Principles of Software Construction: Objects, Design, and Concurrency

Organizing Systems at Scale: Modules, Services, Architectures

Claire Le Goues Vincent Hellendoorn

Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Administrative

Exam Thursday.

HW6 release.

Where we are

Design for

understanding

change/ext.

reuse

robustness

...

Small scale: One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
Subtype Polymorphism ✓ Information Hiding, Contracts ✓ Immutability ✓ Types Unit Testing ✓	Domain Analysis ✓ Inheritance & Del. ✓ Responsibility Assignment, Design Patterns, Antipattern ✓ Promises/ Reactive P. ✓ Integration Testing ✓	GUI vs Core ✓ Frameworks and Libraries ✓, **APIs** ✓ **Module systems, microservices** Testing for Robustness ✓ CI ✓, DevOps, Teams

Breaking Changes

Backward Compatible Changes

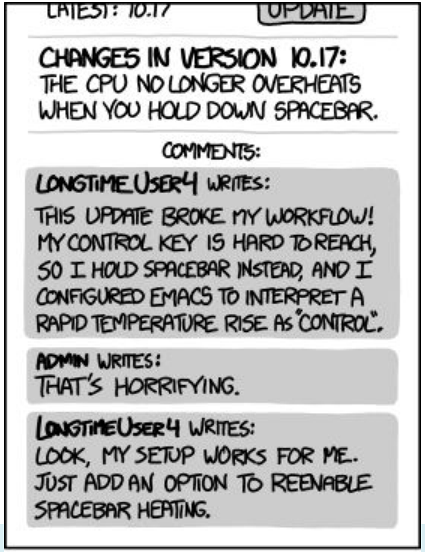
Can add new interfaces, classes

Can add methods to APIs,
but cannot change interface implemented by clients

Can loosen precondition and tighten postcondition,
but no other contract changes

Cannot remove classes, interfaces, methods

Clients may rely on undocumented behavior and even bugs


LATEST: 10.17
UPDATE
CHANGES IN VERSION 10.17:
THE CPU NO LONGER OVERHEATS WHEN YOU HOLD DOWN SPACEBAR.
COMMENTS:
LONGTIME USER4 WRITES:
THIS UPDATE BROKE MY WORKFLOW! MY CONTROL KEY IS HARD TO REACH, SO I HOLD SPACEBAR INSTEAD, AND I CONFIGURED EMACS TO INTERPRET A RAPID TEMPERATURE RISE AS "CONTROL".
ADMIN WRITES:
THAT'S HORRIFYING.
LONGTIMEUSER4 WRITES:
LOOK, MY SETUP WORKS FOR ME. JUST ADD AN OPTION TO REENABLE SPACEBAR HEATING.

EVERY CHANGE BREAKS SOMEONE'S WORKFLOW.

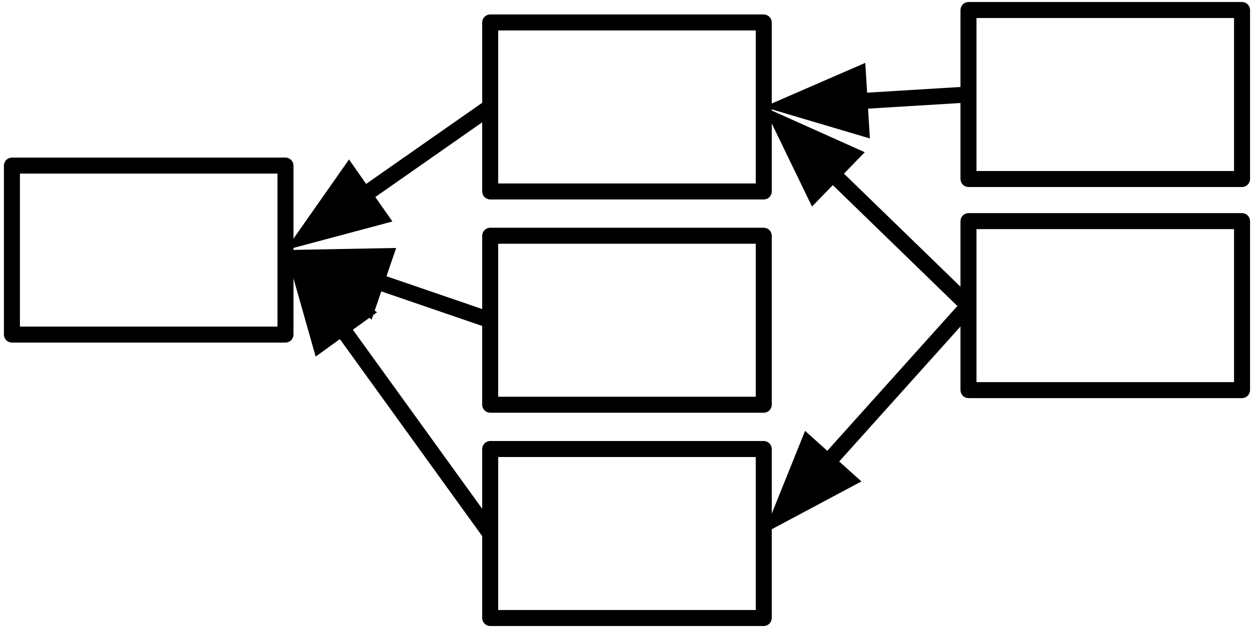
Breaking Changes

Not backward compatible (e.g., renaming/removing method)

Clients may need to change their implementation when they update

or even migrate to other library

May cause costs for rework and interruption, may ripple through ecosystem



Software Ecosystem

Breaking changes can be hard to avoid

Need better planning? (Parnas' argument)

Requirements and context change

Bugs and security vulnerabilities

Inefficiencies

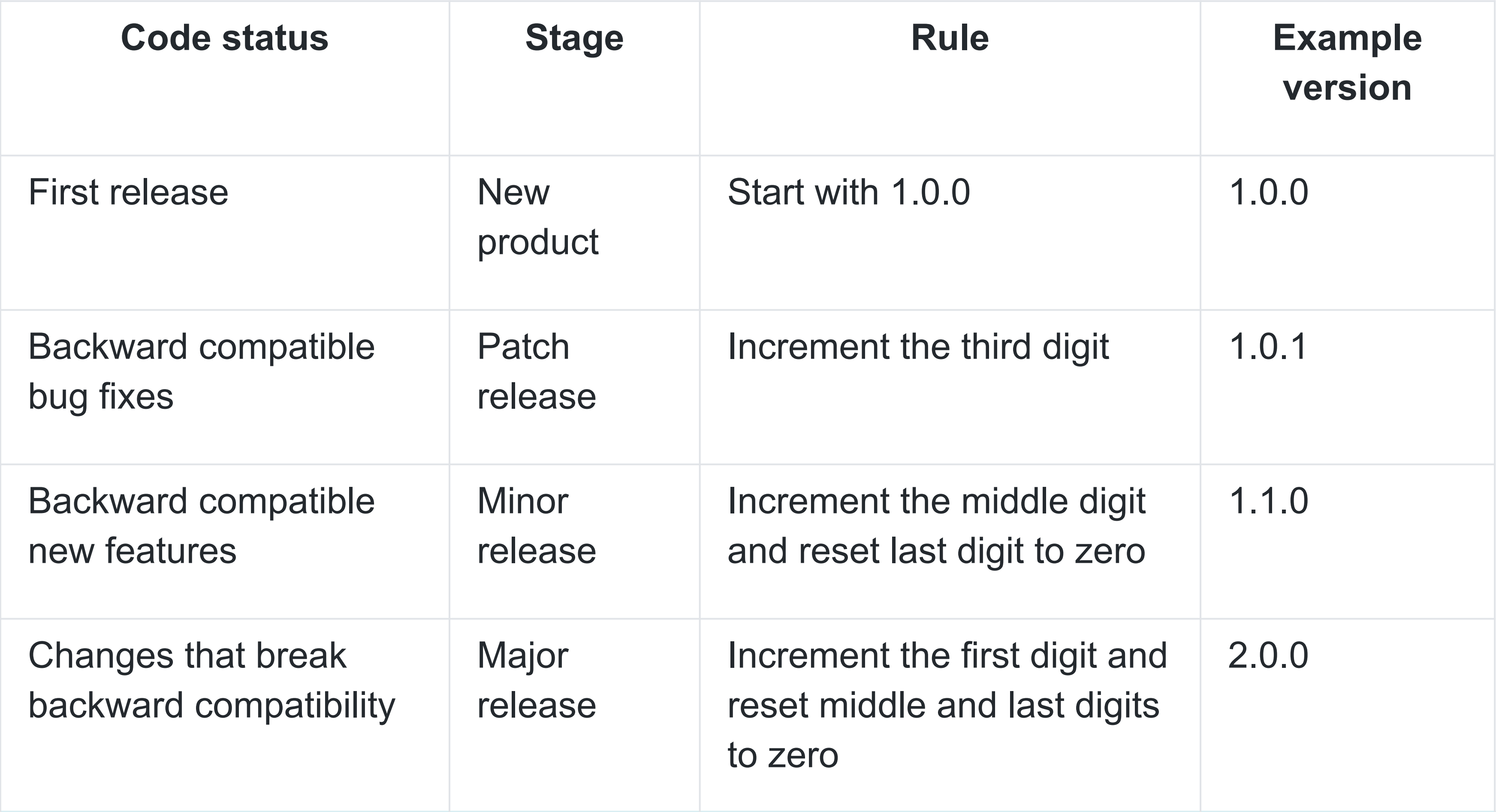
Rippling effects from upstream changes

Technical debt, style

Semantic Versioning

Given a version number MAJOR.MINOR.PATCH, increment the:

1. MAJOR version when you make incompatible API changes,
2. MINOR version when you add functionality in a backwards compatible manner, and
3. PATCH version when you make backwards compatible bug fixes.



Code status	Stage	Rule	Example version
First release	New product	Start with 1.0.0	1.0.0
Backward compatible bug fixes	Patch release	Increment the third digit	1.0.1
Backward compatible new features	Minor release	Increment the middle digit and reset last digit to zero	1.1.0
Changes that break backward compatibility	Major release	Increment the first digit and reset middle and last digits to zero	2.0.0

https://docs.npmjs.com/about-semantic-versioning

Cost distributions and practices are community dependent

eclipse

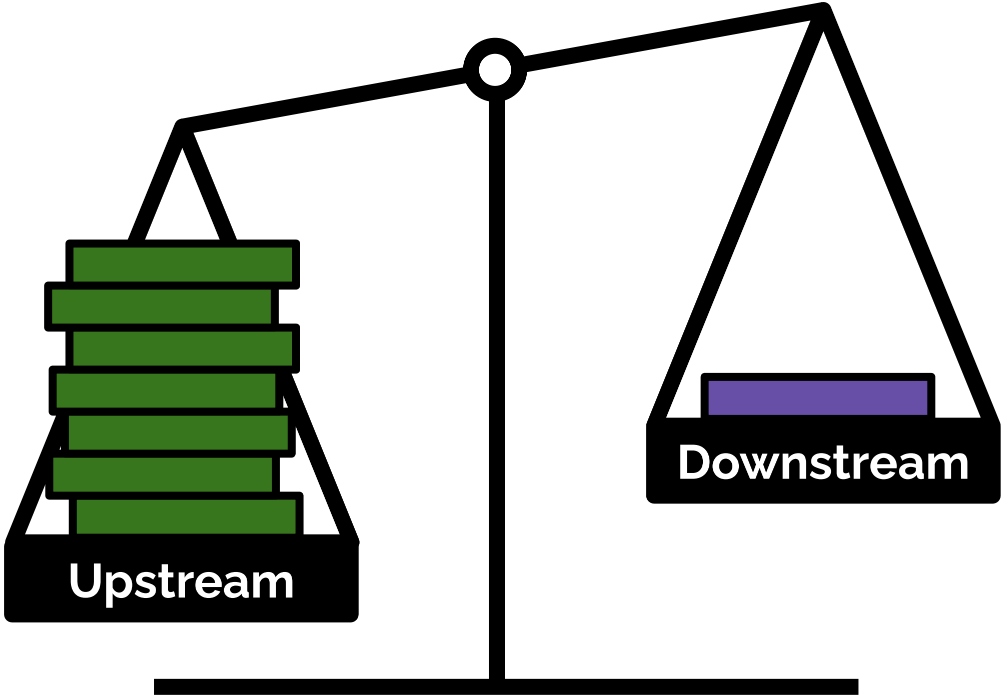
Backward compatibility to reduce costs for **clients**

