Microservice Resiliency

From Front to Back End

QCon São Paulo, 2017

Lance Ball, Senior Software Engineer, Red Hat

Who am I?

Senior Software Engineer, Red Hat

Who am I?

Senior Software Engineer, Red Hat

RED HAT® JBOSS® MIDDLEWARE

redhat.

Who am I?

Senior Software Engineer, Red Hat

RED HAT® JBOSS® MIDDLEWARE

project:odd

redhat.

Who am I?

Senior Software Engineer, Red Hat

RED HAT® JBOSS® MIDDLEWARE

project:odd

node JS®

redhat.

µ Service

> “ software applications as suites of independently deployable services

https://martinfowler.com/articles/microservices.html

redhat.

µ Service

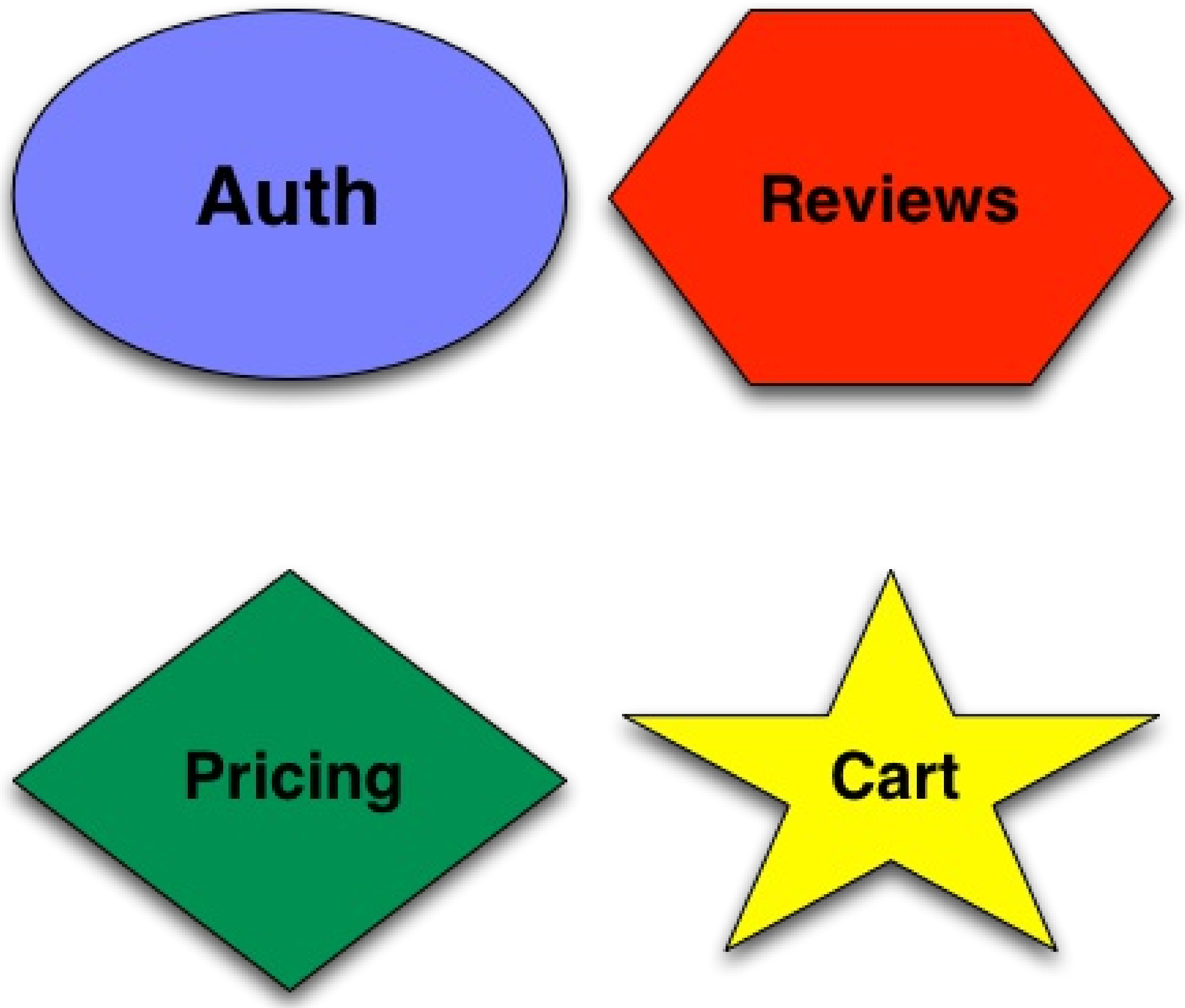
> “ software applications as suites of independently deployable services

https://martinfowler.com/articles/microservices.html

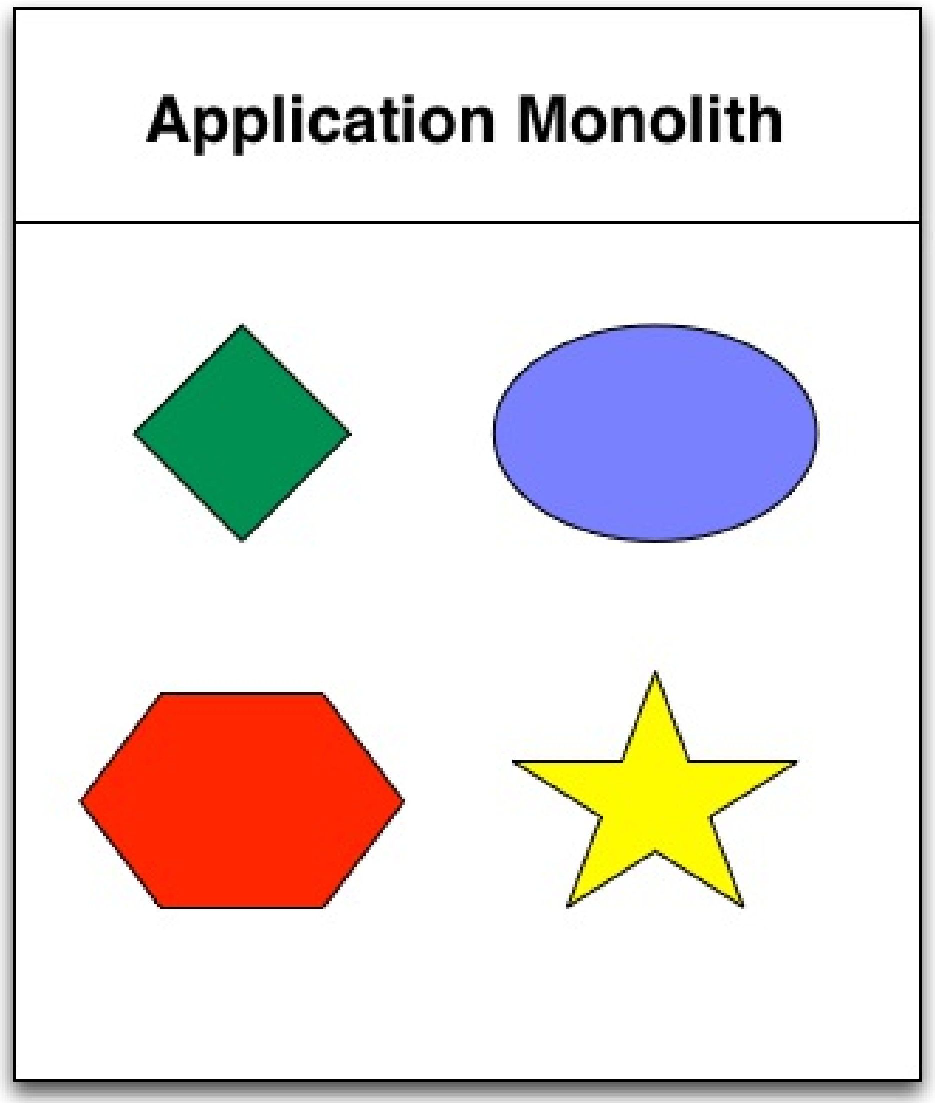
But what does this mean?!

