# SRE

17-313 Spring 2024

Foundations of Software Engineering

https://cmu-313.github.io

**Michael Hilton** and Eduardo Feo Flushing

# Administrivia

- Midterm 2 Thursday April 18th
- Final Exam attendance Mandatory:
  - Monday, April 29, 2024 05:30pm-08:30pm
  - If you will be celebrating Passover, let us know ASAP to support alternatives.
  - Conflicts come talk to us as well
- TA's needed for fall, let me know if you are interested.

# The Art of SLOs

*In the midst of **chaos**, there is also ~~opportunity~~ **reliability***

*— Sun Tzu,* The Art of War

# Agenda

- Terminology
- Why your services *need* SLOs
- Spending your error budget
- Choosing a good SLI
- Developing SLOs and SLIs

# **S**ervice **L**evel **I**ndicator

A **quantifiable** measure of service **reliability**

# **S**ervice **L**evel **O**bjectives

Set a **reliability target** for an SLI

# Users? Customers?

**Customers** are users who **directly pay** for a service

# Services *Need* SLOs

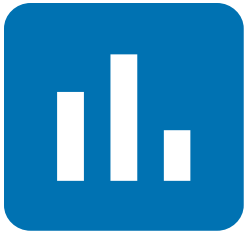
# Don't believe us?

"Since introducing SLOs, the **relationship** between our operations and development teams has **subtly but markedly improved**."

— *Ben McCormack, Evernote;* The Site Reliability Workbook, Chapter 3

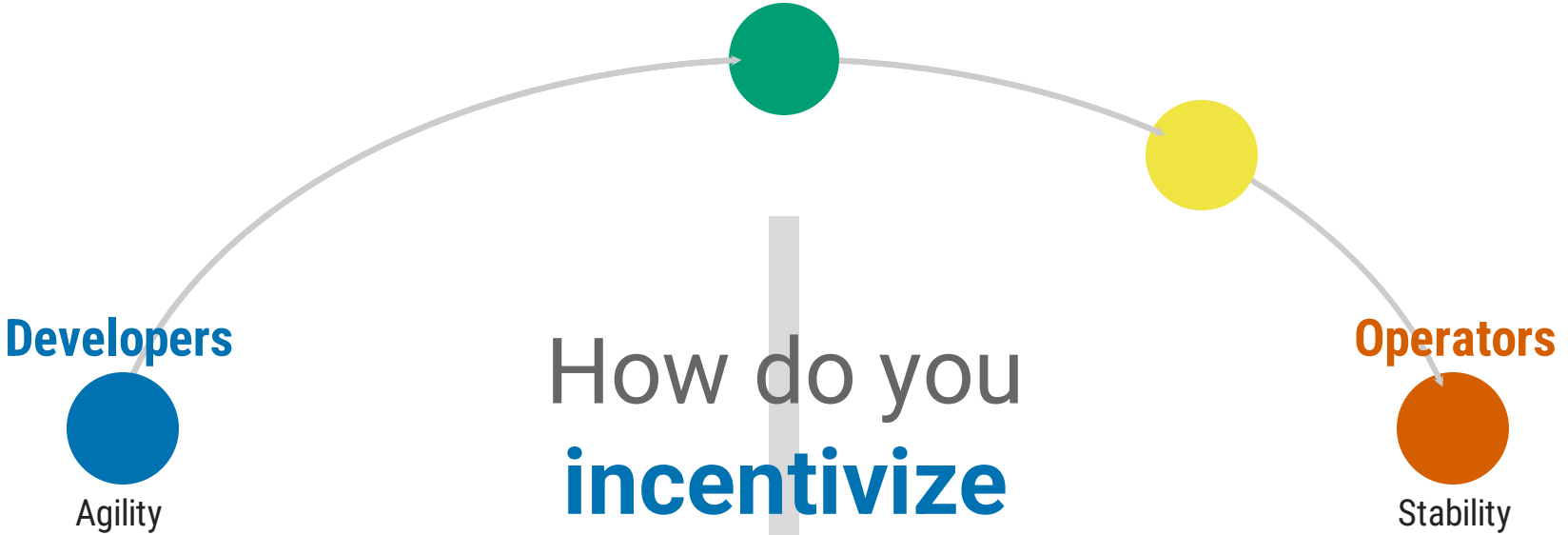
"... it is difficult to *do your job well* without clearly defining *well*. SLOs **provide the language** we need to **define *well***."

— *Theo Schlossnagle, Circonus;* Seeking SRE, Chapter 21

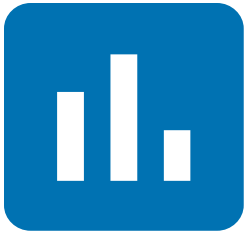


The **most important feature** of any system is its **reliability**

1



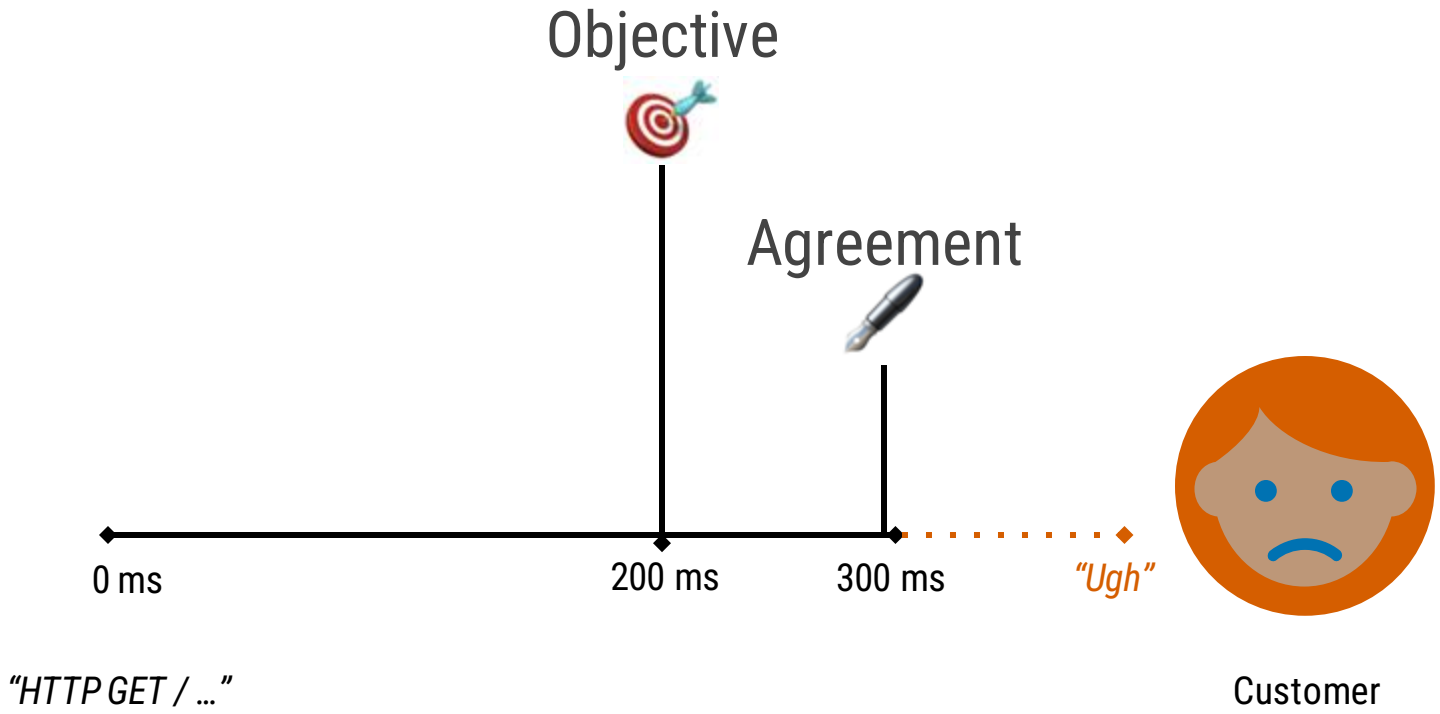

How do you **incentivize** reliability?



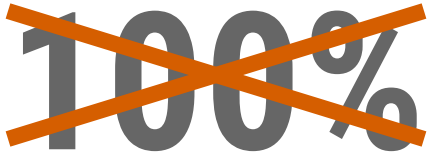
A **principled** way
to agree on the
**desired reliability**
of a service

# What does "**reliable**" mean?

Think about Netflix, Google Search, Gmail, Twitter…
how do you tell if they are ‘working’?


Objective
Agreement
0 ms
200 ms
300 ms
"Ugh"
"HTTP GET / ..."
Customer

When do we need to make a service **more reliable**?



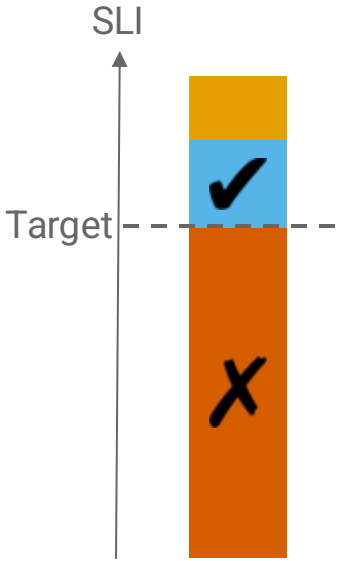

100% is the **wrong** reliability target for basically **everything**

— ***Benjamin Treynor Sloss**, VP 24x7, Google;* Site Reliability Engineering, Introduction

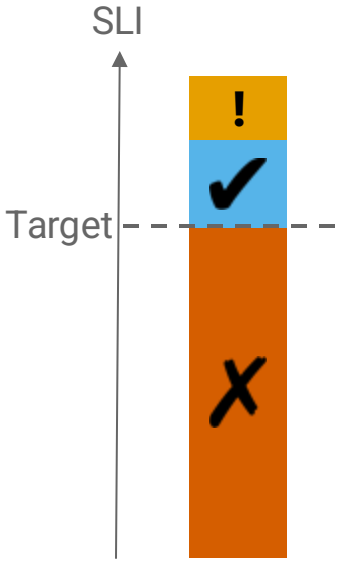
SLOs should capture the performance and availability levels that, if **barely met**, would keep the **typical customer** of a service happy

**“meets SLO targets” ⇒ “happy customers”**
**“sad customers” ⇒ “misses SLO targets”**

Measure SLO achieved & try to be ***slightly*** over target...




"Workflow", Randall Munroe, XKCD
Source: https://xkcd.com/1172/




…but don't be too much better or users will **depend on it**

# Error Budgets

An SLO implies an **acceptable level** of unreliability

*This is a **budget** that can be **allocated***

# Implementation Mechanics

Evaluate SLO **performance** over a set **window**, e.g. 28 days

Remaining budget **drives prioritization** of engineering effort

What should we **spend**
our error budget on?

# Error budgets can accommodate

- releasing new **features**
- expected system **changes**
- inevitable **failure** in hardware, networks, etc.
- planned **downtime**
- risky **experiments**

# Benefits of error budgets

- **Common incentive for devs and SREs**
  Find the right balance between innovation and reliability
- **Dev team can manage the risk themselves**
  They decide how to spend their error budget
- **Unrealistic reliability goals become unattractive**
  These goals dampen the velocity of innovation

- **Dev team becomes self-policing**
  The error budget is a valuable resource for them
- **Shared responsibility for system uptime**
  Infrastructure failures eat into the error budget

# Activity

Reliability Principles

Dear Colleagues,

The negative press from our recent outage has convinced me that we *all* need to take the reliability of our services more seriously. In this open letter, I want to lay down three reliability principles to guide your future decision making.

The first principle concerns our users. We let them down, but they deserve better. They deserve to be *happy* when using our services!

Our business must ...

1. ... rebuild user trust by making a financial commitment to reliability.
2. ... find ways to help our users tolerate or enjoy future outages.
3. ... meet our users expectations of reliability before building features.
4. ... build the features that make our users happy faster.
5. ... never suffer another outage, ever again!

The second principle concerns the way we build our services. We have to change our development process to incorporate reliability.

Our business must...

1. … choose to fail fast and catch errors early through rapid iteration.
2. … have Ops engage in the design of new features to reduce risk.
3. … only release new features publicly when they are shown to be reliable.
4. … build and release software in small, controlled steps.
5. … reduce feature iteration speed when our systems are unreliable.

The third principle concerns our operational practices. What we're doing today isn't working. Our Ops teams are burned out and our incident rate is too high. We have to do things differently to improve!

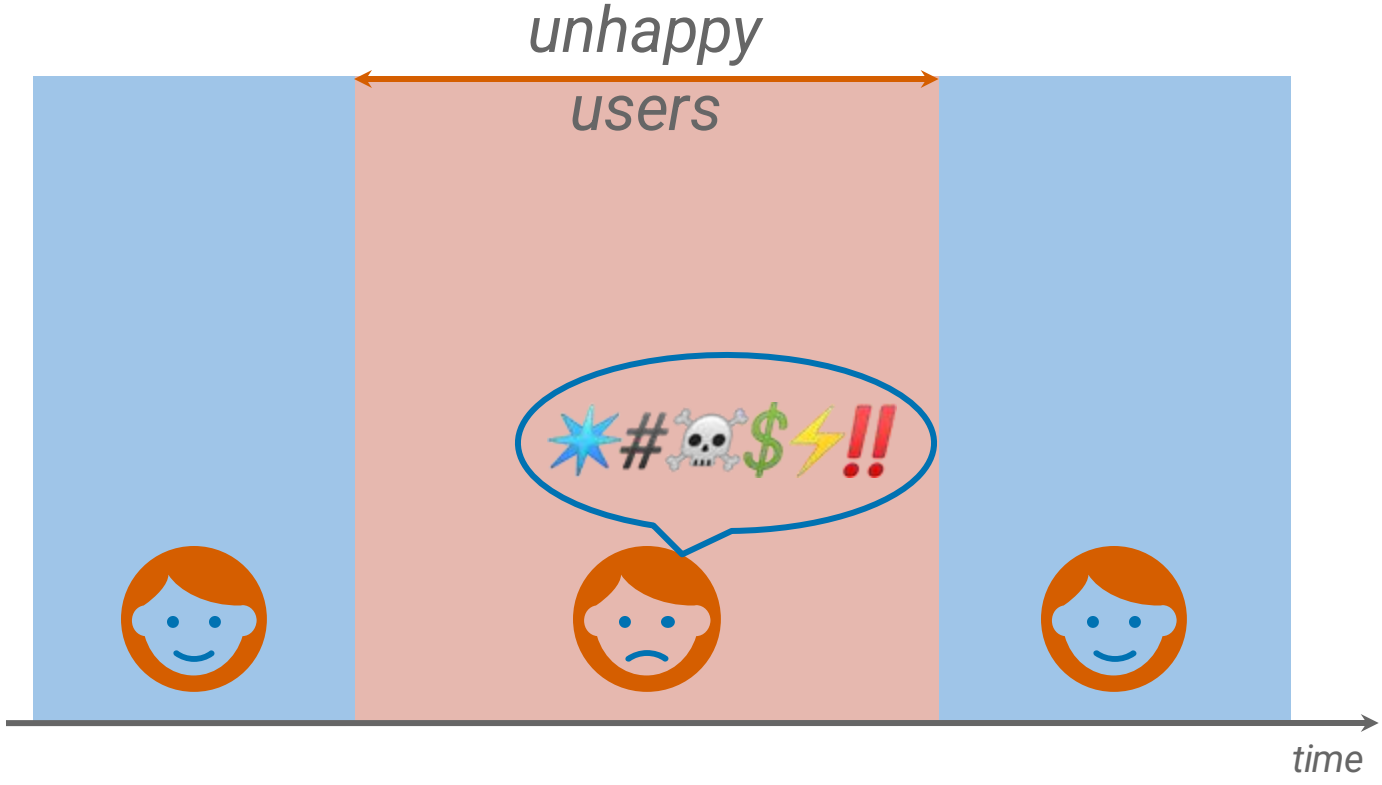
Our business must...

