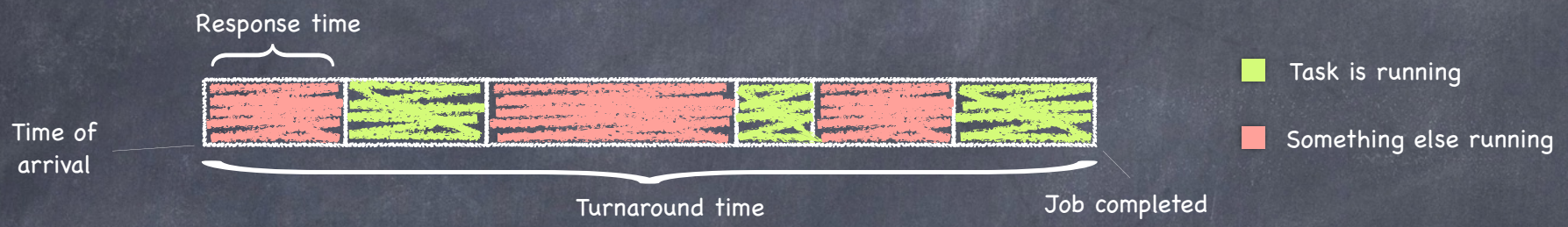
Metrics



- Response time
  - How long between job's arrival and first time job runs?
- Total waiting time
  - How much time on ready queue but not running?
    - sum of "red" intervals above
- Execution time: sum of "green" intervals
- Turnaround time: "red" + "green"
  - Time between a job's arrival and its completion
- Throughput: jobs completed/unit of time (e.g. 10 jobs/sec)

Other Concerns

- Fairness: Who get the resources?
  - Equitable division of resources
- Starvation: How bad can it get?
  - Lack of progress by some job
- Overhead: How much useless work?
  - Time wasted switching between jobs
- Predictability: How consistent?
  - Low variance in response time for repeated requests

When does the Scheduler Run?

- Non-preemptive
  - job runs until it voluntarily yields the CPU
    - process blocks on an event (e.g., I/O or P(sem))
    - process explicitly yields
    - process terminates
- Preemptive
  - all of the above, plus timer and other interrupts
    - when processes can't be trusted
  - incurs some context switching overhead

Context switch overhead

- Cost of saving registers (including, if appropriate, page table register)
- Cost of scheduler determining which process/thread to run next
- Cost of restoring registers (including, if appropriate, page table register)
- Cost of flushing caches
  - L1, L2, L3, TLB

The Perfect Scheduler

- Minimizes response time and turnaround time for each job
- Maximizes overall throughput
- Maximizes resource utilization ("work conserving")
- Meets all deadlines
- Is fair: everyone makes progress, no one starves
- Is envy-free: no core envies the schedule assigned to another core
- Has zero overhead

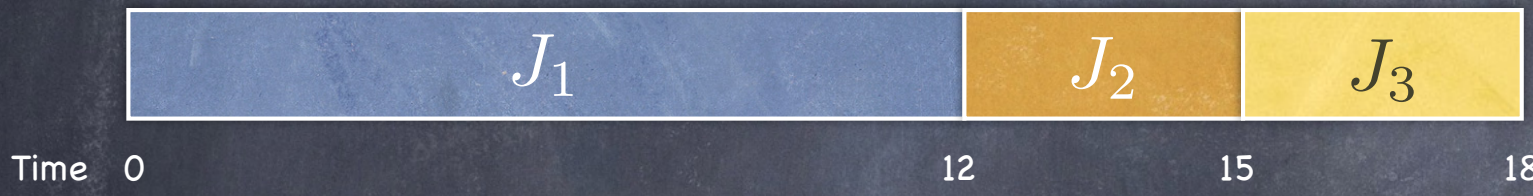
Alas, no such scheduler exists...

