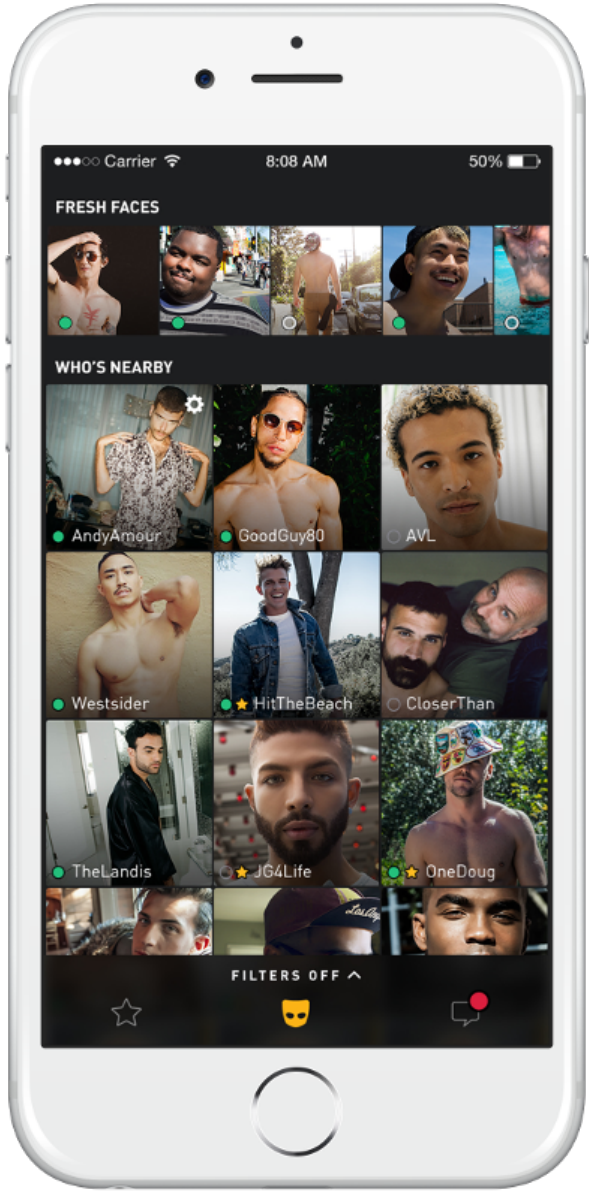
Guaranteed SLAs with higher-order functions, and no Magic Numbers!

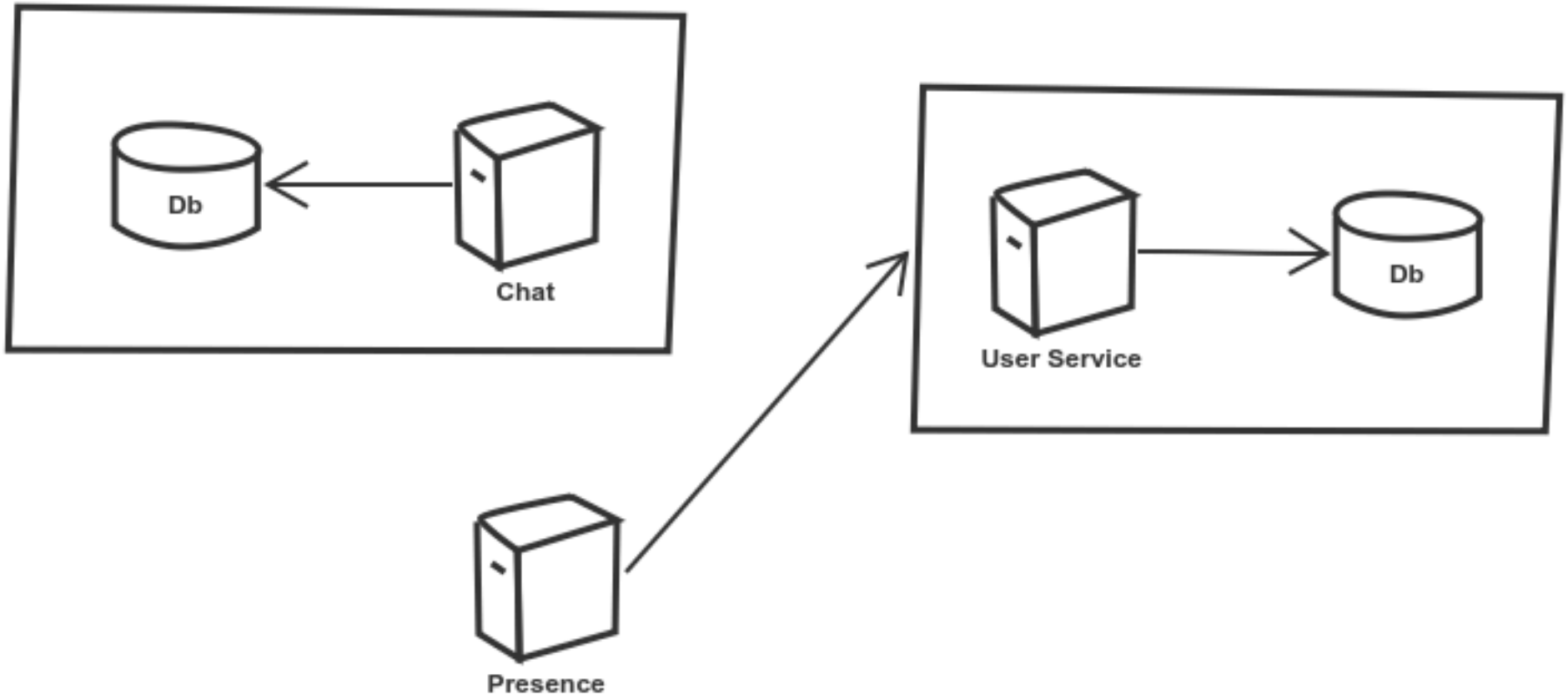
Rafał Studnicki
Lambda Days 2018

Introduction

What is Grindr?

