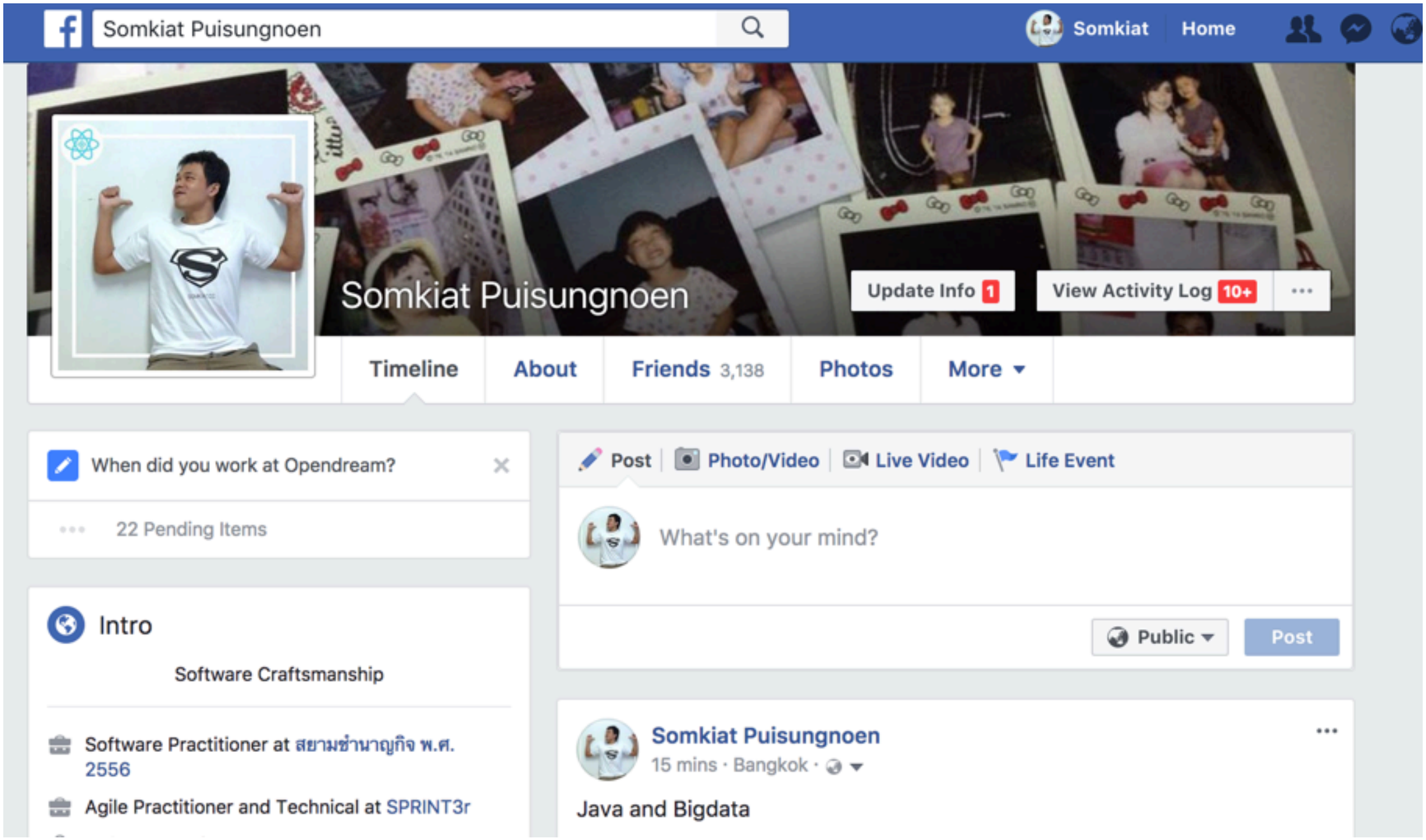
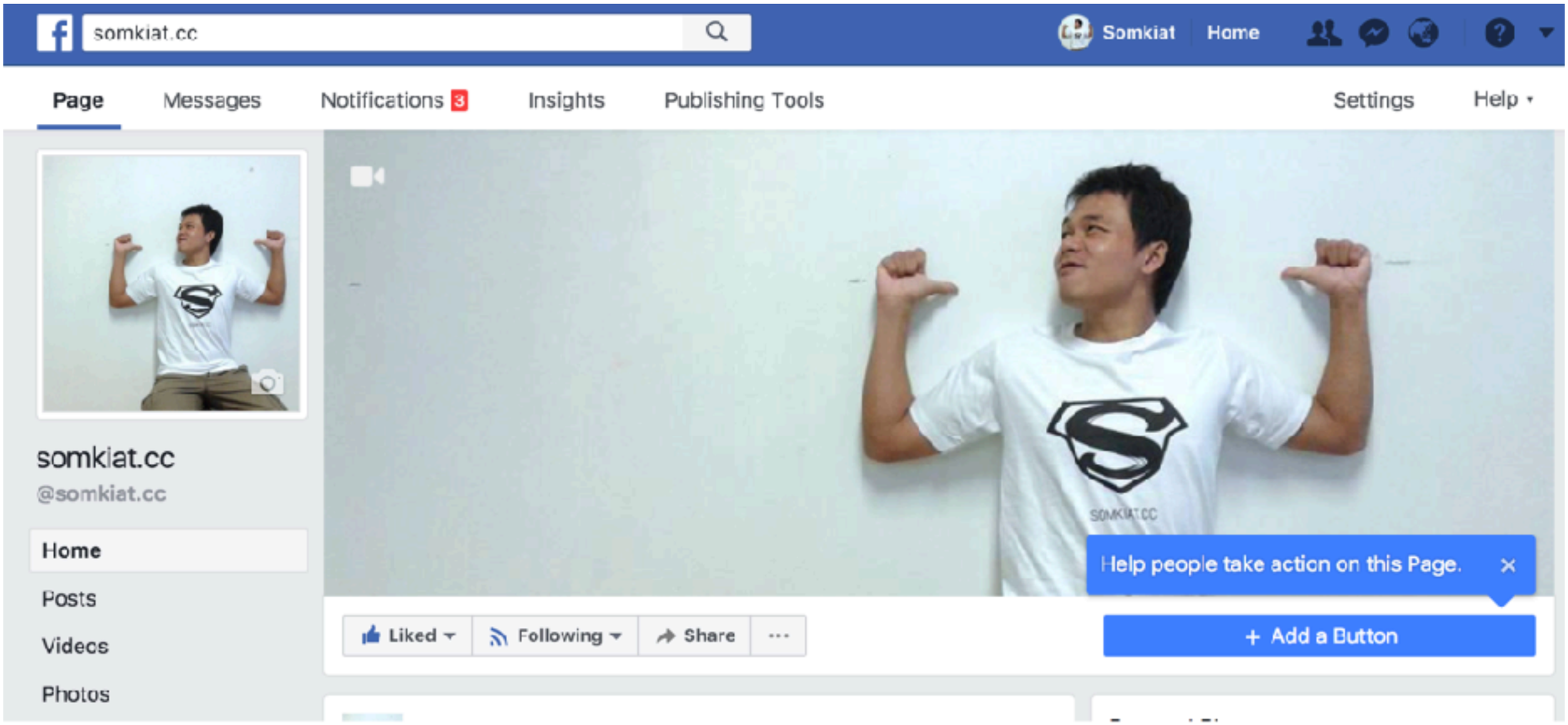
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Microservices
Develop, Testing, Deploy





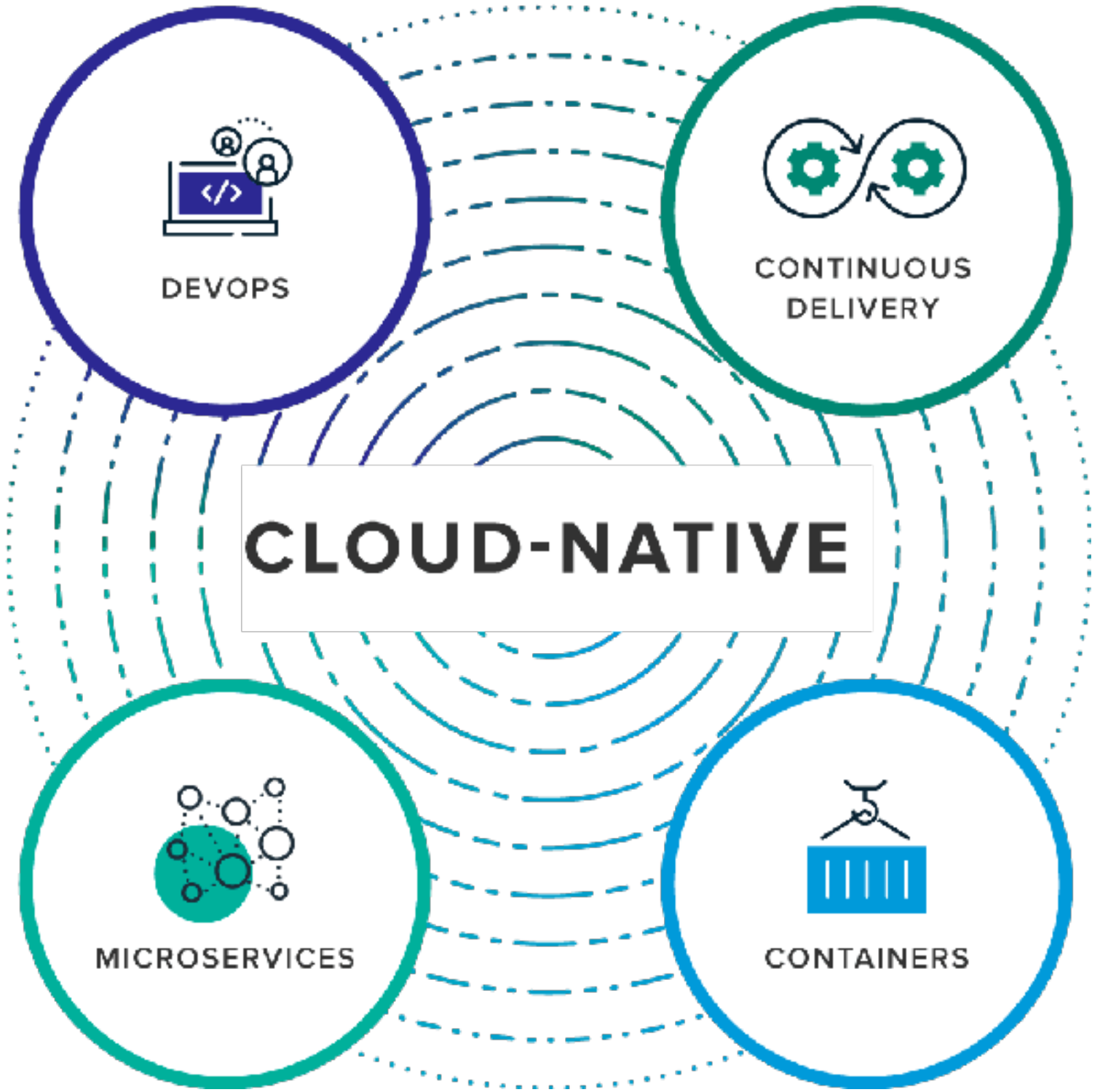
https://github.com/up1/course_microservices-3-days

Module 2 : Develop

Recap Microservice
Properties of Microservice
Microservice 1.0 - 4.0
How to develop Microservice ?
How to test Microservice ?
12-factors app
Workshop

Module 3 : Deploy

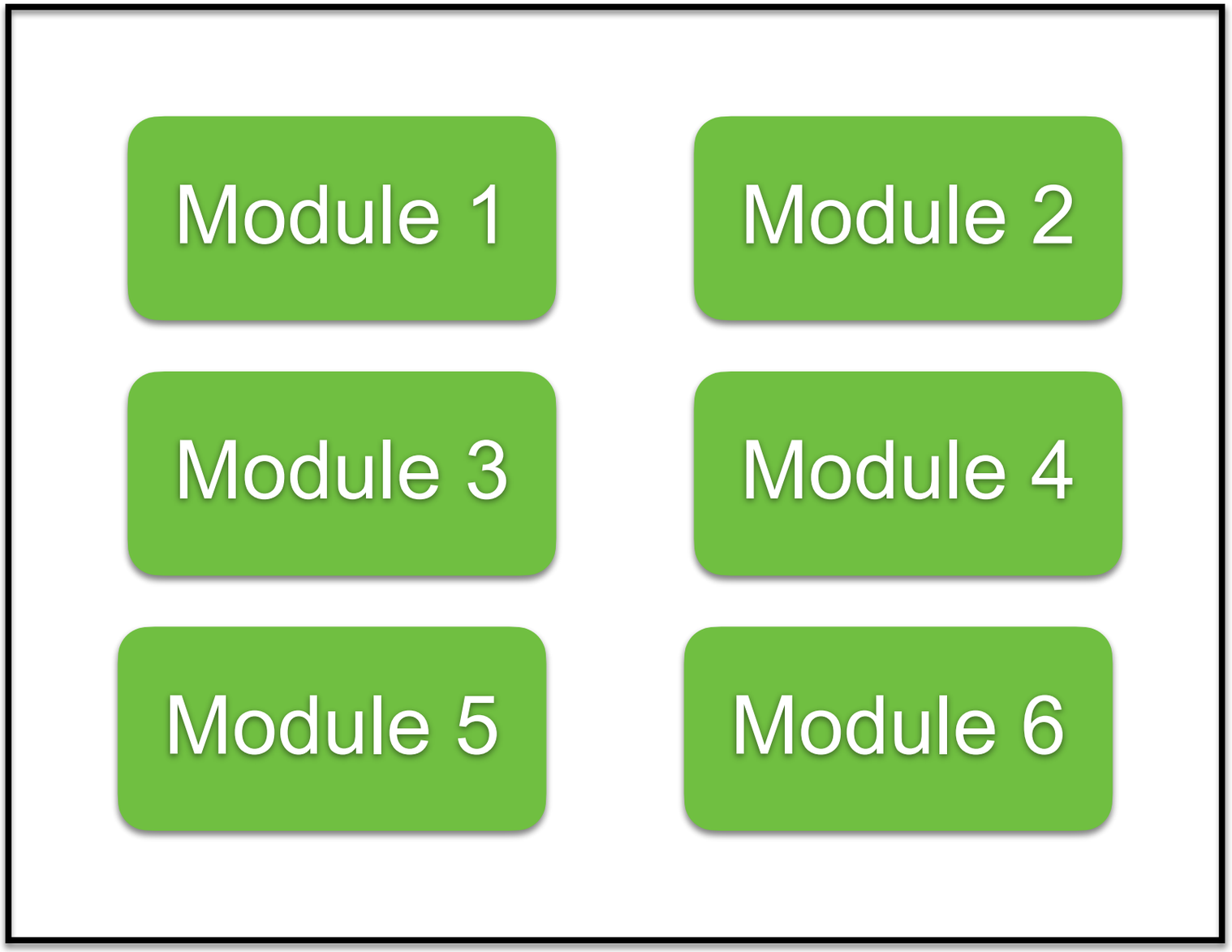
How to deploy Microservice ?
Continuous Integration and Delivery
Practices of Continuous Integration
Deployment strategies
Working with containerization (Docker)
Workshop



