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

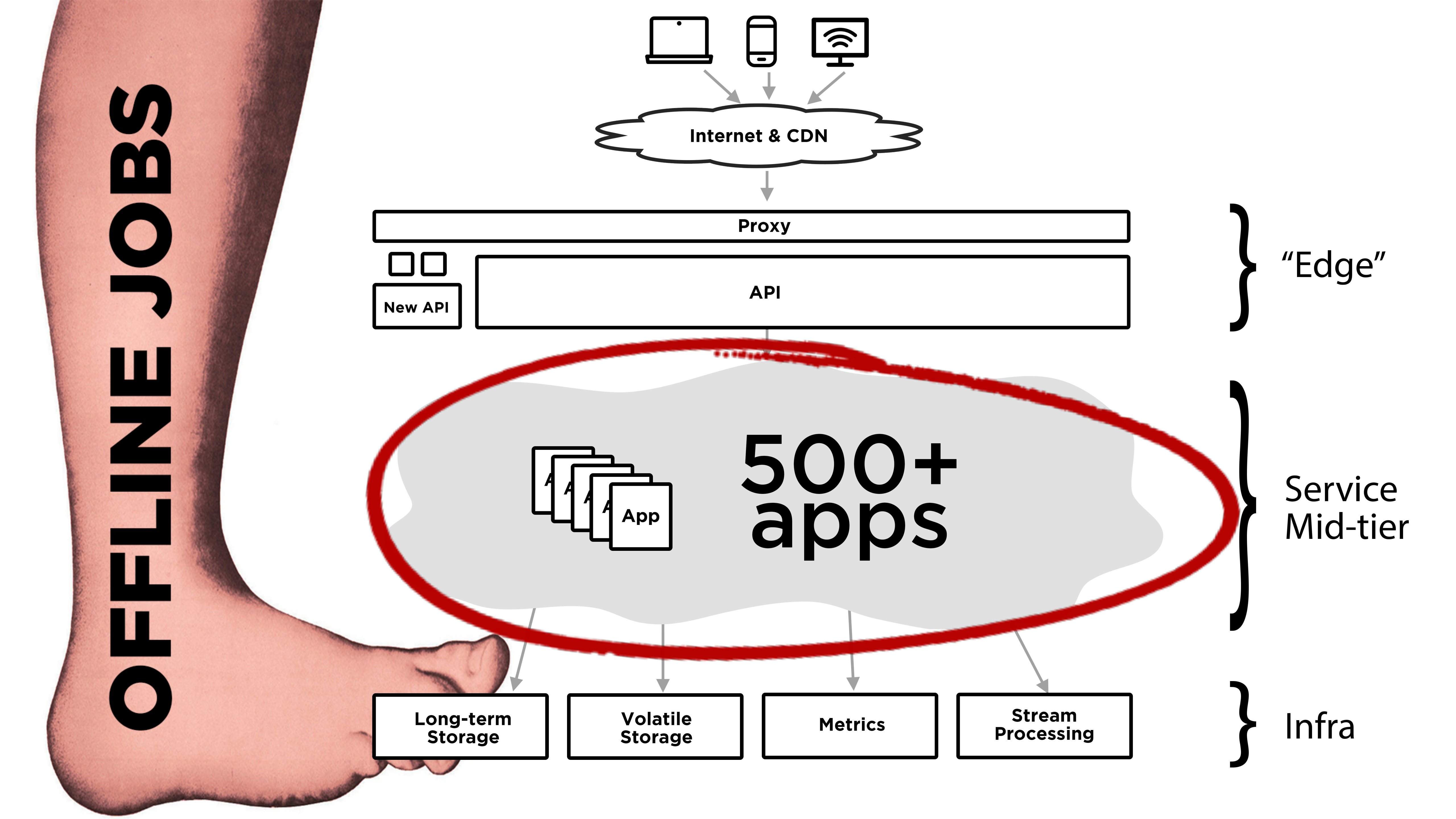
1

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
Internet & CDN
Proxy
New API
API
App
500+ apps
Long-term Storage
Volatile Storage
Metrics
Stream Processing
"Edge"
Service Mid-tier
Infra

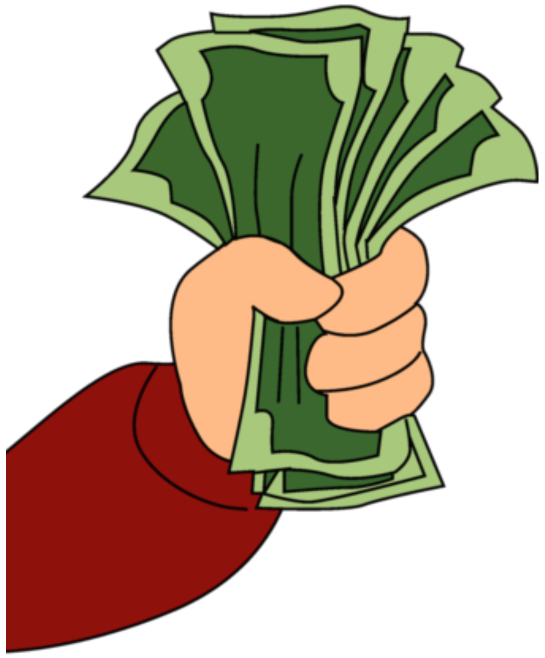
Runtime Platform

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:

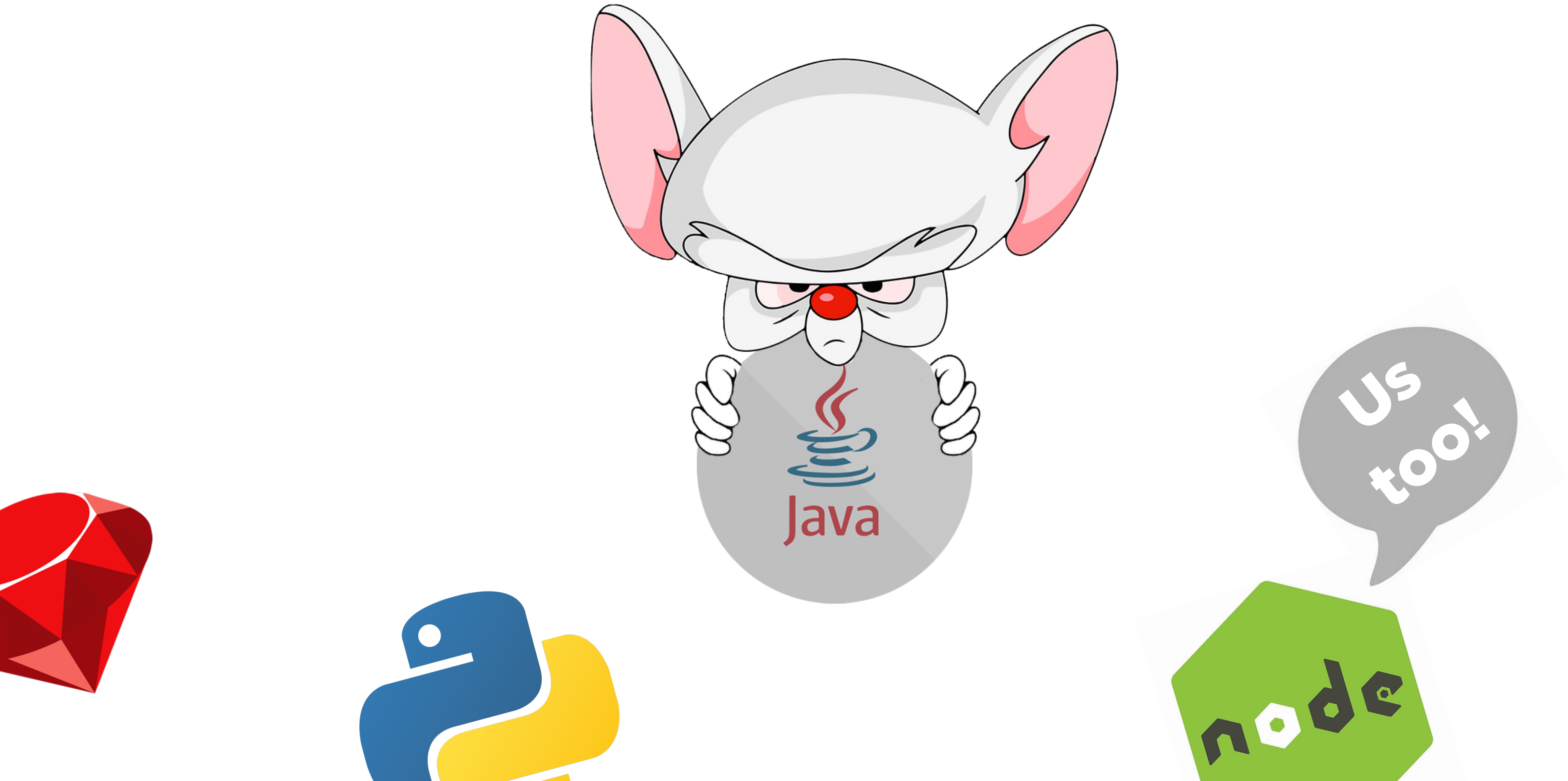
- 😩 Hard to take advantage of evolving best practices
- 😩 Owning client-side logic is complex and stressful
- 😩 Non-Java experience is hard

