UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

Exam 1: Review

Questions answered in this lecture:

What are some useful things to remember about virtualization?

Announcements

P1: Graded in Learn@UW; If major surprises see your TA (or 537-help@cs)

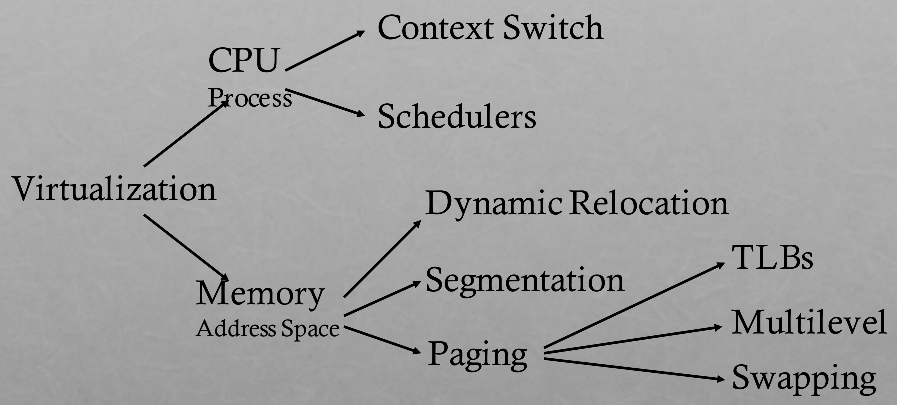
P2:

- Pace: Good to have finished Shell (Part A) by now
- Spend more time on Scheduler (Part B)
  - Purpose of graph is to demonstrate all aspects of scheduler are working correctly

Exam

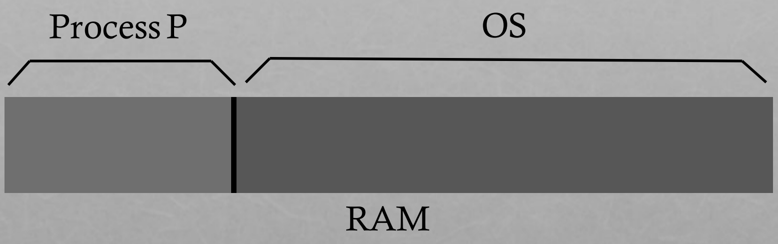
- Two hours – 7:15 – 9:15 pm in 272 Bascom Hall
- Bring #2 pencils and student id
- All multiple choice
- Covers everything so far in course:
  - Lectures + Reading + Homework + Project 1
  - Chapters 1 - 24, excluding 10 (Multiprocessor Scheduling), 17 (Free-Space Management), and 23 (VAX/VMS Virtual Memory System)
  - Look over sample exams

Review: Easy Piece 1



What questions did you ask?

How are System Calls Performed?

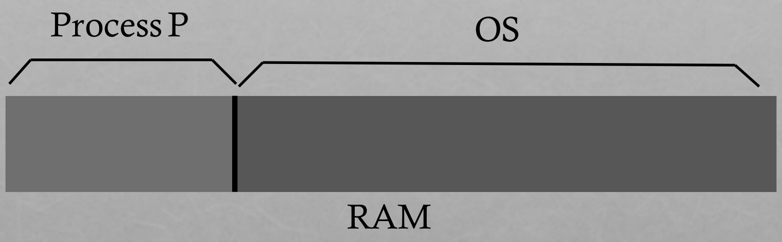


P can only see its own memory because of **user mode**
(other areas, including kernel, are hidden)

