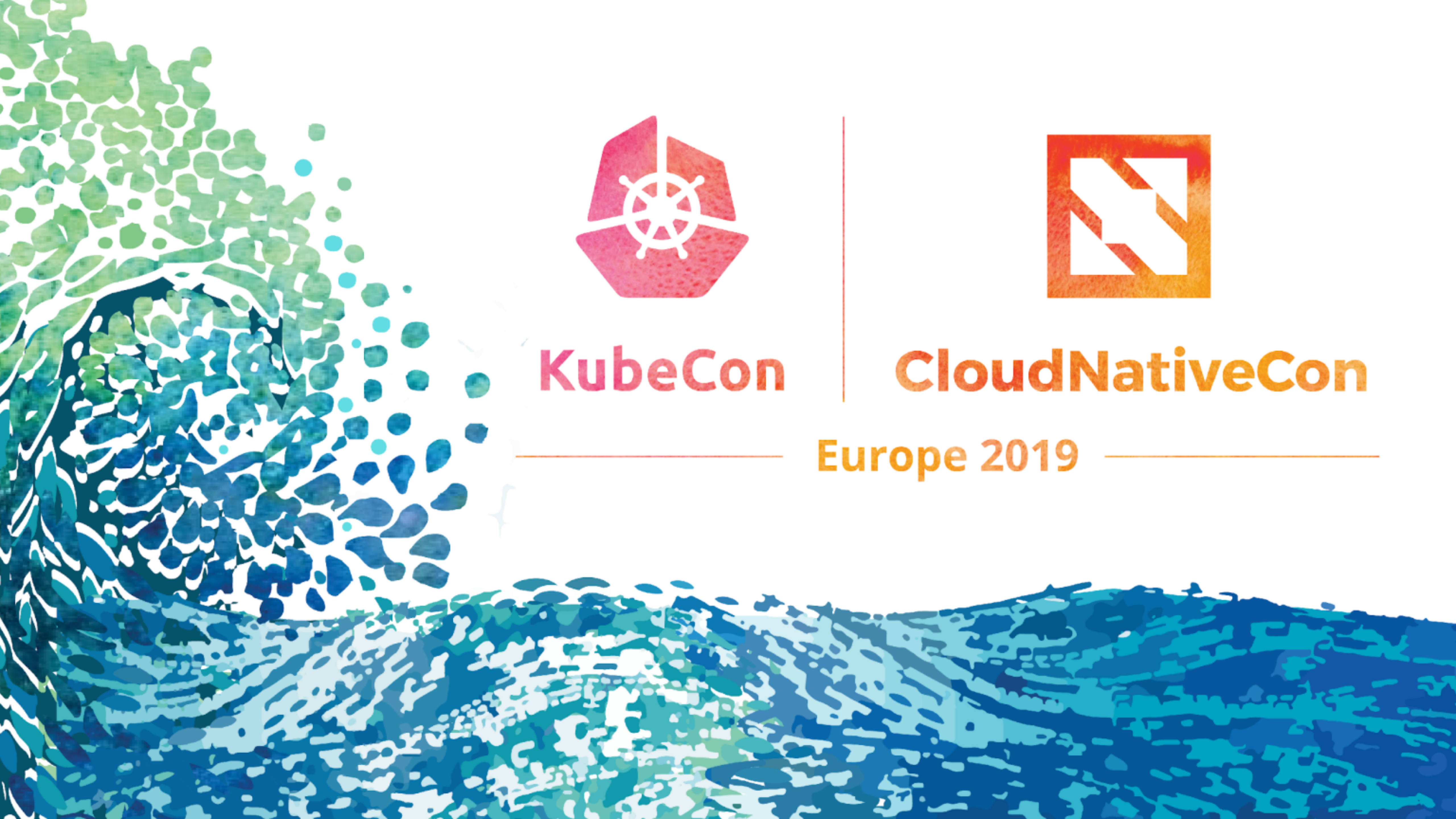


KubeCon | CloudNativeCon

Europe 2019

Microservices for the Masses

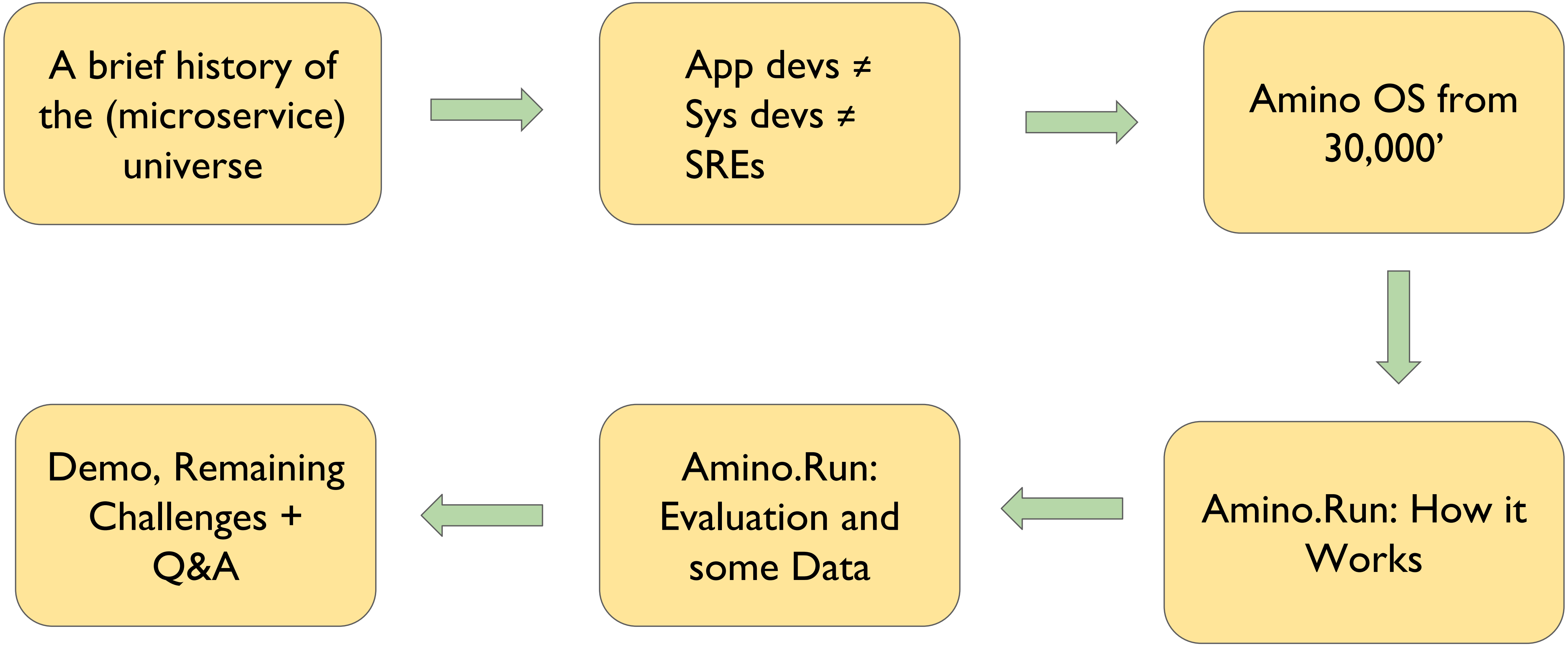
KubeCon | CloudNativeCon

Europe 2019

Integrating the Amino OS Distributed Cloud-native Programming Platform with Kubernetes (github.com/Amino-OS)

Quinton Hoole (Tech VP, Futurewei)
Irene Zhang (Univ. of Washington, & Microsoft Research)

Overview



A brief history of the (microservice) universe

App devs ≠ Sys devs ≠ SREs

Amino OS from 30,000’

Amino.Run: How it Works

Amino.Run: Evaluation and some Data

Demo, Remaining Challenges + Q&A

1.
A Brief History of the (Microservice) Universe

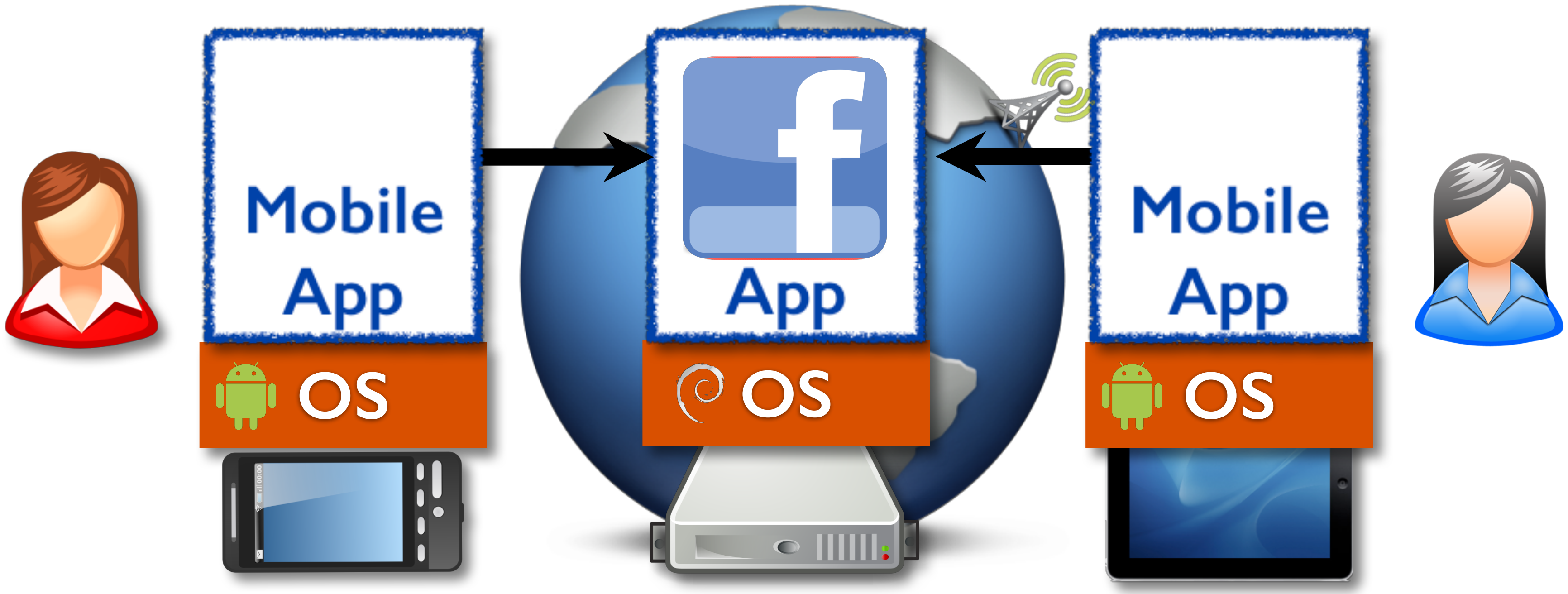
Once upon a time, applications were..

- single user
- single platform
- single node

Life was good for mere mortal app devs…

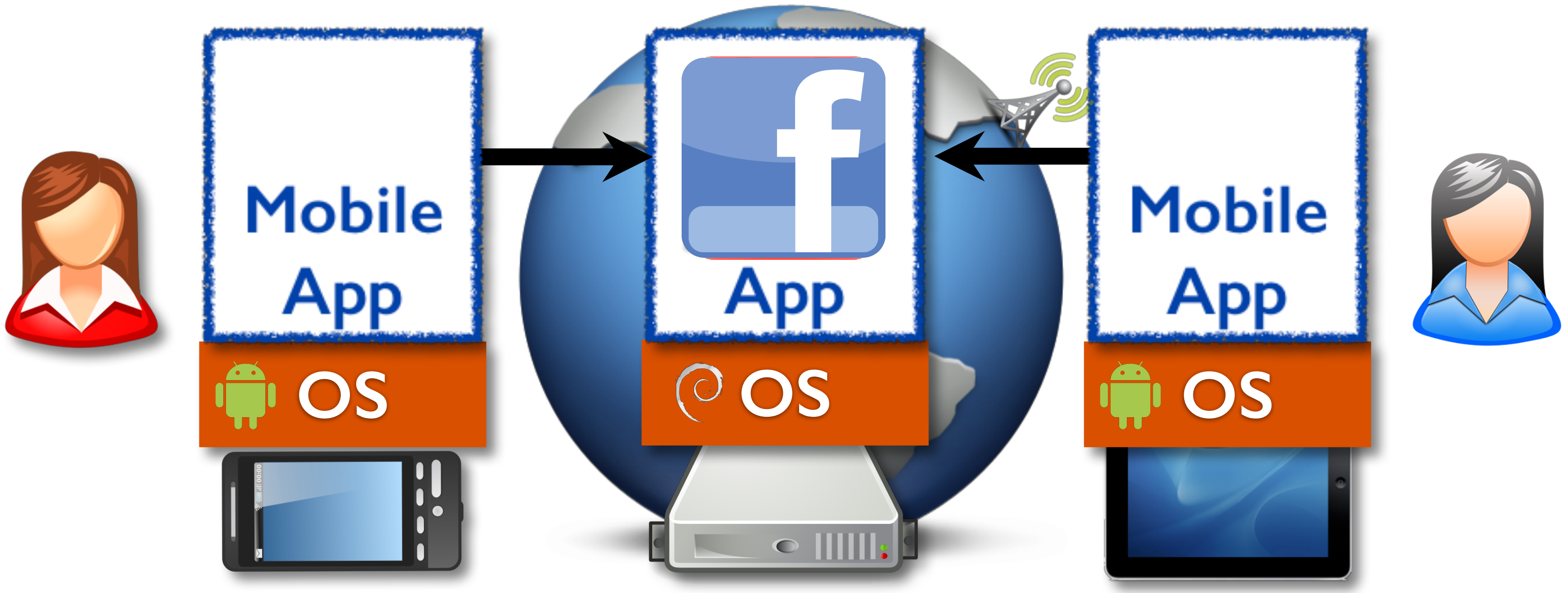
- Single-machine OS's work well
- Local procs, virtual memory, files, locks…
- Pick one (or two?) good programming languages
- App devs could understand their platform

Then "Suddenly" Everything Changed...



- Cloud Computing
- "Mobile-first"
- Ubiquitous Connectivity (Wifi... 3G… 4G… 5G…)

So Now Today's Applications are Very Different...



