Control Loops:

How to make &

check

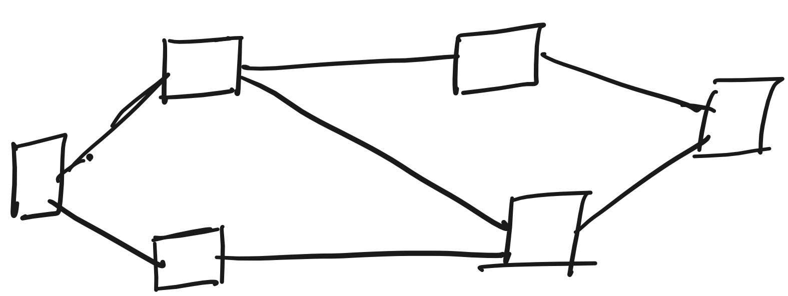
Misc.

→ Some observations from the midterm

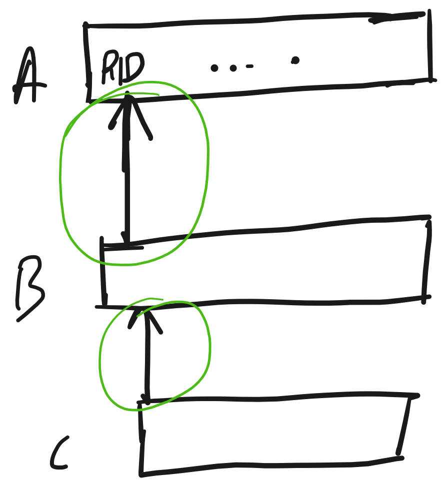
- RPC message size vs CPT

RPC message ↓ Local

CPT ↓ ¬Local



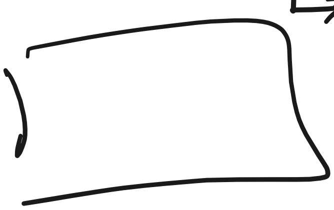
- SPANS vs. LOGS



A RID, T_0, . . .
B RID, T_1, . . .
C RID, T_2, . . .

TODAY: CONTROL LOOPS

- What are they?
  - ↳ Things that observe



INPUTS:
- cluster state
- request rate
- workloads ← where requests are coming from
- ...

↳ take action to meet some objective
- ↳ Min SLO violations
- → Max tput
- → Min resource waste

1 μs
20 μs late
300 ns

Acceptable latency: X
% of requests with latency > X

→ . . .

An idea that predates computers by several decades

How to steer torpedoes

Ships

. . .

You have already seen one control loop:

DQBarge

But: we did not talk about the design there but instead let model do heavy lifting.

Today a deeper look at the actual loops.

