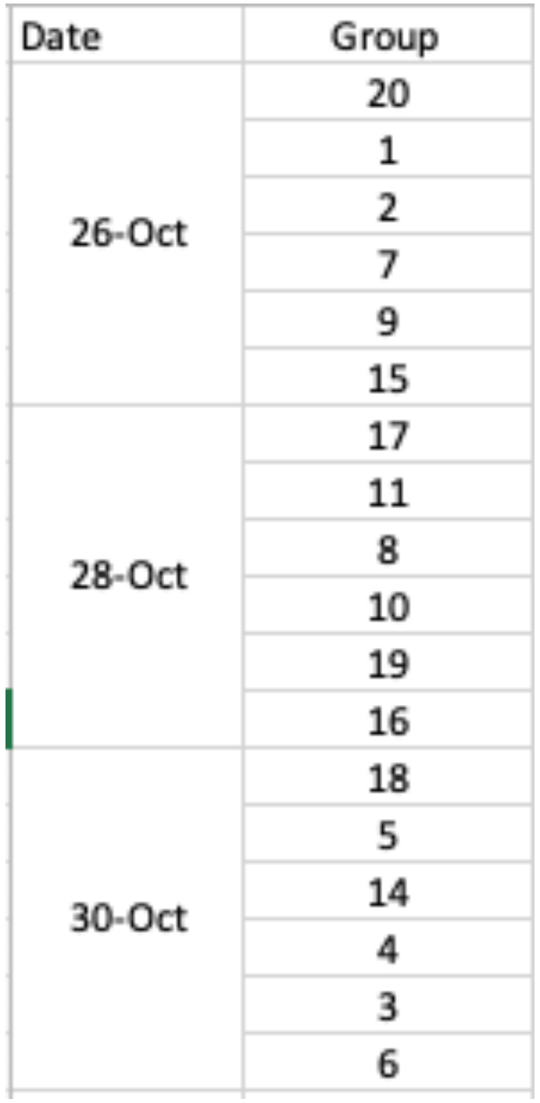
ECE444: Software Engineering

Architecture4: Styles and Hypes

Shurui Zhou

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Midterm Presentation



Date	Group
26-Oct	20
	1
	2
	7
	9
	15
28-Oct	17
	11
	8
	10
	19
	16
30-Oct	18
	5
	14
	4
	3
	6

- If your group cannot present in the lecture, please let me know and send me the 8min video by 10/25.

Spike in Agile

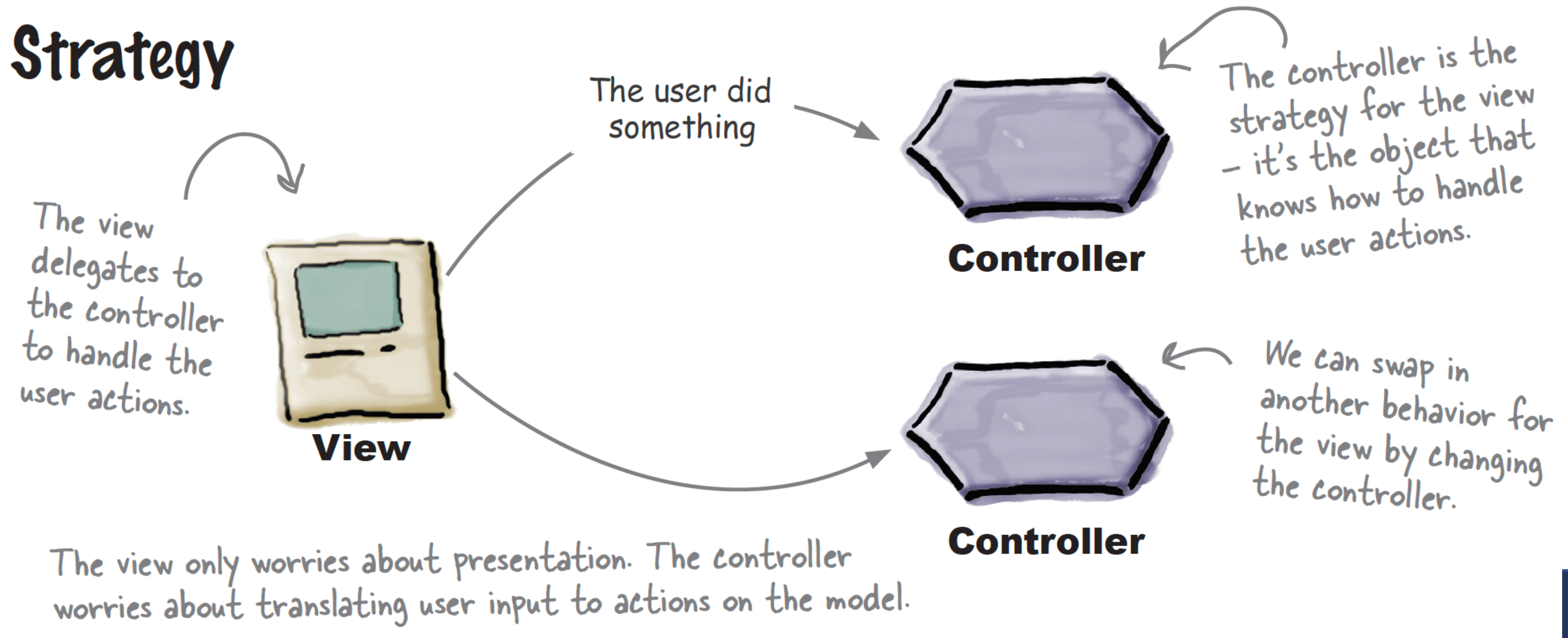
- A special type of user story that is used to gain the knowledge necessary to reduce the risk of a technical approach, better understand a requirement, or increase the reliability of a story estimate
- Has a maximum time-box size
- For example:
  - The team may not have knowledge of a new technology, and spikes may be used for basic research to ensure the feasibility of the new technology (domain or new approach).
  - A story requires to be implemented using a 3rd party library with API that is poorly written and documented.
  - The story may contain significant technical risk, and the team may have to do some experiments or prototypes to gain confidence in a technological approach that may allow them to commit the user story to some future timebox.

Classification of patterns

- **Creational patterns**
  - **Singleton**
  - **Factory Method**
- **Structural patterns**
  - **Composite**
- **Behavioral patterns**
  - **Strategy**
  - **Observer**

MVC Architecture

- Model – Observer Pattern
- View – Composite + Strategy
- Controller -- Strategy Pattern


Strategy
The user did something
The controller is the strategy for the view – it's the object that knows how to handle the user actions.
The view delegates to the controller to handle the user actions.
View
Controller
We can swap in another behavior for the view by changing the controller.
Controller
The view only worries about presentation. The controller worries about translating user input to actions on the model.

Cargo cult programming

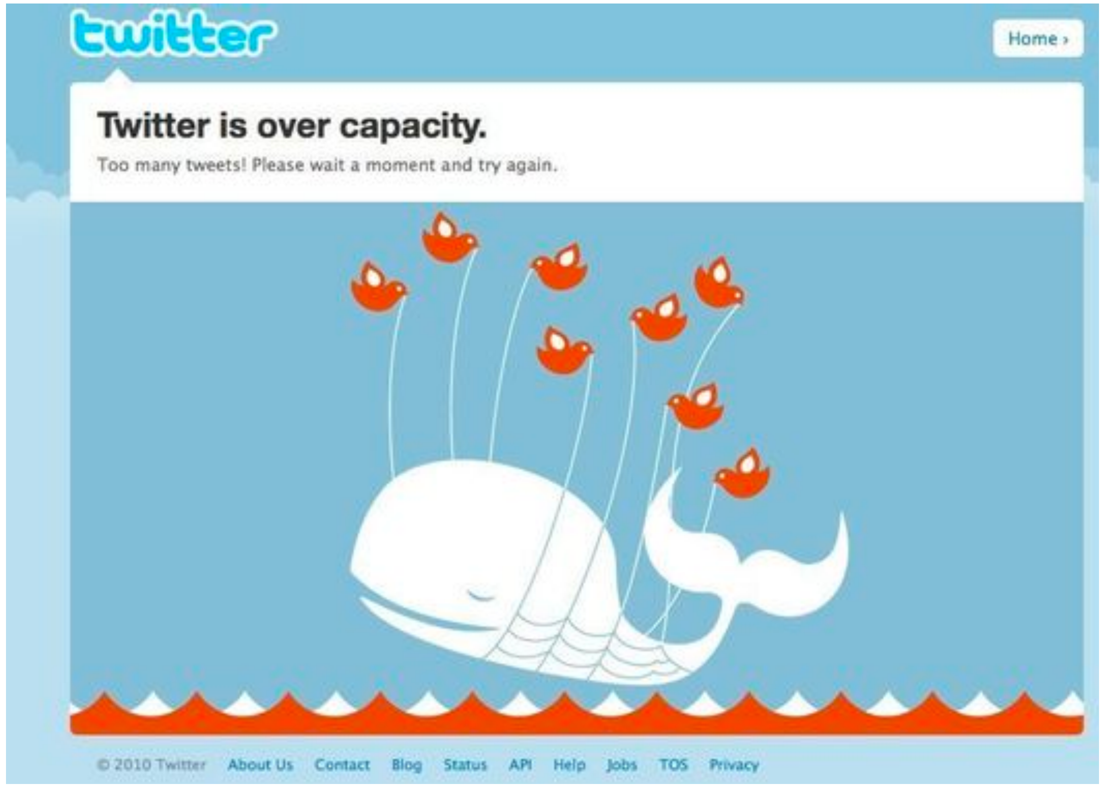
Are SOLID principles Cargo Cult?

It looks like a plane, but will it fly?

https://blog.ndepend.com/are-solid-principles-cargo-cult/

Learning Goals

- Understand history of Microservices
- Reason about tradeoffs of Microservices architectures.



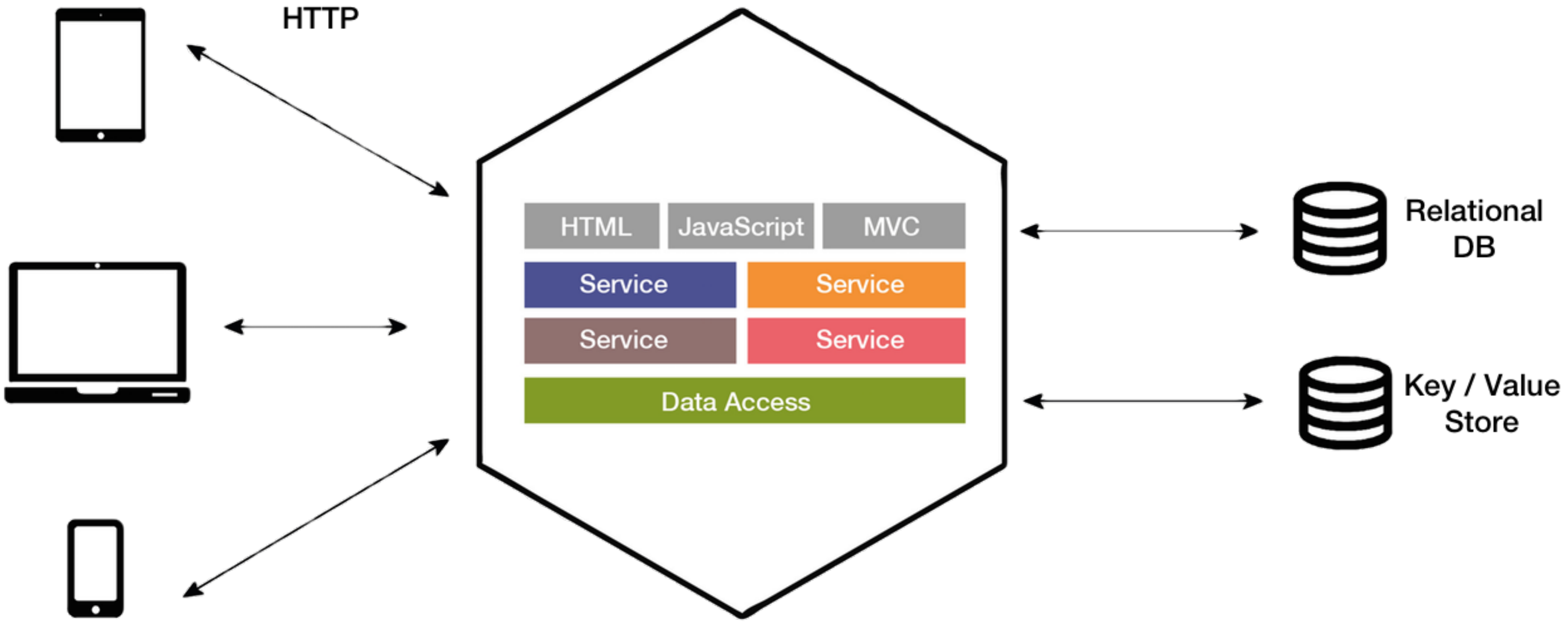
Twitter is over capacity.

