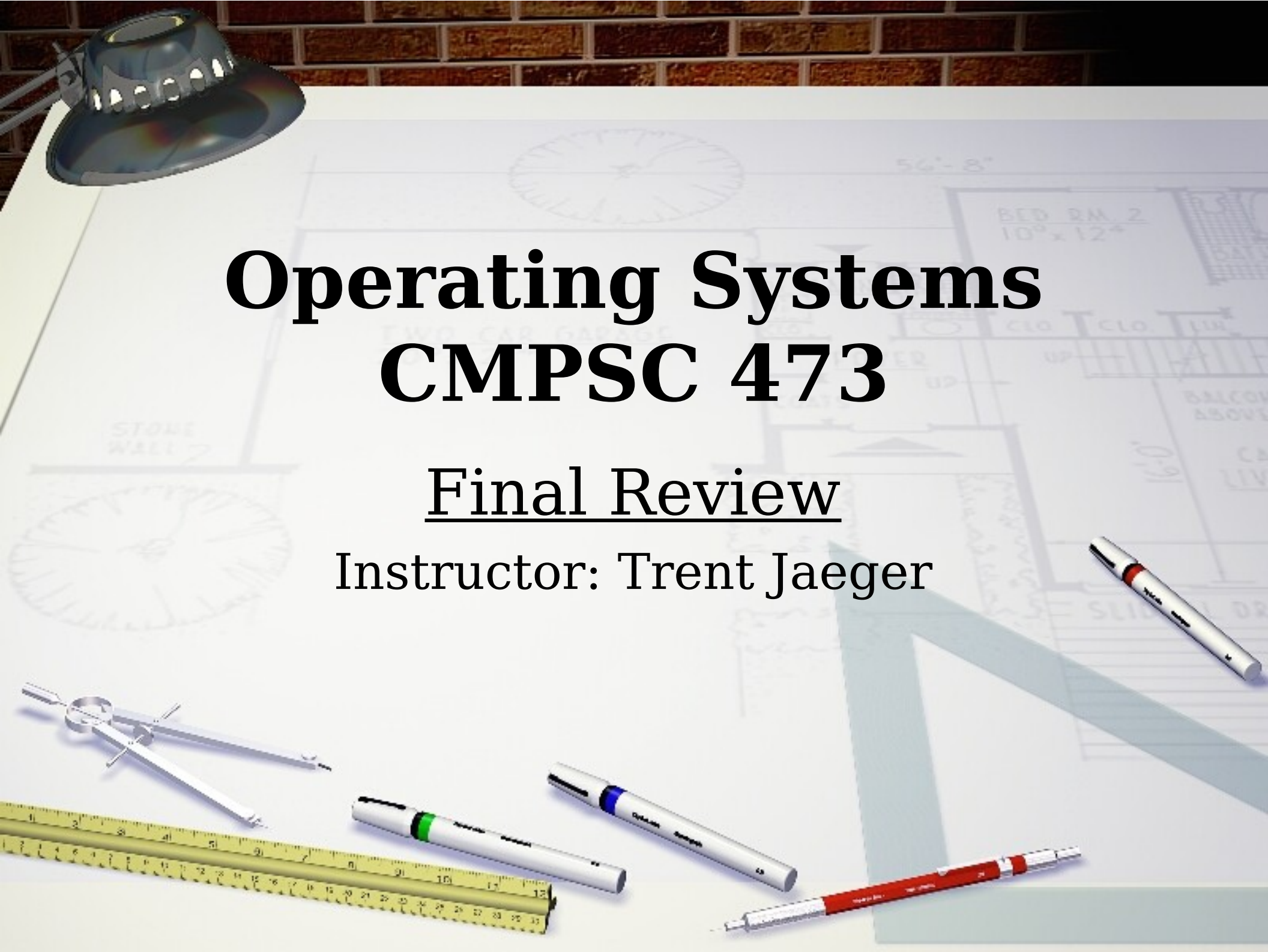


Operating Systems CMPSC 473

Final Review

Instructor: Trent Jaeger

Exam

- Four parts
  - (1) True/False – 10 questions (20 pts)
  - (2) Fill-in – 5 questions (19 pts)
  - (3) Short answer – 5 questions (20 pts)
  - (4) Problem solving – 5 multipart questions (50 pts)
- Broad, but not open-ended

True/False

- The *process control block* of a process may be used by the operating system after the process exits.

TRUE **FALSE**

Fill-in

- Privileged instructions can only be executed in

  _____ mode.

Short Answer

- What is the maximum practical limit for the size of virtual memory?

Problem Solving

- Multipart questions
- E.g., #6 and #7 of HW3 and #8 of HW4

HW3 #1

1. (*10pts*) Show how the *readers-writers* mutual exclusion algorithm below either satisfies or fails to satisfy mutex, progress, and bounded waiting.

Readers Writers Algorithm

```
        int nreaders = 0;
        sem m, wrt; /* initialized to 1 */

Reader() {
        P(m);
        nreaders++;
        if ( nreaders == 1 ) P(wrt);
        V(m);
        /*** access shared data ***/
        P(m);
        nreaders–;
        if ( nreaders == 0 ) V(wrt);
        V(m);
}

Writer() {
        P(wrt);
        /*** access shared data ***/
        V(wrt);
}
```

Mutex Liveness Conditions

- **1. no deadlock:** if a process is in its entry section at some time, then later some process is in its critical section
- **2. progress/no lockout/no starvation:** if a process is in its entry section at some time, then later the *same* process is in its critical section
- **3. bounded waiting:** no lockout + while a process is in its entry section, other processes enter the critical section no more than a certain number of times.
- These conditions are increasingly strong (each implies the previous ones)
  - Exercise: Prove this

HW3 #1

1. (*10pts*) Show how the *readers-writers* mutual exclusion algorithm below either satisfies or fails to satisfy mutex, progress, and bounded waiting.

- **Bounded waiting is not satisfied** as there is no bound on the number of readers that may access the shared data before a writer may be allowed to proceed. Writer may be starved failing **progress**.

HW3 #2

2. (*10pts*) William Faulkner (the author) states that a writer needs three things: experience, observation, and imagination. Suppose that three threads implement automated authoring, but each has a knowledge base of only one of the three elements. Another three threads produce knowledge on two of the elements at a time. Given the semaphore state defined below and the individual thread implementations, identify why this solution fails and suggest a solution. **Hint: Look at "Classical Synchronization Problems" in the Little Book of Semaphores.**

Setup Defs

```
inputSem = Semaphore(1)
experience = Semaphore(0)
observation = Semaphore(0)
imagination = Semaphore(0)
```

Input Thread 1

```
inputSem.wait()
experience.signal()
observation.signal()
```

Author Thread 1

```
experience.wait()
observation.wait()
inputSem.signal()
```

HW3 #2

4.5 Cigarette smokers problem

The cigarette smokers problem problem was originally presented by Suhas Patil [8], who claimed that it cannot be solved with semaphores. That claim comes with some qualifications, but in any case the problem is interesting and challenging.

Four threads are involved: an agent and three smokers. The smokers loop forever, first waiting for ingredients, then making and smoking cigarettes. The ingredients are tobacco, paper, and matches.

We assume that the agent has an infinite supply of all three ingredients, and each smoker has an infinite supply of one of the ingredients; that is, one smoker has matches, another has paper, and the third has tobacco.

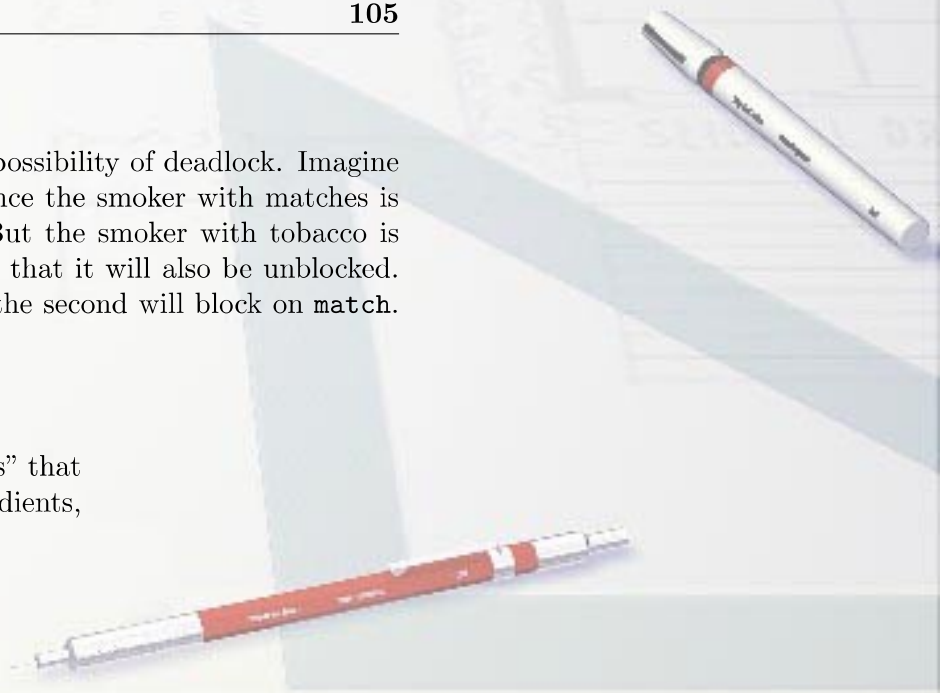
The agent repeatedly chooses two different ingredients at random and makes them available to the smokers. Depending on which ingredients are chosen, the smoker with the complementary ingredient should pick up both resources and proceed.