What does this have to do with tracing

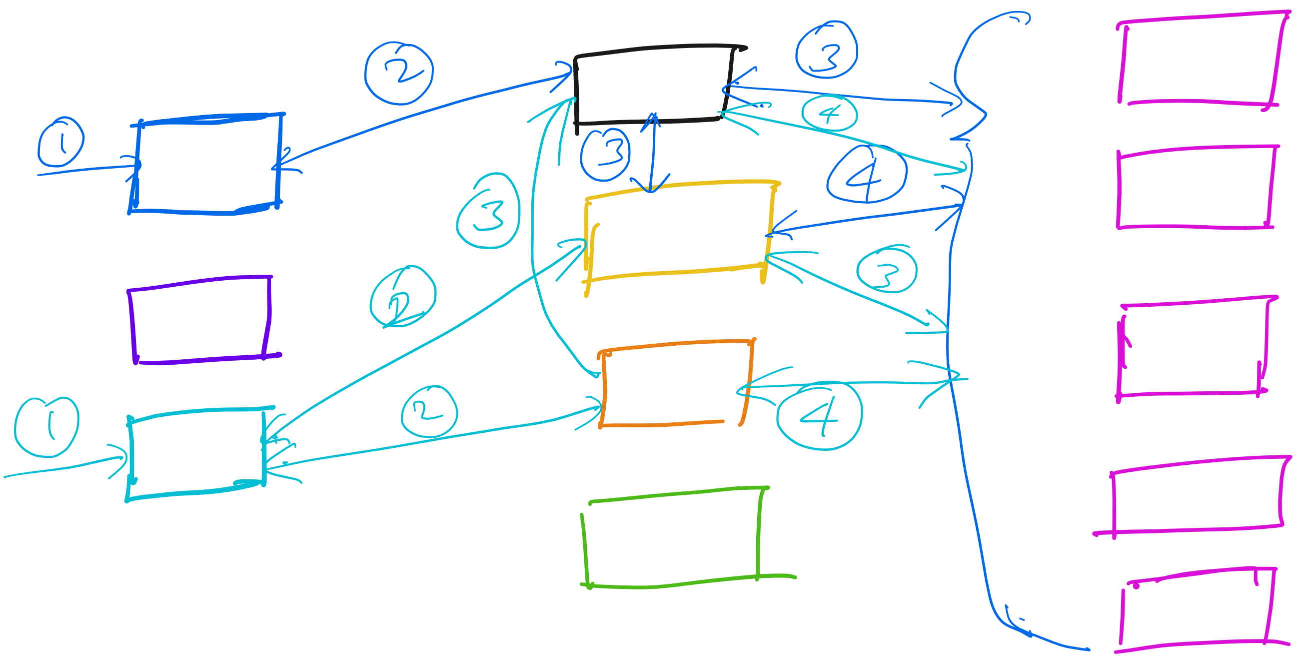
Common theme of all three papers

↳ Local decisions require global information

→ { What global information is required?

How to gather?

Scheduling, Admission Control, and Request Routing



- A single service can have a lot of different requests to process
  - ↳ From different upstream services
    - workflows (in the sense we saw earlier)
    - users
    - ...

With different
- resource requirements
- execution times
- SLOs
- Utility

Scheduling: Decide order in which

are processed

pending requests are processe

- Admission control

From DQBarge: Sometime best not to issue a request that is unlikely to complete in time.

More generally: Need to limit number of requests in the system for stability

Why?

Admission control: How many requests to "allow into the system"?

↳ How to disallow requests?

→ Return an error.

→ Silently drop them.

→ Make them not issue a request.

- Request routing

Many services are replicated

↳ If a request is going to be admitted, which replica should process it.

[Chosen replica then schedules the request]

For all three: We should ask what is desirable

↳ Across the cluster } UTILITY FUNCTION
→ Within a service }

POSSIBILITIES

↳ Fairness b/w - workflows,
- upstream services
- user
- ...

Max-Min fairness

→ First come first serve

→ Priority based

→ Minimize violated SLO

→ ...

Wisp & C3 focus on minimizing violated SLOs.

Why?

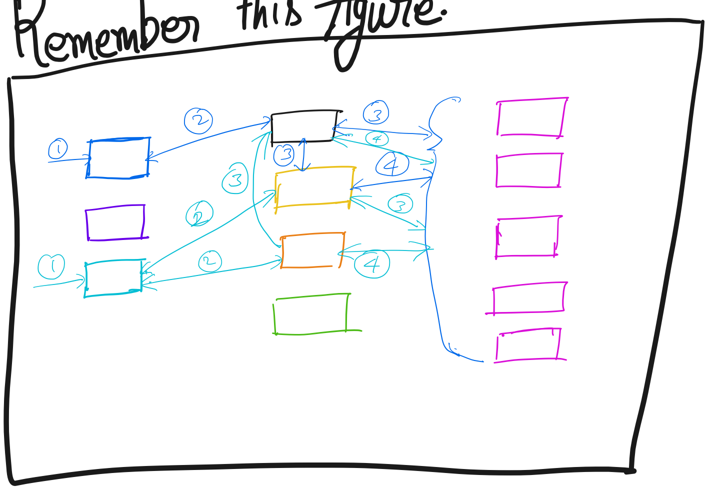
- Common concern in multi-tenant environments for control loops managed by operators that manage tenant behavior

  ↳ Sort of reflect economic incentives

- Less common in
  - academic settings
  - networking
  - operating systems

  } Fairness or some other metric is more common

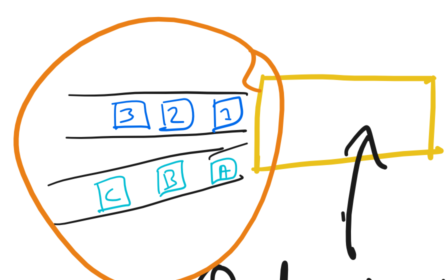
• Remember this figure.



