

JavaCro15



Application architecture evolution,  
from simple script to microservices

Igor Buzatović  
Inovativni trendovi d.o.o.

Simple script


Microservices

# Application architecture evolution, from simple script to microservices

Igor Buzatović  
Inovativni trendovi d.o.o.

Application architecture evolution,  
from simple script to microservices

Igor Buzatović  
Inovativni trendovi d.o.o.

**Simple script** 

- No MVC
- Script file per request type (URI)
- All aspects (getting input params, fetching data, presenting data) mixed together

**Monolith** 

Components of "Monolith"

- UI (web framework)
- Service Layer
- Data access layer (JPA)
- Data storage (DBMS, MySQL)



**App. evolution examples**

2007  
Microservices - Monoliths with a Data + API service

2010  
Service-oriented - SOA, REST, Cloud, Microservices

2015  
Microservices - Cloud, API, Data, Mobile, IoT, Big Data



# *Simple script*

- No MVC
- Script file per request type (URI)
- All aspects (getting input params, fetching data, presenting data) mixed together

# Monolith

## Components of "Monolith"

- UI (web framework)
  - Service Layer
  - Data access layer (JPA)
- + Data storage (RDBMS, noSQL)



# *App. evolution examples*

eBay

Monolithic Perl -> Monolithic C++ -> Java -> microservices

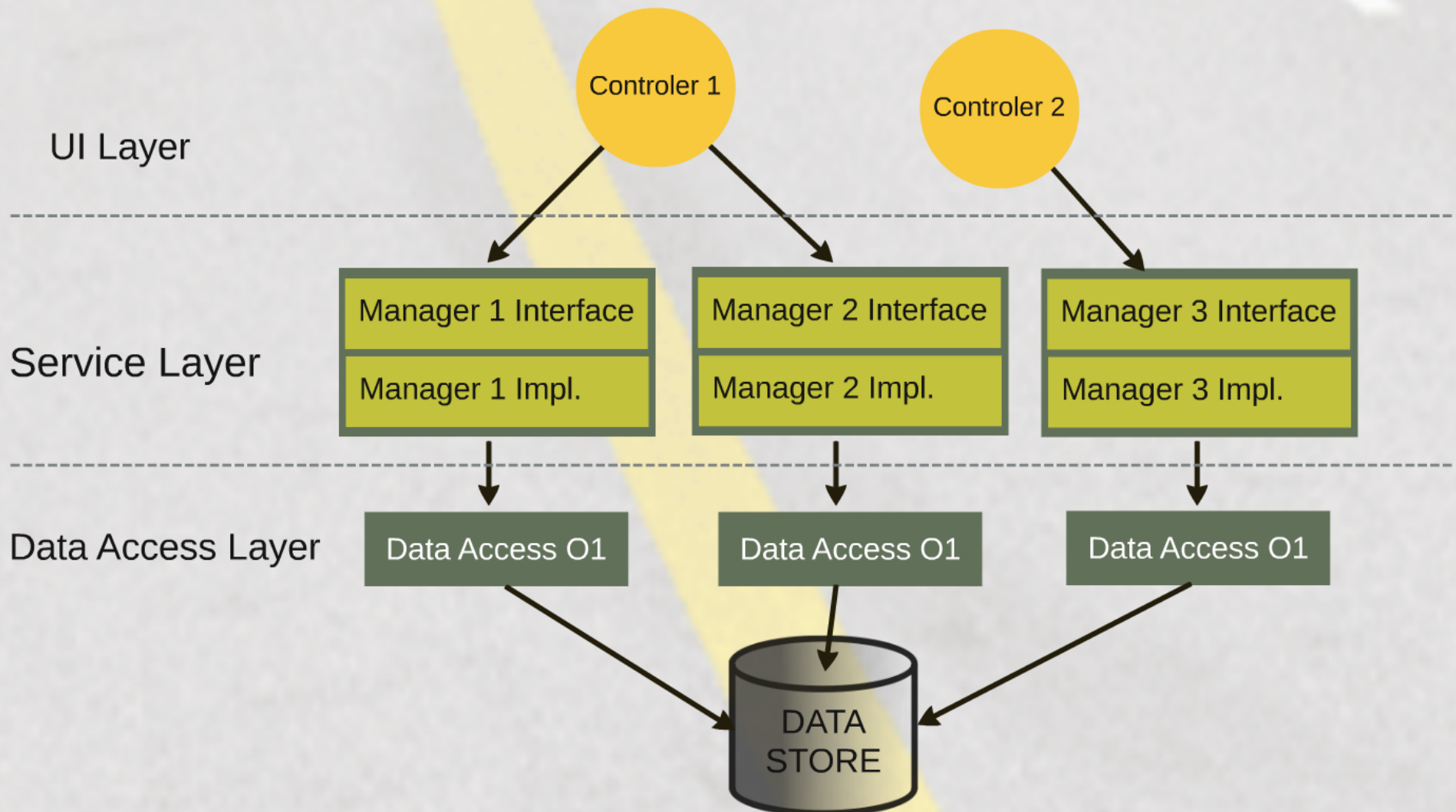
Twitter

Monolithic Rails -> JS / Rails / Scala -> microservices

Amazon

Monolithic C++ -> Perl / C++ -> Java / Scala -> microservices

# Monolith - components



# *Monolith -> Microservices, why ?*

High load handling ?

Easier to handle large data ?

