Principles of Software Construction: Objects, Design, and Concurrency

Organizing Systems at Scale: Modules, Services, Architectures

Christian Kästner Vincent Hellendoorn

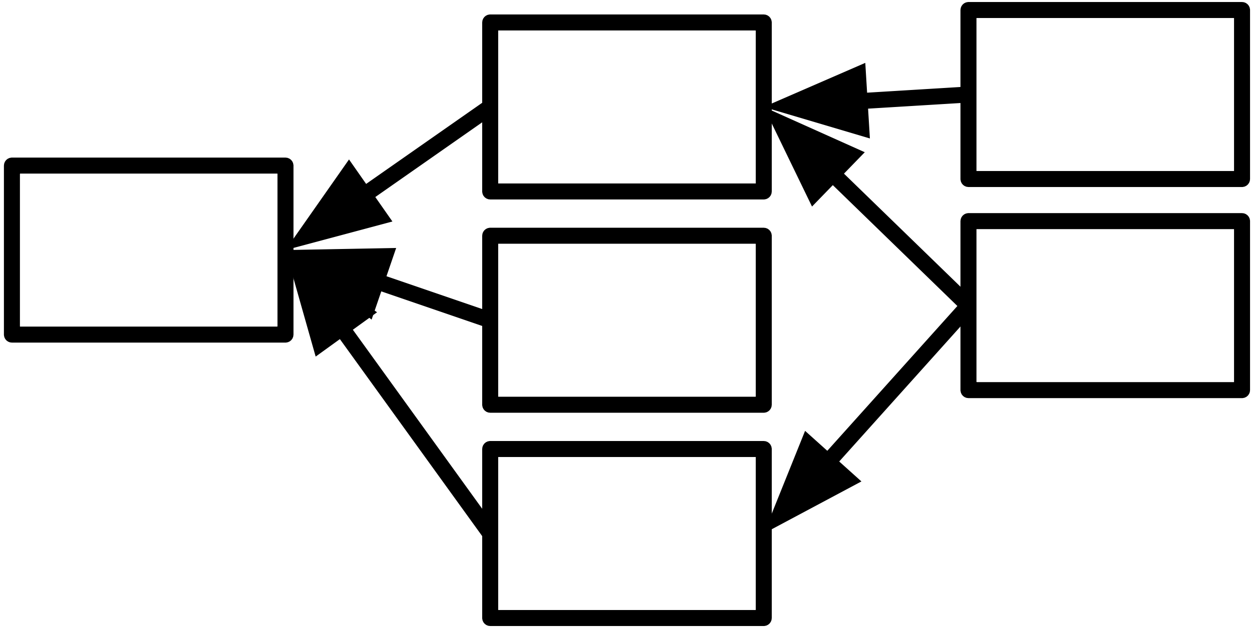
Carnegie Mellon University
School of Computer Science

institute for SOFTWARE RESEARCH

Where we are

	Small scale: One/few objects	*Mid scale:* Many objects	*Large scale:* Subsystems
Design for understanding change/ext. reuse robustness ...	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

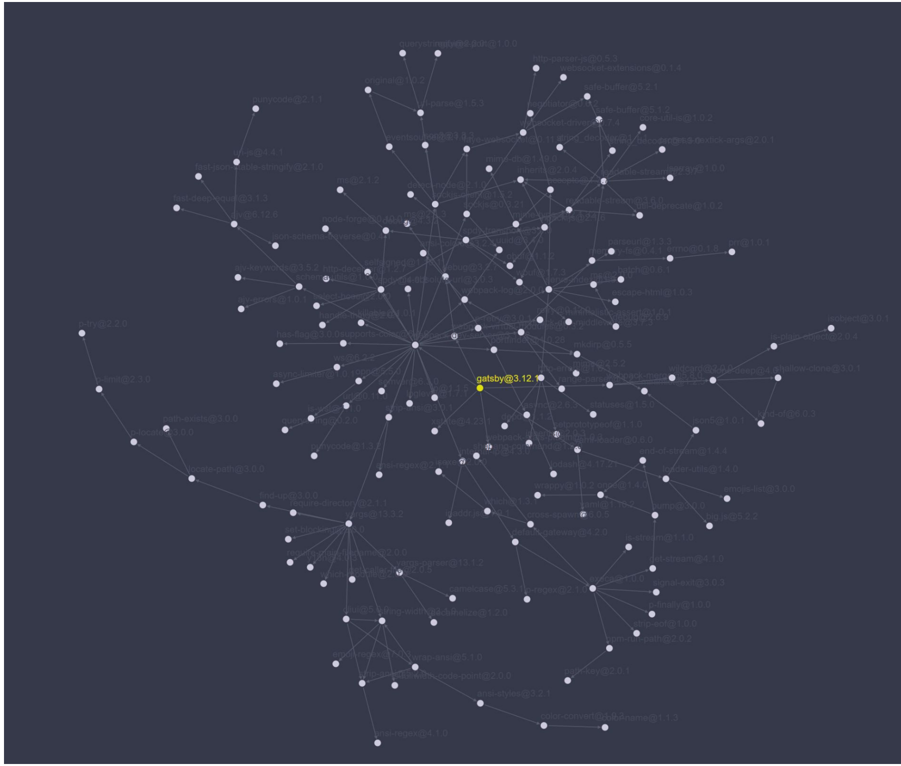
Software Supply Chains / Software Ecosystems



Recall: Modern Software Engineering

Nobody wants to write a million lines of code.

- Instead, you use libraries
  - E.g., import Android => +12M LOC
  - You don't write most of the code you use
    - And why would you want to?
- And your libraries use libraries
  - Et cetera
  - https://npm.anvaka.com/#/view/2d/gatsby



Libraries.io

Search open source packages, frameworks and tools... | Search

Libraries.io monitors **6,216,328** open source packages across **32** different package managers, so you don't have to. Find out more

Discover new software

Search 6.22M packages by license, language or keyword, or explore new, trending or popular packages.

Explore

Monitor your dependencies

Stay up to date with notifications of updates, license incompatibilities or deleted dependencies.

Login

Maintain your OSS project

Understand your users and make informed decisions about features with usage and version data.

Login

Use Libraries.io data

Use Libraries.io data in your applications, services or research. Use our API to stay up to date.

