

From Zero to Production-Ready in Minutes

Tim Bozarth
@timbozarth



NETFLIX

Today's focus

Dev Experience:
Level up your Eng
Effectiveness

Agenda

- 1. It was the best of times...**
- 2. Best practices made easy**
- 3. Goodbye hand-written clients**
- 4. From NIH to OSS**

1

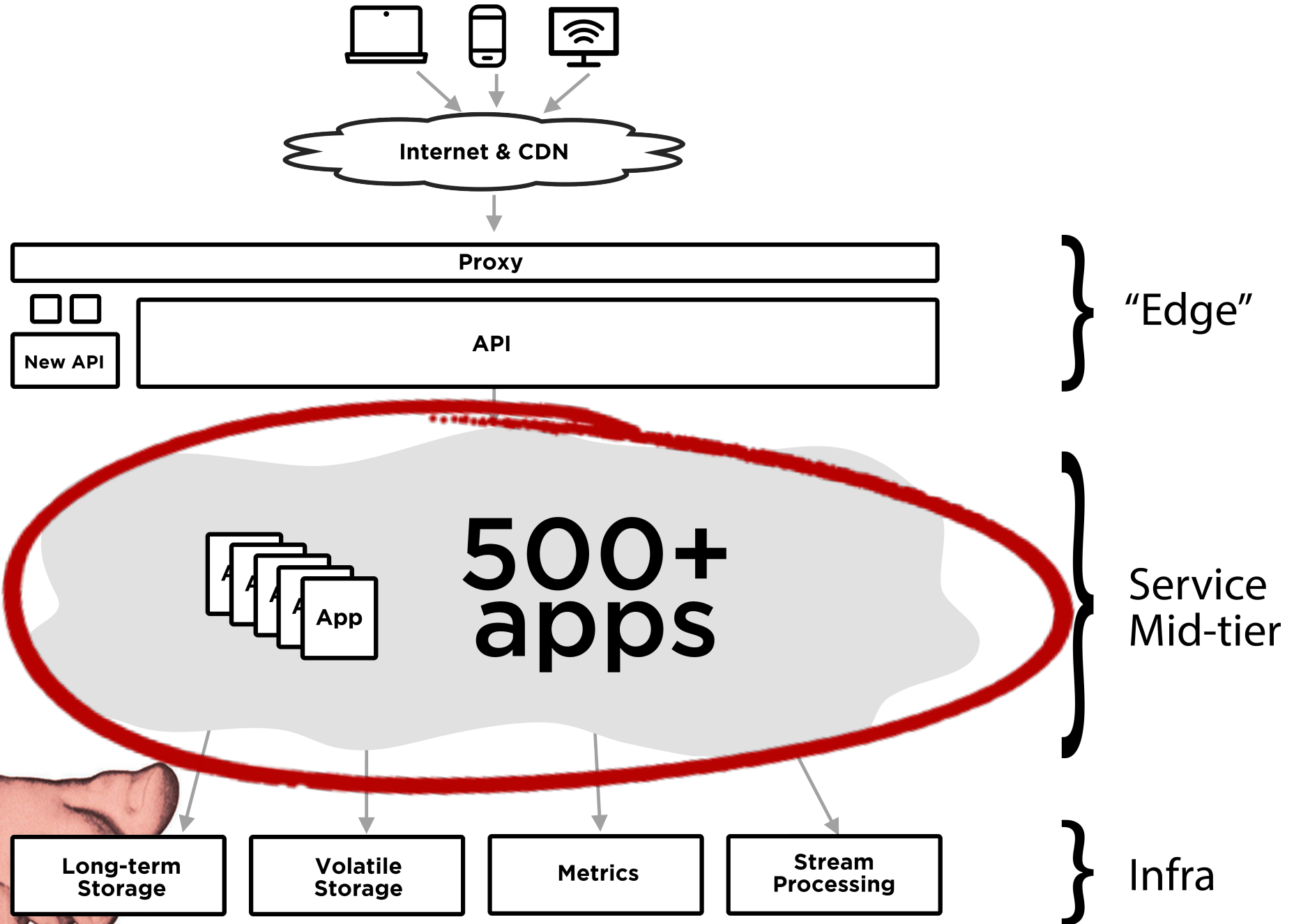
It was the best of
times...

(ie: The story of the skeletons in our closet)

About Netflix..

- ▶ **100m+ members**
- ▶ **1000+ developers**
- ▶ **190+ countries**
- ▶ **1/3 US download traffic**
- ▶ **500+ microservices**
- ▶ **Over 100,000 VMs**

OFFLINE JOBS



Runtime Platform

Enable developers to productively create and integrate software in the Netflix ecosystem.

Major Investments in Platform

High Availability = **Winning
Moments of
Truth**

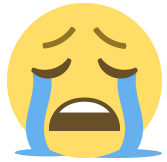
High Availability =



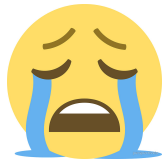




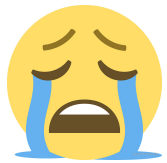
Challenges:



Hard to take advantage of evolving best practices



Owning client-side logic is complex and stressful



Non-Java experience is hard

Challenges:





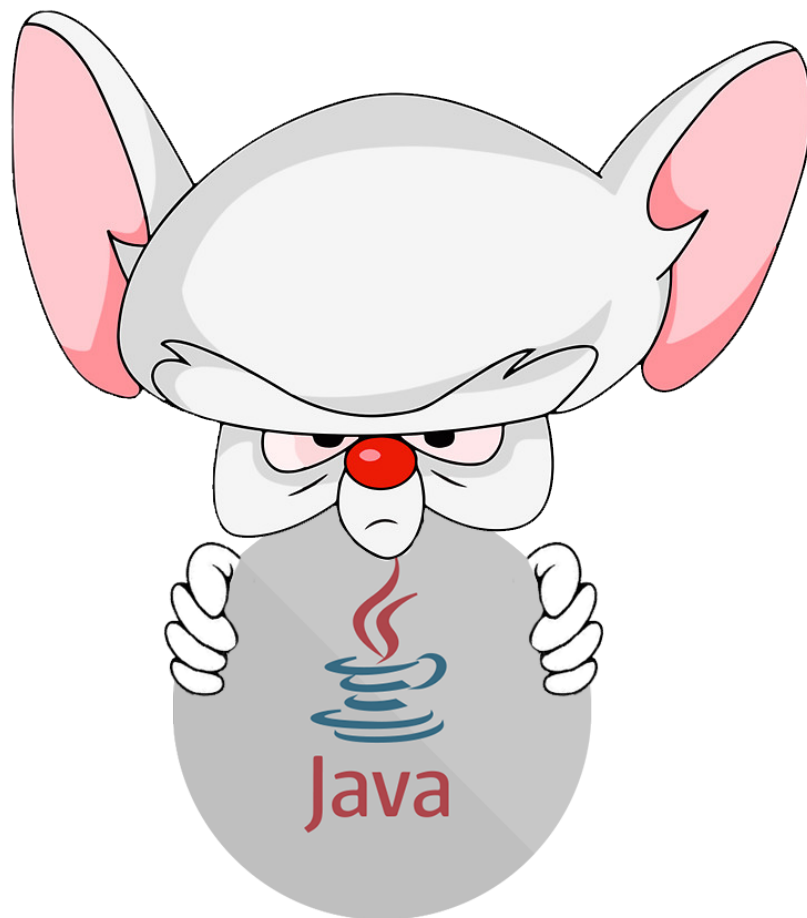
Productivity++

(availability is table stakes)

