adaptTo()

APACHE SLING & FRIENDS TECH MEETUP

2 - 4 SEPTEMBER 2019

From 0 to HERO in under 10 seconds

Radu Cotescu, Karl Pauls - Adobe

Who
are we?

adaptTo()

Mr. Scripting

@raducotescu

- Computer Scientist @ Adobe, Basel, Switzerland
- Member of the Apache Software Foundation
- Apache Sling PMC member
- Maintainer of HTL for Apache Sling
- Initiator of the Apache Sling Validation Framework

adaptTo()

Mr. Classloader

@karlpauls

- Computer Scientist @ Adobe, Basel, Switzerland
- Member of the Apache Software Foundation
- Apache Sling and Apache Felix PMC (VP) member
- Co-Author of OSGi in Action

Apache Sling Scripting Reloaded[0]

Wait, is this 2018 again?!?!

Apache Sling Scripting Bundle Tracker[1]

Core principles:

1. **pack scripts** into OSGi **bundles**
2. define the resource types as **versioned capabilities**, with **versioned requirements** (Java APIs, other resource types to which scripts delegate or which scripts extend)
3. allow the platform to do what it’s made to: wire things

Apache Sling Scripting Bundle Tracker[1]

What:

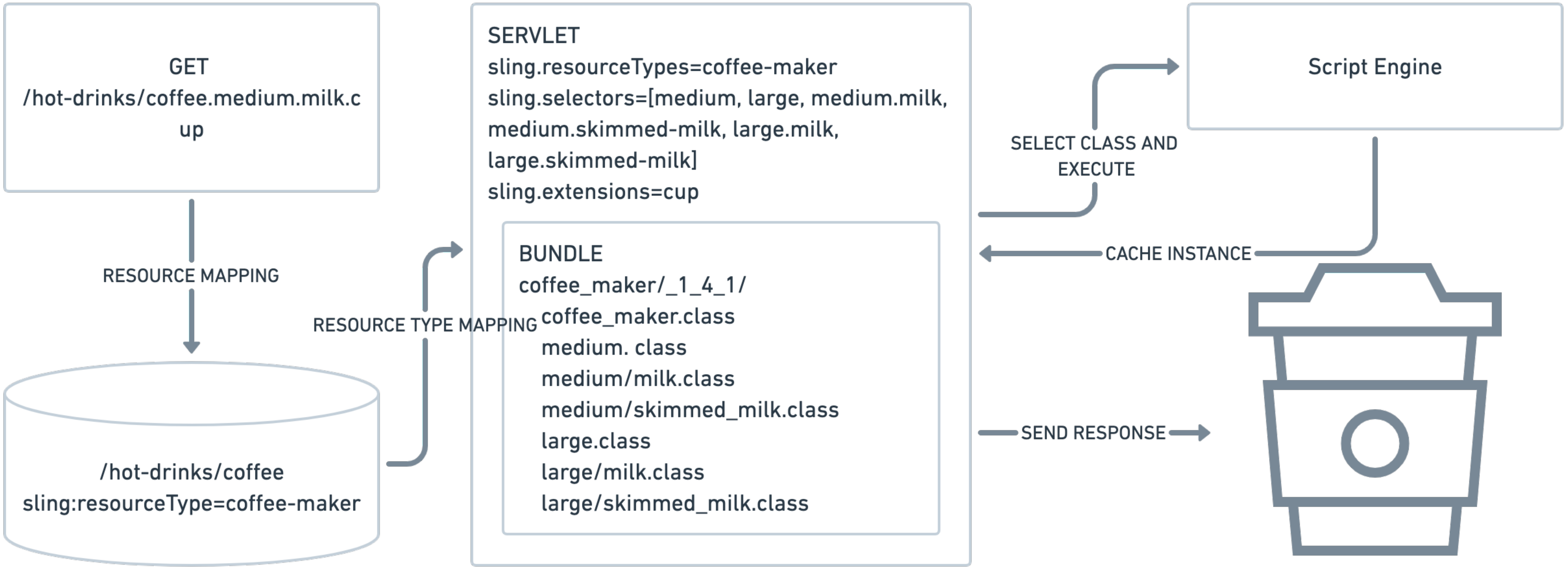
1. add-on module to which bundles that provide scripts have to be wired explicitly
2. reuses the already established mechanisms for registering servlets in Apache Sling
3. allows building light-weight instances that can be thrown into production with very little warm-up, when using precompiled scripts

adaptTo()

