Monkeys in Lab Coats

Applied Failure Testing Research at

NETFLIX

The whole is greater than the sum of its parts.

- Aristotle
[Metaphysics]

The Professor

The Practitioner

Peter Alvaro

Kolton Andrus

Ex-Berkeley, Ex-Industry

Assistant Prof @ Santa Cruz

Misses the calm of PhD life

Likes prototyping stuff

Ex-Netflix, Ex-Amazon

‘Chaos’ Engineer

Misses his actual pager

Likes breaking stuff

Measures of Success

Academic

H-Index

Grant warchest

Department ranking

Industry

Availability (i.e. 99.99% uptime)

Number of Incidents

Reduce Operational Burden

An Unlikely Team?


NETFLIX
Fail U re as
238.029
+2
+3
+4
+5
+6
92
2-8-18-32-21-9-2
a Se rvice
78.96
-2
+1
+2
+4
+6
34
2-8-18-6

Works Great!

but ... it’s manual

Surely there is a better way ...


Chrome File Edit View History Bookmarks People Window Help
https://www.youtube.com/watch?v=ggCffvKEJmQ
YouTube
Lineage-driven fault injection
Goal: top-down testing that
• finds all of the fault-tolerance bugs, or
• certifies that none exist
RICON
RICON 2014: Keynote 2 Day 2: Peter Alvaro - Outwards from the Middle of the Maze
Basho Technologies
Subscribe
6,890
Published on Oct 31, 2014
Peter Alvaro, PhD candidate at UC Berkeley
Outwards from the middle of the maze
Wednesday, October 29th
SHOW MORE
Up next
Autoplay
RICON 2014: Jayadev Chandrasekhar, YAHOO! Cloud Infrastructure
Basho Technologies
1,221 views
"I See What You Mean" by Peter Alvaro
Strange Loop
16,378 views
Distributed Systems Archeology (Michael Bernstein) - RICON West 2013
Basho Technologies
3,765 views
RICON 2014: David Greenberg, Two Sigma - Mesos: The Operating System for your
Basho Technologies
2,814 views
DSN 2014 Keynote: "Sibyl: A System for Large Scale Machine Learning at Google"
Tech Talk
7,275 views
Grand Prix Pittsburgh Round 8
ggslive
Recommended for you
RICON 2014: Ayşylu Greenberg, Google - Benchmarking You're Doing It Wrong
Basho Technologies
3,139 views
Wicked Good Ruby 2013 - Bloom: A Language for Disorderly Distributed
Confreaks
1,031 views
♫ 4 HOURS ♫ Classical Music for Studying and Concentration - Best of Bach - Classical
Classical Music
354,769 views
RICON 2014: Keynote 1 Day 2: Dave
ViewSonic
HDMI 1080p Full HD
amazon

Free lunch?

The End?

(Academia + Industry)

Let’s build it

“Can we, pretty please?”

Freedom and Responsibility

NETFLIX Core Value

Responsibility

Academic

Industry

Prove that it works

Show that it scales

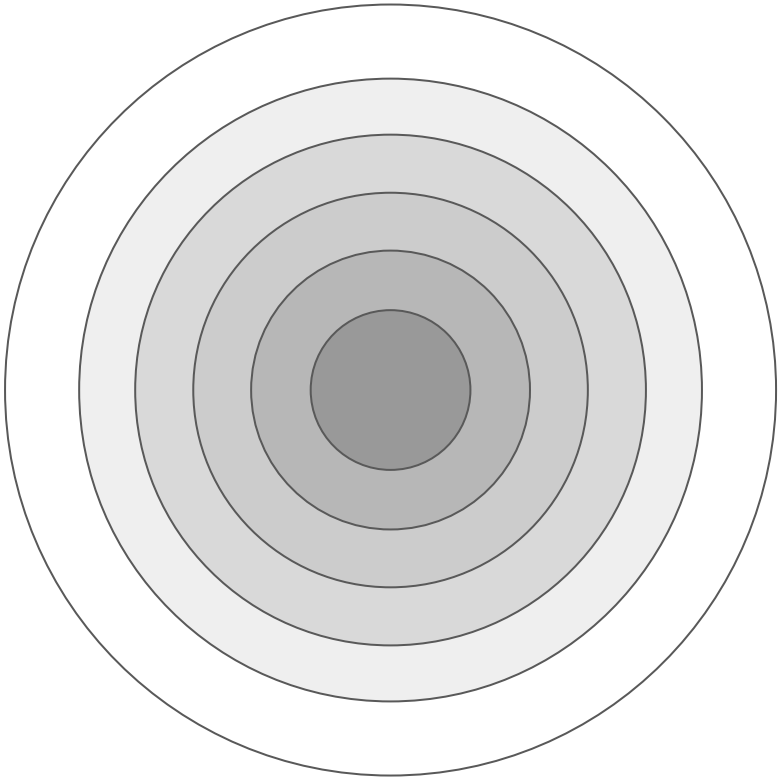
Find real bugs

The Big Idea

Lineage Driven Fault Injection

What could *possibly* go wrong?

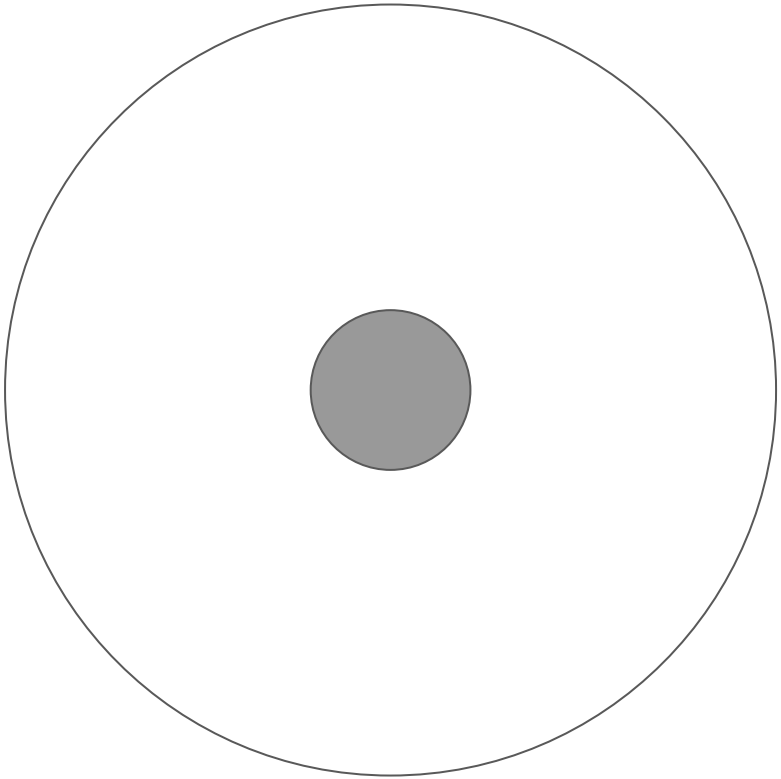
Consider computation involving 100 services