Documentation

Supported Package Managers

npm 2.1M Packages

Go 444K Packages

Maven 443K Packages

PyPI 405K Packages

Packagist

NuGet

Rubygems

CocoaPods

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 }
```

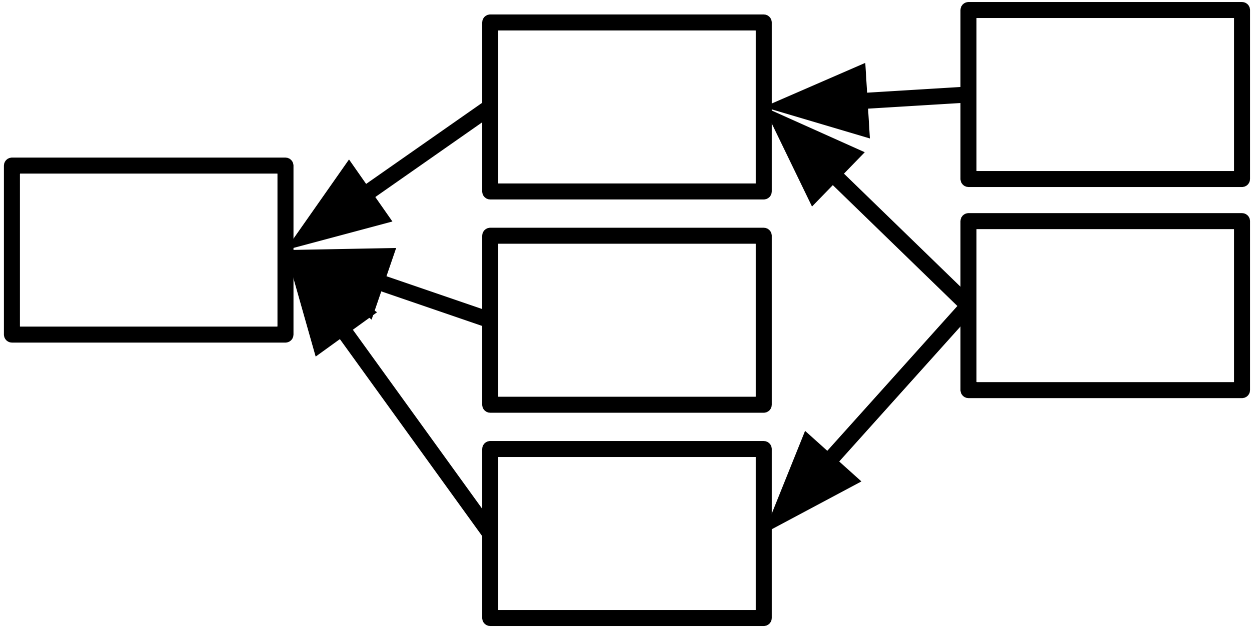
JavaScript Modules and Packages

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

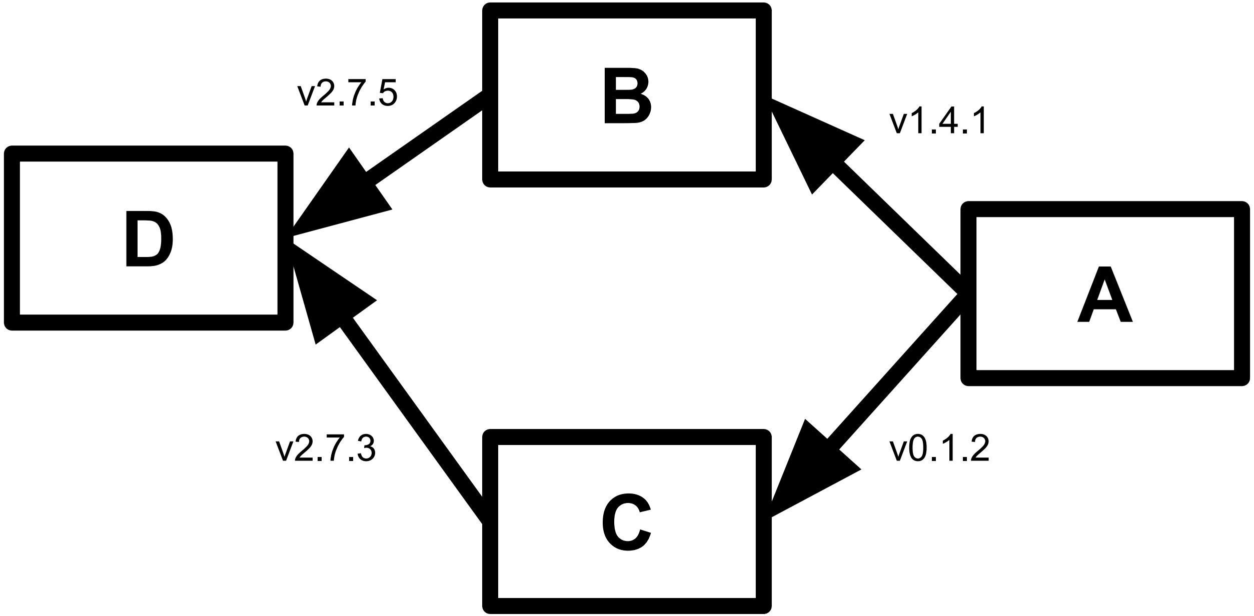
Dependency Graphs



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

EQUIFAX

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?

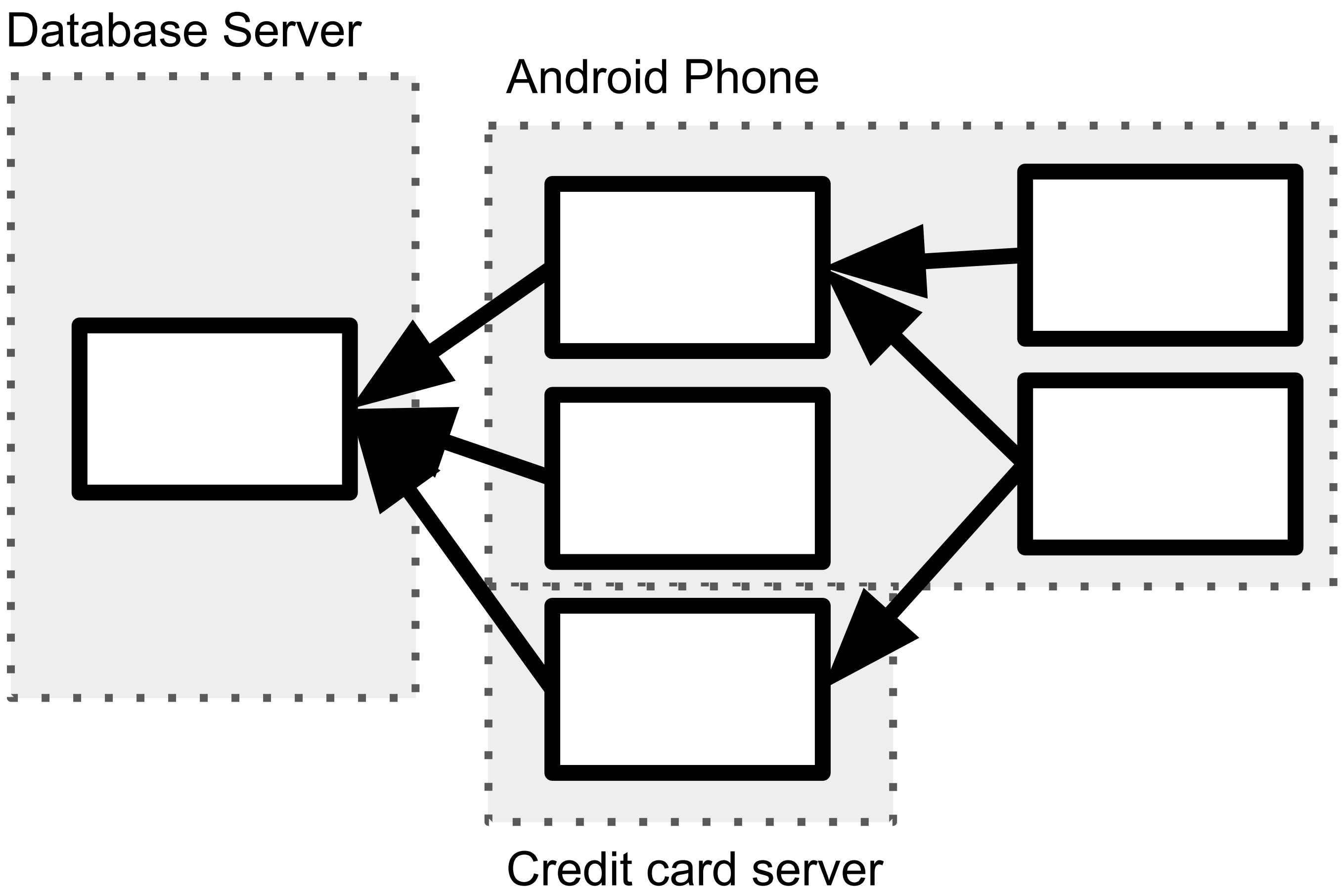
- Indicators?

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?

Distributed System Benefits

Scalability

Very strong encapsulation (only APIs public)

Computation beyond local resources

Independent deployment, operations, and evolution

Also multiple containers on single system

