Madaari
Ordering For The Monkeys

Agenda

- Distributed Systems and Chaos Engineering : State Of The Union
- Lineage Driven Fault Injection : A Brief Primer
- LDFI : Ordering Of Faults
- Bringing LDFI to the Enterprise
- Results
- Future Work

Industry + Academia = Win !!

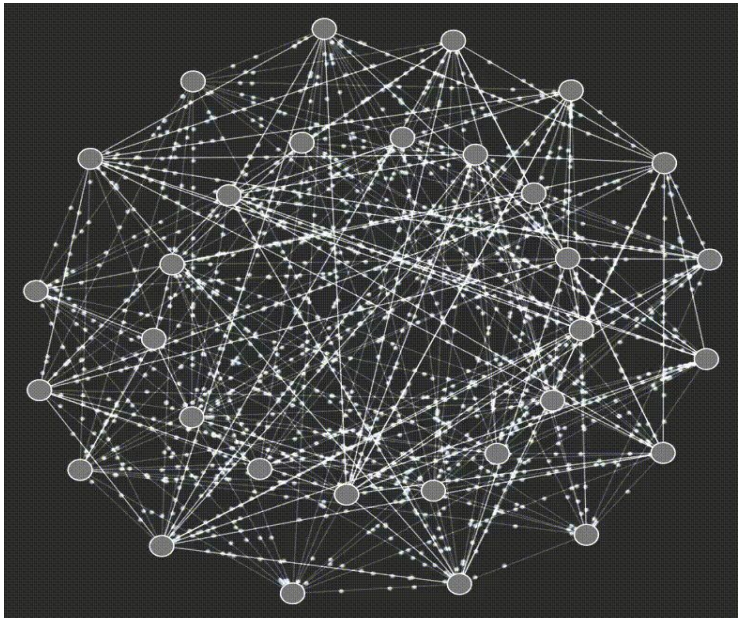
Joint work between eBay and Disorderly Labs

- Dr. Peter Alvaro (UCSC)
- Kamala Ramasubramanian (UCSC)
- eBay SRE Team

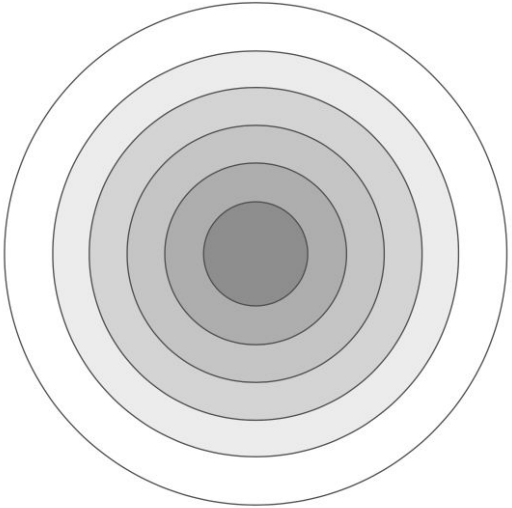
Madaari : a trainer who teaches a monkey to perform tricks

The Problem : Testing Distributed Systems

Microservices Death Star



Combinatorial Space of Failures



Consider 100 Services

Fault Search Space : 2^{100}

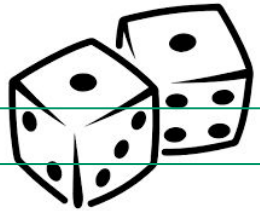
Fault Cardinality	Possible Faults
1	100
4	3 Million

Chaos Engineering : A Possible Solution

- Failure is inevitable, let’s fail in a controlled environment
- Proactively inject failure in your system to reveal weaknesses
- *Perturbation* and *observation* of large-scale systems

Chaos Engineering : A Brief Primer

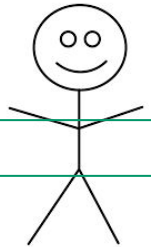
No Model Of The System



Random Fault Injection

Can’t quantify progress

A genius holds the mental model of the system



Guided Fault Injection

Doesn’t scale well !!