4.5.1 Deadlock #6

The problem with the previous solution is the possibility of deadlock. Imagine that the agent puts out tobacco and paper. Since the smoker with matches is waiting on `tobacco`, it might be unblocked. But the smoker with tobacco is waiting on `paper`, so it is possible (even likely) that it will also be unblocked. Then the first thread will block on `paper` and the second will block on `match`. Deadlock!

4.5.2 Smokers problem hint

The solution proposed by Parnas uses three helper threads called “pushers” that respond to the signals from the agent, keep track of the available ingredients, and signal the appropriate smoker.



HW3 #3

3. (*8pts*) Suppose you have two threads X and Y, consisting of two statements each, $x1$ followed by $x2$ and $y7$ followed by $y8$, respectively. Using one or more semaphores ensure that $x1$ is run before $y8$ and $y7$ is run before $x2$. Note that your solution should not restrict the order of execution between $x1$ and $y7$.

- Need both X and Y to wait for each to fulfill a requirement.

HW3 #3

3. (*8pts*) Suppose you have two threads X and Y, consisting of two statements each, $x1$ followed by $x2$ and $y7$ followed by $y8$, respectively. Using one or more semaphores ensure that $x1$ is run before $y8$ and $y7$ is run before $x2$. Note that your solution should not restrict the order of execution between $x1$ and $y7$.

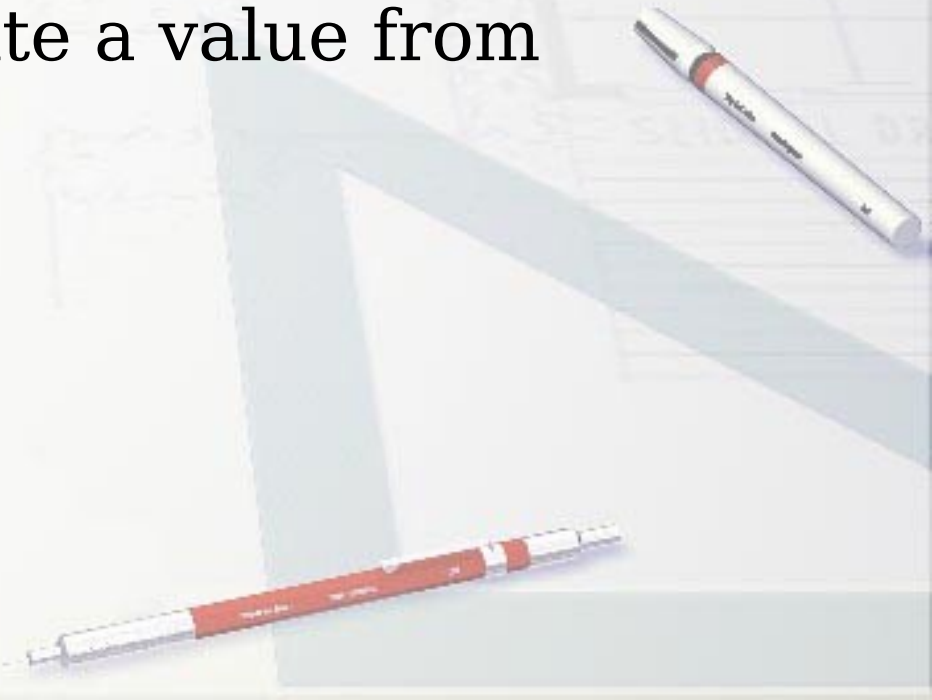
- Need both X and Y to wait for each to fulfill a requirement.

answer: For thread X: $x1$, $signal(xArrived)$, $wait(yArrived)$, $x2$ and mirror image for for thread Y
-2pts for each missing signal and wait.

HW3 #4

4. (*10pts*) In questions 1-3 at the end of Chapter 26 of OSTEP, explain whether or not there is a race condition, explaining the reasoning behind your answer. Answer Question 6 at the end of Chapter 26 of OSTEP.

- 1-3: Processes read a value from memory and decrement in registers
- 6: Processes read and write a value from memory



HW3 #4

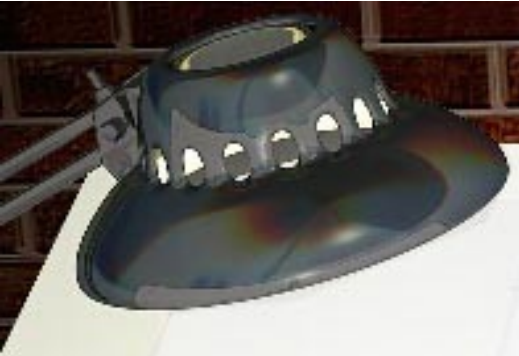
4. (*10pts*) In questions 1-3 at the end of Chapter 26 of OSTEP, explain whether or not there is a race condition, explaining the reasoning behind your answer. Answer Question 6 at the end of Chapter 26 of OSTEP.

- 1-3: No race – must be a write operation to memory
  - Registers are solely assigned to one thread
  - Saved on context switch
- 6: Race – on reading and writing from memory

HW3 #5

5. (*10pts*) Write monitor solutions for the readers-writers problem and the dining philosophers problem.

- What's a monitor?



Monitors

- An abstract data type consisting of
  - Shared data
  - Operations/procedures on this shared data
- External world only sees these operations (not the shared data or how the operations and sync. are implemented).
- Only 1 process can be “active” within the monitor at any time

  i.e., of all the processes that are executing monitor code, there can be at most 1 process in ready queue (rest are either blocked or not in monitor!)

HW3 #5

5. (*10pts*) Write monitor solutions for the readers-writers problem and the dining philosophers problem.

- Examples – No locks as monitor data structure provides mutual exclusion
- Need condition variables or semaphores to block/awake threads under conditions

http://courses.cs.vt.edu/cs5204/fall00/Supplemental/ReadersWriters.html is a good one for reader-writers

http://www.csee.wvu.edu/ jdm/classes/cs550/notes/tech/mutex/dp-mon.html appears to work for dining philosophers.

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(a) (2pts) Draw the *Needs* table for the system.

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(a) (2pts) Draw the *Needs* table for the system.

answer: (a)

Needs (A/B/C)
3/2/4
5/0/2
1/2/3
1/3/0

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(b) (3pts) Suppose the number of free resources are: $A = 1$, $B = 3$, $C = 1$. Find a process sequence that will prove that this system is in a *safe state* or identify that it is not in a *safe state*.

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(b) (3pts) Suppose the number of free resources are: $A = 1$, $B = 3$, $C = 1$. Find a process sequence that will prove that this system is in a *safe state* or identify that it is not in a *safe state*.

(b) P4, P3, P1, P2

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(c) (3pts) Suppose that process P_1 makes a request for $C = 1$. Would this result in a safe state (describe why or why not)?

- Not safe: Can run P4, but no one else

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(d) (4pts) Suppose that the request in step (c) is granted by the system (from the free resources in (b)). Will the following request result in a *deadlock*? $P_1 = (3,2,2)$, $P_2 = (5,0,1)$, $P_3 = (1,0,3)$, $P_4 = (1,1,0)$. If there is no deadlock, show a legal process sequence.

Allocated request

P1 1/2/1 3/2/2

P2 2/1/0 5/0/1

P3 1/0/3 1/0/3

P4 2/3/2 1/1/0

1 3 0

P4 3 6 2

P1 4 8 3

P3 5 8 6

P2 7 9 6

3 4 2

HW3 #6

6. (*12pts*) Consider a system with 4 processes (P_1 through P_4) and 3 resource types A, B, C. Suppose at time T_0 the following snapshot has been taken:

Process ID	Allocated (A/B/C)	Max (A/B/C)
P_1	1/2/1	4/4/5
P_2	2/1/0	7/1/2
P_3	1/0/3	2/2/6
P_4	2/3/2	3/6/2

(d) (4pts) Suppose that the request in step (c) is granted by the system (from the free resources in (b)). Will the following request result in a *deadlock*? $P_1 = (3, 2, 2)$, $P_2 = (5, 0, 1)$, $P_3 = (1, 0, 3)$, $P_4 = (1, 1, 0)$. If there is no deadlock, show a legal process sequence.

(d) No deadlock

after P4: 3/4/2
after P1: 4/6/3
after P3: 5/6/6
after P2: 7/7/6

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(a) (2pts) Supposing different threads can execute `streamInput` and `streamOutput`, identify the shared data in the two functions.

