Java EE for Cloud Native and Microservices

2016/12/2

Anil Gaur
GVP Engineering
Oracle Cloud Platform
Application Development

JavaOne ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

AppDev is About our Customer's Entire Portfolio

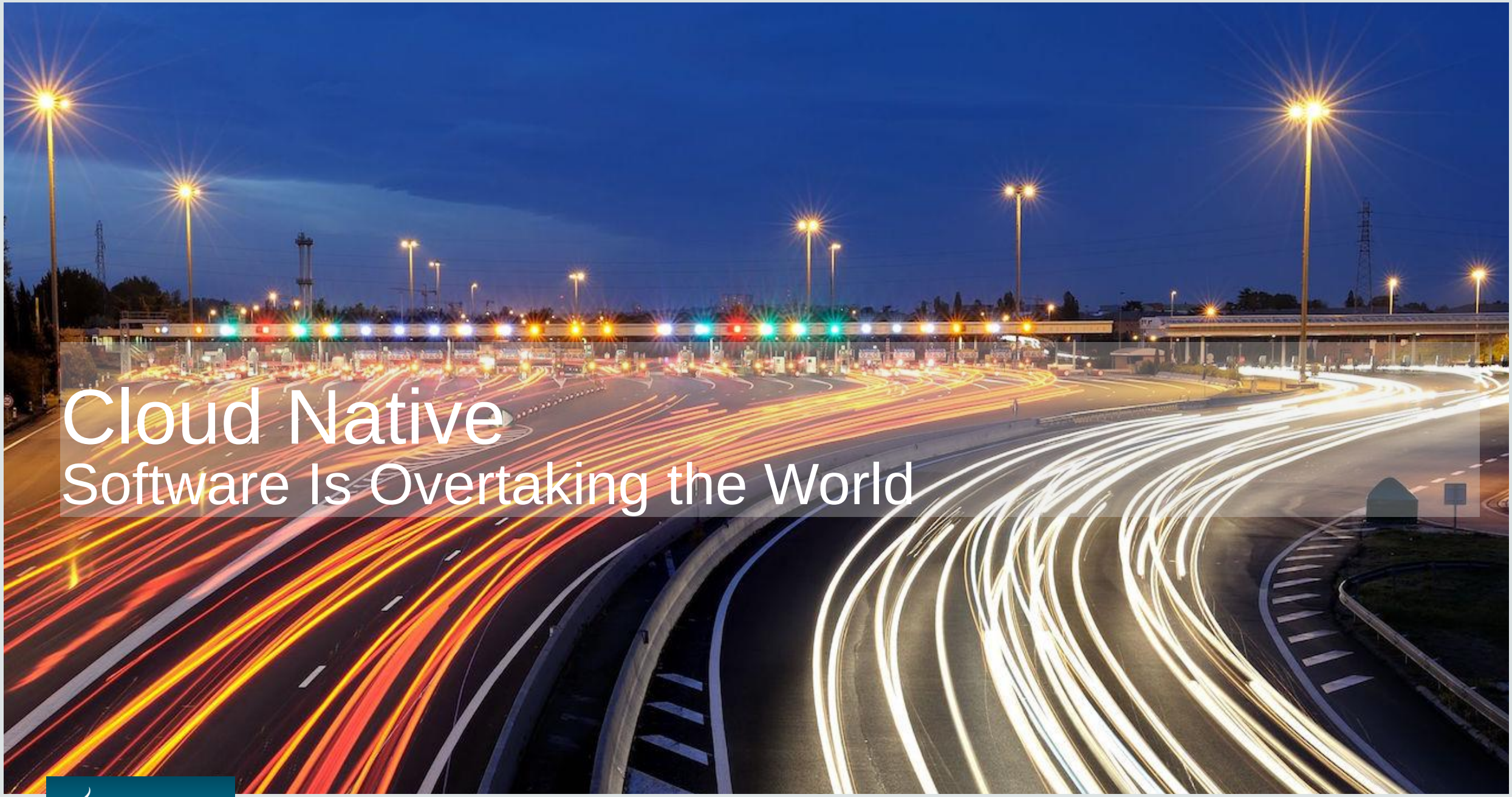
Core Software - Keep the Lights On

Differentiation Software - Run Current Business

Innovation Software - Find the Next Business

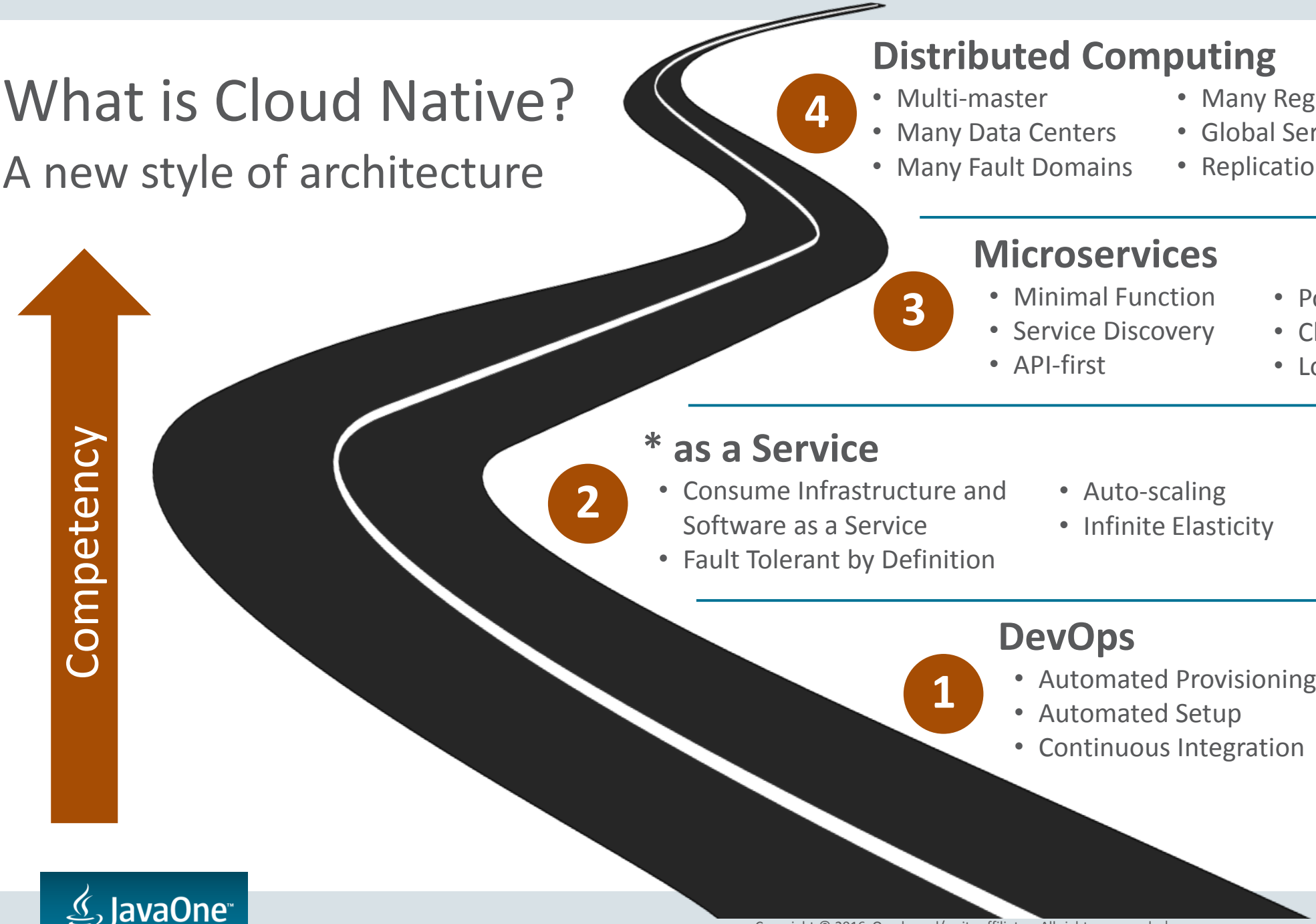
Business-centric	⟷	*IT-centric*
Top Line Growth	⟷	*Bottom Line Savings*
Release Hourly	⟷	*Release Quarterly*
Fail Early	⟷	*Fail Late*
Bespoke Software	⟷	*Packaged Software*
Agile	⟷	*Waterfall*
Product-based	⟷	*Project-based*

JavaOne ORACLE



What is Cloud Native?

A new style of architecture



4

Distributed Computing

- Multi-master
- Many Data Centers
- Many Fault Domains
- Many Regions
- Global Server Load Balancing
- Replication

3

Microservices

- Minimal Function
- Service Discovery
- API-first
- Polyglot
- Choreography
- Loose Coupling

2

* as a Service

- Consume Infrastructure and Software as a Service
- Fault Tolerant by Definition
- Auto-scaling
- Infinite Elasticity

1

DevOps

- Automated Provisioning
- Automated Setup
- Continuous Integration
- Continuous Delivery
- Automated Testing
- Agile
- Culture Change

