# BUILDING EVOLUTIONARY ARCHITECTURES

# THE FIGHTERS

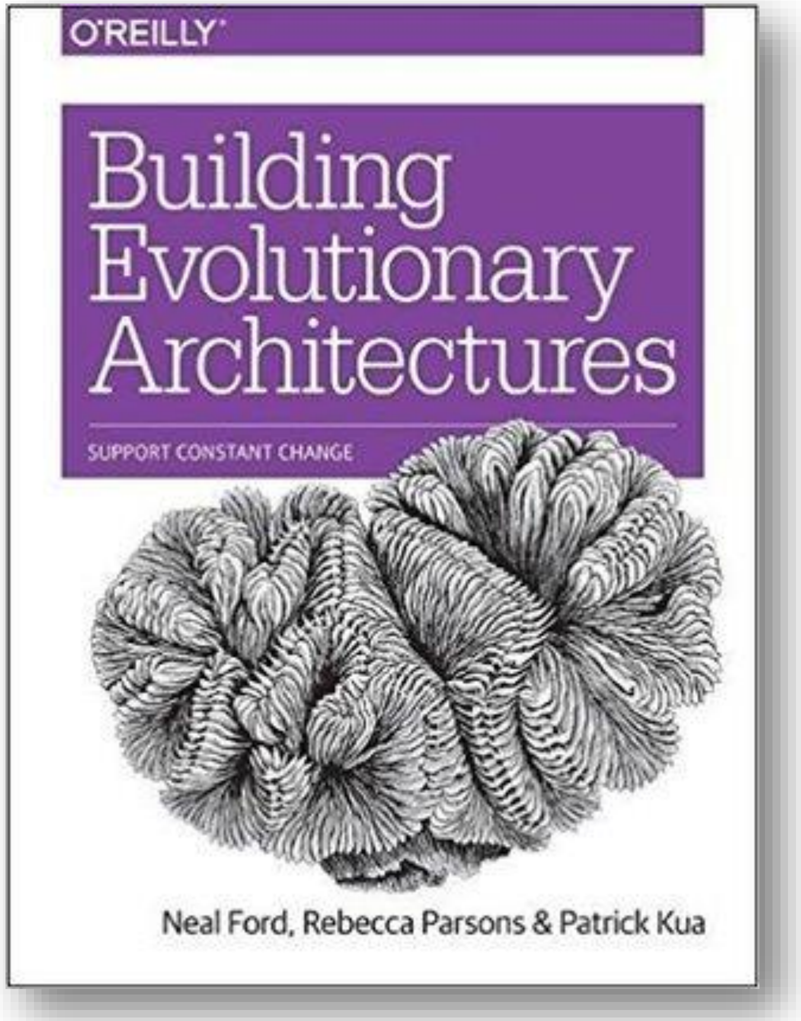
MUREX™

LUXOFT

Xavier RENÉ-CORAIL
@xcorail

Ionuţ BALOŞIN
@ionutbalosin

## INSPIRATIONS



O'REILLY

Building Evolutionary Architectures

SUPPORT CONSTANT CHANGE

Neal Ford, Rebecca Parsons & Patrick Kua

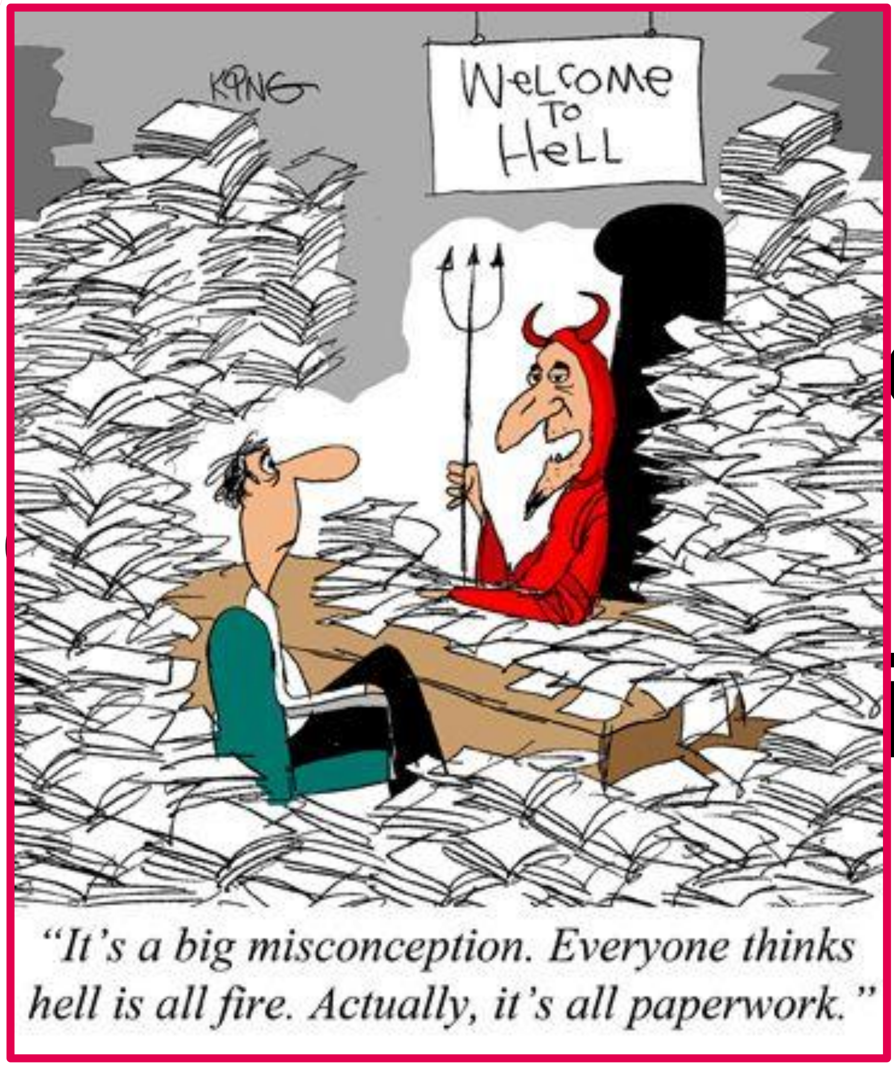
# AGILE & ARCHITECTURE

ENEMIES ? …

… OR FRIENDS ?

WORKING TOGETHER

## MISCONCEPTIONS

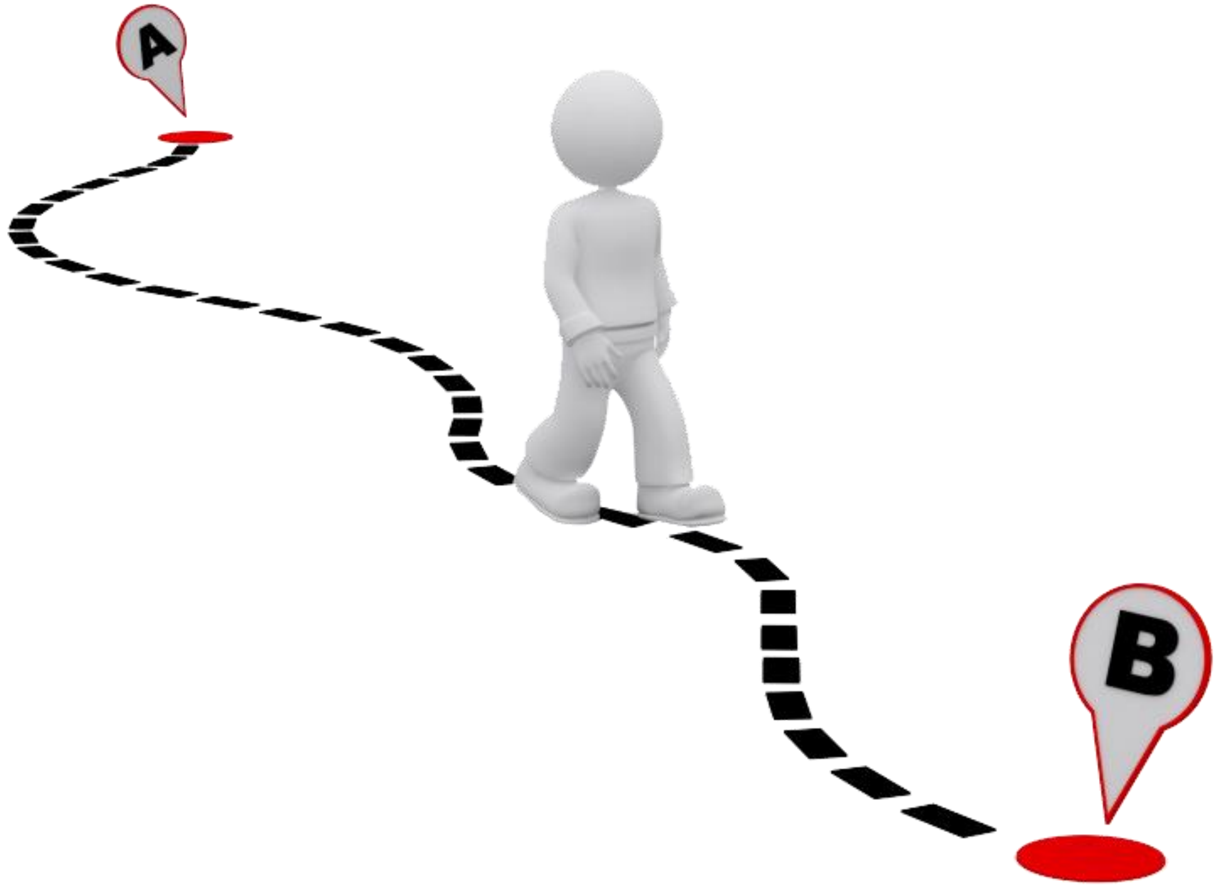


"It's a big misconception. Everyone thinks hell is all fire. Actually, it's all paperwork."

