

# Circuit Breaker

Stop the Fire Before It Happens

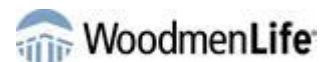
---

Nebraska Code Conference  
May 19, 2016

Brice McIver, Software Consultant  
*[bmciver@keyholesoftware.com](mailto:bmciver@keyholesoftware.com)*



# Sponsors



# Agenda

Electricity

1

Software

2

Implementation

3

Hystrix

4

Summary

5

# Agenda

Electricity

1

Software

2

Implementation

3

Hystrix

4

Summary

5

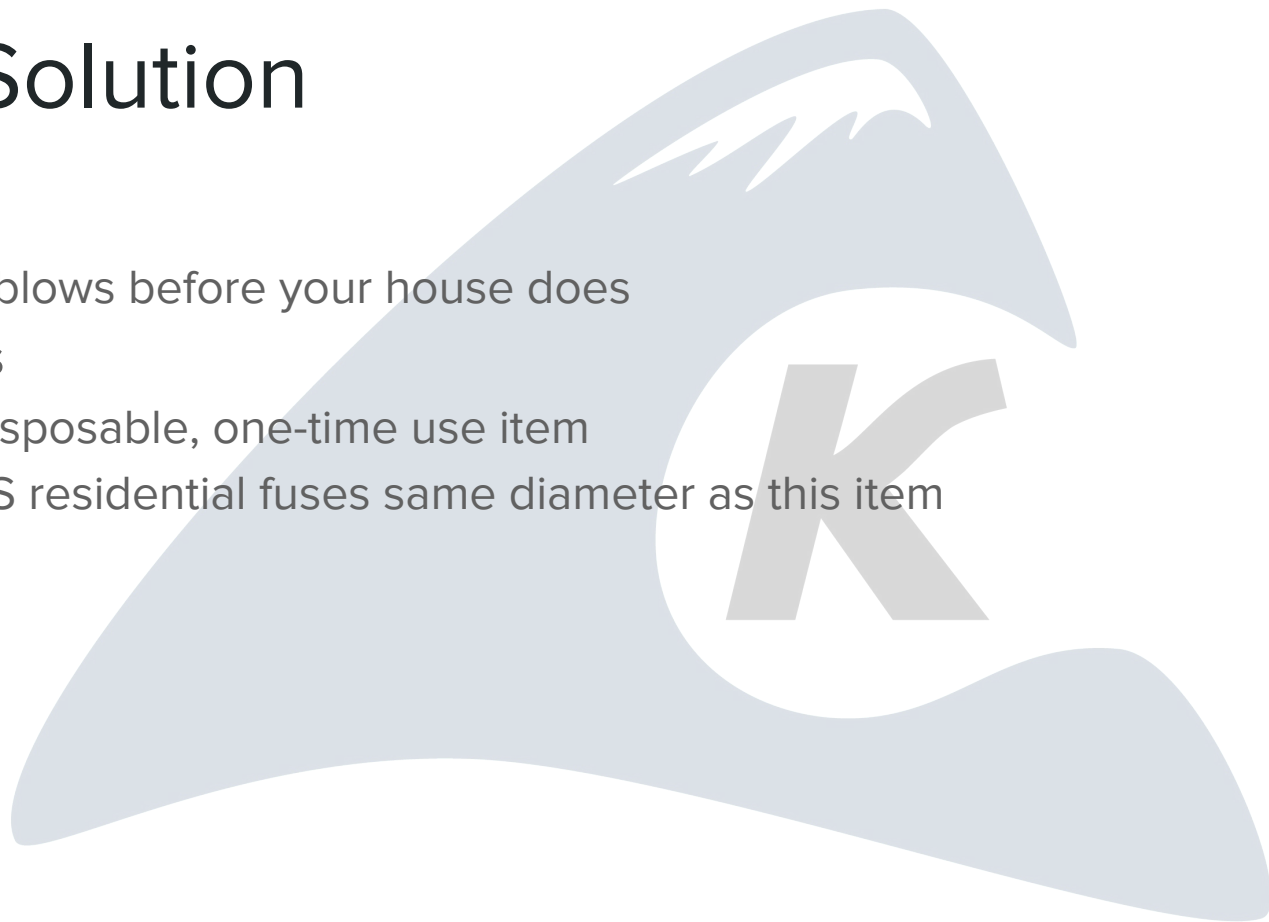
# In the Beginning

- People
  - Like electricity
  - Want electricity in their houses
  - Want to power lots of things with electricity
- Joule heating
  - $H \propto I^2 \cdot R \cdot t$
  - Wires get hot
  - Walls get hot



# First Solution

- Fuses
  - It blows before your house does
- Issues
  - Disposable, one-time use item
  - US residential fuses same diameter as this item









# Second Solution

- Circuit Breaker
  - Works like a fuse, but resettable
- Operation
  - Detect excess current
  - Fail
  - Open the circuit
- Allows one subsystem to fail without destroying the entire system

