RELIABILITY IN MODERN CLOUD SYSTEMS

Summer 2025

LOGISTICS

ASSIGNMENT 2

- Grades will be released Friday

ASSIGNMENT 3

- Due Date: Wednesday, June 18th 5pm
- Description: Reproducing a retry storm metastability failure
  - Not using the luggage sharing application
  - Instead using Hotel Reservation from DeathStarBench benchmark suite

ASSIGNMENT 4: OPEN ENDED PROJECT

- Potential topics have been posted on CMS
- Will be done in teams of 2 (1 team will have 3 members)
  - Each group will work on a separate project
- Projects will be assigned by the instructors
  - Send top 5 project preferences and teams via e-mail by Monday 5pm CEST
  - Email Id: vaastav@mpi-sws.org
- If choosing a project from intermediate difficulty as a top choice then please include a justification as to why

ASSIGNMENT 4 GRADING

- Each member of the team will be assigned the same grade
- 40% Presentation (16th July, 2025)
  - 25% on content
  - 10% Q/A
  - 5% presentation
- 60% Implementation (due 21st July, 2025 9am PST)
  - 30% Technical Implementation * Difficulty Bonus
  - 10% weekly check ins (every monday during office hours)
  - 10% Integration + Use of Blueprint
  - 10% Ease-of-use + Documentation

5% Bonus available for those who successfully produce a Pull Request for Blueprint

METASTABILITY ANALYSIS DISCUSSION

DISCUSSION THEMES

- How to analyze systems for metastability without actually executing the system?

ANALYZING METASTABILITY FAILURES

ANALYZING METASTABILITY FAILURES – KEY PROBLEM

We want to identify where our systems are vulnerable to metastable faillures **before they fail!**

ANALYZING METASTABILITY FAILURES – CHALLENGES

We want to identify where our systems are vulnerable to metastable faillures **before they fail!**

Analyzing for metastability is difficult

- Systems are arbitrarily large and complex
- Large parameter space
- Need to experiment with large parameter sweeps to find out potential breaking limits of the system

ANALYZING METASTABILITY FAILURES – CHALLENGES

We want to identify where our systems are vulnerable to metastable faillures **before they fail!**

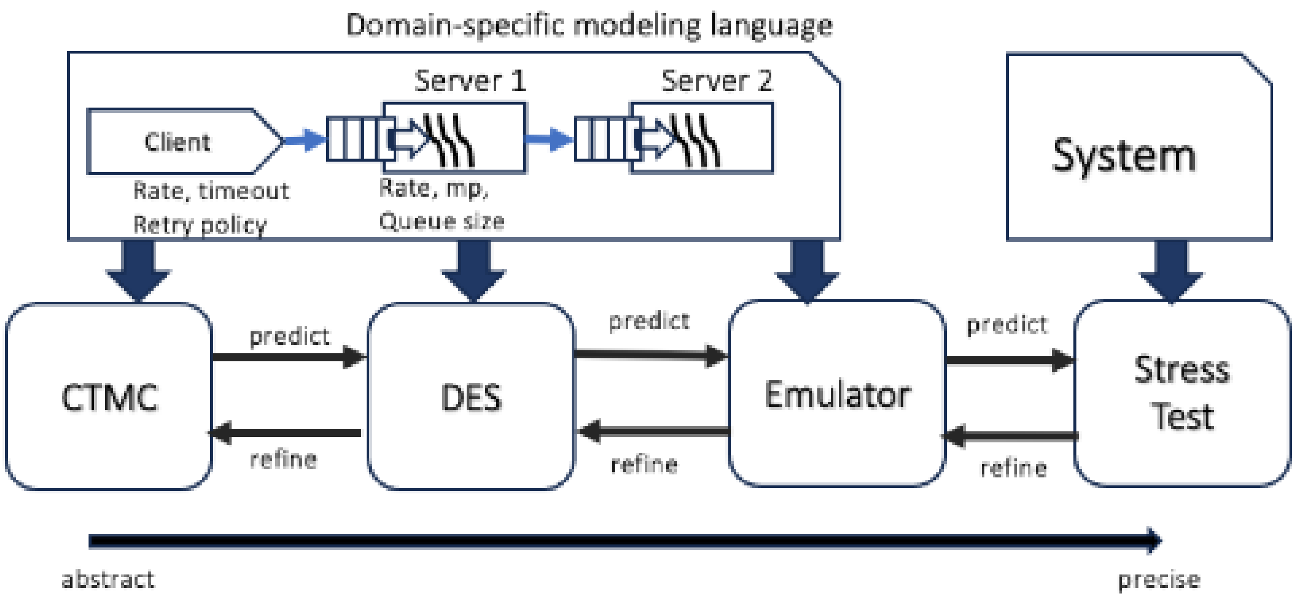
Analyzing for metastability is difficult

- ❖ Systems are arbitrarily large and complex
- ❖ Large parameter space
- ❖ Need to experiment with large parameter sweeps to find out potential breaking limits of the system

Requires a lot of manual effort

ANALYZING WITHOUT EXECUTING

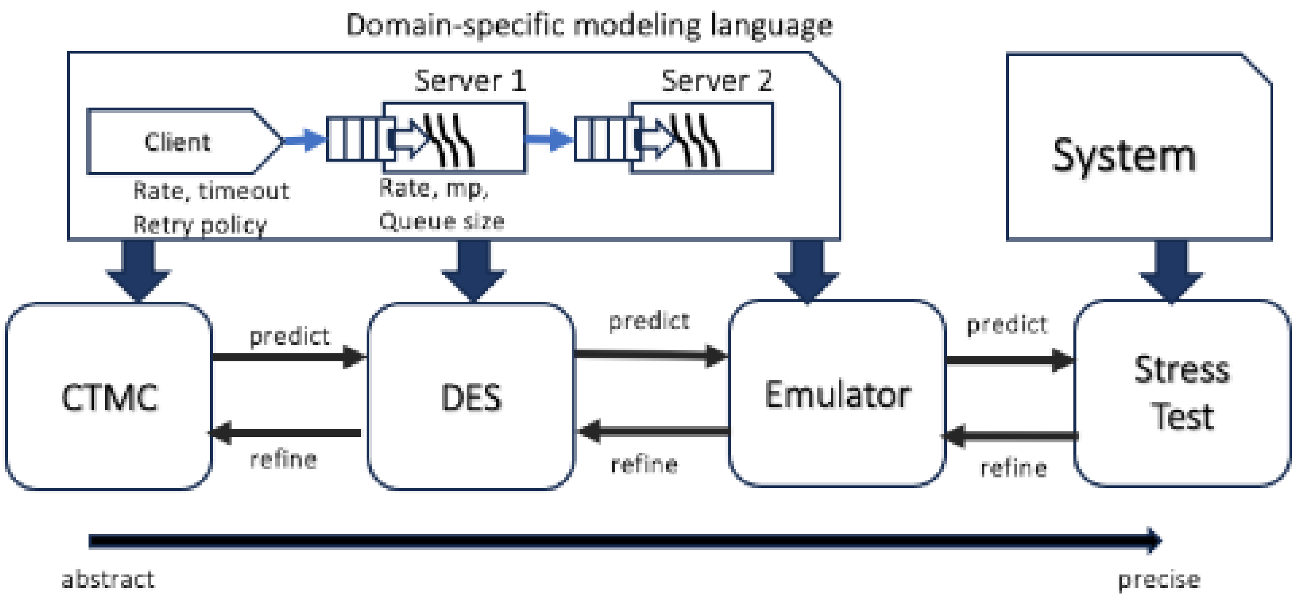
Idea: Instead of executing the system and doing large parameter sweeps, analyze the same server configurations at different levels of abstraction



