CSE 451 Winter 2013

Section 10

Anton Osobov
aosobov@cs

Material adapted from previous offerings of CSE 451
Specifically from slides by Gary Kimura, Ed Lazowska, and Tom Anderson

Reminders

- No Quiz Tomorrow (3/15)
- Final
  - Wednesday, 3/20, 2:30 - 4:20
  - Closed book, closed note

Topics for Today

- Final Review
  - Will go over some key concepts you should understand
  - These slides may not cover all topics that will be on the final

Processes

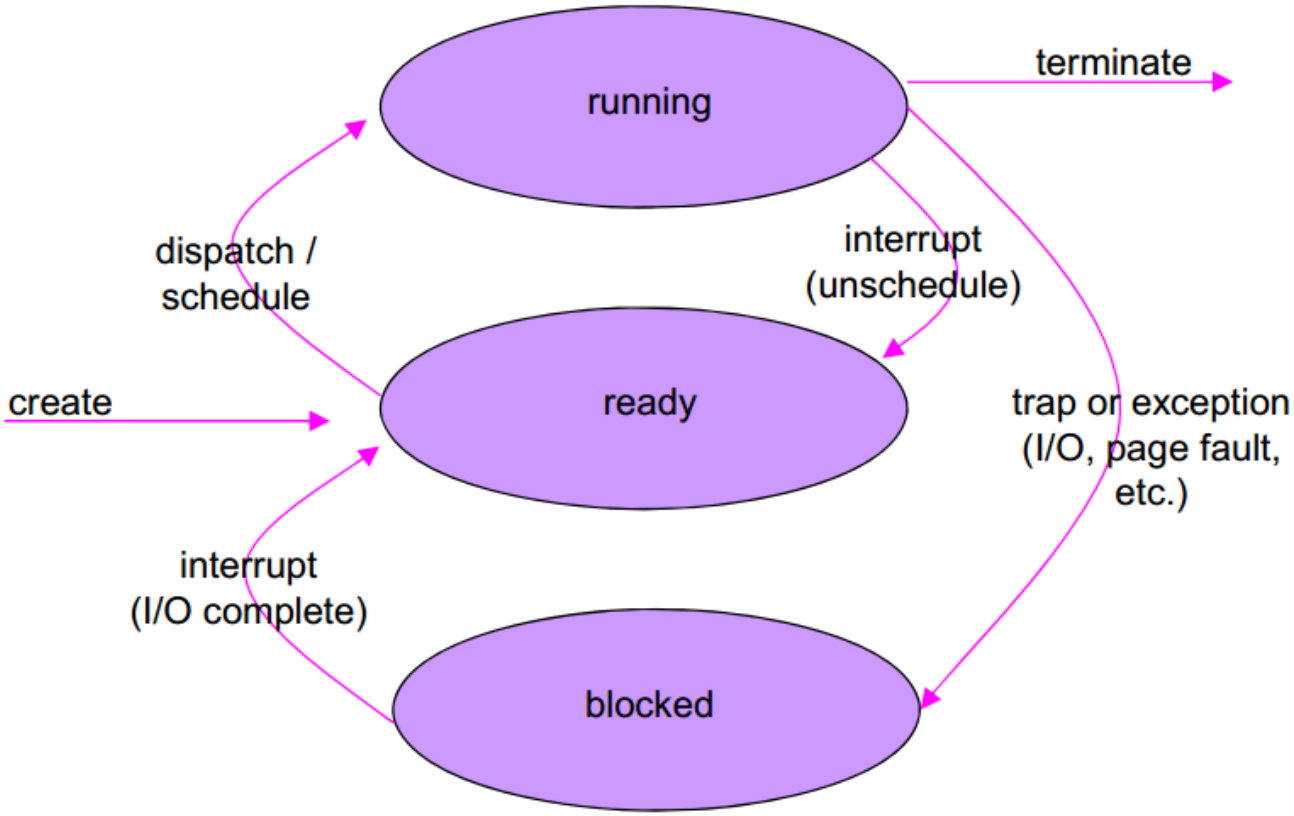
- Process = fundamental abstraction for program execution
- Process made up of:
  - an address space which contains:
    - the code for the running program
    - the data for the running program
  - at least one thread with state
    - Registers, IP
    - Stack and stack pointer
  - a set of OS resources
    - open files, network connections, sound channels, etc

Processes

- State queues
  - Which states, what transitions are possible?
  - When do transitions happen?

