



CompSci 401: Cloud Computing

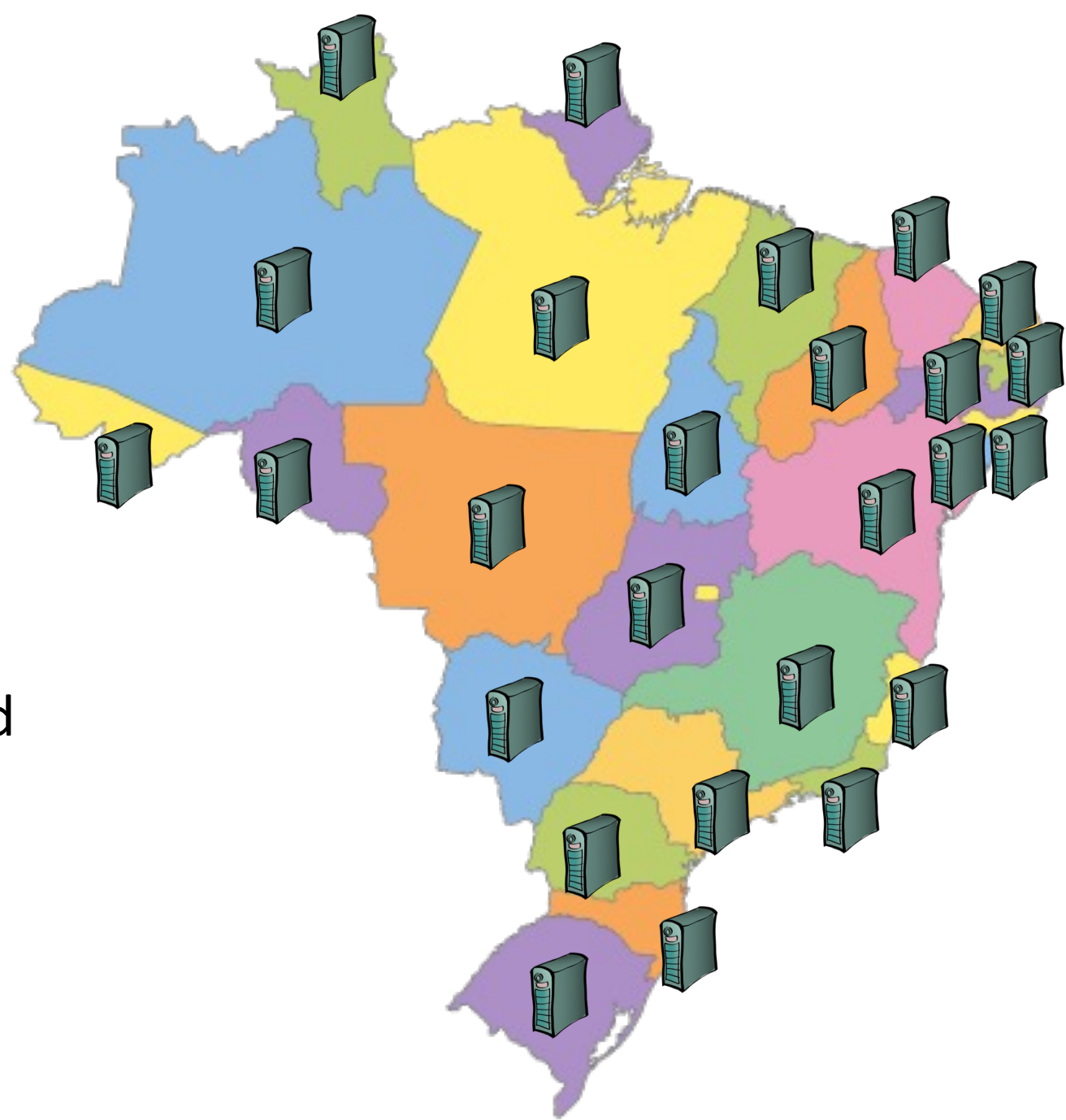
# Managing Distributed Systems

Prof. Ítalo Cunha



# Distributed driver's license management system

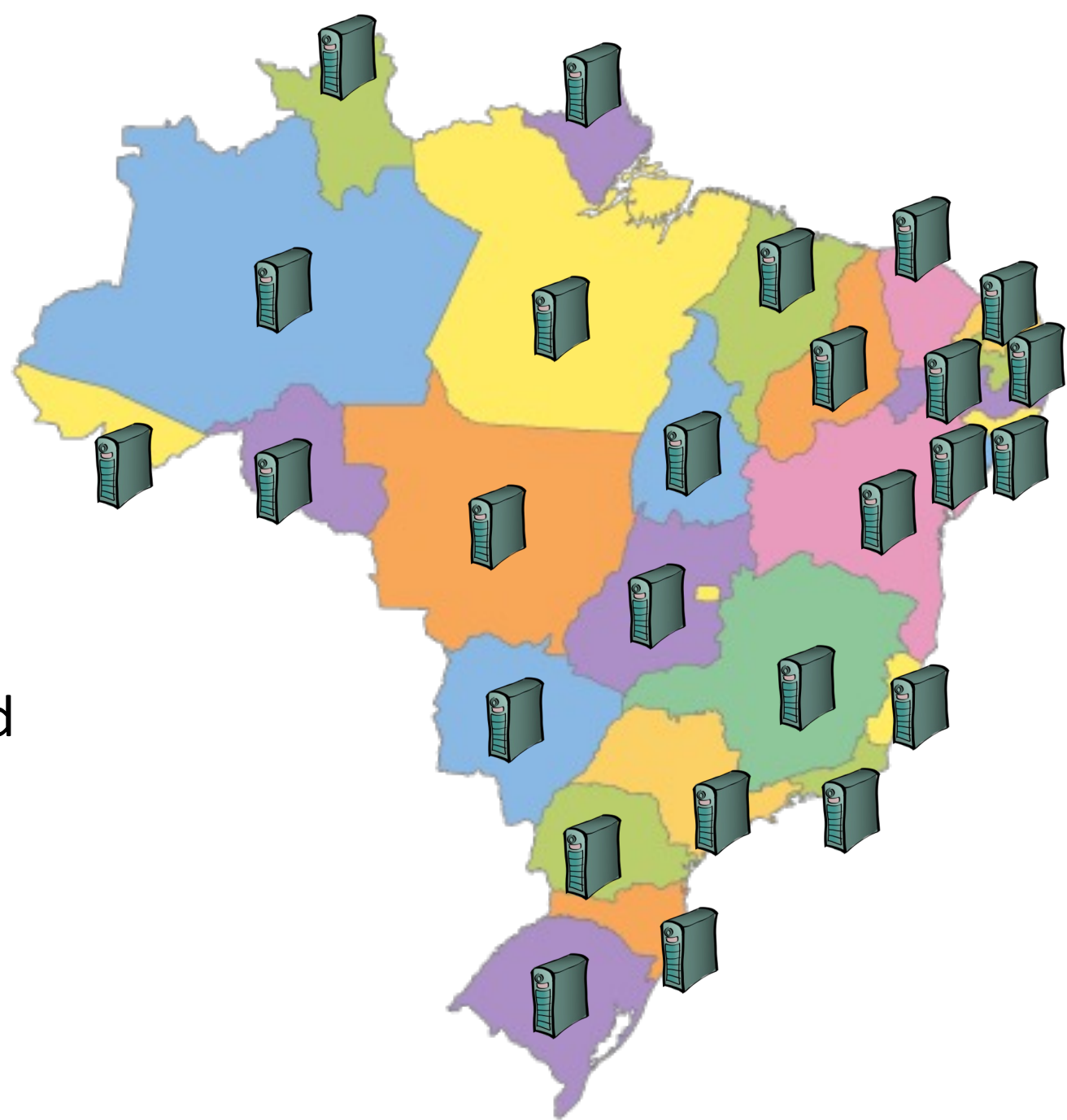
- One node per state
- Each node keeps information for drivers in that state
- Systems interoperate when cross-state information is needed





# Distributed driver's license management system

- One node per state
- Each node keeps information for drivers in that state
- Systems interoperate when cross-state information is needed
- Efficient design if most accesses are local
  - For example, lower latency



# How to keep the system running?

- Local IT staff can troubleshoot
  - Reboot servers, restart nodes
  - Repair network links



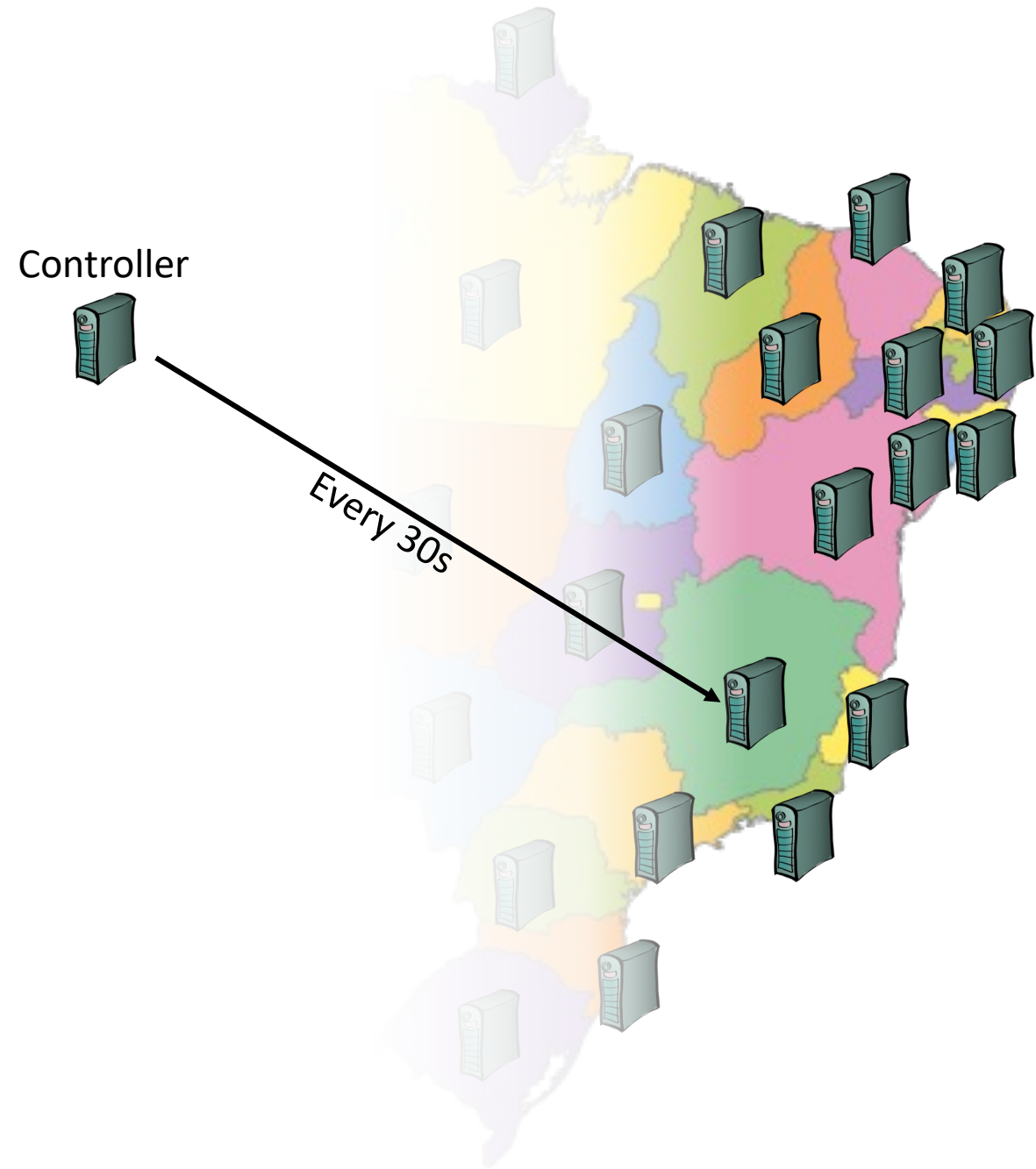
# Monitoring a distributed system

- Periodically probe nodes
  - What frequency?
    - Liveliness vs overhead
- How to probe a node
  - What to check?
    - Thoroughness vs complexity



# Monitoring a distributed system

- Periodically probe nodes
  - What frequency?
    - Liveliness vs overhead
- How to probe a node
  - What to check?
    - Thoroughness vs complexity