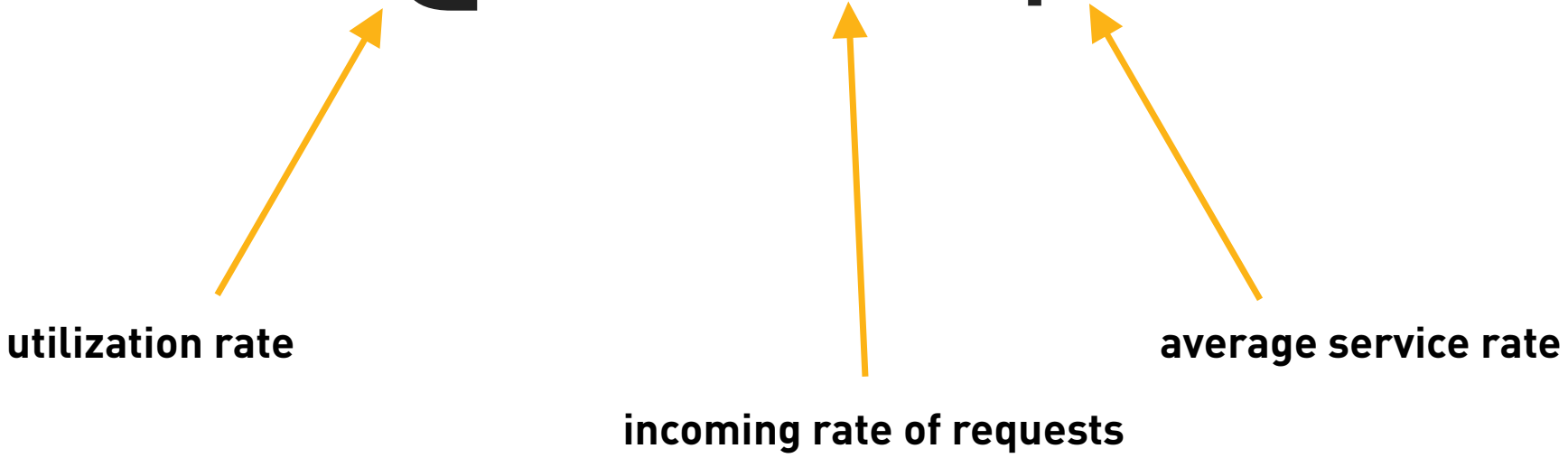


Backend stack

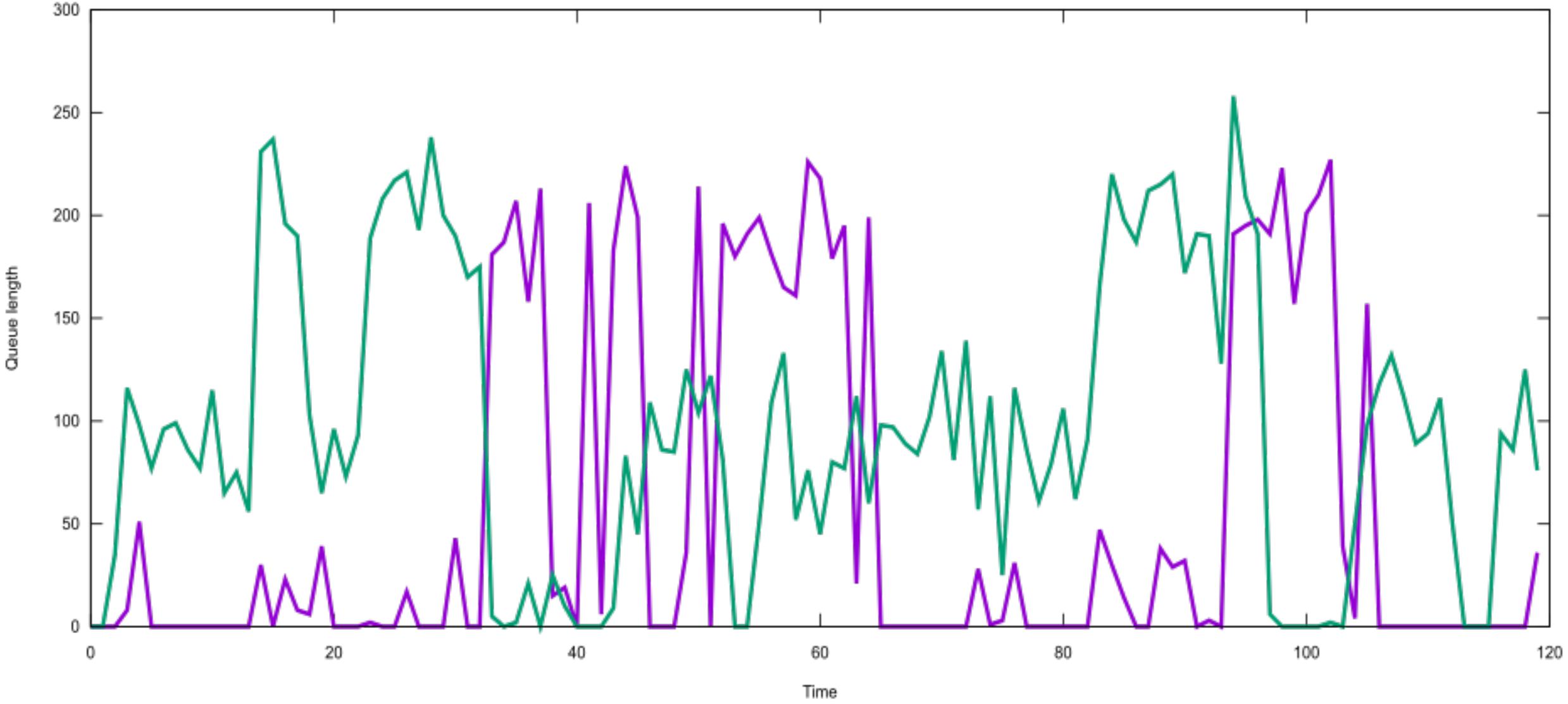


Utilization Law

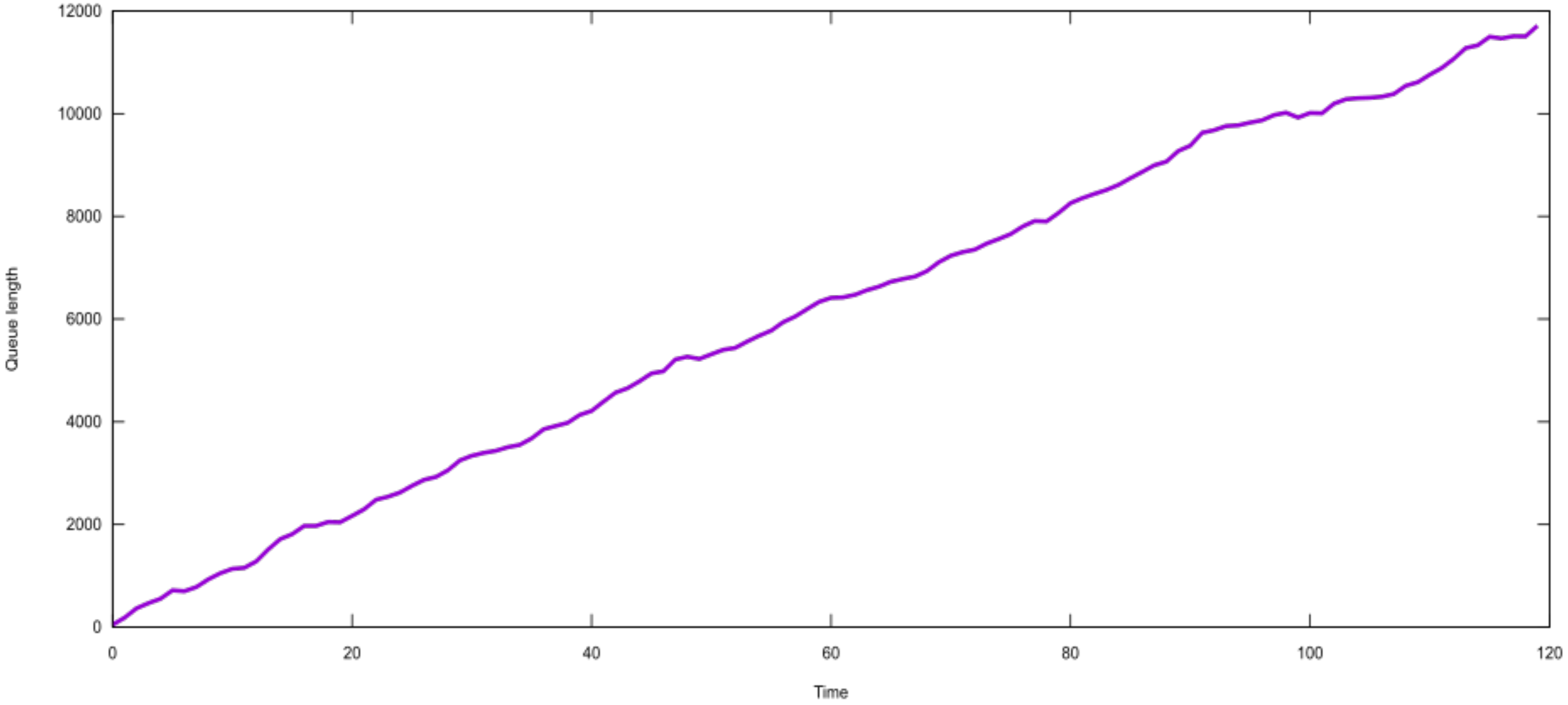
$$\rho = \lambda / \mu$$



Utilization Law illustrated λ =1000 (purple μ=2000, green μ=1100)

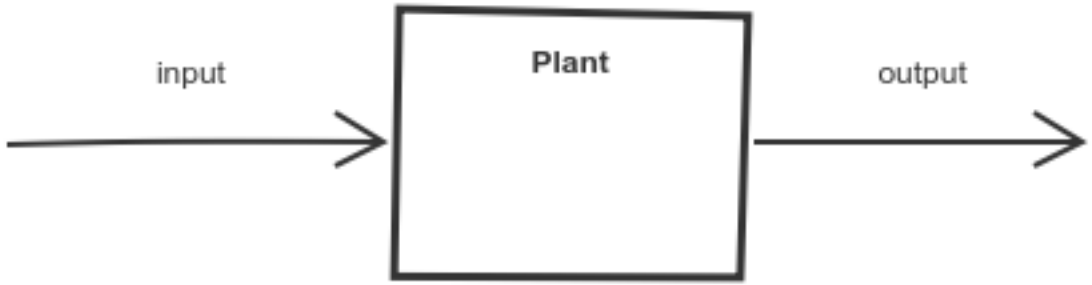


Utilization Law illustrated (λ=1000, μ=900)



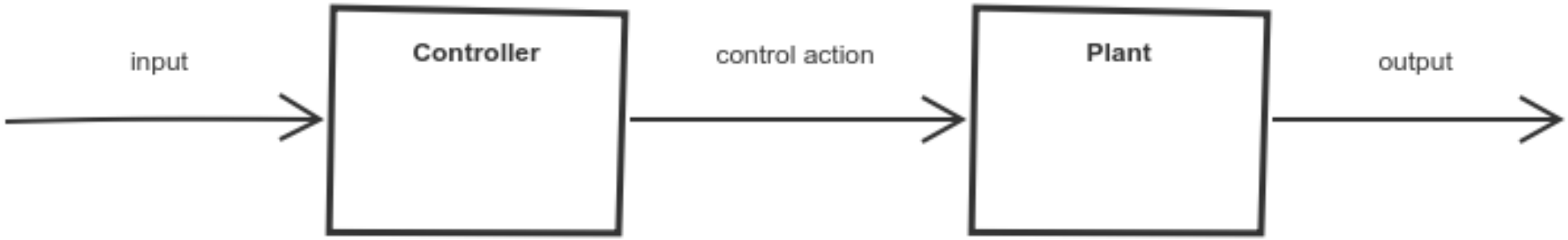
Taking control

Control Theory

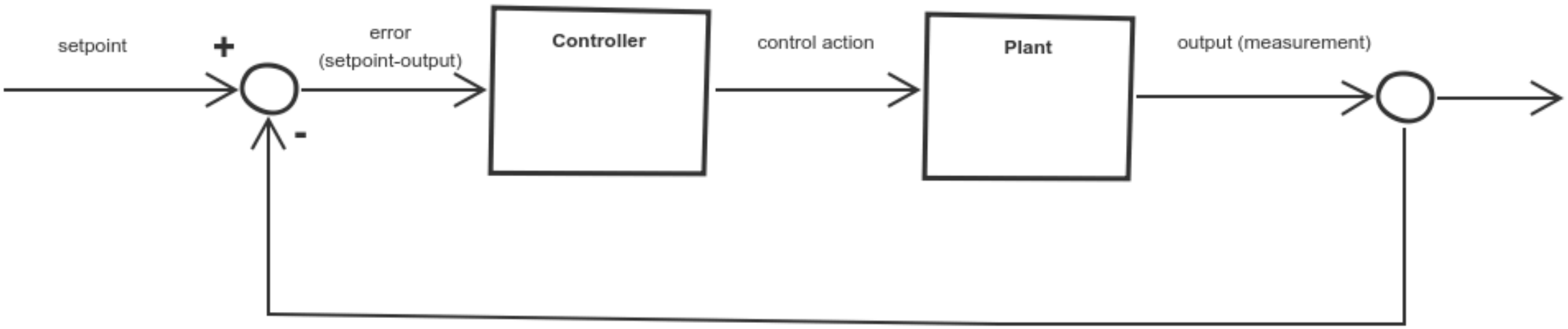



```elixir
defmodule Component do
  @callback init(any) :: state
  @callback step(state, input) :: {output, state}
end
```

