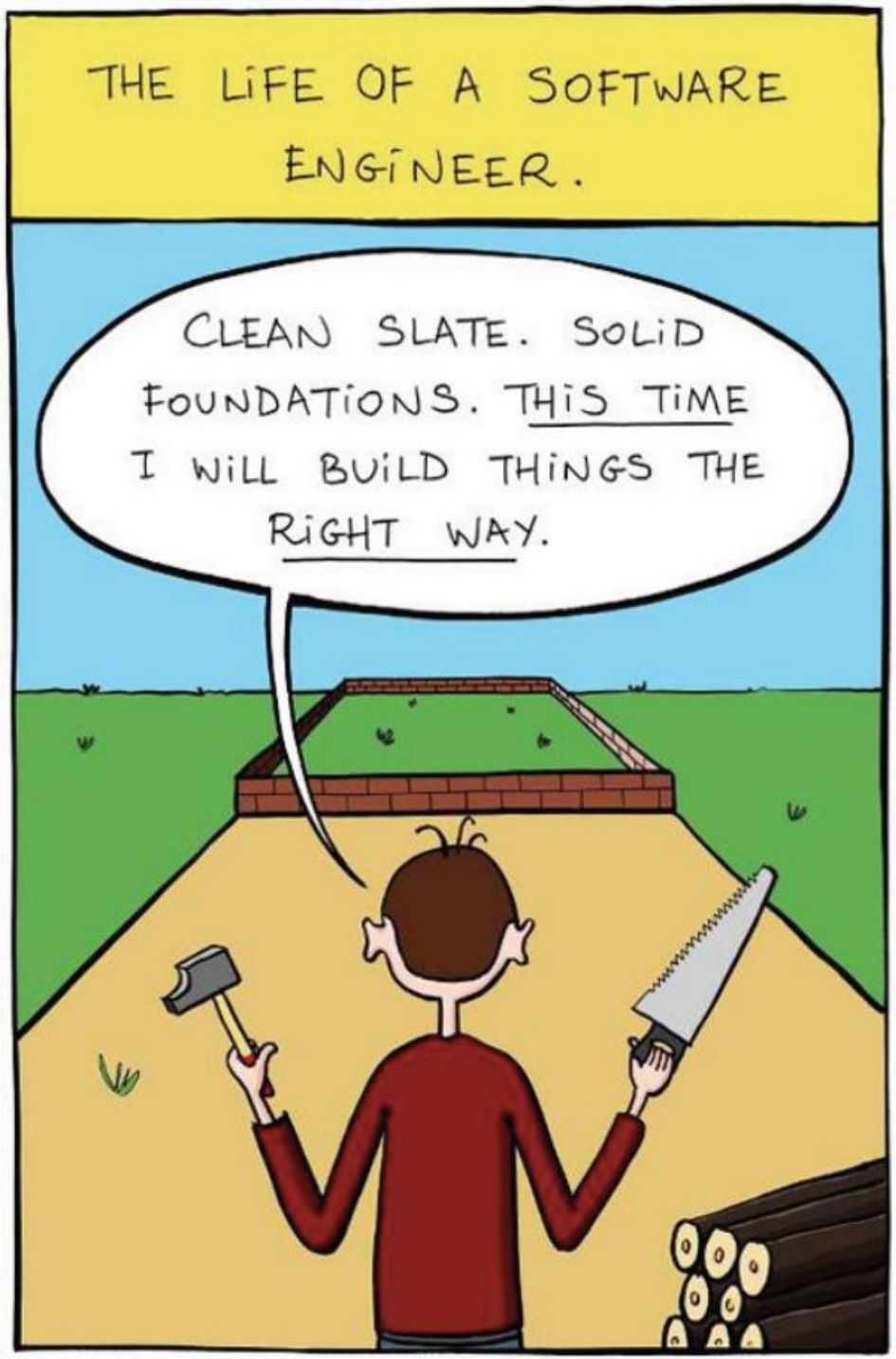
Microservices

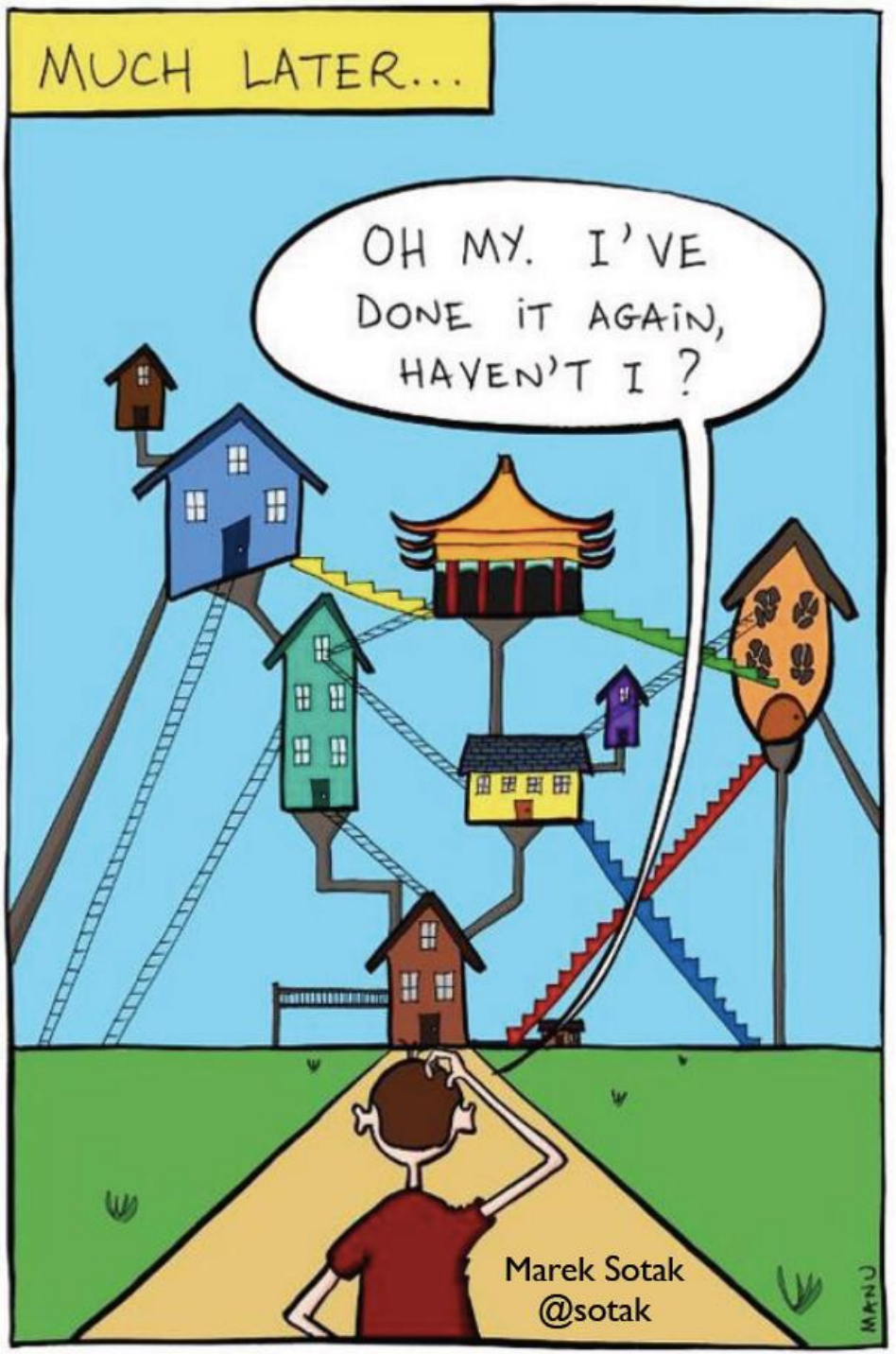
Desarrollo de Aplicaciones Cliente-Servidor

