Introduction to Operating Systems
CS 1550

Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Announcements

- Upcoming deadlines
  - Homework 7 is due **this Friday**
  - Quiz 1 and Lab 2 due on Tuesday 2/28 at 11:59 pm
  - Project 2 is due Friday 3/17 at 11:59 pm
- Talk by candidate faculty
  - This Wednesday 3/15 @ 10 am at 5317 Sennott Square
  - Donuts will be served!

Last Lecture …

- How to implement Condition Variables and Locks using semaphores
- CPU scheduling
  - SJF
  - Priority
  - RR

Today …

- CPU scheduling
  - Multi-Level Feedback Queues
  - CPU burst estimation
- Memory

Multilevel Feedback Scheduling

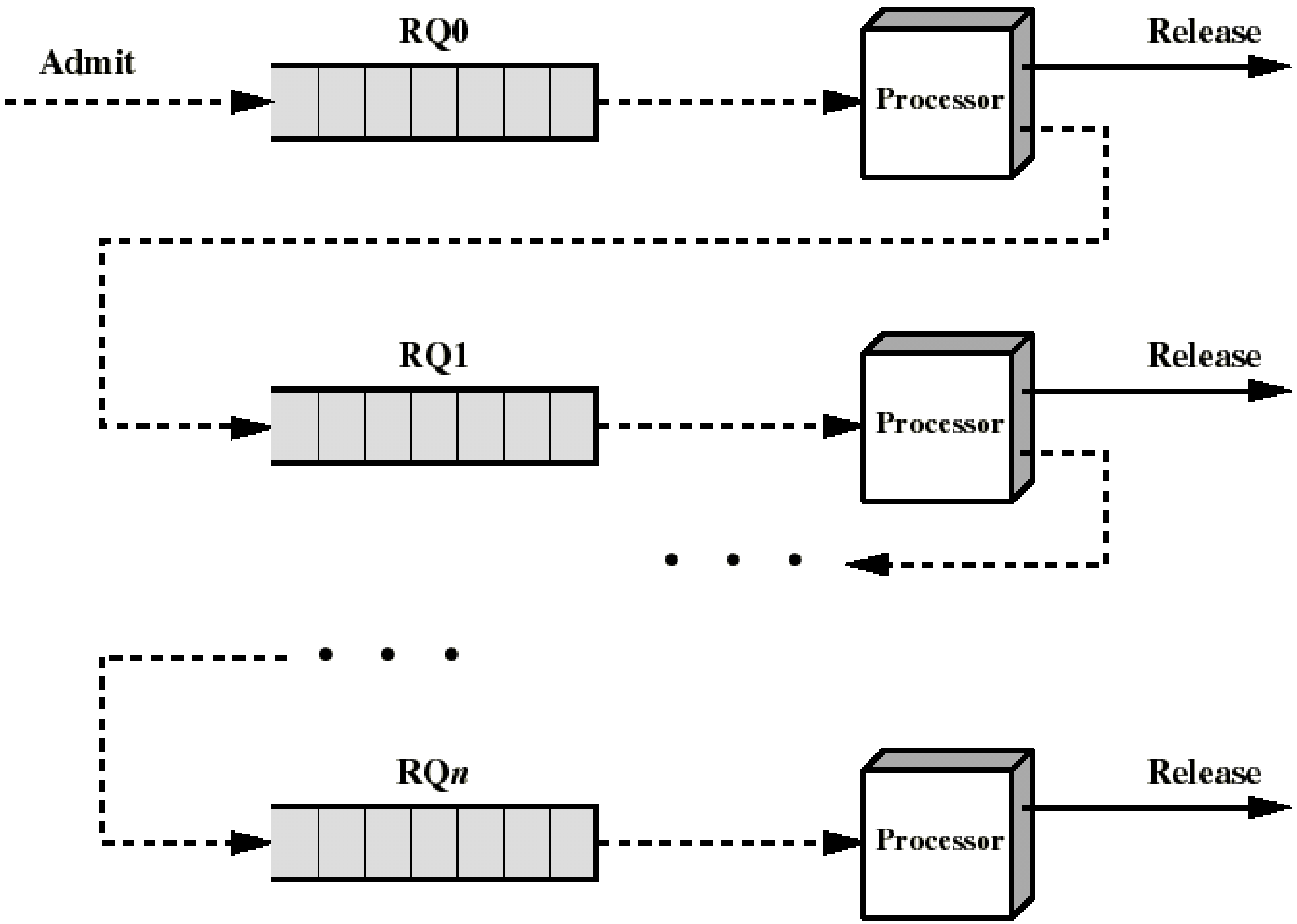
- Preemptive scheduling with dynamic priorities
- N ready to execute queues with decreasing priorities:
  - $P(RQ_0) > P(RQ_1) > ... > P(RQ_{N-1})$
- Dispatcher selects a process for execution from RQ_i only if RQ_{i-1} to RQ_0 are empty

Multilevel Feedback Scheduling

- New process are placed in RQ_0
- After the first quantum, they are moved to RQ_1, and to RQ_2 after the second quantum, … and to RQ_{N-1} after the Nth quantum
- I/O-bound processes remain in higher priority queues.
  - CPU-bound jobs drift downward.
  - Hence, long jobs may starve

Multiple Feedback Queues

Different RQs may have different quantum values



Time Quantum for feedback Scheduling

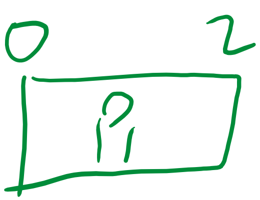
- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

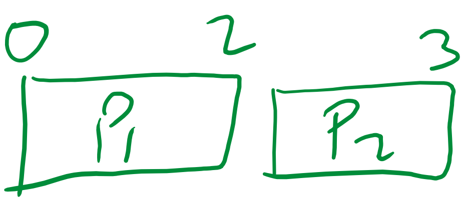
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

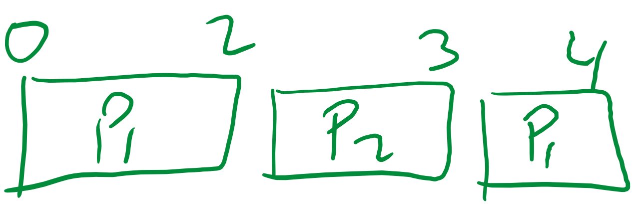
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



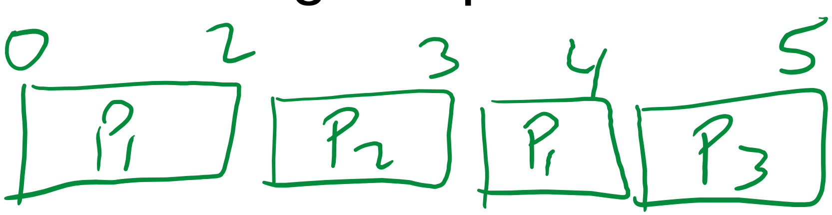
Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high

- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$

- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

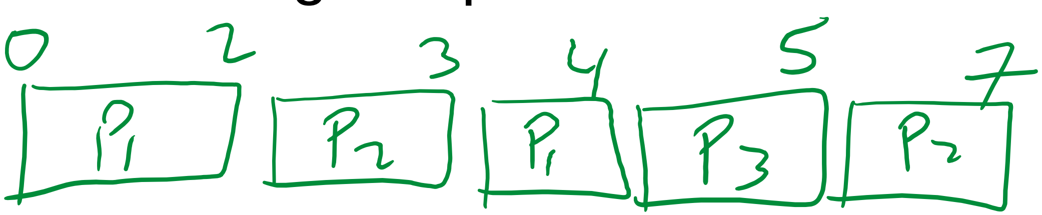
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

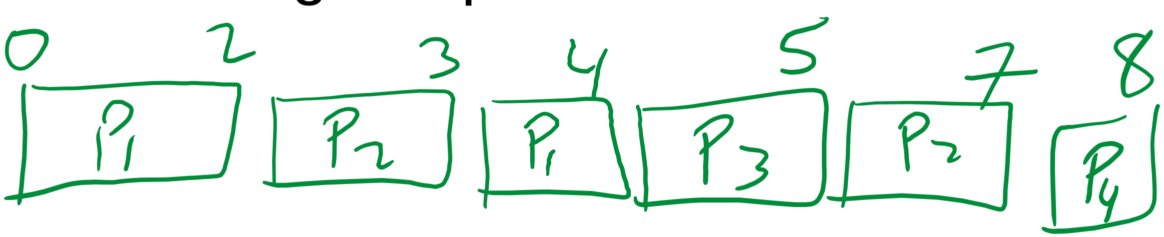
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