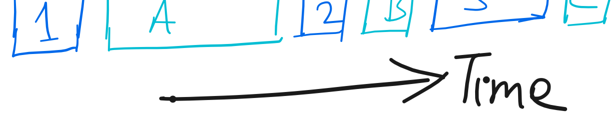
Many options

- FCFS
- Round Robin

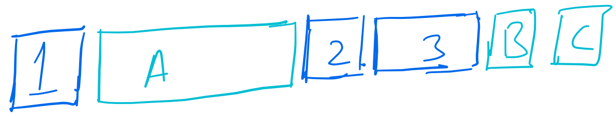
o Request scheduling



Order in which requests are processed.

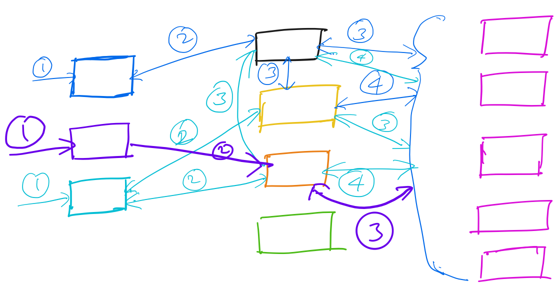


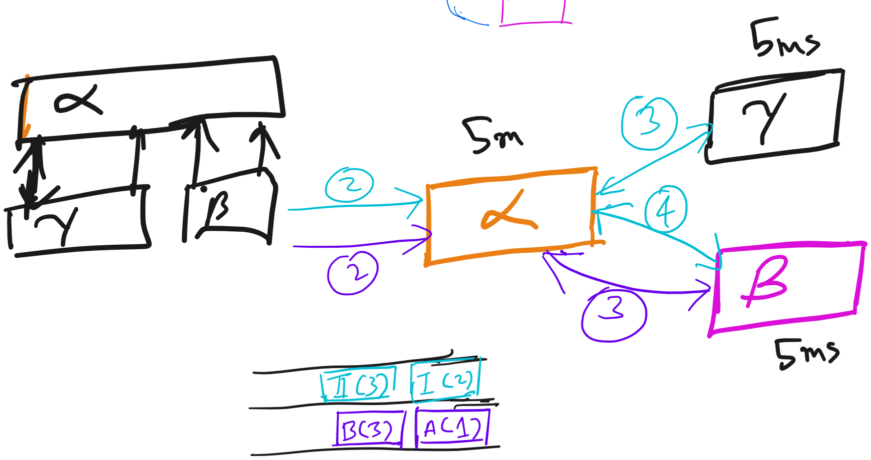
- Processing Time Fairness

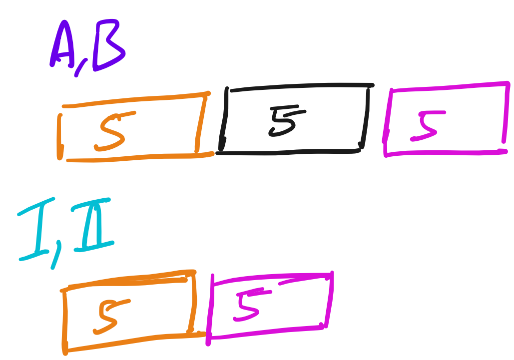


- ooo

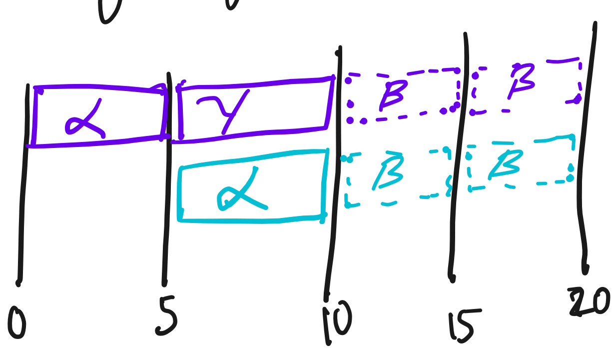
Looking at one service might not be good for minimizing SLO violations



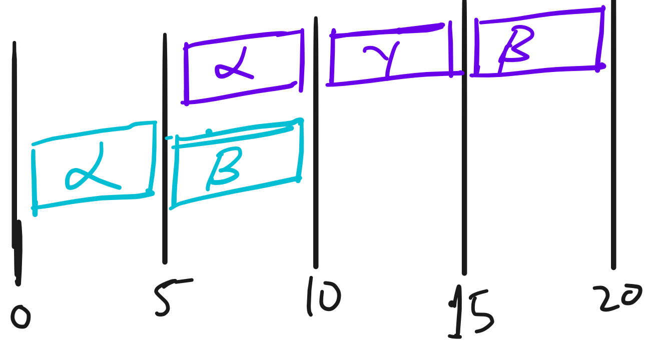




If A goes first (assuming nothing else is in the system



If I goes first,



Observe: Choice at α determines

↳ Overall completion time for I

↳ But has no (or smaller) impact on A's completion time.