ome

ions

# RESPONDING TO CHANGE



# BE RIGHT THE 1ST TIME

**Is there one thing**

**that will never change?**

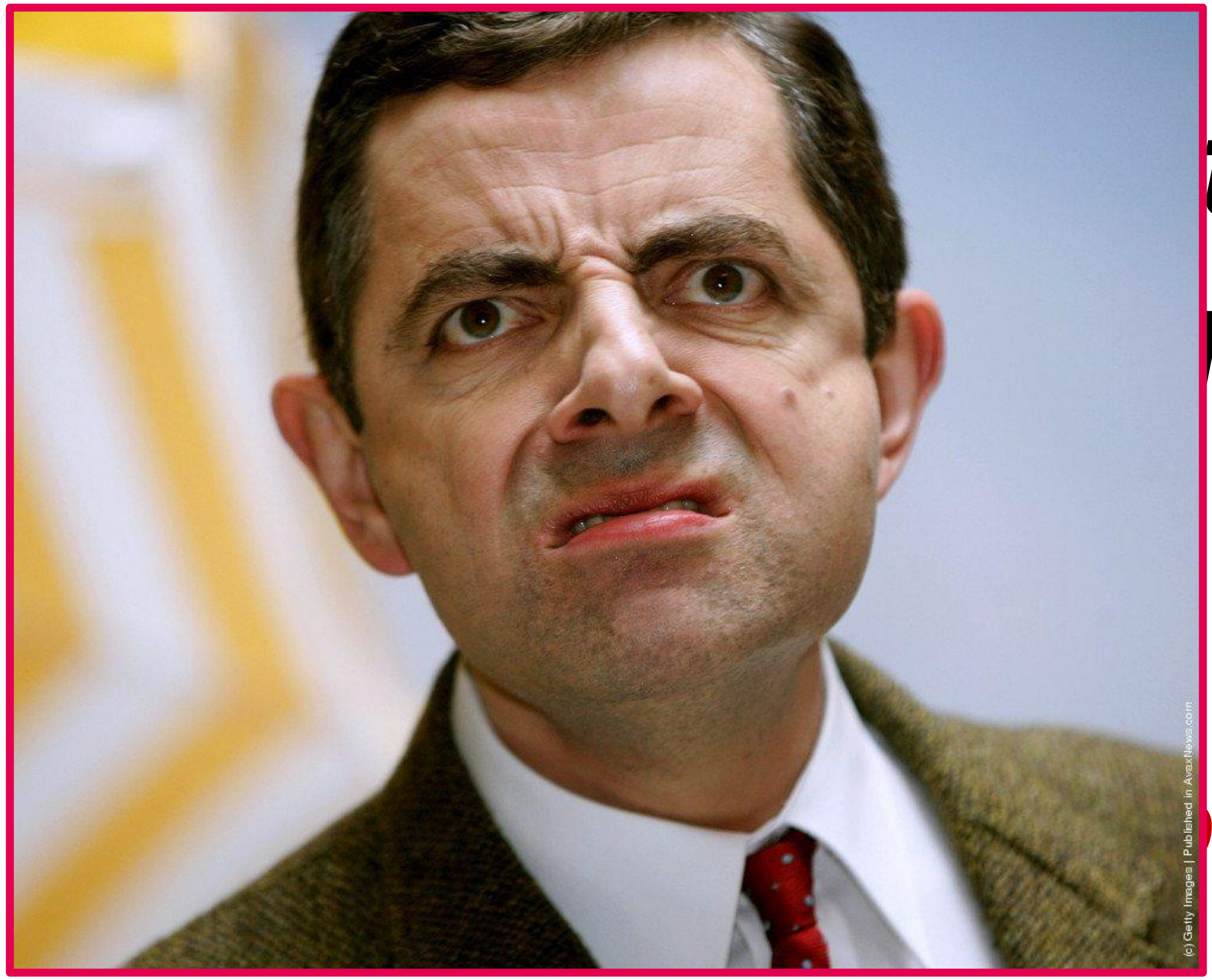
# AGILE & ARCHITECTURE

ENEMIES ? …

… OR FRIENDS ?

WORKING TOGETHER

## SOFTWARE ARCHITECTURE IS ...



*t*

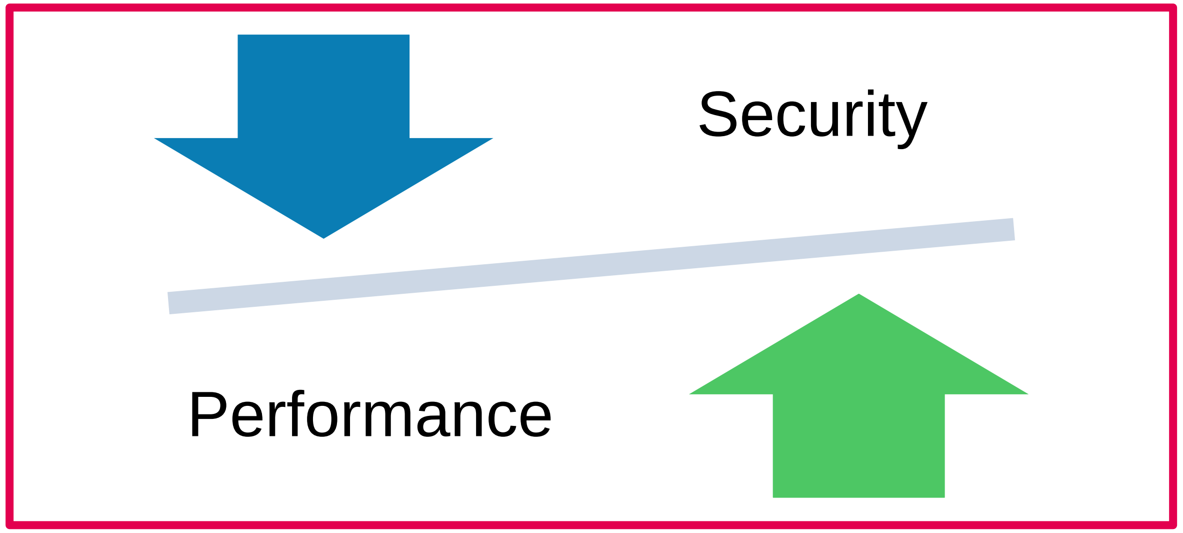
*r*

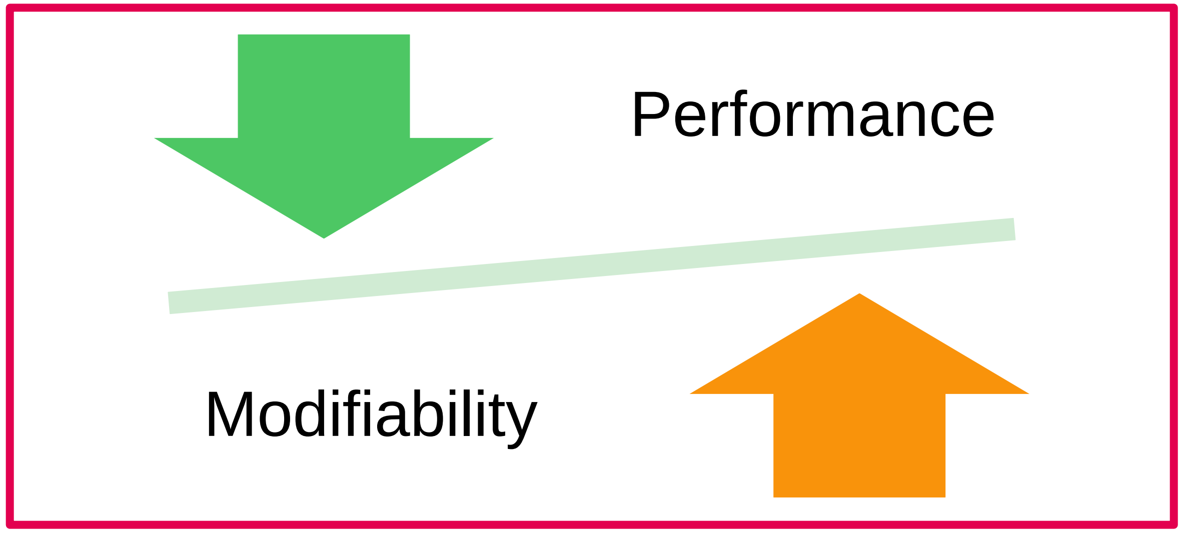
*n*

## WORLD OF ‘-ILITIES

upgradability
operability
reliability
discoverability
tailorability
flexibility
sustainability
scalability
autonomy
affordability
determinability
accountability
fault-tolerance
compatibility
recoverability
maintainability
transparency
portability
installability
repeatability
composability
accessibility
customizability
exnetsibility
relevance
security
agility
seamlessness
survivability
reusability
stability
adaptability
evolvability
safety
debugability
testability
modifiability
manageability
distributability
administrability
traceability
inspectability
configurability
performance
auditability
serviceability
understandability
security
availability
usability
interoperability
interchangeability
deployability
durability

# DIVERGENT '-ILITIES





## WE NEED TO ELICIT

## SOFTWARE ARCHITECTURE IS …

**STAKEHOLDER PRIORITISATION**

- ✓ AVAILABILITY
- ✓ SCALABILITY
- ✓ SECURITY
- ✓ PERFORMANCE
- ✓ …

***"The important stuff (whatever that is)"***

***Ralph Johnson***

**VITAL QUALITY ATTRIBUTES**

- ✓ MODIFIABILITY
- ✓ TESTABILITY
- ✓ FLEXIBILITY
- ✓ MAINTAINABILITY
- ✓ …

**EVOLVABILITY IS A SHARED KEY INGREDIENT**

# AGILITY IS ...

MANIFESTO

- ✓ ORGANISATION
- ✓ COLLABORATION
- ✓ PLANNING
- ✓ SOFTWARE

“The ability to move quickly and easily”