1. … share responsibility for reliability between Ops and Dev teams.
2. … tie operational response and team priorities to a reliability goal.
3. … make our systems more resilient to failure to cut operational load.
4. … give Ops a veto on all releases to prevent failures reaching our users.
5. … route negative complaints on Twitter directly to Ops pagers.

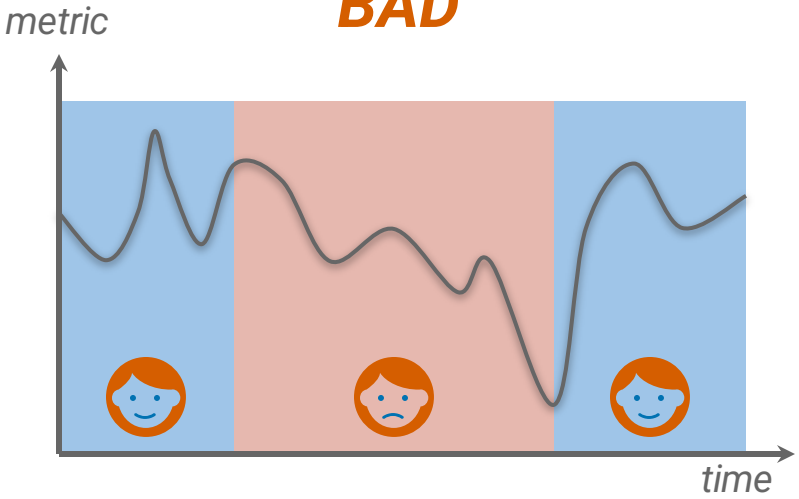
To put these principles into practice, we are going to borrow some ideas from Google! The next step is to define some SLOs for our services and begin tracking our performance against them.

Thanks for reading!
*Eleanor Exec*, CEO

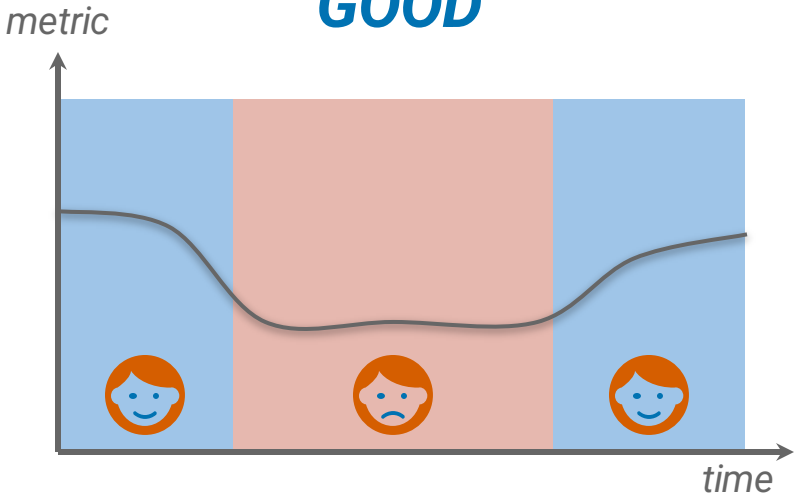
# Choosing a Good SLI

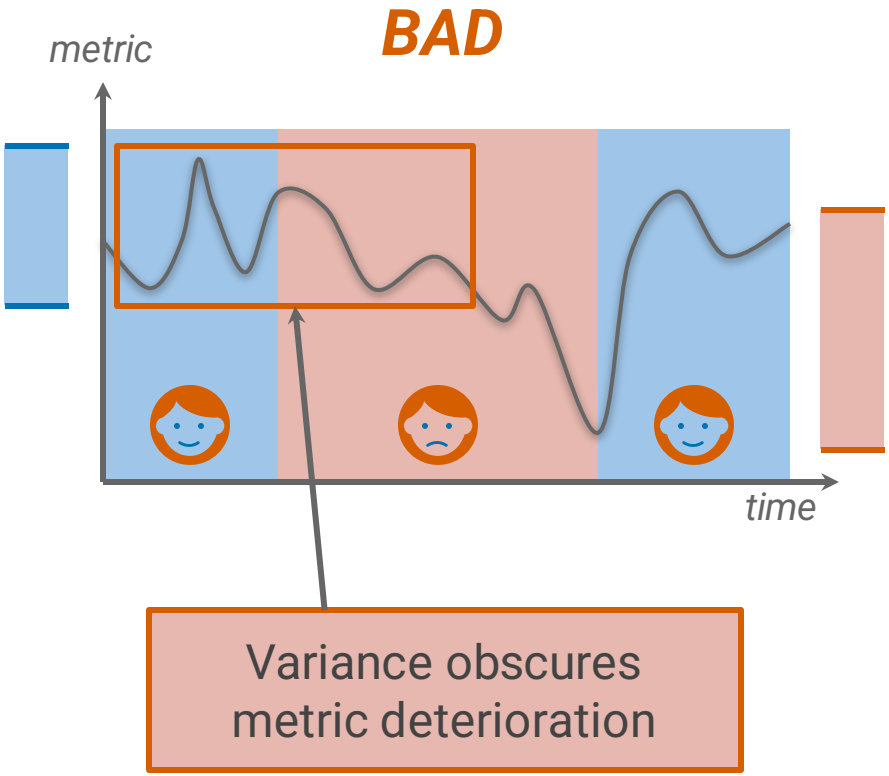

unhappy
users
time

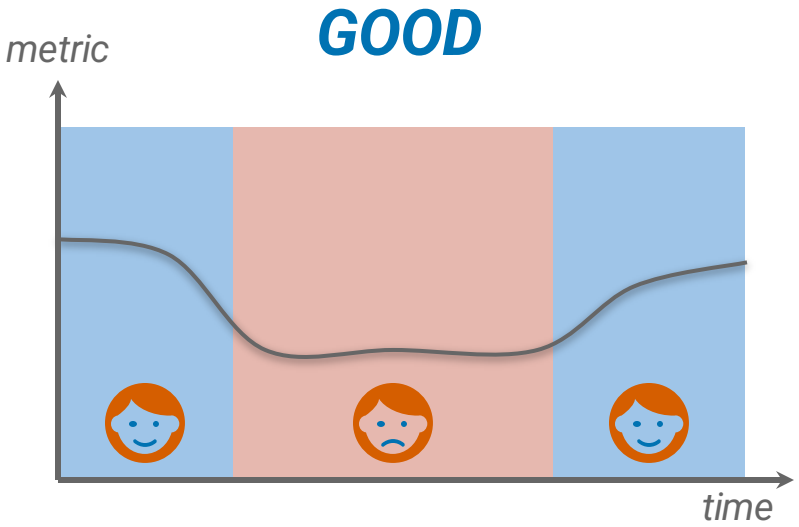
## BAD

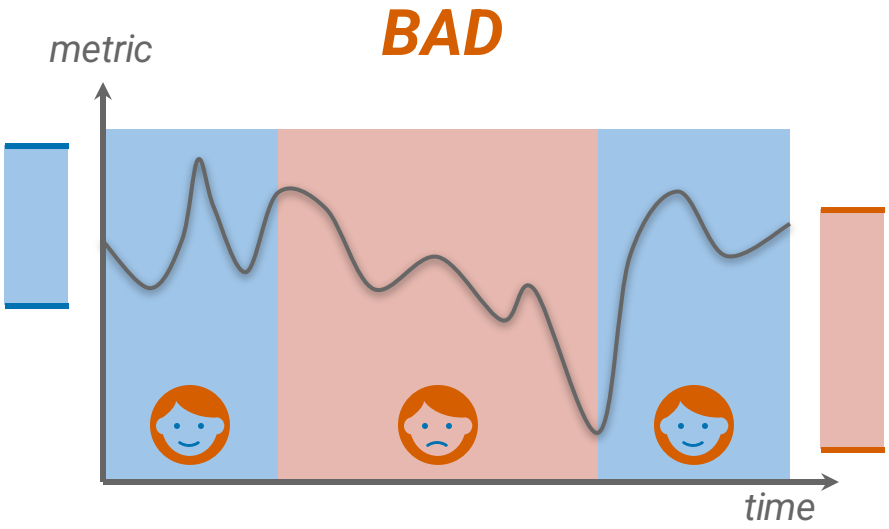


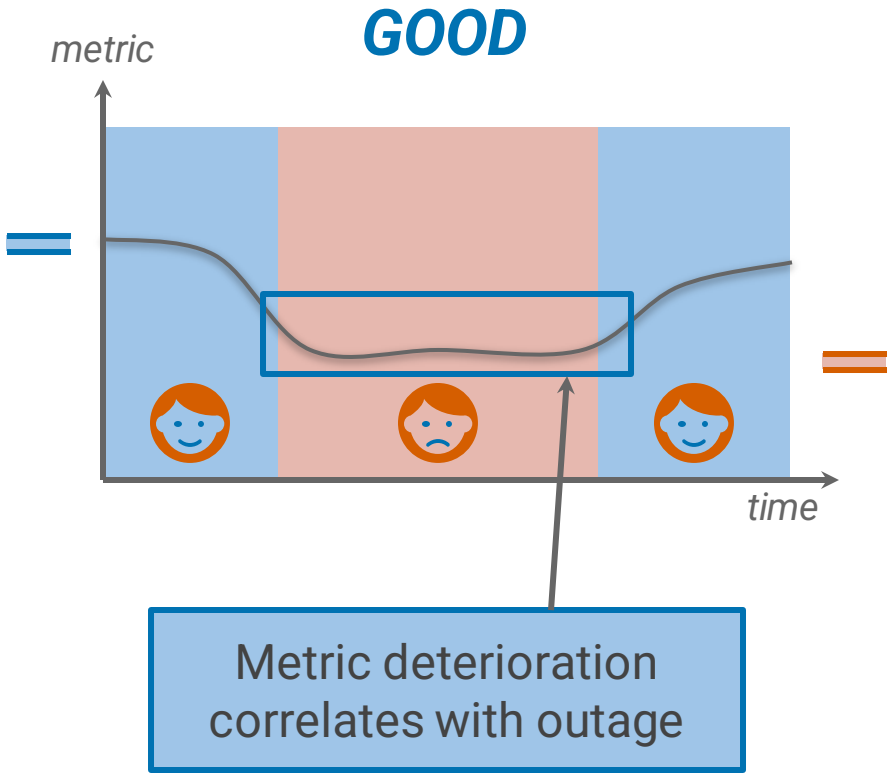

## GOOD

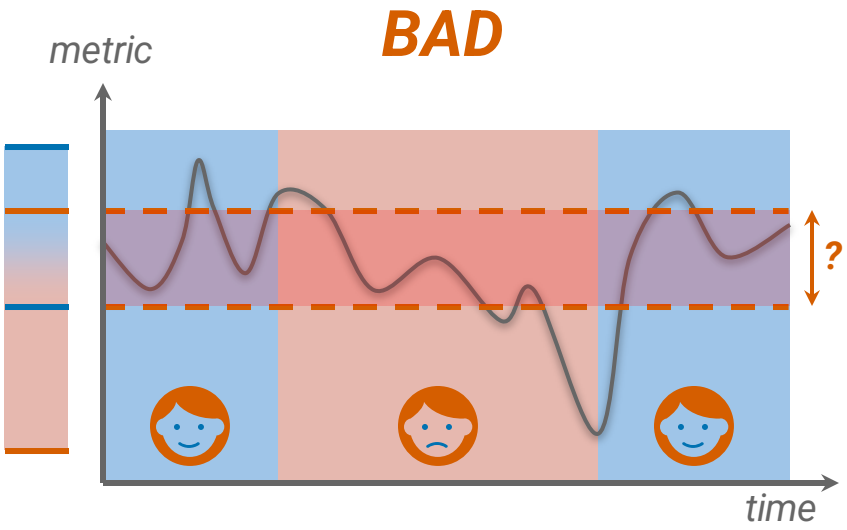


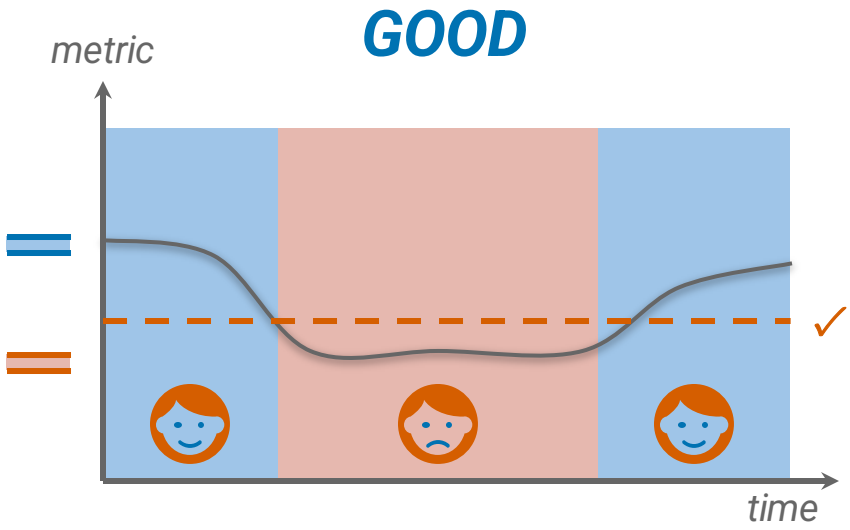

BAD
metric
time
Variance obscures
metric deterioration


GOOD
metric
time


BAD
metric
time

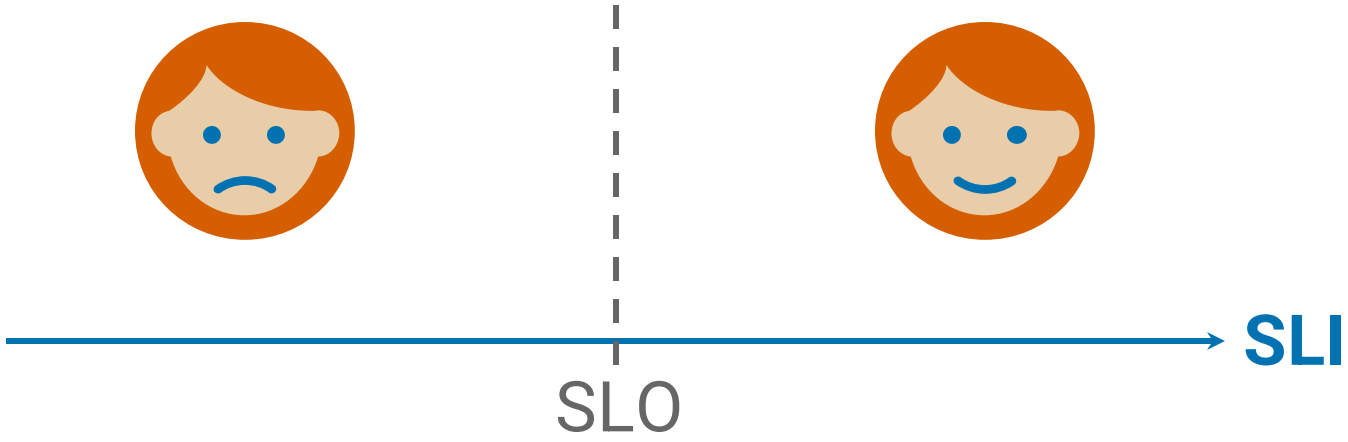

GOOD
metric
time
Metric deterioration
correlates with outage




Metric provides poor signal-to-noise ratio

Metric provides good signal-to-noise ratio

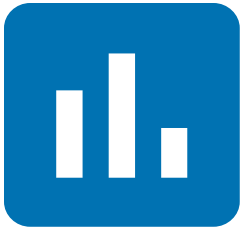

SLI
SLO

$$\text{SLI} : \left( \frac{\text{good events}}{\text{valid events}} \right) \times 100\%$$