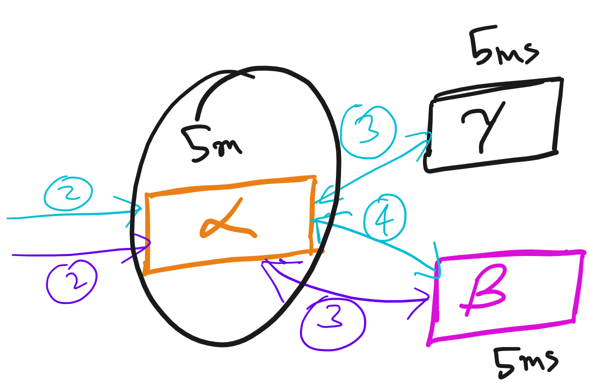
But delayed I could have violated SLO?

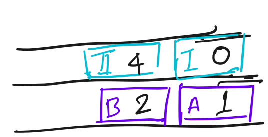
(I know this example sounds contrived, but...)

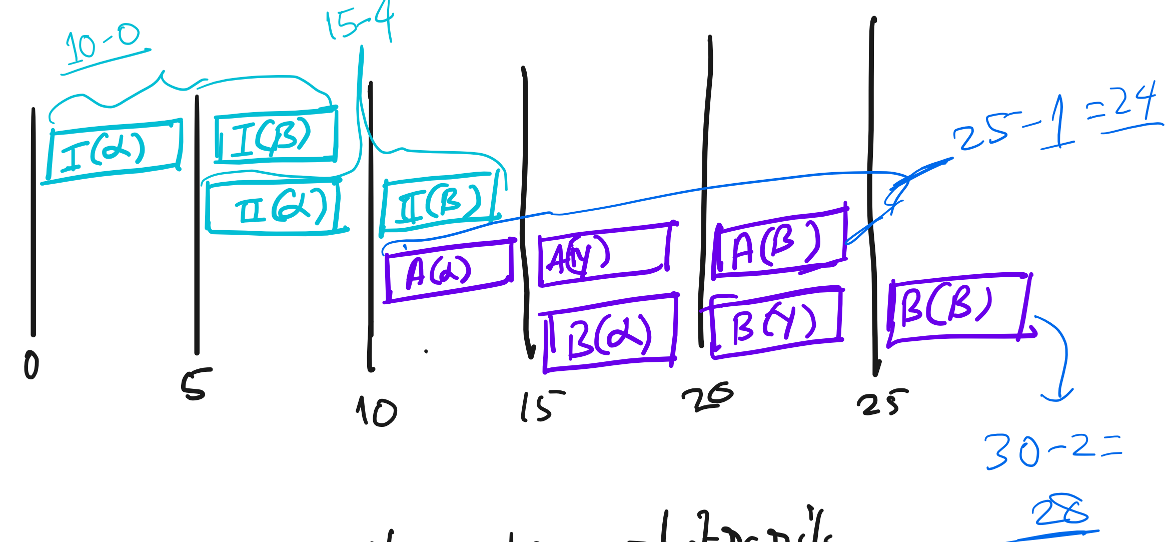
One option: Strict priority: More stringent SLO first

Lower latency first.

Problem:







Can delay low priority jobs arbitrarily.

What is desirable

↳ Account for time left when scheduling.