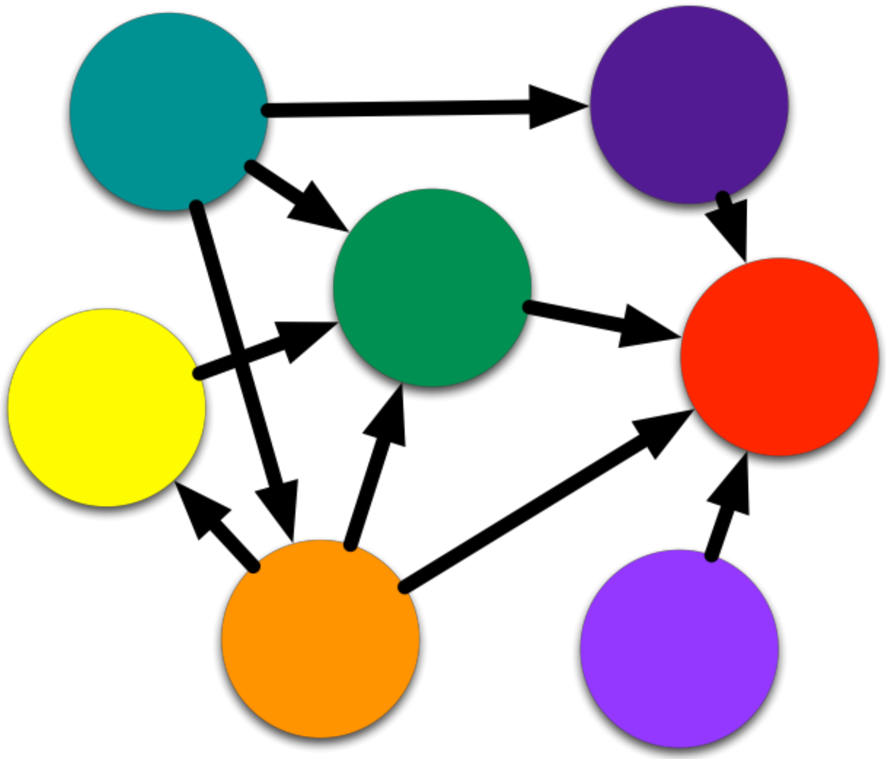
- Multi-user,
- Multi-platform,
- Multi-language,
- Multi-node,
- Always-on,
- Autoscaling,
- **Distributed Systems Nightmares!**

So Containers, Kubernetes and Microservices Saved the Day

Apps could be:

- Decomposed into independently deployable Containers
- Programatically orchestrated, driven by declarative configuration
- Developed in many different languages
  Java/Kotlin for Android, ObjC/Swift for IOS, Go/Java/Python/C/C++/… for Linux/Windows...
- Hooked together using service meshes
  Linkerd, Envoy, Istio…
- Configured, deployed, monitored and upgraded by expert devops/SREs (basically infrastructure Ninjas).

Turns out, it’s still really, really difficult…

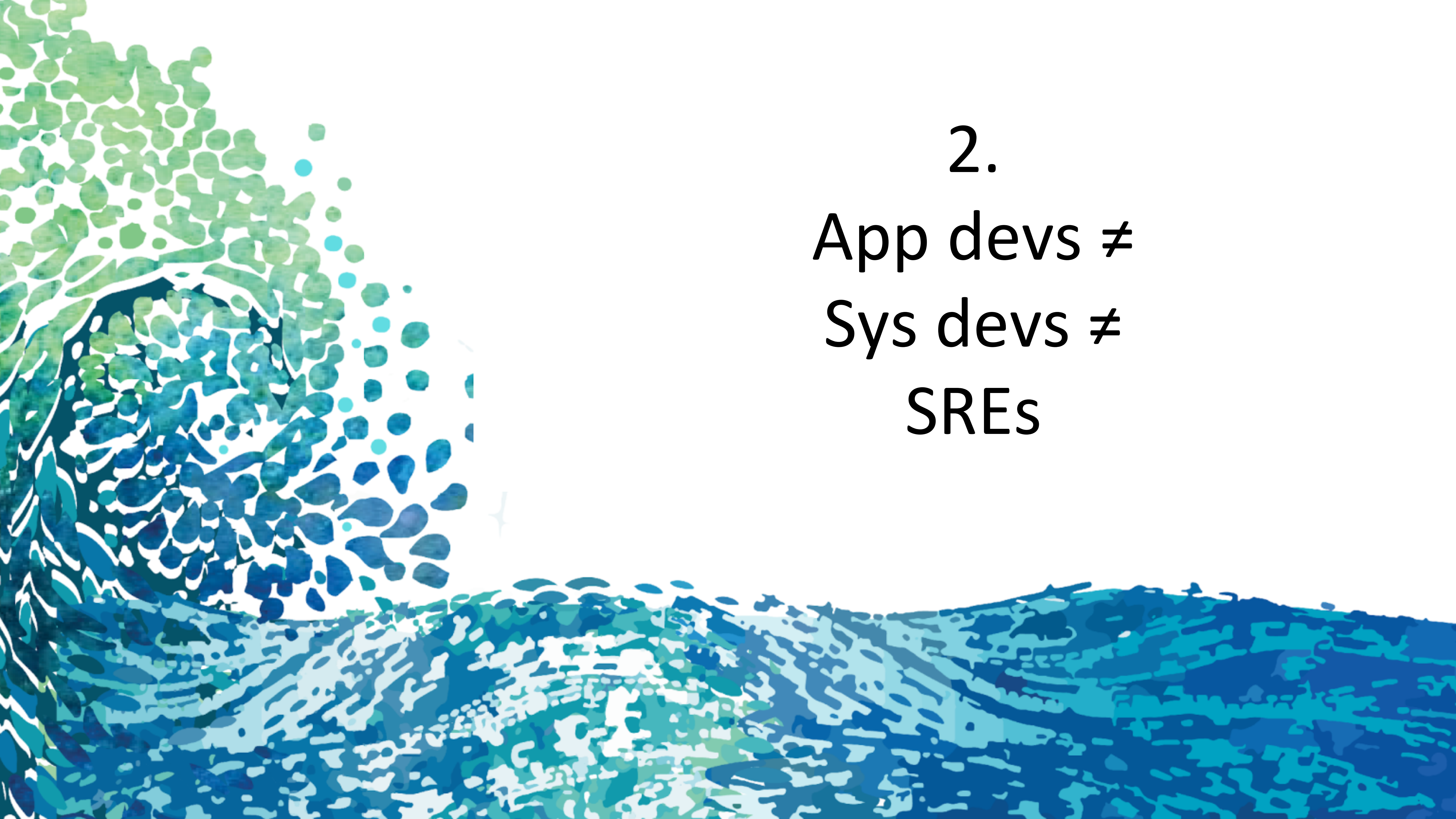


Developers still have to write the (really hard) stuff in the containers:

- distributed concurrency, synchronization,
- reliable RPC, fault tolerance,
- replication, leader election, sharding,
- code and data migration,
- observability, fault diagnosis
- As well as all the obvious
- remote invocation, load balancing, etc…

These sound like distributed systems problems!

PROFESSIONAL SYSTEMS PROGRAMMER REQUIRED.
DO NOT ATTEMPT AT HOME.



2.
App devs ≠
Sys devs ≠
SREs

Specialization...

App Devs

- Know their app domain very well.
  - Social Networking
  - Travel
  - Finance
  - …
- Need to move really fast.
- Don’t give a hoot about distributed systems algorithms, exponential backoff, PAXOS/Raft,...

Sys Devs

- Are really interested in understanding and solving hard distributed systems problems.
- Are in very short supply.
- Typically don’t understand your specific business needs.

SREs/DevOps Engineers

- Understand what happens when your specific customers hit your specific app, e.g.
  - Capacity/scaling requirements
  - Optimal sharding schemes
  - What breaks and why.
  - What needs to be replicated, updated etc and how.

3.
Amino OS from 30,000’

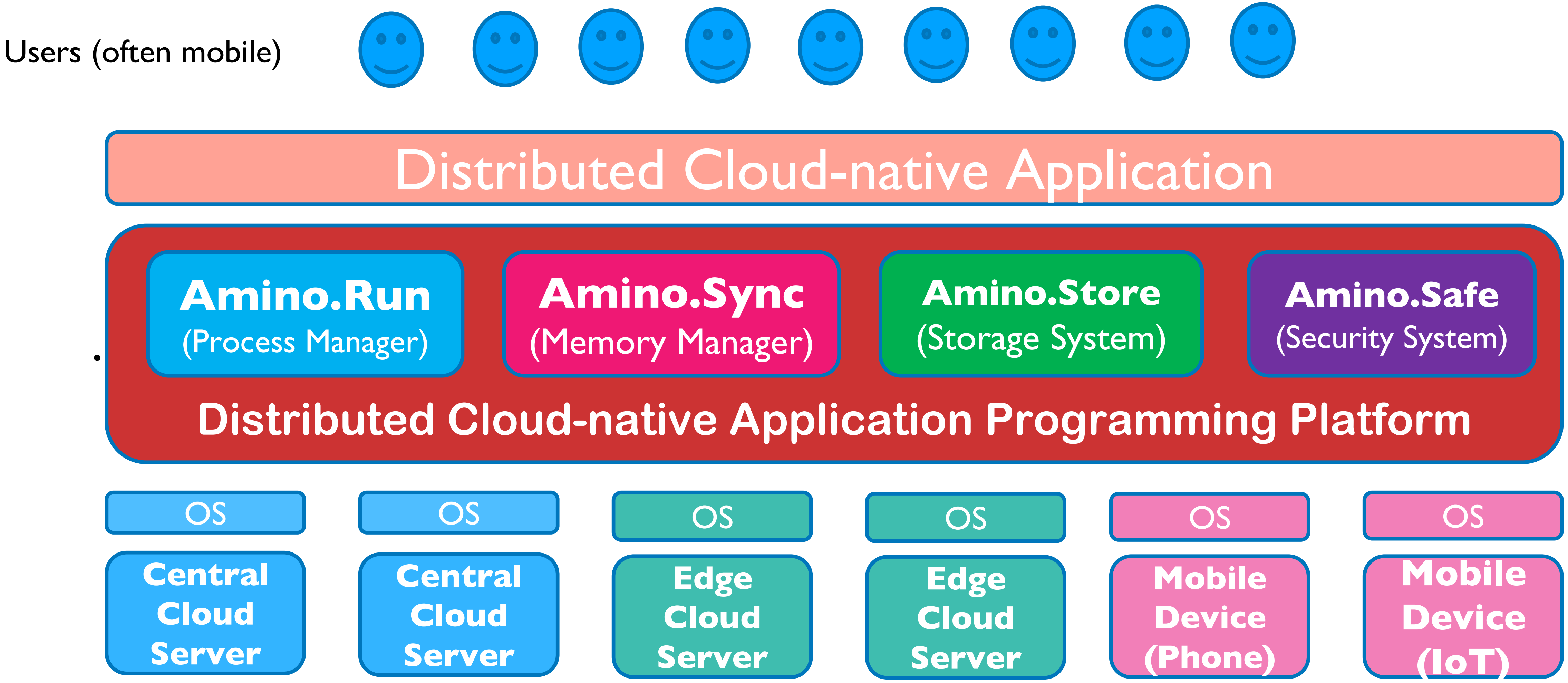
What is Amino OS?

Amino OS is an umbrella project, the goal of which is to create a distributed platform for coding and running distributed (cloud, edge and mobile) microservice-based applications. It has four main components:

- **Amino.Run**: A distributed microservice runtime (we'll focus on this today).
- **Amino.Sync**: A reactive data synchronization service that provides configurable consistency guarantees
- **Amino.Store**: A high-performance distributed transactional storage service
- **Amino.Safe**: A distributed privacy and security manager

Amino OS

Distributed Cloud-native Application Programming Platform



Users (often mobile)

Distributed Cloud-native Application

Amino.Run (Process Manager)

Amino.Sync (Memory Manager)

Amino.Store (Storage System)

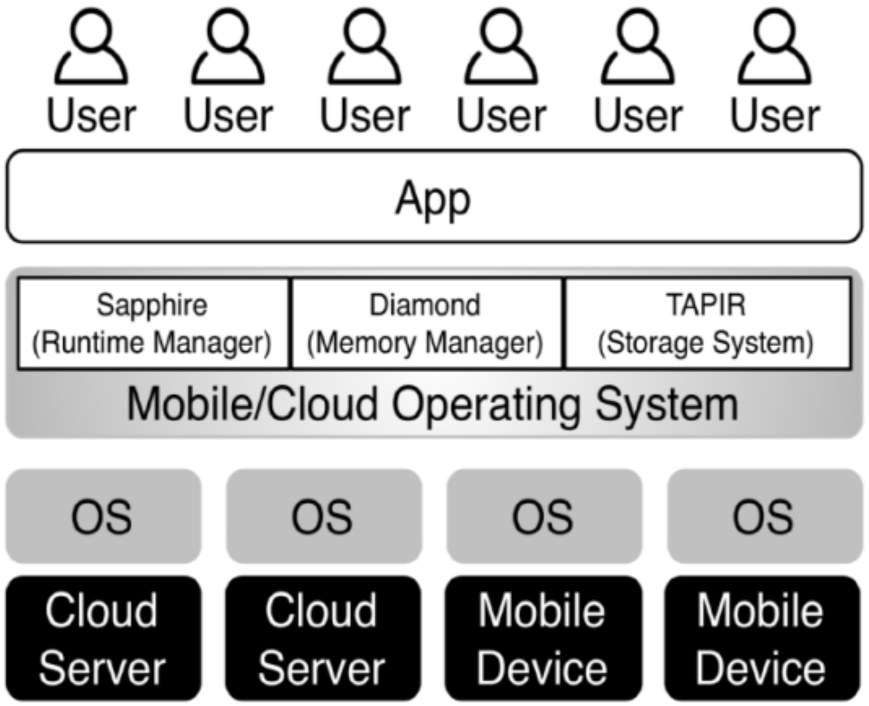
Amino.Safe (Security System)

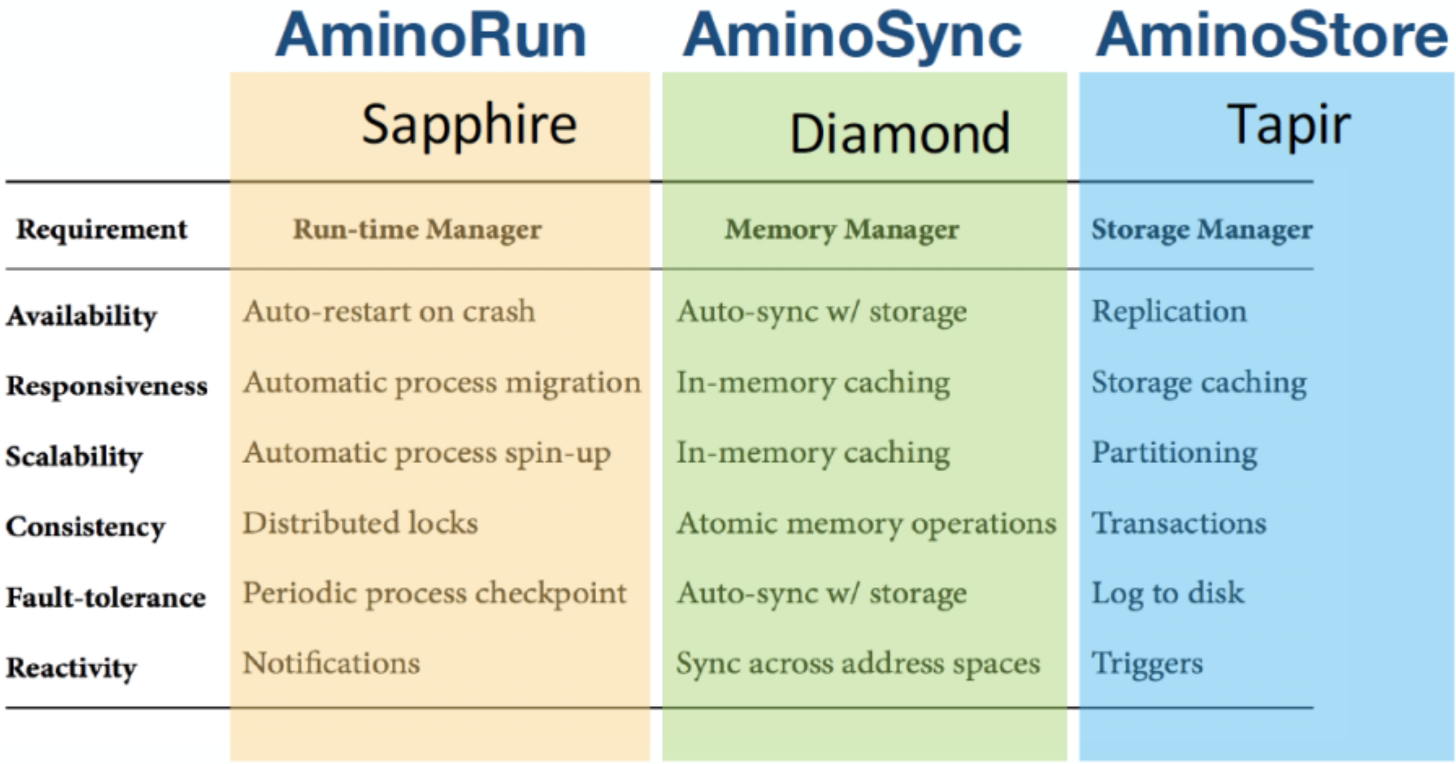
Distributed Cloud-native Application Programming Platform

OS	OS	OS	OS	OS	OS
Central Cloud Server	**Central Cloud Server**	**Edge Cloud Server**	**Edge Cloud Server**	**Mobile Device (Phone)**	**Mobile Device (IoT)**

What is Amino OS?