Basic Scheduling Algorithms

- FIFO (First In First Out) a.k.a. FCFS
- SJF (Shortest Job First)
- EDF (Earliest Deadline First)
  - preemptive
- Round Robin
  - preemptive
- Shortest Remaining Time First (SRTF)
  - preemptive

FIFO

- Jobs J_1, J_2, J_3 with compute time 12, 3, 3. Same arrival time (so can be scheduled in any order)
  - Scenario 1: Schedule order J_1, J_2, J_3



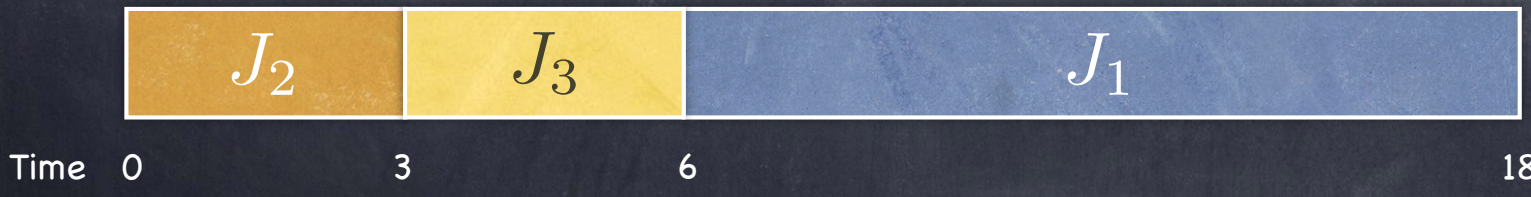
Average Turnaround Time: (12+15+18)/3 = 15

FIFO

- Jobs J_1, J_2, J_3 with compute time 12, 3, 3. Same arrival time (so can be scheduled in any order)
  - Scenario 1: Schedule order J_1, J_2, J_3

Average Turnaround Time: (12+15+18)/3 = 15

  - Scenario 2: Schedule order J_2, J_3, J_1



Average Turnaround Time: (3+6+18)/3 = 9

Average turnaround time very sensitive to schedule order!

FIFO Roundup

Simple
Low overhead
No starvation

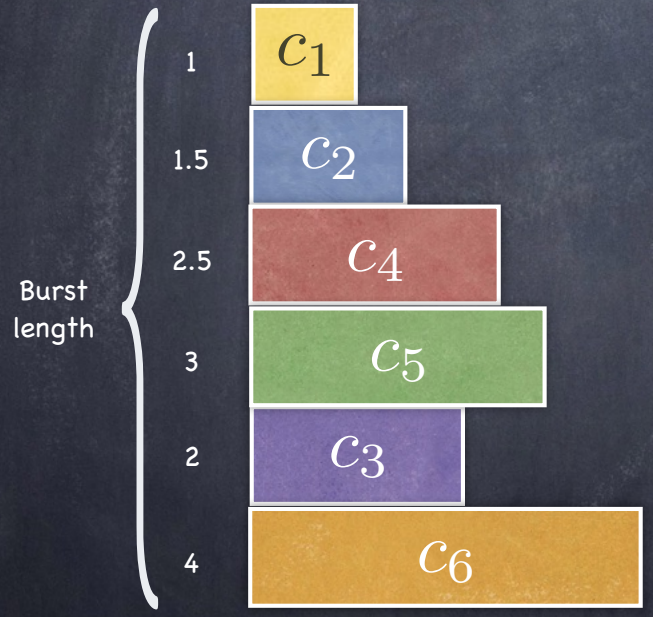
Average turnaround time very sensitive to order/ arrival time

Not responsive to interactive tasks

How to minimize average turnaround time?

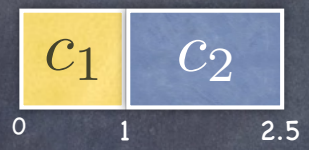
SJF: Shortest Job First

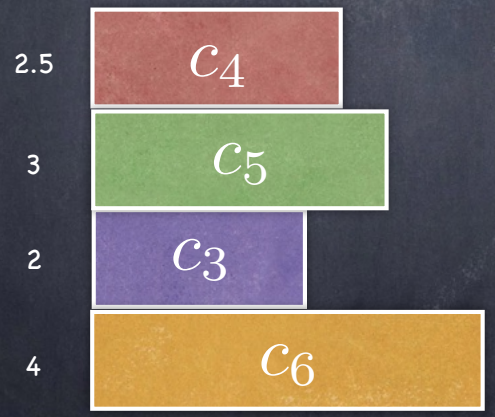
- Schedule jobs in order of estimated completion time
  (or, better, shortest length of next CPU burst!)



