# CS 558: COMPUTER SYSTEMS (OS)

## TUTORIAL

Dr. Samit Bhattacharya
Dept. of Comp. Sc. & Engg.,
IIT Guwahati, Assam, India

# *Processes & Scheduling*

# PROCESS

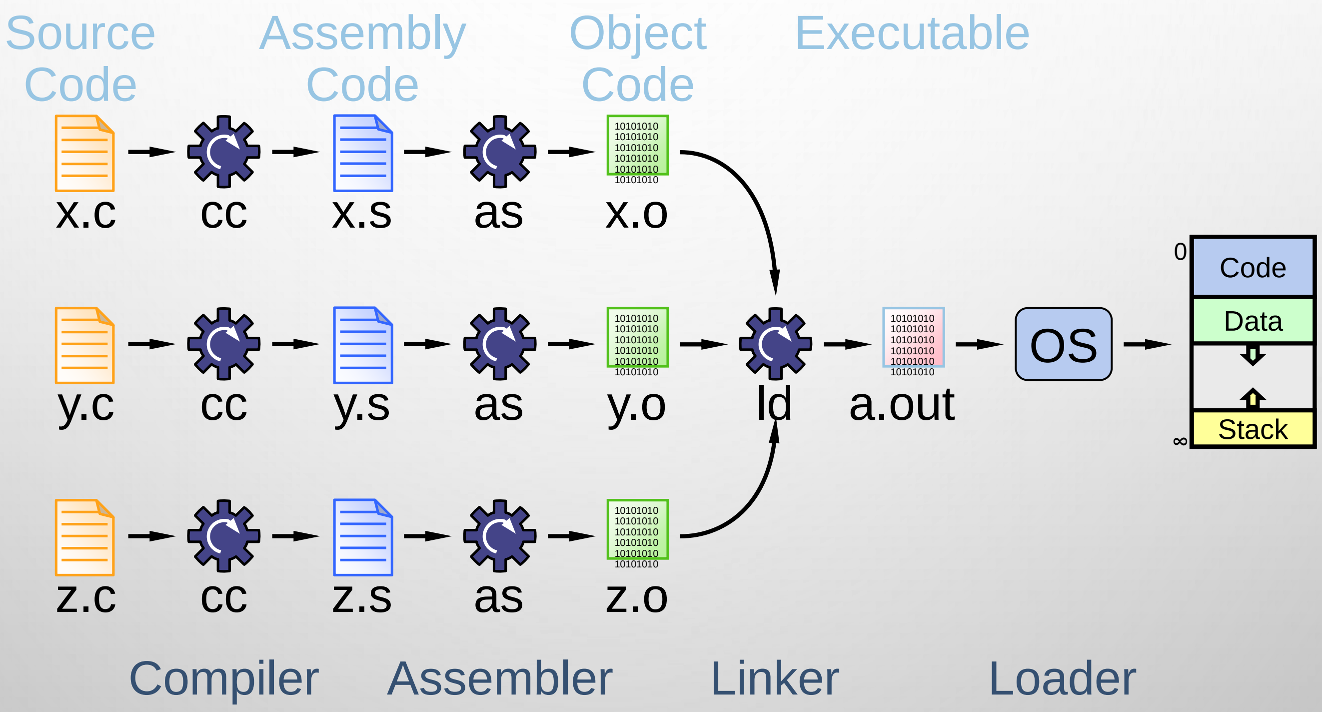
- PROGRAM DURING EXECUTION
  - PROGRAM = STATIC FILE (IMAGE)
  - PROCESS = EXECUTING PROGRAM = PROGRAM + EXECUTION STATE
- BASIC UNIT OF EXECUTION IN AN OS

# PROGRAM TO PROCESS

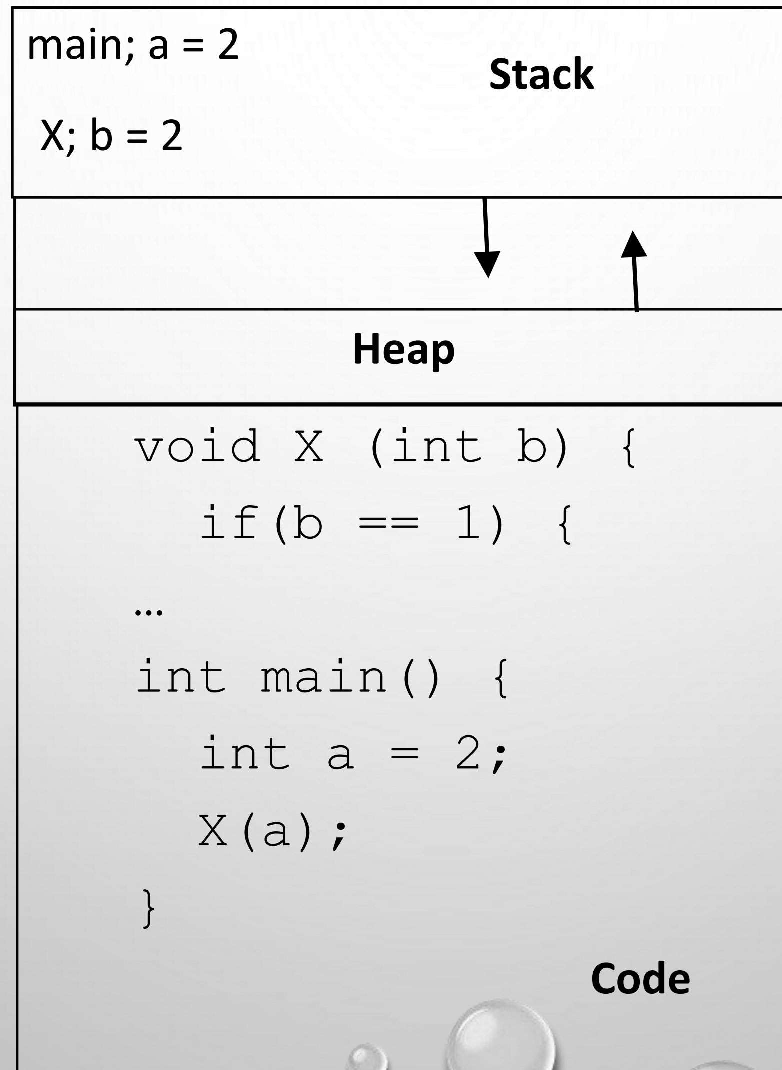
- Consider the following program

```
void X (int b) {
   if(b == 1) {
…
int main() {
   int a = 2;
   X(a);
}
```

