Welcome back

MEMORY: SWAPPING, MIDTERM 1 REVIEW

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

Project 3 due today! → new test cases

Code Review

Starting from Friday. Look for email from TA → next week

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?

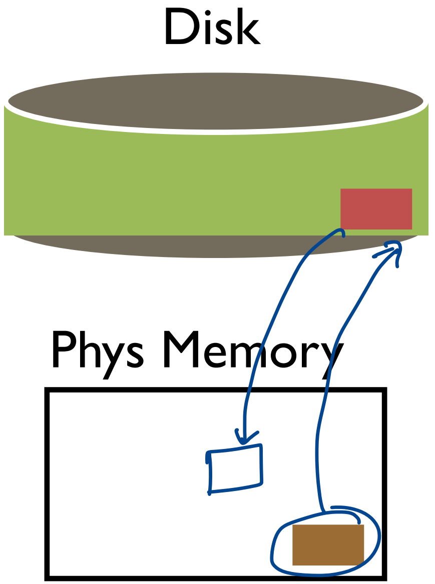
Midterm 1 Review

→ CPU virtualization

Memory virtualization

RECAP

Support apps where virtual memory > physical memory



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

PTE

addr on disk

17

page is on disk

in memory

What if access vpn 0xb?

Trap into OS

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

lookahead

PAGE REPLACEMENT POLICIES

When do we migrate
↳ Prefetch (sequential)

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 → Earliest in page is replaced

- 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 evict
↳ least recently used page

- 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

A B C D A B C

Misses □ □ □ □ >3

3 <3 ?

=3

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?

□ □ □

3 misses. ~~A~~ ~~B~~ C

miss. D B C

miss. D A C

miss. D A B

miss. E A B

miss E C B

miss E C D

9 misses

10 misses

□ □ □ □

4. A B C D

miss. E B C D

miss. E A C D

miss. E A B D

miss. E A B C

miss. D A B C

miss. D E B C

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

linked list

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?