Feed-forward (open loop) control



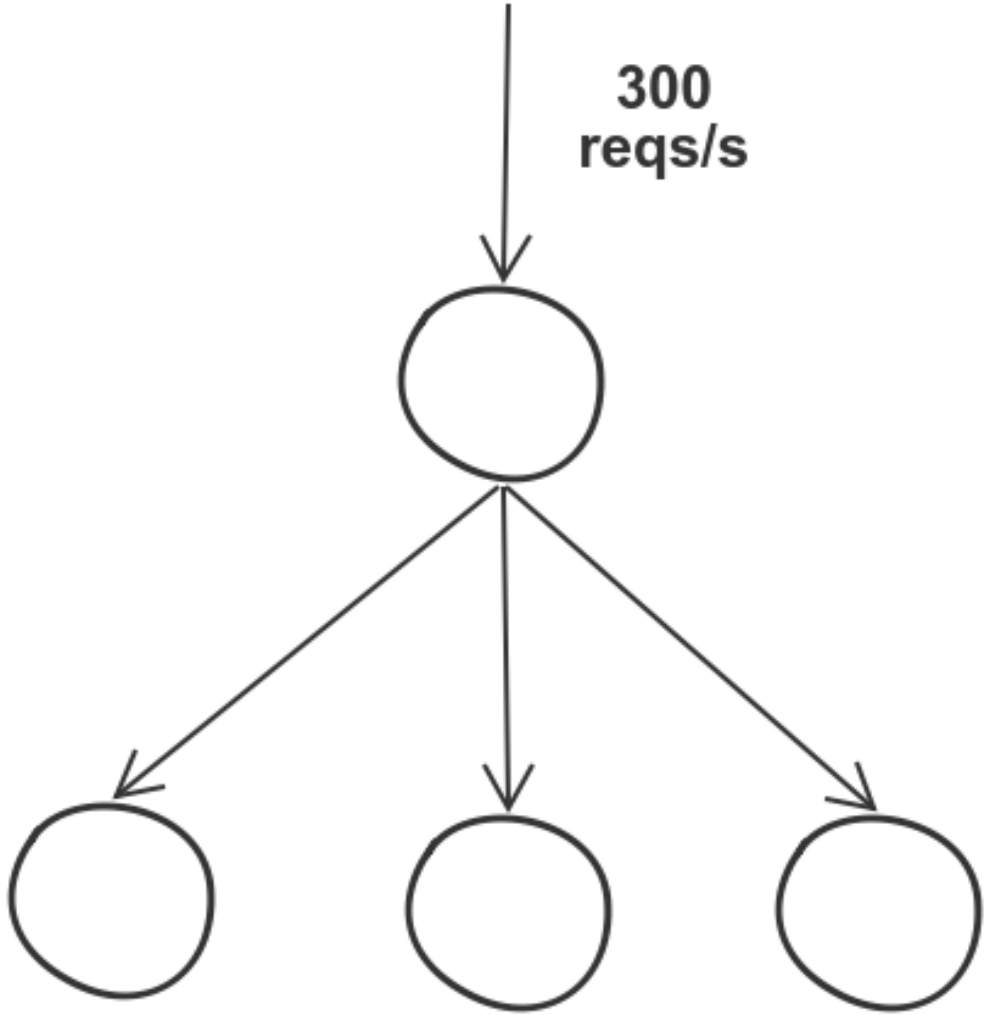
Feedback (closed loop) control



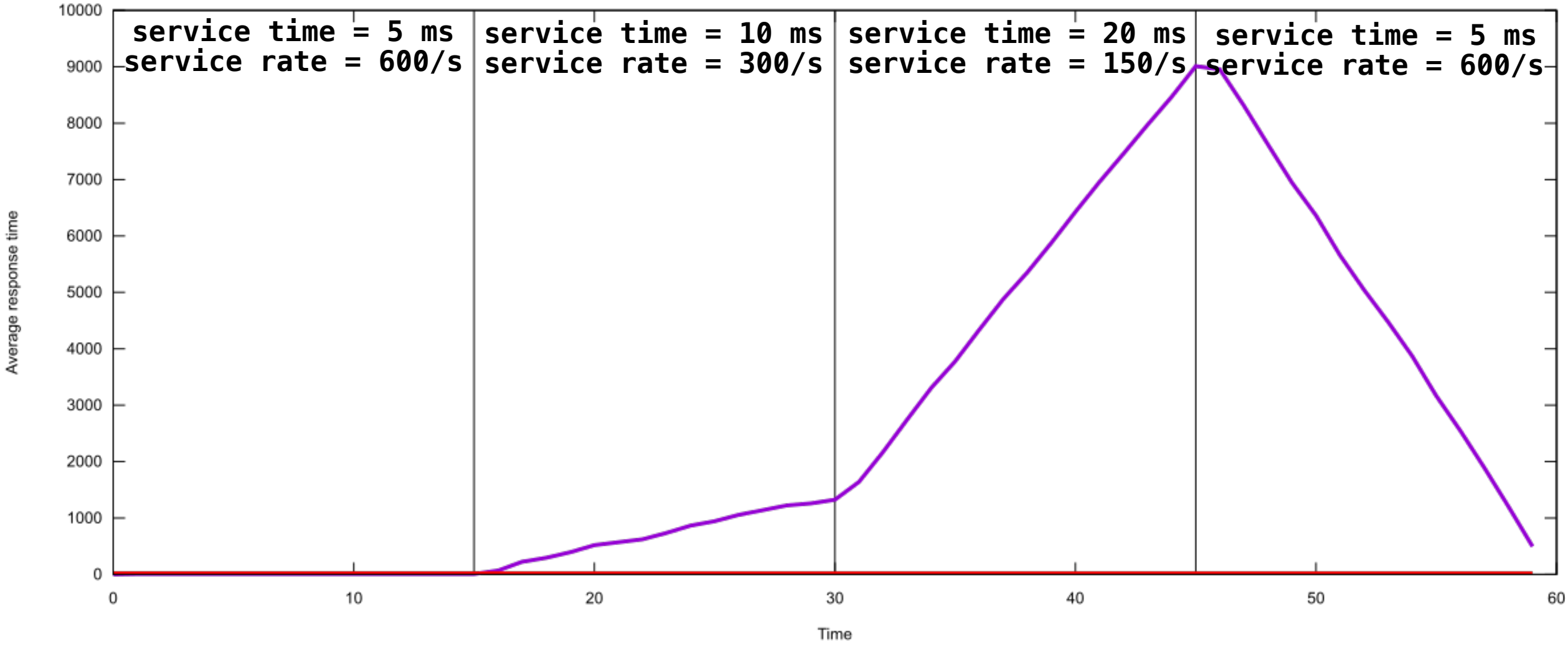
Feedback (closed loop) control

- Thermostat
- Cruise control
- Autopilot
- AWS Auto Scaling
- HTTP Live Streaming
- Redis key eviction
- TDD

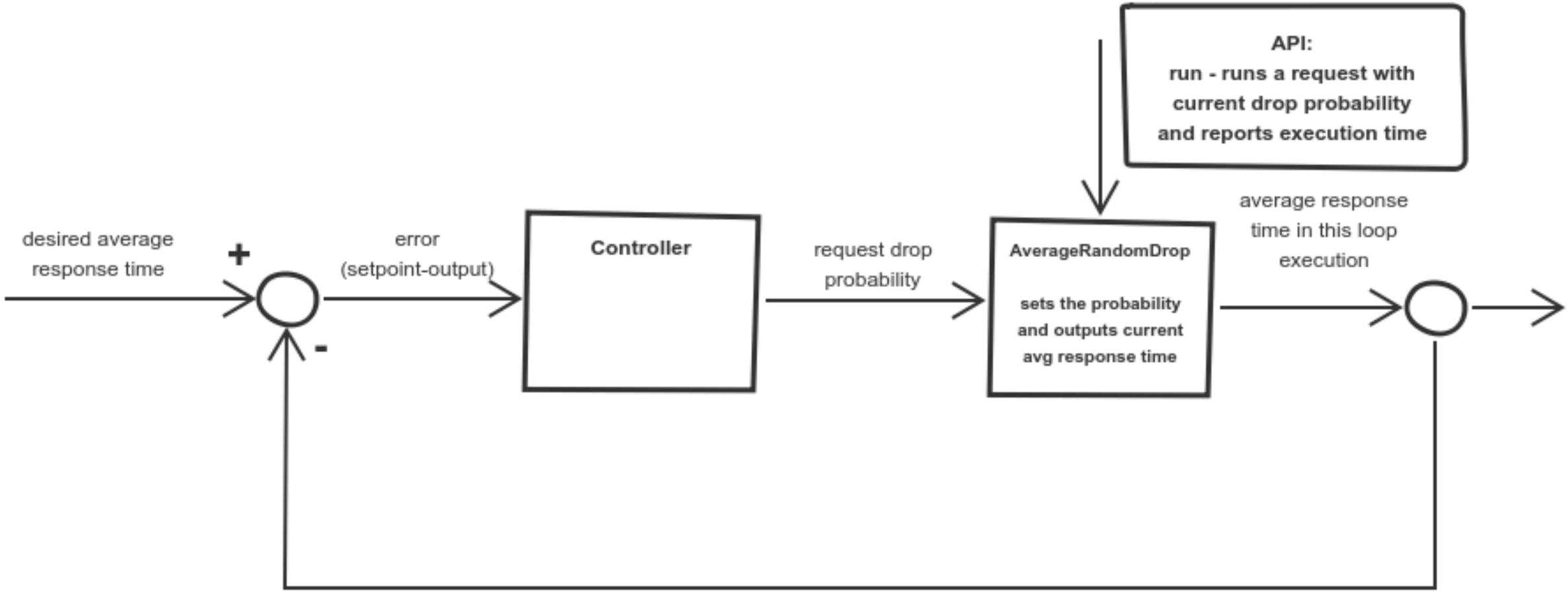
Three queues simulated



Three queues, naive approach



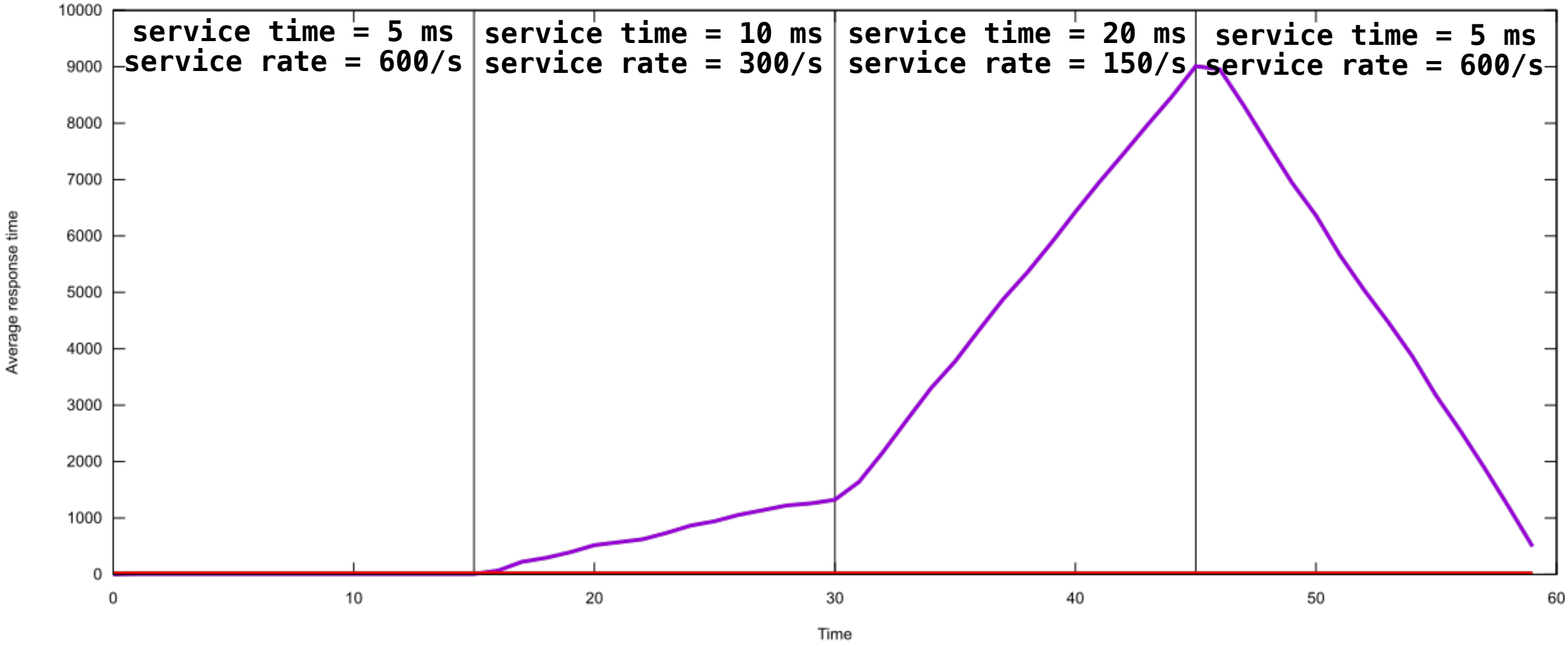
Random dropping based on average response time



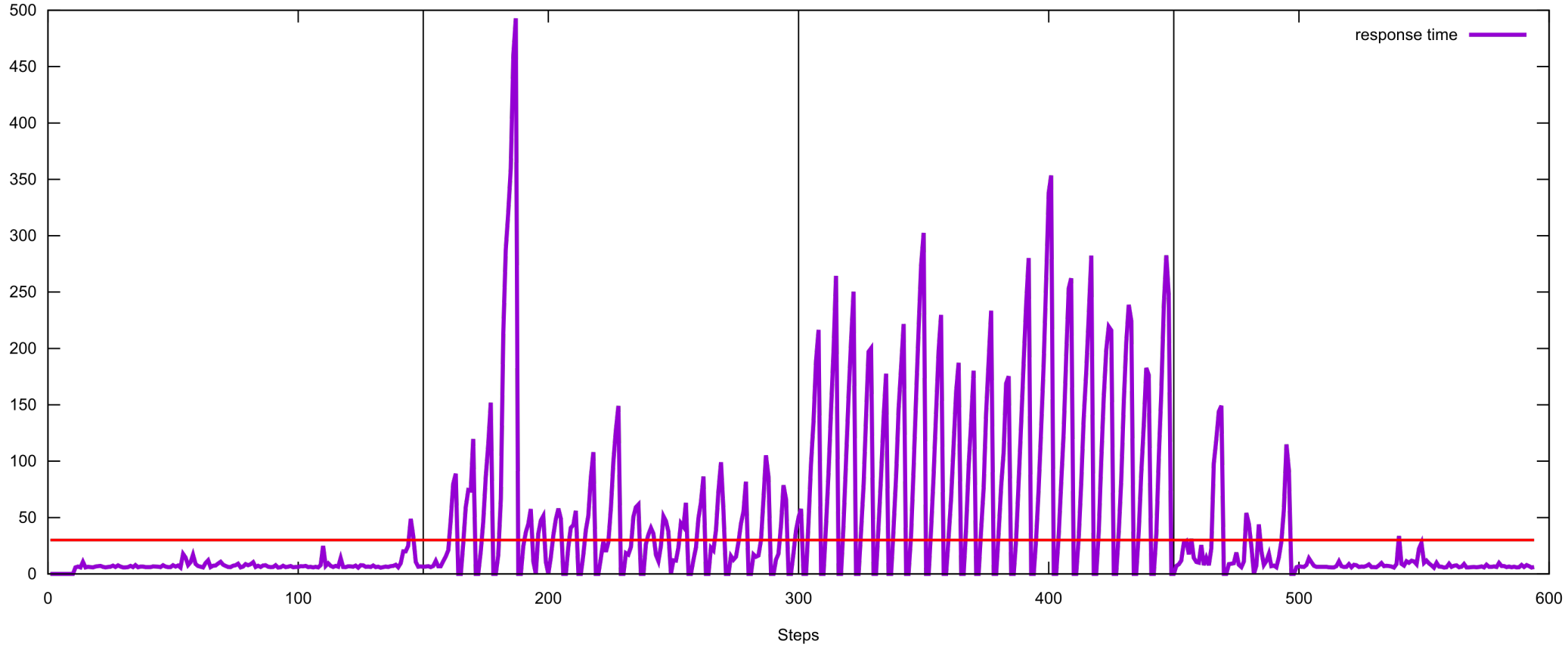
Proportional control

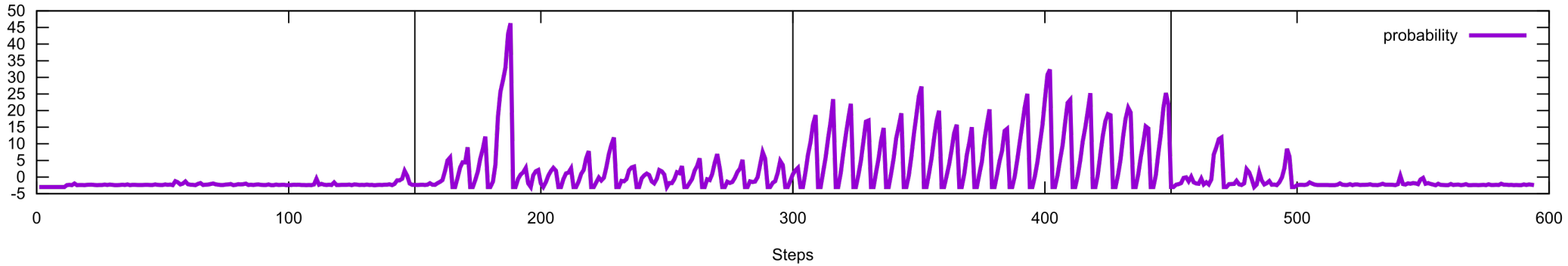
```
defmodule Controller do
  @behaviour Component

  def init(_), do: :no_state

  def step(st, input) do
    {input*@magic_number, st}
  end
end
```

