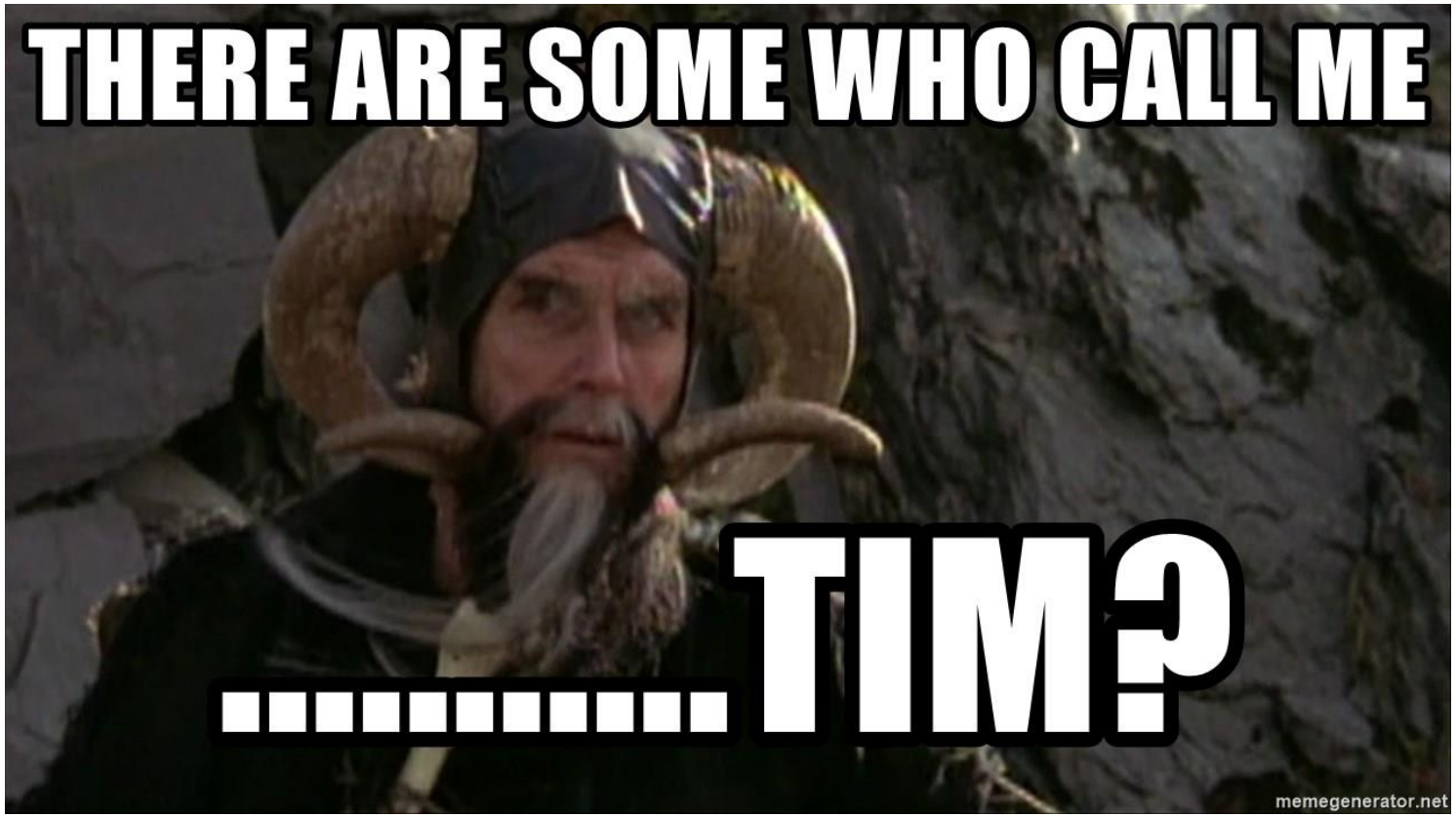
O11y vs Monitoring

What we talk about when we talk about Observability

Timothy Mahoney

Stockholm Splunk Users Group

January 19, 2023



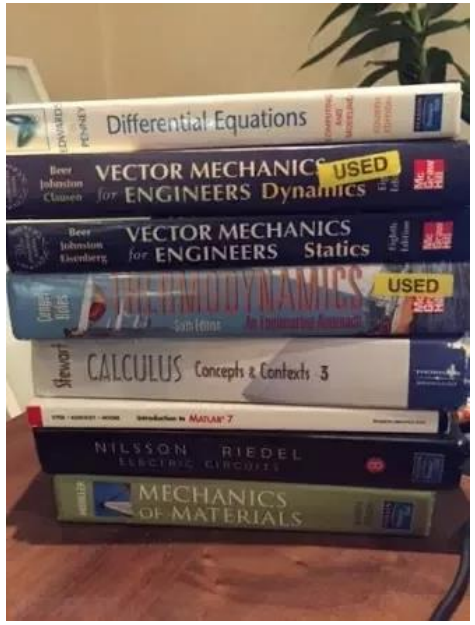
Who am I?

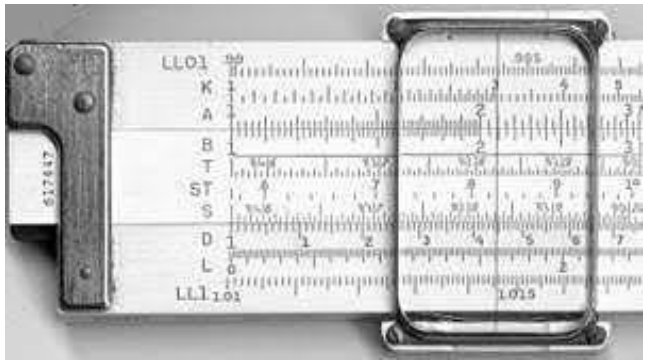
- Timothy Mahoney
- Senior Systems Engineer at Swedish furniture company you've probably never heard of..
- Splunk Derp Gun co-presenter
- Volunteer Arbiter for RIPE
- Former Satellite Engineer

What I do

- Senior Systems Engineer in the Observability Pipeline Team
- Observability Framework
- OpenTelemetry and OpenTelemetry Collector
- Documentation, Examples, Demos, Labs, Tests
- INGKA Native Clouders program training lab on O11y and Distributed Tracing

What are we even talking about?





Control Theory

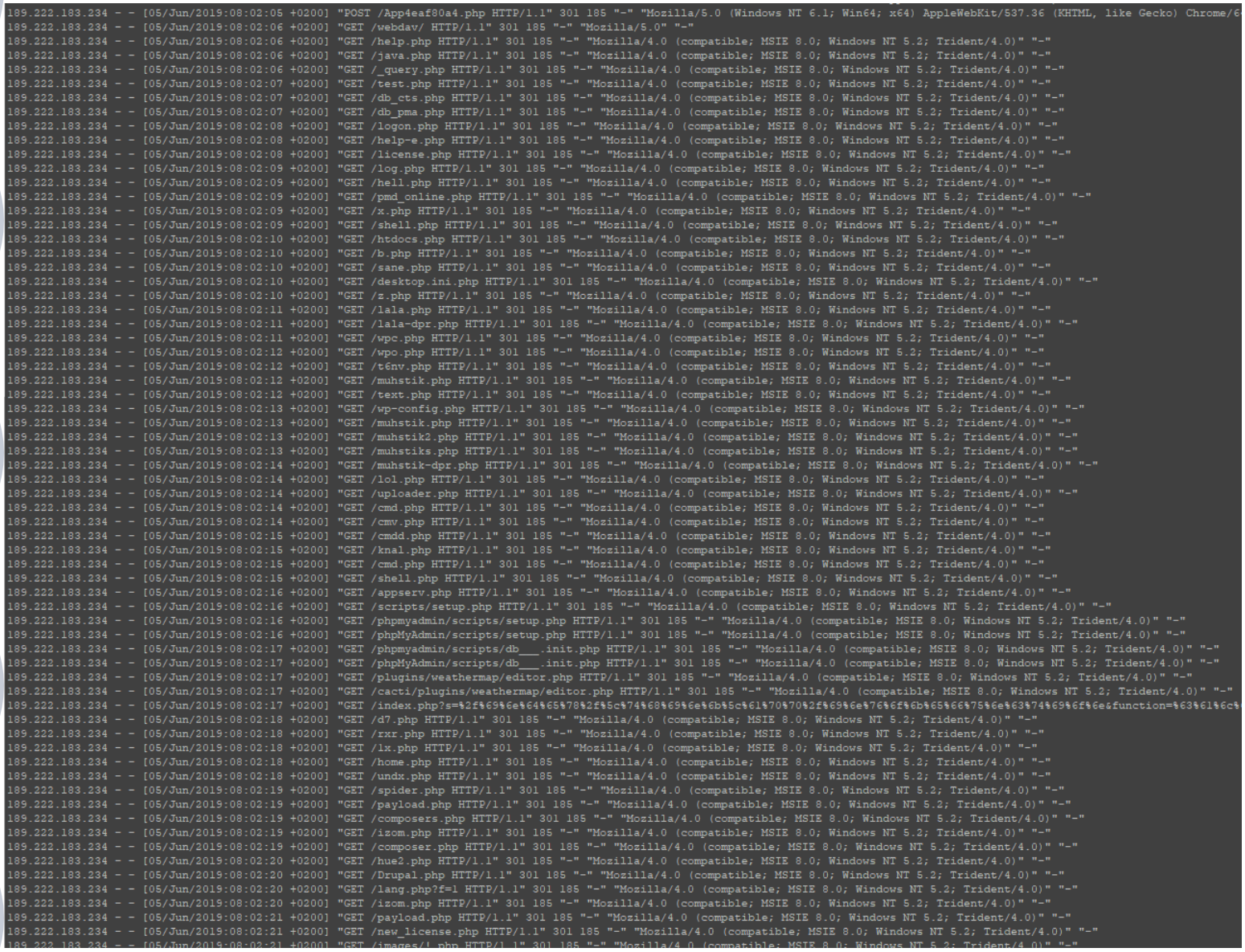
- Control Theory revolves around systems
- Observability in the control theory context is a generalization
- Observability is gaining internal insight into a system from its external signals

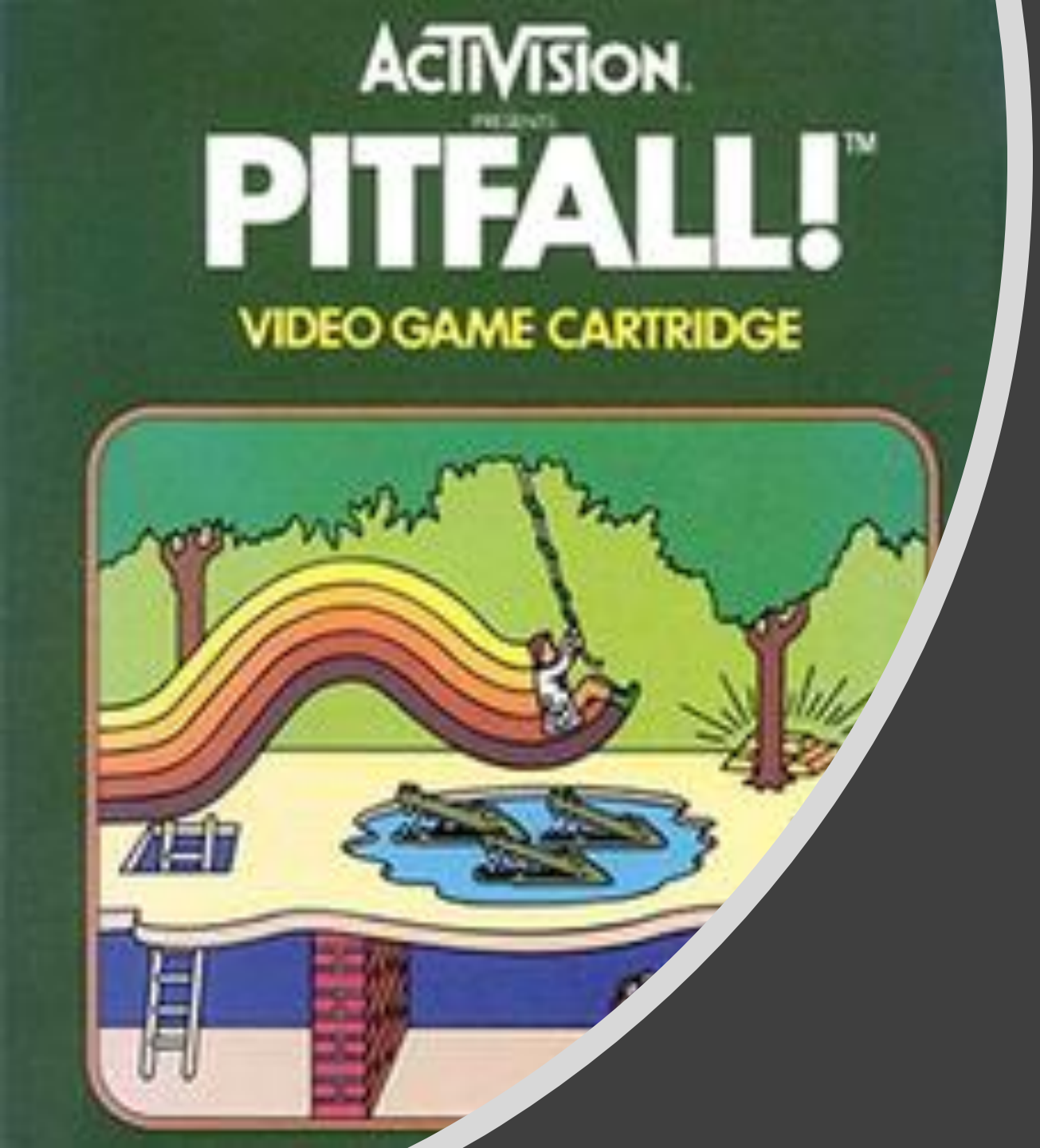
O11y in the IT systems context

- IT Operational Observability Signals
- The overused "Three Pillars of Observability"
- Different types of systems use different types of observability.