"API Prime Directive: When evolving the Component API from to release to release, do not break existing Clients"

https://wiki.eclipse.org/Evolving_Java-based_APIs

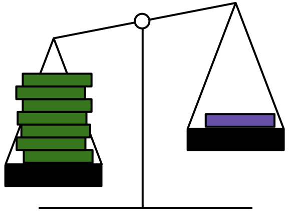
Values

Backward compatibility for clients



Yearly synchronized coordinated releases

Backward compatibility for clients



Willing to accept high costs + opportunity costs

Educational material, workarounds

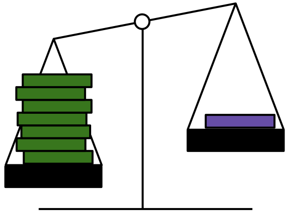
API tools for checking

Coordinated release planning

No parallel releases

Upstream

Backward compatibility for clients



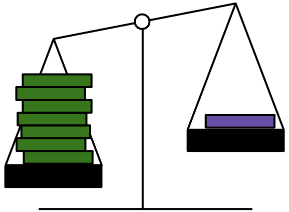
Convenient to use as resource

Yearly updates sufficient for many

Stability for corporate users

Downstream

Backward compatibility for clients



Perceived stagnant development and political decision making

Stale platform; discouraging contributors

Coordinated releases as pain points

SemVer prescribed but not followed

Friction

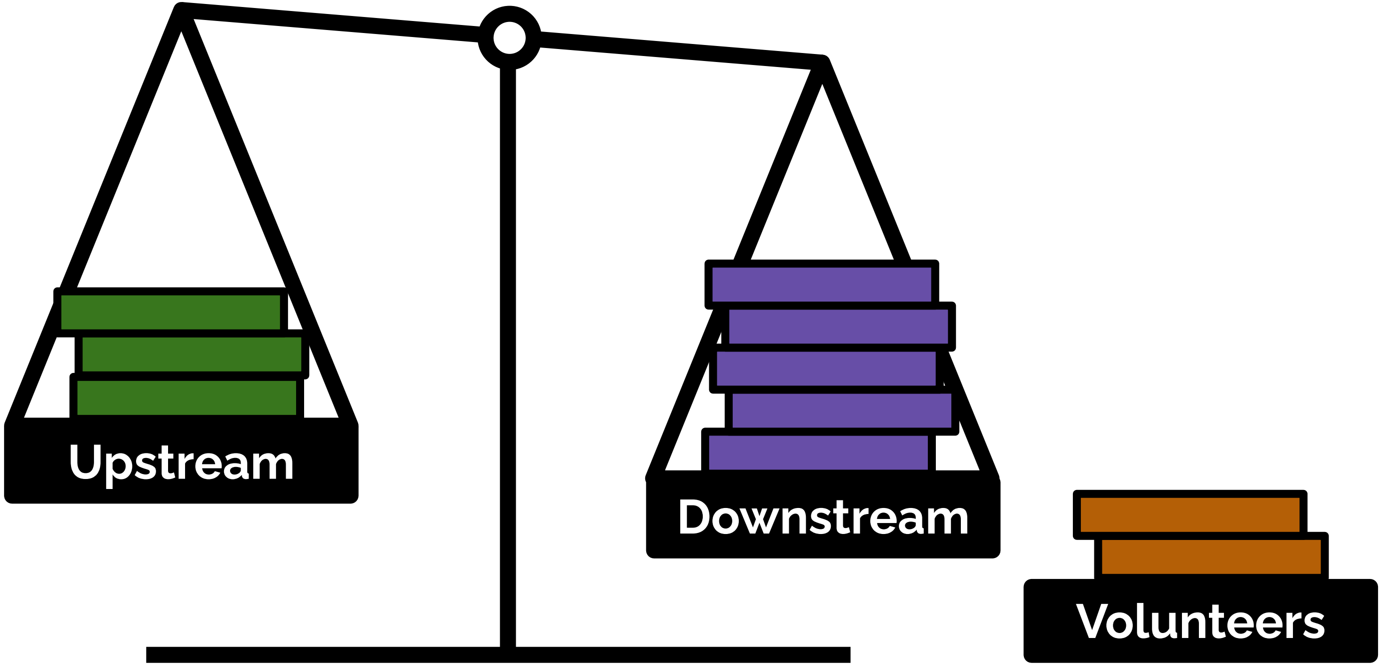
> “Typically, if you have hip things, then you get also people who create new APIs on top ... to create the next graphical editing framework or to build more efficient text editors. ... And these things don’t happen on the Eclipse platform anymore.”

Ease for **end users** to install and update packages

"CRAN primarily has the academic users in mind, who want timely access to current research" [R10]

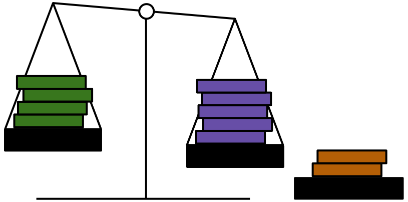
Values

Timely access to current research for end users



Continuous synchronization, ~1 month lag

Timely access to current research for end users



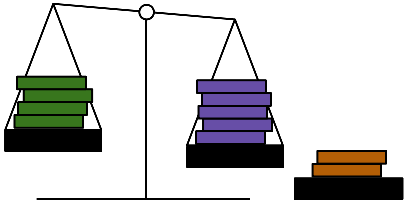
Snapshot consistency within the ecosystem (not outside)

Reach out to affected downstream developers: resolve before release

Gatekeeping: reviews and automated checking against downstream tests

Upstream

Timely access to current research for end users



Waiting for emails, reactive monitoring

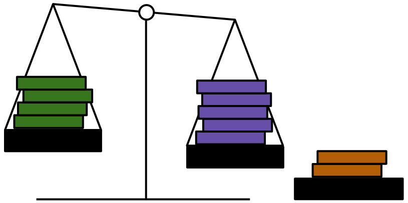
Urgency when upstream package updates

Dependency = collaboration

Aggressive reduction of dependencies, code cloning

Downstream

Timely access to current research for end users



Urgency and reacting to updates as burden vs. welcoming collaboration

Gatekeeping works because of prestige of being in repository

Updates can threaten scientific reproducibility

Friction

> “And then I need to [react to] some change ... and it might be a relatively short timeline of two weeks or a month. And that's difficult for me to deal with, because I try to sort of focus one project for a couple weeks at a time so I can remain productive.”

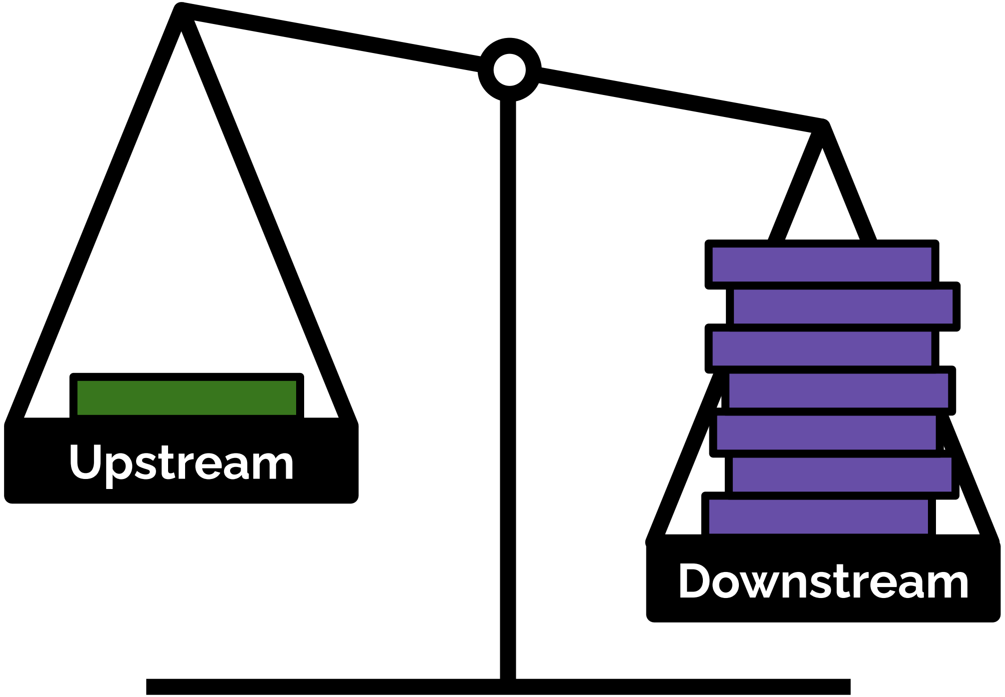
node
JS
TM

Easy and fast for **developers** to publish and use packages

Open to rapid change,
no gate keeping,
experimenting with APIs until they are right

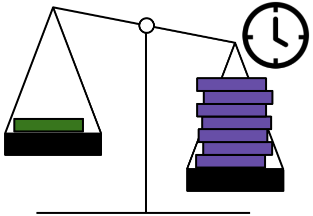
Values

Easy and fast to publish and use for developers



Decoupled pace, update at user's discretion

Easy and fast to publish and use for developers



Breaking changes easy

More common to remove technical debt, fix APIs

Signaling intention with SemVer

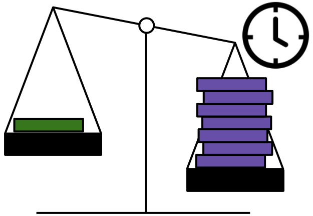
No central release planning

Parallel releases more common

Upstream

node JS™

Easy and fast to publish and use for developers



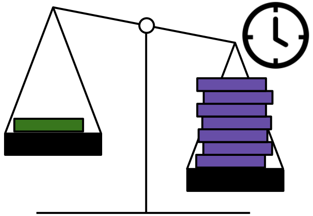
Technology supports using old + mixed revisions; decouples upstream and downstream pace

Choice to stay up to date

Monitoring with social mechanisms and tools (e.g., greenkeeper)

Downstream

Easy and fast to publish and use for developers



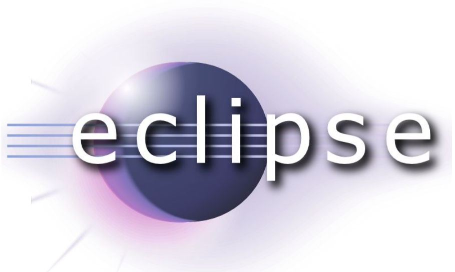
Rapid change requires constant maintenance

Emphasis on tools and community, often grassroots

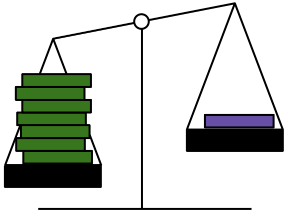
Friction

“Last week’s tutorial is out of date today.”