Three queues, naive approach

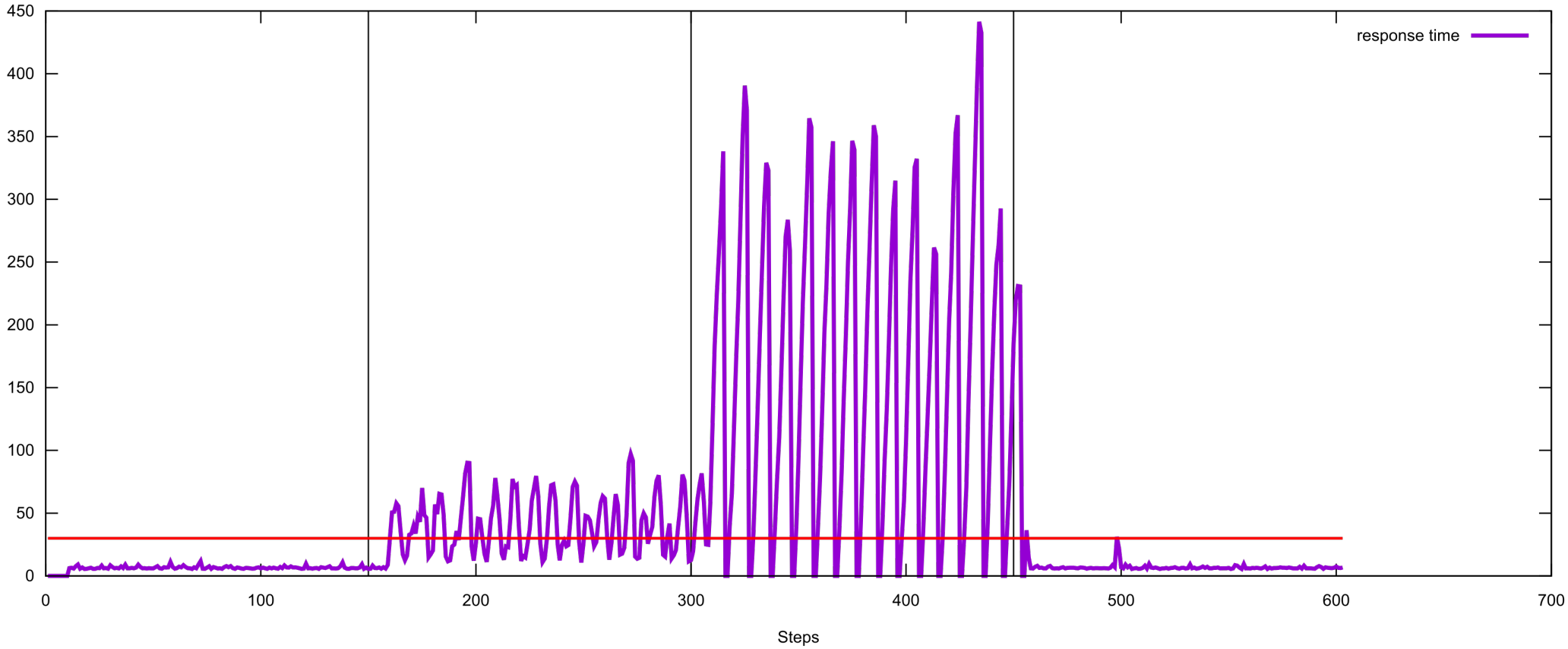


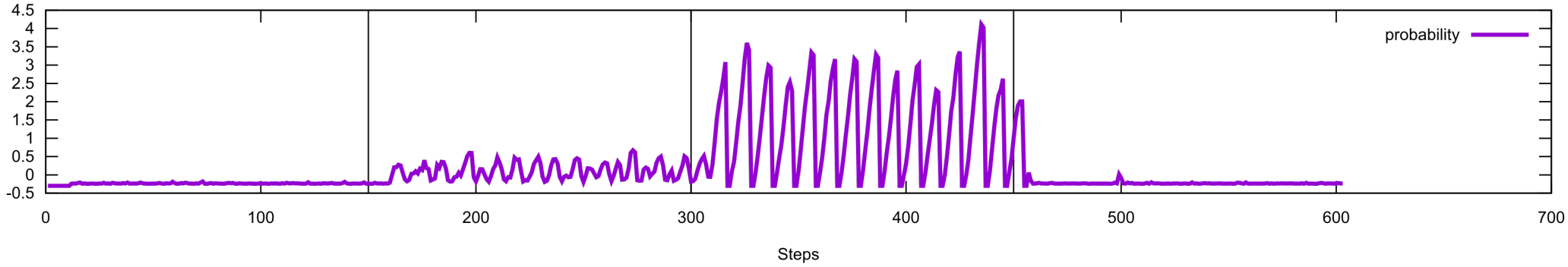
magic_number = 0.1



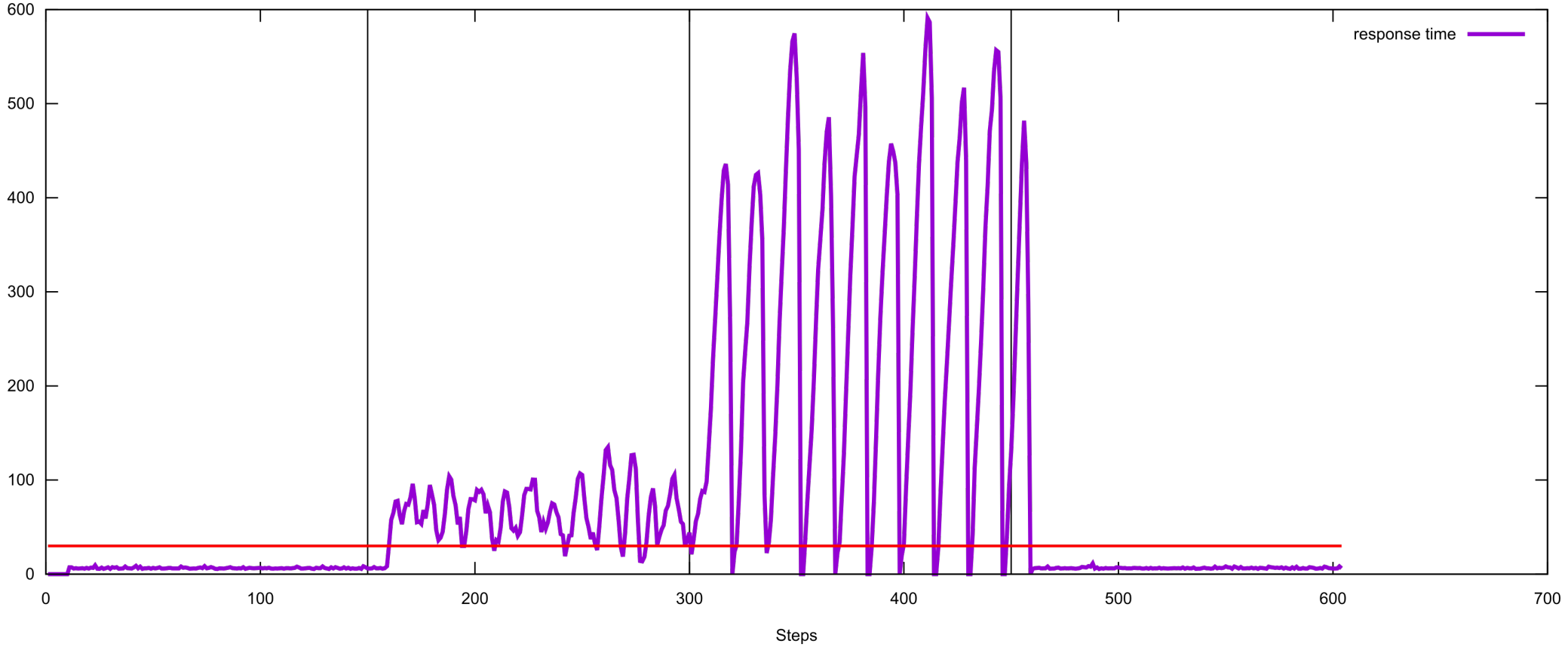


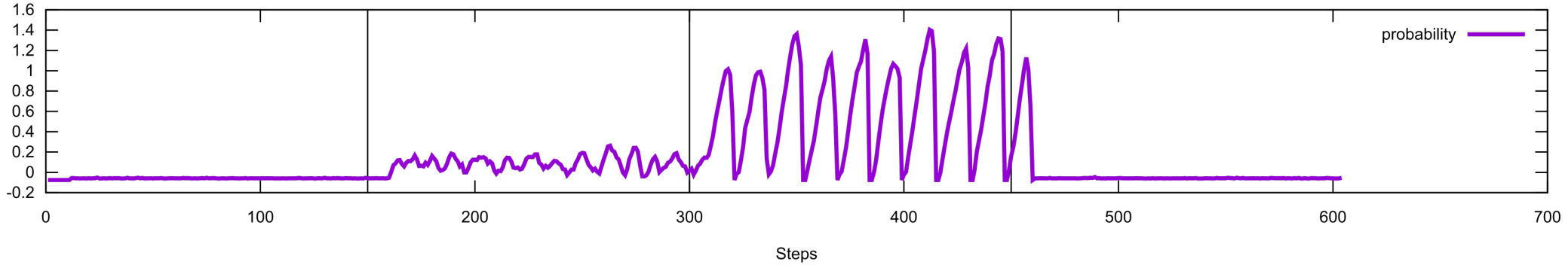
magic_number = 0.01





magic_number = 0.0025

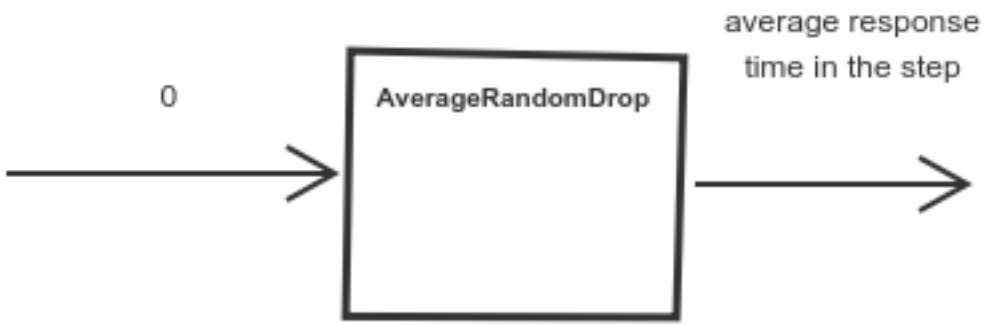




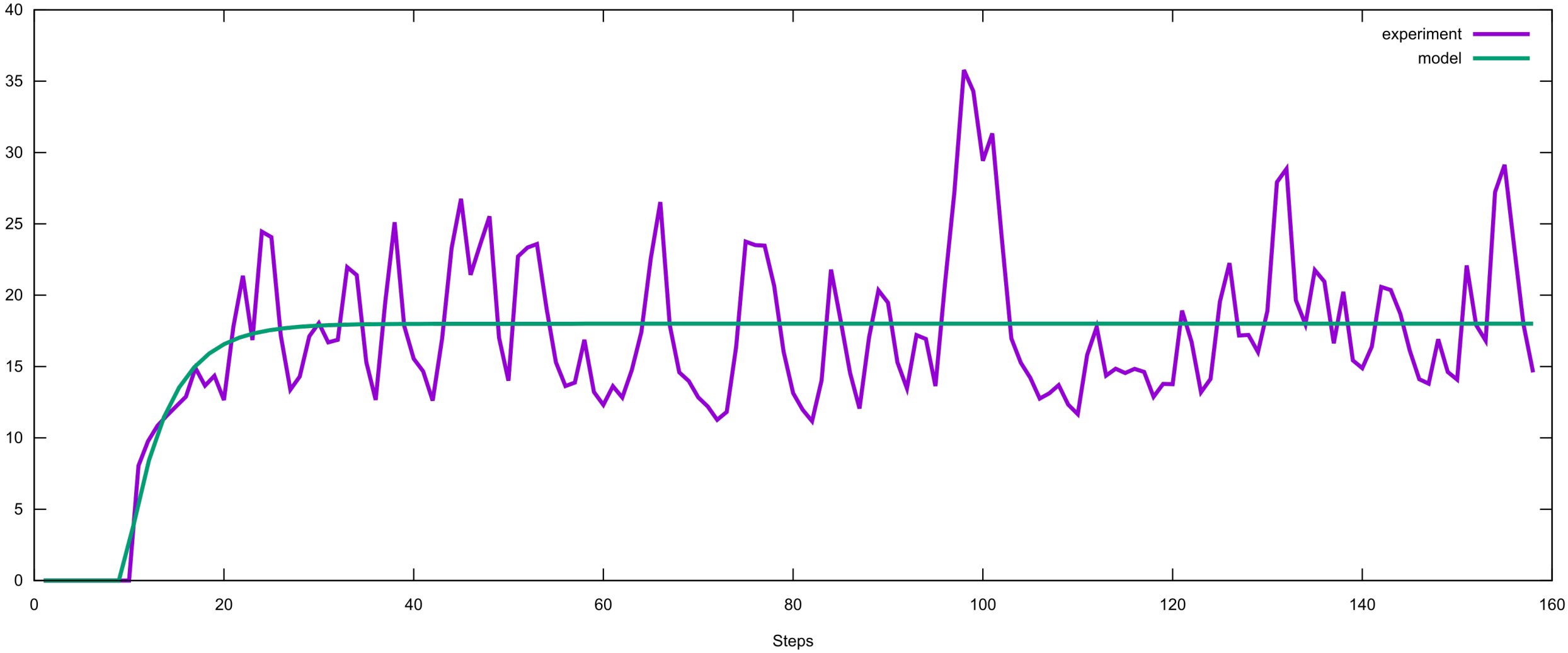
Proportional and integral controller

```
defmodule Controller do