Lineage Driven Fault Injection aka LDFI

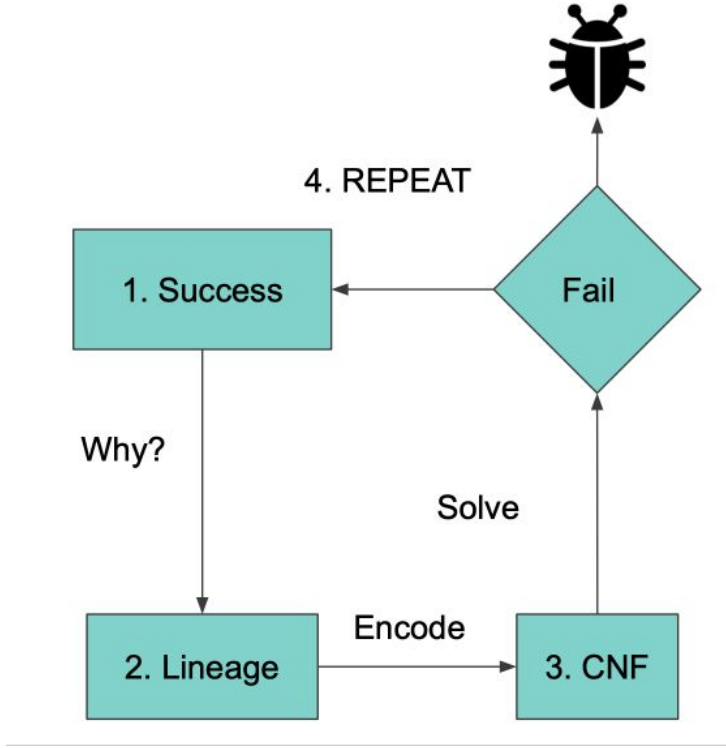
CLAIM : Fault Tolerance = Redundancy

- Use *explanations* of *successful* outcomes to search for faults that can drive the system into a bad state
- *Observing* successful executions enables LDFI to build a model of the *redundancy* of the system

LDFI : Building Blocks

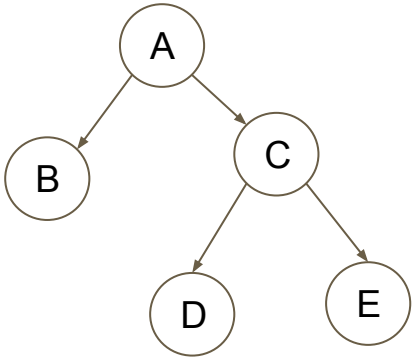
Recipe:

1. Start with a successful outcome. Work backwards.
2. Ask *why* it happened ? Ans. Lineage (Traces)
3. Convert lineage to a CNF formula and solve the decision problem (using a SAT solver)
4. Lather, rinse, repeat

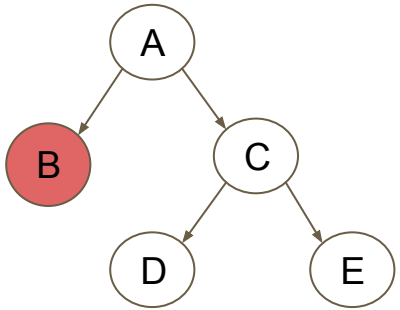


Encoding the Lineage

(A v B v C v D v E)



(A v C v D v E)

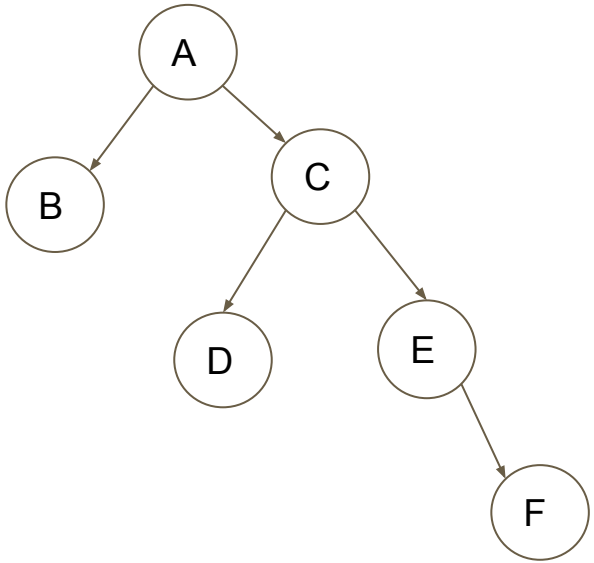


(A v B v C v D v E) ^ (A v C v D v E)