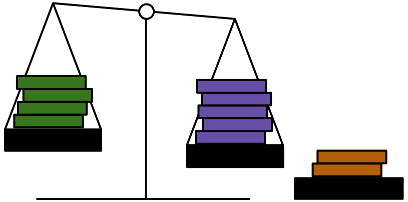
Contrast

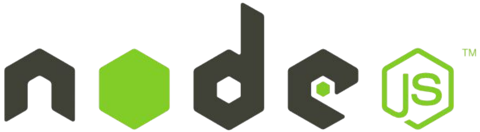


Backward compatibility for clients

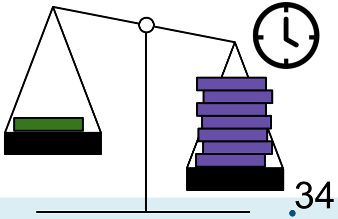


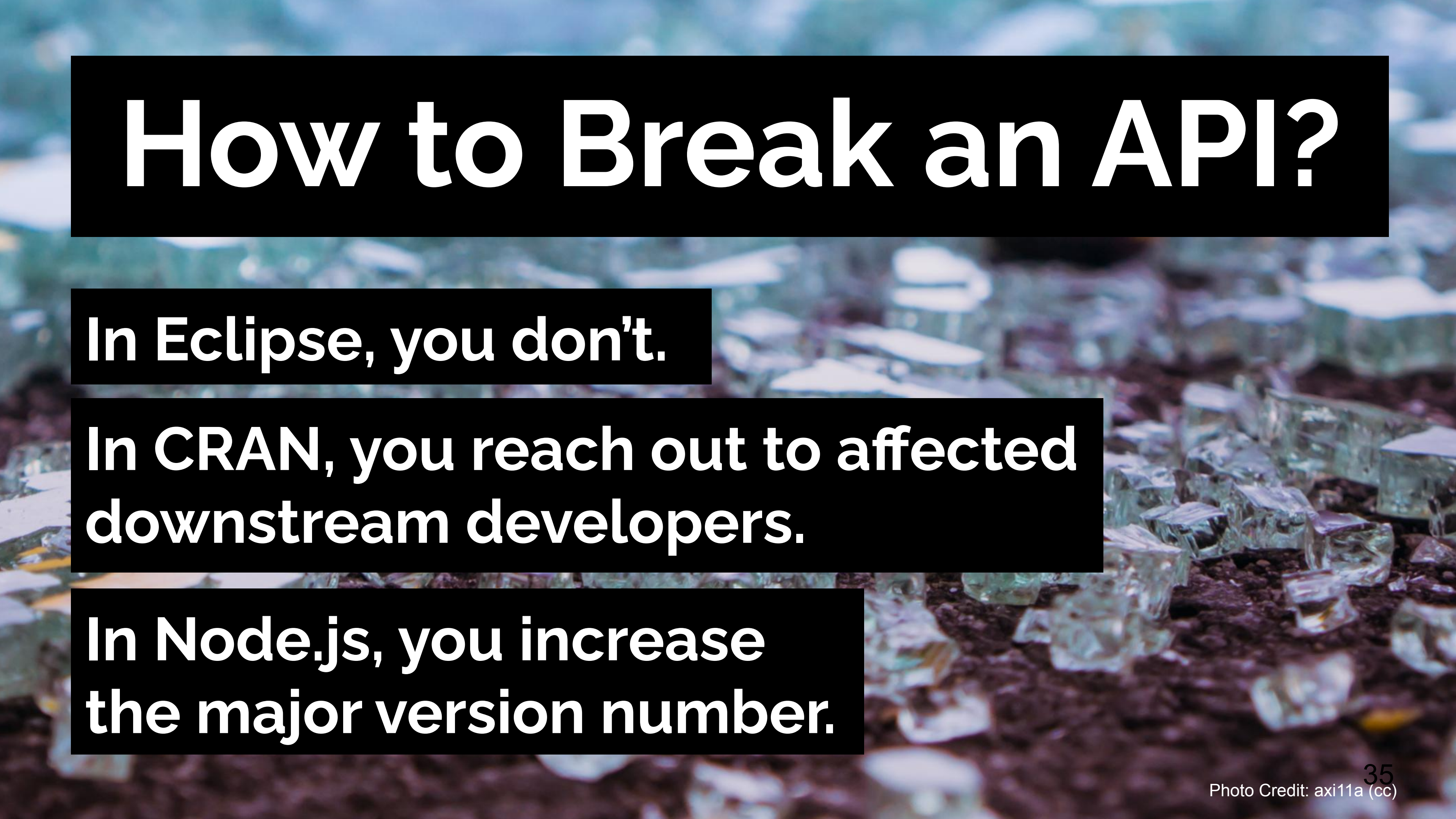
Timely access to current research for end users





Easy and fast to publish/use for developers





How to Break an API?

In Eclipse, you don't.

In CRAN, you reach out to affected downstream developers.

In Node.js, you increase the major version number.

Photo Credit: axi11a (cc)

Traditional Library Reuse

Static/dynamic linking against binaries (e.g., .DLLs)

Copy library code into repository

Limitations?

Package Managers

Refer to library releases by name + version

Immutable storage in repository

Dependency specification in repository

Package manager downloads / updates dependencies

- Maven, npm, pip, cargo, nuget, …

Release libraries to package repository

Module Systems

Foundation for distributing and reusing libraries

Packaging code (binary / source)

Linking against code in a module without knowing internals

Java: Packages and Jar Files

Packages structure name space, avoid naming collisions (edu.cmu.cs17214…)

Public classes are globally visible

- package visibility to hide within package
- no way to express visibility to select packages

.jar files bundle code (zip format internally)

- Java can load classes from all .jar files in classpath
- Java does not care where a class comes from, loads first that matches name

Classpath established at JVM launch

Packages enough?

```
edu.cmu.cs214.santorini

edu.cmu.cs214.santorini.gui

edu.cmu.cs214.santorini.godcards

edu.cmu.cs214.santorini.godcards.impl

edu.cmu.cs214.santorini.logic

edu.cmu.cs214.santorini.utils
```

Toward Module Systems

Stronger encapsulation sometimes desired

Expose only select public packages (and all public classes therein) to other modules

Dynamic adding and removal of modules desired

OSGi (most prominently used by Eclipse)

- Bundle Java code with Manifest
- Framework handles loading with multiple classloaders

```
Bundle-Name: Hello World
Bundle-SymbolicName: org.wikipedia.helloworld
Bundle-Description: A Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Bundle-Activator: org.wikipedia.Activator
Export-Package:
org.wikipedia.helloworld;version="1.0.0"
Import-Package:
org.osgi.framework;version="1.3.0"
```

Java Platform Module System

Since Java 9 (2017); built-in alternative to OSGi

Modularized JDK libraries itself

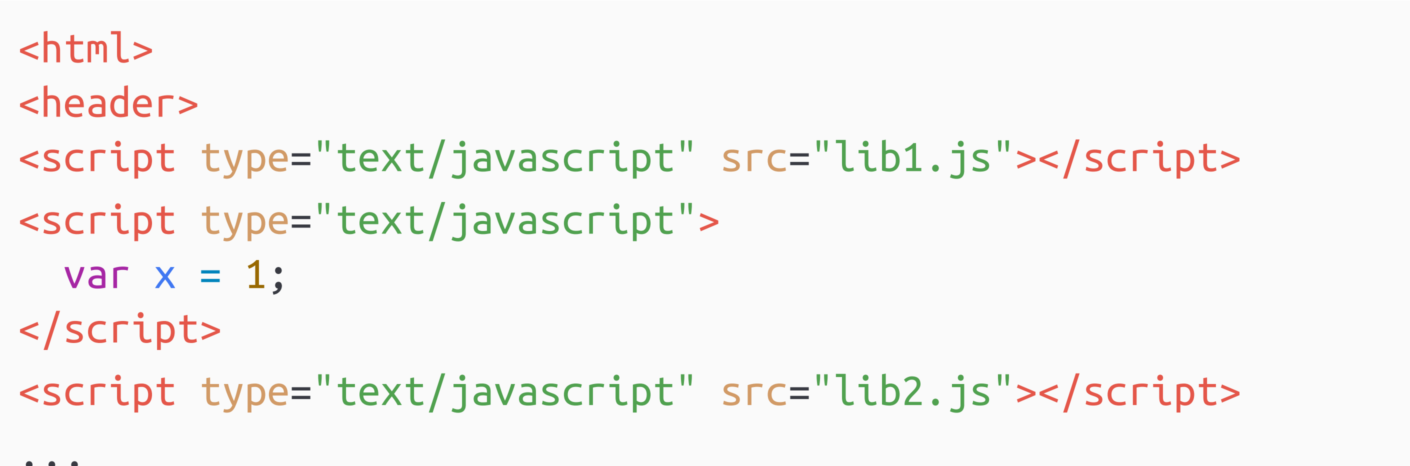
Several technical differences to OSGi (e.g., visibility vs access protection, handling of diamond problem)

```
module A {
     exports org.example.foo;
     exports org.example.bar;
}
module B {
     require A;
}
```

Toward JavaScript Modules

Traditionally no module concept, import into flat namespace

Creating own namespaces with closures/module pattern



```
<html>
<header>
<script type="text/javascript" src="lib1.js"></script>
<script type="text/javascript">
  var x = 1;
</script>
<script type="text/javascript" src="lib2.js"></script>
...
```

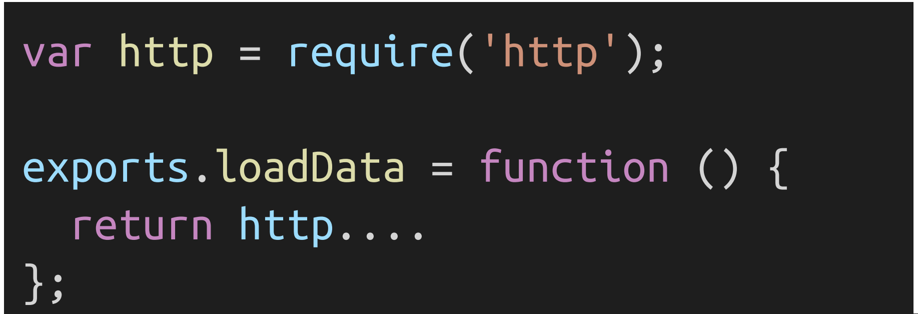
The Module Pattern

```
<html>
<header>
<script type="text/javascript" src="lib1.js"></script>
<script type="text/javascript">
  const m1 = (function () {
    const export = {}
    const x = 1;
    export.x = x;
    return export;
  }());
</script>
<script type="text/javascript" src="lib2.js"></script>
...
```

Node.js Modules (CommonJS)

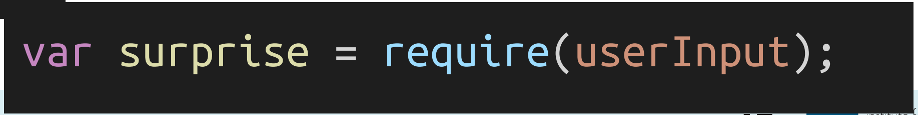
Function `require()` to load other module, dynamic lookup in search path

