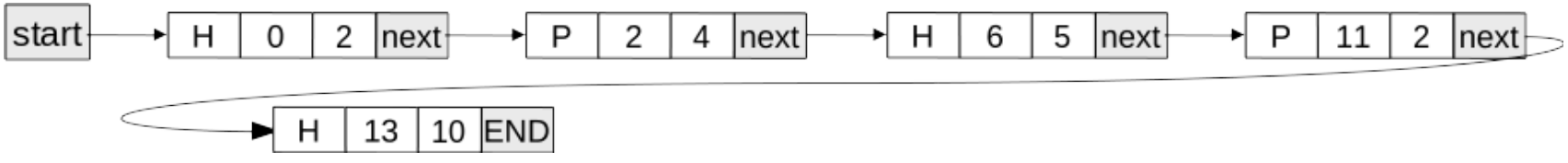


Dynamic Partitioning

Allocating Available Memory: First Fit

- **First fit** starts scanning **from the start** of the linked list until a link is found which represents **free space of sufficient size**
  - If requested space is **the exact same size** as the "hole", all the space is allocated
  - 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**"

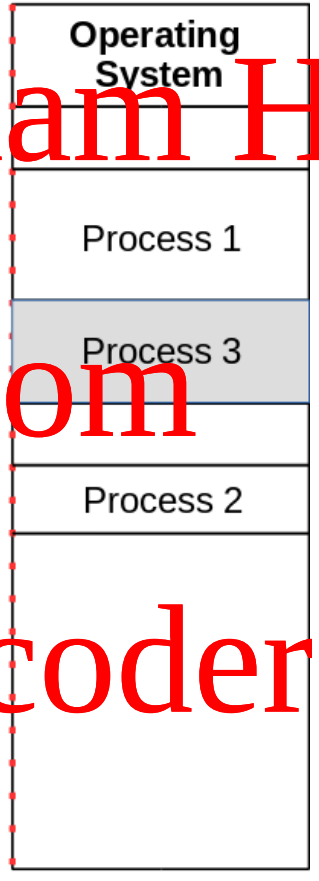


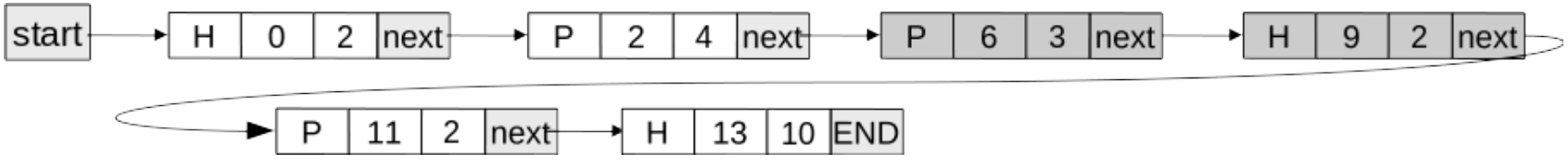


Dynamic Partitioning

Allocating Available Memory: First Fit

- **First fit** starts scanning **from the start** of the linked list until a link is found which represents **free space of sufficient size**
  - If requested space is **the exact same size** as the "hole", all the space is allocated
  - 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**"

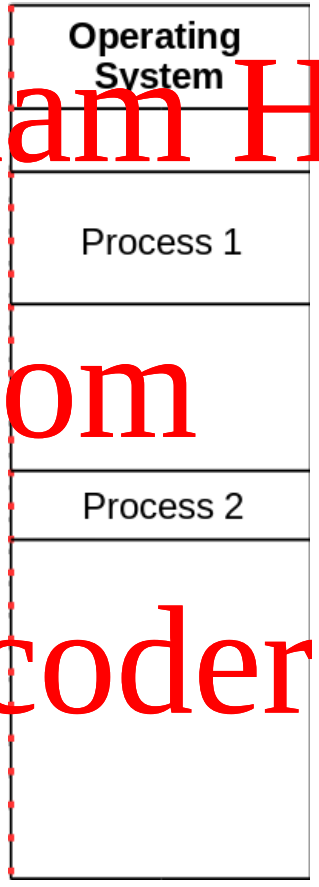


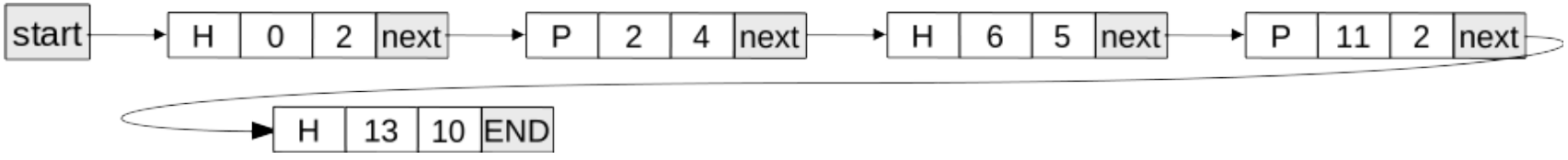


Dynamic Partitioning

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!