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(a) (2pts) Supposing different threads can execute `streamInput` and `streamOutput`, identify the shared data in the two functions.

(a) StreamEntries and count

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(b) (2pts) State how you would modify the code of *streamInput* to enforce mutual exclusion to these variable(s). Use `mutex_lock` and `mutex_unlock`.

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(b) (2pts) State how you would modify the code of *streamInput* to enforce mutual exclusion to these variable(s). Use `mutex_lock` and `mutex_unlock`.

(b) add mutex_locks around streamEntries and count in streamInput

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(c) (2pts) To which classical synchronization problem does the code above correspond?

(d) (3pts) State how you would modify the code above (both functions) (using *semaphores*) to ensure that a thread blocks if it tries to output a stream entry where none is available, but is awoken when entries become available,

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(c) (2pts) To which classical synchronization problem does the code above correspond?

(c) bounded buffer

(d) (3pts) State how you would modify the code above (both functions) (using *semaphores*) to ensure that a thread blocks if it tries to output a stream entry where none is available, but is awoken when entries become available,

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(e) (3pts) If there are many readers (callers of `streamOutput`) and only one writer (caller of `streamInput`), can multiple readers run `streamOutput` concurrently? Why or why not?

HW3 #7

7. (*12pts*) Answer questions about concurrency control for the following code.

Data Stream Code

```
        entry StreamEntries[Size];
        int count;

void streamInput( entry *E )
{
        StreamEntries[count] = E;
        count = count + 1;

}

entry *streamOutput( )
{
        entry *E = StreamEntries[count];
        count = count - 1;

        return E;
}
```

(e) (3pts) If there are many readers (callers of `streamOutput`) and only one writer (caller of `streamInput`), can multiple readers run `streamOutput` concurrently? Why or why not?

(e) No. Even "readers" modify streamEntries buffer.

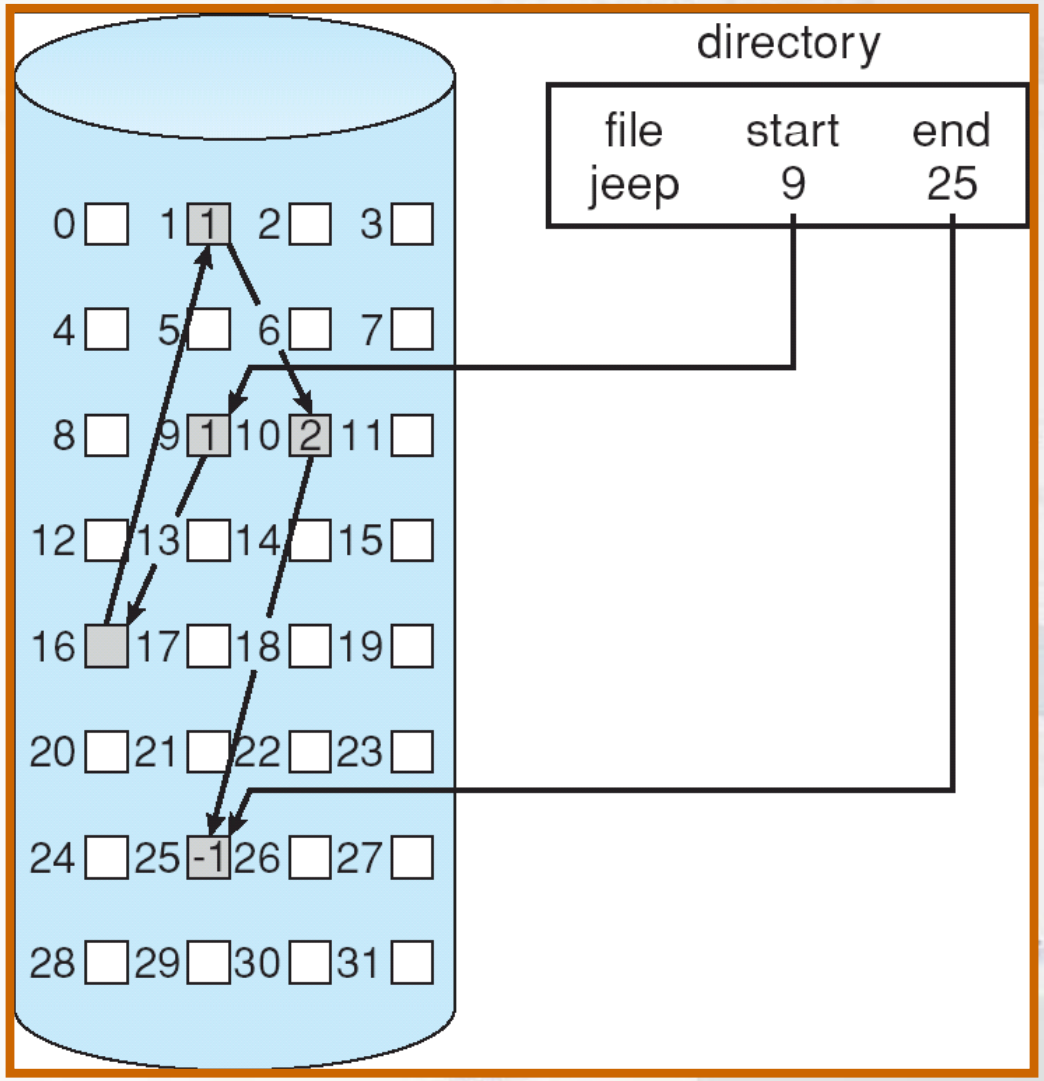
HW4 #1

1. (*10pts*) What is the problem with allocating file data blocks in a linked list where the first few bytes of each data block refer to the next data block of the file? How does a UNIX file system eliminate this problem?

- Recall FAT filesystem
  - What problem it was designed to avoid

File Allocation: Linked List

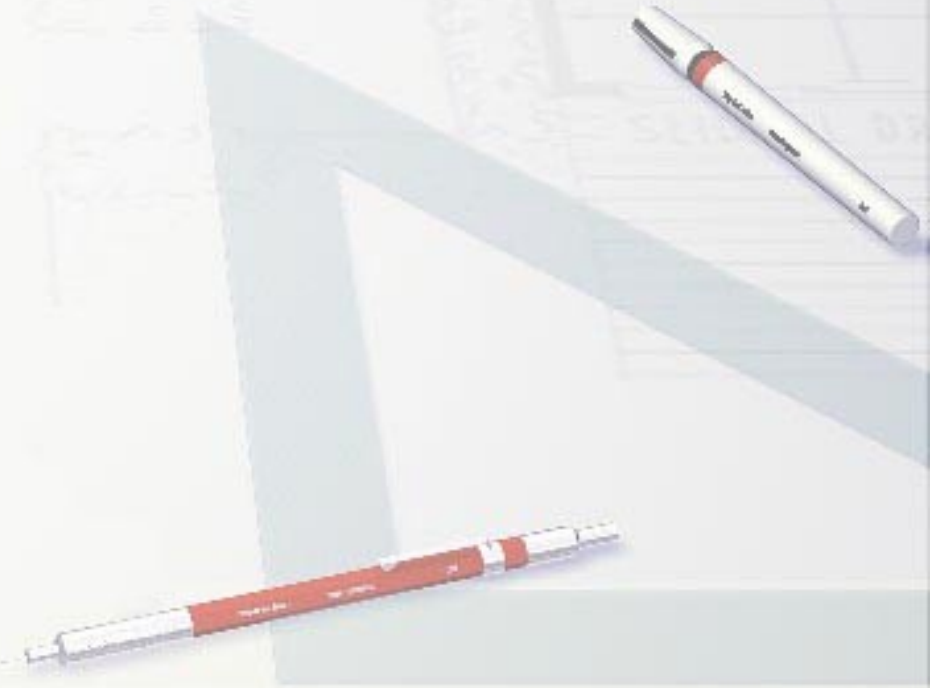
- Linked list
- S.L.O.W.!



HW4 #2

2. (*10pts*) Name three specific file system inconsistencies that may be caused by a machine crash?

- Recall fsck and journaling

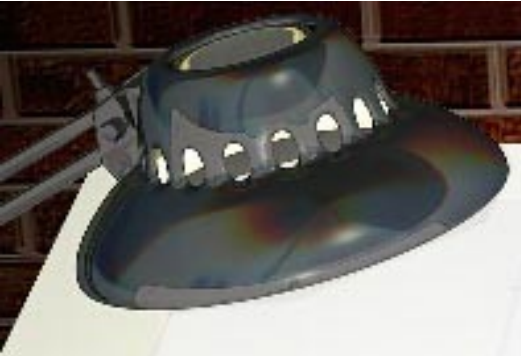


HW4 #2

2. (*10pts*) Name three specific file system inconsistencies that may be caused by a machine crash?

- Recall fsck and journaling

answer: Cached file data blocks may be lost if write-back caching is used. Inode to block pointers may be inconsistent. File to directory pointers may become inconsistent. Free block state may become inconsistent.