Dr. Jorge E. Villaverde

Agenda

- What are micro-services?
- Why do we want them? Or maybe not..
- What are the key challenges?

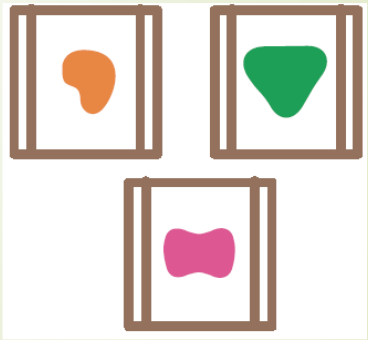

THE LIFE OF A SOFTWARE ENGINEER.
CLEAN SLATE. SOLID FOUNDATIONS. THIS TIME I WILL BUILD THINGS THE RIGHT WAY.


MUCH LATER...
OH MY. I'VE DONE IT AGAIN, HAVEN'T I ?
Marek Sotak
@sotak
MANU

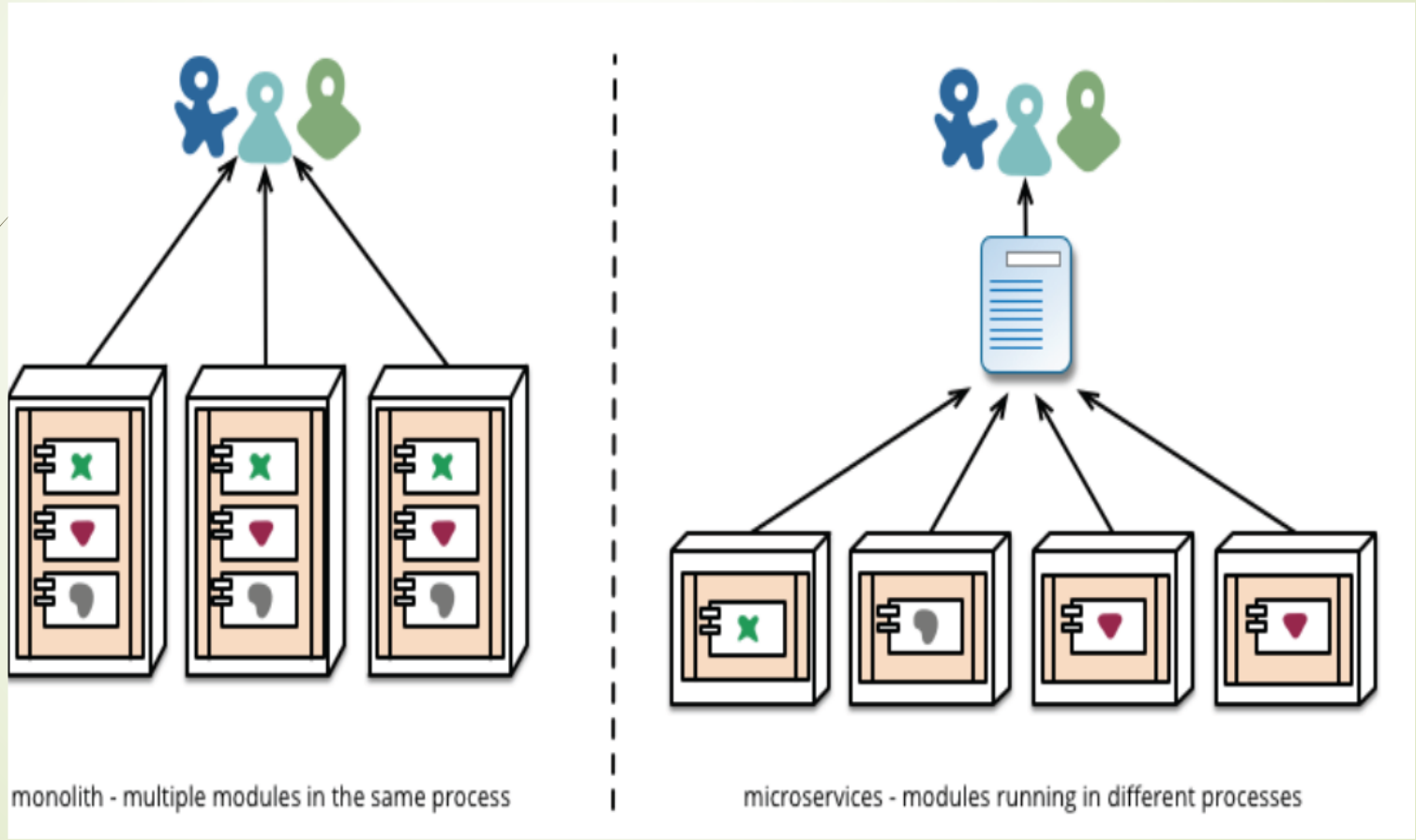
What's a microservice?