# Agenda

Electricity

1

Software

2

Implementation

3

Hystrix

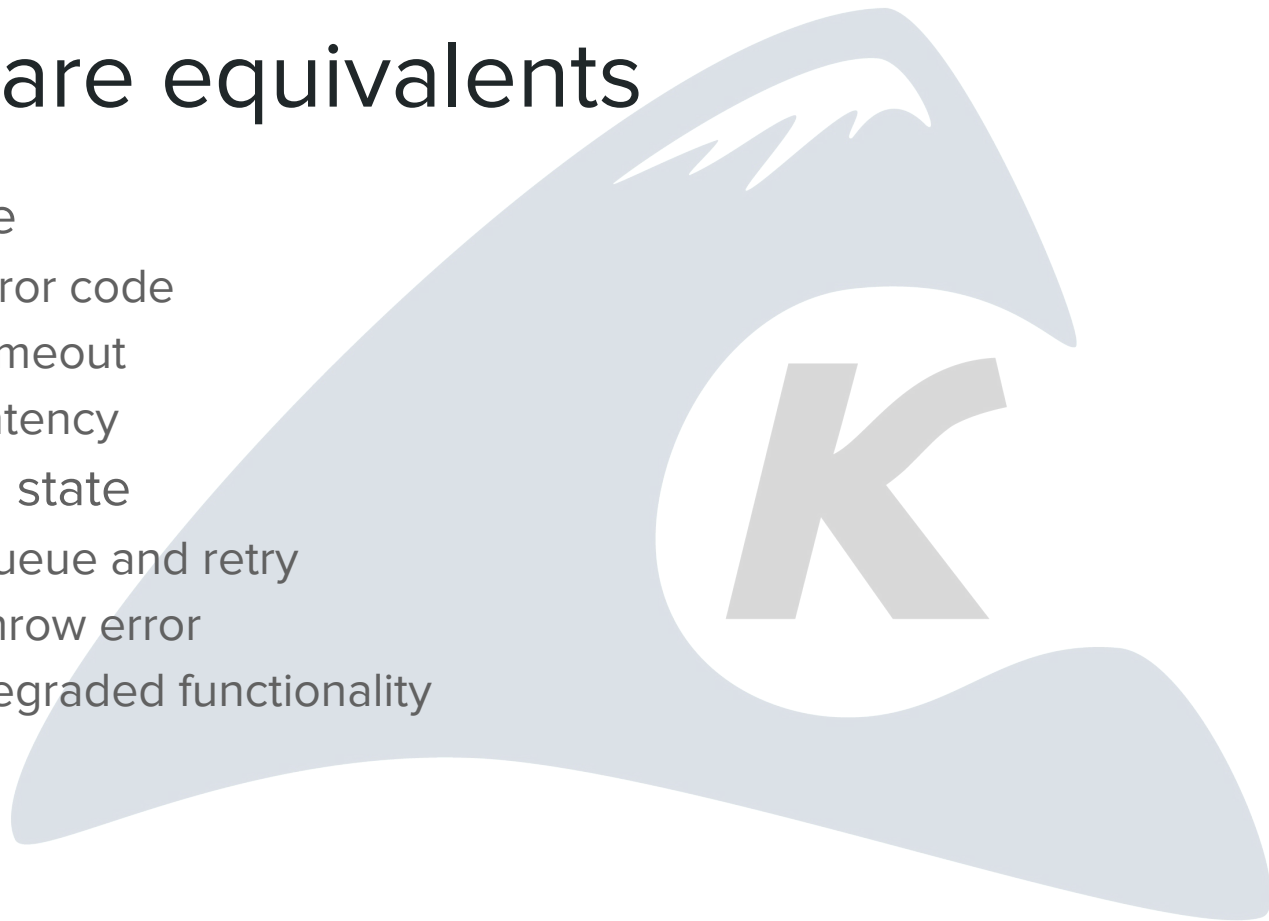
4

Summary

5

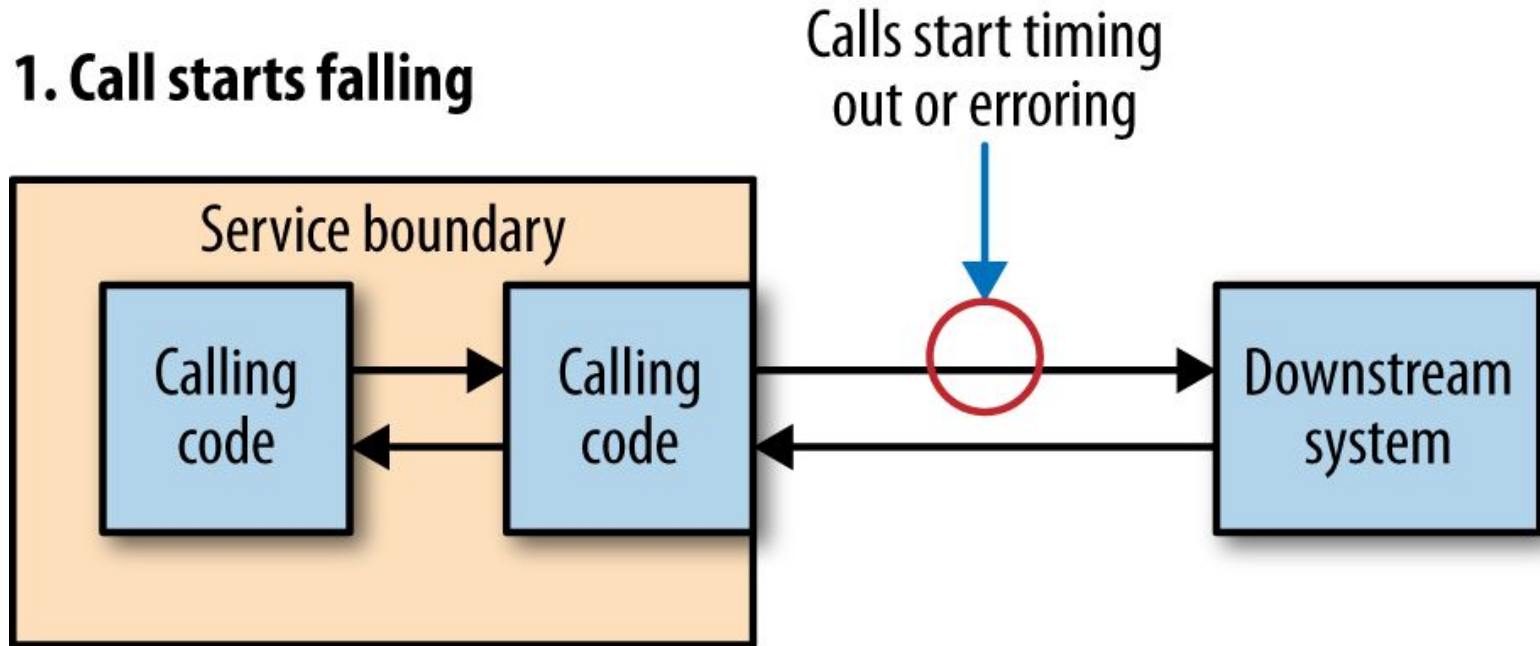
# Software equivalents

- Failure
  - Error code
  - Timeout
  - Latency
- Blown state
  - Queue and retry
  - Throw error
  - Degraded functionality
- Reset