HW4 #3

3. (*10pts*) Detail the OS steps to implement the following UNIX system calls for file processing (assume that the file object is only on disk initially):
   (a) open the file
   (b) read a block from that file
   (c) read a block from a raw disk device file (already opened)

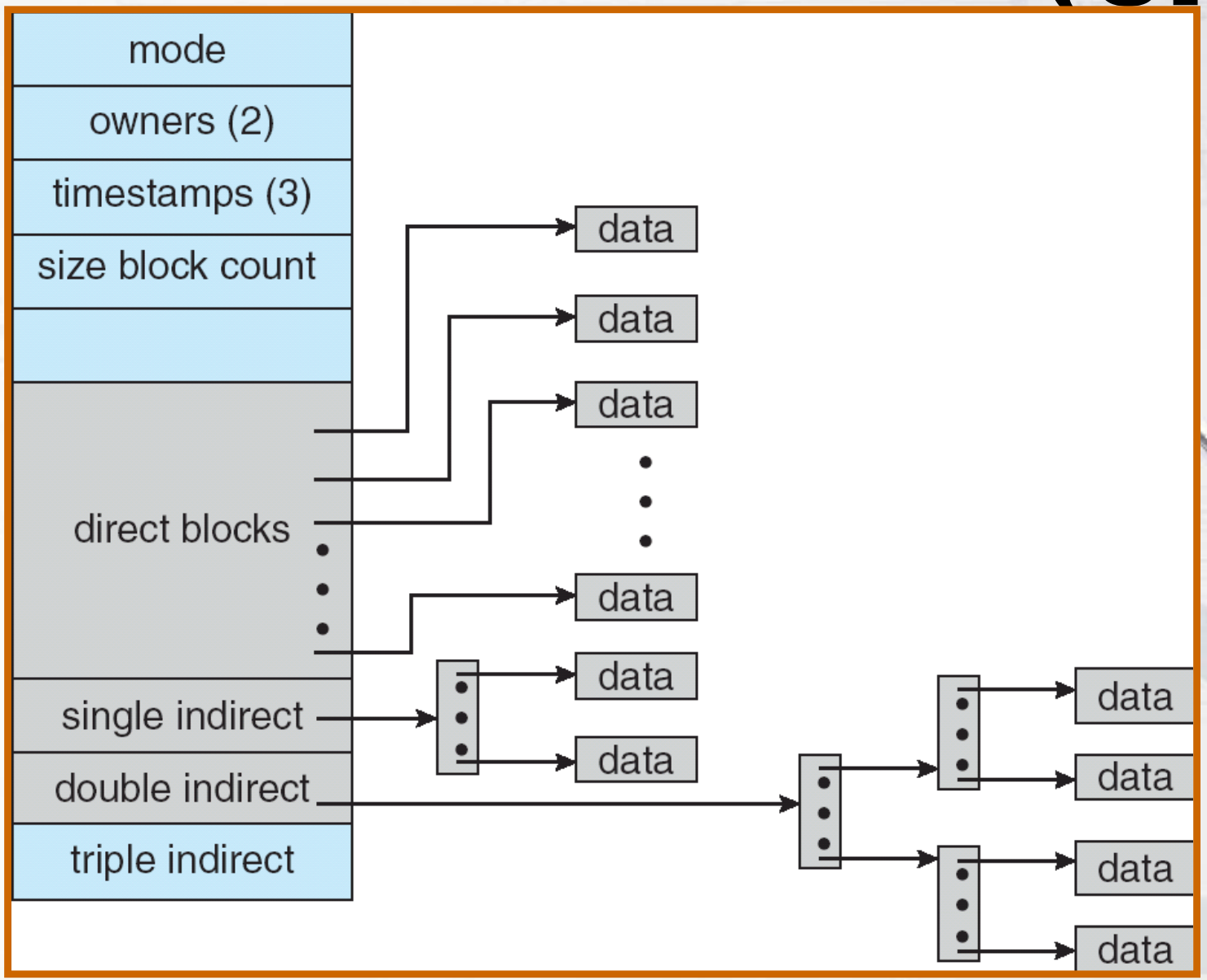
- Recall UNIX filesystem operations

Accessing a file block in UNIX (con't): /a/b/c

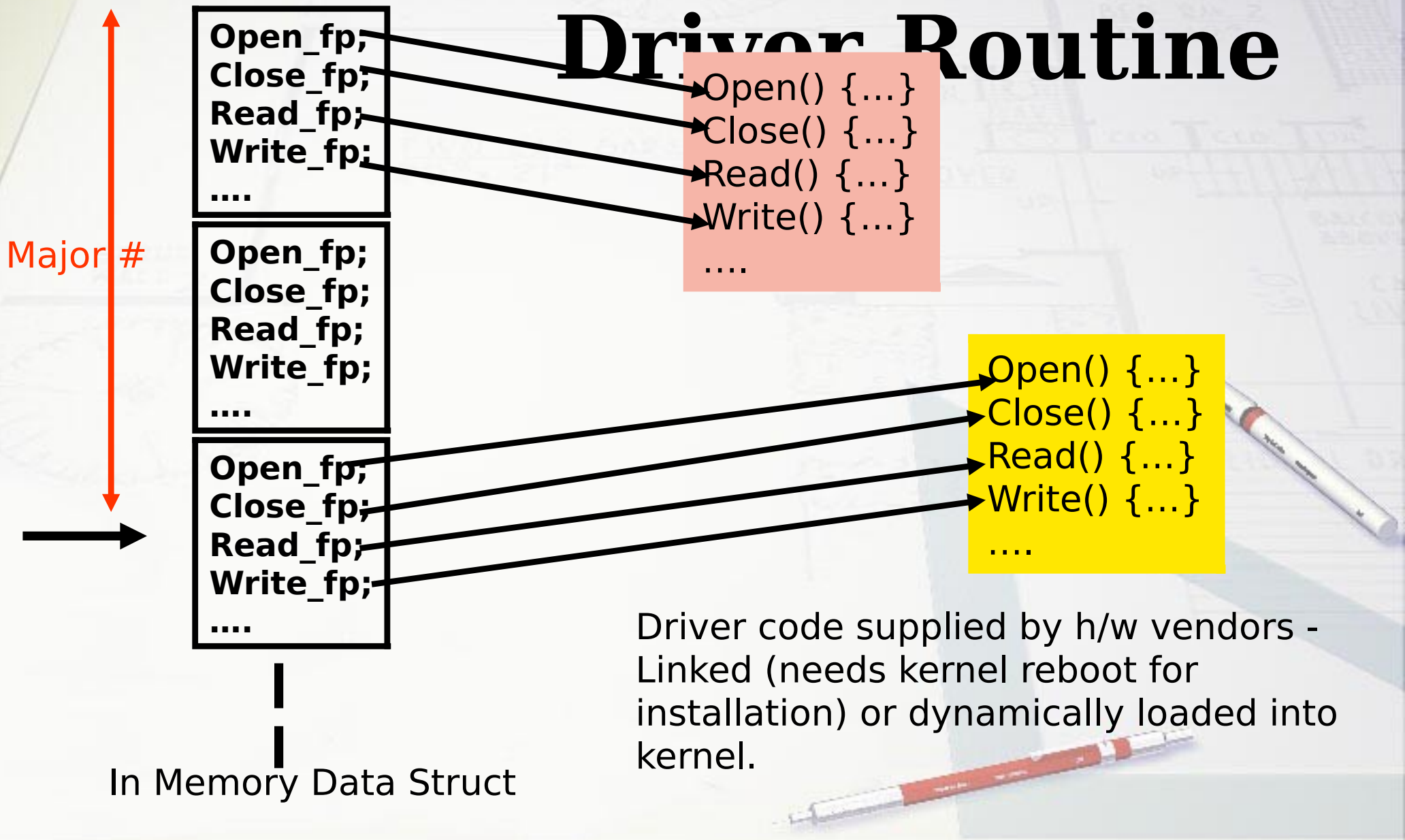
- Get block after block of "b" until entry for "c" is found (gives its inode #)
- Get inode of "c" from disk
- Find out whether block you are searching for is in 1st 10 ptrs, or 1-level or 2-level or 3-level indirect.
- Based on this you can either directly get the block or retrieve it after going through the levels of indirection.

- Imagine searching through the inodes each time you do a read() or write() on a file
- Too much overhead!
- However, once you have the i-node of the file (or a FAT entry in DOS), then it is fairly efficient!
- You want to cache the inode (or the id of the FAT entry) for a file in memory and keep re-using it.