Challenges:

Productivity++

(availability is table stakes)

Complexity $\overset{\text{especially in clients}}{\wedge}$ is the mind killer.


Java
Us too!
node

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.

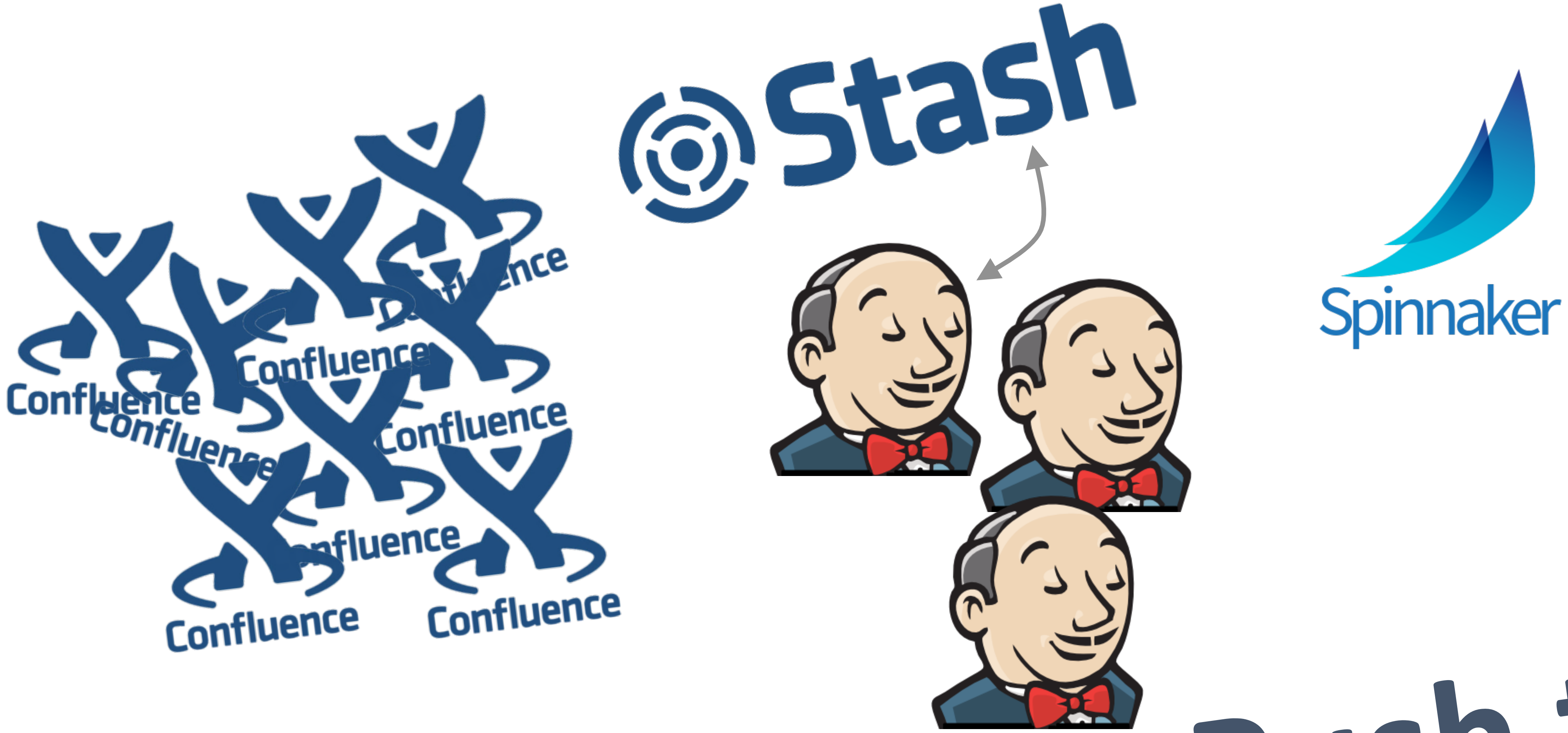
YEOMAN

+

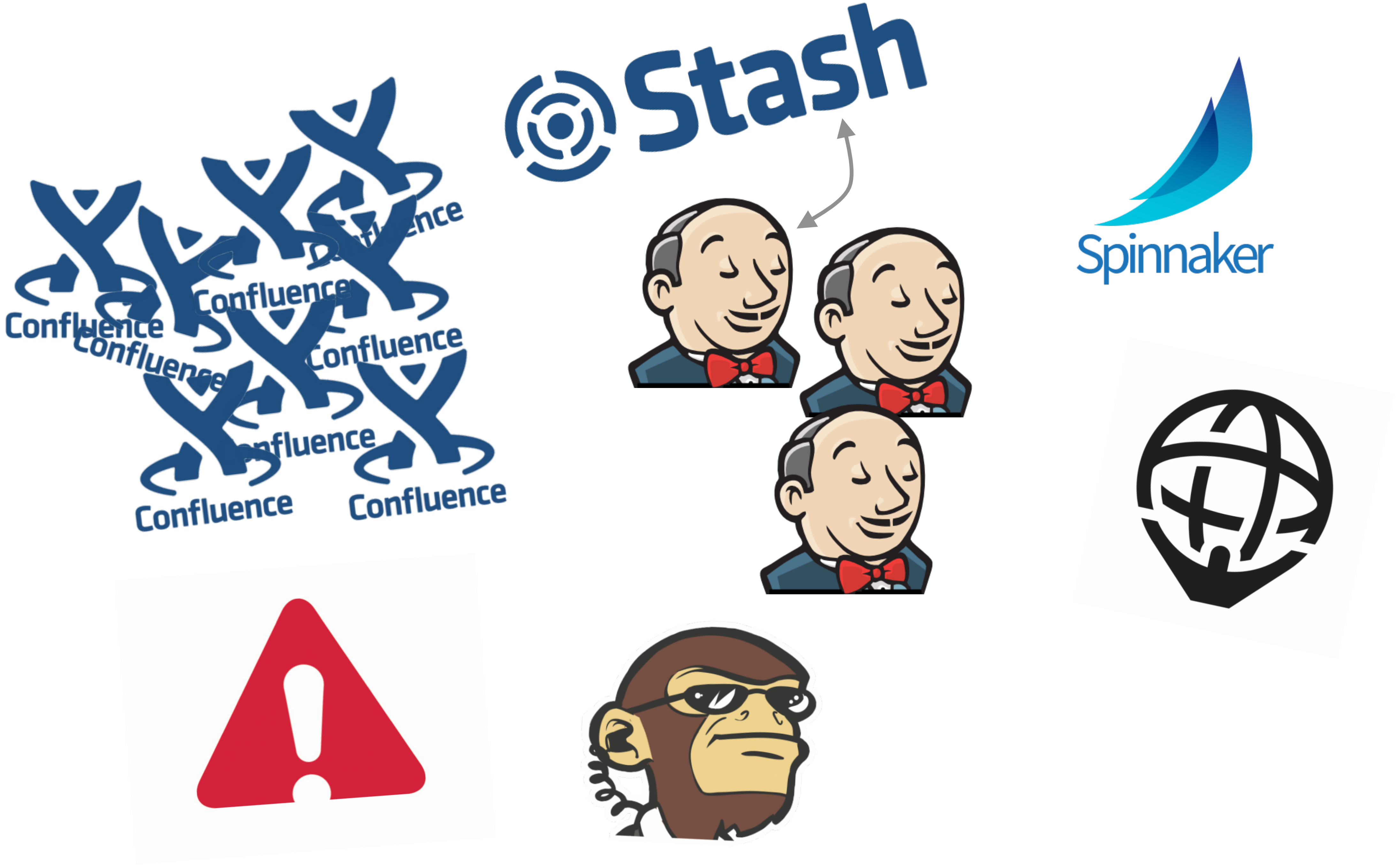