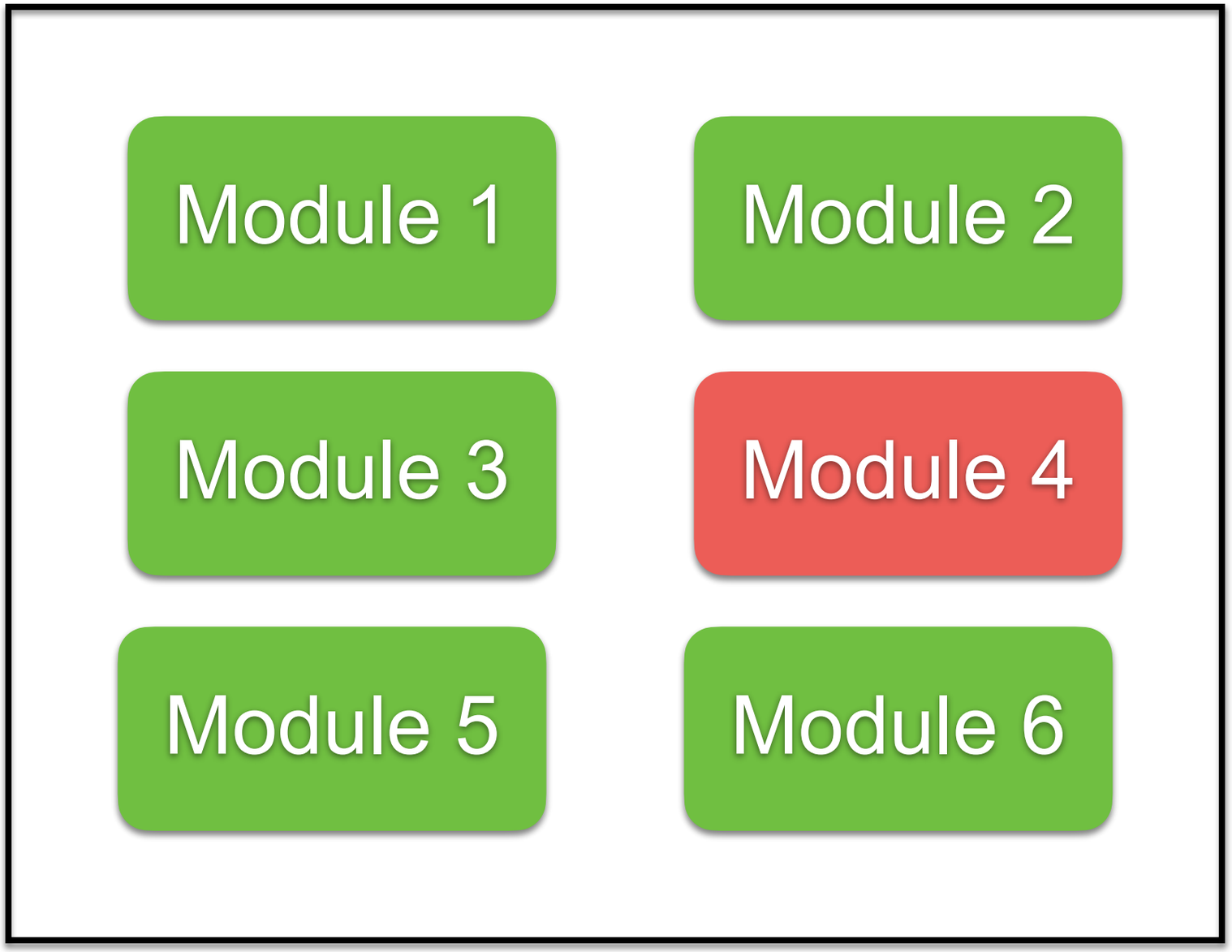
https://pivotal.io/cloud-native

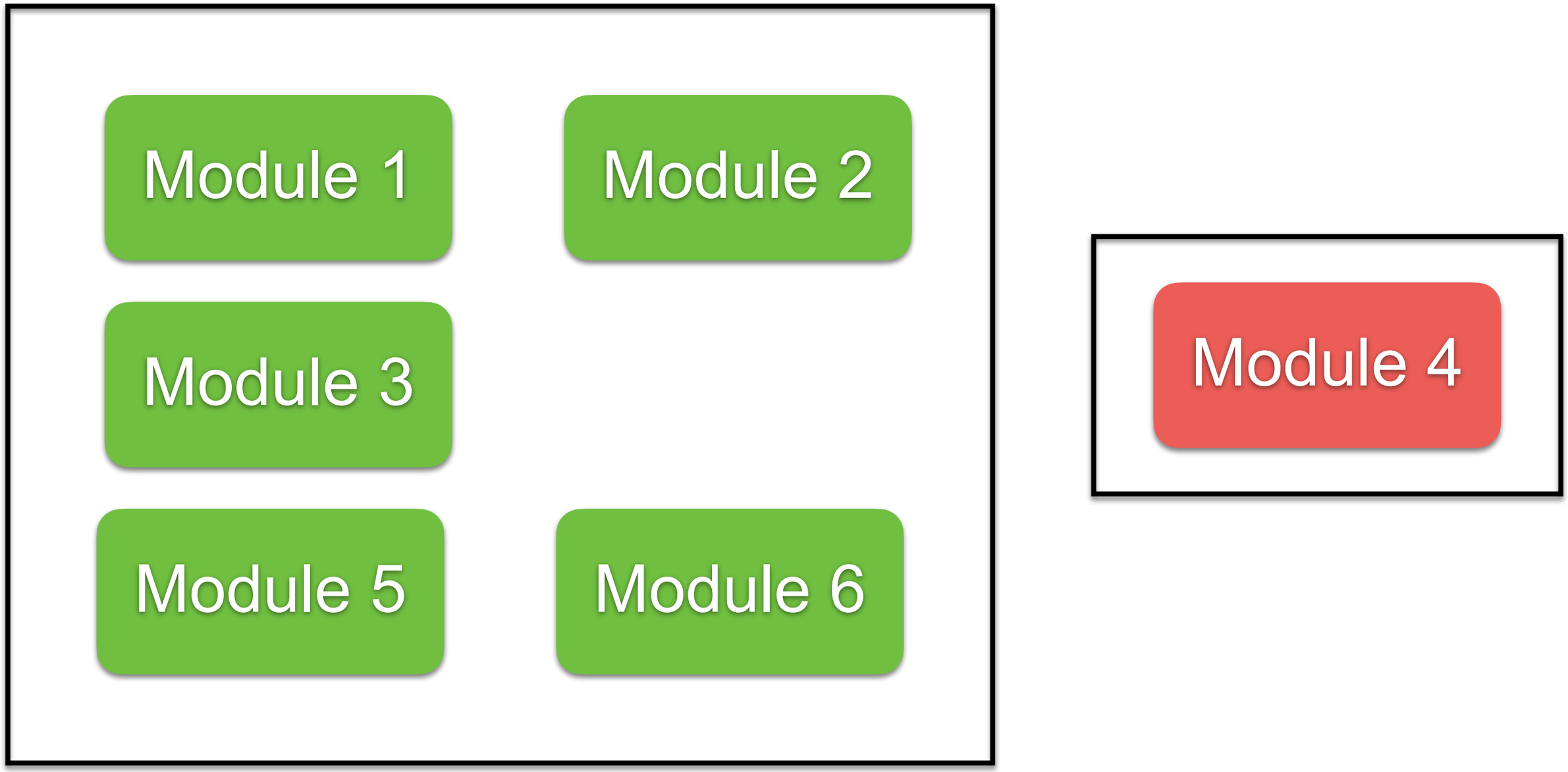
Microservices

Let’s start with good monolith

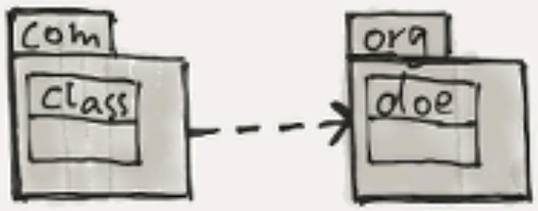


Find your problem

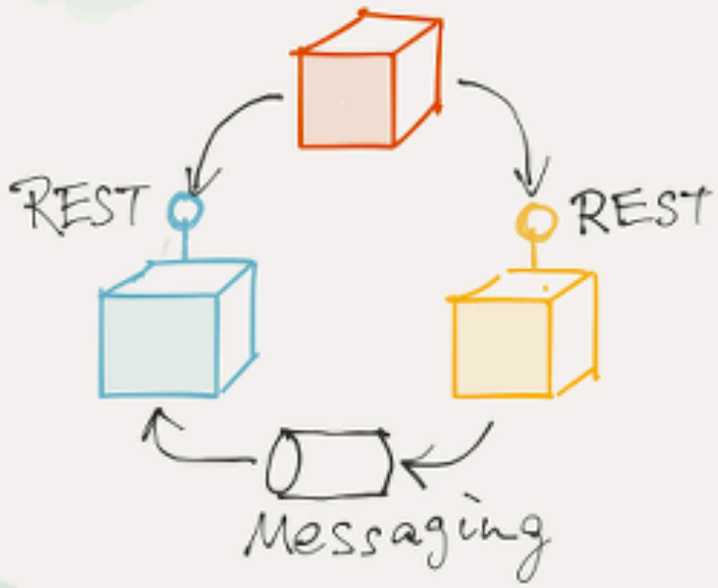




Architecture



Microservices

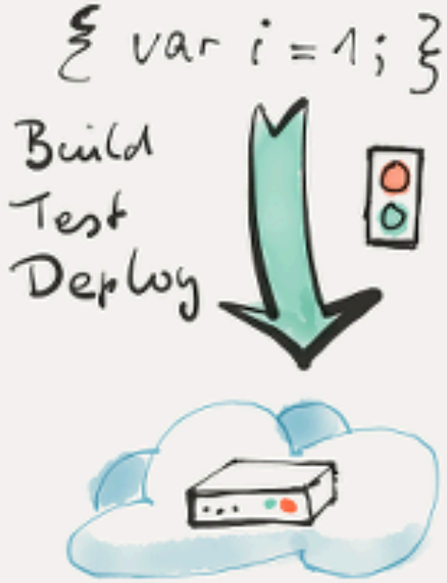


People & Teams

Communication
Collaboration

Deployment

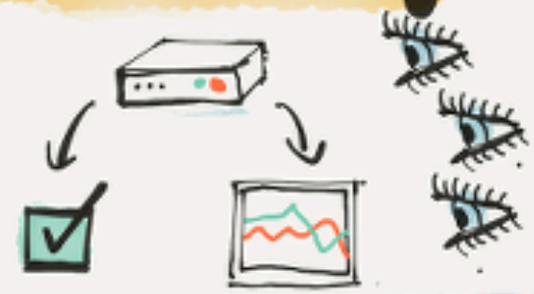
Continous Delivery



Infrastructure

Provisioning

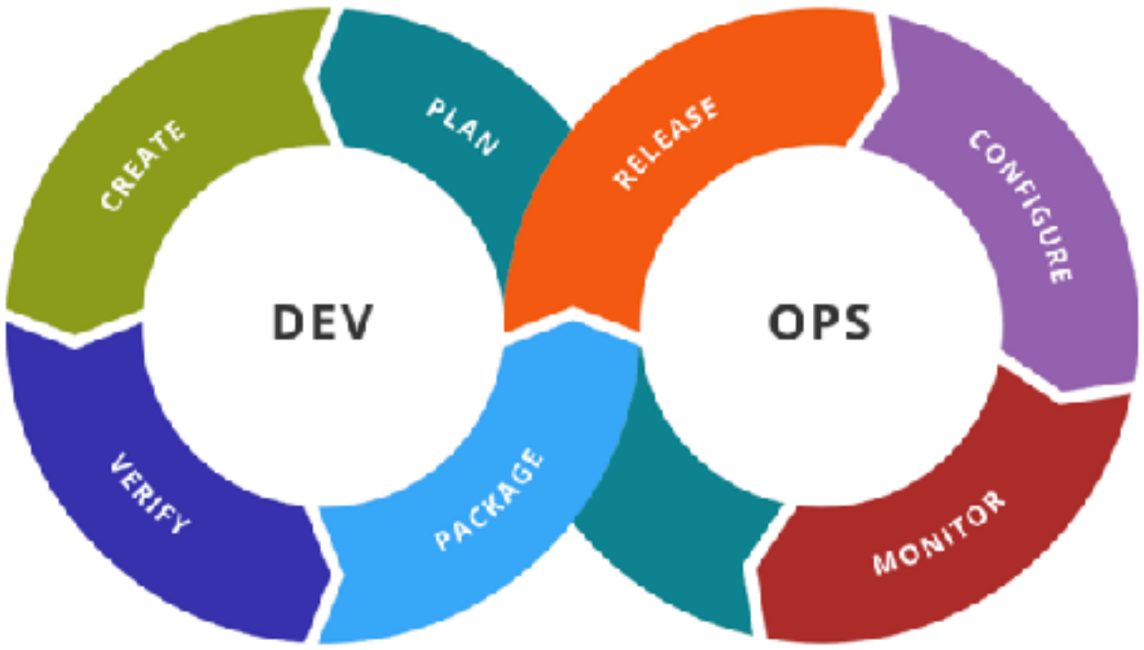
Monitoring



Features & Technology

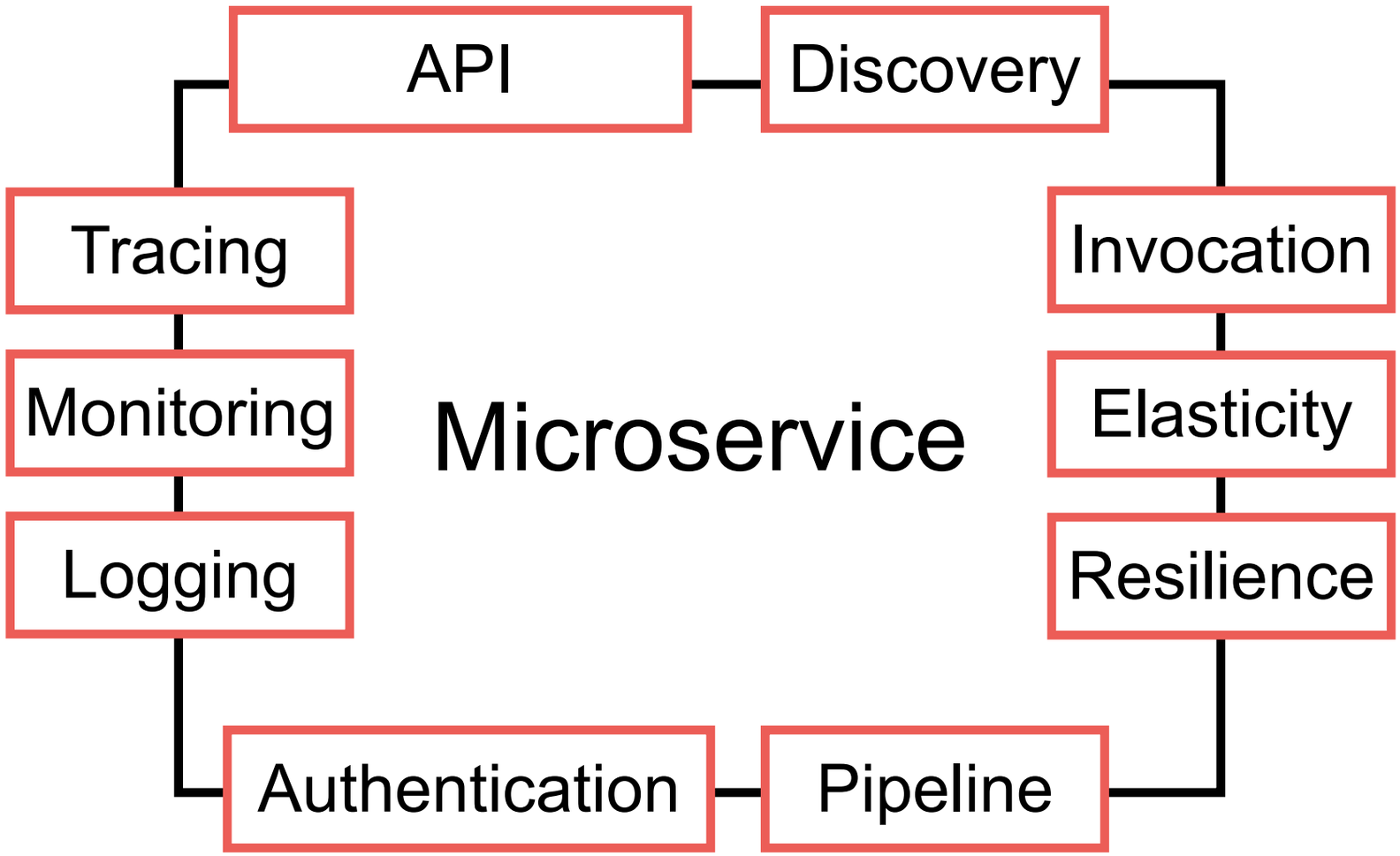
Microservice prerequisites

Rapid provisioning
Basic monitoring
Rapid Application Deployment
DevOps culture

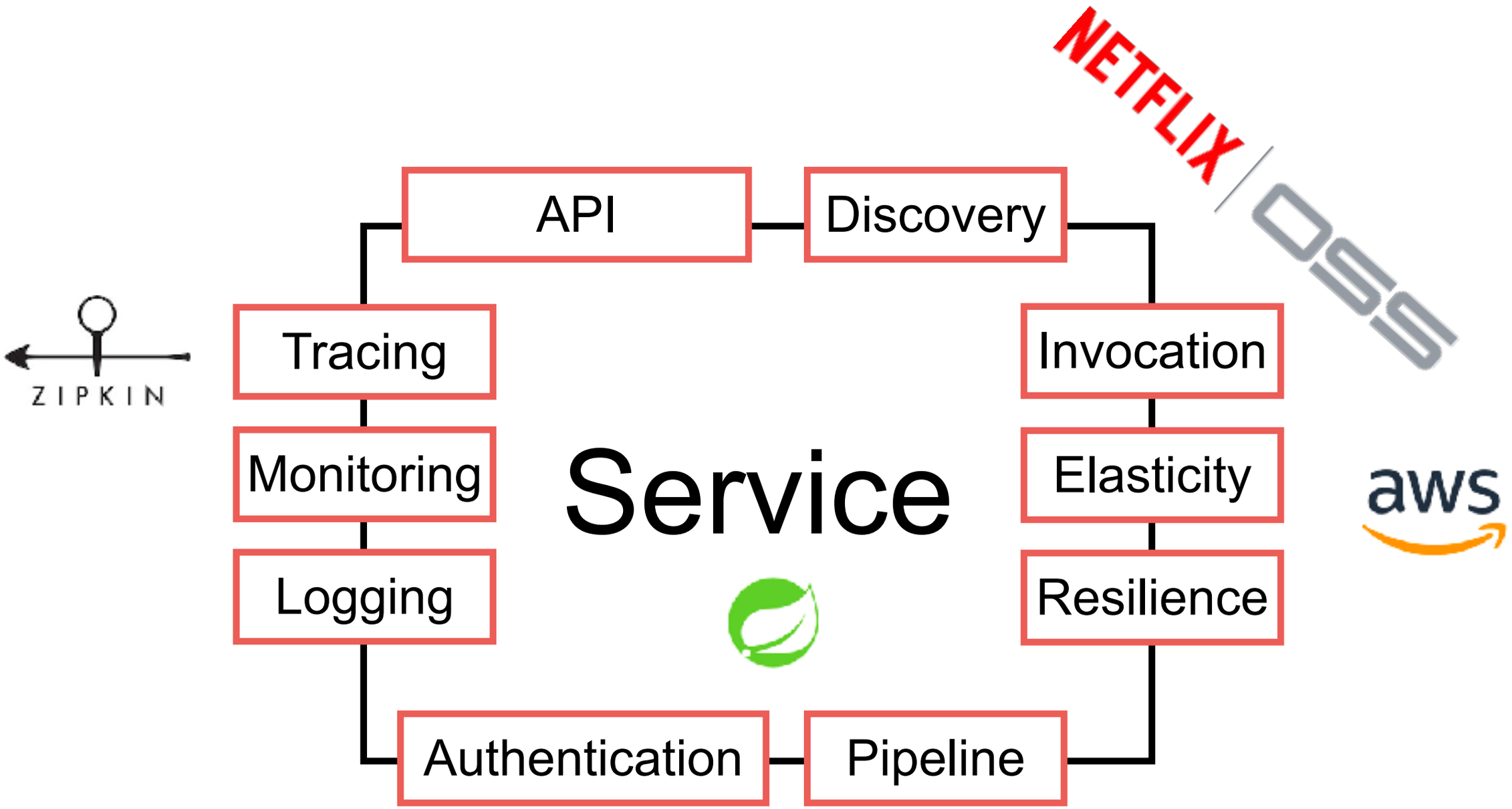


Module 2 : Develop

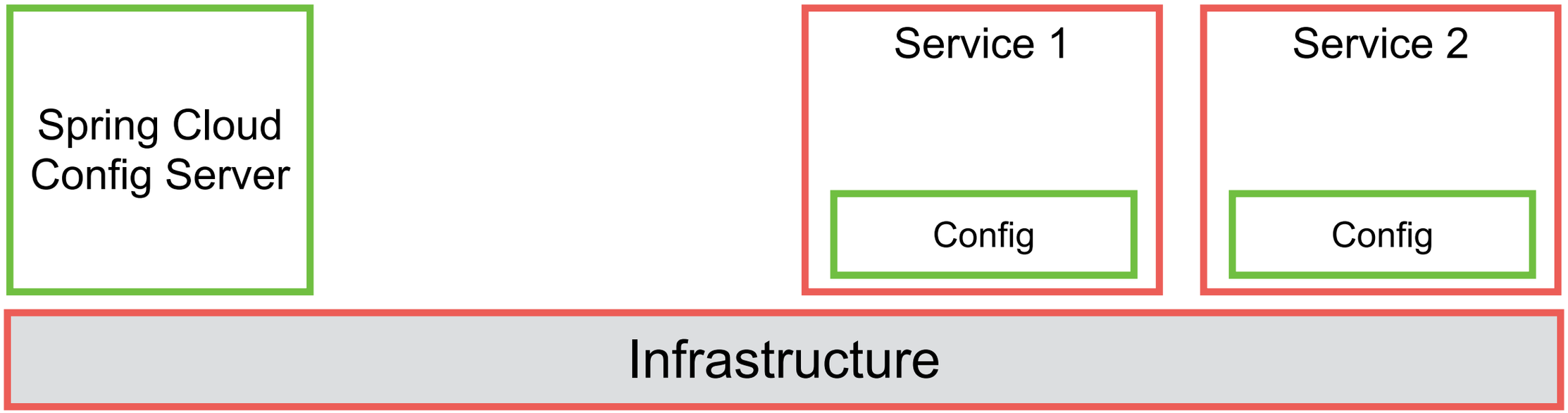
Properties of Microservice



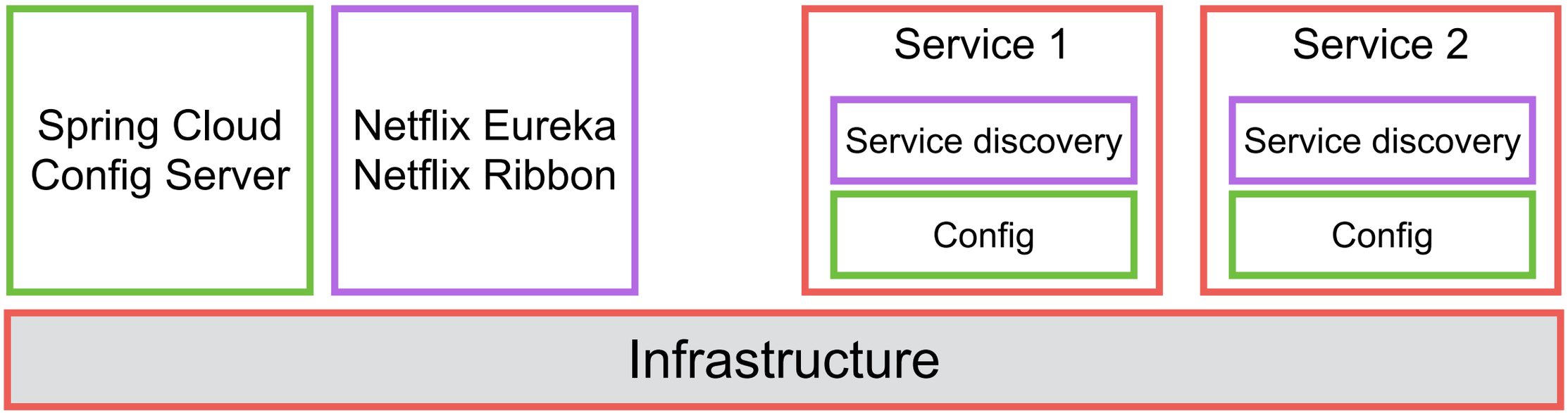
Microservice 1.0



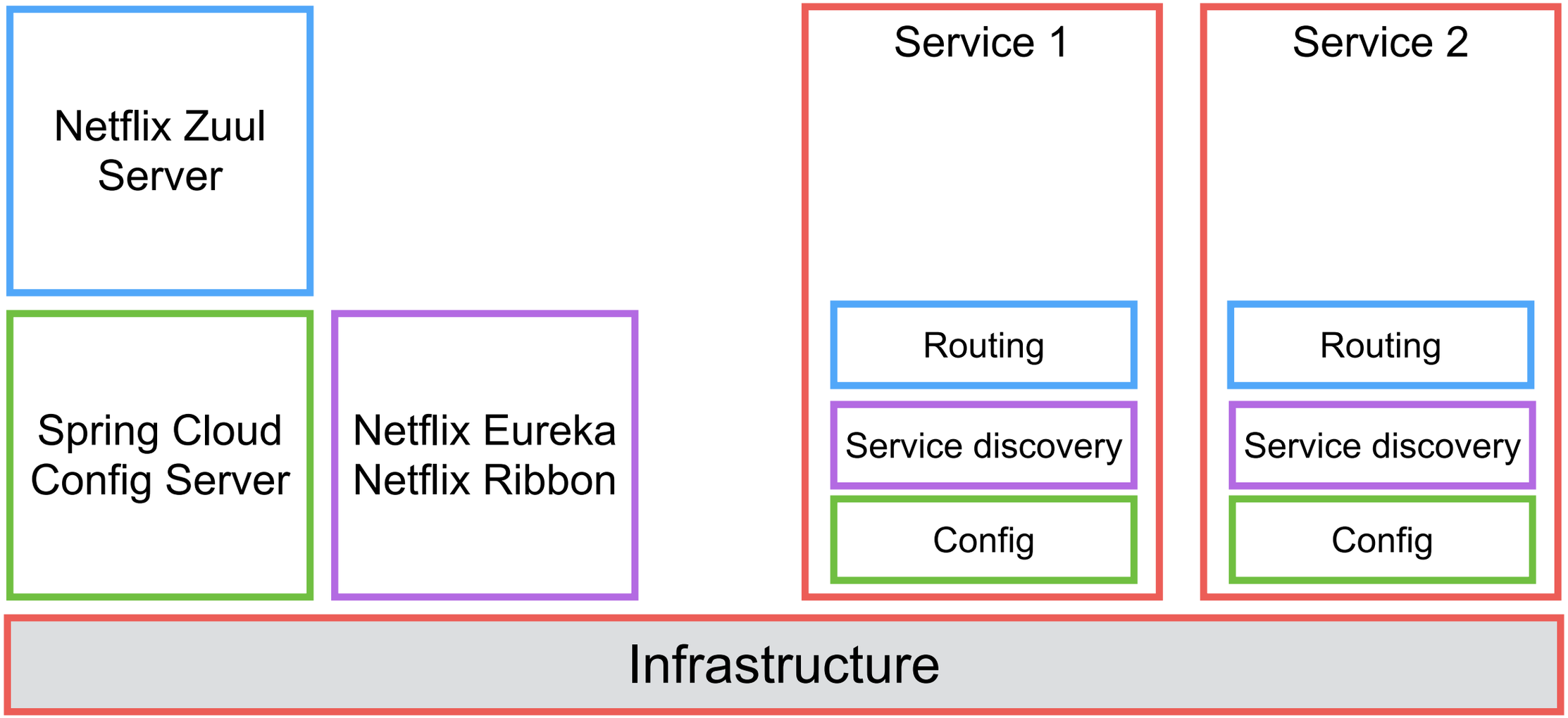
Configuration



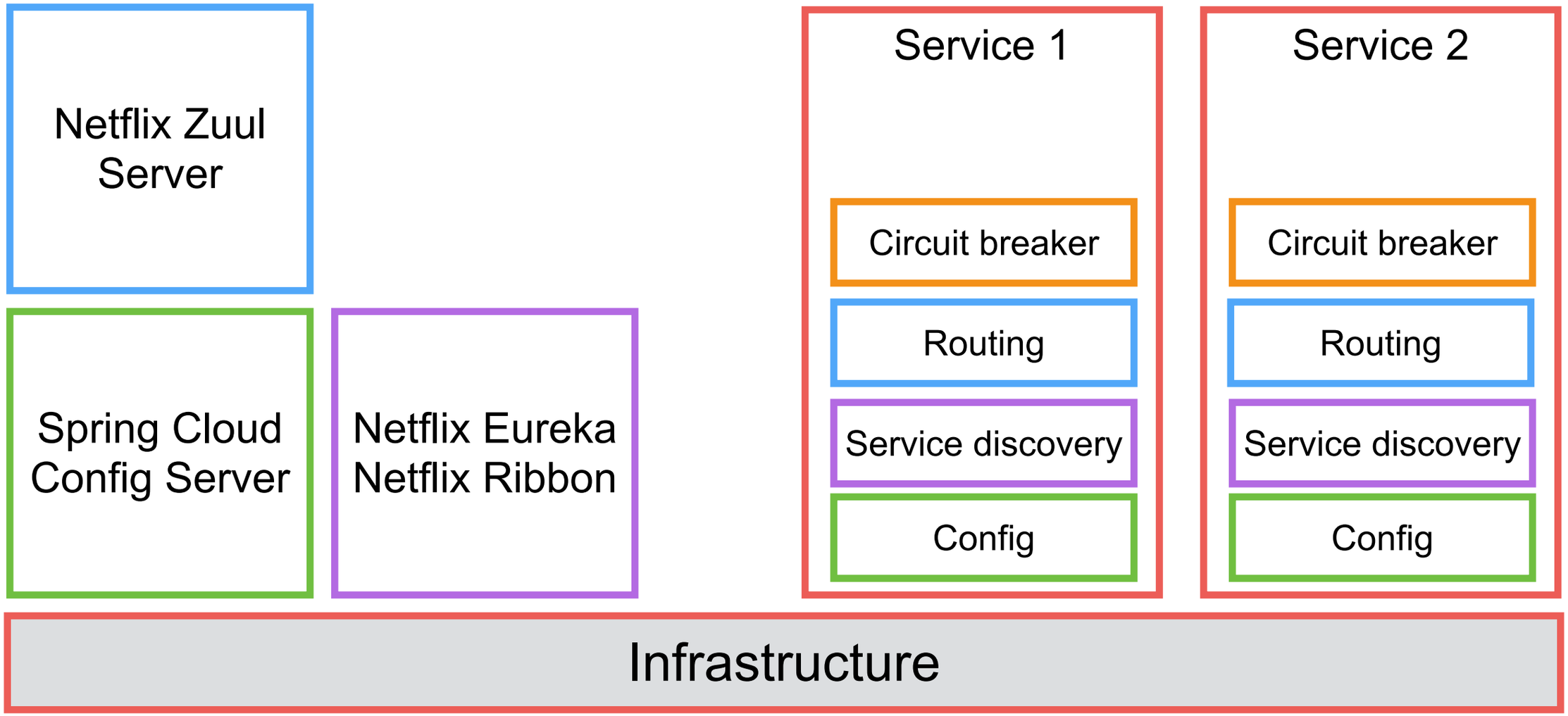
Service Discovery



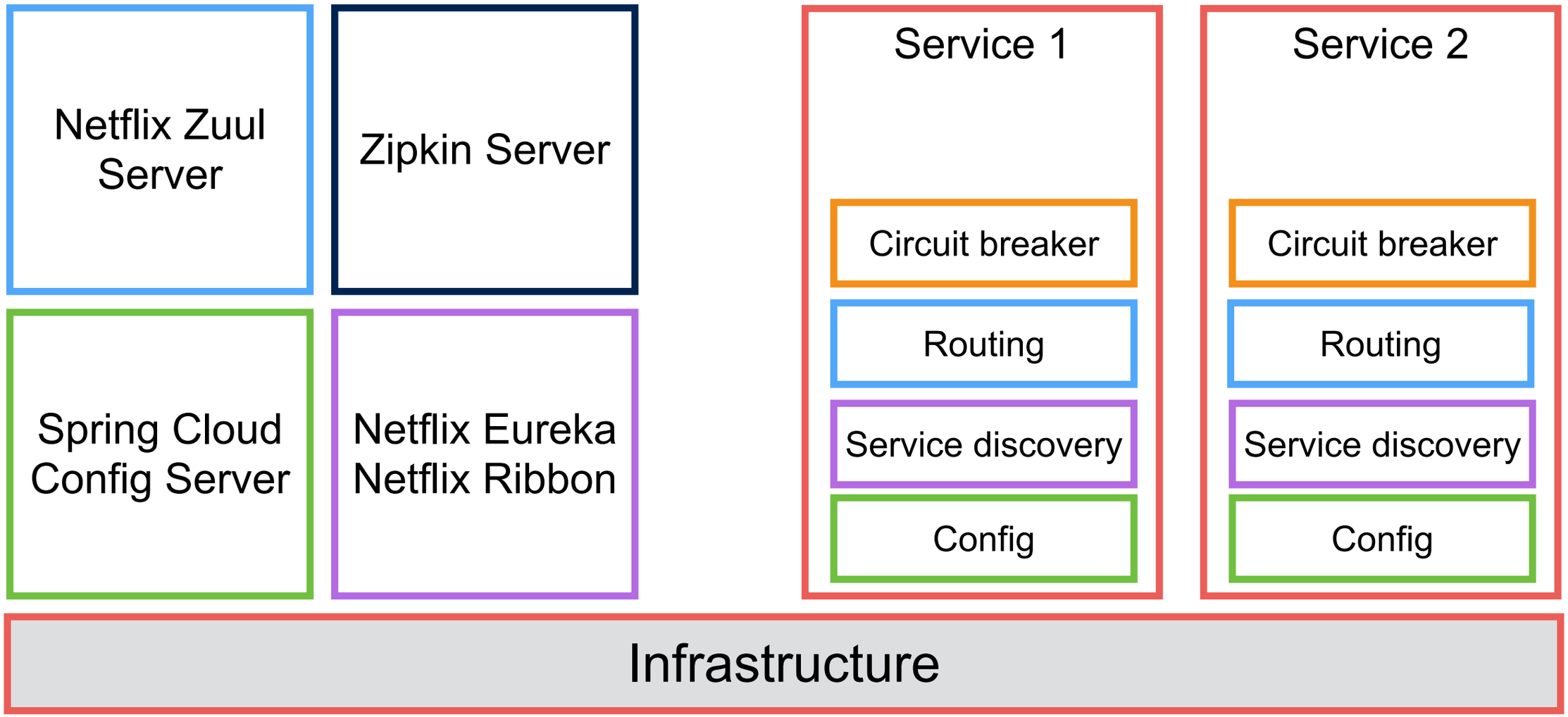
Dynamic Routing



Fault Tolerance



Tracing and Visibility

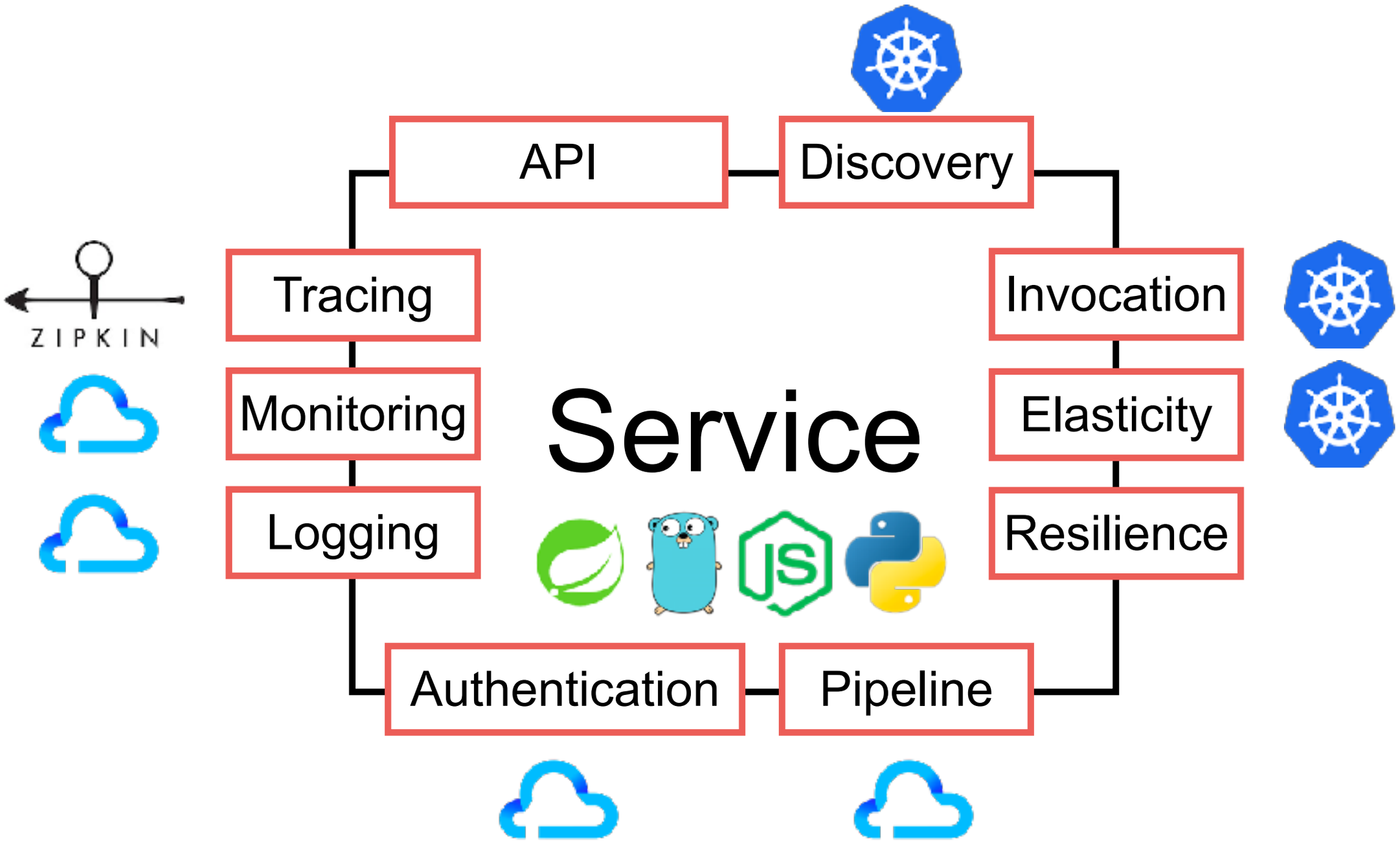


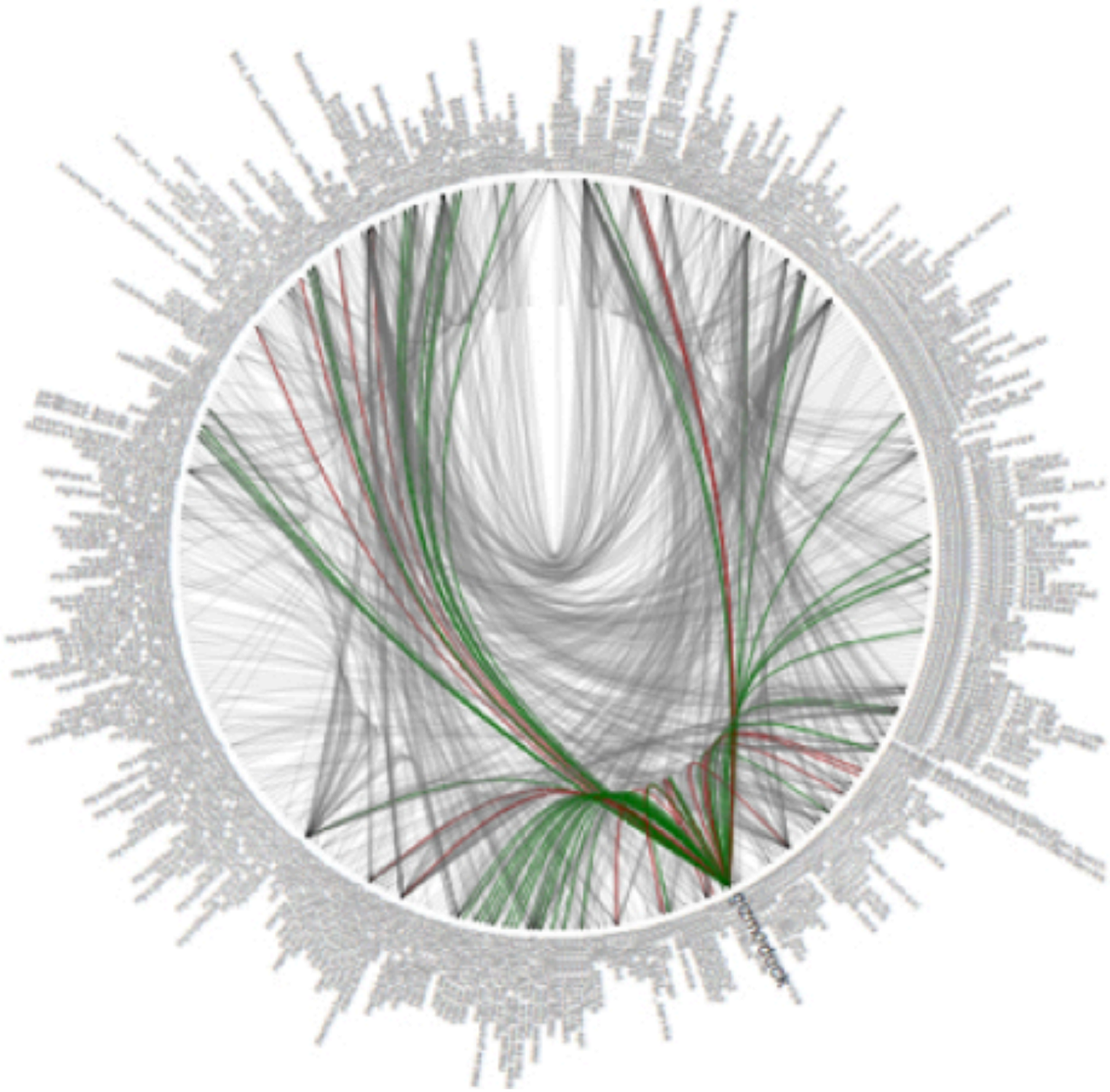
Microservice 1.0

JVM only
Add libraries to your code/service

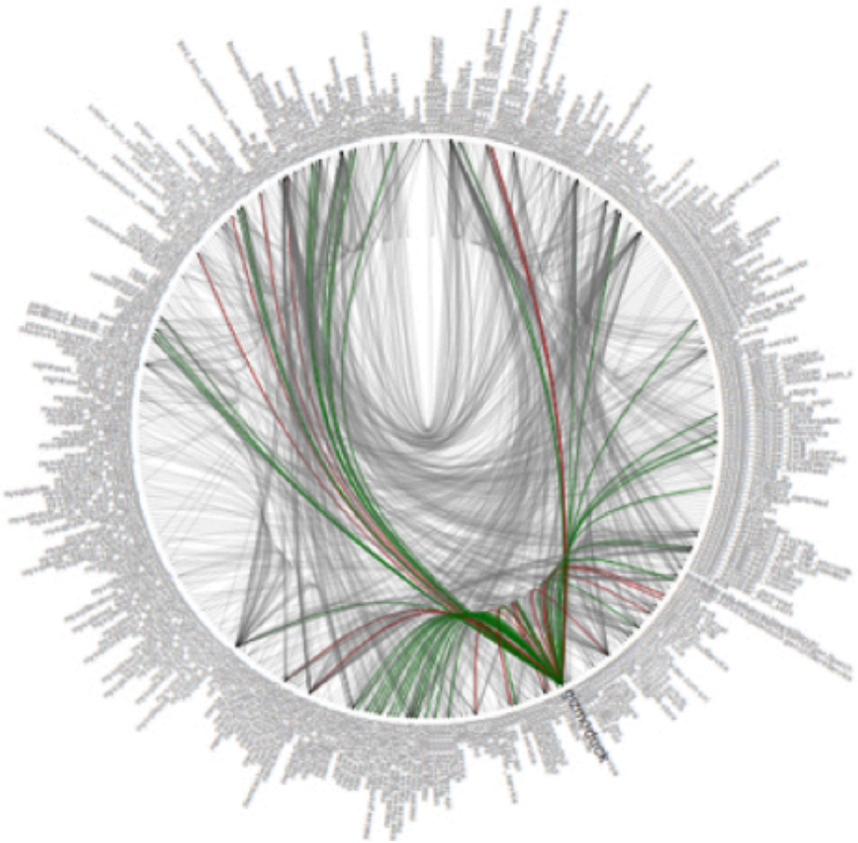
kubernetes

Microservice 2.0



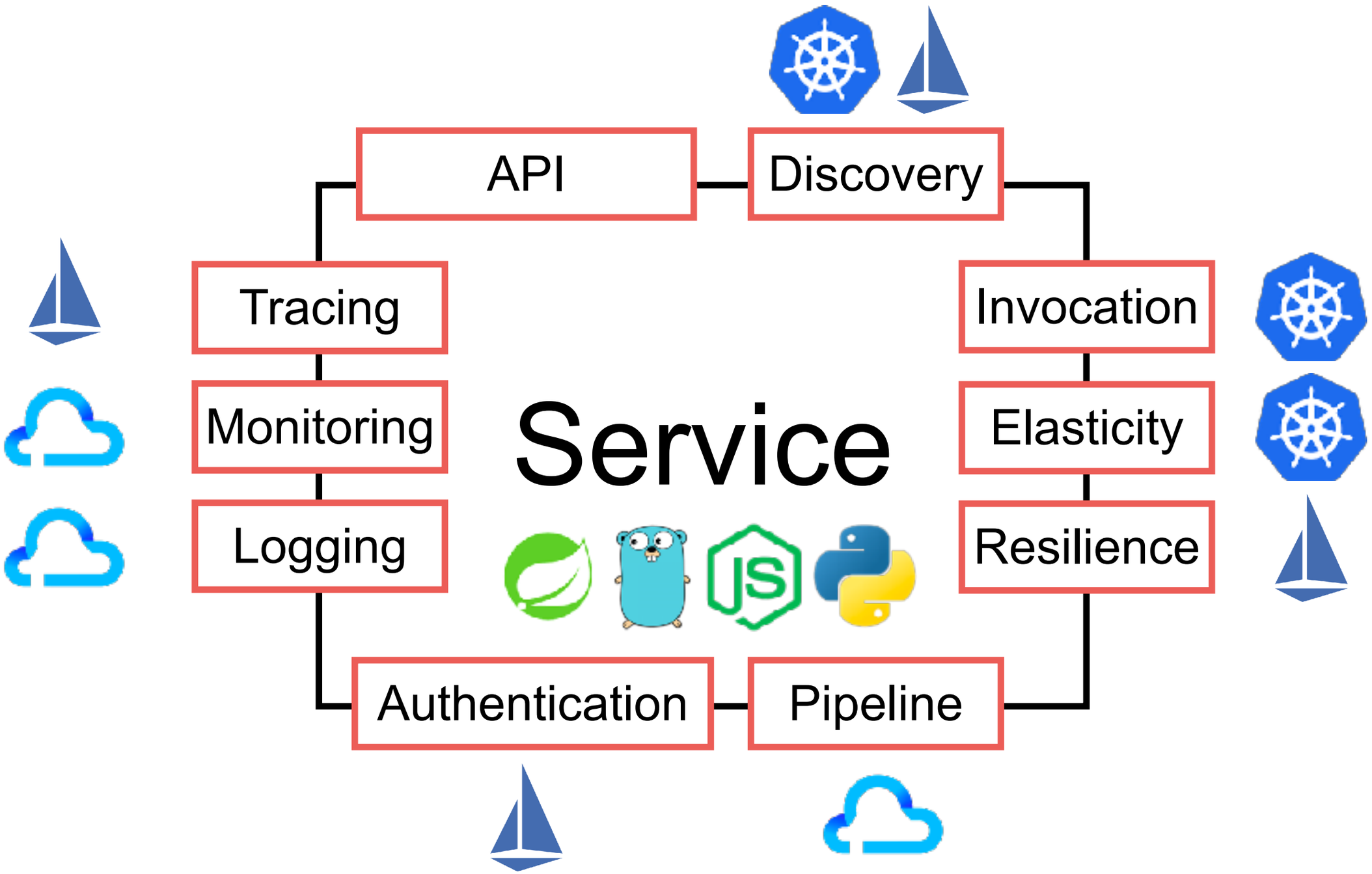


Service Mesh

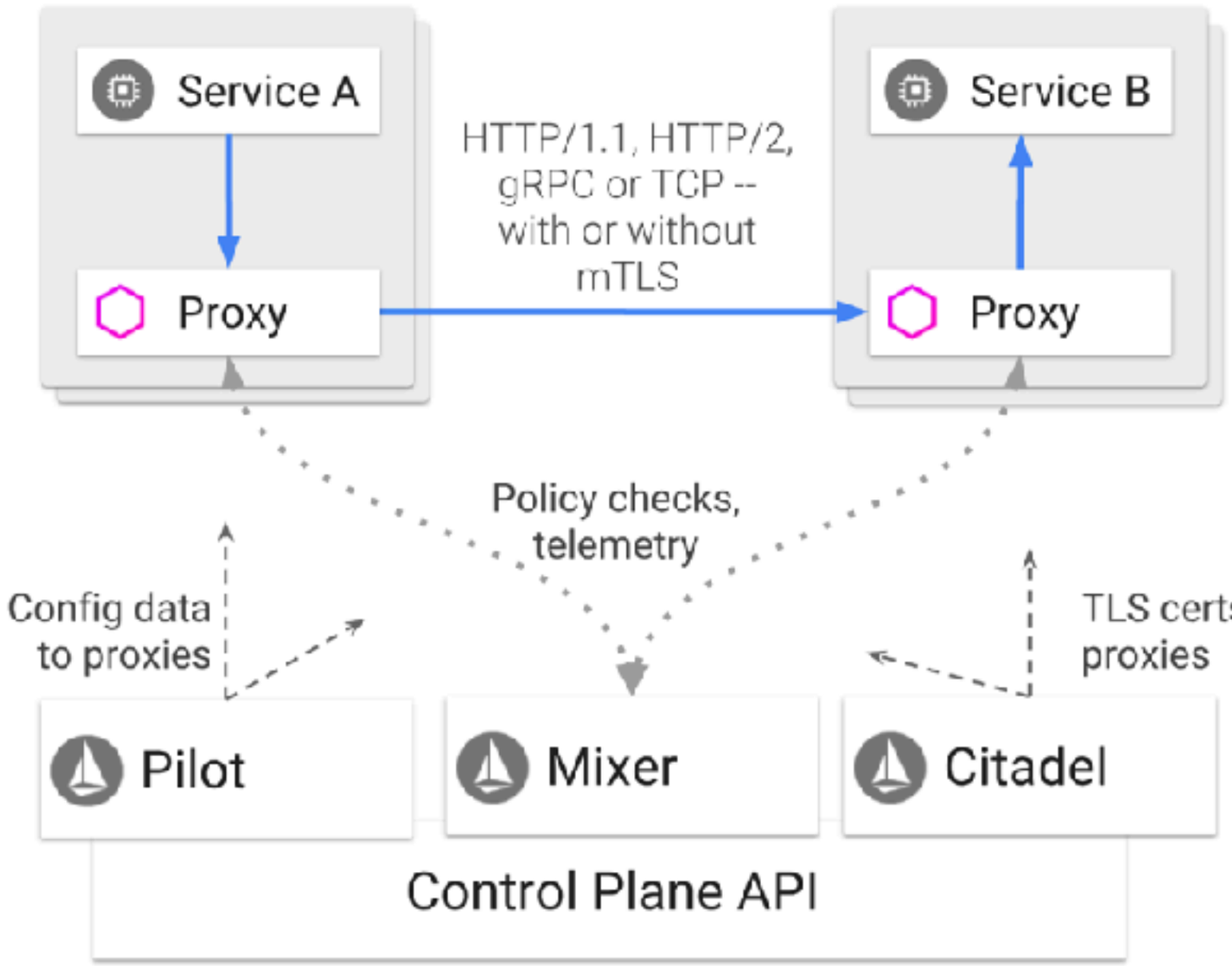


Service Mesh with Istio

Microservice 3.0

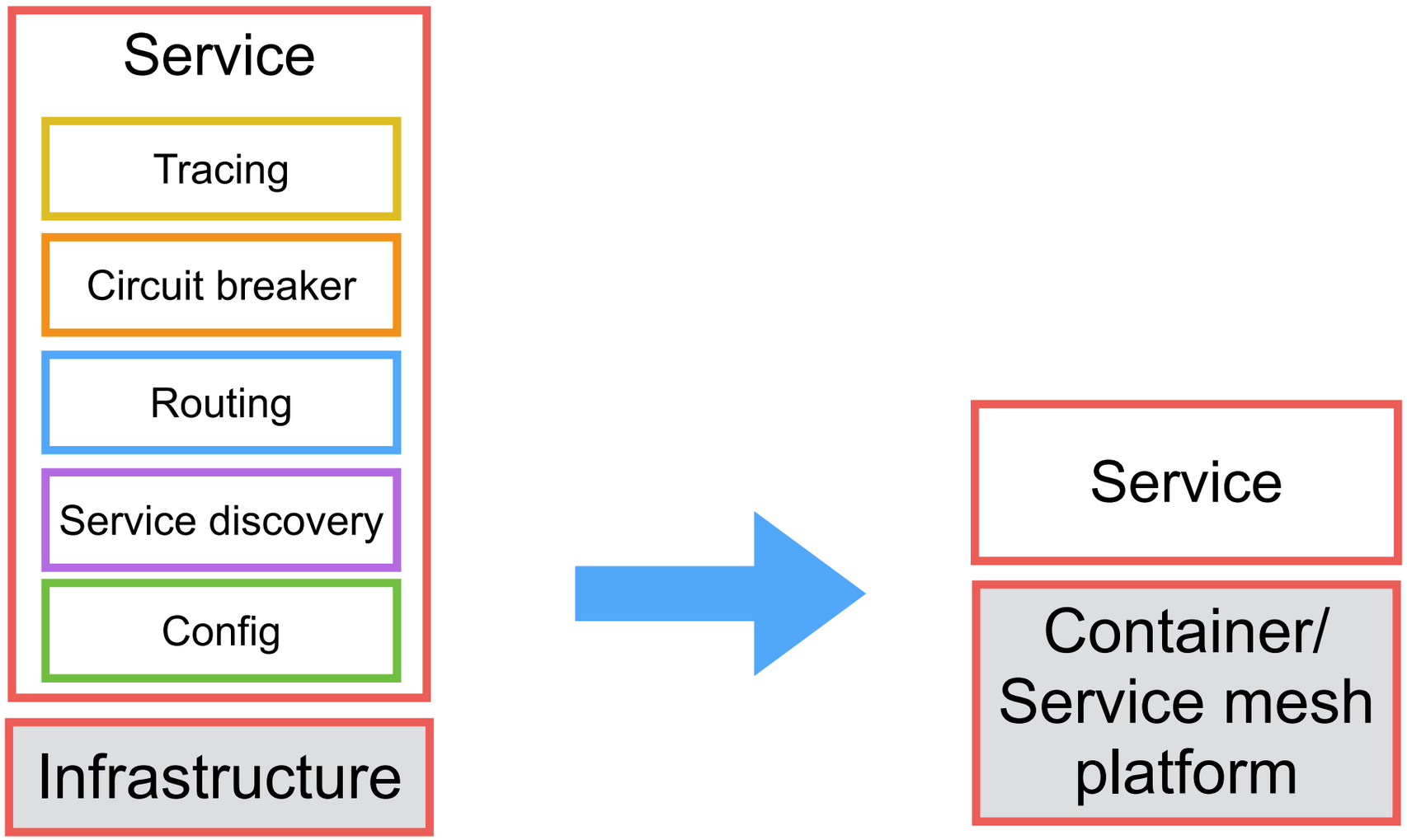


Istio



https://istio.io/docs/concepts/what-is-istio/

Microservice Evolution



Function-as-a-Service (FaaS)

Microservice 4.0 == FaaS

OPENFAAS

APACHE
OpenWhisk™

fission

Kubeless

Workshop
Microservice 1.5

Twelve-Factor App

Twelve-Factor App

https://12factor.net/

Initial to build app for Heroku

Principles to cloud and container native app

1. Codebase

Codebase must be tracked in version control and will have many deploy

2. Dependencies

Dependencies are explicitly declared and isolated

Use dependency management tools to shared libraries

3. Configuration

Store configuration in the environment

Add configuration in environment variables or config files

4. Backing services

Treat backing services as an attached resource

Should easy to deploy and change

5. Build, Release, Run

Always have a build and deploy strategy

Build strategies for repeated builds, versioning of running system and rollback

6. Processes

Execute the application as a stateless process

7. Port Binding

Expose services via port bindings

8. Concurrency

Scale out with the process model

9. Disposability

Quick application startup and shutdown times
Graceful shutdown

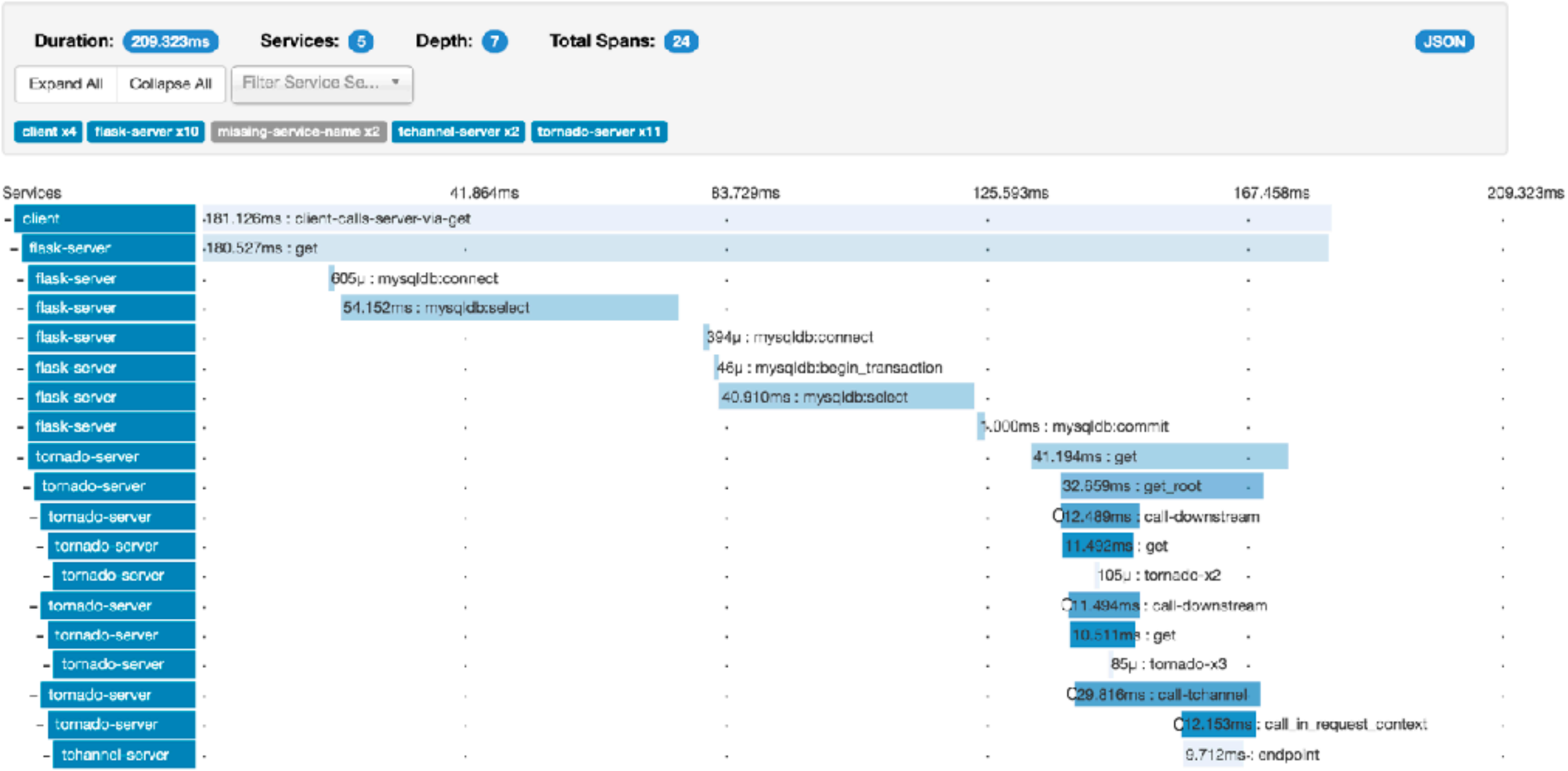
10. Dev/prod parity

Application is treated the same way in dev, staging and production

11. Log management

Treated as an event stream

Logging/Tracing



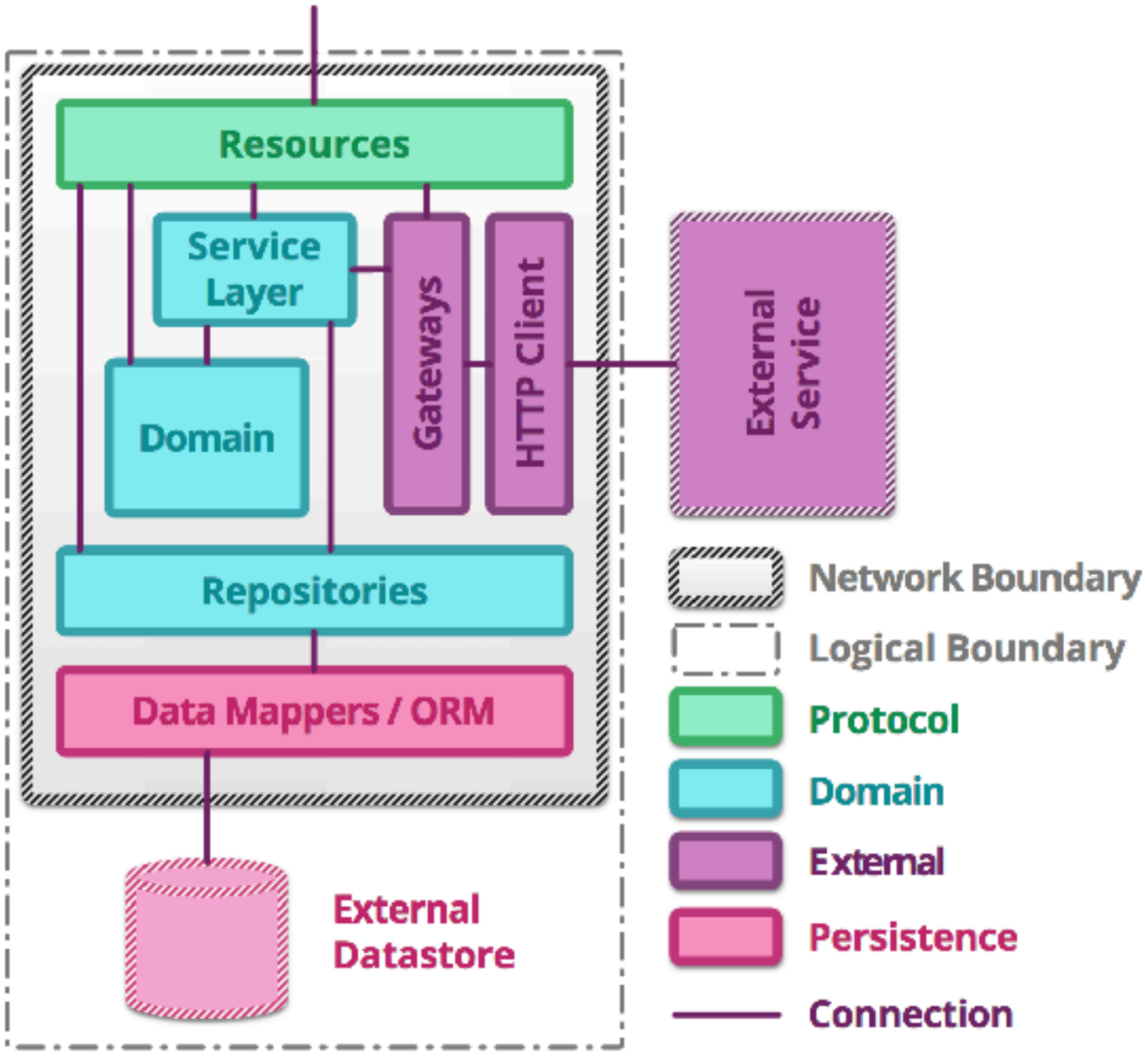
https://zipkin.io/

12. Admin tasks

Treated the same way like the rest of the application

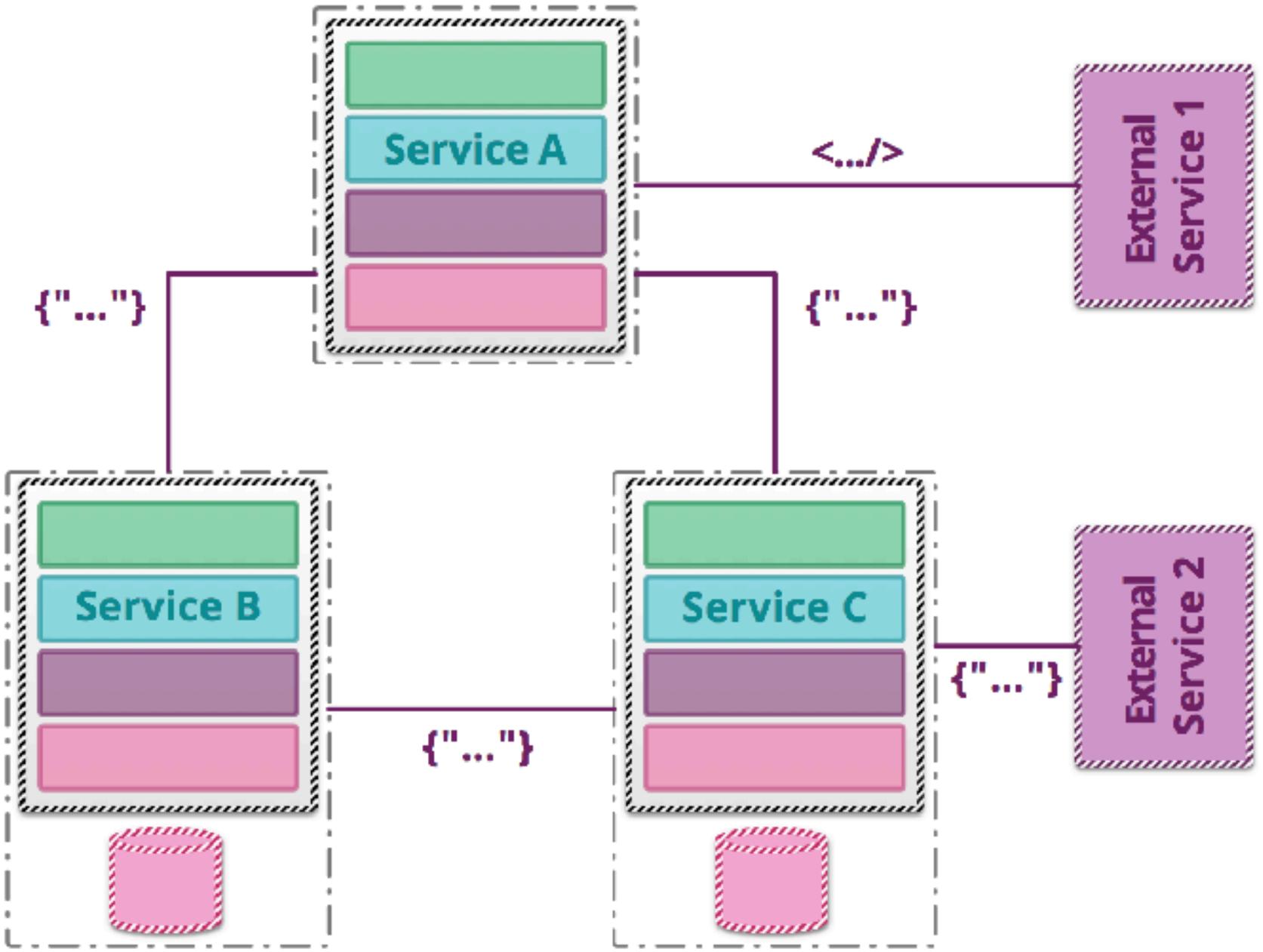
Microservice Testing

Service structure



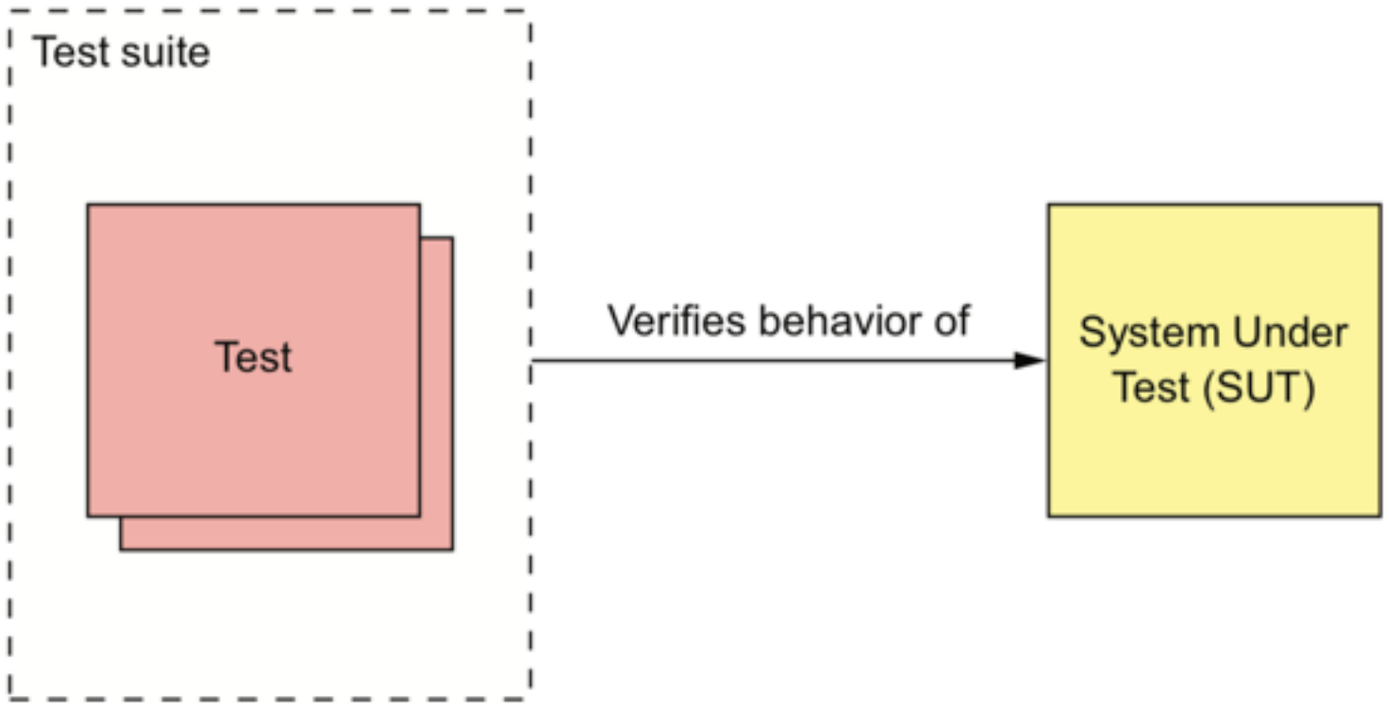
https://martinfowler.com/articles/microservice-testing