Pay per transaction / storage / use

Distributed System Problems

Distributed system problem!

All kinds of new problem scenarios: Unavailable services, delayed responses, missing responses, out of order responses, network segmentation, clock problems, unannounced changes of service behavior

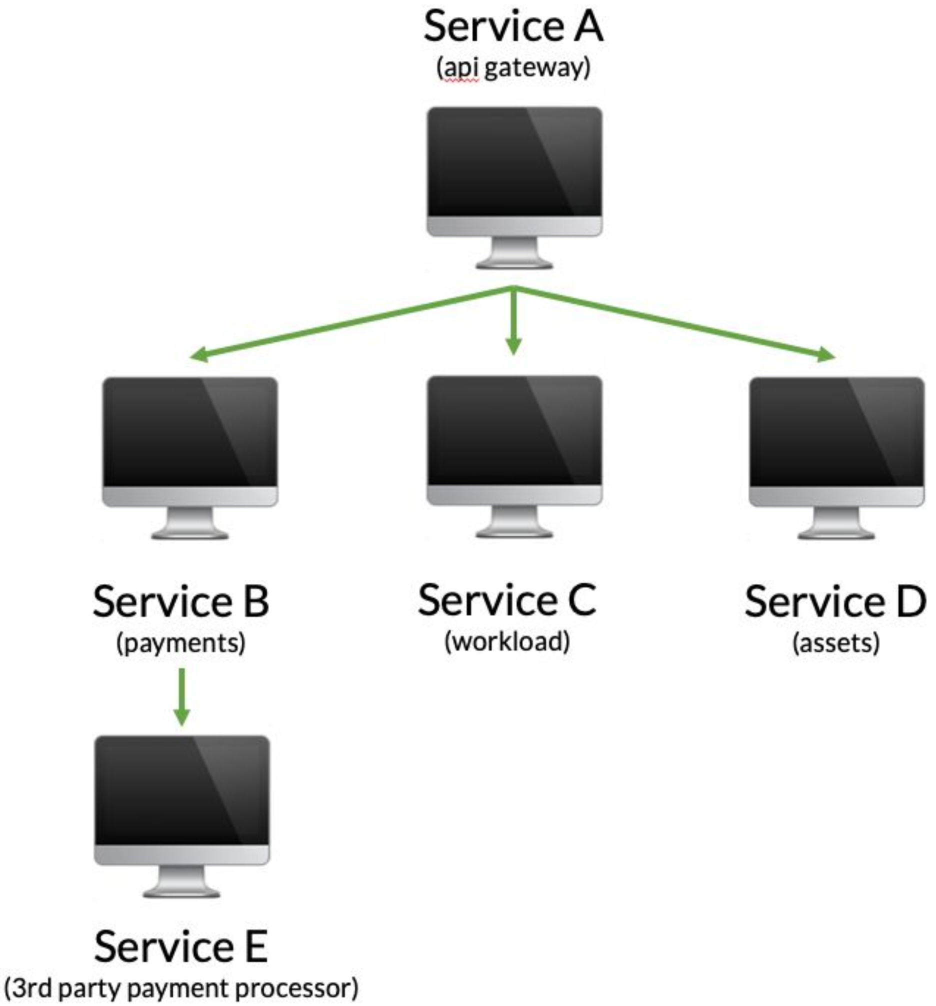
Distributed systems is hard! Do not underestimate! Build on existing abstractions!

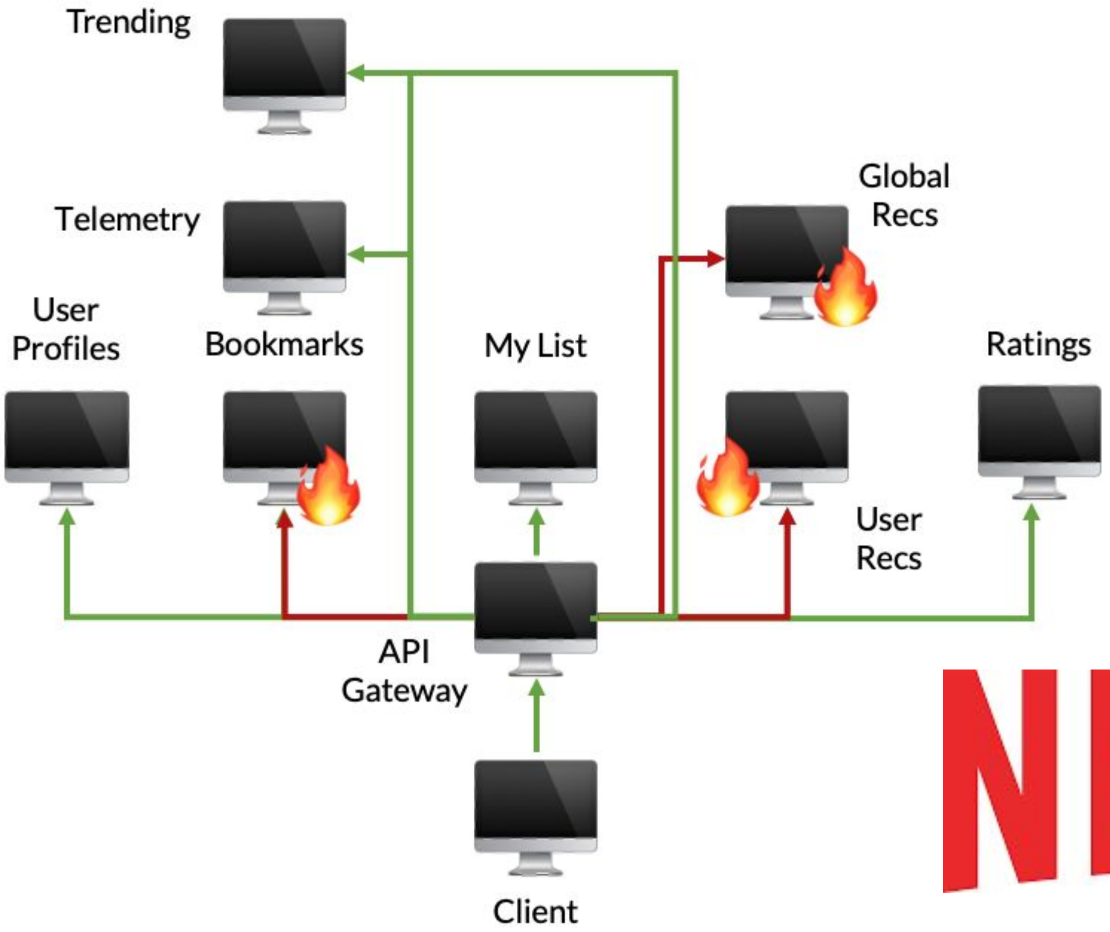
Shifting complexities to the network

Fallacies of distributed computing by Peter Deutsch

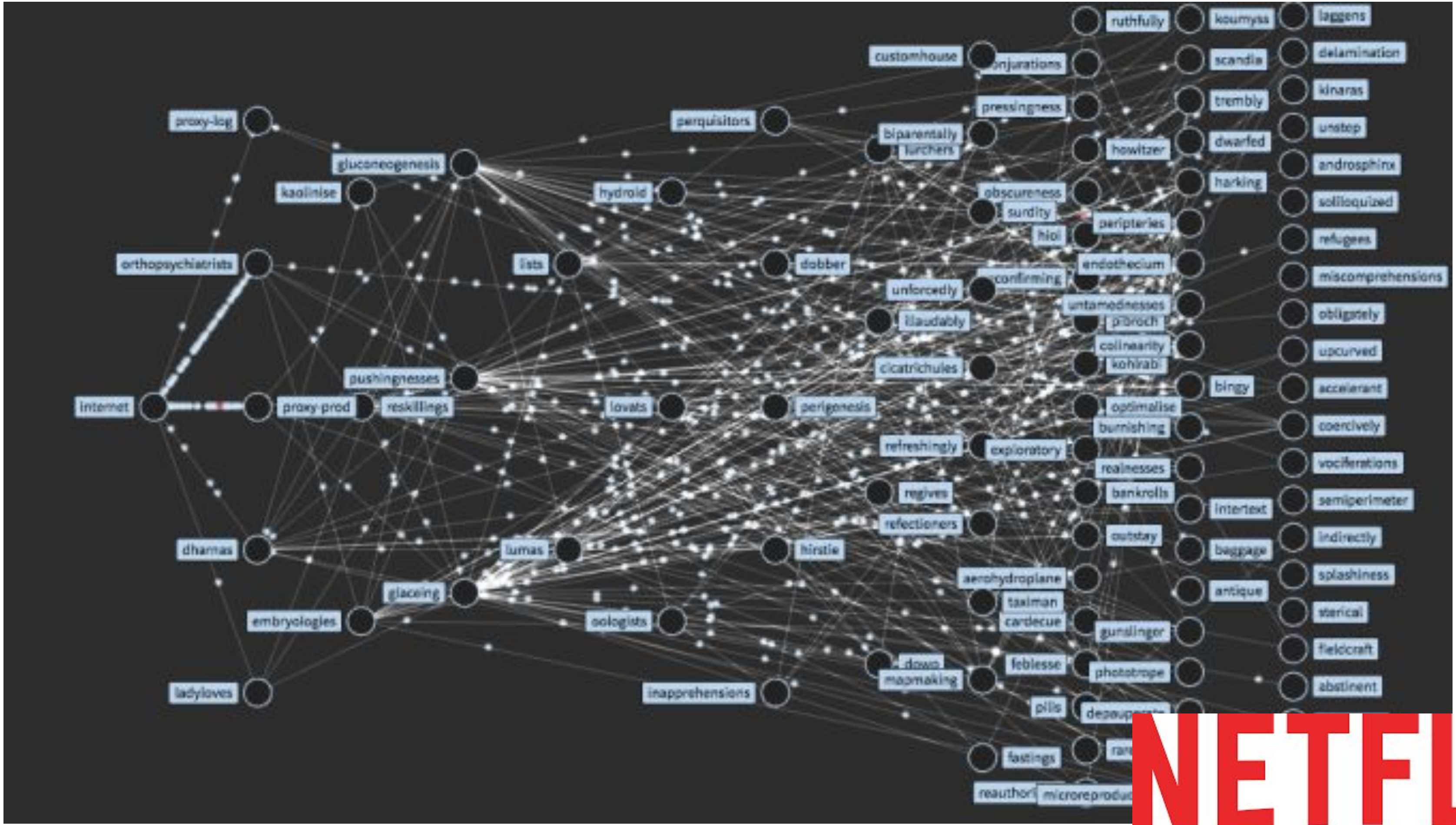
1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.


Microservices
Microservices Everywhere

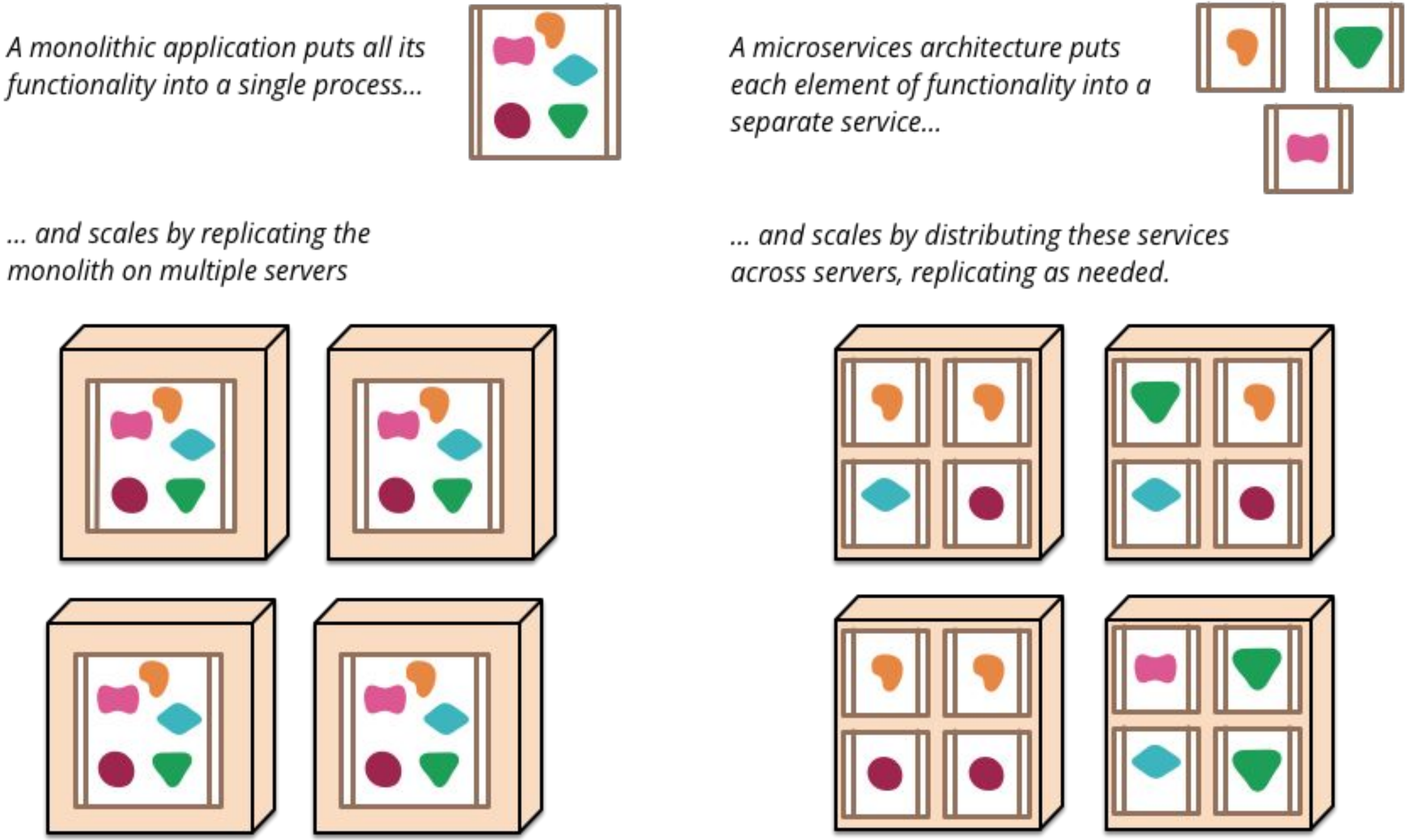

Service A
(api gateway)
Service B
(payments)
Service C
(workload)
Service D
(assets)
Service E
(3rd party payment processor)



