Failures &

Fault Tolerance

Failures are common!

- You should find this statement absurd.
    - How many of you have had laptops fail on you?
    - How long between failures?
    - What was the failure?

And yet...

Jeff Dean: MTBF for server ~ 30 years

Jeff Dean's

still leads to 1 failure/day when using 10,000 servers.

Typical year at a datacenter 1-5% of disks fail [2009]

~1 PDU will fail each year ~ 500 machines down for 1-6 hours

~20 rack failures

~1000 server failures

Annecdotal "personal observation"

- Booting 15 EC2 instances almost always succeeds

- Booting 100 EC2 instances never succeeds (for me)
  - ↳ At least 1-2 instances stuck somewhere

- Reliability of 1 server $\neq$ Cluster reliability

How to address this problem?

- Design fault tolerant systems
  - ↳ But how? Many ways to skin this cat

- Rep. state machine
- 2 phase commit
- Primary backup replication
- Stateless computation w/external store
- ...

→ Each approach

↳ Makes different assumptions

→ Has different performance & efficiency tradeoffs

→ Application fault tolerance

↳ Inherits all assumptions & trade-offs

→ Hard to reason about & test correctness under failures.

→ Changes to applications/services can change assumptions & tradeoffs

Change the assumed failure model

ALL IN ALL MAKES APPLICATION ADMINISTRATION HARD!!

Netflix's thoughts on doing better

Mantra (Yahoo, Google, FB, Amazon, etc.) : ENGINEER SERVICES FOR FAILURES

↳ SERVICES MUST REMAIN AVAILABLE DESPITE SOME CLASS OF FAILURES

NETFLIX : DON'T WAIT FOR THINGS TO FAIL; PROACTIVELY INJECT FAILURES !!

Why? Easy to undo injected failures.

Find fault tolerance bugs sooner

CHAOS MONKEY.

Q: What to fail?

A: Choose at random.

Benefits : - Simple

- Assumes little or no information

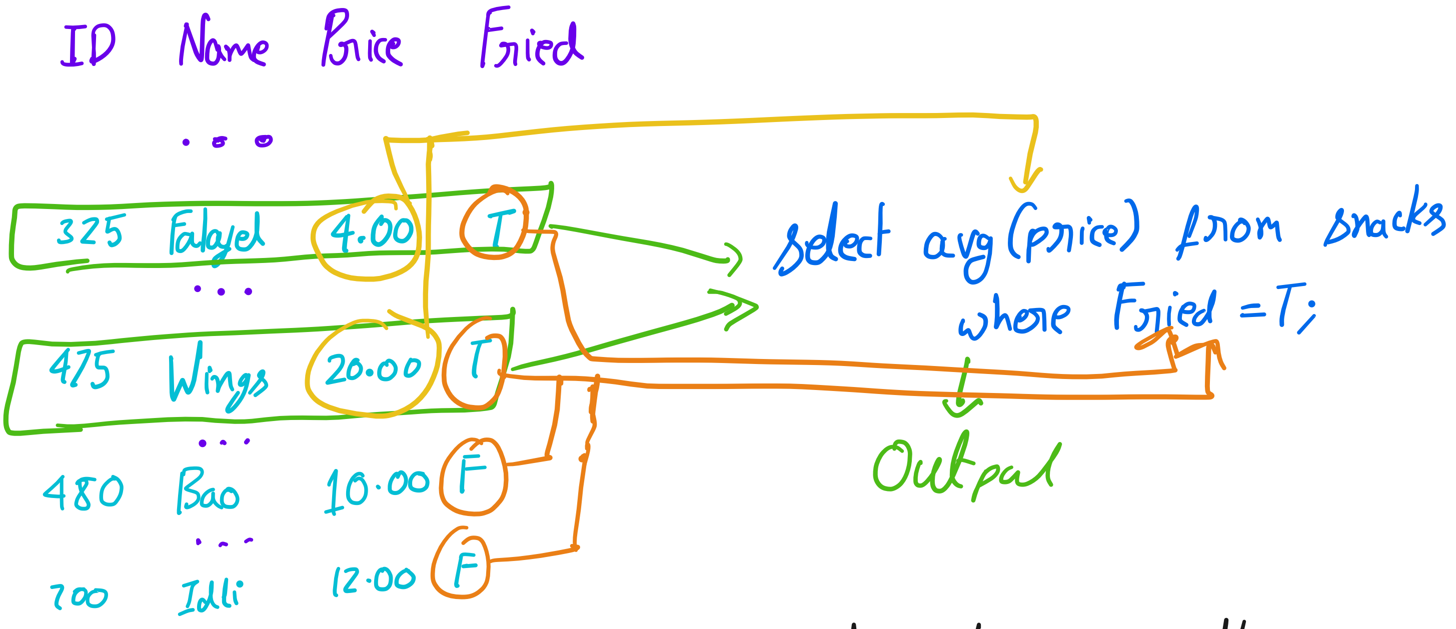
- Given enough time, explores many/most failures

Problems :- Inefficient!