What's in an application?

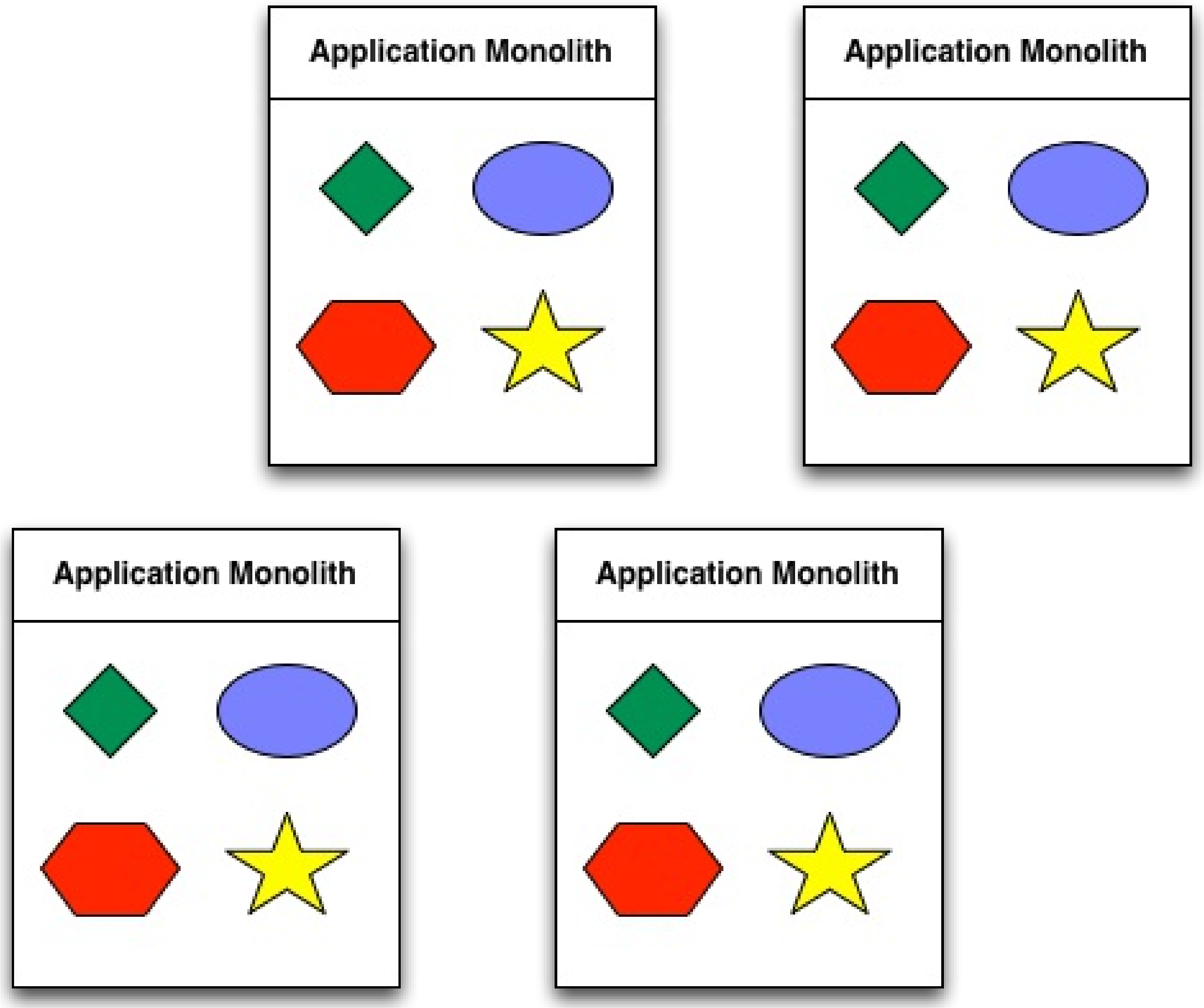
Stuff



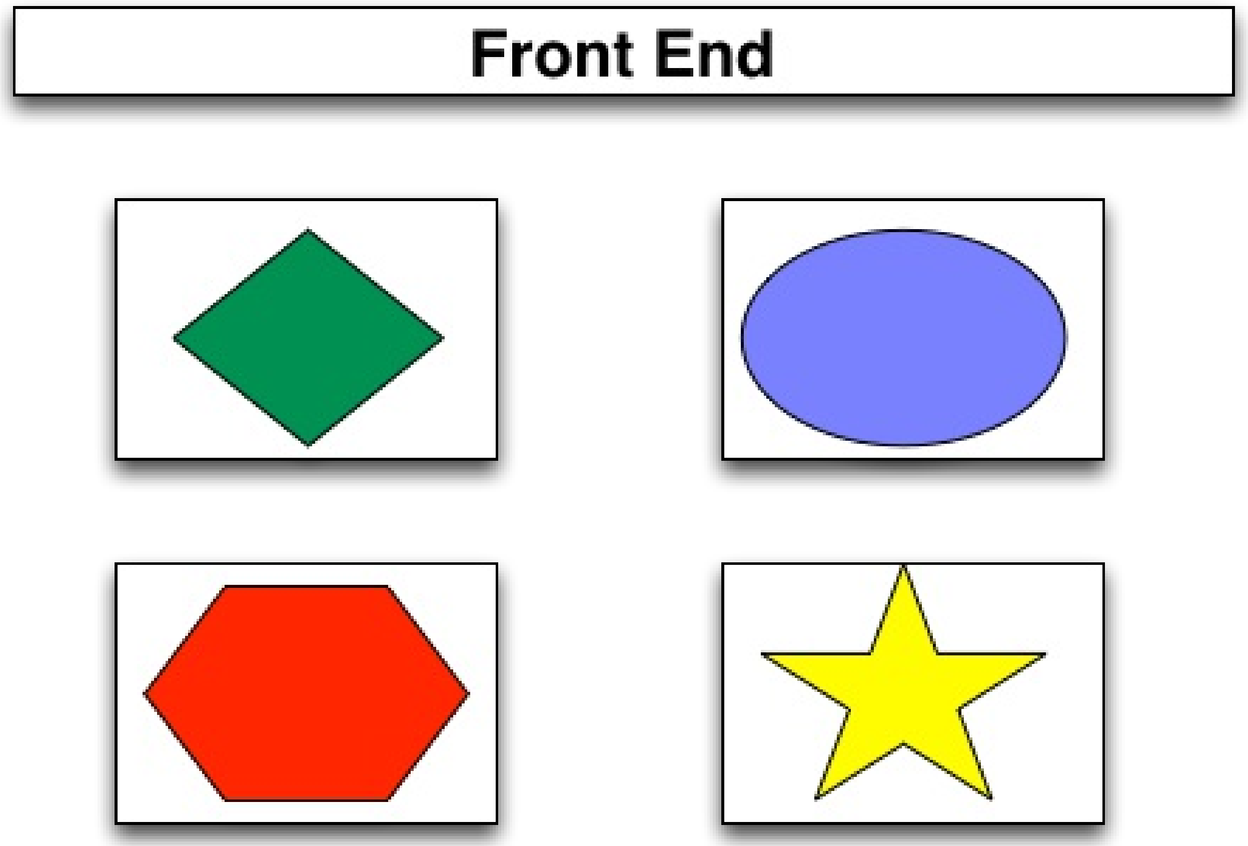
Monolithic application



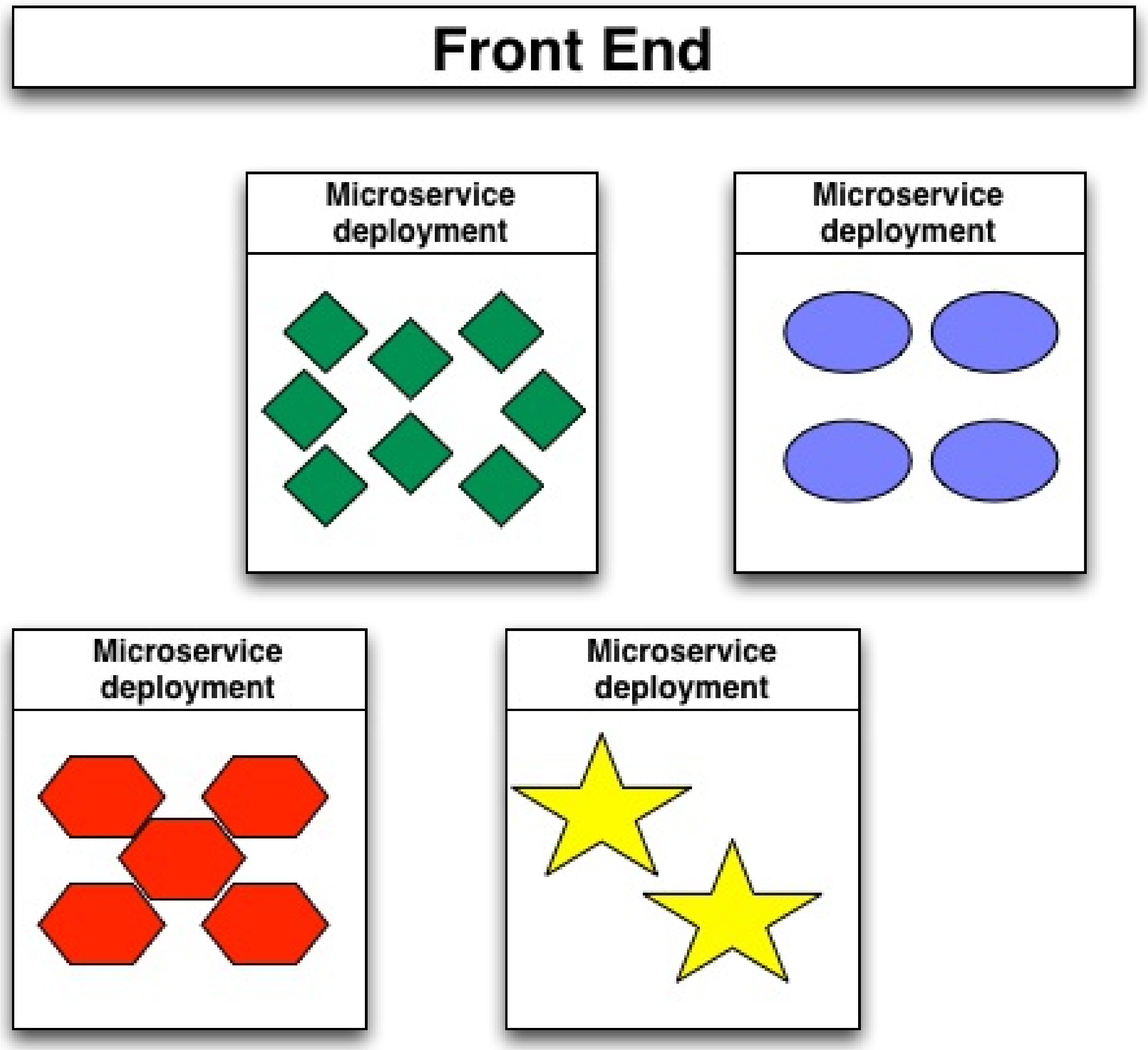
