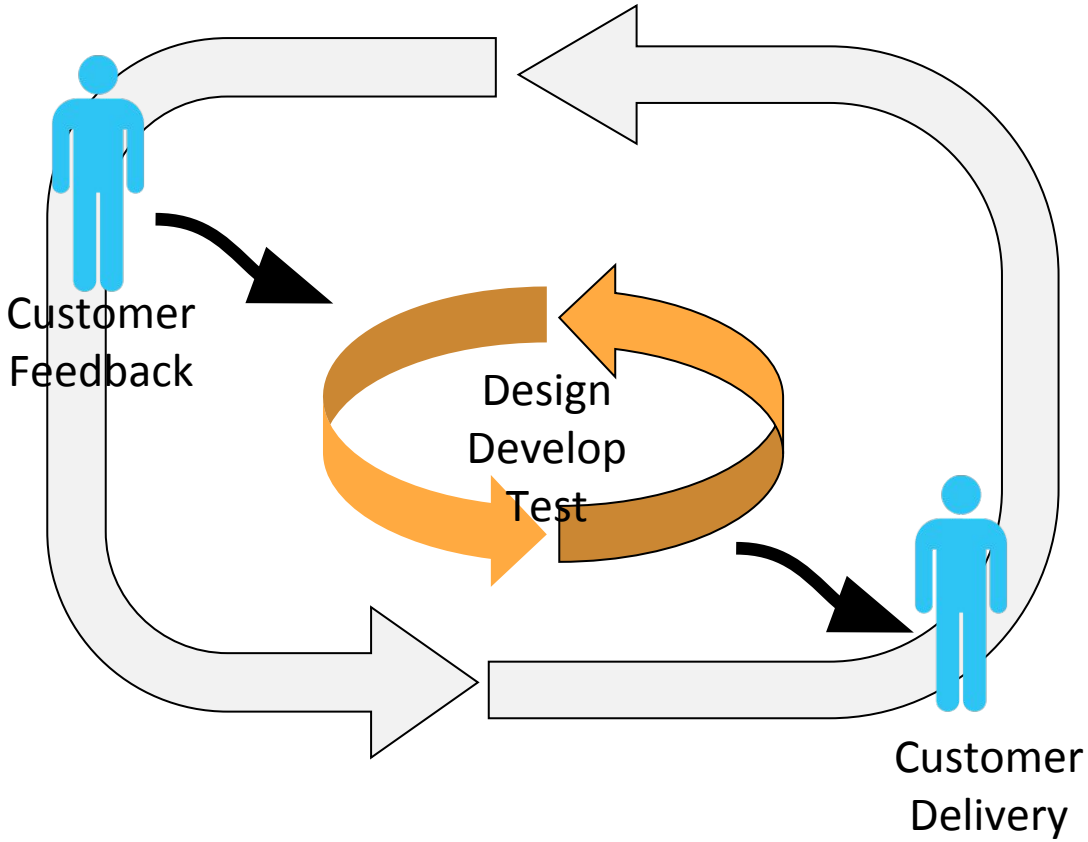
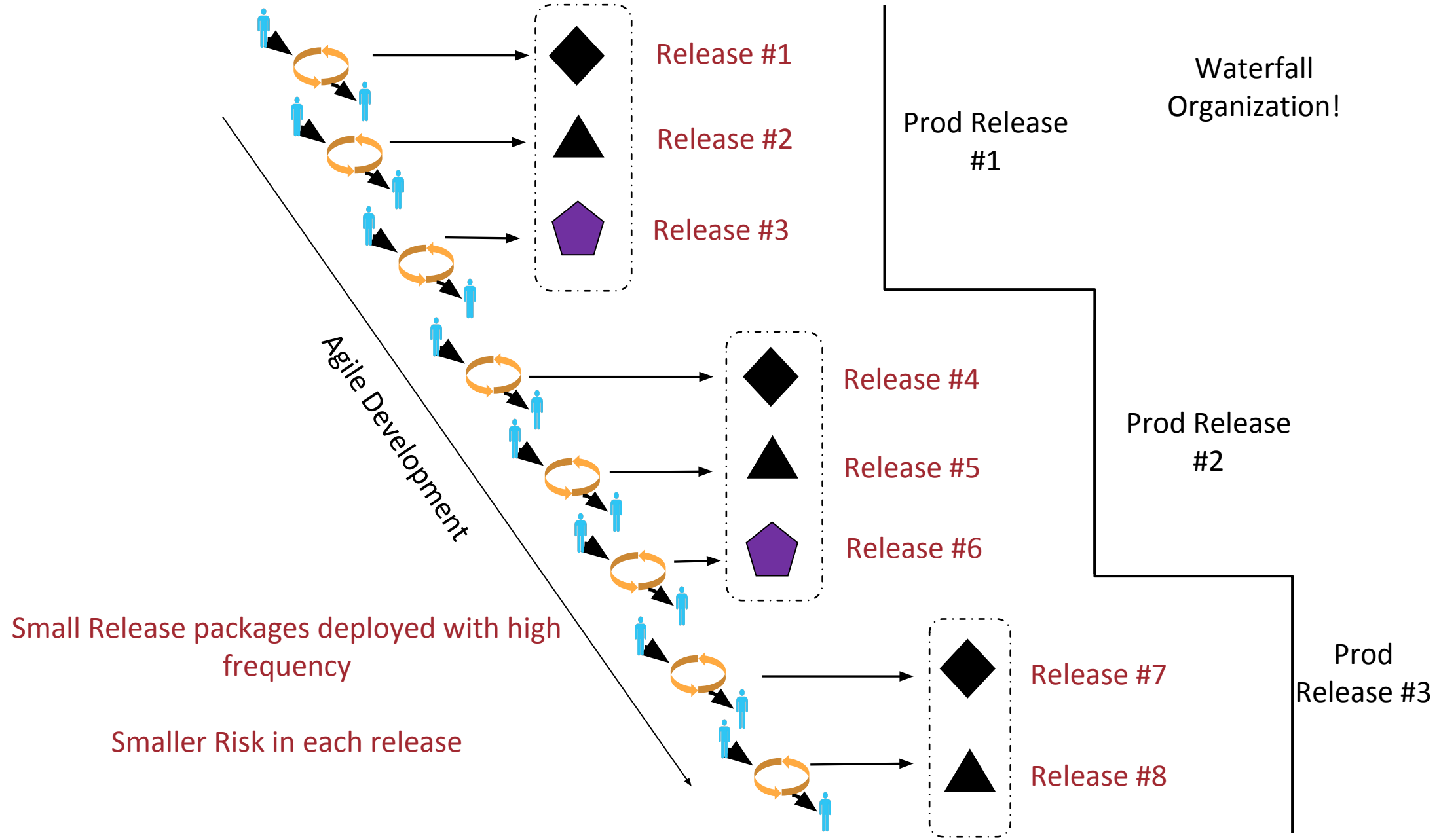
Microservices Architecture