Multiple services

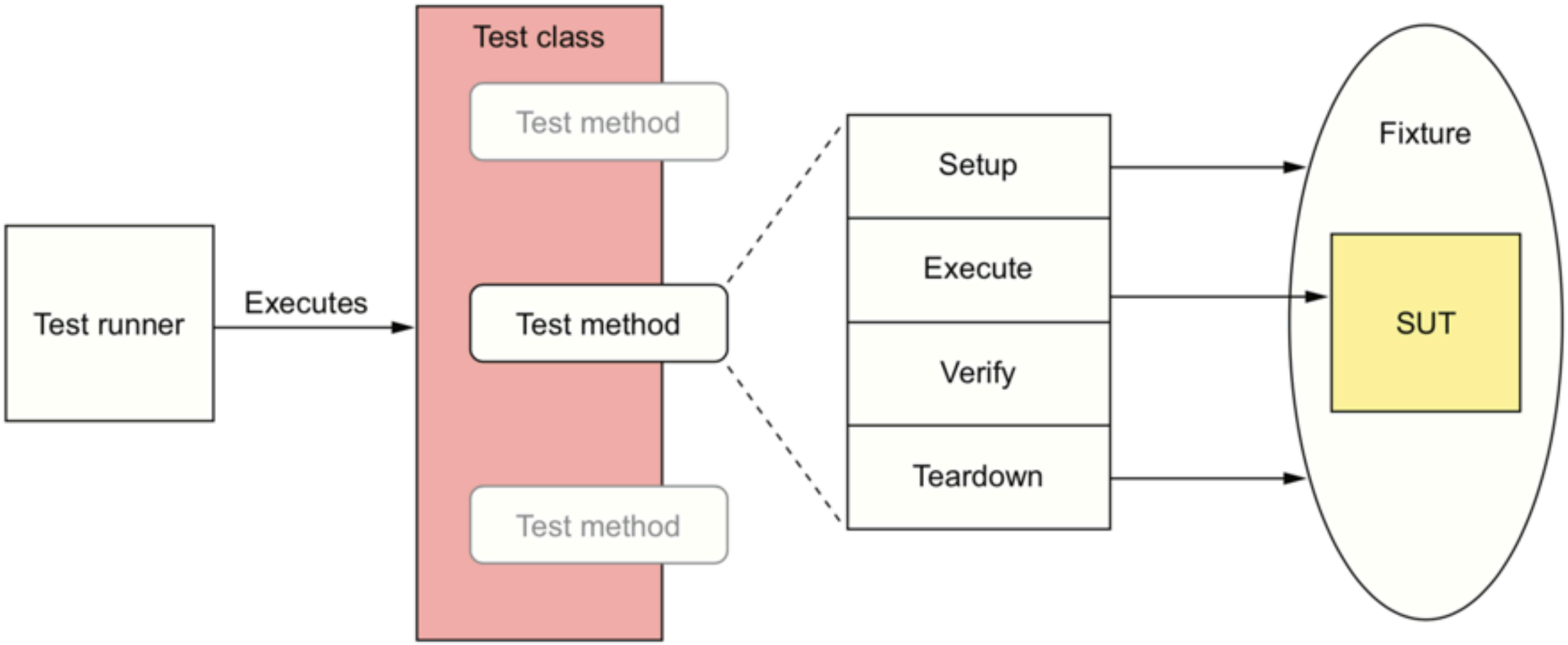


https://martinfowler.com/articles/microservice-testing

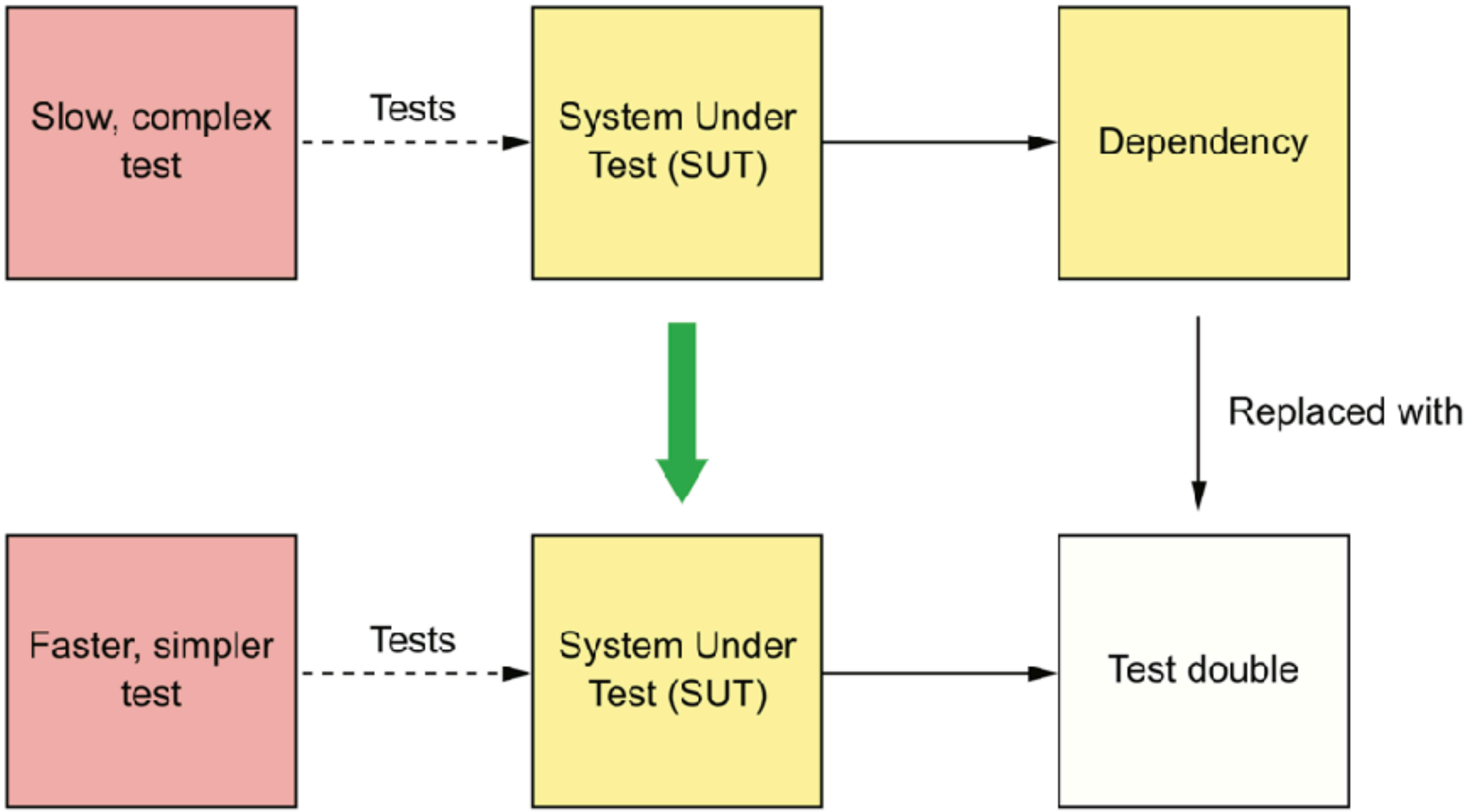
Testing

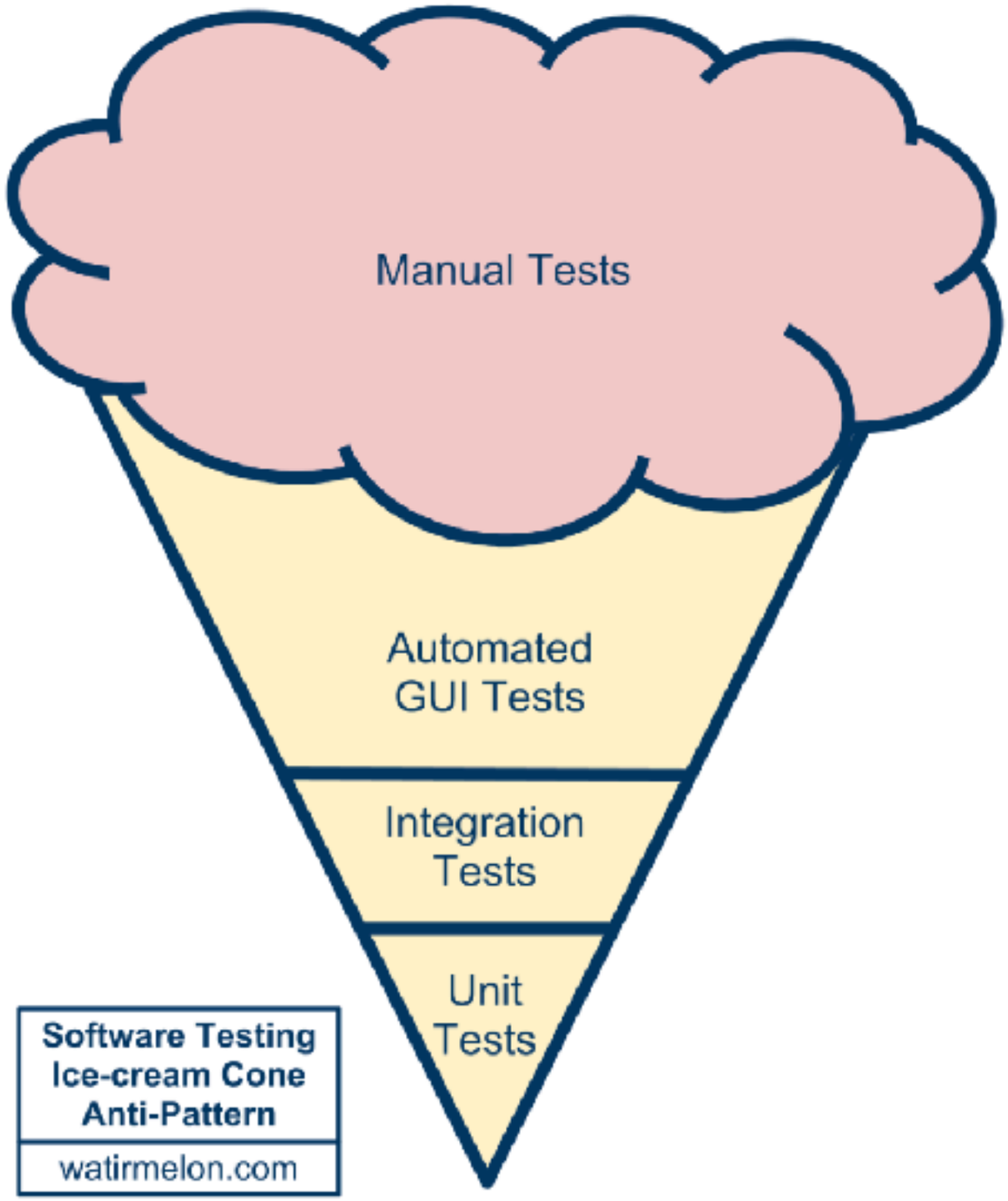


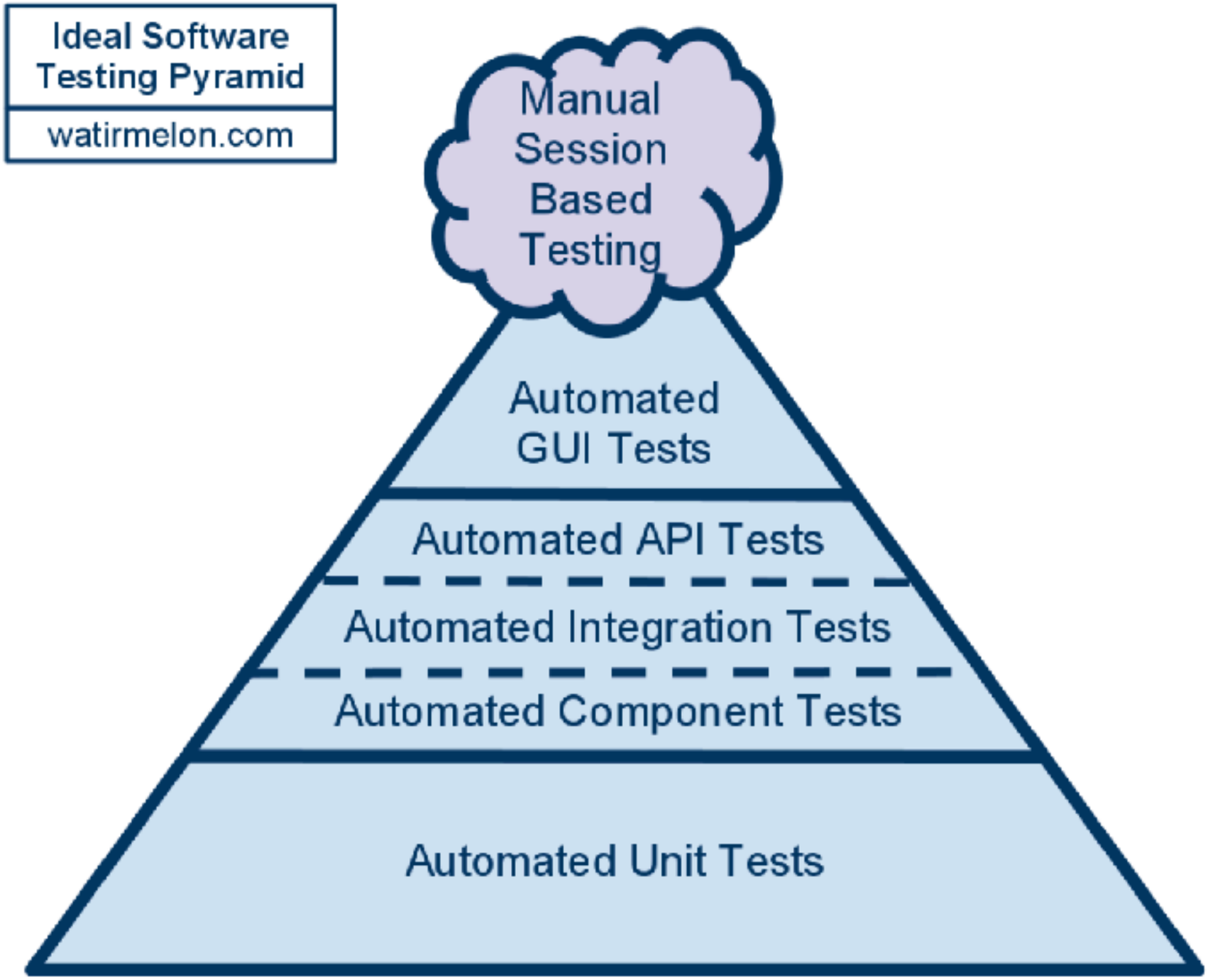
Structure of Automated tests

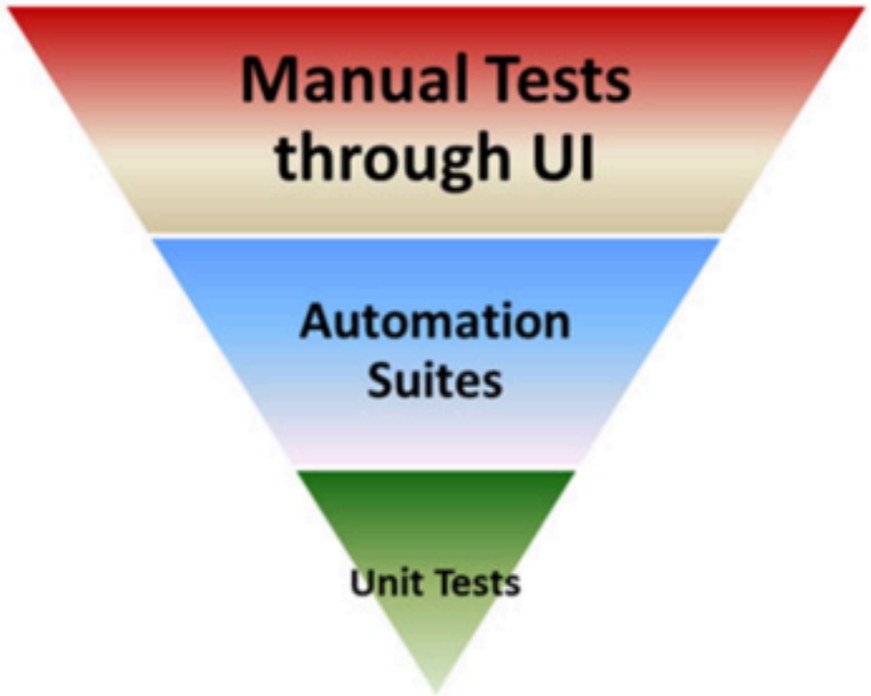


Working with Test double

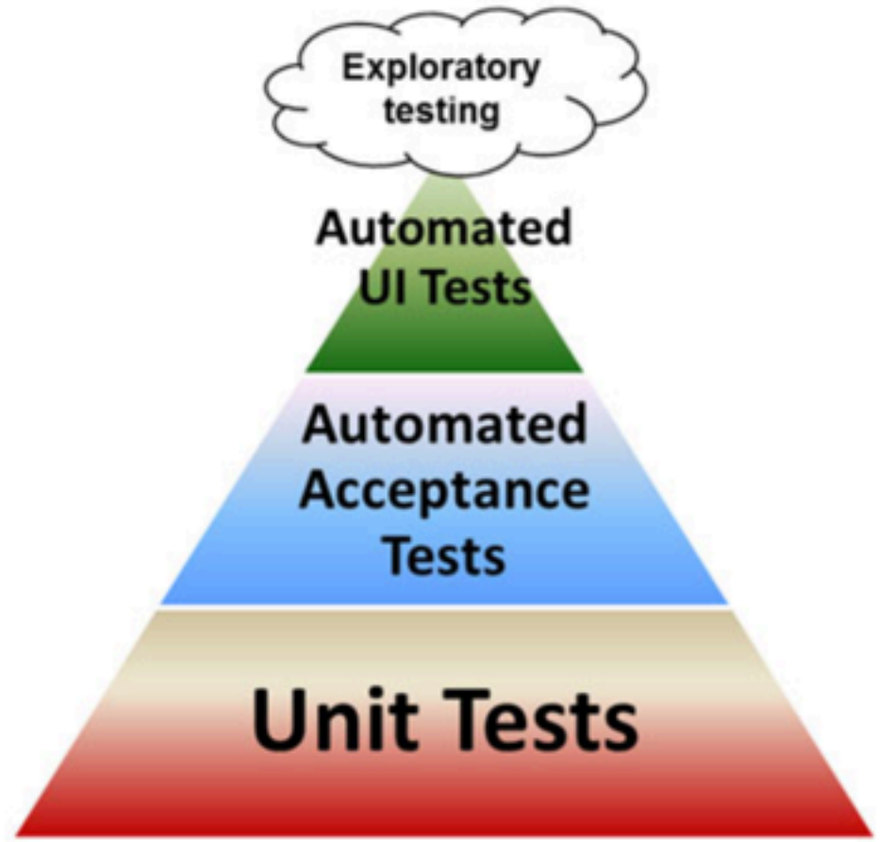




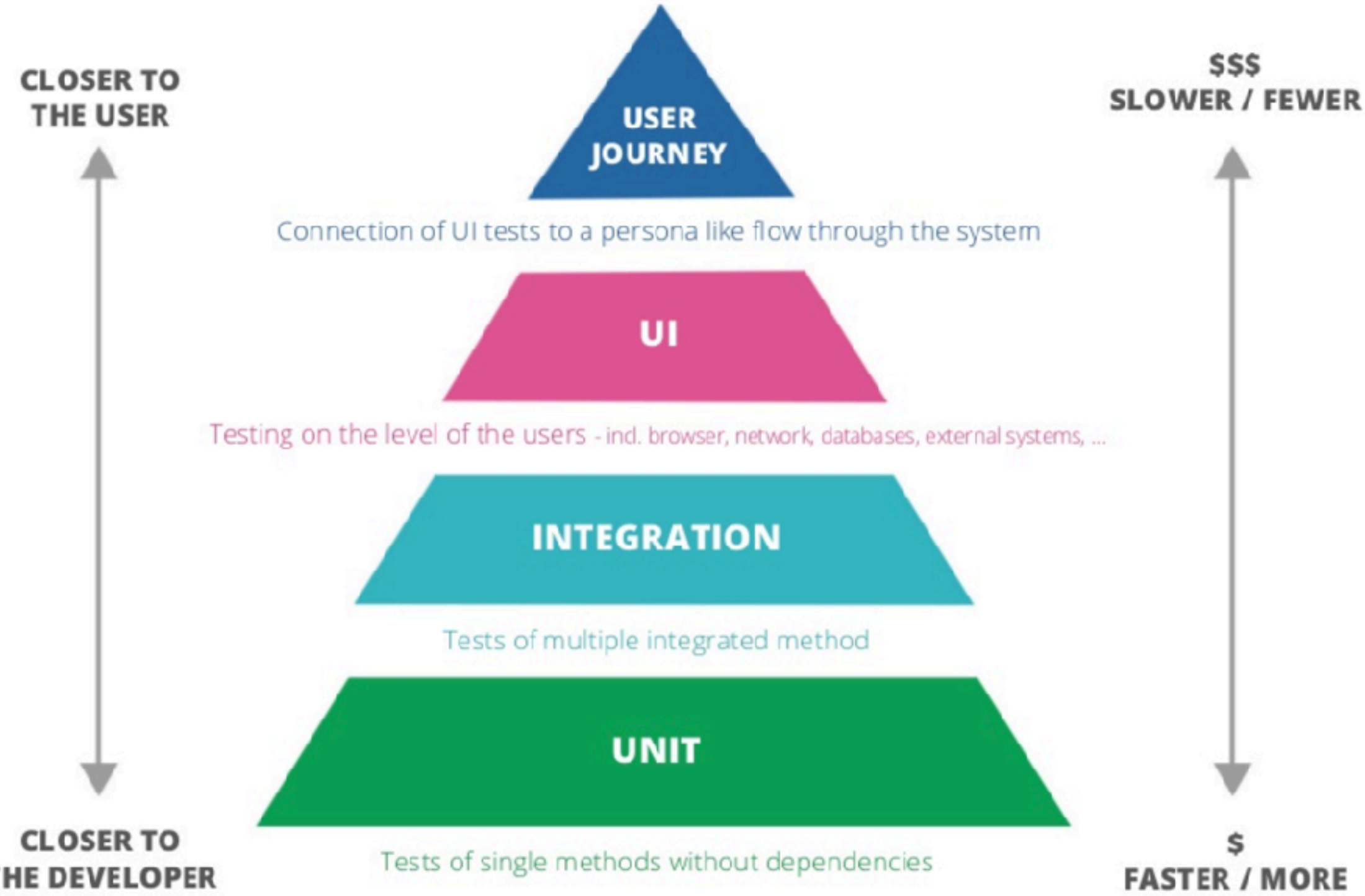




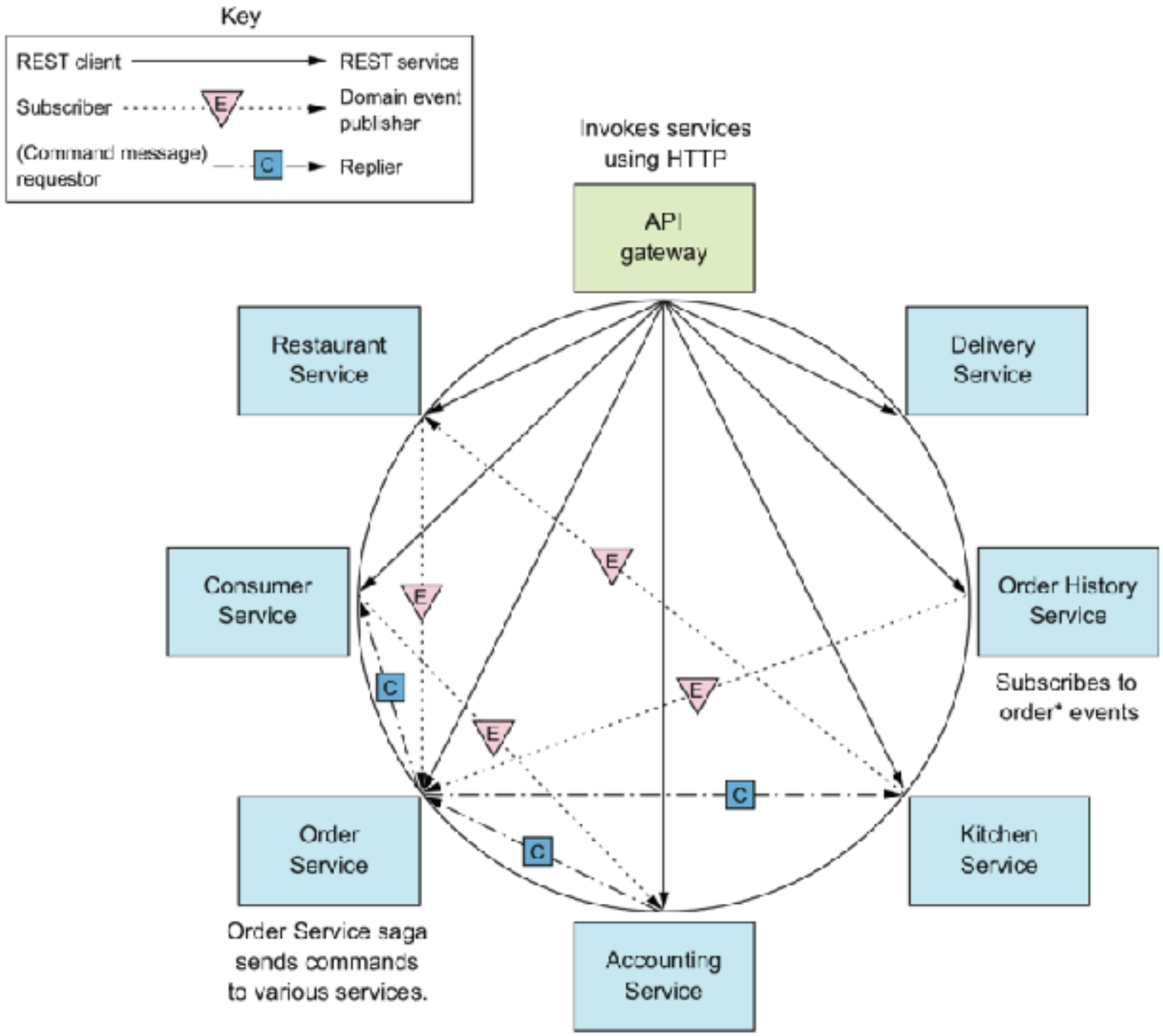
Traditional
(find bugs)



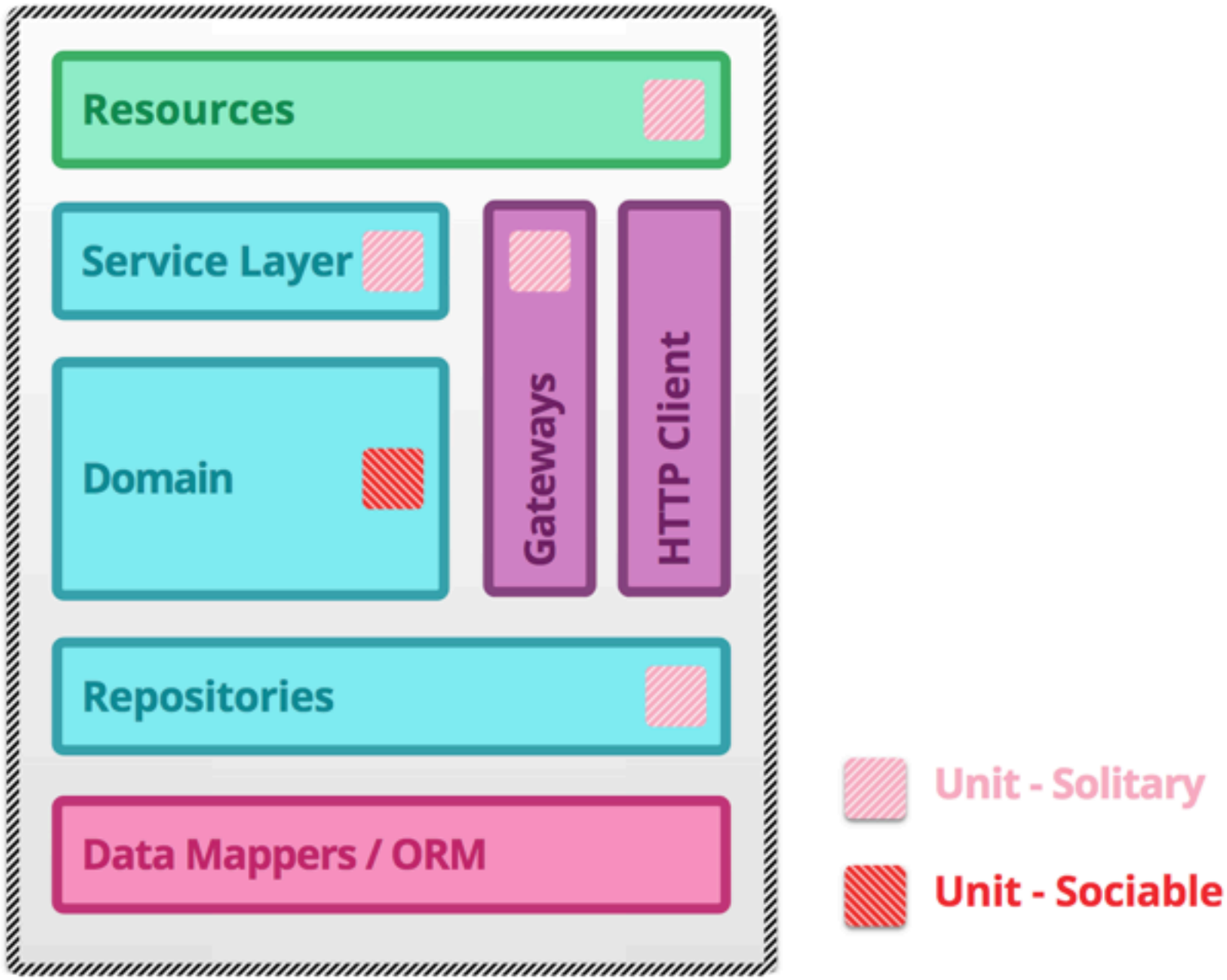
Agile
(prevent bugs)



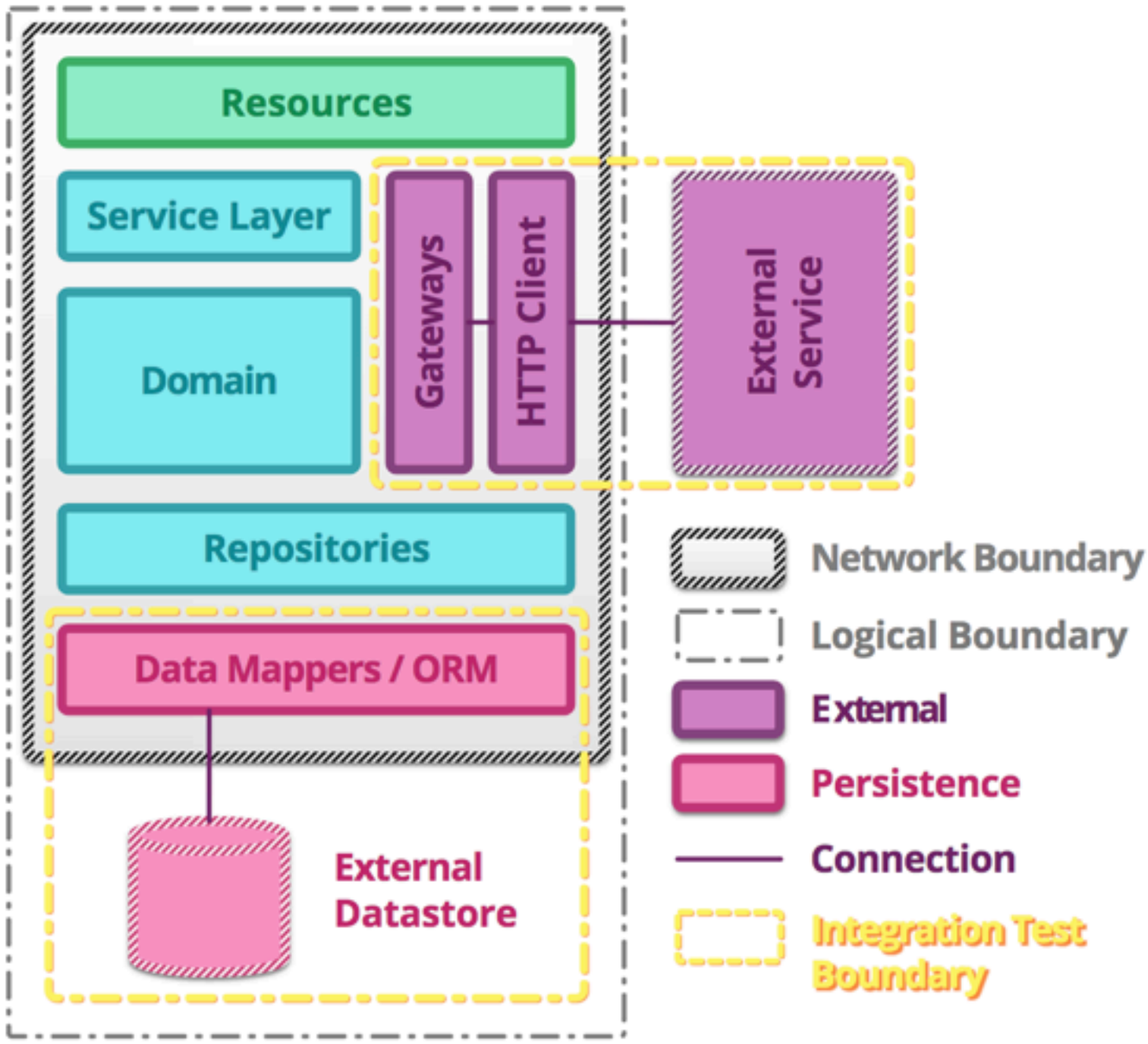
Testing Microservices ?



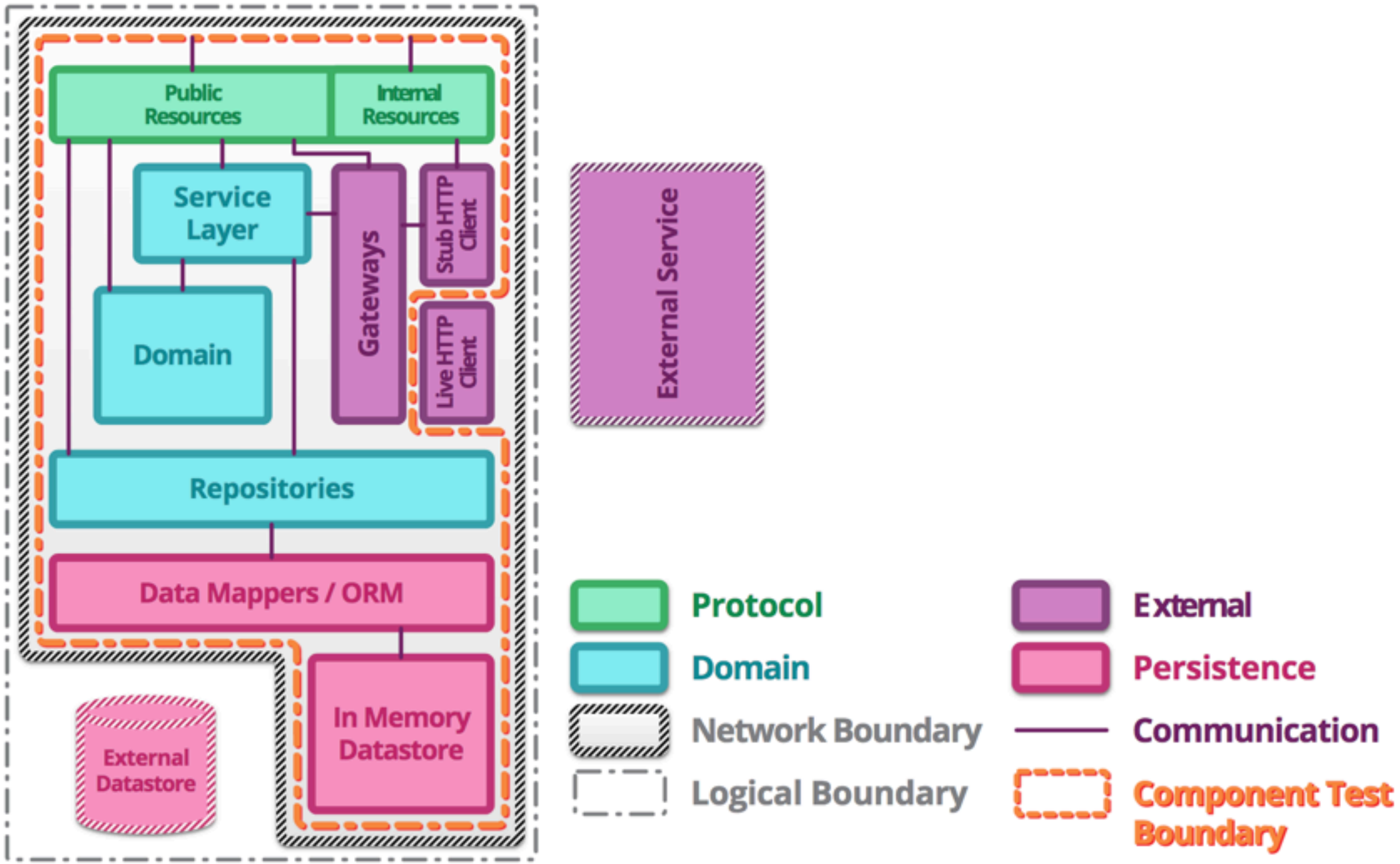
Unit testing



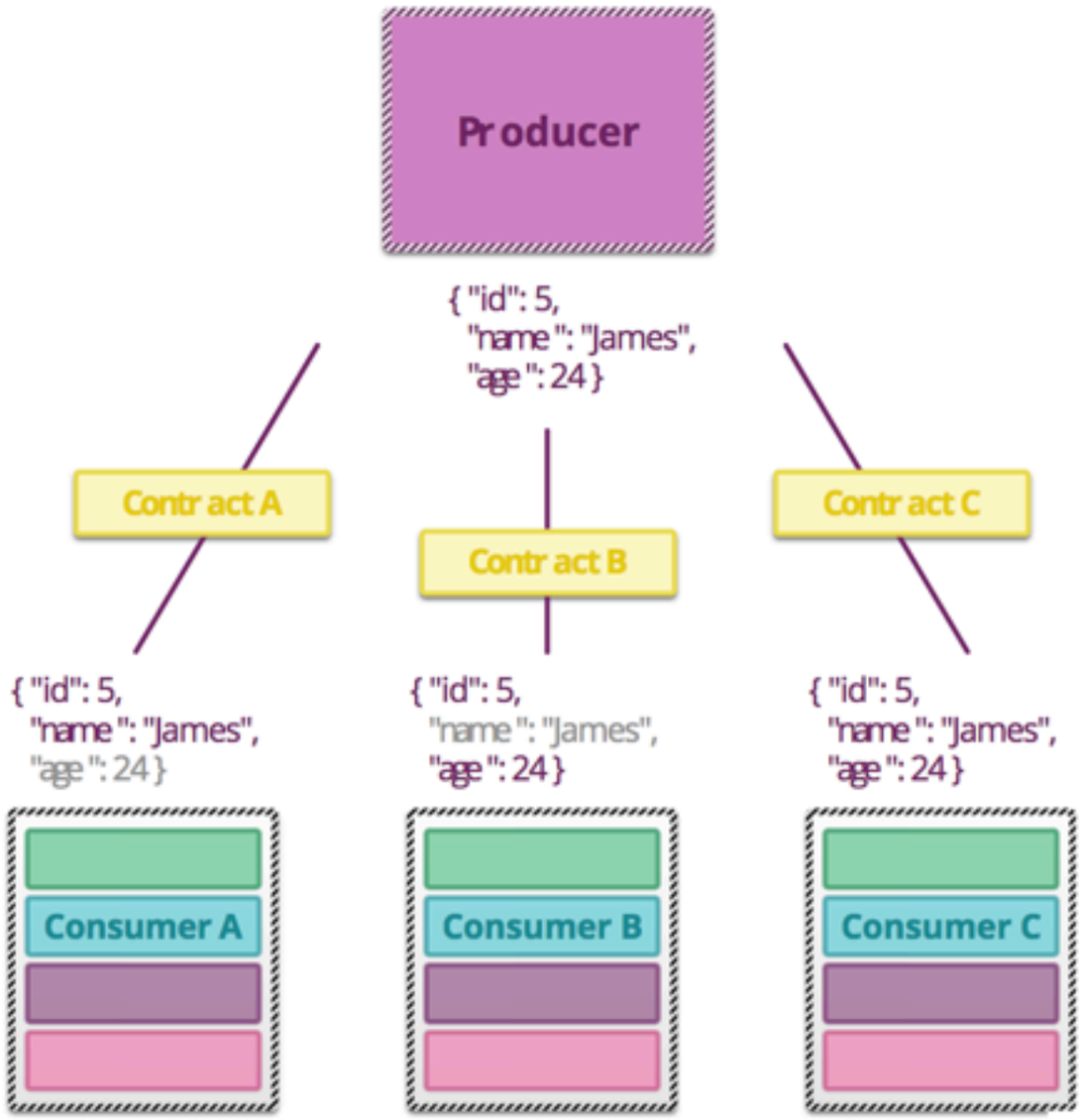
Integration testing



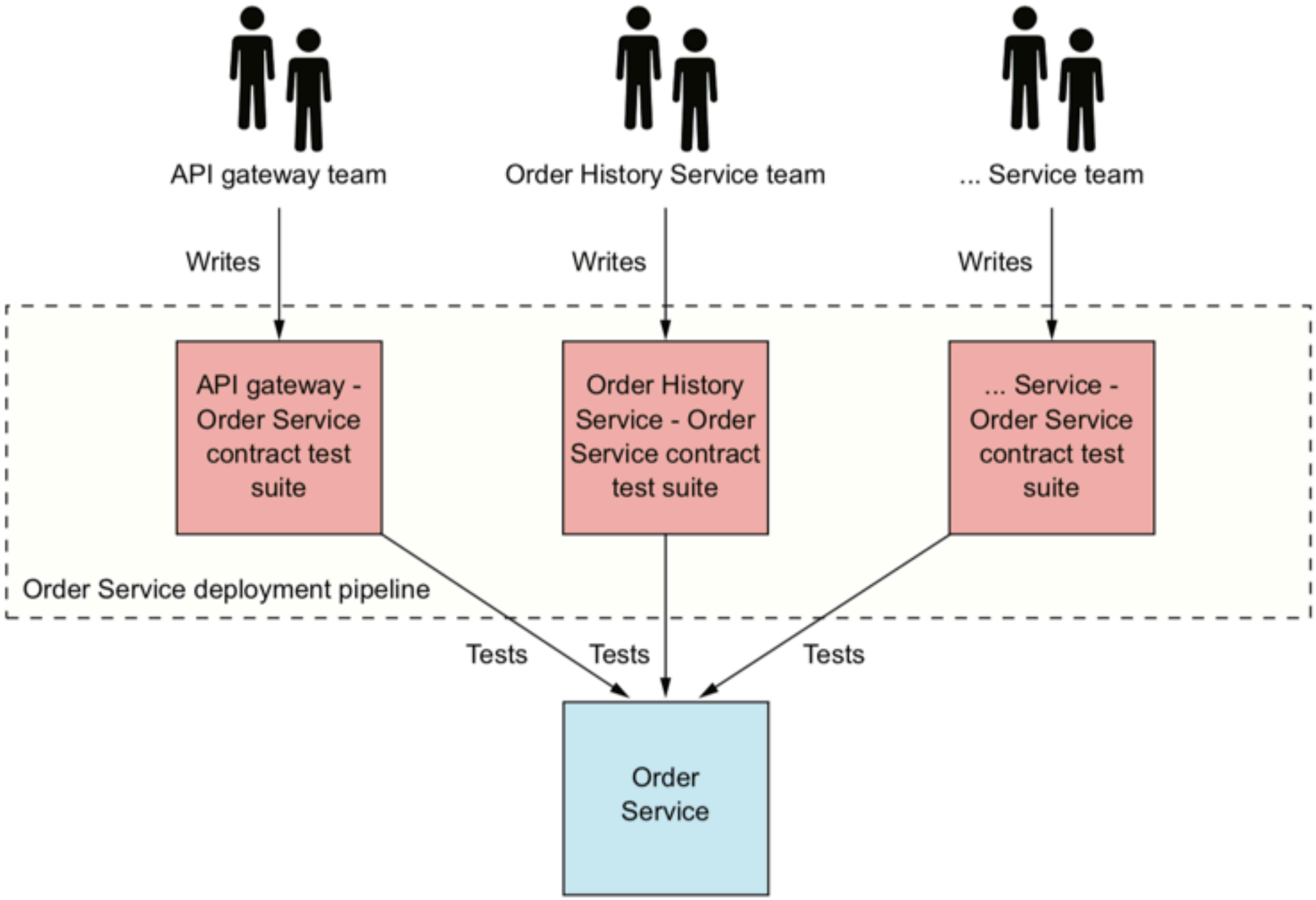
Component testing



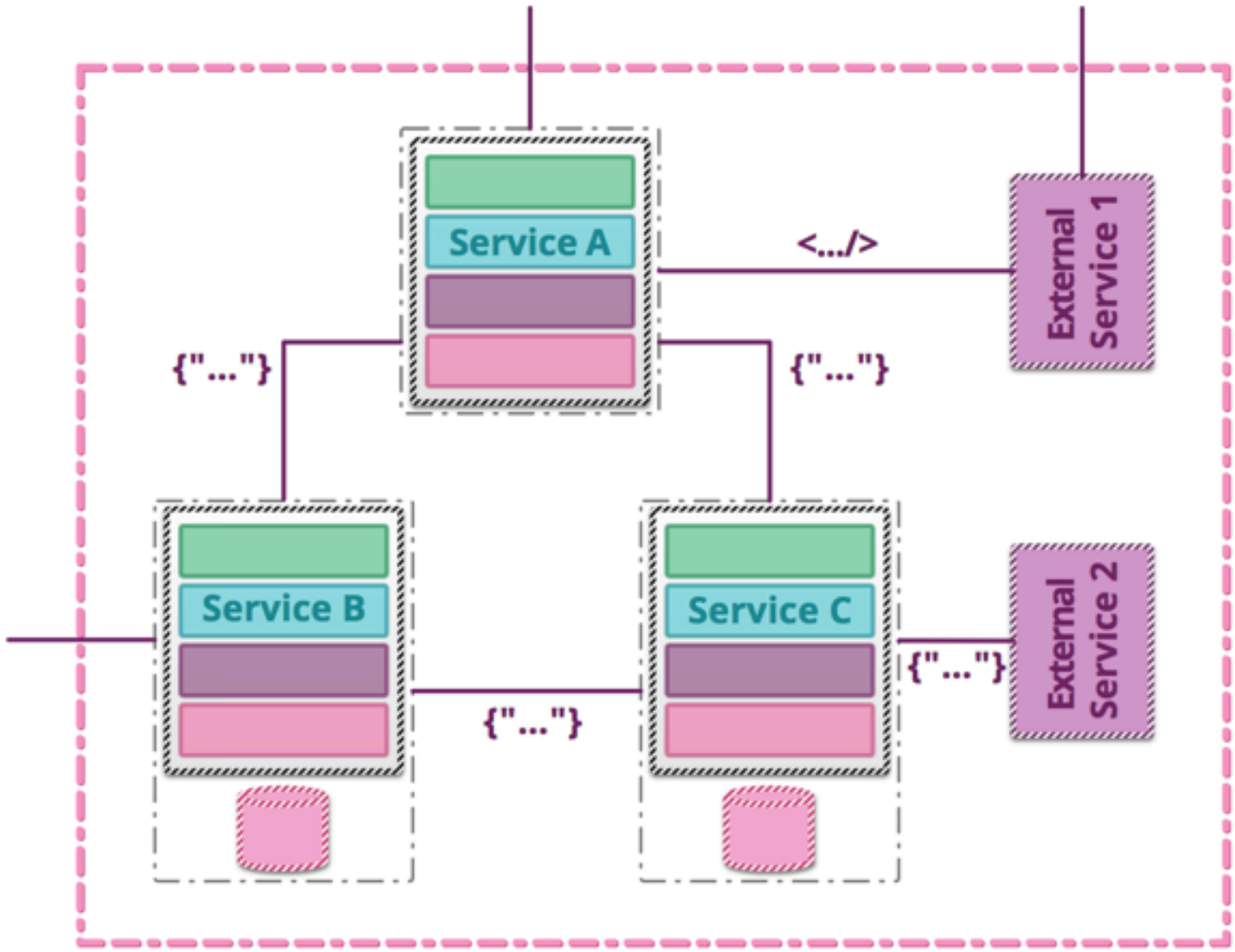
Contract testing



Consumer Contract testing



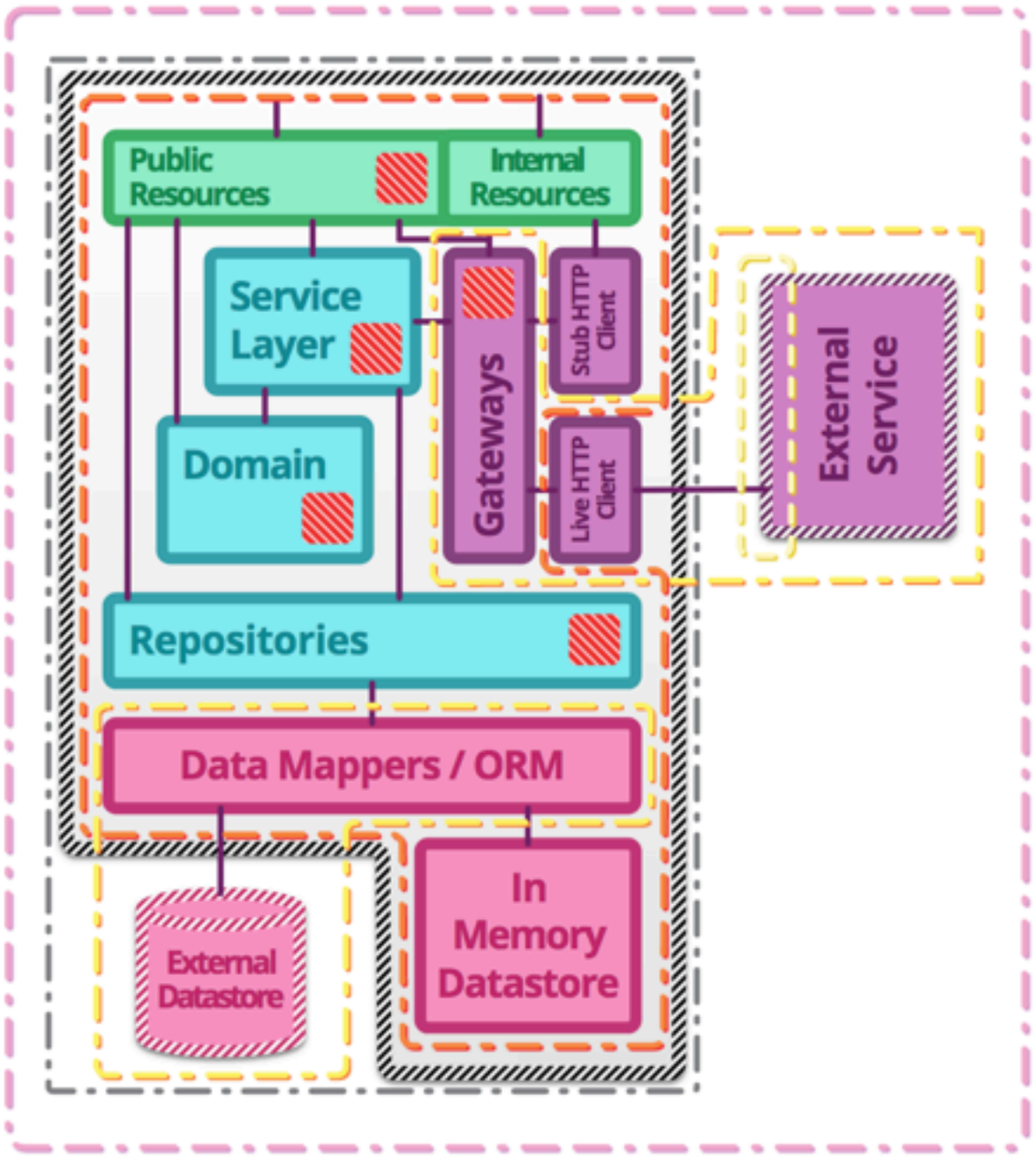
End-to-End testing



Summary

Unit tests : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.



Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

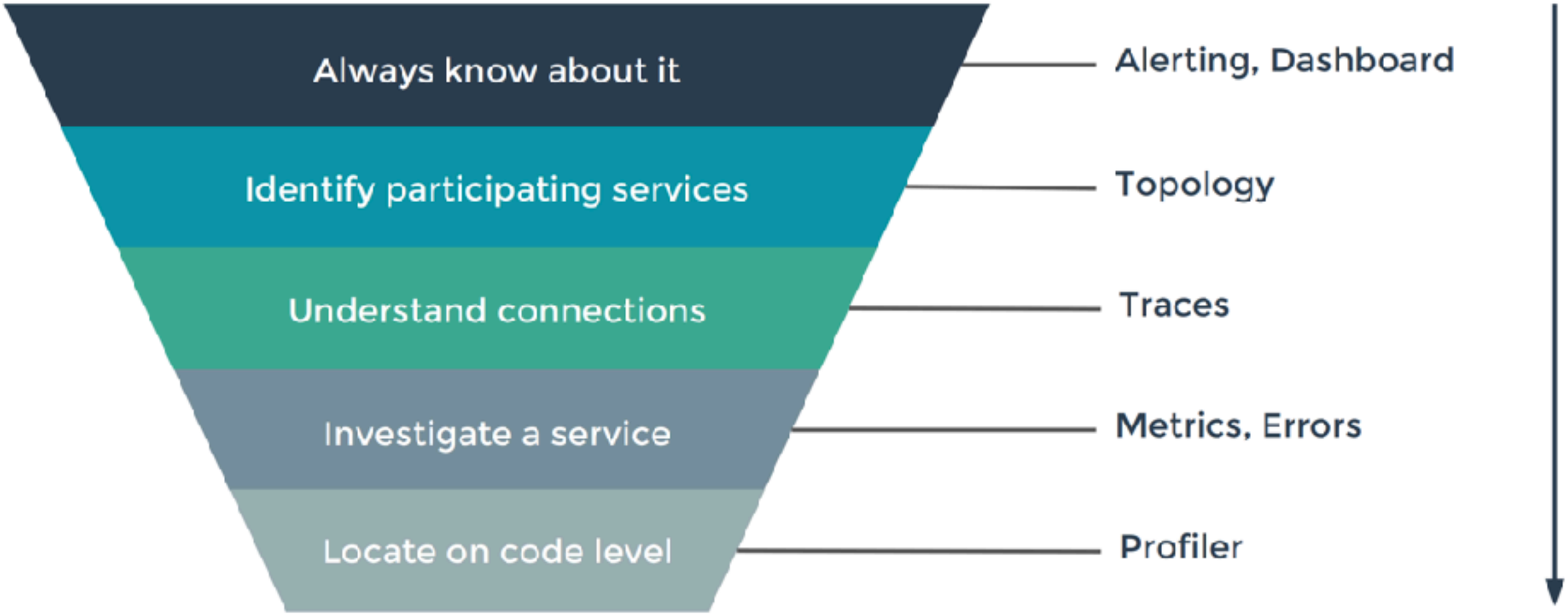
What is your testing strategy ?

Performance testing ?

Security testing ?

How to find an issue ?

How to find an issue ?



Develop production-ready services

Important quality attributes

Security
Configurability
Observability

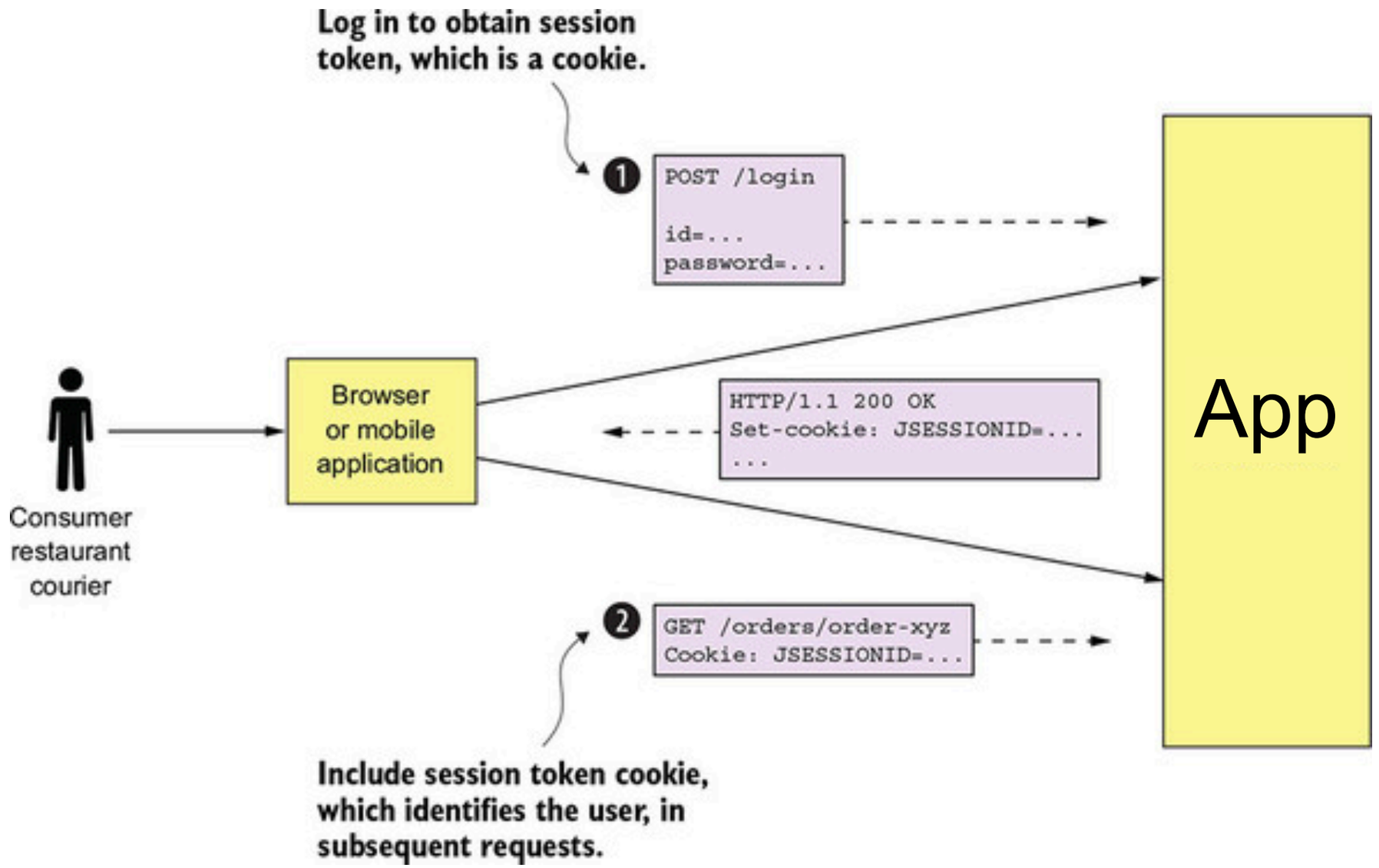
Secure services

Develop secure services

Authentication
Authorization
Auditing
Secure interprocess communication

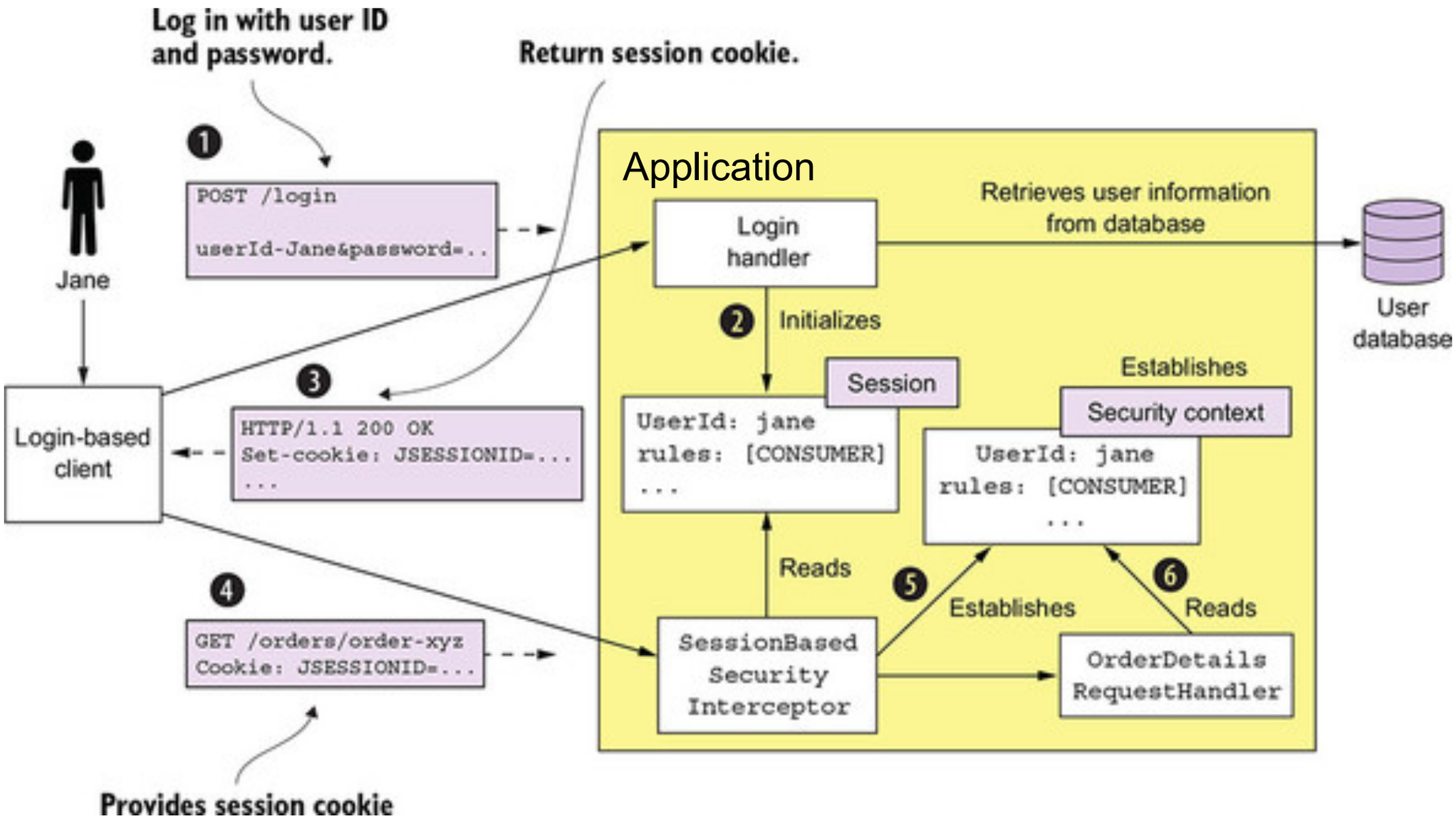
Security in traditional application

Keep security information in browser's cookie



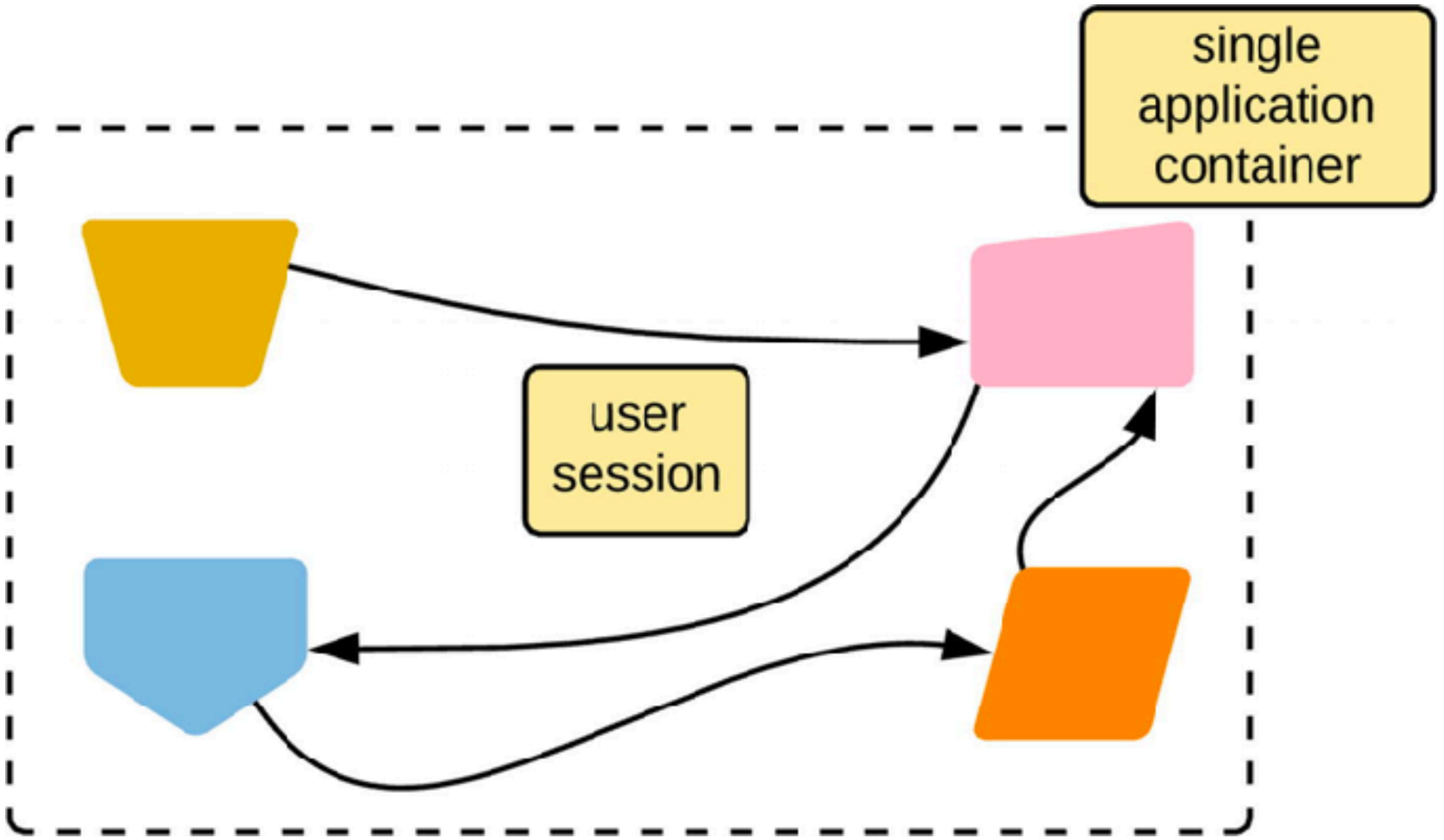
Security in traditional application

Implement security process in application



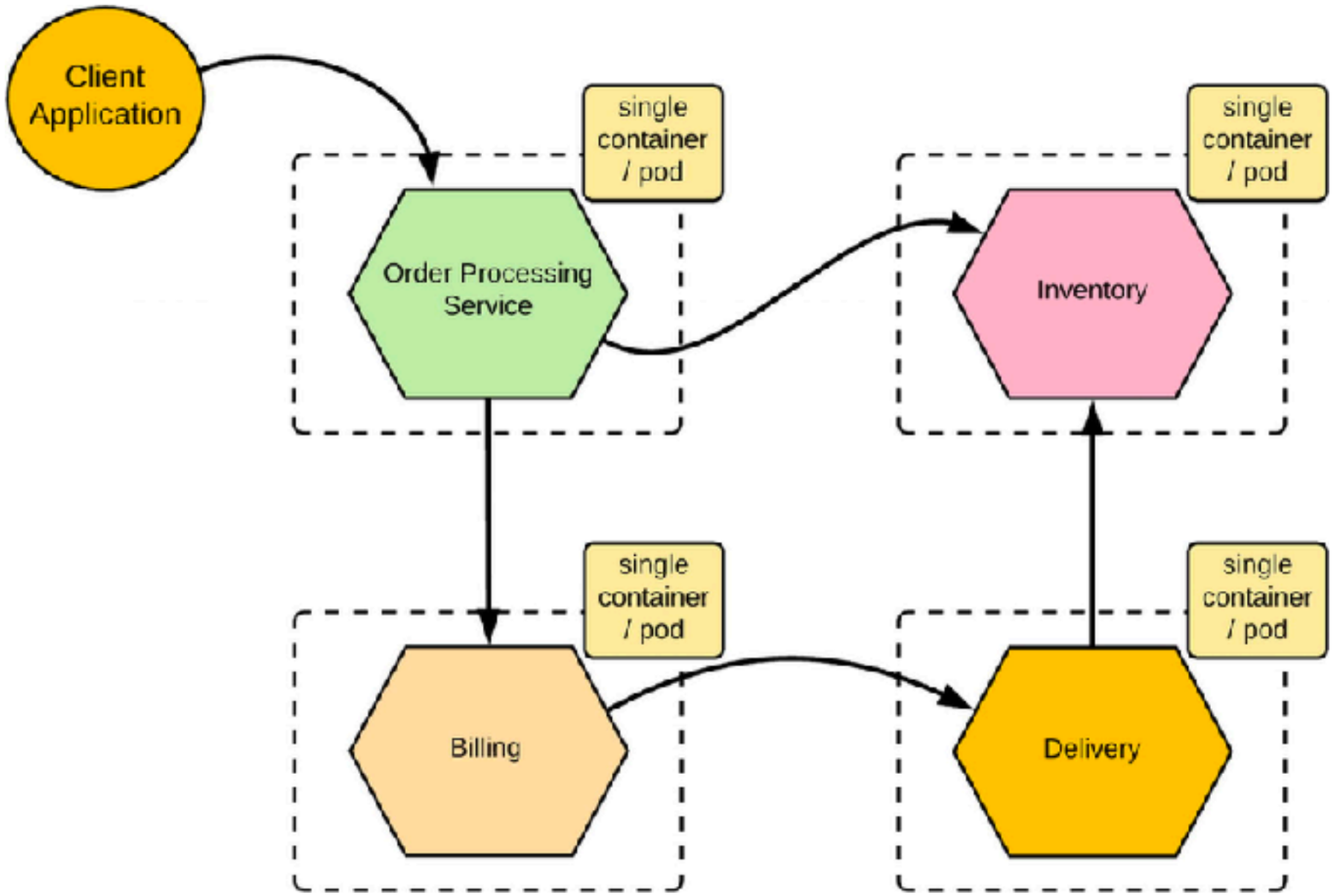
Security in traditional application

Sharing a user session in application



Security in multiple services ?