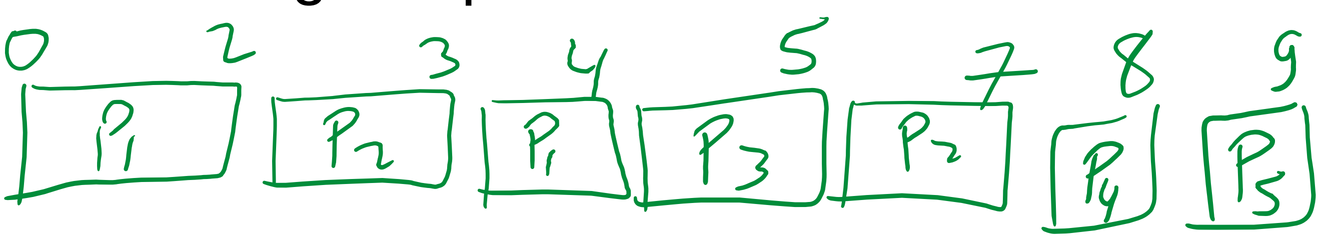
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

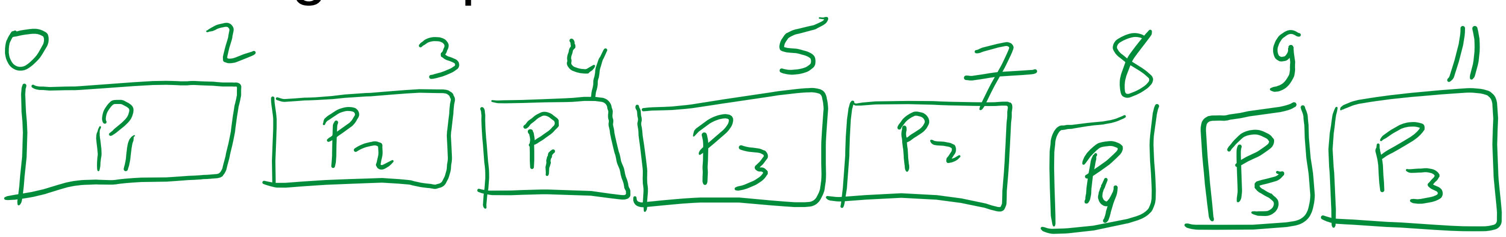
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

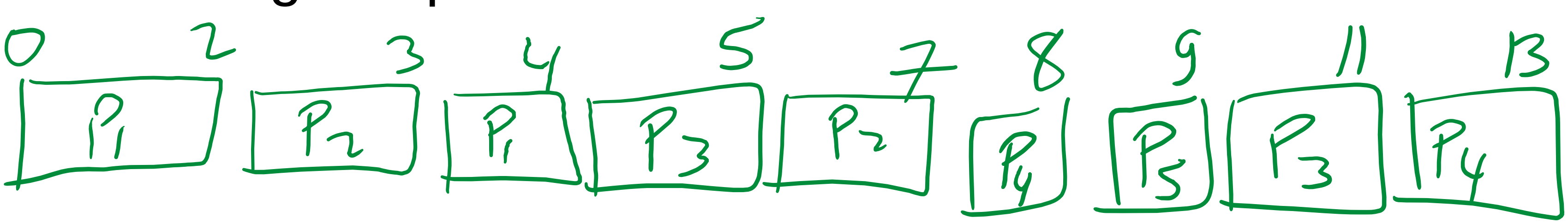
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

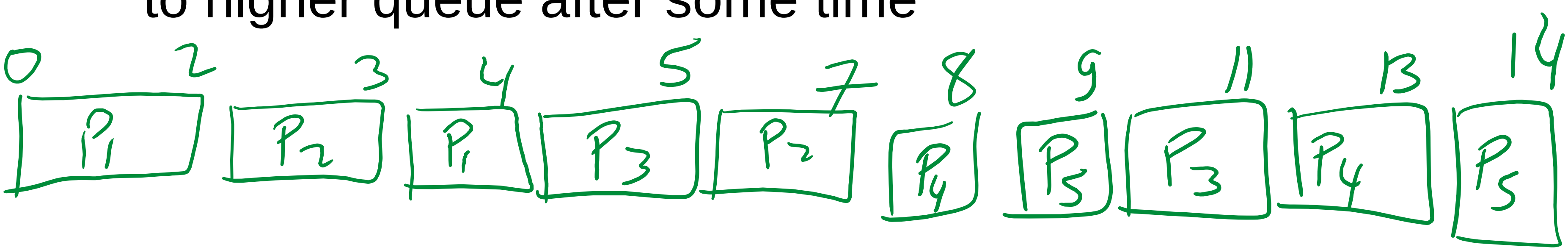
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

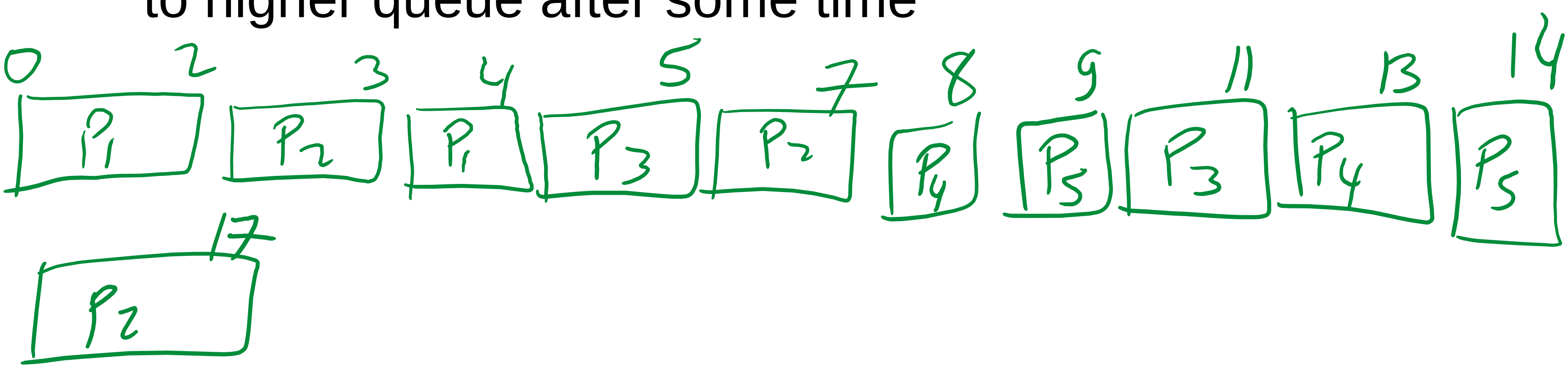
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



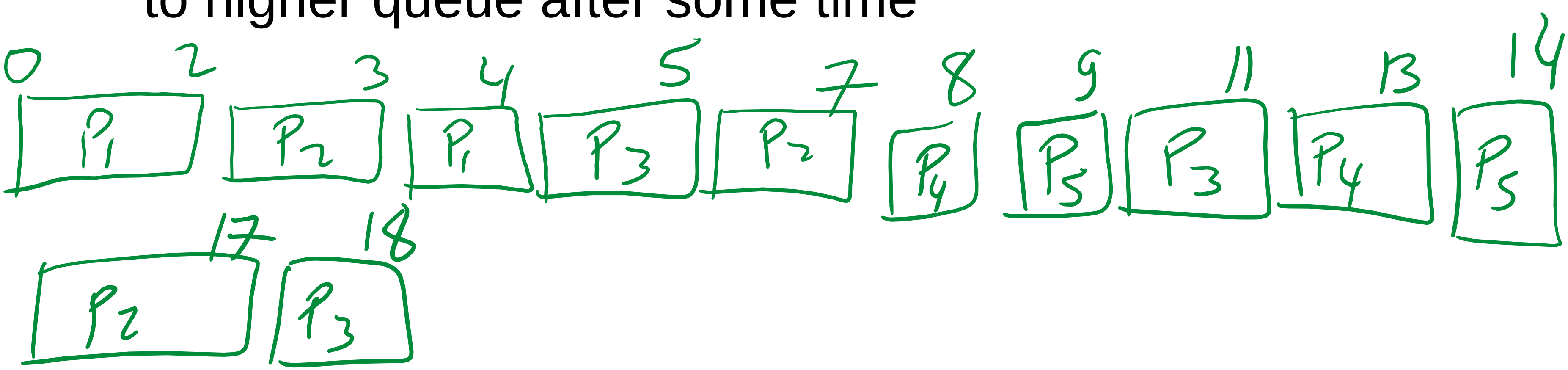
Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high

- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$

- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

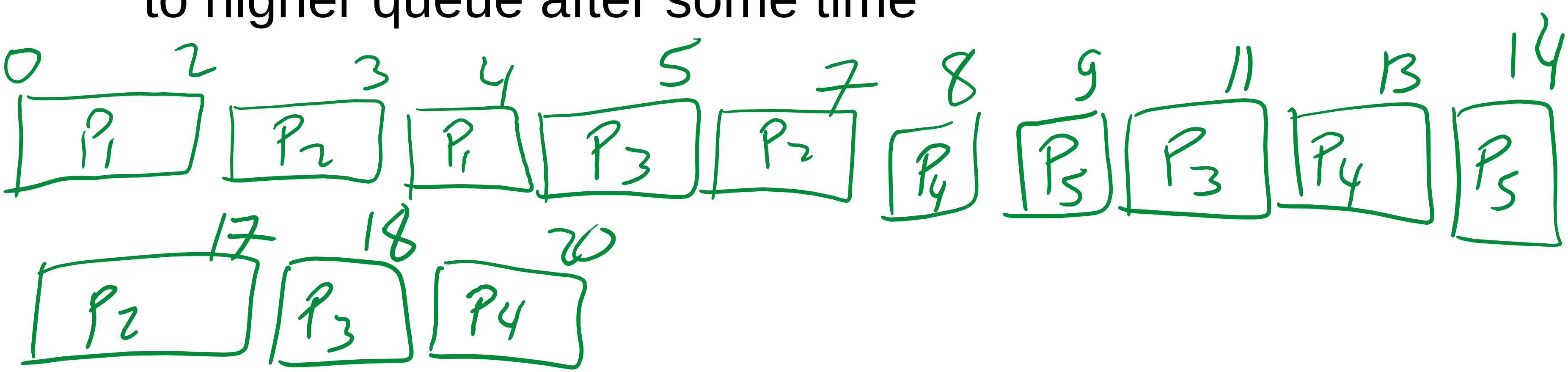
Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Time Quantum for feedback Scheduling

- With a fixed quantum time, the turn-around time of longer processes can be high
- To alleviate this problem, the time quantum can be increased based on the depth of the queue
  - Time quantum of $RQ_i = 2^{i-1}$
- May still cause longer processes to suffer starvation.
  - Possible fix is to promote a process to higher queue after some time

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Algorithm Comparison

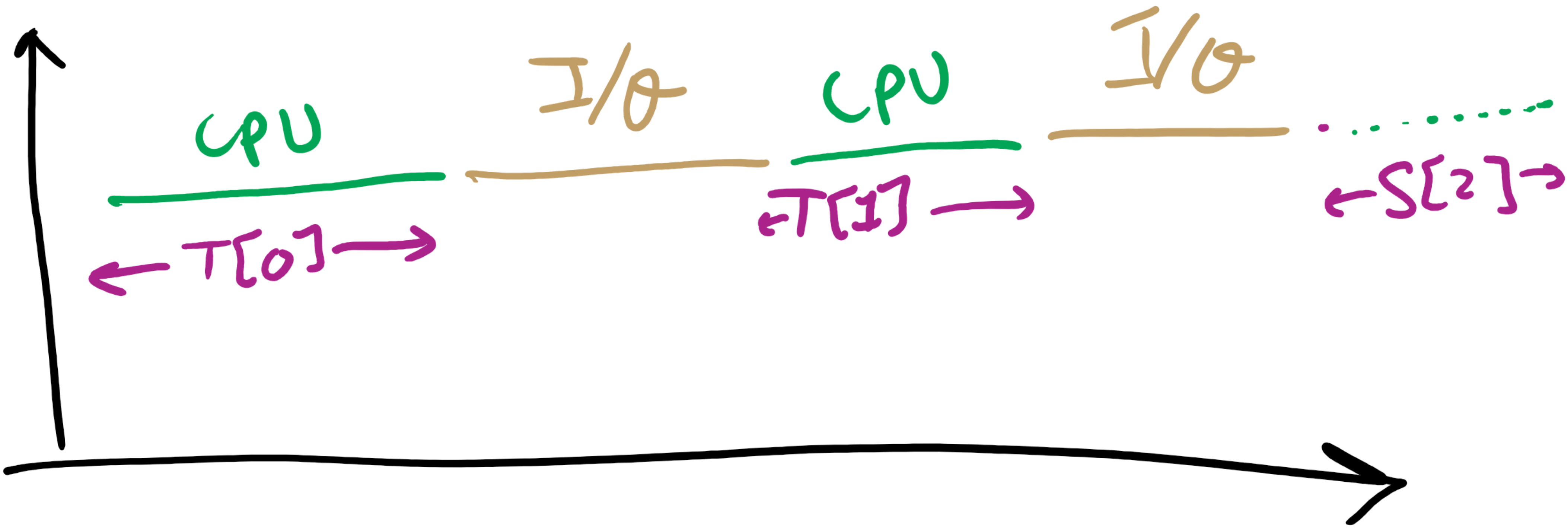
- Which one is the best?
- The answer depends on many factors:
  - the system workload (extremely variable)
  - hardware support for the dispatcher
  - relative importance of performance criteria (response time, CPU utilization, throughput...)
  - The evaluation method used (each has its limitations...)

Back to SJF: CPU Burst Estimation

- Let T[i] be the execution time for the ith instance of this process: the actual duration of the ith CPU burst of this process

- Let S[i] be the predicted value for the ith CPU burst of this process. The simplest choice is:
  - $S[n+1] = (1/n)(T[1]+\ldots+T[n]) = (1/n) \sum_{\{i=1 \text{ to } n\}} T[i]$
- This can be more efficiently calculated as:
  - $S[n+1] = (1/n)\ T[n] + ((n-1)/n)\ S[n]$
- This estimate, however, results in equal weight for each instance

CPU Burst Estimation



Estimating the required CPU burst

- Recent instances are more likely to better reflect future behavior

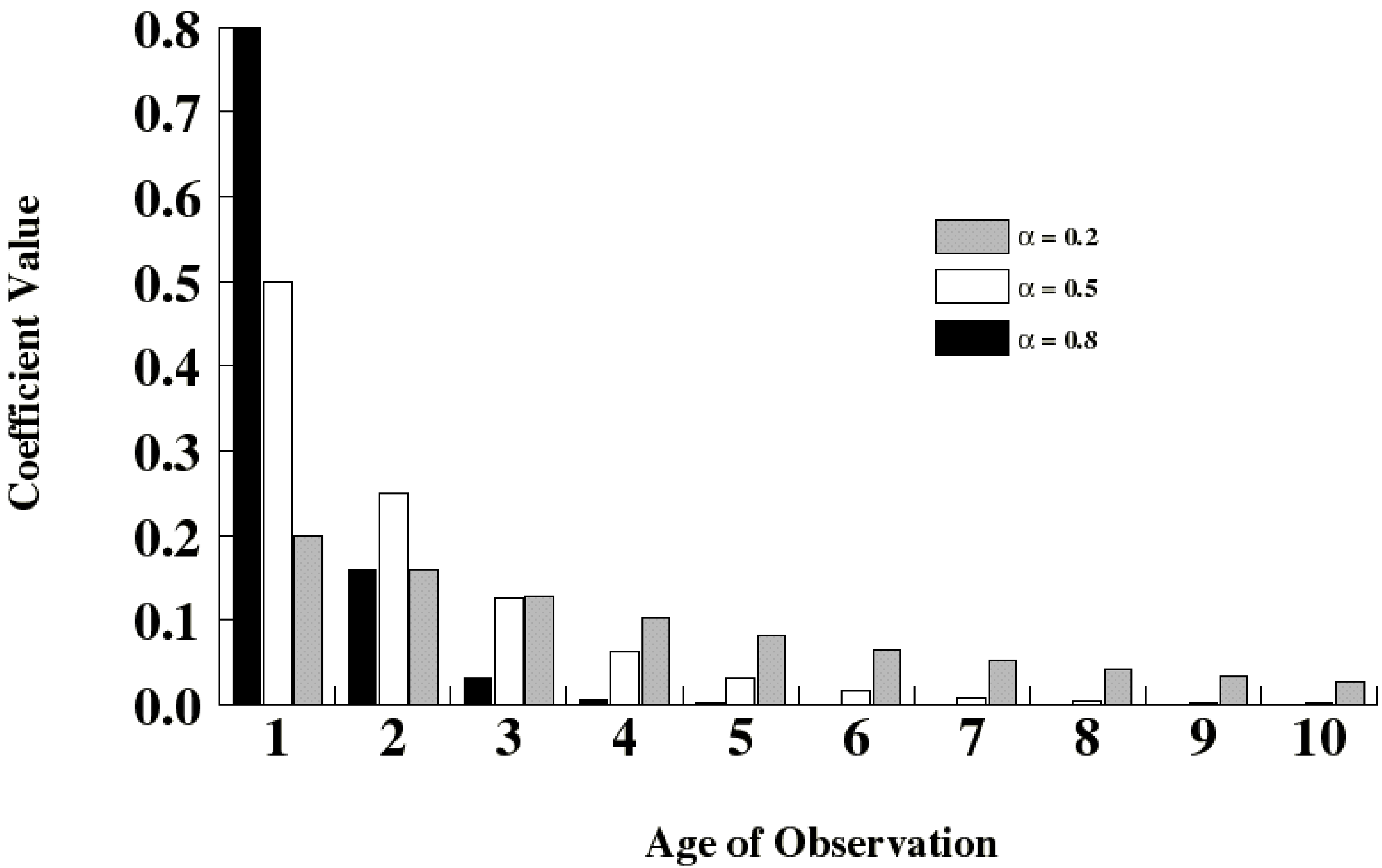
- A common technique to factor the above observation into the estimate is to use exponential averaging :

