Processes and Resources

CS-350 - Fall 2024

CS-350 – Fundamentals of Computing Systems
2024 – Renato Mancuso

BOSTON UNIVERSITY

From Program to Process

sticking to the computing flavor

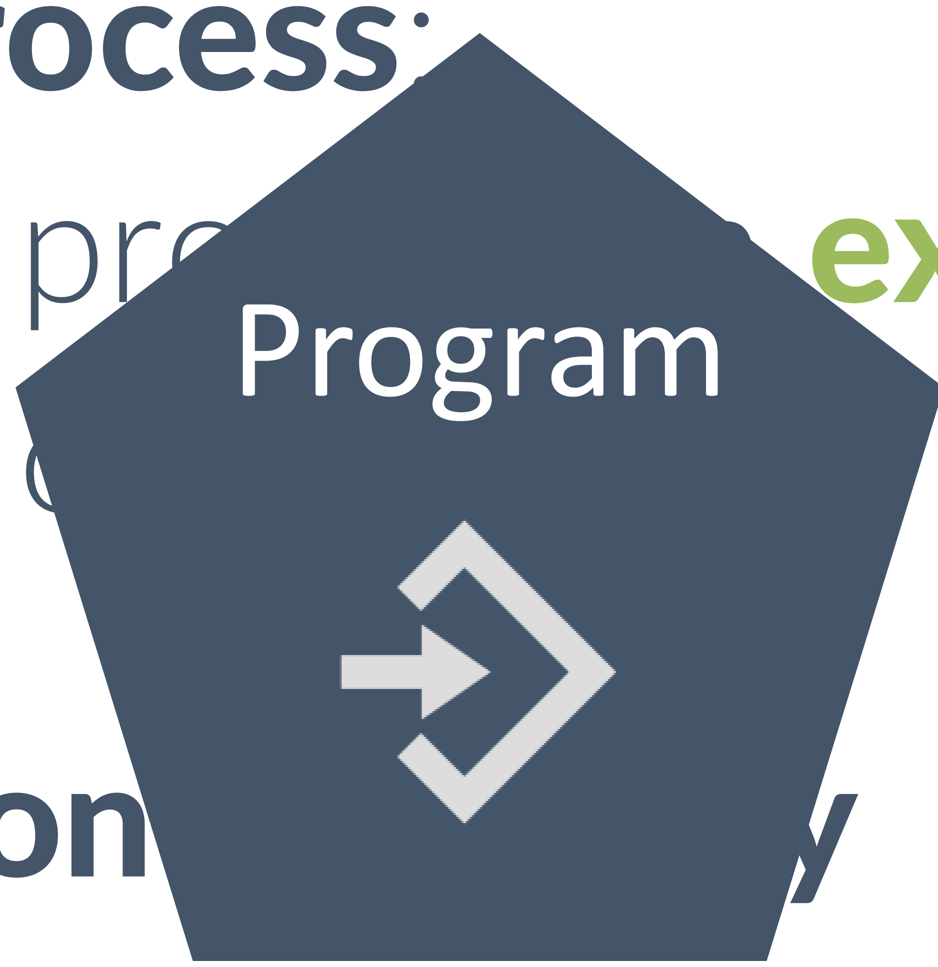
Simply put:

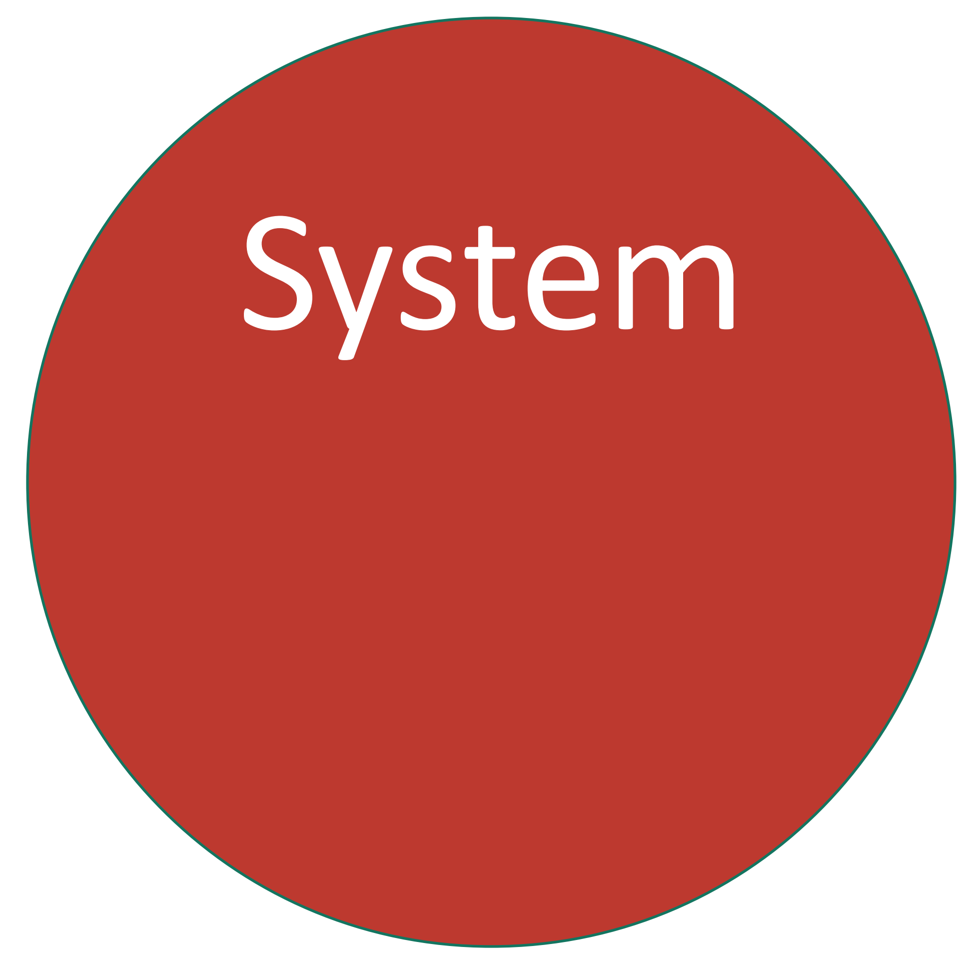
Process:

pr **execution**

(

…on **y**

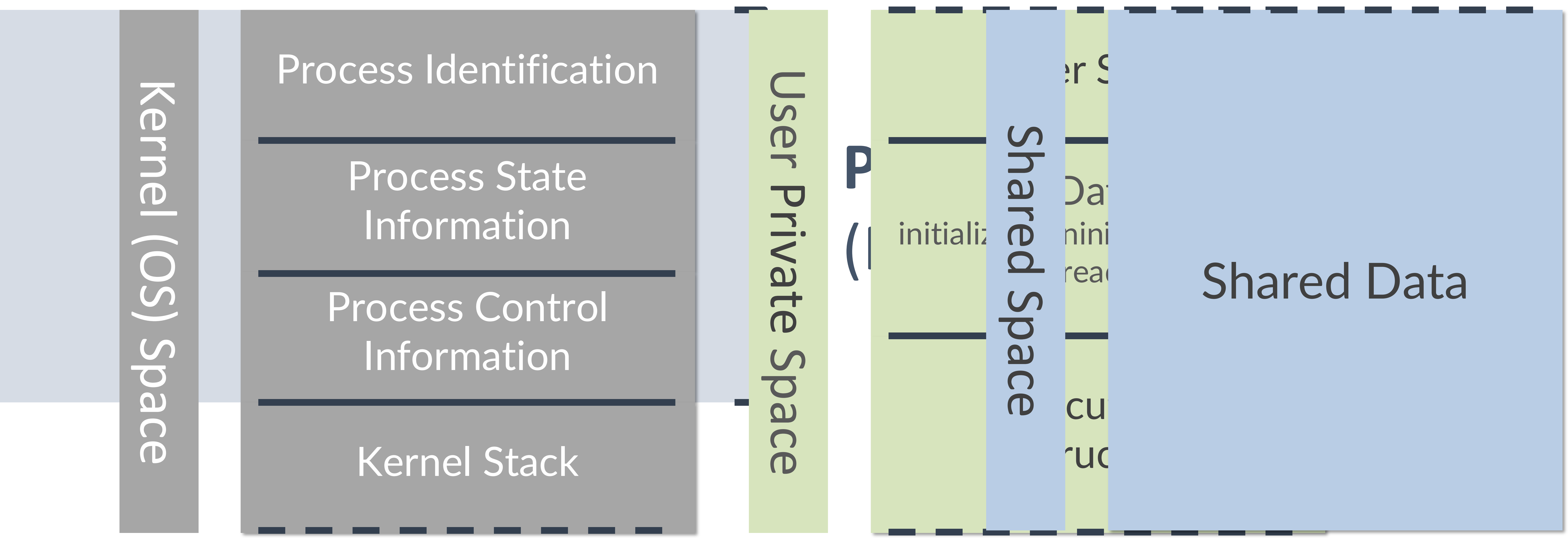




From Program to Process

sticking to the computing flavor

And to be “alive”, a **state** is necessary



BOSTON UNIVERSITY

From Program to Process

sticking to the computing flavor

But we can **abstract away** the state
and only focus on macro-phases

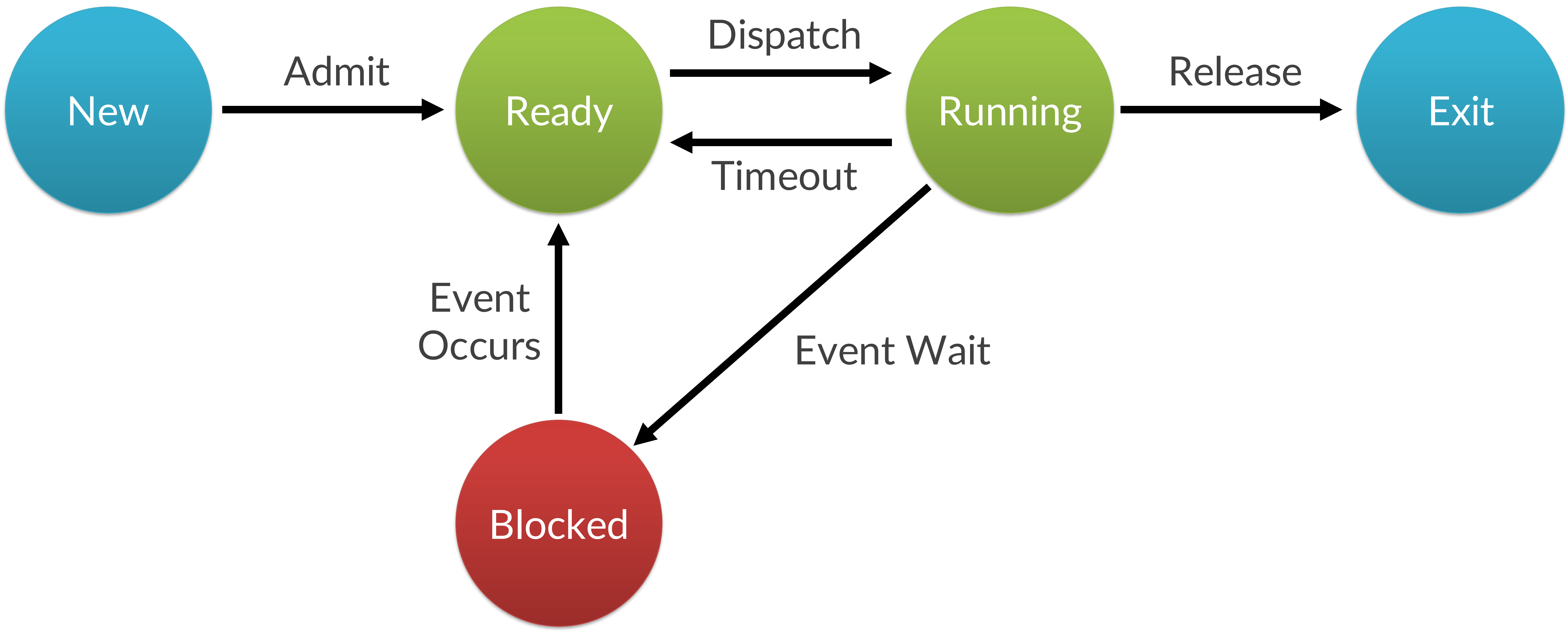
1. **Consume** some **resource** (e.g. CPU)
2. Request / wait for an **external** service / event
3. Do nothing until external service done

And back up!