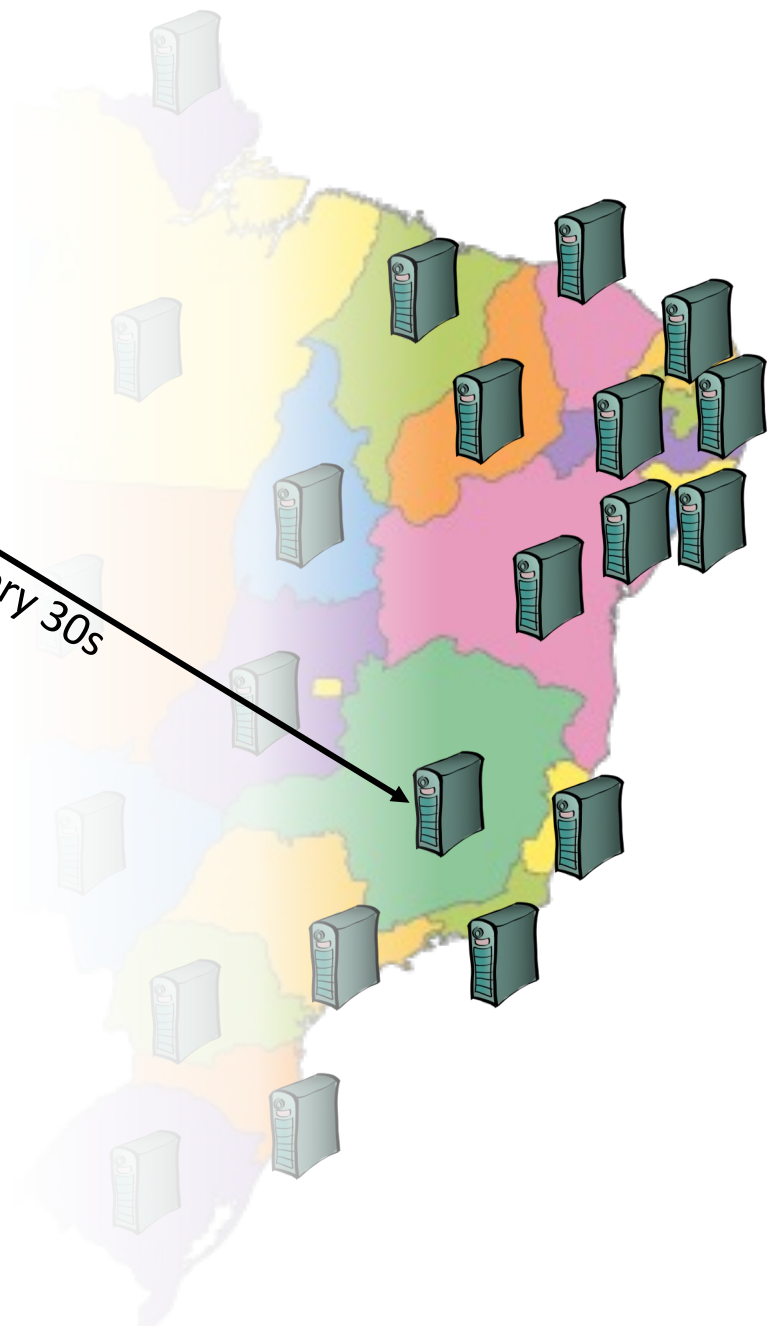
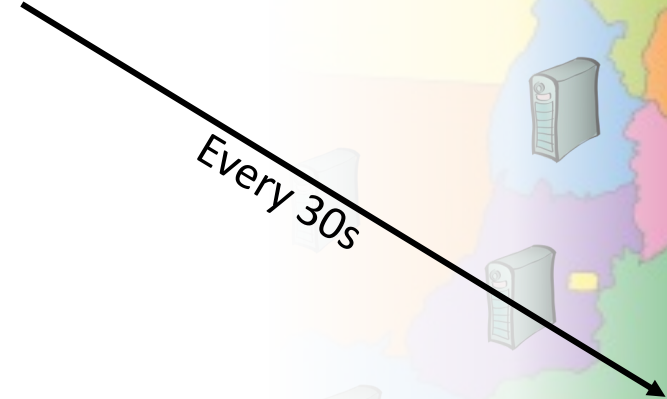
# Monitoring a distributed system

- Periodically probe nodes
  - What frequency?
    - Liveliness vs overhead
- How to probe a node
  - What to check?
    - Thoroughness vs complexity
  - Ping → Check that the network is working
  - Simple request → Check network and app
  - Complex request → Network, app, and database

Controller



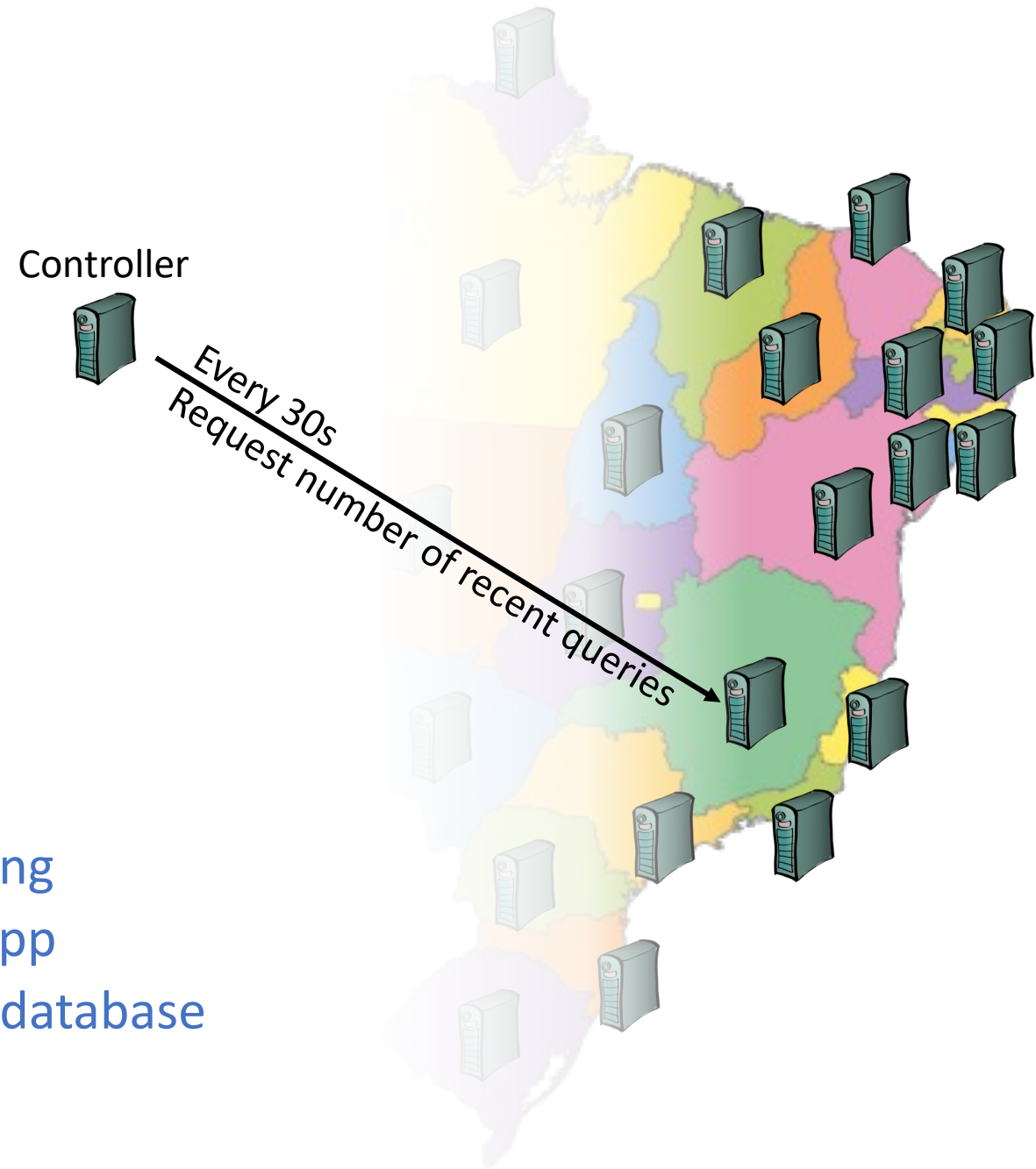
Every 30s





# Monitoring a distributed system

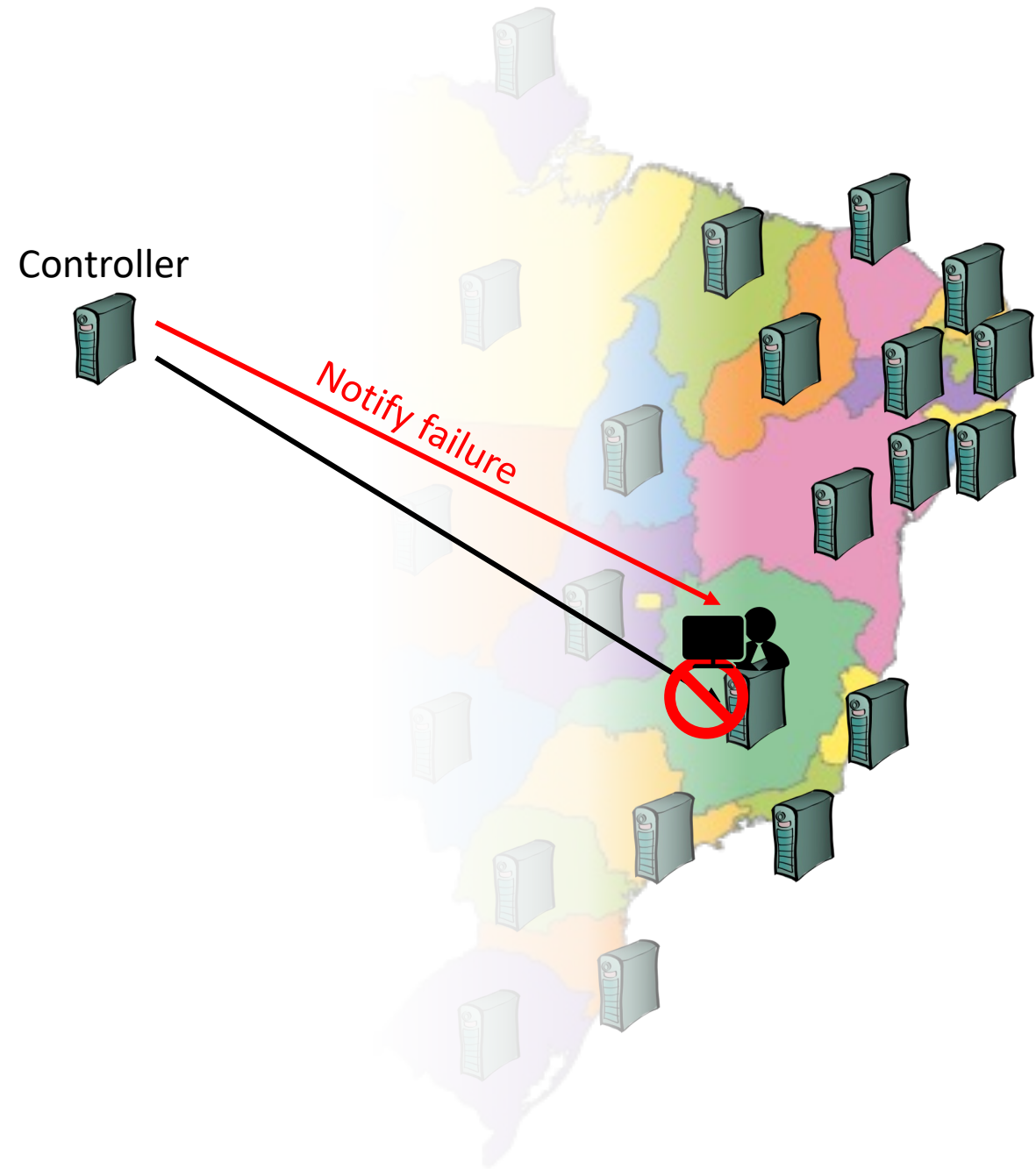
- Periodically probe nodes
  - What frequency?
    - Liveliness vs overhead
- How to probe a node
  - What to check?
    - Thoroughness vs complexity
  - Ping → Check that the network is working
  - Simple request → Check network and app
  - Complex request → Network, app, and database





# Monitoring a distributed system

- Periodically probe nodes
  - What frequency?
    - Liveliness vs overhead
- How to probe a node
  - What to check?
    - Thoroughness vs complexity
- Notify staff if failure detected





CompSci 401: Cloud Computing

# Managing Cloud Applications

Prof. Ítalo Cunha



# Differences between distributed systems and cloud-native applications

## **Distributed System**

- Monolithic applications on each node
- Multiple instances of one application

## **Cloud Native**

- Containers/pods in each microservice
- Multiple instances of multiple microservices

Controller needs to handle a significantly larger number of instances.



# Differences between distributed systems and cloud-native applications

## **Distributed System**

- Monolithic applications on each node
- Multiple instances of one application
- Static set of instances
- Persistent applications

## **Cloud Native**

- Containers/pods in each microservice
- Multiple instances of multiple microservices
- Instances may move (placement)
- Ephemeral containers

Cannot know which instances to monitor in advance.

