Lancaster University

Lecture 5: Memory Virtualisation

Lecture Plan

Lancaster University

- CPU virtualization allows processes to share a CPU.
  - Process, process states
  - Scheduling, system calls

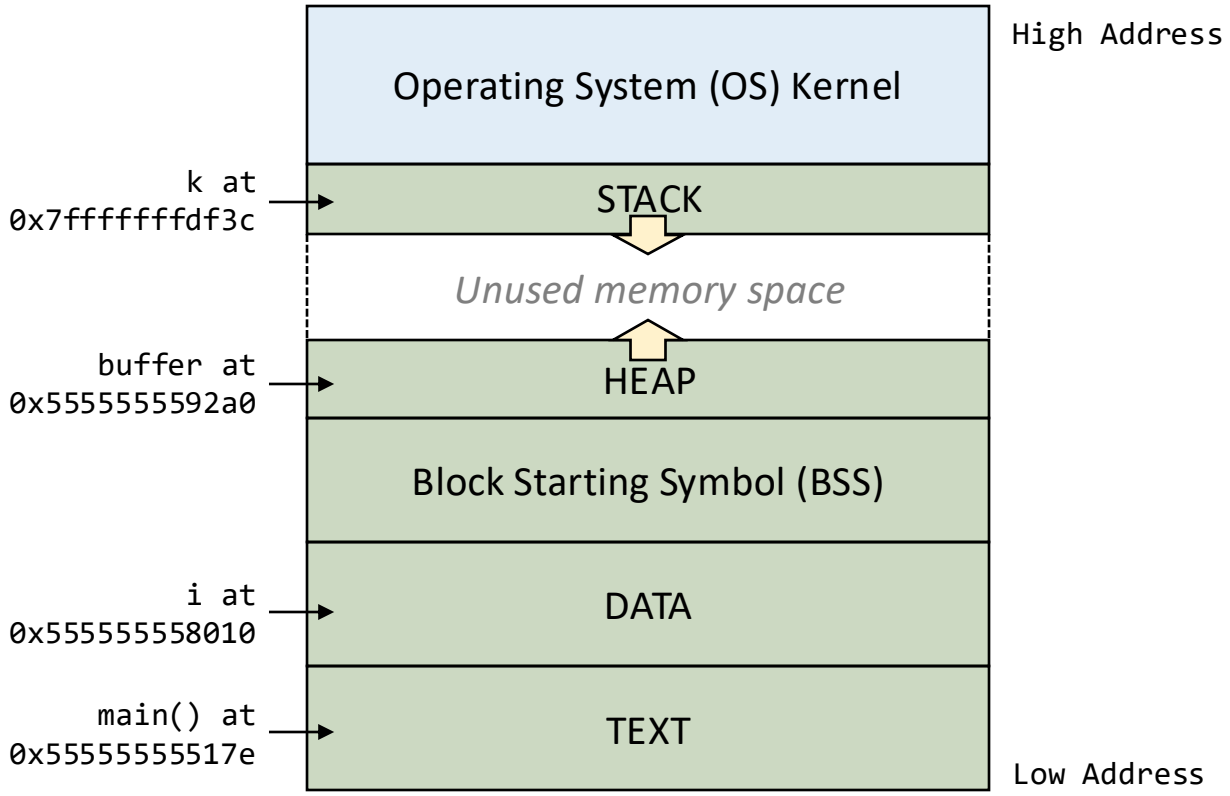
- What about other resources?
- Let's explore how memory is virtualised in modern OSes.
  - Remind the process memory layout
  - Design from scratch an efficient, secure, and developer-friendly memory

Memory layout – Bringing it all together

Lancaster University

SCC131_example3.c

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  int i = 5;
 5
 6  int func1(int a)
 7  {
 8     int k = 5;
 9  }
10
11  int main()
12  {
13     char * buffer;
14     buffer = (char*)malloc(i+1);
15     if (buffer == NULL) exit(1);
16     func1(i);
17     return 0;
18  }
```



Example

Lancaster University

```c
#include <stdio.h>
#include <unistd.h>

int i = 0;

int main() {
 i = getpid();
 printf("[%d] i address is %p\n", i, &i);
 return 0;
}
```

