ORACLE

Mikroszolgáltatások és adatkezelés

Sárecz Lajos - felhő architect, Oracle
Fekete Zoltán - technológiai tanácsadó (cloud és on-prem), Oracle
Petrohán Zsolt - technológiai tanácsadó (cloud és on-prem), 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, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Napirend

- Mikroszolgáltatások és Monilitikus architektúra összehasonlítása
- Mikroszolgáltatás architektúra futtató környezet felhőben - Kubernetes
- Adatkezelés Mikroszolgáltatás architektúrában – Oracle Adatbáziskezelő

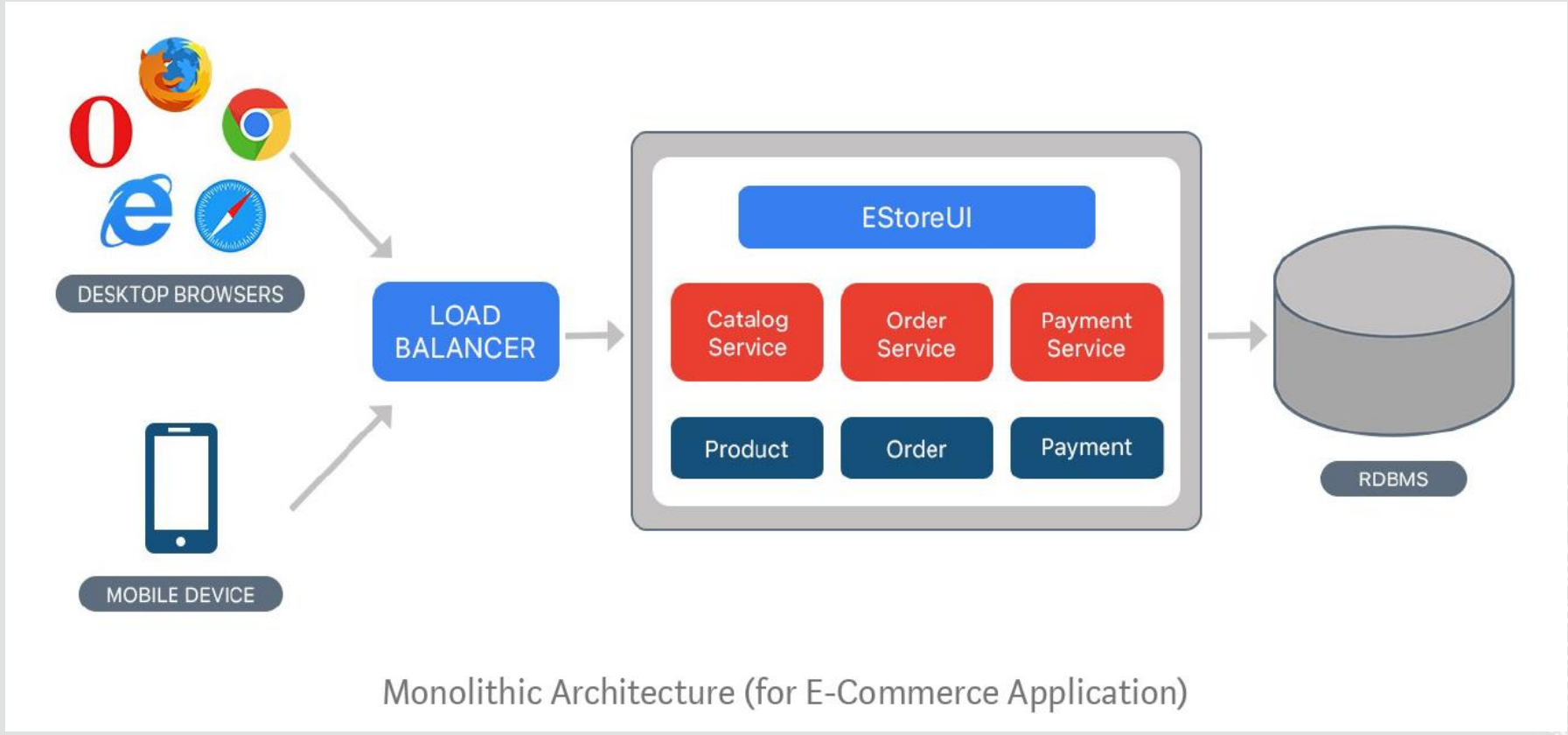
Az architektúra tervezés alapkövetelményei

Architektúra tervezés

Egy jól kitalált architektúra a megvalósítást olcsóbbá teszi, meggyorsítja a bevezetést és megbízhatóvá, karbantarthatóvá teszi a szoftvert.

Monolitikus architektúra

A monolitikus azt jelenti, hogy egy darabból áll. Az alkalmazások világos, logikailag moduláris felépítésűek, az alkalmazás mégis monolitként van összeállítva, telepítve és működtetve.



Monolithic Architecture (for E-Commerce Application)

Monolitikus architektúra

Előnyei

- Egyszerű fejlesztés
- Egyszerű tesztelés
- Egyszerű telepítés
- Egyszerű horizontális skálázhatóság

Hátrányai

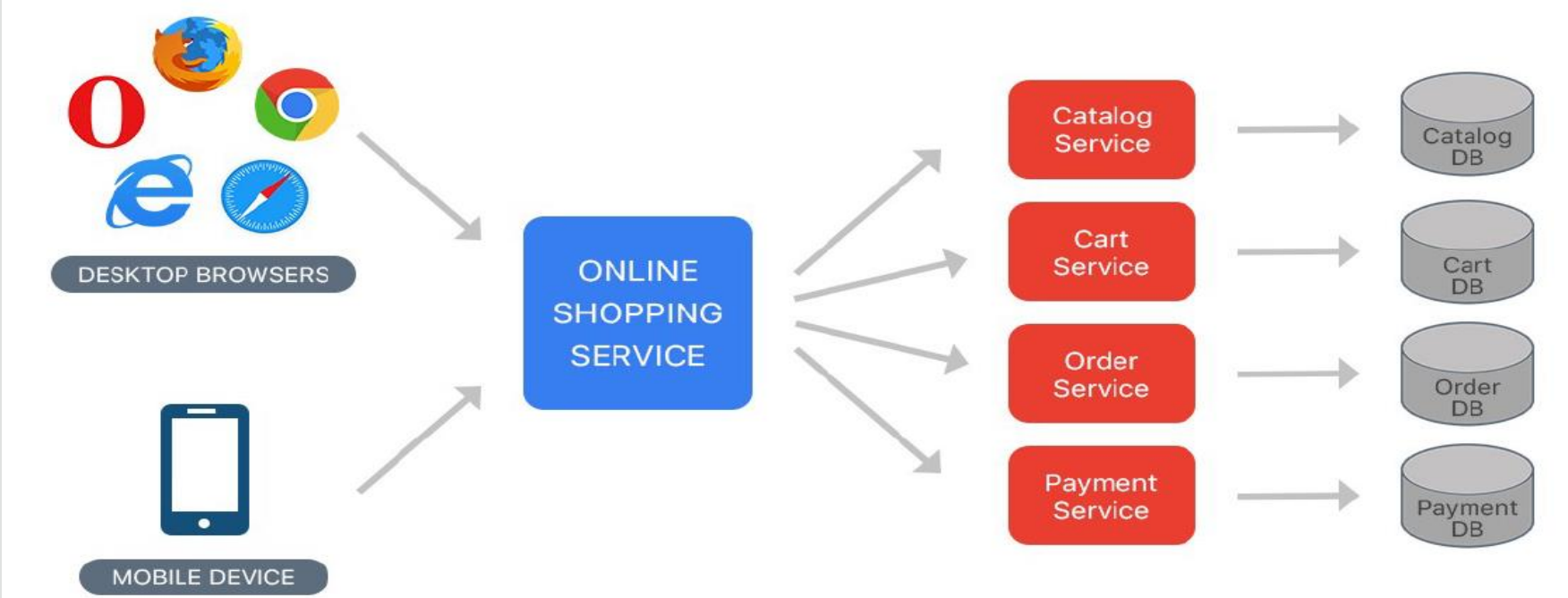
- Karbantartás
- Alkalmazás méret
- Frissítés
- Skálázhatóság
- Megbízhatóság
- Fejlesztési sebesség
- Nehéz új és fejlett technológiákat alkalmazni