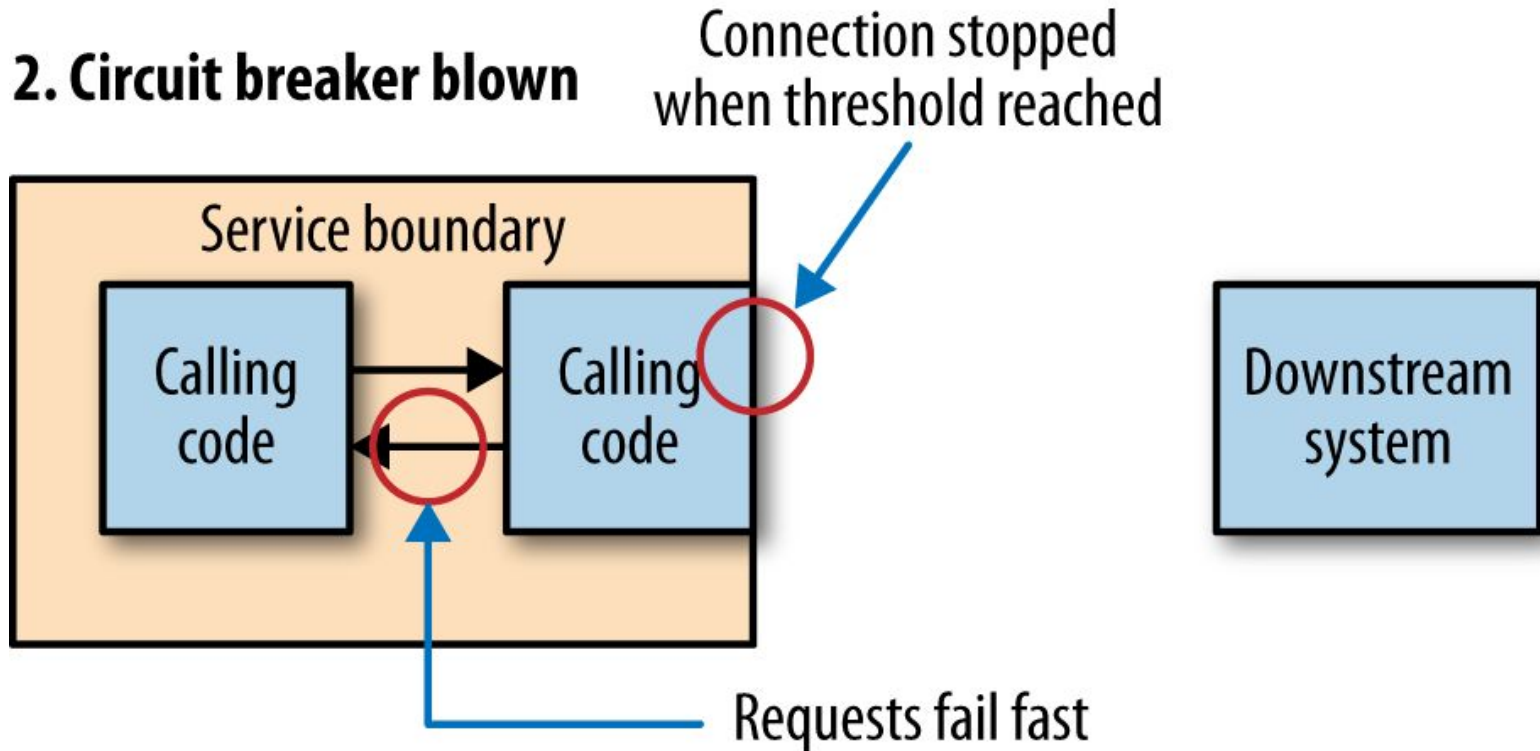
# Lifecycle

## 1. Call starts falling



# Lifecycle

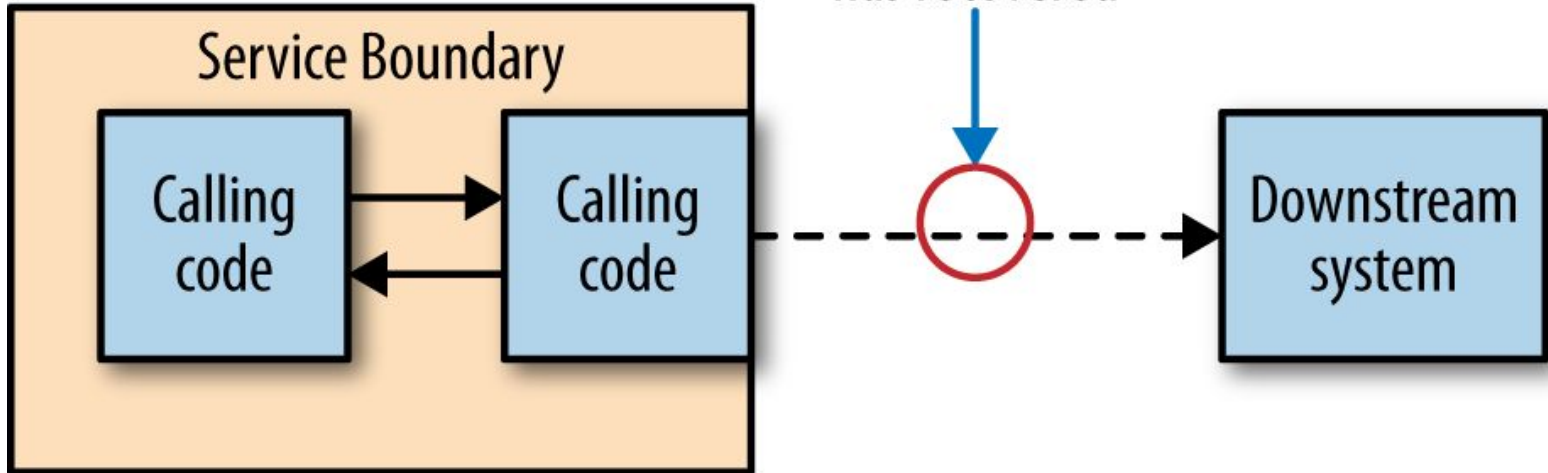
## 2. Circuit breaker blown



