

MEMORY: SWAPPING, MIDTERM 1 REVIEW

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

Project 3 due **today!**

Code Review

Starting from Friday. Look for email from TA

10-15 mins meeting with TA

- Overall Code Structure
- Detailed explanation of a function in the solution
- Use of libraries within the solution
(including explanation of documentation on a used library call)

Rubric on Canvas (under Files → Uploaded Media)

MIDTERM 1 DETAILS

Oct 15th from 5.45pm to 7.15pm

Last name: A-K go to Van Vleck B102

Last name: L-Z go to Ingraham B10

Bring #2 Pencil, UW Student ID

One page cheat sheet (8.5x11 paper, two sided ok)

No calculator (we will give a table with powers of 2)

AGENDA / LEARNING OUTCOMES

Memory virtualization

How we support virtual mem larger than physical mem?

What are mechanisms and policies for this?

Midterm I Review

RECAP

Disk



Phys Memory



PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

What if access vpn 0xb?

VIRTUAL MEMORY MECHANISMS

First, hardware checks TLB for virtual address

- if TLB hit, address translation is done; page in physical memory

Else ...

- Hardware or OS walk page tables
- If PTE designates page is present, then page in physical memory (i.e., present bit is cleared)

Else

- Trap into OS (not handled by hardware)
- OS selects victim page in memory to replace
 - Write victim page out to disk if modified (use dirty bit in PTE)
- OS reads referenced page from disk into memory
- Page table is updated, present bit is set
- Process continues execution

PAGE REPLACEMENT POLICIES

OPT: Replace page **not used for longest time in future**

- Advantages: Guaranteed to minimize number of page faults
- Disadvantages: Requires that OS predict the future; Not practical,

FIFO: Replace page that has been in memory the longest

- Advantages: Fair: All pages receive equal residency; Easy to implement
- Disadvantage: Some pages may always be needed

LRU: Least-recently-used: Replace page not used for longest time in past

- Advantages: With locality, LRU approximates OPT
- Disadvantages:
 - Harder to implement, must track which pages have been accessed
 - Does not handle all workloads well

PAGE REPLACEMENT COMPARISON

Add more physical memory, what happens to performance?

LRU, OPT:

- Guaranteed to have fewer (or same number of) page faults
- Smaller memory sizes are guaranteed to contain a subset of larger memory sizes
- Stack property: smaller cache always subset of bigger

FIFO:

- Usually have fewer page faults
- Belady's anomaly: May actually have **more** page faults!

FIFO PERFORMANCE MAY DECREASE!

Consider access stream: ABCDABEABCDE

Physical memory size: 3 pages vs. 4 pages

How many misses with FIFO?

IMPLEMENTING LRU

Software Perfect LRU

- OS maintains ordered list of physical pages by reference time
- When page is referenced: Move page to front of list
- When need victim: Pick page at back of list
- Trade-off: Slow on memory reference, fast on replacement

Hardware Perfect LRU

- Associate timestamp register with each page
- When page is referenced: Store system clock in register
- When need victim: Scan through registers to find oldest clock
- Trade-off: Fast on memory reference, slow on replacement (especially as size of memory grows)

In practice

LRU is an approximation anyway, so approximate more?