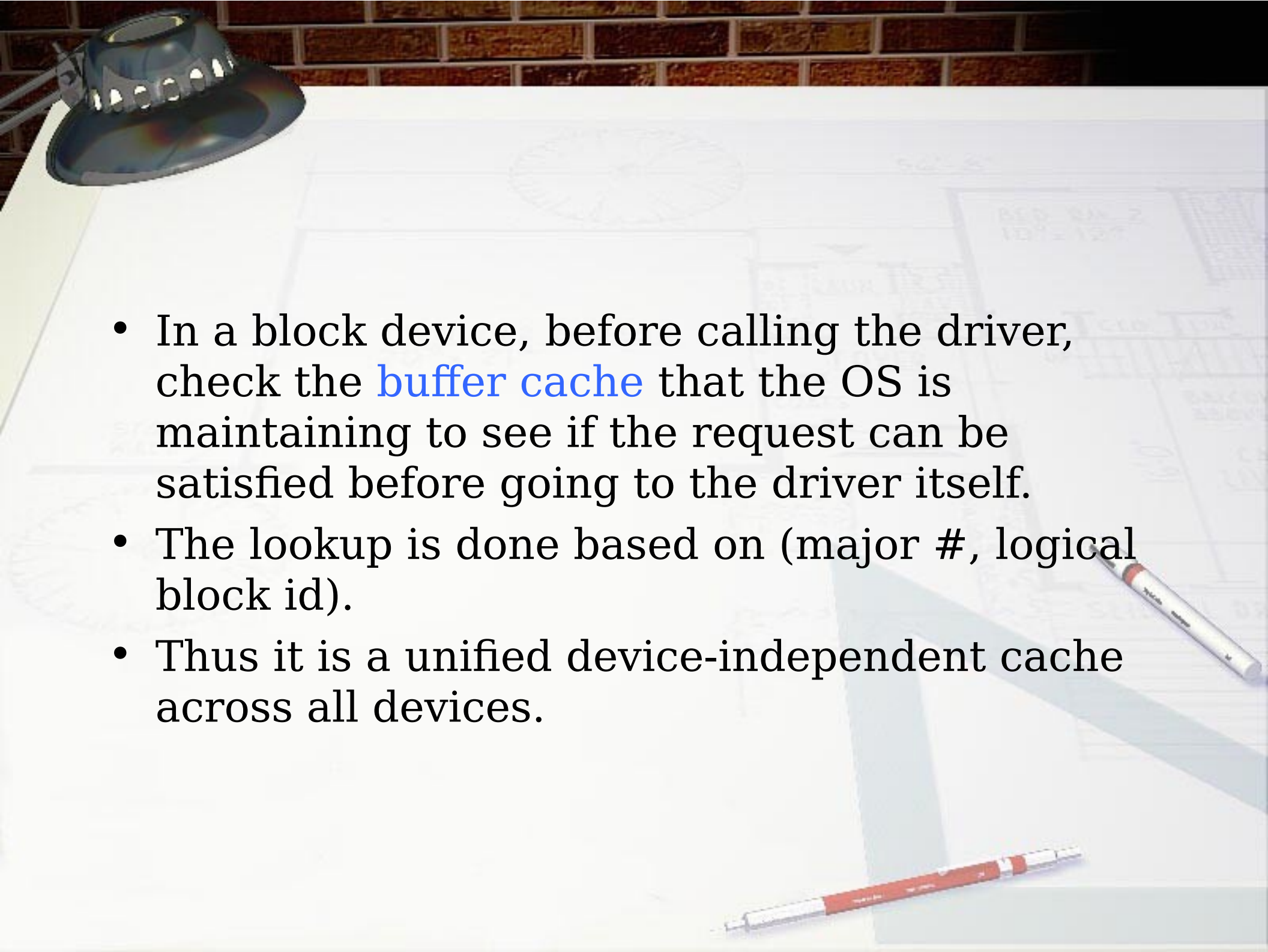
Direct & Indirect Blocks (UNIX)



Getting to the Driver Routine



Driver code supplied by h/w vendors - Linked (needs kernel reboot for installation) or dynamically loaded into kernel.



- In a block device, before calling the driver, check the buffer cache that the OS is maintaining to see if the request can be satisfied before going to the driver itself.
- The lookup is done based on (major #, logical block id).
- Thus it is a unified device-independent cache across all devices.

HW4 #4

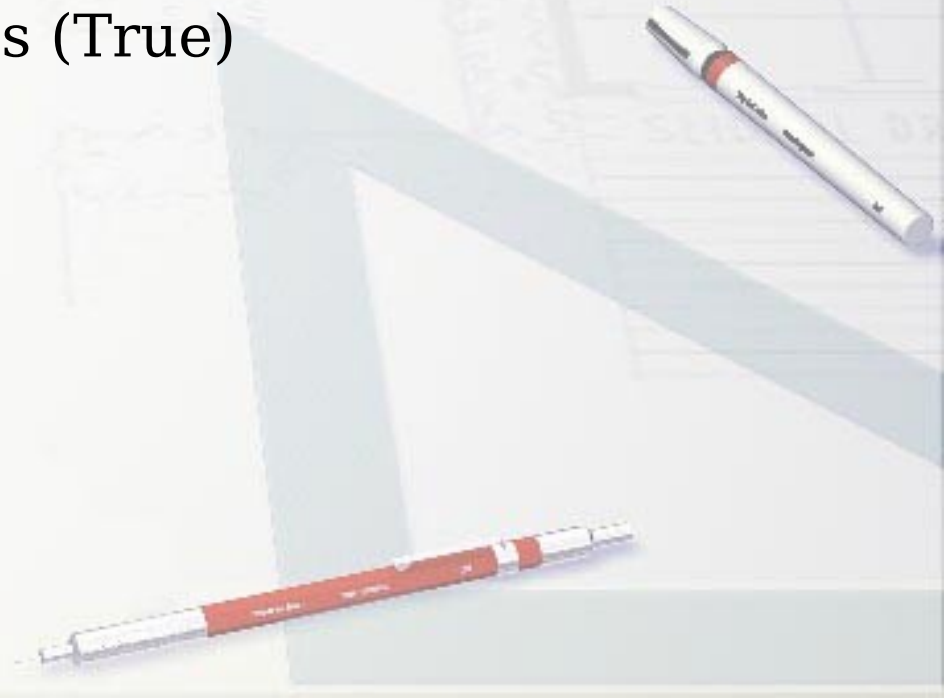
4. (*5pts*) True or False: a read() system call on a directory may result in a disk write operation. Explain your answer.

- What happens to an inode during a read operation?

HW4 #4

4. (*5pts*) True or False: a read() system call on a directory may result in a disk write operation. Explain your answer.

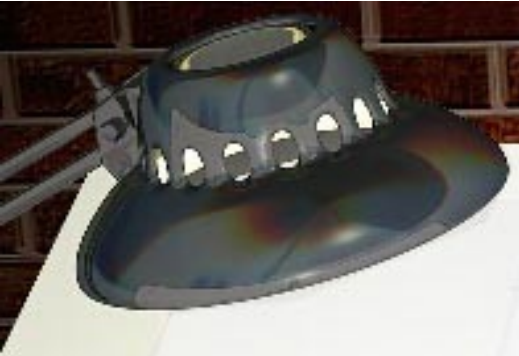
- What happens to an inode during a read operation?
  - May update inode attributes (True)



HW4 #5

5. (*10pts*) What is the maximum file size in a system with i-nodes of size 256 Bytes with the following properties: (i) 128 Bytes are used for storing various file properties leaving 128 Bytes for layout information, (ii) 72 Bytes for direct pointers, (iii) 32 Bytes for single indirect pointers, (iv) 16 Bytes for double indirect pointers, and (v) 8 Bytes for triple indirect pointers. Finally, each pointer is 8 Bytes long and the block size is 4KB.

- Direct and indirect pointers



HW4 #5

5. (*10pts*) What is the maximum file size in a system with i-nodes of size 256 Bytes with the following properties: (i) 128 Bytes are used for storing various file properties leaving 128 Bytes for layout information, (ii) 72 Bytes for direct pointers, (iii) 32 Bytes for single indirect pointers, (iv) 16 Bytes for double indirect pointers, and (v) 8 Bytes for triple indirect pointers. Finally, each pointer is 8 Bytes long and the block size is 4KB.