Scaling a monolith



Microservice application

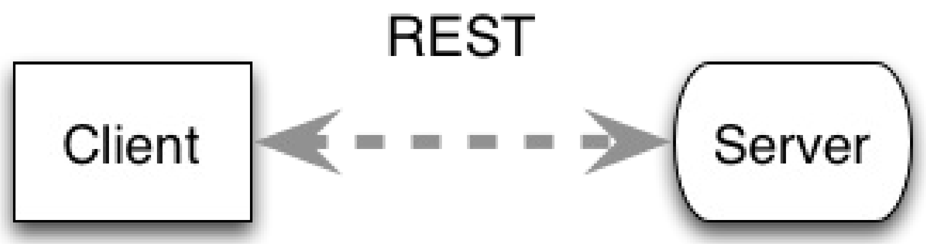


Scaled microservices

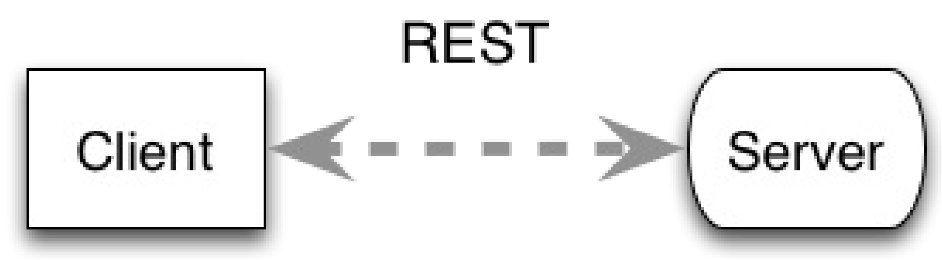


Wait... isn't this the UX track?

