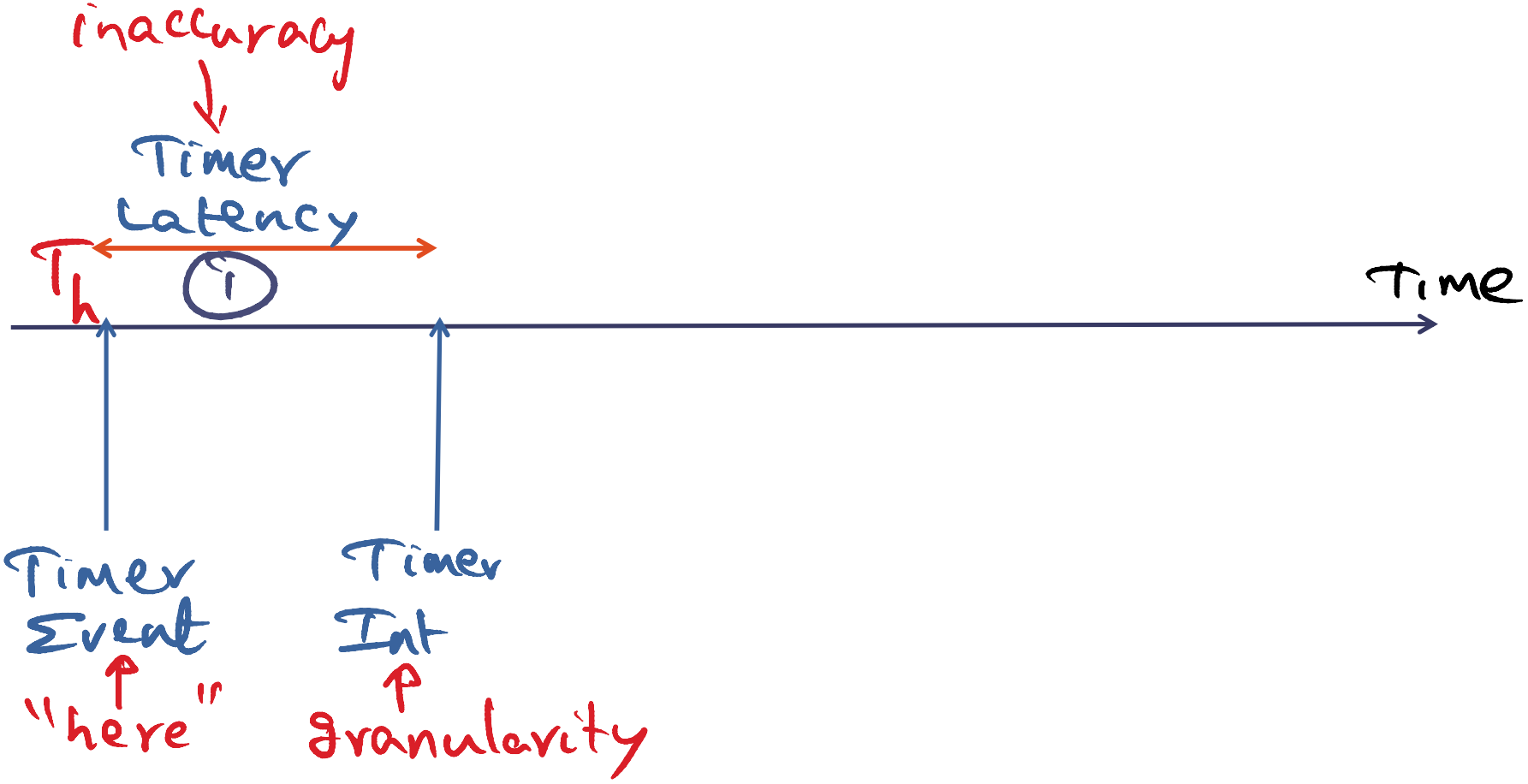
Lesson outline

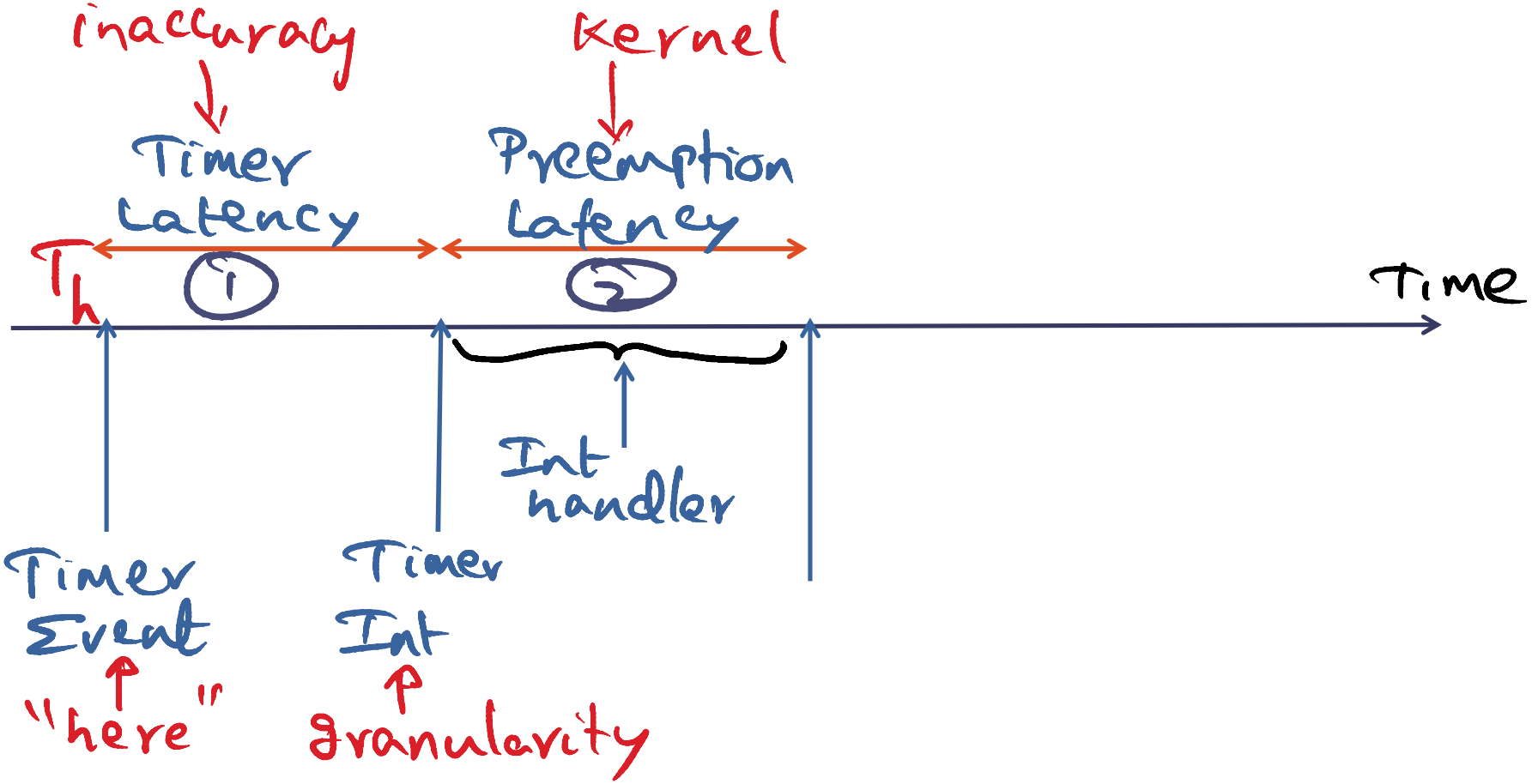
→ TS- Linux

Persistent Temporal Streams

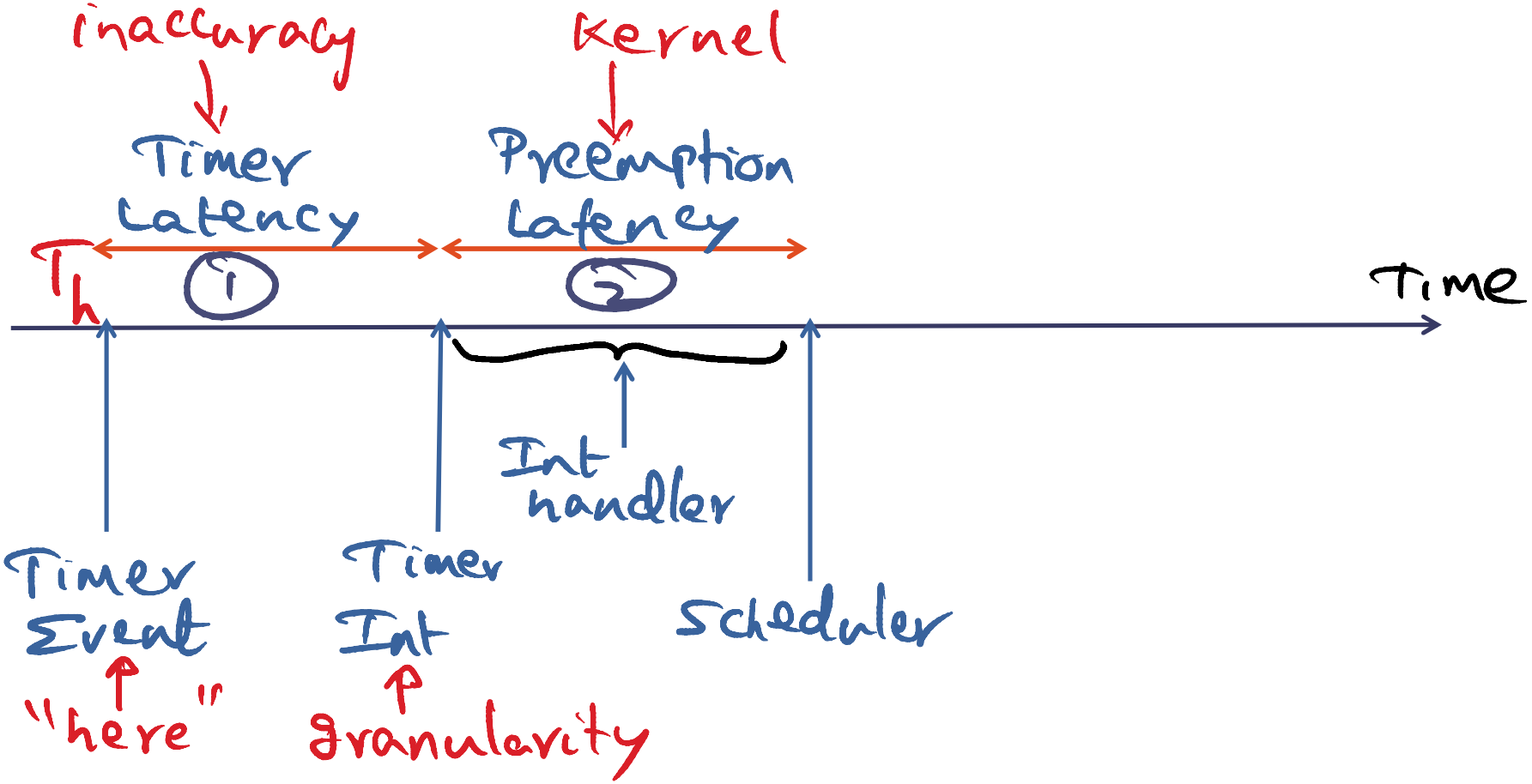
Sources of Latency



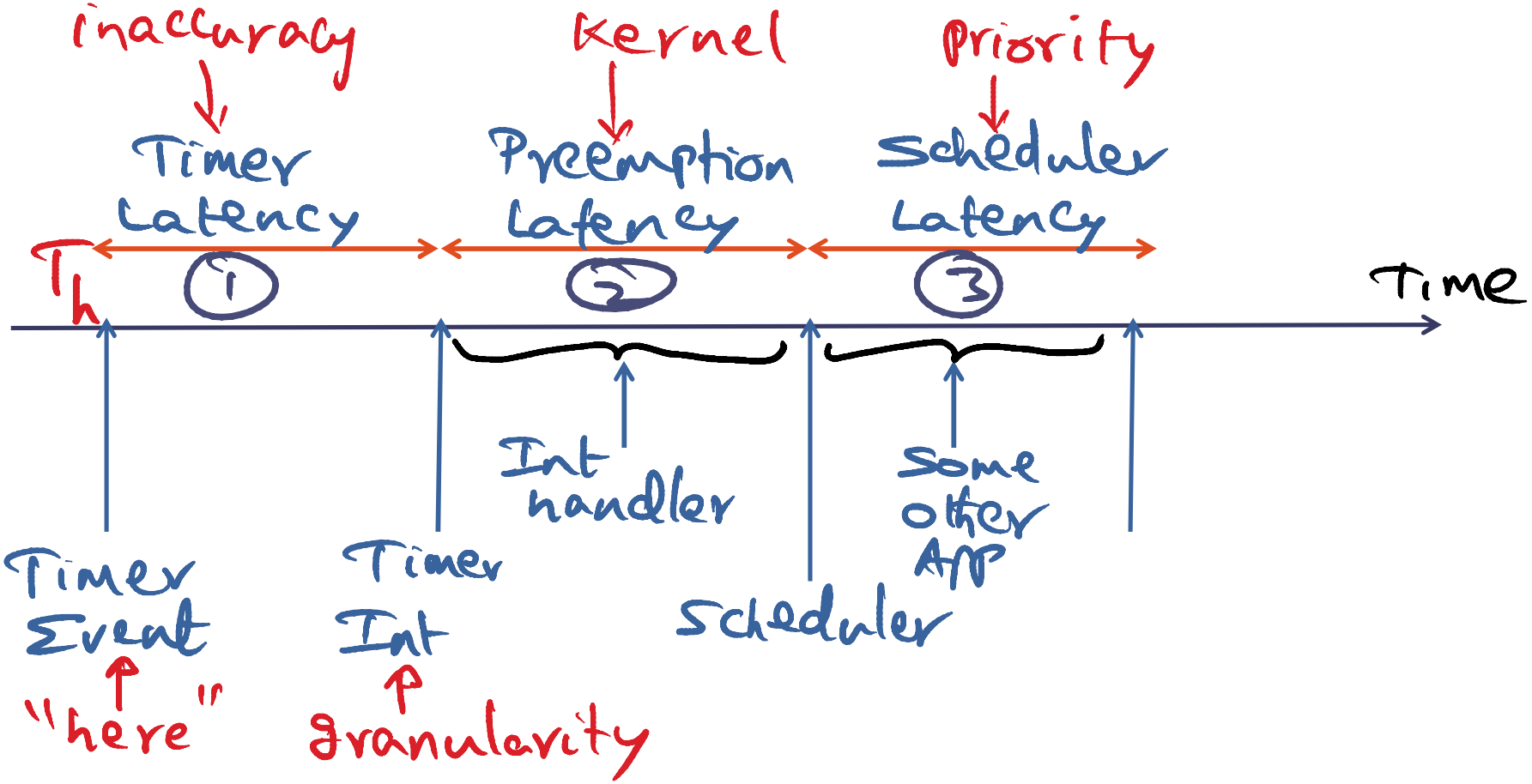
Sources of Latency



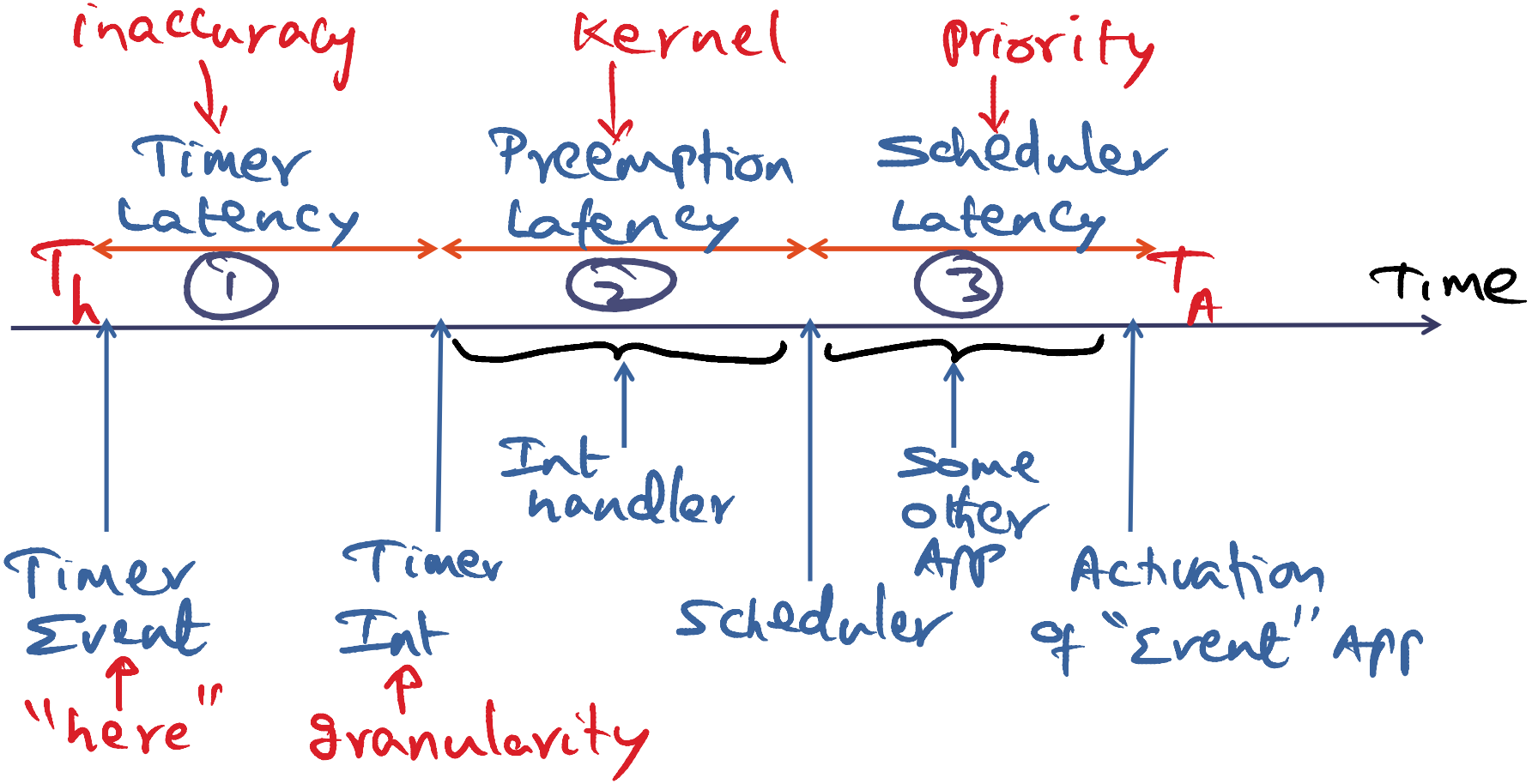
Sources of Latency



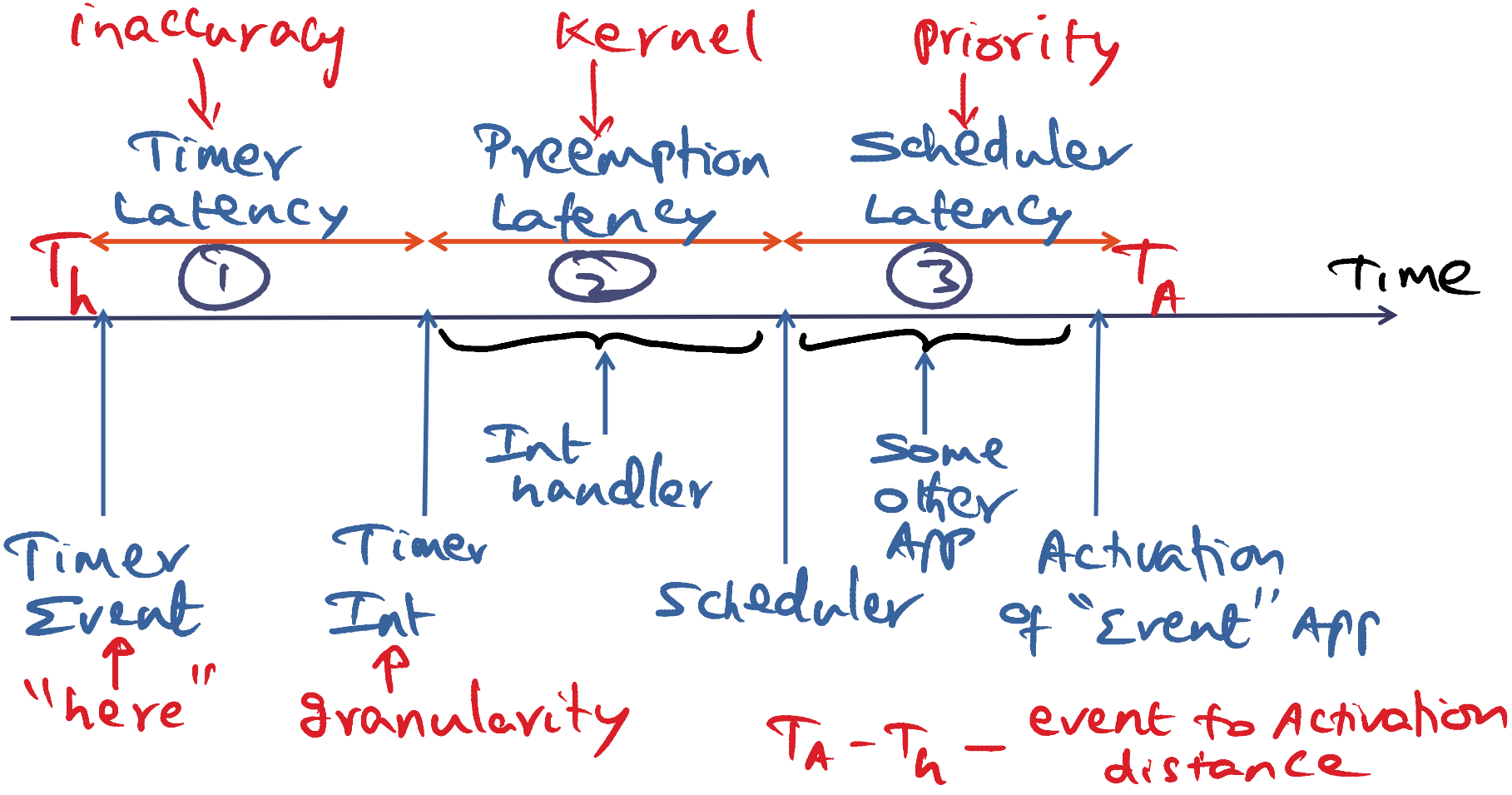
Sources of Latency