(to SLO violation)

But, at a service, time until SLO violation is likely depends on

↳ Time spent upstream

→ Wait time at service } Local

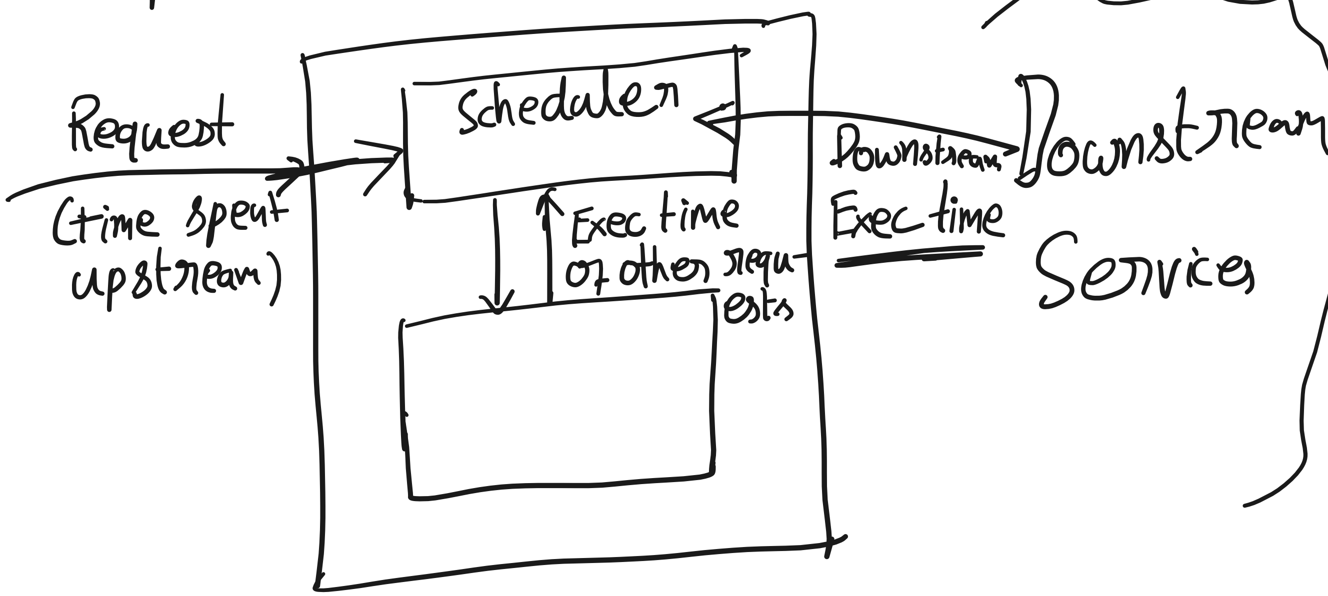
→ ~~Execution time at service~~

→ E[Wait time downstream]

→ E[Execution time downstream]

→ E[Execution

Control loop



How to estimate? EWMA

$$z_i = \lambda x_i + (1-\lambda) z_{i-1}$$

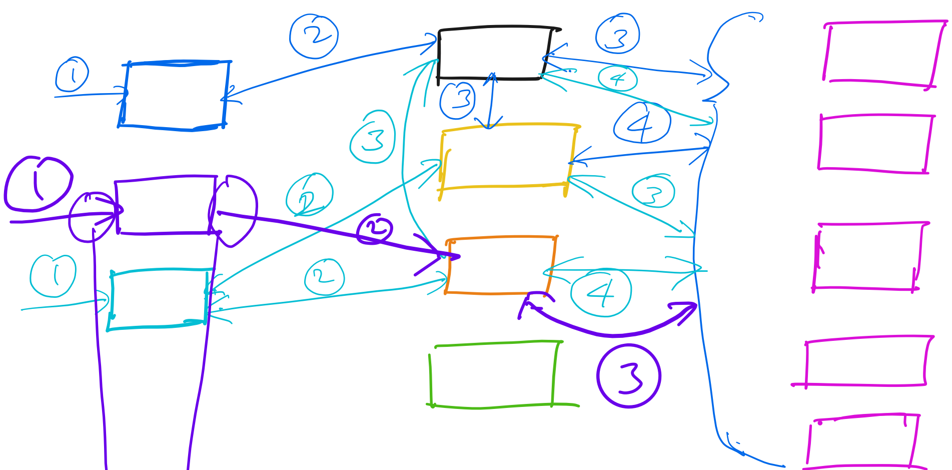
↓ Contribution from old measurements drops exponentially!!

Estimates useful for many policies

- → EDF (Earliest deadline first)
- → SRTF/ SRPT (Shortest remaining proc. time first)
- → SJF

But: Policy alone cannot prevent SLO violations!

