Detecting service degradation and failures at scale

A PayPal Story

SRECON APAC 2019

PayPal

Who are we / About us

Yegya Narayanan
Sr.Architect
Tweets @gynarayan

Veeramani Gandan
Sr. Engg. Manager
Tweets @vgandanvs

Payments at Scale*

200+
Markets

100
Currencies

267M
Users

TPV
$578B

Multiple Data Centers

2700+
services

200K+
Servers

PayPal

*stats from 2018

Challenges with monitoring applications

Microservice architecture

Combination of stateful and stateless applications

Applications distributed across multiple regions

Applications deployed for active / active processing

Request processing can span multiple regions

Challenge: How can we monitor applications at such a scale?

Challenge: How can we monitor applications at such a scale?

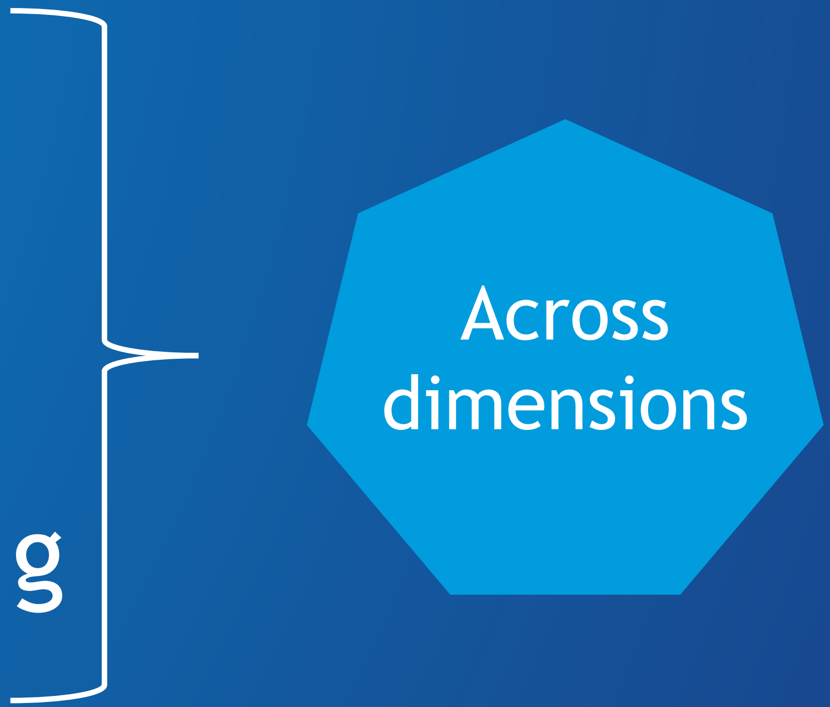
Our approach: Use logs to derive the golden signals

When services degrade

Increase in error rates

Increase in latency

Drops in request processing



CAL: Central Application Logging

Logging solution for PayPal

CAL: Central Application Logging

Logging solution for PayPal

Provides Three pillars of observability

CAL: Central Application Logging

Logging solution for PayPal

Provides Three pillars of observability

Application logging for monitoring & Triaging

CAL: Central Application Logging

Logging solution for PayPal

Provides Three pillars of observability

Application logging for monitoring & Triaging

Distributed Tracing with 100% coverage

CAL: Central Application Logging

Logging solution for PayPal

Provides Three pillars of observability

Application logging for monitoring & Triaging

Distributed Tracing with 100% coverage

Metrics for monitoring

Application log structure

Application logs have an implicit structure

- Type of operation (URL or API)
- Name of operation
- Latency
- Status (success, failure, Bad data)