- Direct and indirect pointers

answer: (9 * 4096) + (4 * 512 * 4096) + (2 * 512 * 512 * 4096) + (1 * 512 * 512 * 512 * 4096) = ...
— note: 512 = 4K (blocksize) / 8 (bytes per ptr)

HW4 #6

6. (*10pts*) Answer OSTEP Homework question 39.3 (end of Chapter 39) regarding the `tail` command

- Interfaces
  - Stat, lseek, open, read, close

HW4 #6

6. (*10pts*) Answer OSTEP Homework question 39.3 (end of Chapter 39) regarding the `tail` command

- Interfaces
  - Stat, lseek, open, read, close
- Open the file
  - Open
- Seek to place k bytes from end
  - Lseek with offset of –k and SEEK_END directive for whence
- Read from there to end
  - Read k bytes
- Close

HW4 #7

6. (*10pts*) Answer OSTEP Homework question 39.3 (end of Chapter 39) regarding the `tail` command

- Interfaces
  - Stat, lseek, open, read, close
- Open the file
  - Open
- Seek to place k bytes from end
  - Lseek with offset of –k and SEEK_END directive for whence
- Read from there to end
  - Read k bytes
- Close

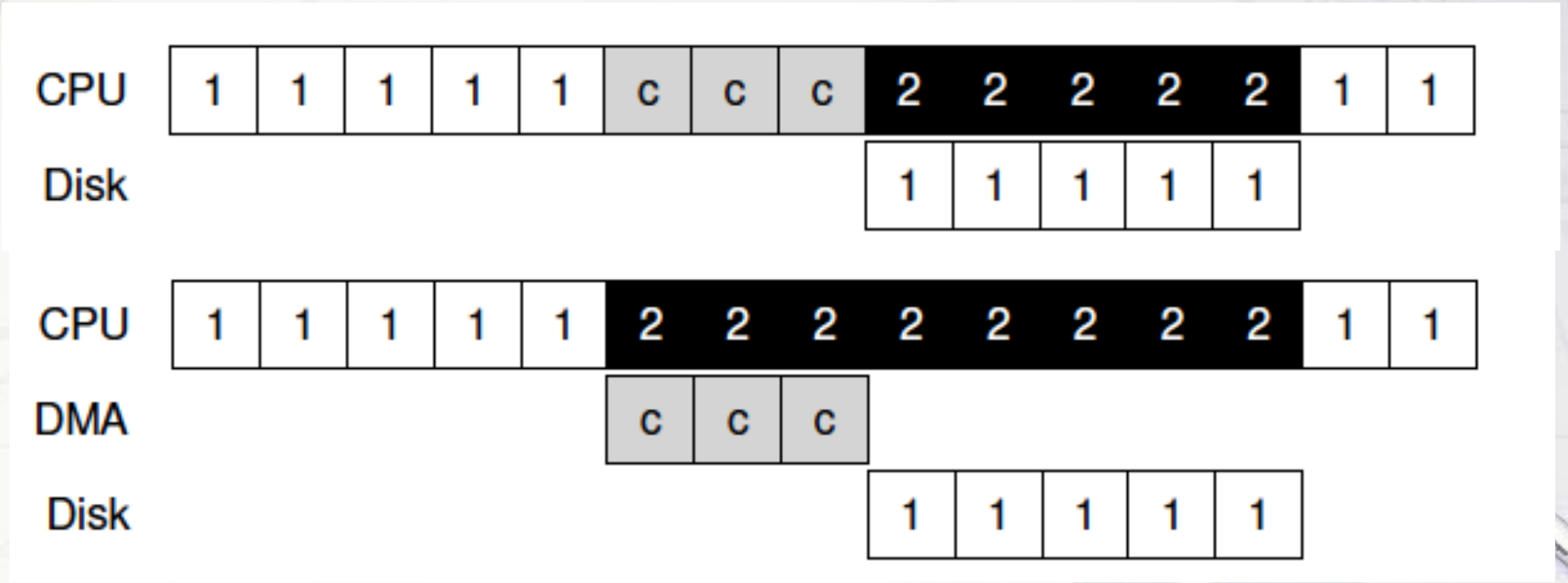
HW4 #7

7. (*10pts*) Suppose a process runs for 10 time units and then reads from a file. If the file reading takes 5 time units and it takes 5 time units to copy the data from the disk device to main memory, how many free cycles are made available to run other processes if the disk device uses DMA and the disk driver uses interrupts?

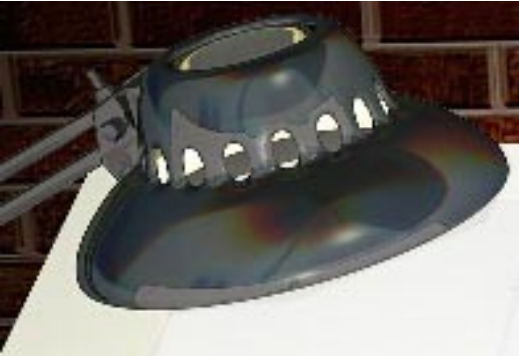
- Direct Memory Access parallelism

Direct Memory Access

- E.g., writing some data to the disk



- DMA engine takes care of issuing copying-related commands, and interrupts (main) CPU when finished
- Single DMA in older systems, per-device DMA these days
- FFT: role of MMU and virtual memory manager



HW4 #7

7. (*10pts*) Suppose a process runs for 10 time units and then reads from a file. If the file reading takes 5 time units and it takes 5 time units to copy the data from the disk device to main memory, how many free cycles are made available to run other processes if the disk device uses DMA and the disk driver uses interrupts?

- Direct Memory Access parallelism

answer: All the reading and copying will be available to another process - so 10 time units

HW4 #8

8. (*10pts*) Consider the set of requests below in disk scheduling (min cylinder: 0; max cylinder 99; drive head is at cylinder 41): 16, 46, 81, 11, 76, 51.

   (a) (2pts) What is the required head movement for *first-come, first-served*?

   (b) (2pts) What is the required head movement for *shortest seek time first*?

   (c) (2pts) What is the required head movement for *Circular SCAN*?

   (d) (2pts) What is the required head movement for *Circular-LOOK*?

HW4 #8

8. (*10pts*) Consider the set of requests below in disk scheduling (min cylinder: 0; max cylinder 99; drive head is at cylinder 41): 16, 46, 81, 11, 76, 51.

   (a) (2pts) What is the required head movement for *first-come, first-served*?

   (a) 25+30+35+70+65+25=250

   (b) (2pts) What is the required head movement for *shortest seek time first*?

   (b) (41→46)5+(46→51)5+(51→76)25+(76→81)5+(81→16)65+(16→11)5=110

   (c) (2pts) What is the required head movement for *Circular SCAN*?

   (c) (41→46)5+(46→51)5+(51→76)25+(76→81)5+(81→99)18+100+(0→11)11+(11→16)5=174

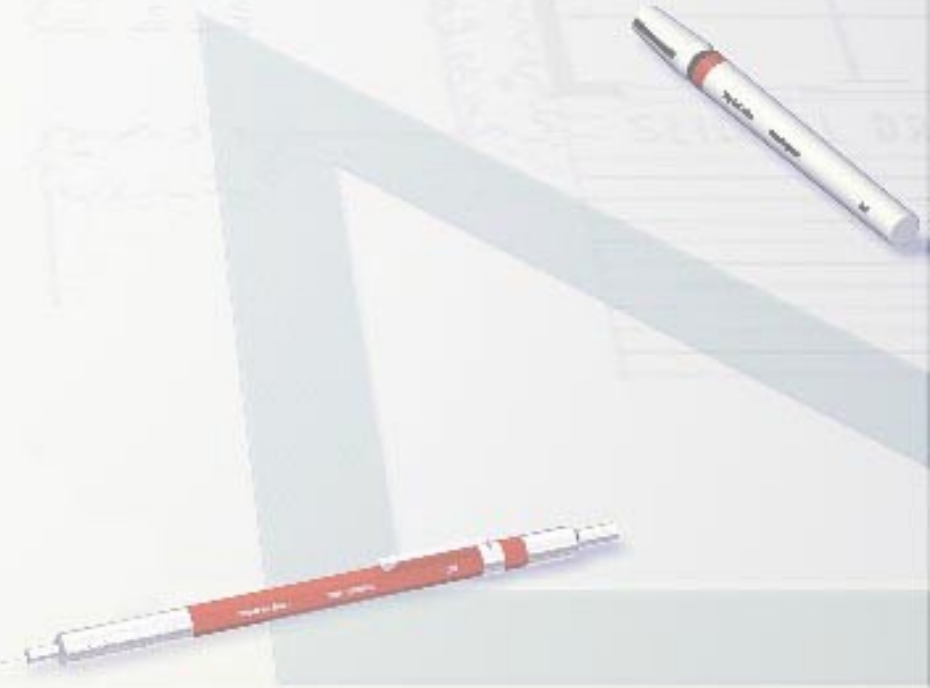
   (d) (2pts) What is the required head movement for *Circular-LOOK*?

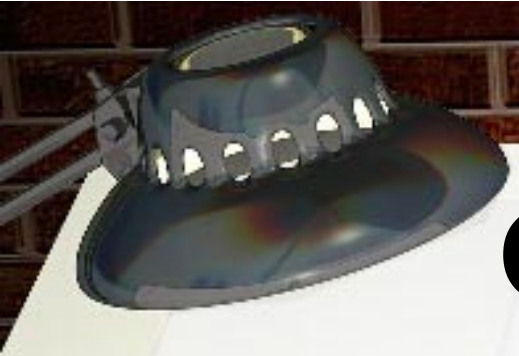
   (d) (41→46)5+(46→51)5+(51→76)25+(76→81)5+(81→11)70+(11→16)5=115

HW4 #8

(e) (2pts) What is the main limitation of *shortest seek time first*?