```
> ./test; ./test
[57849] i address is 0x555555558014
[57850] i address is 0x555555558014
```

How can two instances of the same program use the same memory address, but store different values?

OS memory structure

Lancaster University

procfs (the proc filesystem) is a Linux virtual filesystem that exposes kernel and process information as files. It lives under /proc and mounted boot.

Example files

/proc/<pid>/status	Process info (state, memory, threads, etc.)
/proc/<pid>/maps	Virtual memory map of a process
/proc/<pid>/fd/	File descriptors opened by the process
/proc/<pid>/cmdline	Command-line arguments

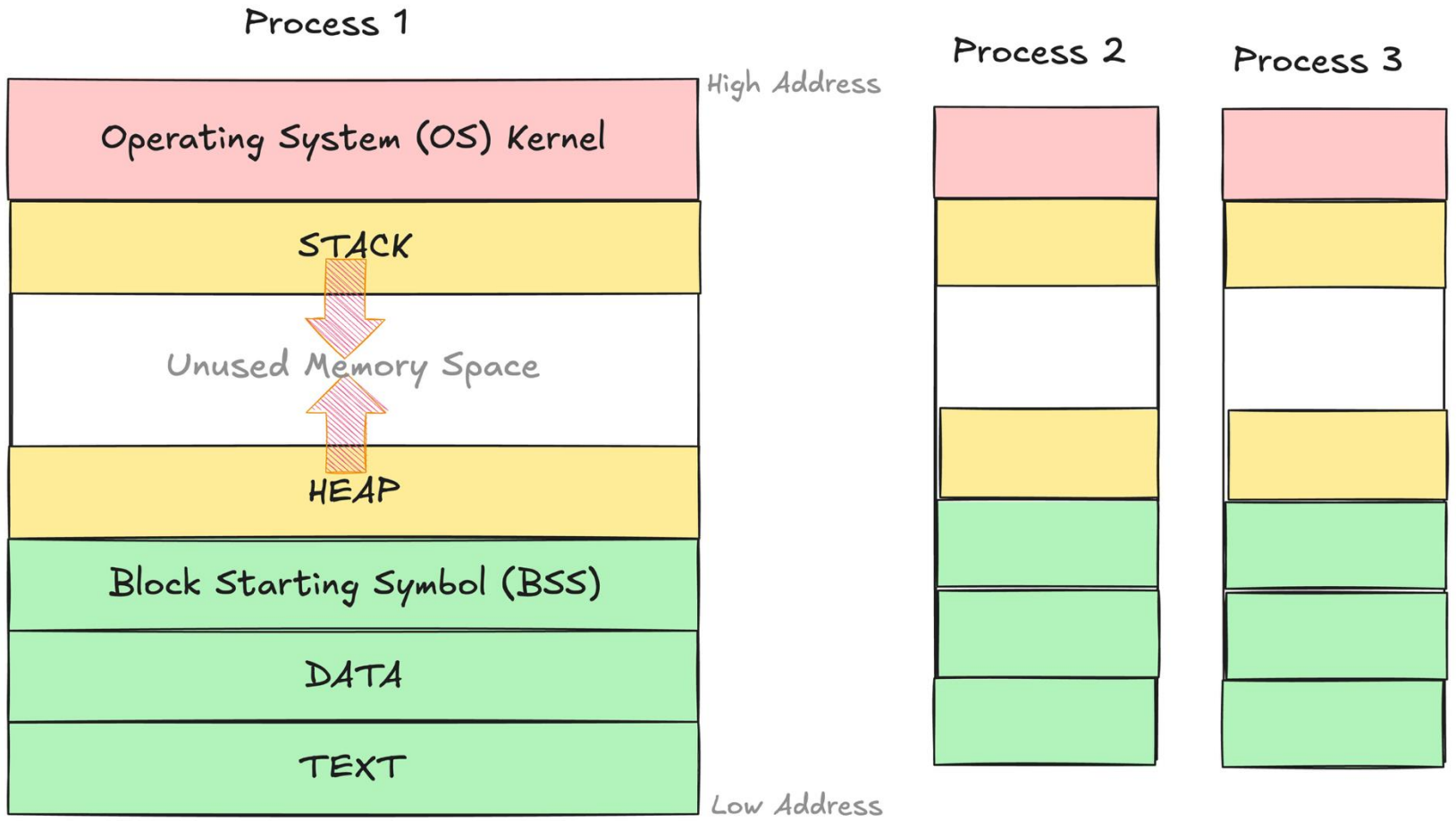
```
> cat /proc/self/maps
```

Adress range Permission Offset File location

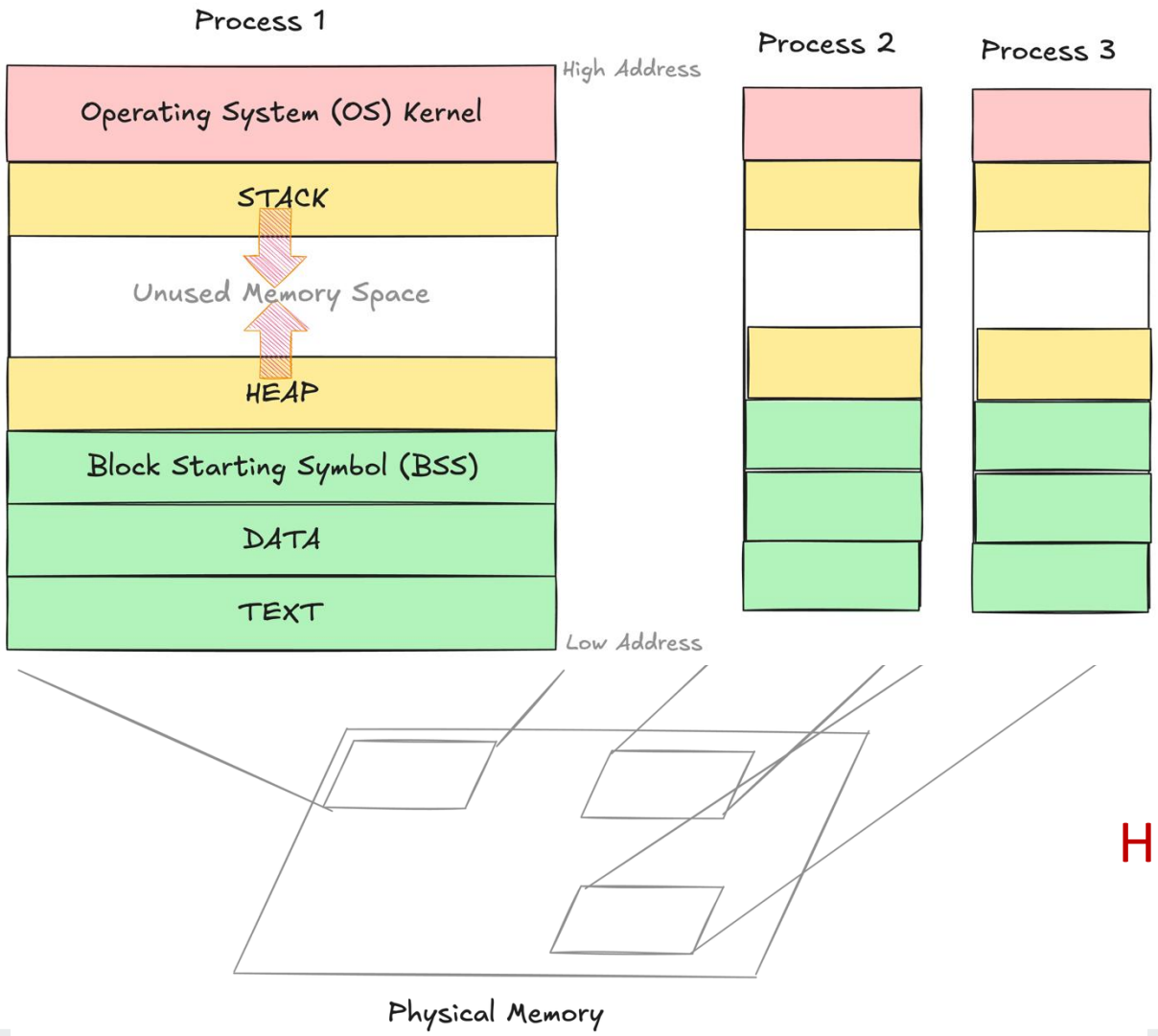
```
555555554000-555555556000 r--p 00000000 fc:00 4196283                    /usr/bin/cat
555555556000-55555555c000 r-xp 00002000 fc:00 4196283                    /usr/bin/cat
…
555555560000-555555581000 rw-p 00000000 00:00 0                          [heap]
7ffff7800000-7ffff7aec000 r--p 00000000 fc:00 4194424                    /usr/lib/locale/locale-archive
7ffff7c00000-7ffff7c28000 r--p 00000000 fc:00 4195686                    /usr/lib/x86_64-linux-gnu/libc.so.6
…
7ffff7fbb000-7ffff7fbd000 r--p 00000000 00:00 0                          [vvar]
7ffff7fbd000-7ffff7fbf000 r--p 00000000 00:00 0                          [vvar_vclock]
7ffff7fbf000-7ffff7fc1000 r-xp 00000000 00:00 0                          [vdso]
7ffff7fc1000-7ffff7fc2000 r--p 00000000 fc:00 4195619                    /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
…
7fffffffde000-7ffffffff000 rw-p 00000000 00:00 0                         [stack]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

A Split View Approach to Memory

Lancaster University



A Split View Approach to Memory



How does the OS delivers this abstraction?

