



MICROPROFILE WITH HELIDON

dhananjayan

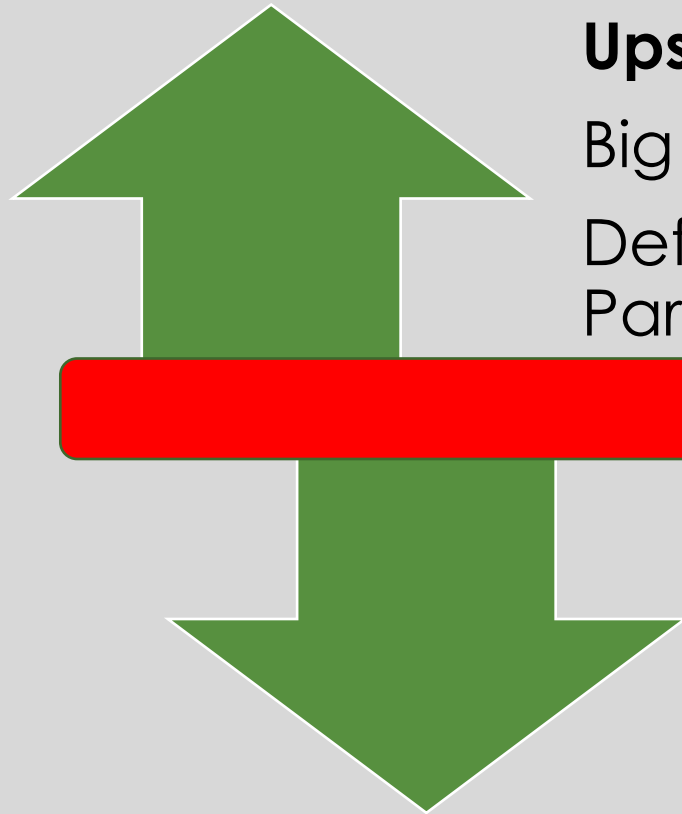
Breaks

- 11.30 – 11.50
- 1 – 2
- 3.40 – 4.00

Pre-requisite

- Name, Role and Objective
- Working Knowledge of Java EE
- REST / Services Architecture - Developing /Deploying /Debugging
- Monitoring Tools – Awareness
- Awareness to Cloud Computing
- Awareness to Container Architecture

View – Agile Projects



Upstream → Integration

Big Picture - Leverage

Define Contract for Service,
Parameters

Down stream

Implement the service capability

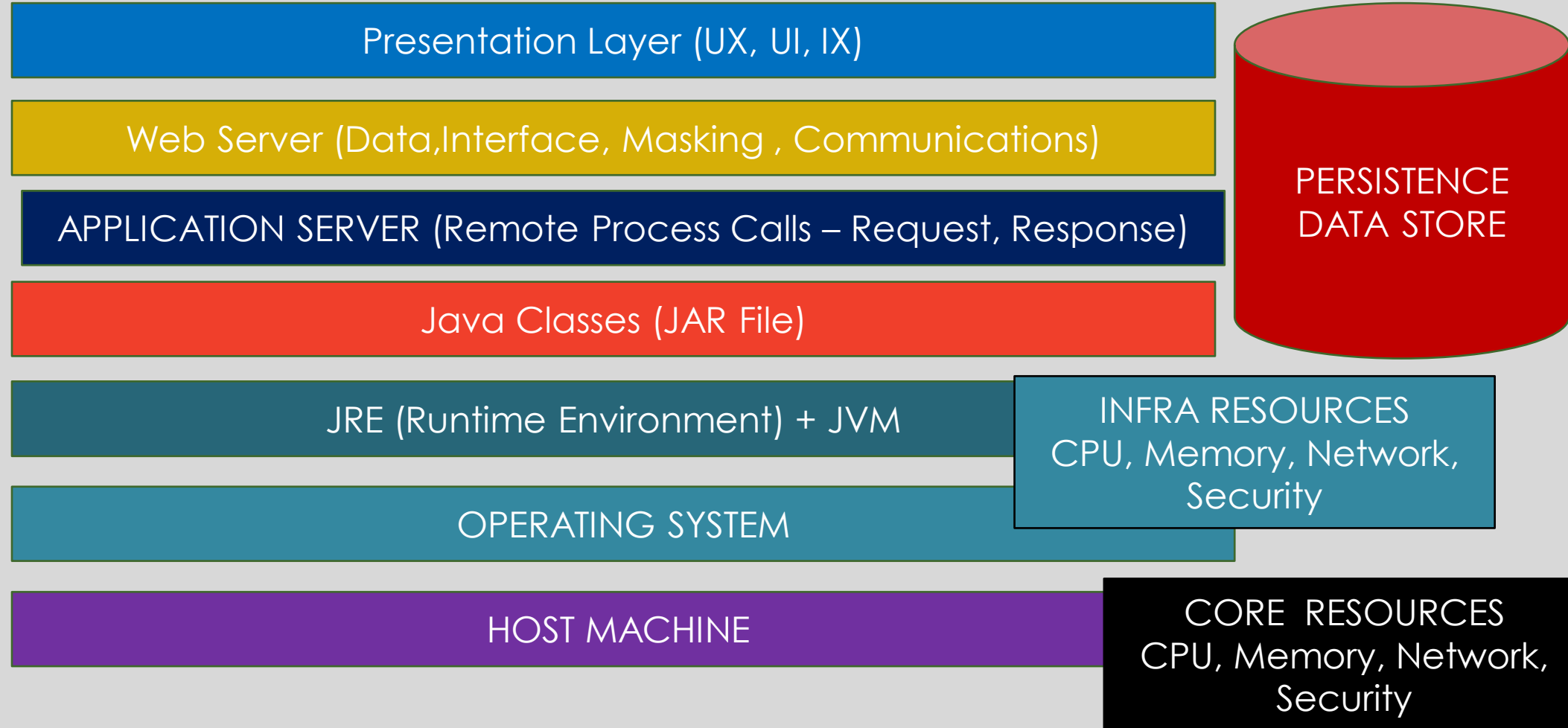
Developing capability

Architectural Styles

- Data Centered Architecture
- Data Flow Architecture
- Call and Return Architecture (changes)
- Object Oriented Architecture
- Layered Architecture (n-tier)
- Event Driven Architecture (EMA)
- Monolithic (Legacy)
- **Microservices Architecture**

Applications

Tightly Coupled
Separation of Concerns
Change – Code goes to ICU ?



Cloud Application

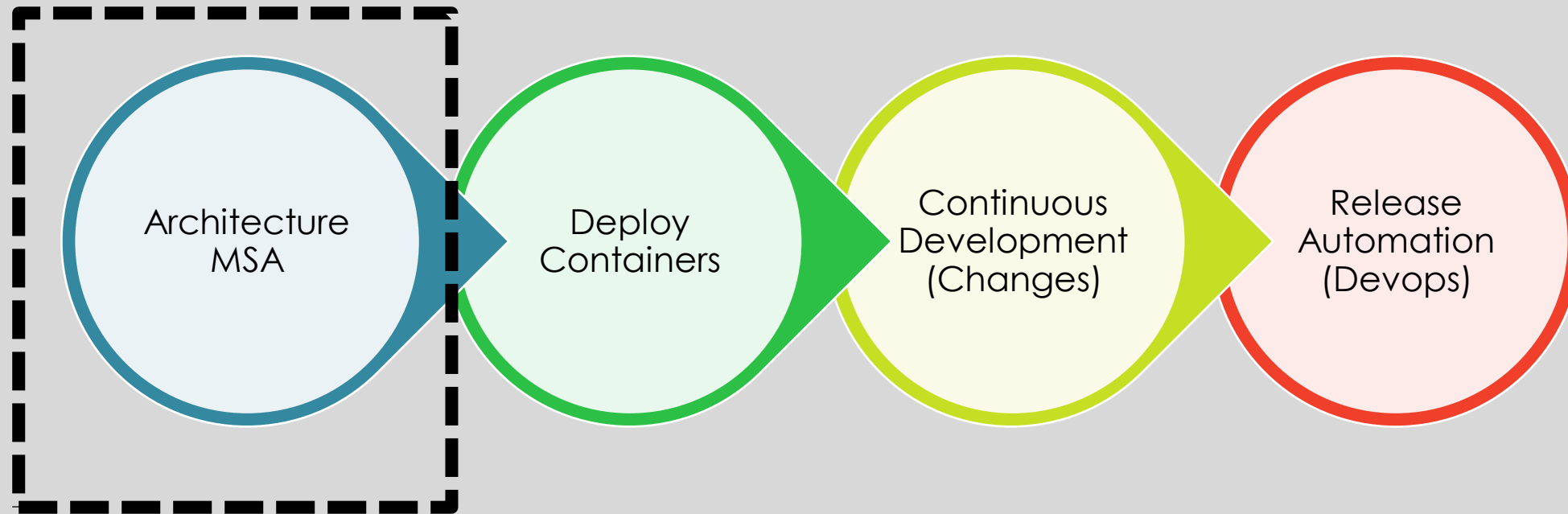
Highly Available (Capabilities) / (Infrastructure)

Adapt to changes Frequently – Uptime (Available)

Optimize the cost of Infrastructure – No Big Bang Approach, Pay per Usage

NO-Compromise on Performance (Application) & Security

Cloud Native Applications



Evolution

App Delivery	App Design	App Deploy	App Host or Integrate
All in one Release Minimize Release Maximise Features	Tightly coupled (Monolithic)	Host Machine (BareMetal Server Boxes)	Farm – Data Center
<u>Boxed</u> Releases Incremental Release More Releases, Less Features (Agile in Dev/QA)	Layered Architecture Persistent DB Interconnected Service Services are Independent	Virtual Machines (Workloads) Tightly coupled	On -Premises
Agile in Dev and Ops (Devops) Constant Evolution	Loosely Coupled – Independent Service (Microservice Architecture)	Develop once Deploy Anywhere Container (Independent processes)	Cloud Infrastructure

Cloud Native Application Consideration

- Portability (Develop Once Deploy anywhere)
- Define the scope of Independence (Slide 11)
- Capability Change (Code change) is only change
- Faster Changes (Move Faster, Market Faster) and Automate Fast changes
- Automation changement – Release management processes

Logical Meaning of Microservices

1

Independent by Definition (JAR,WAR or TAR)

2

Independent by Deployment (Containers or VM or Host Machines)

3

Independent by Scale (Replicate – Multiple Copies – No SPOF*)

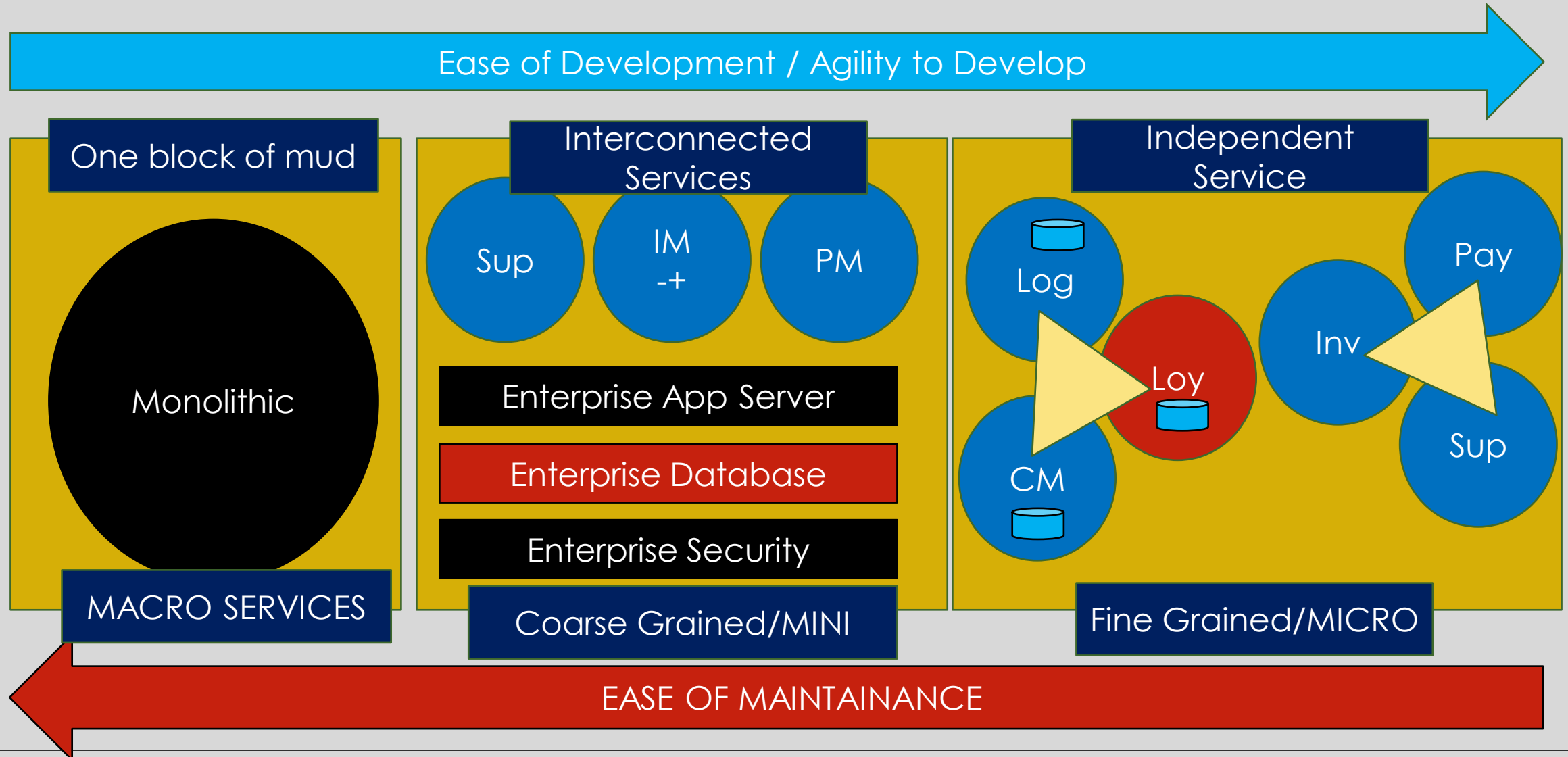
4

Independent by Test (Test the Individual Capability – Health, Trace and Verify)

5

Independent by Data (Data Structure is away from RDBMS)

Fine Grained Applications..