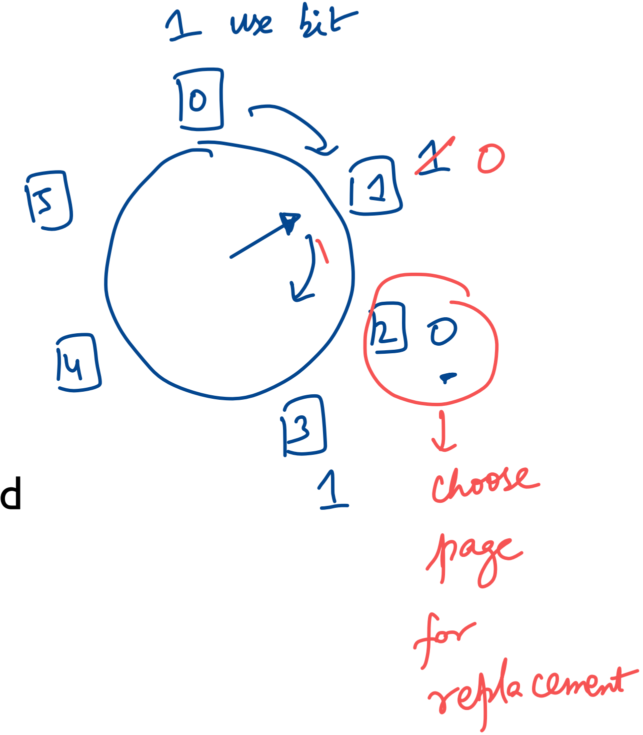
CLOCK ALGORITHM

Hardware

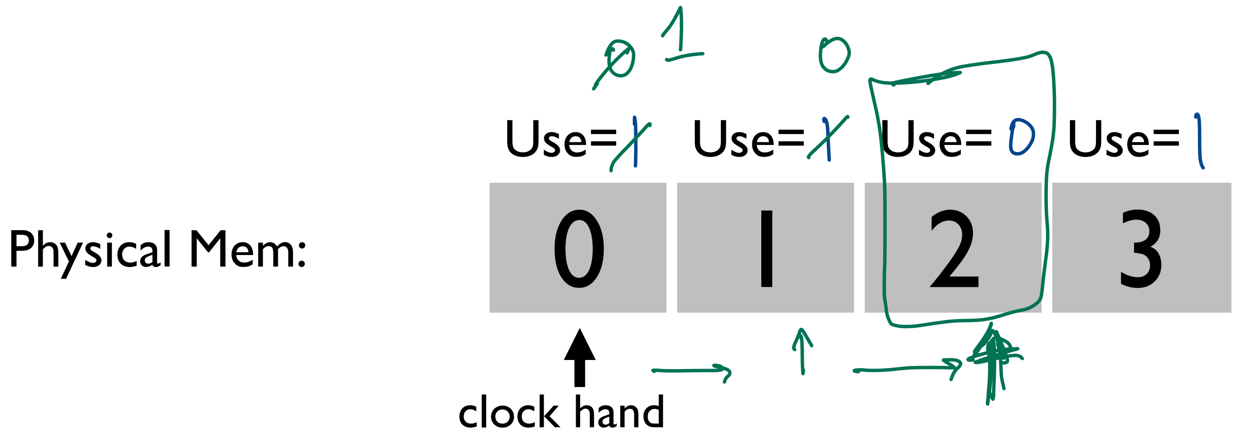
- Keep use (or reference) bit for each page frame
- When page is referenced: set use bit