Module: JavaScript file, can write to export object



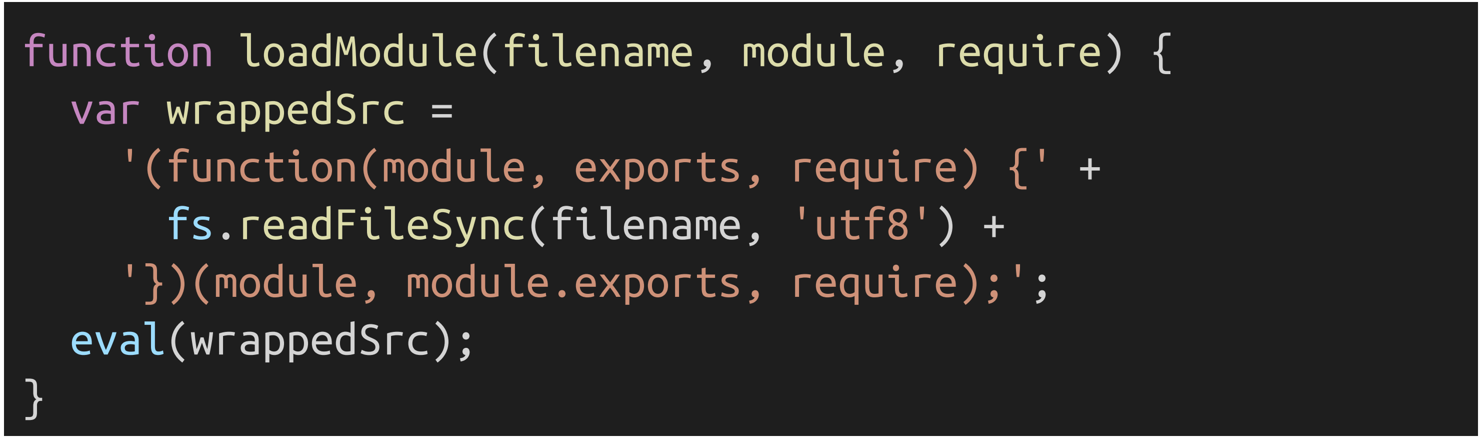
```
var http = require('http');

exports.loadData = function () {
  return http....
};
```



```
var surprise = require(userInput);
```

Node uses Module Pattern Internally



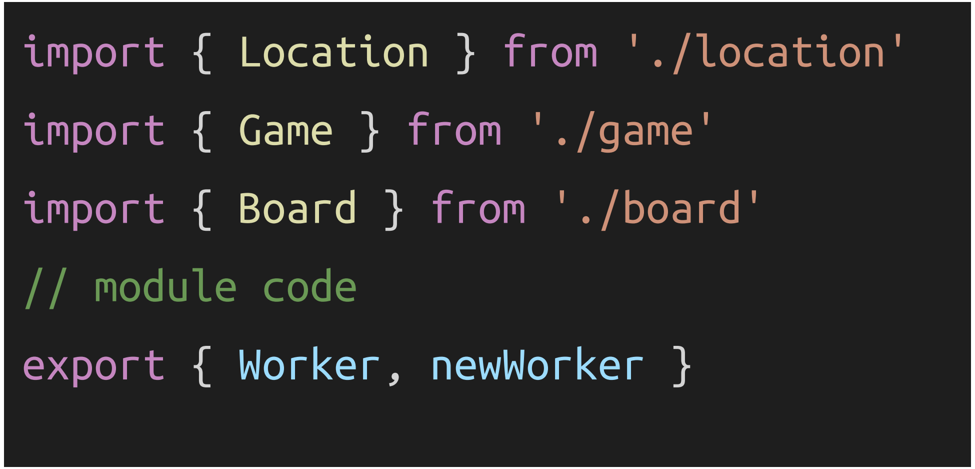
```
function loadModule(filename, module, require) {
  var wrappedSrc =
    '(function(module, exports, require) {' +
      fs.readFileSync(filename, 'utf8') +
    '})(module, module.exports, require);';
  eval(wrappedSrc);
}
```

ES2015 Modules

Syntax extension for modules (instead of module pattern)

Explicit imports / exports

Static import names (like Java), supports better reasoning by tools



```
import { Location } from './location'
import { Game } from './game'
import { Board } from './board'
// module code
export { Worker, newWorker }
```

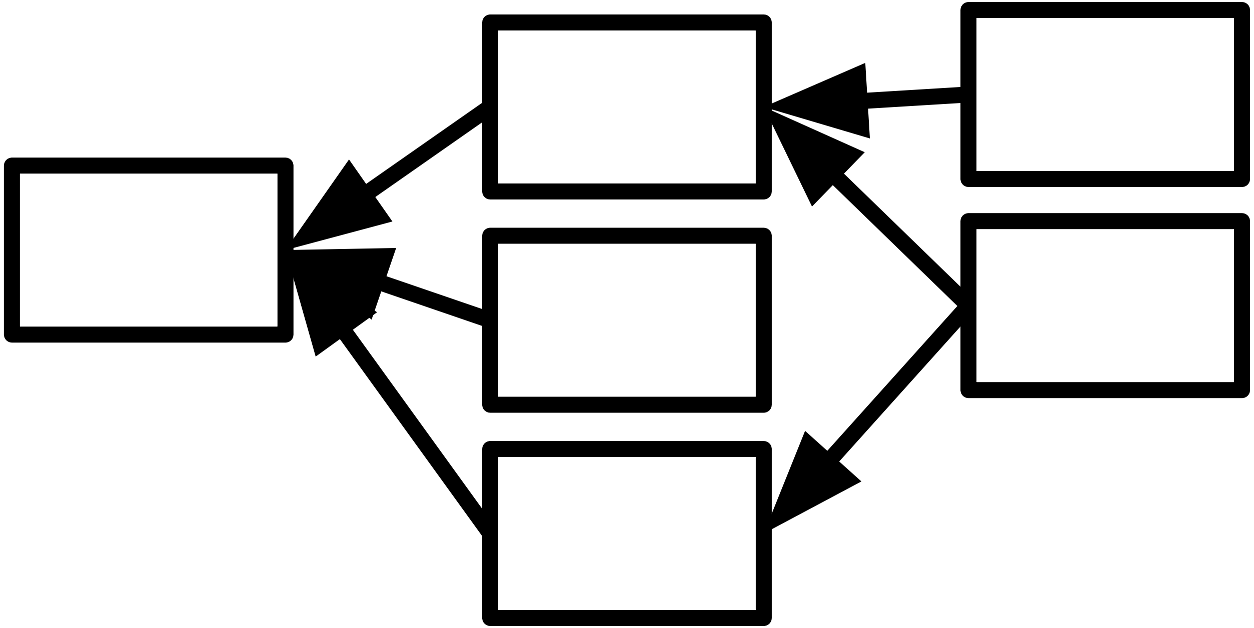
JavaScript Modules and Packages

Modules always decide what to export (values, functions, classes, …) -- everything else only visible in module

Directory structure only used for address in import

Packages typically have one or more modules
and a name and version

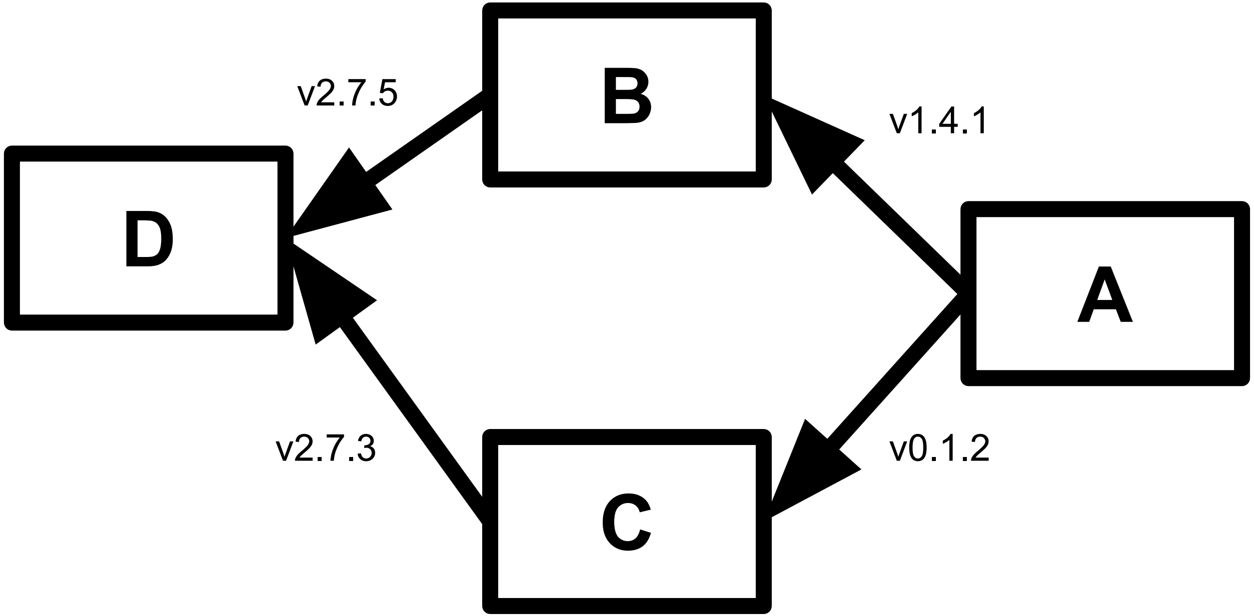
Dependency Graphs



Acyclic

Versioned dependency edges

The Diamond Problem



What now?

Summary: Modules

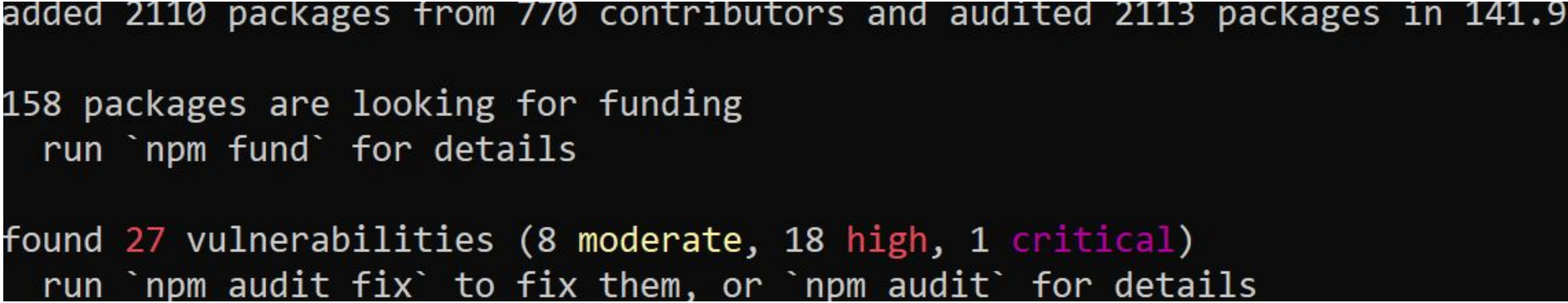
Encapsulation at Scale

Decide which of many classes or packages to expose

Building a dependency graph between modules

Cost of Dependencies

Recall: Ever looked at NPM Install’s output?