CNCF

- Open Source Framework
- Fine grained Architecture
- Dynamic Orchestration
- Dynamic Resource management
- In house / off –the-rack
- Integrator IT , Ops and Services

Cloud Native Application Principles

- (1) **Loosely Coupled**
 - Suite of Services (I/O)
 - Runs on it own process (container in real time)
 - Communicate through Light weight – http
 - Built for Business capability
 - Independently Deployable
 - Supported by Automation

CN Principles...

- (2) **Multiple Frameworks – Develop and Breed**
 - Multi Framework Development
 - Polygot (Multiple Datastructures)
 - Common runtime across selective platforms (native runtime) like graalVM
 - ML Data Science (Analytics across streams) – Python
 - REST API (Java for Rest API) – Spring Boot , Java EE
 - WebSocket programming - node JS

CN Principles..

- **Centered Around API**

- Interaction
- Collaboration (Communicate)
- Use Light weight API – Http
 - REST API (Security, Header, Context, Session)
 - Binary Protocols (gRPC, Protocol Buffers) - Internal Servers
 - MQ/APMQ (Messaging , 1-#)
 - Stateless Applications

CN Principles ..

- **Resilience**

- Not to Avoid , Respond or Repair or Recupperate
- Strategies
- Ability to run

- **Continuously monitor**

- Take action on Failure
- Agile
- Elastic



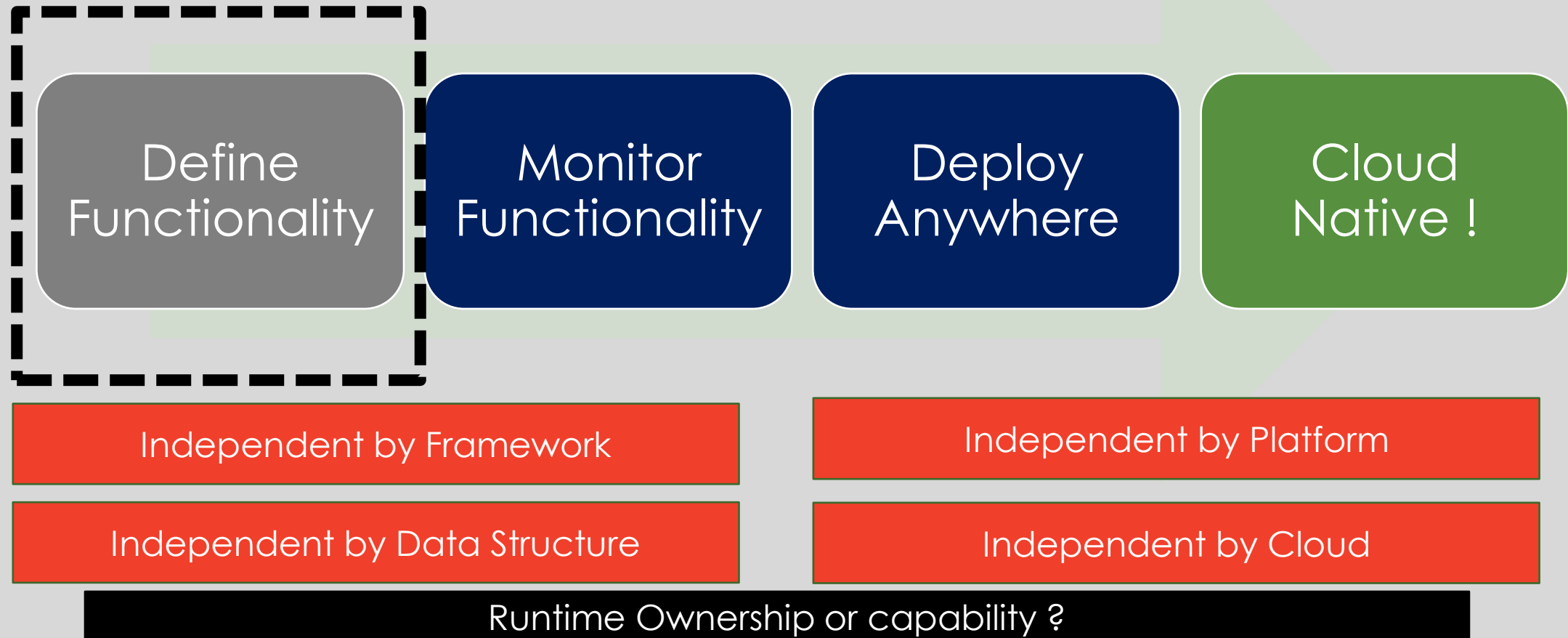
Probes on
Metrics

Metrics

Dashboard
View

Logs

Migration Transformation

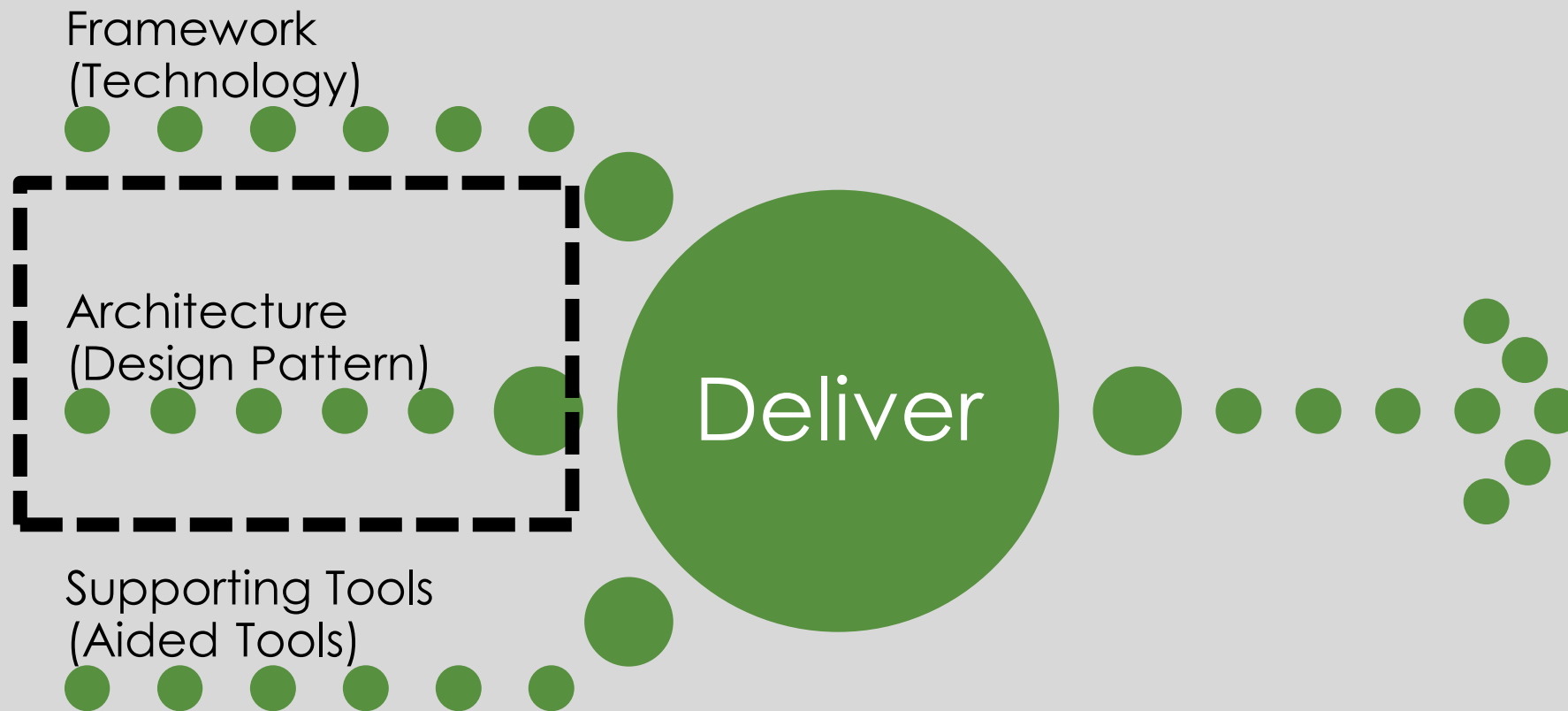


Solution Patterns

(*applicable for all design styles...)

Generating	Structural	Behavioural	Data Driven	Enterprise
+ Singleton + Factory + Clone + Prototype	+ Adapter + Façade + Decorator + Proxy + Bridge	+ Observer + Visitor + Mediator + Pub Sub + Iterator	+ Lazy Loading + CQRS + 2 PC + Shared DB	+ Container Architecture + Paging Models / Streams or Cache

