Reliably Releasing Software

Foundations of Software Engineering

Christopher S. Meiklejohn

Software Engineer, DoorDash

Adjunct Faculty, Carnegie Mellon University

Carnegie Mellon University

Goals

Identify the core challenges with modifying, testing, and deploying applications **safely.**

Describe and **differentiate** the possible techniques for ensuring **reliable** and **safe delivery of software at scale.**

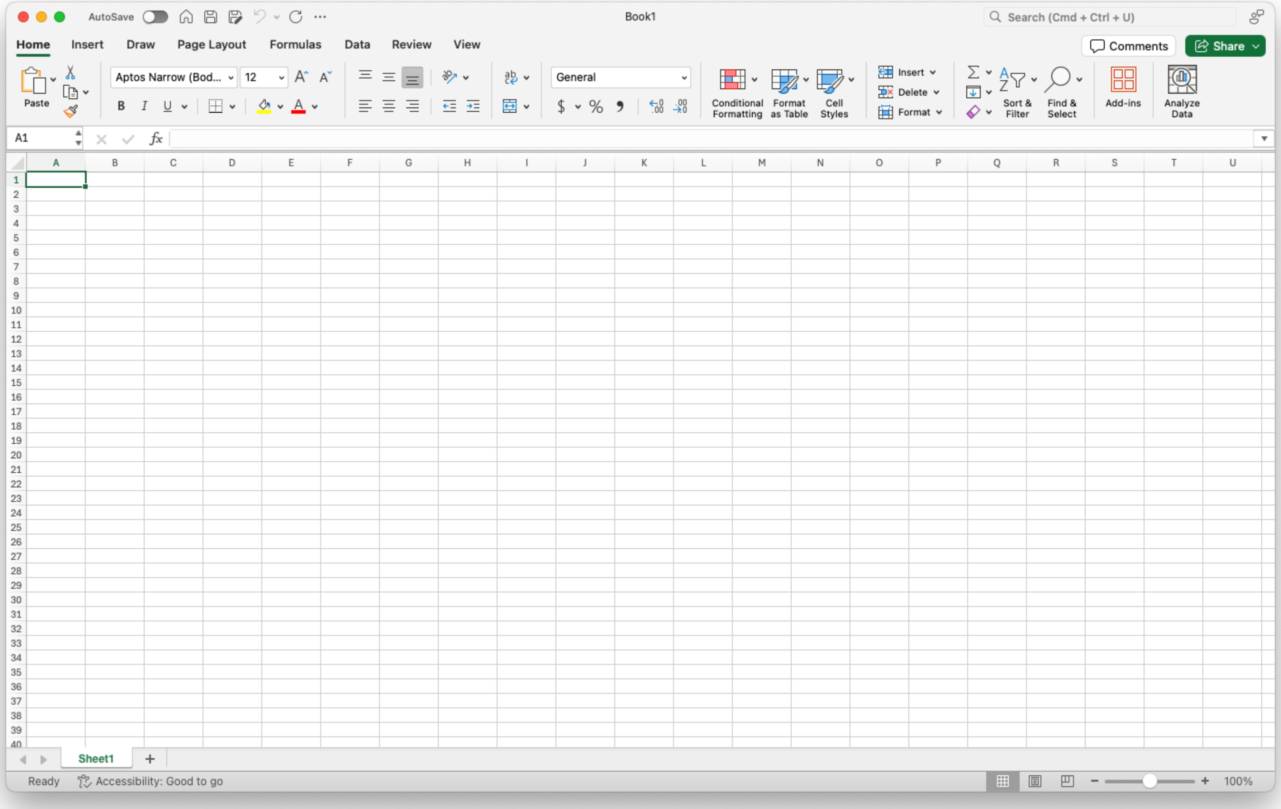
Practice identifying problematic changes and how to go about **making changes safely.**

How Do You Change This Software?

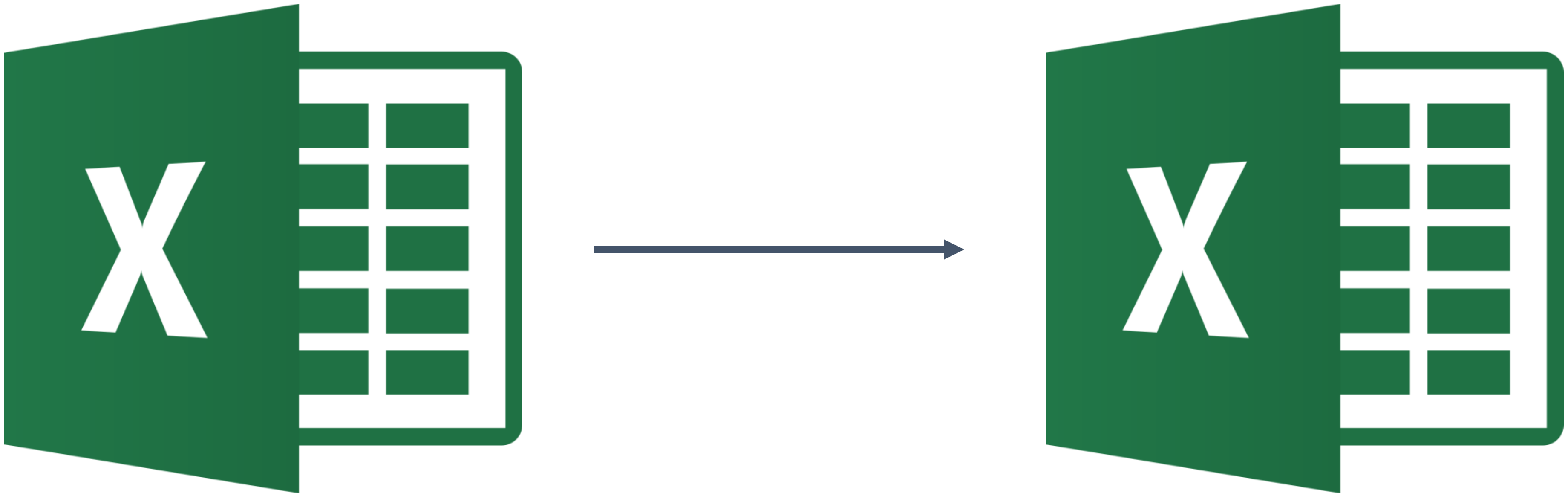
Modify
Implement one or more changes in the application and build the new version of the application.

Test
Test the application using a test suite or QA process to ensure application works correctly.

Release
Create new version of the software, users close their existing version and install it and open the new version.



App Upgrade: One Version To The Next



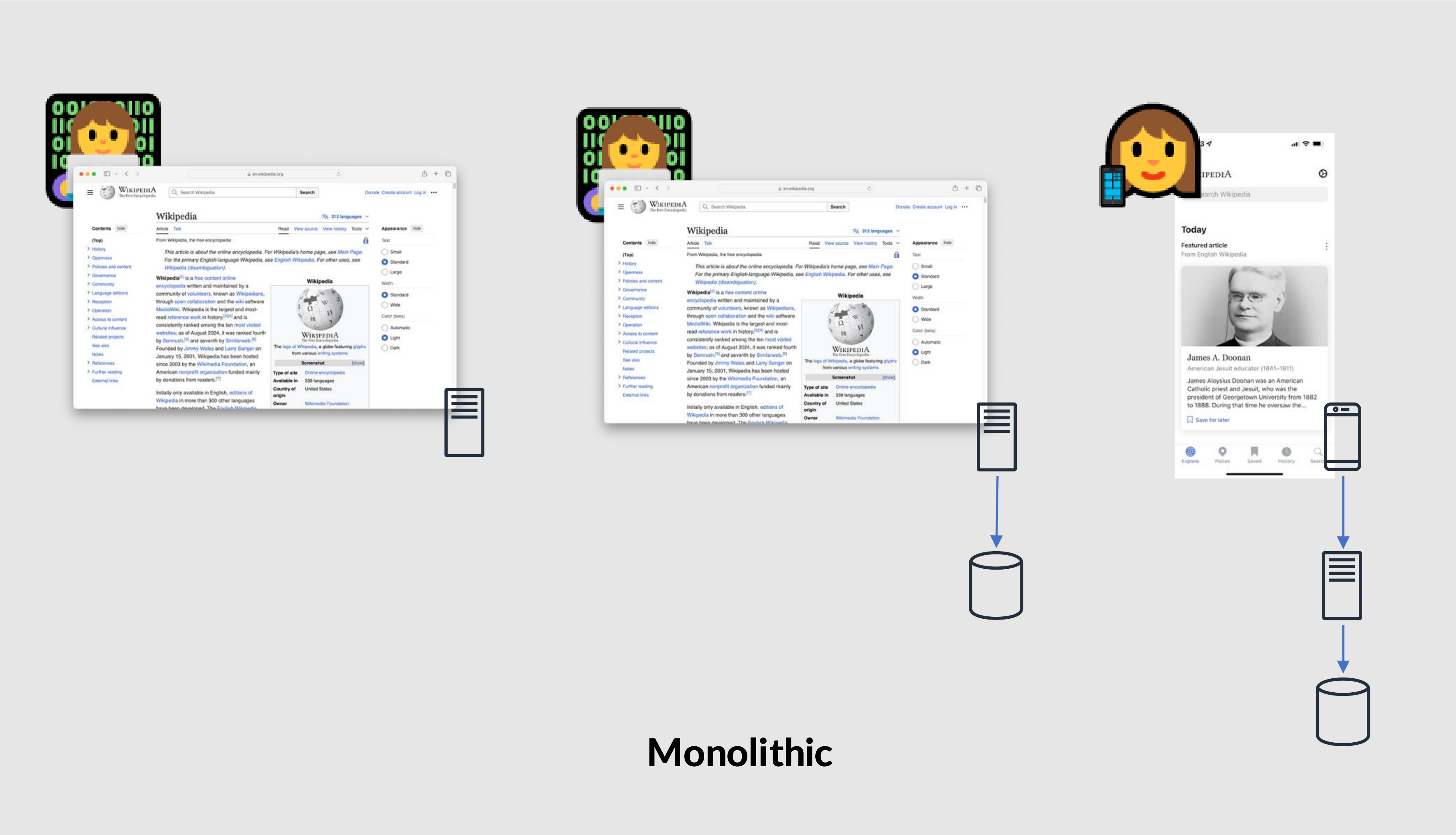
V1.0

V1.1

Similarly, if we want to **scale up this application to more users**, we just have users **install more copies of this application on *their* computer.**

This detail will **become important later.**

What About This Software?



en.wikipedia.org

Wikipedia The Free Encyclopedia | Search Wikipedia | Search | Donate Create account Log in

Wikipedia

312 languages

Article Talk | Read View source View history Tools

Contents hide

(Top)
- History
- Openness
- Policies and content
- Governance
- Community
- Language editions
- Reception
- Operation
- Access to content
- Cultural influence
- Related projects
- See also
- Notes
- References
- Further reading
- External links

From Wikipedia, the free encyclopedia

This article is about the online encyclopedia. For Wikipedia's home page, see Main Page. For the primary English-language Wikipedia, see English Wikipedia. For other uses, see Wikipedia (disambiguation).