Recap: Virtualization

Lancaster University

- A combination of indirection and multiplexing
- Create a **virtual version of a resource** (hardware, OS, network, or storage) so that multiple independent systems or applications can share it as if each has its own dedicated resource
- Examples:
  - virtual memory
  - virtual modem
  - Cloud computing
  - Santa claus
- Can cleanly and dynamically reconfigure the system
- Simplifies programming for developers; you can code without worrying synchronization with other programs

Memory Virtualisation

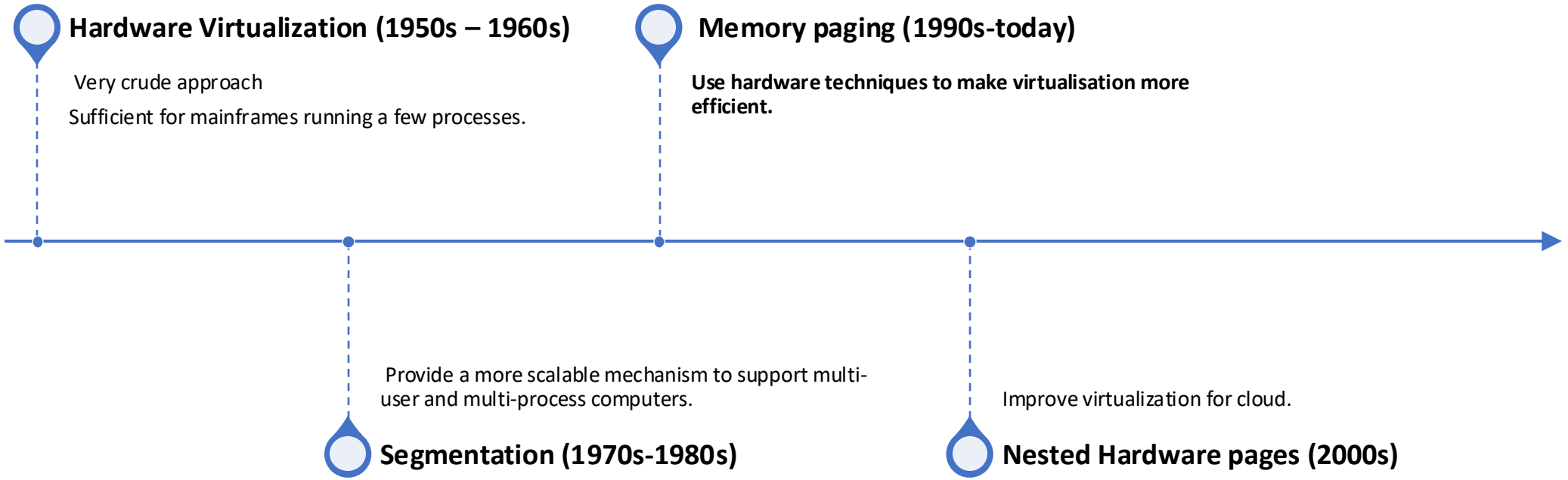
Lancaster University

- OSes use spatial virtualisation to share address spaces between processes.

- Goals:
  - **Transparency**: programmers should be unaware of memory virtualisation
  - **Efficiency**: virtualisation must not affect performance or negatively impact space allocation
  - **Protection**: processes must not be able to view or edit memory areas from other programs.