# Differences between distributed systems and cloud-native applications

## Distributed System

- Monolithic applications on each node
- Multiple instances of one application
- Static set of instances
- Persistent applications
- Fixed number of instances

## Cloud Native

- Containers/pods in each microservice
- Multiple instances of multiple microservices
- Instances may move (placement)
- Ephemeral containers
- Autoscaling creates/destroys instances

New instances may appear, and instances may disappear. Disappearing instances *is OK!*



# CompSci 401: Cloud Computing

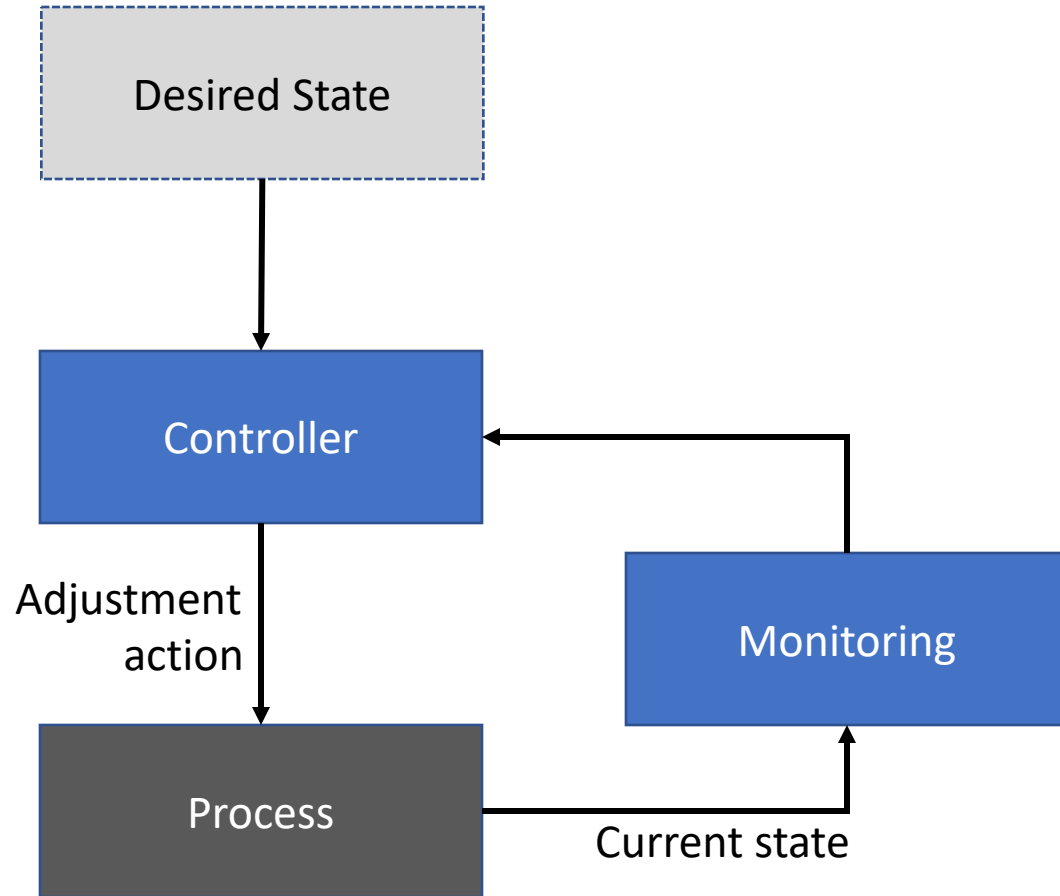
# Control Loops

Prof. Ítalo Cunha

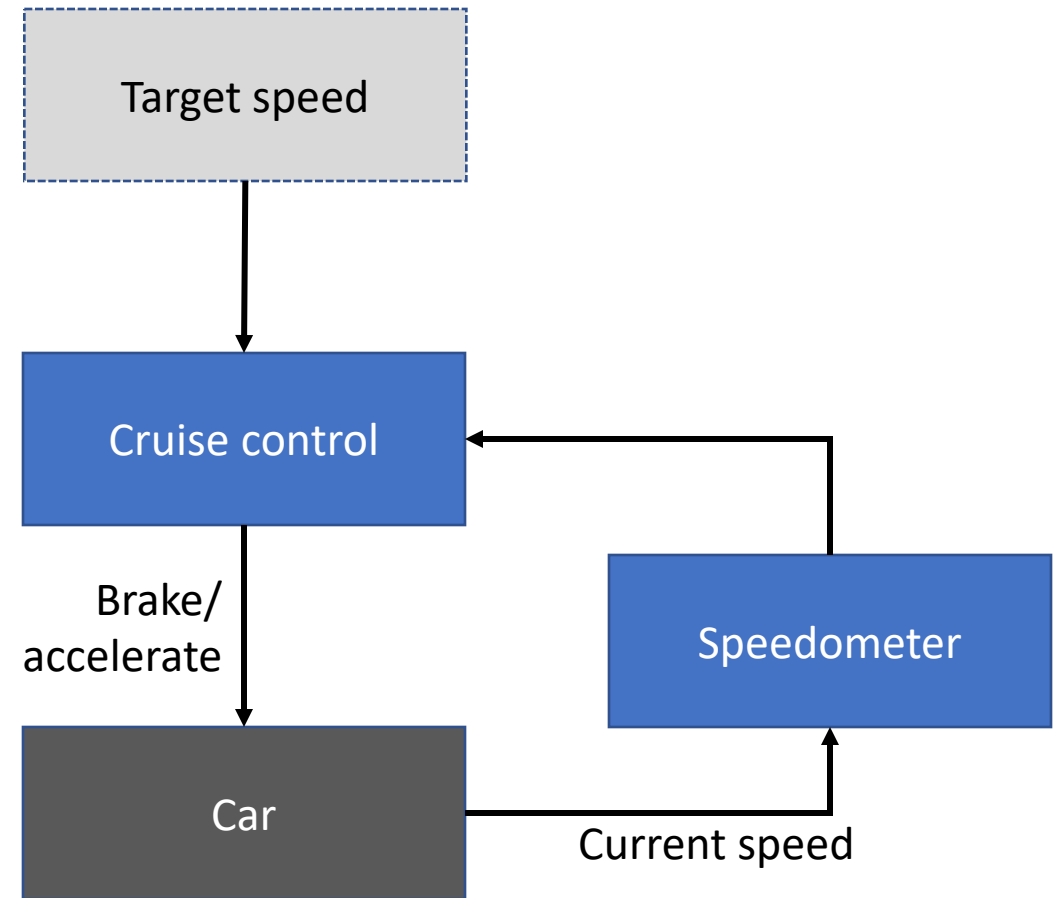
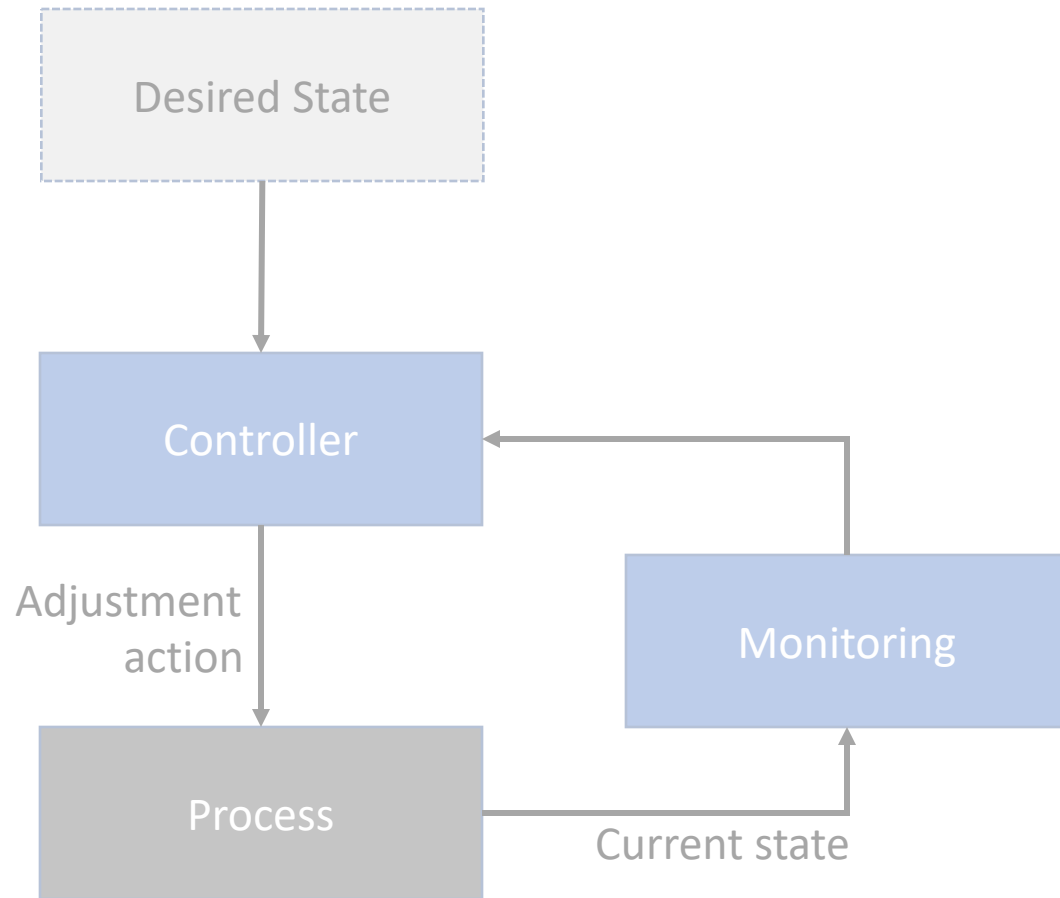




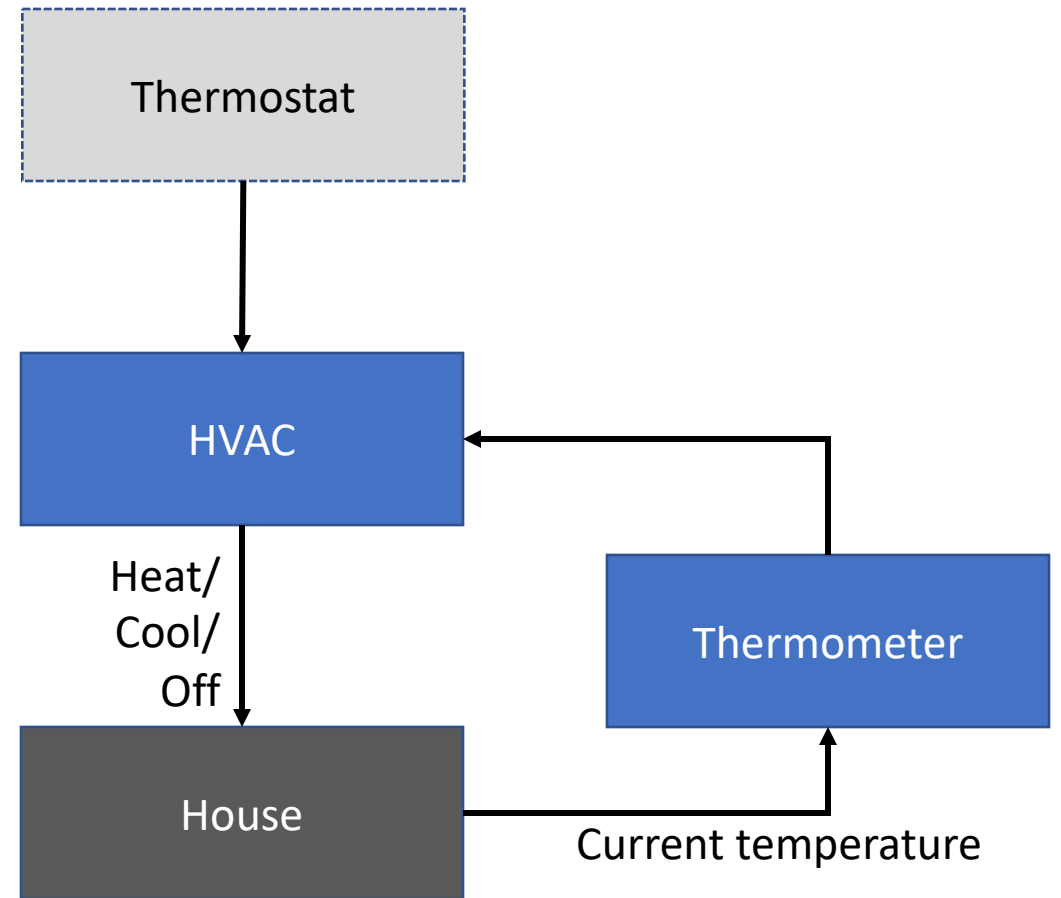
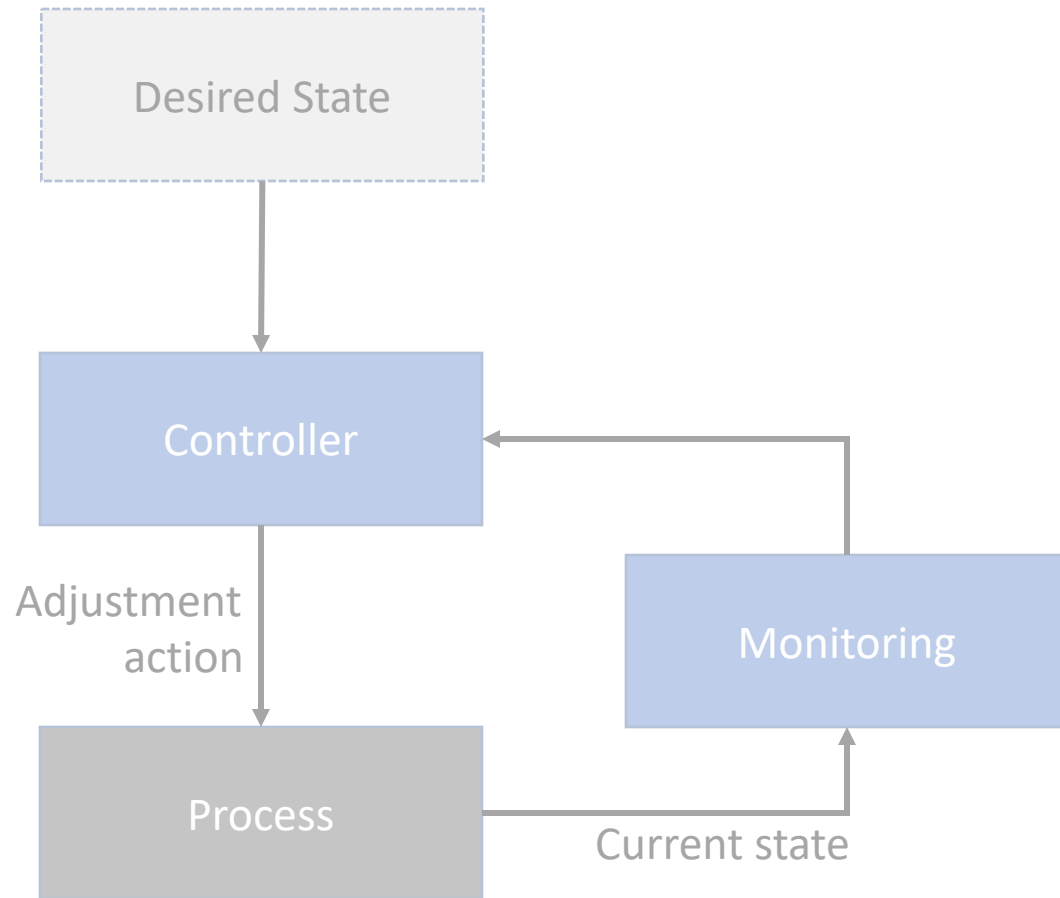
# Control loop



