AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

COP343

Observability best practices at Amazon

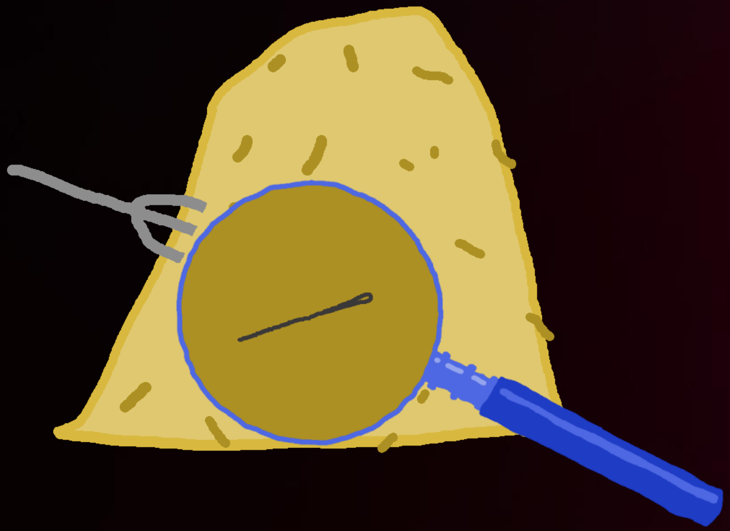
David Yanacek (he/him)

Sr. Principal Engineer
AWS Monitoring & Observability

Ian McGarry (he/him)

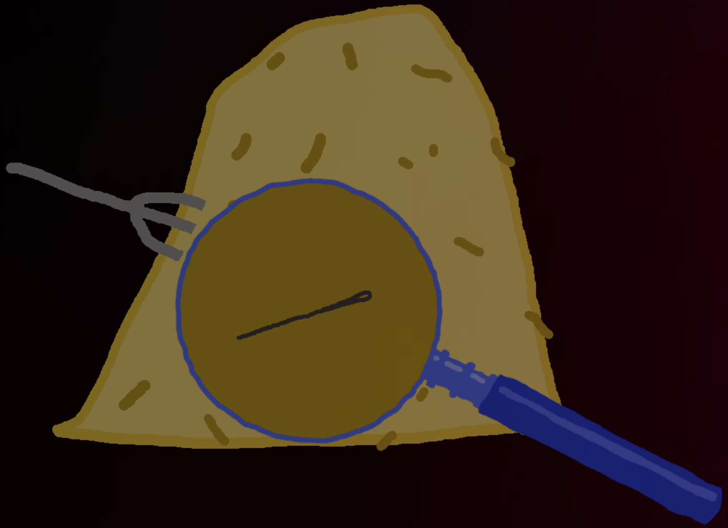
Director of Engineering
AWS Monitoring & Observability

Table of contents



Chapter one

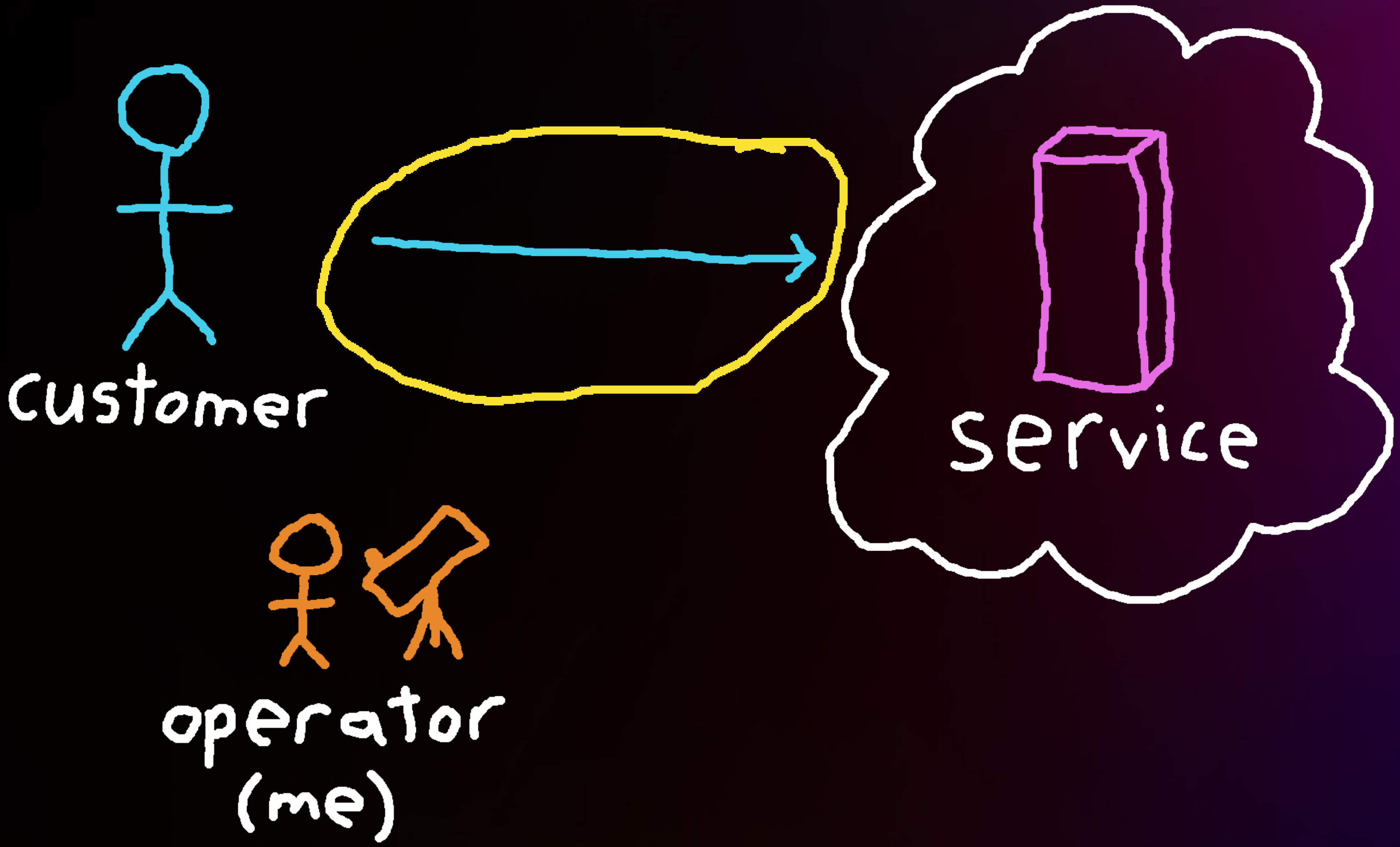
The DevOps
flywheel at Amazon



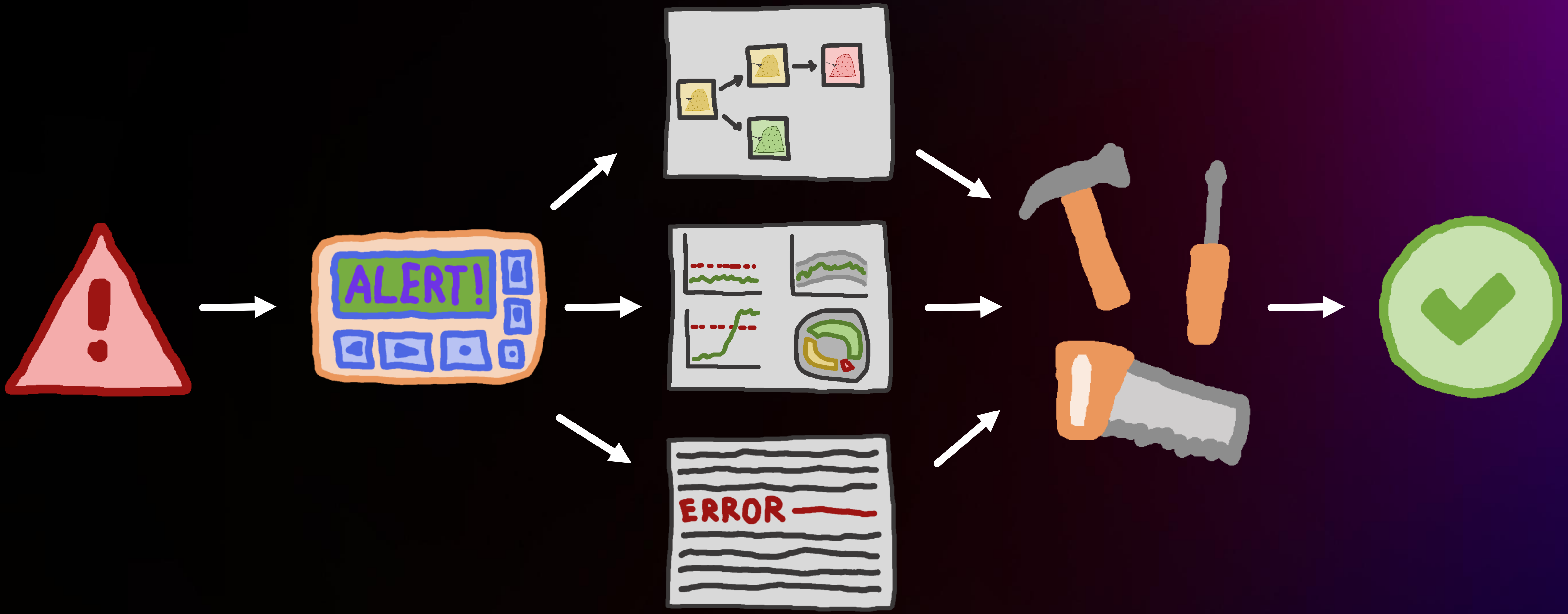
Find the
root cause

Measure from
everywhere

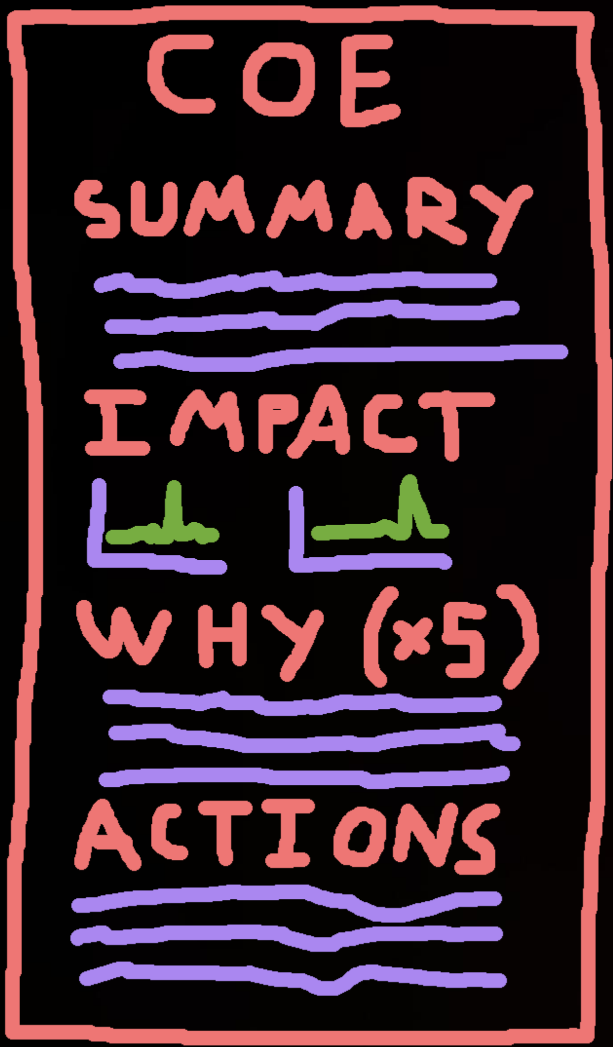
Observing customer experience



Responding to incidents

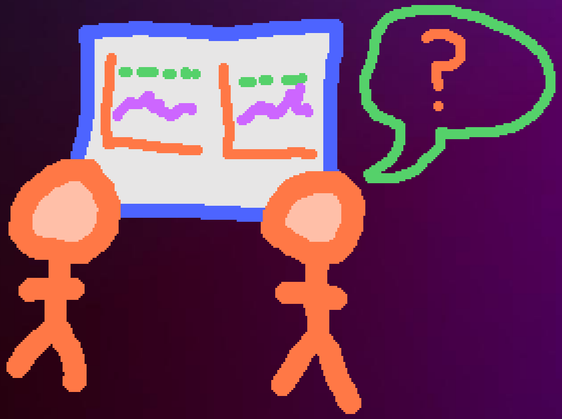


Post-incident retrospectives



```
  ror talking
  Retrying call aft
N]      Connect failed to r
RROR]   Error talking to data
INFO]   Finished with call to
[ERROR] Error talking to databa
[WARN]  Cache miss retrieving
[FATAL] Critical timeout talking
[WARN]  Retry after 5 seconds to
[ERROR] Error talking to databa
WARN]   Waiting 5 seconds befor
NFO]    Making new connection
FO]     Removing connection f
]       Retry after 5 secon
        Error talking to
```

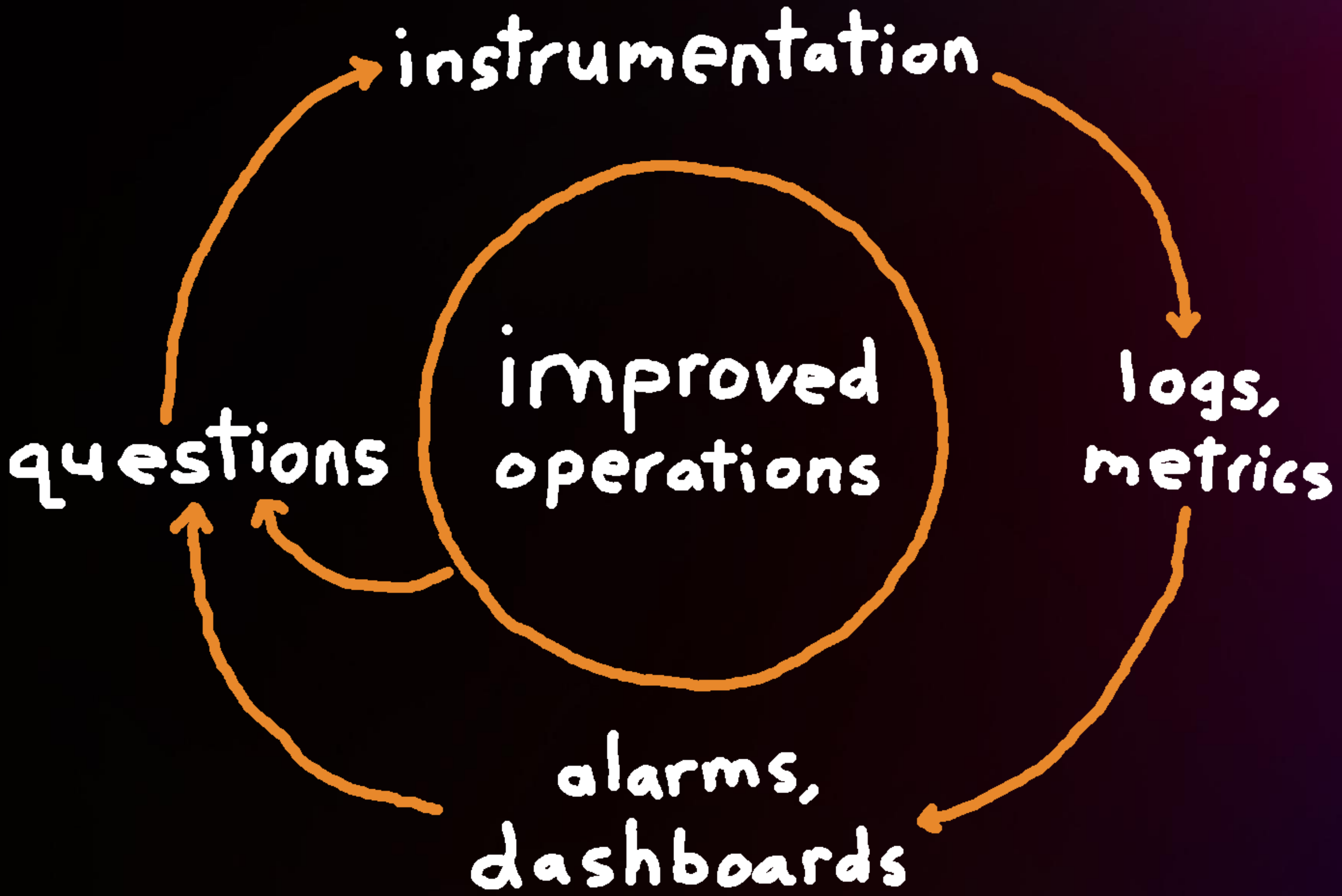
Weekly operations meeting(s)



Observability at scale

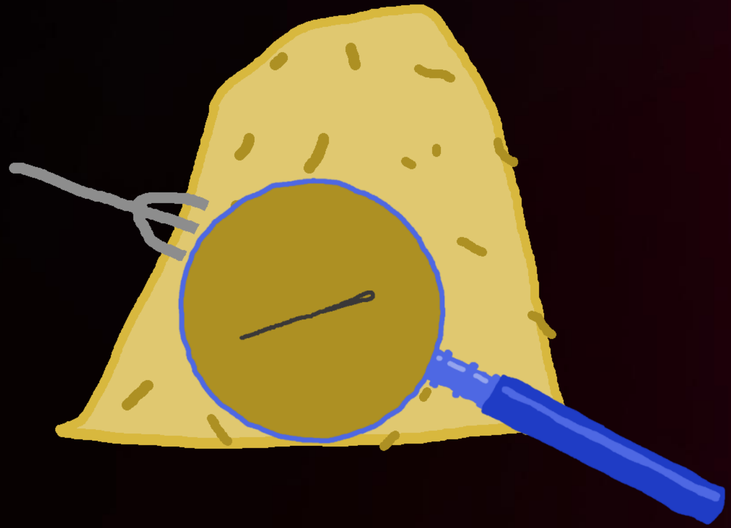
"Amazon CloudWatch is used to monitor more than 9 quadrillion metric observations, ingests more than 5 exabytes of logs per month." *As of November 2022*

The cycle of observability



Chapter two

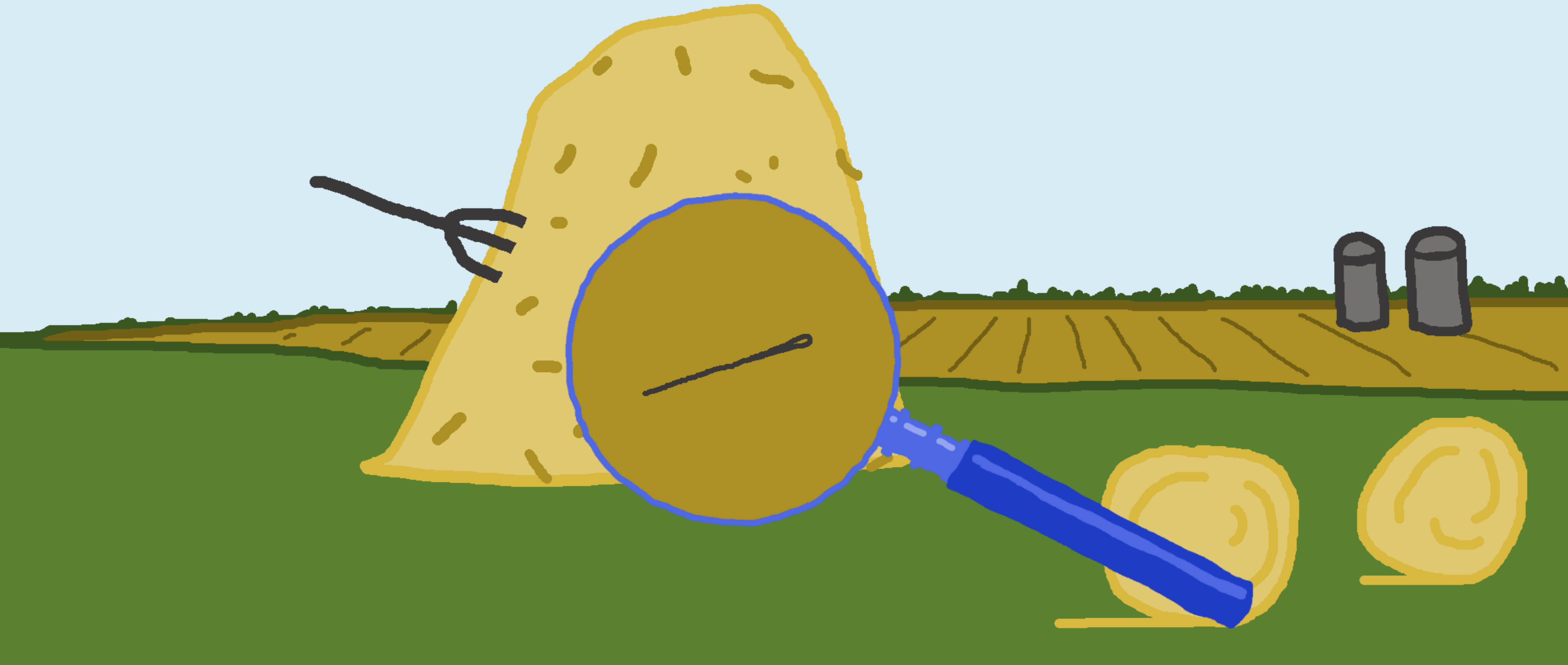
The DevOps flywheel at Amazon



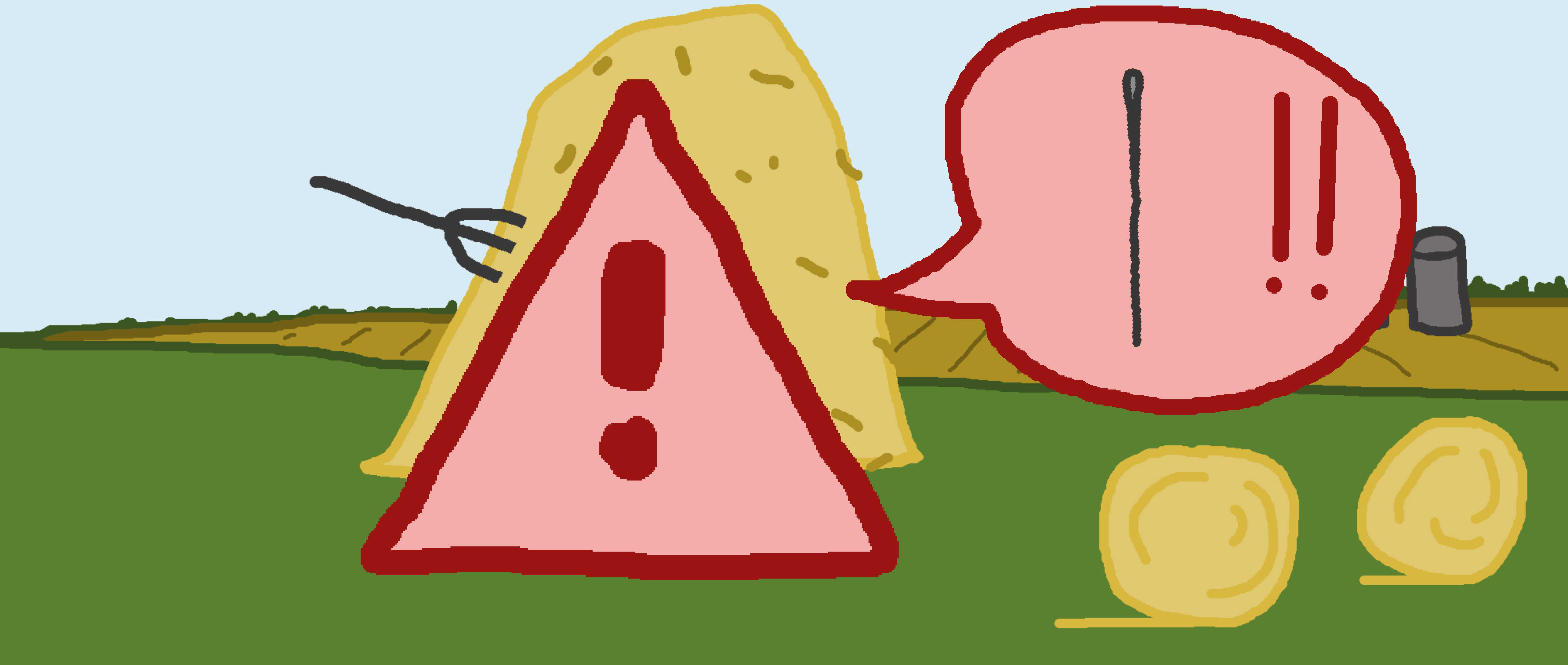
Find the root cause

Measure from everywhere

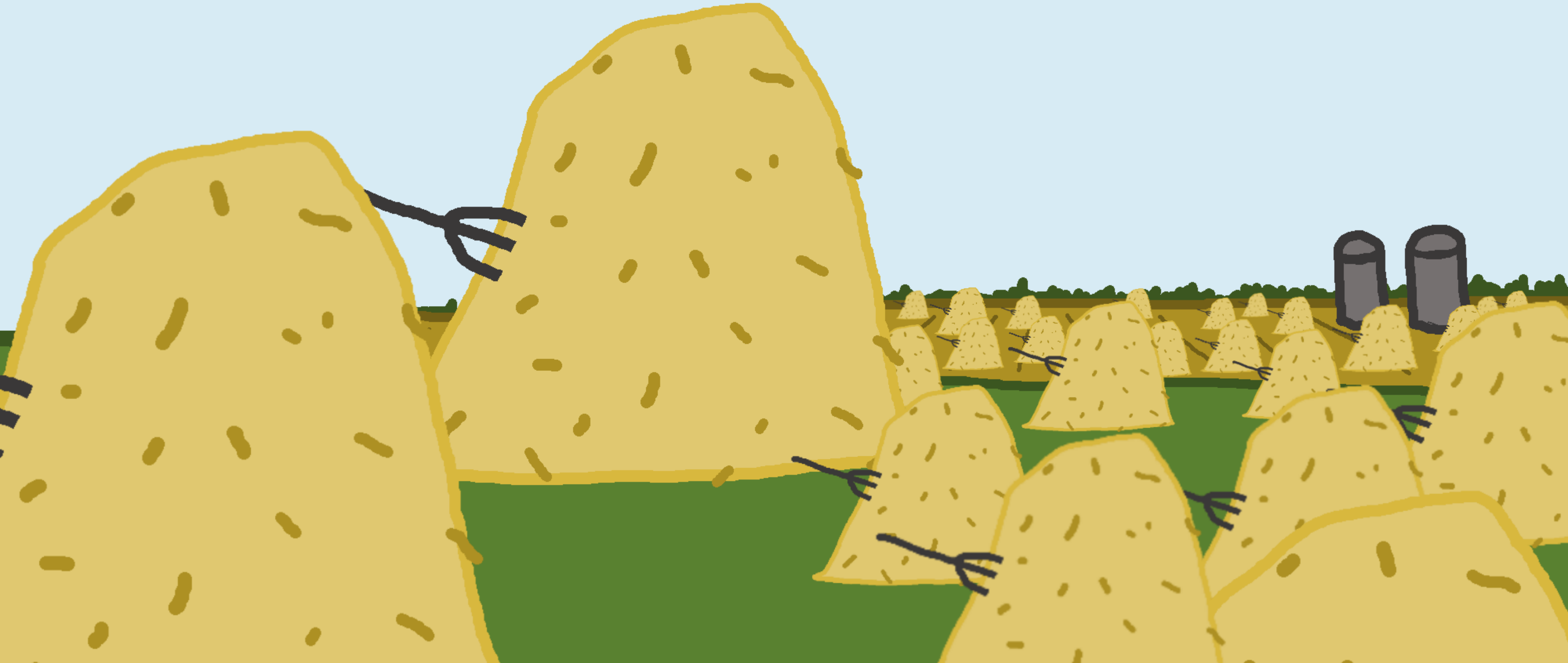
Finding the needle in the haystack



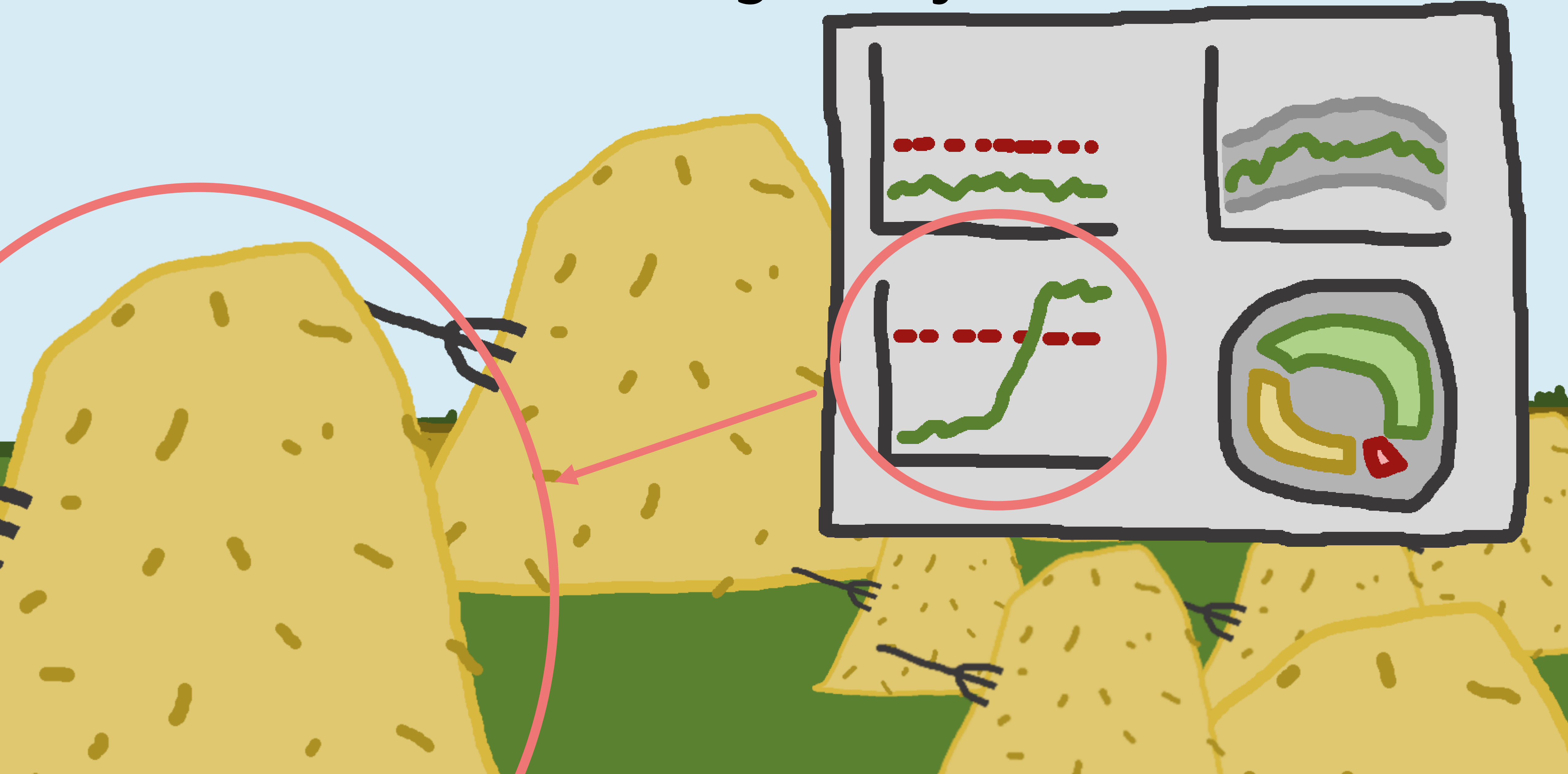
Alarm: Awareness of a needle



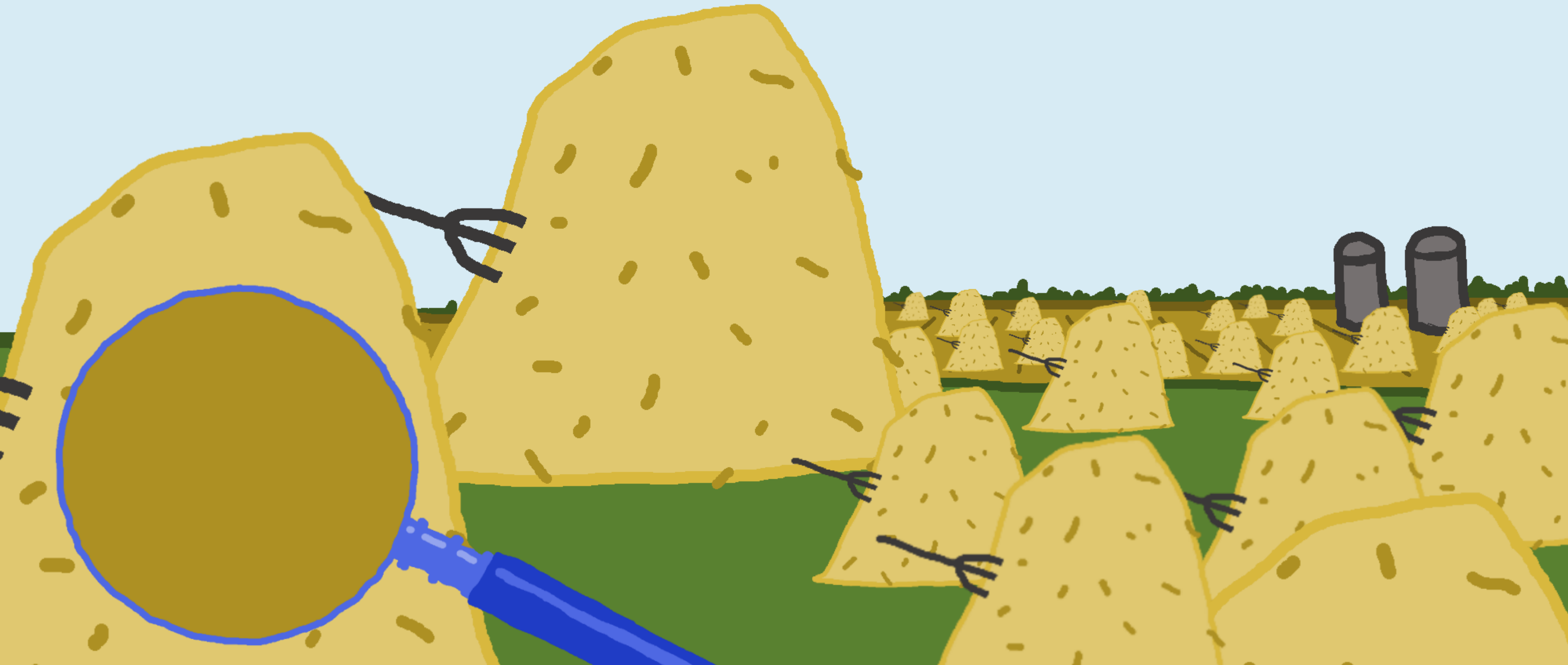
Which haystack?