Miért legyen mégis Monolitikus

- Érdemes átgondolni
  - Meglévő tudás felhasználása sokkal egyszerűbb mint egy komplex mikroszolgáltatás architektúra tervezése, kialakítása
  - A Monolitikus architektúra nem anti-pattern
- Ha lassú a fejlesztés, próbáljuk ki a következőket
  - Fejlesztési folyamat optimalizálás
  - Tesztelés automatizálás
  - Telepítés automatizálás
  - Növeljük a csapatok autonómiáját
  - Modulok kiterjesztése
  - Több funkcionális csoport létrehozása
  - Elavult technológia helyett modern technológiára épülő monolitikus alkalmazás
- Monolitikus alkalmazás + SOA (meglévő alkalmazások esetén)

Mikroszerviz architektúra

Az alkalmazásainkat **komponensekből építjük fel.** A komponensek mind teljesen különálló folyamatok, melyek egymással szabványos kérésekkel kommunikálnak. Ennek a megközelítésnek a legnagyobb előnye, hogy így a különböző komponensek **egymástól függetlenül publikálhatóak, frissíthetőek.**



Microservices Architecture (for E-Commerce Application)

”Mikro” a Mikroszolgáltatásban != kisebb szolgáltatás

A Mikro lényege: kevésbé összetett

Monolitok

Modul 1 Modul 2 Modul N

Üzleti logika

Adatbázis

Támogatnia kell az összes modul követelményeit

Teljes futtató környezet ami támogatja az összes felhasználói esetet

Mikroszolgáltatások

API

Egy modul követelményeit kell támogatnia

Logika

Adatbázis

Könnyű futtató környezet, amelyek egy dolgot tesznek

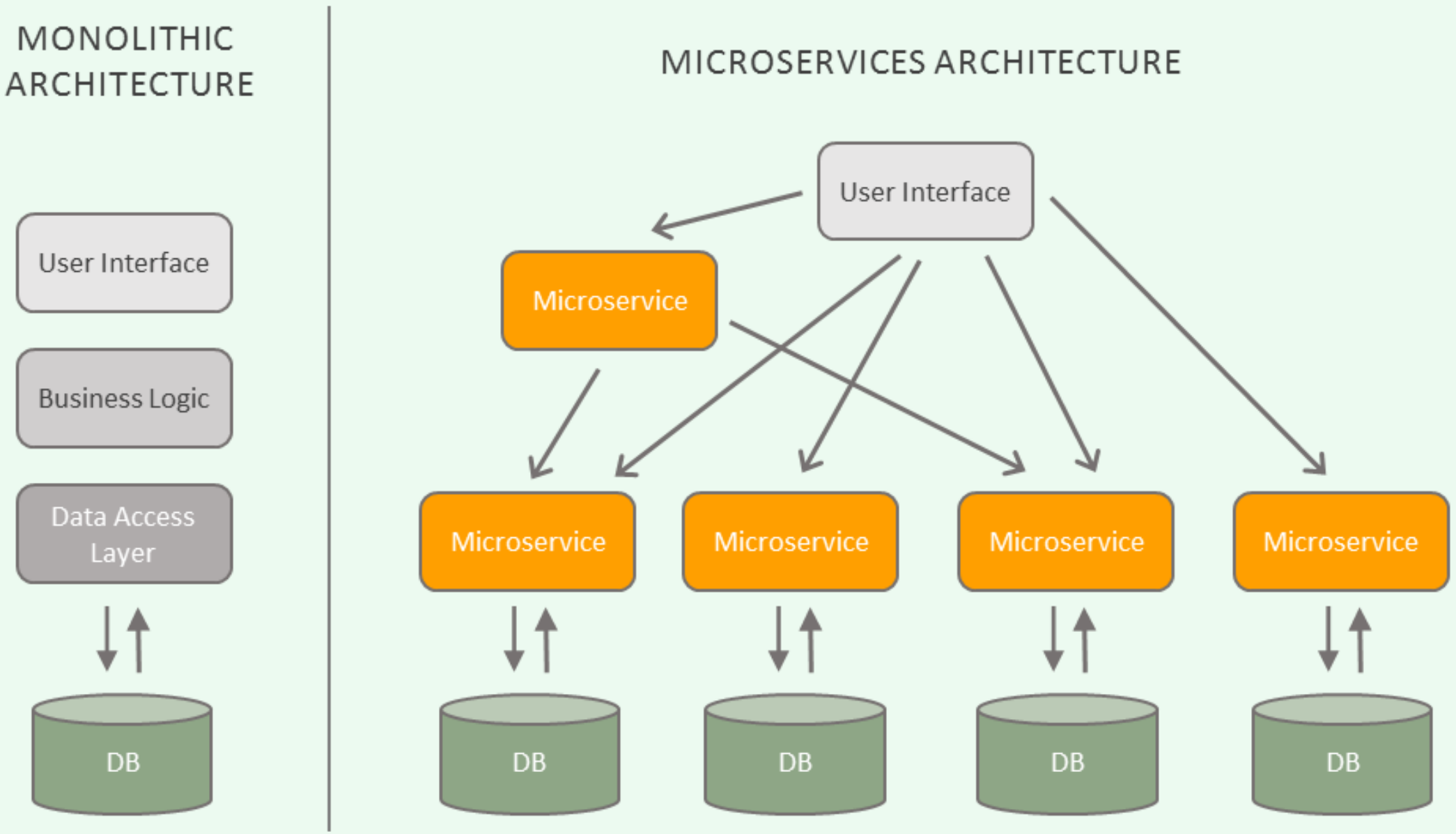
Mikroszerviz architektúra

Előnyei

- Rugalmas fejlesztés
- Megbízhatóság
- Fejlesztési sebesség
- Komplex alkalmazások építése
- Népszerű