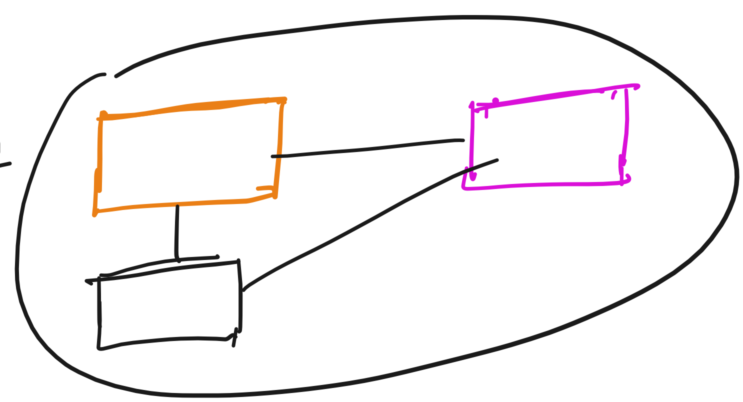
Admission Control



How many requests/second (to avoid violating SLO) ?

How many RPS to avoid SLO violations for ● x ● ?

Capacity of



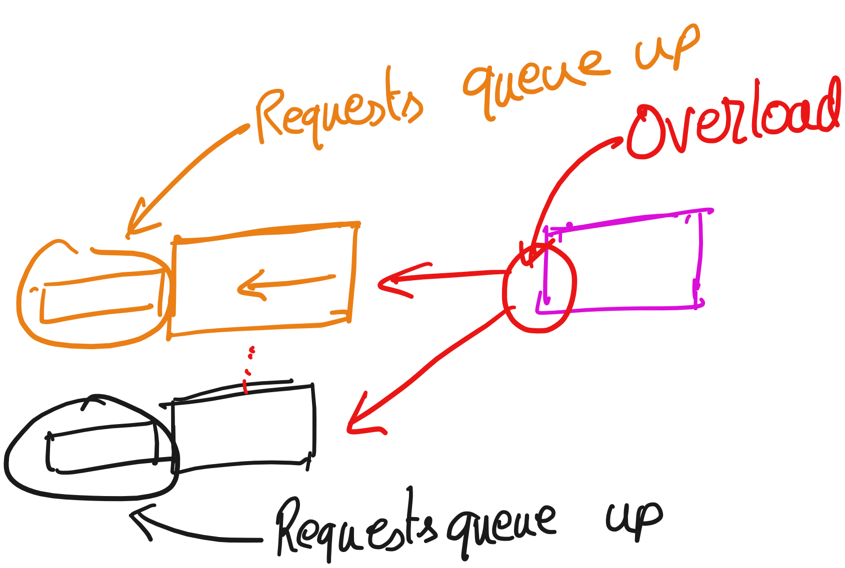
?

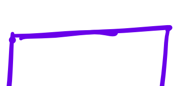
Needed by  on upstream.

But how to estimate?
↳ Changes depending on workload
Depends on 

↳ Might also vary over time
↳ S/w changes 
↳ automated updates
↳ ...

Control loop: Estimate (guess) capacity. How?
Assumption: Back pressure



 Can use queue length at  to estimate

capacity.

⇒ Limit RPS to keep queuing within a small range?

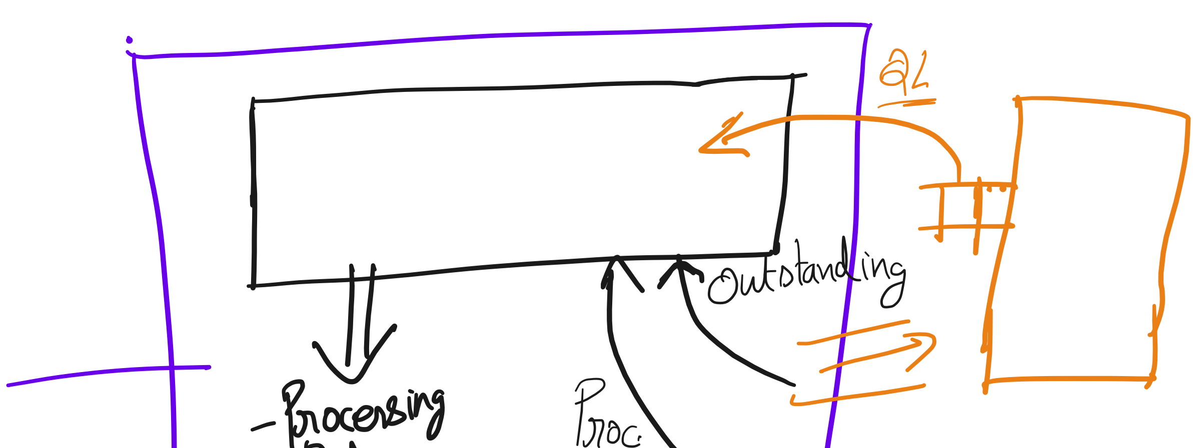
Problem: Both [purple box] & [cyan box] contribute to queue at [orange box], as does downstream overload.