Service Lifecycle

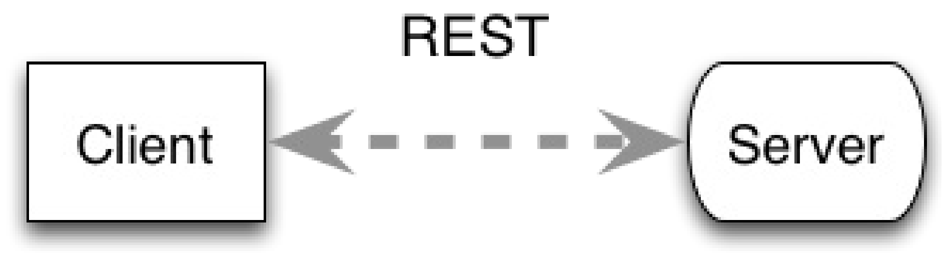


Service Lifecycle



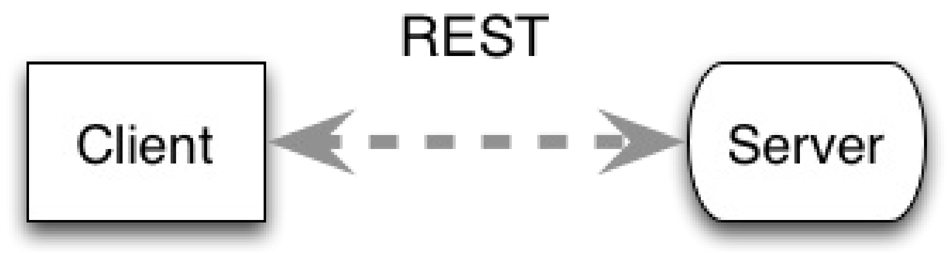
➥ Client makes a request

Service Lifecycle



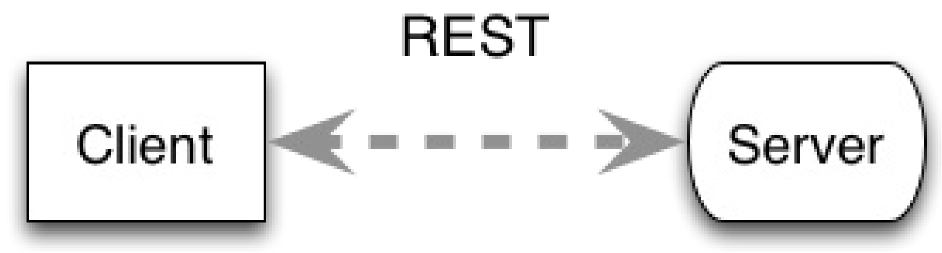
- Client makes a request
- Server provides a response

Service Lifecycle



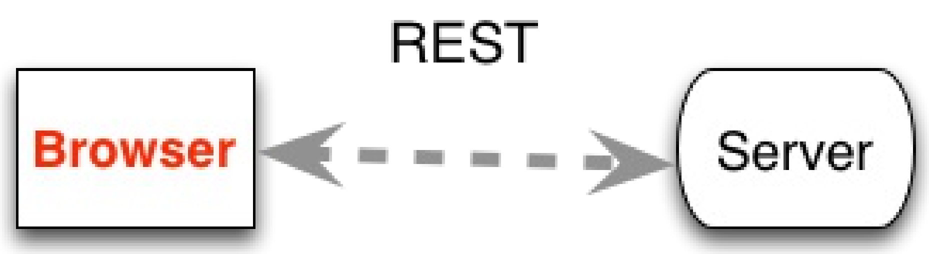
- Client makes a request
- Server provides a response
- Often using HTTP transport

Service Lifecycle

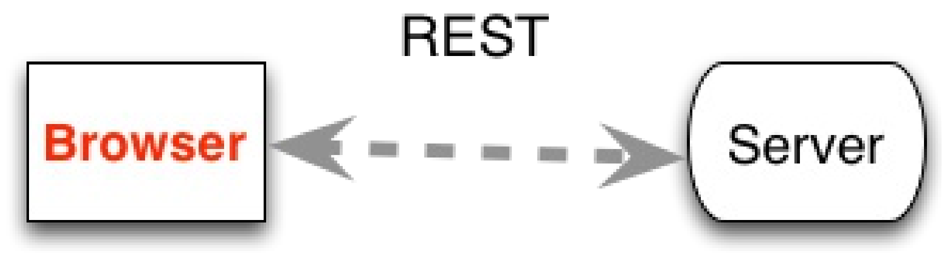


- Client makes a request
- Server provides a response
- Often using HTTP transport
- Often with JSON data format