BOSTON UNIVERSITY

Transition State Diagram

It's diagrams all the way down



Transition State Diagram

It's diagrams all the way down

Ready — **Ready to** consume the resource

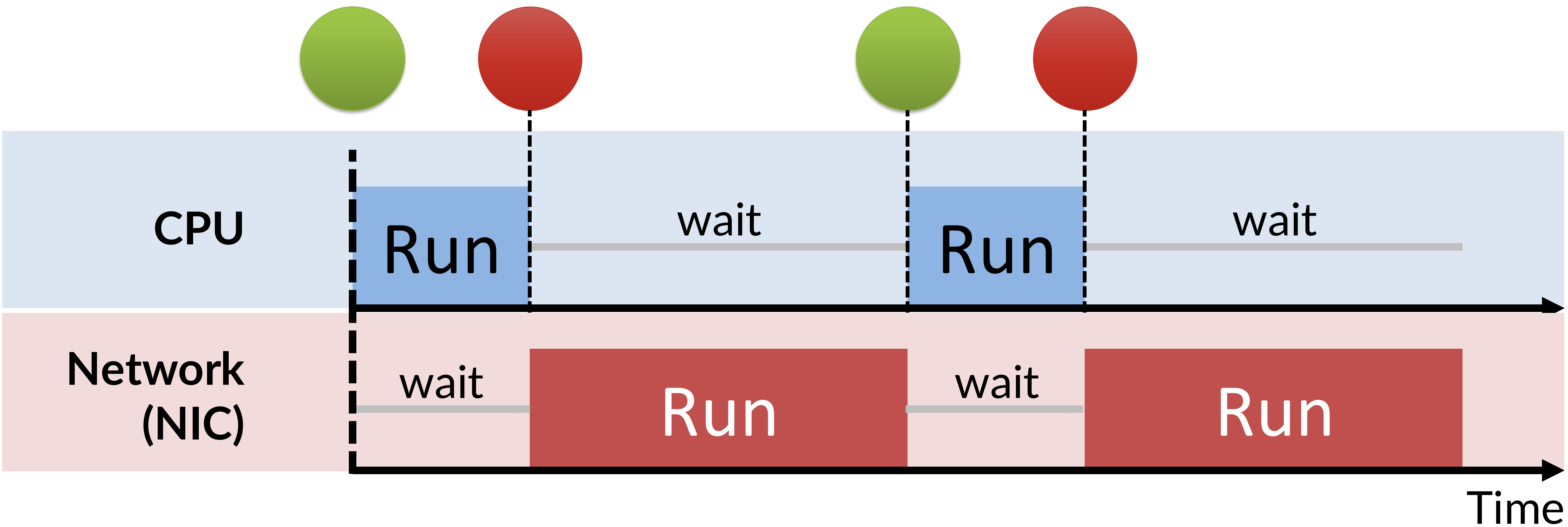
Running — **Currently** consuming the resource

Blocked — **Not competing** for resource
because waiting for **some event**

BOSTON UNIVERSITY

Resource Usage Timeline

still a diagram, but over time

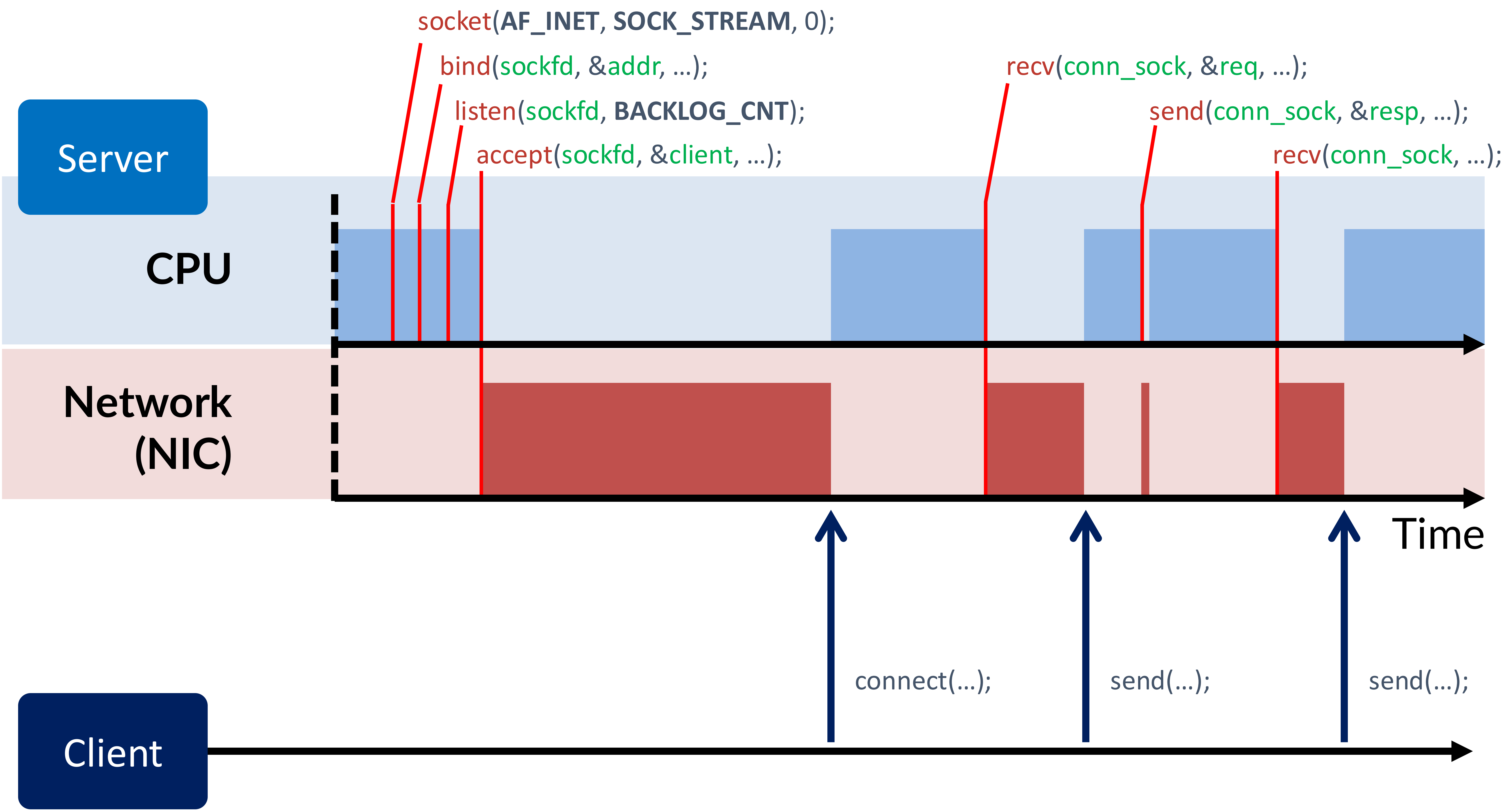


While blocked on I/O, **CPU idle**!

Uh and **the same for I/O** when CPU busy

Let's Build a Server!

coincidentally also what you are doing in your first assignment

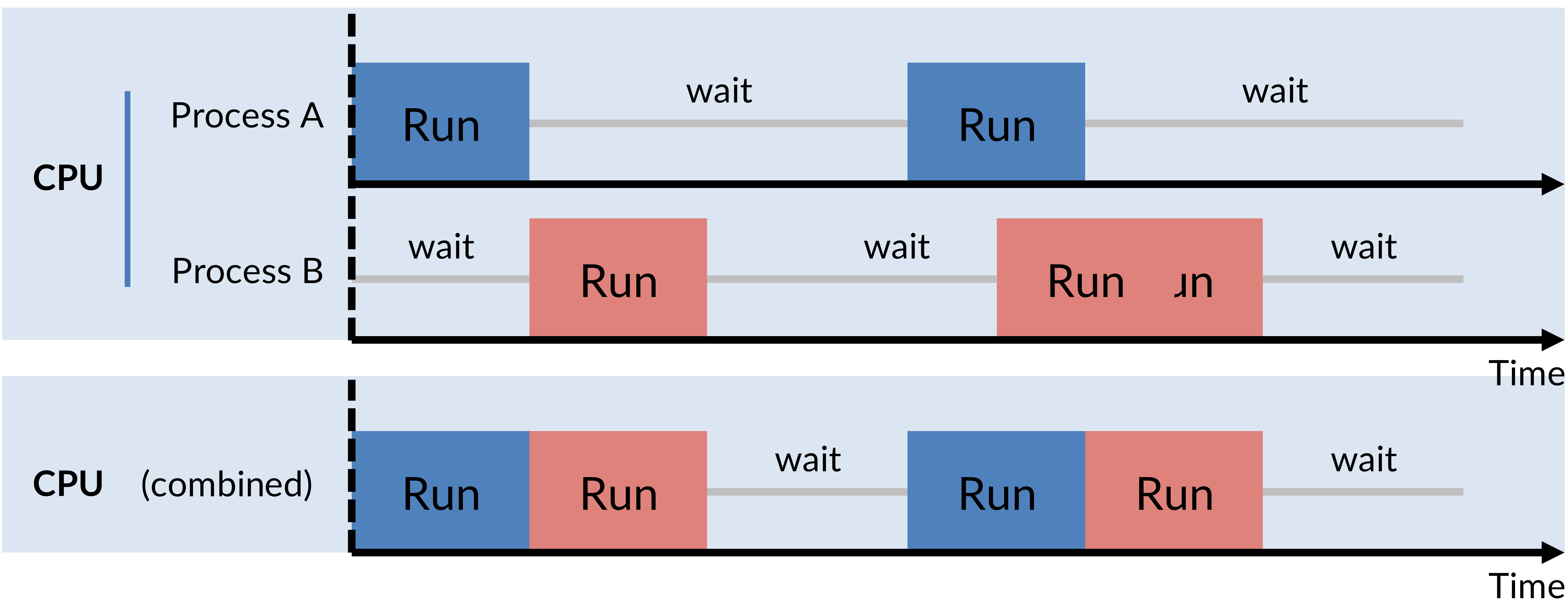


BOSTON UNIVERSITY

Multi-Programming

ooh jeez here we go

we throw **more processes** to the CPU?