Hátrányai

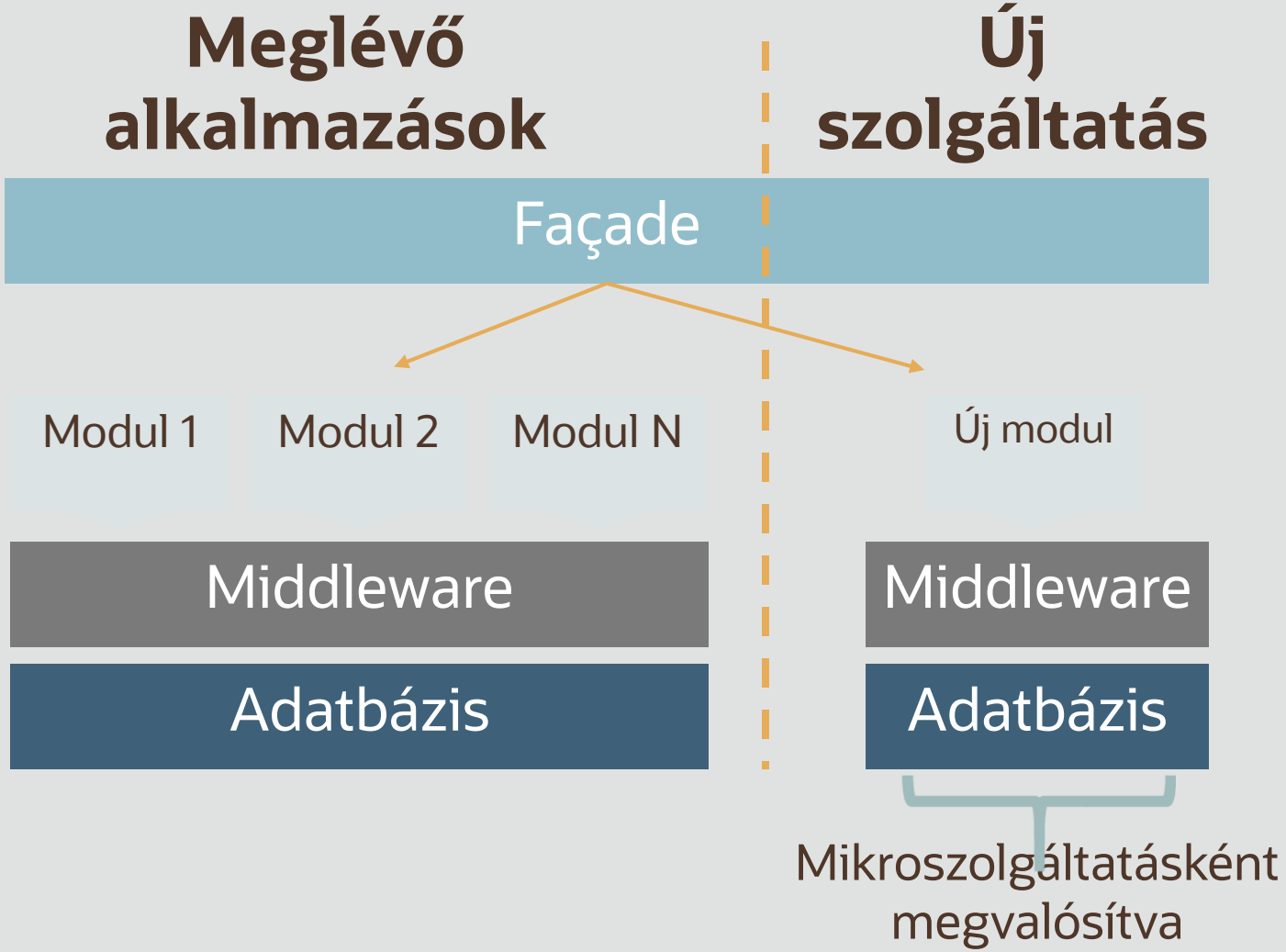
- Extra komplexitás
- Konzisztencia
- Képzett fejlesztő csapatot igényel
- Rossz tervezés esetén bonyolult architektúra, ami semmi előnyt nem nyújt.


MONOLITHIC ARCHITECTURE
User Interface
Business Logic
Data Access Layer
DB
MICROSERVICES ARCHITECTURE
User Interface
Microservice
Microservice
Microservice
Microservice
Microservice
DB
DB
DB
DB