SJF: Shortest Job First

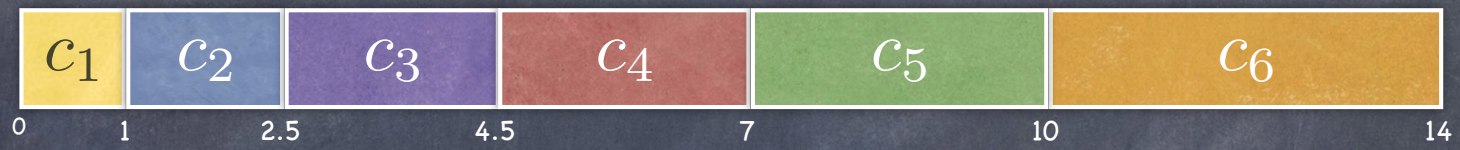
- Schedule jobs in order of estimated completion time



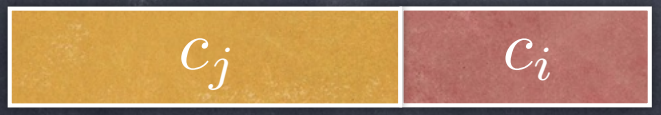


SJF: Shortest Job First

- Schedule jobs in order of estimated completion time



- Average Turnaround time (att): 39/6 = 6.5
- Would a different schedule produce a lower turnaround time?

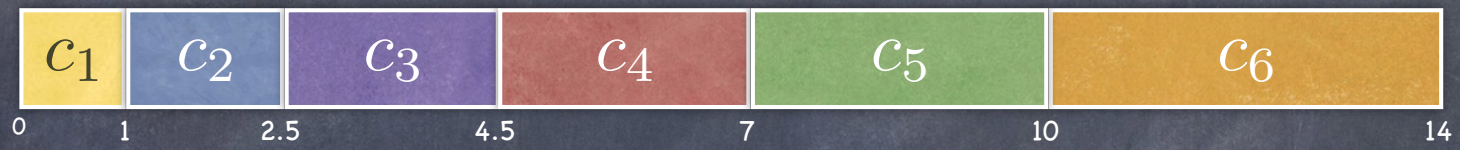
Consider 

where $c_i < c_j$

att = $(c_j + (c_i + c_j))/2$

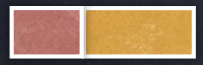
SJF: Shortest Job First

- Schedule jobs in order of estimated completion time



- Average Turnaround time (att): 39/6 = 6.5
- Would a different schedule produce a lower turnaround time?

Consider  where $c_i < c_j$



att = $(c_i + (c_i + c_j))/2$

$<$

att = $(c_j + (c_i + c_j))/2$

SJF Roundup

Optimal average turnaround time

Job's turnaround time depends on length of other jobs

Pessimal variance in turnaround time for a given task

Need to estimate execution time

Can starve long jobs

SJF Roundup

Optimal average turnaround time

Job's turnaround time depends on length of other jobs

Pessimal variance in turnaround time for a given task

Need to estimate execution time

Can starve long jobs

Shortest Process Next (SJF for interactive jobs)

- Enqueue in order of estimated completion time
  - Exponential moving average (EMA): Use recent history as indicator of near future
- Let $t_n =$ duration of n^{th} CPU burst
  $\tau_n =$ estimated duration of n^{th} CPU burst
  $\tau_{n+1} =$ estimated duration of next CPU burst

$$\tau_{n+1} = \alpha\tau_n + (1-\alpha)t_n$$

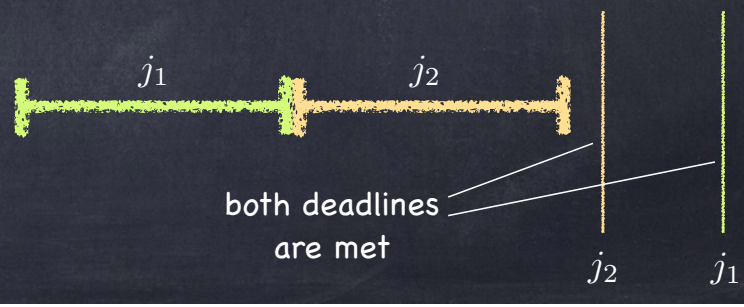
$0 \leq \alpha \leq 1$ determines weight placed on past behavior

Earliest Deadline First (EDF)

- Schedule in order of earliest deadline; preemptive
- If a schedule exists that meets all deadlines, then EDF will generate that schedule!
  - does not even need to know the execution times of the jobs!