Memory Virtualisation timeline

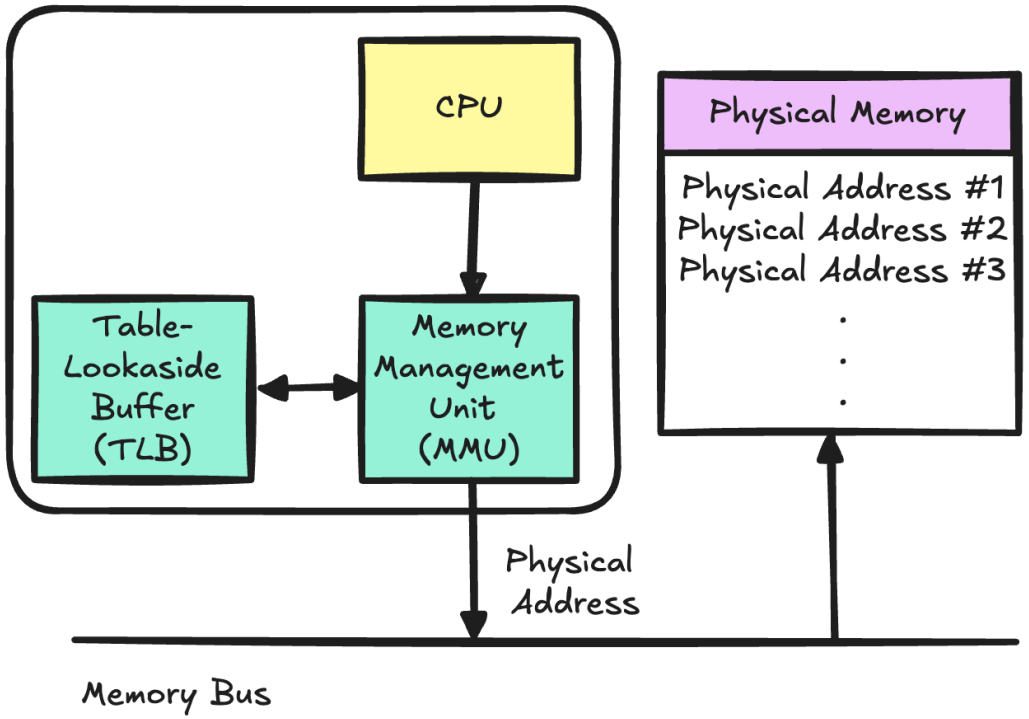
Lancaster University



Memory Components

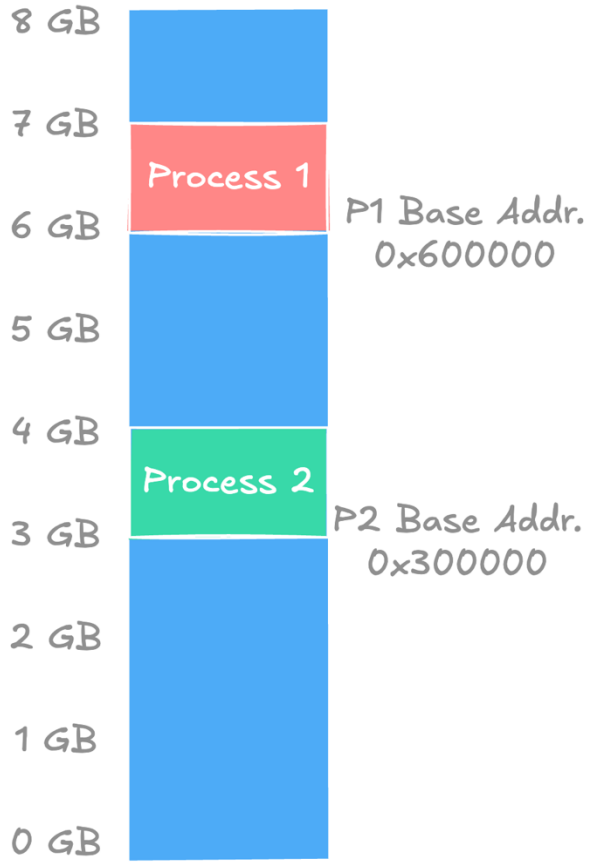
Lancaster University

- *Physical Memory*: The set of actual addresses of bytes on the RAM chips of a computer.
- *Virtual Memory*: The set of addresses visible to a process.
- *Process address space*: the range of addresses visible by a process.
  - Static: code and global variables
  - Dynamic: stack, head
- Memory Management Unit: a hardware component of the CPU that translate virtual pages to physical memory frames.
- Why do we need dynamic memory?
  - The amount of required memory may be task-dependent
  - Input size may be unknown at compile time
  - Conservative pre-allocation would be wasteful
  - Recursive functions (invocation frames)



Memory Virtualisation - Base and bounds (1)

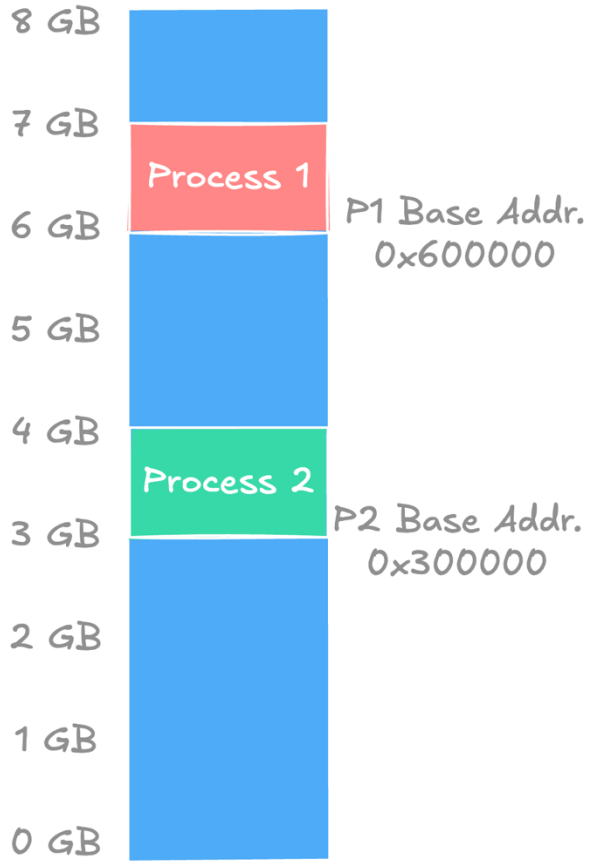
Lancaster University



- Hardware-based address translation using base and bounds; was the first attempt for memory virtualization.
  - Use two registers: a base register stores the start address in physical memory and bound stores the size.
  - Separate Physical Memory into dynamic blocks called **pages**.
  - Each process is allocated a page.
  - *physical address = virtual address + base*
- Example: Virtual address space is 1G for each process.
  - P1: 0x600000 - 0x6FFFFF
  - P2: 0x300000 - 0x6FFFFF
- A virtual address 0x1000 maps into address:
  - P1 = 0x601000, P2 = 0x301000

Memory Virtualisation - Base and bounds(2)

Lancaster University



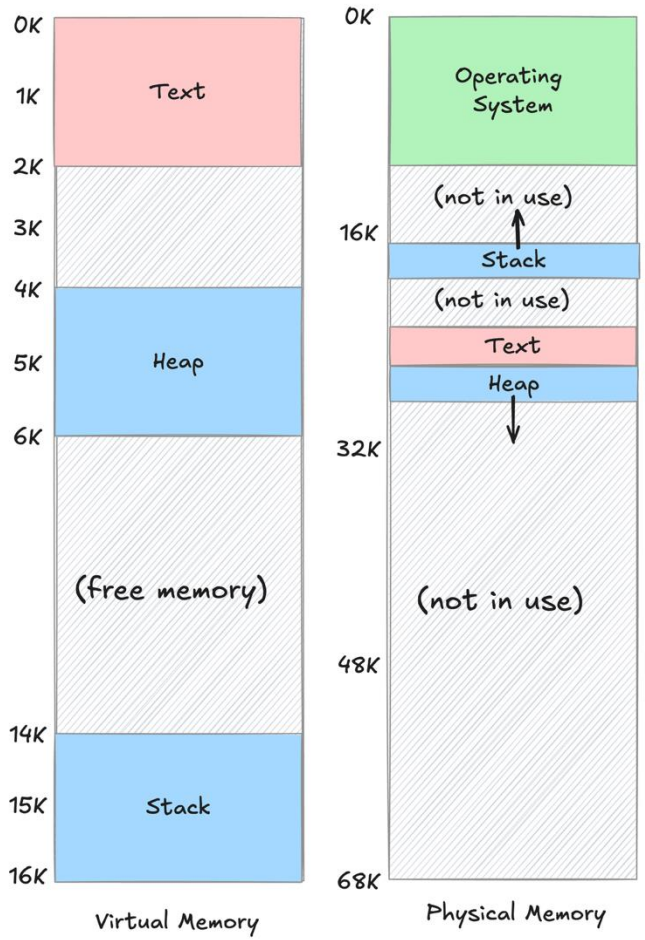
- To protect memory, use an arithmetic check to control memory access.
- base_addr <= addr < base_addr + 0xFFFFF
- Everytime a process is deschedule, then the base and bound registers must be updated.
- Problem: **internal fragmentation**
  - All memory is continuously allocated
  - Waste of physical memory – remember the large space chunk between stack and heap
  - No memory sharing between processes

Memory Virtualisation - Segmentation

Lancaster University

- Break physical memory into segments.
  - Each program section has a pair of base and bounds values.
- Memory sharing : Use flags to indicate if programs can read, write or execute a segment.
  - Read-only code segments shared between processes.
  - Process can access shared memory as its own virtual memory and ensure isolation.