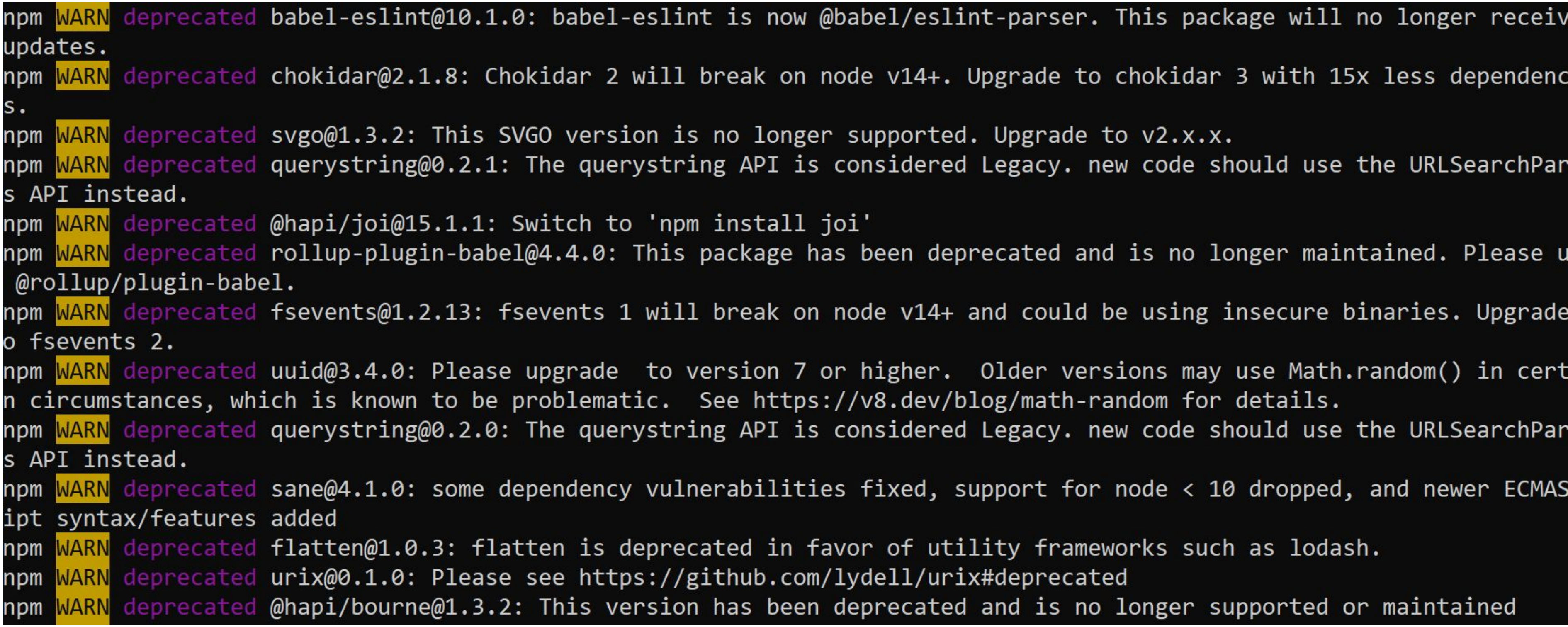


```
added 2110 packages from 770 contributors and audited 2113 packages in 141.9

158 packages are looking for funding
  run `npm fund` for details

found 27 vulnerabilities (8 moderate, 18 high, 1 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
```

Recall: Ever looked at NPM Install's output?



```
npm WARN deprecated babel-eslint@10.1.0: babel-eslint is now @babel/eslint-parser. This package will no longer receiv
updates.
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependenc
s.
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.
npm WARN deprecated querystring@0.2.1: The querystring API is considered Legacy. new code should use the URLSearchPar
s API instead.
npm WARN deprecated @hapi/joi@15.1.1: Switch to 'npm install joi'
npm WARN deprecated rollup-plugin-babel@4.4.0: This package has been deprecated and is no longer maintained. Please u
 @rollup/plugin-babel.
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade
o fsevents 2.
npm WARN deprecated uuid@3.4.0: Please upgrade  to version 7 or higher.  Older versions may use Math.random() in cert
n circumstances, which is known to be problematic.  See https://v8.dev/blog/math-random for details.
npm WARN deprecated querystring@0.2.0: The querystring API is considered Legacy. new code should use the URLSearchPar
s API instead.
npm WARN deprecated sane@4.1.0: some dependency vulnerabilities fixed, support for node < 10 dropped, and newer ECMAS
ipt syntax/features added
npm WARN deprecated flatten@1.0.3: flatten is deprecated in favor of utility frameworks such as lodash.
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated @hapi/bourne@1.3.2: This version has been deprecated and is no longer supported or maintained
```

Monitoring for Vulnerabilities

Dependency manager helps knowing what dependencies are used (“bill of materials”)

Various tools scan for known vulnerabilities -- use them

Have a process

Many false positive alerts, not exploitable

Recommended reading:
https://republicans-oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf

Supply Chain Attacks more common

Intentionally injecting attacks in packages

- Typosquatting: expres
- Malicious updates: us-parser-js

Review all packages? All updates?

Sandbox applications? Sandbox packages?

Using a Dead Dependency?

No more support?

No fixes to bugs and vulnerabilities?

What now?

Open Source Health and Sustainability

Predict which packages will be maintained next year?

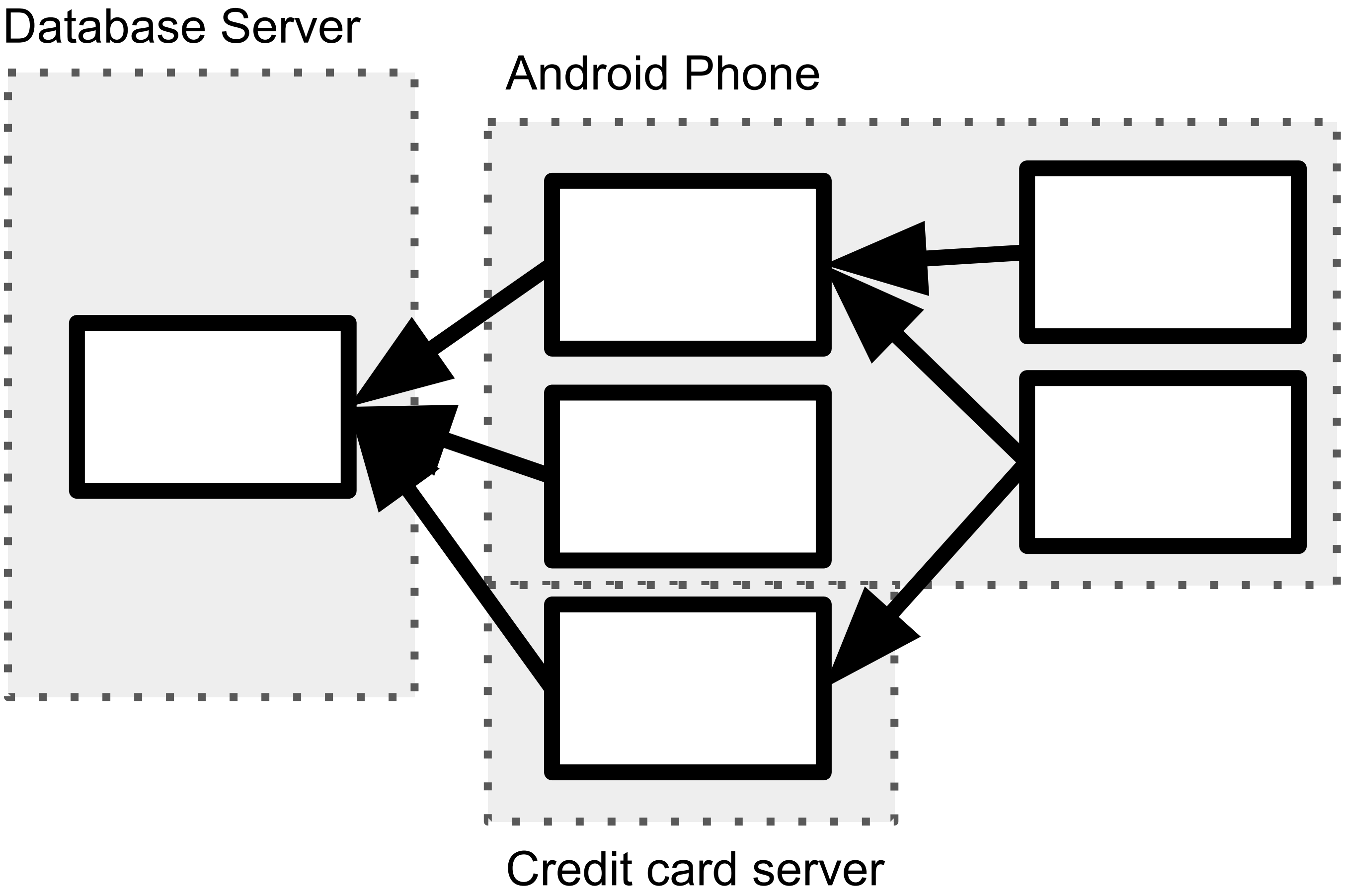
- Indicators?

Motivation of maintainers?

Who funds open source?

Commercial dependencies? Commercial support?

Distributed Modules


Database Server
Android Phone
Credit card server

Distributed Systems

Remote procedure calls instead of function calls

Typically REST API to URL

Benefits? Drawbacks?

REST APIs

REST (or RESTful – representational state transfer) API

API of a web service “that conforms to the constraints of the REST architectural style.”

Uniform interface over HTTP requests

- Send parameters to URL, receive data (JSON, XML common)
- Stateless: Each request is self-contained

Language independent, distributed

REST API Design

All the same design principles apply

Document the API, input/output formats and error conditions!

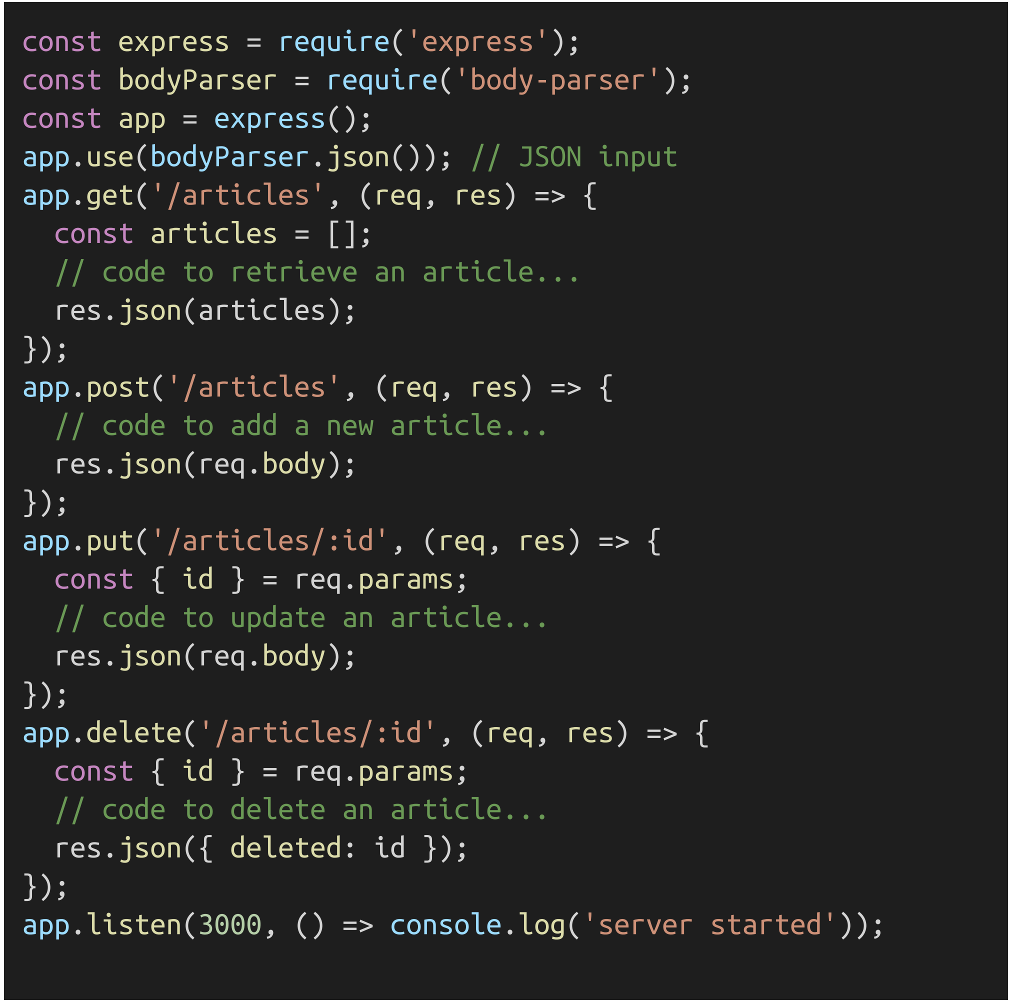
CRUD Operations

Path correspond to nouns, not verbs, nesting common:

- `/articles`, `/state`, `/game` `/articles/:id/comments`

GET (receive), POST (submit new), PUT (update), and DELETE requests sent to those paths

Parameters for filtering, searching, sorting, e.g., `/articles?sort=date`



```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json()); // JSON input
app.get('/articles', (req, res) => {
  const articles = [];
  // code to retrieve an article...
  res.json(articles);
});
app.post('/articles', (req, res) => {
  // code to add a new article...
  res.json(req.body);
});
app.put('/articles/:id', (req, res) => {
  const { id } = req.params;
  // code to update an article...
  res.json(req.body);
});
app.delete('/articles/:id', (req, res) => {
  const { id } = req.params;
  // code to delete an article...
  res.json({ deleted: id });
});
app.listen(3000, () => console.log('server started'));
```

REST Specifics

- JSON common for data exchange: Define and validate schema -- many libraries help
- Return HTTP standard errors (400, 401, 403, 500, …)
- Security mechanism through SSL/TLS and other common practices
- Caching common
- Consider versioning APIs `/v1/articles`, `/v2/articles`

Excursion: Testing in Distributed Systems

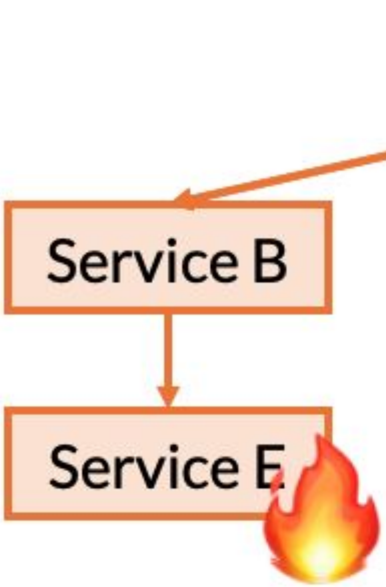
REST API Calls and Testing

Test happy path

Test also error behavior!

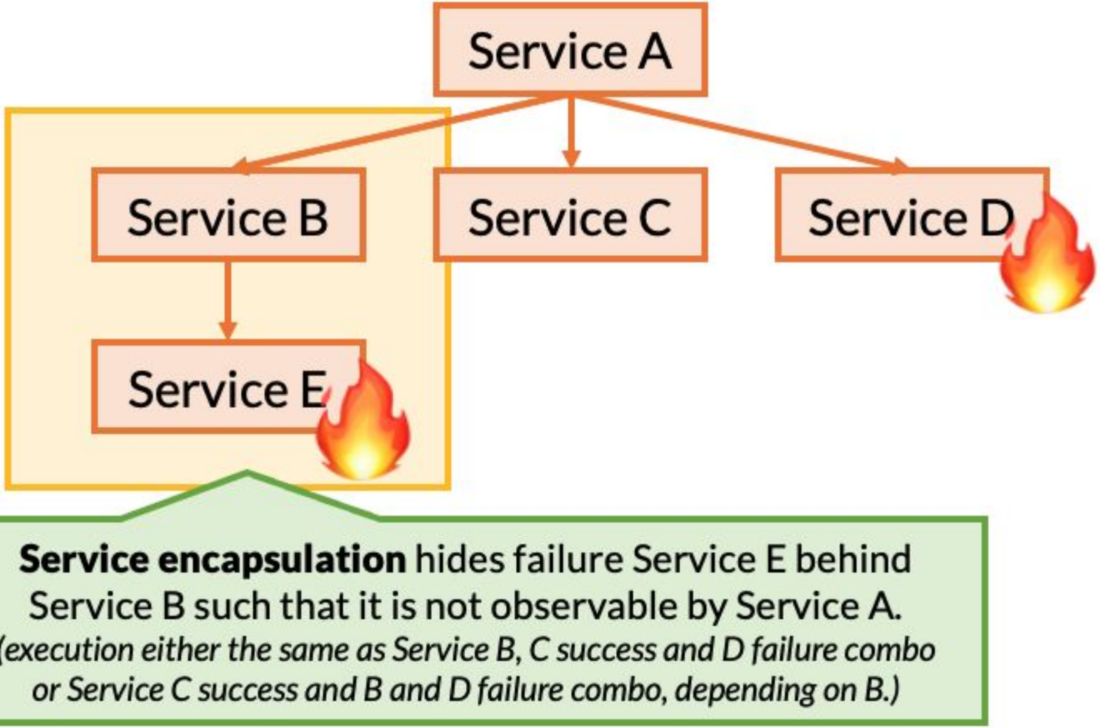
- Correct timeout handling? Correct retry when connection down?
- Invalid response detected?
- Graceful degradation?

Need to understand possible error behavior first



http://christophermeiklejohn.com/filibuster/2021/10/14/filibuster-4.html

Handle Errors Locally



Service encapsulation hides failure Service E behind Service B such that it is not observable by Service A.
(execution either the same as Service B, C success and D failure combo or Service C success and B and D failure combo, depending on B.)

http://christophermeiklejohn.com/filibuster/2021/10/14/filibuster-4.html

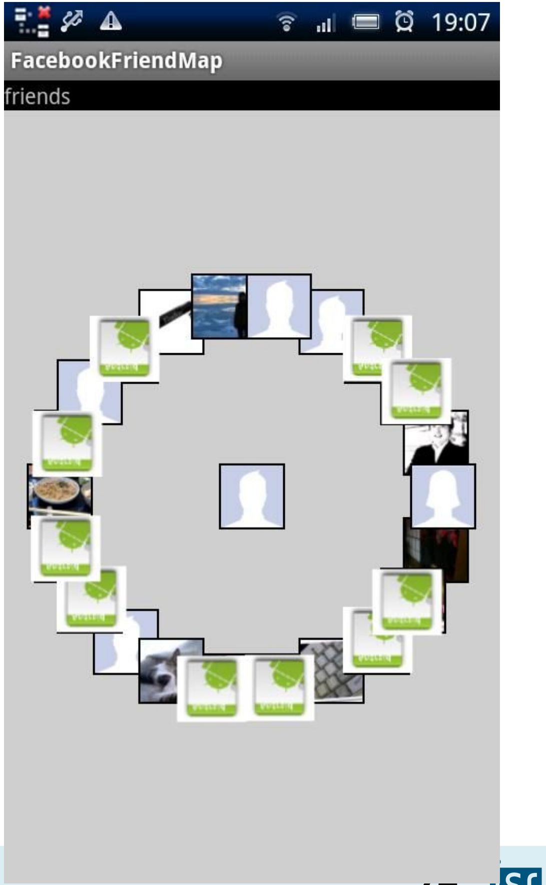
How to test?

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

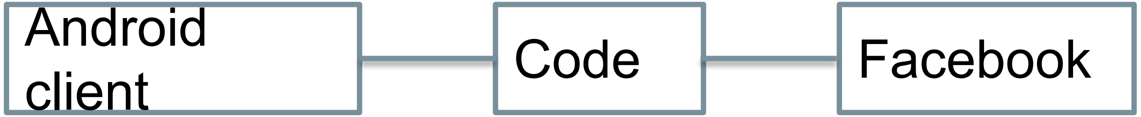
Return of the Test Doubles!

Recall: Facebook Example

- 3rd party Facebook apps
- Android user interface
- Backend uses Facebook data

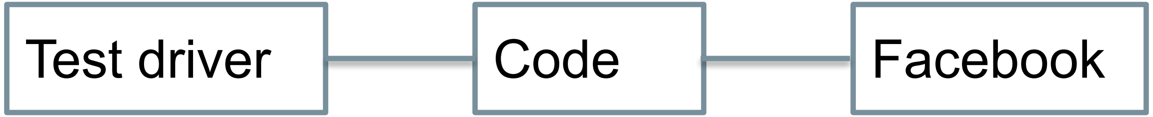


Testing in real environments



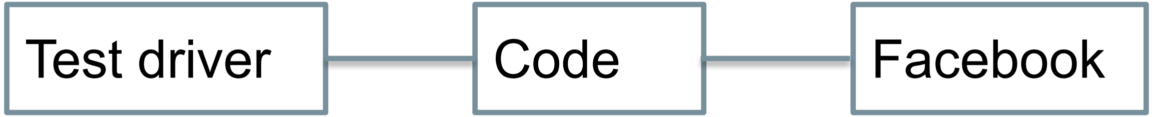
```
void buttonClicked() {
   render(getFriends());
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new FacebookAPI(c);
   List<Node> persons = api.getFriends("john");
   for (Node person1 : persons) {
      ...
   }
   return result;
}
```

Eliminating Android dependency



```
@Test void testGetFriends() {
   assert getFriends() == ...;
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new FacebookAPI(c);
   List<Node> persons = api.getFriends("john");
   for (Node person1 : persons) {
      ...
   }
   return result;
}
```

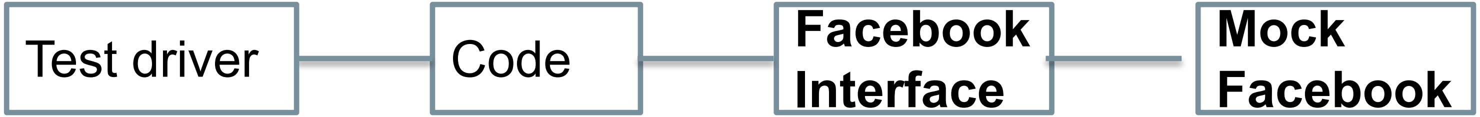
Eliminating the Remote Service Dependency



```
@Test void testGetFriends() {
   assert getFriends() == ...;
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookAPI api = new FacebookAPI(c);
   List<Node> persons = api.getFriends("john");
   for (Node person1 : persons) {
      ...
   }
   return result;
}
```

Replace by Double

Introducing a Double (Stub)


Test driver

Code

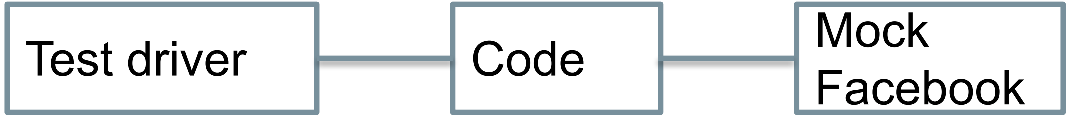
Facebook Interface

Mock Facebook

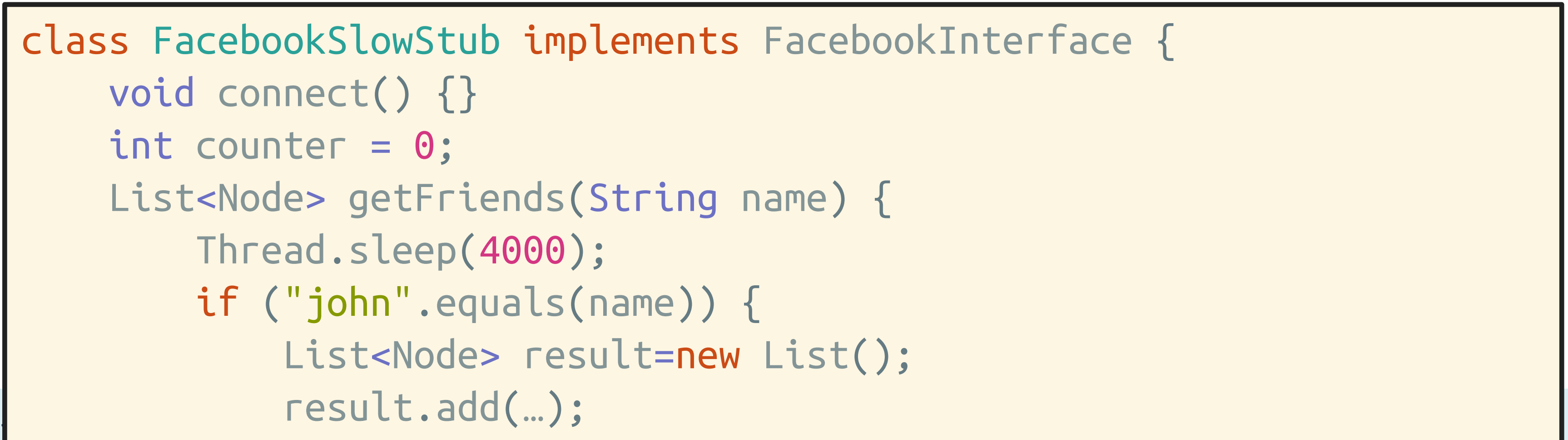
```
@Test void testGetFriends() {
   assert getFriends() == …;
}
List<Friend> getFriends() {
   Connection c = http.getConnection();
   FacebookInterface api = new FacebookStub(c);
   List<Node> person
   for (Node person1
       for (Node per
         …
       }
   }
```

```
class FacebookStub implements FacebookInterface {
    void connect() {}
    List<Node> getFriends(String name) {
        if ("john".equals(name)) {
            List<Node> result=new List();
            result.add(…);
```