  @behaviour Component

  def init(_), do: 0

  def step(i, input) do
    {input*@magic_number1 + (i+input)*@magic_number2, i+input}
  end
end
```

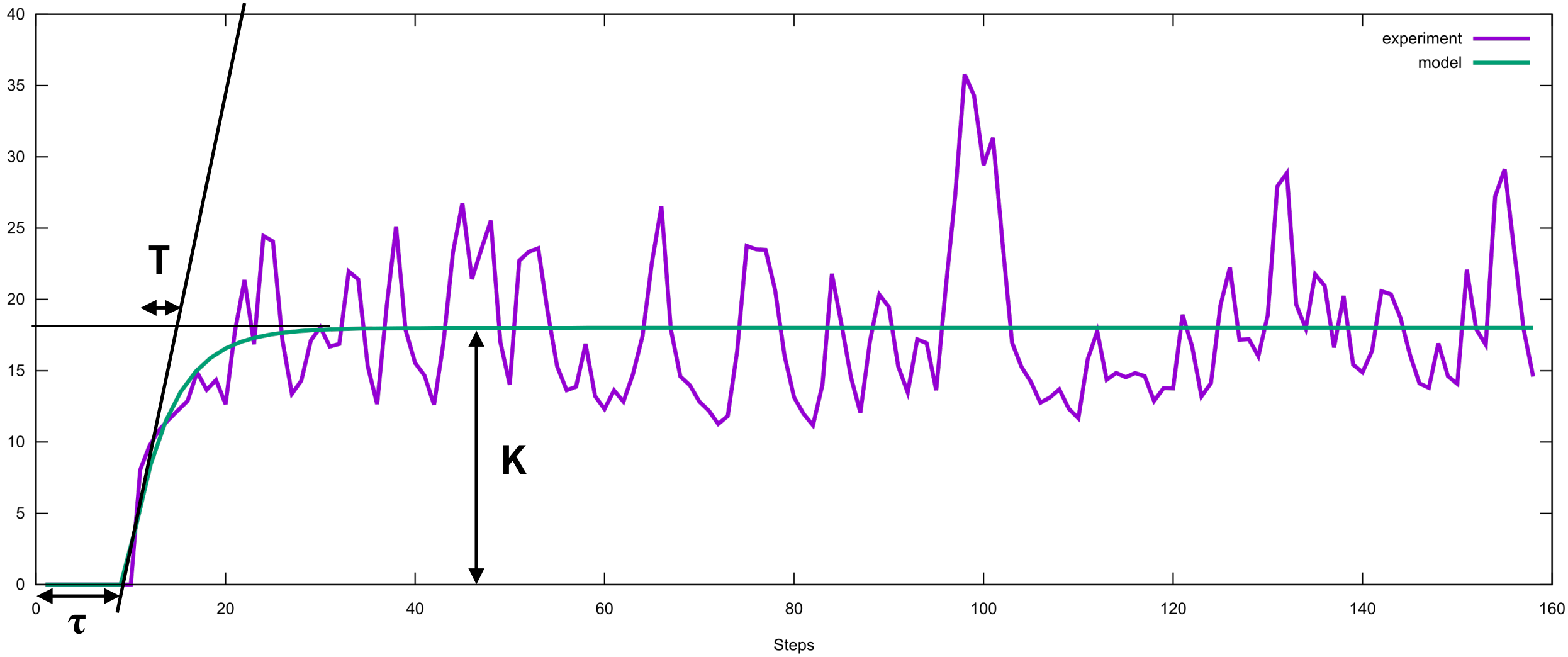
Step response experiment



Step response experiment



Step response experiment



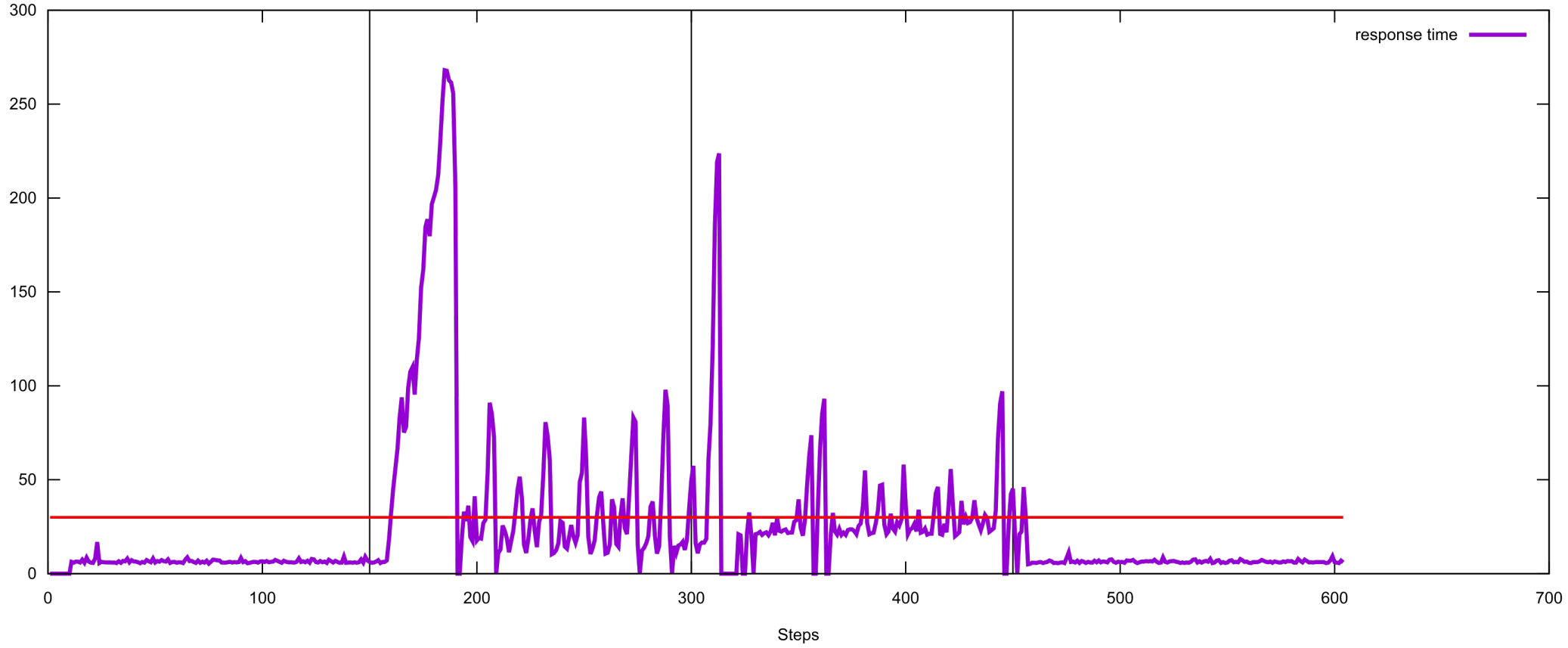
Step response experiment

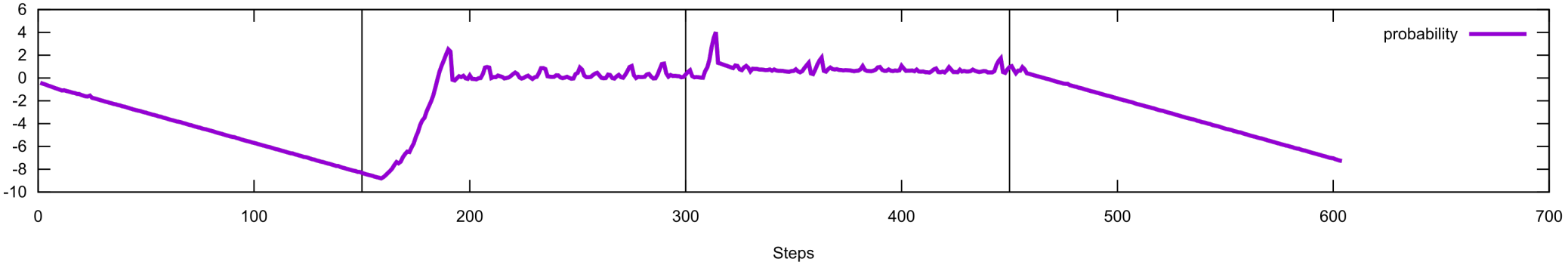
```
K   = 18.0016
T   = 4.16737
tau = 9.46876
```

