Summarizing

Everything

Poster Session

- Reminder: Next week, in class
  - ↳ Do not print your poster
    - ↳ Just set up your laptop, etc. to show it

→ Come up with how you would present what you did to someone in 5ish minutes

Suggestions

Goal: Communicate two things

- Question or problem you

considered

- What was interesting about
  - ↳ Your approach
  - → Your findings
  - → ...

Audience: Your classmates

Assume knowledge of things from class + general CS educatio

Do Not Assume knowledge of the proposal, etc. you posted.

→ These are just suggestions. Sometimes yo might be better off ignoring them & going a different way.

Final Exam

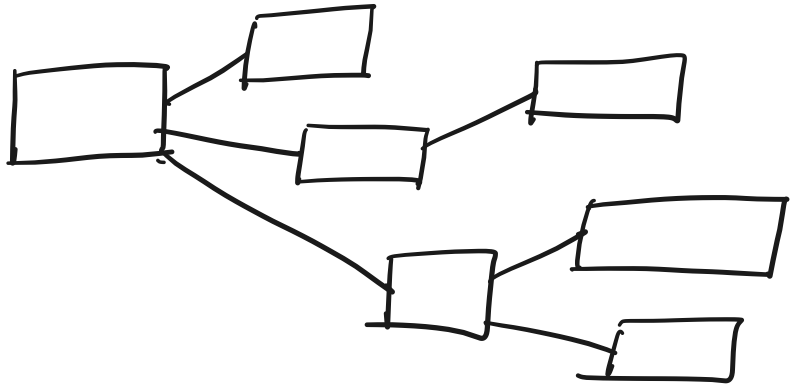
- Yes, we have one

May 11 : 6-7:50pm; This room

- Covers everything
- Similar in structure to midterm
- Also open book, etc.

Where we are

01 - Building & deploying applications made up of many services is common now



Some debate about how many services

Many
↳ More flexibility ?
→ Easier to get more people to work on app
→ ...

Fewer
↳ Easier to improve performance, reason about correctness

→ ...

But using >1 per-application has been norm forever

WS ↔ DB

WS ↔ File system

Adding more services is likely necessary now

- Caches/edge to get lower latency ↗ New apps
  ↳ reduce network cost
- Split functionality depending on h/w
  on which things execute
  ↳ Accelerators: GPU, ...

- ...

O2: We expect more from the applications we use.

→ Rely on Internet services for nearly all civic functions

→ Have several services where latency & tput matter

- Video conferencing
- Video streaming

- Gaming
- ...

Need to build more reliable applications

↳ Fewer failures

→ More consistent performance

→ ...

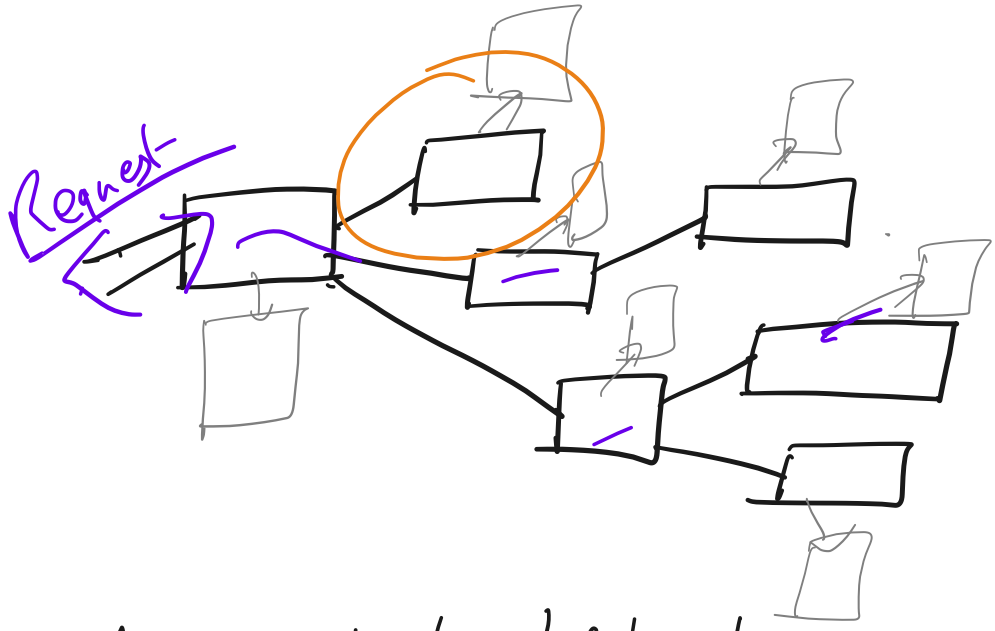
How?