CLOCK ALGORITHM

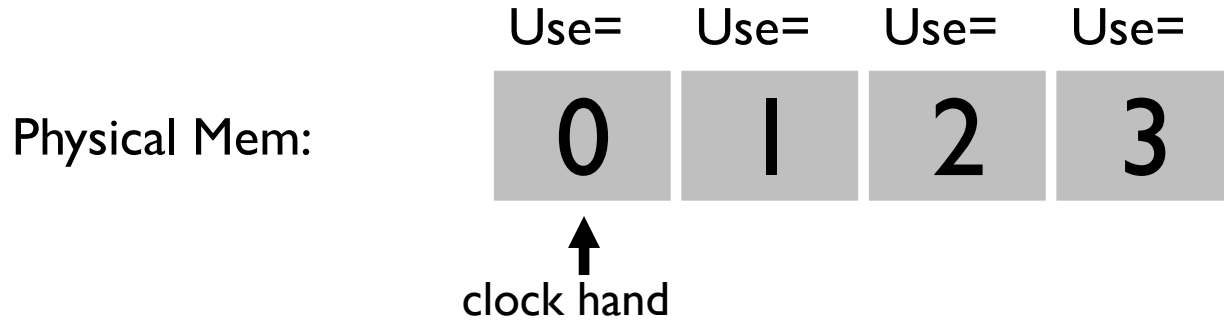
Hardware

- Keep use (or reference) bit for each page frame
- When page is referenced: set use bit

Operating System

- Page replacement: Look for page with use bit cleared (has not been referenced for awhile)
- Implementation:
 - Keep pointer to last examined page frame
 - Traverse pages in circular buffer
 - Clear use bits as search
 - Stop when find page with already cleared use bit, replace this page

CLOCK: LOOK FOR A PAGE



Use = 1,1,0,1 to begin

CLOCK EXTENSIONS

Replace multiple pages at once

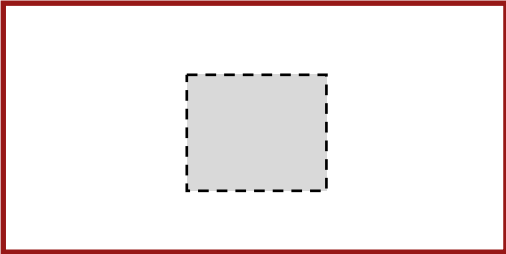
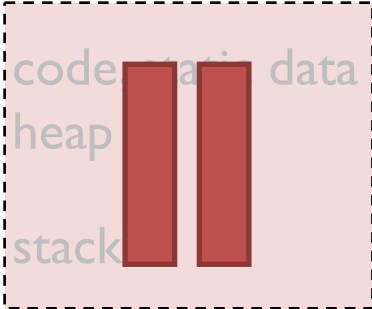
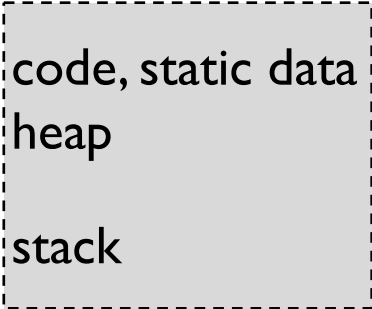
- Intuition: Expensive to run replacement algorithm and to write single block to disk
- Find multiple victims each time and track free list

Use dirty bit to give preference to dirty pages

- Intuition: More expensive to replace dirty pages
 - Dirty pages must be written to disk, clean pages do not
- Replace pages that have use bit and dirty bit cleared

MIDTERM 1 REVIEW

PROCESS AND TIME SHARING



CPU

HOW TO CREATE A PROCESS?

Unix-like OS use *fork()*

Fork() - Clones the calling process to create a child process

Make copy of code, data, stack etc.