Best Practices

Historically:

“Let’s go!”



Push to prod!!?

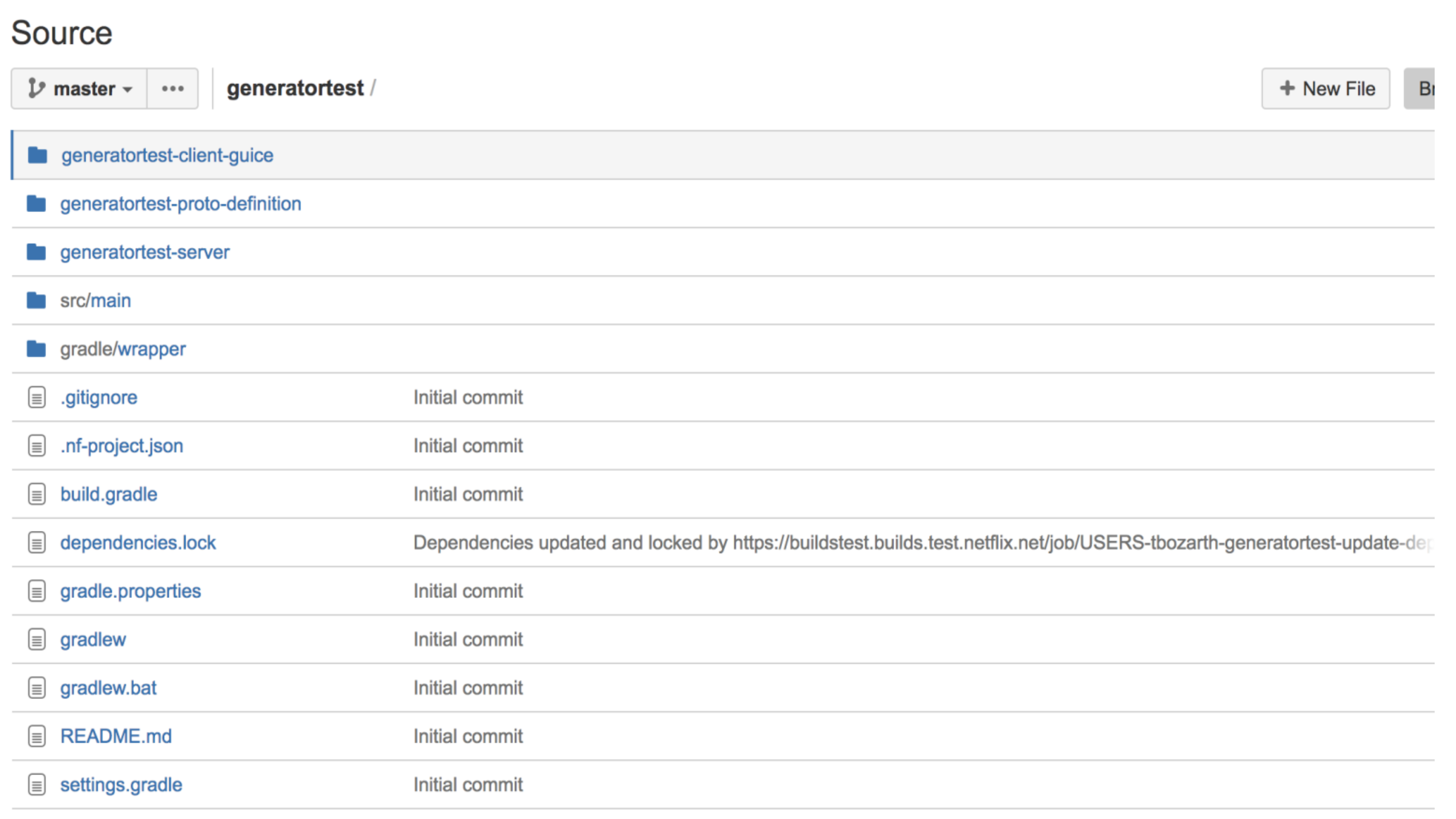

Confluence
Confluence
Confluence
Confluence
Confluence
Confluence
Confluence
Confluence
Stash
Spinnaker

Confluence
Confluence
Confluence
Confluence
Confluence
Confluence
Confluence
Stash
Spi