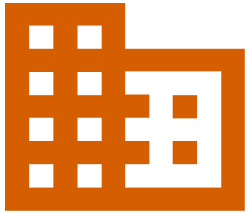
# 3–5 SLIs*

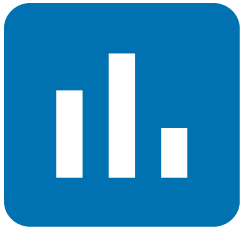
* per user journey


SLI
SLO



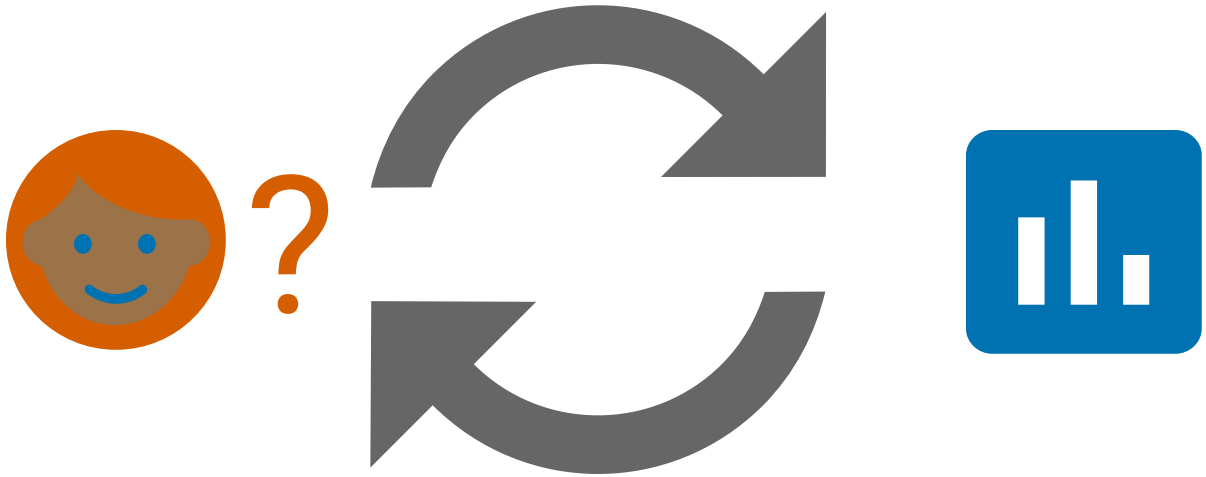
What **performance** does the **business** need?





**User expectations**
are *strongly* tied to
**past performance**

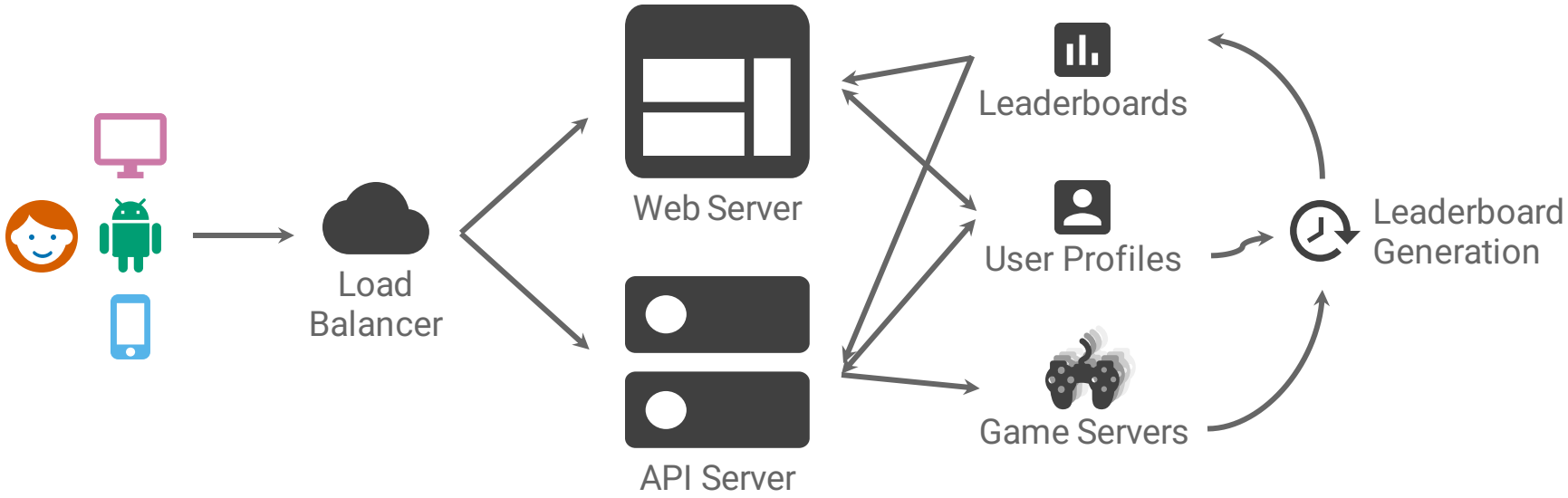
Continuous



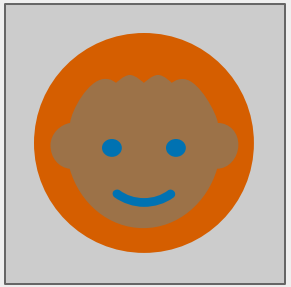
Improvement

# Developing SLOs and SLIs

# Our Game: Fang Faction



https://fangfactiongame.com/profile/someuser



SomeUser

Tribe of Frog

Faction Score: **31337**

*Midwest Canyon*

1. Tri-Bool
2. **65535**
3. Tri Repetae **61995**
4. Triassic Five **52391**
5. Tricksy Hobbits **37164**
6. Tribe of Frog **31337**

Trite Examples **29243**

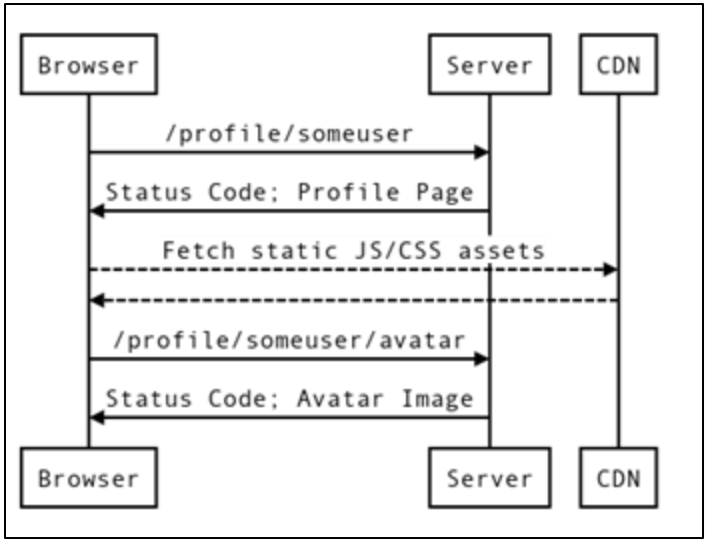
# SomeUser's Profile

Faction Name: Tribe of Frog

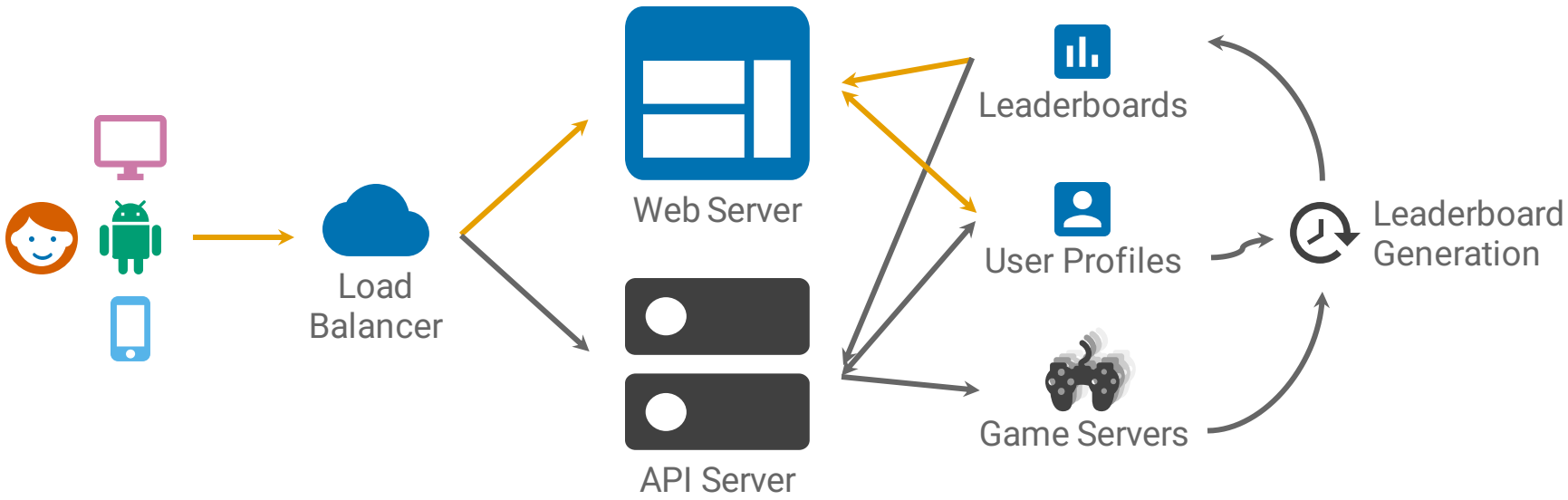
Leader Name: SomeUser

Email Address: user@example.com

Update



# Loading a Profile Page



# SLI Menu

**Request / Response**

Availability
Latency
Quality

**Data Processing**

Coverage
Correctness
Freshness
Throughput

**Storage**

Throughput
Latency

## Availability

The **profile page** should load **successfully**

## Latency

The **profile page** should load **quickly**

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **valid** requests served **successfully**.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **valid** requests served **faster** than a threshold.

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **valid** requests served **successfully**.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **valid** requests served **faster** than a threshold.

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **HTTP GET** requests for **/profile/{user}** or **/profile/{user}/avatar** served **successfully**.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **HTTP GET** requests for **/profile/{user}** served **faster** than a threshold.

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **HTTP GET** requests
for **/profile/{user}** or **/profile/{user}/avatar**
served **successfully**.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **HTTP GET** requests
for **/profile/{user}**
served **faster** than a threshold.

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **HTTP GET** requests
for **/profile/{user}** or **/profile/{user}/avatar**
that have ***2XX***, ***3XX*** or ***4XX*** **(excl. 429)** status.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **HTTP GET** requests
for **/profile/{user}**
served **within *X* ms**.

# SLI Menu

**Measurement Strategies**

Application-level Metrics

Logs Processing

Front-end Infra Metrics

Synthetic Clients/Data

Client-side Instrumentation

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **HTTP GET** requests
for **/profile/{user}** or **/profile/{user}/avatar**
that have ***2XX***, ***3XX*** or ***4XX* (excl. 429)** status
measured at the **load balancer**.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **HTTP GET** requests
for **/profile/{user}**
that send their **entire response within *X* ms**
measured at the **load balancer**.

# Activity

Postmortem

# Postmortem: Blank Profile Pages

## *Impact*

From 08:43 to 13:17 CEST, users accessing their profile pages received incomplete responses. This rendered them unable to view or edit their profile.

## *Root Causes and Trigger*

The proximate root cause was a bug in the web server's handling of unicode HTML templates. The trigger was `commit a6d78d13`, which changed the profile page template to support localization, but at the same time accidentally introduced unicode quotation marks (U+201C “, U+201D ”) into the template HTML. When the web server encountered these instead of the standard ascii quotation mark (U+0022 "), the template engine aborted rendering of the output.

## *Detection*

Because the aborted rendering process did not throw an exception, the HTTP status code for the incomplete responses was still `200 OK`. The problem thus went undetected by our SLO-based alerts. The support and social media teams manually escalated concerns about a substantially increased level of complaints relating to the profile page at 12:14 CEST.

## *Lessons Learned*

Things that went well:

- Support and social media teams were able to find the correct escalation path and successfully contact the ops team.

Things that went poorly:

- HTTP status code SLIs could not detect incomplete responses.
- Web server used a severely outdated, vendored version of the templating engine with a substantially broken unicode support.

Where we got lucky:

- User profile page is relatively unimportant to our revenue stream.

## *Action Items*

... to be determined.

# Question: How can we improve these SLIs?

## Availability

The **profile page** should load **successfully**

- How do we define **success**?
- Where is the success / failure **recorded**?

The proportion of **HTTP GET** requests
for **/profile/{user}** or **/profile/{user}/avatar**
that have ***2XX***, ***3XX*** or ***4XX* (excl. 429)** status
measured at the **load balancer**.

## Latency

The **profile page** should load **quickly**

- How do we define **quickly**?
- When does the timer **start / stop**?

The proportion of **HTTP GET** requests
for **/profile/{user}**
that send their **entire response within *X* ms**
measured at the **load balancer**.