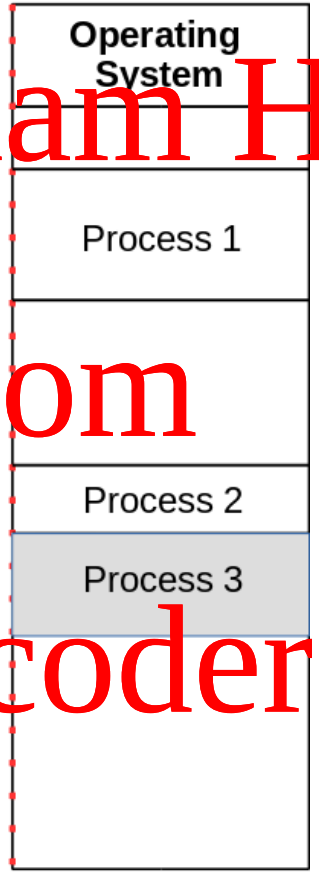


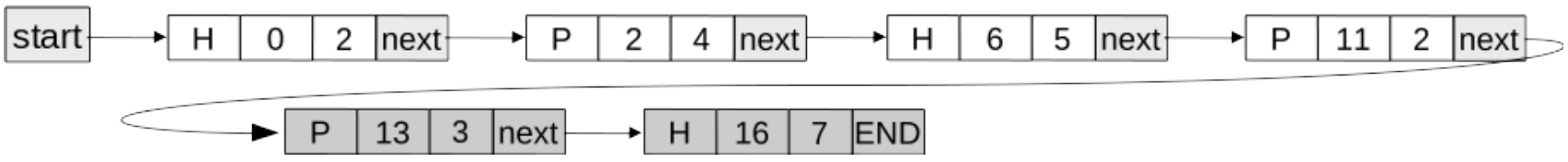


Dynamic Partitioning

Allocating Available Memory: 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

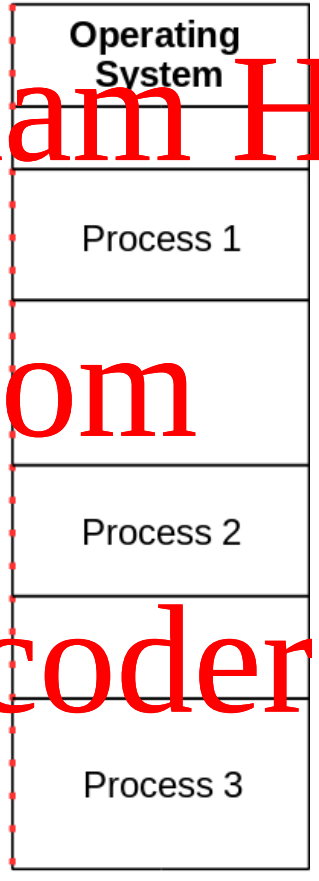
Allocating Available Memory: First Fit and Next Fit

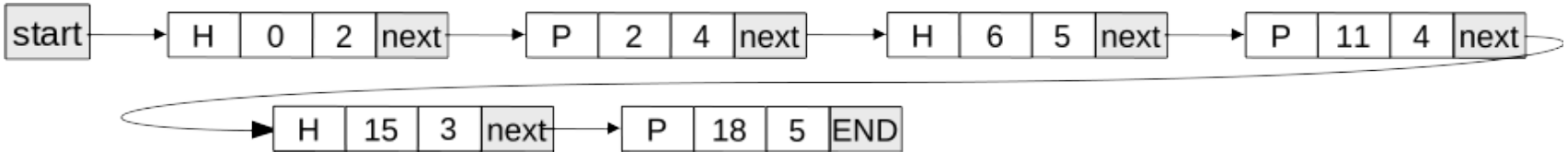
- **First fit** is a fast allocation method that just looks for the **first available hole**
  - It doesn't take into account that there may be **a hole later** in the list that **exactly(-ish)** fits the requested size
  - First fit **may break up a big hole** when the right size hole exists later on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

Dynamic Partitioning

Allocating Available Memory: Best Fit

- 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)

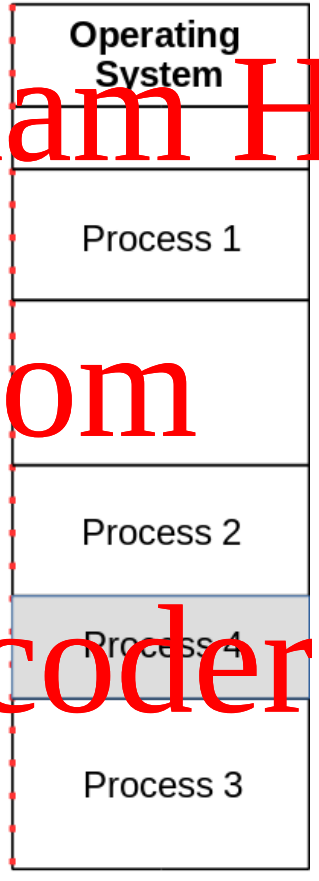


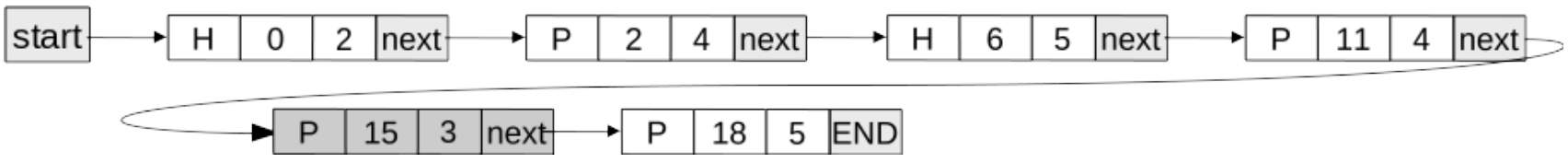


Dynamic Partitioning

Allocating Available Memory: Best Fit

- 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)