Sanjay Dwivedi

Enable Agile Development & Continuous Delivery – Need for Speed



Enable Agile Development & Continuous Delivery – Need for Speed



Reduction of batch size is the recurring theme for speed

Irakli Nadareishvili et. al. State in their book is next step in lean software engineering

1. Agile – smaller batches in project management.
2. Lean – smaller batches in product management.
3. CI/CD – smaller batches in ops/deployment and QA.
4. Microservices – smaller batches of design/architecture!

"Slay the monolith": Microservices close the alignment loop!

Irakli Nadareishvili et. al.: Microservices Architecture: Aligning Principles, Practices, and Culture

Why Microservices?

Architecture Style that came about because Web architecture supporting eCommerce do not work in new Digital world!

They were pre-dominantly Monolithic Architectures

&

Deployed in company data centers

Why Microservices?

Microservices Architecture Style is first style developed:

POST Continuous Delivery

&

POST DevOps

&

POST Cloud Revolution

To Support Rapid Change in an Increasingly Digital World!

Microservices – A Distributed Software / Application Architecture

- Microservices Architecture Style is fundamentally a distributed software / application architecture
- 8 Fallacies of Distributed Computing Apply
- Foundational Building blocks of distributed system are required
  - Networks
  - Network Protocols
  - Application Protocols
  - Message Serialization/DeSerialization
  - IPC Mechanism: RPC or Messaging Mechanism
  - Service Registry & Discovery
  - Distributed Consensus Protocol to reason and arrive at a consensus on state of distributed system
  - Failure is guaranteed, architecture & design must model failure and bake in resiliency

Microservices: Definition

- "Fine grained SOA architecture done the UNIX way" – James Lewis, Thoughtworks
- Follow the Unix Philosophy
  - "Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface" - Doug McMcIlroy, Head of the Bell Labs CSRC & Inventor of the Unix pipe
  - The Unix philosophy emphasizes building short, simple, clear, modular, and extensible code that can be easily maintained and repurposed by developers other than its creators. The Unix philosophy favors composability as opposed to monolithic design.
  - **Single Responsibility Principle!**
  - **Prefers Simplicity Over Complexity!**

Microservices: Definition

The microservice architectural style is an approach to developing a single application

- As a suite of small services
- Each running in its own process
- Communicating with lightweight mechanisms, often an HTTP resource API
- These services are
  - built around business capabilities
  - independently deployable by fully automated deployment machinery
- Services may be written in different programming languages and use different data storage technologies
- There is a bare minimum of centralized management of these services

Martin Fowler: http://martinfowler.com/articles/microservices.html

Microservices: Definition

If every service has to be updated in concert, it's not loosely coupled!

Loosely coupled service oriented architecture with bounded contexts

If you have to know about surrounding services you don't have a bounded context.

Adrian Cockcroft: http://www.slideshare.net/adriancockcroft/dockercon-state-of-the-art-in-microservices

Microservices Architecture Characteristics

Characteristics of Microservices Architecture Style	
Componentization via Services A component is a unit of software that is independently replaceable and upgradeable Main reason for using services as components (rather than libraries) is that services are independently deployable	Decentralized Data Management Bounded Context Master & Metadata Management becomes important
Organized around Business Capabilities Conway's Law of system design & communication	Infrastructure Automation - DevOps Culture
Products not Projects A team should own a product over its full lifetime Amazon's notion of "you build, you run it"	Design for Failure - Highly distributed in nature and failure is expected, so design for failure
Smart endpoints and dump pipes Anti- ESB & SOA where pipes are complex, and intelligent	Evolutionary Design
Decentralized Governance Independent microservice capability evolution with forward compatible service contracts	Stateless & Immutable

- Microservice Architectures are about both Technology & Business – Requires Cultural Change

Martin Fowler: http://martinfowler.com/articles/microservices.html

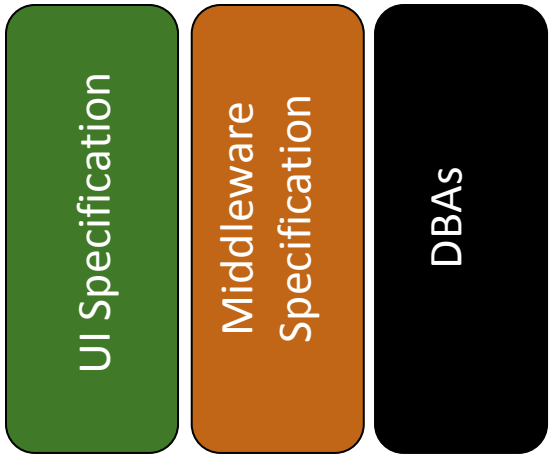
Benefits of Microservices Architecture Style

- Simple (but Hard)
- Modularity based on Component Services
  - Not libraries
  - Loosely coupled
- Enables Scaling of Development
  - Independent teams deliver features and services can evolve independently
- Independent Deployments - Change cycles de-coupled / Enable Frequent Deploys
  - Smaller units to deploy!
- Efficient Scaling
  - Scaling requires scaling services that require greater resource instead of entire app
- ~~Individual components less intimidating to new developers~~
  - Single responsibility principle! Easier to grasp!
- Eliminates long-term commitments to technical stack
  - Microservices can be written in any language, use any database and run on any platform
  - **No long term bets on single technology!**