Apache Sling Scripting Bundle Tracker[1]

4. provides the mechanism for deploying truly versionable scripts, with explicit dependencies, by relying on the OSGi framework
5. removes the need of a separate ScriptCache
6. removes additional pressure on the persistence layer
7. simplifies instance and application upgrades
8. Maven plugin for generating requirements and capabilities

If there's something cool from 2018



adaptTo()

From 0 to HERO in under 10 seconds

adaptTo()

However, it’s 2019

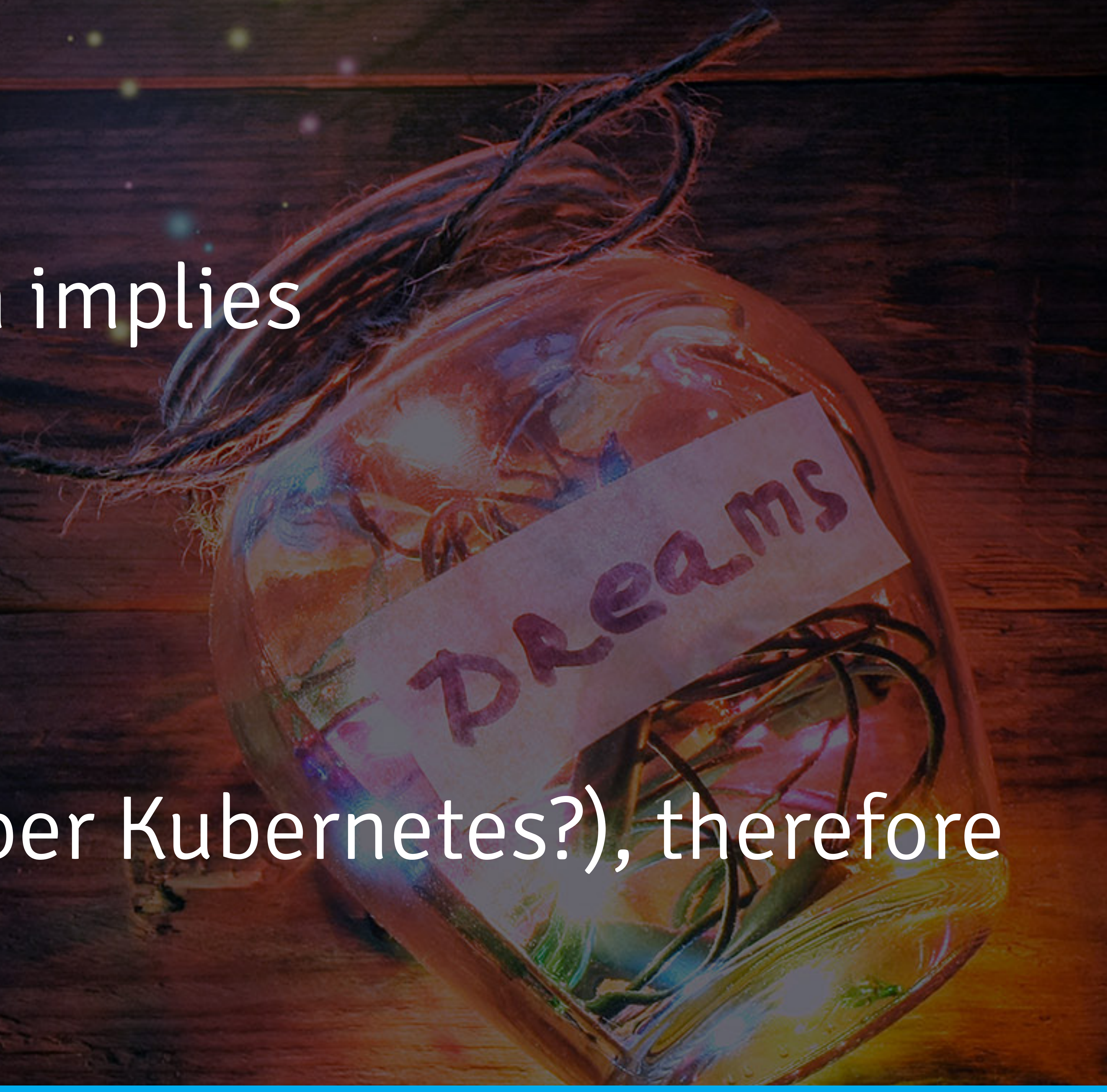
And in 2019 the buzzwords are…

Microservices Cloud Native DevOps Stateless XaaS

If Sling would be a Docker image […]

A container should be:

- ✓ immutable
- ✓ reasonably lightweight, which implies
  - ✓ small disk footprint
  - ✓ small memory footprint
  - ✓ blazing fast start-up time
- ✓ horizontally scalable (remember Kubernetes?), therefore preferably stateless