ANALYZING WITHOUT EXECUTING

Idea: Analyze the same server configurations at different levels of abstraction

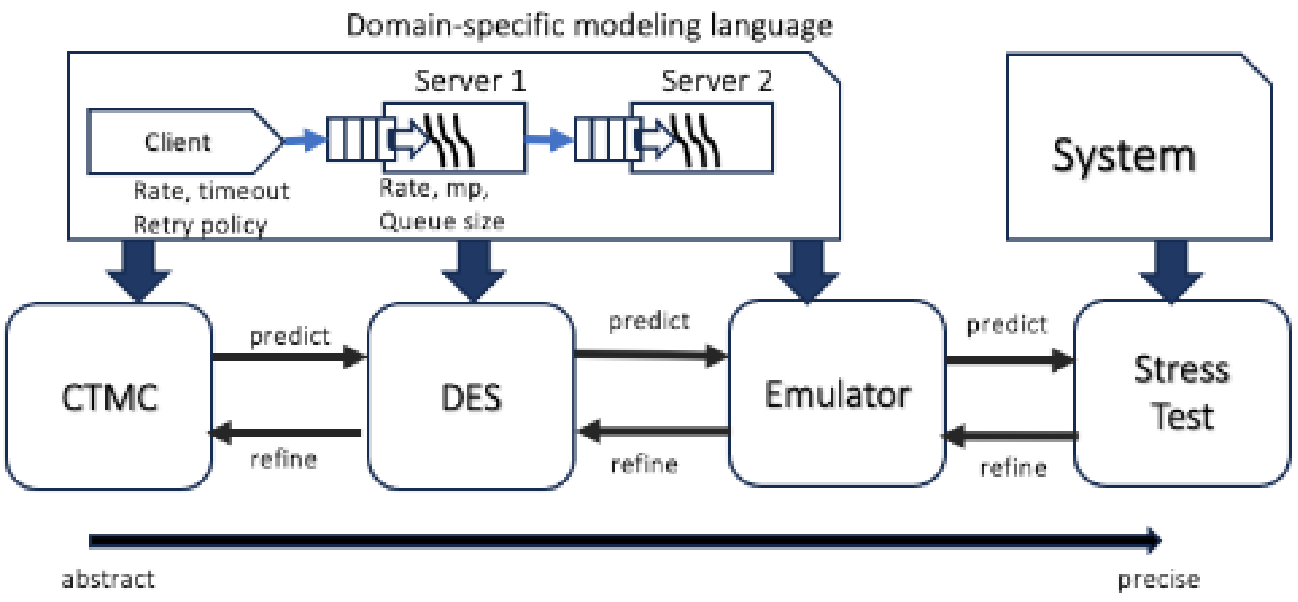
Key Benefit: Cost of analyzing with abstractions is lower!



ANALYZING WITHOUT EXECUTING

Connect the tools at different abstractions to maintain precision

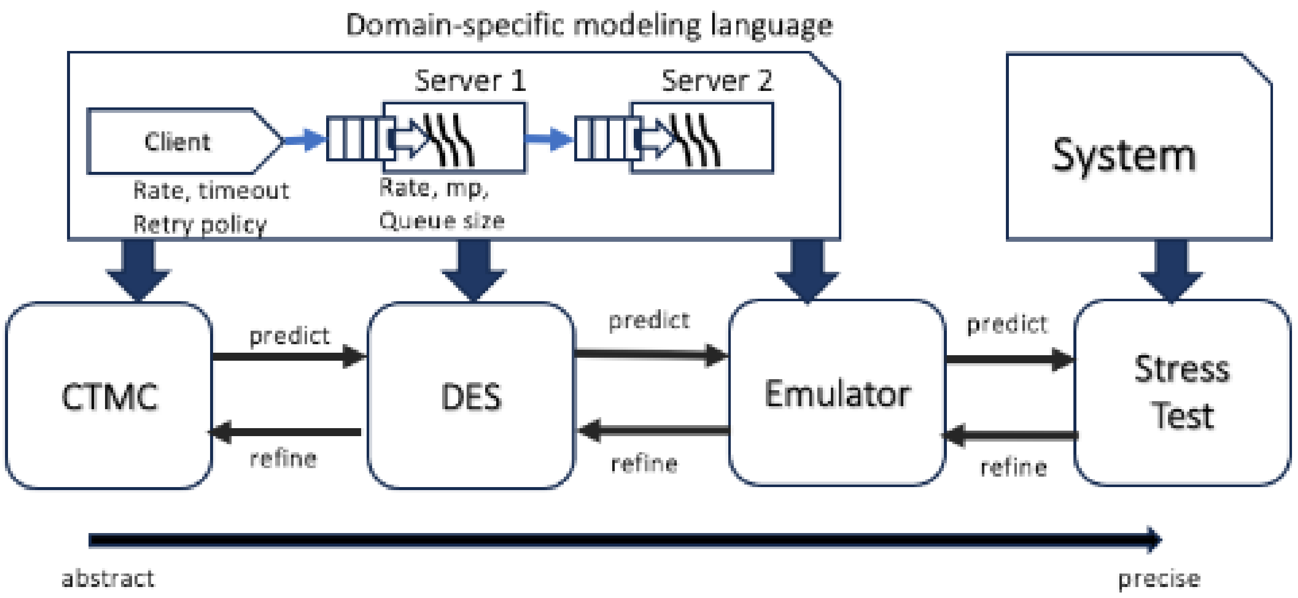
Key Benefit: Cost of analyzing with abstractions is lower!



ANALYZING WITHOUT EXECUTING

Key Benefit: Cost of analyzing with abstractions is lower!

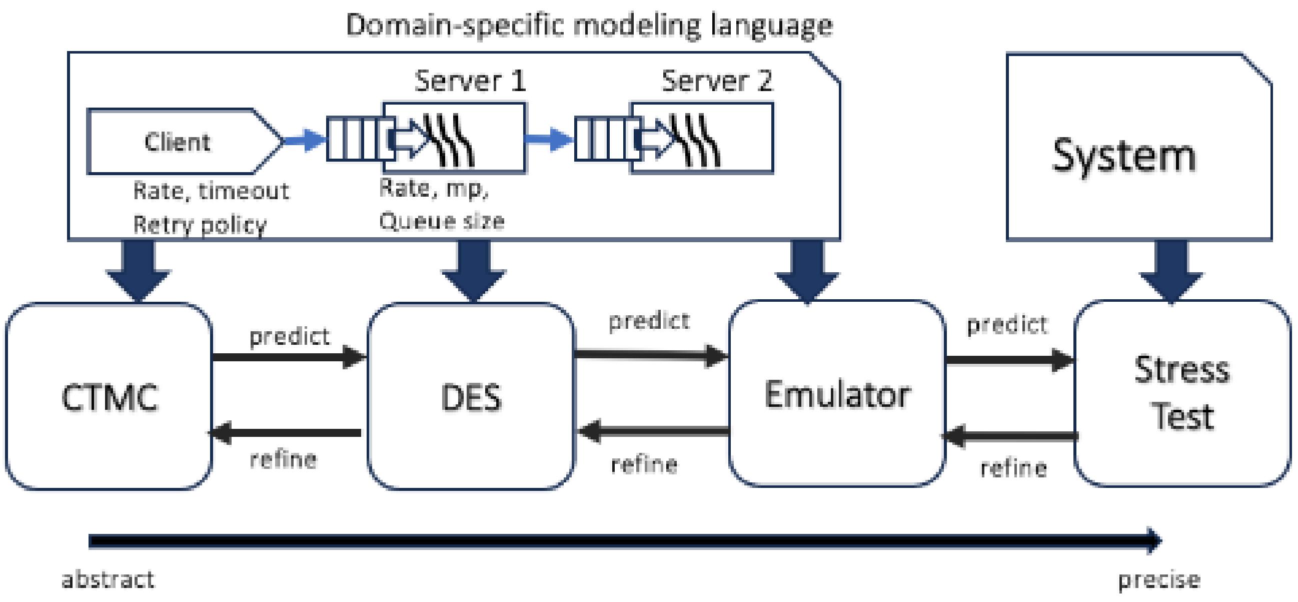
Predictions from abstract models direct experiments to specific parameters



ANALYZING WITHOUT EXECUTING

Predictions from abstract models direct experiments to specific parameters

Key Benefit: No need for large parameter sweeps



4 DIFFERENT MODELS

- CTMC: Continuous-Time Markov Chains
- DES: Discrete Event Simulator
- Emulation
- Stress Testing

CTMC

- Model the system as a CTMC (Continuous-Time Markov Chain)