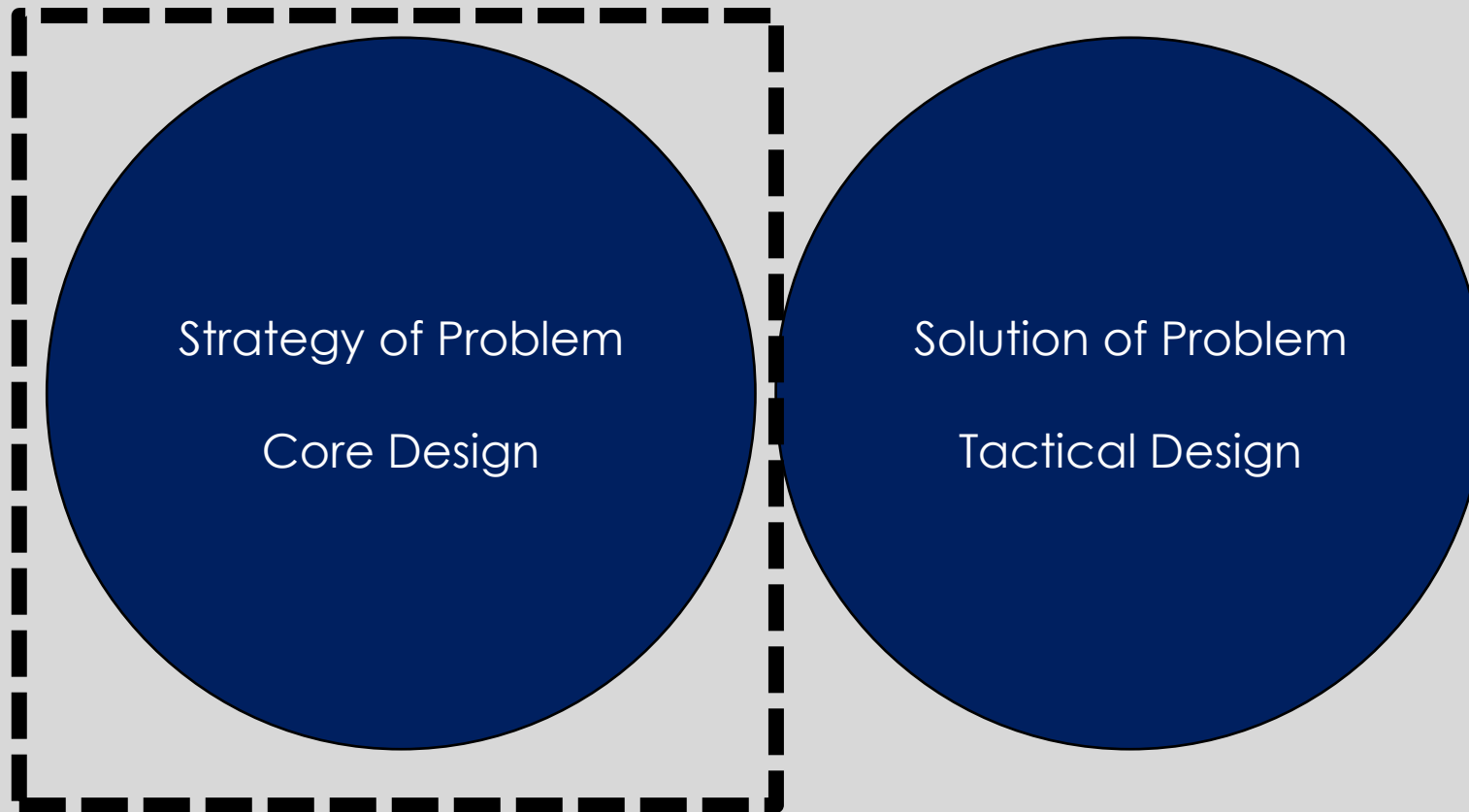
Yield Microservices



Domain Driven Design

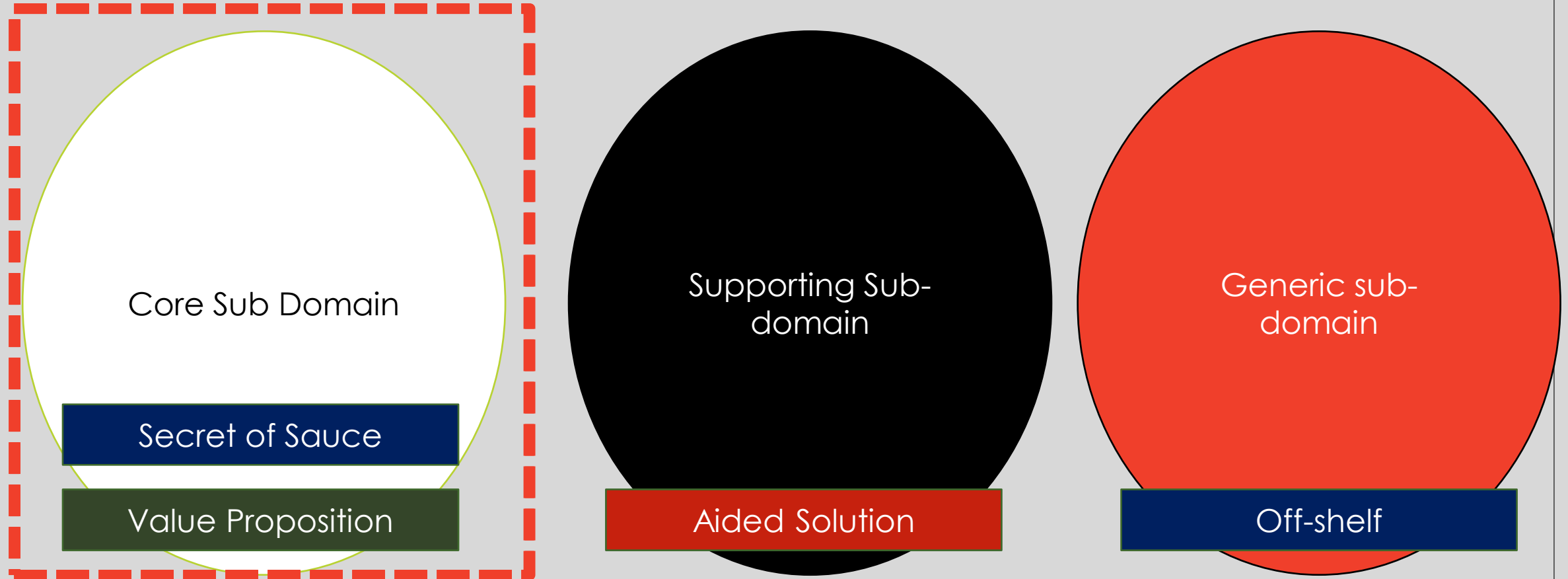
- Aid the journey in Microservices
 - Focus on the Solution not the Technology (**ubiquitous Language**)
 - Focus on Business Capabilities (Problem)
 - When not to divide, when to divide
 - Dependency Services (Strongly Coupled) – Attached Services
 - Independent Services (Loosely Coupled)

Domain Driven Design

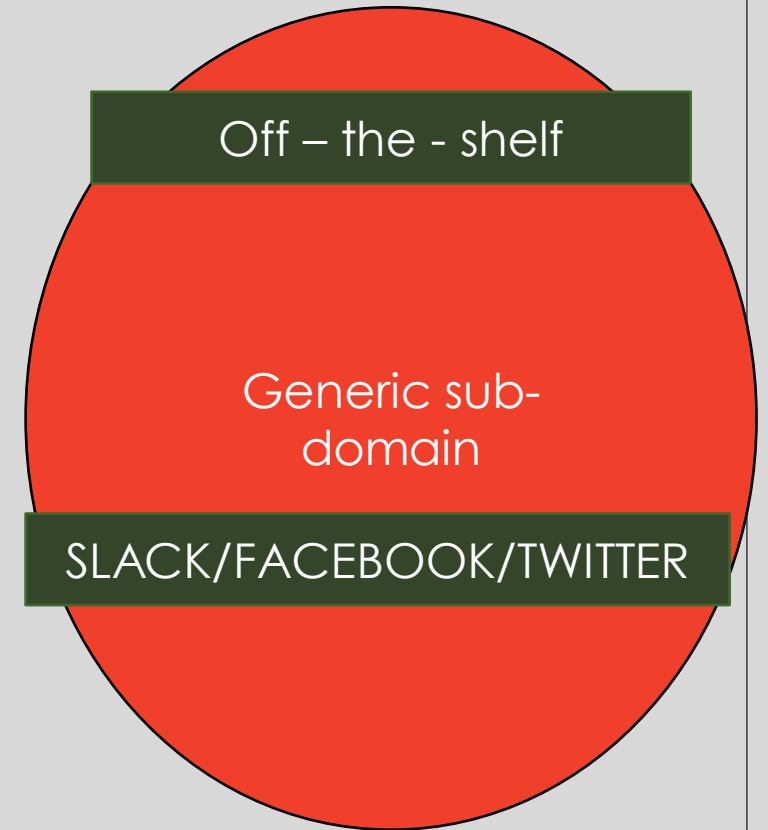
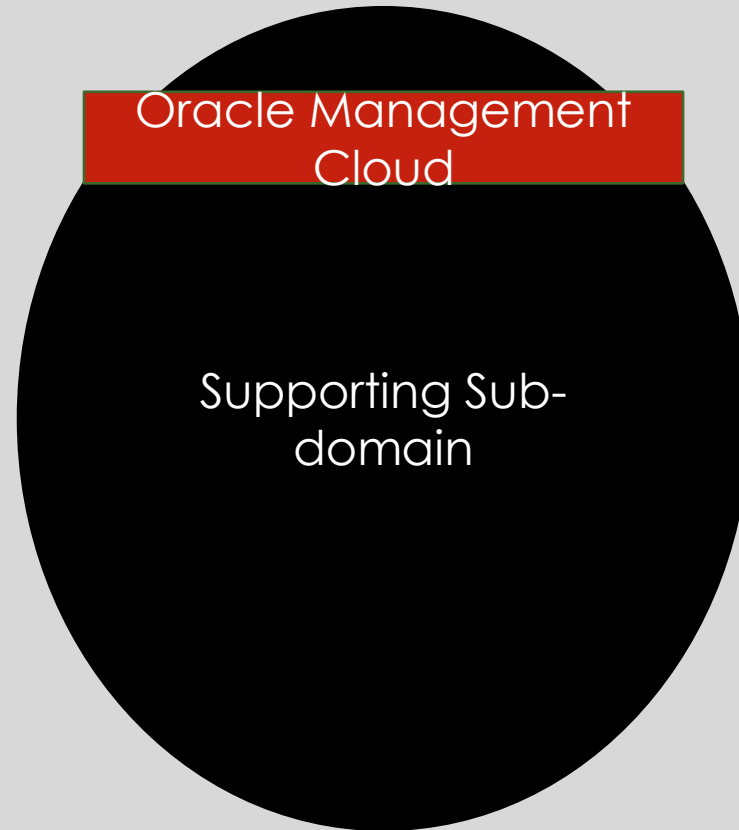
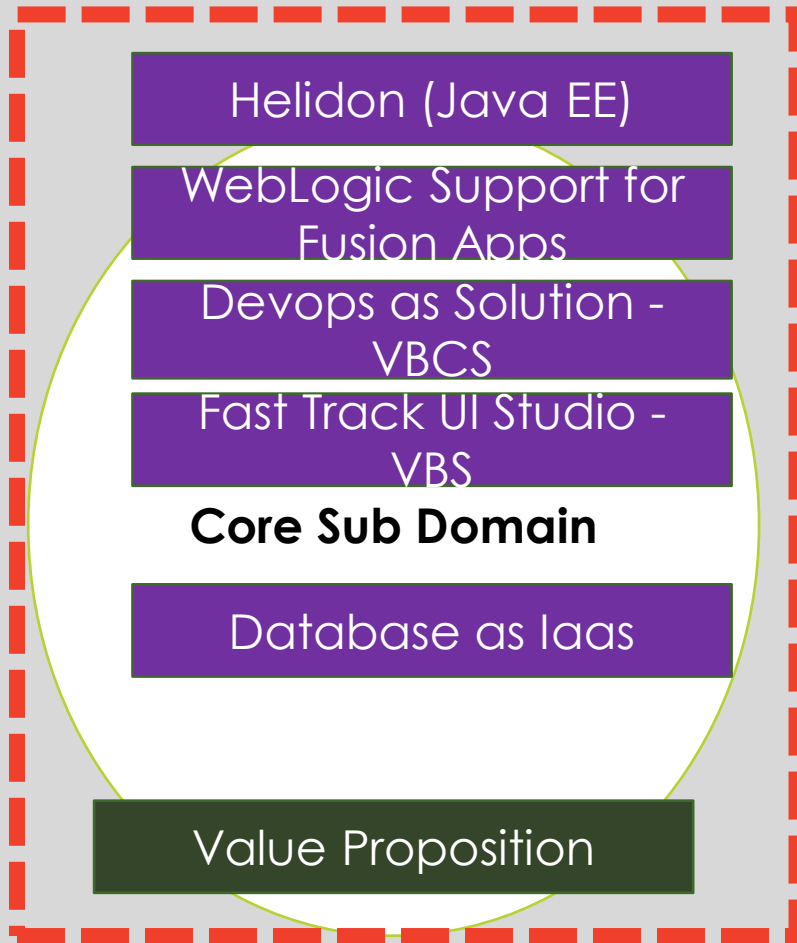


Strategy of the Problem

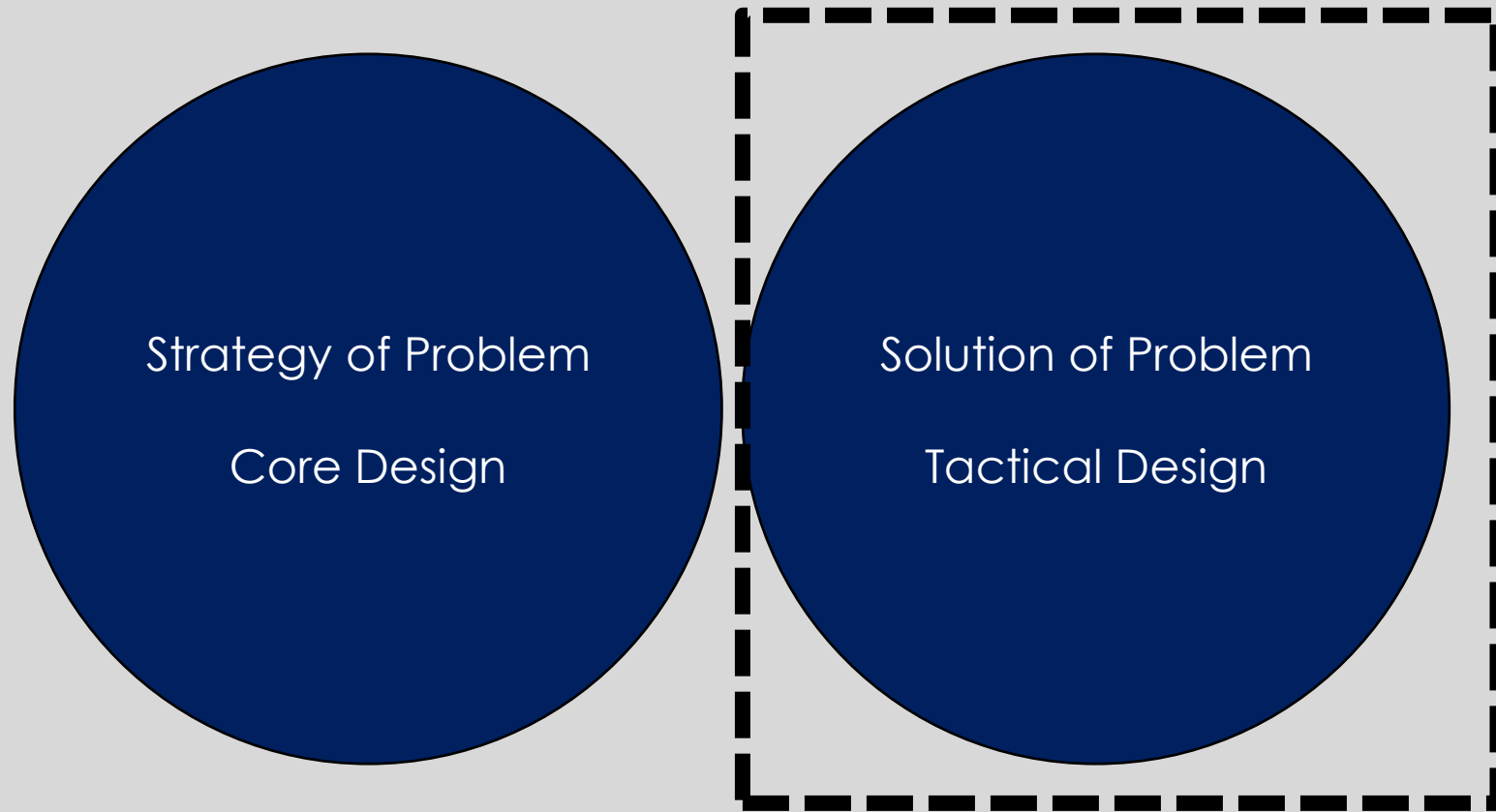
What to Solve ?