Step response experiment

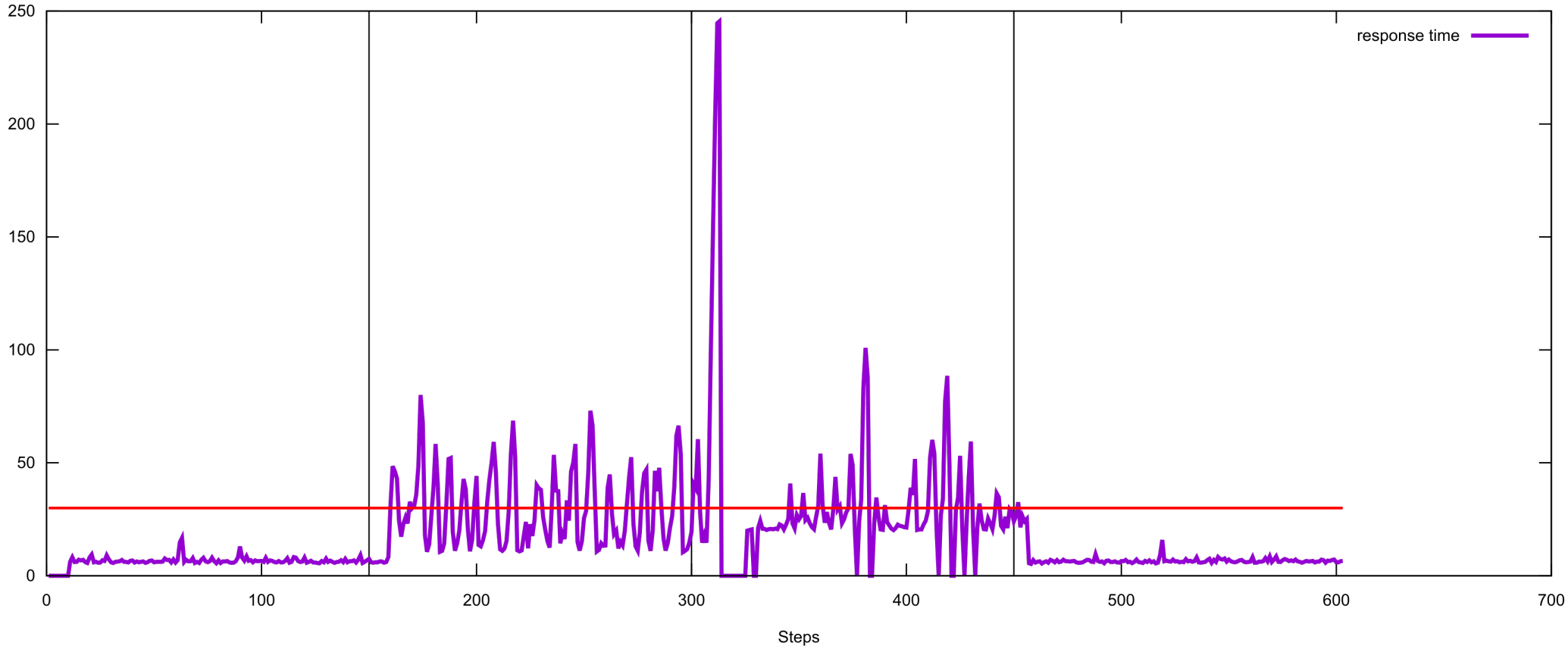
```
K   = 18.0016
T   = 4.16737
tau = 9.46876

magic_number1 = p = 0.0022393268359234 45
magic_number2 = i = 0.0077013107873548 67
```

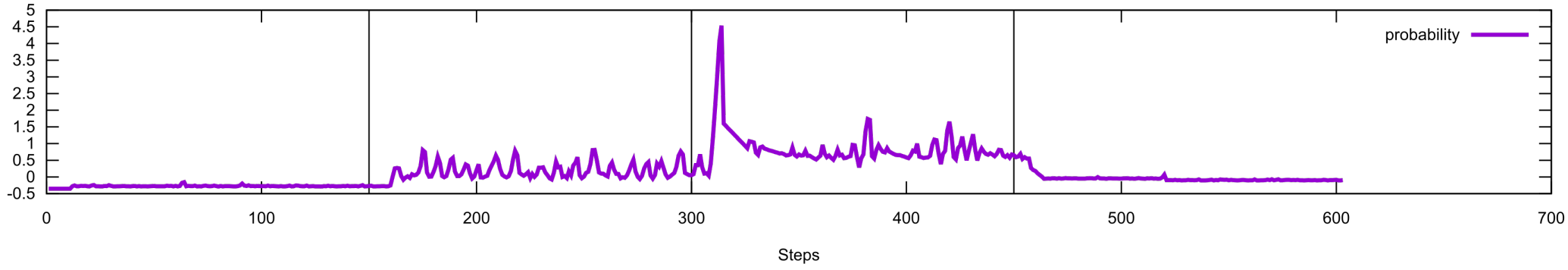
Proportional and integral control



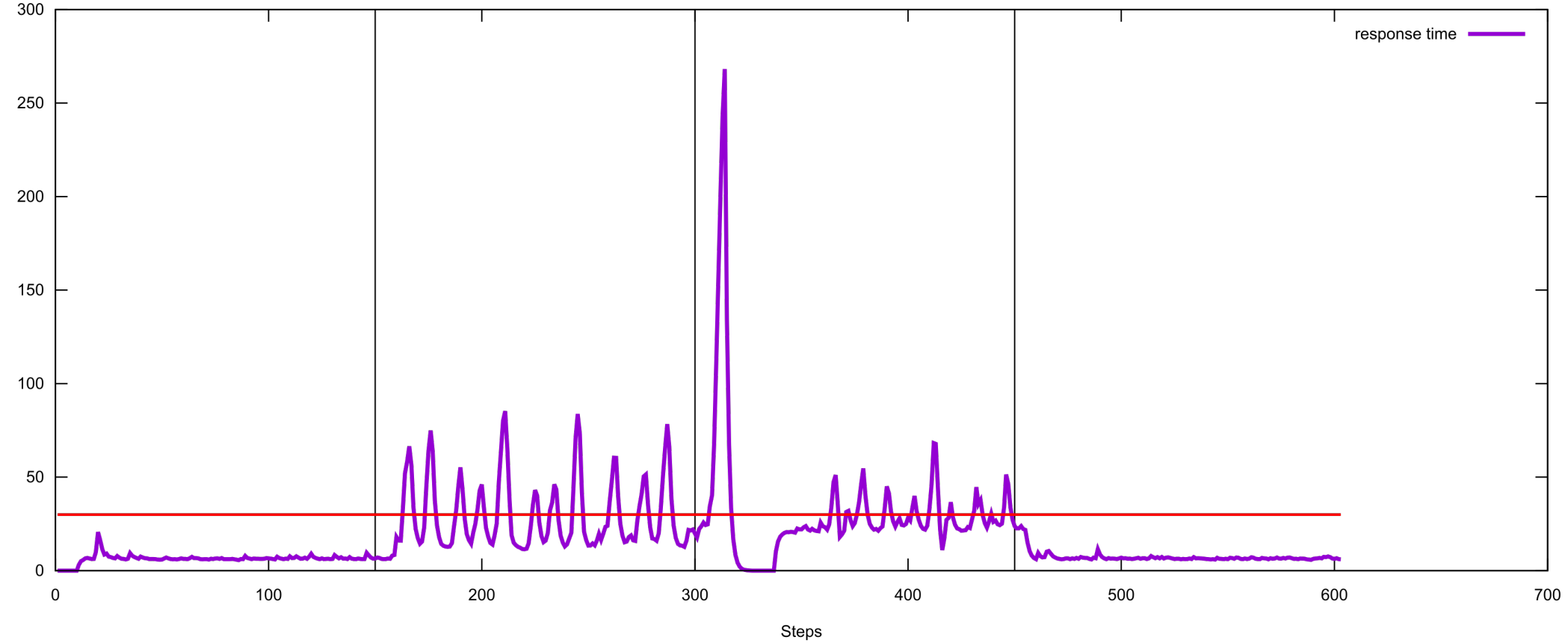


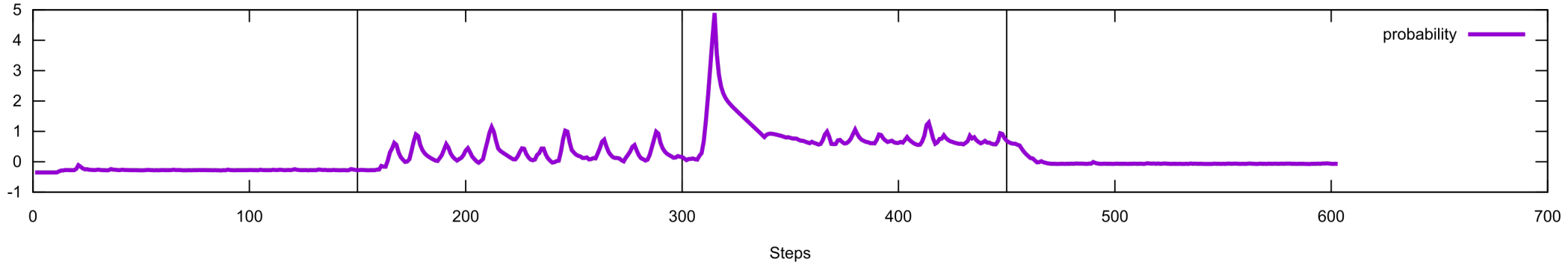
Proportional and integral control, with conditional integration