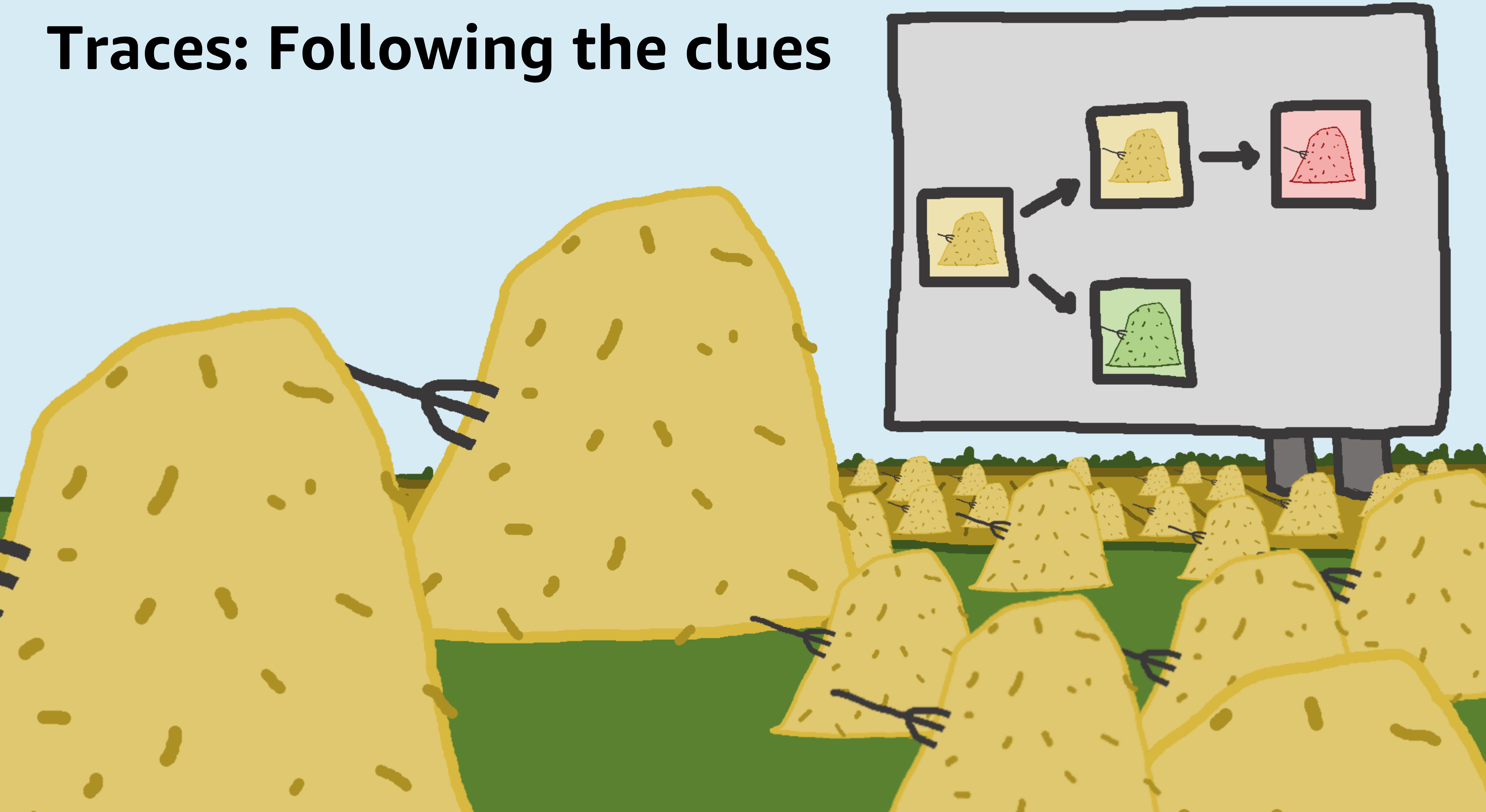
Dashboard: Find the right haystack



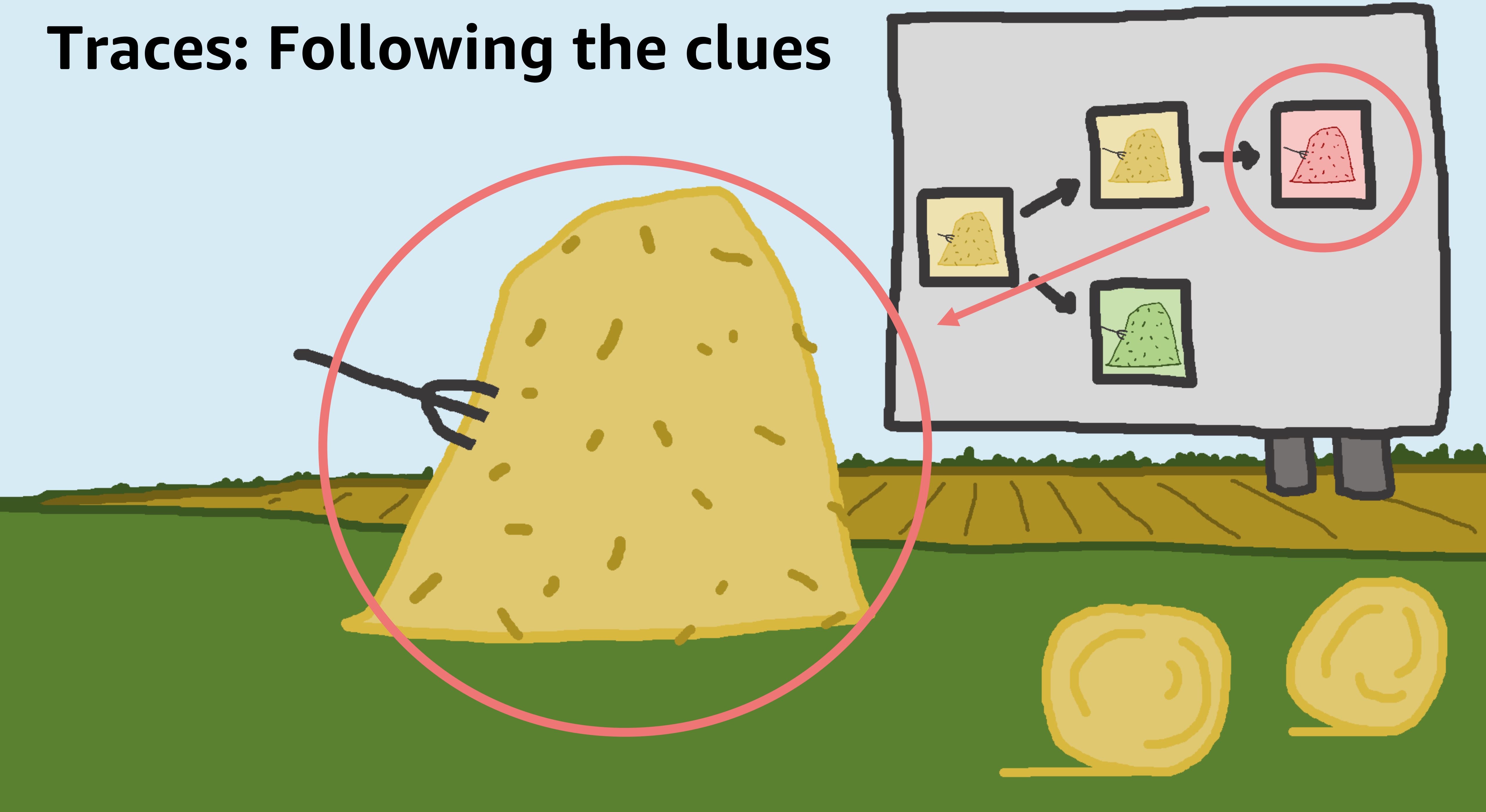
Wrong answer, no needle



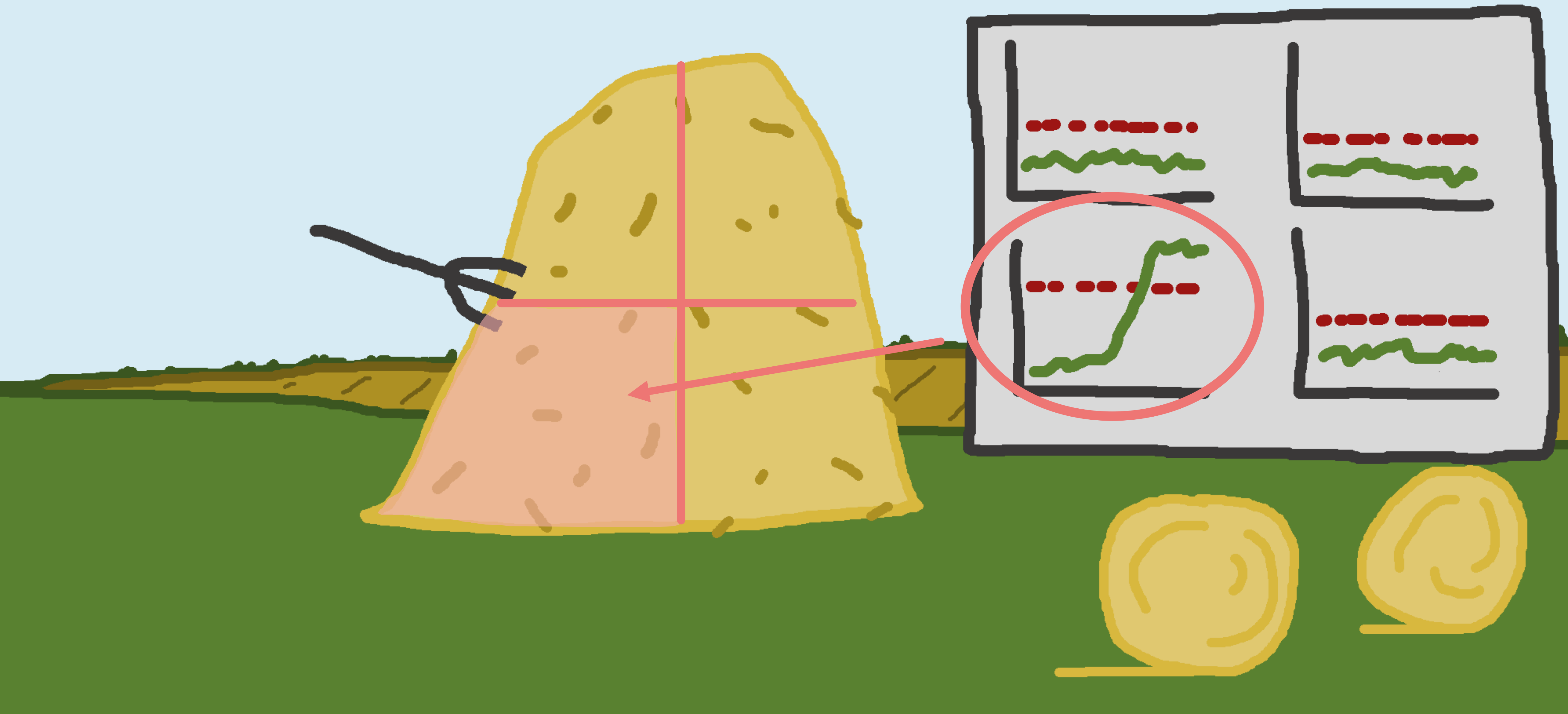
Traces: Following the clues



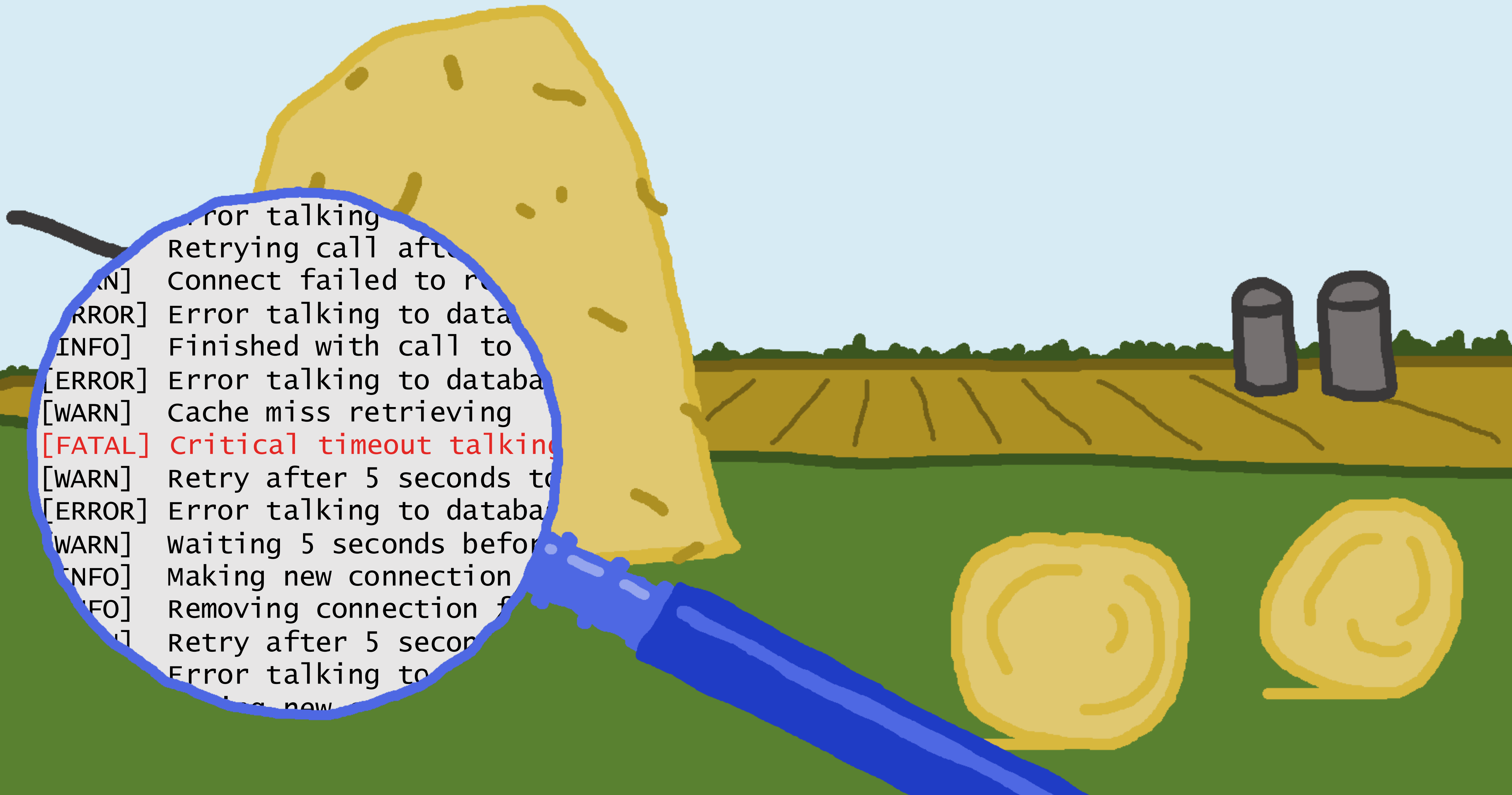
Traces: Following the clues



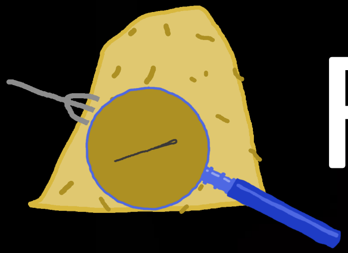
Metrics: Find the part of the haystack



Logs: Find the needle

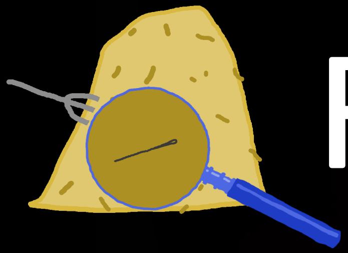


```
        ror talking
        Retrying call aft
    N]  Connect failed to r
 RROR]  Error talking to data
 INFO]  Finished with call to
[ERROR] Error talking to databa
[WARN]  Cache miss retrieving
[FATAL] Critical timeout talking
[WARN]  Retry after 5 seconds to
[ERROR] Error talking to databa
 WARN]  Waiting 5 seconds befor
  NFO]  Making new connection
   FO]  Removing connection
        Retry after 5 secon
        Error talking to
```



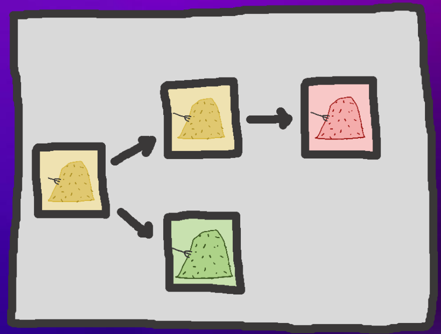
Finding the root cause

- Propagate trace information
- Build lots of dashboards
- Get multi-dimensional
- Use high cardinality metrics
- Dig even deeper with log analysis
- Look at the raw logs
- Analyze using profiler

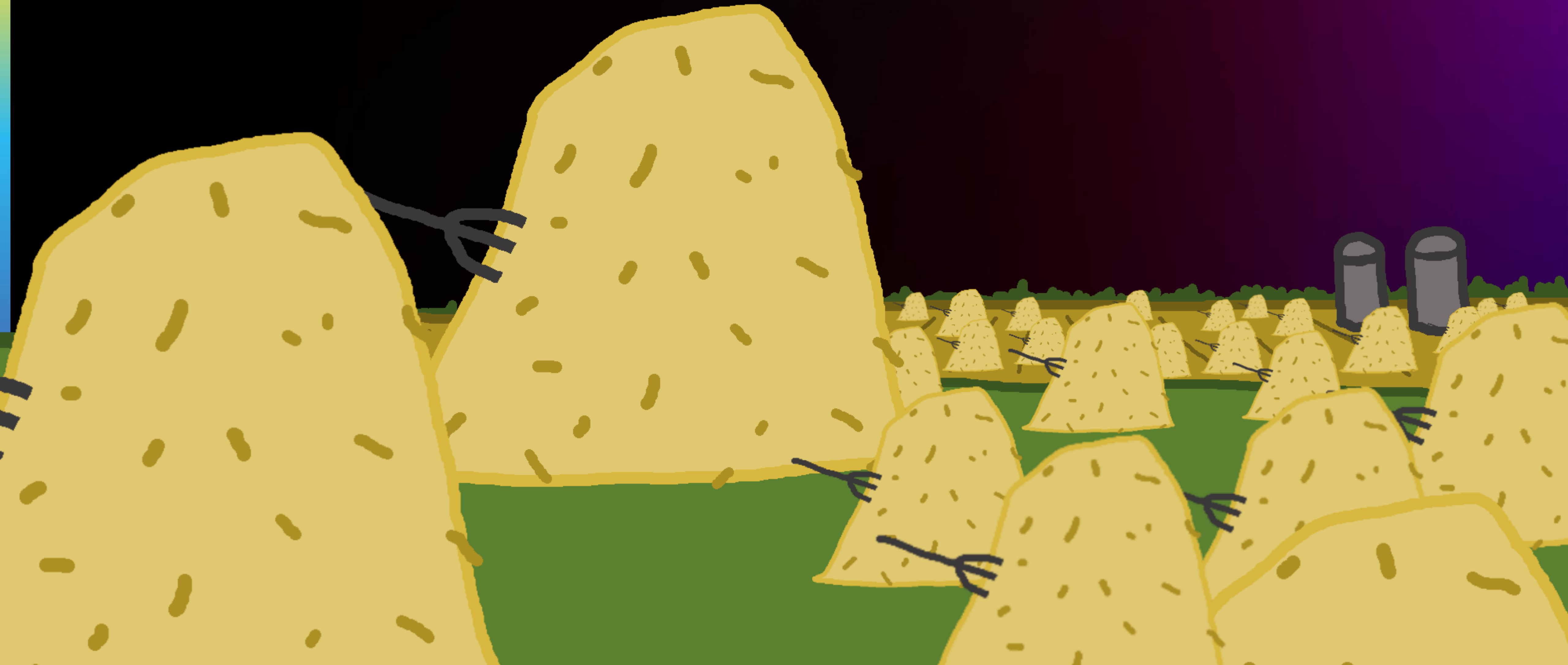


Finding the root cause

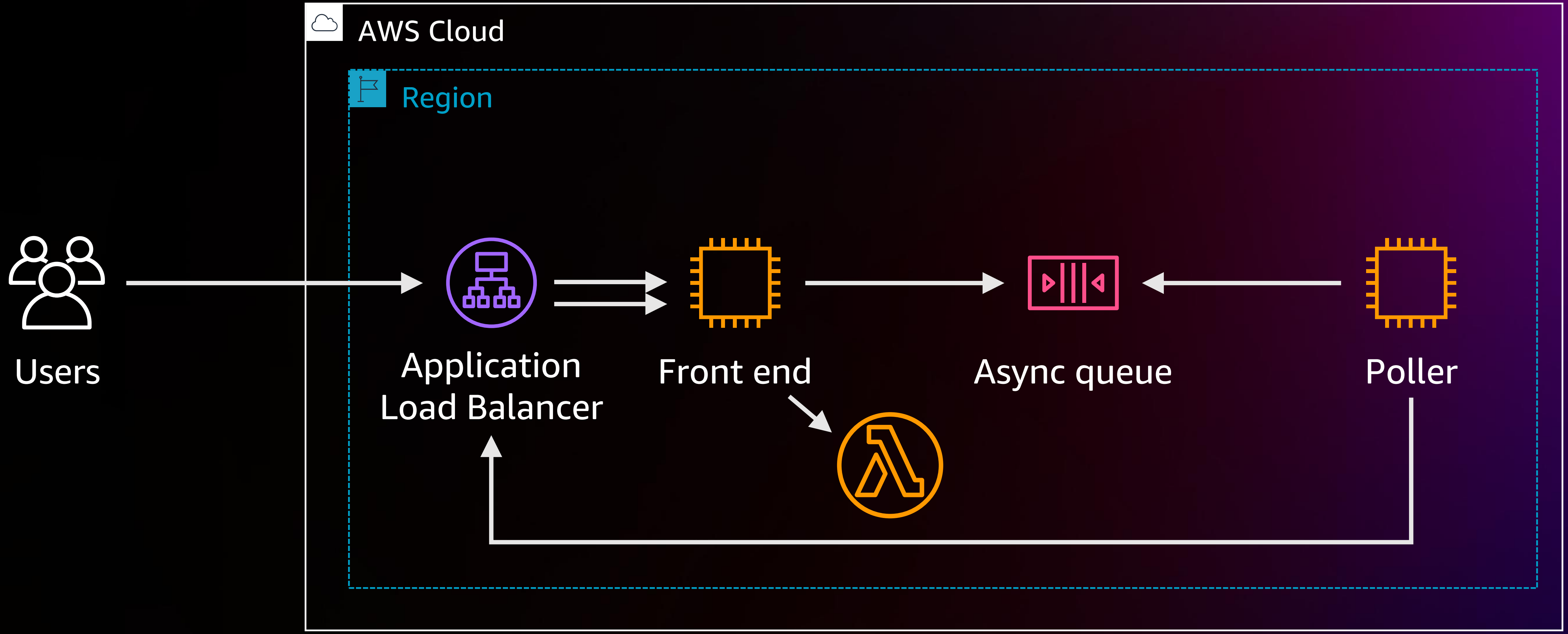
- **Propagate trace information**
- Build lots of dashboards
- Get multi-dimensional
- Use high cardinality metrics
- Dig even deeper with log analysis
- Look at the raw logs
- Analyze using profiler



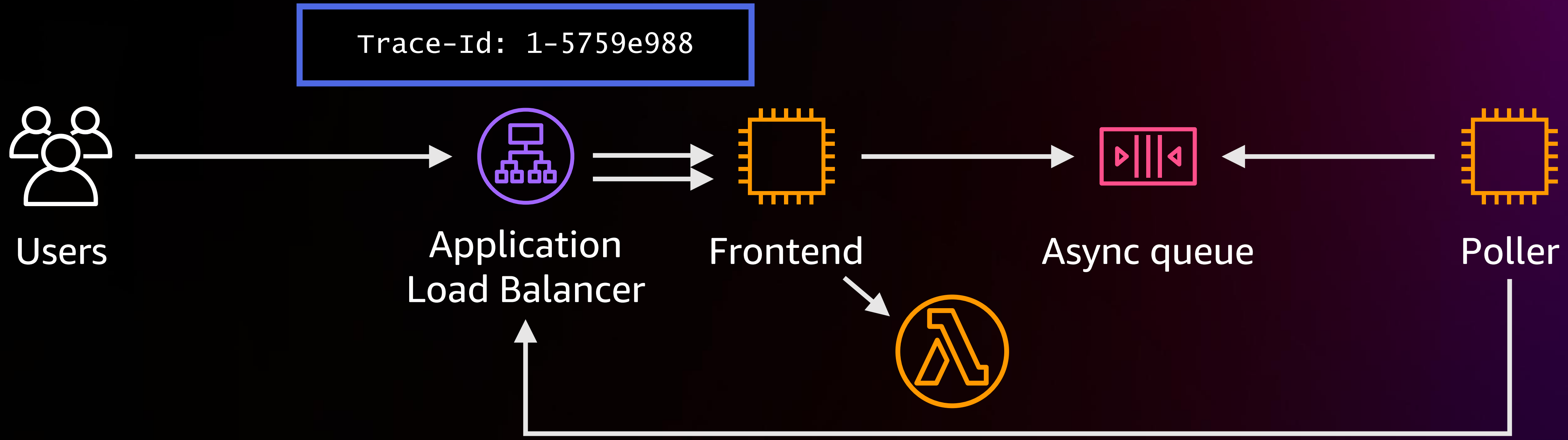
Finding the right haystack with traces



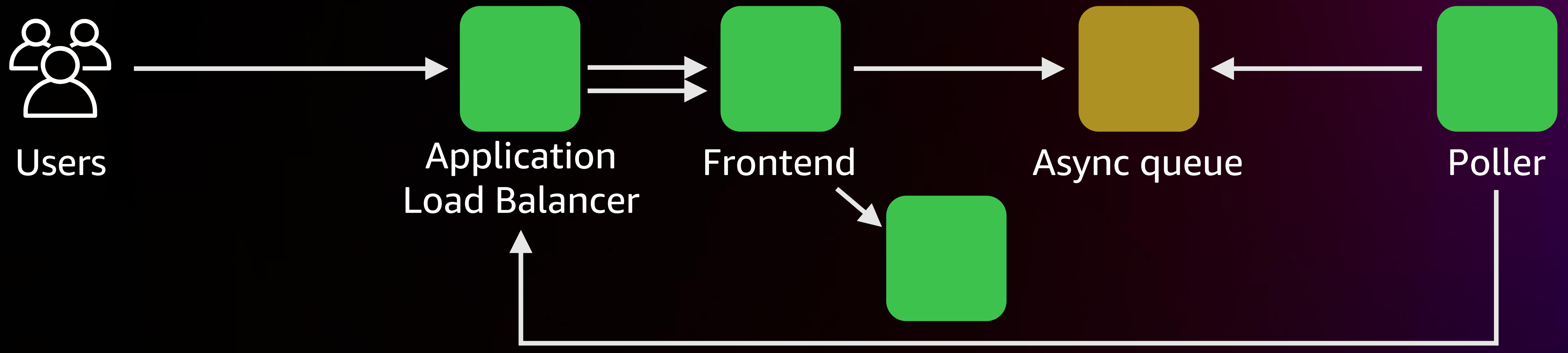
Reasoning about distributed systems

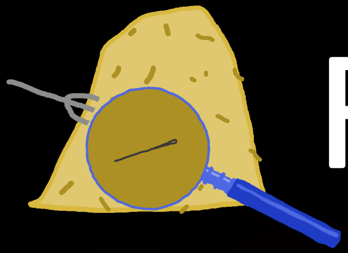


Distributed tracing



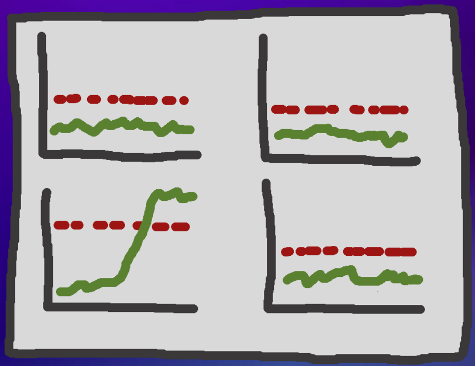
Distributed system-wide visibility



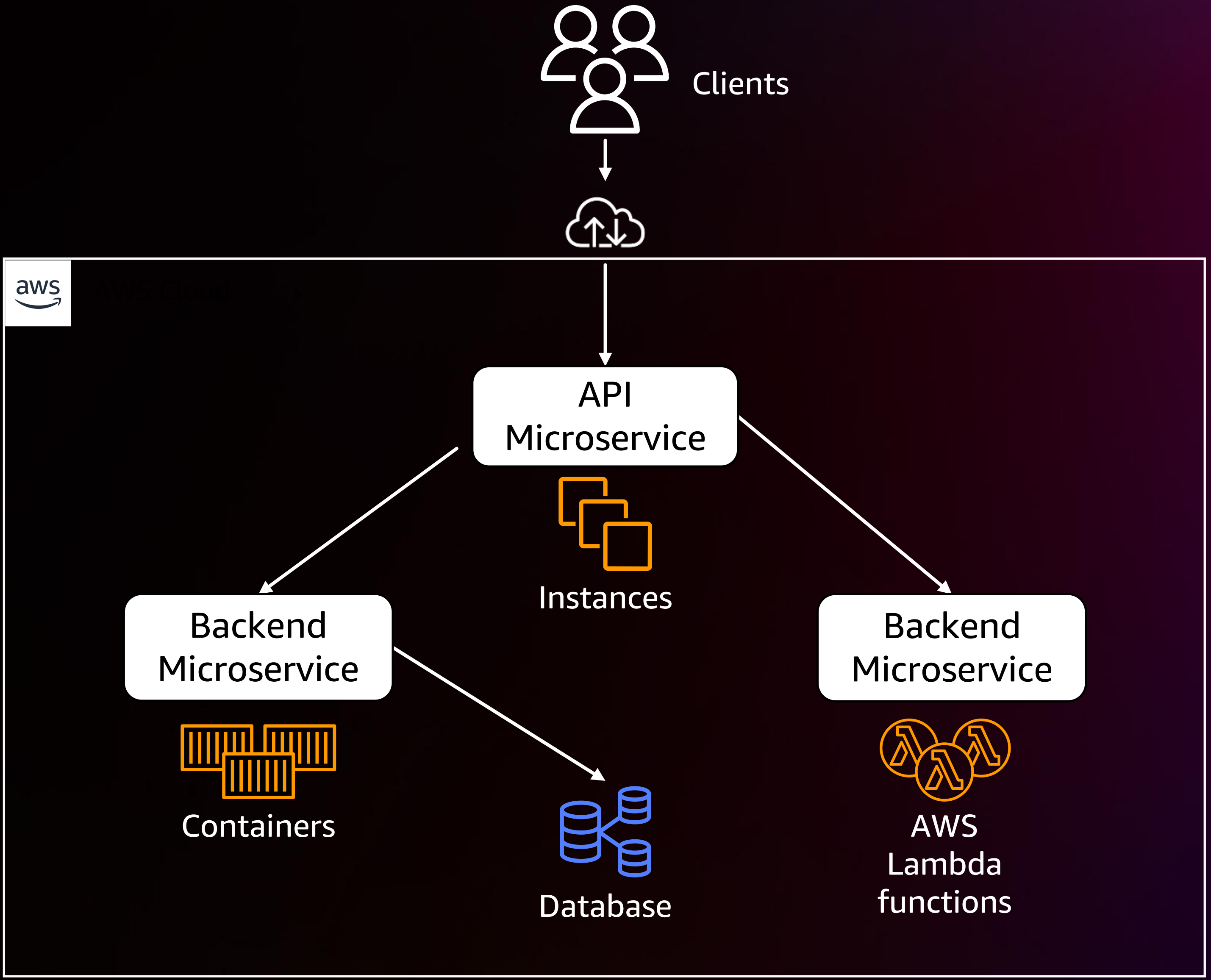


Finding the root cause

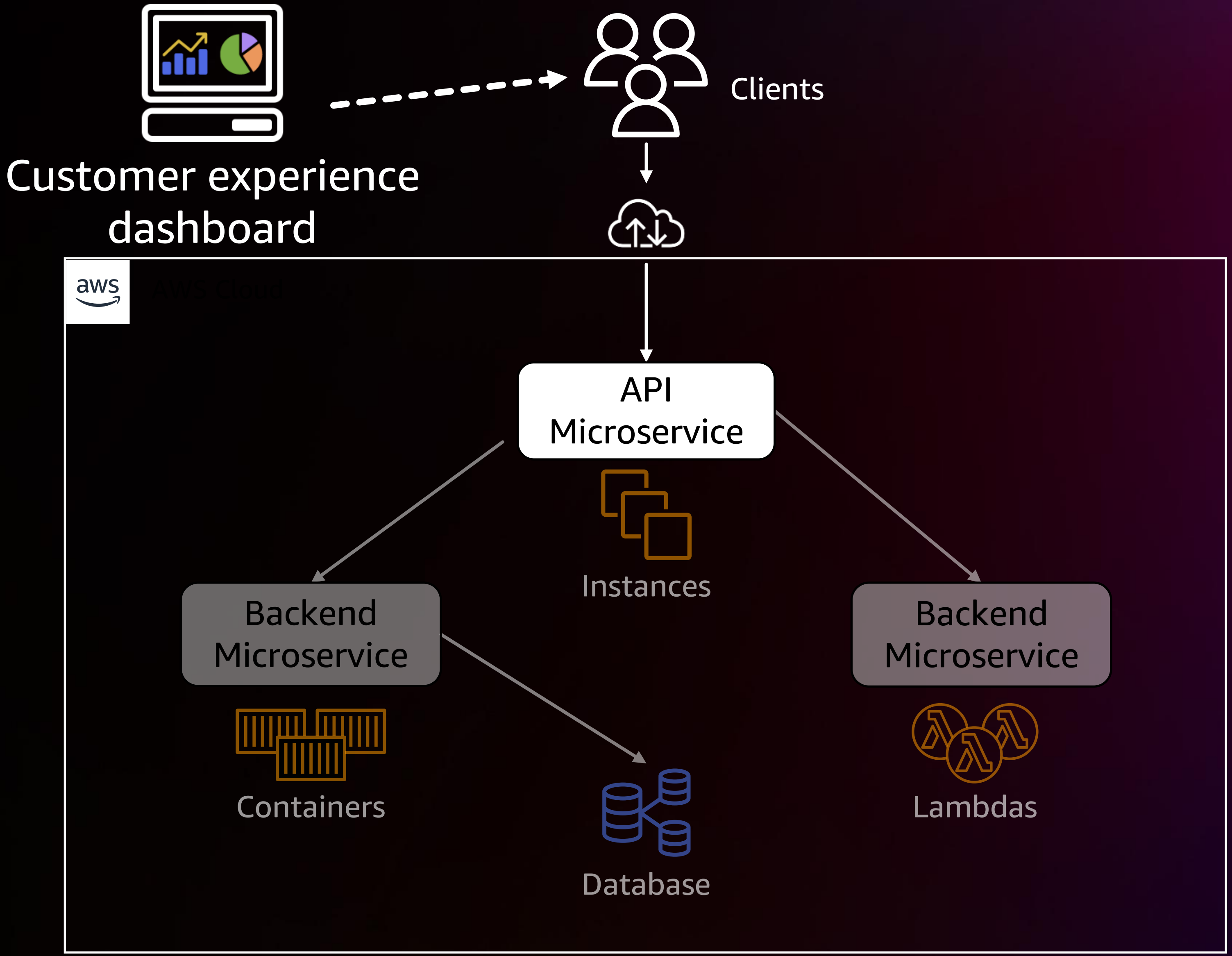
- Propagate trace information
- **Build lots of dashboards**
- Get multi-dimensional
- Use high cardinality metrics
- Dig even deeper with log analysis
- Look at the raw logs
- Analyze using profiler