# CREATING A PROCESS

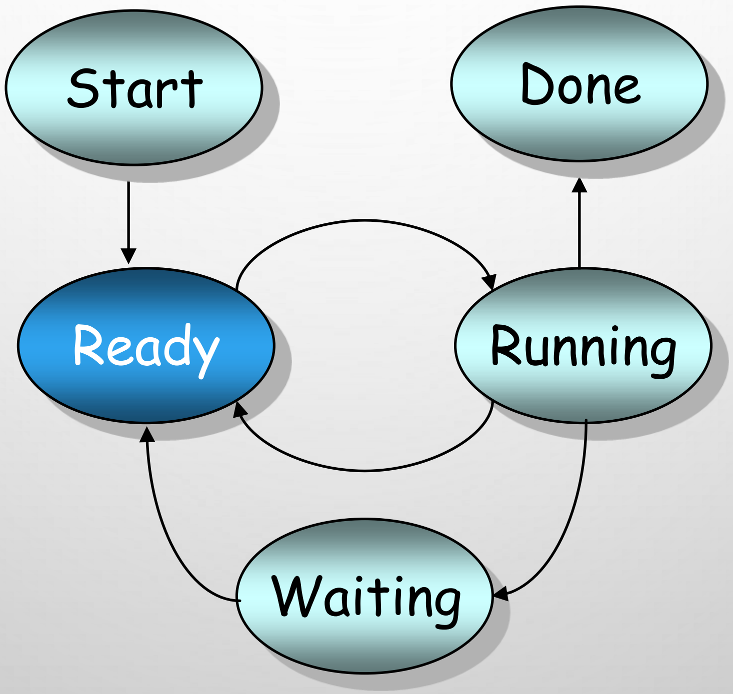


# PROCESS IN MEMORY



# PROCESS LIFE CYCLE

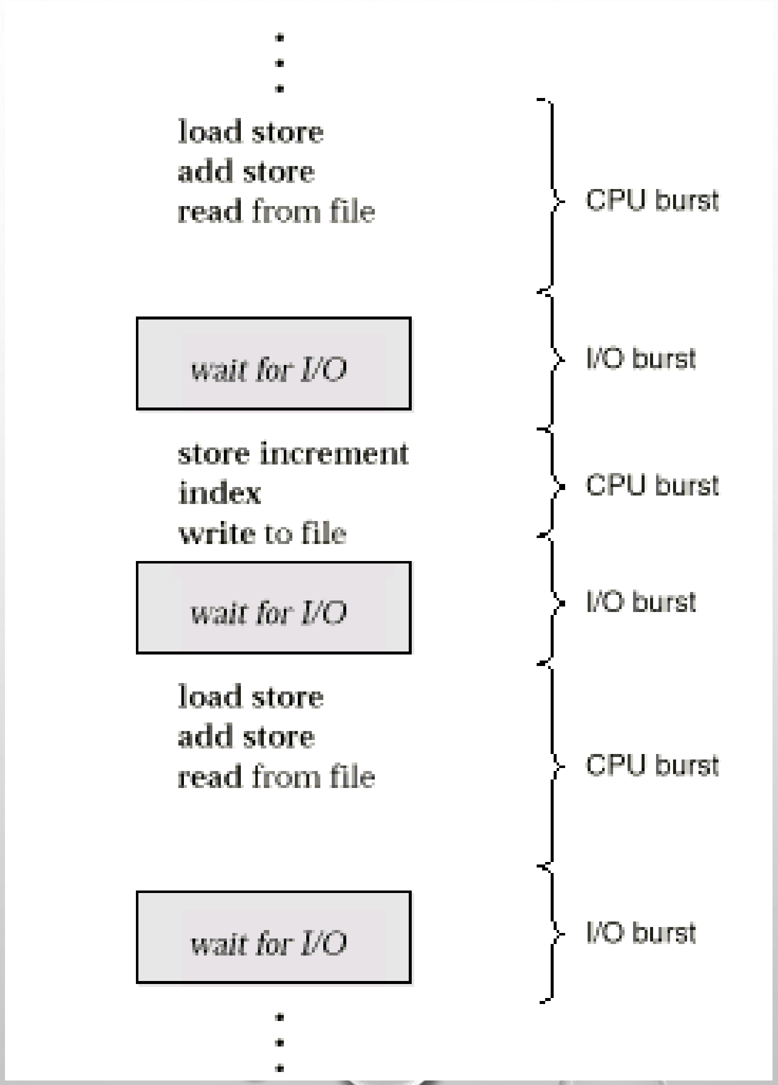
- PROCESSES ARE ALWAYS EITHER *EXECUTING*, *WAITING TO EXECUTE* OR *WAITING FOR AN EVENT* TO OCCUR



# PROCESS LIFE CYCLE

- A *preemptive scheduler* will force a transition from running to ready
- A non-preemptive scheduler *waits*

# ALTERNATING CPU AND I/O BURSTS



# TWO OS MODULES

- SCHEDULER
- DISPATCHER

# SCHEDULER

- SELECTS FROM THE *READY* PROCESSES
- SCHEDULING DECISIONS OCCUR WHEN PROCESS
  1. SWITCHES FROM RUNNING TO WAITING STATE
  2. SWITCHES FROM RUNNING TO READY STATE
  3. SWITCHES FROM WAITING TO READY
  4. TERMINATES

# DISPATCHER

- GIVES CONTROL OF THE CPU TO THE PROCESS SELECTED BY THE SCHEDULER
- INVOLVES
  - SWITCHING CONTEXT
  - SWITCHING TO USER MODE
  - JUMPING TO THE PROPER LOCATION IN THE USER PROGRAM TO RESTART THAT PROGRAM
- *DISPATCH LATENCY* – TIME IT TAKES FOR THE DISPATCHER TO STOP ONE PROCESS AND START ANOTHER RUNNING

# FIRST-COME, FIRST-SERVED (FCFS)

| PROCESS | BURST TIME |
|---|---|
| $P_1$ | 24 |
| $P_2$ | 3 |
| $P_3$ | 3 |

- ASSUME PROCESSES ARRIVE AS: $P_1$ , $P_2$ , $P_3$
  THE GANTT CHART FOR THE SCHEDULE IS:


- WAITING TIME FOR $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- AVERAGE WAITING TIME: $(0 + 24 + 27)/3 = 17$

# FCFS SCHEDULING (CONT.)

- NOW CONSIDER AN ALTERNATIVE SEQUENCE OF PROCESS ARRIVAL


- WAITING TIME FOR $P_1 = 6; P_2 = 0; P_3 = 3$
- AVERAGE WAITING TIME: $(6 + 0 + 3)/3 = 3$
- MUCH BETTER THAN PREVIOUS CASE

- *CONVOY EFFECT* OR *HEAD-OF-LINE BLOCKING*
  - SHORT PROCESS BEHIND LONG PROCESS

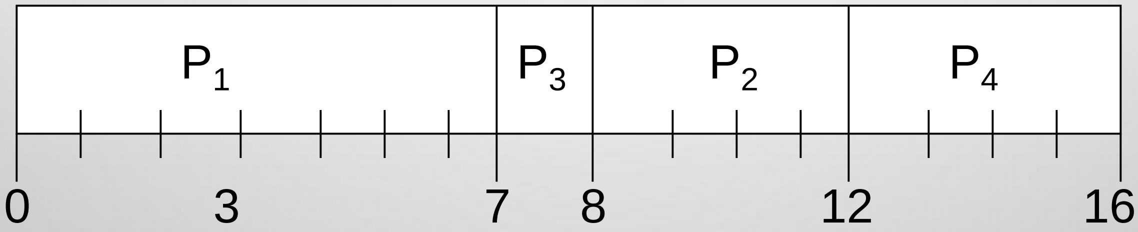
# SHORTEST-JOB-FIRST (SJR) SCHEDULING

- PROCESS DECLARES ITS CPU BURST LENGTH
- TWO SCHEMES
  - *NON-PREEMPTIVE* – ONCE CPU ASSIGNED, PROCESS NOT PREEMPTED UNTIL ITS CPU BURST COMPLETES
  - *PREEMPTIVE* – IF A NEW PROCESS WITH CPU BURST LESS THAN REMAINING TIME OF CURRENT, PREEMPT
    SHORTEST-REMAINING-TIME-FIRST (SRTF)
- **SJF IS OPTIMAL** – GIVES MINIMUM AVERAGE WAITING TIME FOR A GIVEN SET OF PROCESSES

# EXAMPLE - NON-PREEMPTIVE SJF

| PROCESS | ARRIVAL TIME | BURST TIME |
|---|---|---|
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |

- SJF (NON-PREEMPTIVE) – GANTT CHART



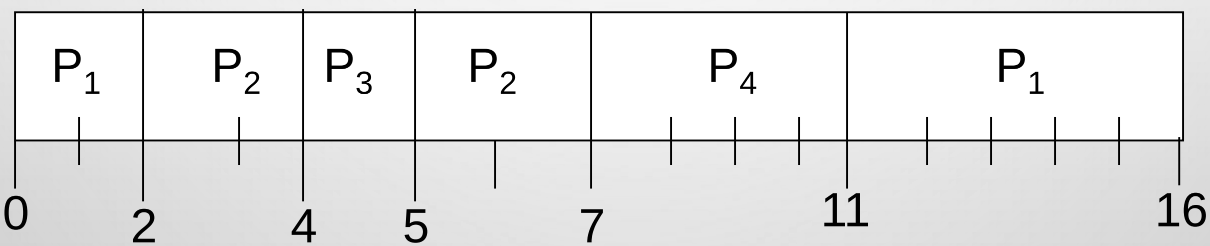

- **AVERAGE WAITING TIME**

  **= (0 + 6 + 3 + 7)/4 = 4**

# EXAMPLE OF PREEMPTIVE SJF

| PROCESS | ARRIVAL TIME | BURST TIME |
| --- | --- | --- |
| $P_1$ | 0.0 | 7 |
| $P_2$ | 2.0 | 4 |
| $P_3$ | 4.0 | 1 |
| $P_4$ | 5.0 | 4 |

- SJF (PREEMPTIVE) – GANTT CHART




- AVERAGE WAITING TIME

  = (9 + 1 + 0 +2)/4 = 3

# ROUND ROBIN (RR)

- EACH PROCESS ASSIGNED A *TIME QUANTUM*, USUALLY 10-100 MILLISECONDS
  - AFTER THIS, PROCESS MOVED TO END OF READY Q
- *N* PROCESSES IN READY QUEUE, TIME QUANTUM = Q, THEN EACH PROCESS GETS 1/*N* OF THE CPU TIME IN CHUNKS OF AT MOST Q TIME UNITS AT ONCE
  - NO PROCESS WAITS MORE THAN (*N*-1)Q TIME UNITS
- PERFORMANCE
  - Q LARGE $\Rightarrow$ FIFO
  - Q SMALL $\Rightarrow$ Q MUST BE LARGE WITH RESPECT TO CONTEXT SWITCH, OTHERWISE OVERHEAD TOO HIGH

# EXAMPLE: RR, QUANTUM = 20

| PROCESS | BURST TIME |
|---|---|
| $P_1$ | 53 |
| $P_2$ | 17 |
| $P_3$ | 68 |
| $P_4$ | 24 |

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_4$ | $P_1$ | $P_3$ | $P_3$ |
|---|---|---|---|---|---|---|---|---|---|

0 20 37 57 77 97 117 121 134 154 162

TYPICALLY, HIGHER AVERAGE TURNAROUND THAN SJF, BUT BETTER *RESPONSE*

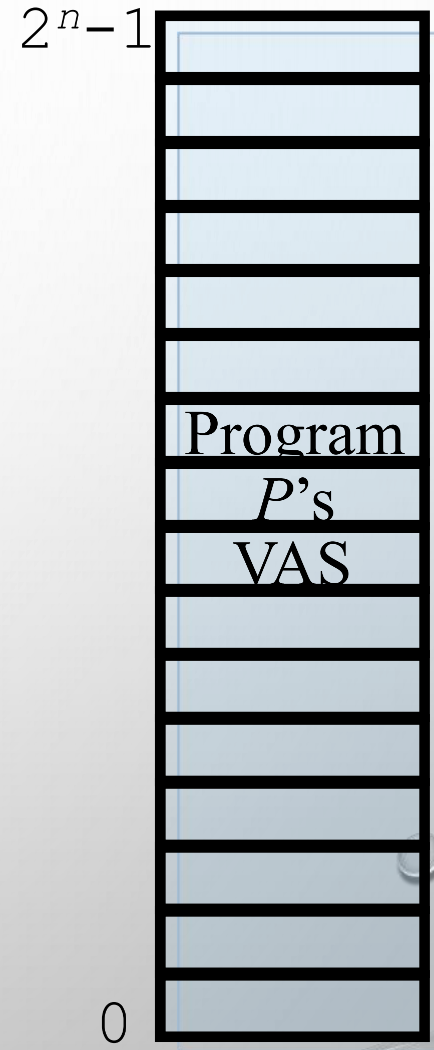
# VIRTUAL MEMORY AND ADDRESS TRANSLATION

# VIRTUAL MEMORY

## CONCEPT

- KEY PROBLEM: HOW CAN ONE SUPPORT PROGRAMS THAT REQUIRE MORE MEMORY THAN IS PHYSICALLY AVAILABLE?

- HIDE PHYSICAL SIZE OF MEMORY FROM USERS
  - MEMORY IS A "LARGE" *VIRTUAL ADDRESS SPACE* OF $2^N$ BYTES
  - ONLY PORTIONS OF VAS ARE IN PHYSICAL MEMORY AT ANY ONE TIME (INCREASE MEMORY UTILIZATION)



# REALIZING VIRTUAL MEMORY

## PAGING

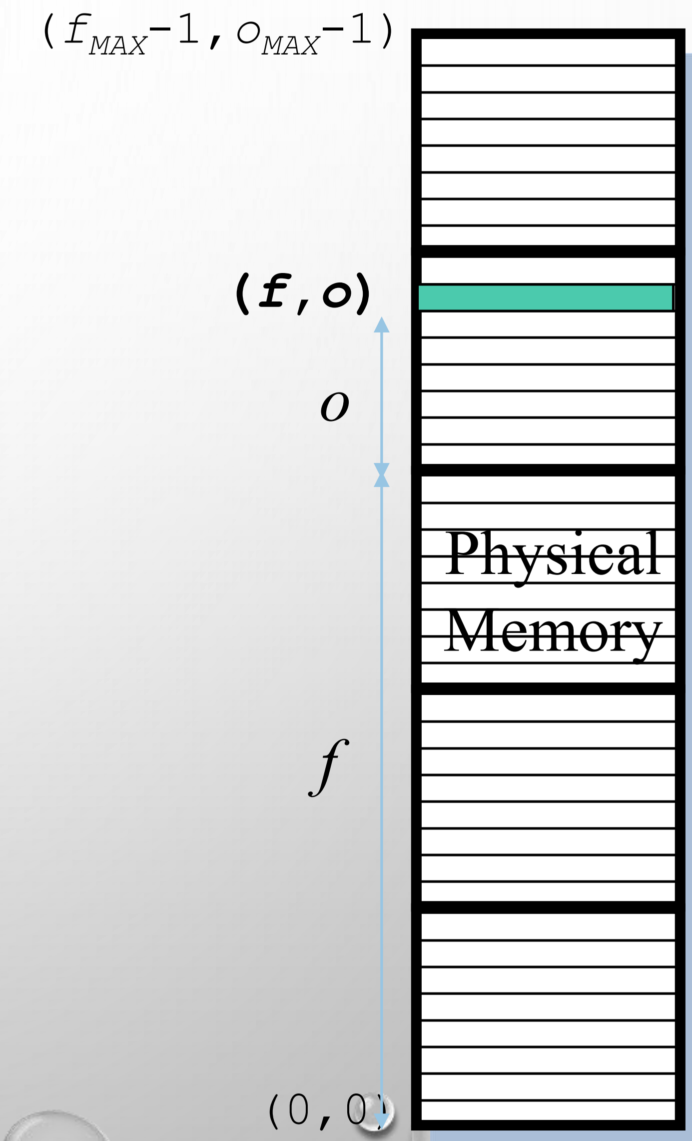
- PHYSICAL MEMORY PARTITIONED INTO EQUAL SIZED *PAGE FRAMES*
  - PAGE FRAMES AVOID EXTERNAL FRAGMENTATION

