# Control Theory In Container Orchestration

Vallery Lancey
Lead DevOps Engineer, Checkfront

# Container Orchestration Fundamentals

@vllry

# Goals of Container Management

- Reproducibility.
- Cohabitation.
- *Auto-management of instances.*

@vllry

# System Management

**Traditional:** a sysadmin examines the system, makes a judgement, and performs an action.

**Automatic:** the system tracks its own state, and translates the state to some internal action.

@vllry

# Key Auto-Management Features

- Allocate appropriate resources.
- Manage network based on container health & state.
- Reap unhealthy containers.
- Maintain container headcount.
- Auto-scale container groups.

# Control Theory

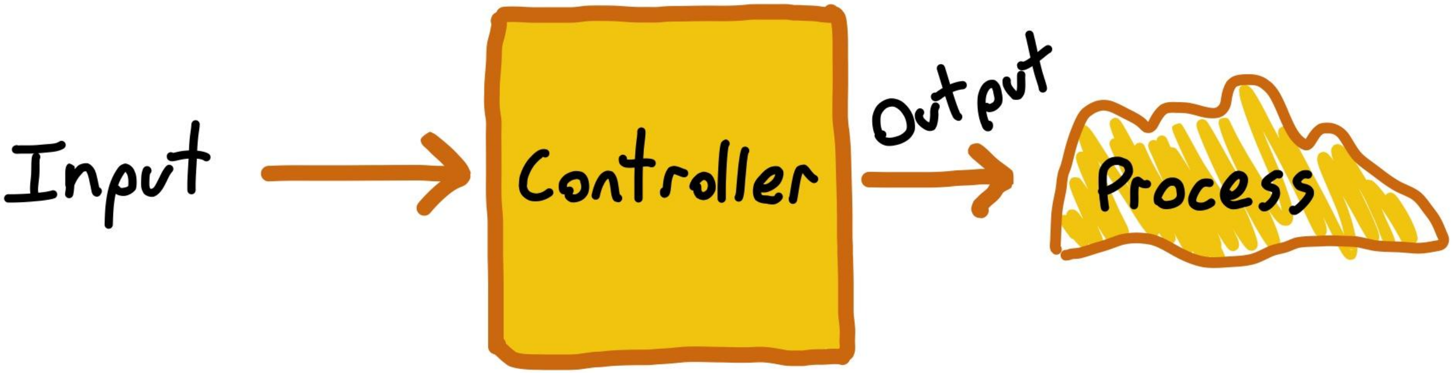

@vllry

# What is Control Theory?

- Engineering topic: how to manage a system using human and internal controls.
- Used heavily in...
  - Physical device design
  - Plant/factory management
  - Electrical engineering

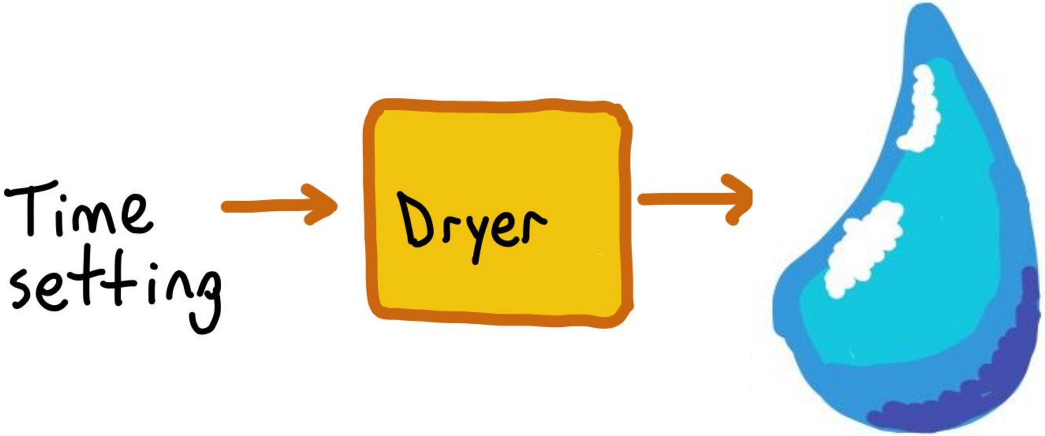
# A Controller

- Inputs dictate what the controller should do (setpoint).
- Outputs dictate what the controlled process should do.



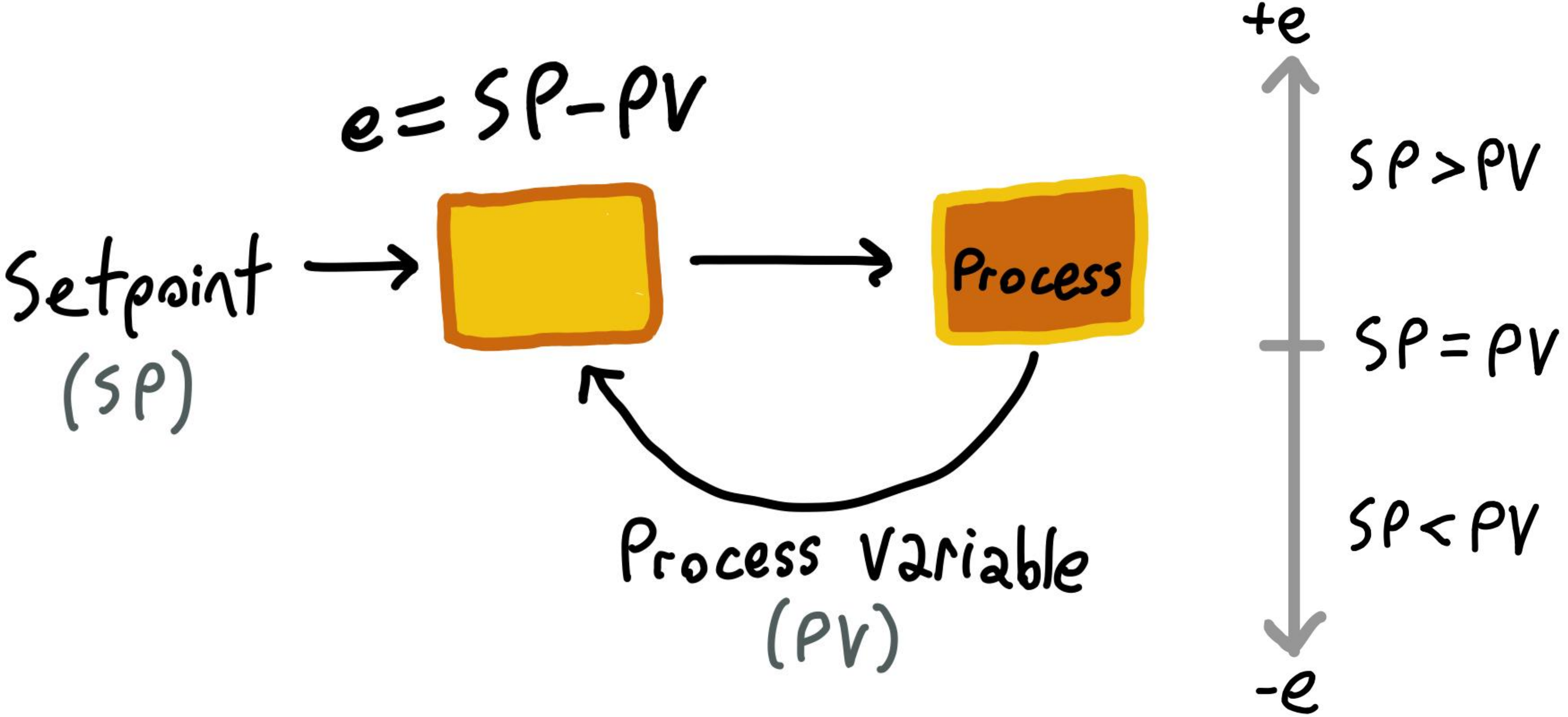
# Open Loop Controllers

- A controller with only inputs and outputs is an *open loop controller.*
- Can't respond to *feedback* from the controlled process.