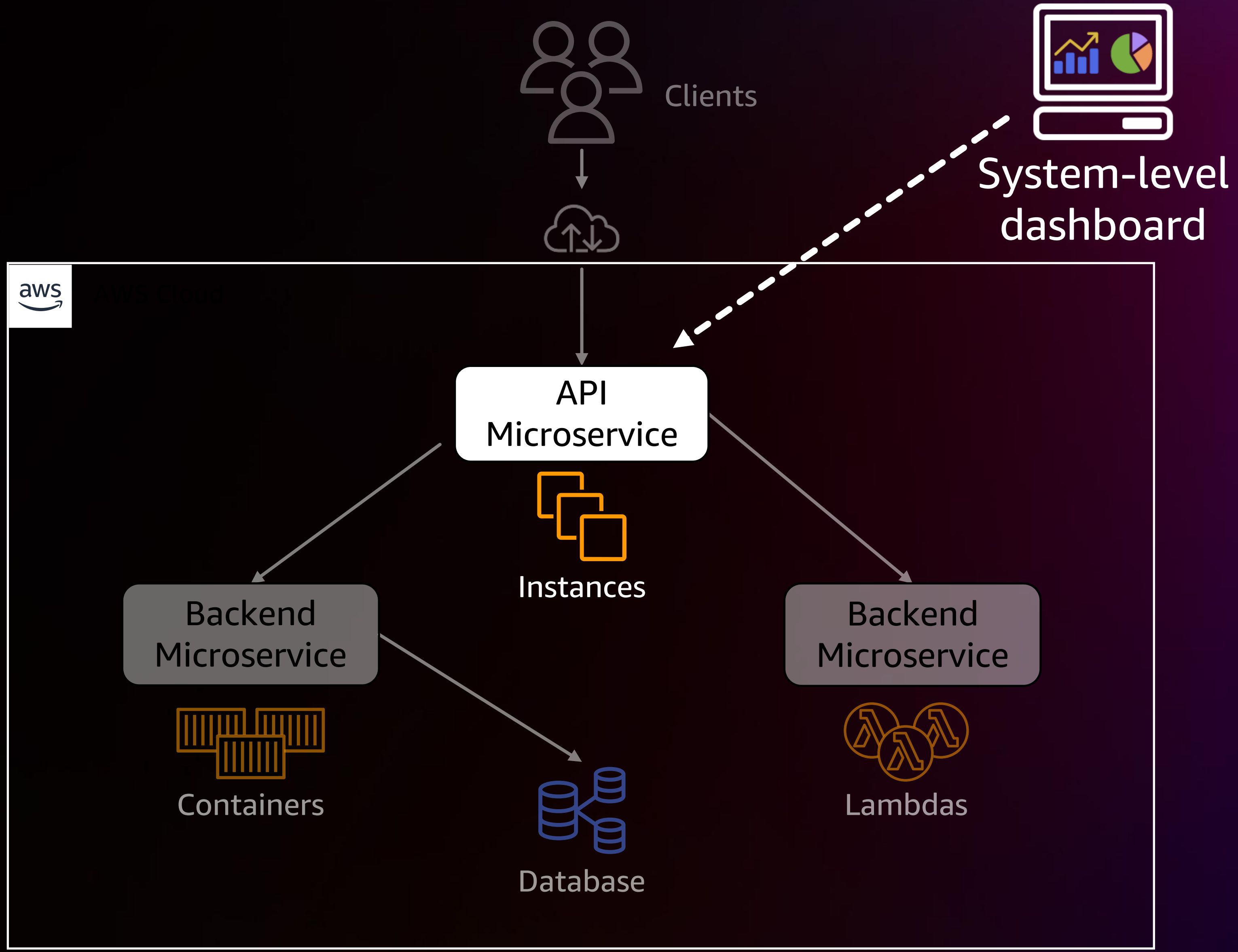
Hierarchy of dashboards



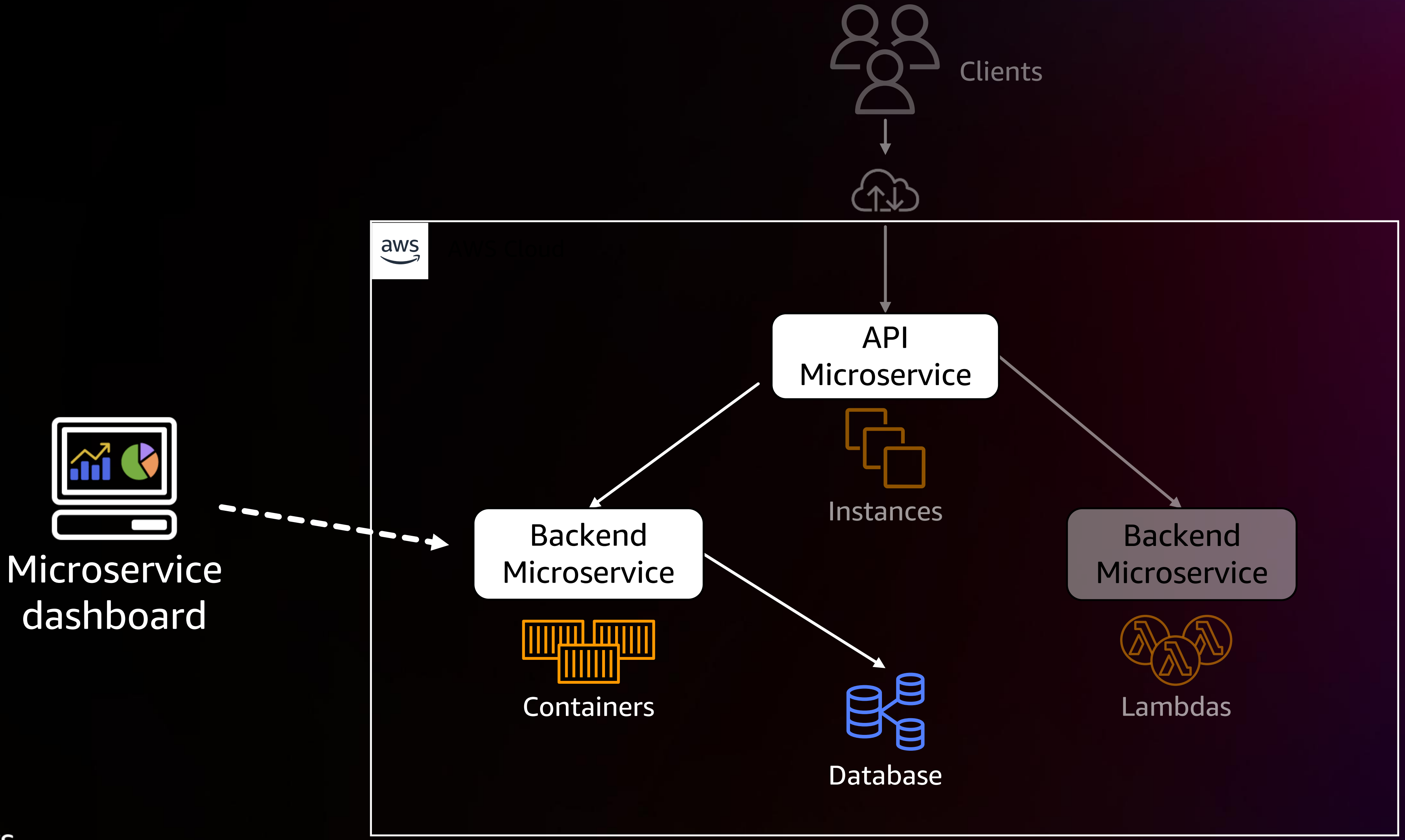
Customer experience dashboards



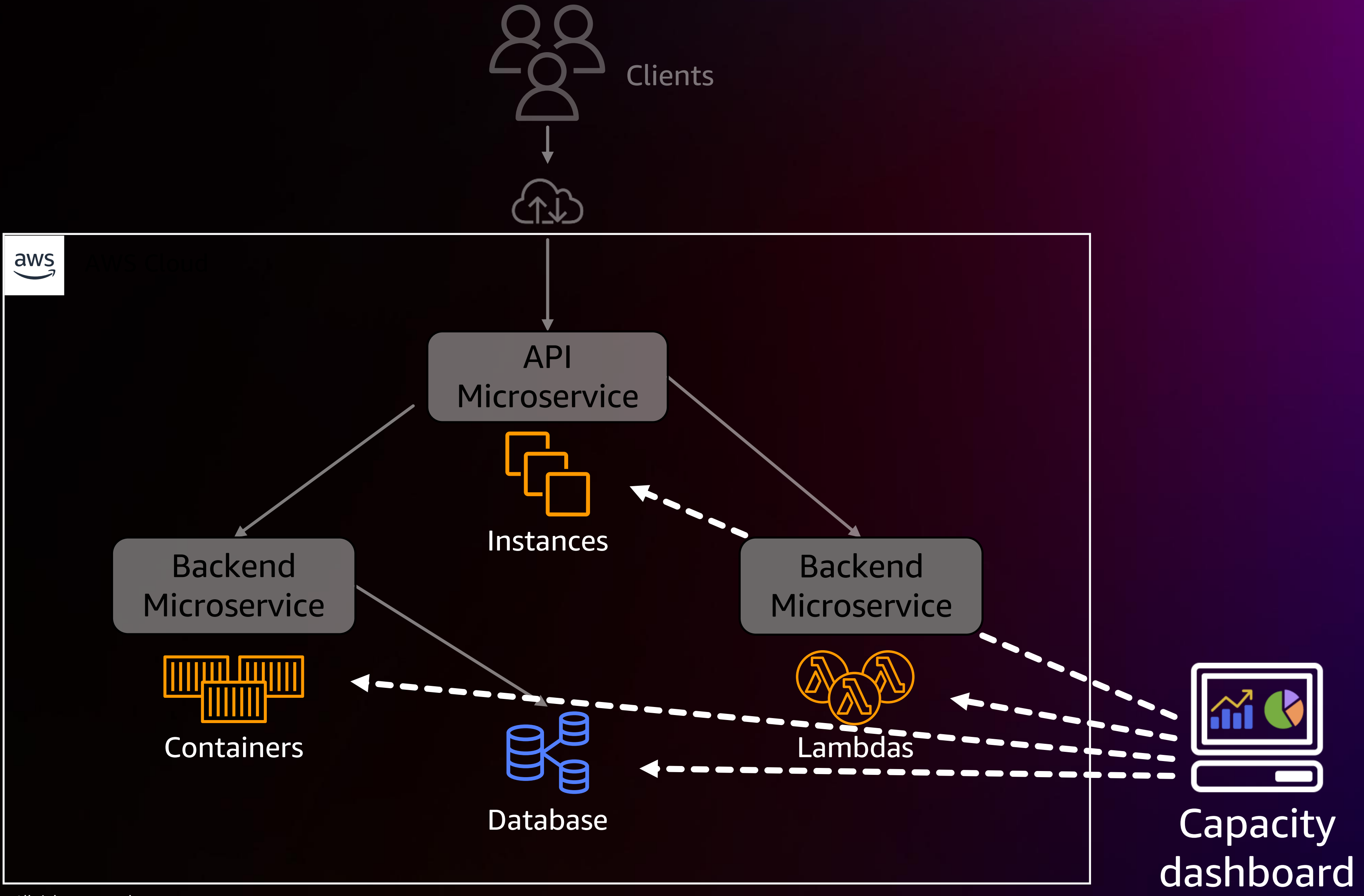
System-level dashboards



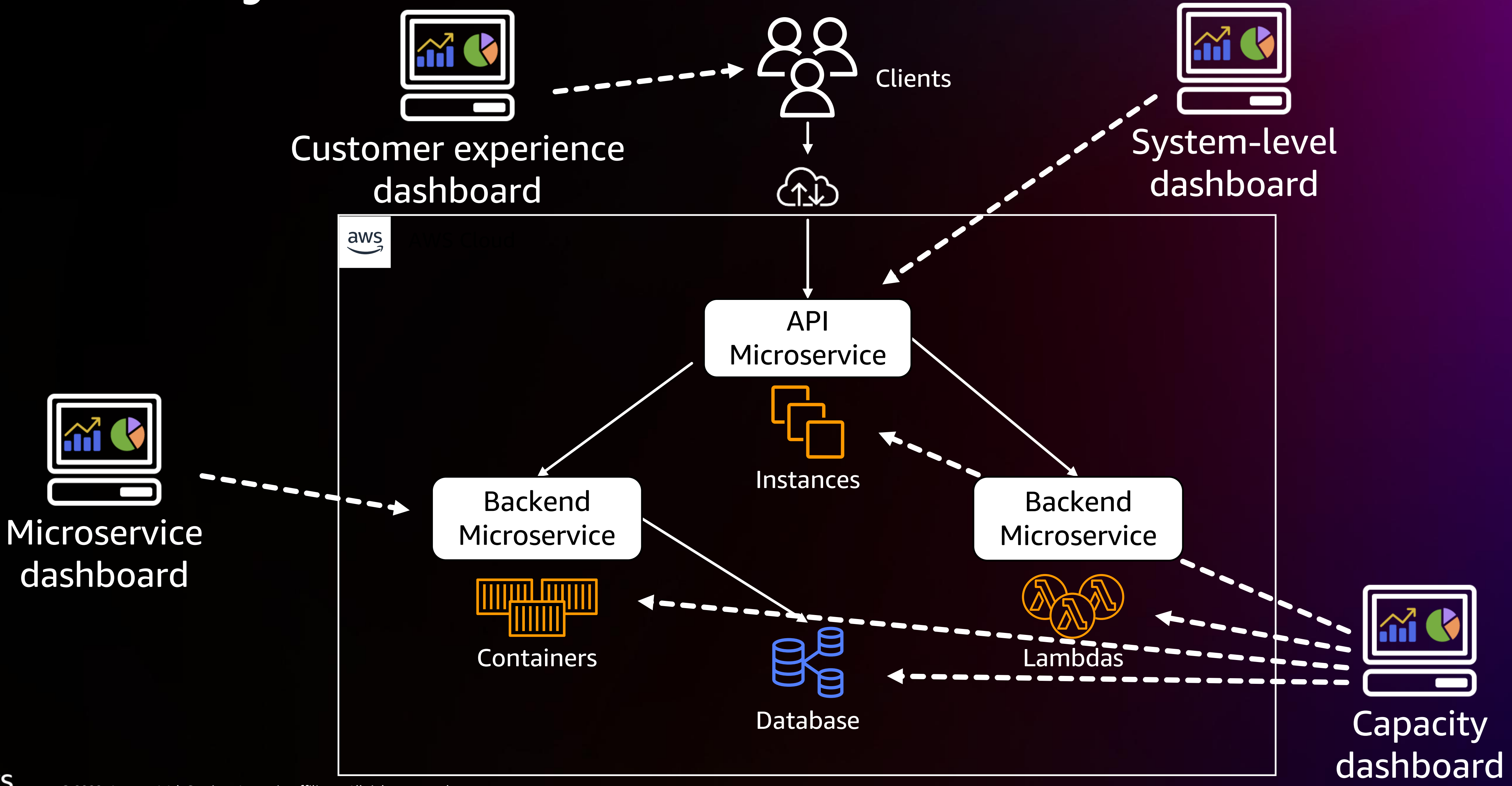
Microservice dashboards



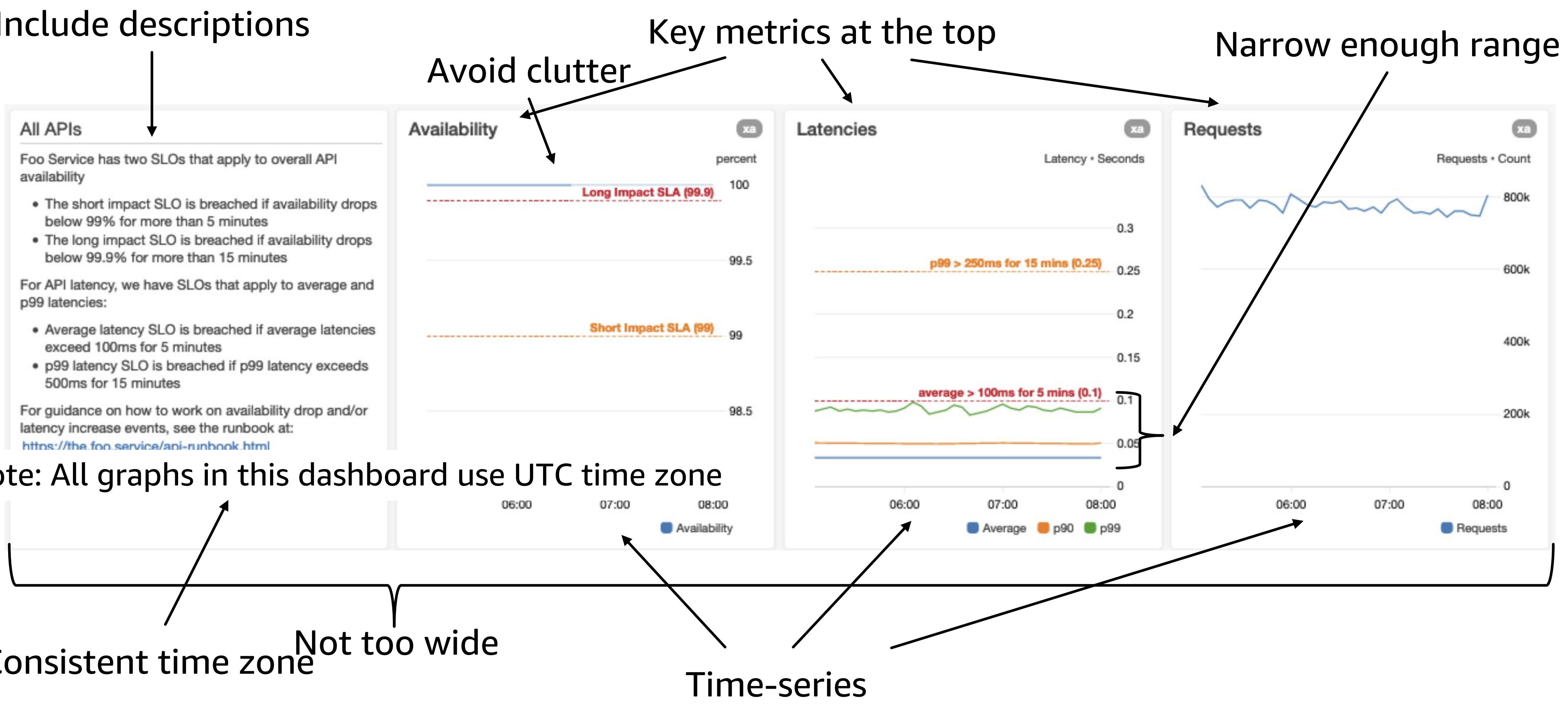
Capacity dashboards



Hierarchy of dashboards



Dashboard design



All APIs

Foo Service has two SLOs that apply to overall API availability

- The short impact SLO is breached if availability drops below 99% for more than 5 minutes
- The long impact SLO is breached if availability drops below 99.9% for more than 15 minutes

For API latency, we have SLOs that apply to average and p99 latencies:

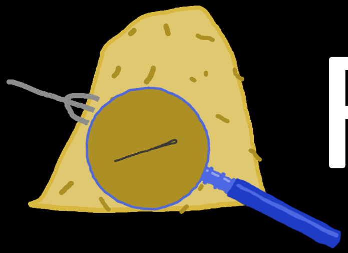
- Average latency SLO is breached if average latencies exceed 100ms for 5 minutes
- p99 latency SLO is breached if p99 latency exceeds 500ms for 15 minutes

For guidance on how to work on availability drop and/or latency increase events, see the runbook at:
https://the.foo.service/api-runbook.html

Note: All graphs in this dashboard use UTC time zone

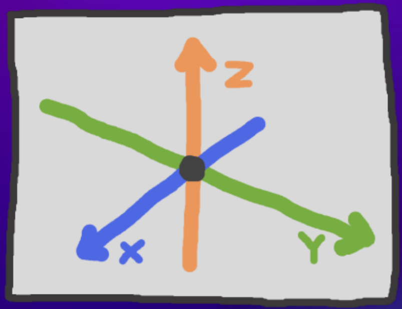
The dashboard is never done

https://aws.amazon.com/builders-library/building-dashboards-for-operational-visibility/

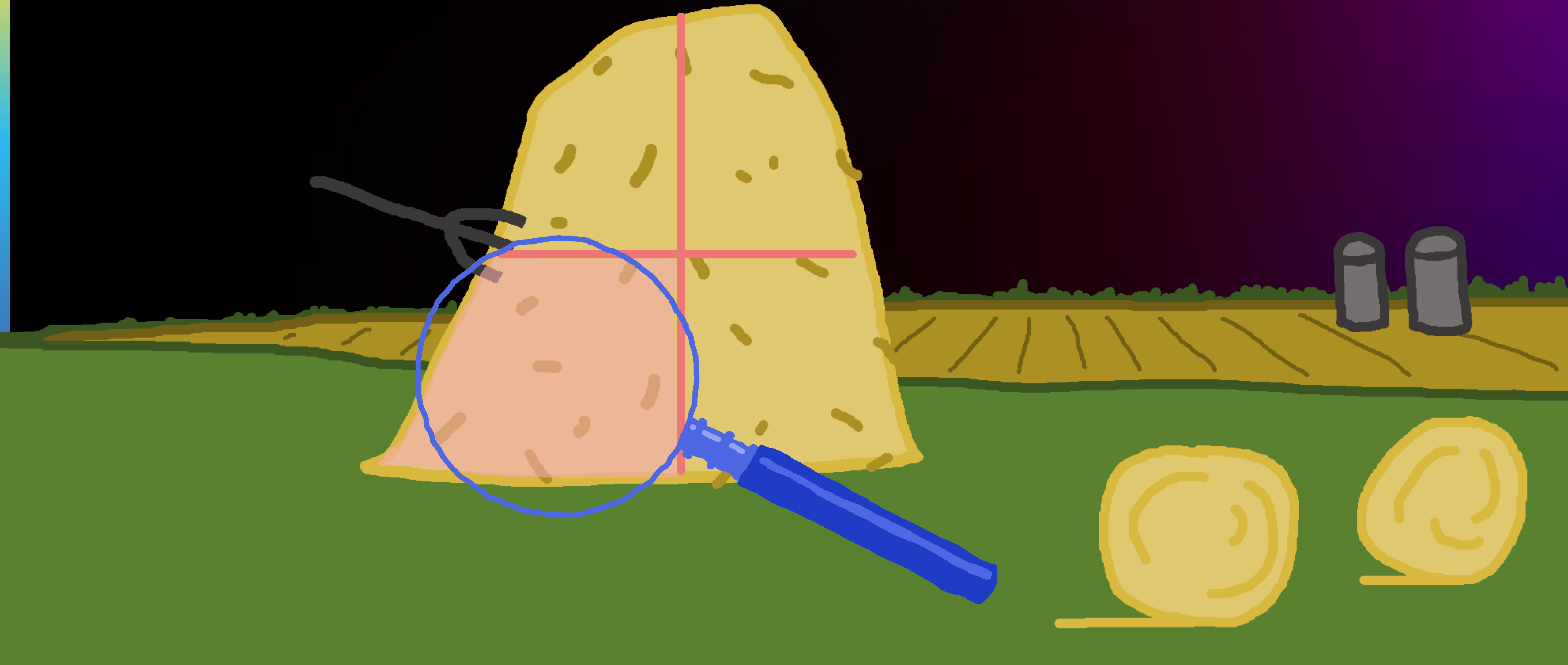


Finding the root cause

- Propagate trace information
- Build lots of dashboards
- **Get multi-dimensional**
- Use high cardinality metrics
- Dig even deeper with log analysis
- Look at the raw logs
- Analyze using profiler



Use metric dimensions to drill down



Log everything: Embedded Metrics Format

```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "ProductInfoService",
        "Dimensions": [],
        "Metrics": [
          { "Name": "Time", "Unit": "Milliseconds" }
  ]}]},
  "CustomerId": "user@example.com",
  "InstanceId": "us-west-2c",
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

Instrumentation

Configuration

Example: Simple web service code

```
public ProductInfo getProductInfo(String customerId, String productId) {

  ProductInfo info = cache.get(customerId, productId);

  if (info == null) {
    info = db.query(customerId, productId);
  }

  return info;
}
```

What product or customer was this for?

Was this a cache hit or miss?

If it failed, was it a timeout or did it return an error?

How long did the db query take? Did it return anything?

Example: Instrumenting code

```
public ProductInfo getProductInfo(String customerId, String productId,
      Metrics metrics) {

  metrics.addProperty("CustomerId", customerId);
  metrics.addProperty("ProductId", productId);

  ProductInfo info = cache.get(customerId, productId, metrics);

  if (info == null) {
    info = db.query(customerId, productId, metrics);
  }

  return info;
}
```

Record what we're working on

Propagate per-request context object

Instrumentation in the log

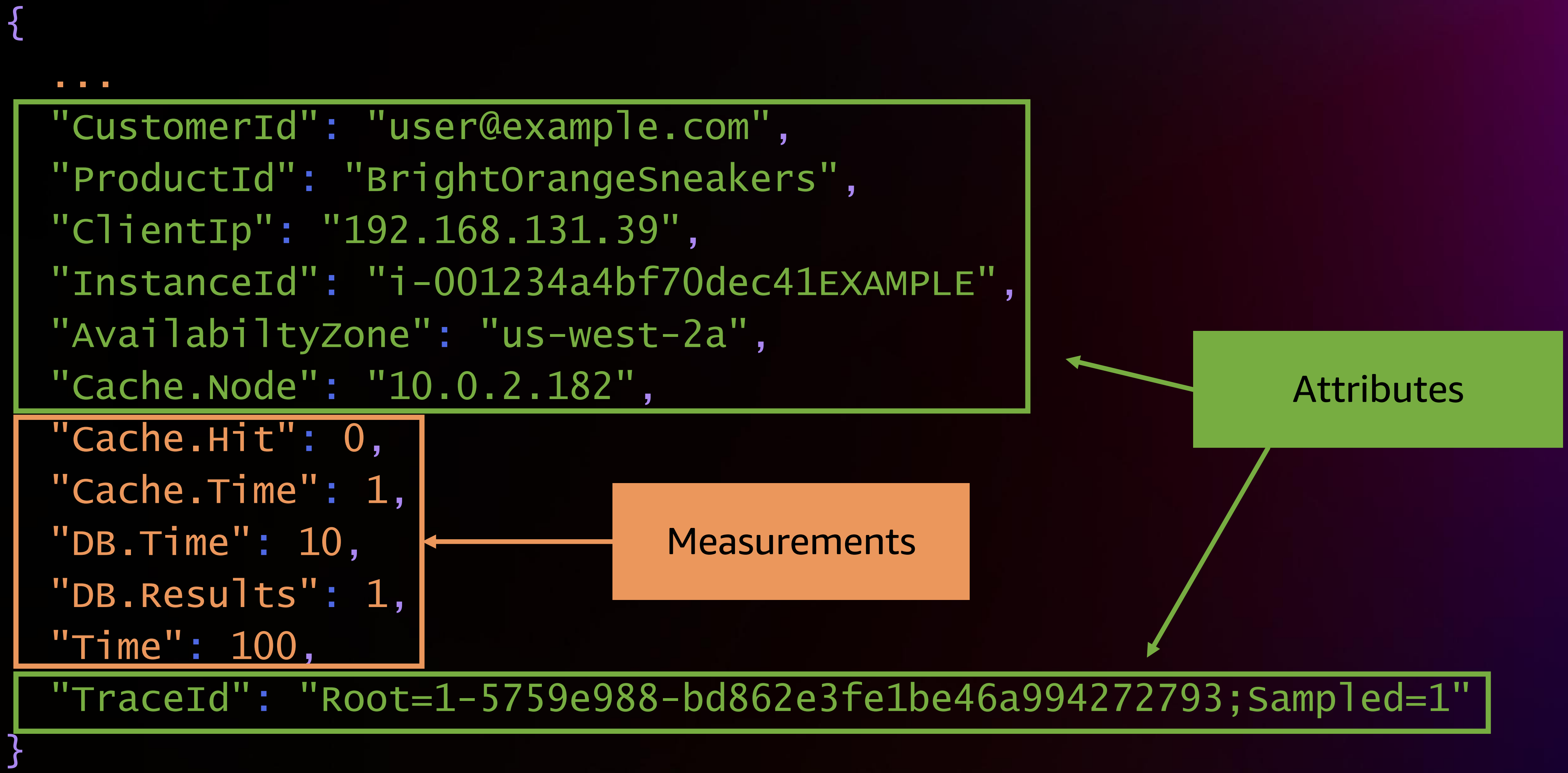
```
{
  ...
  "CustomerId": "user@example.com",
  "ProductId": "BrightOrangeSneakers",
  "ClientIp": "192.168.131.39",
  "InstanceId": "i-001234a4bf70dec41EXAMPLE",
  "AvailabiltyZone": "us-west-2a",
  "Cache.Node": "10.0.2.182",
  "Cache.Hit": 0,
  "Cache.Time": 1,
  "DB.Time": 10,
  "DB.Results": 1,
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