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

select page 2 for replacement

Page 0 is accessed → set its use bit to 1

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 which are not dirty

- 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

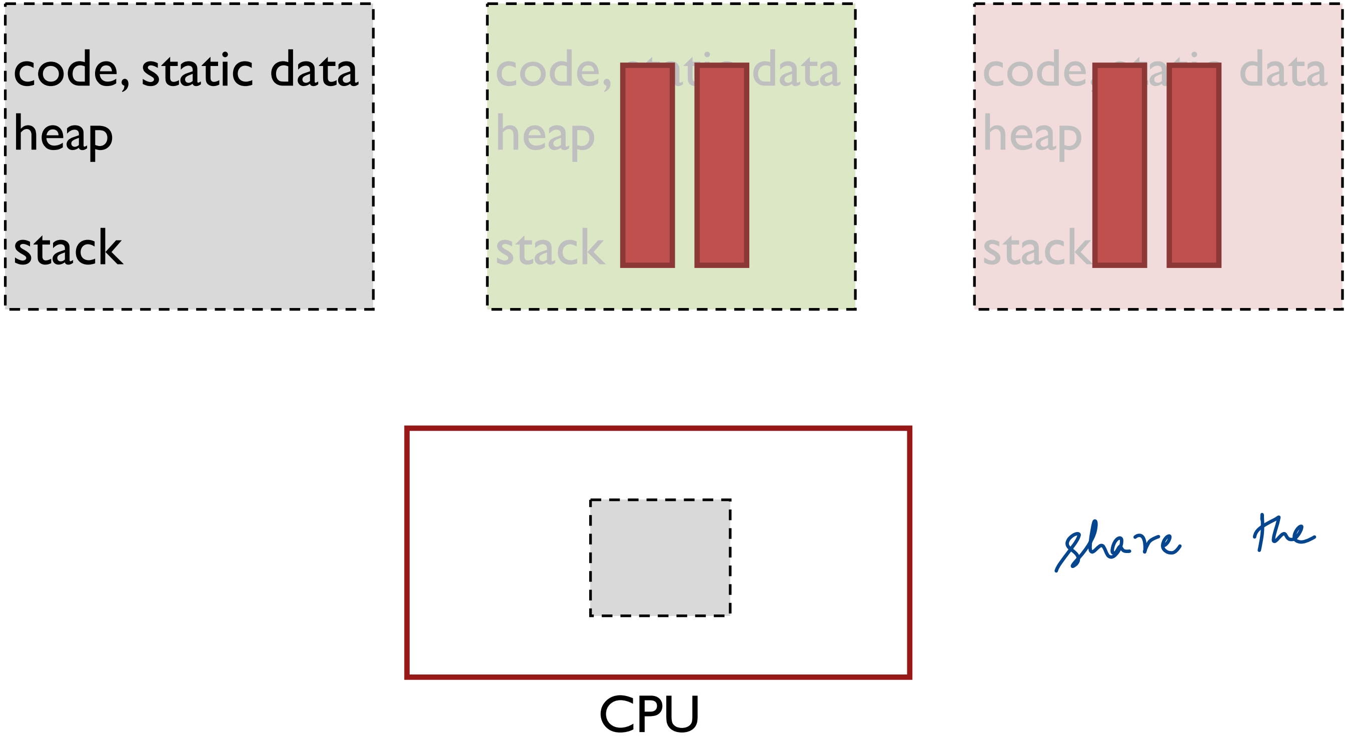
→ modified in memory

MIDTERM 1 REVIEW

PROCESS AND TIME SHARING

CPU virtualization

→ Process abstraction



share the 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

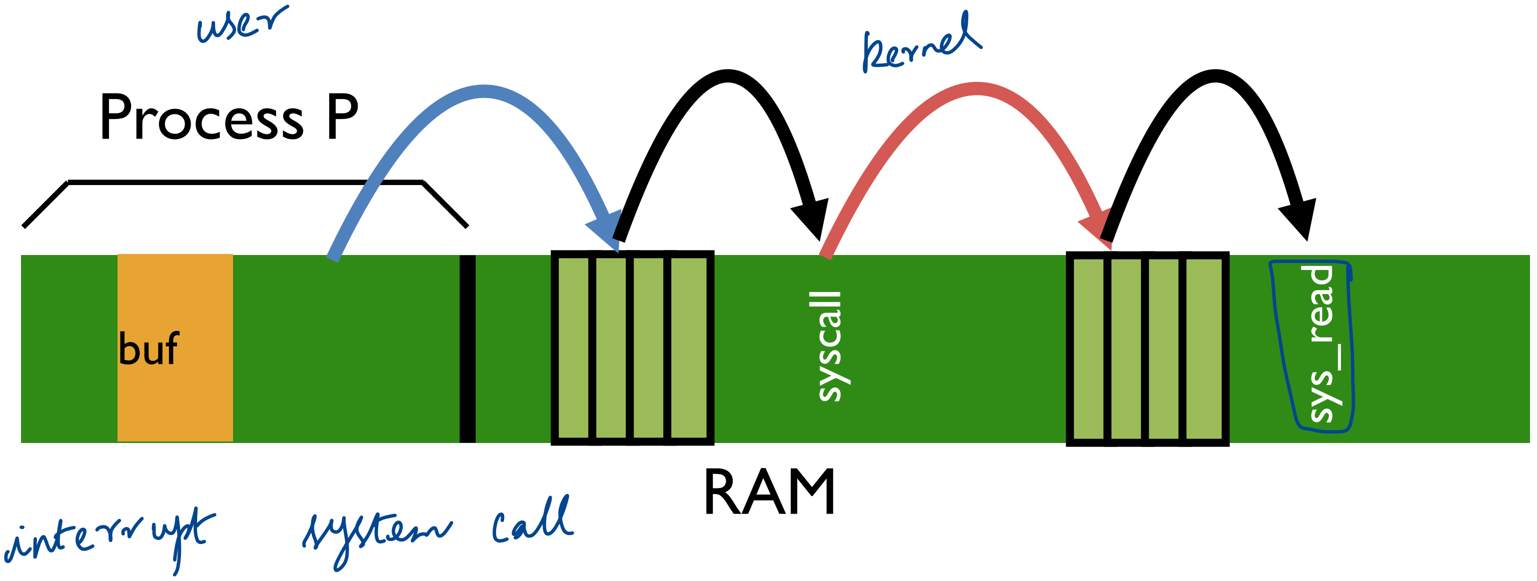
same memory contents as parent

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

time sharing

timer interrupt
save regs(A) to k-stack(A)
move to kernel mode
jump to trap handler