Wikipedia[a] is a free content online encyclopedia written and maintained by a community of volunteers, known as Wikipedians, through open collaboration and the wiki software MediaWiki. Wikipedia is the largest and most-read reference work in history,[3][4] and is consistently ranked among the ten most visited websites; as of August 2024, it was ranked fourth by Semrush,[5] and seventh by Similarweb.[6] Founded by Jimmy Wales and Larry Sanger on January 15, 2001, Wikipedia has been hosted since 2003 by the Wikimedia Foundation, an American nonprofit organization funded mainly by donations from readers.[7]

Initially only available in English, editions of Wikipedia in more than 300 other languages have been developed. The English Wikipedia

Wikipedia

Wikipedia The Free Encyclopedia

The logo of Wikipedia, a globe featuring glyphs from various writing systems

Screenshot [show]

Type of site	Online encyclopedia
Available in	339 languages
Country of origin	United States
Owner	Wikimedia Foundation

Appearance hide

Text
- ○ Small
- ● Standard
- ○ Large

Width
- ● Standard
- ○ Wide

Color (beta)
- ○ Automatic
- ● Light
- ○ Dark

Wikipedia | Search Wikipedia

Today

Featured article

From English Wikipedia

James A. Doonan

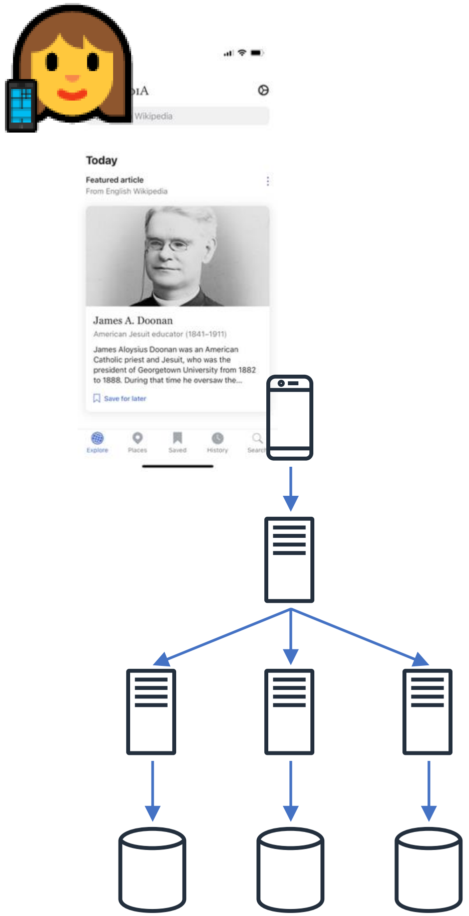
American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...

Save for later

Monolithic

Microservice



What Are The Differences?

Location

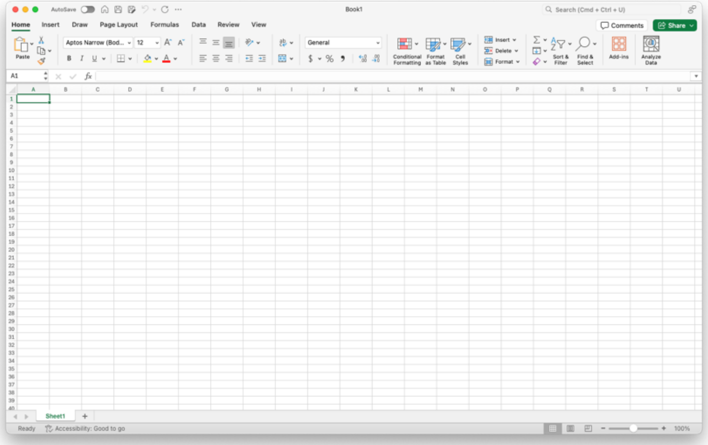
Servers, not Devices
Application runs on server and is **deployed to cloud.**
It’s **not installed** on client’s device.

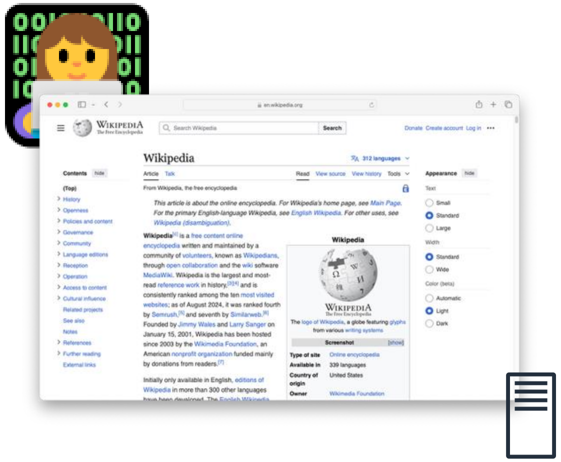
Scaling

“Scale out”
Scaling is achieved by increasing the server capacity, instead of installing the software on more clients.

Availability

“Always On”
Applications are upgrade in place, typically aiming for zero-downtime.

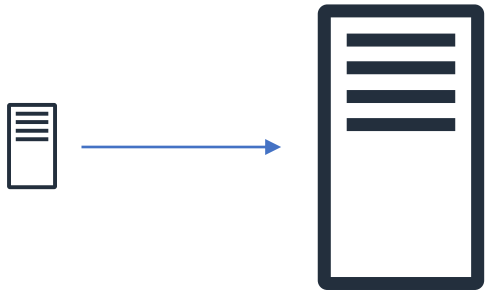




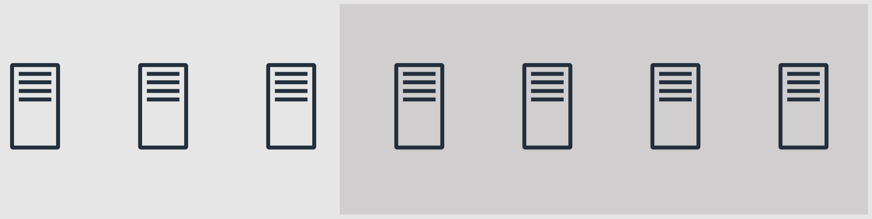
Scaling and Deployments: Intertwined

Scaling

Vertical Scaling



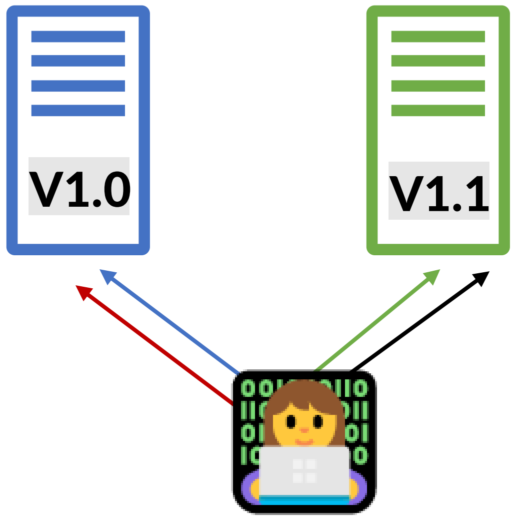
Horizontal Scaling



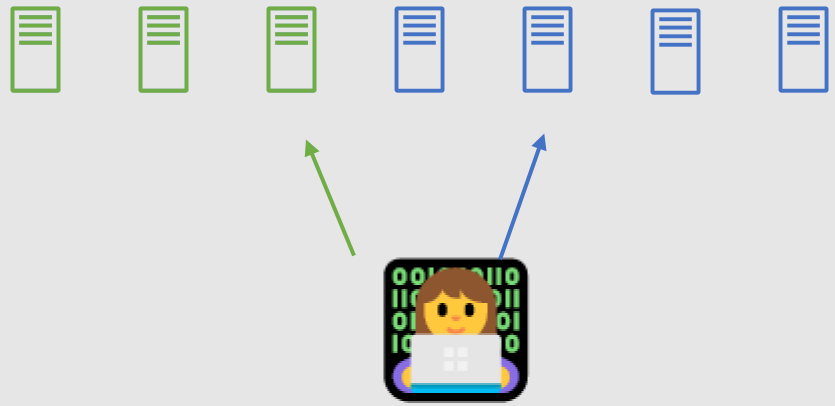
Red/Black: switch
Blue/Green: incremental traffic

Deployment

Red/Black or Blue/Green



Rolling Upgrade



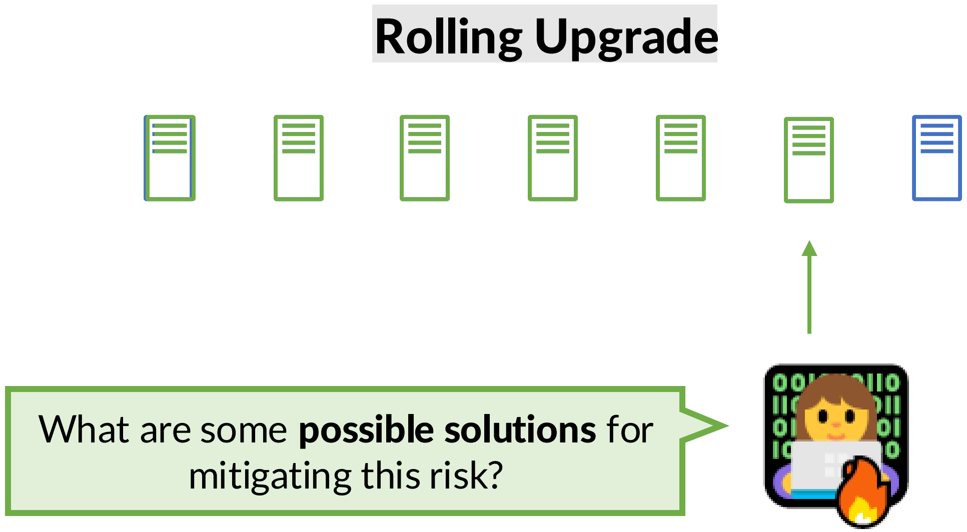
Bugs?

Rollouts Are Slow
Applications may have **thousands of server instances**, rollouts can take multiple hours.

Bugs Might Take a While To Surface
Error rate might be low, might take a while to detect, might be manually reported.

High Cost/Impact For Bugs
Every second of a bug may indicate possible user error. *(e.g., can't request a ride)*

Can't Immediately Rollback
Not enough capacity to immediately rollback *(i.e., blue nodes)* and deployment of old code is as slow as the new code.



What are some **possible solutions** for mitigating this risk?

Dark Launch

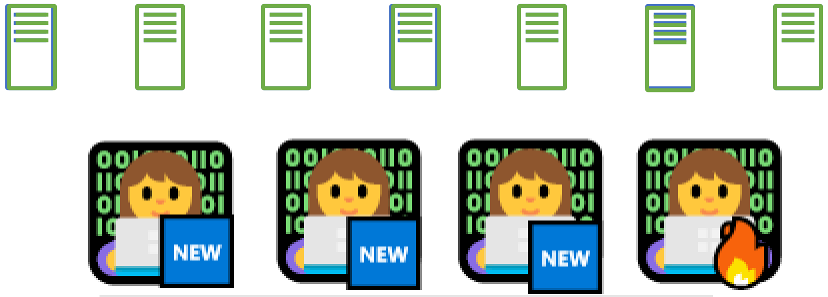
Solution: **Dark Launch**

Rollout with Features Dark
Perform rollout of code at the "same" existing version with all new features turned "off" – no-op rollout.

Incremental Ramp of Flag
Incrementally enable feature to users based on percentage and roll out to employee (or other limited cohort first) for early detection *(i.e., dogfooding.)*

Rollback: First Response
Ensure that code can be rolled back immediately on the first indication of issue.

Rolling Upgrade with Dark Feature



Incremental Feature Release

Remember to write tests with the feature flag = **false and true prior to rollout!**

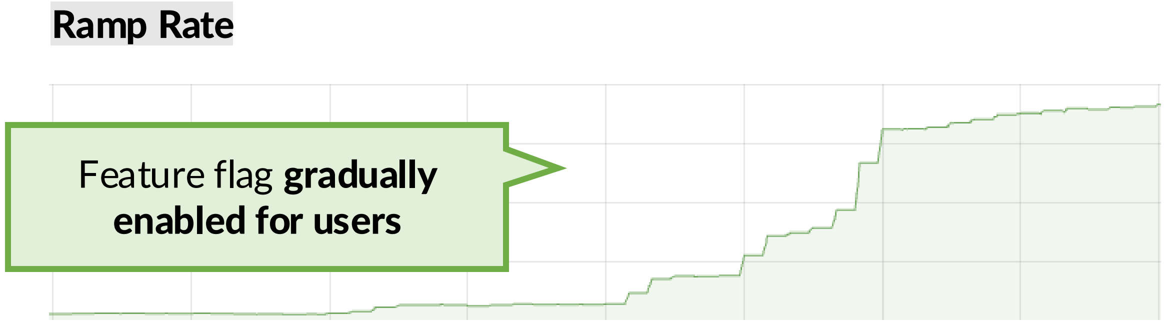
Dark Launch: Observability

How do you **identify a rollout problem?**

Hit Rate
Use metrics tracking new code execution to track introduction of new feature.