It all begins with logs..

- "Check the logs"
- Format is often dictated by 3rd party vendors, haphazardly structured or both.
- Cleanup with props, transforms, schema on the fly
- Responsibility to turn log events into useful data lies with the tooling.





The pitfalls of printf()

- Devs use printf() to write logs
- Common practice
- Syslog or stdout
- Format? Context? Correlation?

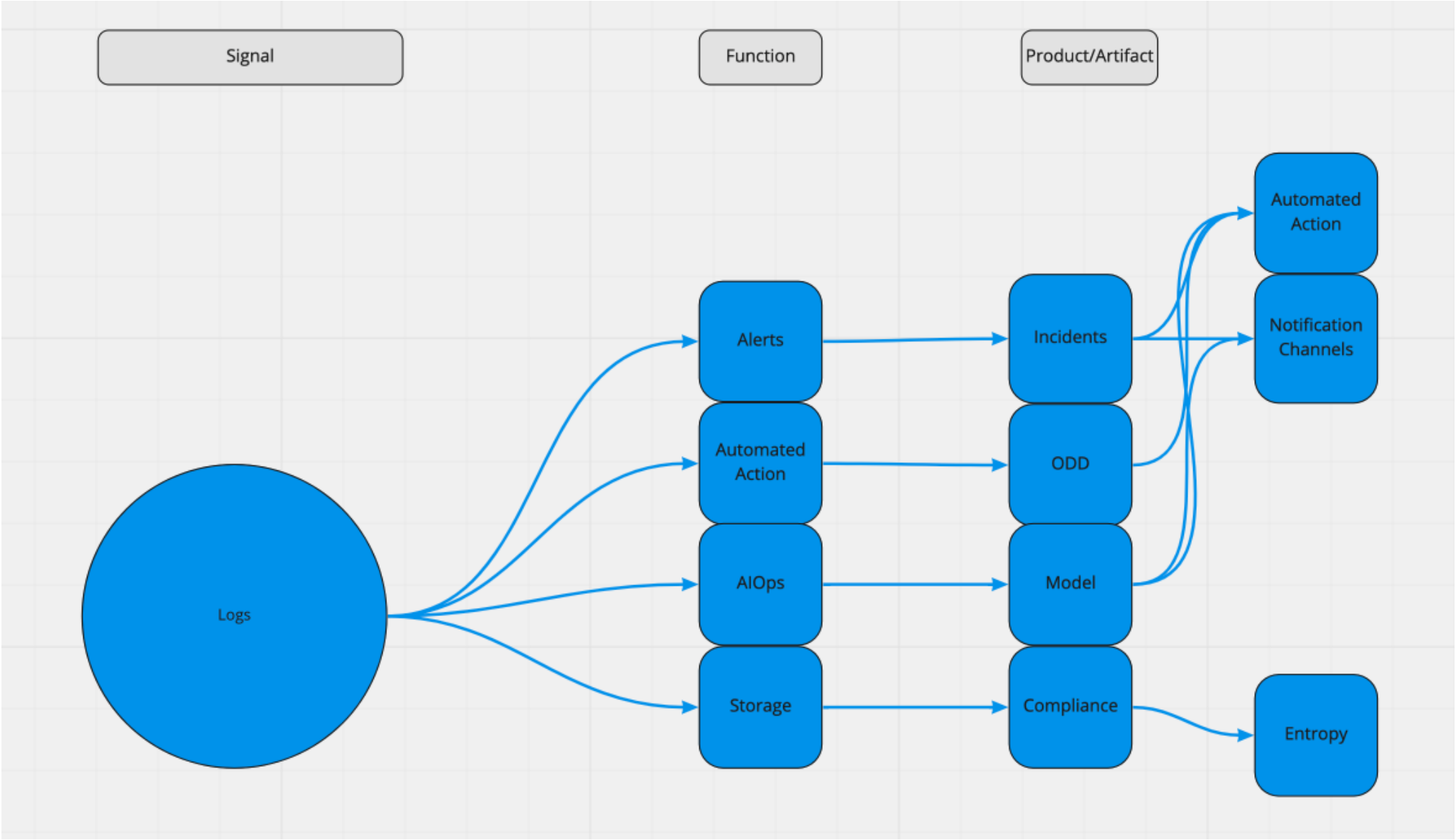
Observability in the age of Microservices

- Dev/Ops
- Stateful and Stateless Workloads
- Containerized workloads
- CI/CD
- Microservices trade the simplicity of the application for the complexity of the system.

Making logs great again:

- Structured Events
- Arbitrarily Wide Events
- JSON or ProtoBuf
- Event driven architecture, application does its thing and emits an event.
- It as though devs read our minds! Structure your data!

```
{
  "Timestamp": 1586960586000, // JSON needs to make a decision about
                              // how to represent nanoseconds.
  "Attributes": {
    "http.status_code": 500,
    "http.url": "http://example.com",
    "my.custom.application.tag": "hello",
  },
  "Resource": {
    "service.name": "donut_shop",
    "service.version": "semver:2.0.0",
    "k8s.pod.uid": "1138528c-c36e-11e9-a1a7-42010a800198",
  },
  "TraceId": "f4dbb3edd765f620", // this is a byte sequence
                                 // (hex-encoded in JSON)
  "SpanId": "43222c2d51a7abe3",
  "SeverityText": "INFO",
  "SeverityNumber": 9,
  "Body": "20200415T072306-0700 INFO I like donuts"
}
```


Signal
Function
Product/Artifact
Logs
Alerts
Automated Action
AIOps
Storage
Incidents
ODD
Model
Compliance
Automated Action
Notification Channels
Entropy

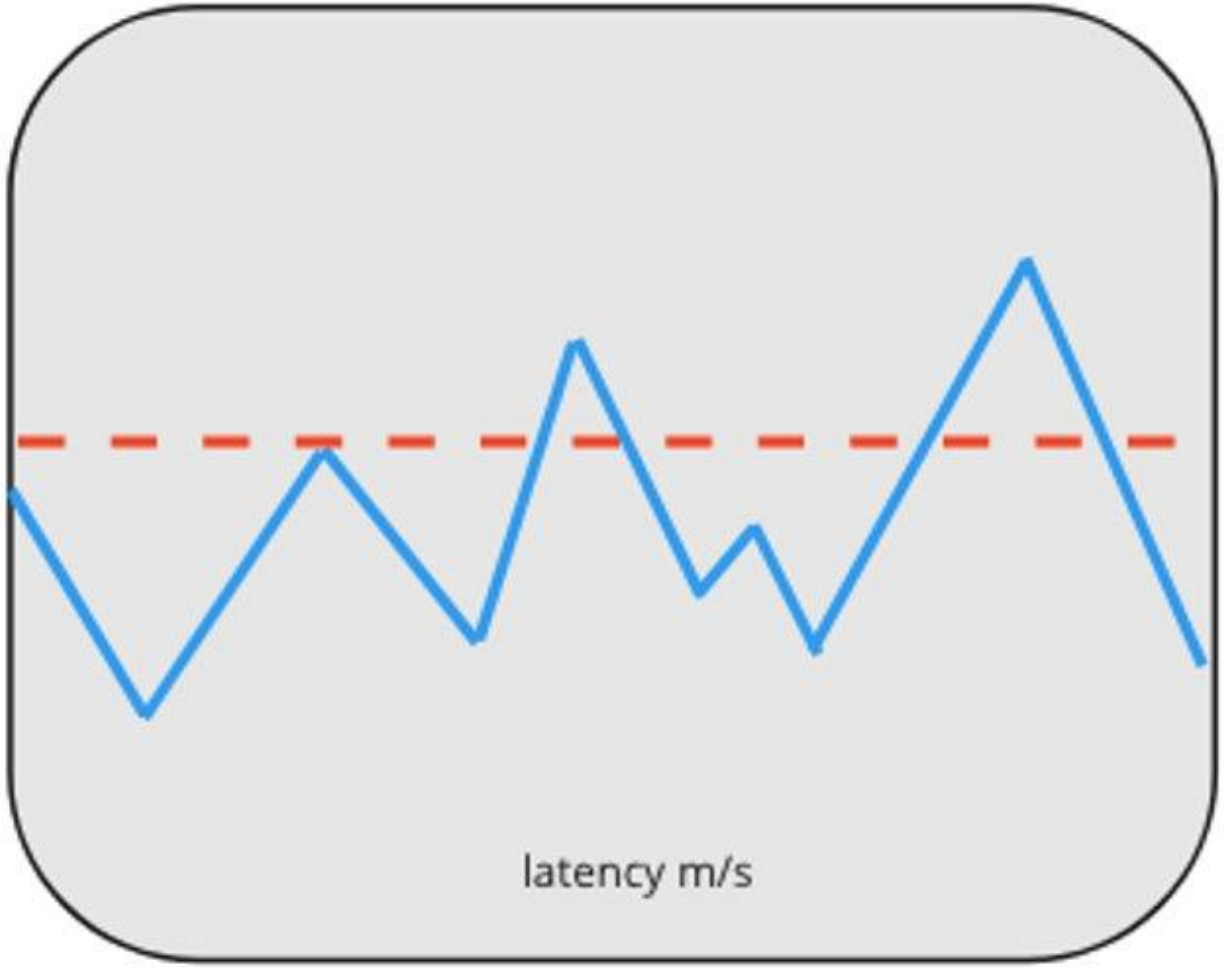
Metrics

Where we started:

- Proprietary metrics in application monitor and control suites.
- Metrics field extracted from log data.

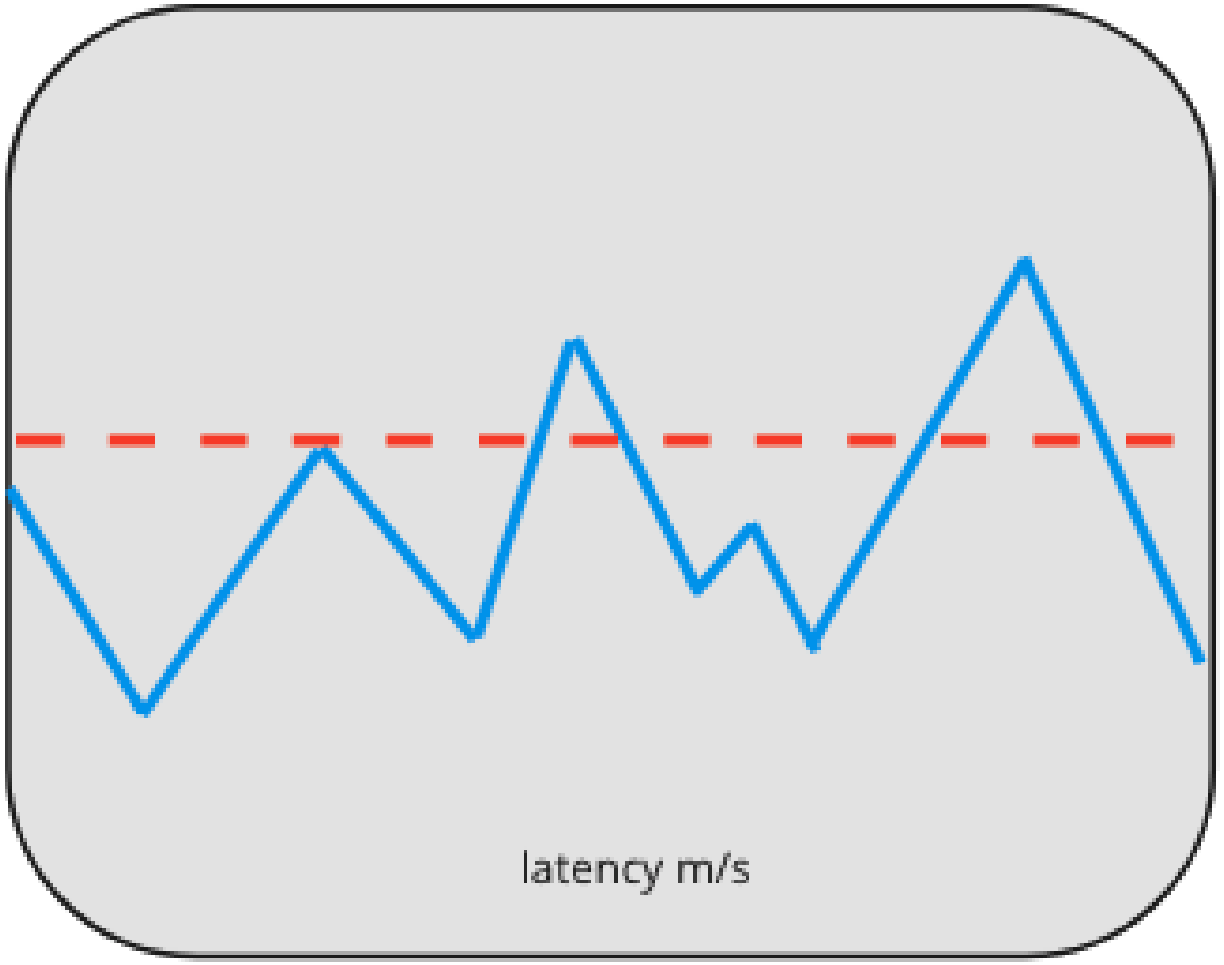
Where we are going:

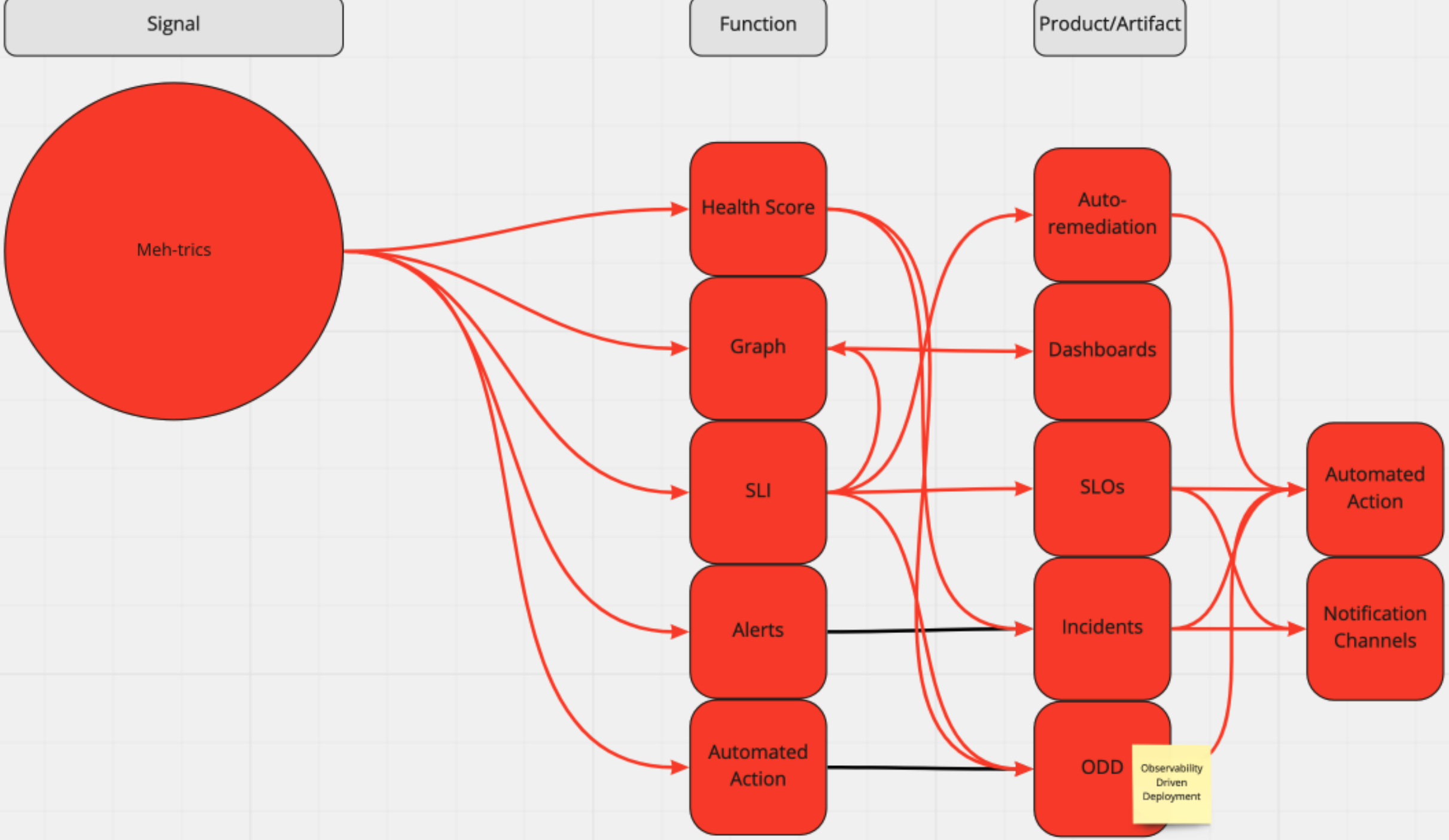
- Metrics as a telemetry data type



Metrics should be…

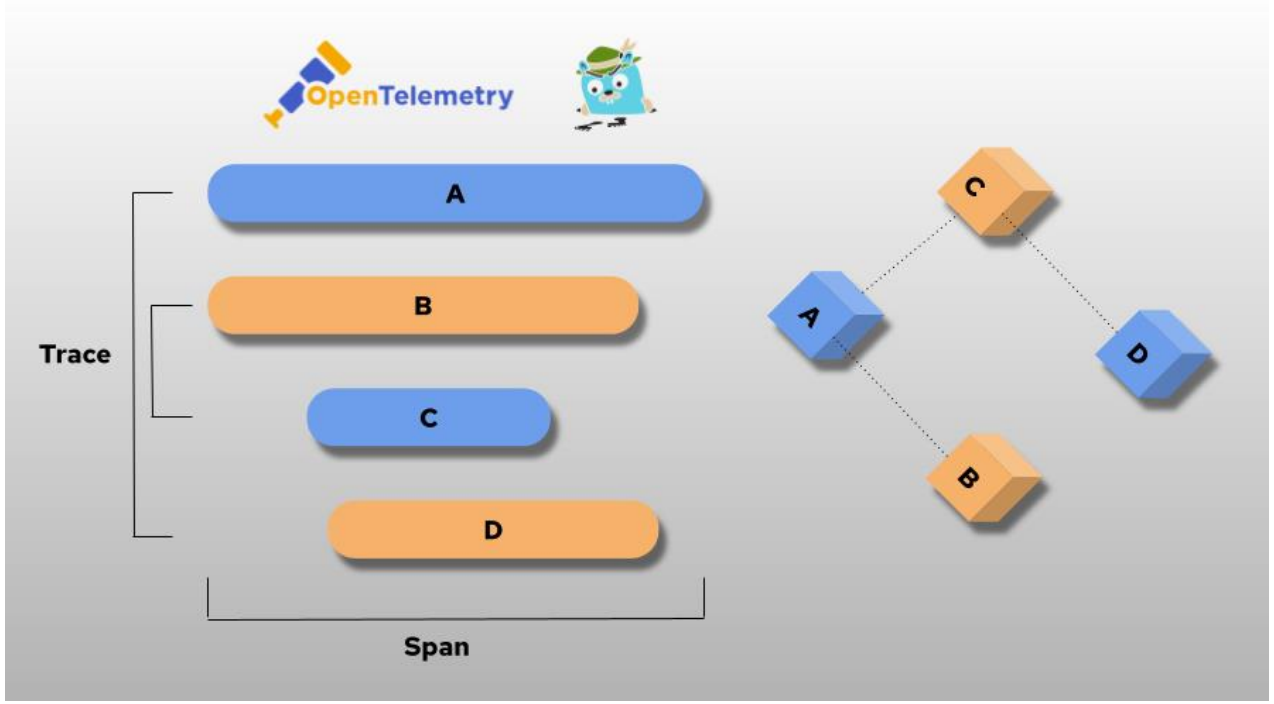
- LOW cardinality data
- Gauge
- Delta
- Histogram
- Can have Labels
- Can have exemplars
- Adding cardinality to metrics data increases the size and cost of the timeseries db that stores metrics.




Signal
Function
Product/Artifact
Meh-trics
Health Score
Graph
SLI
Alerts
Automated Action
Auto-remediation
Dashboards
SLOs
Incidents
ODD
Observability Driven Deployment
Automated Action
Notification Channels

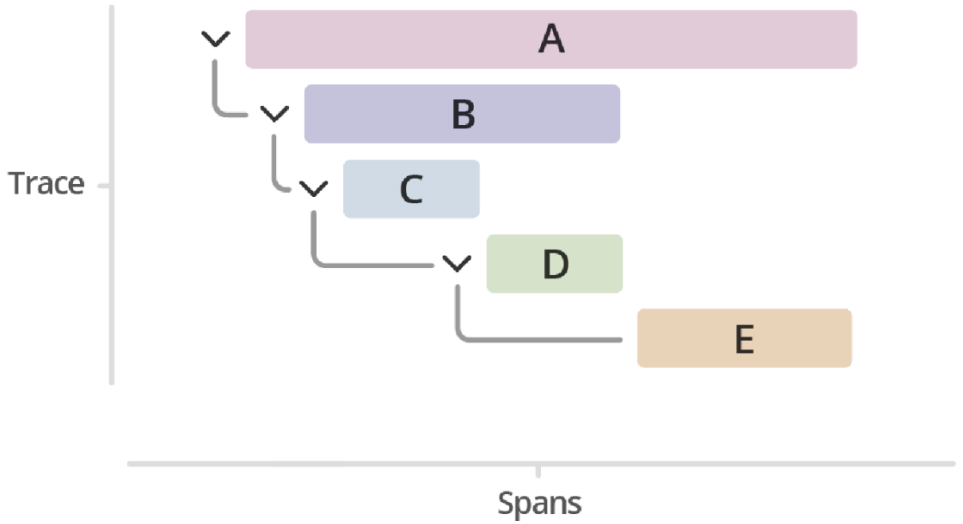
Traces

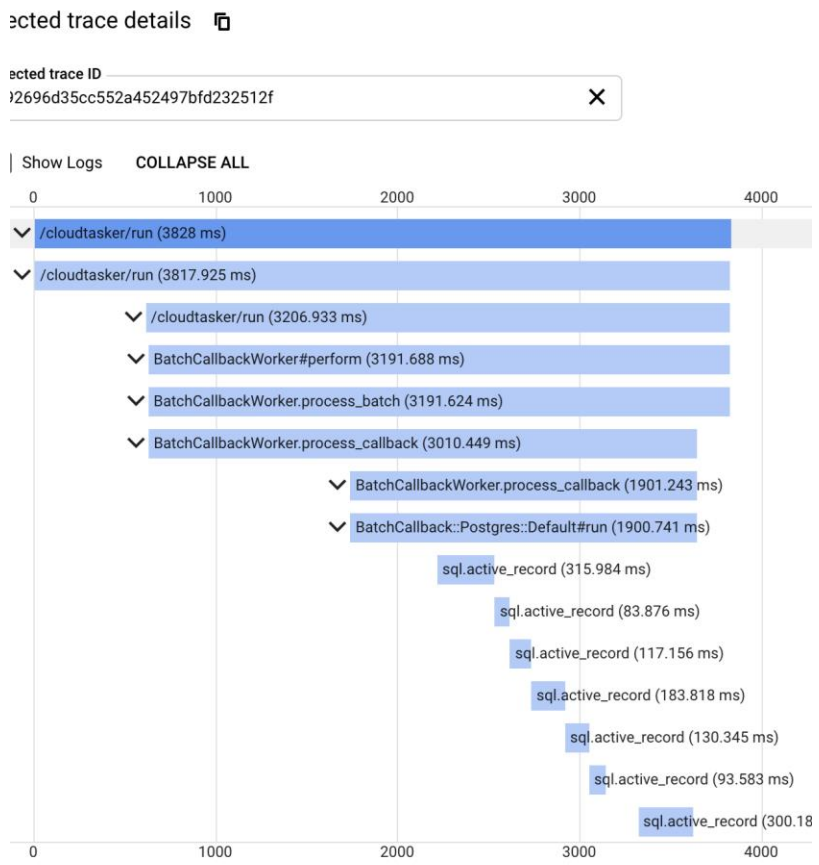
- With distributed applications and microservices, how can we find where our application is spending the most time and where it is failing?



How do traces work?

- Traces inherit a Trace ID from the Root Span (A)
- Traces propagate the Trace ID on to all child spans(B,C,D,E)
- If a request is received without any trace information, a new Trace ID is created.
- The trace propagation is not limited by system.





What do traces look like?