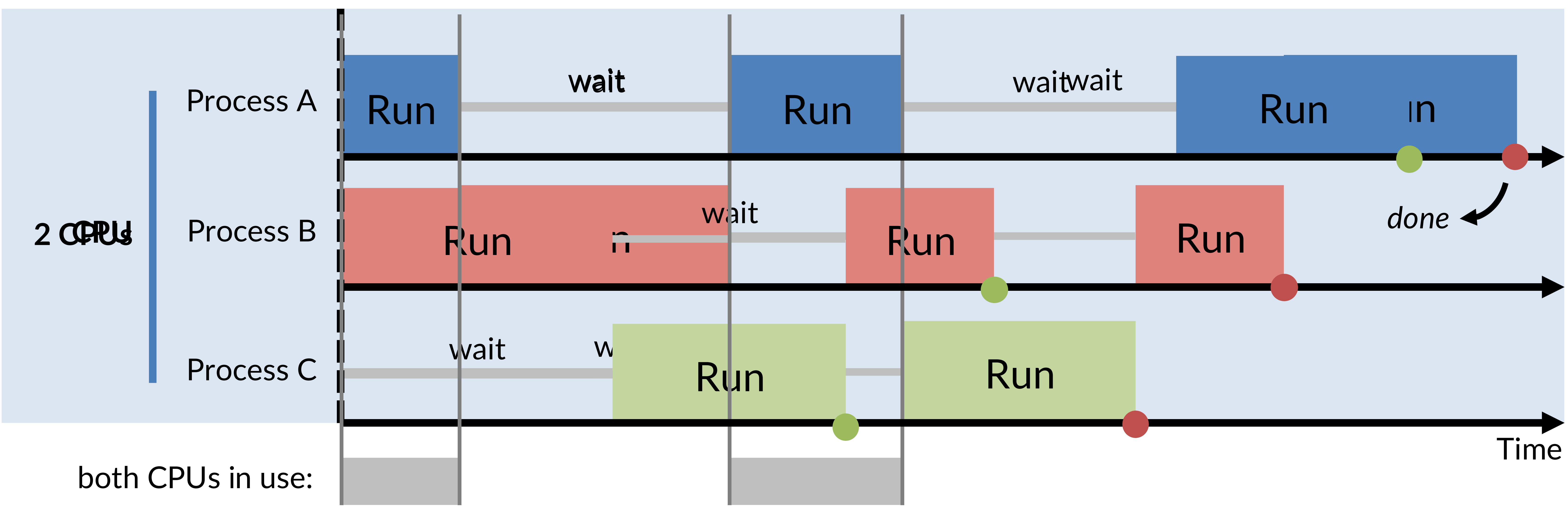


BOSTON UNIVERSITY

Multi-Processing

yep, before it was multi-*programming*

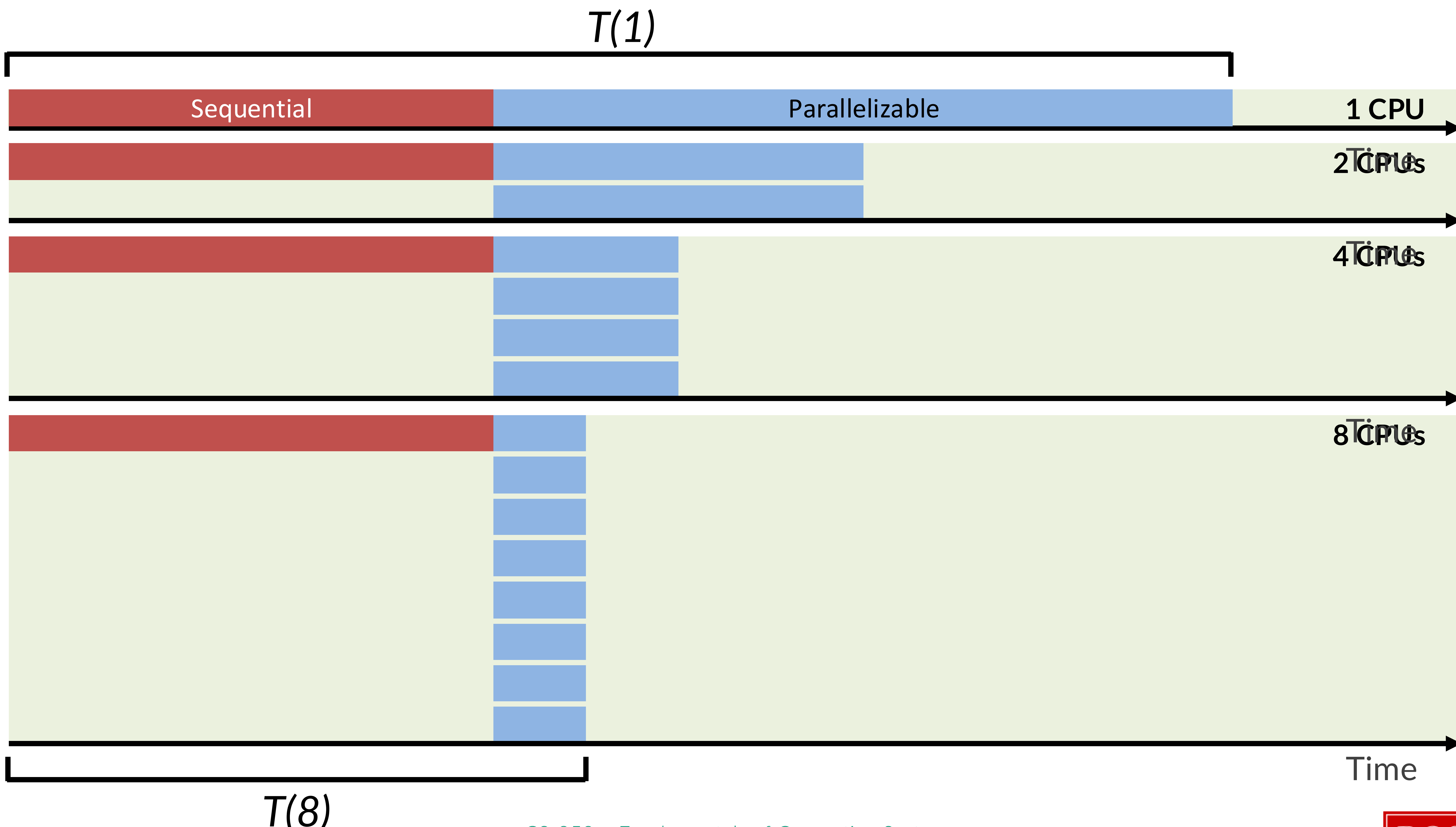
we throw **more processors**



BOSTON UNIVERSITY

But there is a Limit

time for the bad news

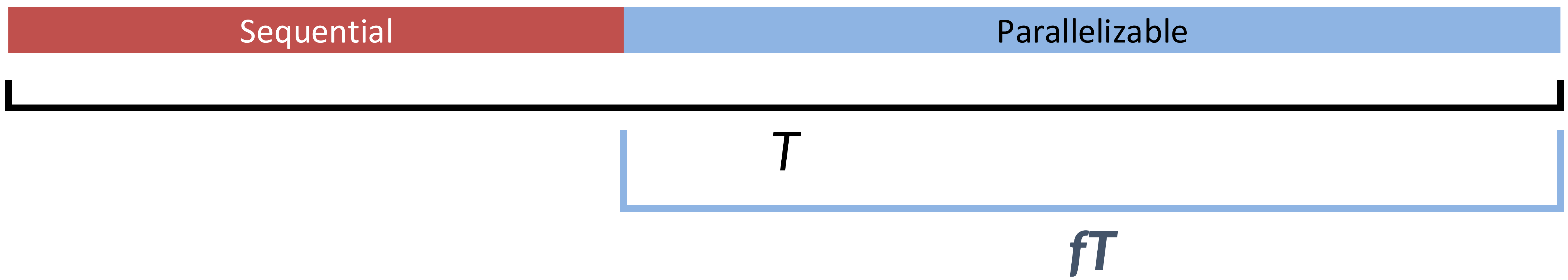


BOSTON UNIVERSITY

Computing the Speedup

obey the law

If the **fraction** of parallelizable code is ***f***



speedup =

T(N) =

Amdahl's Law

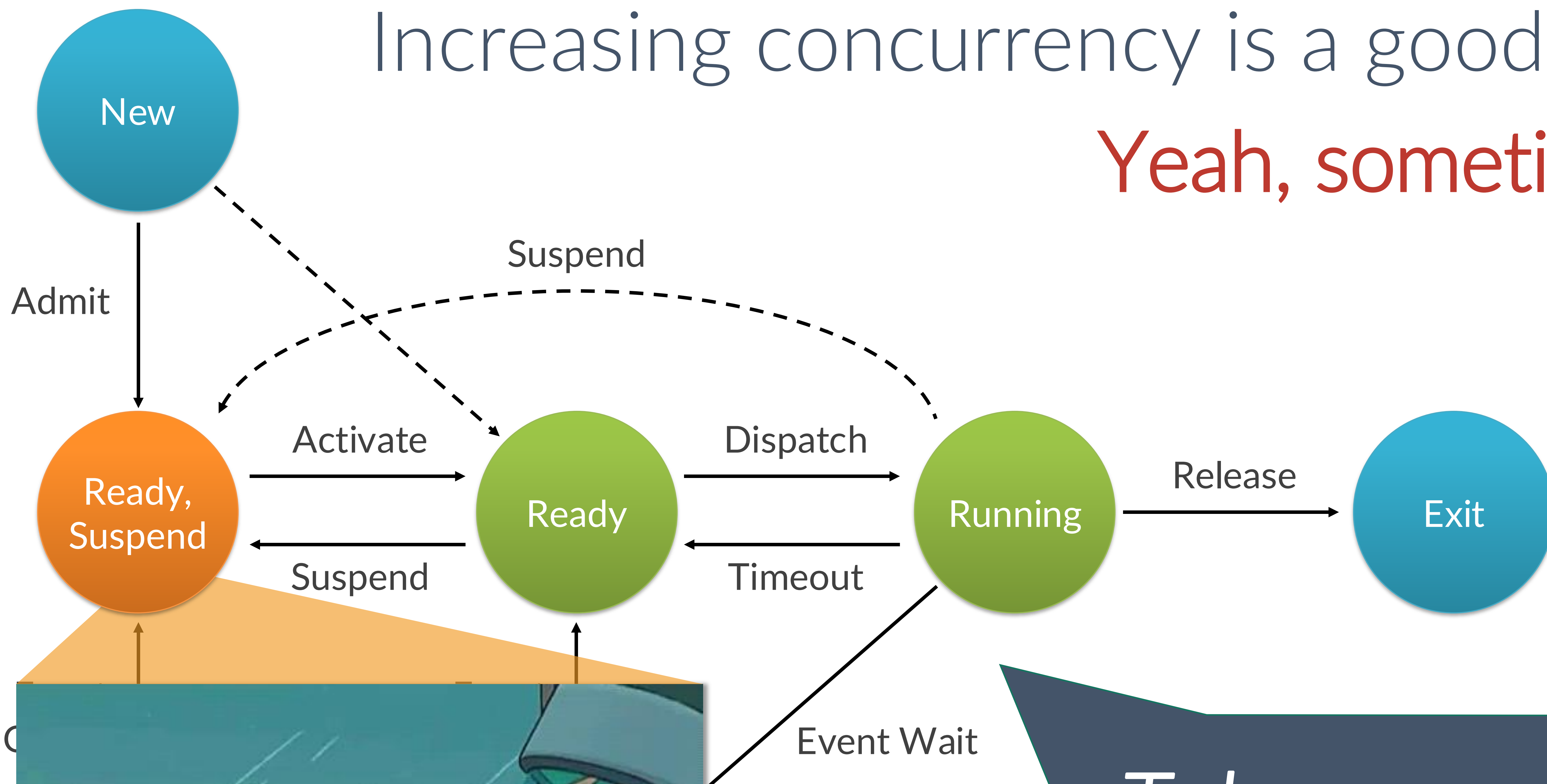
parallelizing stuff since 1967

Managing Concurrency

there is a new sheriff in town

Increasing concurrency is a good idea.

Yeah, sometimes.