Too many tweets! Please wait a moment and try again.

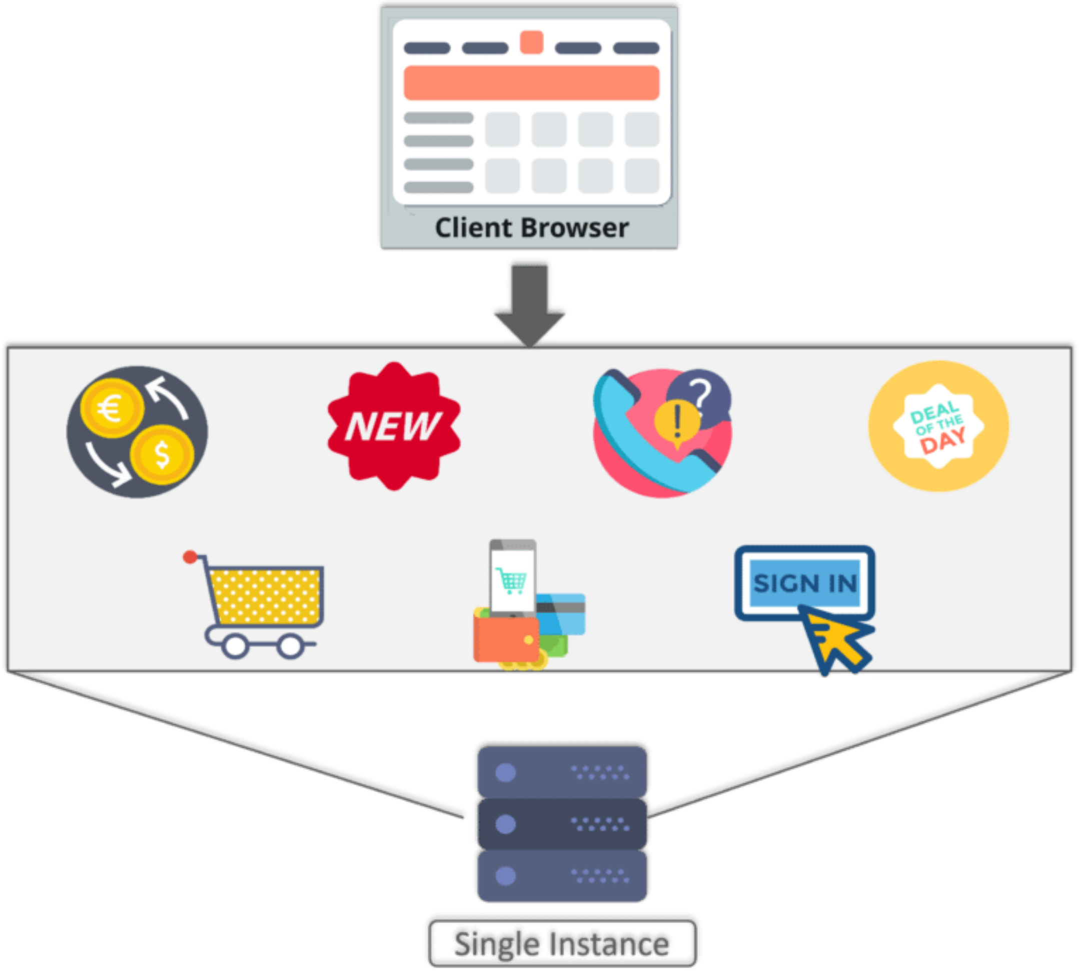
© 2010 Twitter About Us Contact Blog Status API Help Jobs TOS Privacy

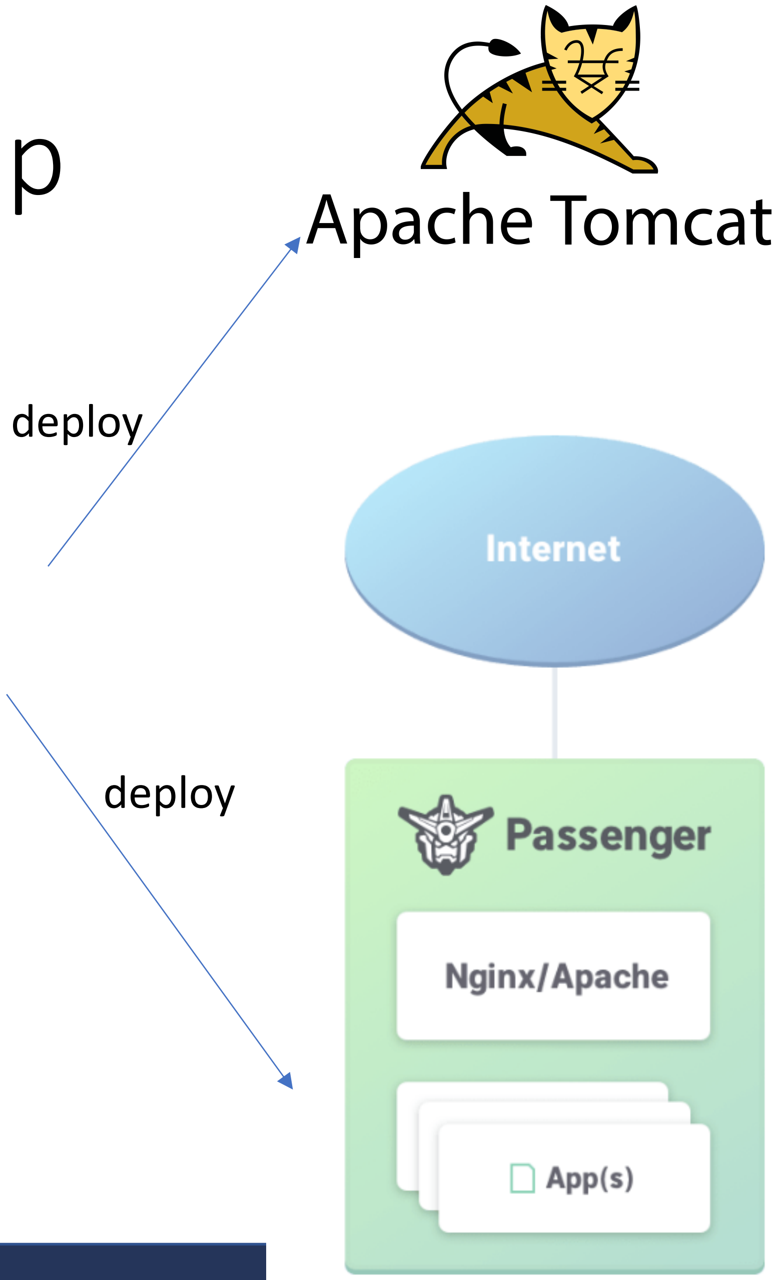
"After that experience, we determined we **needed to step back**. We then determined we needed to **re-architect** the site to support the continued growth of Twitter and to keep it running smoothly."

Monolithic Architecture



Example: a shopping cart app

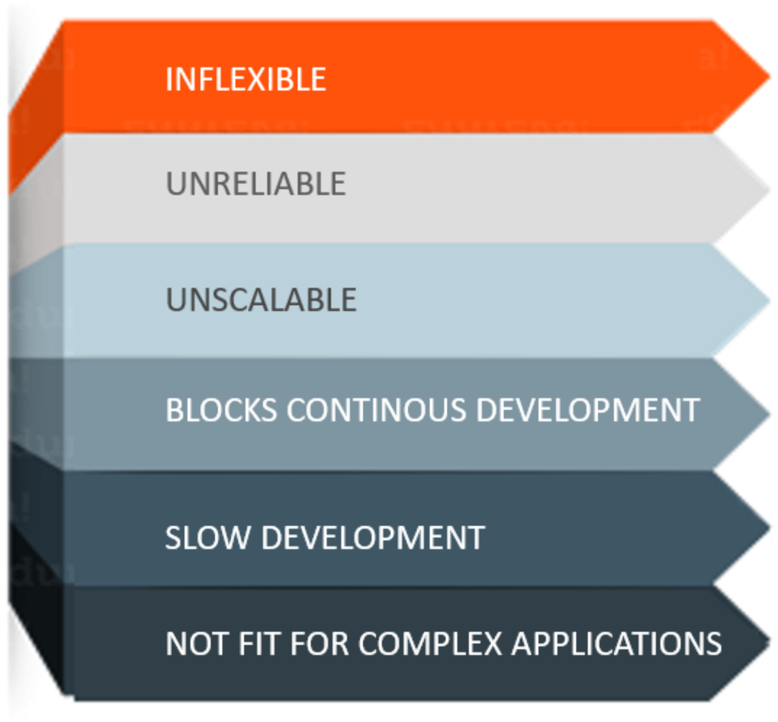




Monolithic Architecture Benefits

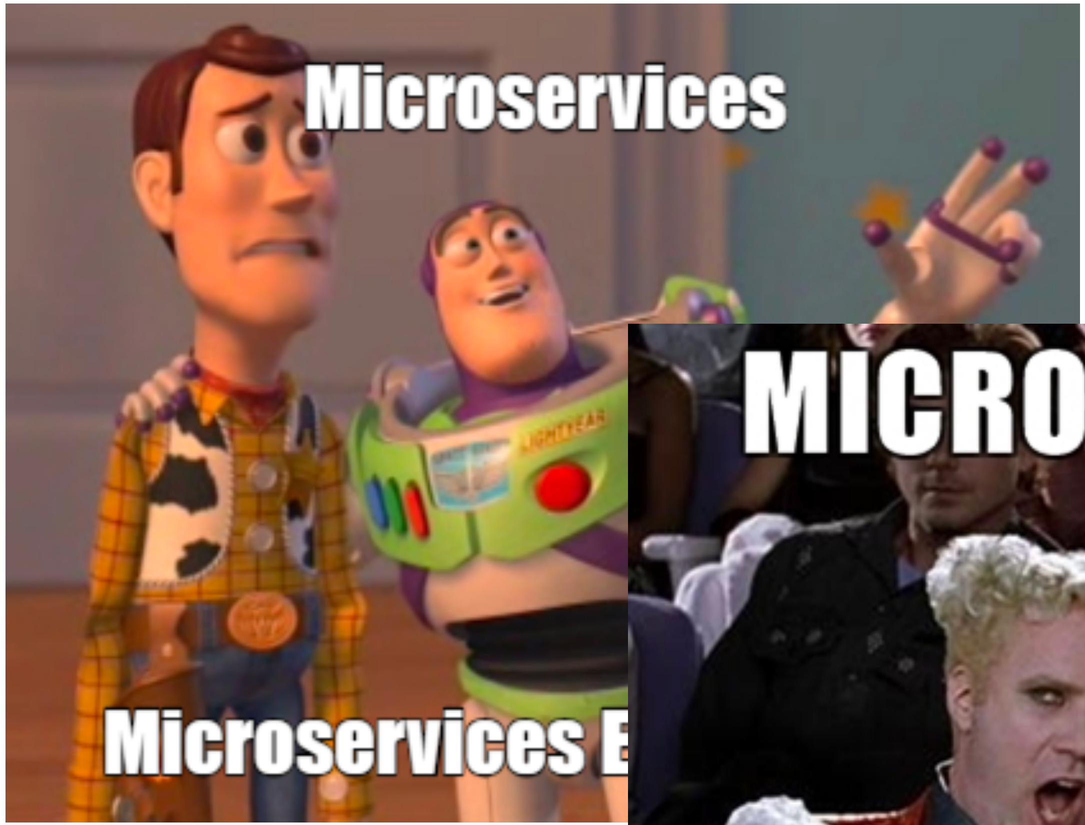
- Simple to develop
- Simple to deploy
- Simple to scale