ORACLE

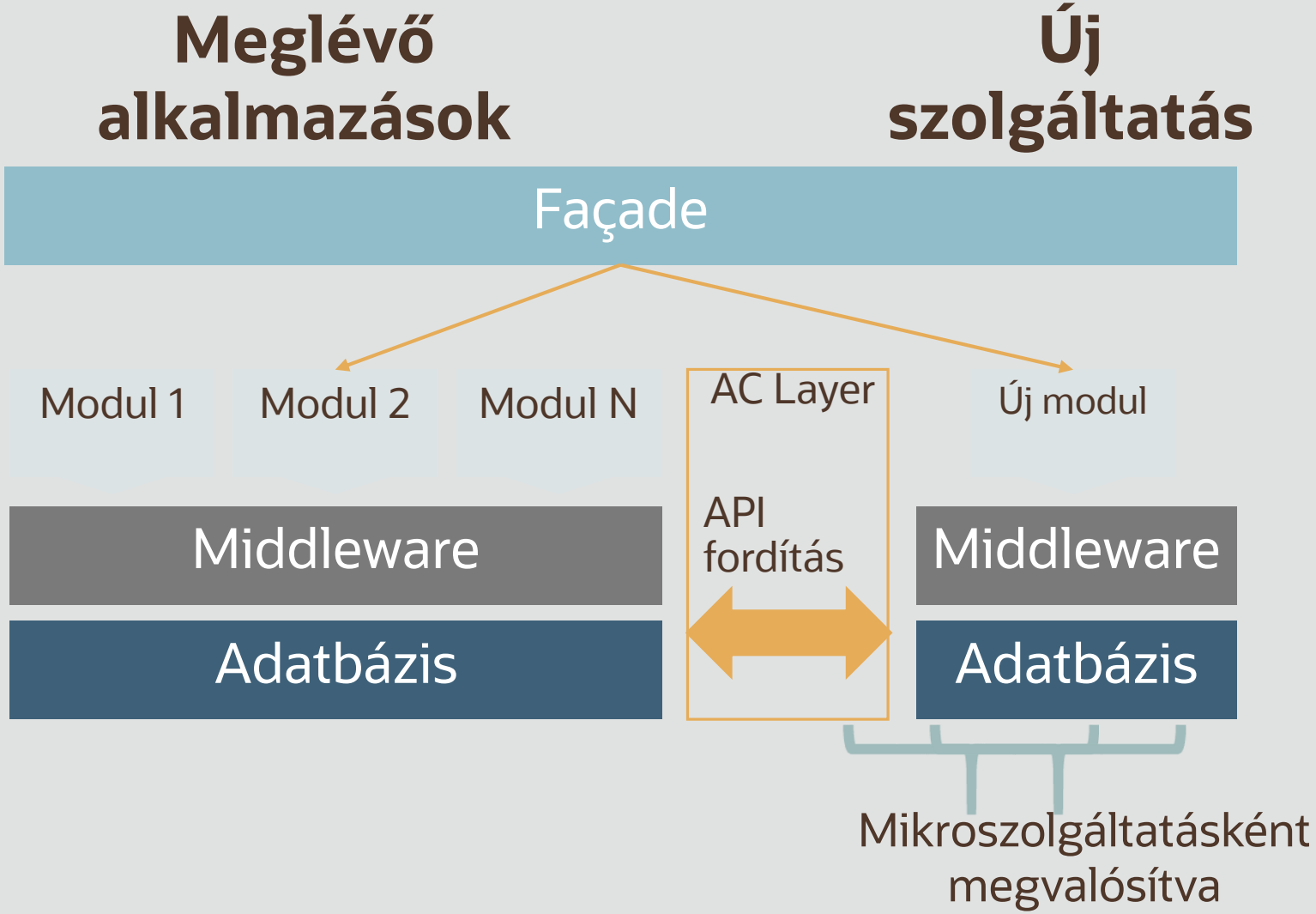
Együttműködési minták

Monolit a mikroszolgáltatásokhoz a Strangler (leépítő) mintázat segítségével



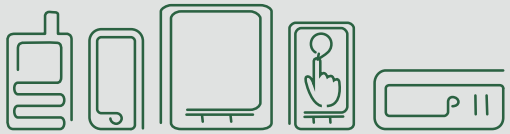
- Az új szolgáltatás vagy meglévő komponensek mikroszolgáltatásokként kerülnek megvalósításra
- A Facade a felhasználói igényeket a megfelelő alkalmazáshoz irányítja
- Az idő múlásával egyre több szolgáltatás költözött az új architektúrába

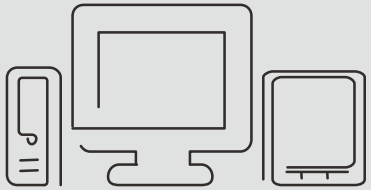
Anti-Corruption Layer (sérülésgátló réteg) minta

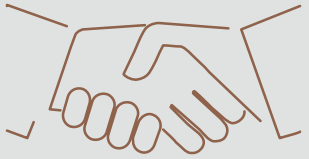


- Akkor használjuk, amikor az új szolgáltatásoknak hozzá kell férniük a régi alkalmazáshoz
- Lefordítja a fogalmakat a meglévő alkalmazásból az újra és fordítva