Fault injection

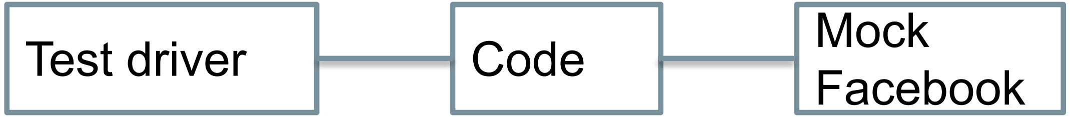


- Mocks can emulate failures such as timeouts
- Allows you to verify the robustness of system



```
class FacebookSlowStub implements FacebookInterface {
    void connect() {}
    int counter = 0;
    List<Node> getFriends(String name) {
        Thread.sleep(4000);
        if ("john".equals(name)) {
            List<Node> result=new List();
            result.add(…);
```

Fault injection

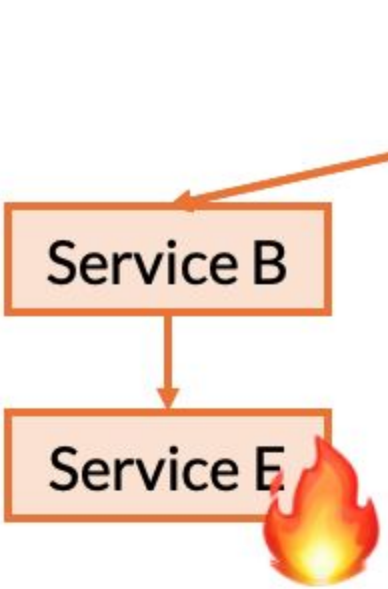


```
class FacebookErrorStub implements FacebookInterface {
    void connect() {}
    int counter = 0;
    List<Node> getFriends(String name) {
        counter++;
        if (counter % 3 == 0)
            throw new SocketException("Network is unreachable");
        if ("john".equals(name)) {
            List<Node> result=new List();
            result.add(…);
            return result;
```

Chaos Engineering

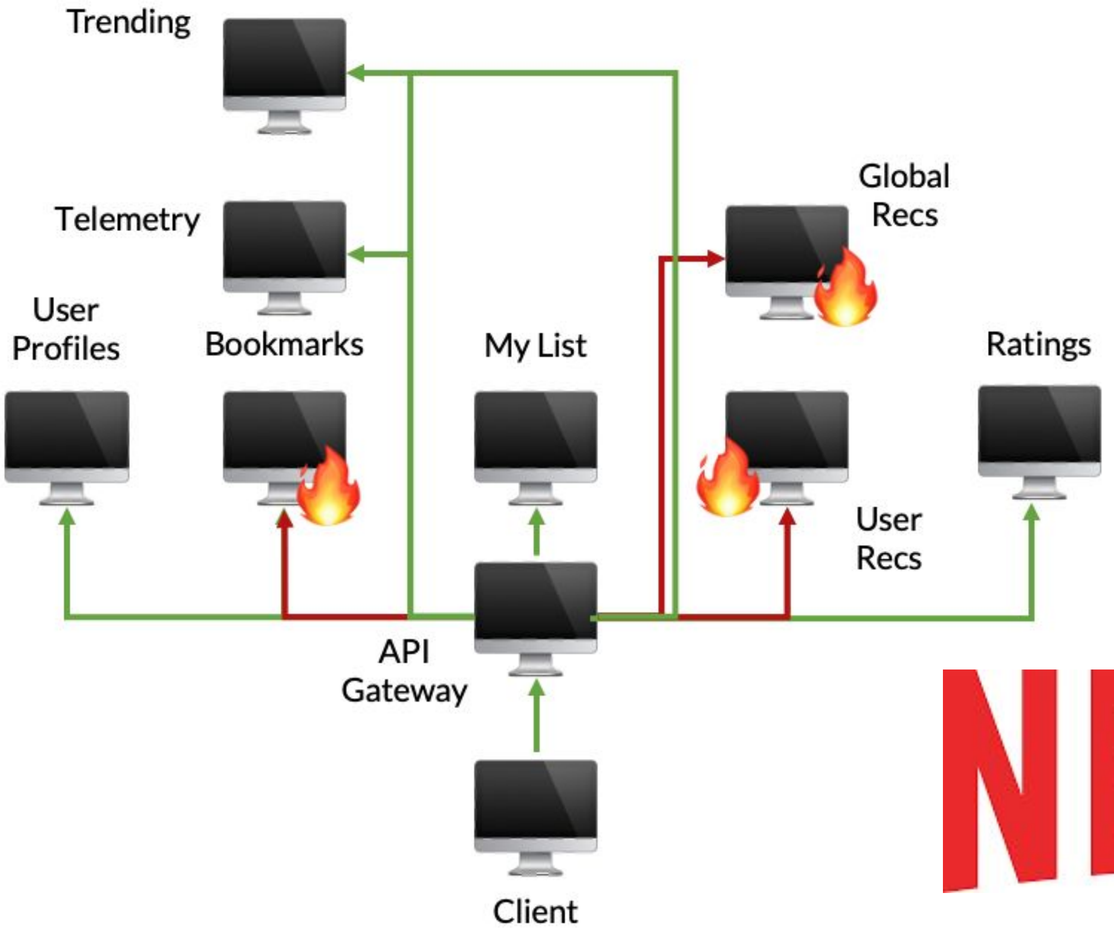
Experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production

NETFLIX



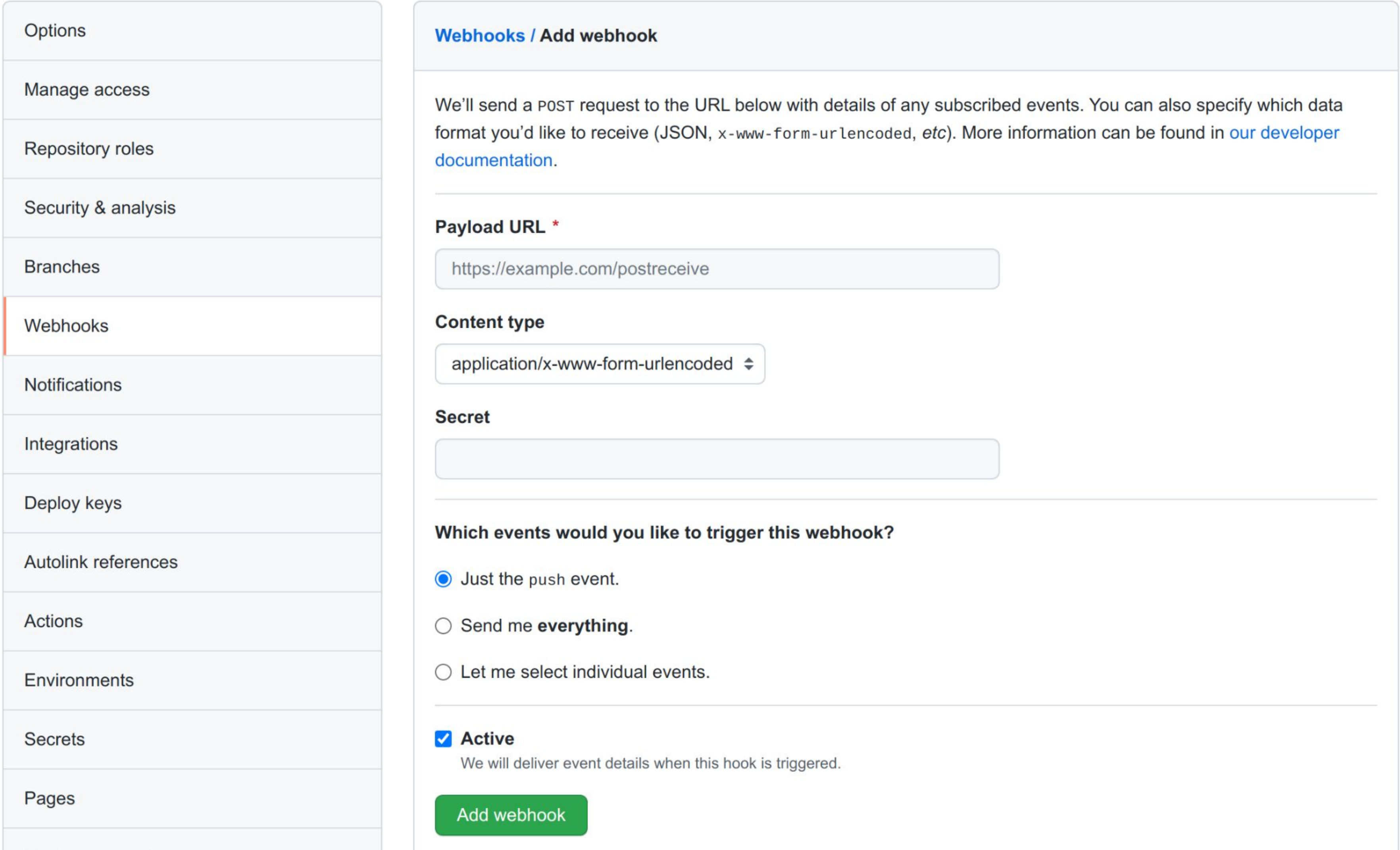
http://christophermeiklejohn.com/filibuster/2021/10/14/filibuster-4.html

Distributed Event-Based System



NETFLIX

http://christophermeiklejohn.com/filibuster/2021/10/14/filibuster-4.html



Webhooks / **Add webhook**

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *

https://example.com/postreceive

Content type

application/x-www-form-urlencoded

Secret

Which events would you like to trigger this webhook?

● Just the `push` event.

○ Send me **everything**.

○ Let me select individual events.

☑ **Active**

We will deliver event details when this hook is triggered.