P wants to call read() but no way to call it directly

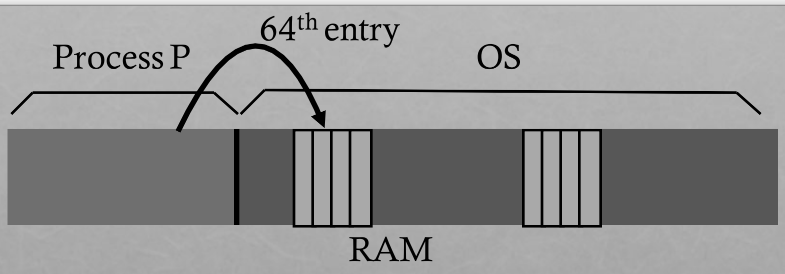
OS is not part of P's address space

System Call



read():
```
movl $6, %eax;   int $64
```

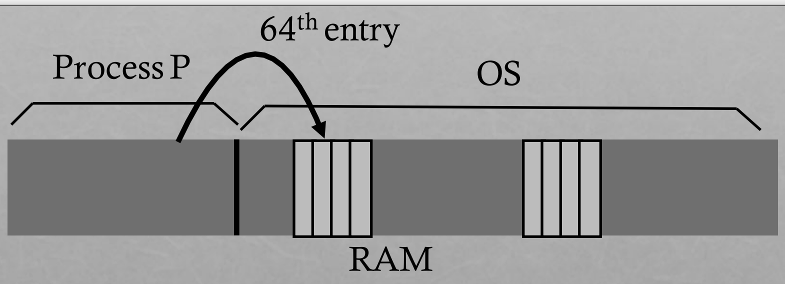
System Call



```
movl $6, %eax;   int $64
```

syscall-table index   trap-table index

System Call

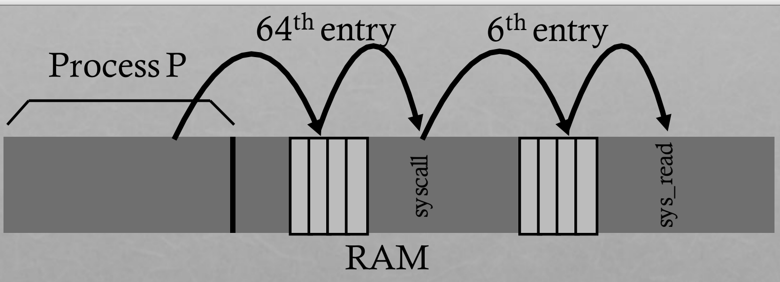


```
movl $6, %eax;   int $64
```

syscall-table index   trap-table index

Kernel mode: we can do anything!

System Call

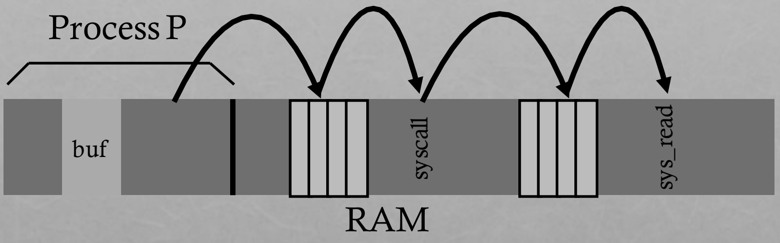


```
movl $6, %eax;   int $64
```

syscall-table index   trap-table index

Follow entries to correct system call code

System Call

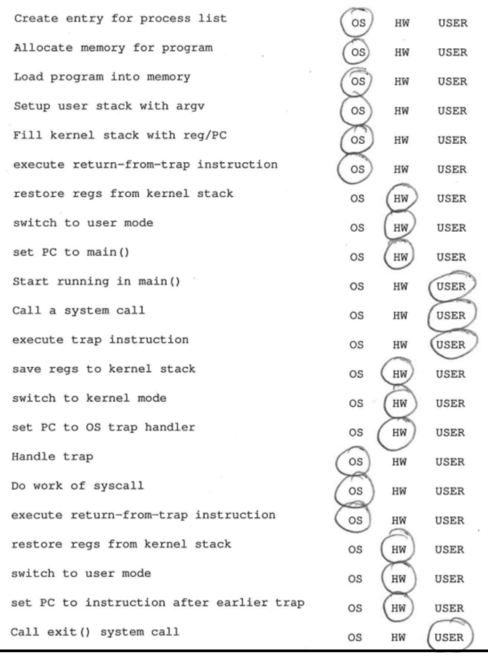


```
movl $6, %eax;   int $64
```

syscall-table index   trap-table index

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

HW, OS, or User Process?



Create entry for process list	[✓ OS]	HW	USER
Allocate memory for program	[✓ OS]	HW	USER
Load program into memory	[✓ OS]	HW	USER
Setup user stack with argv	[✓ OS]	HW	USER
Fill kernel stack with reg/PC	[✓ OS]	HW	USER
execute return-from-trap instruction	[✓ OS]	HW	USER
restore regs from kernel stack	OS	[✓ HW]	USER
switch to user mode	OS	[✓ HW]	USER
set PC to main()	OS	[✓ HW]	USER
Start running in main()	OS	HW	[✓ USER]
Call a system call	OS	HW	[✓ USER]
execute trap instruction	OS	HW	[✓ USER]
save regs to kernel stack	OS	[✓ HW]	USER
switch to kernel mode	OS	[✓ HW]	USER
set PC to OS trap handler	OS	[✓ HW]	USER
Handle trap	[✓ OS]	HW	USER
Do work of syscall	[✓ OS]	HW	USER
execute return-from-trap instruction	[✓ OS]	HW	USER
restore regs from kernel stack	OS	[✓ HW]	USER
switch to user mode	OS	[✓ HW]	USER
set PC to instruction after earlier trap	OS	[✓ HW]	USER
Call exit() system call	OS	HW	[✓ USER]

Process API: HW in Book

Write a program using fork(). The child process should print "hello"; the parent process should print "goodbye". You should try to ensure that the child process always prints first; can you do this without calling wait() in the parent?