## Availability

Proportion of **HTTP GET** requests
for **/profile/{user}** or **/profile/{user}/avatar**
that have ***2XX***, ***3XX*** or ***4XX* (excl. 429)** status
measured at the **load balancer**

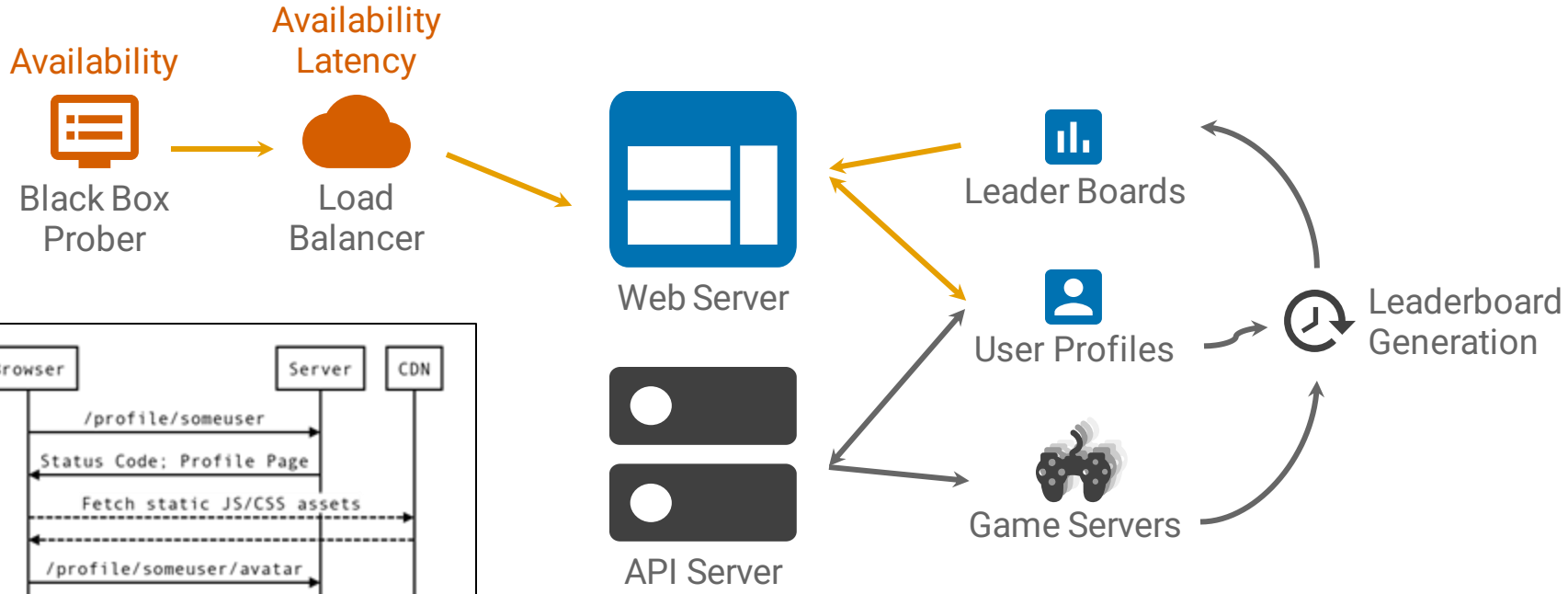
*and*

Proportion of **HTTP GET** requests
for **/profile/prober_user** and **all linked resources**
returning **valid HTML containing "ProberUser"**
measured by a **black-box prober** every 5s

## Latency

Proportion of **HTTP GET** requests
for **/profile/{user}**
that send their **entire response within *X* ms**
measured at the **load balancer**

# Do the SLIs cover the failure modes?



# Activity

## Define SLO Targets

# What goals should we set for the reliability of our journey?

Your objectives should have both a **target** and a **measurement window**

| Service | SLO Type | Objective |
|---|---|---|
| Web: User Profile | Availability | **99.95% successful** in **previous 28d** |
| Web: User Profile | Latency | **90%** of requests **< 500ms** in **previous 28d** |
| ... | ... | |

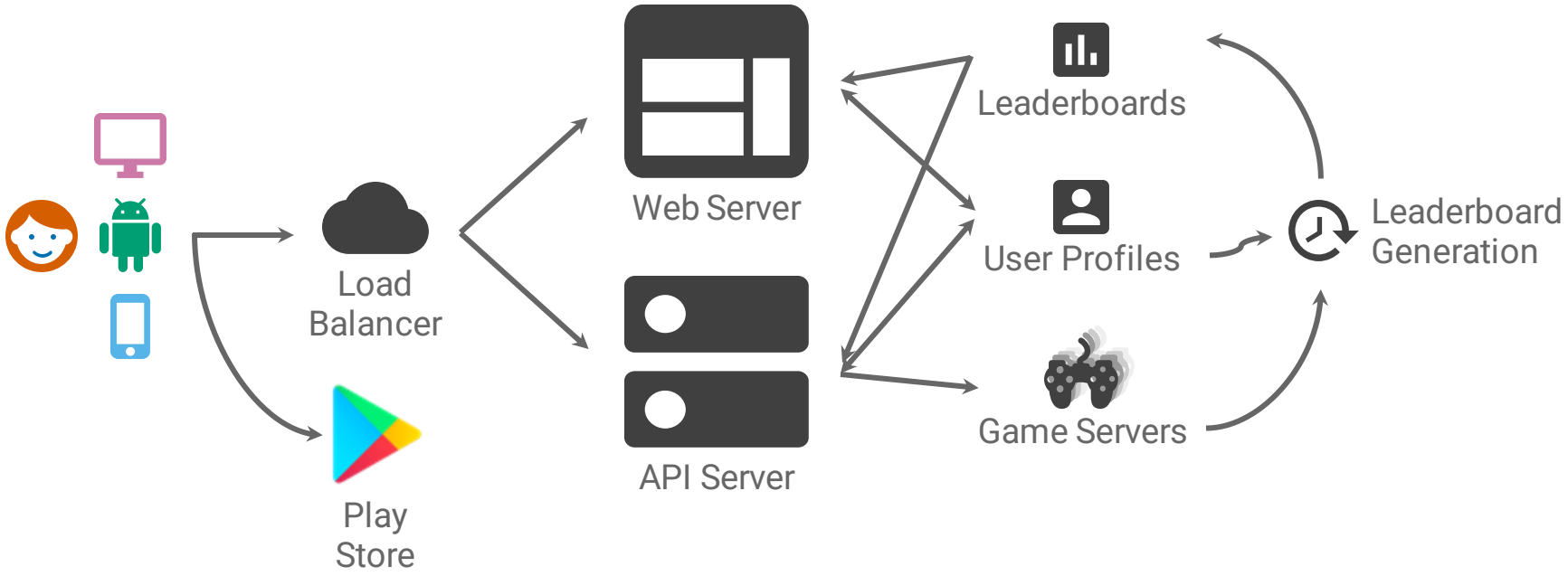
# Workshop: Let's develop some more SLIs and SLOs!

Follow the **process** we demonstrated for the *Buy In-Game Currency* journey:

1. Choose **SLI specifications** from the menu (see booklet, p6)
2. Substitute **definitions** in to create a detailed **SLI implementation**
3. Walk through user journey and look for **coverage gaps**
4. Set **aspirational SLOs** based on **business needs**

**Booklet: https://sre.google/resources/practices-and-processes/art-of-slos/**

# Our Game: Fang Faction



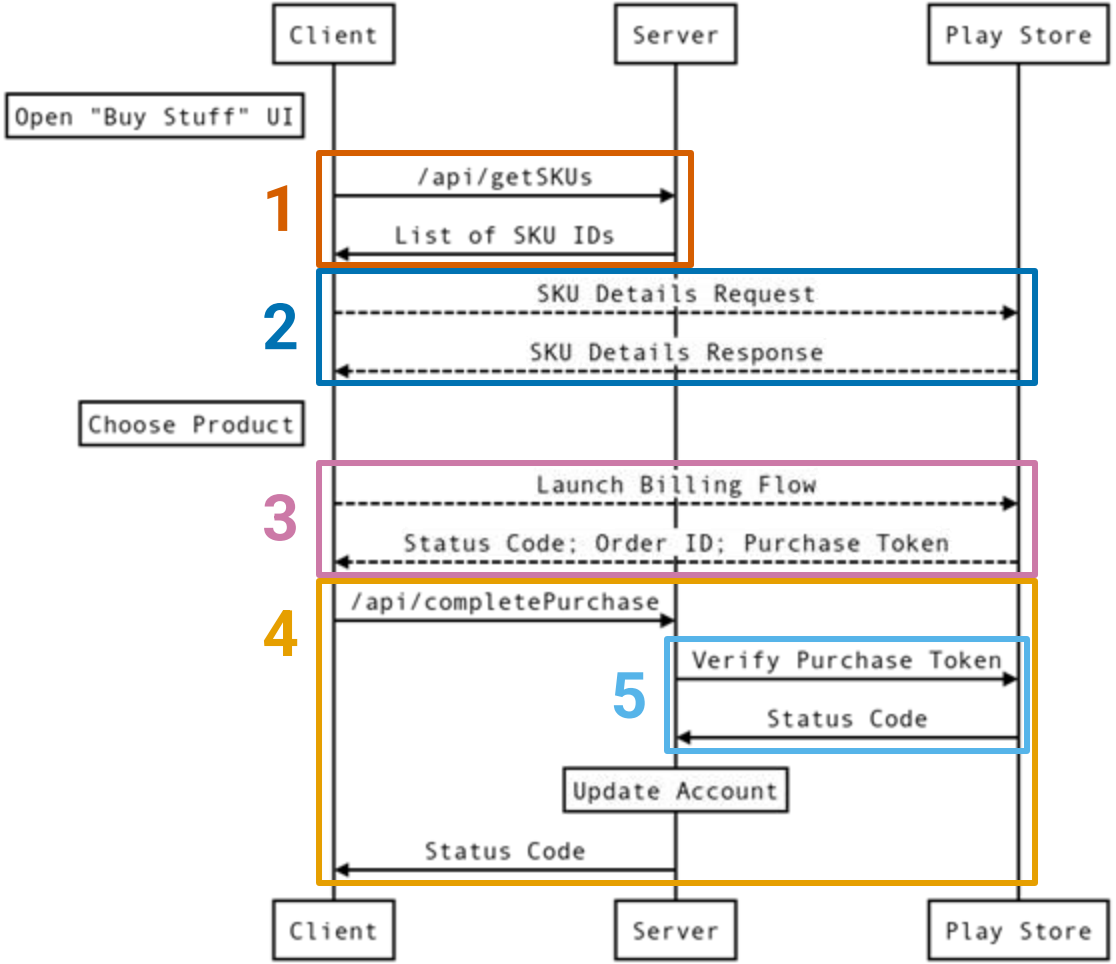
# Buy In-Game Currency

## Model Answer

# Break Down The Journey

**Five** request/response pairs

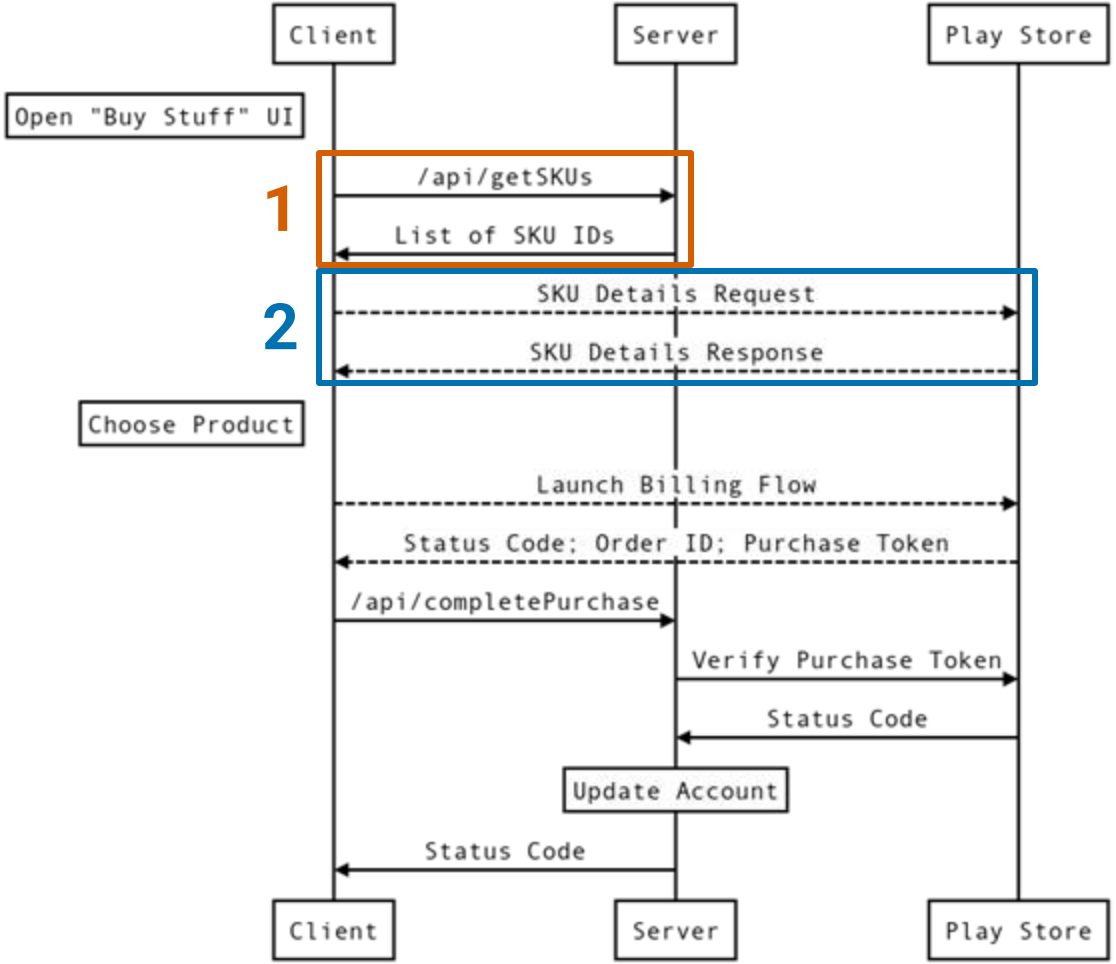
1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store