Sources of Latency



Sources of Latency



Choice of Timer	Pro	Con
Periodic	periodicity	event recognition latency

Choice of Timer	Pro	Con
Periodic	periodicity	event recognition latency
One-shot	timely	overhead

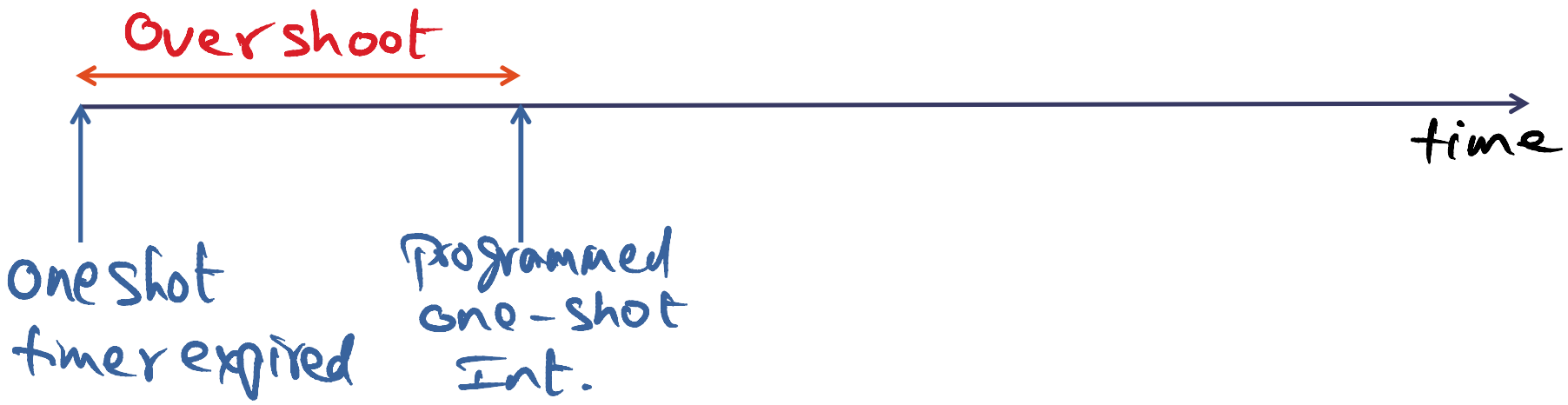
Choice of Timer	Pro	Con
Periodic	periodicity	event recognition latency
One-shot	timely	overhead