  - $S[n+1] = \alpha\, T[n] + (1- \alpha)\, S[n] \;;\quad 0 < \alpha < 1$

CPU burst Estimate (Exponential Average)

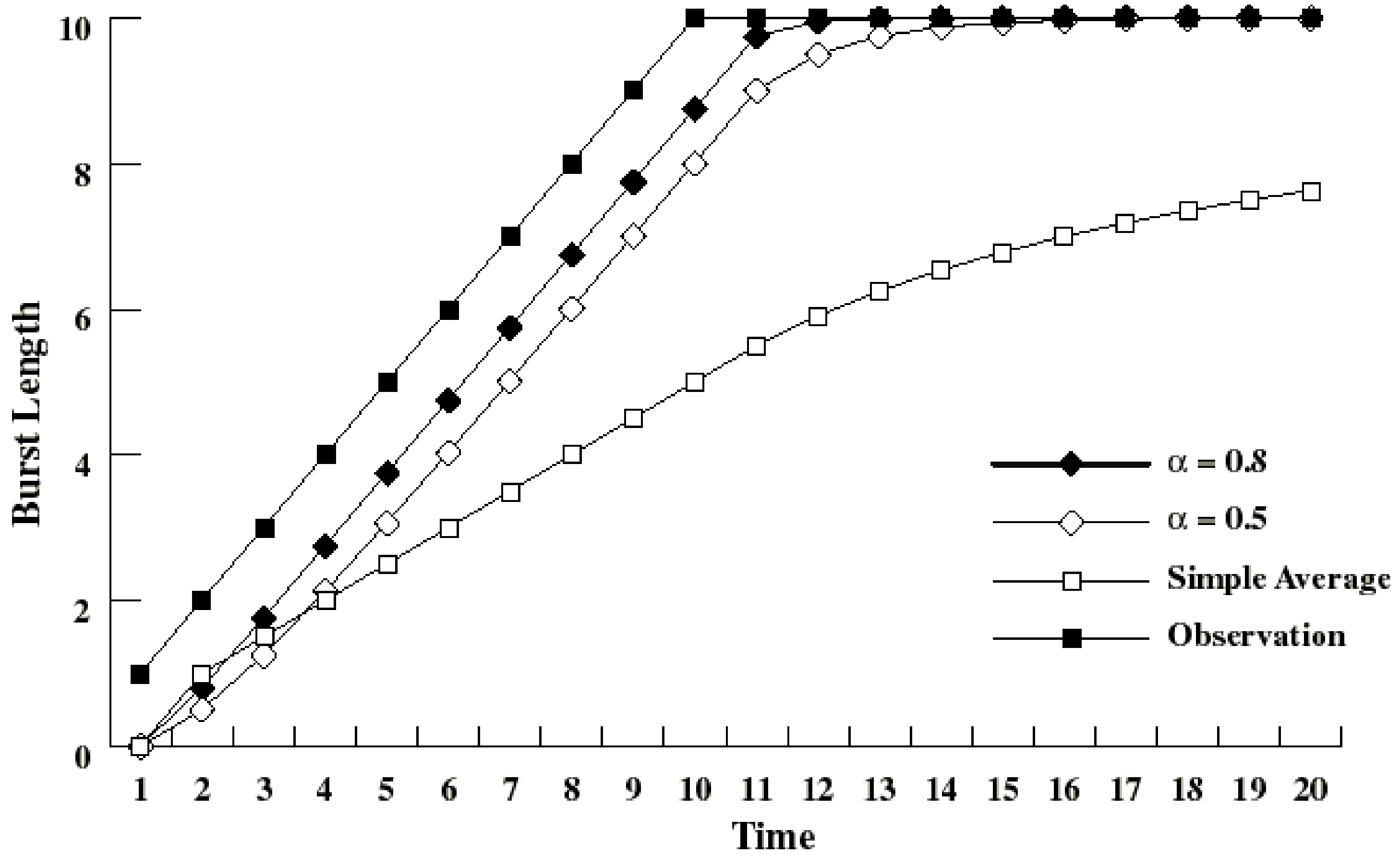
- Recent instances have higher weights, whenever $\alpha > 1/n$
- Expanding the estimated value shows that the weights of past instances decrease exponentially
  - $S[n+1] = \alpha T[n] + (1-\alpha)\alpha T[n-1] + \ldots (1-\alpha)^{i} \alpha T[n-i] +$
  - $\ldots + (1-\alpha)^{n} S[1]$
  - The predicted value of 1st instance, S[1], is usually set to 0 to give priority to new processes

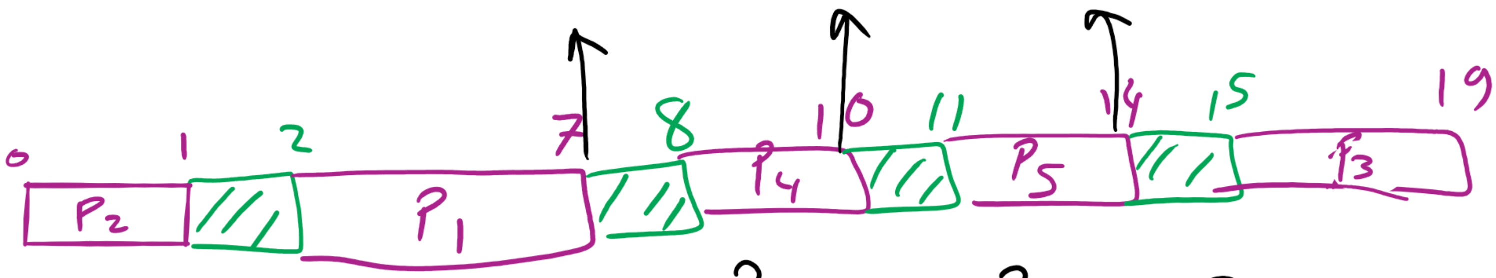
Exponentially Decreasing Coefficients



Exponentially Decreasing Coefficients

- S[1] = 0 to give high priority to new processes
- Exponential averaging tracks changes in process behavior much faster than simple averaging





$$A.R.T. = \frac{\overset{P_1}{(7-2)} + \overset{P_2}{(1-0)} + \overset{P_3}{(19-8)} + \overset{P_4}{(10-4)} + \overset{P_5}{(14-7)}}{5}$$