- Amino OS is based on several years of distributed systems research done by Irene and her team at the University of Washington Systems Lab in Seattle, WA.
- Amino OS is the result of 2 years of collaboration between Quinton, Venu and Irene's teams.





	AminoRun	AminoSync	AminoStore
	Sapphire	Diamond	Tapir
Requirement	**Run-time Manager**	**Memory Manager**	**Storage Manager**
Availability	Auto-restart on crash	Auto-sync w/ storage	Replication
Responsiveness	Automatic process migration	In-memory caching	Storage caching
Scalability	Automatic process spin-up	In-memory caching	Partitioning
Consistency	Distributed locks	Atomic memory operations	Transactions
Fault-tolerance	Periodic process checkpoint	Auto-sync w/ storage	Log to disk
Reactivity	Notifications	Sync across address spaces	Triggers

We’ll Focus on Amino.Run in this Talk

- Goals
- Architecture and How it Works
- Deployment Managers
- Experience and Evaluation
- Demo
- Q&A

Amino.Run Goals

1. Separate microservice application logic from system and deployment code.

2. Make application code extremely simple and intuitive

3. Allow devs and SRE's to easily make, combine and change automated application deployment choices across arbitrary servers and devices (cloud, edge, mobile, IoT etc)

4. Support arbitrary programming languages

5. Performance!

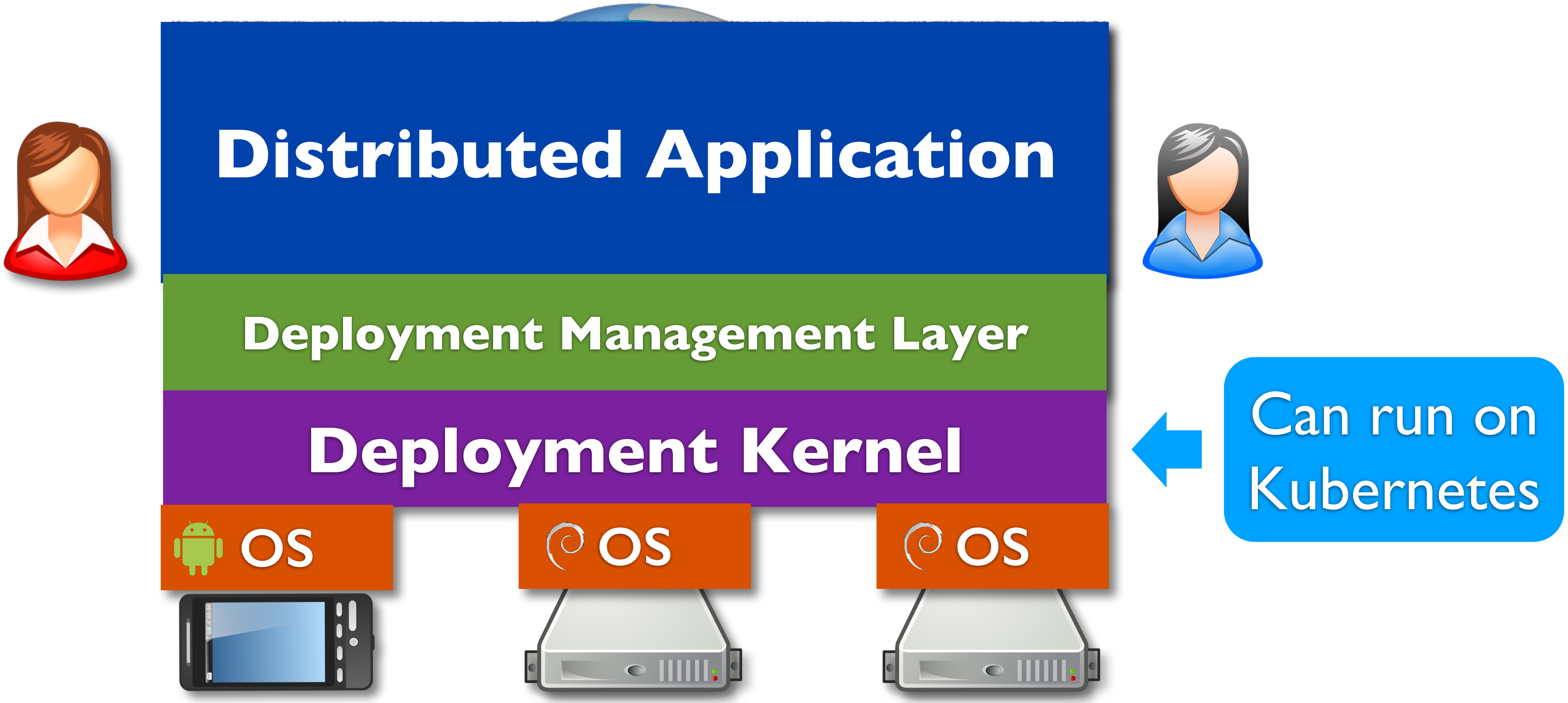
6. Optionally integrate with external infrastructure systems (like Kubernetes, Istio etc) in a very natural way.

Our Solution

A new system architecture that supports:

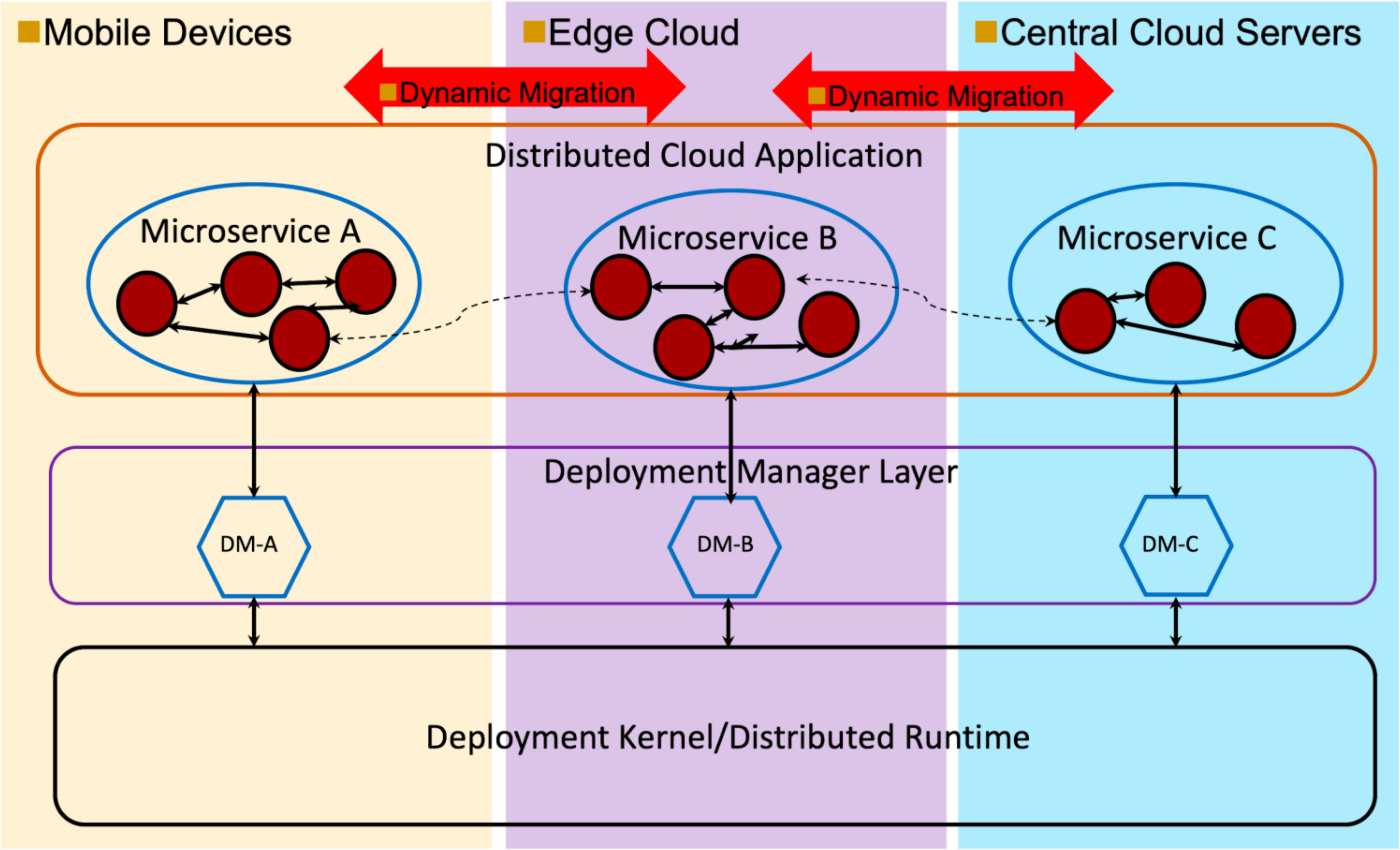
- pluggable and extensible <u>deployment managers</u>
- across arbitrary programming languages (using GraalVM)
- and operating systems

Amino.Run Architecture



Can run on Kubernetes

Example Deployment – Edge Cloud

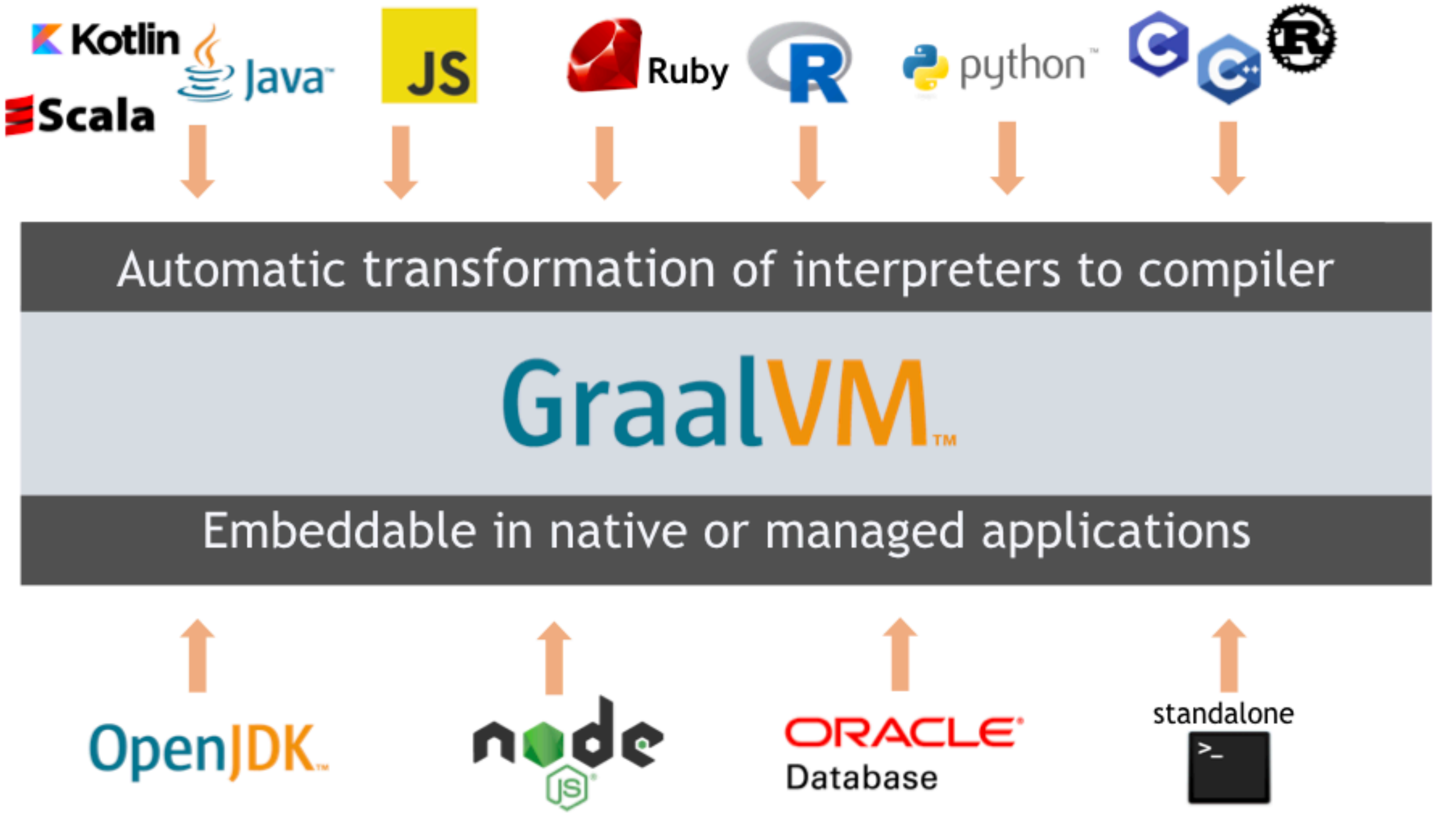


Amino.Run Application

Partitioned into ***Microservices***, which:

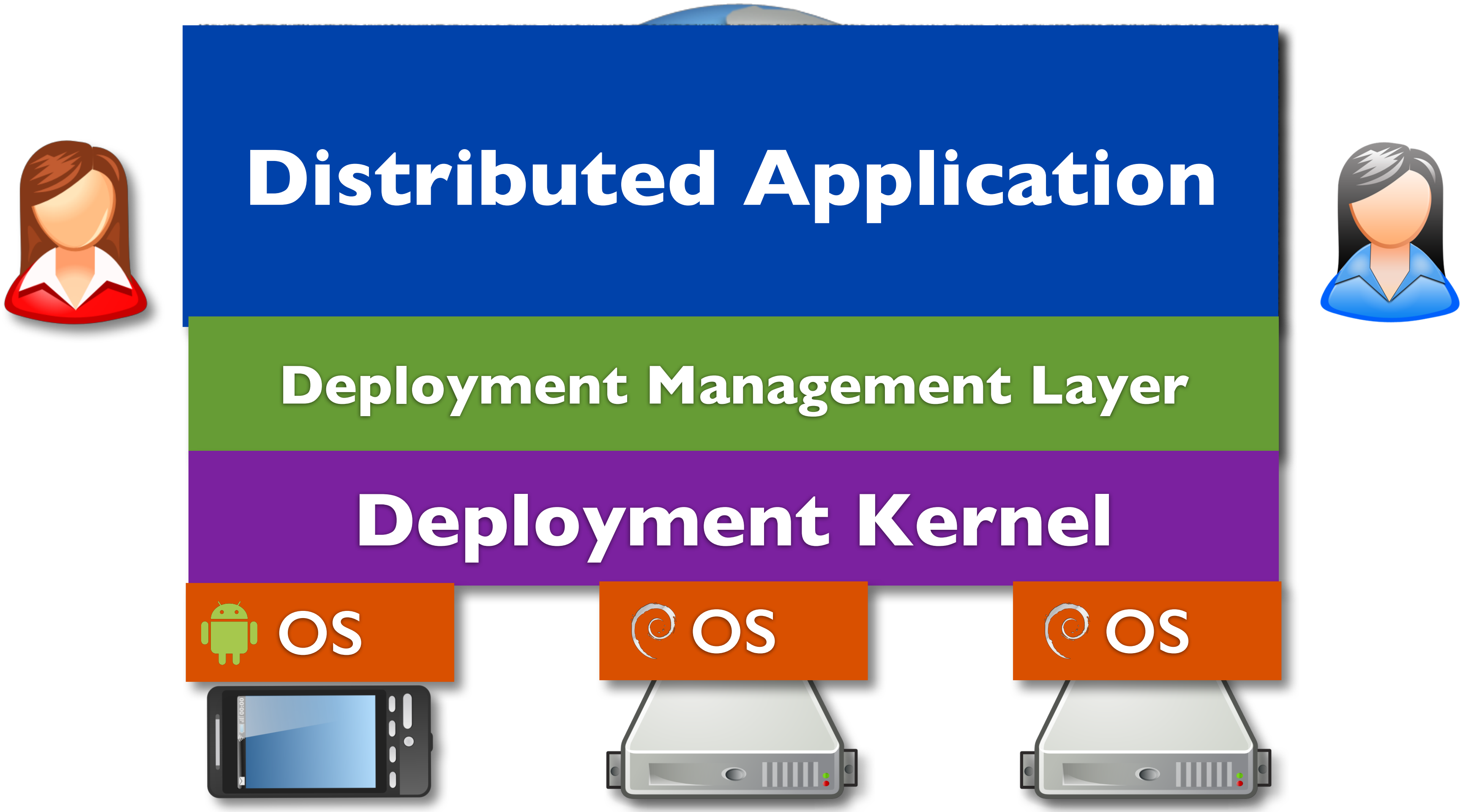
- Run in a single address space with transparent RPC.
- Execute anywhere and move transparently and intelligently.
- Provide a unit of distribution for deployment managers.
- May be written in any programming language (using GraalVM)
- Can pass data structures transparently between programming languages (using GraalVM Polyglot)

A brief word about multi-language and GraalVM


Kotlin
Scala
Java
JS
Ruby
R
python
Automatic transformation of interpreters to compiler
GraalVM
Embeddable in native or managed applications
OpenJDK
node JS
ORACLE
Database
standalone
>_

- High-performance polyglot VM (think JVM)
- Native via Ahead-of-Time compilation, or JIT
- Embeddable
- Allows Microservices, Amino Kernel and DMs all in different languages

Amino.Run Architecture

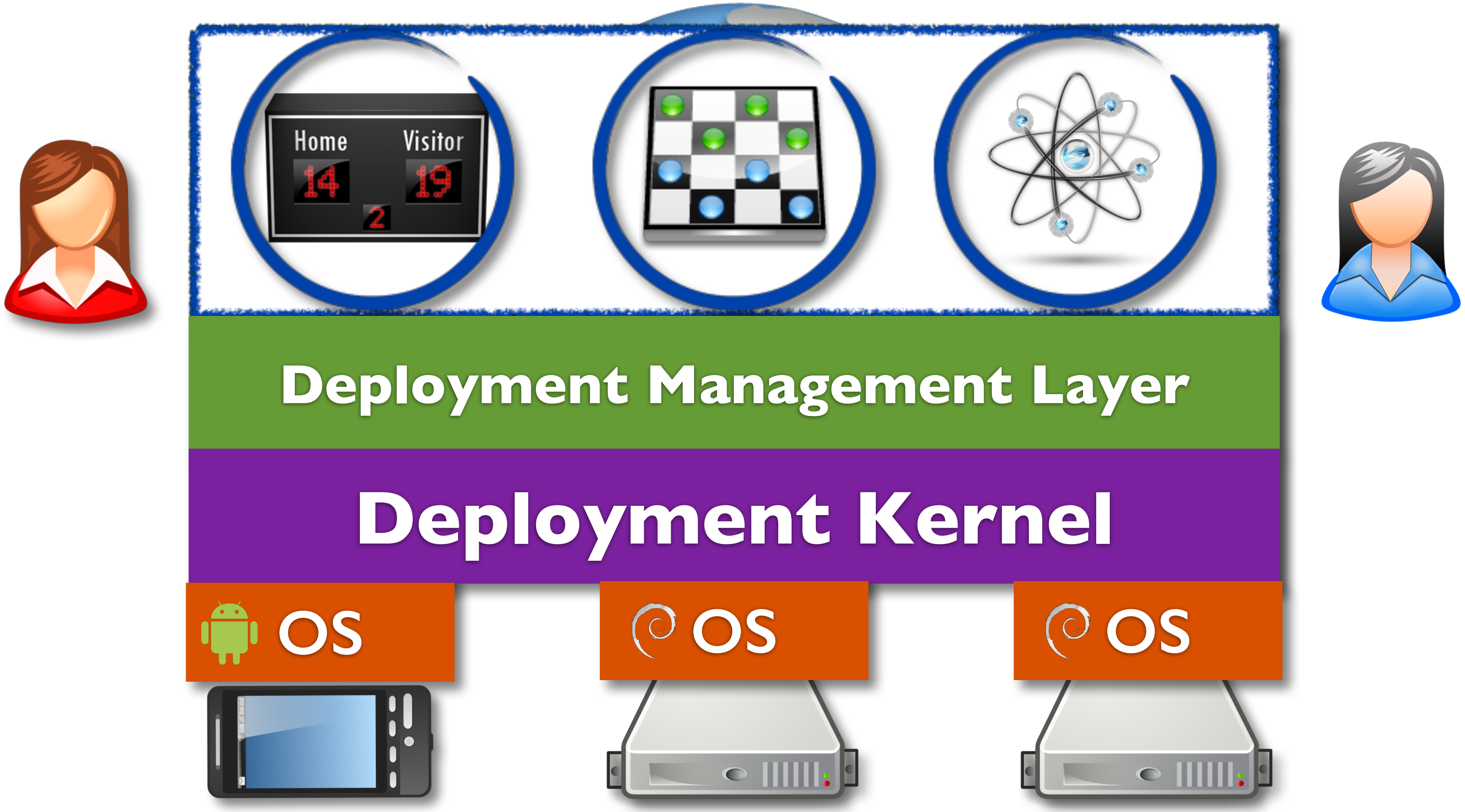


Deployment Kernel

Provides ***best-effort distribution services***, including:

- Microservice instantiation, replication, tracking, and migration.
- Making and routing RPC to Microservices.
- Managing, distributing and running deployment managers.

Amino.Run Architecture

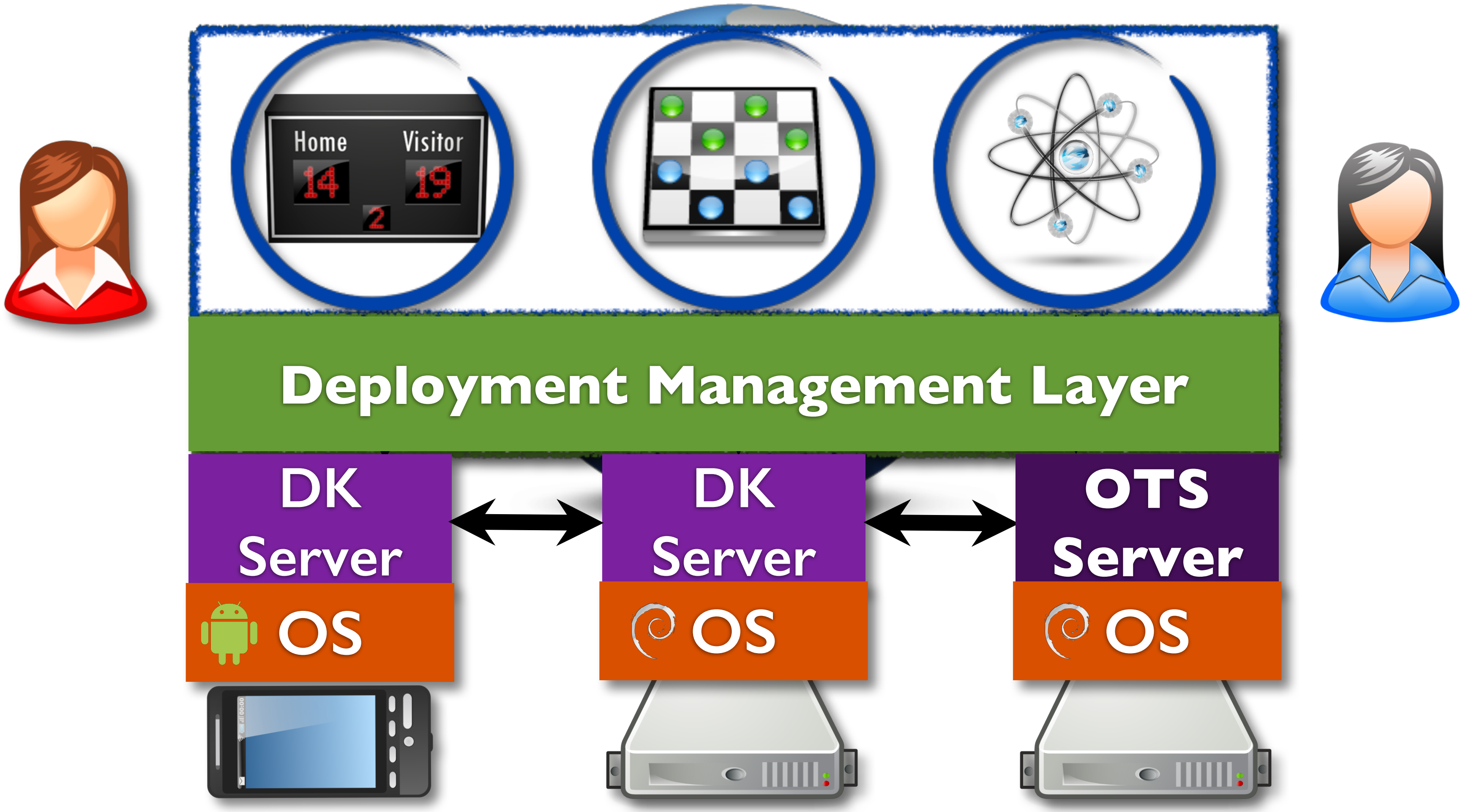


Deployment Management Layer

Consists of ***deployment managers***, which:

- Extend the functions and guarantees of the deployment kernel.
  - Sharding, Method Replication, Caching etc
- Interpose on Microservice calls and events.
- Easy to choose and change <u>without</u> modifying the application.
- Can be arbitrarily combined! (with some obvious restrictions)
  - E.g. Replicated shards, Transactional replicas, Retries over sharded transactions, etc…

Amino.Run Architecture



Deployment Management Layer

DK Server

DK Server

OTS Server

OS

OS

OS

Deployment Managers

Caching

Distributed Transaction Management

Checkpointing

Replication

Autoscaling

Dynamic Code and Data Migration

And many more…

Deployment Manager Library

Primitives	Caching	Serializability	Checkpoint	Replication	Mobility	Scalability
Immutable	Explicit Caching	Serializable RPC	Explicit Checkpoint	RSM-Cluster	Explicit Migration	LoadBalanced Frontend
AtLeastOnce RPC	Lease Caching	Locking Transactions	Periodic Checkpoint	RSM-Geo	Dynamic Migration	Scale-up Frontend
Keep In Place	Writethrough Caching	Optimistic Transactions	DurableRPC	RSM-P2P	Explicit Code-offload	LB Master-slave
Keep On Device	Consistent Caching		Durable Transactions		Code-offload	
Keep In Cloud						

Extensible with the Deployment Manager API!