Processes

- State queues
  - Which states, what transitions are possible?
  - When do transitions happen?



Processes

- Process manipulation
  - What does fork() do?
  - What about exec()?

Threads

- What is a thread?
  - Why are they useful?
- How does thread scheduling differ from process scheduling?

Threads v Processes

Overview

- Process
  - Isolated with its own virtual address space
  - Contains process data like file handles
  - Lots of overhead
  - Every process has at least one kernel thread
- Kernel Threads
  - Shared virtual address space
  - Contains running state data
  - Less overhead
  - From the OS's point of view, this is what is scheduled to run on a CPU
- User Threads
  - Shared virtual address space, contains running state data
  - Kernel unaware
  - Even less overhead

Threads v Processes

Trade-offs

- Process
  - Secure and isolated
  - Kernel aware
  - Creating a new process brings lots of overhead (address space)
- Kernel Threads
  - No need to create a new address space
  - No need to change address space in context switch
  - Kernel aware
  - Still need to enter kernel to context switch
- User Threads
  - No new address space, no need to change address space
  - No need to enter kernel to switch
  - Kernel is unaware. No multiprocessing. Synch I/O block all user threads

Threads v Processes

- When would using separate processes be advantageous over using separate threads?

Threads v Processes

- When would using separate processes be advantageous over using separate threads?
  - Separate processes ideal for large tasks that share little or no data
  - Ideal if each processes is “heavyweight”
  - Example: Chrome uses separate processes for tabs to get sandboxing

Scheduling

- When does scheduling happen?
  - Job changes state, interrupts, exceptions, job creation
- Scheduling goals?
  - Maximize CPU utilization
  - Maximize job throughput
  - Minimize {turnaround time | waiting time | response time}
  - Batch vs interactive: what are their goals?
    - Throughput/utilization vs response time
- What is starvation? What causes it?
- Know the differences between scheduling algorithms:
  - FCFS/FIFO, SPT, RR, priority, MLFQ

Synchronization

- Why do we need it?
  - Data coordination? Execution coordination?
  - What are race conditions? When do they occur?
  - When are resources shared? (variables, heap objects, ...)
- What is mutual exclusion?
  - What is a critical section?
  - What are the requirements of critical sections?
    - Mutual exclusion, progress, bounded waiting, performance
  - What are the mechanisms for programming critical sections?
    - Locks, semaphores, monitors, condition variables

Locks

- What does it mean for acquire/release to be atomic?

Monitors

- When would it make sense to use a Mesa monitor over a Hoare monitor, and vice versa?

Monitors

- When would it make sense to use a Mesa monitor over a Hoare monitor, and vice versa?
  - A Mesa monitor is better used for situations where overall speed is more important, because a context switch isn't required.
  - A Hoare monitor is better for situations where a thread absolutely needs to execute immediately after it finishes waiting, e.g. if the thread is running a time-critical task.