# Break Down The Journey

Journey has **two** parts. **A:** Fetch SKUs

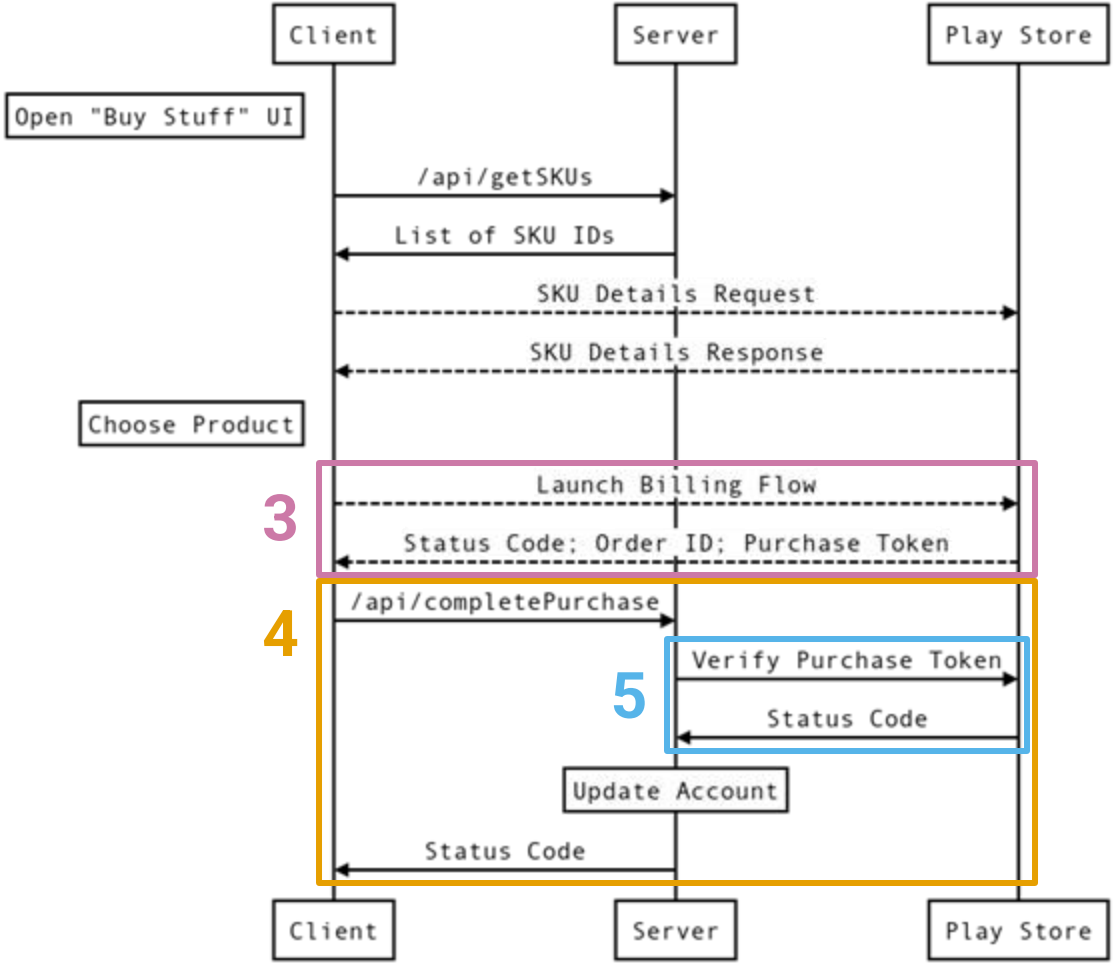
1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store



# Break Down The Journey

Journey has **two** parts. **B:** Buy Item

1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store

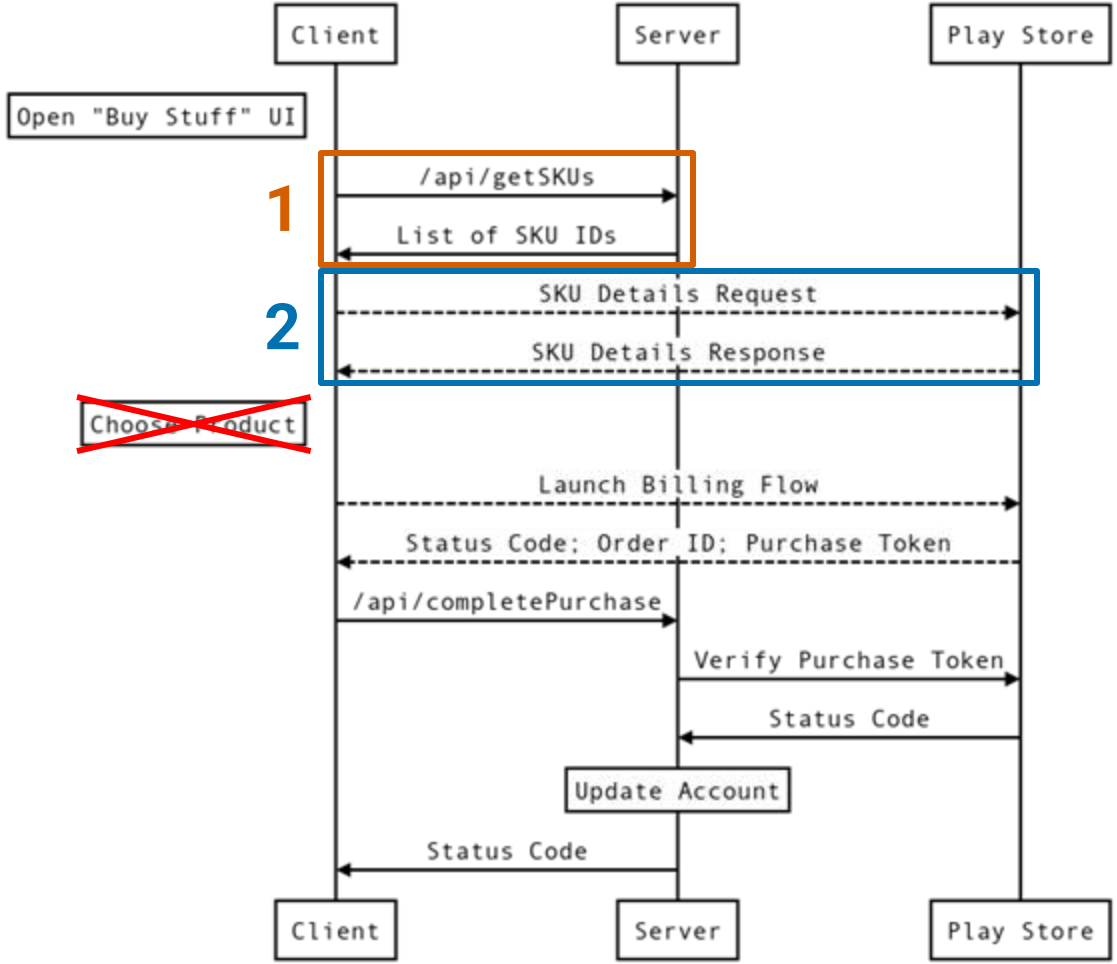


# Break Down The Journey

User might choose **not** to buy an item :-(

1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store

We have to treat these parts **separately**!

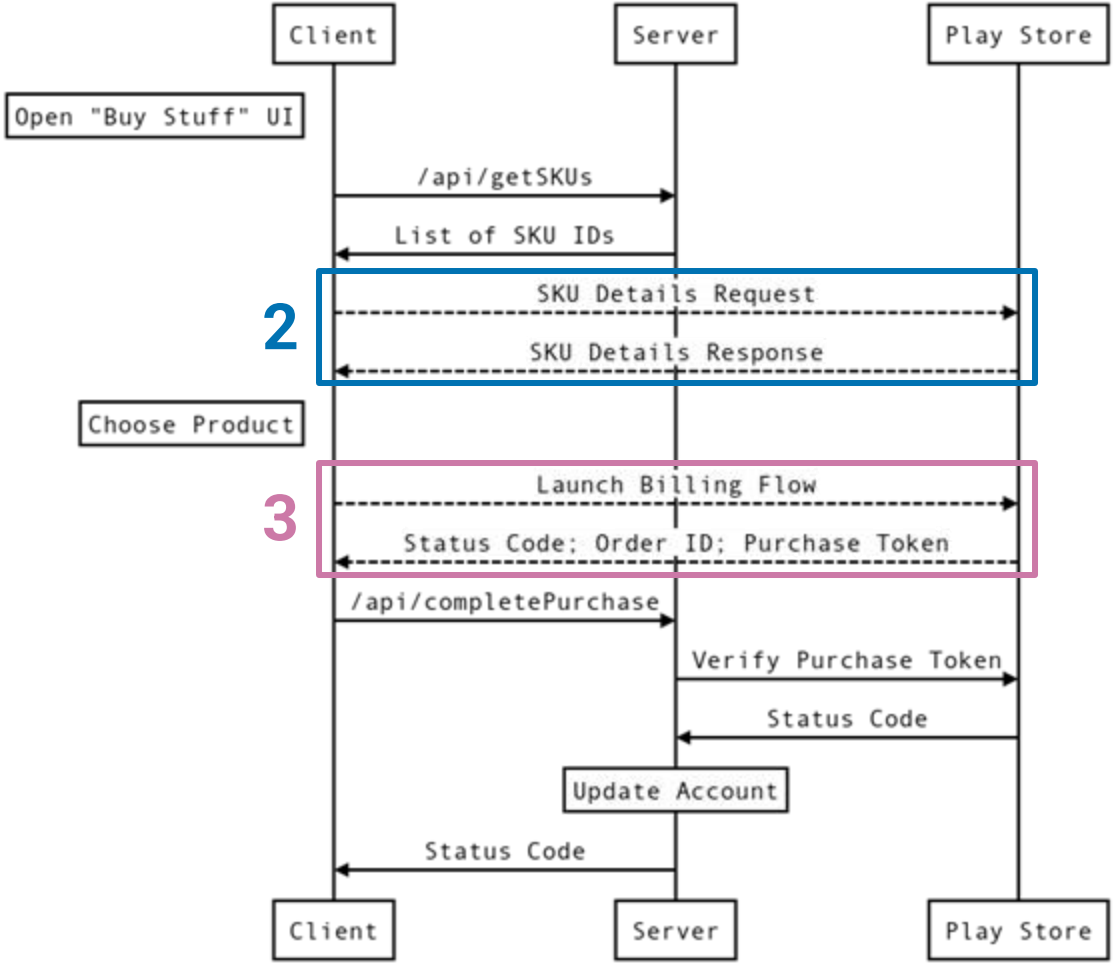


# Break Down The Journey

**Two** requests don't hit API server at all!

1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store

Server or load balancer metrics **may not give enough coverage** of the journey :-(



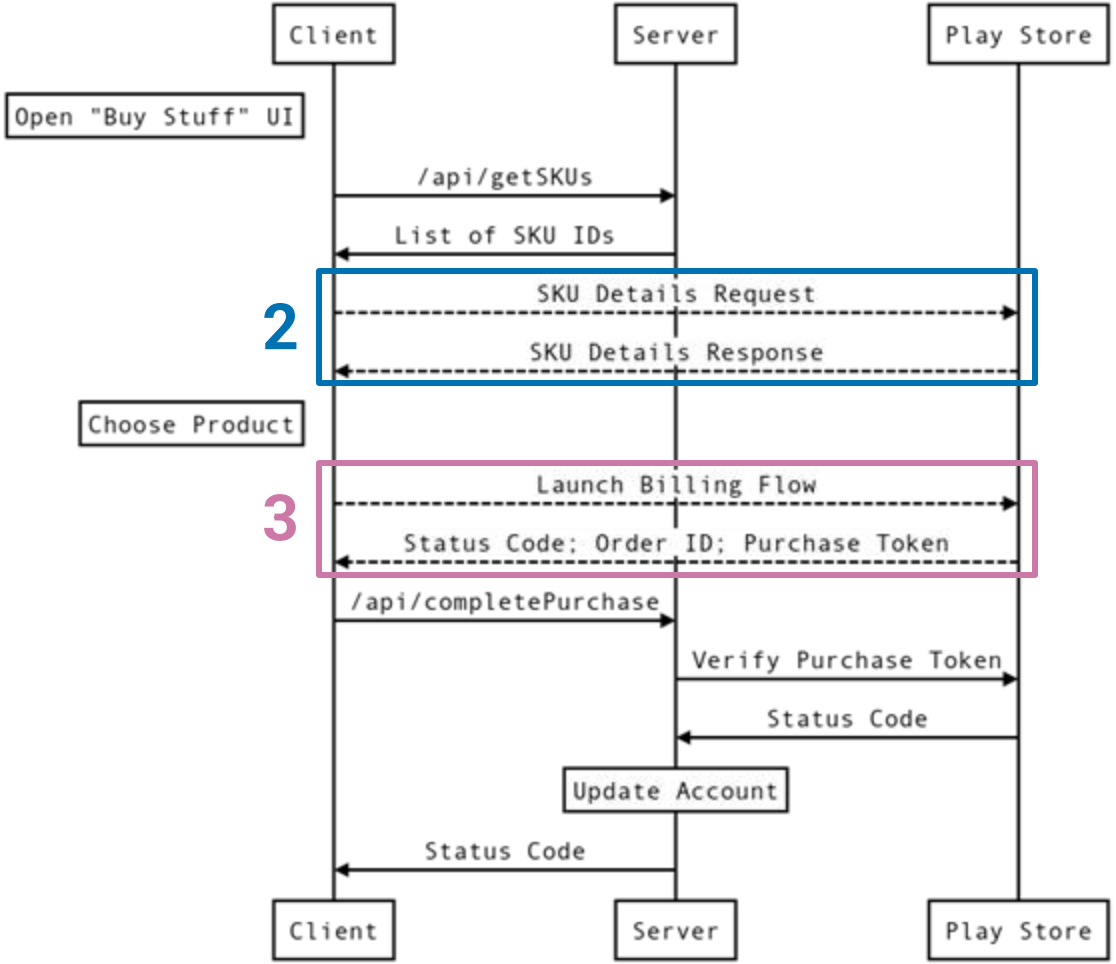
# Break Down The Journey

**Two** requests don't hit API server at all!

1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store

Server or load balancer metrics **may not give enough coverage** of the journey :-(

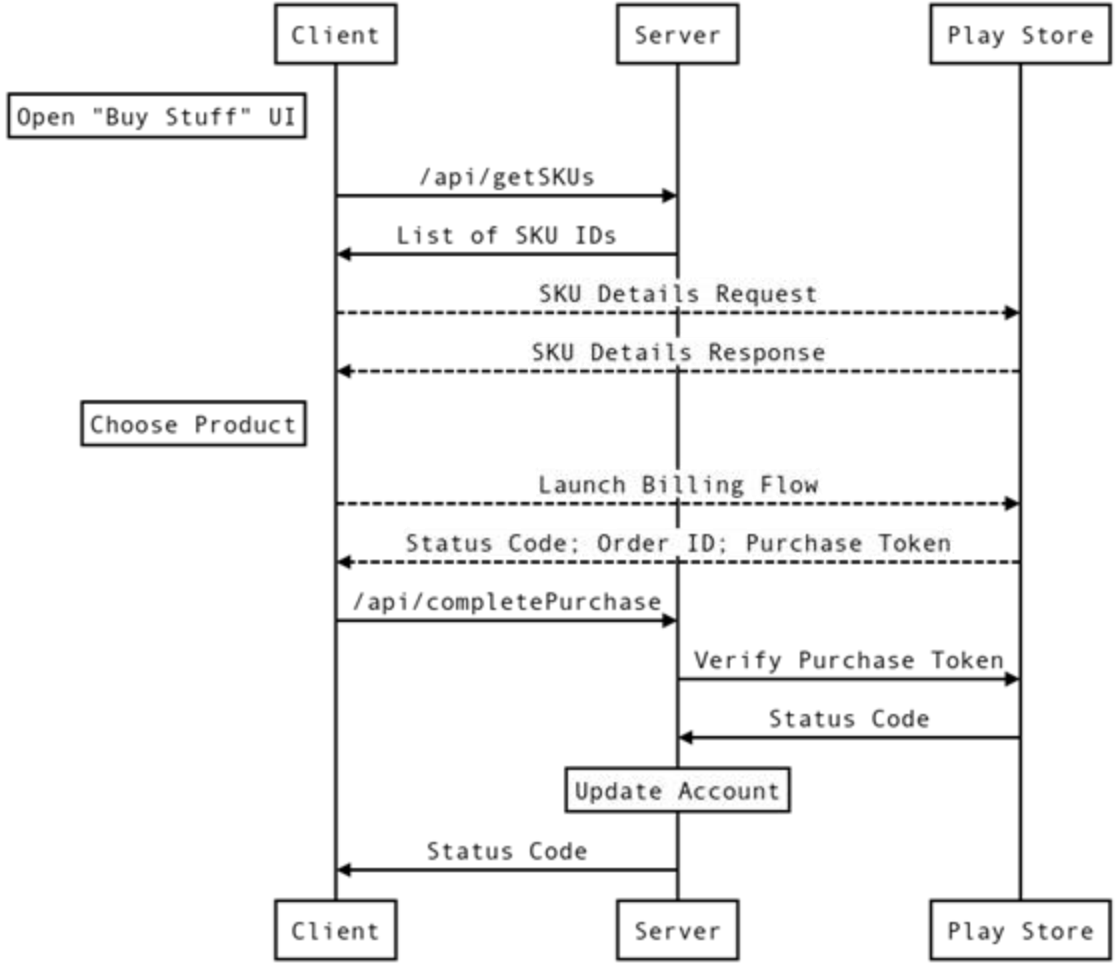
… we'll have to ask our users to **consent** to client-side telemetry.



