

# CS 162 Midterm Review Spring 2007

Thomas Kho <[tkho@eecs](mailto:tkho@eecs)>  
February 26, 2007

*Note: the illustrations in this presentation are by Silberschatz, Galvin and Gagne ©2002  
Based on the Spring 2005 review by Karl Chen and Adrian Mettler*

# Available Study Materials

- Professor's Lecture Notes
- Student Lecture Notes
- Discussion Notes
- Sample Midterm in Reader
- Reader, Textbook

# What is an operating system?

- Batch systems
- Uniprogramming
- Multiprogramming

# Function of an operating system

- Coordinator
  - Resource allocation and management
  - Sharing and protection
  - Concurrency, I/O, memory, files, network
- Extended Machine
  - Present nice interface to messy hardware
  - Standard library & interface (FS, VM, etc)
- Illusion that shared resources are unshared

# Processes

- What are they?
- How are they made?
- How are they represented in an OS?
  - E.g. what state do they have?
- How many processes can run on a CPU at once?
- What are process states?

# Processes

- *An execution stream in the context of a particular process state.*
- Running code + local state
- process : program :: play : script

# Process Control Block

- Process state
  - PC, Processor Status Word
  - General purpose & FP registers
  - Process ID
  - Memory state
  - Scheduling info
  - Accounting, etc
  - Pointers to other info (open files, etc)
- Process table: collection of all PCBs

# Threads

- What are they?
- How are they made?
  - How are they joined?
- What state do they maintain?
- How many can run on a CPU at once?
- How do they differ from processes?
- Why separate threads from processes?

# Threads

- Each has a PC, registers, stack
- Shares code, data, OS resources with other processes
- Task – the set of threads sharing code, data ,etc
- Low context-switch overhead
- Why use threads?
  - Better programming model, parallelism on MP, low overhead

# Threads, Cont. . .

- What is different between a thread and a process context switch?
- How does a thread go from inactive to running?
  - What needs to happen?
- Can we use both threads and processes?
  - Why would we do this?

# Cooperating threads

- What is the difference between multiprocessing and multiprogramming?
- What is the definition of ‘run concurrently’?
- What are independent threads?
- What are cooperating threads?

# Independent and Cooperating Processes

- Independent processes – no shared state, deterministic execution
- Cooperating processes: shared state, reproducible results
- We want processes to cooperate for:
  - Shared resources
  - Speed (overlapped operation)
  - Modularity (toolbox philosophy)

# Exceptions

- What is an interrupt?
  - What happens when one occurs?
- What is a trap/exception?
  - What happens when one occurs?

# Atomicity

- What is an atomic operation?
- Why are they important?
- How can you ensure an instruction is atomic?

# Synchronization

- What is synchronization?
- What is a critical section?
- What is mutual exclusion?
- What is busy-waiting? (or 'spinning')
- What is a lock?
  - What are its properties?

# Too-Much-Milk Problem

- Synchronization – using atomic ops to ensure correct operation of cooperating processes
- Mutual exclusion – mechanism to ensure only one person is doing something at a time
- Critical section – code in which one process may be executing at any time
- Review the Milk examples on your own

# Synchronization, cont. . . .

- What is a semaphore?
  - What are its properties?
- What is a condition variable?
  - What are its properties?
  - How are these different from semaphores?
- What is a monitor?
  - What is a Mesa-style monitor?
  - What is a Hoare-style monitor?

# Synchronization

- Mutual exclusion requirements
  - Mutual exclusion and progress
  - Also desirable – bounded waiting, efficient, simple
- Semaphore – synchronization variable of non-negative integer values. Two **atomic** ops:
  - P() if ( $\text{sem} == 0$ ) *wait* until ( $\text{sem} > 0$ ) then  $\text{sem}--$
  - V()  $\text{sem}++$ , wake a blocked process
- Semaphores allow mutual exclusion and/or synchronization

# Producer/Consumers Example

*Initialize empties = N, fulls = 0, mutex = 1;*

Producer process:

P(empties);

P(mutex);

get empty buffer from pool of empties;

V(mutex);

produce data in buffer;

P(mutex);

add full buffer to pool of fulls;

V(mutex);

V(fulls);

Consumer process:

P(fulls);

P(mutex);

get full buffer from pool of fulls;

V(mutex);

consume data in buffer;

P(mutex);

add empty buffer to pool of empties;

V(mutex);

V(empties);

# Readers/Writers Example

- Multiple concurrent readers, one writer at a time
- AR, WR, AW, WW
- Scheduling: writers get preference
- Initialization:  $OKToRead = 0$ ;  $OKToWrite = 0$ ,  $Mutex = 1$ ,  $AR = WR = AW = WW = 0$ ;