Soft	reduced overhead	Polling overhead, latency

Choice of Timer	Pro	Con
Periodic	periodicity	event recognition latency
One-shot	timely	overhead
Soft	reduced overhead	Polling overhead, latency
Firm	Combines all of the above	

Firm Timer Design

Accurate timing with low overhead

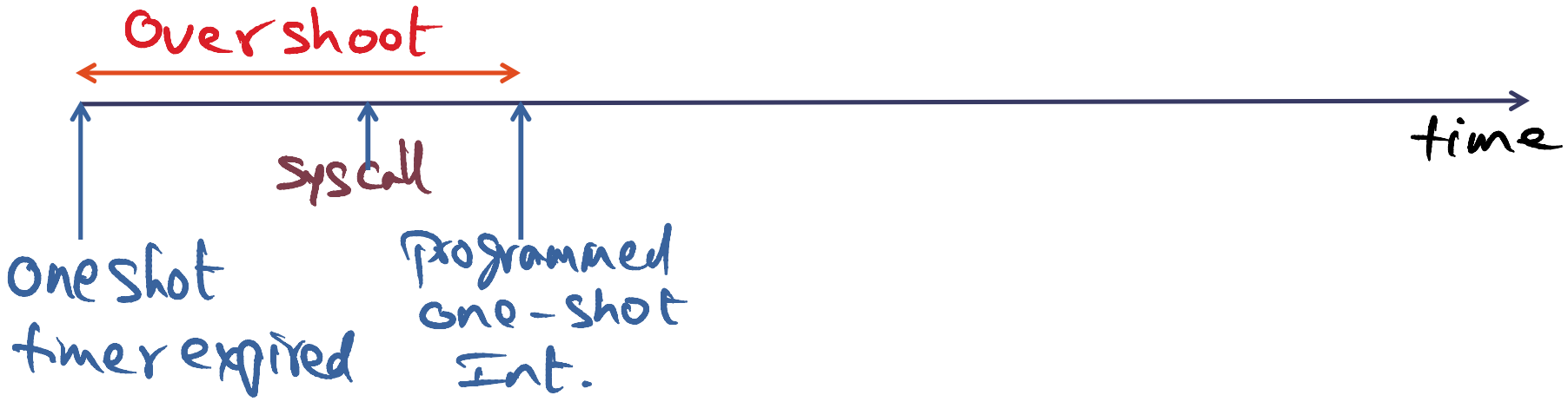
- Combine one-shot and soft timers
- Overshoot: Knob between hard and soft timers



