BREX

# Observing Elixir Microservices

Thomas Césaré-Herriau & Vamsi Chitters

BREX

## Table of Contents

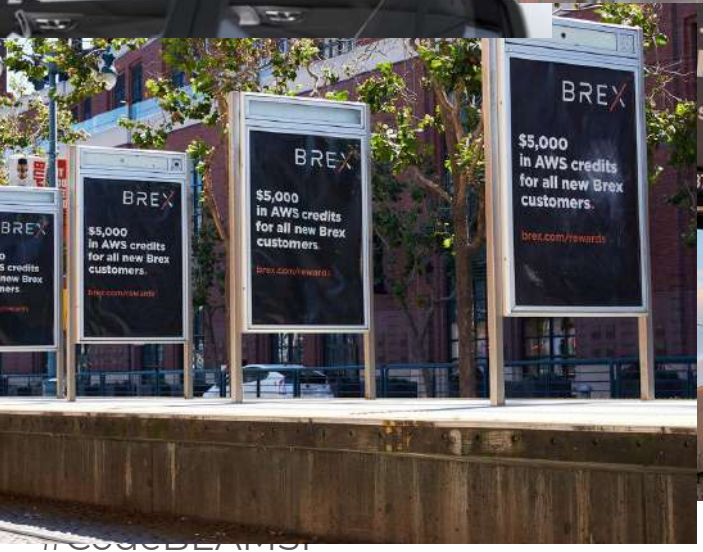
# What is Brex?

BREX


BREX


$5,000
in AWS credits
for all new Brex
customers.
brex.com/rewards

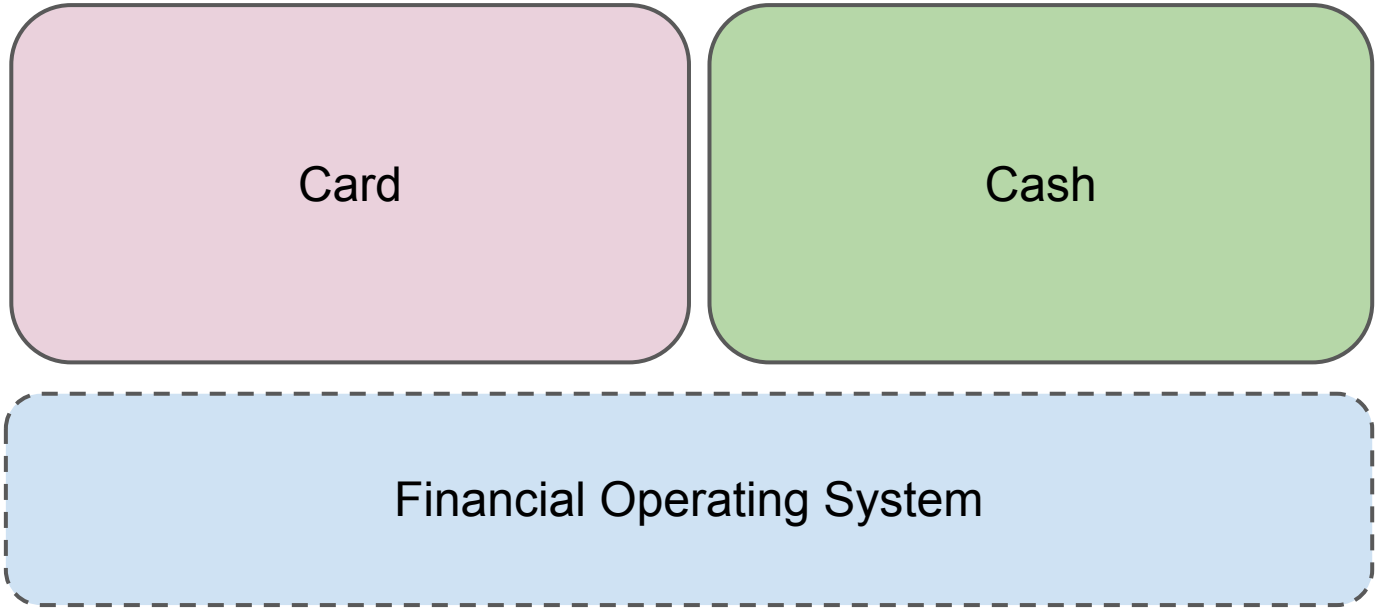

BREX
$101,552,027.49
in rewards back to startups in 3 months
BREX.COM/REWARDS


BREX
$5,000
in AWS credits
for all new Brex
customers.
brex.com/rewards


BREX
Rewards better
than a money tree.
76

BREX

# What is Brex?




**Brex is rebuilding B2B financial products**

BREX

# What is Brex?

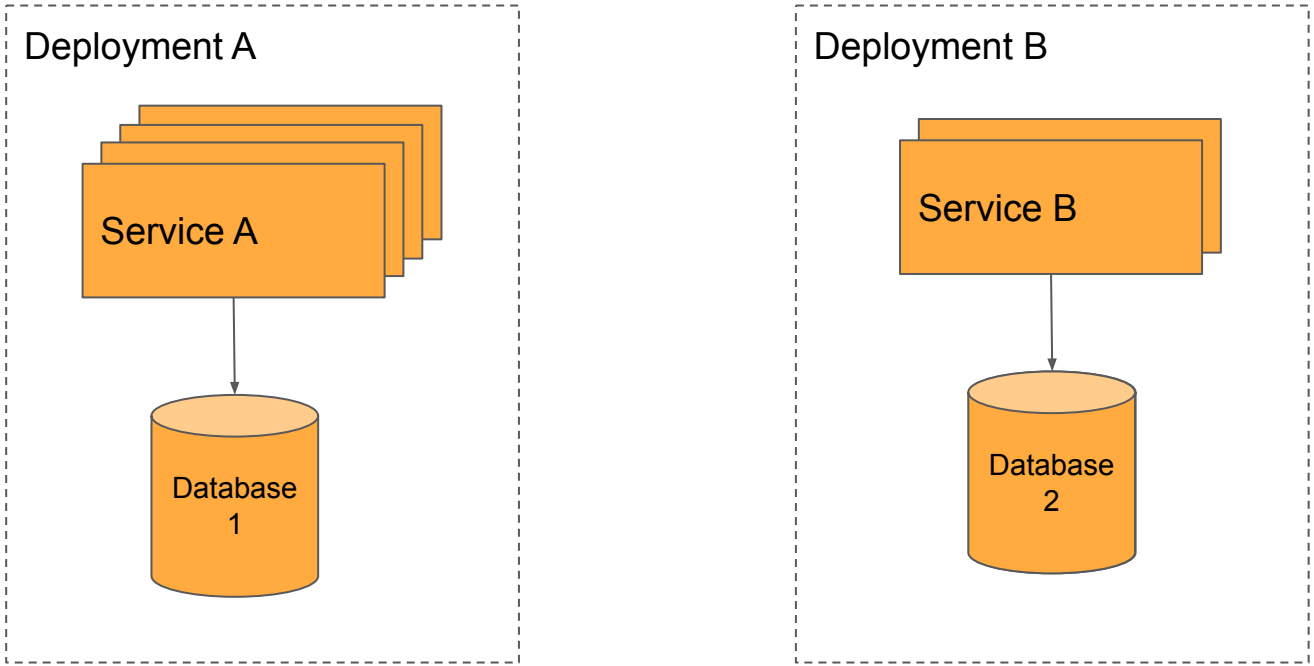
- 4 offices (NY, SF, YVR, SLC)
- 10 engineering teams
- 90% of backend codebase in Elixir
- More than 30 microservices

BREX

# 30 microservices?

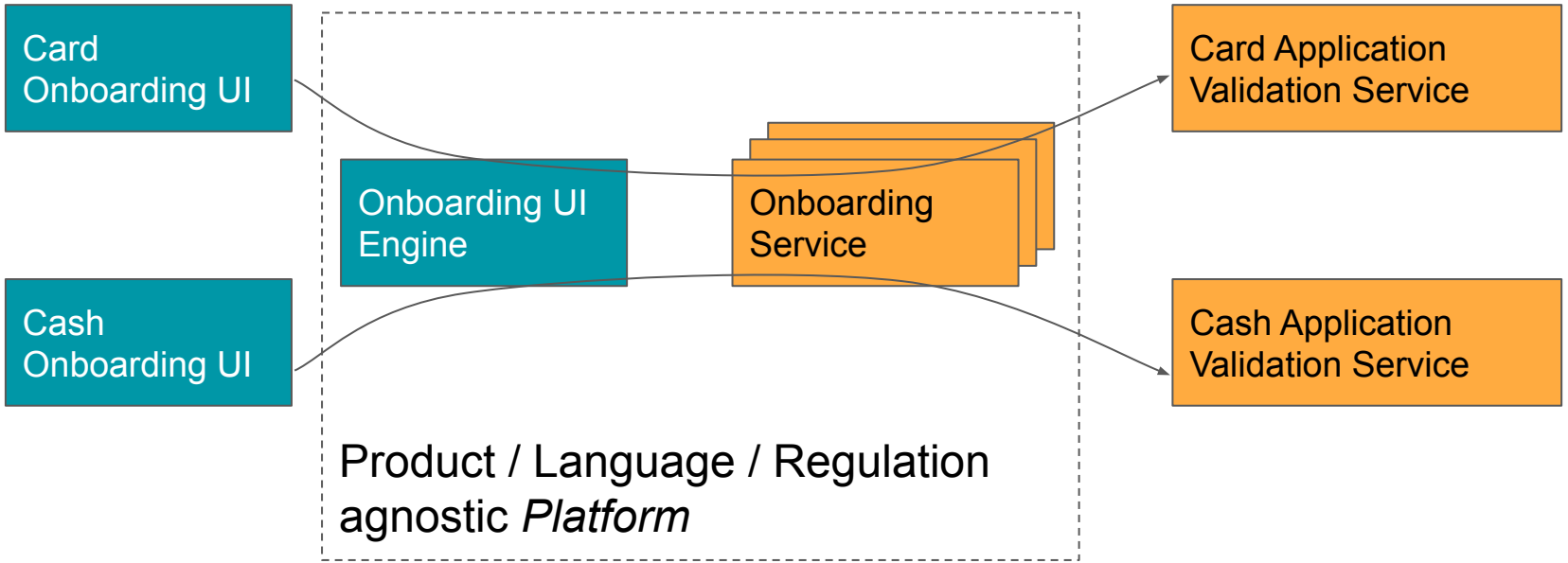
BREX

# Enables services to be developed, deployed and scaled independently



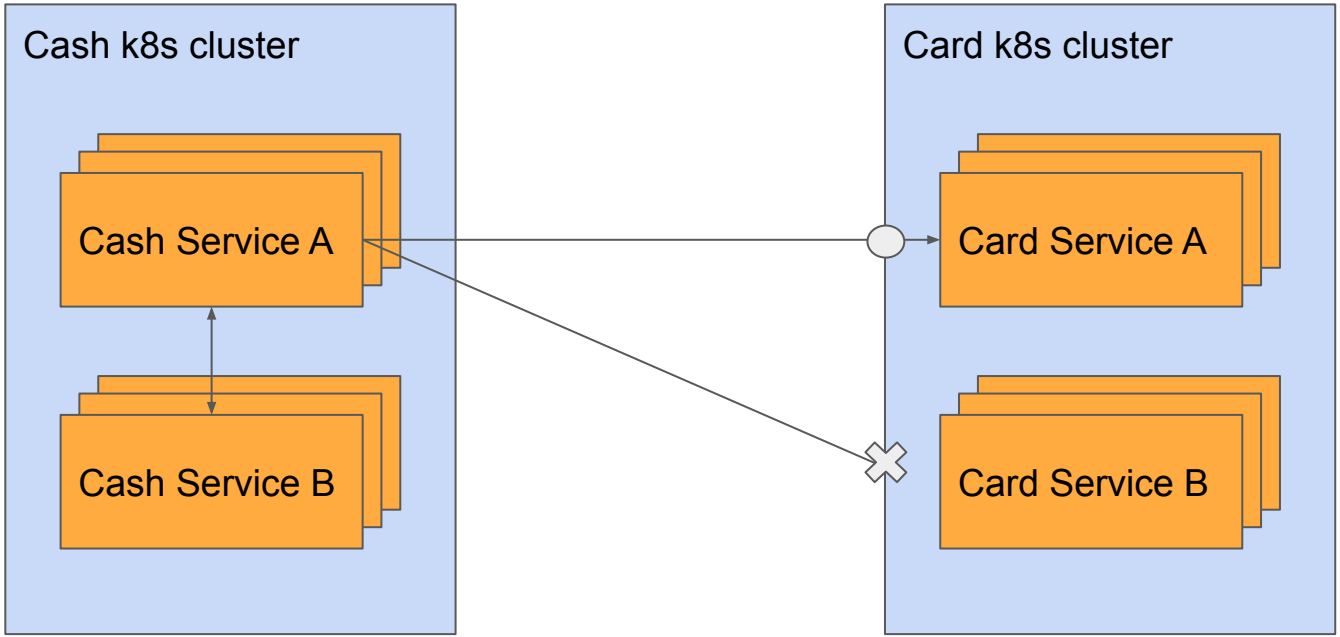
BREX

# Platform approach to building our systems



BREX

# Isolate products (Cash vs Card)



BREX

# Microservices at Brex

- Enables services to be developed, deployed and scaled independently
- Platform approach to building our systems
- Isolate products (Cash vs Card)

BREX

# Brex Communication Infrastructure

- **gRPC:** synchronous RPC calls between Brex Services
- **Events Infrastructure (Kafka):** asynchronous communication

# Observability In Theory

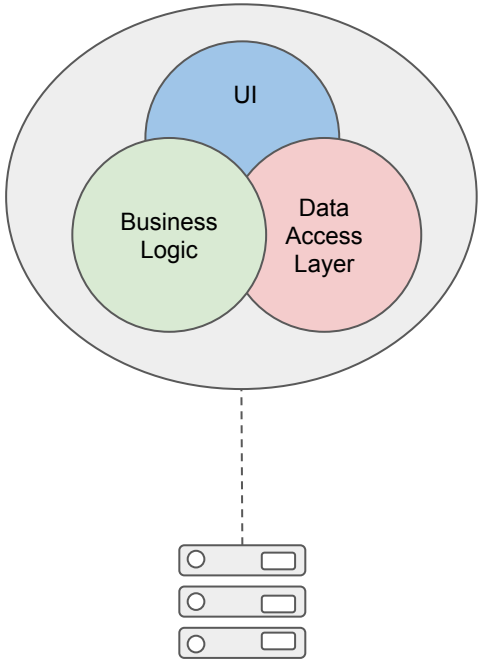

Why *observability*? And why not *monitoring*?

BREX

# Traditional Monitoring

BREX

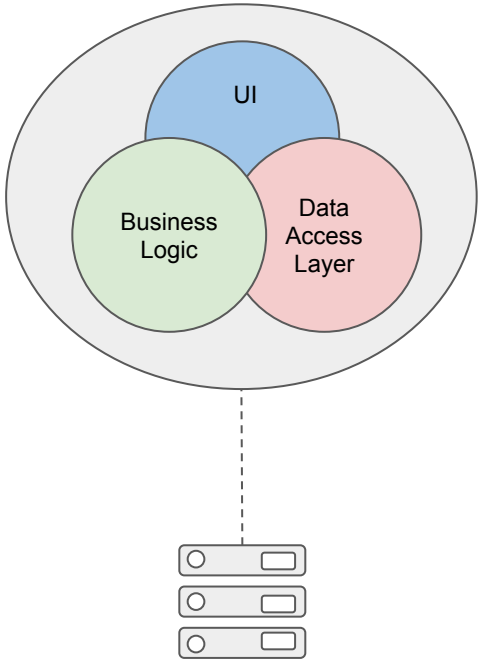
# Traditional Monitoring




Monolithic Services

BREX

# Traditional Monitoring




Monolithic Services

Developers building software while Ops/SRE implement monitoring

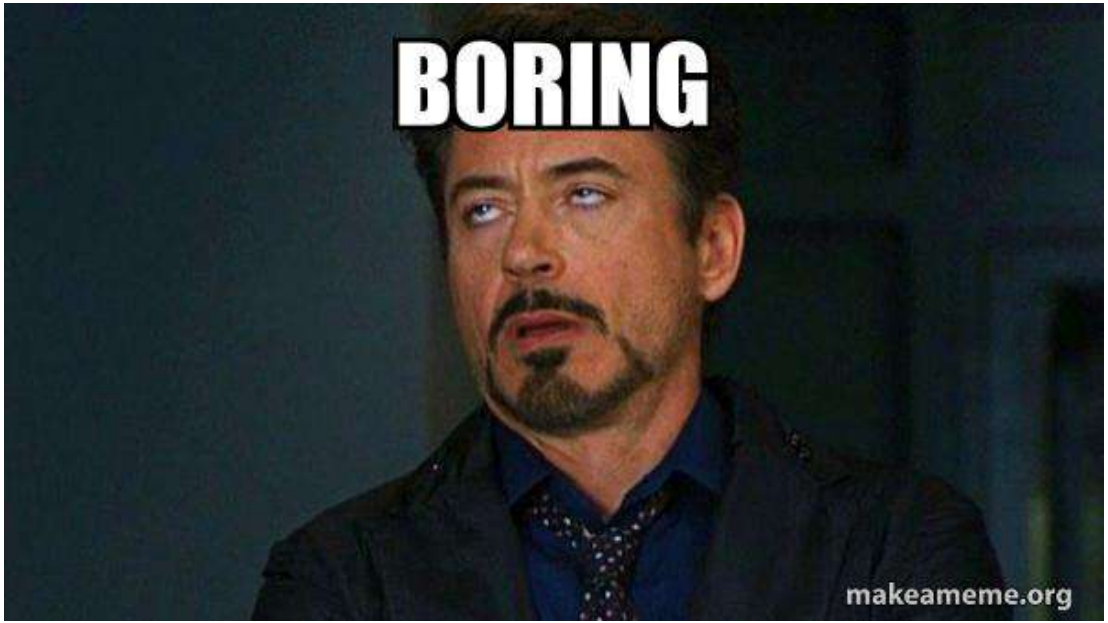
BREX

# Limitations of traditional **monitoring**

- Mostly focused on **known failures**
- Historically not designed for **distributed systems**
- Traditionally implemented **after** a system is built