Application log structure

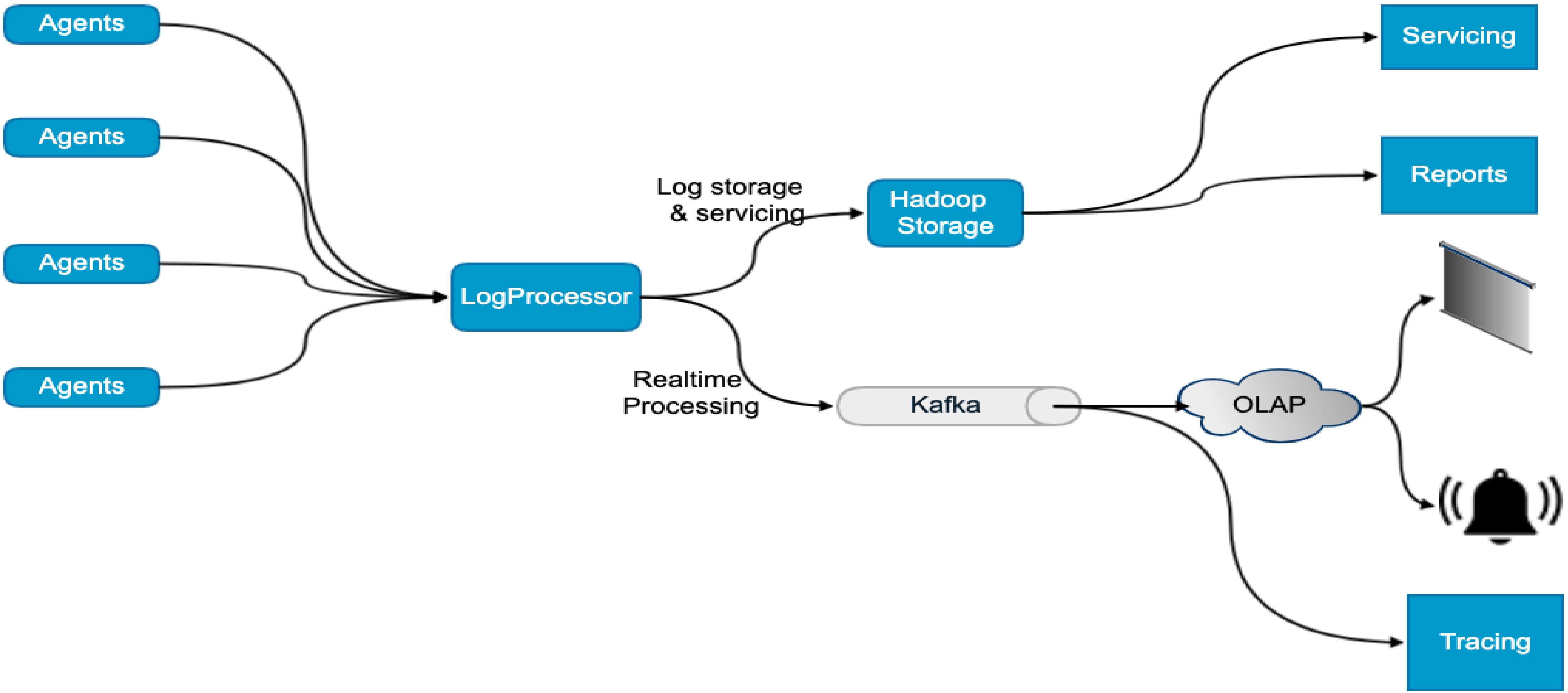
Application logs have an implicit structure

- Type of operation (URL or API)
- Name of operation
- Duration
- Status (success, failure, Bad data)

Metadata is added at deployment

- Region, host, application, build version

Log Processing Pipeline



Log processing

Logs are centralized within a region

- Processed and stored locally for servicing
- Log volume is a significant factor for replication

Log processing

Logs are centralized within a region

- Processed and stored locally for servicing
- Log volume is a significant factor for replication

Specific log types are filtered for real time processing

- Generate metrics through custom OLAP and Druid
- Provide distributed tracing

Some stats

10+ Trillion messages per day

Some stats

10+ Trillion messages per day

1.3PB of uncompressed logs

Some stats

10+ Trillion messages per day

1.3PB of uncompressed logs

500M+ messages per minute for metrics

Some stats

10+ Trillion messages per day

1.3PB of uncompressed logs

500M+ messages per minute for metrics

~8M unique traces per minute

Why not sampling?

Every customer interaction is unique

Why not sampling?

Every customer interaction is unique

Some interactions are more important

Why not sampling?

Every customer interaction is unique

Some interactions are more important

Servicing the request is data driven

- Same API can have different call stacks for two different users
- Same API can have different call stacks for same user

Why not sampling?