Push vs Pull: RPC vs Callbacks

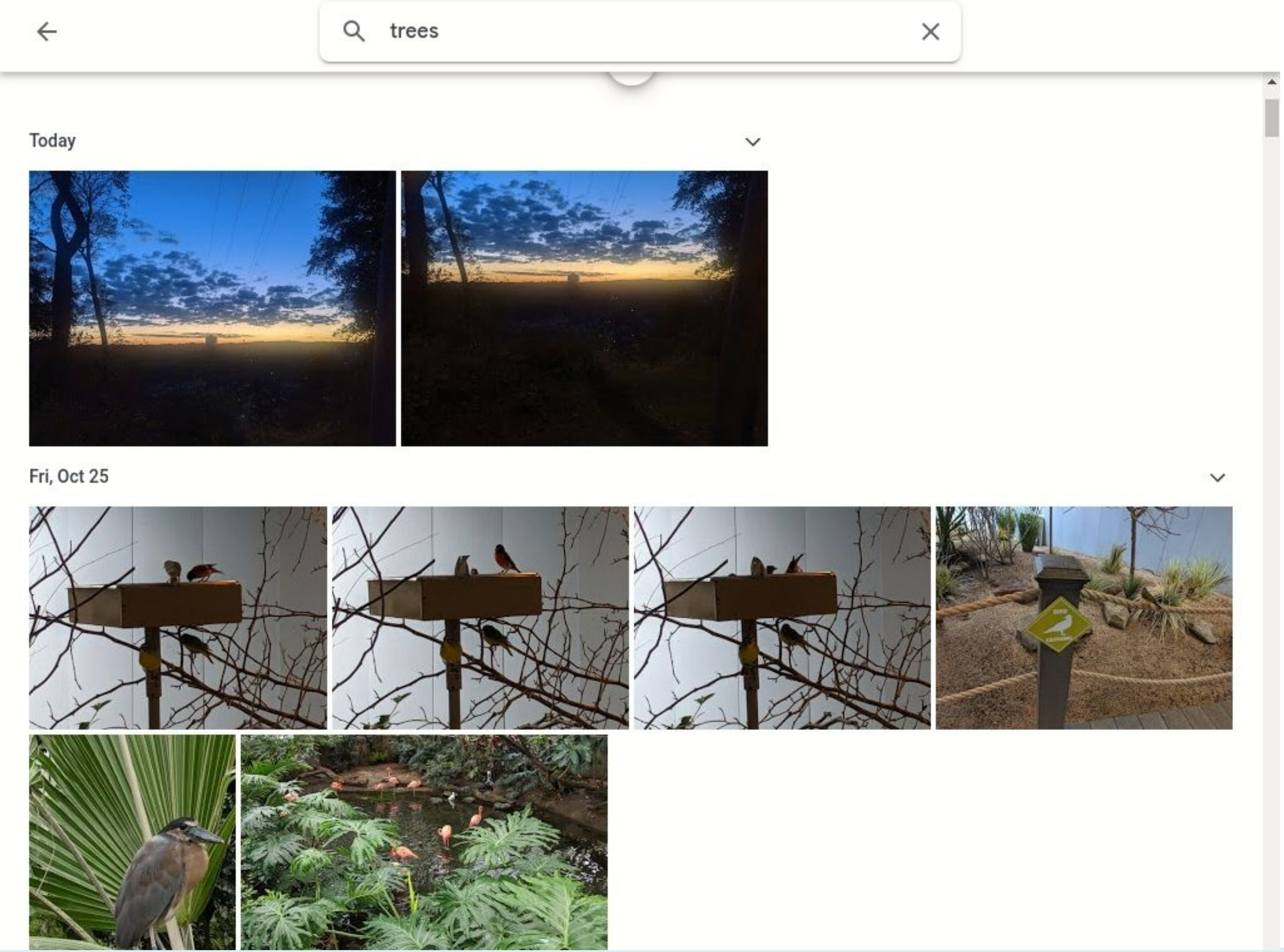
Both libraries and frameworks possible with RPC

- Netflix: Gateway calls and orchestrates services (pull; Strategy Pattern)
- GitHub WebHooks: GitHub pushes events to custom URL (Observer Pattern)

Reactive Programming and Event/Stream Processing

Stream processing: Distributed system design based on event queuing and processing

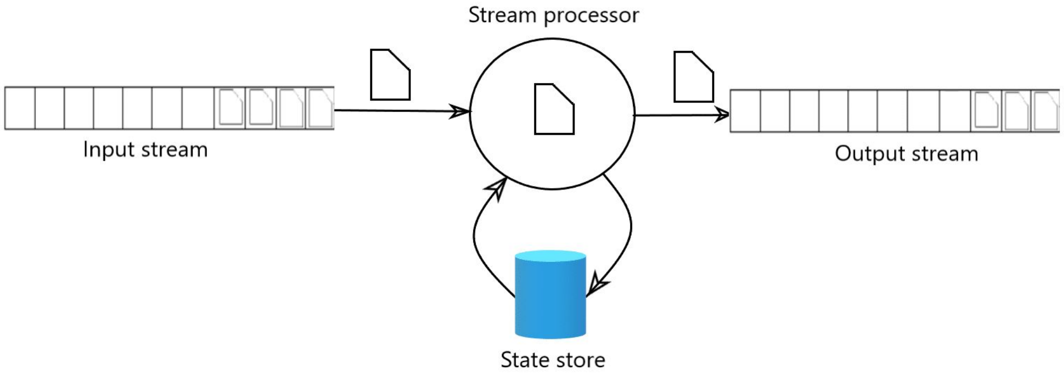
Example:
Tagging of many images
Indexing for search

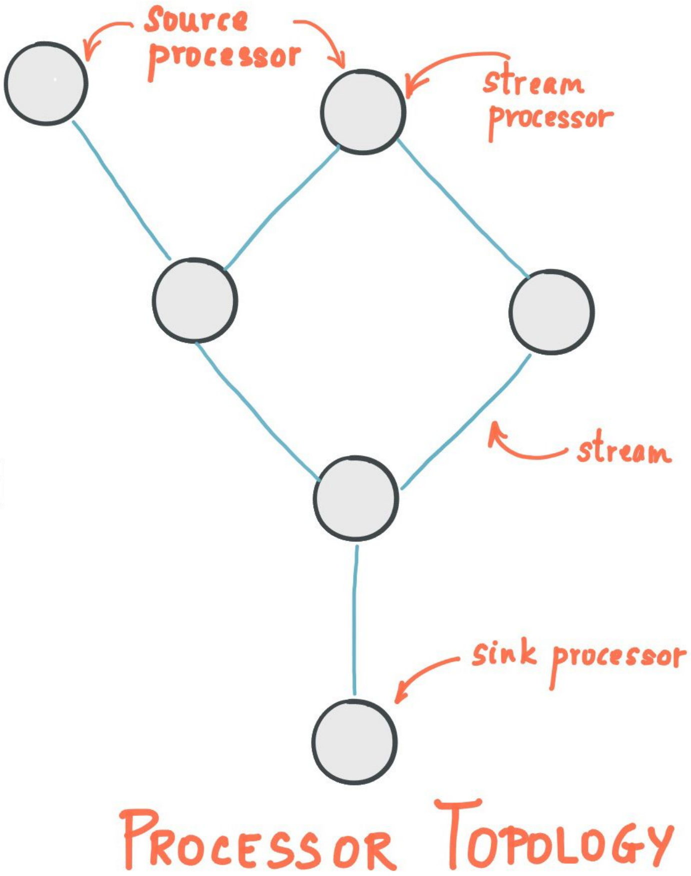


FIXME: kill this Recall: RxJava

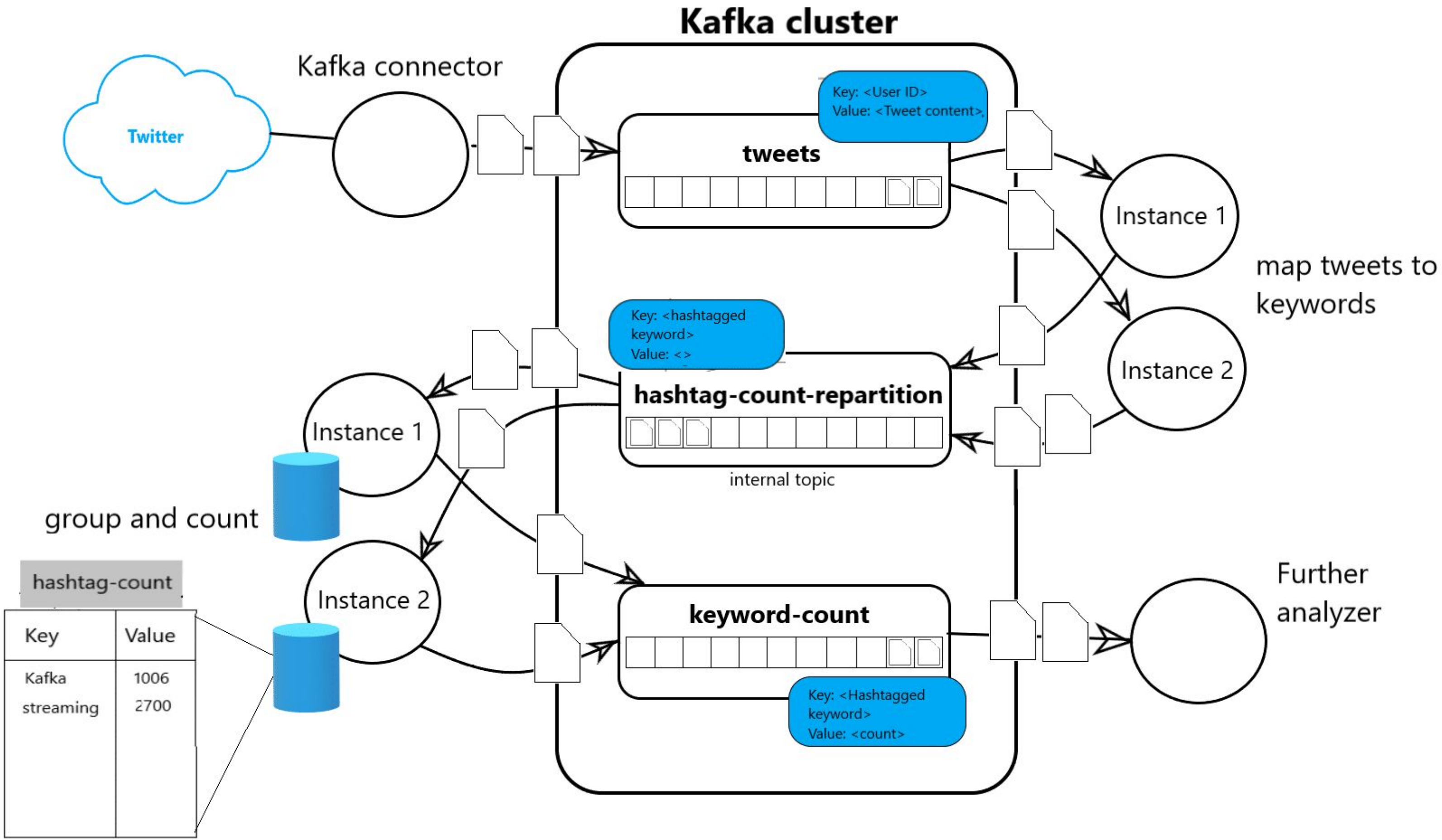
```
PublishSubject<Integer> x = PublishSubject.create();
PublishSubject<Integer> y = PublishSubject.create();
Observable<Integer> z = Observable.combineLatest(x, y,
(a,b)->a+b);
z.subscribe(System.out::println);
x.onNext(3);
y.onNext(5);
x.onNext(5);
```

Apache Kafka





https://www.novatec-gmbh.de/en/blog/kafka-101-series-part-2-stream-processing-and-kafka-streams-api/



Key	Value
Kafka	1006
streaming	2700

https://www.novatec-gmbh.de/en/blog/kafka-101-series-part-2-stream-processing-and-kafka-streams-api/

```java
final String topic = "topicName";
final Consumer<String, String> consumer = new KafkaConsumer<>();
consumer.subscribe(Arrays.asList(topic));

try {
     while (true) {
       ConsumerRecords<String, String> records = consumer.poll(100);
       for (ConsumerRecord<String, String> record : records) {
         String key = record.key();
         String value = record.value();
         // process data
       }
     }
} finally {
     consumer.close();
}
```

Kafka Consumer Code Example

Summary

Heavy reliance on dependencies

- Package managers and module systems help organize
- Manage costs and risks of dependencies

Modularly organize systems at scale

- Modules
- Distributed systems
- Microservices
- Event-based systems / stream processing

Testing with Stubs and Chaos Engineering