Segment	Base	Size	Direction	Flags
Code	32K	2K	1	*Read-Execute*
Heap	34K	2K	1	*Read-Write*
Stack	28K	2K	0	*Read-Write*

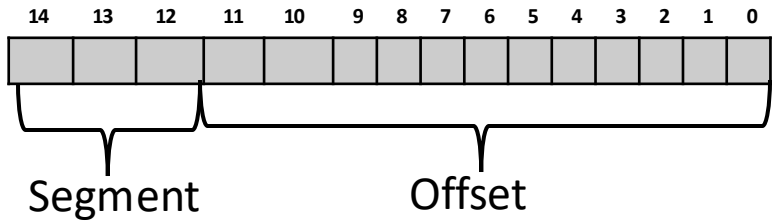


Segmentation – Address Translation

Lancaster University

- Using lots of if statements is slow to check and translate virtual to physical addresses.
- Use bitmasks to speed up the process.

Segment	Mask	Value	Base
Code	0x3800	0x0000	0x10XX
Heap	0x3800	0x0800	0x12XX
Stack	0x3800	0x1000	0x1AXX



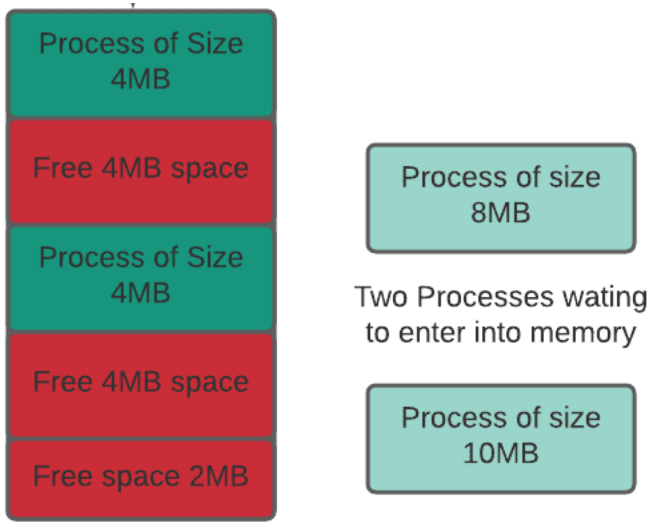

```
// get top 2 bits of 14-bit VA
Segment = (VirtualAddress & SEG_MASK) >> SEG_SHIFT;

// now get offset
Offset = VirtualAddress & OFFSET_MASK;
if (Offset >= Bounds[Segment]) {
 RaiseException(PROTECTION_FAULT);
} else {
 PhysAddr = Base[Segment] + Offset;
 Register = AccessMemory(PhysAddr);
}
```

Any lookup without a match is a segmentation fault, e.g. 0x8000.

Segmentation Limitations

Lancaster University

- External Fragmentation
  - When segments are variable size, free memory can become fragmented.
  - Even if free memory can satisfy a request, it may not be contiguous.
  - Memory compaction can solve the problem, but computationally expensive.
- Complex memory management
  - OS must keep track of base and limit for every segment of every process.
  - Dynamic growth (e.g., expanding the stack or heap) may require relocating other segments.
  - Makes allocation and relocation overhead high.
- Memory compaction: rearranging (shuffling) processes in memory so that all free memory is combined into one large contiguous block.

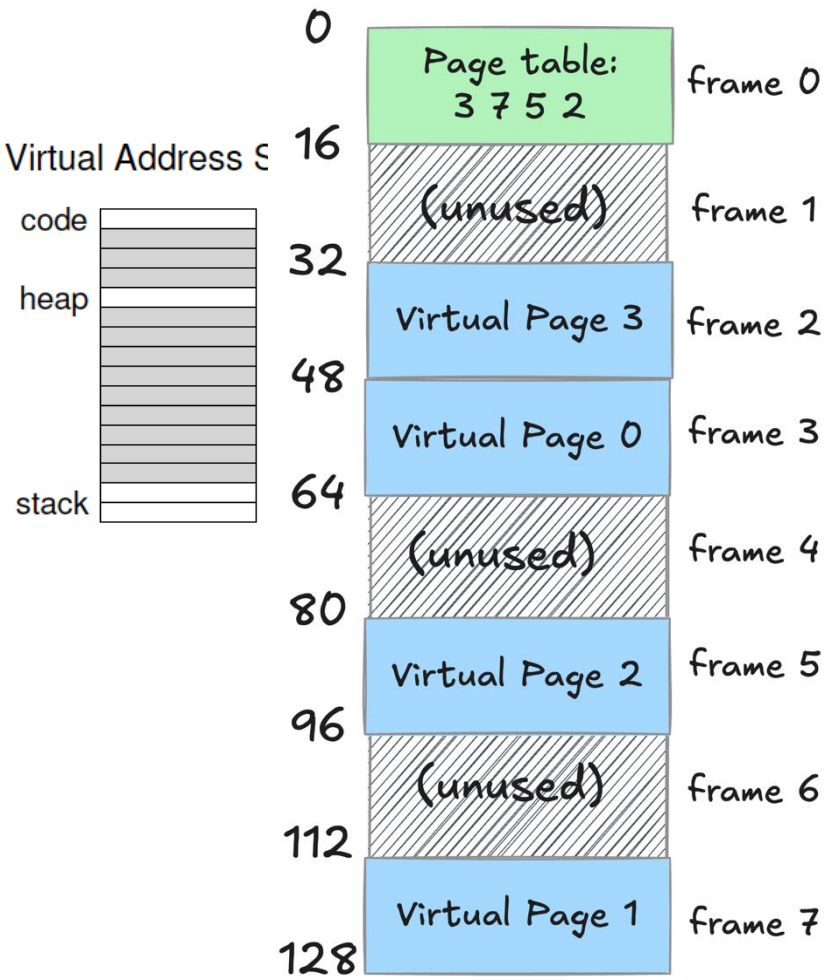


Variable size fragments creates fragmentation and fixed size memory segments waste resources. What if we make segments smaller?