In the Browser

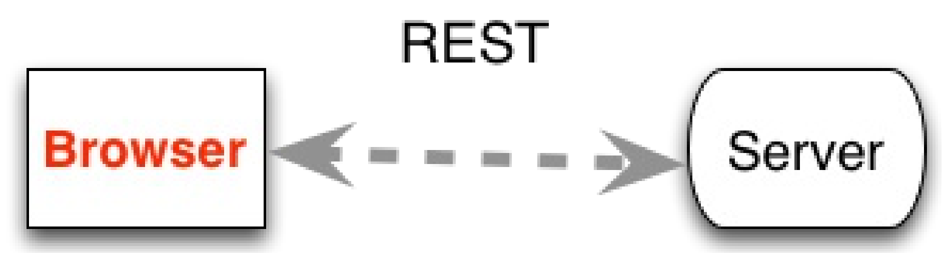


In the Browser



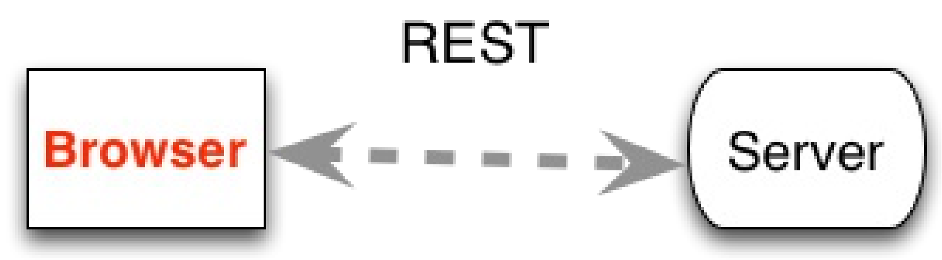
- XMLHttpRequest

In the Browser



- XMLHttpRequest
- JQuery

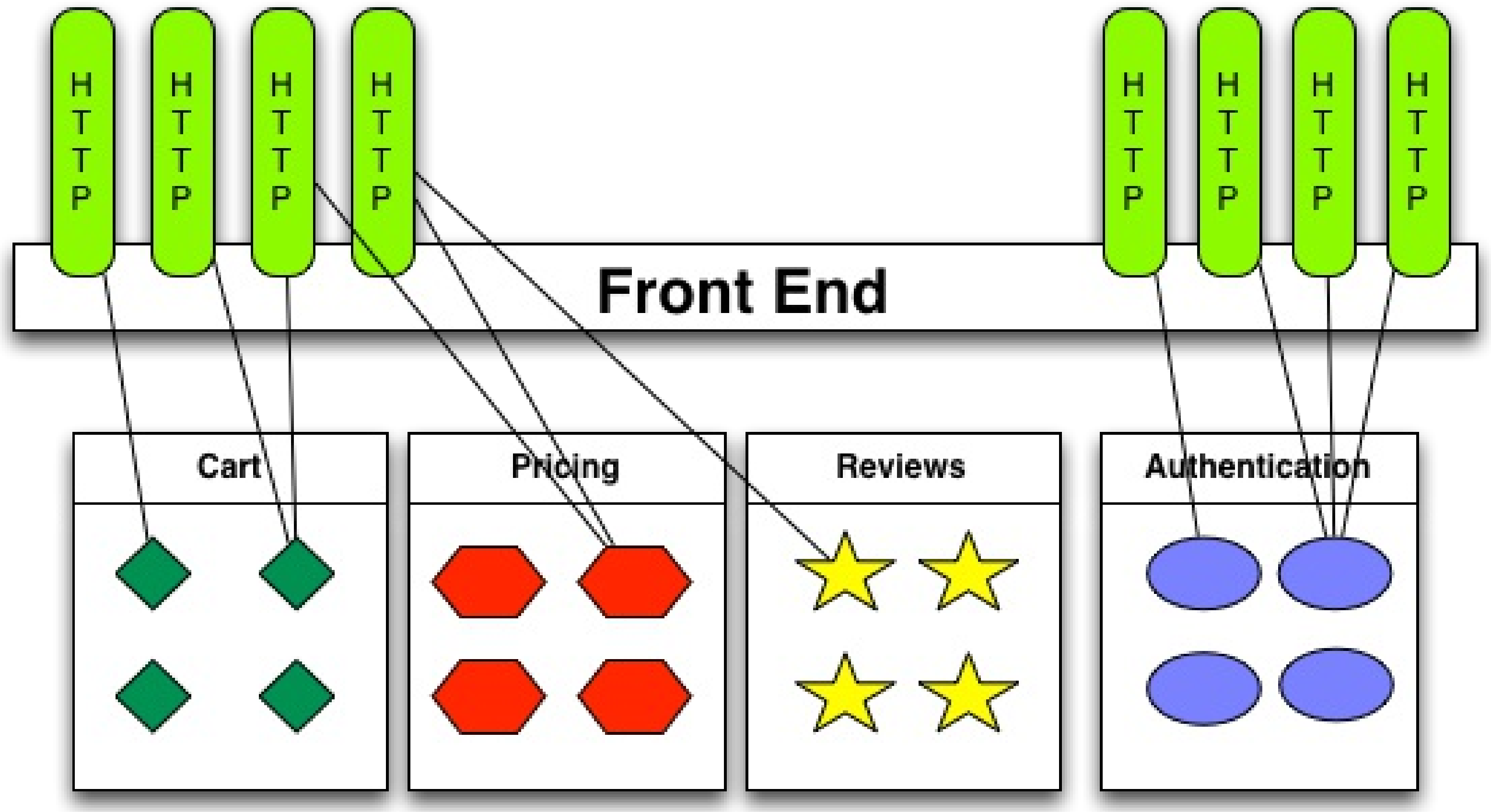
In the Browser



- XMLHttpRequest
- JQuery
- AJAX

Microservice Requests

(simplified)