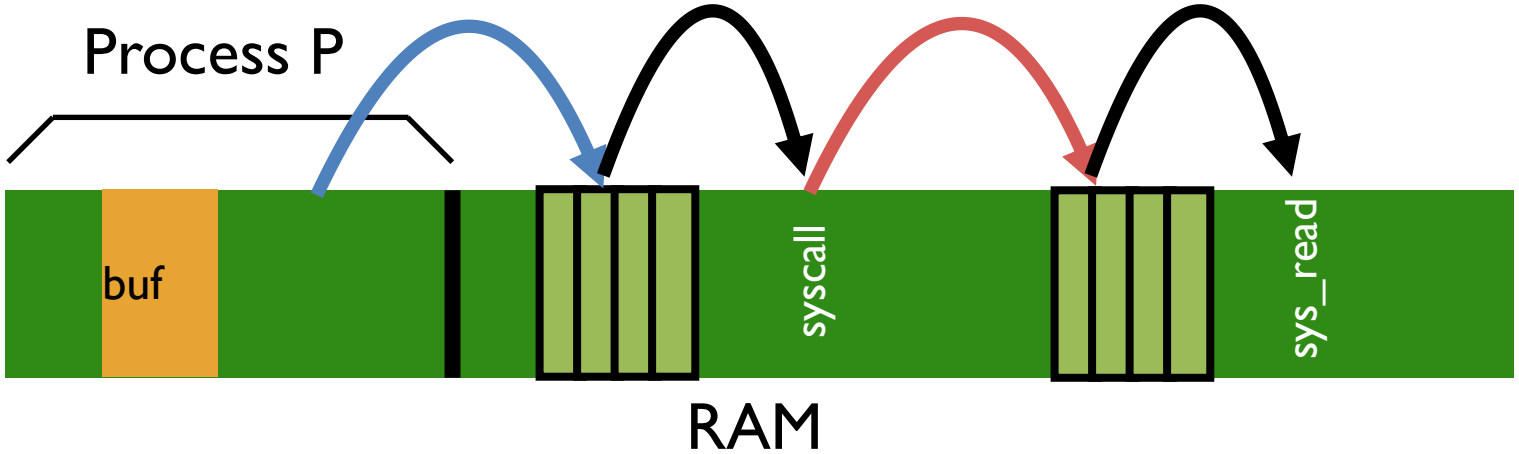
Add new process to ready list

Exec(char *file): Replace current data and code with file

Advantages: Flexible, clean, simple

Disadvantages: Wasteful to perform copy and overwrite of memory

SYSTEM CALL



Kernel can access user memory to fill in user buffer
return-from-trap at end to return to Process P

Operating System

Hardware

Program Process A

Handle the trap

Call switch() routine

save kernel regs(A) to proc-struct(A)

restore kernel regs(B) from proc-struct(B)

switch to k-stack(B)

return-from-trap (into B)

timer interrupt

save regs(A) to k-stack(A)

move to kernel mode

jump to trap handler

restore regs(B) from k-stack(B)