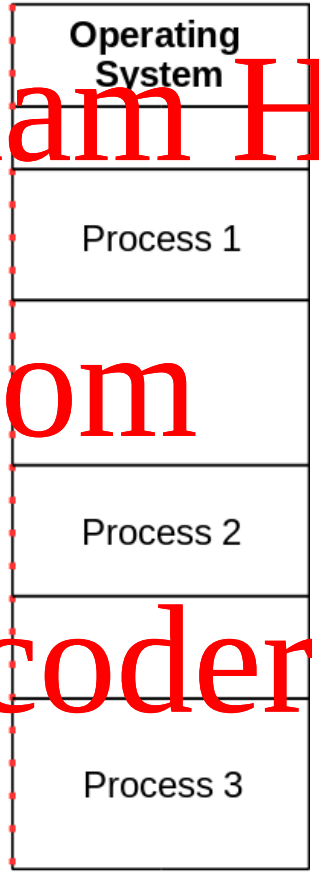


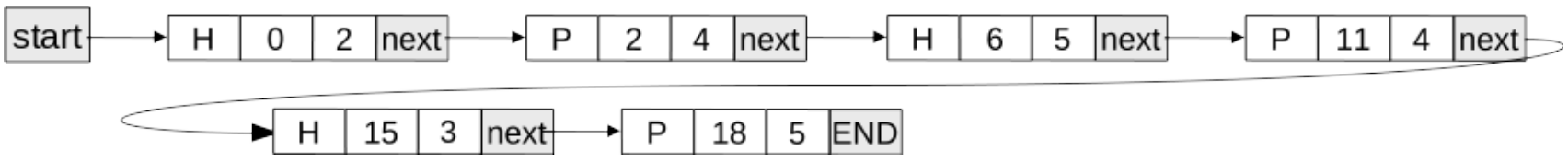


Dynamic Partitioning

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**!

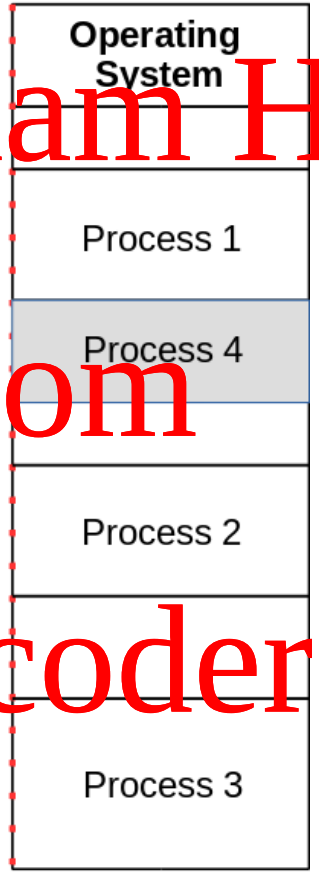


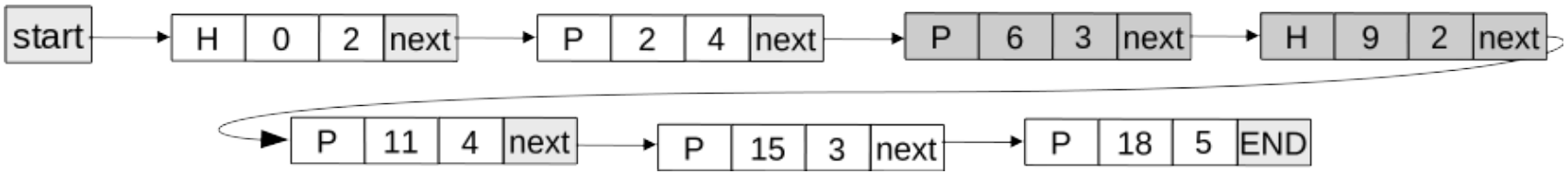


Dynamic Partitioning

Allocating Available Memory: 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

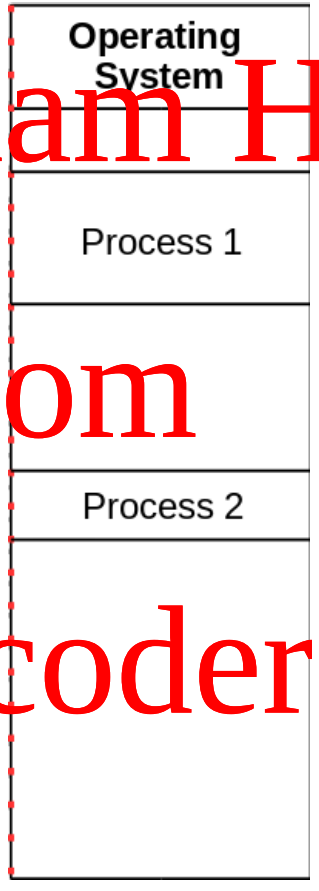
Allocating Available Memory: Quick Fit and Others

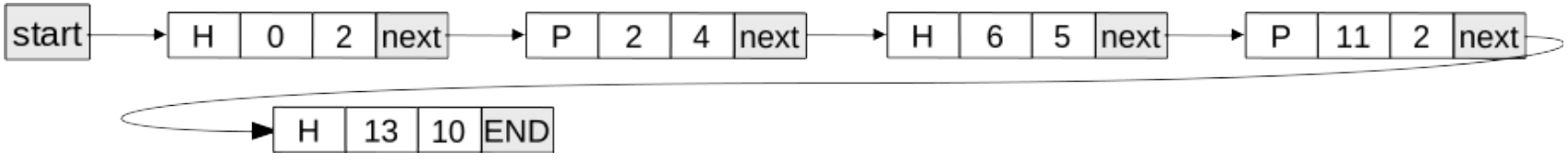
- **Quick fit** maintains **lists of commonly used sizes**
  - For example a separate list for each of 4K, 8K, 12K, 16K, etc., holes
  - **Odd sizes** can either go into the **nearest size** or into a **special separate list**
- It is **much faster** to find the required size hole using **quick fit**
- Similar to **best 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**
- 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