especially in clients



**Complexity is the
mind killer.**



Runtime Platform

Enable developers to productively create and integrate software in the Netflix ecosystem.

2

Best-practices made easy

(Better living through less complexity)

Generators

Generators

What:

Gives you a deployed app on the “paved road” in minutes.

Generators

Why:

**To make it easy to adopt,
understand, and build
production-ready apps.**

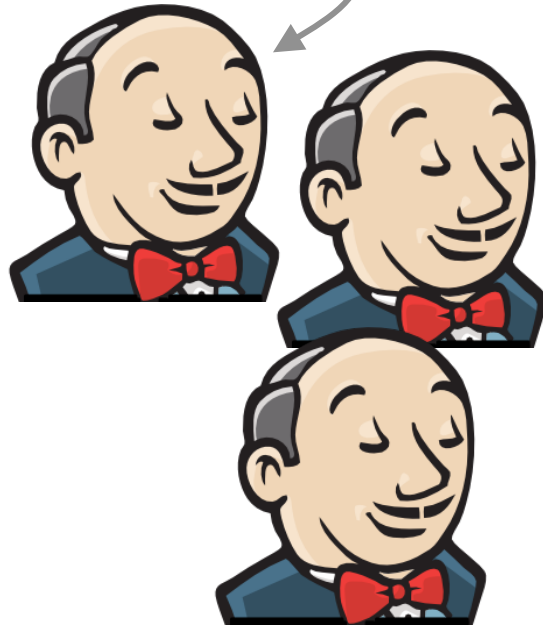
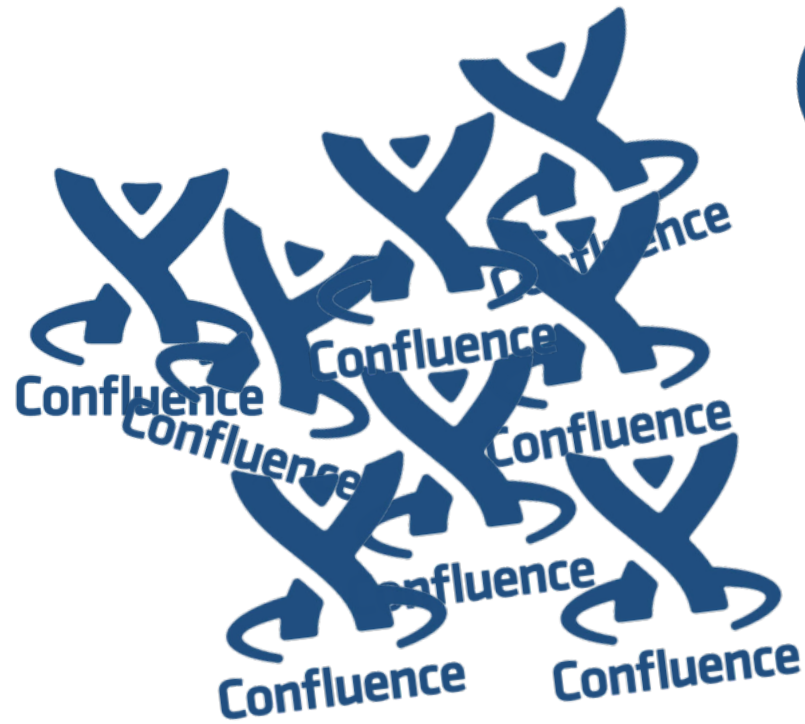


+

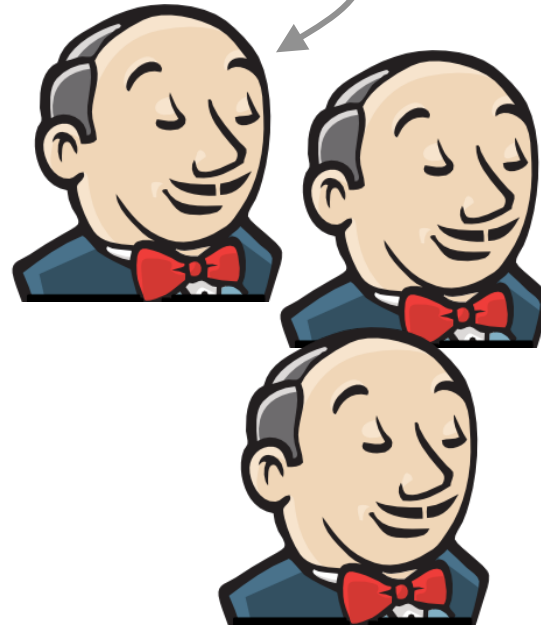
**Best
Practices**

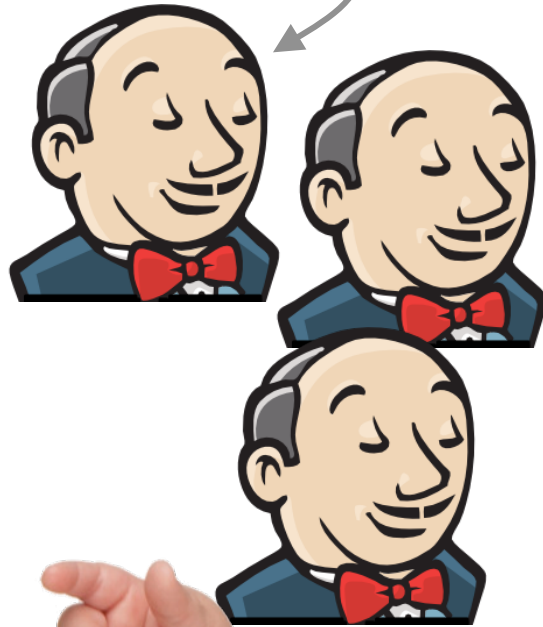
Historically:

“Let’s go!”