Search Space: 2^{100} executions

“Depth” of bugs

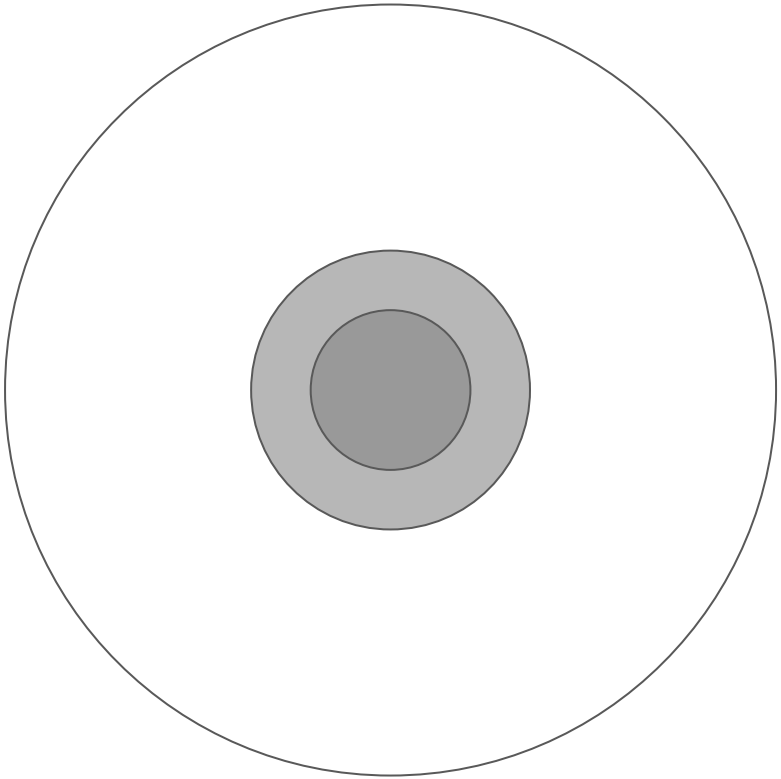
Single Faults



Search Space:
100 executions

“Depth” of bugs

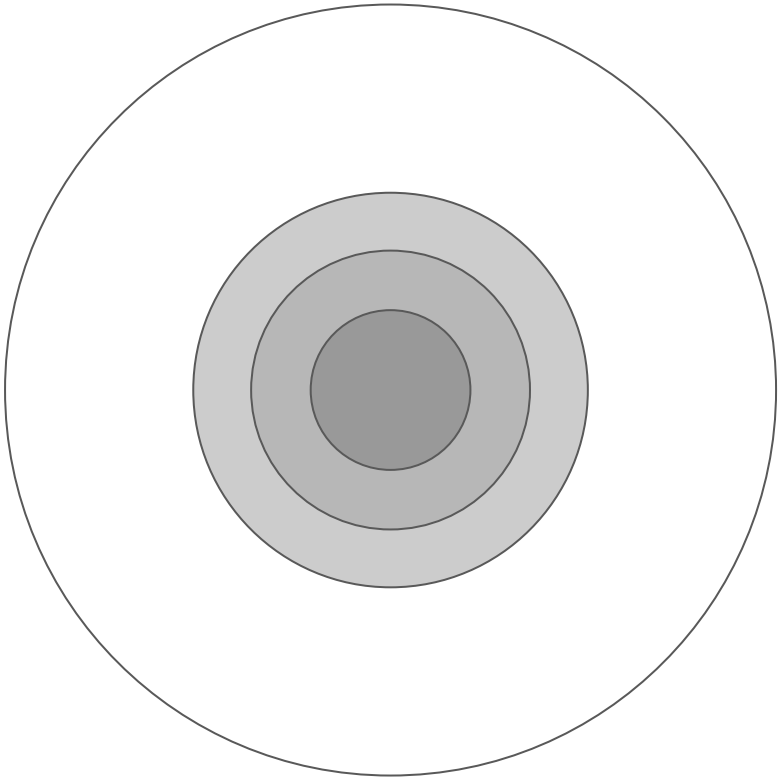
Combination of 4 faults



Search Space:
3M executions

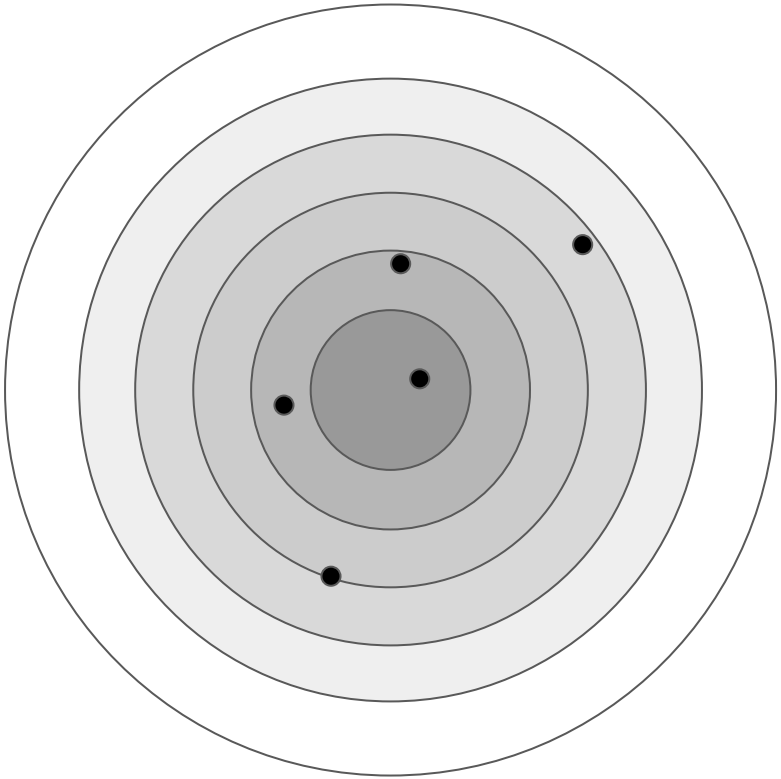
“Depth” of bugs

Combination of 7 faults



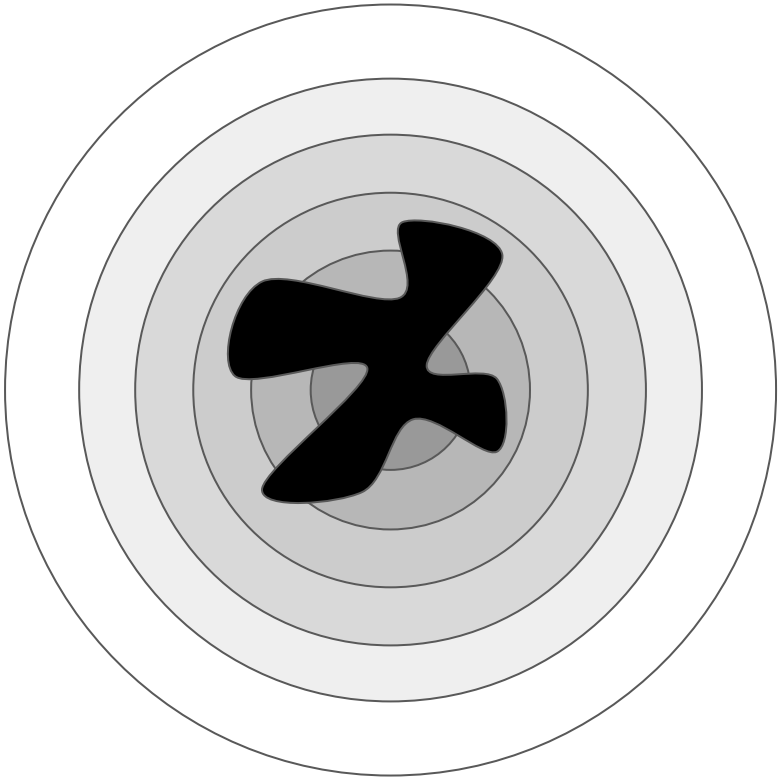
Search Space:
16B executions

Random Search



Search Space:
2^{100} executions

Engineer-guided Search



Search Space:
???

Fault-tolerance “is just” redundancy

Lineage-driven Fault Injection

Peter Alvaro
UC Berkeley
palvaro@cs.berkeley.edu

Joshua Rosen
UC Berkeley
rosenville@gmail.com

Joseph M. Hellerstein
UC Berkeley
hellerstein@cs.berkeley.edu

ABSTRACT

Failure is always an option; in large-scale data management systems, it is practically a certainty. Fault-tolerant protocols and components are notoriously difficult to implement and debug. Worse still, choosing existing fault-tolerance mechanisms and integrating them correctly into complex systems remains an art form, and programmers have few tools to assist them.

We propose a novel approach for discovering bugs in fault-tolerant data management systems: *lineage-driven fault injection*. A lineage-driven fault injector reasons *backwards* from correct system outcomes to determine whether failures in the execution could have prevented the outcome. We present MOLLY, a prototype of lineage-driven fault injection that exploits a novel combination of data lineage techniques from the database literature and state-of-the-art satisfiability testing. If fault-tolerance bugs exist for a particular configuration, MOLLY finds them rapidly, in many cases using an order of magnitude fewer executions than random fault injection. Otherwise, MOLLY certifies that the code is bug-free for that configuration.

enriching new system architectures with well-understood fault tolerance mechanisms and henceforth assuming that failures will not affect system outcomes. Unfortunately, fault-tolerance is a *global* property of entire systems, and guarantees about the behavior of individual components do not necessarily hold under composition. It is difficult to design and reason about the fault-tolerance of individual components, and often equally difficult to assemble a fault-tolerant system even when given fault-tolerant components, as witnessed by recent data management system failures [16, 57] and bugs [36,49].

Top-down testing approaches—which perturb and observe the behavior of complex systems—are an attractive alternative to verification of individual components. Fault injection [1,26,36,44,59] is the dominant top-down approach in the software engineering and dependability communities. With minimal programmer investment, fault injection can quickly identify shallow bugs caused by a small number of independent faults. Unfortunately, fault injection is poorly suited to discovering rare counterexamples involving complex combinations of multiple instances and types of faults (e.g., a network partition followed by a crash failure). Ap-

But how do we know redundancy when we see it?

Hard question: “Could a bad thing ever happen?”

Easier: “Exactly *why* did a good thing happen?”

“What could have gone wrong?”

Lineage-driven fault injection

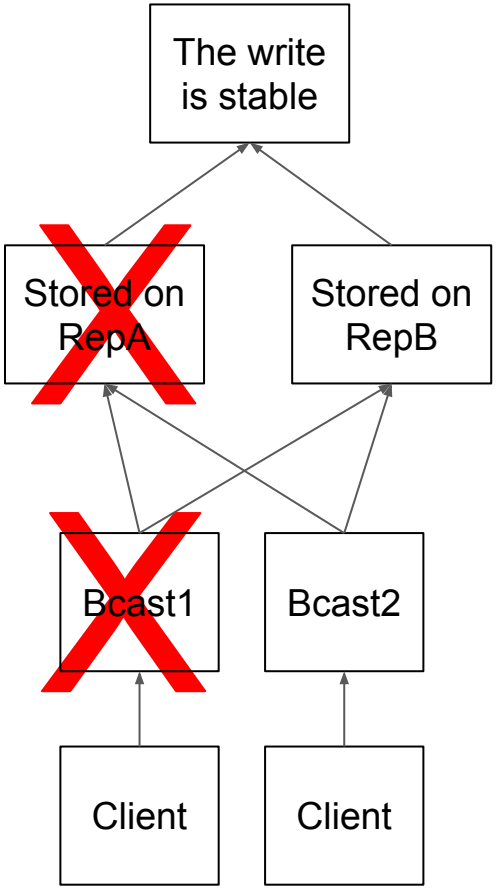
Why did a good thing happen?

Consider its *lineage*.

What could have gone wrong?

Faults are *cuts* in the lineage graph.

Is there a cut that breaks all supports?



Lineage-driven fault injection

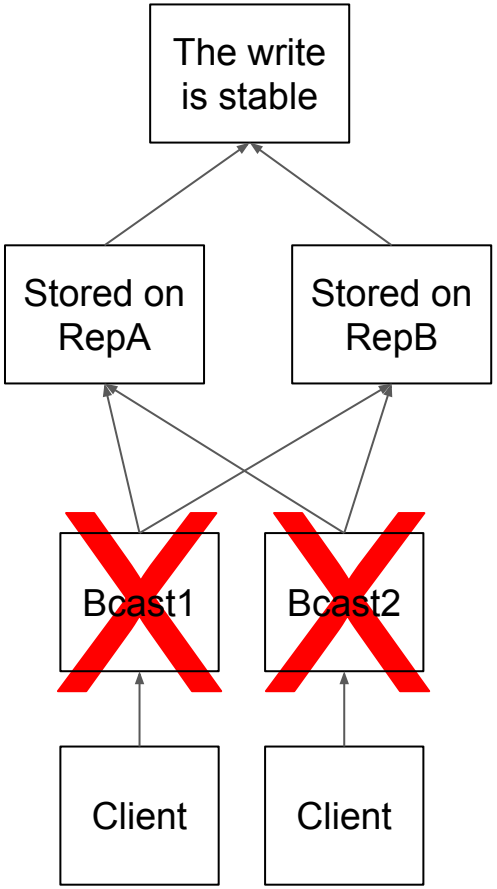
Why did a good thing happen?

Consider its *lineage*.

What could have gone wrong?