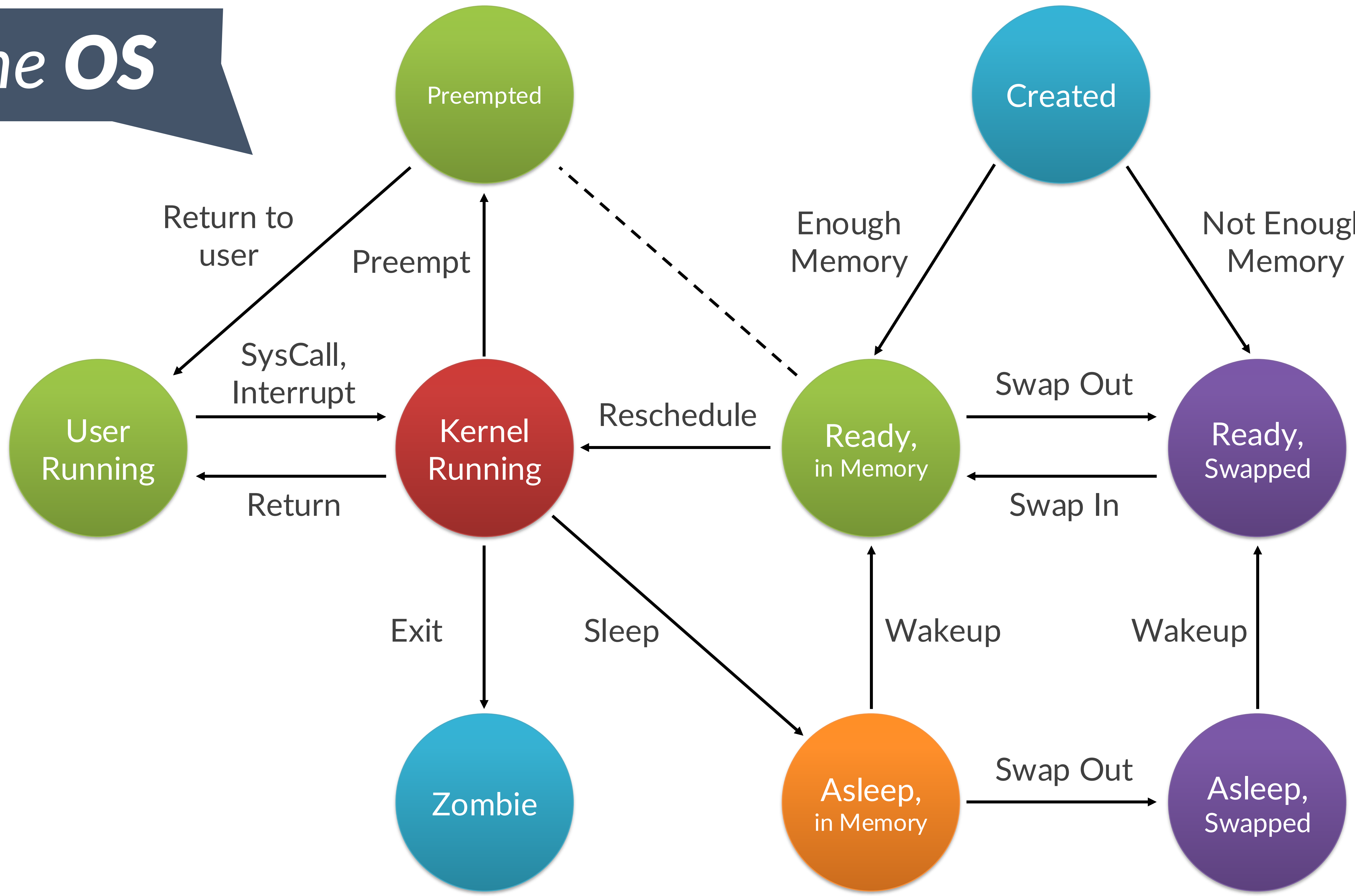
Take some processes out of the game

BOSTON UNIVERSITY

Managing Concurrency

there is a new sheriff in town

*All hail the **OS***



BOSTON UNIVERSITY

System Model & Metrics

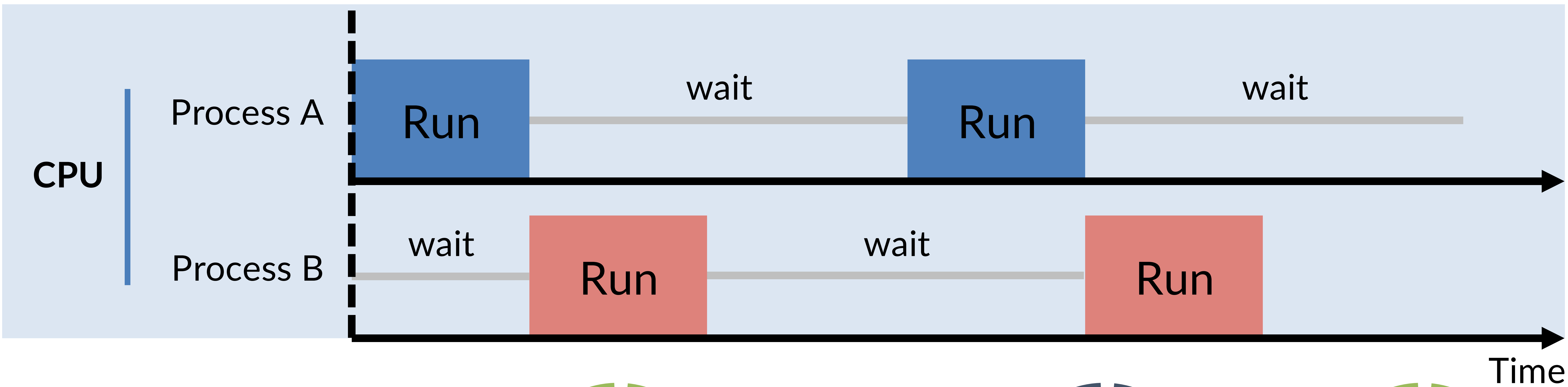
CS-350 - Fall 2024

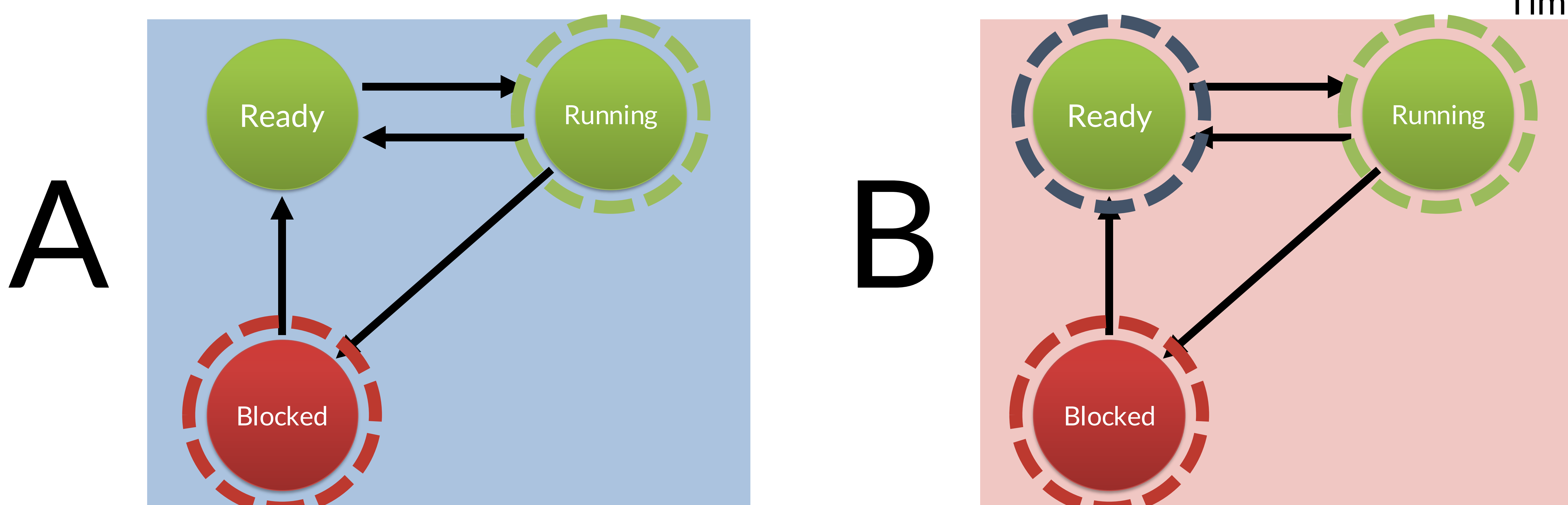
CS-350 – Fundamentals of Computing Systems
2024 – Renato Mancuso