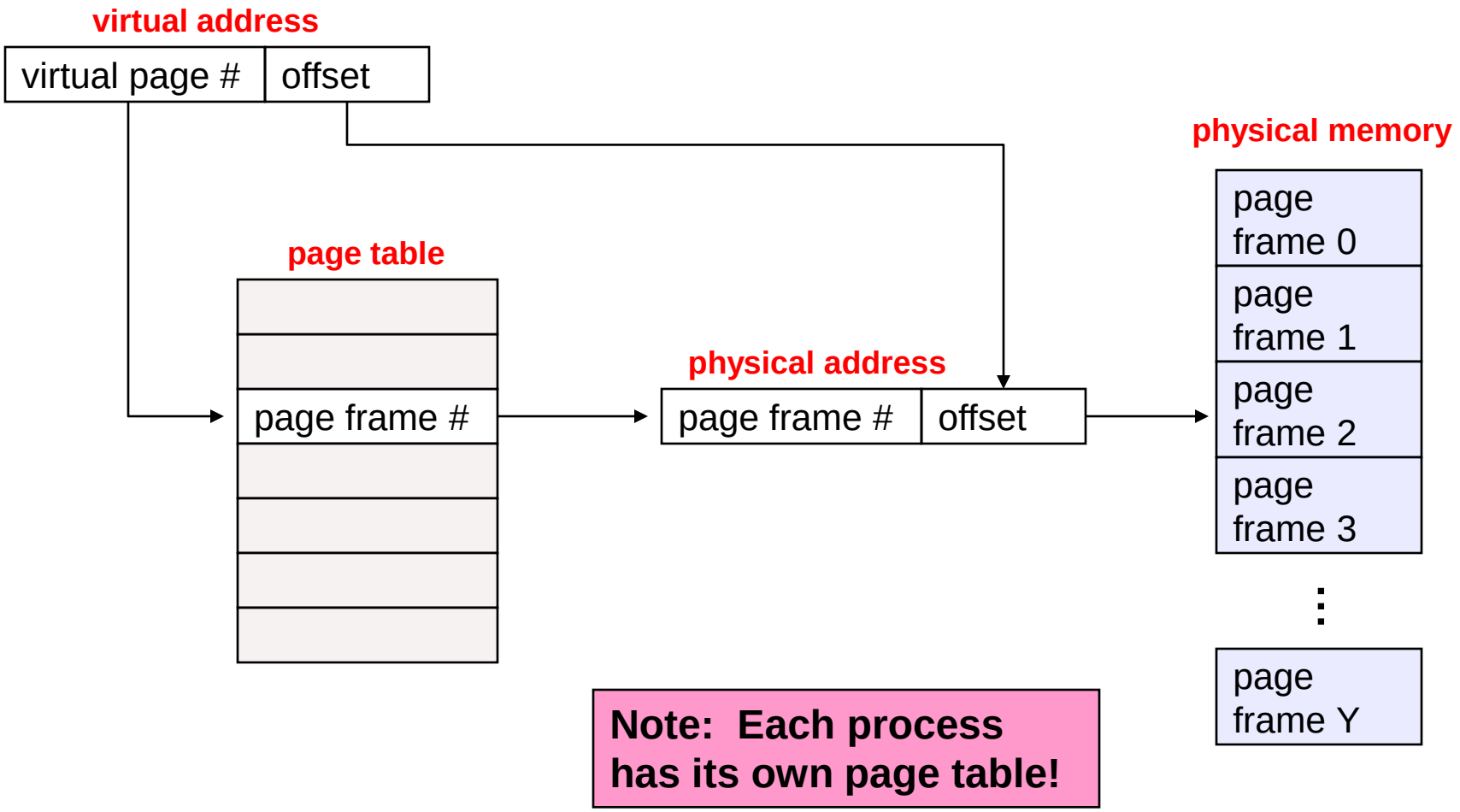
Virtual Memory

- What happens on a virtual memory access?

Virtual Memory

- What happens on a virtual memory access?
  - Address translation
    - Page table lookup
    - TLB
  - Page fault?
    - Page replacement
    - Process/queue management
- How does all of the overhead pay off?
  - Locality!
    - Temporal and spatial

Virtual Memory



Page Replacement

- Algorithms:
  - Belady's, FIFO, LRU, LRU Clock, Working Set, PFF
  - Local vs global
- How/why are any of these better or worse than the others?
- What happens when paging goes wrong?
  - Thrashing!

More Virtual Memory

- What problem does the TLB address?