- Waitpid, sleep, other synchronization primitives such as condition variables and semaphores (next topic!)

Is it possible for child process to wait for a parent or does it always have to be the other way around?

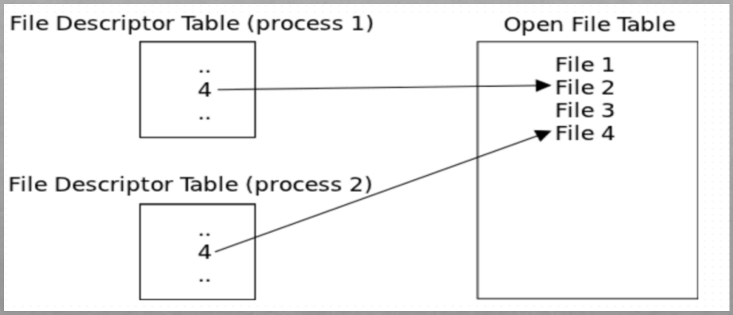
- Wait() and waitpid() apply to children processes

Typical workflow of creating a new process is to call exec in child after forking. Would there ever be a reason to create a child and call exec in the parent instead?

- No good reason I can think of

Process API

If a parent and a child can access the same file descriptor, why does closing a file descriptor in a child not effect the parent process? Is it just because the file descriptor table is unique for each, but each entry references the same file?

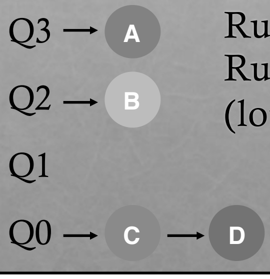

File Descriptor Table (process 1)
..
4
..
Open File Table
File 1
File 2
File 3
File 4
File Descriptor Table (process 2)
..
4
..

Multi-level Feedback Queue (MLFQ) Rules

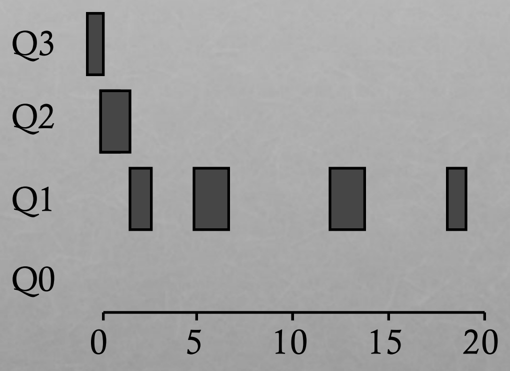
Rule 1: If priority(A) > Priority(B),
A runs

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

More rules:
Rule 3: Processes start at top priority
Rule 4: If job uses whole slice, demote process (longer time slices at lower priorities)



Job that performs I/O periodically



Stays in Q1 queue as long as doesn't use entire Q1 timeslice

Prevent Gaming

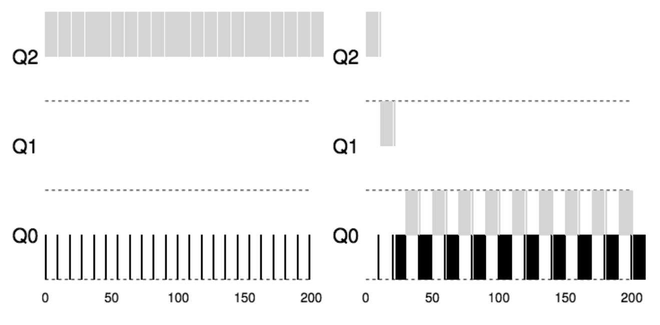


Figure 8.6: **Without (Left) and With (Right) Gaming Tolerance**

Problem: High priority job could trick scheduler and get mor
CPU by performing I/O right before time-slice ends

Fix: Account for process's total run time at priority level downgrade when exceed threshold

How are virtual addresses generated?

- What do addresses look like from the program's perspective? (from the user process's perspective)
- Generated by compiler and contents of registers

Quiz: Memory Accesses?