Challenges of Monolithic Architecture


INFLEXIBLE
UNRELIABLE
UNSCALABLE
BLOCKS CONTINOUS DEVELOPMENT
SLOW DEVELOPMENT
NOT FIT FOR COMPLEX APPLICATIONS

- ***Inflexible*** *— Monolithic applications cannot be built using different technologies*
- ***Unreliable*** *— Even if one feature of the system does not work, then the entire system does not work*
- ***Unscalable*** *— Applications cannot be scaled easily since each time the application needs to be updated, the complete system has to be rebuilt*
- ***Blocks Continuous Development*** *— Many features of the applications cannot be built and deployed at the same time*
- ***Slow Development*** *— Development in monolithic applications take lot of time to be built since each and every feature has to be built one after the other*
- ***Not Fit For Complex Applications —*** *Features of complex applications have tightly coupled dependencies*

Microservices


Microservices
Microservices E

WHAT ARE MICROSERVICES?
OR ANTS???!
makeameme.org

MICROSERVICES...
...SO HOT RIGHT NOW
memegenerator.net

https://microservices.io/articles/whoisusingmicroservices.html

shopify

COMCAST

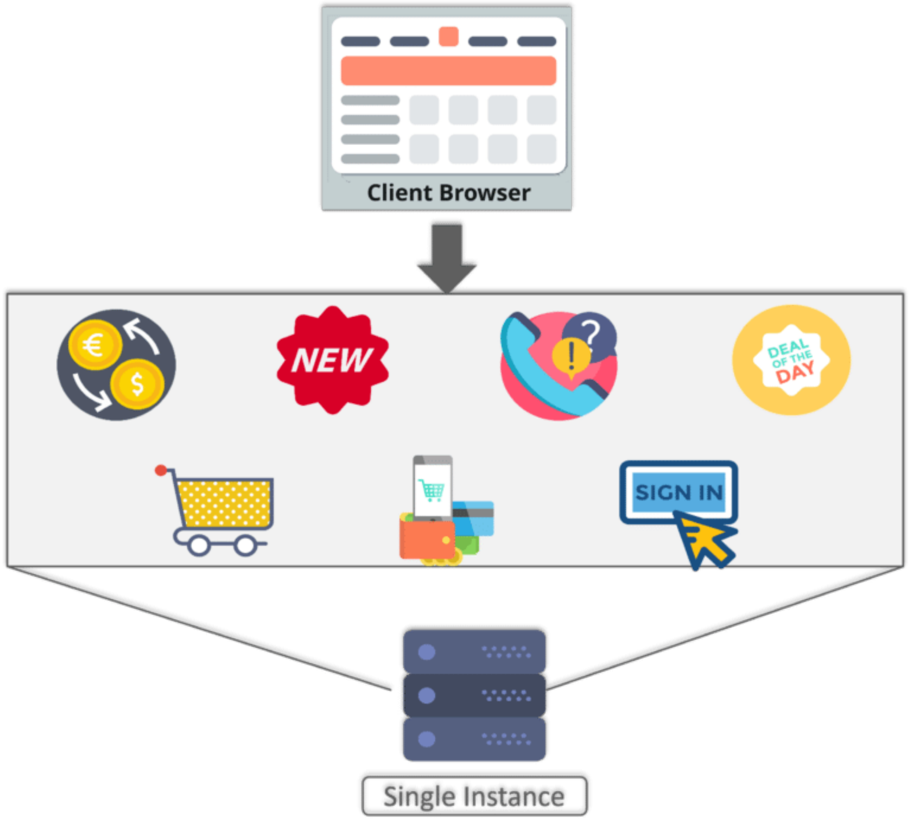
NETFLIX

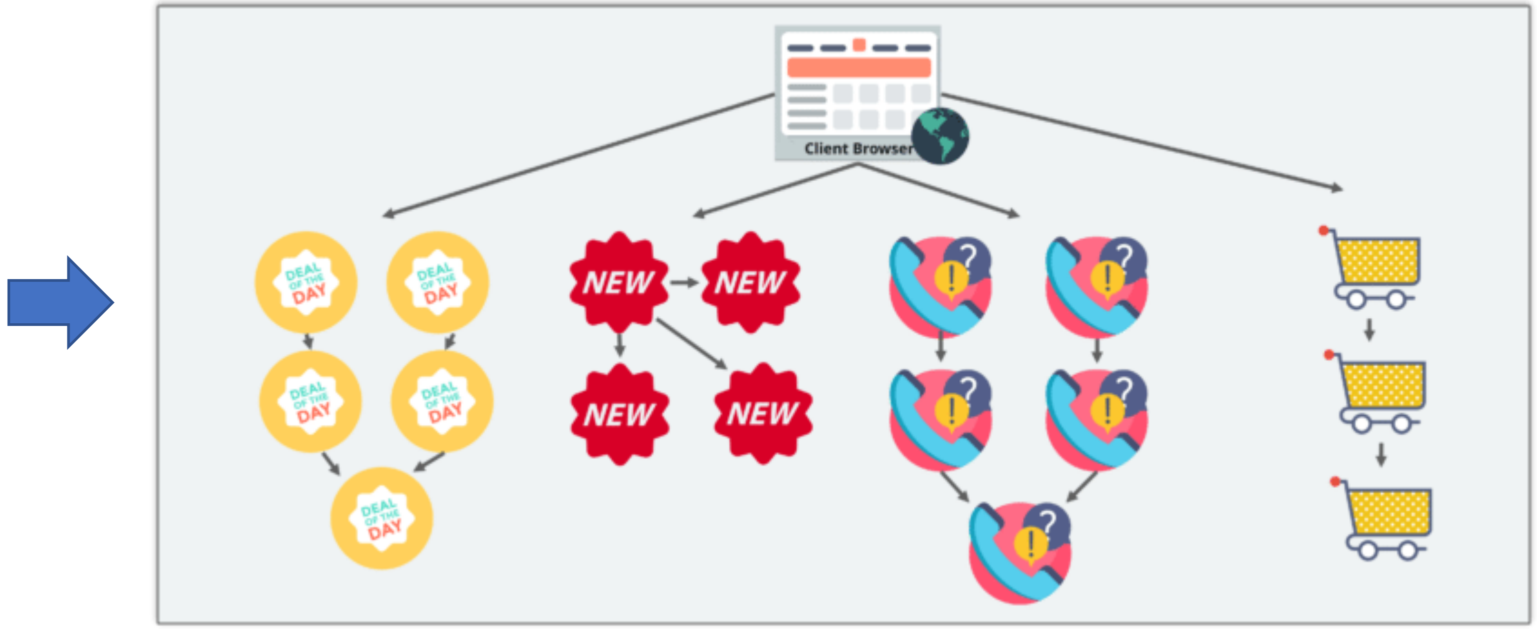
ebay

UBER

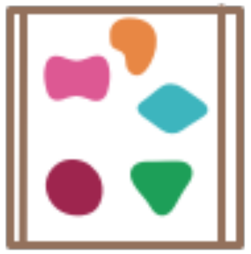
GROUPON

Use case: Shopping Cart Application

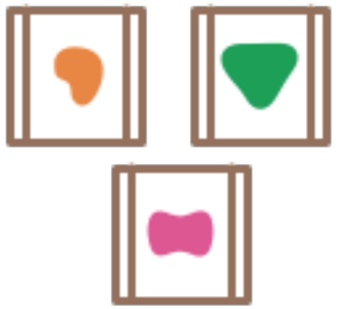




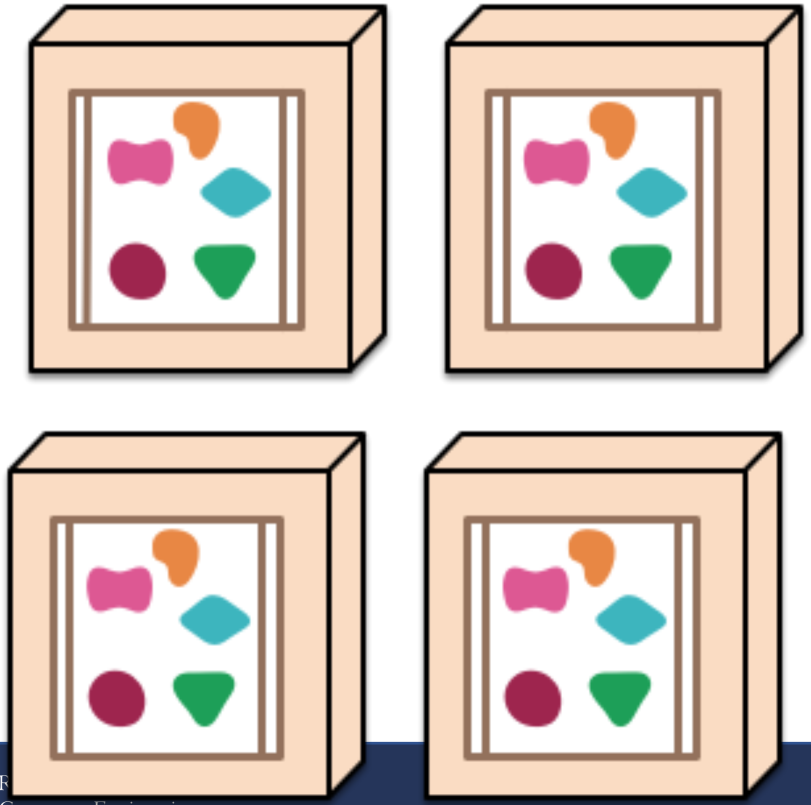
A monolithic application puts all its functionality into a single process...