A memory address is a pair ($f$, $o$)

$f$ — frame number ($f_{max}$ frames)

$o$ — frame offset ($o_{max}$ bytes/frames)

Physical address = $o_{max} \times f + o$

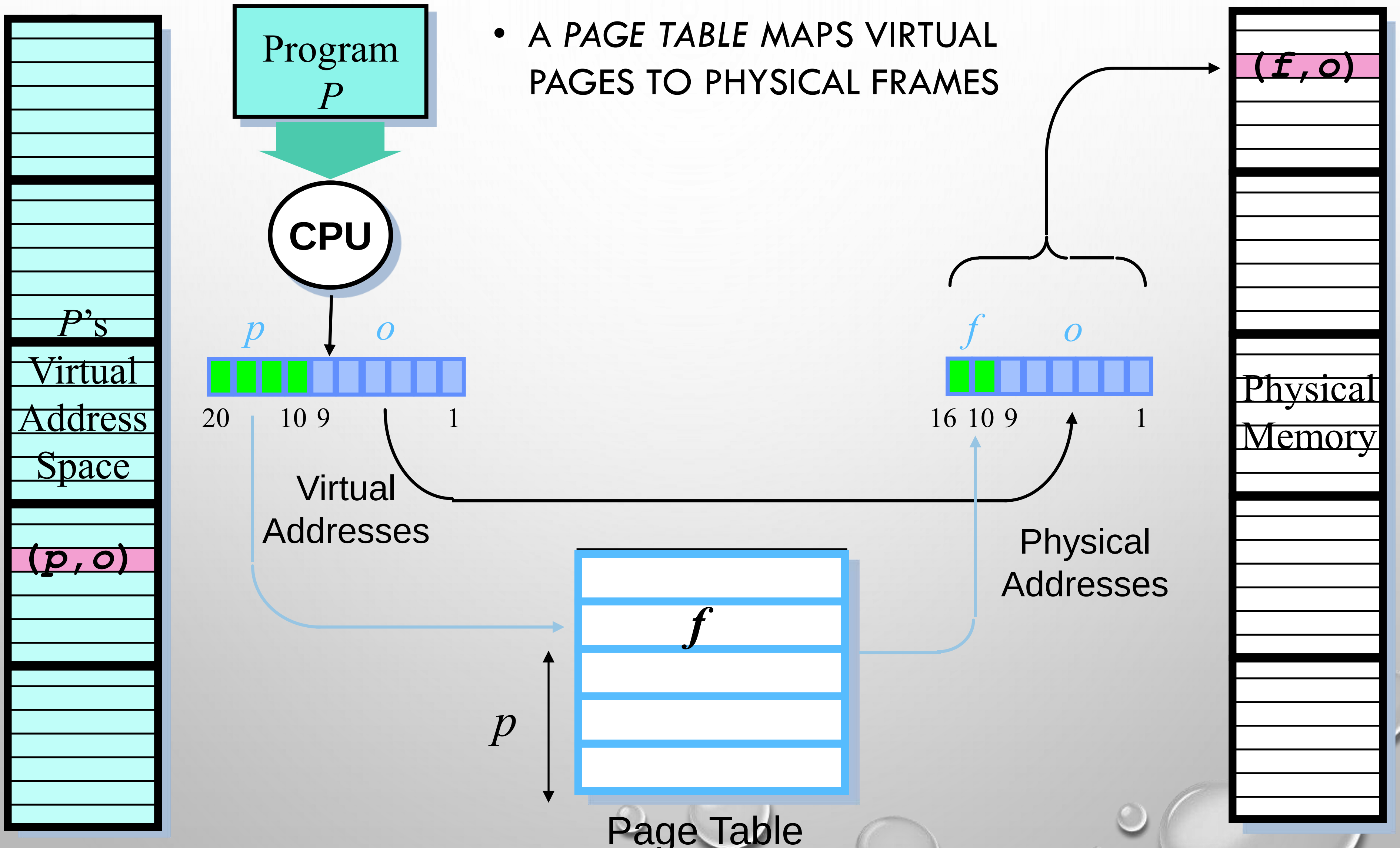



$f$ $o$

# PAGING

## VIRTUAL ADDRESS TRANSLATION

- A *PAGE TABLE* MAPS VIRTUAL PAGES TO PHYSICAL FRAMES

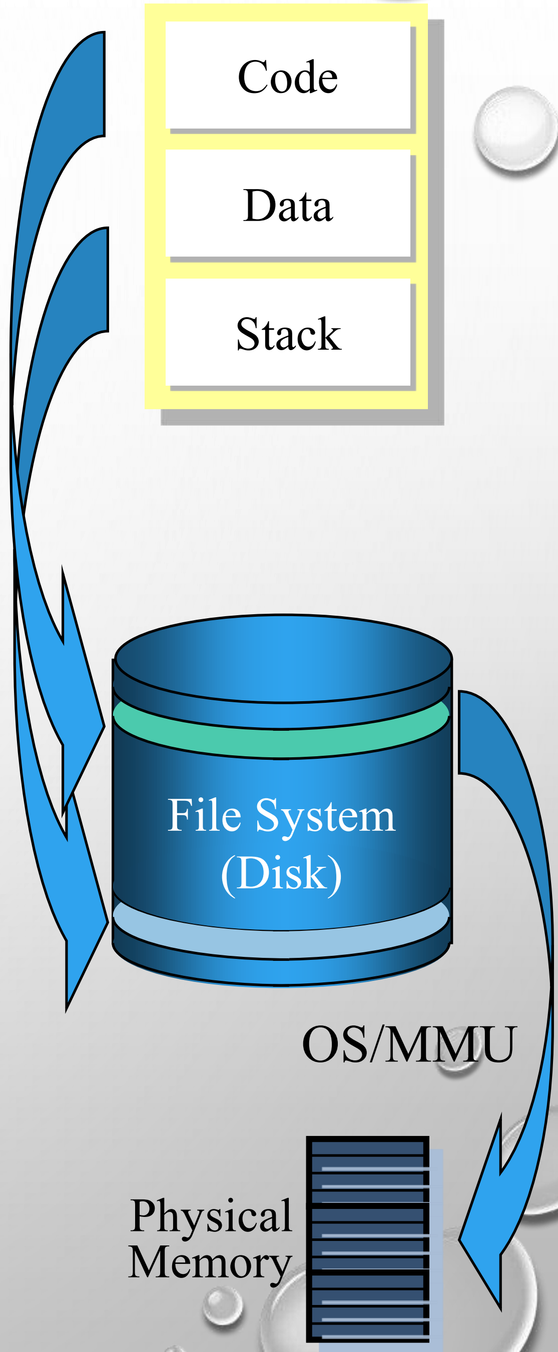


# VIRTUAL MEMORY (PAGING)

## THE BIGGER PICTURE

- A PROCESS'S VAS IS ITS CONTEXT
  - CONTAINS ITS CODE, DATA, AND STACK
- CODE PAGES ARE STORED IN A USER'S FILE ON DISK
  - SOME ARE CURRENTLY RESIDING IN MEMORY; MOST ARE NOT
- DATA AND STACK PAGES ARE ALSO STORED IN A FILE
  - ALTHOUGH THIS FILE IS TYPICALLY NOT VISIBLE TO USERS
  - FILE ONLY EXISTS WHILE A PROGRAM IS EXECUTING

