Scheduling

Science

Computer Science

CS 450: Operating Systems

Michael Saelee <lee@iit.edu>

IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

§Overview

scheduling: ***policies*** & ***mechanisms*** used to allocate a ***resource*** to some set of ***entities***

resource & ***entities***: ***CPU*** & ***processes***

other possibilities:

- resources: memory, I/O bus/devices
- entities: threads, users, groups

policy: high-level "what"

- aka scheduling *disciplines*

mechanism: low-level "how"

- e.g., interrupts, context switch

(we'll start with ***policy*** first)

essential idea:

- CPU(s) are a *limited* resource
- efficiently allow for time-sharing of CPU(s) amongst multiple processes
  - enables *concurrency* on a single CPU

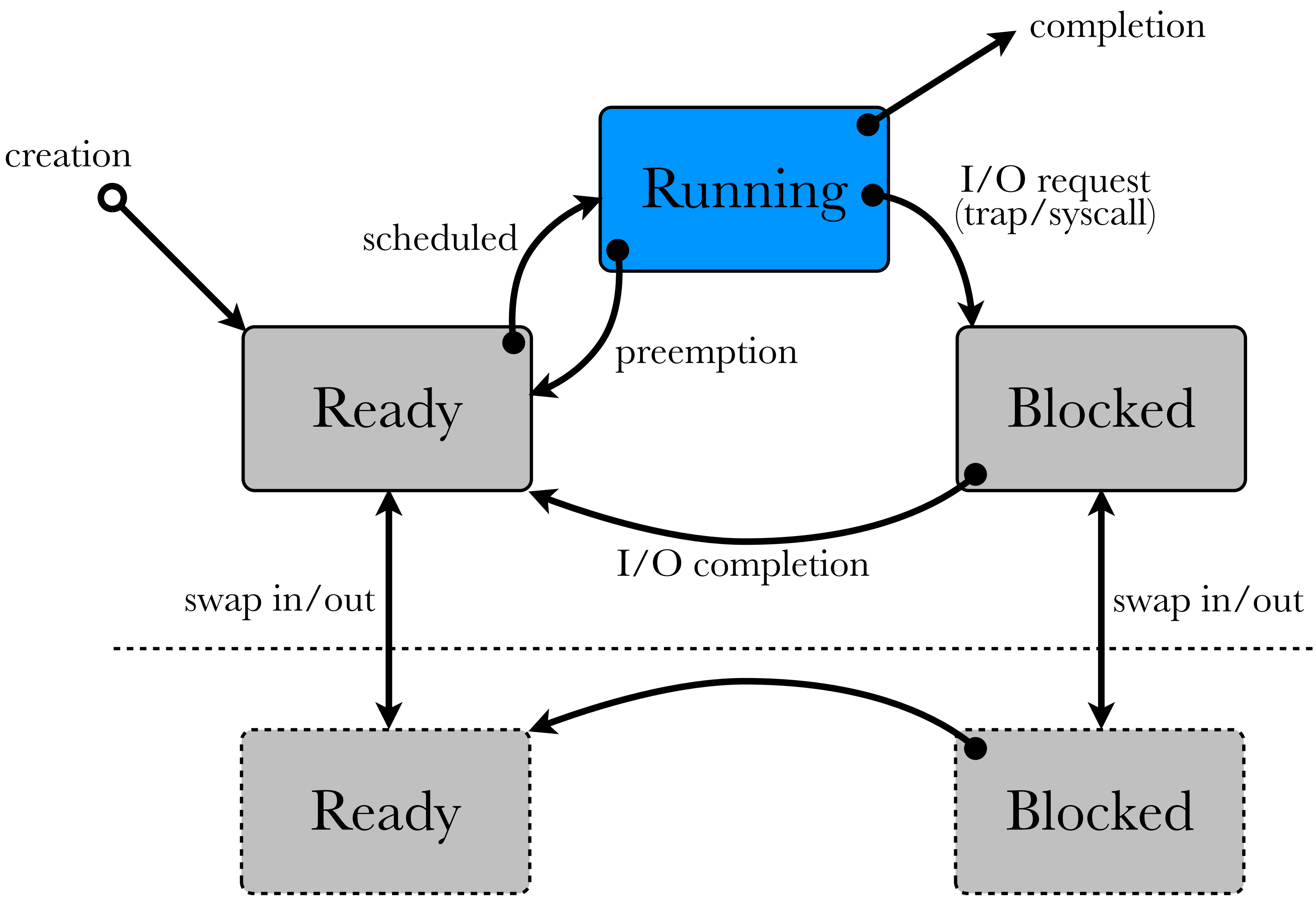
at a high level (policy), only concern ourselves with *macro process state*

one of **running**, **ready**, or **blocked**

running = consuming CPU

ready = “runnable”, but not running

blocked = not runnable
(e.g., waiting for I/O)


completion
creation
Running
I/O request
(trap/syscall)
scheduled
preemption
Ready
Blocked
I/O completion
swap in/out
swap in/out
Ready
Blocked

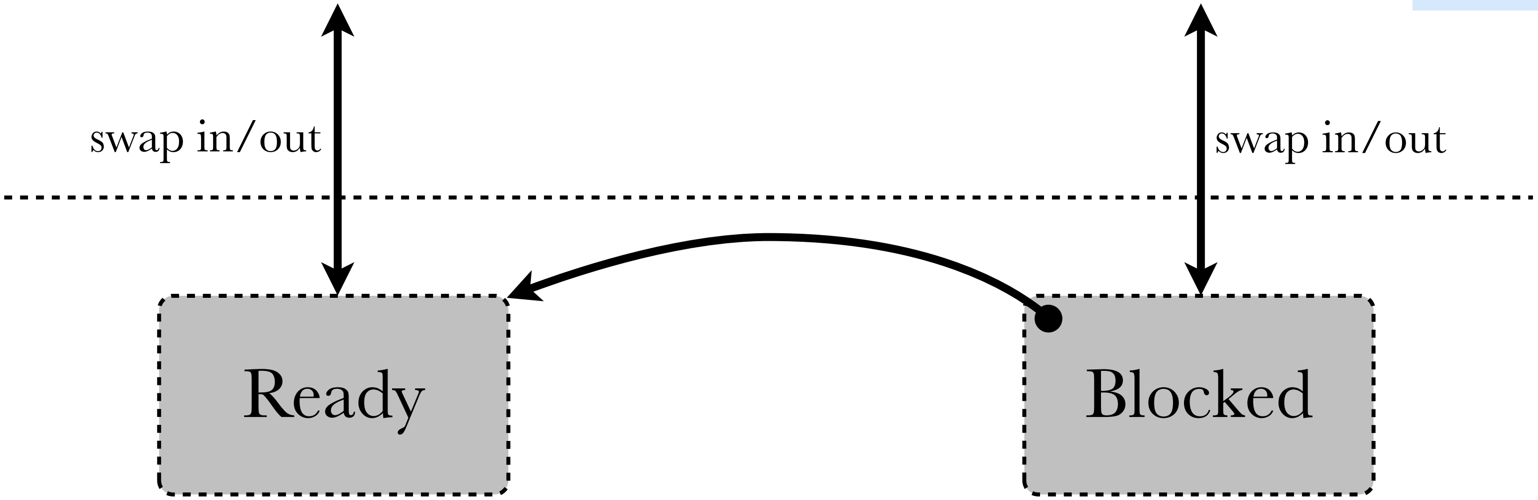
preemptive scheduling

☑ running → ready transition

non-preemptive scheduling

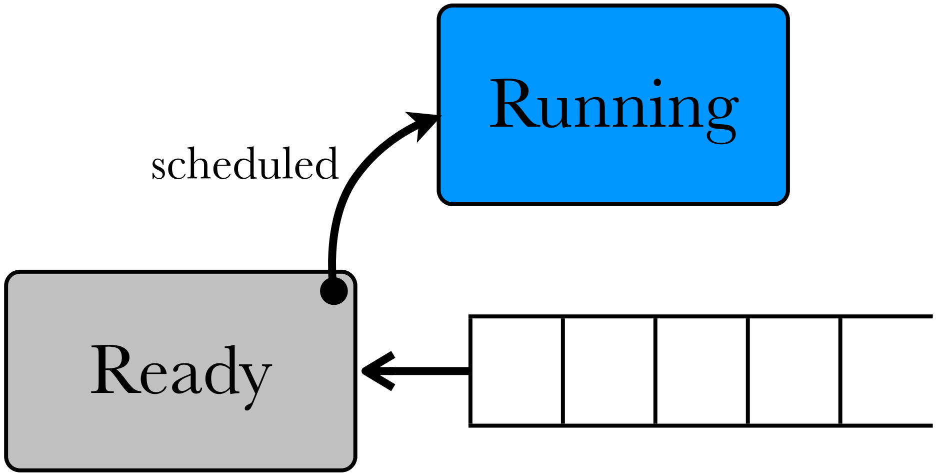
☒ running → ready transition

i.e., *not* = *batch*!



domain of the “swapper” — separate from the CPU scheduler

- frequency in seconds vs. ms
- ignore for now



convenient to envision a *ready queue/set*

scheduling policy is used to select the next running process from the ready queue policies vary by:

1. preemptive vs. non-preemptive
2. factors used in selecting a process
3. goals; i.e., *why* are we selecting a given process?

scheduling goals are usually predicated on *optimizing* certain *scheduling metrics*

— can be *provable* or based on *heuristics*

§Scheduling Metrics

metrics we'll be concerned with:

- turnaround time
- wait time
- response time
- throughput
- utilization

turnaround time:

$$T_{turnaround} = T_{completion} - T_{creation}$$

i.e., total time to complete process

turnaround time depends on much more than the scheduling discipline!

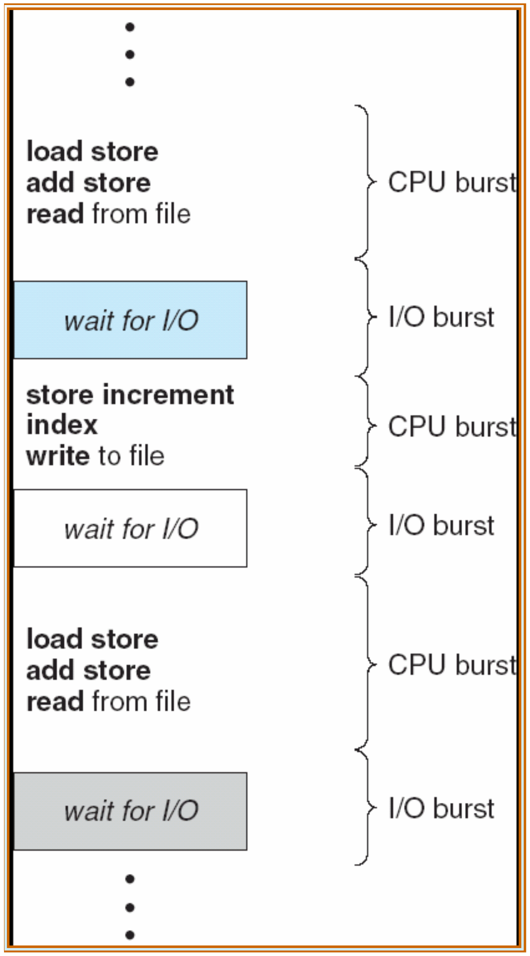
- process runtime
- process I/O processing time
- how many CPUs available
- how many other processes need to run

wait time: time spent in ready queue

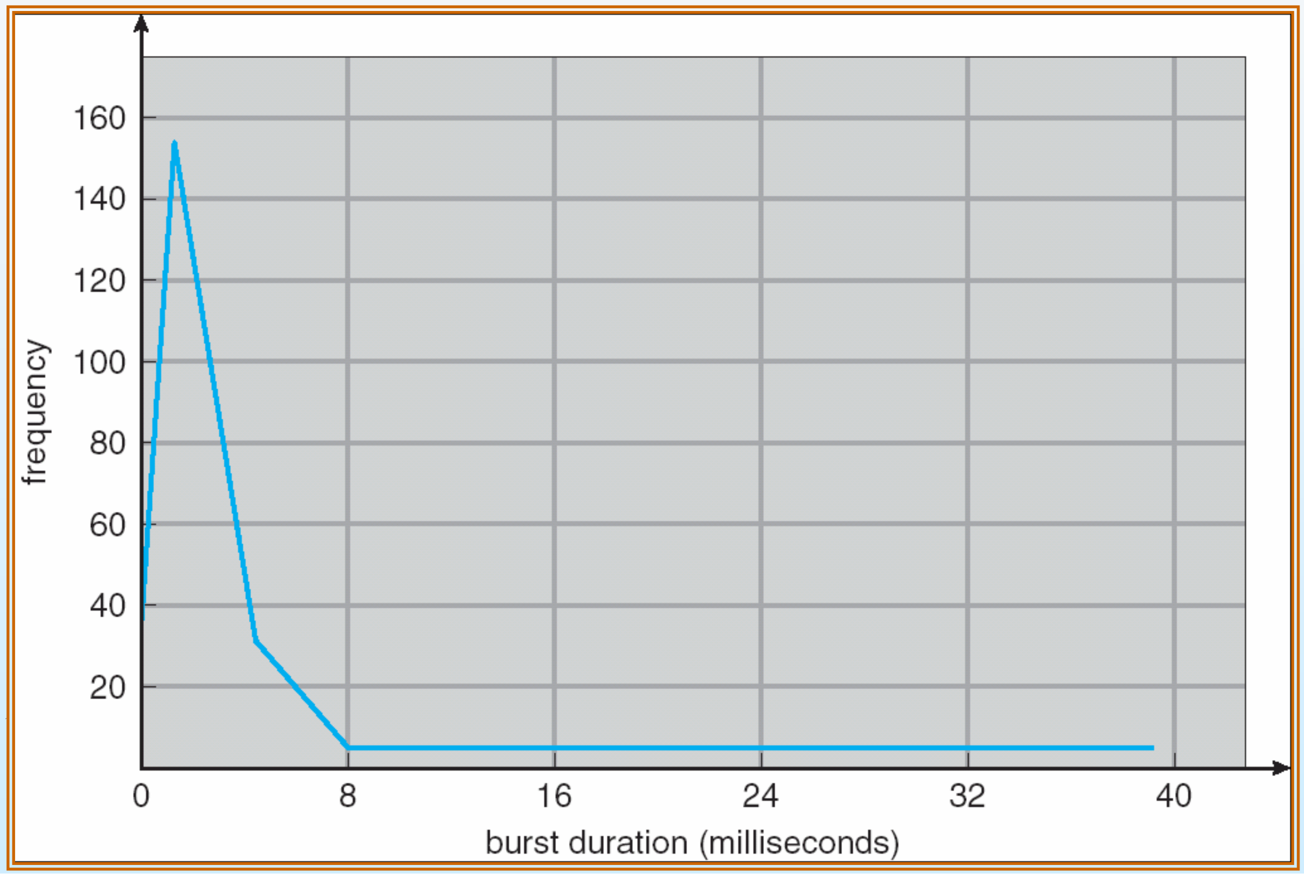
i.e., how long does the scheduler force a runnable process to wait for a CPU

- better gauge of *scheduler's* effectiveness turnaround & wait time are measured over the course of an *entire process* — sometimes refer to as the "job"

- not a very useful metric for *interactive* processes
  - which typically alternate between CPU & I/O *bursts*, indefinitely



“bursty” execution



burst length histogram

can take measurements *per-burst*

i.e., from first entry into ready queue
to completion *or* transition to blocked

- burst turnaround time, aka *response time*
- burst wait time

throughput:

number of completed jobs or bursts per time unit (e.g., N/sec) *utilization*:

% of time CPU is busy running jobs

- note: CPU can be idle if there are no active jobs *or* if all jobs are blocked!

another (subjective) metric: *fairness*

- what does this mean?
- how to measure it?
- is it useful?

§Scheduling Policies

1. **F**irst **C**ome **F**irst **S**erved

Process	Arrival Time	Burst Time
P_1	0	24
P_2	0	3
P_3	0	3

“Gantt chart”

Wait times: $P_1 = 0$, $P_2 = 24$, $P_3 = 27$

Average: $(0+24+27)/3 = 17$

Convoy Effect

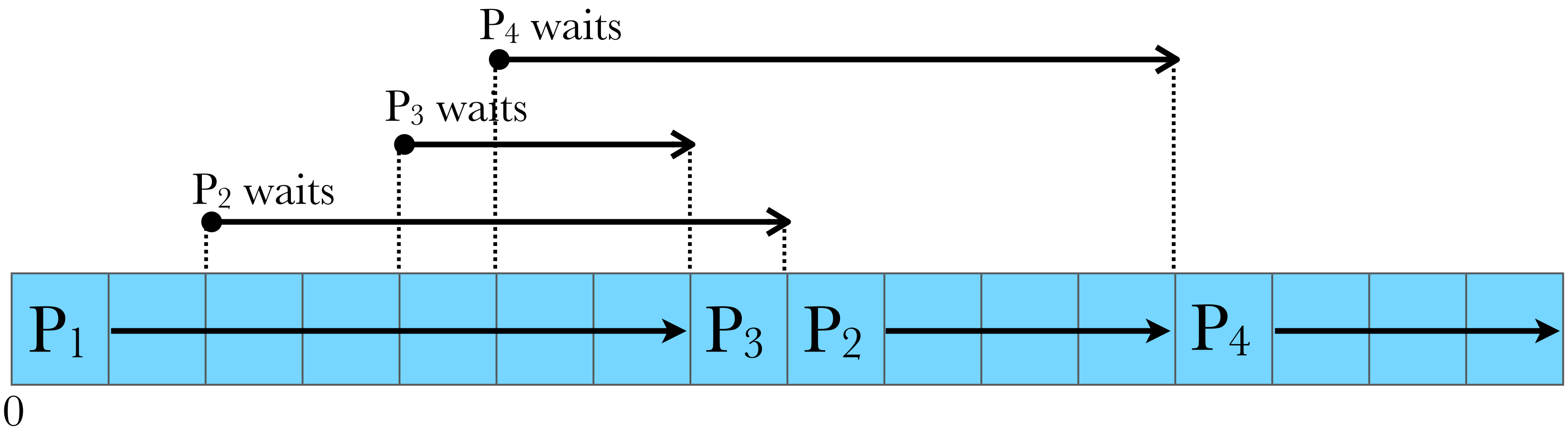
Process	Arrival Time	Burst Time
P_3	0	3
P_2	0	3
P_1	0	24

Wait times: $P_1 = 6, \; P_2 = 3, \; P_3 = 0$

Average: $(6+3+0)/3 = 3$

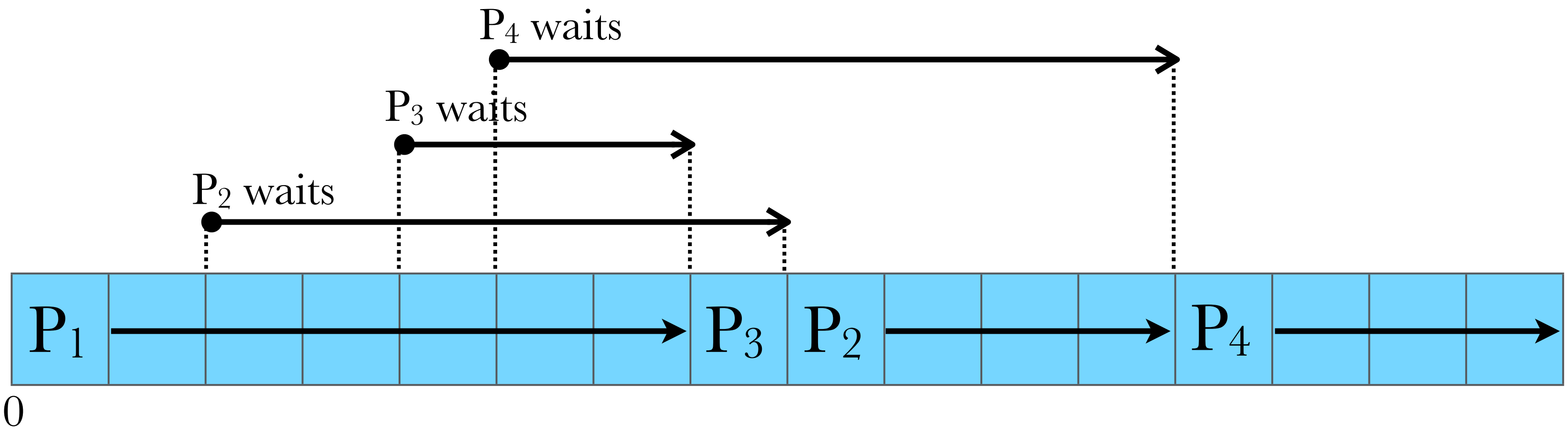
2. **S**hortest **J**ob **F**irst

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Non-preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Wait times: $P_1 = 0$, $P_2 = 6$, $P_3 = 3$, $P_4=7$