# Lifecycle

## 3. Health checks sent

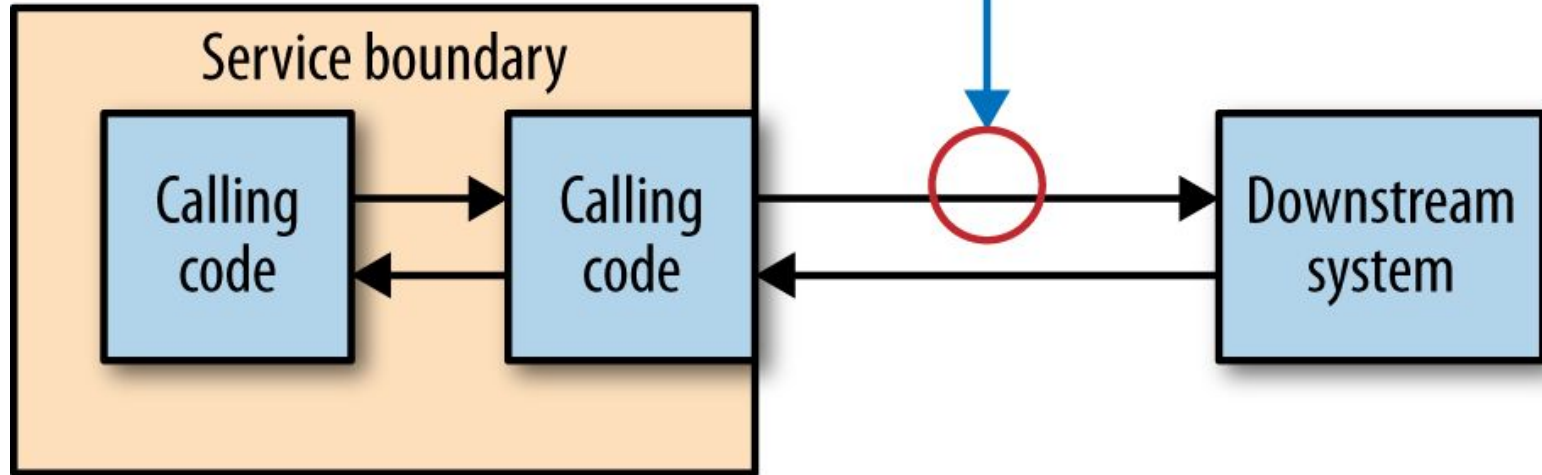
Occasional health checks sent  
to see if the downstream system  
has recovered