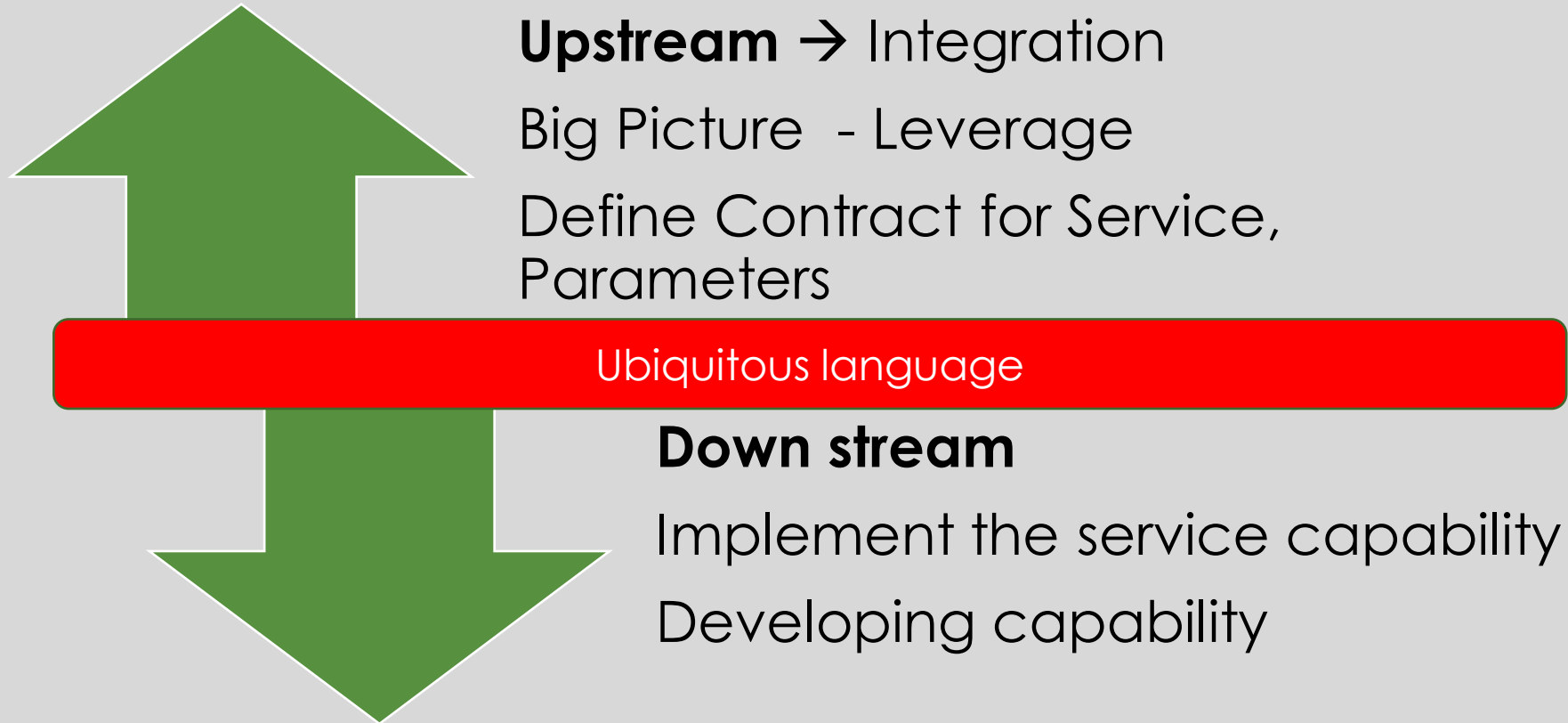
Oracle Cloud Infrastructure (Dev) ?