- Important components:
- Span Context – serialized and propagated, immutable
- Attributes – key-value pair metadata
- Events – structured log message
- Span Links – Link multiple or async spans
- Span Status – OK, Unset, Error
- Span Kind – server, client, internal, producer, consumer


```
{
    "name": "Hello-Greetings",
    "context": {
        "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
        "span_id": "0x5fb397be34d26b51",
    },
    "parent_id": "0x051581bf3cb55c13",
    "start_time": "2022-04-29T18:52:58.114304Z",
    "end_time": "2022-04-29T18:52:58.114435Z",
    "attributes": {
        "http.route": "some_route1"
    },
    "events": [
        {
            "name": "hey there!",
            "timestamp": "2022-04-29T18:52:58.114561Z",
            "attributes": {
                "event_attributes": 1
            }
        },
        {
            "name": "bye now!",
            "timestamp": "2022-04-29T22:52:58.114561Z",
            "attributes": {
                "event_attributes": 1
            }
        }
    ],
}
```

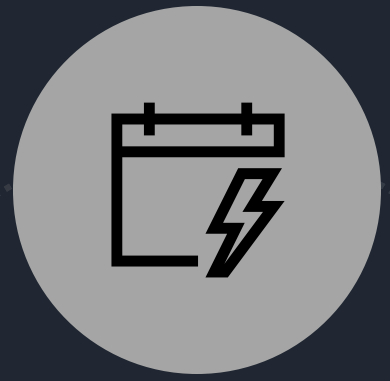
What is a span?

- **Span Metadata:**
- Service
- Operation
- Span ID
- Parent ID
- Time (Start – End)
- Duration
- Relative Start


```
{
  "trace_id": "7bba9f33312b3dbb8b2c2c62bb7abe2d",
  "parent_id": "",
  "span_id": "086e83747d0e381e",
  "name": "/v1/sys/health",
  "start_time": "2021-10-22 16:04:01.209458162 +0000 UTC",
  "end_time": "2021-10-22 16:04:01.209514132 +0000 UTC",
  "status_code": "STATUS_CODE_OK",
  "status_message": "",
  "attributes": {
    "net.transport": "IP.TCP",
    "net.peer.ip": "172.17.0.1",
    "net.peer.port": "51820",
    "net.host.ip": "10.177.2.152",
    "net.host.port": "26040",
    "http.method": "GET",
    "http.target": "/v1/sys/health",
    "http.server_name": "mortar-gateway",
    "http.route": "/v1/sys/health",
    "http.user_agent": "Consul Health Check",
    "http.scheme": "http",
    "http.host": "10.177.2.152:26040",
    "http.flavor": "1.1"
  },
  "events": [
    {
      "name": "",
      "message": "OK",
      "timestamp": "2021-10-22 16:04:01.209512872 +0000 UTC"
    }
  ]
}
```

"We're just going to turn the logs off..."

YOU'RE GOING TO WHAT?



SPAN EVENTS

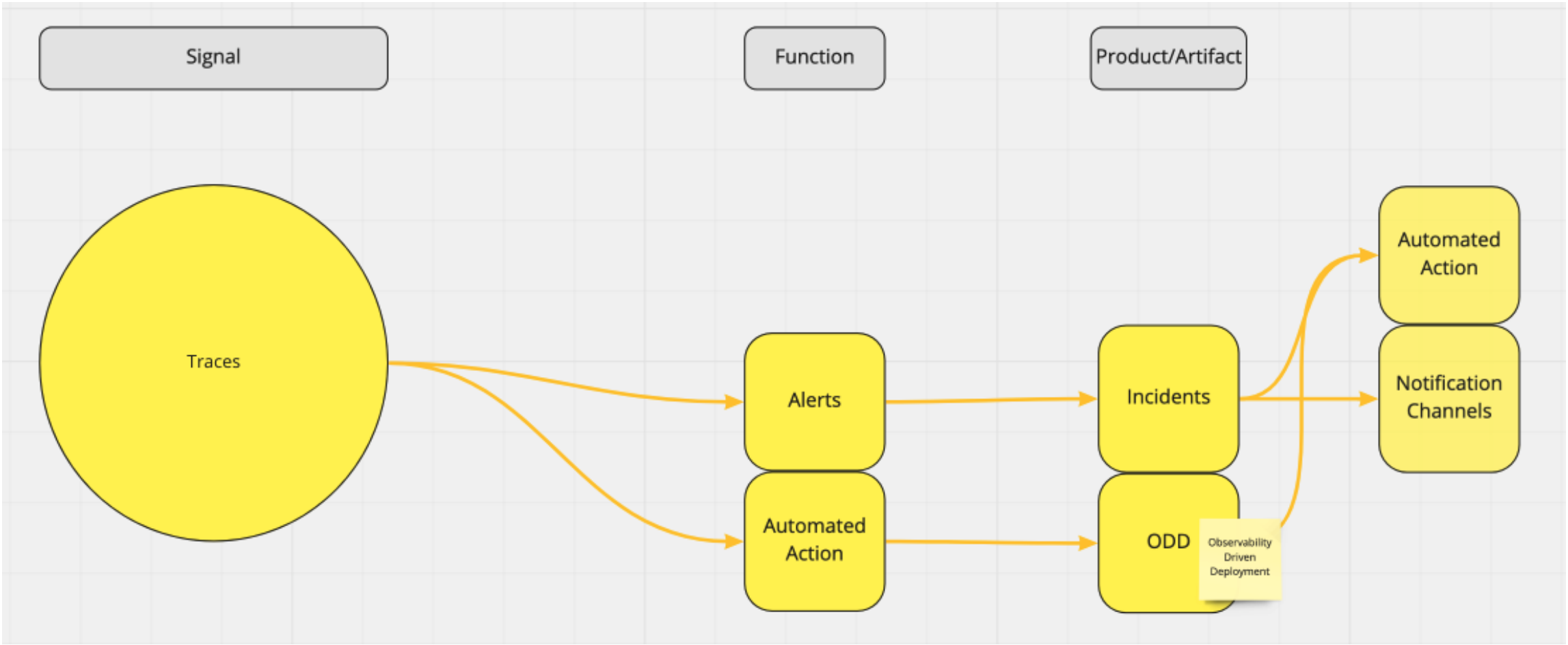
Trace propagation, message queues and databases.

Context propagation in PubSub messages

Context propagation in SQL message fields

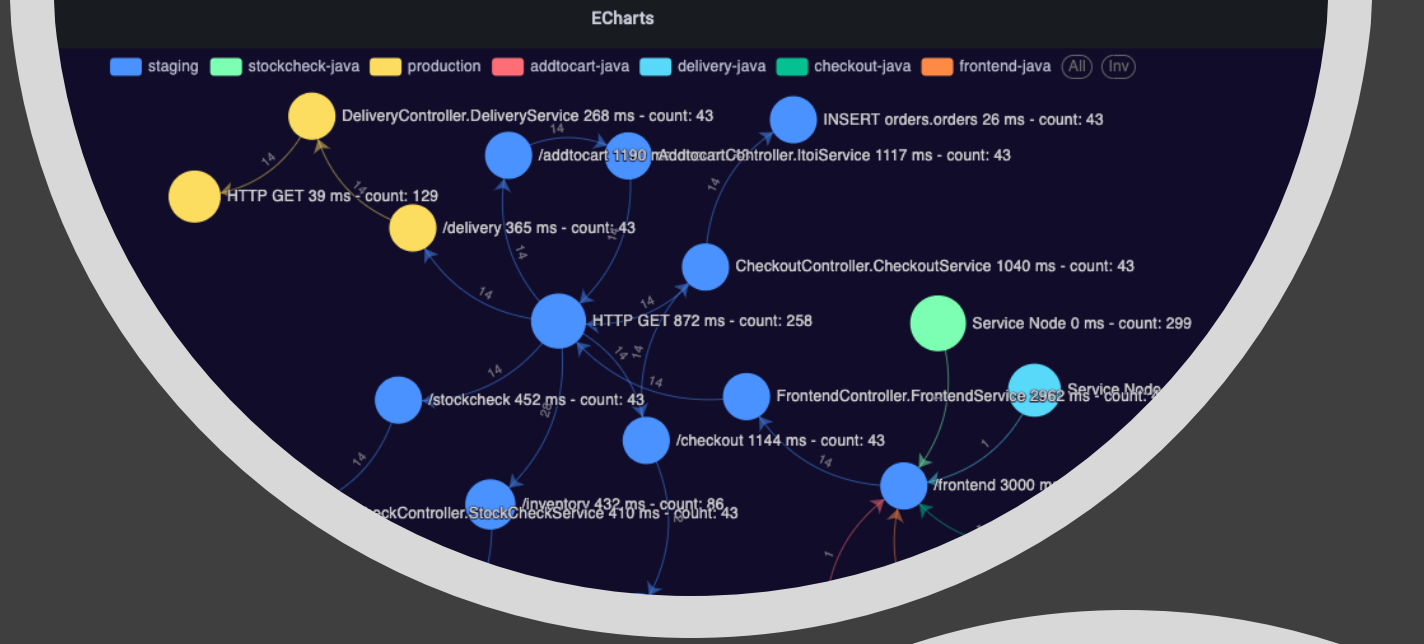
Tracing for CI/CD Pipelines

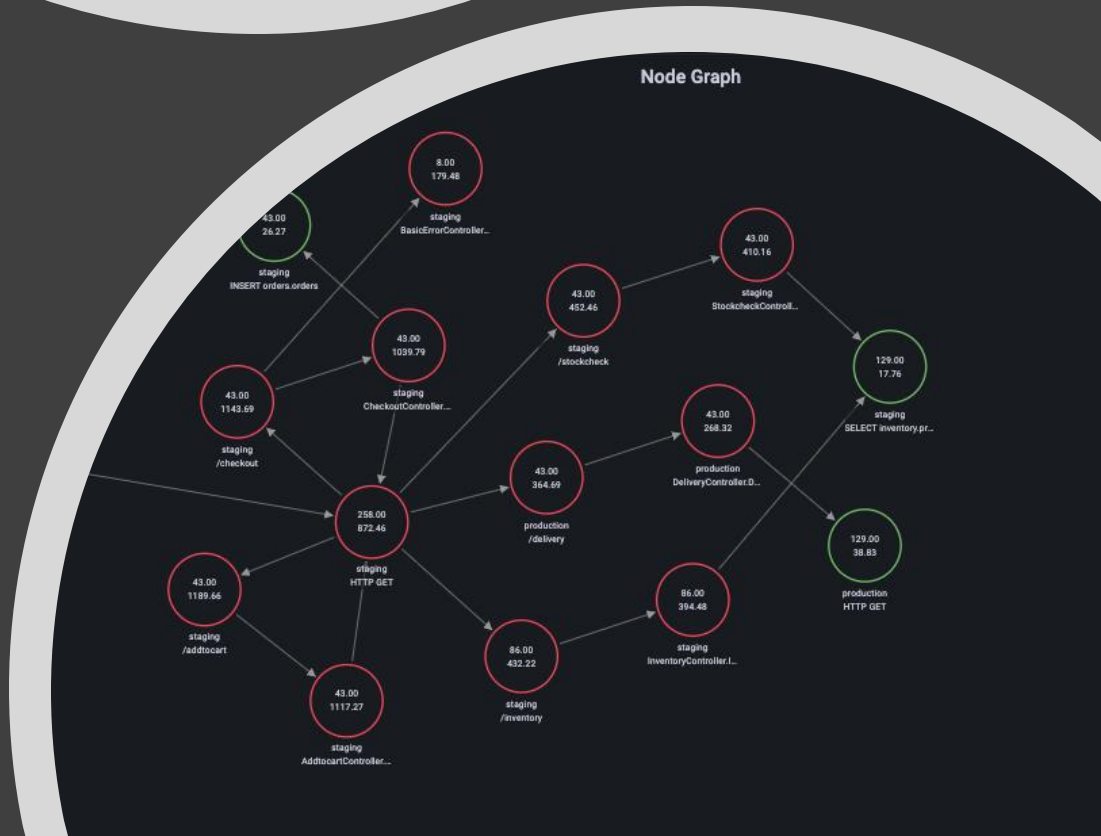
- Trace a build process
- Why does a build take so long?
- What is delaying our deployment?