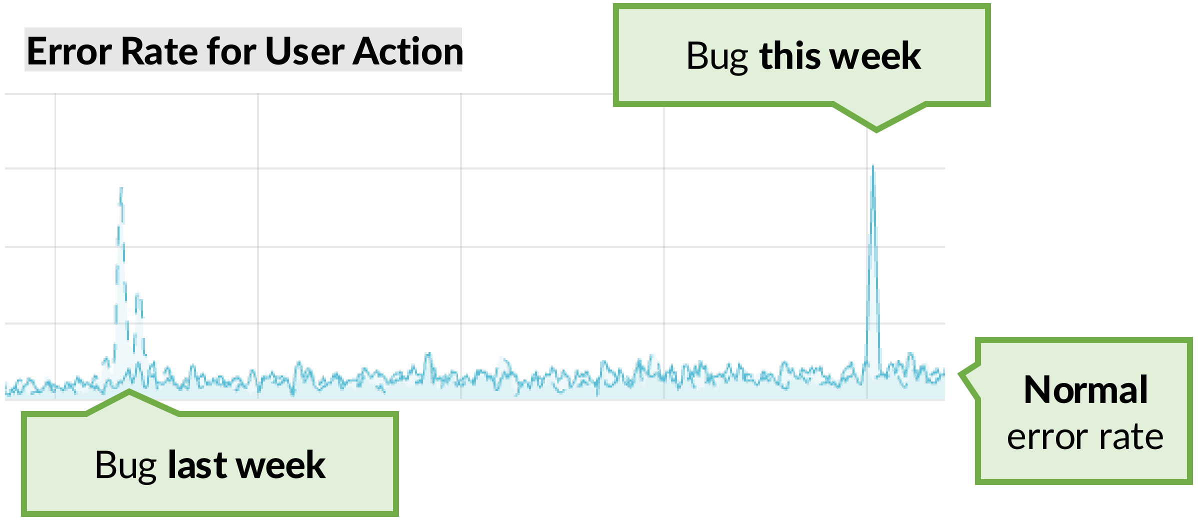
Error Rates
Use metrics tracking error rates and compare with week-over-week for derivations.

Remember: some errors may be normal depending on the metric.
Correlate them with the feature ramps.



Ramp Rate

Feature flag **gradually enabled for users**



Error Rate for User Action

Bug **this week**

Bug **last week**

Normal error rate

Databases: Changing the Database

Modifications to Database + Application
Often, you will have to
- modify the database (*e.g.*, new column)
- with the application (*e.g.*, new code)
for new features.

You are developing a **a new feature to highlight certain pages on Wikipedia.**

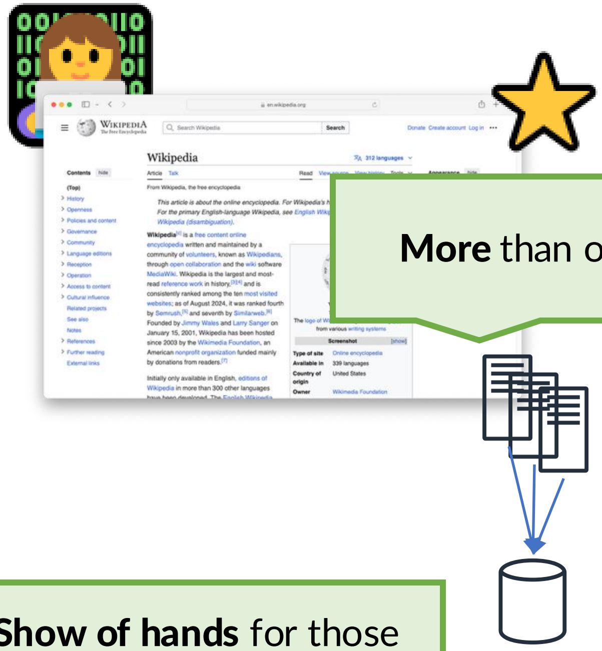
Application Code **Before:**

```
SELECT title, content FROM pages WHERE url = “…”
```

Application Code **After:**

```
SELECT title, content, starred FROM pages WHERE url = “…”
```

We need to modify the database to add a `starred` **field.**

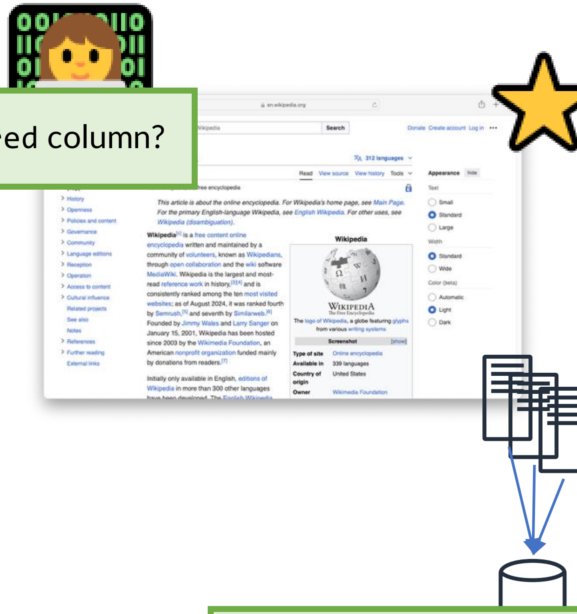


More than one server!

Show of hands for those who have **used SQL before!**

Databases: What’s Hard About This?

We have one database schema, **how do we change it?**
(recall: we have to add a new field called `starred`)



What type is the `starred` column?

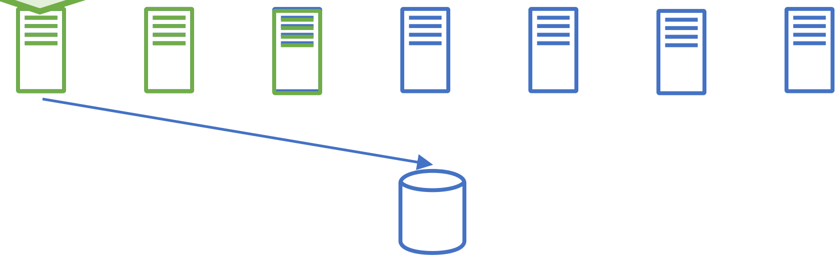
No Rolling Upgrades
Can’t synchronize rolling upgrade between app + database, no rolling upgrade for DB, even schema changes in distributed databases are atomic across nodes.

In short: changes are **atomic.**

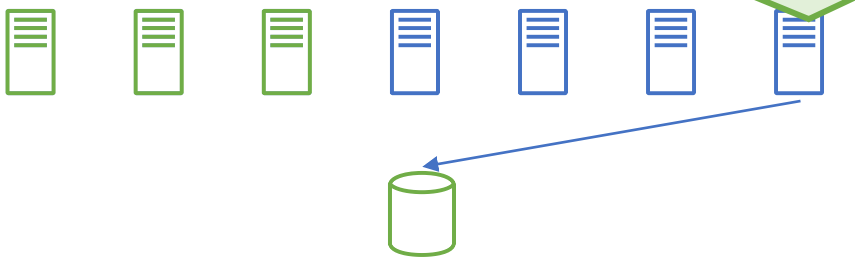
What type of problems does a **rolling upgrade of our app code** introduce if our DB change takes effect immediately?

Problems During Rolling Upgrade/Release

New version might be **incompatible with old DB**
(i.e., access starred before there.)



Old version might be **incompatible with new DB version.**
What scenarios might this be?



Database Changes: Adding a New Field

1. **Add new field to the database using a migration.**
New field added to the schema, but nothing uses it.
Nothing *(i.e., indexes, integrity constraints, etc.)* can use this field and field **must be nullable.**

2. **Dark Launch Application With Code To Write Field**
Dark launch new version of application with code to begin writing the new field.
Gradually roll out feature that writes the new field.

Code to write field **may contain a bug** *(e.g., serialization.)*

3. **Dark Launch Application With Code To Read Field**
Dark launch new version of application with code to begin reading the new field.
Gradually roll out feature that writes the new field. **Must handle nulls!**

Code to read field **may contain a bug** *(e.g., logic error.)*

Only after you've rolled out features to 100% of all users and waited for bug reports:

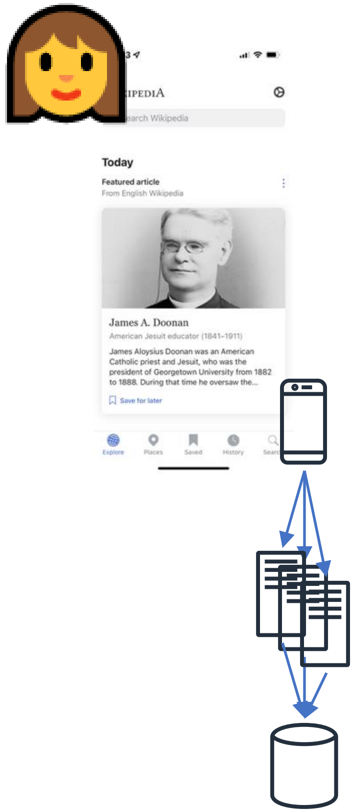
4. **Remove Migration Code**
Deploy version of code without migration *(i.e., feature flags.)*
You can't dark launch this, otherwise you'll loop indefinitely.

Mobile Clients: Another Moving Piece

Modifications to DB + App + Client
Many times you will have to modify the database with the application **and the mobile client** for new features.

Release Coordination
Can't synchronize updates: **mobile application modifications must be done ahead of time and submitted to the App Store/Google Play.**

Data Interchange
Backwards compatible message formats must be used and code must be able to **handle feature being absent/present.**
(think: removing a field in JSON)

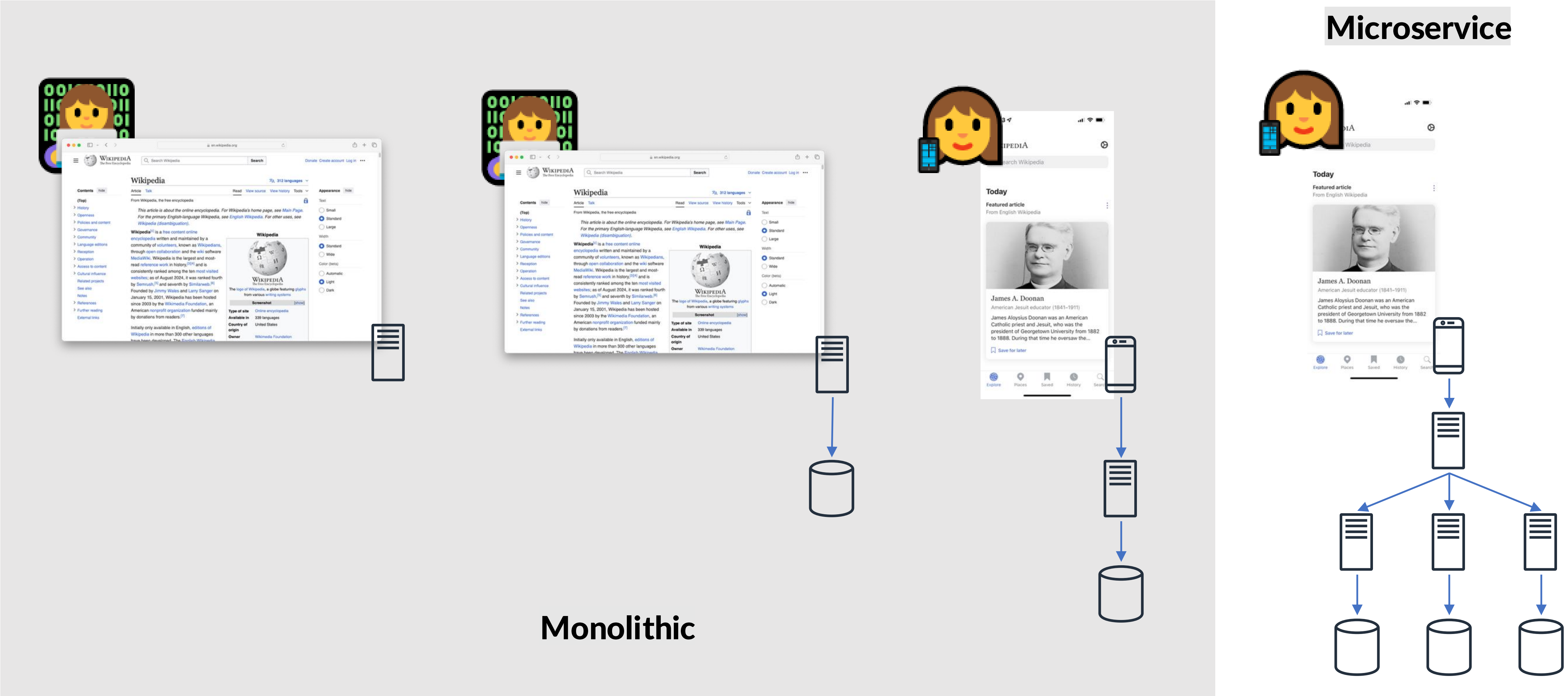


Today

Featured article
From English Wikipedia

James A. Doonan
American Jesuit educator (1841–1911)
James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...