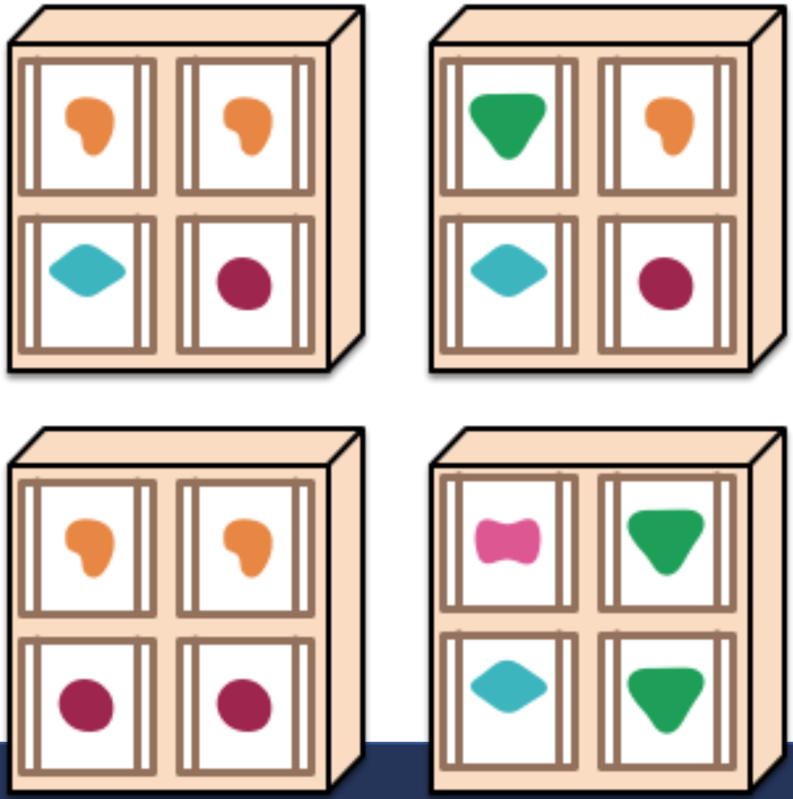
A microservices architecture puts each element of functionality into a separate service...



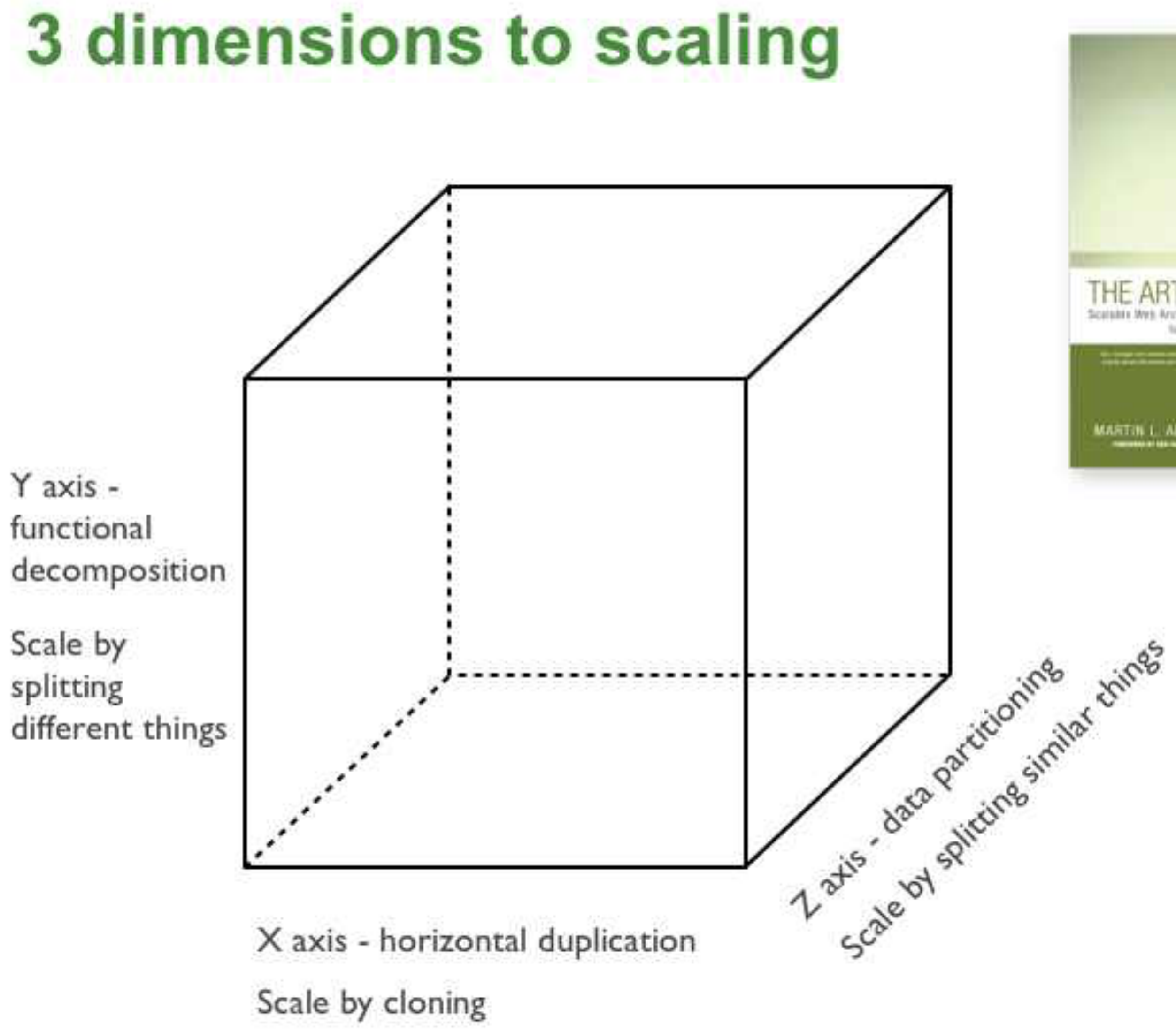
... and scales by replicating the monolith on multiple servers

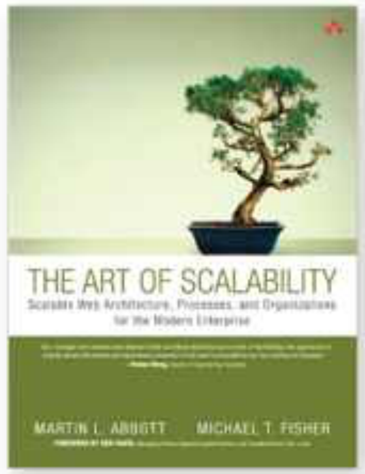


... and scales by distributing these services across servers, replicating as needed.