Operational Complexity

Microservices Visualized

https://twitter.com/ThePracticalDev/status/845285541528719360

Problems

Problems

- Timeouts

Problems

- Timeouts
- Network saturation

Problems

- Timeouts
- Network saturation
- Programmer error

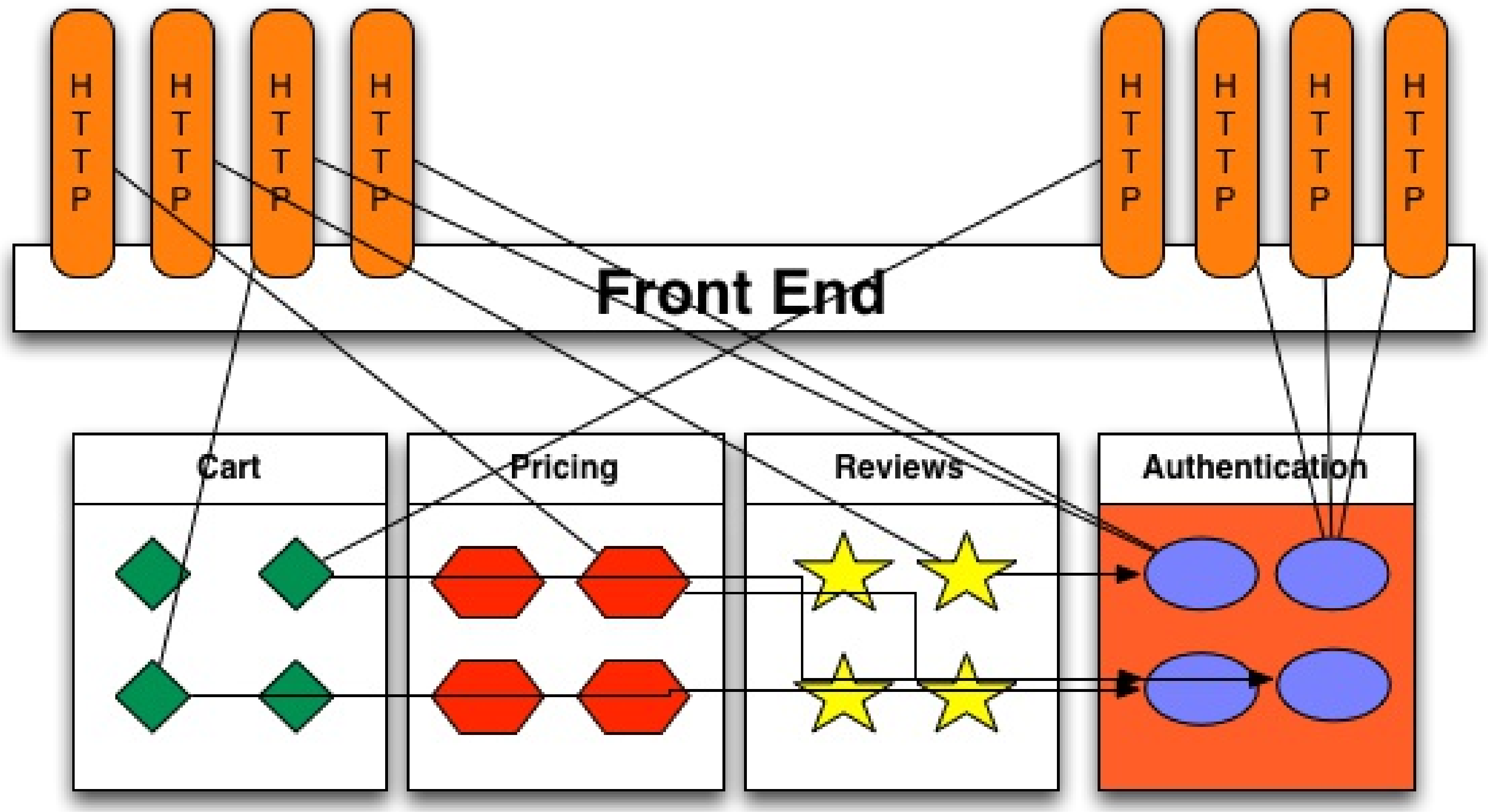
Problems

- Timeouts
- Network saturation
- Programmer error
- Disk failure

Problems

- Timeouts
- Network saturation
- Programmer error
- Disk failure
- Transitive dependencies

Cascading failures



redhat.

How to deal with all this

How to deal with all this

- ➯ Limit single points of failure

How to deal with all this

- Limit single points of failure
- Shed load when possible

How to deal with all this

- Limit single points of failure
- Shed load when possible
- Provide fallback behavior

How to deal with all this

- Limit single points of failure
- Shed load when possible
- Provide fallback behavior
- Optimize failure discovery

Circuit Breaker

Circuit Breaker

- Calls that could fail are wrapped

Circuit Breaker

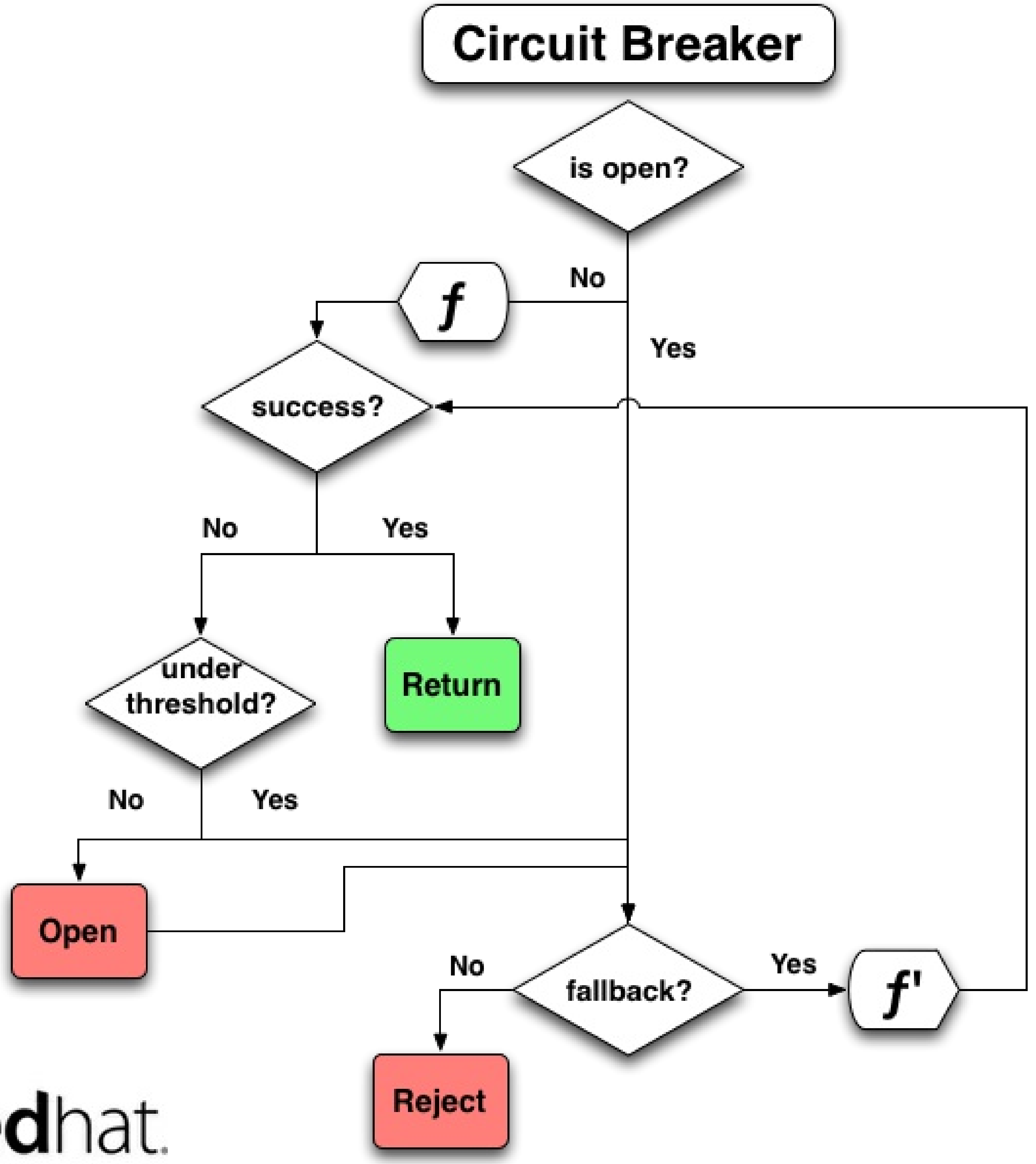
- Calls that could fail are wrapped
- Circuit opens at a failure threshold