Oxford Dictionary

XP DESIGN

- ✓ MAINTAINABILITY
- ✓ TESTABILITY
- ✓ SIMPLICITY

**SOFTWARE EVOLVABILITY IS AT THE HEART OF AGILITY**

# AGILE & ARCHITECTURE

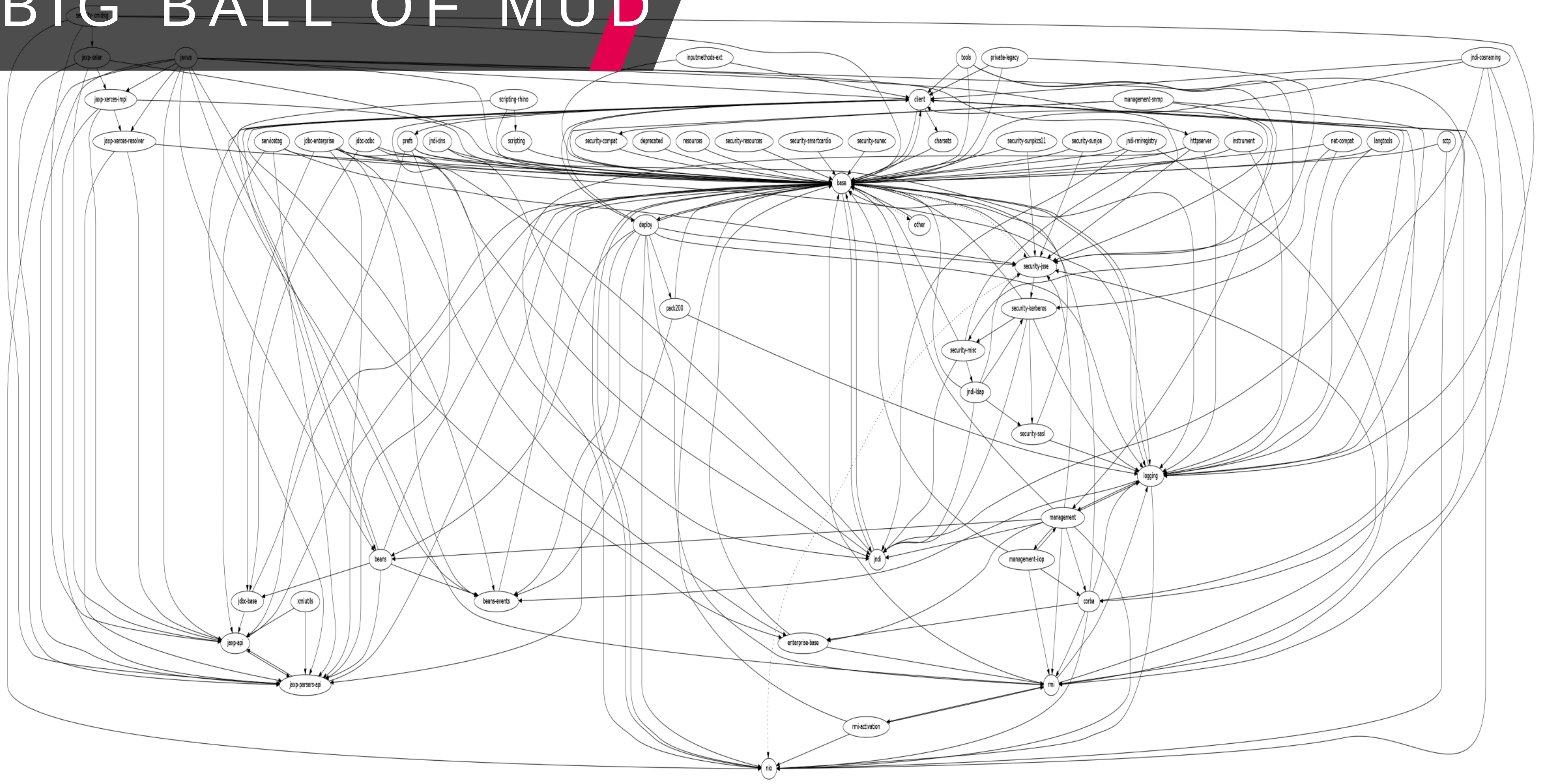
ENEMIES ? …

… OR FRIENDS ?