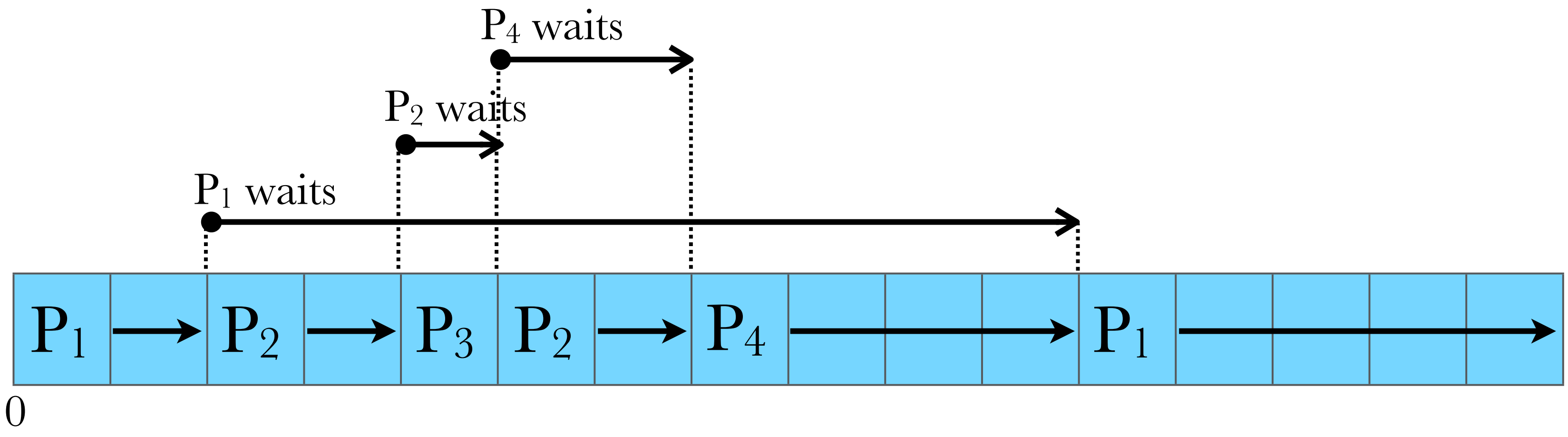
Average: $(0+6+3+7)/4 = 4$

can we do better?

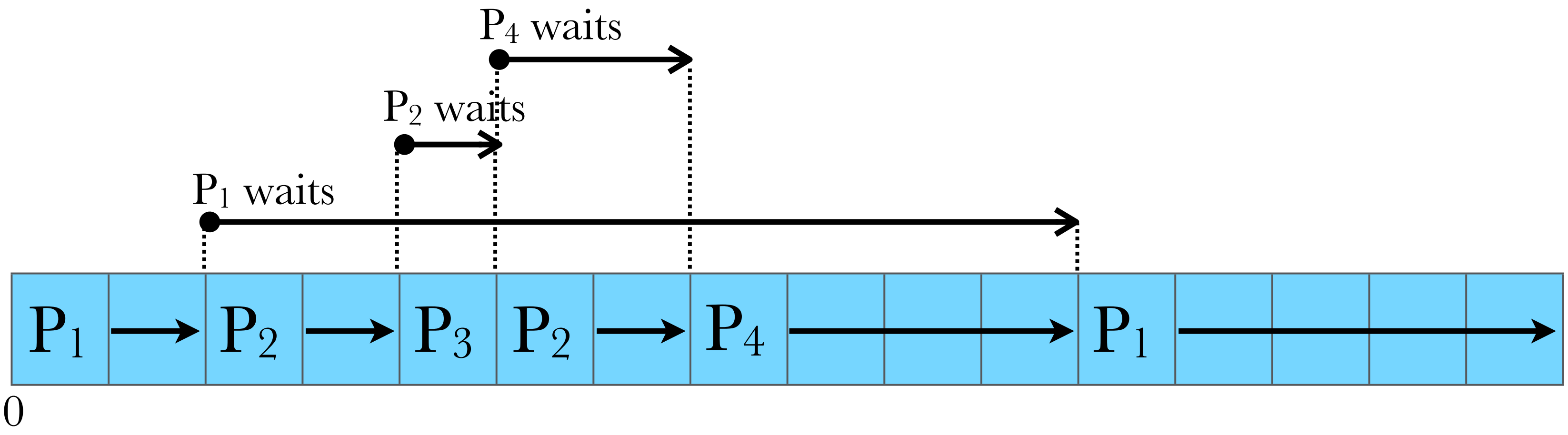
Yes! (theoretically): **Preemptive** SJF

a.k.a. **S**hortest-**R**emaining-**T**ime-**F**irst

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Wait times: $P_1 = 9$, $P_2 = 1$, $P_3 = 0$, $P_4 = 2$

Average: $(9+1+0+2)/4 = 3$

SJF/SRTF are *greedy* algorithms;

i.e., they always select the *local optima*

greedy algorithms don’t always produce *globally* optimal results (e.g., hill-climbing) consider 4 jobs arriving at t=0, with burst lengths t_0, t_1, t_2, t_3

avg. wait time if scheduled in order?

$$= \frac{3t_0 + 2t_1 + t_2}{4}$$

$$= \frac{3t_0 + 2t_1 + t_2}{4}$$

— a *weighted average*; clearly minimized by running shortest jobs first.

I.e., SJF/PSJF are provably optimal w.r.t. wait time!

at what cost?

… potential *starvation*!

(possible for both non-preemptive & preemptive variants) also, we've been making two simplifying assumptions:

1. context switch time = 0
2. burst lengths are known in advance

(1) will be dealt with later;

(2) is a serious problem!

typically predict future burst lengths based on past job behavior

- simple moving average
- exponentially weighted moving average (EMA)

Observed: $\rho_{n\text{-}1}$

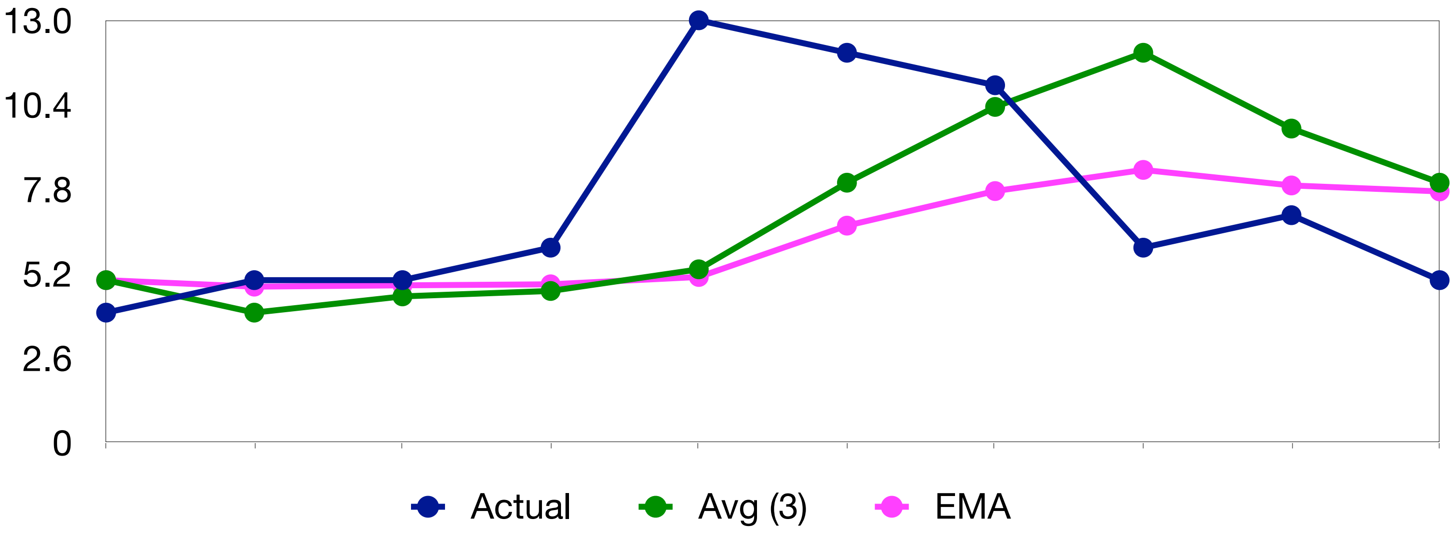
Estimated: $\sigma_{n\text{-}1}$

Weight (α): $0 \leq \alpha \leq 1$

EMA: $\sigma_n = \alpha \cdot \rho_{n\text{-}1} + (1-\alpha) \cdot \sigma_{n\text{-}1}$

Actual	**Avg (3)**	**Error**	**EMA**	**Error**
4	5.00	1.00	5.00	1.00
5	4.00	1.00	4.80	0.20
5	4.50	0.50	4.84	0.16
6	4.67	1.33	4.87	1.13
13	5.33	7.67	5.10	7.90
12	8.00	4.00	6.68	5.32
11	10.33	0.67	7.74	3.26
6	12.00	6.00	8.39	2.39
7	9.67	2.67	7.92	0.92
5	8.00	3.00	7.73	2.73
Avg err:		**2.78**		**2.50**

EMA Alpha: **0.2**



how to deal with starvation?

one way: enforce *fairness*

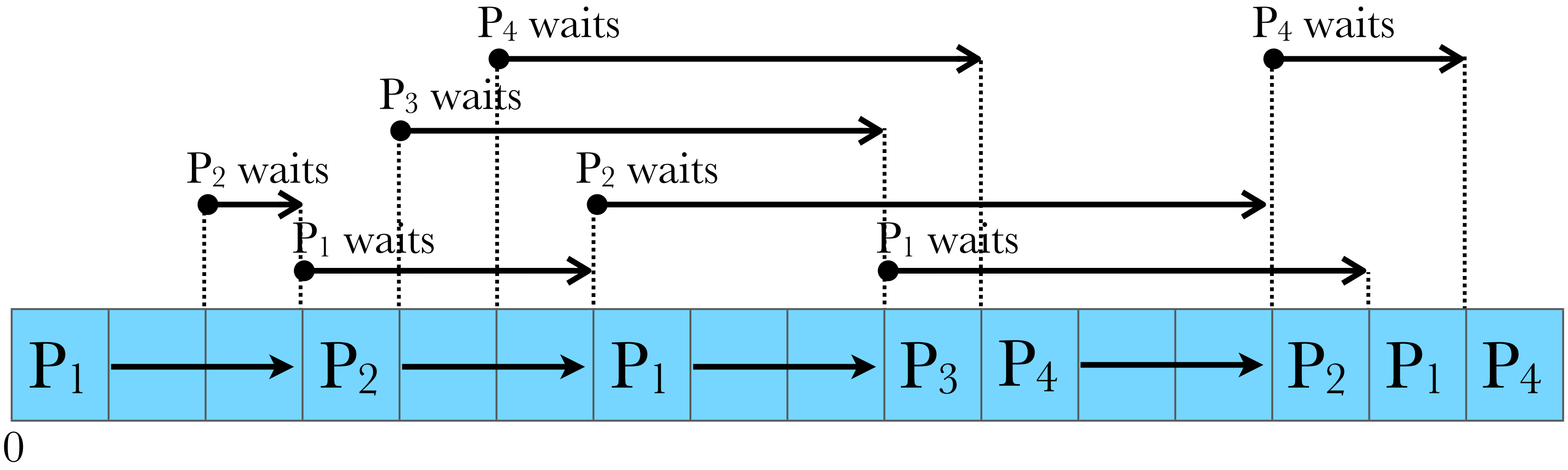
3. *Round Robin*: the "fairest" of them all

- FIFO queue
- each job runs for max *time quantum*
- if unfinished, re-enter queue at back

Given time quantum q and n jobs:

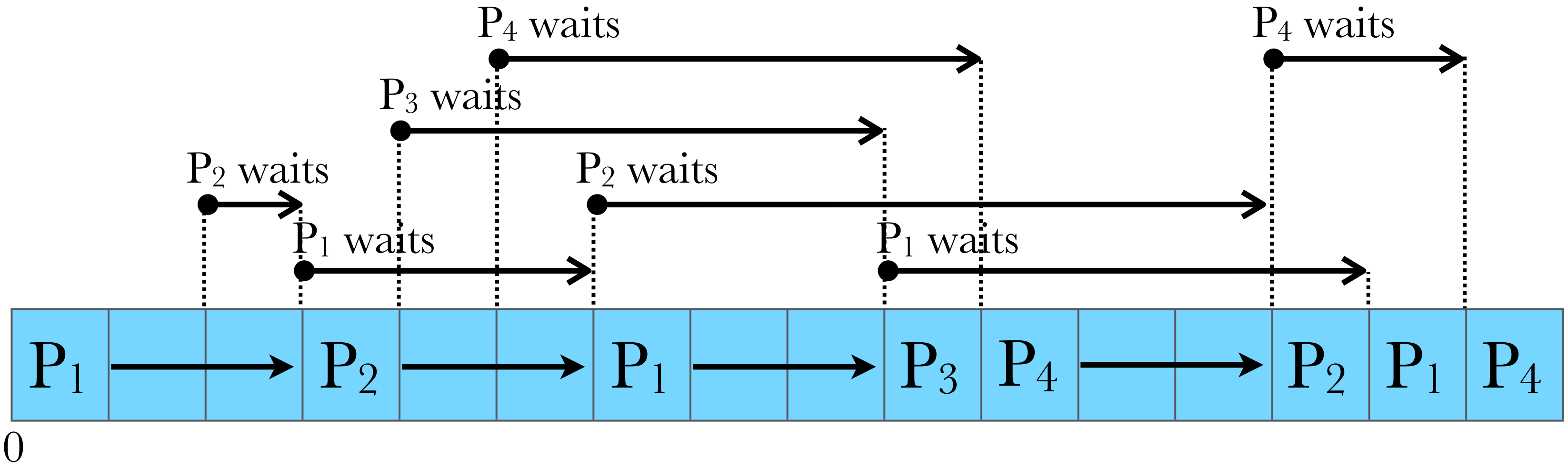
- max wait time = $q \cdot (n - 1)$
- each job receives $1/n$ timeshare

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Round Robin, $q=3$

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



Wait times: P1 = 8, P2 = 8, P3 = 5, P4 = 7

Average: (8+8+5+7)/4 = 7

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

	Avg. Turnaround	Avg. Wait Time
RR $q=1$	9.75	5.75
RR $q=3$	11	7
RR $q=4$	9	5
RR $q=7$	8.75	4.75

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

(CST=1)	Avg. Turnaround	Avg. Wait Time
RR $q=1$	20.25	13.25
RR $q=3$	16.25	11.25
RR $q=4$	11.50	7.25
RR $q=7$	10.25	6.25

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

(CST=1)	Throughput	Utilization
RR q=1	0.125	0.500
RR q=3	0.167	0.667
RR q=4	0.190	0.762
RR q=7	0.200	0.800

q large ⇒ FIFO

q small ⇒ big CST overhead

generally, try to tune *q* to help tune responsiveness (i.e., of *interactive* processes)

may use:

- predetermined max response threshold
- median of EMAs
- process profiling

RR permits CPU-hungry jobs to run periodically, but prevents them from monopolizing the system (compare to FCFS and SJF)

… but also introduces *inflexible systemic overhead*: constant context switching

Fairness is overrated!

Can exercise more *fine-grained* control by introducing a system of *arbitrary priorities*

- computed and assigned to jobs dynamically by scheduler
- highest (current) priority goes next

SJF is an example of a priority scheduler!

- jobs are weighted using a burst-length prediction algorithm (e.g., EMA)
- priorities may vary over job lifetimes

Recall: SJF is prone to *starvation*

Common issue for priority schedulers

- combat with *priority aging*

4. **H**ighest **P**enalty **R**atio **N**ext

- example of a priority scheduler that uses aging to avoid starvation

Two statistics maintained for each job:

1. total CPU execution time, t
2. “wall clock” age, T

Priority, "penalty ratio" = T / t

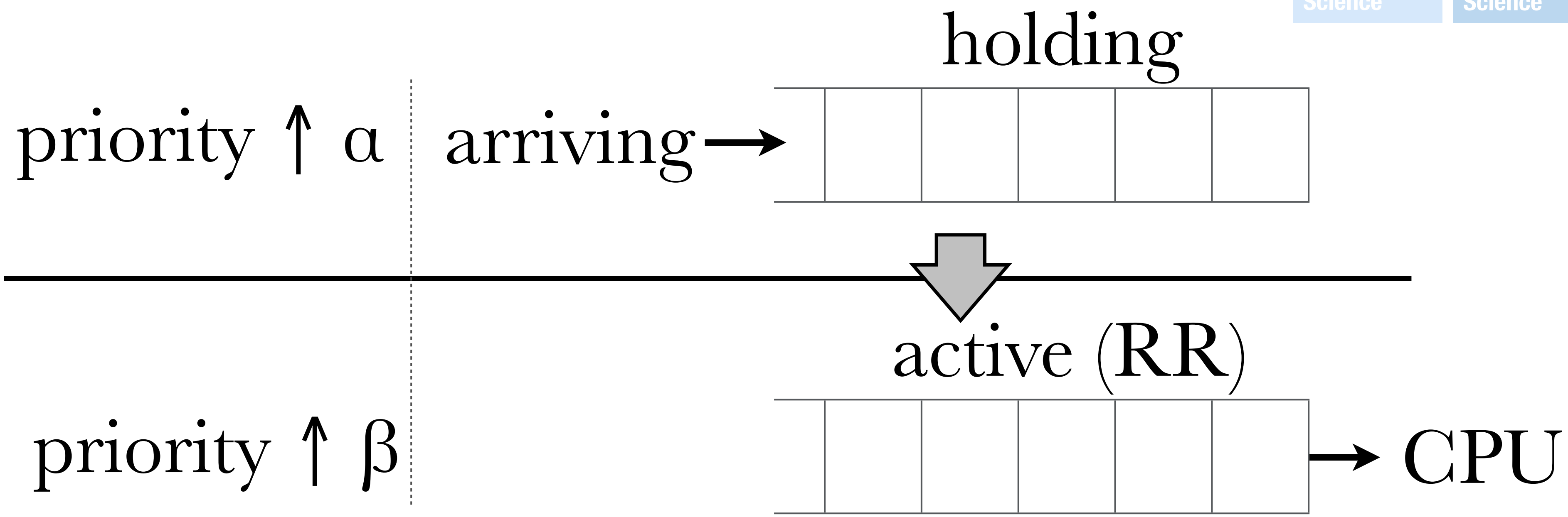
- ∞ when job is first ready
- decreases as job receives CPU time

HPRN in practice would incur too many context switches (due to very short bursts)