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


proxy-log
gluconeogenesis
kaolinise
orthopsychiatrists
internet
proxy-prod
reskillings
pushingnesses
dharmas
embryologies
glaceing
ladyloves
lists
hydroid
perquisitors
lumas
lovats
oologists
inapprehensions
dobber
perigenesis
hirstie
customhouse
biparentally
lurchers
unforcedly
illaudably
cicatrichules
refreshingly
regives
refectioners
mapmaking
fastings
pressingness
obscureness
surdity
hioi
confirming
untamednesses
pibroch
collinearity
kohlrabi
optimalise
burnishing
exploratory
realnesses
bankrolls
outstay
aerohydroplane
taximan
cardecue
gunslinger
feblesse
phototrope
pilis
ruthfully
howitzer
peripteries
endothecium
koumyss
scandia
trembly
dwarfed
harking
bingy
intertext
baggage
antique
laggens
delamination
kinaras
unstop
androsphinx
soliloquized
refugees
miscomprehensions
obligately
upcurved
accelerant
coercively
vociferations
semiperimeter
indirectly
splashiness
sterical
fieldcraft
abstinent

NETFLIX



source: http://martinfowler.com/articles/microservices.html

Microservices

Building applications as suite of small and easy to replace services

- fine grained, one functionality per service
- (sometimes 3-5 classes)
- composable
- easy to develop, test, and understand
- fast (re)start, fault isolation

Modelled around business domain

Interplay of different systems and languages, no commitment to technology stack

Easily deployable and replicable

Embrace automation, embrace faults

Highly observable

Technical Considerations

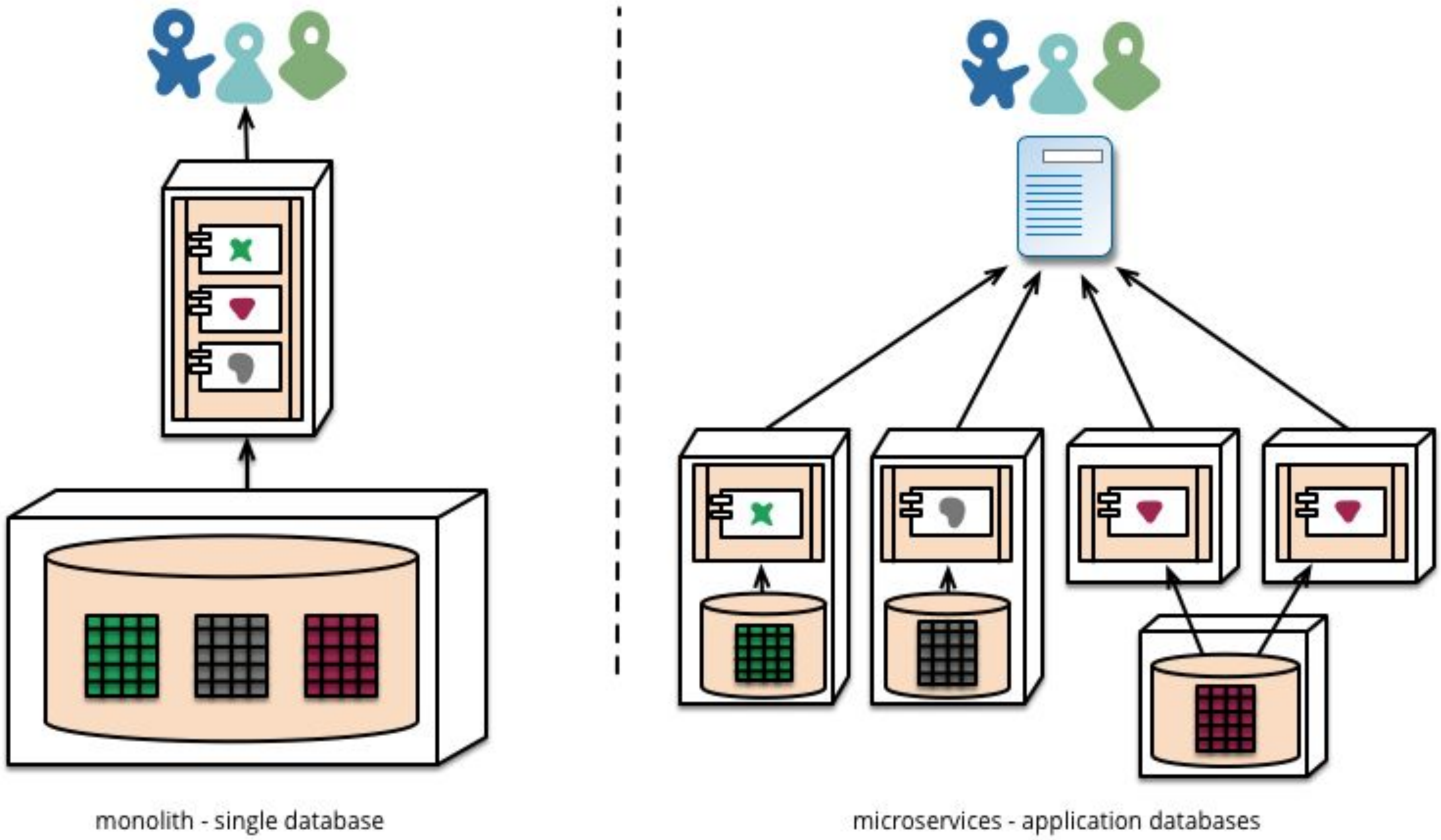
REST APIs, again

Independent development and deployment

Self-contained services (e.g., each with own database)

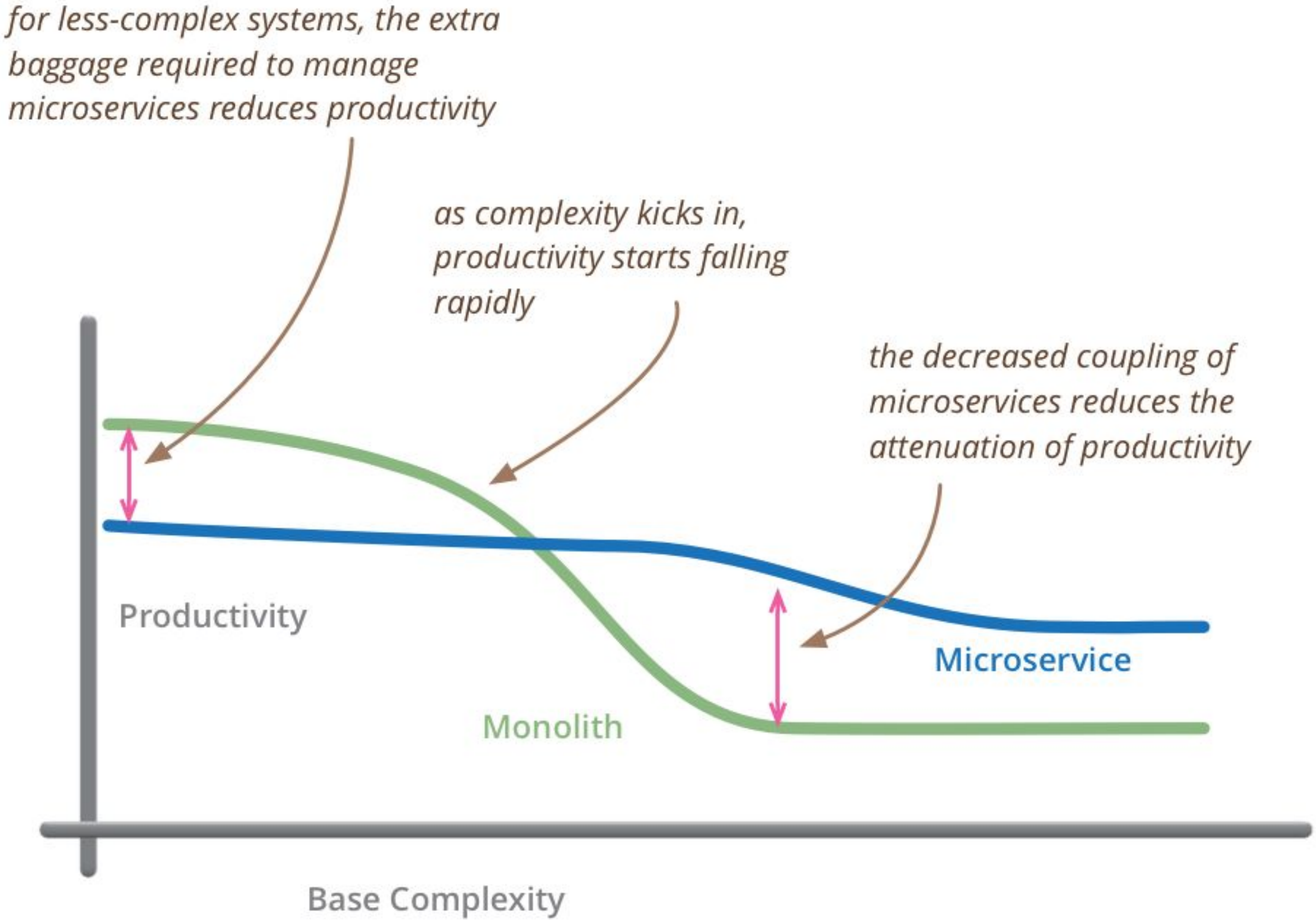
- multiple instances behind load-balancer