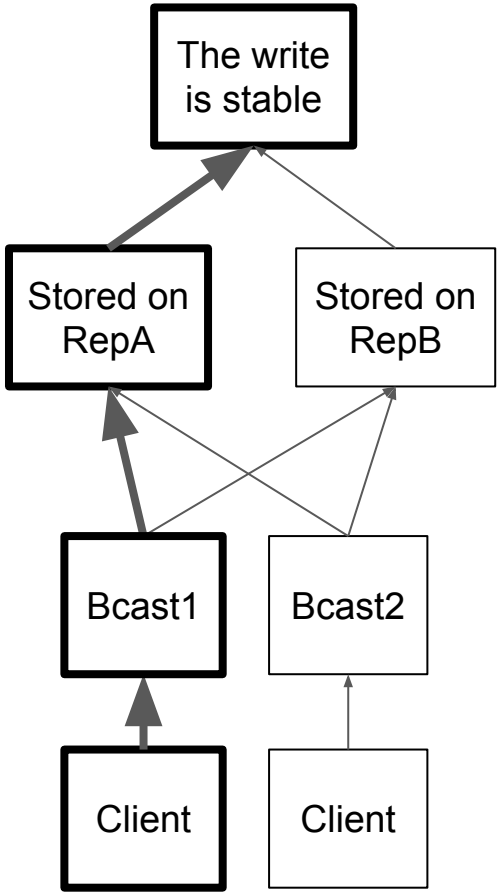
Faults are *cuts* in the lineage graph.

Is there a cut that breaks all supports?



What would *have to* go wrong?

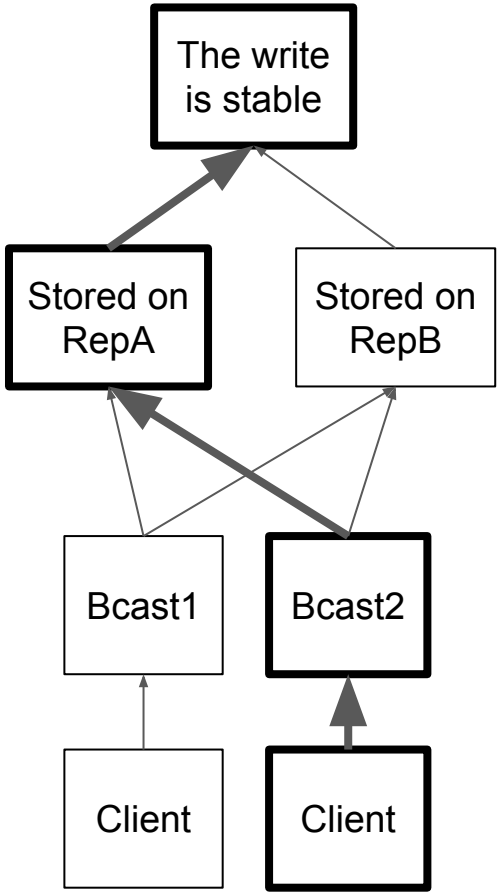
(RepA OR Bcast1)



What would *have to* go wrong?

(RepA OR Bcast1)

AND (RepA OR Bcast2)

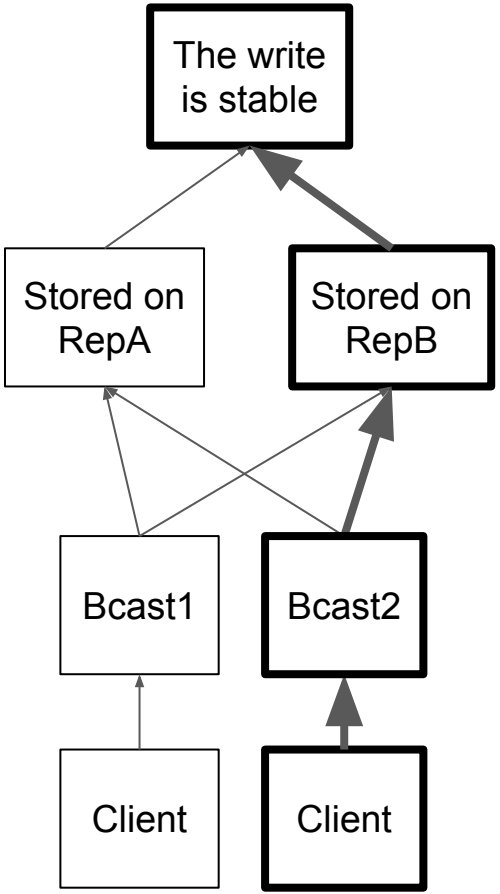


What would *have to* go wrong?

(RepA OR Bcast1)

AND (RepA OR Bcast2)

AND (RepB OR Bcast2)



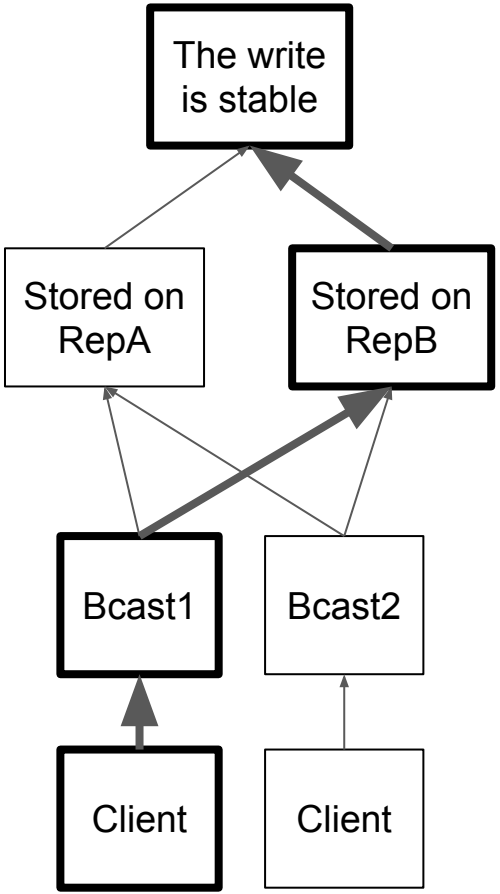
What would *have to* go wrong?

(RepA OR Bcast1)

AND (RepA OR Bcast2)

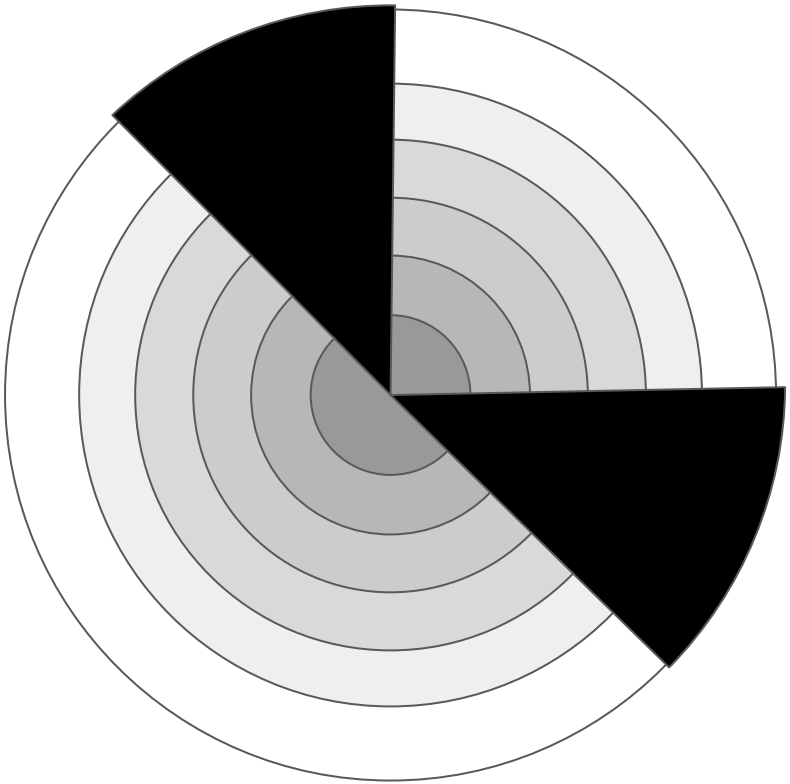
AND (RepB OR Bcast2)

AND (RepB OR Bcast1)



Search Space Reduction

Each Experiment finds a bug, OR

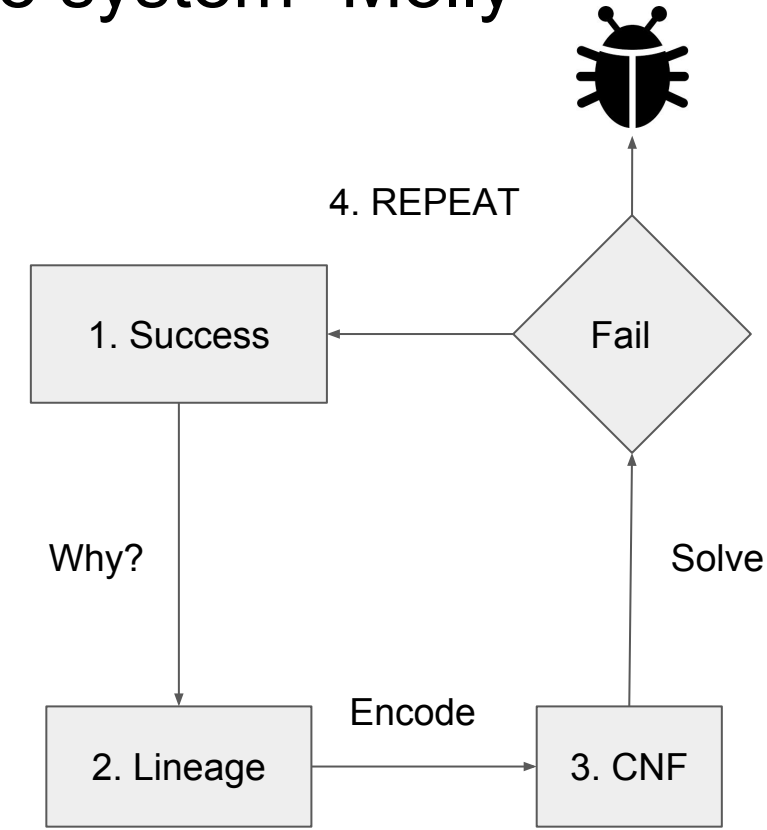


Reduces the Search space

The prototype system “Molly”

Recipe:

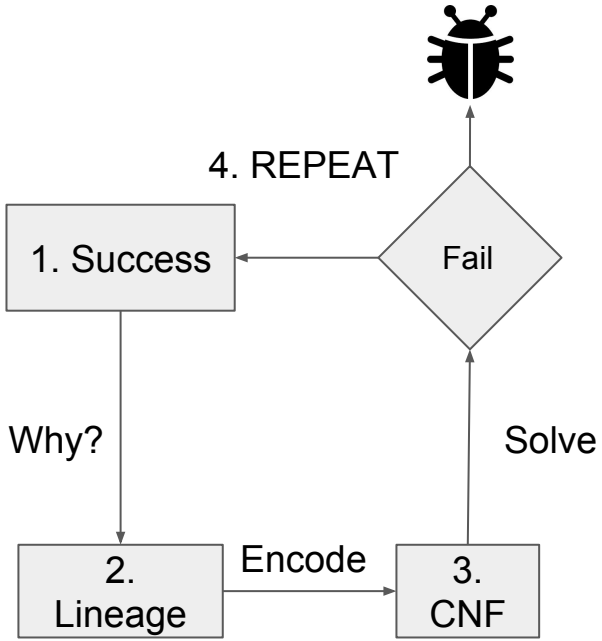
1. Start with a successful outcome. Work backwards.
2. Ask *why* it happened: Lineage
3. Convert lineage to a boolean formula and solve
4. Lather, rinse, repeat



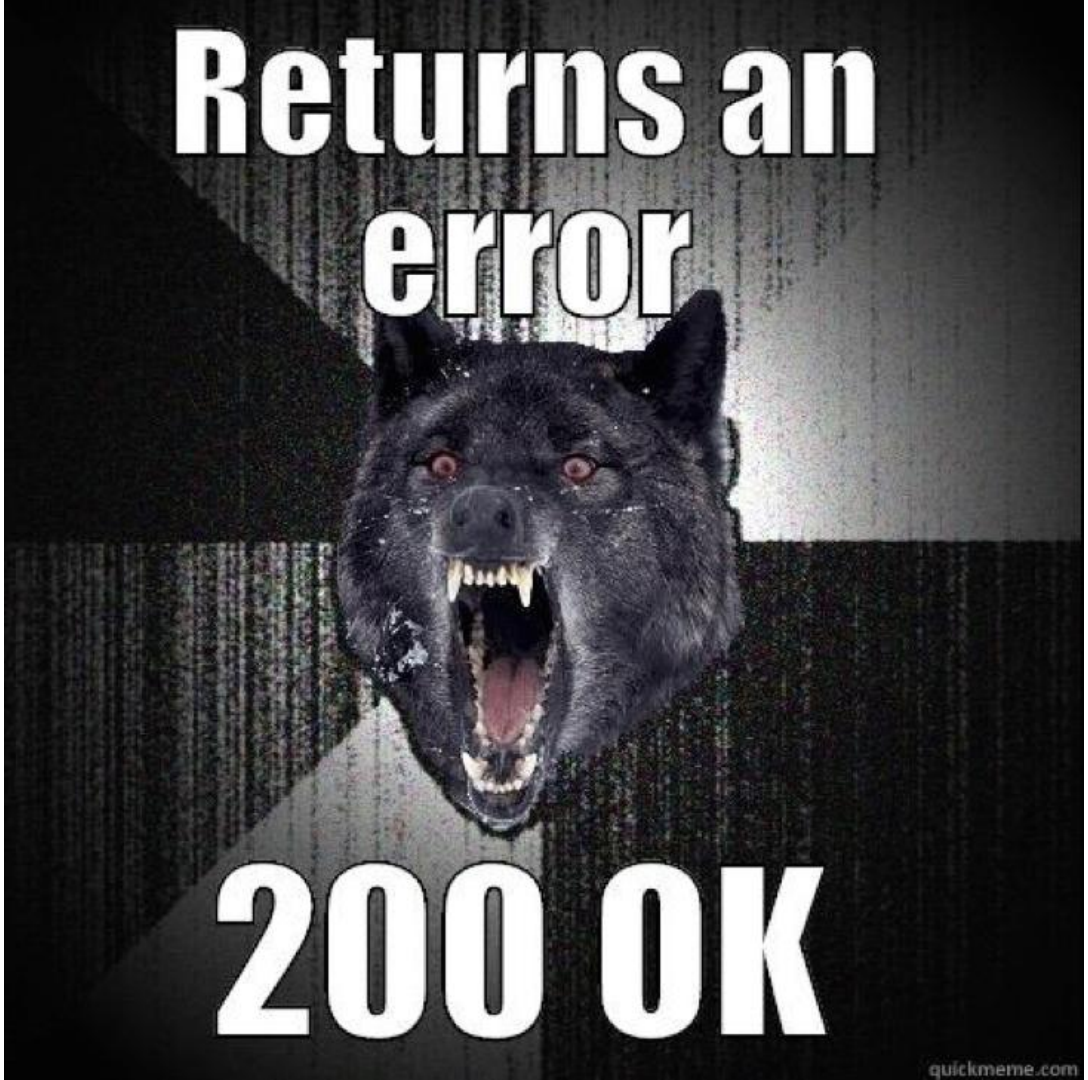
The Big Idea

Meets Production

1. Start with a successful outcome



What is success?


Returns an error
200 OK
quickmeme.com

“Start with the customer and work backwards”

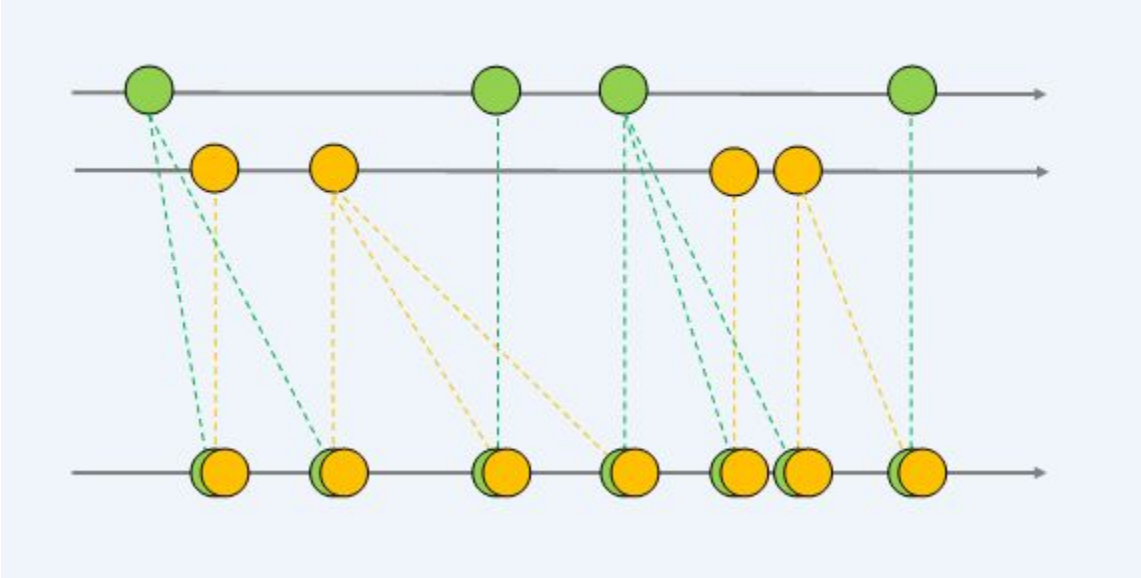
amazon Leadership Principle


NETFLIX
XBOX 360
NETFLIX
UNBREAKABLE
KIMMY SCHMIDT
SAMSUNG
NETFLIX
MARCO
POLO

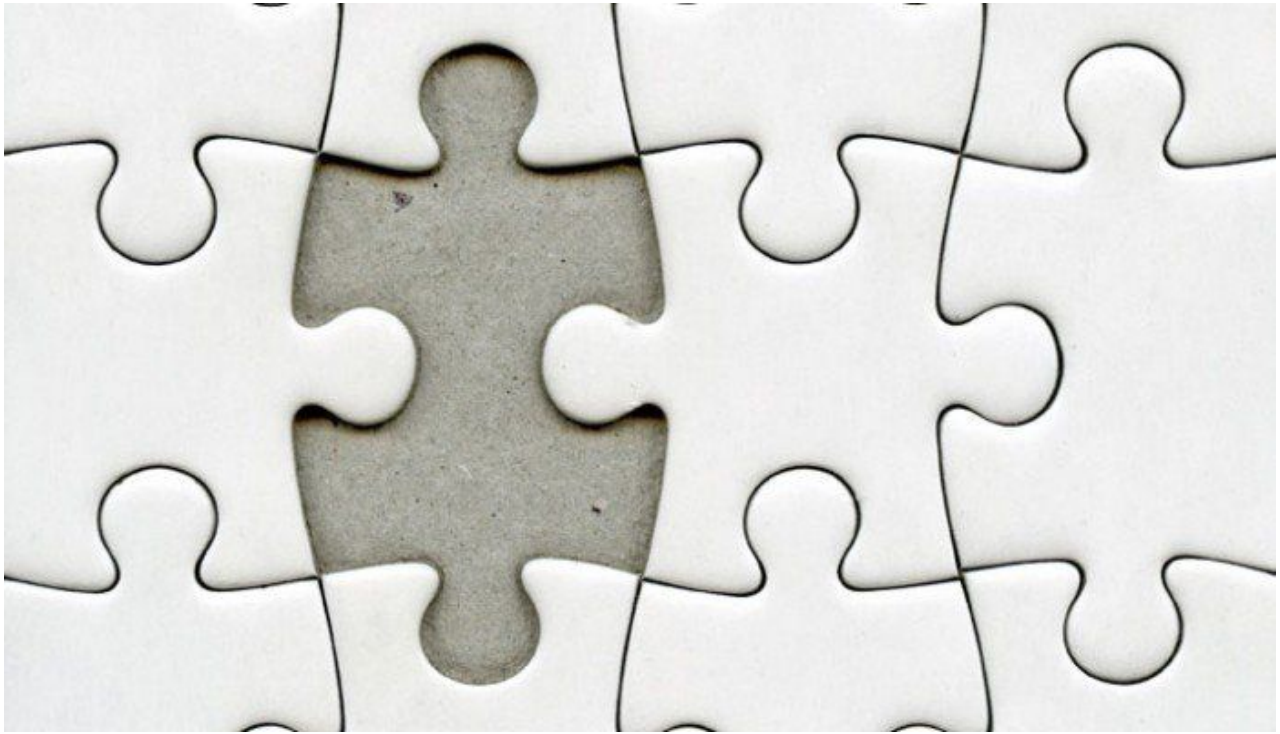

NETFLIX
Unable to connect to Netflix. Please try again or visit:
www.netflix.com/tvhelp
Try Again
Exit
ui-200
NETFLIX