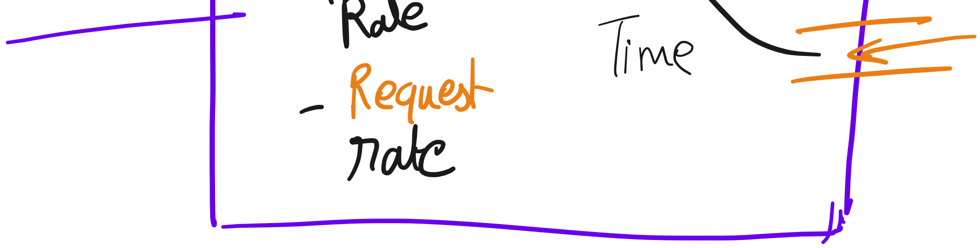
→ For [purple box] & [cyan box] need to split capacity b/w them

→ For downstream overload: need to reduce rate

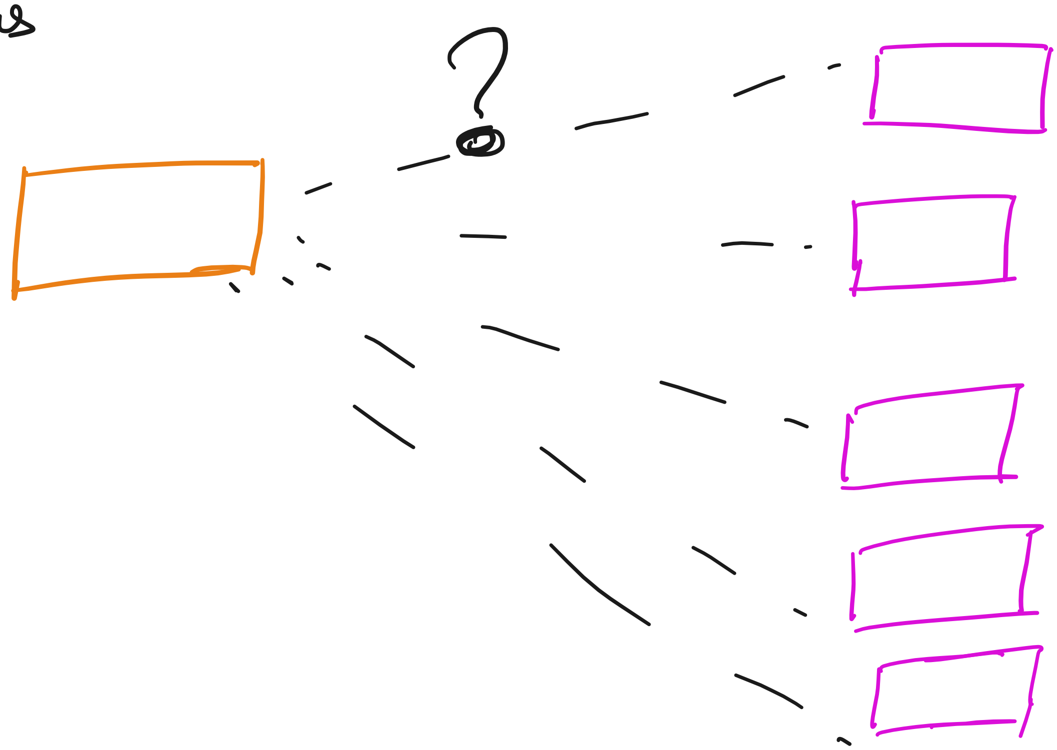
→ Track # of requests from self

→ Track response time for each request



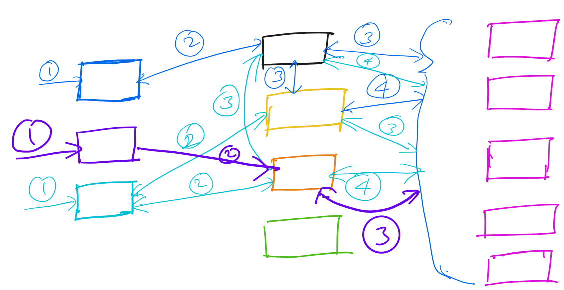


But what if we can select among many different replicas



Which  should  go to?