# Closed Loop Controller

- Contains feedback from the process to the controller.
- The controller is able to self-correct to achieve the desired *outcome.*


e=SP-PV
Setpoint
(SP)
Process
Process Variable
(PV)
+e
SP>PV
SP=PV
SP<PV
-e

# The Math is Unfortunate

- Control theory is split into *linear* (PV changes linearly with control) and *nonlinear* problems.
- Most of our problems are nonlinear.
- Nonlinear problems have fewer known methods, and are often reduced to simplified linear problems.

# Applying Control Theory To Containers

@vllry

```
while True {
    currentState = getCurrentState()
    desiredState = getDesiredState()
    makeConform(currentState, desiredState)
}
```

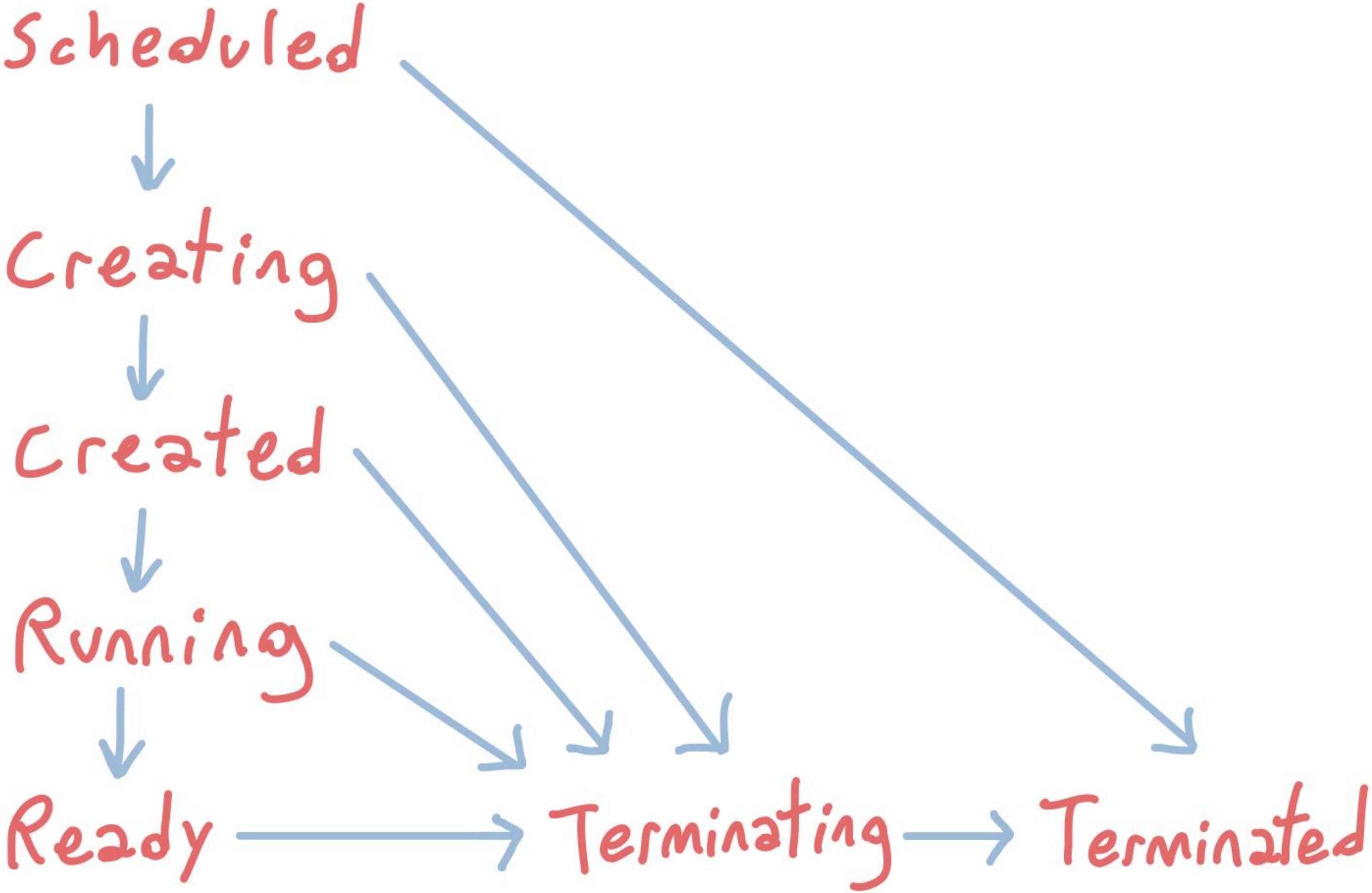
**Setpoint**

**Process Variable**

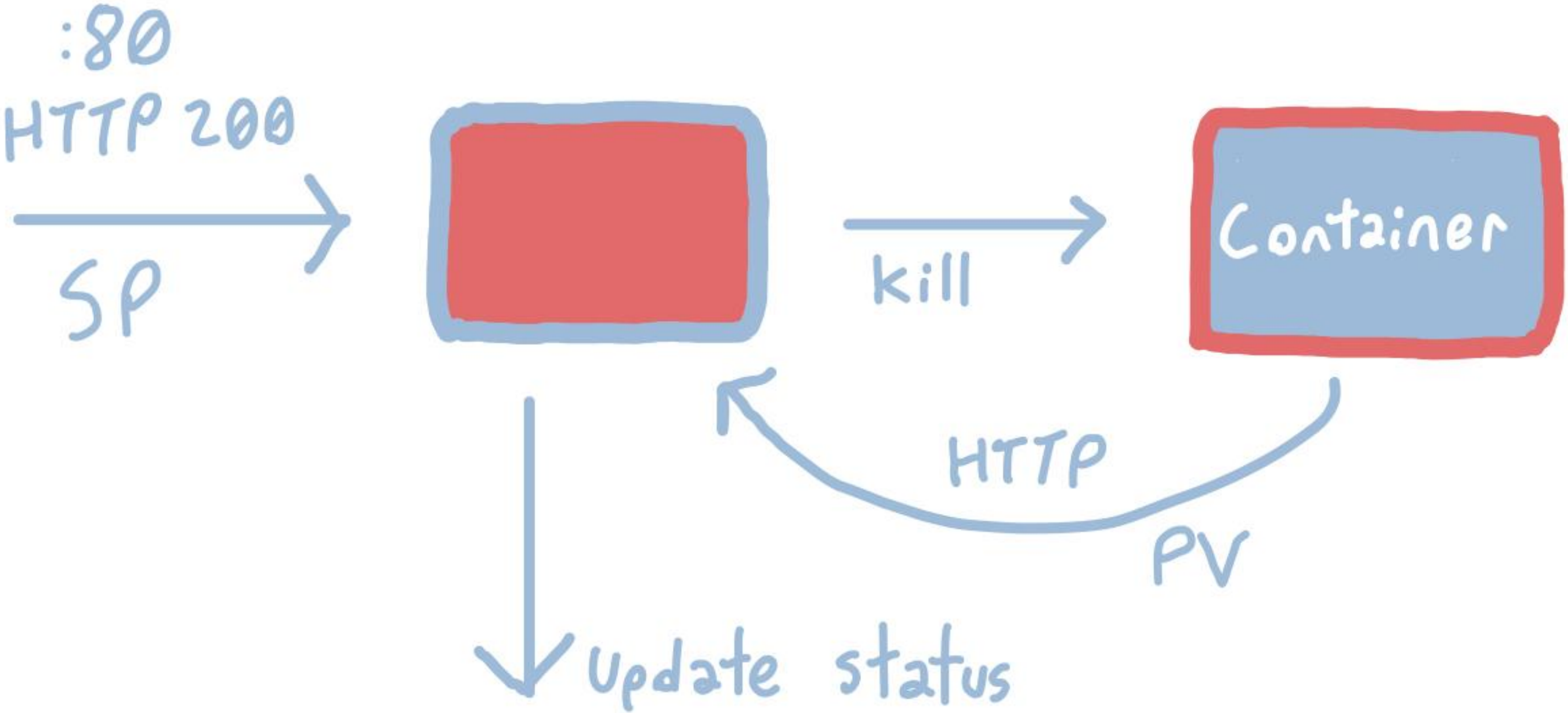
# Container Lifecycle: Readiness Probe

- When a container is launched, we don’t want to serve it traffic before it’s ready.
- A readiness probe uses some “OK” response (EG HTTP 200) to decide when.
- What do we need to build this?
    - Container lifecycle status
    - Probe destination
    - Probe behaviour config