Streamline deployment



source: http://martinfowler.com/articles/microservices.html

Overhead



but remember the skill of the team will outweigh any monolith/microservice choice

Excursion: Testing in Distributed Systems

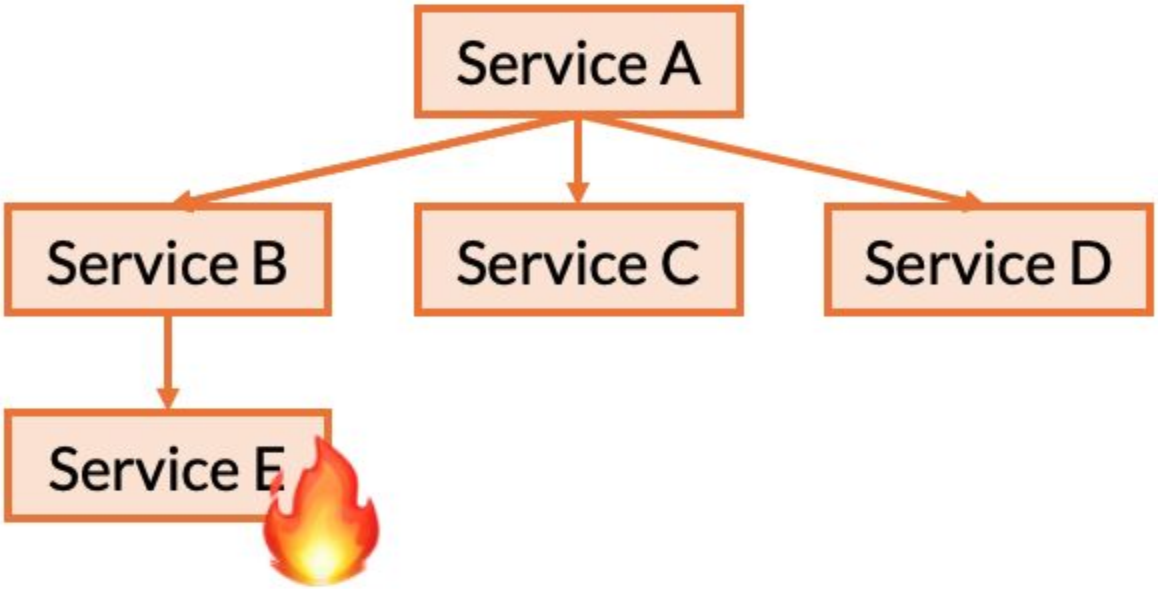
REST API Calls and Testing

Test happy path

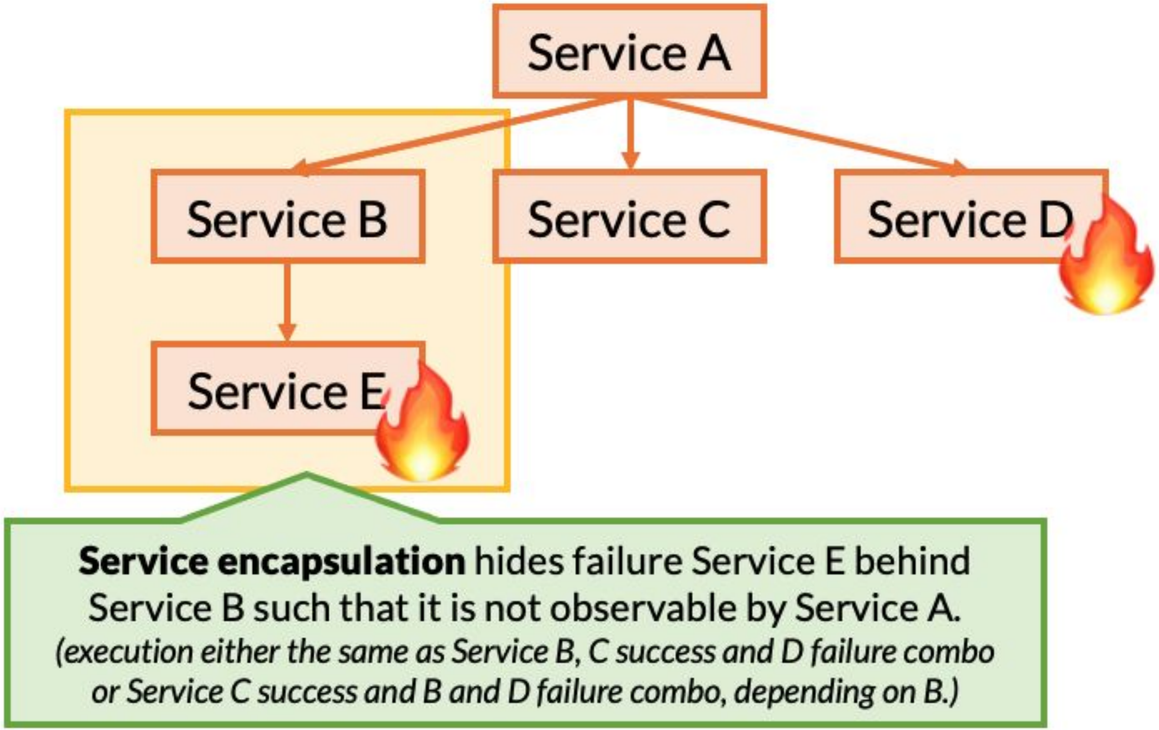
Test also error behavior!

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

Need to understand possible error behavior first


Service A
Service B
Service C
Service D
Service E

Handle Errors Locally



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

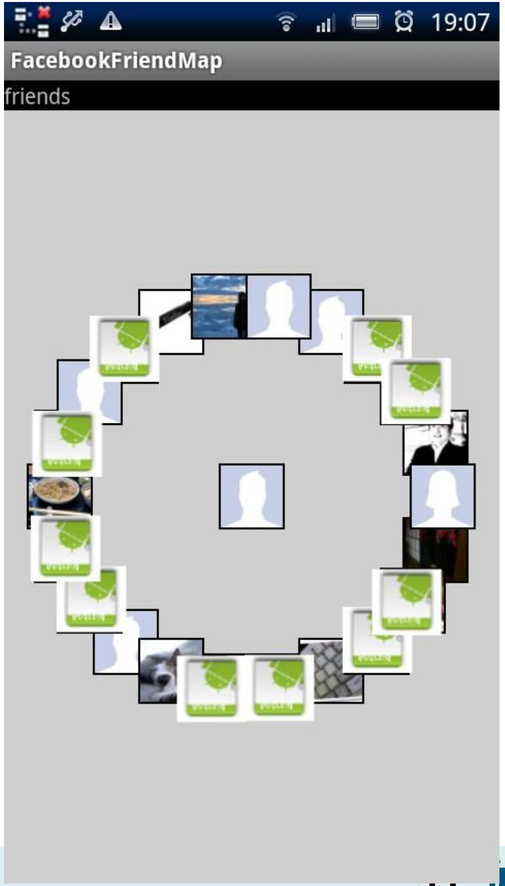
How to test?

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

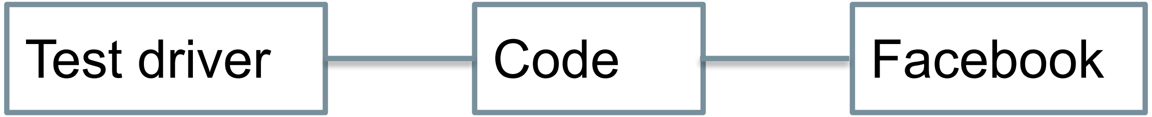
- 3rd party Facebook apps
- Android user interface
- Backend uses Facebook data



Testing in real environments