Interaction between multiple services



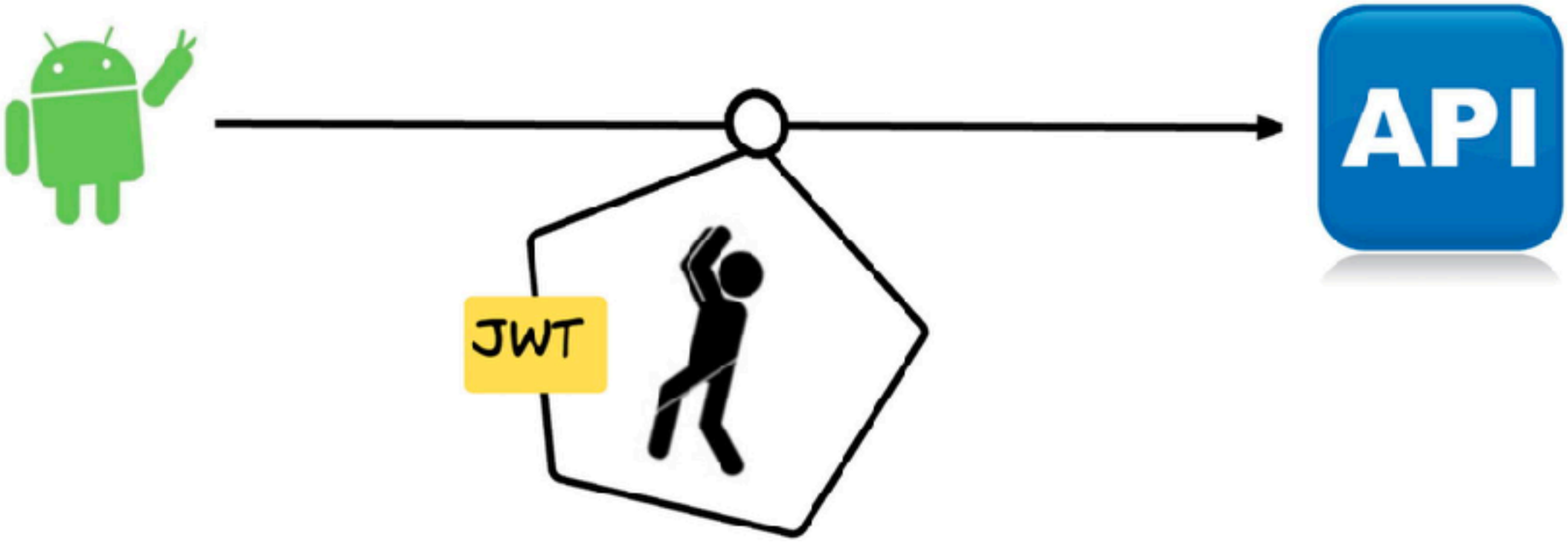
Secure service-to-service communication

JSON Web Token (JWT)

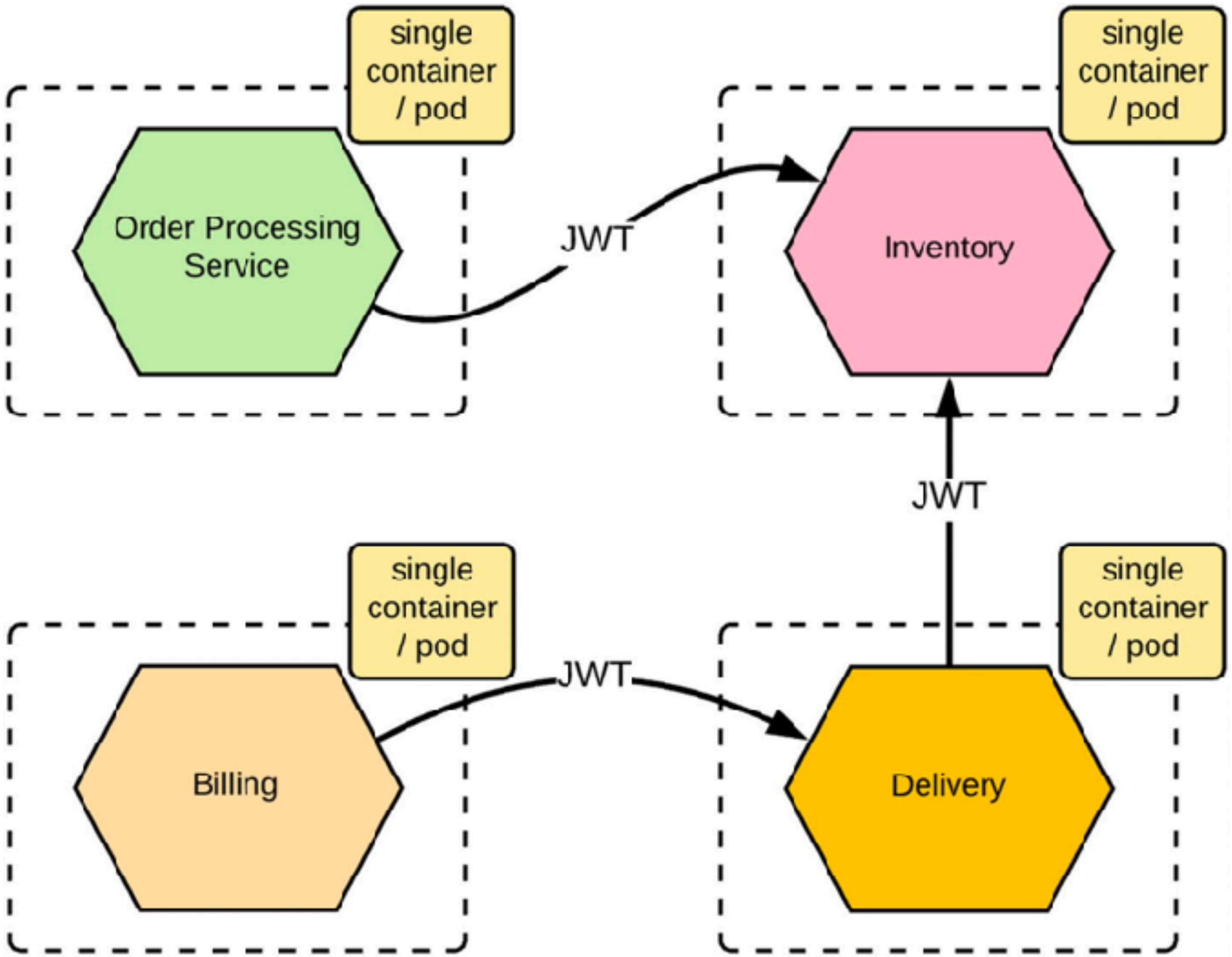
Transport Layer Security (TLS) mutual authentication

JSON Web Token (JWT)

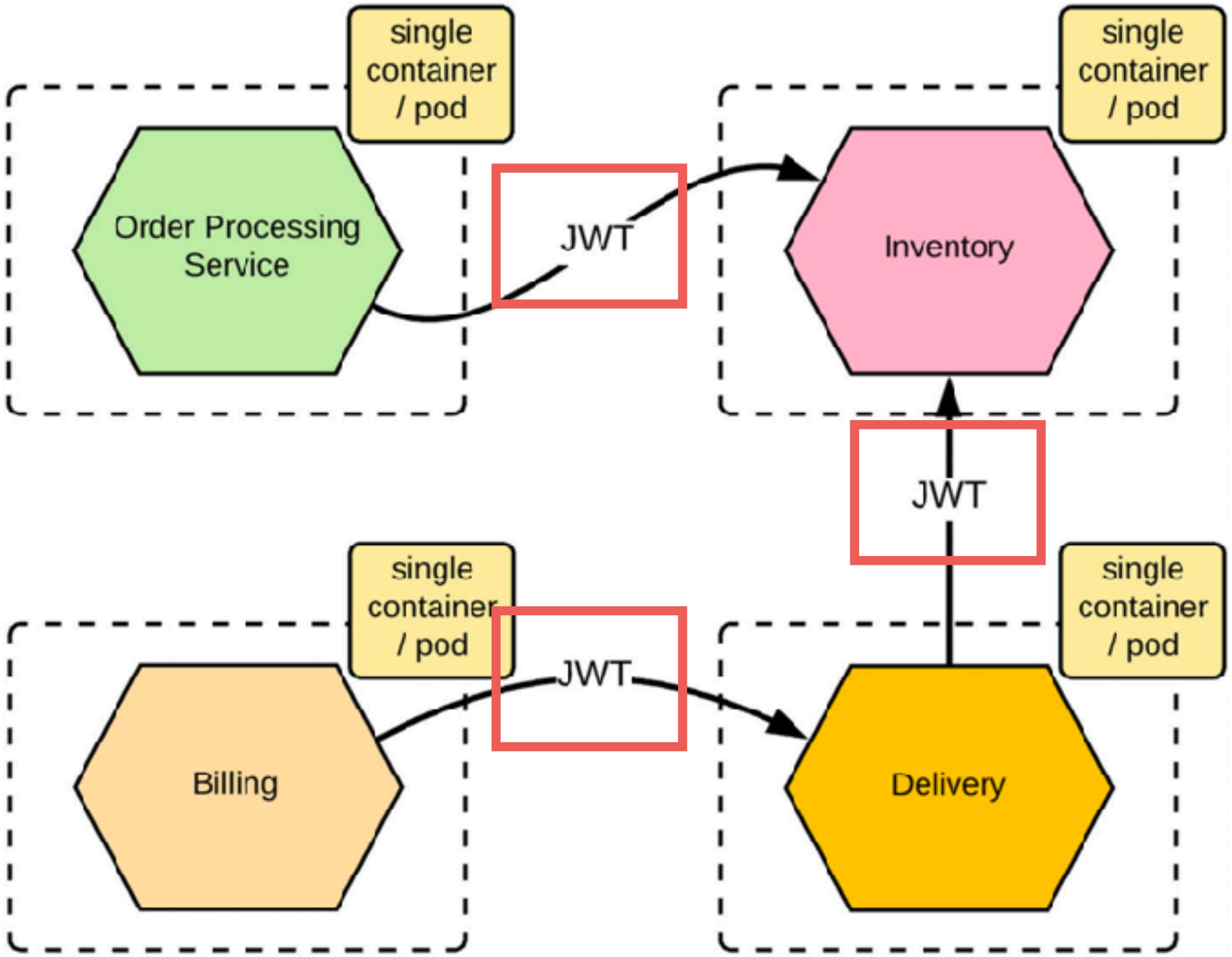
Define a container to transport data between interested parties



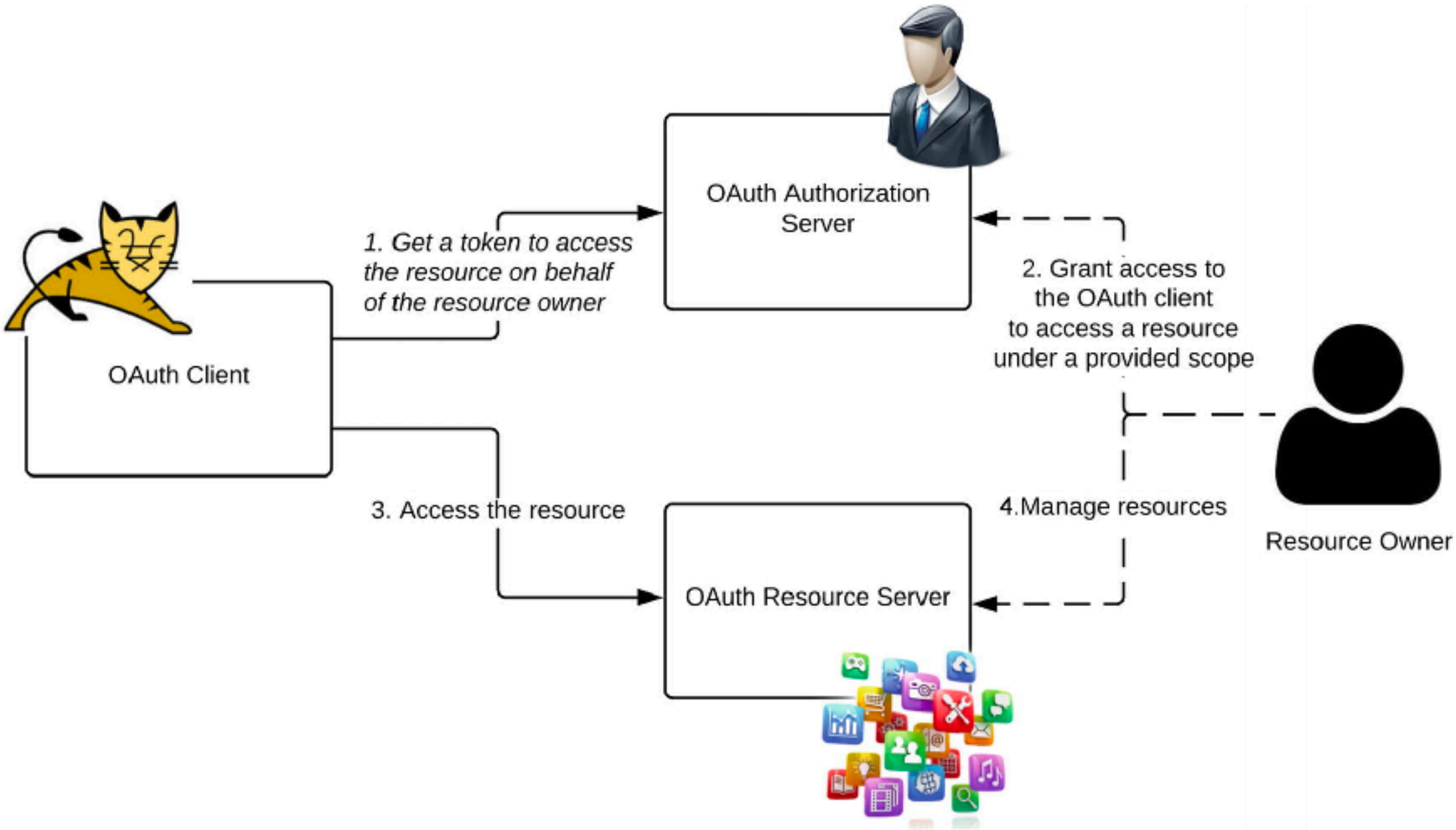
Passing user context as JWT



Problem ?

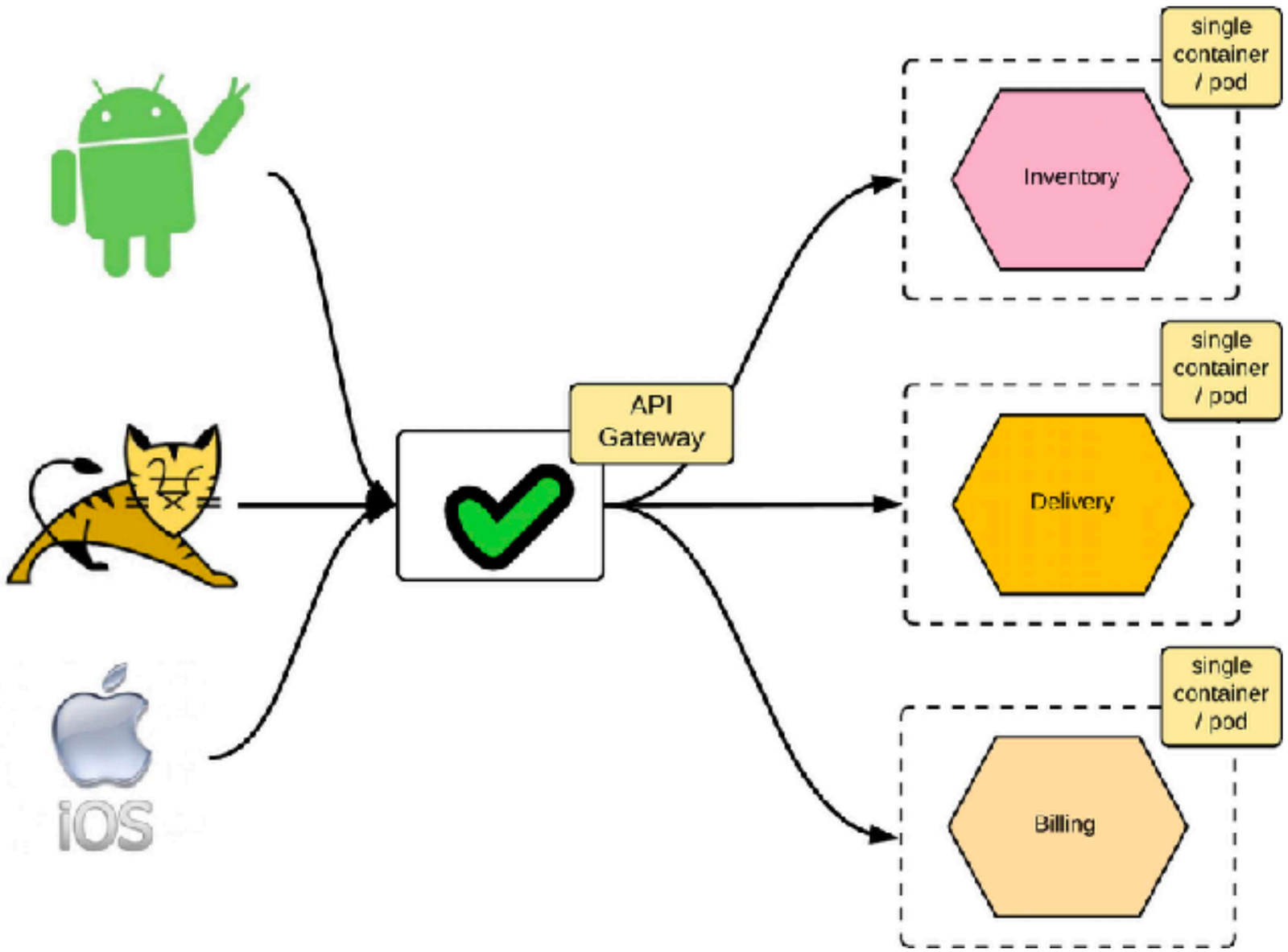


OAuth 2.0



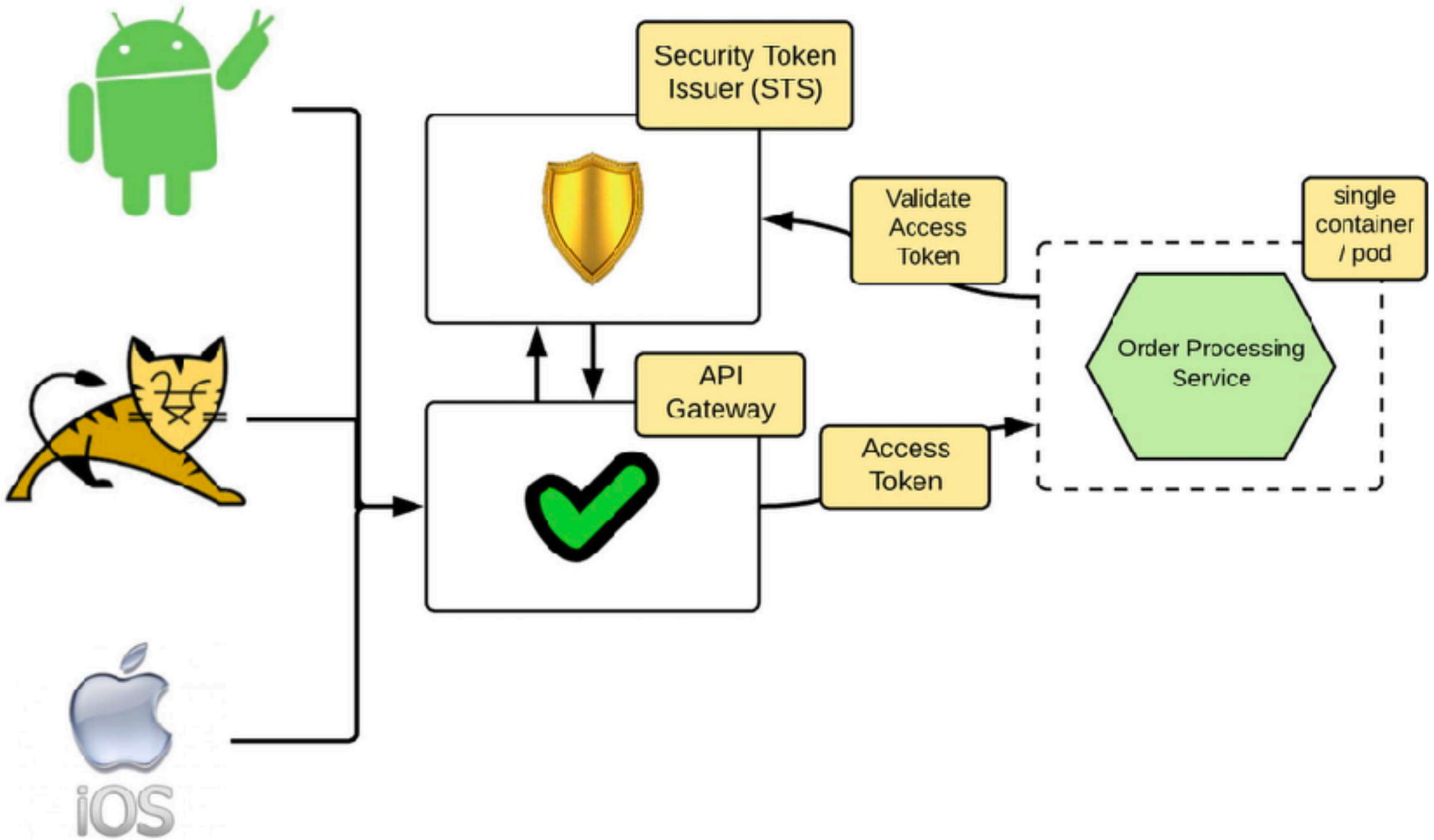
Centralize pattern

Using API gateway pattern

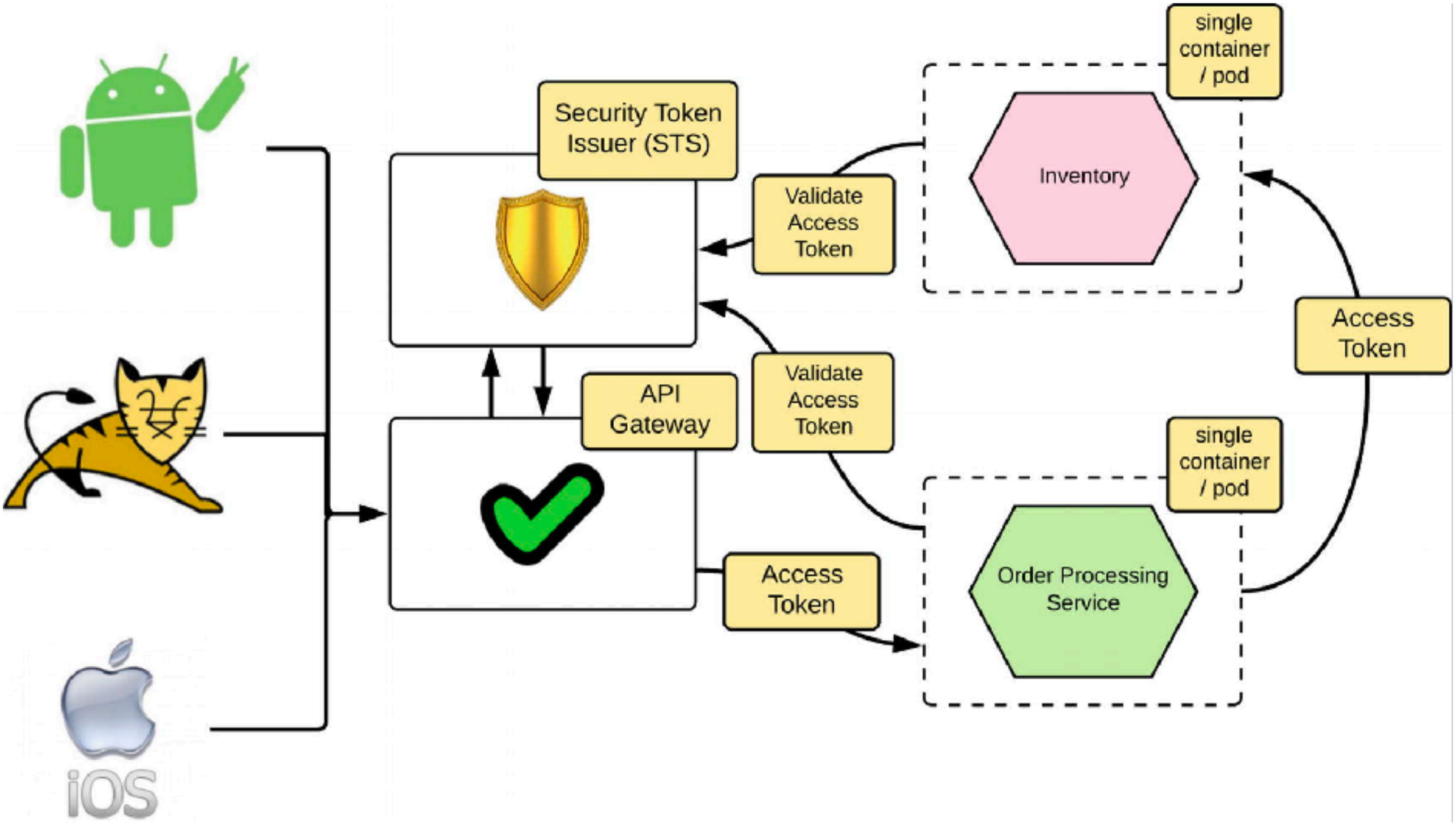


Centralize pattern

Using API gateway pattern

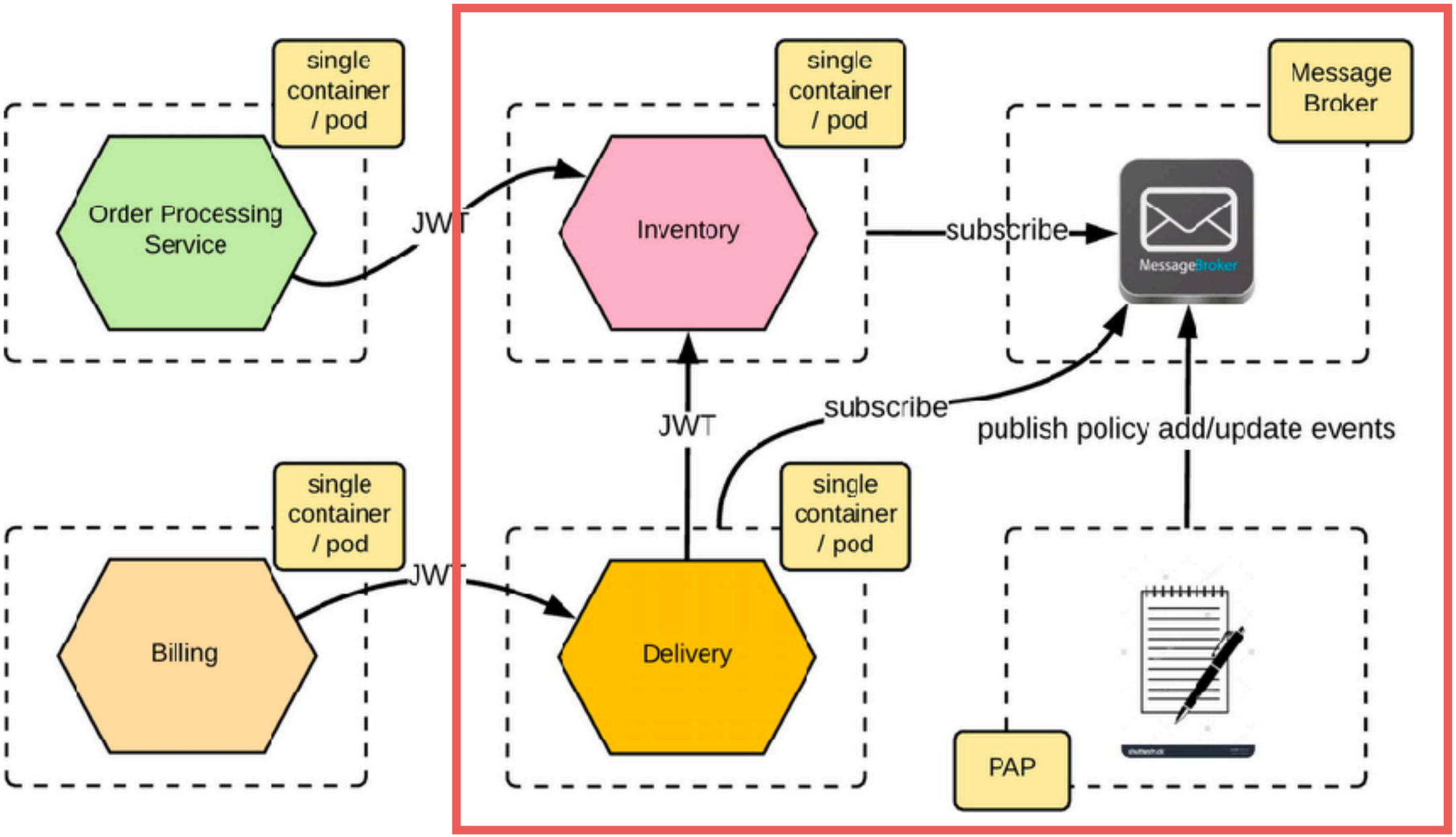


API gateway with OAuth 2.0



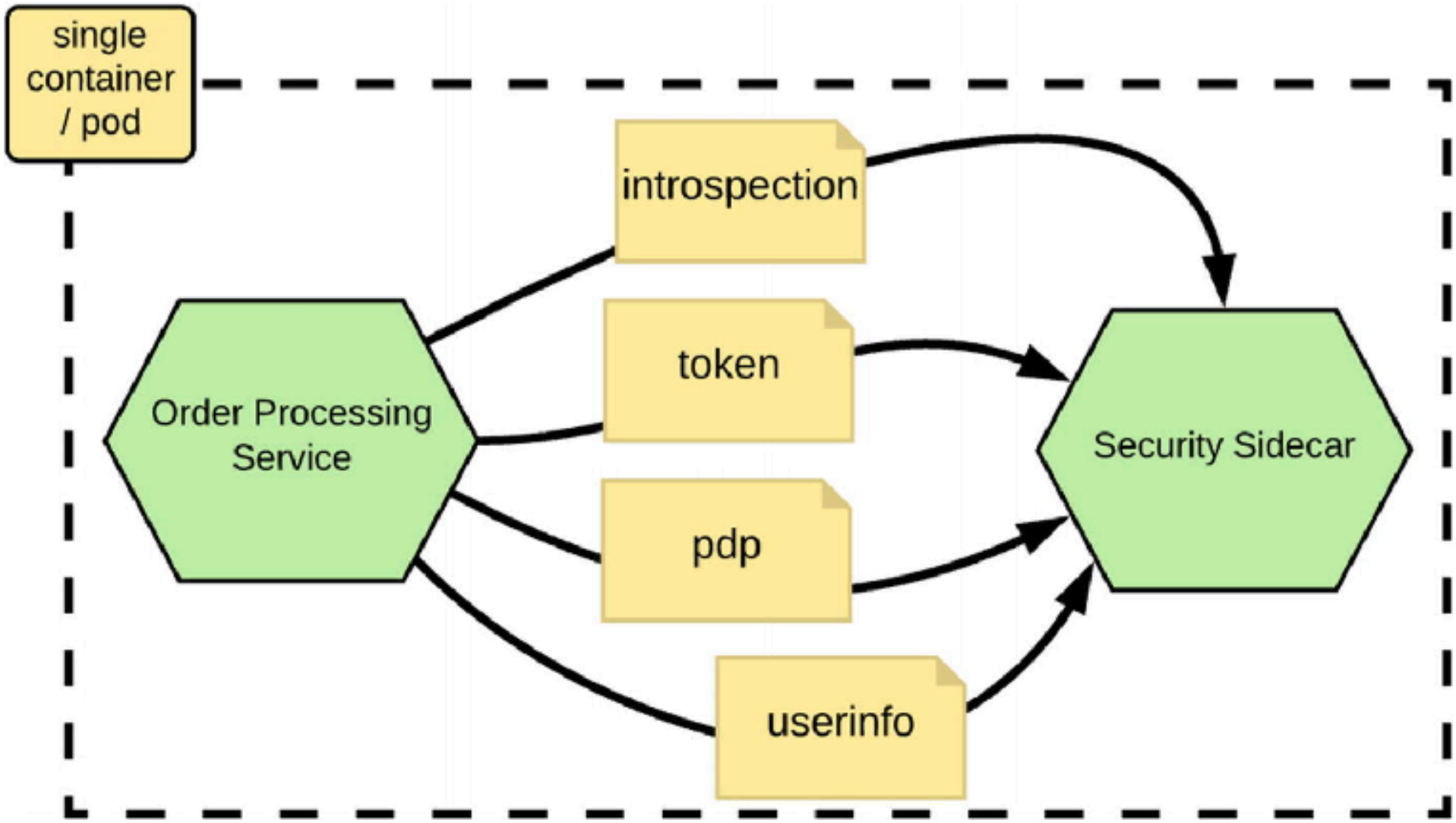
Access control of services

Policy Administration Point

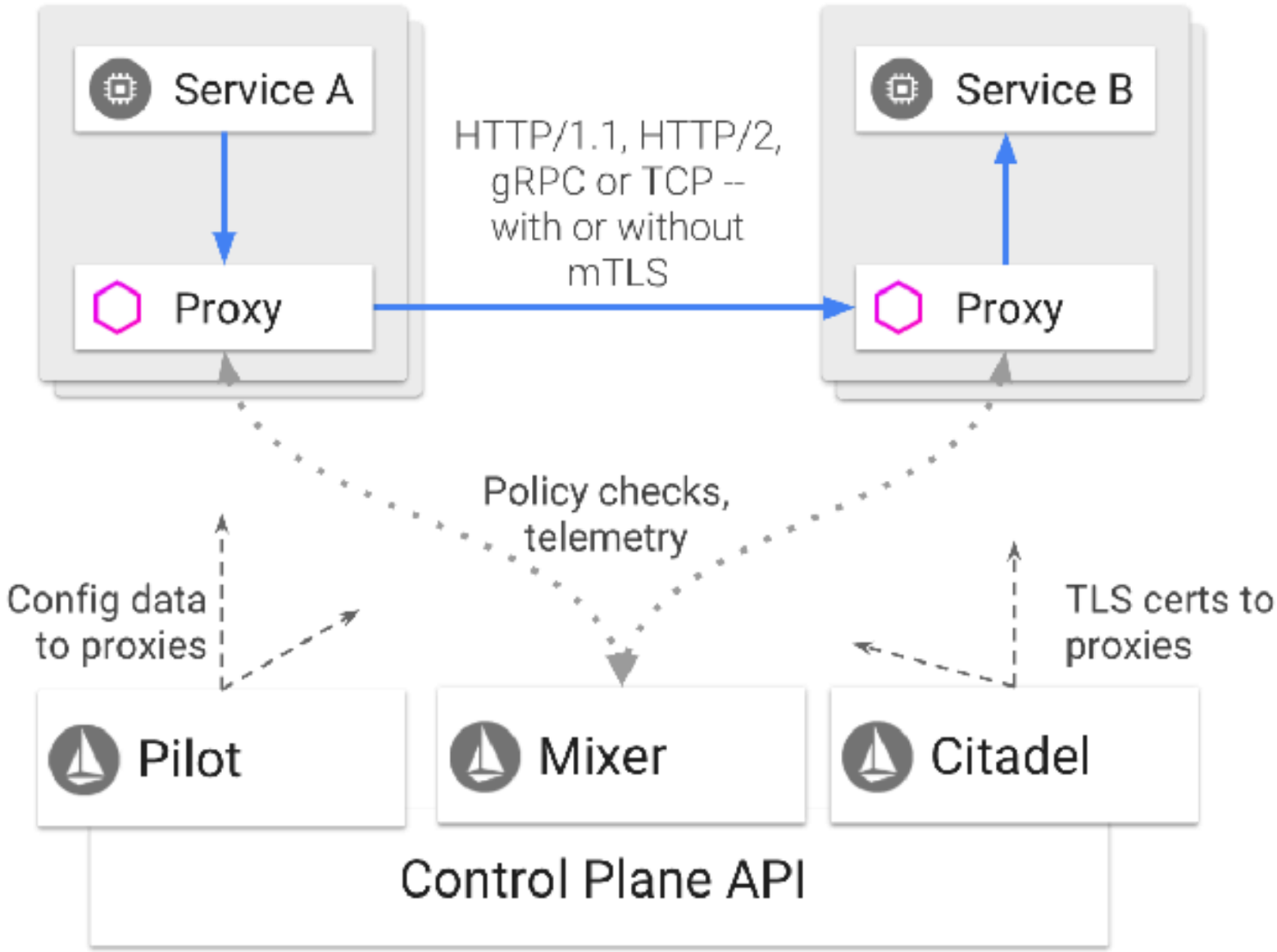


Working sidecar

Security sidecar



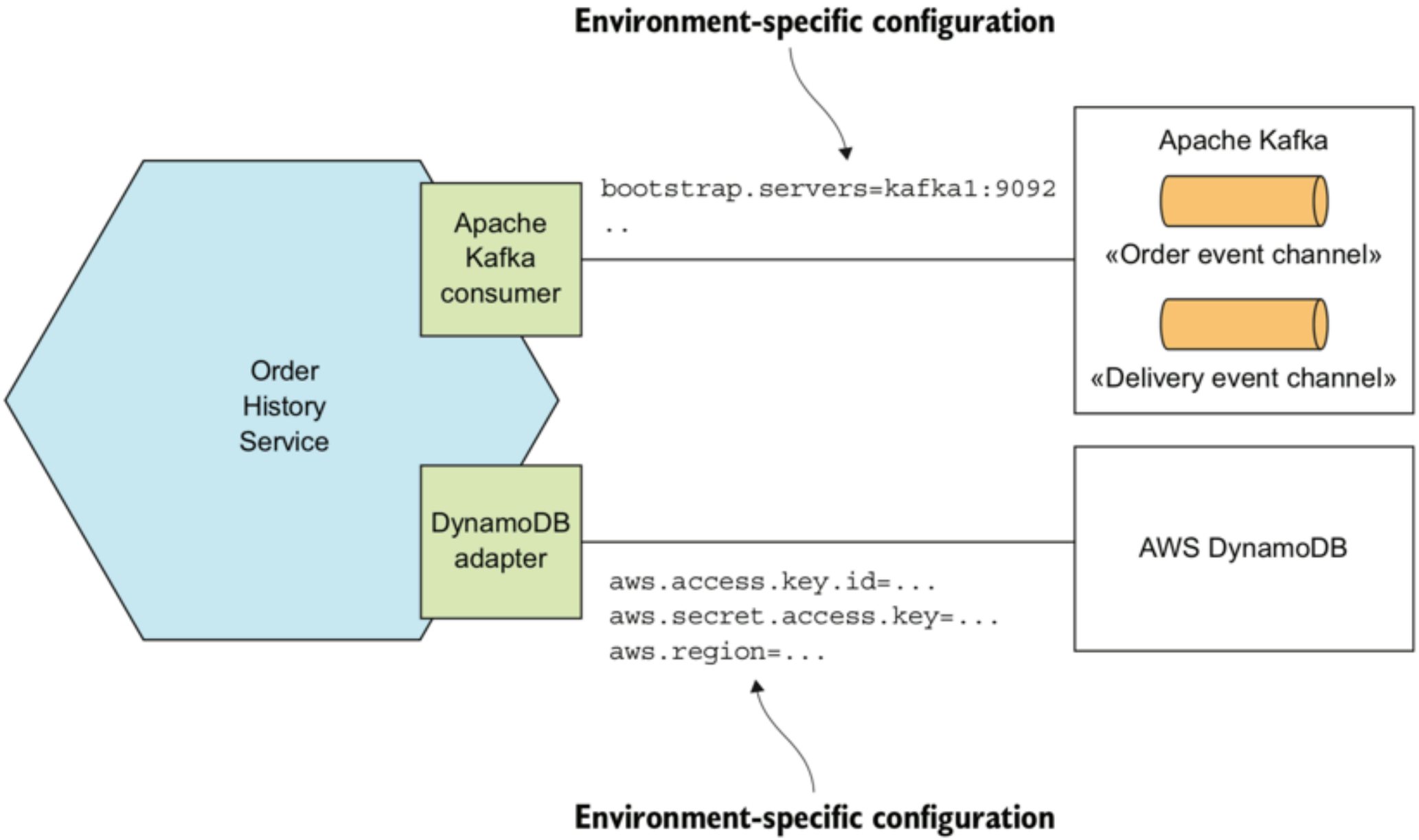
Security sidecar



https://istio.io/docs/concepts/what-is-istio/

Configurable services

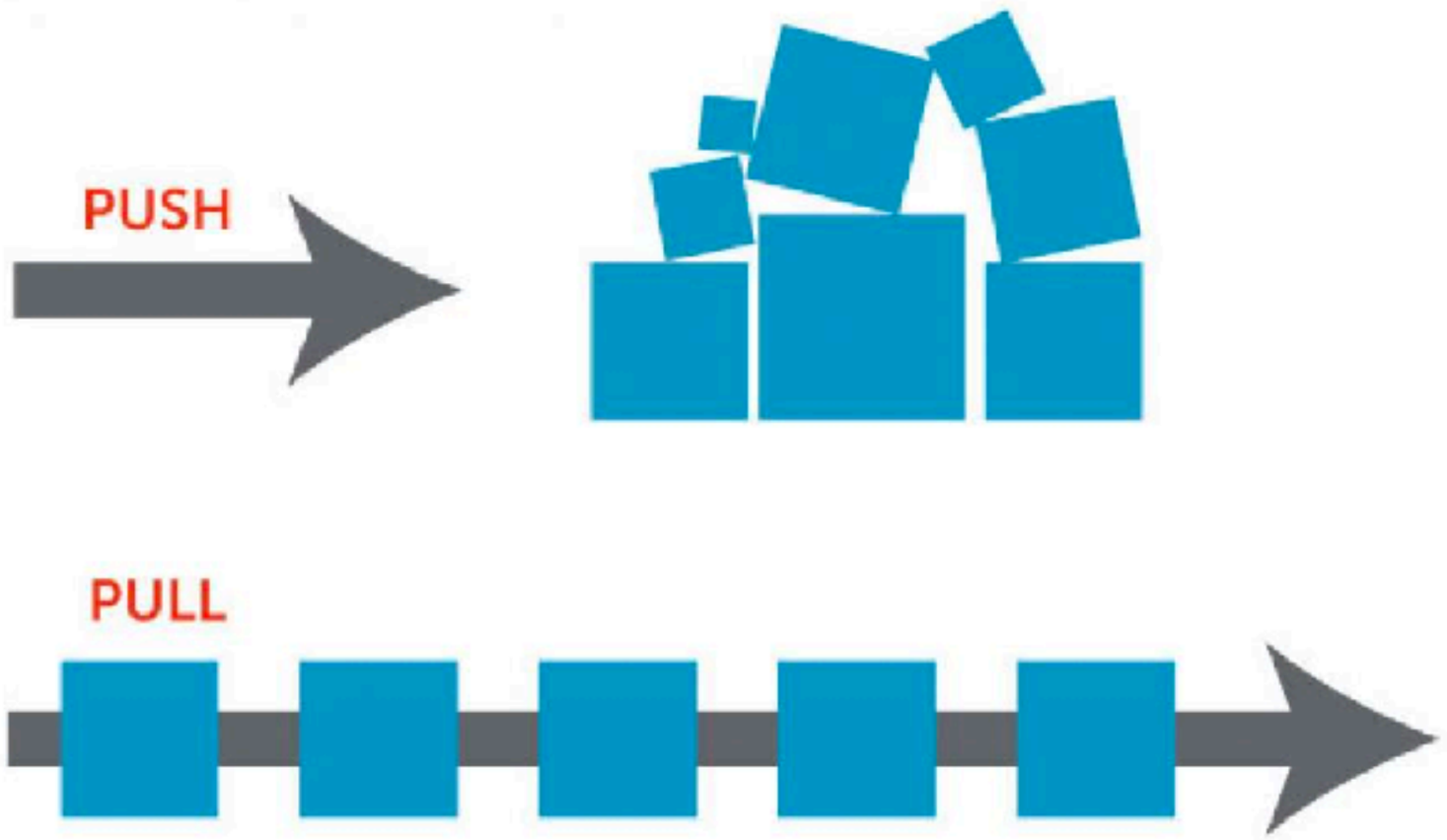
Design configurable services



External configuration models

Push model

Pull model

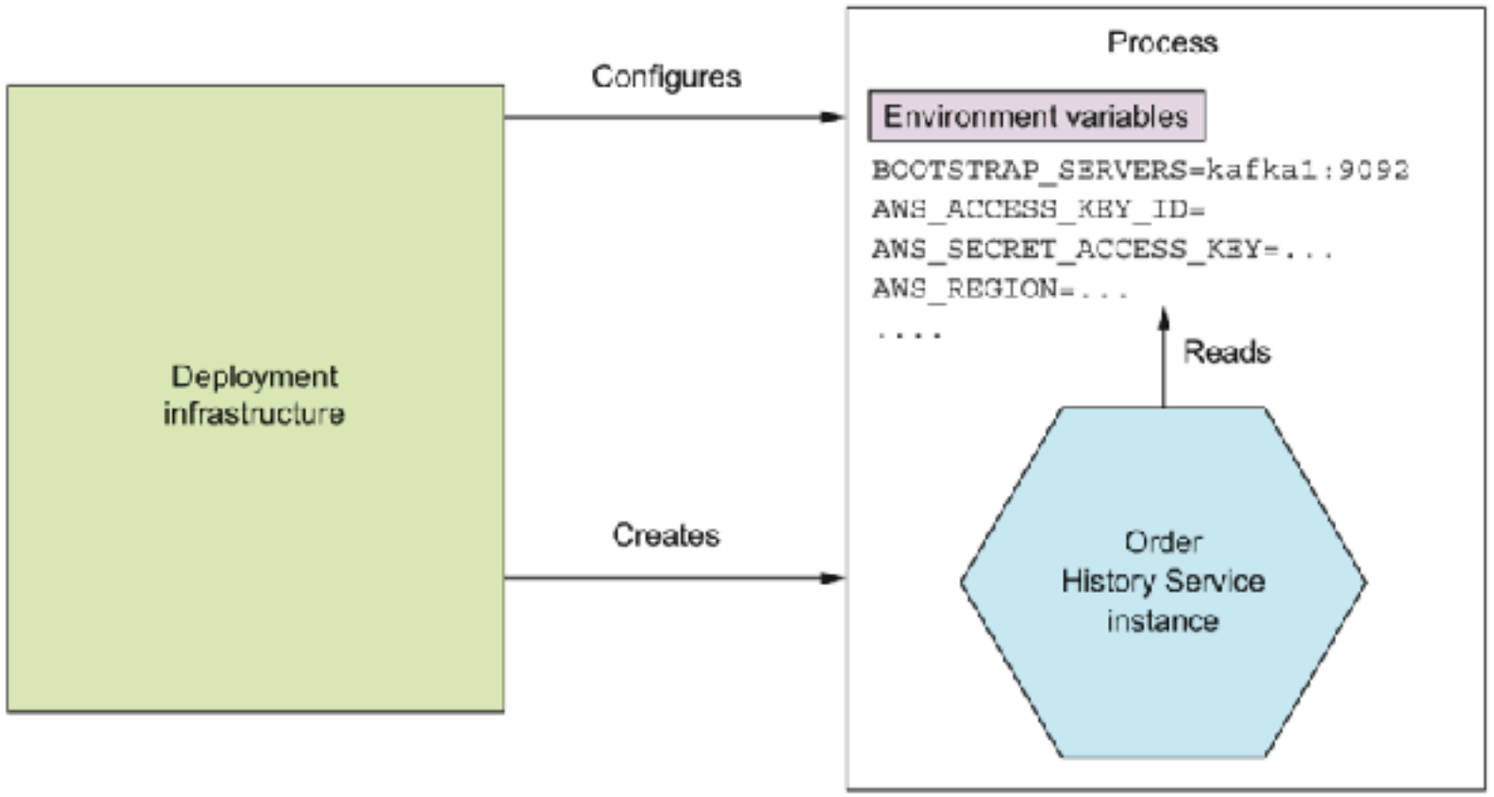


Push model

Pass the configuration to service

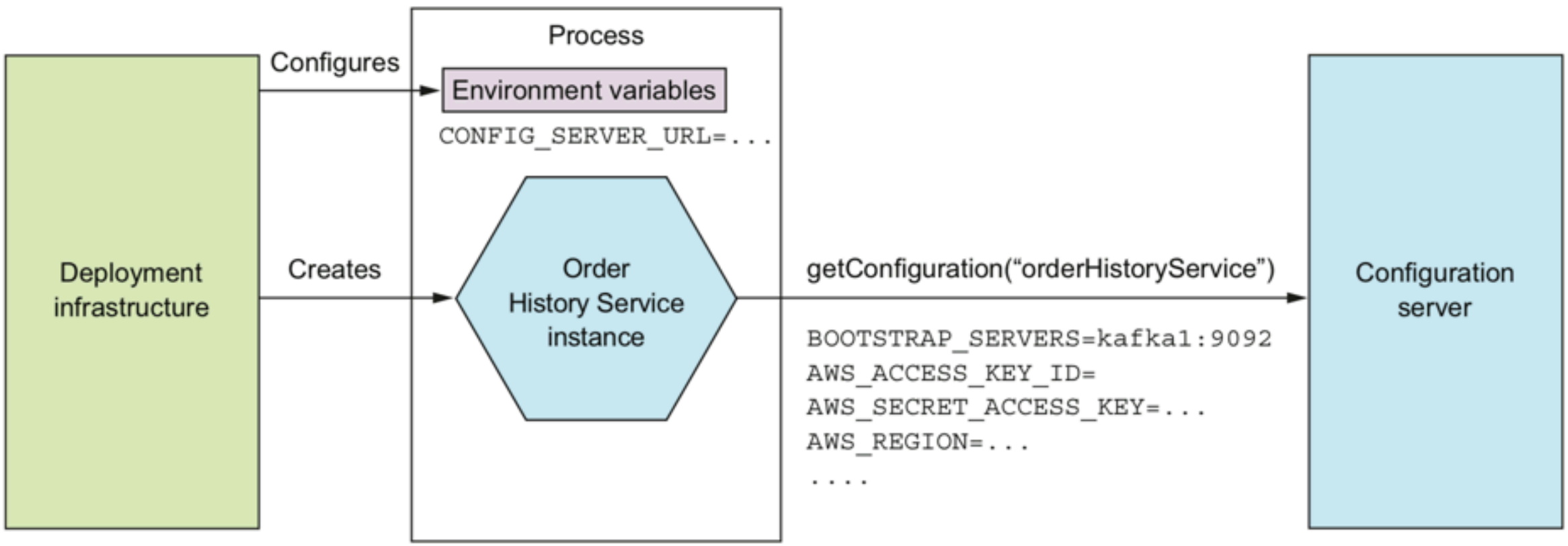
OS environment variables

Configuration files



Pull model

Service read configuration from configuration server



Benefits of configuration server

Centralized configuration
Transparent decryption of sensitive data
Dynamic reconfiguration

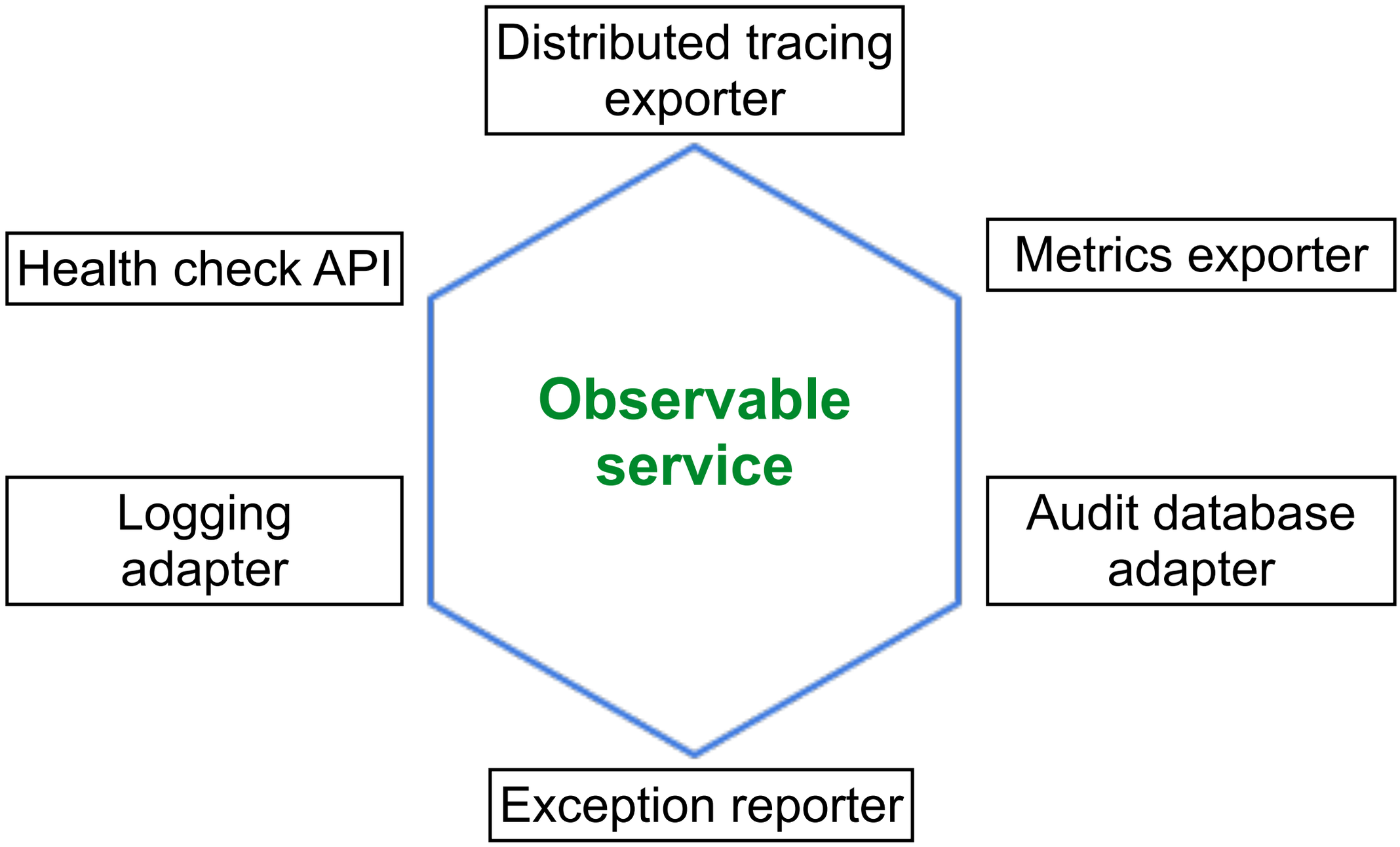
Drawbacks of configuration server

Need setup and maintain !!