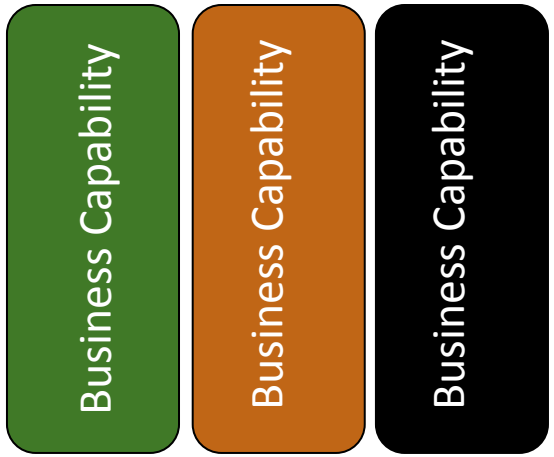
Architecture & Design Choices that inhibit these are deal breakers

Organized Around Business Capabilities

Siloed
Functional Teams
organized by technology
layer

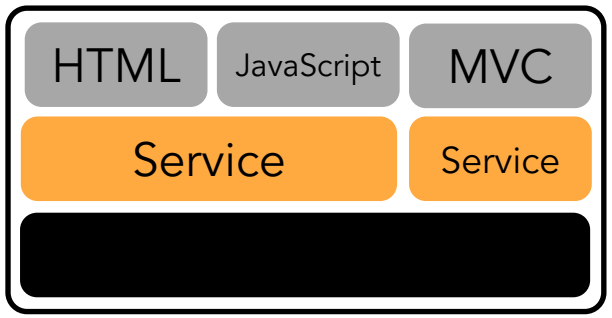


Cross-
Functional Teams organized
around business capability



Full Stack Engineering
Teams

Siloed
Application
Architectures

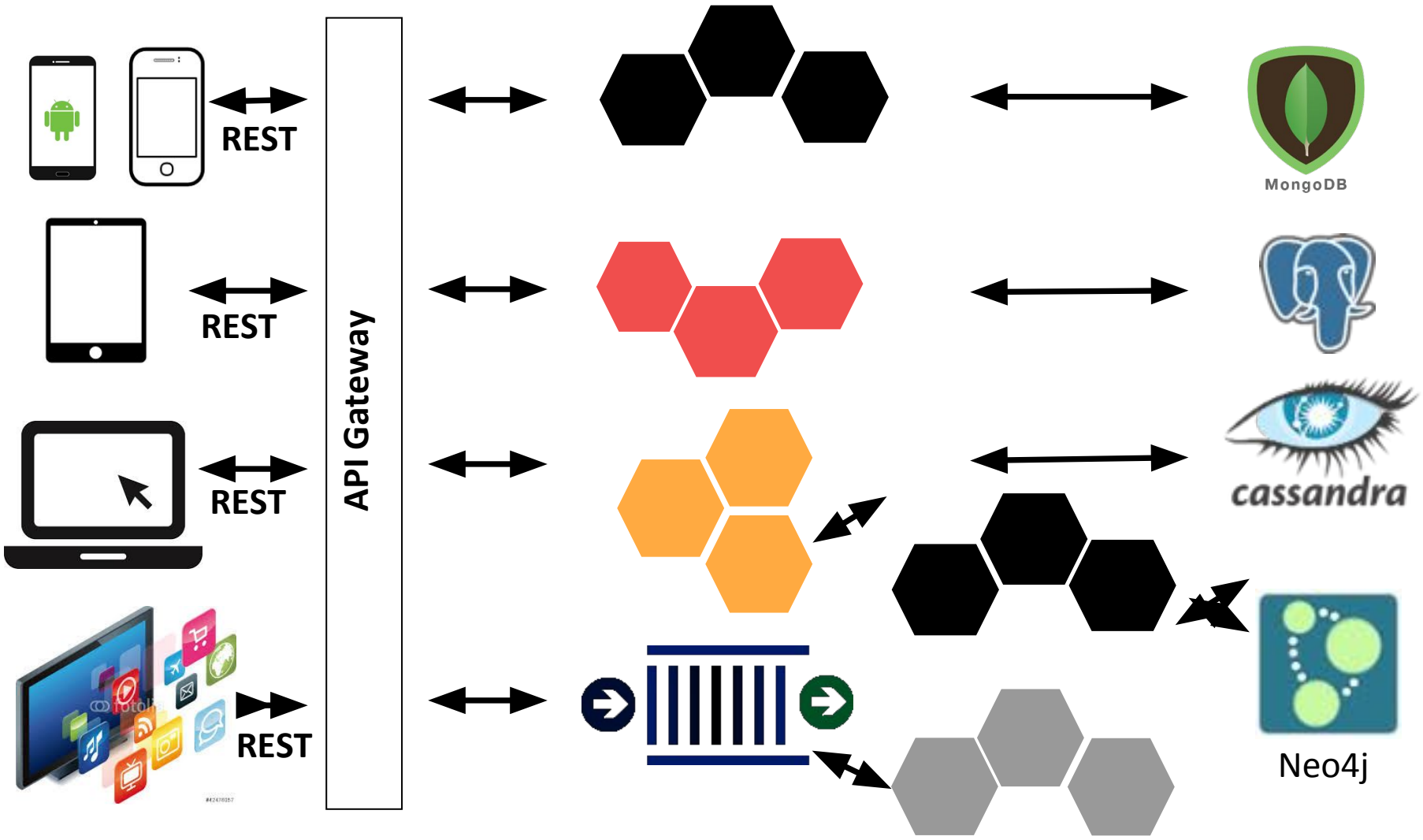


Monolithic
Application

Microservice
Architectures

Martin Fowler: http://martinfowler.com/articles/microservices.html#OrganizedAroundBusinessCapabilities