WORKING TOGETHER

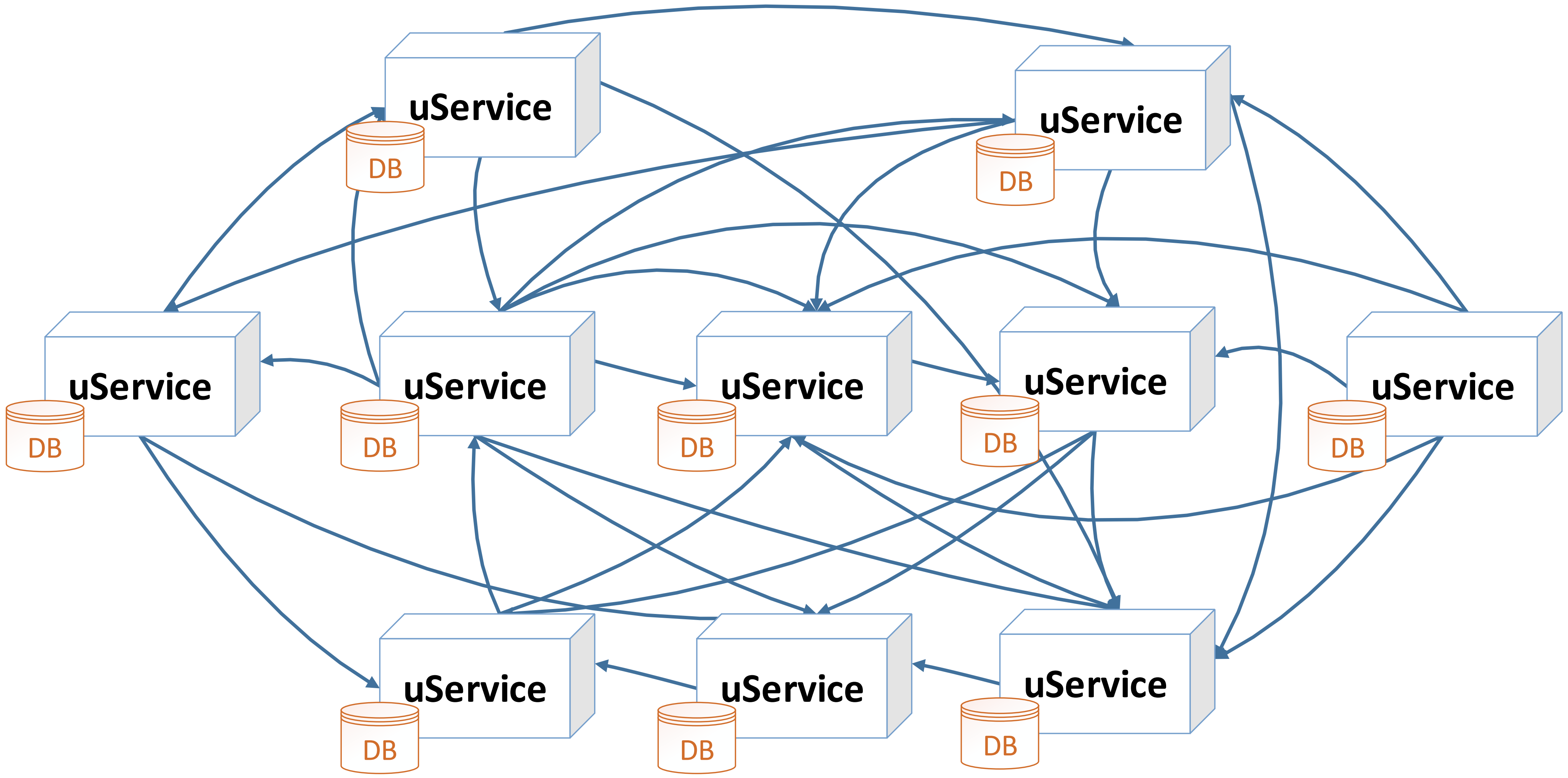
# NOT AGILE ARCHITECTURES

# BIG BALL OF MUD

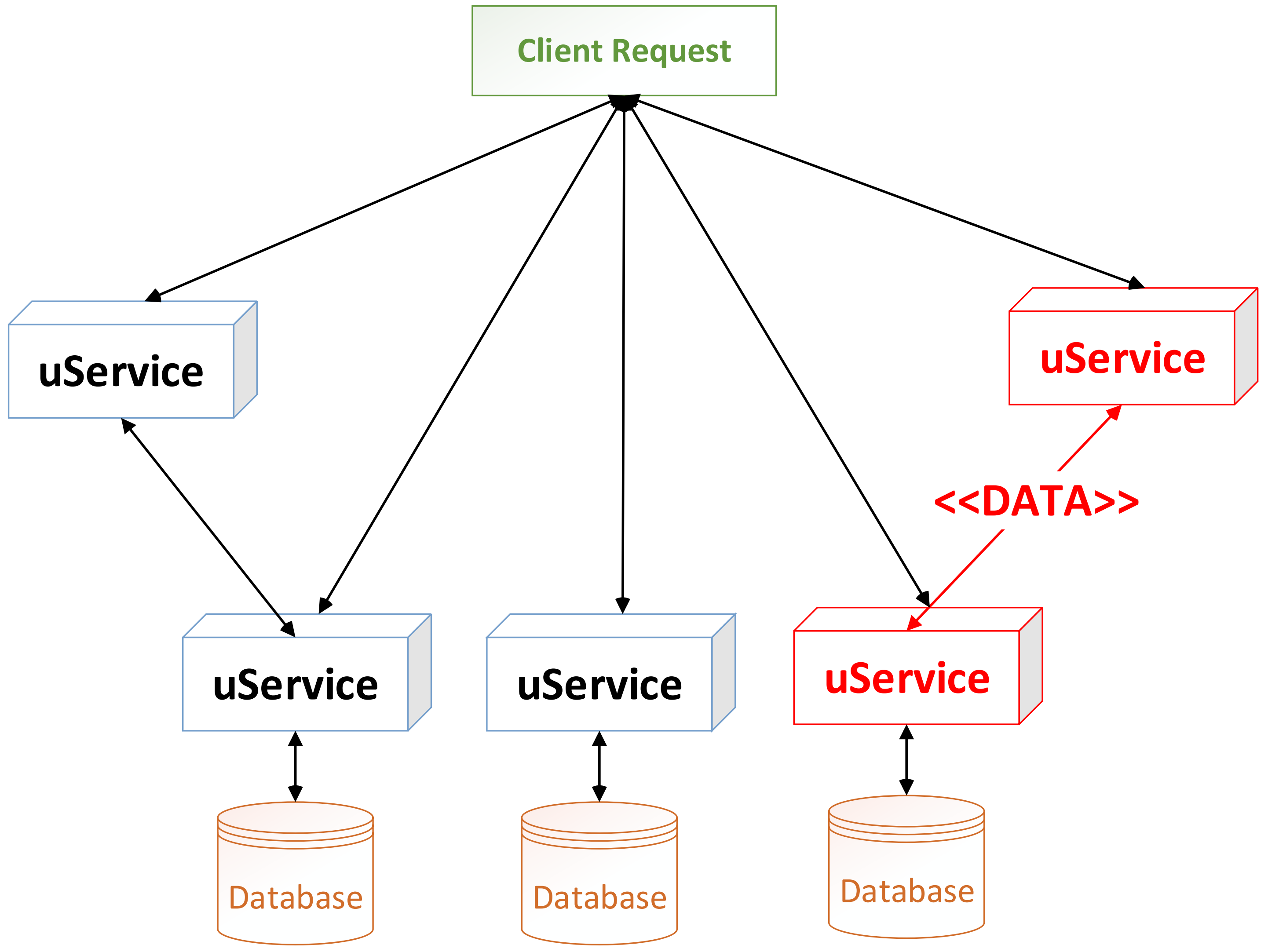


**Module graph for JDK 7 b65 rt.jar, mid 2009**

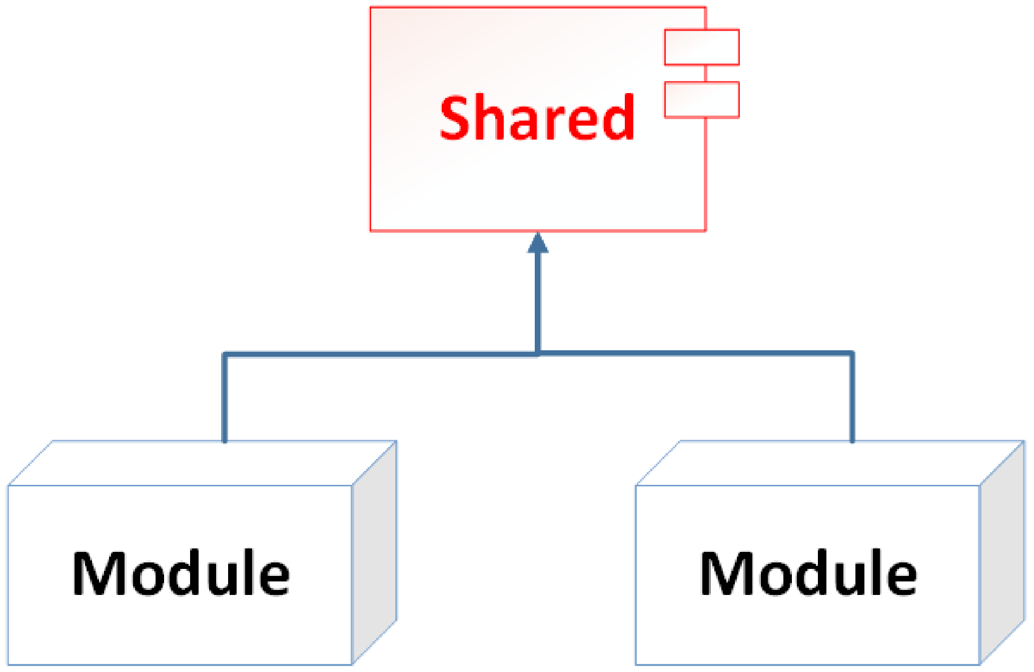
# SERVICES DEPENDENCY HELL



## DATA DEPENDENCY FLOWS

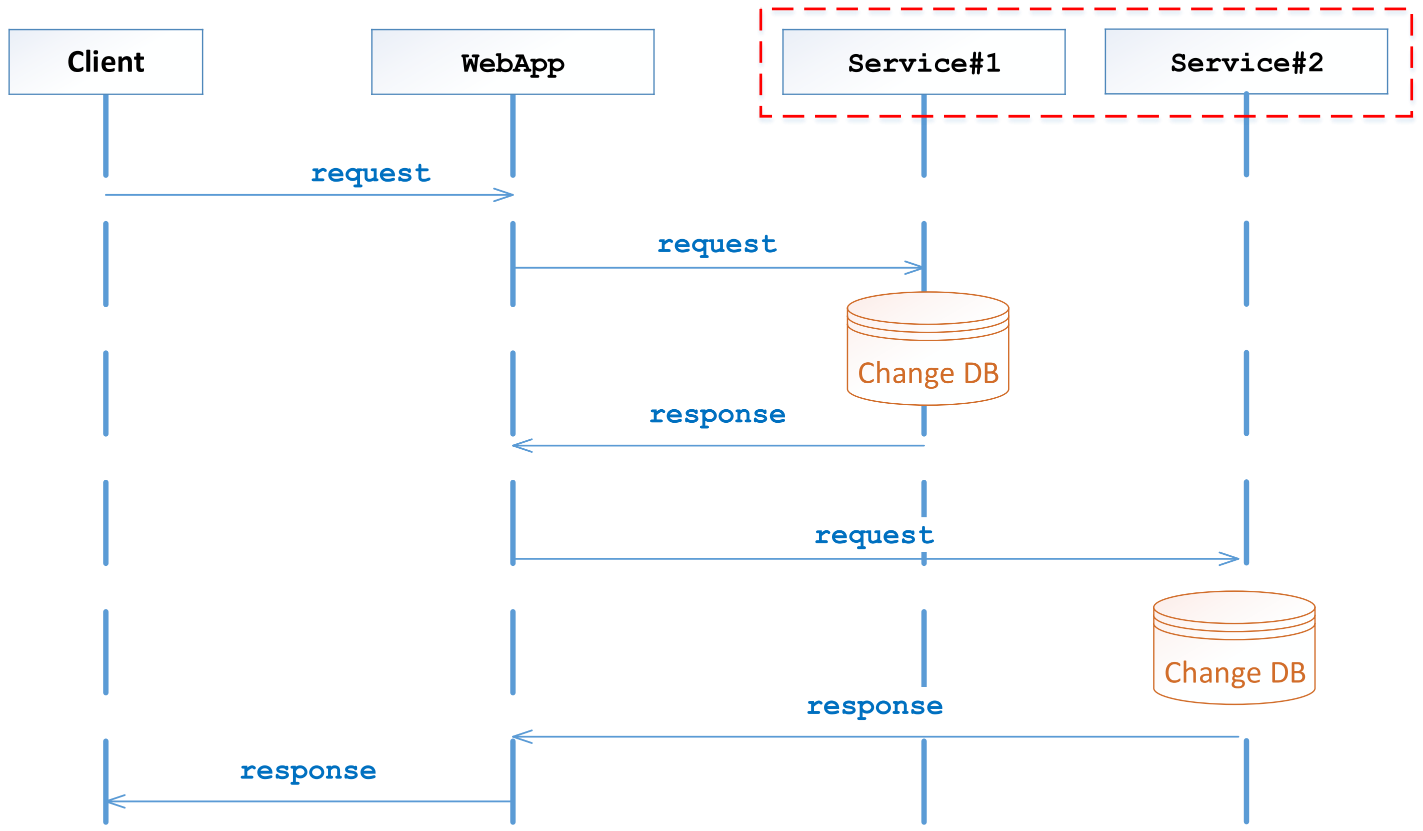


## SHARED PACKAGES



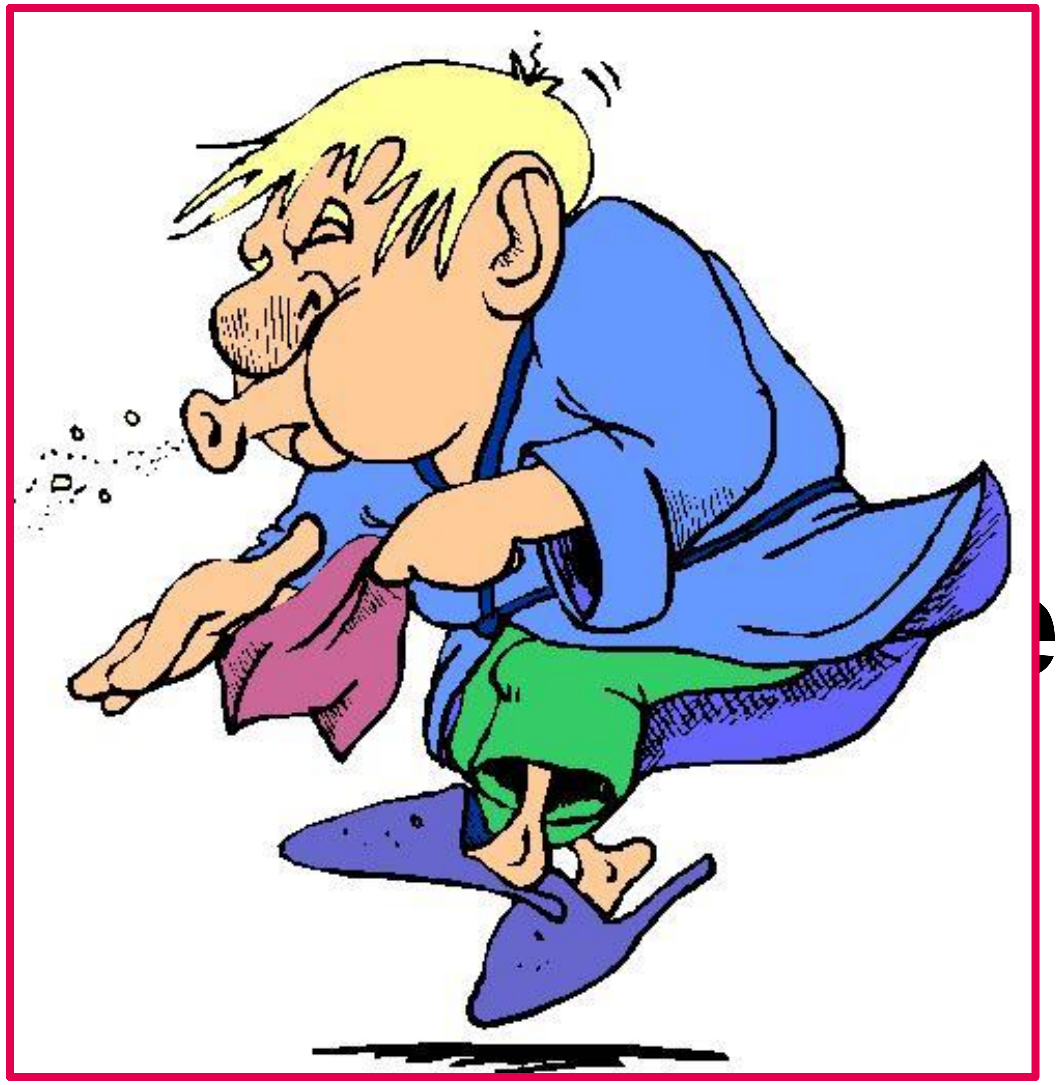
INCREASE COMPONENTS COUPLING

# DISTRIBUTED TRANSACTIONS



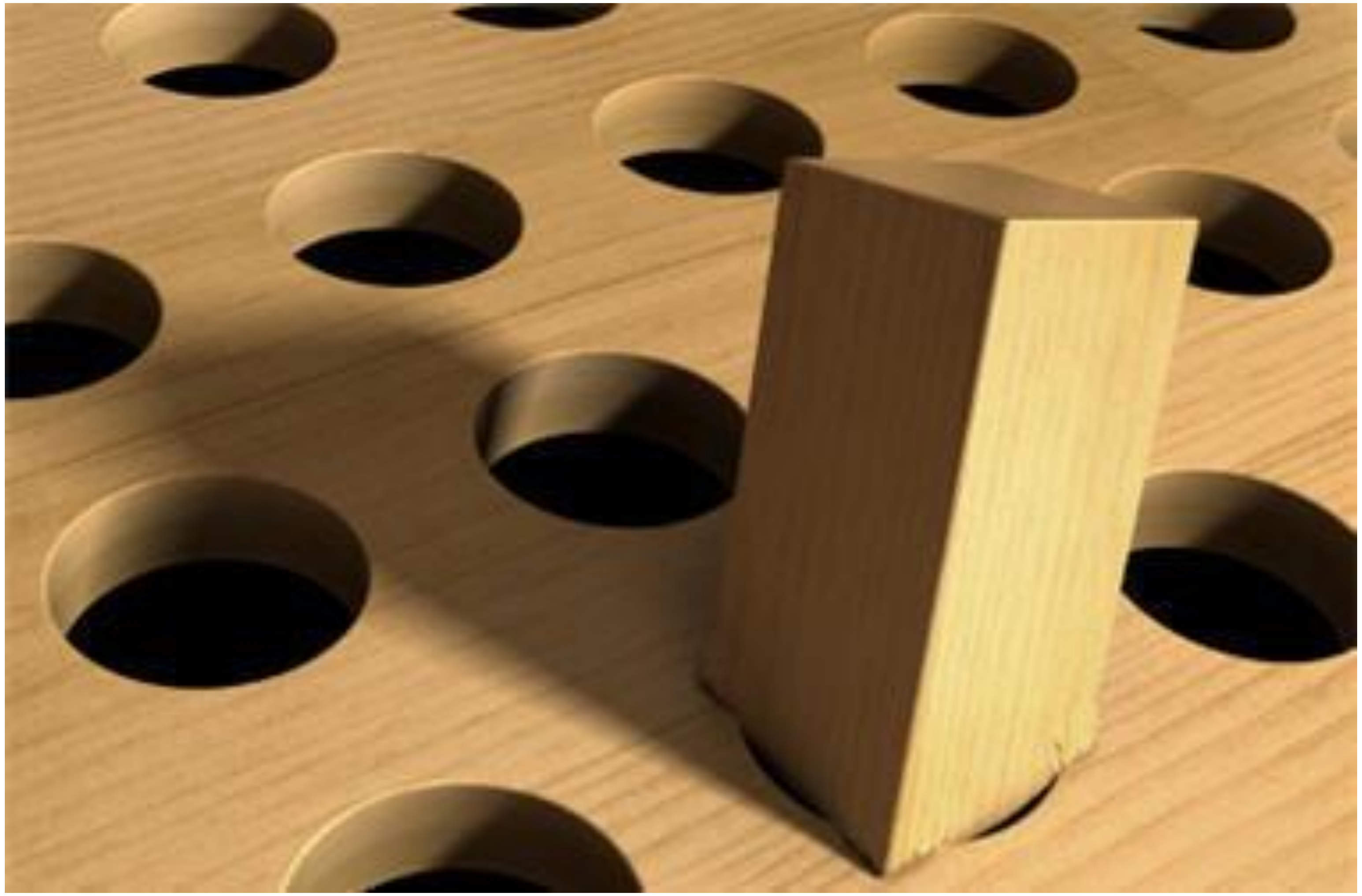
## INCREASE COMPONENTS COUPLING

## SYMPTOMS



es

## DIFFICULT TO INCORPORATE CHANGES



## SLOW RELEASE CYCLES

I RUN
I'M SLOWER THAN A HERD OF TURTLES STAMPEDING THROUGH PEANUT BUTTER
BUT I RUN

## DIFFICULT TO CARRY OUT

SUCCESSFUL RECIPE