# Buy Flow
# What SLIs?

Buy Flow journey is
**Request / Response**

SLI menu suggests we use
**Availability** and **Latency** SLIs



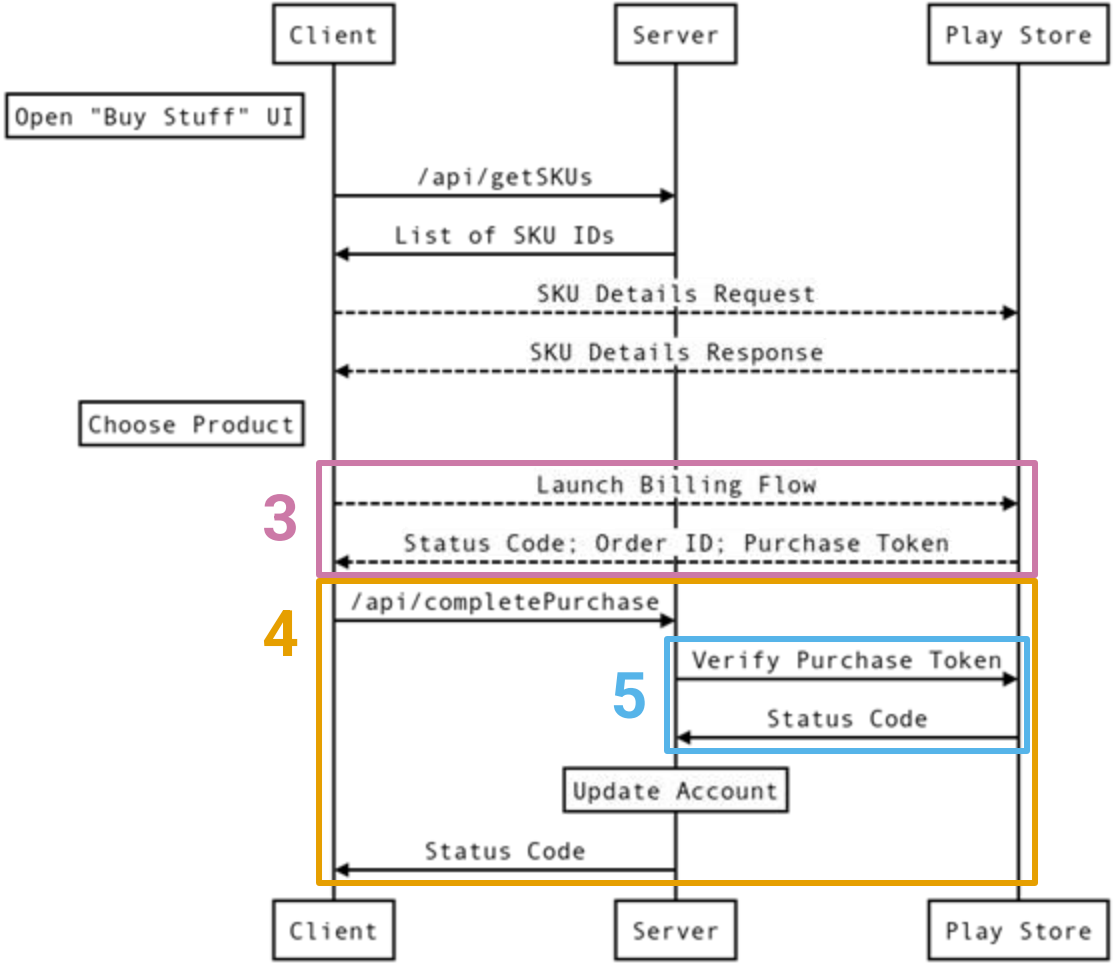
# Buy Flow Availability: Specification

**B** makes money, so let's start with that

1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store

Availability SLI Specification

The proportion of ***valid*** requests served ***successfully***.



# Buy Flow Availability: Valid Requests

Availability SLI

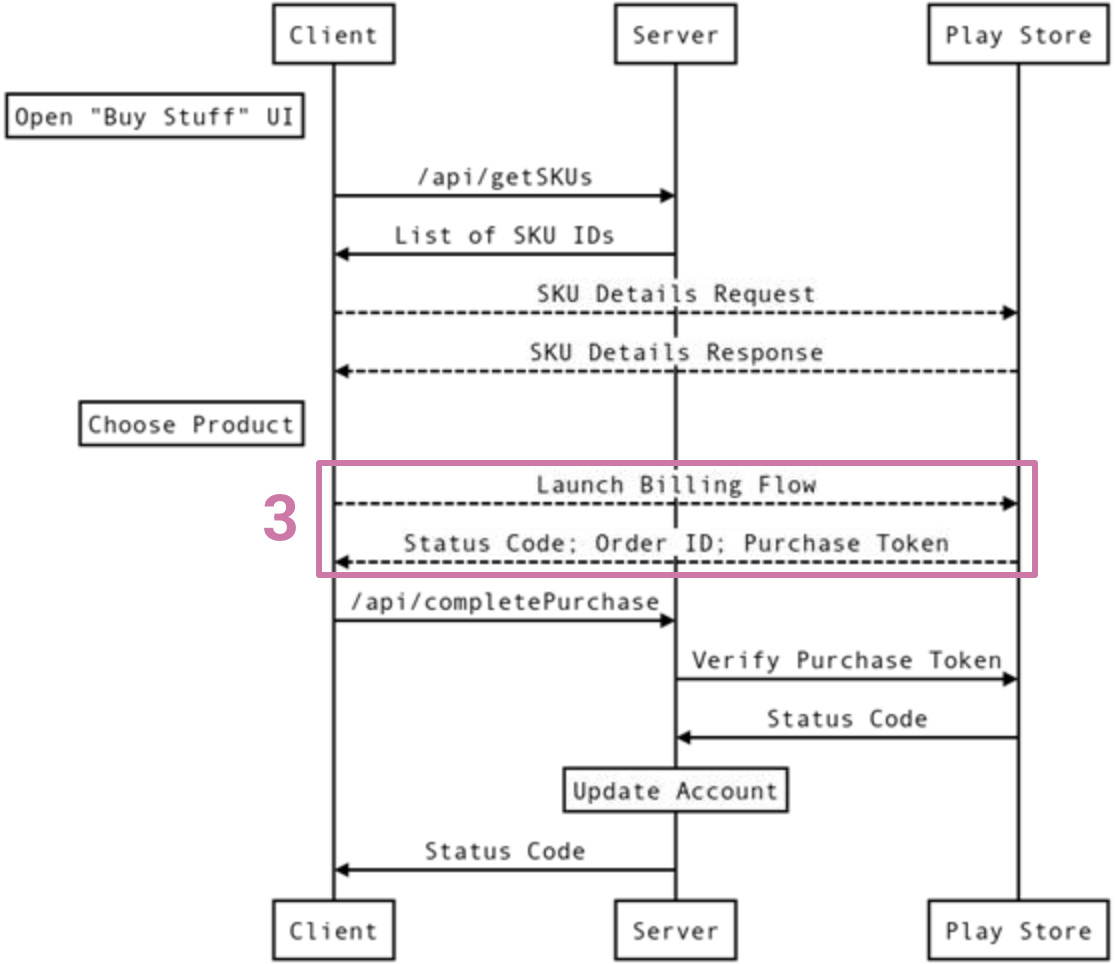
The proportion of ***valid*** requests served ***successfully***.

… but which requests are ***valid***?

3. User launches Play billing flow
4. Send token to API server?
5. Verify token with Play Store?

Launching the billing flow indicates a user's ***intent*** to buy a product

Users **consenting** to client-side telemetry collection allows us to **track** this intent



# Buy Flow Availability: Success Criteria

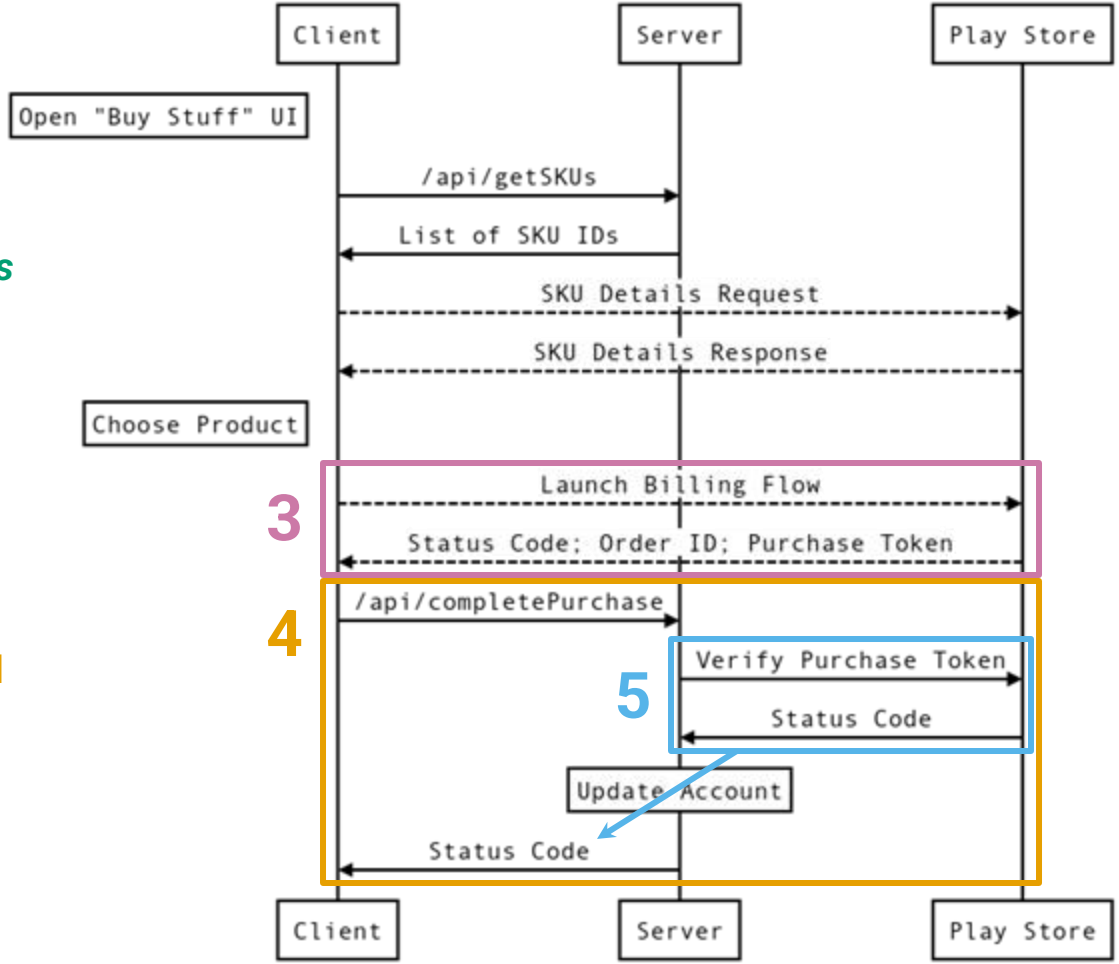
Availability SLI

The proportion of ***launched billing flows from users consenting to collection*** served ***successfully***.

… and how do we determine ***success***?

All interactions must be successful!

3. Good status code; purchase token
4. Good status code; account updated
5. Good status code; valid token
   - *Return **402** to API call when token is invalid*



# Buy Flow Availability: Measurement

Availability SLI

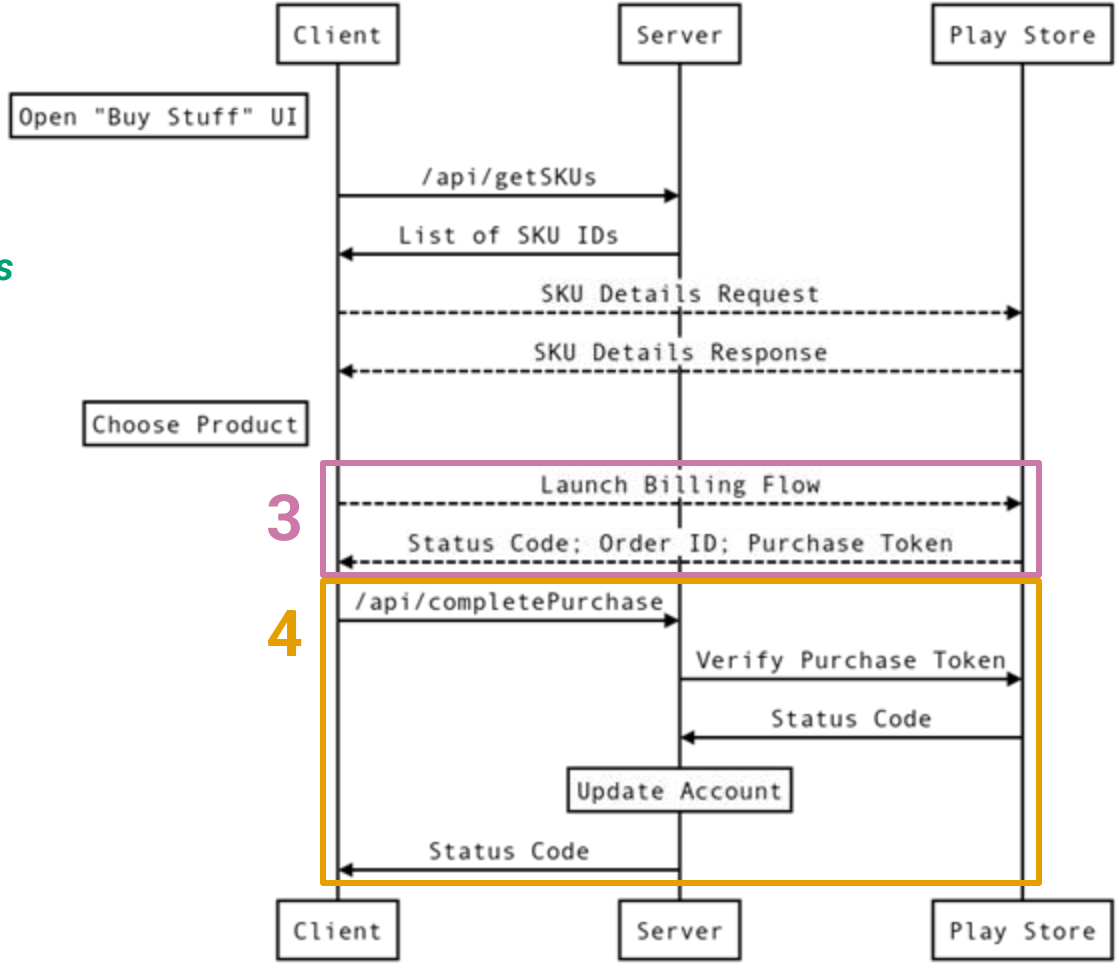
The proportion of ***launched billing flows from users consenting to collection*** where ***the billing flow returns***:

- OK *and a* purchase token
- *or* FEATURE_NOT_SUPPORTED
- *or* ITEM_UNAVAILABLE
- *or* USER_CANCELED

and ***/api/completePurchase returns***:

- 200 OK *and* Parseable JSON
- *or* 402 Payment Required

… but where are we ***measuring*** this?