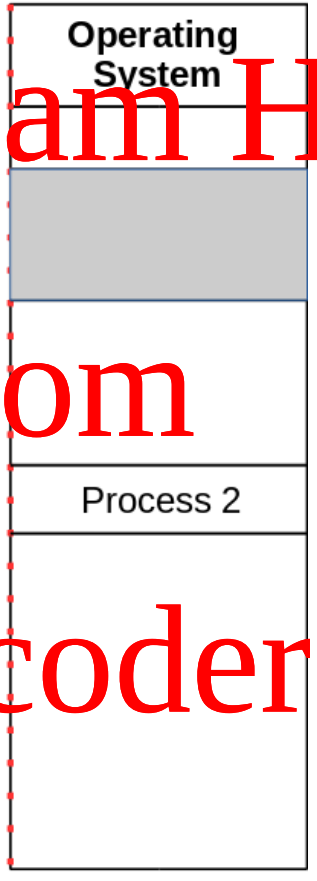


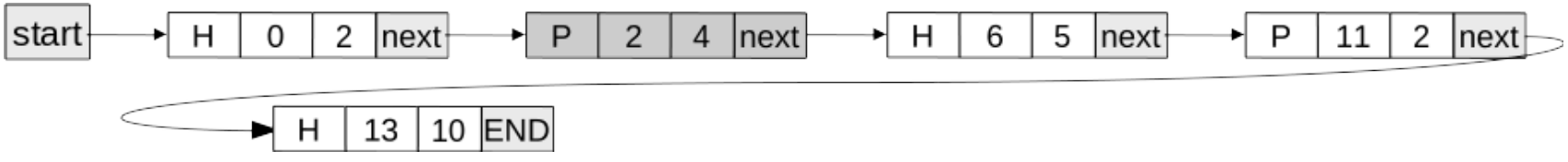


Dynamic Partitioning

Managing Available Memory: Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- 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

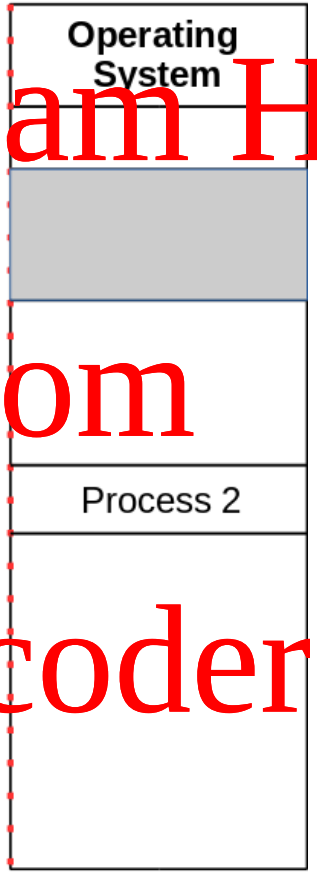


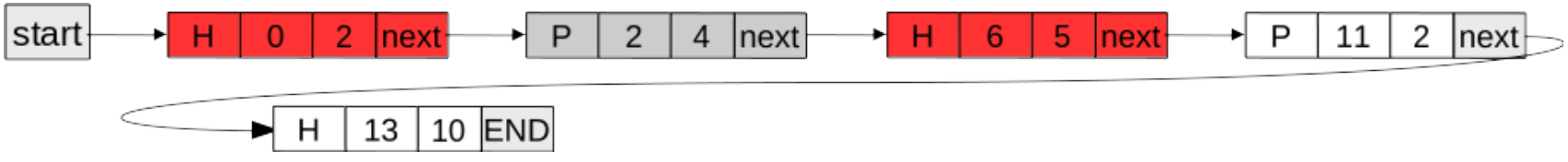


Dynamic Partitioning

Managing Available Memory: Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- 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

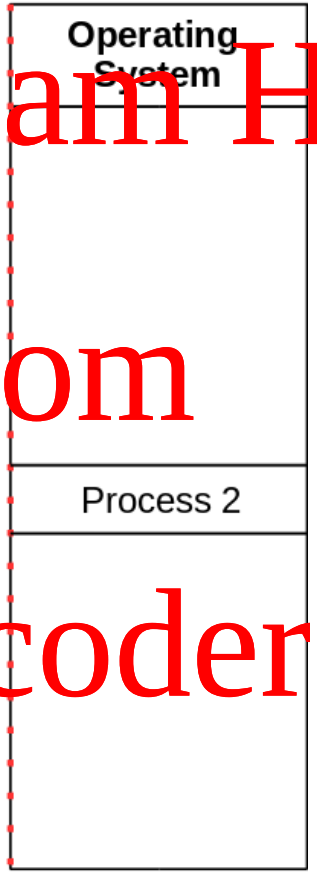




Dynamic Partitioning

Managing Available Memory: Coalescing

- **Coalescing** (joining together) takes place when two **adjacent entries** in the linked list **become free**
- 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: 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

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?

Paging
Principles

- **Paging** uses the principles of **fixed partitioning** and **code re-location** to devise a new **non-contiguous management scheme**:
  - Memory is split into much **smaller blocks** and **one or multiple blocks** are allocated to a process
    - e.g., a 1KB process would take up 3 blocks of 4 KB
  - These blocks **do not have to be contiguous in main memory**, but the process still **perceives them to be contiguous**

- Benefits compared to contiguous schemes include:
  - **Internal fragmentation** is reduced to the **last "block"** only
  - There is **no external fragmentation**, since physical blocks are **stacked directly onto each other** in main memory

Paging
Principles (Cont'ed)