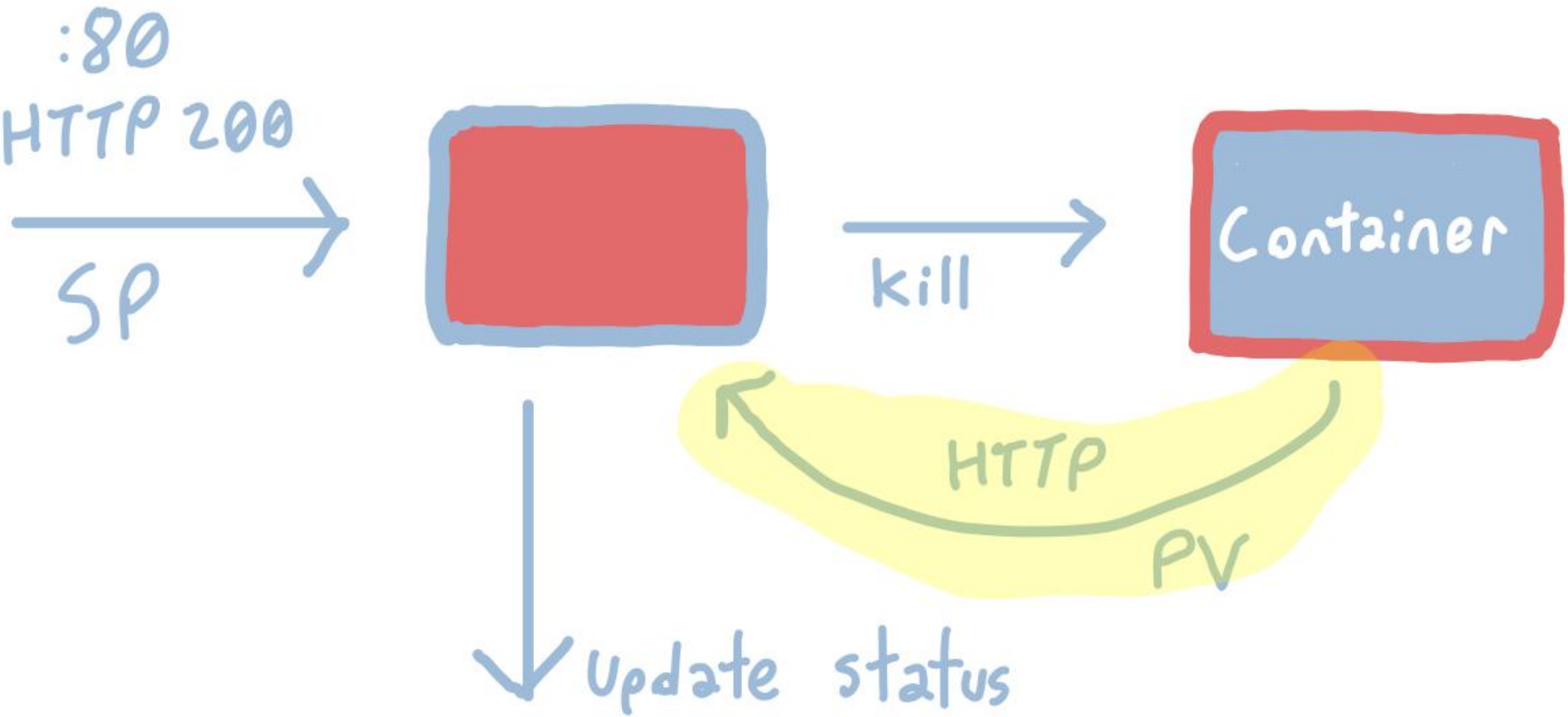
@vllry

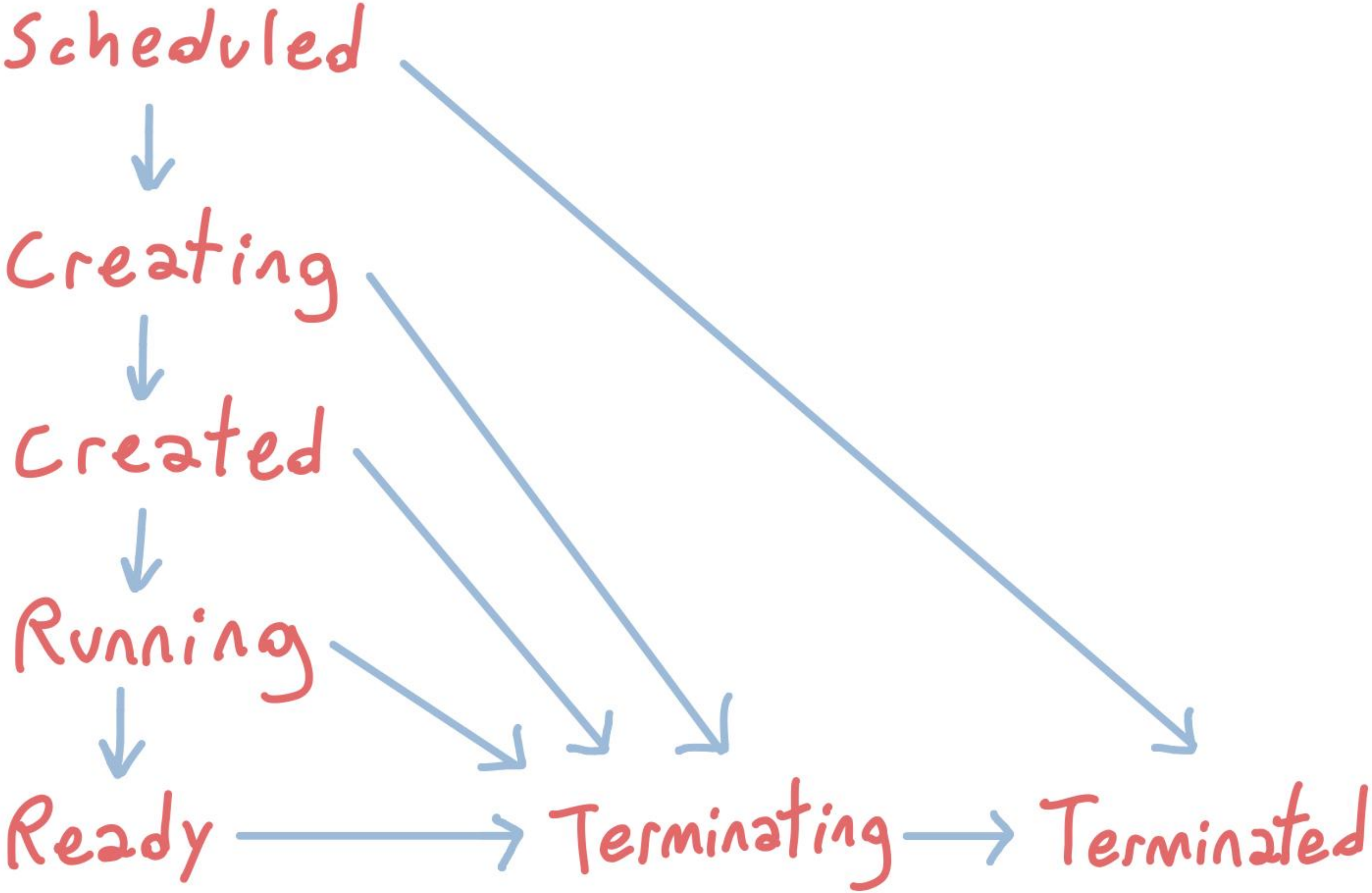