# Readers/Writers Example

Reader Process:

```
P(Mutex);
if ((AW+WW) == 0)
{
    V(OKToRead);
    AR = AR+1;
}
else WR = WR+1;
V(Mutex);
P(OKToRead);
-- read the necessary data;
P(Mutex);
AR = AR-1;
if (AR==0 && WW>0)
{
    V(OKToWrite);
    AW = AW+1;
    WW = WW-1;
}
V(Mutex);
```

Writer Process:

```
P(Mutex);
if ((AW+AR+WW) == 0) { (do we
    need WW?)
    V(OKToWrite);
    AW = AW+1;
}
else WW = WW+1;
V(Mutex);
P(OKToWrite);
-- write the necessary data;
P(Mutex);
AW = AW-1;
if (WW>0) {
    V(OKToWrite);
    AW = AW+1;
    WW = WW-1;
} else while (WR>0) {
    V(OKToRead);
    AR = AR+1;
    WR = WR-1;
}
V(Mutex);
```

CS162 Spring 2007 Midterm 1 Review

# Synchronization

- Look at Dining Philosophers example on your own

# Synchronization, cont. . . .

- What is a mutex?
- Can you build one kind of synchronization primitive out of another?
- What is important about test-and-set and swap?

# Monitors

- Hoare *Monitors* paper in the reader
- Monitor: condition variables + associated lock
- Shared data, operations on data, sync. & sched.
- `condvar.wait()` - sleeps until woken by `signal()`
  - release monitor lock, put process to sleep
  - reacquire lock immediately when woken
- `condvar.signal()` - wake up **at most** one waiter
  - do nothing if no waiters
- Explicit separation of mutex and scheduling

# Deadlock

- What is deadlock?
  - What are the four conditions of deadlock?
- What are examples of preemptible and non-preemptible resources?
- What are the two approaches to the deadlock problem?
- What are ways to prevent deadlock?
  - What is a ‘safe state’?

# Deadlock, cont. . . .

- Bankers Algorithm
- Graph Algorithm
  - What is a resource allocation graph?

# Deadlock, cont. . . .

- Sample problem:
  - Either show a complete safe sequence or show that there isn't one.

proc	hasX	hasY	maxX	maxY
A	10	20	65	65
B	0	70	70	90
C	30	10	60	40
D	50	80	100	200

a. available: X: 40 Y: 40

b. available: X: 30 Y: 30

# Scheduler

- What is a dispatcher/scheduler?
  - What are its responsibilities?
- When does it run?
- How does it maintain control of the CPU?
  - How does it put a process on the CPU?

# Dispatcher/Scheduler

- Scheduler decides who runs next, dispatcher runs it
- OS regains control via `yield()` or interrupts
- EIT: Exceptions = Interrupts + Traps
  - Interrupts (async) vs traps (sync)
- `fork()`
- Process states: running, ready, blocked

# Scheduling

- What are scheduling goals?
  - What is flow time?
  - What is throughput?
  - What is response time?
  - What is overhead?

# Scheduling

- Goals
  - High utilization
  - Minimize overhead
  - Minimize response time
  - Maximize throughput
  - Guarantee service to some
  - Minimize number of interactive users
  - Shorts jobs quick
  - Avoid starvation
  - Minimize response time variance
  - Satisfy external constraints
  - Meet deadlines
  - Graceful degradation

# Scheduling

- Real goal
  - minimize flow time
  - minimize variance of flow / service time
- External constraints
  - Fixed priorities, fixed fraction of machine, realtime

# Scheduling, cont. . . .

- How are processes scheduled on the CPU?
  - FIFO (First in first out)
  - RR (Round robin)
  - SJF (Shortest job first)
  - SRTF (Shortest remaining time first)
  - SET (Shortest elapsed time)
  - EQ, or MLFQ (Multi-level feedback queues)
  - Lottery/stride scheduling

# Scheduling, cont. . .

- How do job characteristics influence scheduling choice?
- What is the difference between a preemptive and non-preemptive scheduling policy?
- What is an adaptive scheduling policy?
- Jobs vs. processes
- Open vs closed systems

# Scheduling

- Adaptive – based on runtime behavior
  - SET
  - Foreground/background, MLFB
  - Exponential queue (multi-level feedback queue)
- Fair-share – priority based on recent CPU usage

# Queuing

- Open vs. closed system
- Throughput – jobs completed per second
  - varies with scheduler in closed system
- Job characteristics – runtime highly skewed
- Minimize flow time => minimize avg. users in sys
- $N = \lambda * F$ 
  - avg users in sys = arrival rate \* mean flow time

# Linkers & Loaders

- Why divide a program into segments?
- Which segments?
- What problem does the linker solve?
- How does it solve it?
- What is the symbol table?
- What is the relocation table?

# Memory allocation

- Why do we need dynamic memory allocation?
  - Recursive procedures, complex data structures, ...
- Stack allocation
- Heap allocation
- Reference counts
- Garbage collection

# Protection

- Multiprogramming without protection
  - How do linkers and loaders work?
- What are hardware protection modes?
  - Why do we need these?
- How can a user program access OS services?
  - How does the protection mode switch?