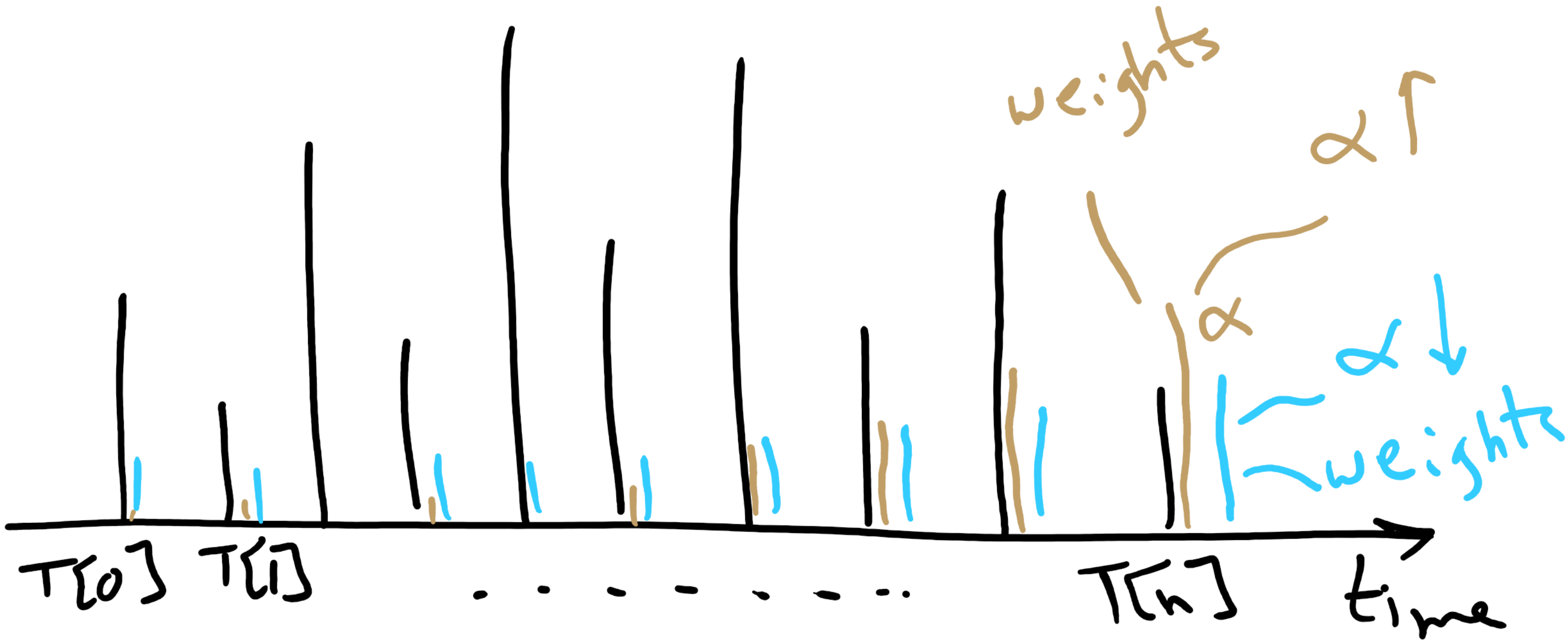
$$= \frac{5+1+11+6+7}{5} = 6\text{ms}$$

$$\text{CPU util} = \frac{19-4}{19} = \frac{15}{19}$$

CPU Burst Estimation

$$S[n+1] = \alpha\, T[n] + (1-\alpha)\, S[n]$$

$$0 < \alpha < 1$$



In an ideal world…

- The ideal world has memory that is
  - Very large
  - Very fast
  - Non-volatile (doesn't go away when power is turned off)
- The real world has memory that is:
  - Very large
  - Very fast
  - Affordable!

  ⇒Pick any two…
- Memory management goal: make the real world look as much like the ideal world as possible

Memory hierarchy

- What is the memory hierarchy?
  - Different levels of memory
  - Some are small & fast
  - Others are large & slow
- What levels are usually included?
  - Cache: small amount of fast, expensive memory
    - L1 (level 1) cache: usually on the CPU chip
    - L2 & L3 cache: off-chip, made of SRAM
  - Main memory: medium-speed, medium price memory (DRAM)
  - Disk: many gigabytes of slow, cheap, non-volatile storage
- Memory manager handles the memory hierarchy

Problem of the Day

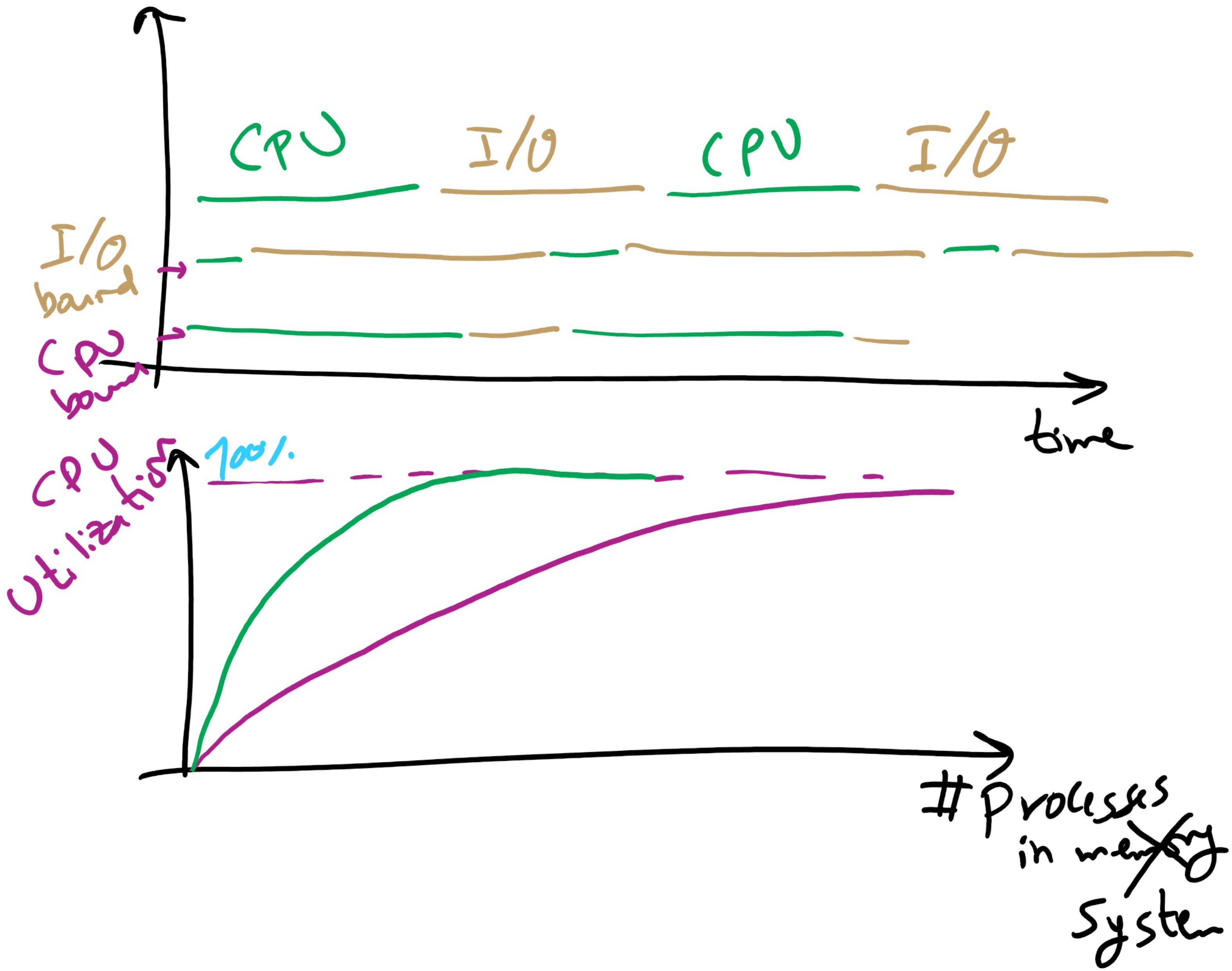
How can we share computer’s memory between multiple processes?

How can we protect each process’s memory partition from other processes?

How many programs is enough?

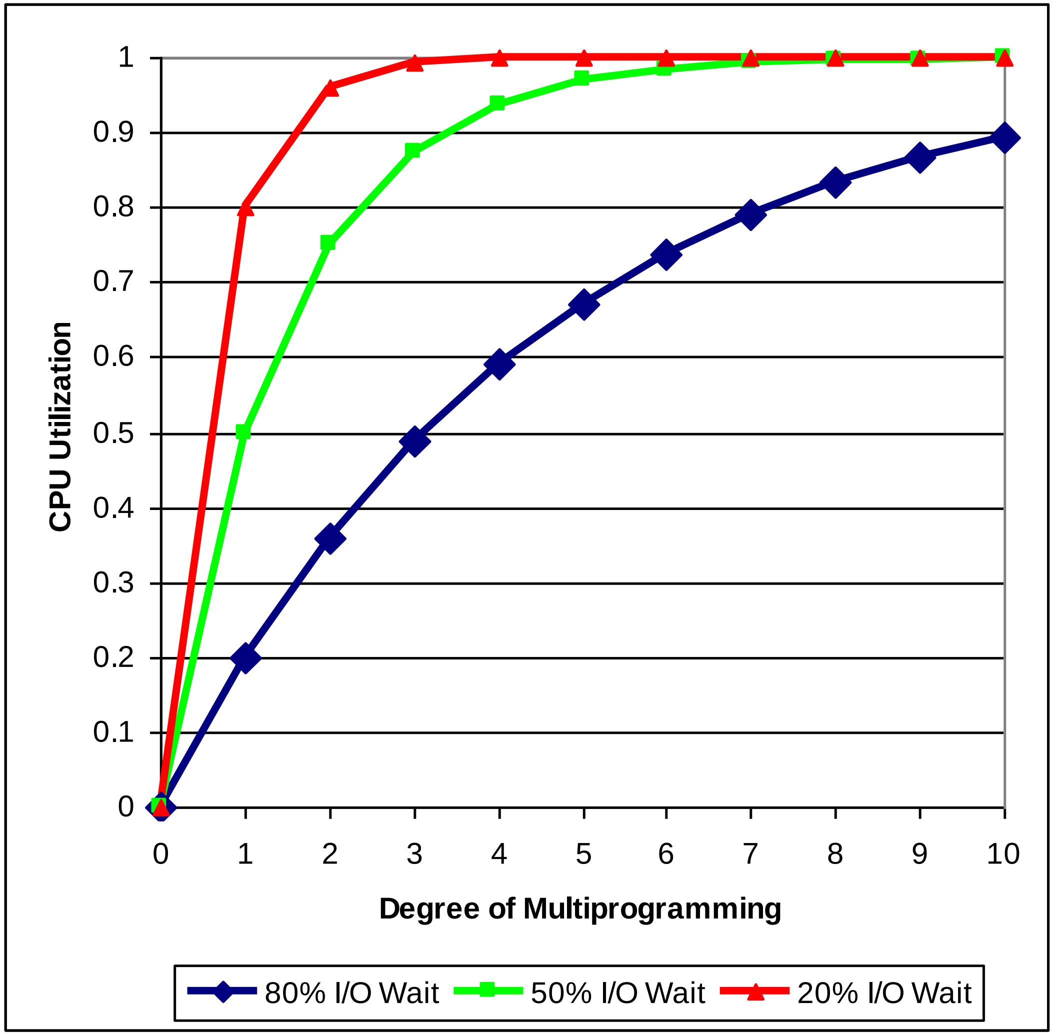
- Several memory partitions (fixed or variable size)
- Lots of processes wanting to use the CPU
- Tradeoff
  - More processes utilize the CPU better
  - Fewer processes use less memory (cheaper!)
- How many processes do we need to keep the CPU fully utilized?
  - This will help determine how much memory we need
  - Is this still relevant with memory costing $10/GB?