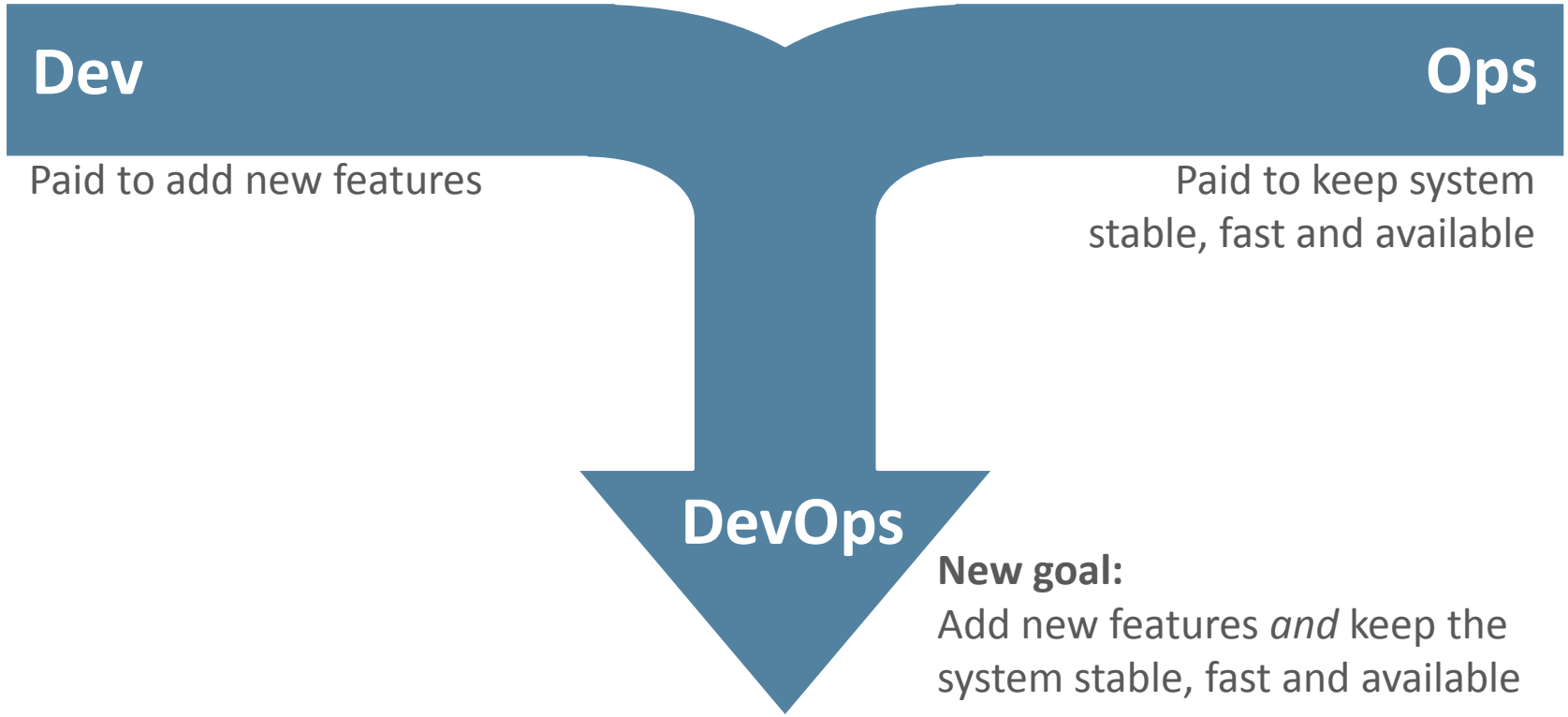
JavaOne ORACLE

Prerequisite #1 - DevOps

DevOps seeks to solve this

DevOps Principles

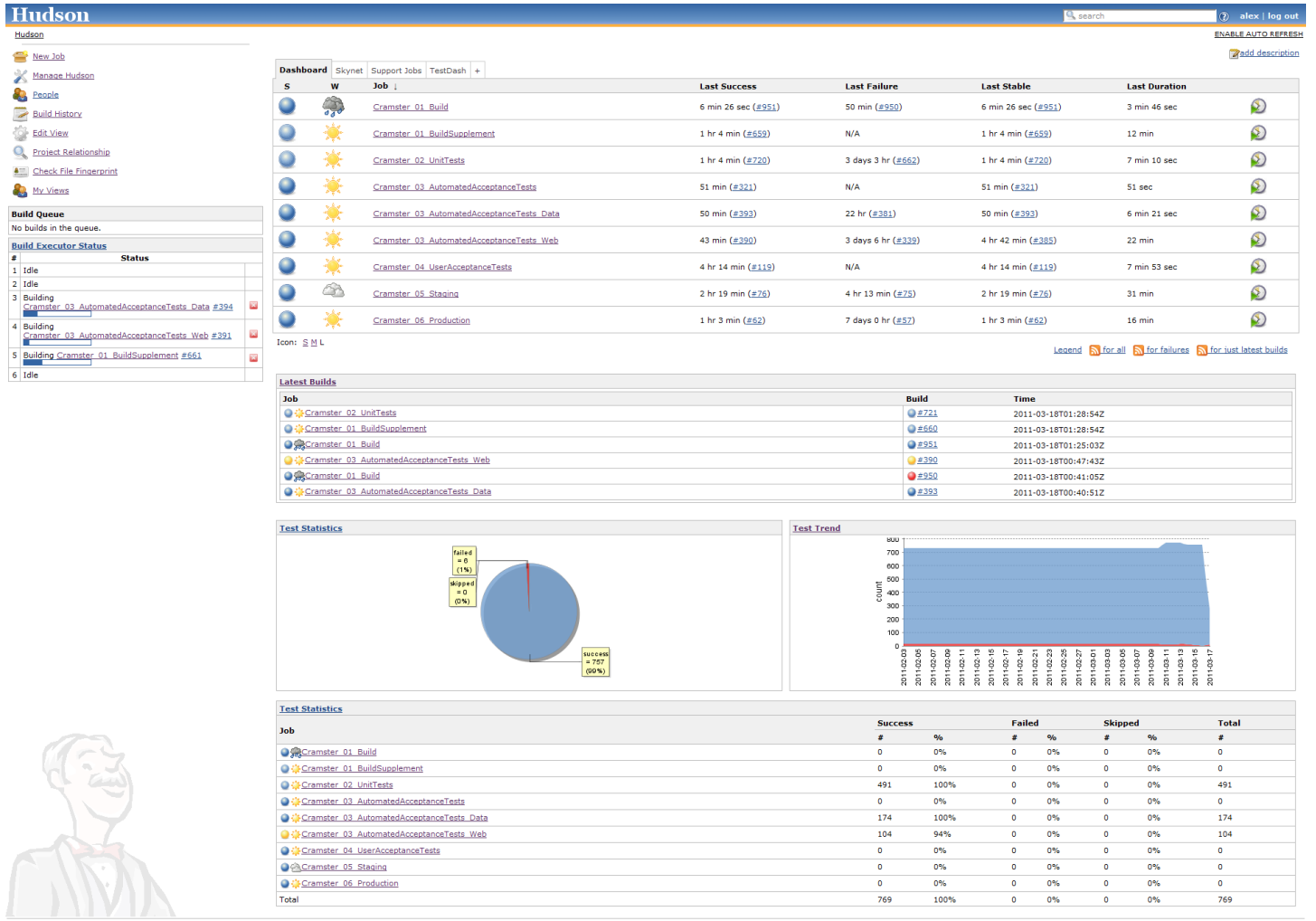
Cultural movement enabled by technology



Everyone Should Be Able to Build Anything At Any Time

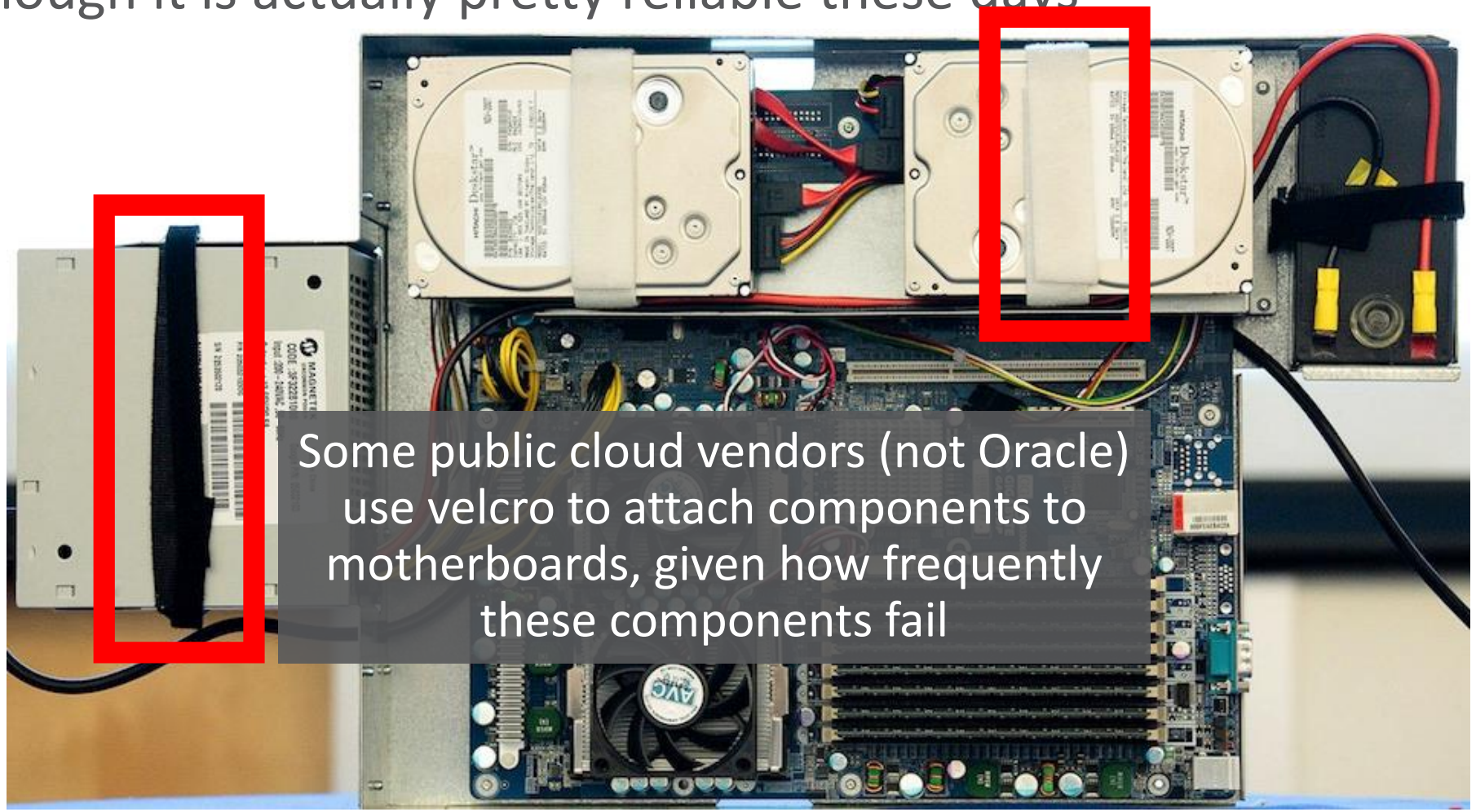
Set it and forget it

- Manual one button build/deploy
- Scheduled builds - every day, every week, etc
- Builds triggered by code checkins
- If post-build validation fails, report it