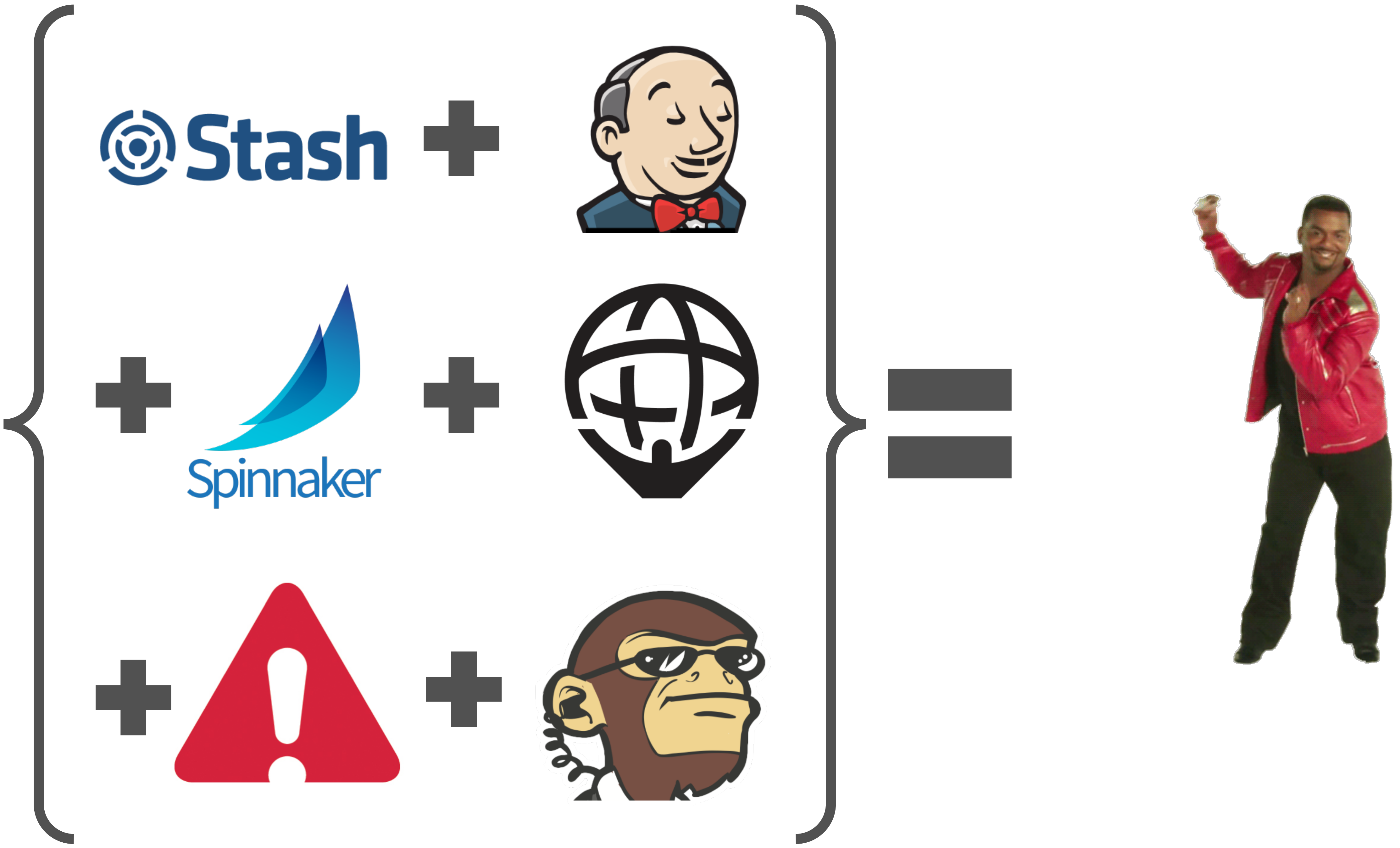
With Generators:

“Let’s go!”

```
tbozarth:helloworld$ 
```



generatortest-client-guice	
generatortest-proto-definition	
generatortest-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-de
gradle.properties	Initial commit
gradlew	Initial commit
gradlew.bat	Initial commit
README.md	Initial commit
settings.gradle	Initial commit


Stash
Spinnaker

But wait! There’s more!

(Consistency)