BREX

Last but not least, **rhymes with...**



BREX

# Observability

*“Observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs”*

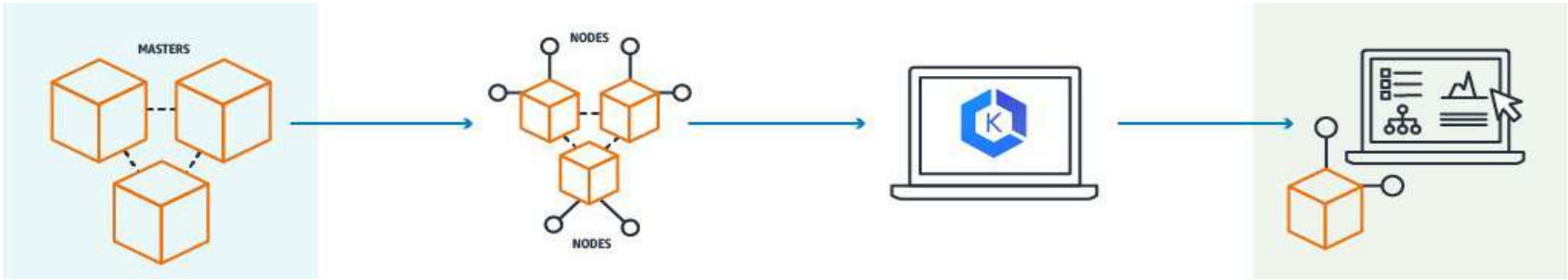
BREX

# Observability

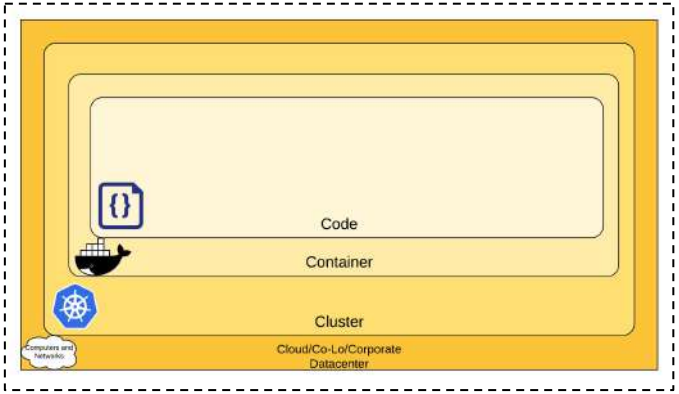
- Understand quickly **new, unpredicted failures**
- Principle of designing **observable** services
- A **cloud native** approach

BREX

# Observability: a cloud native approach

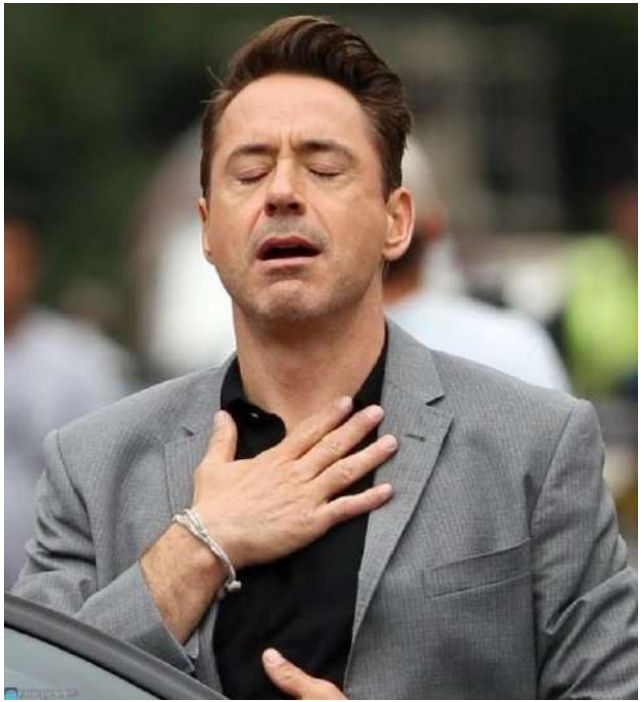



- Containerized
- Dynamically orchestrated
- Microservices-oriented



BREX

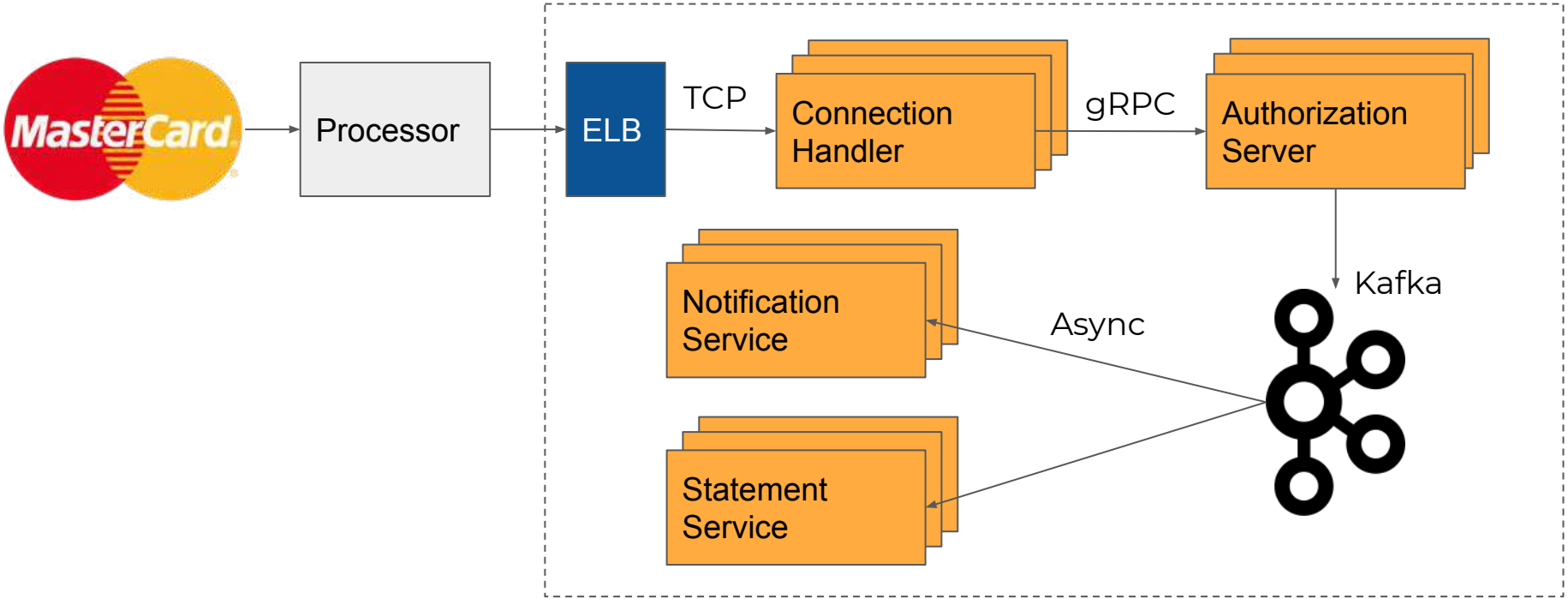
# *Observability*: so much better



# Observability In Practice

BREX

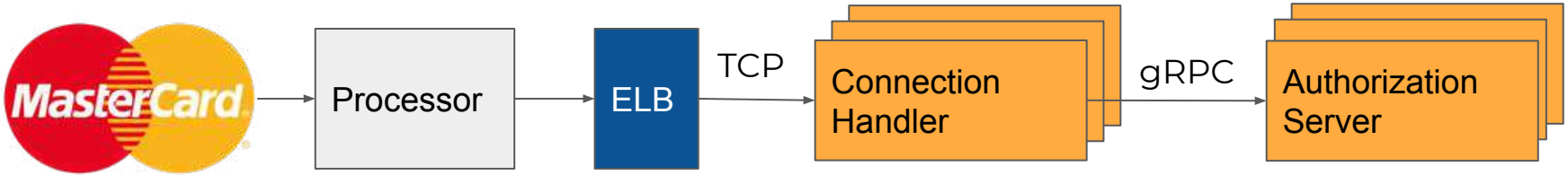
# Case Study: Authorization flow



# Question 1

BREX

# Why are authorizations timing out?




- Credit card network notifying our systems of requests taking more than 2s
- Multiple services in the path of an authorization: which one is the bottleneck?

BREX

Question: Why are authorizations timing out?

# Solution: Distributed tracing

BREX

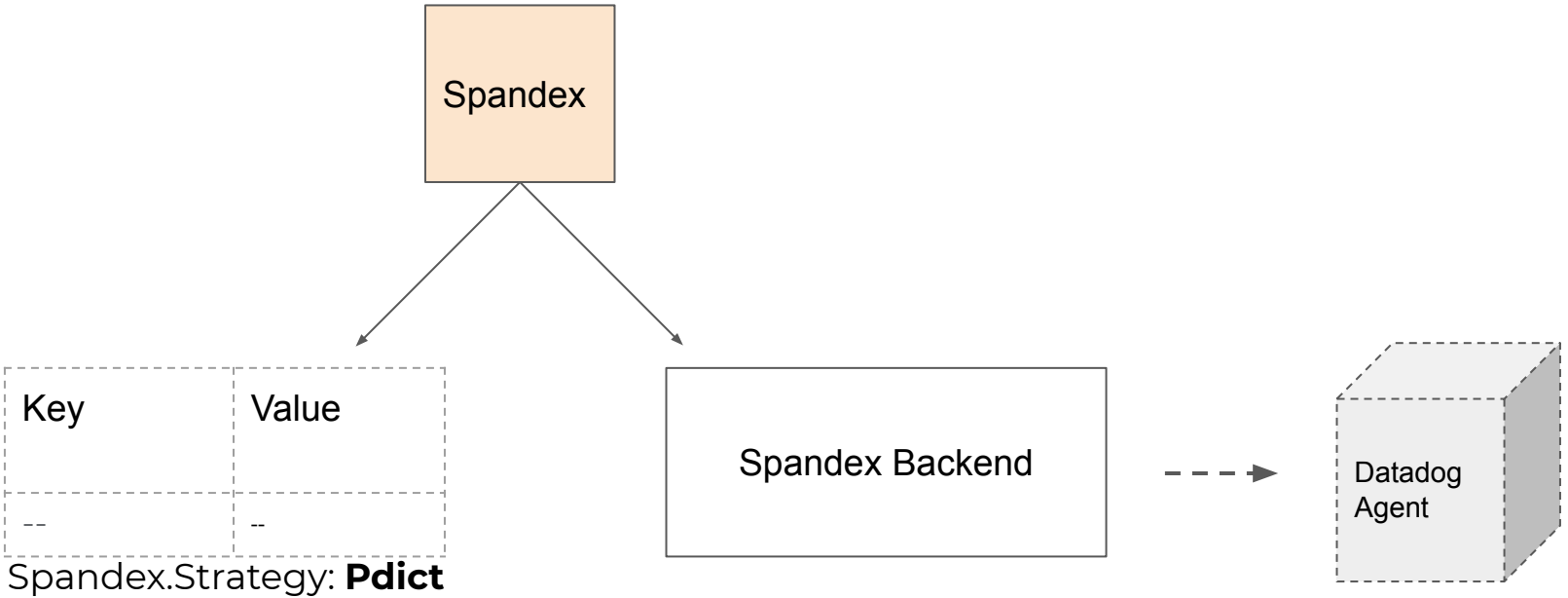
Question: Why are authorizations timing out?

# Solution: Distributed tracing

- `spandex` library: Tracer API for Elixir with pluggable backend
  - Supports Datadog as backend (through the Agent)
  - Comes with built-in `Plug` adapters
- Interceptor pattern to serialize and deserialize tracing context in gRPC headers

BREX

# Spandex Components

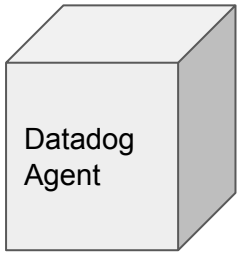


BREX

# Spandex (default **Pdict** strategy) Example

{:ok, _} = Tracer.start_trace("123", “network”)

Spandex




| Key | Value |
| --- | --- |
| `{:spandex_trace , Network.Tracer}`<br><br>`where Network.Tracer is an example trace_key` | %Spandex.Trace{bag gage: [], id: 707096, priority: 1, **spans: [], stack: []**} |

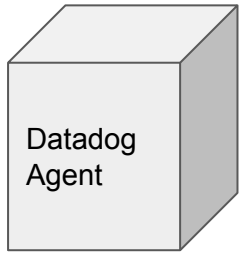
BREX

# Spandex (default **Pdict** strategy) Example

```
{:ok, _} = TestTracer.start_trace("123", “network”)

TestTracer.start_span("queue")
```

Spandex




| Key | Value |
| --- | --- |
| `{:spandex_trace , Network.Tracer}`<br><br>`where Network.Tracer is an example trace_key` | %Spandex.Trace{bag gage: [], id: 707096, priority: 1, spans: [], **stack: [%Spandex.Span{id: 744186, name: "123", trace_id: 707096, ...}]}** |

BREX

# Spandex (default **Pdict** strategy) Example

```
{:ok, _} = TestTracer.start_trace("123", “network”)

TestTracer.start_span("queue")

TestTracer.finish_span()
```

Spandex

Datadog Agent

| Key | Value |
| --- | --- |
| `{:spandex_trace , Network.Tracer}`<br><br>`where Network.Tracer is an example trace_key` | %Spandex.Trace{bag gage: [], id: 707096, priority: 1, **spans: [%Spandex.Span{id: 875246, parent_id: 744186, trace_id: 707096, ...}],** stack: [...]} |

BREX

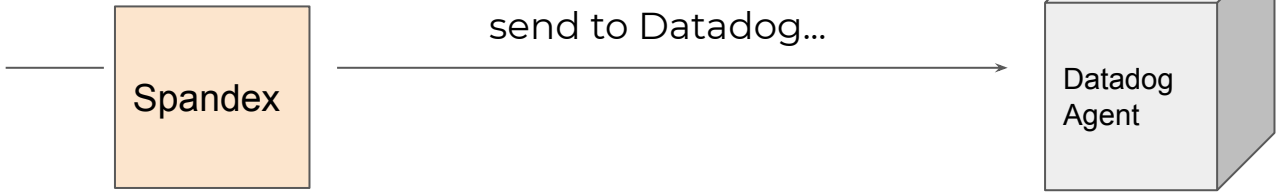
# Spandex (default **Pdict** strategy) Example

```
{:ok, _} = TestTracer.start_trace("123", “network”)

TestTracer.start_span("queue")

TestTracer.finish_span()

TestTracer.finish_trace()
```