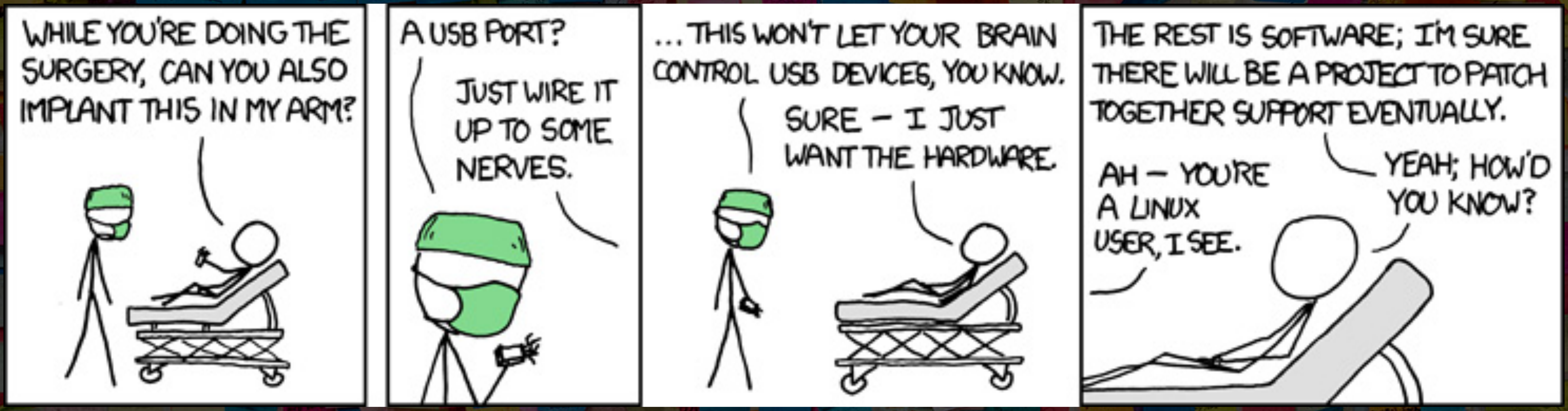


[...] under 100 MB [...]

1. Sprinkle Apache Sling Feature Model to taste:
    1. no configurations on disk
    2. lightweight launcher (no caching of artifacts)
2. Mix a bit of `jlink` to create a custom JRE:
    1. use `jdeps` to figure out the dependencies
    2. try to avoid `java.desktop`, if possible
3. Base your image on `alpine`

[...] and completely stateless

1. Use a customised Sling setup – minimum number of bundles
   - JCR-less Sling (we know, it's heresy) - the Resource Provider API can be used to expose Resources from anywhere


WHILE YOU'RE DOING THE SURGERY, CAN YOU ALSO IMPLANT THIS IN MY ARM?
A USB PORT?
JUST WIRE IT UP TO SOME NERVES.
...THIS WON'T LET YOUR BRAIN CONTROL USB DEVICES, YOU KNOW.
SURE – I JUST WANT THE HARDWARE.
THE REST IS SOFTWARE; I'M SURE THERE WILL BE A PROJECT TO PATCH TOGETHER SUPPORT EVENTUALLY.
AH – YOU'RE A LINUX USER, I SEE.
YEAH; HOW'D YOU KNOW?