SOA 2.0 referencia architektúra







Üzleti API-k

Intergációs logika / SOA réteg

CX

HCM

ERP

EPM

Legacy, etc

SaaS

Nyilvántartási rendszere

Microservices

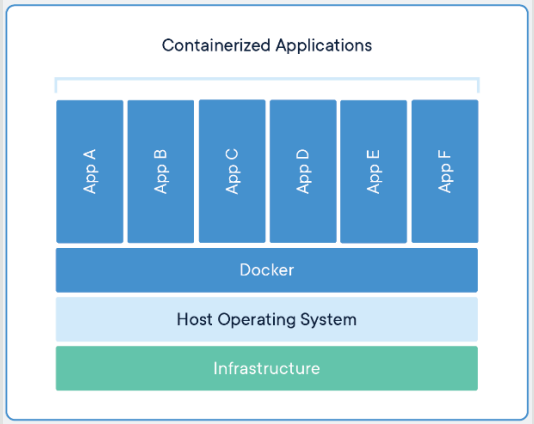
Innovációs rendszerek

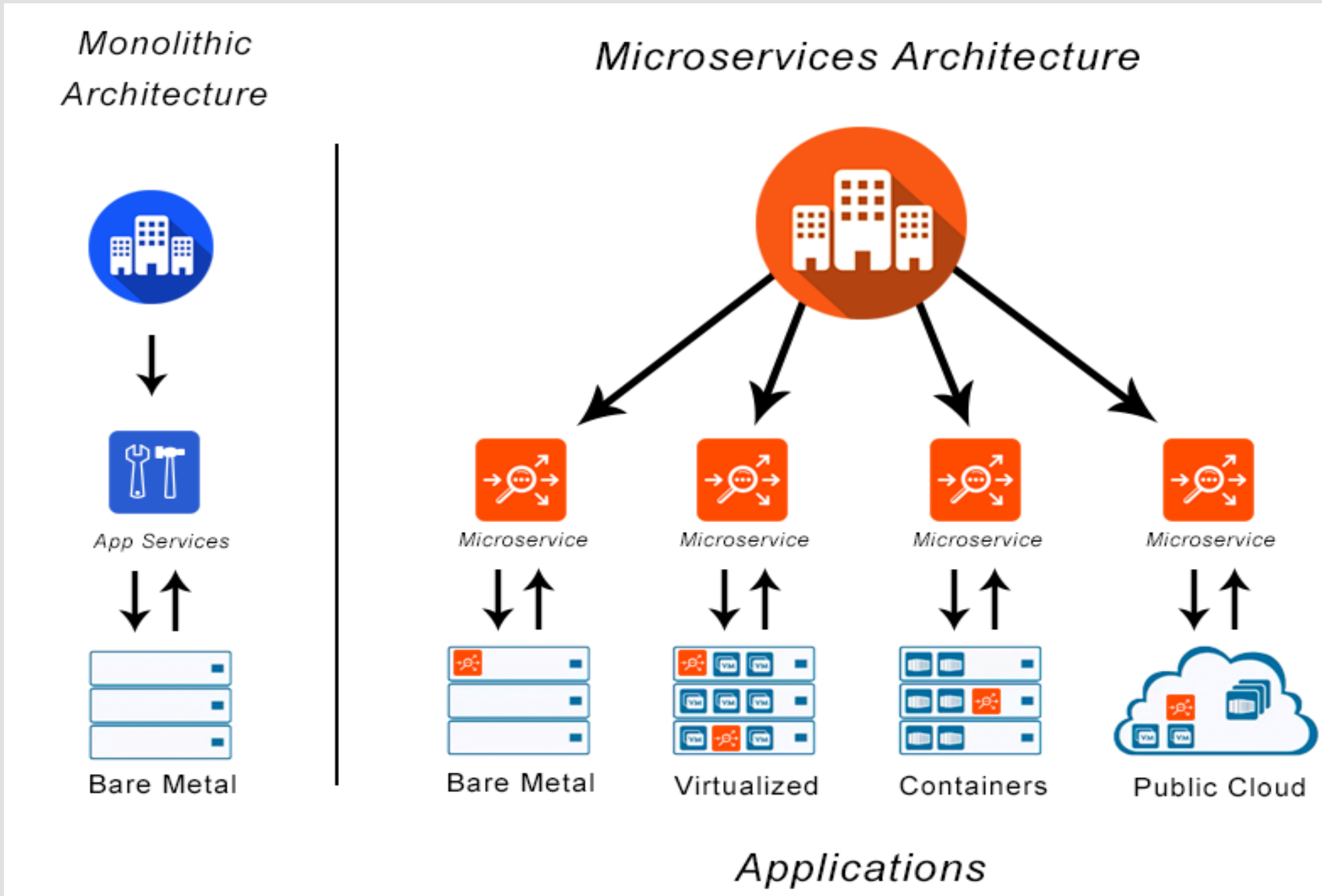
Rendszerek megkülönböztetése

Üzenetküldő réteg

ORACLE

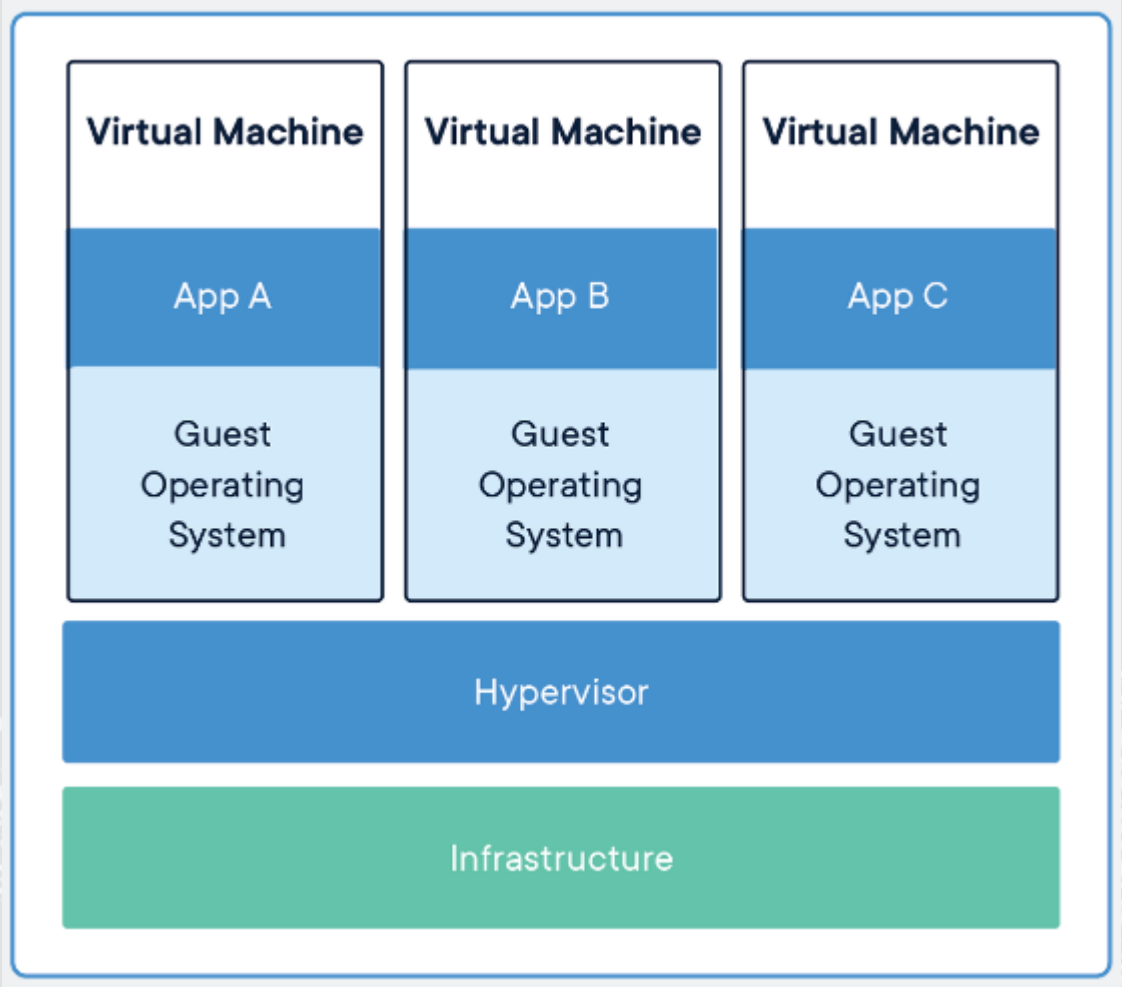
Konténerek




Monolithic Architecture
Microservices Architecture
App Services
Bare Metal
Microservice
Microservice
Microservice
Microservice
Bare Metal
Virtualized
Containers
Public Cloud
Applications

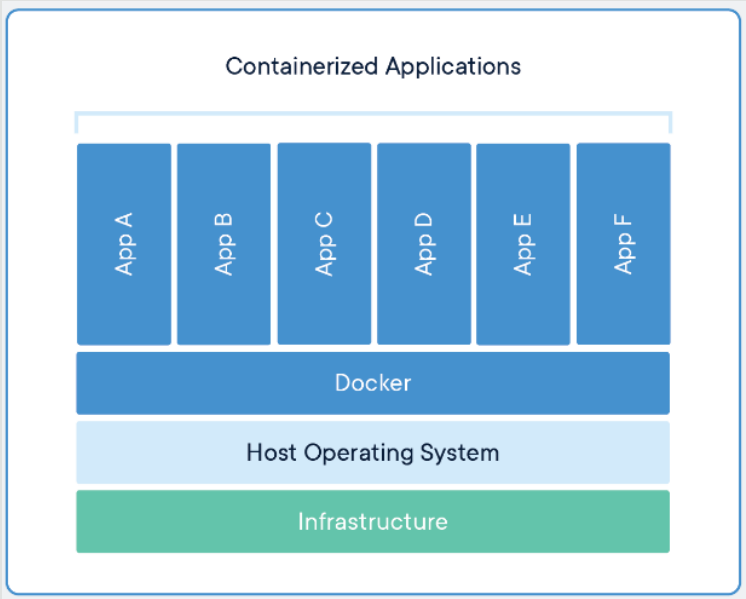
Virtuális gépek

- Valódi számítógép emulációja, szinte pontosan úgy hajt végre bármilyen feladatot, mint egy valódi, fizikailag létező számítógép
- A VM a gazdaszámítógép erőforrásain (RAM, CPU, stb.) osztozkodik
- *Hypervisor* réteg biztosít átjárást a fizikai gép operációs rendszere és a VM-ek között
- Minden egyes VM **tartalmazza a teljes szoftver-, és hardver stacket**



Konténerek

- Operációs rendszer szintű virtualizációt biztosít
- A konténerek nem az egész stacket, csupán a felhasználói teret virtualizálják, a többi erőforrás pedig közös
- Nincs virtualizált hardver és OS
- A konténerek csupán az alkalmazásunkhoz kötődő könyvtárak, binárisok és a felhasználó terület tartalmazzák
- A konténer nagyságrendekkel kisebb overheaddel bír a VM-ekhez képest



Hol jön a képbe a Docker?

- Az egyik legelterjedtebb konténer keretrendszer
- Erősen támaszkodik a Linux kernel nyújtotta szeparációs lehetőségekre
- Könnyű használhatóság
- Hordozható – onprem, cloud
- Sebesség – kicsi konténerek, gyors indulás
- Modularitás és a skálázhatóság

Docker hiányosságai — miért van szükségünk konténer orchestration megoldásra?