Firm Timer Design

Accurate timing with low overhead

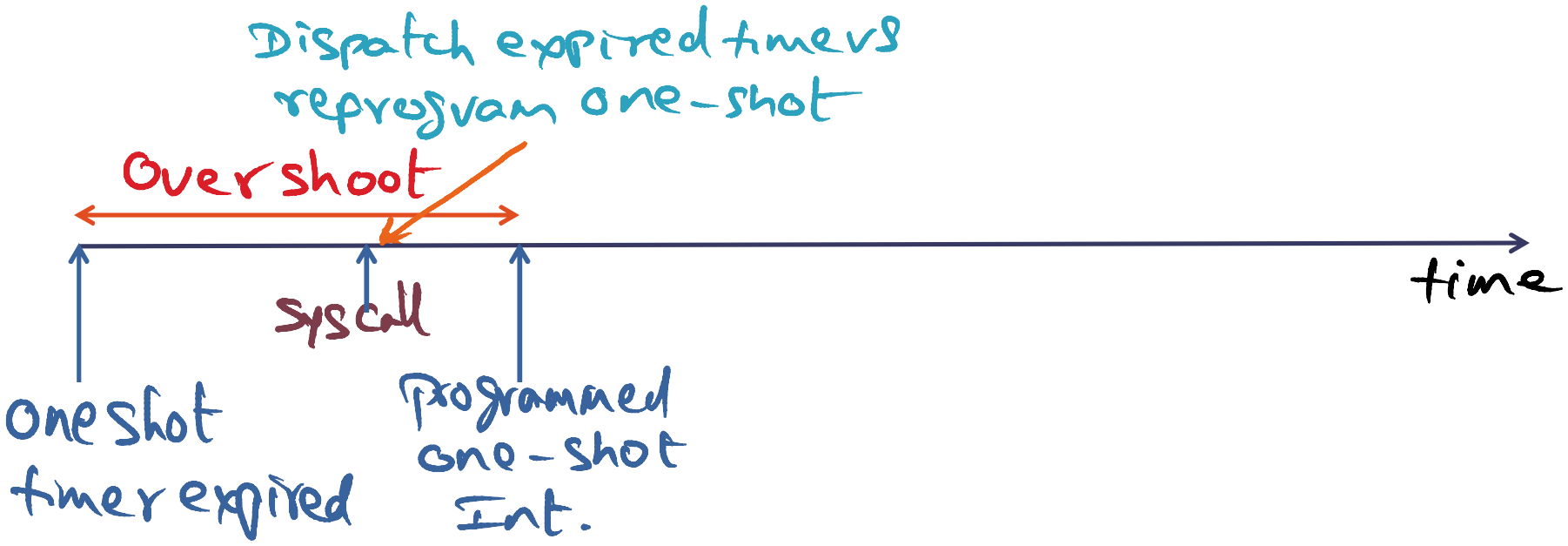
- Combine one-shot and soft timers
- Overshoot: Knob between hard and soft timers



Firm Timer Design

Accurate timing with low overhead

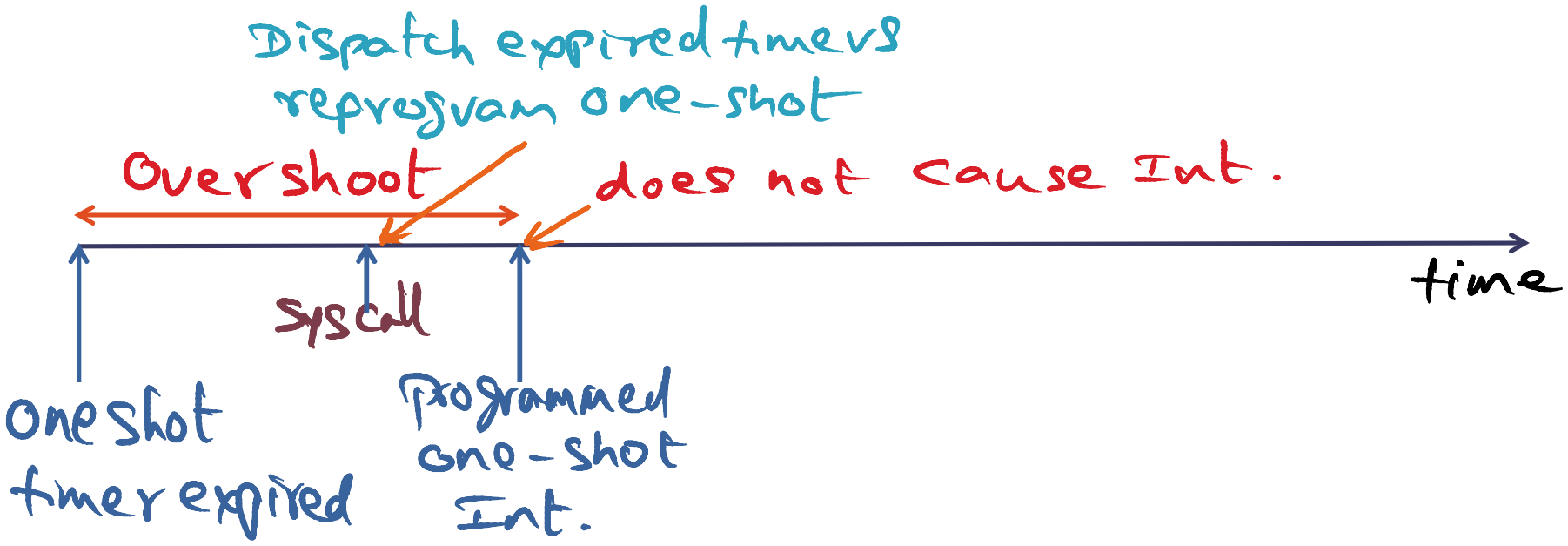
- Combine one-shot and soft timers
- Overshoot: Knob between hard and soft timers



Firm Timer Design

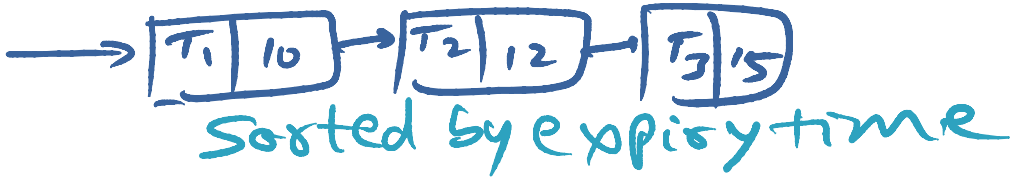
Accurate timing with low overhead

- Combine one-shot and soft timers
- Overshoot: Knob between hard and soft timers