Push to
prod!!!?





With Generators:

“Let’s go!”

```
tbozarth:hellworld$ █
```

Source

master ▾



generatortest /

+ New File

generator-test-client-guice

generator-test-proto-definition

generator-test-server

src/main

gradle/wrapper

.gitignore Initial commit

.nf-project.json Initial commit

build.gradle Initial commit

dependencies.lock Dependencies updated and locked by https://buildstest.builds.test.netflix.net/job/USERS-tbozarth-generatortest-update-dep

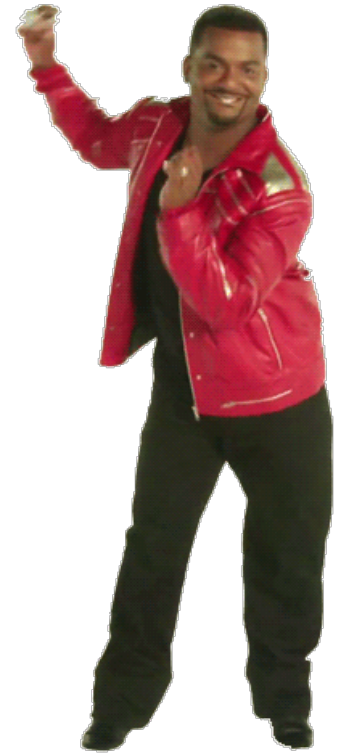
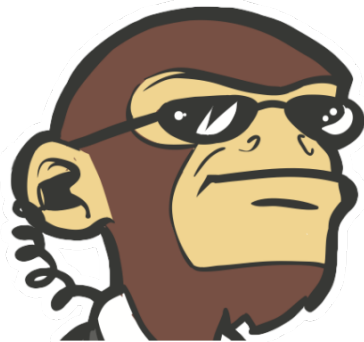
gradle.properties Initial commit

gradlew Initial commit

gradlew.bat Initial commit

README.md Initial commit

settings.gradle Initial commit



**But wait! There's
more!**

(Consistency)

Meta-note

Components != PaaS

3

**Goodbye
hand-written
client libraries**



@Netflix

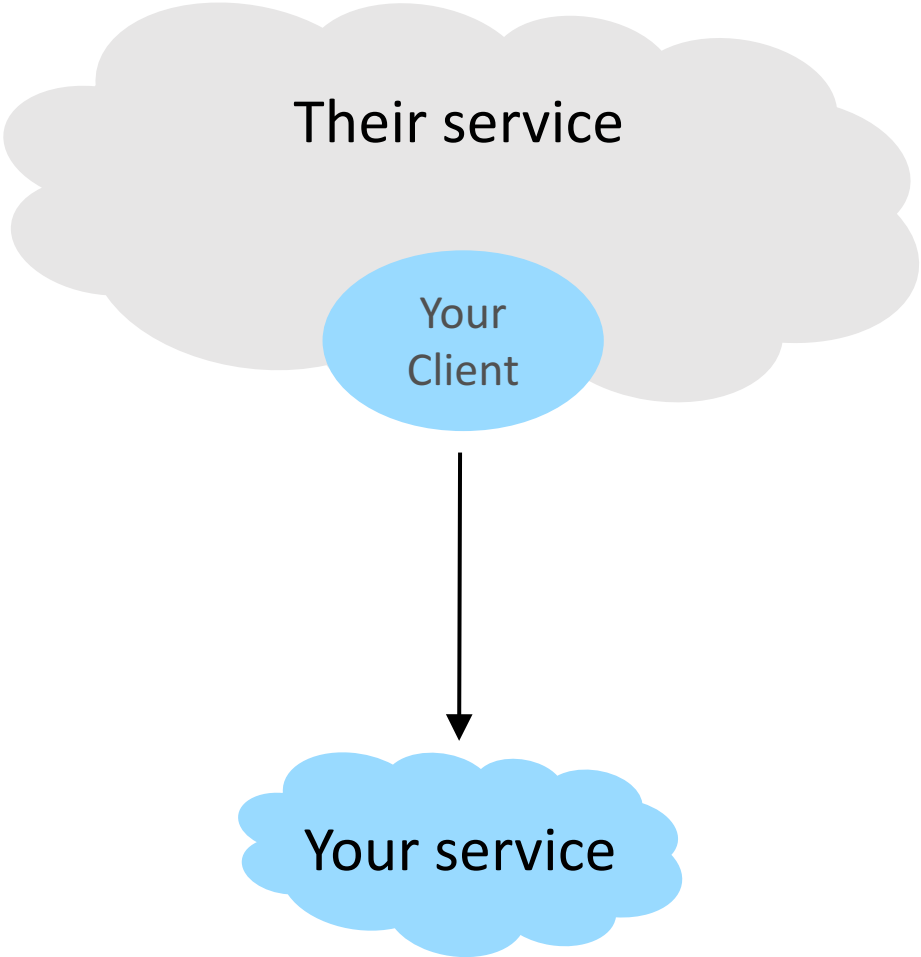
**every service owner
is responsible for a
client**