ⓐ Shortest queue?



→ All of  might make same decision.
⟹ Overload?

→ No coordination!

↳ Queue length ≠ response time.

Why?

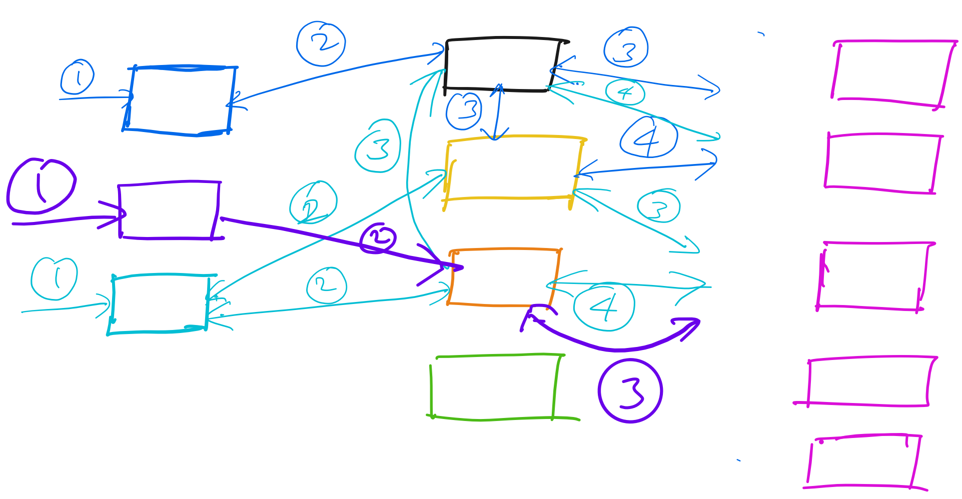
(b) Fastest?

↳ How to predict?

(c) C3??

"Based on historic information?"

Why would this possibly work in our case?



The case of symmetric decisions.

The case with different request lengths.

Rate at which a control loop can adapt?

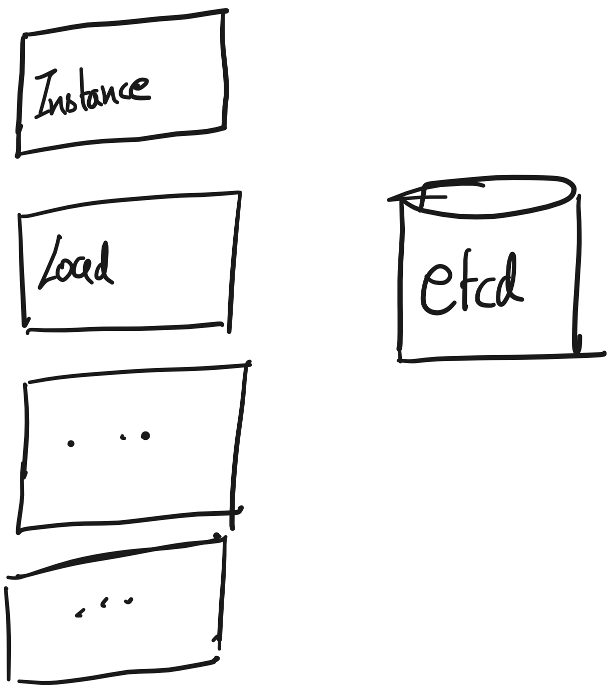
Testing & debugging control loop:

Challenges

- Defining correct vs incorrect is hard
- Hard to figure out sequence of feedback to use to drive the tests
- Hard to associate action with a particular

input.

A.R.T targets a case where some of these problems can be assumed away



What this enables

- Mutation based input generation

- Associating (maybe) action & input

- Test oracles

Extending Beyond this setting