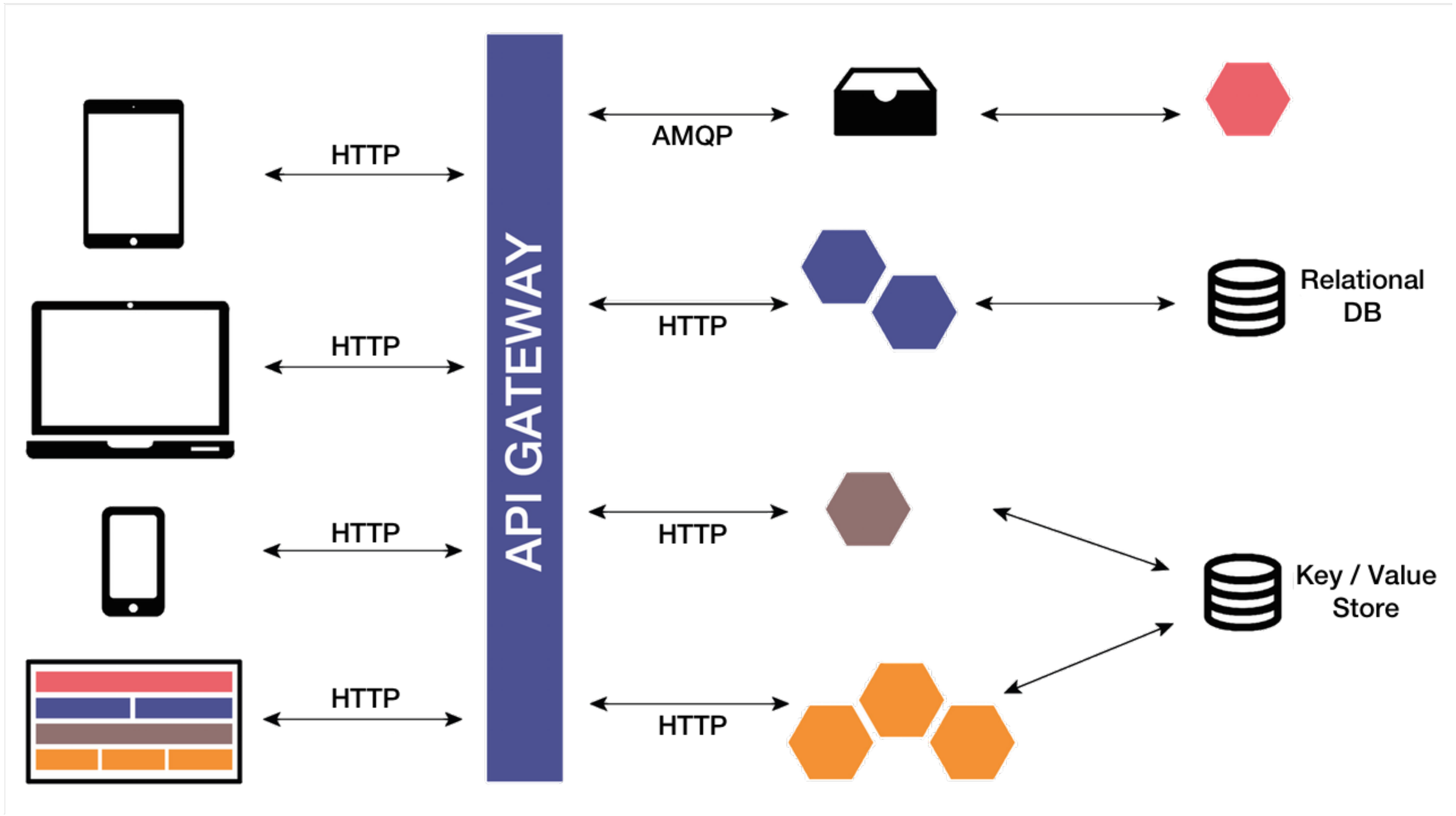


The scale Cube

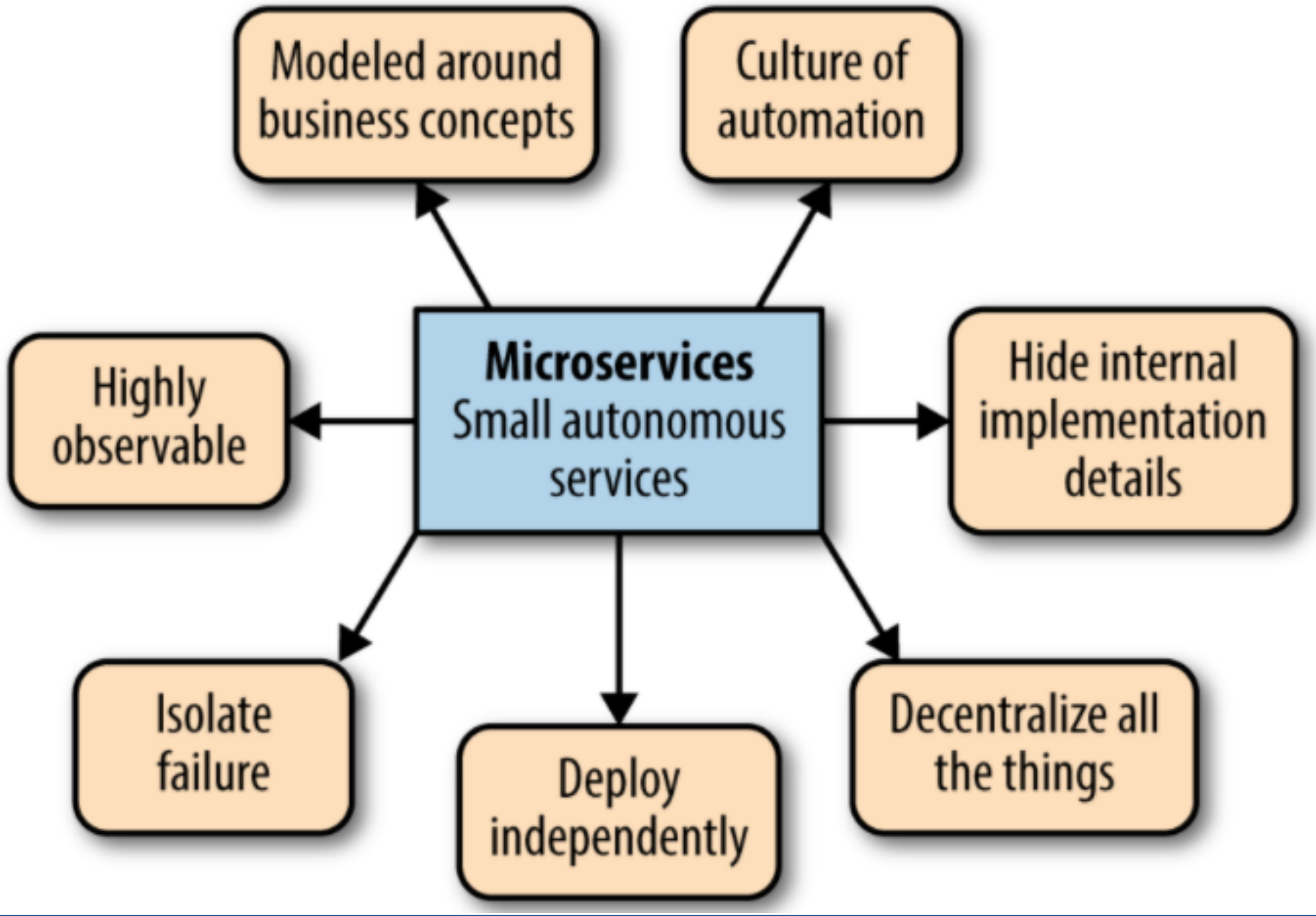

3 dimensions to scaling
Y axis - functional decomposition
Scale by splitting different things
X axis - horizontal duplication
Scale by cloning
Z axis - data partitioning
Scale by splitting similar things



- X-axis: running multiple copies of an application behind a load balancer.
- Y-axis: split the app into services
  - Verb-based
  - Noun-based
- Z-axis: each server is responsible for only a subset of the data.

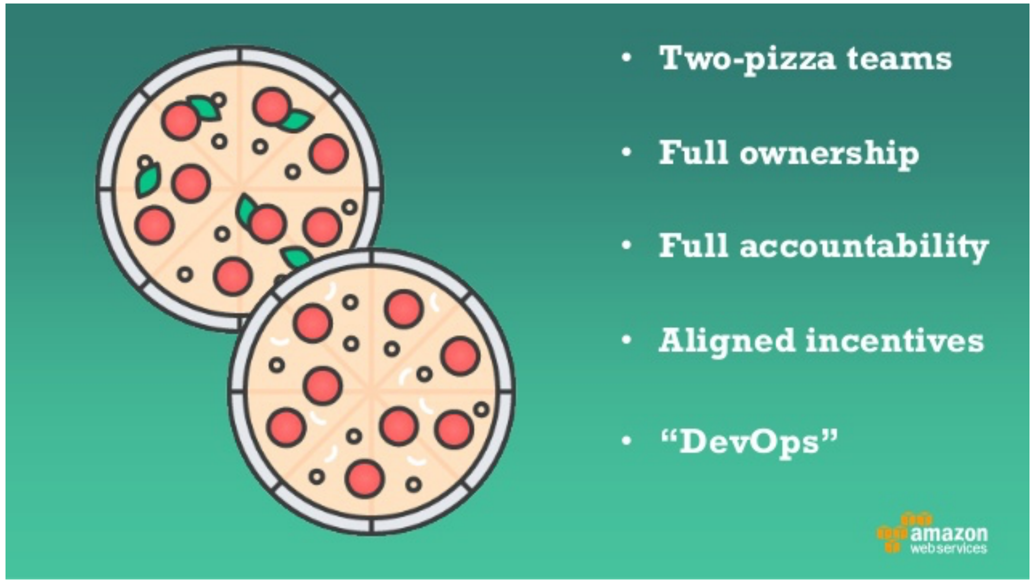

HTTP
HTTP
HTTP
HTTP
API GATEWAY
AMQP
HTTP
HTTP
HTTP
Relational DB
Key / Value Store

Principle of Microservices


Modeled around business concepts
Culture of automation
Highly observable
Microservices Small autonomous services
Hide internal implementation details
Isolate failure
Deploy independently
Decentralize all the things

Benefits of Microservices

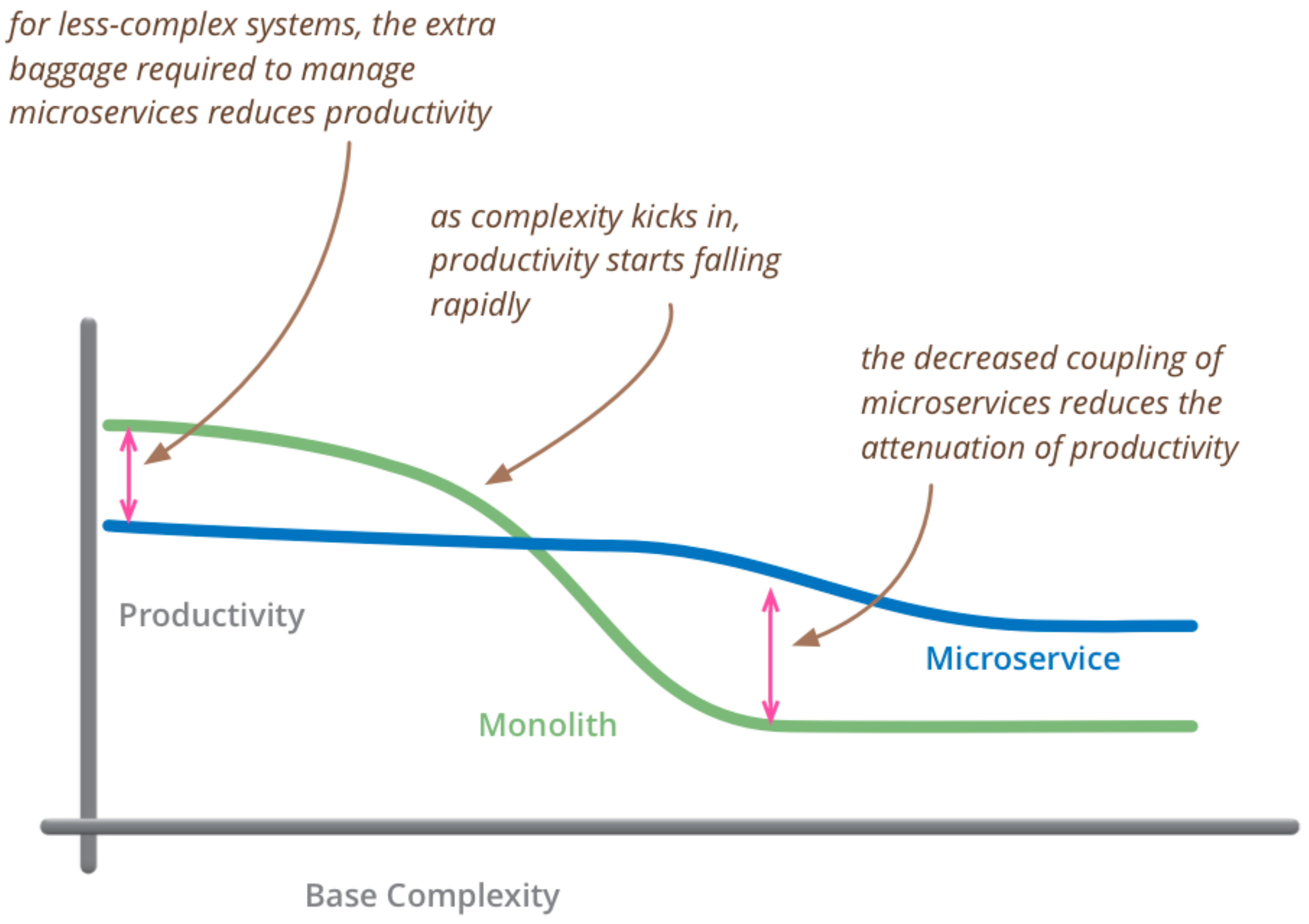
- Faster and simpler deployments and rollbacks
- Elimination of long-term commitment to a single technology stack
- Improved fault isolation
- Independently scalable services
- Technology diversity
- Ability to write new features as plugins



Drawbacks of Microservices

- Increased network communication
- Serialization between microservices
- Additional complexity in testing a distributed system
- Increased complexity in deployment

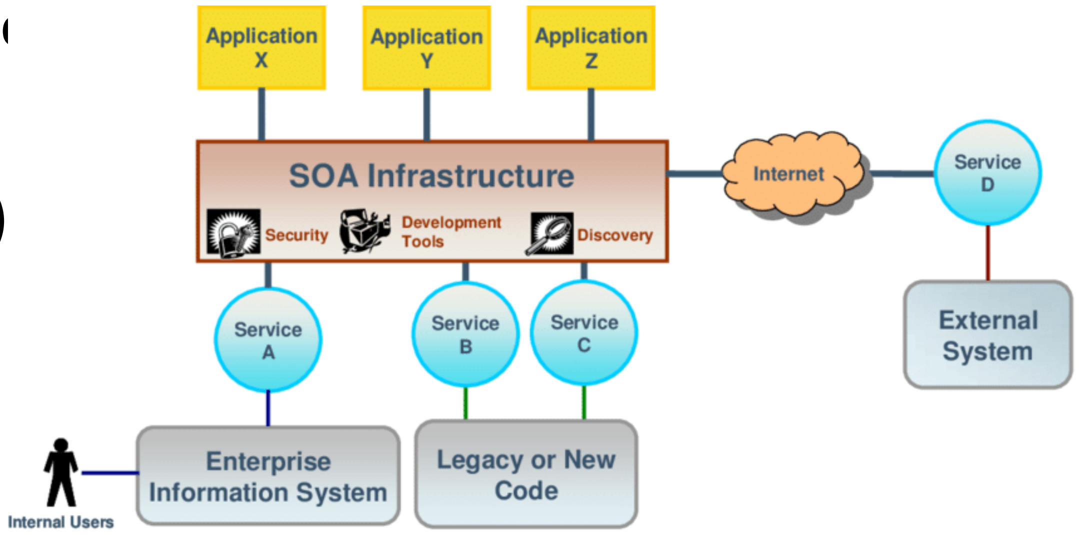
Microservies overhead



but remember the skill of the team will outweigh any monolith/microservice choice