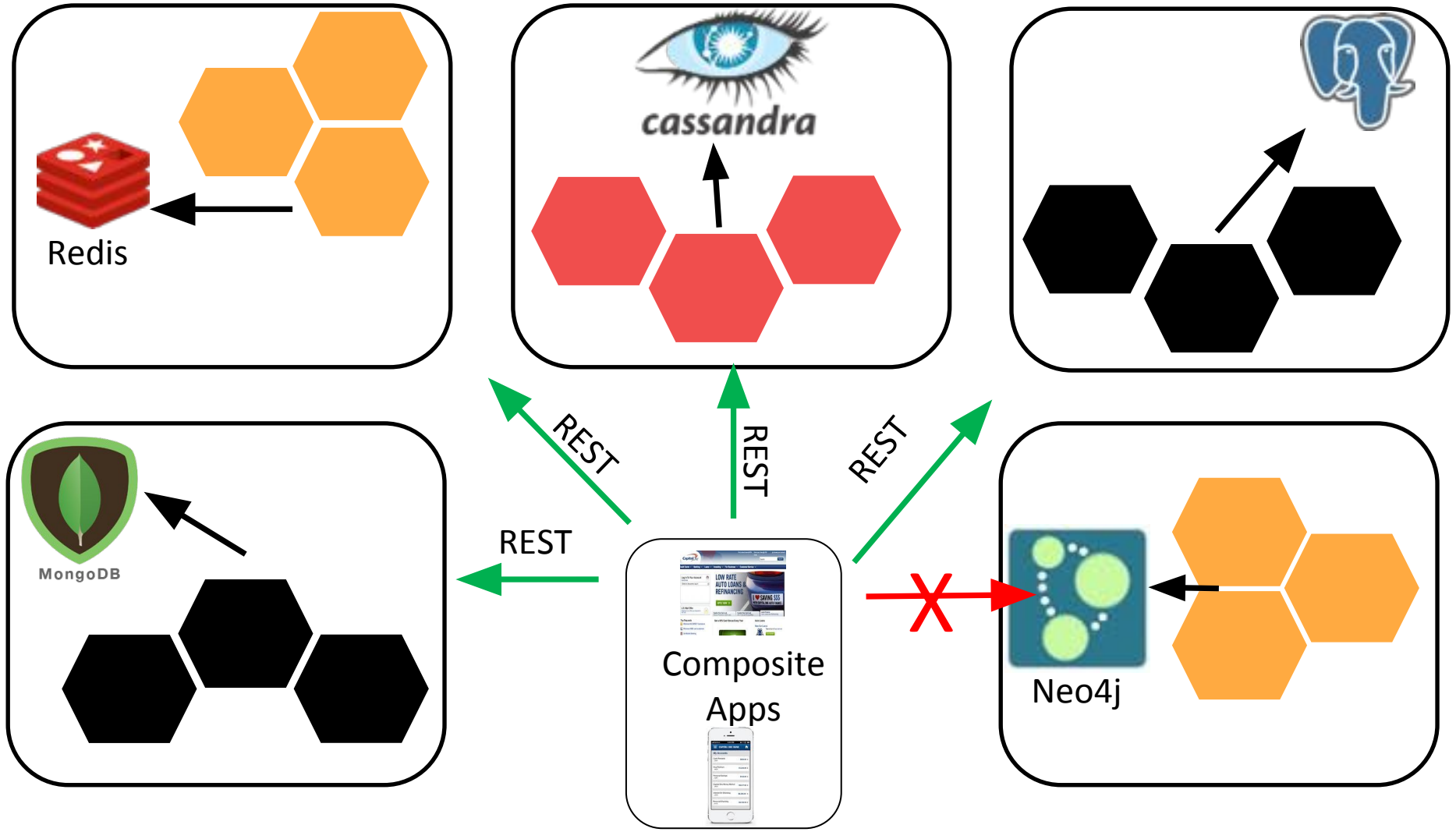
Diversity of Clients



Microservices allow developing new customer experiences rapidly

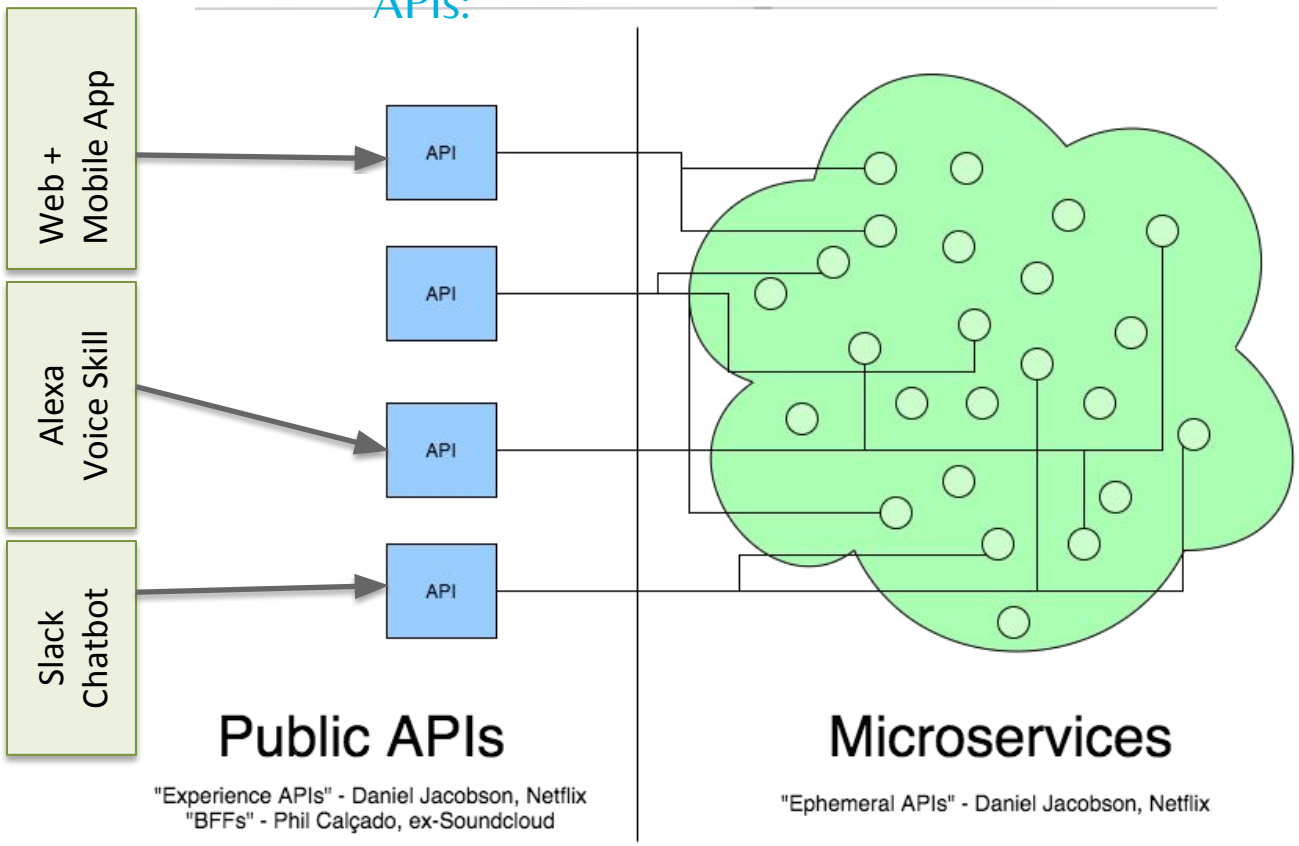
Decentralized Data Management & Polygot Persistence