Signal
Function
Product/Artifact
Traces
Alerts
Automated Action
Incidents
ODD
Observability Driven Deployment
Automated Action
Notification Channels

Using traces to map services

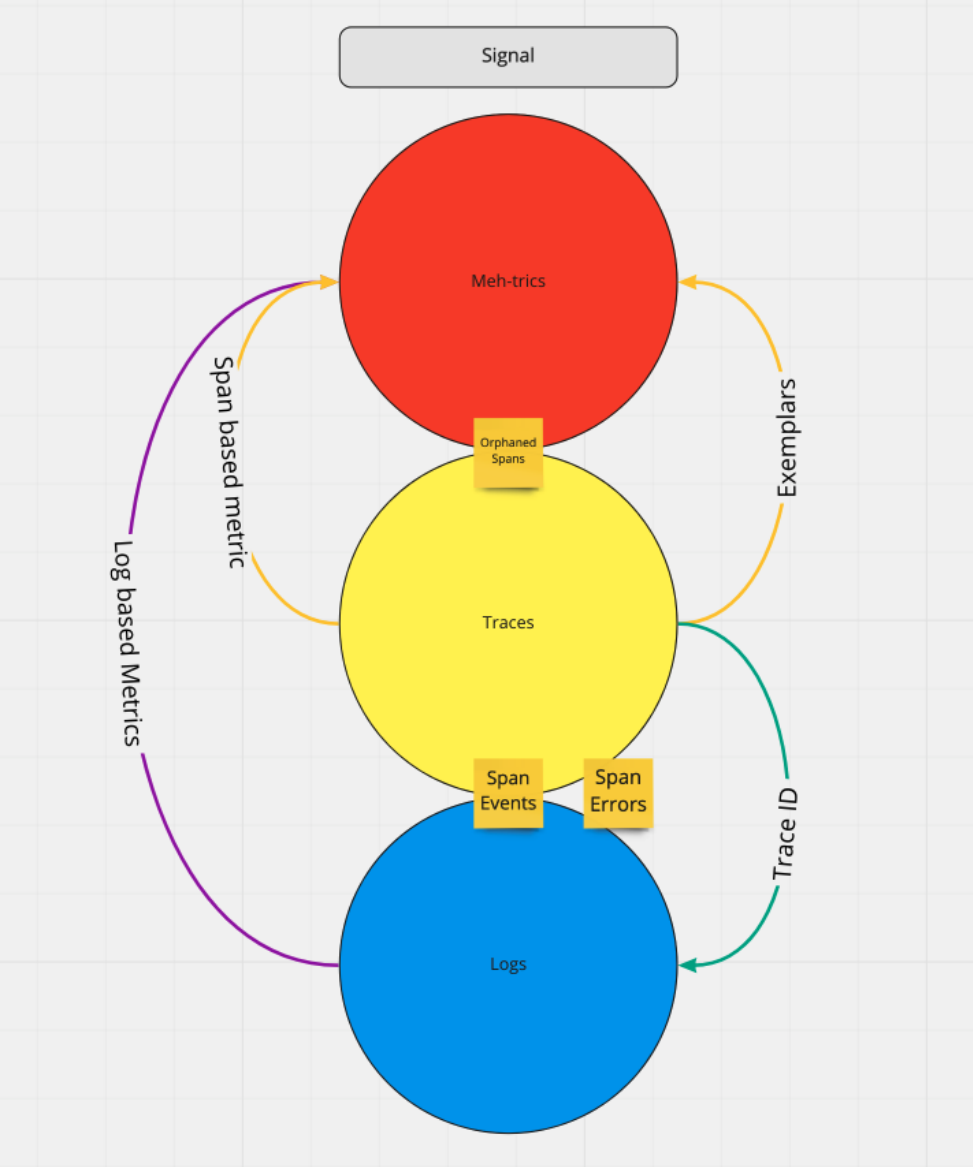
- Trace baggage can include key value pairs to map to services, shopping carts, users, to spans.
- Trace data can be used to map a system "as-built"
- Many teams have no clue who their consumers are





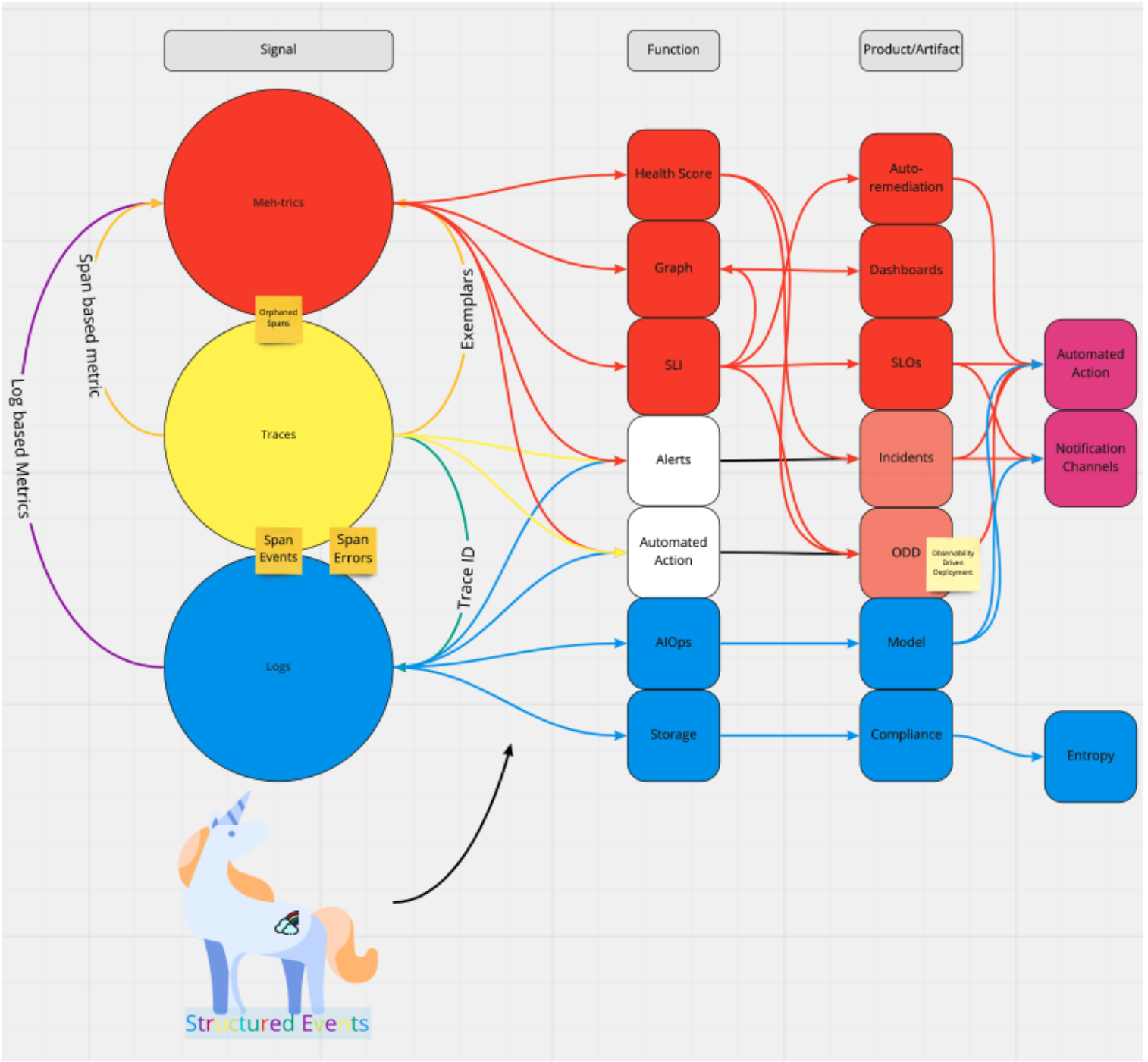
Correlating Signals

- Log based Metrics
- TraceID in Logs
- Span Events
- Orphaned Spans
- Exemplars



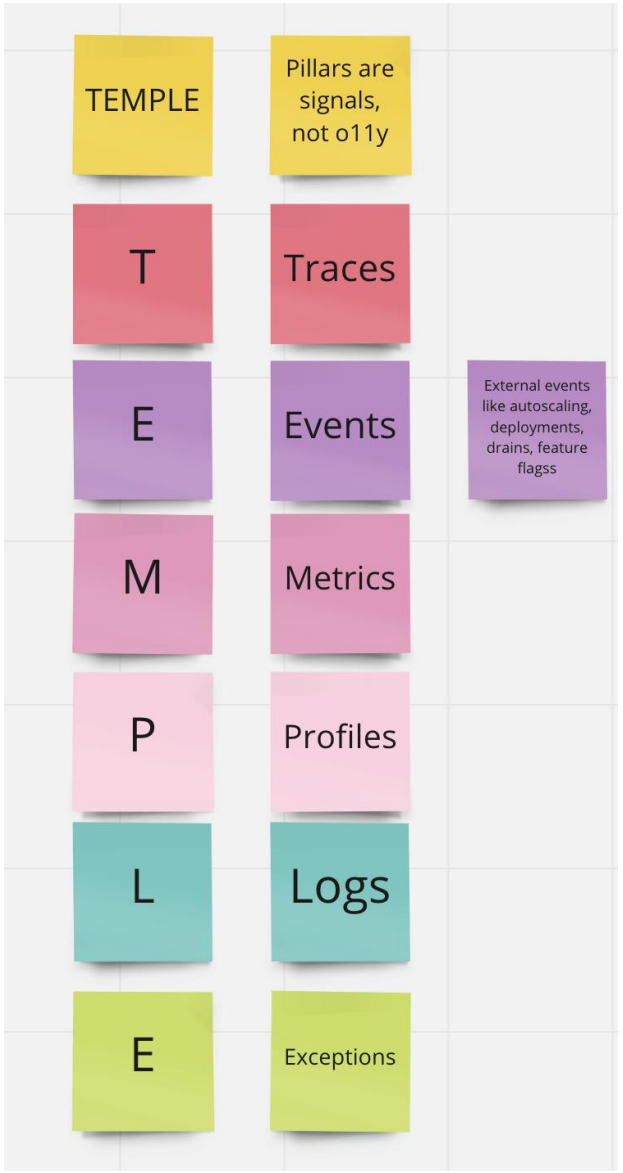
Putting it all together.

- The functions and products of observability often rely heavily on metrics
- Metrics, traces and logs are the most common signals but Observability in this context is not limited to them.



TEMPLE

- Continuous Profiling
- External Events
- Exceptions
- eBPF



ttps://medium.com/@YuriShkuro/temple-six-pillars-of-observability-4ac3e3deb402

OpenTelemetry

- This changes everything
- Open Source CNCF Project
- Standard for Telemetry Data
- Widely adopted
- OTLP Transport Specification

Instrumentation Libraries

- Libraries and SDKs available from opentelemetry.io
- Kubernetes can auto-inject instrumentation libraries for Java, Python and Node.JS applications.
- Automagic Instrumentation

Project Maturity

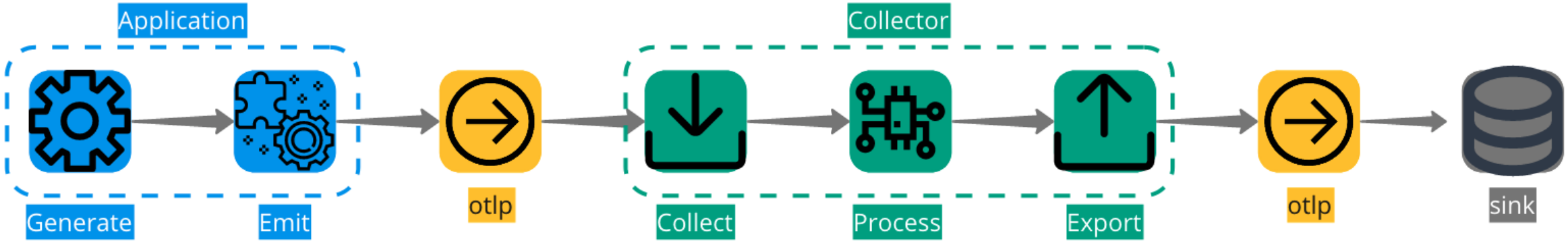
- Current status of metric, log and tracing support in the OpenTelemetry collector and in a number of languages.
- *I cannot guarantee this is current.. It's close, though.

	Traces	Metrics	Logs
Collector	GA	Beta (User API RC)	Beta
Java	GA	RC	
JS	GA	RC	
.Net	GA	GA	
Python	GA	RC	
Go	GA	In Progress	
Ruby	In Progress	In Progress	
C++	GA	GA	
PHP	In Progress	In Progress	

OpenTelemetry Collector

- Provided by OpenTelemetry Project
- Contrib with Vendor Specific Exporters
- Collect, Process and Export Telemetry Data
- Observability Pipeline Team offers a vendor specific contrib

Role of the OpenTelemetry Collector



Sinks

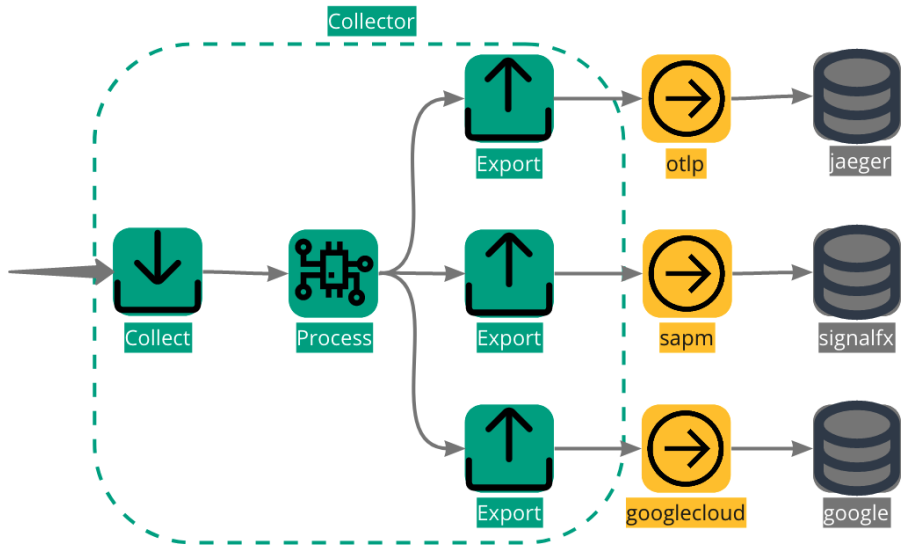
- Where your application sends its telemetry signals.