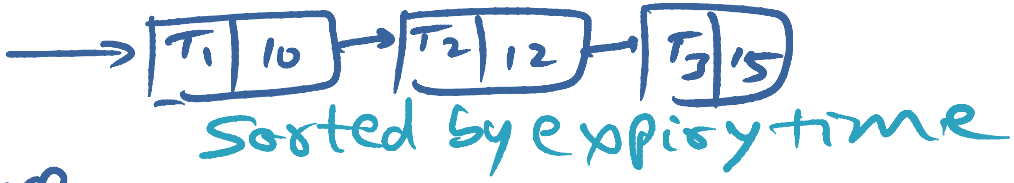
Firm Timer Implementation

Timer-Q data structure



Firm Timer Implementation

Timer-Q data structure



Apic timer hardware

- reprogramming: few cycles

Firm Timer Implementation

Timer-Q data structure

Sorted by expiry time

Apic timer hardware

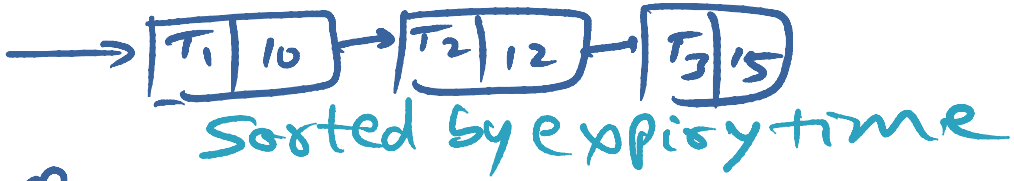
- reprogramming: few cycles

Soft timers

- eliminates need for fielding oneshot int

Firm Timer Implementation

Timer-Q data structure



Apic timer hardware

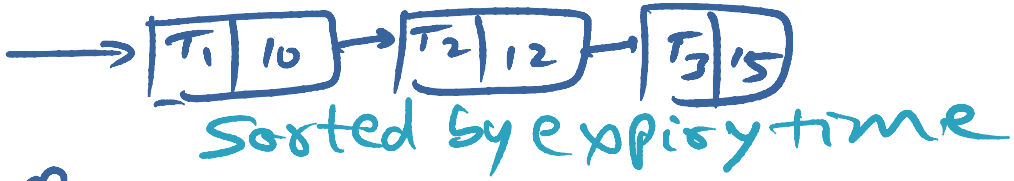
- reprogramming: few cycles

Soft timers

- eliminates need for fielding oneshot int

Firm Timer Implementation

Timer-Q data structure



Apic timer hardware

- reprogramming: few cycles

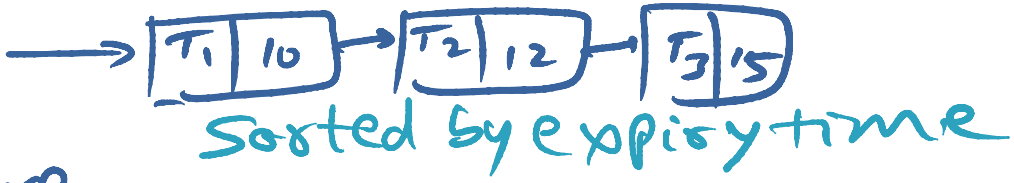
Soft timers

- eliminates need for fielding oneshot int

Long one shot distance

Firm Timer Implementation

Timer-Q data structure



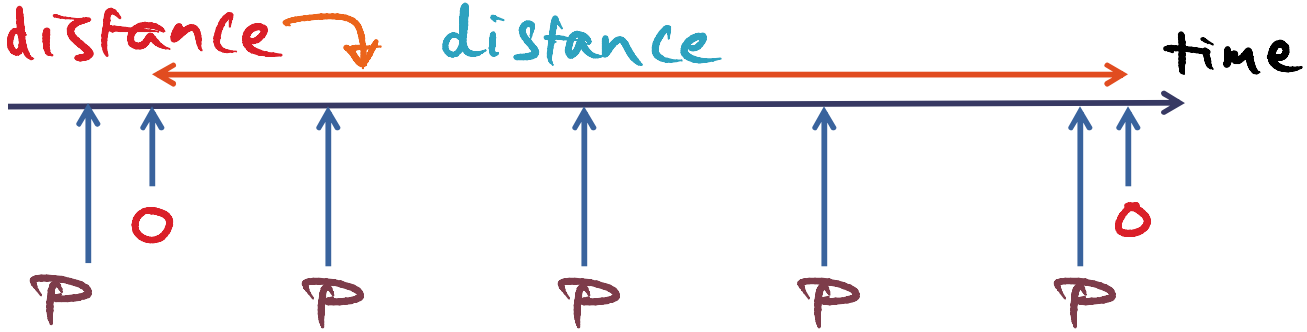
Apic timer hardware

- reprogramming: few cycles

Soft timers

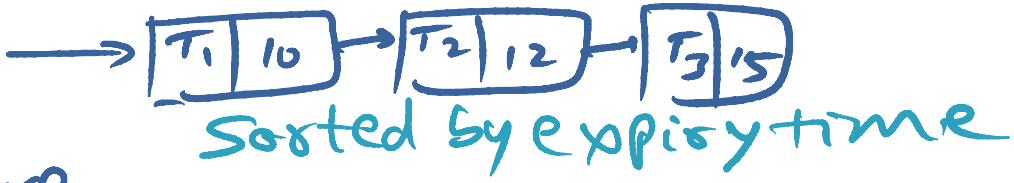
- eliminates need for fielding oneshot int

Long one shot distance



Firm Timer Implementation

Timer-Q data structure



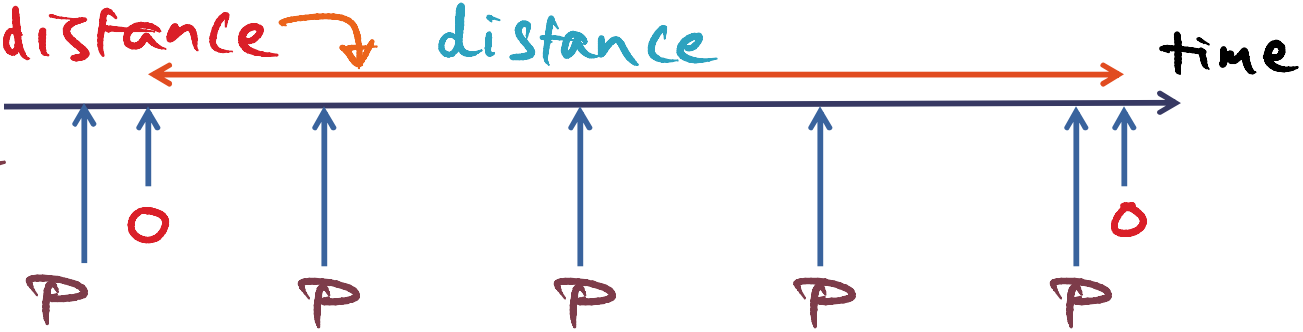
Apic timer hardware

- reprogramming: few cycles

Soft timers

- eliminates need for fielding oneshot int

Long one shot distance ⇒ use periodic



Today

- RT + mm (Lesson 10 review)
  - * TS-Linux
  - * PTS

Final week

- Security (Lesson 11)
  - * Terminologies
  - * Security in AFS

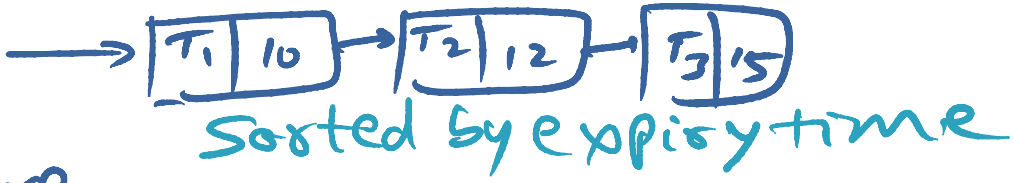
Group Activity

* Sources of latency in timer events?

* Solution approach for each Source?

Firm Timer Implementation

Timer-Q data structure



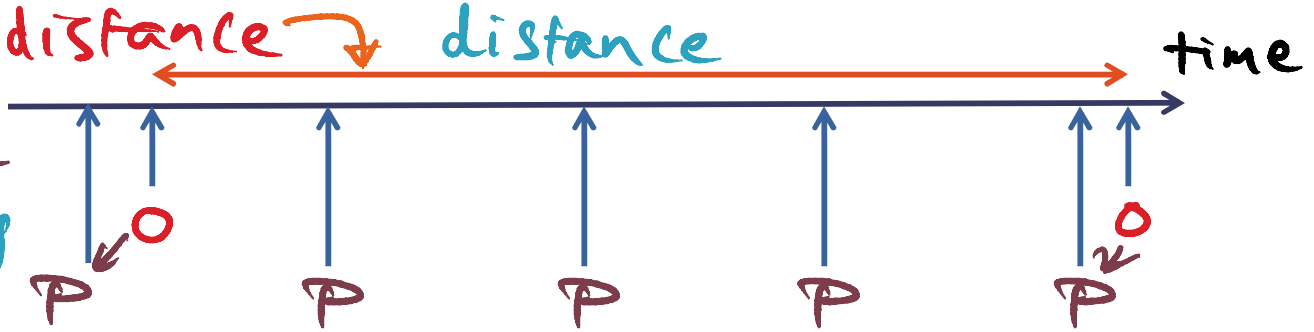
Apic timer hardware

- reprogramming: few cycles

Soft timers

- eliminates need for fielding oneshot int

Long one shot distance ⇒ use periodic dispatch @ preceding Periodic



Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel

Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel
- Allow preemption anytime kernel not manipulating shared data structures

Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel
- Allow preemption anytime kernel not manipulating shared data structures

Lock-Breaking Preemptible Kernel

- Combines the above two ideas

Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel
- Allow preemption anytime kernel not manipulating shared data structures

Lock-Breaking Preemptible Kernel

- Combines the above two ideas

```
acq(C)
  :
rel(C)
```

Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel
- Allow preemption anytime kernel not manipulating shared data structures

Lock-Breaking Preemptible Kernel

- Combines the above two ideas

```
acq()                    acq()
  :                        manipulate shared data
rel()        change      rel()
             to          reacq()
                           :
                         rel()
```

Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel
- Allow preemption anytime kernel not manipulating shared data structures

Lock-Breaking Preemptible Kernel

- Combines the above two ideas

```
acq()
  :
rel()
```

change to

```
acq()
  manipulate shared data
rel() → preempt kernel
acq()
  :
rel()
```

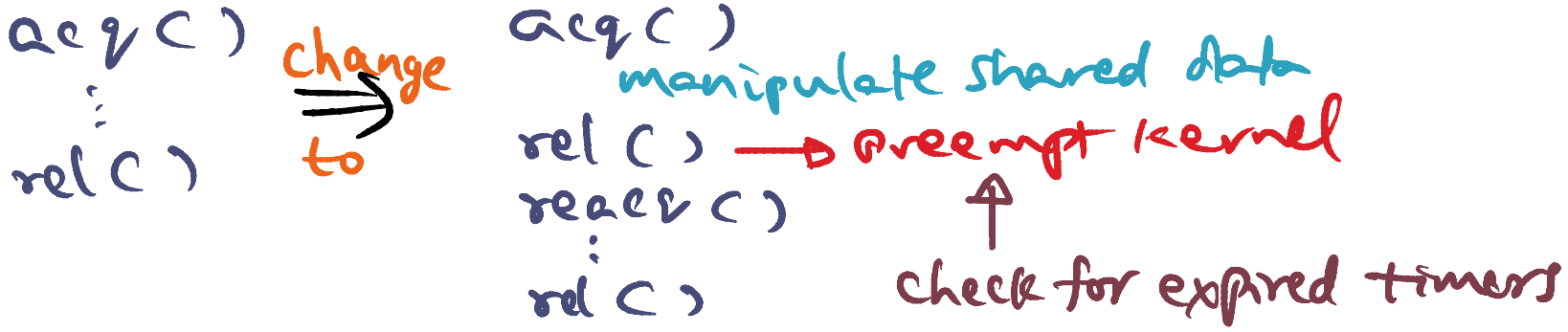
Reducing Kernel Preemption Latency

Approaches

- Explicit insertion of preemption points in kernel
- Allow preemption anytime kernel not manipulating shared data structures