CTMC

- Model the system as a CTMC (Continuous-Time Markov Chain)
- CTMC Basics:
  - System can be in 1 of many finite states
  - System transitions from 1 state to another with some transition probability
  - Markov Property holds: The next state of the system *only* depends on the current state of the system (and none of the previous states)

CTMC

- Model the system as a CTMC (Continuous-Time Markov Chain)
- Key factors:
  - Request rate from clients
  - Number of retries per request, timeouts
  - Length of the queue at any service, max size of the queue
  - Number of requests waiting to be retried (how we do workload amplification)

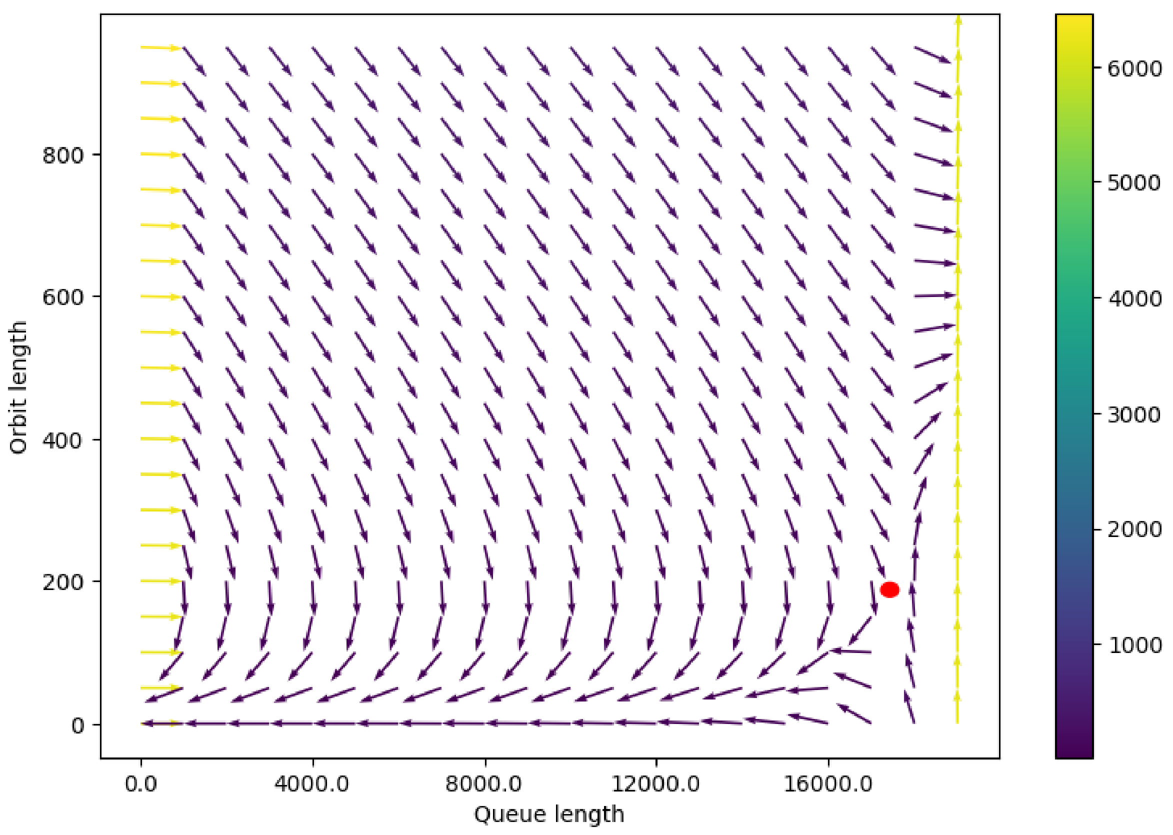
CTMC - DSL

```python
def program():
  api = { 'insert': Work(10, [],) }
  server = Server('simple', api,
                  qsize=150, thread_pool=1)
  src = Source('client', 'insert', rate=5,
                  timeout=5, retries=5)
  p = Program('SimpleService')
  return p.add_server(server).add_source(src).
    connect('client', 'simple')
```

CTMC – SAMPLE ANALYSIS

Red dot shows where the metastable failure will happen

Queue size: 20000
Orbit length: 50



DISCRETE EVENT SIMULATION

- Replace some of the mathematical abstractions with actual implementations of the core abstractions

- Use a discrete-event simulator to simulate the behavior of the system
  - Simulates one of the chosen pathways in the CTMC
  - Allows white-box testing

EMULATION

- Add real-world factors not modeled by the simulation
- Includes resource contention, load balancing, rpc framework
- Bare-bones service that does no work but sleeps for a specific amount of time chosen from a distribution

STRESS TESTING

- Run a workload along with triggers against the actual deployed system
- This is essentially executing a targeted subset of the original parameter sweeps
- This provides the necessary confirmation of the predictions!
  - Without this confirmation, predictions are not actionable

ROOT CAUSE ANALYSIS

MAJOR STAGES OF AN INCIDENT

MAJOR STAGES OF AN INCIDENT

- **Detection:** Incident is either detected by a service monitor or reported by a customer

MAJOR STAGES OF AN INCIDENT

- **Detection:** Incident is either detected by a service monitor or reported by a customer

- **Root Cause Analysis:** Investigation by an On-Call Engineer to identify the root cause

MAJOR STAGES OF AN INCIDENT

- **Detection:** Incident is either detected by a service monitor or reported by a customer

- **Root Cause Analysis:** Investigation by an On-Call Engineer to identify the root cause

- **Mitigation:** Executing steps to *temporarily* fix the impact of the incident

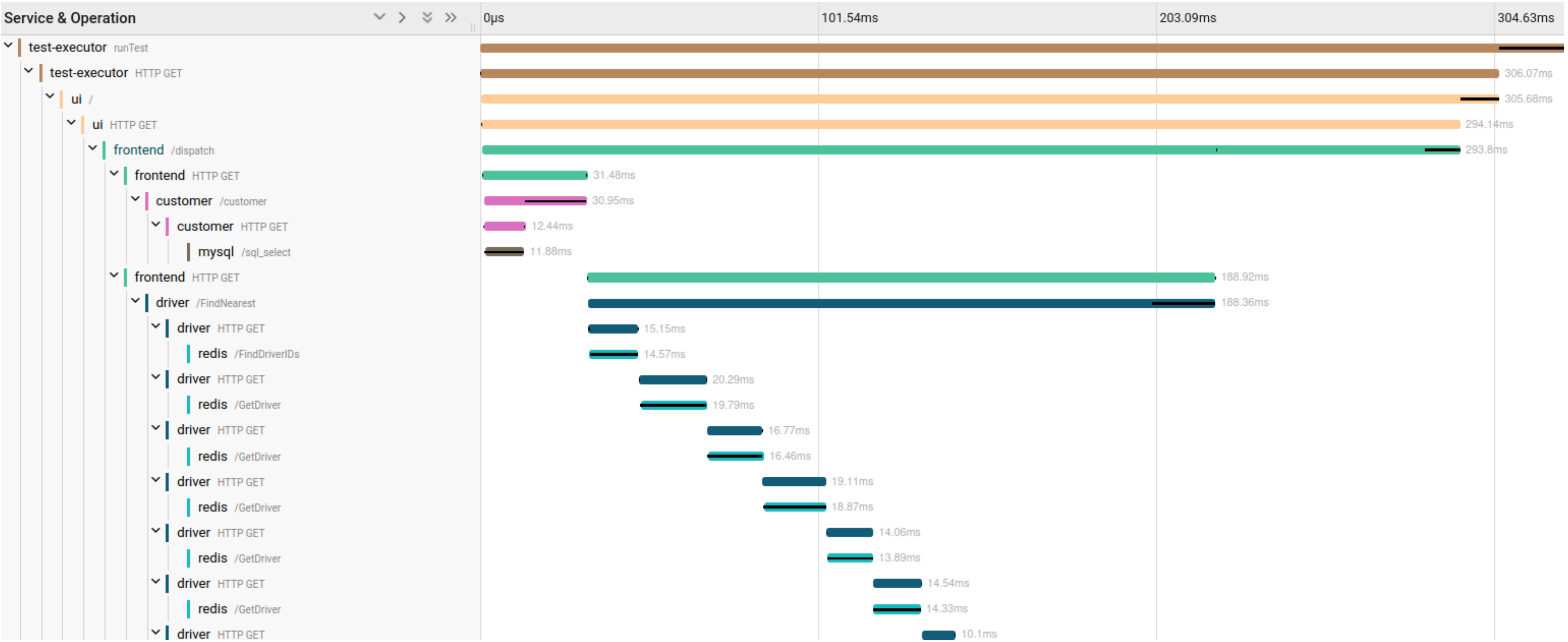
PERMANENT FIXES

Permanent Fixes requires more in-depth root cause analysis

- ❖ During incident management the main thing is to quickly mitigate the impact
- ❖ Fixing the underlying issue requires more complex analyses

