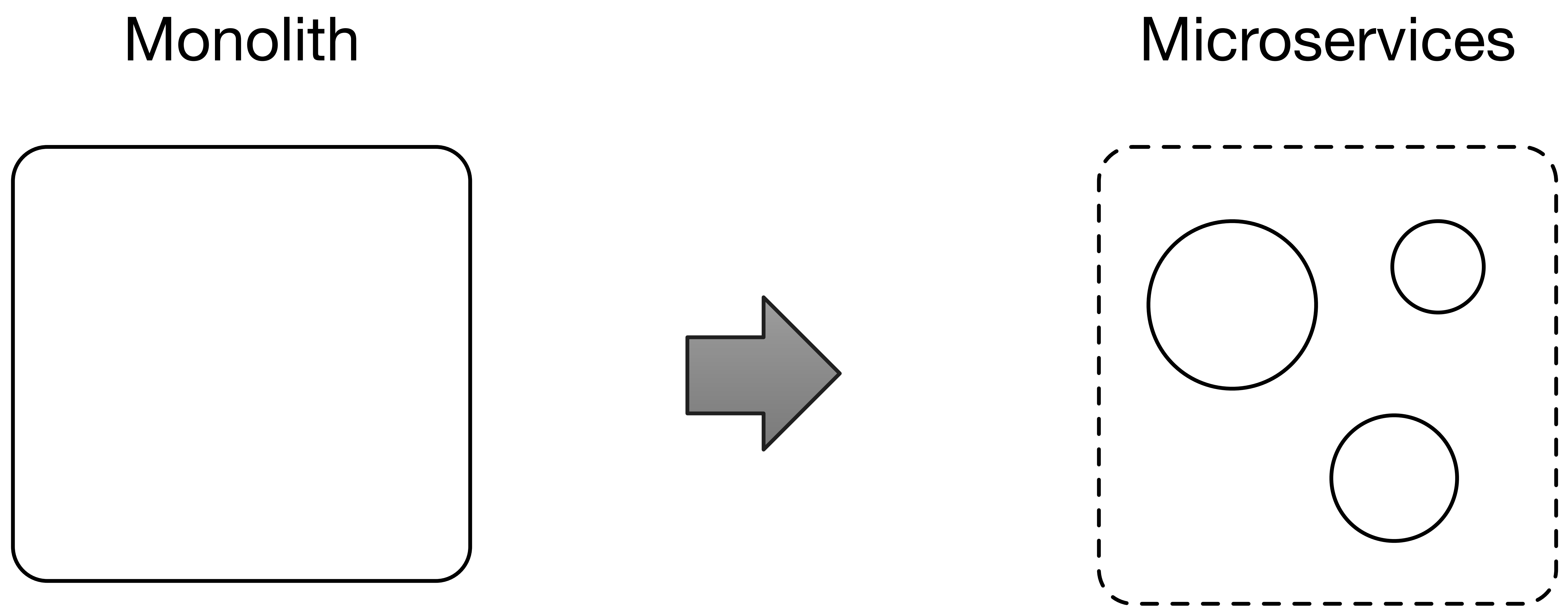
RED HAT DEVELOPERS

Rightsize your Services with Wildfly Swarm

Heiko Braun
Principal Software Engineer
May 2017

Monoliths and microservices

Either/or?



“There is always a well-known solution to every human problem - neat, plausible, and *wrong*”

– Mencken, H. L. *Prejudices: Second Series* (1920)

“A map *is not* the territory it represents, but, if correct, it has a *similar structure* to the territory, which <u>accounts for its usefulness</u>.”

– Alfred Korzybski, *Science and Sanity* (1933)

What’s in a name?

“Software Architecture: the fundamental organization of a system embodied in its **components**, their **relationships** to each other and to the environment and the **principles** guiding its design and evolution.”

–IEEE in their standard IEEE 1471
(which was later adopted by ISO/IEC 42010).

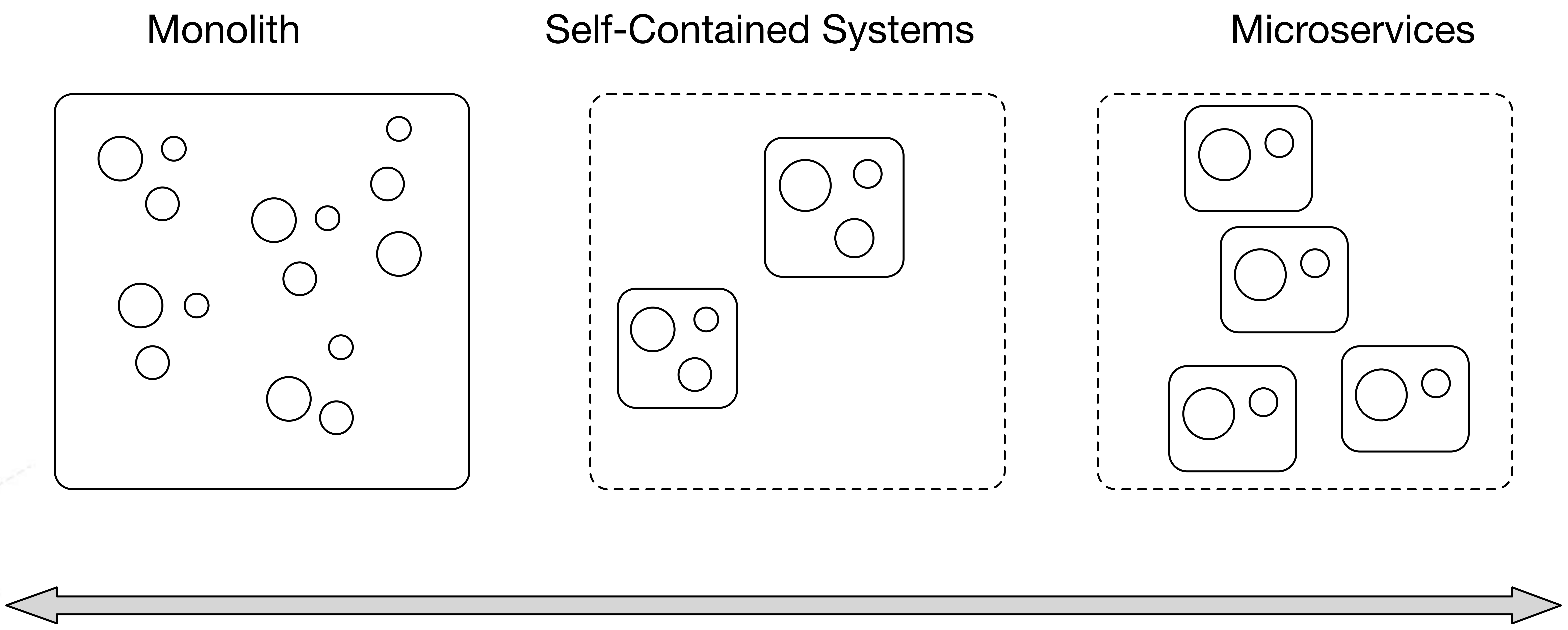
Architectural properties

	Monolithic architecture	Microservice architecture
Components	Few	Many
Relationships	Space: co-located Time: change together	Space: distributed Time: change independently
Principles	Uniformity	Diversity

(This is a non exhaustive list. Add your own ideas)

RED HAT DEVELOPERS

Continuum of architectural choices



For self-contained systems see http://scs-architecture.org/

Coming from Java EE, where does it put you?