Initial %rip = 0x10
%rbp = 0x200

```
0x10:  movl  0x8(%rbp),  %edi
0x13:  addl  $0x3,  %edi
0x19:  movl  %edi,  0x8(%rbp)
```

%rbp is the base pointer:
points to base of current stack frame

%rip is instruction pointer (or program counter, PC)

Memory Accesses to what virtual addresses?

1) Fetch instruction at addr 0x10
Exec:

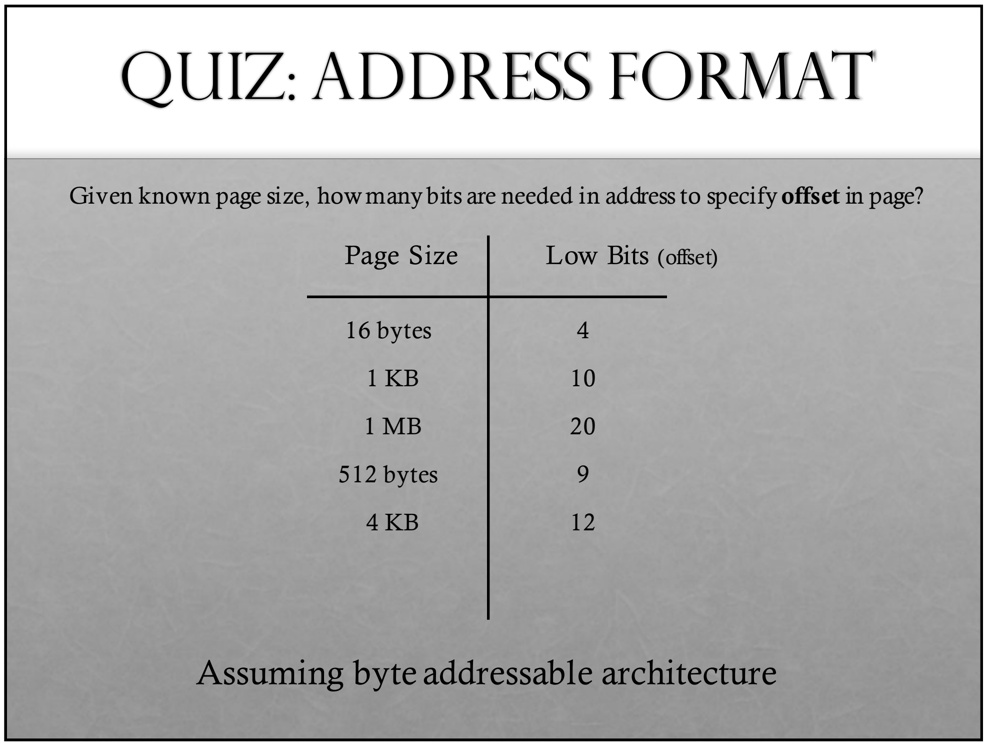
2) load from addr 0x208

3) Fetch instruction at addr 0x13
Exec:

no memory access

4) Fetch instruction at addr 0x19
Exec:

5) store to addr 0x208

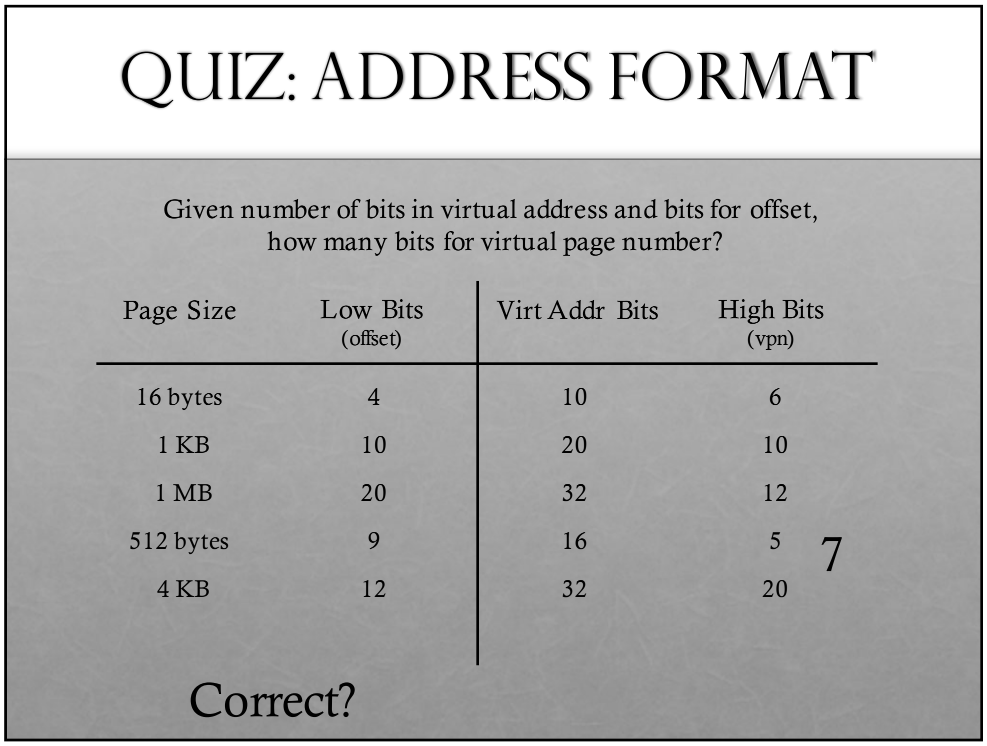


Quiz: Address Format

Given known page size, how many bits are needed in address to specify **offset** in page?

Page Size	Low Bits (offset)
16 bytes	4
1 KB	10
1 MB	20
512 bytes	9
4 KB	12

Assuming byte addressable architecture



Quiz: Address Format

Given number of bits in virtual address and bits for offset, how many bits for virtual page number?

Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)
16 bytes	4	10	6
1 KB	10	20	10
1 MB	20	32	12
512 bytes	9	16	5 7
4 KB	12	32	20

Correct?

Quiz: Address Format

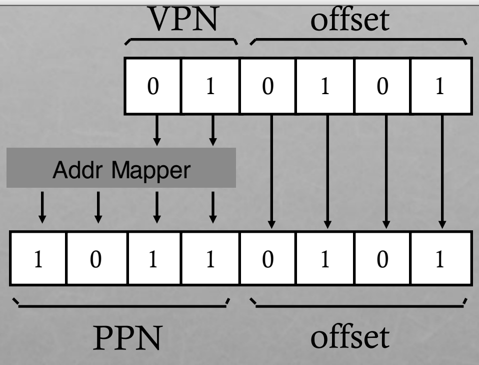
Given number of bits for vpn,
how many virtual pages can there be in an address space?

Page Size	Low Bits (offset)	Virt Addr Bits	High Bits (vpn)	Virt Pages
16 bytes	4	10	6	64
1 KB	10	20	10	1 K
1 MB	20	32	12	4 K
512 bytes	9	16	7	128
4 KB	12	32	20	1 M

Tells you how many entries are needed in page tables!

Virtual => Physical Page Mapping

Number of bits in virtual address format does not need to equal number of bits in physical address format



How should OS translate VPN to PPN?

For segmentation, OS used a formula (e.g., phys addr = virt_offset + base_reg)

For paging, OS needs more general mapping mechanism

What data structure is good? Big array: pagetable

Where are pagetables stored?

How big is a typical page table?
- assume **32-bit** address space
- assume 4 KB pages
- assume 4 byte page table entries (PTEs)

Final answer: 2 ^ (32 - log(4KB)) * 4 = **4 MB**
- Page table size = Num entries * size of each entry
- Num entries = num virtual pages = 2^(bits for vpn)
- Bits for vpn = 32– number of bits for page offset
  = 32 – lg(4KB) = 32 – 12 = 20
- Num entries = 2^20 = 1 MB
- Page table size = Num entries * 4 bytes = 4 MB

Implication: Store each page table in memory
- Hardware finds page table base with register (e.g., CR3 on x86)

What happens on a context-switch?
- Change contents of page table base register to newly scheduled process
- Save old page table base register in PCB of descheduled process

Quiz: How big are page tables?

How big is each page table?

1. PTE's are **2 bytes**, and **32** possible virtual page numbers

   32 * 2 bytes = 64 bytes
2. PTE's are **2 bytes**, virtual addrs are **24 bits**, pages are **16 bytes**

   2 bytes * 2^(24 – lg 16) = **2^21 bytes** (2 MB)
3. PTE's are **4 bytes**, virtual addrs are **32 bits**, and pages are **4 KB**