Outline

1. Architecture

2. Deployment Managers

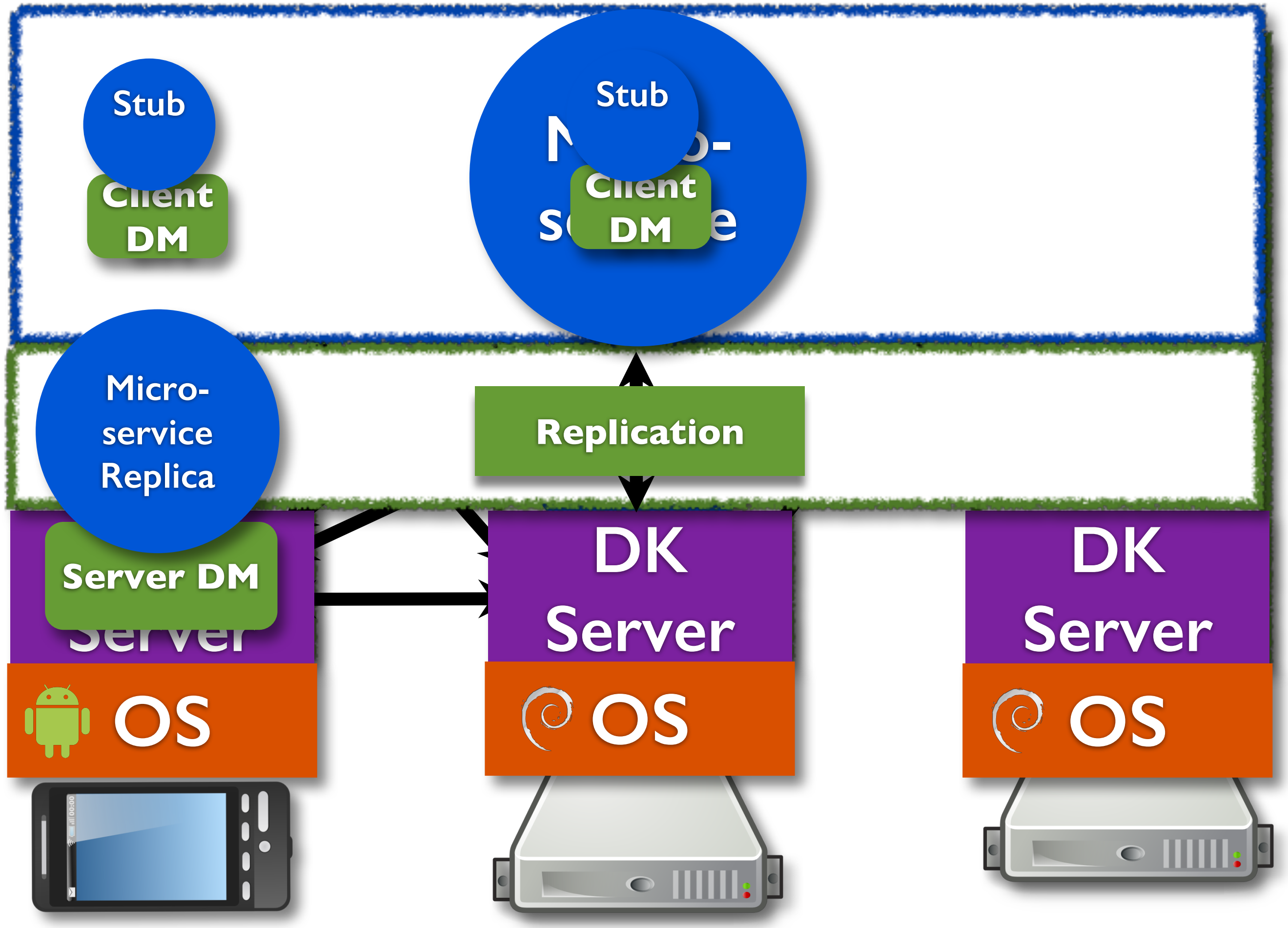
3. Experience and Evaluation

Deployment Manager API

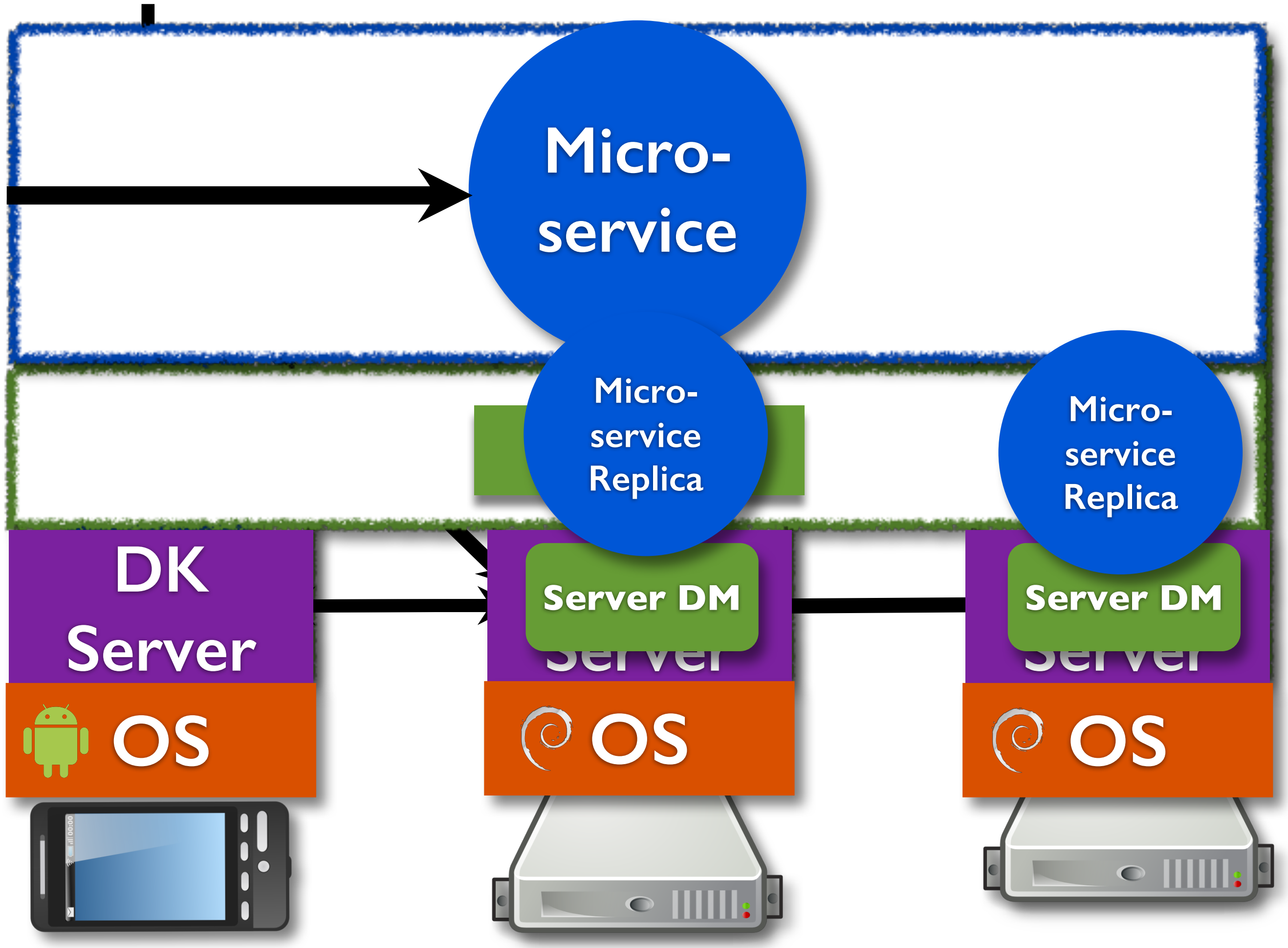
Deployment Manager ("DM") components, which the Amino.Run kernel creates, deploys and invokes automatically:

- **Server-Side DMs:** Co-located with the Microservice Replica (i.e. server process/container).
- **Client-Side DMs:** Co-located with remote references to the Microservice.
- **Group Coordinator DMs:** Co-located with fault-tolerant Microservice Management Service (MMS aka OMS).