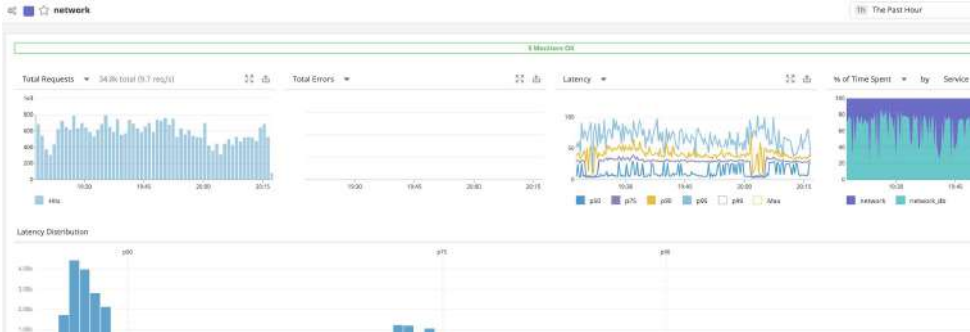

| Key | Value |
| --- | --- |

BREX

# Quick note on consistent service name

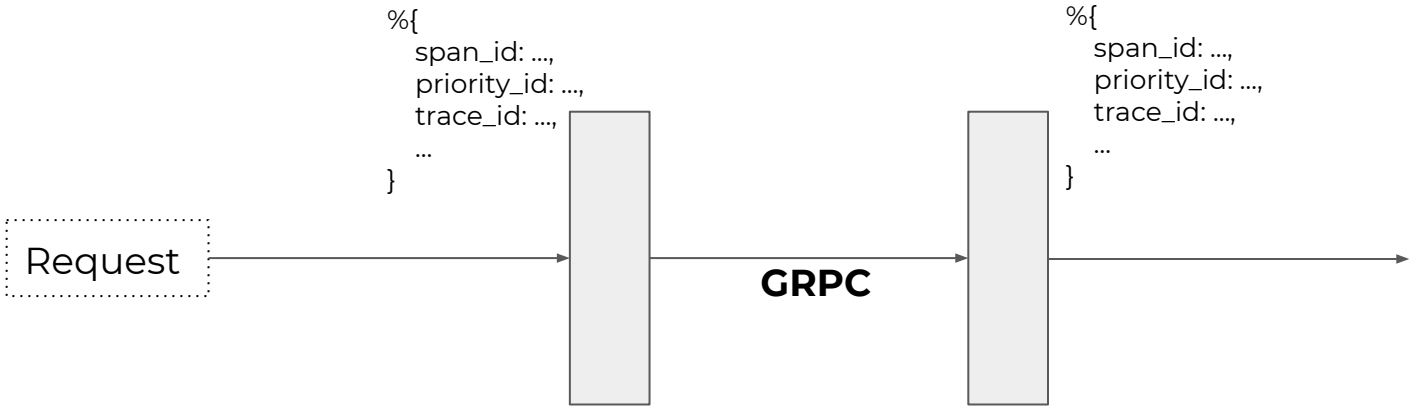
```
64        env:
65        - name: GRPC_PORT
66          value: "8080"
67        - name: ENABLE_NETWORK_PUBLISH_THROUGH_EVENTS
68          value: "true"
69        - name: MIX_ENV
70          value: prod
71        - name: DATADOG_AGENT_HOST
72          valueFrom:
73            fieldRef:
74              fieldPath: status.hostIP
75        - name: DATADOG_AGENT_PORT
76          value: "8126"
77        - name: DATADOG_AGENT_STATSD_PORT
78          value: "8125"
79        - name: SENTRY_PROJECT
80          value: "network"
81        - name: TRACE_SERVICE
82          value: {{ template "app.name" . }}
```

```
26      @doc """
27      Fetches the canonical service name from the system environment.
28      """
29      def service_name() do
30        System.get_env("TRACE_SERVICE") || "unknown_service"
31      end
```



BREX

# Plug / Interceptor pattern

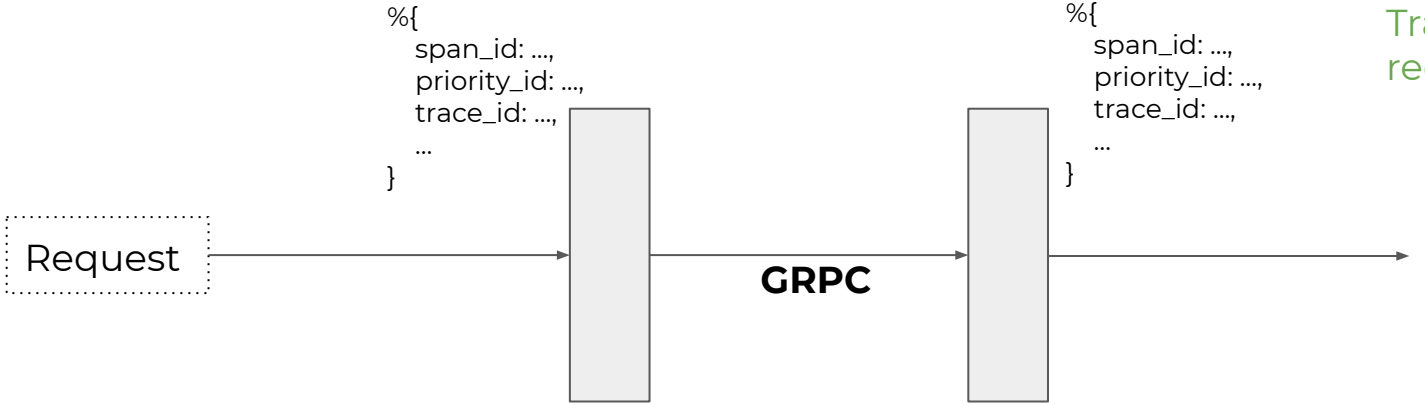



Client Interceptor adds tracing_ctx to header

Server Interceptor retrieves tracing_ctx from header

BREX

# Plug / Interceptor pattern




Tracing context recovered!

Client Interceptor adds tracing_ctx to header

Server Interceptor retrieves tracing_ctx from header

BREX

# Problem: some traces are losing spans

- We mapped lost spans to operations executed within a *spawned process*
  - Tracing context is not propagated
- `spandex` Tracing library stores the tracing context in the `pdict`
  - `pdict` is tied to a process

BREX

Problem: Some traces are losing some spans.

# Solution: `Brex.Context`

- A *caller-aware* context
- When get/set and not yet initialized, looks up the chain of callers
  - `Process.get("$callers")` available since Elixir v1.8
- And stores a parent context into its own `pdict`

BREX

Problem: Some traces are losing some spans.

# How does Brex.Context work?

```
Process.info(pid, :dictionary) → %{
        :brex_context => %Brex.Context{
            identity_context: …
            customer_identity_context: …
            trace_id: …
            span_id: …
            priority: ...
        }
}
```

BREX

Problem: Some traces are losing some spans.

# How does Brex.Context work?

| pid | process dict |
| --- | --- |
| 123 (parent) | %Brex.Context{...trace_id: 1} |
| 456 (child) | |

BREX

Problem: Some traces are losing some spans.

# How does Brex.Context work?

| pid | process dict |
| --- | --- |
| 123 (parent) | %Brex.Context{...trace_id: 1} |
| 456 (child) | |

**PID: 456**

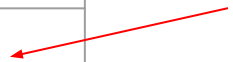
Brex.Context.get_trace_id()

BREX

Problem: Some traces are losing some spans.

# How does Brex.Context work?

| pid | process dict |
| --- | --- |
| 123 (parent) | %Brex.Context{...trace_id: 1} |
| 456 (child) | X |



**PID: 456**

Brex.Context.get_trace_id()

BREX

Problem: Some traces are losing some spans.

# How does Brex.Context work?

| pid | process dict |
| --- | --- |
| 123 (parent) | %Brex.Context{...trace_id: 1} |
| 456 (child) | |

**PID: 456**

Brex.Context.get_trace_id()

iex(1)> Process.get("$callers")

[#PID<123>]

BREX

Problem: Some traces are losing some spans.

# How does Brex.Context work?

| pid | process dict |
| --- | --- |
| 123 (parent) | %Brex.Context{...trace_id: 1} |
| 456 (child) | %Brex.Context{...trace_id: 1} |

**PID: 456**

Brex.Context.get_trace_id()

> 1

We fixed the bottleneck!

# Question 2

BREX

# Background

- Some customers reported not receiving text message after a successful transaction
- Turned out some services would crash for different reasons

BREX

How do we ensure our team is notified when this happens?

BREX

Question: How do we ensure our team is notified when this happens?

# Solution: Alerts on all exceptions/errors

- We rolled out `Sentry` everywhere
- Super easy:
  - `Logger.add_backend(Sentry.LoggerBackend)`

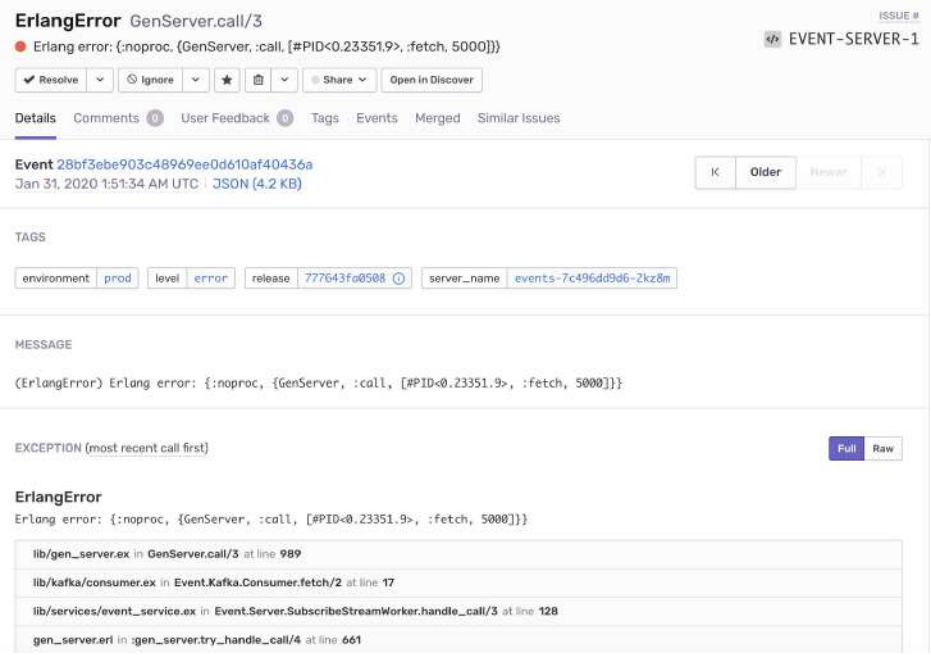
BREX

Question: How do we ensure our team is notified when this happens?

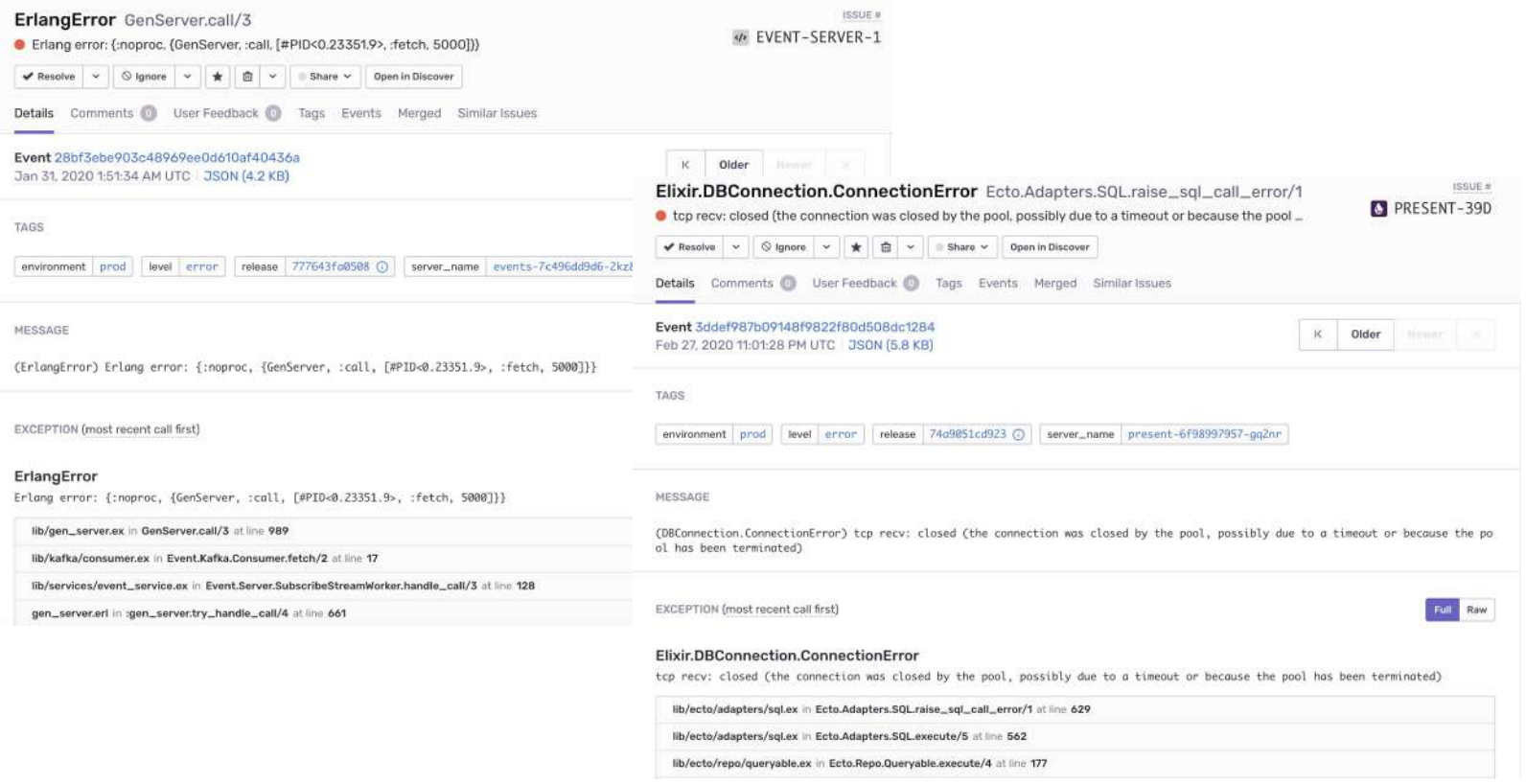
# Sentry everywhere

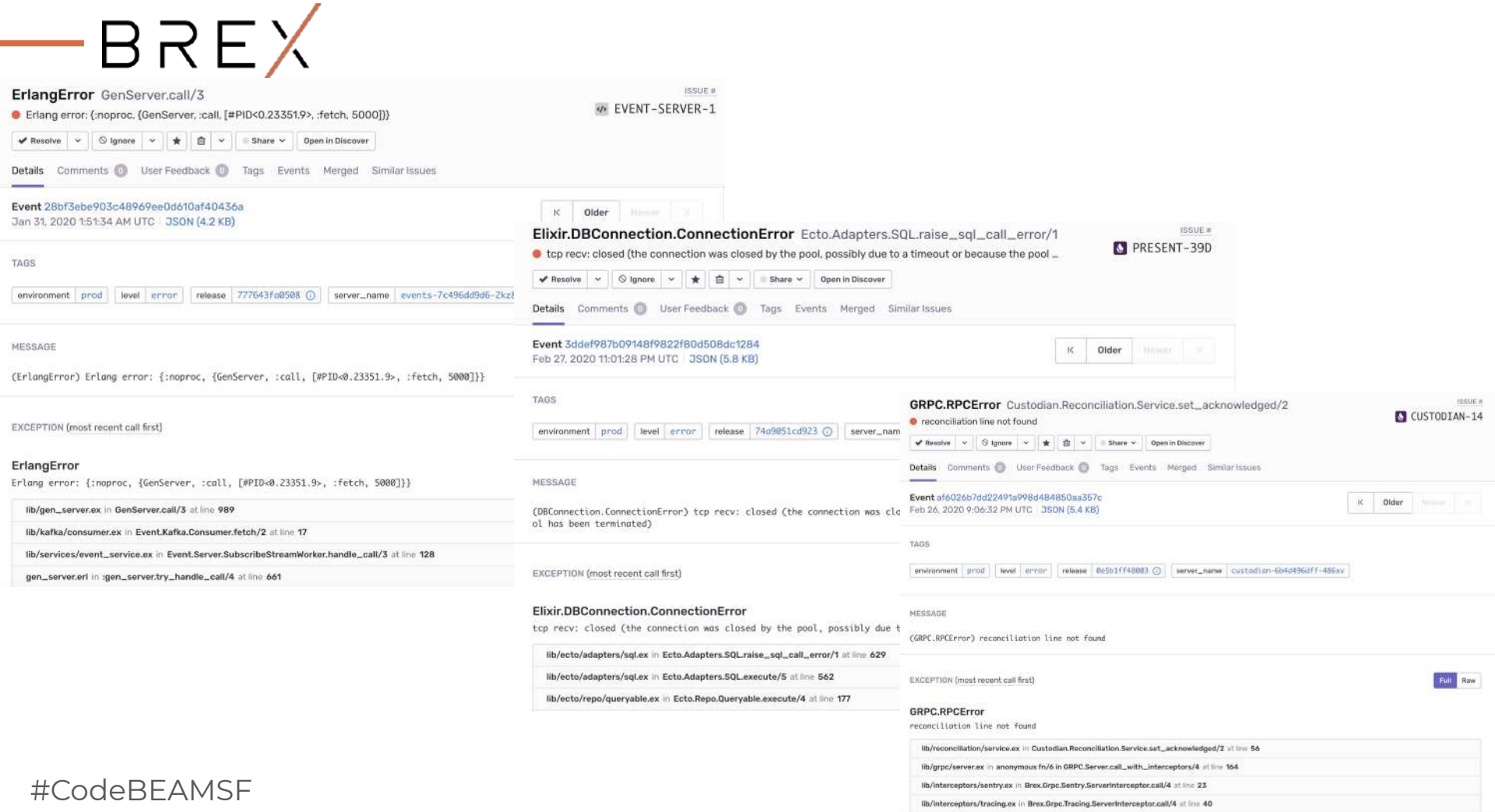
Done! Our microservices architecture is fully monitored!