What About This Software?



Microservice

Monolithic

Microservice Applications

Microservice architecture is an architectural style where applications are constructed from services that communicate over the network using RPC and are developed, scaled and deployed independently.

NETFLIX	UBER	DOORDASH
1,000 services (2021)	**2,200** services *>120 for getting ride* (2016)	**500** services *>100 involved in core flow* (2024)

Microservice applications are the **most common and complex** type of distributed application being built today.

Twitter (2017) operates a > 10k node distributed Hadoop cluster.
However, **most nodes have the same behavior, running the exact same code.**

DoorDash (2024) operates 500 microservices.
Each service provides **different functionality, has a different API, and is deployed continuously.**

Microservices: Socio-Technical Problem

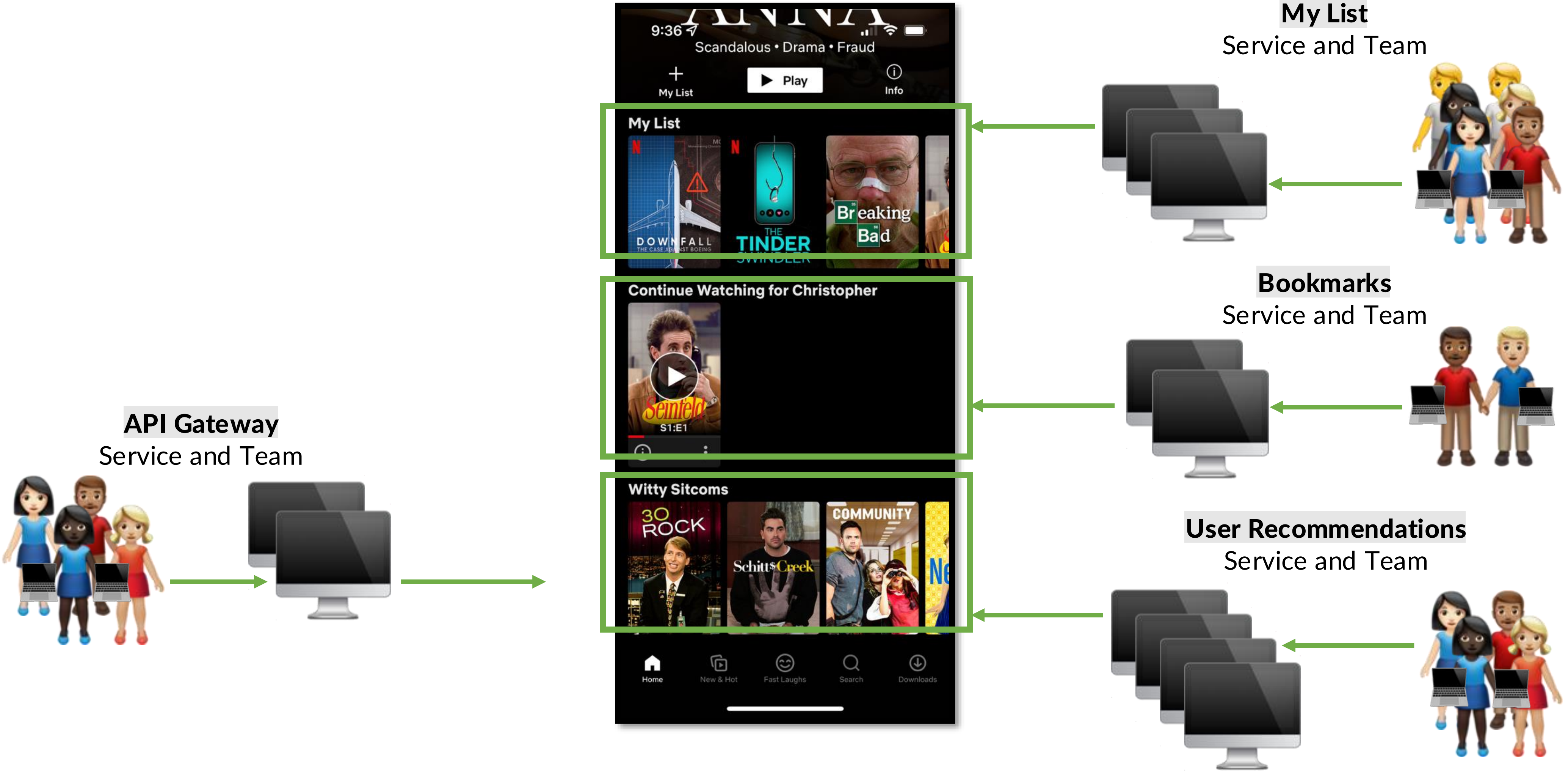
Microservice architectures solve a **socio-technical problem:**

👍 Technical solution to **support rapid feature development at scale** as an organization grows, that breaks down the application into components where no single engineer needs knowledge of the entire application to develop and deploy features.

We would not develop an application this way **unless it was absolutely necessary.**

Technical solution splits code across multiple repositories (and languages) making **it harder to develop, test, analyze, and reason about the application.** *(e.g., IDE support, static and dynamic analysis tools, integration and functional testing, etc.)* 👎

Netflix: Microservice Architecture



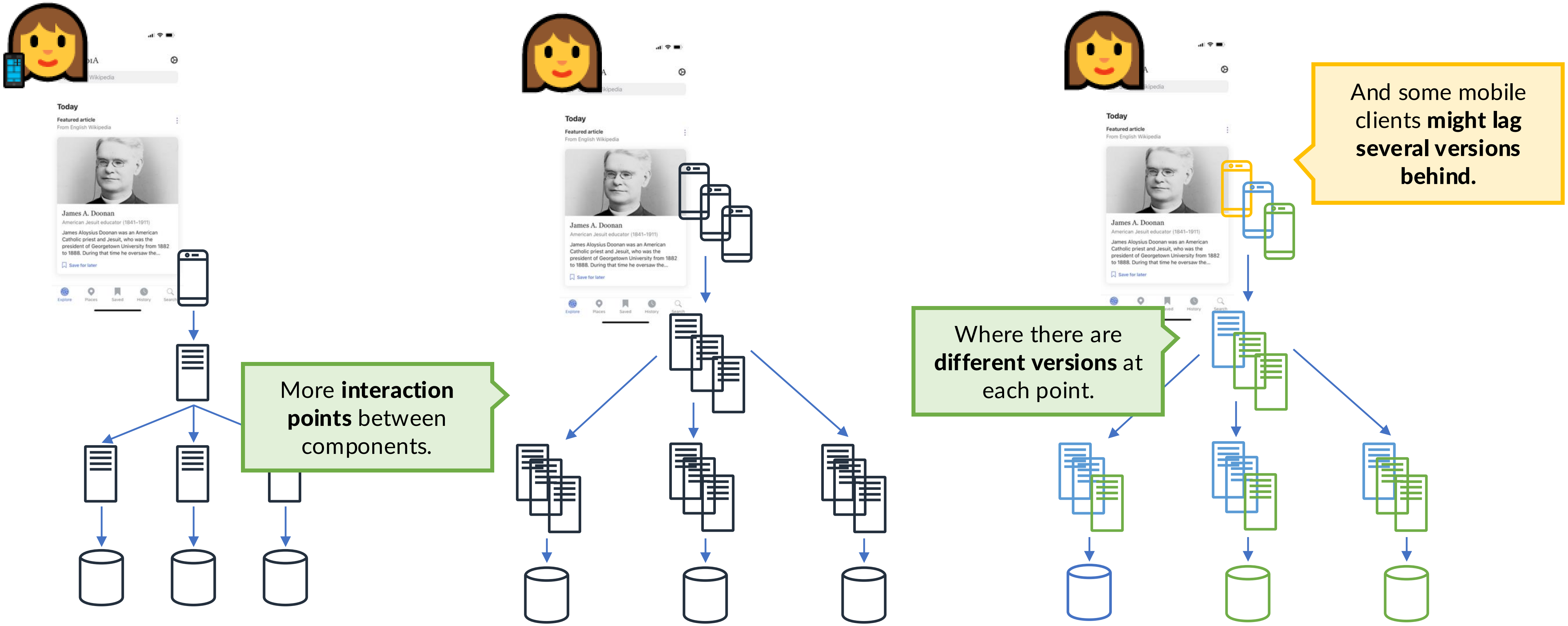
My List
Service and Team

Bookmarks
Service and Team

User Recommendations
Service and Team

API Gateway
Service and Team

Revisiting: Wikipedia



Today

Featured article
From English Wikipedia

James A. Doonan
American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...

Save for later

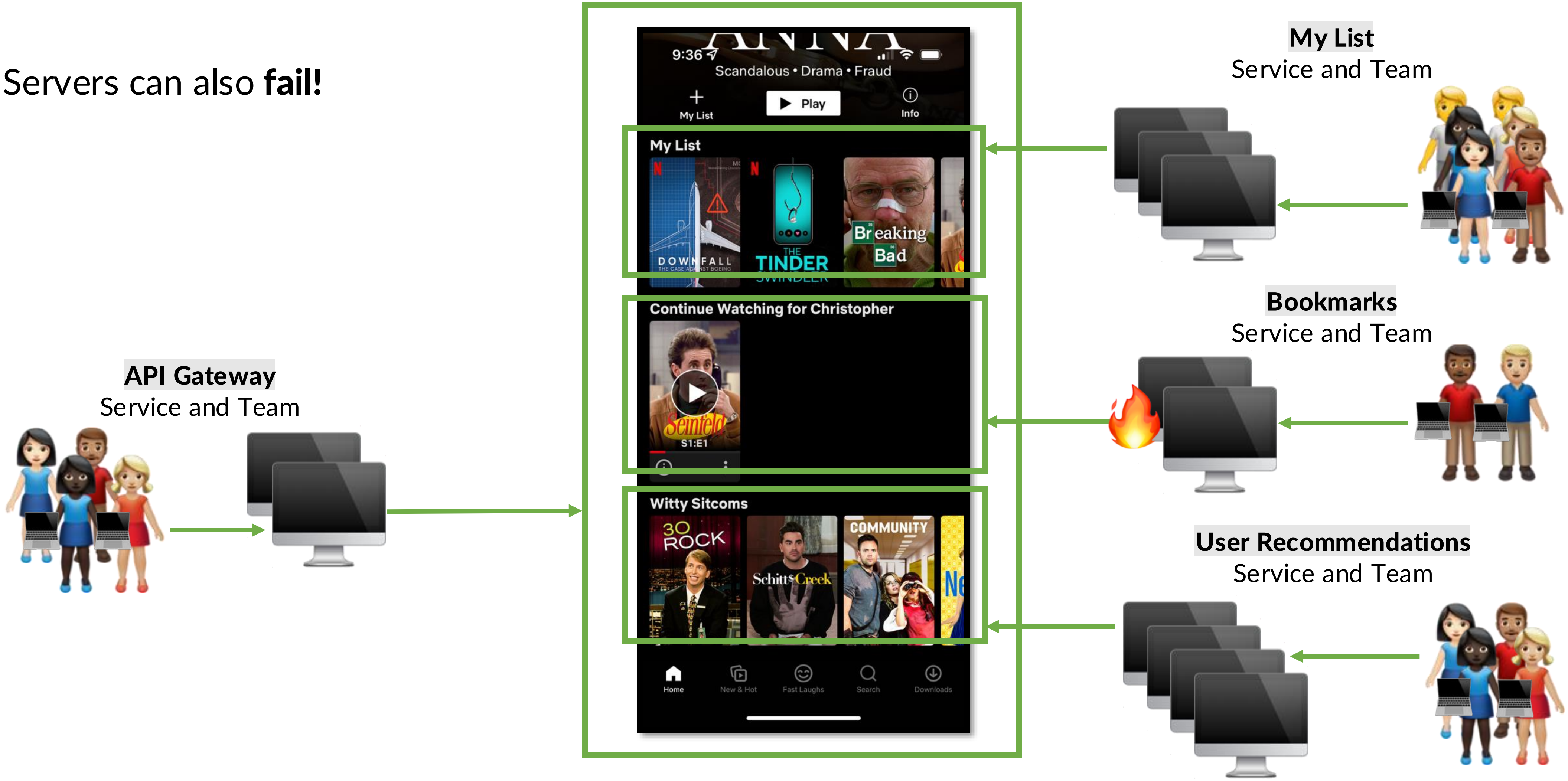
More **interaction points** between components.