Design observable services

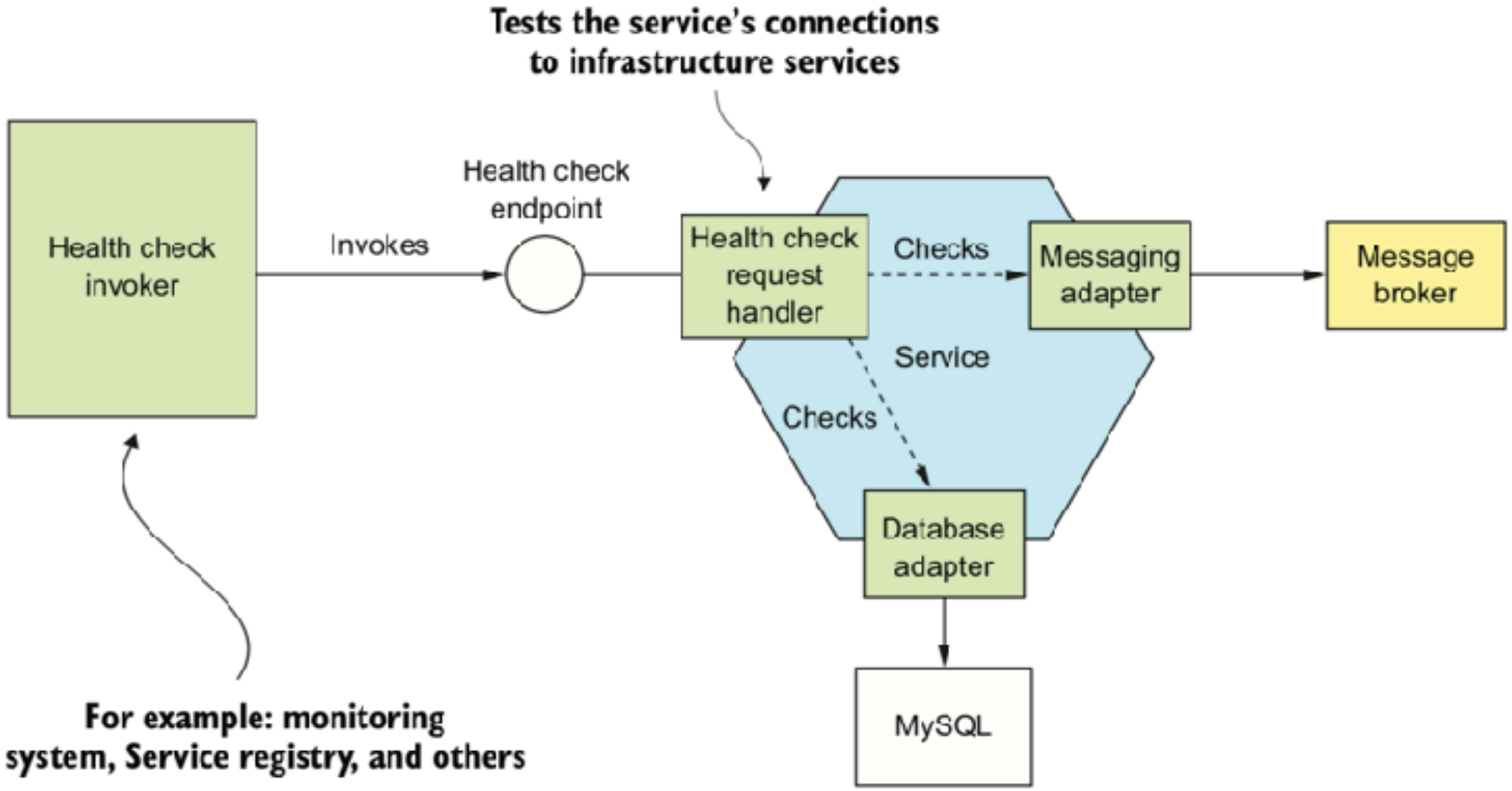
Health check API
Log aggregation
Distributed tracing
Exception tracking
Application metrics
Audit logging

Observable services



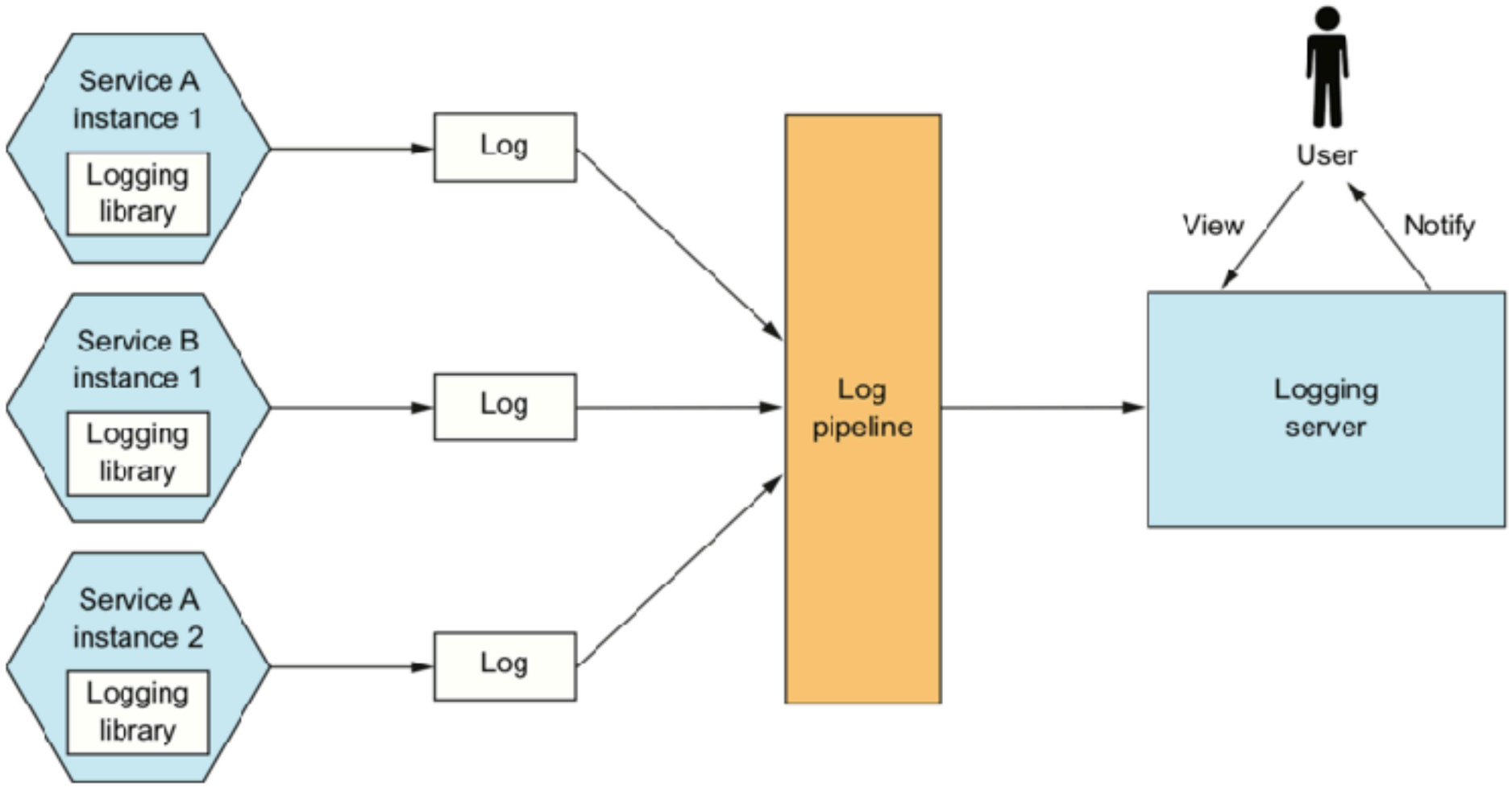
Health check API

Expose an endpoint that return the health of service



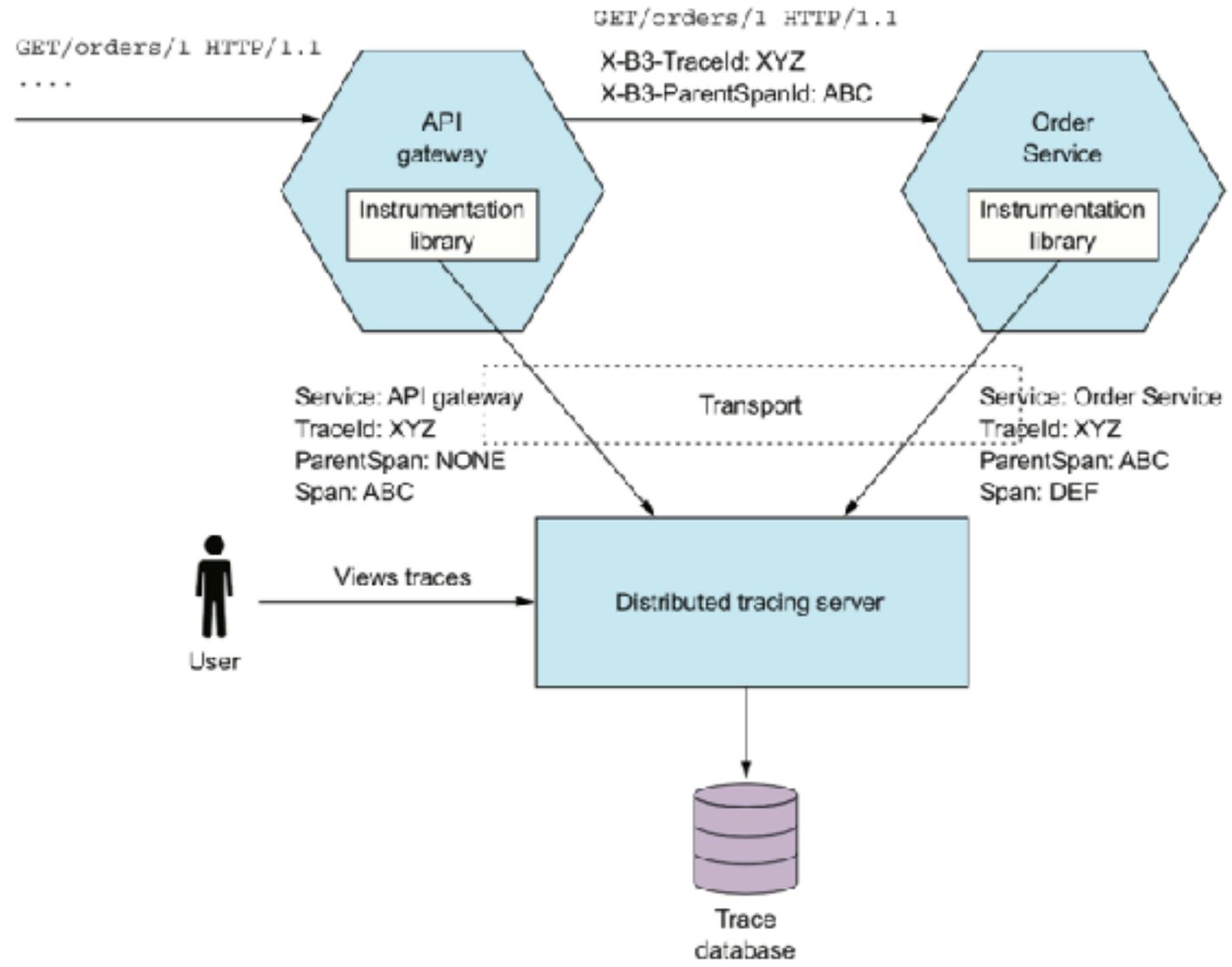
Log aggregation

Log service activity and write logs into a centralized logging server. (searching, alerting)

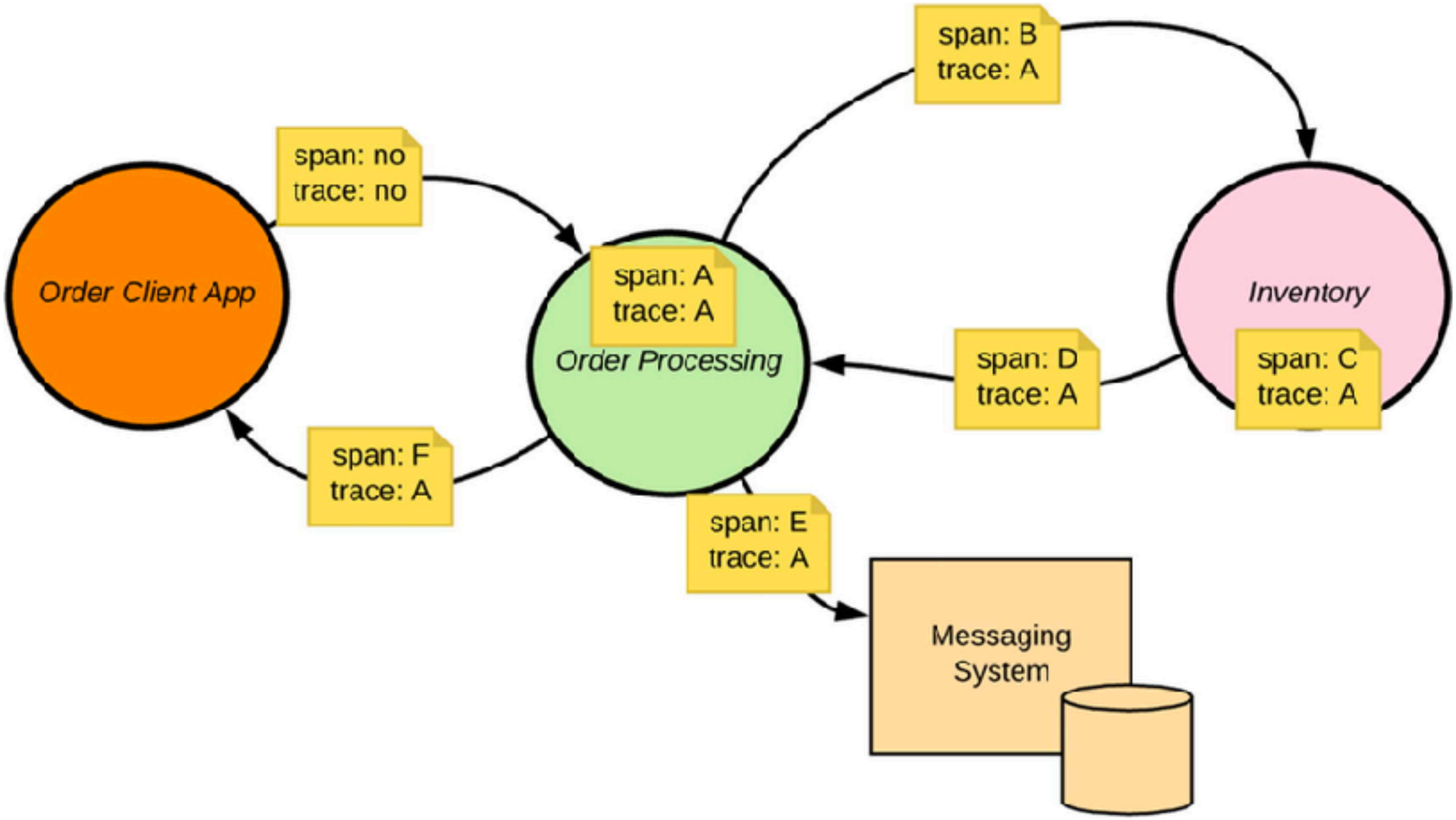


Distributed tracing

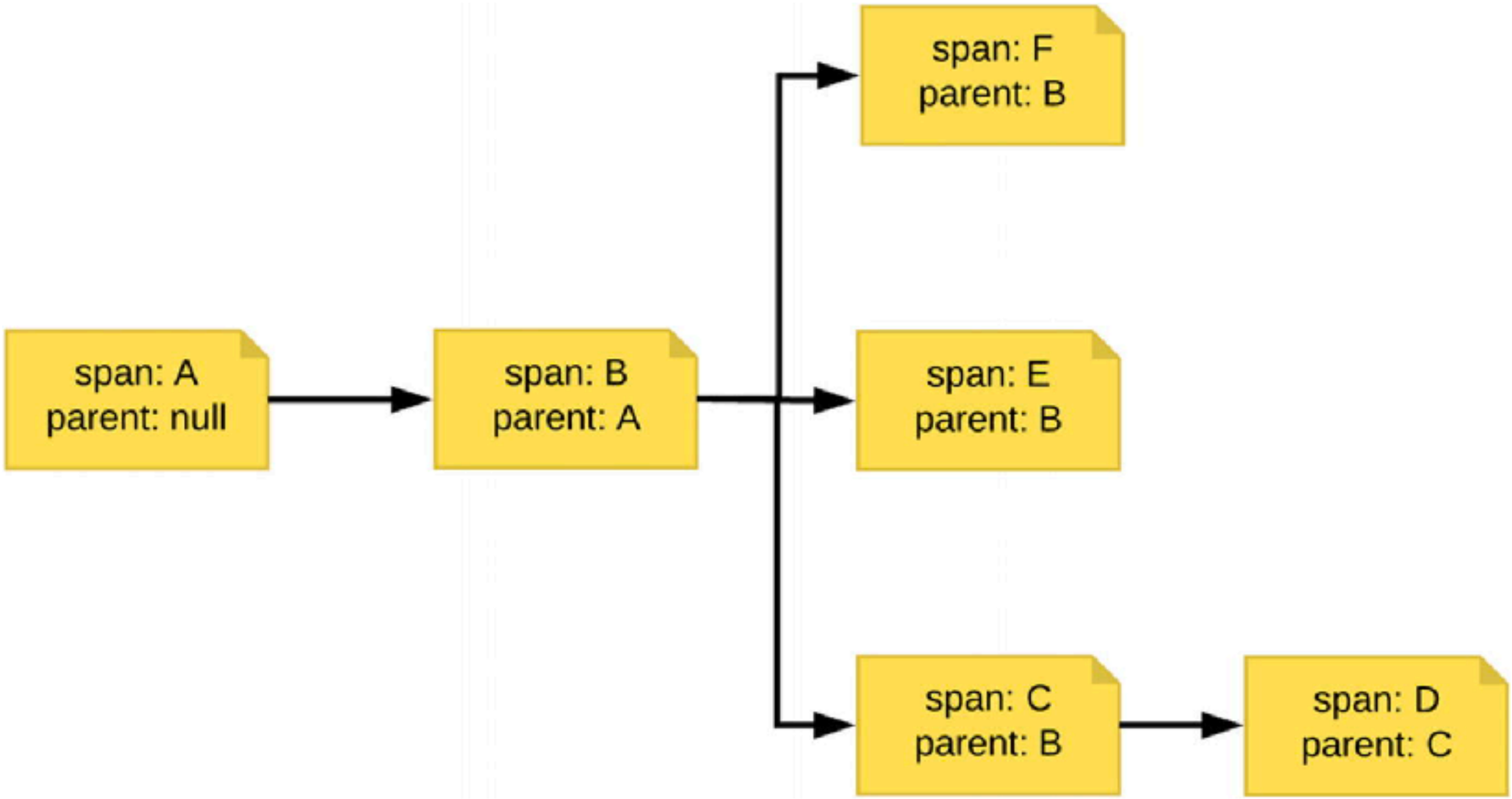
Assign each external request a unique ID and trace requests as flow between services



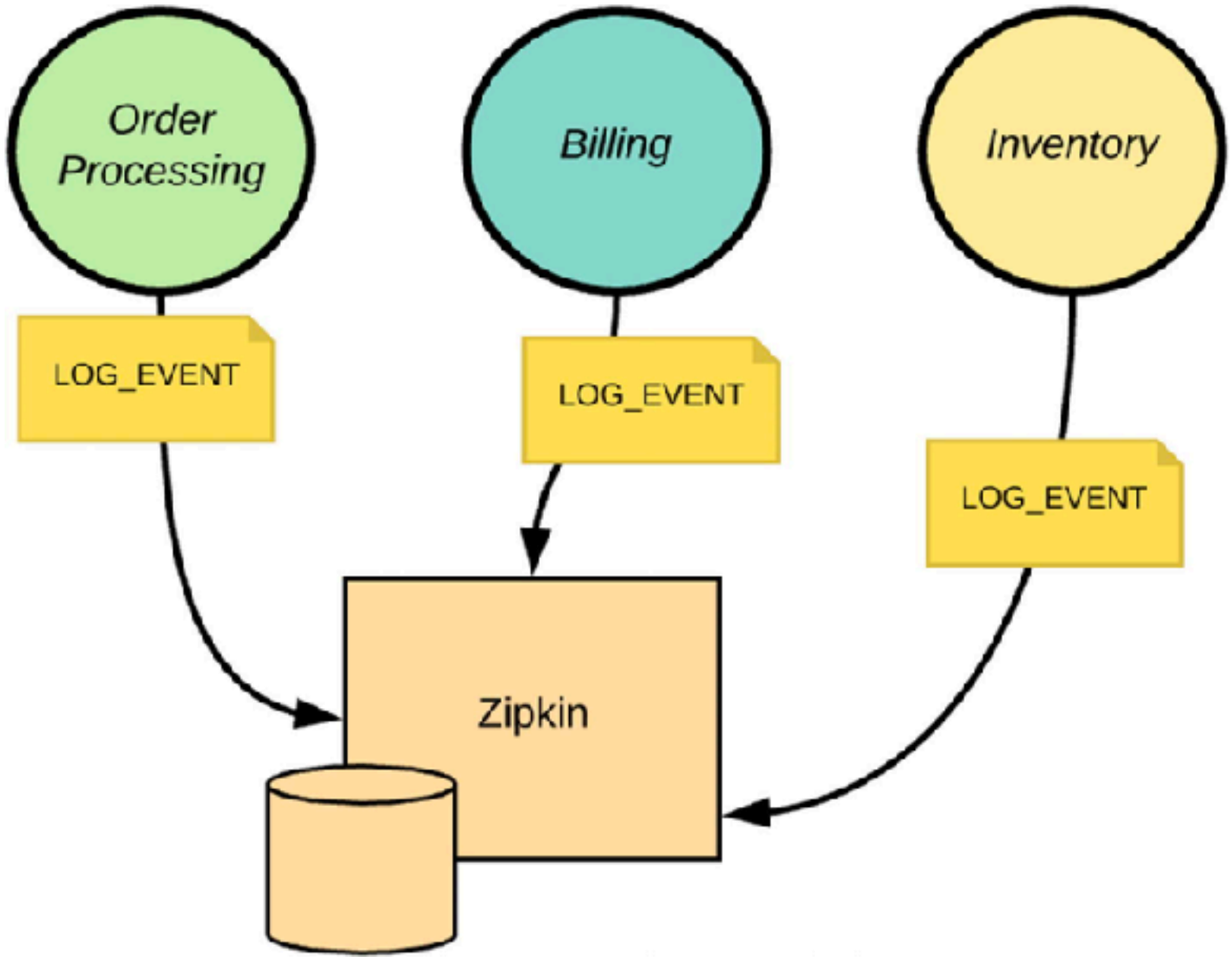
Distributed tracing



Distributed tracing

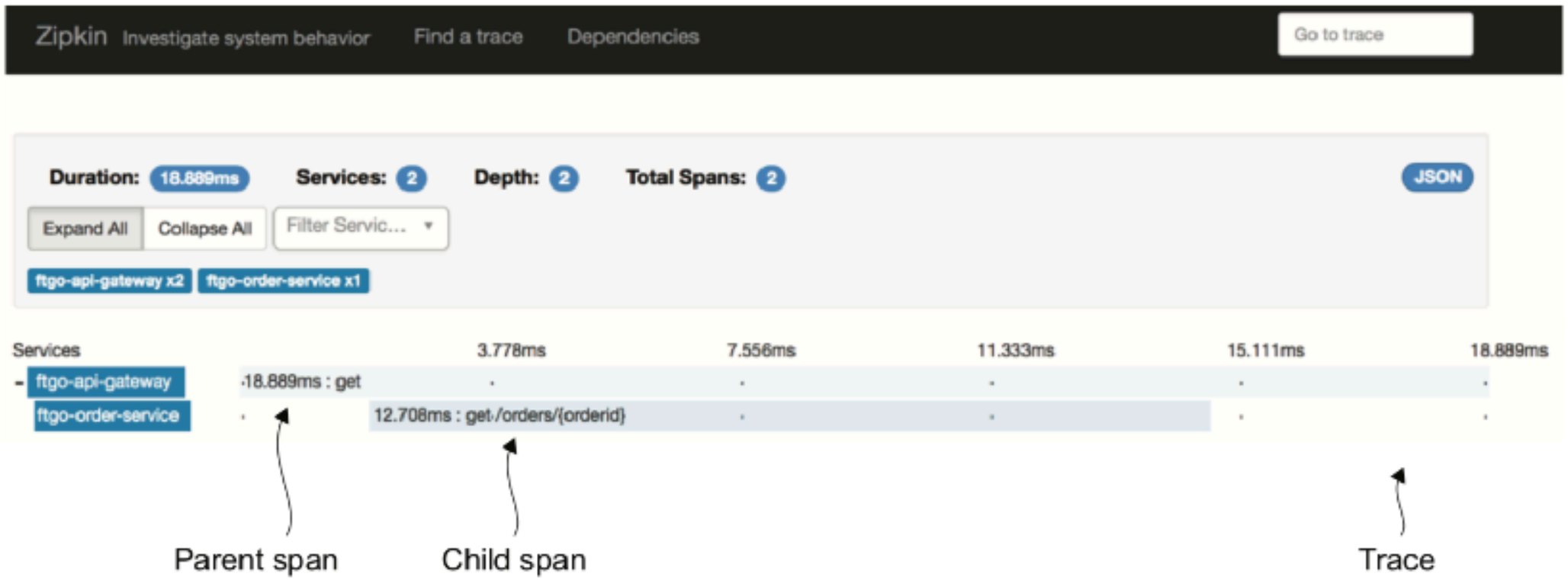


Distributed tracing with Zipkin



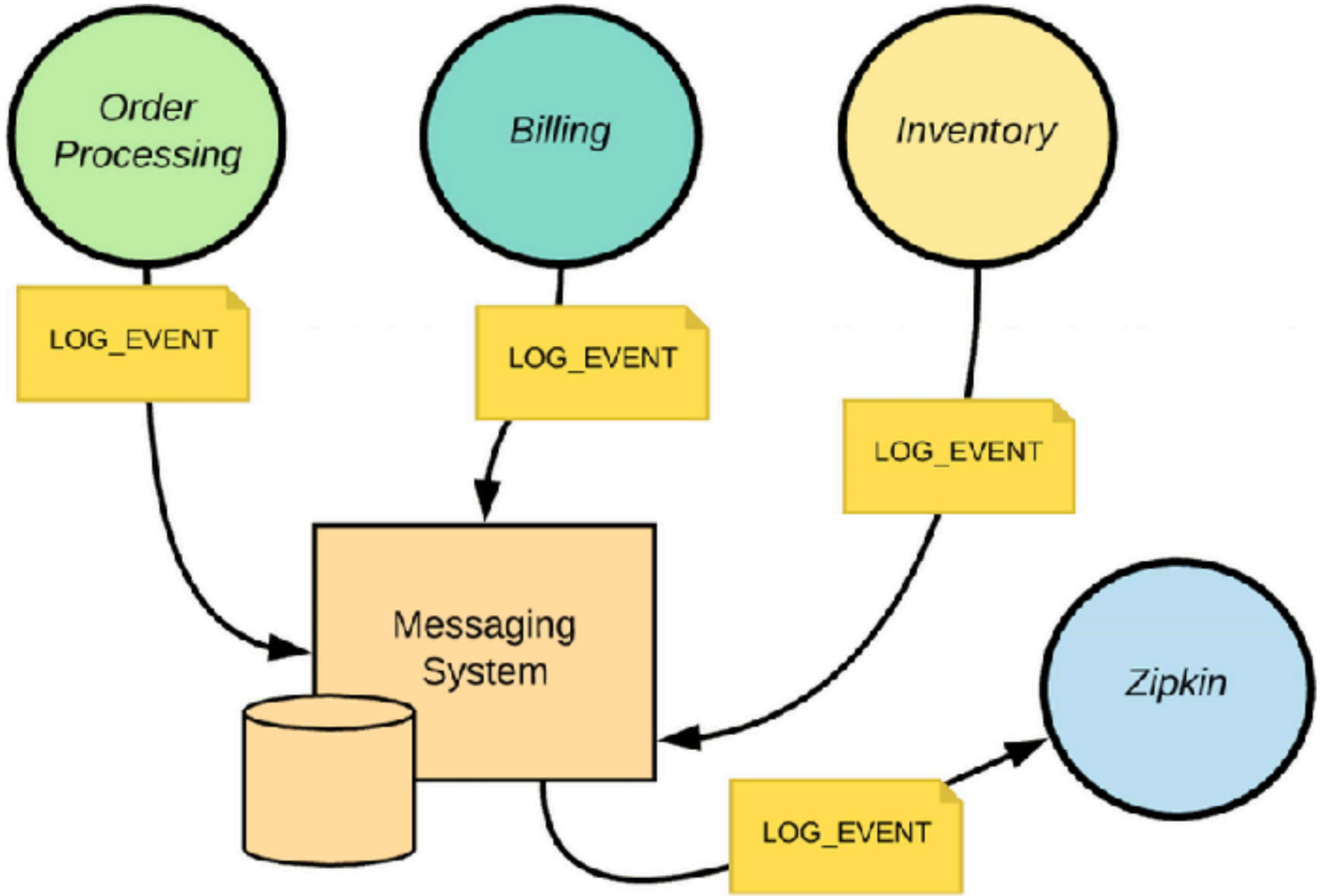
https://zipkin.io/

Distributed tracing with Zipkin



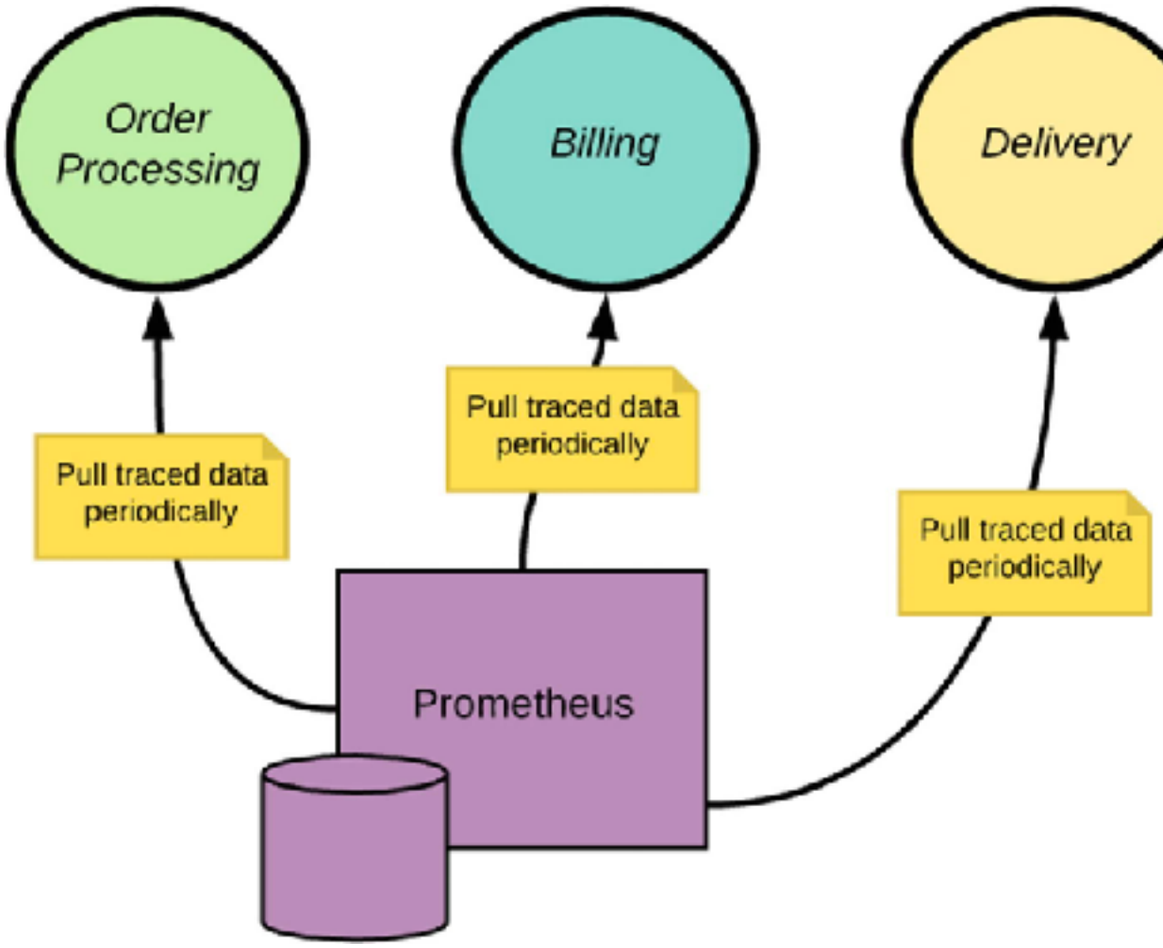
https://zipkin.io/

Distributed tracing with Zipkin



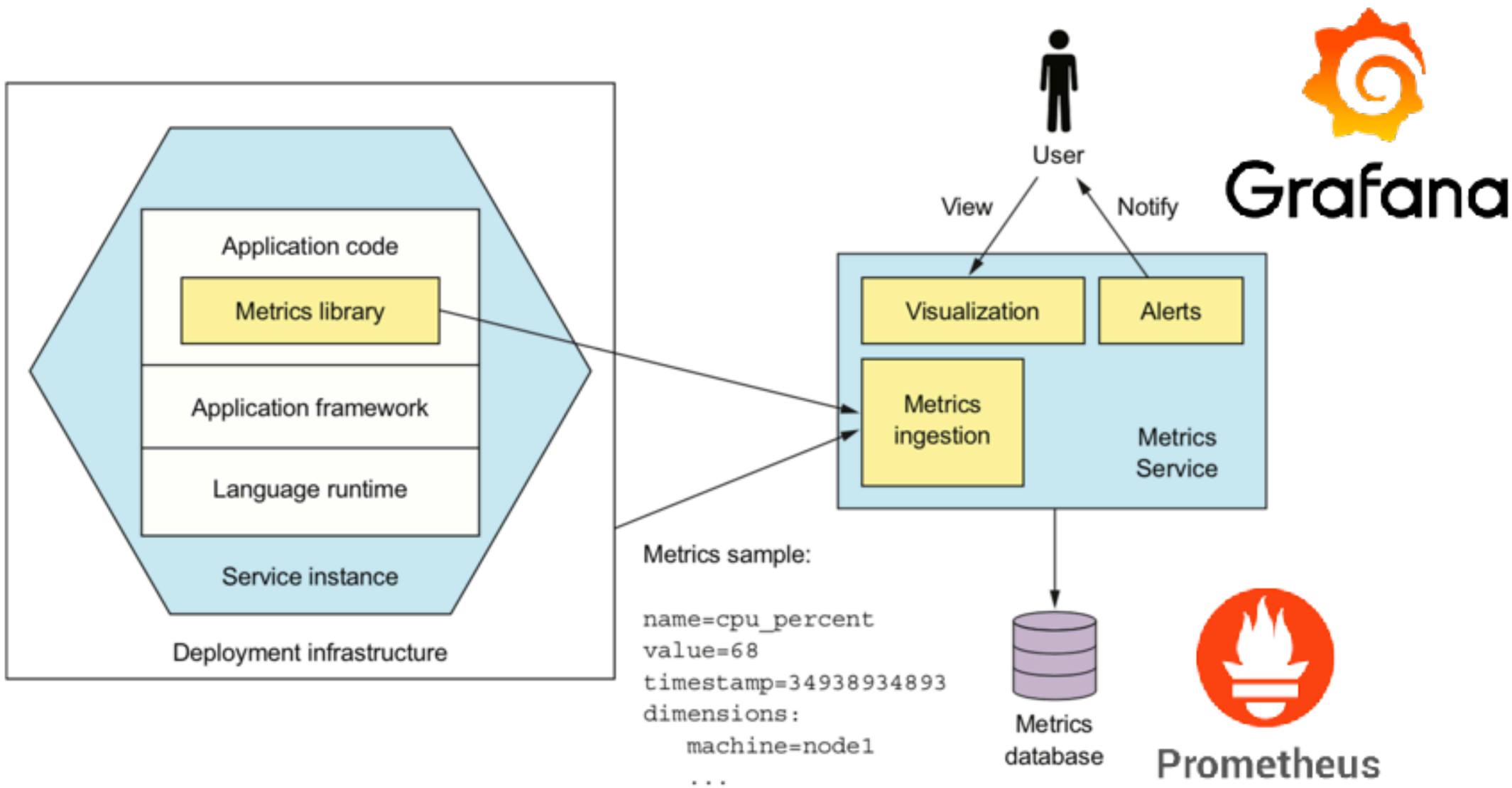
Application metrics

Services maintain metrics and expose to metric server (counters, gauges)



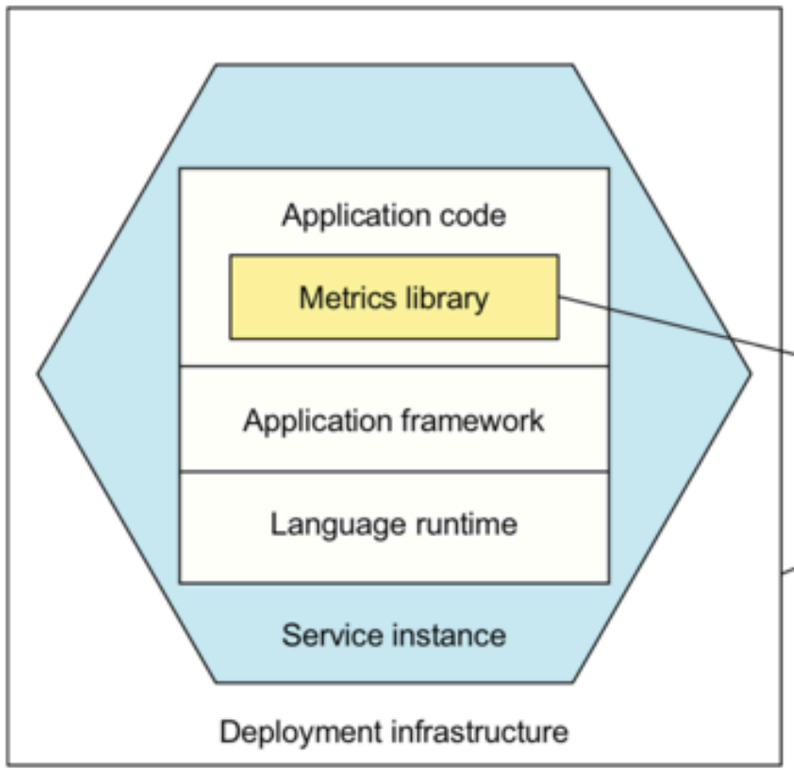
Application metrics

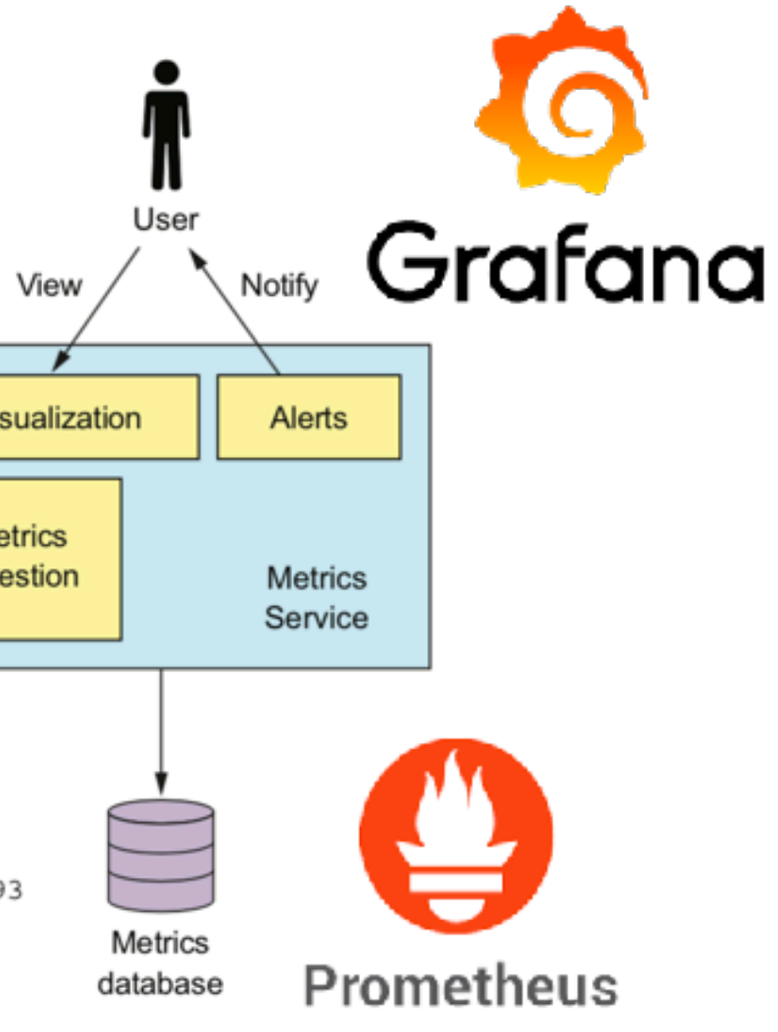
Services maintain metrics and expose to metric server (counters, gauges)



Application metrics

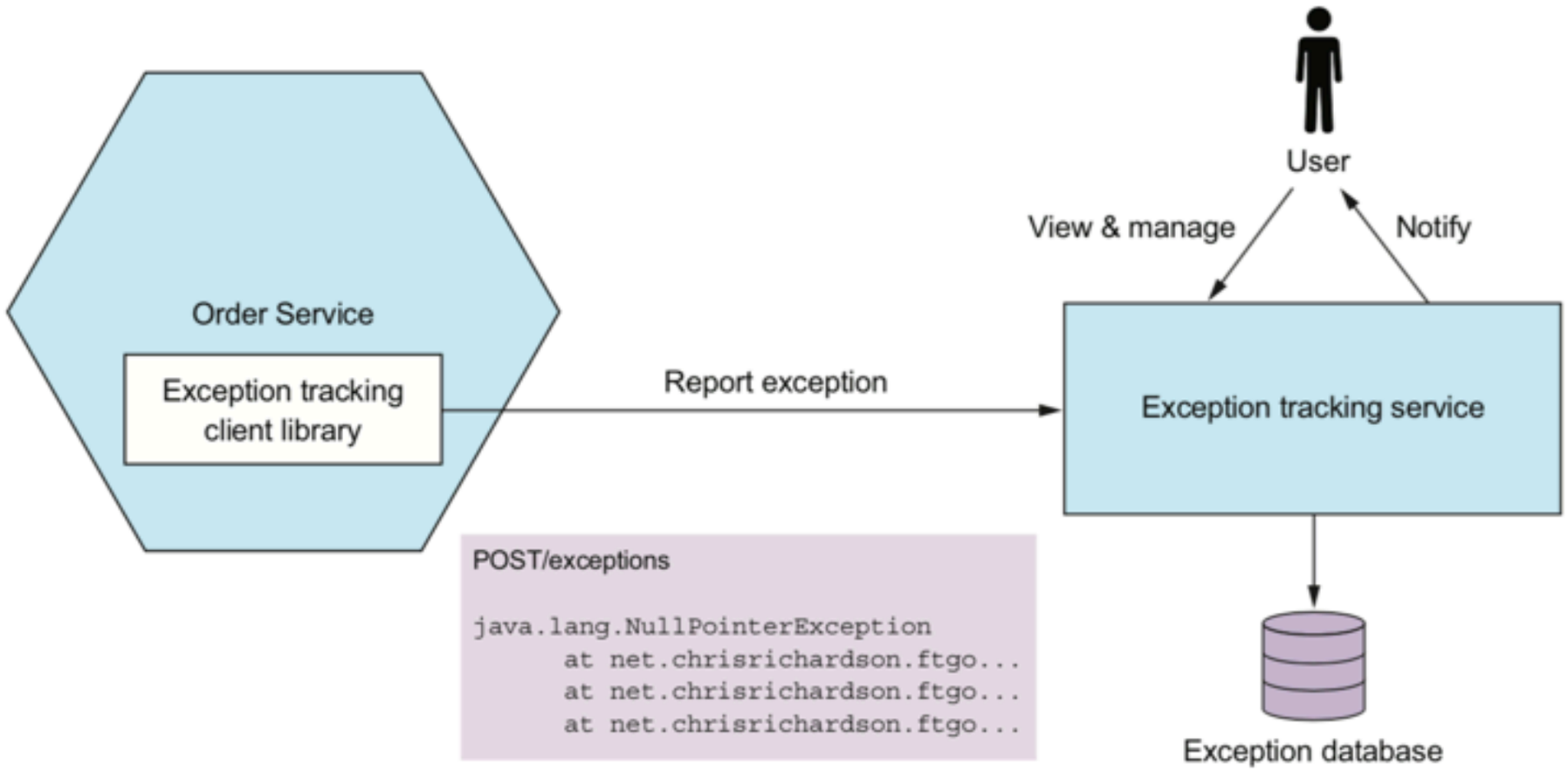
Metrics database => Prometheus
Visualization, Alert => Grafana





Exception tracking

Report exceptions to exception tracking service
Help to identify the root cause



Exception tracking services

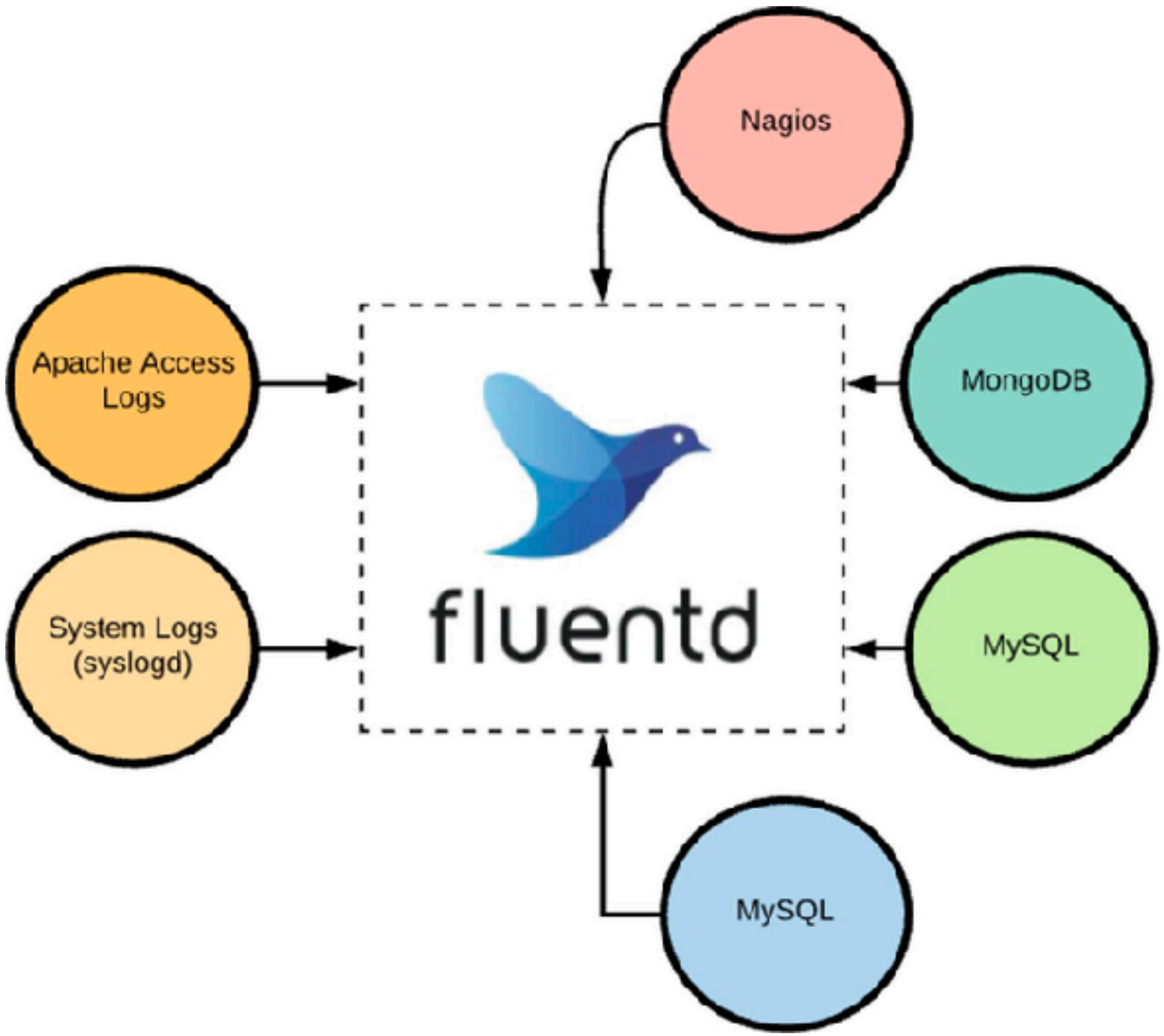
Audit logging

Log of user actions
Help customer support
Ensure compliance
Detect suspicious behaviour

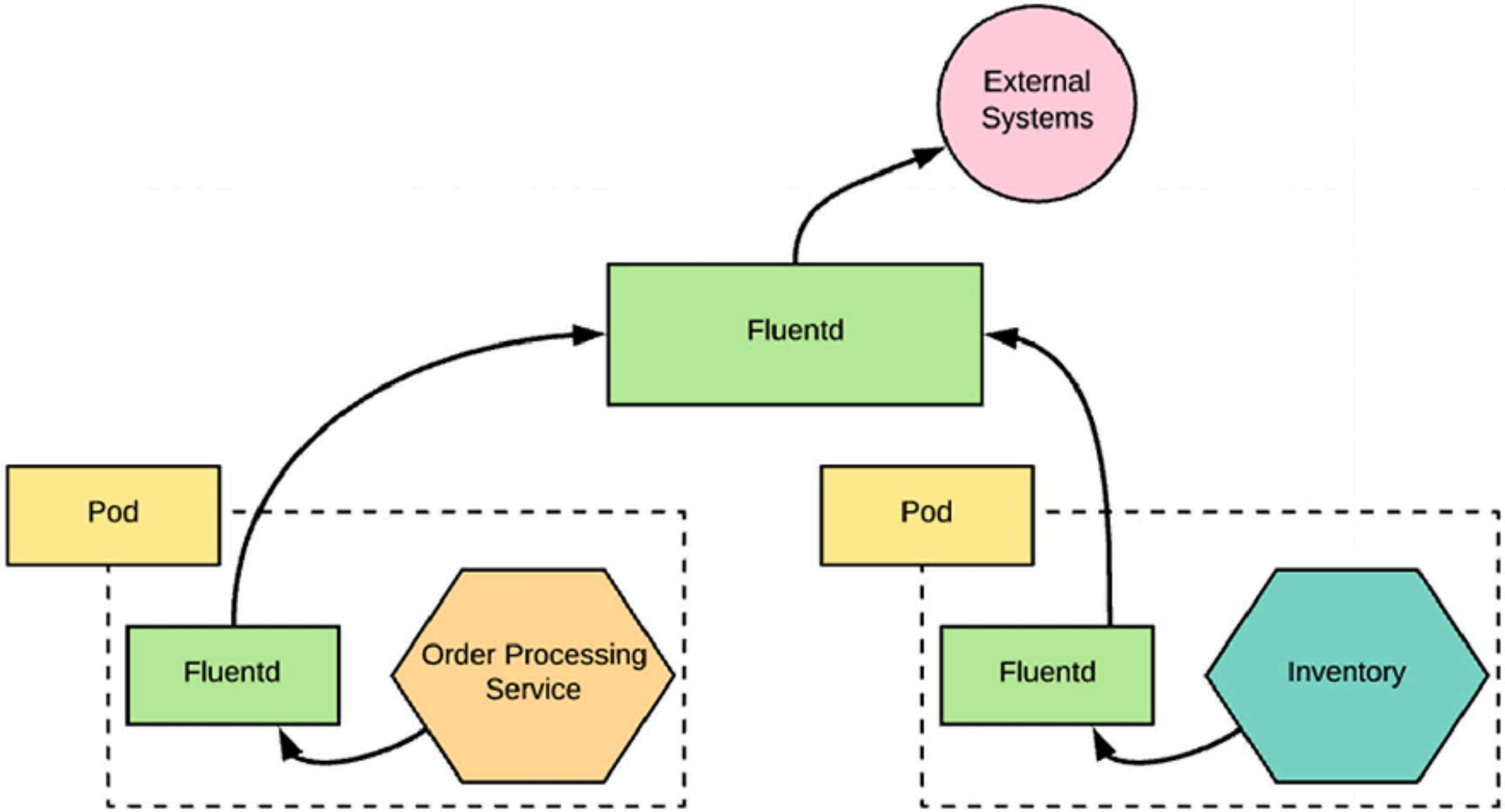
How to implement the audit logging ?