BREX


ErlangError GenServer.call/3
Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
ISSUE #
EVENT-SERVER-1
Resolve
Ignore
Share
Open in Discover
Details
Comments 0
User Feedback 0
Tags
Events
Merged
Similar Issues
Event 28bf3ebe903c48969ee0d610af40436a
Jan 31, 2020 1:51:34 AM UTC | JSON (4.2 KB)
K
Older
Newer
J
TAGS
environment prod
level error
release 777643fa0508
server_name events-7c496dd9d6-2kz8m
MESSAGE
(ErlangError) Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
EXCEPTION (most recent call first)
Full
Raw
ErlangError
Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
lib/gen_server.ex in GenServer.call/3 at line 989
lib/kafka/consumer.ex in Event.Kafka.Consumer.fetch/2 at line 17
lib/services/event_service.ex in Event.Server.SubscribeStreamWorker.handle_call/3 at line 128
gen_server.erl in :gen_server.try_handle_call/4 at line 661

BREX


ErlangError GenServer.call/3
Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
Resolve
Ignore
Share
Open in Discover
ISSUE #
EVENT-SERVER-1
Details
Comments 0
User Feedback 0
Tags
Events
Merged
Similar Issues
Event 28bf3ebe903c48969ee0d610af40436a
Jan 31, 2020 1:51:34 AM UTC | JSON (4.2 KB)
K
Older
Newer
J
TAGS
environment prod
level error
release 777643fa0508
server_name events-7c496dd9d6-2kz8
MESSAGE
(ErlangError) Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
EXCEPTION (most recent call first)
ErlangError
Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
lib/gen_server.ex in GenServer.call/3 at line 989
lib/kafka/consumer.ex in Event.Kafka.Consumer.fetch/2 at line 17
lib/services/event_service.ex in Event.Server.SubscribeStreamWorker.handle_call/3 at line 128
gen_server.erl in :gen_server.try_handle_call/4 at line 661
Elixir.DBConnection.ConnectionError Ecto.Adapters.SQL.raise_sql_call_error/1
tcp recv: closed (the connection was closed by the pool, possibly due to a timeout or because the pool _
Resolve
Ignore
Share
Open in Discover
ISSUE #
PRESENT-39D
Details
Comments 0
User Feedback 0
Tags
Events
Merged
Similar Issues
Event 3ddef987b09148f9822f80d508dc1284
Feb 27, 2020 11:01:28 PM UTC | JSON (5.8 KB)
K
Older
Newer
J
TAGS
environment prod
level error
release 74a9051cd923
server_name present-6f98997957-gq2nr
MESSAGE
(DBConnection.ConnectionError) tcp recv: closed (the connection was closed by the pool, possibly due to a timeout or because the pool has been terminated)
EXCEPTION (most recent call first)
Full
Raw
Elixir.DBConnection.ConnectionError
tcp recv: closed (the connection was closed by the pool, possibly due to a timeout or because the pool has been terminated)
lib/ecto/adapters/sql.ex in Ecto.Adapters.SQL.raise_sql_call_error/1 at line 629
lib/ecto/adapters/sql.ex in Ecto.Adapters.SQL.execute/5 at line 562
lib/ecto/repo/queryable.ex in Ecto.Repo.Queryable.execute/4 at line 177


ErlangError GenServer.call/3
Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
Resolve
Ignore
Share
Open in Discover
ISSUE #
EVENT-SERVER-1
Details
Comments
User Feedback
Tags
Events
Merged
Similar Issues
Event 28bf3ebe903c48969ee0d610af40436a
Jan 31, 2020 1:51:34 AM UTC | JSON (4.2 KB)
TAGS
environment prod
level error
release 777643fa0508
MESSAGE
(ErlangError) Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
EXCEPTION (most recent call first)
ErlangError
Erlang error: {:noproc, {GenServer, :call, [#PID<0.23351.9>, :fetch, 5000]}}
lib/gen_server.ex in GenServer.call/3 at line 989
lib/kafka/consumer.ex in Event.Kafka.Consumer.fetch/2 at line 17
lib/services/event_service.ex in Event.Server.SubscribeStreamWorker.handle_call/3 at line 128
gen_server.erl in :gen_server.try_handle_call/4 at line 661
Older
Elixir.DBConnection.ConnectionError Ecto.Adapters.SQL.raise_sql_call_error/1
tcp recv: closed (the connection was closed by the pool, possibly due to a timeout or because the pool
PRESENT-39D
Event 3ddef987b09148f9822f80d508dc1284
Feb 27, 2020 11:01:28 PM UTC | JSON (5.8 KB)
TAGS
environment prod
level error
release 74a9051cd923
MESSAGE
(DBConnection.ConnectionError) tcp recv: closed (the connection was clo
ol has been terminated)
EXCEPTION (most recent call first)
Elixir.DBConnection.ConnectionError
tcp recv: closed (the connection was closed by the pool, possibly due t
lib/ecto/adapters/sql.ex in Ecto.Adapters.SQL.raise_sql_call_error/1 at line 629
lib/ecto/adapters/sql.ex in Ecto.Adapters.SQL.execute/5 at line 562
lib/ecto/repo/queryable.ex in Ecto.Repo.Queryable.execute/4 at line 177
GRPC.RPCError Custodian.Reconciliation.Service.set_acknowledged/2
reconciliation line not found
CUSTODIAN-14
Event af6026b7dd22491a998d484850aa357c
Feb 26, 2020 9:06:32 PM UTC | JSON (5.4 KB)
TAGS
environment prod
level error
release 0e5b1ff48003
server_name custodian-6b4d496dff-486xv
MESSAGE
(GRPC.RPCError) reconciliation line not found
EXCEPTION (most recent call first)
Full
Raw
GRPC.RPCError
reconciliation line not found
lib/reconciliation/service.ex in Custodian.Reconciliation.Service.set_acknowledged/2 at line 56
lib/grpc/server.ex in anonymous fn/6 in GRPC.Server.call_with_interceptors/4 at line 164
lib/interceptors/sentry.ex in Brex.Grpc.Sentry.ServerInterceptor.call/4 at line 23
lib/interceptors/tracing.ex in Brex.Grpc.Tracing.ServerInterceptor.call/4 at line 40

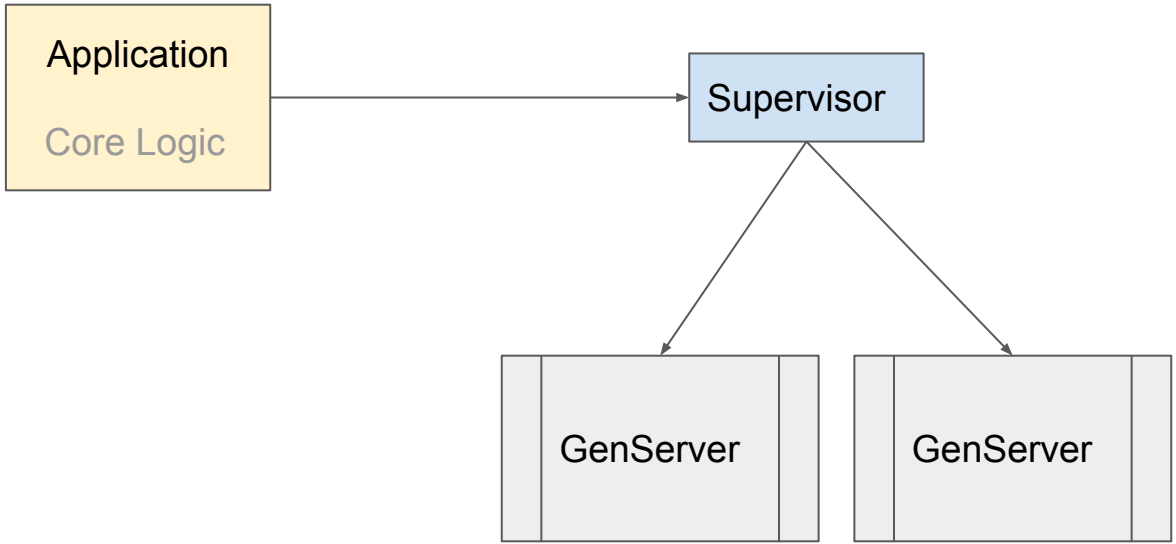
#CodeBEAMSF

BREX

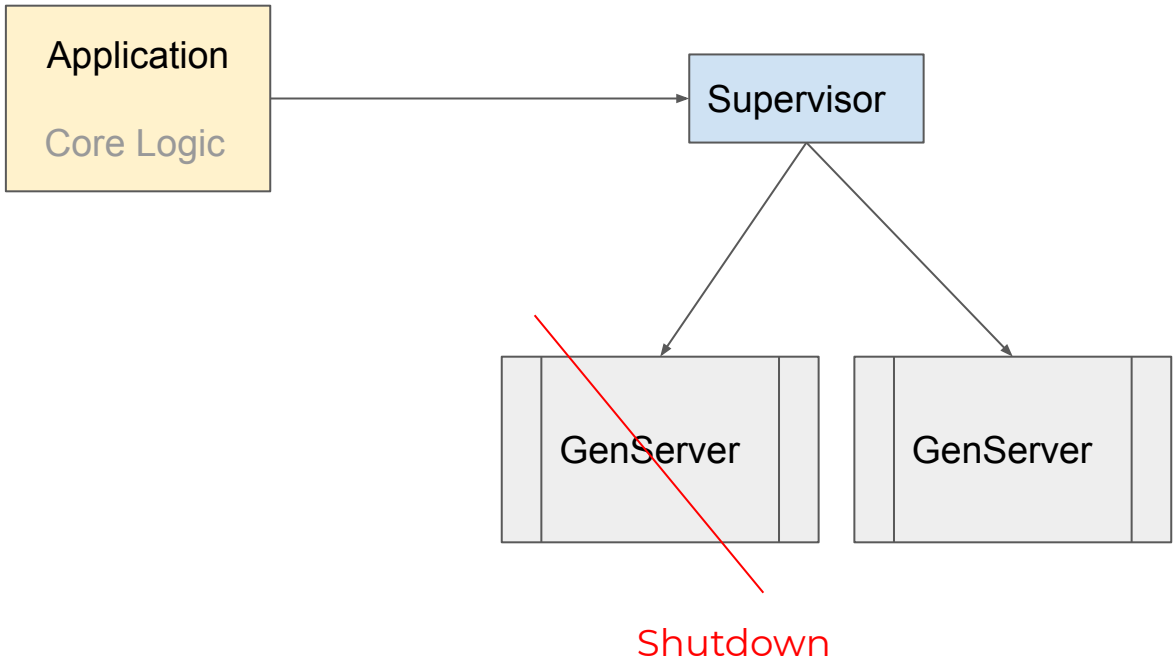
# Problem: Alert fatigue and false positives

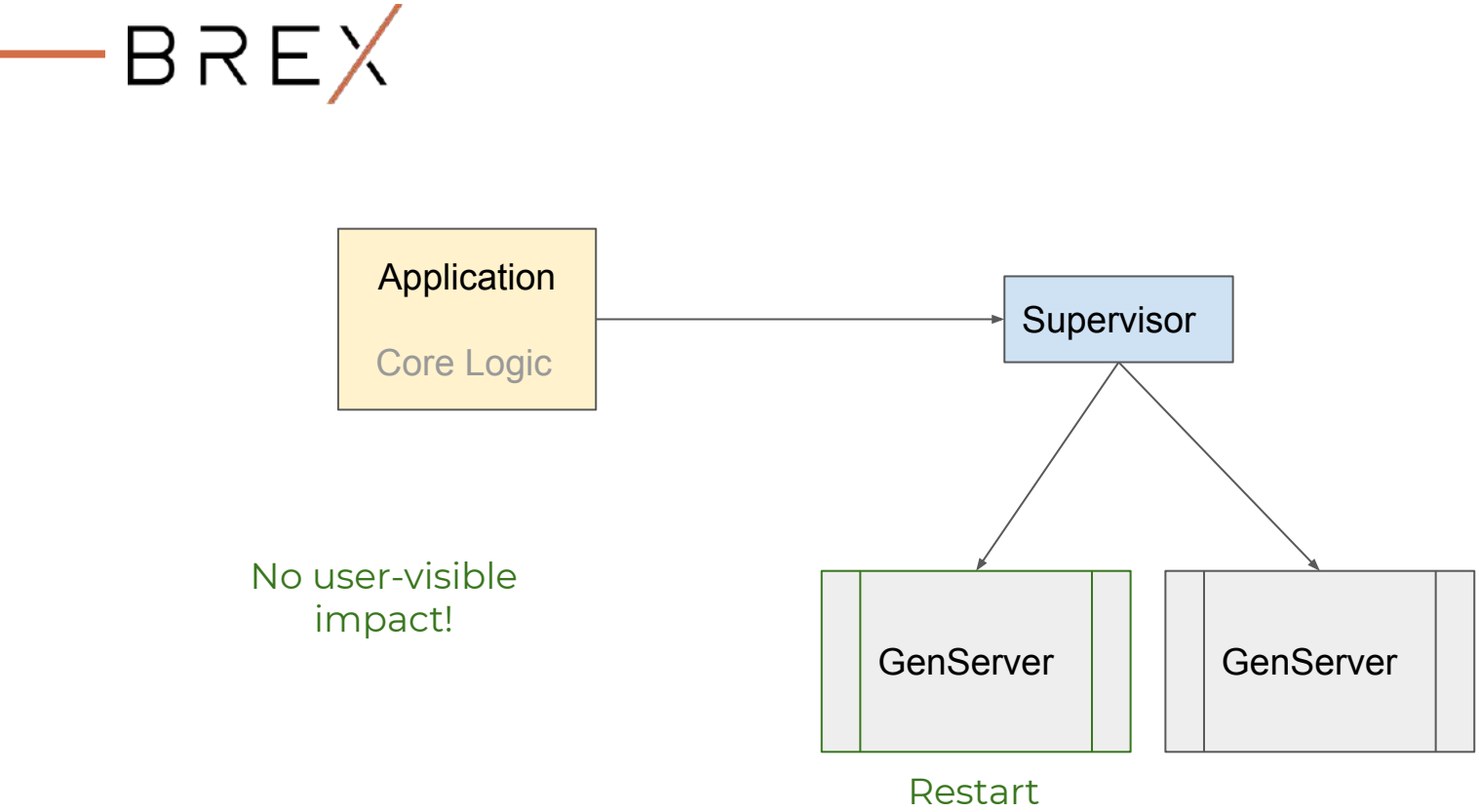
- Every single crash/exception was reported
- Conflicts with “*Let it crash*” Erlang philosophy
- Many exceptions would recover through the Supervision tree

BREX



BREX


Application
Core Logic
Supervisor
GenServer
GenServer
Shutdown


Application
Core Logic
Supervisor
No user-visible impact!
GenServer
GenServer
Restart

BREX

Problem: Alert fatigue and false positives

# Solution: Symptom Based Alerts

- *Symptom* vs *Cause*

BREX

Problem: Alert fatigue and false positives

# Solution: Symptom Based Alerts

- *Symptom* vs *Cause*
- How do we track *user-visible* impact?

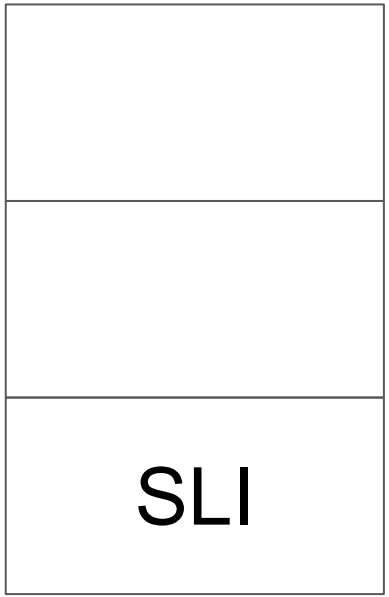
# SLI/SLO/SLAs

BREX

# SLIs & SLOs

BREX

# Introduction of SLIs/SLO/As



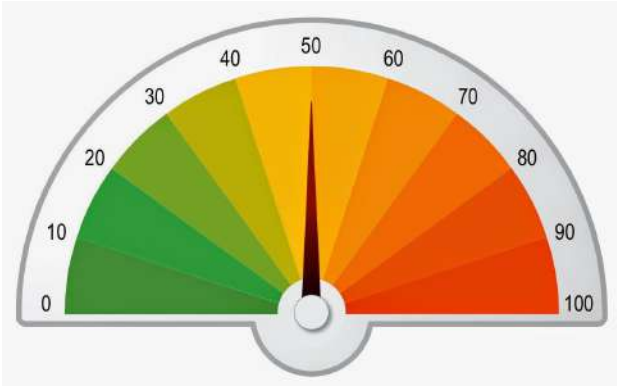

*"a carefully defined quantitative measure of some aspect of the level of service that is provided"*

- SRE Bible (Google SRE Handbook)