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

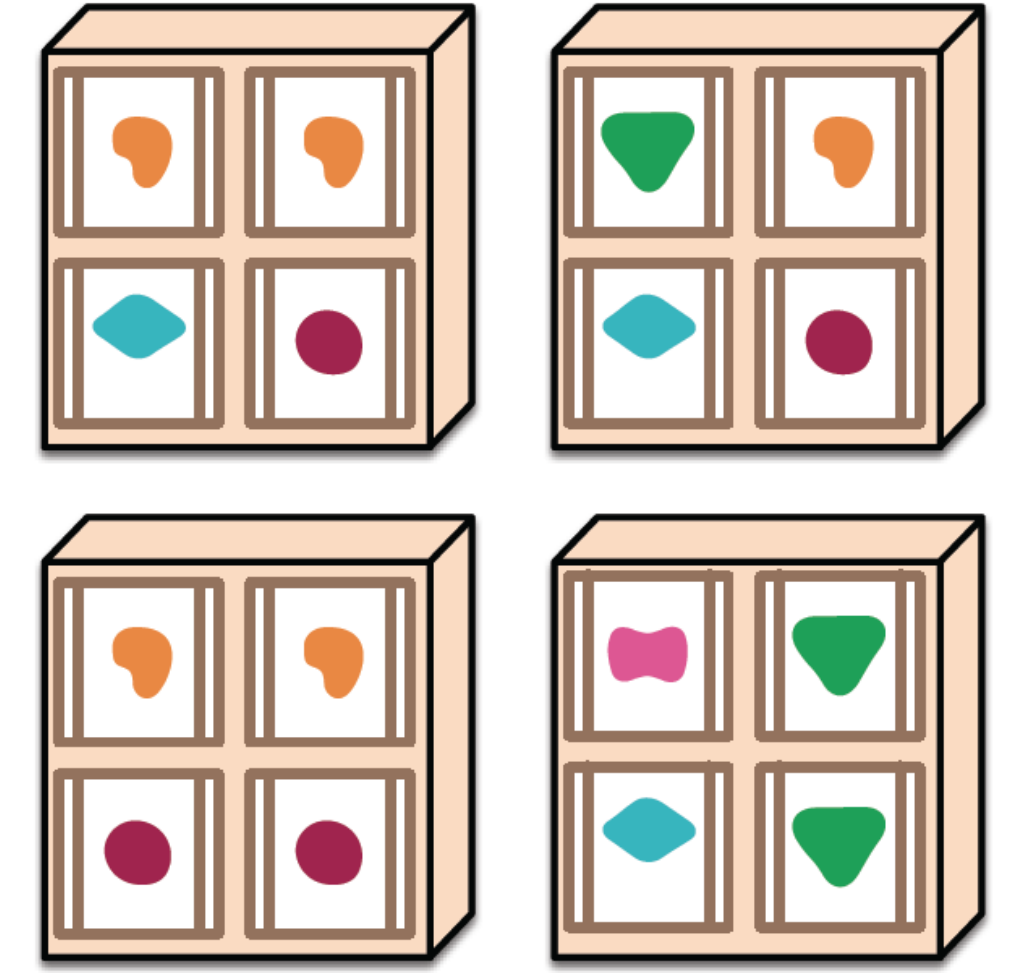
A microservices architecture puts each element of functionality into a separate service…