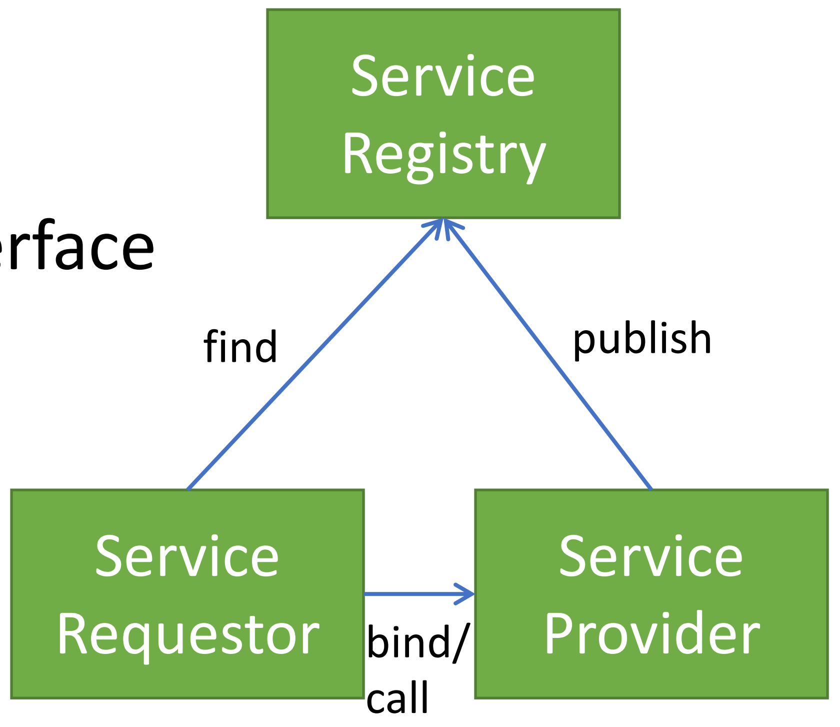
Broker Pattern

- A collection of services distributed across multiple servers
- Separates users of services (clients) from providers of services (servers) by inserting an intermediary, called a *broker*
- Benefit: modifiability, availability, performanc
- Downside: add complexity, latency
- Example: Service-Oriented Architecture (SOA)



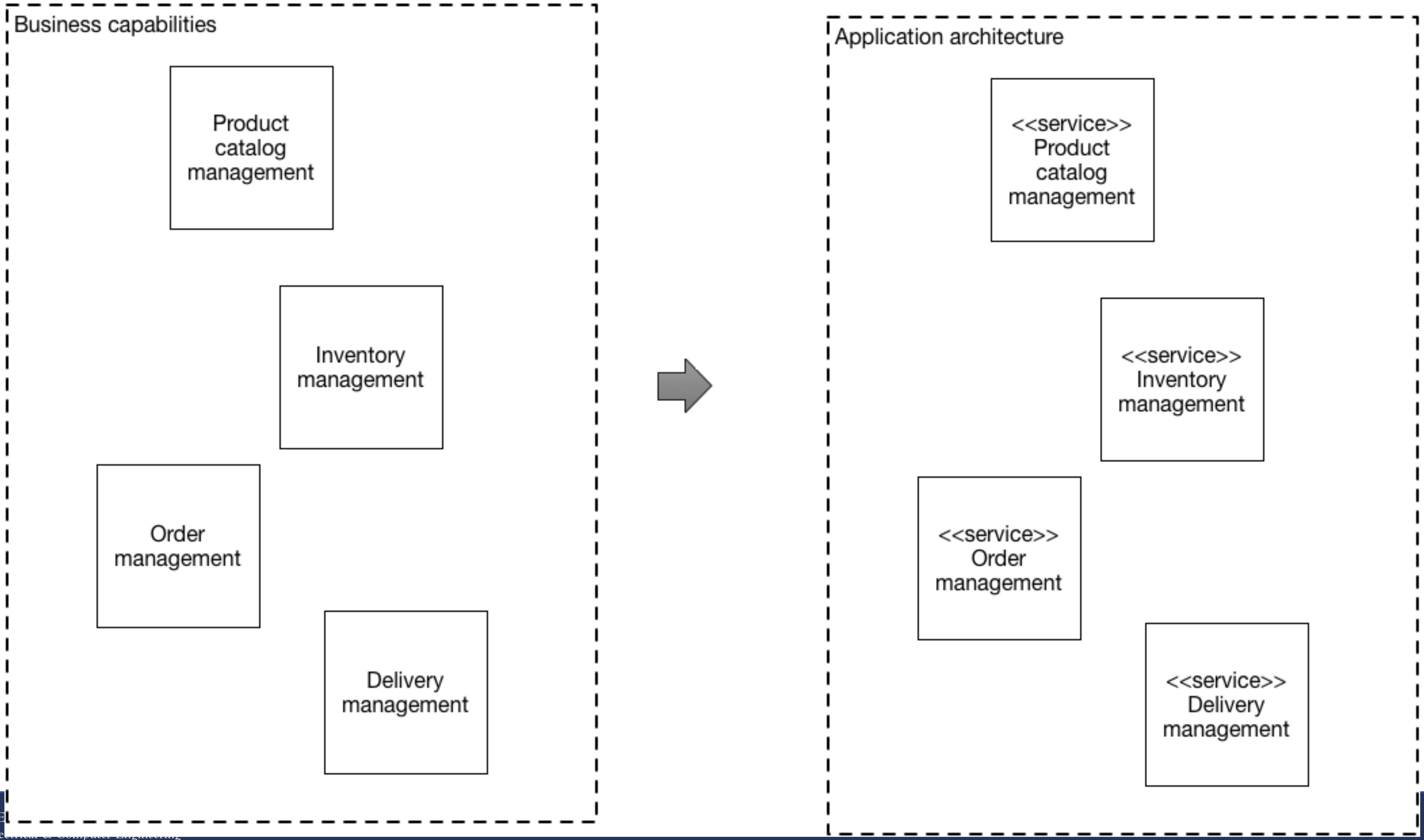
SOA

- Service: self-contained functionality
- Remote invocation, language-independent interface
- Dynamic lookup possible
- Often used to wrap legacy systems



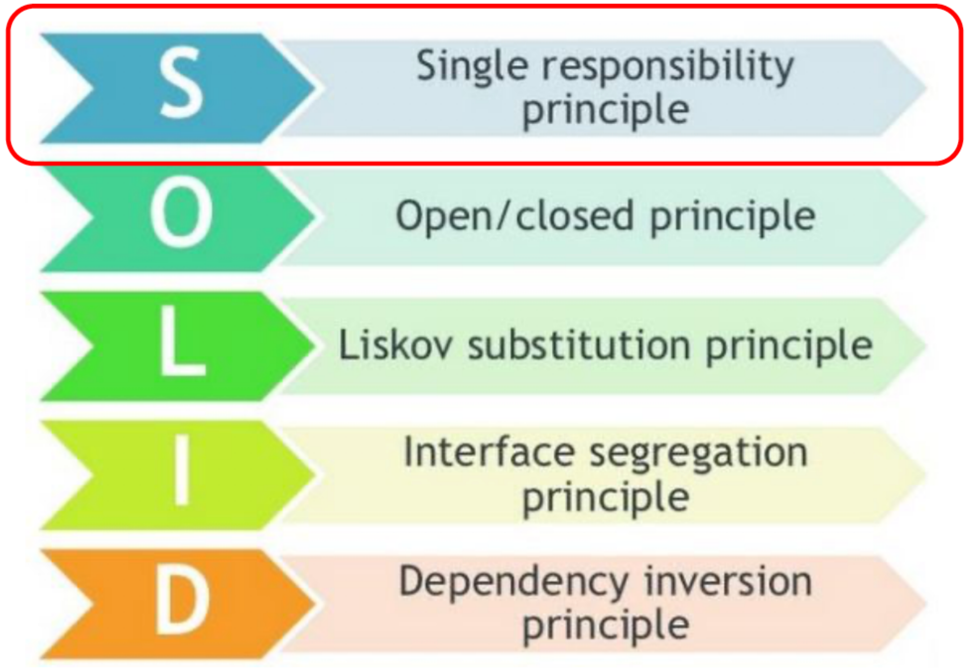
How to decompose the application into services?

- Decompose by business capability
- Decompose by verb or use case
- Decompose by by nouns or resources


Business capabilities
Product catalog management
Inventory management
Order management
Delivery management
Application architecture
<<service>> Product catalog management
<<service>> Inventory management
<<service>> Order management
<<service>> Delivery management

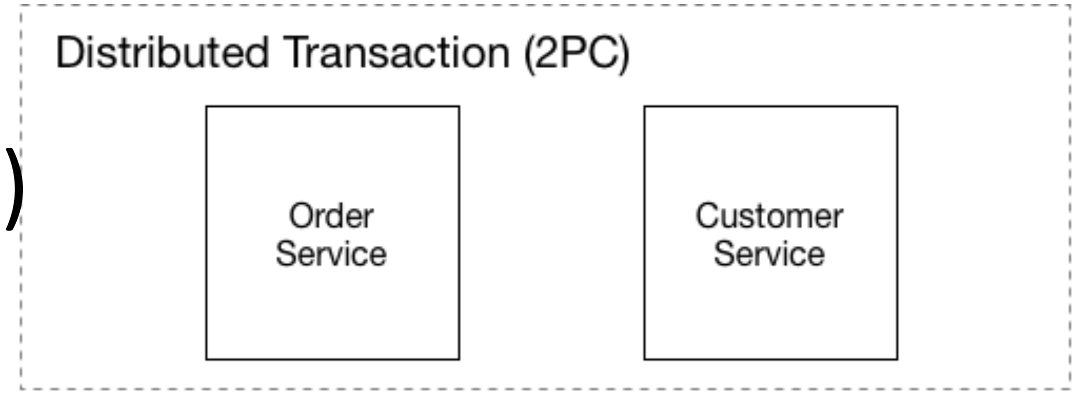
How to decompose the application into services?

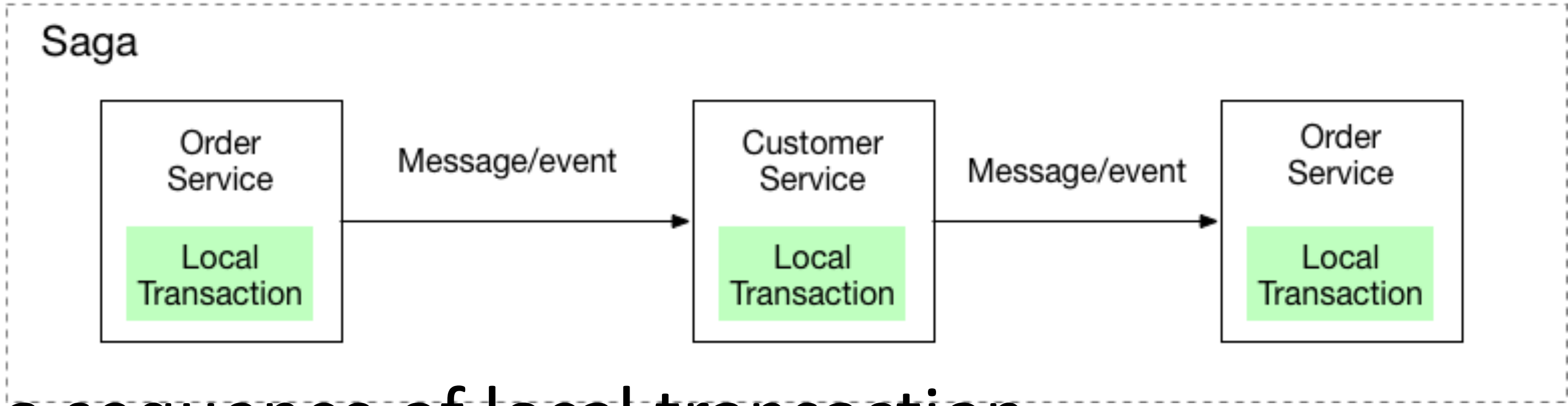
- Decompose by business capability
- Decompose by verb or use case
- Decompose by by nouns or resources


S
Single responsibility principle
O
Open/closed principle
L
Liskov substitution principle
I
Interface segregation principle
D
Dependency inversion principle

How to maintain data consistency?