# Control loop

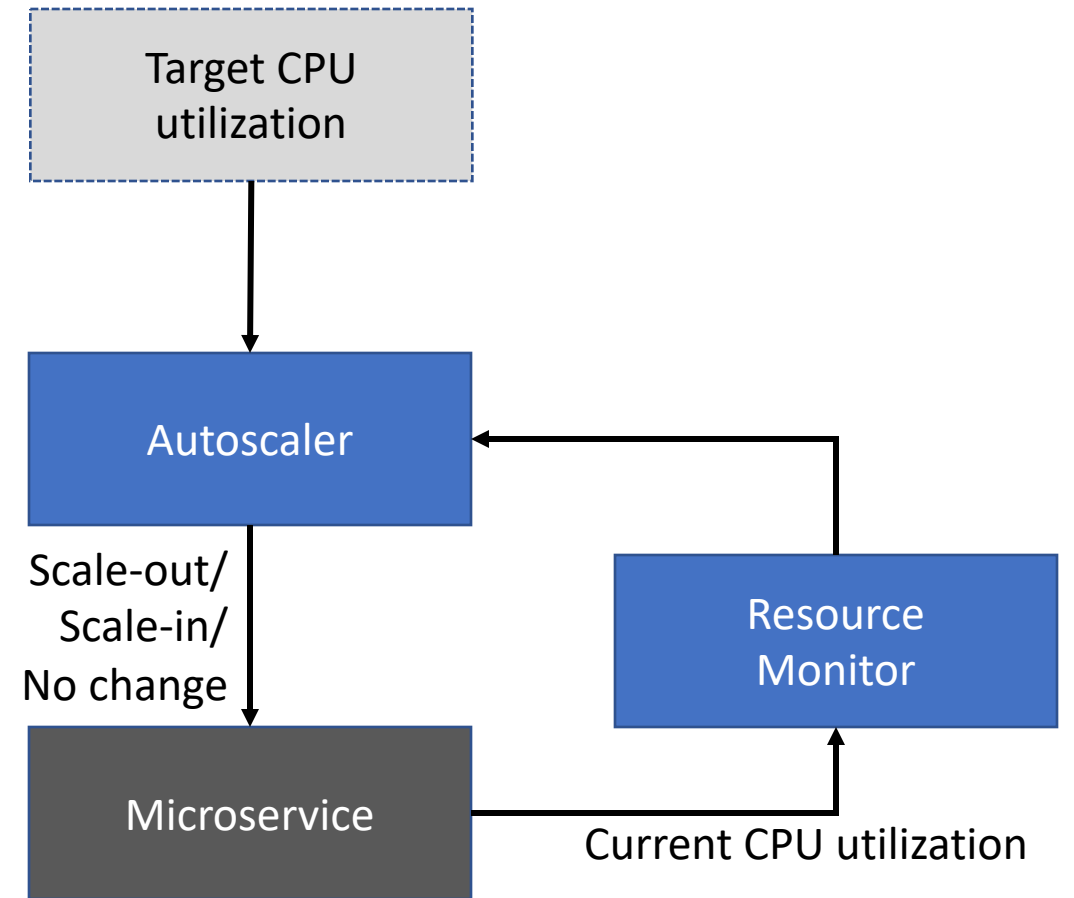
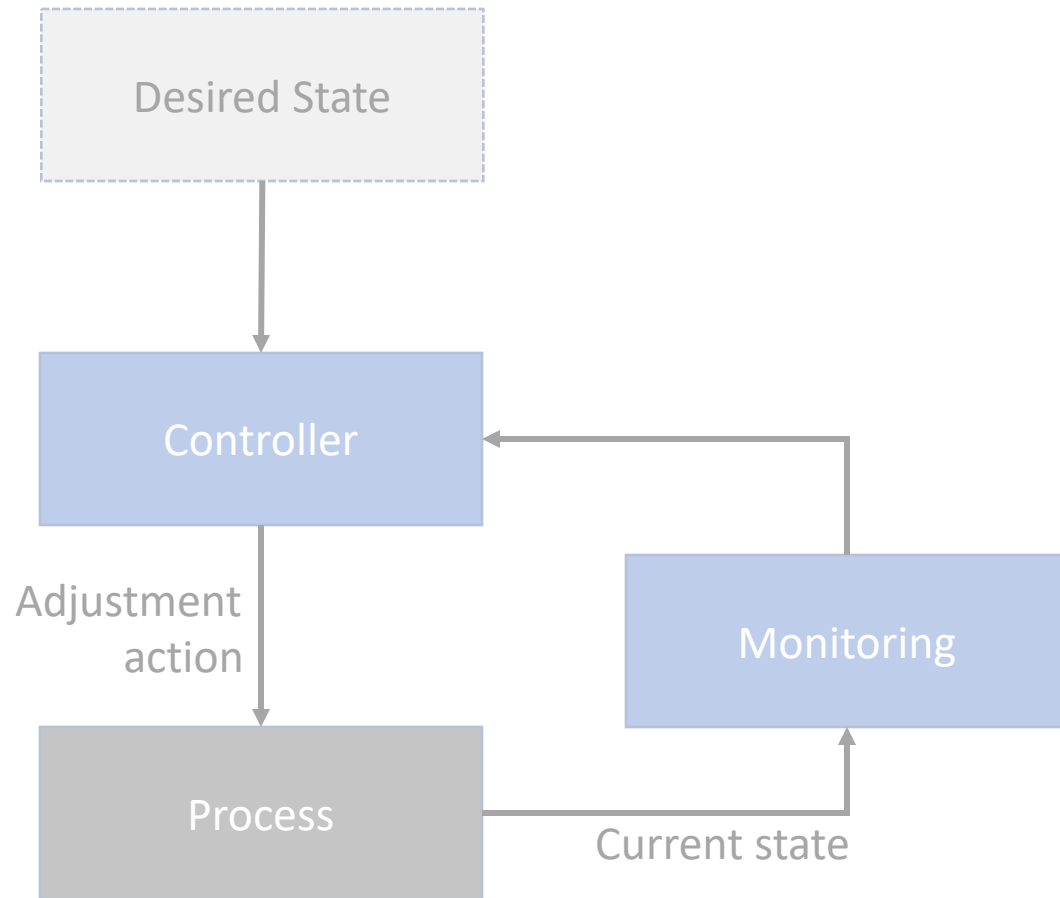


# Control loop





# Control loop

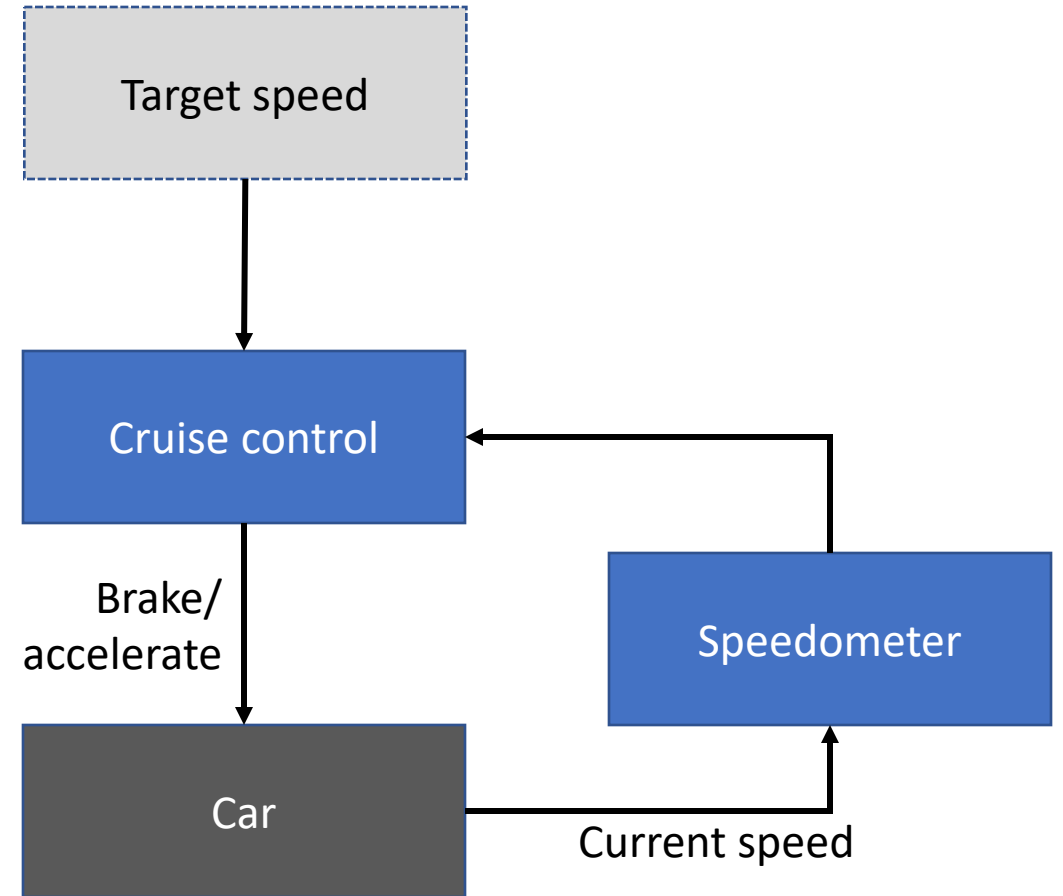


# Declarative interface

- We say what is the desired state
- *Not how to achieve it*

# Declarative interface

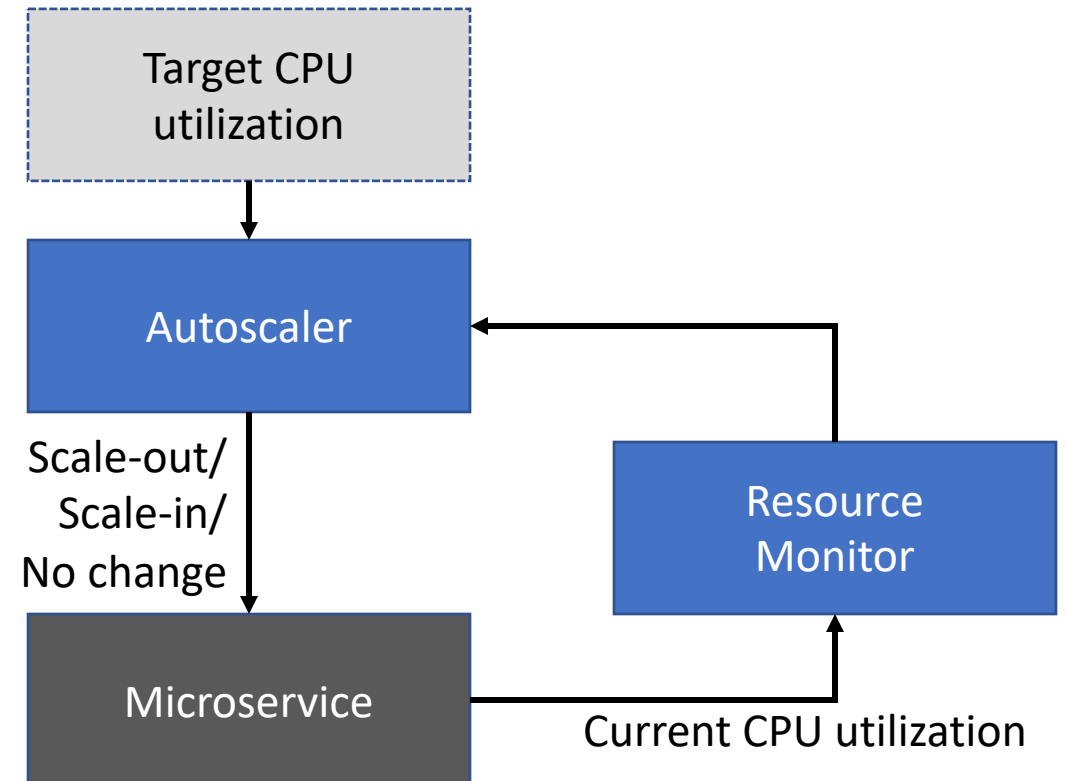
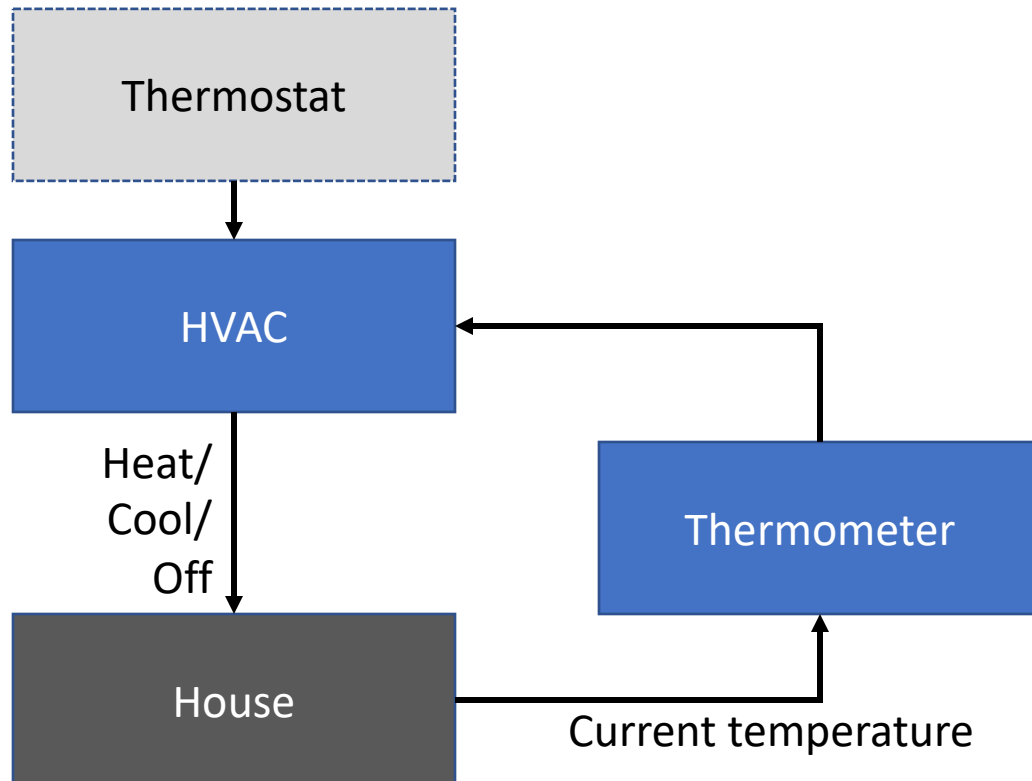
- We say what is the desired state
- *Not how to achieve it*