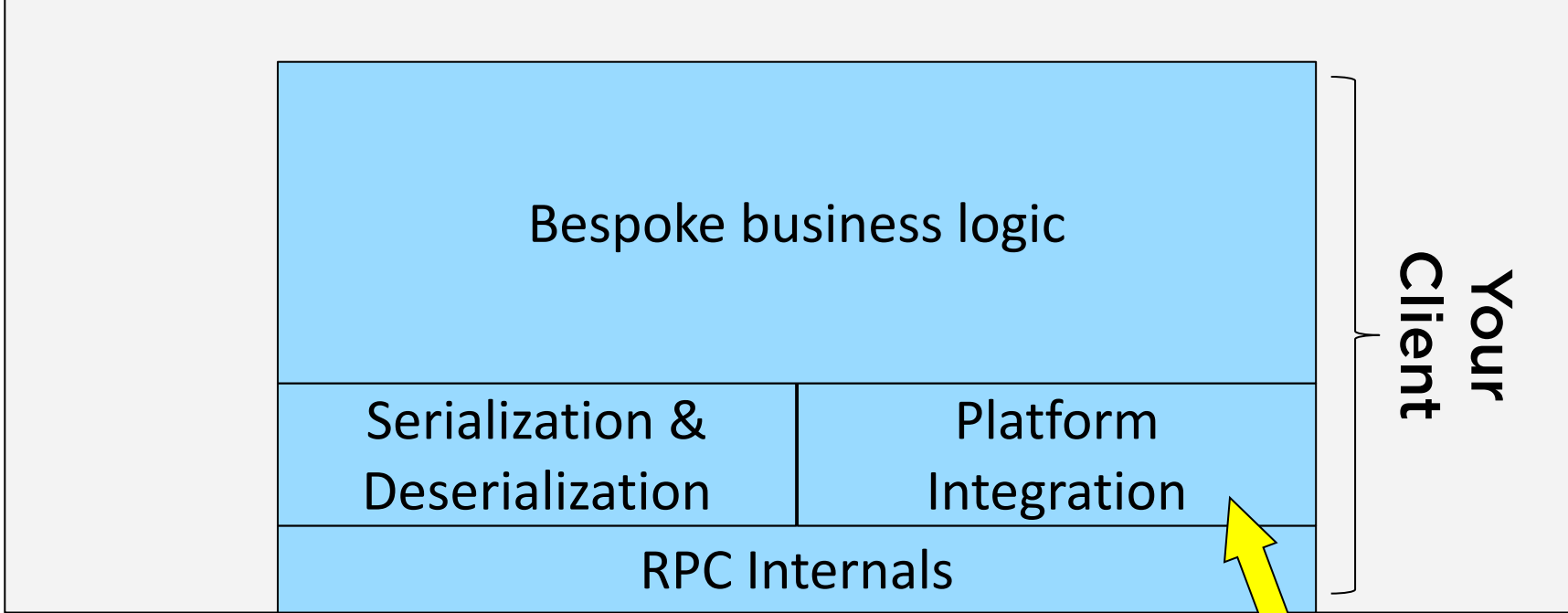


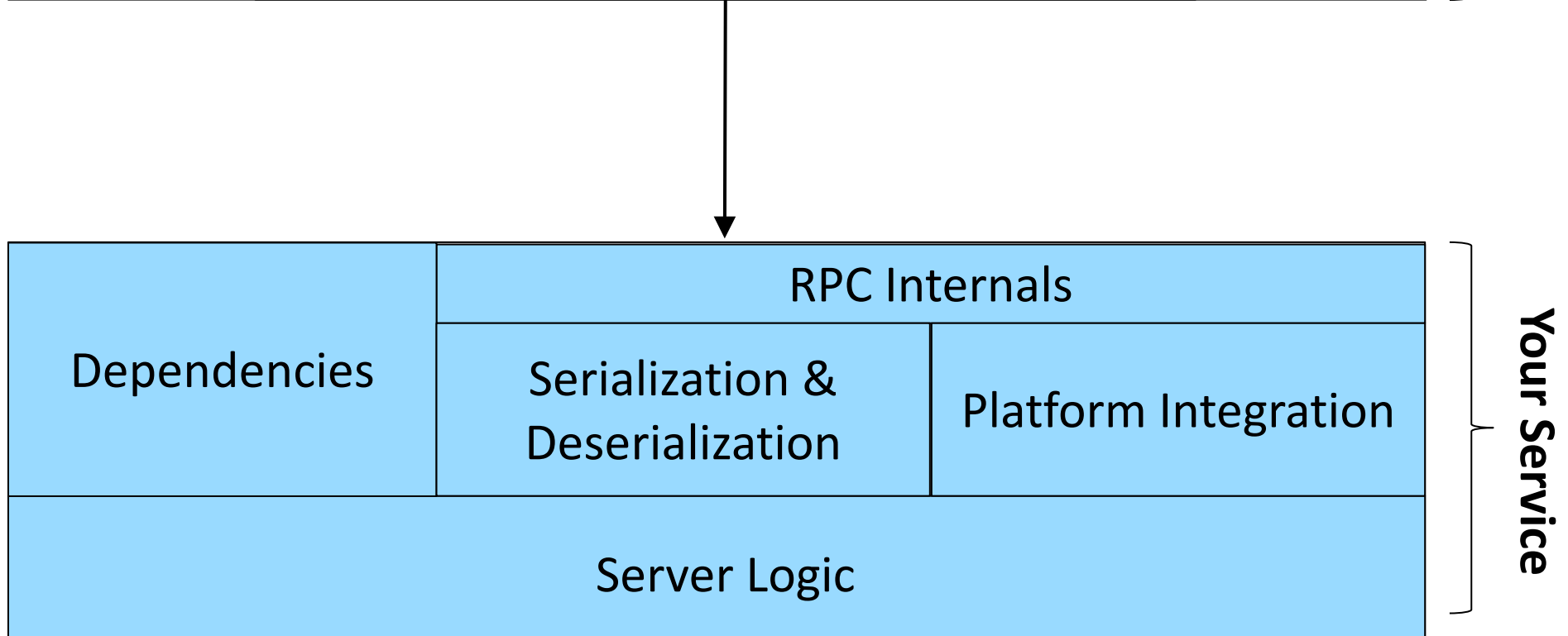
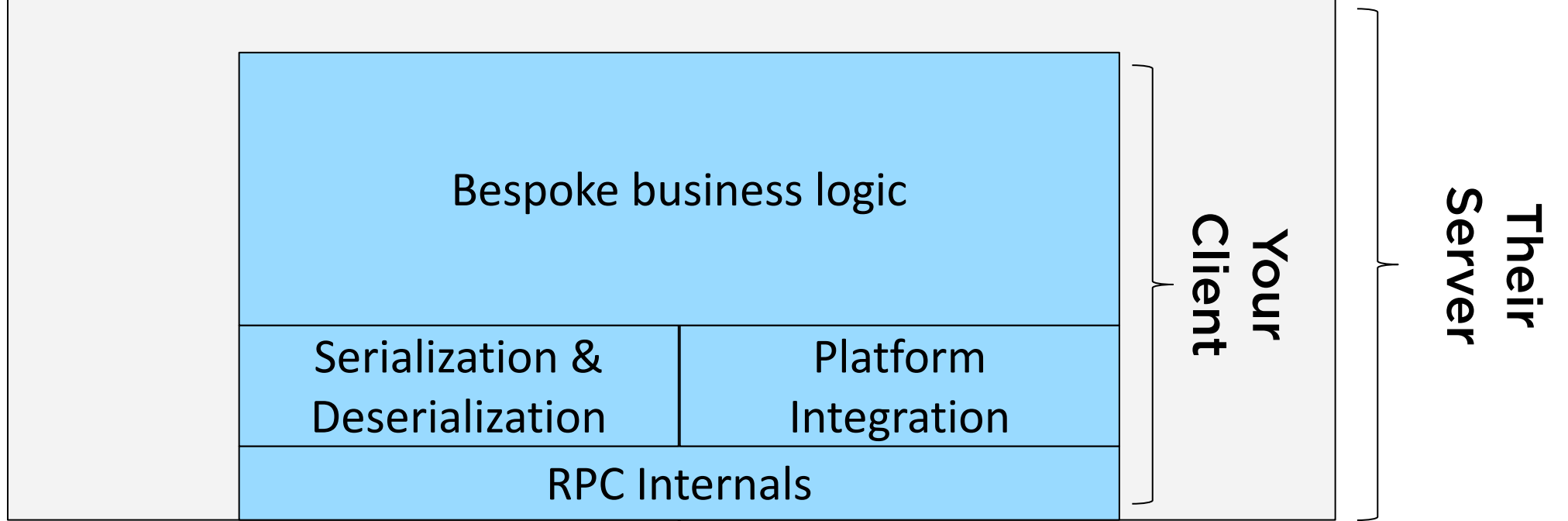
Clients defend themselves from failure