— likely institute minimum burst quanta

5. Selfish RR

- example of a more sophisticated priority based scheduling policy



$\beta = 0$: RR

$\beta \geq (\alpha \neq 0)$: FCFS

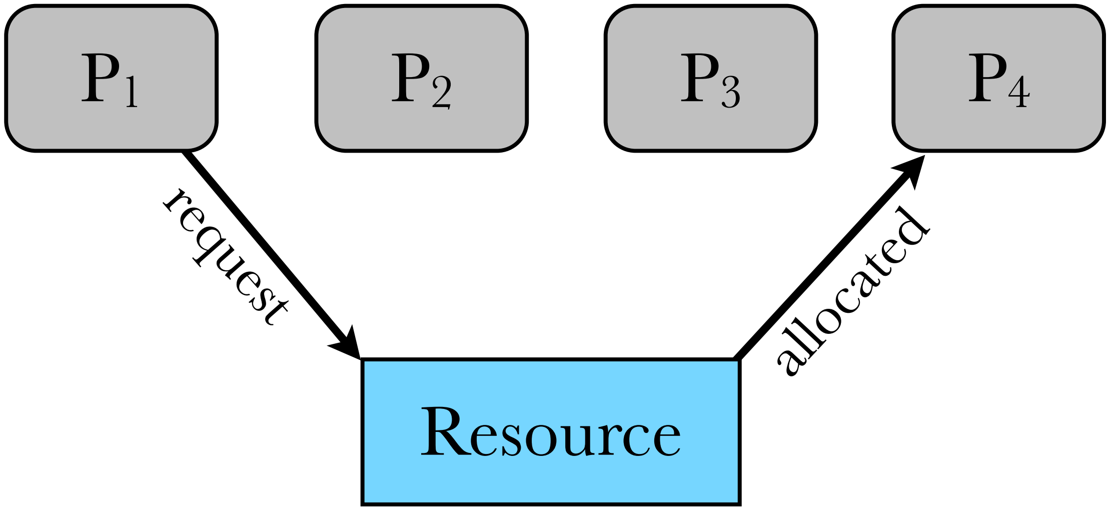
$\beta > (\alpha = 0)$: RR in batches

$\alpha > \beta > 0$: “Selfish” (ageist) RR

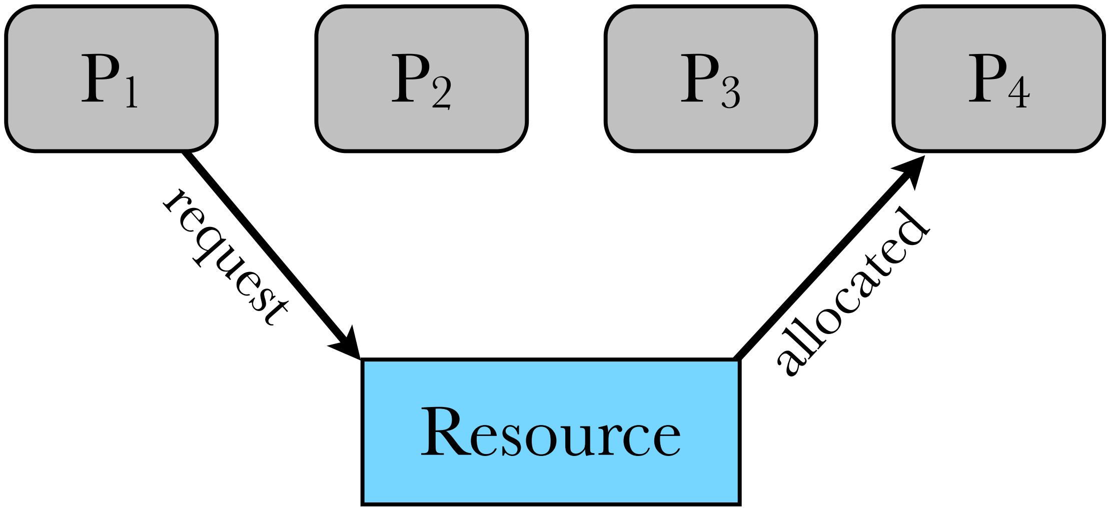
Another problem (on top of starvation) possibly created by priority-based scheduling policies: *priority inversion*

Process	Priority	State
P_1	High	Ready
P_2	Mid	Ready
P_3	Mid	Ready
P_4	Low	Ready

Process	Priority	State
P_1	High	*Running*
P_2	Mid	Ready
P_3	Mid	Ready
P_4	Low	Ready

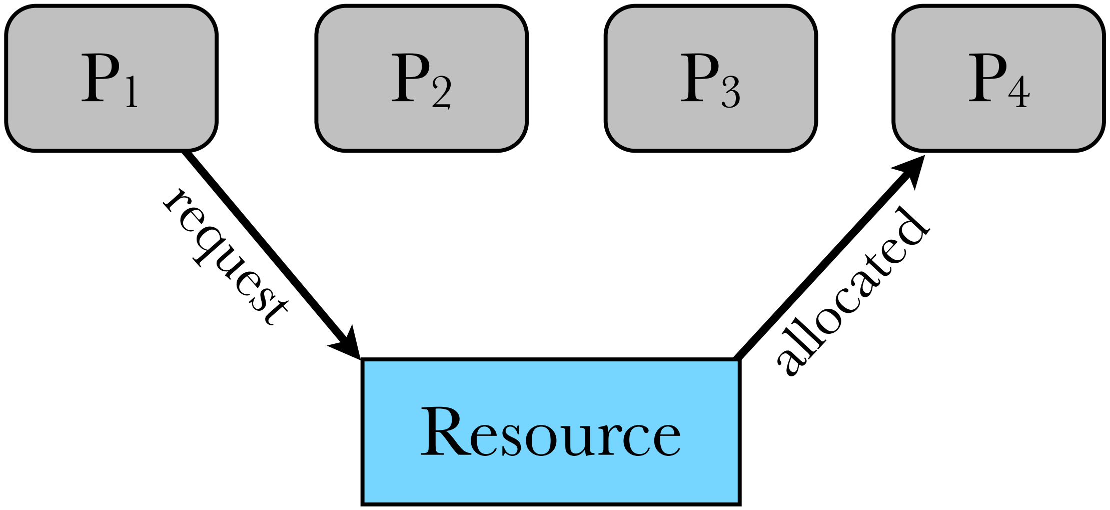


Process	Priority	State
P_1	High	***Blocked***
P_2	Mid	Ready
P_3	Mid	Ready
P_4	Low	Ready



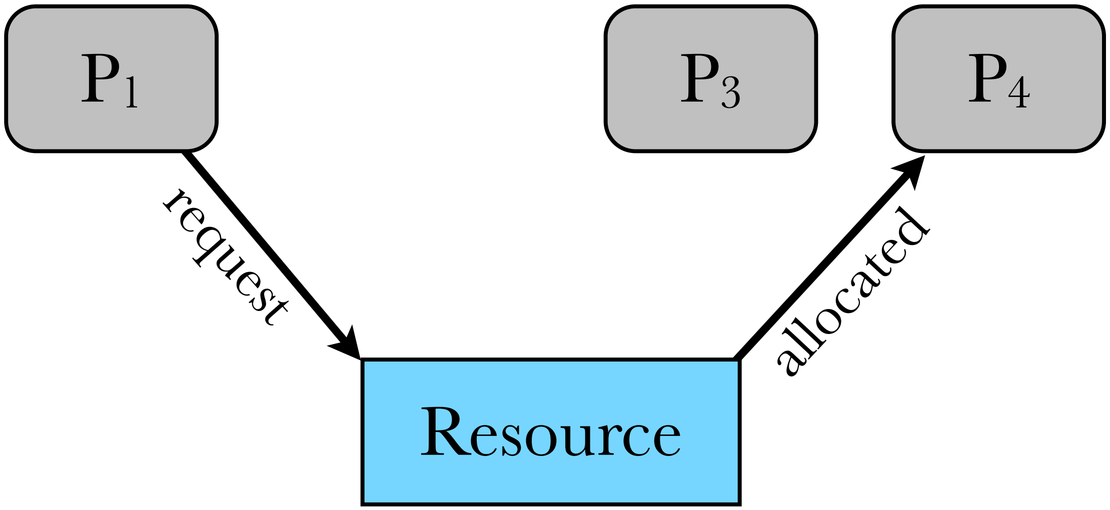
(*mutually exclusive* allocation)

Process	Priority	State
P_1	High	***Blocked***
P_2	Mid	*Running*
P_3	Mid	Ready
P_4	Low	Ready



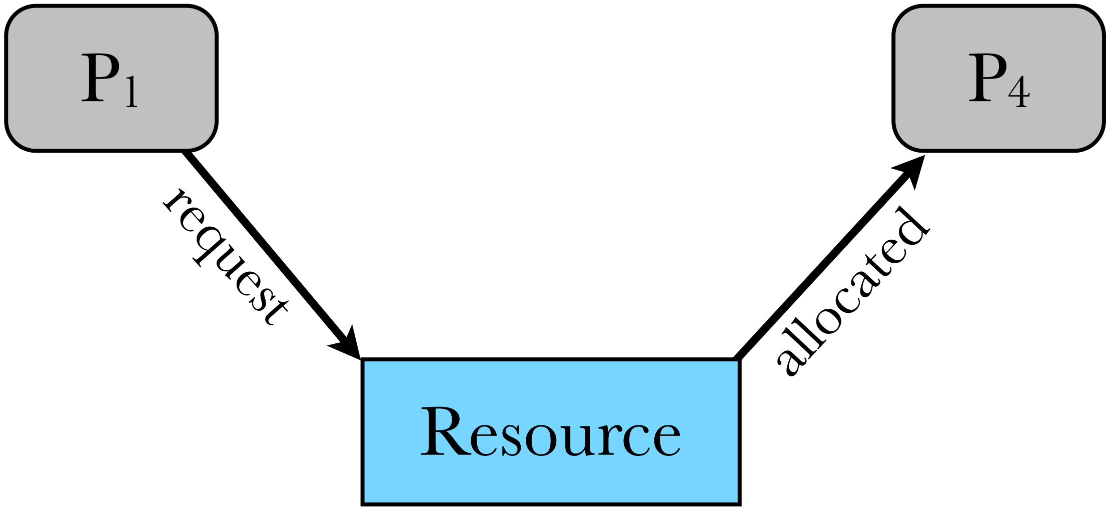
(*mutually exclusive* allocation)

Process	Priority	State
P_1	High	***Blocked***
~~P_2~~	~~Mid~~	~~Done~~
P_3	Mid	*Running*
P_4	Low	Ready



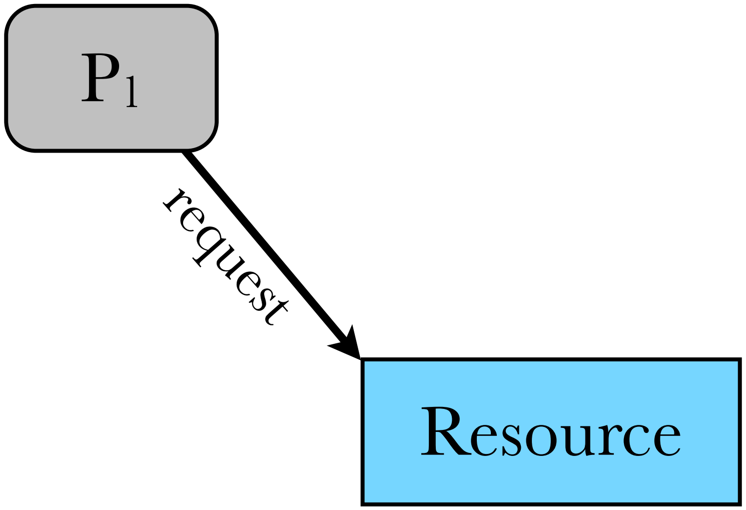
(*mutually exclusive* allocation)

Process	Priority	State
P_1	High	***Blocked***
~~P_2~~	~~Mid~~	~~Done~~
~~P_3~~	~~Mid~~	~~Done~~
P_4	Low	*Running*



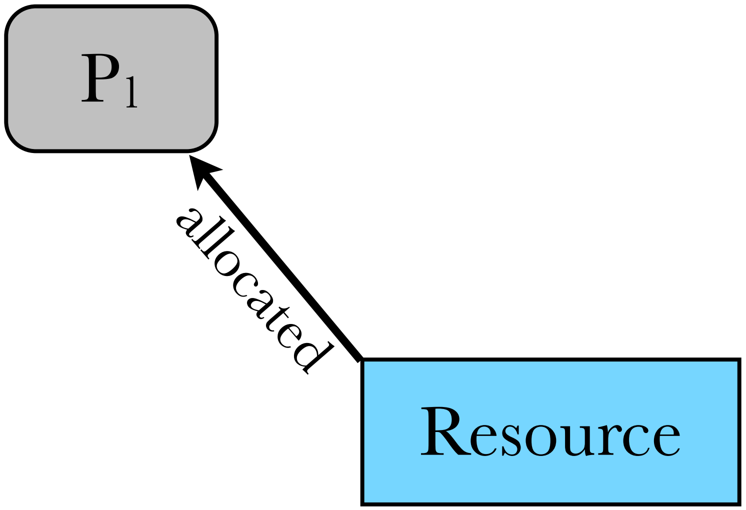
(*mutually exclusive* allocation)

Process	Priority	State
P_1	High	***Blocked***
~~P_2~~	~~Mid~~	~~Done~~
~~P_3~~	~~Mid~~	~~Done~~
~~P_4~~	~~Low~~	~~Done~~



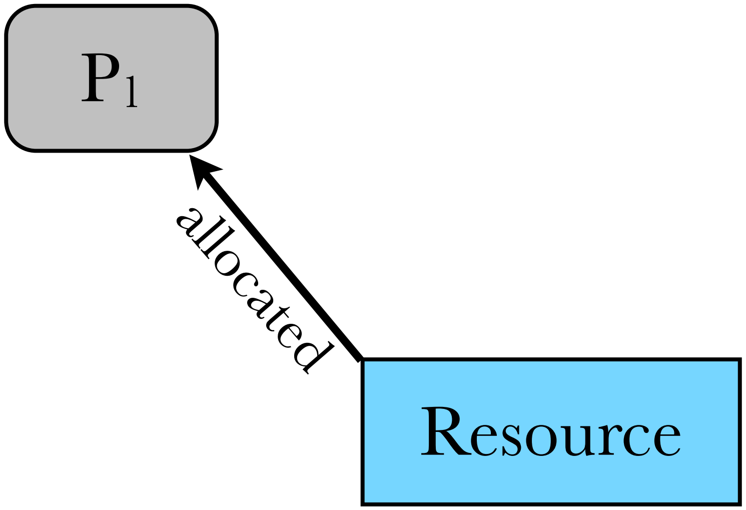
(*mutually exclusive* allocation)

Process	Priority	State
P_1	High	Ready
~~P_2~~	~~Mid~~	~~Done~~
~~P_3~~	~~Mid~~	~~Done~~
~~P_4~~	~~Low~~	~~Done~~



(*mutually exclusive* allocation)

Process	Priority	State
P_1	High	*Running*
~~P_2~~	~~Mid~~	~~Done~~
~~P_3~~	~~Mid~~	~~Done~~
~~P_4~~	~~Low~~	~~Done~~

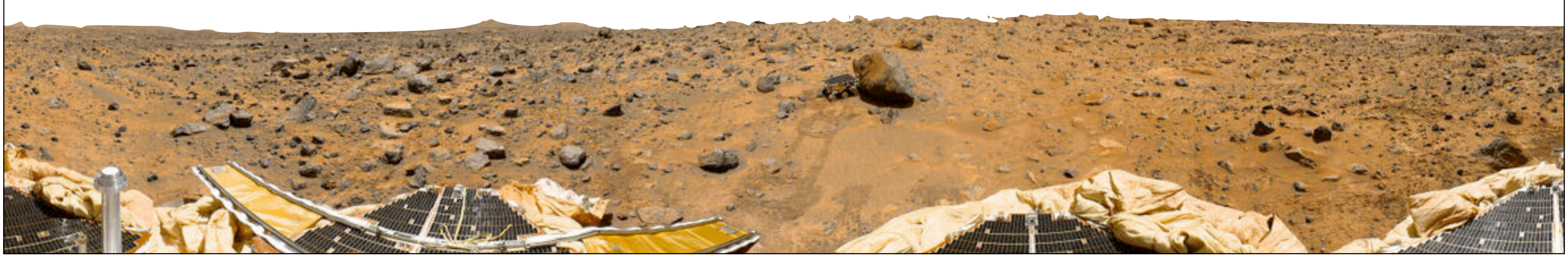


(*mutually exclusive* allocation)

priority inversion: a high priority job effectively takes on the priority of a lower-level one that holds a required resource

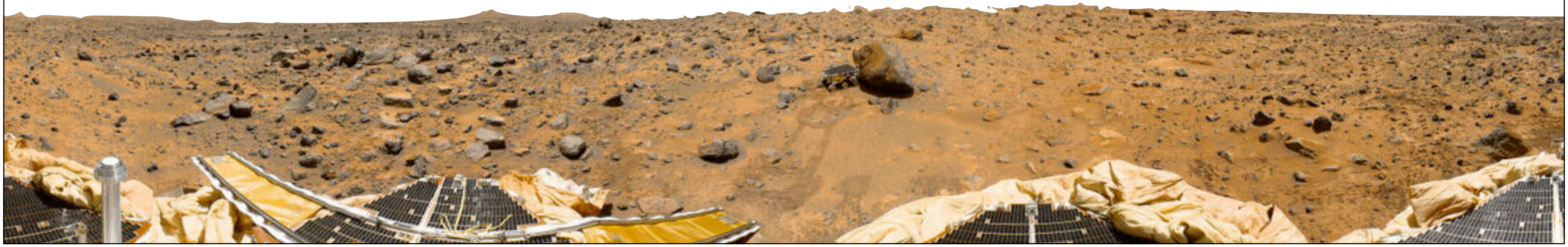
high profile case study: NASA Pathfinder

- spacecraft developed a recurring system failure/reset
  - occurred after deploying data-gathering robot to surface of Mars



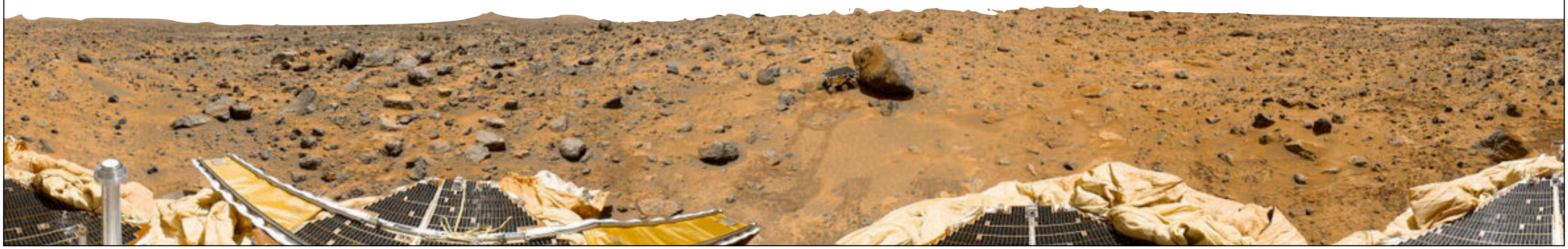
culprits:

- flood of meteorological data
  - low priority of related job: ASI/MET
- a shared, mutually exclusive resource (semaphore guarding an IPC pipe)