Independent Processes

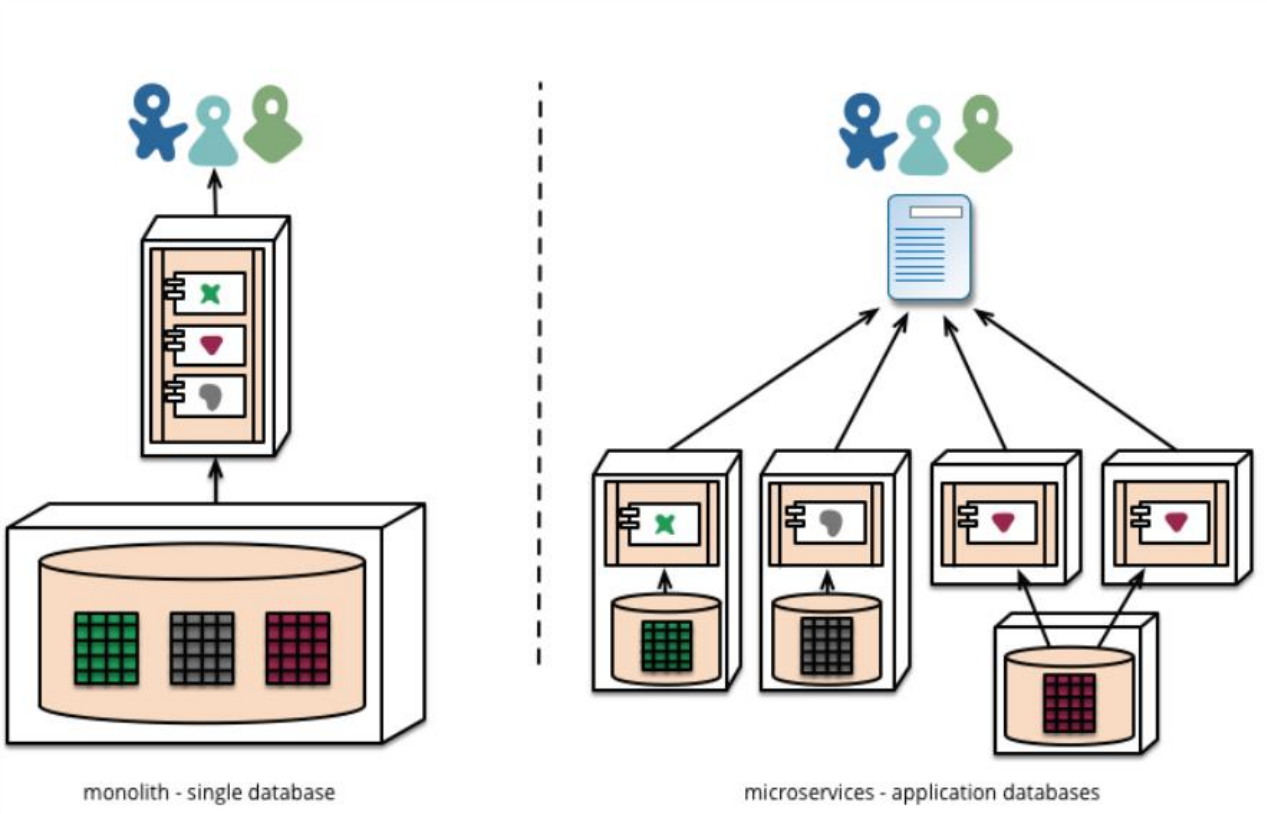


DECOUPLED

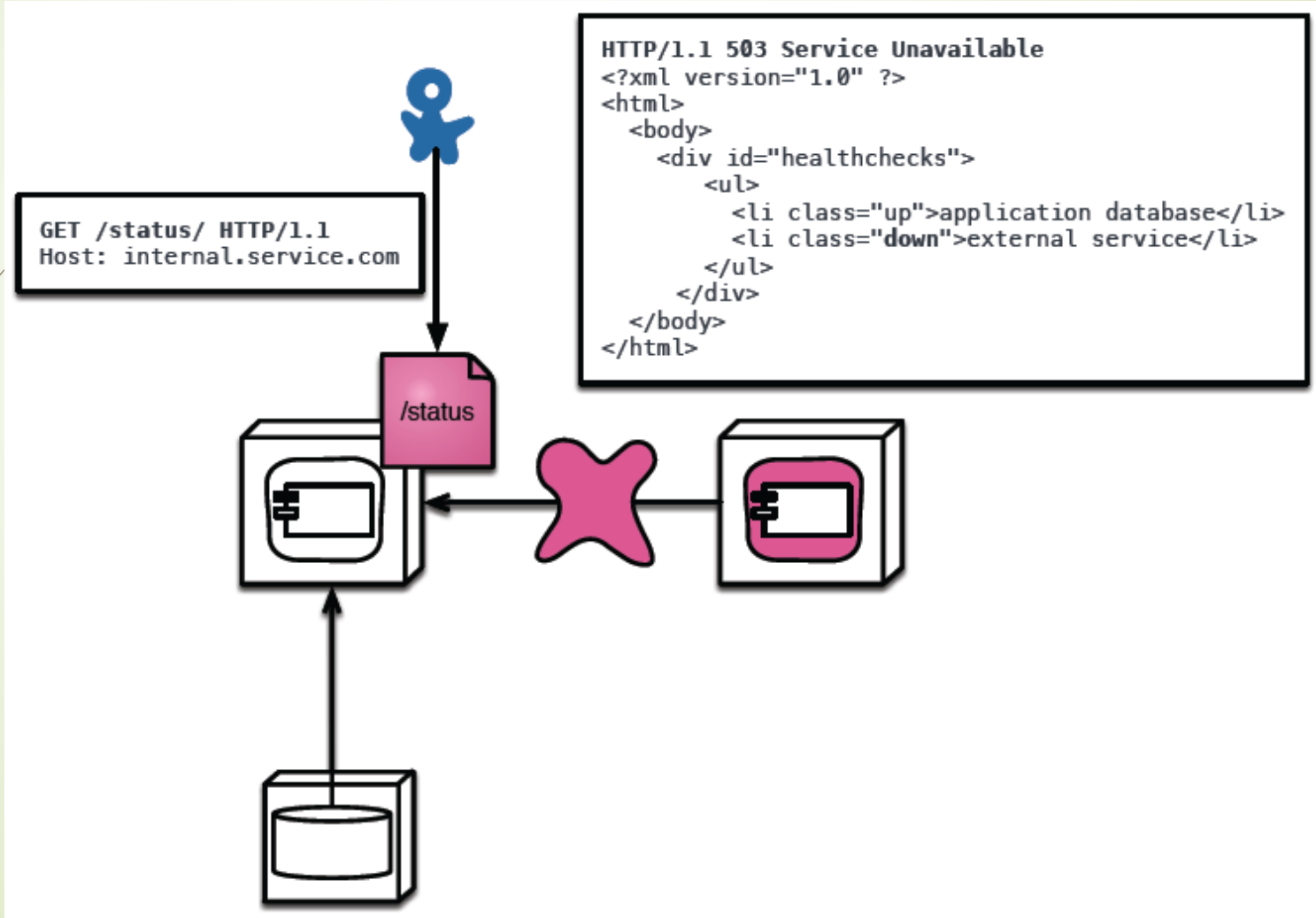


Why do we want them? Or maybe not..

- The **right tool** for the right job



RESILIENCE


GET /status/ HTTP/1.1
Host: internal.service.com
/status
HTTP/1.1 503 Service Unavailable
<?xml version="1.0" ?>
<html>
<body>

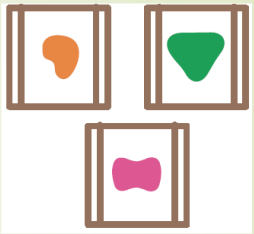
<li class="up">application database
<li class="down">external service

</body>
</html>

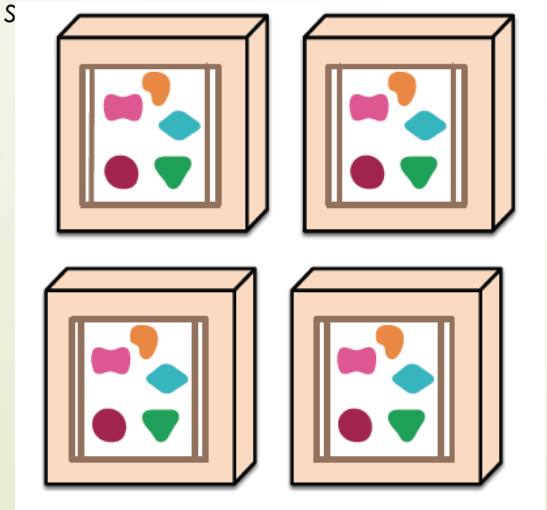
SCALING

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

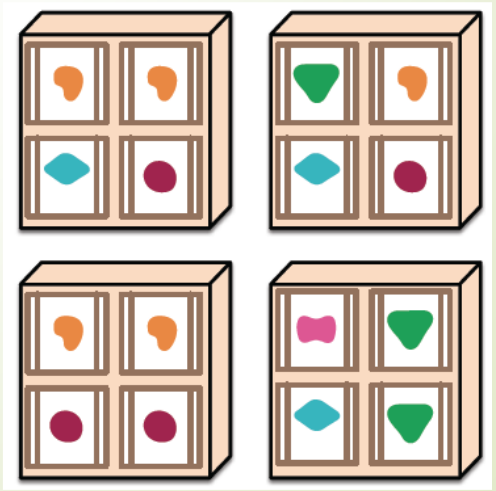
A microservices architecture puts each element of functionality into a separate service…