Lif

#### 4. Circuit breaker reset

Connection reset when  
healthy threshold reached





# Agenda

Electricity

1

Software

2

Implementation

3

Hystrix

4

Summary

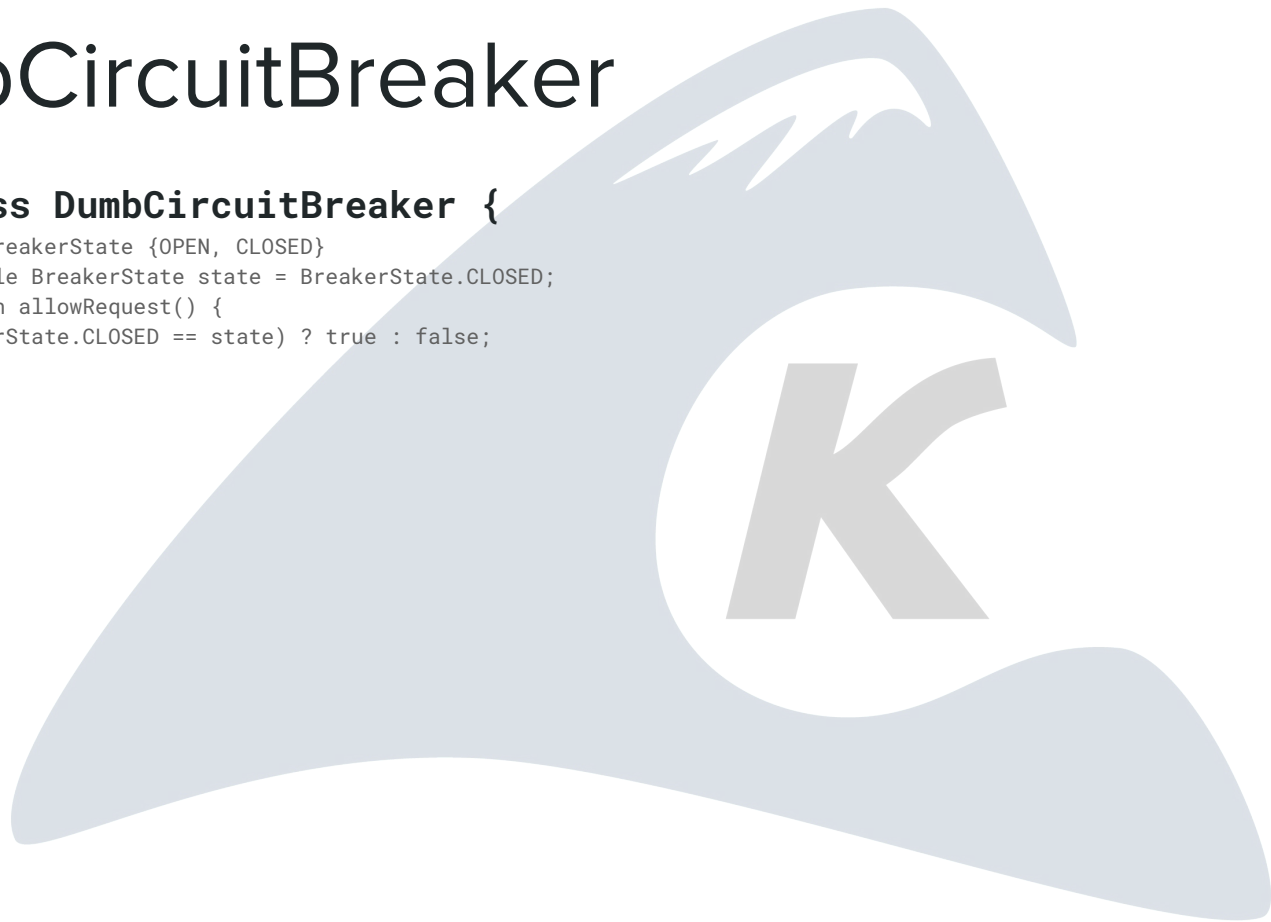
5

# DumbCircuitBreaker

```
public class DumbCircuitBreaker {  
    protected enum BreakerState {OPEN, CLOSED}  
    protected volatile BreakerState state = BreakerState.CLOSED;  
    protected boolean allowRequest() {  
        return (BreakerState.CLOSED == state) ? true : false;  
    }  
}
```