move to user mode

jump to B's IP

Process B

SCHEDULING METRICS, POLICIES

Turnaround time = $completion_time - arrival_time$

Response time = $first_run_time - arrival_time$

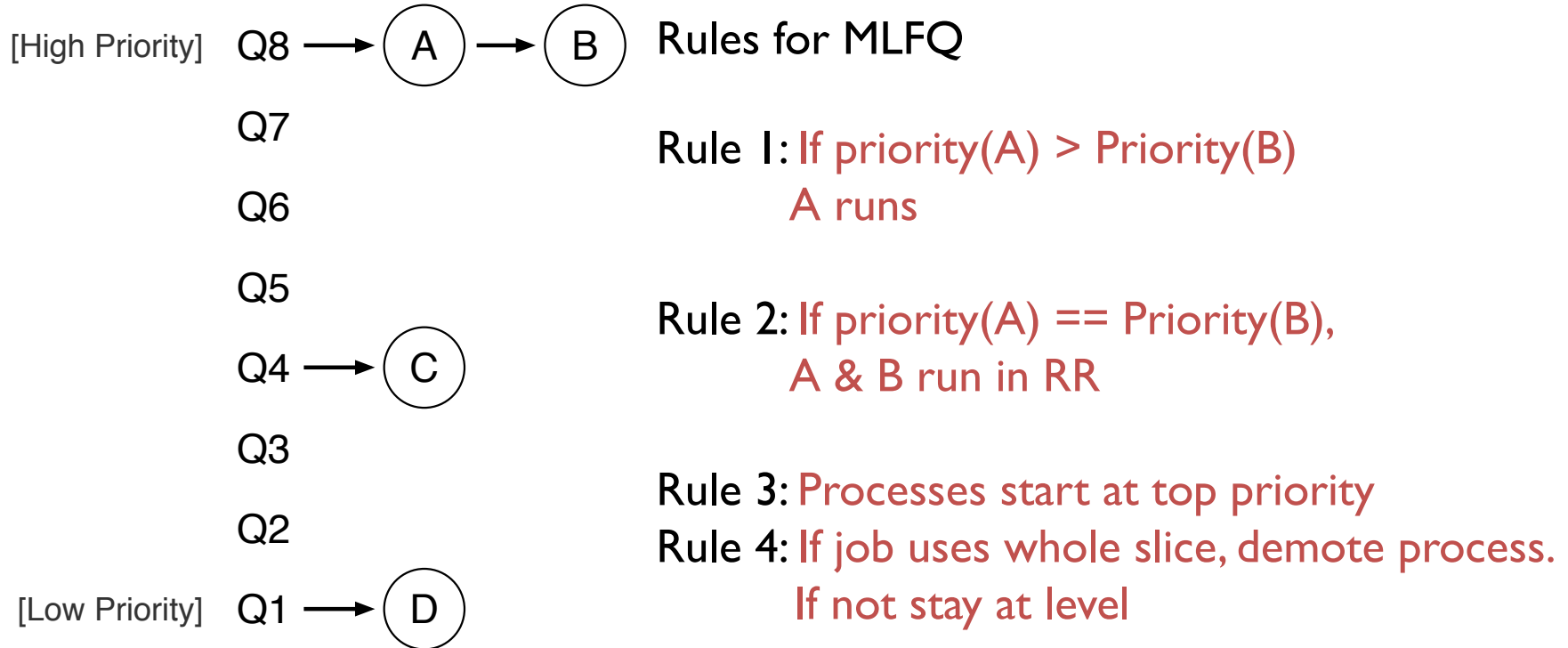
FIFO

Shortest Job First (SJF)

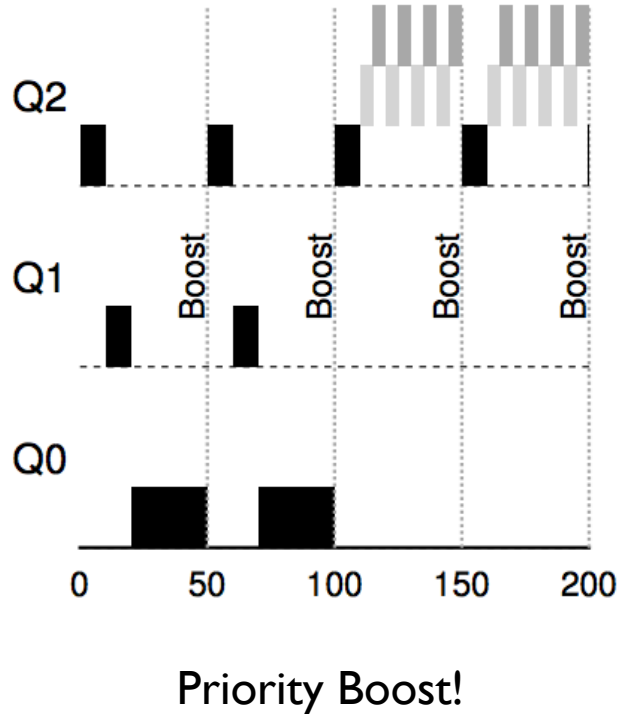
Shortest Time-to-Completion First (STCF)

Round Robin

MULTI-LEVEL FEEDBACK QUEUE



MORE MLFQ STARVATION



Rule 5: After some time period S , move all the jobs in the system to the topmost queue.

Job could trick scheduler by doing I/O just before time-slice end

Rule 4*: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced

PRACTICE QUESTIONS, SURVEY

<https://tinyurl.com/cs537-fa24-survey>



Assume a workload with the following characteristics. If needed, assume a time-slice of 1 second and if there is a choice the newest arriving job is selected.