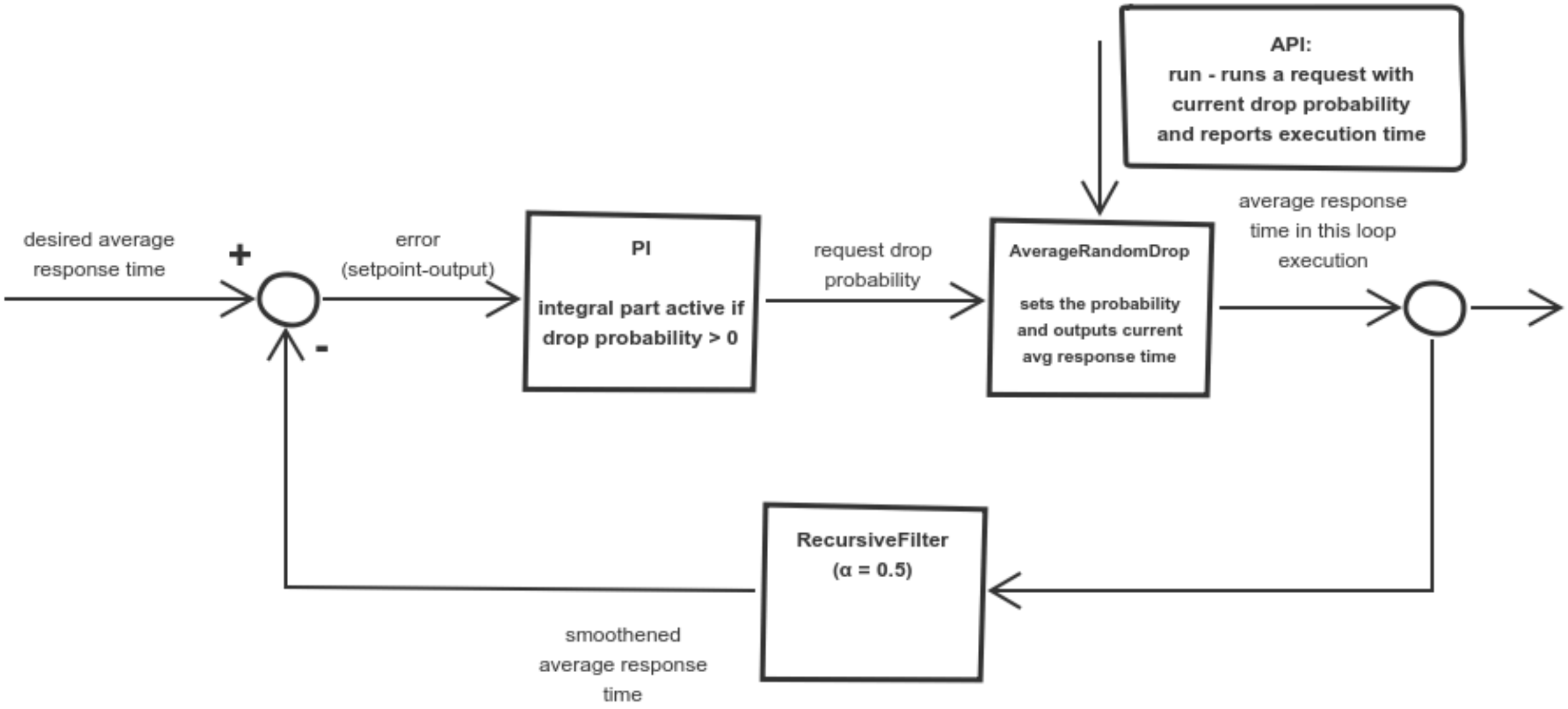


Proportional and integral control, with smoothing

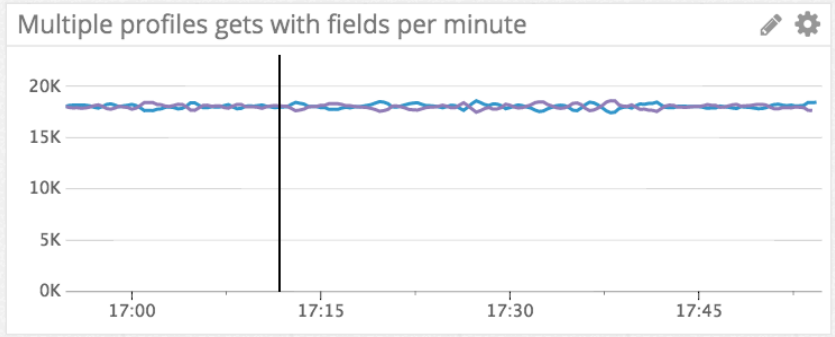


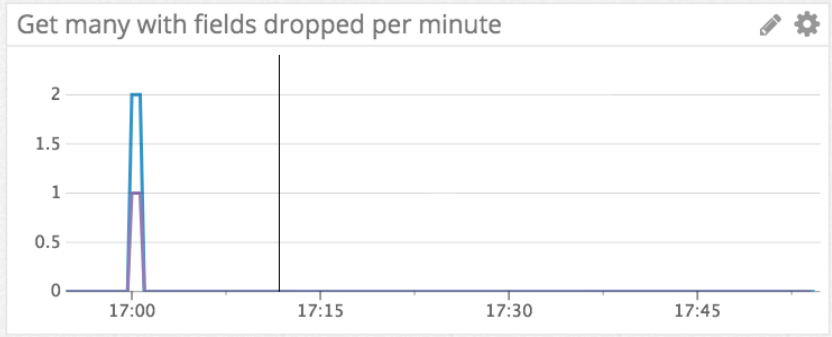


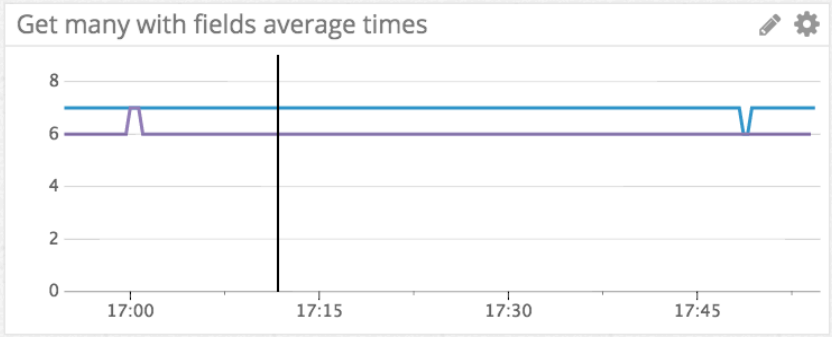
Final Loop


API:
run - runs a request with
current drop probability
and reports execution time
desired average
response time
+
-
error
(setpoint-output)
PI
integral part active if
drop probability > 0
request drop
probability
AverageRandomDrop
sets the probability
and outputs current
avg response time
average response
time in this loop
execution
RecursiveFilter
(α = 0.5)
smoothened
average response
time