(and the foundation to much of Netflix's micro-service success)

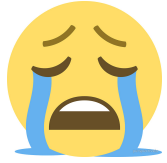








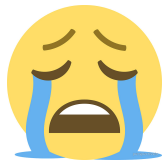
Problems



Server-API changes are a nightmare



So much hand-written RPC-related code



No cross-language client story

**These are solvable
problems**

protobuf
Protocol Buffers

+

GRPC

protobuf
Protocol Buffers



**To strictly define the
interaction model**

← GRPG →

**To seamlessly
integrate into the
broader ecosystem**

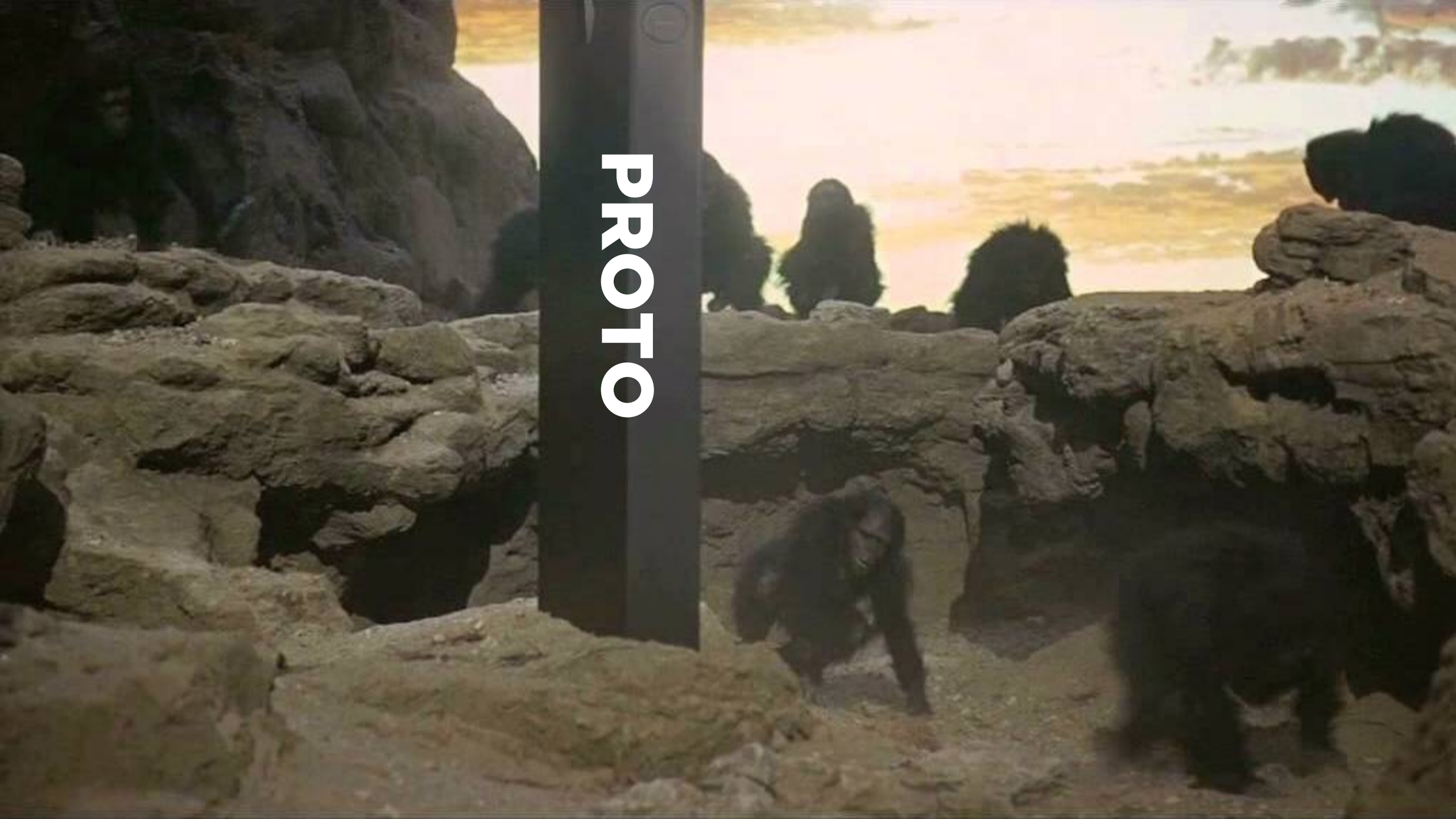


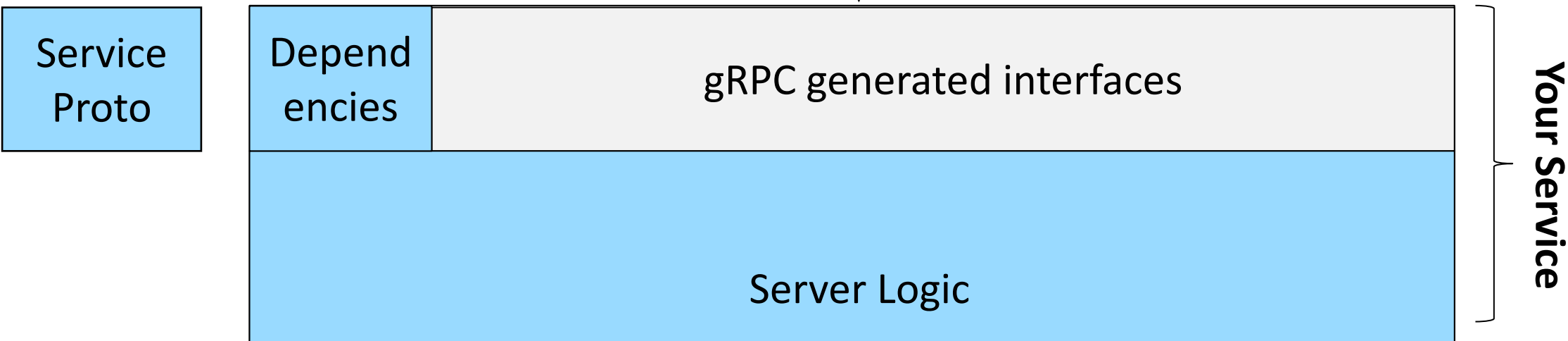
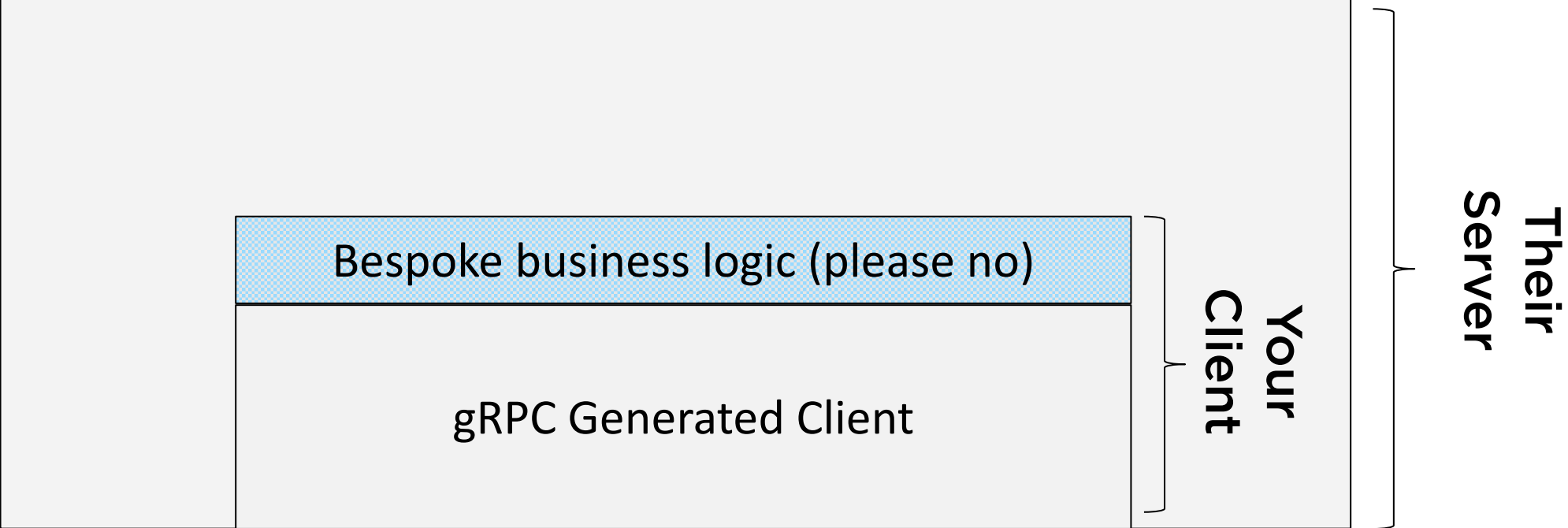
2 big wins:

Code Generation

New Abstraction Layer

PROTO





atching, Circuit-breaker
Fallbacks, Failure
Injection, Discovery,
request-context-tracing
etrics, Retries, Hedge
Requests. oh my!

Interceptors!

Interceptors encapsulate common patterns

(outside the user's typical concern domain)

Client Defense Examples:

- **Fallbacks**
- **Advanced Caching**
- **Retries**
- **Failure Injection**
- **Hedged Requests**
- **Circuit Breakers (Hystrix)**
- **Common analytics & event-logs**
- **... and much more**

Complex,
multi-tier
caching
took a **lot** of
code.

```
// If in forced failback mode return a failback
if(!shouldBeOnline().forceFailback(previousID)) {
    try {
        return CommonUtils.createForceFailback(previousID);
    }
    catch(Exception e) {
        throw new SubscriberClientException(e);
    }
}
// Force - top Request Context
key {
    Account instanceAccount = RequestContextUtil.getClient().getFailbackAccountFromRequestContext(ServerRequestContext.get().previousID);
    // Check this
    if(instanceAccount != null) {
        num.set(1); num.set(1); context.setContext(SubscriberClientContext.create(CONTEXT_PREFIX + "get_amount_from_cc_account");
        return instanceAccount;
    }
} catch (SubscriberClientException e) {}
logger.error(SubscriberClientContext.create(DOUBLE_PREFIX + "get_new_token_success_in failback");
throw new SubscriberClientException(e);
}
}
}

// If failback mode is null;
// If no callID is null;
key {
    boolean isCallIDValid = validateRequestContextParameters();
    boolean isCallIDValid = validateRequestContextParameters();
} catch (SubscriberClientException e) {
    logger.error(e);
    throw new SubscriberClientException("Unable to build the URL for customerID " + customerID);
}
}