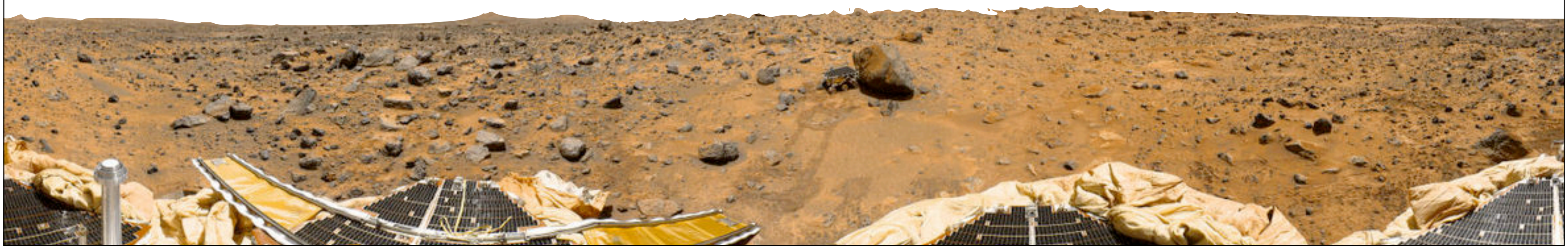
high priority job (for data aggregation & distribution) — bc_dist — required pipe

- but always held by ASI/MET
  - in turn kept from running by various mid-priority jobs



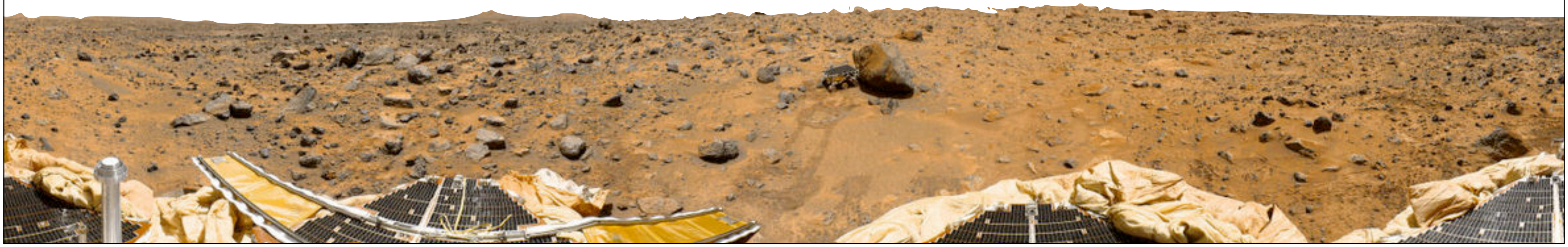
scheduling job determined that bc_dist couldn’t complete per hard deadline

- declared error resulting in system reset!
- re-produced in lab after 18-hours of simulating spacecraft activities



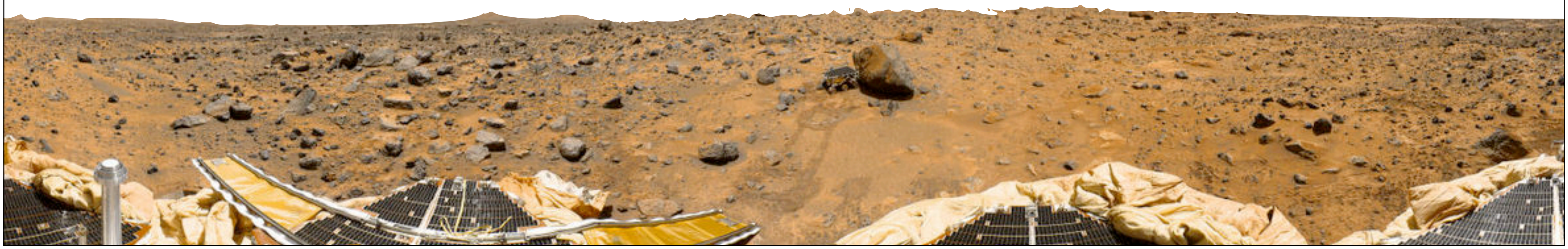
fix: *priority inheritance*

- job that blocks a higher priority job will inherit the latter's priority
  - e.g., run ASI/MET at bc_dist's priority until resource is released



how?

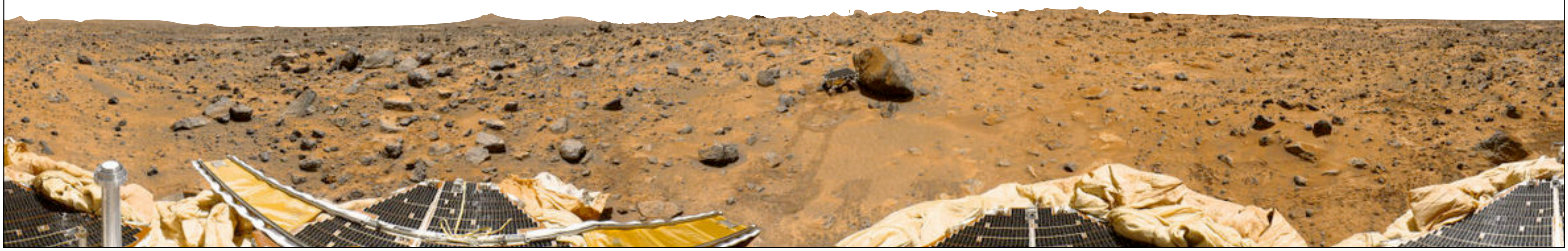
- enabling priority inheritance via semaphores (in vxWorks OS)
  - (why wasn't it on by default?)
- prescient remote (!) tracing & patching facilities built in to system



why did NASA not foresee this?

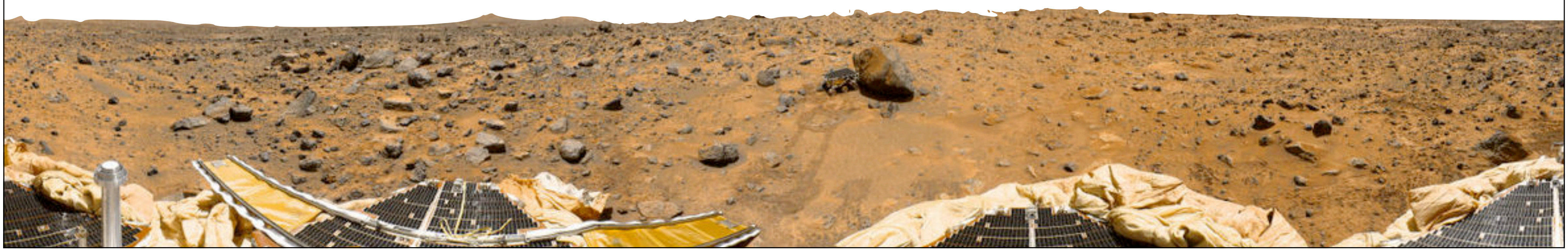
> *"Our before launch testing was limited to the "best case" high data rates and science activities… We did not expect nor test the "better than we could have ever imagined" case."*

- Glenn Reeves
  Software team lead



takeaways:

- scheduling bugs are hard to predict, track down, and fix
- priority inheritance provides a “solution” for priority inversion
- scheduling *is* rocket science!



questions:

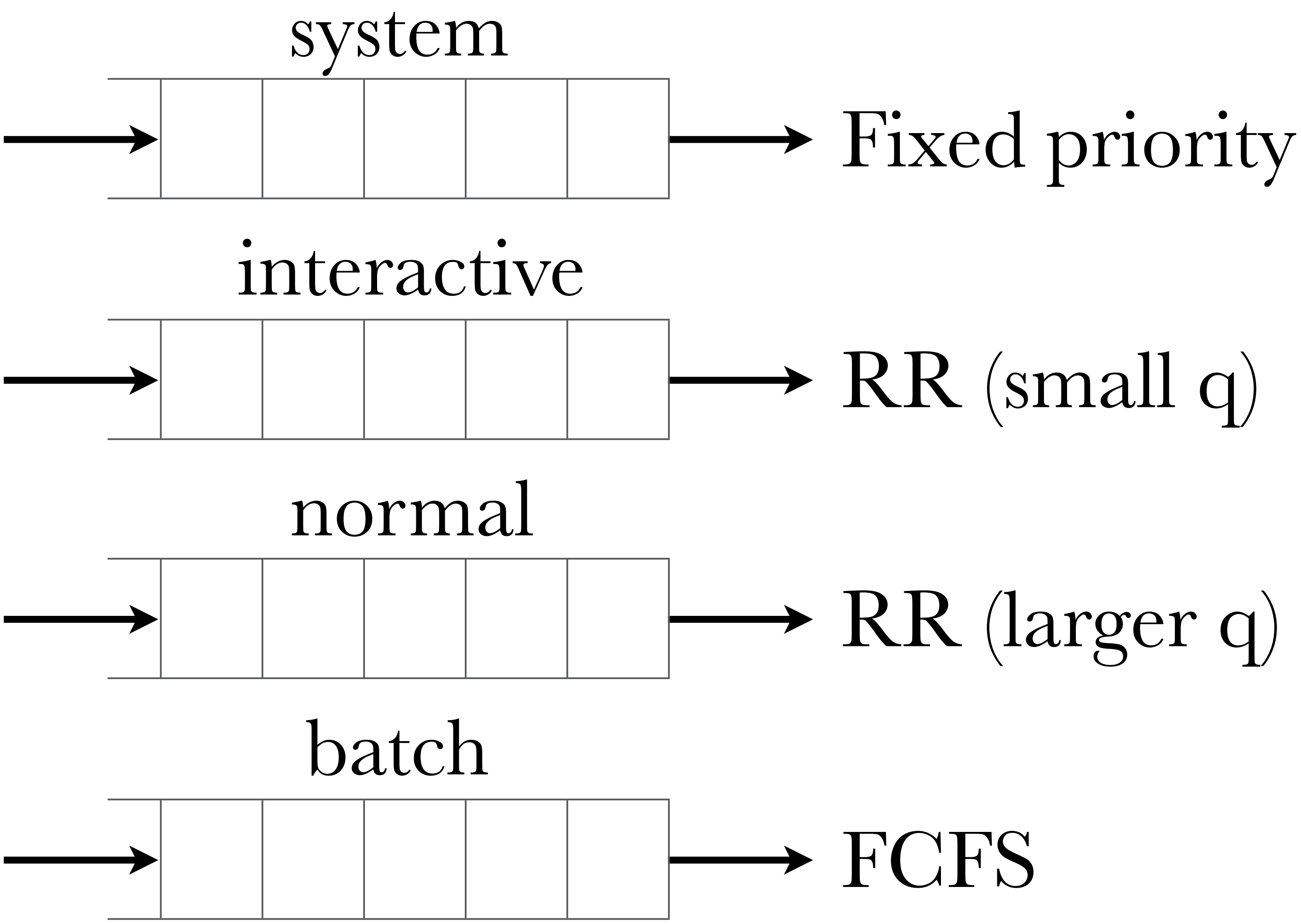
- w.r.t. priority inheritance:
  - pros/cons?
  - how to implement?
- w.r.t. priority inversion:
  - detection? how else to “fix”?
  - effect on non-real-time OS?

Even with the fine-grained control offered by a priority scheduler, hard to impose different *sets of goals* on *groups* of jobs

E.g., top-priority for system jobs, RR for interactive jobs, FCFS for batch jobs

6. Multi-Level Queue (MLQ)

- disjoint ready queues
- separate schedulers/policies for each



system

Fixed priority

interactive

RR (small q)

normal

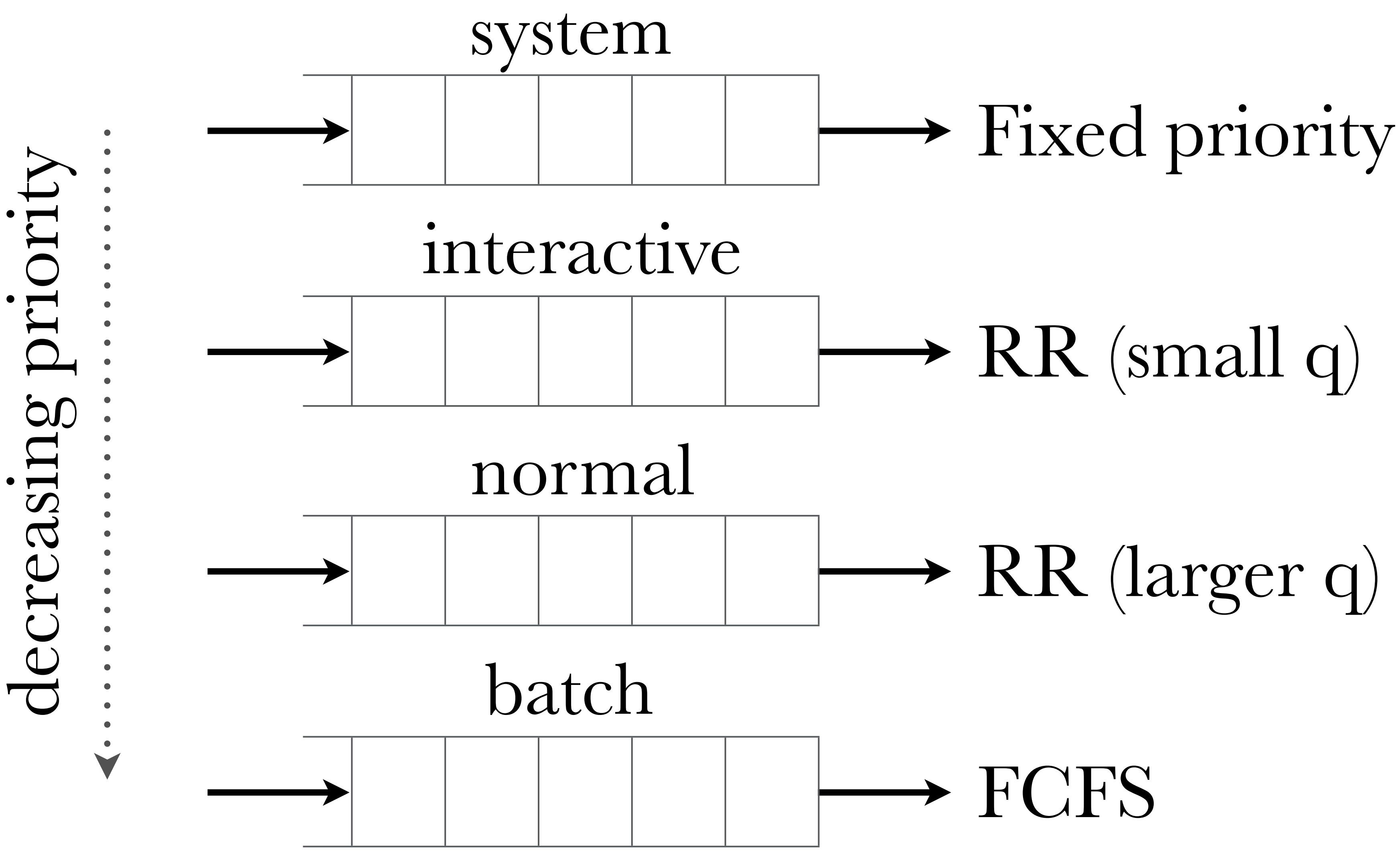
RR (larger q)

batch

FCFS

requires *queue arbitration* strategy in place

approach 1: prioritize top, non-empty queue



decreasing priority

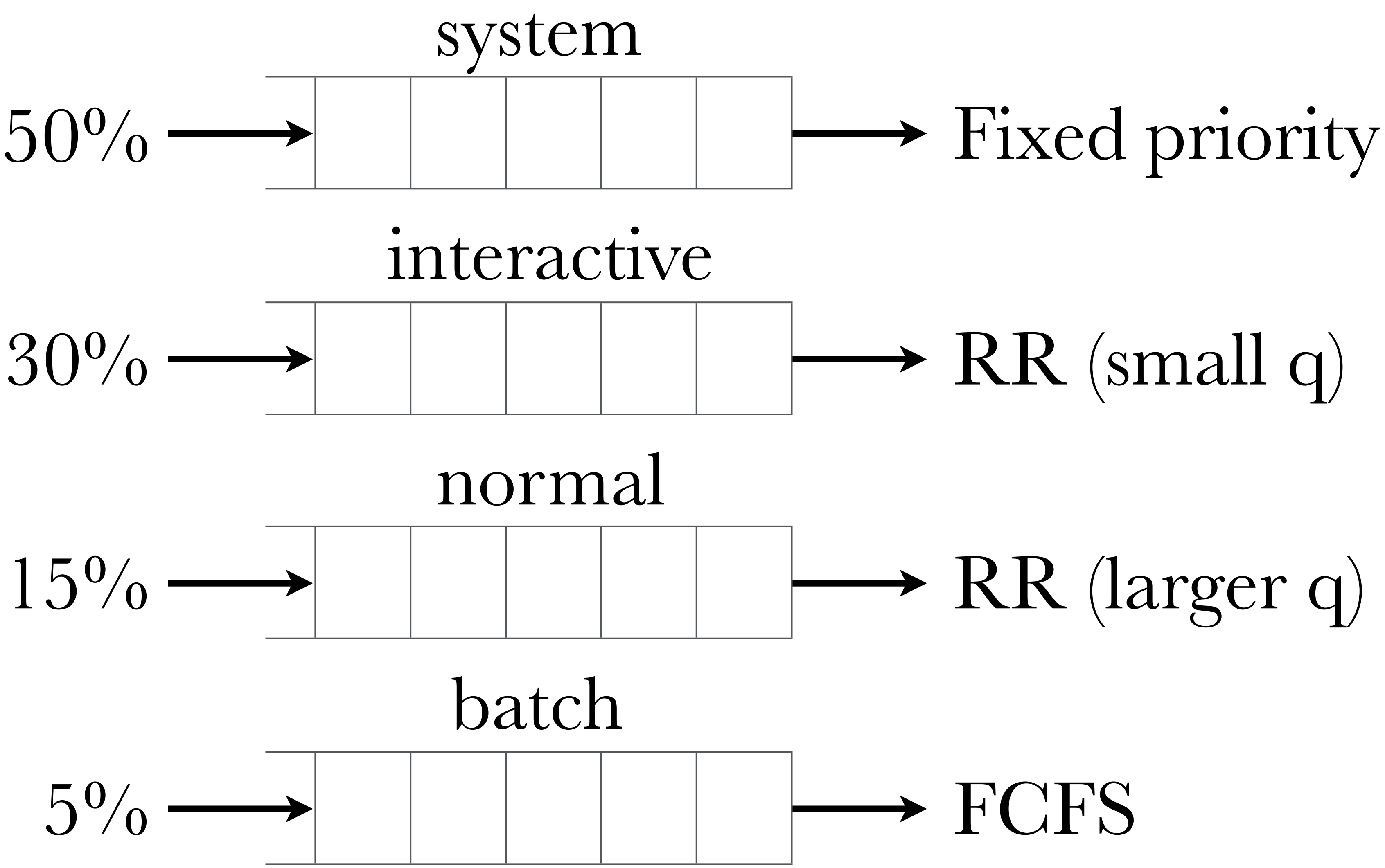
system → Fixed priority

interactive → RR (small q)

normal → RR (larger q)

batch → FCFS

approach 2: aggregate time slices



what processes go in which queues?