BOSTON UNIVERSITY

Tracking a Process

with respect to a single resource

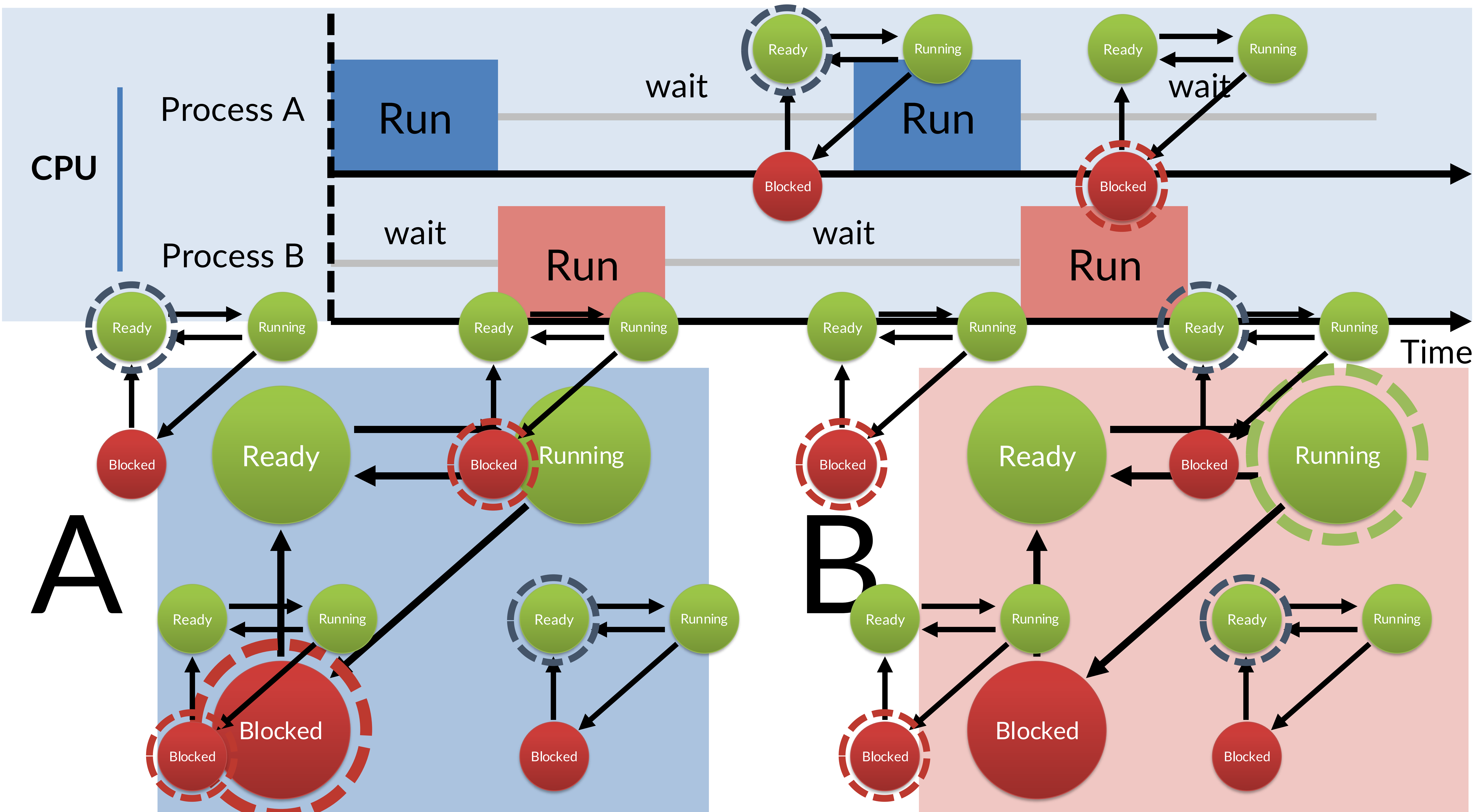




BOSTON UNIVERSITY

Tracking a Process

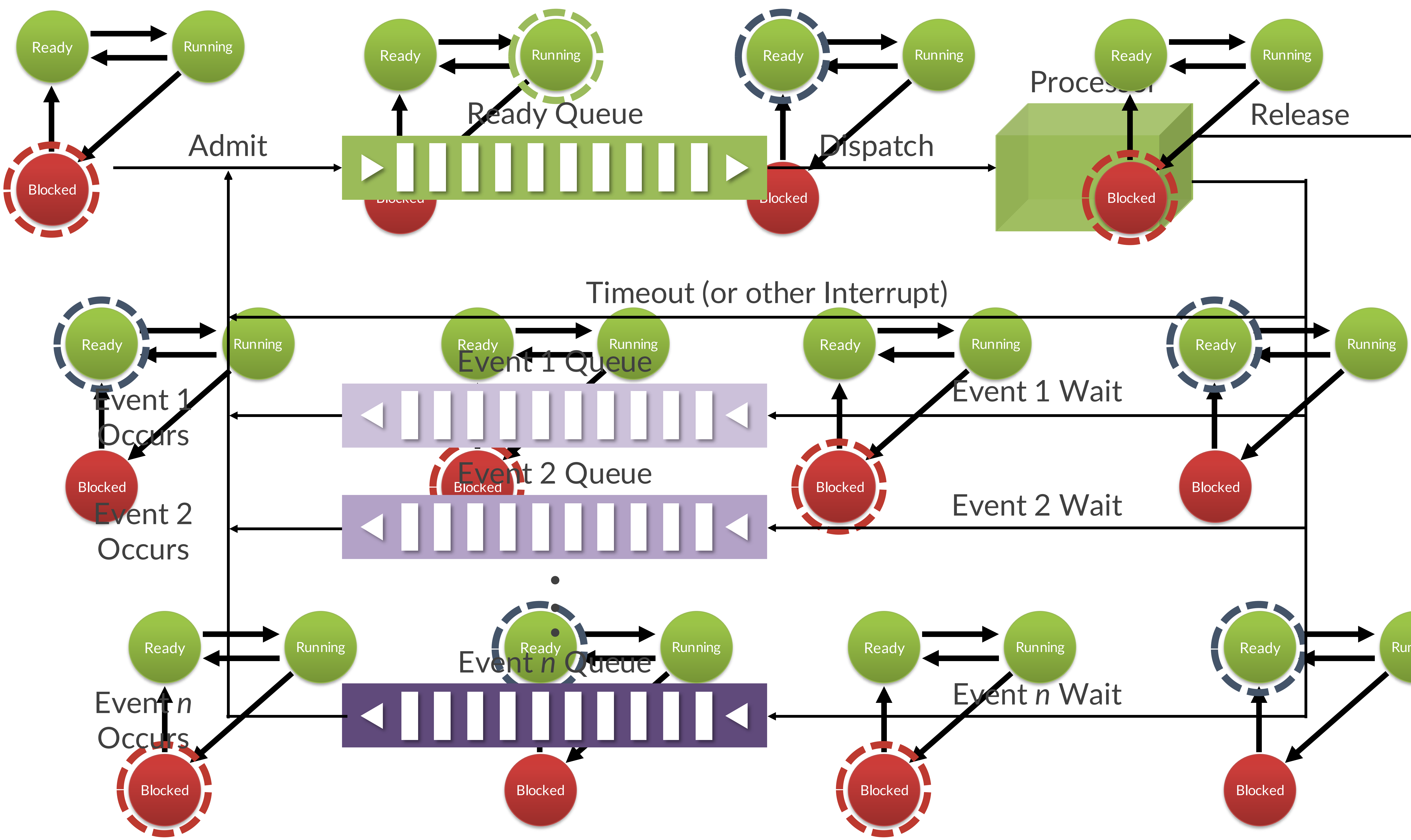
with respect to a single resource



BOSTON UNIVERSITY

Tracking a Process

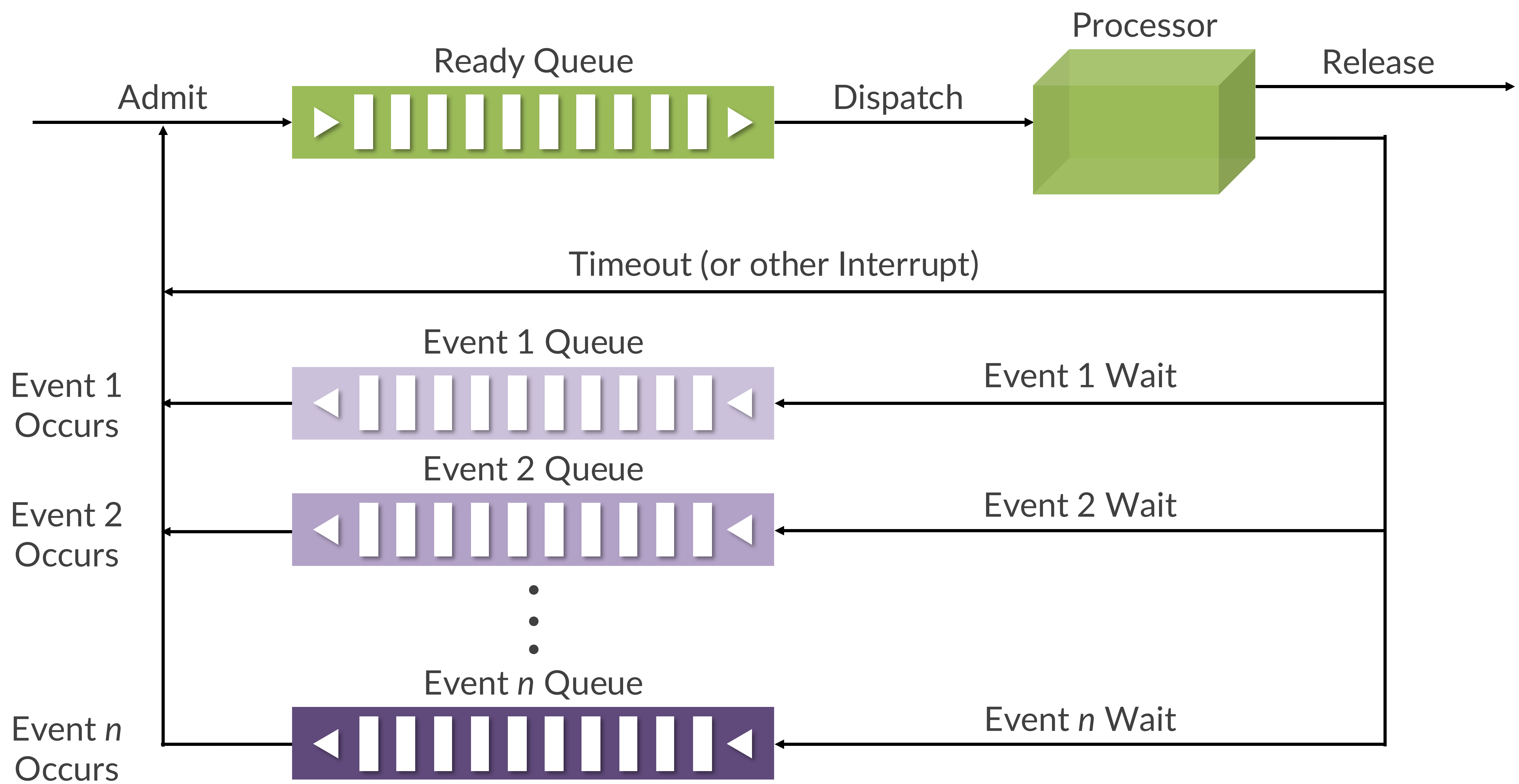
a trippy re-thinking from processes



BOSTON UNIVERSITY

Tracking a System

with multiple resources



BOSTON UNIVERSITY

CS-350 in a Nutshell

oh boy here we go again

Manage the **consumption of resources**

and **coordinate interaction** of concurrent processes

to ensure some **desirable system attributes**

which can be **evaluated** and **contrasted**

Performance is Subjective

the tales of dueling sys-admins and users

System-centric perspective

- Are resources being utilized?
- Is revenue being maximized?
- Is system's wear minimized?
- *…etc.*

Process-centric perspective

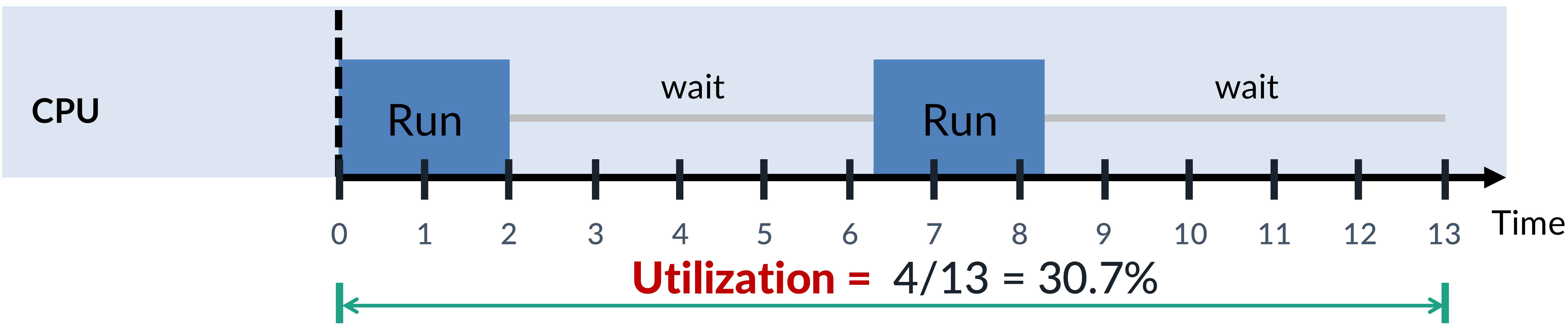
- Average processing time?
- Worst-case processing time?
- Waiting time before service?
- *…etc.*

BOSTON UNIVERSITY

Utilization

aka the "bang" per "buck"

Utilization: ***fraction of time over a given time window during which the resource is busy (not idle).***



NOTE: utilization is a per-resource metric!

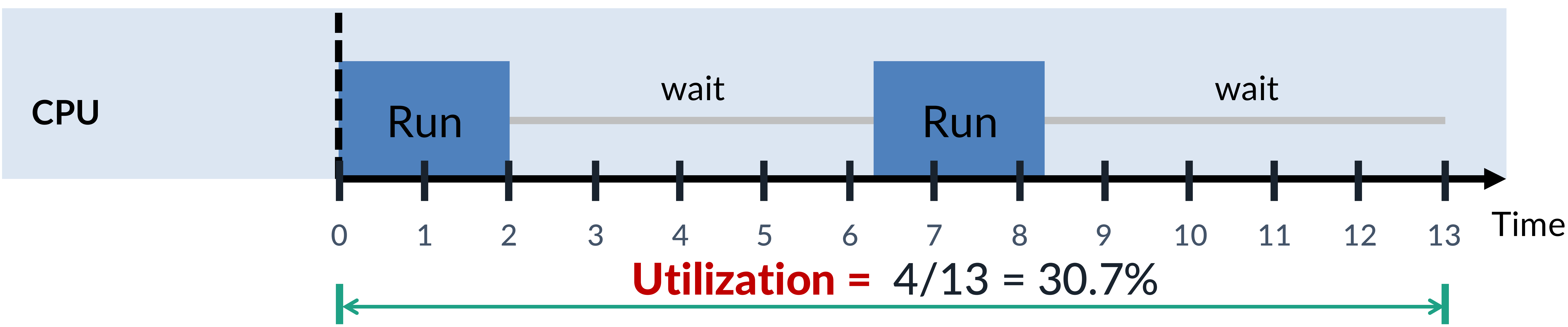
$$\mathbf{U} = \frac{\text{Time Busy}}{\text{Total Time}} = 1 - \frac{\text{Time Idle}}{\text{Total Time}}$$

Multi-Programming & Utilization

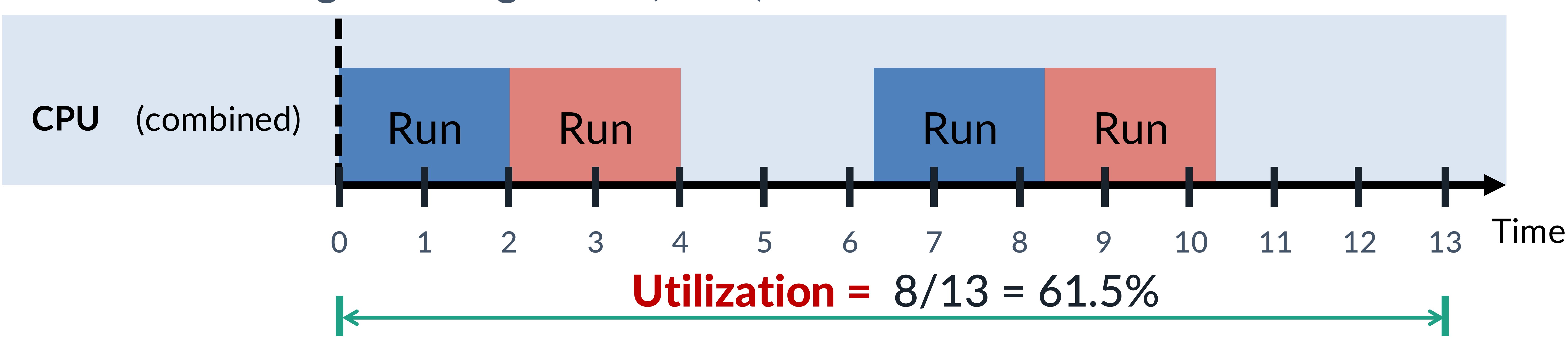
aka the "bang" per "buck"

Utilization: ***fraction of time over a given time window during which the resource is busy (not idle).***

*With Multi-Programming Level (**MPL**) = 1:*



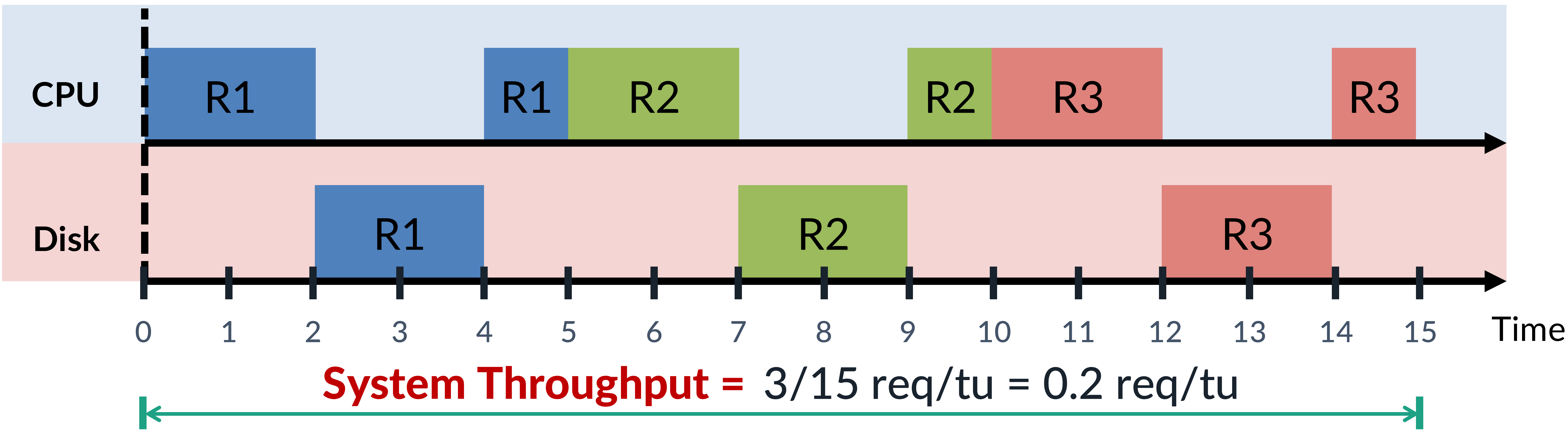
*With Multi-Programming Level (**MPL**) = 2:*



Throughput

a.k.a. output rate of requests

Throughput: ***number of completed requests over a given time window.***



System Throughput = 3/15 req/tu = 0.2 req/tu

NOTE: for single resources, or for the whole system!

$$\textbf{Thr} = \frac{\#\text{Completed Requests}}{\text{Total Time}}$$

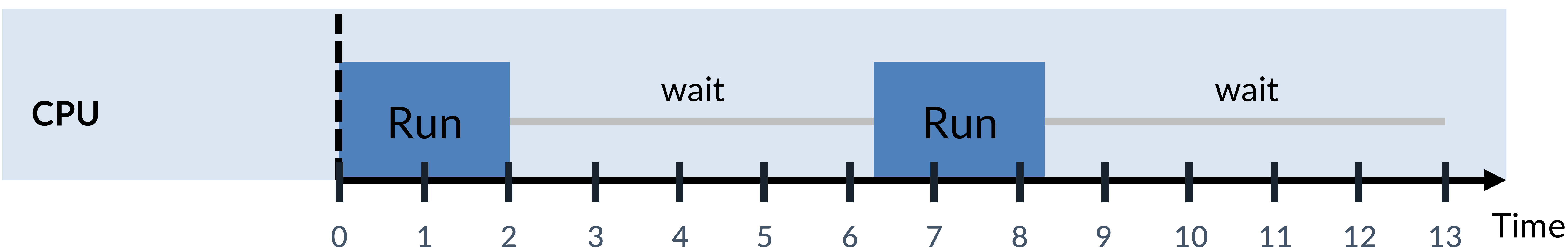
BOSTON UNIVERSITY

Multi-Programming & Throughput

productivity explained

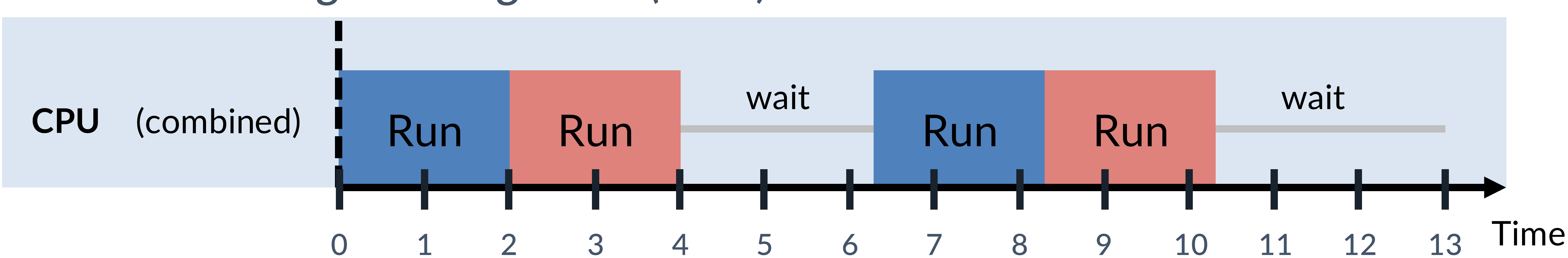
Throughput: ***number of completed requests over a given time window.***

*With Multi-Programming Level (**MPL**) = 1:*



Throughput = 2 every 13 time units = 0.15 req/tu

*With Multi-Programming Level (**MPL**) = 2:*



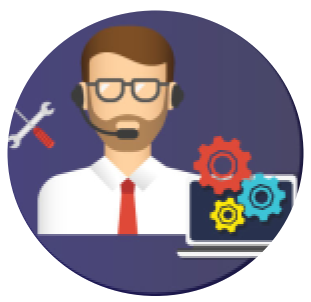
Throughput = 4 every 13 time units = 0.31 req/tu

Fact:

Increasing the MPL improves utilization and throughput.