Circuit Breaker

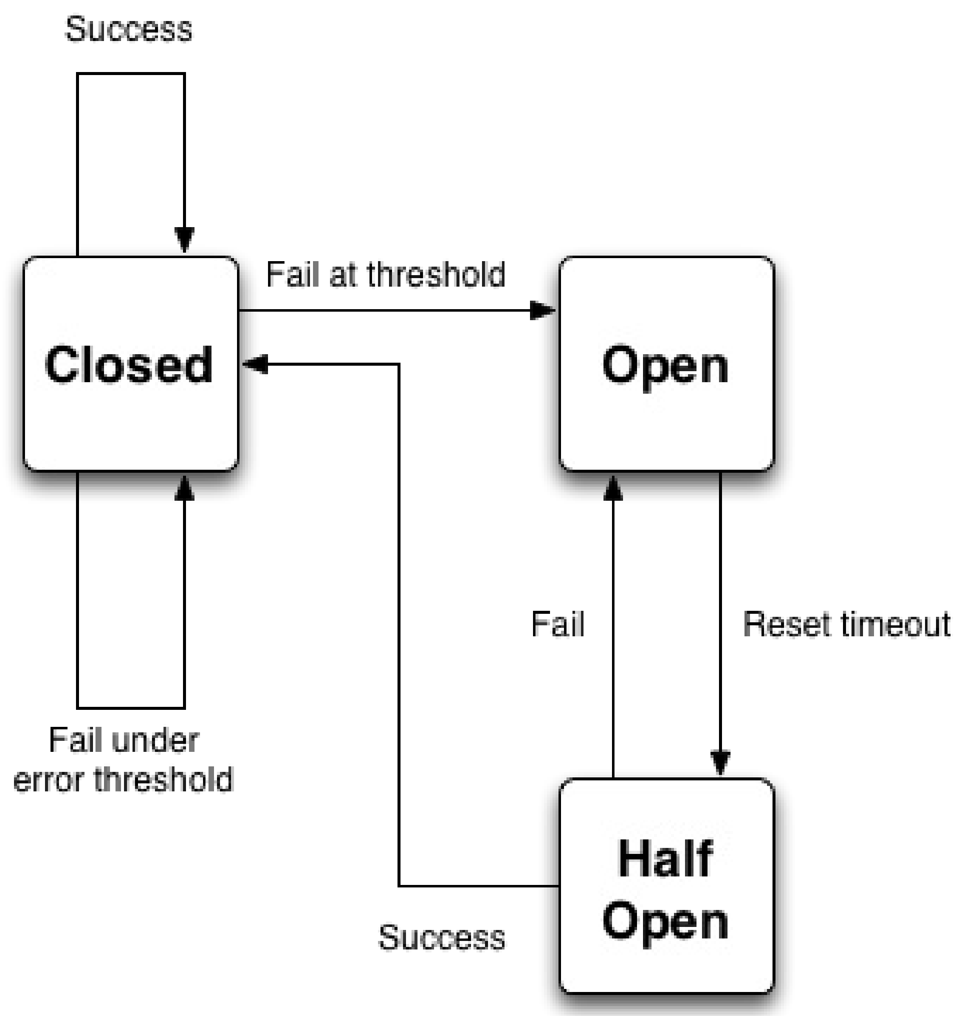
- Calls that could fail are wrapped
- Circuit opens at a failure threshold
- Further calls short circuit for a while

Circuit Breaker

- Calls that could fail are wrapped
- Circuit opens at a failure threshold
- Further calls short circuit for a while
- Later, circuit tries again and trips immediately if there is failure


Circuit Breaker
is open?
No
f
Yes
success?
No
Yes
under threshold?
Return
No
Yes
Open
No
fallback?
Yes
f'
Reject

Circuit State



Async operation that could fail

```
// Use JQuery to get cart info
$.get('http://mystore.com/cart')
  .then((json) => {
    // update the UI with JSON data
  })
  .catch((e) => {
    // oops something went wrong
   console.error(e);
  })
```

Async operation that could fail

```
// Use JQuery to get cart info
$.get('http://mystore.com/cart')
  .then((json) => {
    // update the UI with JSON data
  })
  .catch((e) => {
    // oops something went wrong
   console.error(e);
  })
```

Shed load when possible

Aside - Promsies

```
// Use JQuery to get cart info
$.get('http://mystore.com/cart')
  .then((json) => {
    // update the UI with JSON data
  })
  .catch((e) => {
    // oops something went wrong
   console.error(e);
  })
```

Circuit Breaker Example

```
// Use JQuery's ajax wrapper and circuit breaker
// defaults for failure threshold, timing, etc.
const circuit = circuitBreaker($.get);

circuit.fire('http://nodejs.org/dist/index.json')
  .then((json) => {
    // update the UI with JSON data
  })
  // on failure, just log to console
  .catch(console.error);
```

redhat.

Circuit Breaker Example

```
// Use JQuery's ajax wrapper and circuit breaker
// defaults for failure threshold, timing, etc.
const circuit = circuitBreaker($.get);

circuit.fire('http://nodejs.org/dist/index.json')
  .then((json) => {
    // update the UI with JSON data
  })
  // on failure, just log to console
  .catch(console.error);
```

Circuit Breaker Example

```
// Use JQuery's ajax wrapper and circuit breaker
// defaults for failure threshold, timing, etc.
const circuit = circuitBreaker($.get);

circuit.fire('http://nodejs.org/dist/index.json')
  .then((json) => {
    // update the UI with JSON data
  })
  // on failure, just log to console
  .catch(console.error);
```

Promises vs. Callbacks

```javascript
// Wrap Node.js' fs.readFile as a promise-returning function
const readFile = circuitBreaker.promisify(fs.readFile);

const circuit = circuitBreaker(readFile, options);

circuit.fire('./package.json', 'utf-8')
  .then(console.log)
  .catch(console.error);
```

Circuit Breaker Fallback

Provides default behavior in case of error

```
circuit.fallback((file) => `Sorry, I can't read ${file}`);

// Fallback function is still a success case
circuit.fire('./package.jsob')
  .then((data) => console.log(`package.json: \n${data}`))
  .catch((err) => console.error(`ERR: ${err}`));
```

Circuit Breaker Fallback

Provides default behavior in case of error

```
circuit.fallback((file) => `Sorry, I can't read ${file}`);

// Fallback function is still a success case
circuit.fire('./package.jsob')
  .then((data) => console.log(`package.json: \n${data}`))
  .catch((err) => console.error(`ERR: ${err}`));
```

Caching

Always returns the same value

```
const now = circuitBreaker(Date, { cache: true });
```

Caching

Always returns the same value

```
const now = circuitBreaker(Date, { cache: true });

circuit.fire().then(console.log);
// Mon Apr 10 2017 12:10:26 GMT-0400 (EDT)
circuit.fire().then(console.log);
// Mon Apr 10 2017 12:10:26 GMT-0400 (EDT)
circuit.fire().then(console.log);
// Mon Apr 10 2017 12:10:26 GMT-0400 (EDT)
```

When is this useful?