- self-assigned
  - e.g., UNIX "nice" value
- "profiling" based on initial burst(s)
  - CPU, I/O burst length
  - e.g., short, intermittent CPU bursts
    ⇒ classify as interactive job

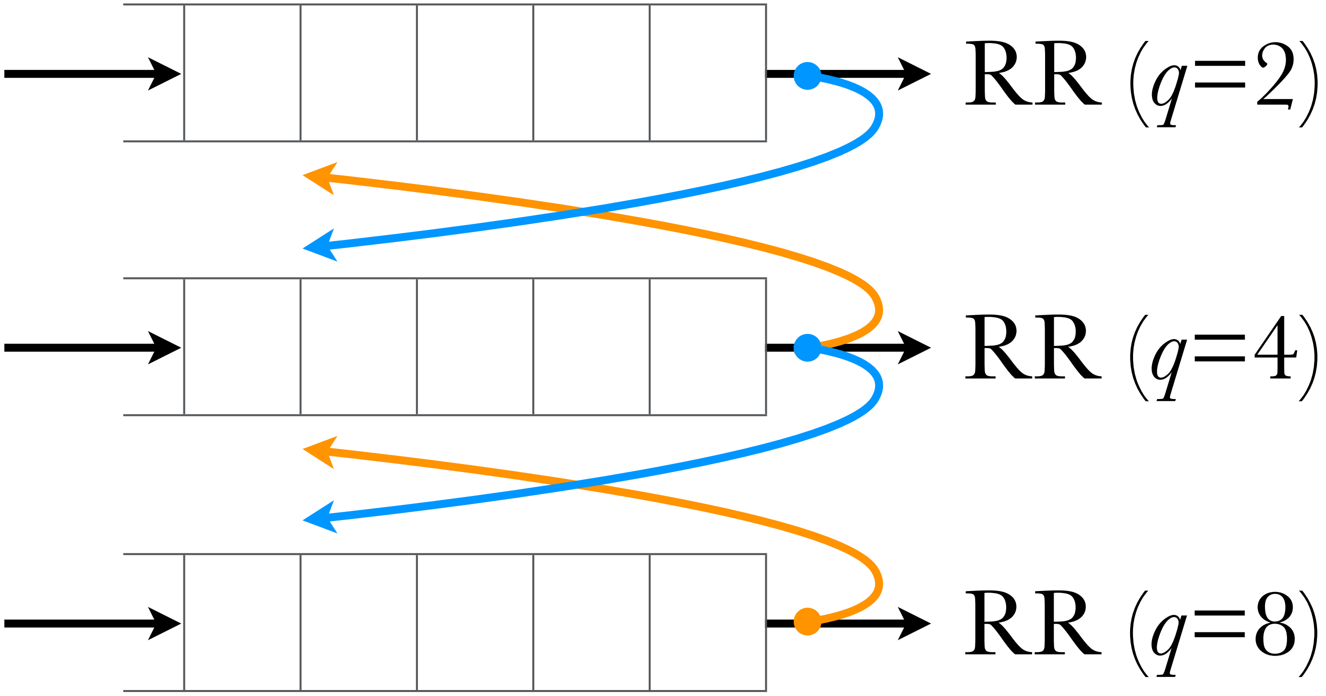
classification issue: what if process characteristics change *dynamically*?

- e.g., photo editor: tool selection (interactive) → apply filter (CPU hungry) → simple edits (interactive)

7. Multi-Level *Feedback* Queue

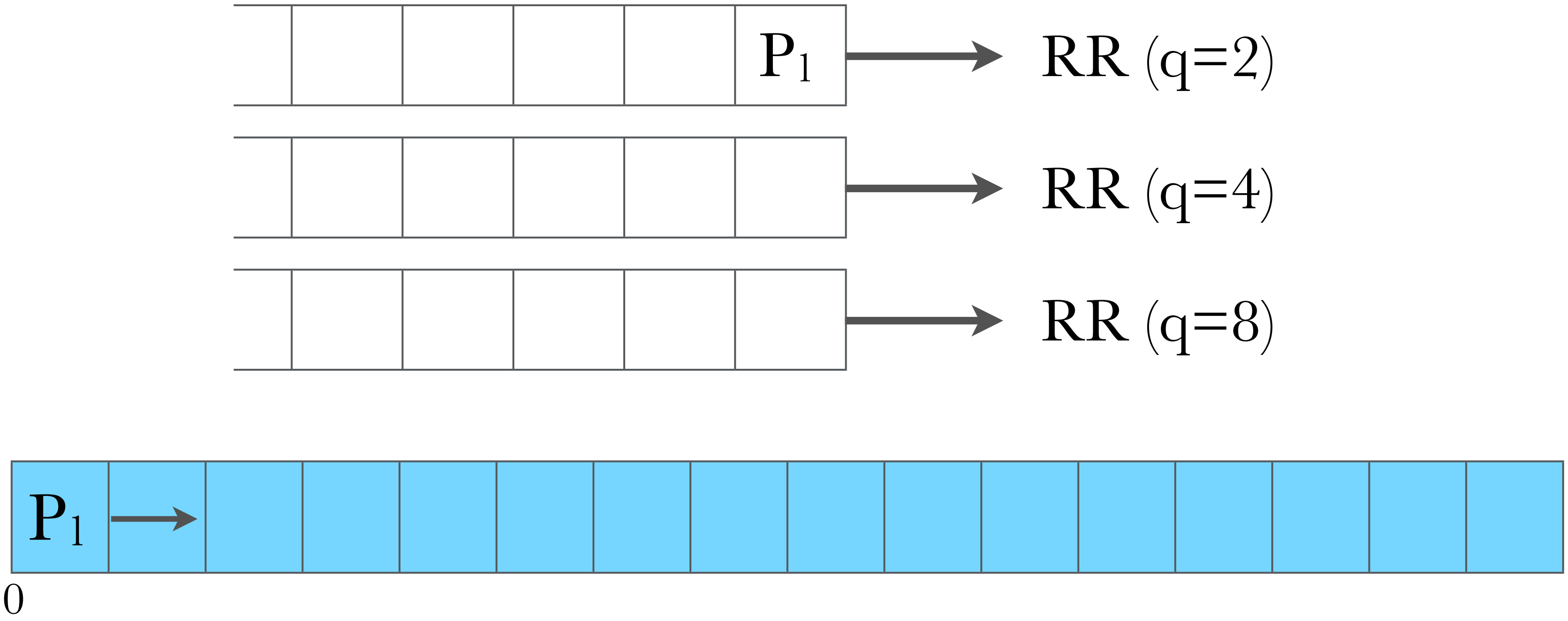
- supports movement between queues after initial assignment
  - based on ongoing job accounting

e.g., 3 RR queues with different q

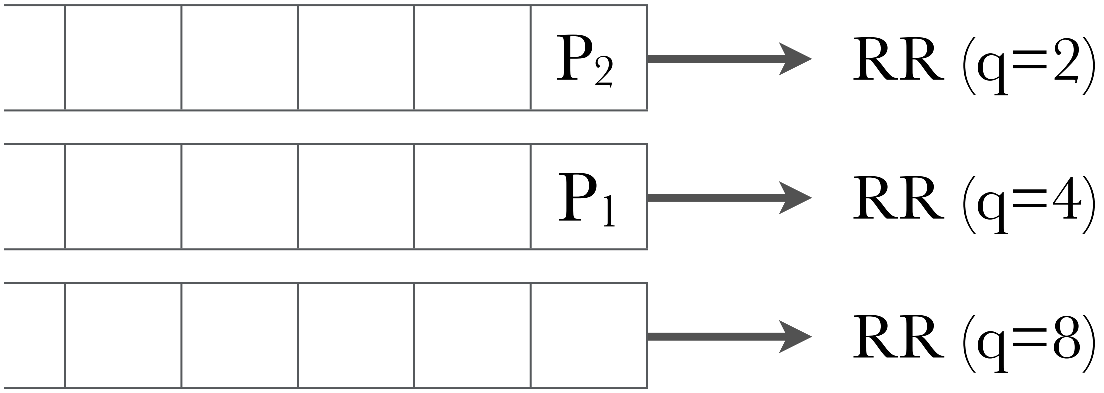


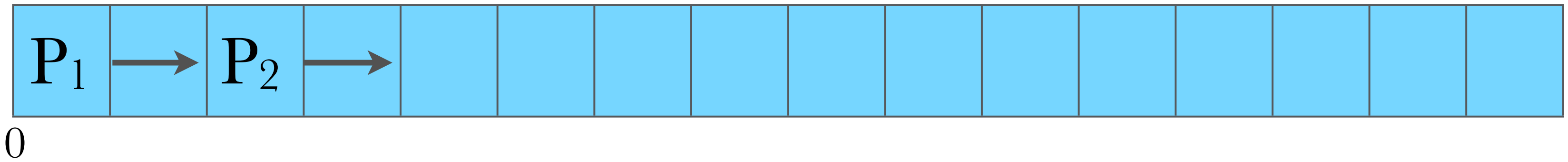
assignment based on q/burst-length fit

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

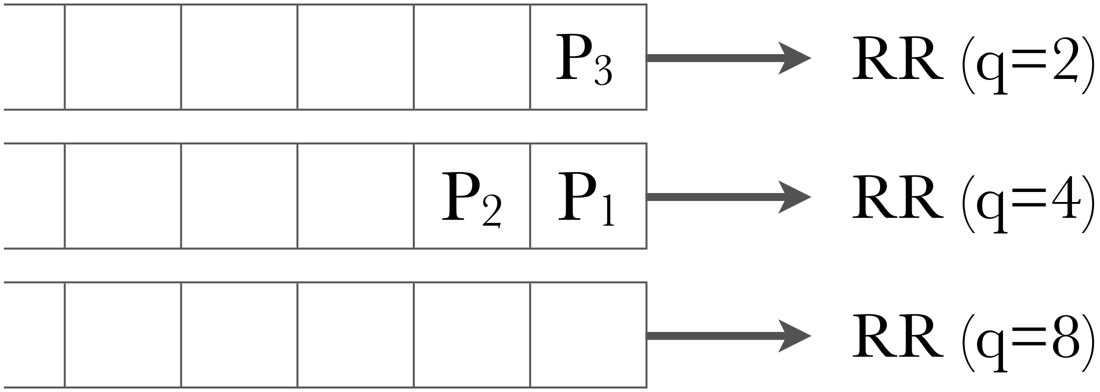


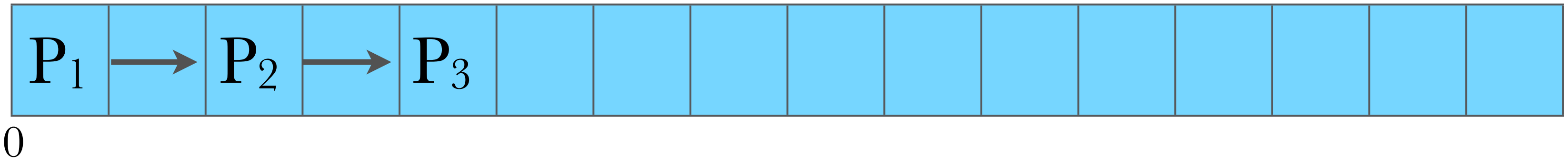
Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



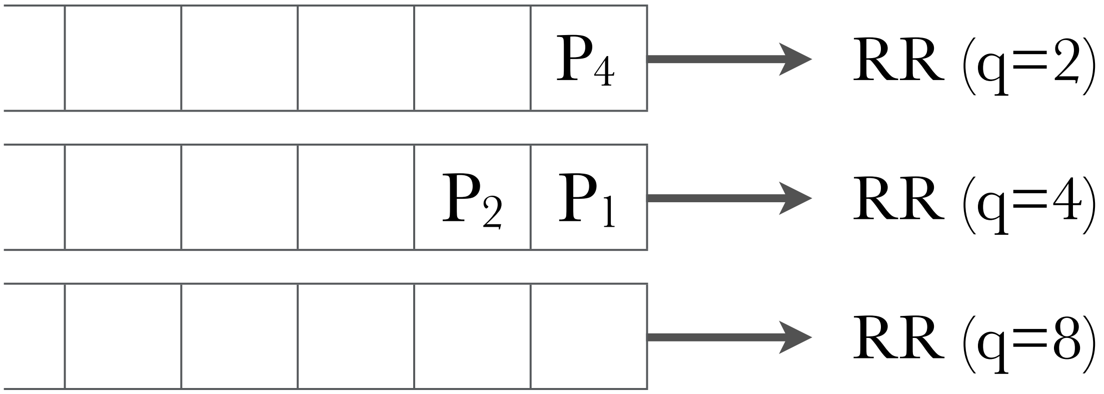


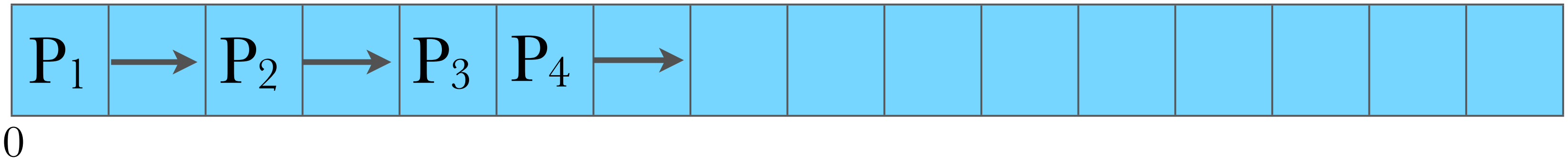
Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



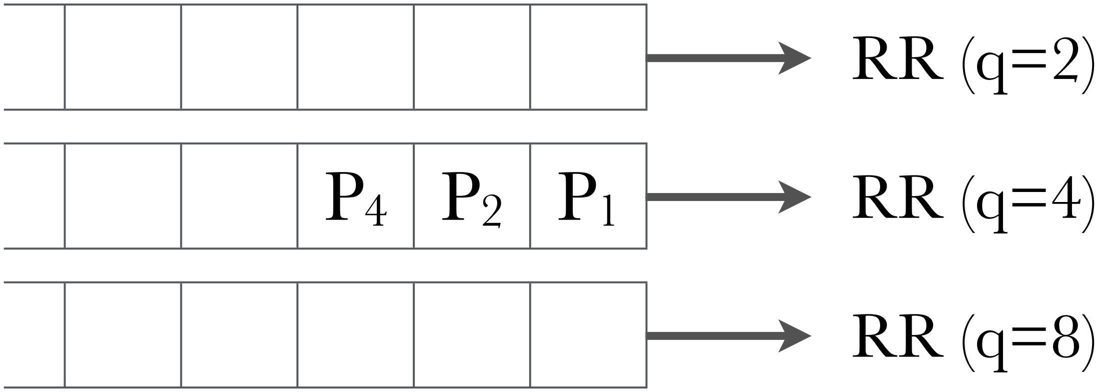


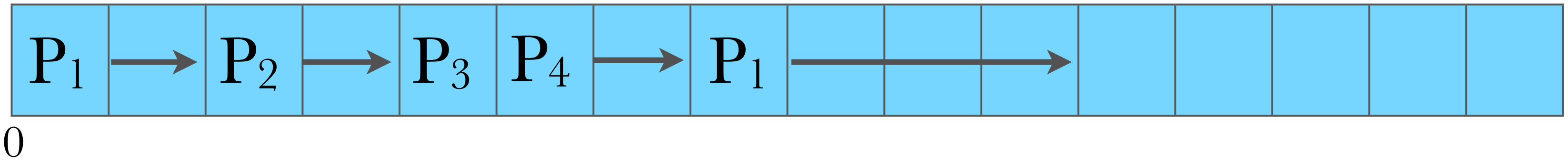
Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



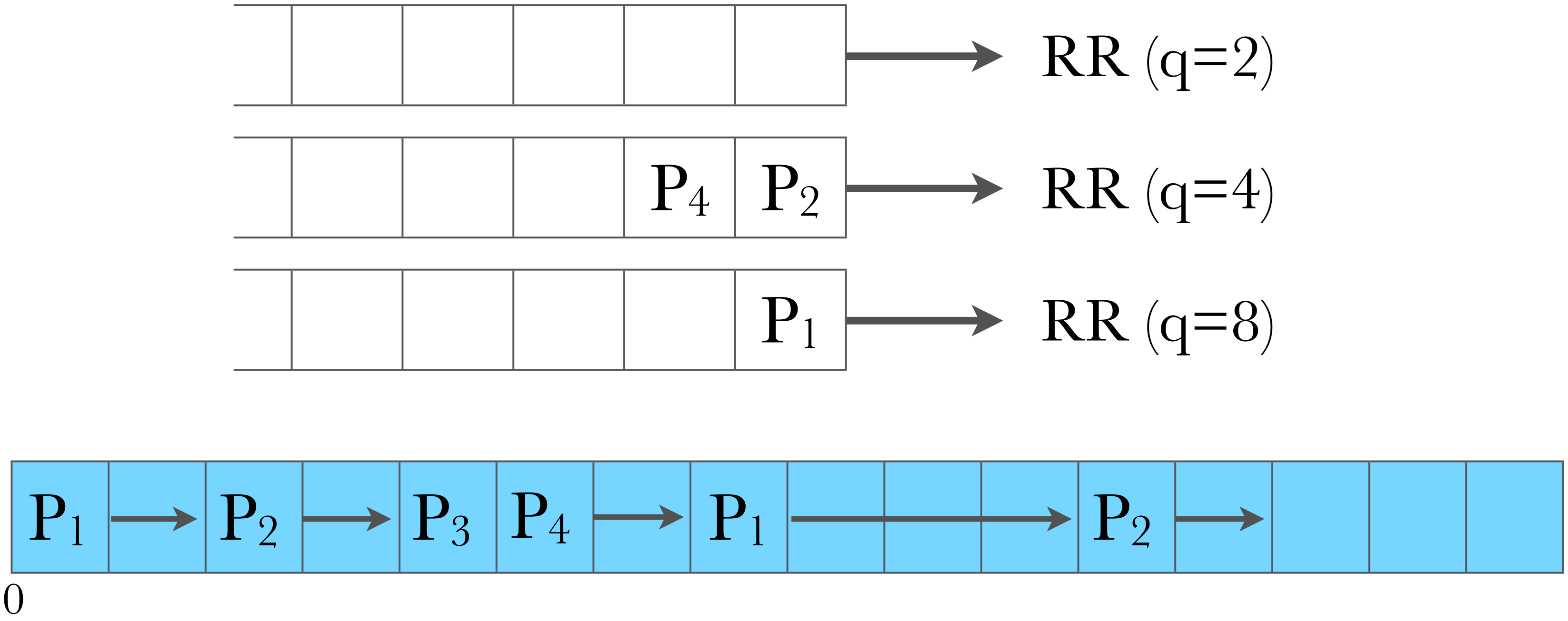


Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

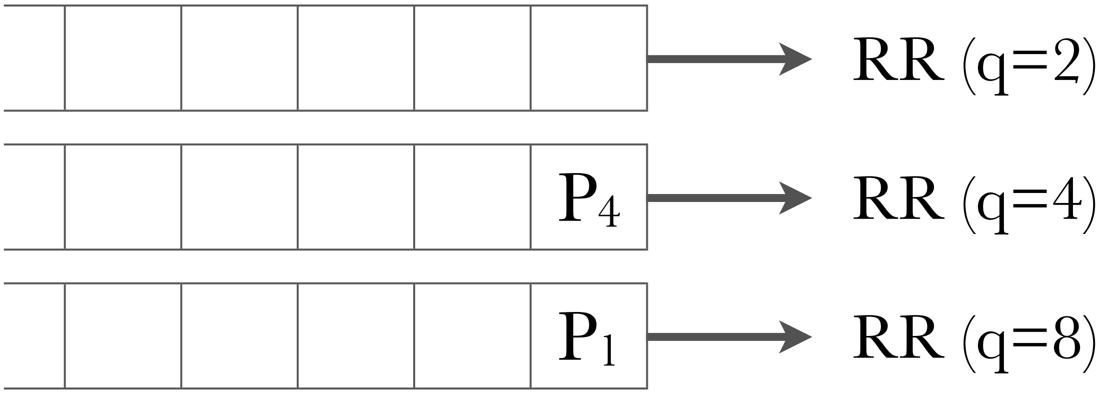


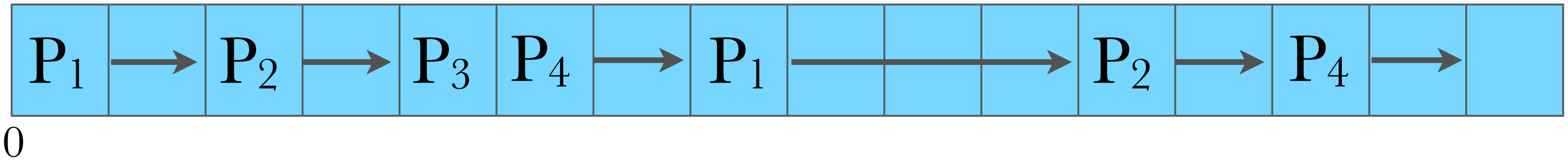


Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4



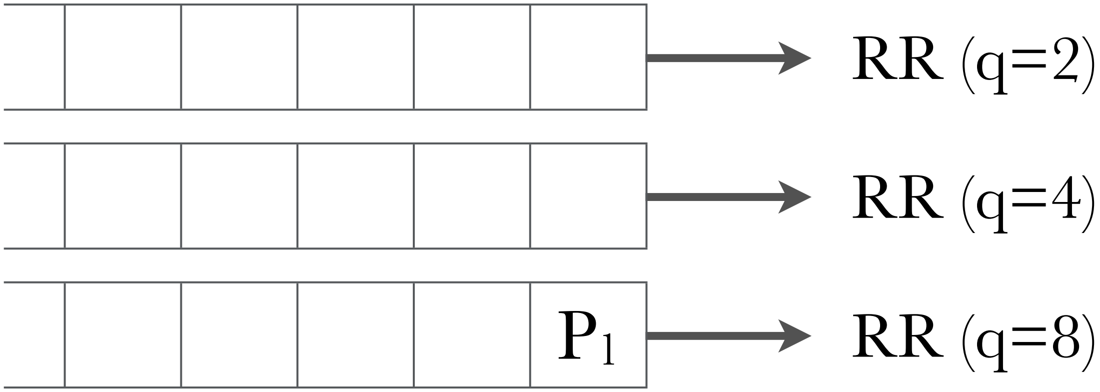
Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