BREX

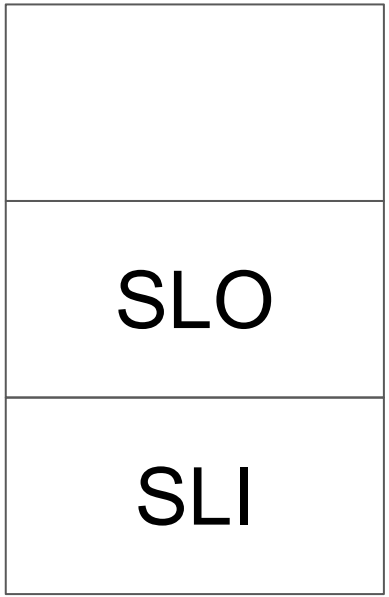
# Introduction of SLIs/SLO/As

SLI



BREX

# Introduction of SLIs/SLO/As

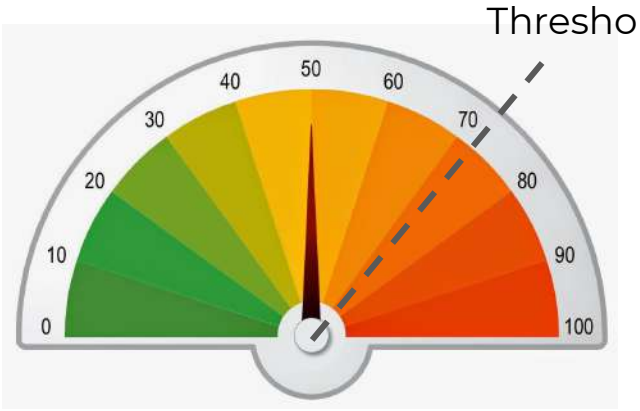



A reasonable % of your SLI

BREX

# Introduction of SLIs/SLO/As

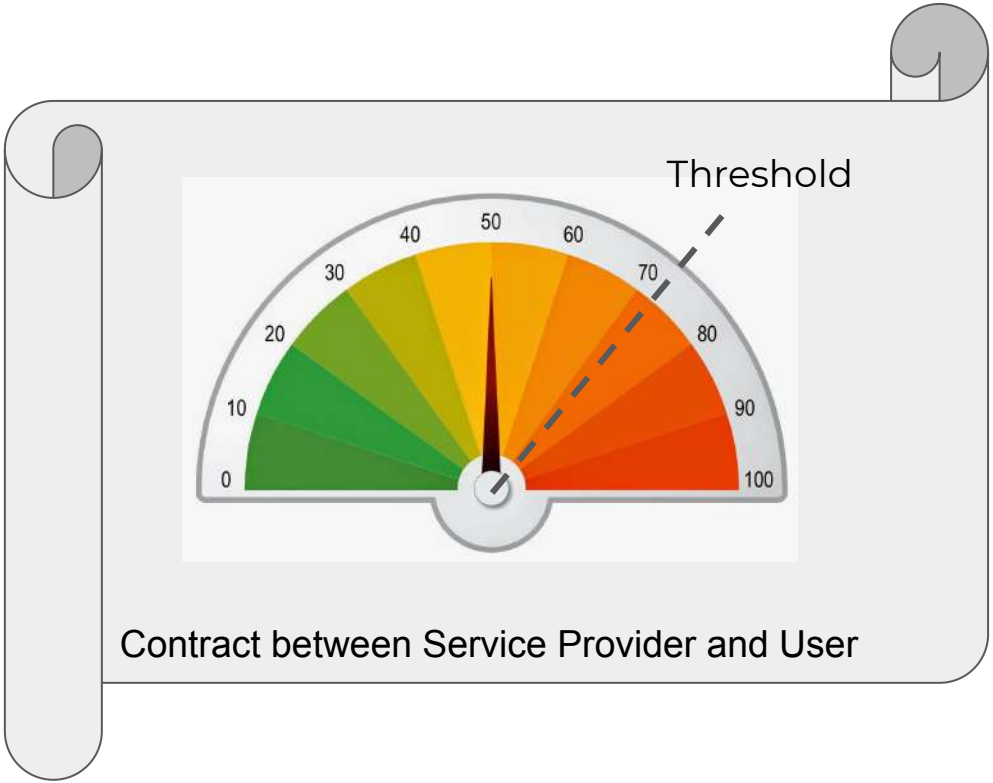
| |
|---|
| |
| SLO |
| SLI |



BREX

# Introduction of SLIs/SLO/As

| SLA |
| --- |
| SLO |
| SLI |



BREX

BREX

# Service Level Indicator

- Request Latency
- System Throughput
- Error Rate
- Availability

BREX

# Service Level Indicator

- Request Latency
- System Throughput
- Error Rate
- Availability

BREX

# Service Level Indicator

What does *Availability* mean for a service?

BREX

# Service Level Indicator: Availability

Service's readiness endpoint `/ready` returns 200

BREX

# Service Level Indicator: Availability

- *Service is up*, but can't connect to downstream services

BREX

# Service Level Indicator: Availability

- *Service is up*, but can't connect to downstream services
- *Service is up*, but is unreachable from external clients
- ...

BREX

This indicator is this dog:

BREX

# Service Level Indicator: Availability

What is a good indicator?

BREX

# Service Level Indicator

- A good SLI measures **User Journey**
  - A user can be a person, another service, a robot...

BREX

# SLI Use case 1: Card Swipe

We have 2s to reply to an authorization request performed by the Processor

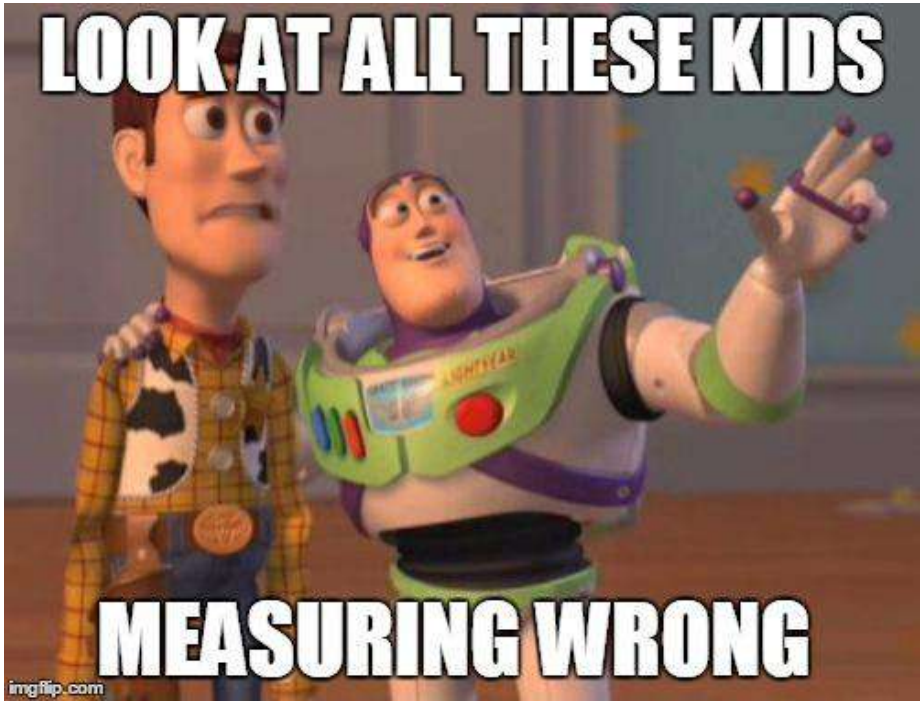
BREX

# SLI Use case 1: Card Swipe

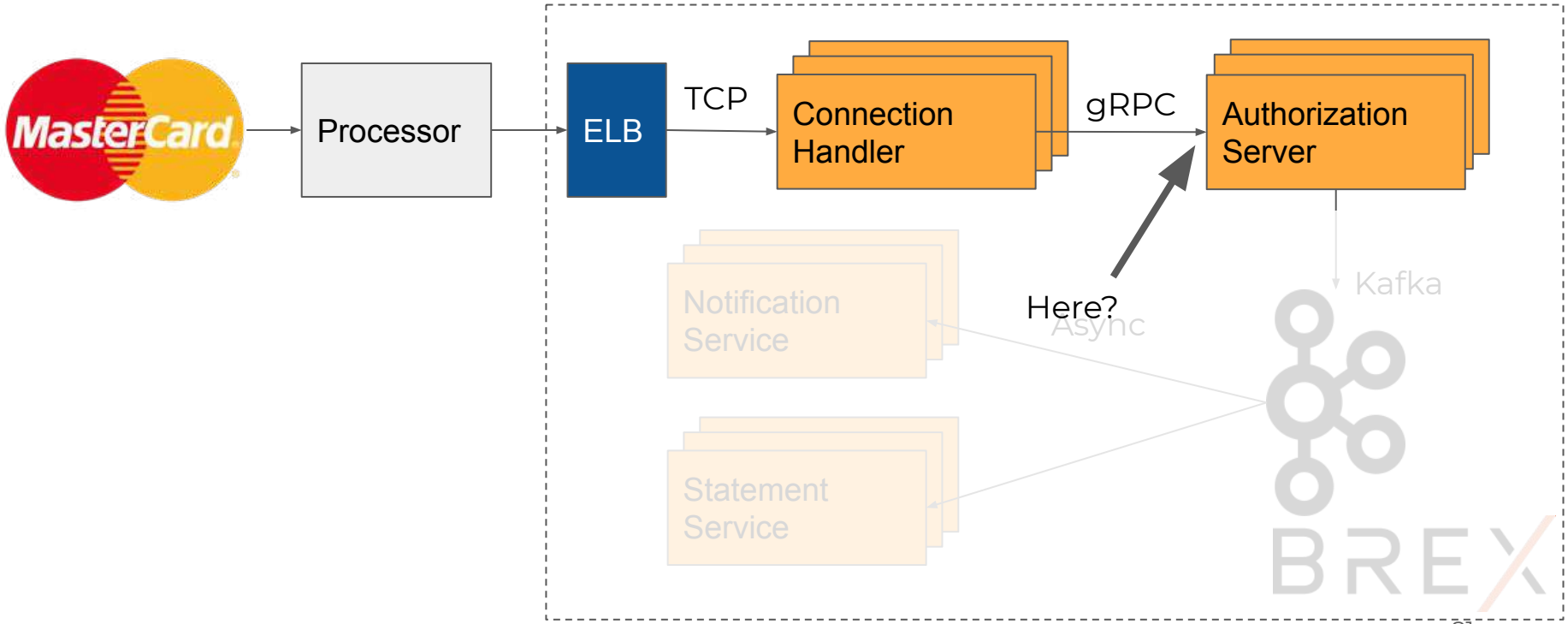
Good SLI: *Requests to our service **/auth** endpoint returns a **valid response** (approved/declined) **within 2s***

BREX

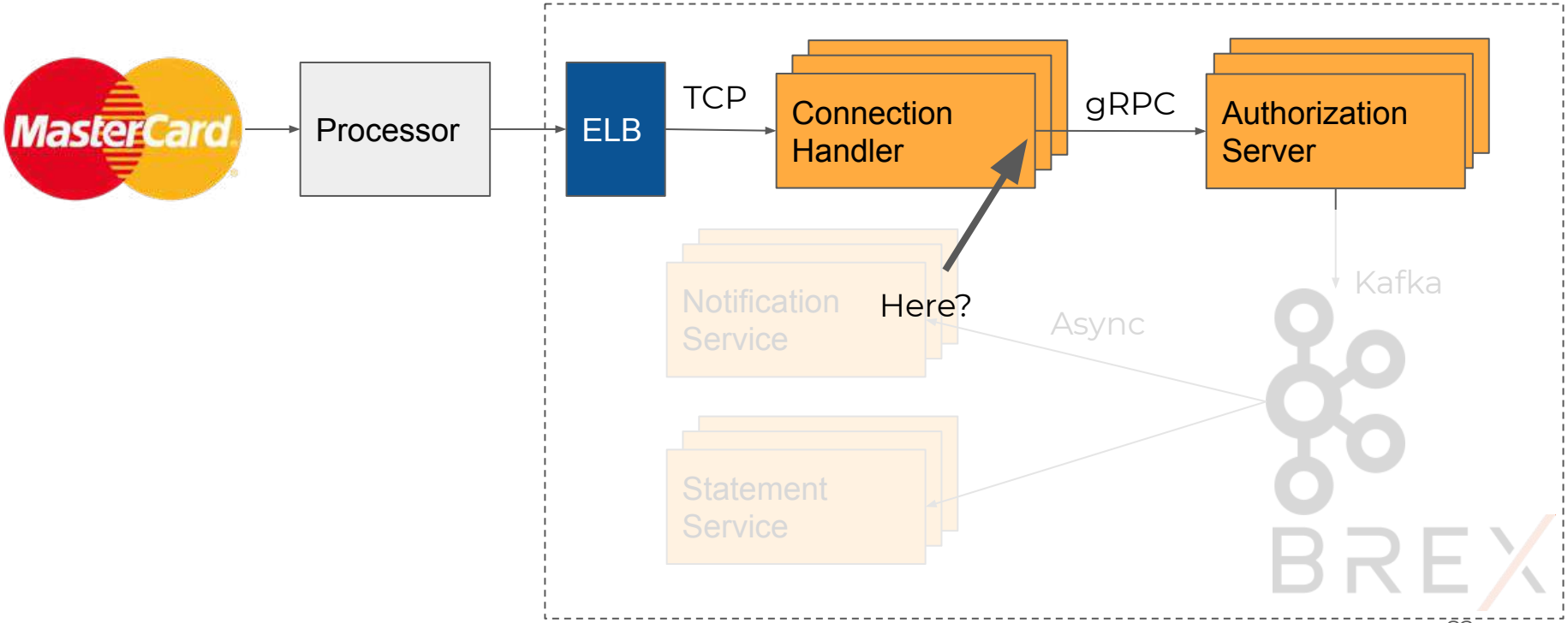
# How to measure it?



SLI: *Requests to our service `/auth` endpoint returns a valid response (approved/declined) within 2s*

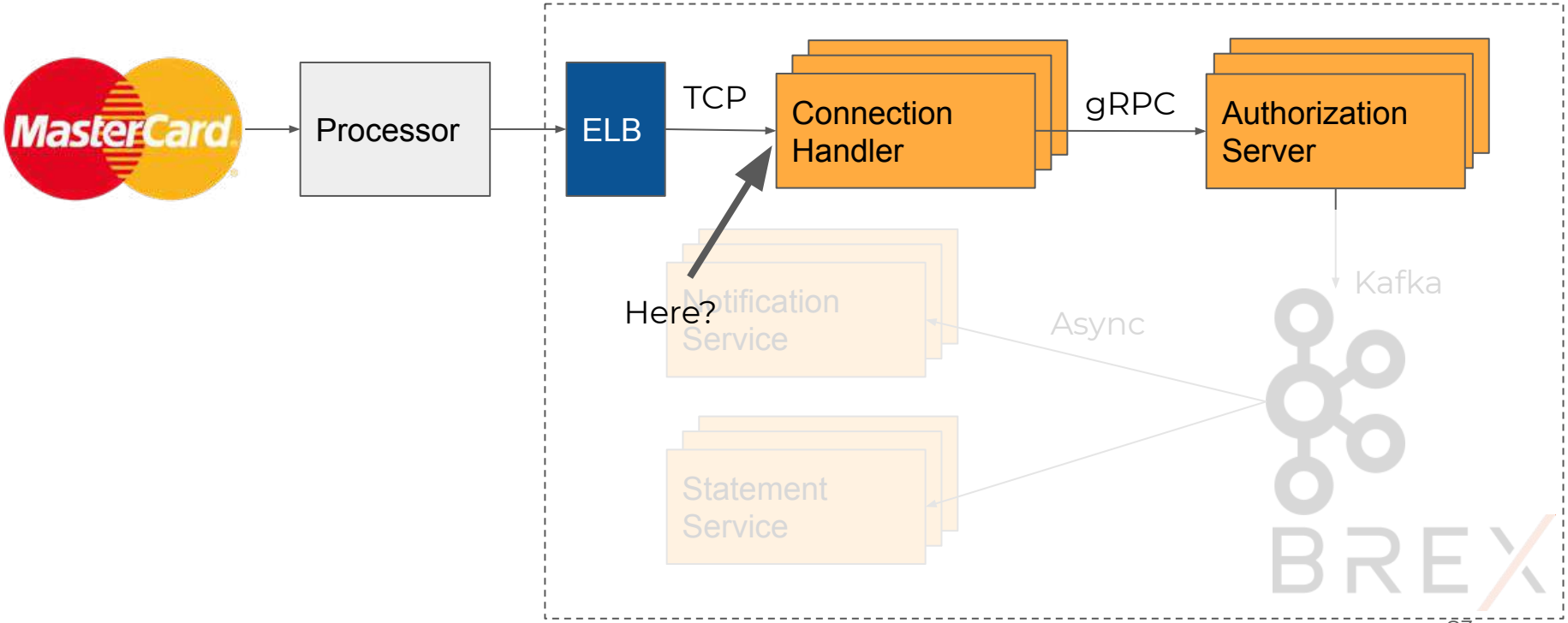


SLI: *Requests to our service `/auth` endpoint returns a valid response (approved/declined) within 2s*