Today's paper : Does more information help?

Lineage & Provenance

- What inputs & code paths contributed to an output.
  [Oldish idea from databases]



ID	Name	Price	Fried
	...		
325	Falafel	4.00	T
	...		
475	Wings	20.00	T
	...		
480	Bao	10.00	F
	...		
700	Idli	12.00	F

```
select avg(price) from snacks
       where Fried = T;
```

Useful to track what data contributes to a result.

Why?

ⓐ Incremental computation / View maintenance

ⓑ Reason about data privacy

ⓒ Debug program : why did we compute this value.

[illegible] /datalog

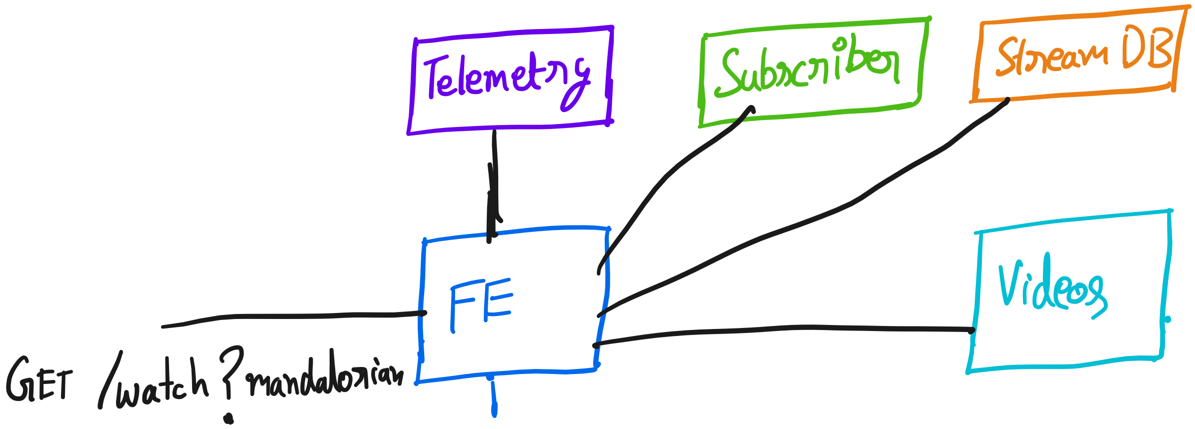
Not restricted to databases/

↳ Roughly similar to PFG.

But tracking is expensive.

What does this have to do with fault tolerance?

- Think of services/microservices as sources & sinks of data.



```
telemetry.record_get()
let subinfo = subscriber.get_session();
if (subinfo.allows(request)) {
   let d= streamdb.get_stream(request);
   telemetry.record-content(d);
   return make-stream(d, videos.get-content(d))
} else {
     telemetry.record-unauth(subinfo);
     return not_authorized(subinfo);
}
```

want to answer two questions:

During testing want to

- What service failures are more likely to impact availability : PRIORITIZE TESTING FOR THOSE.

```
telemetry.record_get()
let subinfo = subscriber.get_session();
if (subinfo.allows(request)) {
    let d = streamdb.get_stream(request);
    telemetry.record_content(d);
    return make_stream(d, videos.get_content(d))
} else {
    telemetry.record_unauth(subinfo);
    return not_authorized(subinfo);
}
```

Telemetry failures might slow down response, but unlikely to impact quality

Think back to DQ Borge etc.

- When to emulate failures?

```
①    telemetry.record_get()
①    let subinfo = subscriber.get_session();
Not effective
after?
     if (subinfo.allows(request)) {
         let d = streamdb.get_stream(request);
```

```
    let d = [illegible]
    telemetry.record-content(d);
    return make-stream(d, videos.get-content(d))
  } else {
②       telemetry.record-unauth(subinfo);
        return not-authorized(subinfo);
  }
```

So might give us enough information to limit how many tests to run

↳ No information + full coverage

↳ Fail all services at all possible program points

→ Lineage

↳ On average fewer (?)

Note: no better in worse case.

Aside: How to target failures?

Elephant in The Room: Analysis above required code analysis

- Static analysis
  - ↳ Very powerful
  - → Fun to think about
  - → Hard to apply/use in the wild
    - ↳ Need to make assumptions about language, execution model, ...
    - ↳ Esp. hard in polyglot

- Can we use lineage style fault injection without a system that records lineage.