But then:

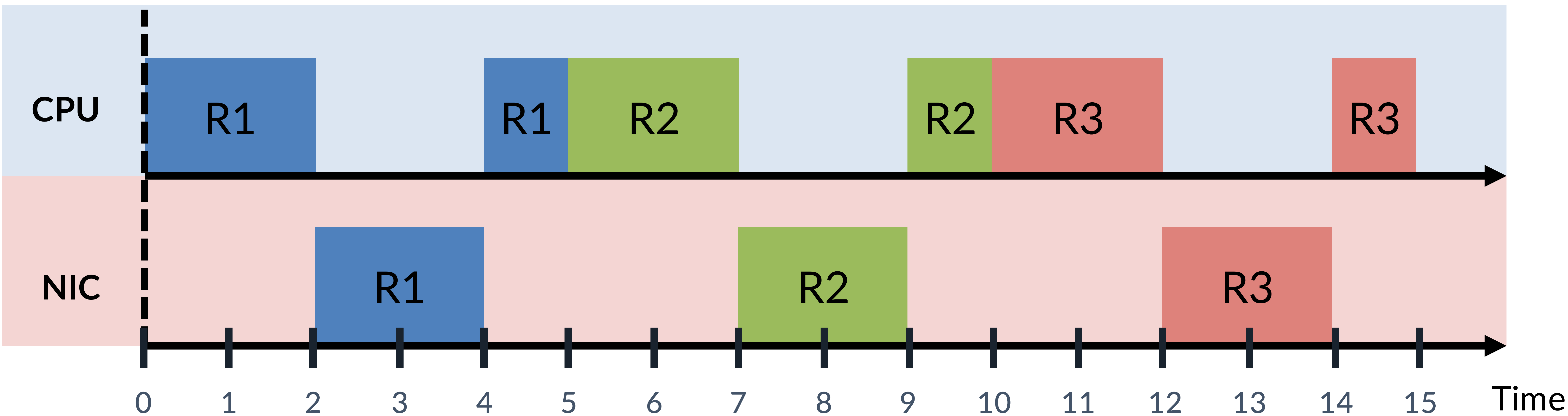
Is this true forever?
I.e. for an arbitrarily high value of MPL?



Capacity
when the party ends

Capacity: ***maximum throughput that can be sustained by the system.***

*With Multi-Programming Level (**MPL**) = 1:*



CPU Utilization = 3/5 = 60%

NIC Utilization = 2/5 = 40%

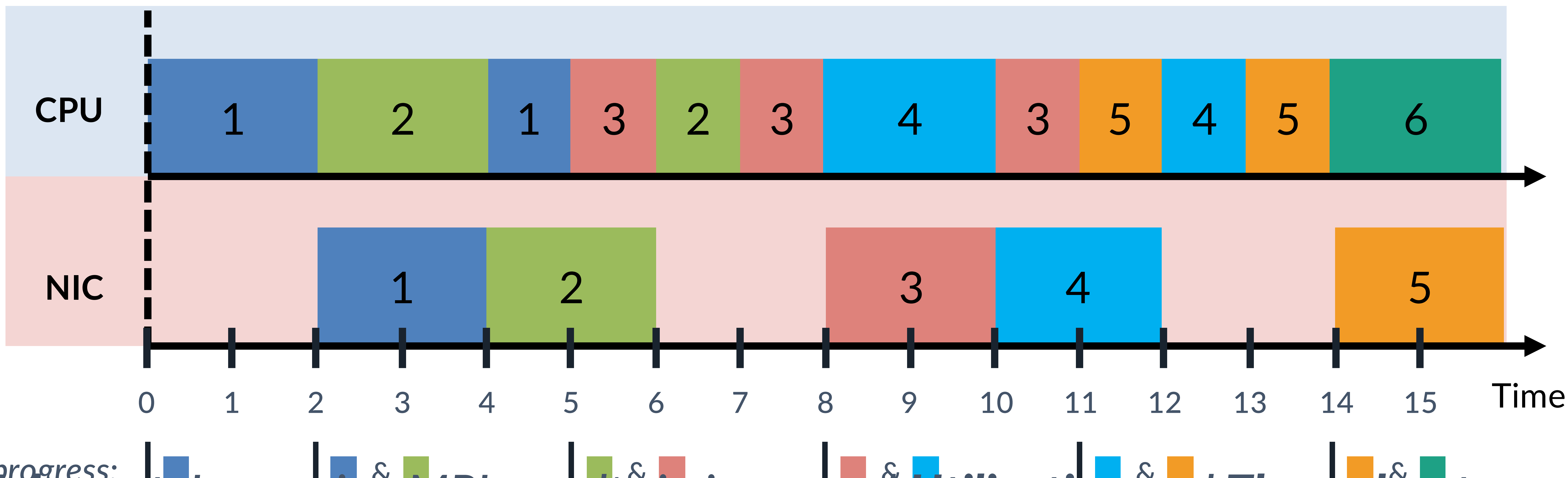
System Throughput = 1/5 req/tu = 0.2 req/tu

Capacity

when the party ends

Capacity: ***maximum throughput that can be sustained by the system.***

*With Multi-Programming Level (**MPL**) = 2:*

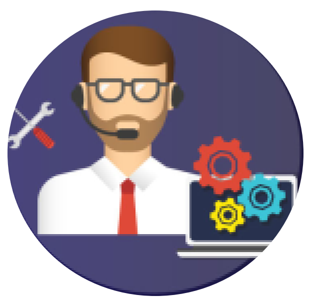


Take-out: *Increasing MPL results in increased **Utilization** and **Throughput***