“Streaming” Data

Joining the Streams



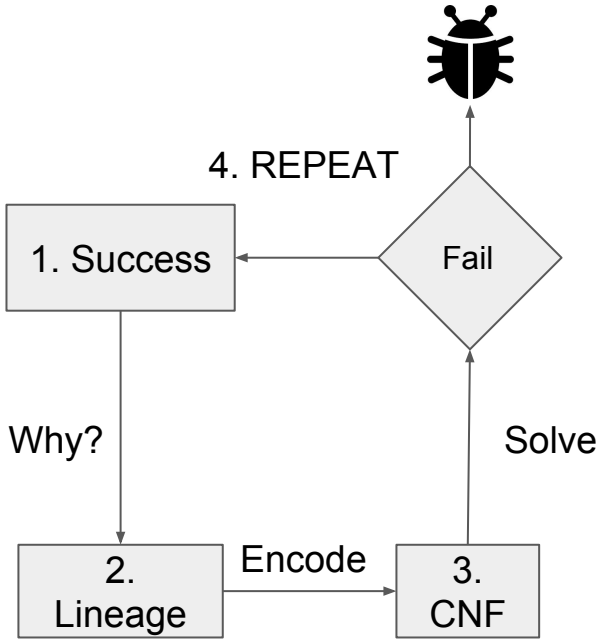
Missing Data?