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)

choose a diff process

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)

→ not premptive

Shortest Time-to-Completion First (STCF)

→ interrupt existing job

Round Robin

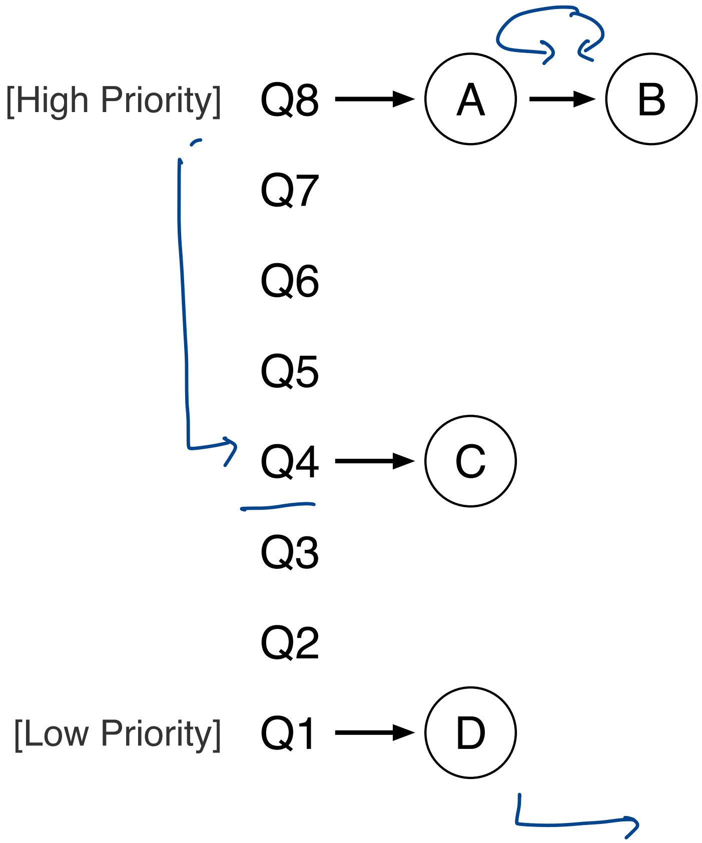
→ time slice

MULTI-LEVEL FEEDBACK QUEUE

Round robin

interactive jobs

batch jobs



starvation

Rules for MLFQ

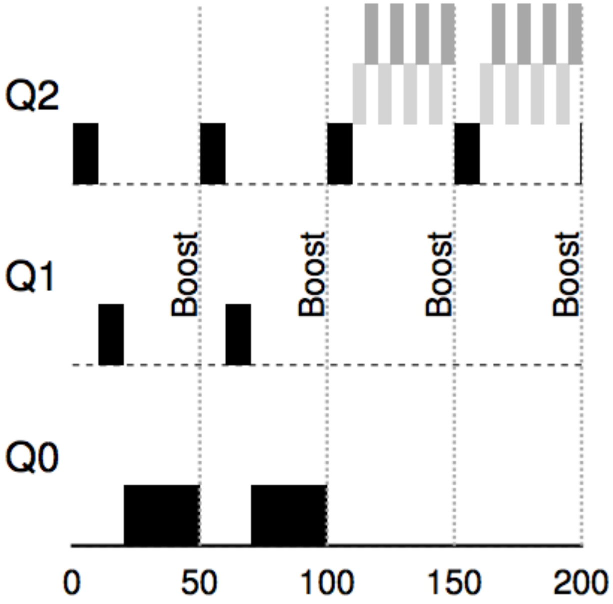
Rule 1: If priority(A) > Priority(B)
A runs

Rule 2: If priority(A) == Priority(B),
A & B run in RR

Rule 3: Processes start at top priority

Rule 4: If job uses whole slice, demote process.
If not stay at level

MORE MLFQ STARVATION



Priority Boost!