Memory Virtualisation - Paging

Lancaster University

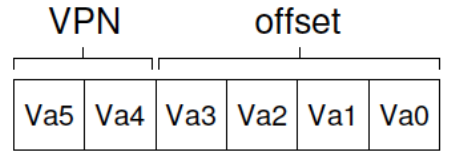
- Memory paging: *chop up space into small, fixed-sized* pieces.
  - In virtual memory, this is called a page, and in physical memory, it is called a frame.
- Page table: stores physical translations for each of the virtual pages of the address space (one table per process).
  - Use bitmasks for efficient lookups.
- Example: 128-byte physcial memory with eight frames of 16 bytes
  - (Virtual Page 0 → Physical Frame 3), (VP 1→PF 7), (VP 2→PF 5), and (VP 3→PF 2).

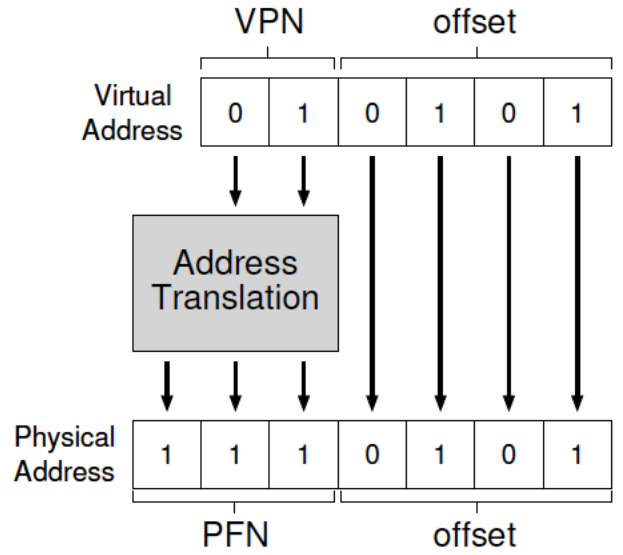


Example Memory Translation

Lancaster University

- Split virtual address into two components: the **virtual page number** (VPN), and the **offset** within the page.
- Use VPN as an index in the page table to find which Physical Frame Number (PFN) has the virtual page.
- Replace VPN with PFN in the address, and you have the physical address.

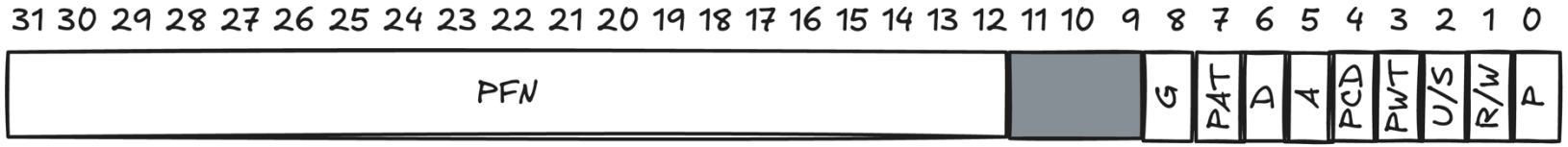




Page table Entry

Lancaster University

- PFN: Physical frame number
- PWT, PCD, PAT, and G determine how hardware caching works for these pages
- accessed bit (A) and a dirty bit (D): used to optimize how pages are cached.
- read/write bit (R/W) which determines if writes are allowed to this page
- user/supervisor bit (U/S) if user-mode processes can access the page
- A present bit indicateswhether this page is in physical memory or on disk (i.e., it has been swapped out).



Memory Paging Performance

Lancaster University

- Page tables can get terribly large
  - Consider a 32-bit address space, with 4KB pages.
  - 20-bit VPN and 12-bit offset (enough to point to 4K bytes).
  - A 20-bit VPN implies that there are 2^{20} translations that the OS would have to manage for each process (that's roughly a million).
  - If a page table entry (PTE) is 4 bytes then we need 4MB per process.
  - If 100 process run in a system then the OS requires 400 MB fjust for memory.
- Page tables are store in memory and each memory translation requires two accesses at least.

Translation-Lookaside Buffer

Lancaster University

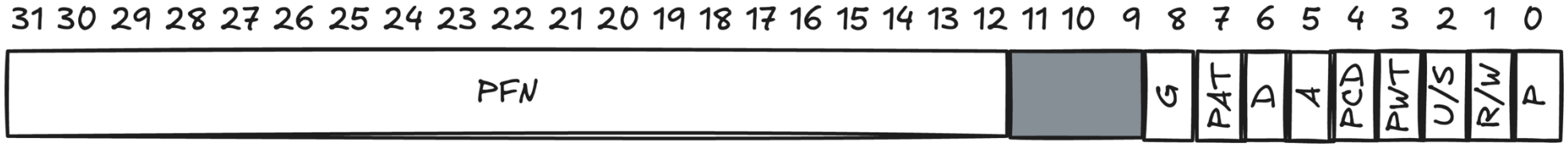
- A TLB is part of the chip's memory-management unit (MMU).
  - A hardware cache of popular virtual-to-physical address translations thus,
  - address-translation cache.
  - The hardware first checks the TLB for a translation.
  - Reduce slow memory access.
- If a page is not in the TLB, i.e. TLB miss, this can trigger a TLB update.
  - Which entry is replaced: Least Recently Used, Least Popular…
- Who manages a TLB miss?
  - X86 and CISC architecture handle this in hardware.
  - RISC architecture, like MIPS, use software TLB miss handlers.

But what about the table size?

Memory Table entries

Lancaster University

- Each page table entry contains a frame pointer (PFN) and metadata.
  - Present bit (P): is the frame in main memory or swap.
  - Read/write bit (R/W) is the page writable
  - User/supervisor bit (U/S): are user-mode processes can access the page
  - PWT, PCD, PAT, and G: how hardware caching works for these pages
  - Accessed bit (A): used to record page popularity
  - Dirty bit (D): record if page has been modified since last load