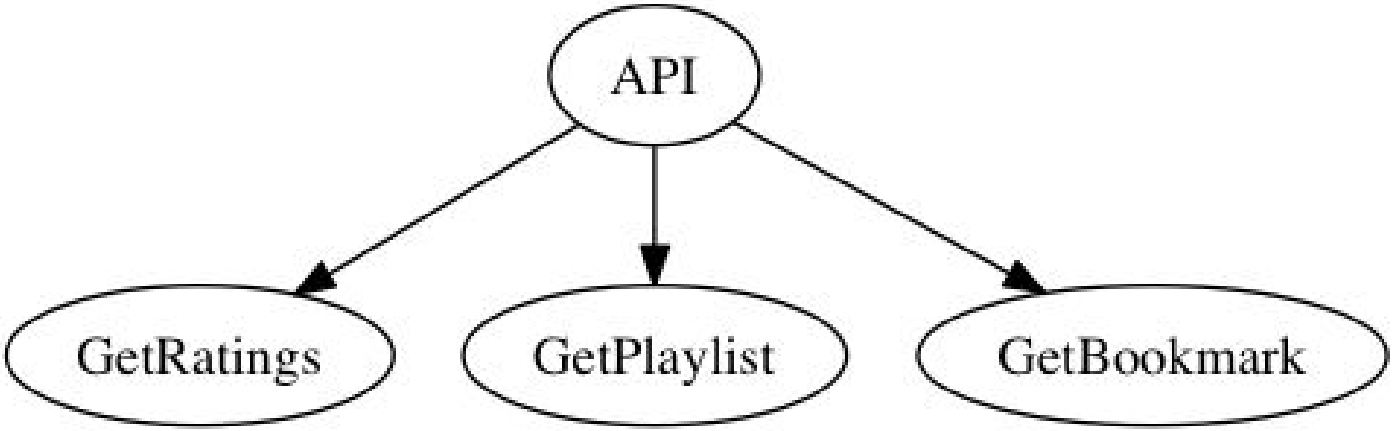
Lesson 1

Work backwards from what you know

2. Ask why it happened

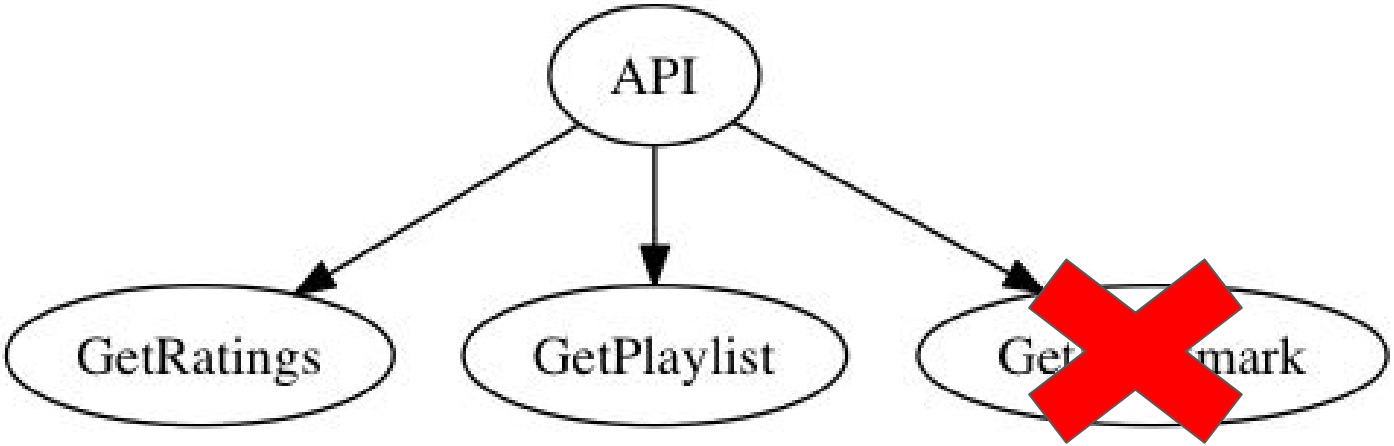


Request Tracing

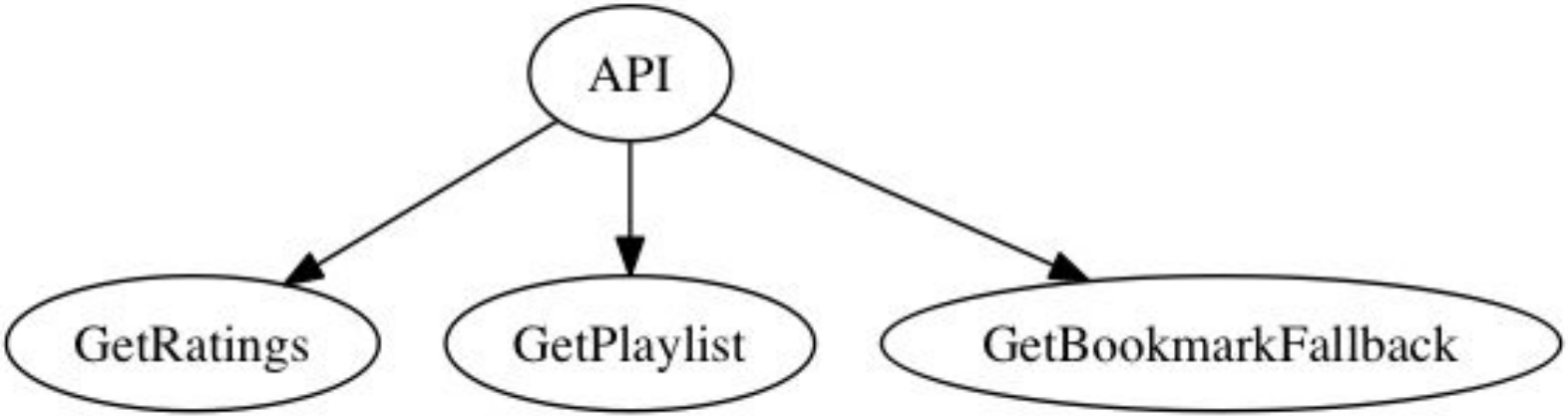



GAME
OVER

Request Tracing

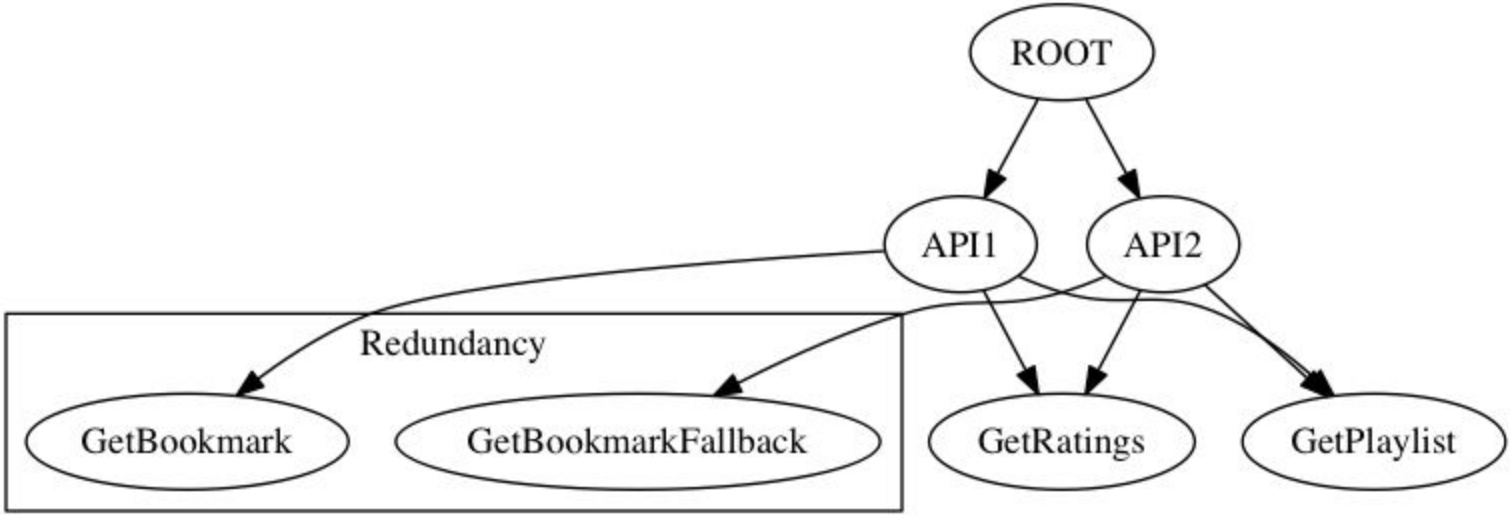


Alternate Execution



Evolution over time

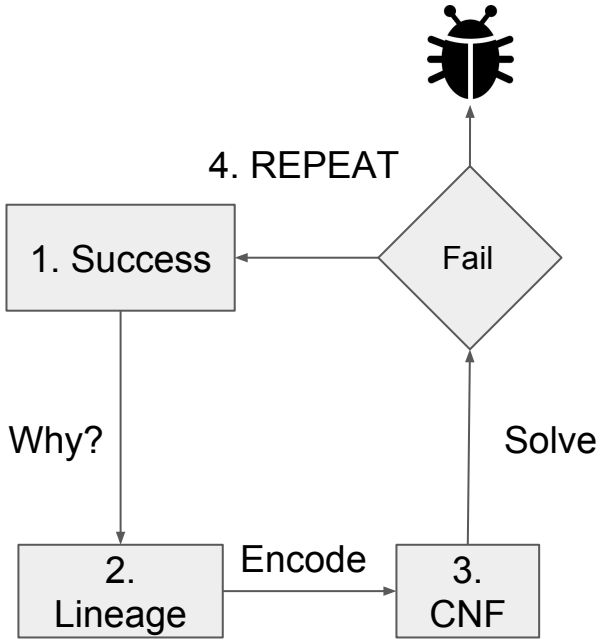
Redundancy through History



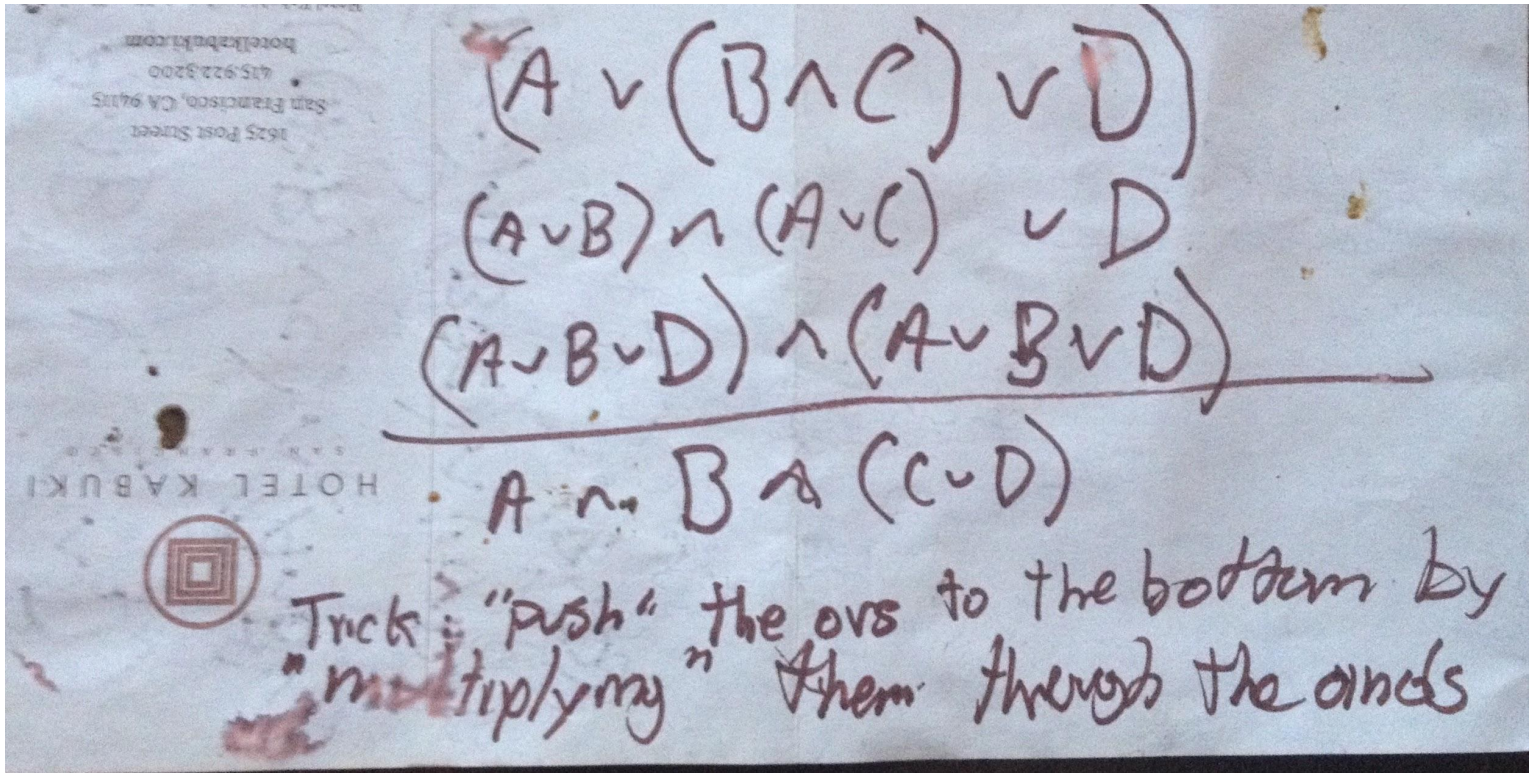
Lesson 2

Meet in the middle

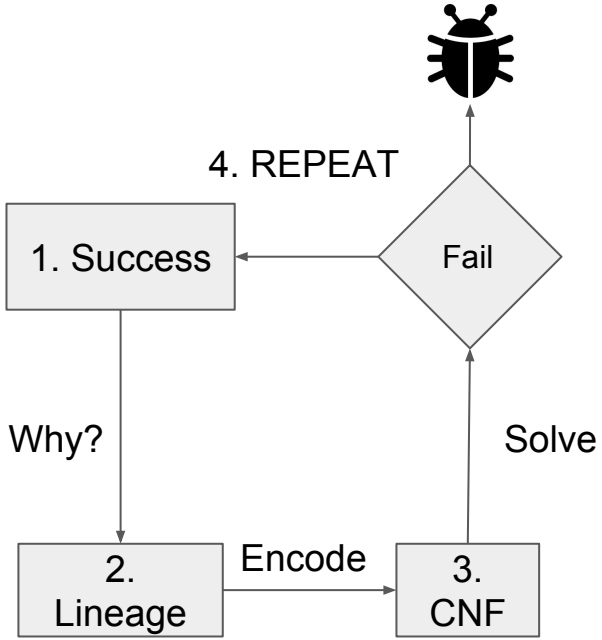
3. Solve



A “small” matter of code

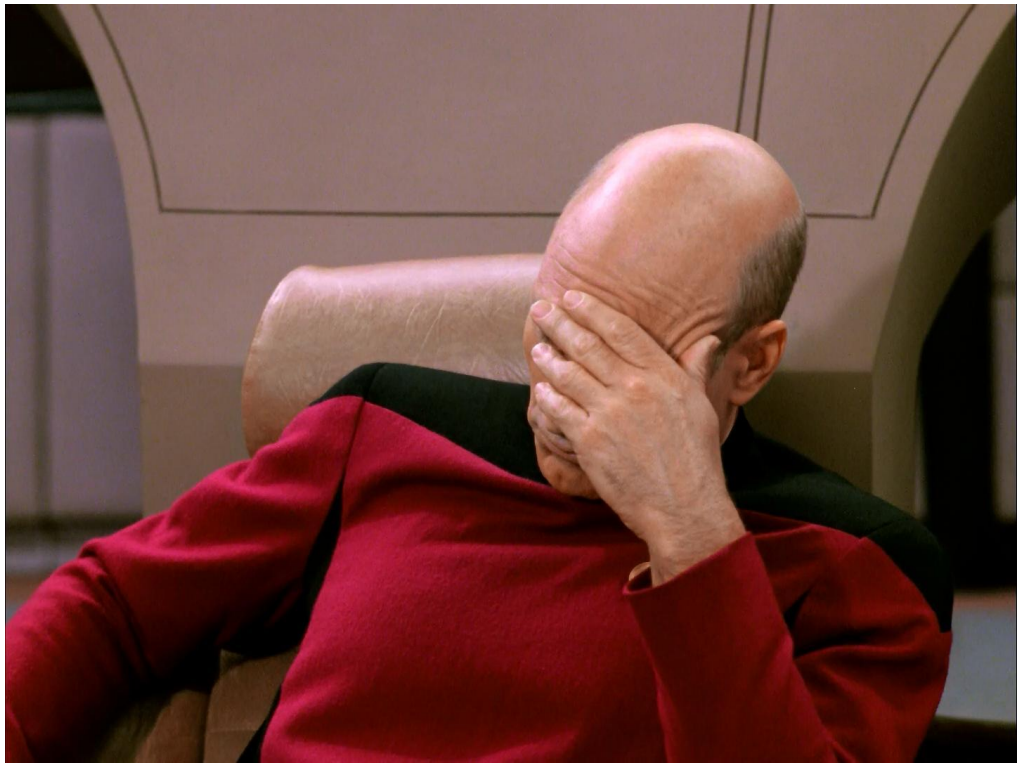


4. Lather, Rinse, Repeat



Turn the crank, right?