Perspectives on Java EE

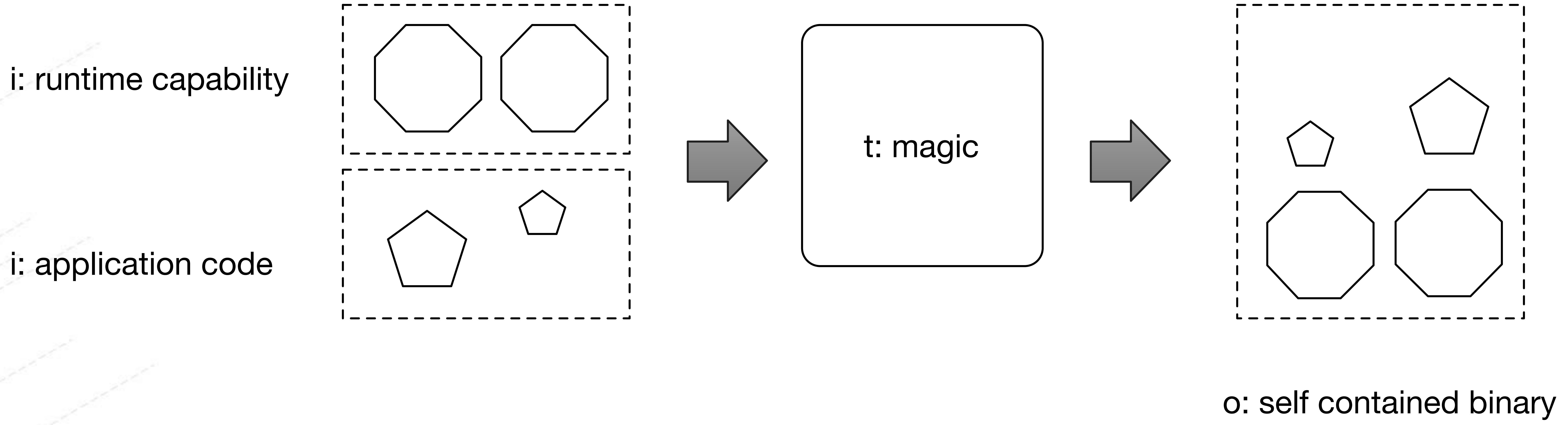
- It’s different things to different people:
  - A collection of (useful) API’s
  - Technical capabilities of a system
  - A love/hate relationship (of the past)
  - (Existing) knowledge and expertise

Meet Wildfly Swarm

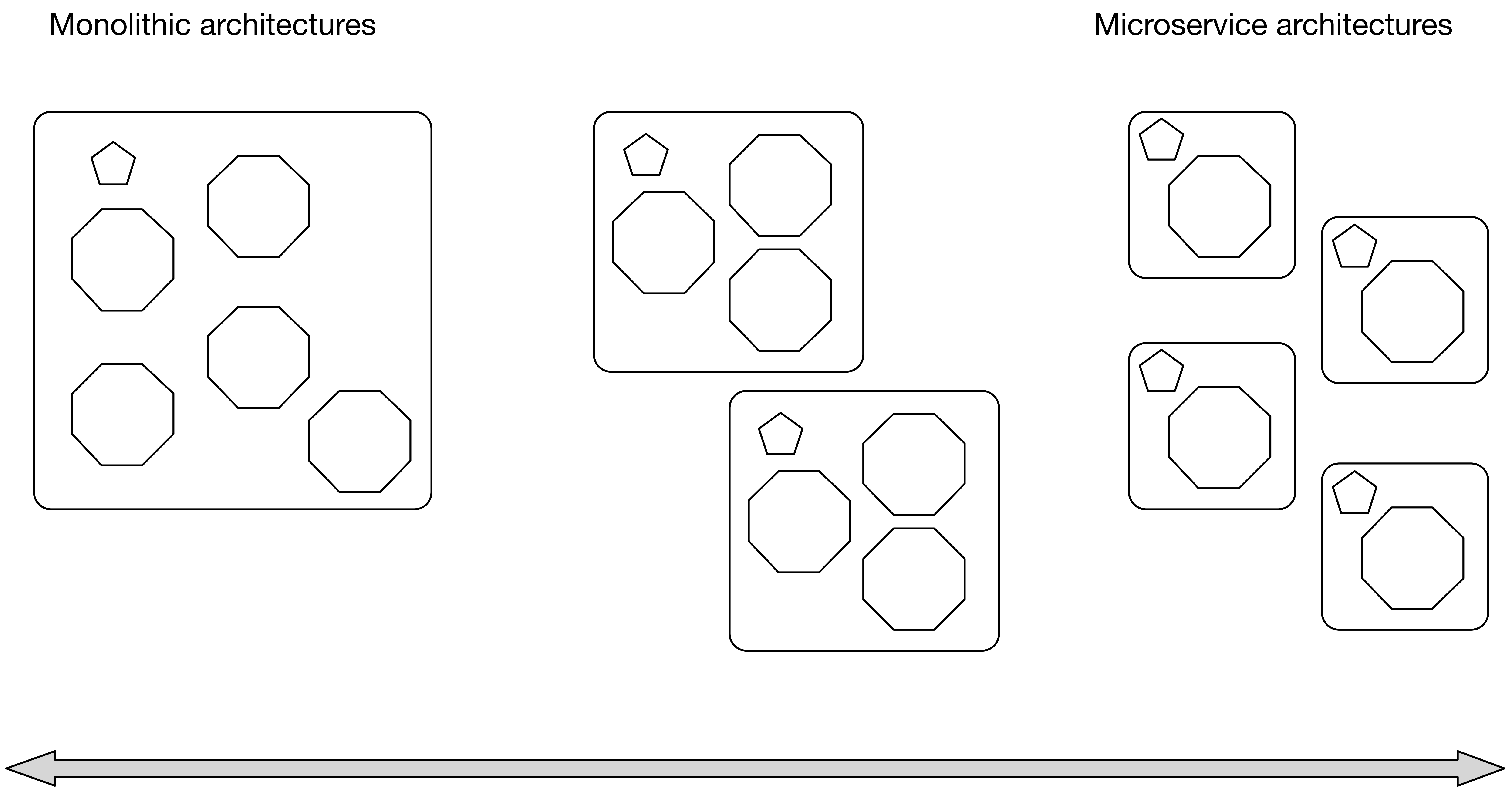
Wildfly Swarm

- OSS Project sponsored by Red Hat
  - http://wildfly-swarm.io
- Sidekick of Wildfly Application Server
- Small, but ambitious and friendly community
- Part of a bigger system of interrelated projects under the JBoss / Red Hat umbrella