*...until **one of the resources** reaches 100% utilization (**bottleneck**).*

*In this case, the system has reached its **Capacity**.*

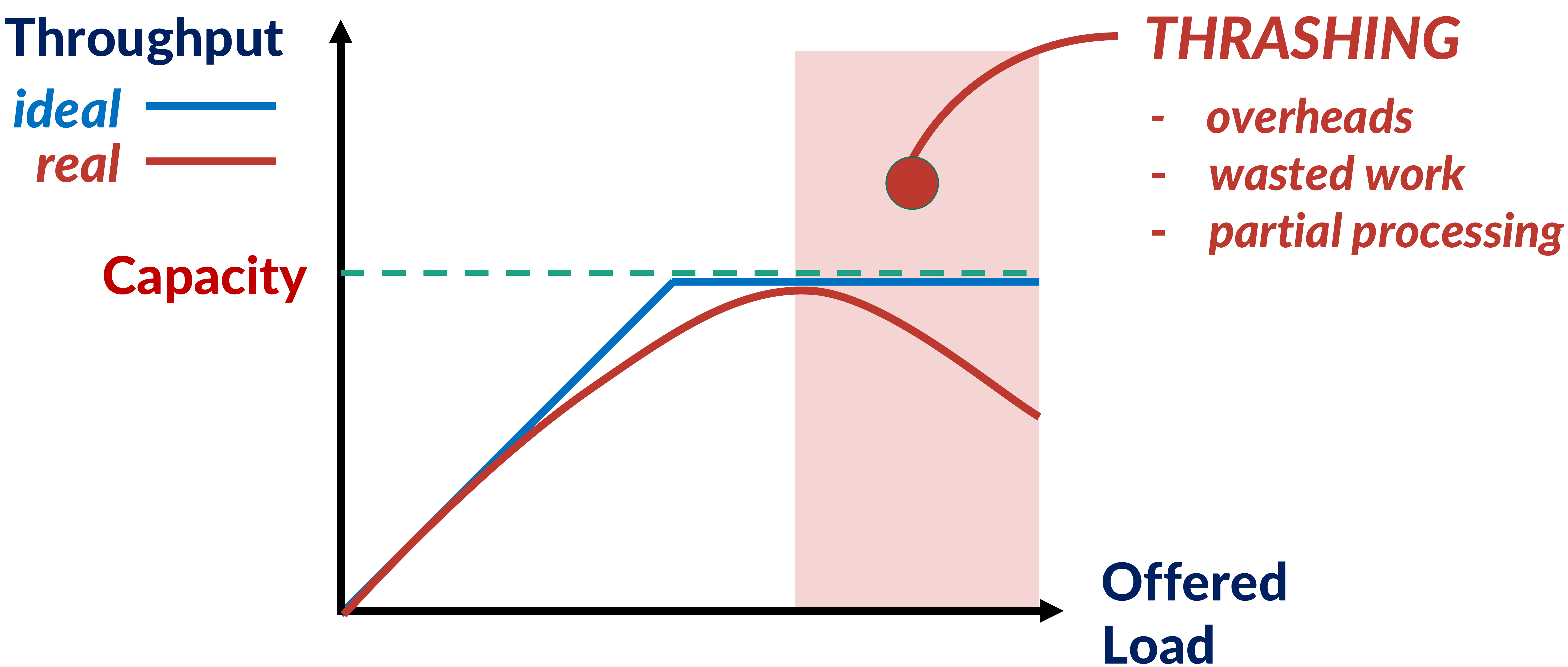
BOSTON UNIVERSITY



Capacity

& why reality is harsh

Capacity: ***maximum throughput that can be sustained by the system.***



Capacity & Bottlenecks

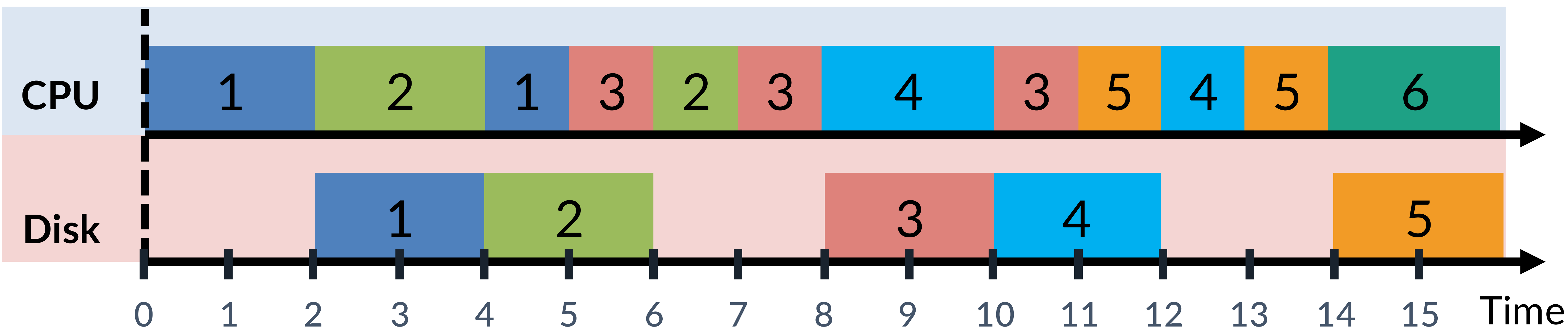
one cannot always blame the butler

Capacity: ***maximum throughput that can be sustained by the system.***

Bottleneck: ***resource whose utilization first reaches 100% (saturation) .***

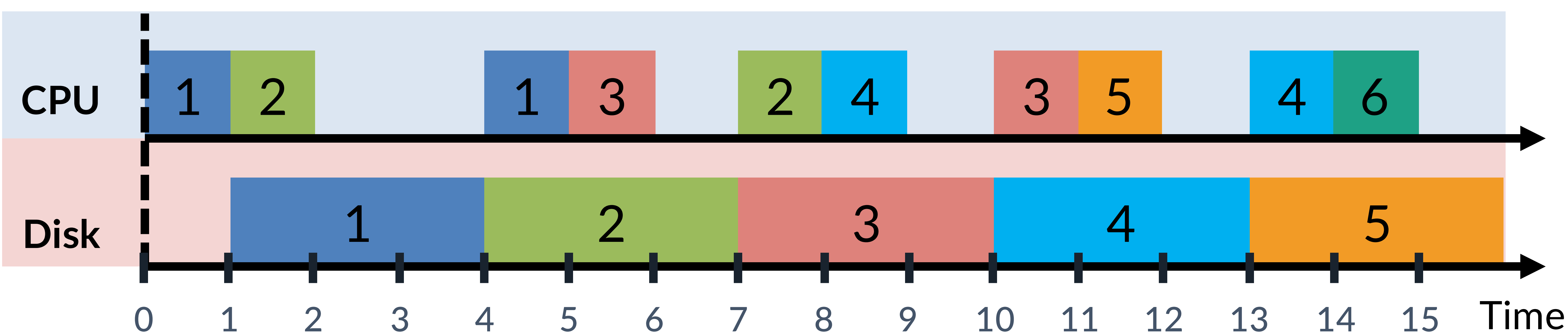
Load #1

CPU is bottleneck



Load #2

Disk is bottleneck

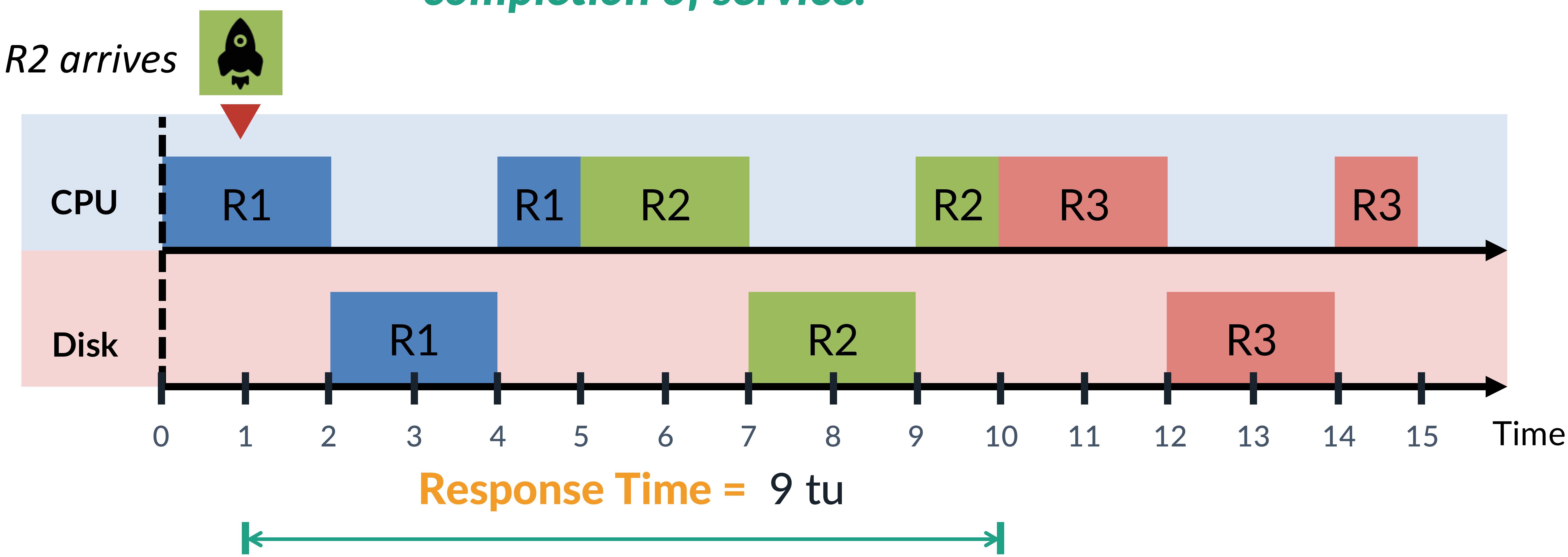


BOSTON UNIVERSITY

Response Time

time for a system trip

Response Time: ***time elapsed between request submission and completion of service.***

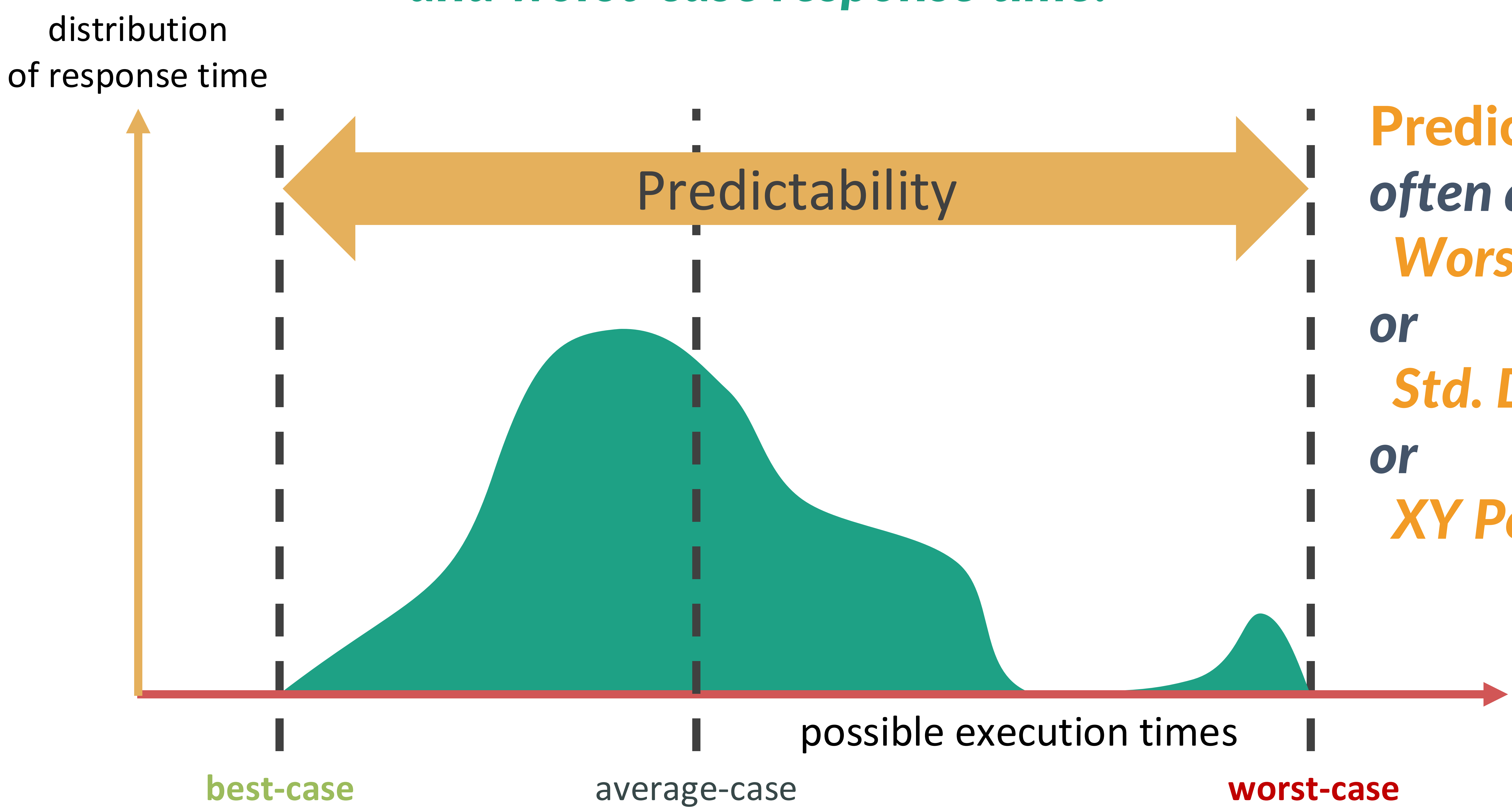


NOTE: Response Time = Turnaround Time = Latency

Predictability

i.e. how "wild" is your system

Predictability: ***Difference between best- or average-case response time and worst-case response time.***

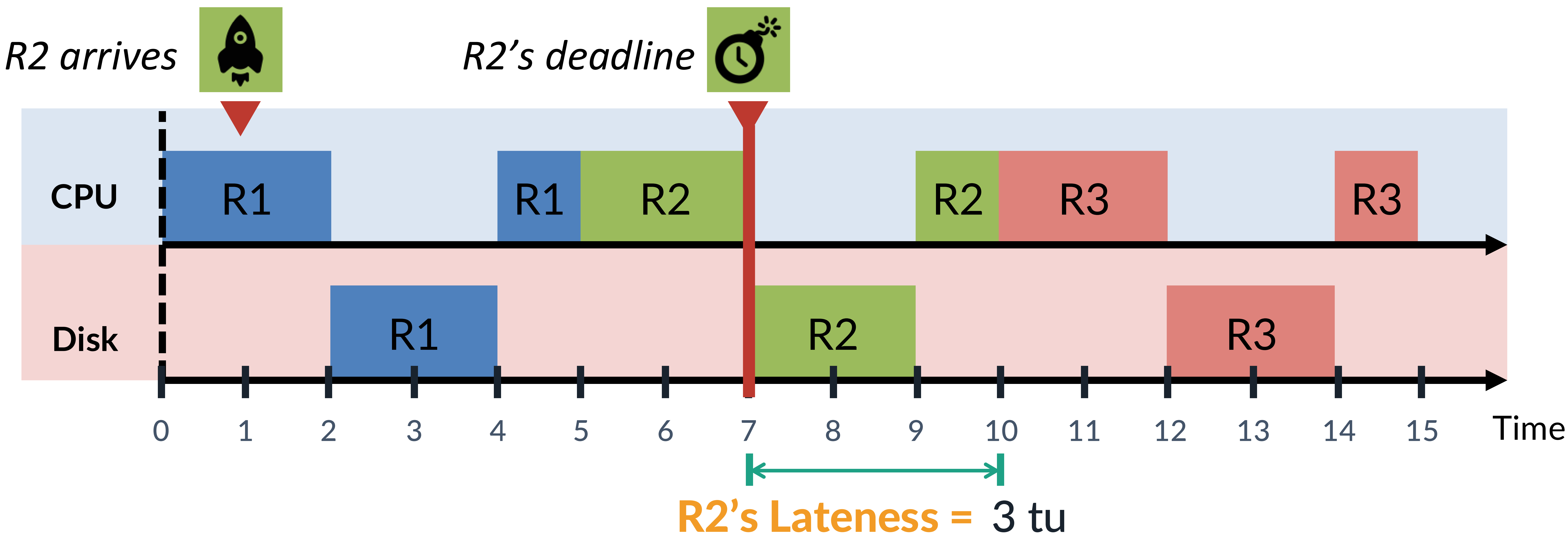


Predictability:
often as:
Worst - Best
or
Std. Dev.
or
XY Percentile

Lateness

how Italian is a system

Lateness: ***how late requests complete service with respect to their deadlines.***

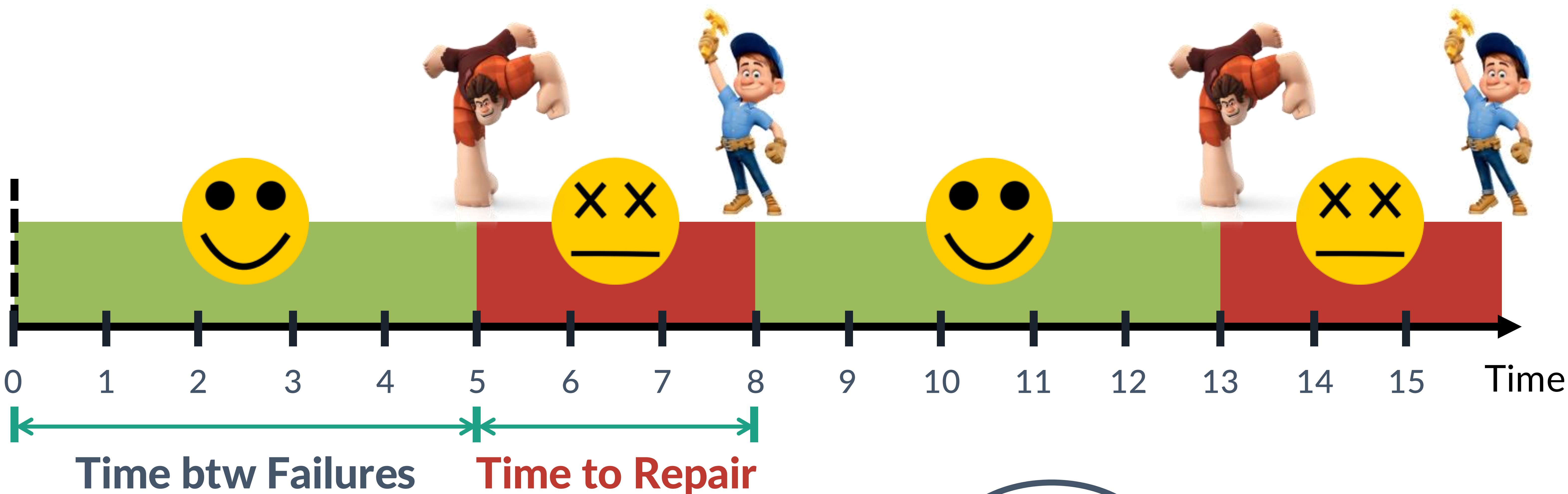


NOTE: For the entire system, compute average lateness

Availability

is the system up?