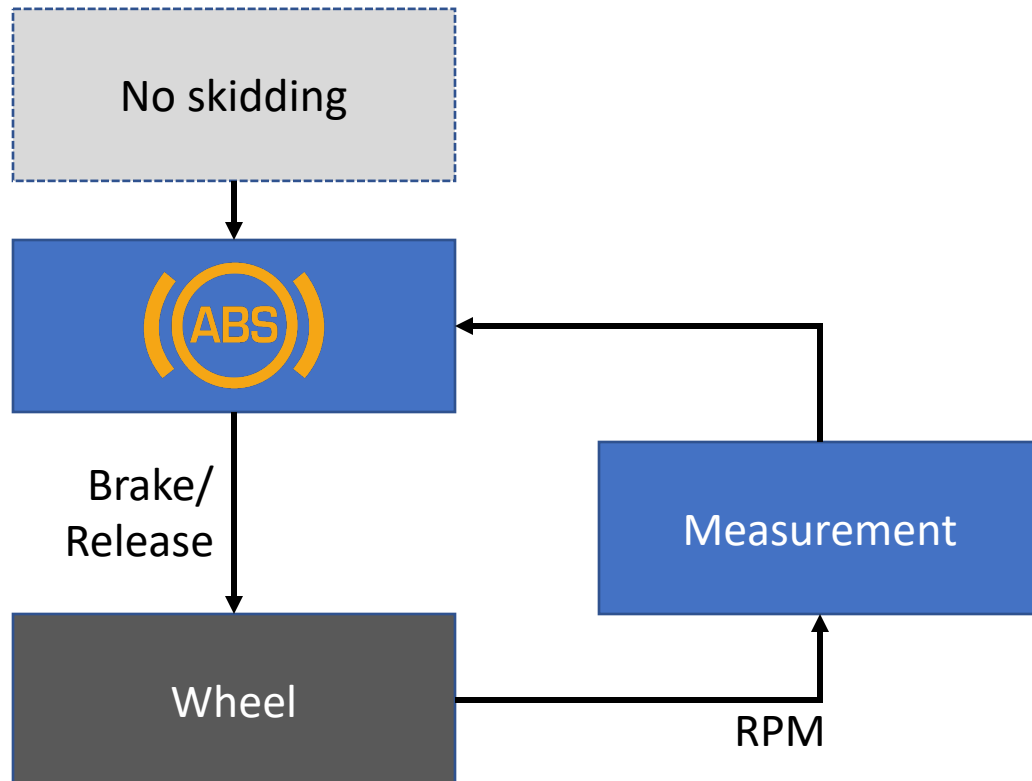
# Control loop delay

- How frequently to measure state and perform control actions?
  - Application dependent



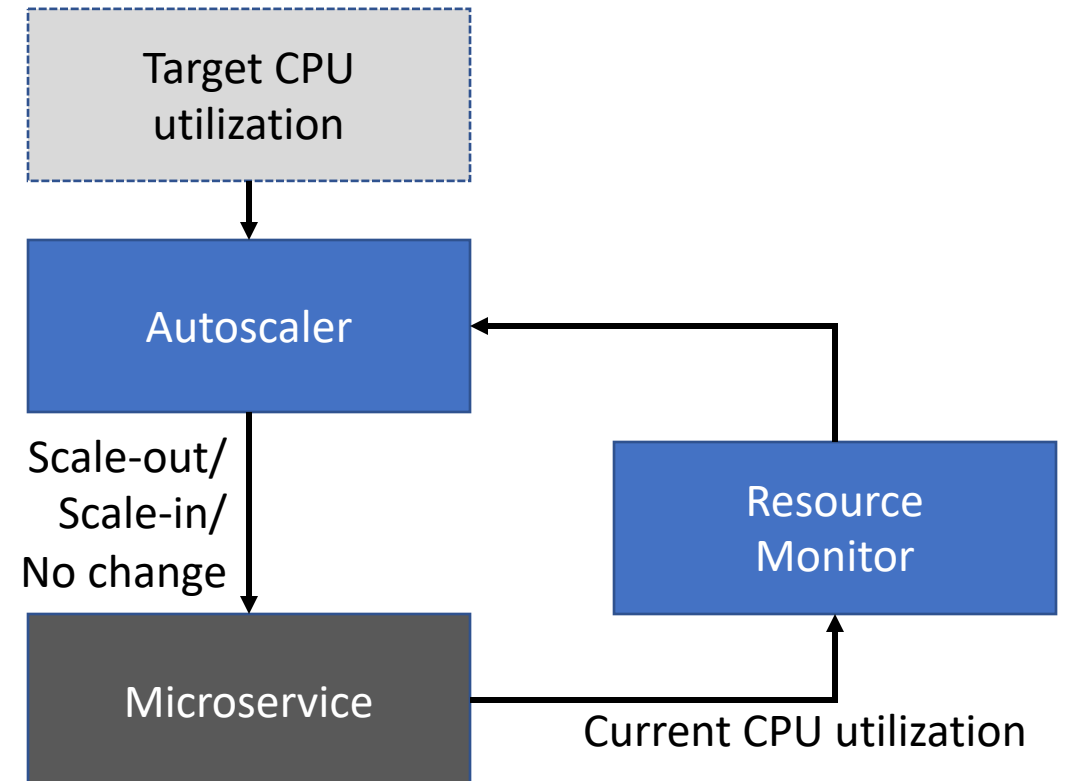
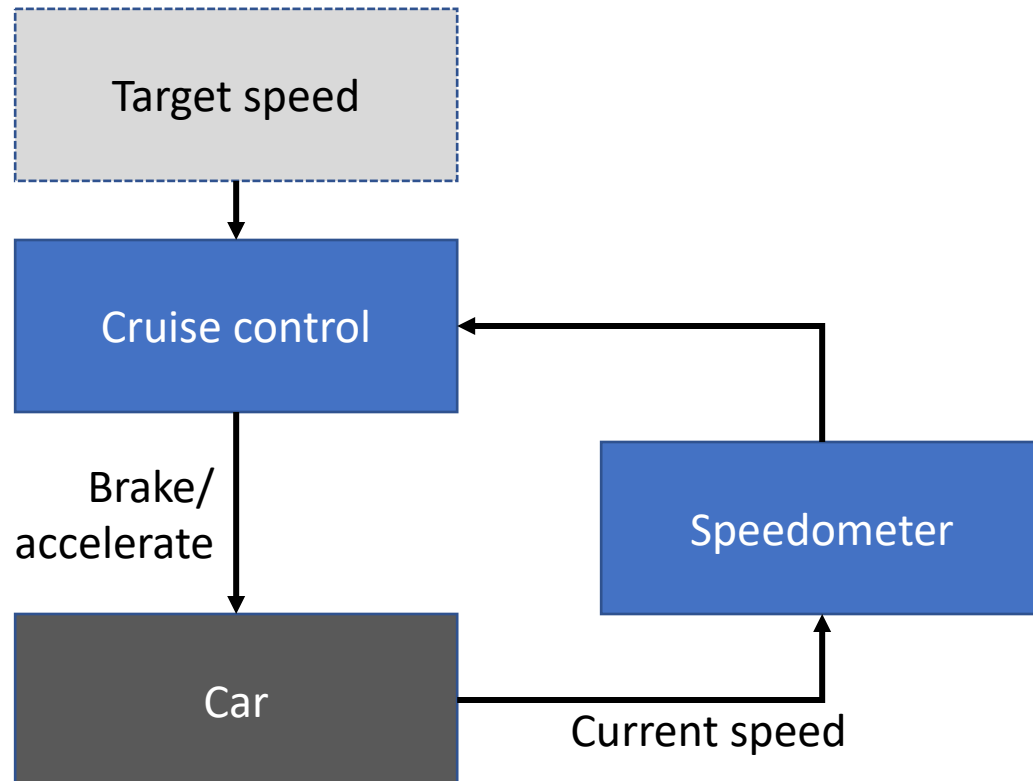
# Control loop delay

- How frequently to measure state and perform control actions?
  - Application dependent

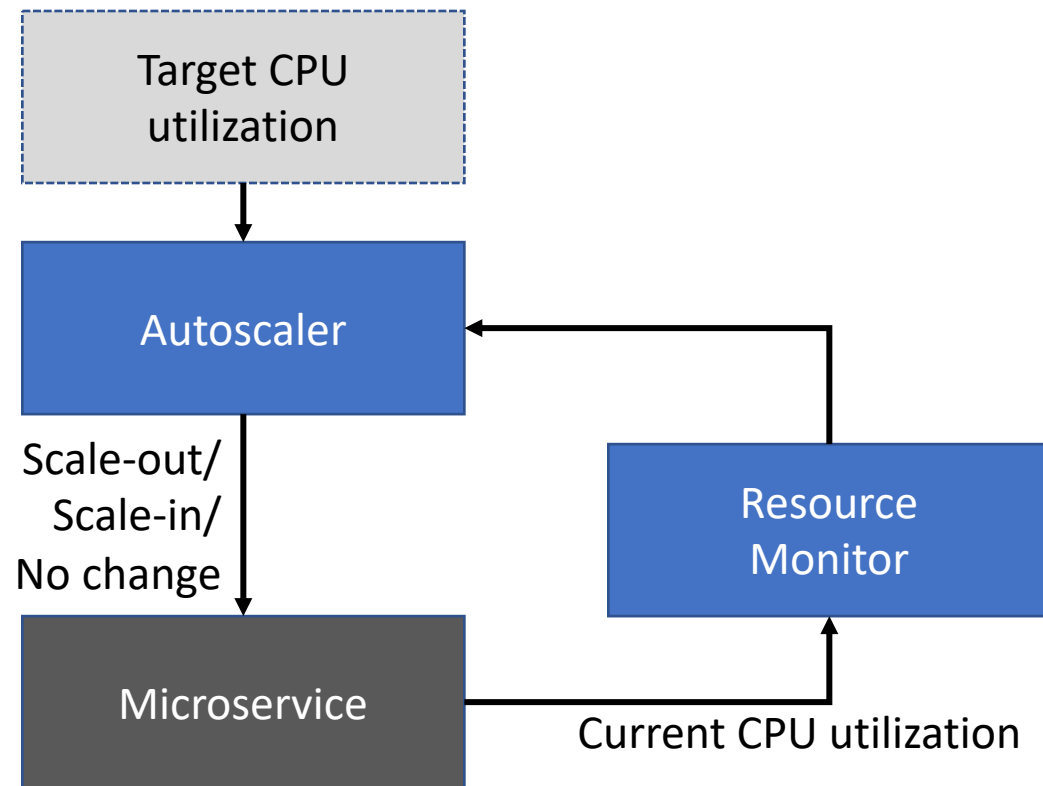


# Hysteresis

- State may take time to change, control system needs to consider



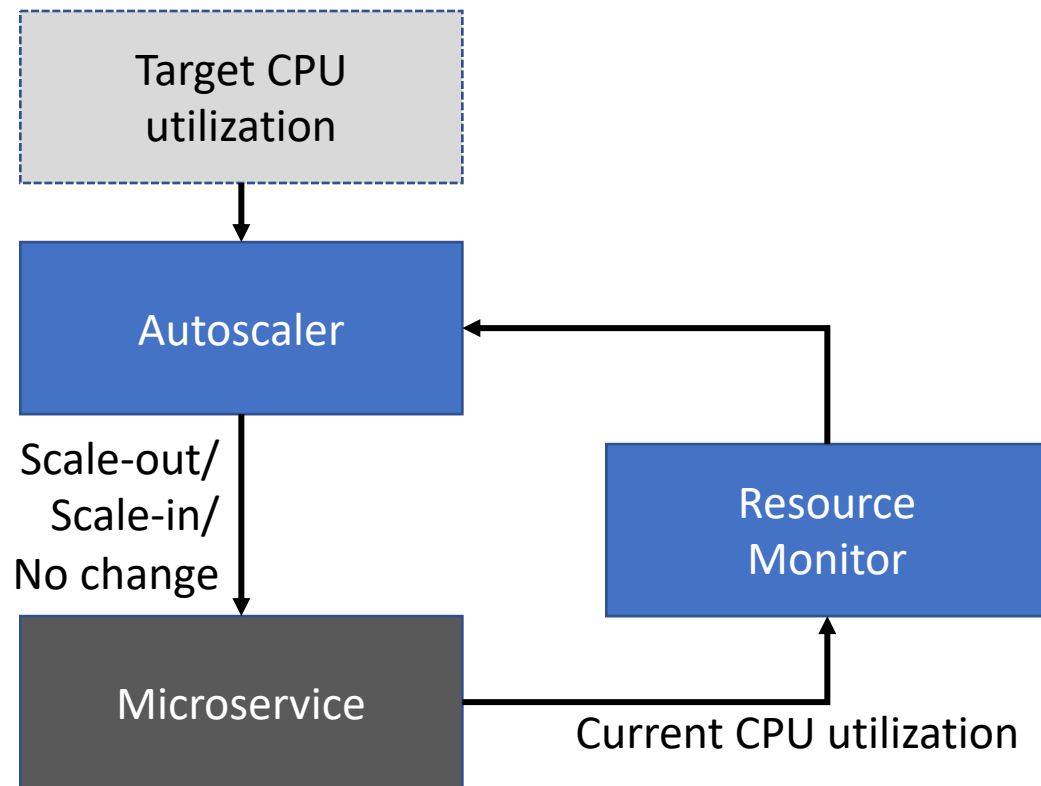
# Hysteresis may lead to undesirable conditions