Job Name	Arrival Time (seconds)	CPU Burst Time (seconds)
A	0	9
B	2	3
C	6	5

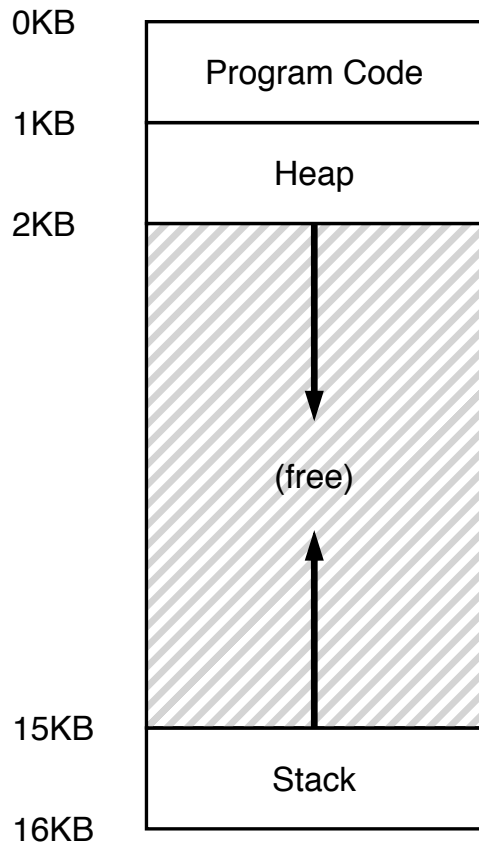
Given a FIFO scheduler, what is the turnaround time of job B?

Given a FIFO scheduler, what is the average turnaround time of the three jobs?

Given a SJF scheduler, what is the turnaround time of job C

Given a RR scheduler, what is the turnaround time of job B

ABSTRACTION, GOALS



Transparency: Process is unaware of sharing

Protection: Cannot corrupt OS or other process memory

Efficiency: Do not waste memory or slow down processes

Sharing: Enable sharing between cooperating processes

DYNAMIC RELOCATION MECHANISMS

Base Register

Translate virtual addresses to physical by adding a fixed offset each time.
Store offset in base register

Base+Bounds

Idea: limit the address space with a bounds register

Base register: smallest physical addr (or starting location)

Bounds register: size of this process's virtual address space

Disadvantages

- Each process must be allocated contiguously in physical memory
- No partial sharing: Cannot share parts of address space

SEGMENTATION

Process now specifies segment and offset within segment

How does process designate a particular segment?

Use part of logical address

Top bits of logical address select segment

Low bits of logical address select offset within segment

How many bits
for segment? 2

Segment	Base	Bounds	R W
0	0x2000	0x6fff	1 0
1	0x0000	0x4fff	1 1
2	0x3000	0xffff	1 1
3	0x0000	0x0000	0 0