OS DETERMINES WHICH PORTIONS OF A PROCESS'S VAS ARE MAPPED IN MEMORY AT ANY ONE TIME



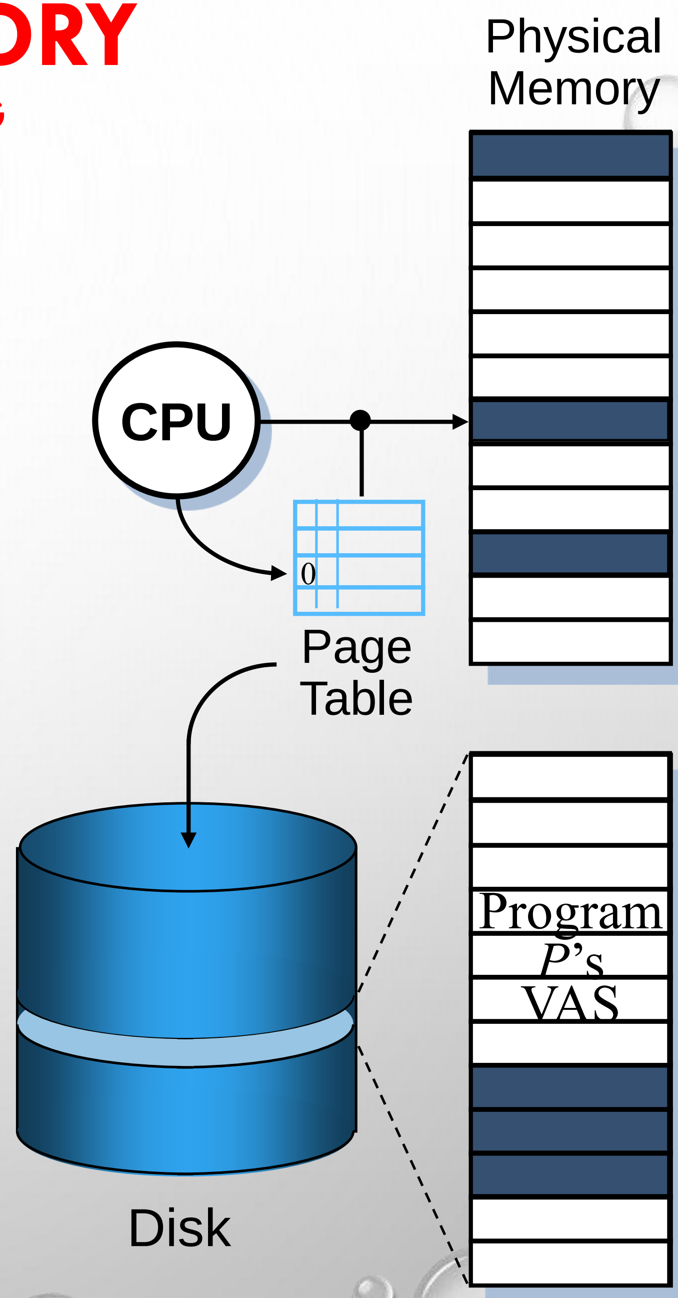
# VIRTUAL MEMORY

## PAGE FAULT HANDLING

- REFERENCES TO NON-MAPPED PAGES GENERATE A *PAGE FAULT*

Page fault handling steps:

Processor runs the interrupt handler
OS blocks the running process
OS reads in the unmapped page
OS resumes/initiates some other process
OS maps the missing page into memory
OS restart the faulting process



# *PAGE REPLACEMENT ALGORITHMS*

# PAGE REPLACEMENT ALGORITHMS

## CONCEPT

- *Local replacement* — Replace a page of the faulting process
- *Global replacement* — Possibly replace the page of another process

# OPTIMAL PAGE REPLACEMENT

- REPLACE THE PAGE THAT WON'T BE NEEDED FOR THE LONGEST TIME IN THE FUTURE

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | *c* | *a* | *d* | *b* | *e* | *b* | *a* | *b* | *c* | *d* |
| Page Frames | 0 | *a* | | | | | | | | | | |
| | 1 | *b* | | | | | | | | | | |
| | 2 | *c* | | | | | | | | | | |
| | 3 | *d* | | | | | | | | | | |
| Faults | | | | | | | | | | | | |
| Time page needed next | | | | | | | | | | | | |

# OPTIMAL PAGE REPLACEMENT

- REPLACE THE PAGE THAT WON'T BE NEEDED FOR THE LONGEST TIME IN THE FUTURE

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | *c* | *a* | *d* | *b* | *e* | *b* | *a* | *b* | *c* | *d* |
| Page Frames | 0 | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *d* |
| | 1 | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* |
| | 2 | *c* | *c* | *c* | *c* | *c* | *c* | *c* | *c* | *c* | *c* | *c* |
| | 3 | *d* | *d* | *d* | *d* | *d* | *e* | *e* | *e* | *e* | *e* | *e* |
| Faults | | | | | | | • | | | | | • |
| Time page needed next | | | | | | $a = 7$<br>$b = 6$<br>$c = 9$<br>$d = 10$ | | | | | | $a = 15$<br>$b = 11$<br>$c = 13$<br>$d = 14$ |

# LEAST RECENTLY USED PAGE REPLACEMENT

- REPLACE THE PAGE THAT HASN'T BEEN REFERENCED FOR THE LONGEST TIME

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | *c* | *a* | *d* | *b* | *e* | *b* | *a* | *b* | *c* | *d* |
| Page Frames | 0 | *a* | | | | | | | | | | |
| | 1 | *b* | | | | | | | | | | |
| | 2 | *c* | | | | | | | | | | |
| | 3 | *d* | | | | | | | | | | |
| Faults | | | | | | | | | | | | |
| Time page last used | | | | | | | | | | | | |

# LEAST RECENTLY USED PAGE REPLACEMENT

- REPLACE THE PAGE THAT HASN'T BEEN REFERENCED FOR THE LONGEST TIME

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Requests | | | *c* | *a* | *d* | *b* | *e* | *b* | *a* | *b* | *c* | *d* |
| Page Frames | 0 | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* |
| | 1 | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* |
| | 2 | *c* | *c* | *c* | *c* | *c* | ***e*** | *e* | *e* | *e* | *e* | ***d*** |
| | 3 | *d* | *d* | *d* | *d* | *d* | *d* | *d* | *d* | *d* | ***c*** | *c* |
| Faults | | | | | | | • | | | | • | • |
| Time page last used | | | | | | $a = 2$<br>$b = 4$<br>$c = 1$<br>$d = 3$ | | | | $a = 7$<br>$b = 8$<br>$e = 5$<br>$d = 3$ | $a = 7$<br>$b = 8$<br>$e = 5$<br>$c = 9$ | |

# LEAST RECENTLY USED PAGE REPLACEMENT

## IMPLEMENTATION

- MAINTAIN A "STACK" OF RECENTLY USED PAGES

| | Time | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | *c* | *a* | *d* | *b* | *e* | *b* | *a* | *b* | *c* | *d* |
| Page Frames | 0 | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* | *a* |
| | 1 | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* | *b* |
| | 2 | *c* | *c* | *c* | *c* | *c* | ***e*** | *e* | *e* | *e* | *e* | ***d*** |
| | 3 | *d* | *d* | *d* | *d* | *d* | *d* | *d* | *d* | *d* | ***c*** | *c* |
| Faults | | | | | | | • | | | | • | • |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| LRU page stack | | | | | | | | | | |
| Page to replace | | | | | | | | | | |