Why do we need more processes?



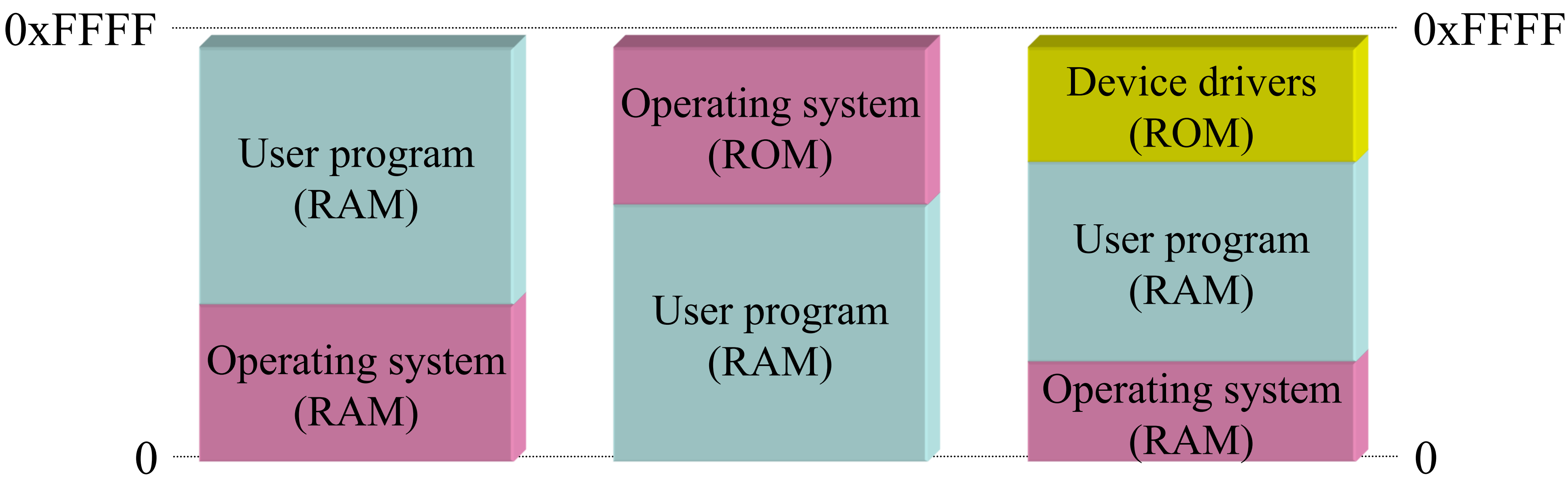
Modeling multiprogramming

- More I/O wait means less processor utilization
  - At 20% I/O wait, 3–4 processes fully utilize CPU
  - At 80% I/O wait, even 10 processes aren't enough
- This means that the OS should have more processes if they're I/O bound
- More processes => memory management & protection more important!

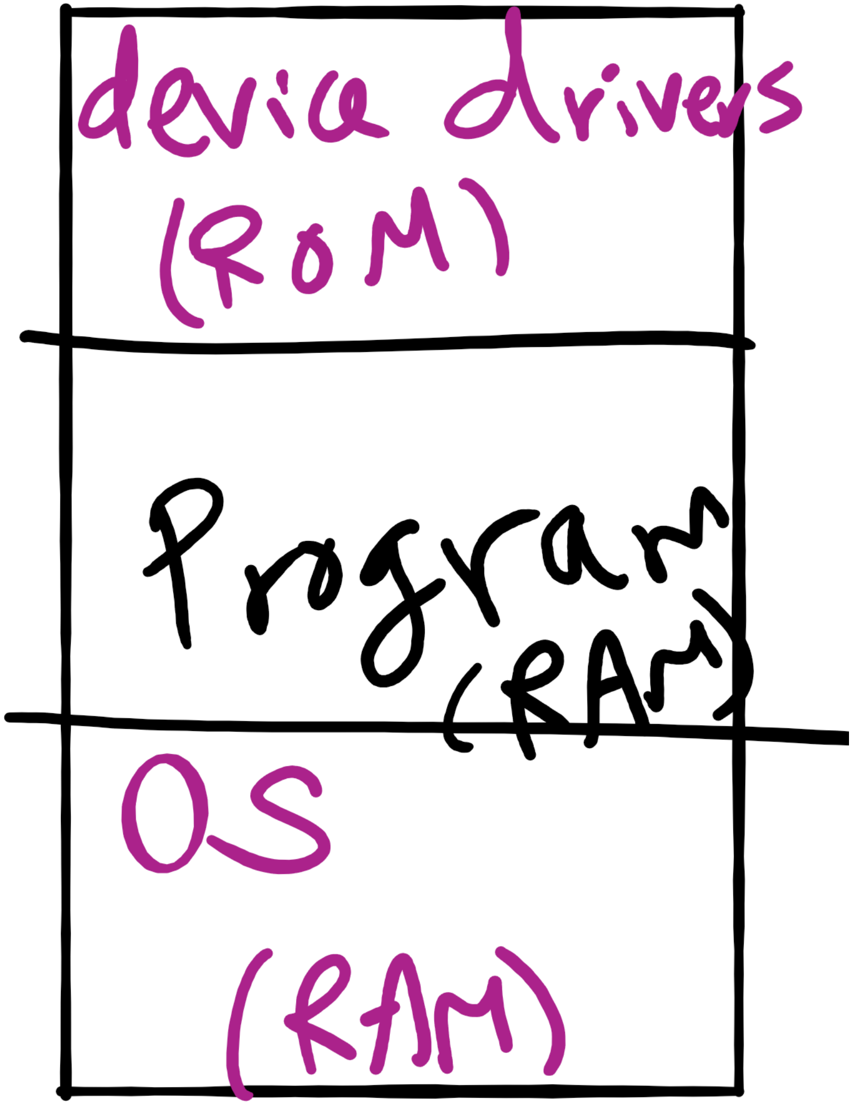


Basic memory management

- Components include
  - Operating system (perhaps with device drivers)
  - Single process
- Goal: lay these out in memory
  - Memory protection may not be an issue (only one program)
  - Flexibility may still be useful (allow OS changes, etc.)
- No swapping or paging