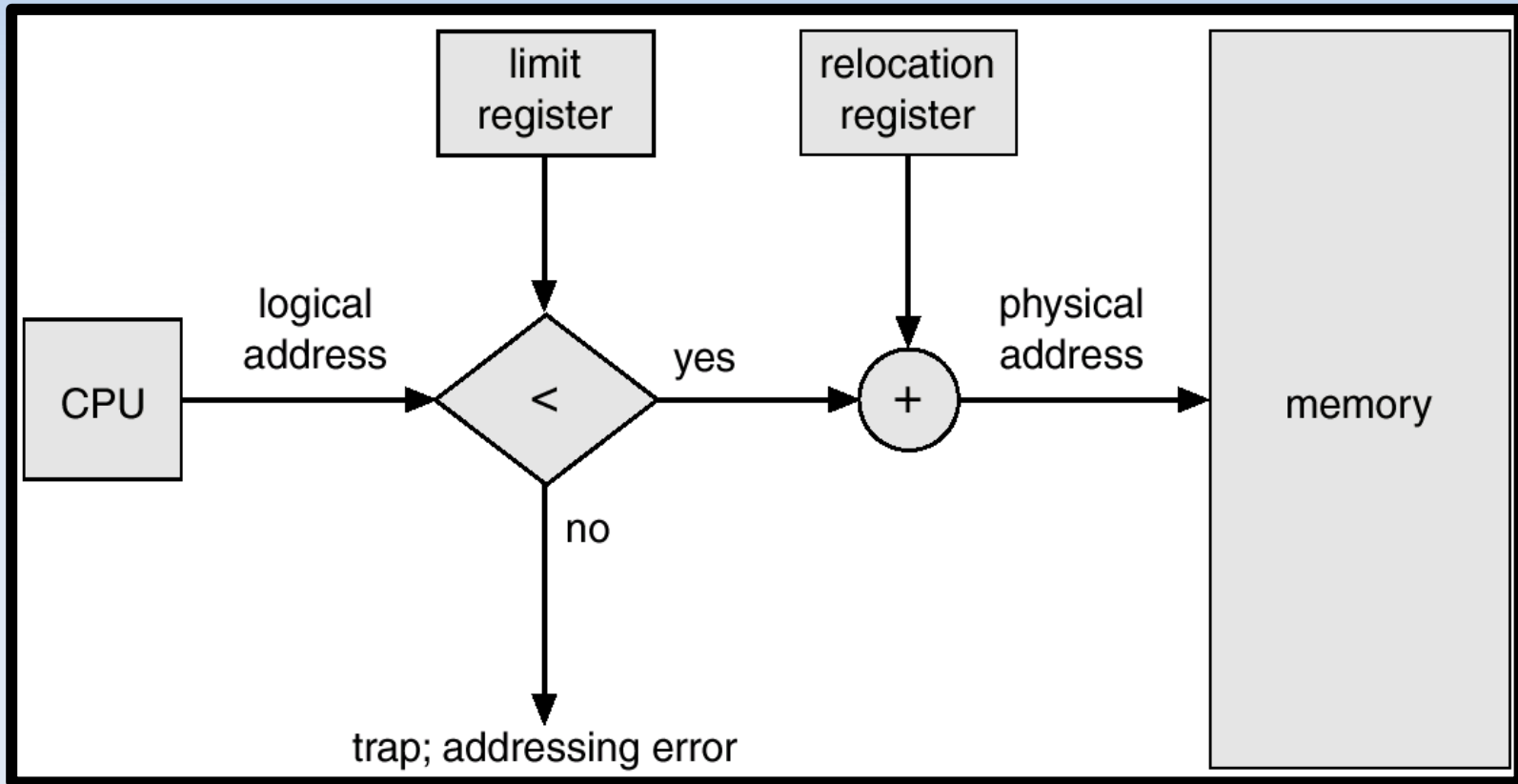
# Address Translation

- Why do we need this?
- What is a Virtual Address? What is a Physical Address?
- What is internal fragmentation? What is external fragmentation?

# Address Translation, cont. . . .

- What is base and bound relocation?
  - What are the advantages
  - What are the disadvantages?
  - What does base-and-bounds relocation add to context switches?
  - What does it add per memory access?