Ans: Kind of? Use traces

FIRST THINGS FIRST: TRACES DO NOT CAPTURE LINEAGE

↳ No information about how results are used

```
telemetry.record_get()
let subinfo = subscriber.get_session();
if (subinfo.allows(request)) {
    let d= streamdb.get_stream(request);
    telemetry.record-content(d);
    return make-stream(d, videos.get-content(d))
} else {
    telemetry.record-unauth(subinfo);
    return not_authorized(subinfo);
}
```

→ Limited information about fallbacks

```
let si = subscriber.get-session()
if (si.is-error()) {
    si = subpolicy.default-sub(request);
}
...
```

Not visible until subscriber has failed

Indistinguishable

```
let si = subscriber.get-session()
if (date.today()= date(04, 01, 2023)){
    let osi= subpolicy.default-sub(request);
}
let use-subi = pick-stronger(si, osi);
...
```

But, still have enough information to

But still have enough information to

- Identify dependencies for a particular execution

  → We already track RPC calls & responses

  When calls are made.

Can add a little more to find fallbacks

→ See how trace changes when services are failed

→ Again, aside on how failures are injected matters here.

What is missing?

- Cannot distinguish benign & non-benign failures
- Cannot always identify failure dependencies

```
let si = subscriber.get-session().
if (si.is-error()) {
    si = subpolicy.default-sub(request);
}
if (!si.allowed(request)) {
        si = subpolicy.maybe_upgrade(request);
}
...
```

- Generally: some opportunities to improv efficiency but not as good as real lineage?

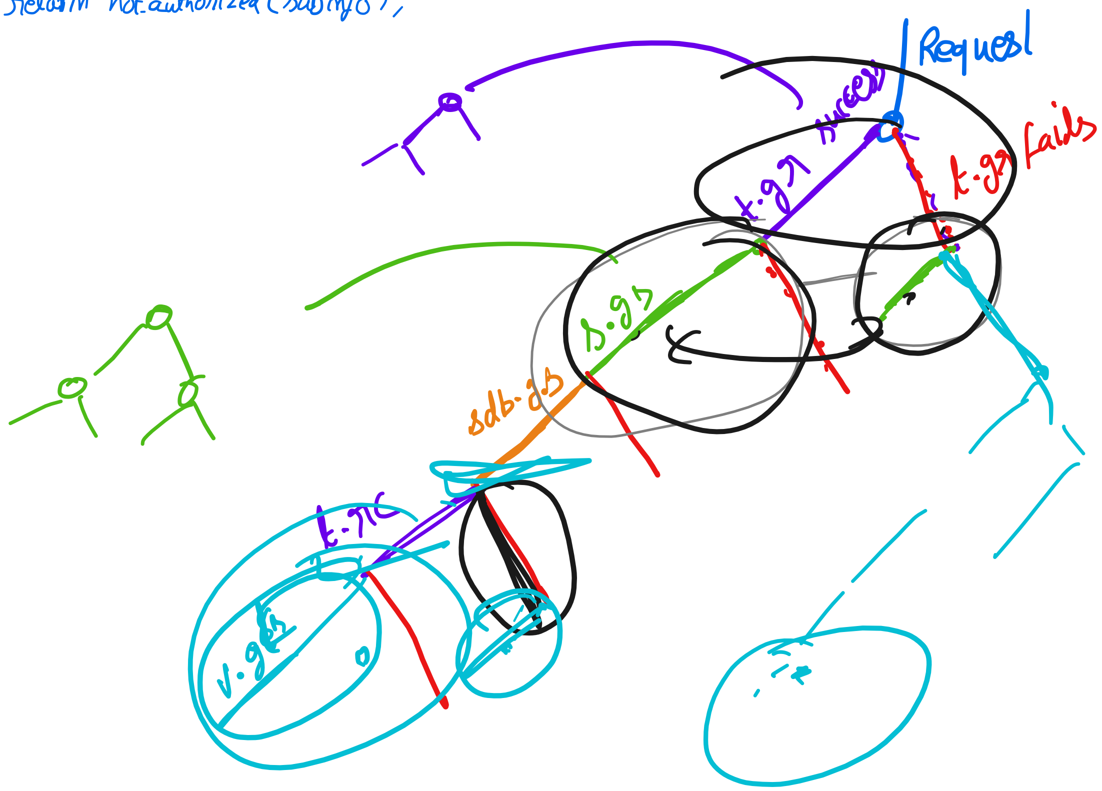
SLFI: How to actually use this information

- In testing rather than production
  ↳ Why?

- Broadly, this is a common approach for testing any scheduling like algorithm.

- Many ways to think about this: I like trees & forests

```
telemetry.record_get()
let subinfo = subscriber.get_session();
if (subinfo.allows(request)) {
  let d = streamdb.get_stream(request);
  telemetry.record_content(d);
  return make_stream(d, videos.get_content(d))
} else {
} else {
    telemetry.record-unauth(subinfo);
    return notauthorized(subinfo);
}
```



- Core questions: Order of exploration

When to prune

↳ Want to at least do it when executions are <u>equivalent</u>

But how to check?

When to stop?

Explore individual services or all of them?