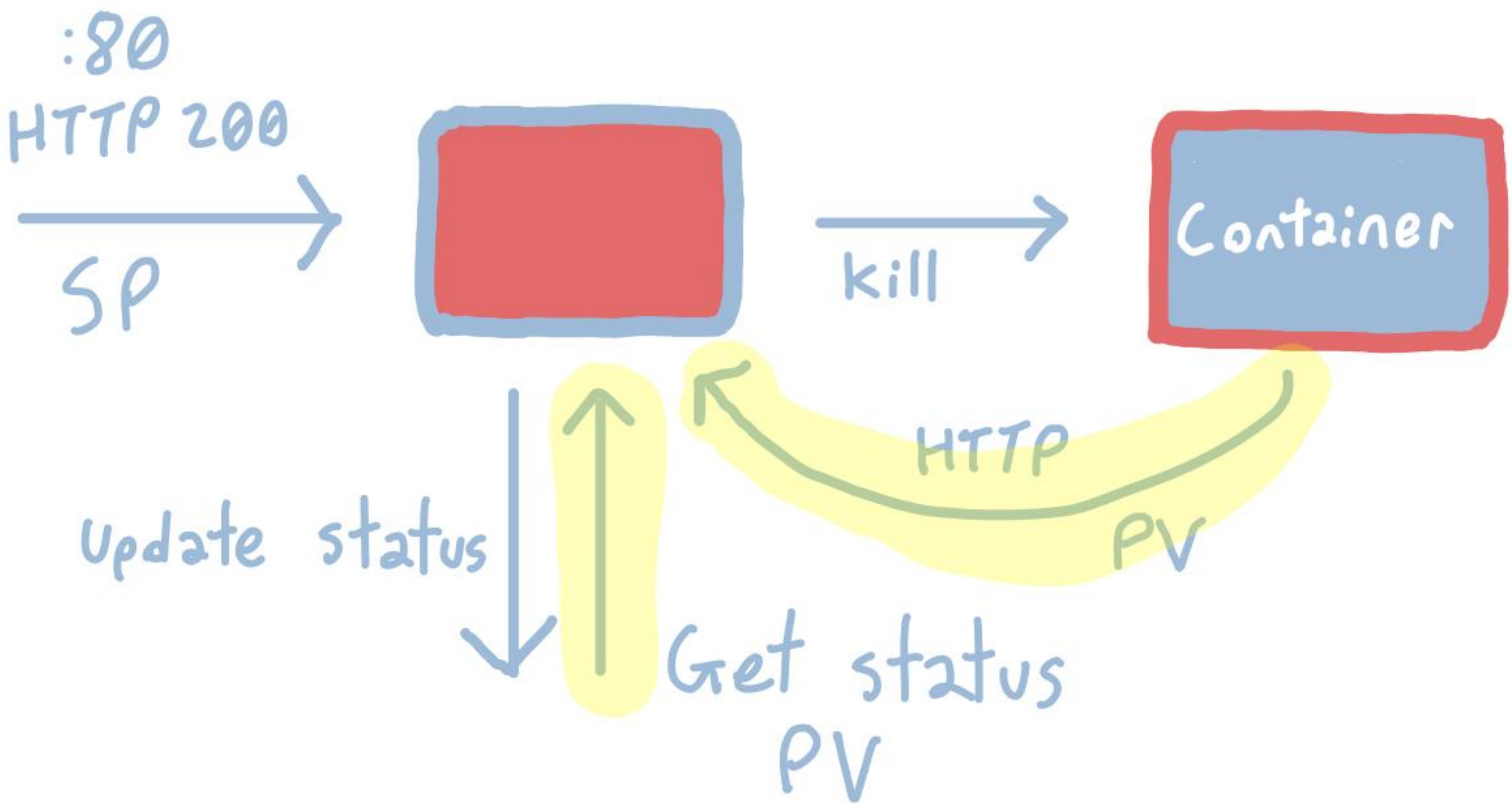
Scheduled
Creating
Created
Running
Ready
Terminating
Terminated