Meta-note

Components != PaaS

3

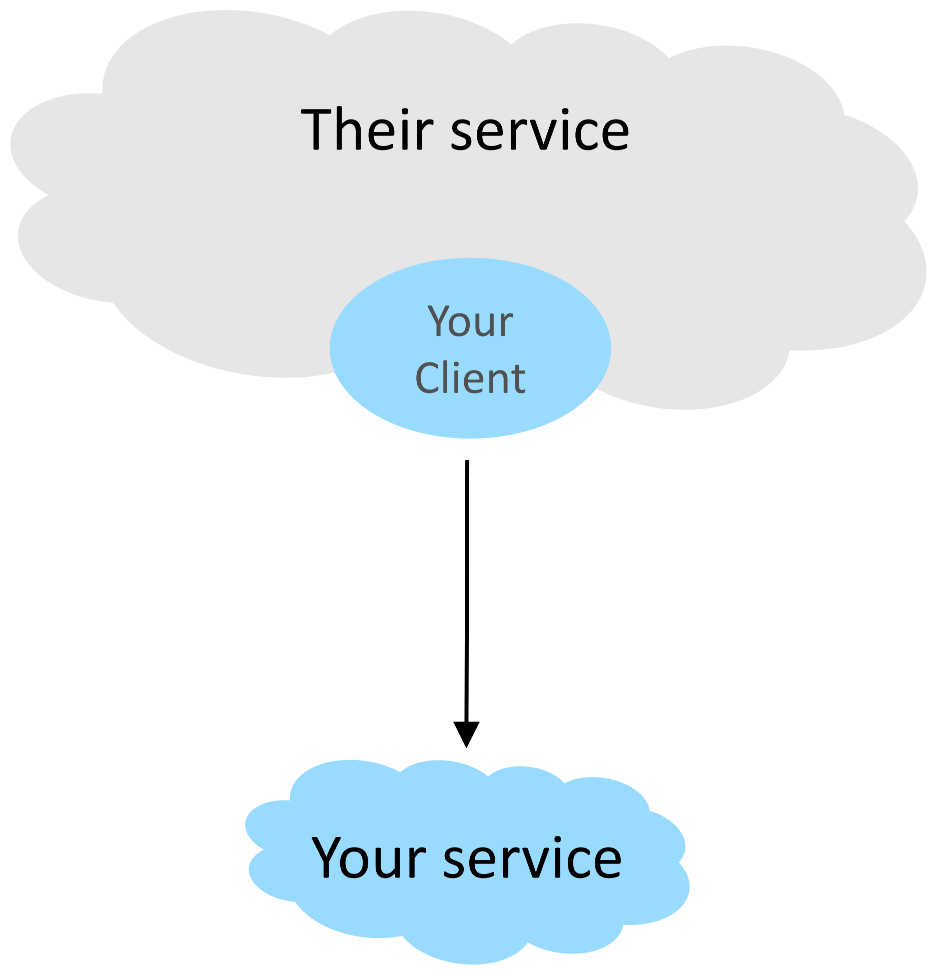
Goodbye hand-written client libraries

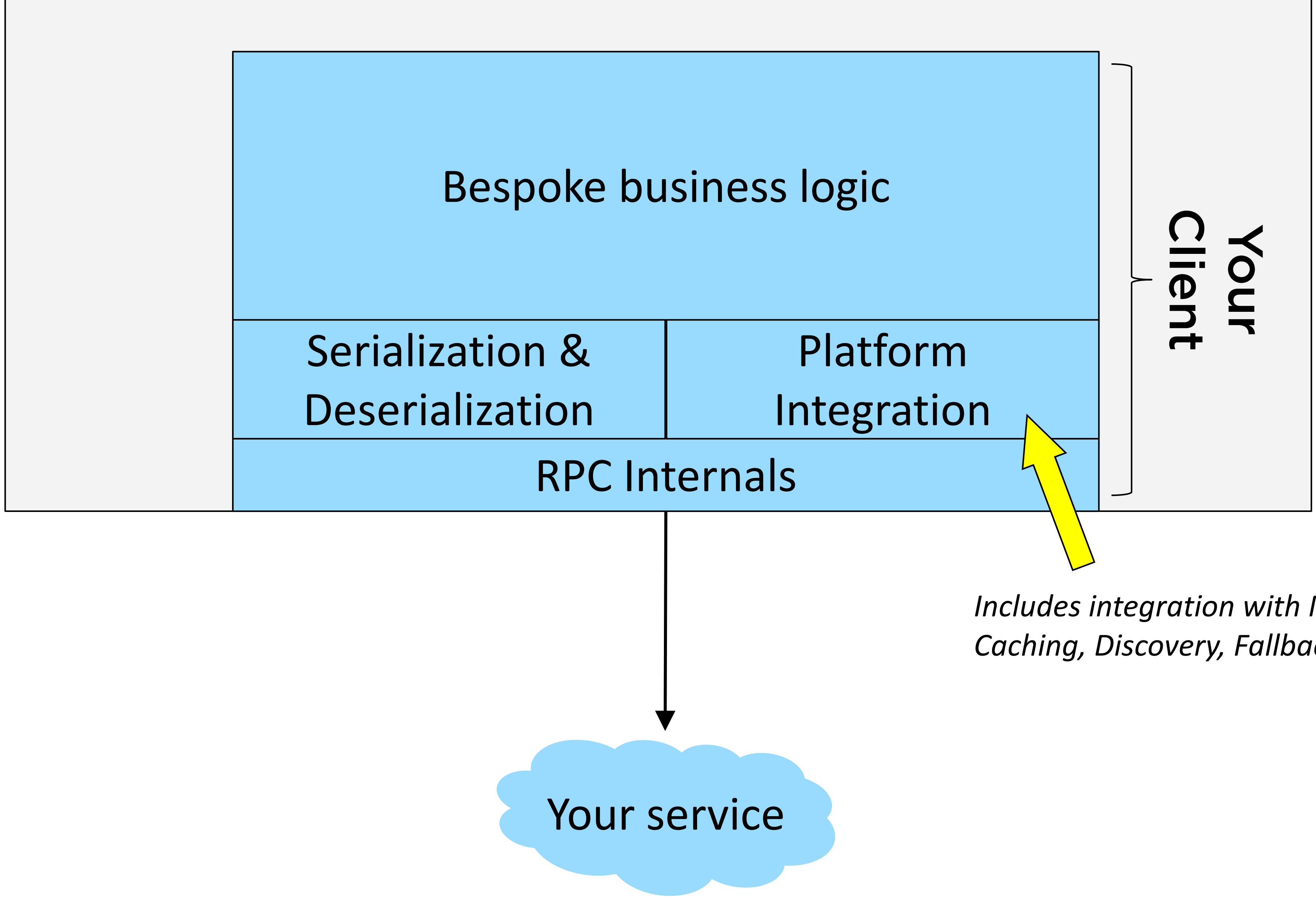
@Netflix

every service owner is responsible for a client

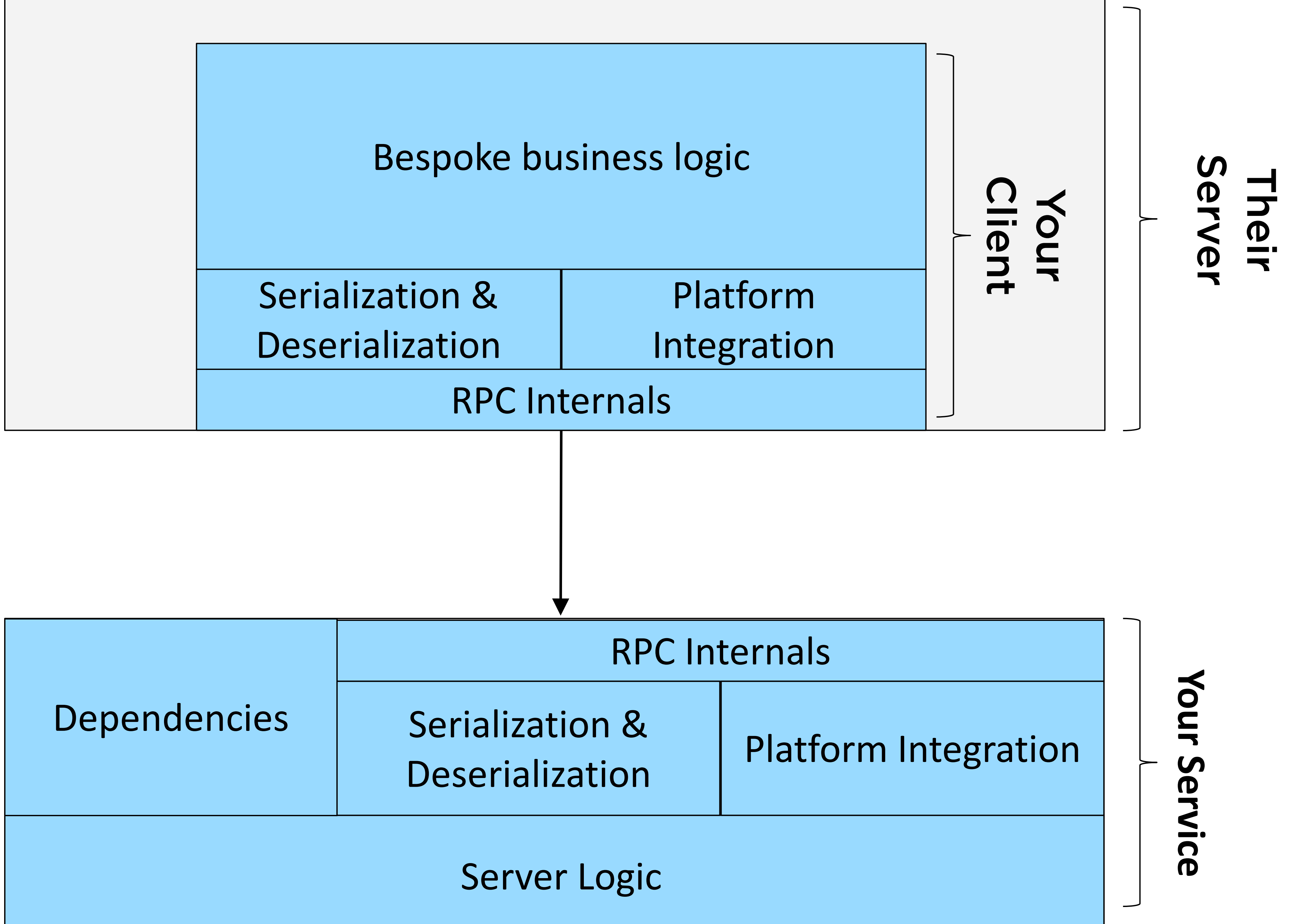
Clients defend themselves from failure

(and the foundation to much of Netflix’s micro-service s[illegible]cess)


Their service
Your Client
Your service



Includes integration with Metrics, Caching, Discovery, Fallbacks, etc...



Bespoke business logic
Serialization & Deserialization
Platform Integration
RPC Internals
Your Client
Their Server
RPC Internals
Dependencies
Serialization & Deserialization
Platform Integration
Server Logic
Your Service