# Buy Flow Availability: Measurement

Availability SLI

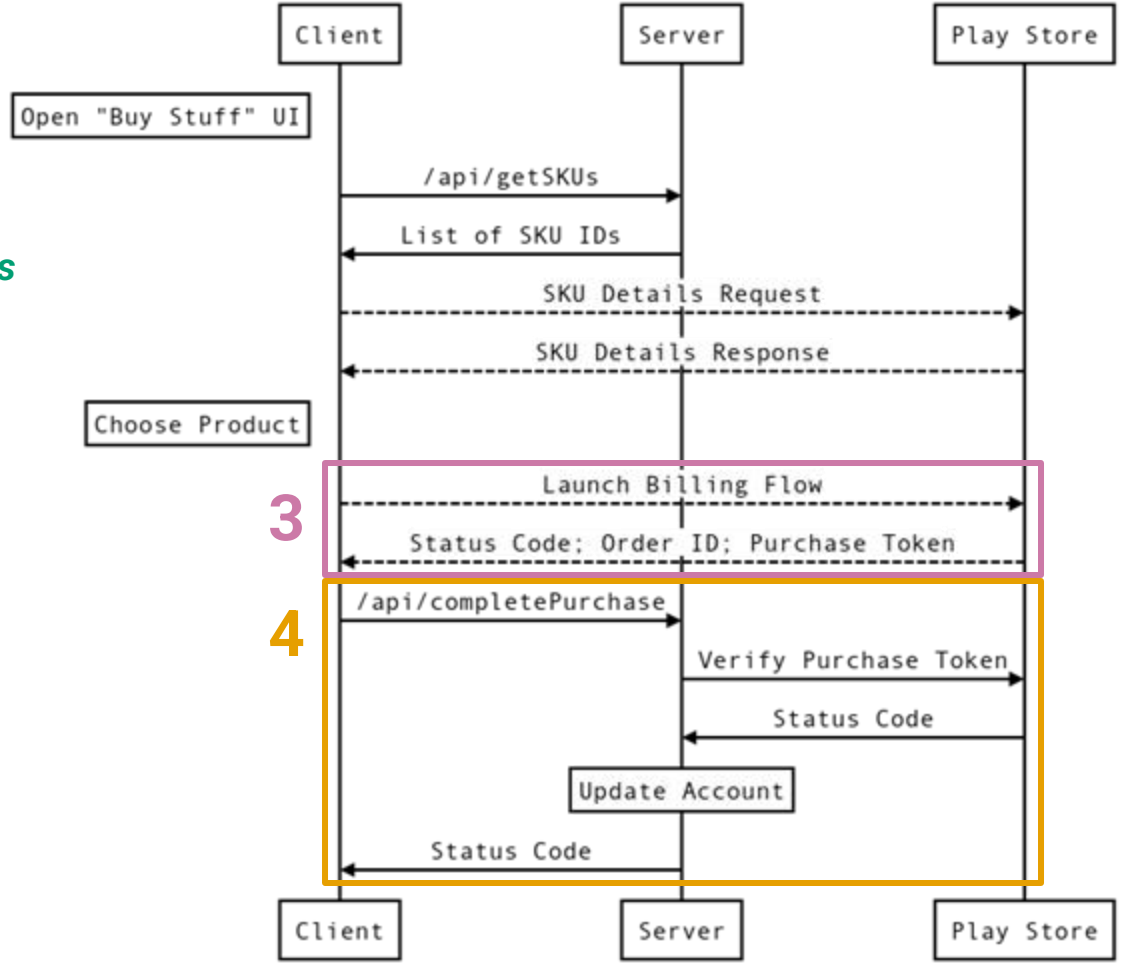
The proportion of ***launched billing flows from users consenting to collection*** where ***the billing flow returns***:

- OK
- *or* FEATURE_NOT_SUPPORTED
- *or* ITEM_UNAVAILABLE
- *or* USER_CANCELED

and ***/api/completePurchase returns***:

- 200 OK
- *or* 402 Payment Required
- *and* Parseable JSON

measured by the ***game client*** and reported back asynchronously.



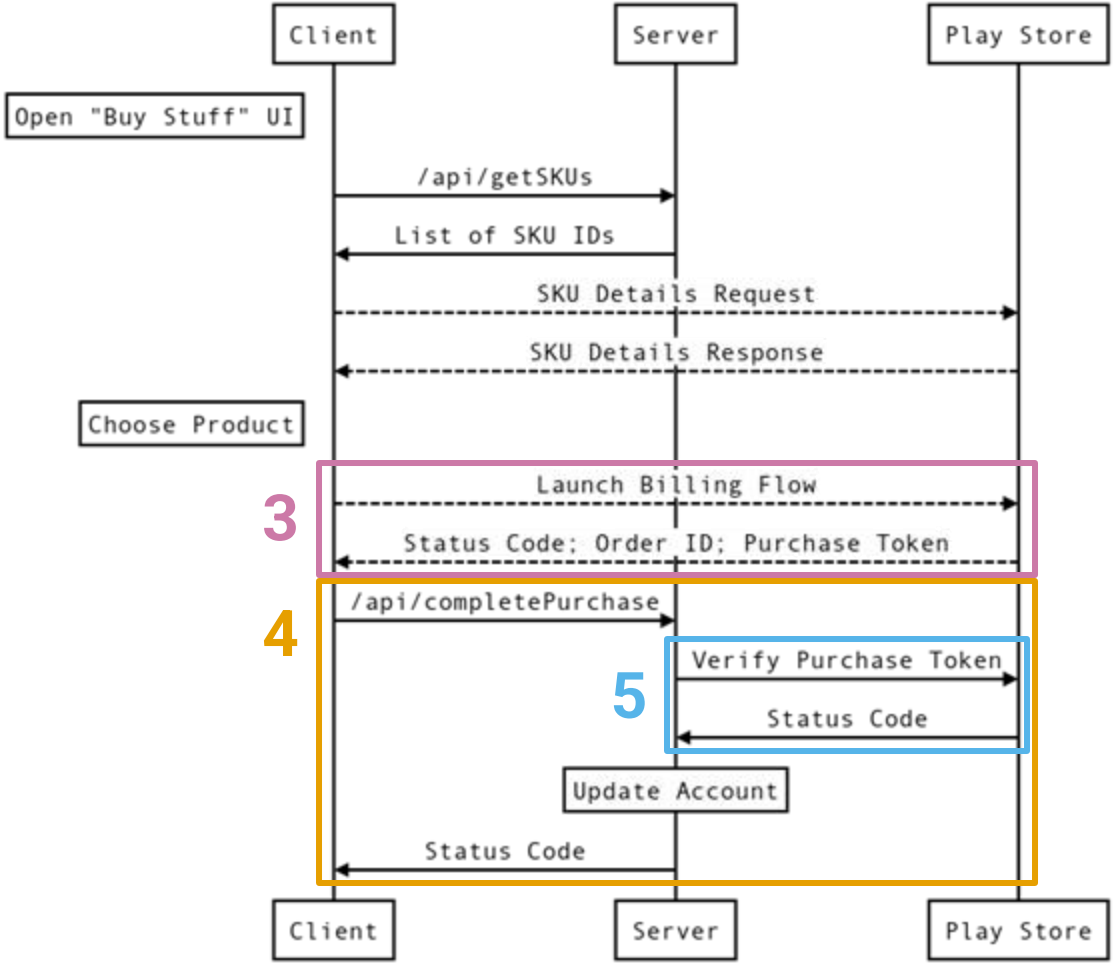
# Buy Flow Latency: Specification

We want to measure latency for **B** too!

1. Fetch list of SKUs from API server
2. Fetch SKU details from Play Store
3. User launches Play billing flow
4. Send token to API server
5. Verify token with Play Store

Latency SLI Specification

The proportion of ***valid*** requests served ***faster*** than a threshold.



# Buy Flow Latency: Valid Requests

Latency SLI

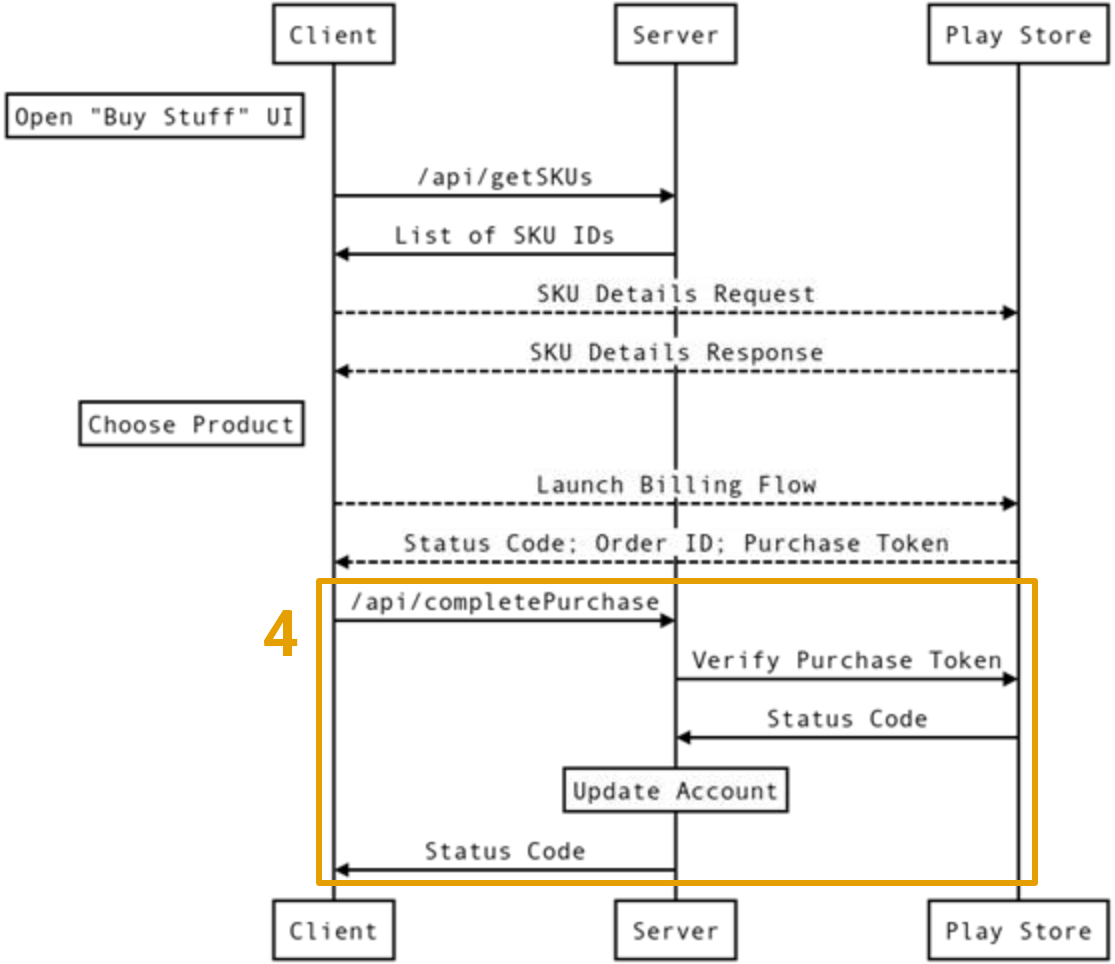
The proportion of ***valid*** requests served ***faster*** than a threshold.

... but which requests are ***valid***?

3. User launches Play billing flow?
4. Send token to API server
5. Verify token with Play Store?

Why not **3**?