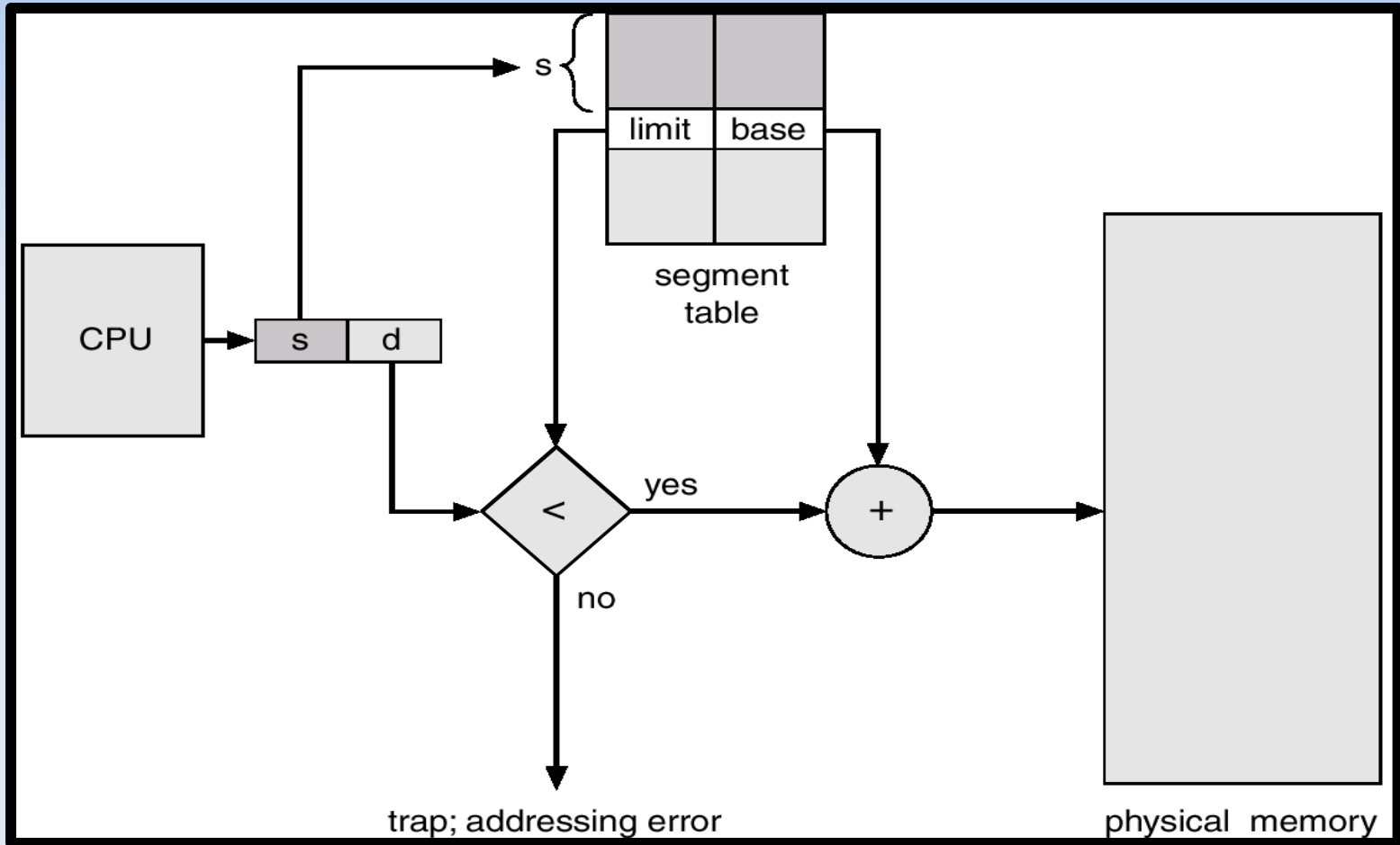
# Address Translation, cont. .



# Address Translation, cont. . . .

- Segmentation
  - What is a segment table?
  - What does segmenting fix that is broken with base and bounds?
  - What is a segment table entry? What does it look like?
    - What is each field for?
  - What is a segment fault?
  - What exactly happens when a program tries to read from memory?
  - What does segmentation add to context switches?