Examples:

- GCP Monitoring
- Splunk
- Loki
- Tempo
- Jaeger
- Zipkin

Fan Out

- Instrument once and send to one or multiple sinks
- Change only the collector configuration to send to a new sink
- Multiple supported exporters using the collector contrib.




```
pipelines:

  traces:
    receivers: [otlp]
    processors: [memory_limiter, resource, batch]
    exporters: [logging, googlecloud, otlphttp, sapm]
```

OpenTelemetry Collector Export Protocols

- Native OTLP
- Jaeger
- Zipkin
- Splunk
- Pub/Sub

The OpenTelemetry Collector is not a protocol translator

- The general idea is that your telemetry is transported via OTLP to a tool/sink and not translated from one contrib protocol to another.

Cloud Agnostic

- **Instrument once, consume anywhere.**
- Use the same instrumentation for your application in different environments.
- Run the same code in Azure, GCP, AliCloud with the same instrumentation and only change the collector config to suit your needs.

How do we differentiate between Monitoring and Observability?

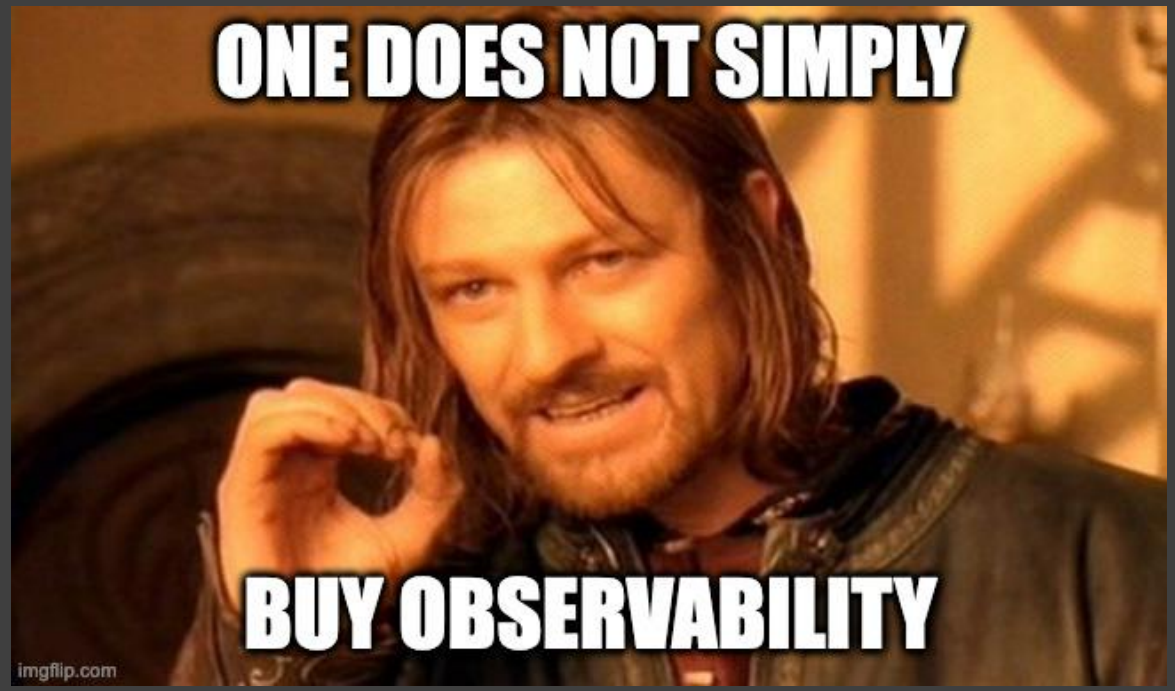
I will admit to listening to way too much O11ycast.

Table 9-1. Factors that vary between systems and software

Factor	Your systems	Your software
Rate of change	Package updates (monthly)	Repo commits (daily)
Predictability	High (stable)	Low (many new features)
Value to your business	Low (cost center)	High (revenue generator)
Number of users	Few (internal teams)	Many (your customers)
Core concern	Is the system or service healthy?	Can each request acquire the resources it needs for end-to-end execution in a timely and reliable manner?
Evaluation perspective	The system	Your customers
Evaluation criteria	Low-level kernel and hardware device drivers	Variables and API endpoint
Functional responsibility	Infrastructure operations	Software development
Method for understanding	Monitoring	Observability

Observability is not the tooling.

- Splunk, Google Monitoring, Grafana, and Jaeger are all tools that interpret, process and act upon observability data. Just by sending data to them, you do not have observability.

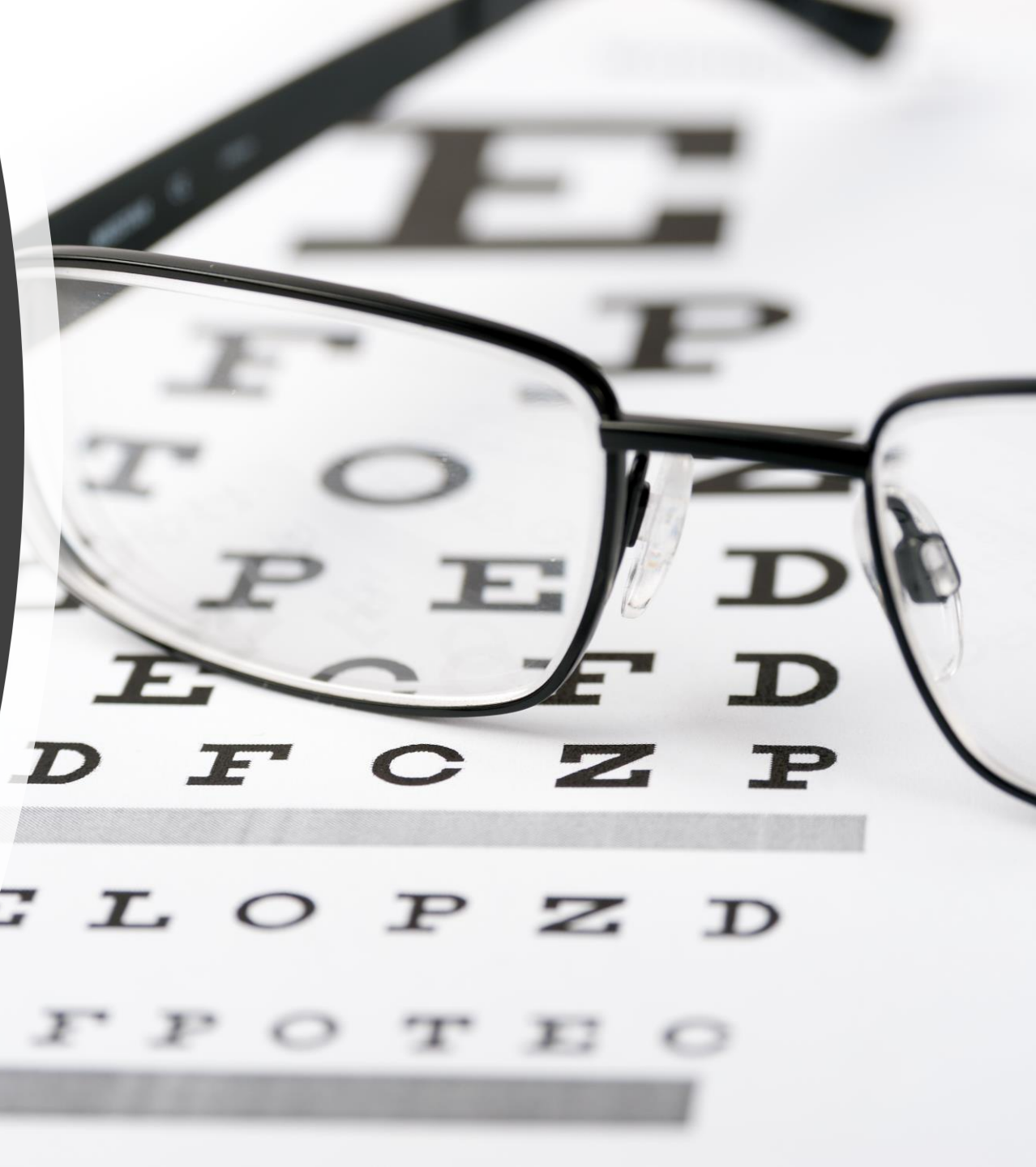


Observability costs money

- It's should be part of any service budget.
- It takes time to instrument code.
- It should be a NFR of any project.
- You can easily defend the cost of your observability data if you are actively using it and getting value from it.

Observability gives value

- How are you even sure your application is working as expected if you can't observe it?
- If you put the effort into instrumenting your code, you will be rewarded with a deeper understanding of how it is working.



Observability costs money

- It's should be part of any service budget.
- It takes time to instrument code.
- It should be a NFR of any project.
- You can easily defend the cost of your observability data if you are actively using it and getting value from it.

Common Challenges in Observability

- Write once, read never database
- Those "I need ALL the data" people
- Dashboards as technical debt
- "You build it, you run it" team decides to run own observability stack
- "What do you mean this isn't a debugging tool?"
- Trying to solve data or business observability issues using only application telemetry.

Site Reliability Engineering

Created by Google

SRE book is freely available

Service Level Agreements
Service Level Objectives
Service Level Indicators

Where do we even begin?

- What metrics can we use to describe the critical aspects of our service?
- Where is the best place to measure them?
- Who is using this service?
- What are our Service Level Indicators?

Metrics: A refresher course in one slide

Delta, Cumulative, Gauge

LETS

- Latency
- Errors
- Traffic
- Saturation

SREs Four Golden Signals

USE

- Utilization
- Saturation
- Errors

System-centric

RED

- Rate
- Error
- Duration

Workload-centric

So how is looking at metrics for SLOs different than just monitoring?

We don't want to analyze metrics for every aspect of our service.

We don't need to create a bunch of special rules and tests based on one-off scenarios.

We are only interested in the metrics that make our users happy.

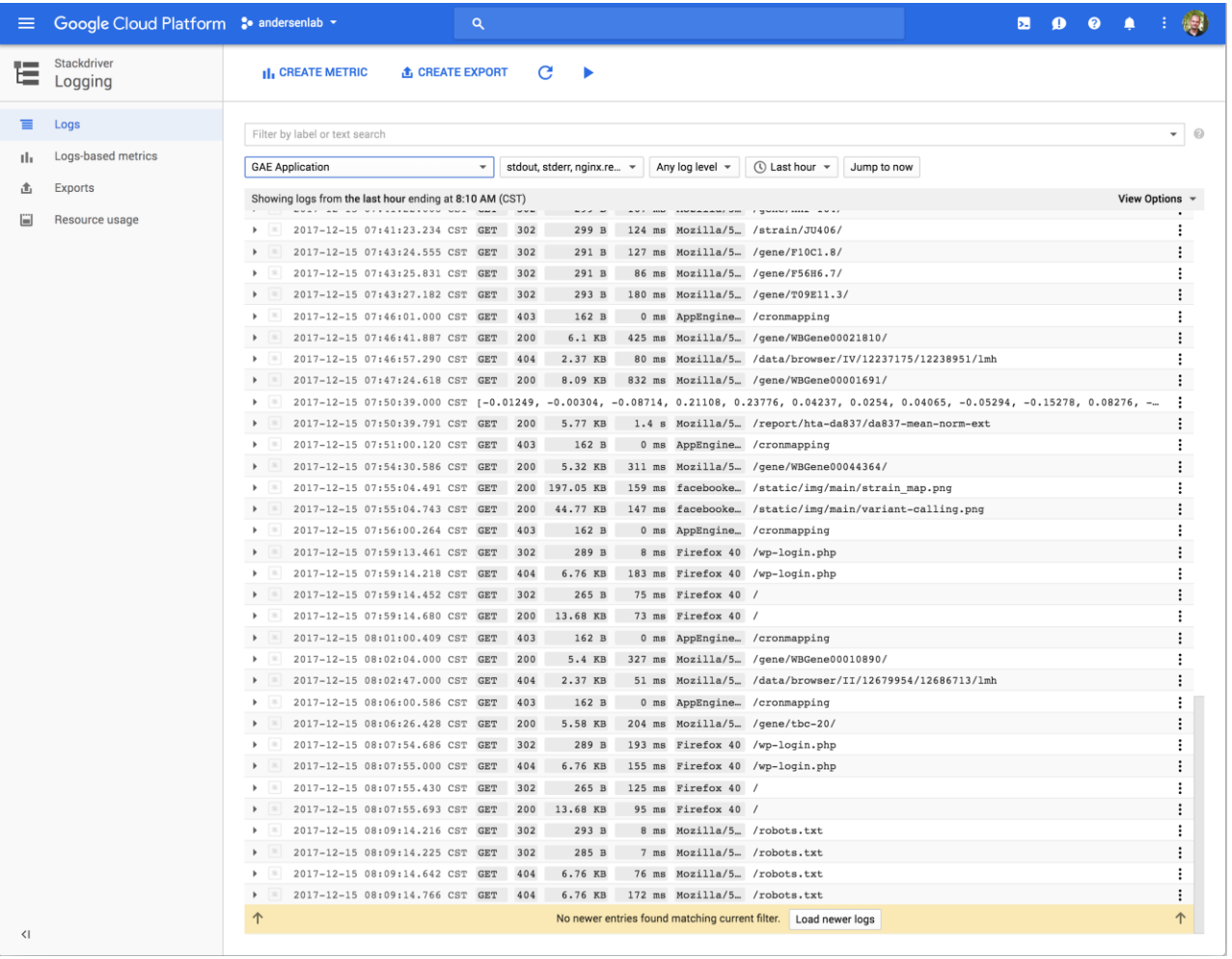
Happy Users
Happy Engineers

Where do I start?