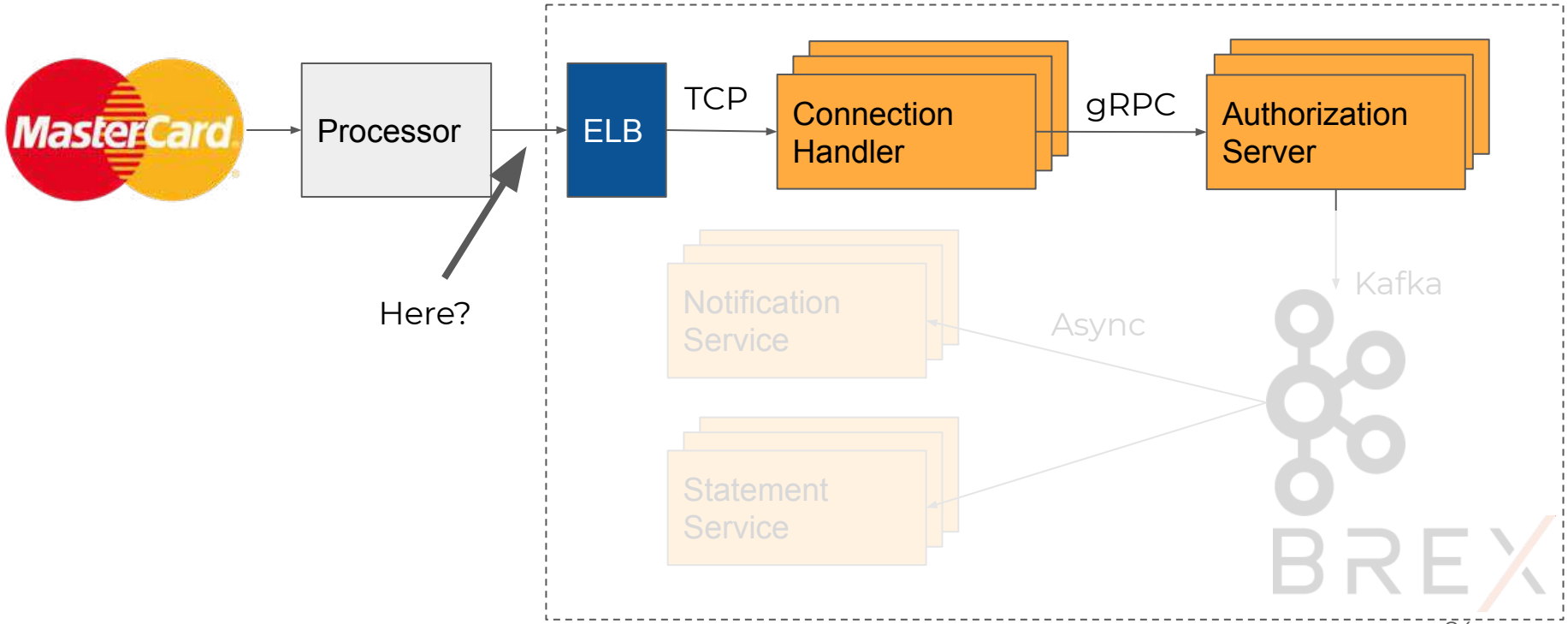


BREX

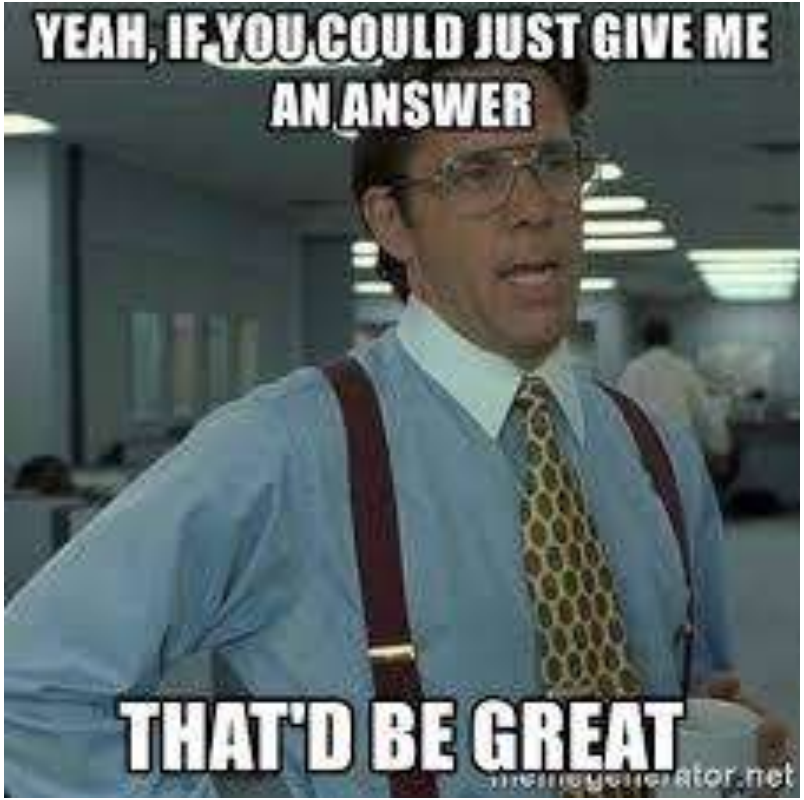
SLI: *Requests to our service `/auth` endpoint returns a valid response (approved/declined) within 2s*



SLI: *Requests to our service `/auth` endpoint returns a valid response (approved/declined) within 2s*



BREX



BREX

# Service Level Indicator

All options are valid and can be measured

BREX

# Service Level Indicator

Sometimes, only a proxy is available

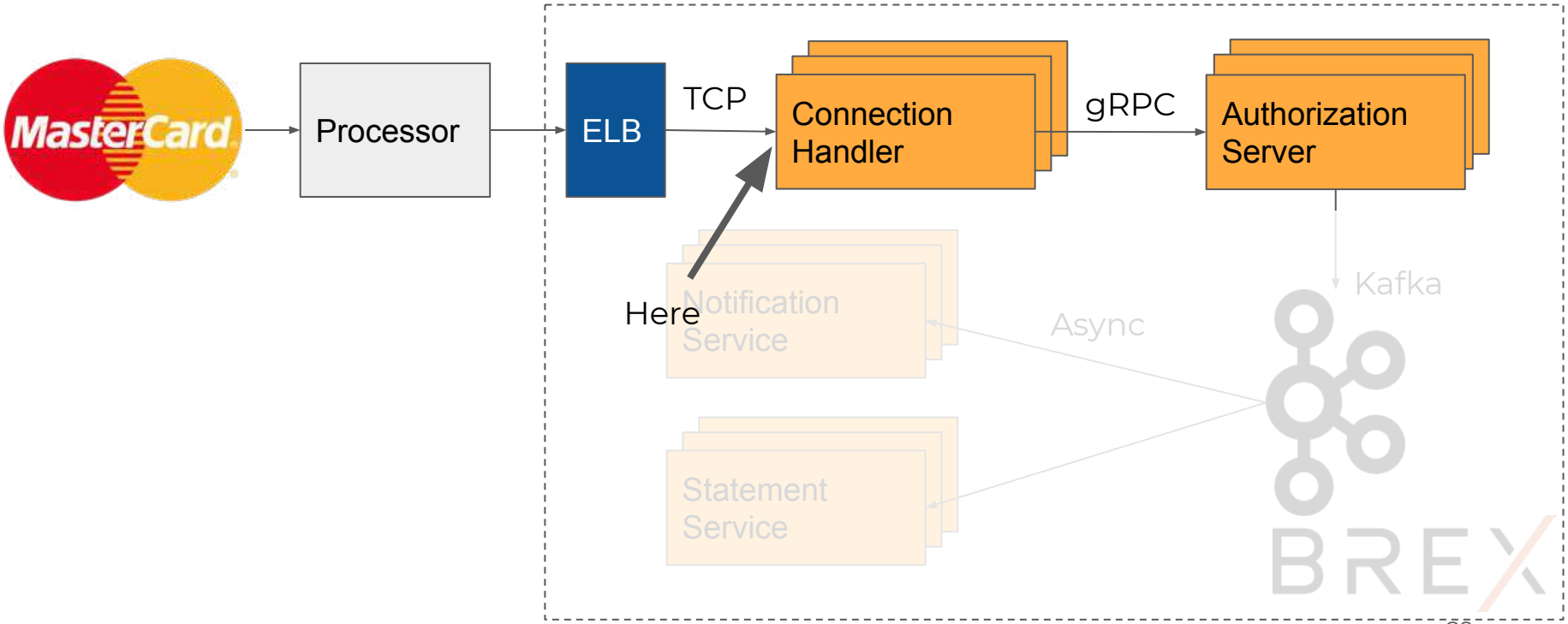
BREX

# Service Level Indicator

The best is the closest to the user experience

BREX

# This one



BREX

# Service Level Indicator

It is **hard** to **define** and **measure**

BREX

# Service Level Indicator - Use Case 2

BREX

# SLI Use Case 2: SMS notification to users

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# SLI Use Case 2: SMS notification to users

SLI: *User receives SMS within 5 minutes of an authorization request approved/declined* ***?***

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# SLI Use Case 2: SMS notification to users

SLI: *User receives SMS within 5 minutes of an authorization request approved/declined* ***?***

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# SLI Use Case 2: SMS notification to users

Practical SLI: *Notification Service successfully performs a request to <Vendor> to send a SMS within 5 minutes of an authorization request approved/declined*

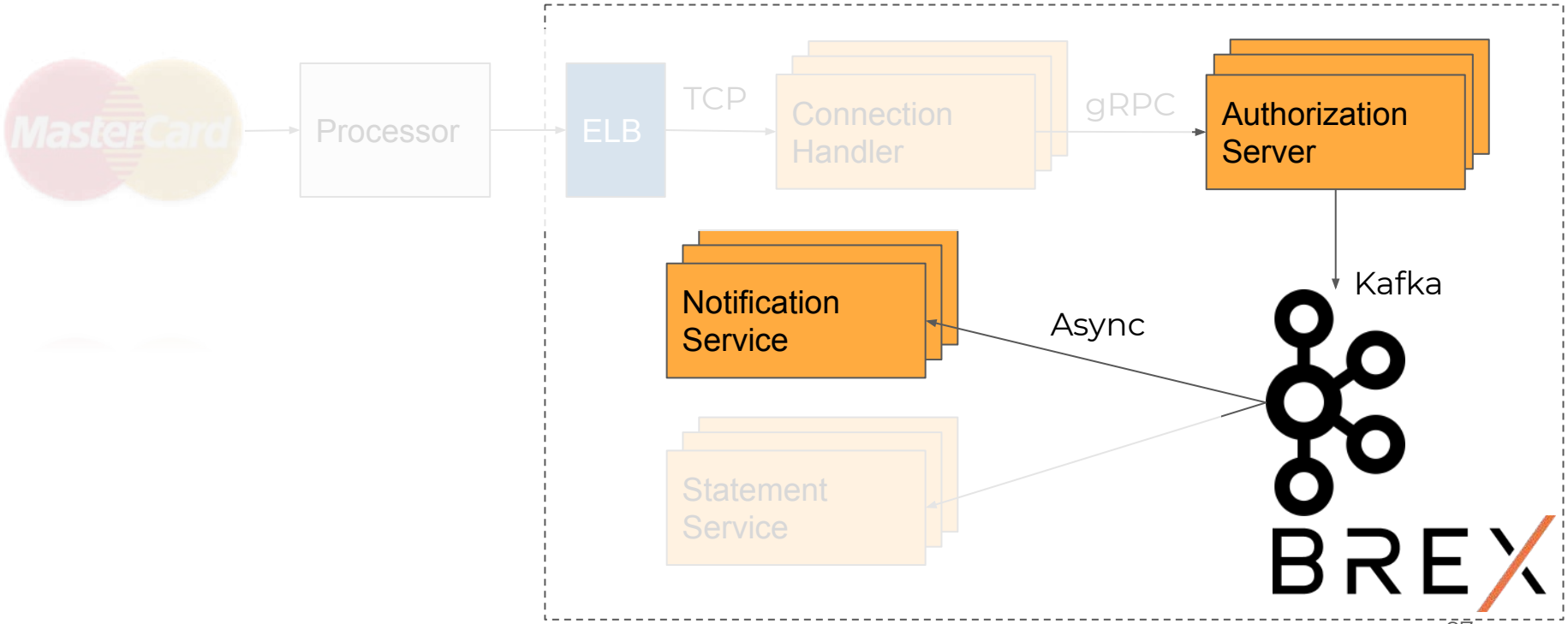
BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# How to measure?

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

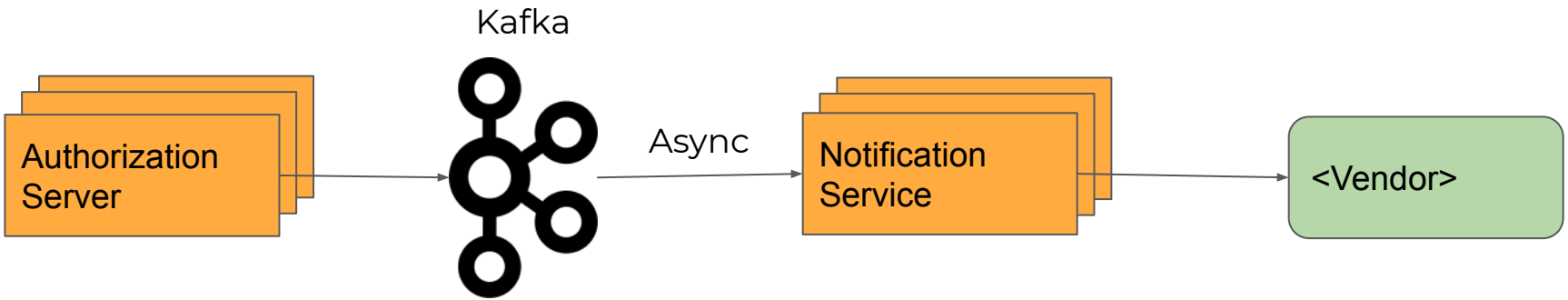
# How to measure?



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

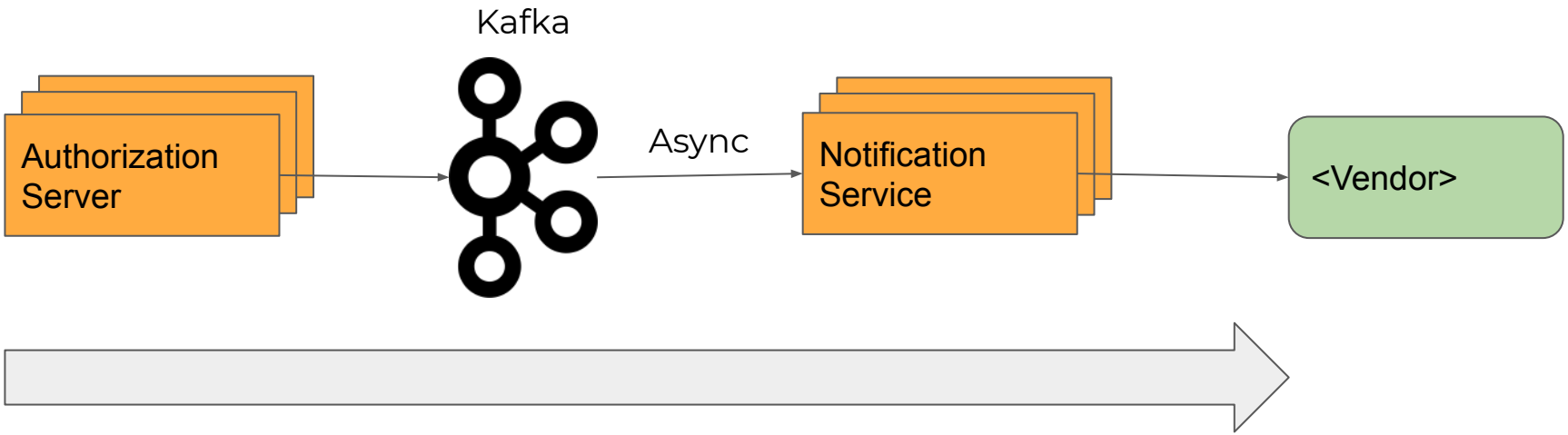
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

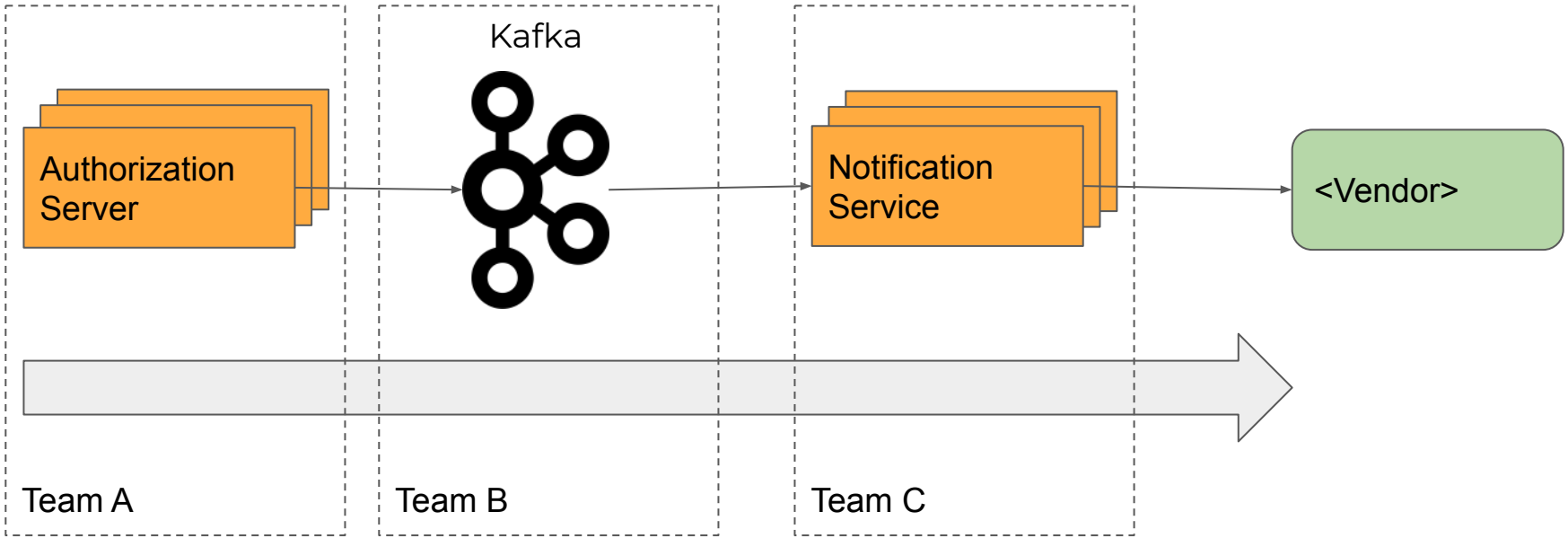
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