# LEAST RECENTLY USED PAGE REPLACEMENT

## IMPLEMENTATION

- MAINTAIN A "STACK" OF RECENTLY USED PAGES

| Time | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Requests | | | c | a | d | b | e | b | a | b | c | d |
| Page Frames | 0 | a | a | a | a | a | a | a | a | a | a | a |
| | 1 | b | b | b | b | b | b | b | b | b | b | b |
| | 2 | c | c | c | c | c | **e** | e | e | e | e | **d** |
| | 3 | d | d | d | d | d | d | d | d | d | **c** | c |
| Faults | | | | | | | • | | | | • | • |

| LRU page stack | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | c | a | d | b | e | b | a | b | c | d |
| | | c | a | d | b | e | b | a | b | c |
| | | | c | a | d | d | e | e | a | b |
| | | | | c | a | a | d | d | e | a |
| Page to replace | | | | | c | | | | d | e |

# PAGE REPLACEMENT ALGORITHMS

## PERFORMANCE

- LOCAL PAGE REPLACEMENT
  - LRU — AGES PAGES BASED ON WHEN THEY WERE LAST USED
  - FIFO — AGES PAGES BASED ON WHEN THEY'RE BROUGHT INTO MEMORY

- TOWARDS GLOBAL PAGE REPLACEMENT … WITH VARIABLE NUMBER OF PAGE FRAMES ALLOCATED TO PROCESSES

**The principle of locality**

- 90% of the execution of a program is sequential
- Most iterative constructs consist of a relatively small number of instructions
- When processing large data structures, the dominant cost is sequential processing on individual structure elements
- Temporal vs. physical locality

# EXPLICITLY USING LOCALITY

## THE WORKING SET MODEL OF PAGE REPLACEMENT

- ASSUME RECENTLY REFERENCED PAGES ARE LIKELY TO BE REFERENCED AGAIN SOON…
- … AND *ONLY* KEEP THOSE PAGES RECENTLY REFERENCED IN MEMORY (CALLED *THE WORKING SET*)
  - THUS PAGES MAY BE REMOVED EVEN WHEN NO PAGE FAULT OCCURS
  - THE NUMBER OF FRAMES ALLOCATED TO A PROCESS WILL VARY OVER TIME
- A PROCESS IS ALLOWED TO EXECUTE ONLY IF ITS WORKING SET FITS INTO MEMORY
  - THE WORKING SET MODEL PERFORMS IMPLICIT LOAD CONTROL

# WORKING SET PAGE REPLACEMENT
## IMPLEMENTATION

- KEEP TRACK OF THE LAST $\tau$ REFERENCES
  - THE PAGES REFERENCED DURING THE LAST $\tau$ MEMORY ACCESSES ARE THE WORKING SET
  - $\tau$ IS CALLED THE *WINDOW SIZE*

# PAGE-FAULT-FREQUENCY PAGE REPLACEMENT

## AN ALTERNATE WORKING SET COMPUTATION

- EXPLICITLY ATTEMPT TO MINIMIZE PAGE FAULTS
  - WHEN PAGE FAULT FREQUENCY IS HIGH — *INCREASE WORKING SET*
  - WHEN PAGE FAULT FREQUENCY IS LOW — *DECREASE WORKING SET*

Algorithm:

Keep track of the rate at which faults occur

When a fault occurs, compute the time since the last page fault

Record the time, $t_{last}$, of the last page fault

If the time between page faults is "large" then reduce the working set

If $t_{current} - t_{last} > \tau$, then remove from memory all pages not referenced in $[t_{last}, t_{current}]$

If the time between page faults is "small" then increase working set

If $t_{current} - t_{last} \leq \tau$, then add faulting page to the working set

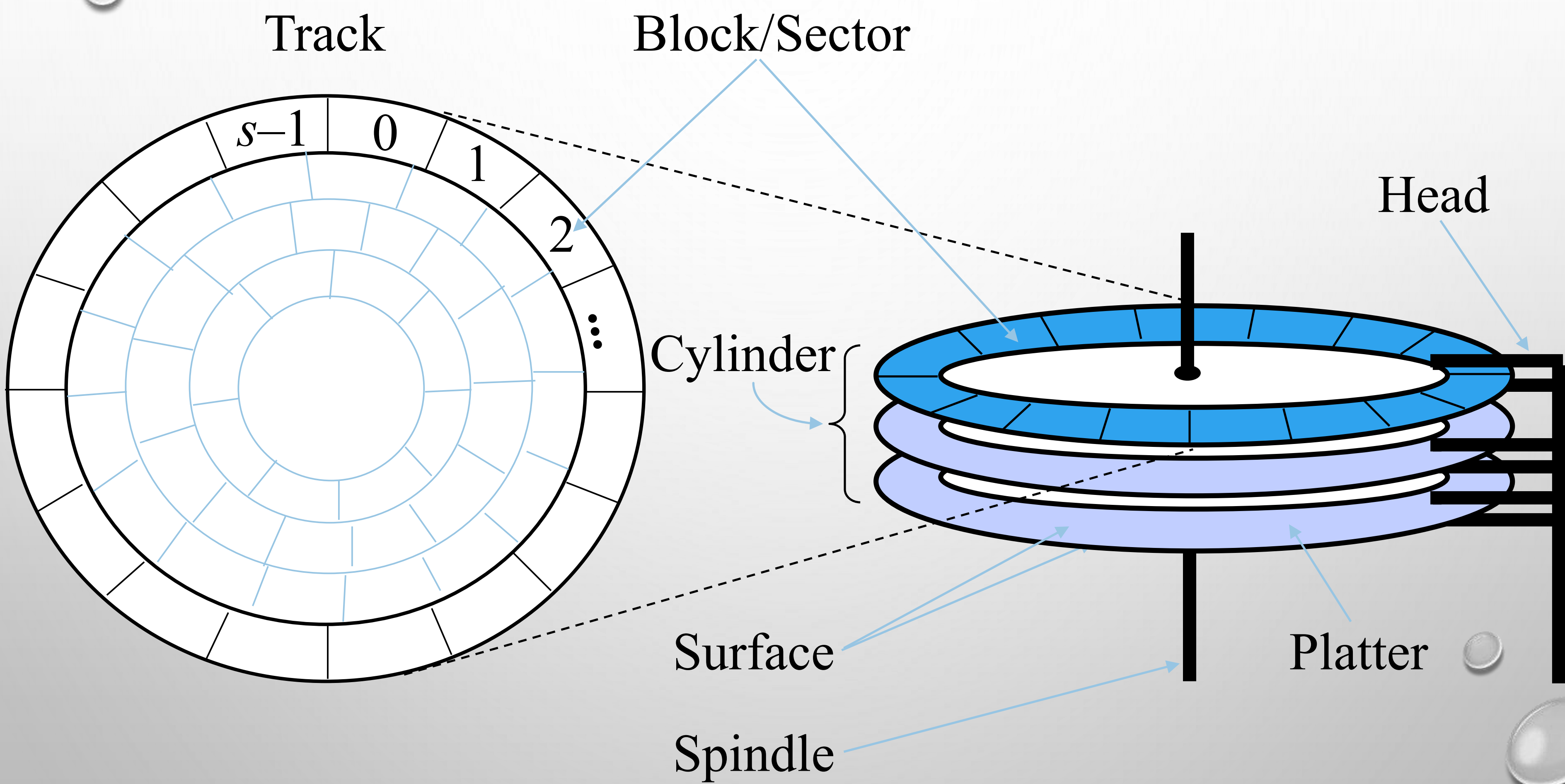
# DISK MANAGEMENT

# DISK TYPES

- ADVANCED TECHNOLOGY ATTACHMENT (ATA)
    - ALSO CALLED IDE (INTEGRATED DRIVE ELECTRONICS), ATAPI, AND UDMA
- SMALL COMPUTER SYSTEM INTERFACE (SCSI)
- MICRODRIVE
- SATA (SERIAL ATA)
- SSD (SOLID STATE DEVICES) – NOT TRADITIONAL DISKS?