- Build more reliable services/components
- Glue them more carefully?

03: Don't have a lot of tools to build reliable systems

- New languages & runtimes
- Better libraries
- Static verification

Therefore, rely on post-facto analysis



Standard approaches to post-facto analysis
- Logs
- Crash dumps.

Hard to know if problem on performance @ service led to problem @ app

⇓

Traces ← Add information to logs to enable correlating information.

This is the main takeaway

Logging in a distributed application

≡ log from each service
+
Causal information to link things together.

Everything else is just about balancing other concerns.

- How often should one log : Efficiency
- What to log : What type of analysis + use does one want

+ Failures:

+ Latency :

+ Resource Efficiency :

+ Throughput :

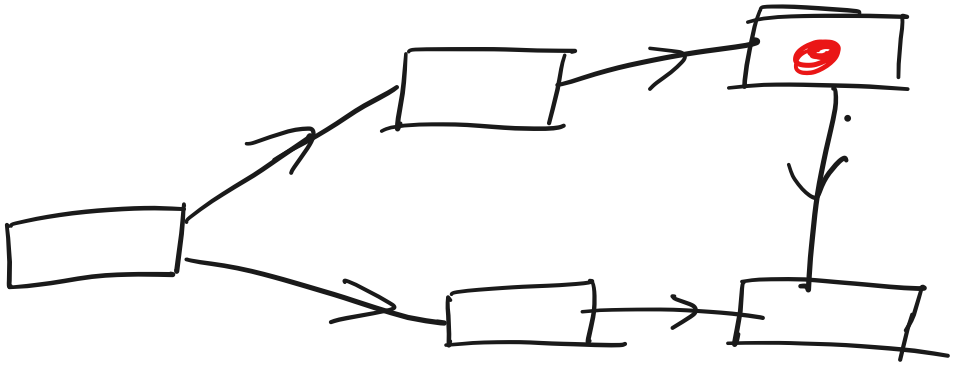
+ Scaling:

+ What algorithm to use:
(PGO)

+ ...

Nearly all of these have analogues in the single machine case.