Informal Proof

- Let S be a schedule of a set of jobs that meets all deadlines
- Let j_1 and j_2 be two neighboring jobs in S so that j_1.deadline > j_2.deadline
- Let S' be S with j_1 and j_2 switched
  - S' also meets all deadlines!
- Repeat until sorted (i.e., bubblesort)
  - Resulting schedule is EDF



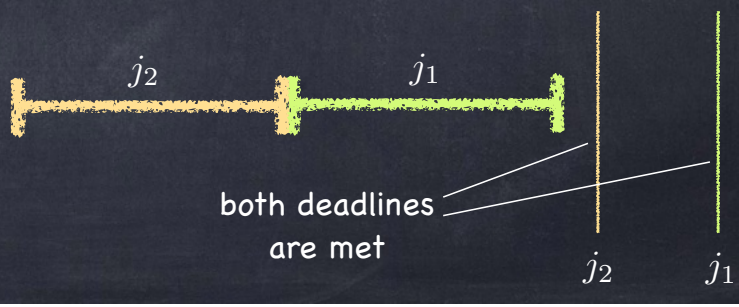
Earliest Deadline First (EDF)

- Schedule in order of earliest deadline; preemptive
- If a schedule exists that meets all deadlines, then EDF will generate that schedule!
  - does not even need to know the execution times of the jobs!

..but only if tasks need only the processor!

Informal Proof

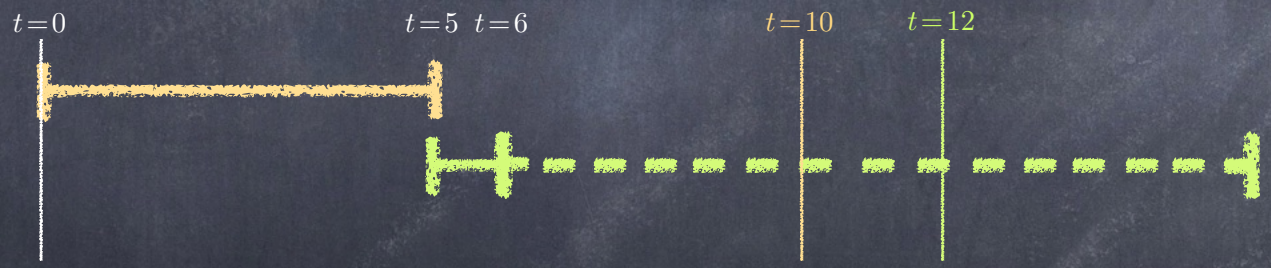
- Let S be a schedule of a set of jobs that meets all deadlines
- Let j_1 and j_2 be two neighboring jobs in S so that j_1.deadline > j_2.deadline
- Let S' be S with j_1 and j_2 switched
  - S' also meets all deadlines!
- Repeat until sorted (i.e., bubblesort)
  - Resulting schedule is EDF



When EDF fails

- Two jobs:
  - j_1 : deadline at $t=12$; 1 unit of computation, 10 of I/O
  - j_2: deadline at $t=10$; 5 units of computation

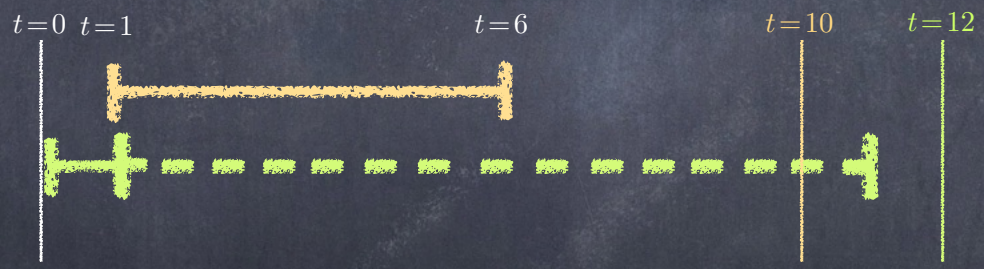
EDF:



but...