:80
HTTP 200
SP
kill
Container
HTTP
PV
Update status


:80
HTTP 200
SP
kill
Container
HTTP
PV
Update status


Scheduled
Creating
Created
Running
Ready
Terminating
Terminated

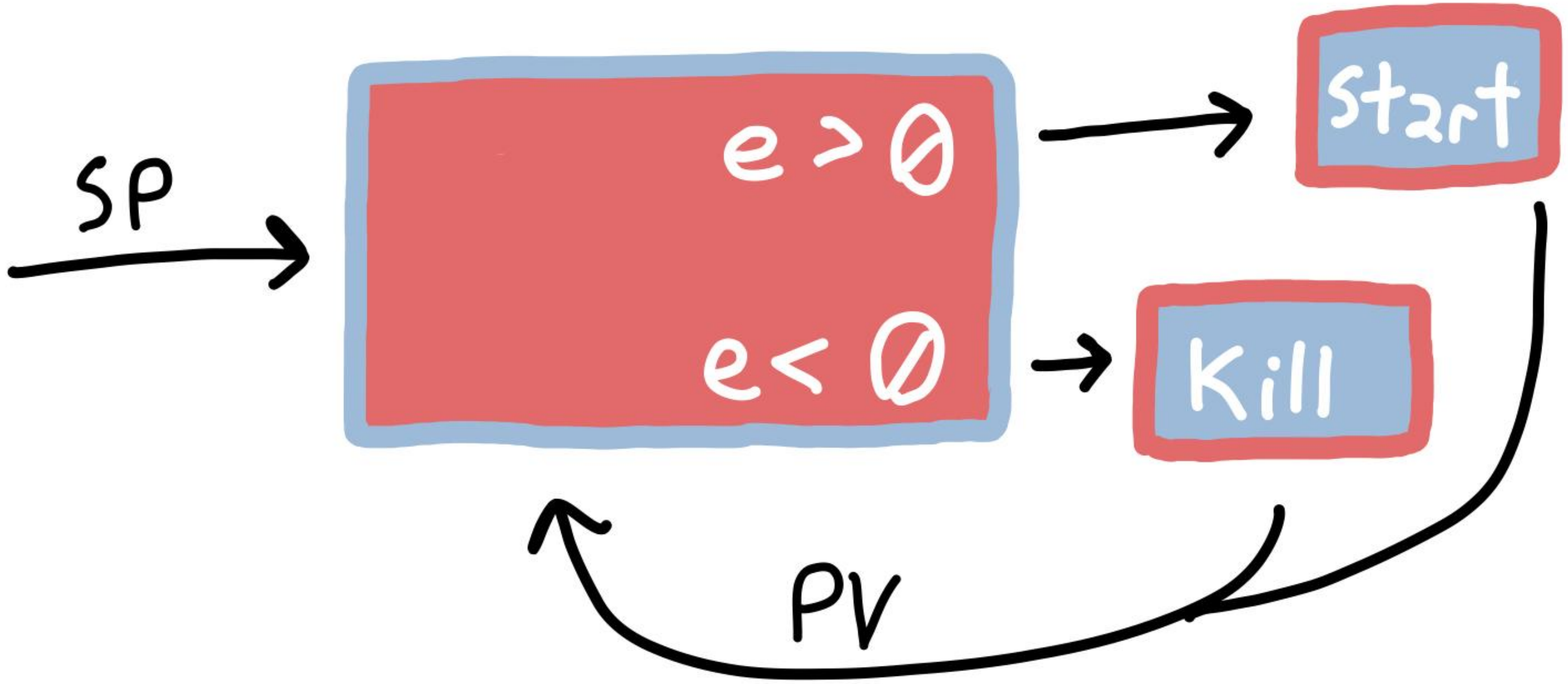

:80
HTTP 200
SP
kill
Container
HTTP
PV
Update status
Get status
PV

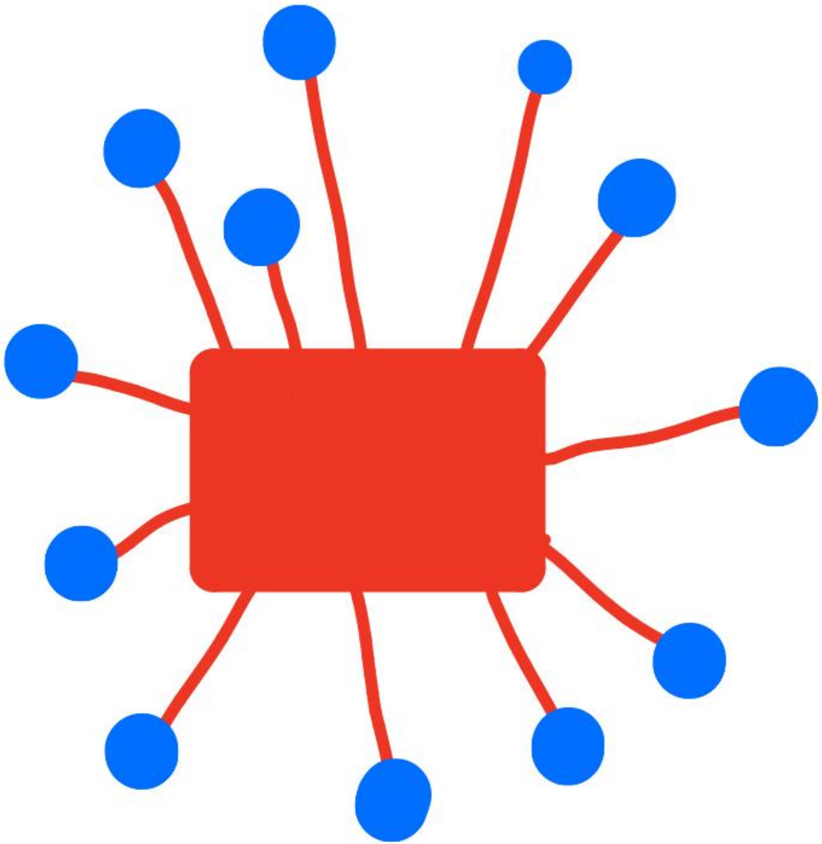
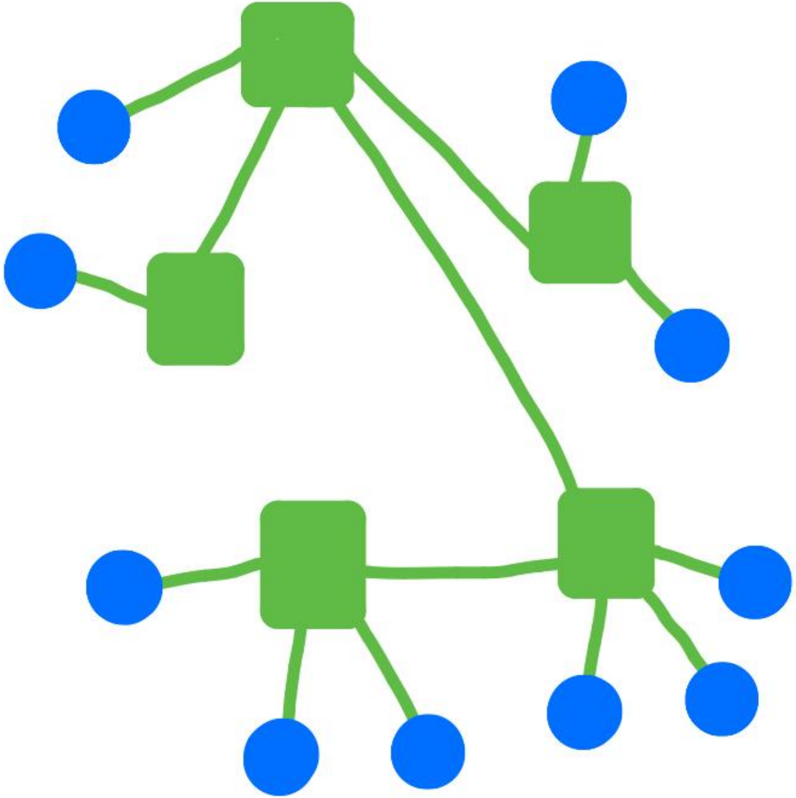
# Replica Headcount

- How do we ensure the right number of container copies exist?
- Need to maintain the desired replica count (**input**).
- Need to check the current number of containers (**feedback**).
- Need to create or terminate containers accordingly (**output**).