Ordering For Faults : Injecting Faults That Matter

- Drawbacks of existing approach
  - LDFI (using SAT) reduces the search space but the search space might still be still large
  - LDFI is a decision problem, solutions are returned in no particular order
- We want to order solutions to:
  - Find the most likely faults before users do!
  - Reduce the search space as much as possible

Ordering Faults : Injecting Faults That Matter

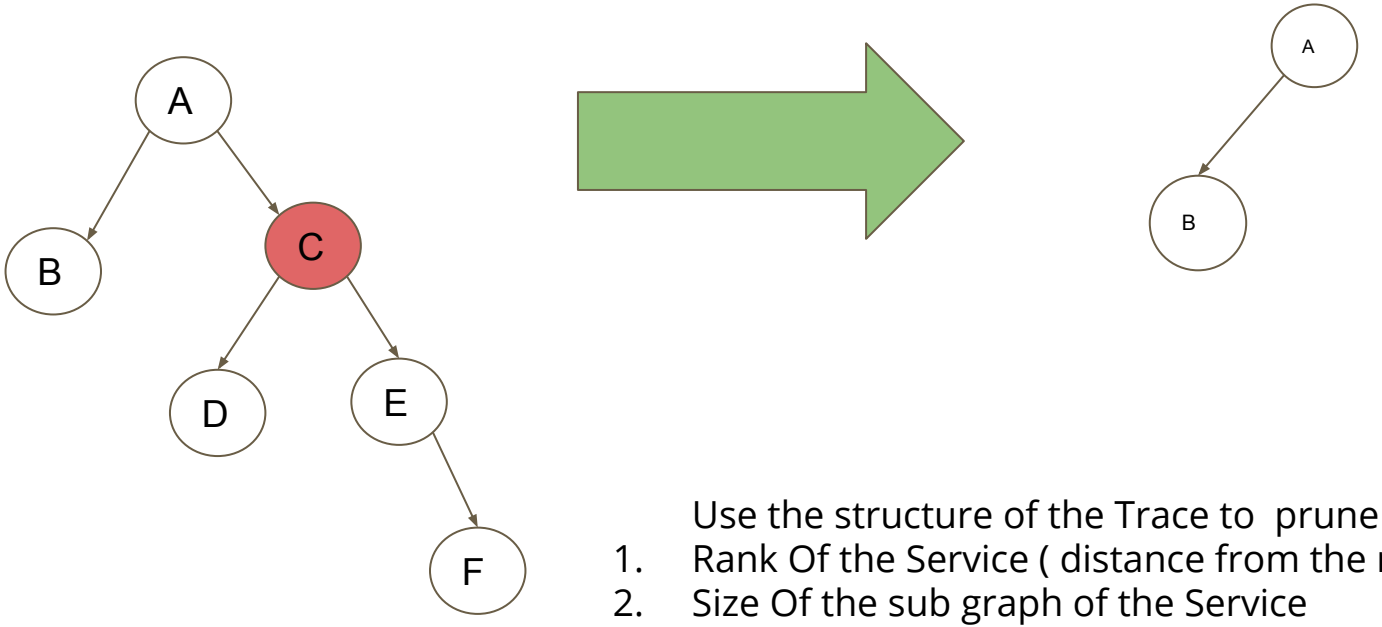


LDFI assumes all faults are equally likely, the reality differs !!