Caller and request info

Infrastructure info

Timing and info around dependencies

Log everything about a unit of work



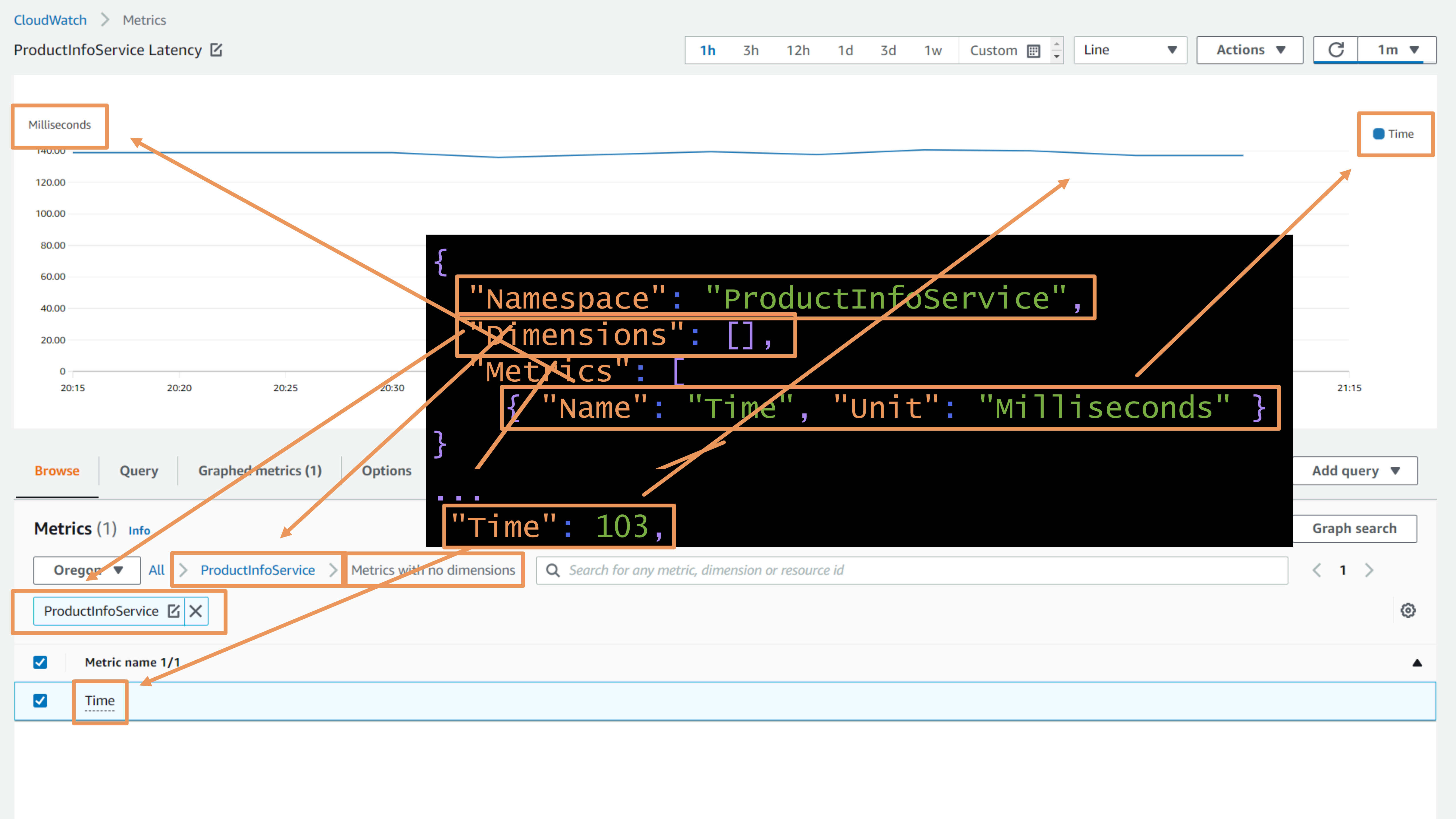
```
{
  ...
  "CustomerId": "user@example.com",
  "ProductId": "BrightOrangeSneakers",
  "ClientIp": "192.168.131.39",
  "InstanceId": "i-001234a4bf70dec41EXAMPLE",
  "AvailabiltyZone": "us-west-2a",
  "Cache.Node": "10.0.2.182",
  "Cache.Hit": 0,
  "Cache.Time": 1,
  "DB.Time": 10,
  "DB.Results": 1,
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

Log everything: Embedded Metrics Format

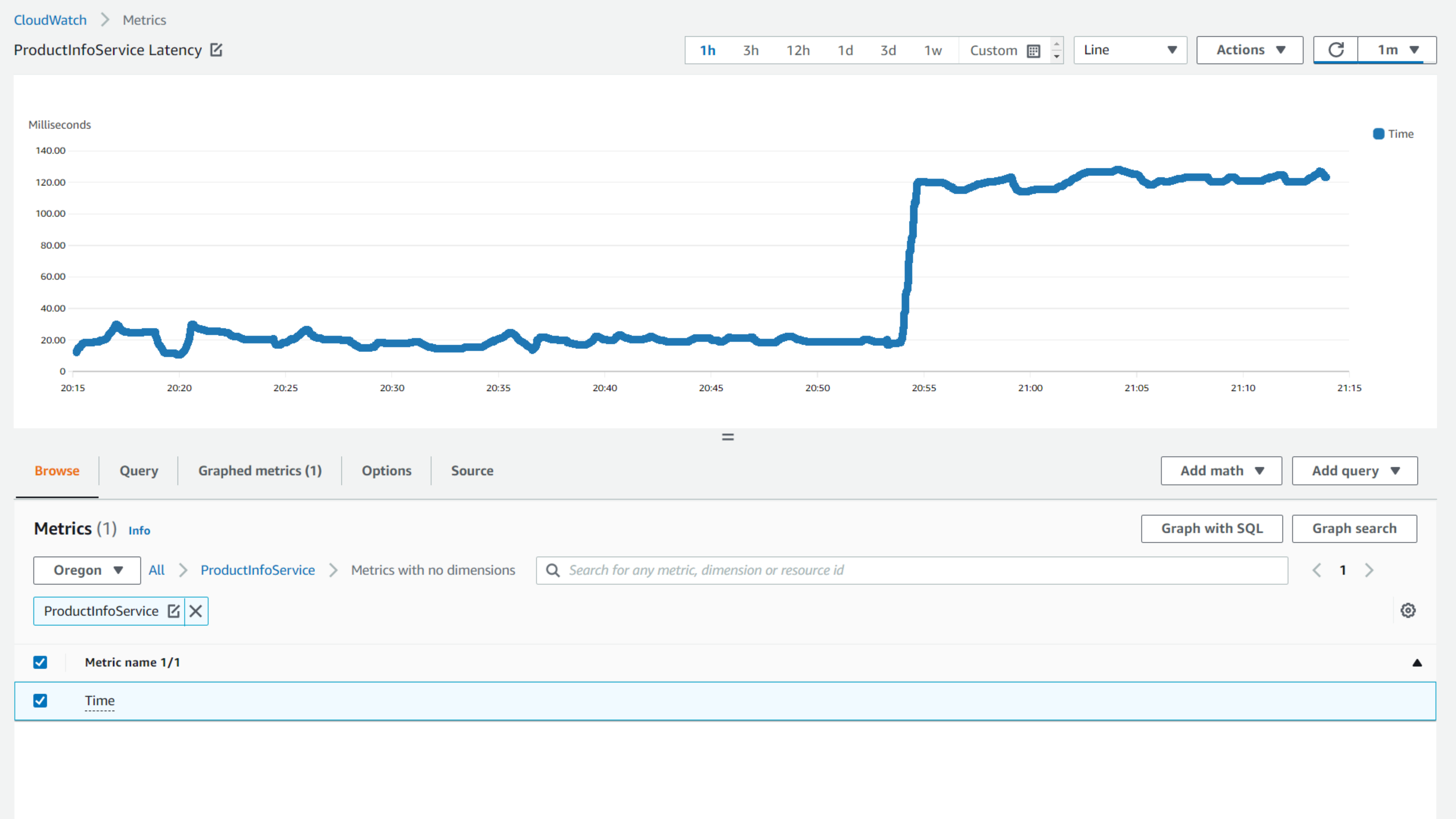
```
{
  "_aws": {
    "Timestamp": 1574109732004,
    "CloudWatchMetrics": [
      {
        "Namespace": "ProductInfoService",
        "Dimensions": [],
        "Metrics": [
          { "Name": "Time", "Unit": "Milliseconds" }
  ]}]},
  "CustomerId": "user@example.com",
  "AvailabilityZone": "us-west-2c",
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

Metric definitions

Defines which data to turn into a metric



```
{
  "Namespace": "ProductInfoService",
  "Dimensions": [],
  "Metrics": [
    { "Name": "Time", "Unit": "Milliseconds" }
}

...
"Time": 103,
```



CloudWatch > Metrics

ProductInfoService Latency

Metrics (1) Info

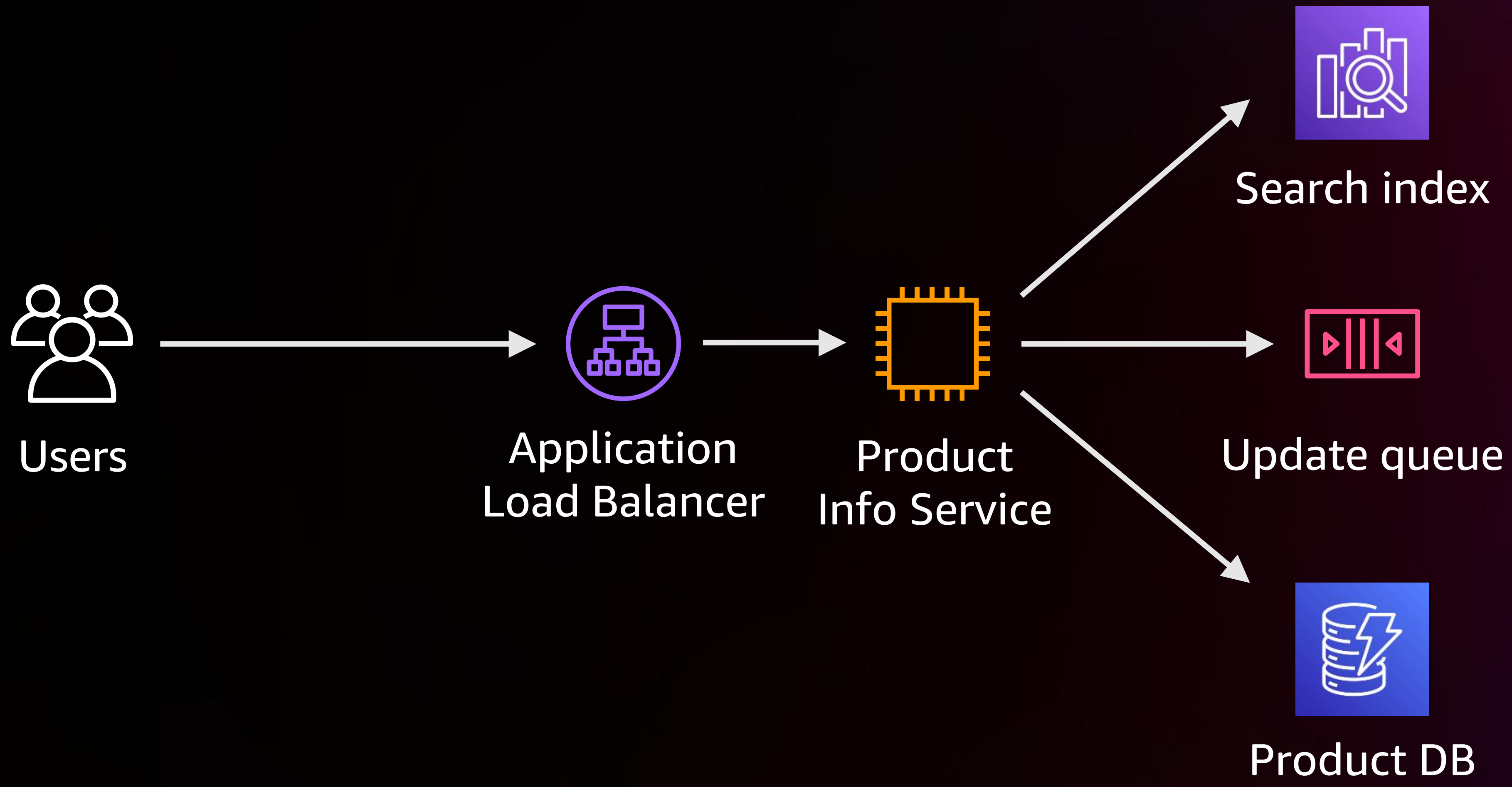
Oregon | All > ProductInfoService > Metrics with no dimensions

Search for any metric, dimension or resource id

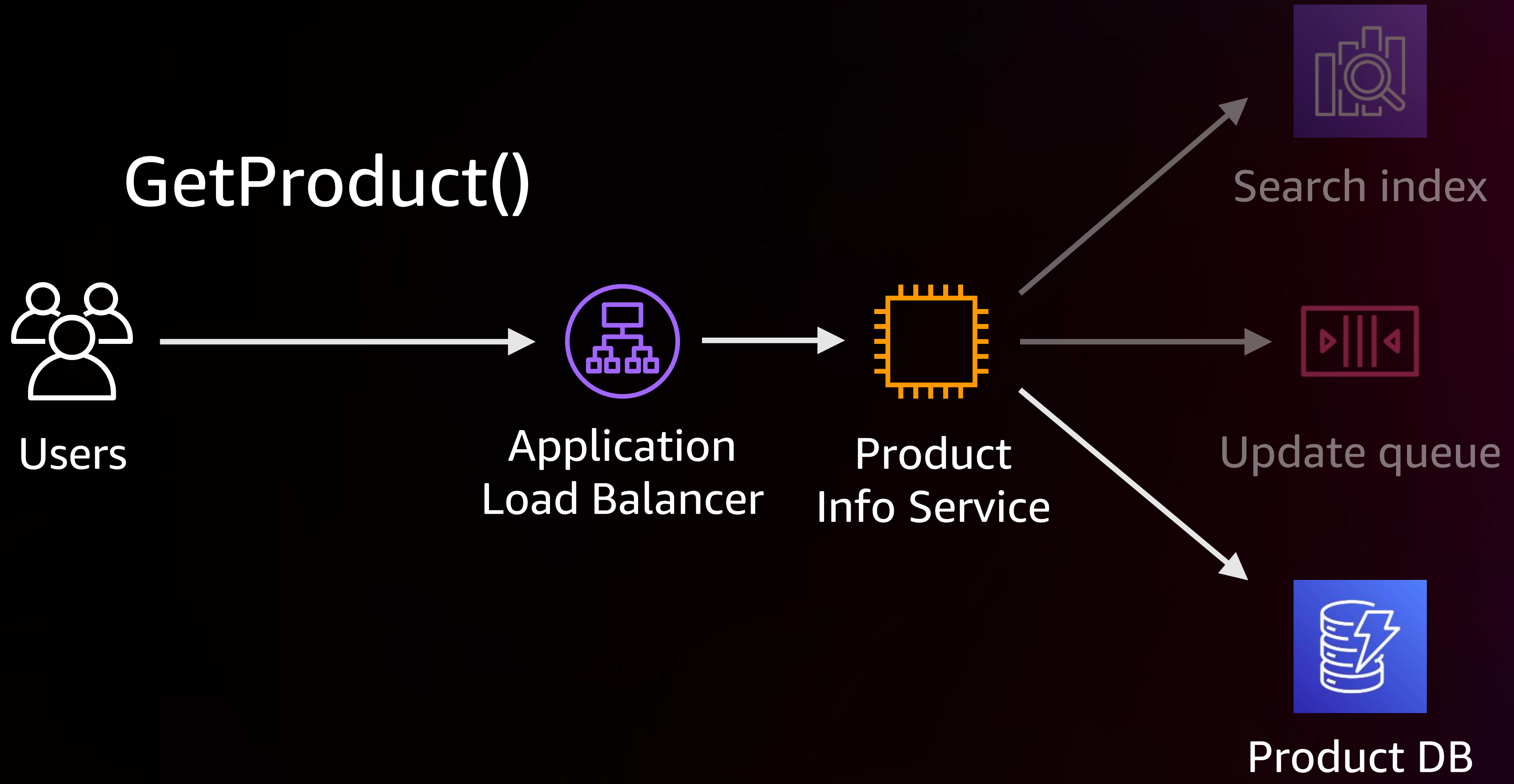
ProductInfoService

	Metric name 1/1
✓	Time

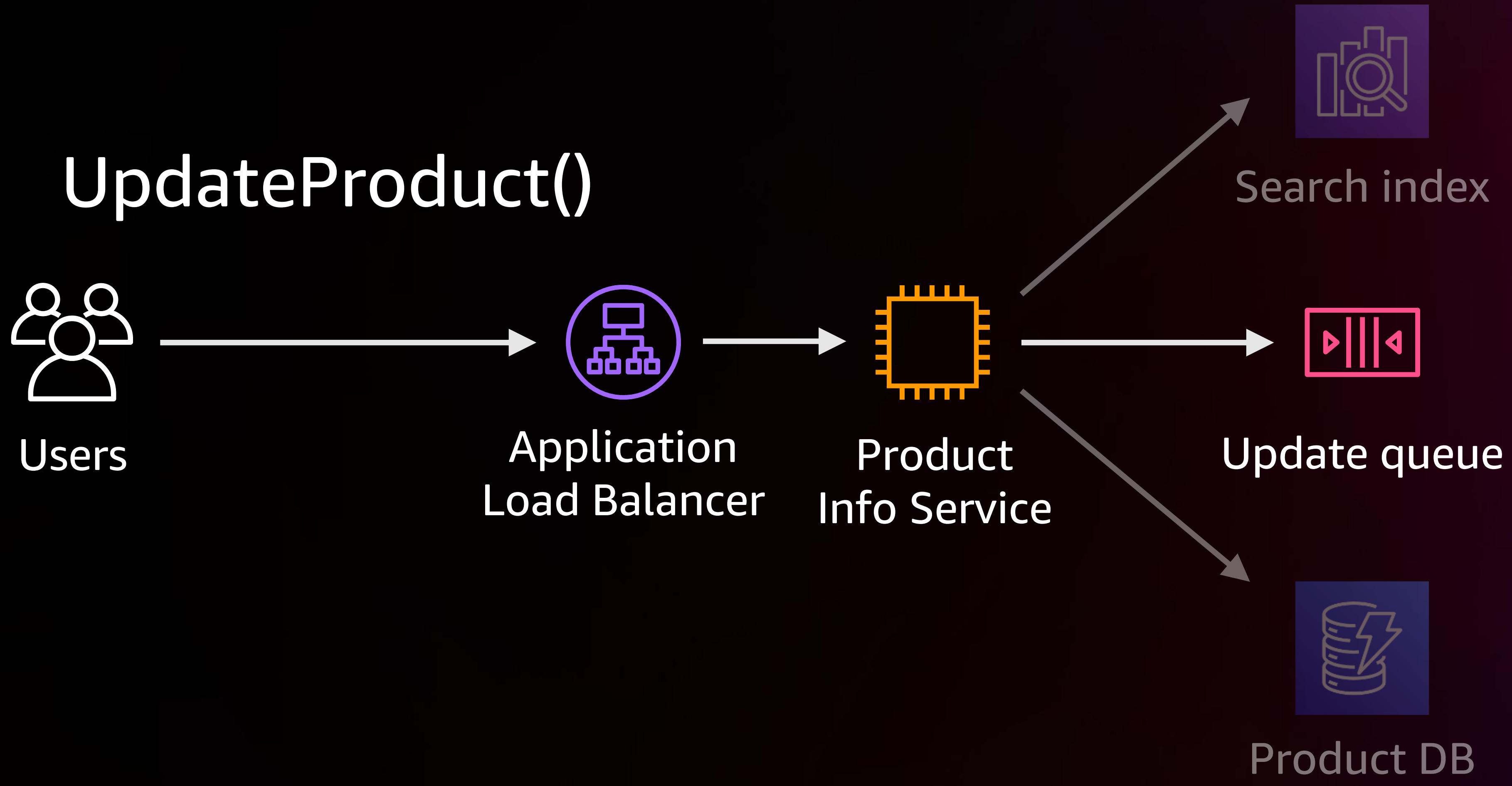
Product Info Service architecture



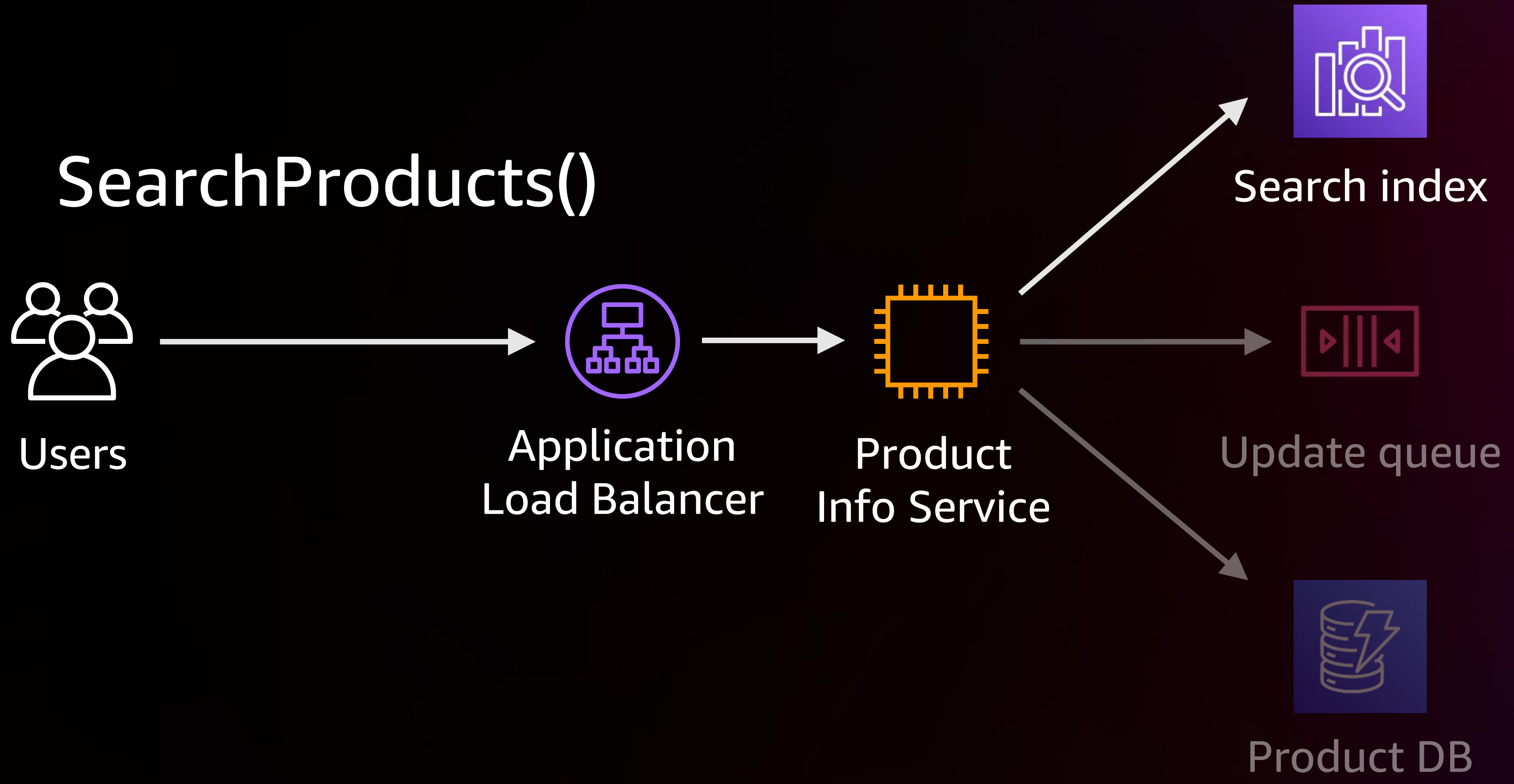
Product Info Service architecture

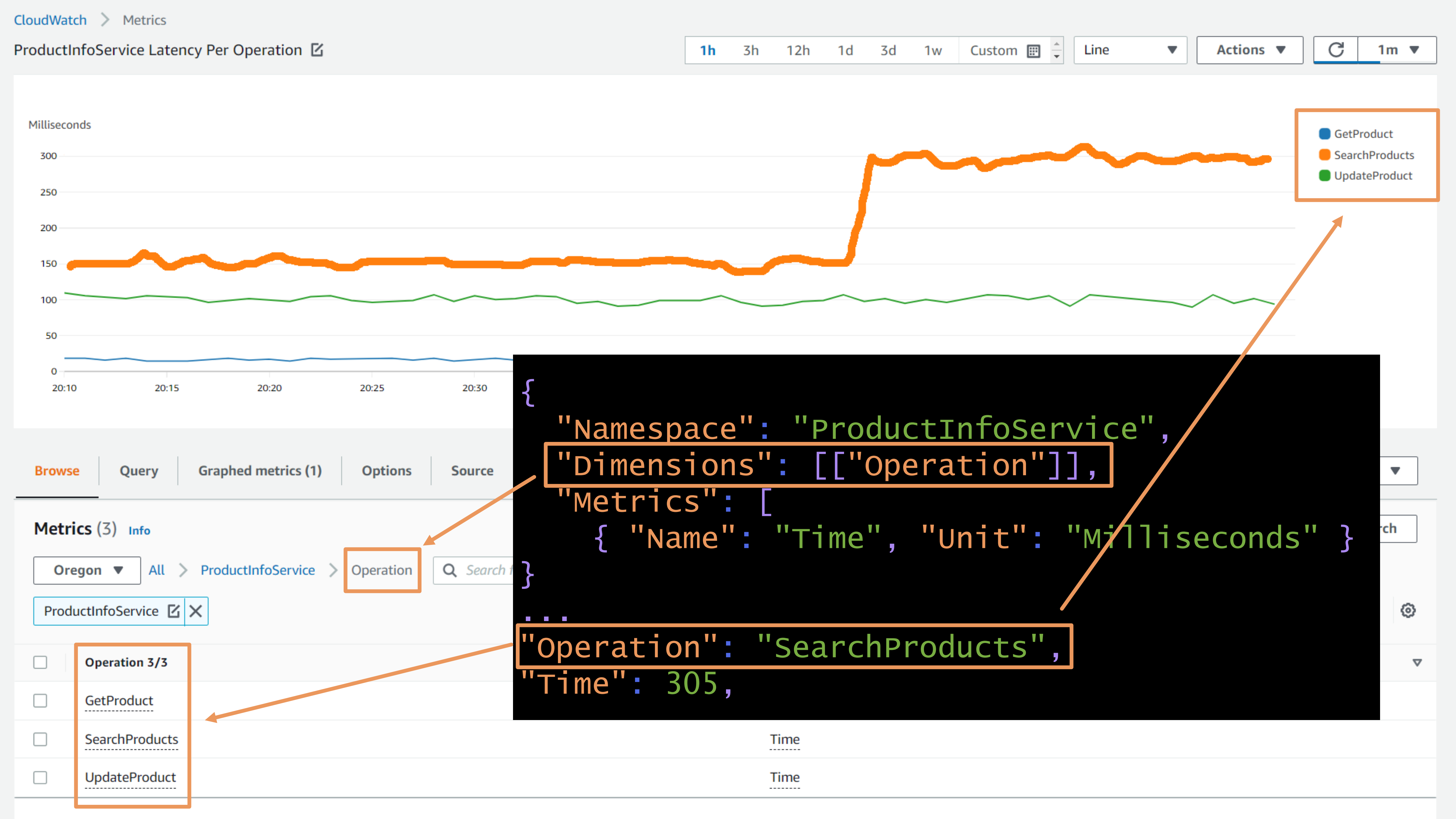


Product Info Service architecture



Product Info Service architecture





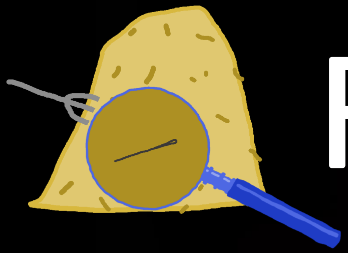
ProductInfoService Latency Per Operation

Metrics (3)

ProductInfoService

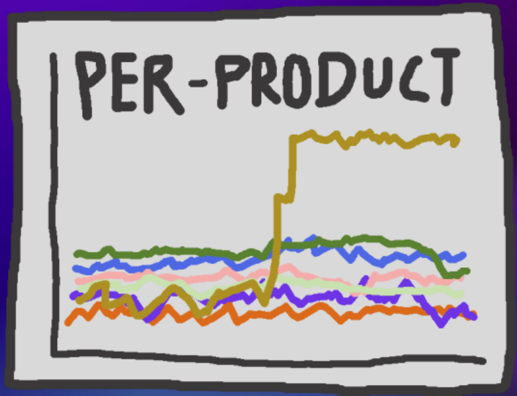
Operation 3/3	
GetProduct	
SearchProducts	Time
UpdateProduct	Time

```
{
  "Namespace": "ProductInfoService",
  "Dimensions": [["Operation"]],
  "Metrics": [
    { "Name": "Time", "Unit": "Milliseconds" }
}
...
"Operation": "SearchProducts",
"Time": 305,
```



Finding the root cause

- Propagate trace information
- Build lots of dashboards
- Get multi-dimensional
- **Use high cardinality metrics**
- Dig even deeper with log analysis
- Look at the raw logs
- Analyze using profiler



Dimensionality

"Show me [metric] per [dimension]"

Measurements	Attributes
Latency	Availability Zone
Errors	API
	EC2 Instance
	Product Id
	Customer Id

(Few of these)

Cardinality

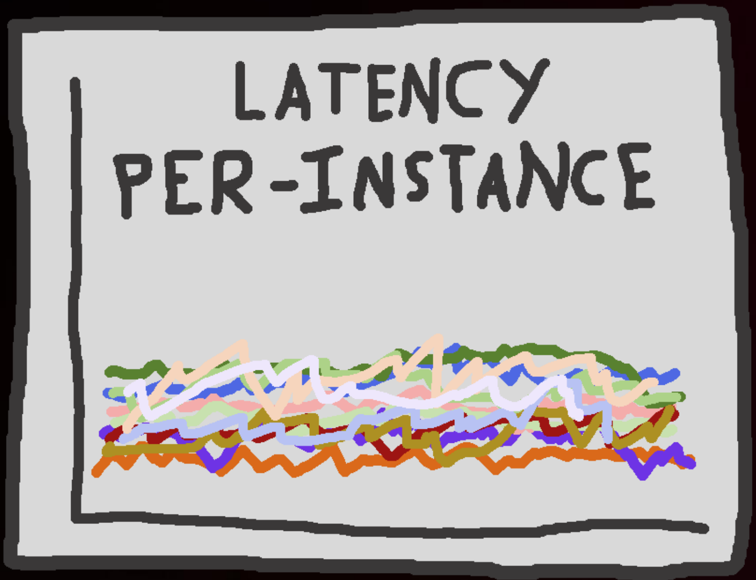
(Many of these)

Analyzing high cardinality metrics

"Are all of my Amazon EC2 instances returning errors?"

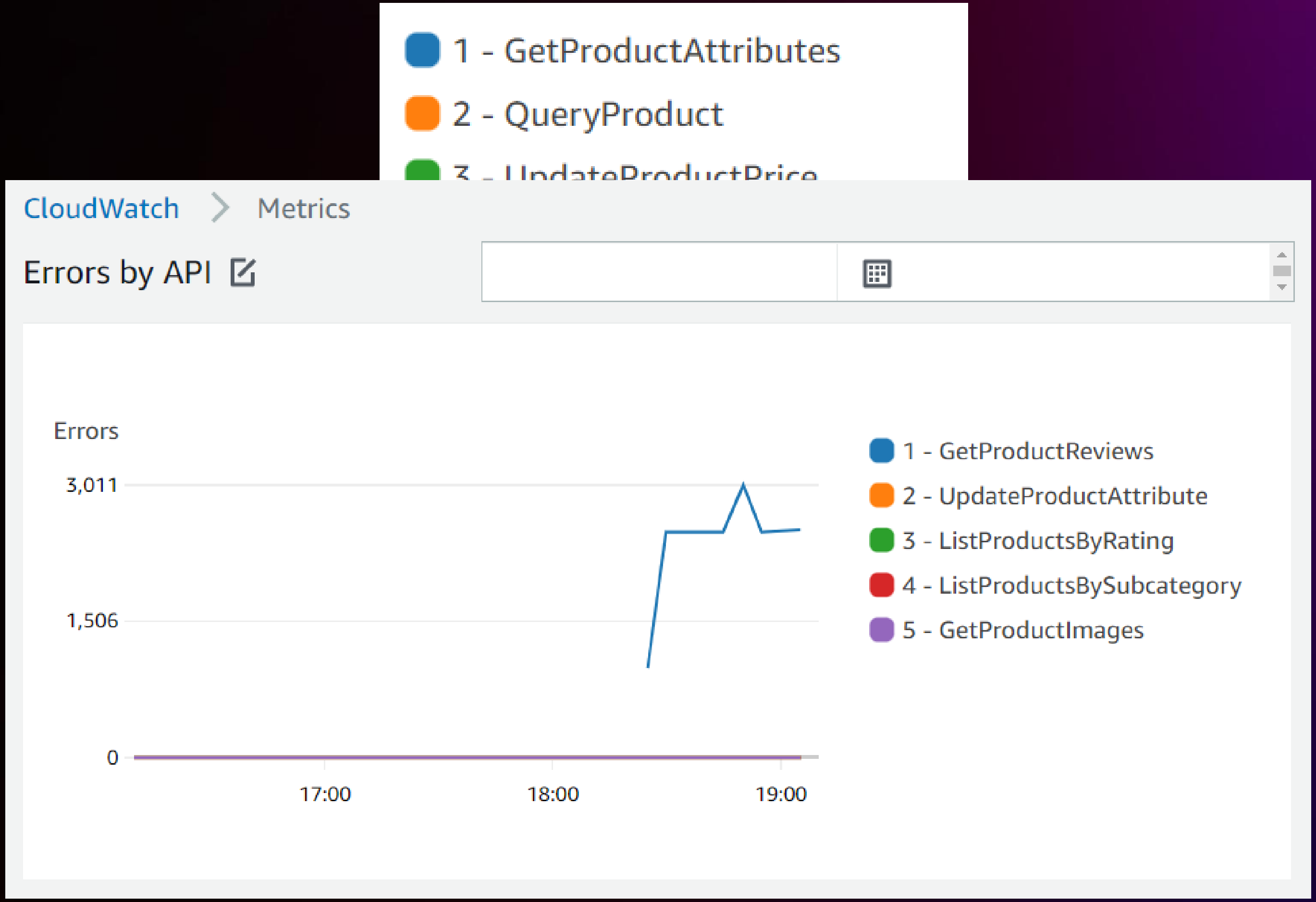
"Are different customer experience different impact?"