@vllry

# Replication Controller



# Autoscaling

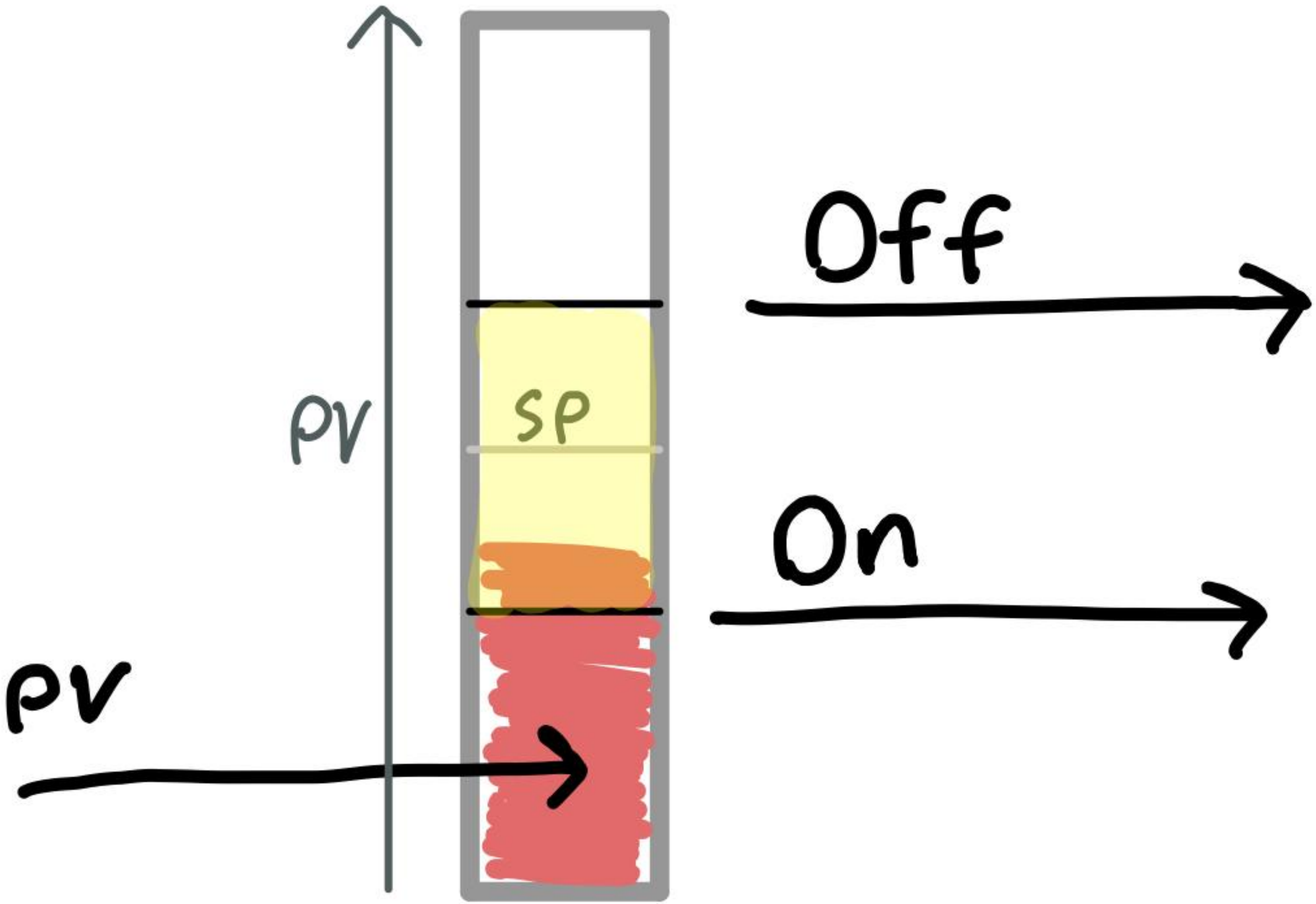
@vllry

# Autoscaling Deployments

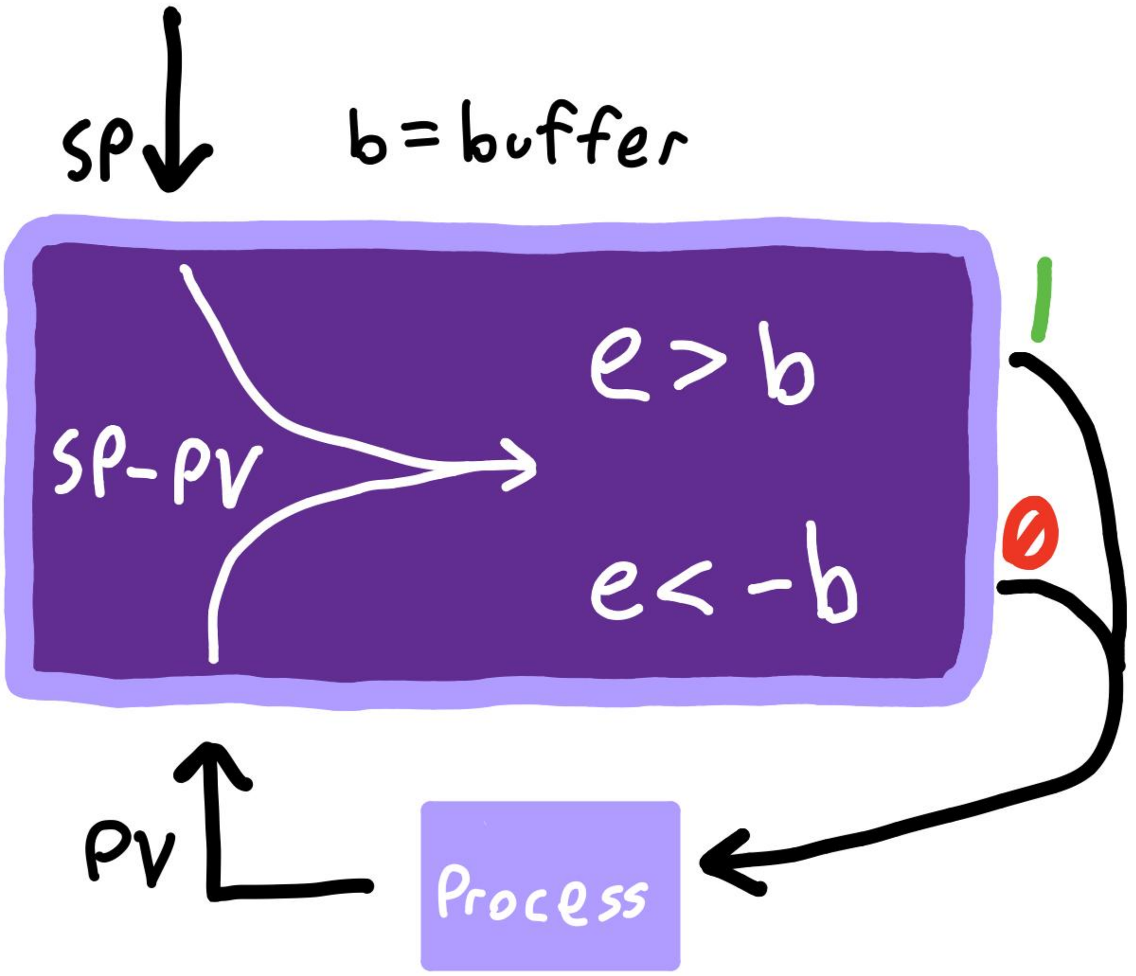
- Need to track a specified metric (CPU use, network I/O, etc).
- Need to increase or decrease replicas if the metric is sufficiently above or below the target.
- Should respond *quickly* and without *overcompensating*.

# Bang-Bang Controller

- Controller with upper and lower bounds, where the set point is never exactly met.
- Process is turned on when one extreme is hit, and turns off when the other is hit.



PV
SP
Off
On
PV



sp
b = buffer
sp-pv
e > b
e < -b
1
0
pv
Process

# Challenges in Designing a Controller

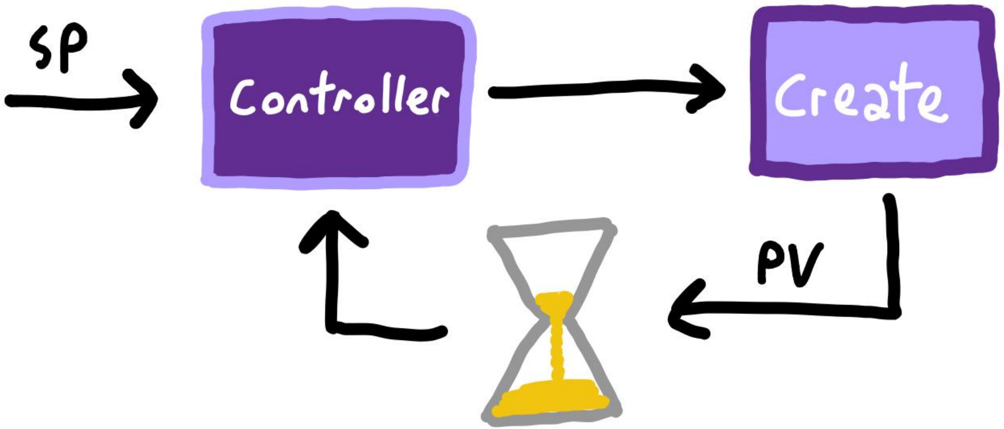
- Accepting a “close enough” error, rather than thrashing.
- Responding quickly without overcompensating.
  - Predict the right replica setpoint.
  - Account for the delay in SP->PV propagation.