```java
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




```java
@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 driver — Code — Facebook

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

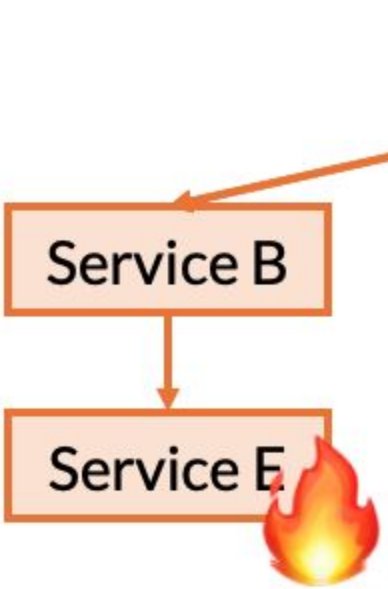
Test driver — Code — Mock Facebook

```java
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

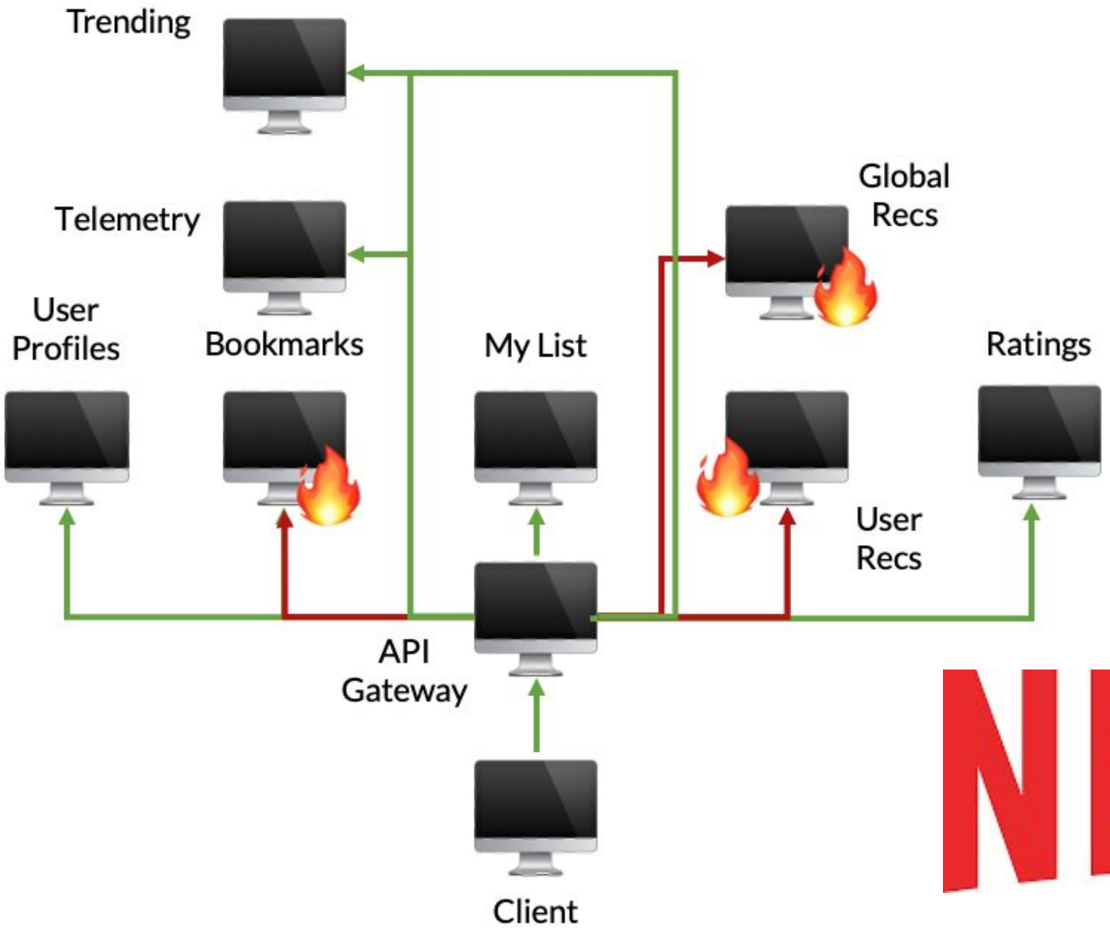
Experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production

NETFLIX



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

Distributed Event-Based System



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

- Options
- Manage access
- Repository roles
- Security & analysis
- Branches
- Webhooks
- Notifications
- Integrations
- Deploy keys
- Autolink references
- Actions
- Environments
- Secrets
- Pages

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.

[x] **Active**

We will deliver event details when this hook is triggered.

Add webhook

ute for
WARE
ARCH

Push vs Pull: RPC vs Callbacks

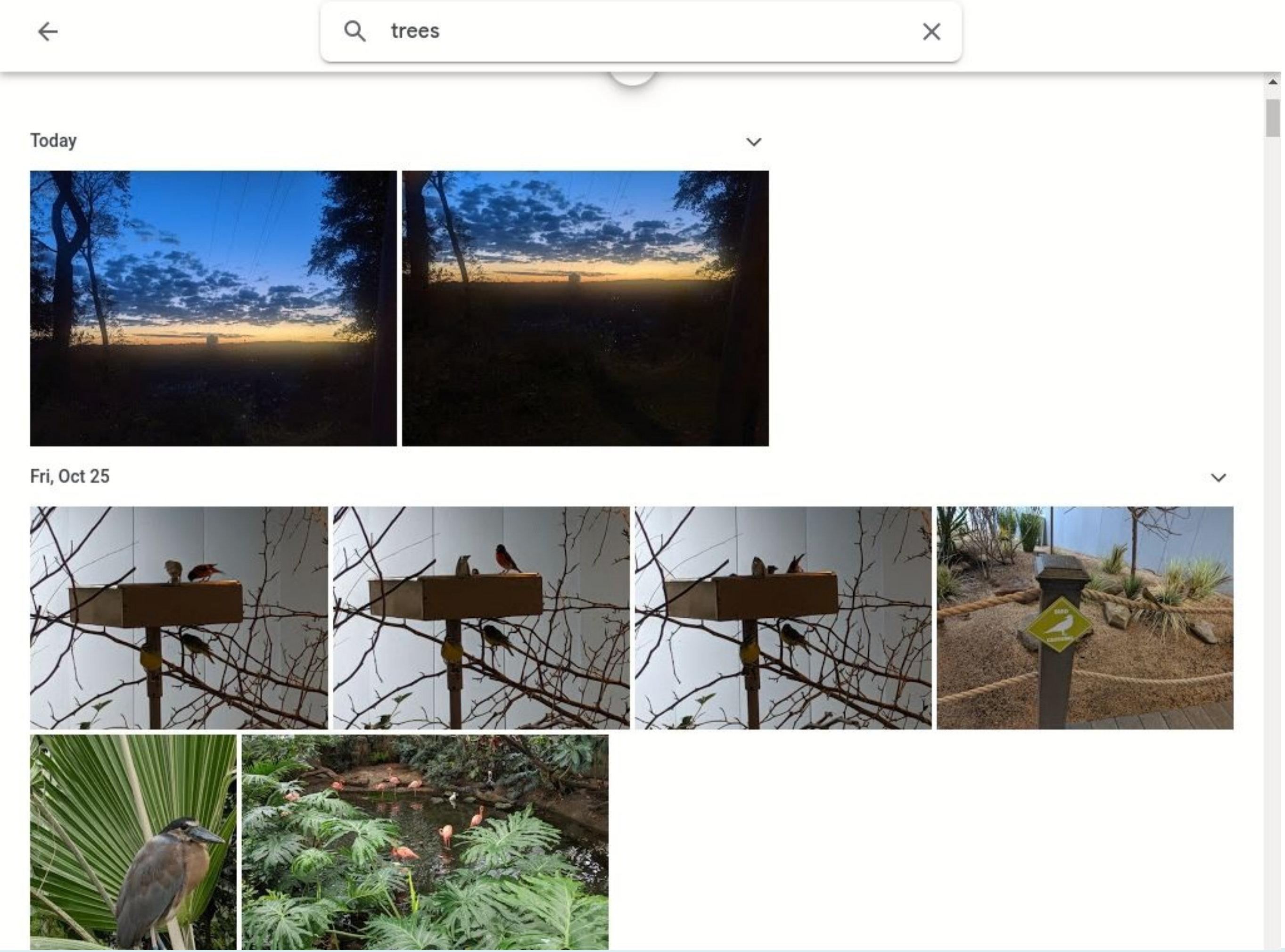
Both libraries and frameworks possible with RPC