Assume Your Infrastructure is Unreliable

Even though it is actually pretty reliable these days

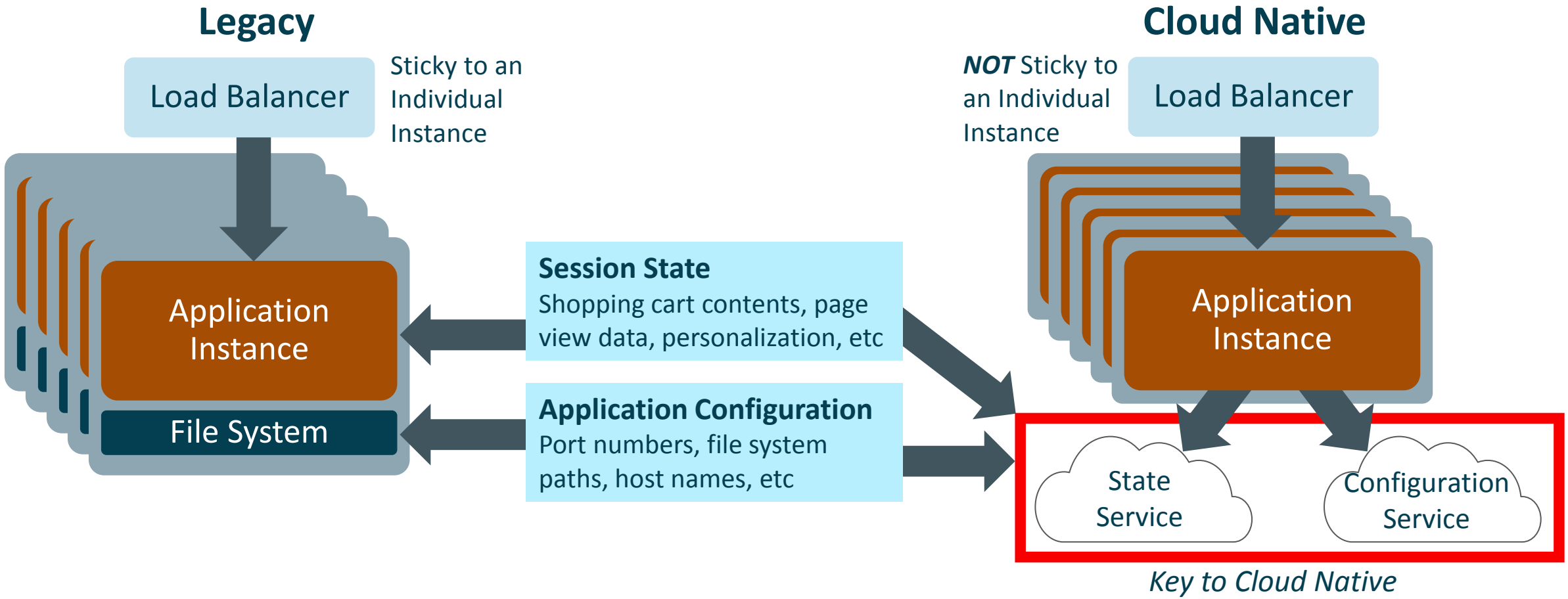


Remove All Hard-coded IPs, Host Names, etc

Use service discovery, DNS, etc instead. *Everything* should be dynamic

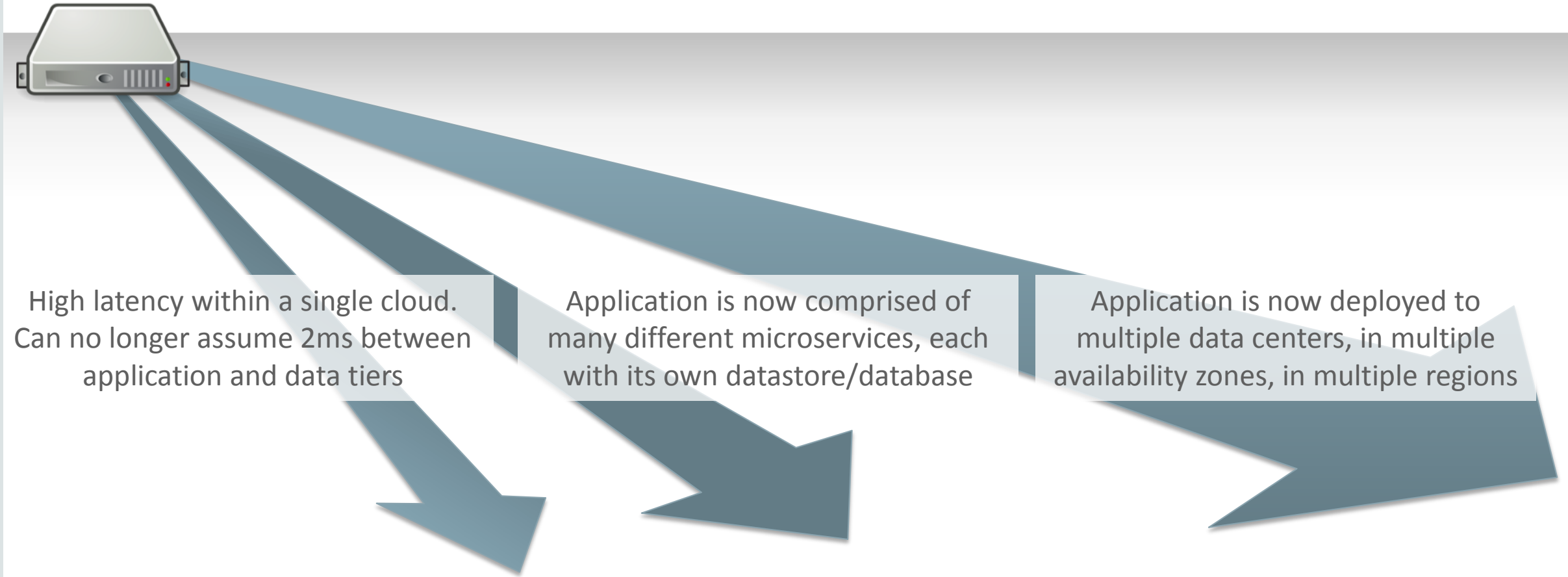
Make Your Middle Tier Stateless

Push all state and configuration down to highly available cloud services



Remember That Latency is Everywhere

Asynchronously make calls to all remote systems



High latency within a single cloud. Can no longer assume 2ms between application and data tiers

Application is now comprised of many different microservices, each with its own datastore/database

Application is now deployed to multiple data centers, in multiple availability zones, in multiple regions

Microservices Architecture & Technology

ORACLE

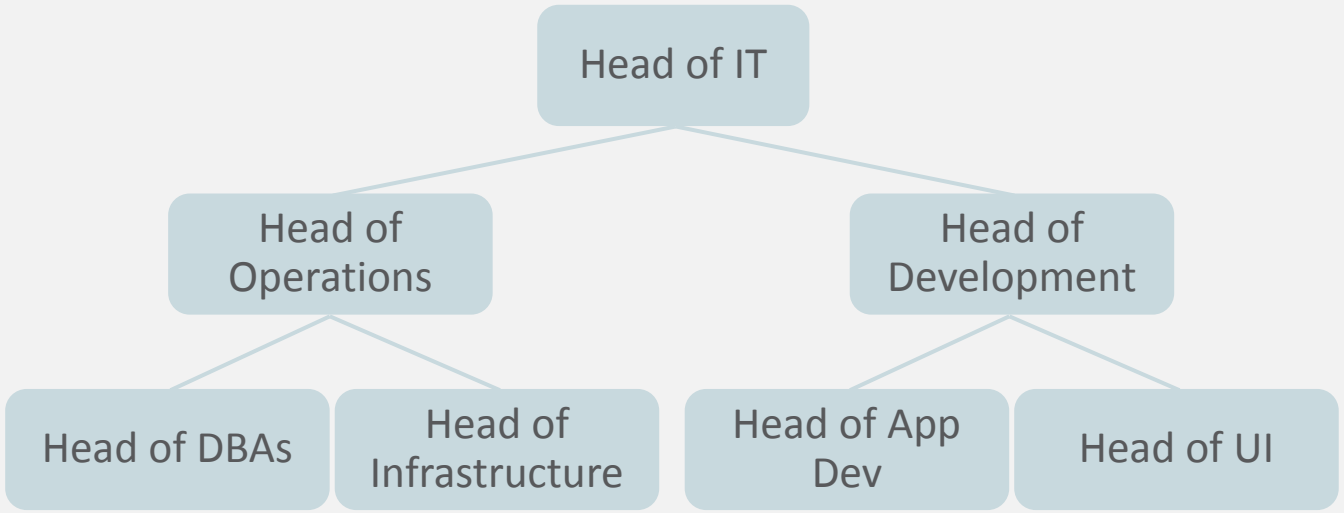
JavaOne ORACLE

Why Microservices?