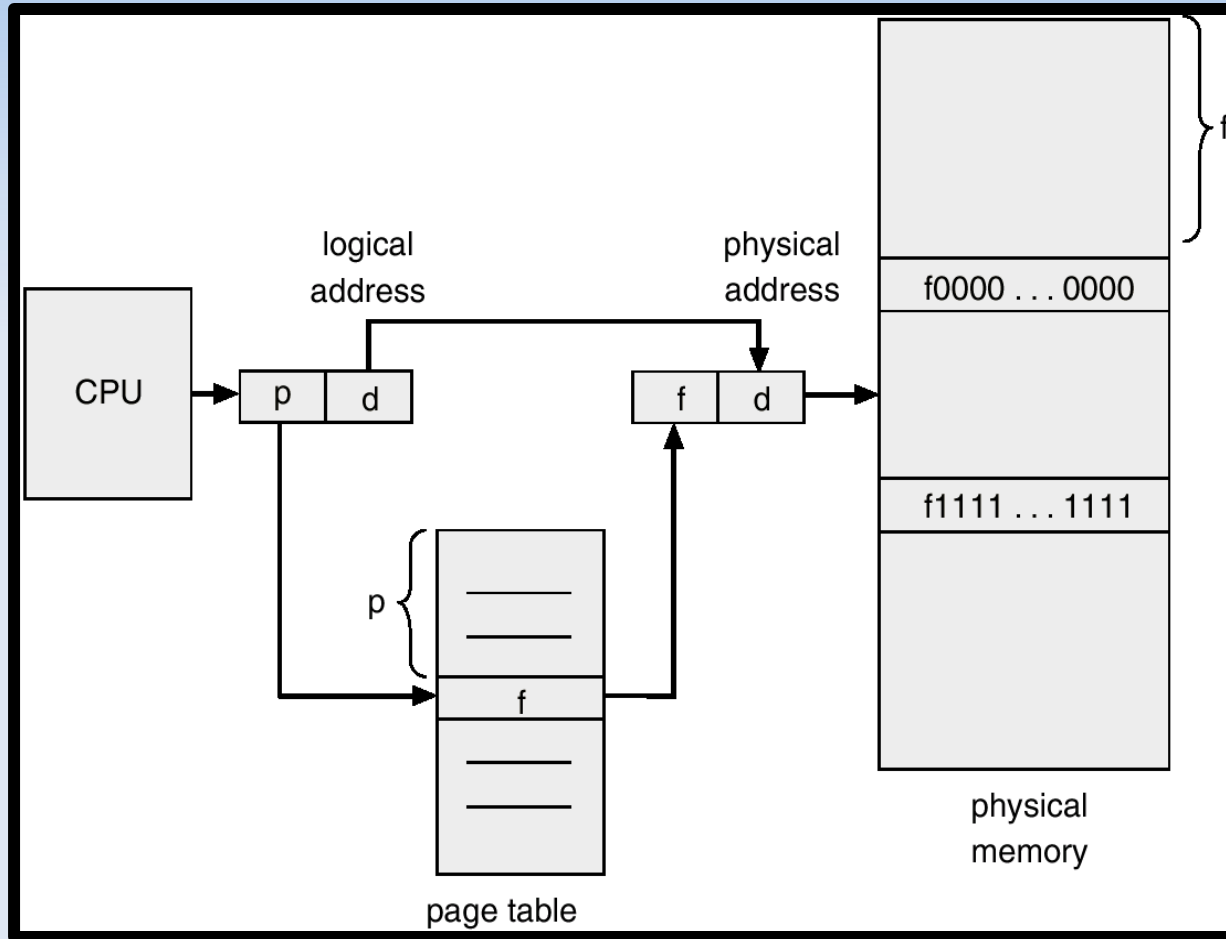
# Address Translation, cont. .



# Address Translation, cont. . . .

- Paging
  - What is a page table?
  - What does paging fix that is broken in segmenting?
  - What does a page table entry look like?
    - What is each field for?
  - What is a page fault?
  - What exactly happens when a program tries to read from memory?