final long startTime = System.currentTimeMillis();
boolean isOffline = true;
if (CONFIGURABLE_CLIENT_CACHE_ACTIVE.get() == false == null) {
    // make sure the call is not null.
    boolean isOffline = validateRequestContextParameters();
    boolean keyId = getCallID().getOrDefault();
    long responseId = null;
    try {
        responseId = SubscriberClientContext.getLogForKey().getOrDefault(keyId, "0000");
    }
    catch(SubscriberClientException e) {
        if(!shouldBeOnline().forceFailback(previousID)) {
            logger.error(e);
            return null;
        }
    }
} catch (SubscriberClientException e) {
    logger.error(e);
    return null;
}

/** If the cache is enabled and Account lookup returned null = key ID lookup */
if (CONFIGURABLE_CLIENT_CACHE_ACTIVE.get() == false == null) {
    try {
        responseId = SubscriberClientContext.getLogForKey().getOrDefault(keyId, "0000");
    }
    catch(SubscriberClientException e) {
        if(!shouldBeOnline().forceFailback(previousID)) {
            logger.error(e);
            return null;
        }
    }
}
}

if (responseId != null) {
    final Account acct = createValidatedKey(responseId, Account.class, validateContext());
    context.setContext(SubscriberClientContext.create(CONTEXT_PREFIX + "EXTERNAL_CACHE_HIT");
    context.setContext(SubscriberClientContext.create(CONTEXT_PREFIX + "EXTERNAL_CACHE_HIT"));
    // This is not a long call. Not just adding this just in case somebody is checking for it
    this.setRequestCode = 0;
    num.set(1); num.set(1); context.setContext(SubscriberClientContext.create(CONTEXT_PREFIX + "get_subscriber_by_pid_return_cache");
    return acct;
}
}

// If in new long key
final SubscriberContext initializedAccount = createValidatedKey(responseId, SubscriberContext.class, validateContext());
context.setContext(SubscriberClientContext.create(CONTEXT_PREFIX + "EXTERNAL_CACHE_HIT");
context.setContext(SubscriberClientContext.create(CONTEXT_PREFIX + "EXTERNAL_CACHE_HIT"));
} catch (SubscriberClientException e) {
    logger.error(e);
    return null;
}
}
```

```
option (com.netflix.proto.options.cacheable) = {  
  key: "prefix-#{customer_id}"  
};
```

(In proto)

```
configurer  
  .register(EvCacheChannelFeature.class)  
  .register(GuavaCacheChannelFeature.class)  
  .register(RequestVariableCacheChannelFeature.class)
```

(In client
config)

Whaaaaa? 7 lines!?

gRPC ❤️ languages!