Where there are **different versions** at each point.

And some mobile clients **might lag several versions behind.**

…Just One More Thing

Servers can also **fail!**



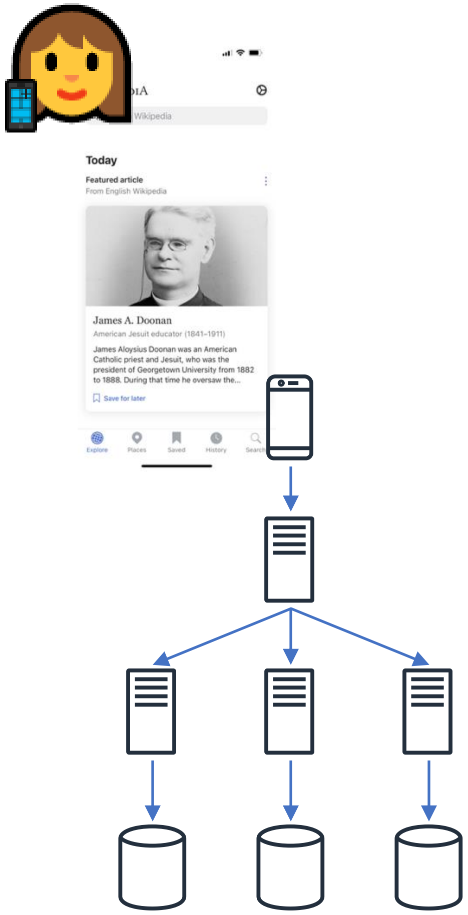
API Gateway
Service and Team

My List
Service and Team

Bookmarks
Service and Team

User Recommendations
Service and Team

Partial Failure



ıA

Wikipedia

Today

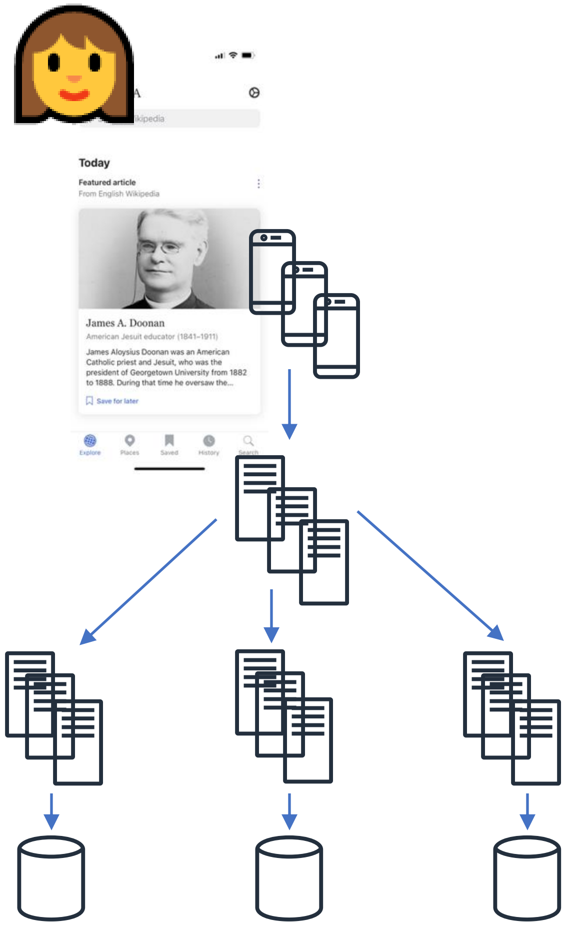
Featured article

From English Wikipedia

James A. Doonan

American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...



A

kipedia

Today

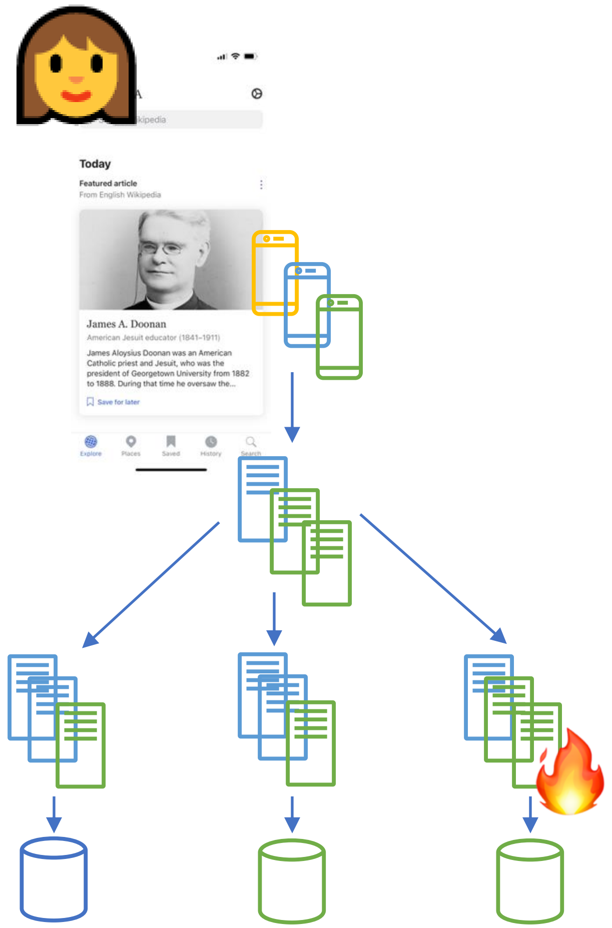
Featured article

From English Wikipedia

James A. Doonan

American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...



A

kipedia

Today

Featured article

From English Wikipedia

James A. Doonan

American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...

Partial Failure in Microservices: Different

…but, microservices are also susceptible to **partial failure:**

1. **Failed node causing connection errors.**
   Prior to removal by health check, application must still tolerate and respond to errors.

2. **Bad deployments.**
   Number of nodes return error responses *(e.g., 500 Internal Server Error)* before removal.

3. **Service failures only with certain arguments.**
   Service returns errors when provided with certain arguments by a caller only. *(e.g., NPE, etc.)*

4. **Dependencies of a given RPC method may be malfunctioning.**
   Direct dependencies of a service may slow down, timeout, or fail in other ways.

Microservice Application: Audible

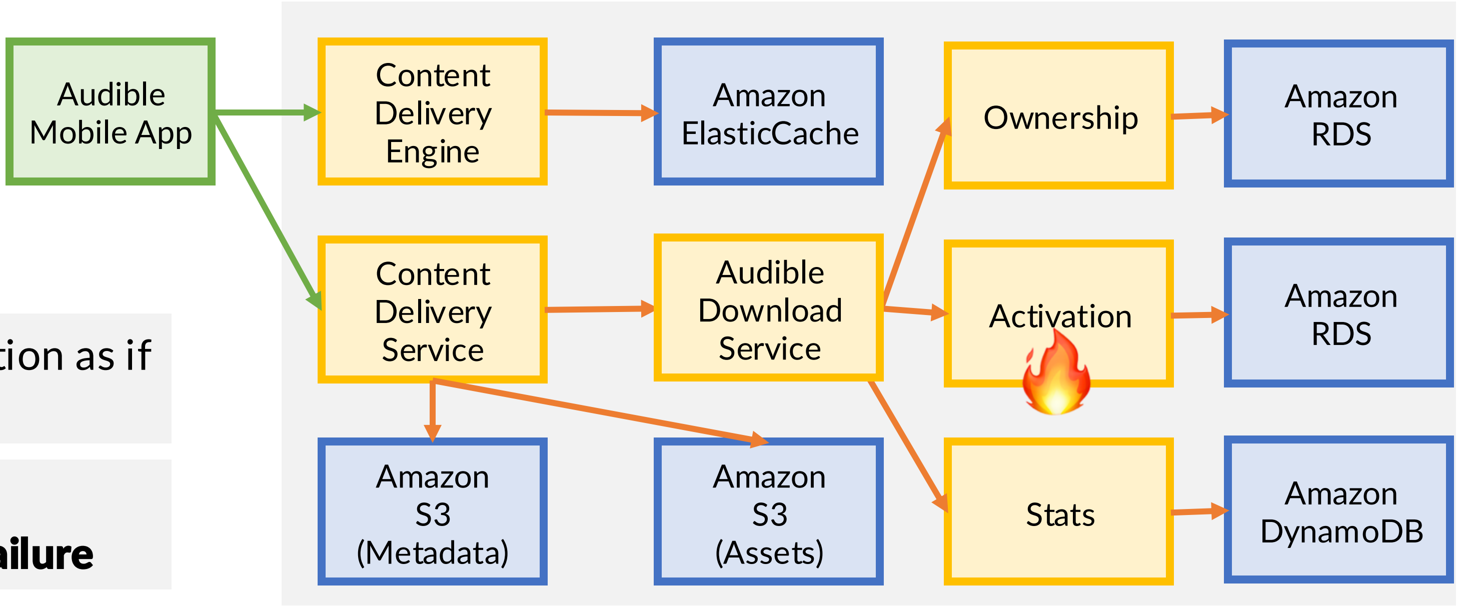
One solution to **partial failure:**

1. Build the microservice application as if it's a **monolithic application**

2. Fail the entire request **if any dependency returns a failure**

These are called **hard dependencies.**

Alternatively,
should we **embrace failure?**



Audible Mobile App

Content Delivery Engine

Amazon ElastiCache

Ownership

Amazon RDS

Content Delivery Service

Audible Download Service

Activation 🔥

Amazon RDS

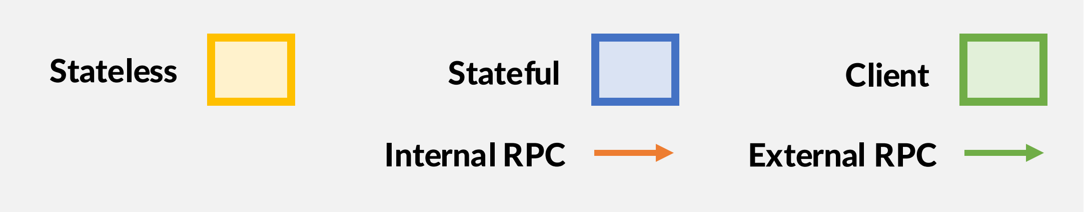
Amazon S3 (Metadata)

Amazon S3 (Assets)