- Frequent hits, infrequent change
- E.g. username

```
const username = circuitBreaker(fetchUsername, { cache: true

// periodically clear the cache
setInterval(_ => username.clearCache(), 5000);
```

redhat.

Events

Circuit breakers are event emitters

```
  // Update the UI specifically for timeout errors
  circuit.on('timeout',
   () => $(element).prepend(
     mkNode(`${route} is taking too long to respond.`)));
```

Events

Circuit breakers are event emitters

```
// Update the UI specifically for timeout errors
circuit.on('timeout',
  () => $(element).prepend(
    mkNode(`${route} is taking too long to respond.`)));
```

- `fire`
- `reject`
- `timeout`
- `success`
- `failure`
- `open`
- `close`
- `halfOpen`
- `fallback`
- `snapshot`

Status

```
// create a 10 sec window with 10 buckets of 1 sec
const circuit = circuitBreaker(asyncFunc, {
   rollingCountTimeout: 10000,
   rollingCountBuckets: 10
});

// status is calculated every time status is accessed
const status = circuit.status

// print the entire statistical window
console.log(status.window);

// print the rolling stats
console.log(status.stats);
```

redhat.

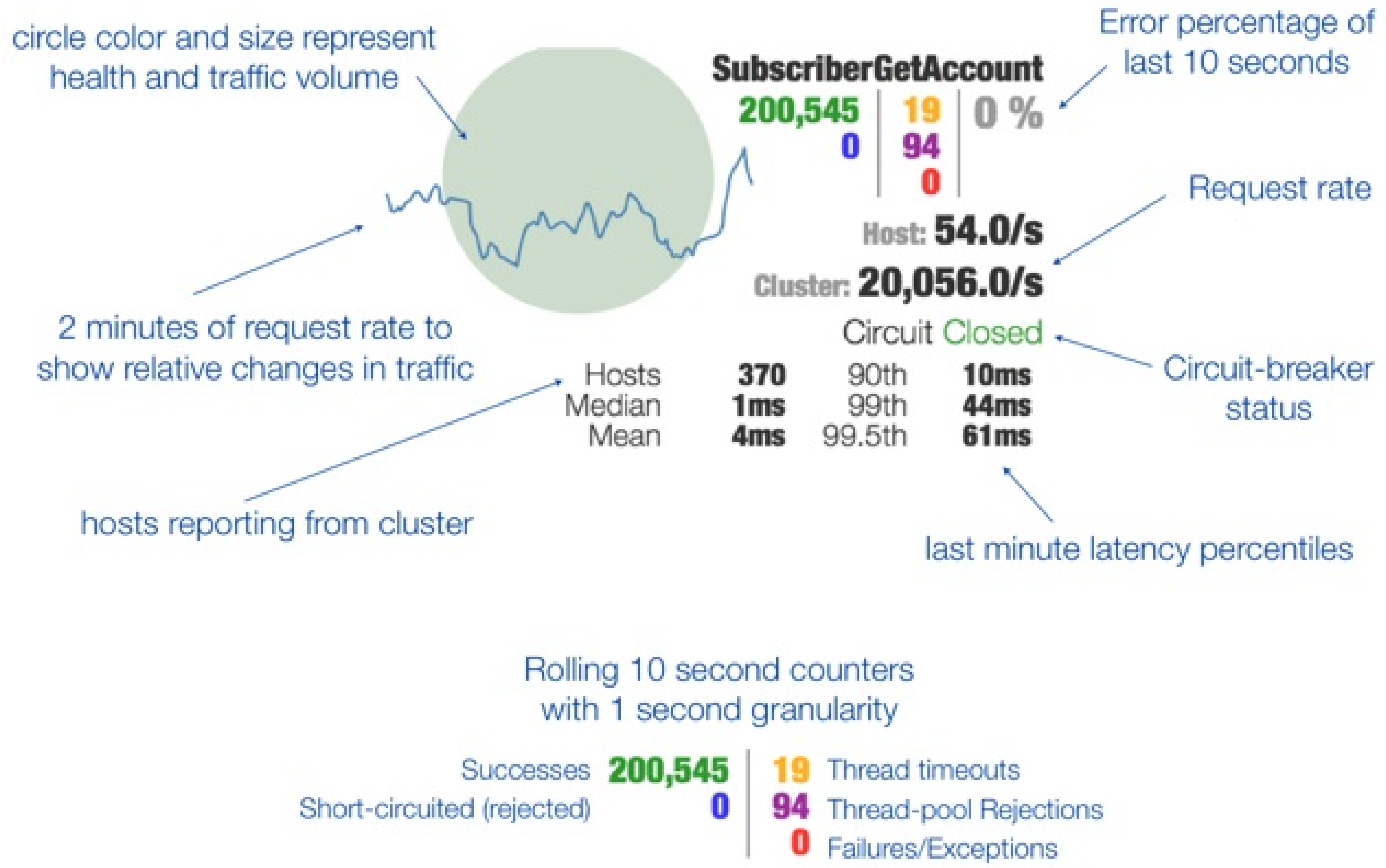
Status

```
// create a 10 sec window with 10 buckets of 1 sec
const circuit = circuitBreaker(asyncFunc, {
  rollingCountTimeout: 10000,
  rollingCountBuckets: 10
});

// status is calculated every time status is accessed
const status = circuit.status

// print the entire statistical window
console.log(status.window);

// print the rolling stats
console.log(status.stats);
```

Status

```
// print the rolling stats
console.log(status.stats);

// { failures: 3,
//   fallbacks: 4,
//   successes: 44,
//   rejects: 4,
//   fires: 48,
//   timeouts: 1,
//   cacheHits: 0,
//   cacheMisses: 0 }
```

Dashboard


circle color and size represent health and traffic volume
SubscriberGetAccount
200,545
19
0 %
0
94
0
Error percentage of last 10 seconds
Request rate
Host: 54.0/s
Cluster: 20,056.0/s
2 minutes of request rate to show relative changes in traffic
Circuit Closed
Circuit-breaker status
Hosts 370
Median 1ms
Mean 4ms
90th 10ms
99th 44ms
99.5th 61ms
hosts reporting from cluster
last minute latency percentiles
Rolling 10 second counters with 1 second granularity
Successes 200,545
Short-circuited (rejected) 0
19 Thread timeouts
94 Thread-pool Rejections
0 Failures/Exceptions

http://techblog.netflix.com/2012/12/hystrix-dashboard-and-turbine.html

Demo

Obrigado & Questions

http://lanceball.com/qcon-saopaulo-2017/
https://github.com/lance/qcon-saopaulo-2017
Twitter - @lanceball
GitHub - @lance