- Each service manages its own data
- Uses persistence technology that is most suitable for the service
- Microservices do not share database
- Allows for de-coupling, independent evolution and deployments



Microservices != APIs

Microservices are not your public APIs:



- **Microservice architecture is the *implementation of your system*, it is not public API Interface of your system that external or most internal client should depend directly on – Irakli**

Microservices Architecture Tradeoffs

- Microservices architecture style bring costs and benefits
- Many teams have found MSA to be productivity sapping burden!
- Choose wisely!

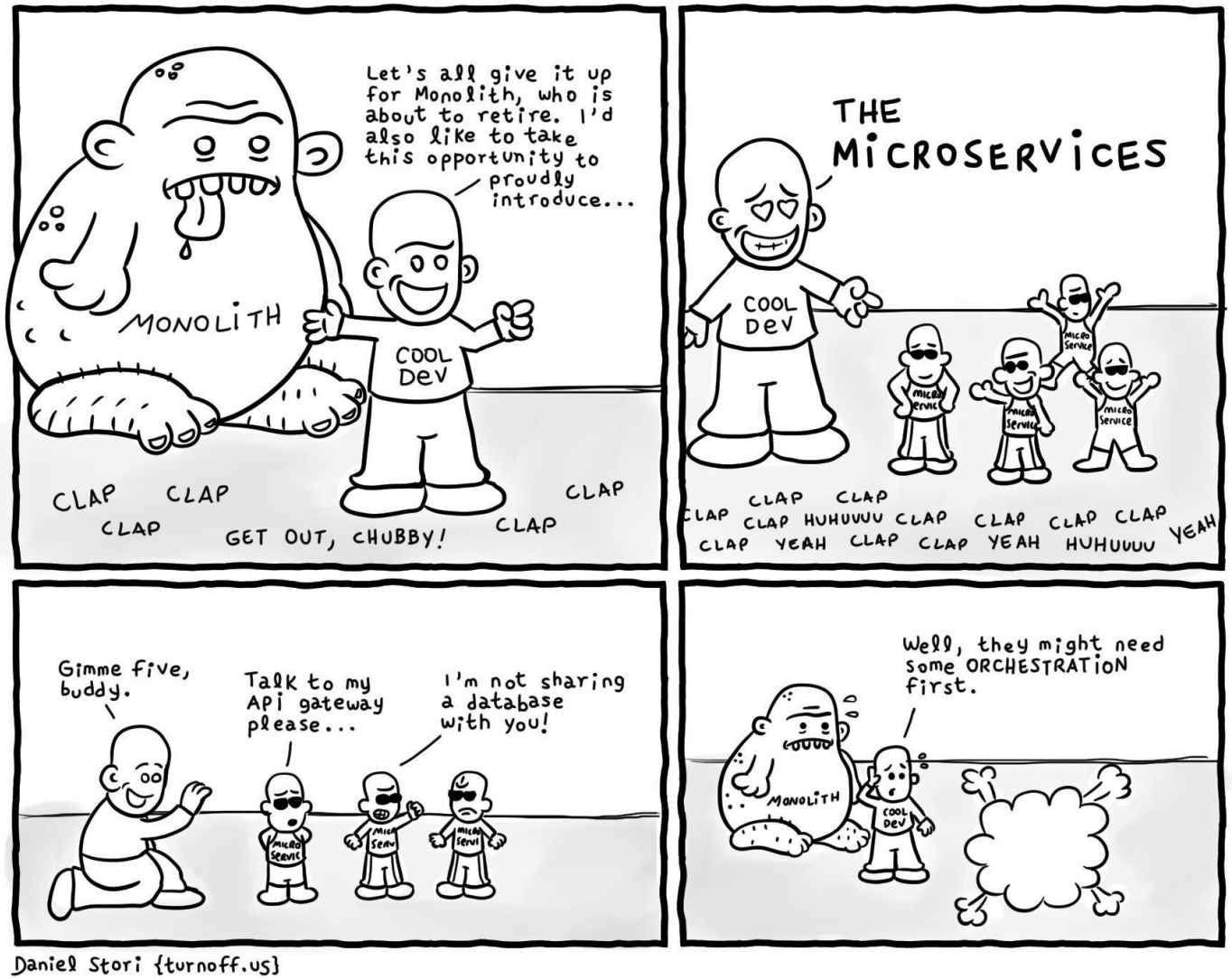
Microservices provide benefits…	…But come with costs
Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.	Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
Independent Deployments: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.	Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.	Operational Diversity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