- Production környezet kihívásaira a konténer önmagában nem ad választ
- Hibatűrő környezet
- Igény szerint skálázhatóság
- Rendelkezésre állás
- Konténerek nem képesek egymást felderíteni
- Konténerek közötti kommunikáció

A konténer orchestration ezekre a dolgokra kínál megoldást egybe csomagolva.

kubernetes

Kubernetes
Production kész konténer orchestration

A Kubernetes egy open-source megoldás konténereink automata telepítéséhez, skálázáshoz és menedzseléshez.

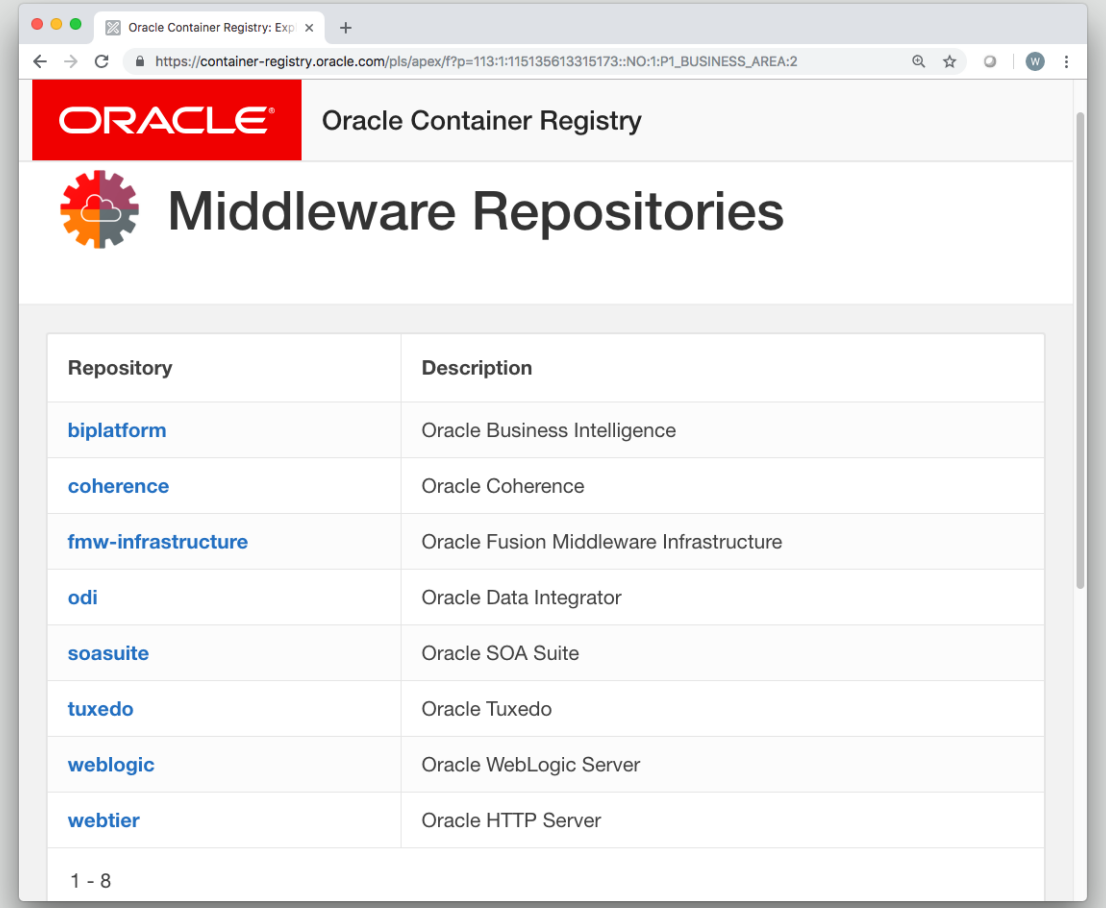
- A Google által indított projekt, 15+ év tapasztalat csak Google –nél
- Közösségi támogatás, jelenleg a Cloud Native Computing Foundation (CNCF) alá tartozik
- A konténereket futtató hosztok klaszterként kezelhetők
- Konténereink központilag, tetszőleges hosztokon tudjuk szétszórni
- Különböző hosztokon futó konténerek összekapcsolhatók, egyben tarthatók
- Állandó erőforrás felügyelet és optimalizálást biztosít a konténerek skálázásával és áthelyezésével
- Alkalmazásainkat automatikusan, leállás nélkül terjeszthetjük, frissíthetjük
- Végrehajtott frissítések és terjesztések visszavonhatók
- Magas rendelkezésre állás a klaszter által
- Kubernetes segítségével konténereink skálázása és terheléselosztása platformfüggetlen maradhat

ORACLE

Oracle megoldások a témában

Fusion Middleware Products on Kubernetes

- Support key FMW products in production
  - Leverage WebLogic Kubernetes tools
- Oracle ADF (FMW Infrastructure)
  - Supported today
- SOA
  - Developer release supported today
  - Production support in H1CY2020
- IDM
  - OIG Production support in H1CY2020
  - Access, Directory support in H1CY2020
- WebCenter Sites, Portal, Content
  - Support planned for H12020



Project Helidon

- Java-könyvtárak halmaza a mikroszolgáltatások fejlesztéséhez
- Mikroszolgáltatási keretrendszer
  https://helidon.io
- Nyílt forráskód
  https://github.com/oracle/helidon
- Támogatja a szabványokat
  https://microprofile.io/
- Támogatott aktív projekt
  https://helidon.slack.com
  https://github.com/oracle/helidon/issues