## Sequence of events

- T1: Resource monitor tells autoscaler that the microservice is overloaded
- T2: Scale-out 2 instances
- T3: New instances start

# Hysteresis may lead to undesirable conditions

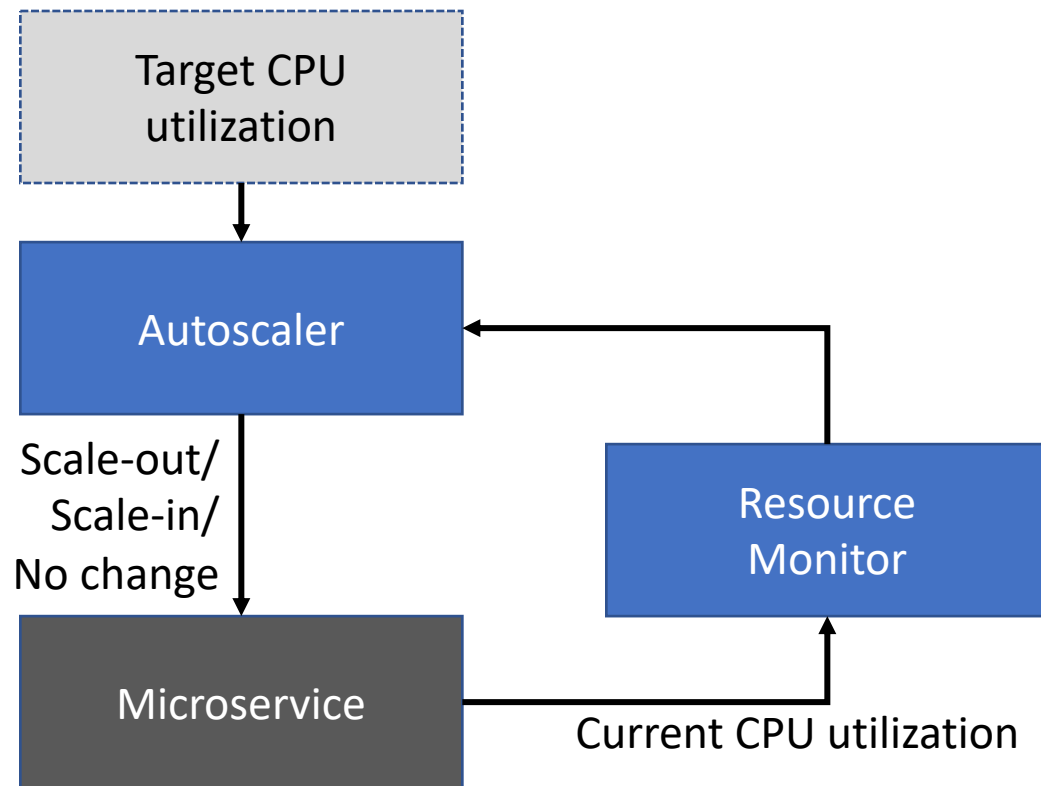


## Sequence of events

- T1: Resource monitor tells autoscaler that the microservice is overloaded
- T2: Scale-out 2 instances
- T3: New instances start
- T4: Load balancer starts to distribute load
- T5: Resource monitor tells autoscaler that the microservice is overloaded
- T6: Scale-out 2 instances



# Hysteresis may lead to undesirable conditions



## Sequence of events

- T1: Resource monitor tells autoscaler that the microservice is overloaded
- T2: Scale-out 2 instances
- T3: New instances start
- T4: Load balancer starts to distribute load
- T5: Resource monitor tells autoscaler that the microservice is overloaded
- T6: Scale-out 2 instances