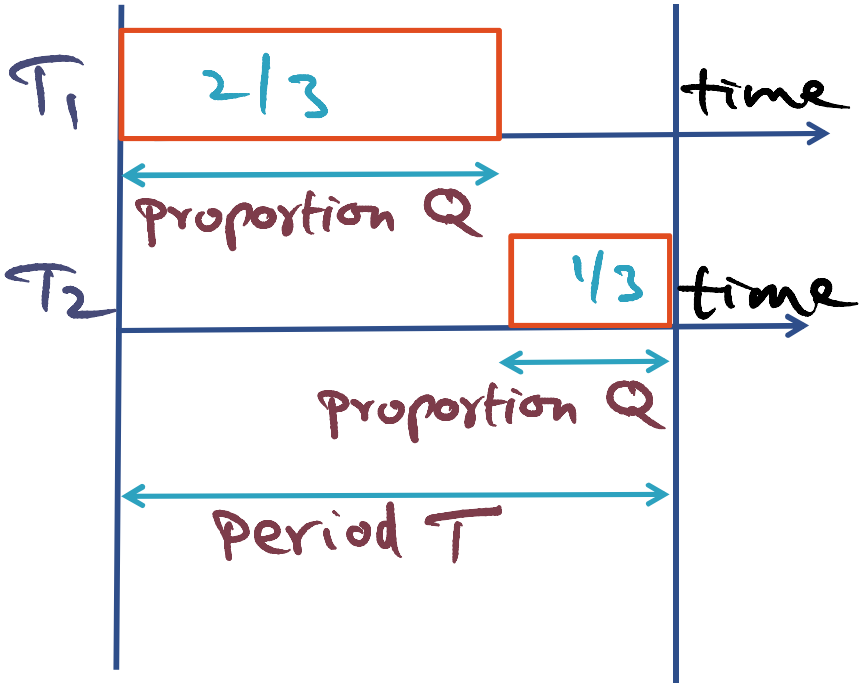
Lock-Breaking Preemptible Kernel

- Combines the above two ideas



Reducing scheduling latency

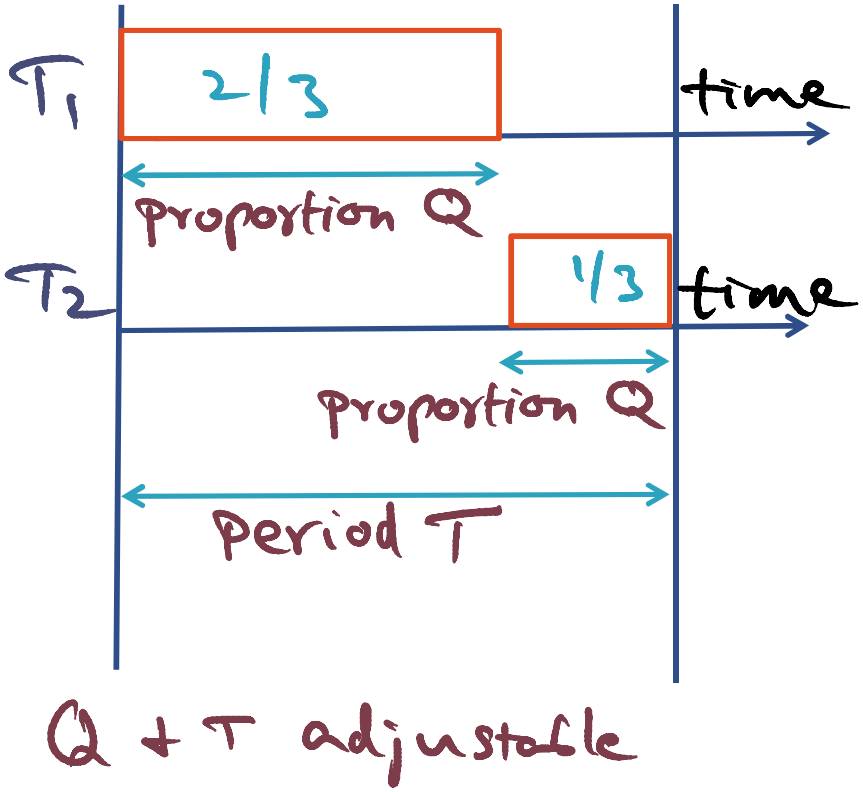
Proportional Period sched



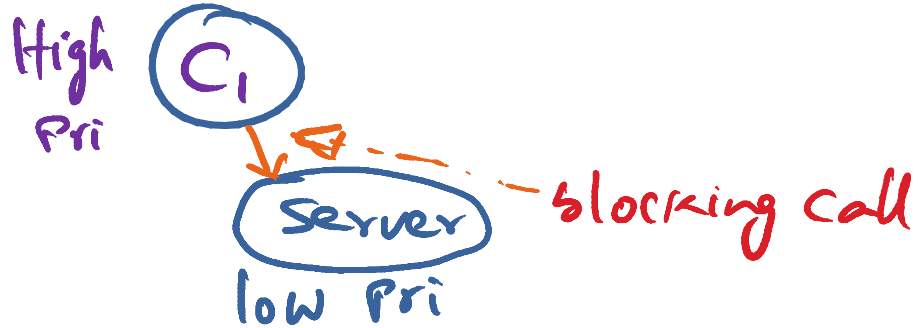
Q + T adjustable

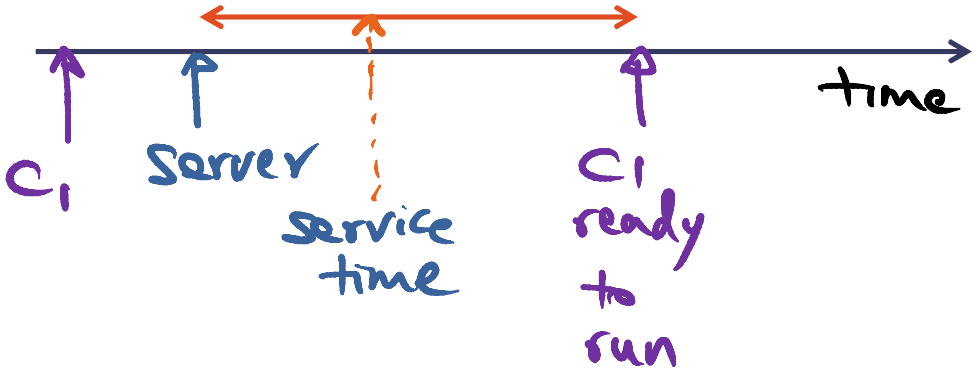
Reducing scheduling latency

Proportional Period sched



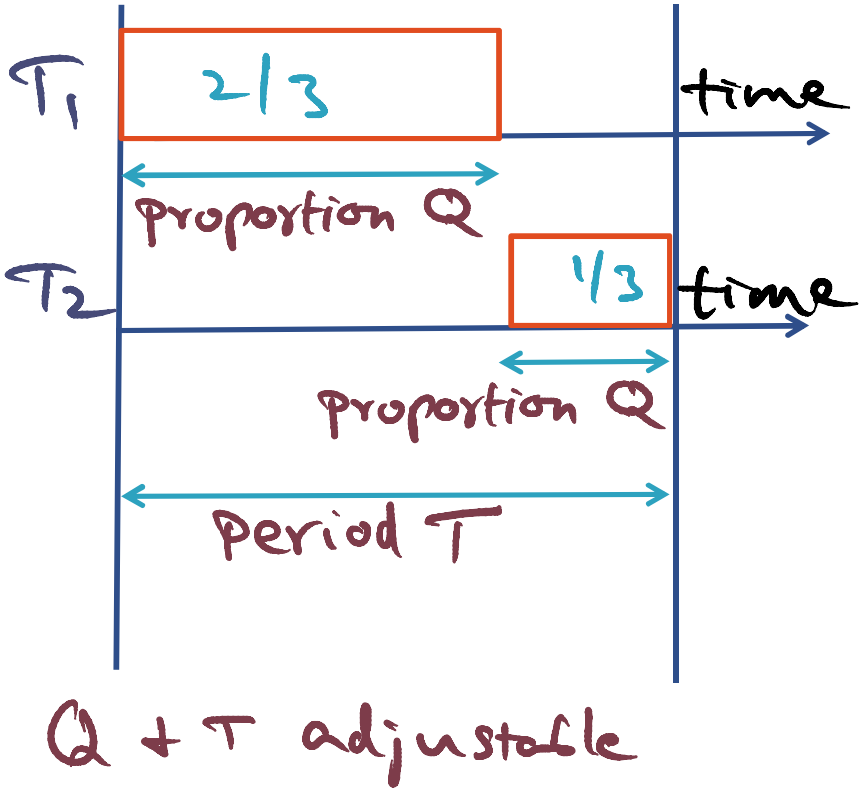
Q + T adjustable



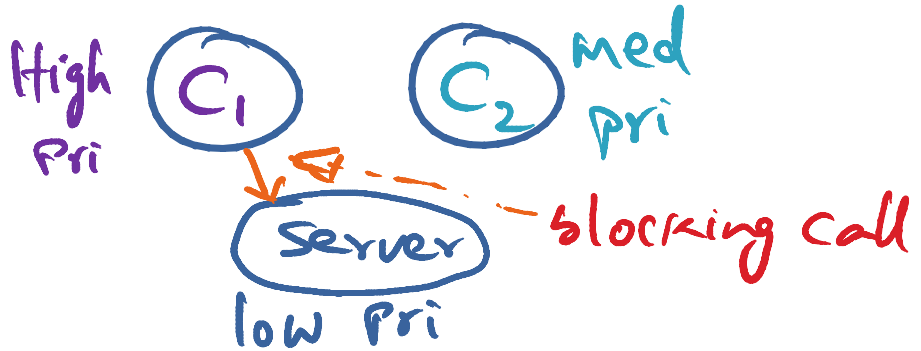


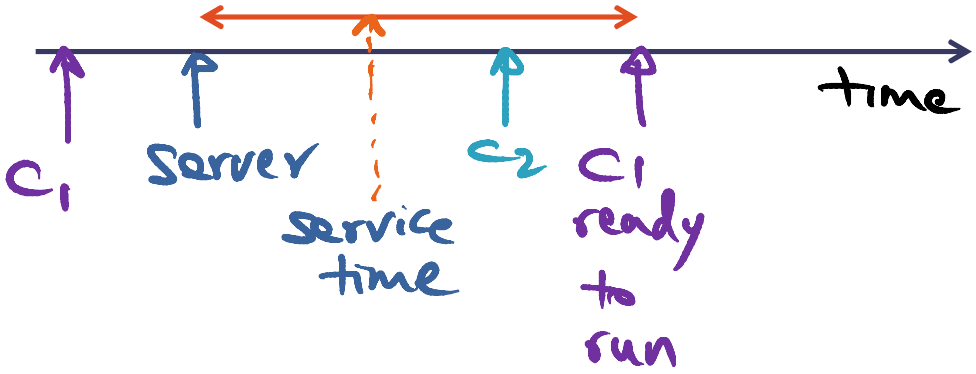
Reducing scheduling latency

Proportional Period sched



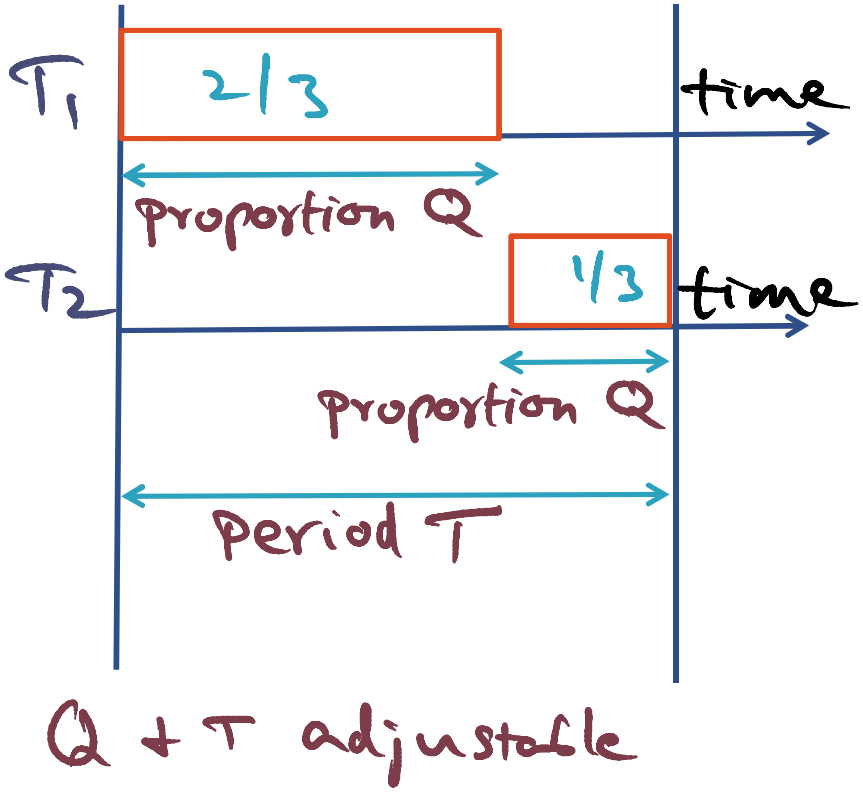
Q + T adjustable