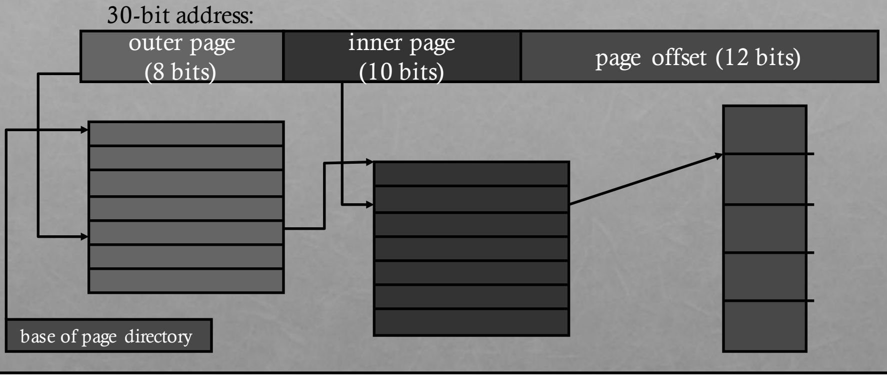
   4 bytes * 2^(32 – lg 4K) = **2^22 bytes** (4 MB)
4. PTE's are **4 bytes**, virtual addrs are **64 bits**, and pages are **4 KB**

   4 bytes * 2^(64 – lg 4K) = **2^54 bytes**

3) Multilevel Page Tables

Goal: Allow each page table to be allocated non-contiguously

Idea: Page the page tables
- Creates multiple levels of page tables; outer level "page directory"
- Only allocate page tables for pages in use
- Used in x86 architectures (hardware can walk known structure)



Quiz: Multilevel

page directory		page of PT (@PPN:0x3)		page of PT (@PPN:0x92)	
PPN	valid	PPN	valid	PPN	valid
0x3	1	0x10	1	-	0
-	0	0x23	1	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	0x80	1	-	0
-	0	0x59	1	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	-	0	-	0
-	0	-	0	0x55	1
0x92	1	0x25	1	0x45	1

translate 0xfffff

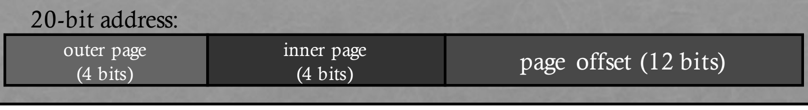
0x45fff

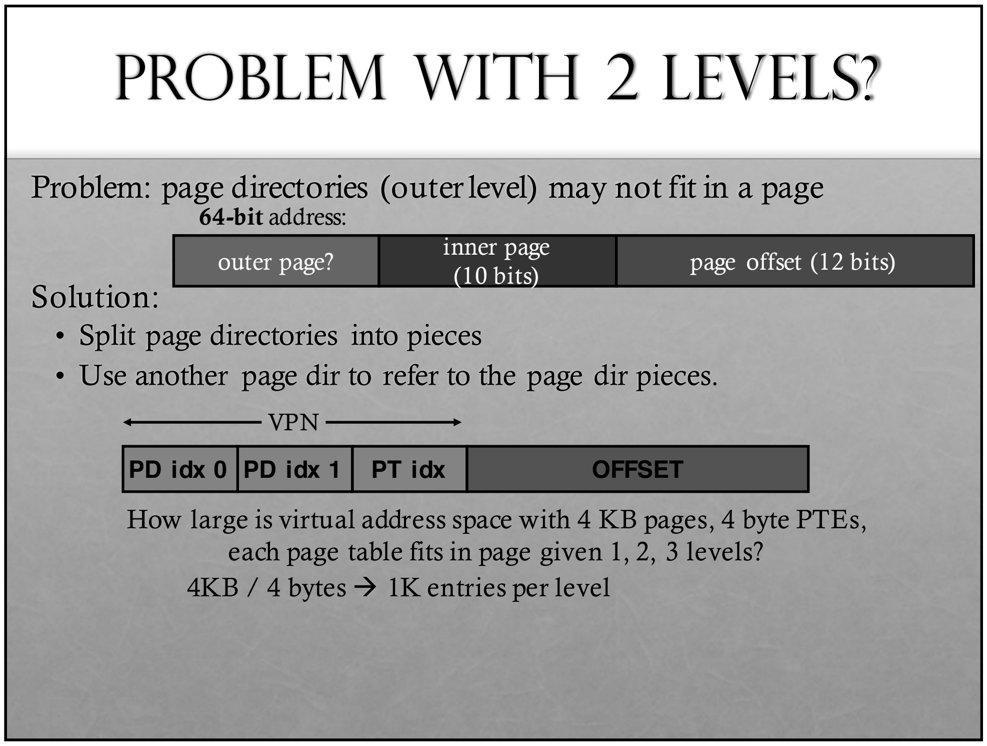
translate 0x00000

0x10000

translate 0x0ffff

0x25fff





Problem with 2 levels?