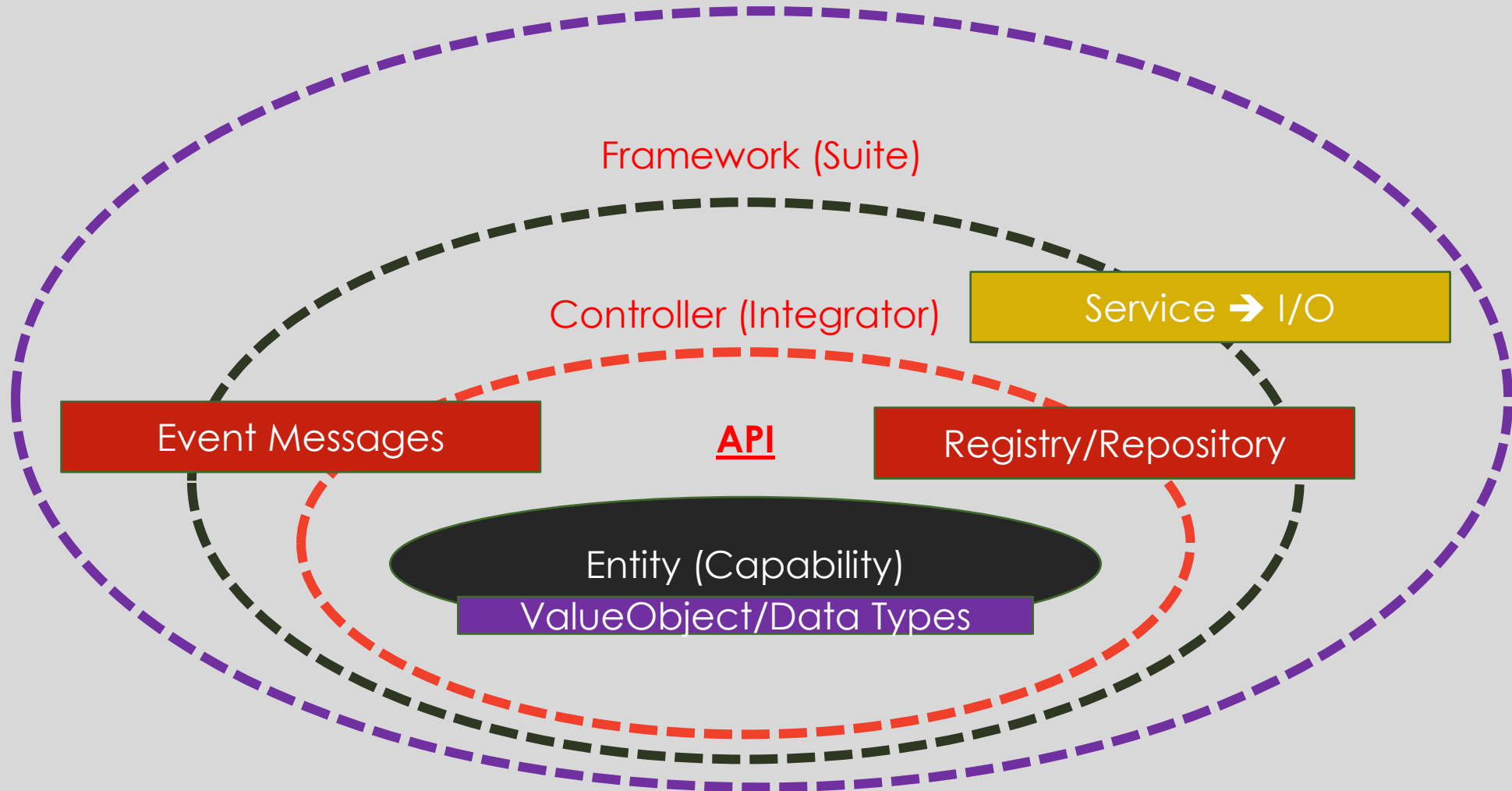
Domain Driven Design



View – Agile Projects



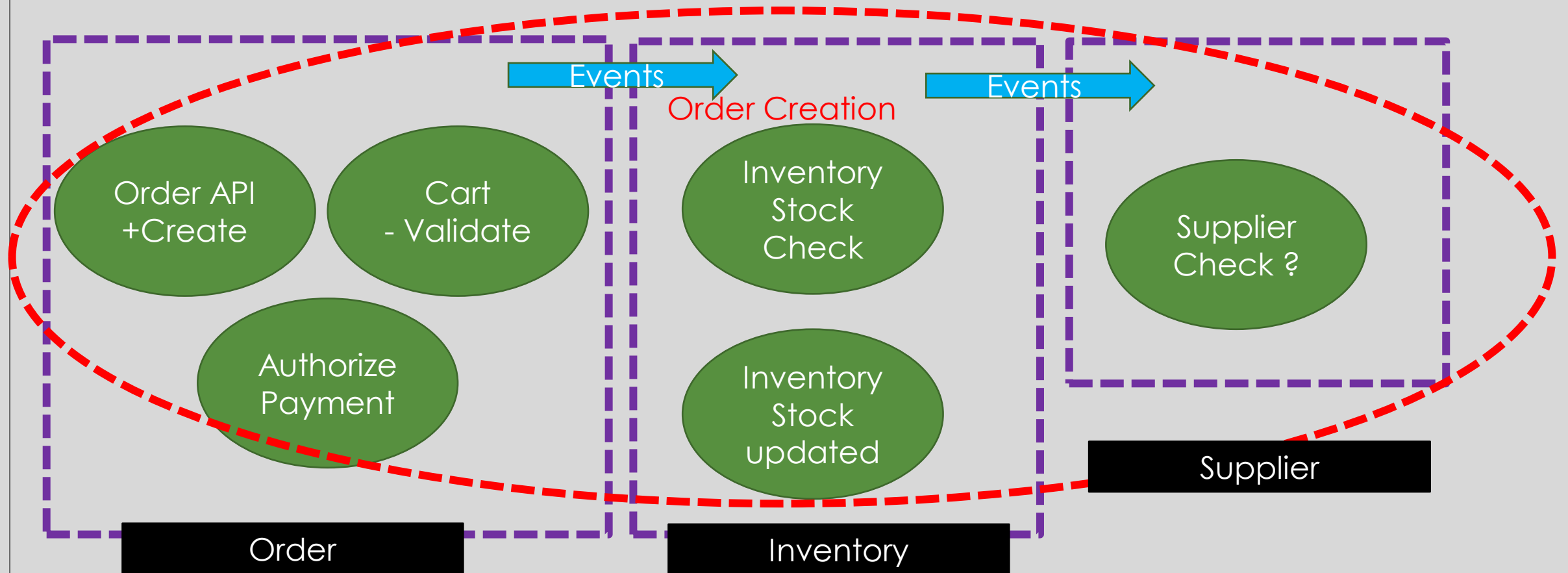
Tactical Part of Problem



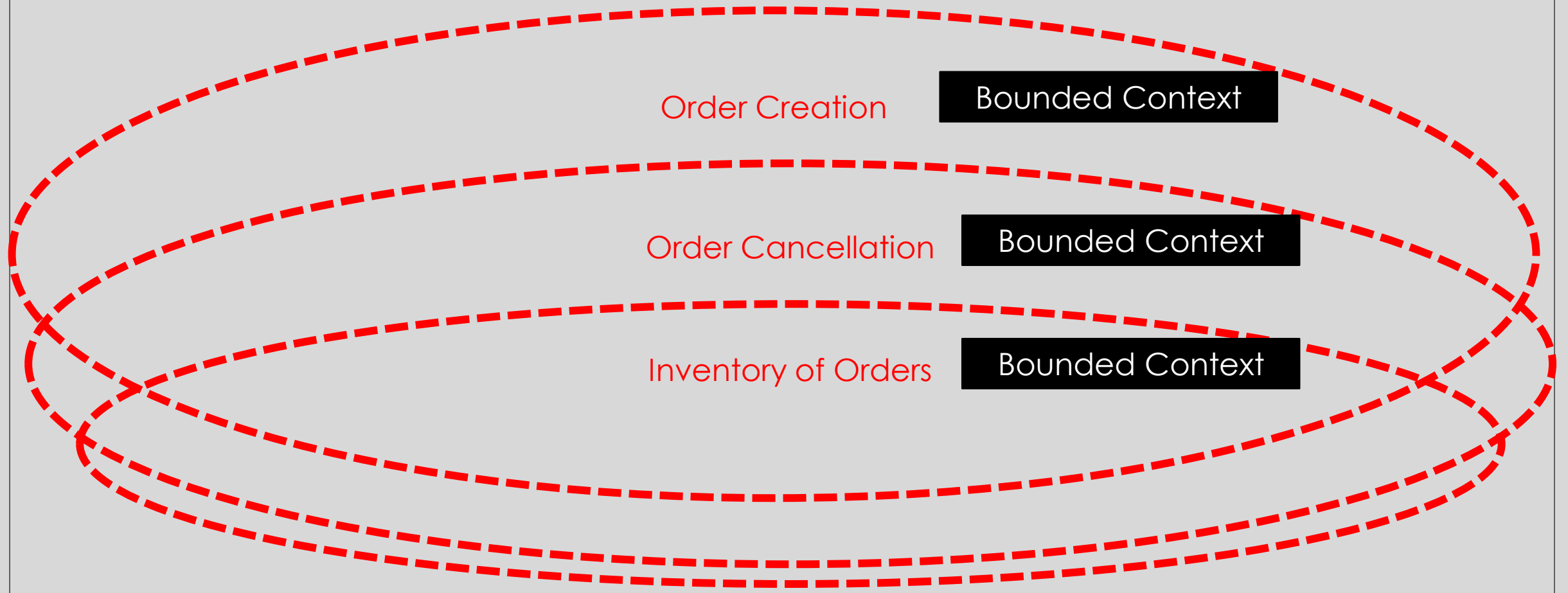
Bounded Context

Entities in a common solution delivers a functionality

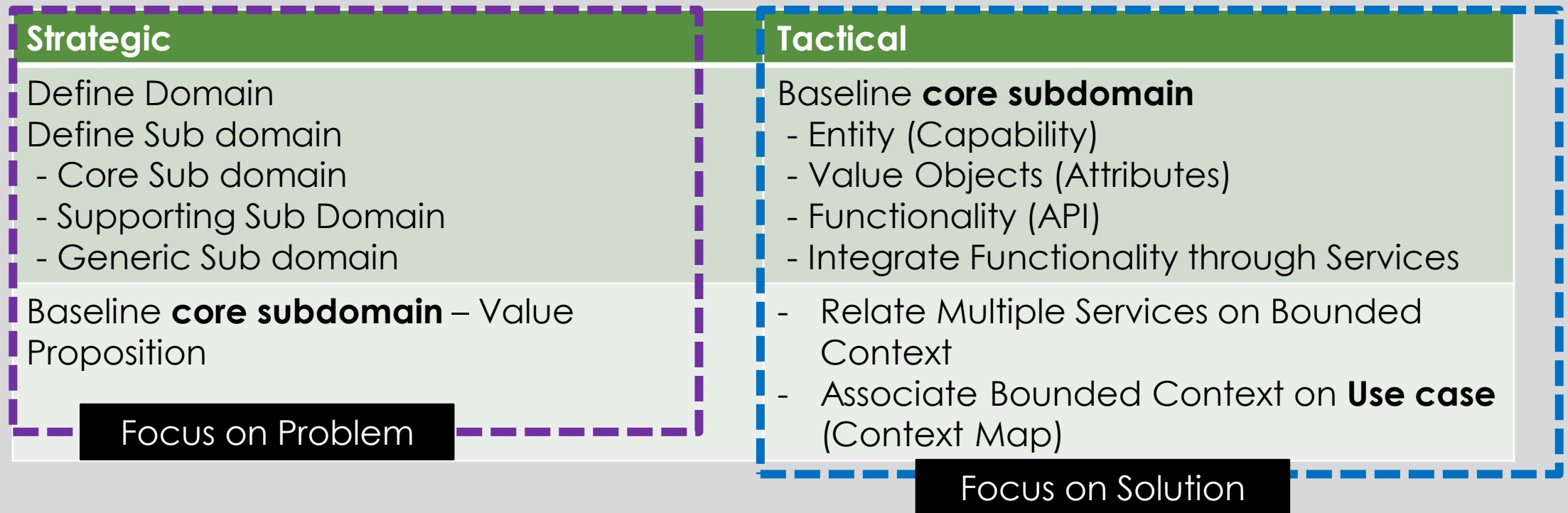
Order Creation (use case)



Context Map → Related Bounded Context



DDD for Microservices



Infrastructure setup pre-requisite

For users with Maven

In ~/.m2
(Rename this
directory)

Install JDK

Java JDK
16

SET JAVA_HOME

Install Maven

Maven
3.6+

#mvn --version

Productivity Tools

Windows – GITBASH
Mac - Terminal

IDE – Eclipse IDE



Provisioning from Development...

Scheduling /Orchestration	Runtime
Service Communication Service Gateway /API Gateway Service Proxy Service Mesh	Where is your Mount point (what data) Container Network Security/Keys/Values/Vault Automation and Configuration

12 –app factors for **microservices**

Project Properties

- Code to manage versions - Code Base (SCM Tools – GitHub, Bit Bucket) VSTS - versioning
- Declare all your dependencies (isolate your dependencies) – Maven or Gradle
- Configuration of Parameters or Resources (Isolate Configure Resources) (Env Variables)

12 –app factors (2)

Run Properties

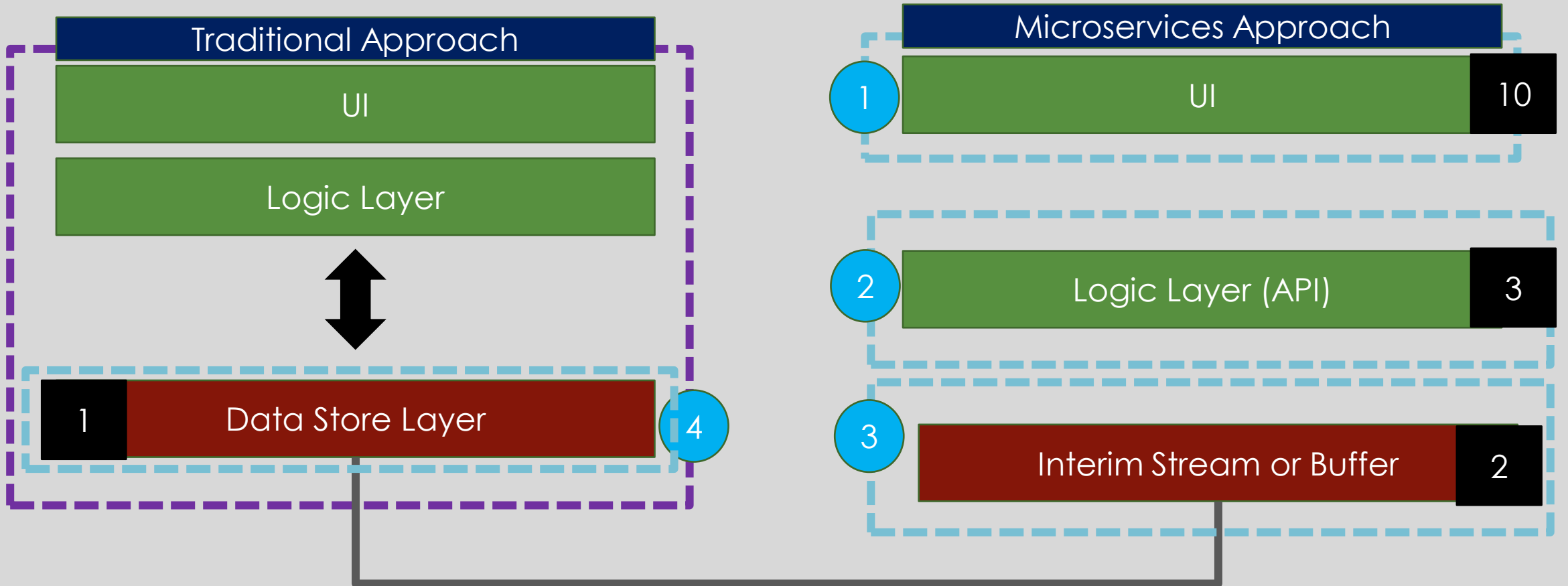
- Treat Dependent Services as Attached Services
- Build, Deploy and Release should be treated in same environment/as separate stages
- Run your Microservice application as Independent Process (Container)
- Port Bindings – Expose/Port Forward/Publish Rest calls
- Concurrency – Scale out as Process model
- Disposability – Start up and Shutdown (Garbage Collectors)

12-app factors

Management Properties

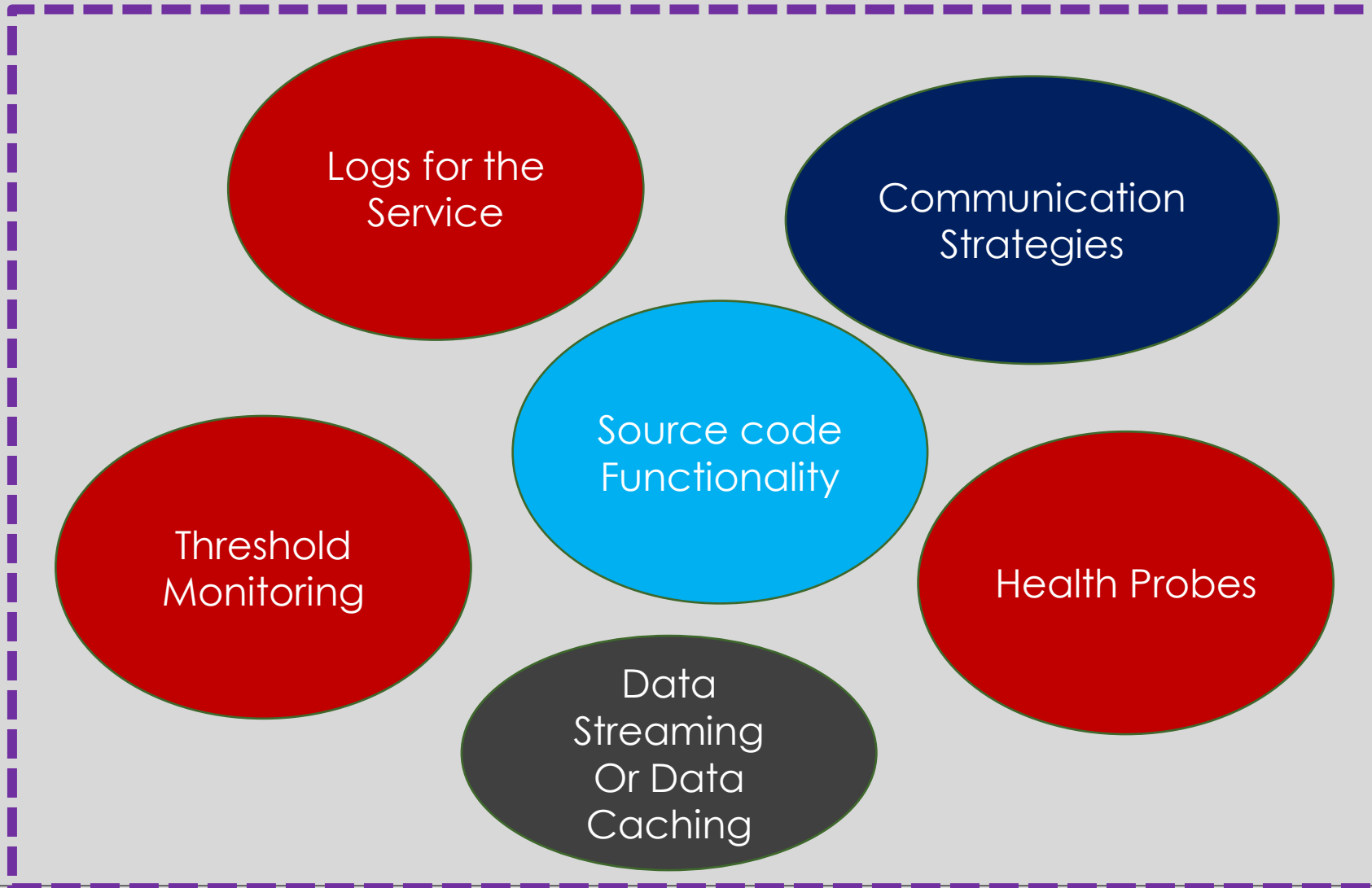
- Collect Data (Logs), Evaluate Metrics , Keep Monitoring (Resilience)
- Administration Console or Configuration Management / Application Management
- Code base versioning → Staging for Builds for Versioning (Rollback on changes)

SDU (Single Deployable Unit)