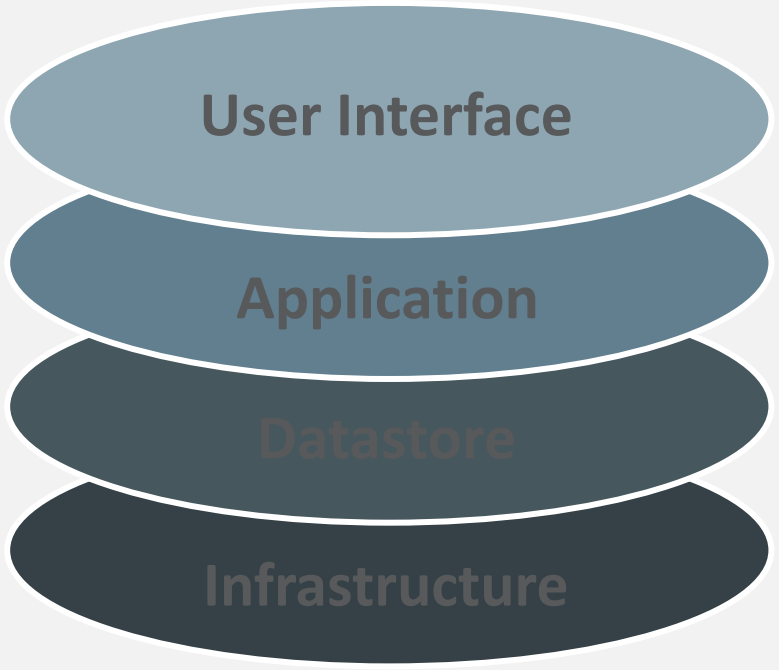
Conway's Law In Action

Any piece of software reflects the organizational structure that produced it

Typical Enterprise Organization Structure



Resulting Software



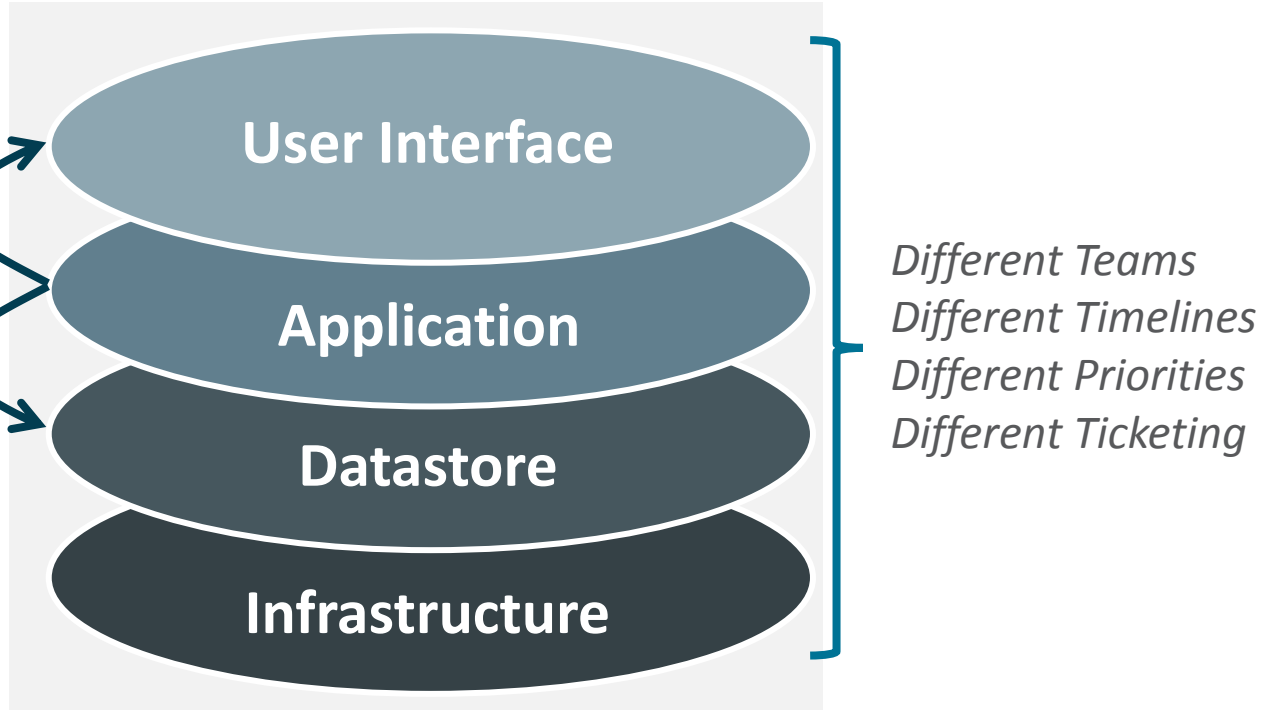
An Enormous Monolith

Even Simple Changes Are Hard to Implement With Monoliths

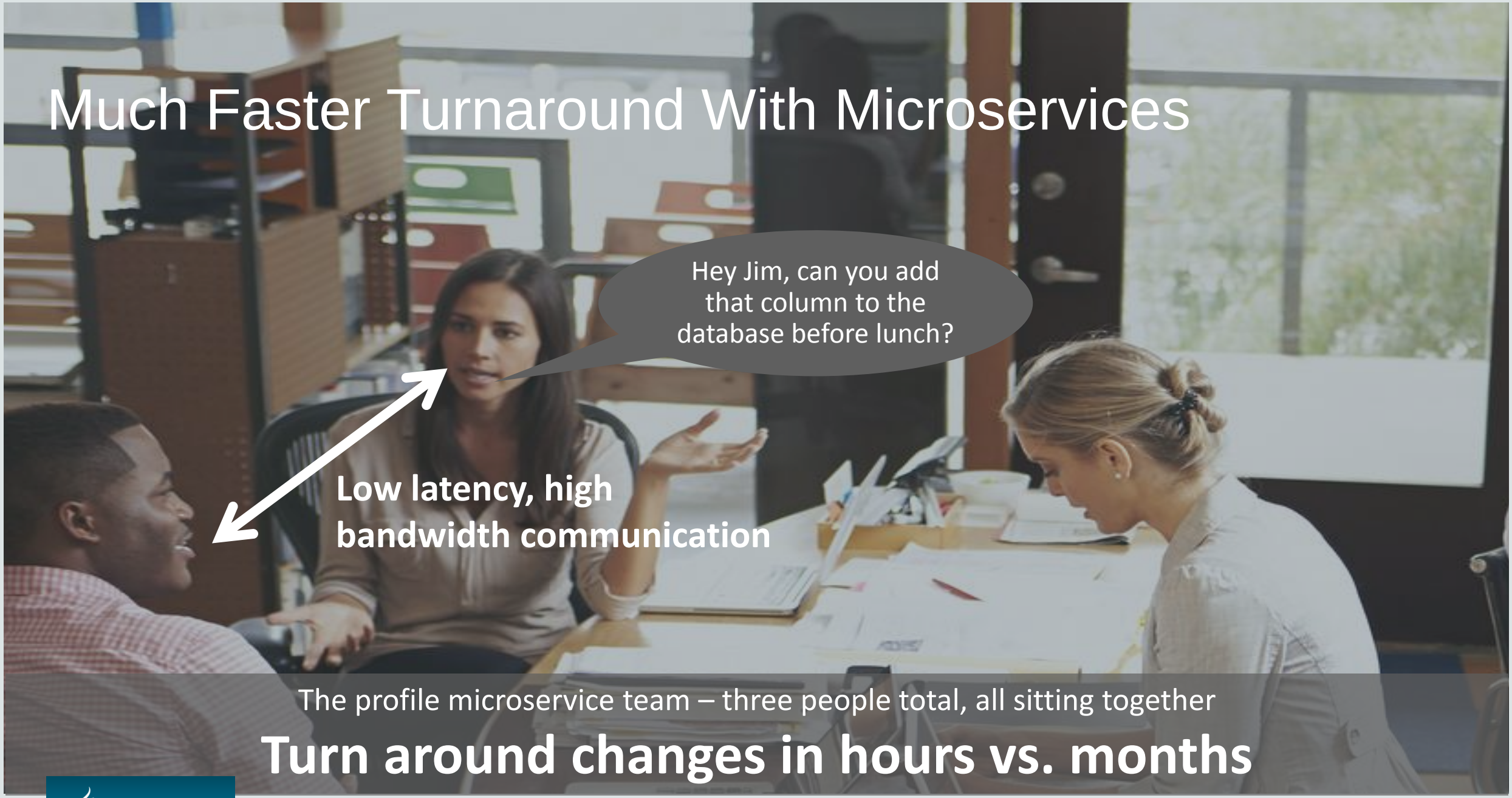
Organizational boundaries introduce the need to extensively coordinate

New requirement: Add a birthdate property to the customer's profile. How does this get implemented?

3. Application developer tickets UI team to have them add that property to the profile screens
2. Application developer adds the new property to the application-level code
1. Application developer tickets DBAs to have them add that property as a column in the database



Much Faster Turnaround With Microservices

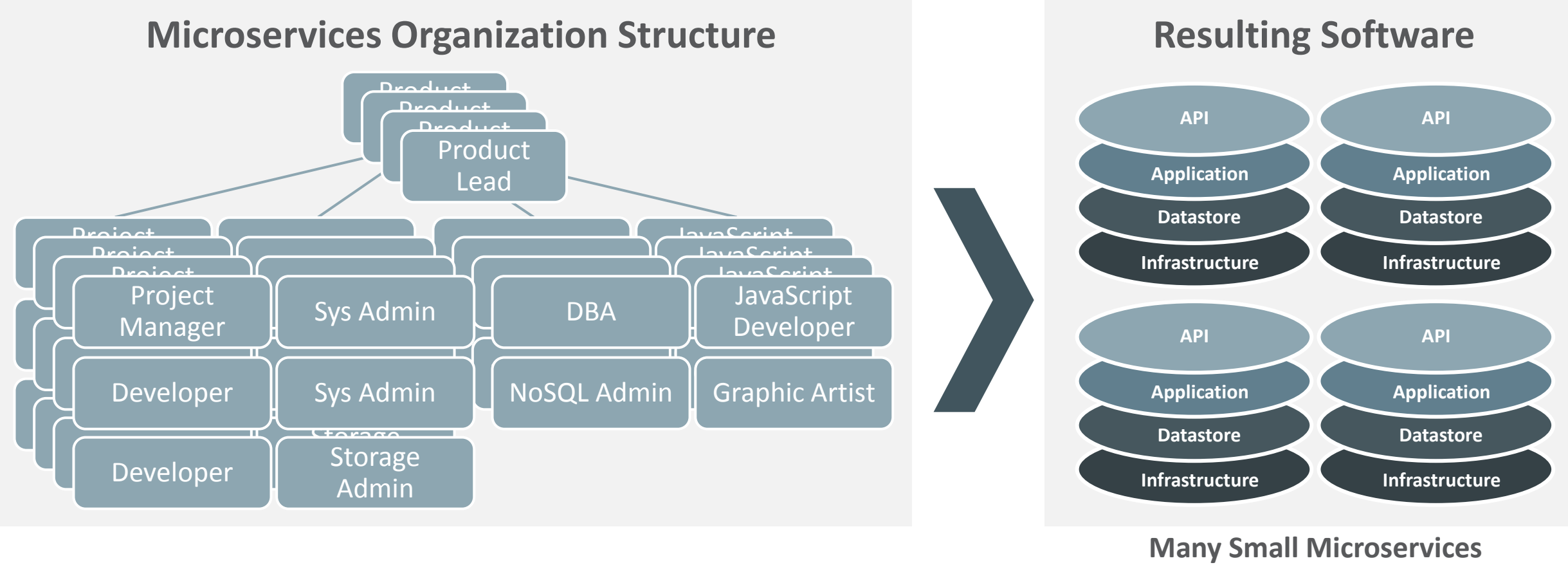


The profile microservice team – three people total, all sitting together

Turn around changes in hours vs. months

Re-structure Your Organization – Put Conway's Law to Work

Build small product-focused teams – *strict one team to one microservice mapping*



Defining Characteristics of Microservices

- Tens of versions of each microservice running concurrently
- Thousands of microservices
- Polyglot - each microservice can have its own tech stack
- Firm boundaries between services
- Distributed coupling - loose coupling, extensive messaging

Common Microservices Best Practices

Common Microservice Adoption Use Cases

I want to **extend** my existing monolithic application by adding microservices on the periphery.

I want to **decompose** an existing modular application into a microservices-style application

I want to build a **net new** microservices-style application from the ground up.

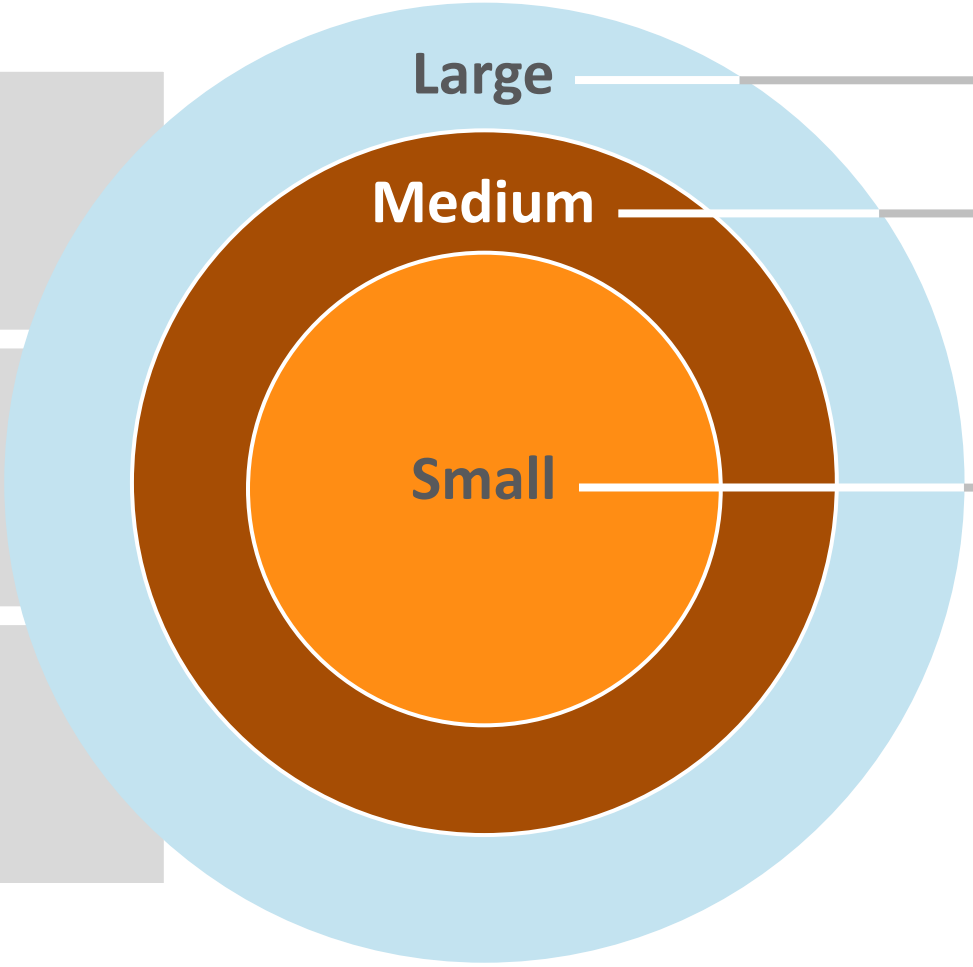
How Big Should a Microservice Be?

Can have hundreds of microservices for a larger application

Do One Thing and Do It Well

Avoid Inter-dependencies

Focus on Business Capabilities



11-15 People
Example: Order Microservice

4-10 People
Example: Inventory Microservice

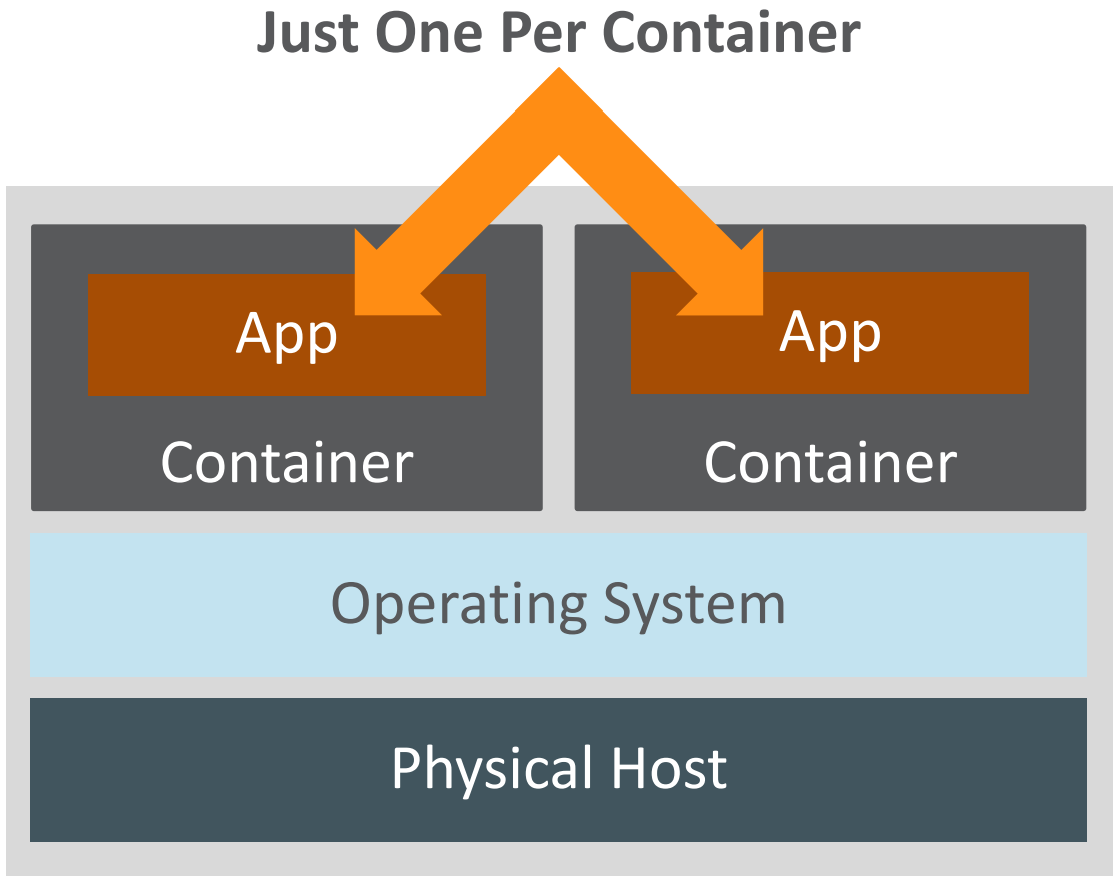
1-3 People
Example: Order Status Microservice