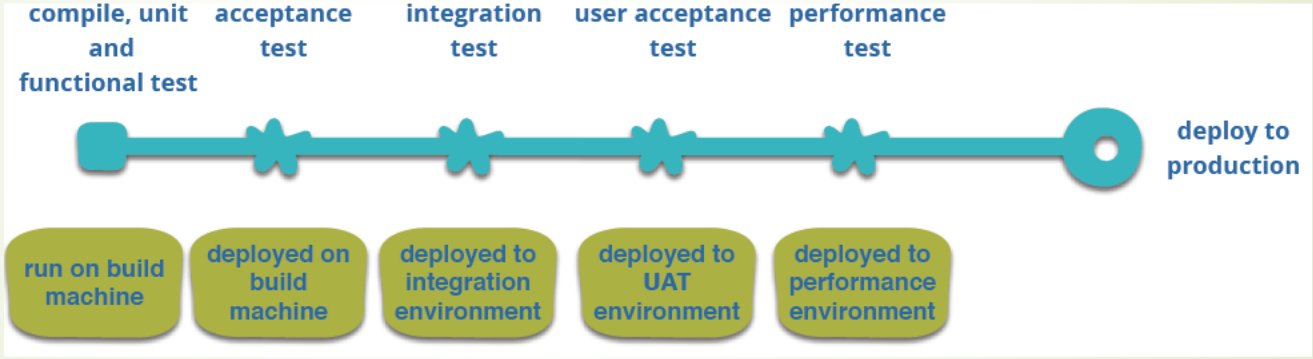
… and scales by replicating the monolith on multiple s



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



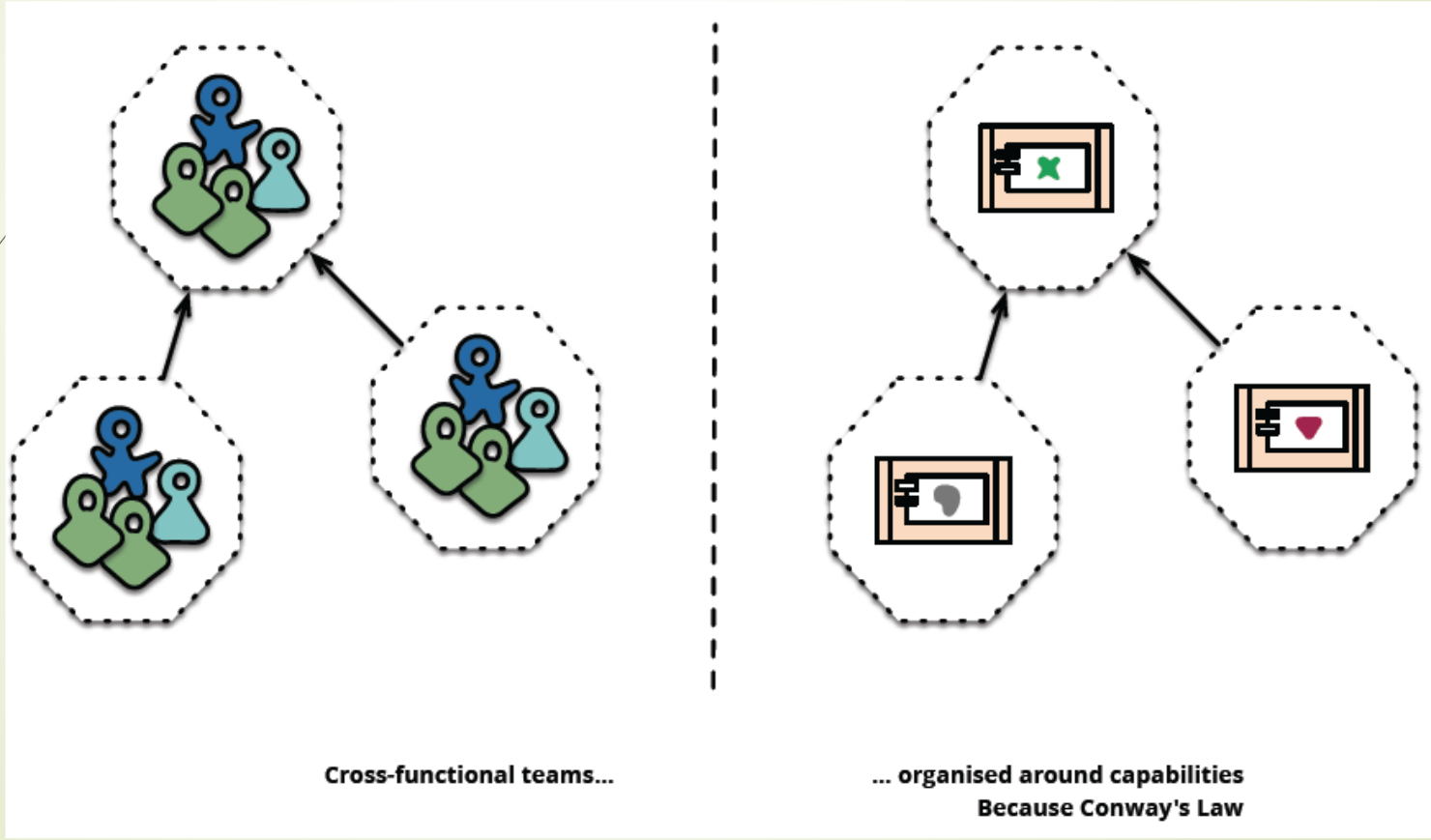
DEPLOYMENT


compile, unit and functional test
acceptance test
integration test
user acceptance test
performance test
deploy to production
run on build machine
deployed on build machine
deployed to integration environment
deployed to UAT environment
deployed to performance environment

Conway's law

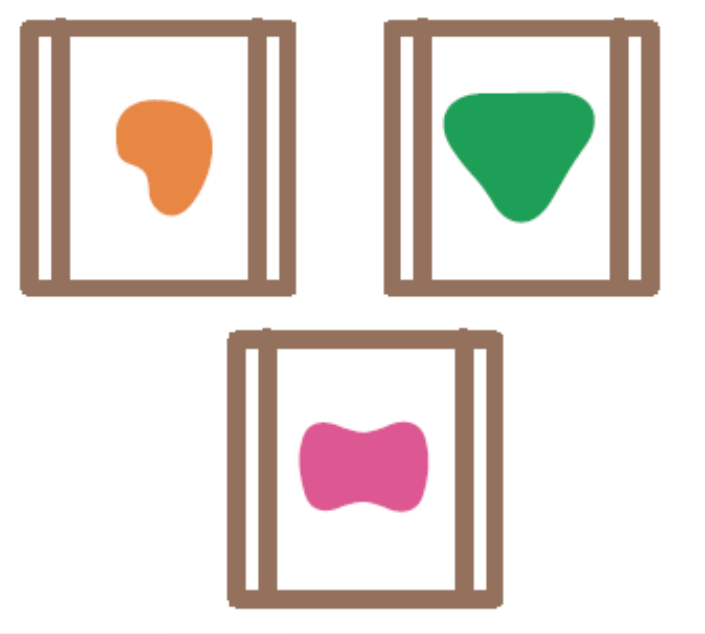
- "organizations which design systems ... are **constrained to designs which are copies of the communication structures** of these organizations"