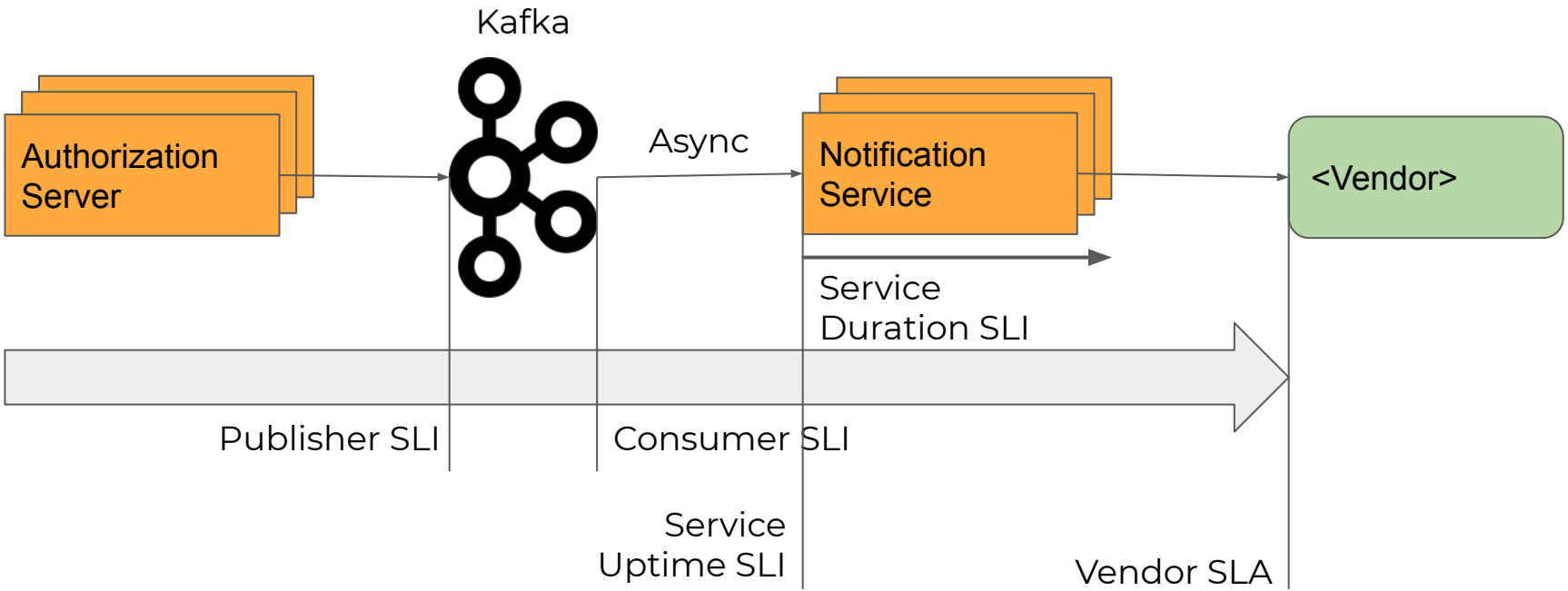
# Measuring higher-level SLIs

## Approach 1

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

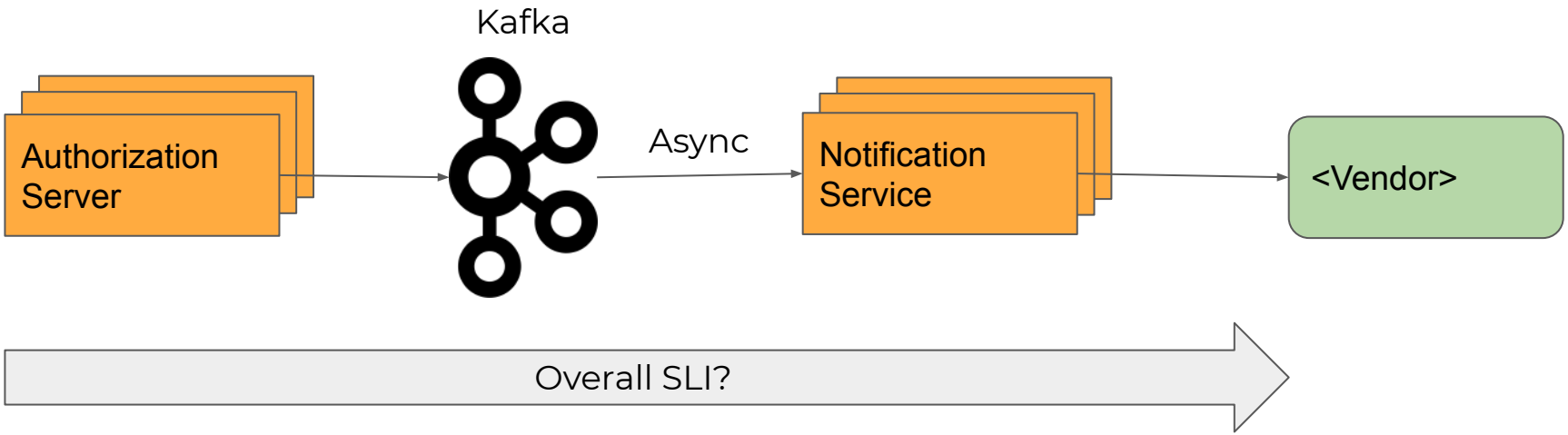
# Measuring higher-level SLIs

Decouples teams!

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

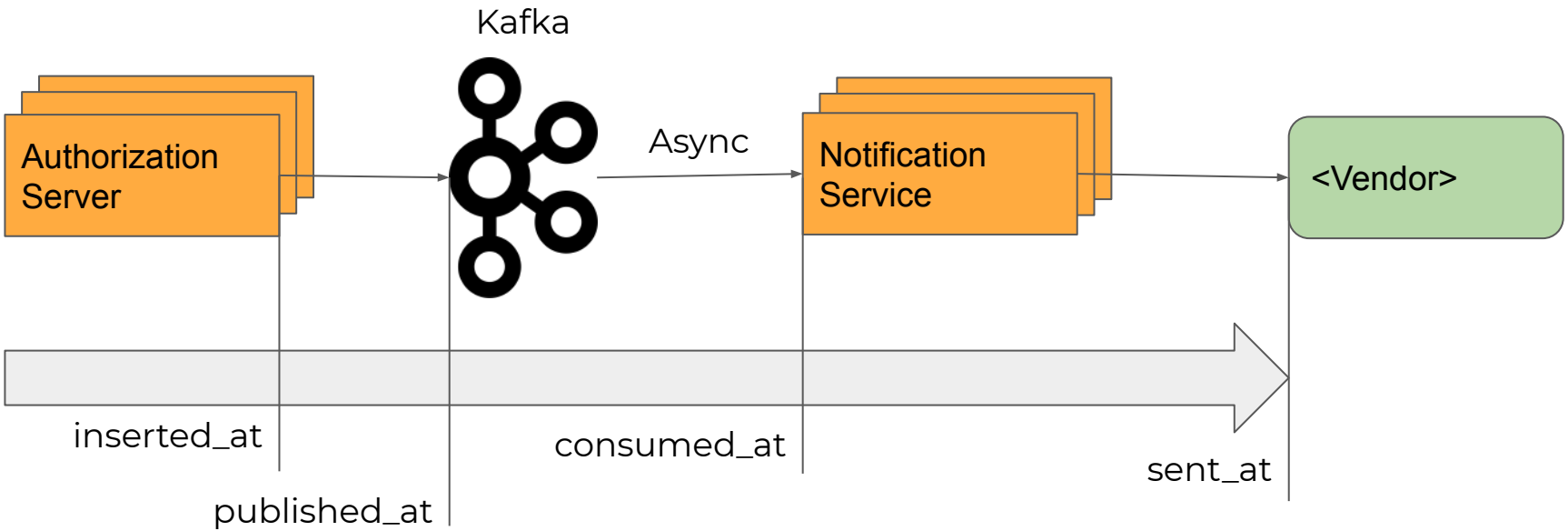
# Measuring higher-level SLIs

## Another Approach

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

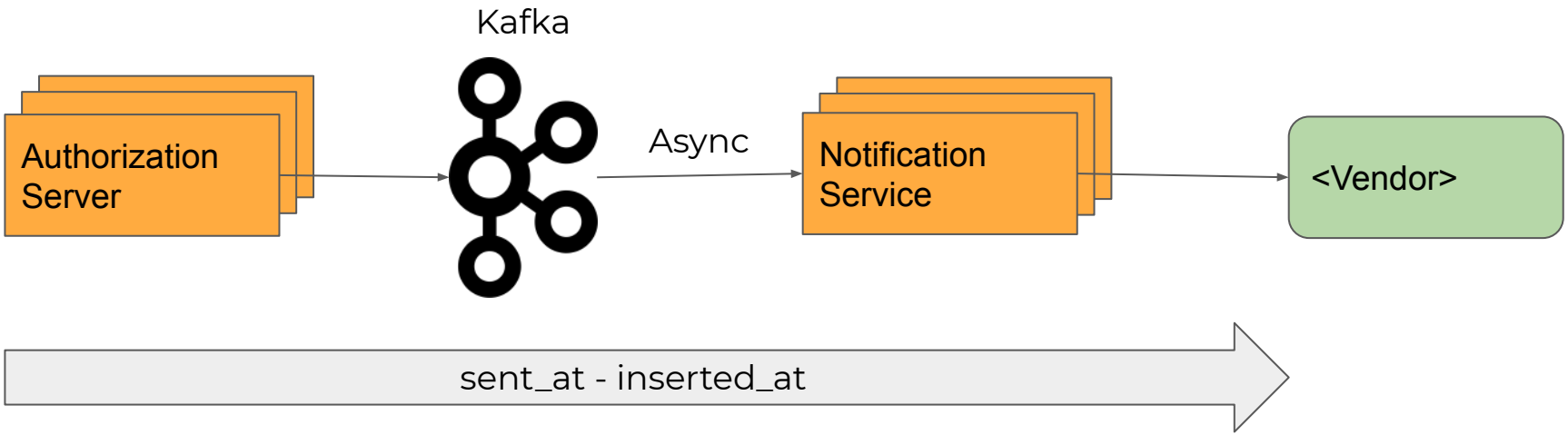
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

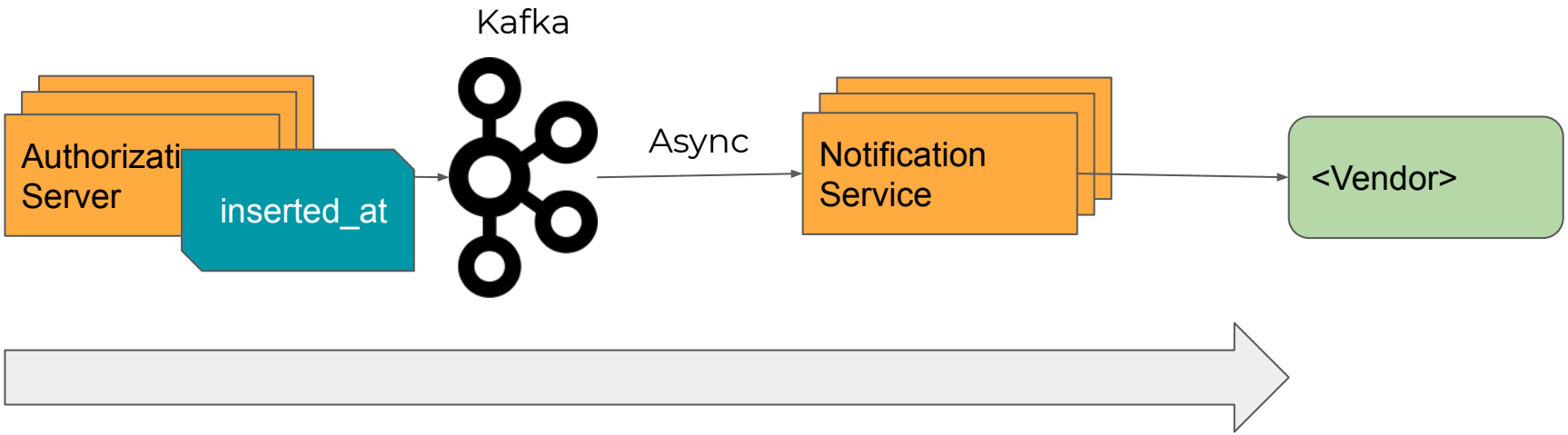
# Measuring higher-level SLIs

How to do this?

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

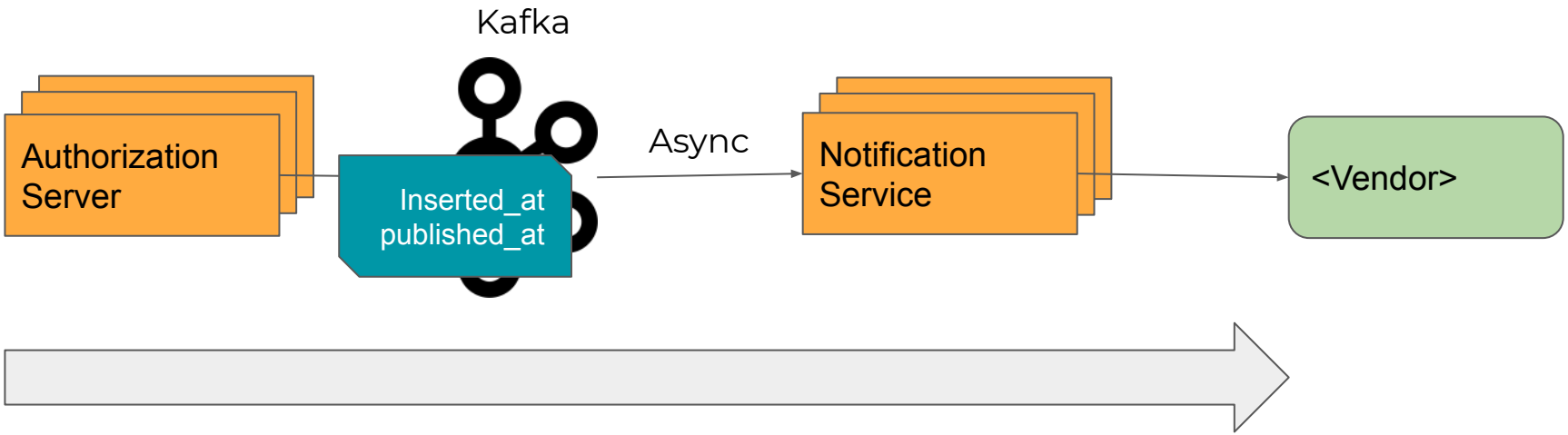
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

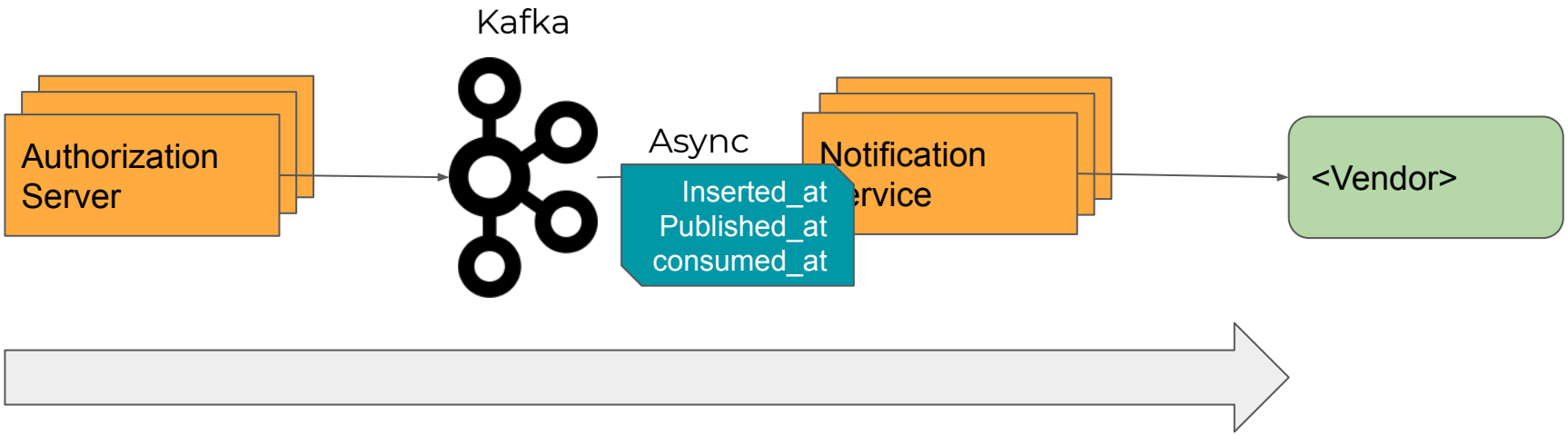
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