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

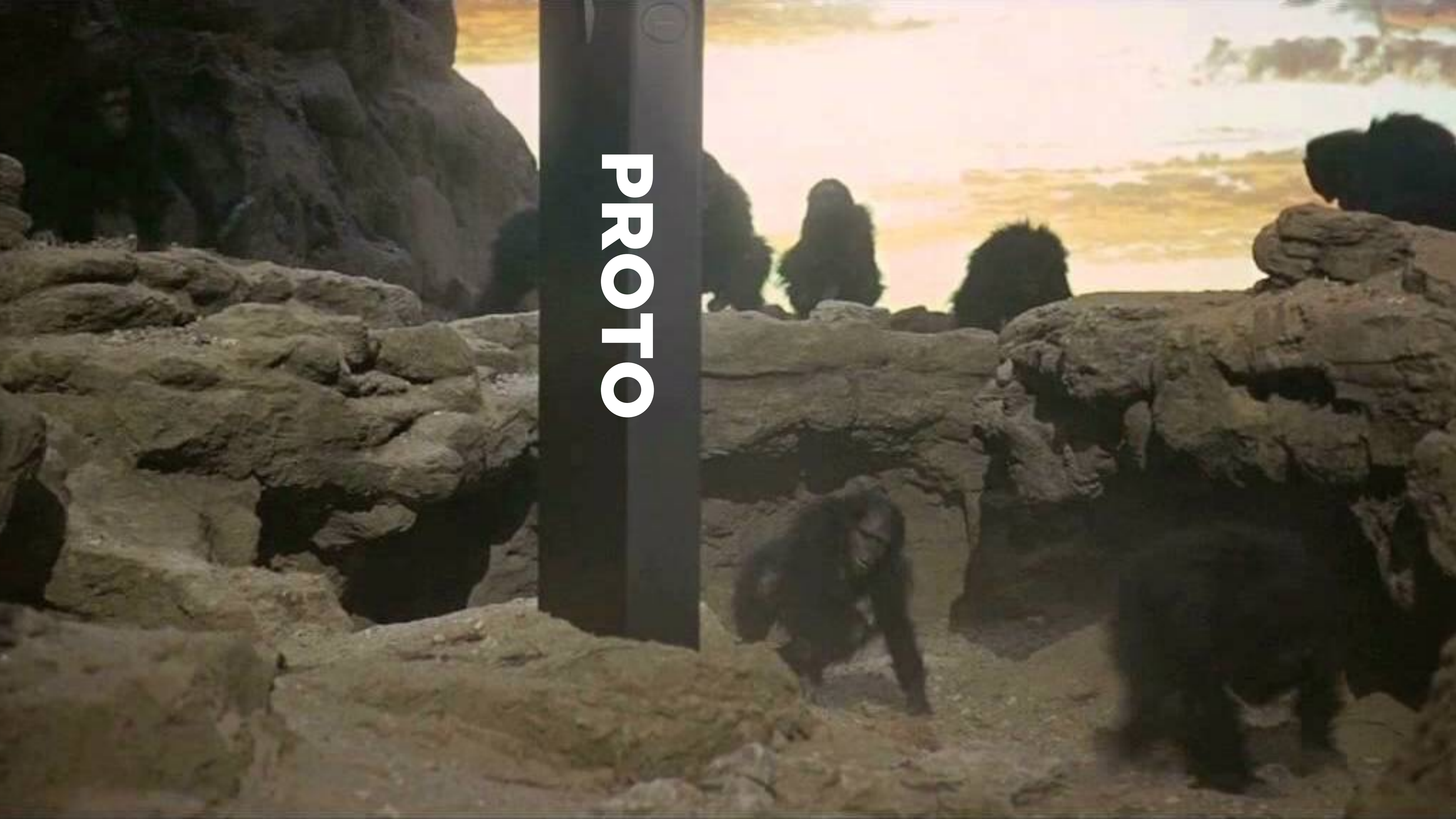
GRPC

To seamlessly integrate into the broader ecosystem

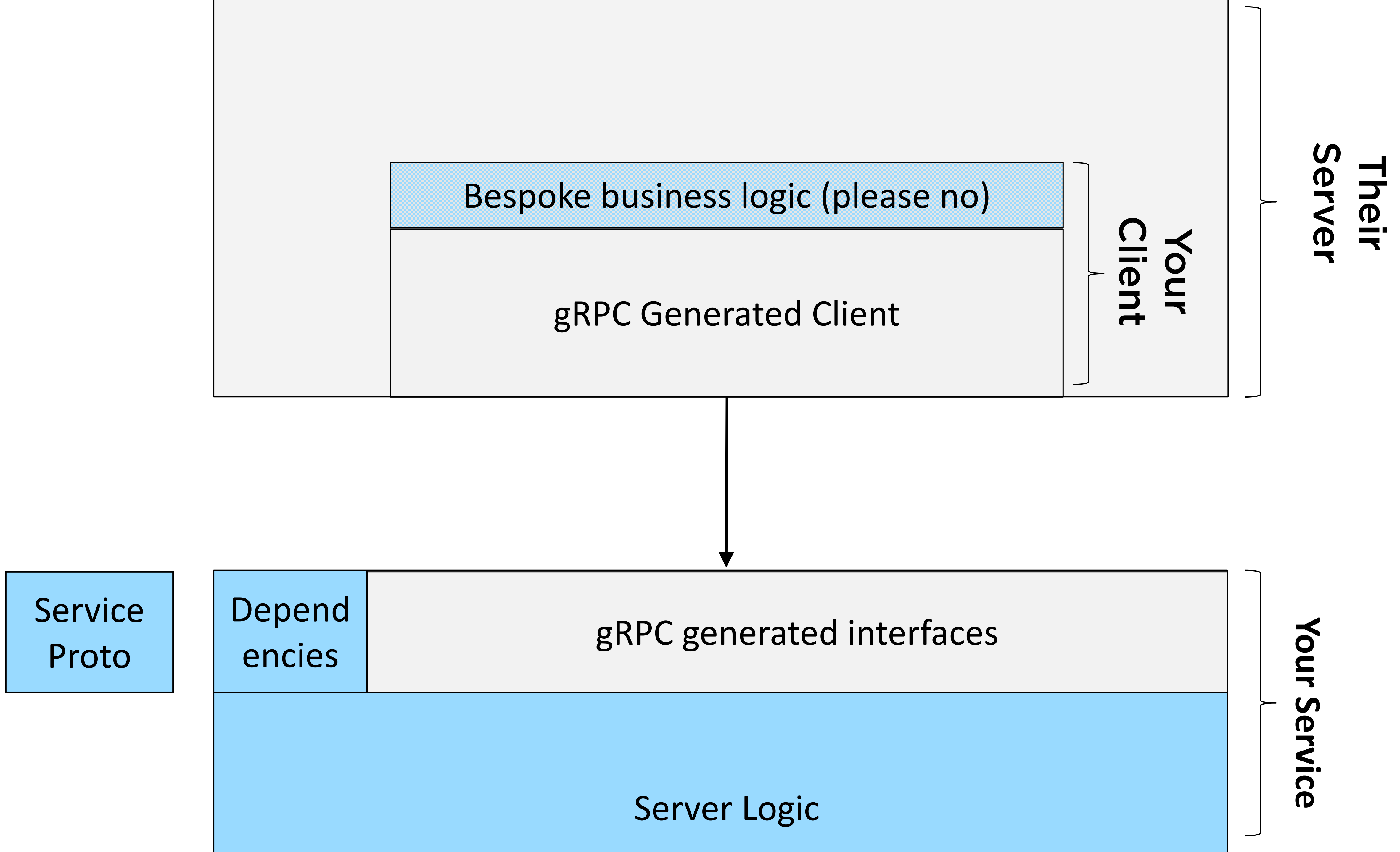
2 big wins:

Code Generation

New Abstraction Layer



PROTO


Bespoke business logic (please no)
gRPC Generated Client
Your Client
Their Server
Service Proto
Depend encies
gRPC generated interfaces
Server Logic
Your Service

ching, Circuit-breake
Fallbacks, Failure
Injection, Discovery,
equest, Context-tracin
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.

```
[illegible]
```

```
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!?

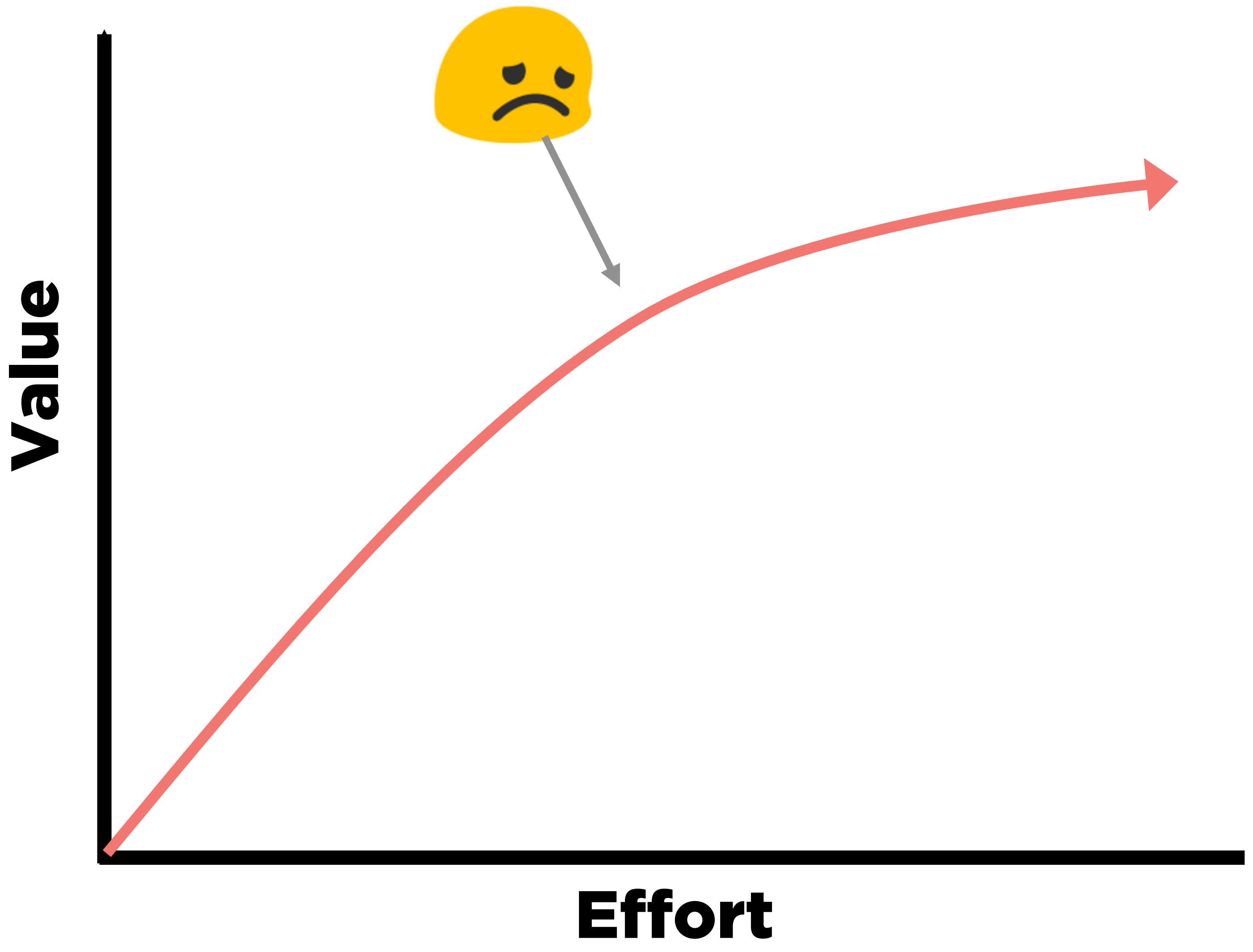
gRPC ❤️ languages!

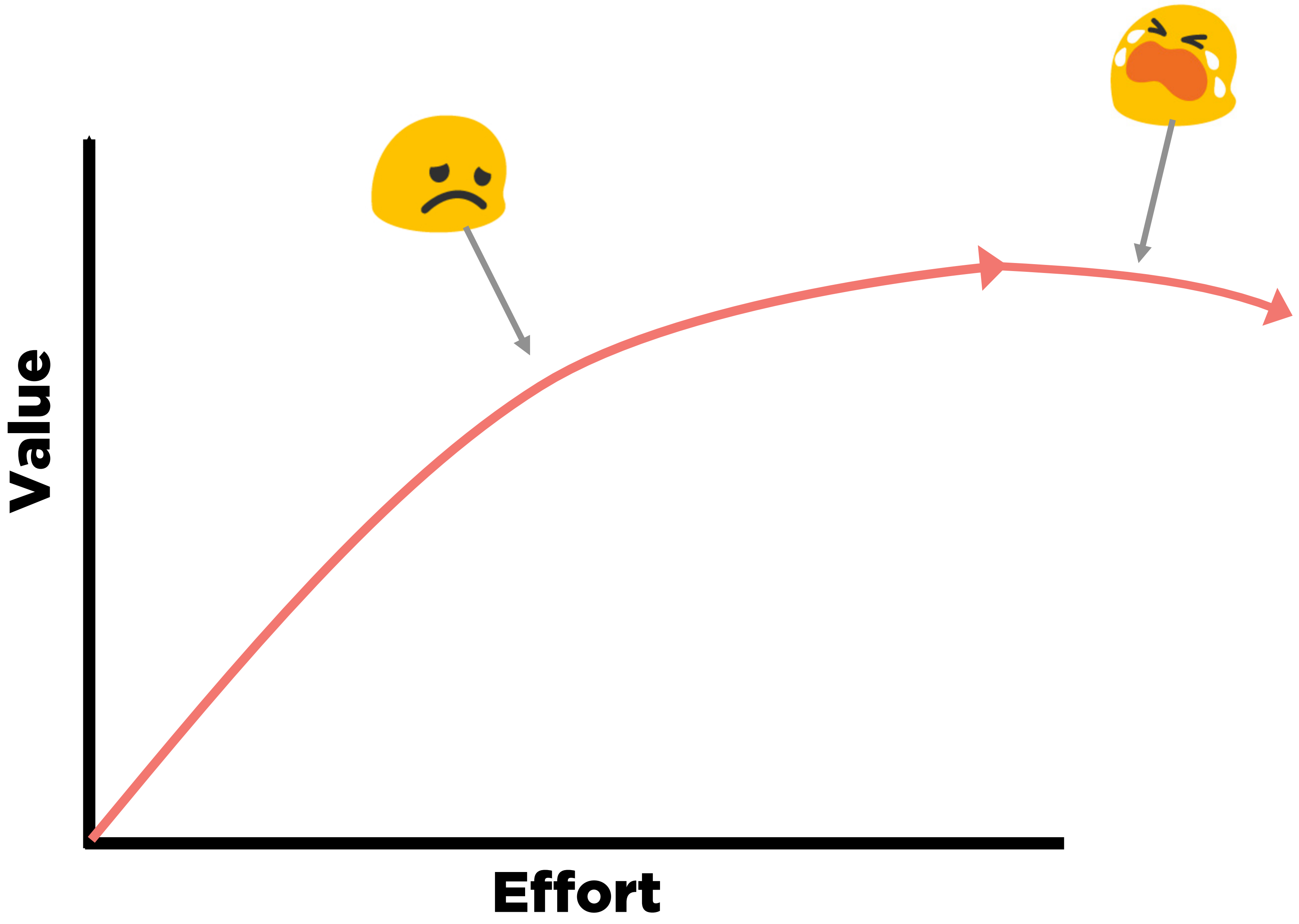