When EDF fails

- Two jobs:
  - j_1 : deadline at $t=12$; 1 unit of computation, 10 of I/O
  - j_2: deadline at $t=10$; 5 units of computation



- Need to think of jobs at a finer granularity:
  - Real deadline for the computing portion of j_1 is 2!

EDF Roundup

Meets deadlines if possible (but beware...)

Free of starvation

CPU-bound jobs will make
I/O-bound jobs wait

Cannot decide when to
run jobs without deadlines

Round Robin

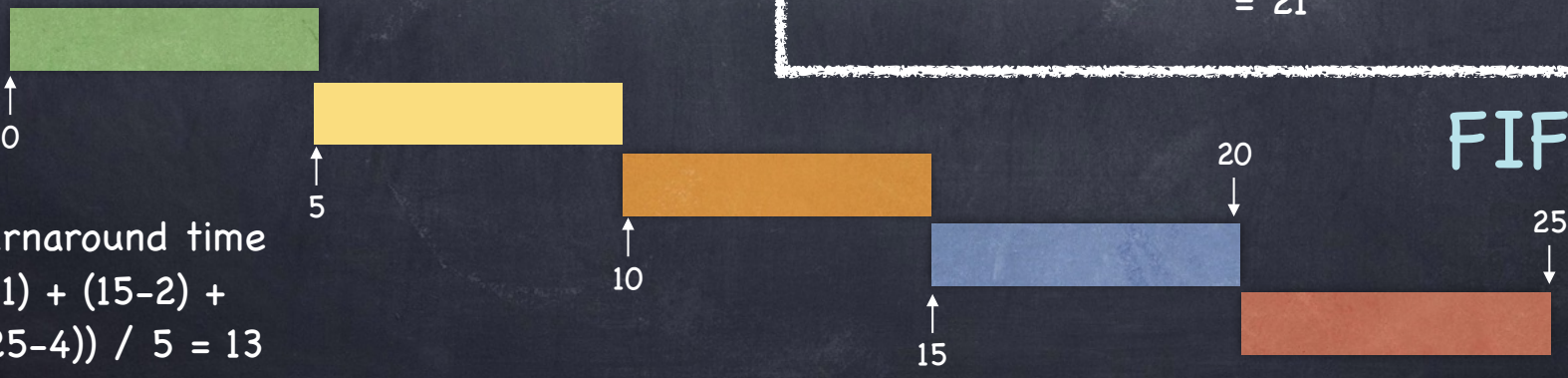
- Each process is allowed to run for a quantum
- Context is switched (at the latest) at the end of the quantum — preemption!
- Next job to run is the one that hasn't run for the longest amount of time
- What is a good quantum size?
  - Too long, and it morphs into FIFO
  - Too short, and too much time lost context switching
  - Typical quantum: about 100X cost of context switch (~100ms vs. << 1ms)

Round Robin vs FIFO

5 jobs of about equal length (5 units of time)