# ANATOMY OF A DISK

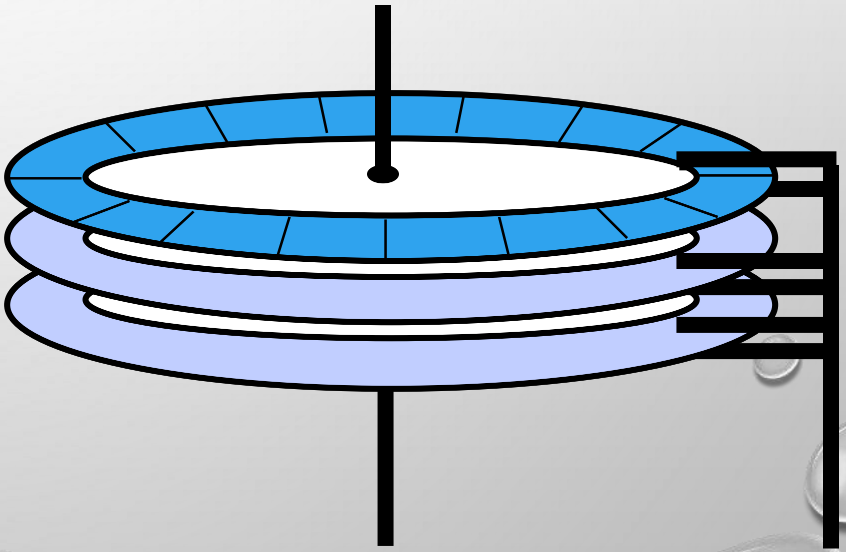
## BASIC COMPONENTS



# EXAMPLE

## SEAGATE 73.4 GB FIBRE CHANNEL ULTRA 160 SCSI DISK

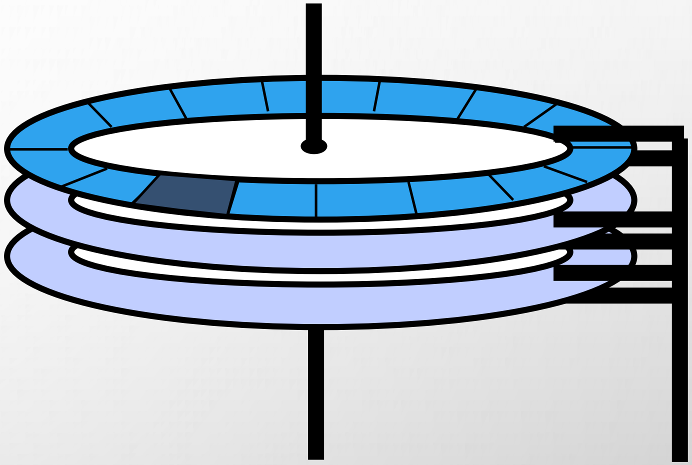
- SPECS:
  - 12 PLATTERS
  - 12 ARMS
  - 24 HEADS
  - 14,100 TRACKS
  - VARIABLE # OF SECTORS/TRACK
  - 512 BYTES/SECTOR



# DISK OPERATIONS

## READ/WRITE OPERATIONS

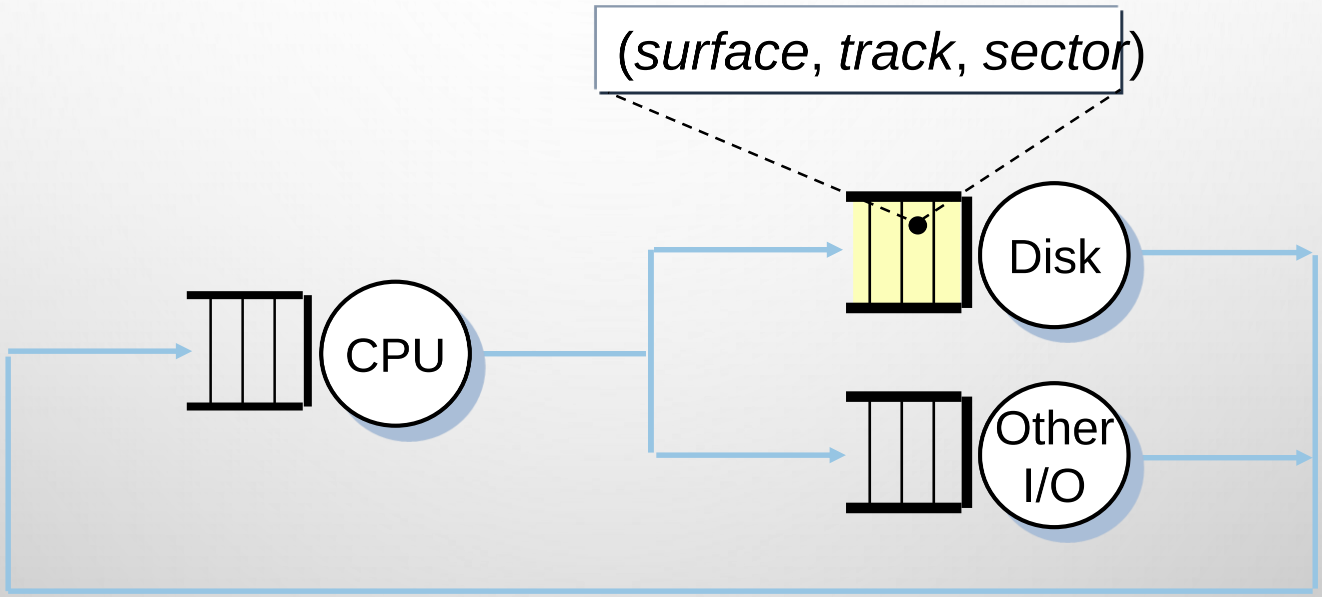
- PRESENT DISK WITH A SECTOR ADDRESS
  - *DA = (DRIVE, SURFACE, TRACK, SECTOR)*
- HEADS MOVED TO APPROPRIATE TRACK
  - SEEK TIME
  - SETTLE TIME
- THE APPROPRIATE HEAD IS ENABLED
- WAIT FOR THE SECTOR TO APPEAR UNDER THE HEAD
  - "ROTATIONAL LATENCY"
- READ/WRITE THE SECTOR
  - "TRANSFER TIME"