# DumbCircuitBreaker

```
public class DumbCircuitBreaker {  
    protected enum BreakerState {OPEN, CLOSED}  
    protected volatile BreakerState state = BreakerState.CLOSED;  
    protected boolean allowRequest() {  
        return (BreakerState.CLOSED == state) ? true : false;  
    }  
}
```



# DumbCircuitBreaker

```
public class DumbCircuitBreaker {  
    protected enum BreakerState {OPEN, CLOSED}  
    protected volatile BreakerState state = BreakerState.CLOSED;  
    protected boolean allowRequest() {  
        return (BreakerState.CLOSED == state) ? true : false;  
    }  
}
```



K

# DumbCircuitBreaker

```
public class DumbCircuitBreaker {  
    protected enum BreakerState {OPEN, CLOSED}  
    protected volatile BreakerState state = BreakerState.CLOSED;  
    protected boolean allowRequest() {  
        return (BreakerState.CLOSED == state) ? true : false;  
    }  
}
```



K

# DumbCircuitBreaker

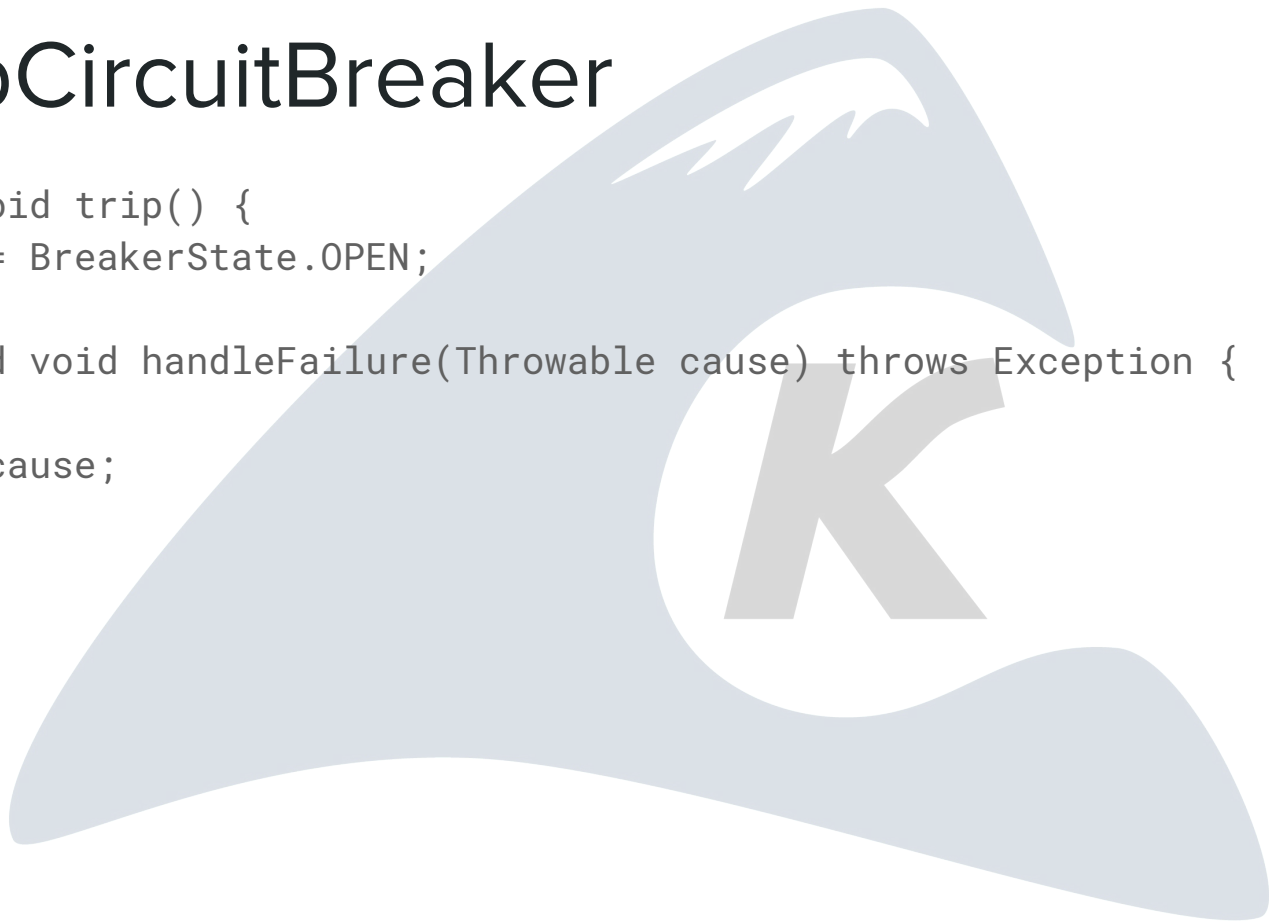
```
public class DumbCircuitBreaker {  
    protected enum BreakerState {OPEN, CLOSED}  
    protected volatile BreakerState state = BreakerState.CLOSED;  
    protected boolean allowRequest() {  
        return (BreakerState.CLOSED == state) ? true : false;  
    }
```