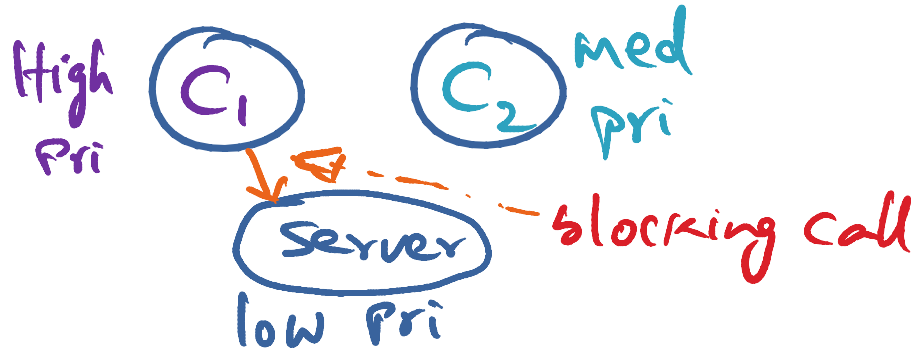


Reducing scheduling latency

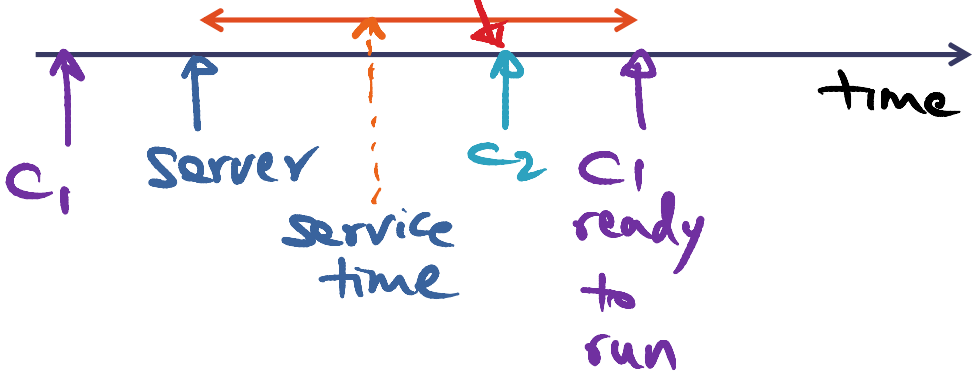
Proportional Period sched



Q + T adjustable

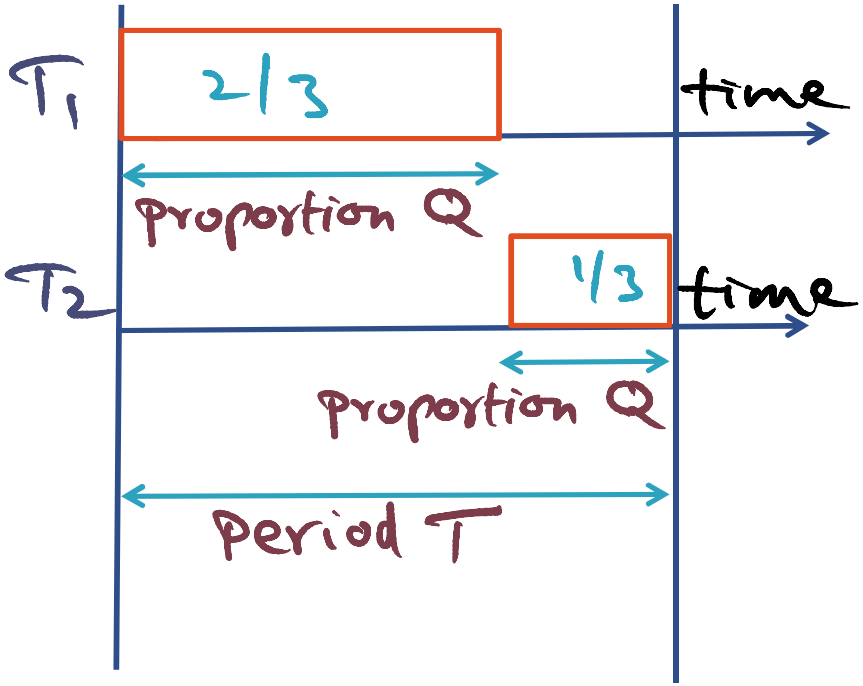


C2 preempts server
⇒ priority inversion for C1



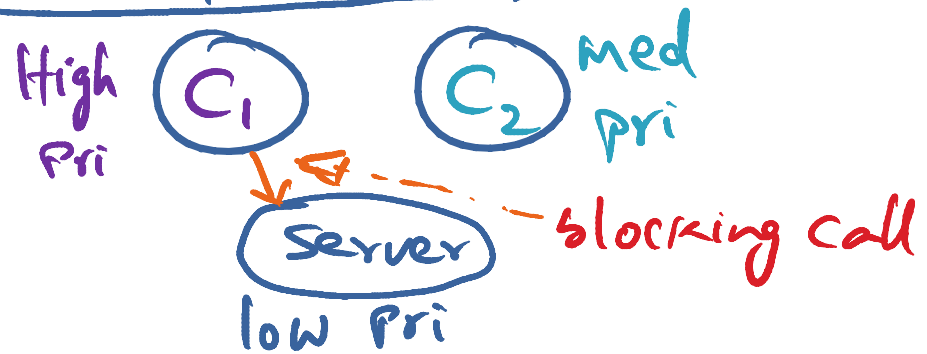
Reducing scheduling latency

Proportional Period sched

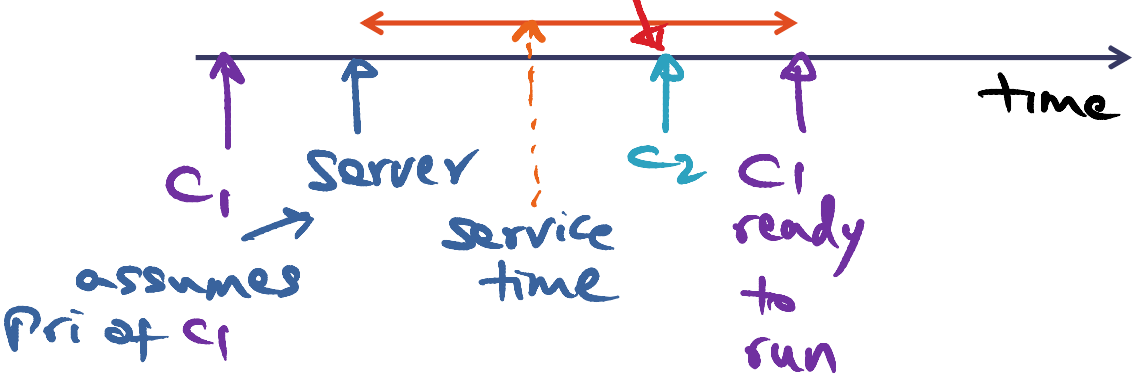


Q + T adjustable

Priority-based sched

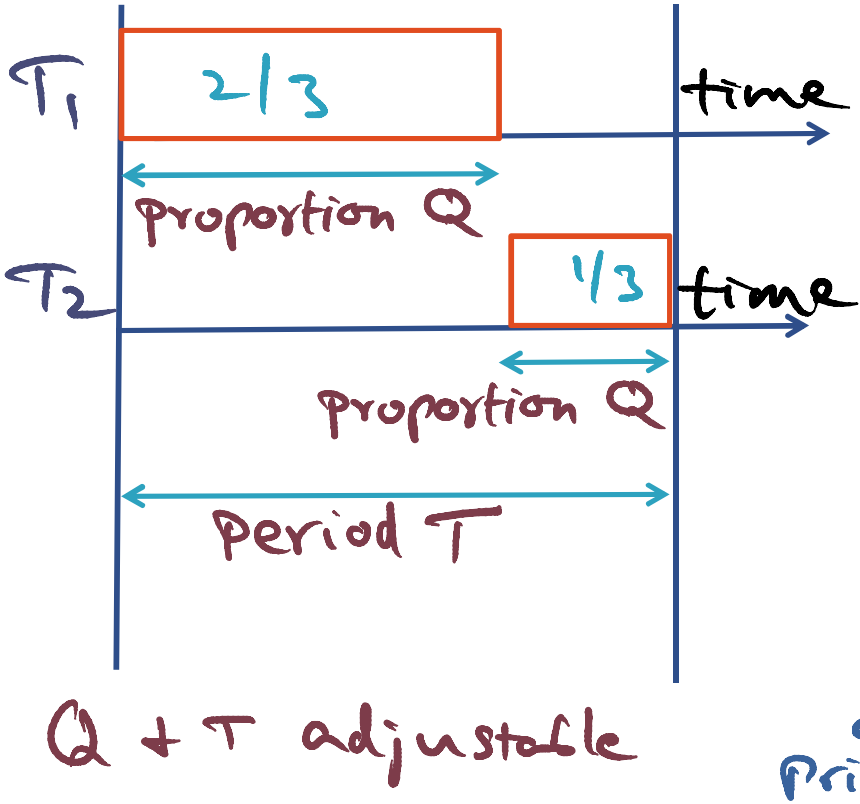


C2 preempts server
⇒ priority inversion for C1



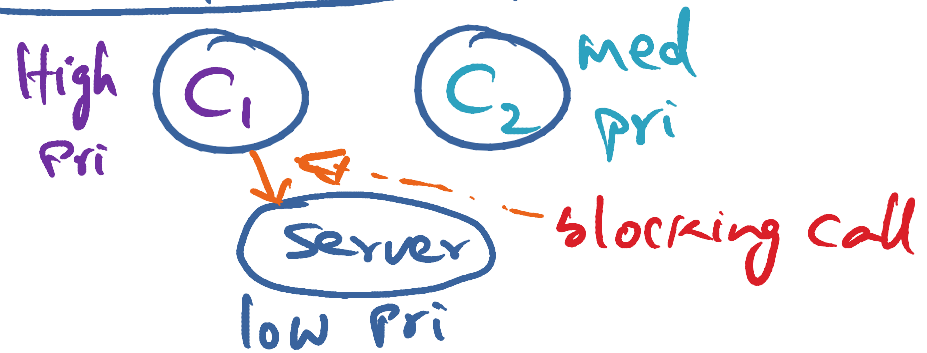
Reducing scheduling latency

Proportional Period sched

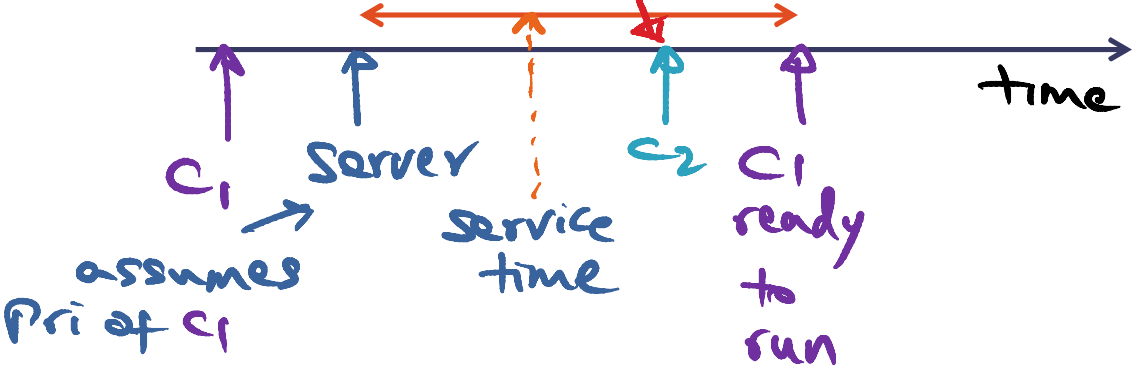