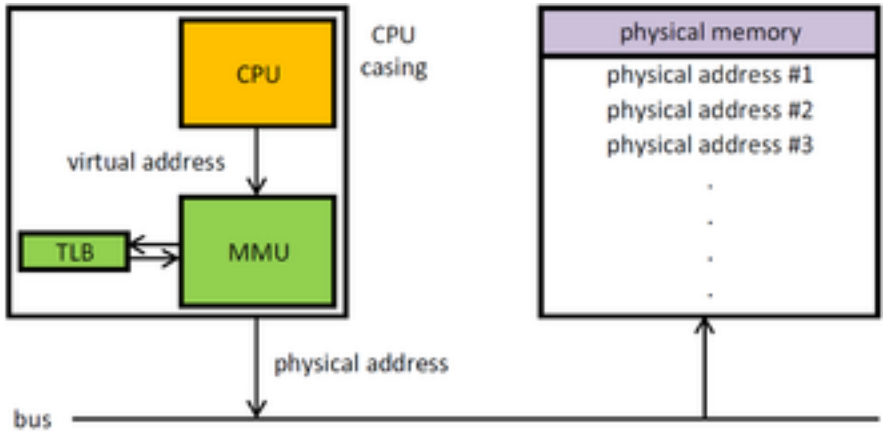


Translation-Lookaside Buffer

Lancaster University

- A TLB is part of the chip's memory-management unit (MMU), and is simply a hardware cache of popular virtual-to-physical address translations
  - address-translation cache.
- Upon each virtual memory reference, the hardware first checks the TLB to see if the desired translation is held therein; if so, the translation is performed (quickly) without having to consult the page table (which has all translations)
- The TLB contains virtual-to-physical translations that are only valid for the currently running process; these translations do not apply to other processes. As a result, when switching from one process to another, the hardware or OS (or both) must be careful to ensure that the about-to-be-run process does not accidentally use translations from some previously run process.

- Which page entries should be replaced when the TLB is full?
  - Most recent? Least accessed? Oldest?



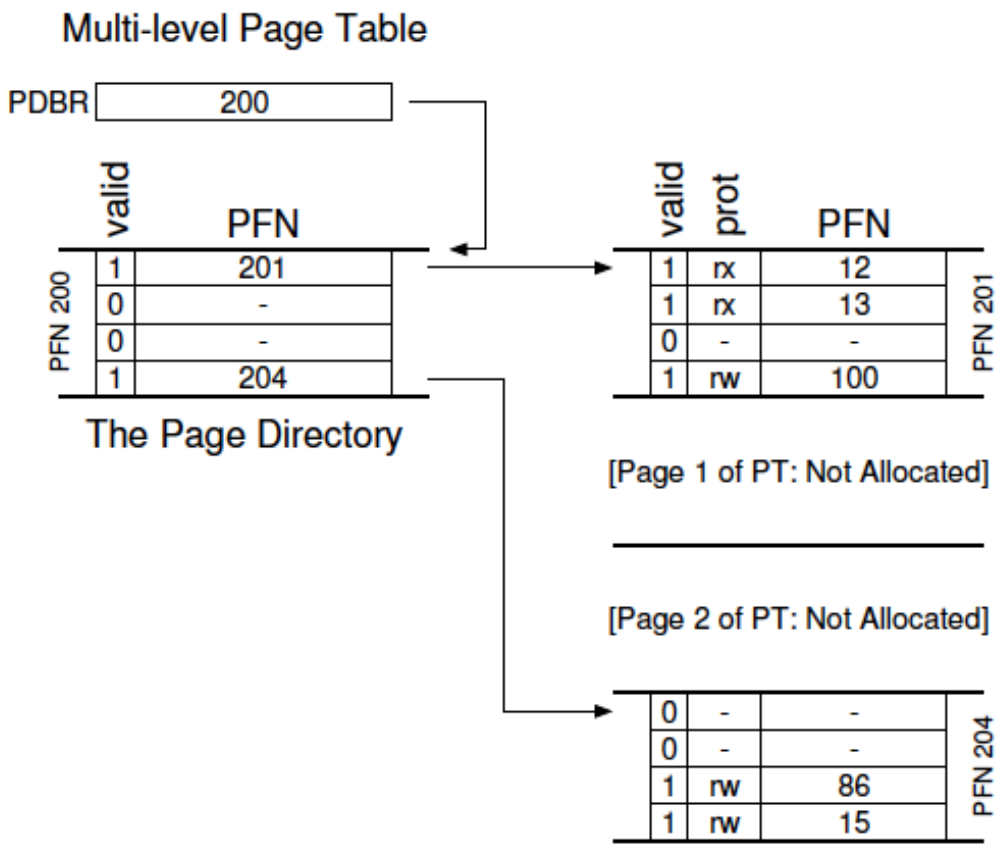
Page Table Size

Lancaster University

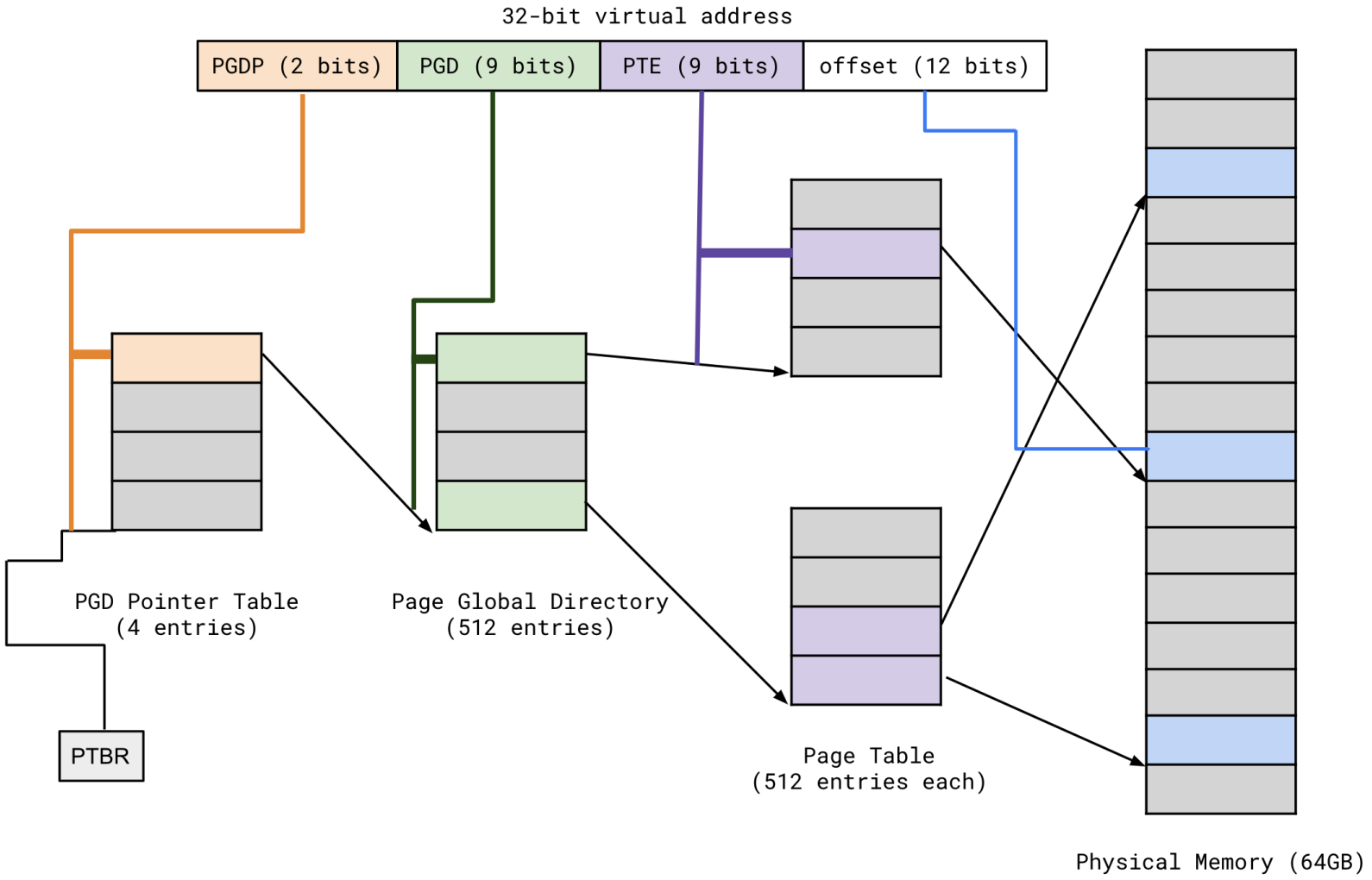
- Imaging a CPU with a 32-bit address space with 4KB pages.
  - 20-bit VPN and 12-bits offset (0-4095).
  - 2^{20} entries in the page table.
  - If 4 bytes per entry, we need ~4 MB of memory page state per process
  - If 100 process run in parallel, we need 400MB…
  - … which is too expensive to store in CPU/MMU.
- Page tables can get terribly large, much bigger than the small segment table or base/bounds pair we have discussed previously.
- The Page Table is stored in memory; every memory lookup requires two memory accesses to read data.