Each Team Should Treat Other Teams as External Vendors

Good fences make good neighbors

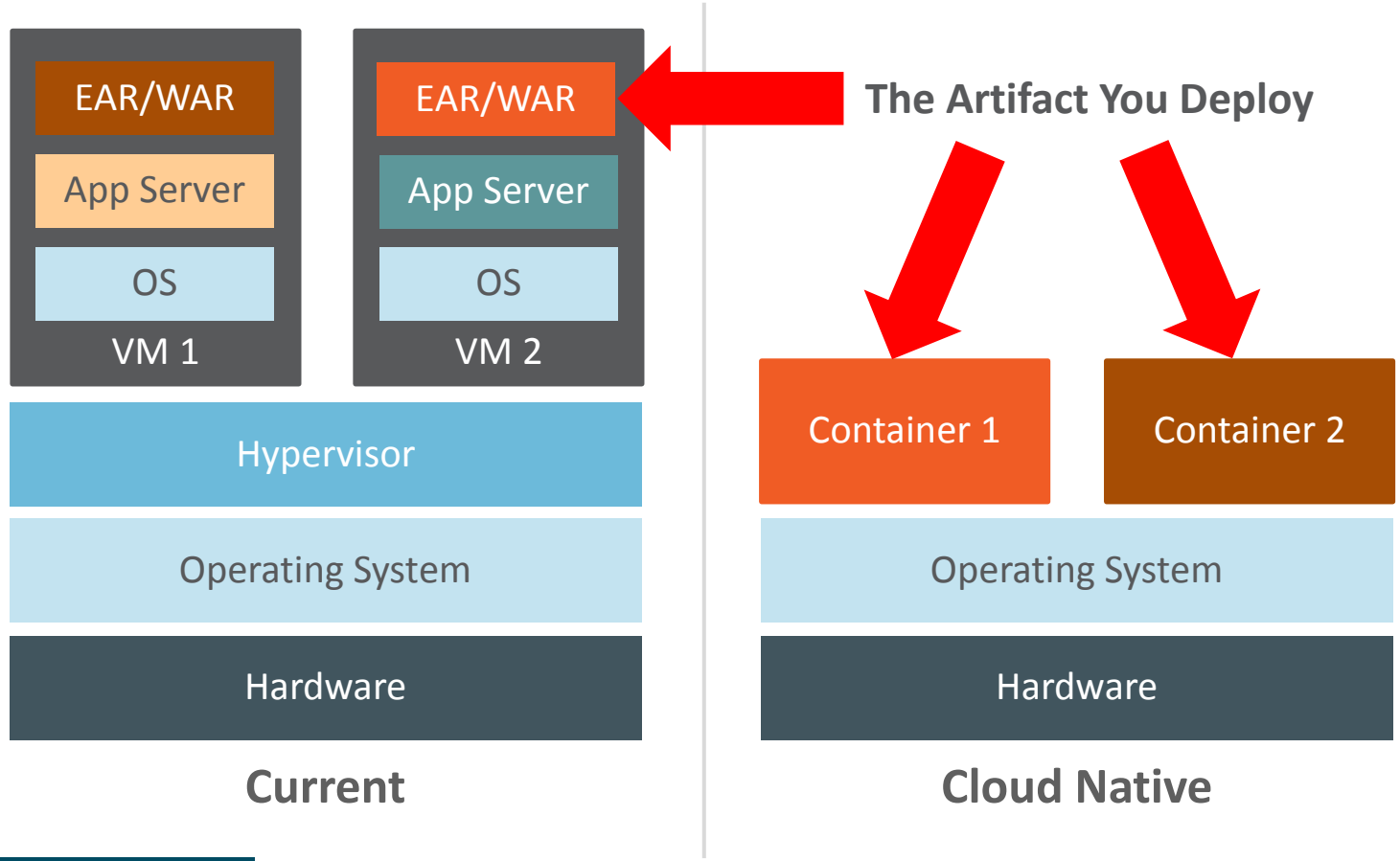
One Instance Per Container is Typical

- Best to run one instance (unique host/port combination) per container
- Running multiple instances of the same application or different applications will make scheduling very difficult
- Expose one port per container



Artifacts Are Now Immutable Containers – Not EARs, WARs

- Containers can have anything in them and are highly portable



- No more installing a JVM, app server, and then deploying the artifacts to them
- Build the container once, deploy it anywhere. Can include complex environment variables, scripts, etc
- Containers should be free of state and configuration
- Containers should not assume they are able to write to a persistent local file system

Java EE – What's Next?

Java™
Enterprise Edition

New AppDev Style
for Cloud and Microservices

Java EE - Available On Premise and in the Cloud

Cloud

Red Hat OPENSHIFT

IBM Bluemix

ORACLE Oracle Cloud PaaS

On Premise

IBM WebSphere

Red Hat JBoss

ORACLE WebLogic

Choice of Implementations

CAUCHO

NEC

Cosminexus

TmaxSoft

Red Hat® JBoss Enterprise Application Platform

WildFly

OW2

IBM

SAP

INFORS

FUJITSU

ORACLE FUSION MIDDLEWARE WEBLOGIC SERVER

Tong Tech®

GERONIMO

Rapid Changes Over Past Few Years

Driven by increasing business needs

Microservices

Apps divided into many small pieces

Distributed Computing

Many data centers, AZs, regions, etc.

Polyglot

Java leads but use of others increasing

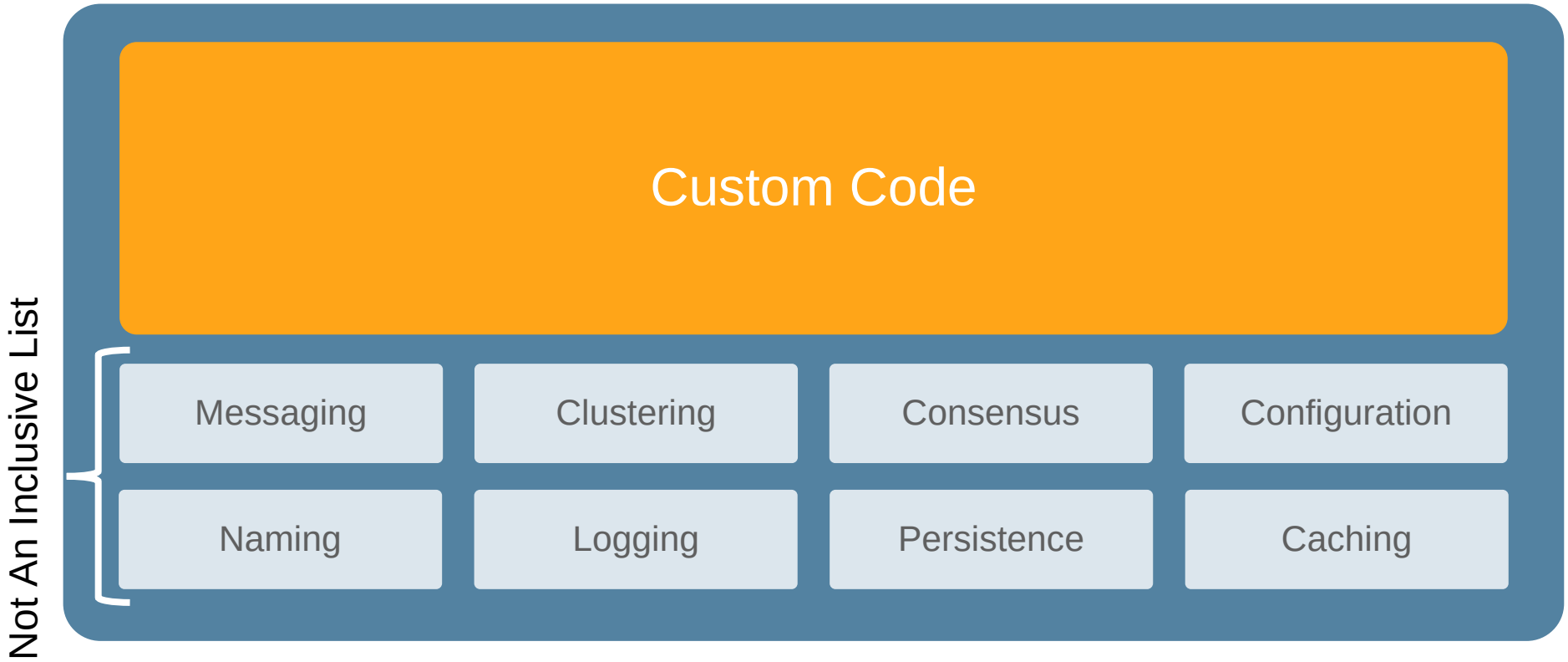
New Technology

Docker, Cloud, DevOps, etc.

Back to Basics: Apps Have Always Required the Same Building Blocks

Doesn't matter what decade we're in – building blocks are fairly constant

Problem Statement

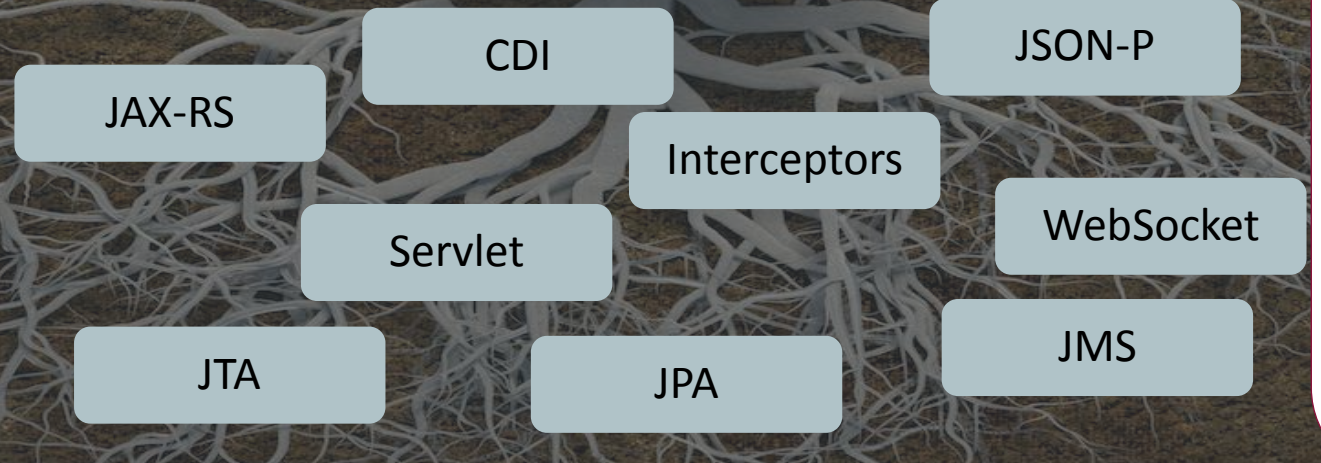


How Do We Make This Easier?