VISUALIZATION TOOLS AND DASHBOARDS

SINGLE TRACE VIEW



Jaeger Trace View

SINGLE TRACE VIEW + AGGREGATE DATA

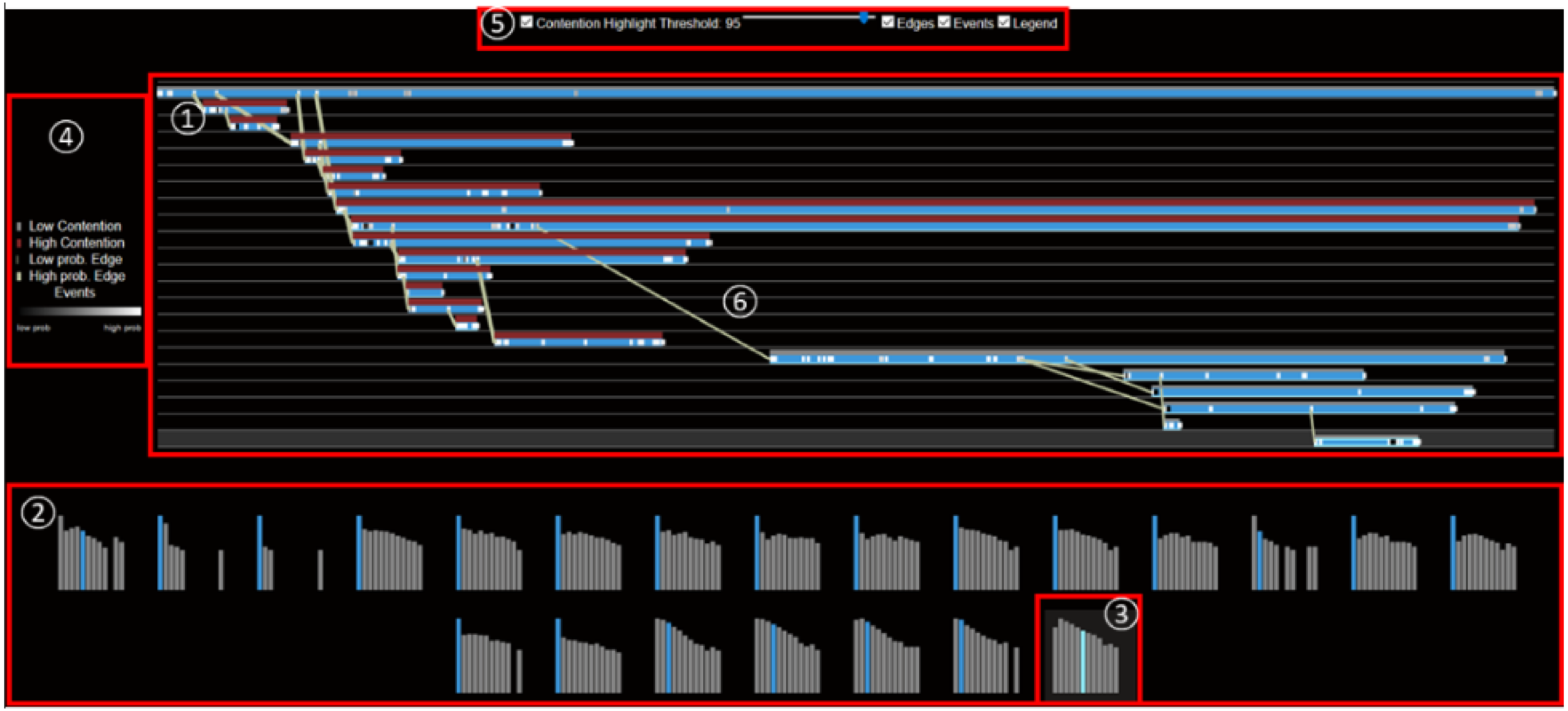
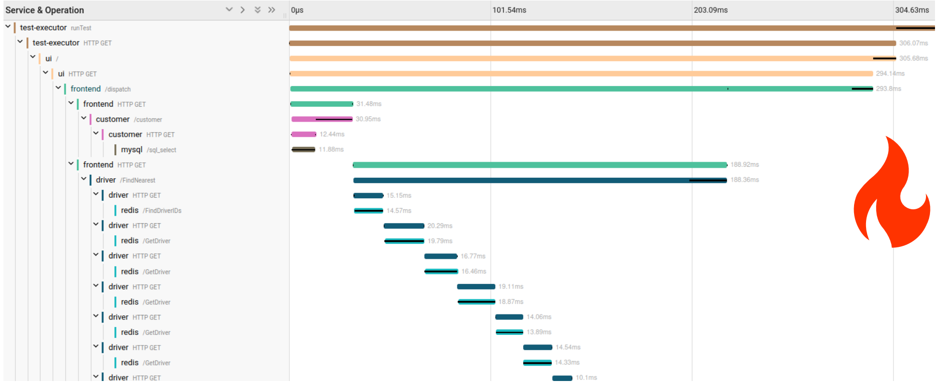


Figure 2: TraVista's Gantt chart visualization extended with aggregate data.

TRACE COMPARISONS

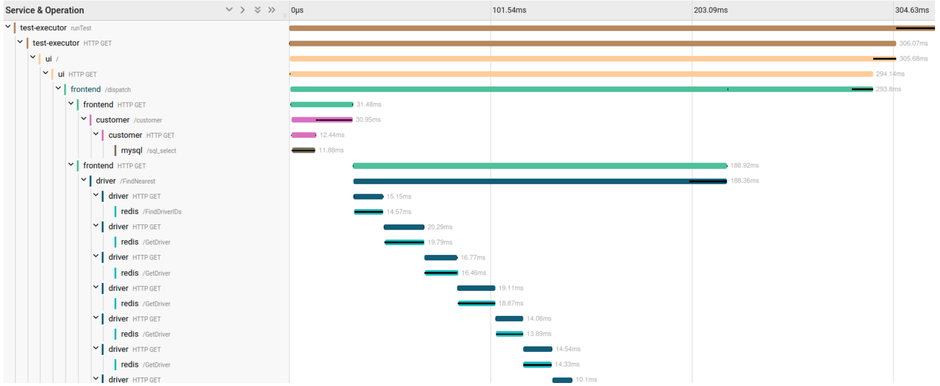
BUGGY TRACES DEVIATE FROM NORMAL TRACES

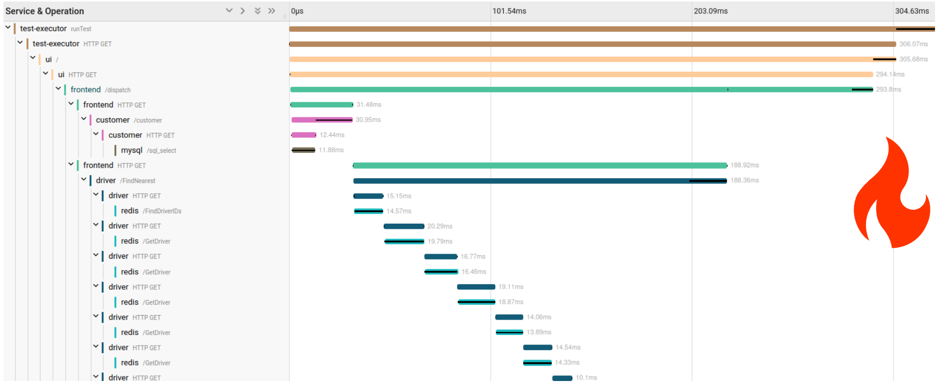


Performance and Correctness issues manifest as mutations in trace execution timings and structure