C++

Java

Python

Go

Ruby

C#

Node.js

Android Java

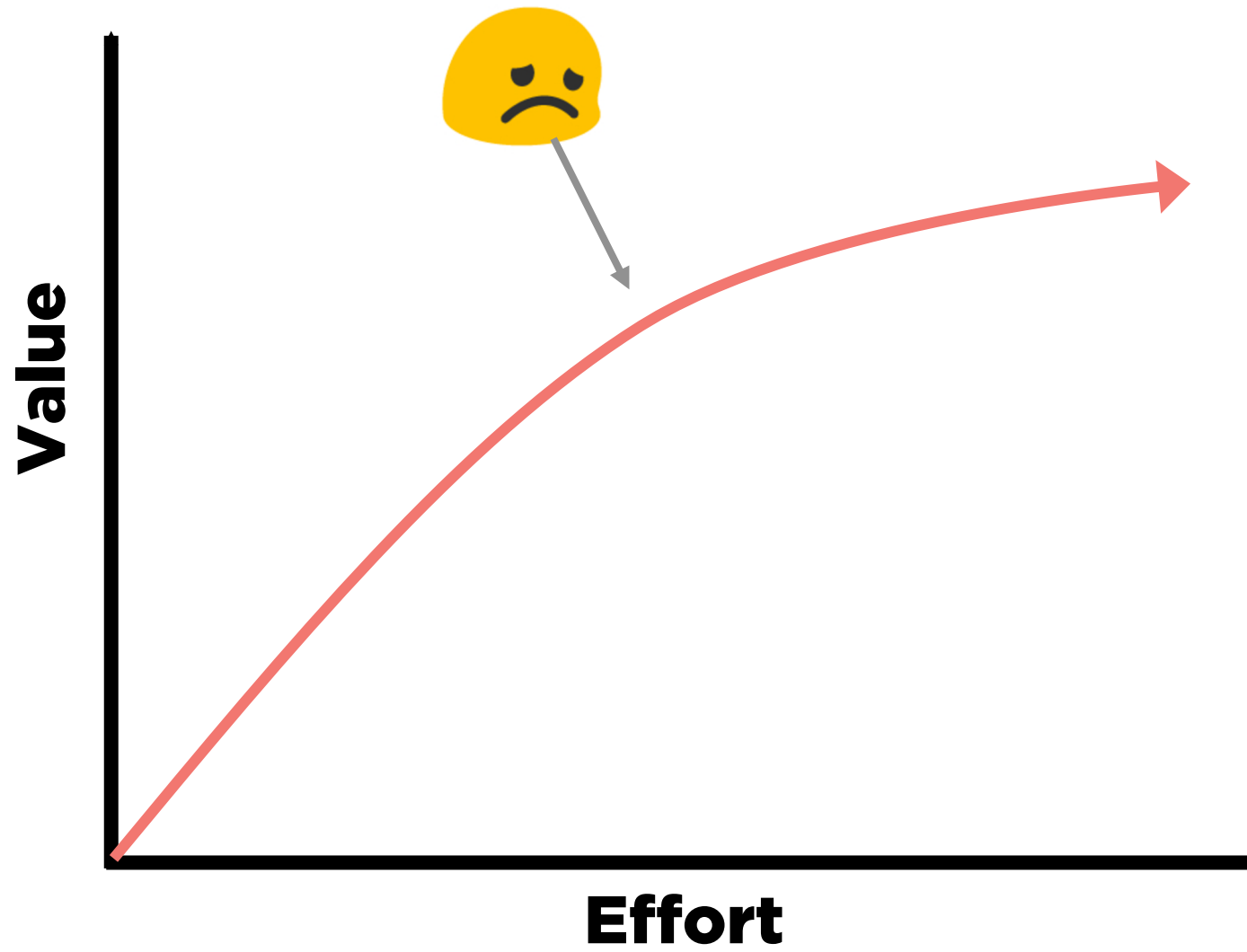
Objective-C

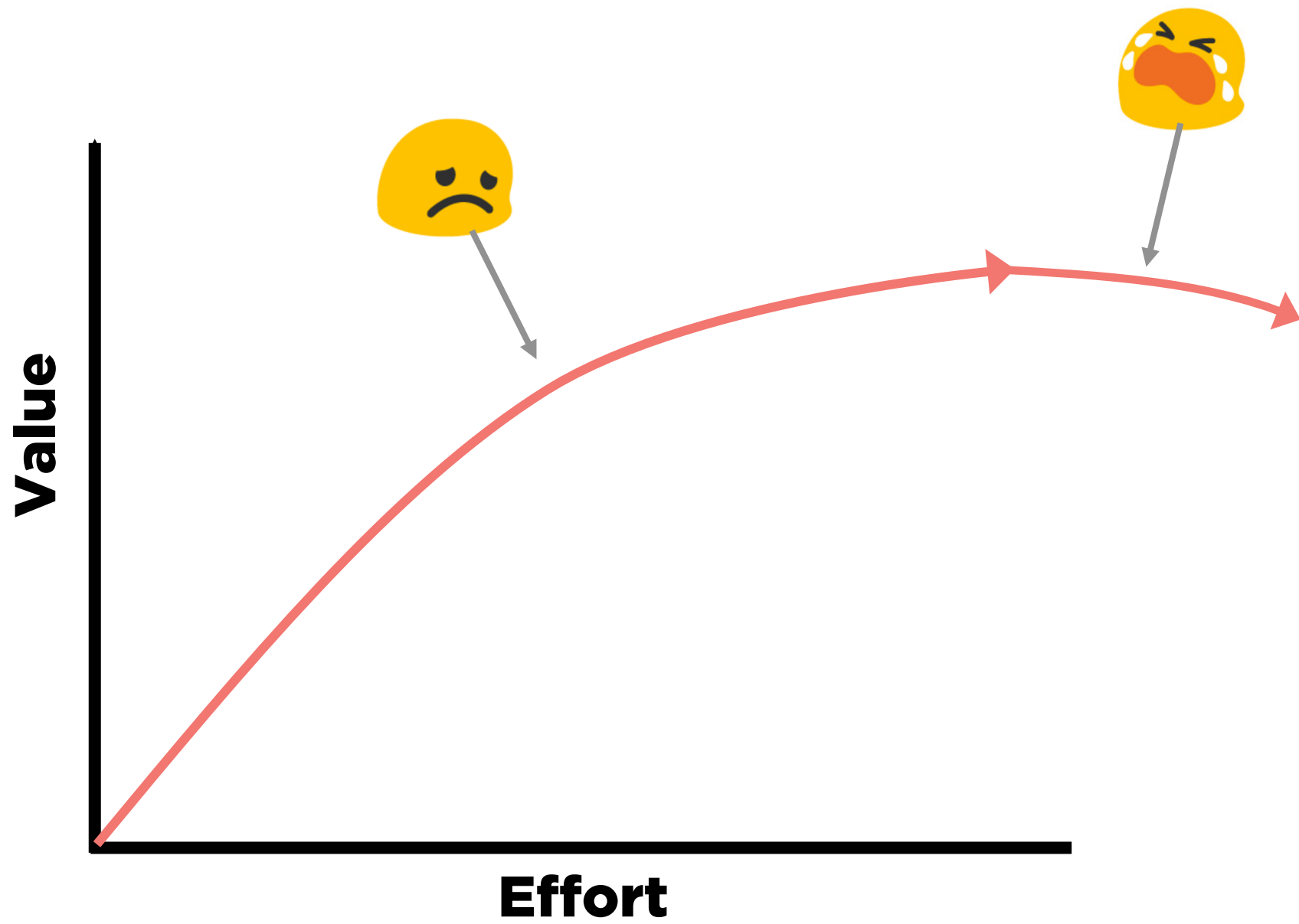
PHP

4

NIH → OSS

OBVIOUS







**With every step
comes the decision
to take another.**



**Inertia is a powerful
force, and a terrible
strategy.**

**Favor commodity
when it's not our
core competency**

(oh right! AWS!)



Wrapping up...

Everything discussed is done