https://xkcd.com/644/

[...] and completely stateless

2 . /content exposed through remote Resources
3. Precompiled component scripts served through bundles (separation of concerns [1])
4. Immutable deployment
   - Feature Model only

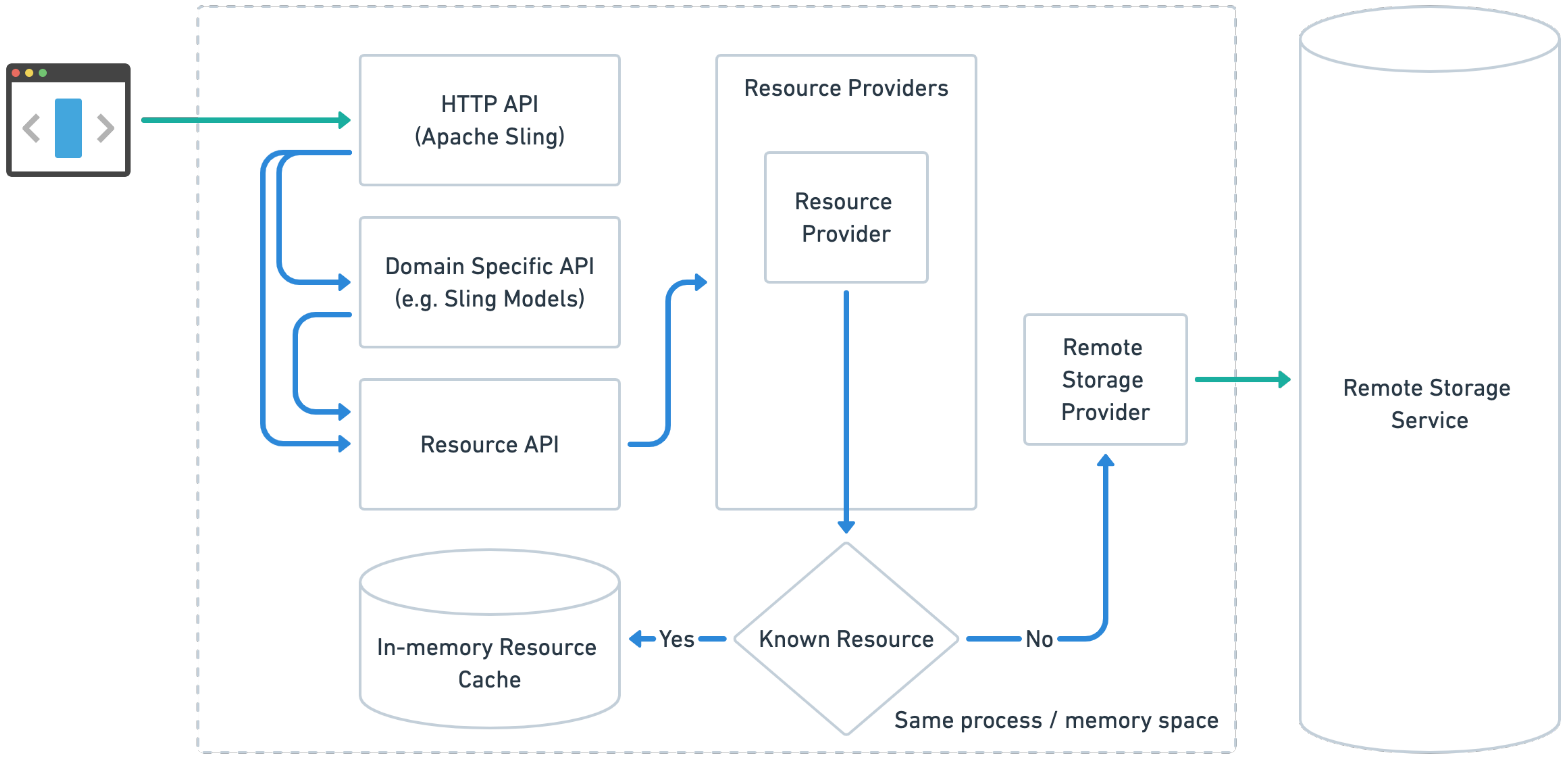
RemoteStorageProvider API

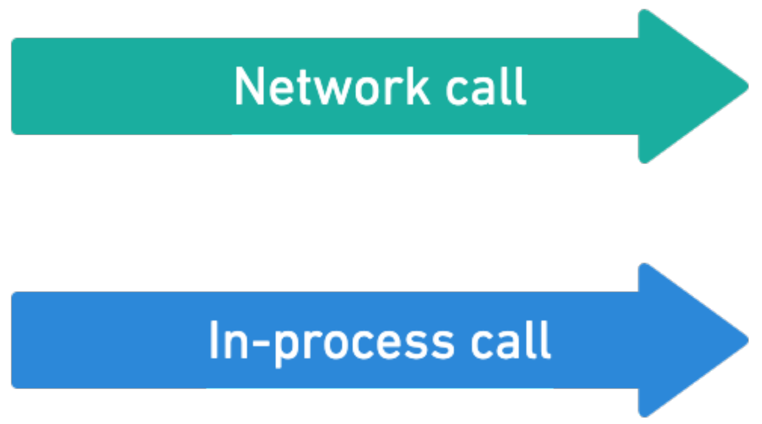
```java
/**
 * A {@code RemoteStorageProvider} is responsible for retrieving the {@link RemoteResourceReference}s
 * corresponding to a certain Sling path.
 */
@ProviderType
public interface RemoteStorageProvider {

    @Nullable
    RemoteResourceReference findResource(@NotNull String slingPath, @NotNull Map<String, Object> authenticationInfo);

    @Nullable
    File getFile(@NotNull RemoteResourceReference reference, @NotNull Map<String, Object> authenticationInfo);

    @Nullable
    Directory getDirectory(@NotNull RemoteResourceReference reference, @NotNull Map<String, Object> authenticationInfo);

}
```