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-survey1

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. → tie breaker

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

RR

A A B A B A C

B A

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

A . . . . . . A B B B C C C C C

0 9 10 11 12 17

12 − 2 = 10

finish arrival

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

A = 9

B = 10

C = 17 − 6 = 11

$$\text{Avg} = \frac{9+10+11}{3} = 10$$

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

↳ not premptive

$\underbrace{A \ldots \ldots A}_{9}$ B B B C C

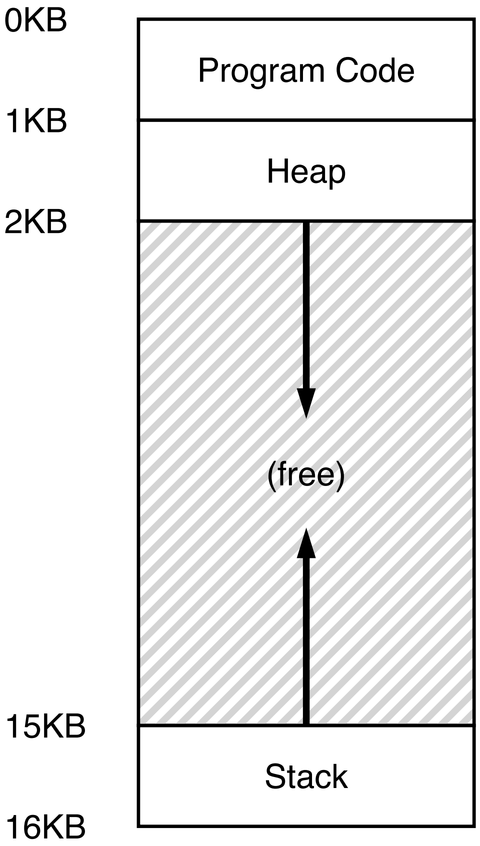
↳ B is shorter

11 → same

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

$\boxed{8 - 2 = 6}$

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

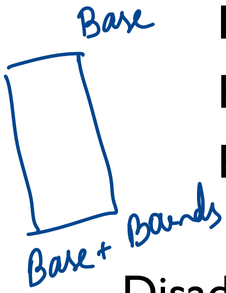
DYNAMIC RELOCATION MECHANISMS

Base Register

Translate virtual addresses to physical by adding a fixed offset each time.

Store offset in base register → Protection

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

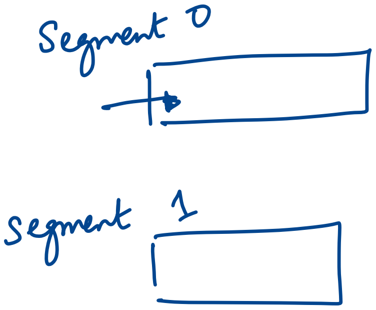
↗ check if virtual addr is less than bound

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

Add the offset to base

How many bits for segment? 2

How many bits for offset? 12

Segment	Base	Bounds	R W
0	0x2000	0x6ff	1 0
1	0x0000	0x4ff	1 1
2	0x3000	0xfff	1 1
3	0x0000	0x000	0 0