Add logging code in business logic
Use AOP (Aspect-Oriented Programming)
Use event sourcing

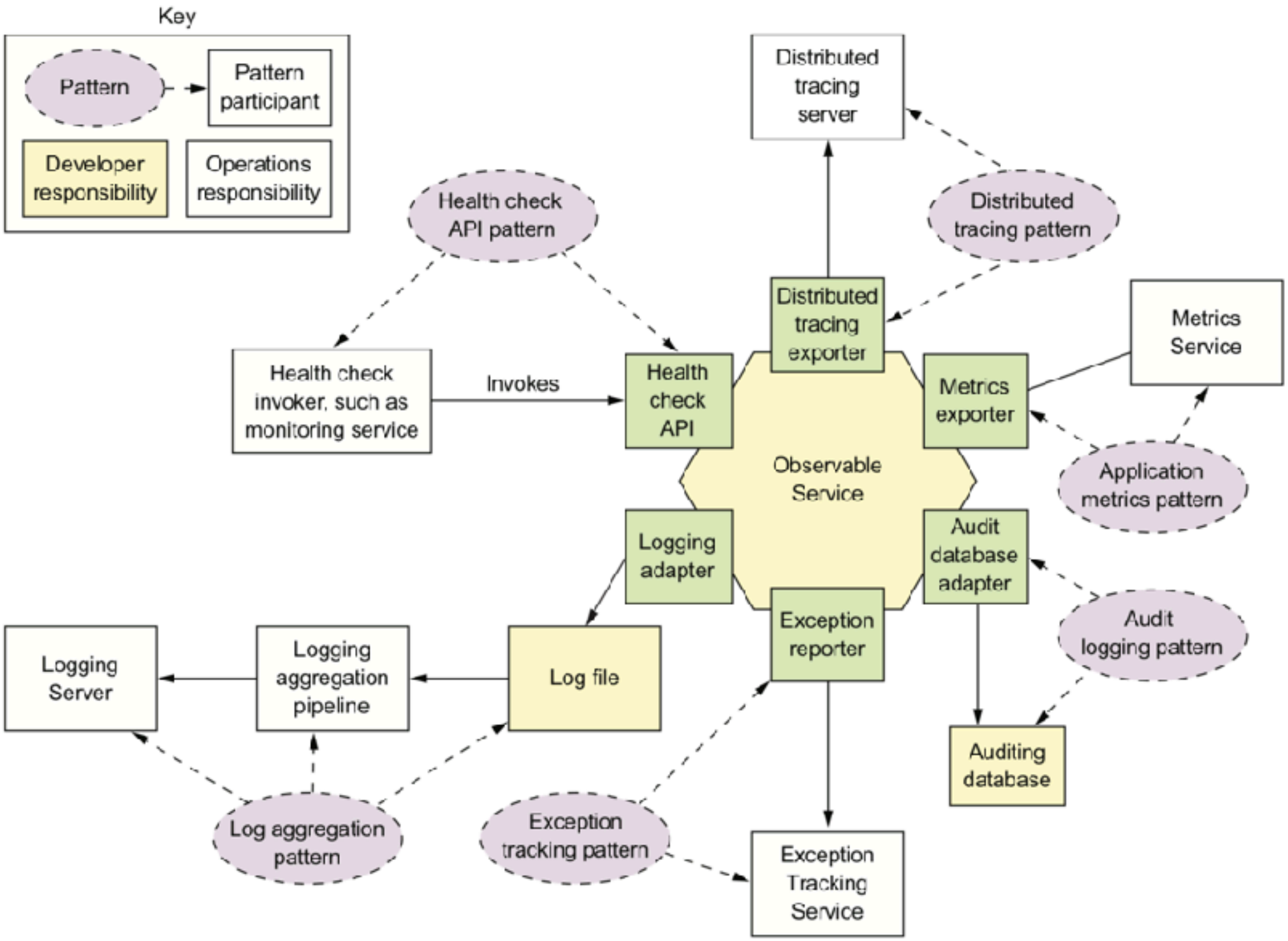
Centralize logging



Centralize logging



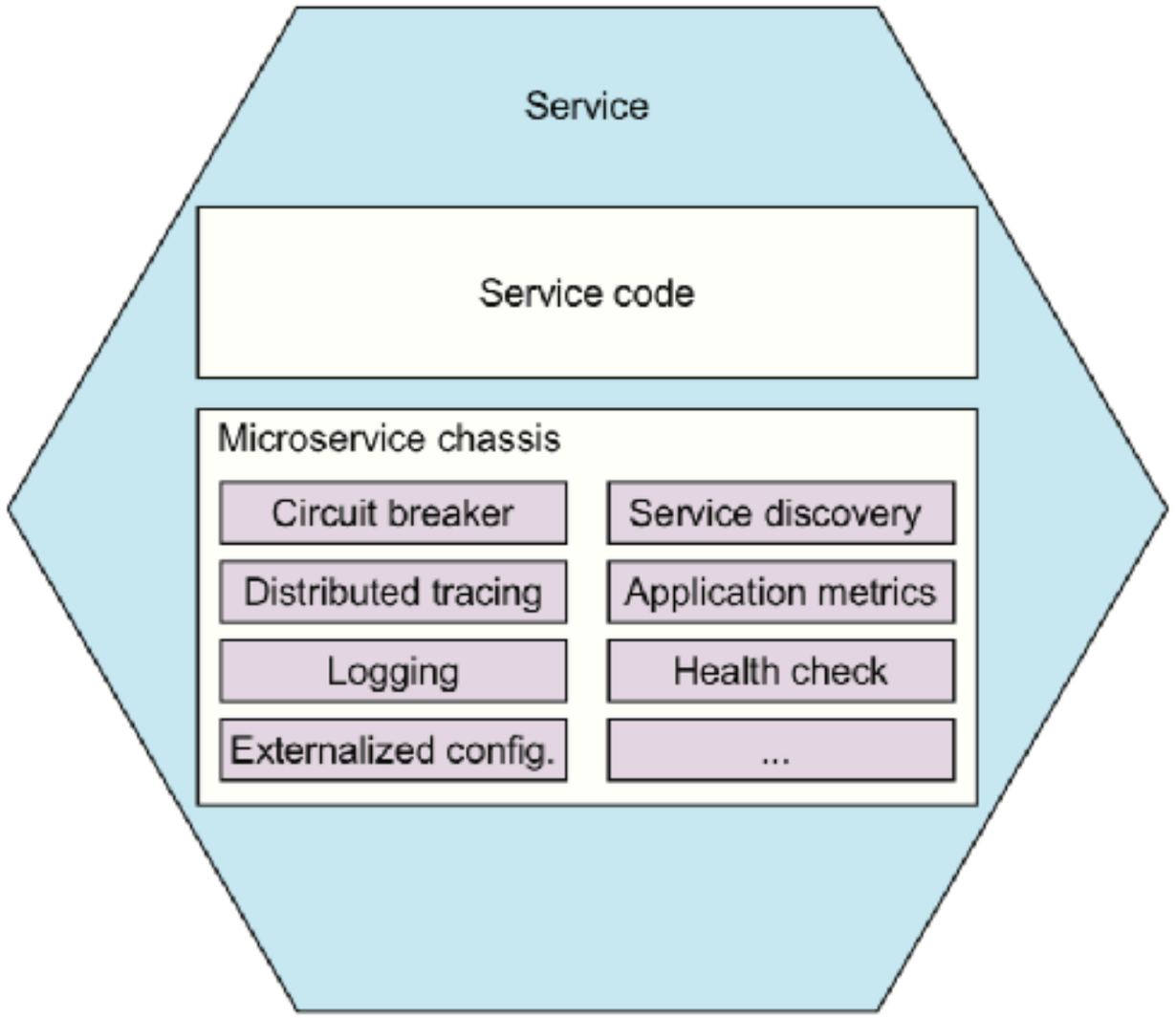
Observable services



Workshop

Microservice framework

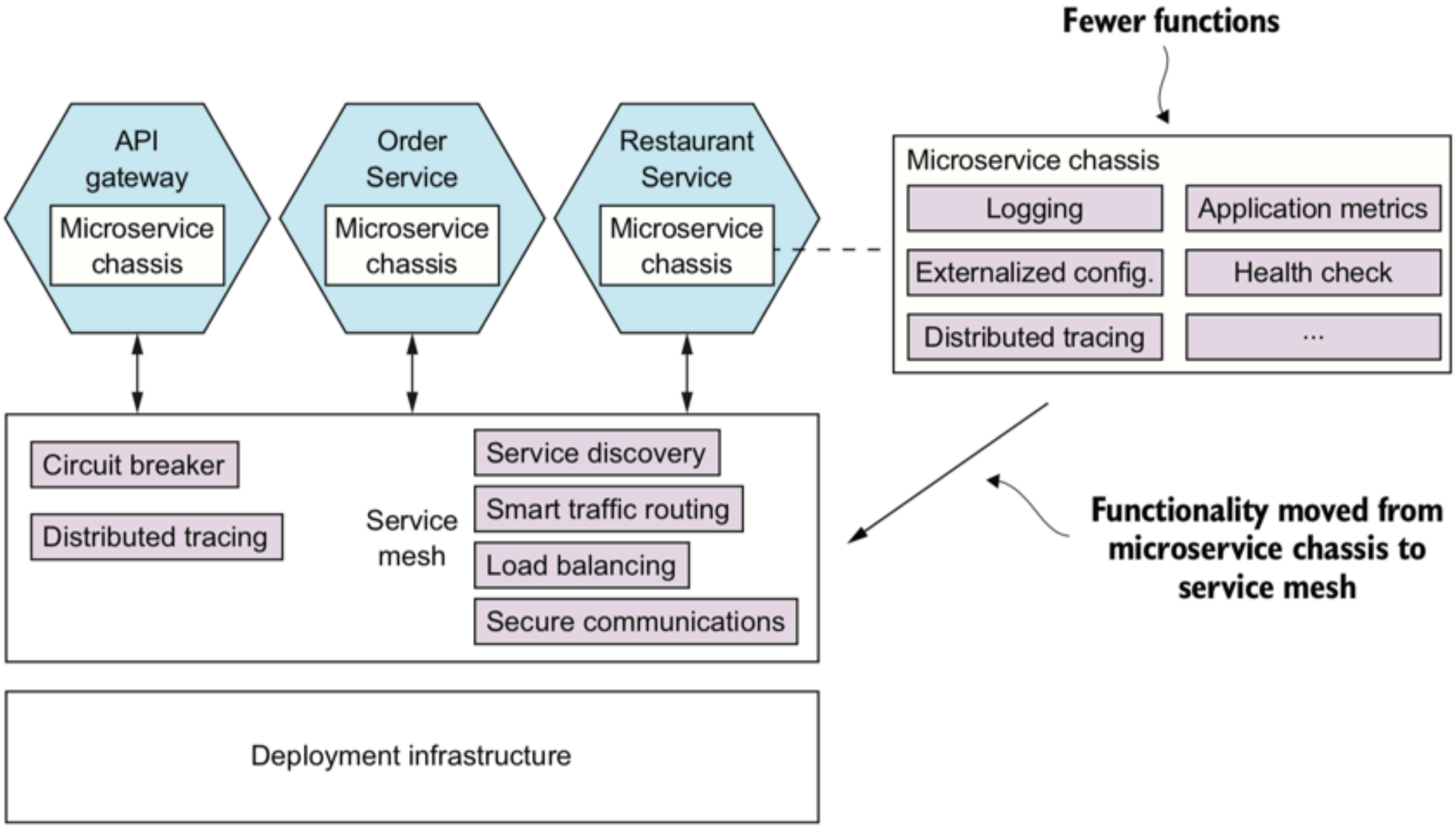
Microservice framework



Microservice framework

Programming language	Framework
Java	Spring boot Spring cloud
Golang	Go-kit Micro

Service Mesh



Service Mesh

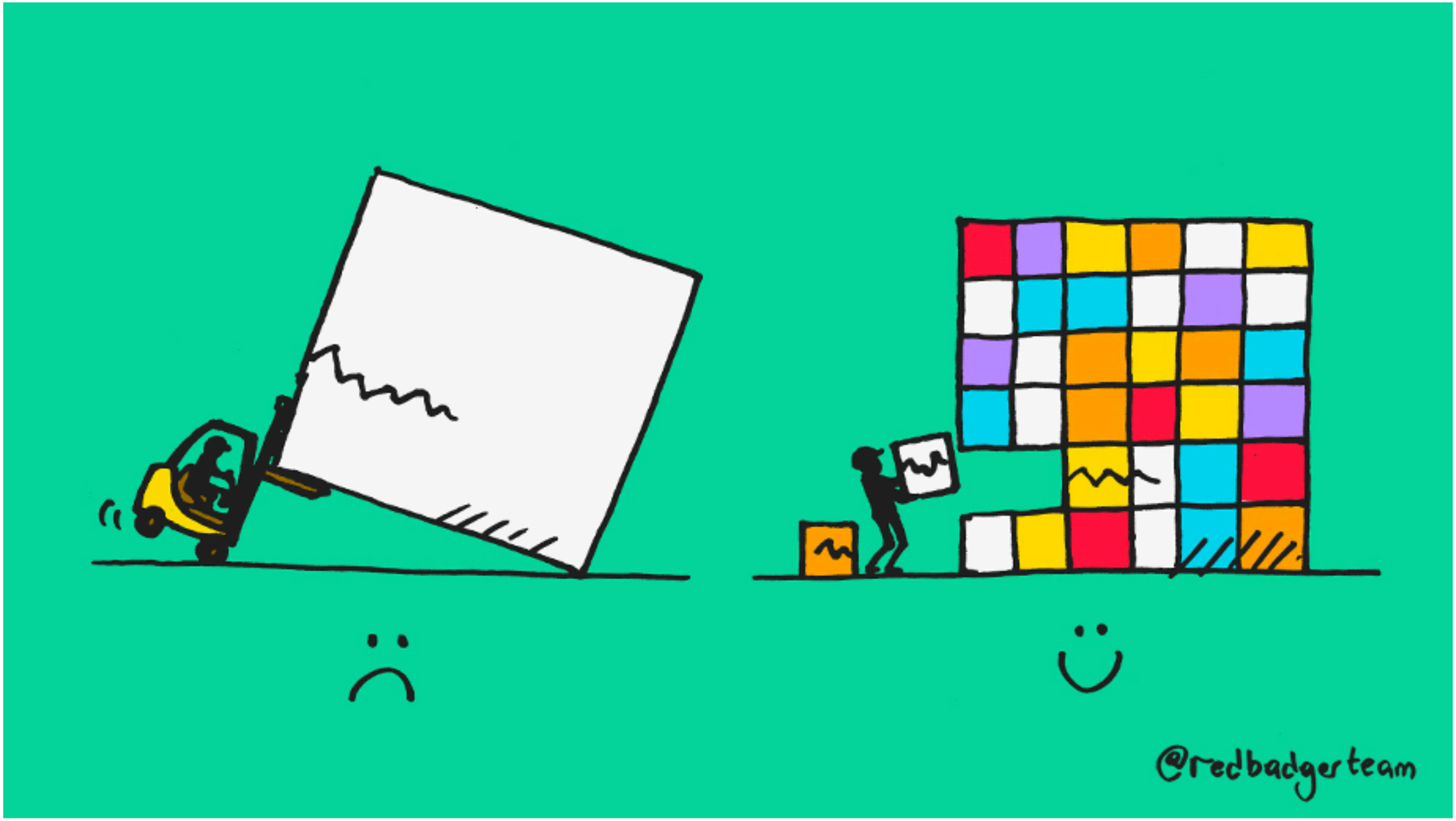
Istio

Connect, secure, control, and observe services.

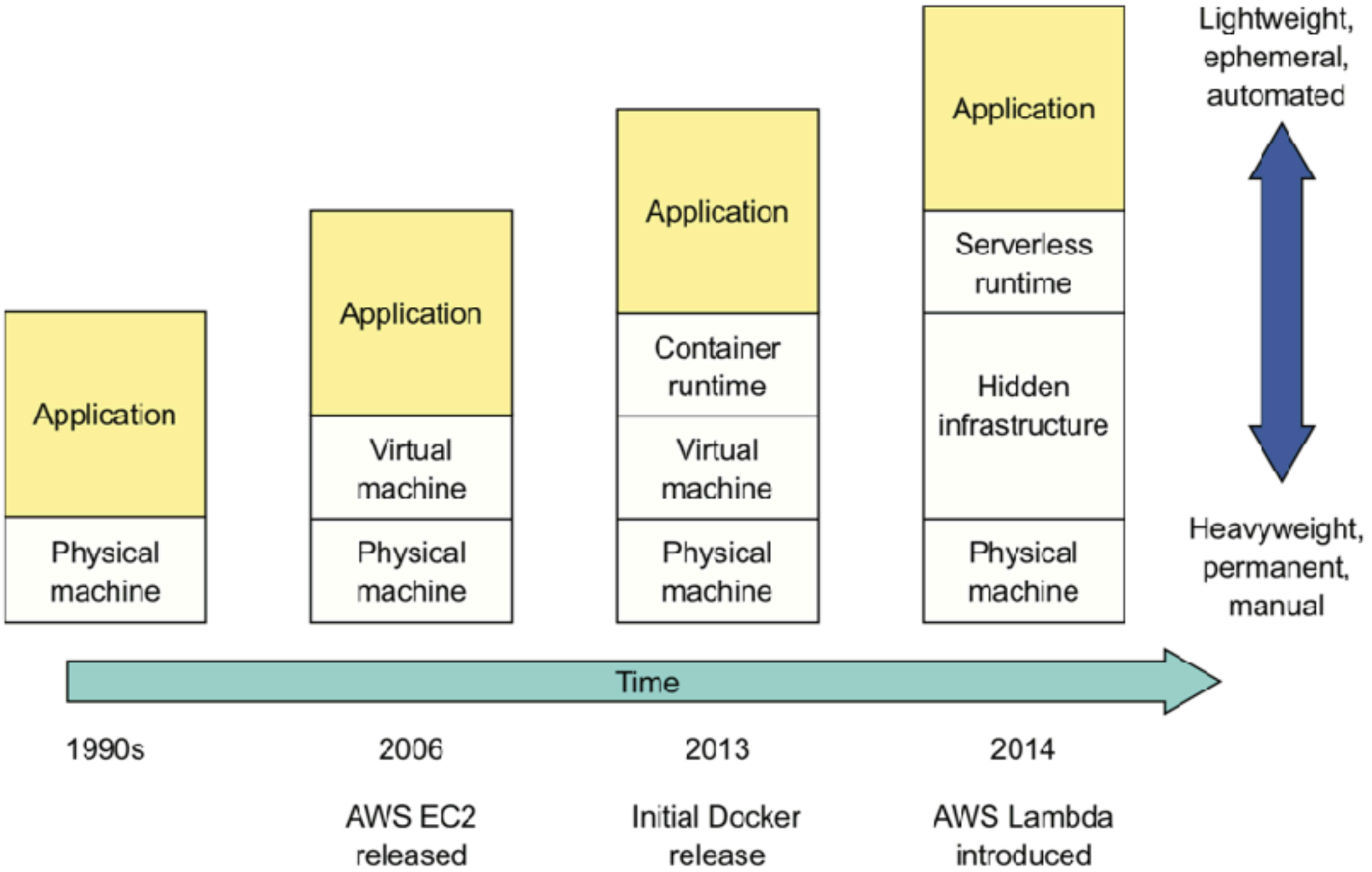
CONDUIT

linkerd

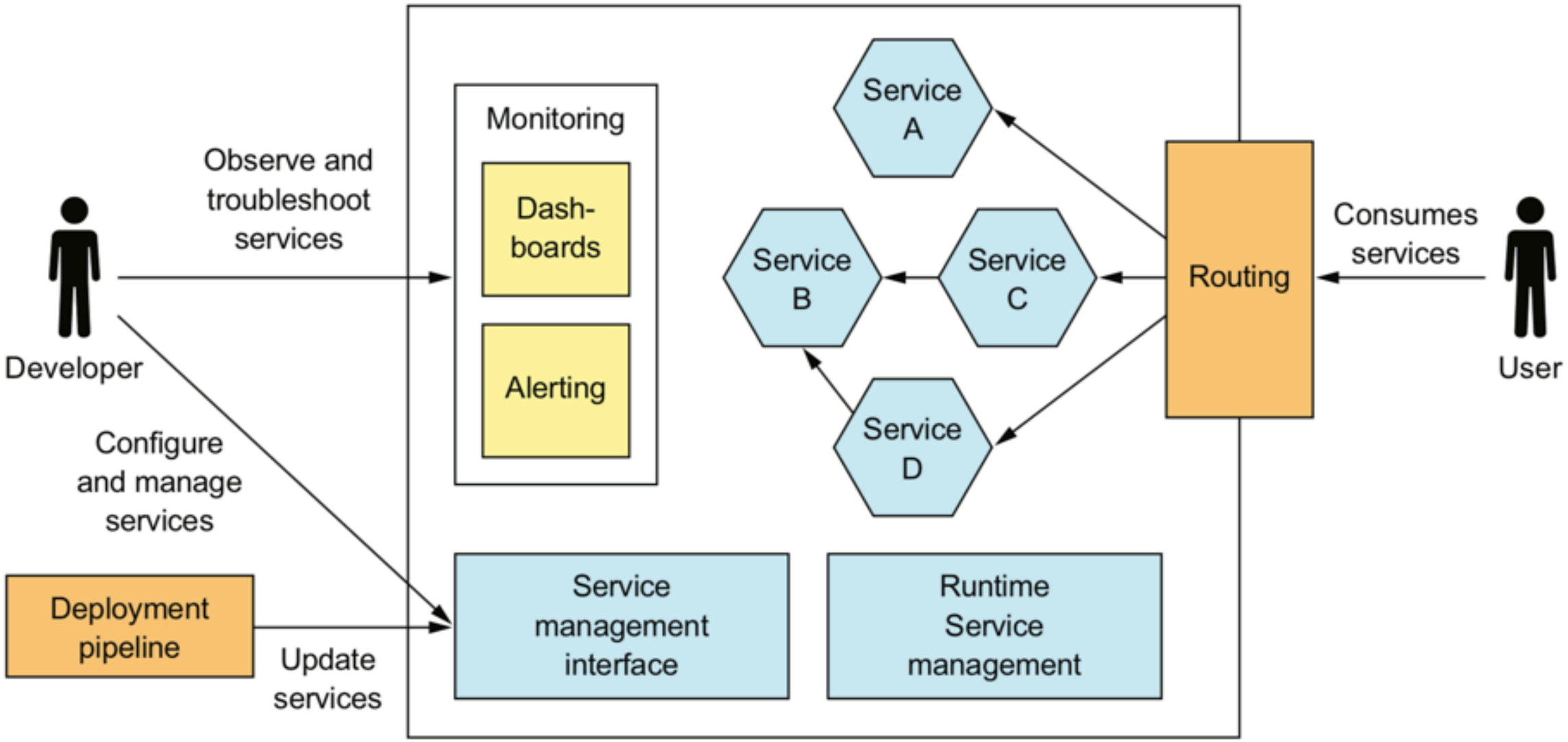
Module 3 : Deploy



Infrastructure



View of production environment



Deploy Microservices

Language-specific packaging

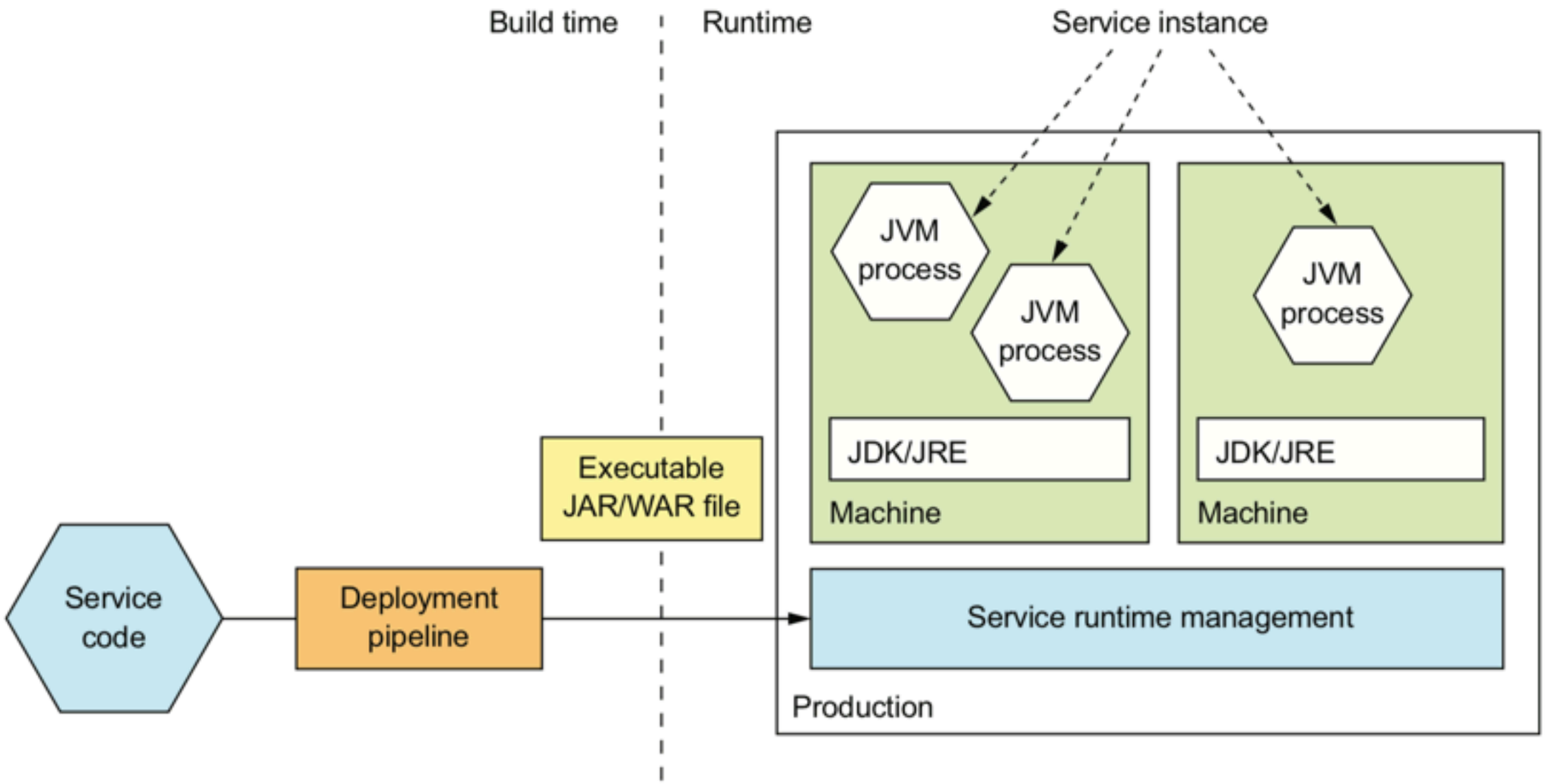
Virtual Machine (VM)

Container

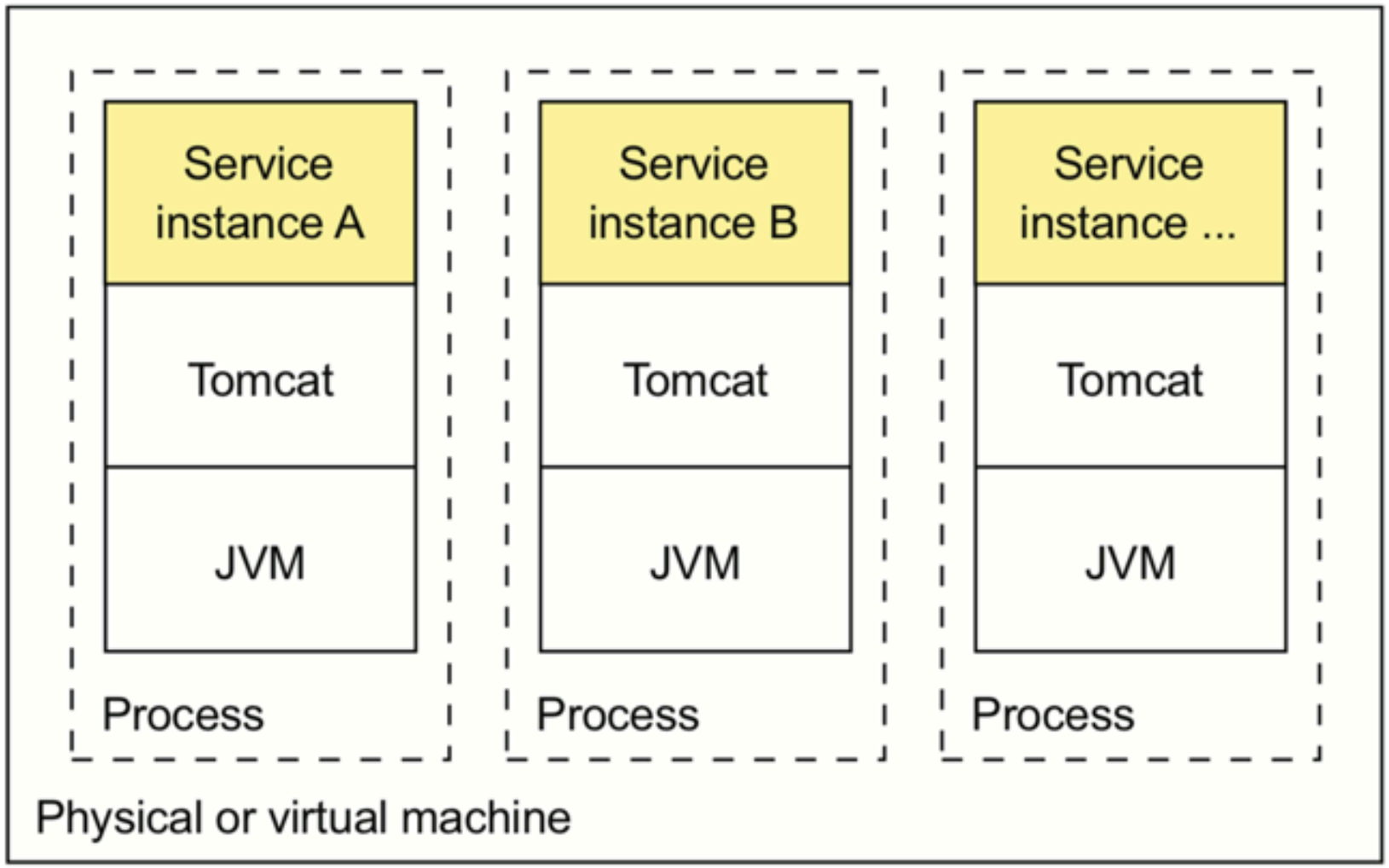
Kubernetes

Serverless/FaaS

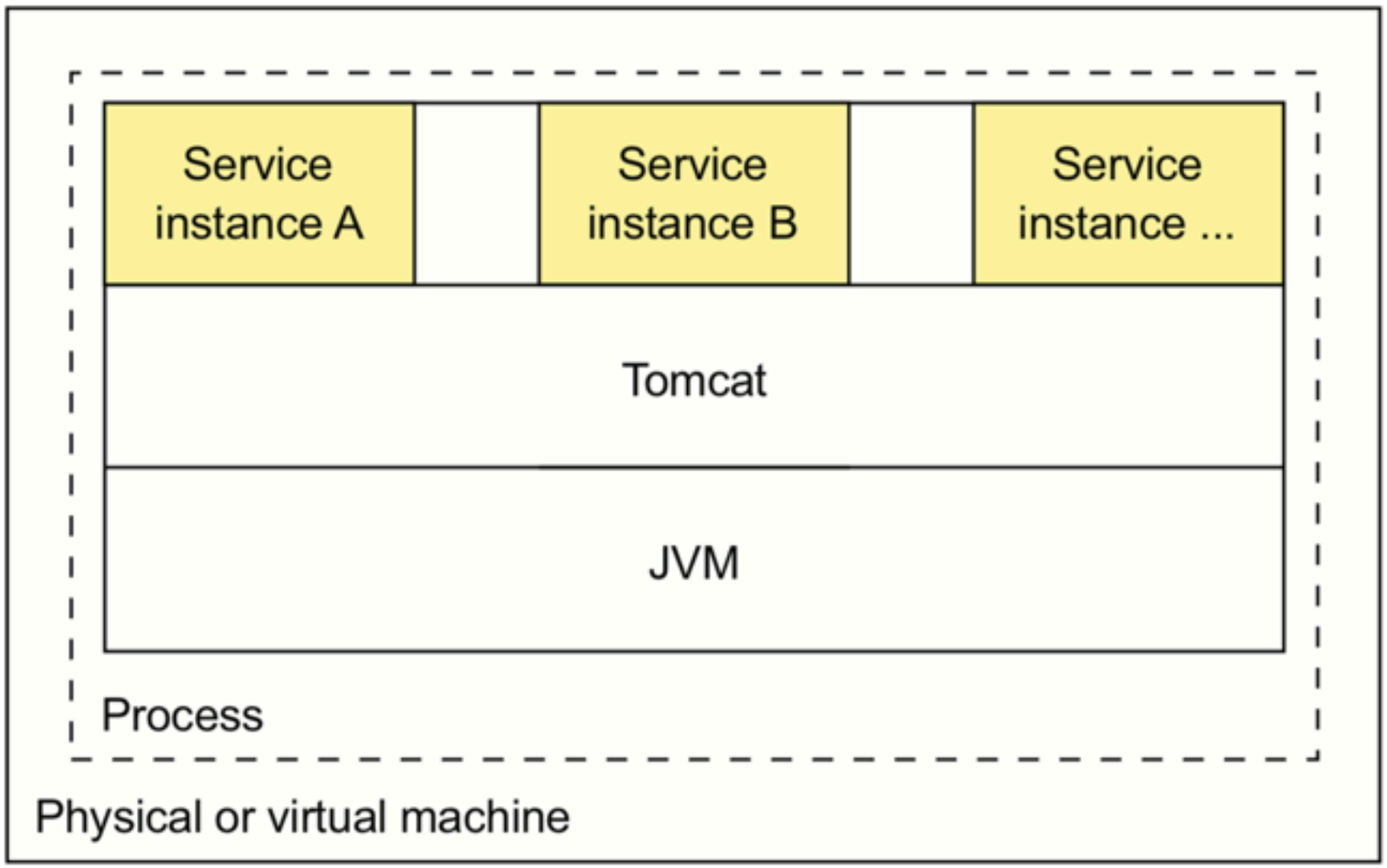
1. Language-specific packaging



Multiple services on same machine



Multiple services on same process



Benefits

Fast deployment

Efficient resource utilization

Service instances’s resources are constrained

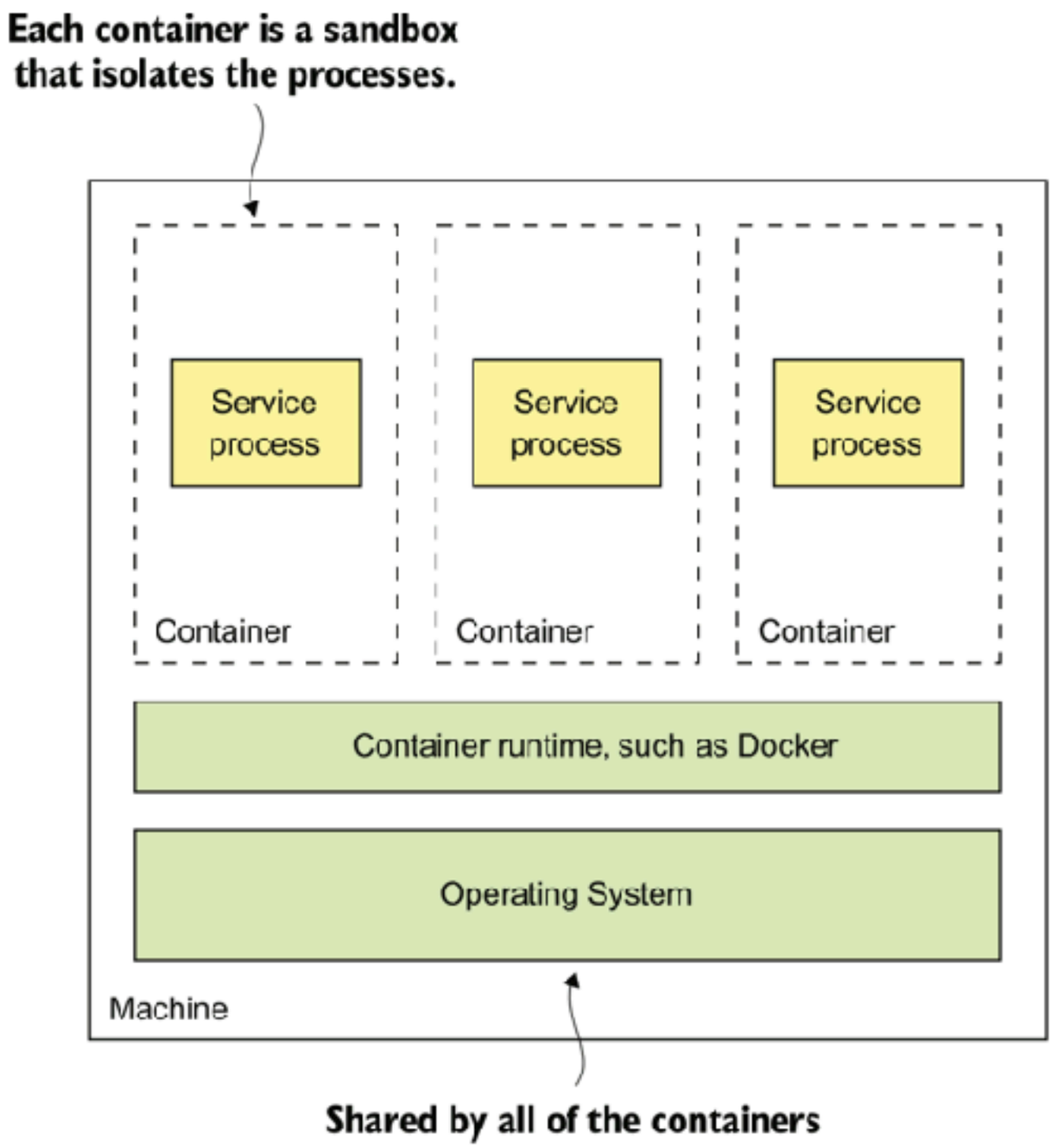
Drawbacks

Lack of encapsulation of technology stack

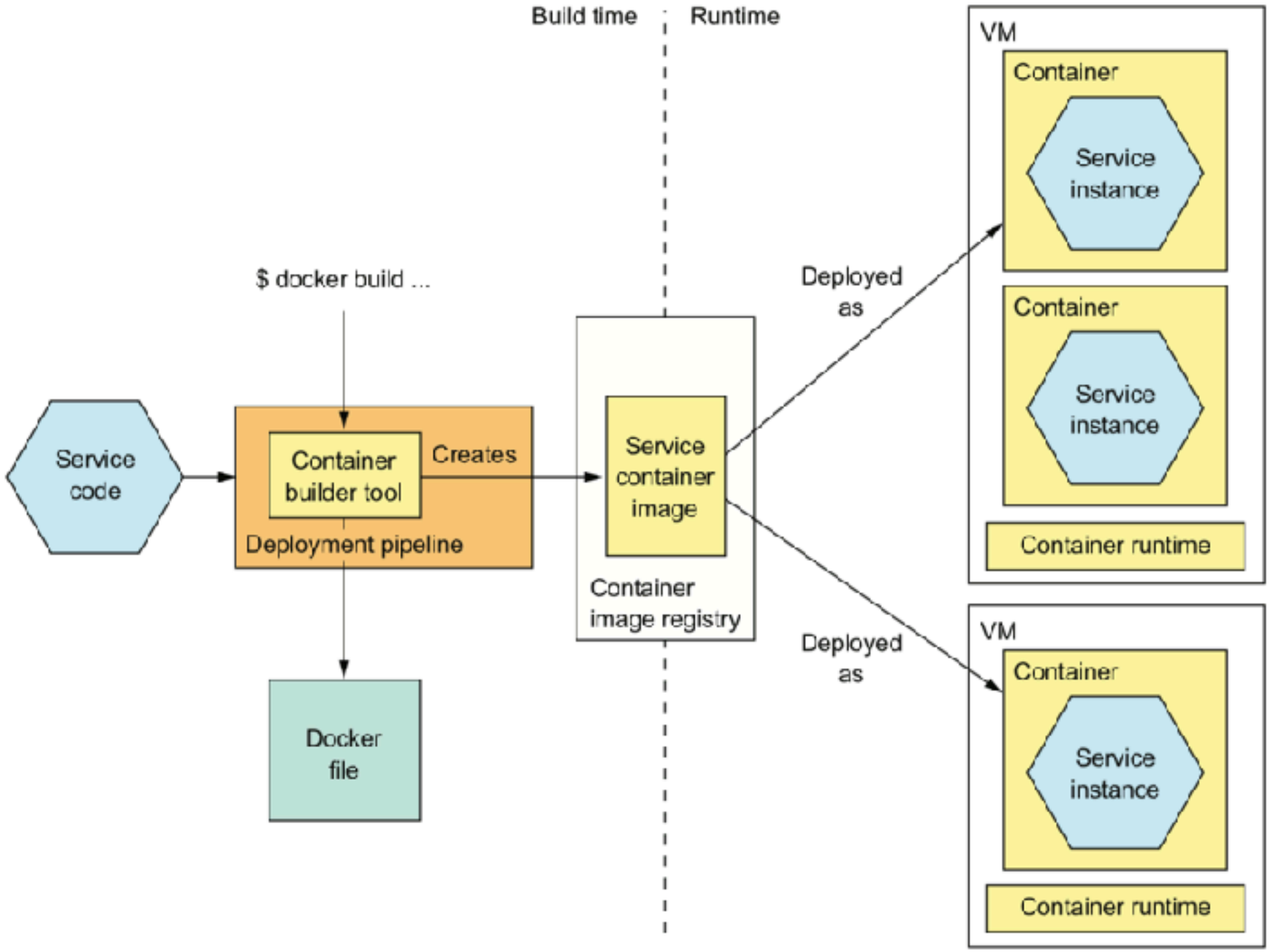
No ability to constrain resources of service

Lack of isolation

2. Working with container



Deployment with container



Deploy services with Docker

Build a docker image
Push docker image to a registry
Run docker container
Working with docker-compose

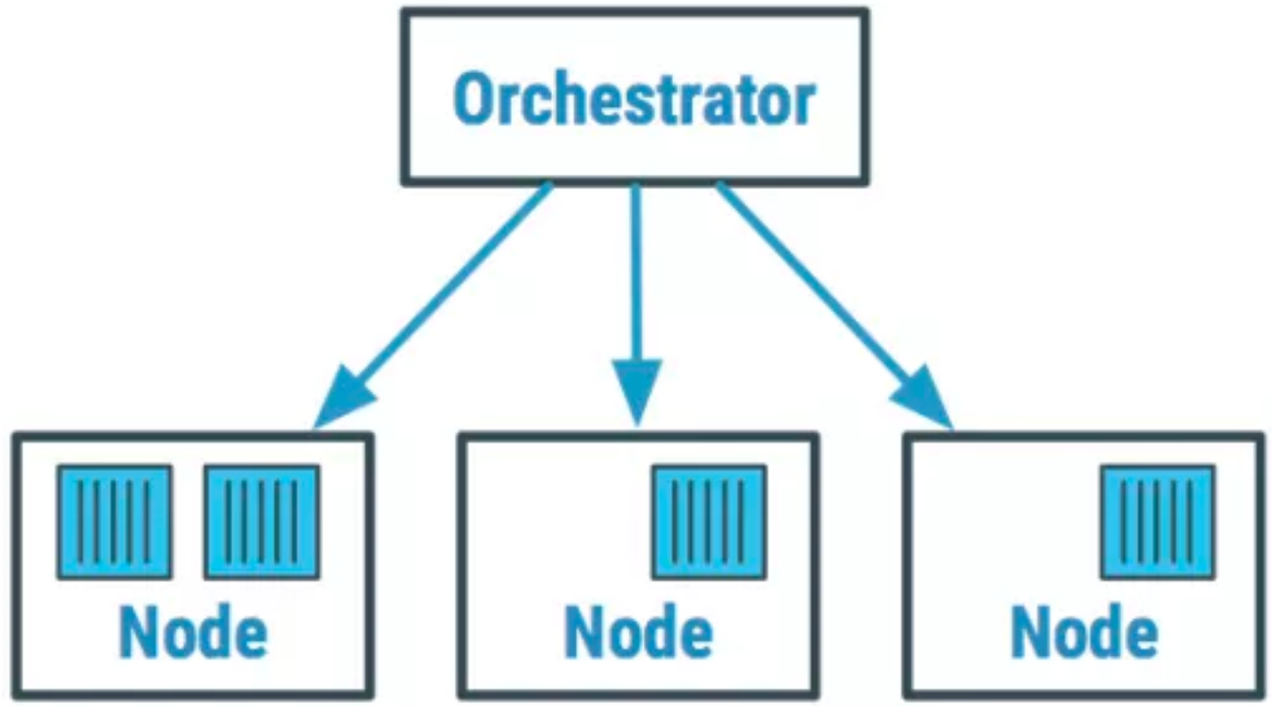
Benefits

Encapsulate technology stack

Service instances are isolated

Service instances’s resources are constrained

Container Orchestration ?