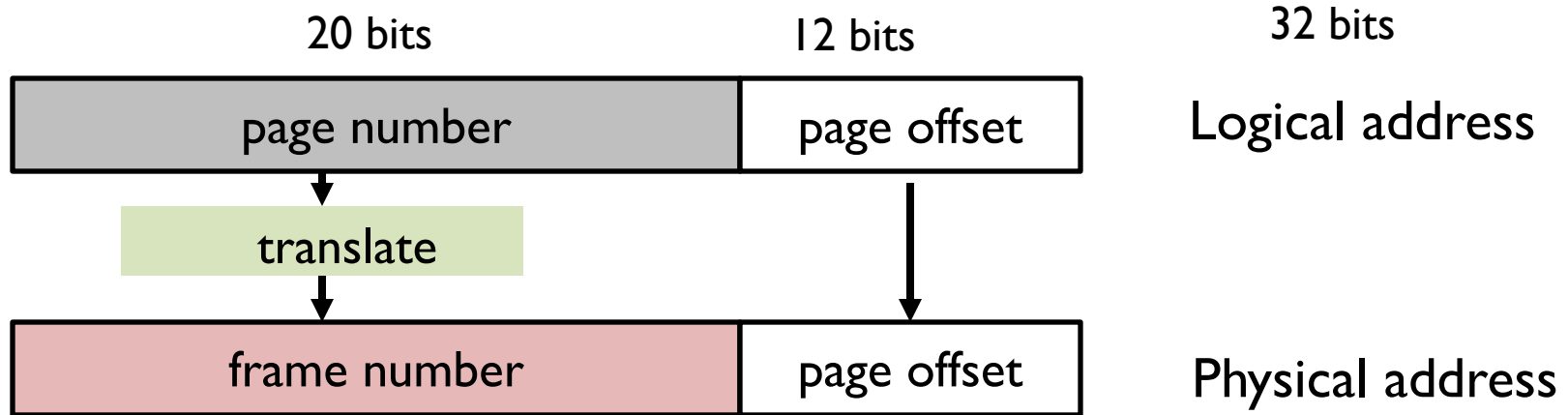
How many bits
for offset? 12

14 bit logical address
4 segments

PAGING

How to translate logical address to physical address?

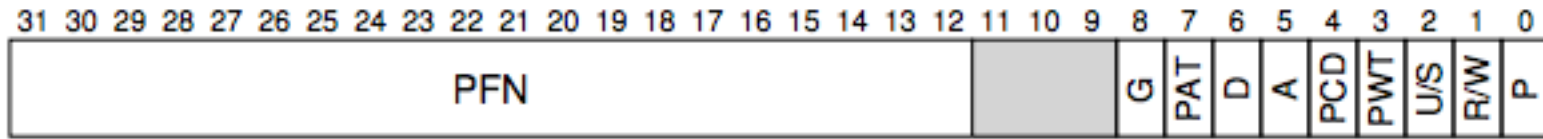
- High-order bits of address designate page number
- Low-order bits of address designate offset within page



No addition needed; just append bits correctly!

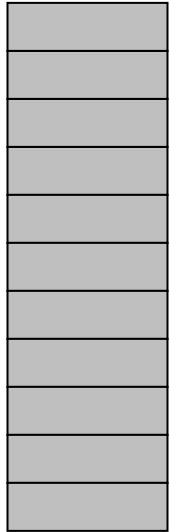
PAGETABLES

Per-Process Linear page table



VPN

0



2^n

Additional memory reference to page table →

Page table must be stored in memory
MMU stores only base address of page table

Storage for page tables may be substantial

Requires PTE for all pages in address space

Entry needed even if page not allocated ?

TLBS

Pages are great, but accessing page tables for every memory access is slow

Cache recent page translations → TLB

- MMU performs TLB lookup on every memory access

TLB performance depends strongly on workload

- Sequential workloads perform well
- Workloads with temporal locality can perform well

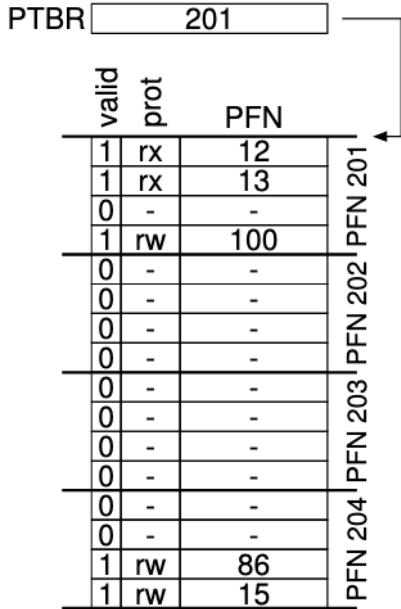
In different systems, hardware or OS handles TLB misses