Conway's law

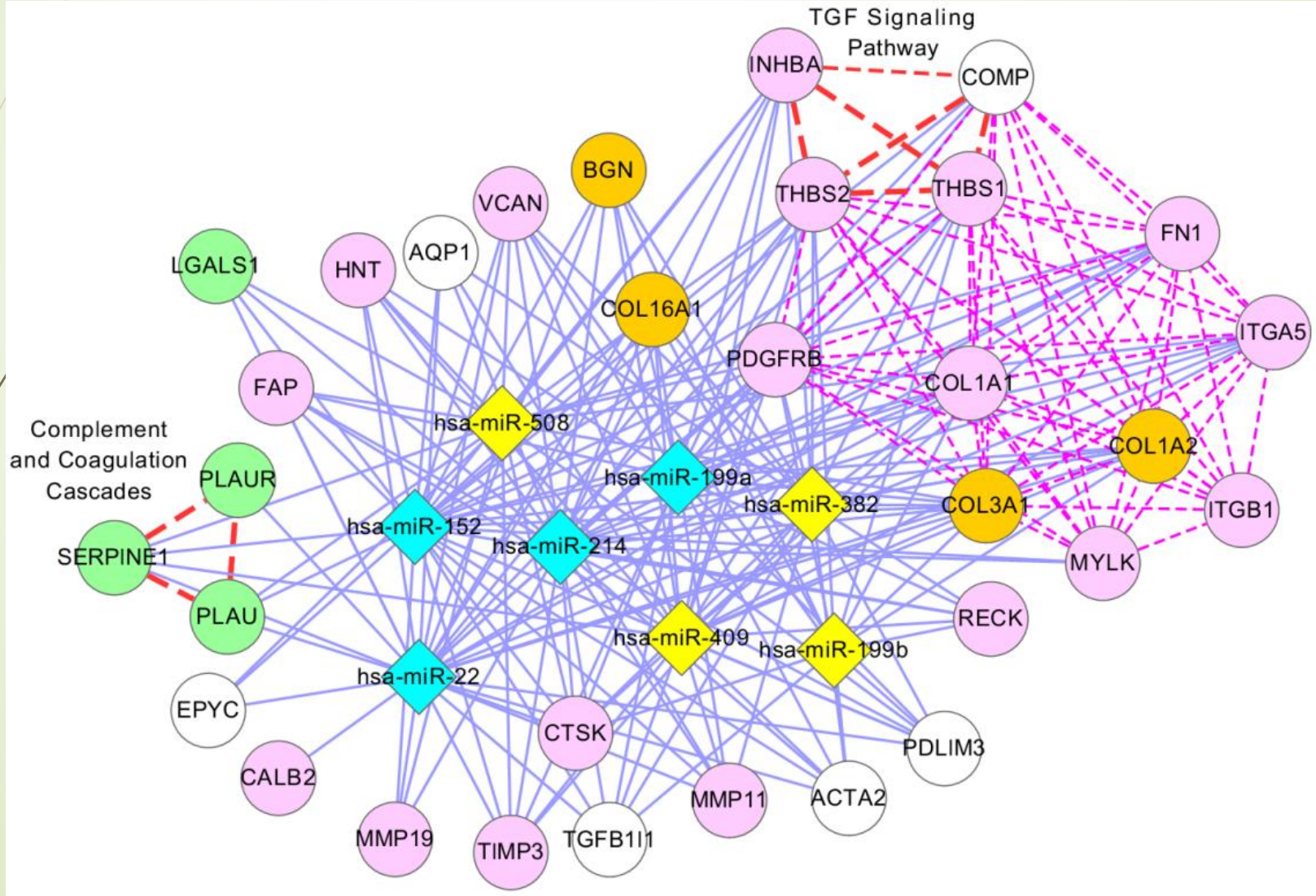

Cross-functional teams...
... organised around capabilities
Because Conway's Law

REPLACEABLE SERVICES

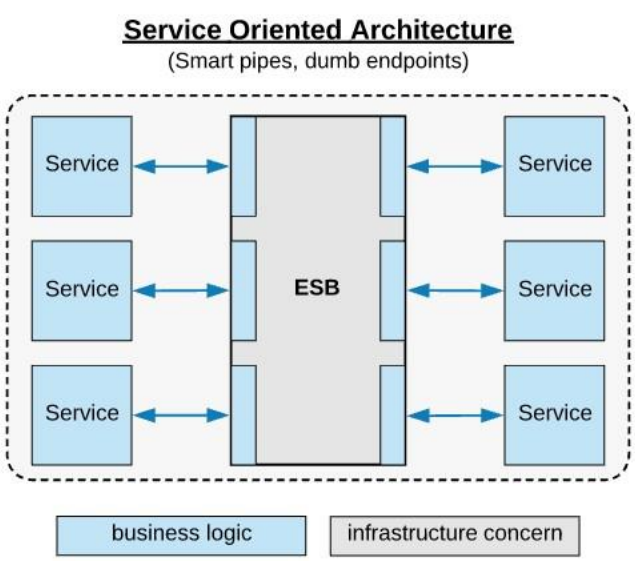
A microservices architecture puts each element of functionality into a separate service…