ASIDE: What about crash dumps

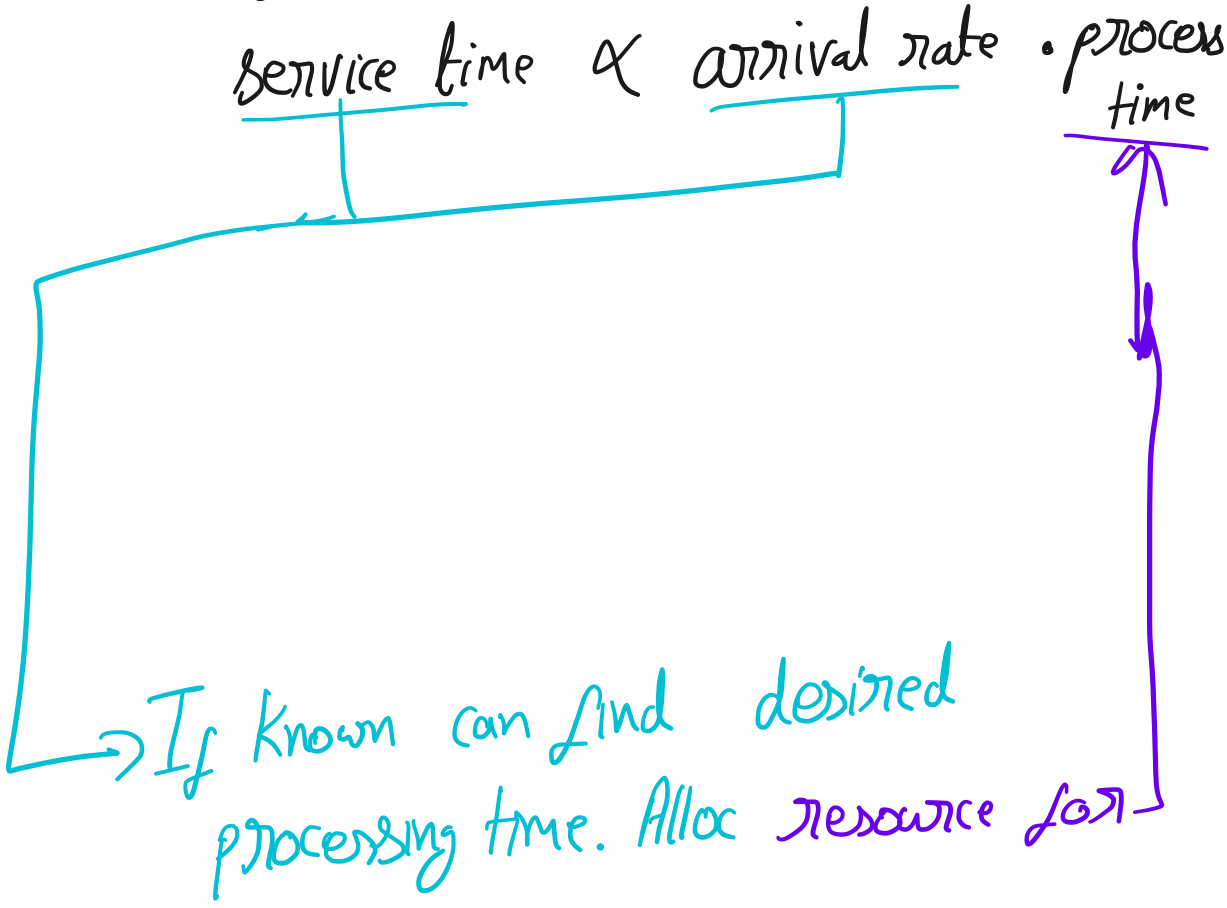


Other questions we considered

- How to achieve stable performance.

  + Generally

service time $\propto$ arrival rate $\cdot$ processing time



Problem

- Arrival rate varies. Might change by a lot
- Processing time is not constant, even w/ fixed resources.

But we want consistent performance

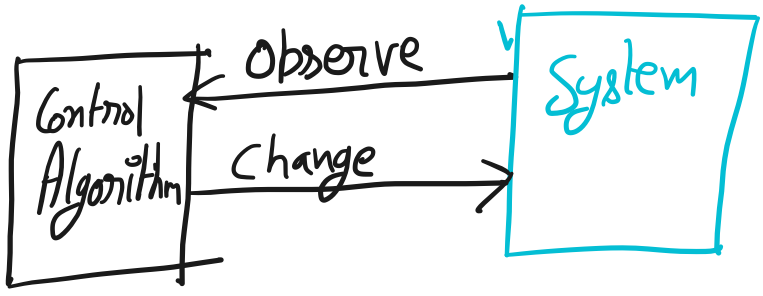
How?

① Adjust system parameters dynamically

② Drop requests to limit max arrival rate

Did not really look at this

① Control loops, RL, ..



ⓐ What to observe?

ⓑ What to change?

© By how much?

- How to test distributed applications

Problem: Large input space

↳ Request types

→ Other requests in the system

→ State of services

Unlikely to be able to cover all possible scenarios.

Ways to cover scenarios that are likely to matter?

```
if (x > 0) {
  ...
} else {
  ...
}
```

Aside: Why post-facto debugging is necessary?

Aside: The effectiveness of testing.

Where do we go from here

- We didn't really spend a lot of time on debugging.

When & where you might use any of this