Java alapú keretrendszerek

helidon SE

- Microframework
- Tiny Footprint
- Functional style
- Reactive
- Transparent

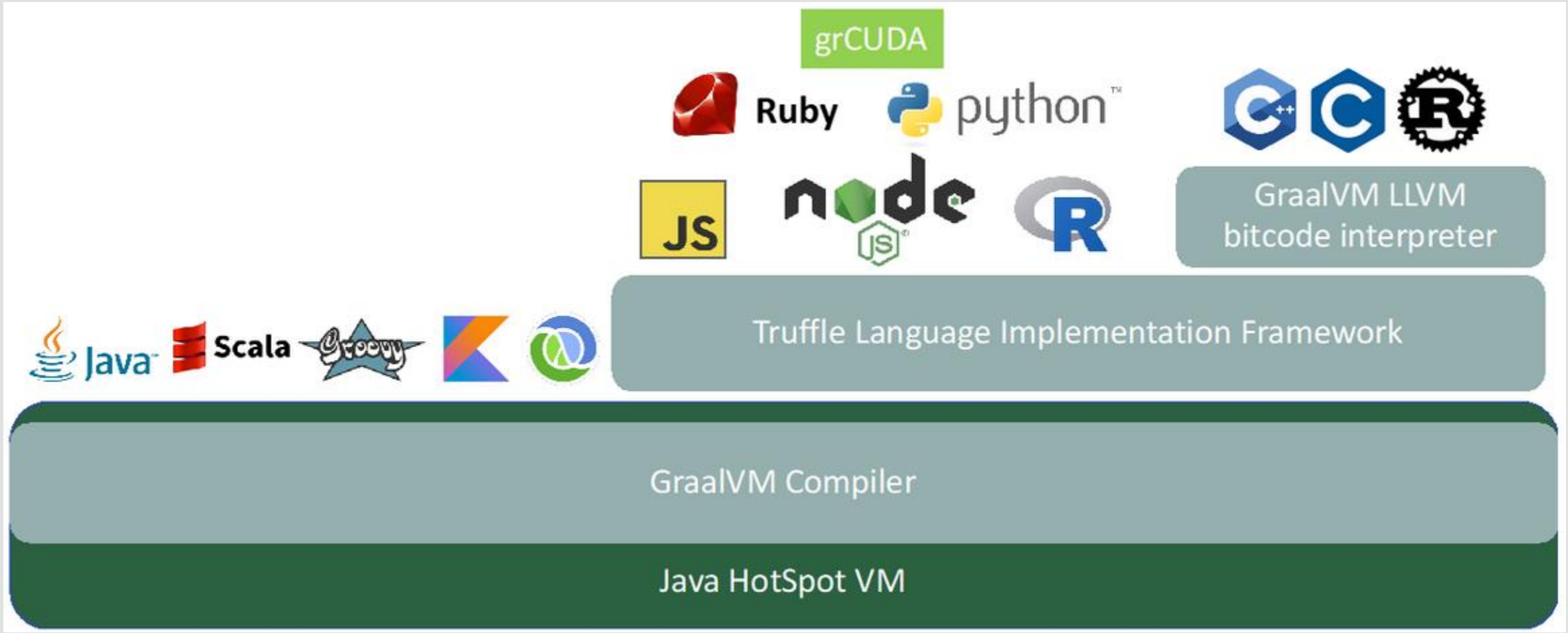
helidon MP

- MicroProfile 2.2
- Small Footprint
- Declarative style
- Dependency Injection
- CDI, JAX-RS, JSON-P/B

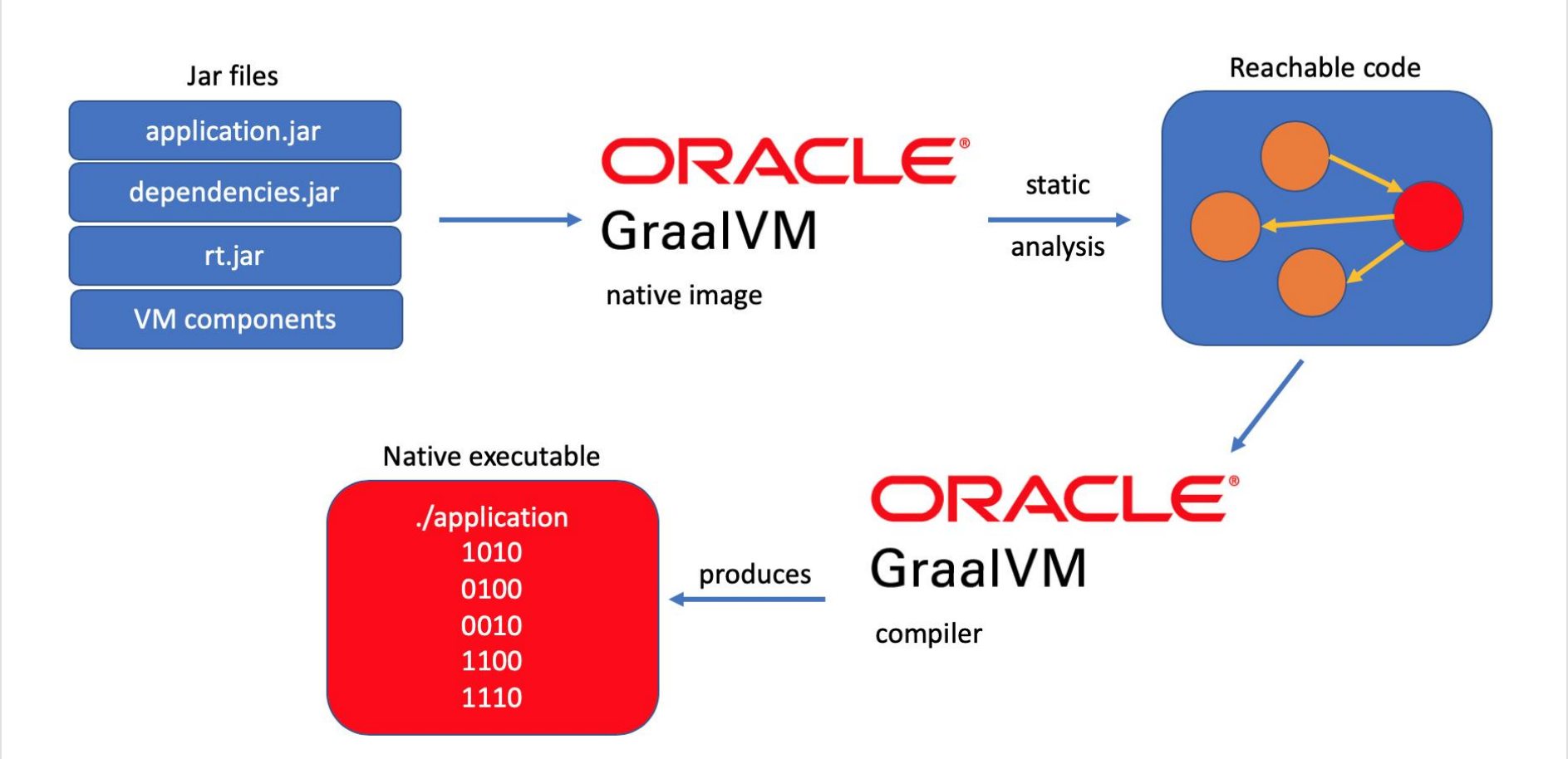
GraalVM

- Nagyteljesítményű polyglot virtuális gép
- A Java SE szabványára épül
- Alkotóelemei
  - GraalVM fordító (Just In Time (JIT), Ahead-in-Time (AOT, native image)
  - GraalVM Native images
  - Polyglot képességek (Javascript, Ruby, R, and Python)
  - LLVM Interpreter (C, C++)
  - Language Implementation Framework