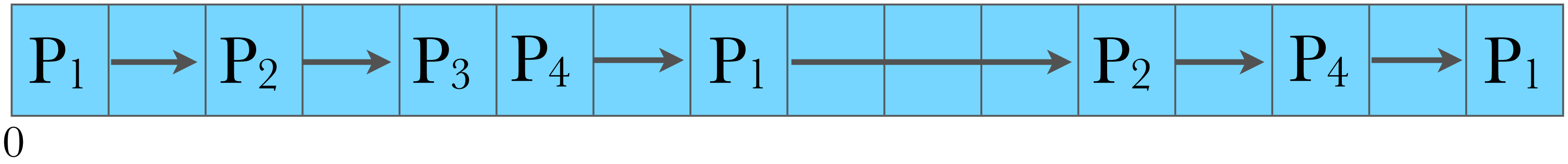




IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

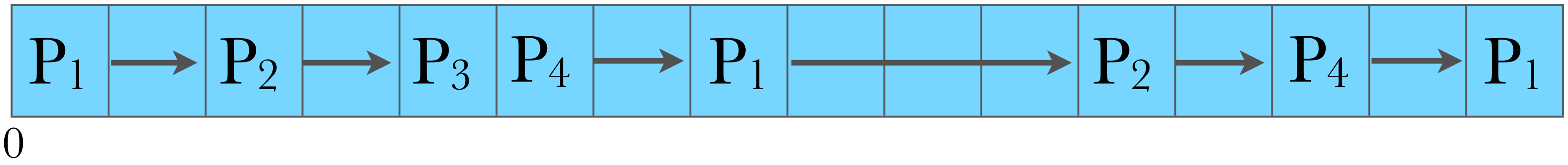




Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

Wait times: $P_1 = 9$, $P_2 = 7$, $P_3 = 0$, $P_4 = 6$

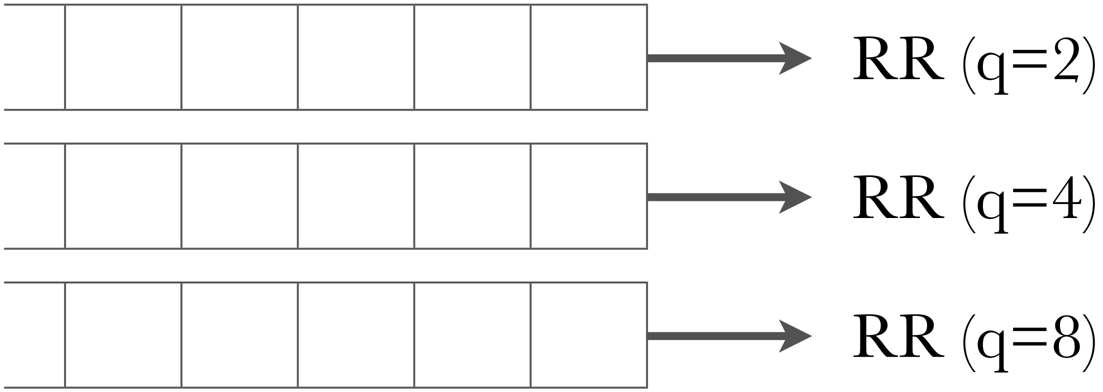
Average: $(9+7+0+6)/4 = 5.5$ *(vs 7 for RR, q=3)*

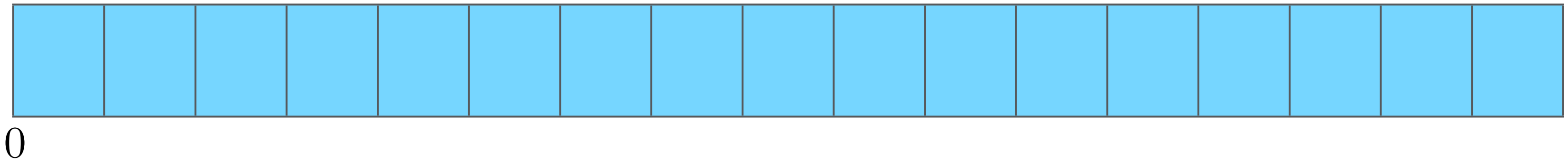


- following I/O, processes return to previously assigned queue
- when to move up?
  - for RR, when burst ≤ q

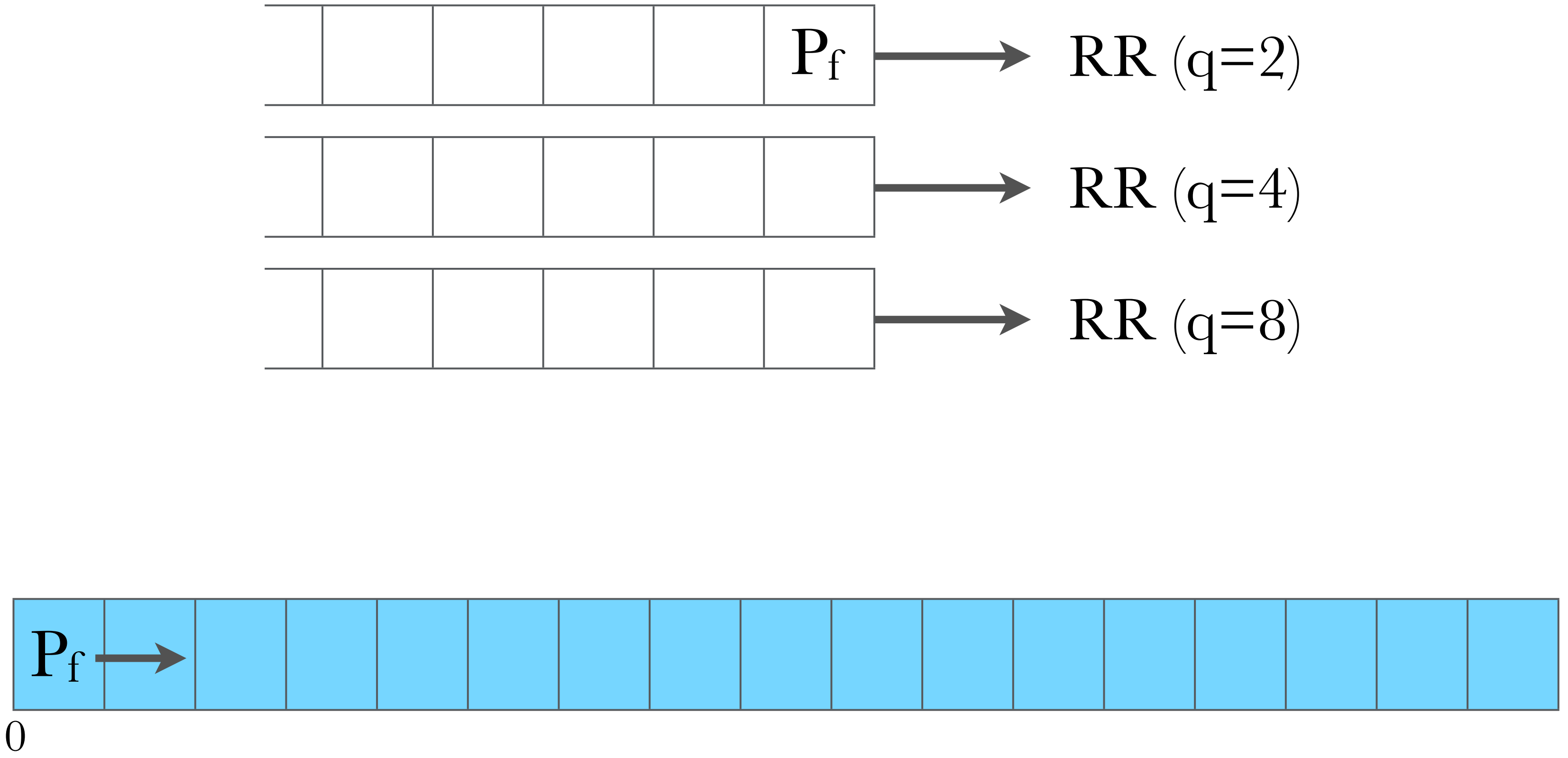
e.g., P_{flaky} arrives at t=0

CPU burst lengths = 7, 4, 1, 5 (I/O between)

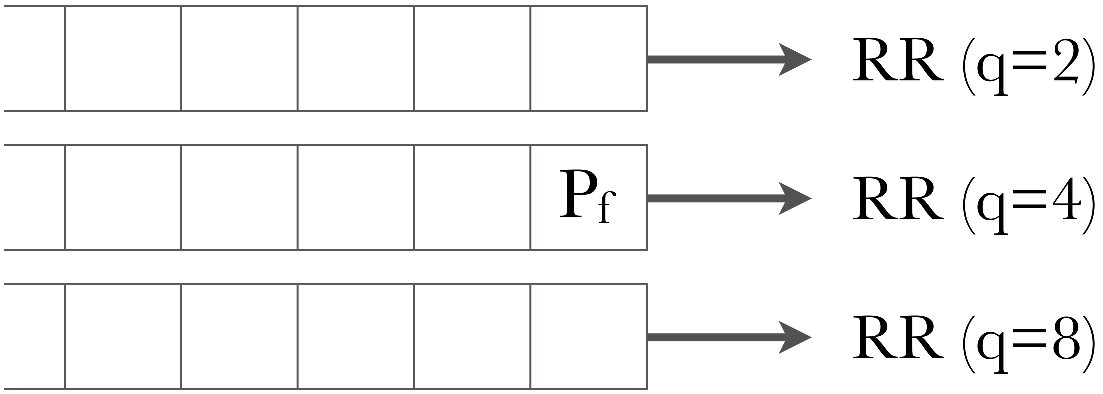


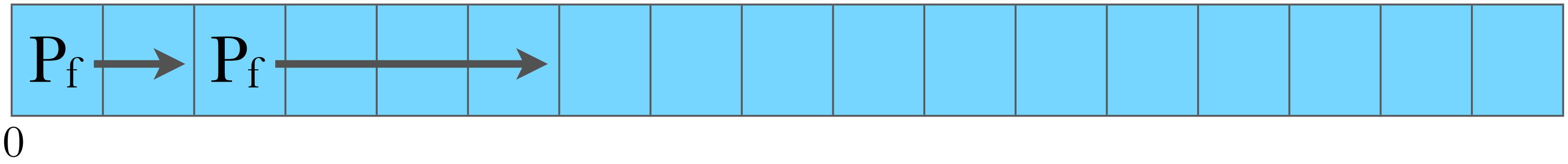


e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)

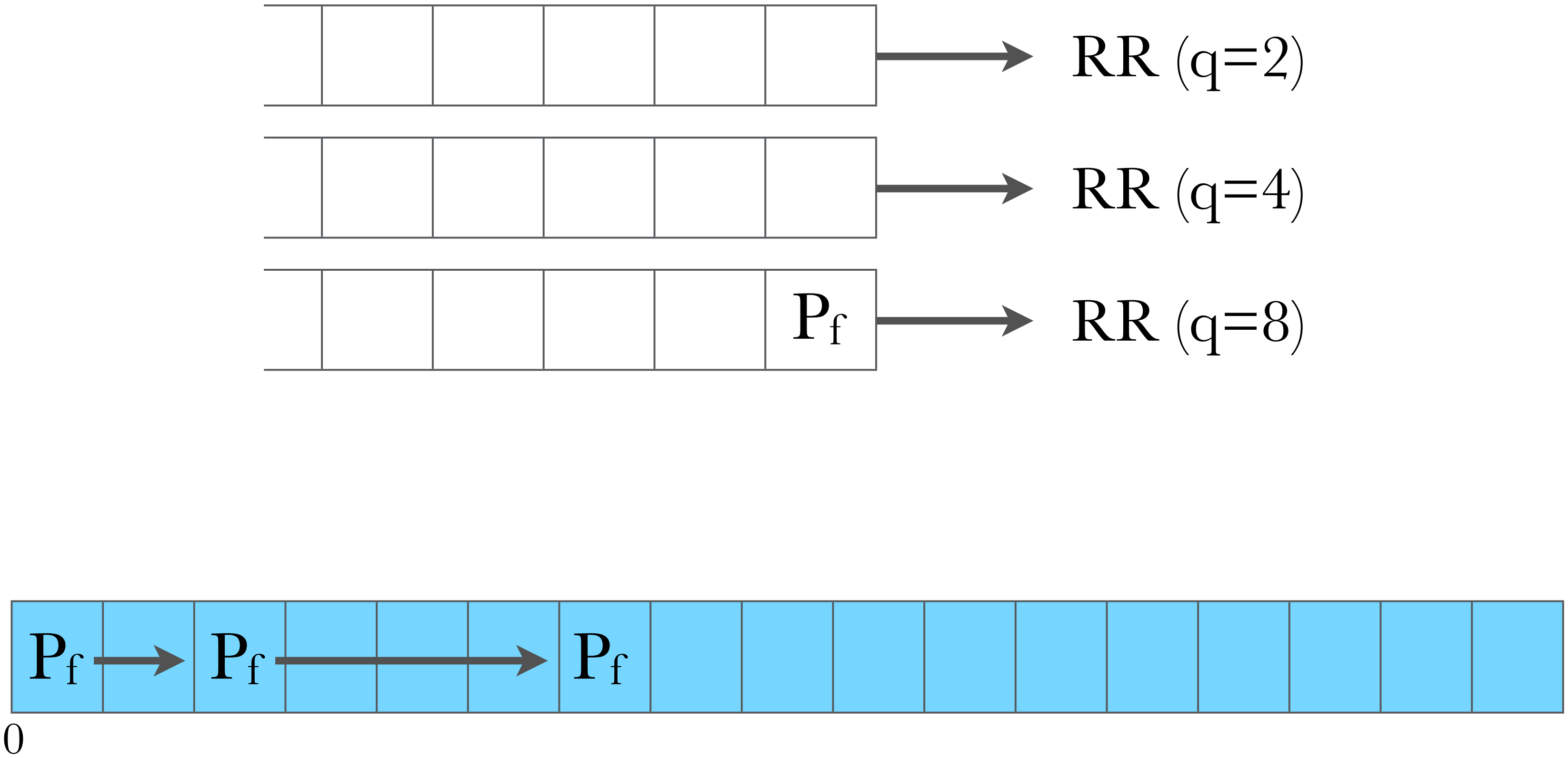


e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)