- ❖ The bug may cause extra operation(s) to happen
- ❖ The bug may prevent certain operation(s) from executing
- ❖ The bug may increase the timing of operation(s)

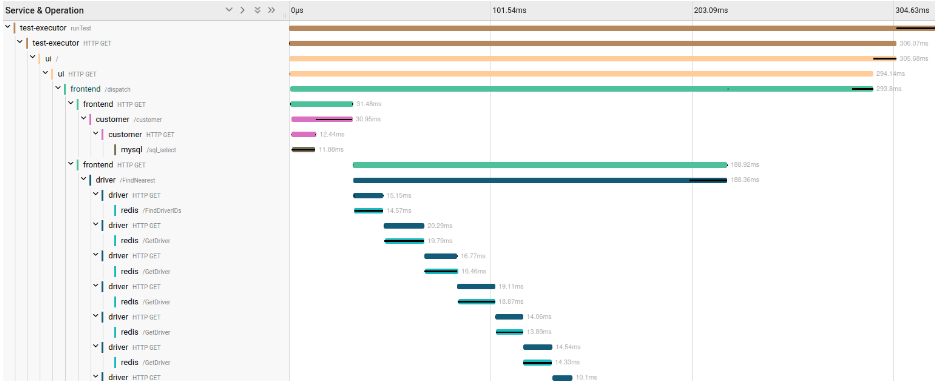
KEY DEBUGGING TASK: COMPARING TWO TRACES

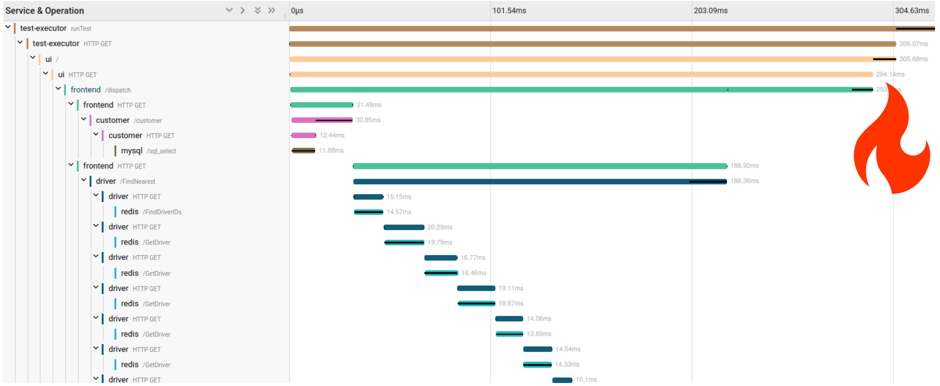




To identify probable root cause(s), developers must compare the buggy trace with a normal behaviour trace

COMPARING TWO TRACES IS NON-TRIVIAL





Traces can be too big

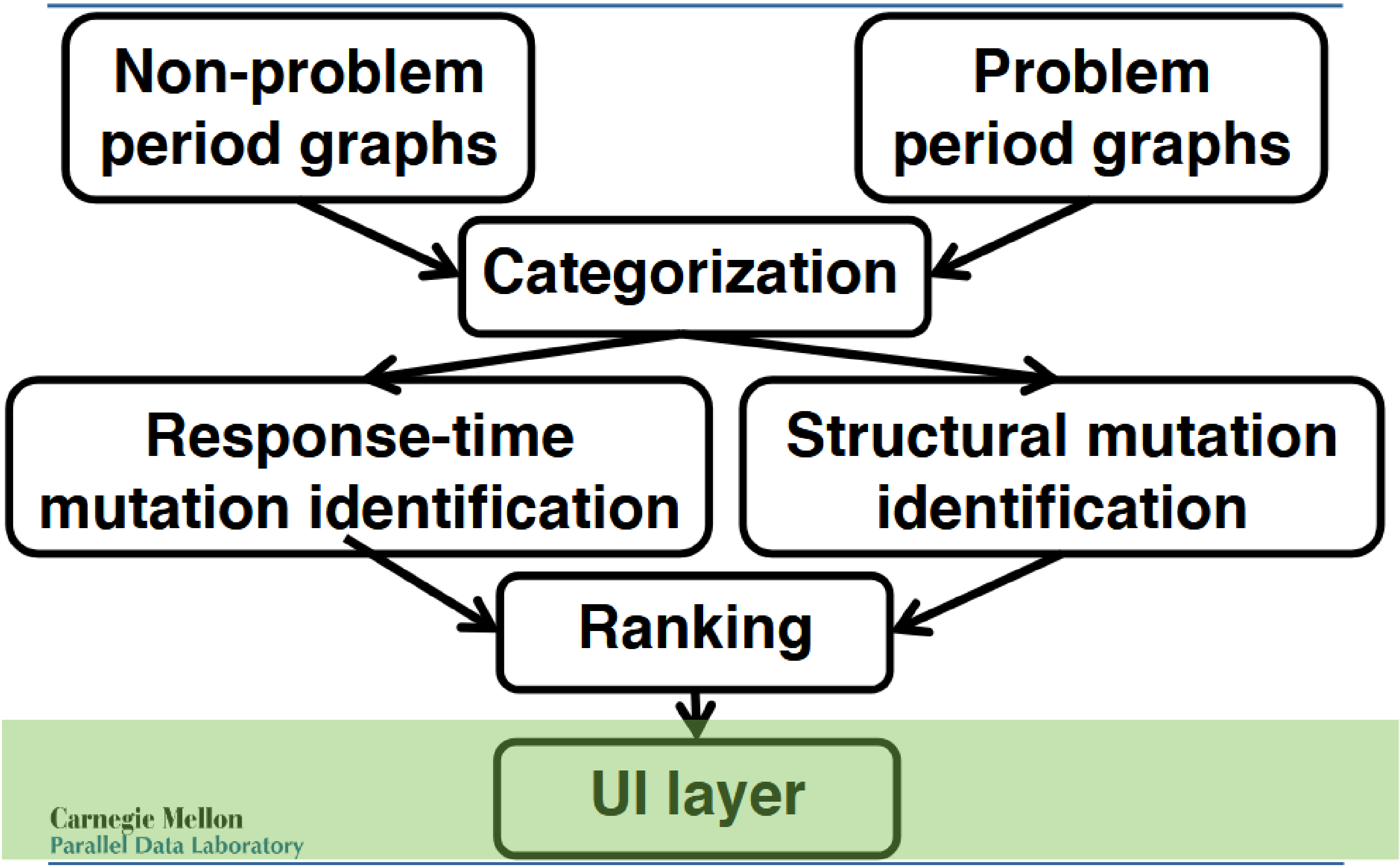
- ❖ Thousands of spans
- ❖ Difficult to compare changes visually

Two traces can differ in small but important ways

- ❖ Non-trivial for users to find these small changes

COMPARING REQUEST FLOWS

SPECTROSCOPE WORKFLOW



COMPARING REQUEST FLOWS

Spectroscope is a tool for comparing two request flows

- Converts each request flow into a single string
- Categorizes each request flow and collects statistics for each category
- Comparisons are done within a category
- Provides heuristics for identifying mutations, precursors, and for ranking them

CRITICAL PATH ANALYSIS

CRITICAL PATH ANALYSIS

In this approach, we find the critical path of the request through the whole execution trace.