Orchestration tools

Configuration Management

CI/CD orchestration

Container orchestration

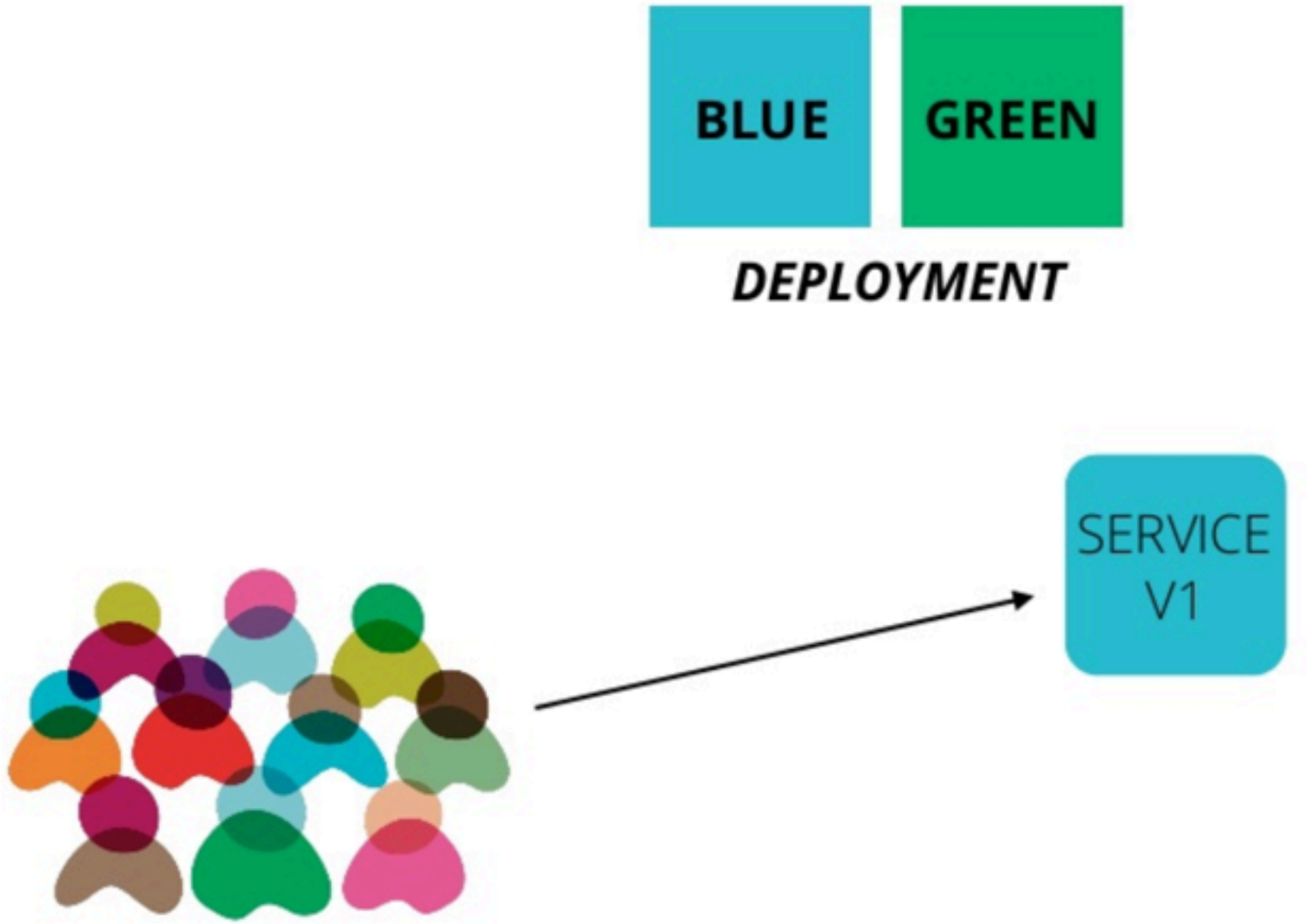
Cloud-specific orchestration

PaaS orchestration

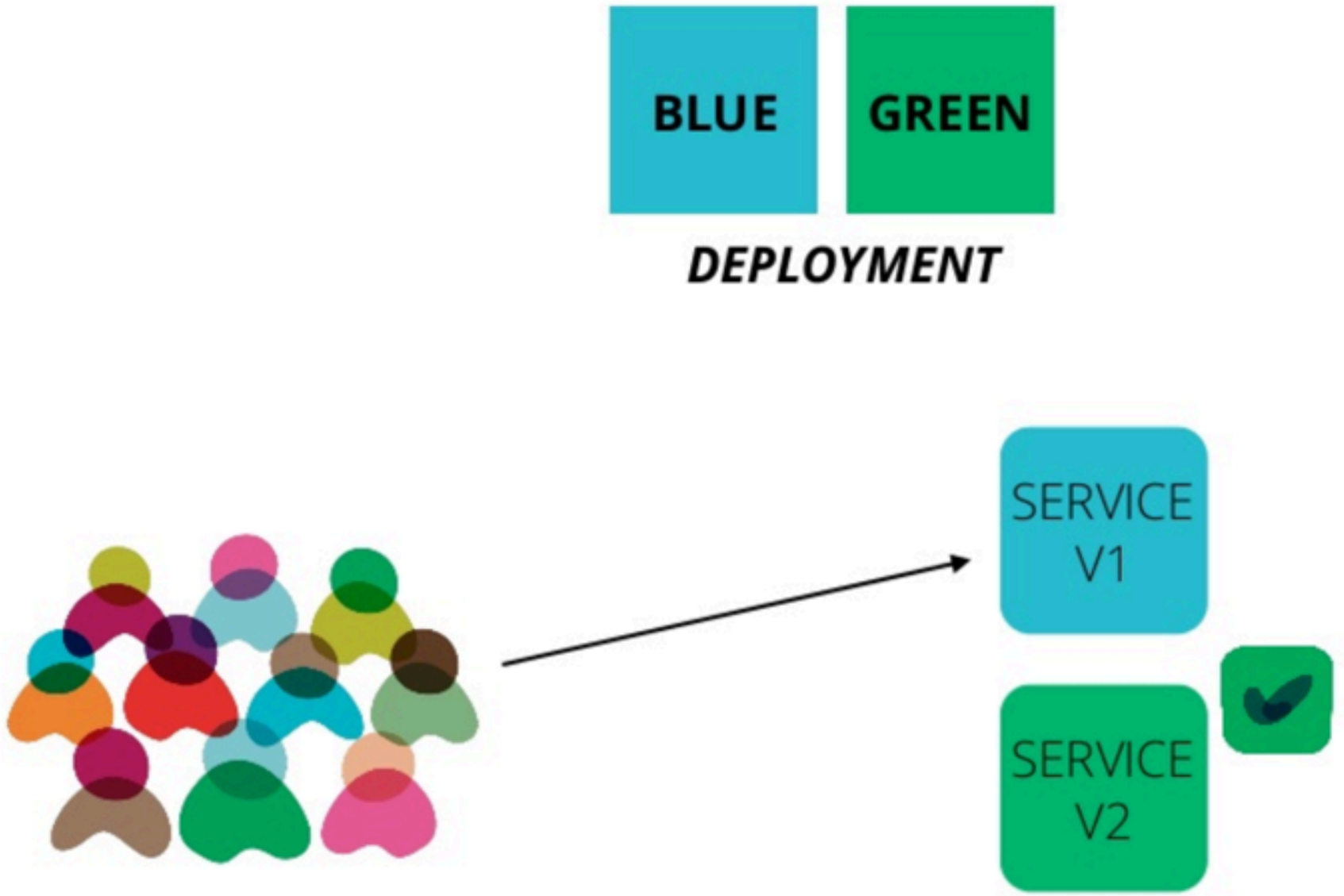
Deployment strategies

Blue Green Deployment

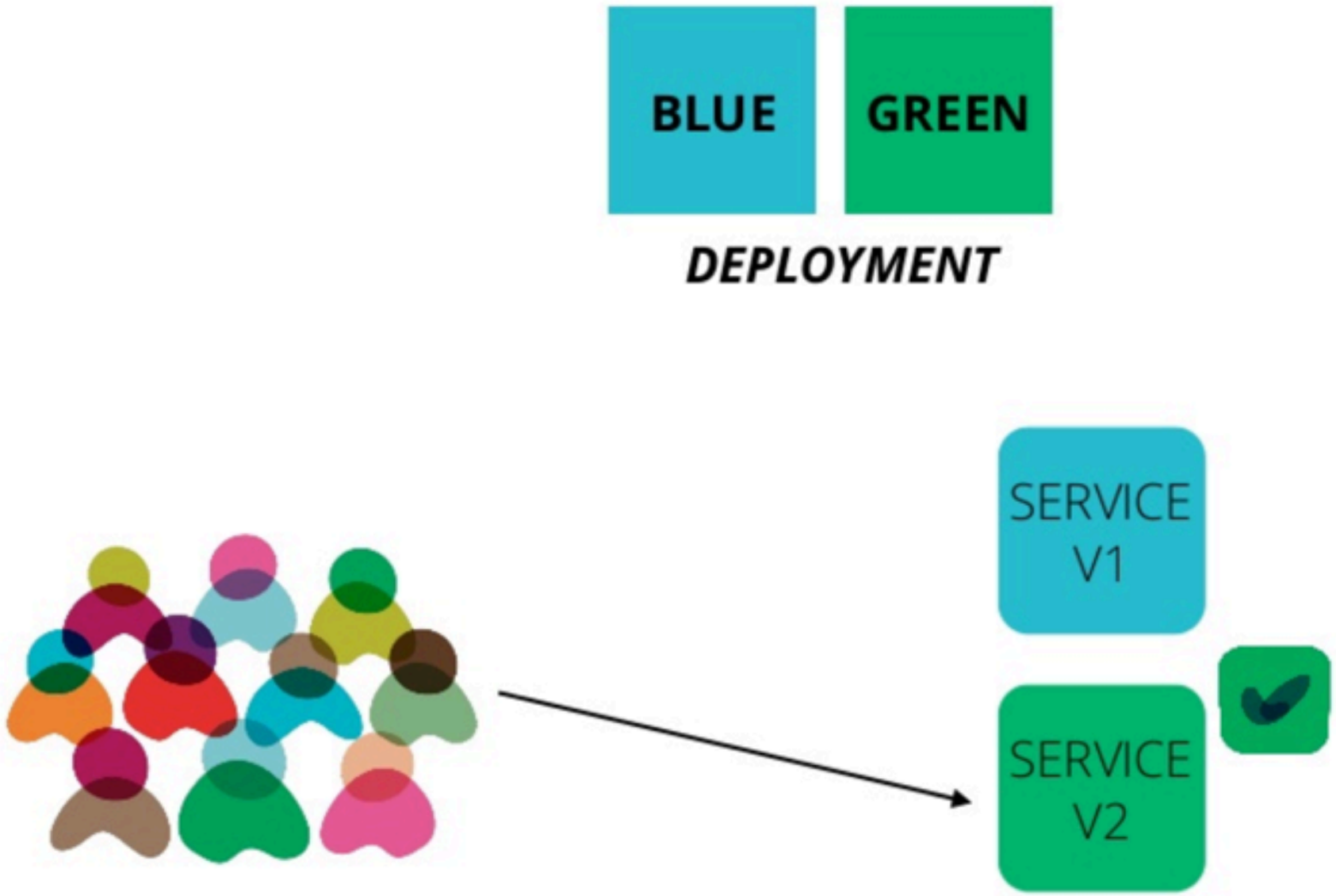
Blue Green Deployment



Blue Green Deployment



Blue Green Deployment

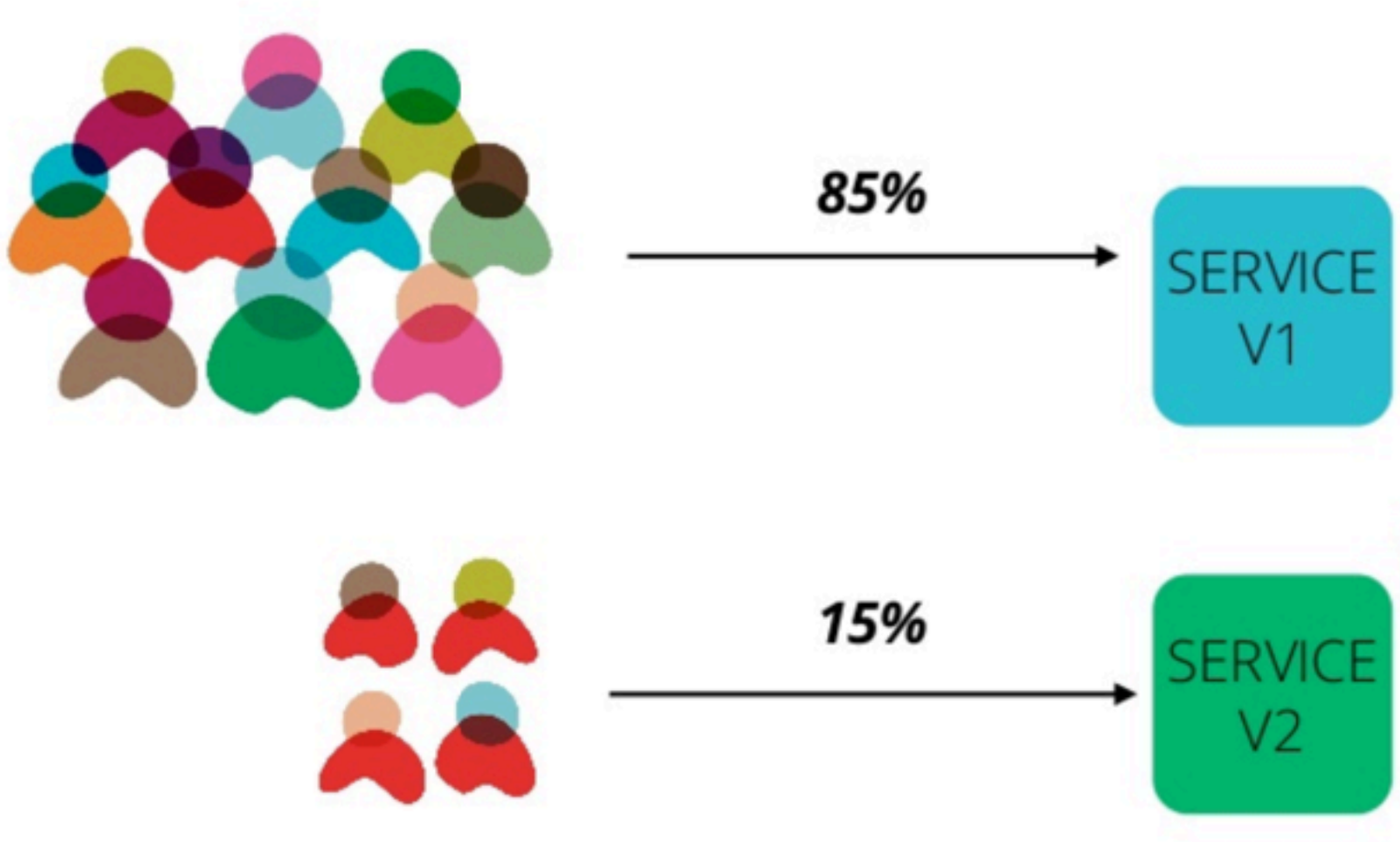


Canary Release

Canary Release

Canary Release

Canary Release



Mean Time to Recover (MTTR)

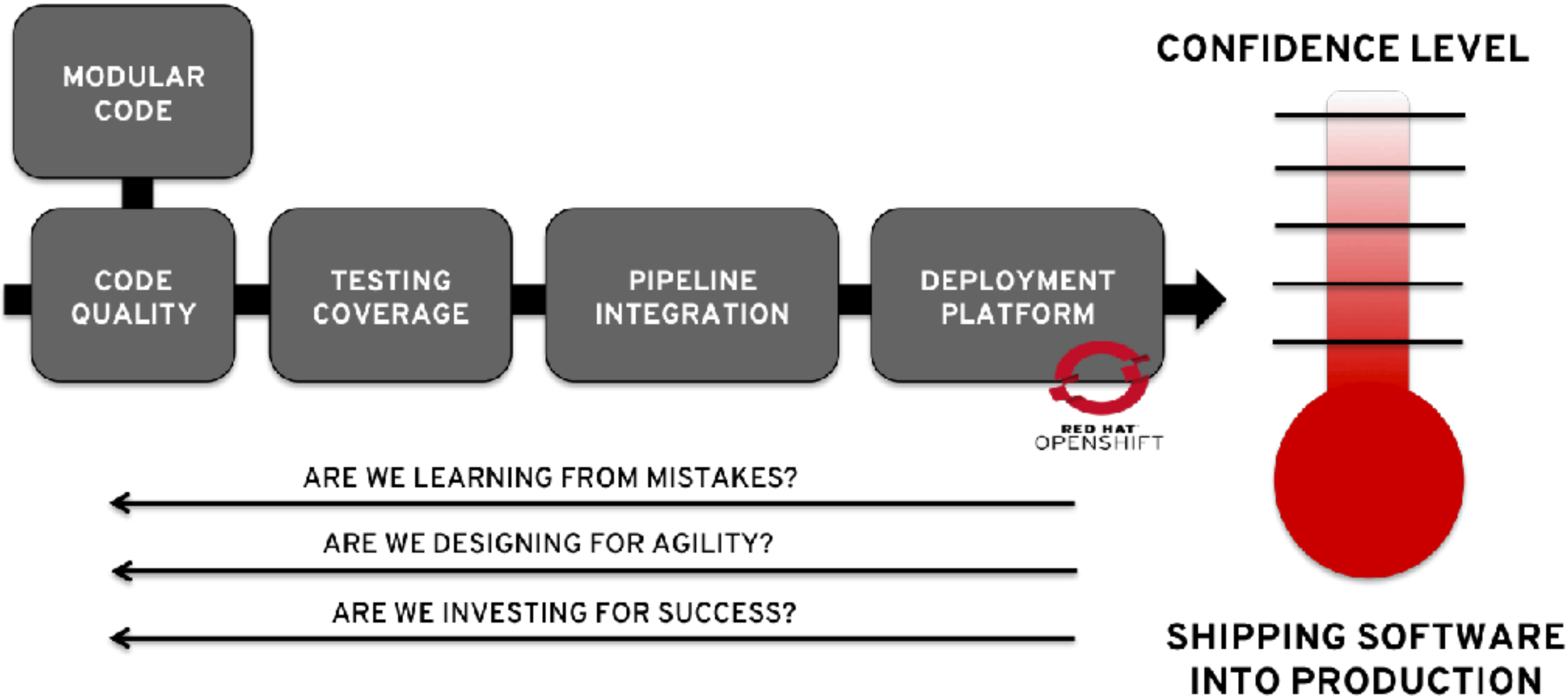
Mean Time to Recover (MTTR)

Tests are very important to reduce amount of defects in your systems. However, it's important to acknowledge that bugs will always happen in production.

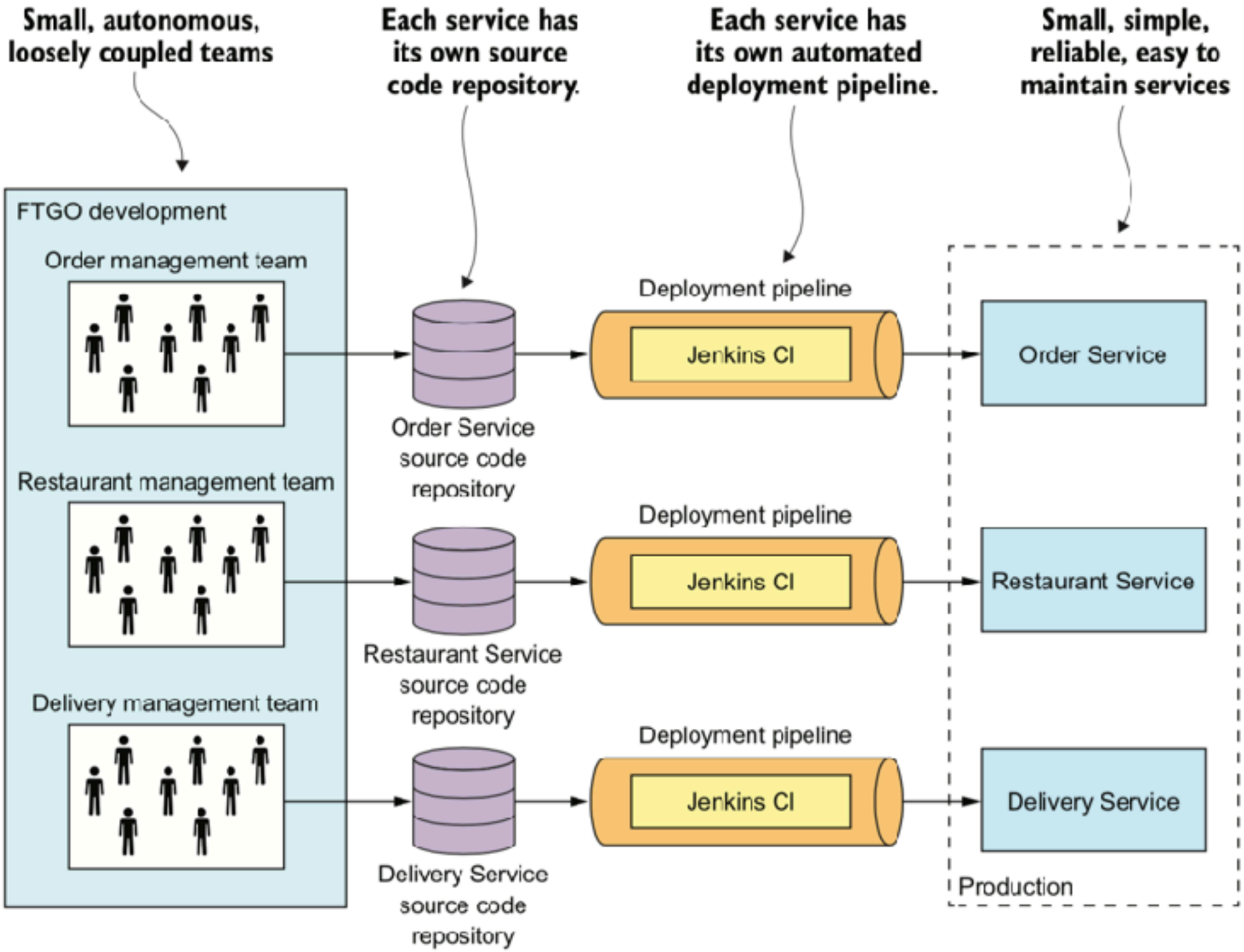
Mean Time to Recover (MTTR)

How **fast to recover** from them will help determining our success !

What can i measure ?



Deployment pipeline in each service

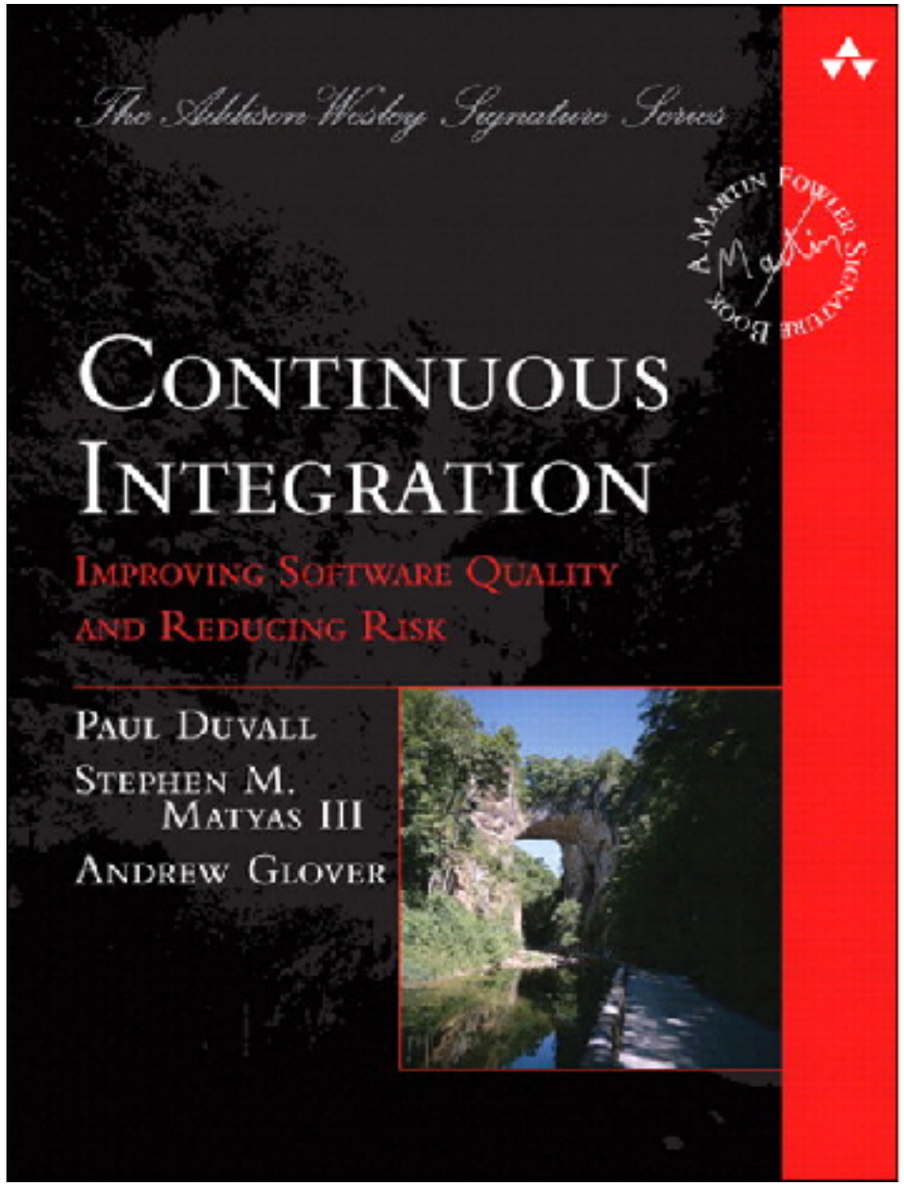


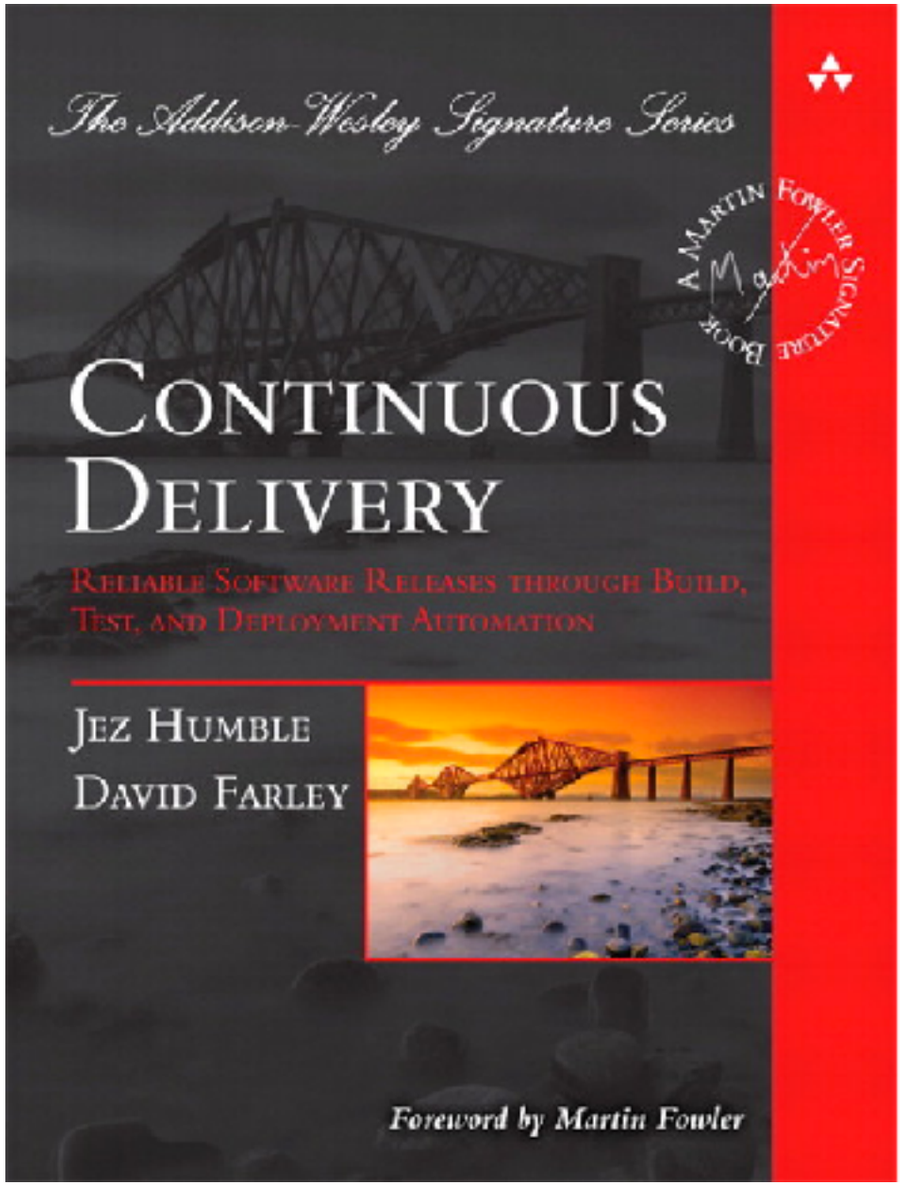
Start with Continuous Integration Continuous Delivery

“Behind every successful agile project, there is a

Continuous Integration System”

Improve quality and reduce risk

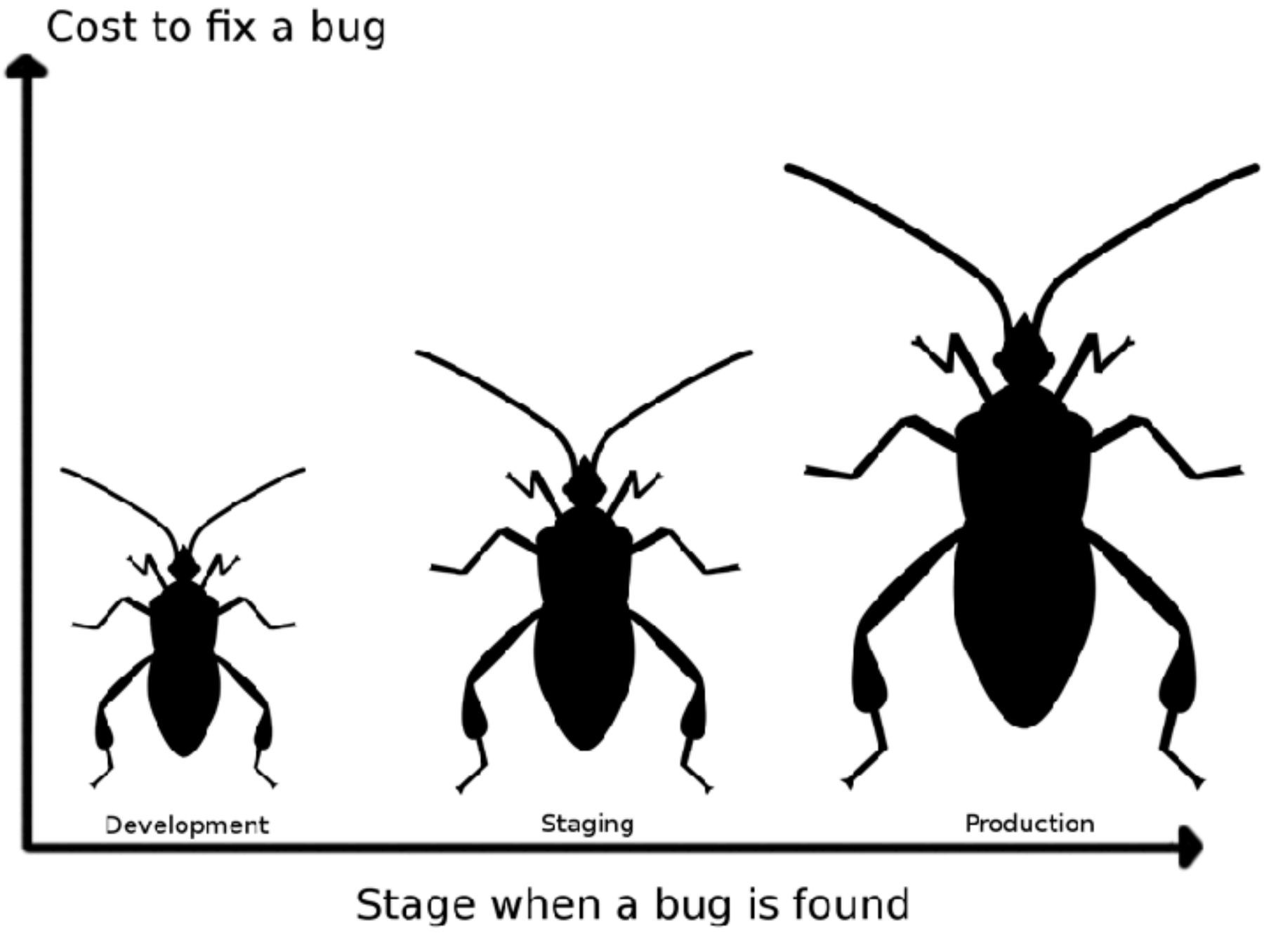


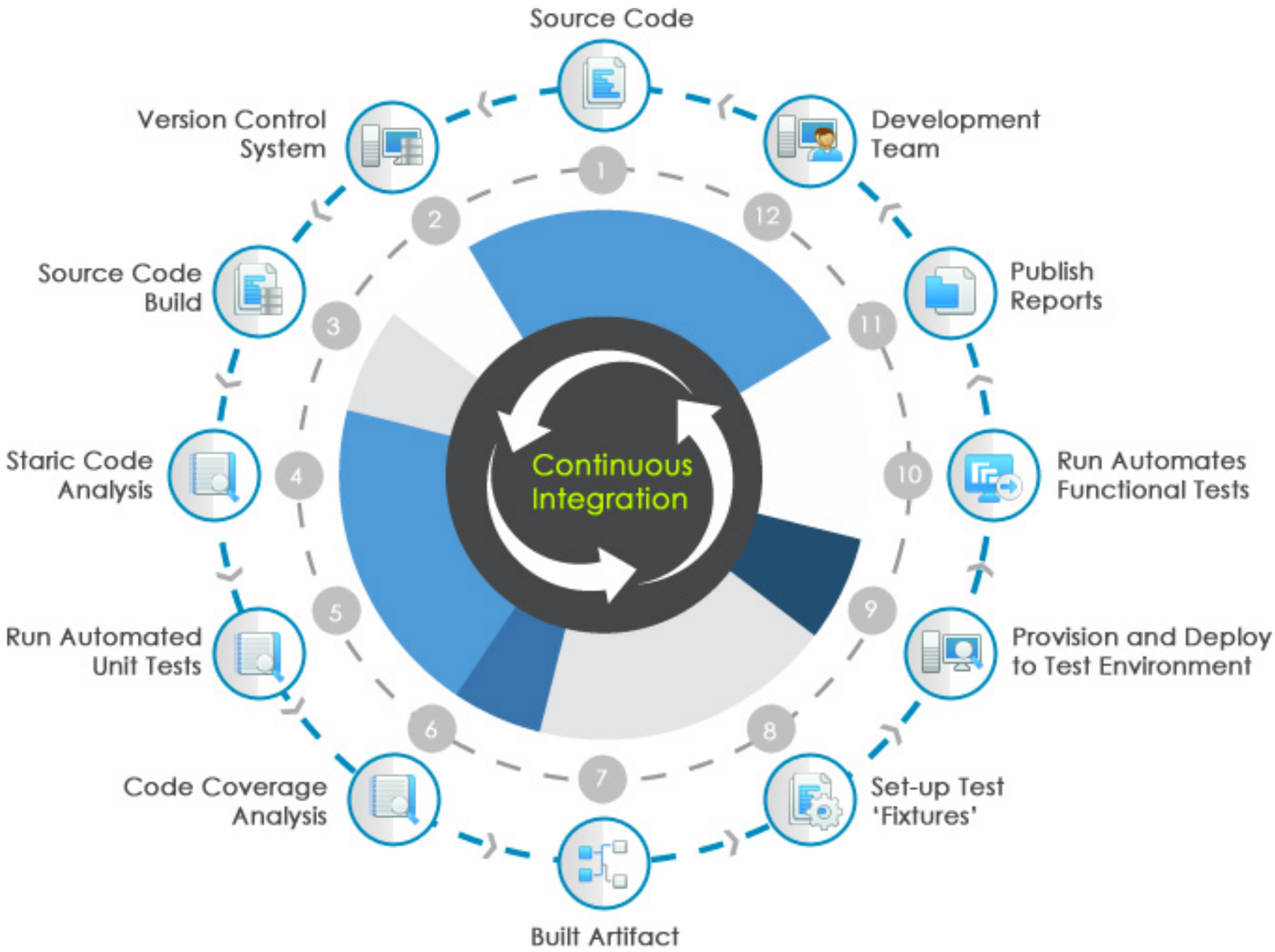


The cost of integration

1. Merging the code
2. Duplicate changes
3. Test again again !!
4. Fixing bugs
5. Impact on stability

The cost of integration





Jenkins

Bamboo

TeamCity

go

Visual Studio
Team Foundation Server

Hudson

Travis

wercker

circleci

CI is about what people do not about what tools they use

Continuous Integration

Discipline to **integrate frequently**

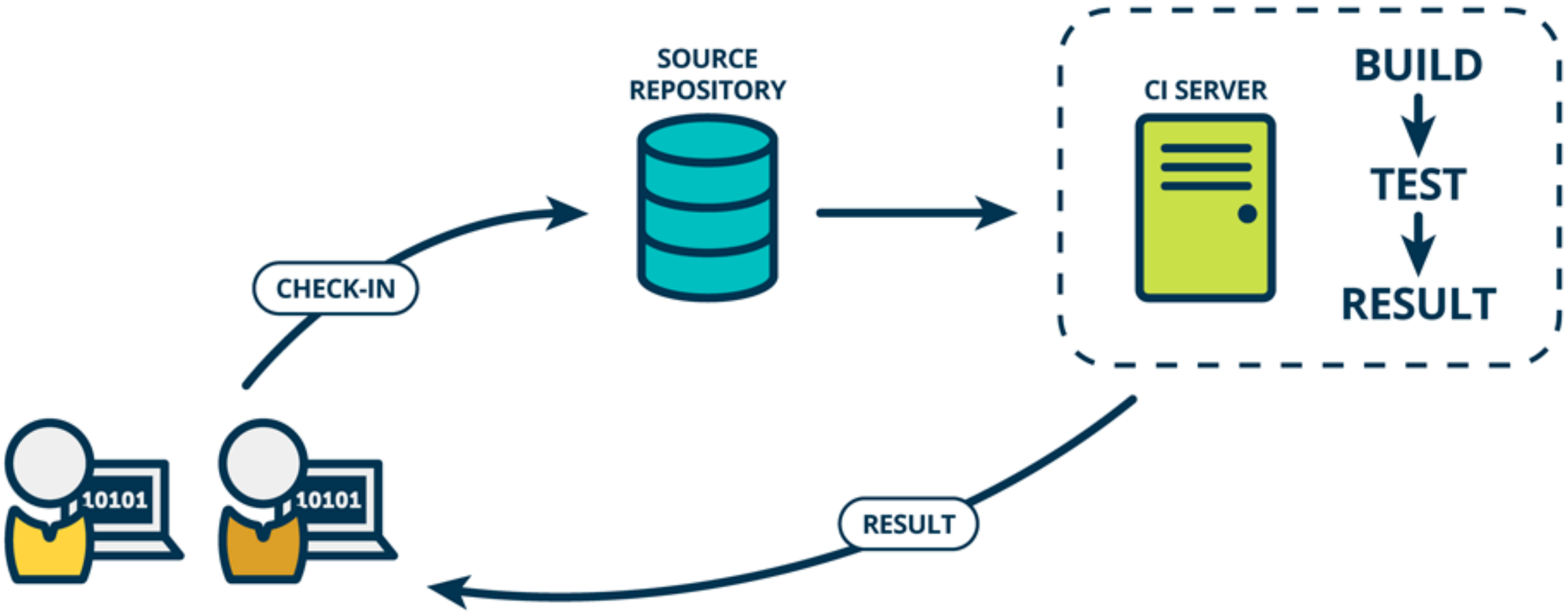
Continuous Integration

Strive to make **small change**

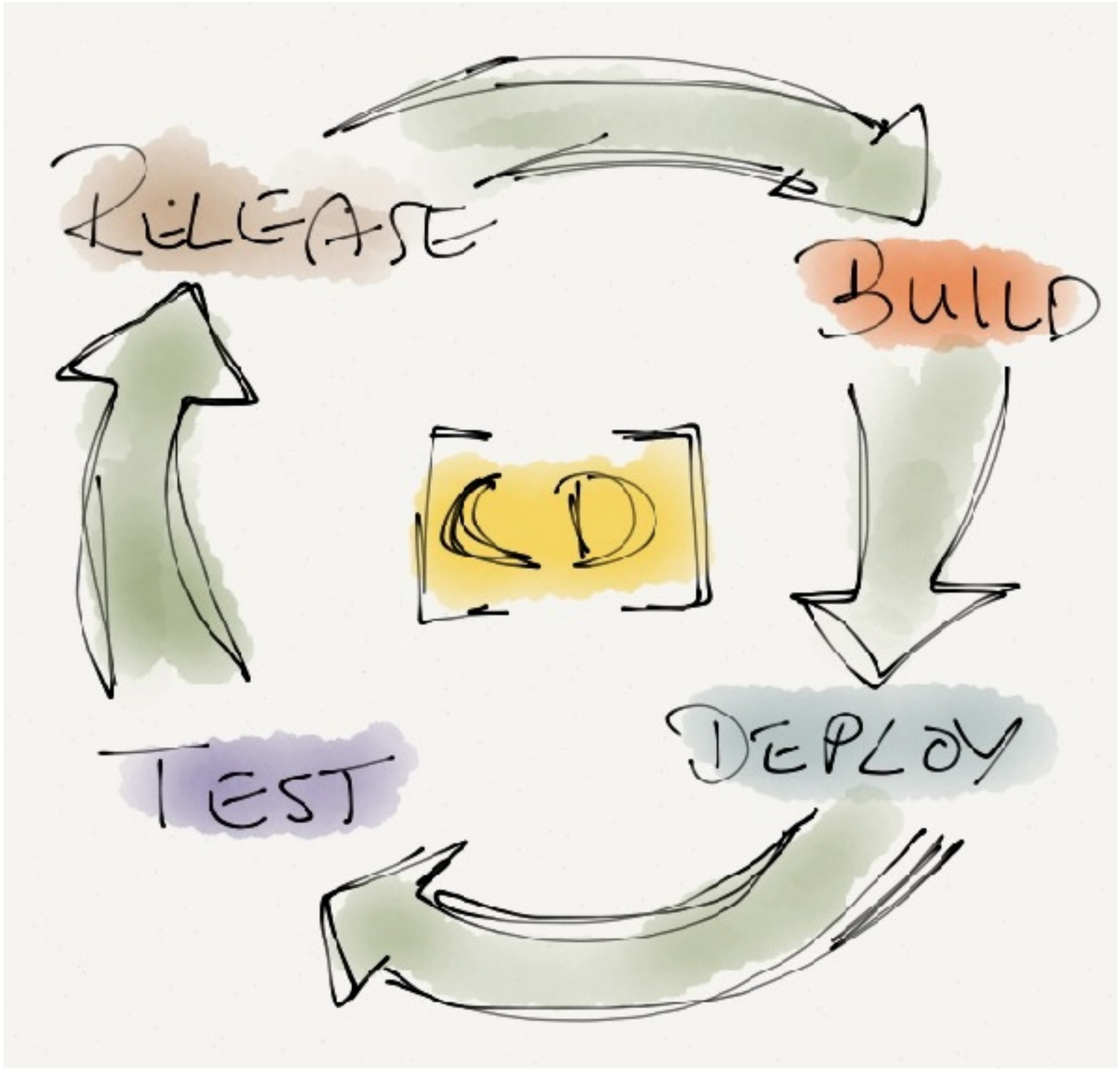
Continuous Integration

Strive for **fast feedback**

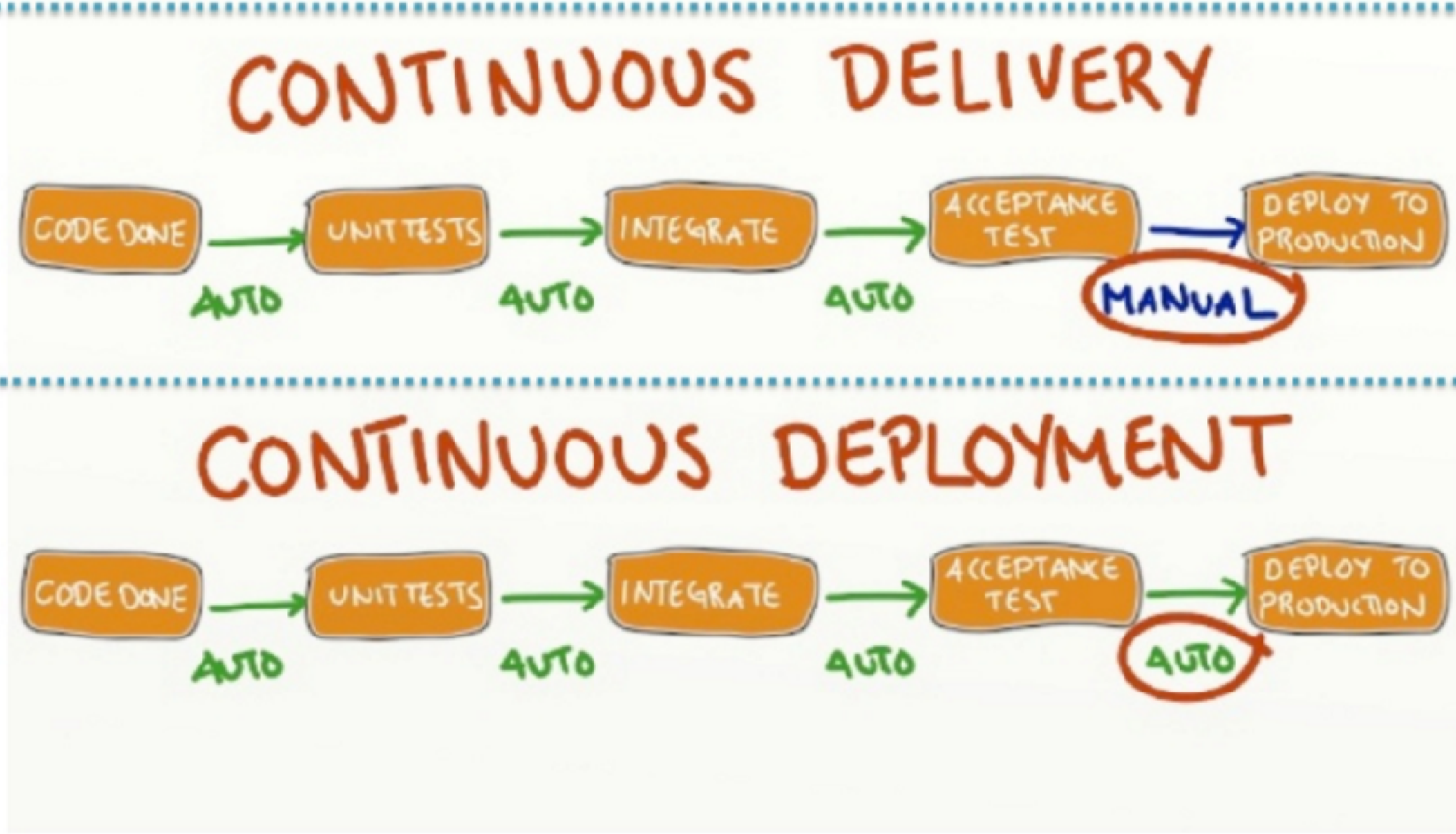
Continuous Integration



CD ?

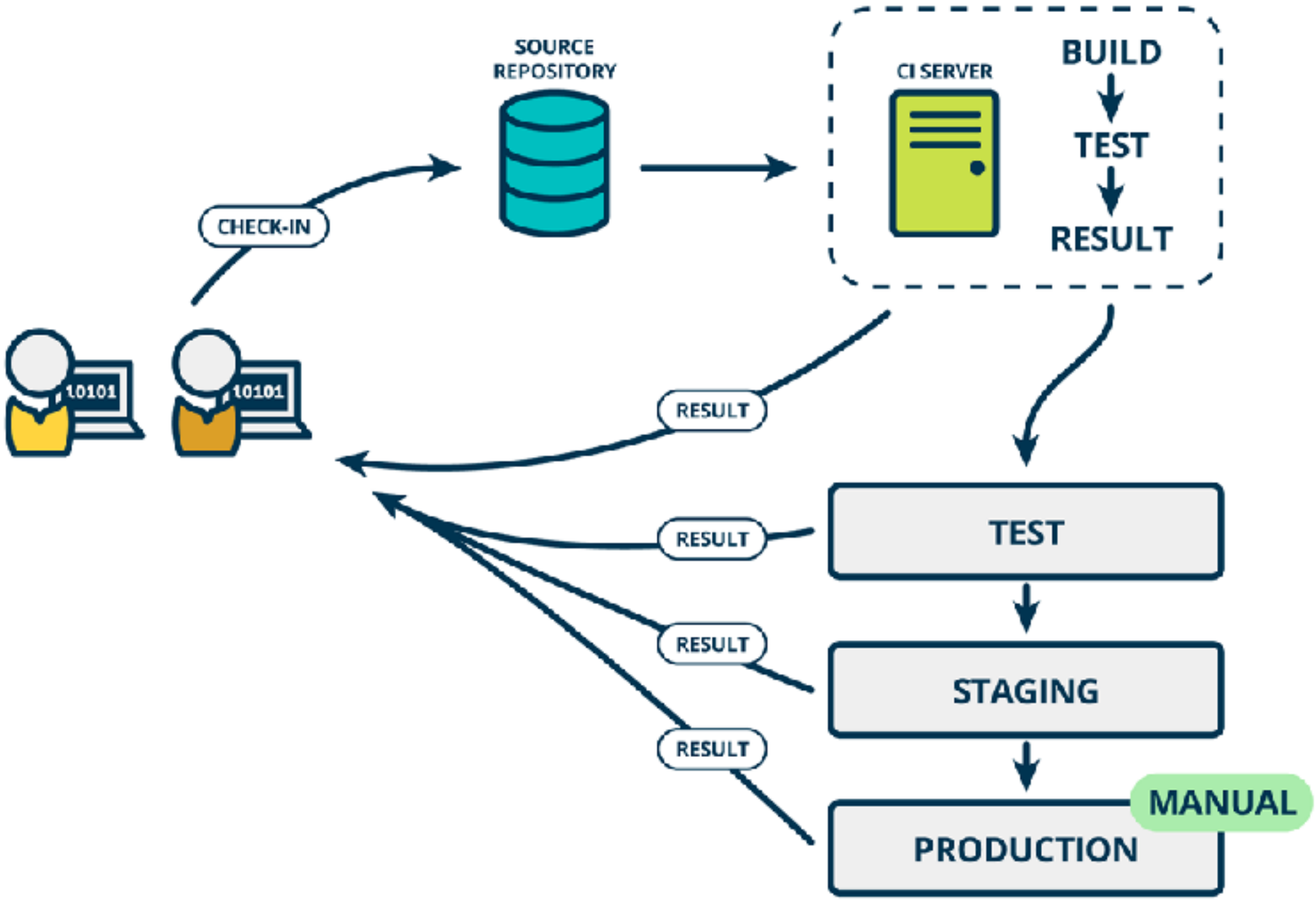


CD ?