The basic idea



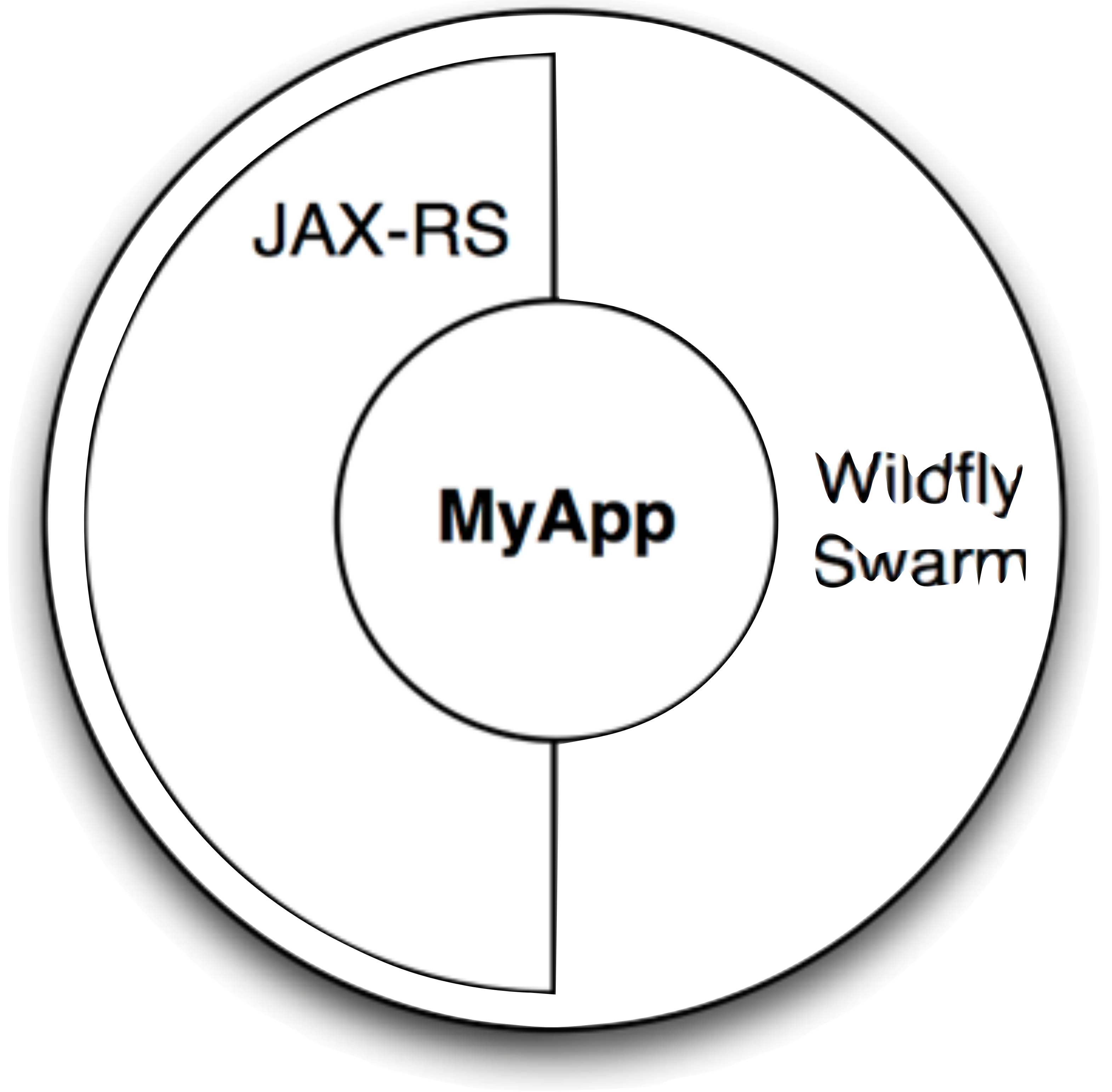
“Right Size” the runtime



Self-contained (Uber) jar

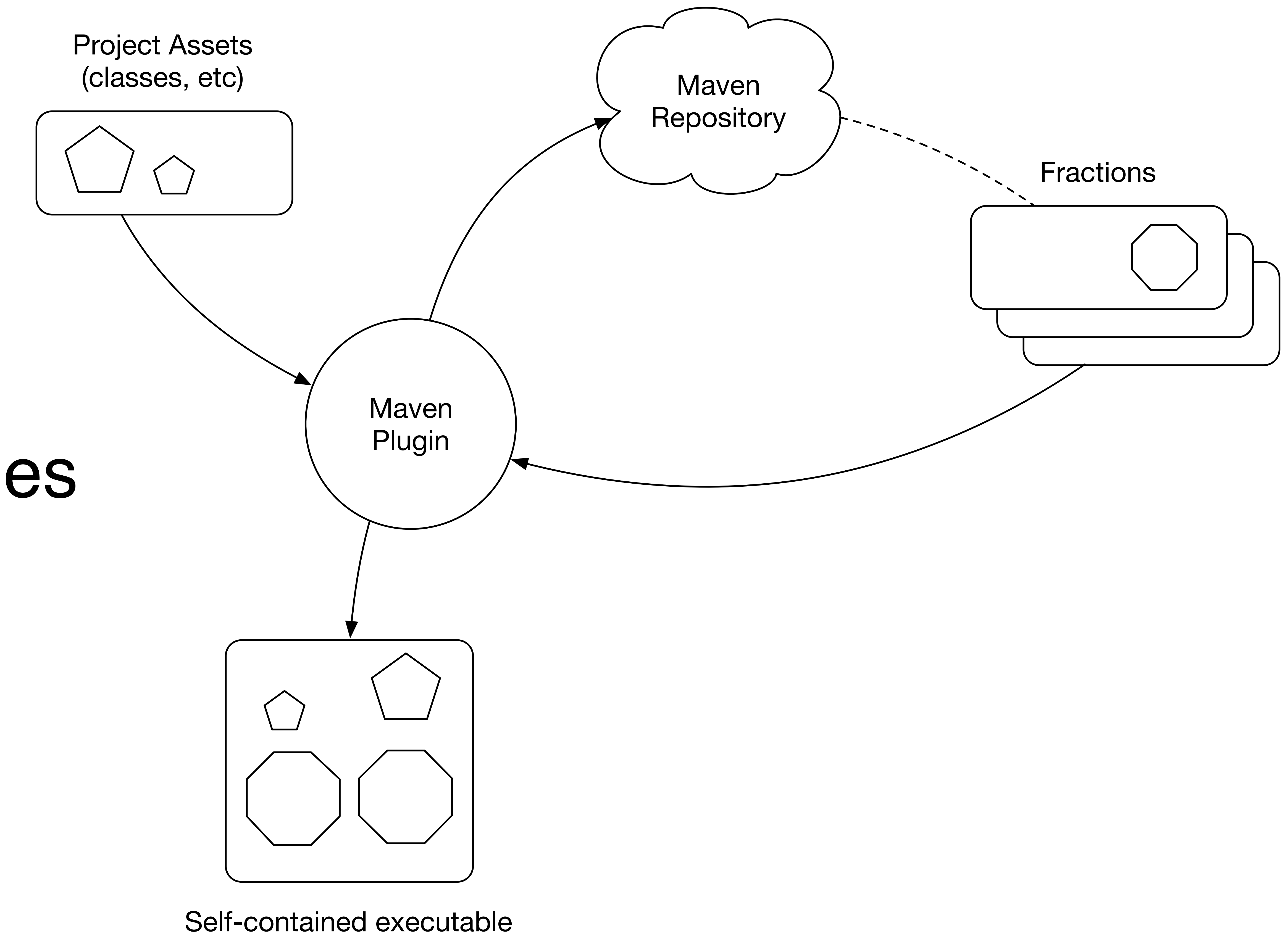
- bundles your application
- the *Fractions* to support it
- an internal maven repo with the dependencies
- bootstrap code