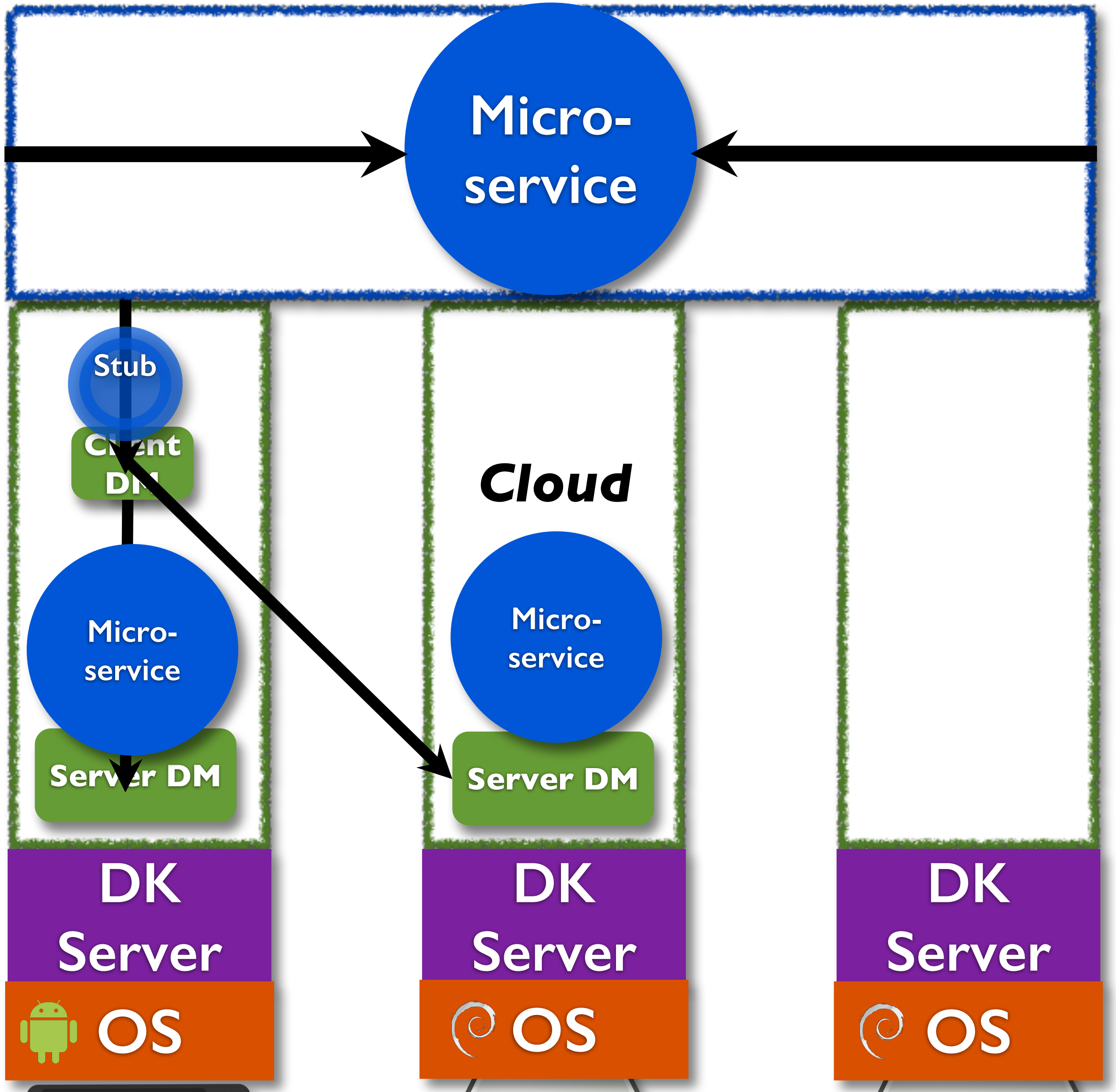
Deployment Manager Architecture



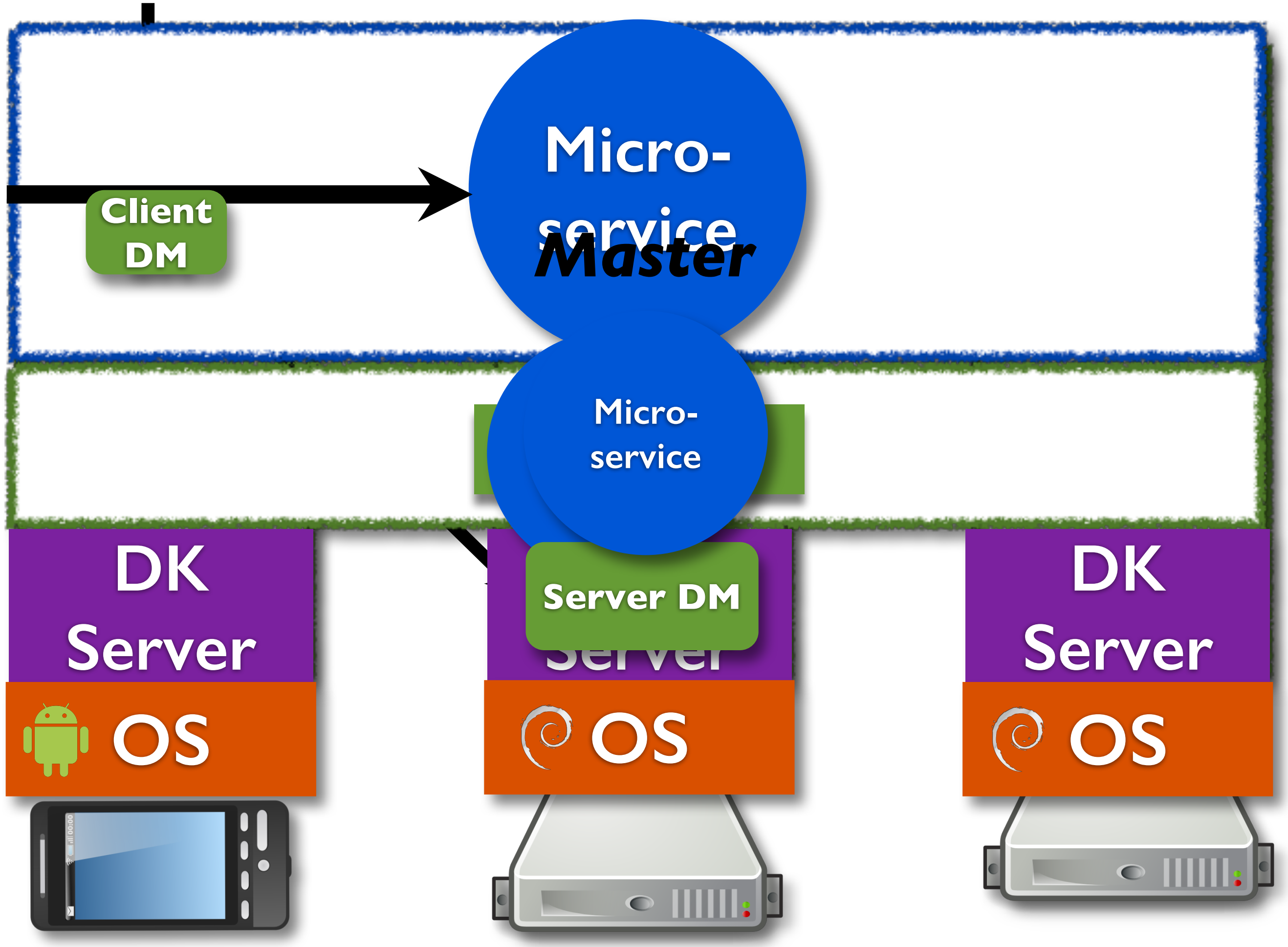
Replicating a Microservice



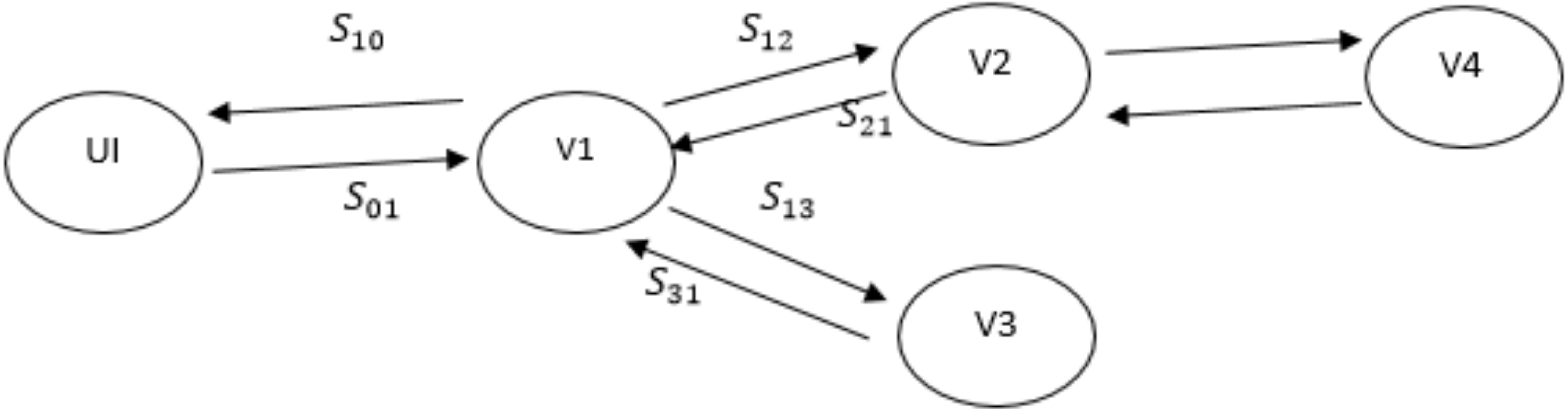
Offloading a Microservice



Caching Microservice State

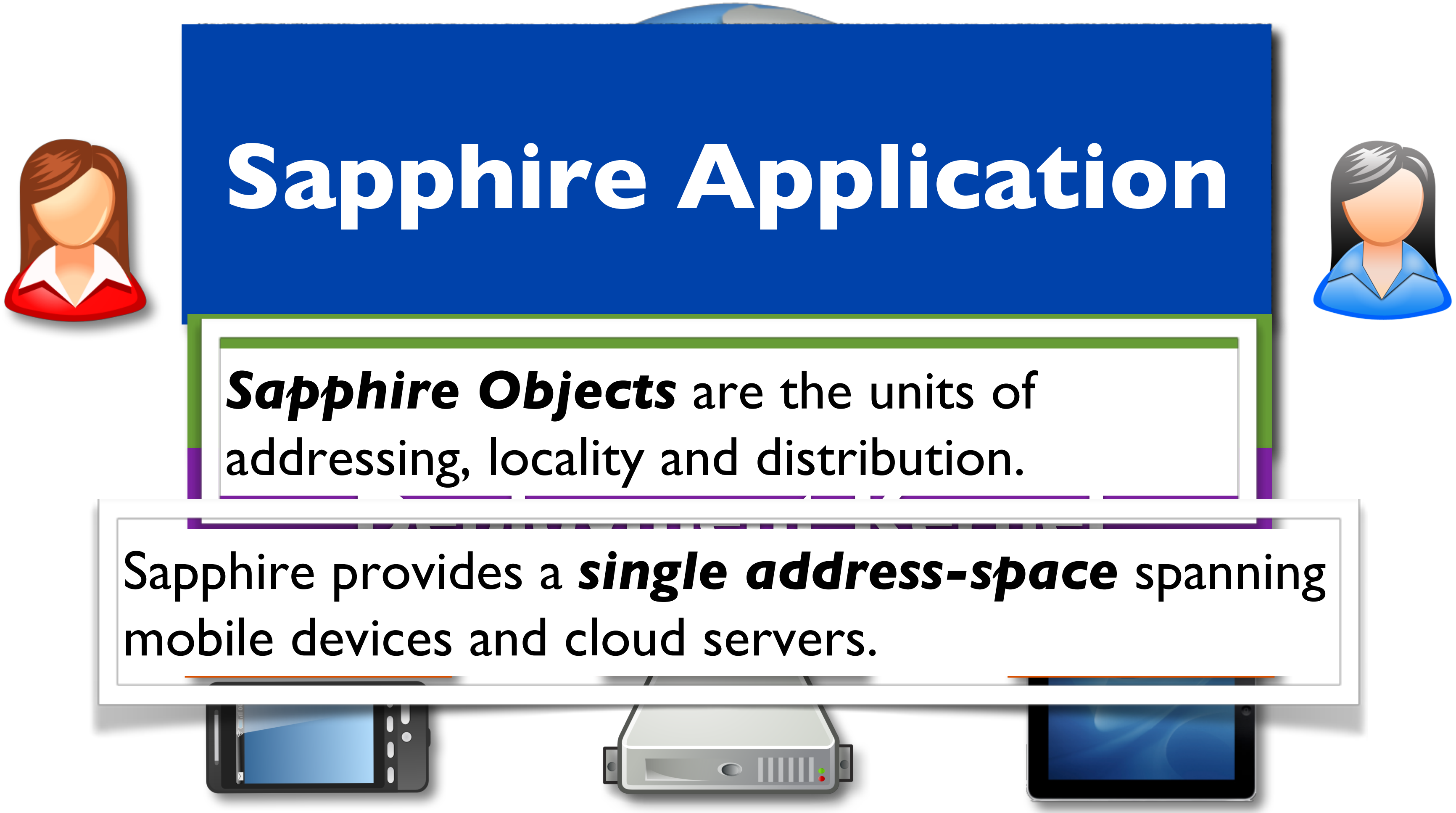


- Automatic Stateful Object Migration



- See more in the demo later.

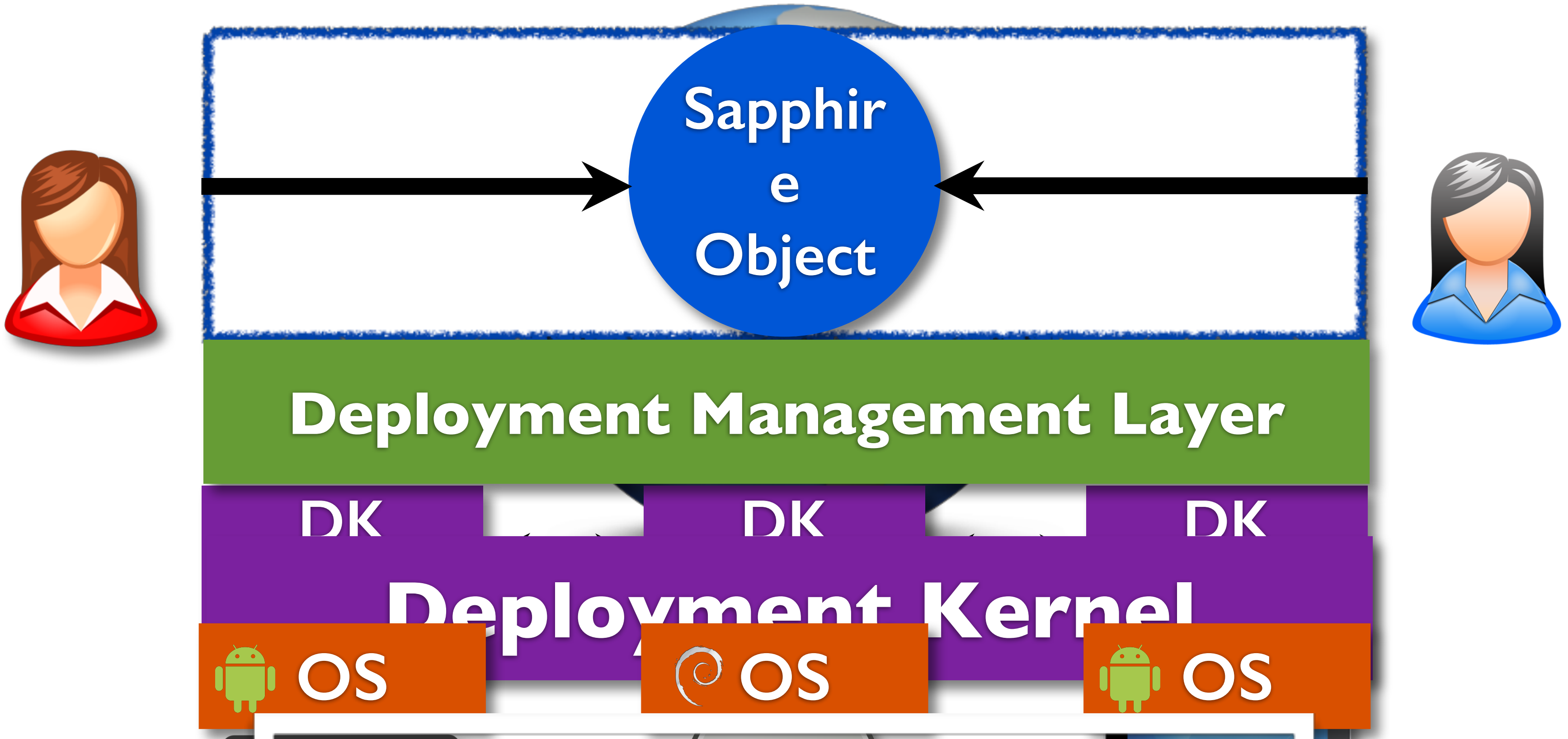
Sapphire Architecture



Sapphire Objects are the units of addressing, locality and distribution.

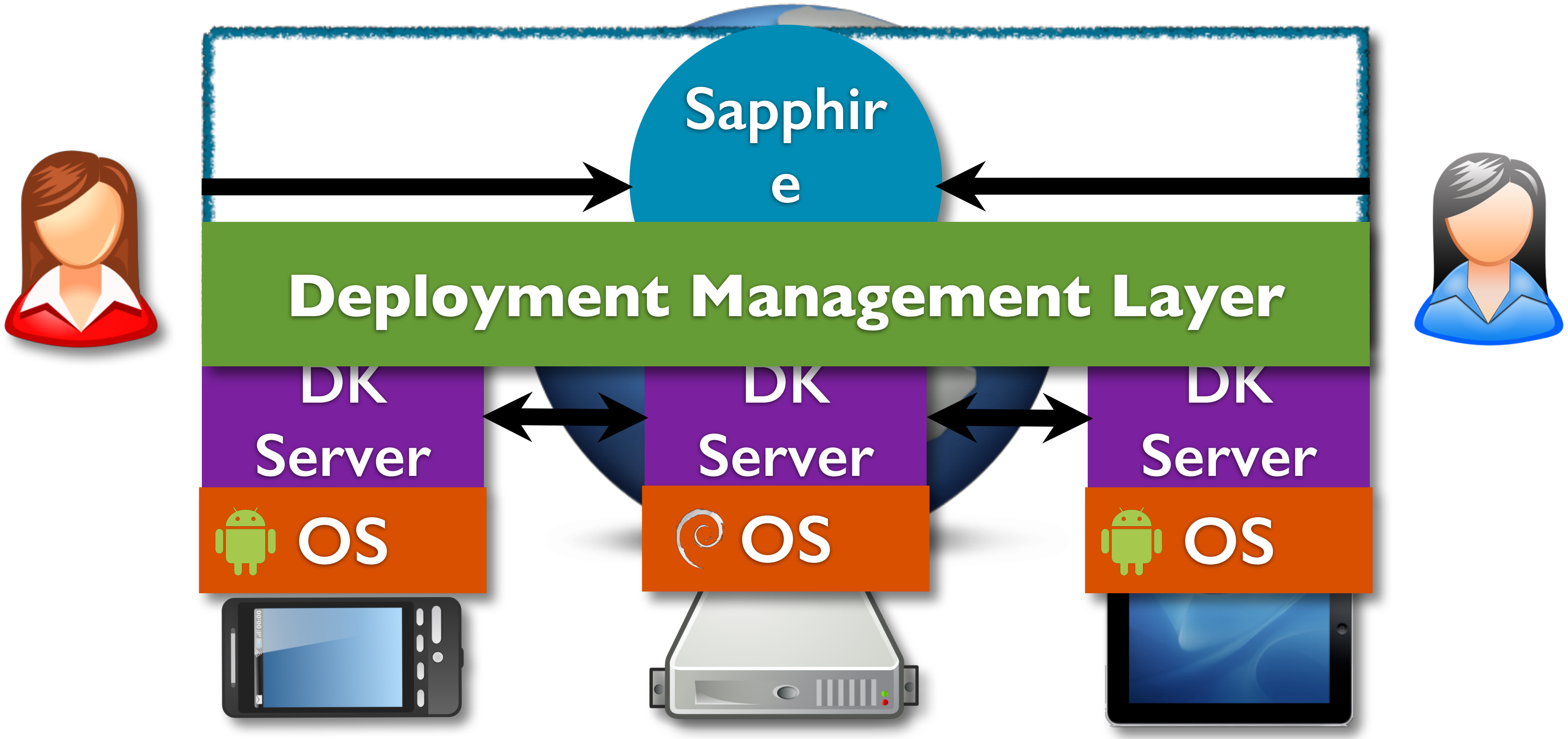
Sapphire provides a ***single address-space*** spanning mobile devices and cloud servers.

Sapphire Architecture: Deployment Kernel

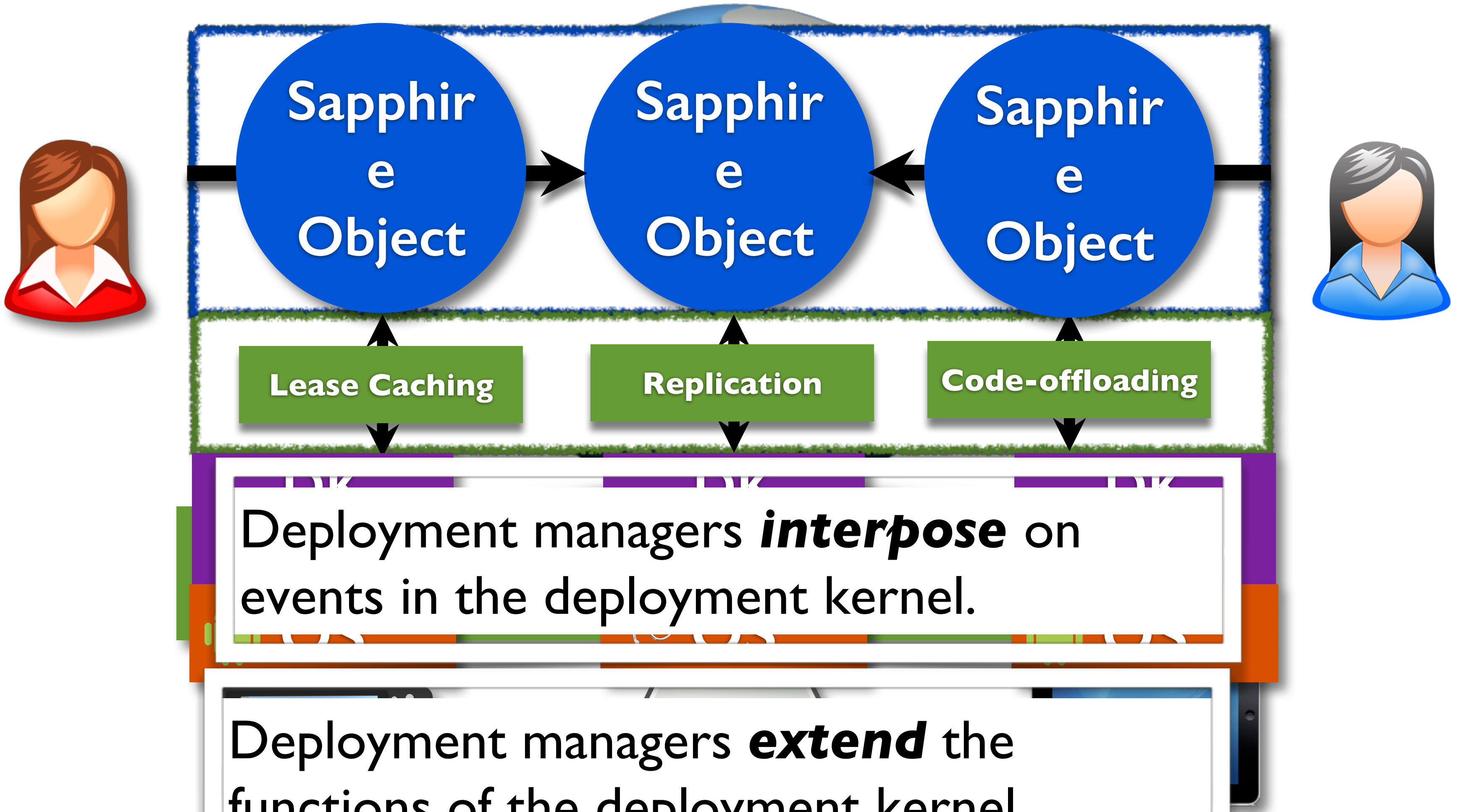


The deployment kernel provides core distribution services with ***few guarantees***.