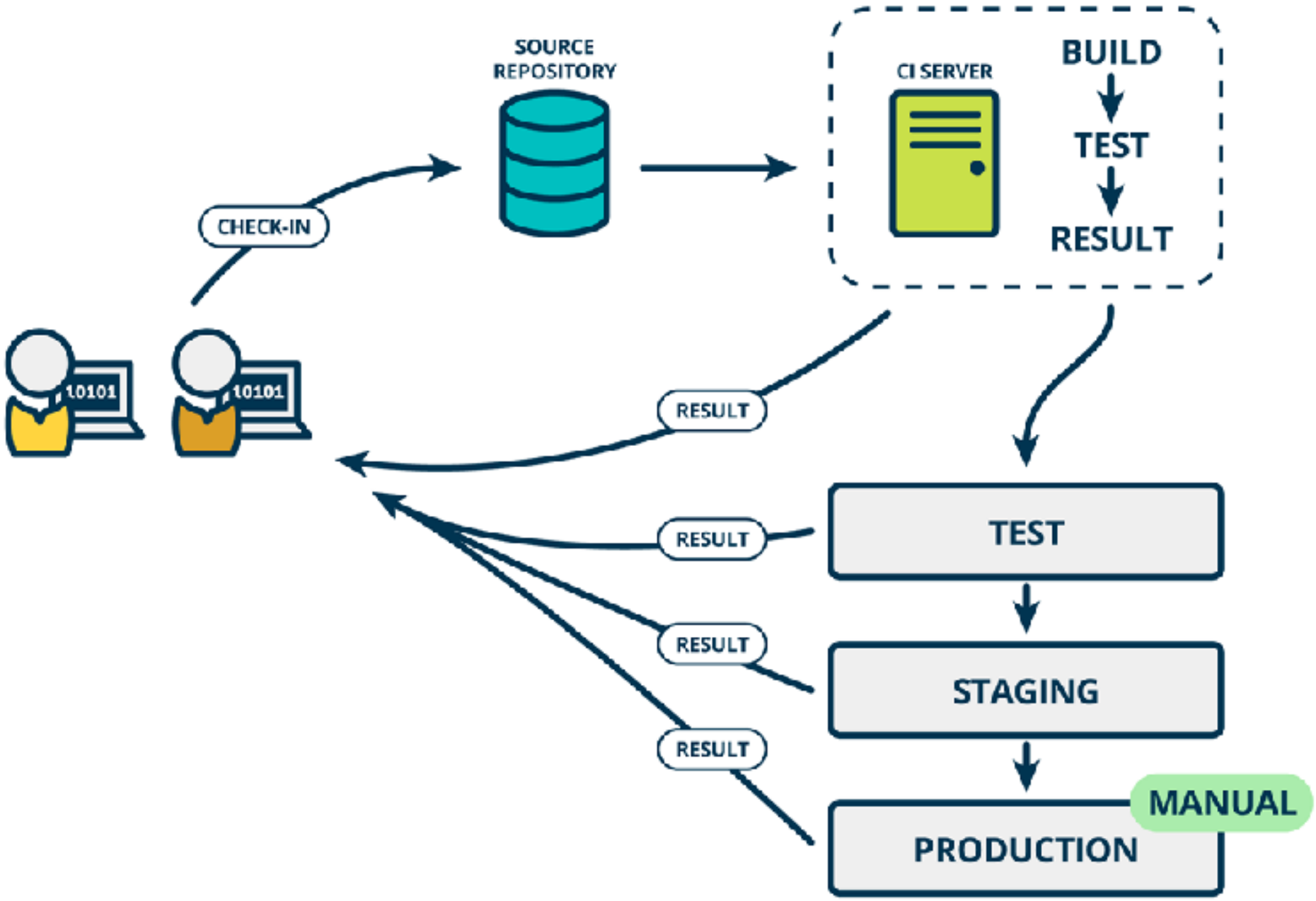


http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

Continuous Delivery



Rise of DevOps



Continuous Integration

is a Software development **practices**

Practice 1

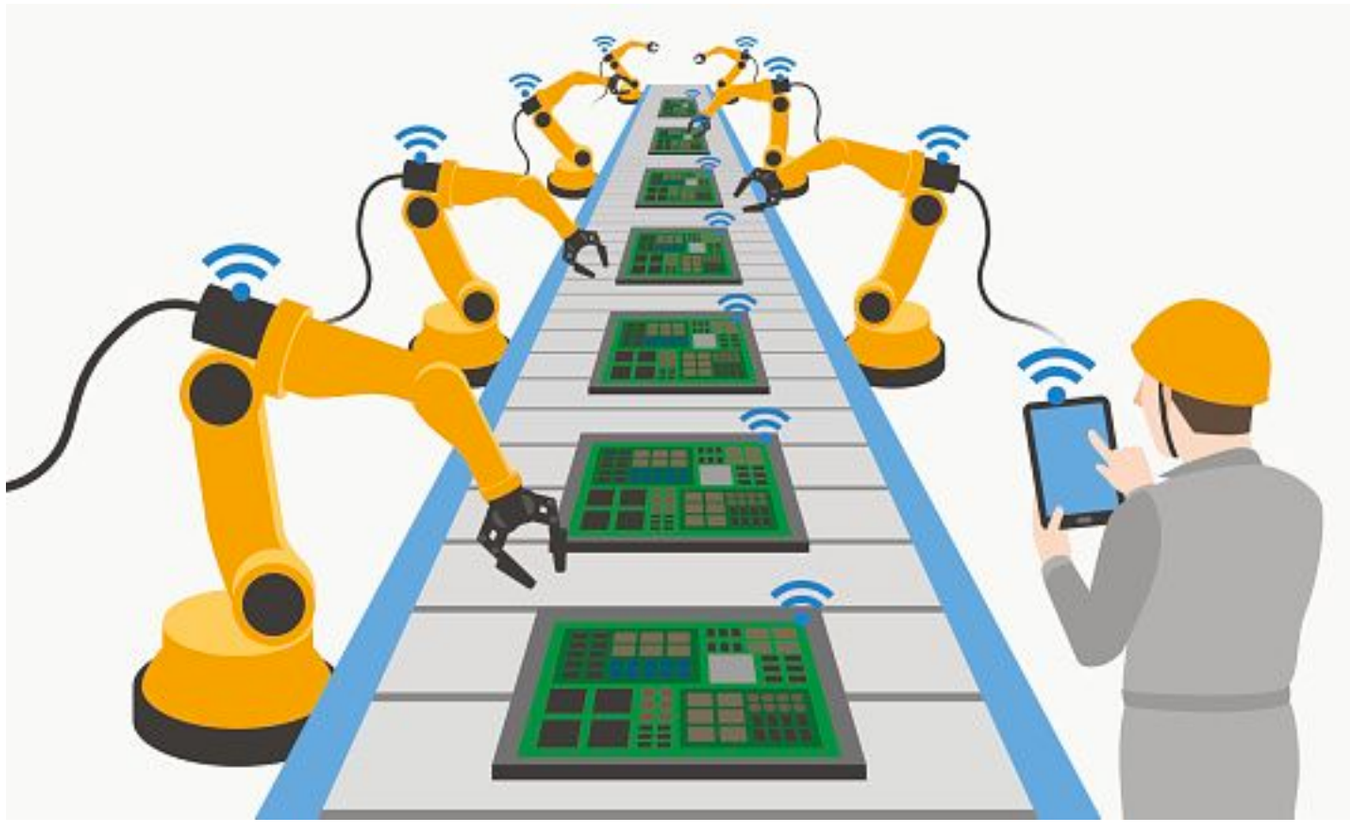
Maintain a single source repository

In general, you should store in source control everything you need to build anything

Practice 2

Automated the build

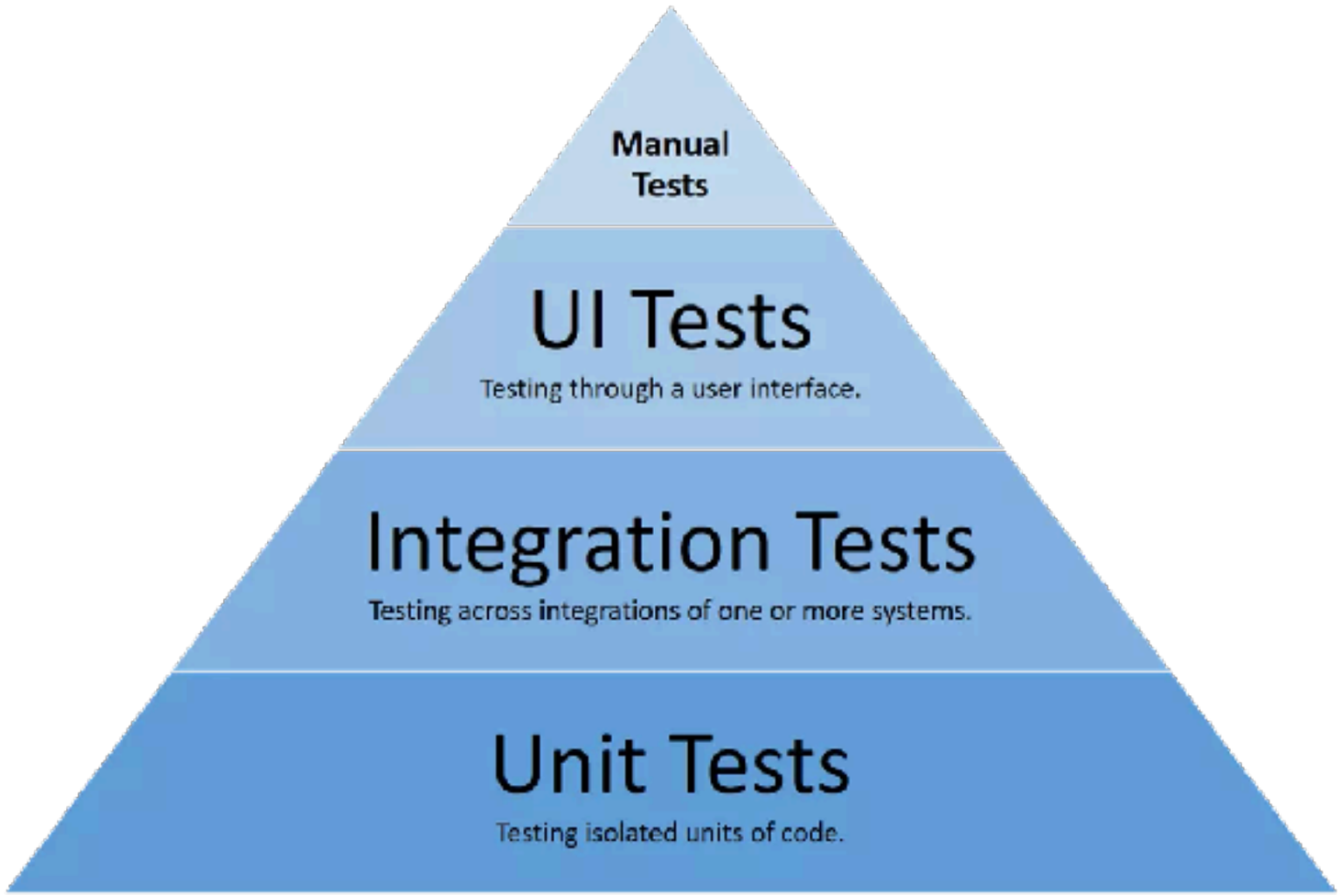
Automated environment for builds



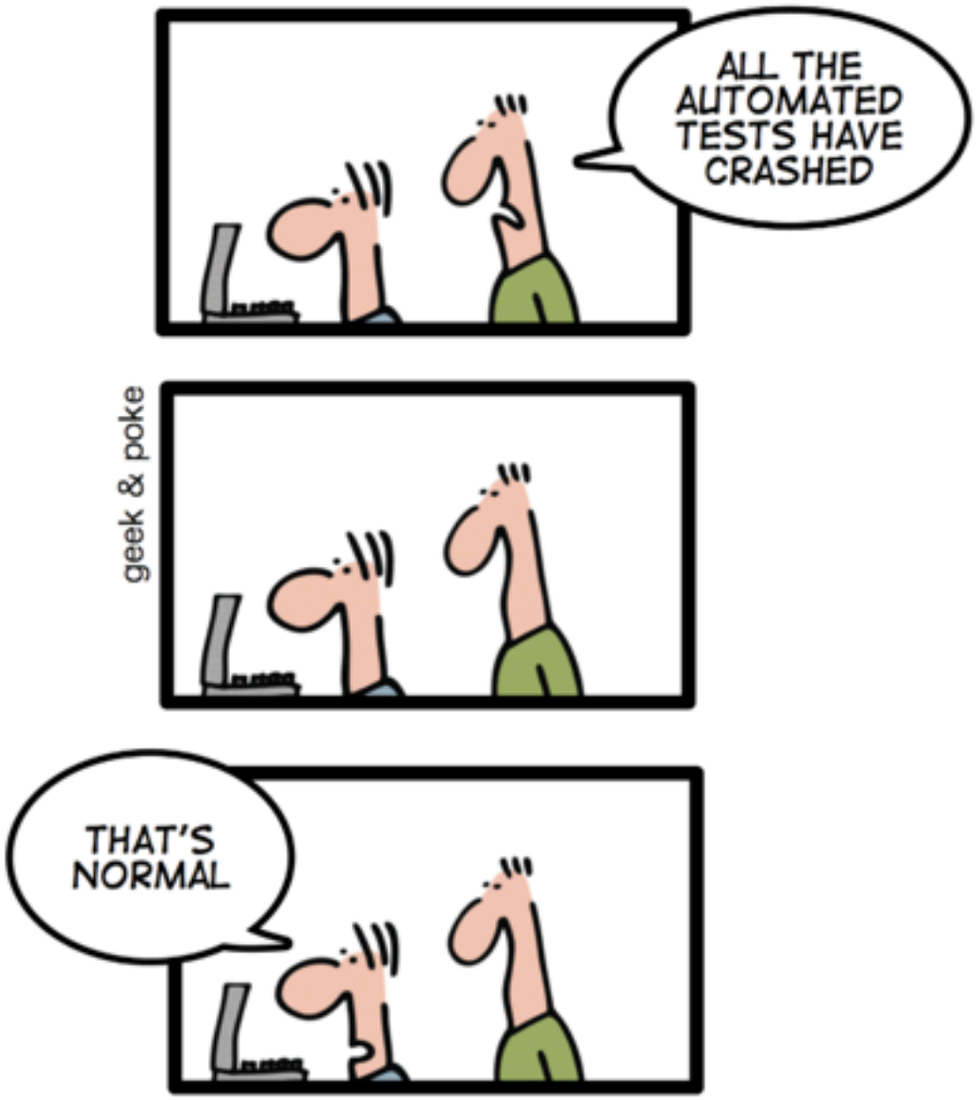
Practice 3

Make your build **self-testing**

Build process => compile, linking and **testing**



TODAY: CONTINUOUS INTEGRATION GIVES YOU THE COMFORTING FEELING TO KNOW THAT EVERYTHING IS NORMAL

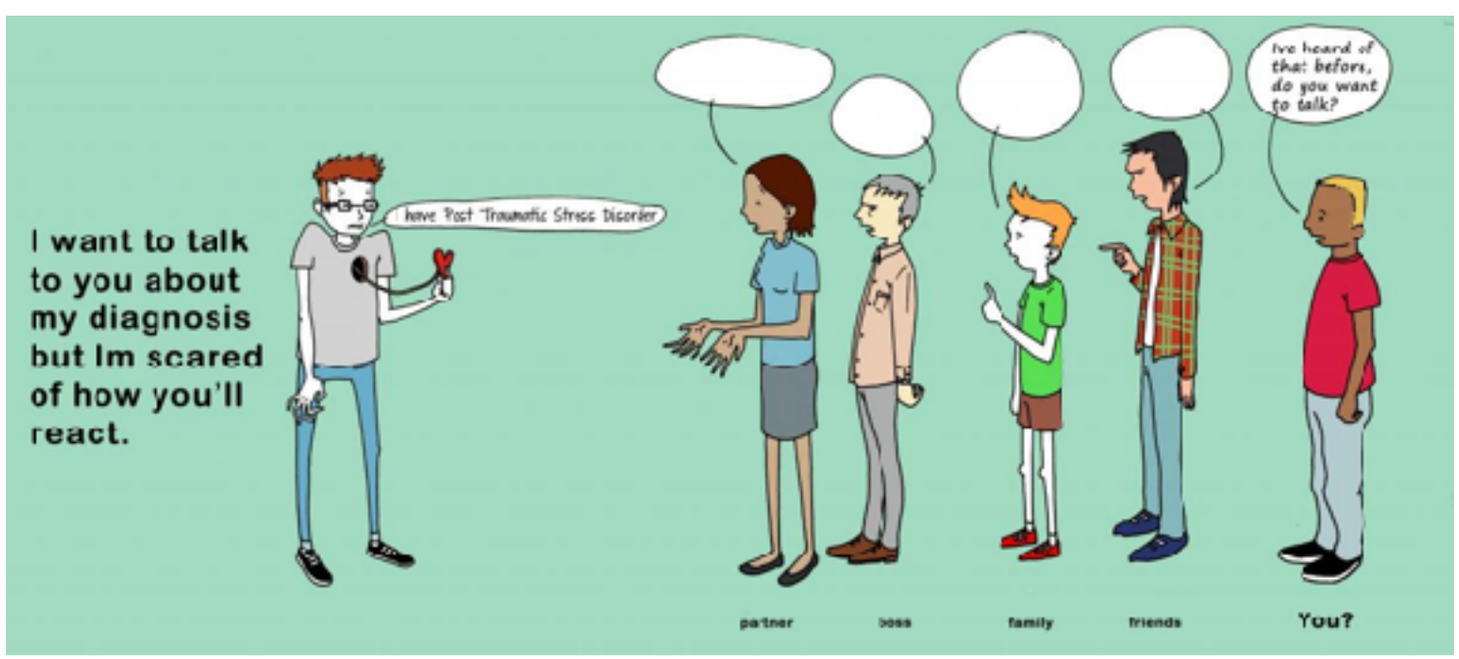


Practice 4

Everyone **commits** to the mainline everyday

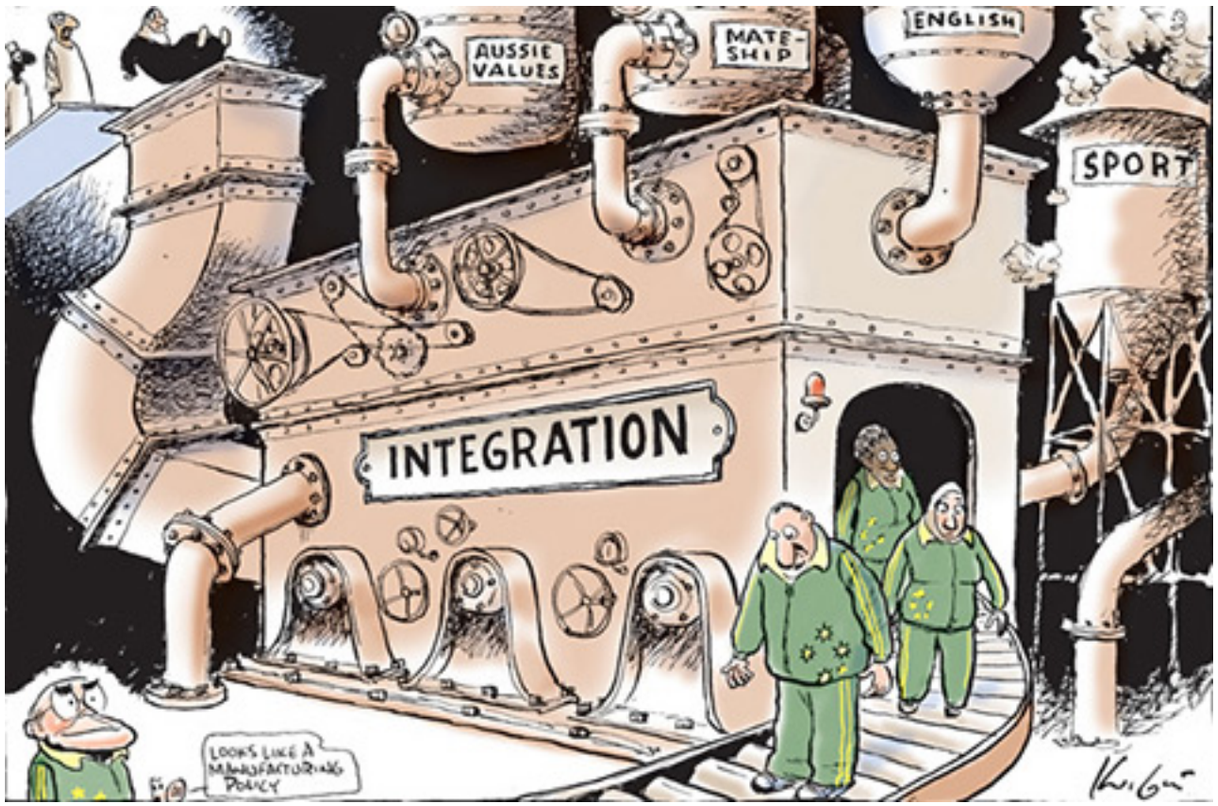
Integration is about **communication**

Integration allows developers to **tell** other developers



Practice 5

Every commits should build the mainline on an **Integration machine**



Nightly build is not enough for Continuous Integration

Practice 6

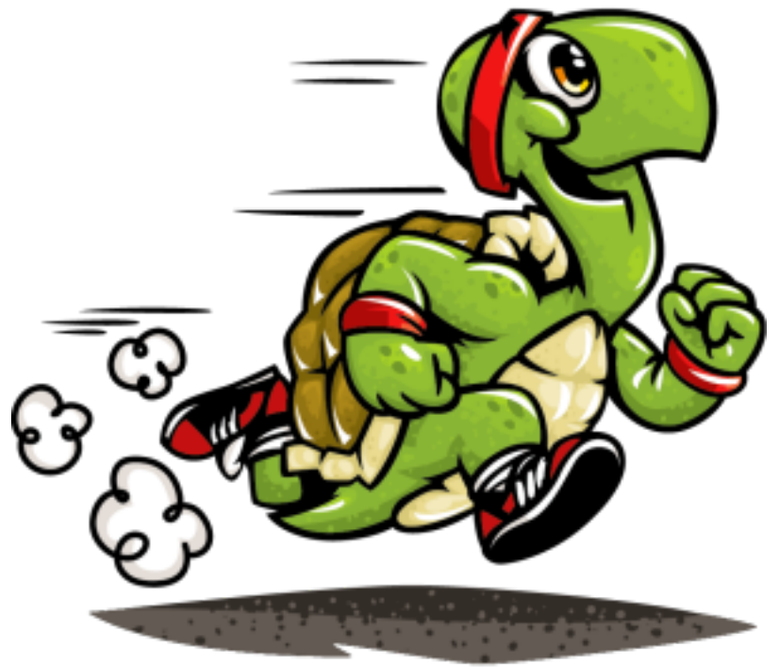
Fix broken builds **immediately**

"Nobody has a higher priority task than fixing the build"

Practice 7

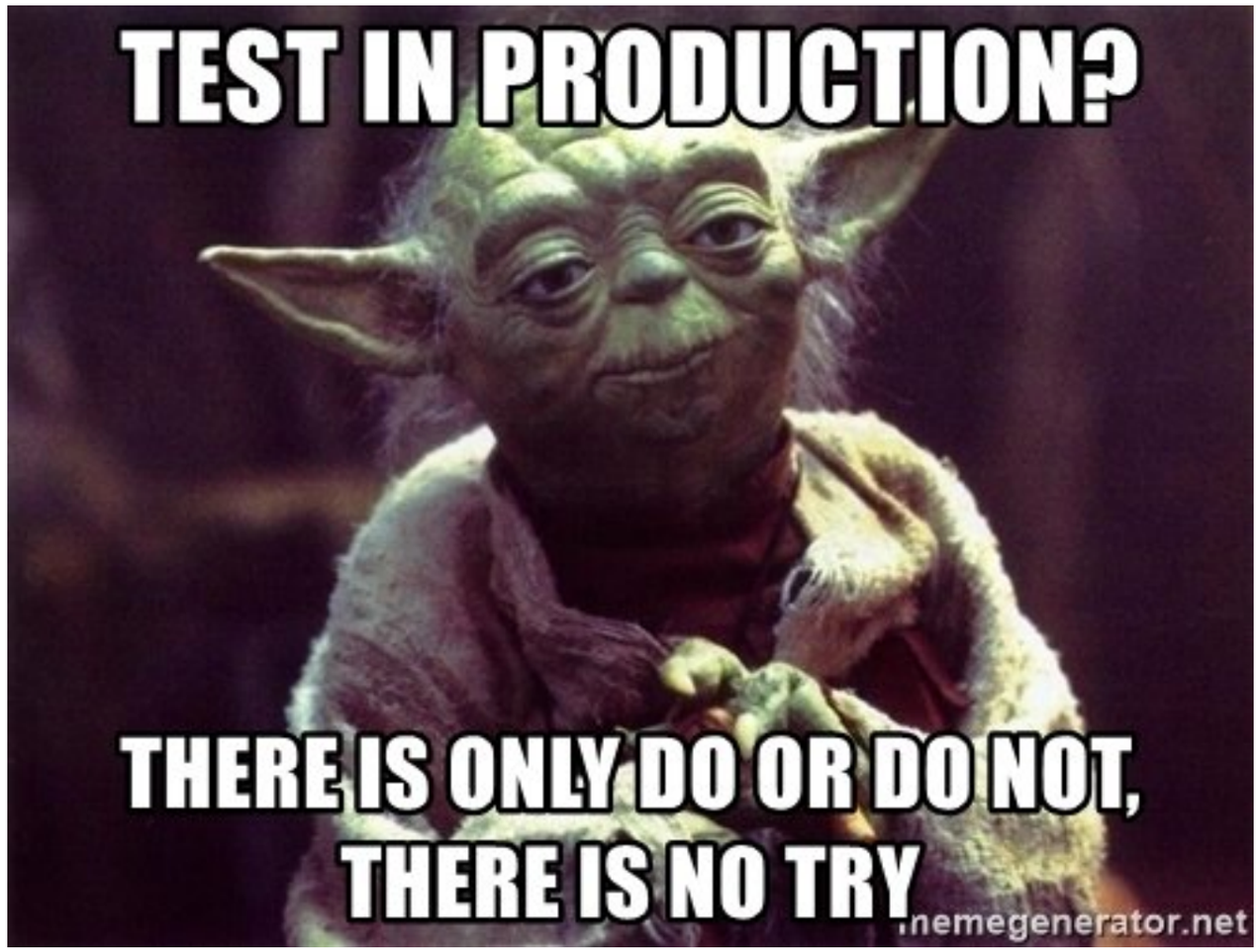
Keep the build **fast**

Continuous Integration is to provide rapid feedback



Practice 8

Test in clone of the **Production** environment



Practice 9

Make it easy for anyone to get
the latest executable

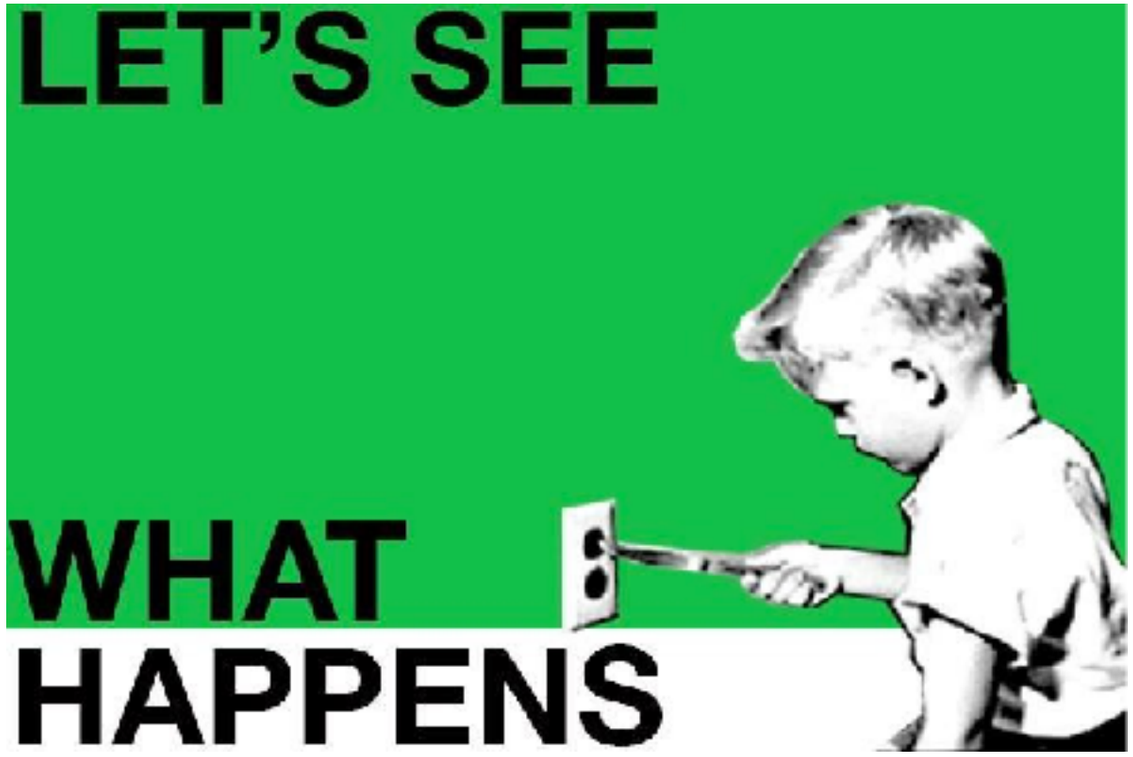
Make sure well known place where people can find

Practice 10

Everyone can see what's happening

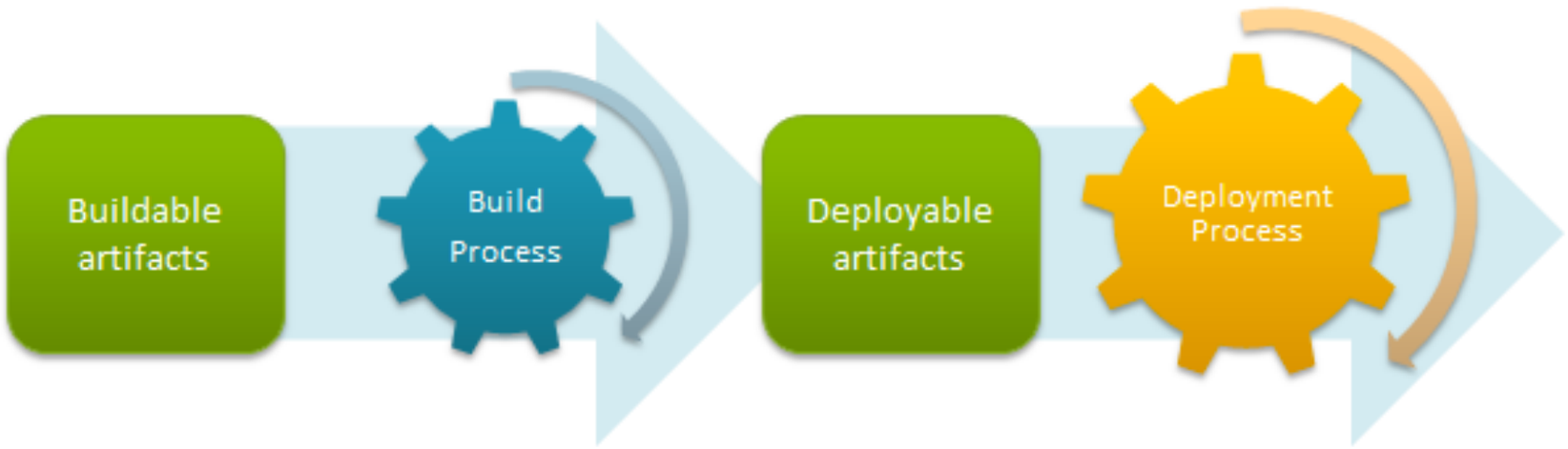
Easier to see the state of the system and changes

Show the good information



Practice 11

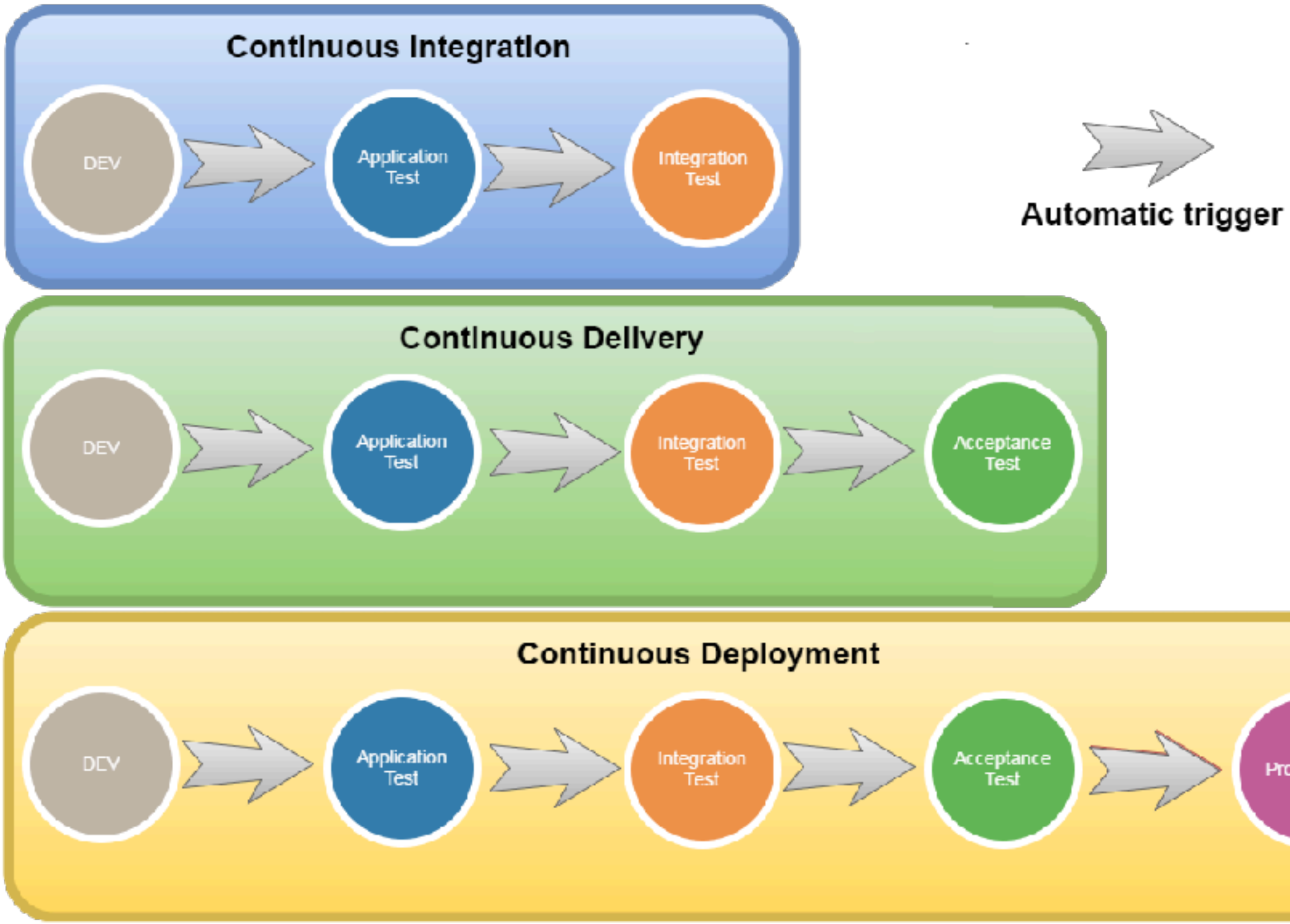
Automated deployment



Continuous Delivery

Continuous Delivery

Use version control for all production artifacts
Automate your deployment process
Implement continuous integration (CI)
Use trunk-based development methods
Implement test automation
Support test data management
Integrate security into software development process



How to achieve the CI ?

1. Use good version control

Local

Centralize

Distributed

2. Choose Branch strategy

Main only

Development isolation

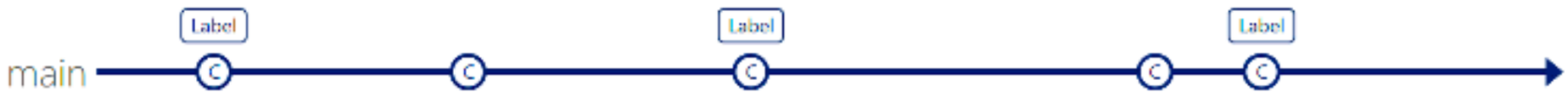
Feature isolation

Release isolation

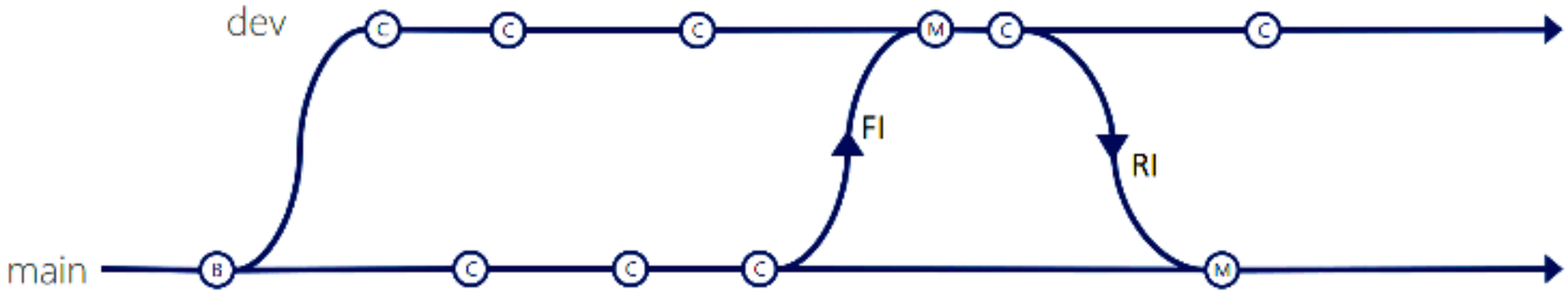
Service and Release isolation

Service, Hotfix and Release isolation

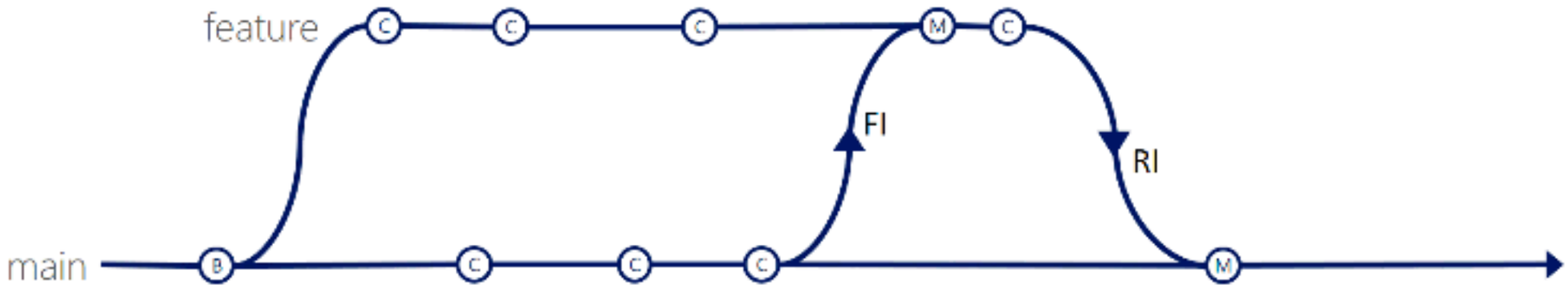
Main only



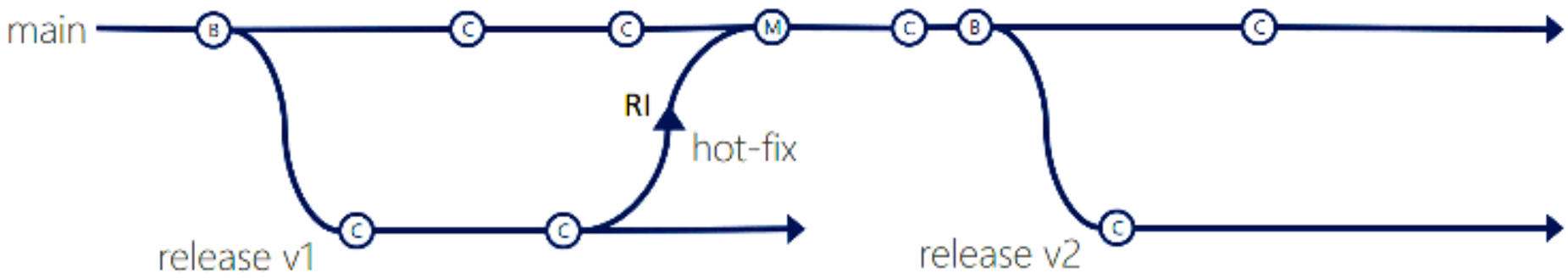
Development isolation



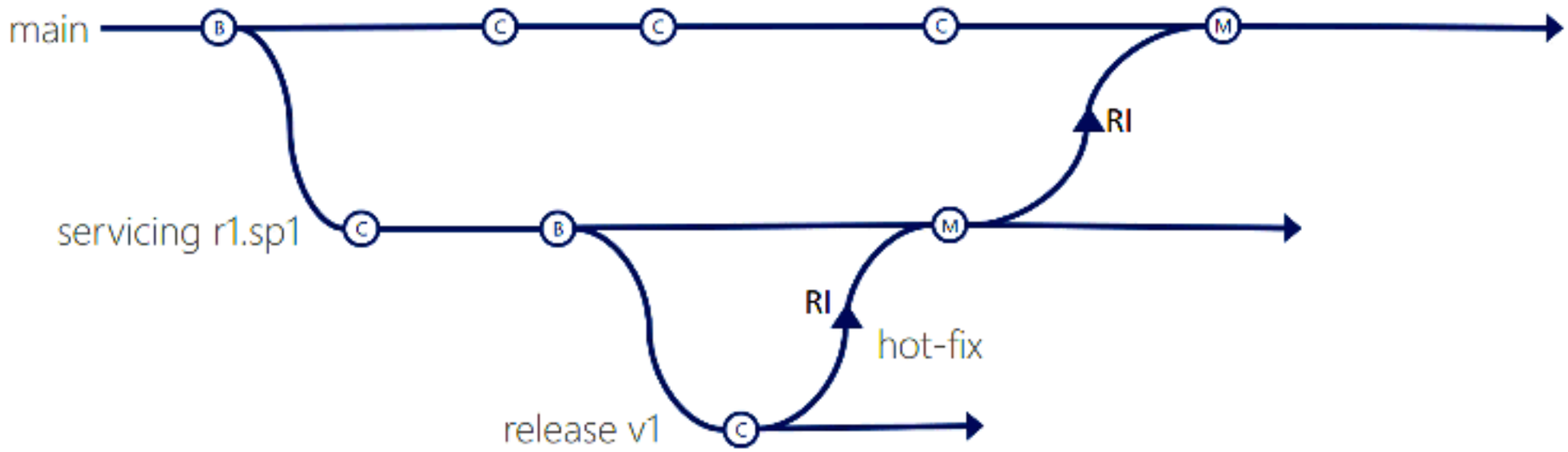
Feature isolation



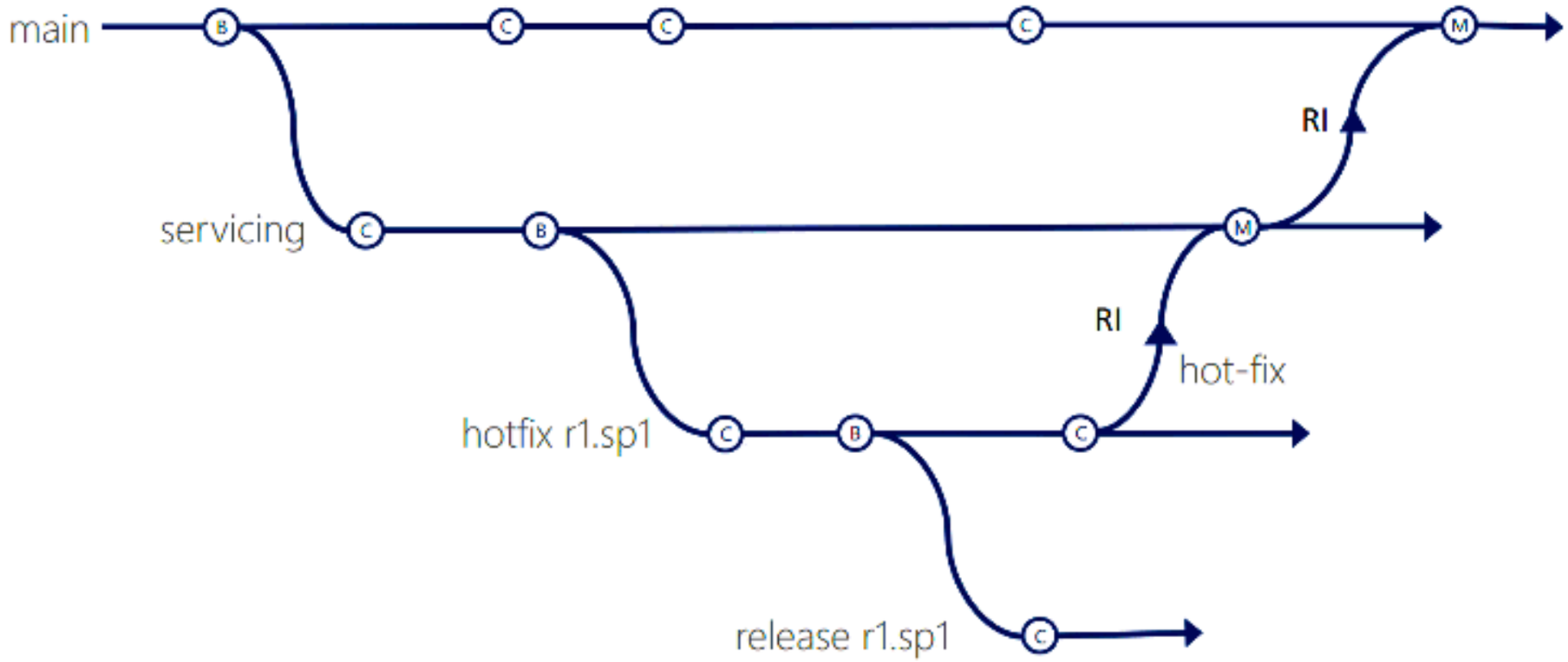
Release isolation



Service and Release isolation



Service, Hotfix, Release isolation

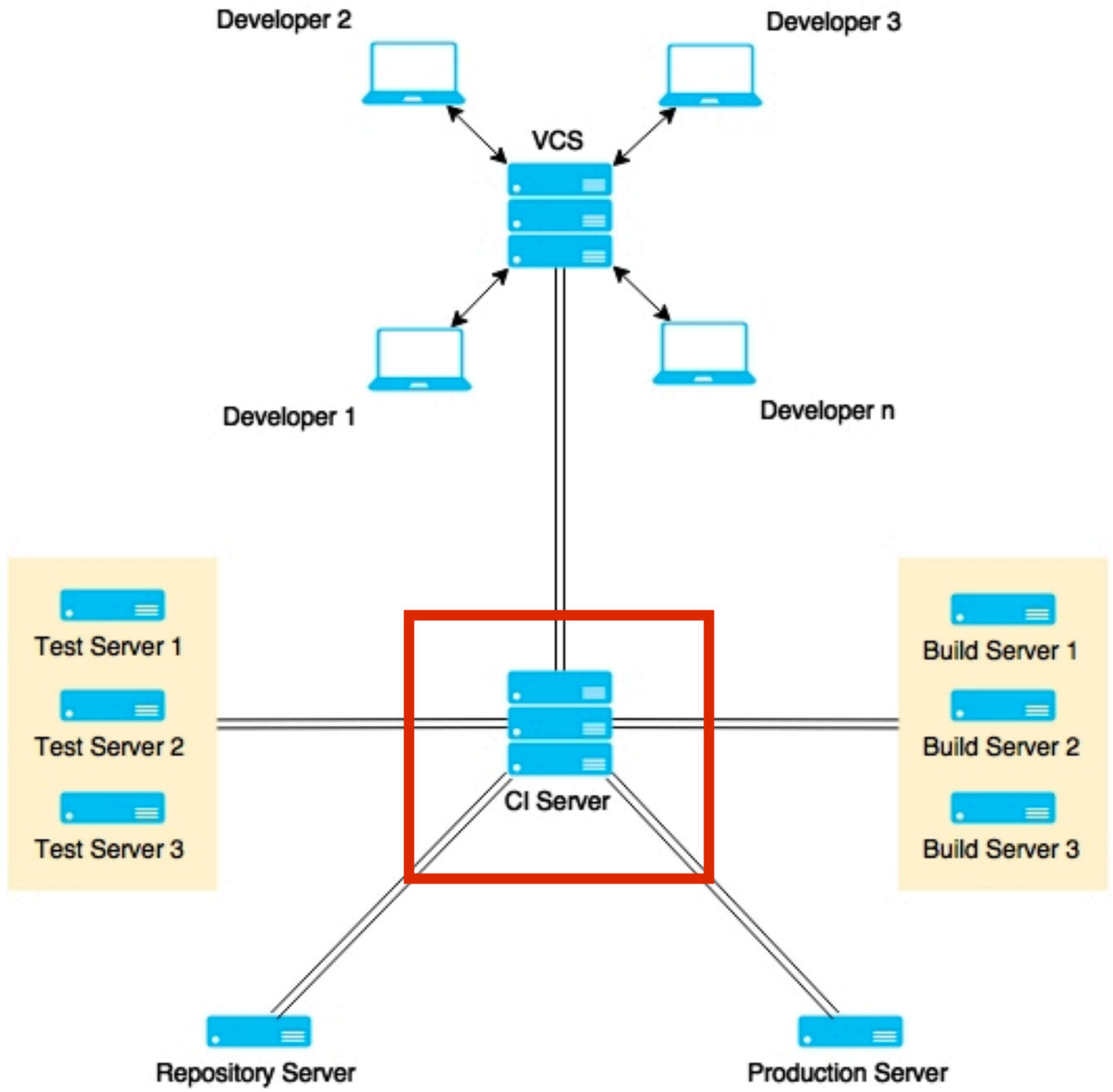


Validate, Validate and Validate

Suggestion

Keeping branches short-lived, merge conflicts are keep to as few as possible

3. Use good CI tool



Jenkins

Bamboo

TeamCity

go

Visual Studio Team Foundation Server

Hudson

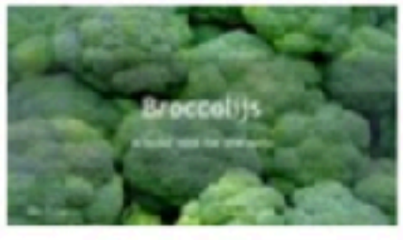
Travis

wercker

circleci

4. Use good build tool

- Javascript
  - Gulp, Grunt, Brocolli



- C#/.NET
  - Nant, MSBuild

- Java/JVM
  - Ant, Maven, Gradle, SBT, Leiningen

More …

Use static code analysis

Automated testing

Automated deployment

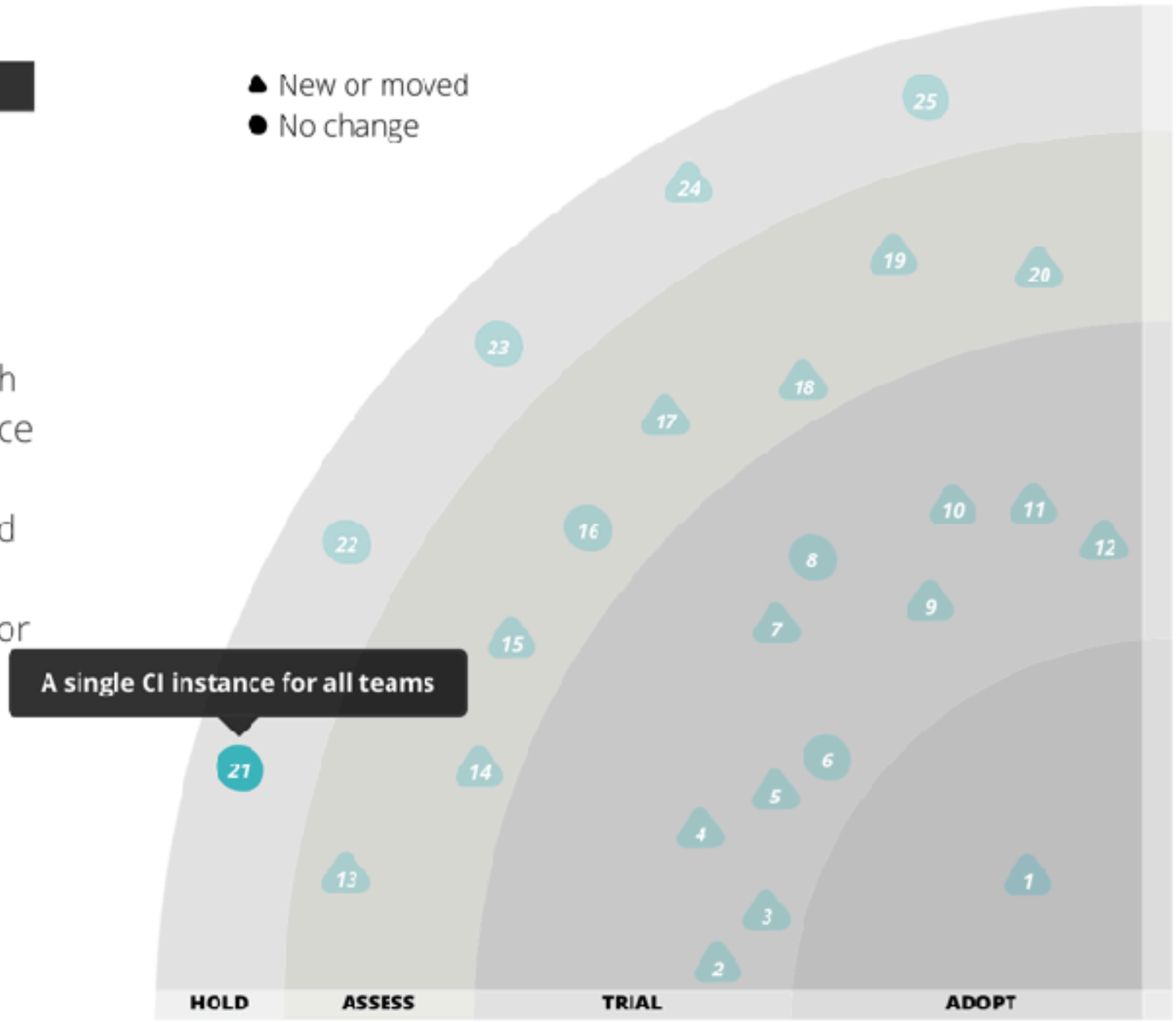
People discipline/habit

Anti-pattern for CI Server

HOLD ?

21. A single CI instance for all teams

We're compelled to caution, again, against creating **a single CI instance for all teams**. While it's a nice idea in theory to consolidate and centralize Continuous Integration (CI) infrastructure, in reality we do not see enough maturity in the tools and products in this space to achieve the desired outcome. Software delivery teams which must use the centralized CI offering regularly have long delays depending on a central team to perform minor configuration tasks, or to troubleshoot problems in the shared infrastructure and tooling. At this stage, we continue to recommend that organizations limit their centralized investment to establishing patterns, guidelines and support for delivery teams to operate their own CI infrastructure.



https://www.thoughtworks.com/radar/techniques