Problem: page directories (outer level) may not fit in a page

64-bit address:

outer page?	inner page (10 bits)	page offset (12 bits)

Solution:

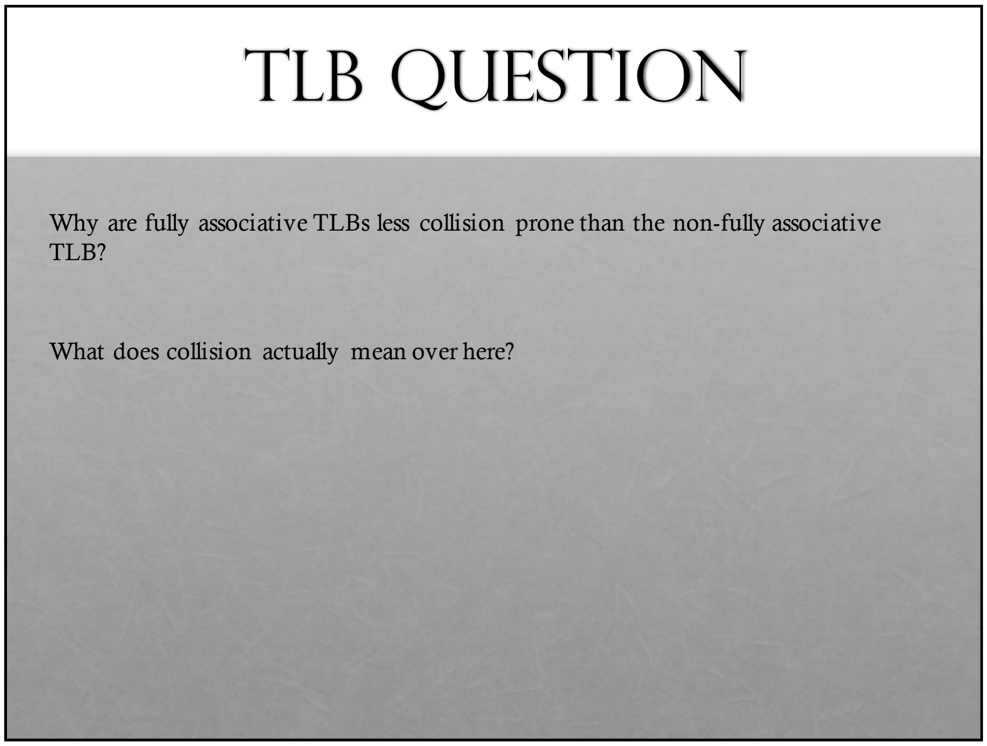
- Split page directories into pieces
- Use another page dir to refer to the page dir pieces.

←— VPN —→

PD idx 0	PD idx 1	PT idx	OFFSET

How large is virtual address space with 4 KB pages, 4 byte PTEs, each page table fits in page given 1, 2, 3 levels?

4KB / 4 bytes → 1K entries per level

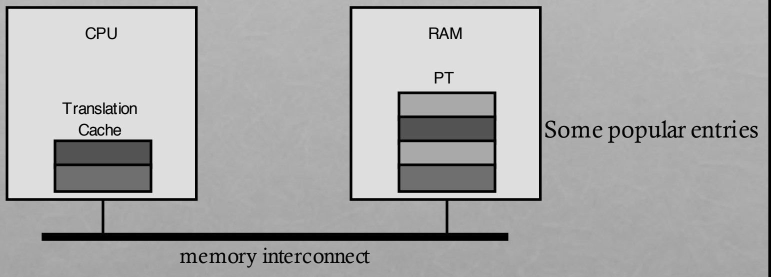


TLB Question

Why are fully associative TLBs less collision prone than the non-fully associative TLB?

What does collision actually mean over here?

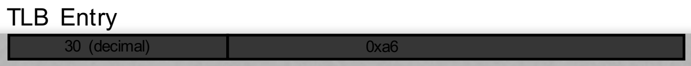
Translation Lookaside Buffer (TLB)