- There is also the notion of a *Hollow* launch-pad type of jar.



Concept of a Fraction

- A tangible unit, embodied in a maven artefact
  - To support the compositional aspect in Swarm
- Provides the “runtime” capabilities to your application
- Means to configure the system
  - With reasonable defaults



Fraction use cases

- Fractions support explicit and implicit configuration
  - In many cases you won’t need to configure anything
- Fractions can be detected or explicitly declared
  - The most simple case is a <war> project, with just the maven plugin
- All of EE is supported in Swarm:
  - JPA, JAX-RS, EJB, JMS, …

Using WildFly Swarm

Integrated with Maven

pom.xml:
```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.wildfly.swarm</groupId>
      <artifactId>bom-all</artifactId>
      <version>
        ${version.wildfly.swarm}
      </version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

pom.xml:
```
<plugin>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>
     wildfly-swarm-plugin
  </artifactId>
  <version>
    ${version.wildfly.swarm}
  </version>
  <executions>
    <execution>
      <goals>
        <goal>package</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Build and Run a service

Build Your Project

```
mvn package
```

Start The Service

```
java -jar *-swarm.jar
```

Success!

```
$ curl http://localhost:8080
```

Demo: Turning a JEE Application into a Wildfly Swarm Service

Moving further to the right …
towards service oriented systems

(very likely running in the cloud)

Shifting complexities

When moving to service oriented systems

- You separate out the components – the complexity moves elsewhere:
  - In monolithic architectures you have to coordinate the updates to the software prior to releasing it
  - In microservices architectures you have to manage a multitude of distributed services being updated in parallel
- It’s not about “right” or “wrong”, it’s about “better” or “worse” (according to your circumstances)

Organisational Competencies

… to benefit from Microservice Architectures

- M. Fowler [1] identifies a set of baseline competencies:
  - Rapid provisioning
  - Rapid application deployment
  - Monitoring
- “These competencies *should be universally present* across software organisations”

[1] http://martinfowler.com/bliki/MicroservicePrerequisites.html

Extended requirements

On tools, infrastructure & way of working

- Scope of system components goes beyond your application runtime:
  - i.e. Cloud infrastructure, CI/CD, etc
- Tools like Swarm can support new architectures, by:
  - Extending the functional scope
  - Providing new integrations (libraries, 3rd party systems ,etc)
  - Extending the programming models
  - Supporting new operational requirements
- **This however means going beyond Java EE …**

Wildfly Swarm in cloud-native systems

RED HAT DEVELOPERS

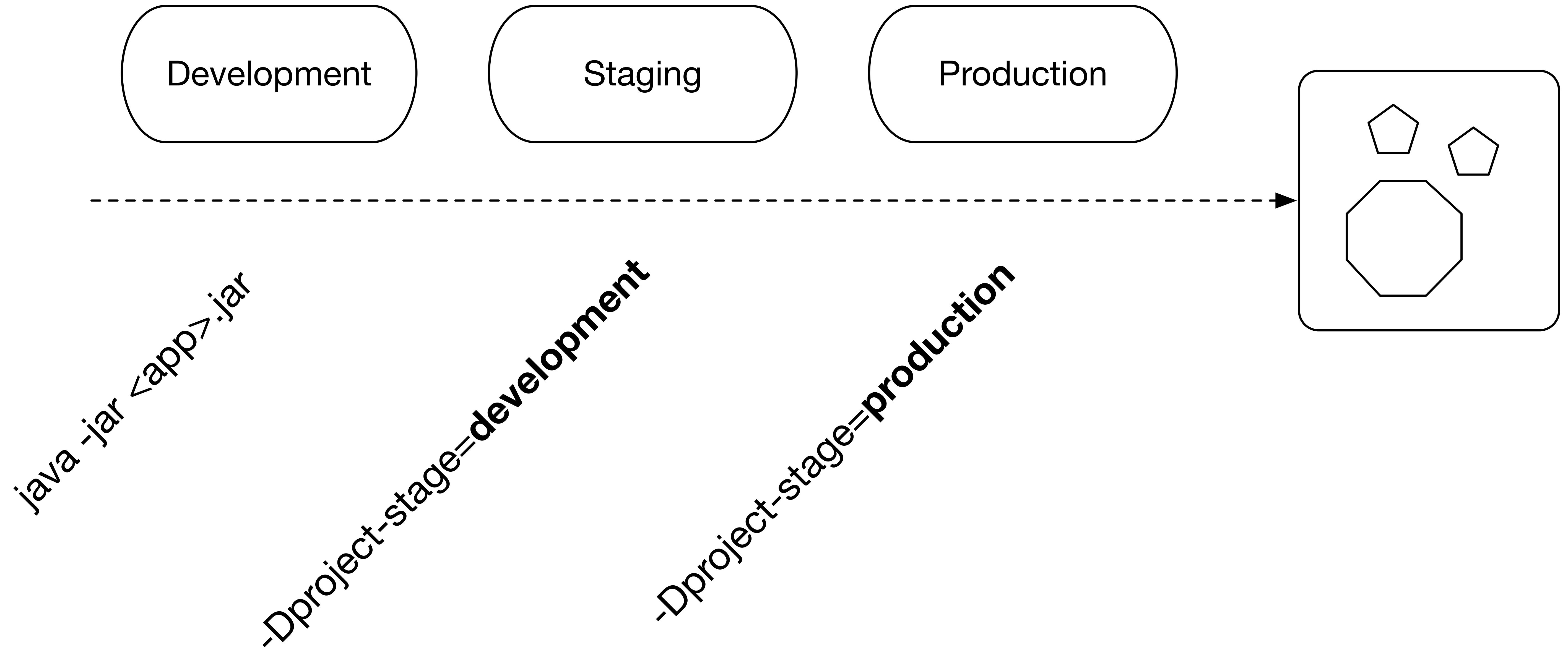
Immutable deployments

project-defaults.yml

```
logger:
    level: DEBUG
database:
  jdbc:
    url: foo
```