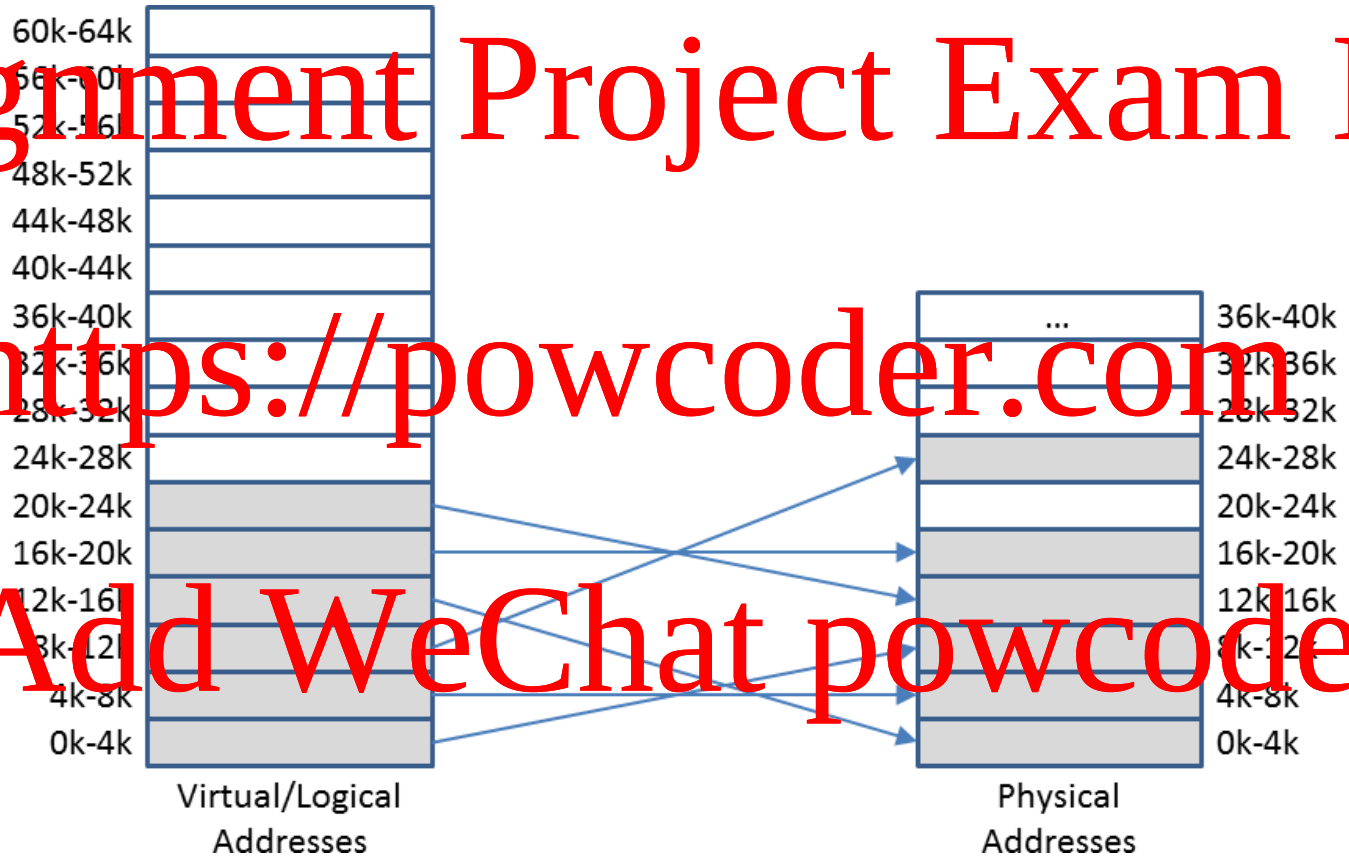


Figure: Paging in main memory with multiple processes

Paging
Principles (Cont'ed)

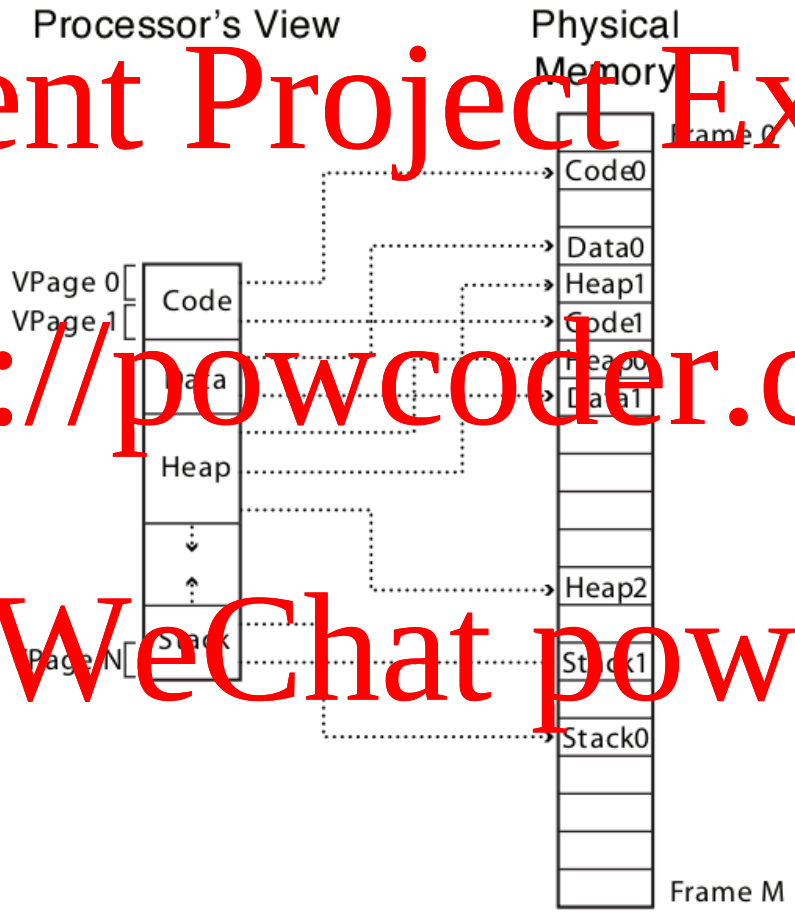


Figure: Paging in main memory with multiple processes (Anderson)