Martin Fowler: http://martinfowler.com/articles/microservice-trade-offs.html

Microservices – Not a Free Lunch

- Significant Operations Overhead
- Substantial DevOps Skills Required
- Distributed System Complexity
  - 8 Fallacies of Distributed Computing are very much relevant!
- App composition using microservices requires discipline from developers
  - **Failure is Expected: Service may not be available! Plan for it**
  - Avoid cascading failures! (hint: circuit breakers!)
- Implicit Interfaces & Public API backward compatibility!
  - Plan for service versioning
- Duplication of Effort
- Duplication of Data
- Testability Challenges

Need for Orchestration



Emergent Challenges with Microservices Architecture

- (Externalized) Configuration Management
- Communication Protocols (RPC, Messaging, Domain-Specific)
- Service Registry & Discovery
- Routing & Load Balancing
- Resiliency (Circuit Breakers!)
- Monitoring & Metrics

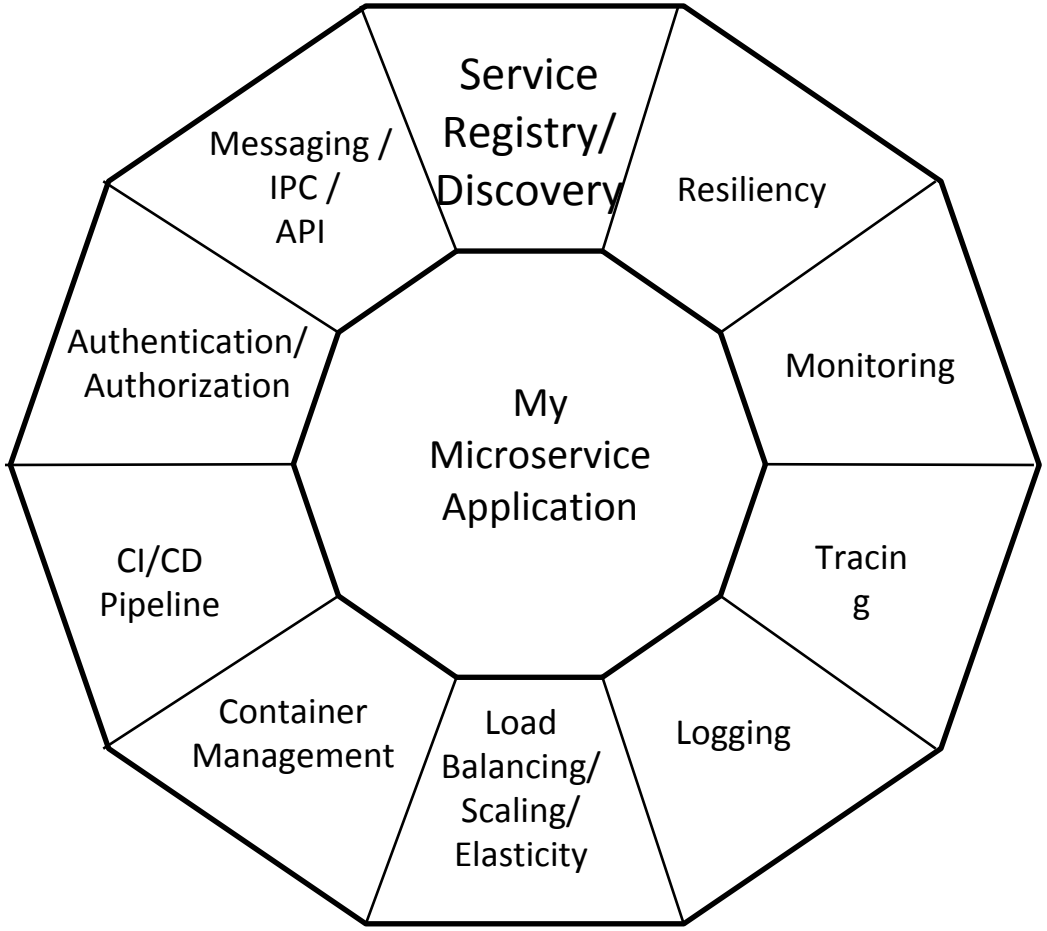
- Environment Provisioning
- On-Demand/Automatic Scaling
- Container Management & Resource Scheduling
- Monitoring/ Failover/Resiliency/Auto Restarts
- Log Aggregation

Chris Richardson: http://microservices.io/patterns/index.html

Platform Services Needed for Microservices

Application / Service is unit of:

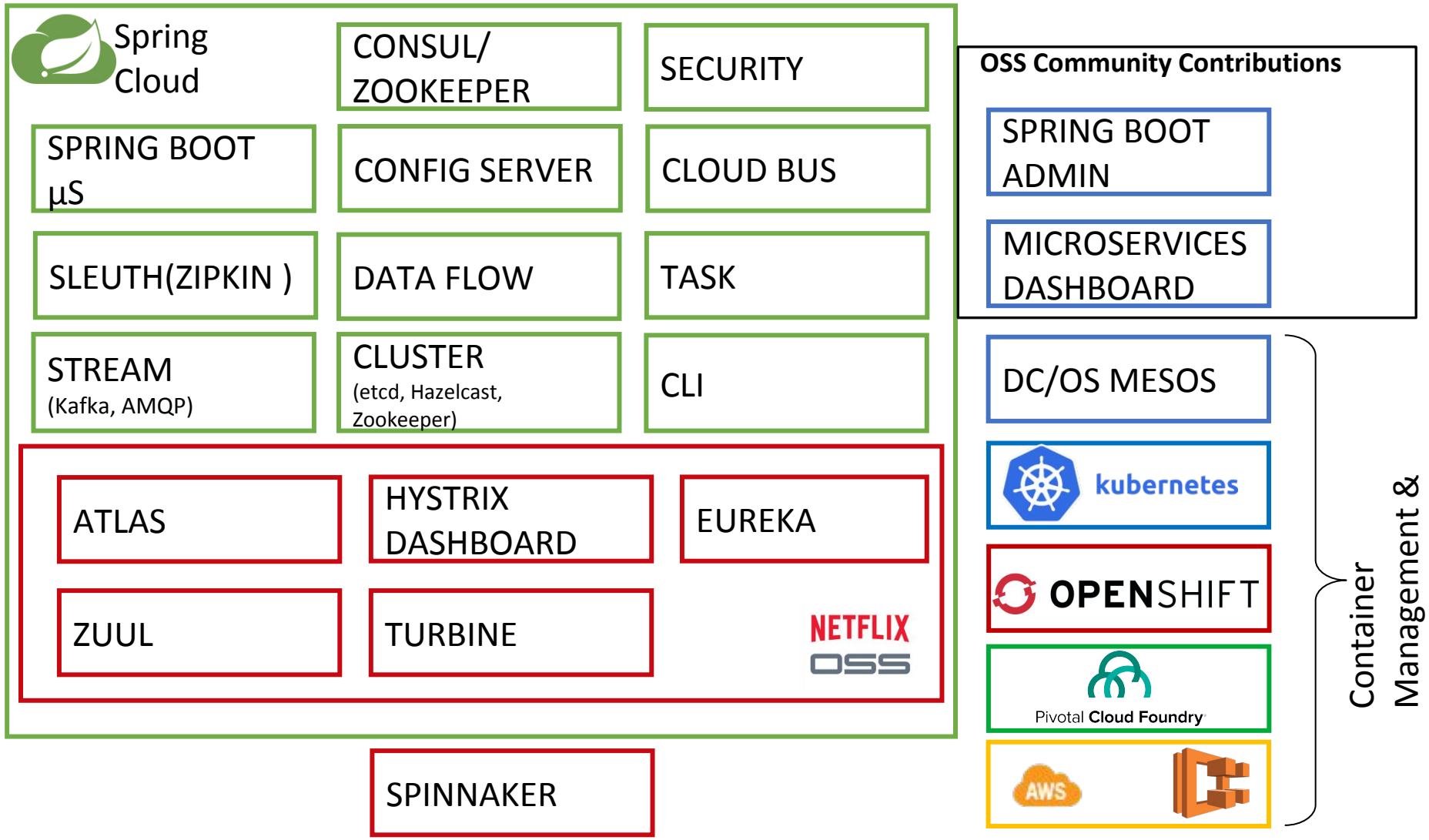
- Resiliency
- Scaling
- Elasticity
- Load Balancing
- Discovery



Requires:

- Container Management & Resource Scheduling
- Log Aggregation
- Monitoring
- Distributed Tracing
- Authentication/Authorization
- API / IPC / Messaging
- CI/CD Pipeline

Microservices Toolkit – Spring Boot with Spring Cloud/Netflix OSS Components



Spring Cloud with Netflix OSS Components, built on Spring Boot offers a comprehensive Microservices toolkit for JVM with:

- Externalized Config Management
- Service Registry/Discovery via Eureka/Consul
- Messaging: HTTP, gRPC, Kafka, AMQP, protobuf, thrift etc.
- Resiliency/Circuit Breaker
- Server side Load Balancing & Proxy
- Monitoring/Metric via Spring Actuator
- Zipkin based tracing
- Container Management, Resource Scheduling, Self-Healing & Resiliency requires using other solutions
- 3rd party tools for CI/CD pipeline such as Netflix Spinnaker
- Log Aggregation via PaaS or ELK/Splunk etc.

http://projects.spring.io/spring-cloud/

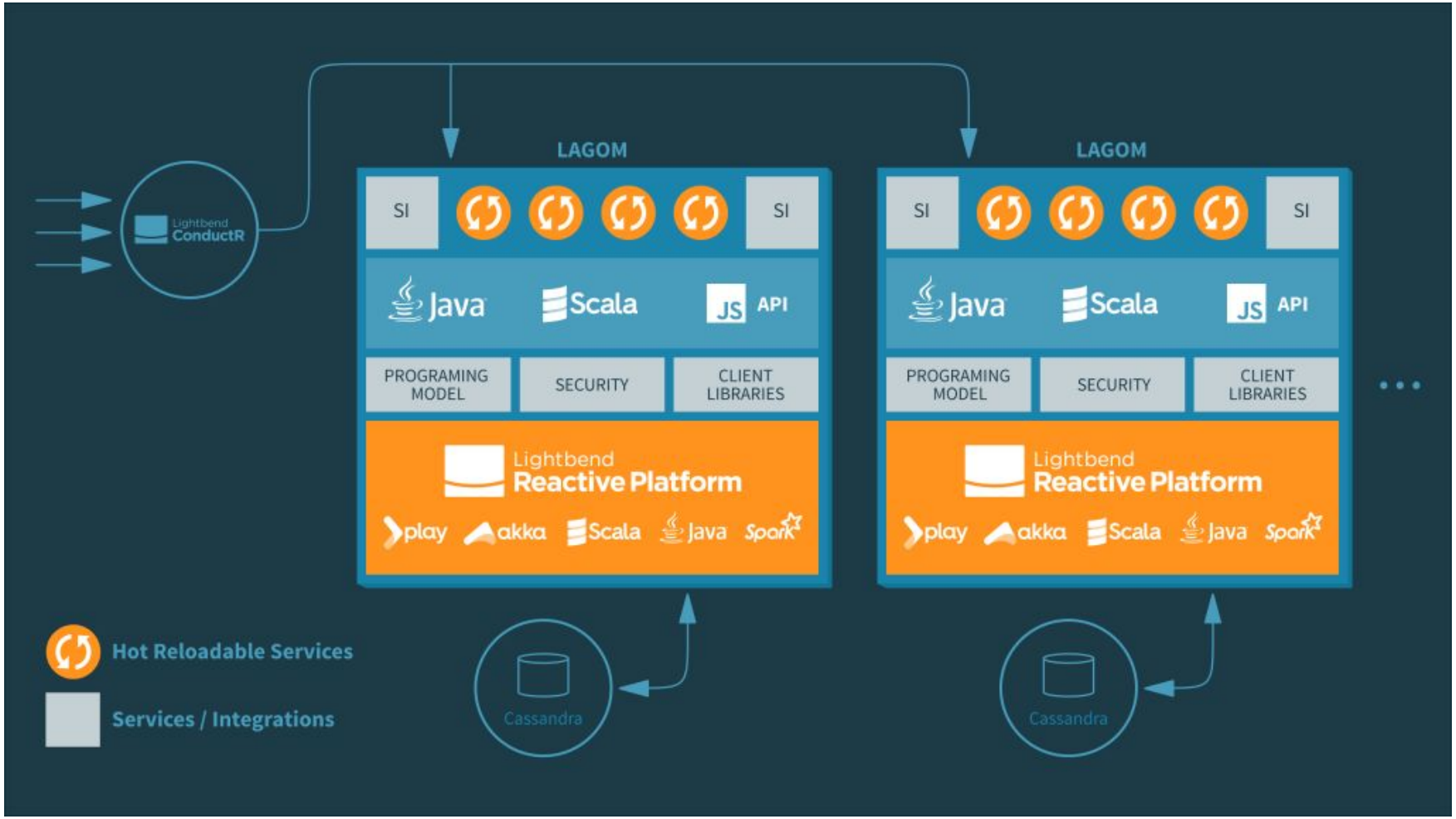
Microservices Platform & Toolkit - Lagom