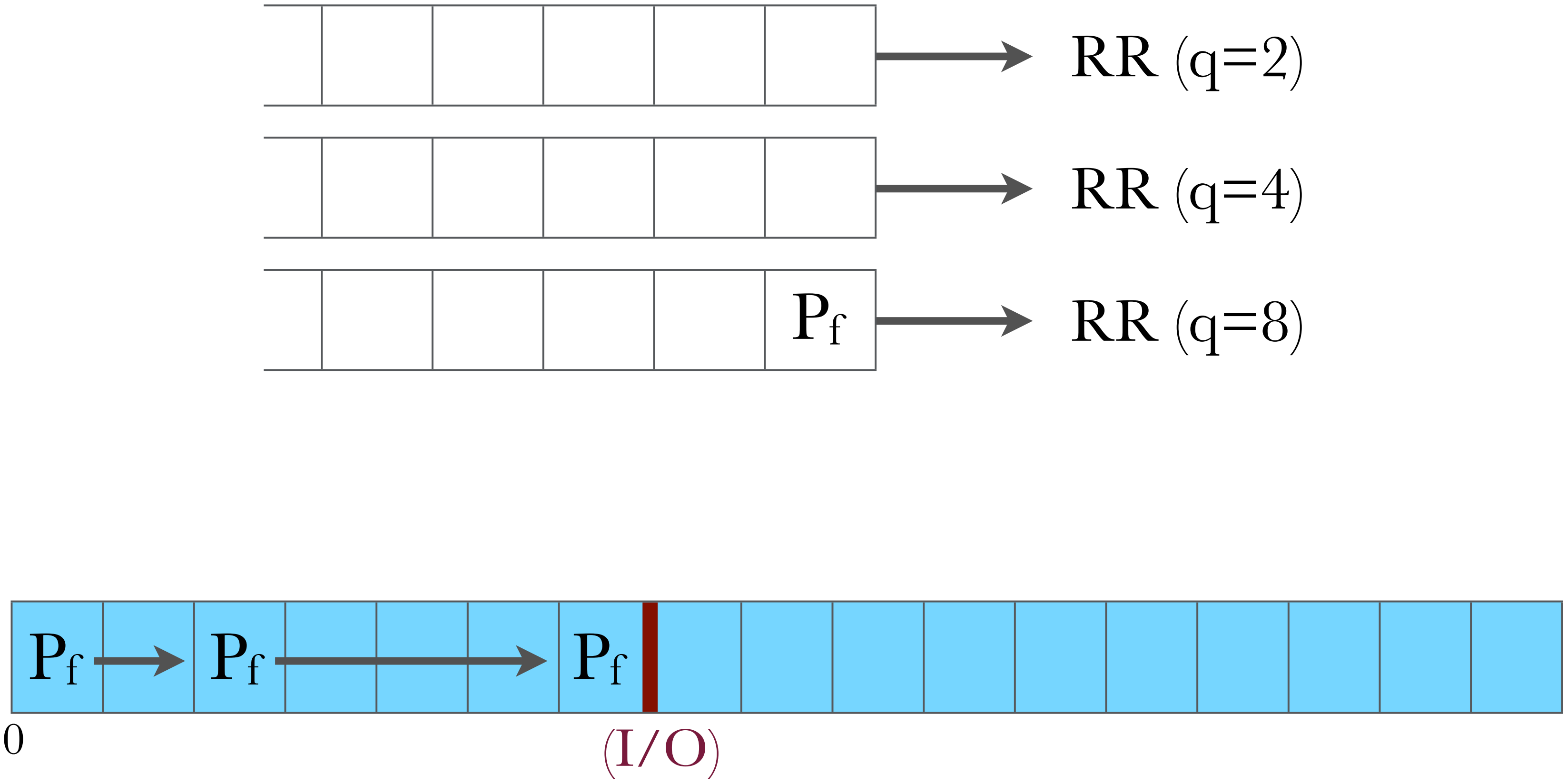




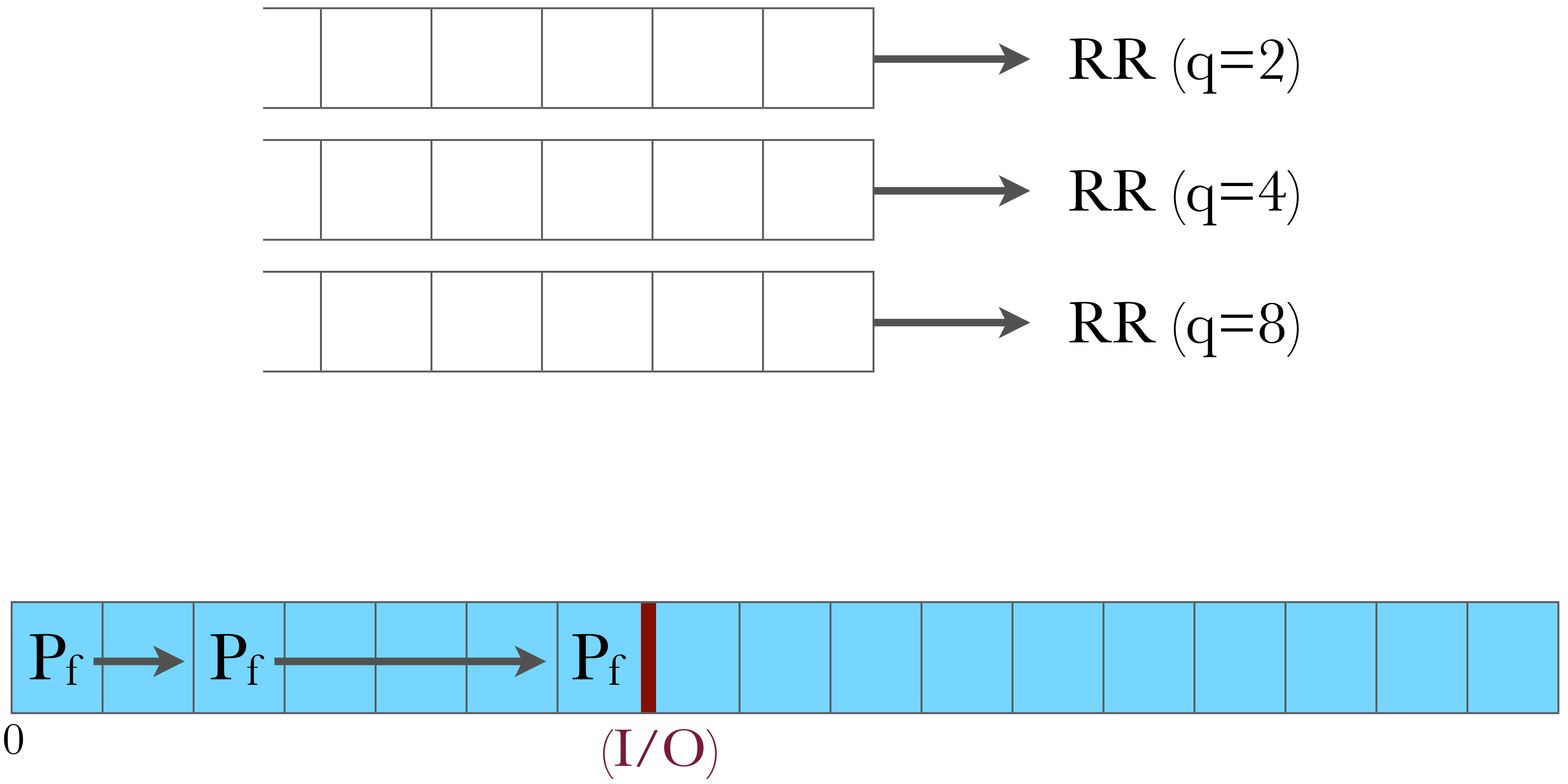
e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)



e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)



e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)

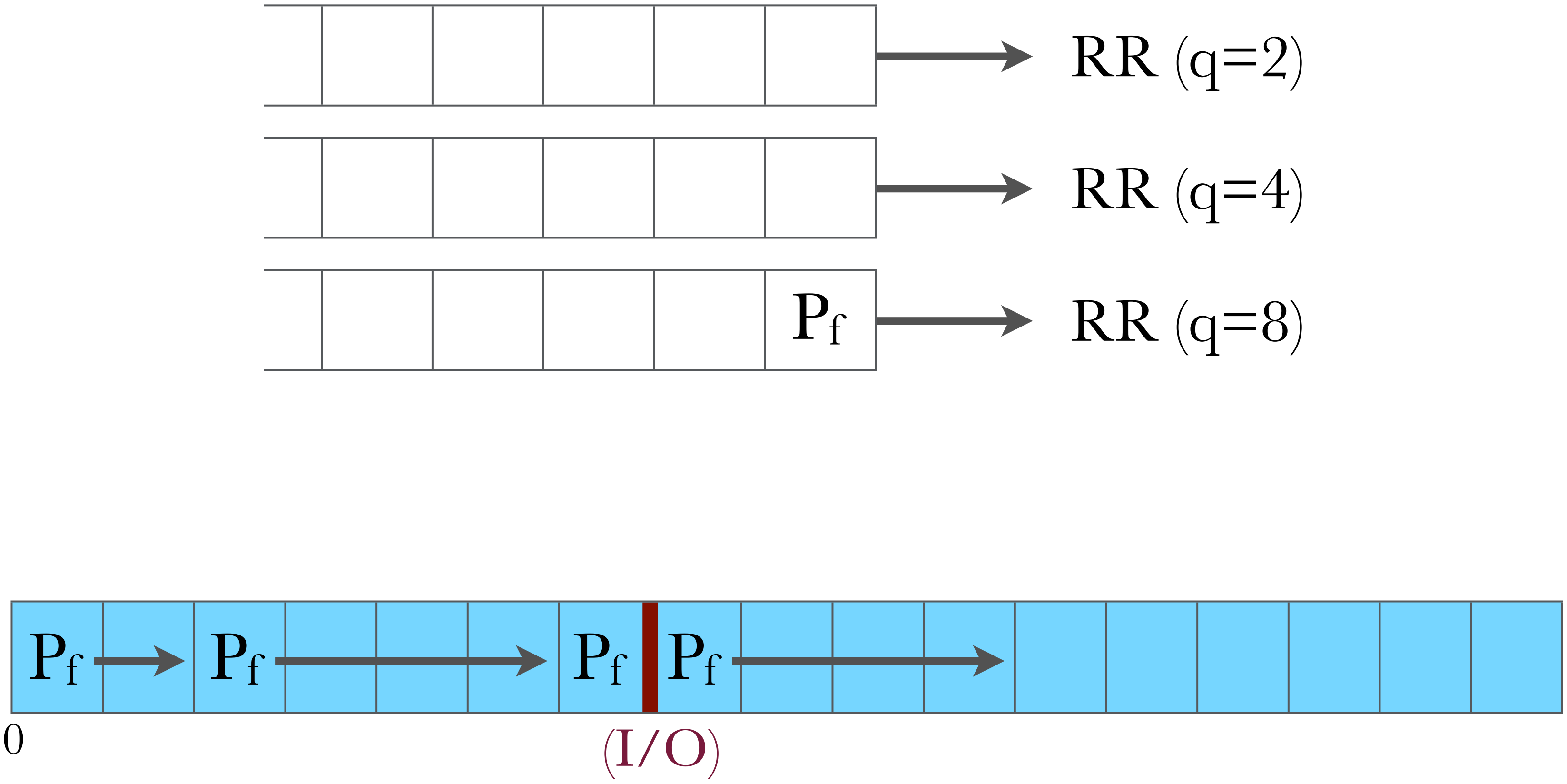


RR (q=2)

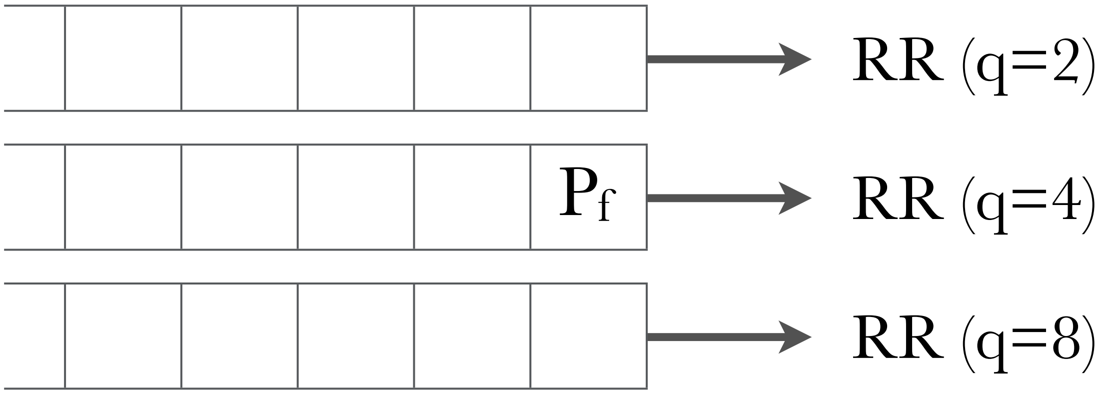
RR (q=4)

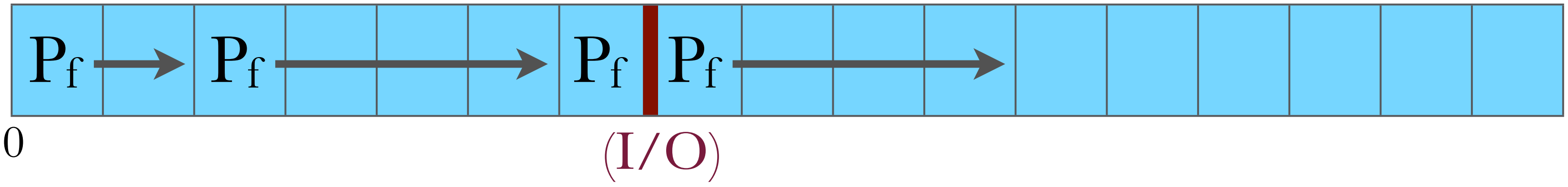
RR (q=8)

e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)



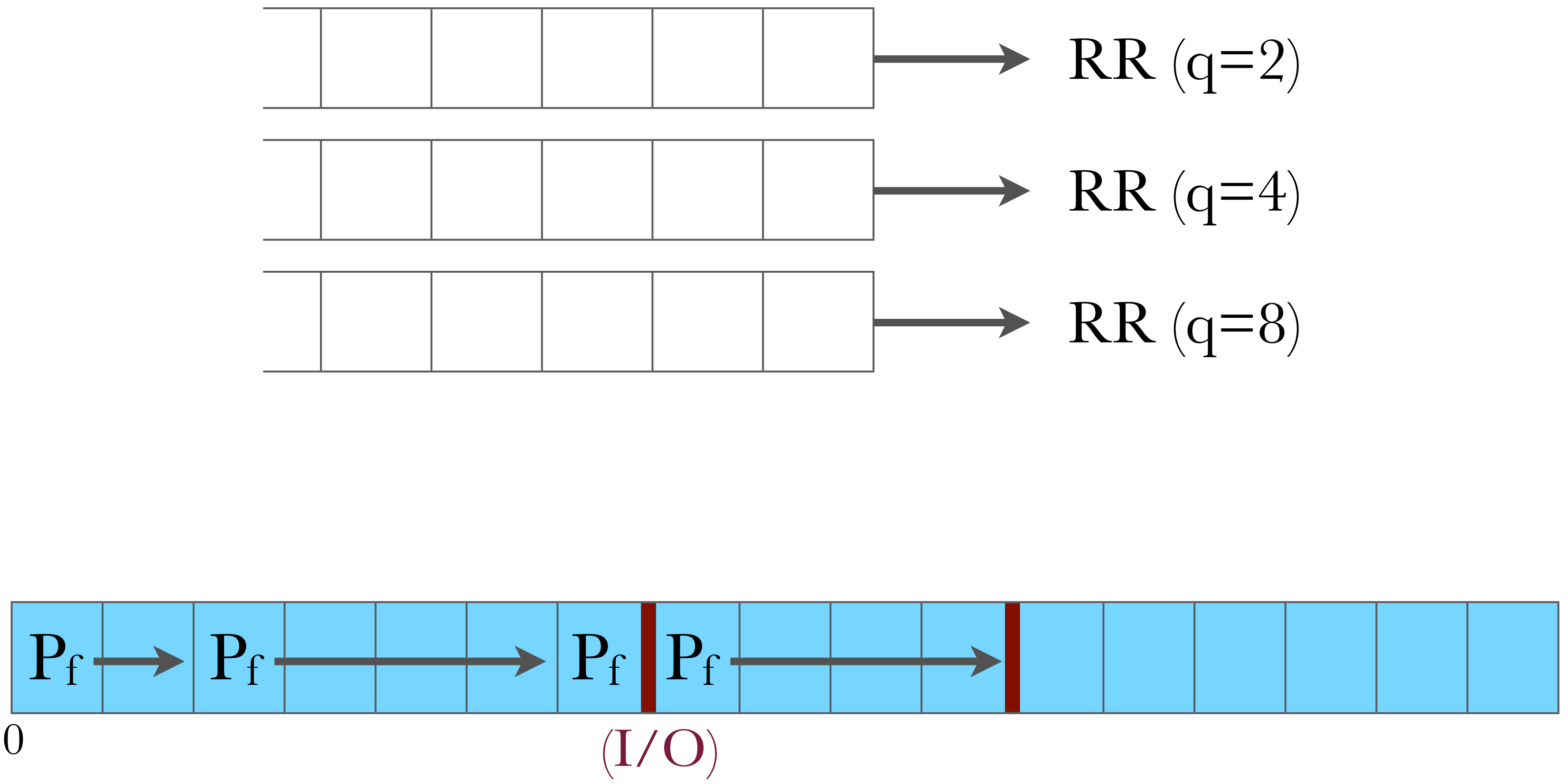
e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)





e.g., P_{flaky} arrives at t=0

CPU burst lengths = 7, 4, 1, 5 (I/O between)

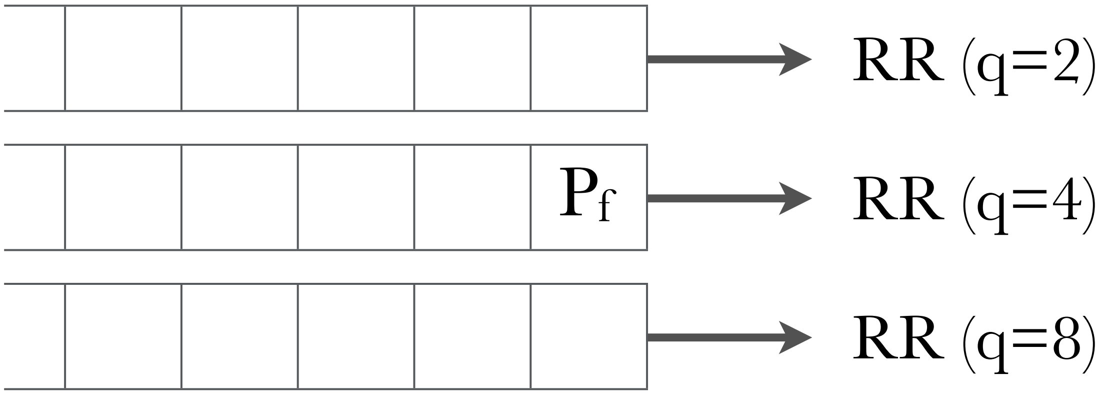


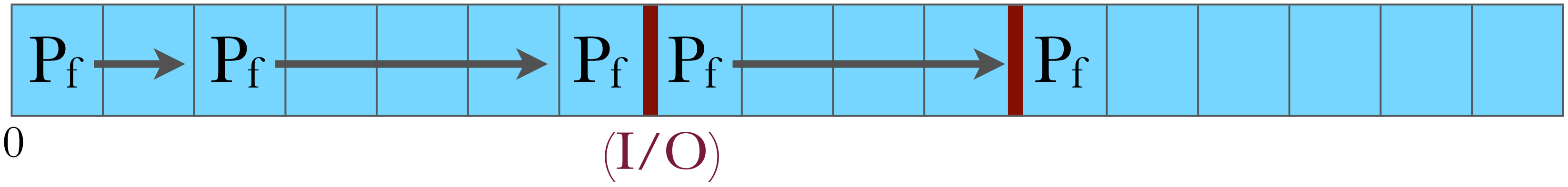
RR (q=2)

RR (q=4)

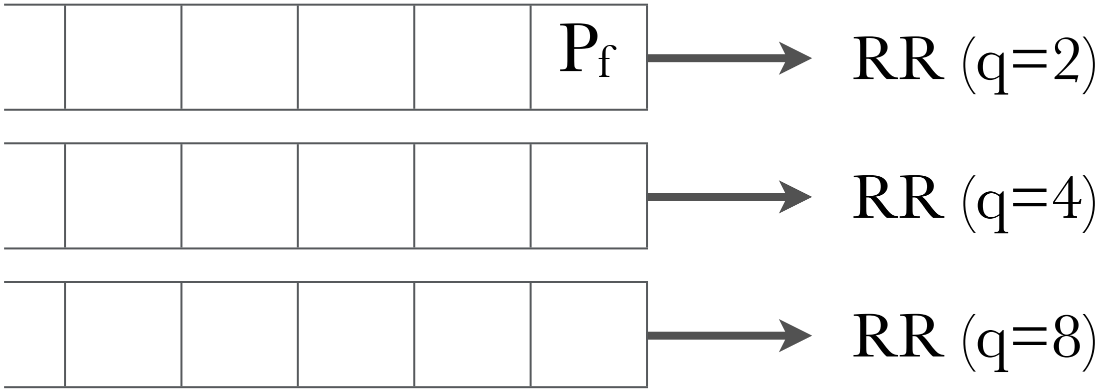
RR (q=8)

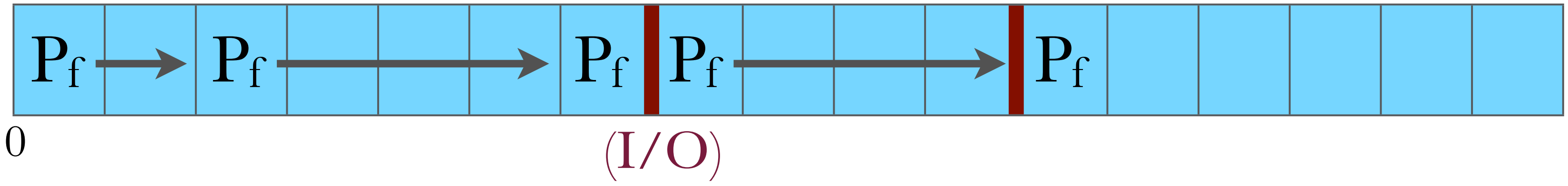
e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)



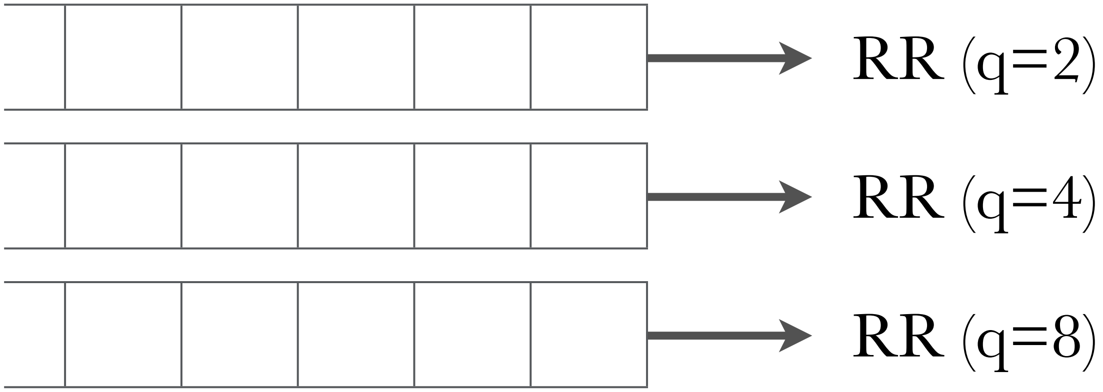


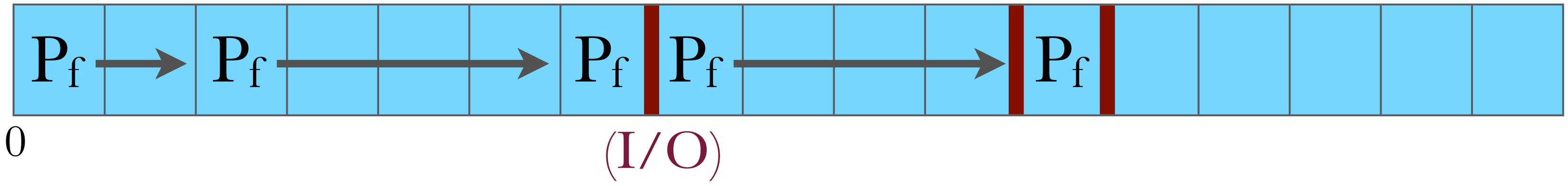
e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)