"Is just one Product Id having problems?"



CloudWatch Metric Insights

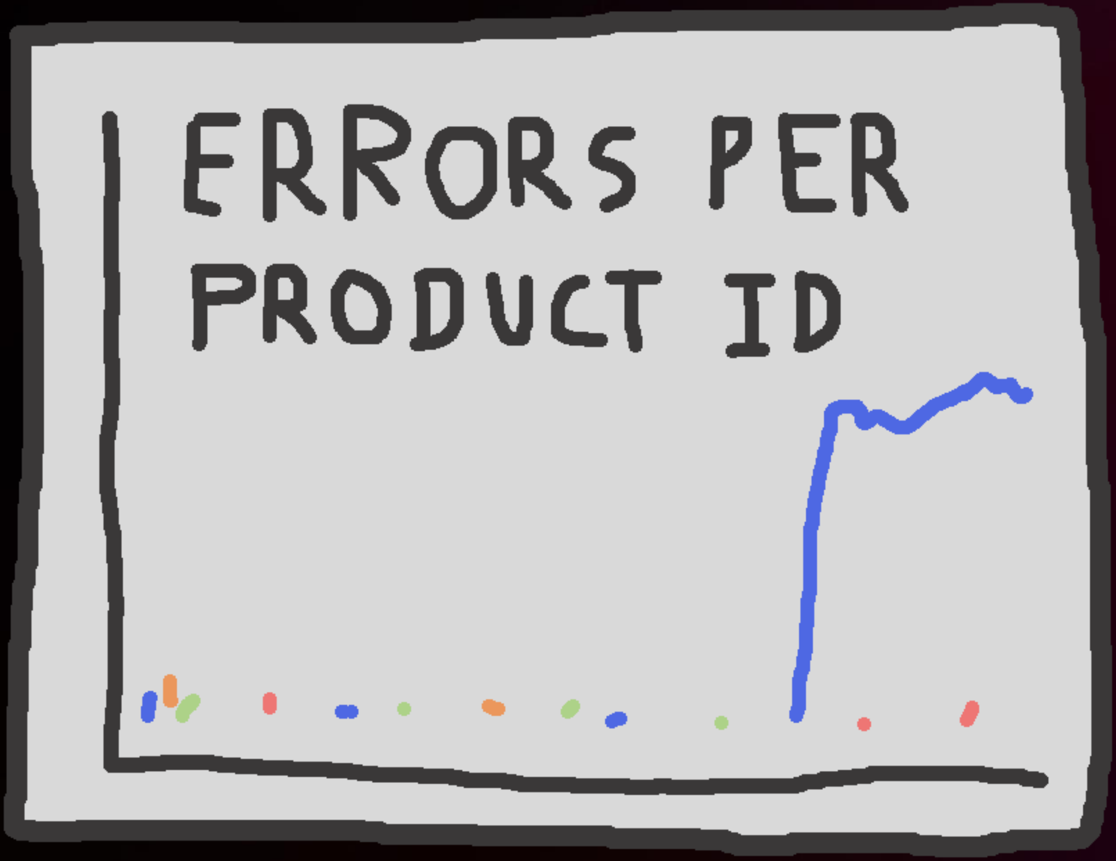
```
SELECT SUM(Errors)
FROM ProductInfoService
GROUP BY Operation
ORDER BY SUM() DESC
LIMIT 5
```



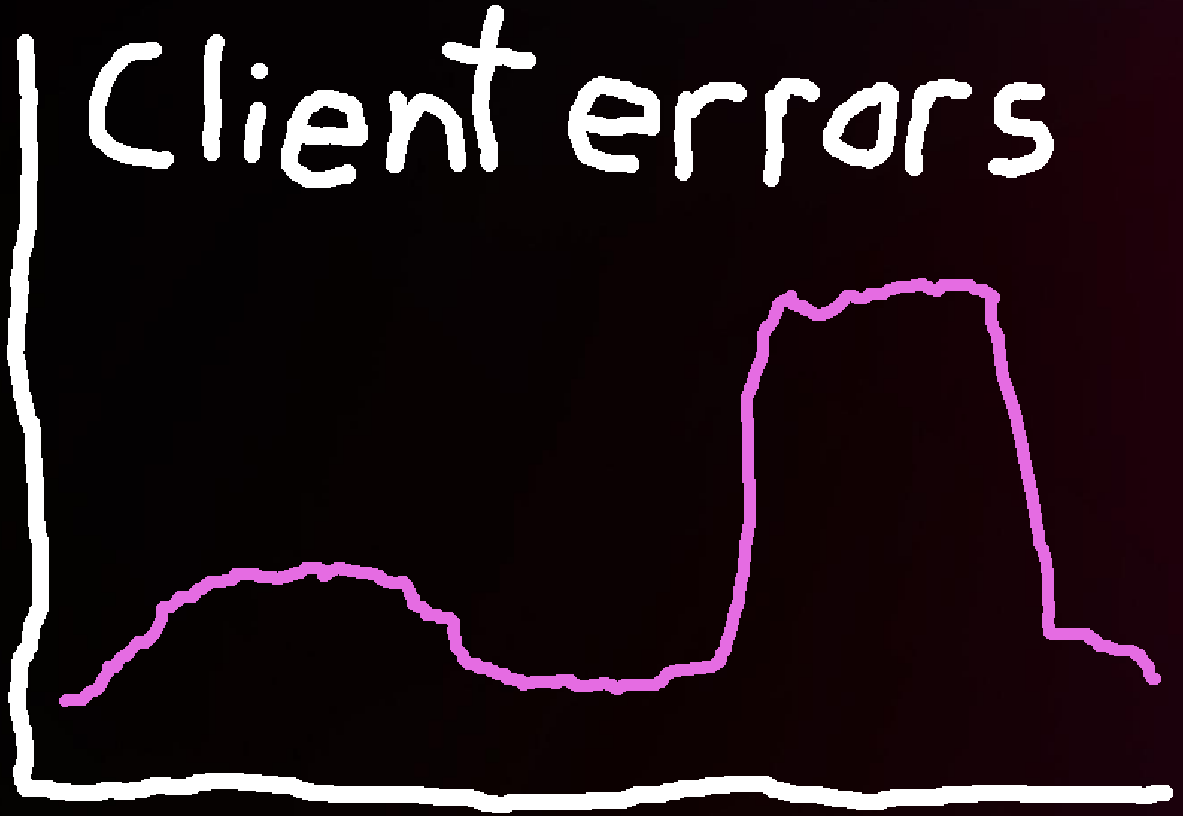
...

(many more)

CloudWatch Contributor Insights

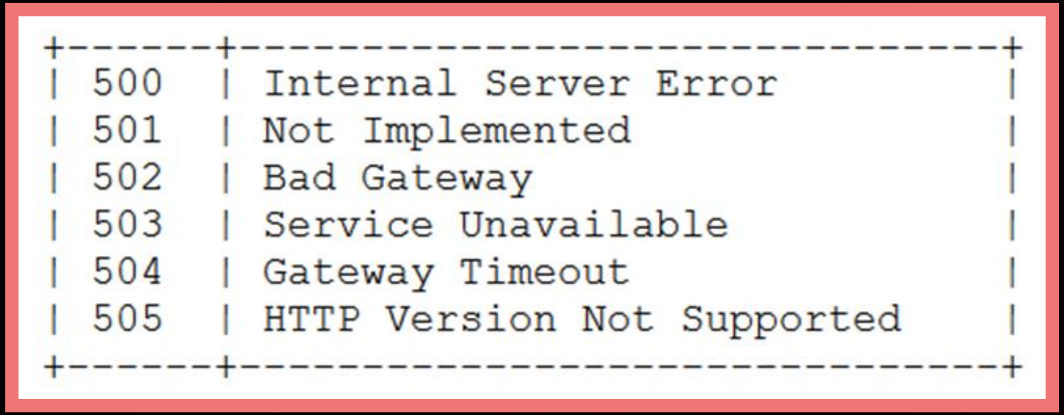


Combining dimensionality and cardinality



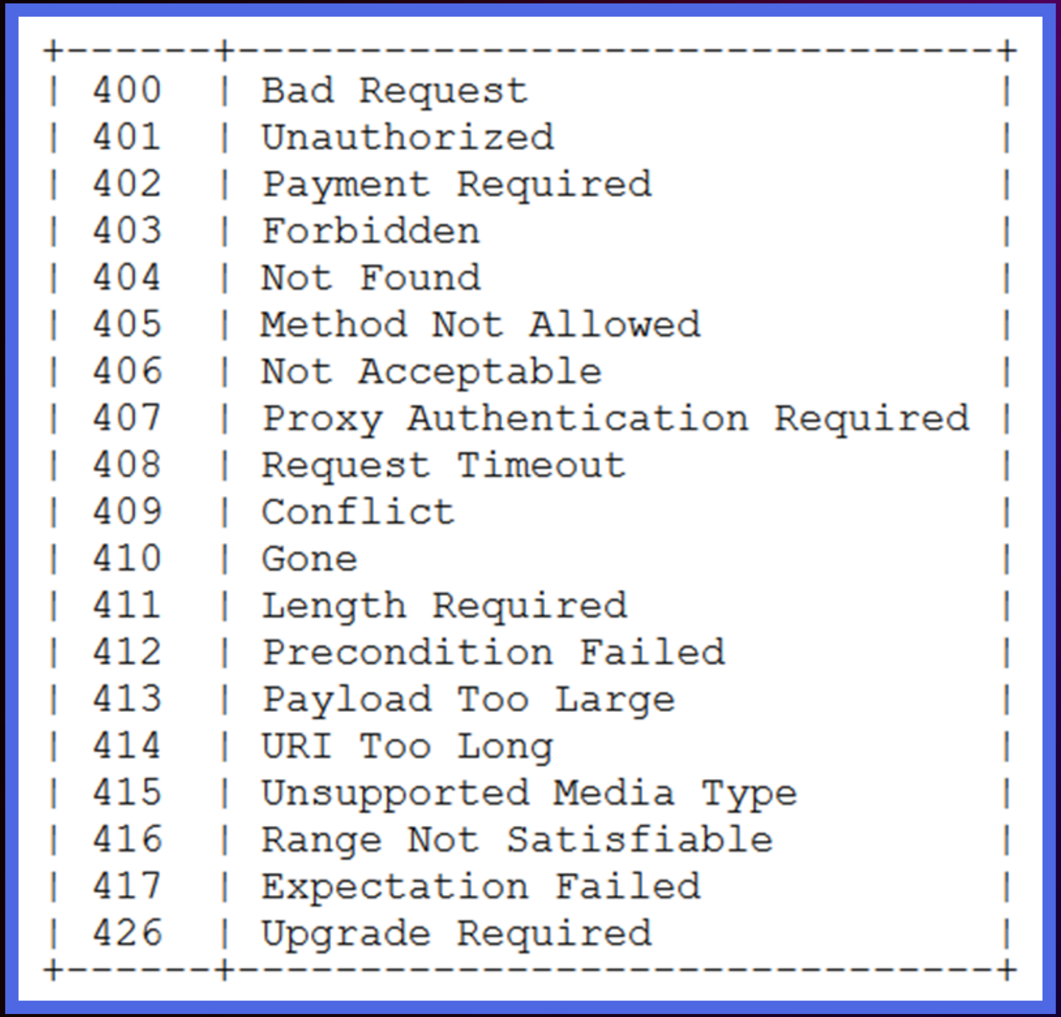
Client vs. server fault

Server fault (5XX)



500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Client fault (4XX)



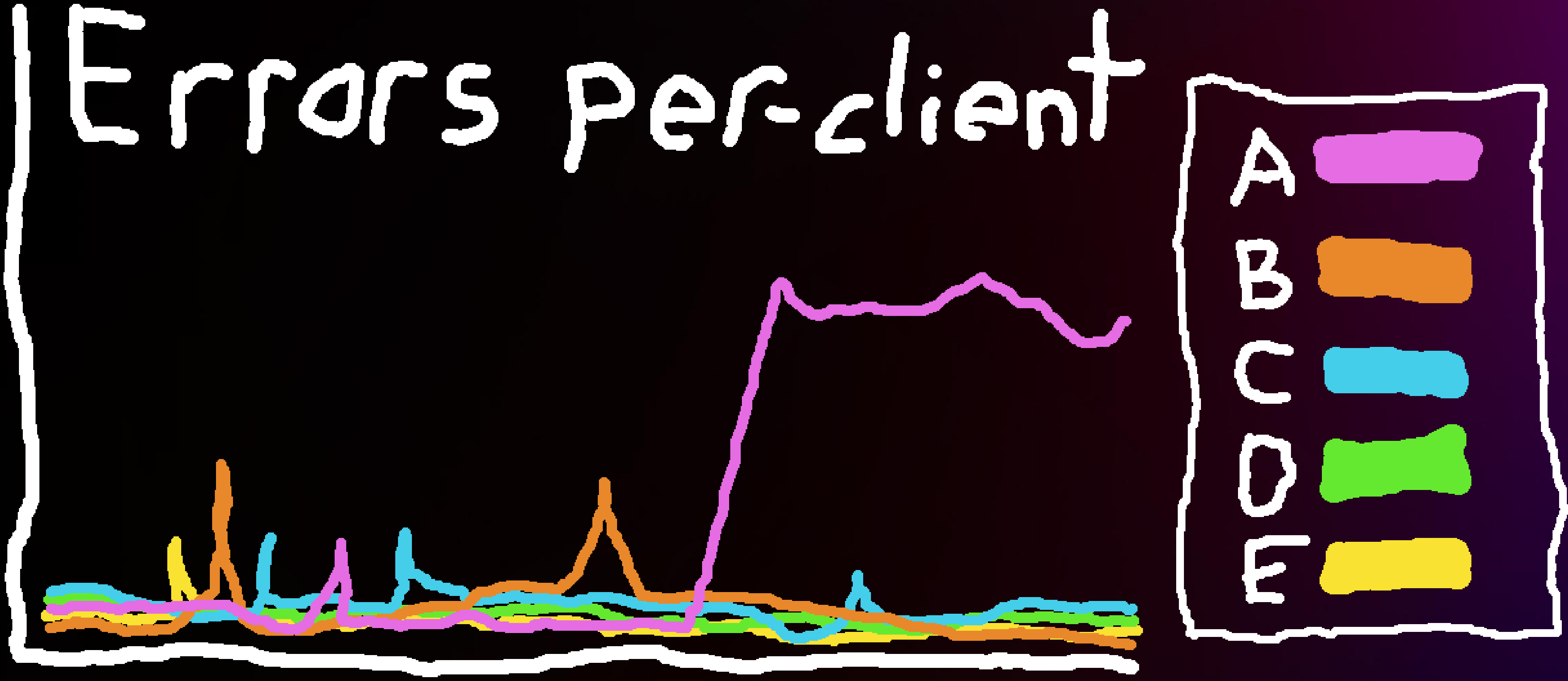
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	URI Too Long
415	Unsupported Media Type
416	Range Not Satisfiable
417	Expectation Failed
426	Upgrade Required

Input validation bugs

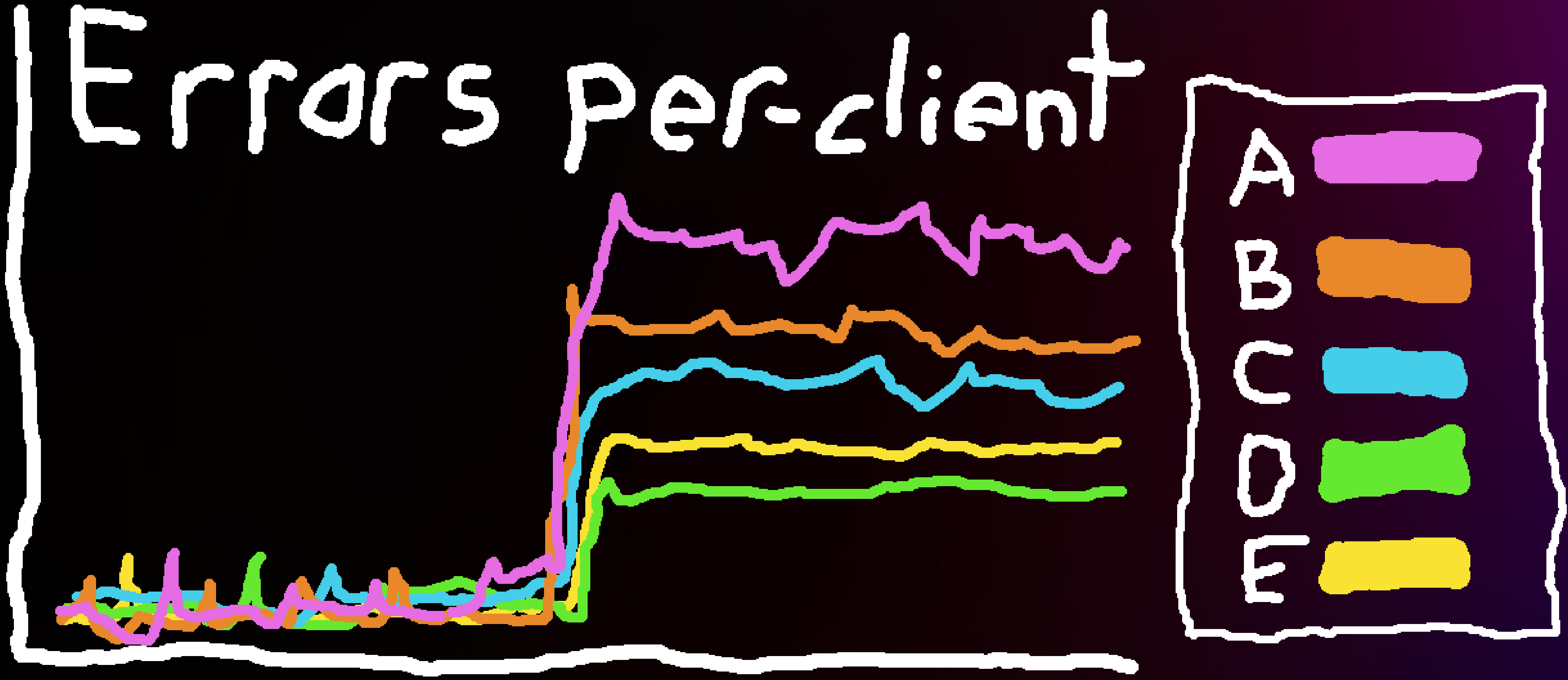
(Bug: Making input validation **more** strict)

```
   public void validate() {
-      if(input.length() > 140) {
+      if(input.length() > 100) {
         throw new ClientError();
       }
   }
```

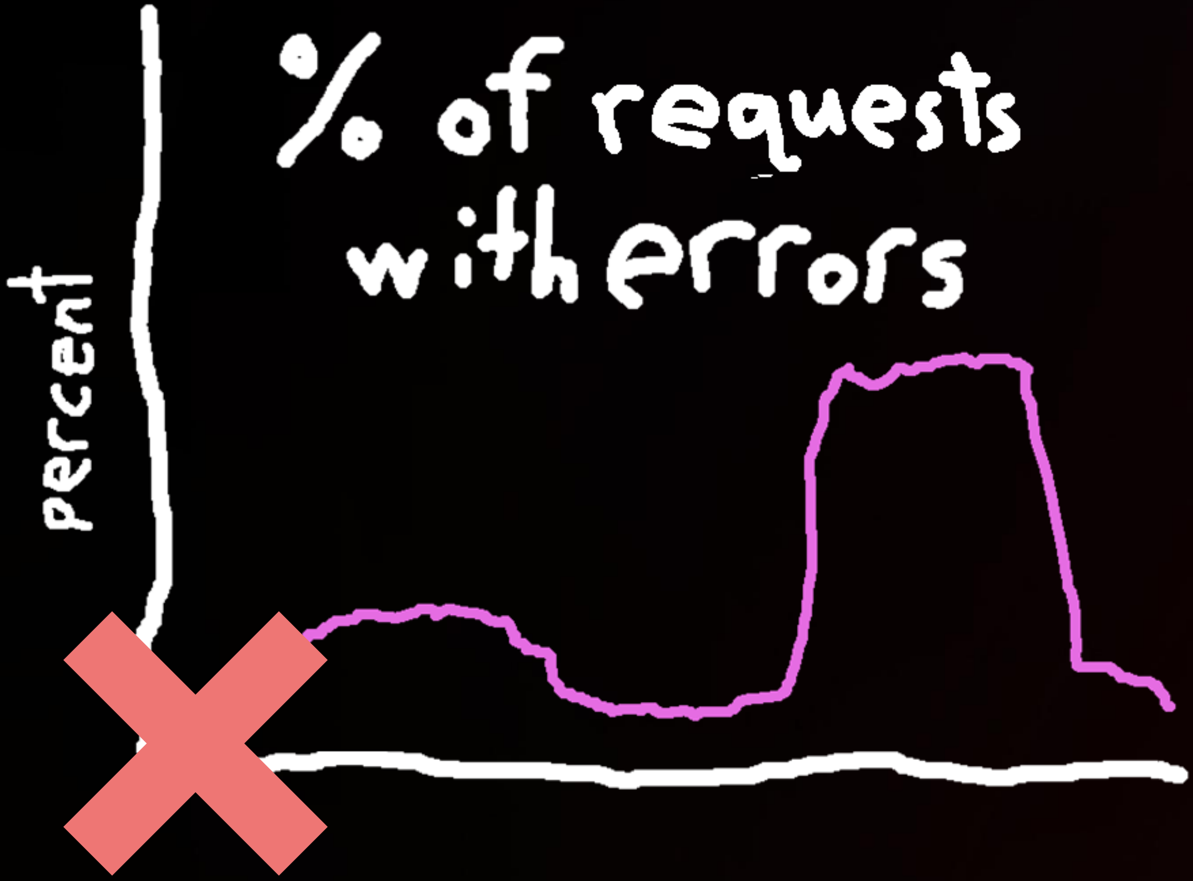
Graphing top contributors

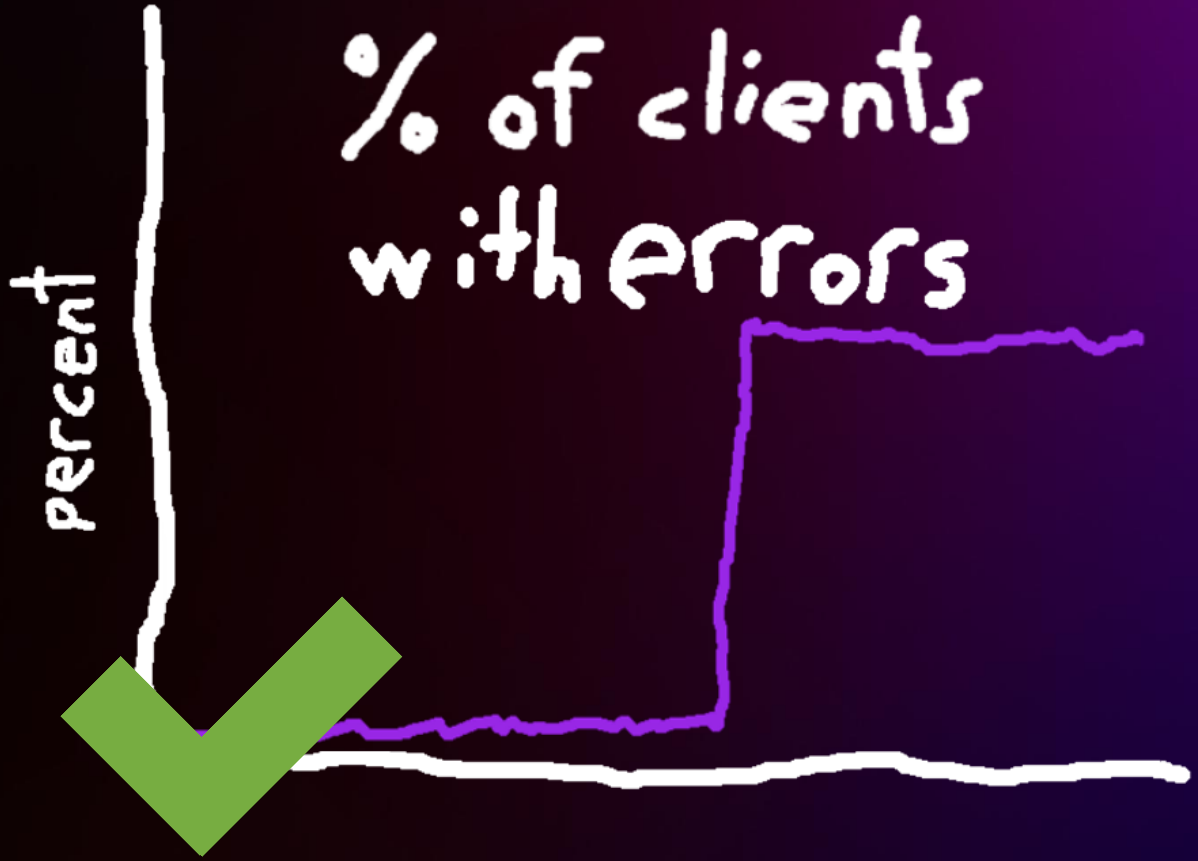


Graphing top contributors



Metrics from another dimension



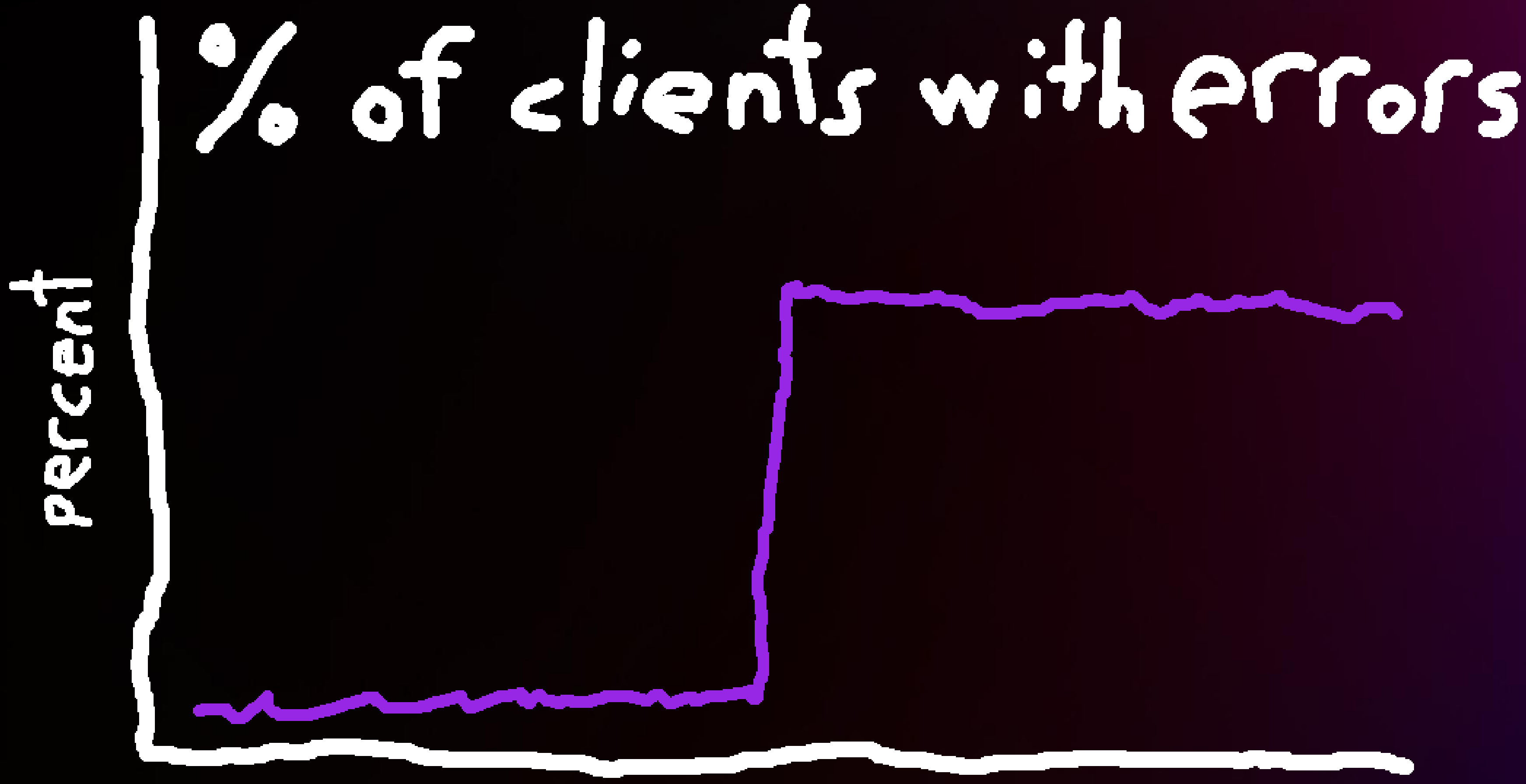


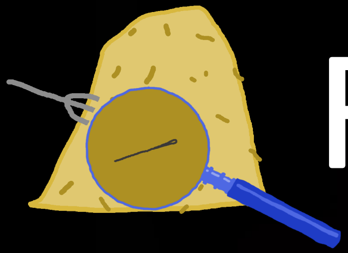
Dimensionality and cardinality together

$$\frac{\#\ of\ customers\ with\ errors}{\#\ of\ customers}$$

$$\frac{INSIGHT\_RULE\_METRIC(ErrorsPerAccount, UniqueContributors)}{INSIGHT\_RULE\_METRIC(RequestsPerAccount, UniqueContributors)}$$