Q + T adjustable

Priority-based sched

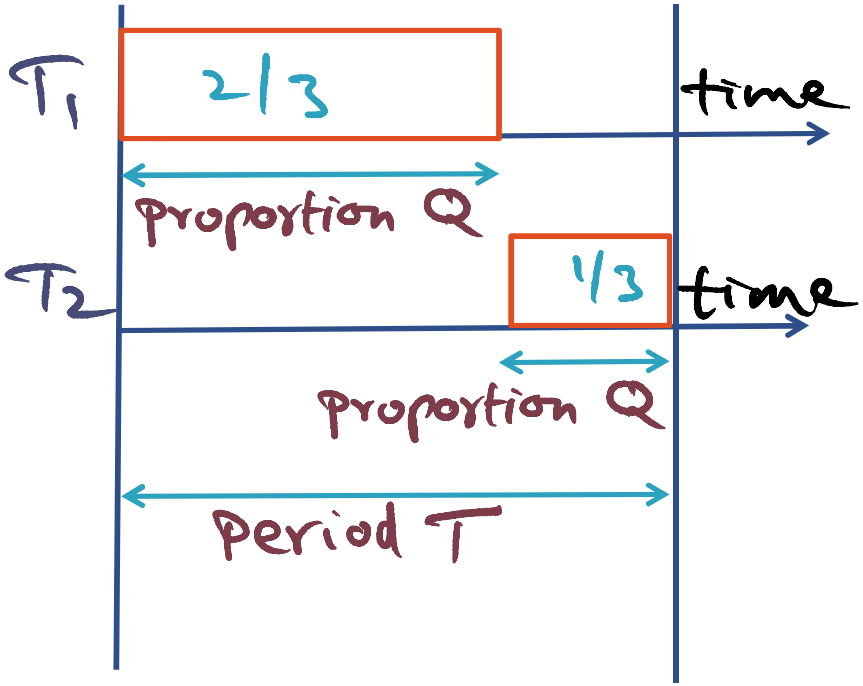


C2 ~~preempts~~ server
⇒ ~~priority~~ inversion for C1



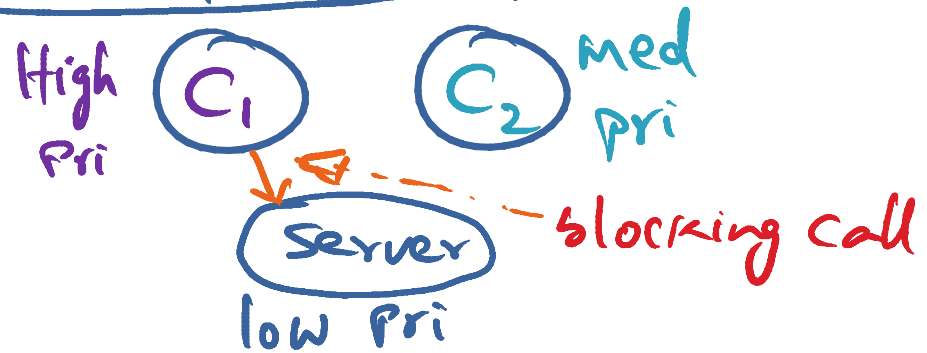
Reducing scheduling latency

Proportional Period sched

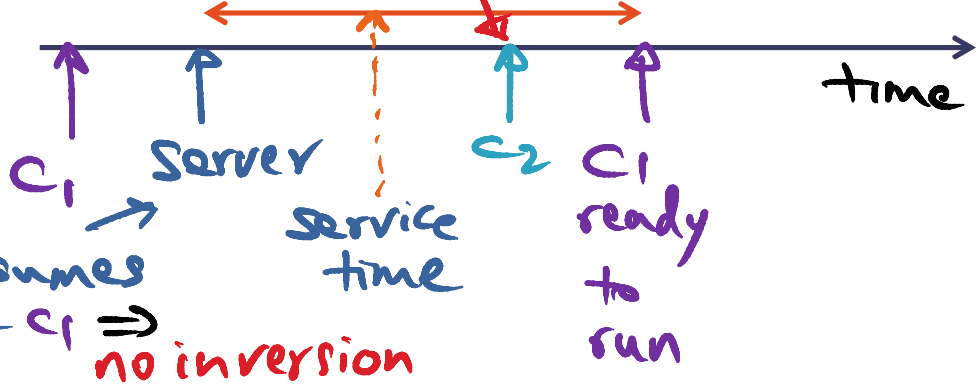


Q + T adjustable

Priority-based sched



C2 ~~preempts~~ server
⇒ ~~priority~~ inversion for C1



Key takeaways

- TS-Linux provides QoS guarantees for real-time apps running on commodity OS like Linux.
- Admission control using the proportional-period scheduling TS-Linux ensures that throughput-oriented tasks are not shut out from getting CPU time.