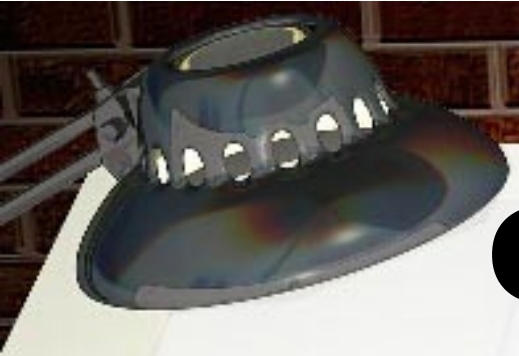
- Starvation of requests on cylinders near the ends of the disk





CPU Virtualization - Things

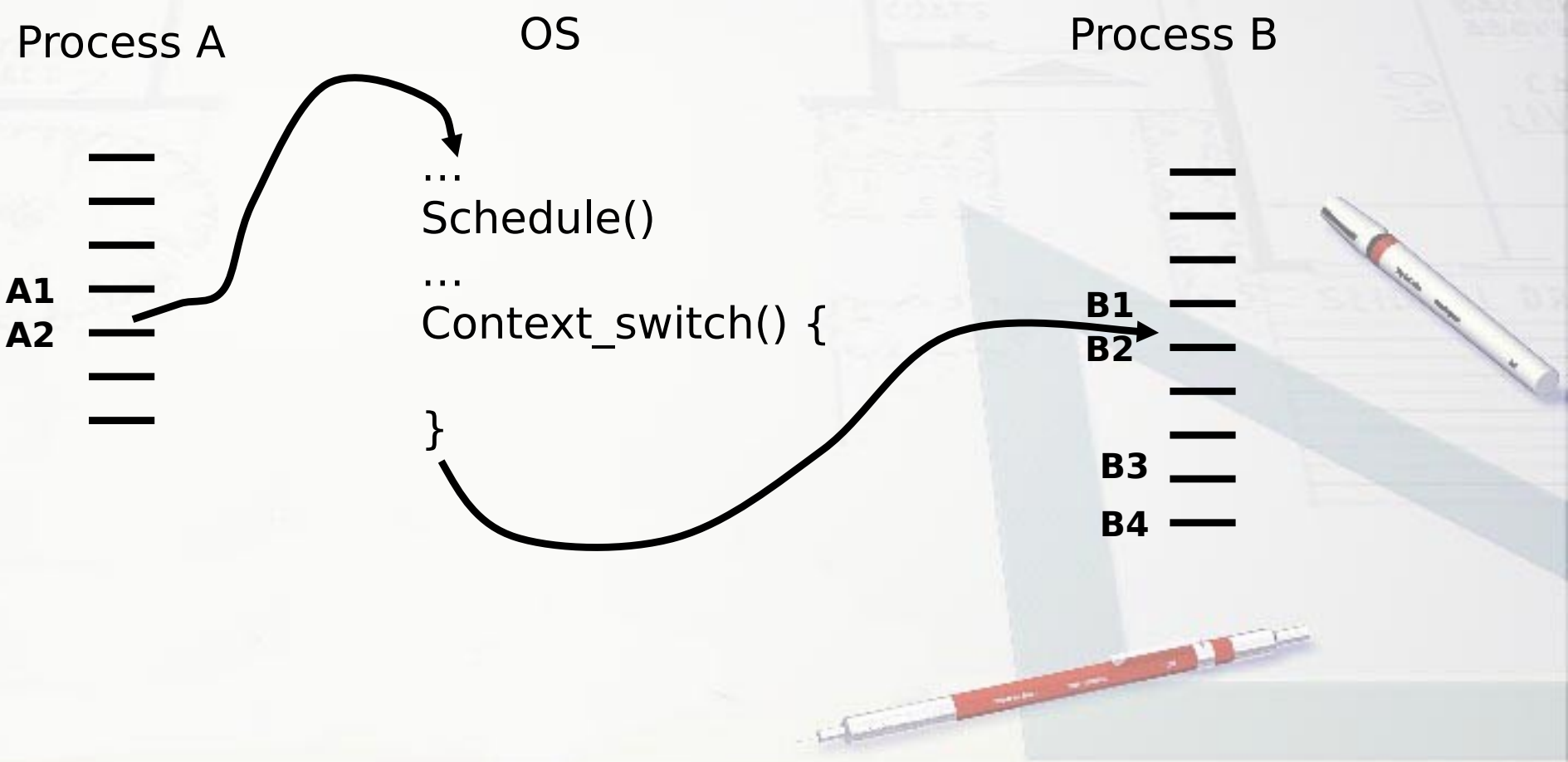
- Address space
- Kernel and user modes
- Traps and interrupts
- Context switch
- Process Control Block
- Execution (scheduling) states



CPU Virtualization – Problems

- Where to store different types of data? Address space
  - Remember kernel memory is mapped into process address space
- How to limit access to privileged instructions? Kernel and user modes
  - When user mode code executes a privileged instruction - trap
- How to get the OS to run? Traps and interrupts
  - What's the difference between them?
- How to switch from one process to another? Context switch
- How to store user state? Which to store? Process Control Block
- How to determine which processes can run? Execution (scheduling) states

Context Switch

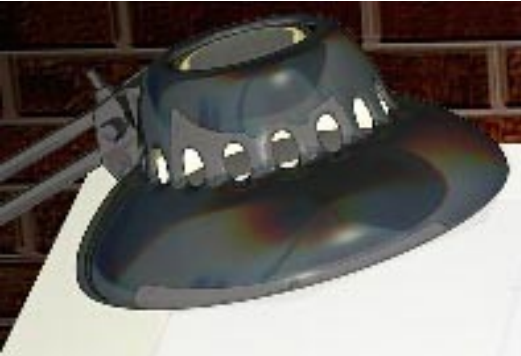


Context Switch

OS @ boot (kernel mode)	Hardware	
initialize trap table		
	remember addresses of...	
	syscall handler	
	timer handler	
start interrupt timer		
	start timer	
	interrupt CPU in X ms	

OS @ run (kernel mode)	Hardware	Program (user mode)
		Process A
		...
	timer interrupt	
	save regs(A) to k-stack(A)	
	move to kernel mode	
	jump to trap handler	
Handle the trap		
Call `switch()` routine		
save regs(A) to proc-struct(A)		
restore regs(B) from proc-struct(B)		
switch to k-stack(B)		
return-from-trap (into B)		
	restore regs(B) from k-stack(B)	
	move to user mode	
	jump to B's PC	
		Process B
		...

Figure 6.3: **Limited Direct Execution Protocol (Timer Interrupt)**



CPU Scheduling - Things

- Scheduling queues
- Scheduling algorithms
- Preemption
- Fairness
- Waiting Time
- CPU Burst
- Multilevel Feedback Queues
- Proportional Share

CPU Scheduling - Problems

- How to model the scheduling problem? Scheduling queues
- How to choose what to schedule? Scheduling algorithms
- How to determine when to make decisions? Preemption
- What to ensure/optimize? Fairness and Waiting Time
- How to estimate the impact of a scheduling decision? CPU Burst
- How to design fair and effective scheduling? Multilevel Feedback Queues and Proportional Share

Example of Multilevel Feedback Queue

2. (*5pts*) Consider the following *preemptive multilevel queue scheduler* and process information. Note the quantum restarts on each reschedule.

   - Queue Q_0 (Priority 10, Quantum 3): Schedule Processes in Shortest Job First and Demote to Q_1 if use quantum
   - Queue Q_1 (Priority 5, Quantum 7): Schedule Processes in Round Robin and Promote to Q_0 if not use quantum

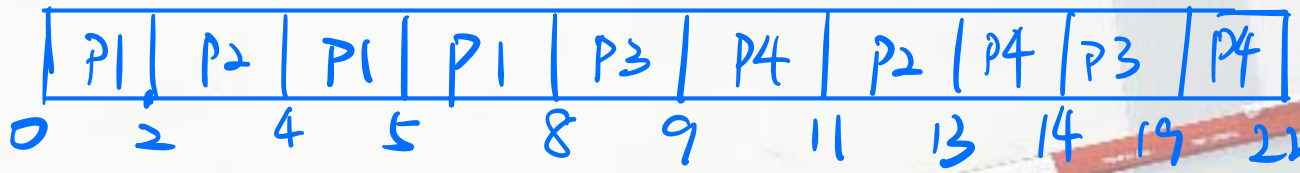
Process ID	CPU Burst Time	Arrival Time	Priority (Queue)
P1	6	0	10
P2	2	2	10
P3	6	4	5
P4	6	9	10
P2	2	11	?