Metrics from another dimension

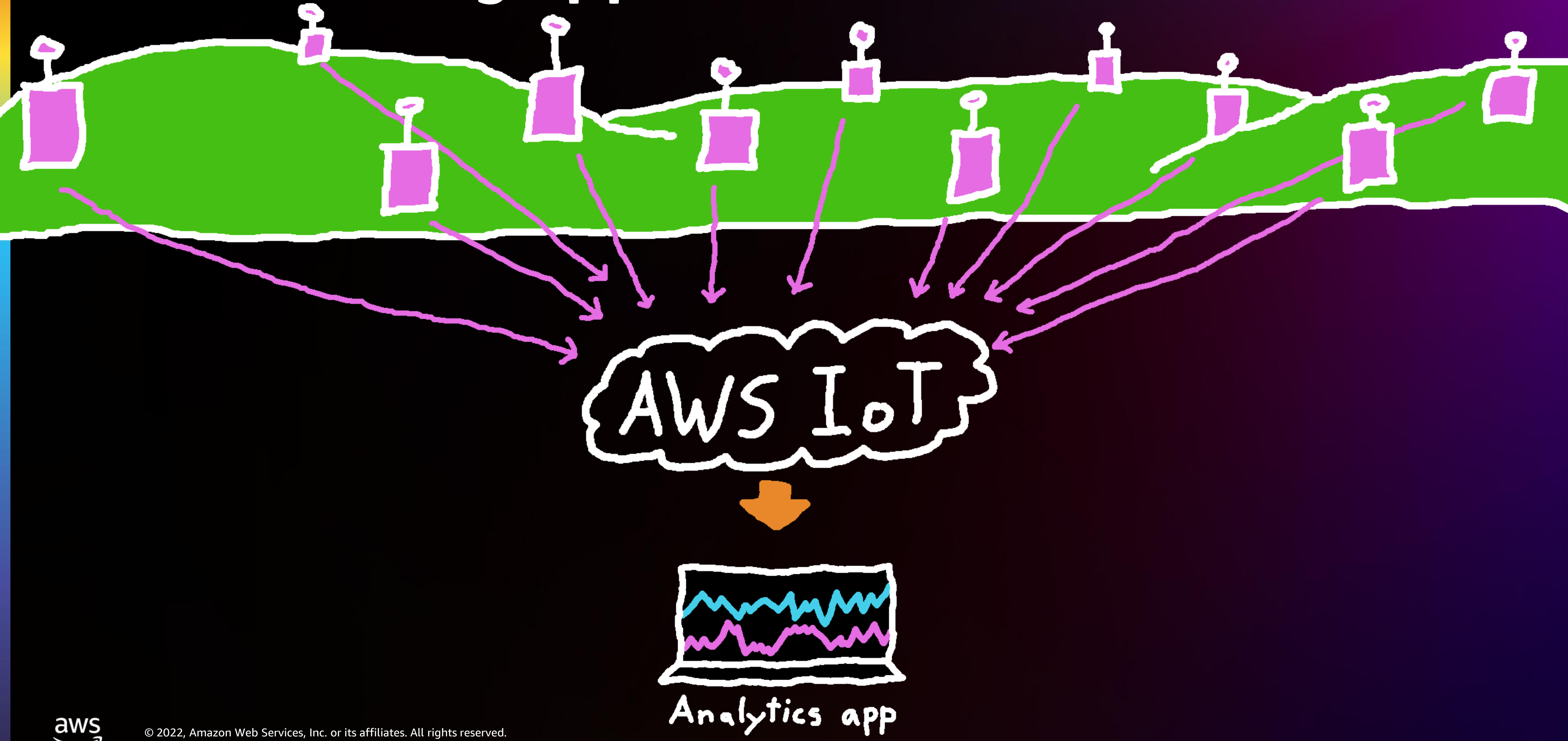




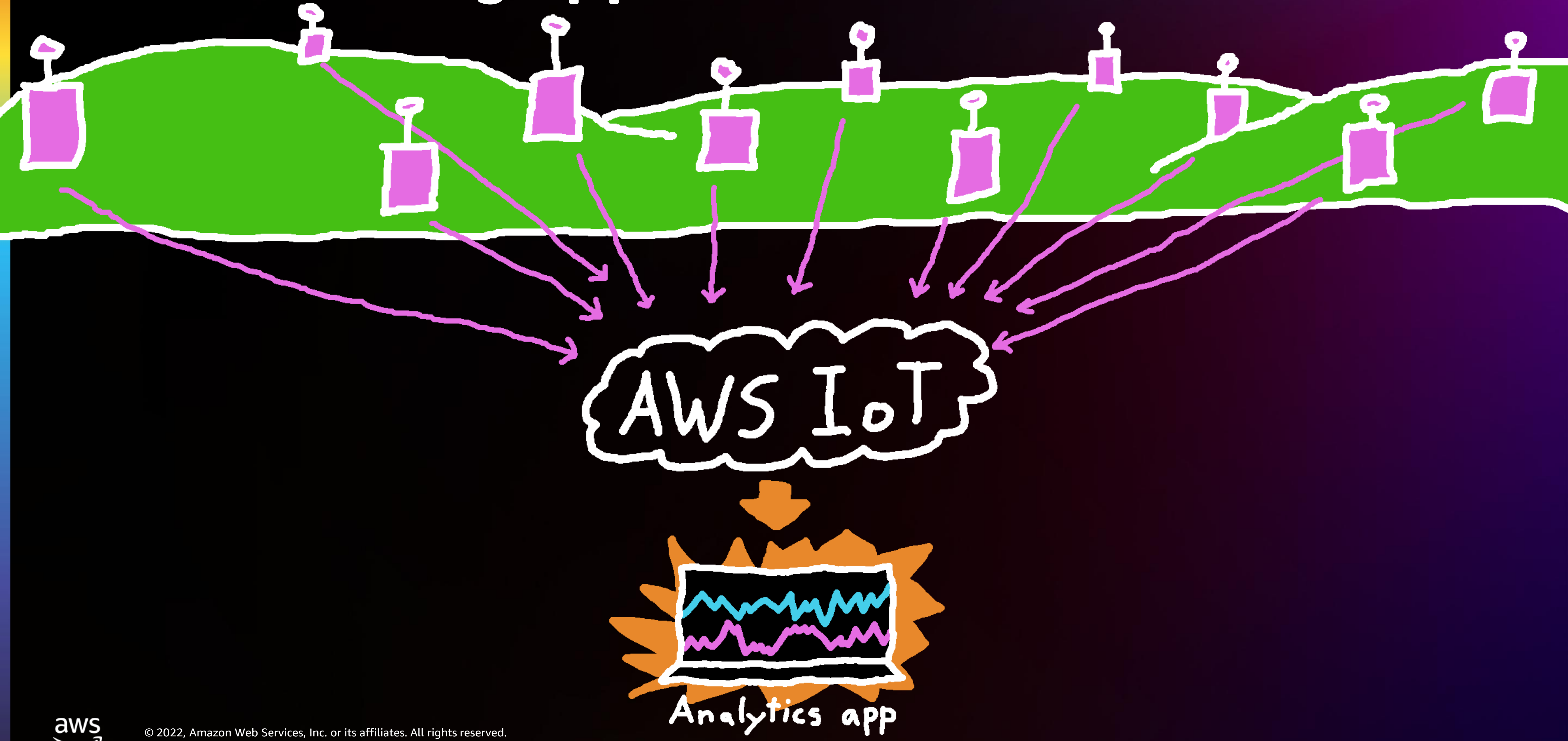
Finding the root cause

- Propagate trace information
- Build lots of dashboards
- Get multi-dimensional
- Use high cardinality metrics
- **Dig even deeper with log analysis**
- Look at the raw logs
- Analyze using profiler

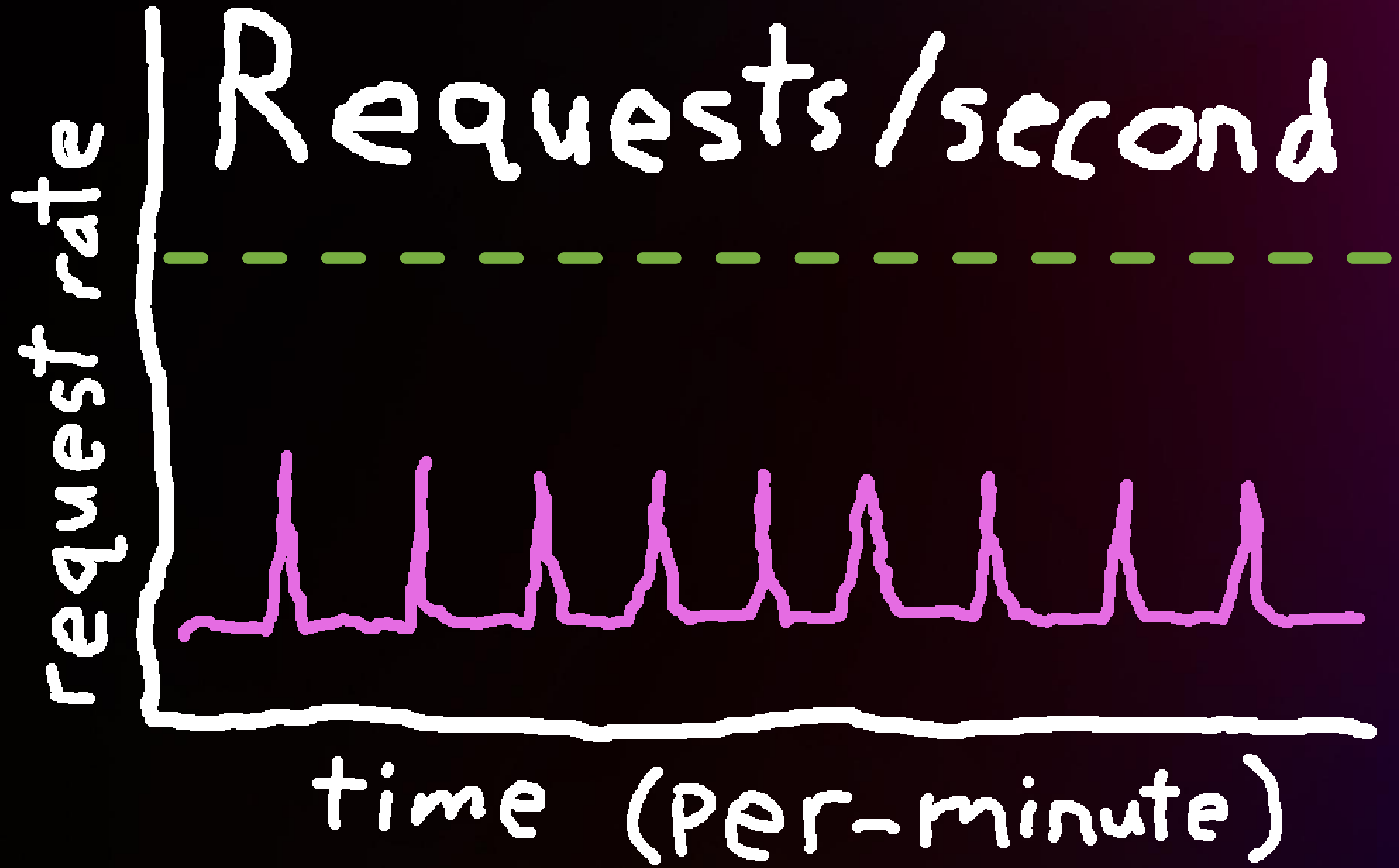
IoT monitoring application



IoT monitoring application



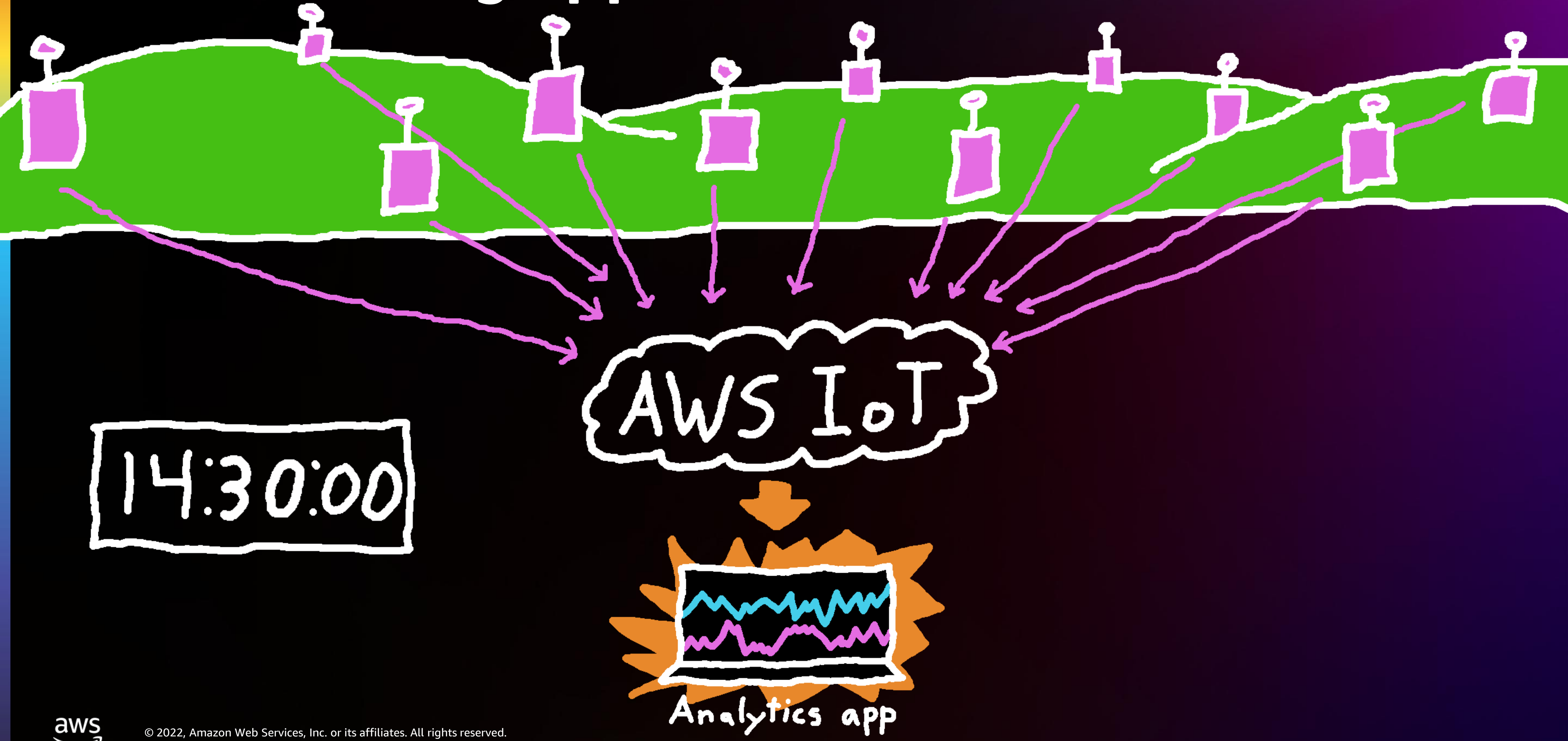
When request rates are a lie



IoT monitoring application

14:29:59
AWS IoT
Analytics app

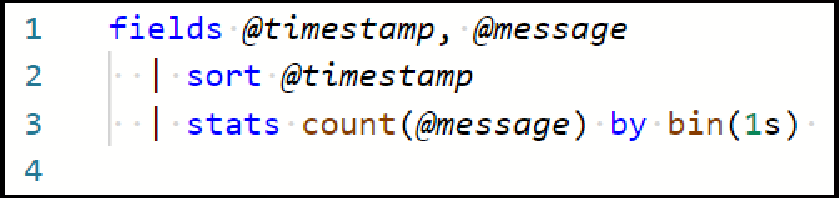
IoT monitoring application



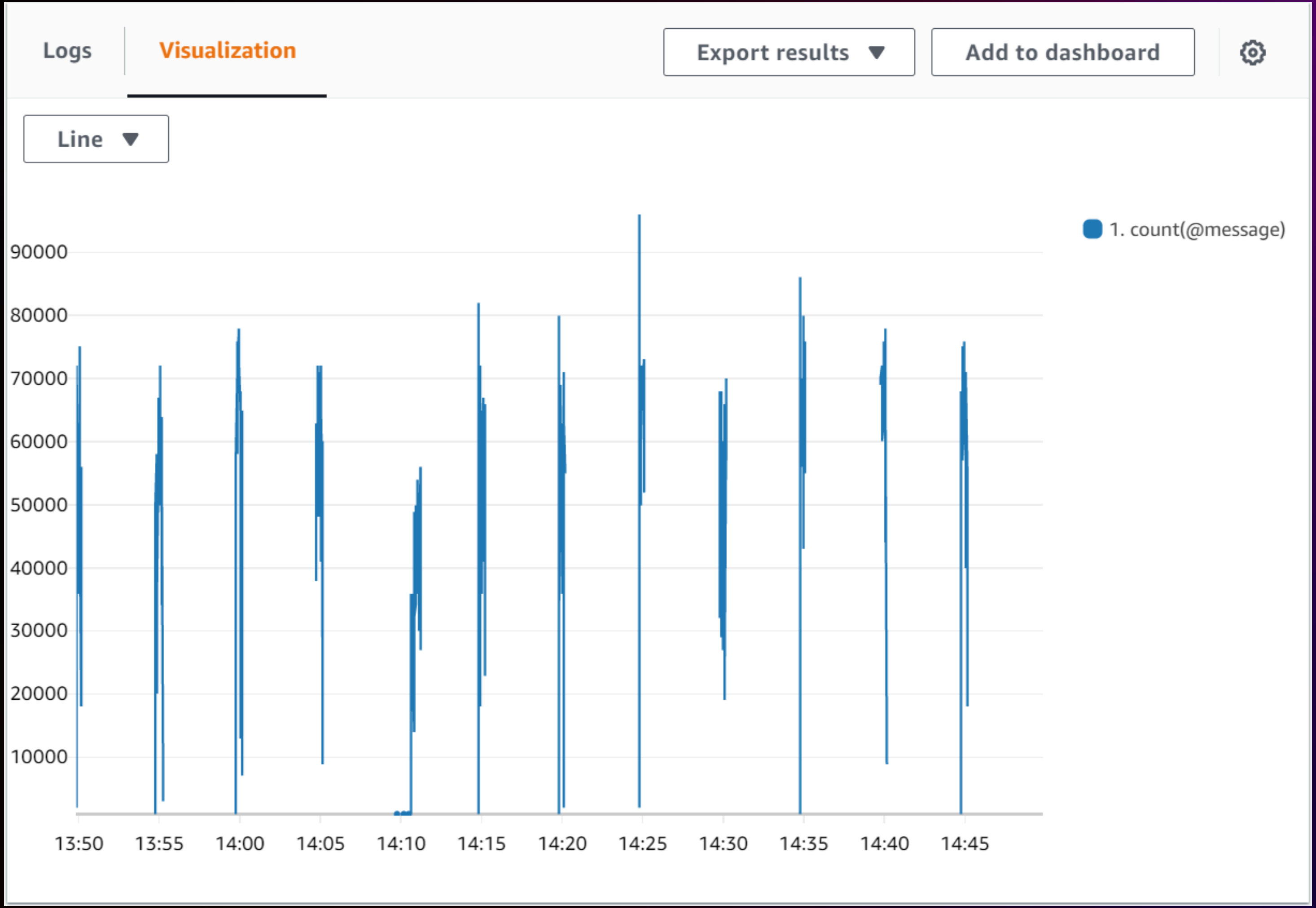
IoT monitoring application

14:30:01
AWS IoT
Analytics app

Slice and dice log analysis



```
fields @timestamp, @message
  | sort @timestamp
  | stats count(@message) by bin(1s)
```



Aggregation defined up front

```
{
  "Operation": "GetProduct",
  "CustomerId": "user@example.com",
  "ProductId": "BrightOrangeSneakers",
  "ClientIp": "192.168.131.39",
  "InstanceId": "i-001234a4bf70dec41EXAMPLE",
  "AvailabiltyZone": "us-west-2a",
  "Cache.Node": "10.0.2.182",
  "Cache.Hit": 0,
  "Cache.Time": 1,
  "DB.Time": 10,
  "Error": 0,
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

Aggregation defined up front

```
{
  "Operation": "GetProduct",
  "CustomerId": "user@example.com",
  "ProductId": "BrightOrangeSneakers",
  "ClientIp": "192.168.131.39",
  "InstanceId": "i-001234a4bf70dec
  "AvailabiltyZone": "us-west-2a",
  "Cache.Node": "10.0.2.182",
  "Cache.Hit": 0,
  "Cache.Time": 1,
  "DB.Time": 10,
  "Error": 0,
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

“Show me latency per-operation”

```
"Dimensions": [["Operation"]],
"Metrics": [
  { "Name": "Time",
    "Unit": "Milliseconds" }]
```

One-off aggregation

```
{
  "Operation": "GetProduct",
  "CustomerId": "user@example.com",
  "ProductId": "BrightOrangeSneakers",
  "ClientIp": "192.168.131.39",
  "InstanceId": "i-001234a4bf70dec
  "AvailabiltyZone": "us-west-2a"
  "Cache.Node": "10.0.2.182",
  "Cache.Hit": 0,
  "Cache.Time": 1,
  "DB.Time": 10,
  "Error": 0,
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

"Show me the top 20 products that are returning errors for requests from IP address 192.168.131.39"

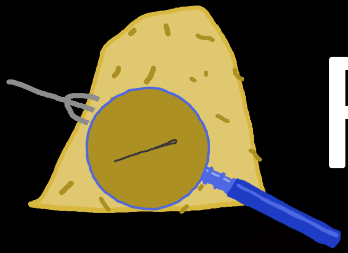
```
filter ClientIp='192.168.131.39' and Error > 0
| stats count() as errorCount by ProductId
| sort errorCount desc
| limit 20
```

One-off aggregation

```
{
  "Operation": "GetProduct",
  "CustomerId": "user@example.com",
  "ProductId": "BrightOrangeSneakers",
  "ClientIp": "192.168.131.39",
  "InstanceId": "i-001234a4bf70dec
  "AvailabiltyZone": "us-west-2a",
  "Cache.Node": "10.0.2.182",
  "Cache.Hit": 0,
  "Cache.Time": 1,
  "DB.Time": 10,
  "Error": 0,
  "Time": 100,
  "TraceId": "Root=1-5759e988-bd862e3fe1be46a994272793;Sampled=1"
}
```

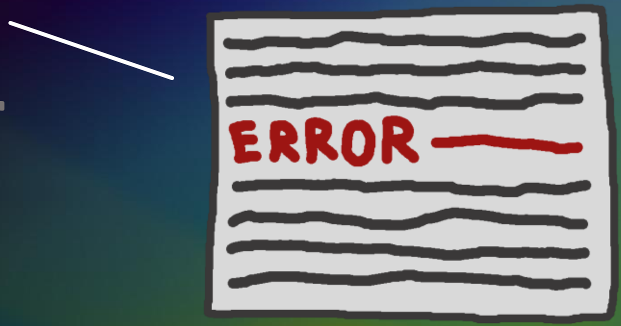
“Show me the products with the lowest cache hit rate, and which cache node they landed on

```
stats (sum(Cache.Hit) / count()) as cacheHitRate
    by ProductId, Cache.Node
| sort cacheHitRate
| limit 20
```

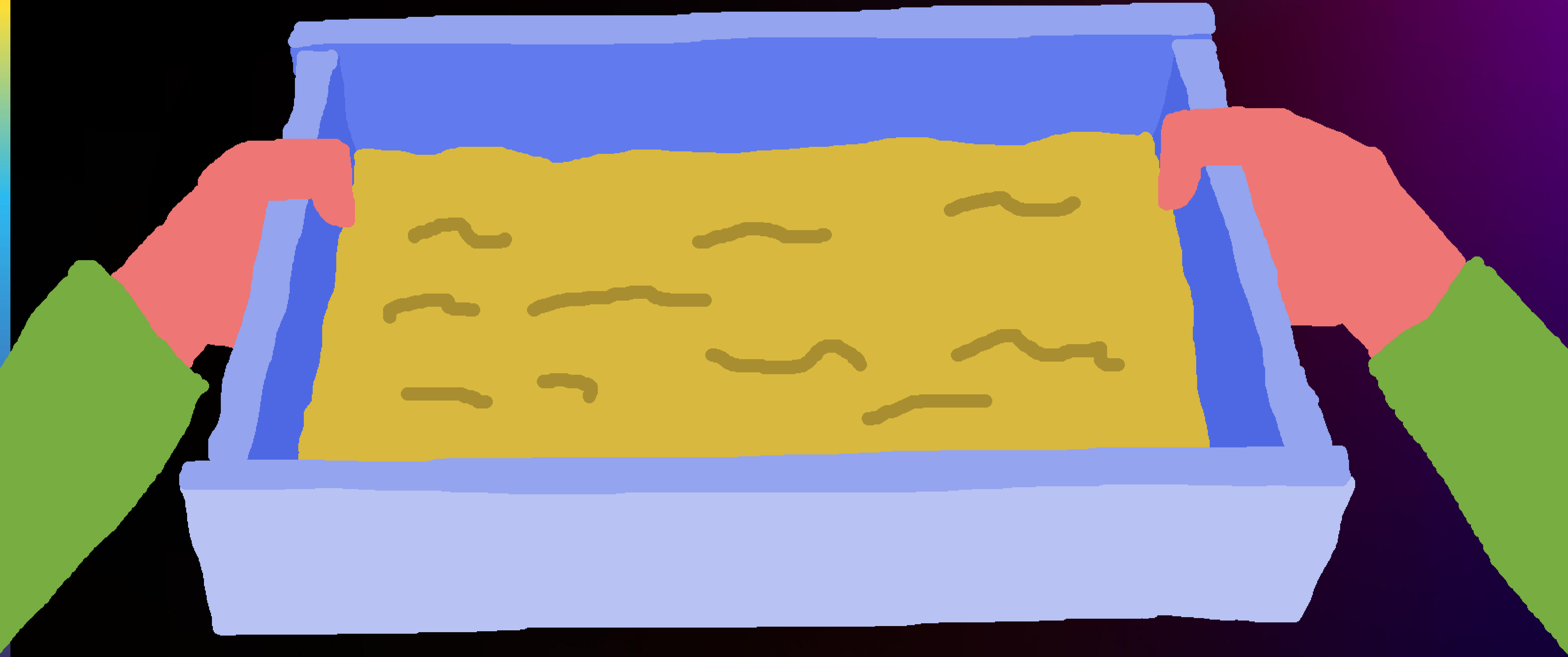


Finding the root cause

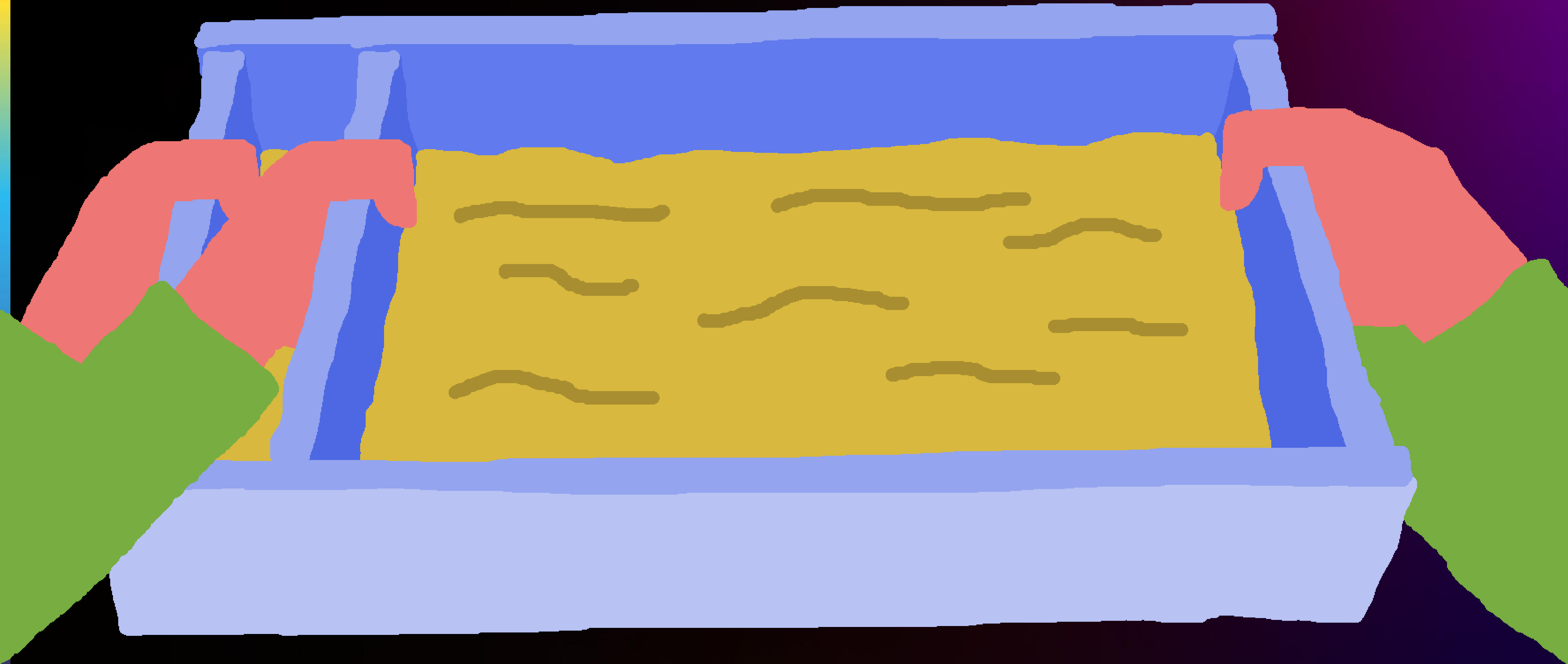
- Propagate trace information
- Build lots of dashboards
- Get multi-dimensional
- Use high cardinality metrics
- Dig even deeper with log analysis
- **Look at the raw logs**
- Analyze using profiler



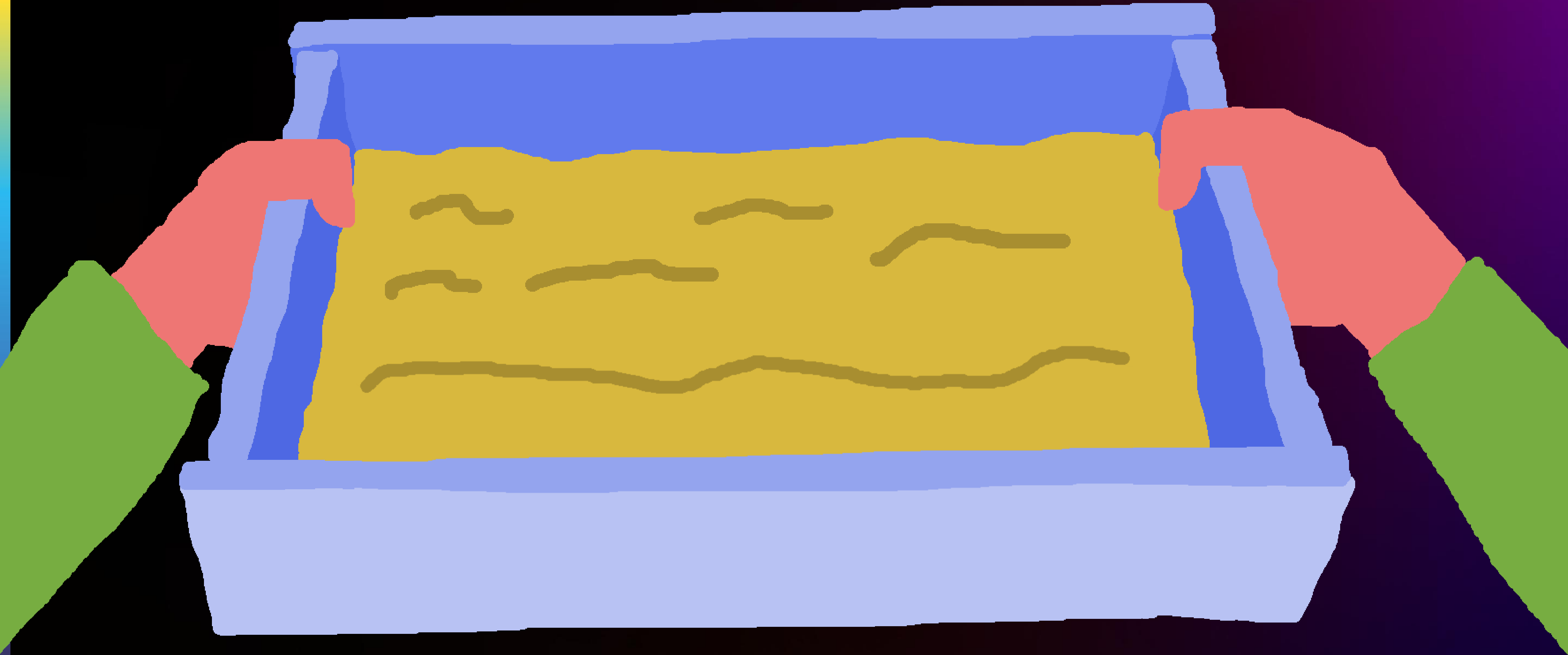
Sifting through logs



Sifting through logs



Sifting through logs



Raw logs

"I'm seeing errors talking to Amazon DynamoDB."

"Did the request make it to DynamoDB?"

```
java.net.UnknownHostException: dynamodb.us-east-1.amazonaws.com:
unknown error
  at java.net.Inet6AddressImpl.lookupAllHostAddr(Native Method)
  at java.net.InetAddress$2.lookupAllHostAddr(InetAddress.java:928)
  at java.net.InetAddress.getAddressesFromNameService(InetAddress.java:1323)
  at java.net.InetAddress.getAllByName0(InetAddress.java:1276)
  at java.net.InetAddress.getAllByName(InetAddress.java:1192)
  at java.net.InetAddress.getAllByName(InetAddress.java:1126)
  at com.amazonaws.SystemDefaultDnsResolver.resolve(SystemDefaultDnsResolver.j
```

Raw logs

“I’m seeing errors talking to Amazon DynamoDB.”

“What did DynamoDB say back?”

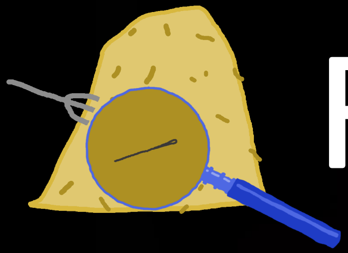
```
com.amazonaws.services.dynamodbv2.model.ConditionalCheckFailedException:
The conditional request failed (Service: AmazonDynamoDBv2; Status Code: 400;
Error Code: ConditionalCheckFailedException; Request ID: ...)
  at com.amazonaws.http.AmazonHttpClient.handleErrorResponse(AmazonHttpClient.
  at com.amazonaws.http.AmazonHttpClient.executeOneRequest(AmazonHttpClient.ja
  at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.java:4
  at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:302)
  at com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient.invoke(AmazonDynam
  at com.amazonaws.services.dynamodbv2.AmazonDynamoDBClient.updateItem(AmazonD
```

Raw logs

"I'm seeing errors talking to Amazon DynamoDB."

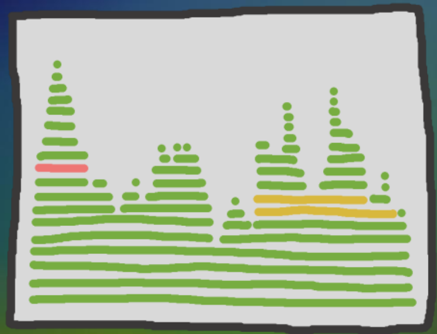
"What happened right before that?"