K

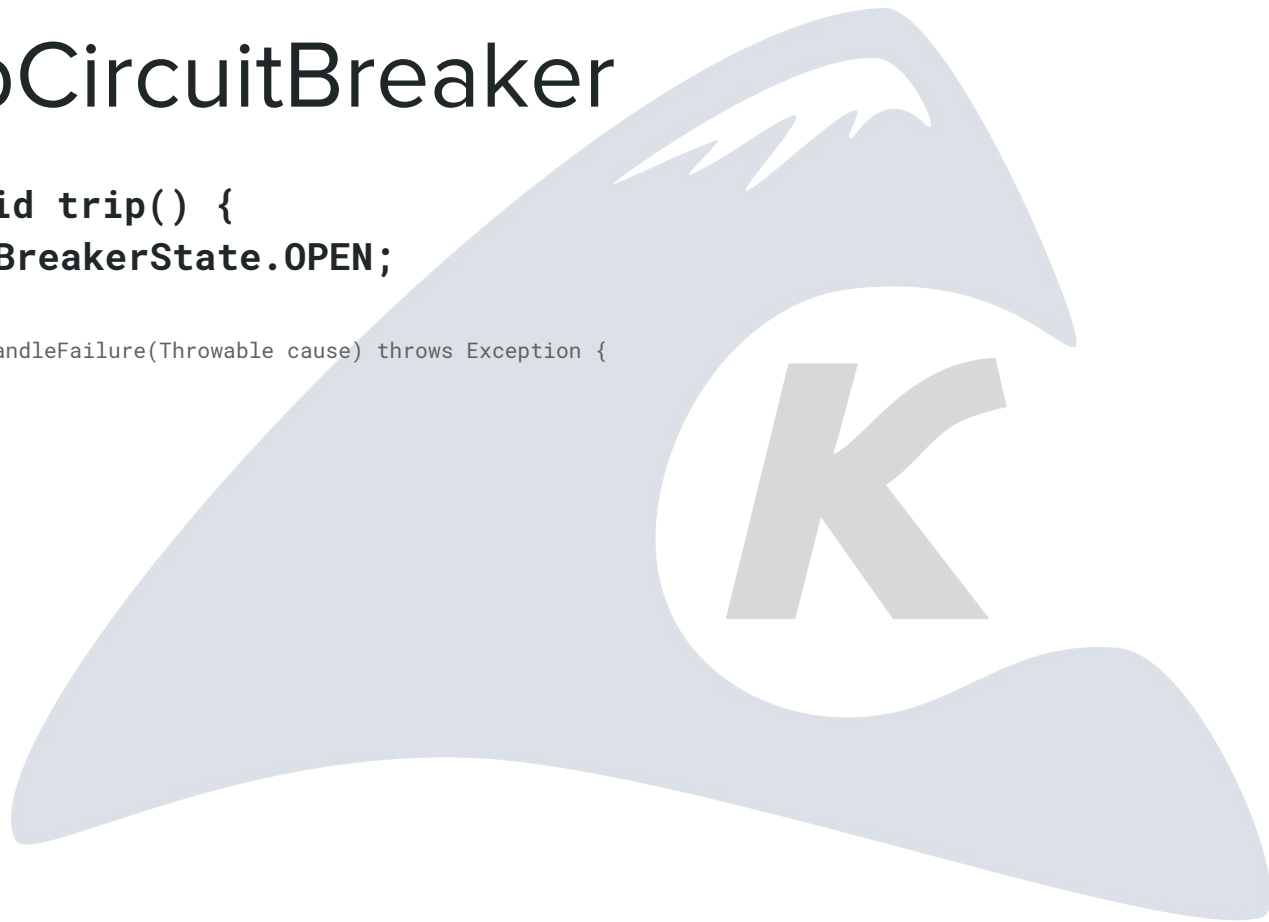
# DumbCircuitBreaker

```
public void trip() {  
    state = BreakerState.OPEN;  
}  
protected void handleFailure(Throwable cause) throws Exception {  
    trip();  
    throw cause;  
}
```



# DumbCircuitBreaker

```
public void trip() {  
    state = BreakerState.OPEN;  
}  
protected void handleFailure(Throwable cause) throws Exception {  
    trip();  
    throw cause;  
}
```