## WHAT ARE THE INGREDIENTS FOR ACHIEVING AGILE ARCHITECTURES ?

## DESIGN PRINCIPLES

### KEEP IN MIND DESIGN PRINCIPLES

Do not sacrifice **loose-coupling** for performance

Prefer **composition** over inheritance

Design **messages atomic** and **services stateless**

Design remote interfaces **coarse-grained**

**High cohesion** inside modules, components

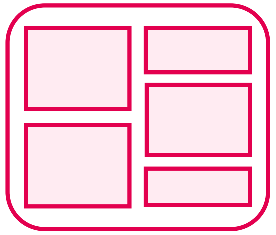
**SOLID**

**DRY**

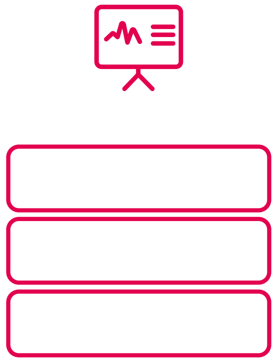
**YAGNI**

# ARCHITECTURAL STYLES

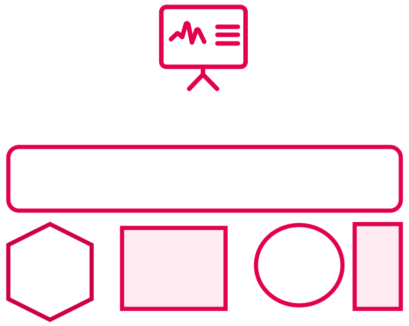
MONOLITH



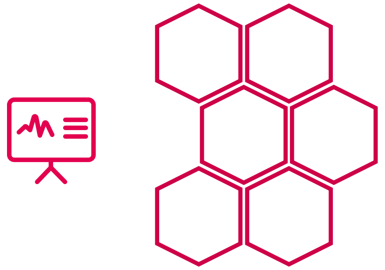
LAYERED



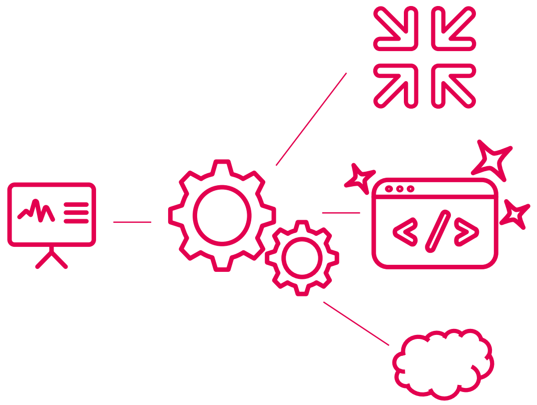
SOA



uSERVICES



SERVERLESS



# COMPONENT BOUNDARIES

COMPONENT

**communication mechanisms with external**

- protocols
- synchronisation type
  - synchronous
  - asynchronous
- transmission way
  - one way
  - bi-directional