Stats

Amazon DynamoDB

Audible
Audiobook streaming service



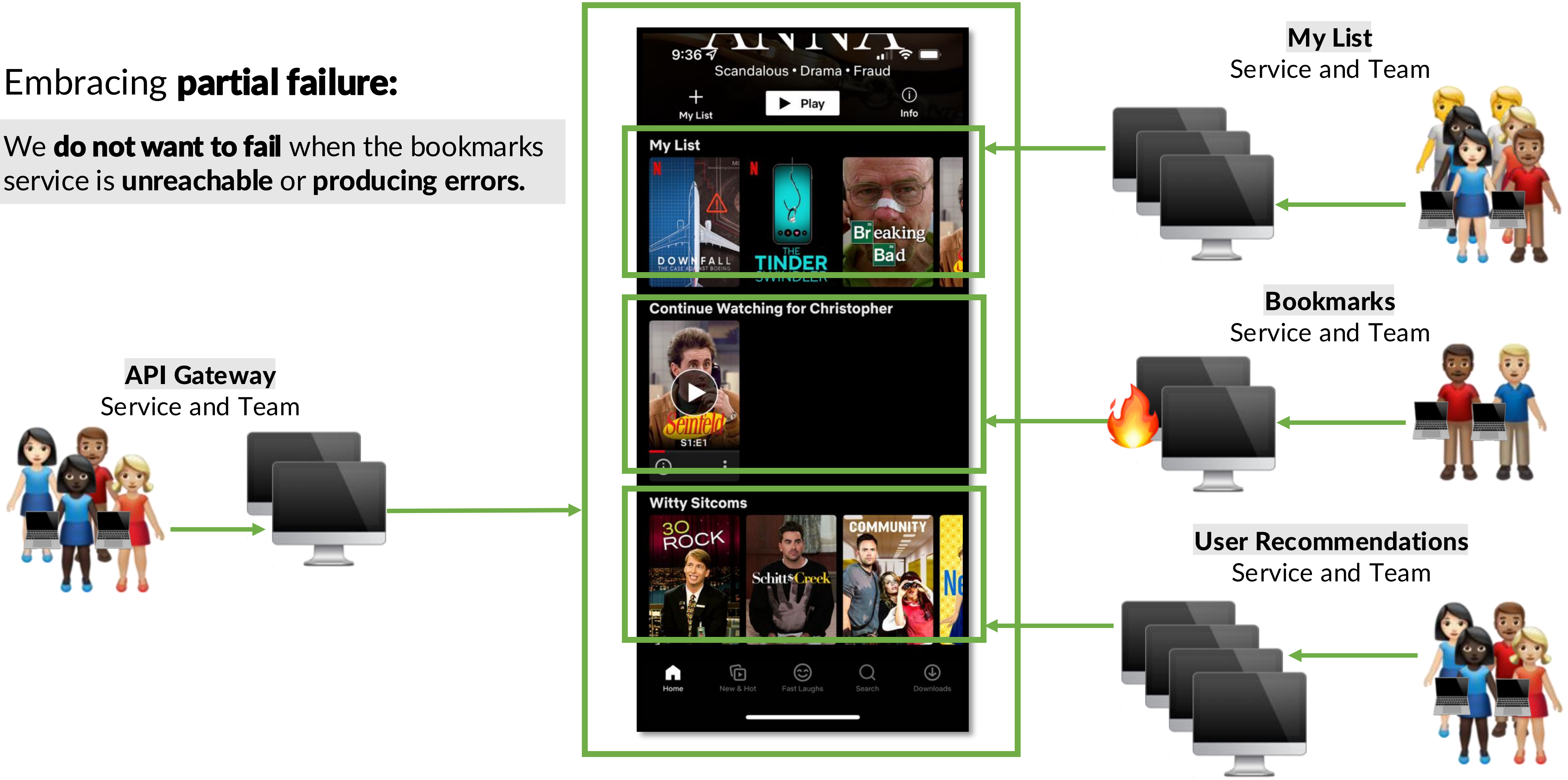
Stateless 🟨 **Stateful** 🟦 **Client** 🟩

Internal RPC → **External RPC** →

Microservice Application: Netflix

Embracing **partial failure:**

We **do not want to fail** when the bookmarks service is **unreachable** or **producing errors.**



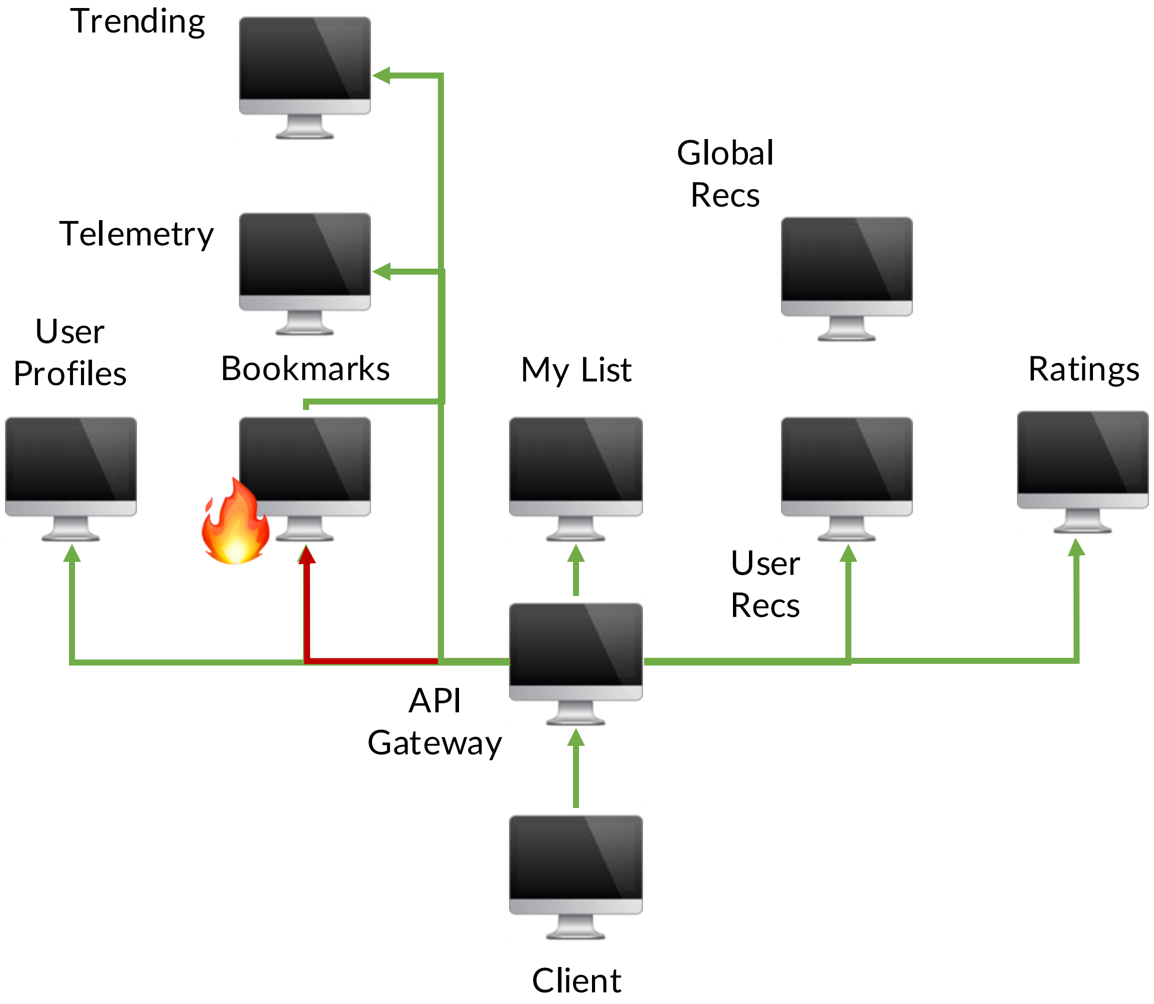
API Gateway
Service and Team

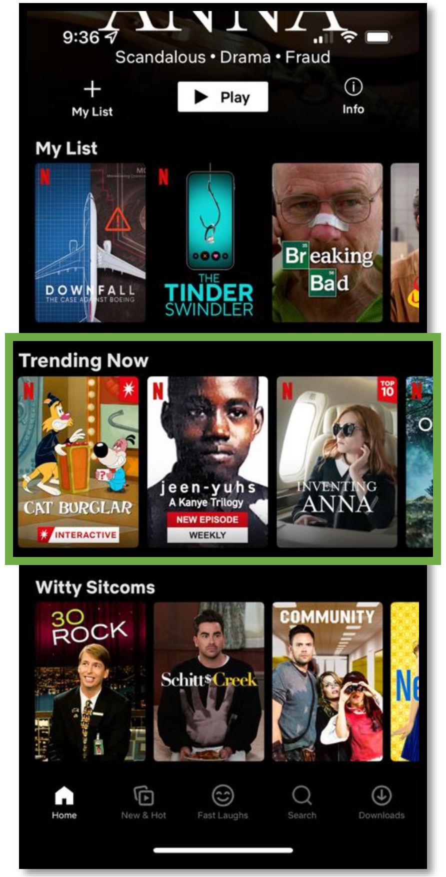
My List
Service and Team

Bookmarks
Service and Team

User Recommendations
Service and Team

What *should* happen?





Fallbacks:

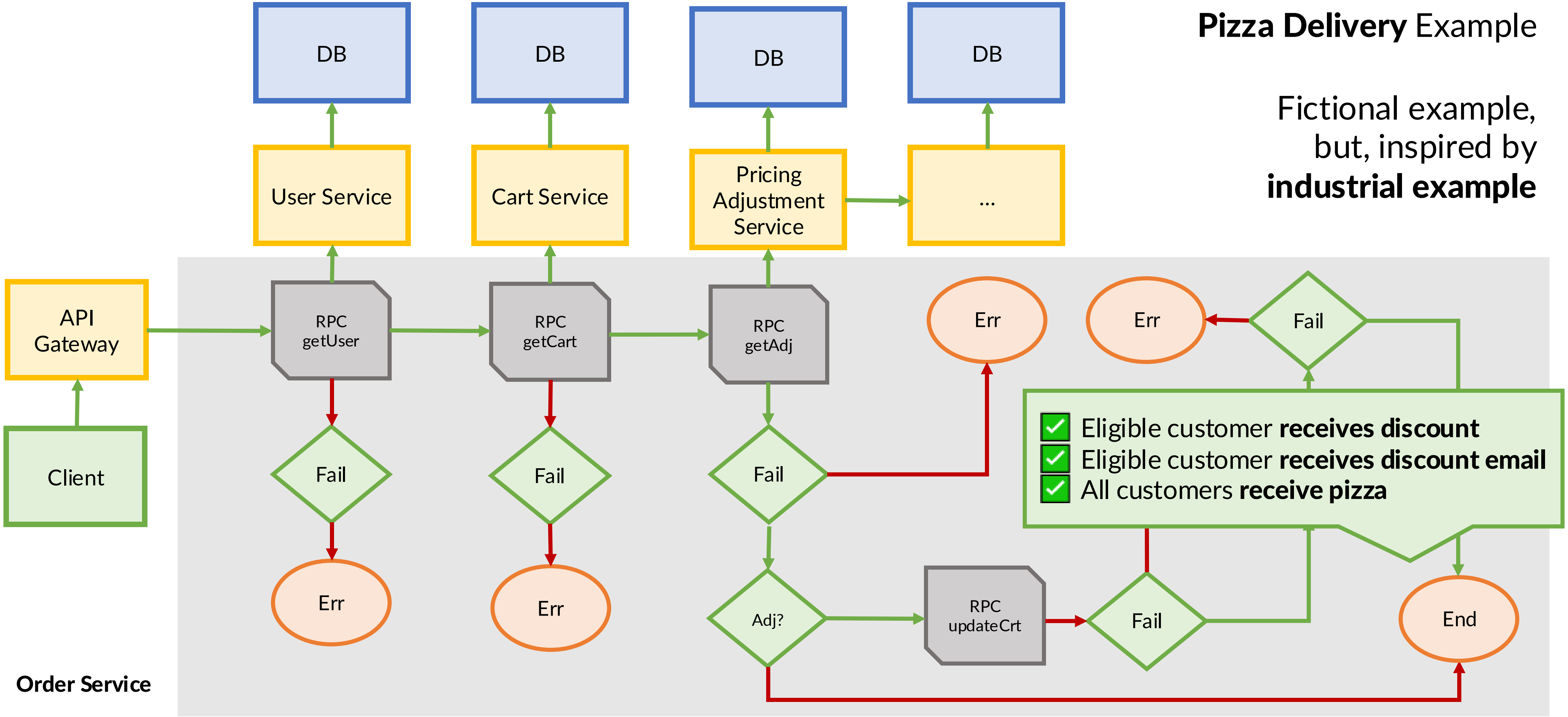
Developers specify **alternative application logic** in the event of dependency failure.

These are called
soft dependencies.

What *actually* happens? 🤔

We need to **test** it.