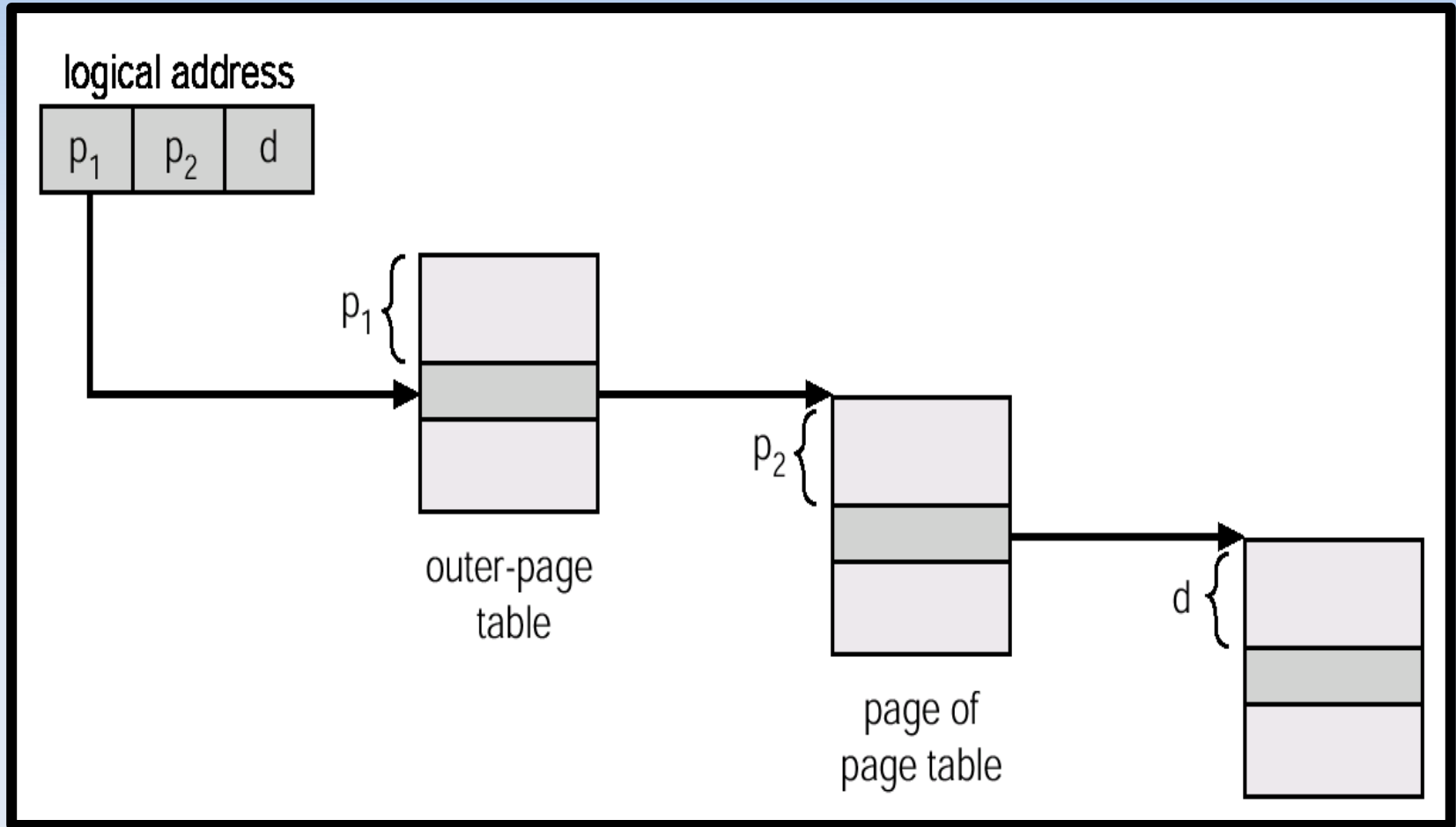
# Address Translation, cont. .



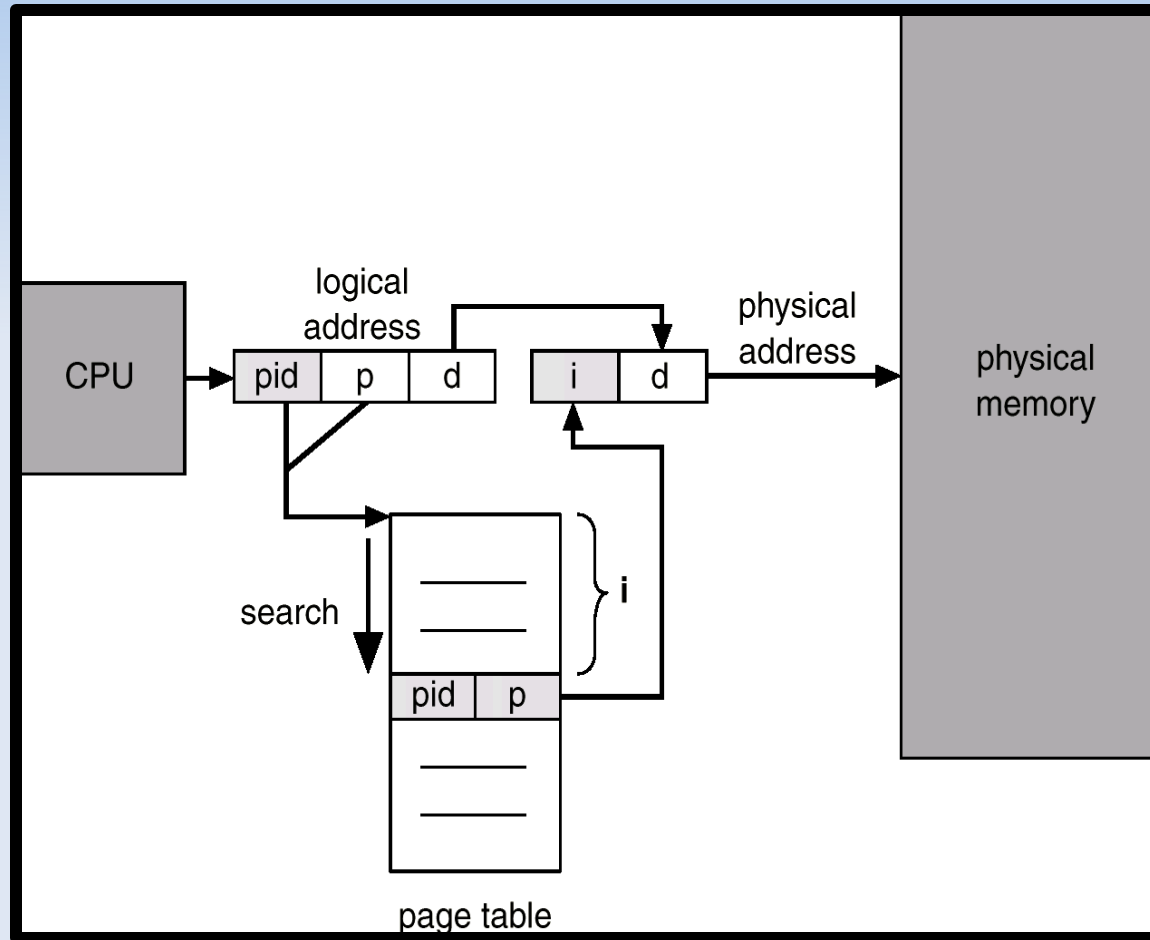
# Address Translation, cont. . . .