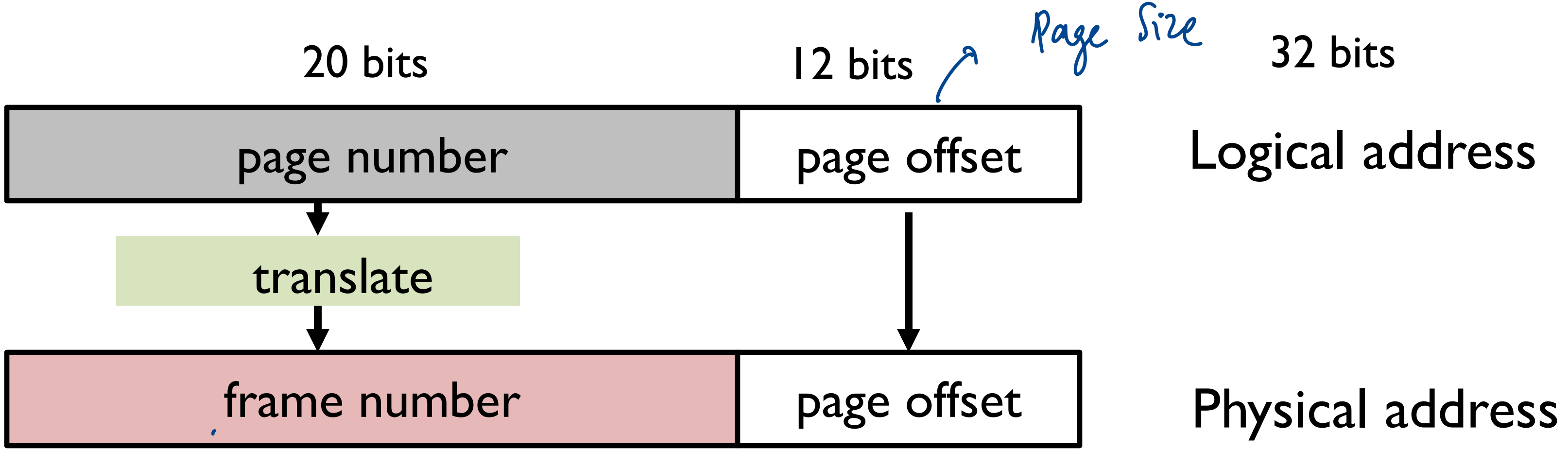
14 bit logical address
4 segments

PAGING

fixed size page

How to translate logical address to physical address?

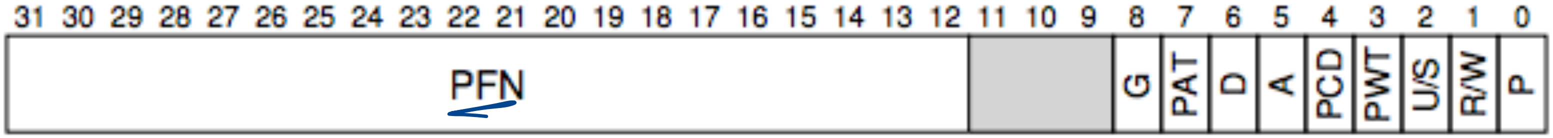
- High-order bits of address designate page number
- Low-order bits of address designate offset within page

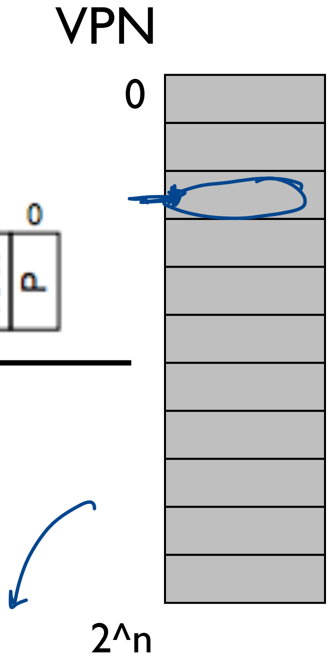


No addition needed; just append bits correctly!

PAGETABLES

Per-Process Linear page table





for every process

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 → cache VPN → PPN mappings in MMU