Example: Purchase Application

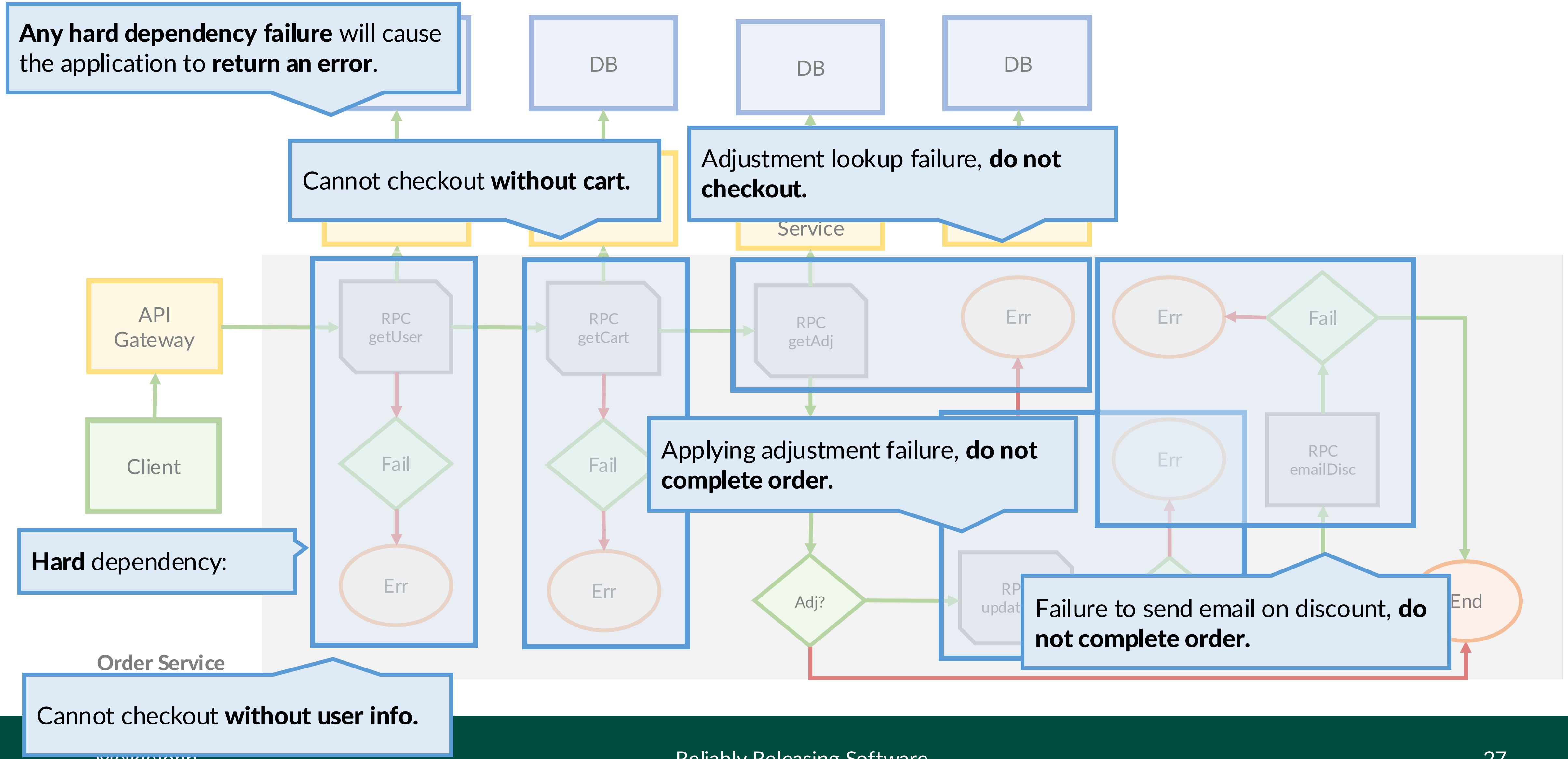


✓ Eligible customer **receives discount**
✓ Eligible customer **receives discount email**
✓ All customers **receive pizza**

Pizza Delivery Example

Fictional example,
but, inspired by
industrial example

Purchase: Hard Dependencies



Any hard dependency failure will cause the application to **return an error**.

Cannot checkout **without cart.**

Adjustment lookup failure, **do not checkout.**

Applying adjustment failure, **do not complete order.**

Hard dependency:

Failure to send email on discount, **do not complete order.**

Cannot checkout **without user info.**

Active Learning: Dependency Types

👨‍💻 **“Not great.”**

“Failure of any dependency forces application to **fail the checkout process.”**

Discuss with you neighbor(s) and **answer the following:**

1. **What might we want to change about the way this application handles failure?**
   (i.e., the business logic, not the application behavior)

2. **How will we make sure they are “good” changes?**
   (i.e., the business logic doesn’t negatively affect the business.)

3. **You guessed it, I’m looking for metrics. What are they?**
   (you knew this question was coming.)

Results of Testing the Application

👨‍💻 **“Not great.”**

“Failure of any dependency forces application to f

Business logic decisions conditional on failure **that cannot be automatically determined.**

Identified Problems:

1. Not being able to **send the discount email** shouldn’t cancel the order with an error.

👨‍💻 *To Fix:* Allow the order to be processed **regardless of email failure.**

2. Customers **not eligible for a discount cannot checkout if pricing adjustment call fails.** *(where, it would have returned $0, anyway.)*

👨‍💻 *To Fix:* **Assume a pricing adjustment of $0** when the call fails.

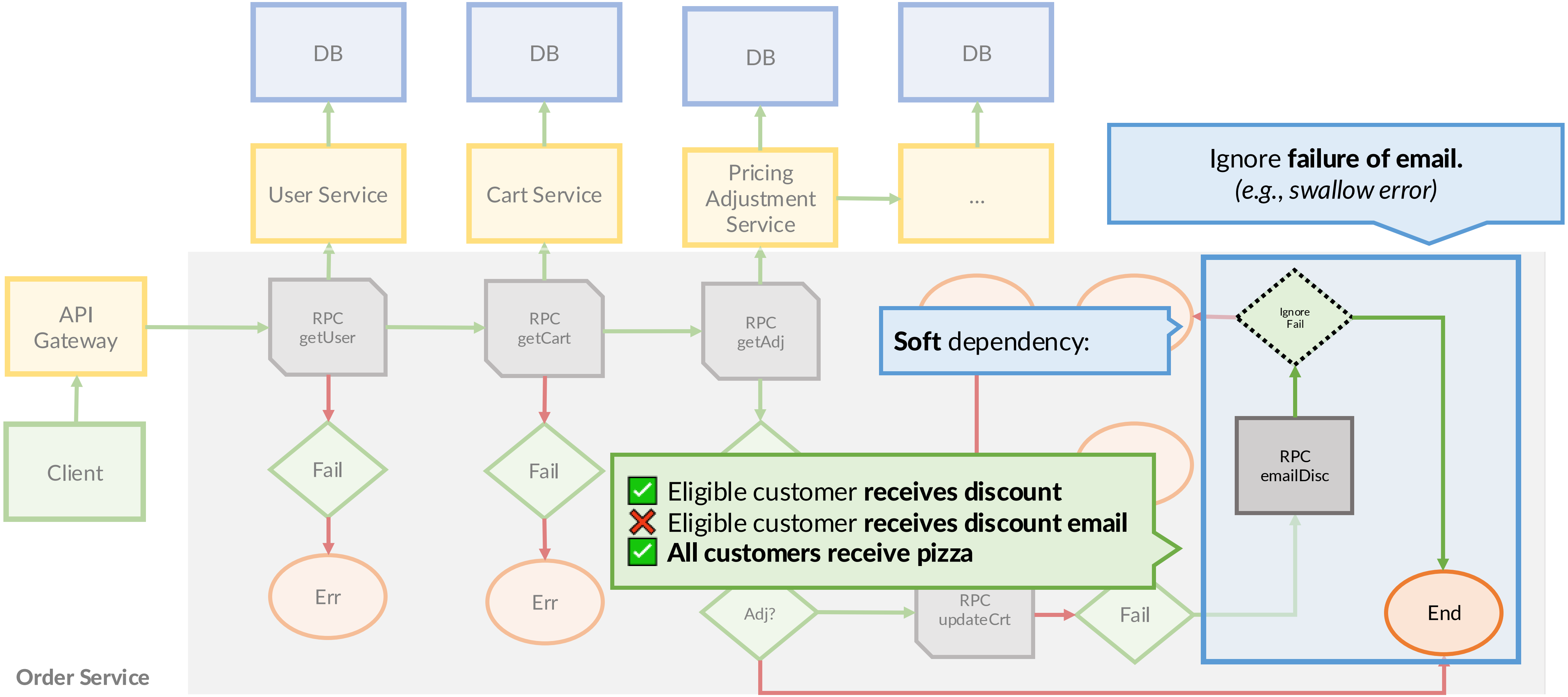
Cannot reason about the RPC in isolation without understanding the broader context.

Corollary:

3. Update Cart (on adjustment > $0) **should continu**

👨‍💻 *Ensure:* ***Ask user who is eligible for an adjustment to try again*** *where the call (may) succeed as user may only be making purchase based on available discount (i.e., first time discount.)*

Purchase: Ignored Soft Dependency Failures

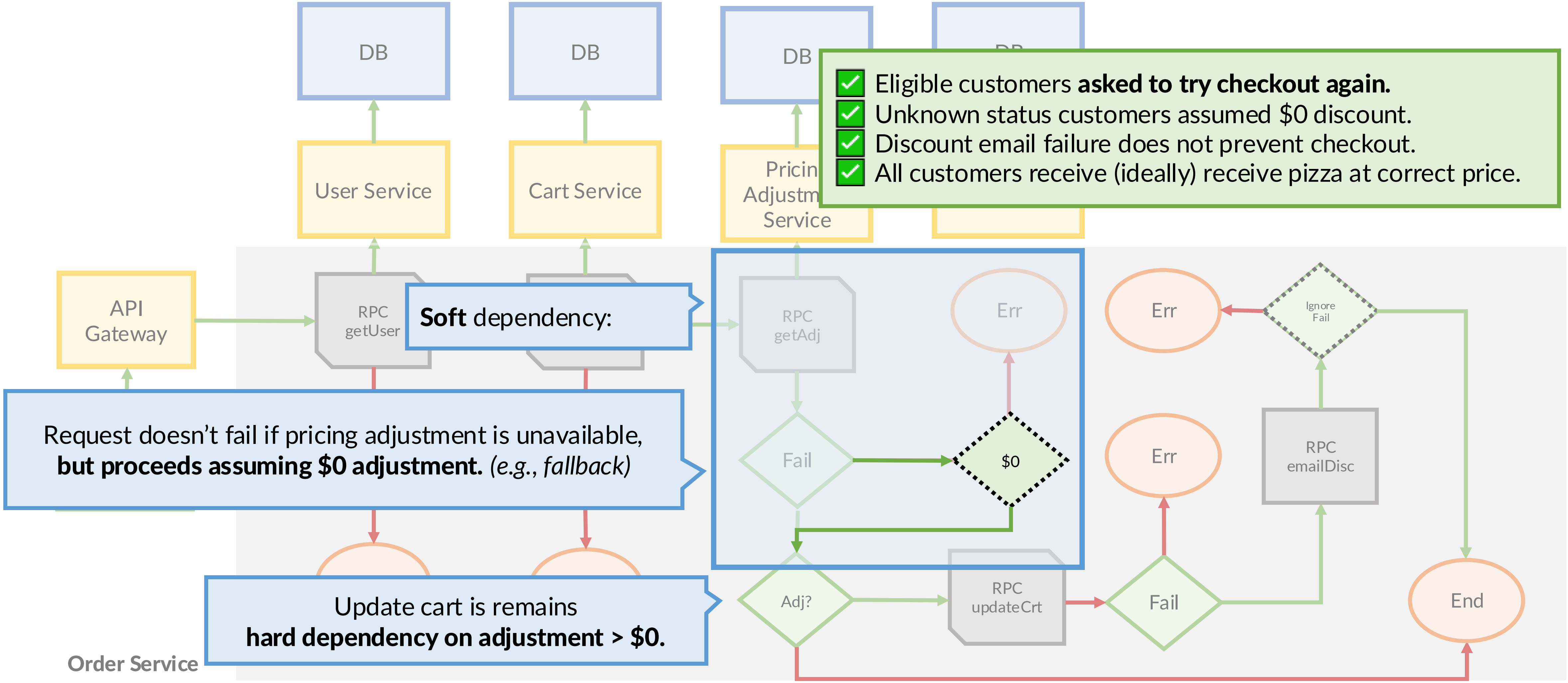


Ignore **failure of email.**
(e.g., swallow error)

Soft dependency:

☑ Eligible customer **receives discount**
☒ Eligible customer **receives discount email**
☑ **All customers receive pizza**

Purchase: Soft Dependencies with Fallbacks



- ✓ Eligible customers **asked to try checkout again.**
- ✓ Unknown status customers assumed $0 discount.
- ✓ Discount email failure does not prevent checkout.
- ✓ All customers receive (ideally) receive pizza at correct price.