```
[WARN]  2022-11-14T20:37:58.617Z   DynamoDB call timed, out; retrying 1/3
```



Finding the root cause

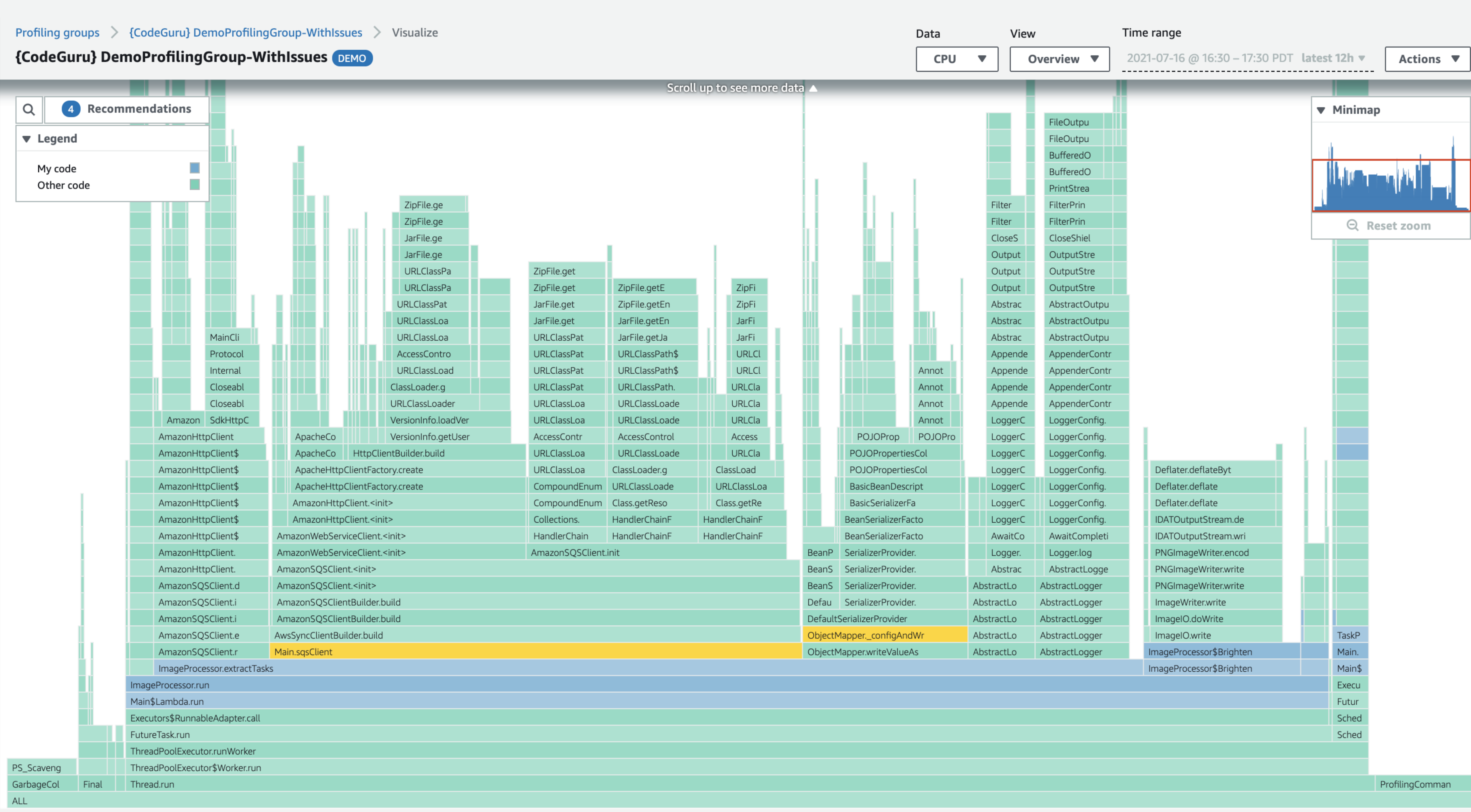
- Propagate trace information
- Build lots of dashboards
- Get multi-dimensional
- Use high cardinality metrics
- Dig even deeper with log analysis
- Look at the raw logs
- **Analyze using profiler**



Example: Inefficient code

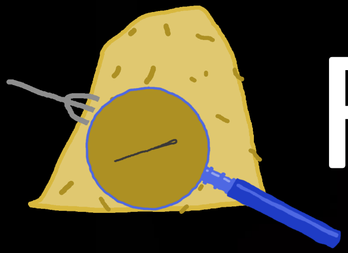
```
public ProductInfo getProductInfo(String customerId, String productId) {

  ProductInfo info = cache.get(customerId, productId);

  // compute pi just to make sure
  // TODO refactor or optimize
  for (int i = 0; i < 10000; i++) {
    calculatePI();
  }

  if (info == null) {
    info = db.query(customerId, productId);
  }

  return info;
}
```

No timer around this part of the code


Profiling groups > {CodeGuru} DemoProfilingGroup-WithIssues > Visualize

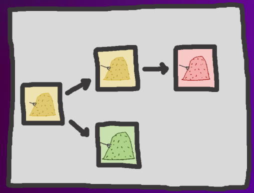
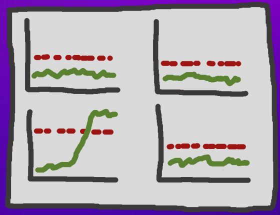
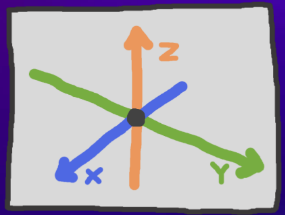
{CodeGuru} DemoProfilingGroup-WithIssues DEMO

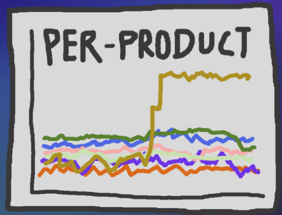
Data: CPU ▼ View: Overview ▼ Time range: 2021-07-16 @ 16:30 – 17:30 PDT latest 12h ▼ Actions ▼

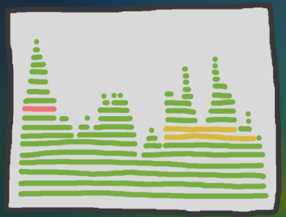


Finding the root cause

- Propagate trace information
- Build lots of dashboards
- Get multi-dimensional
- Use high cardinality metrics
- Dig even deeper with log analysis
- Look at the raw logs
- Analyze using profiler

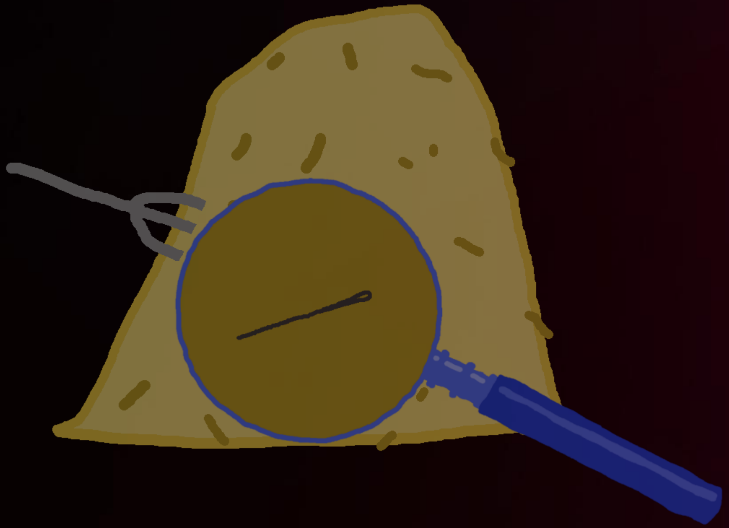








Chapter three

The DevOps
flywheel at Amazon



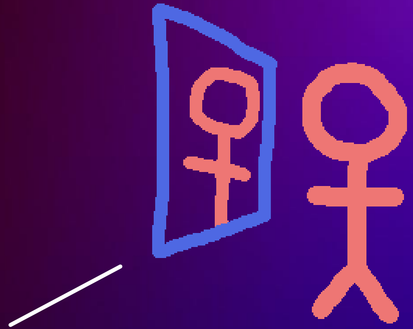
Find the
root cause

Measure from
everywhere

Measure from everywhere

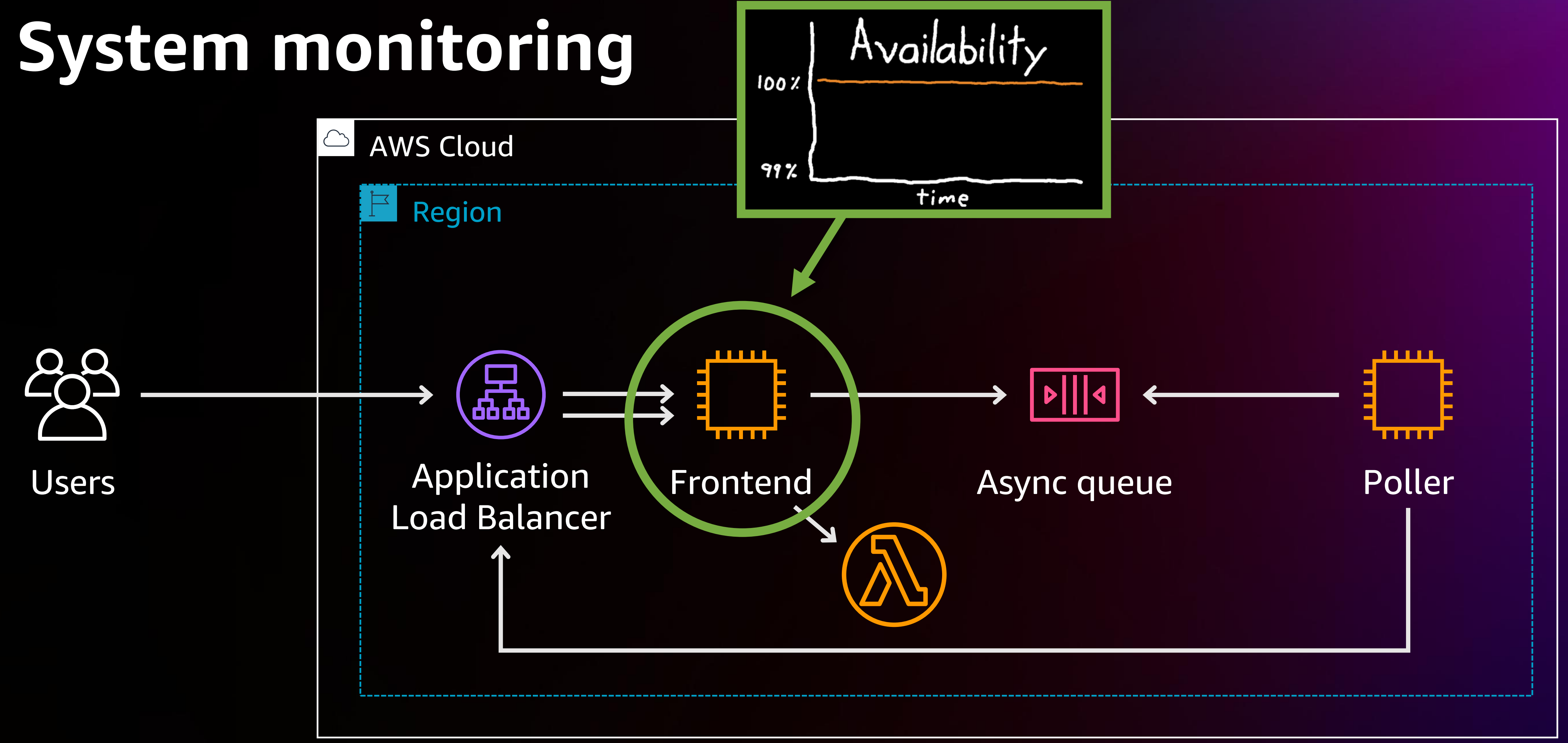
- Be your own customer
- Instrument close to the customer
- Alarm on everything

Measure from everywhere

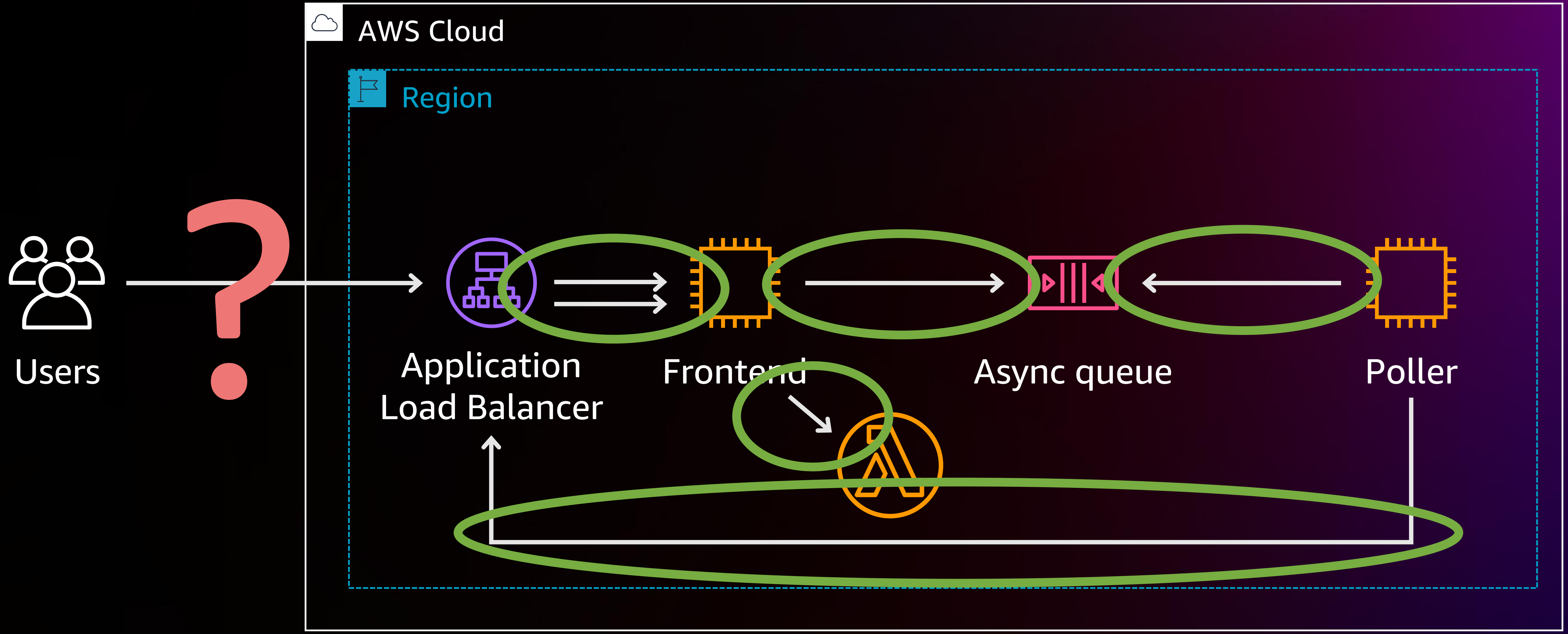


- **Be your own customer**
- Instrument close to the customer
- Alarm on everything

System monitoring



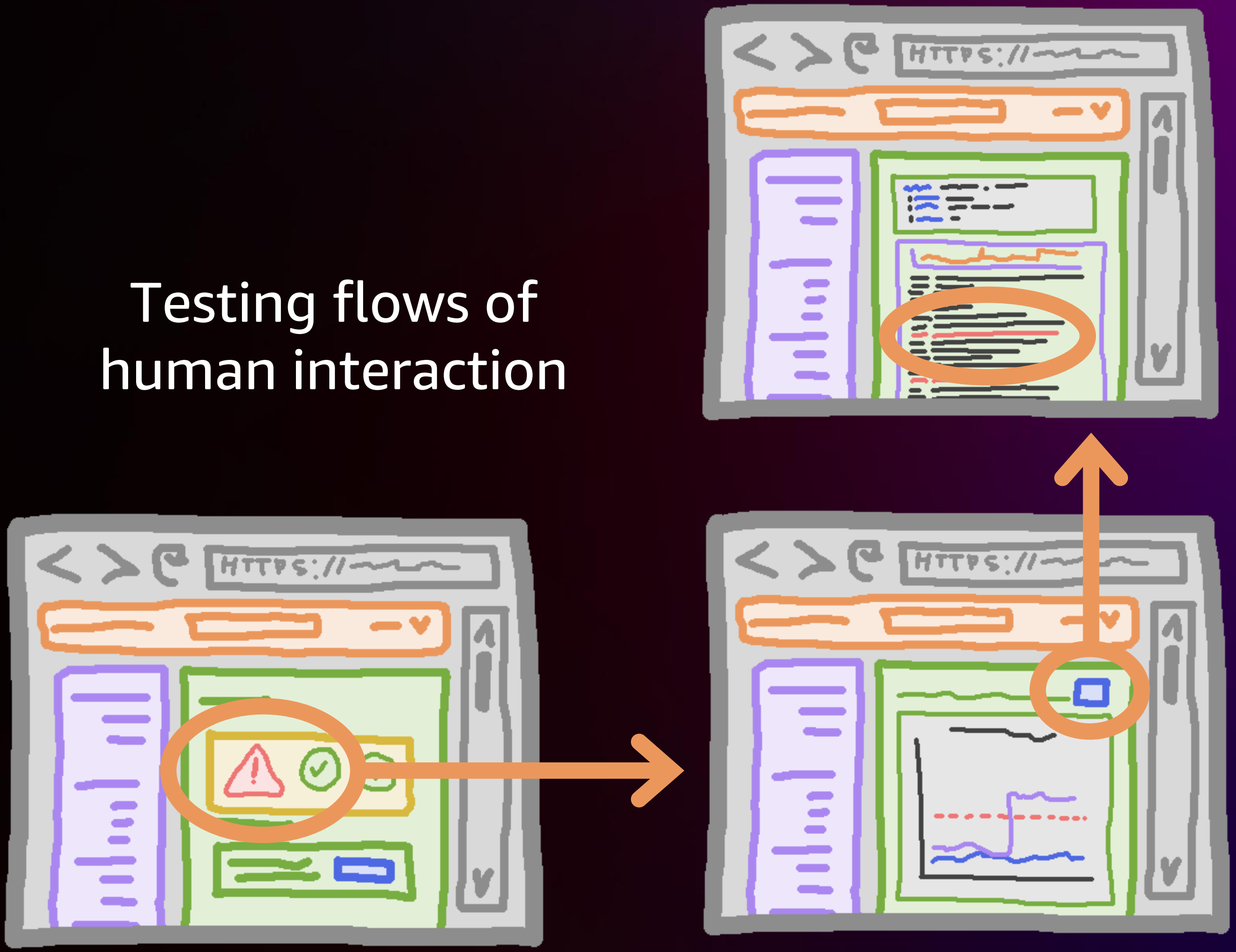
System monitoring



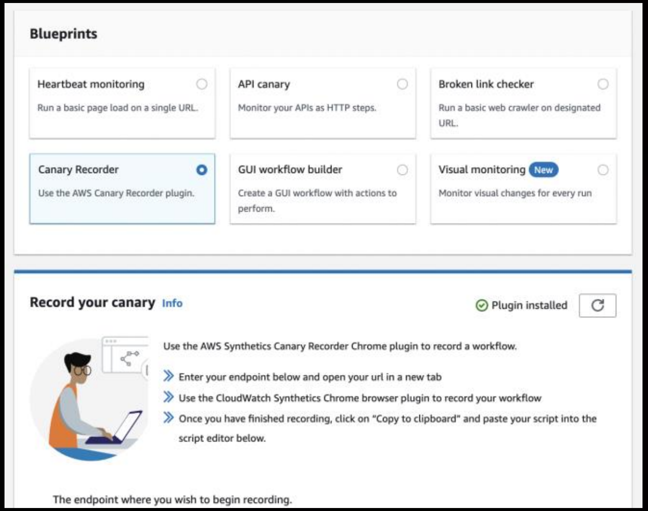
Solving browser testing

Diverse device types, browsers

Testing flows of human interaction



CloudWatch Synthetics



Blueprints

Heartbeat monitoring Run a basic page load on a single URL.	API canary Monitor your APIs as HTTP steps.	Broken link checker Run a basic web crawler on designated URL.
Canary Recorder Use the AWS Canary Recorder plugin.	GUI workflow builder Create a GUI workflow with actions to perform.	Visual monitoring New Monitor visual changes for every run

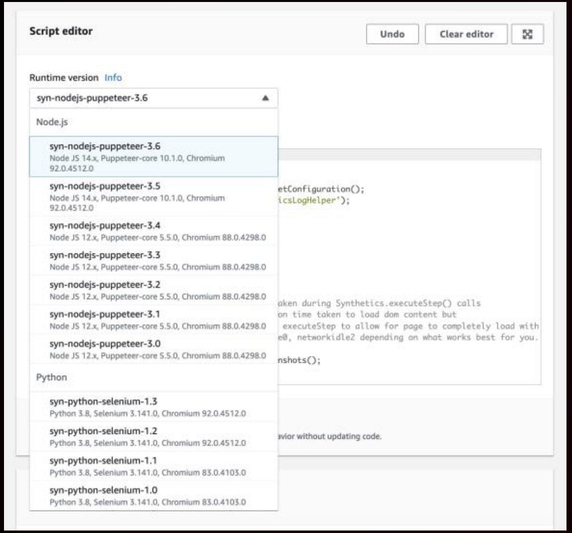
Record your canary Info — Plugin installed

Use the AWS Synthetics Canary Recorder Chrome plugin to record a workflow.

- Enter your endpoint below and open your url in a new tab
- Use the CloudWatch Synthetics Chrome browser plugin to record your workflow
- Once you have finished recording, click on "Copy to clipboard" and paste your script into the script editor below.

The endpoint where you wish to begin recording.

Pre-built blueprint scripts & visual script reorder (chrome plugin)



Script editor

Runtime version Info

syn-nodejs-puppeteer-3.6

Node.js

- syn-nodejs-puppeteer-3.6 — Node JS 14.x, Puppeteer-core 10.1.0, Chromium 92.0.4512.0
- syn-nodejs-puppeteer-3.5 — Node JS 14.x, Puppeteer-core 10.1.0, Chromium 92.0.4512.0
- syn-nodejs-puppeteer-3.4 — Node JS 12.x, Puppeteer-core 5.5.0, Chromium 88.0.4298.0
- syn-nodejs-puppeteer-3.3 — Node JS 12.x, Puppeteer-core 5.5.0, Chromium 88.0.4298.0
- syn-nodejs-puppeteer-3.2 — Node JS 12.x, Puppeteer-core 5.5.0, Chromium 88.0.4298.0
- syn-nodejs-puppeteer-3.1 — Node JS 12.x, Puppeteer-core 5.5.0, Chromium 88.0.4298.0
- syn-nodejs-puppeteer-3.0 — Node JS 12.x, Puppeteer-core 5.5.0, Chromium 88.0.4298.0

Python

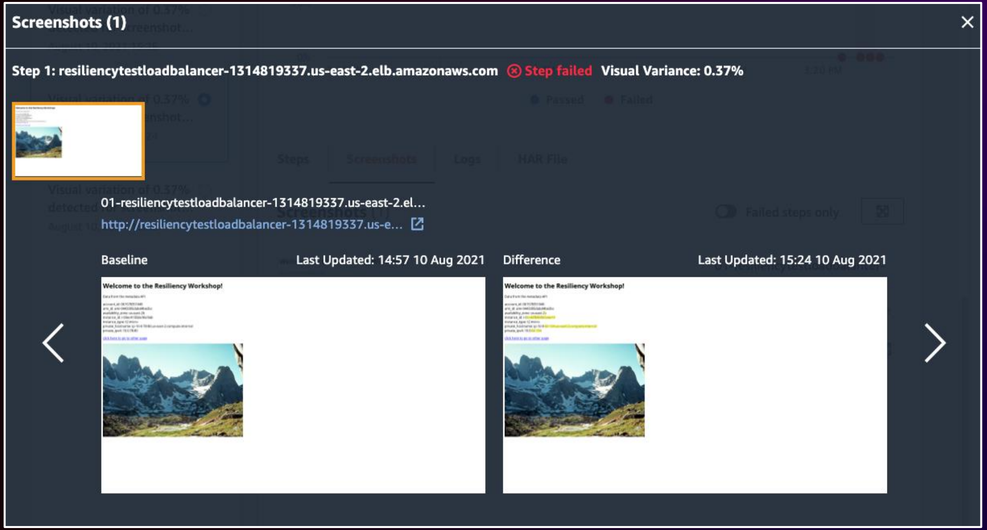
- syn-python-selenium-1.3 — Python 3.8, Selenium 3.141.0, Chromium 92.0.4512.0
- syn-python-selenium-1.2 — Python 3.8, Selenium 3.141.0, Chromium 92.0.4512.0
- syn-python-selenium-1.1 — Python 3.8, Selenium 3.141.0, Chromium 83.0.4103.0
- syn-python-selenium-1.0 — Python 3.8, Selenium 3.141.0, Chromium 83.0.4103.0

```
etConfiguration();
icsLogHelper');

aken during Synthetics.executeStep() calls
on time taken to load dom content but
 executeStep to allow for page to completely load with
e0, networkidle2 depending on what works best for you.