Comprehensive Framework

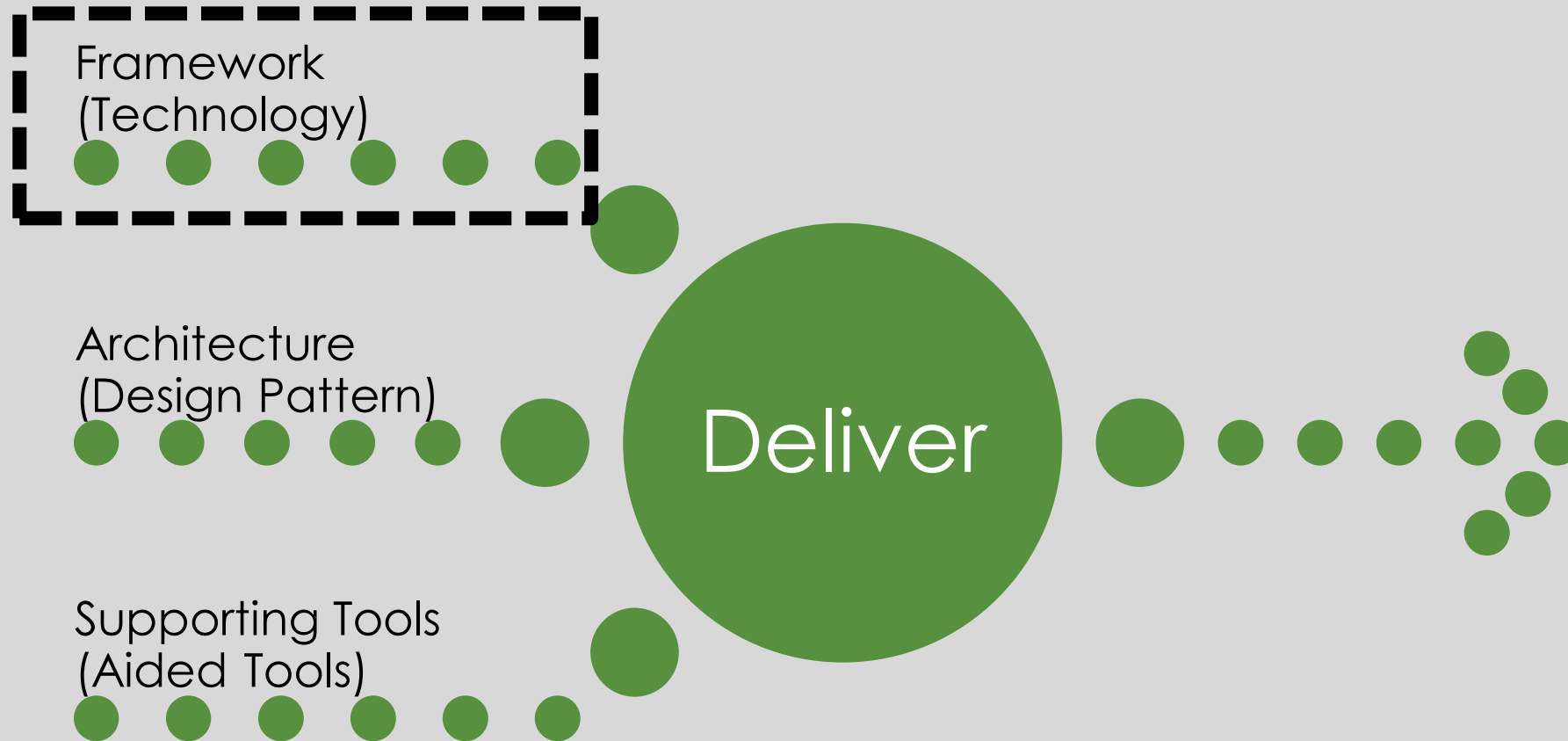
Interceptor Services



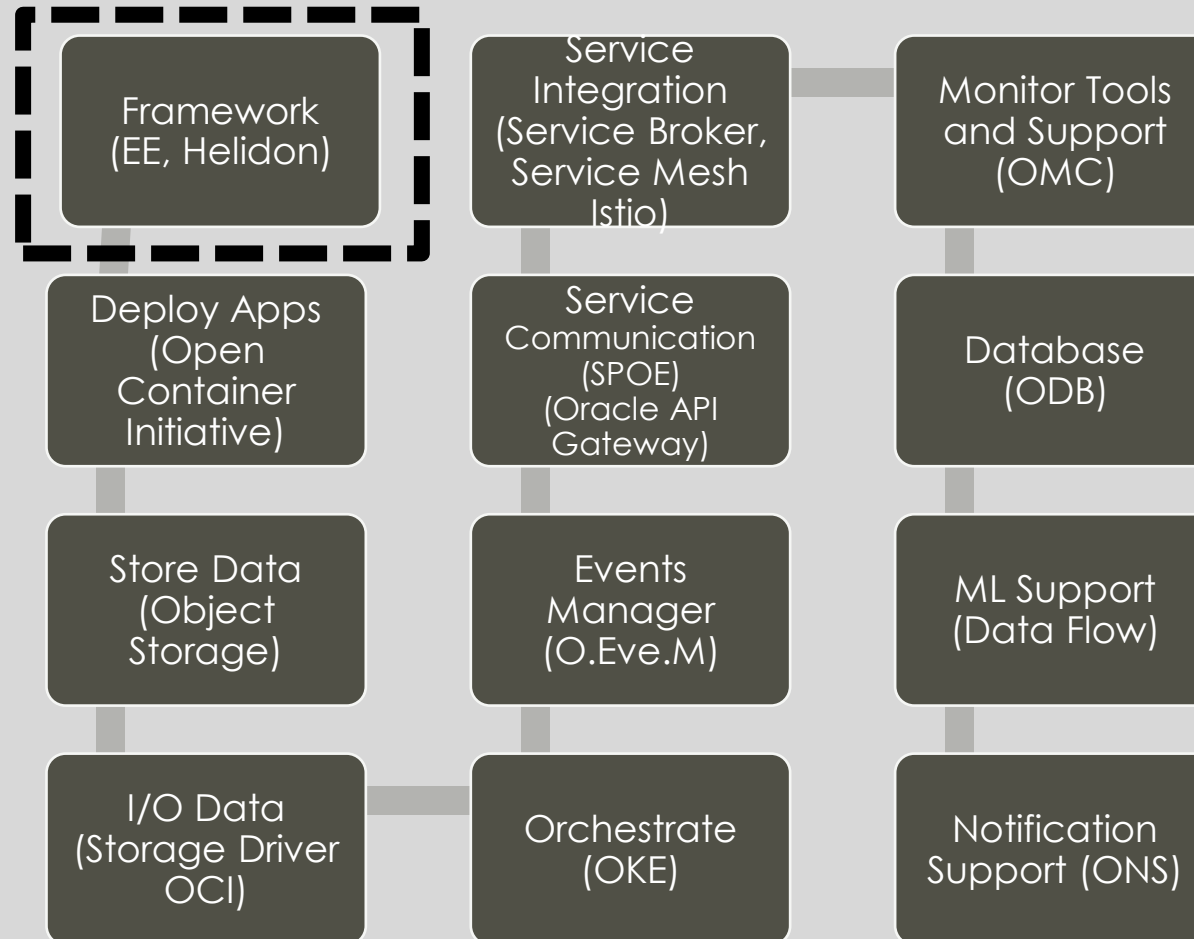
Day 2

- Microservices with OCI
- Polygot Approach
- Scale Cube
- Maturity Journey
- Helidon Architecture
- Helidon Approach
- Helidon Components – CDI , JSON, Healthchecks (Microprofile)
- Configure Server /Environments

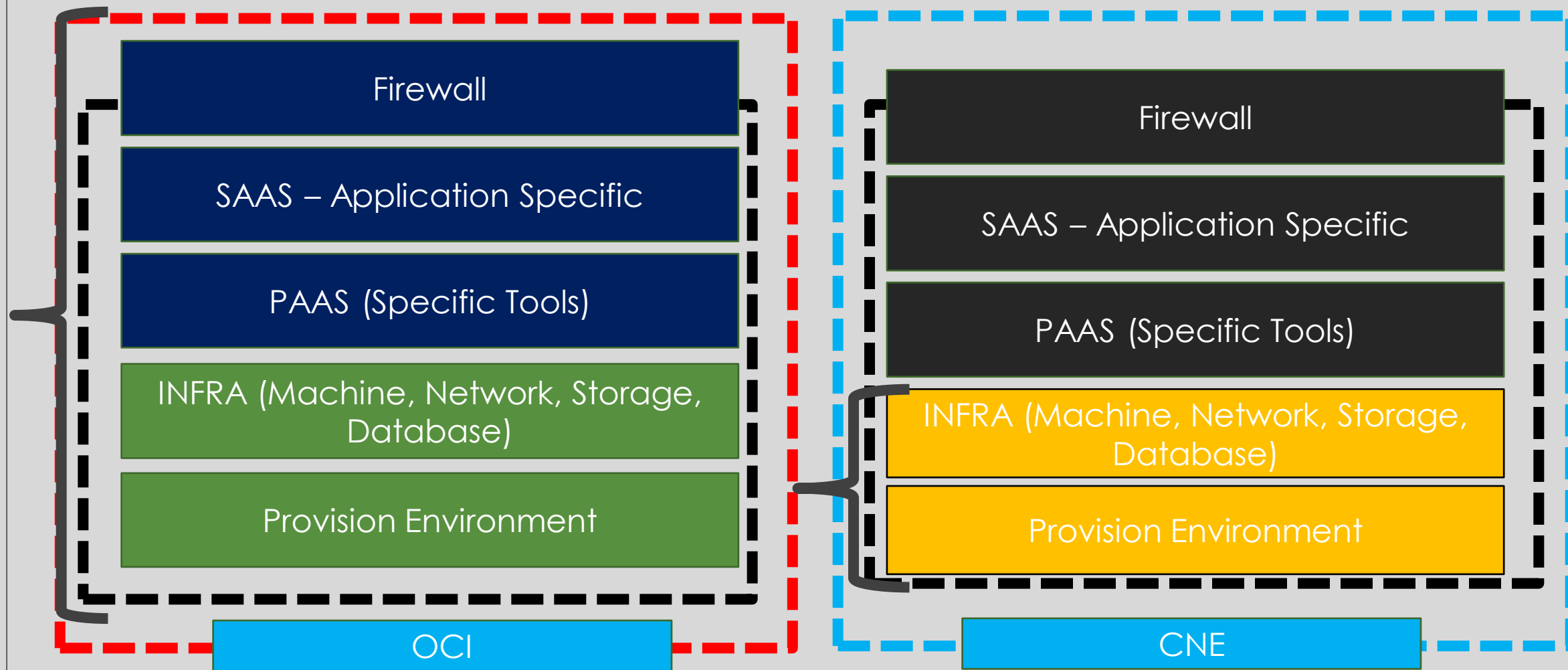
Yield Microservices



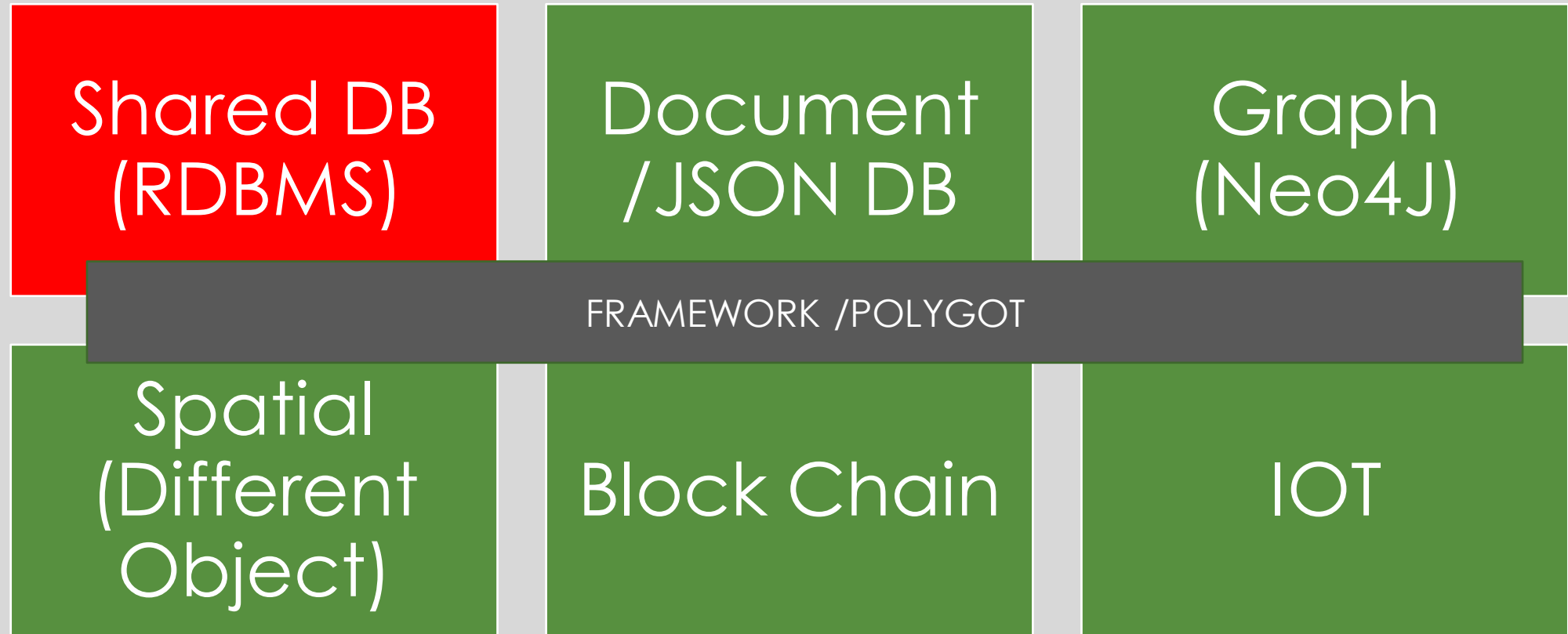
Integration with OCI (MSA)