   Which process will be running (or finishing) at time 10?

   answer: (A) P1; (B) P2; (C) P3; (D) P4

3. (*5pts*) Based on the table for question 2, which process will be running (or finishing) at time 21?

   answer: (A) P1; (B) P2; (C) P3; (D) P4



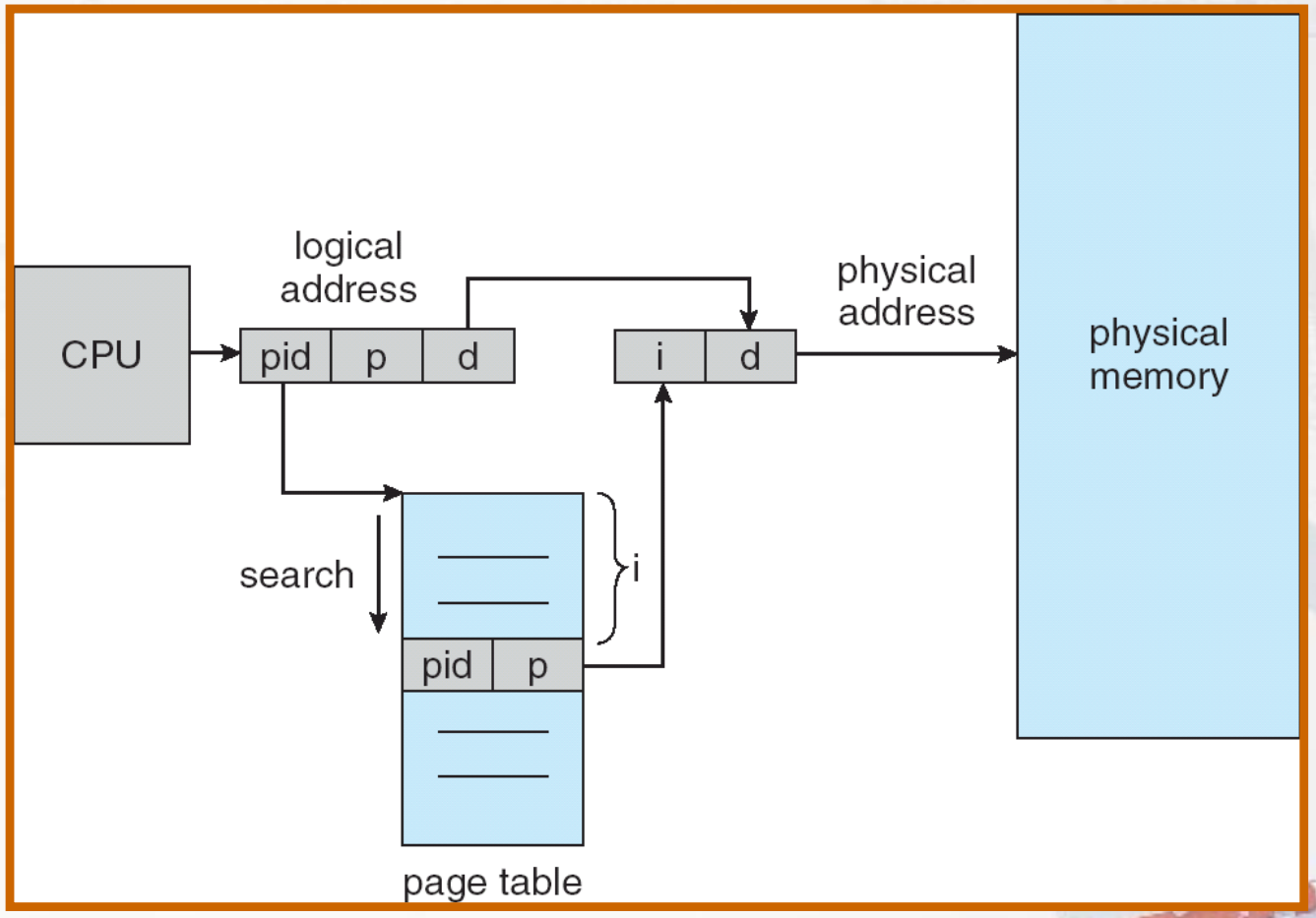
Memory Virtualization - Things

- Virtual memory vs Physical memory
- Address translation
- Allocation approach – contiguous, non-contiguous
- Allocation algorithms
- Fragmentation
- Paging system – MMU, TLB, Page Tables,
- Page table design

Memory Virtualization - Things

- How to abstract physical configuration? Virtual memory vs Physical memory
- How to map between abstraction and physical? Address translation
- How to allocate physical memory resource? contiguous, non-contiguous and algorithms
- How to avoid wasting the resource? Fragmentation
- How to implement efficient non-contiguous? Paging system – MMU, TLB, Page Tables
- How to find page entries quickly with modest memory overhead? Page table design

Inverted Page Table Architecture



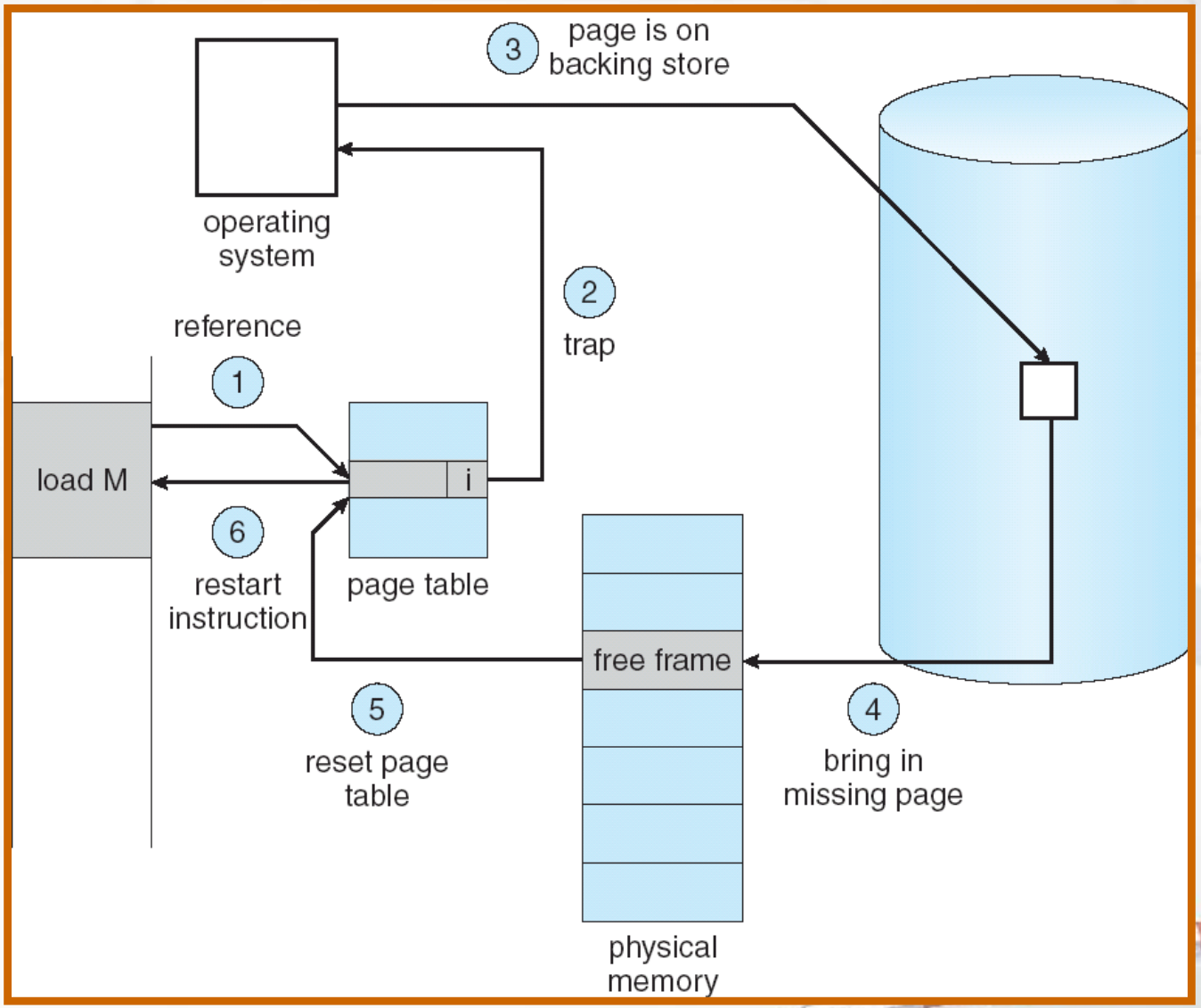
Paging - Things

- Page Fault Handling
- Demand Paging
- Page Replacement
- Page Replacement Algorithms
- Belady’s anomaly
- Shared memory
- Copy-on-write

Paging - Things

- What happens when no free frames? Page Fault Handling and Demand Paging
- How to choose a frame to evict? Page Replacement Algorithms
- What property should a good replacement algorithm have (or not have)? Belady's anomaly
- How do we reuse frames for same stuff? Shared memory
- How do we optimize process creation? Copy-on-write

Steps in Handling a Page Fault



Threads - Things

- Multi- vs single-threaded (sharing)
- Address space
- Thread Control Block
- Threading models
- Kernel vs. User threads
- User-level thread library
- Signals
- Thread context switching

Threads - Things

- How to minimize redundancy? Multi- vs single-threaded (sharing)
- How does multi-threading change OS concepts? Address space and Thread Control Block
- How to schedule threads? Threading models and Kernel vs User threads
- How to maintain threads in user-space? User-level thread library
- How to get OS (or other process) to notify? Signals
- How to switch among threads in user-space? Thread context switching

Multi-Threaded vs. Single-Threaded