Soft dependency:

Request doesn't fail if pricing adjustment is unavailable,
but proceeds assuming $0 adjustment. *(e.g., fallback)*

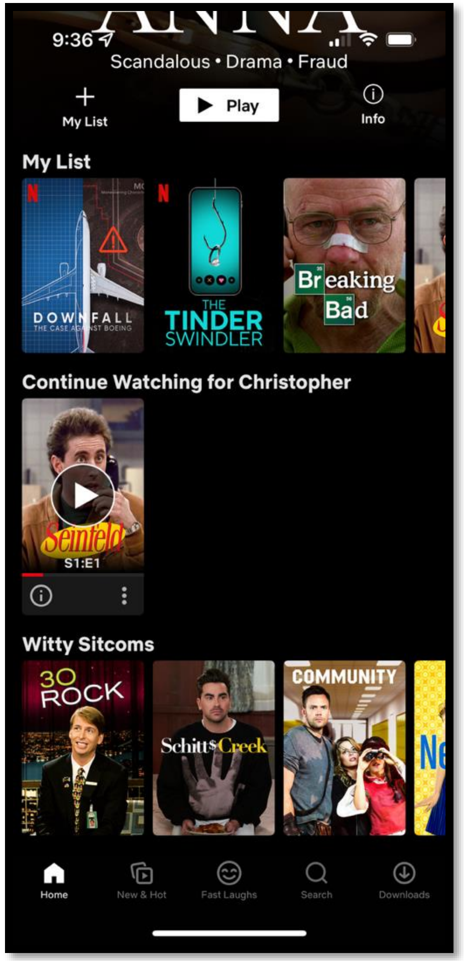
Update cart is remains
hard dependency on adjustment > $0.

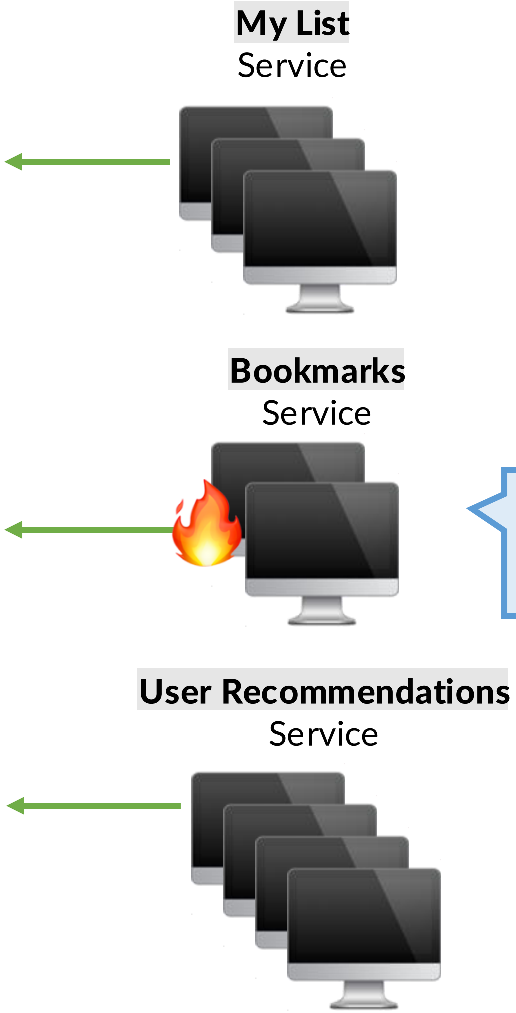
Where to Start: Simple Mocking

Mocking failure:

Simple mocks for network calls
can simulate failure as well as success.

API Gateway
Service



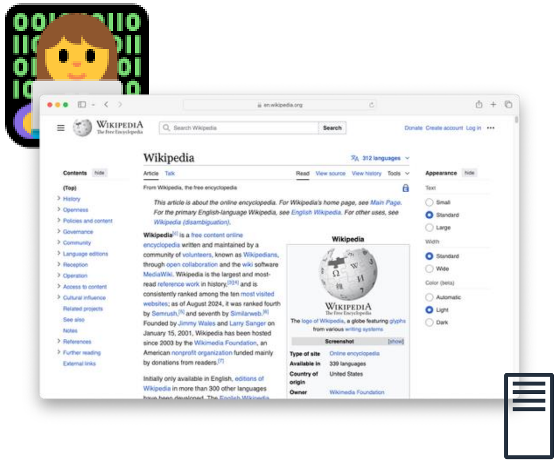


Replace with **mock that returns error.**

Test my API gateway service by **sending it a request to load page.**

Test asserts that **behavior is correct when failure present.**

What About This Software?



en.wikipedia.org

Wikipedia The Free Encyclopedia | Search Wikipedia | Search | Donate Create account Log in

Wikipedia

312 languages

Article Talk | Read View source View history Tools

Contents hide
(Top)
History
Openness
Policies and content
Governance
Community
Language editions
Reception
Operation
Access to content
Cultural influence
Related projects
See also
Notes
References
Further reading
External links

From Wikipedia, the free encyclopedia

This article is about the online encyclopedia. For Wikipedia's home page, see Main Page. For the primary English-language Wikipedia, see English Wikipedia. For other uses, see Wikipedia (disambiguation).

Wikipedia[a] is a free content online encyclopedia written and maintained by a community of volunteers, known as Wikipedians, through open collaboration and the wiki software MediaWiki. Wikipedia is the largest and most-read reference work in history,[3][4] and is consistently ranked among the ten most visited websites; as of August 2024, it was ranked fourth by Semrush,[5] and seventh by Similarweb.[6] Founded by Jimmy Wales and Larry Sanger on January 15, 2001, Wikipedia has been hosted since 2003 by the Wikimedia Foundation, an American nonprofit organization funded mainly by donations from readers.[7]

Initially only available in English, editions of Wikipedia in more than 300 other languages have been developed. The English Wikipedia

Wikipedia

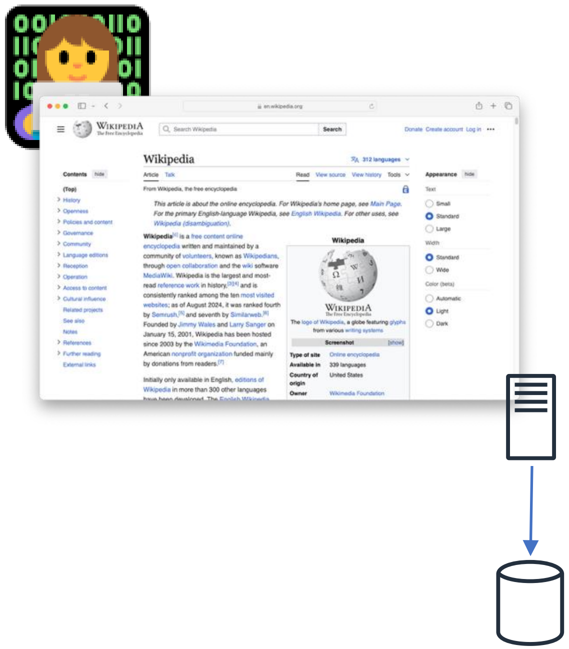
Wikipedia The Free Encyclopedia

The logo of Wikipedia, a globe featuring glyphs from various writing systems

Screenshot [show]

Type of site	Online encyclopedia
Available in	339 languages
Country of origin	United States
Owner	Wikimedia Foundation

Appearance hide
Text
○ Small
● Standard
○ Large
Width
● Standard
○ Wide
Color (beta)
○ Automatic
● Light
○ Dark



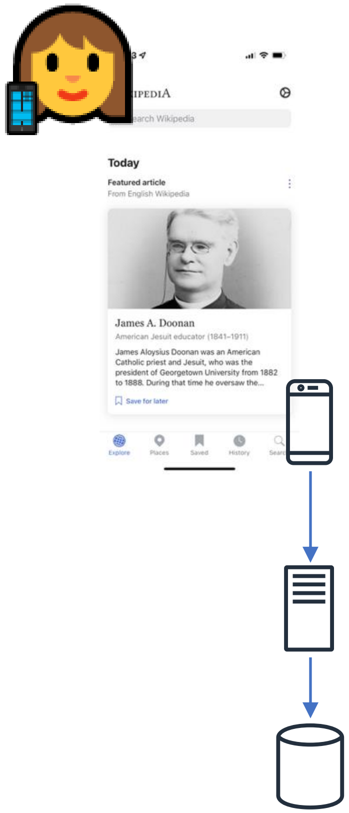
en.wikipedia.org

Wikipedia The Free Encyclopedia | Search Wikipedia | Search | Donate Create account Log in

Wikipedia

From Wikipedia, the free encyclopedia

Wikipedia[a] is a free content online encyclopedia written and maintained by a community of volunteers, known as Wikipedians, through open collaboration and the wiki software MediaWiki.



Wikipedia

Search Wikipedia

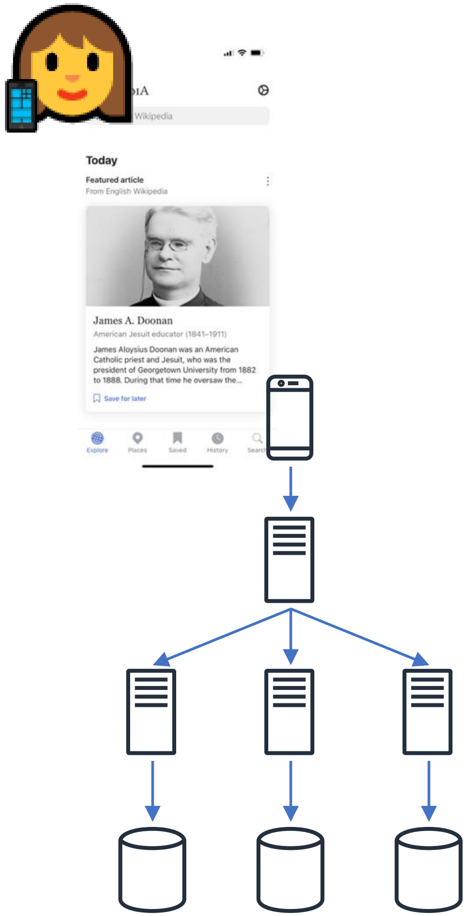
Today

Featured article
From English Wikipedia

James A. Doonan
American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...

Save for later



Today

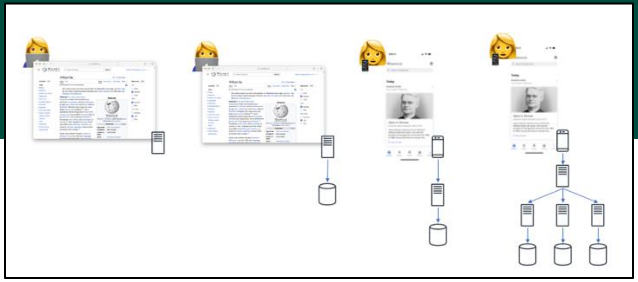
Featured article
From English Wikipedia

James A. Doonan
American Jesuit educator (1841–1911)

James Aloysius Doonan was an American Catholic priest and Jesuit, who was the president of Georgetown University from 1882 to 1888. During that time he oversaw the...

Save for later

Key Takeaways



1. **Controlled rollouts** with feature flags and robust observability are critical risk minimization.
2. **Backwards compatibility** is essential for safe rollouts, especially in microservice architectures.
3. Always ensure the **ability to rollback** and have a **clear rollout/rollback plan.**
4. **Testing** must cover both legacy and new behaviors, including with feature flags on and off.

When dealing with **soft dependencies** in a microservice application:

1. **Test application** flows E2E thoroughly for the **desired outcomes without failure present.**
2. Use **mocks or fakes to simulate failure** to understand if your application continues to do the correct thing under failure **with the same set of test cases.**

In Conclusion

Identified the core challenges in making changes to software safely and reliably in a cloud application.

Examined several authorship, testing, and rollout strategies to release code safely.

Practiced identifying problematic changes and how to go about **making changes safely.**

Any **Questions?**