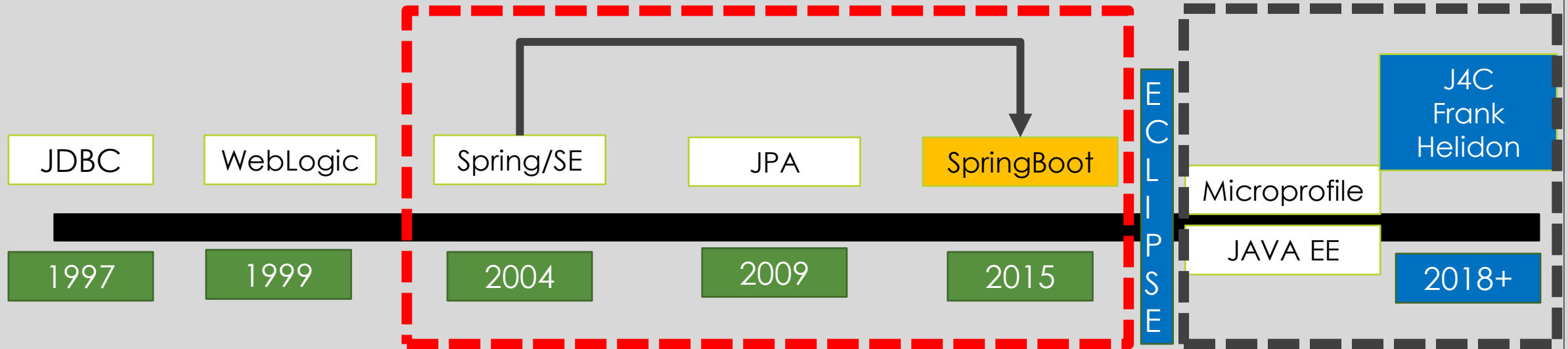
OCI vs CNE



Polygot Approach



Helidon



SE vs MP (properties)

Standard Edition	Micro-profile
Procedural	Declarative Format
Define Milestones → Achieve Execution Plan → Rescue Factor (Exception Handling) → Achieve Outcome	<u>Define Milestones → Define Outcome</u> → Runtime will create an Internal SE code to reach this outcome
Clear Transparency – Defined API / Methods, Parameters	Annotations (CDI) → Outcome OCI (Resource manager)
Portability of Applications / Rewrite	Portability / Rewrite on Business changes

Technology Levers

Hybrid

FULL STACK DEVELOPMENT : MONOLOTHIC, MICROSERVICES, LAYERED ARCHITECTURE
Spring , SpringBoot, Procedural (SE) , OpenJDK

CNA

CLOUD NATIVE APPLICATIONS : MICROSERVICES
Declarative , Fast Track : Helidon MP, Open Liberty

Micro

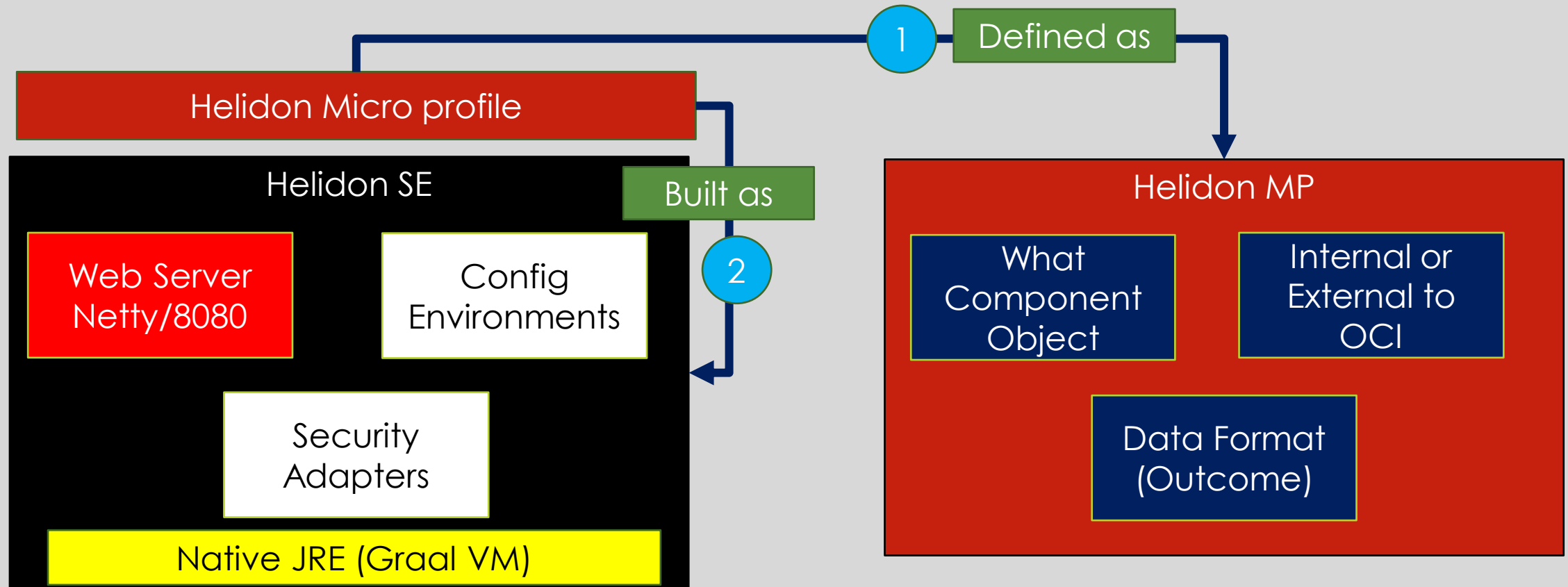
Non-CLOUD , CLOUD NATIVE APPLICATIONS : MICROSERVICES++
Procedural : Helidon SE ,Javalin , SPARK , Micronaut (Serverless)



Helidon SE vs Helidon MP

Helidon SE	Helidon MP
Functional/Procedural Development	Declarative Development
Micro framework (CNA, Layered)	CAN, Microprofile
Reactive (Execution Plan, Exceptions Rescue) – Avoid Risks	Capability Model – Handle Risks
API , Binaries , SE	CDI (Dependency Injection), @Annotations, Java EE
Config, Database , Kafka Connectors, Streams, Jedis, Redis, Connection Pool, OCI Storage Driver , JSON Library (Explicit API to connect to External Product)	CDI , CDI Extensions – Explicit Automated Annotations , JSON (P), JSON (B)

Helidon Architecture (Java EE)



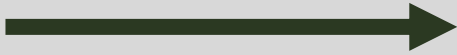
Helidon API

- Default JDK Runtime – JDK 11+, Maven 3.6+
- Other Runtime – Go, Ruby, Python, java Script (node js) , R ,(.NET)
- Simple by Structure, Developed in Modern React way
- Exclusive for Microservices , Light weight

Helidon SE

Configuration	Security	WebServer
<ul style="list-style-type: none">- Configure Environments- Data Sources- Formats (Logs/Outcome)- Extensions (Ports, Bindings)- Dynamic updates (Monitoring – tracing, metrics)	<ul style="list-style-type: none">- Connect to Security Provider (LDAP, Exchange)- Authentication- Authorization- Auditing- Keys Support	<p>X Netty</p> <ul style="list-style-type: none">- Tracing API- Reactive API- Static content support

Helidon MP

- Open source for Enterprise Java
 - Light weight , Microservice Approach
 - Open Tracing API (RCA)
 - Open API
 - Rest Clients (REST API)
 - Configure (MP)
 - gRPC /RPC
 - Fault Tolerance
 - Health Checks, Metrics, Probes
- 
- Integration with OCI (CDI Extension)
 - JWT Authentication
 - Integration with JPA
 - Integration with JTA
 - JSON Parsing and JSON Binding
 - JAX RS (REST API)