- What does paging add to context switches?
- Can you page out anything in memory?
- How big is a page table?
- What is 2-level paging?
  - How does this work?
- What is an inverted page table?
  - How does this work?
  - Why do you need one?

# Address Translation, cont. .



# Address Translation, cont. .



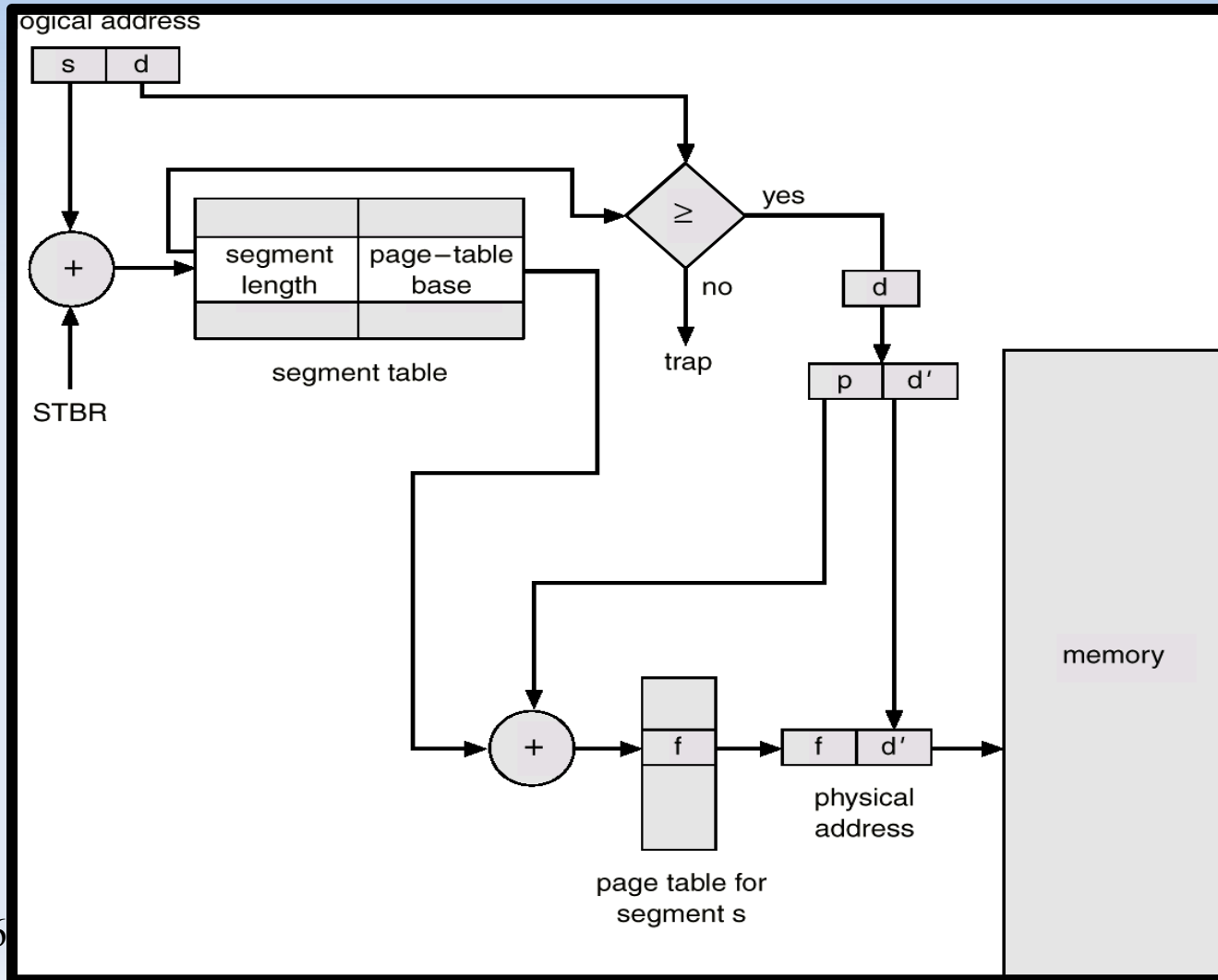
Note: The book calls this “inverted page table”

What Prof. Smith calls “inverted page table” is what the book calls “hashed inverted page table.”