project-ci.yml

```
logger:
    level: INFO
database:
  jdbc:
    url: bar
```


Development
Staging
Production
java -jar <app>.jar
-Dproject-stage=**development**
-Dproject-stage=**production**

Service registration

Advertise service presence

pom.xml:

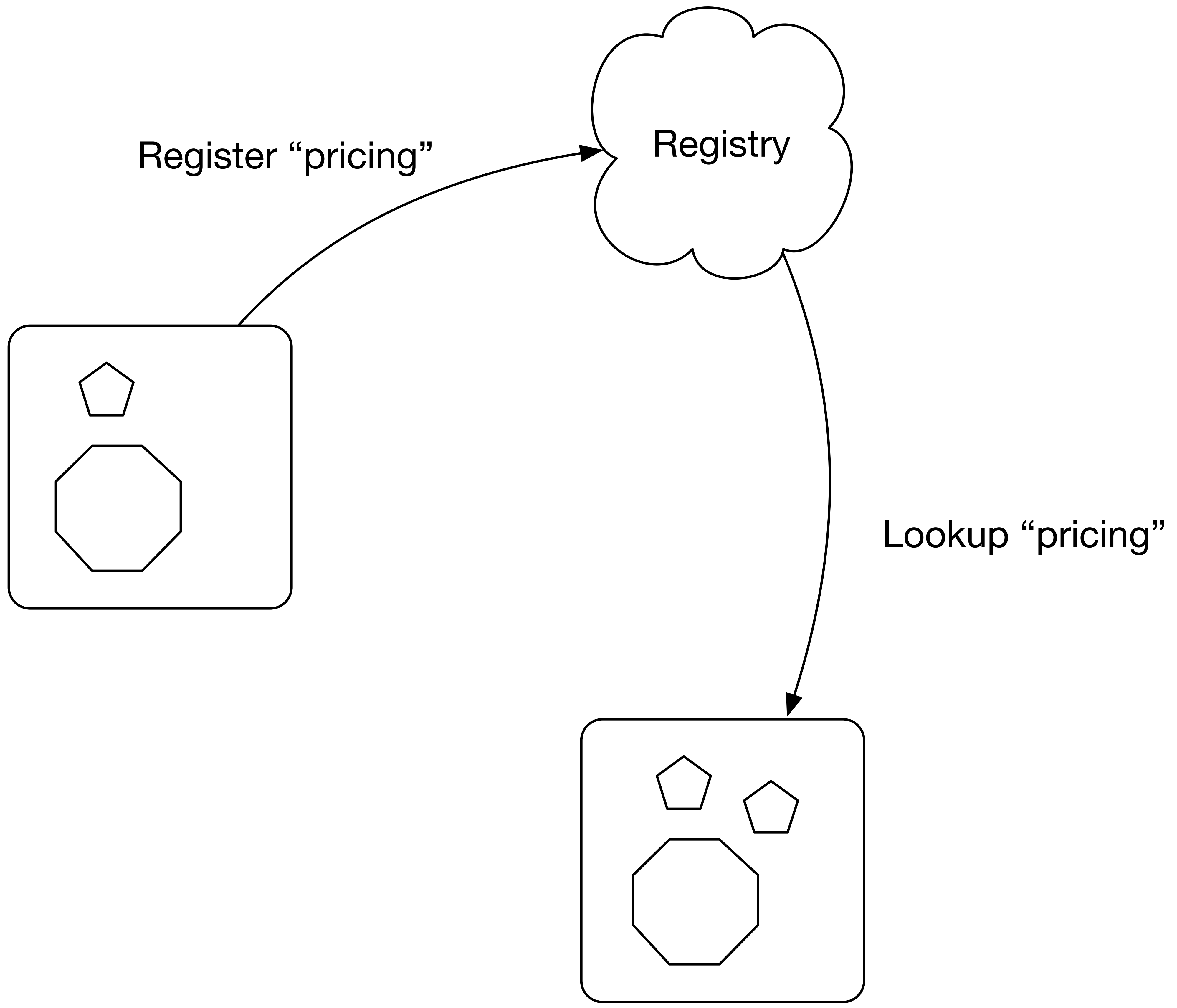
```
<dependency>
  <groupId>
    org.wildfly.swarm
  </groupId>
  <artifactId>
   topology-consul
  </artifactId>
</dependency>
```

project-defaults.yaml:

```
swarm:
  topology:
    consul:
      url: http://localhost:8500
```

PersonsResource.java:

```
@Path("persons")
@Advertise("person-service")
public class PersonResource {
  ...
}
```



API Descriptions (1/2)

Making it easy for people to consume your services

pom.xml:

```
<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>swagger</artifactId>
</dependency>

<dependency>
    <groupId>org.wildfly.swarm</groupId>
    <artifactId>swagger-webapp</artifactId>
</dependency>
```

PersonResource.java:

```
@Path(“persons”)
@Advertise("person-service")
@Api(description = "person resources",
tags = "person")
public class PersonResource {
    @GET
    @Produces("application/xml")
    @ApiOperation(value = "Retrieve all
person resources",
            notes = "Returns a
collection",
            response = Person.class
    )
    public Person[] get() {
    }
}
```

API Descriptions (2/2)

```
"paths": {
"/persons": {
    "get": {
        "description": "Returns a collection",
        "operationId": "get",
        "parameters": [],
        "produces": [
            "application/xml"
        ],
        "responses": {
            "200": {
                "description": "successful operation"
            }
        },
        "summary": "Retrieve all person resources",
        "tags": [
```

Secure Service Access (1/1)

Using single sign on

pom.xml:

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>keycloak</artifactId>
</dependency>
```

project-defaults.yml:

```
swarm:
  deployment:
    javaee7-simple-sample.war:
      web:
        login-config:
          auth-method: KEYCLOAK
        security-constraints:
          - url-pattern: /resources/*
            methods: [GET]
            roles: [admin]
```

Secure Service Access (2/2)

Using Bearer Tokens

```
$ curl http://localhost:8080/
resources

HTTP/1.1 401 Unauthorized
```

```
$ curl -H "Authorization: bearer $TOKEN"
http://localhost:8080/resources/persons

HTTP/1.1 200
```

Resilience

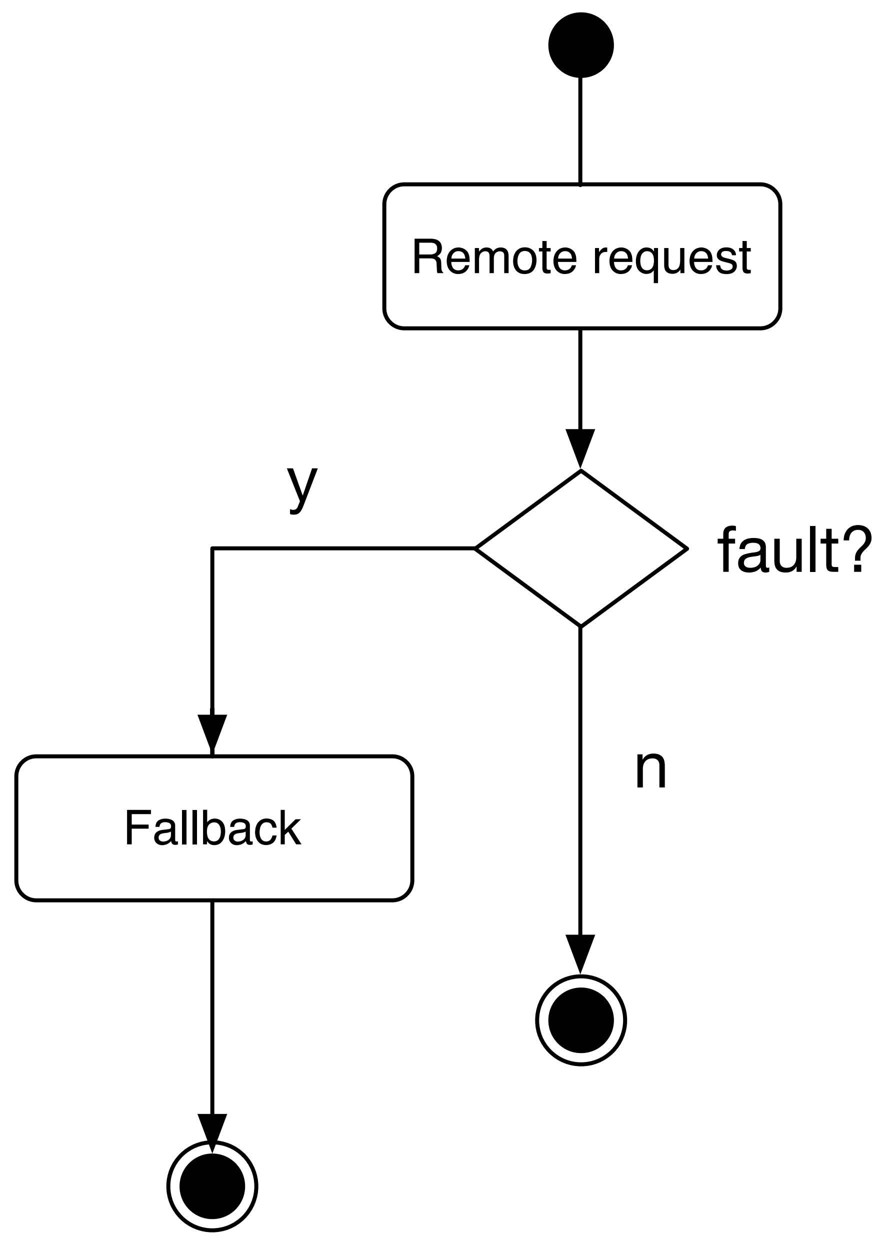
… it’s distributed systems, remember?

```
<dependency>
  <groupId>
    org.wildfly.swarm
   </groupId>
  <artifactId>
   hystrix
  </artifactId>
</dependency>
```

```
@CircuitBreaker(fallbackMethod = "myFallback")
public String isolatedCommand() {
 Client client = ClientBuilder.newClient();
 WebTarget target = client.target(“pricing”);

 Response response = target.get();
 return response.readEntity(String.class);
}

public String myFallback() {
 return ..; // cached values
}
```



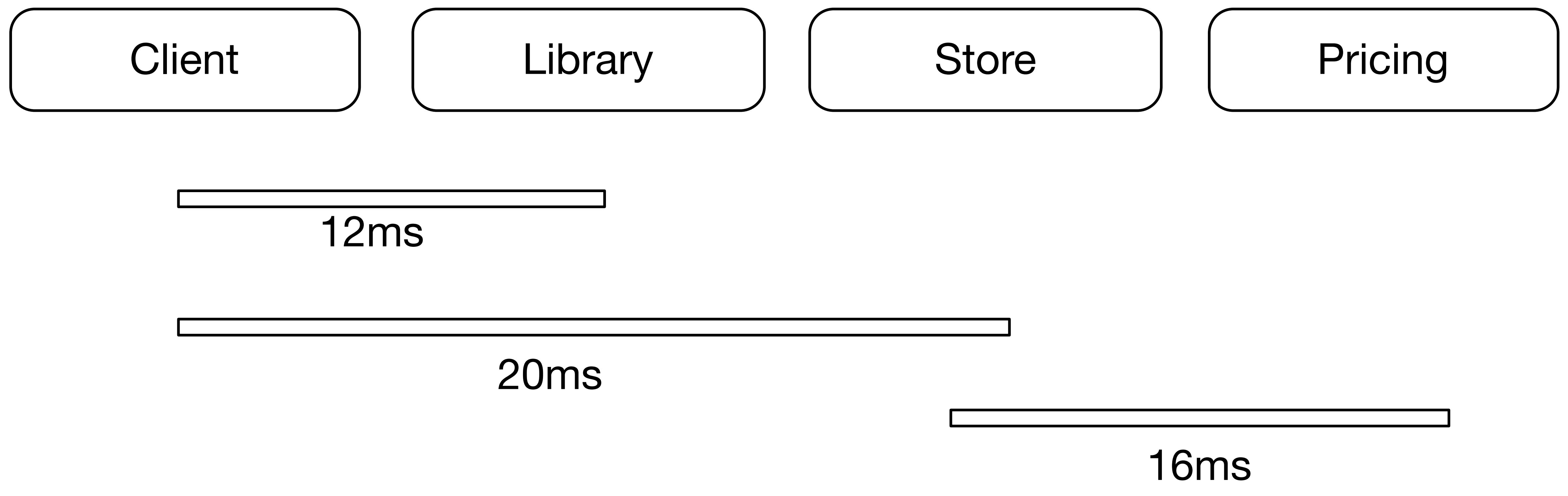
Distributed Tracing

… where do invocations spend there time?

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>zipkin</artifactId>
</dependency>
```

project-defaults.yaml

```
swarm:
  zipkin:
    name: store
    url: http://foobar/api/v1/spans
```