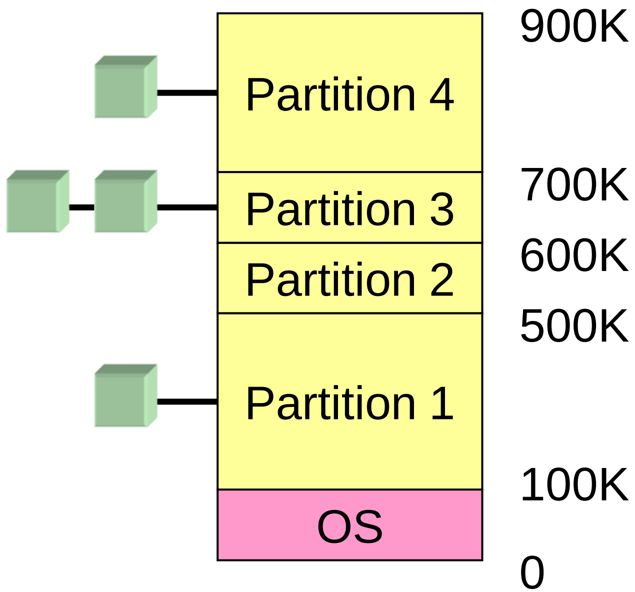


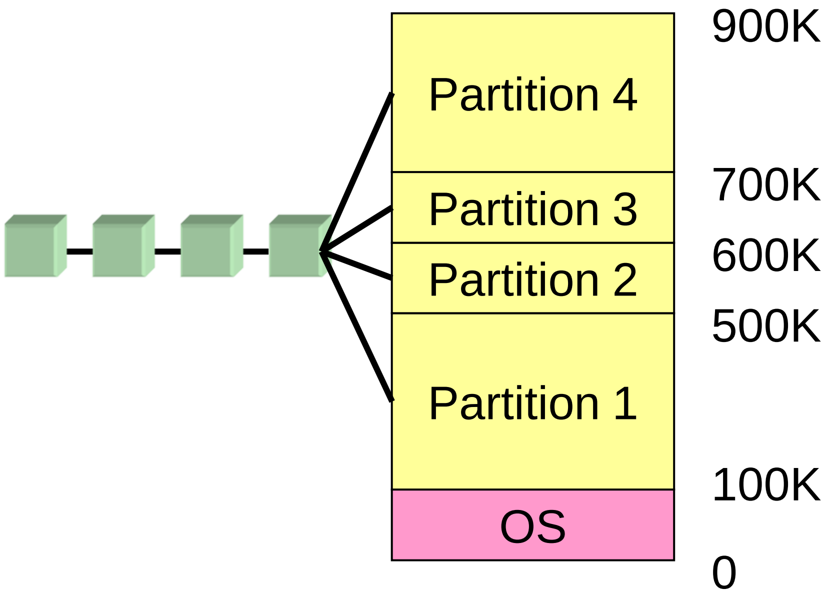
Memory Management for Embedded Systems



Fixed partitions: multiple programs

- Fixed memory partitions
  - Divide memory into fixed spaces
  - Assign a process to a space when it’s free
- Mechanisms
  - Separate input queues for each partition
  - Single input queue: better ability to optimize CPU usage





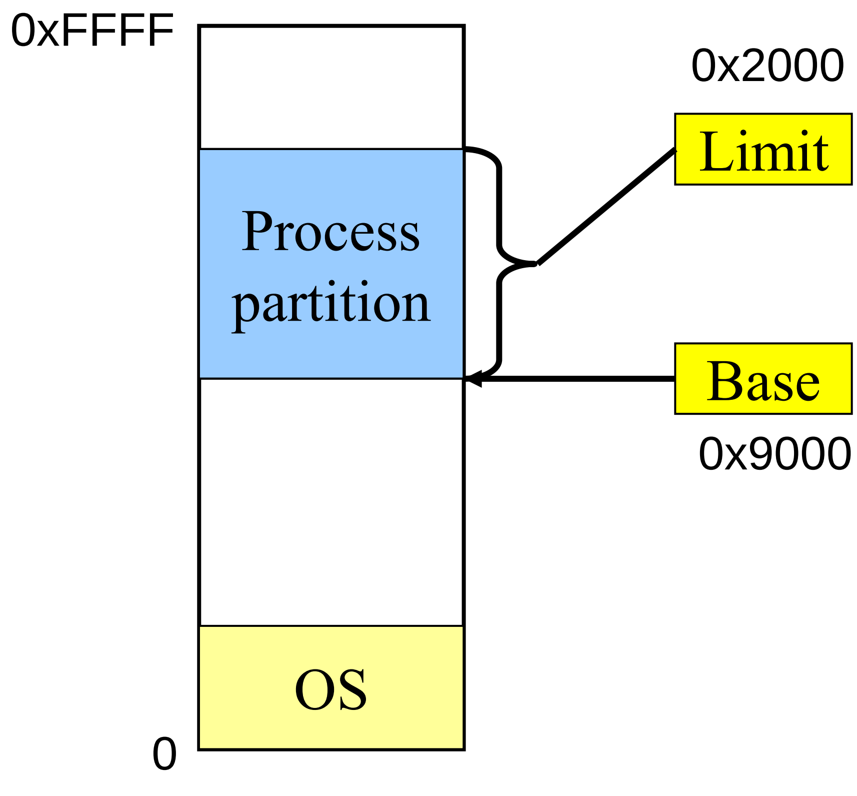
Problem of the Day

How can we share computer’s memory between multiple processes?

How can we protect each process’s memory partition from other processes?

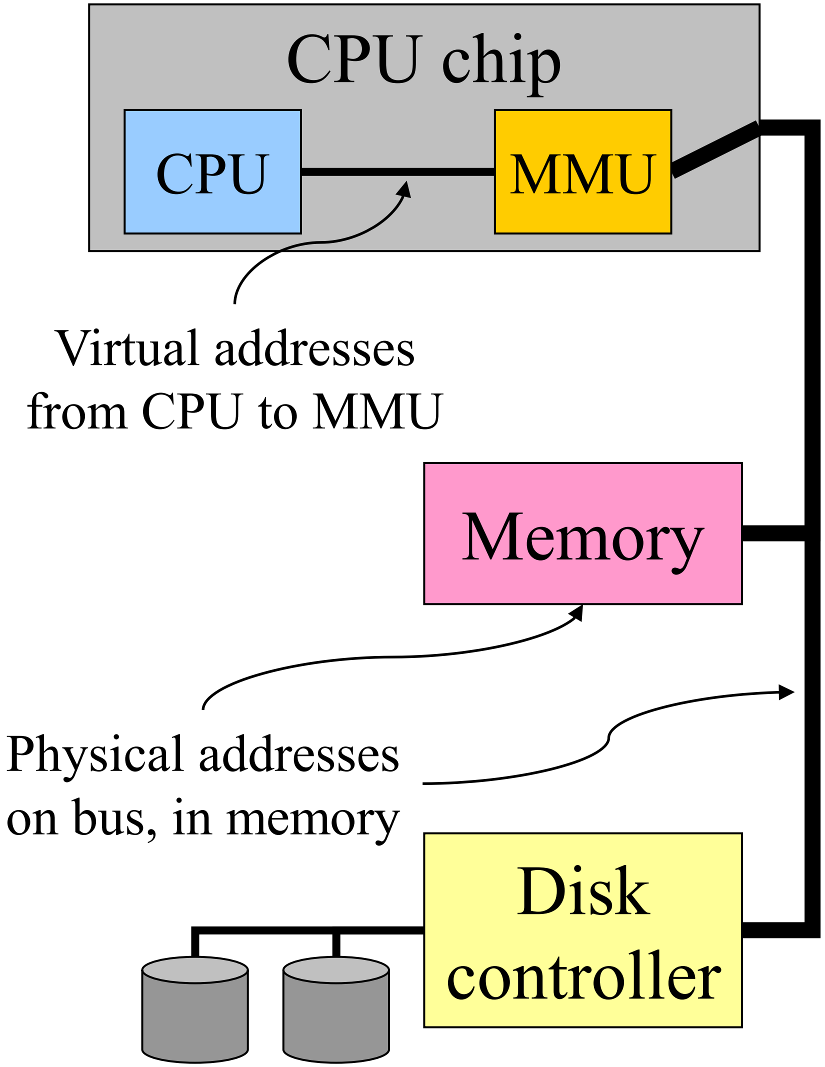
Base and limit registers

- Special CPU registers: base & limit
  - Access to the registers limited to kernel (privileged) mode
  - Registers contain
    - Base: start of the process's memory partition
    - Limit: length of the process's memory partition
- Address generation
  - Physical address: location in actual memory
  - Logical address: location from the process's point of view
  - Physical address = base + logical address
  - Logical address larger than limit => error