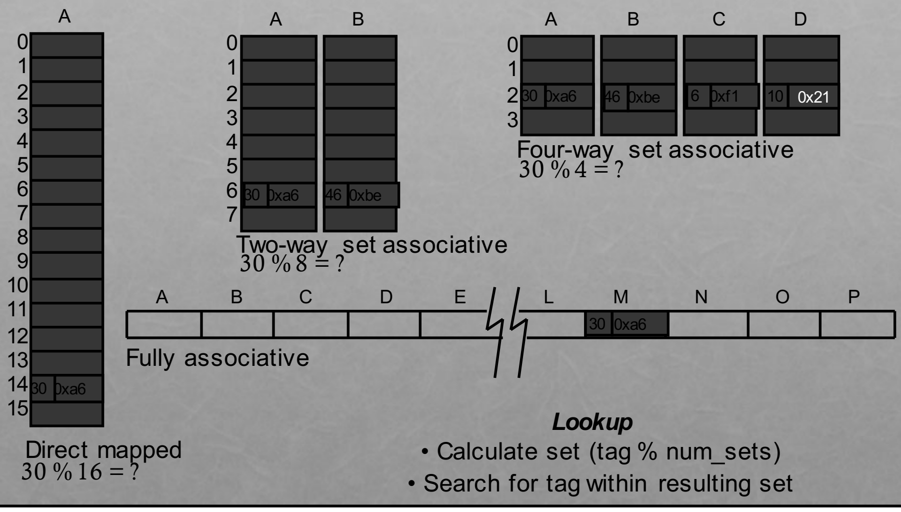
TLB: **T**ranslation **L**ookaside **B**uffer

(this is special hardware!)

TLB Example



Various ways to organize a 16-entry TLB (artificially small)



Lookup

- Calculate set (tag % num_sets)
- Search for tag within resulting set

TLB Associativity Trade-offs

Higher associativity
+ Better utilization, fewer collisions (or conflicts)
– Slower
– More hardware

Lower associativity
+ Fast
+ Simple, less hardware
– Greater chance of collisions (or conflicts)

TLBs usually fully associative

Present vs Valid Bit

- Virtual memory when page is not allocated in physical memory (RAM); instead on disk
- Why is a present bit needed? Why not just use valid bit?

Virtual Address Space Mechanisms

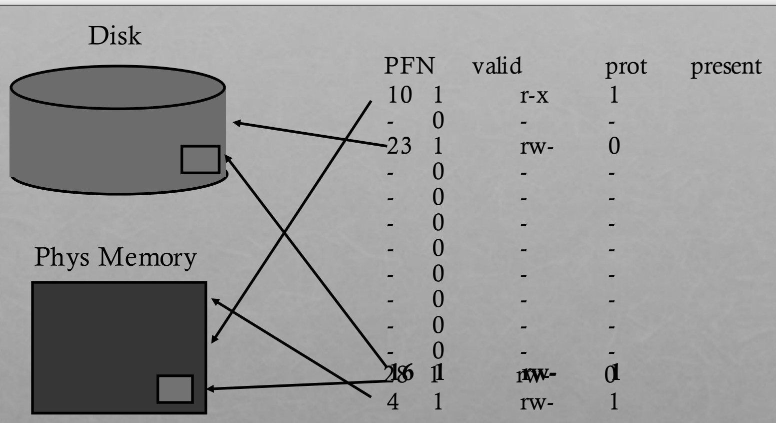
Each page in virtual address space maps to one of three:

- Nothing (error): Free
- Physical main memory: Small, fast, expensive
- Disk (persistent storage): Large, slow, cheap

Extend page tables with an extra bit: `present`

- `permissions (r/w), valid, present`
- Page is not allocated or mapped (not `valid`)
  - Segmentation fault
- Page in memory: `present` bit set in PTE, hold PPN
- Page on disk: `present` bit cleared
  - PTE points to **block address on disk**
  - Causes trap into OS when page is referenced
  - **Trap: page fault**

Present Bit



PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28/16	1	rw-	0/1
4	1	rw-	1

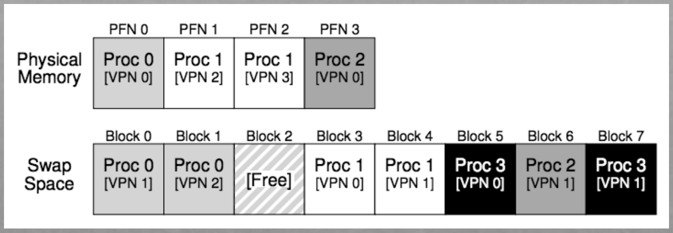
What if access vpn 0xb?

Swapping

Assume: when process starts, all the code that runs has to be loaded in from the disk due to page faults occurring, is this correct?

- Yes, with pure demand paging

Why in diagram 21.1 is proc0's VPN 0 page in memory but not on the disk? Wouldn't proc0's VPN 0 page still be on the disk except it was also copied into main memory?



Good Luck!

- TAs may review for exam more in discussion section or might go over Project material
  - Use form if you care!