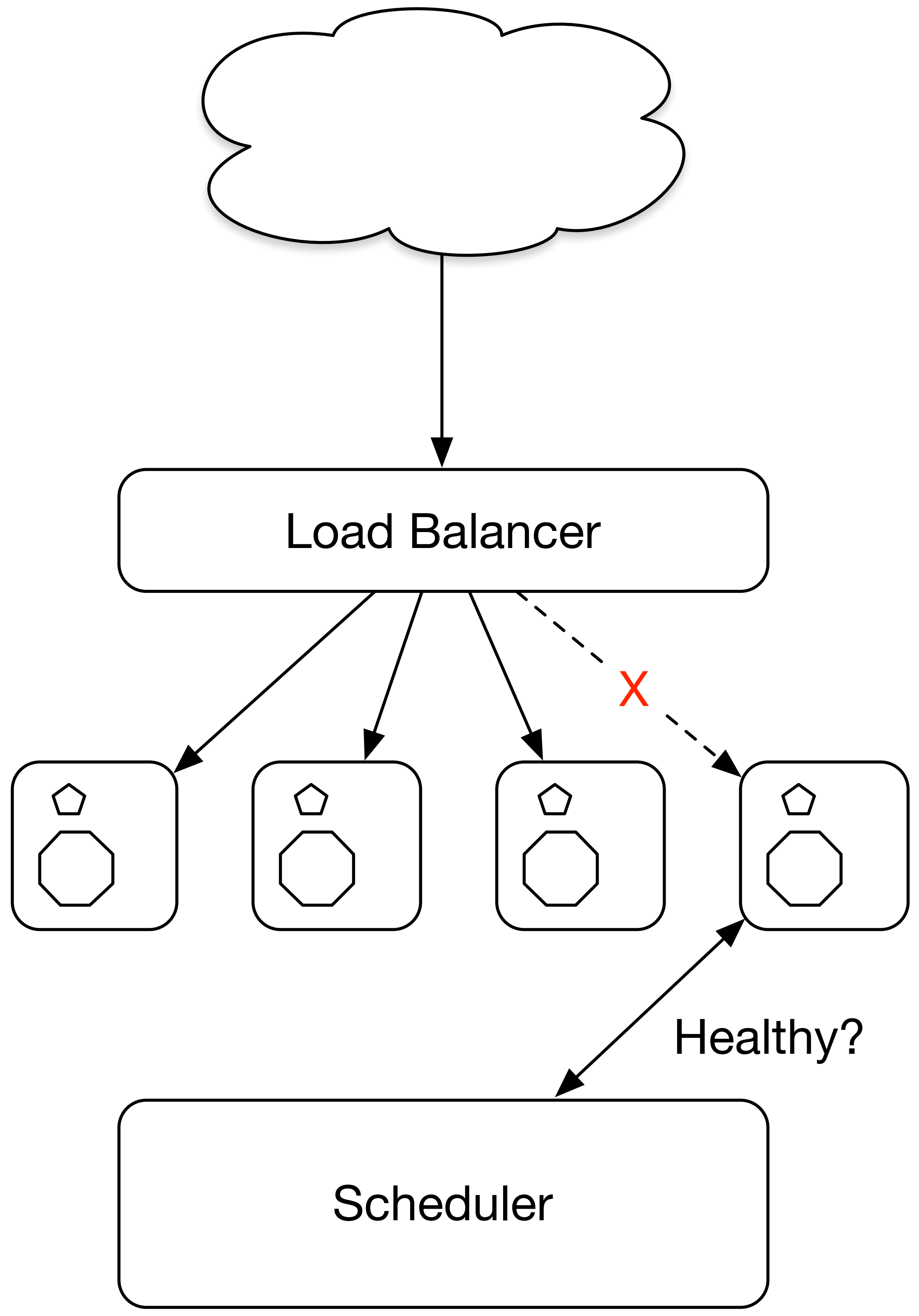


Health checks

Contracts with the cloud platform

```
<dependency>
  <groupId>org.wildfly.swarm</groupId>
  <artifactId>monitor</artifactId>
</dependency>
```

```
@GET
@Path("/check")
@Health
public HealthStatus readiness() {
    return HealthStatus.up();
}
```