More Virtual Memory

- What problem does the TLB address?
  - Increases speed of virtual address translation

More Virtual Memory

- What problem does the TLB address?
  - Increases speed of virtual address translation

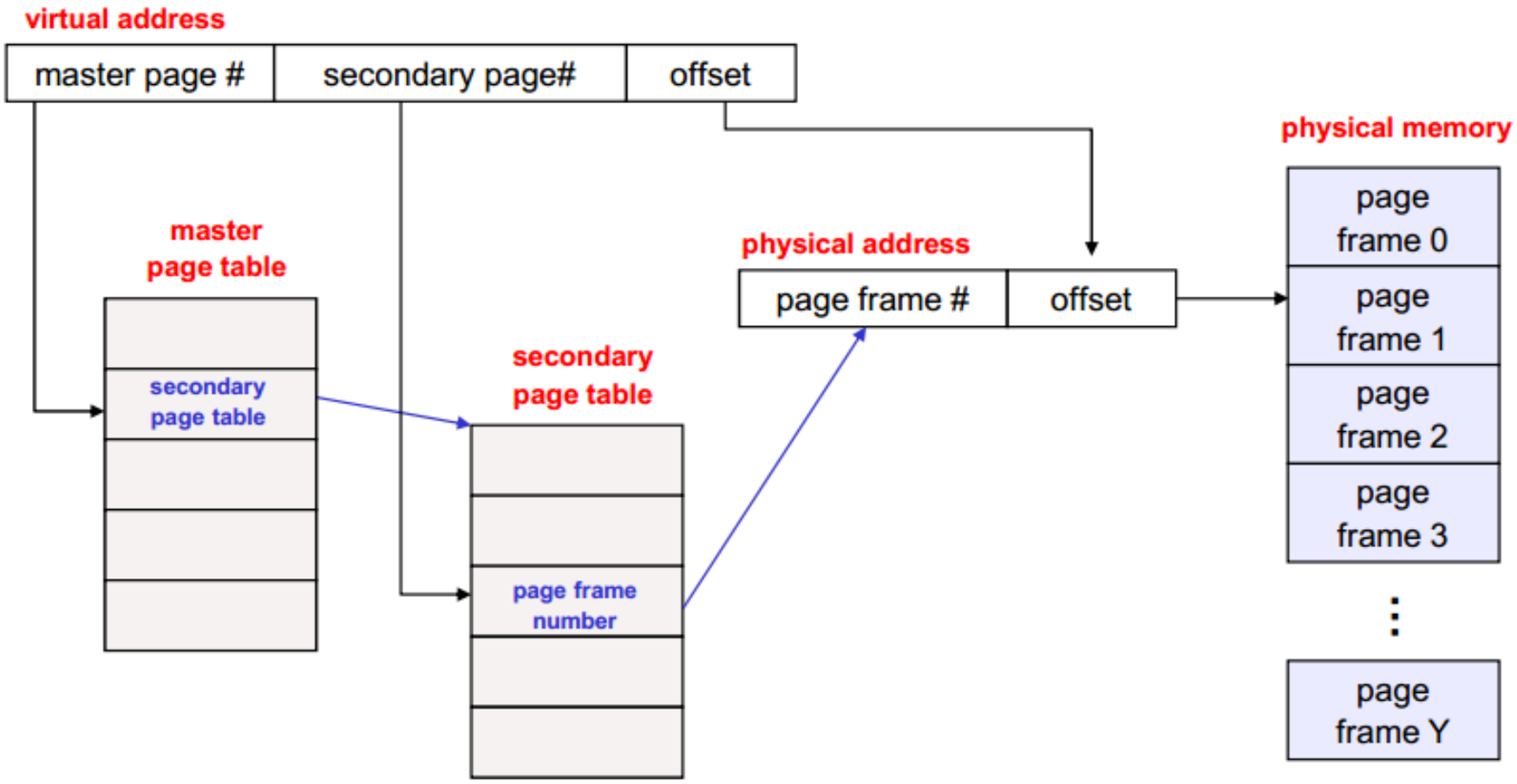
- What problem do two-level page tables address?
  - What's the key concept?

More Virtual Memory

- What problem does the TLB address?
  - Increases speed of virtual address translation

- What problem do two-level page tables address?
  - Huge physical memory requirements of page tables
  - What's the key concept?
    - Indirection

Two-Level Page Tables



Course Evaluations!