# Address Translation, cont. . . .

- How do these methods provide protection among processes?
- How would you combine segmentation with paging?
  - Why would you want to?

# Address Translation, cont. .



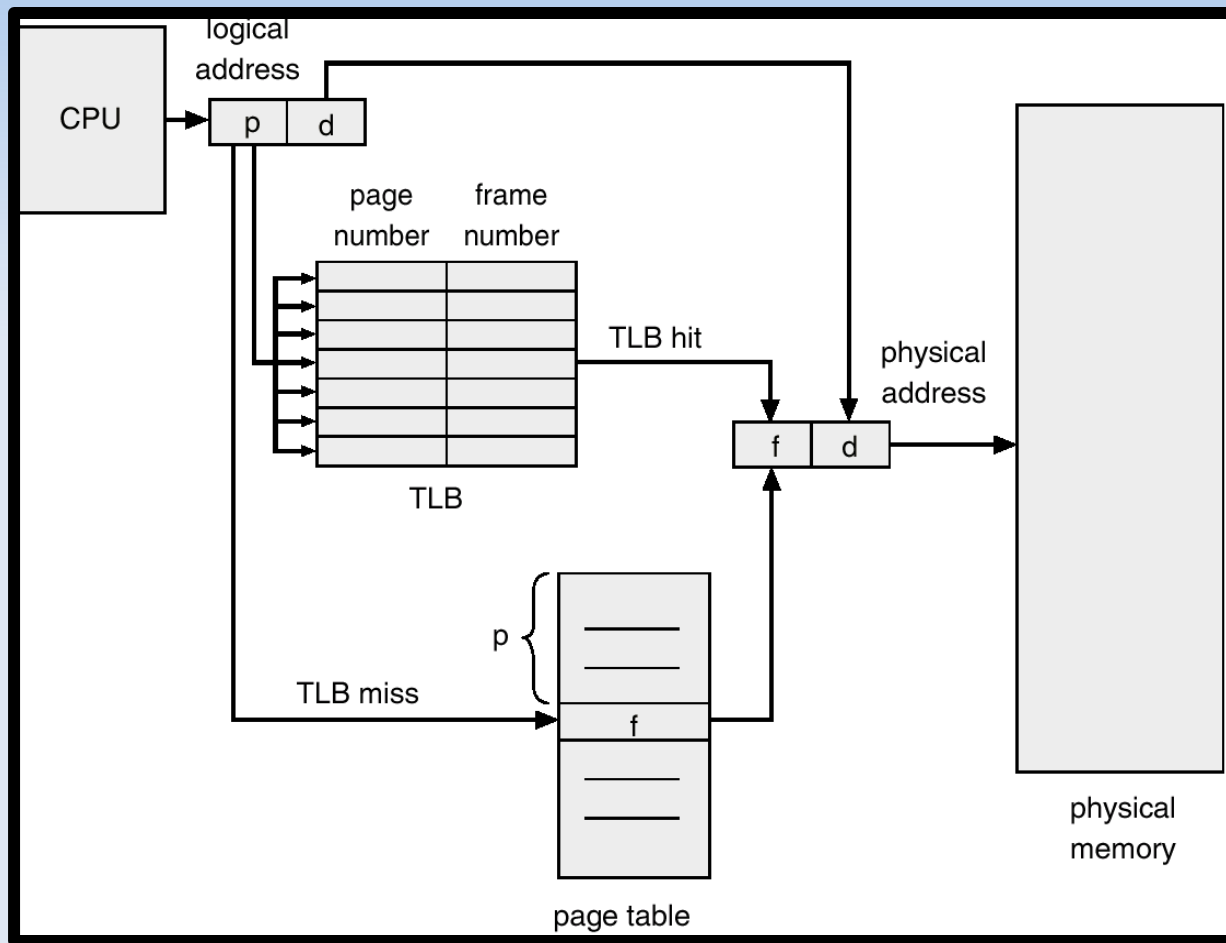
# Caches and TLBs

- What is the primary motivation behind using a cache?
- What are the different methods for searching a cache?
  - What is direct mapped?
  - What is set associative?
  - What is fully associative?
  - What are the advantages and disadvantages of each?

# Caches and TLBs, cont. . . .

- What is a TLB?
  - Why do we use them?
- What happens to the TLB on a context switch?
- What does using a TLB add to handling page faults?
- What is the effective access time of a TLB?
  - E.g. how is it calculated?

# Caches and TLBs, cont. . .



# Caches, cont.

- What happens when you write to a memory location that's swapped out?
  - TLB lookup
  - TLB fault
  - Page table lookup
  - Page fault
  - Context switch