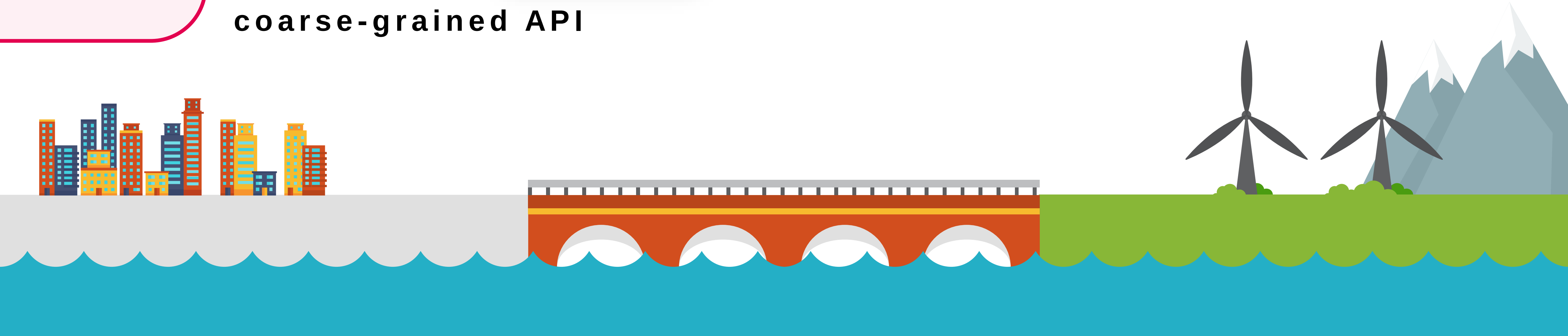
**messages exchanged**

- type
- format

**dependencies**

- depends on
- used by

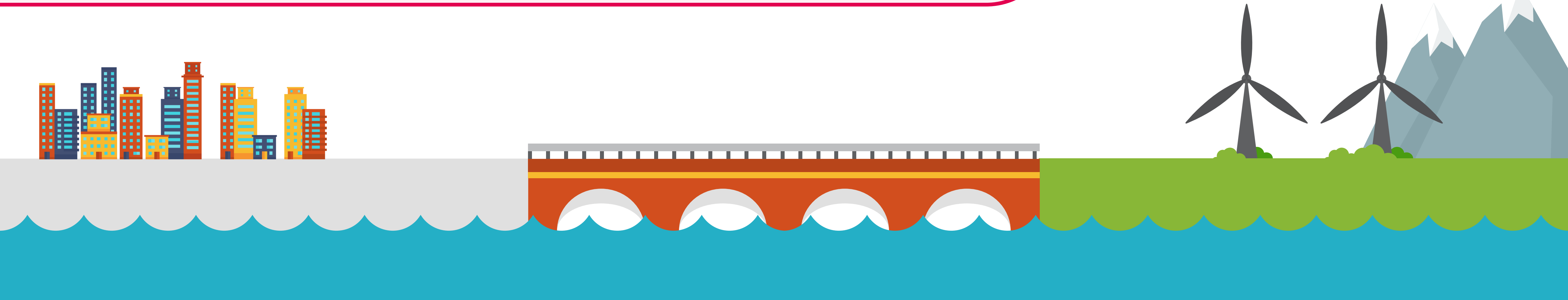
**coarse-grained API**



## COMPONENT INTERNALS

COMPONENT

HIGH COHESION
LOOSE COUPLING
ENCAPSULATION
SOLID
DRY
YAGNI



# OTHER PERSPECTIVES

## MANAGEMENT OF RESOURCES

Time awareness

Threading model

Scheduling strategies

Resource limits / saturation

## OBJECT AND DATA MODEL

Data models

Data entities access levels

Criteria for data retention / preservation

## BINDING TIME DECISIONS

Compile time

Build time

Load time

Run time

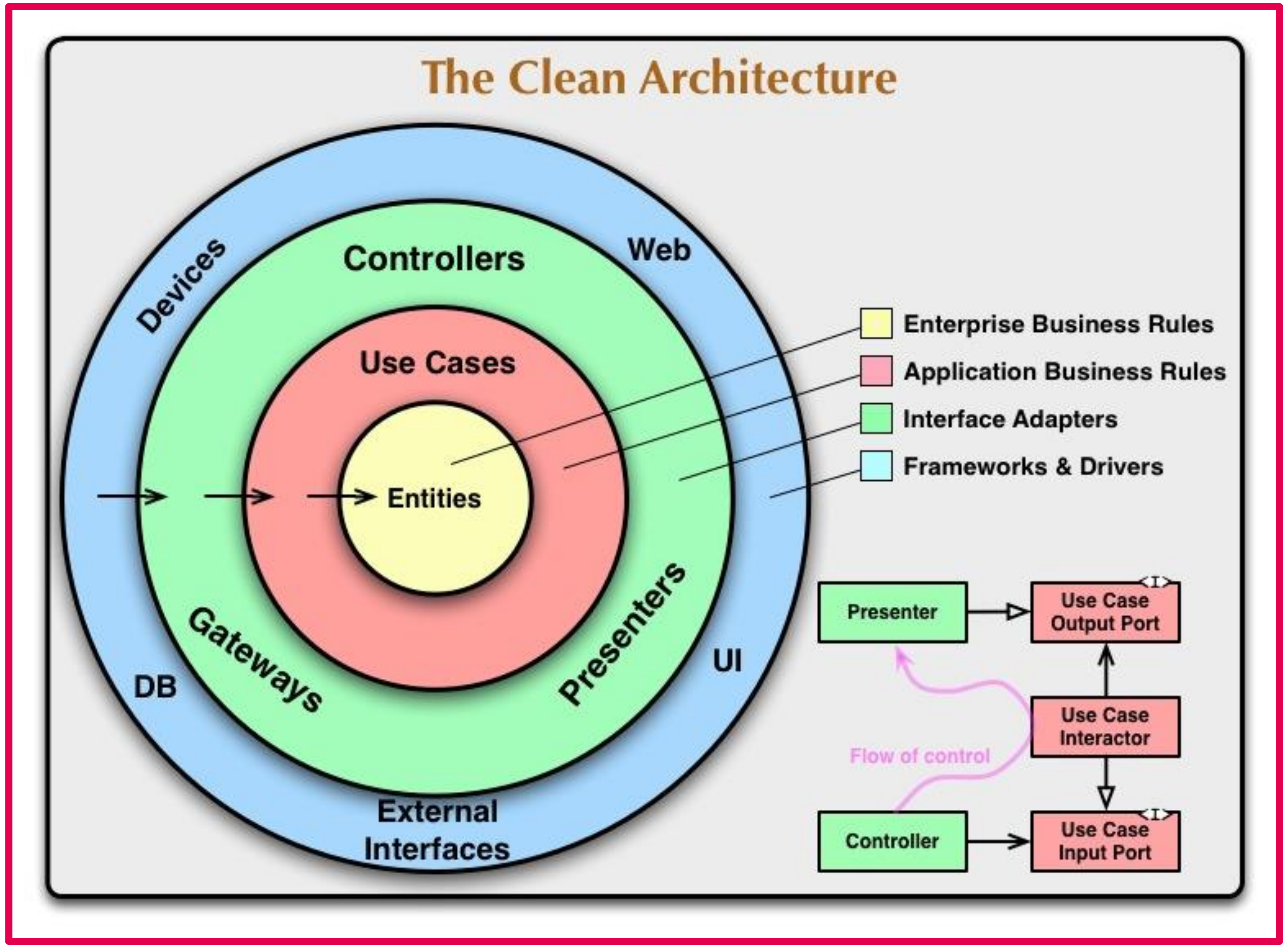
## PAY ATTENTION

PROGRAMMING LANGUAGES

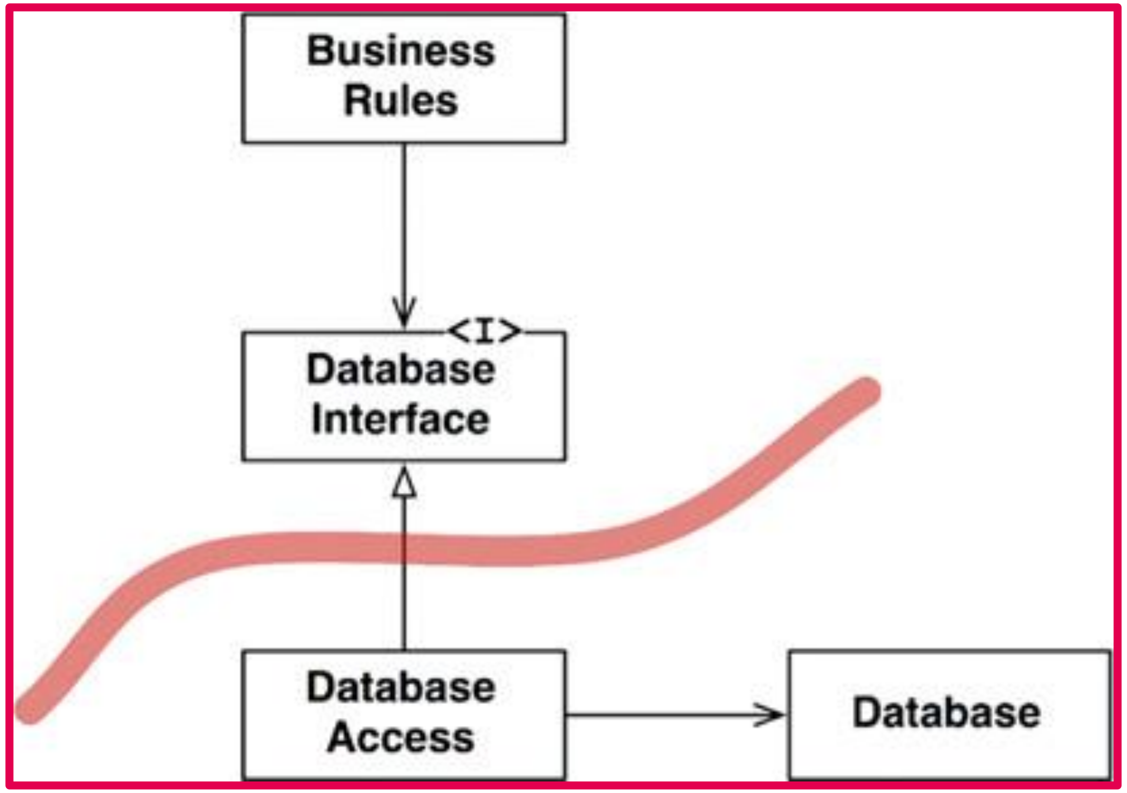
FRAMEWORKS

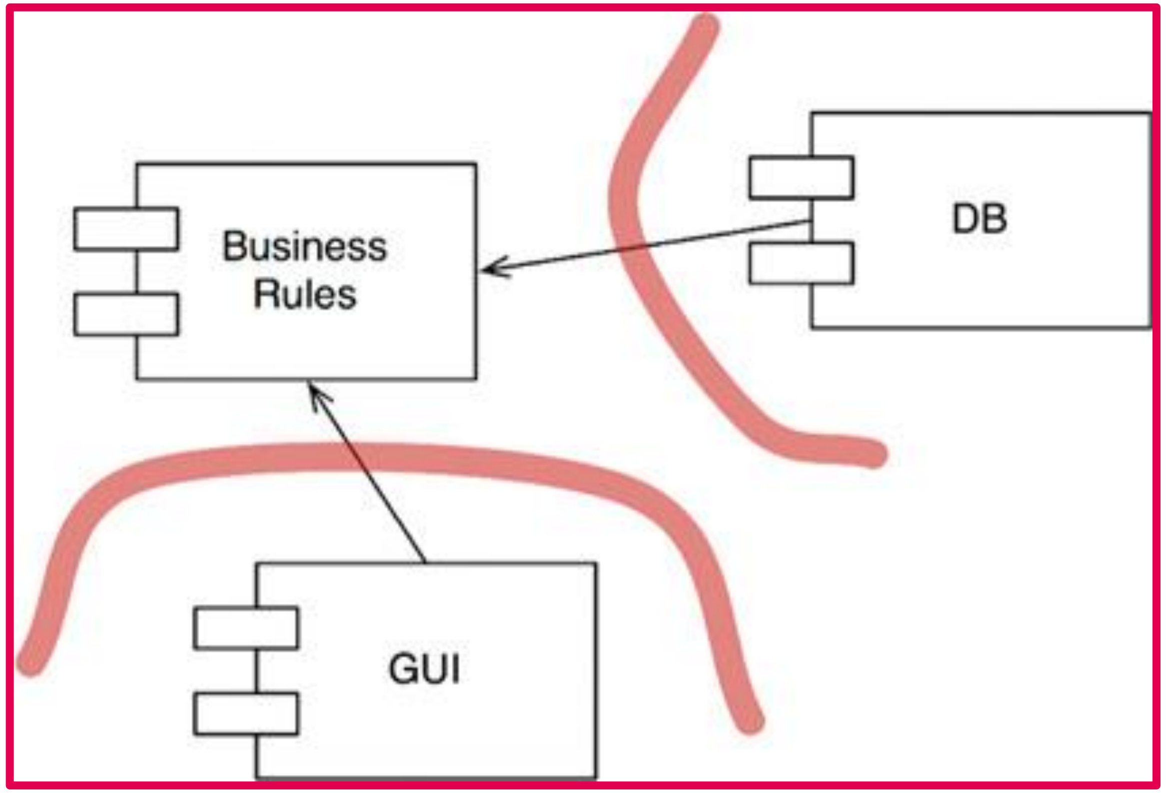
**THESE MIGHT BECOME IREVERSIBLE DECISSIONS**

## THE DEPENDENCY RULE

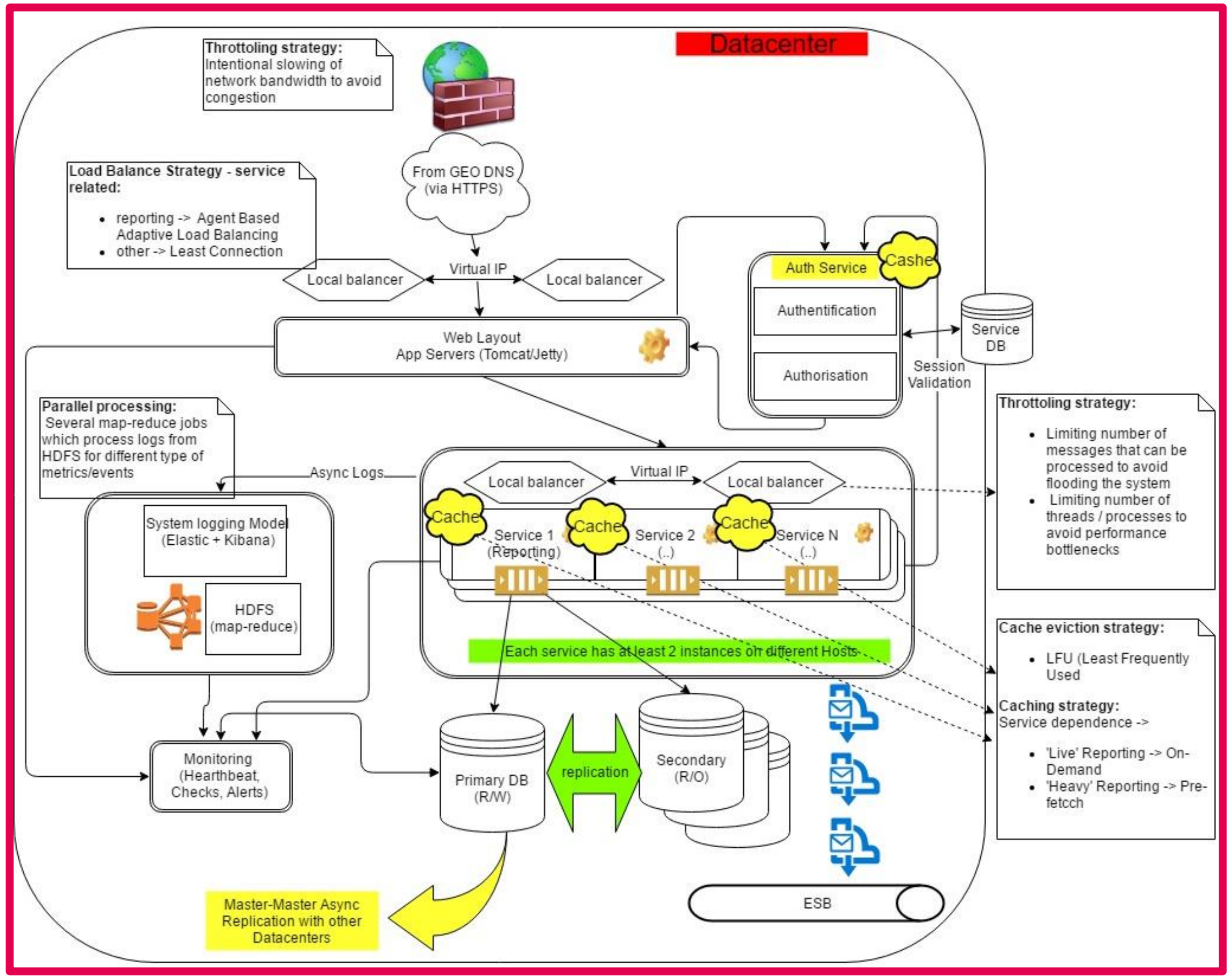


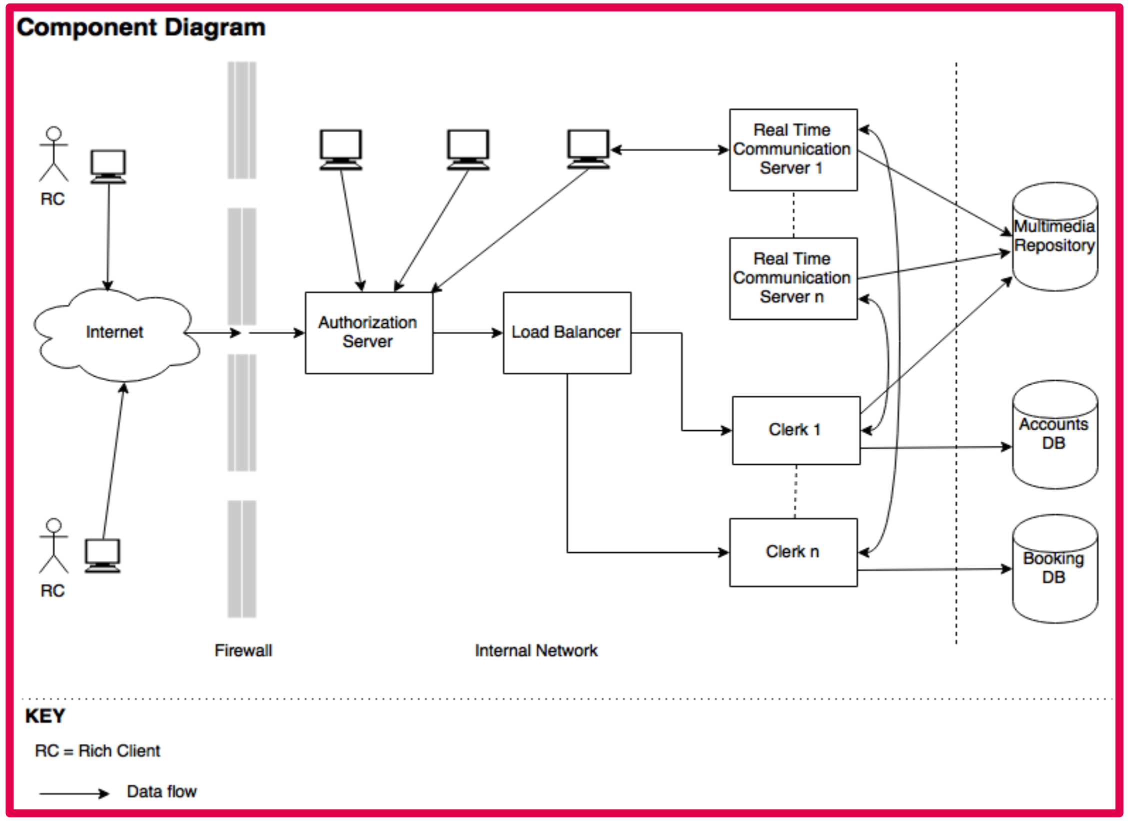
## PROPER SERVICES DEPENDENCIES





# WHICH ONE IS AN AGILE ARCHITECTURE ?


Datacenter
Throttoling strategy:
Intentional slowing of network bandwidth to avoid congestion
Load Balance Strategy - service related:
• reporting -> Agent Based Adaptive Load Balancing
• other -> Least Connection
From GEO DNS (via HTTPS)
Virtual IP
Local balancer
Local balancer
Web Layout
App Servers (Tomcat/Jetty)
Auth Service
Cashe
Authentification
Authorisation
Service DB
Session Validation
Parallel processing:
Several map-reduce jobs which process logs from HDFS for different type of metrics/events
Async Logs
System logging Model (Elastic + Kibana)
HDFS (map-reduce)
Local balancer
Virtual IP
Local balancer
Cache
Service 1 (Reporting)
Cache
Service 2 (..)
Cache
Service N (..)
Each service has at least 2 instances on different Hosts
Throttoling strategy:
• Limiting number of messages that can be processed to avoid flooding the system
• Limiting number of threads / processes to avoid performance bottlenecks
Cache eviction strategy:
• LFU (Least Frequently Used
Caching strategy:
Service dependence ->
• 'Live' Reporting -> On-Demand
• 'Heavy' Reporting -> Pre-fetch
Monitoring (Hearthbeat, Checks, Alerts)
Primary DB (R/W)
replication
Secondary (R/O)
ESB
Master-Master Async Replication with other Datacenters