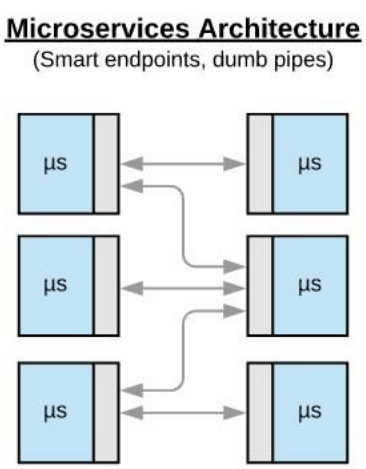


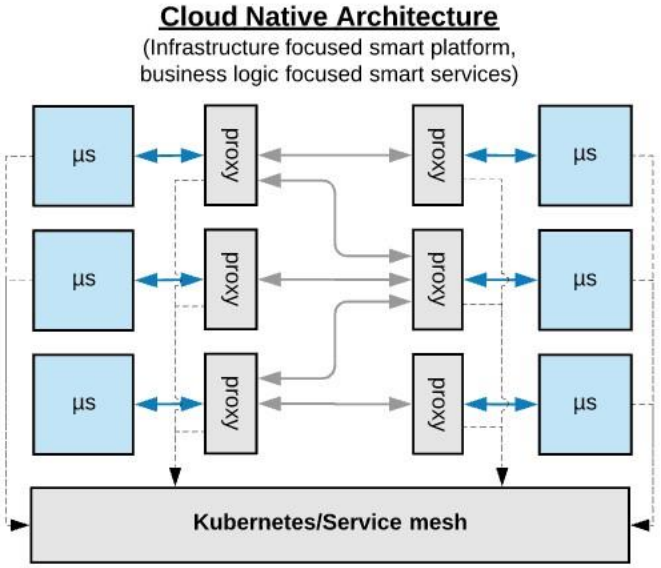
MORE INTEGRATION




EJB

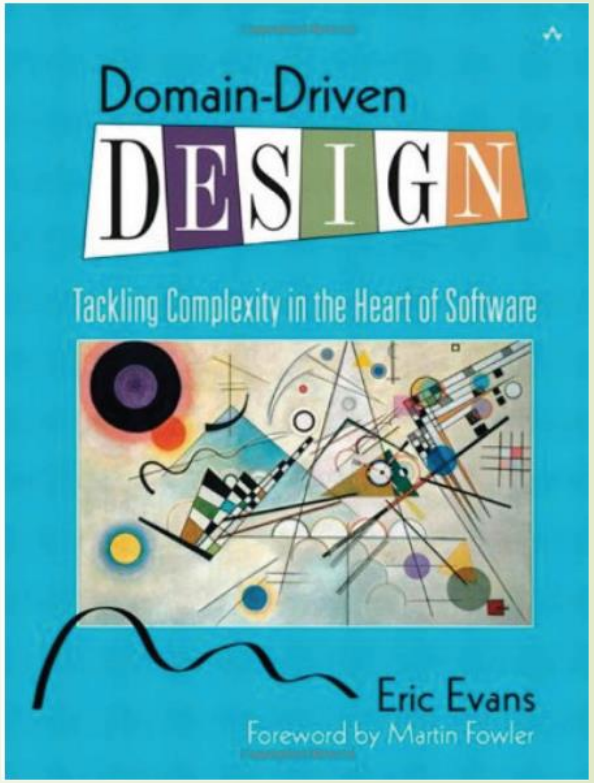

Service Oriented Architecture
(Smart pipes, dumb endpoints)
Service
Service
Service
ESB
Service
Service
Service
Service
business logic
infrastructure concern


Microservices Architecture
(Smart endpoints, dumb pipes)
µs
µs
µs
µs
µs
µs


Cloud Native Architecture
(Infrastructure focused smart platform,
business logic focused smart services)
µs
proxy
proxy
µs
µs
proxy
proxy
µs
µs
proxy
proxy
µs
Kubernetes/Service mesh

What are the key challenges?

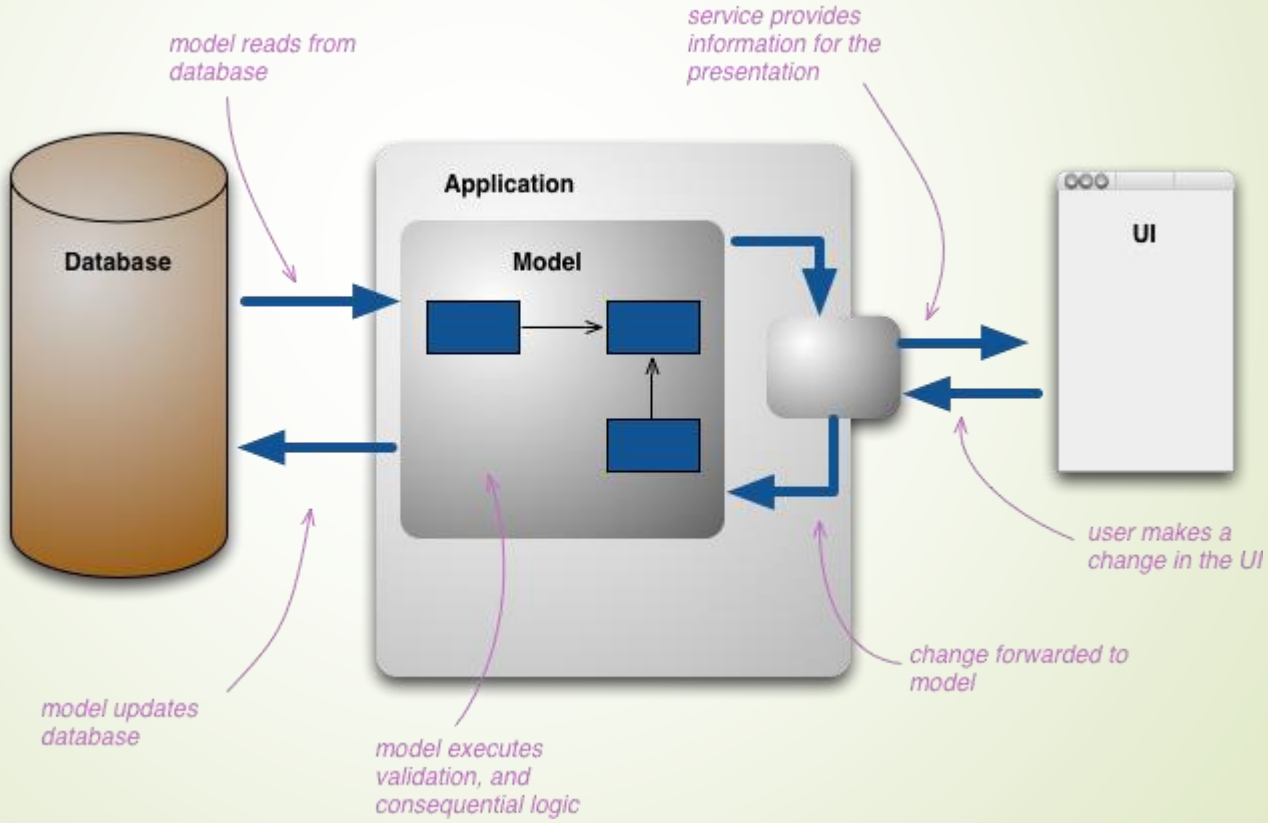
- BOUNDED CONTEXT
  - Domain Driven Design
- CHANGING DATA
  - Refactoring Databases
- TESTING
  - CONSUMER DRIVEN CONTRACTS