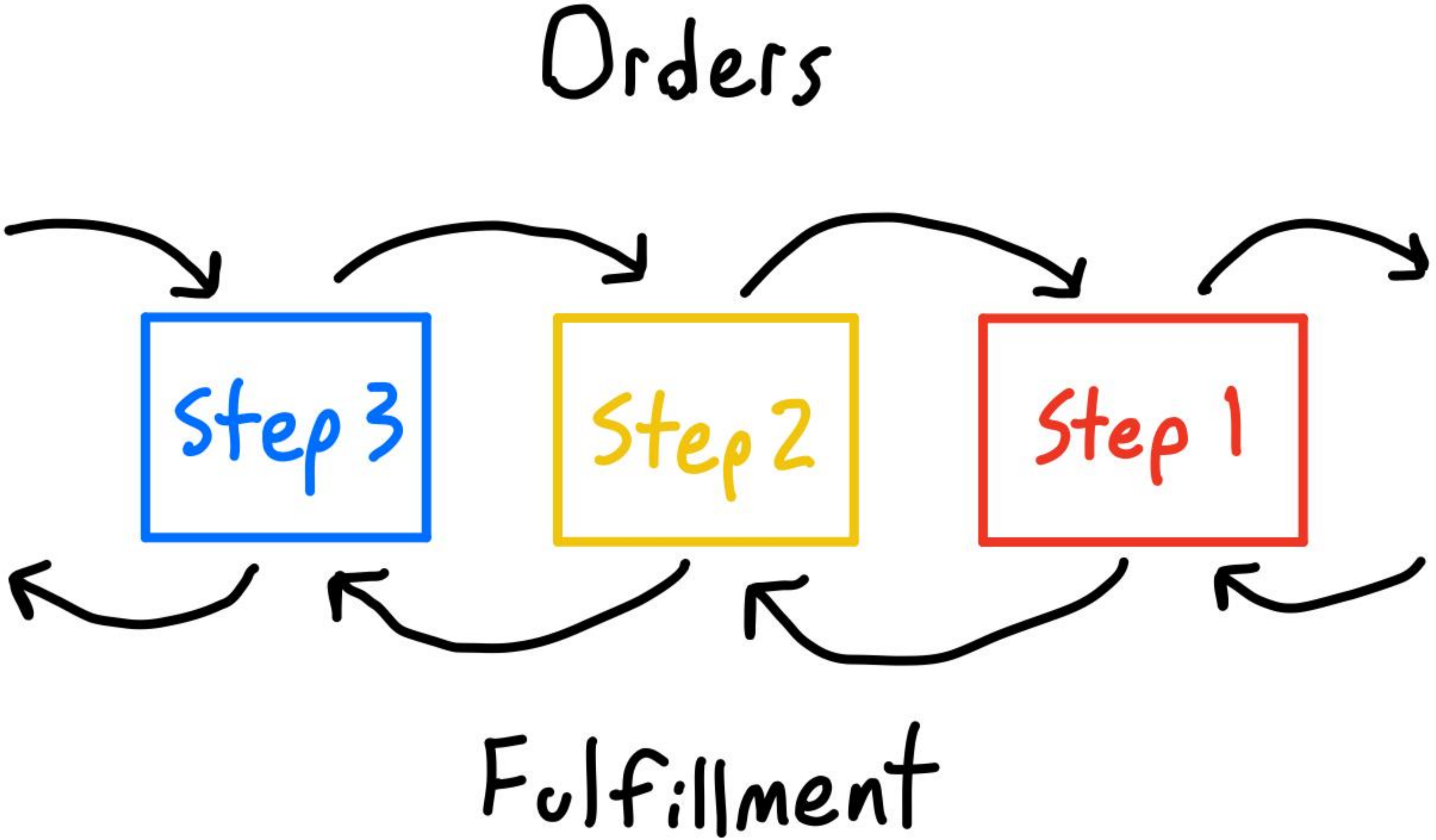
@vllry

# Delayed Response

# Bootup Time

- Containers take time to boot (surprise!)
  - Resource allocation.
  - Image pull & app startup.




Orders
Step 3
Step 2
Step 1
Fulfillment

# Accounting for the Delay

- Must guess if no context.
  - Can wait out the grace period, or...
  - Can define some % of the grace period to overscale after.
- Custom controllers can allow context.
  - Can have a statistical explanation of boot time.
  - Can use a custom readiness probe that shows progress (whitebox).

@vllry

# Matching Demand

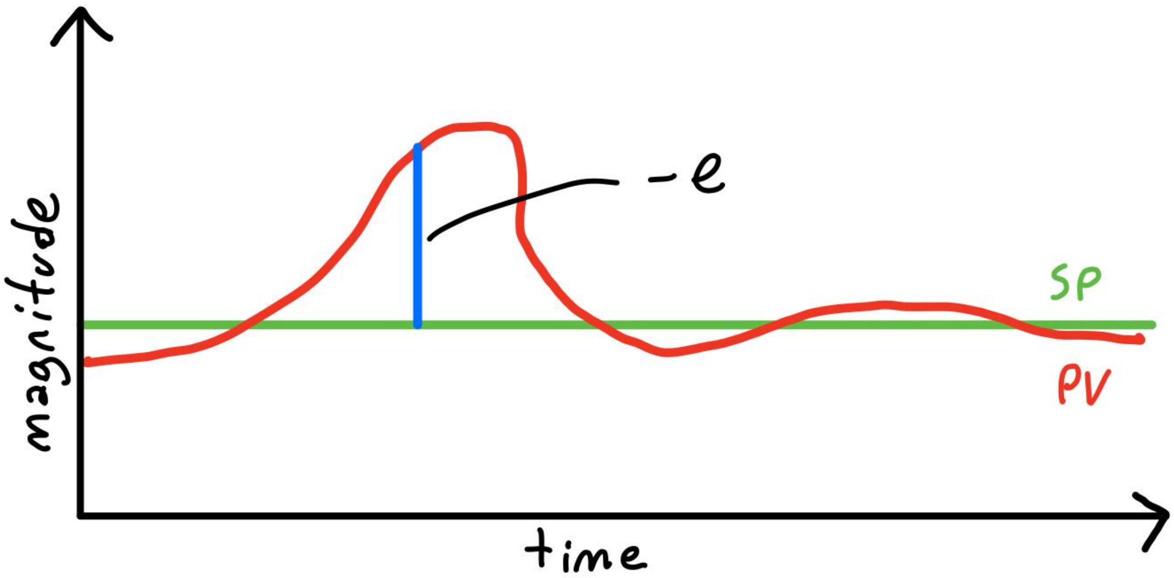
@vllry

# Scale Ramp-Up

- Scaling up quickly is especially important.
- Typical controller approaches:
    - Immediately add enough replicas to satisfy **`load/replicas`** for current load.
    - Keep scaling up each loop, until satisfied.
- Can we keep scaling both fast and precise?