Paging
Principles (Cont'ed)

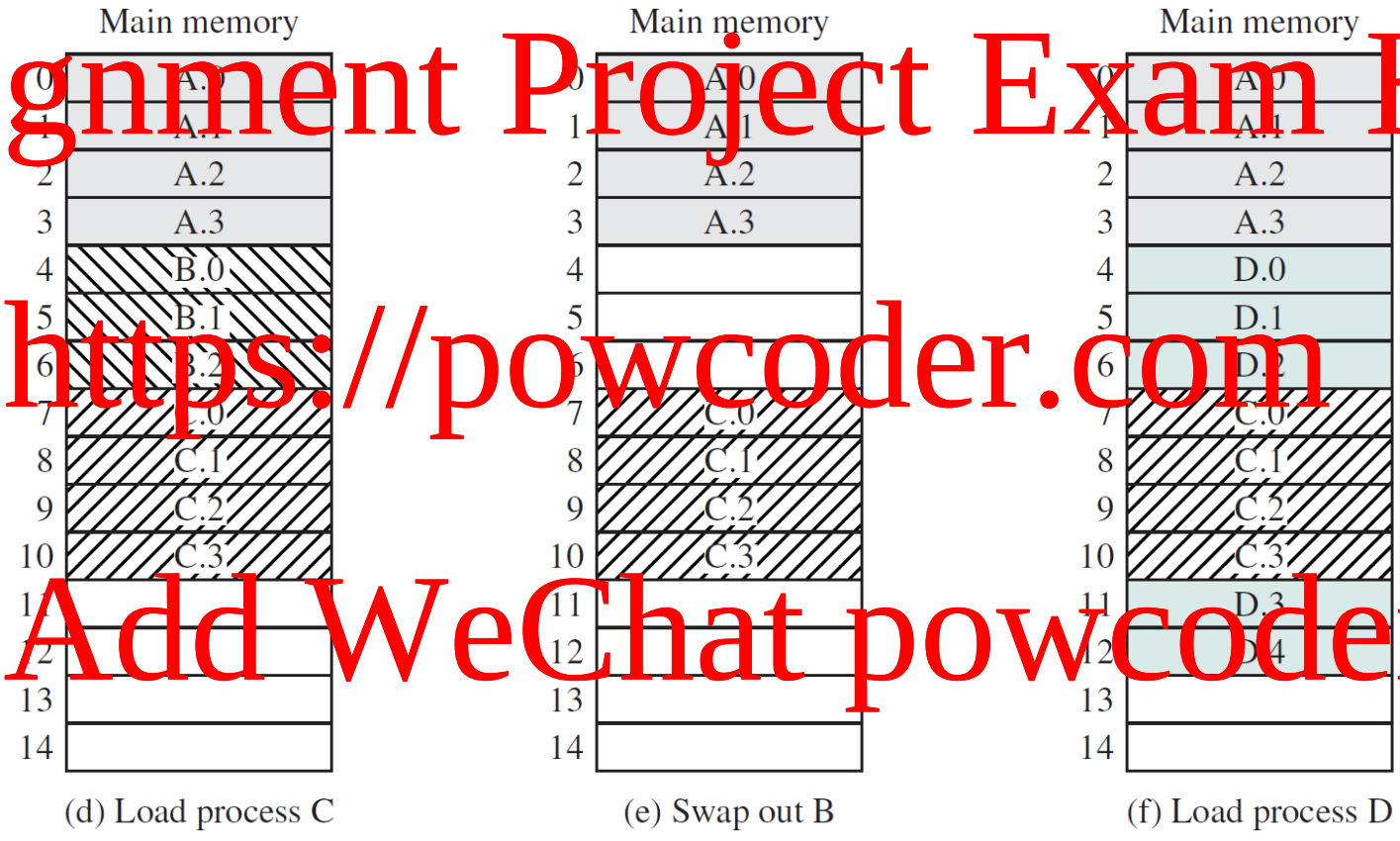


Figure: Concept of Paging (Stallings)

Paging

Principles: Definitions

- A **page** is a small block of **contiguous memory** in the **logical address spade**, i.e. as seen by the process
- A **frame** is a **small contiguous block** in **physical memory**
- Pages and frames (usually) have the **same size**:
  - The size is usually a power of 2
  - Sizes range between 512 bytes and 1Gb