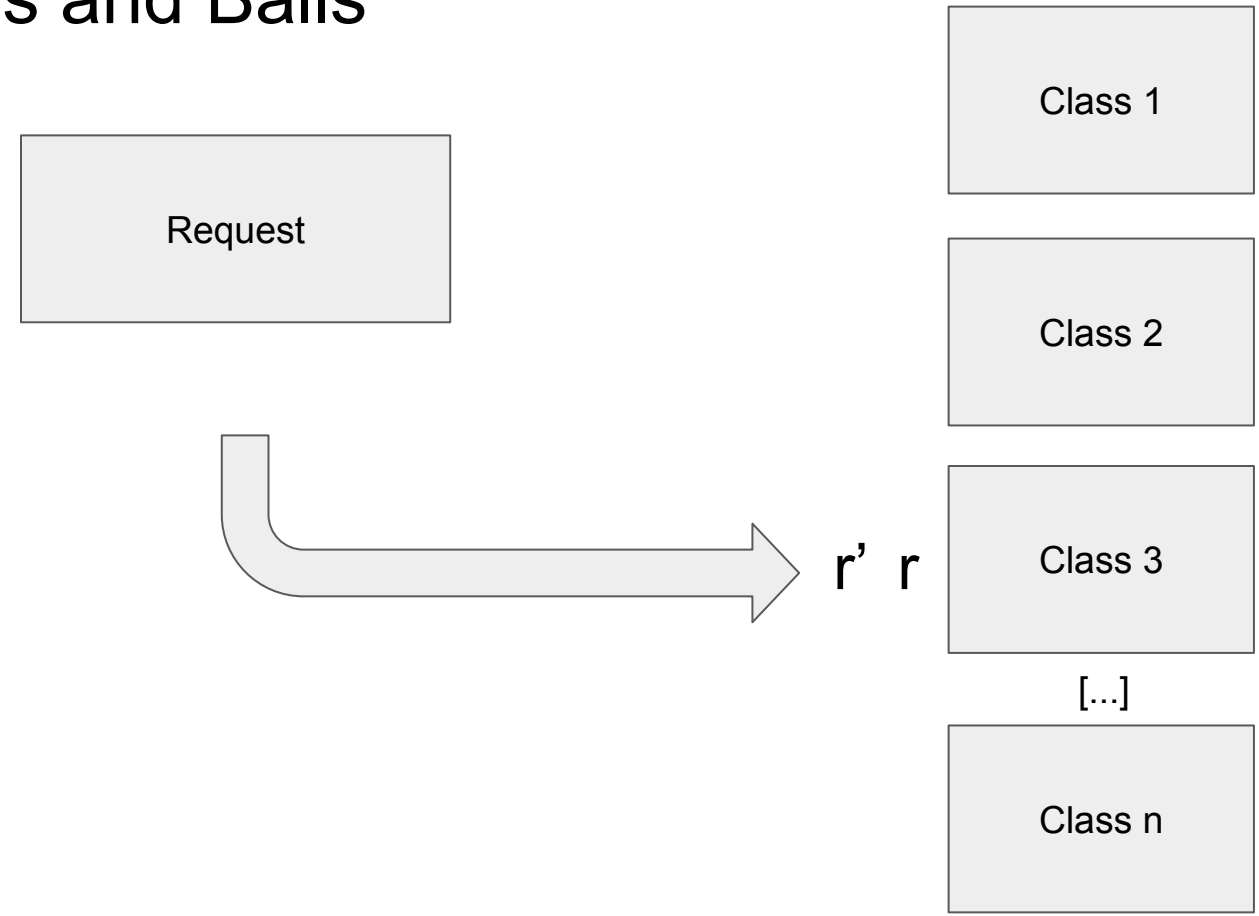
Idempotence



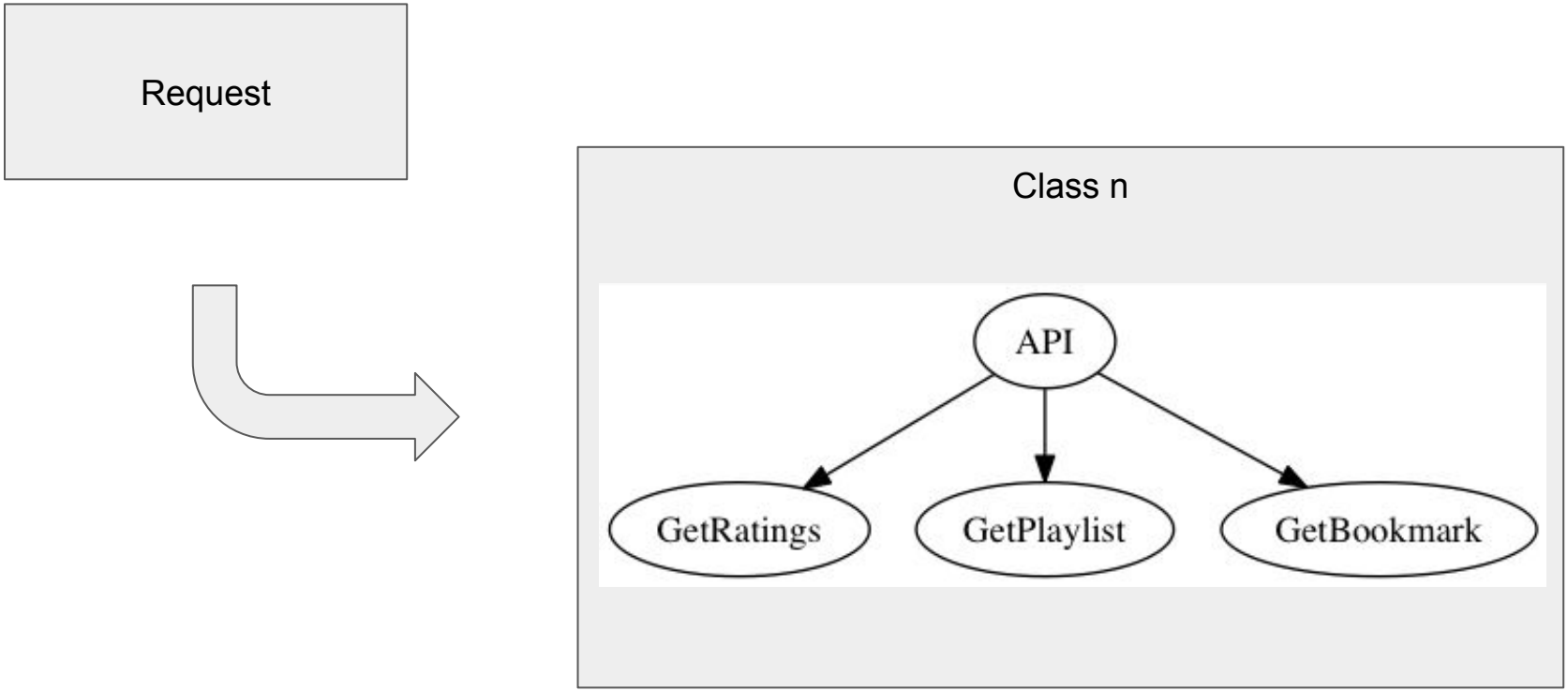

©NEILWADE.COM


GAME
OVER

Bins and Balls

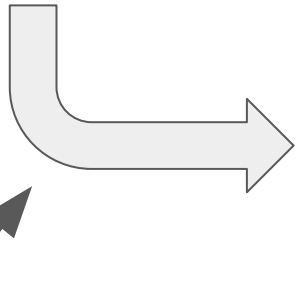


Predicting Request Graphs

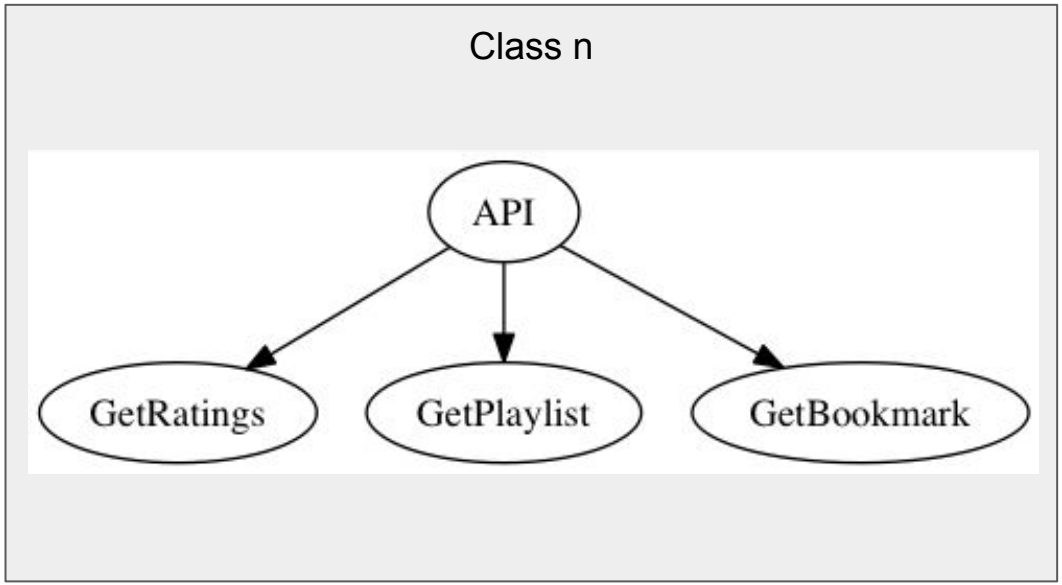


Predicting Request Graphs

Request

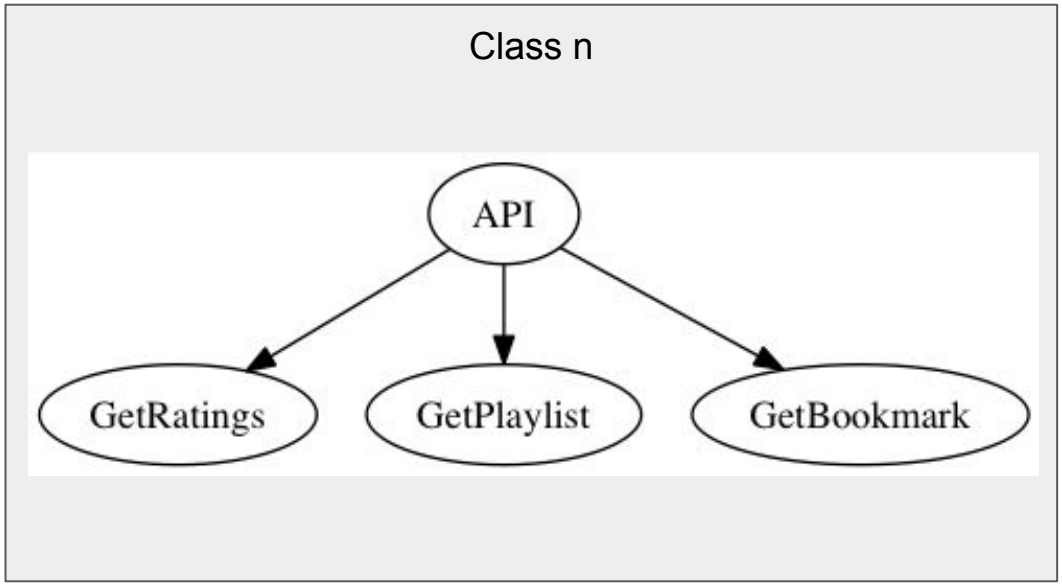


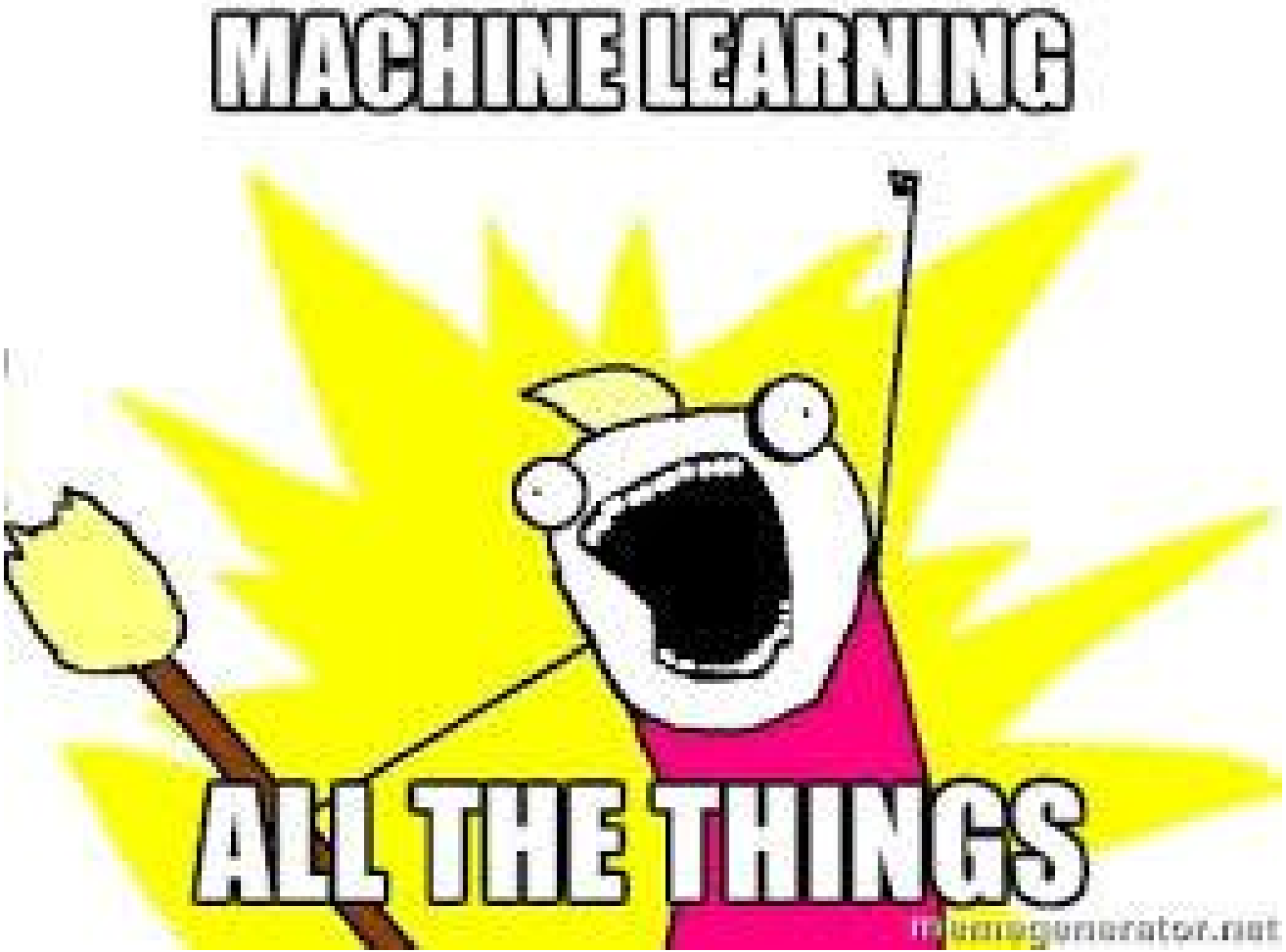
Some function f:
Requests → Classes



Predicting Request Graphs

F(Request) =




MACHINE LEARNING
ALL THE THINGS

Solve the Machine Learning problem?
or the Failure Testing one?

Simplest thing that will work?

Falcor Path Mapping

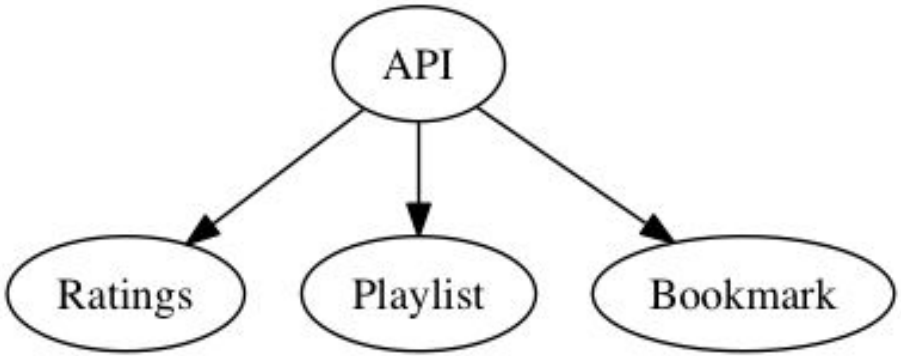
```
["bookmarks”, “recent”]

["playlist", 0, “name”]

["ratings"]
```

=>

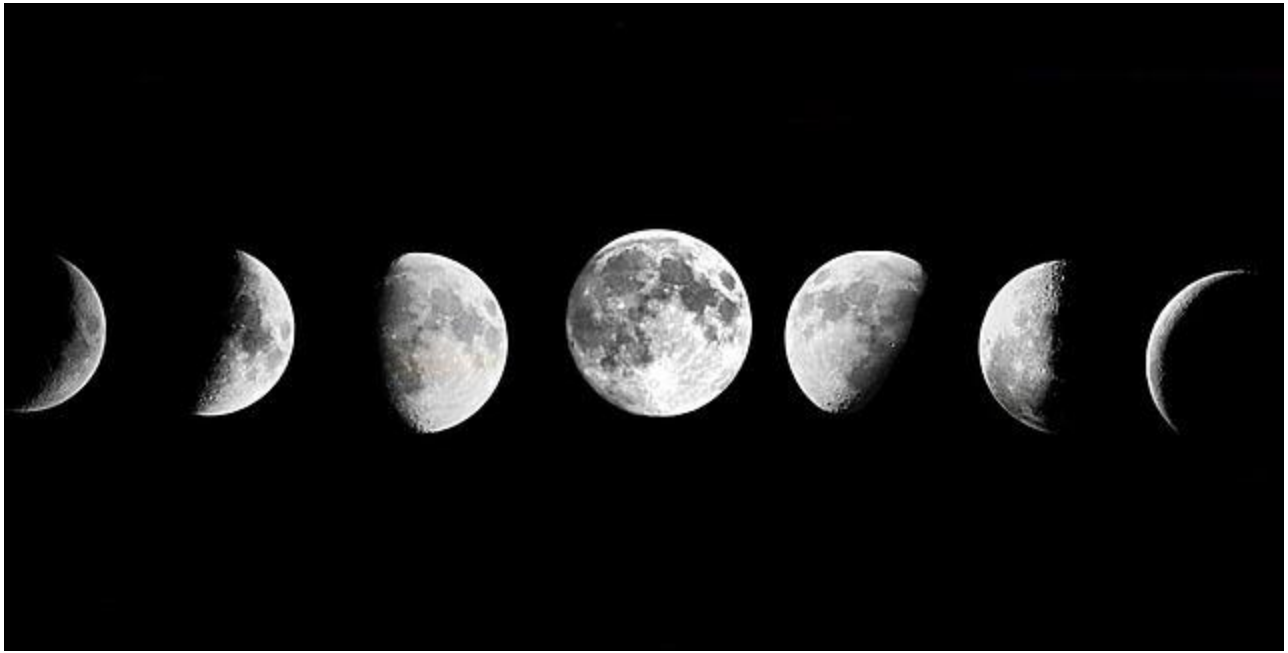
“bookmarks,playlist,ratings”



Lesson 3

Adapt the theory to the reality

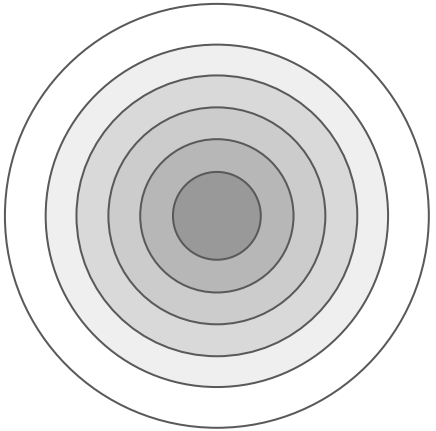
Many moons passed...



Does it work?

YES!

Case study: “Netflix AppBoot”



Services	~100
Search space (executions)	2^{100} (1,000,000,000,000,000,000,000,000,000,000)
Experiments performed	200
Critical bugs found	6

Future Work

Search prioritization

Richer lineage collection