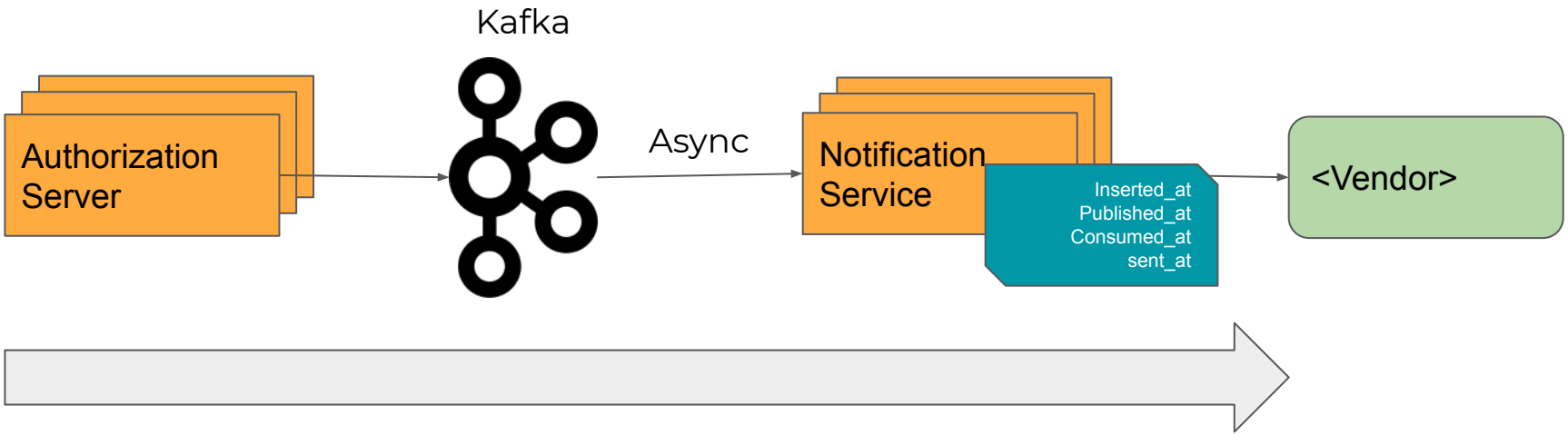
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

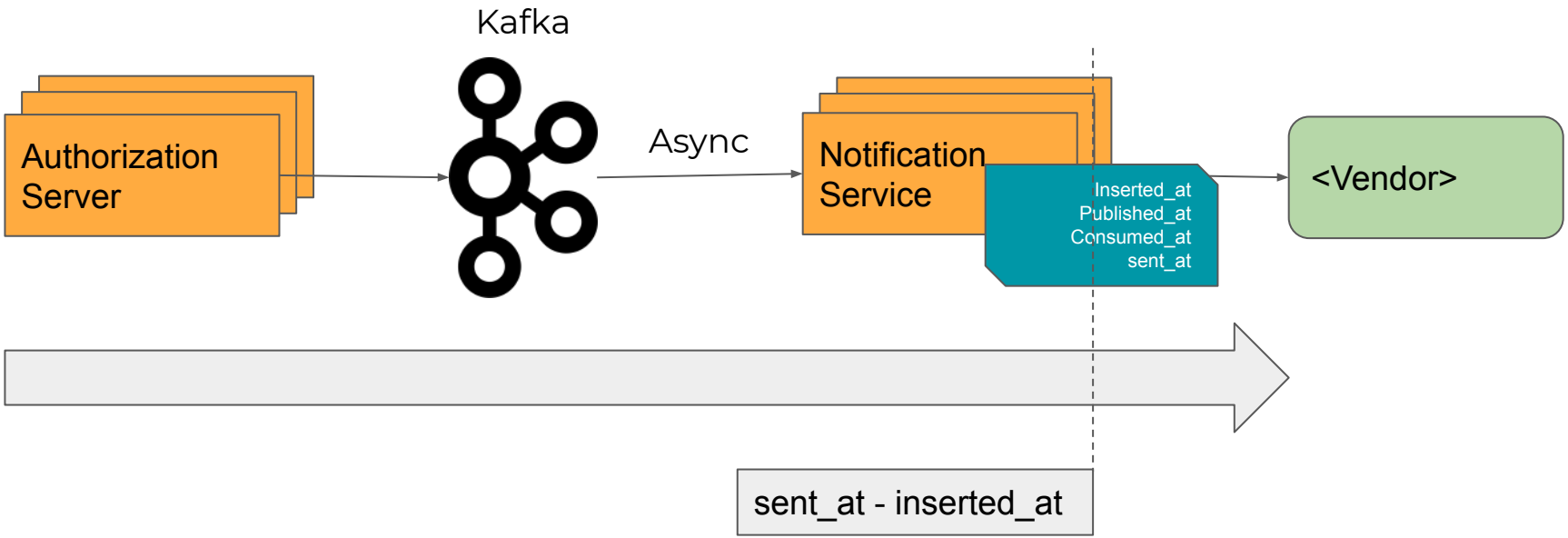
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

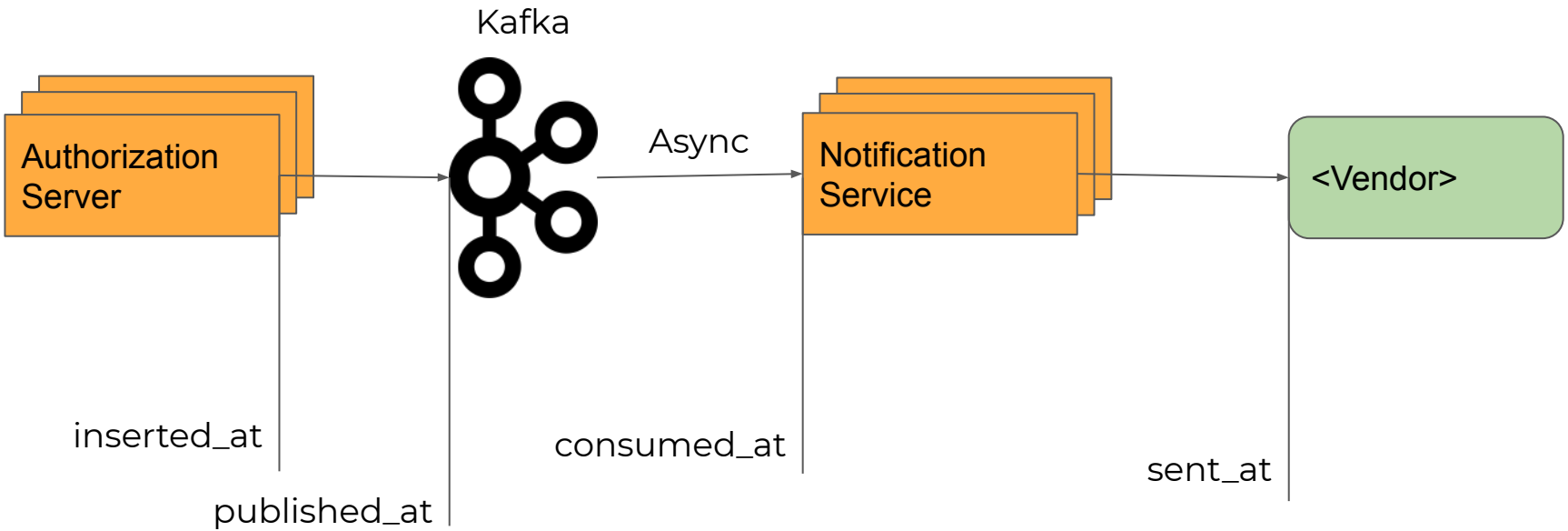
# Measuring higher-level SLIs

Another way to do this?

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

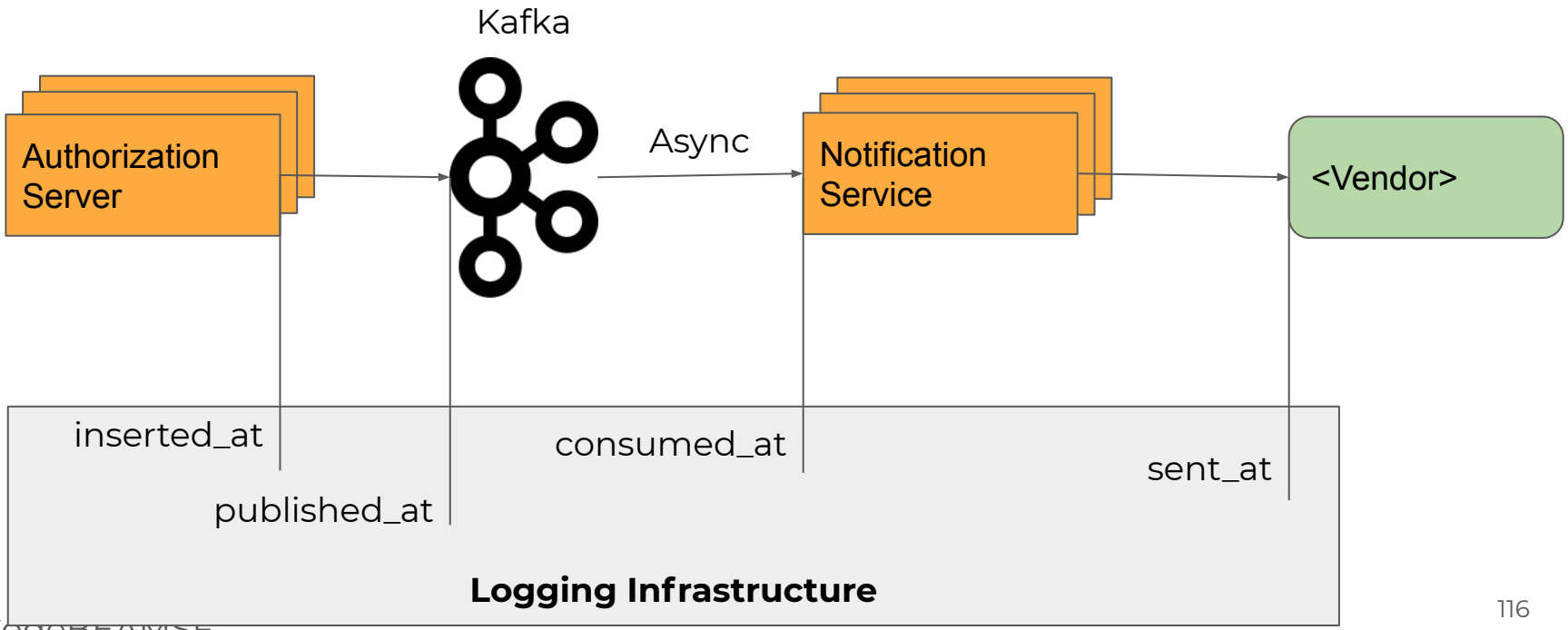
# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs



BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs

**Option 1:** Decompose as lower level SLIs

**Pros:**

- Easier
- Decouples Teams

**Cons:**

- Difficult to estimate overall SLI

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

## Measuring higher-level SLIs

**Option 2:** Package relevant data throughout services

**Pros:**

- Capture the overall SLI

**Cons:**

- Not extensible, need business logic to handle the data

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# Measuring higher-level SLIs

**Option 3:** Rely on a logging/event infrastructure

**Pros:**

- Capture the overall SLI
- Extensible

**Cons:**

- Require such a feature in the logging infra

BREX

For a good user experience, users must receive their transaction SMS at most 5 minutes after the transaction took place

# We did it!

# Lessons Learned

BREX

# Observability is so cool!

BREX

# What has our Observability team at Brex learned so far?

BREX stripe

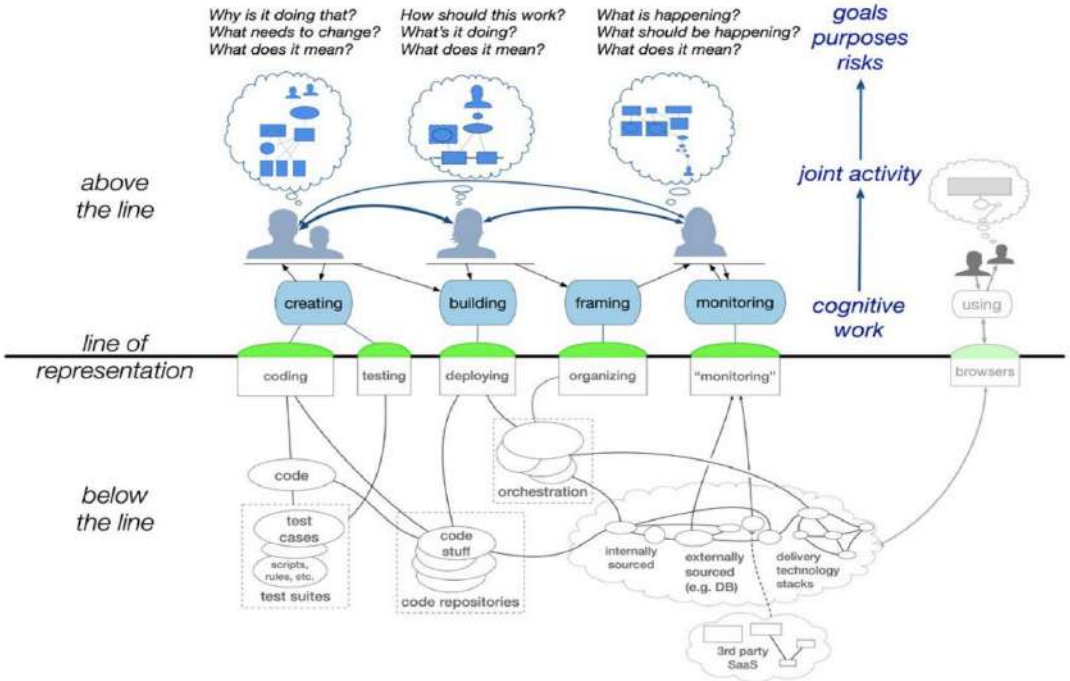
There is no “one-size-fits-all” solution



Observability is not only for high throughput systems

BREX


The earlier Observability is incorporated into the Engineering culture the better



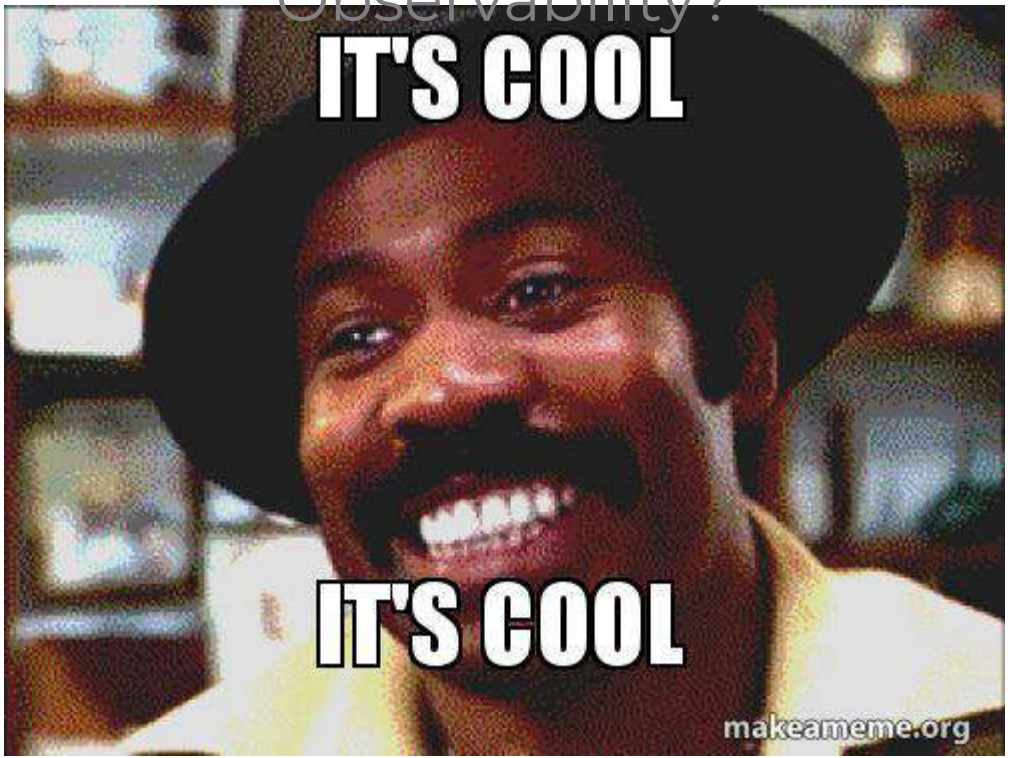

https://queue.acm.org/detail.cfm?id=3380777

# Creates a shared mental model

BREX

BREX

Observability?



# Thank You!

BREX

# References


- Alex Hidalgo - Developing Meaningful SLIs for Fun and Profit (https://vimeo.com/369636345)
- Cory Watson - Dashboard Renaissance: How dashboards work and how to improve them (https://vimeo.com/369638117)
- gRPC - https://grpc.io/
- Honeycomb Intro to Observability - https://docs.honeycomb.io/learning-about-observability/intro-to-observability/
- Jason - https://hexdocs.pm/jason/Jason.html
- Kafka - https://kafka.apache.org/
- OpenTelemetry - https://opentelemetry.io/
- Prometheus - https://prometheus.io/
- Sentry - https://sentry.io
- Spandex - https://github.com/spandex-project/spandex
- Statsd - https://github.com/statsd/statsd
- AWS EKS Image - https://vitalflux.com/wp-content/uploads/2017/11/Deploy-first-cloud-native-apps-on-kubernetes.png
- CloudNative - https://stackify.com/cloud-native/