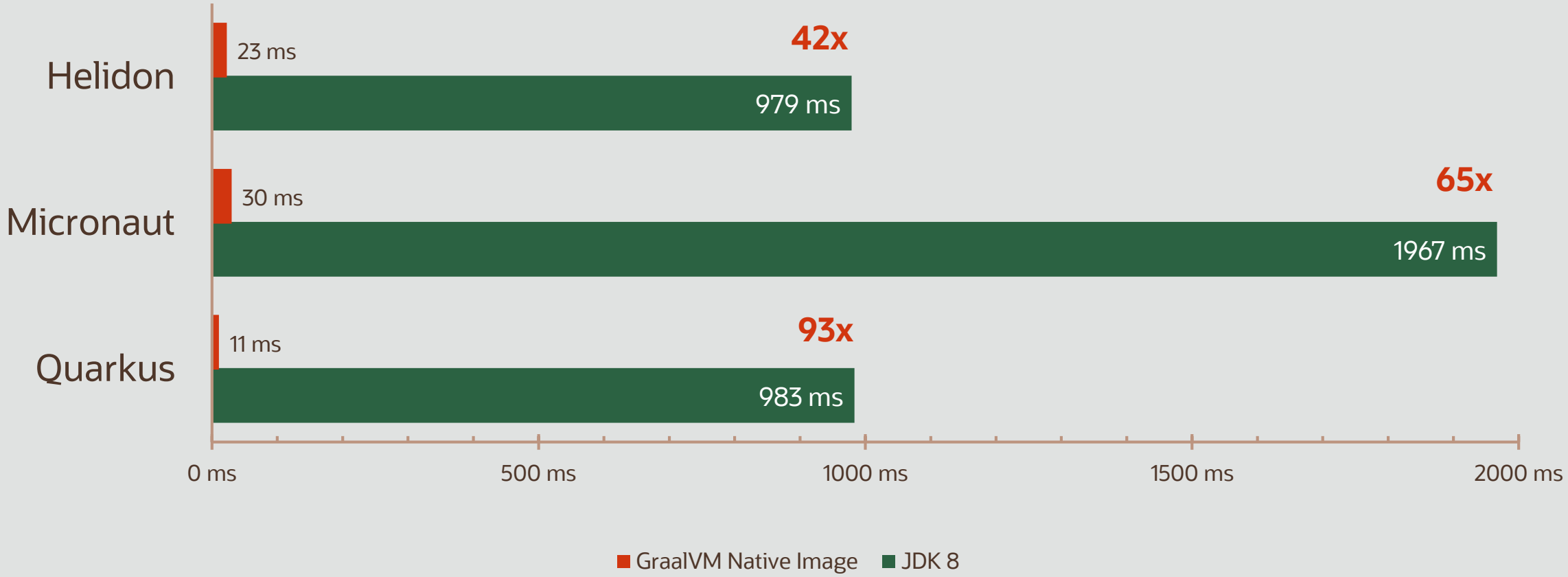
GraalVM architektúra



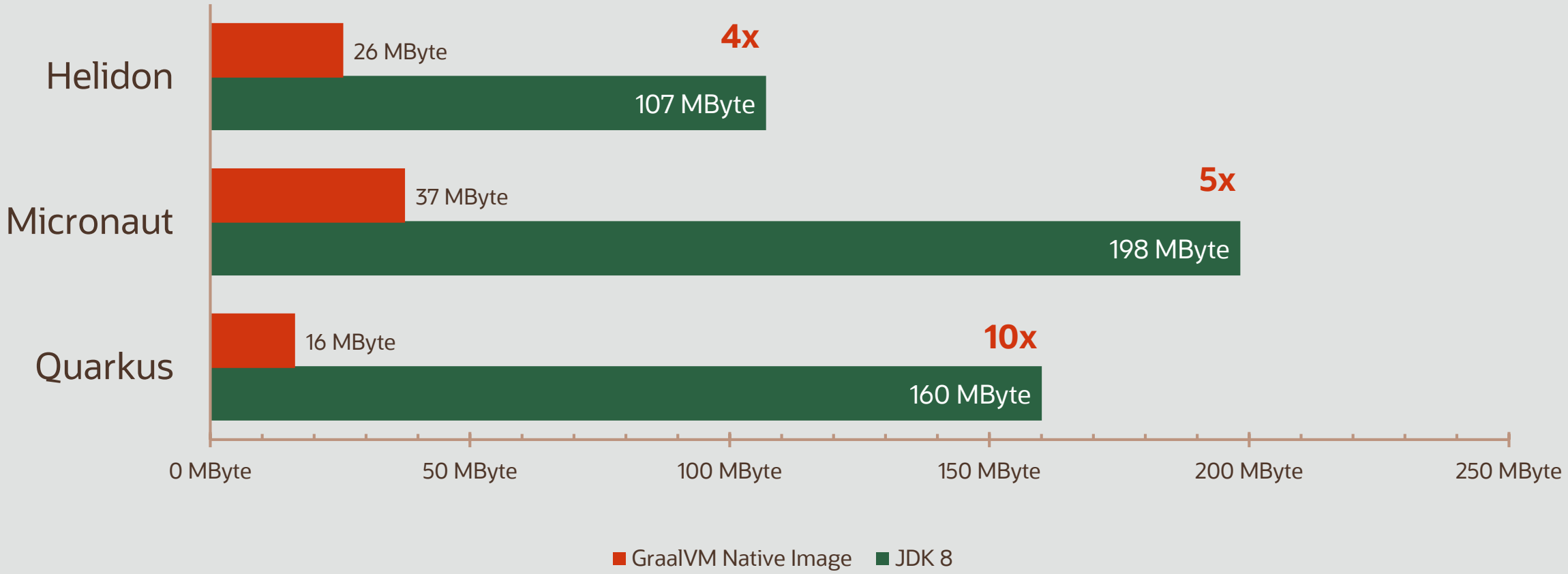
GraalVM - Native Image



Microservices – startup time



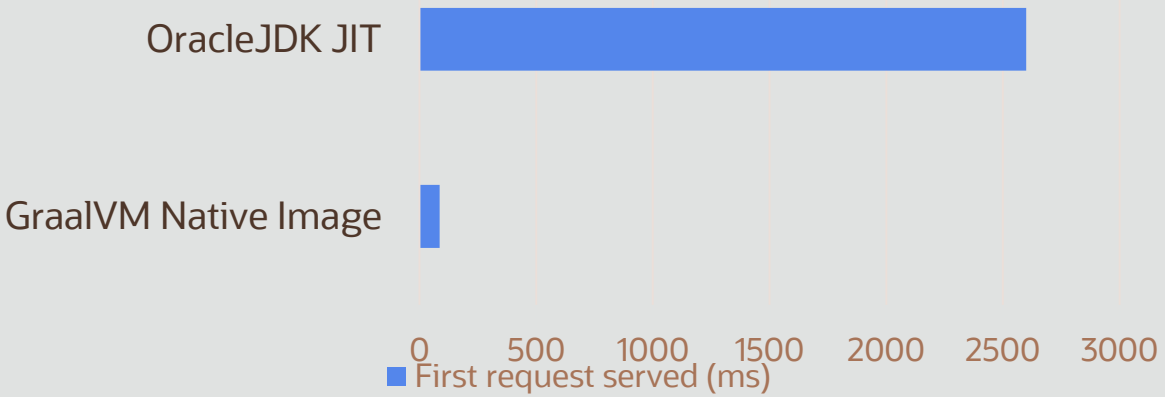
Microservices – memory footprint



SpringBoot apps running on GraalVM native image by Sébastien Deleuze

Allows to start applications almost instantly

Time to first request for Spring Boot 2.2 with Tomcat



https://youtu.be/3eoAxphAUIg

twitter

“We save a lot of money and CPU cycles”

10% performance increase
20% reduction in latency

Chris Thalinger
Staff System Engineer,
Twitter

Thank you

ORACLE