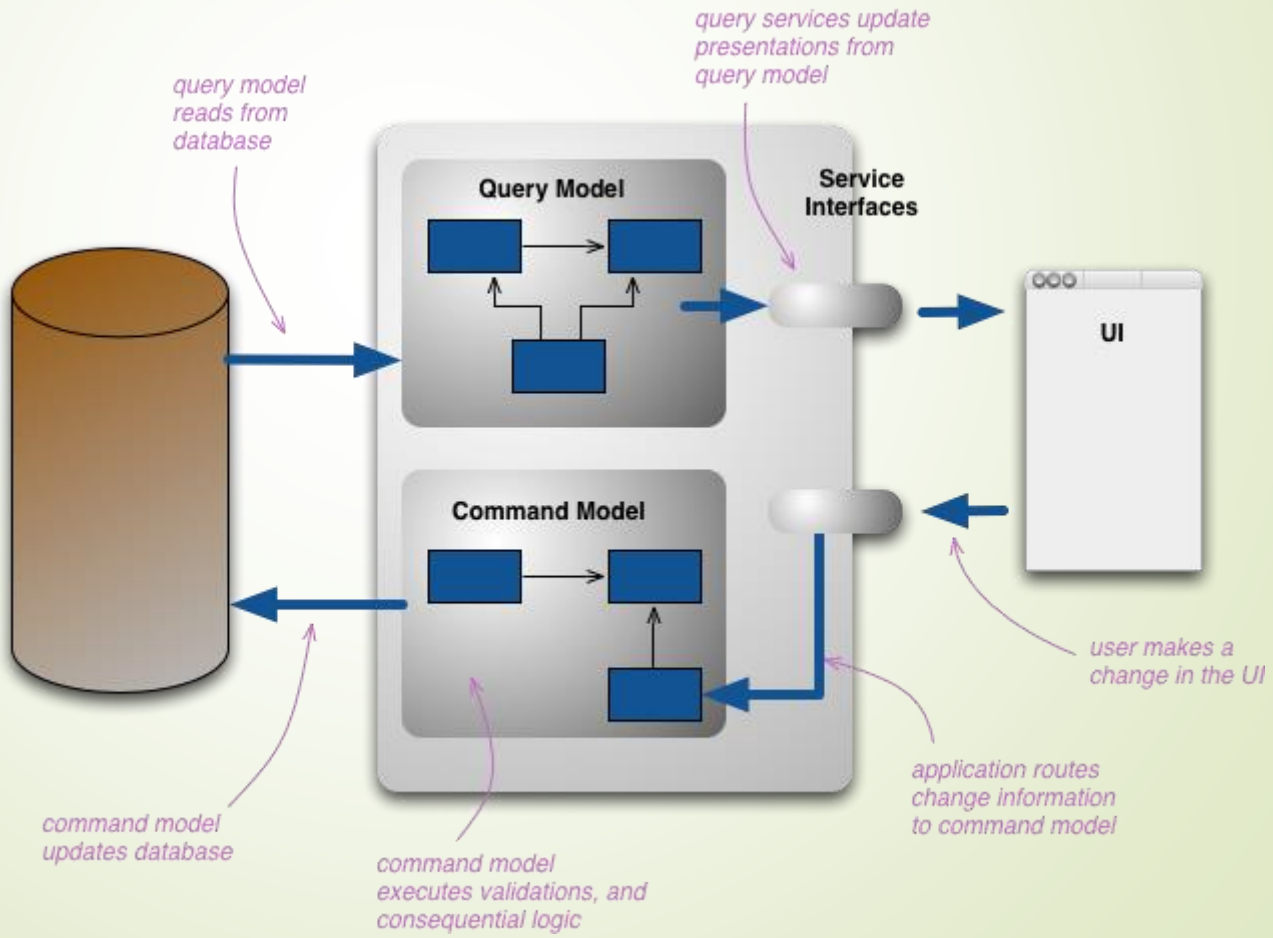
CQRS and Event Sourcing

- Command Query Responsibility Segregation
  - You can use a different model to update information than the model you use to read information.
  - The mainstream approach people use for interacting with an information system is to treat it as a CRUD datastore.
  - We may want to look at the information in a different way to the record store, perhaps collapsing multiple records into one, or forming virtual records by combining information for different places.

CRUD Model


model reads from database
service provides information for the presentation
Database
Application
Model
UI
user makes a change in the UI
change forwarded to model
model updates database
model executes validation, and consequential logic

CQRS Model


query services update presentations from query model
query model reads from database
Query Model
Service Interfaces
UI
Command Model
user makes a change in the UI
application routes change information to command model
command model updates database
command model executes validations, and consequential logic

CQRS Attributes

- As we move away from a single representation that we interact with via CRUD, we can easily move to a task-based UI.
- CQRS fits well with event-based programming models. It's common to see CQRS system split into separate services communicating with Event Collaboration. This allows these services to easily take advantage of Event Sourcing.
- Having separate models raises questions about how hard to keep those models consistent, which raises the likelihood of using eventual consistency.
- For many domains, much of the logic is needed when you're updating, so it may make sense to use EagerReadDerivation to simplify your query-side models.
- If the write model generates events for all updates, you can structure read models as EventPosters, allowing them to be MemoryImages and thus avoiding a lot of database interactions.
- CQRS is suited to complex domains, the kind that also benefit from Domain-Driven Design.

When to Use It?

- Should only be used on specific portions of a system (a BoundedContext in DDD lingo) and not the system as a whole.
- For handling high performance applications. CQRS allows you to separate the load from reads and writes allowing you to scale each independently.

Teorias de las Ciencias de la Computación

- Parnas' information hiding - what pieces are
- Postel's Law - how pieces compose
- Armstrong - how software can respond to the physical world
- Conway's Law - how software interacts with people
- Boehm's The Constructive Cost Model
- Constantine's Cohesion and Coupling
- Dijkstra's theory of cognitive limits ("Go to stmt considered harmful")
- Wirth's stepwise refinement
- Meyer's Design by Contract

Preguntas