**+4 instances may be unnecessary!**

# Hysteresis may lead to instability



# Hysteresis may lead to instability



# Hysteresis may lead to instability





# Hysteresis may lead to instability



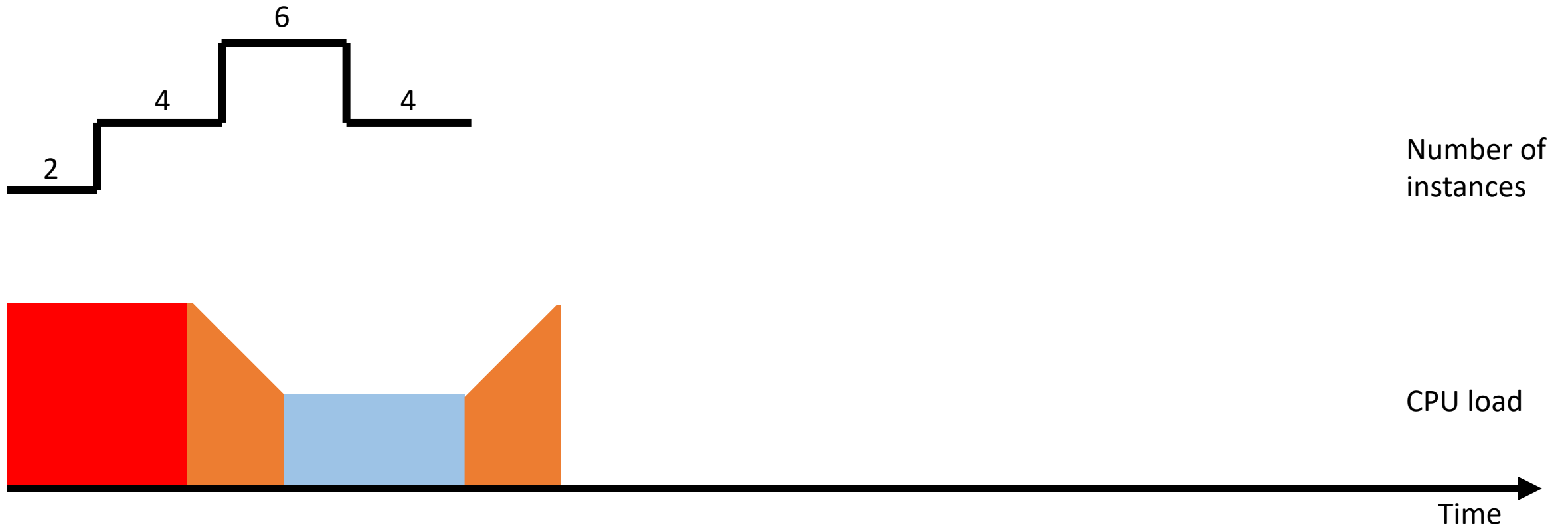
# Hysteresis may lead to instability



# Hysteresis may lead to instability

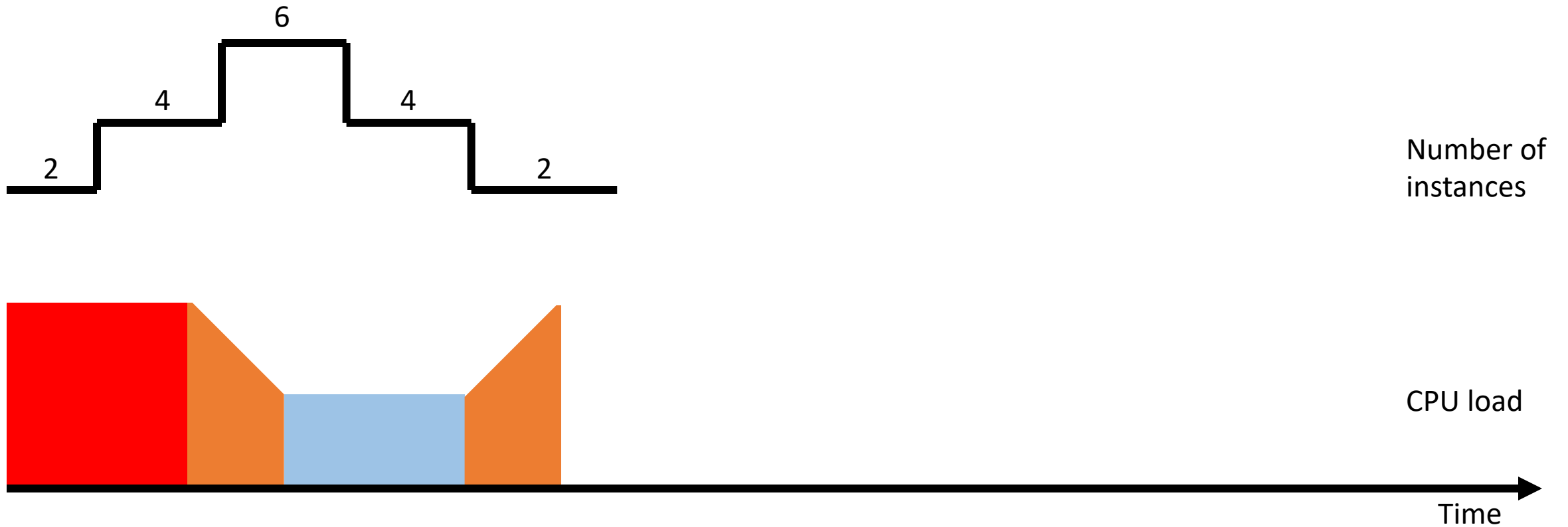


# Hysteresis may lead to instability

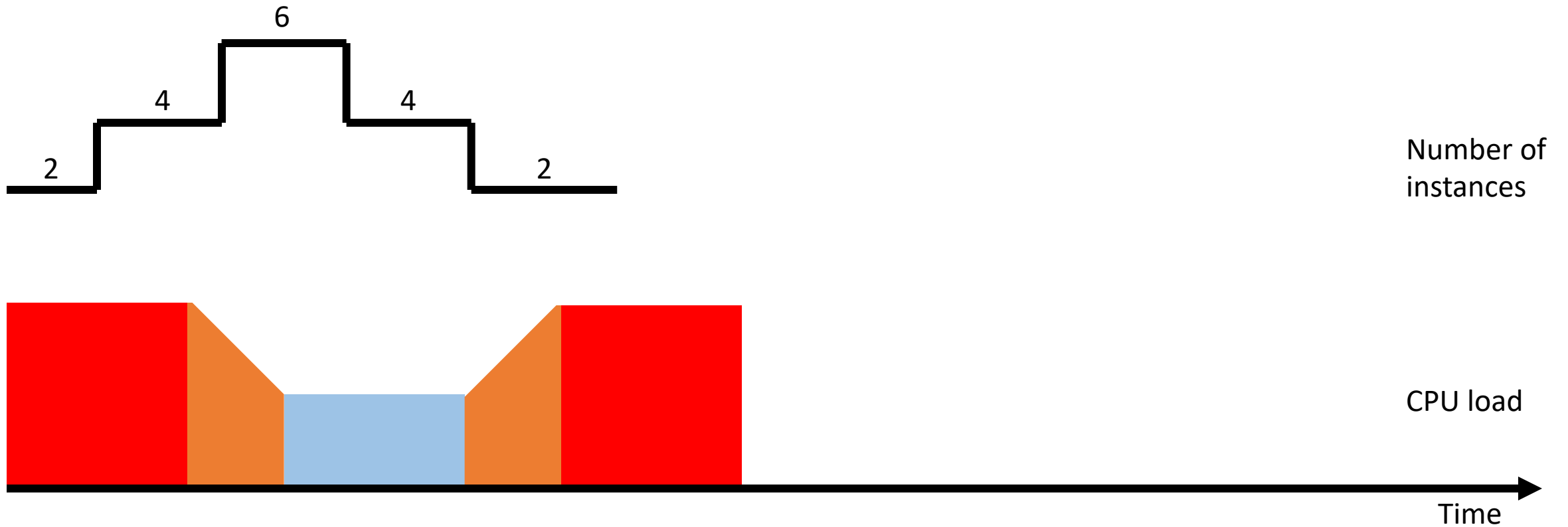




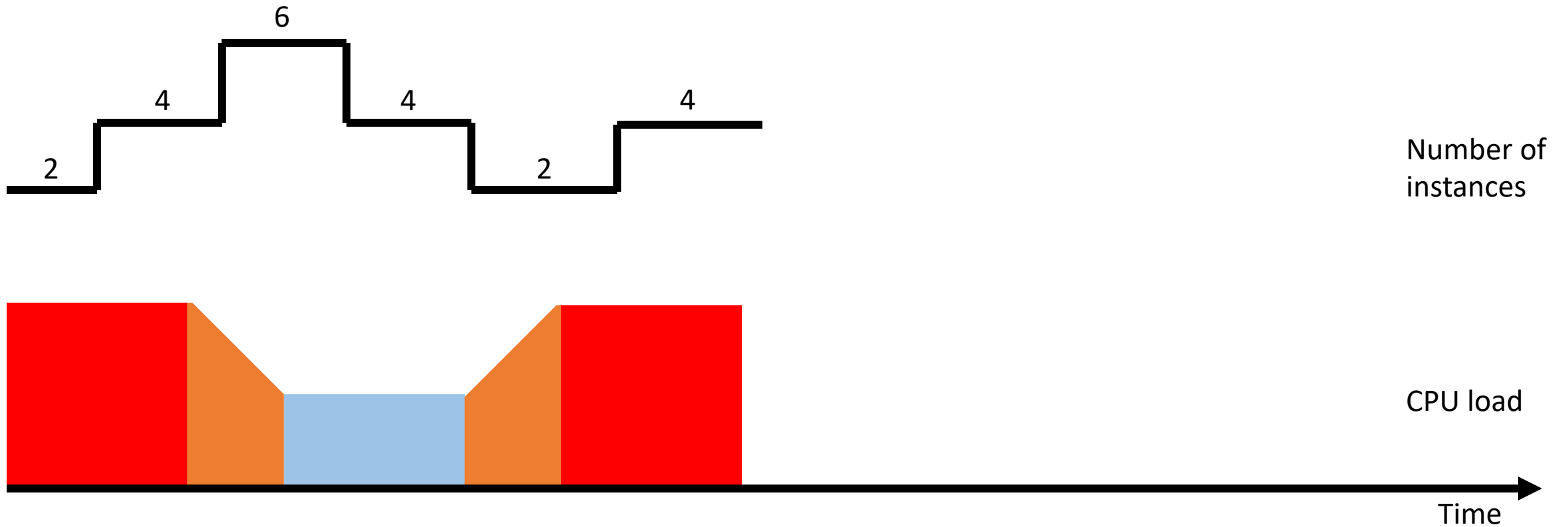
# Hysteresis may lead to instability



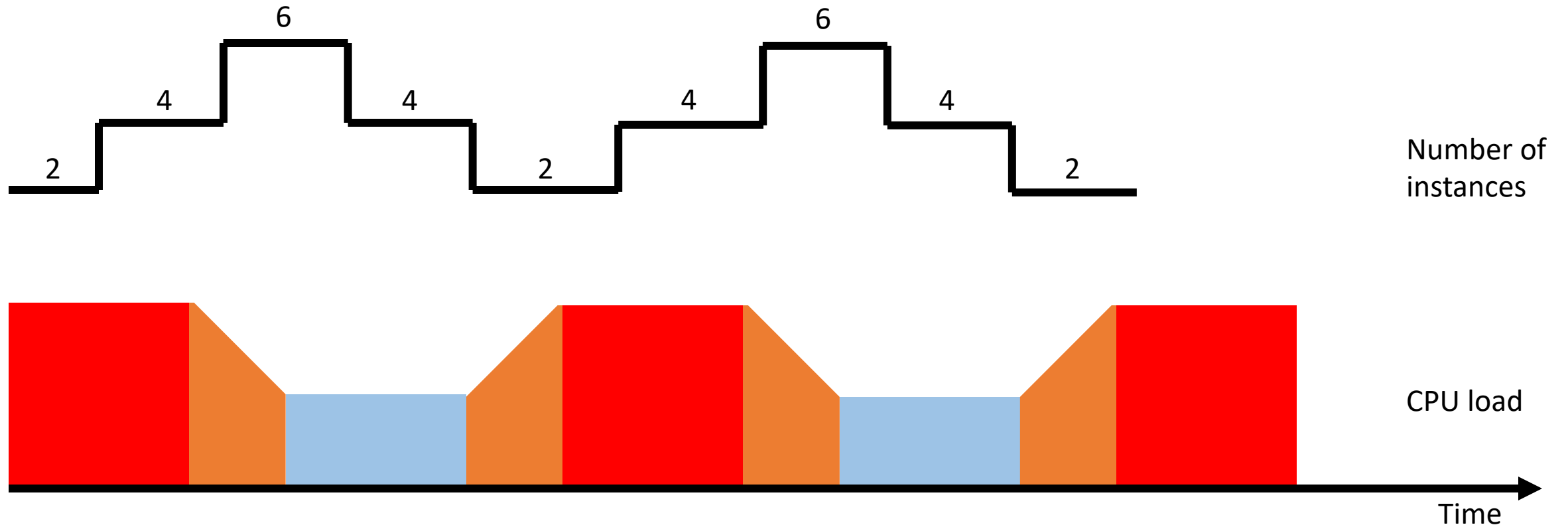
# Hysteresis may lead to instability



# Hysteresis may lead to instability



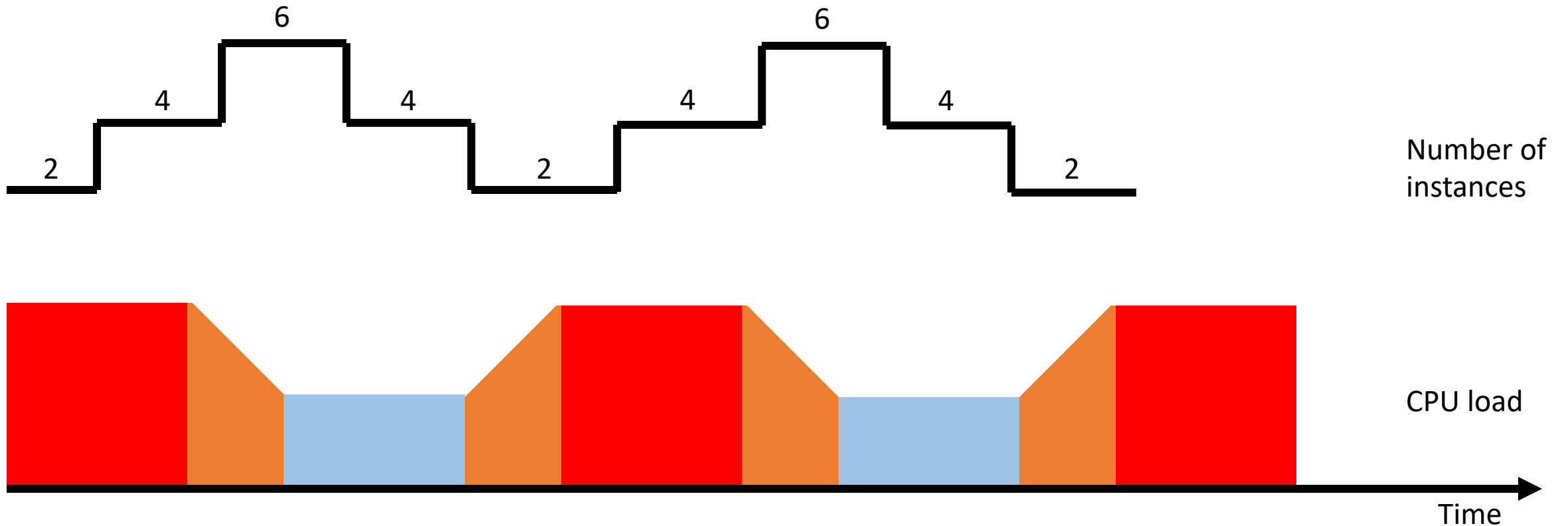
# Hysteresis may lead to instability





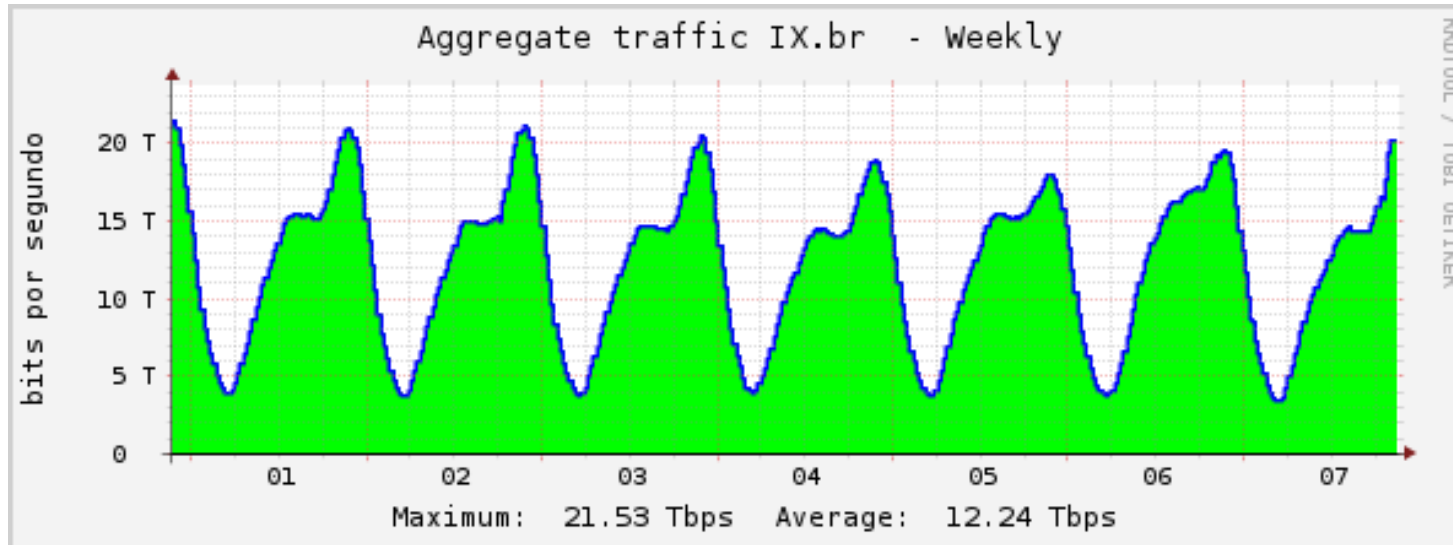
# Instability vs dynamics

- This example consider that the workload is constant



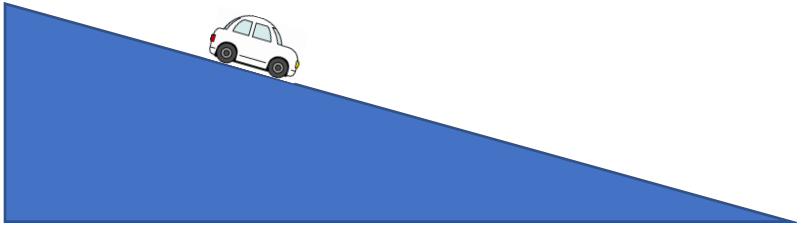
# Instability vs dynamics

- If the workload is changing, a system may never stabilize



Bits per second at an Internet exchange point in Brazil

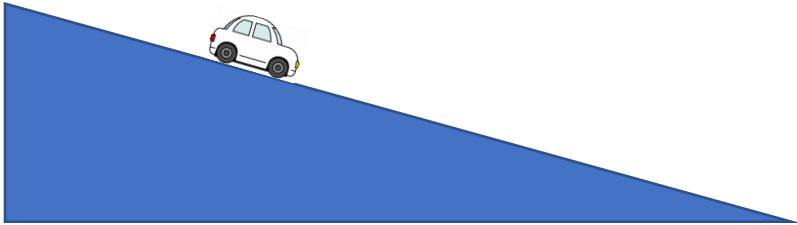
# Action intensity



Cruise control at 60km/h

Breaking super hard, releasing the brake,  
then breaking super hard again...  
Would be very uncomfortable and  
possibly make people sick!

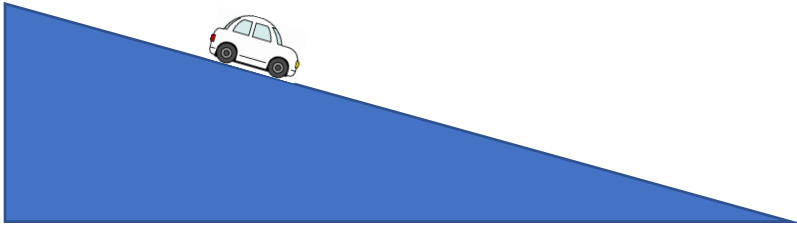
# Response intensity



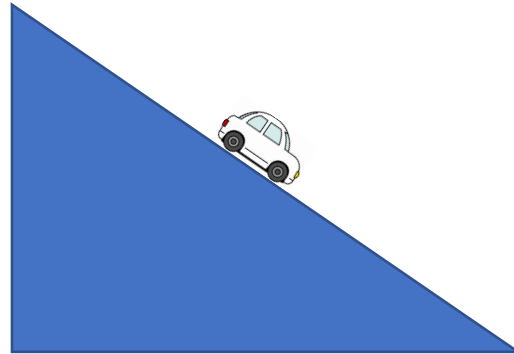
Cruise control at 60km/h

Brake slightly

# Response intensity

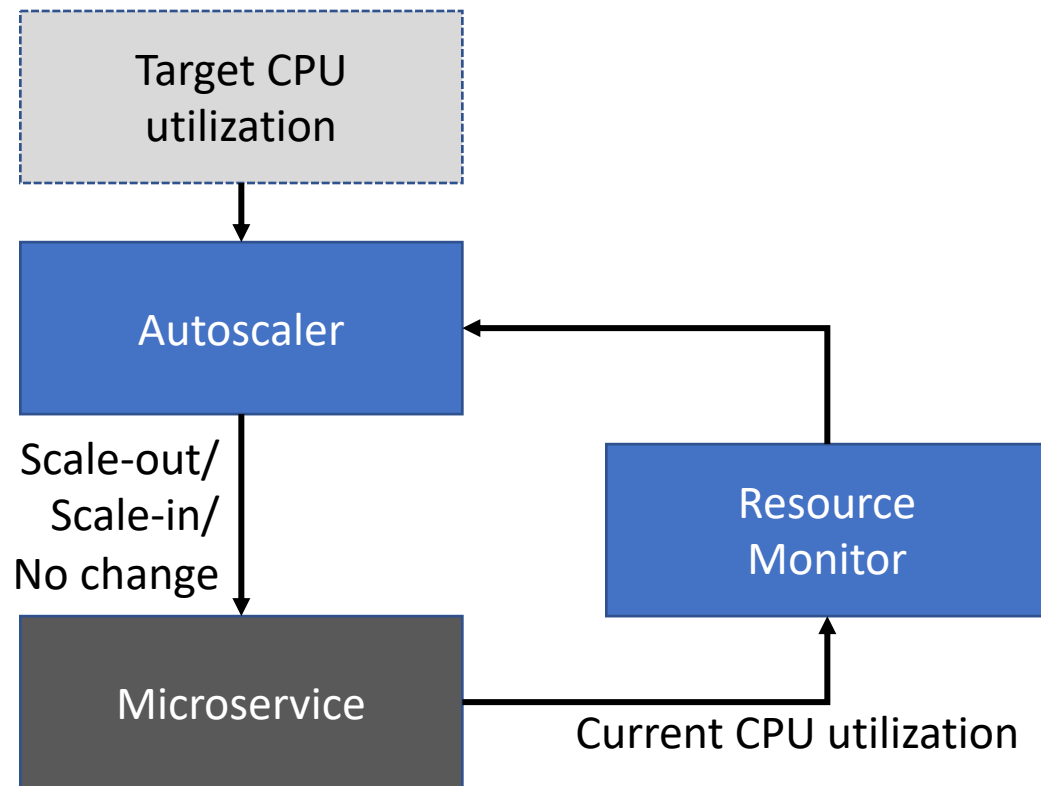


Cruise control at 60km/h  
Brake slightly



Cruise control at 60km/h  
Brake harder

# Why scale-out by 2 instances?!



## Sequence of events

- T1: Resource monitor tells autoscaler that the microservice is overloaded
- T2: Scale-out 2 instances
- T3: New instances start
- T4: Load balancer starts to distribute load
- T5: Resource monitor tells autoscaler that the microservice is overloaded
- T6: Scale-out 2 instances