How do we go from a few latency and error metrics to SLOs?

Metrics are our Service Level Indicators

Specifically chosen and measured at the closest point to the user.

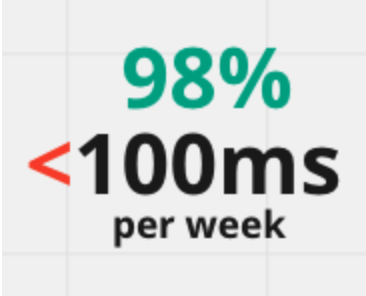


Simplified SLIs

SLI is a measurement of performance.

- Good events vs Total Events by Time
- 200s vs All Http Requests per Month
- All events that aren't 500s against all events.
- Money spent on compute vs GCP budget per year

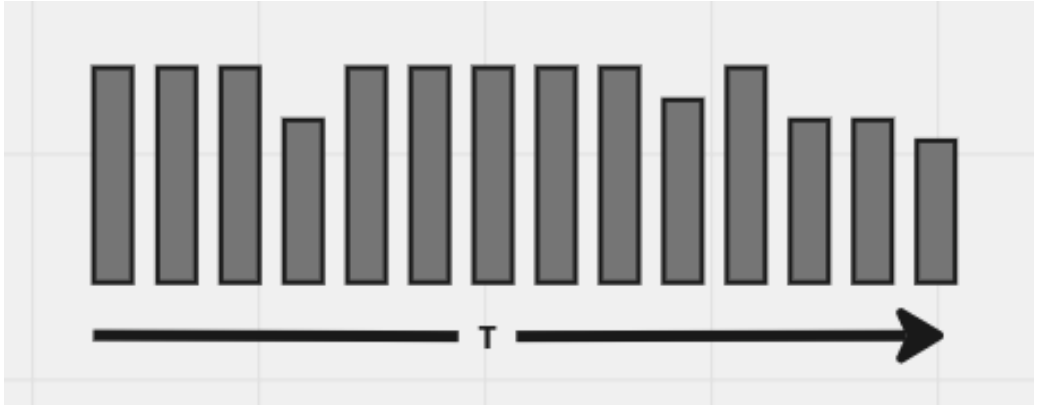
Request based SLIs



Measured counting atomic units of service.
Overall performance, but low granularity.

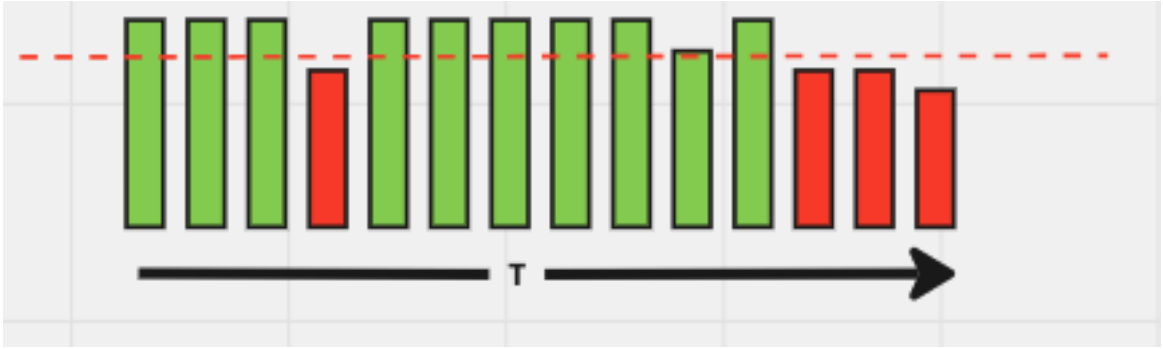
Windows-based SLIs

Group performance by time windows and count good vs bad windows.



95% http 20x responses per 1 minute window

P95 latency metric less than 100ms per 5 minute window



What makes a good SLI?

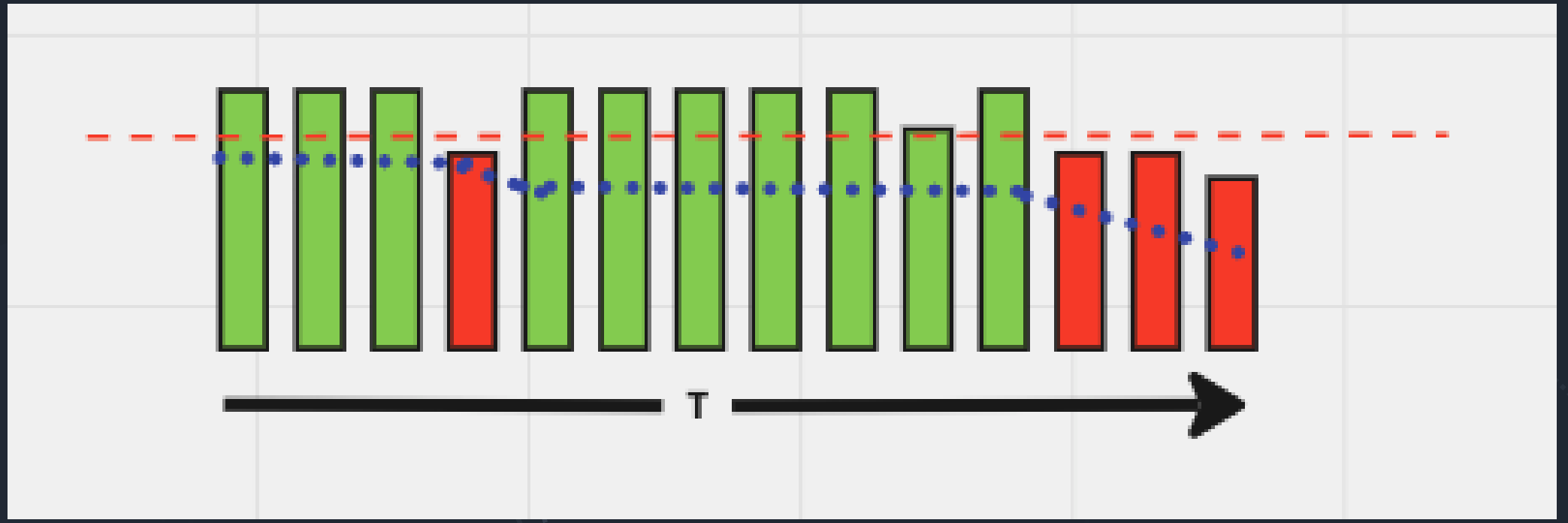
Metrics:

- Delta or Cumulative for Request Based SLI
- Delta, Cumulative or Gauge for Windowed SLI
- Not high cardinality

Time:

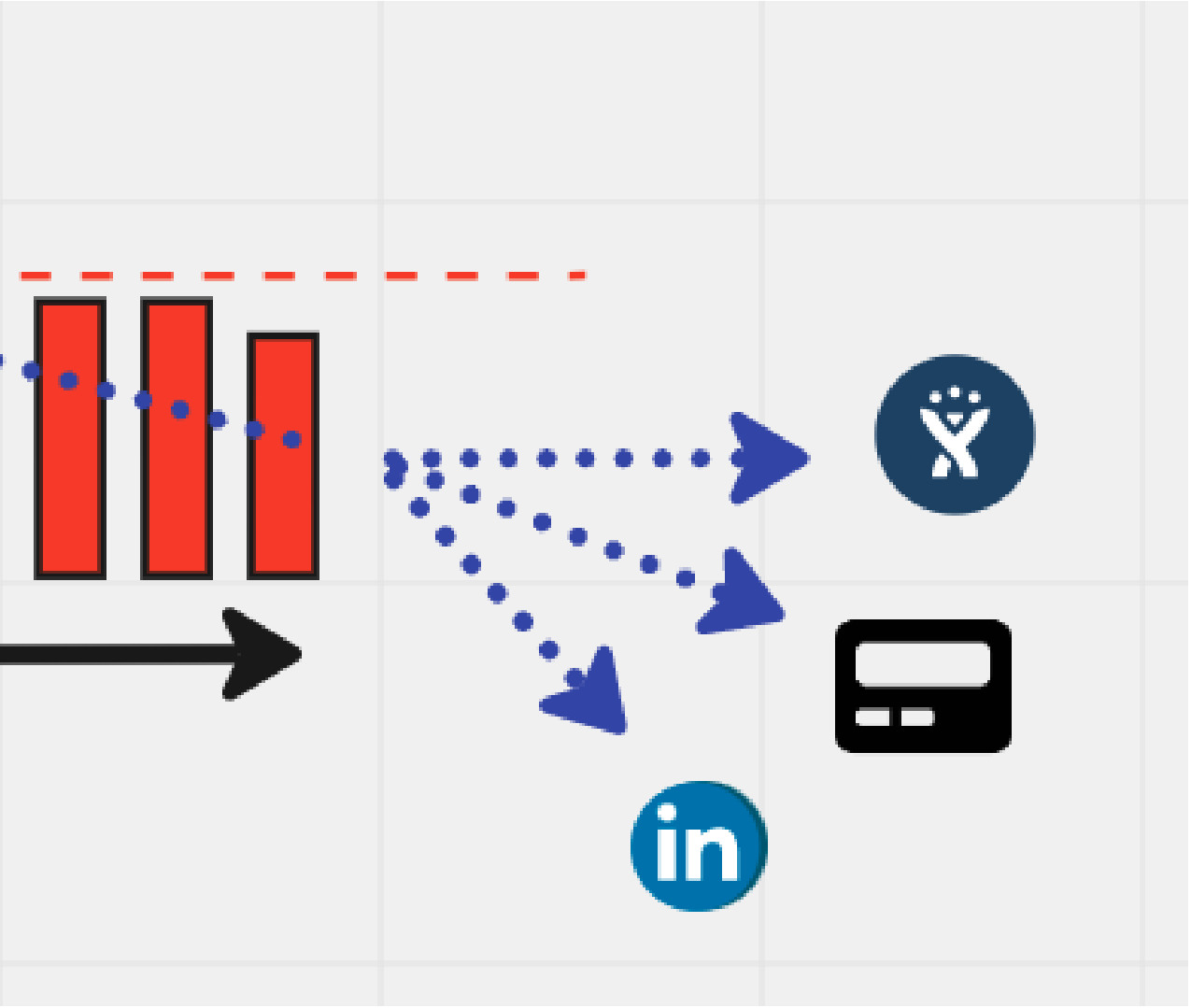
- Hours for alerting
- Weeks for tactical decisions
- Months for strategic decisions

Linear measurement of user happiness.
Percentage



Error Budgets

- Maximum amount of time a technical system can fail without contractual consequences.
- A measurement of the difference between actual and desired performance.
- When are users unhappy, when do people notice.



Burn Rate

The rate at which the error budget is consumed.

Make your alerting decisions based on your burn rate.

Clear indication if a problem needs immediate attention, if it can wait until morning or if it needs to be addressed in the next sprint.

It all comes down to time.

When describing the reliability of a service, the key denominator is time. Requests over time, errors over time, latency over time.

Availability and the mythical nine nines

Available minutes / total minutes

Fundamental layers set the limits of your reliability.
You can't have 99.999% reliability on a 99.9% reliable network with a 98% reliable database.
The cost of additional nines is exponential.

Appropriate Reliability

- What is reasonable for our service?
- What timeframe?
- How is our service being used?
- How many resources are we willing to commit to ensure higher reliability?
- Can we get eyes on a problem in time?

Service Level Agreements

The "do not cross" line.

SLA should always be lower than your SLO.

If you don't have an SLA, consider setting SLOs first as a test of what is possible.

Bust your SLA and the fun police get involved.

The role of a SLO

- Soft limit
- Performance expectation
- Not a fixed contract
- Meant to be revised, reviewed, updated.
- Do you engineer for perfection, or do you set reasonable expectations?

SLOs for Engineers

Do we need to alert the person on call or can this issue wait?

Do we have room in our error budget to deploy a major change?

Did that last deploy change our burn rate?