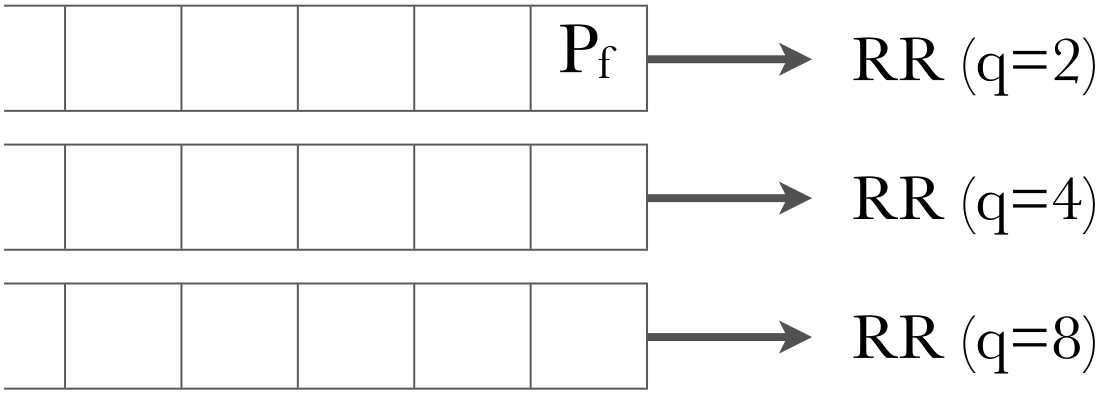
e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)

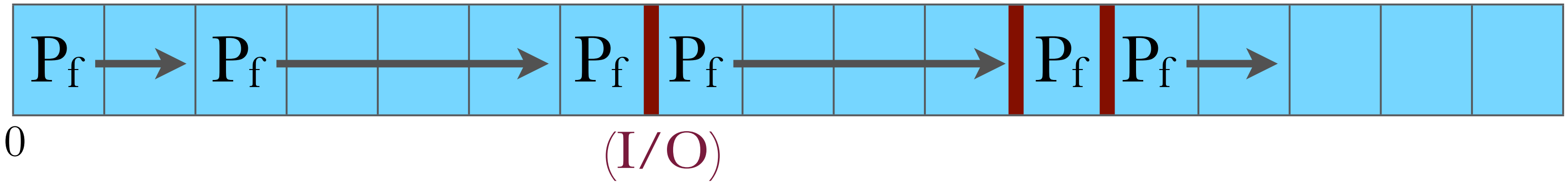




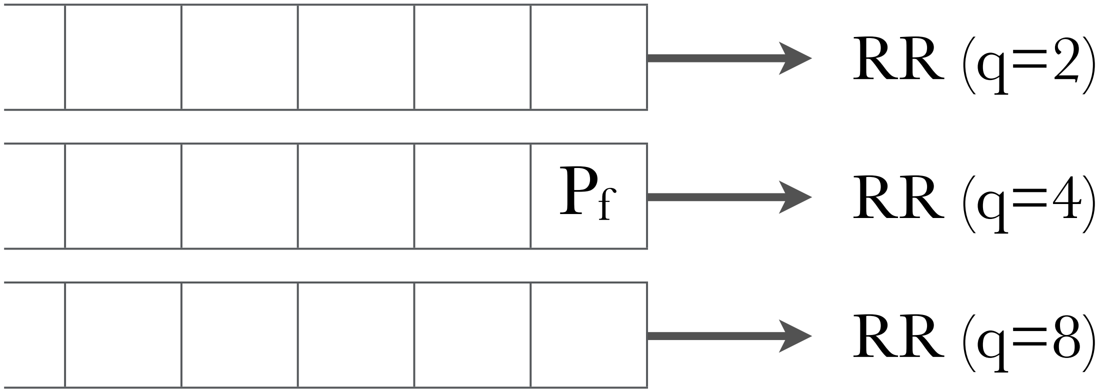
e.g., P_{flaky} arrives at t=0

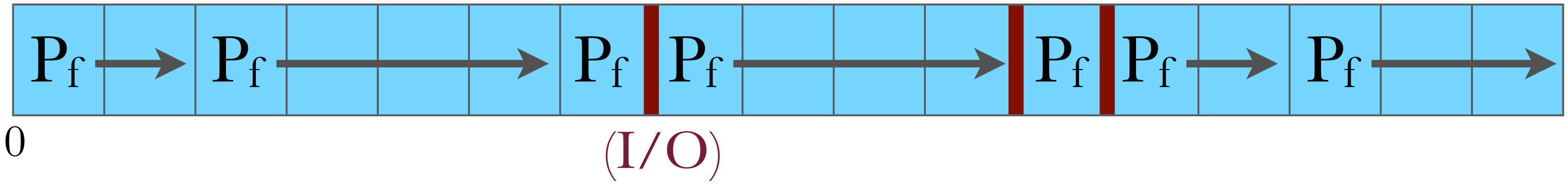
CPU burst lengths = 7, 4, 1, 5 (I/O between)



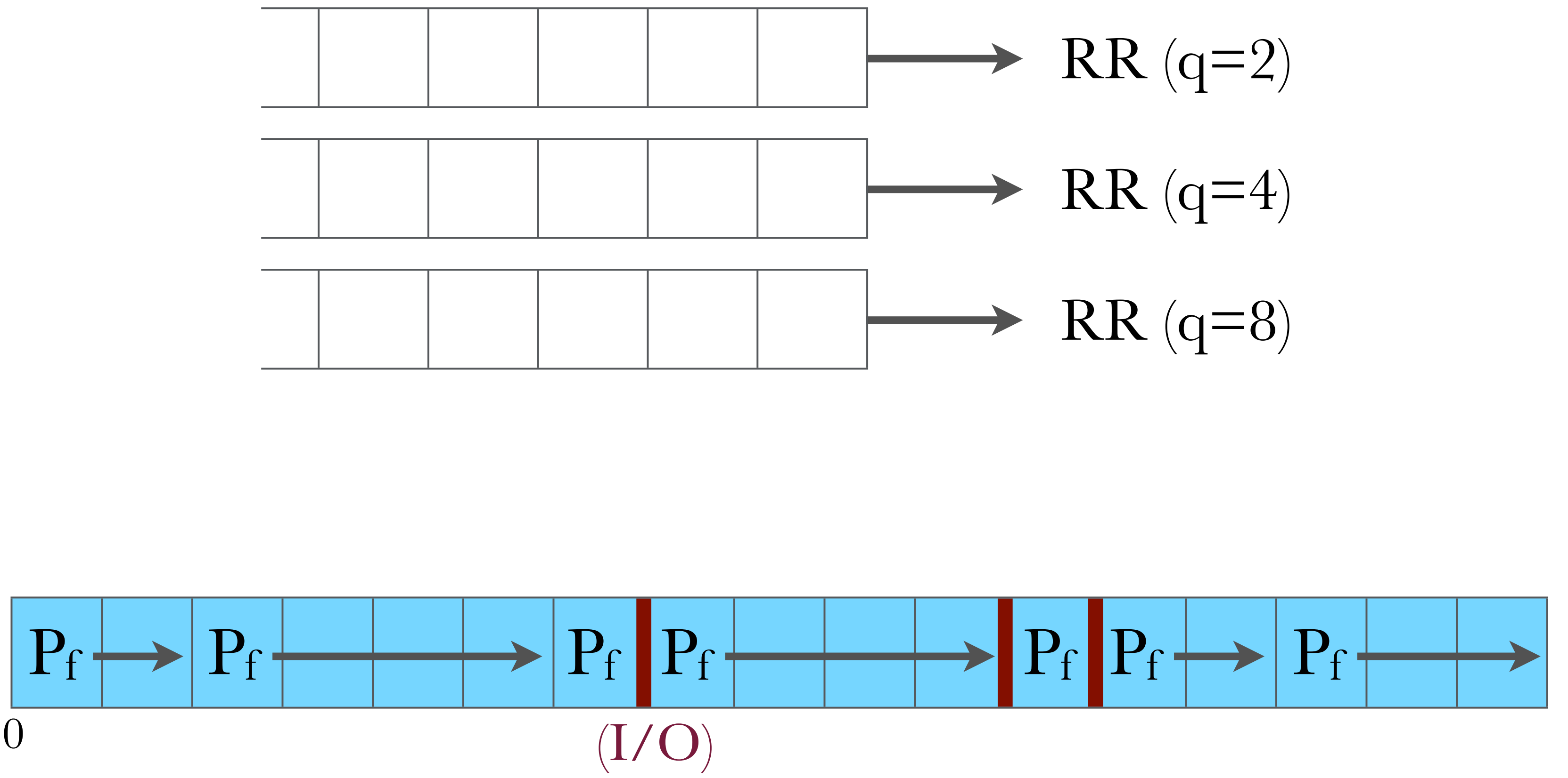


e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)





e.g., P_{flaky} arrives at t=0
CPU burst lengths = 7, 4, 1, 5 (I/O between)



other possible heuristics:

- *multi-queue hops* due to huge bursts
- *exponential backoff* to avoid queue hopping
- *dynamic queue creation* for outliers

§Scheduler Evaluation

i.e., how well does a given scheduling policy perform under different loads?

- typically, w.r.t. scheduling metrics: wait time, turnaround, utilization, etc.

n.b., numerical metrics (e.g., wait time) are important, but may not tell the full story

e.g., how, subjectively, does a given scheduler "feel" under regular load?

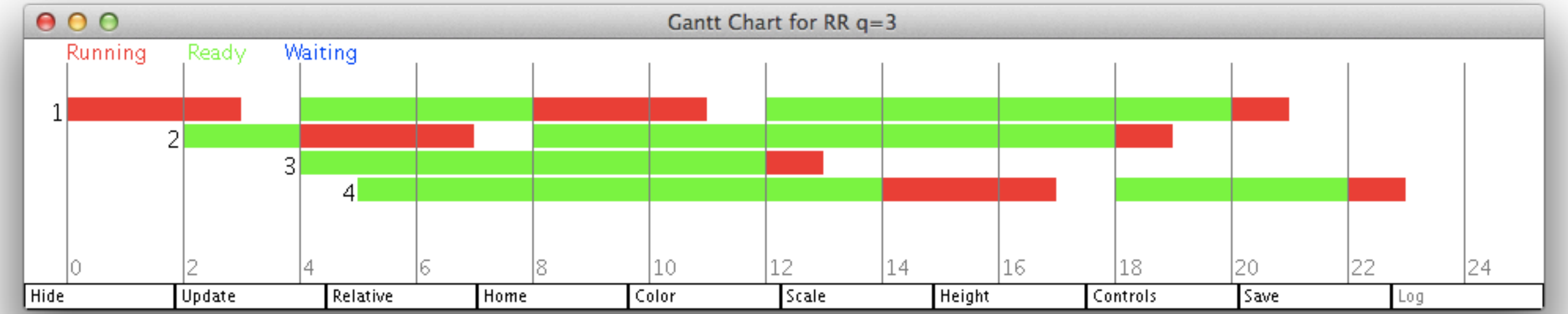
1. paper & pencil computations
2. *simulations* with synthetic or real-world job traces
3. *mathematical models*; e.g., *queueing theory*
4. *real world testing* (e.g., production OSes)

(never fear, you'll try your hand at all!)

e.g., UTSA process scheduling simulator

- specify scheduling discipline and job details in configuration file
  - bursts can be defined discretely, or using *probability distributions*

output: Gantt charts & metrics



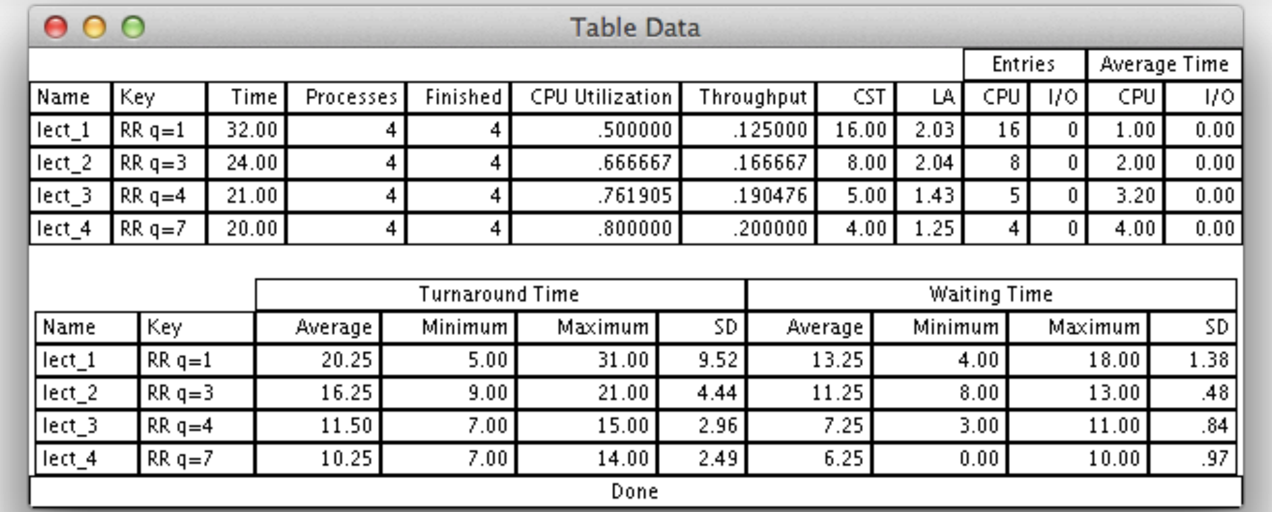


Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
lect_1	RR q=1	32.00	4	4	.500000	.125000	16.00	2.03	16	0	1.00	0.00
lect_2	RR q=3	24.00	4	4	.666667	.166667	8.00	2.04	8	0	2.00	0.00
lect_3	RR q=4	21.00	4	4	.761905	.190476	5.00	1.43	5	0	3.20	0.00
lect_4	RR q=7	20.00	4	4	.800000	.200000	4.00	1.25	4	0	4.00	0.00

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
lect_1	RR q=1	20.25	5.00	31.00	9.52	13.25	4.00	18.00	1.38
lect_2	RR q=3	16.25	9.00	21.00	4.44	11.25	8.00	13.00	.48
lect_3	RR q=4	11.50	7.00	15.00	2.96	7.25	3.00	11.00	.84
lect_4	RR q=7	10.25	7.00	14.00	2.49	6.25	0.00	10.00	.97

Done

SJF *vs.* PSJF *vs.* RR, *q*=10 *vs.* RR, *q*=20

processes: uniform bursts ≤ 20, CST = 1.0

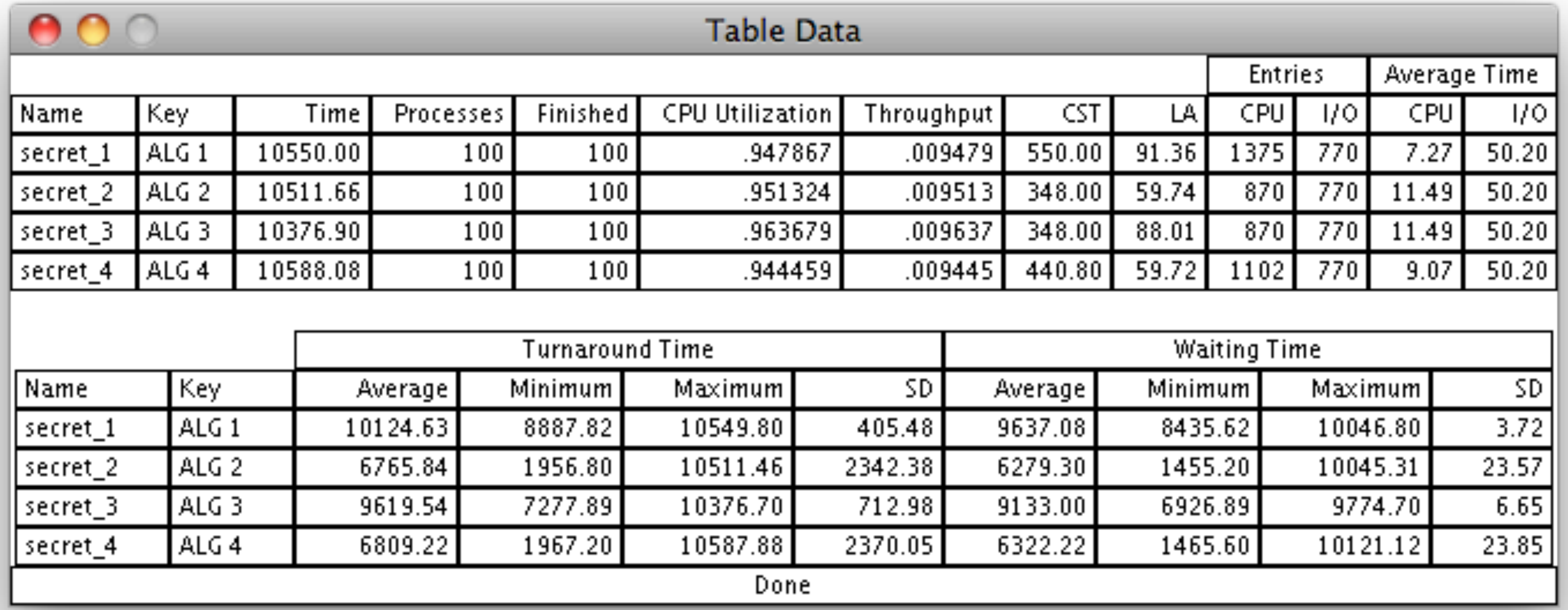


Table Data

									Entries		Average Time	
Name	Key	Time	Processes	Finished	CPU Utilization	Throughput	CST	LA	CPU	I/O	CPU	I/O
secret_1	ALG 1	10550.00	100	100	.947867	.009479	550.00	91.36	1375	770	7.27	50.20
secret_2	ALG 2	10511.66	100	100	.951324	.009513	348.00	59.74	870	770	11.49	50.20
secret_3	ALG 3	10376.90	100	100	.963679	.009637	348.00	88.01	870	770	11.49	50.20
secret_4	ALG 4	10588.08	100	100	.944459	.009445	440.80	59.72	1102	770	9.07	50.20

		Turnaround Time				Waiting Time			
Name	Key	Average	Minimum	Maximum	SD	Average	Minimum	Maximum	SD
secret_1	ALG 1	10124.63	8887.82	10549.80	405.48	9637.08	8435.62	10046.80	3.72
secret_2	ALG 2	6765.84	1956.80	10511.46	2342.38	6279.30	1455.20	10045.31	23.57
secret_3	ALG 3	9619.54	7277.89	10376.70	712.98	9133.00	6926.89	9774.70	6.65
secret_4	ALG 4	6809.22	1967.20	10587.88	2370.05	6322.22	1465.60	10121.12	23.85

Done

Which is which?