CompSci 401: Cloud Computing

# Kubernetes Controllers

Prof. Ítalo Cunha

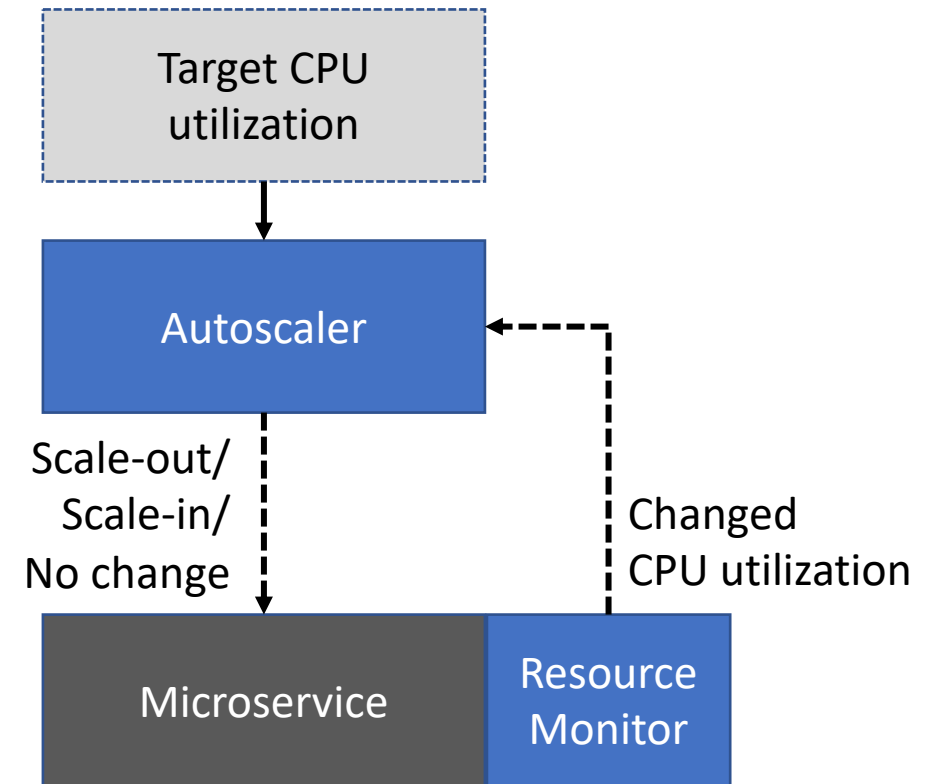


# Kubernetes controllers

- Run a control loop indefinitely
- Compare actual and desired state of the system
- Make adjustments to achieve the desired state

# Event-based control

- Monitors report events to controller
- No need to poll resources periodically for their state
- Only act on state changes



# Event-based control

- Monitors report events to controller
- No need to poll resources periodically for their state
- Only act on state changes
- **Reduced overhead and quicker responses**

# Example Kubernetes controllers

- Node controller
  - Responsible for noticing and responding when nodes fail

# Example Kubernetes controllers

- Node controller
  - Responsible for noticing and responding when nodes fail

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

# Example Kubernetes controllers

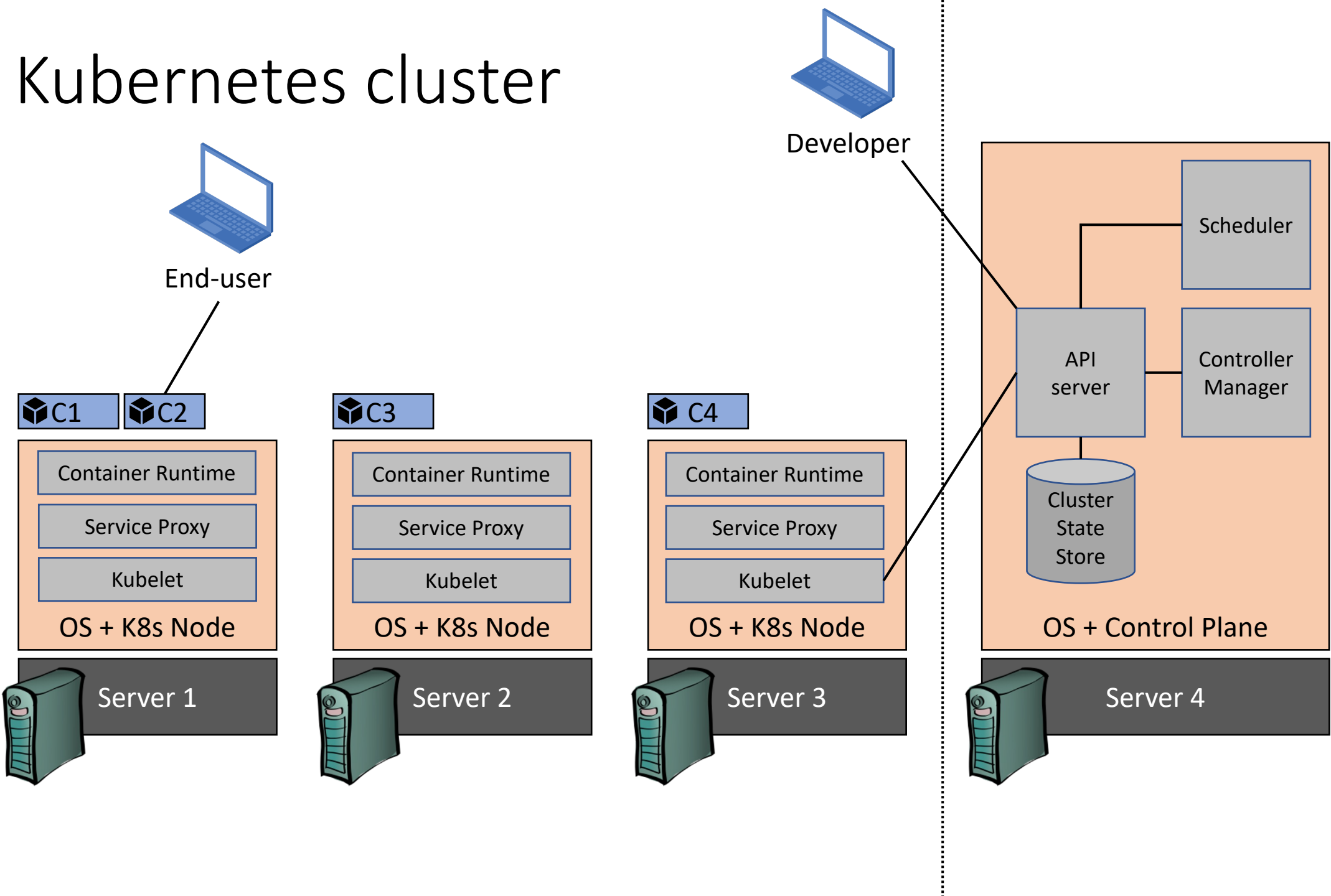
- Node controller
  - Responsible for noticing and responding when nodes go down
- Job controller
  - Watches for Job objects that represent one-off tasks
  - Then creates Pods to run those tasks to completion
- Endpoints controller
  - Naming and discovery to integrate services and jobs
- Service account & token controllers
  - Create default accounts and API access tokens



# Example Kubernetes controllers

- Node controller
  - Responsible for noticing and responding when nodes go down
- Job controller
  - Watches for Job objects that represent one-off tasks
  - Then creates Pods to run those tasks to completion.
- Endpoints controller
  - Naming and discovery to integrate services and jobs
- Service account & token controllers
  - Create default accounts and API access tokens
- ... we can define or own **Resources** and **Controllers**

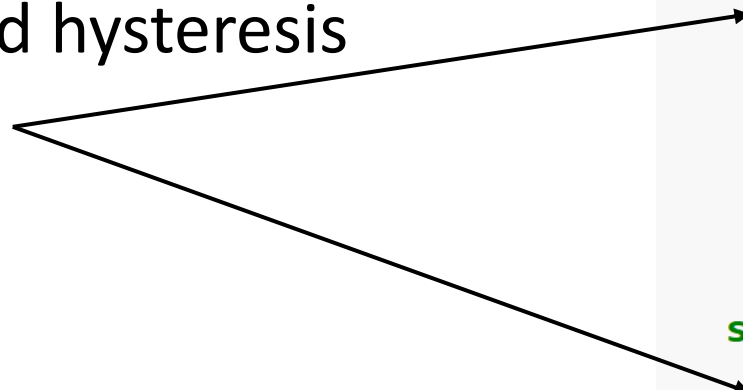
# Kubernetes cluster



# Autoscaling controller

- Control loop delay and hysteresis
  - Stabilization window
  - Period

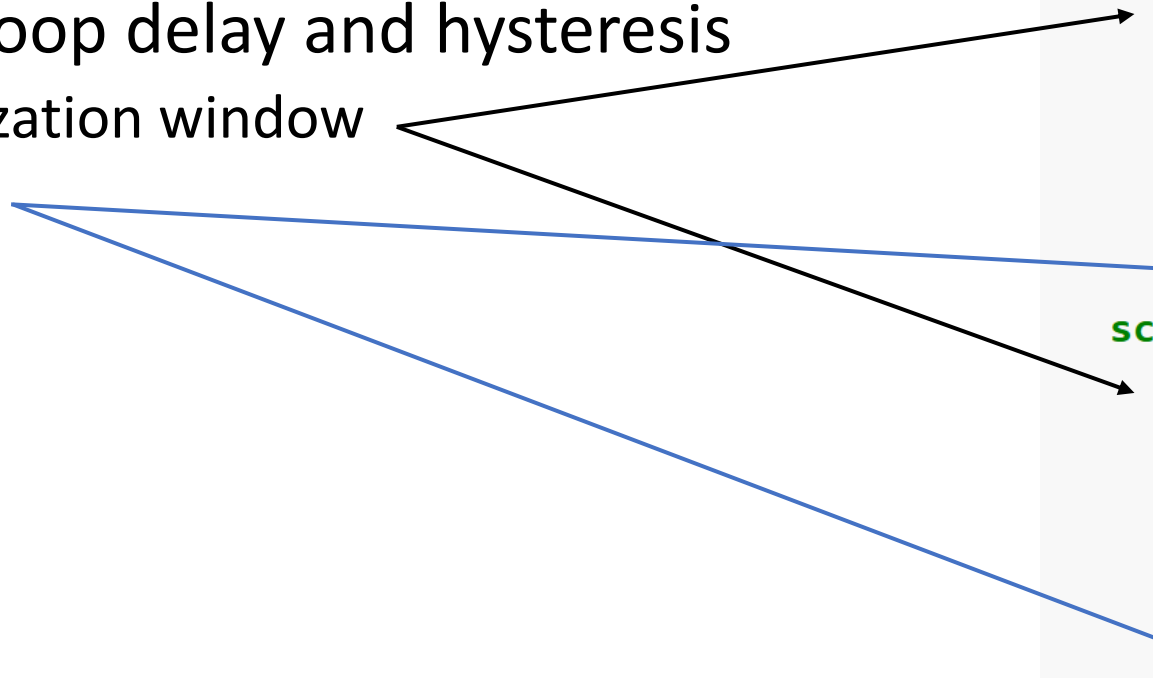
```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
    policies:
      - type: Percent
        value: 100
        periodSeconds: 15
  scaleUp:
    stabilizationWindowSeconds: 0
    policies:
      - type: Percent
        value: 100
        periodSeconds: 15
      - type: Pods
        value: 4
        periodSeconds: 15
  selectPolicy: Max
```

A diagram consisting of two arrows originates from the 'Stabilization window' and 'Period' items in the list on the left. One arrow points from 'Stabilization window' to the 'stabilizationWindowSeconds: 300' field under the 'scaleDown' section of the configuration. The other arrow points from 'Period' to the 'periodSeconds: 15' field under the 'policies' section of the 'scaleDown' section.

# Autoscaling controller

- Control loop delay and hysteresis
  - Stabilization window
  - Period

```
behavior:  
  scaleDown:  
    stabilizationWindowSeconds: 300  
    policies:  
      - type: Percent  
        value: 100  
    periodSeconds: 15  
  scaleUp:  
    stabilizationWindowSeconds: 0  
    policies:  
      - type: Percent  
        value: 100  
      - type: Pods  
        value: 4  
        periodSeconds: 15  
    selectPolicy: Max
```



# Autoscaling controller

- Control loop delay and hysteresis
  - Stabilization window
  - Period
- Action intensity

$$\text{Instances}_{\text{new}} = \left\lceil \text{Instances}_{\text{old}} \left( \frac{\text{currentState}}{\text{desiredState}} \right) \right\rceil$$

```
behavior:
  scaleDown:
    stabilizationWindowSeconds: 300
    policies:
      - type: Percent
        value: 100
        periodSeconds: 15
  scaleUp:
    stabilizationWindowSeconds: 0
    policies:
      - type: Percent
        value: 100
        periodSeconds: 15
      - type: Pods
        value: 4
        periodSeconds: 15
    selectPolicy: Max
```