- ✗ 2PC (Two-phase commit)
- ✓ Saga Pattern





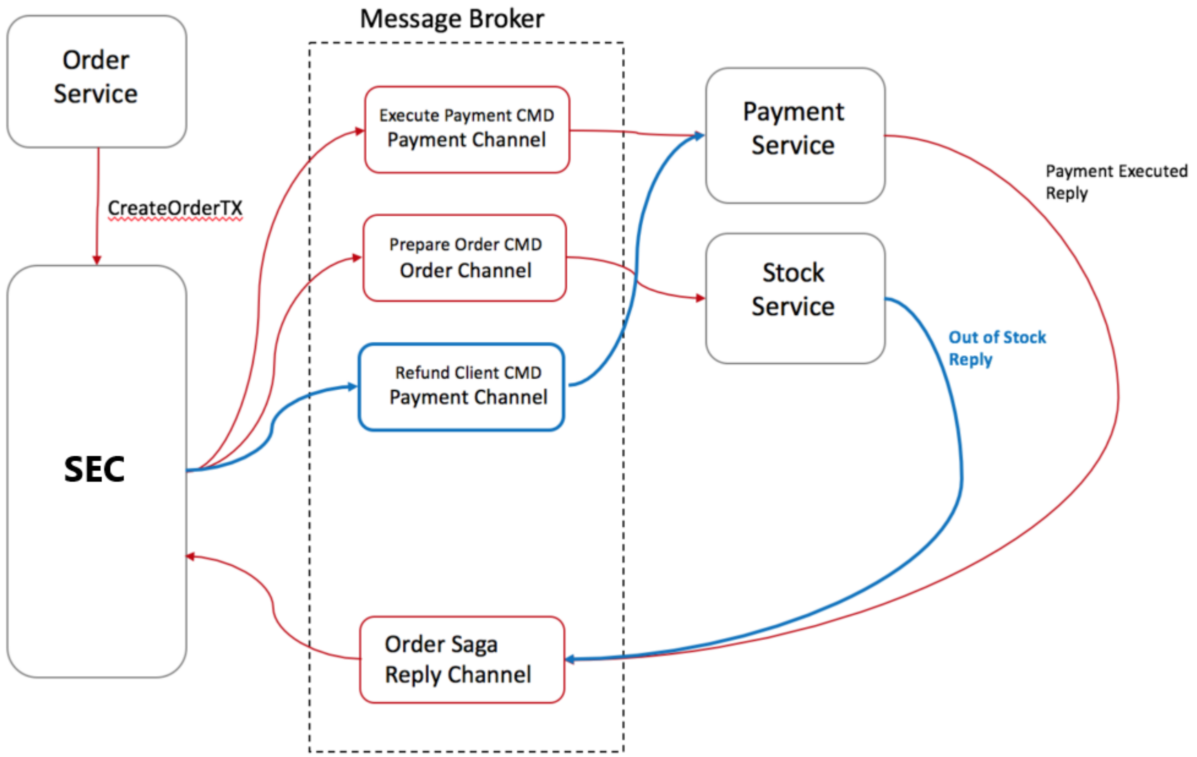
saga – a sequence of local transaction

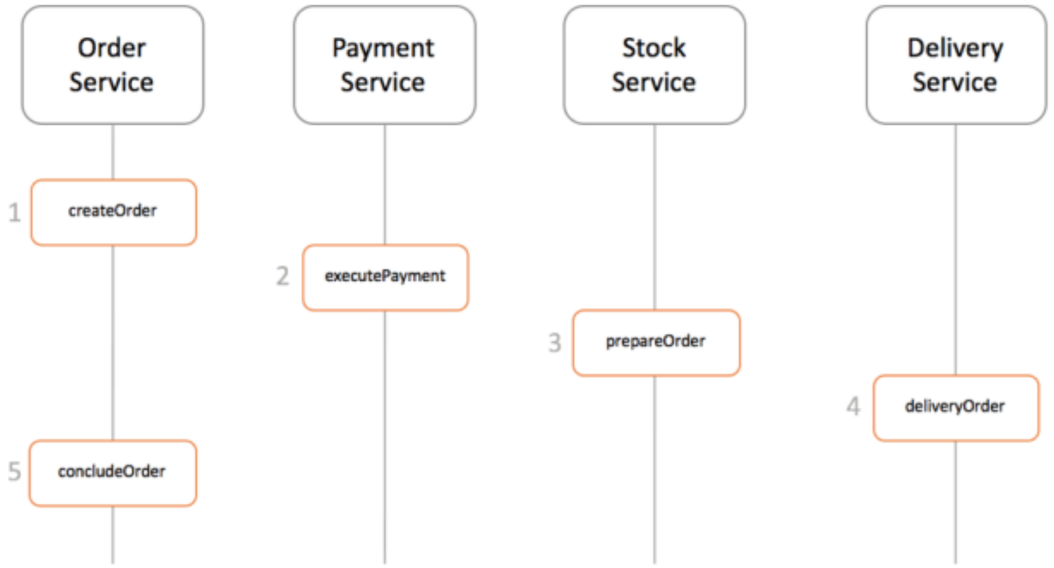
Saga Pattern

The master process called “**Saga Execution Coordinator**” or SEC.

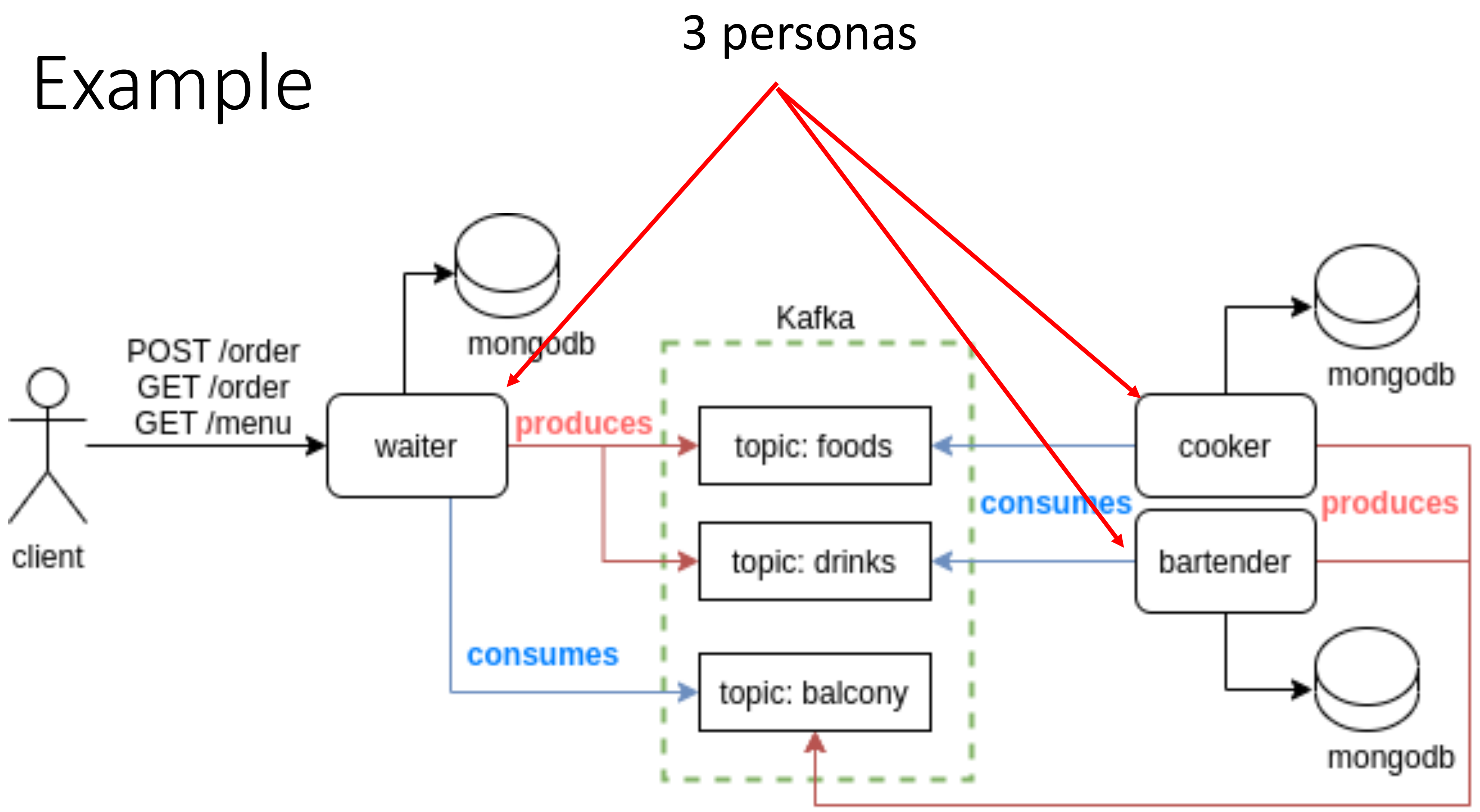
- two ways to achieve sagas
  - Choreography : each local transaction publishes domain events that trigger local transactions in other services.
  - Orchestration : an orchestrator (object) tells the participants what local transactions to execute.

Orchestration





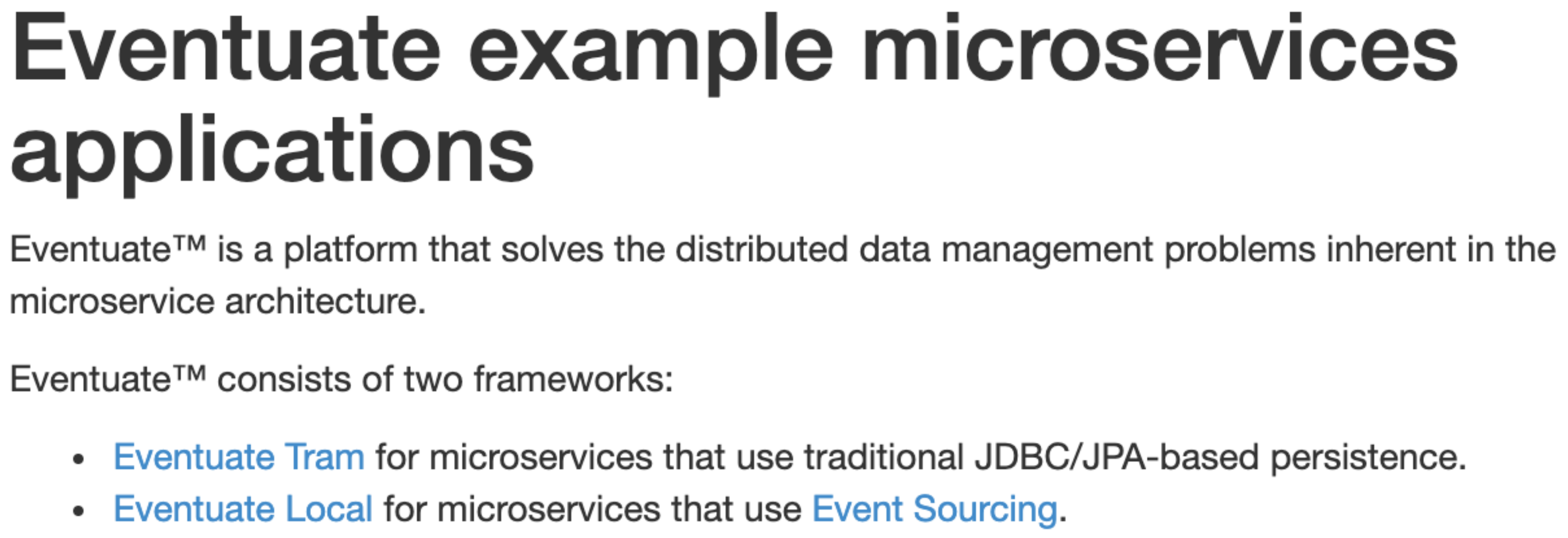
Example



https://github.com/victoramsantos/saga-pattern-example

Other examples and platforms

eventuate



Eventuate example microservices applications

Eventuate™ is a platform that solves the distributed data management problems inherent in the microservice architecture.