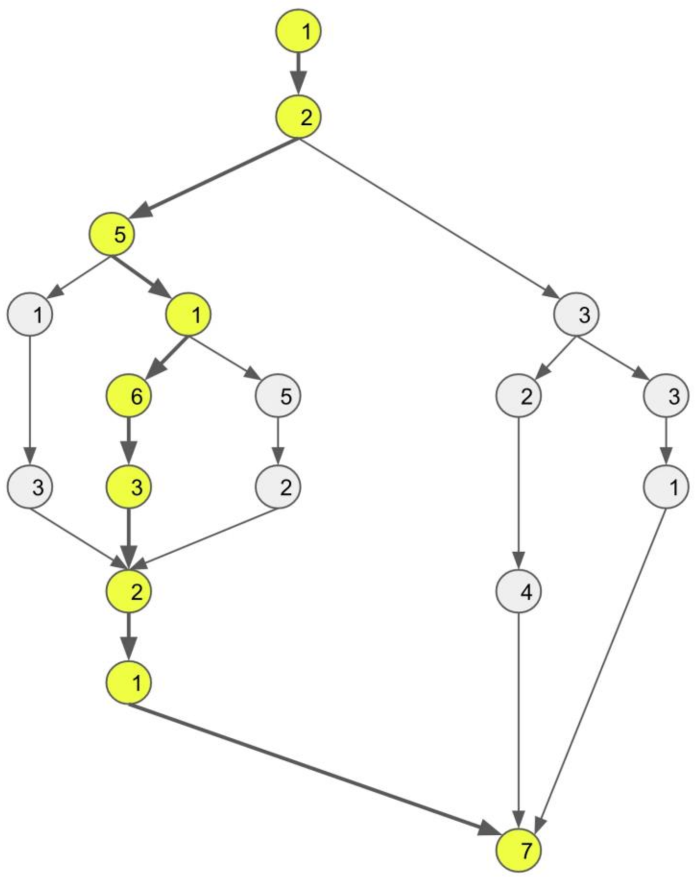
Critical Path is the longest sequence of dependent operations or spans that determines the total end-to-end latency of a request.

CRITICAL PATH - BENEFITS

- Reduces the amount of spans to consider to the key spans that contributed the latency of the request

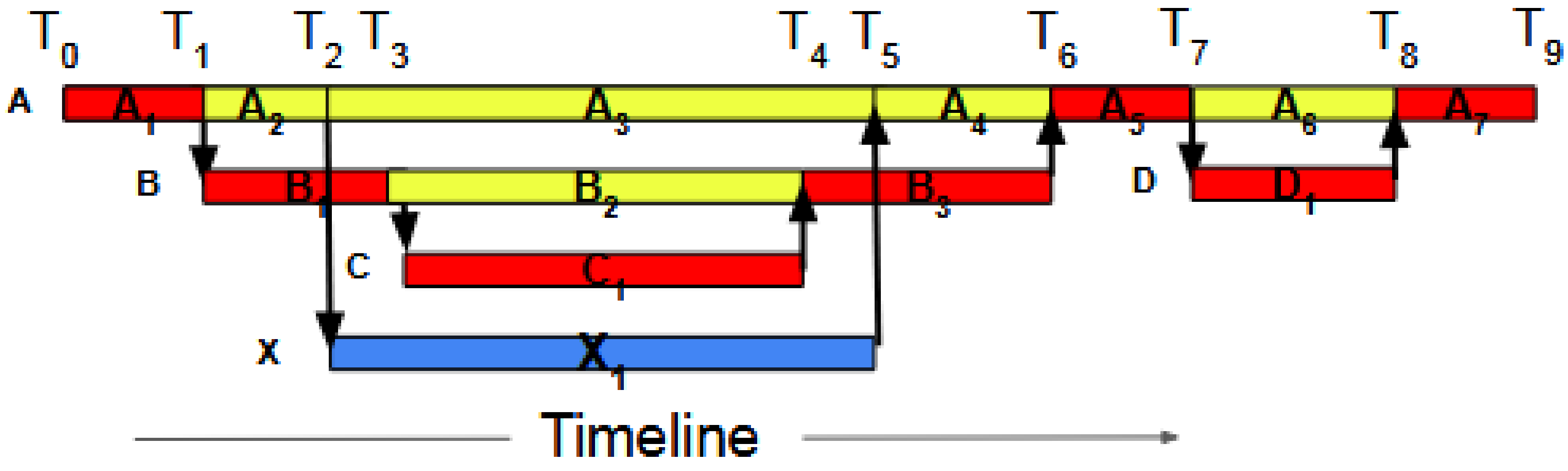
- Easier to understand as compared to full traces

CRITICAL PATH - EXAMPLE



In this execution graph, the path highlighted in yellow is the critical path for the request

CRITICAL PATH - EXAMPLE

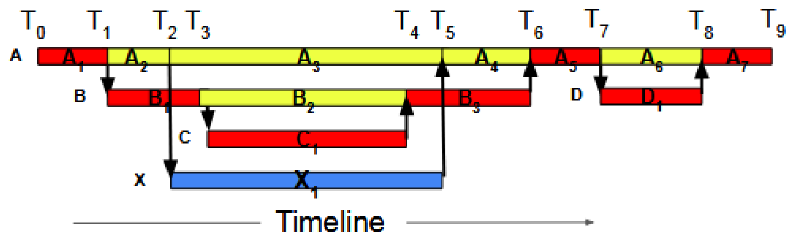


CRISP: Critical Path Analysis of Large-Scale Microservice Architectures, July 2022, Zhang et al

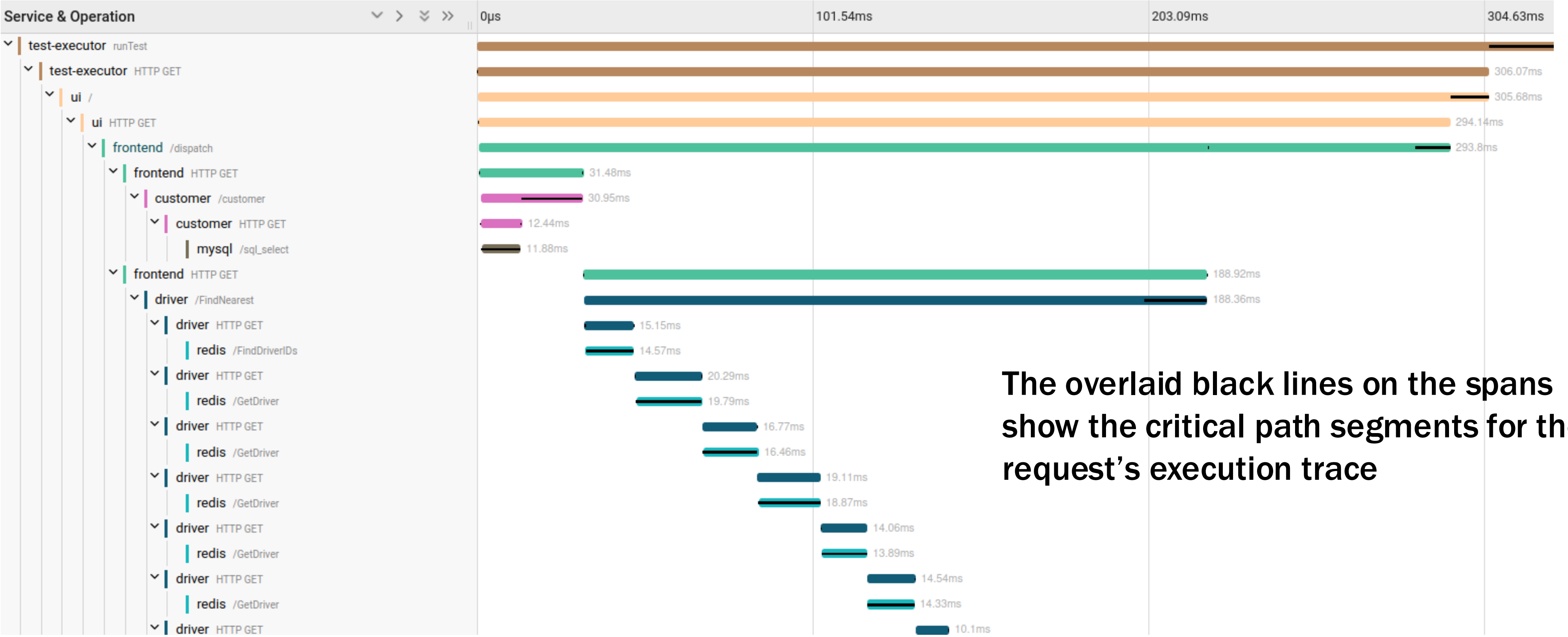
CRITICAL PATH - ALGORITHM

```
def CP(root):
  path = [root]
  if len(root.child) == 0:
    return path
  children = sortDescendingByEndTime(root.children)
  lfc = children[0]
  path.extend(CP(lfc))
  for c in children[1:]:
    if happensBefore(c, lfc):
      path.extend(CP(c))
      lfc = c
  return path
```

Listing 1: Pseudocode to compute critical path.