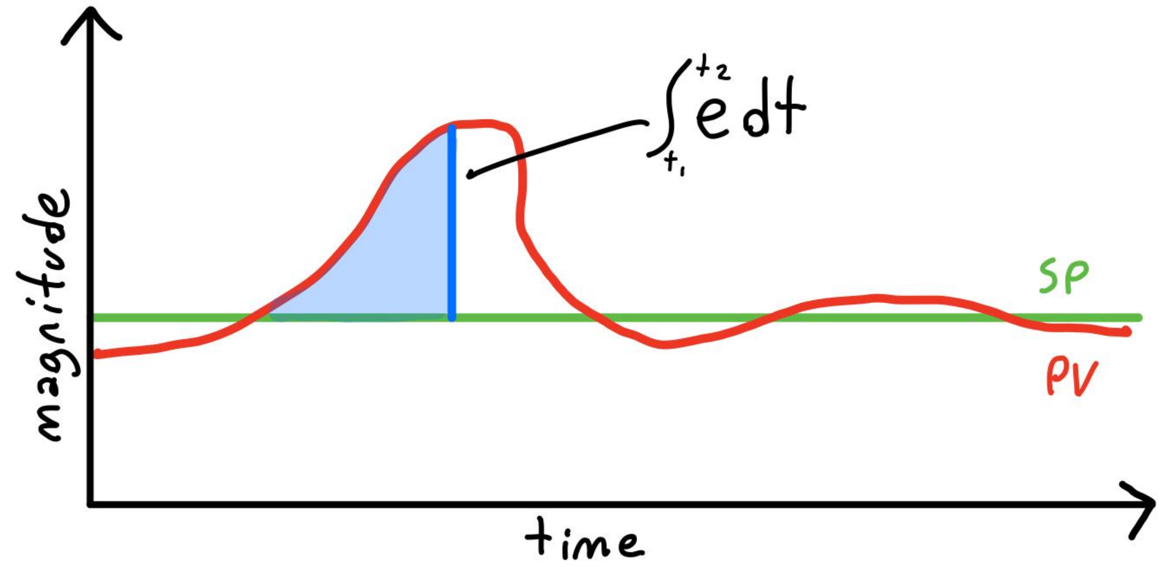
# PID: Proportional

The proportional component is a linear response to the magnitude of the error.



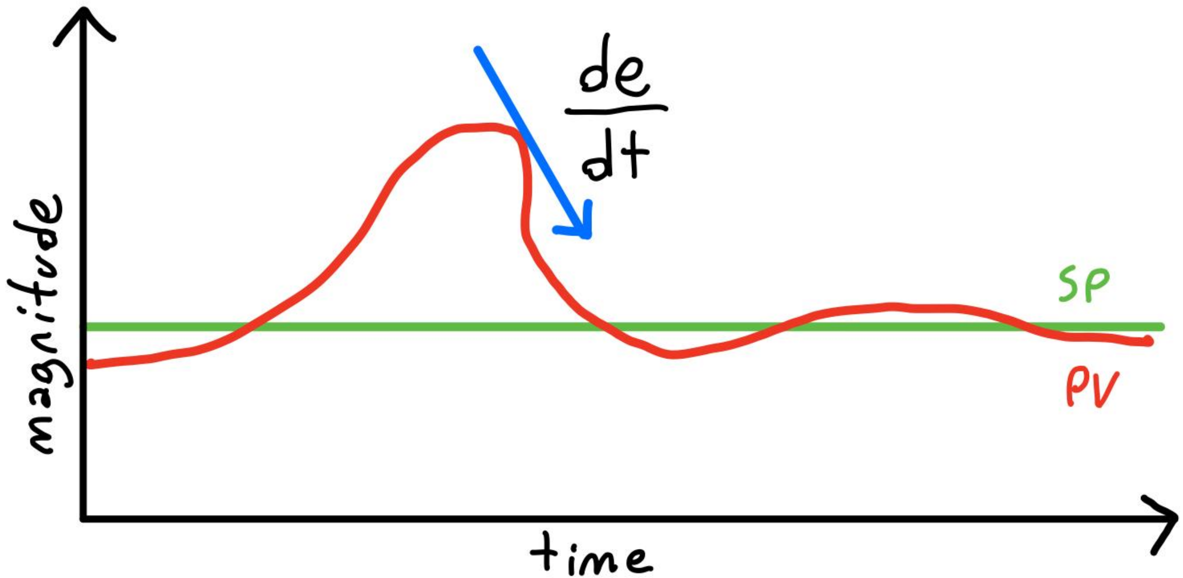
# PID: Integral

The integral component is a *compensator*. It responds to the *magnitude* and *duration* of the error.



# PID: Derivative

The derivative component is a *predictor* of the future error, based on the trend of the current error.



# PID Controllers

- Use the *proportional*, *integral*, and *derivative* components to *react*, *compensate*, and *predict* for required output.
- Each component is tuned using a constant.

$$c_p e(t) + c_i \int_{t_1}^{t_2} e(t)\,dt + c_d \frac{de(t)}{dt}$$

# Autoscaling With a PID Controller

- Proportional and integral components drive scaling.
- Integral and derivative components increase scale speed, at the cost of overcompensating.
- Derivative is “less accurate” but can help in sharp raises/drops.

# Autoscaling in Kubernetes

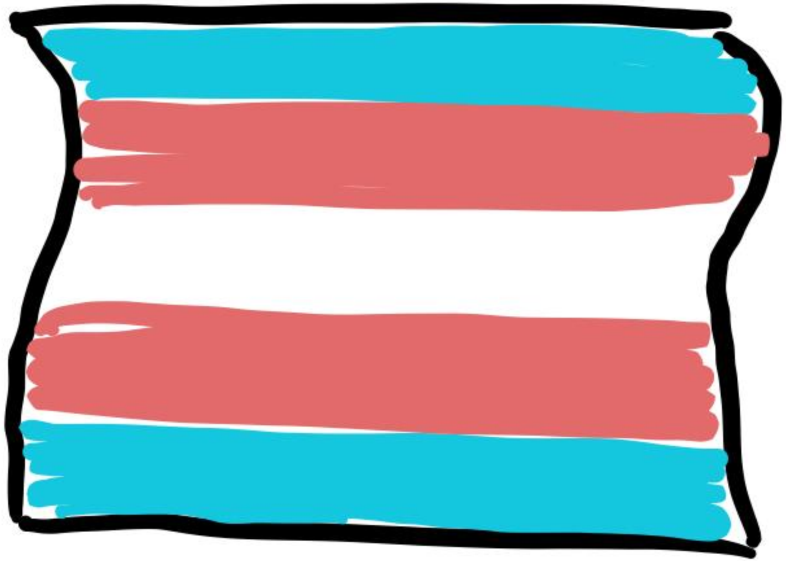
- Kubernetes uses a proportional controller (with a lot of checks and balances.
- Prioritizes gradual resolution over unstable resolution.
- Scaling (Horizontal Pod Autoscaler) updates Deployment spec - doesn’t touch pods itself.

# In Summary

- Ensure any controller has the necessary feedback to properly achieve its outcome.
- Strictly define expectations of any controller.
- Build discrete, transparent, and testable controllers.
- Ensure shared state has a single source (CP).
- Custom controllers are common based on app behaviour and expectations.

# Oh Yeah, Hi!

- I'm a software/systems person at Checkfront (online bookings)
- I work with Kubernetes & "cloud stuff".



@vllry

# Thank You!

Brian Liles & coordinators & staff

Joe Beda

Tim St. Clair

@vllry

# Audience Questions

@vllry