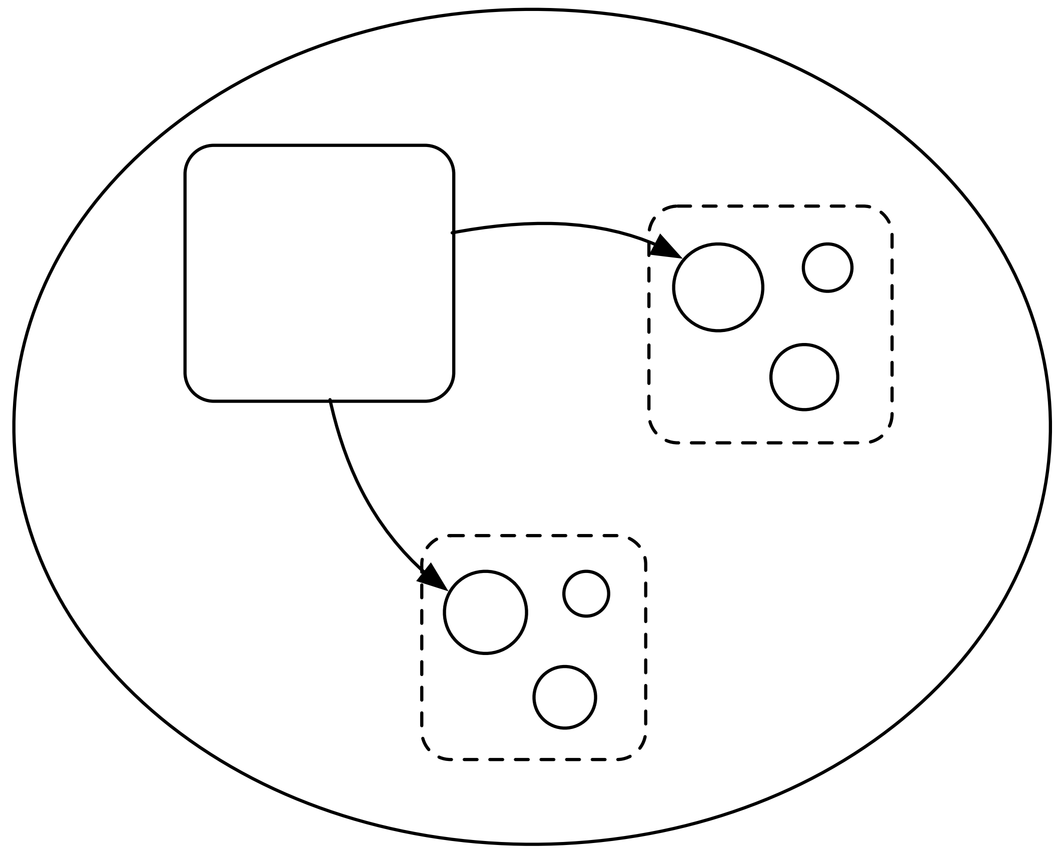
There is much more it

… more then we can cover today

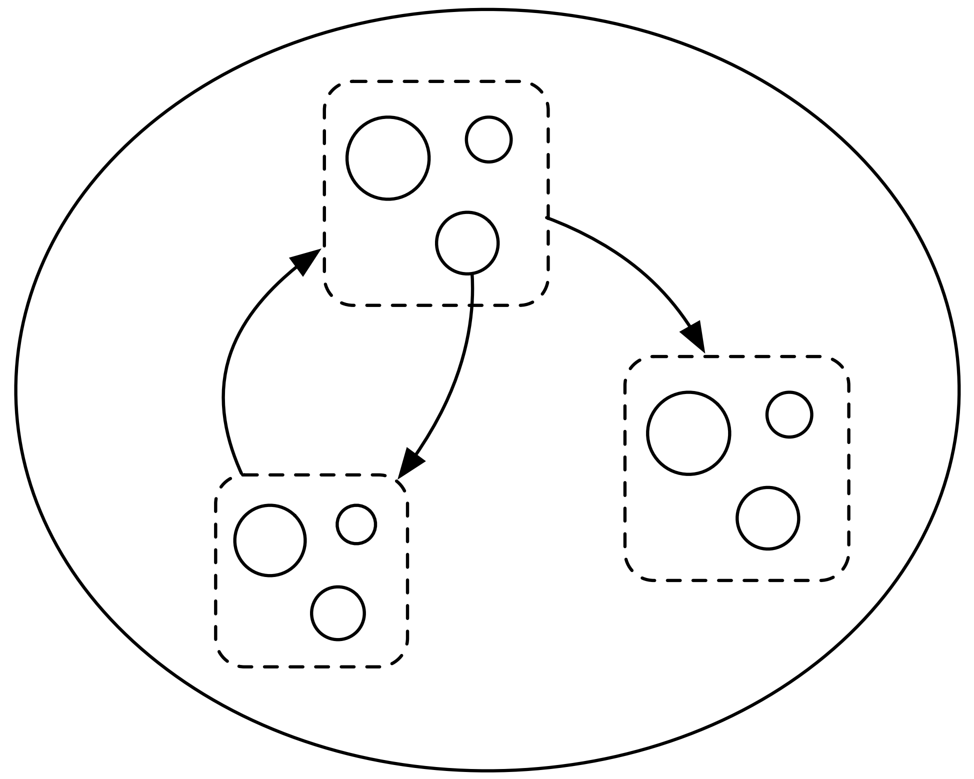
- Logstash/Fluentd
- Netflix Ribbon
- Other service registries
- Openshift / Kubernetes Integration
- Vert.x Integration
- Jolokia
- Infinispan
- Remote Management
- ActiveMQ Integration
- Contract-Based Testing
- …

Spectrum of possibilities

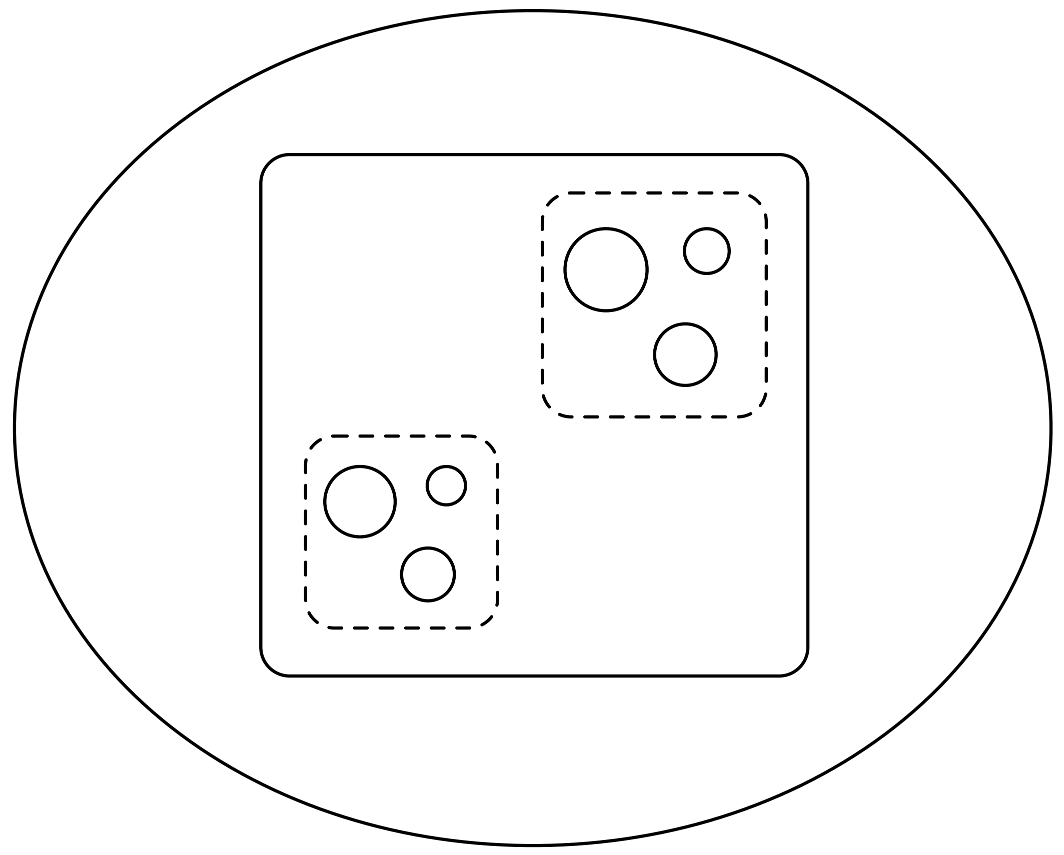
Monolith & microservices



All microservices



Self-Contained systems



RED HAT DEVELOPERS

Thanks!

Visit **http://wildfly-swarm.io** for more information

Join us on IRC: @wildfly-swarm at **freenode.net**