Paging
Relocation

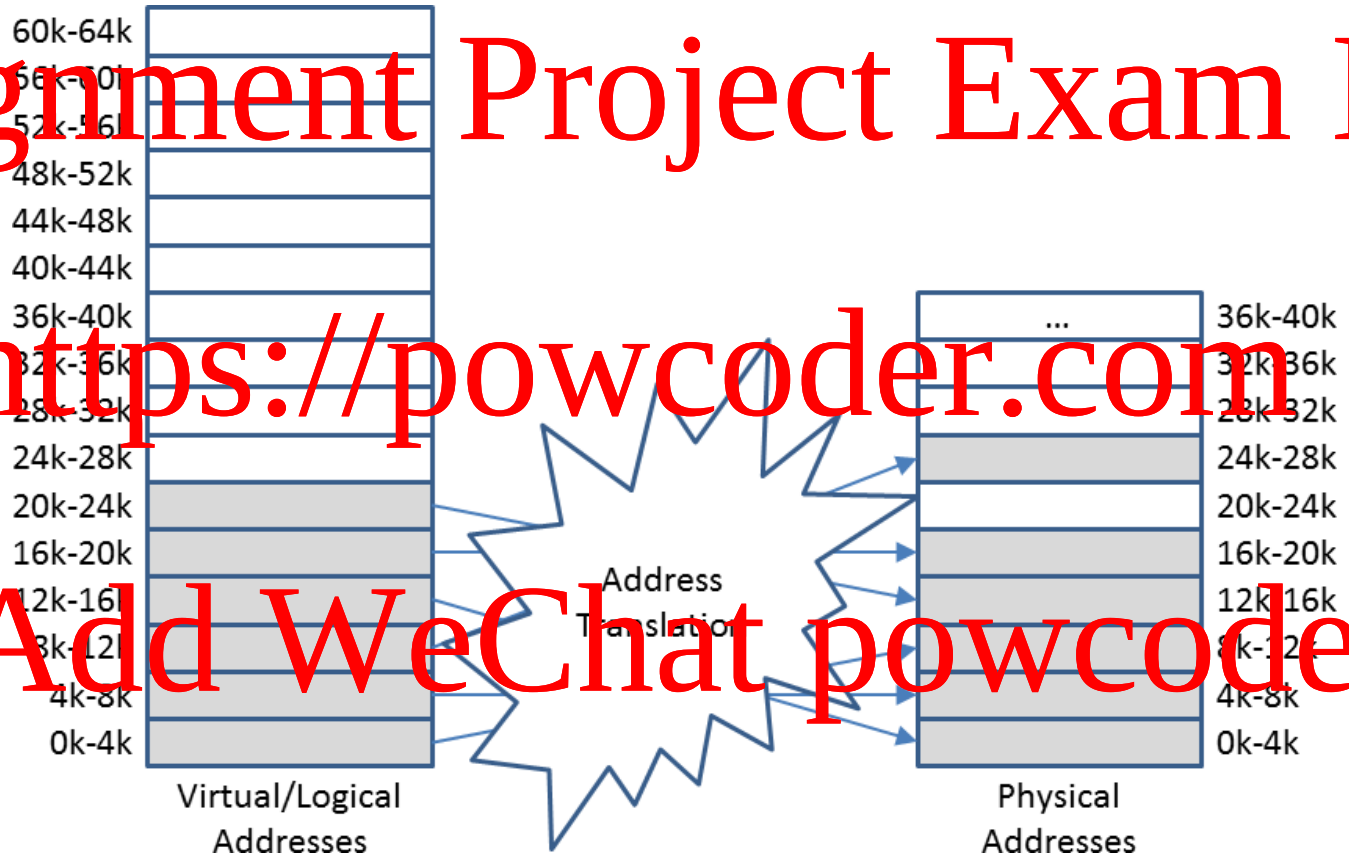


Figure: Address Translation

Paging
Relocation

- **Logical address** (page number, offset within page) needs to be **translated** into a **physical address** (frame number, offset within frame)
- **Multiple "base registers"** will be required:
  - Each logical page needs **a separate "base register"** that specifies the start of the associated frame
  - I.e, a **set of base registers** has to be maintained for each process
- The base registers are stored in the **page table**