- 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

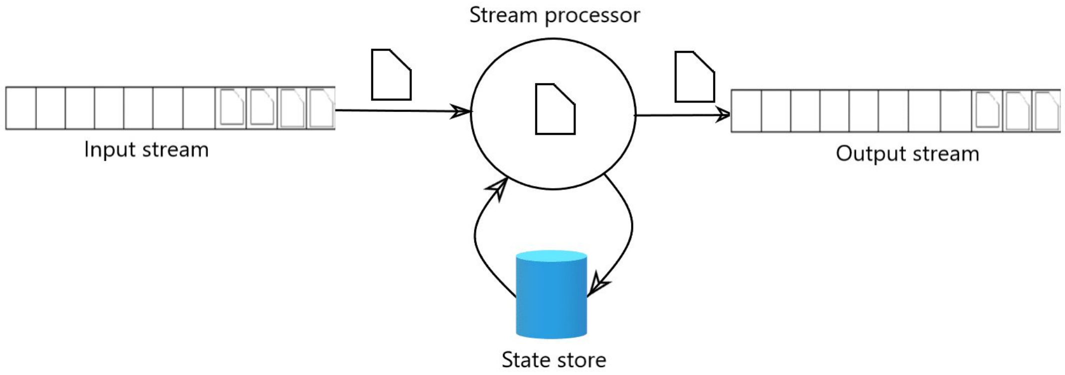
Example: Tagging of many images Indexing for search

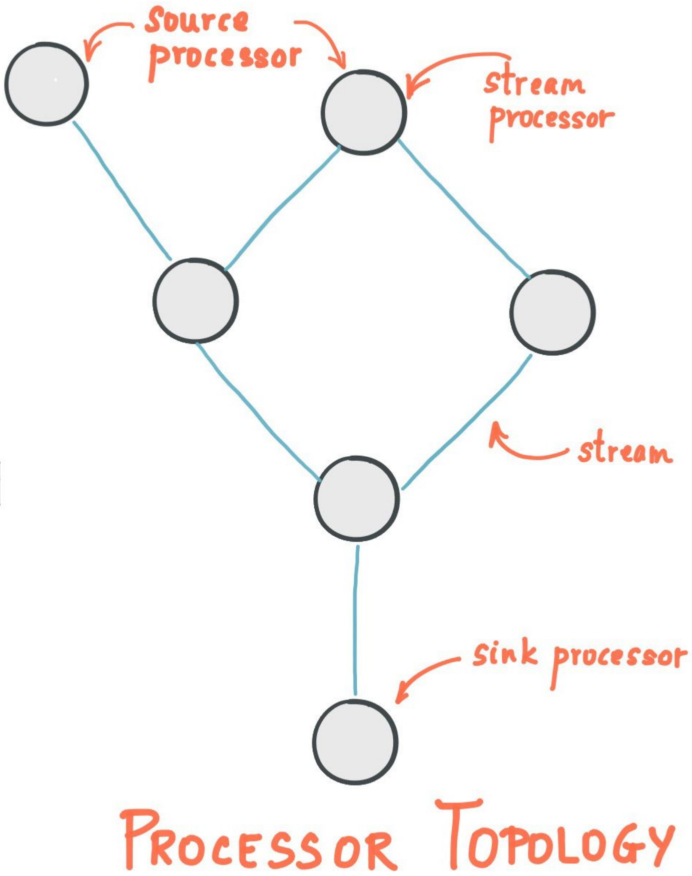


Recall: RxJava

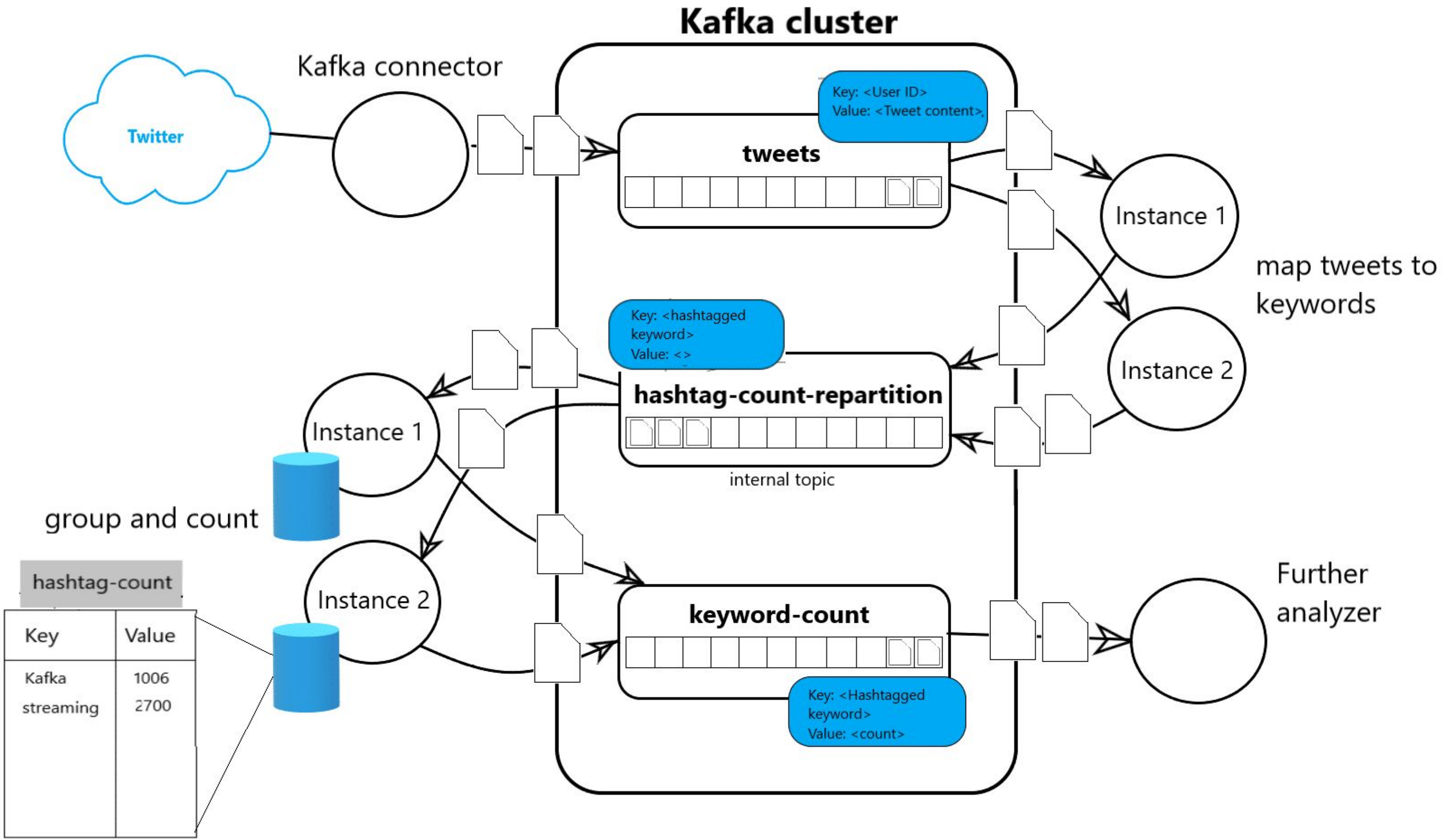
```
PublishSubject<Integer> x = PublishSubject.create();
PublishSubject<Integer> y = PublishSubject.create();
Observable<Integer> z = Observable.combineLatest(x, y,
(a,b)->a+b);
z.subscribe(System.out::println);
x.onNext(3);
y.onNext(5);
x.onNext(5);
```

Apache Kafka





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



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