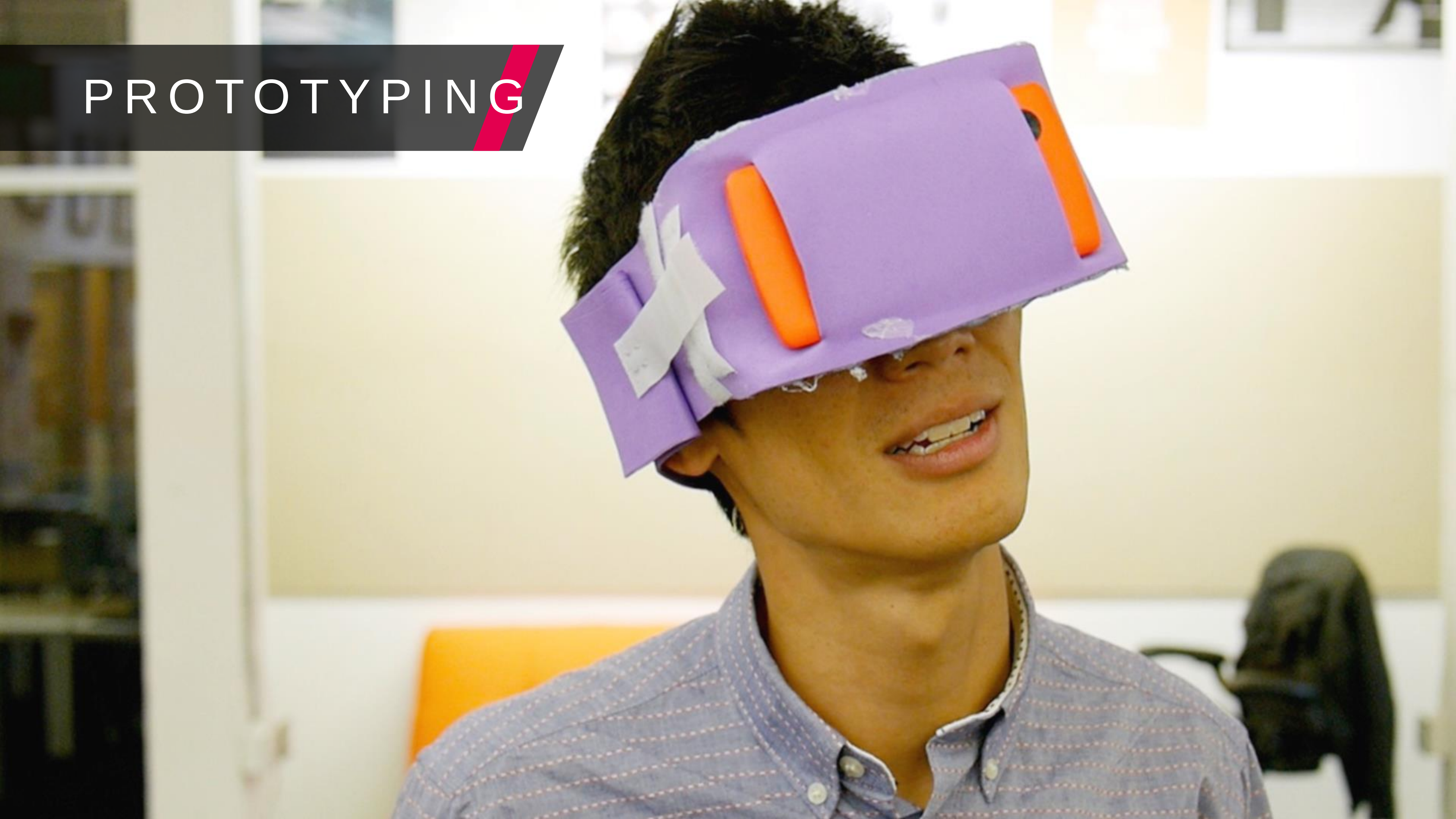
GOOD ARCHITECTURES ENABLES AGILITY

## AGILITY ENABLES ARCHITECTURE DESIGN

## ESSENTIALS FOR THE TRIP

**W op?**
**A er!**

## PROTOTYPING



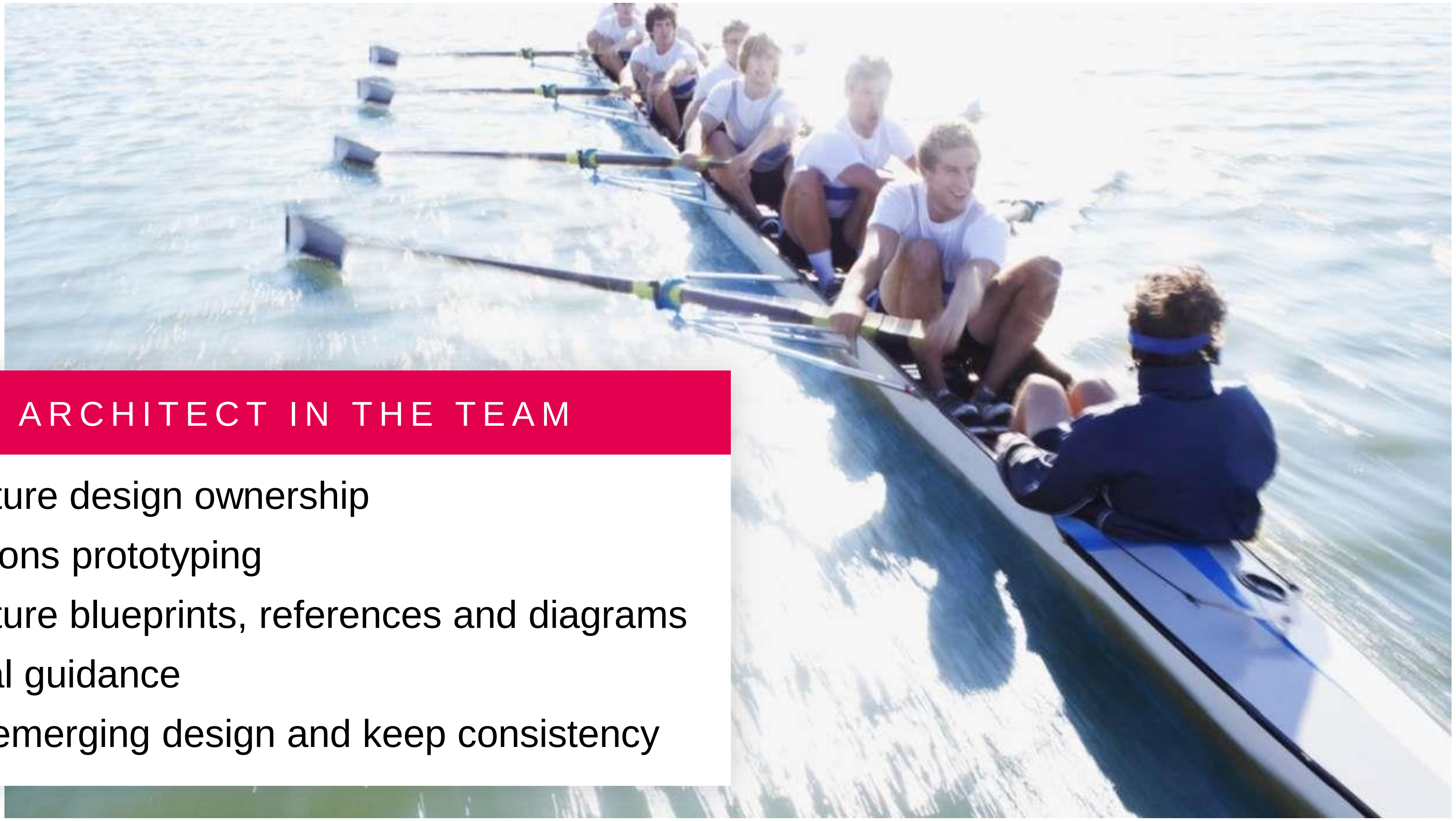
## SAFE TO FAIL

- ✓Fitness Functions
- ✓Feature Toggles
- ✓Green / Blue Deployments

## CONTINUOUS DEPLOYMENT

## ASTRONAUT ARCHITECT

# ALIGNMENT & AUTONOMY



## THE ARCHITECT IN THE TEAM

- ✓ Architecture design ownership
- ✓ Applications prototyping
- ✓ Architecture blueprints, references and diagrams
- ✓ Technical guidance
- ✓ Review emerging design and keep consistency

# BLENDING AGILE AND ARCHITECTURE

# REAL CHALLENGES

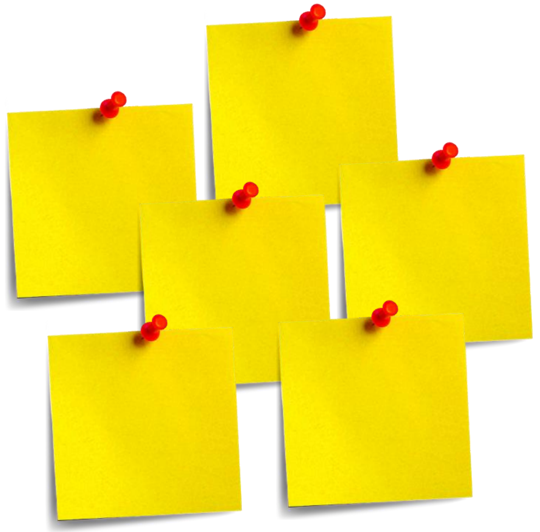
**ARCHITECTURE ~ INTERNAL QUALITY → no visible external value added for Customer**

**Backlog is mainly driven by "Customer value" → architectural activities are not given enough attention**

**You, as ARCHITECT, ensure to make this valuable !**

# BACKLOG INGREDIENTS

Functionality



BACKLOG

Ideas

Evaluation Feedback
Technical Debt

Constraints
Life Cycle Requirements

Architecturally Significant Requirements

# QUOTA RULE

70

FUNCTIONALITY

30

ARCHITECTURAL + OTHER TECHNICAL

WORKING TOGETHER BY MUTUAL AGREEMENTS ☺

# THANK YOU!

@ionutbalosin

@xcorail

@Work at Murex

@Luxoft

https://www.linkedin.com/company/luxoft

https://fr.linkedin.com/company/murex