Building on the Lessons Learned

Be pragmatic

Use lessons learned and successful implementations



Choice of Implementations

CAUCHO

NEC

Cosminexus

TmaxSoft

Red Hat® JBoss Enterprise Application Platform

WildFly

OW2

IBM

SAP

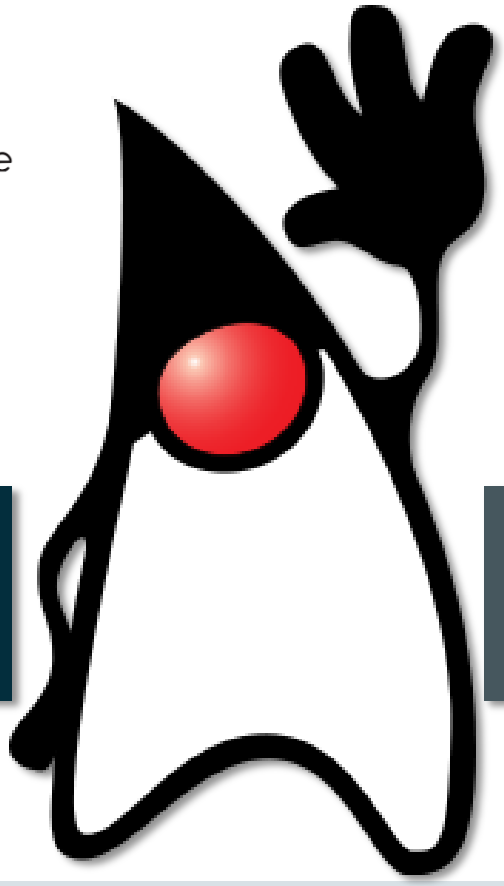
INFORS

FUJITSU

Tong Tech®

GERONIMO

Java EE APIs - Backbone of Leading Open Source Projects



PRIMEFACES

vaadin

Jersey

RESTEasy

MicroProfile

spring boot

jetty://

resin®

WildFly

ORACLE CLOUD

Red Hat® OPENSHIFT

CLOUD Foundary

Web Frameworks	REST	Microservices	Web Containers	Java EE Containers	PaaS

http://tomcat.apache.org/

Cloud Development

Heterogeneous Clients

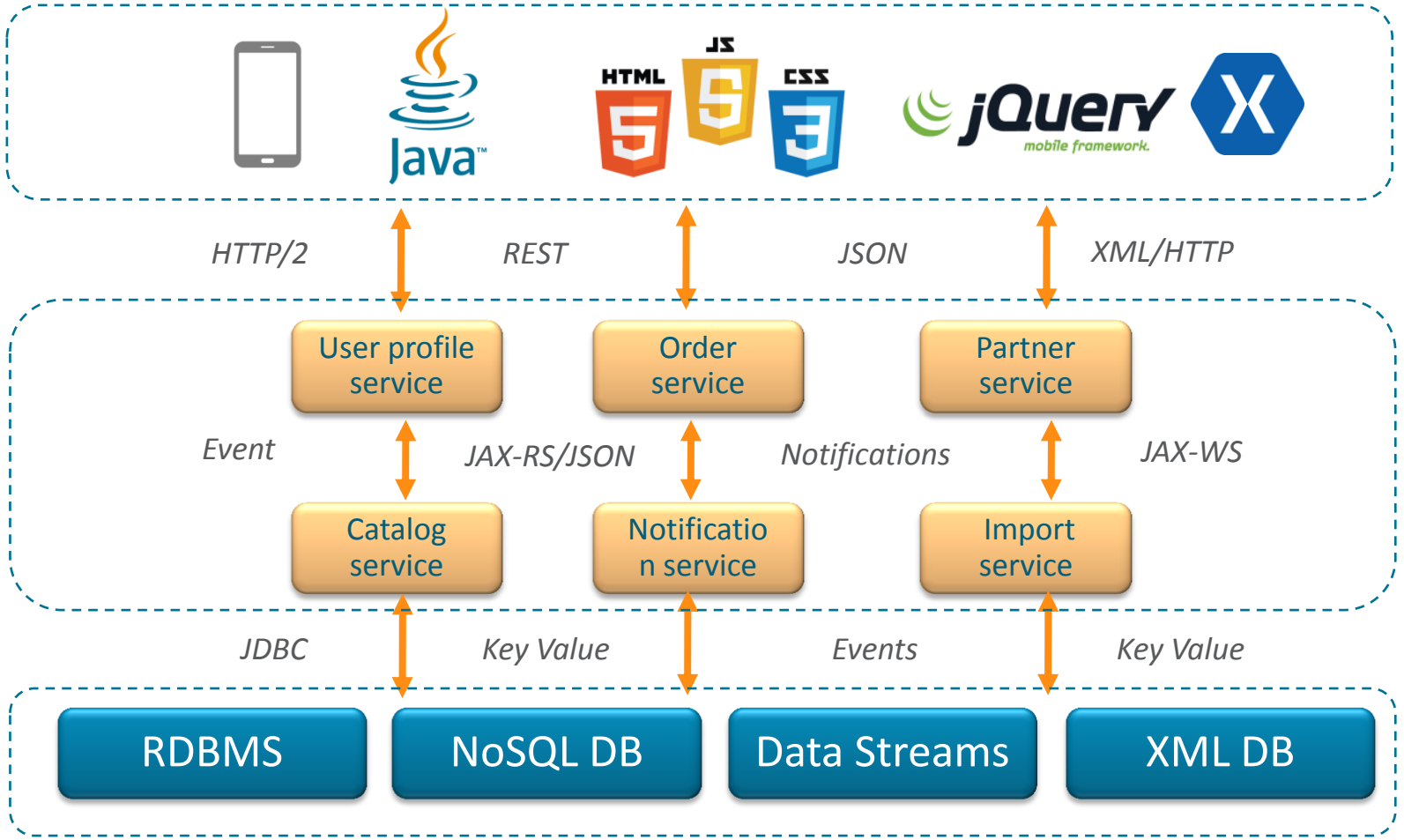
- Mobile, REST, HTML5

Stateless Services

- Managed and scaled independently

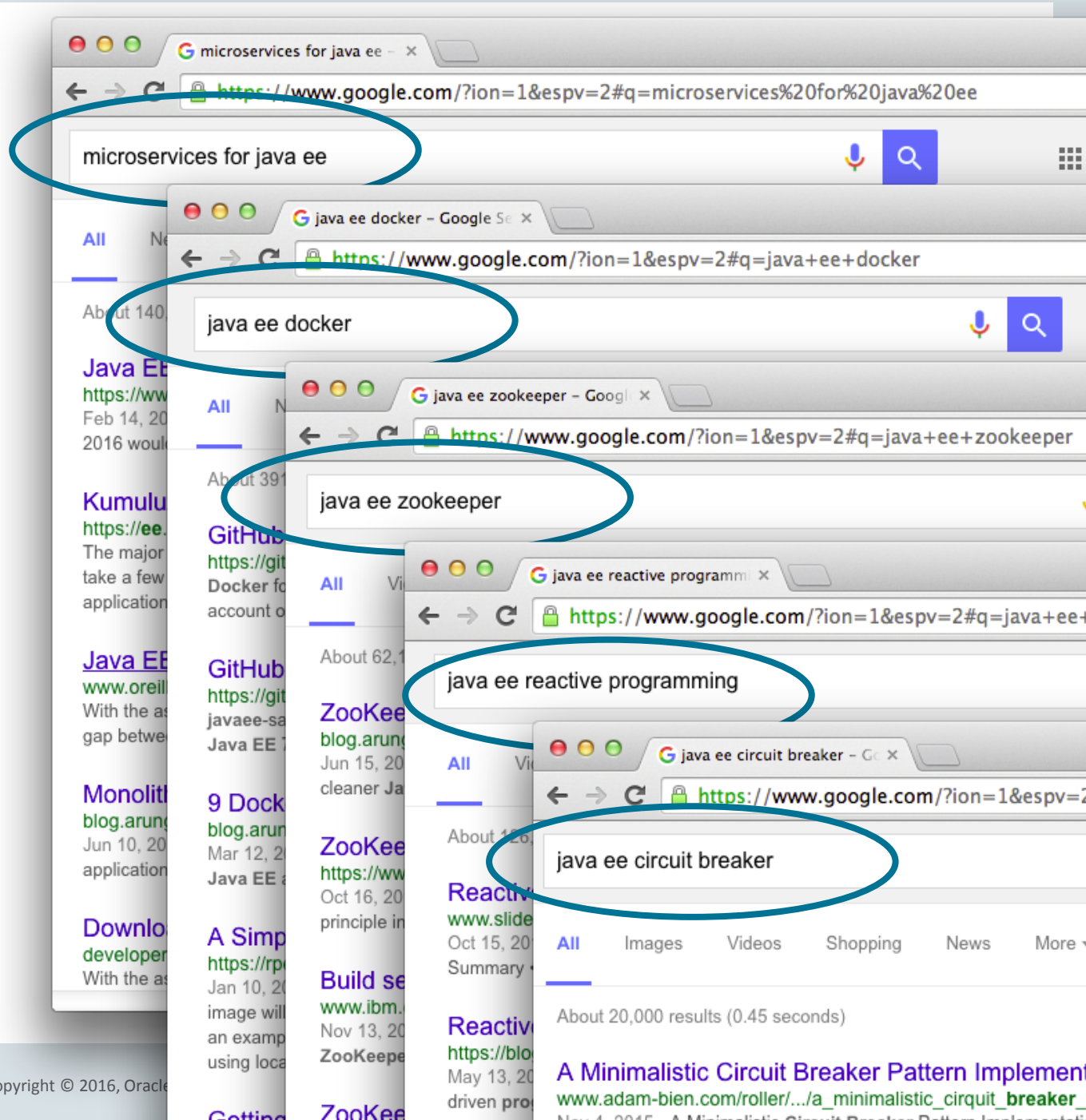
Data Sources

- Relational, non-relational



It’s Confusing!

Too many choices....
Which components?
Overall architecture?
Standards?
Vendor commitment?

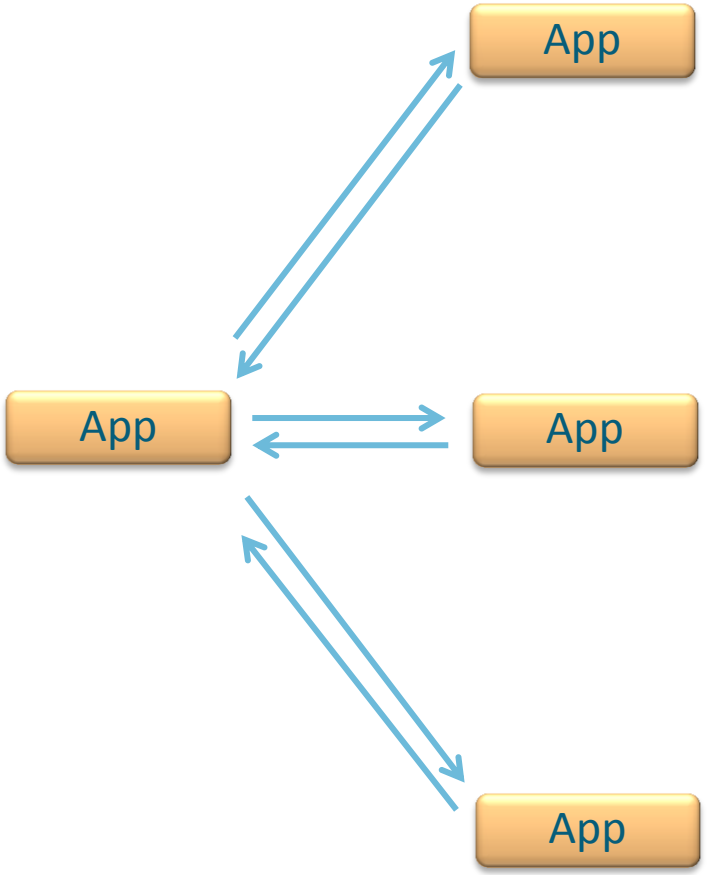


Proposed Enhancements for Java Enterprise Edition

- New AppDev style for Cloud and Microservices
- Build on proven technologies
- Comprehensive
  - Programming Model, Packaging, Portability
- Standards-based
  - This is a proposal only
  - Will work with the community and follow the JCP process