CRITICAL PATH IN ACTION

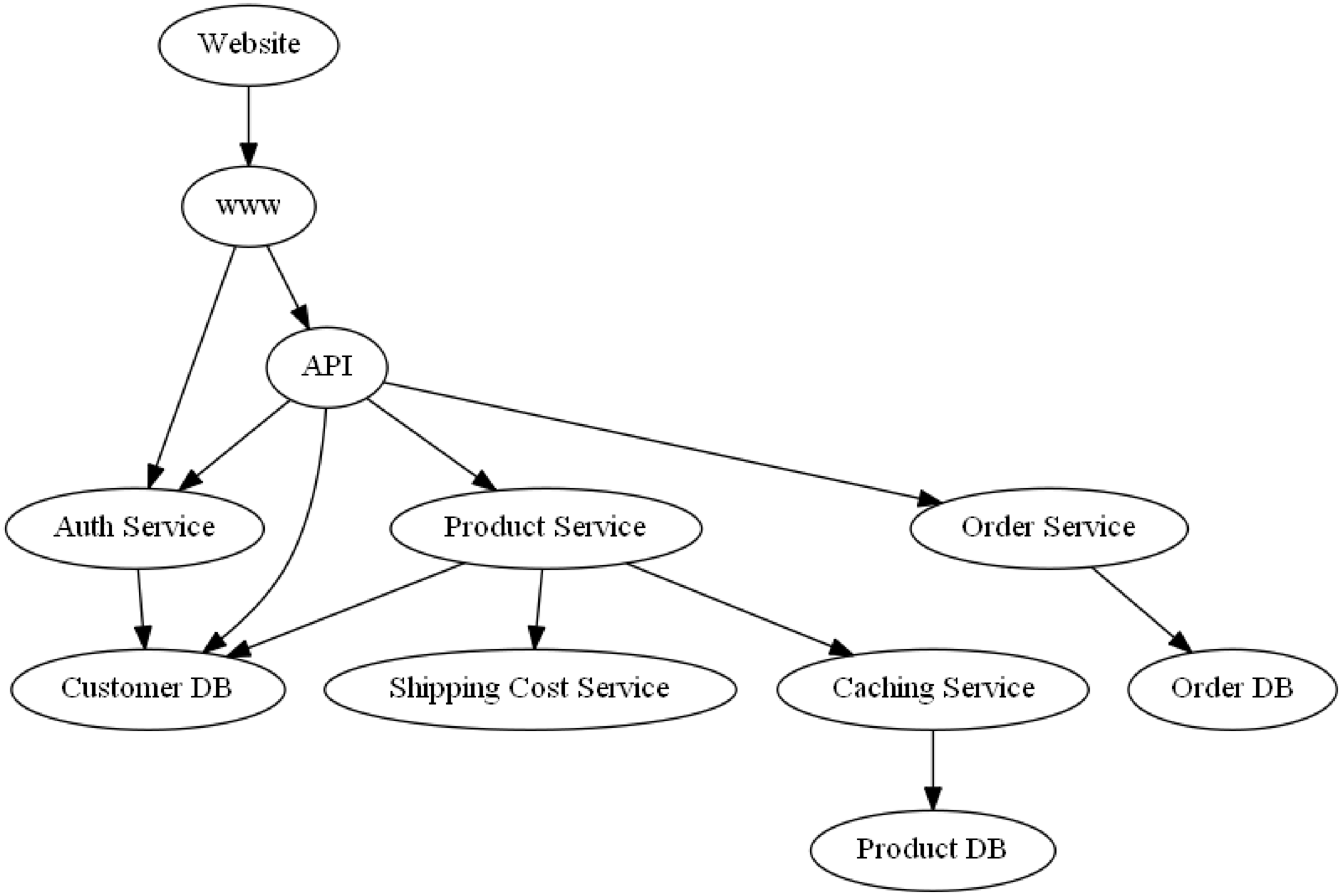


The overlaid black lines on the spans show the critical path segments for the request's execution trace

CAUSAL INFERENCE

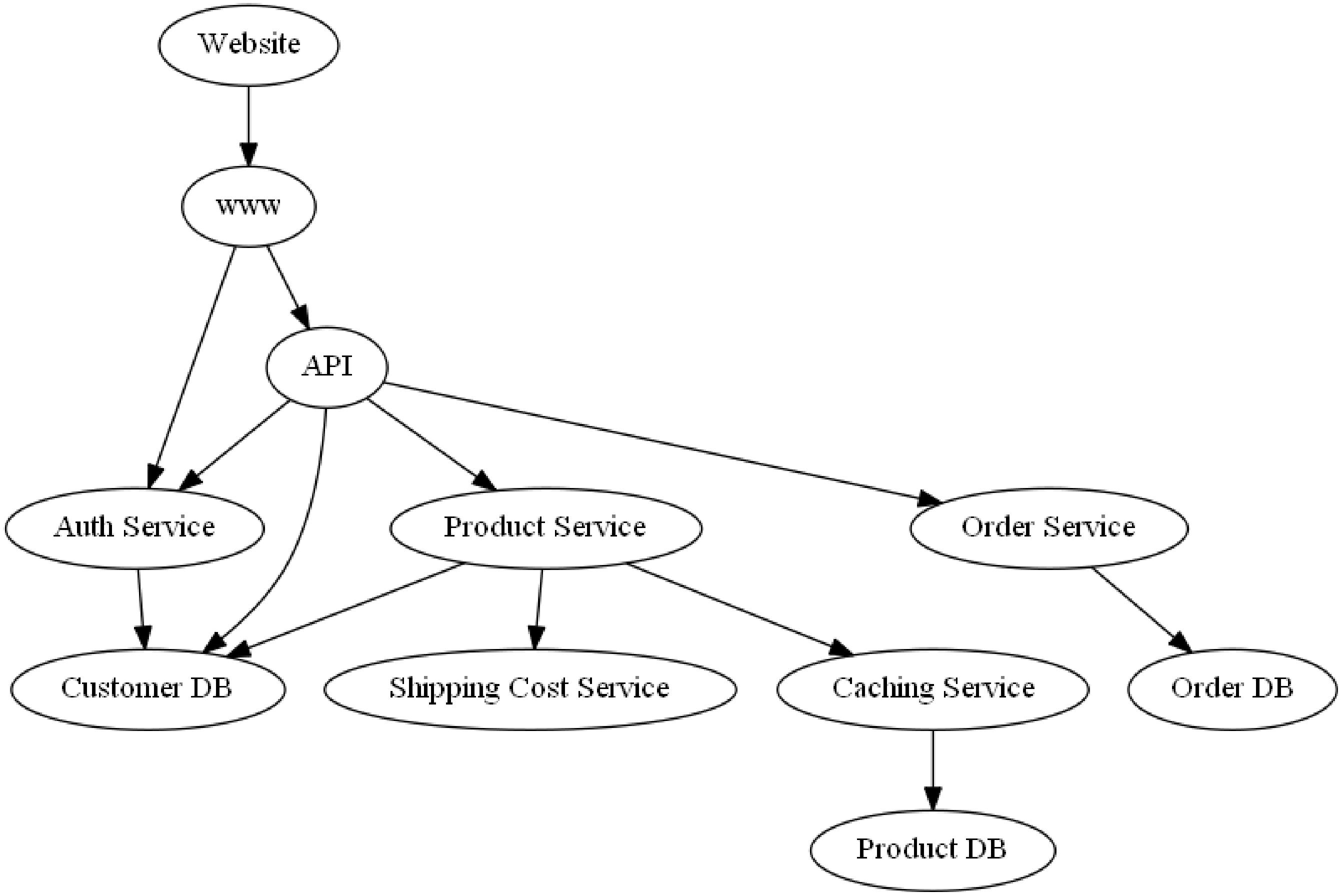
The key idea: Use causal inference and reasoning to figure out the root causes of unexpected observed latencies (or some other metric of interest)