Every customer interaction is unique

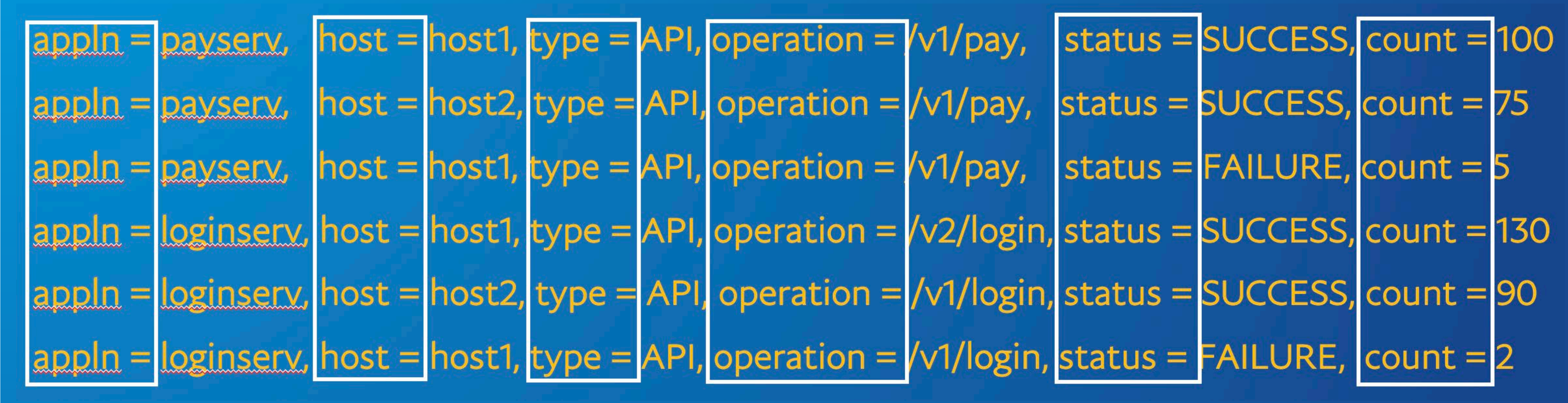
Some interactions are more important

Servicing the request is data driven

- Same API can have different call stacks for two different users
- Same API can have different call stacks for same user

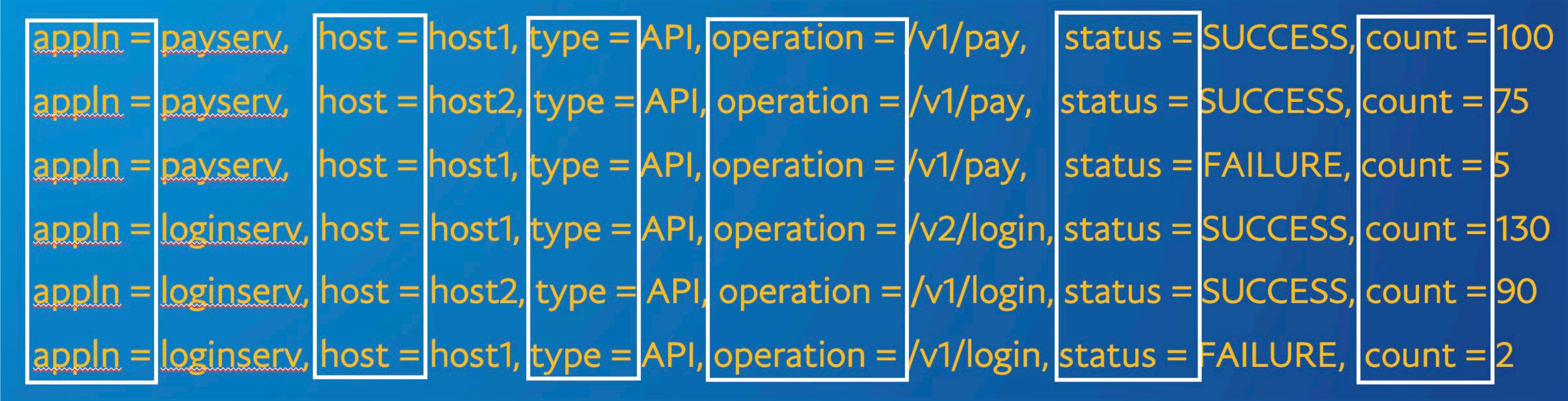
Not sufficient to reason system state at a point in time

Metrics from implicit structure



```
appln = payserv, host = host1, type = API, operation = /v1/pay, status = SUCCESS, count = 100
appln = payserv, host = host2, type = API, operation = /v1/pay, status = SUCCESS, count = 75
appln = payserv, host = host1, type = API, operation = /v1/pay, status = FAILURE, count = 5
appln = loginserv, host = host1, type = API, operation = /v2/login, status = SUCCESS, count = 130
appln = loginserv, host = host2, type = API, operation = /v1/login, status = SUCCESS, count = 90
appln = loginserv, host = host1, type = API, operation = /v1/login, status = FAILURE, count = 2
```