TLBs increase cost of context switches

- Flush TLB on every context switch
- Add ASID to every TLB entry

MULTILEVEL PAGE TABLES

Linear Page Table

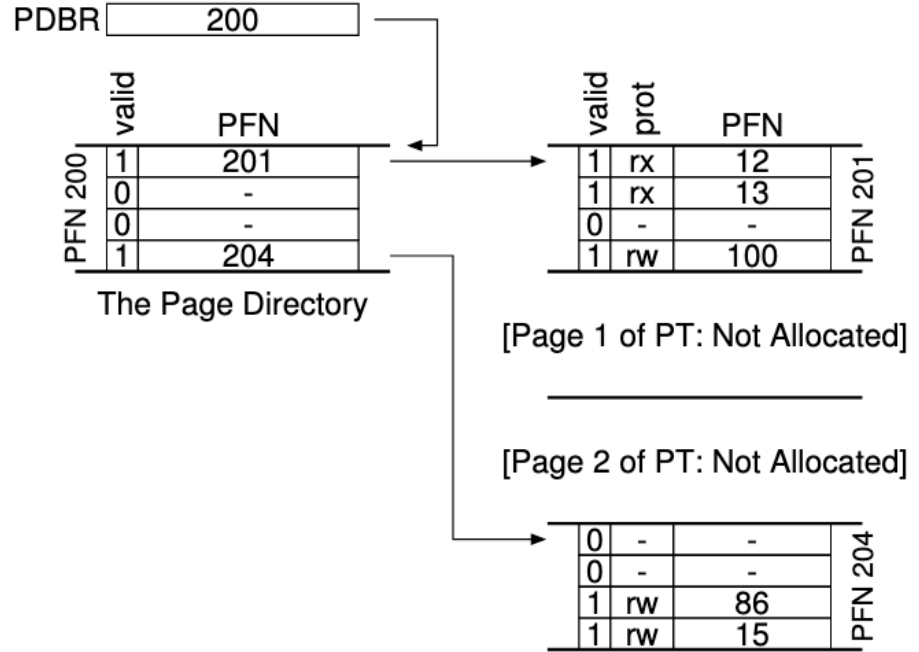


Creates multiple levels of page tables

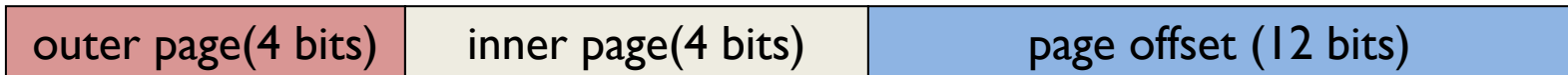
Only allocate page tables for pages in use

Allow page table to be allocated non-contiguously

Multi-level Page Table



20-bit address:



PRACTICE QUESTIONS

On a new system dynamic relocation is performed with a **linear page table**. Assume a system with the following parameters:

Address Space size is 32 KB

Physical Memory size is 128 KB

Page size is 4KB

You are given the following trace of virtual addresses and the physical addresses they translate to. Can you reverse engineer the contents of the page table for this process?

VA 0x00006e19 --> 0x0003e19

VA 0x00004d35 --> 0x000ad35

VA 0x000030d8 --> 0x00050d8

VA 0x0000244d --> 0x001a44d

VA 0x00005665 --> Invalid

Page Table Entry 0

Page Table Entry 2

On a new system dynamic relocation is performed with a **linear page table**. Assume a system with the following parameters:

Address Space size is 32 KB

Physical Memory size is 128 KB

Page size is 4KB

You are given the following trace of virtual addresses and the physical addresses they translate to. Can you reverse engineer the contents of the page table for this process?

VA 0x00006e19 --> 0x0003e19

VA 0x00004d35 --> 0x000ad35

VA 0x000030d8 --> 0x00050d8

VA 0x0000244d --> 0x001a44d

VA 0x00005665 --> Invalid

Page Table Entry 5

Page Table Entry 8

NEXT STEPS

Next class: New module on Concurrency!