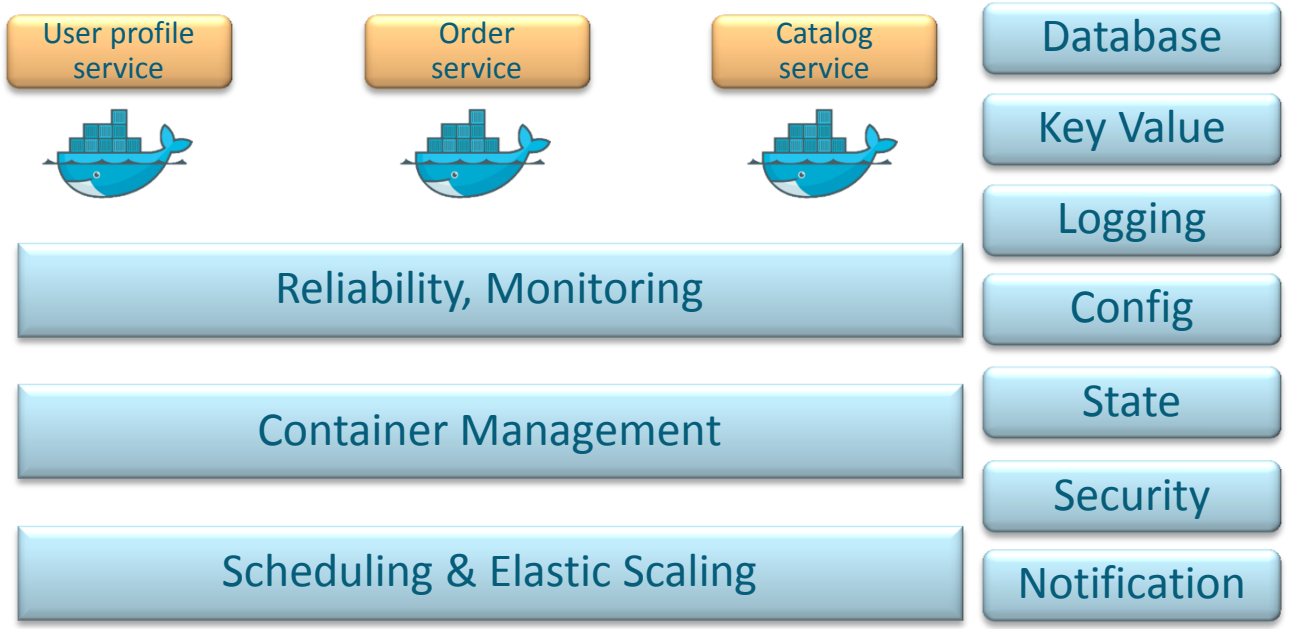
Designed for Agility and Scalability with Security

- Programming model
  - Extend for reactive programming
  - Unified event model, event messaging API
  - JAX-RS, HTTP/2, Lambda, JSON-B, …
- Eventual consistency
  - Automatically event out changes to observed data structures
- Key value/document store
  - Persistence and query interface for Key Value and Document DB
- Security
  - Secret management
  - OAuth/OpenID support



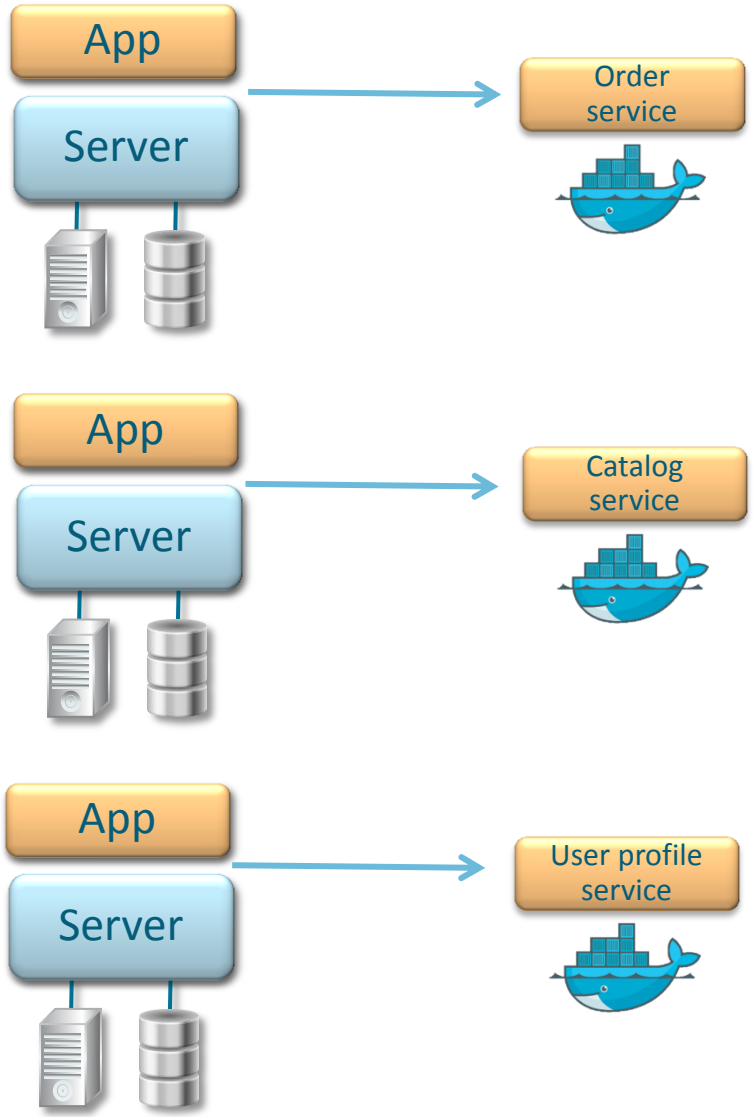
Location Transparency and Resiliency

- Configuration
  - Separate service packaging and configuration
  - API for external configuration
- State
  - API for external state
- Resiliency
  - Circuit breakers
  - Resilient commands
  - Standardized health reporting



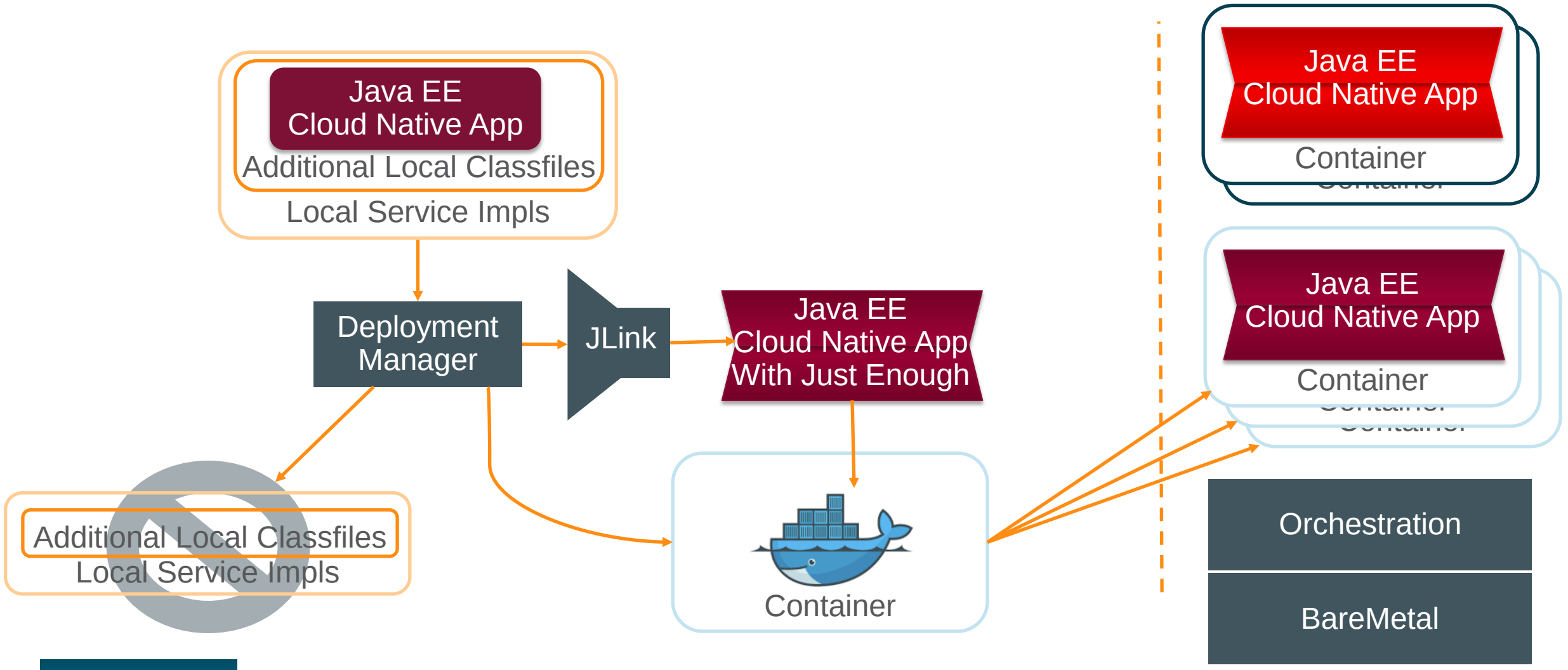
Packaging for Simplicity

- Packaging – Docker model
  - Package applications, runtimes into containers
  - Separate service package and configuration
  - Standalone immutable executable binary
  - Multi-artifact archives, leveraging Java 9
- Serverless
  - New spec – interfaces, packaging format, manifest
  - Ephemeral instantiation
- Multitenancy
  - Increased density
  - Tenant-aware routing and deployment



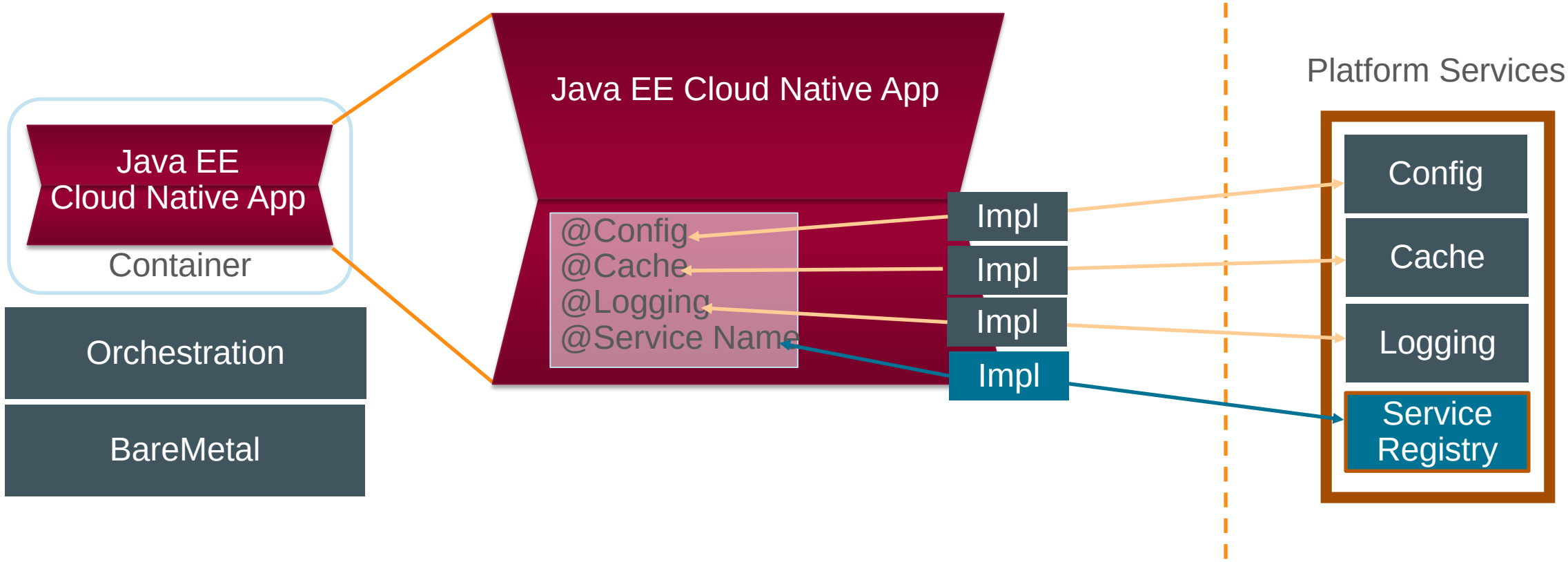
Platform Architecture

Orchestration – Deployment, Scheduling, and Standup



Platform Architecture

Orchestration – Inspection, Injection, and Wiring



Technical Focus Areas

Programming Model

- Extend for reactive programming
- Unified event model
- Event messaging API
- JAX-RS, HTTP/2, Lambda, JSON-B, ...