Intuition : Some faults are more likely than others; incident history usually backs this claim

We want to encode our *intuition* of failure in LDFI

Ordering Faults : Injecting Faults That Matter



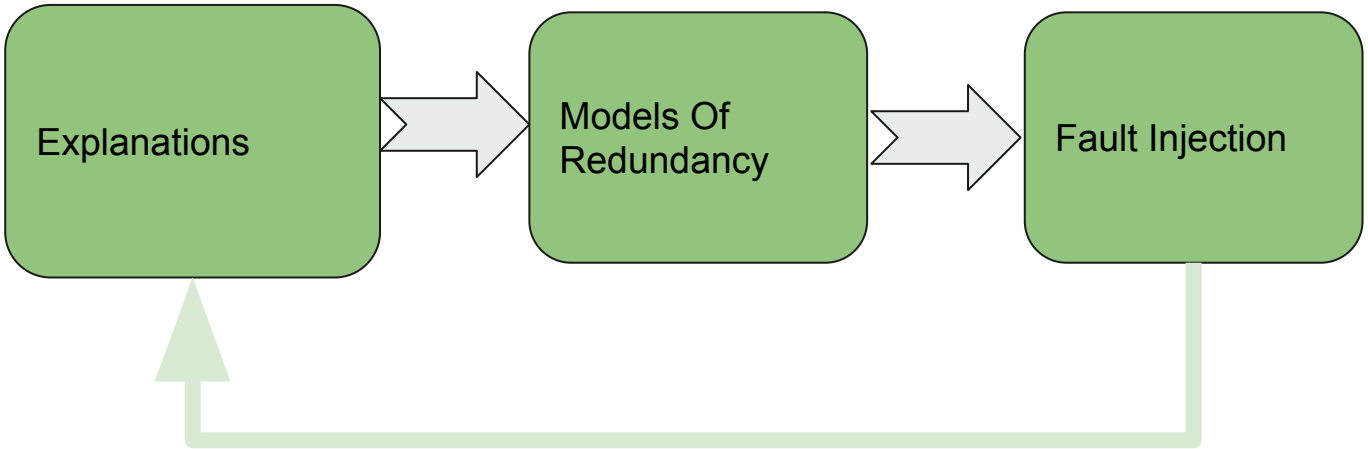
Use the structure of the Trace to prune the Solution Space :

1. Rank Of the Service (distance from the root)
2. Size Of the sub graph of the Service
3. If we survive the failure of C, we will surely survive the failure of D, E and F

Ordering Faults : Injecting Faults That Matter

- All services are not created equal, some services fail more than others
- Likelihood and Containment :
  - P(Node failure) > P(Rack Failure) >> P(Data center failure)
- Historical measures :
  - Time since last release
  - History Of Failure and Bug Rate

LDFI in the Enterprise



Traces = Explanations

What are traces anyway ?

- Ordered Events with context stitched together
- Create the call graphs using service names and endpoints

- Distributed Tracing
  - Call graphs come for free
- Less Ideal (but OK) : Structured Logging
  - We did this too !!

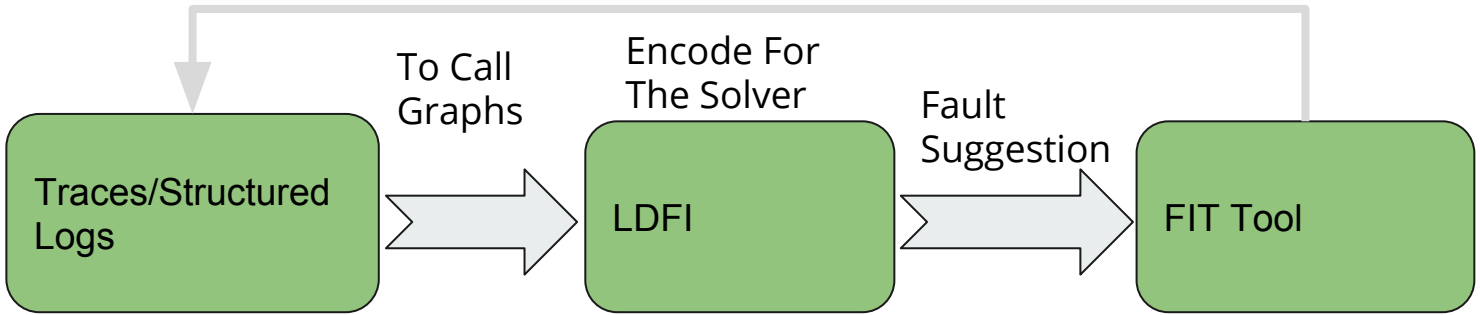
Fault Injection Tool

- We rolled our own (Mowgli)
  - Circuit breaker aware fault injection tool, deals with services and databases
  - Built in safety mechanisms
  - Hooks for AZ level, node level fault injection
  - Audit and Tracking capabilities
- Lots of open source options available
  - Start simple, a script to drop network traffic is also OK
  - https://github.com/dastergon/awesome-chaos-engineering
- Tip : Be safe by default
  - Always have a rollback strategy