RemoteStorageProvider API

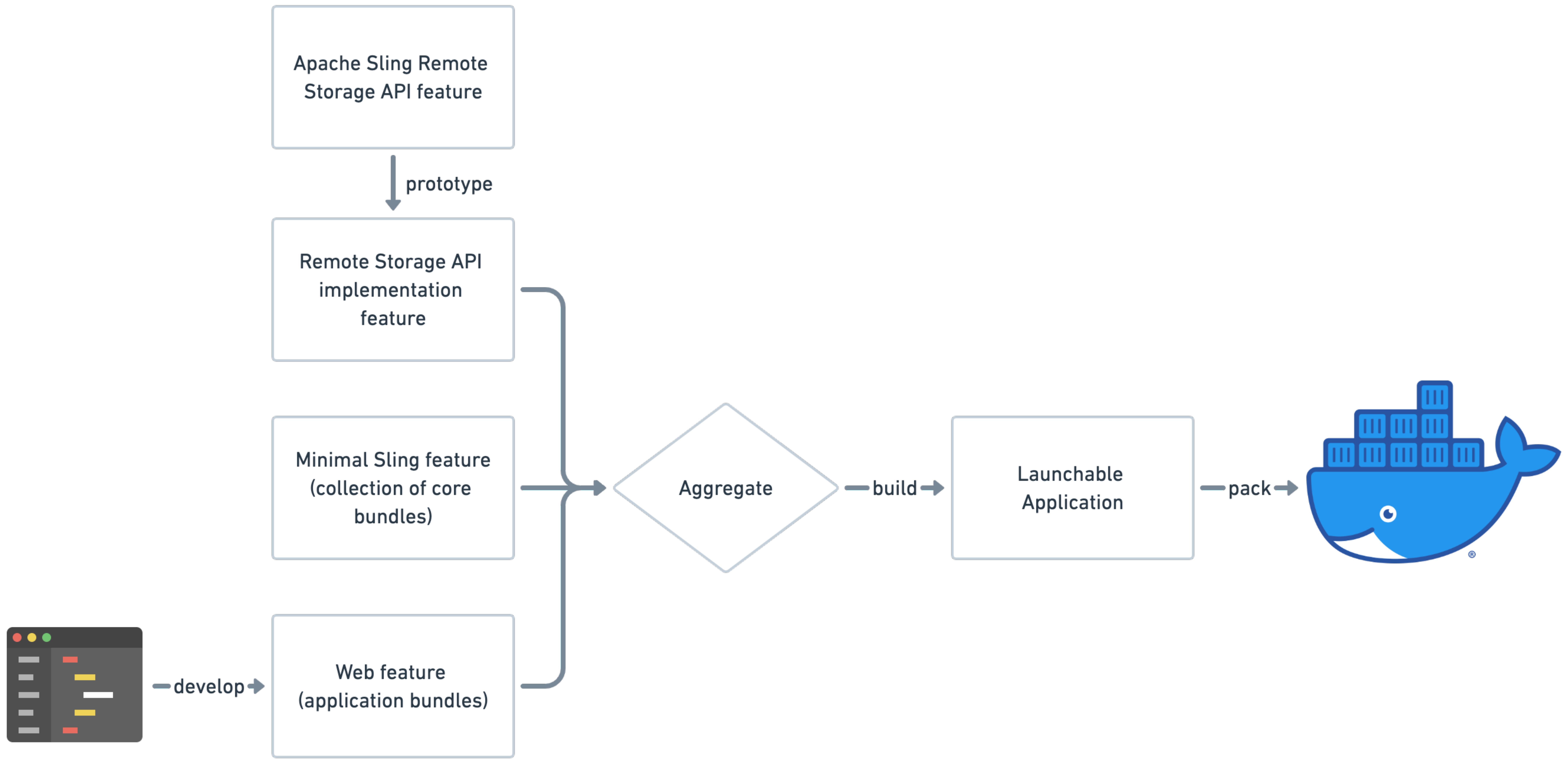
- Experimental! API to have different remote resources hooked into Sling via `ResourceProviders`:
  - 1:1 mapping between a `ResourceProvider` and a `RemoteStorageProvider`
  - Provides a `Resource` tree based on files and folders, with a special `.sling.json` file for defining properties
  - Includes an in-memory caching layer and event handling (optional); should probably delegate this to Redis

RemoteStorageProvider API





Putting it all together


Apache Sling Remote Storage API feature
prototype
Remote Storage API implementation feature
Minimal Sling feature (collection of core bundles)
develop
Web feature (application bundles)
Aggregate
build
Launchable Application
pack

adaptTo()

Demo*

*or how we can embarrass ourselves if things don't work

So what was this demo about?

- ✓ stateless Docker container
- ✓ 75 MB JCR-less Sling web application including the JRE
- ✓ less than 200 MB memory footprint
- ✓ less than 10 seconds elapsed before first rendering*
- ✓ 0 dynamically generated classes (no compilers)

* on a crazy expensive 2018 MacBook Pro ; good luck deploying servers so powerful in production!

Where do we go from here?

OSGi RFP 196 [2]

- Provides a way to use an OSGi framework with custom classloaders (a.k.a. OSGi Connect/PojoSR)

Graal/Substrate VM

- Ahead-of-Time (AOT) Java code compilation

Together with the precompiled bundled scripts it should be possible to perform an AOT compilation of our Sling application as a native image

adaptTo()

Q&A

adaptTo()

Resources

[0] – https://adapt.to/2018/en/schedule/apache-sling-scripting-reloaded.html

[1] - https://github.com/apache/sling-org-apache-sling-scripting-bundle-tracker

[2] - https://github.com/osgi/design/blob/master/rfps/rfp-0196-OSGiConnect.pdf

Assets licensed from https://stock.adobe.com/

Our diagrams were designed with https://whimsical.co/flowcharts/

Code available after the talk at https://github.com/apache/sling-whiteboard/tree/master/it-is-cloudy-here