Availability: ***likelihood that the system will be "up" when a request is submitted.***



$$\text{Availability} = \frac{Up\ Time}{Total\ Time} = \frac{MTBF}{MTBF + MTTR}$$

Mean **Time btw Failures**

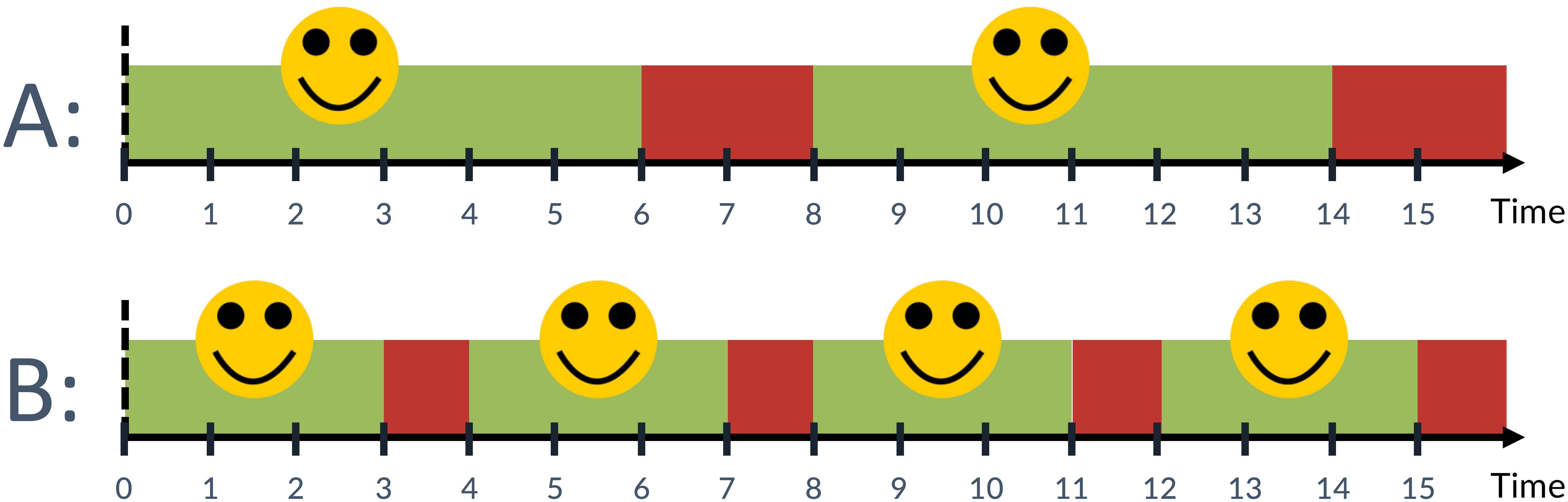
Mean **Time to Repair**

BOSTON UNIVERSITY

Reliability

is the system GOING TO BE up?

Reliability: ***likelihood that the system will be "up" for a certain amount of time.***



Same AVAILABILITY but *different* RELIABILITY

BOSTON UNIVERSITY

Performance Disclaimer

just like a medication's fine print

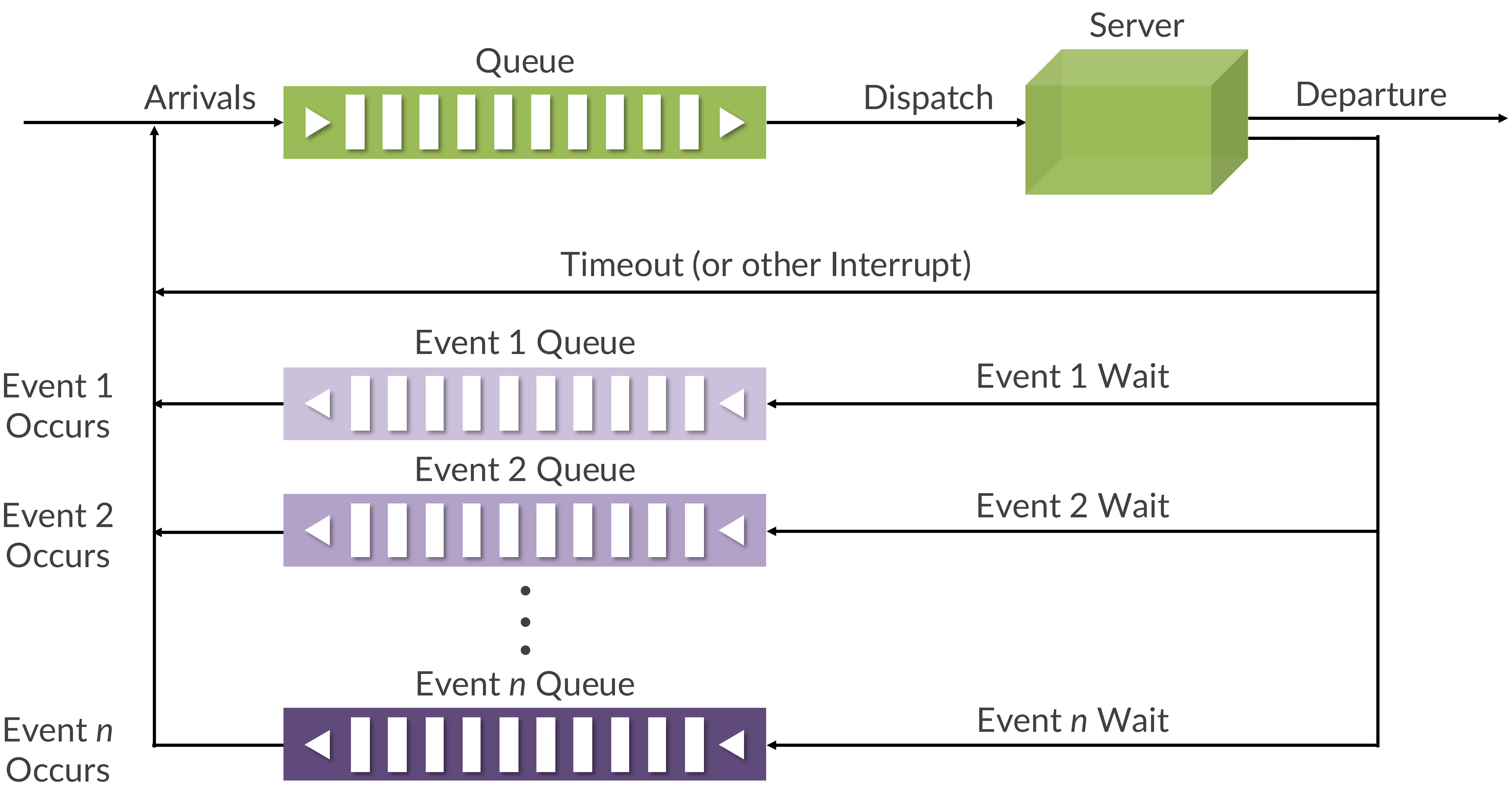
Performance metrics **abstract away** the **REAL behavior** of the system and can be *misleading*.

E.g.: *Average yearly temperature in Boston is 51°F*
Conclusion: ~~no need to but an A/C unit.~~

E.g.: *The new Intel CPU improves FLOPS by 30%*
Conclusion: ~~all my applications will be 30% faster.~~

A System Abstraction

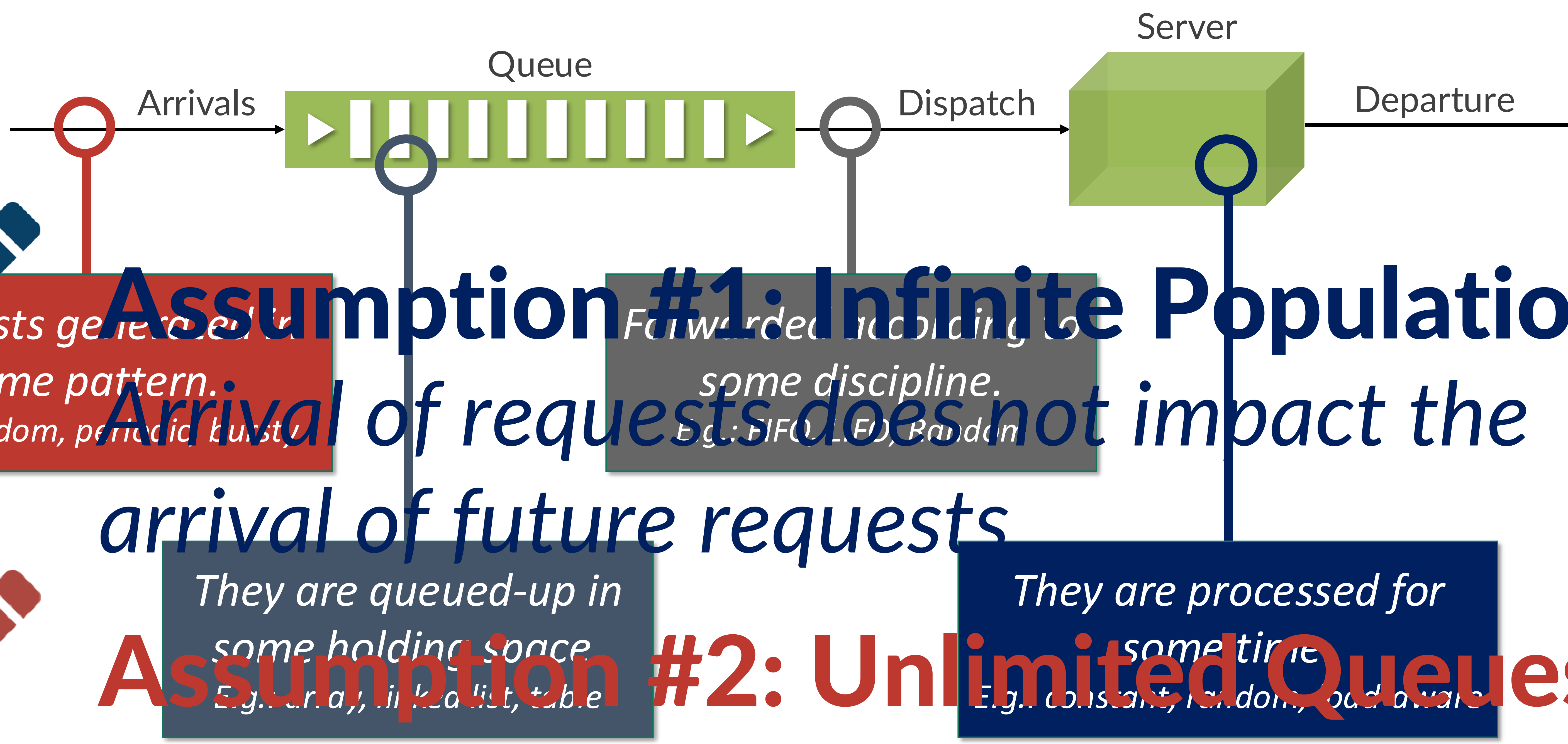
as network of queues



BOSTON UNIVERSITY

A System Abstraction

as network of queues



Assumption #1: Infinite Population

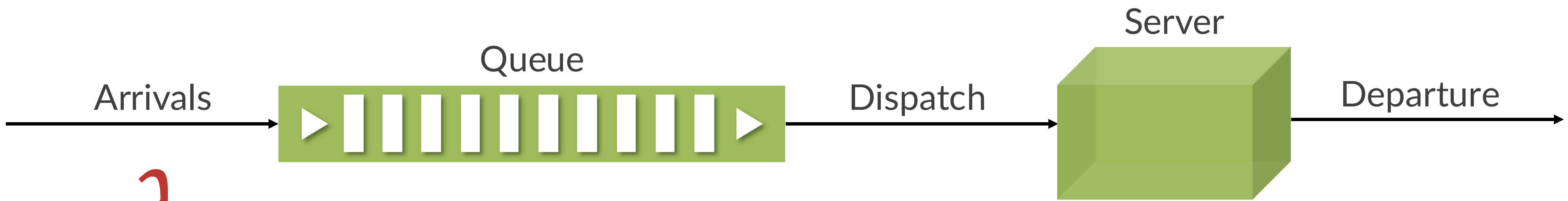
Arrival of requests does not impact the arrival of future requests

Assumption #2: Unlimited Queues

There is always a spot to “wait” for service. Queues never overflow.

A System Abstraction

let's talk about notation



λ *Arrival Rate*

w *items waiting*

T_w *waiting time*

T_s *service time*

ρ *utilization*

q *items in the whole system*

T_q *response time*

Universal Relationships

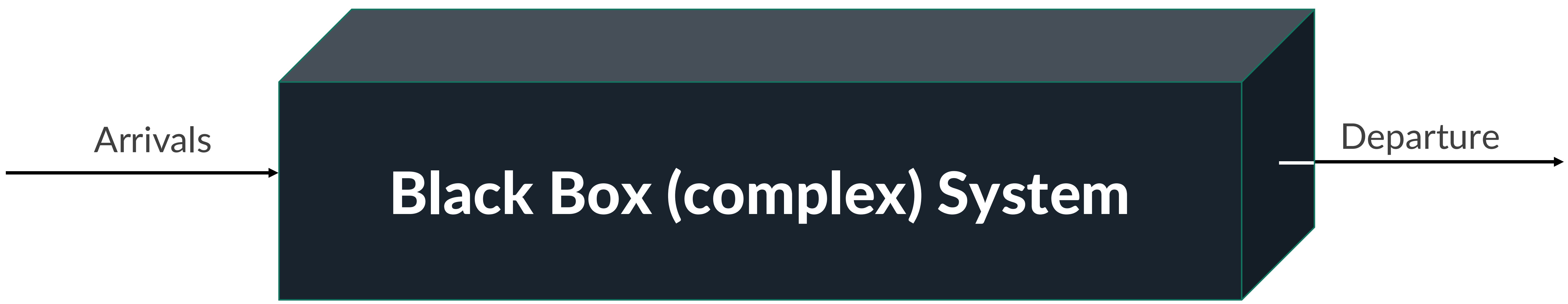
dropping some wisdom

utilization $\rho = \lambda \cdot T_s$ *valid if:* $\lambda < \frac{1}{T_s}$

response time $T_q = T_w + T_s$

Little's Law

the mighty



Assumption: Input flow rate matches output flow rate.

Implies: system at steady-state; no unaccounted deaths.

1 $q = \lambda \cdot T_q$

2 $w = \lambda \cdot T_w$

Universal Relationships

dropping some wisdom

$$T_q = T_w + T_s$$

$$\frac{q}{\lambda} = \frac{w}{\lambda} + T_s$$

$$q = w + T_s \cdot \lambda$$

$$\boxed{q = w + \rho}$$

Little's Law, I choose you

BOSTON UNIVERSITY