nshots();
```

avior without updating code.

Fully customizable scripting in NodeJS and Python using Selenium & Puppeteer

Visually inspect synthetic failures



Screenshots (1)

Step 1: resiliencytestloadbalancer-1314819337.us-east-2.elb.amazonaws.com Step failed **Visual Variance: 0.37%**

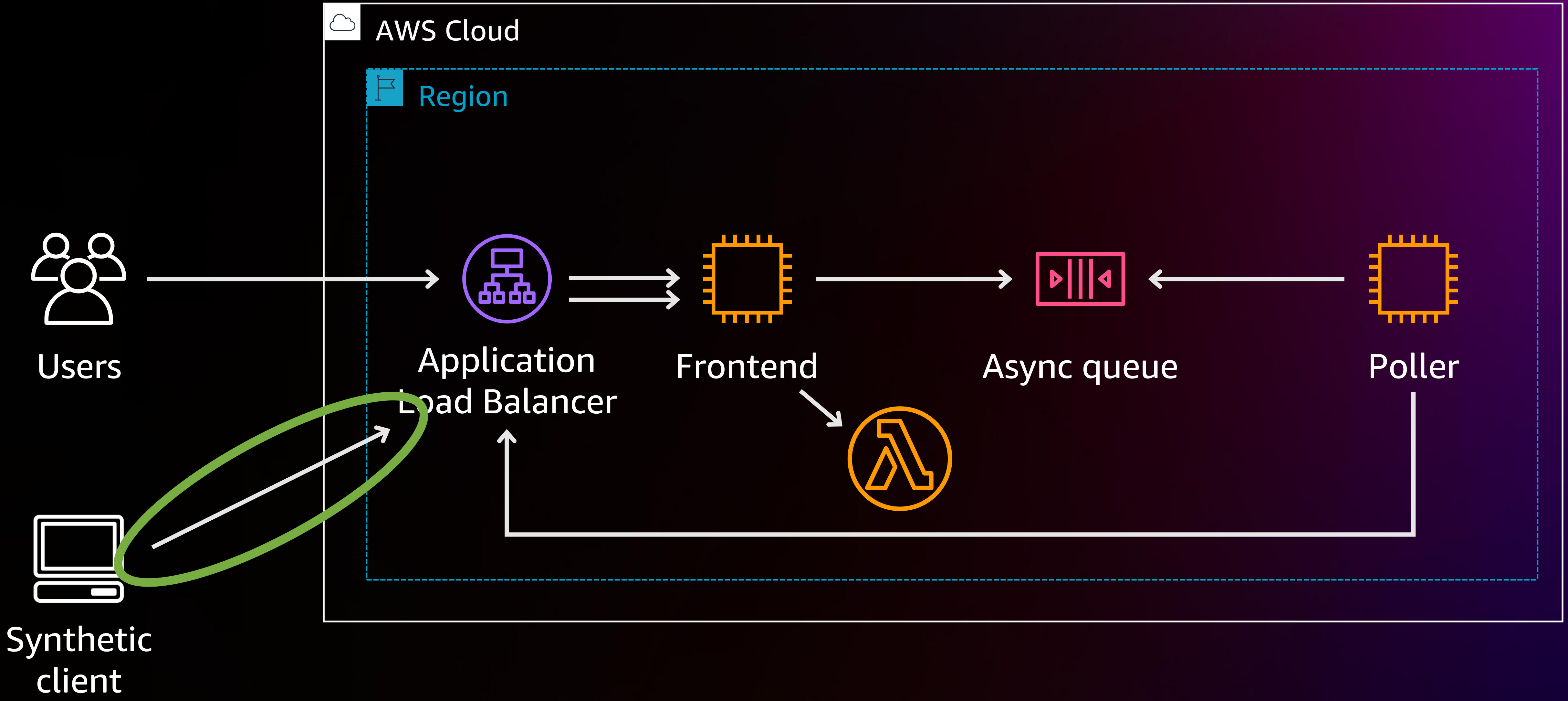
01-resiliencytestloadbalancer-1314819337.us-east-2.el...
http://resiliencytestloadbalancer-1314819337.us-e...

Baseline — Last Updated: 14:57 10 Aug 2021

Difference — Last Updated: 15:24 10 Aug 2021

Welcome to the Resiliency Workshop!
Welcome to the Resiliency Workshop!

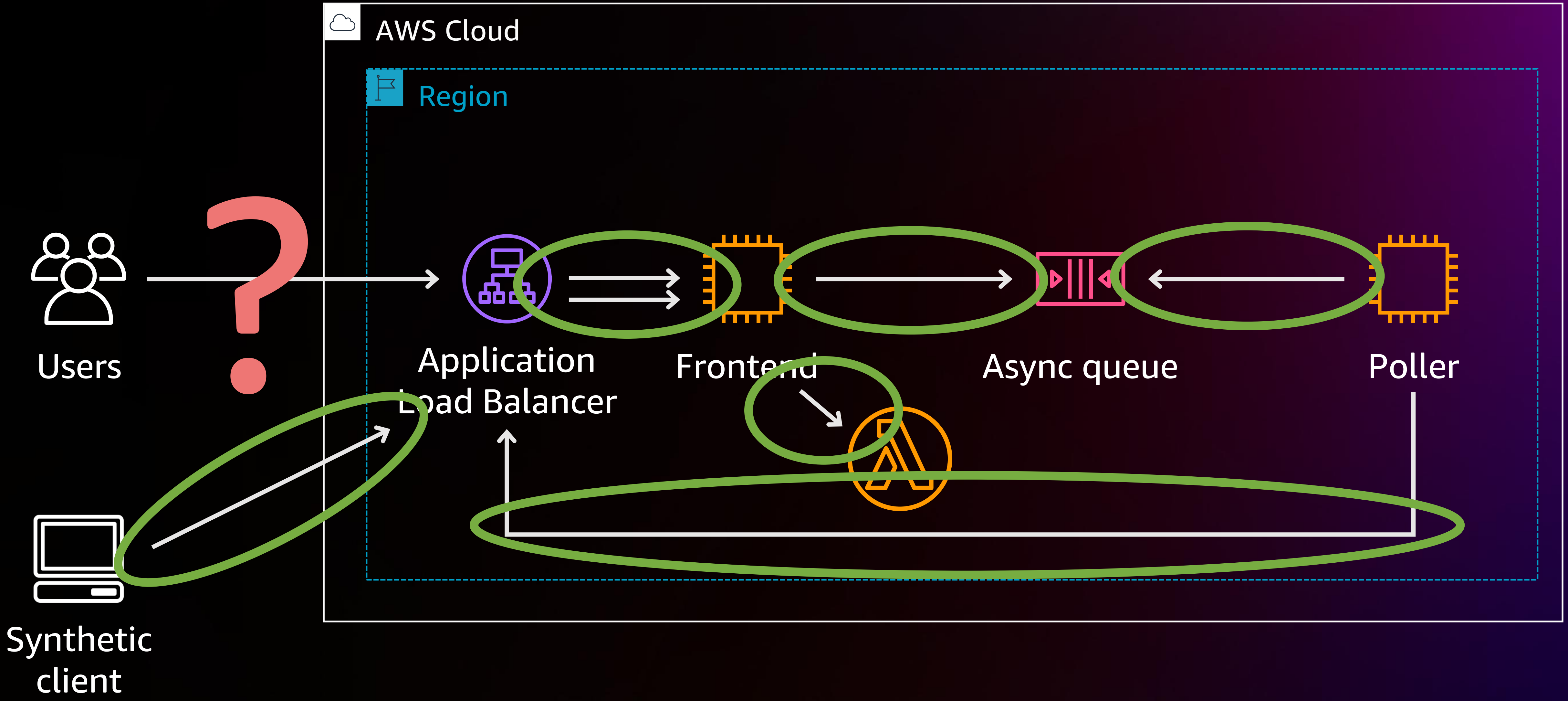
System monitoring



Measure from everywhere

- Be your own customer
- **Instrument close to the customer**
- Alarm on everything

System monitoring



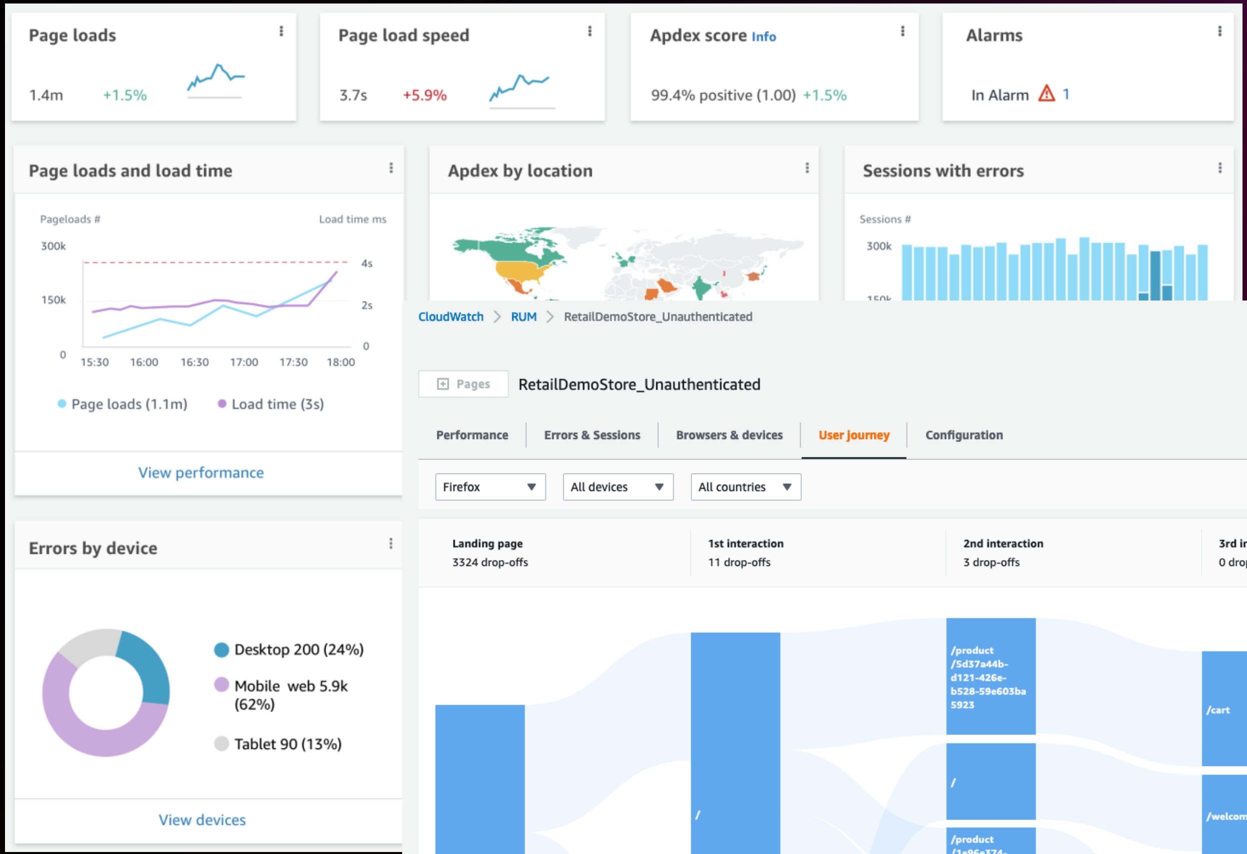
CloudWatch RUM

Quantifying slowness

- Apdex scores
- Errors by sessions, devices, canaries, regions, pages
- Core web vitals
- Largest contentful paint
- First input delay
- Cumulative layout shift

Debugging slowness

- Page load steps
- Resource requests
- View X-Ray traces for errors in RUM sessions
- JavaScript, HTTP errors



Page loads 1.4m +1.5%

Page load speed 3.7s +5.9%

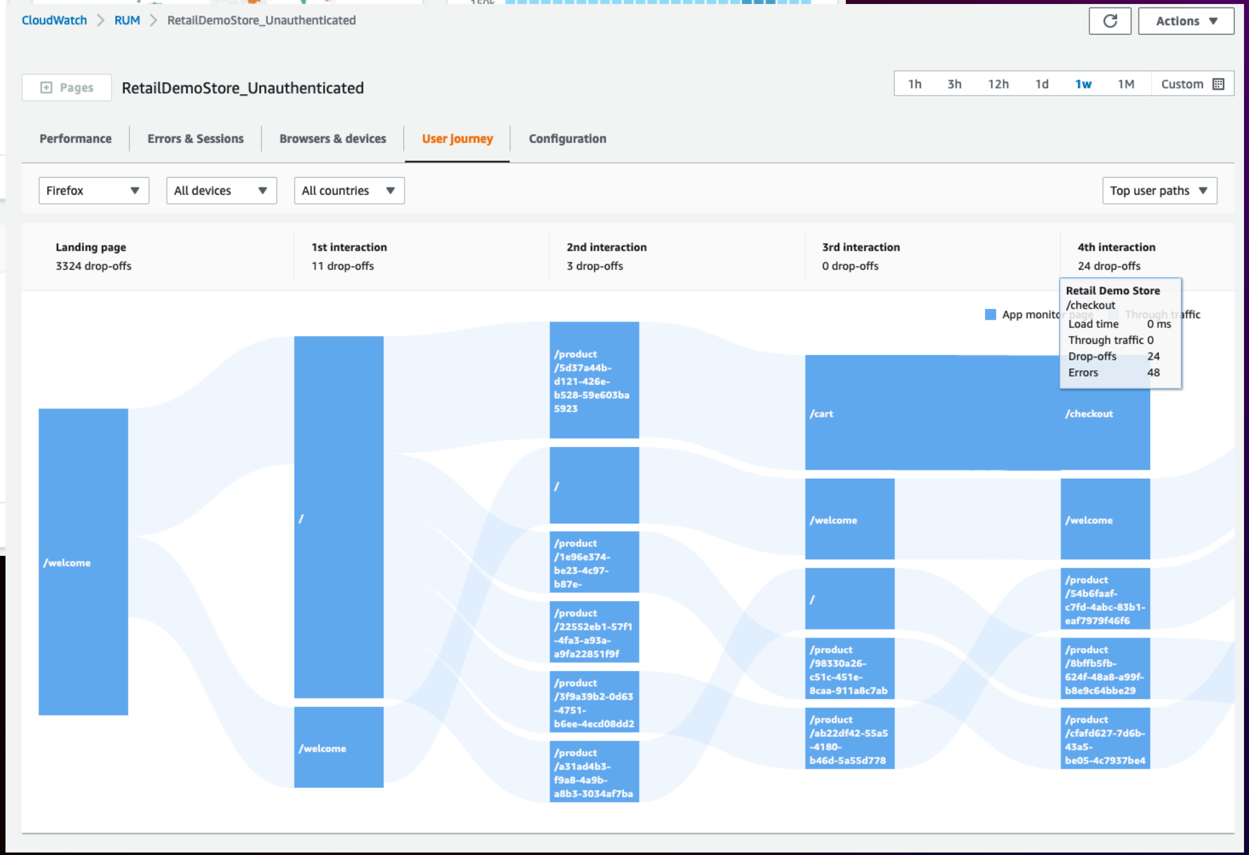
Apdex score Info 99.4% positive (1.00) +1.5%

Alarms In Alarm 1

Desktop 200 (24%)

Mobile web 5.9k (62%)

Tablet 90 (13%)



CloudWatch > RUM > RetailDemoStore_Unauthenticated
Actions
Pages
RetailDemoStore_Unauthenticated
1h
3h
12h
1d
1w
1M
Custom
Performance
Errors & Sessions
Browsers & devices
User journey
Configuration
Firefox
All devices
All countries
Top user paths
App monitor
Through traffic

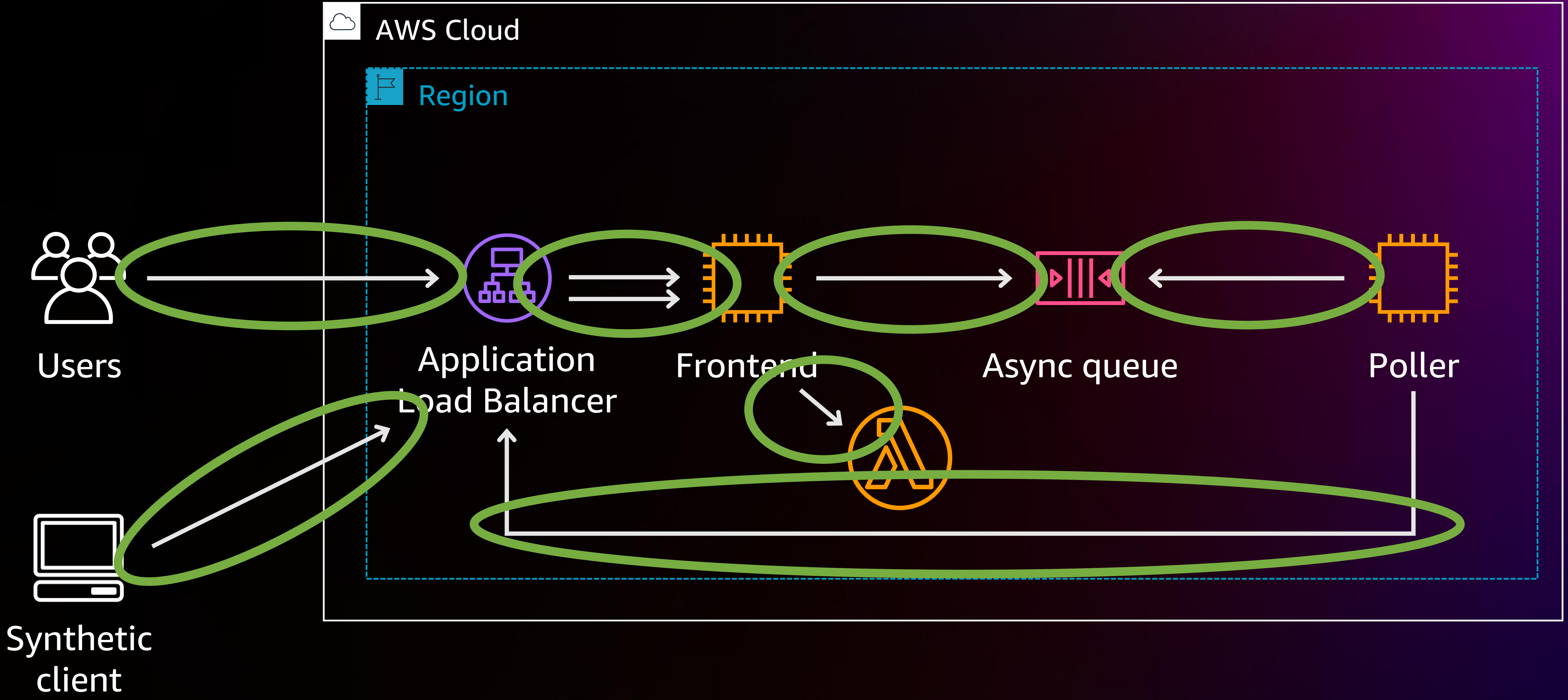
Landing page	1st interaction	2nd interaction	3rd interaction	4th interaction
3324 drop-offs	11 drop-offs	3 drop-offs	0 drop-offs	24 drop-offs

Retail Demo Store /checkout

Load time	0 ms
Through traffic	0
Drop-offs	24
Errors	48

/welcome
/
/welcome
/product /5d37a44b-d121-426e-b528-59e603ba5923
/
/product /1e96e374-be23-4c97-b87e-
/product /22552eb1-57f1-4fa3-a93a-a9fa22851f9f
/product /3f9a39b2-0d63-4751-b6ee-4ecd08dd2
/product /a31ad4b3-f9a8-4a9b-a8b3-3034af7ba
/cart
/welcome
/
/product /98330a26-c51c-451e-8caa-911a8c7ab
/product /ab22df42-55a5-4180-b46d-5a55d778
/checkout
/welcome
/product /54b6faaf-c7fd-4abc-83b1-eaf7979f46f6
/product /8bffb5fb-624f-48a8-a99f-b8e9c64bbe29
/product /cfafd627-7d6b-43a5-be05-4c7937be4

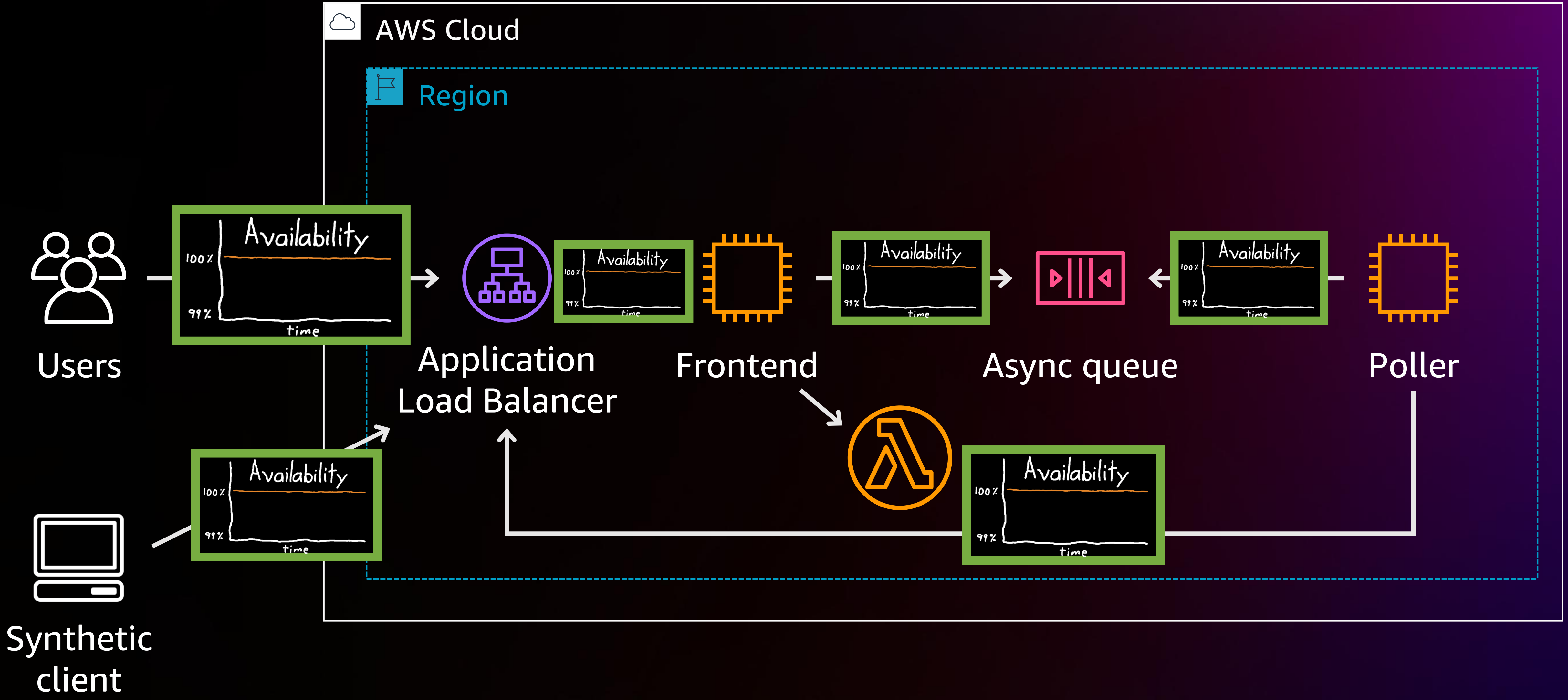
System monitoring



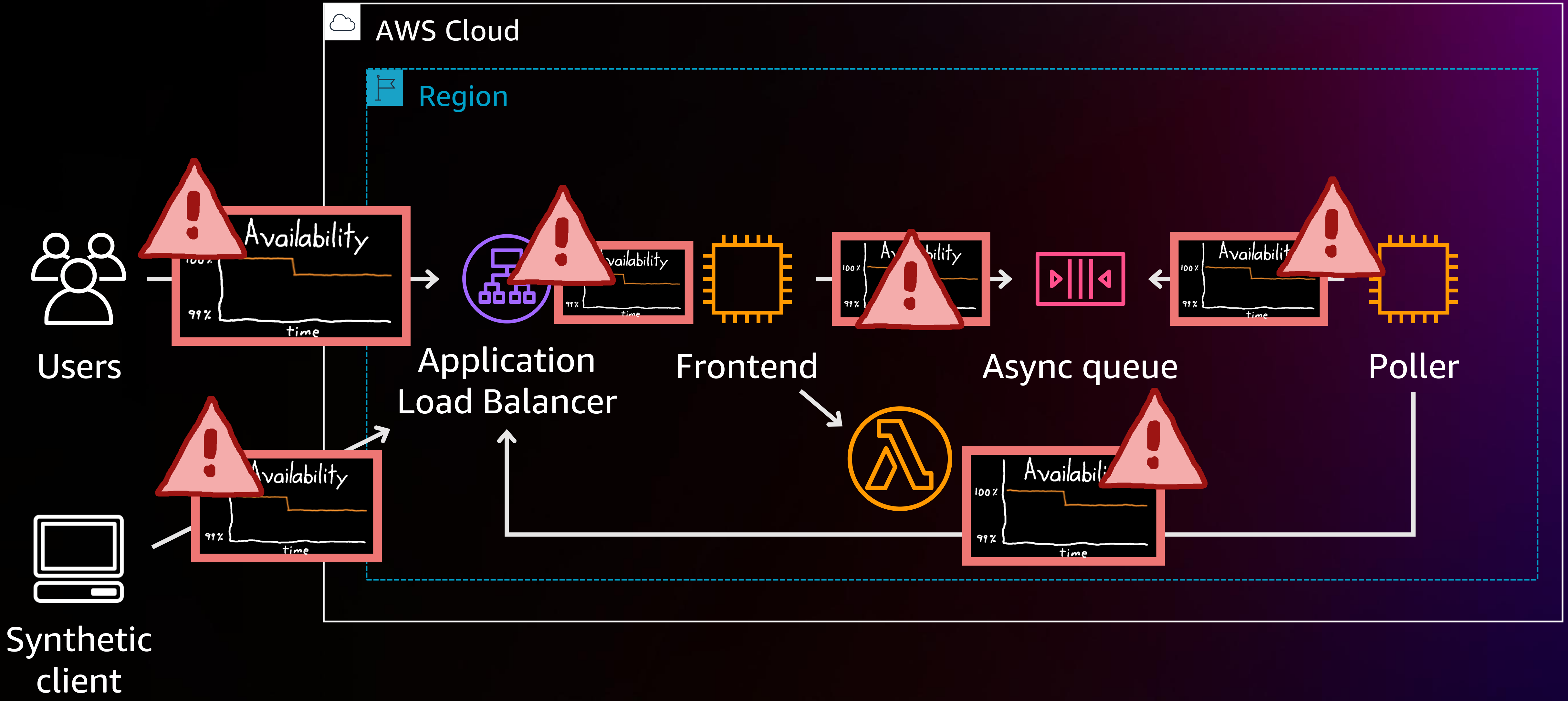
Measure from everywhere

- Be your own customer
- Instrument close to the customer
- **Alarm on everything**

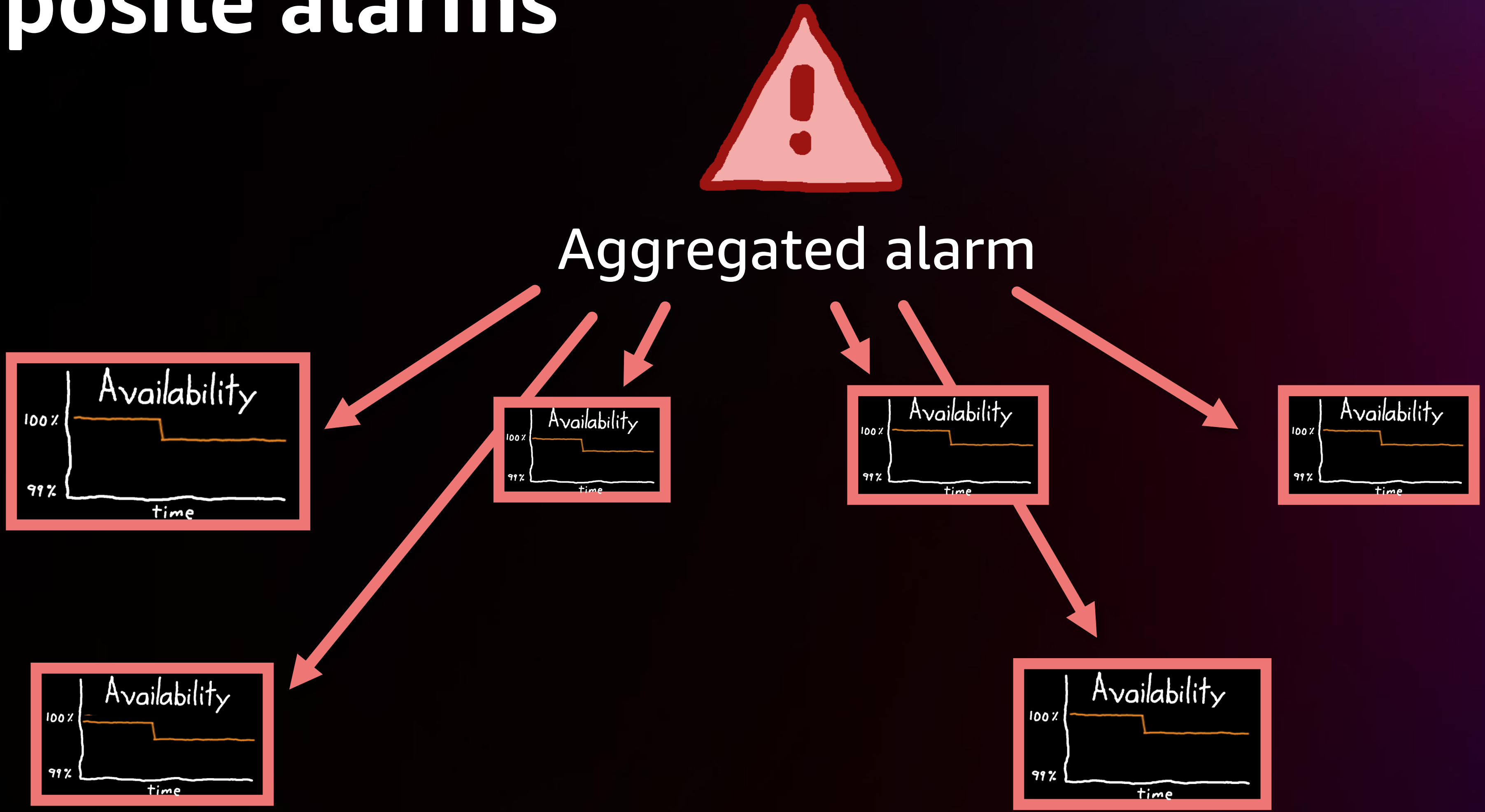
System monitoring



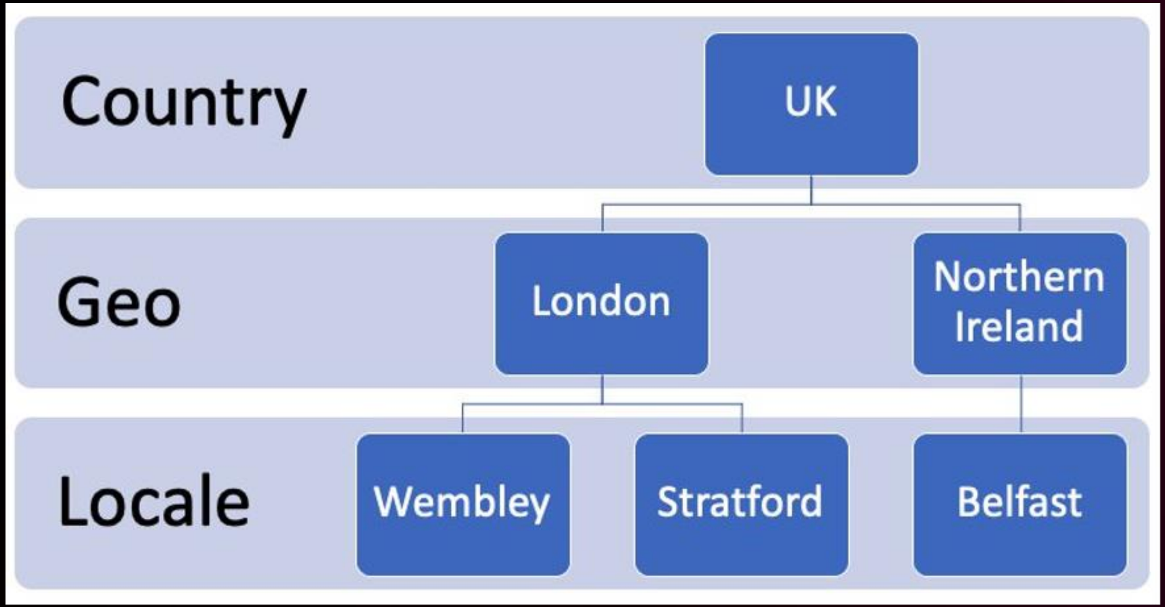
System monitoring



Composite alarms



Reducing noise at scale – BT



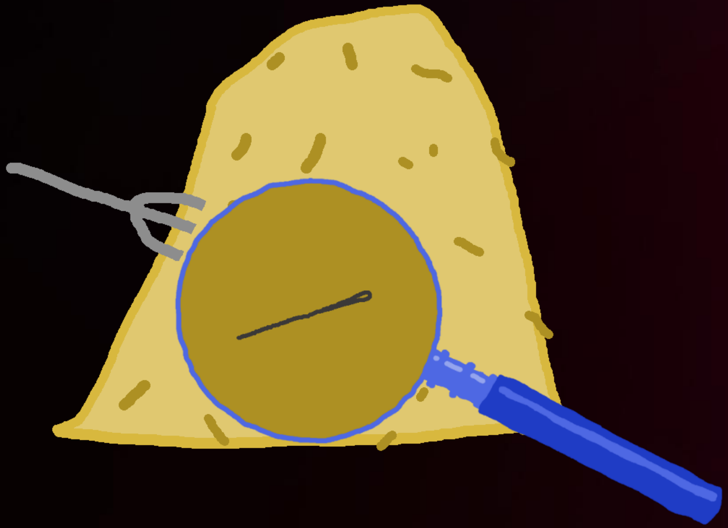
https://aws.amazon.com/blogs/mt/how-bt-uses-amazon-cloudwatch-to-monitor-millions-of-devices/

Measure from everywhere

- Be your own customer
- Instrument close to the customer
- Alarm on everything

Key takeaways

The DevOps flywheel at Amazon

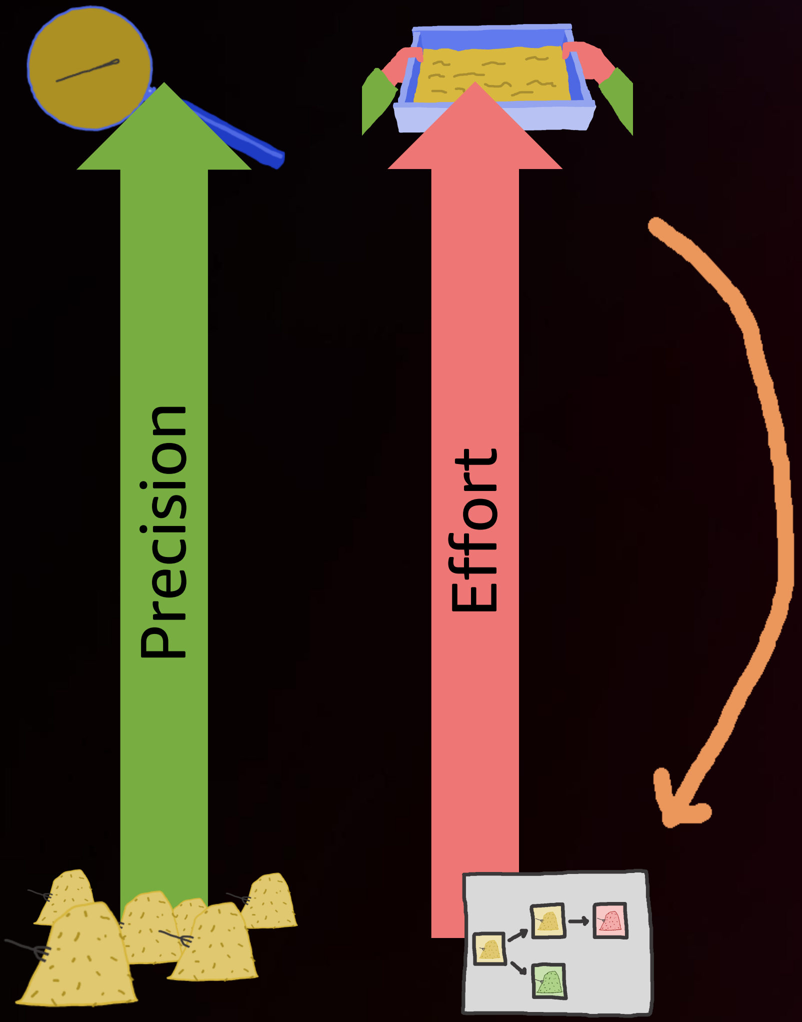


Find the root cause

Measure from everywhere

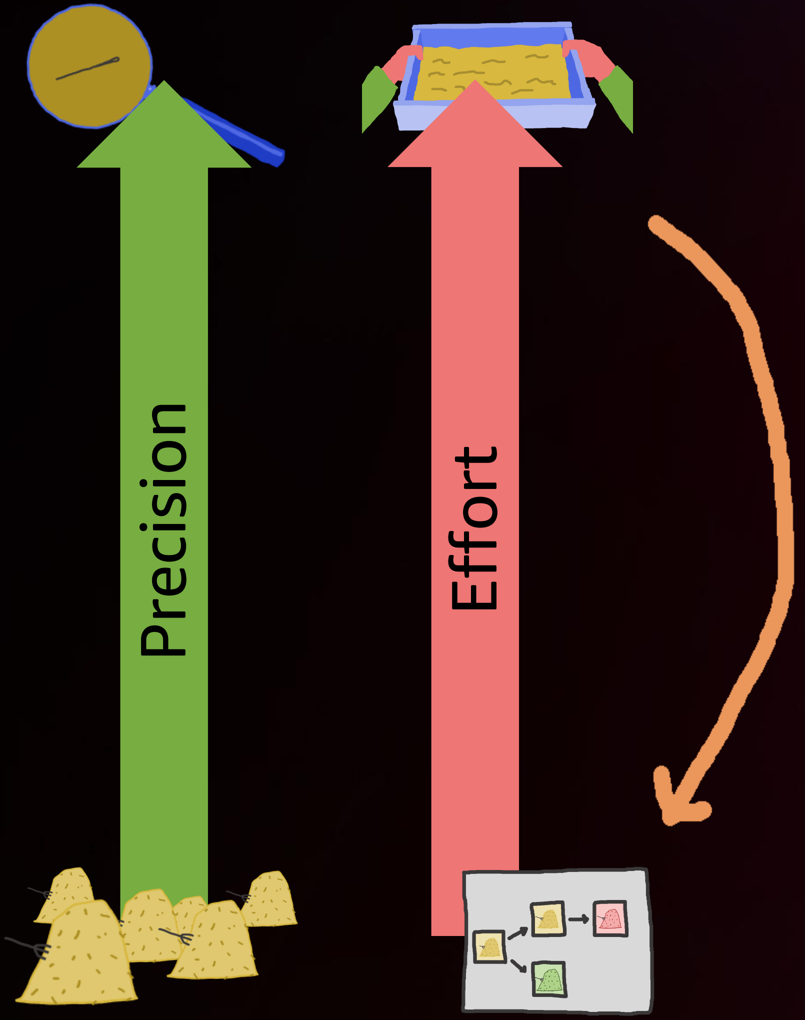
Continuously improving observability

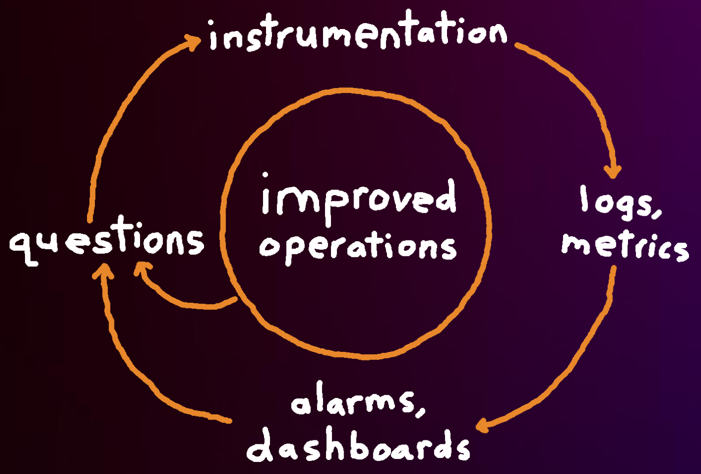
- Raw logs
- Log analysis
- Metrics
- Dashboards
- Service map
- Alarms

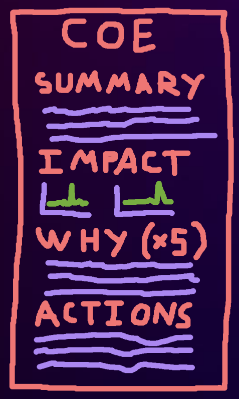


Continuously improving observability

- Raw logs
- Log analysis
- Metrics
- Dashboards
- Service map
- Alarms







Thank you!

David Yanacek
@dyanacek

Ian McGarry
in ianmcgarry

https://aws.amazon.com/builders-library/

Please complete the session survey in the **mobile app**