Interaction Replay

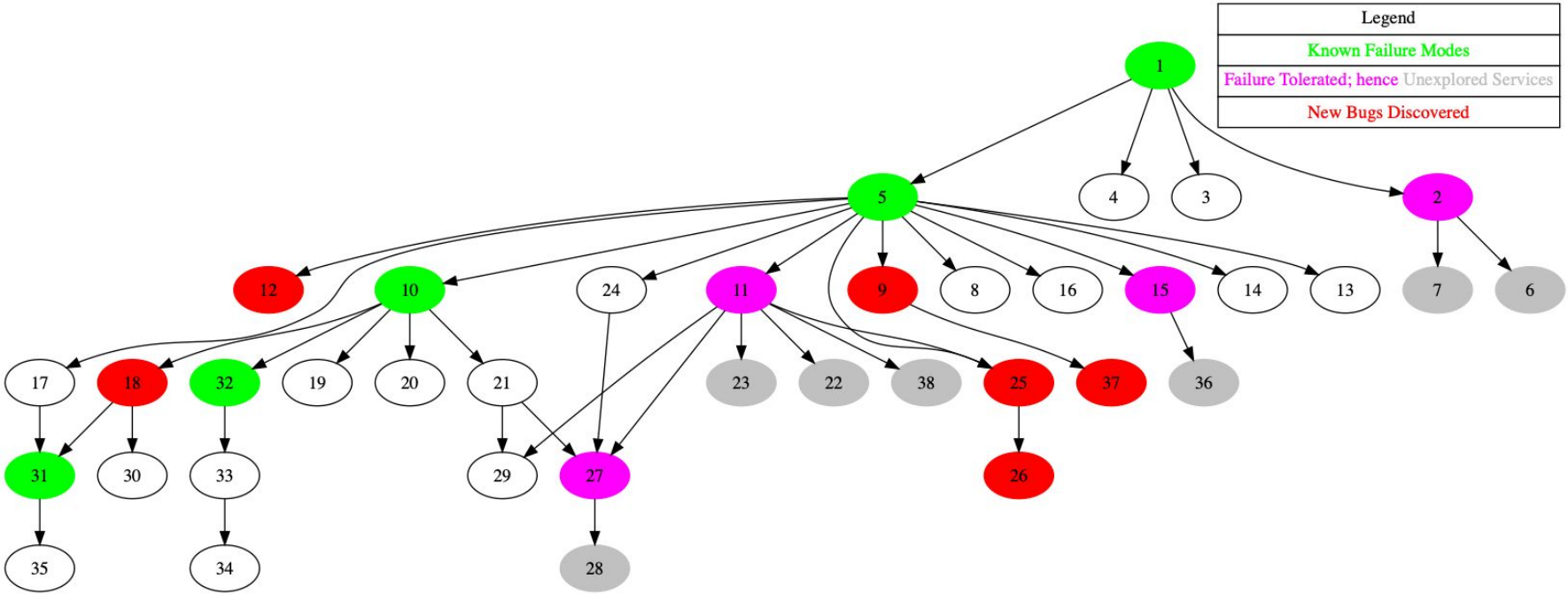
- Ability to *replay* interactions (Tip : E2E Tests)
- Measure of Success
  - A unique binary (yes or no) way of saying whether the execution was successful or not
- Works for Eventually Consistent systems as well, as long as there is finite upper bound on the *eventuality*