- Too variable, SLI will have poor SnR
- Billing flow contains lots of "poking device with a finger" time



# Buy Flow Latency: "Too Slow" Threshold

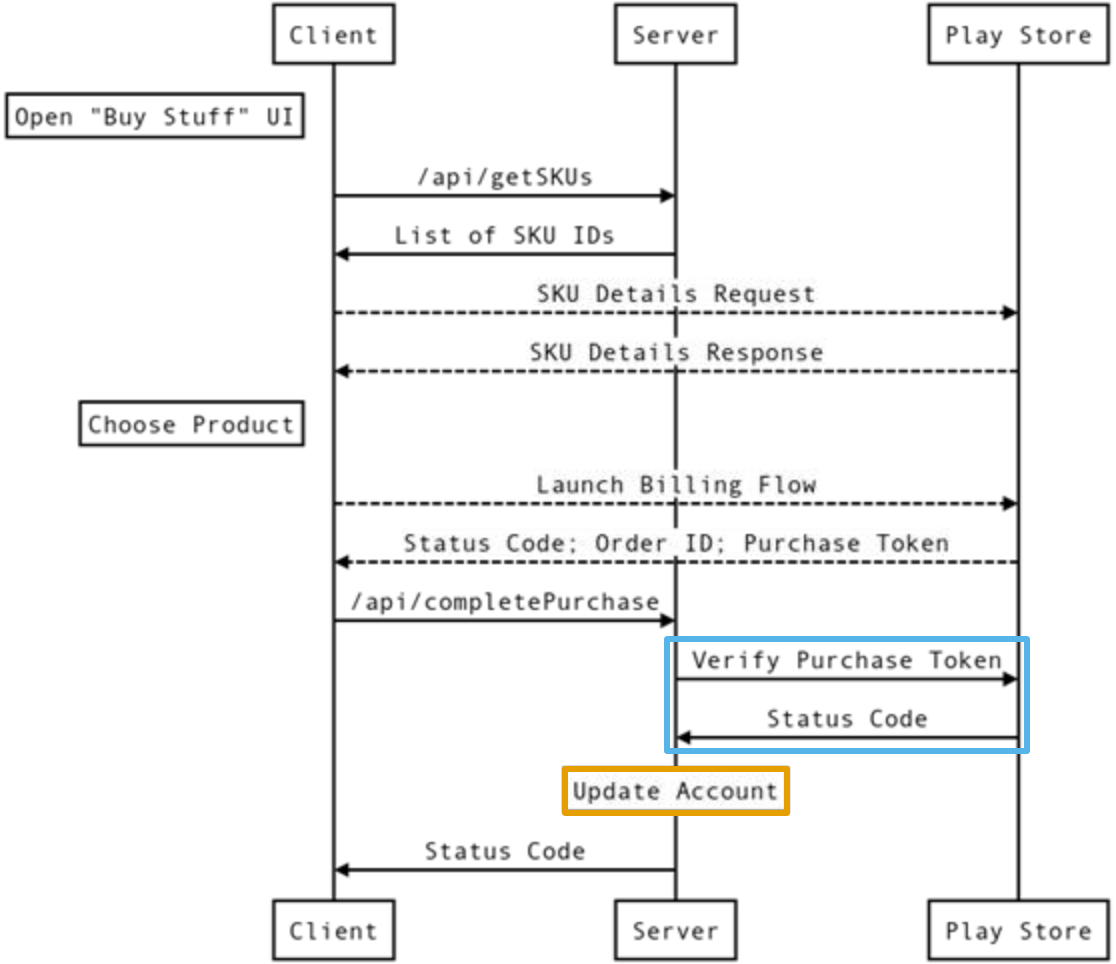
Latency SLI

The proportion of ***/api/completePurchase requests*** served ***faster*** than a threshold.

… and what is ***fast enough***?

Rough estimate time!

- Verify Token ***<= 500ms***
- Database Write ***<= 200ms***
- Round up a bit…

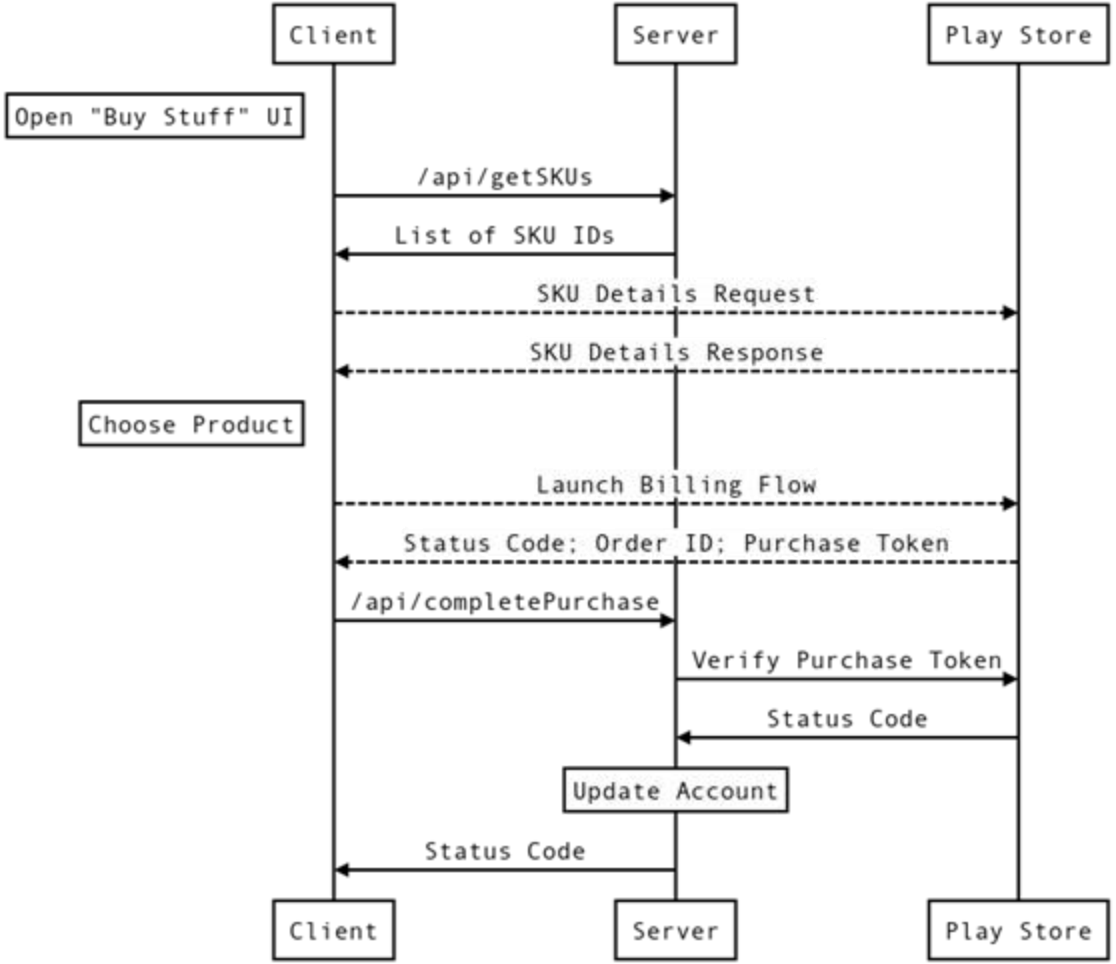


# Buy Flow Latency: Measurement

Latency SLI

The proportion of ***/api/completePurchase requests*** served ***within 1000ms***.

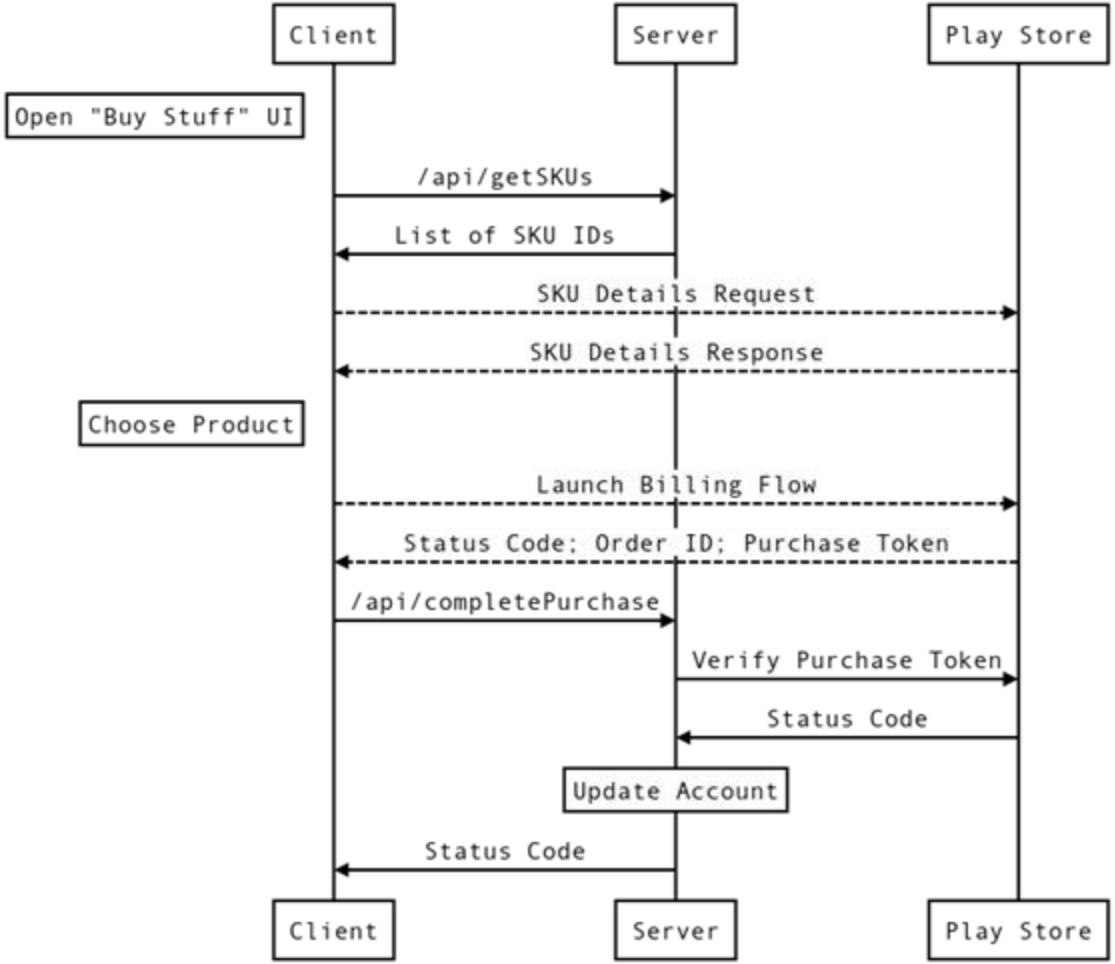
… but where are we ***measuring*** this?
Where does the timer start/stop?



# Buy Flow Latency: Measurement

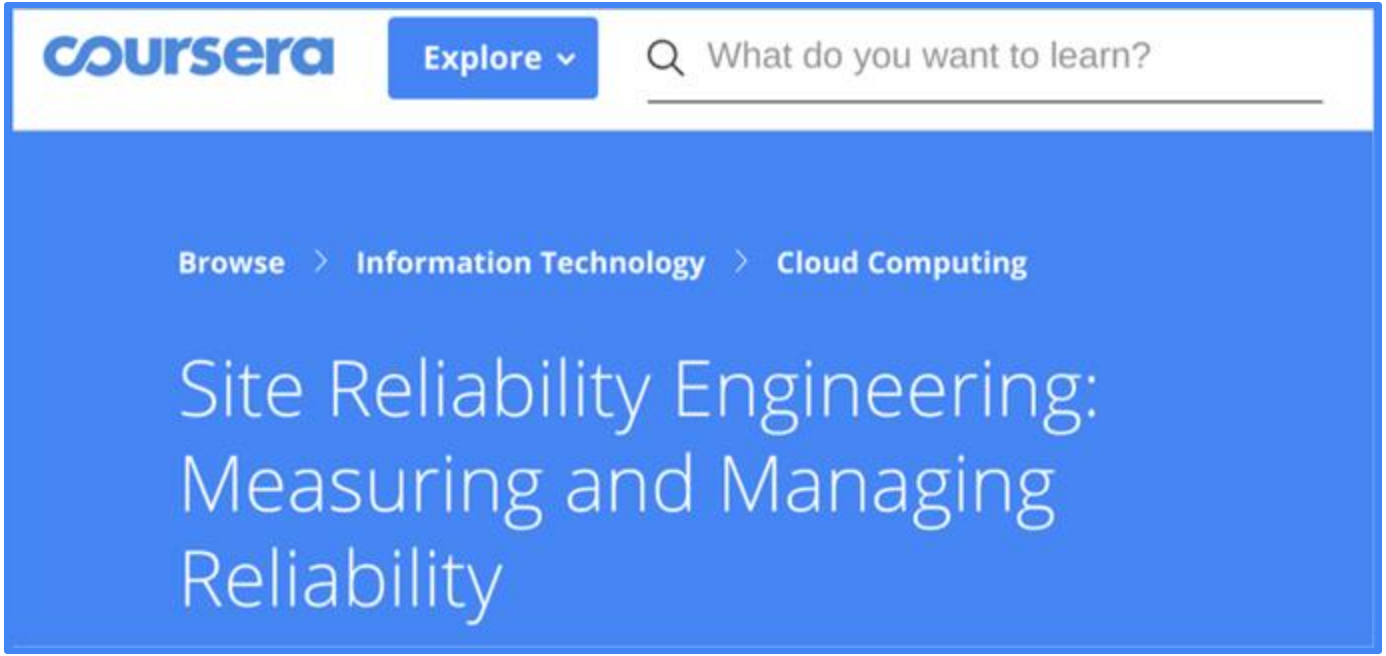
Latency SLI

The proportion of ***/api/completePurchase requests*** where the ***complete response*** is returned to the client ***within 1000ms*** measured at the ***load balancer***.



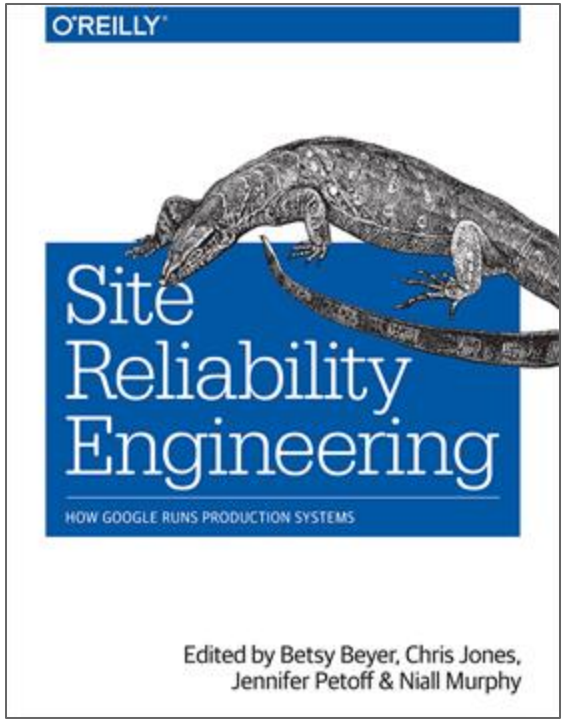
A *brief* word from
our sponsors...

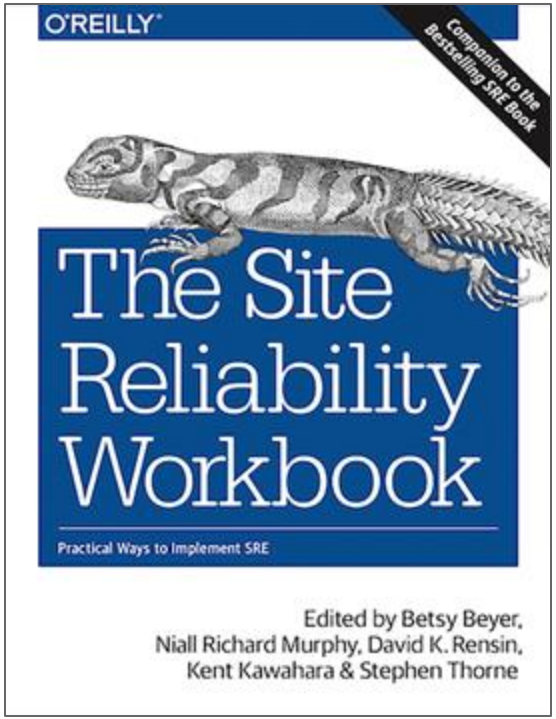
https://cre.page.link/art-of-slos




Want to learn more about SLOs? Take our course on Coursera:

https://cre.page.link/coursera




Both of these are now available in HTML format for free!

https://landing.google.com/sre/books/