Metrics from implicit structure



```
appln = payserv, host = host1, type = API, operation = /v1/pay, status = SUCCESS, count = 100
appln = payserv, host = host2, type = API, operation = /v1/pay, status = SUCCESS, count = 75
appln = payserv, host = host1, type = API, operation = /v1/pay, status = FAILURE, count = 5
appln = loginserv, host = host1, type = API, operation = /v2/login, status = SUCCESS, count = 130
appln = loginserv, host = host2, type = API, operation = /v1/login, status = SUCCESS, count = 90
appln = loginserv, host = host1, type = API, operation = /v1/login, status = FAILURE, count = 2
```

Included (but not shown) are

- Timestamp for causal ordering
- TraceId for distributed tracing
- Keys for business monitoring (e.g. country, flow type, currency)

PayPal

Generating metrics

All types are not equal

Generating metrics

All types are not equal

Use specific log types to capture signals

Generating metrics

All types are not equal

Use specific log types to capture signals

Logs can be annotated for business metrics

Generating metrics

All types are not equal

Use specific log types to capture signals

Logs can be annotated for business metrics

Useful metrics to derive

- Connect failures
- Request rates
- Latency

Metrics from Application logs

Semantic values and metadata converted to tags (dimensions)

Count and latency are aggregated as metrics

Generated Metrics aggregated across regions

Metrics are generated and queried across multiple tags (e.g.)

- Success count for a given application
- Error count per application
- Error count per application per host