LDFI in the Enterprise



- PyCoSAT
- PULP
- SAT4J

Results : Finding Bugs



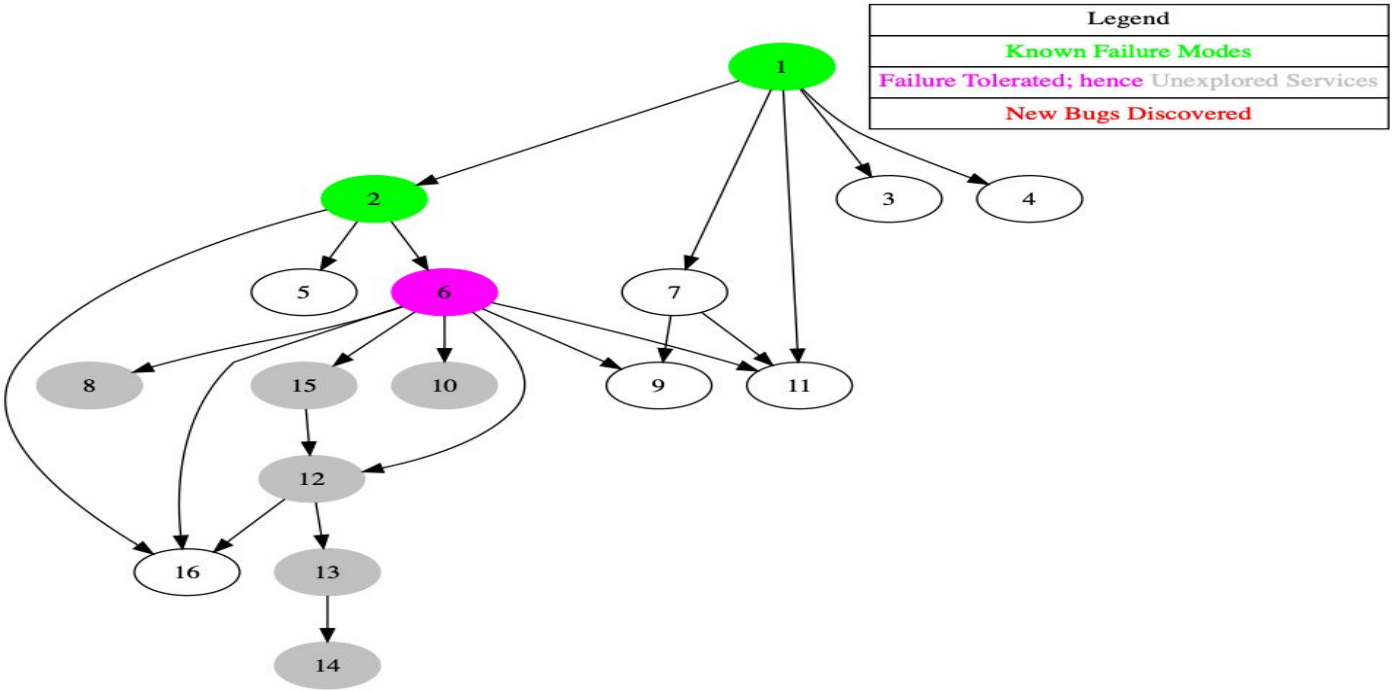
Comparison With Chaos Monkey

Strategy	Fault Experiment Runs (avg.)	Standard Deviation
Ordered LDFI	17	0
Uniform Random	210.35	111.42

How long did it take to find those 5 bugs? A few hours

(An experiment takes ~2 minute, and we did retries to get around our infrastructure)

Results : Finding No Bugs



Madaari : The Road Ahead

- Scalarizing Probabilities of Failure
- SLA verification using strategic Delay Injection
- Fine Grained Fault Injection
- Reason about Stateful systems
- Microservices Only ?
    - Databases, Containers, Service Mesh .. Let’s Go !!

LDFI : The Road Ahead

3 W's For Fault Injection

1. **W**hat to inject ? (type of fault we want to inject)
2. **W**here to inject ? (the target component)
3. **W**hen to inject ? (inject when there are exactly 5 items in the cart !!)

LDFI : The Road Ahead

A Journey from Time to State and back

1. What's time anyway ??
2. Applications have state and change of state gives you implicit order.
3. A rendezvous of state and time gives us precision for fault injection.

Madaari : Key Takeaways

- Industry and Academia can work together for fun(d) and profit
- Limitations of LDFI w.r.t unordered solutions and why ordering matters for chaos engineering experiments
- Understand how LDFI can be integrated in the enterprise by harnessing the observability infrastructure
- Preliminary results of prioritized LDFI and a future direction for the community
- Evangelising new techniques is hard; start small and stay simple

Discussion