A Reactive microservices framework for JVM with:

- Support for CQRS, Event Sourcing
- Resiliency, Scalability, Distributed clustering, async messaging from Akka
- Service Registry/Discovery via 3rd party solutions like Consul, Eureka, ZooKeeper
- Externalized Config via Persistence Services (Cassandra)

ConductR:

- For infrastructure management, container orchestration, deployment, scaling & monitoring
- Support for public & private clouds such as AWS, Azure, Google Cloud, DC/OS, Kubernetes, Docker, Cloud Foundry

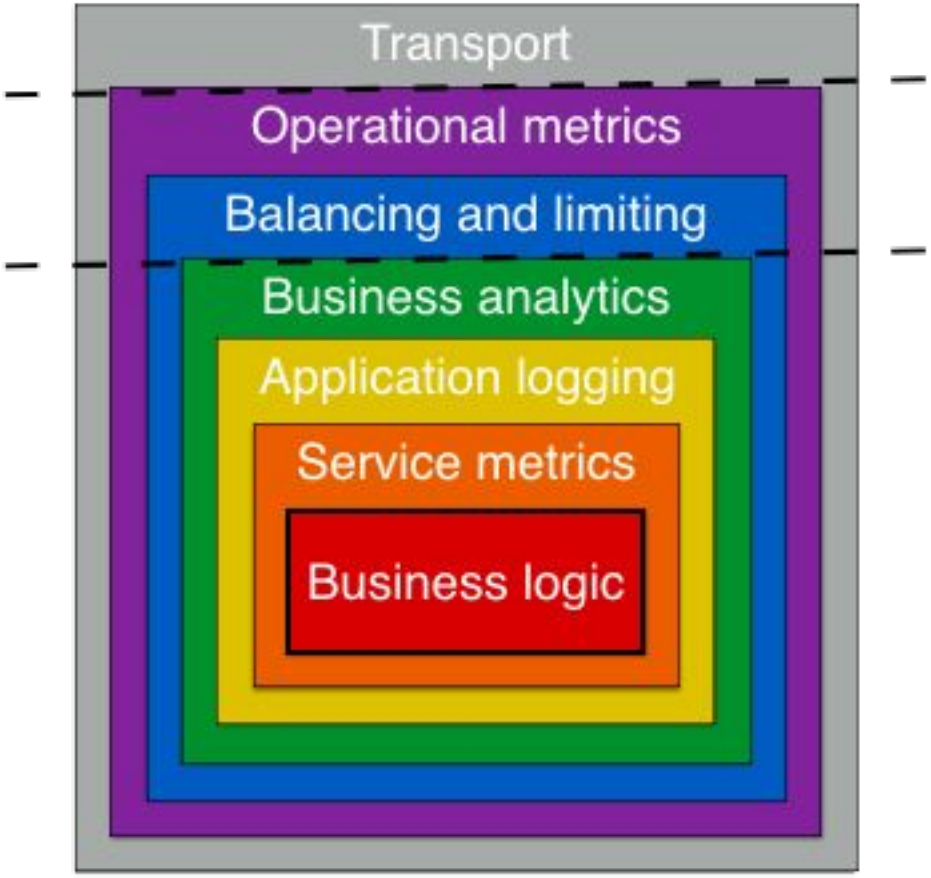


https://www.lagomframework.com/

Microservice Toolkit: Go-kit

MSA toolkit for golang with:

- Support for HTTP, TLS, gRPC, thrift, net/rpc, circuit breakers
- Service Registry via Consul, etcd, DNS SRV records
- Load balancing, Rate Limiting
- Logging
- Health, Metrics & Monitoring via Influxdb, prometheus, Graphite, statsd etc.
- Support to run go-kit on Kubernetes, AWS ECS



https://gokit.io/

You will need a PaaS!

Build your own using Open Source Components:

Using Cloud Native Computing Foundation (CNCF) Sponsored Projects (Kubernetes, Docker, Istio, etc.)

Build using Netflix OSS Components (Spinnaker + Docker or ECS or Mesos and so on)

Apache Mesos/Marathon/Chronos (or packaged distro from Mesosphere - DC/OS)

…

OR

Use one of several wonderful open source PaaS:

Cloud Foundry

OpenShift

Deis (Acquired by Microsoft)

…

OR

Commercial Products such as:

Apcera Platform, Pivotal Cloud Foundry, RedHat OpenShift Enterprise, CoreOS Tectonic, …