More efficient caching

Much easier to understand the code

Each service can use different technologies (including language)

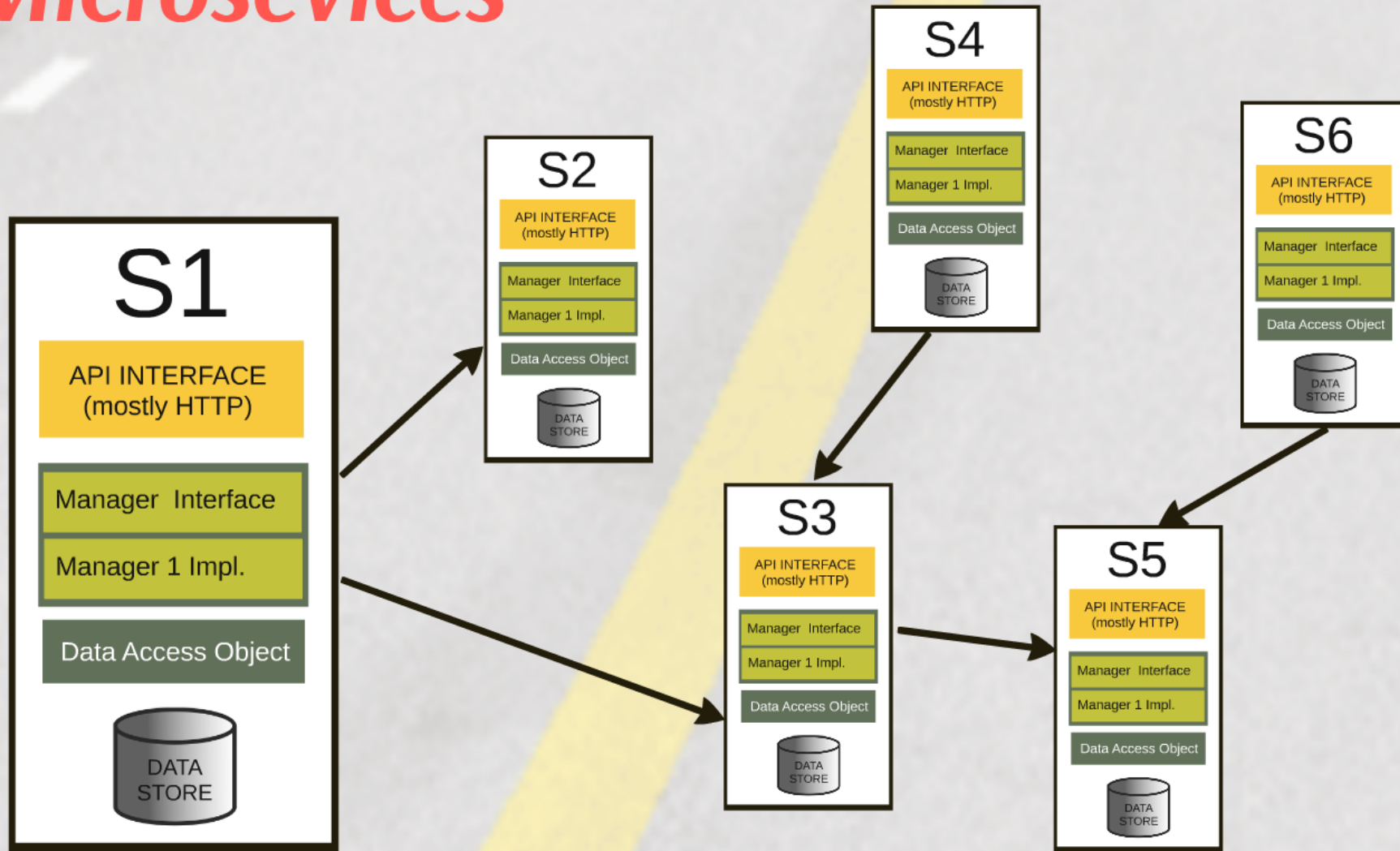


# *Microservice, what is it?*

“Loosely-coupled **service oriented architecture** with bounded contexts”

Adrian Cockcroft

# Microservices



# *Microservices - reactive principles*

## Responsive

The service responds in a timely manner if at all possible. (Reactive manifesto)  
Client protects itself with asynchronous, non-blocking calls

## Resilient

The service stays responsive in the face of failure.  
Resilience is achieved by replication, containment, isolation and delegation.

## Elastic

The service stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.

## Message driven

Asynchronous message-passing over synchronous request-response.

# *Microservices - downsides*

Communication overhaed

Much harder to control transactions

More complicated deployment (multiple clusters ,  
versioning)

More complicated monitoring and debugging

# Netflix OSS Tools



## Asgard

Web-based tool for managing cloud-based applications and infrastructure.

## Hystrix

Latency and fault tolerance library designed to isolate points of access to remote systems, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.

## Ribbon

Inter Process Communication (remote procedure calls) library with built in software load balancers. The primary usage model involves REST calls with various serialization scheme support.


## Eureka

AWS Service registry for resilient mid-tier load balancing and failover.

**Thank  
you!**

Application architecture evolution,  
from simple script to microservices

Igor Buzatović  
Inovativni trendovi d.o.o.

**Simple script** 

- No MVC
- Script file per request type (URI)
- All aspects (getting input params, fetching data, presenting data) mixed together

**Monolith** 

Components of "Monolith"

- UI (web framework)
- Service Layer
- Data access layer (JPA)
- Data storage (DBMS, MySQL)



**App. evolution examples**

2007  
Microservices - Monoliths with a Data + API service

2010  
Service-oriented - SOA, REST, Cloud, or Microservices

2015  
Microservices - Cloud, API, or Service Mesh, or Serverless