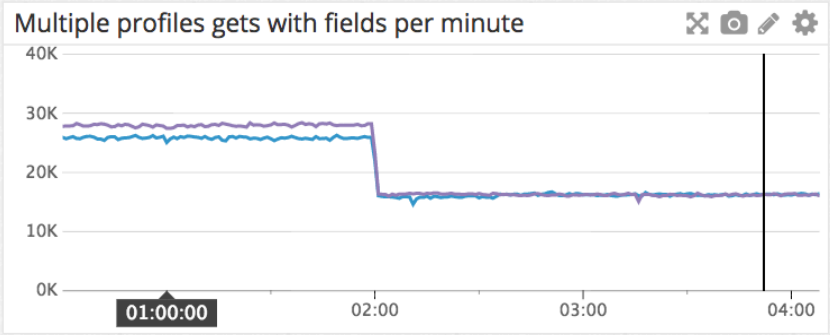
Simulation results #1 - steady state

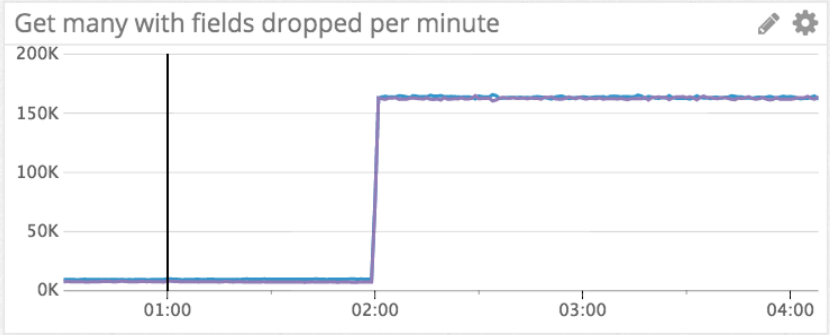


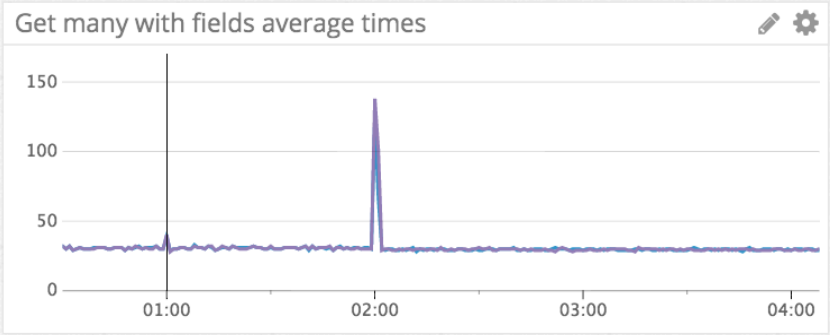




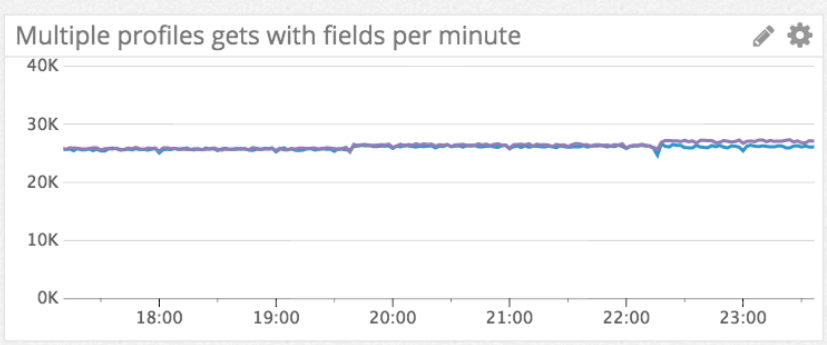
Simulation results #2 - load increase

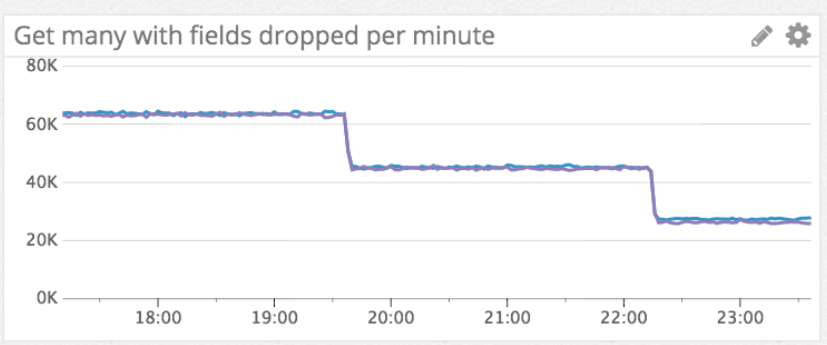


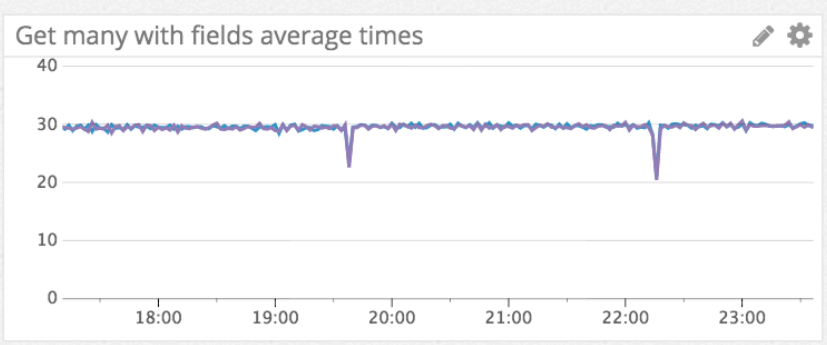




Simulation results #3 - load decrease

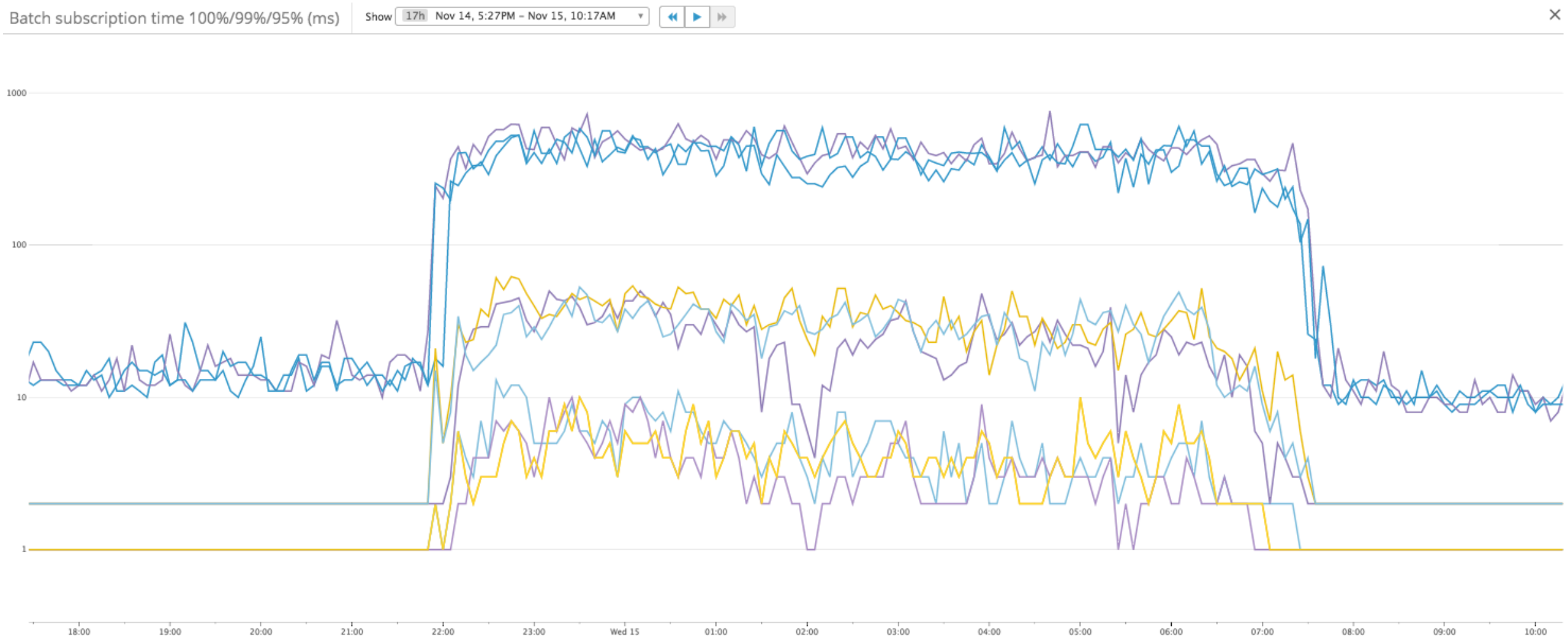




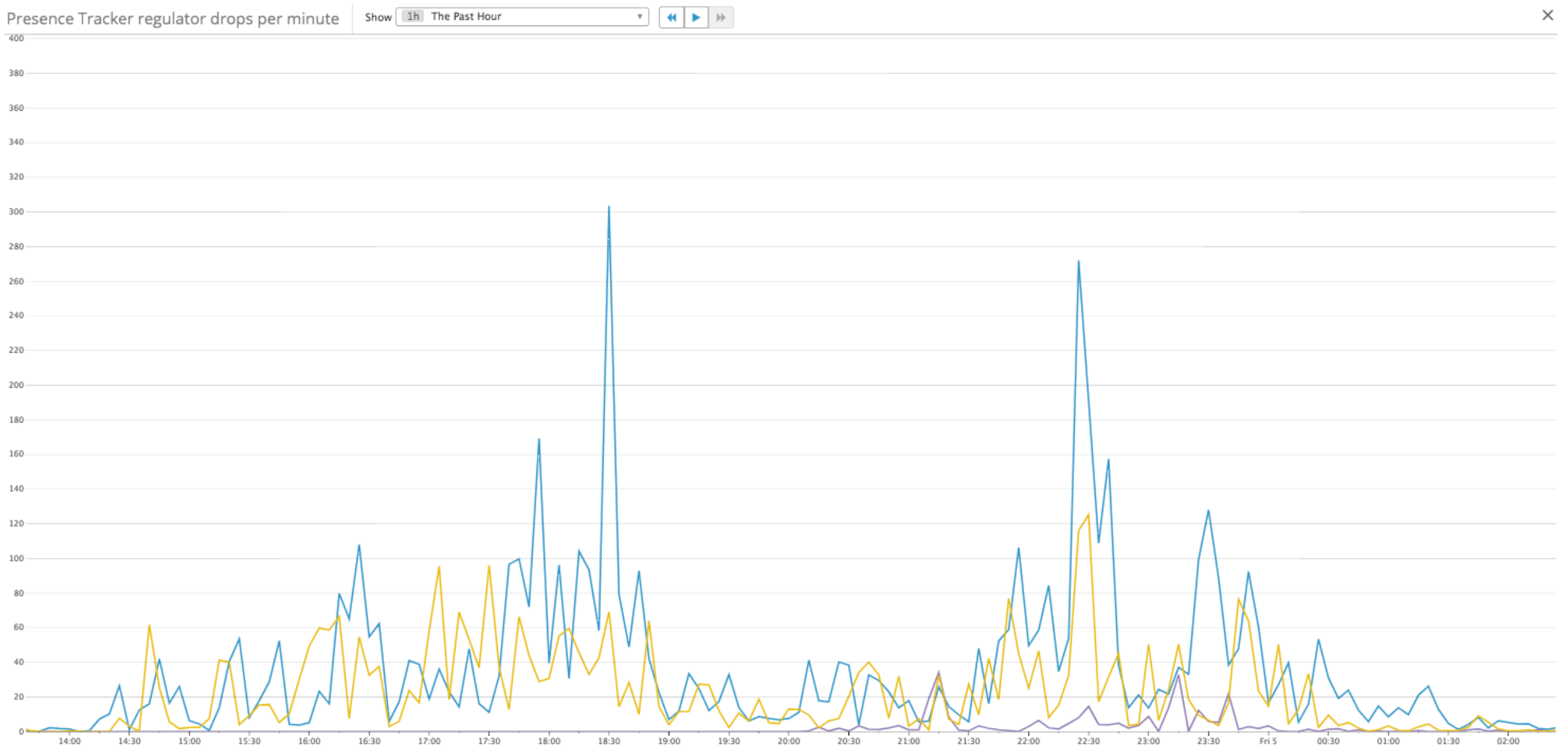


Real world examples

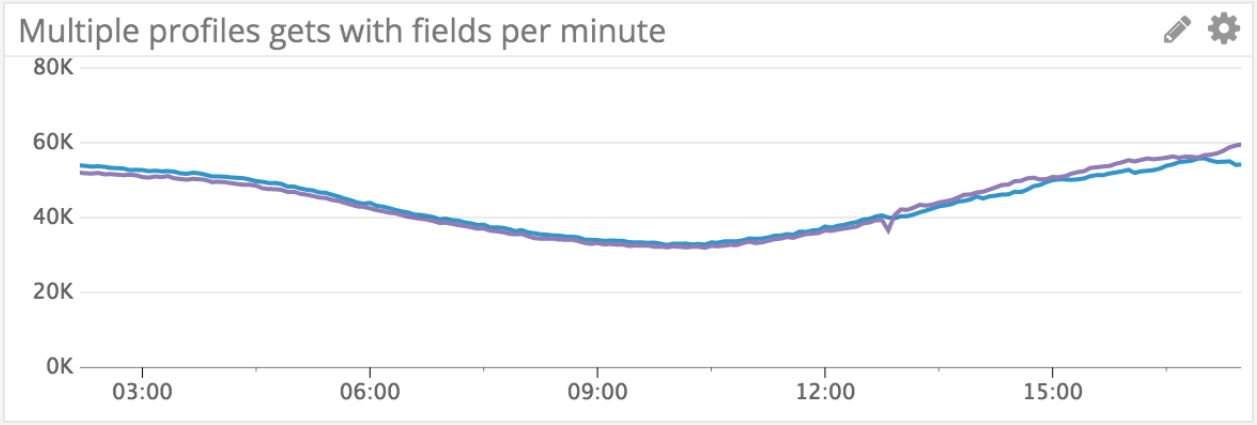
Real world example #1

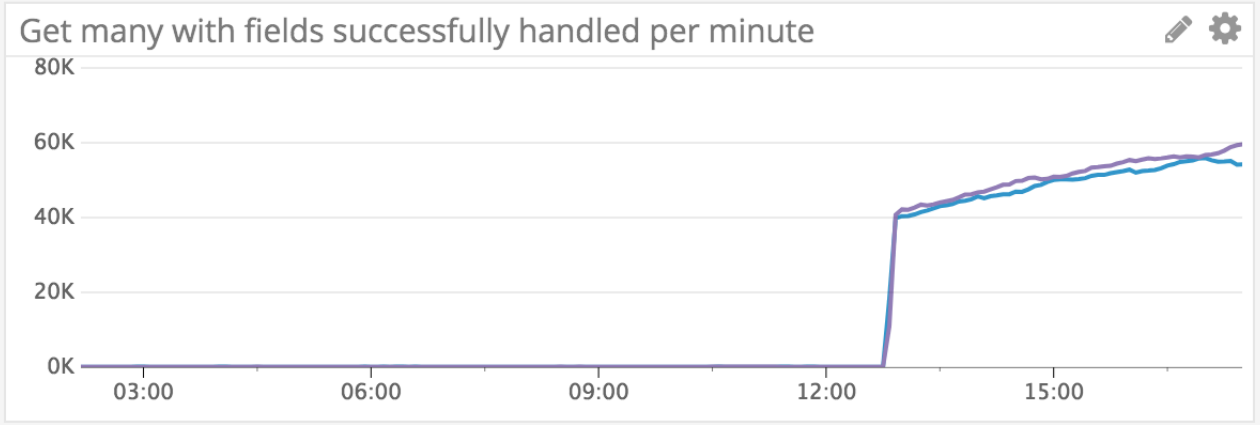


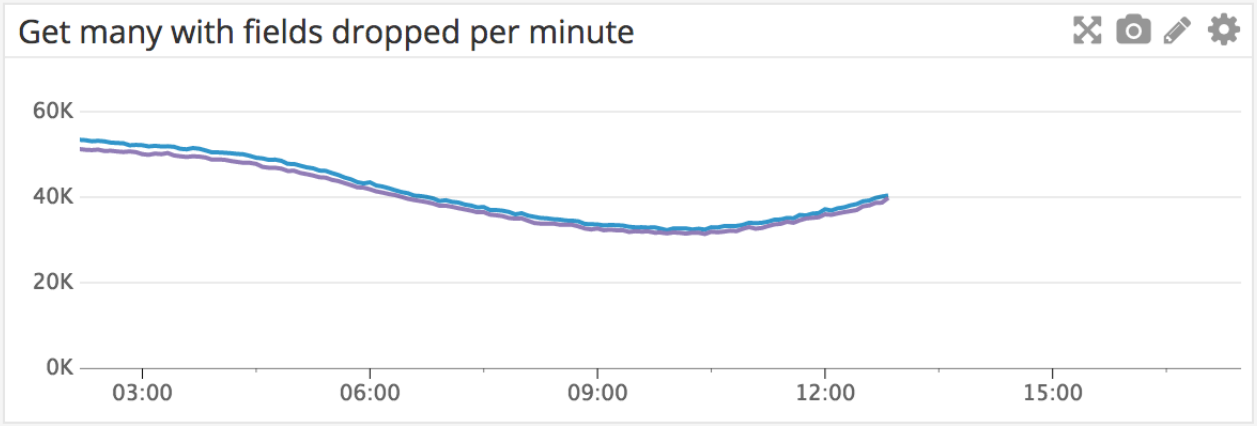
Real world example #1

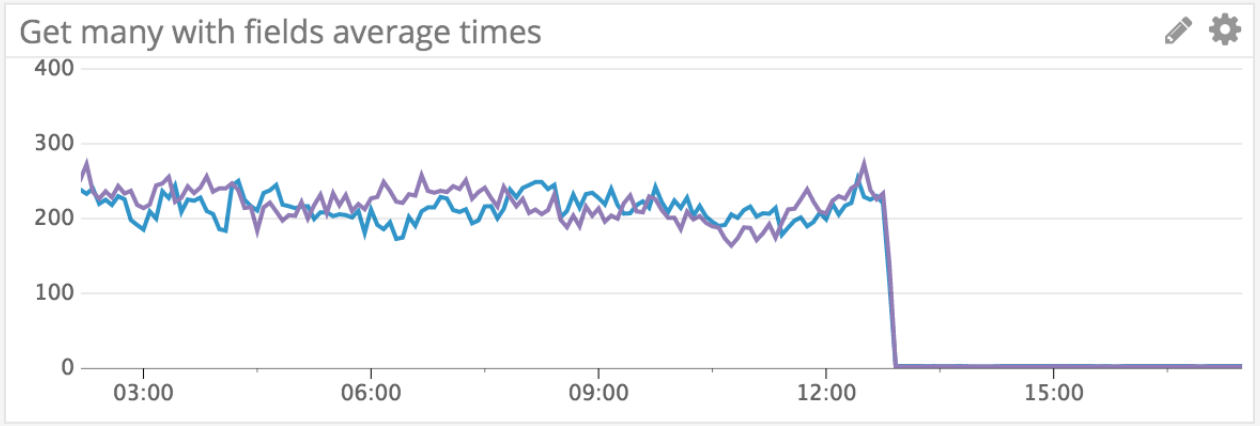


Real world example #2









What else?

- Use this approach in all the backend services, in all points of uncertainty.
- Use it in the client app, for better UX, by providing more lightweight content for low-bandwidth networks.

References

- *Feedback Control for Computer Systems* by Phillip K. Janert
- *Control System Lectures* by Brian Douglas
- *Queues don't fix overload* by Fred Hebert
- *Handling overload* by Fred Hebert
- https://github.com/fishcakez/sbroker
- https://github.com/ferd/dispcount
- https://github.com/uwiger/jobs
- *Surfing on Lava*

Questions?