CDI

CDI

CLOUD ENVIRONMENT

HOST MACHINE

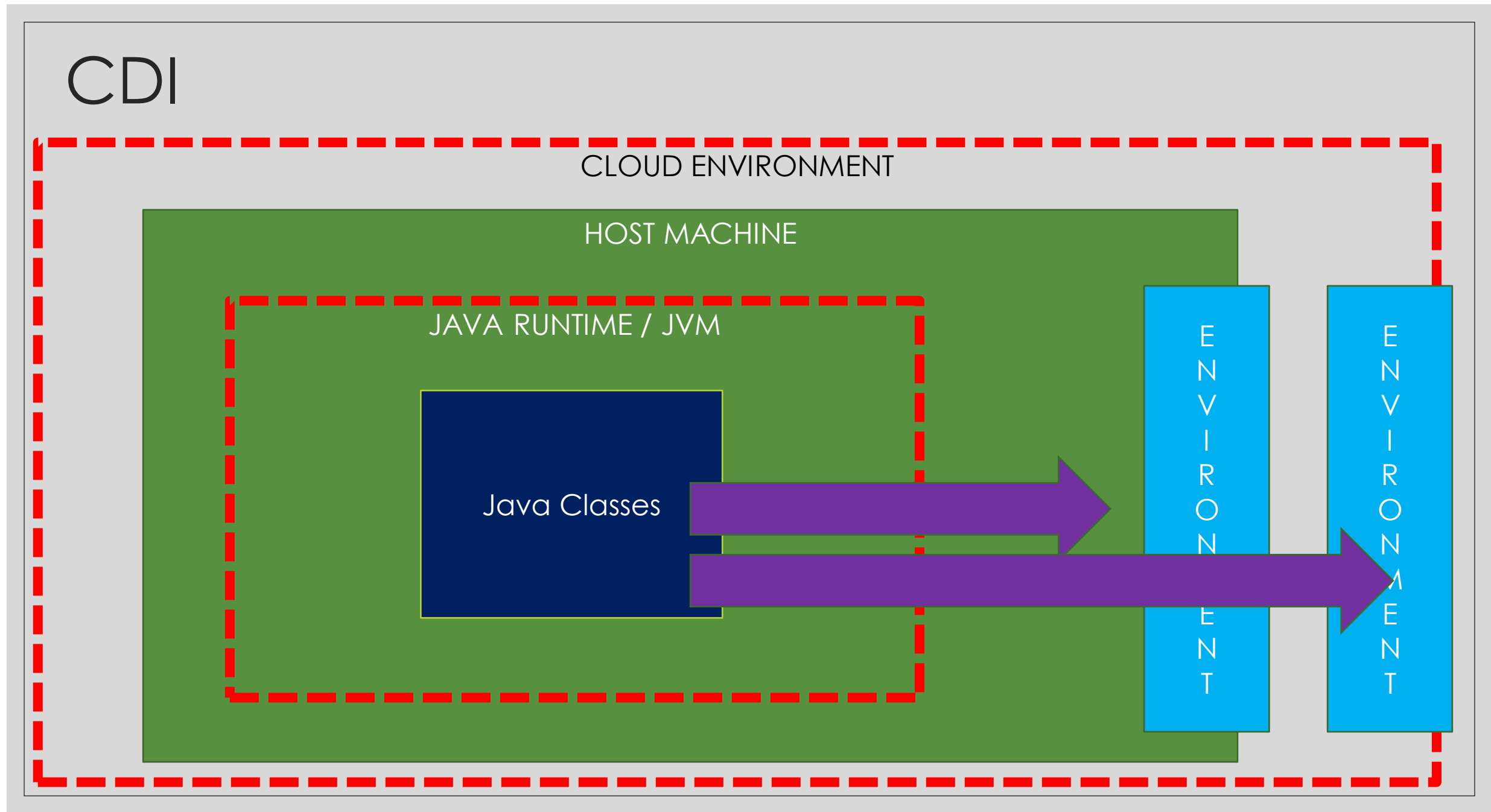
JAVA RUNTIME / JVM

Java Classes

E
N
V
I
R
O
N

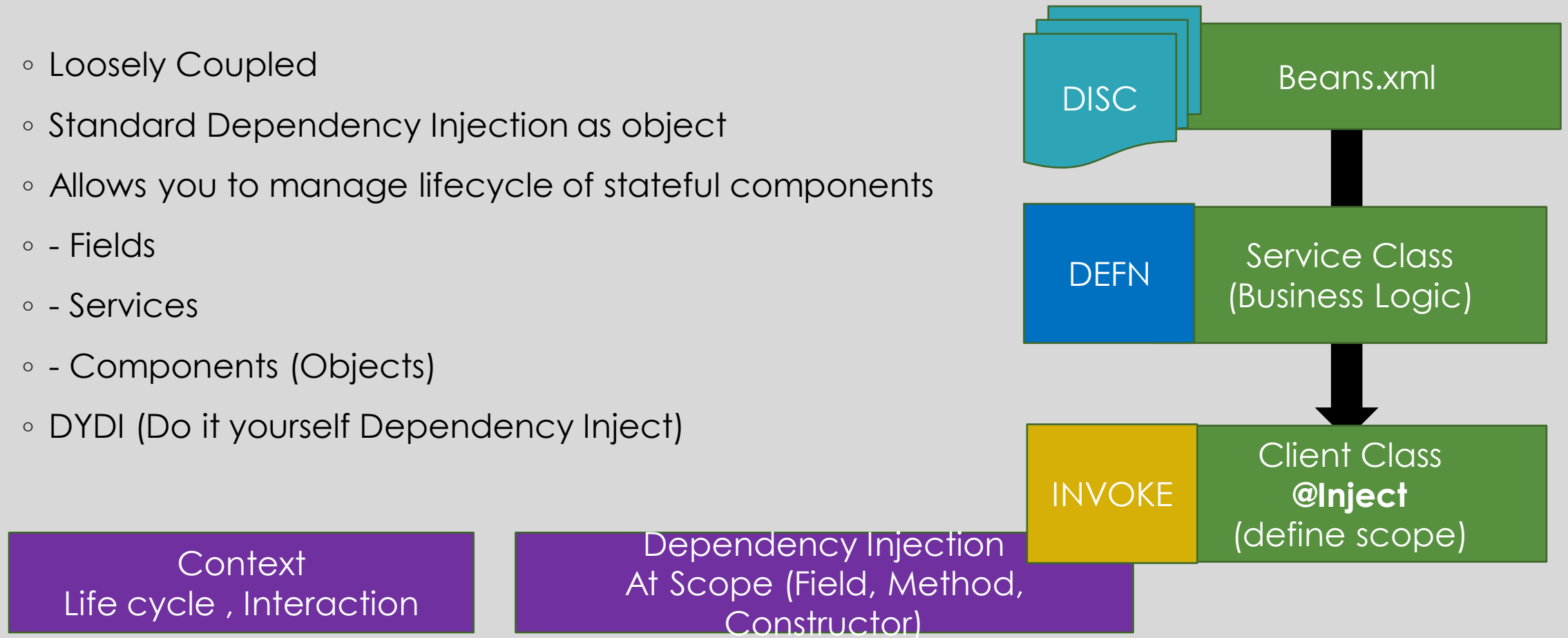
E
N
T

E
N
V
I
R
O
N
M
E
N
T

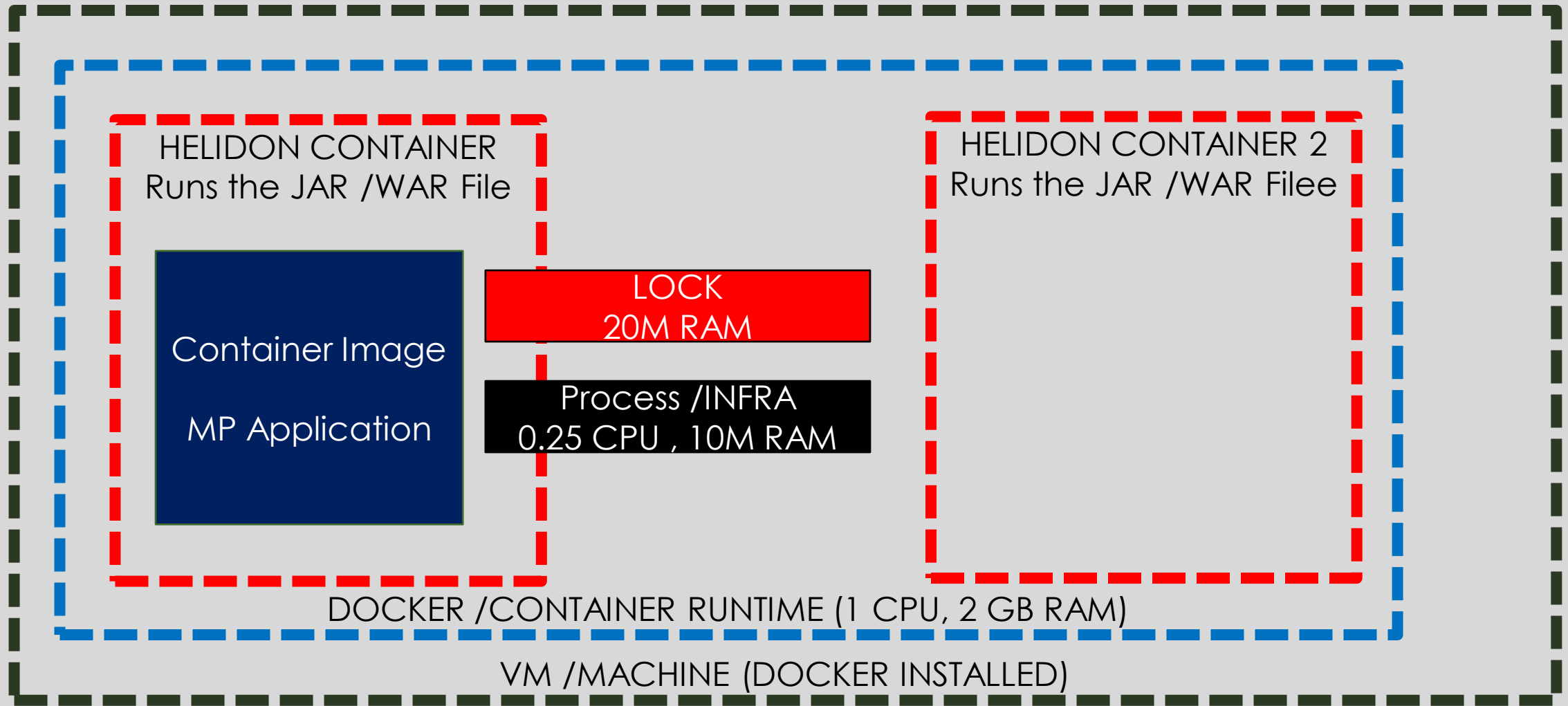


Context and Dependency Injection

- Loosely Coupled
- Standard Dependency Injection as object
- Allows you to manage lifecycle of stateful components
 - - Fields
 - - Services
 - - Components (Objects)
- DYDI (Do it yourself Dependency Inject)



Deploying Helidon Containers



Maven Lifecycle

Create or Generate
Sources from central
Repository

~/.m2

Download of the
Dependencies

#pom.xml

Resource Graph

Compile and Validate

mvn package

Build
Environment

Jar file

Container
Image
VM Application
(makefile)

Setting up Helidon Microprofile Application

Maven Project
Template

Microservices
Architecture Format

SE/**MP**

Review Project
Resources in Eclipse

Life cycle Configured

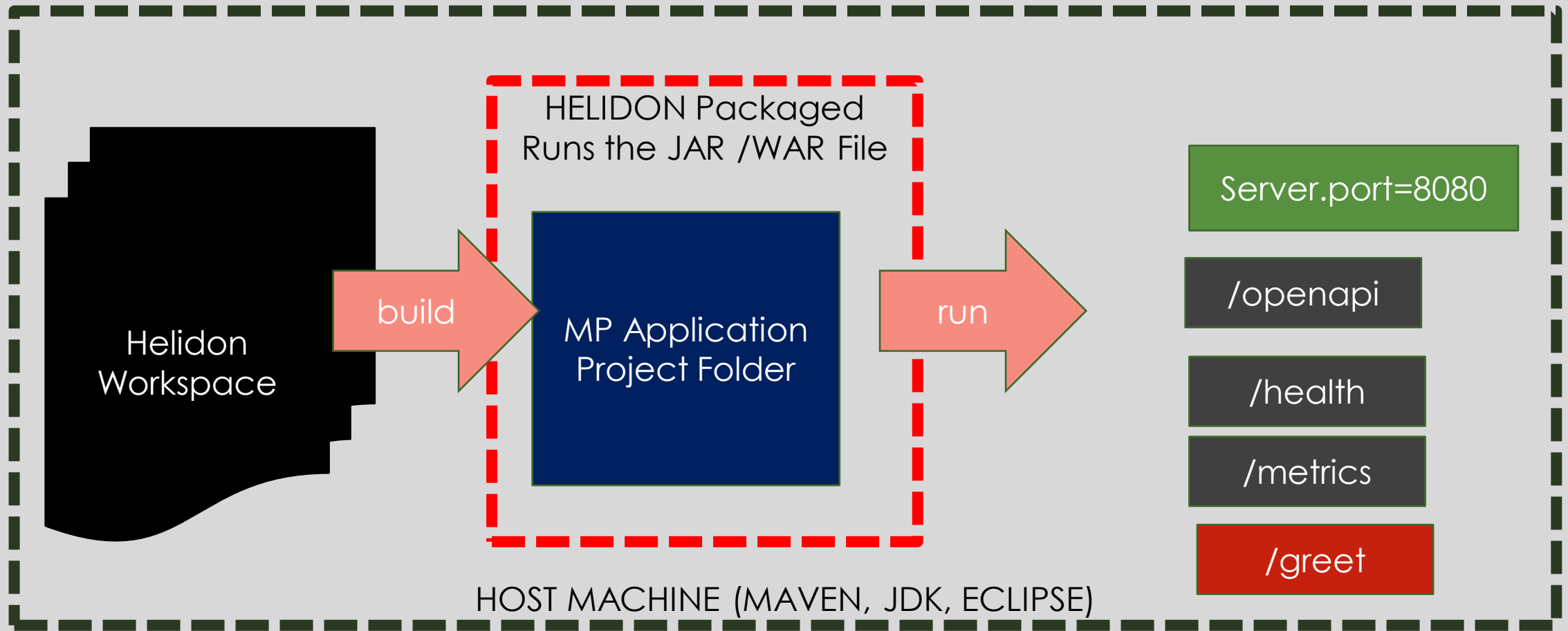
Compile the
package

#mvn package

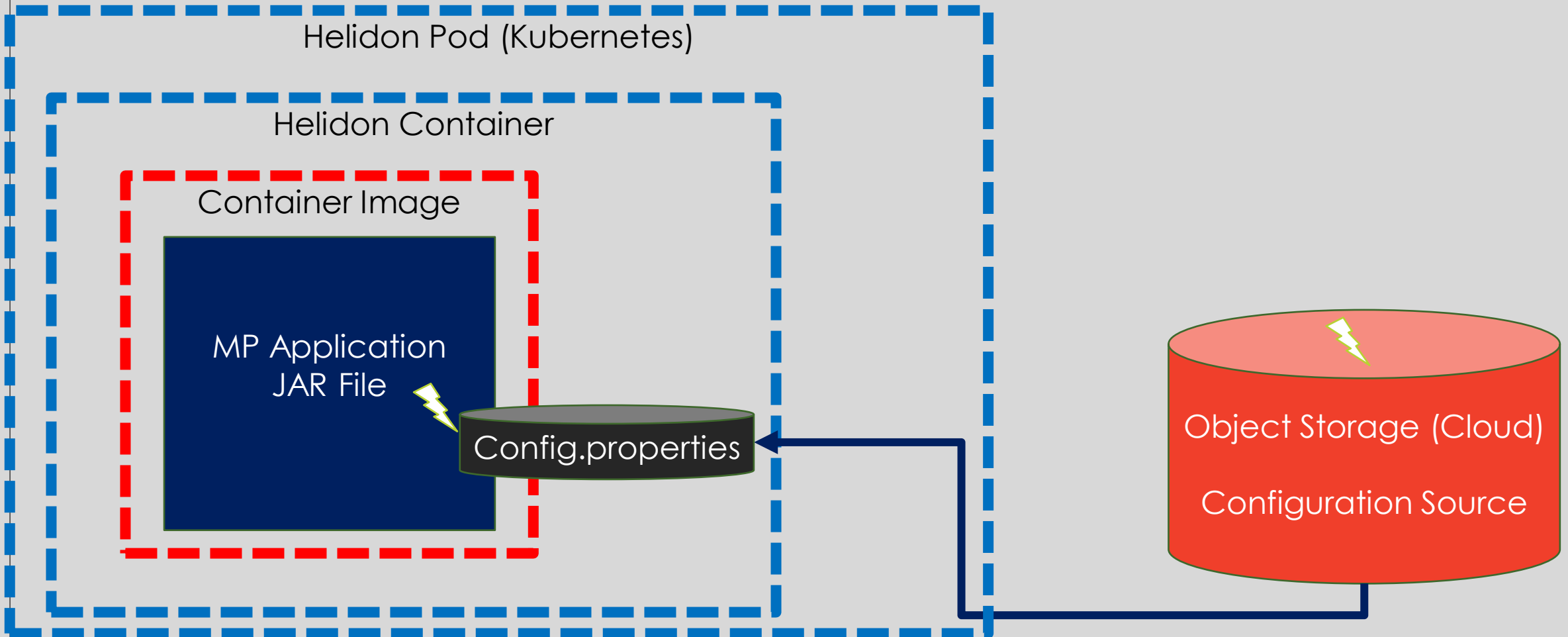
Review End
Points

/greet
/health
/metrics
/openapi

Launch Micro-profile Application (Slide 55)



Configure Properties



Configuration Scope

- **Config**
 - Comprehensive Configuration
 - INI Notation (/etc/hosts)
 - JSON Notation
 - YAML Notation
 - .Properties Notation
 - **Customize Precedence of Resources**
 - **Make the precedence – optional or mandatory**

Configuring the Main Server

1

Configuring Environment Variables

2

Order of Precedence

Xyz.properties

1

Microprofile-
config.properties

//Local to project

2

Environment Variables

java -D APP_GREETING=

Configure Server

- `io.helidon.micprofile.server.Server`
- Server Builder → Static Server Instance
- Start builder → `builder().start ()` (CRUD on Server)
- Configure Server → `Server.builder().config (Config()). build().start()`
- `Config()` → Config
 - `Disableenvironmentsource()` /stop default configuration path
 - `sources (1) //ClassPath`
 - `source (2)`

Use case : Start and Configure Server

Helidon MP
Template

Create Custom
Build Main ()
//Start Server
Instance

Without Config
Scope

Custom Config
Resource Scope

Custom Start Server
Instance with
Configuration

T
E
S
T

Use case 2 – Extending

Helidon MP
Template

Custom Config
Resource Scope

Custom Start Server
Instance with
Configuration
And Precedence

T
E
S
T