CAUSAL INFERENCE



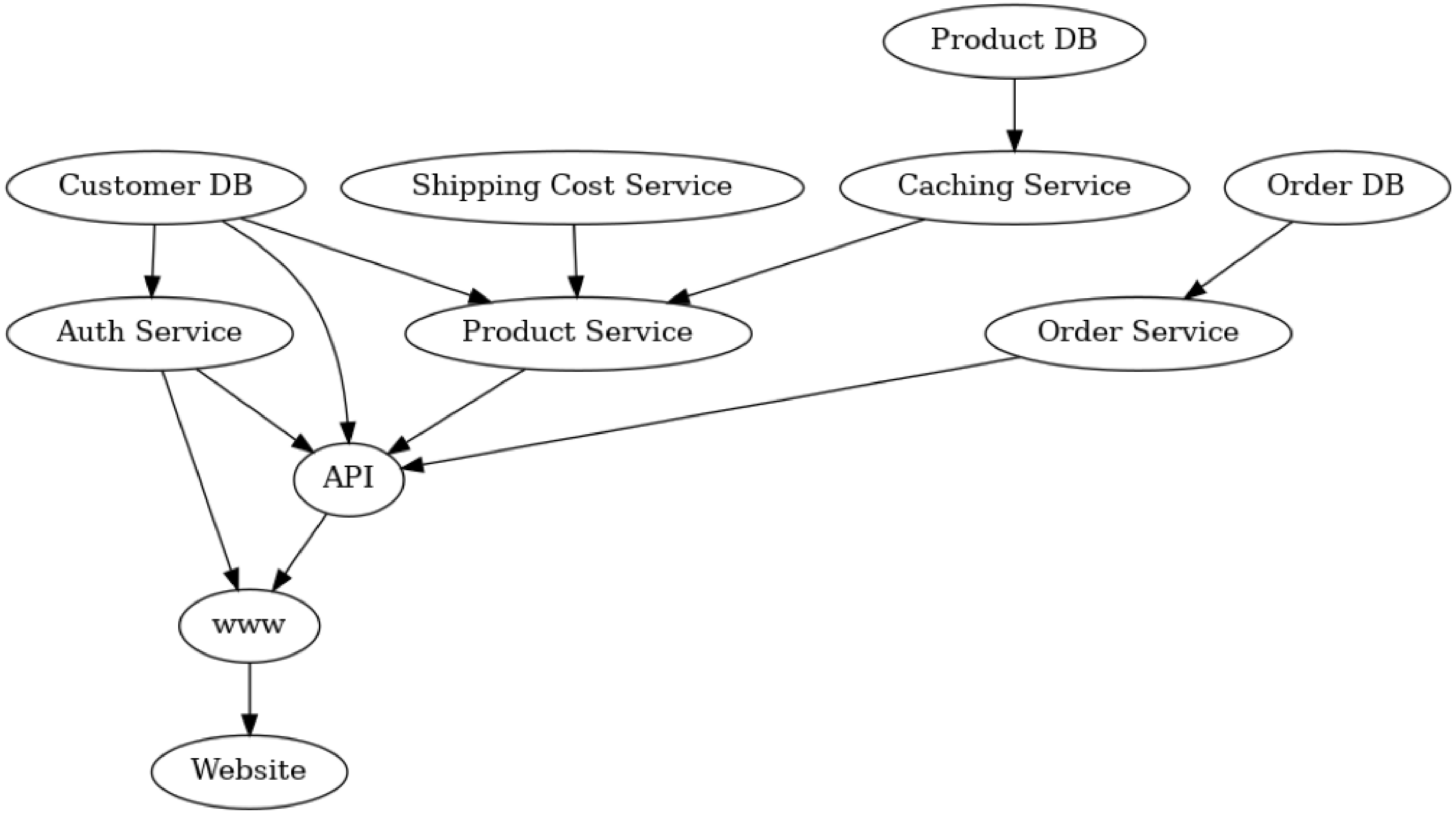
Root cause analysis (RCA) of latencies in a microservice architecture, dowhy Documentation

STEP 1: BUILDING A CAUSAL GRAPH



Root cause analysis (RCA) of latencies in a microservice architecture, dowhy Documentation

STEP 1: BUILDING A CAUSAL GRAPH

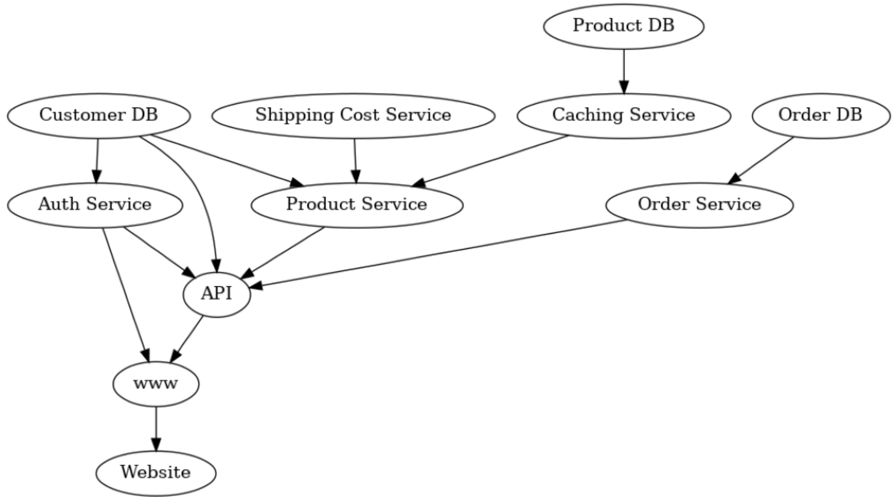


Root cause analysis (RCA) of latencies in a microservice architecture, dowhy Documentation

STEP 2: SETTING UP CAUSAL EQUATIONS

Caching_service_latency = product_db_latency * A + Constant

Product_service_latency = Customer_db_latency * B + Shipping_cost_service_latency * C + Caching_service_latency * D + Constant

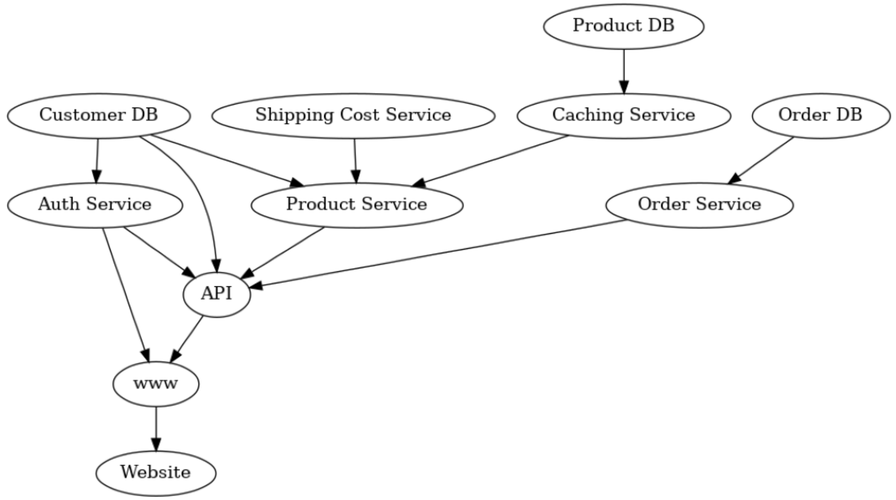


STEP 2: SETTING UP CAUSAL EQUATIONS

Execution order defines how to combine latencies

Sequential calls can be combined by simply adding latencies

Concurrent calls can be combined by taking the max



STEP 3: FIT THE MODEL

- Use collected metrics data to fit the model at each node
- Each request is essentially a row in the data used to "train" the model at every node
- This basically finds the values of the coefficients for each value in the causal equation

STEP 4: USE THE MODEL TO ATTRIBUTE OUTLIER SCORES

- Provide the outlier data and plug it in to the model to see which service is attributing the most for the outlier behavior

- Higher the attribution score for a service, the more it impacts to the request being an outlier

CAUSAL INFERENCE

- There are a lot of methods for scoring as well as for constructing the causal graph automatically from collected data

- This is still a progressing field and we do not have an answer to what is the best method and how well does this work

DISCUSSION THEMES

- What is the best method for doing root cause analysis?
- What are the challenges for extracting critical paths?
- How do we efficiently compare two traces?
  - How do we compare 1 trace to a group of traces?
  - How do we compare two different groups of traces?