# DISK HEAD SCHEDULING

## MAXIMIZING DISK THROUGHPUT

- IN A MULTIPROGRAMMING/TIMESHARING ENVIRONMENT, A QUEUE OF DISK I/O REQUESTS CAN FORM



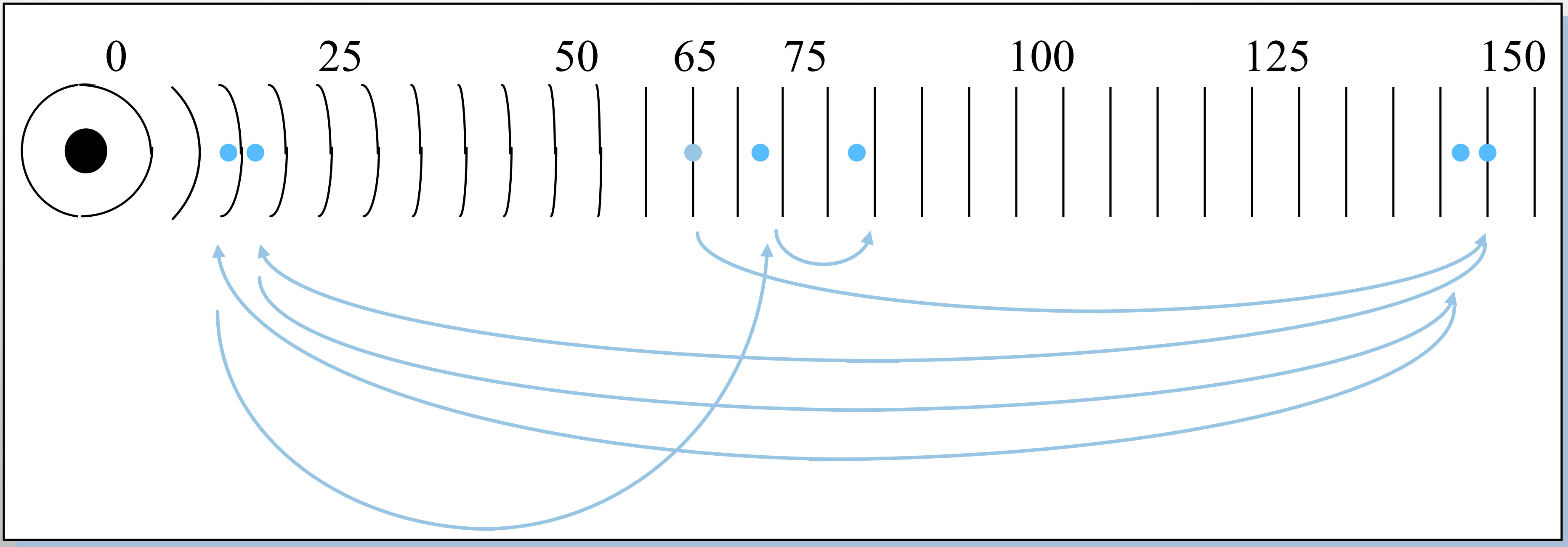

The OS maximizes disk I/O throughput by minimizing head movement through *disk head scheduling*

# FCFS

- ASSUME A QUEUE OF REQUESTS EXISTS TO READ/WRITE TRACKS:

| | | | | | |
|---|---|---|---|---|---|
| 83 | 72 | 14 | 147 | 16 | 150 |

  - HEAD IS ON TRACK 65




*FCFS* scheduling results in the head moving 550 tracks

Can we do better?

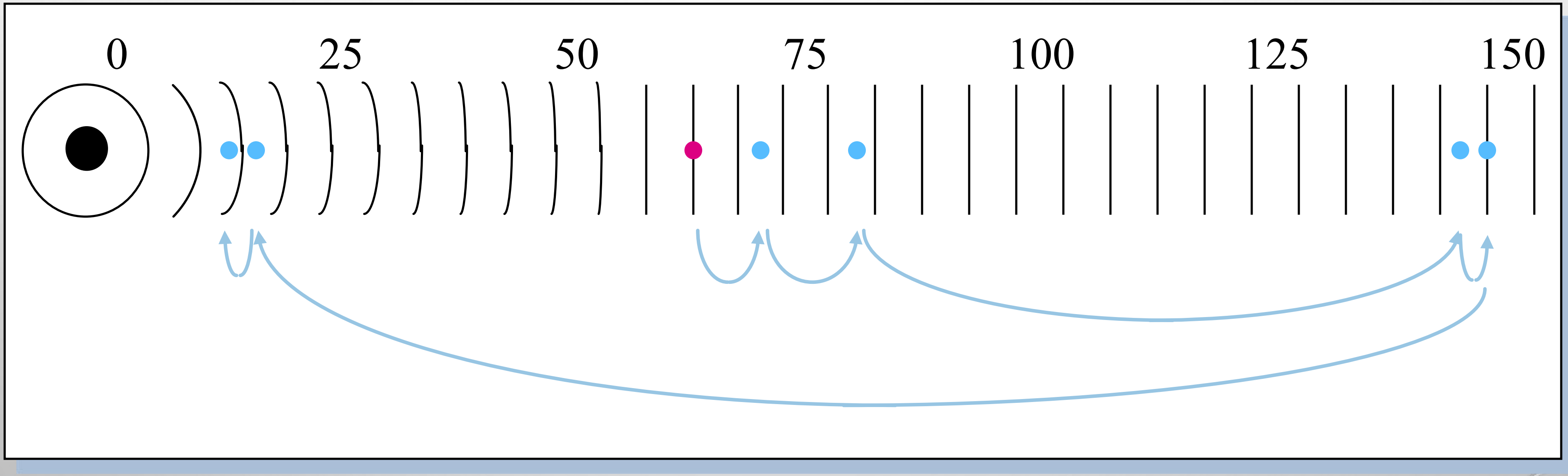
# SHORTEST SEEK TIME FIRST (SSTF)

- REARRANGE QUEUE FROM:

| | | | | | |
|---|---|---|---|---|---|
| 83 | 72 | 14 | 147 | 16 | 150 |

TO:

| | | | | | |
|---|---|---|---|---|---|
| 14 | 16 | 150 | 147 | 83 | 72 |




*SSTF* scheduling results in the head moving 221 tracks

Can we do better?

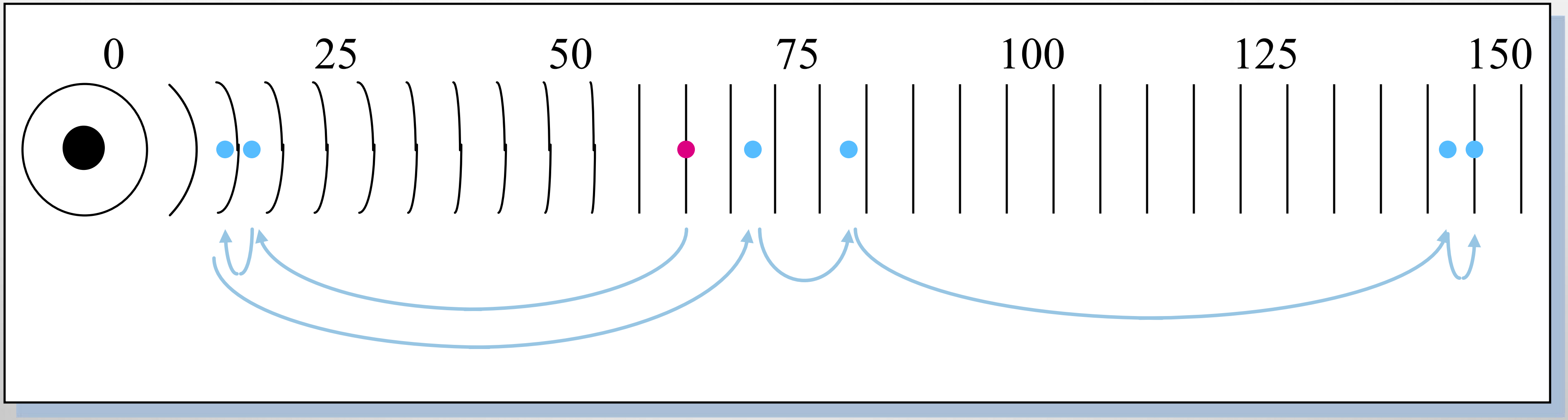
# SCAN SCHEDULING

- REARRANGE QUEUE FROM:

| | | | | | |
|---|---|---|---|---|---|
| 83 | 72 | 14 | 147 | 16 | 150 |

TO:

| | | | | | |
|---|---|---|---|---|---|
| 150 | 147 | 83 | 72 | 14 | 16 |




"*SCAN*" (elevator) scheduling: Move the head in one direction until all requests have been serviced and then reverse

Moves the head 187 tracks

# NOTE ON ASSIGNMENT

- PROCESS SCHEDULING
- RAGE REPLACEMENT – LOCAL AND/OR GLOBAL
- DISK SCHEDULING

- WILL BE UPLOADED SOON
- QUESTIONS?