Sapphire Architecture:
Deployment Managers



Sapphire Architecture:
Deployment Managers

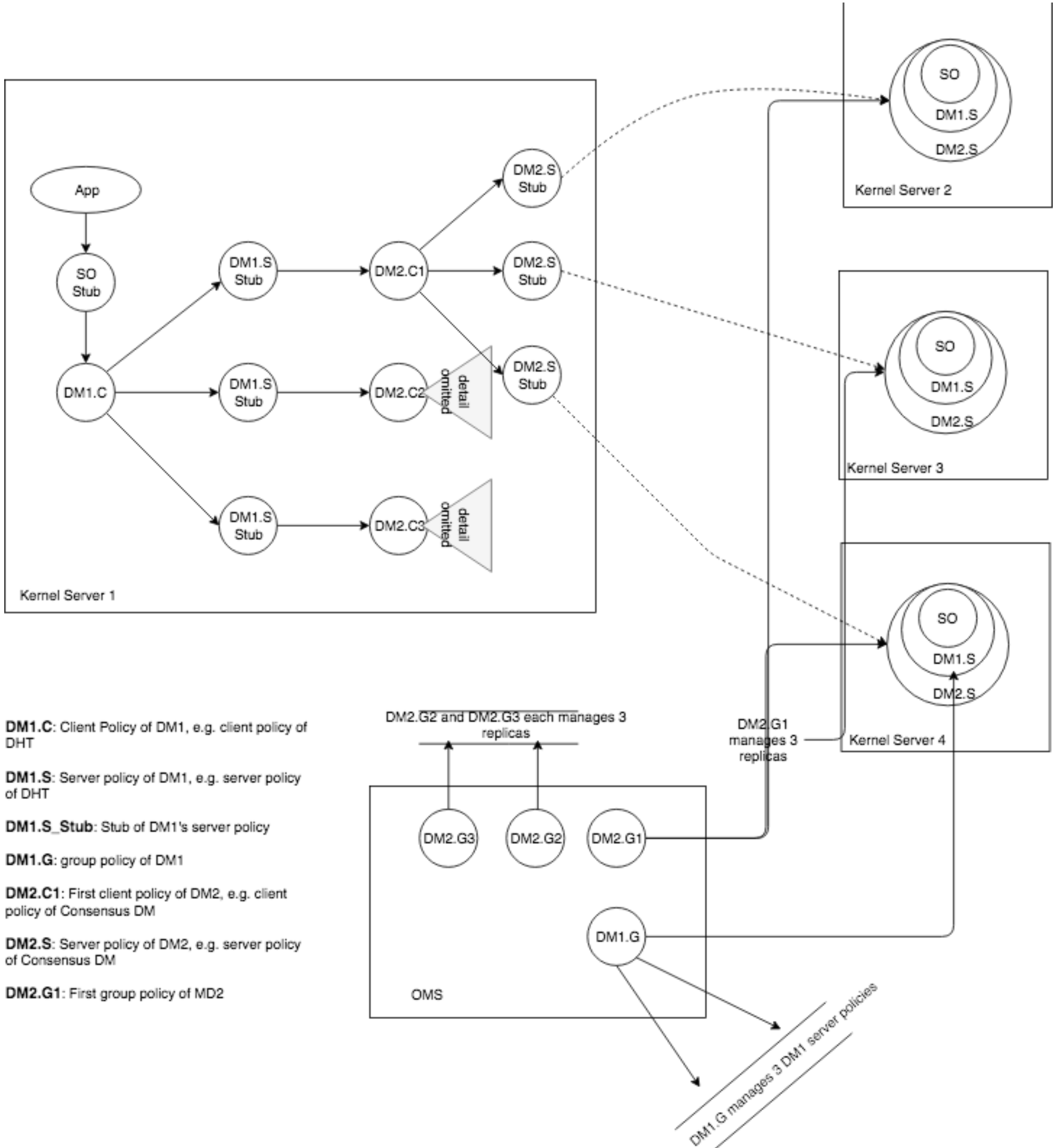


Deployment managers ***interpose*** on events in the deployment kernel.

Deployment managers ***extend*** the functions of the deployment kernel.

A Brief Note Regarding Composition of Deployment Managers

- Implemented through chaining deployment managers
- Done automatically by the kernel
- SRE just provides ordering (via config)



DM1.C: Client Policy of DM1, e.g. client policy of DHT

DM1.S: Server policy of DM1, e.g. server policy of DHT

DM1.S_Stub: Stub of DM1's server policy

DM1.G: group policy of DM1

DM2.C1: First client policy of DM2, e.g. client policy of Consensus DM

DM2.S: Server policy of DM2, e.g. server policy of Consensus DM

DM2.G1: First group policy of MD2

Outline

1. **Architecture**
2. **Deployment Managers**
3. **Experience and Evaluation**

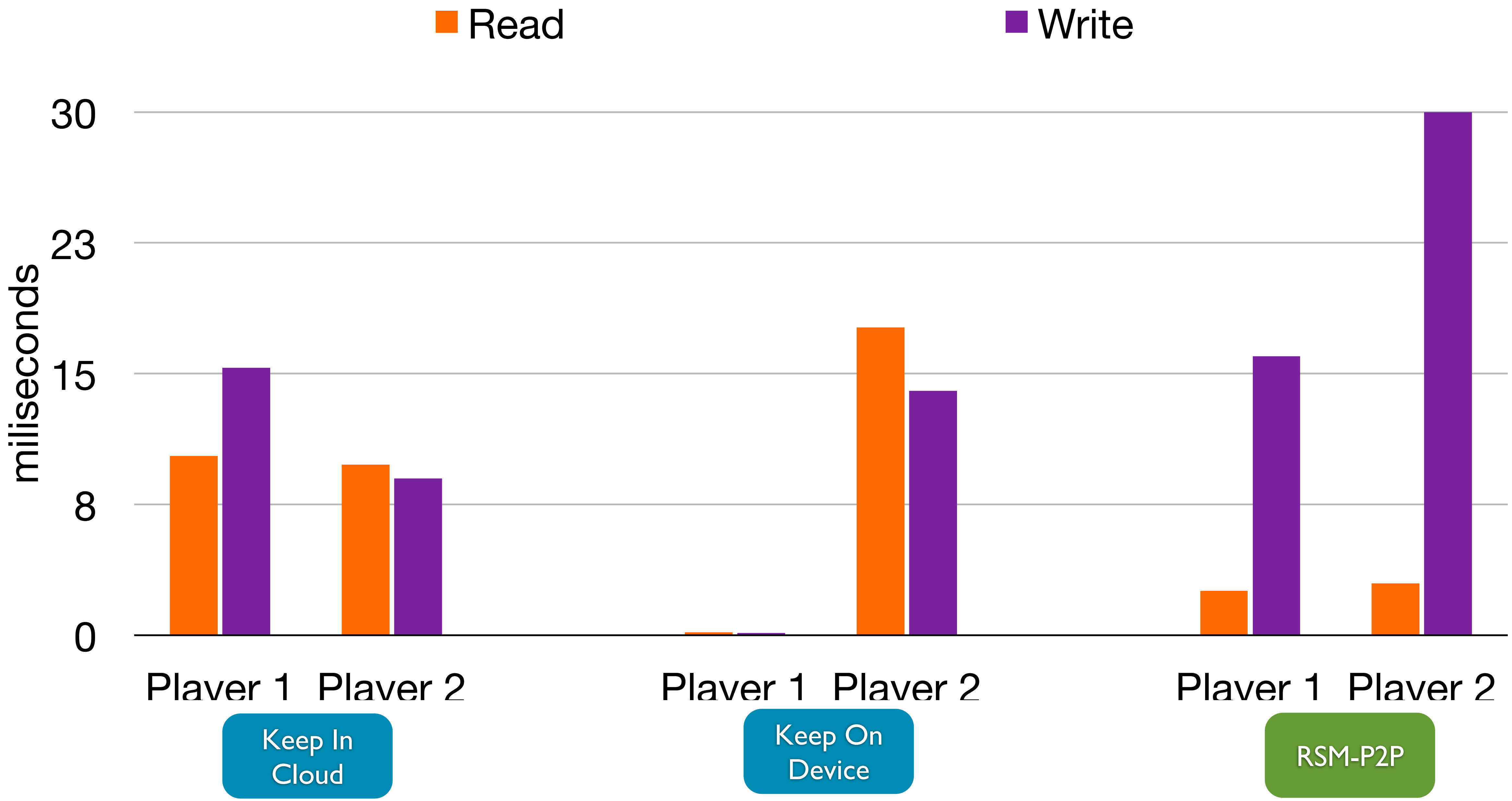
Experimental Setup

	Dell Server	Nexus 7	Nexus S
CPU	8-core Intel Xeon 2GHz	4-core ARM Cortex A9 1.3GHz	1-core ARM Cortex A8 1GHz
RAM	8GB	1GB	512MB
OS			

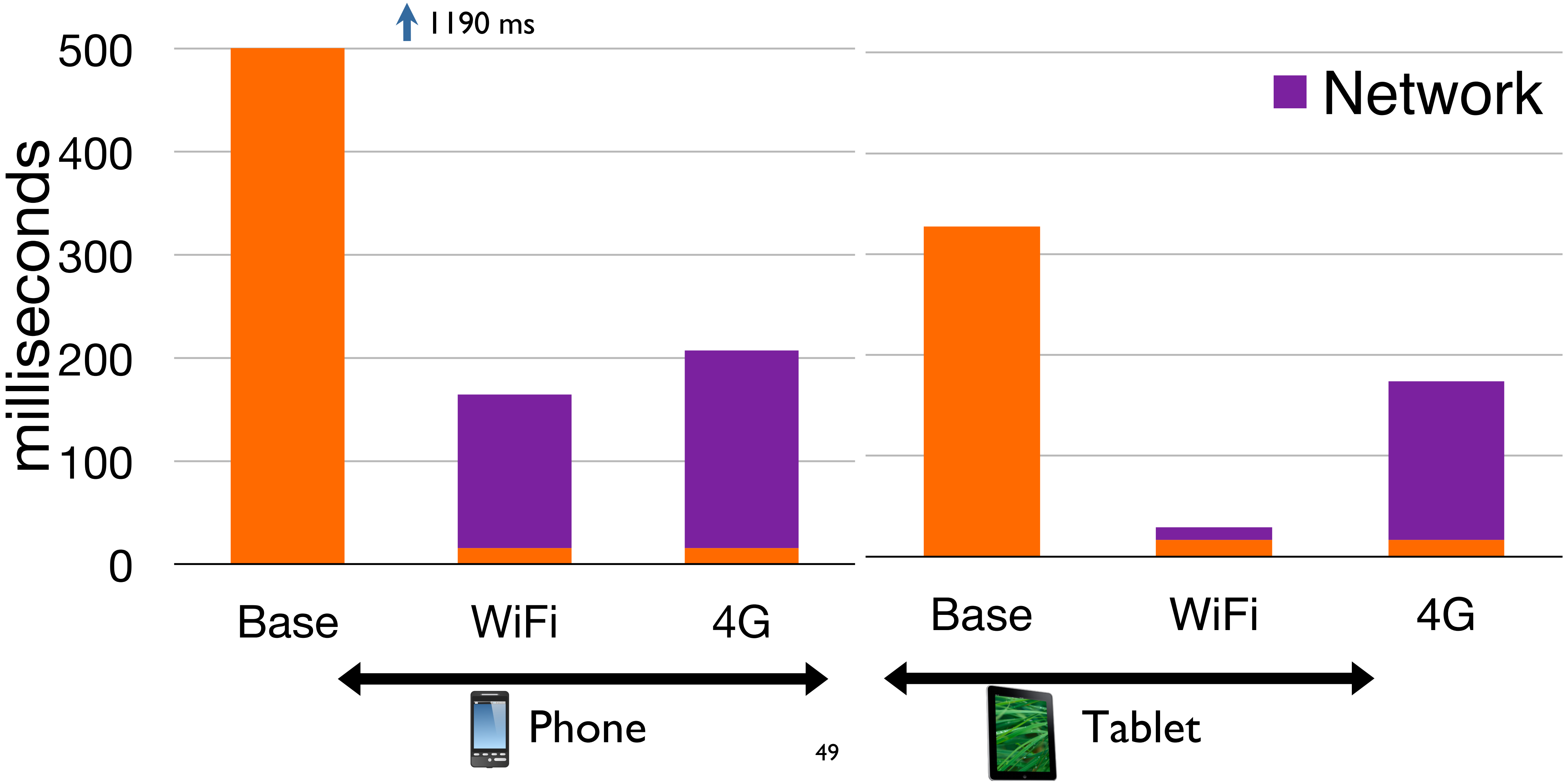
Experimental Setup

	Dell Server	Nexus S
CPU	8-core Intel Xeon 2GHz	1-core ARM Cortex A8 1GHz
RAM	8GB	512MB
OS		

Peer-to-Peer Multiplayer Game



Code-offloading for Physics Engine



Summary

Modern microservices implement difficult distributed deployment tasks.

Amino.Run is a new programming system for deploying interesting distributed applications including cloud-native, mobile/cloud, edge/cloud.

Deployment managers makes it easy to choose, combine, and customize deployment options.

Next Steps?