Detection & Response

Based on historical trends

- WoW, DoD

Percentile distribution

- Latency

Across dimensions

When services degrade

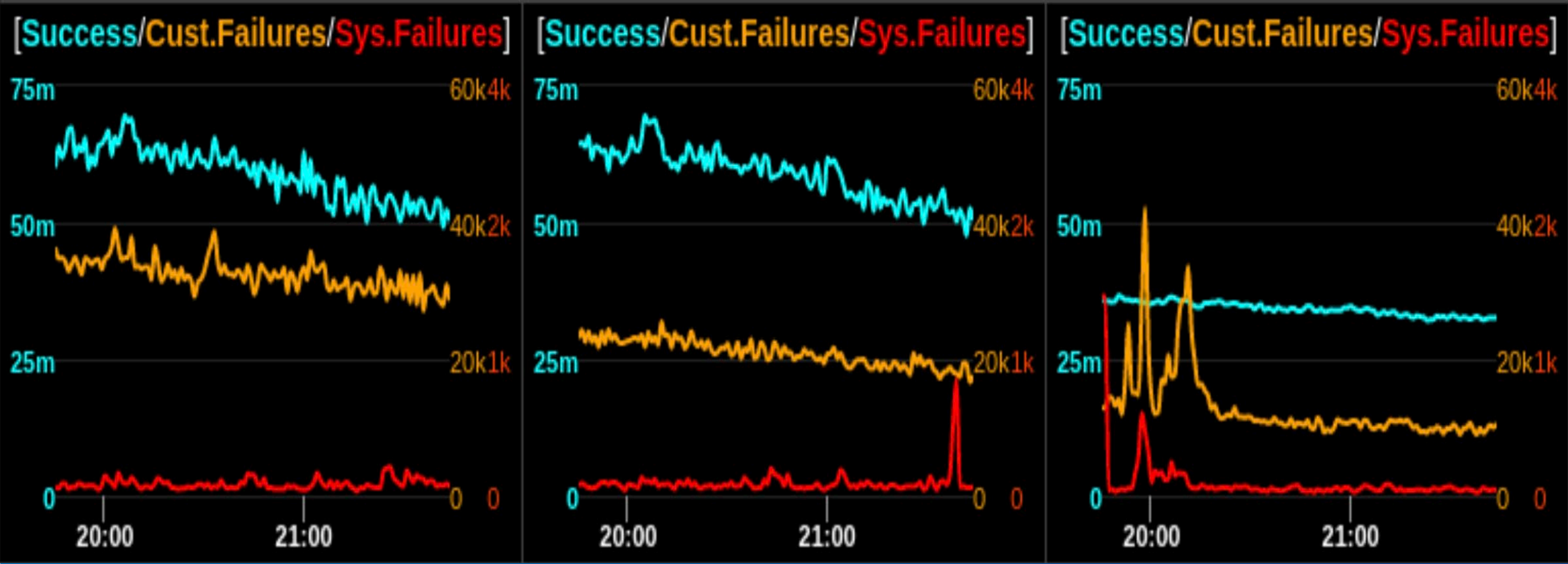
Increase in error rates

Increase in latency

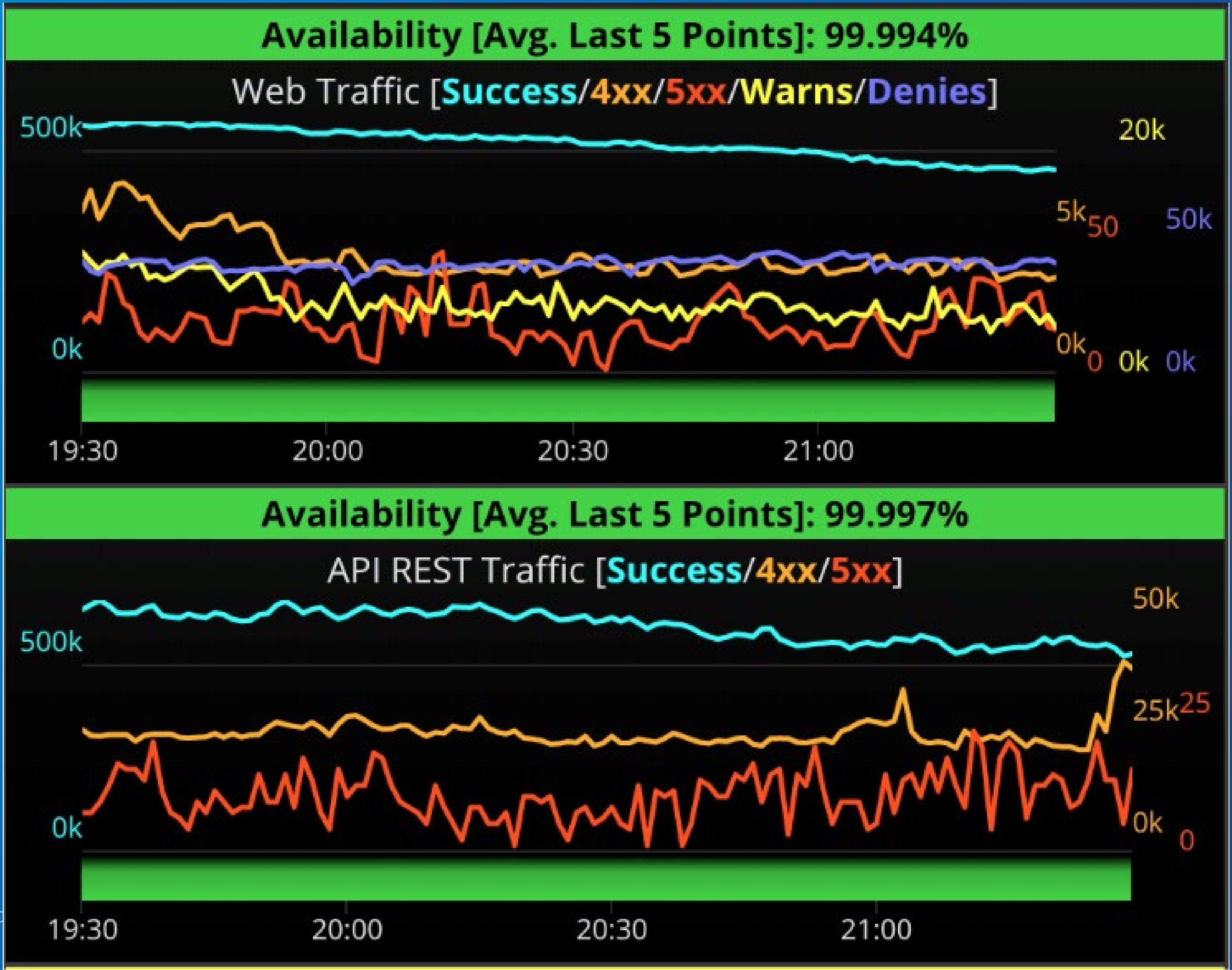
Drops in request processing

Across dimensions

Metric based view



Real time dashboards



Availability [Avg. Last 5 Points]: 99.994%

Web Traffic [Success/4xx/5xx/Warns/Denies]

Availability [Avg. Last 5 Points]: 99.997%

API REST Traffic [Success/4xx/5xx]

Our learnings

Log based metrics better indicator of application performance

Our learnings

Log based metrics better indicator of application performance

Aggregation better with metrics than logs

Our learnings

Log based metrics better indicator of application performance

Aggregation better with metrics than logs

Metrics for TTD and Logs for TTR

Our learnings

Log based metrics better indicator of application performance

Aggregation better with metrics than logs

Metrics for TTD and Logs for TTR

Logging hygiene important to reduce noise

Thank you

Question time ☺