Packaging

- Package applications, runtimes into services
- Standalone immutable executable binary
- Multi-artifact archives

Key Value/Doc Store

- Persistence and query interface for key value and document DB

Eventual Consistency

- Automatically event out changes to observed data structures

Serverless

- New spec – interfaces, packaging format, manifest
- Ephemeral instantiation

Configuration

- Externalize configuration
- Unified API for accessing configuration

Multitenancy

- Increased density
- Tenant-aware routing and deployment

State

- API to store externalized state

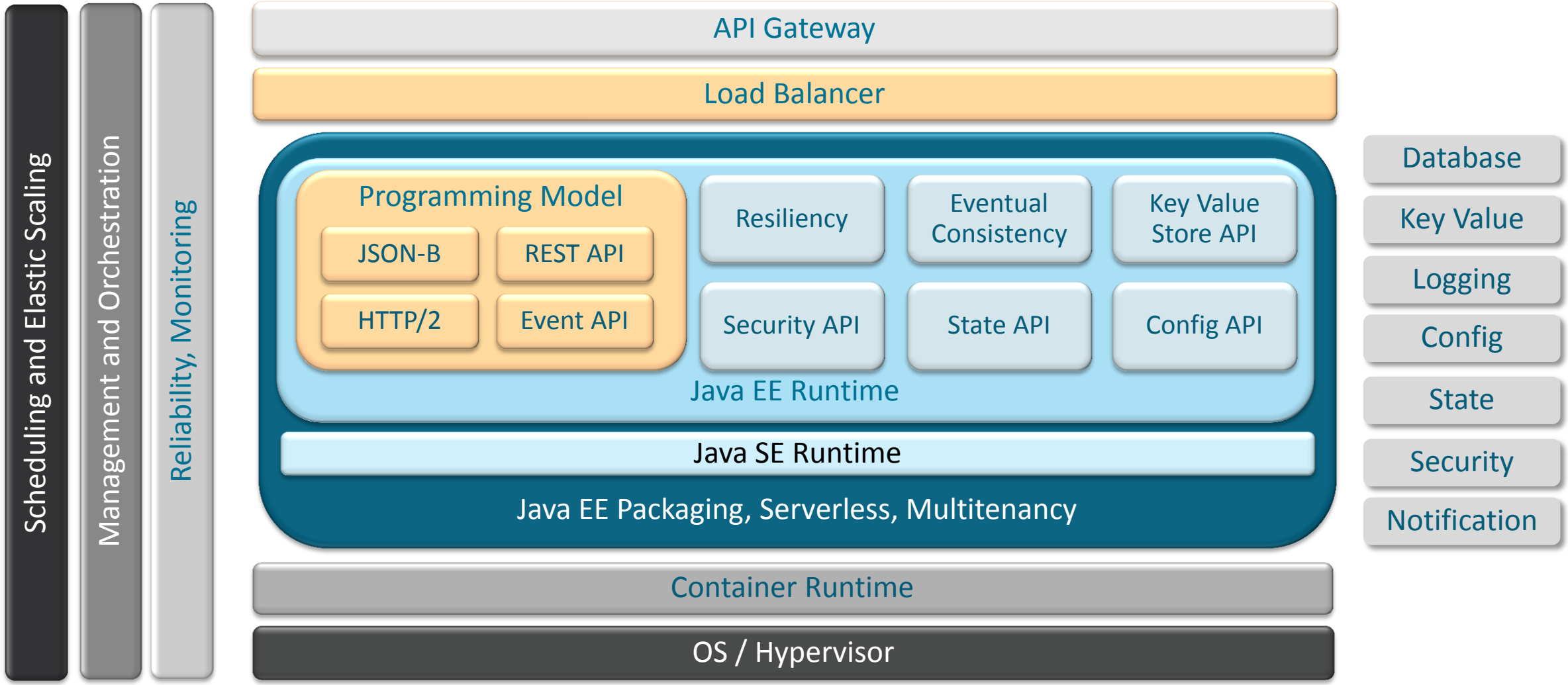
Resiliency

- Extension to support client-side circuit breakers
- Resilient commands
- Standardize on client-side format for reporting health

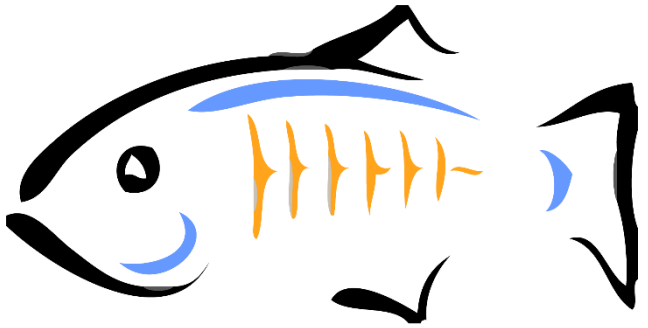
Security

- Secret management
- OAuth
- OpenID

Proposed Platform Architecture

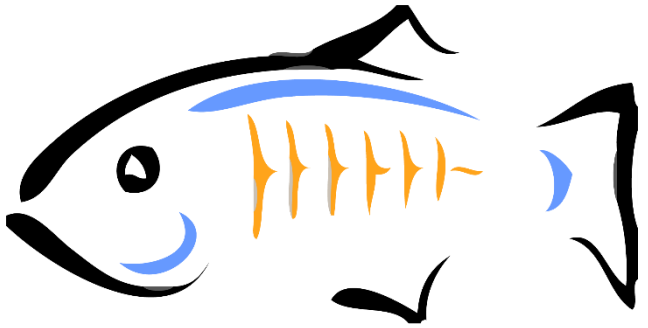


Java EE 7



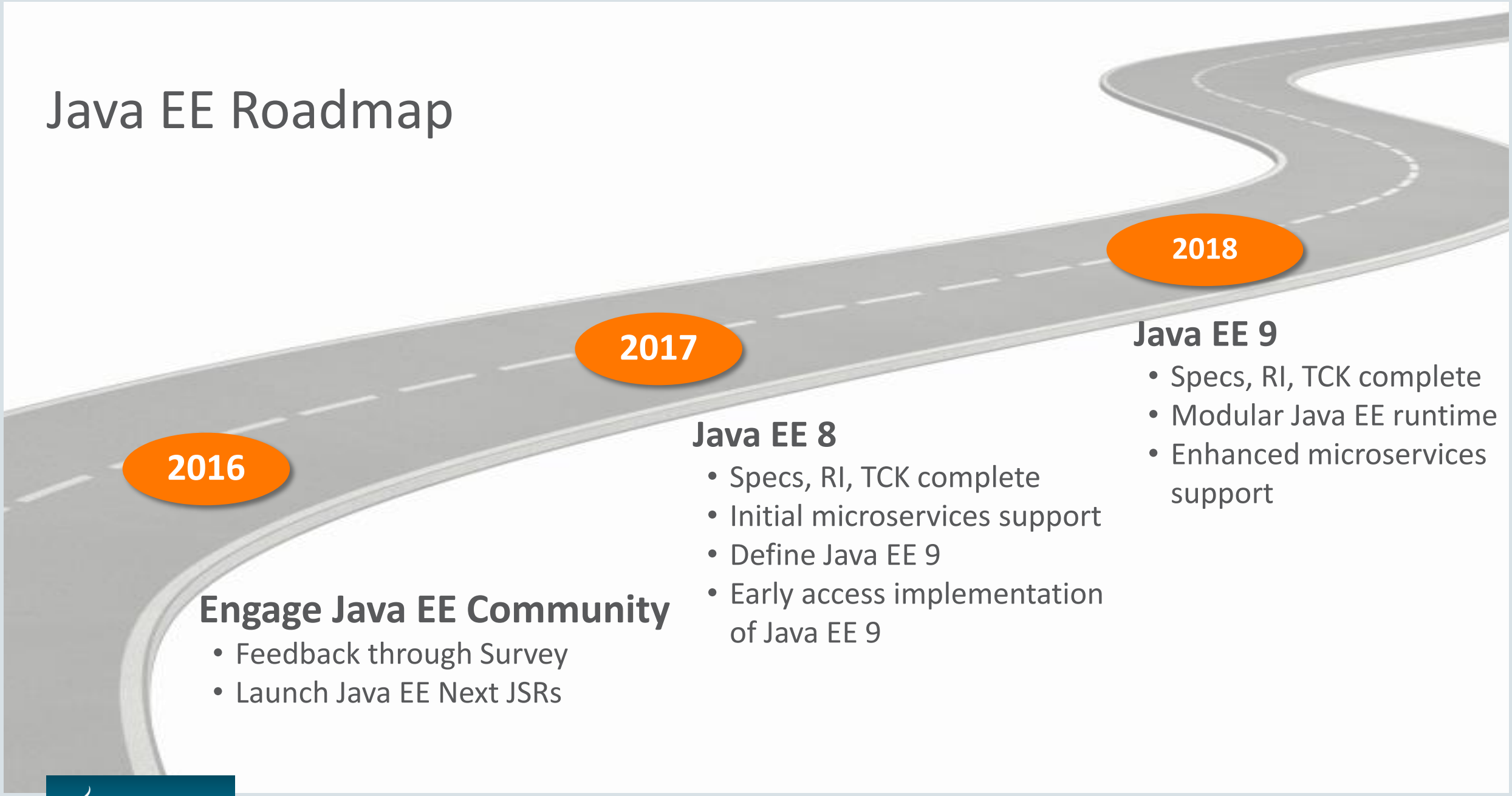
Batch	Dependency Injection	JACC	JAXR	JSTL	Management
Bean Validation	Deployment	JASPIC	JMS	JTA	Servlet
CDI	EJB	JAX-RPC	JSF	JPA	Web Services
Common Annotations	EL	JAX-RS	JSON-P	JavaMail	Web Services Metadata
Concurrency EE	Interceptors	JAX-WS	JSP	Managed Beans	WebSocket
Connector	JSP Debugging	JAXB			

Java EE 8 (Revised Proposal, 2016)



Batch	Dependency Injection	JACC	JAXR	JSTL	Management
Bean Validation	Deployment	JASPIC	JMS	JTA	Servlet
CDI	EJB	JAX-RPC	JSF	JPA	Web Services
Common Annotations	EL	JAX-RS	JSON-P	JavaMail	Web Services Metadata
Concurrency EE	Interceptors	JAX-WS	JSP	Managed Beans	WebSocket
Connector	JSP Debugging	JAXB	JSON-B	Security	

Java EE Roadmap



2016

Engage Java EE Community

- Feedback through Survey
- Launch Java EE Next JSRs

2017

Java EE 8

- Specs, RI, TCK complete
- Initial microservices support
- Define Java EE 9
- Early access implementation of Java EE 9

2018

Java EE 9

- Specs, RI, TCK complete
- Modular Java EE runtime
- Enhanced microservices support

ご質問・ご相談等ございましたら、終了後もお受けしております

あなたにいちばん近いオラクル

Oracle Digital

0120-155-096

（平日9:00-12:00 / 13:00-18:00）

http://www.oracle.com/jp/contact-us/overview/index.html

Oracle Digital 検索

各種無償支援サービスもございます。


Java™
ORACLE®