Logical address: 0x1204
Physical address:
0x1204+0x9000 = 0xa204

Virtual and physical addresses

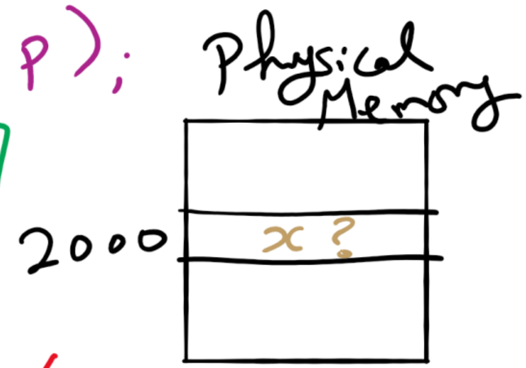


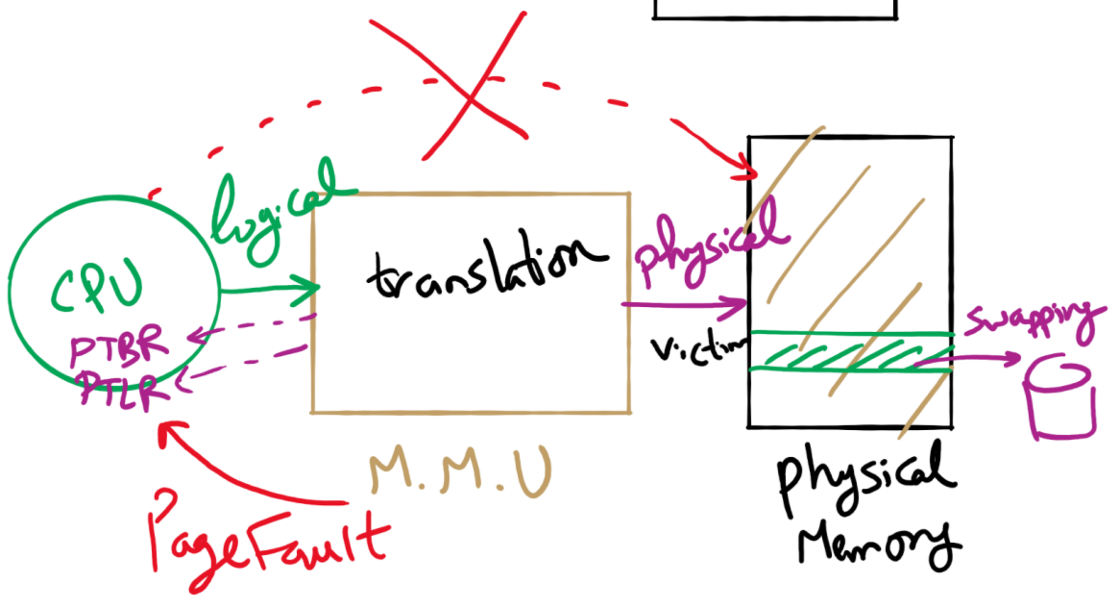
- Program uses *virtual addresses*
  - Addresses local to the process
  - Hardware translates virtual address to *physical address*
- Translation done by the ***M**emory **M**anagement **U**nit*
  - Usually on the same chip as the CPU
  - Only physical addresses leave the CPU/MMU chip
- Physical memory indexed by physical addresses

Address Translation

```
int x = 5;
int *p = &x;
printf("%p\n", p);
```

$\boxed{2000}$



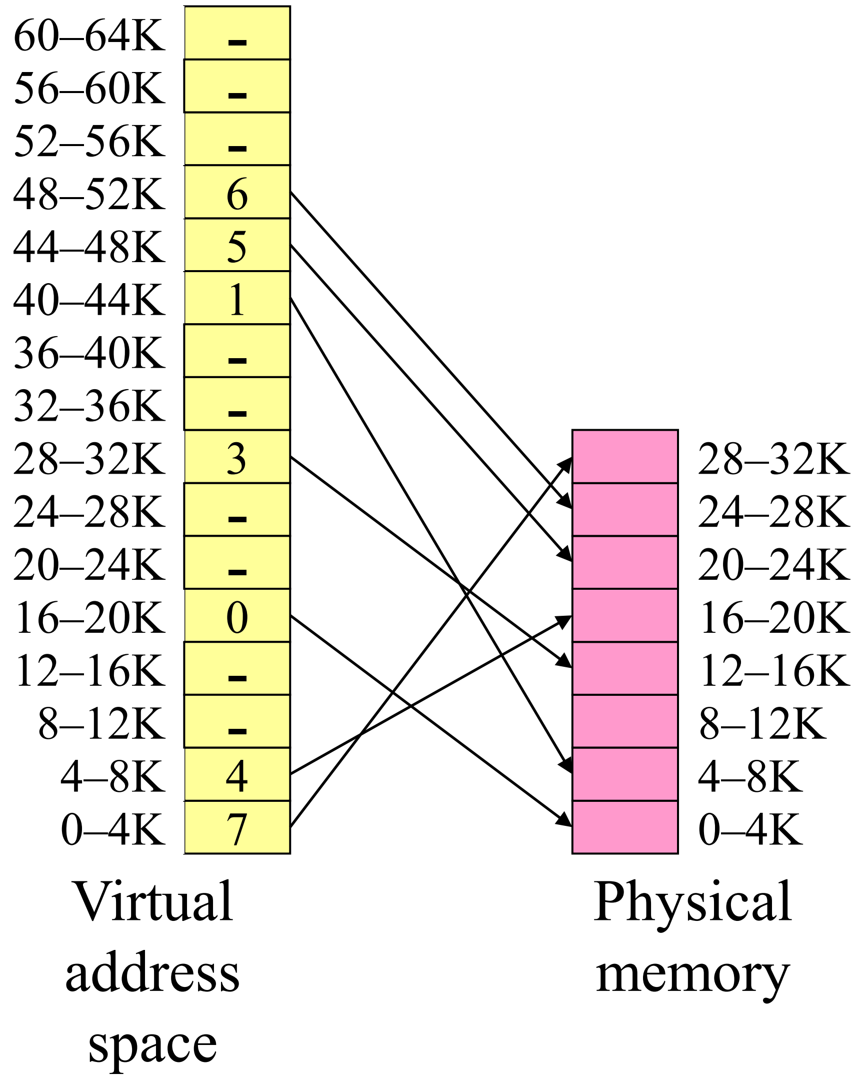


Virtual memory

- Basic idea: allow the OS to hand out more memory than exists on the system
- Keep recently used stuff in physical memory
- Move less recently used stuff to disk
- Keep all of this hidden from processes
  - Processes still see an address space from 0 – max address
  - Movement of information to and from disk handled by the OS without process help
- Virtual memory (VM) especially helpful in multiprogrammed system
  - CPU schedules process B while process A waits for its memory to be retrieved from disk

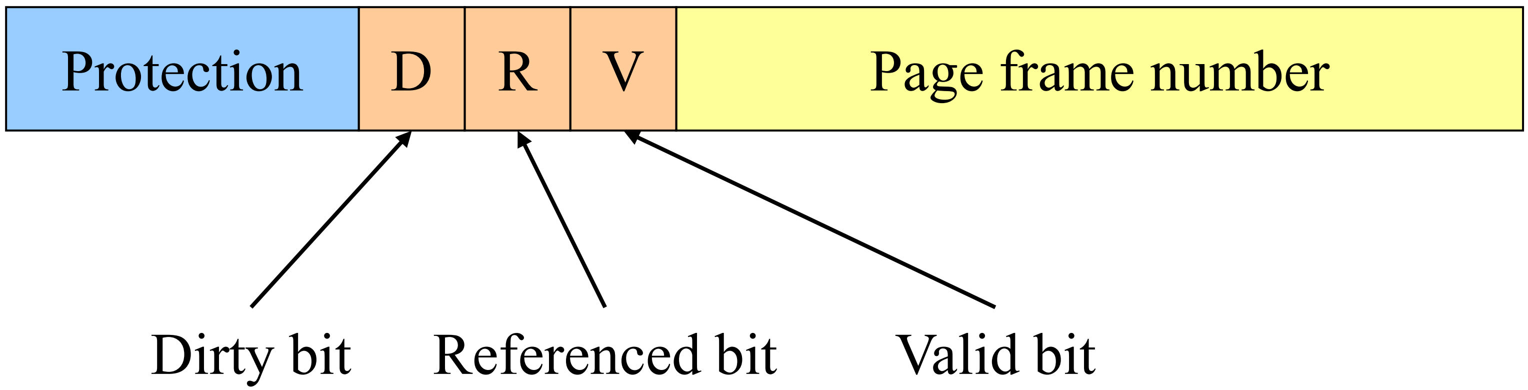
Paging and page tables

- Virtual addresses mapped to physical addresses
  - Unit of mapping is called a *page*
  - All addresses in the same virtual page are in the same physical page
  - *Page table entry* (PTE) contains translation for a single page
- Table translates virtual page number to physical page number
  - Not all virtual memory has a physical page
  - Not every physical page need be used
- Example:
  - 64 KB virtual memory
  - 32 KB physical memory



What's in a page table entry?

- Each entry in the page table contains
  - Valid bit: set if this logical page number has a corresponding physical frame in memory
    - If not valid, remainder of PTE is irrelevant
  - Page frame number: page in physical memory
  - Referenced bit: set if data on the page has been accessed
  - Dirty (modified) bit :set if data on the page has been modified
  - Protection information



Implementing page tables in hardware

- Page table resides in main (physical) memory
- CPU uses special registers for paging
  - Page table base register (PTBR) points to the page table
  - Page table length register (PTLR) contains length of page table: restricts maximum legal logical address
- Translating an address requires two memory accesses
  - First access reads page table entry (PTE)
  - Second access reads the data / instruction from memory
- Reduce number of memory accesses
  - Can’t avoid second access (we need the value from memory)
  - Eliminate first access by keeping a hardware cache (called a *translation lookaside buffer* or TLB) of recently used page table entries