Eventuate™ consists of two frameworks:

- Eventuate Tram for microservices that use traditional JDBC/JPA-based persistence.
- Eventuate Local for microservices that use Event Sourcing.

How are services packaged and deployed?

- Container
- Serverless deployment
- Platform as a Service (PaaS)

CLOUD FOUNDRY

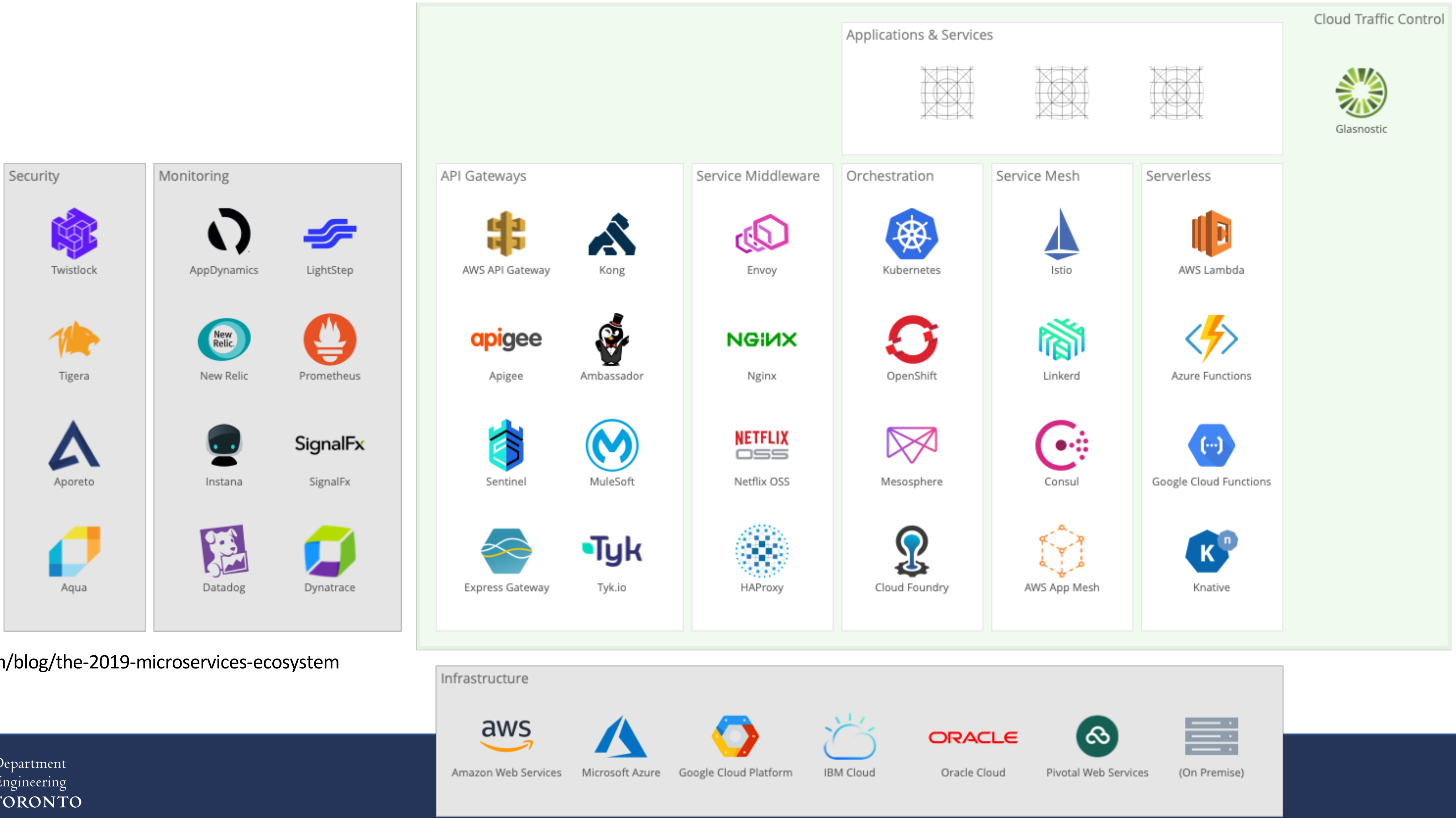
AWS Elastic Beanstalk

Easy to begin, Impossible to outgrow

AWS Lambda

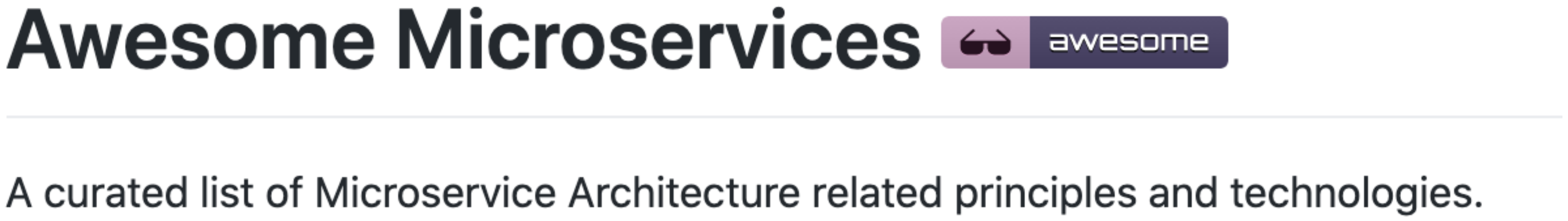
kubernetes

The 2019 Microservices Ecosystem



https://glasnostic.com/blog/the-2019-microservices-ecosystem

Technology Stacks


Awesome Microservices

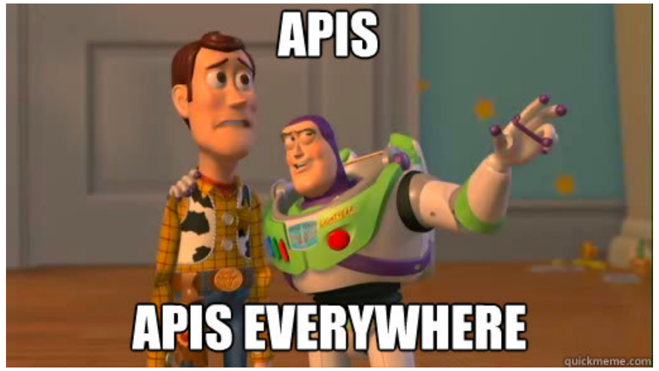
awesome

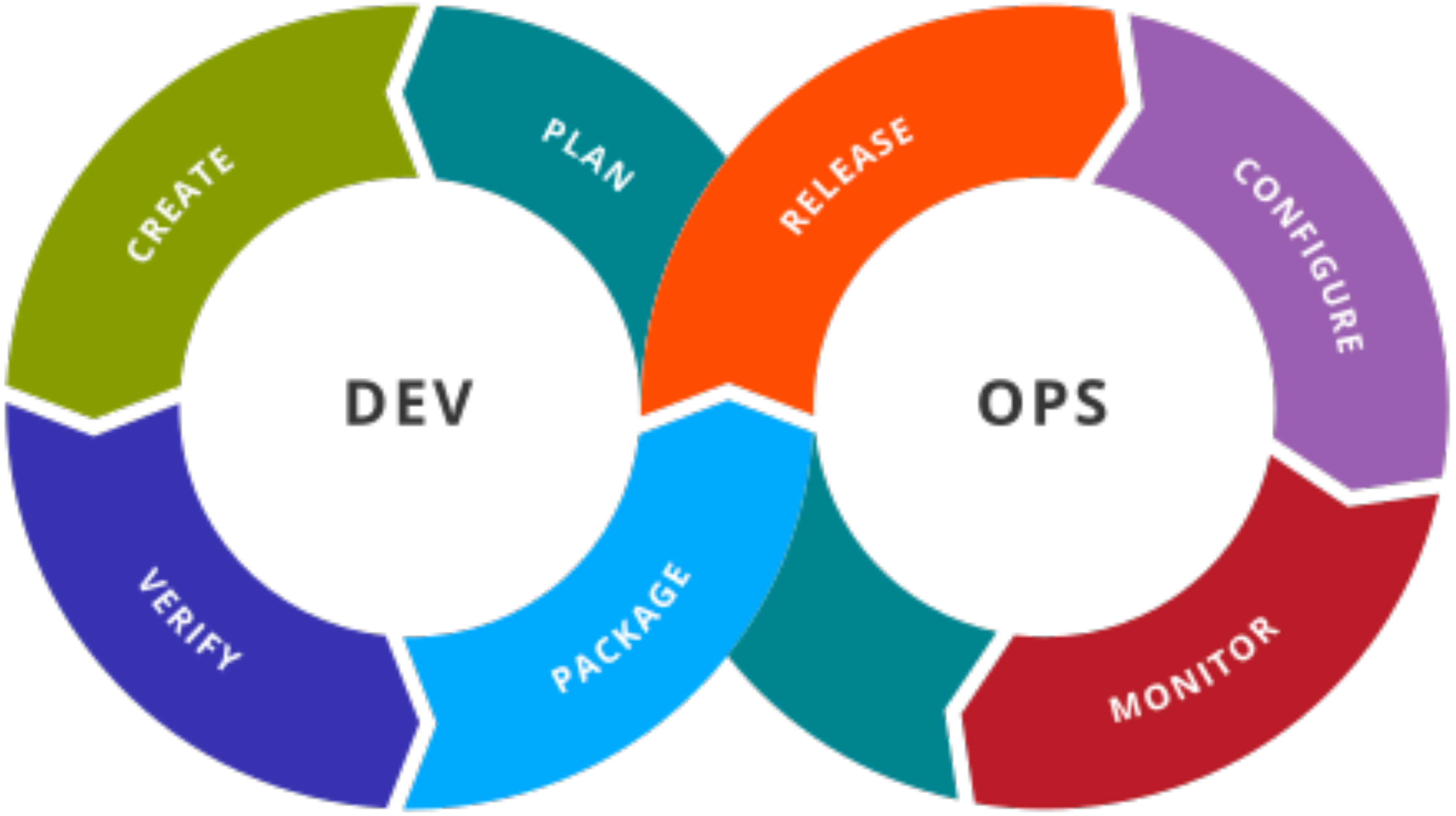
A curated list of Microservice Architecture related principles and technologies.

https://github.com/mfornos/awesome-microservices

Discussion of Microservices

- Are they really “new”?
- Do microservices solve problems, or push them down the line?
- What are the impacts of the added flexibility?
- Beware “cargo cult”
- “If you can’t build a well-structured monolith, what makes you think microservices is the answer?” – Simon Brown
- Leads to more API design decisions




DEV
OPS
PLAN
CREATE
VERIFY
PACKAGE
RELEASE
CONFIGURE
MONITOR