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

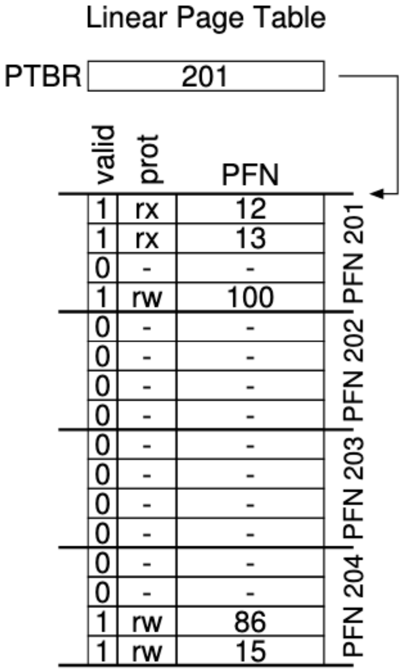
In different systems, hardware or OS handles TLB misses

simple page table for hardware managed TLB

TLBs increase cost of context switches

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

MULTILEVEL PAGE TABLES



Linear Page Table

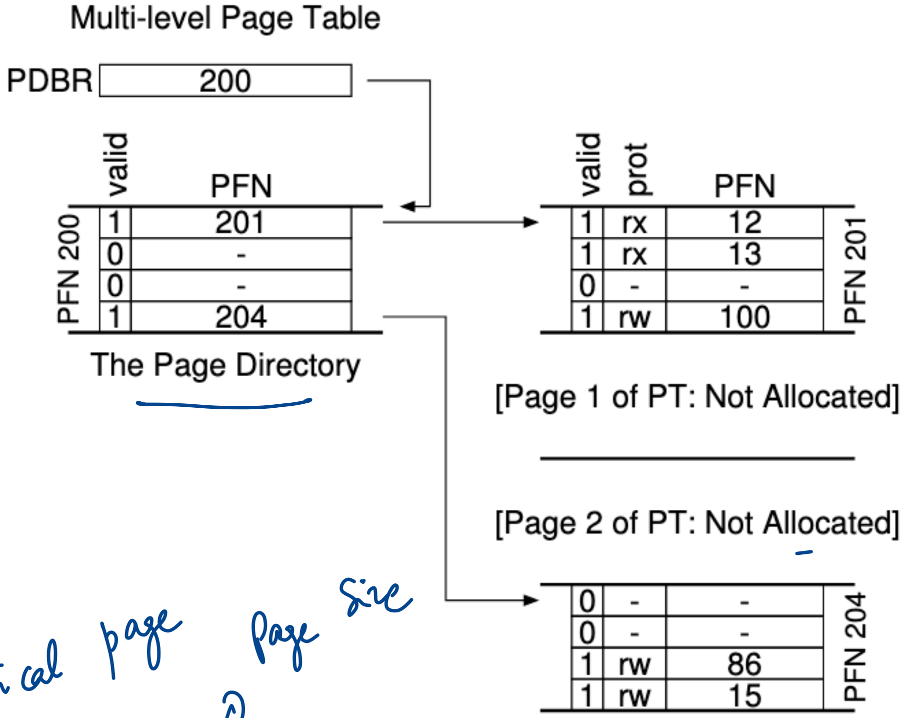
PTBR 201

valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	
0	-	-	PFN 202
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 203
0	-	-	
0	-	-	
0	-	-	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

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

PDBR 200

valid	PFN	
1	201	PFN 200
0	-	
0	-	
1	204	

The Page Directory

valid	prot	PFN	
1	rx	12	PFN 201
1	rx	13	
0	-	-	
1	rw	100	

[Page 1 of PT: Not Allocated]

[Page 2 of PT: Not Allocated]

valid	prot	PFN	
0	-	-	PFN 204
0	-	-	
1	rw	86	
1	rw	15	

Number of PTEs in 1 physical page

Page size

20-bit address:

outer page(4 bits)	inner page(4 bits)	page offset (12 bits)

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 VPN 6 → PPN 3
VA 0x00004d35 --> 0x000ad35 offset = 12 bit
VA 0x000030d8 --> 0x00050d8
VA 0x0000244d --> 0x001a44d
VA 0x00005665 --> Invalid

Page Table Entry 0 → VPN = 0 → Cannot determine

Page Table Entry 2 → VPN = 2 0x 1a

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 → 8 valid virtual pages (0 to 7)
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 Invalid

Page Table Entry 8 not a valid VPN

NEXT STEPS

Next class: New module on Concurrency!