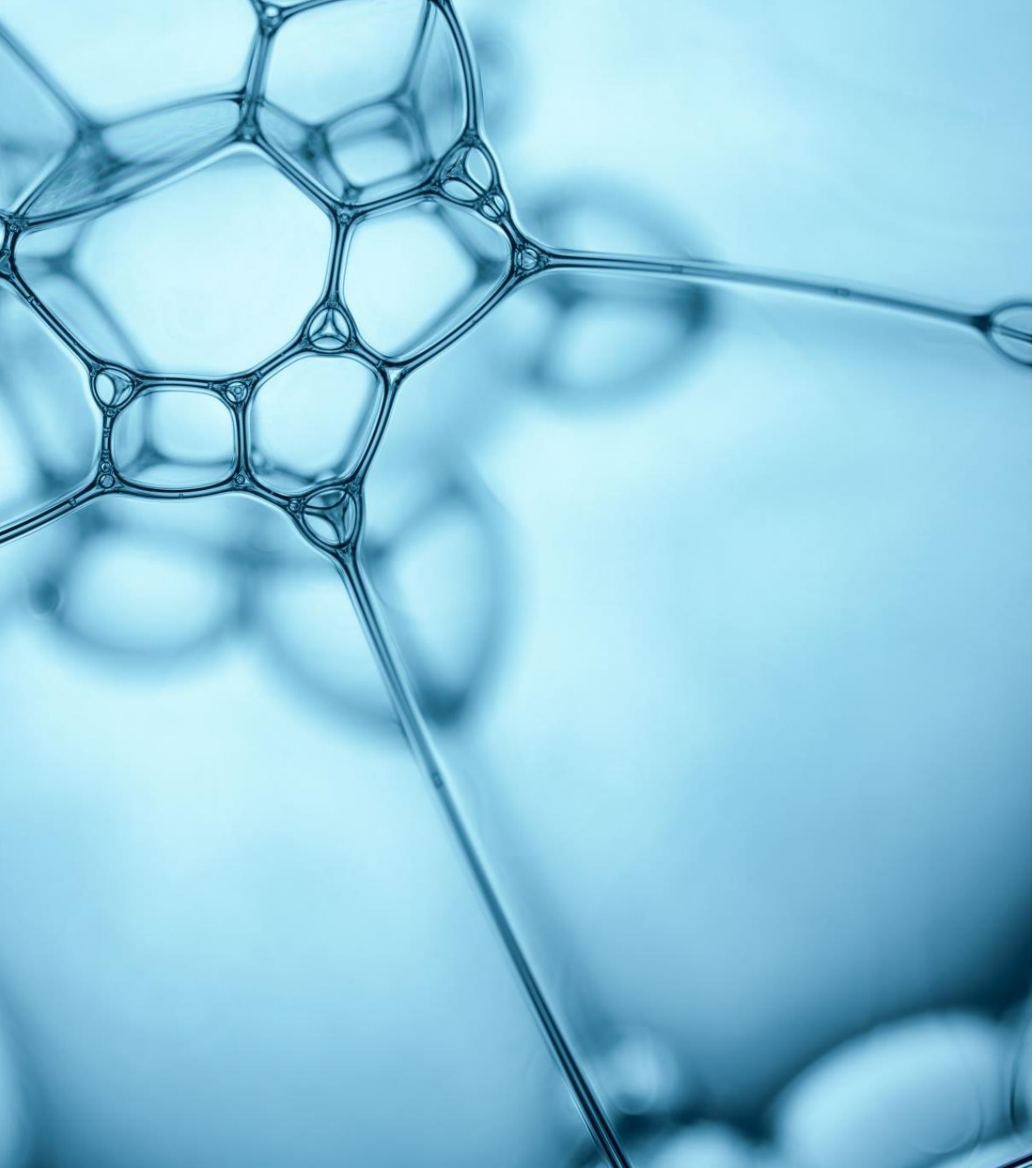
We blew through our error budget, let's do a blameless post-mortum.

SLOs for Product Owners

Are we living up to our SLAs?

Do we have room to add more features?

Are we making changes that make the consumers happy?



SLOs for Engineering Managers

Do we have the resources to increase our SLA?

Are we meeting our SLO goals?

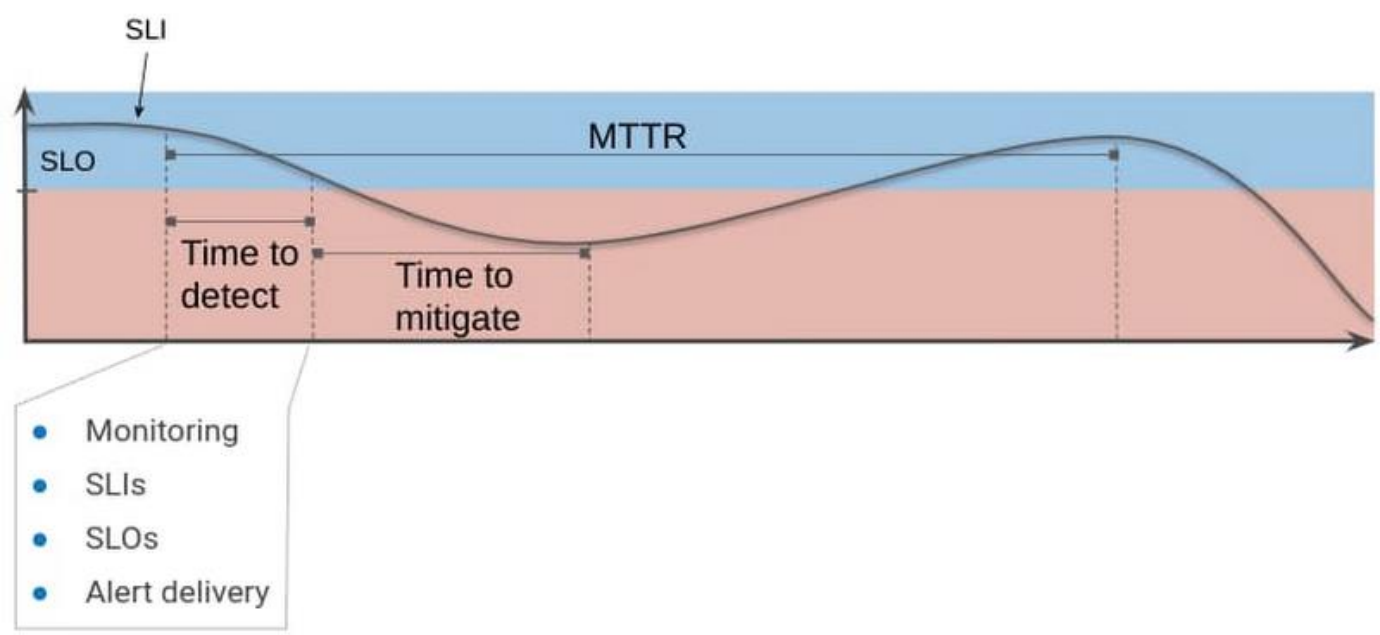
Do we focus on developing new features or increasing reliability?

Are we burning up on-call time for non-critical issues?

Do we focus on features or stability in the next sprint?

SLOs in the wild

- Share your SLIs and SLOs to create a reasonable expectation of service level for your consumers.
- Choose a synchronous or asynchronous means of sending data to an API based on latency and peak request SLIs.
- Communicate service issues to stakeholders.
- Allow teams to understand service issues without having deep technical knowledge of your service or share all your telemetry.



But my service is feature driven...

- SLO can be thought of as acceptable level of risk.
- SLOs can be an integral part of your DORA metrics.

Is it good enough?

Comparing SLOs to ITSI

Philosophical Differences

- Free vs Proprietary
- Decentralized vs Centralized
- Clear boundaries between business and operational data
- Tool agnostic vs Splunk specific
- Integral component driving discussion between engineers, EM and PO

Design Differences

- Measure as close as possible to consumer vs weighted service decomp
- SLO data can be collected by APIs*
- Burn Rates vs Adaptive Thresholds
- SLIs vs KPIs
- Metrics consumed where they are produced vs single source of truth
- DORA metrics vs Event Analytics
- Tracing as an automagic service map

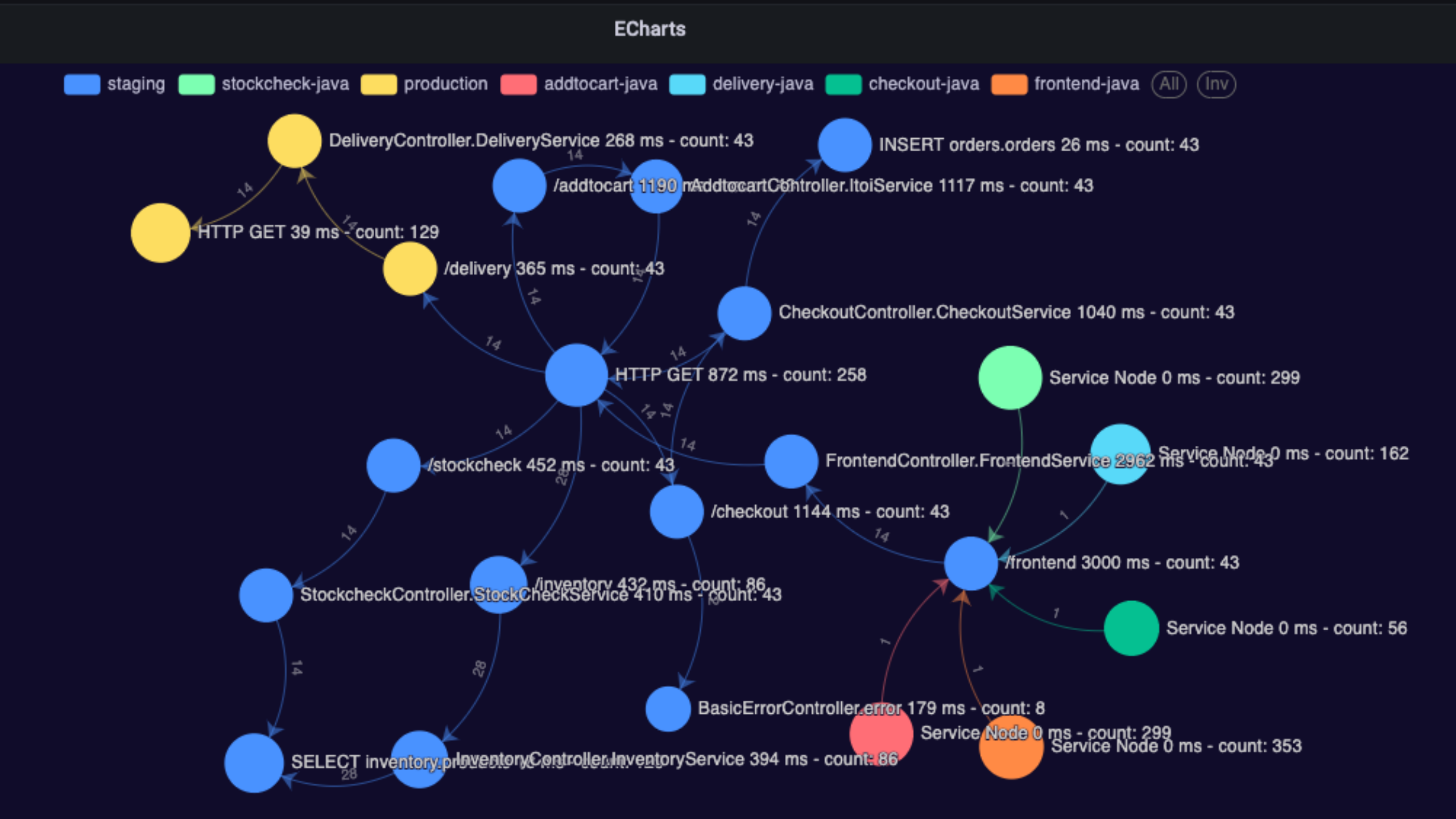
Thoughts?

Are you adopting SRE?

Is anyone mixing SRE and ITSI in production?

Would SLI or SLO data scraped from Prometheus and API or a Time-Series database be of any use as a KPI in ITSI?

Is anyone using trace propagation data to create service maps in Splunk?


ECharts
staging
stockcheck-java
production
addtocart-java
delivery-java
checkout-java
frontend-java
All
Inv
DeliveryController.DeliveryService 268 ms - count: 43
INSERT orders.orders 26 ms - count: 43
/addtocart 1190 ms - count: 43
AddtocartController.ItoiService 1117 ms - count: 43
HTTP GET 39 ms - count: 129
/delivery 365 ms - count: 43
CheckoutController.CheckoutService 1040 ms - count: 43
HTTP GET 872 ms - count: 258
Service Node 0 ms - count: 299
/stockcheck 452 ms - count: 43
FrontendController.FrontendService 2962 ms - count: 43
Service Node 0 ms - count: 162
/checkout 1144 ms - count: 43
/frontend 3000 ms - count: 43
StockcheckController.StockCheckService 410 ms - count: 43
/inventory 432 ms - count: 86
Service Node 0 ms - count: 56
BasicErrorController.error 179 ms - count: 8
Service Node 0 ms - count: 299
Service Node 0 ms - count: 353
SELECT inventory.products 42 ms - count: 172
InventoryController.InventoryService 394 ms - count: 86

Find out more here:

- https://monitoring.love/community/
- https://communityinviter.com/apps/cloud-native/cncf
- https://opentelemetry.io
- https://info.honeycomb.io/observability-engineering-oreilly-book-2022
- https://www.heavybit.com/library/podcasts/o11ycast
- https://sre.google
- https://medium.com/@YuriShkuro/temple-six-pillars-of-observability-4ac3e3deb402

Thanks!


TAKE THE FIRST STEPS
ON YOUR OBSERVABILITY JOURNEY
imgflip.com