Exploring temporal interleavings

Richer device metrics

Request class creation

Better experiment selection

Lessons

Work backwards from what you know

Meet in the middle

Adapt the theory to the reality

Academia + Industry

~~Academia + Industry~~

Academia ✖ Industry

Thank You!

Peter Alvaro

@palvaro

palvaro@ucsc.edu

Kolton Andrus

@KoltonAndrus

kolton@gremlininc.com

References

- Netflix Blog on ‘Automated Failure Testing’ http://techblog.netflix.com/2016/01/automated-failure-testing.html

- Netflix Blog on ‘Failure Injection Testing’ techblog.netflix.com/2014/10/fit-failure-injection-testing.html

- ‘Lineage Driven Fault Injection’ http://people.ucsc.edu/~palvaro/molly.pdf

Photo Credits

- http://etc.usf.edu/clipart/4000/4048/children_7_lg.gif
- http://cdn.c.photoshelter.com/img-get2/I0000MIN8fL0q8AA/fit=1000x750/taiwan-hiking-river-tracing-walking.jpg
- http://i.imgur.com/iWKad22.jpg
- https://blogs.endjin.com/2014/05/event-stream-manipulation-using-rx-part-2/
- http://youpivot.com/category/features/
- https://www.cloudave.com/33427/boards-need-evolve-time/
- https://www.linkedin.com/pulse/amelia-packager-missing-data-imputation-ramprakash-veluchamy