RR

Average Turnaround time
(21 + (22-1) + (23-2) + (24-3) + (25-4)) / 5
= 21

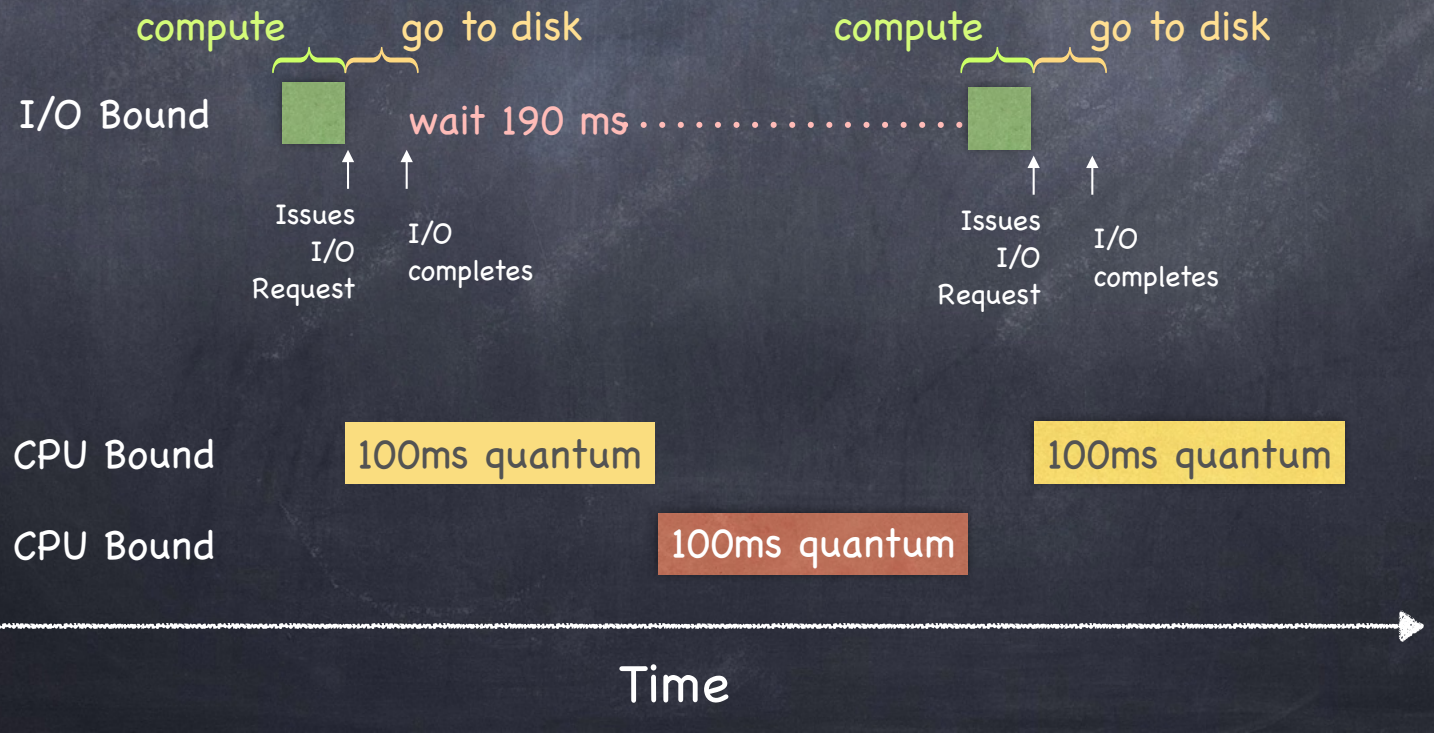


FIFO/SJF

Average Turnaround time
(5 + (10-1) + (15-2) +
(20-3) + (25-4)) / 5 = 13

At least it is fair..?

- Mix of one I/O-bound and two CPU-bound jobs
  - I/O-bound: compute; go to disk; repeat



Round Robin Roundup

No starvation

Can reduce response time

Overhead of context switching

Mix of I/O and CPU bound

Particularly bad average turnaround for close-in arrival-time, equal length jobs

SJF

- J_1 arrives at time 0; J_2, J_3 arrive at time 10

Average Turnaround Time:
100+(110-10)+(120 -10)/3
= 103.33

SJF + Preemption

- J_1 arrives at time 0; J_2, J_3 arrive at time 10

Average Turnaround Time:
100+(110-10)+(120 -10)/3
= 103.33

- With a preemptive scheduler — SRTF

Shortest Remaining Time First

At end of each quantum, scheduler selects job with the least remaining time to run next

- Often same job is selected, avoiding a context switch...
- ...but new short jobs see improved response time

Average Turnaround Time:
(120-0)+(20-10)+(30-10)/3
= 50

SRTF Roundup

Good response time and turnaround time of I/O bound processes

Bad turnaround time and response time for CPU bound processes

Need estimate of execution for each job

Starvation