Paging

Relocation: Address Translation

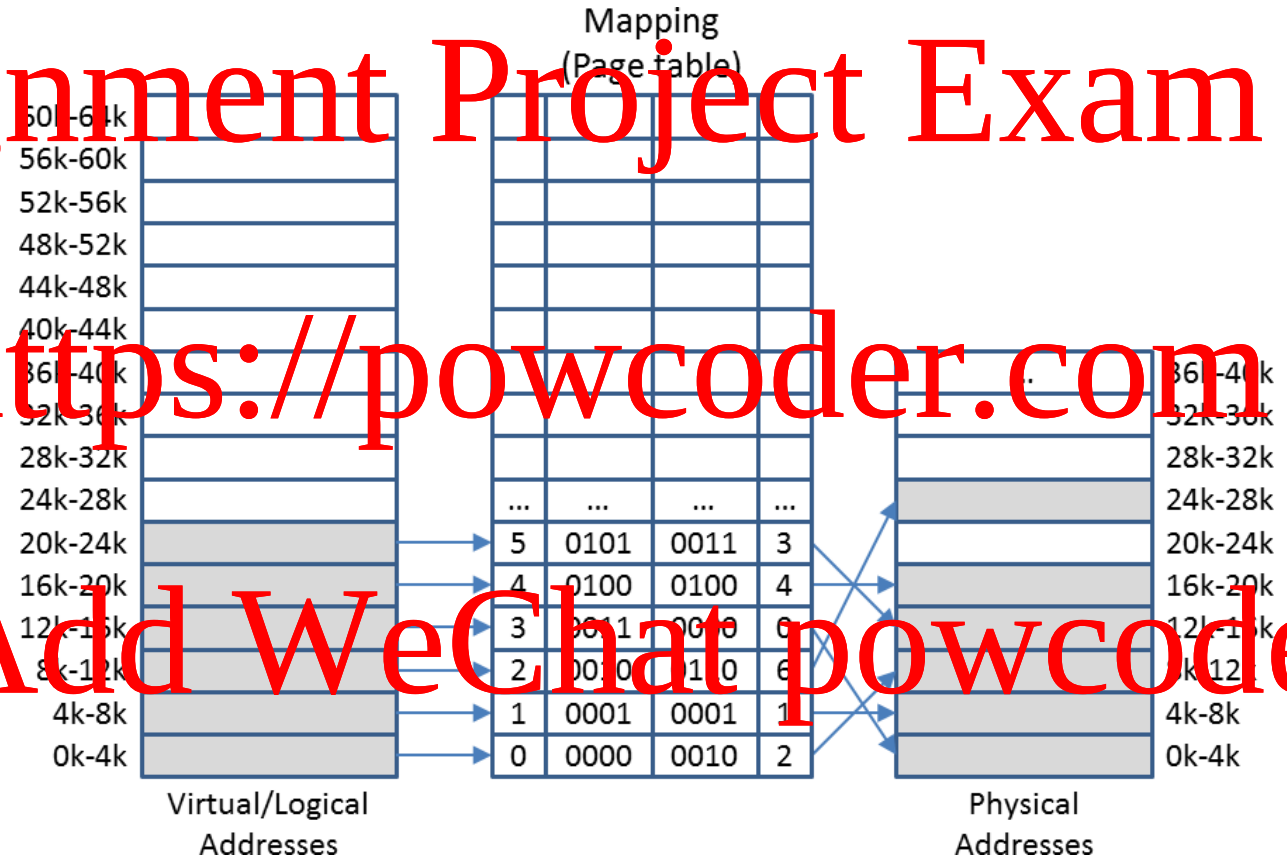


Figure: Address Translation

Paging

Relocation: Page Tables

- The page table can be seen as **a function**, that **maps the page number** of the logical address **onto the frame number** of the physical address
  - frameNumber=f(pageNumber)

- The **page number** is used as **index to the page table** that lists the **number of the associated frame**, i.e. it contains the location of the frame in memory

- Every process has its **own page table** containing its own "base registers"
- The **operating system** maintains a **list of free frames**

Paging

Address Translation: Implementation

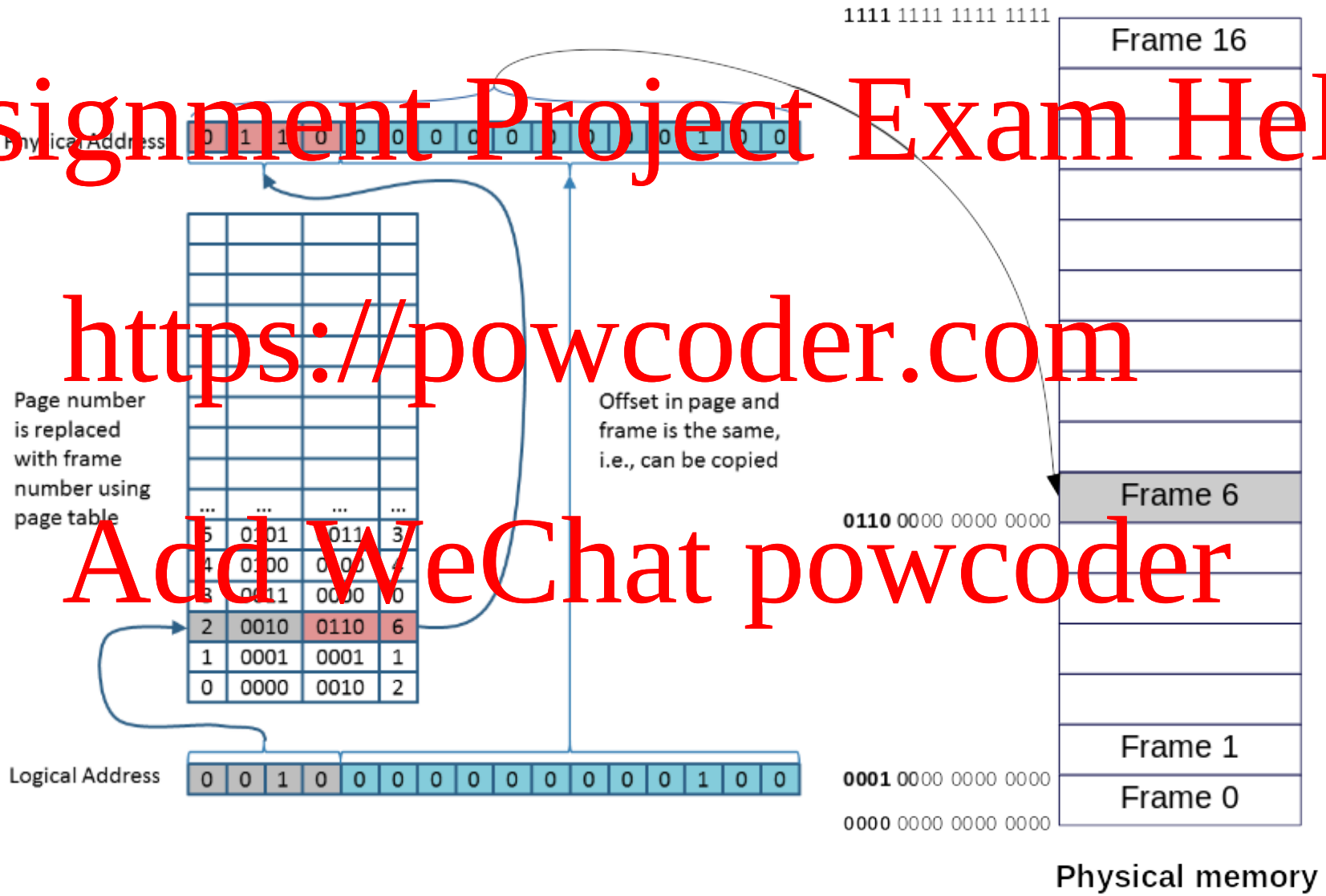


Figure: Address Translation

Problem

Page tables and memory addressing

Given a 64-bit machine that uses paging, and a page/frame size of 4096 bytes:

- What would be the maximum number of frames?

  Hint: First compute the maximum number of bytes you can address with a 64-bit machine.

- How many pages do we have in a 17 kilobytes process? How much memory are we wasting in the last partition?

- Submit your answers at:

  `https://b.socrative.com/login/student/`

  Room name: G52OSC

Recap

Take-Home Message[1]

- Memory **allocation**, **coalescing** and **compacting** in dynamic partitioning
- **Paging**, **page tables**, and **address translation**

[1] Tanenbaum Section 3.2, 3.3, Stallings Section 7.3