- Migrating state that’s not inside the application or Amino system (e.g. local files, Linux timers etc).
- Some rough edges between certain language combinations.
- Additional plugins for external systems (Istio, etcd, TiKV, etc)
- Federations and disconnected Edge scenarios.

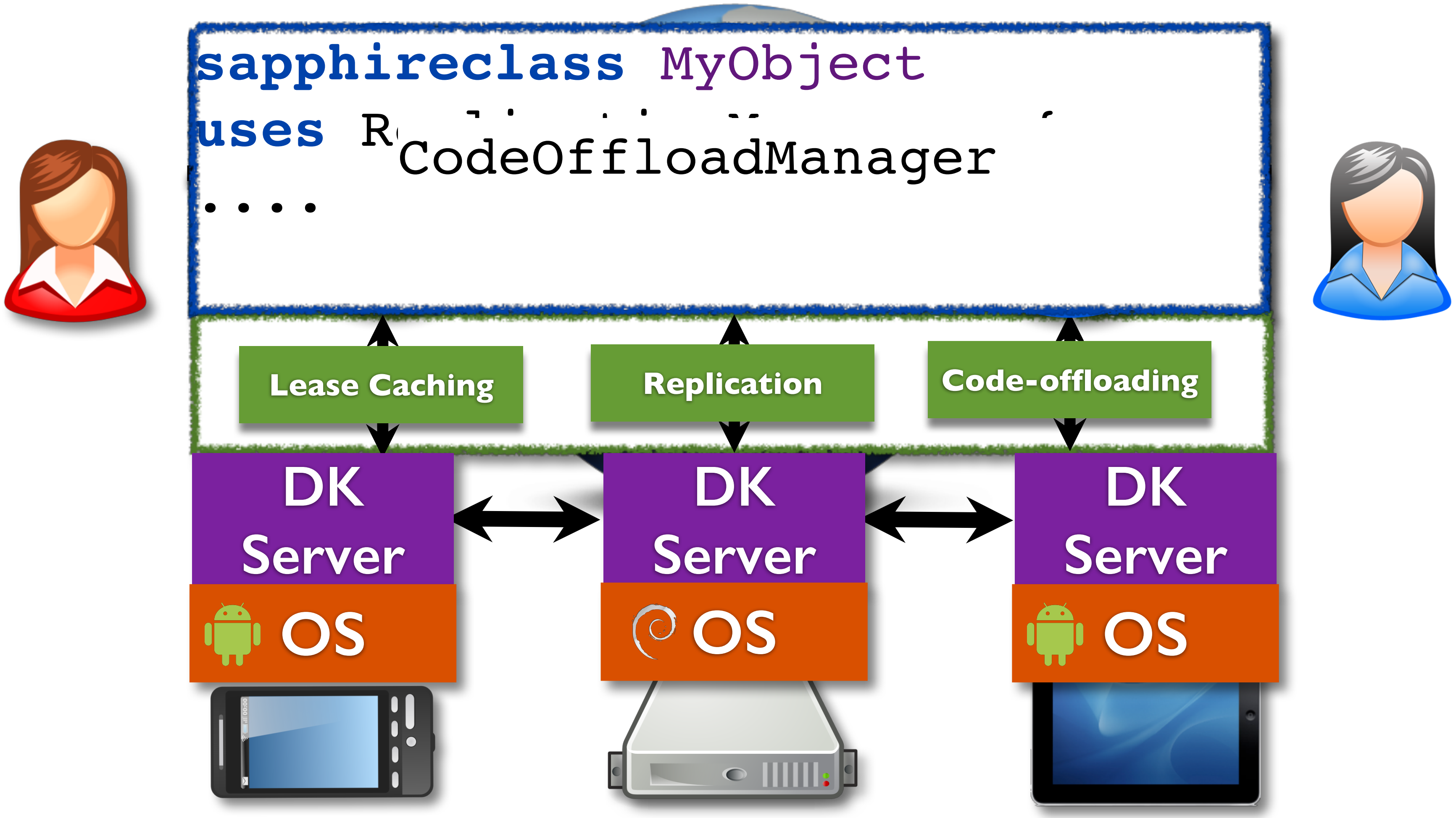
Get Involved

- Slack channel: Amino-OS.slack.com
- Web site: www.Amino-OS.io
- Contributions most welcome
- Repo: github.com/Amino-OS/Amino.Run

Demo

Q & A

Sapphire Architecture:
Deployment Managers

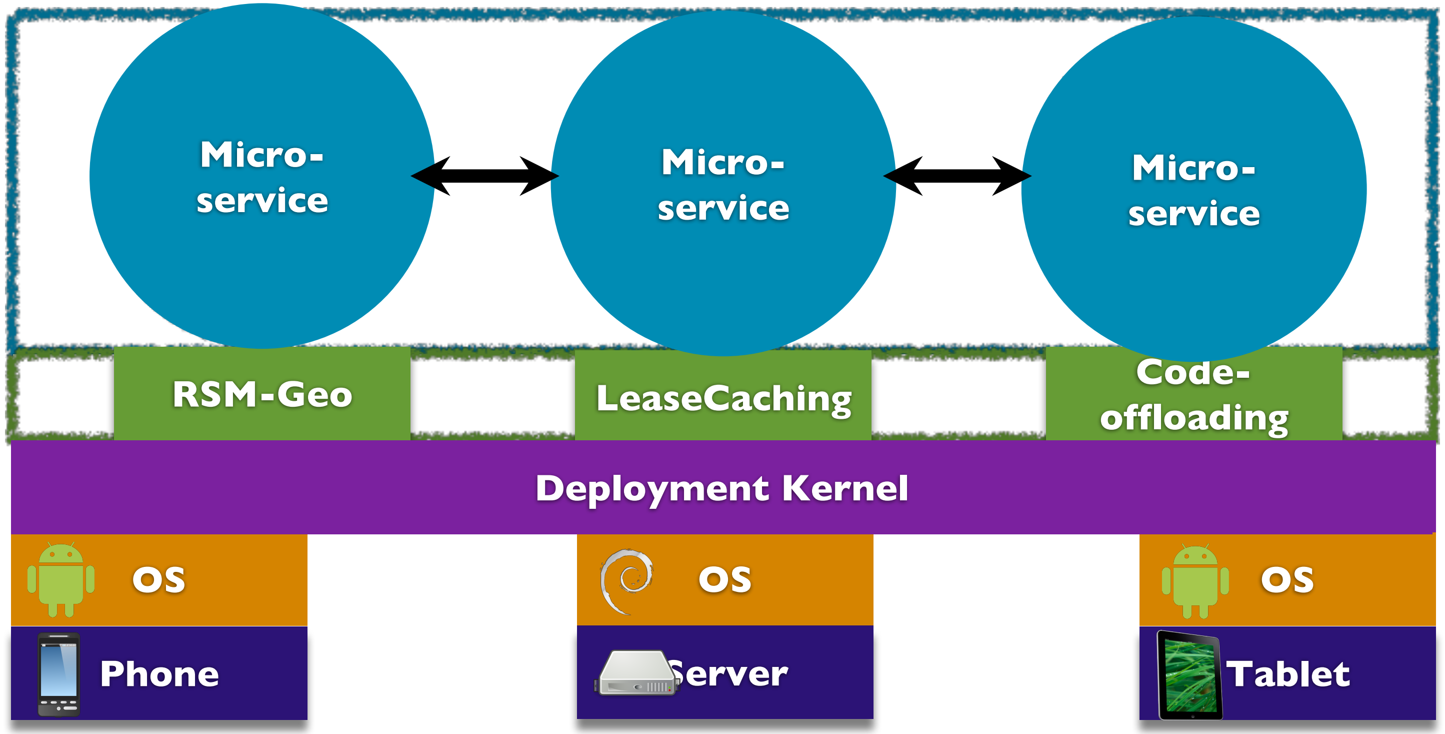


```
sapphireclass MyObject
uses R CodeOffloadManager
....
```

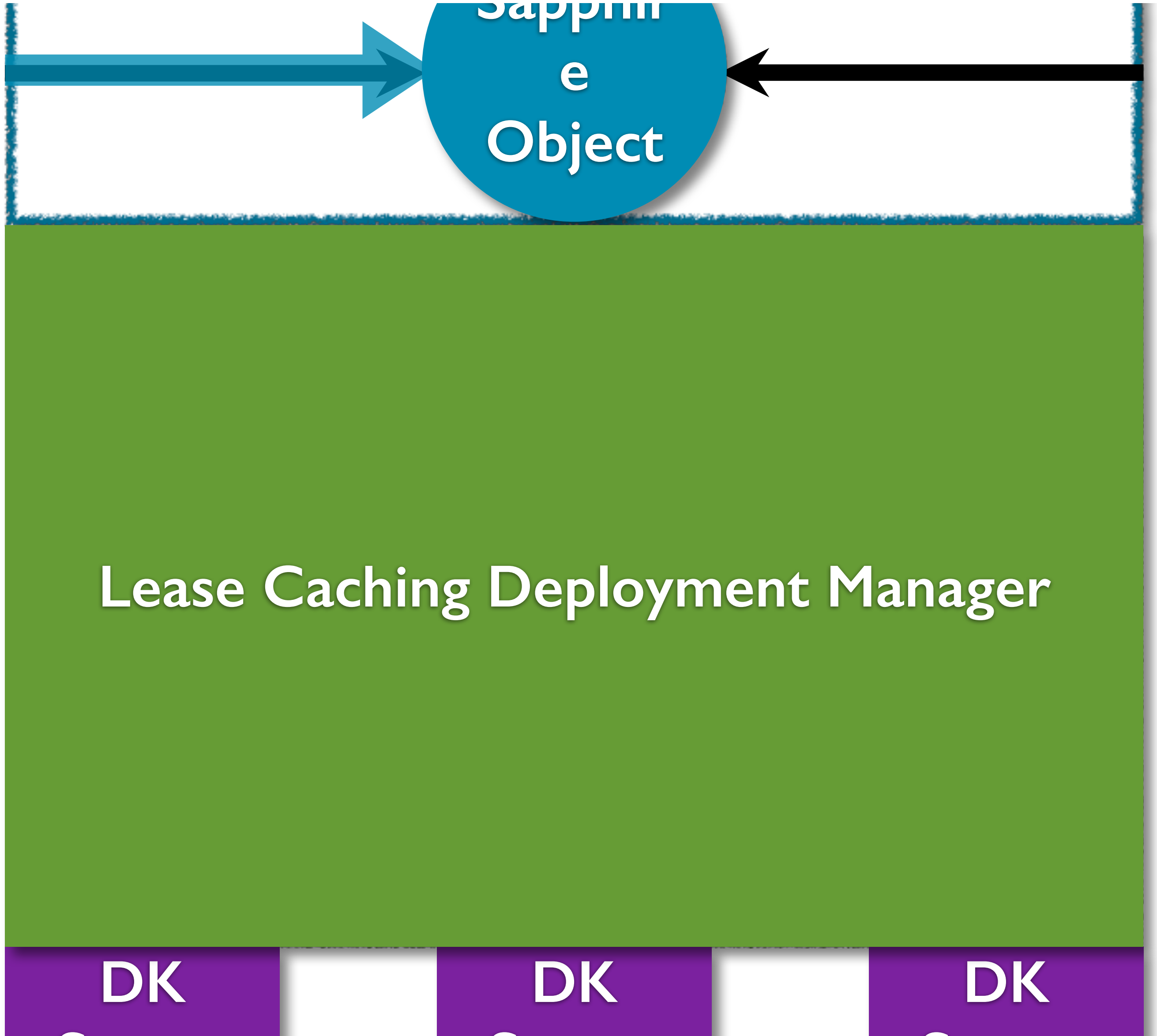
Deployment Manager Composition

- Implemented through chaining deployment managers (done automatically by the kernel, SRE just provides ordering via config)

Sapphire Deployment Manager API



Lease Caching Manager



Sapphir
e
Object

Lease Caching Deployment Manager

DK

DK

DK

1. Why are applications harder to build today?

Applications mix application logic and deployment.

2. What would make applications easier to build?

A customizable and extensible programming platform.

3. How does Sapphire make applications easier to build?

Sapphire has a flexible architecture that supports pluggable and extensible deployment managers.

Sapphire

Distributed programming platform for mobile/cloud applications.

The Goal

Separate deployment code from application logic.*

The Solution

A flexible and extensible distributed kernel/runtime system with pluggable and customizable *deployment managers*.*

* Keep the application
in control of deployment decisions.

Sapphire

- Eases the programming of mobile/cloud applications.
- Provides flexibility in choosing and changing deployment decisions.
- Gives programmers fine-grained control over performance trade-offs.

Work in progress ...

11 Sapphire applications built or ported so far.

Fully-featured Twitter-clone in **783 LoC**.

26 Sapphire Object Managers implemented.

Paxos state-machine replication in **129 LoC**.

What we have done...

11 Sapphire applications built or ported.

Fully-featured Twitter-clone in **783 LoC**.

26 Sapphire Object Managers implemented.

Paxos state-machine replication in **129 LoC**.

What you can do ...

Cache a cloud app on a mobile device in **1 LoC**.

Offload a mobile app to the cloud in **1 LoC**.

Change a client-server app to P2P in **1 LoC**.

Sapphire

A distributed programming environment for mobile/cloud applications that consists of:

An object programming model for applications

An extensible object management library

A distributed runtime system

5 Things To Do In 1 Line of Sapphire Code:

1. Make an object globally accessible by marking it as a Sapphire Object.
2. Cache an object on a device and keep it consistent.
3. Replicate an object and keep it consistent using Paxos.
4. Offload an object from a device to the cloud.
5. Deploy an object peer-to-peer across clients.

Contributions

- New distributed object model for the wide-area environment and heterogeneous compute platforms.
- Runtime library of common deployment strategies and distributed management tasks.
- Customizable and extensible distributed runtime system for mobile devices and cloud servers.

F.A.Q.

model of your choice]?

Unlike previous object systems, Sapphire’s object model is designed for the wide-area environment and heterogeneous compute platforms.

Runtime library of common deployment strategies and distributed management tasks.

What about performance trade-offs?

Programmers can both customize and

Experience and Evaluation

- 11 applications built and/or ported to Sapphire, including a Twitter clone in less than 800 LoC.

“I had little knowledge of distributed systems going into this project ... writing the application was surprisingly simple ... requiring only a shallow knowledge of distributed systems.”

- 26 SOMs, including code offloading and Paxos replication, each less than 180 LoC.

“Building runtime management in a SOM is easy if you have done event-based programming... you don’t have to worry about monitoring the state of things across the application ... with DVM support for distribution tasks like replication and placement most of the hard work is done for you.”

Goals

1. Create a uniform distributed programming platform.
2. Keep the programmer aware of performance costs.
3. Separate application logic from deployment and distribution logic.
4. Give the programmer control of performance trade-offs.

SOM Framework

Framework for building application-agnostic distributed runtime extensions that:

- Manage the distribution and runtime of one Sapphire Object via interposition on the Sapphire DVM.
- Extend the semantics or performance of the Sapphire DVM.
- Encompass the policy and mechanism of one distributed management task.