C++	Java	Python
Go	Ruby	C#
Node.js	Android Java	Objective-C
	PHP	

4

NIH → OSS

OBVIOUS


Value
Effort


Value
Effort

"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!)

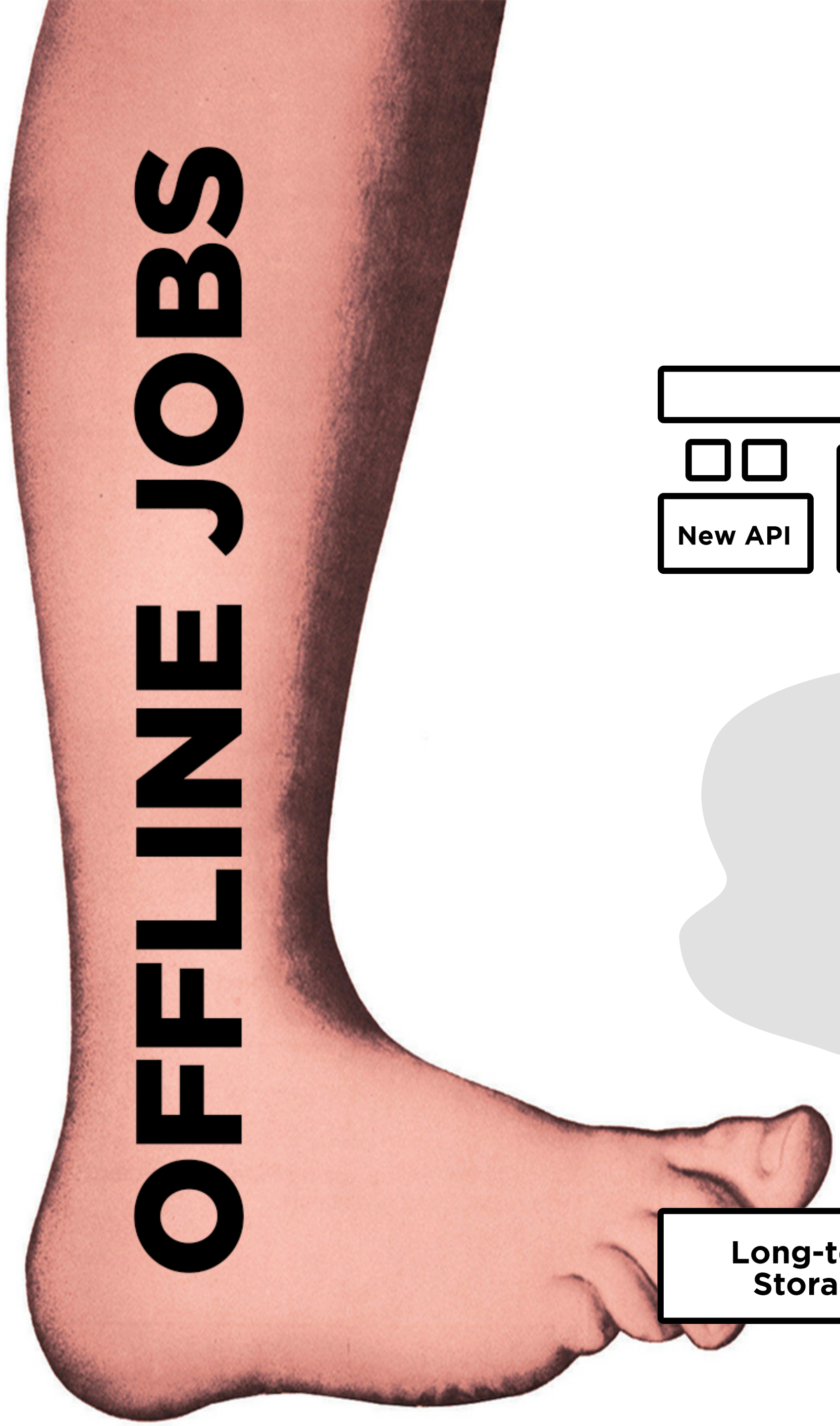
Wrapping up...

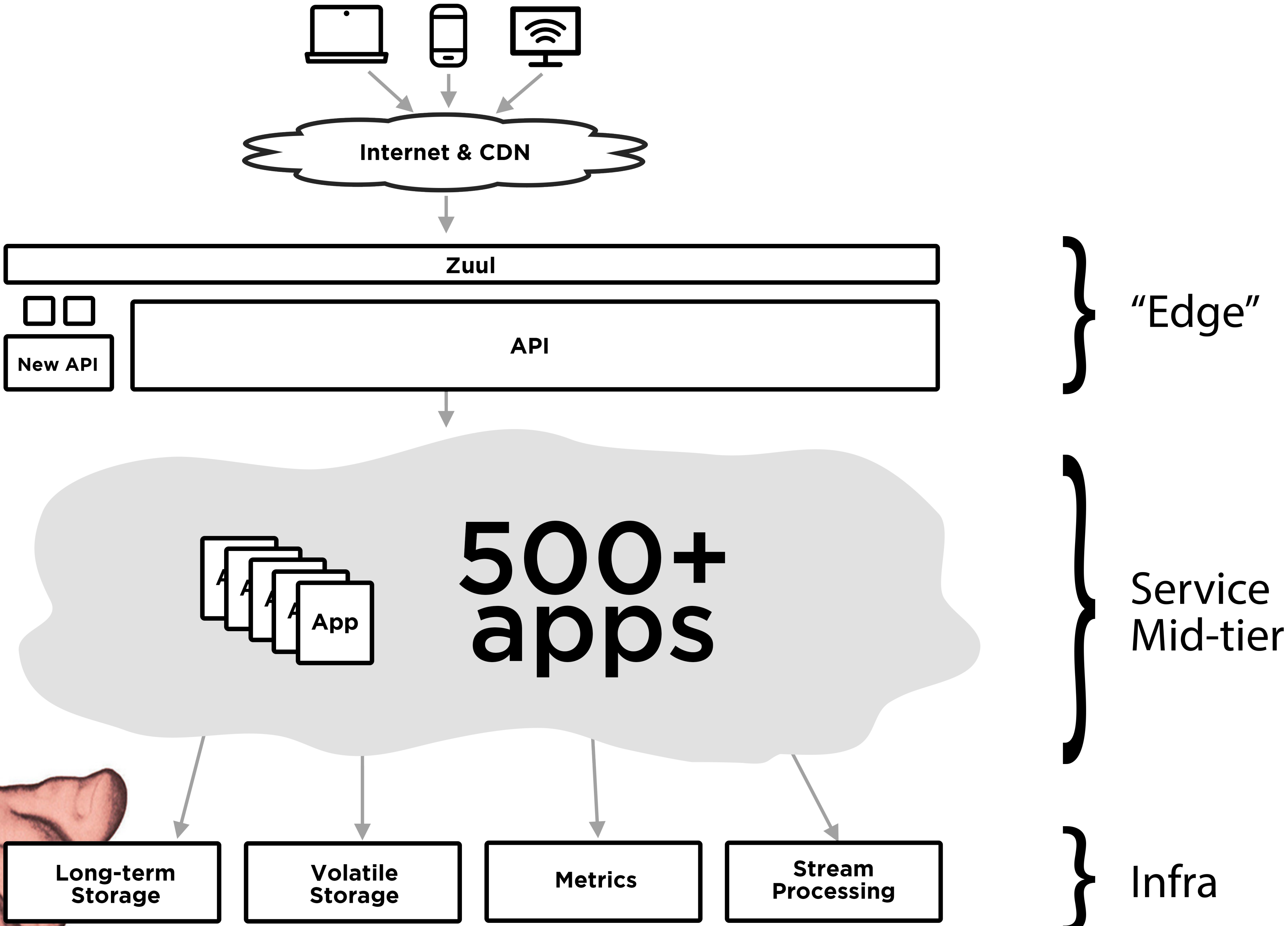
Everything discussed is done

- 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


Internet & CDN
Zuul
New API
API
App
500+ apps
Long-term Storage
Volatile Storage
Metrics
Stream Processing
"Edge"
Service Mid-tier
Infra