- ▶ gRPC = 10%+ of Netflix RPC
- ▶ 800+ projects made with generators
- ▶ 100+ services currently deployed from generators
- ▶ This stuff = Default for 6-12 months



Code generation is the short & long term solution



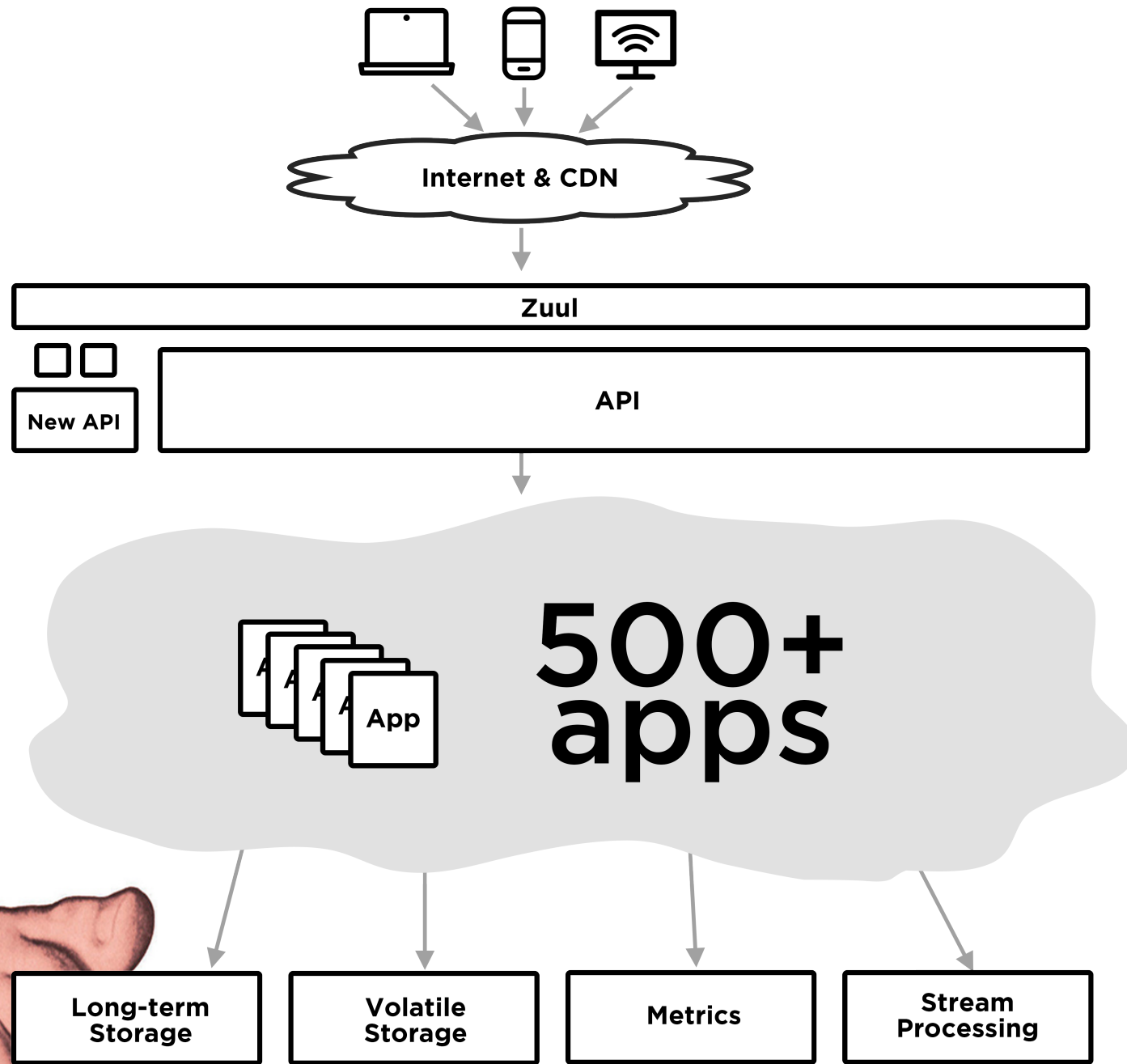
IDLs = micro-services' best friend



Don't build stuff you don't need to

<Appendix>

OFFLINE JOBS



} "Edge"

} Service
Mid-tier

} Infra