Multi-layer Page Tables

- Chop up the page table into page-sized units
  - if an entire page of page-table entries (PTEs) is invalid, don't allocate that page of the page table at all.
- To track whether a page of the page table is valid (and if valid, where it is in memory), use a new structure, called the page directory.
  - The page directory thus either can be used to tell you where a page of the page table is, or that the entire page of the page table contains no valid pages.
- This can quickly reduce the 4M per process, to a smaller number.



X86 page table structure



Memory API

Lancaster University

`malloc(size_t size)`	Allocate memory bytes of memory in the heap. WARNING: Always check that the return value is not NULL.
`free(void *ptr)`	Release previously malloced memory. NOTE: You can free a pointer onlt once. Double free will cause a SEGFAULT.
`calloc(size_t count, size_t size)`	Contiguously allocates enough space for count objects that are size bytes.
`realloc(void *ptr, size_t size)`	Change the size of the allocation pointed to by ptr to size, and returns ptr.

Languages like Python and Java abstract memory management for you;
They can detect memory requirements and dynamically control allocation.

Memory Allocator

Lancaster University

- A **memory allocator** is an OS service that manages **dynamic allocation and deallocation of memory** for processes.
  - It decides **where** in memory to place a new block when requested.
  - It keeps track of which regions are **free** and which are **in use**.
  - Minimize internal fragmentation
  - Manages both virtual pages and physical memory.
- Lots of approaches to select best location:
  - **First fit**: place in the first free block big enough:
  - **Best fit**: place in the smallest suitable free block.
  - **Worst fit**: place in the largest block (leaving smaller blocks for smaller requests).
  - **Buddy allocator**: split memory into powers of two, so merging is efficient.
  - **Slab allocator**: pre-allocate fixed-size chunks of object

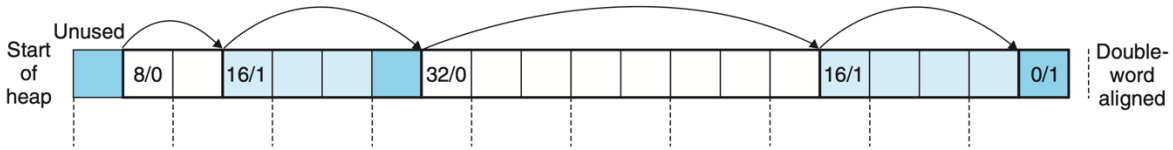
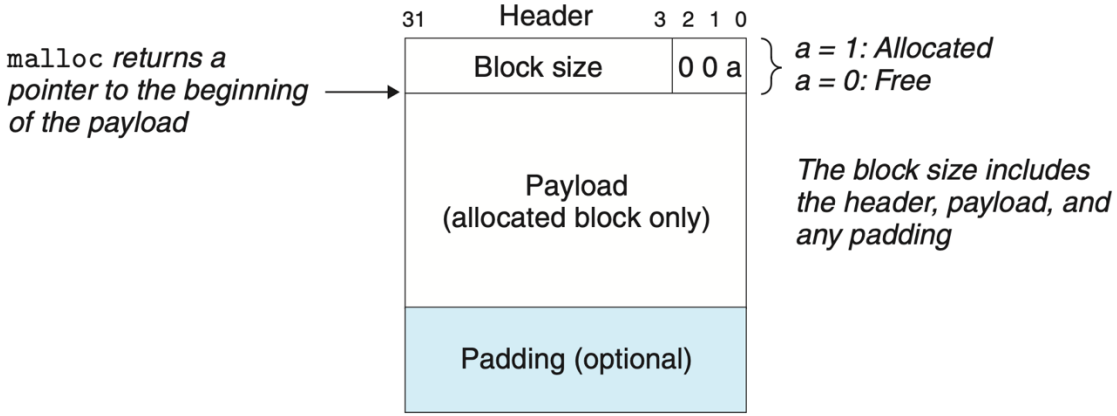


Figure 9.36 **Organizing the heap with an implicit free list.** Allocated blocks are shaded. Free blocks are unshaded. Headers are labeled with (size (bytes)/allocated bit).



Conclusion

Lancaster University

- Memory virtualisation is a key mechanism to share efficiently and securly memory, through a simple programming model.
- Memory virtualisation has evolved over the years:
  - Hardware virtualisation
  - Segmentation
  - Paging
- X86 memory model uses multi-table pages to effective and secure paging.
  - Table-lookaside can improve memory access time.
- Next topic: Interprocess communication and threads