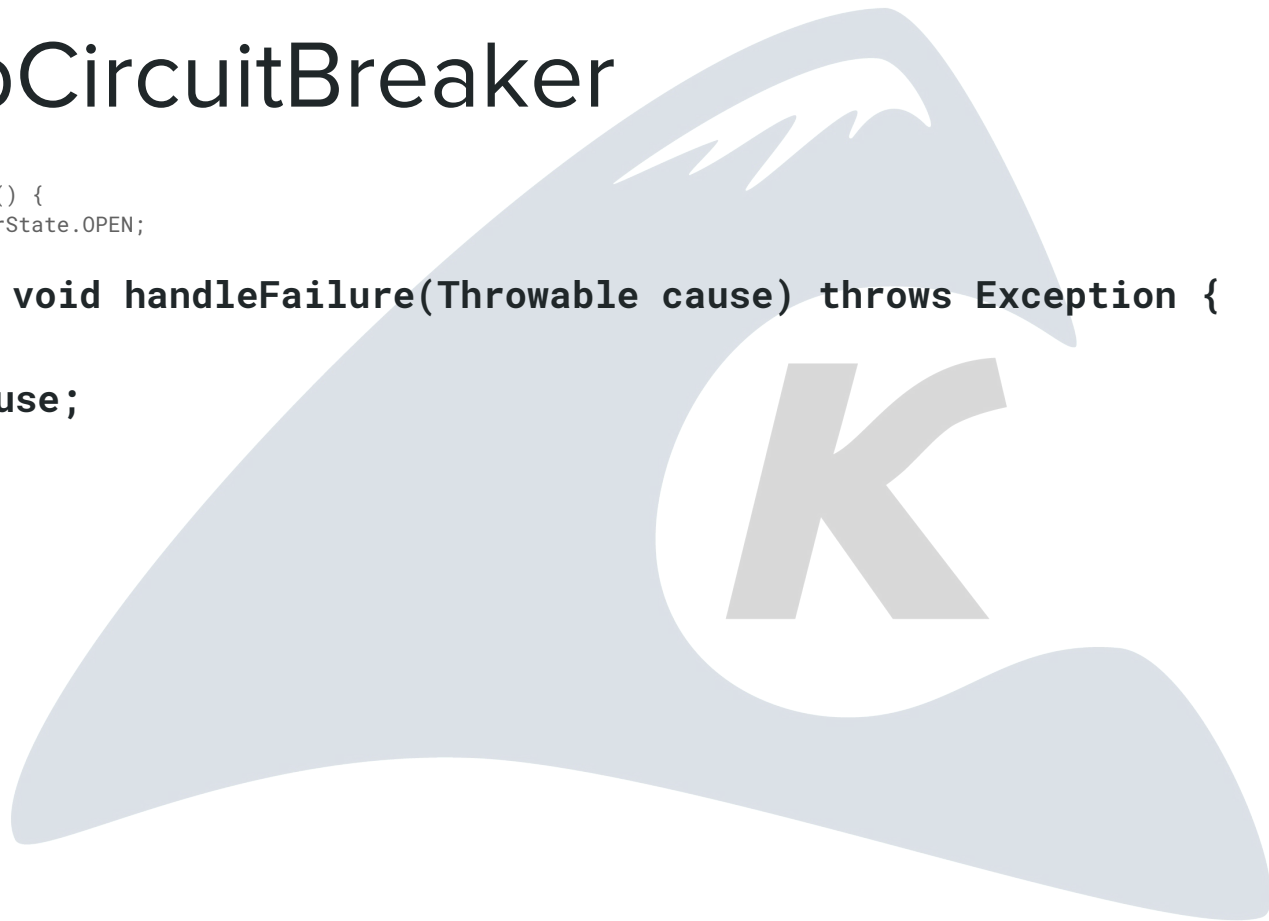




# DumbCircuitBreaker

```
public void trip() {  
    state = BreakerState.OPEN;  
}
```

```
protected void handleFailure(Throwable cause) throws Exception {  
    trip();  
    throw cause;  
}
```



# DumbCircuitBreaker

```
public <V> V invoke(Callable<V> c) throws Exception {  
    if (!allowRequest()) {  
        throw new CircuitBreakerException();  
    }  
    try {  
        V result = c.call();  
        return result;  
    } catch (Throwable cause) {  
        handleFailure(cause);  
    }  
    throw new IllegalStateException("not possible");  
}
```

# DumbCircuitBreaker

```
public <V> V invoke(Callable<V> c) throws Exception {  
    if (!allowRequest()) {  
        throw new CircuitBreakerException();  
    }  
    try {  
        V result = c.call();  
        return result;  
    } catch (Throwable cause) {  
        handleFailure(cause);  
    }  
    throw new IllegalStateException("not possible");  
}
```



# DumbCircuitBreaker

```
public <V> V invoke(Callable<V> c) throws Exception {  
    if (!allowRequest()) {  
        throw new CircuitBreakerException();  
    }  
    try {  
        V result = c.call();  
        return result;  
    } catch (Throwable cause) {  
        handleFailure(cause);  
    }  
    throw new IllegalStateException("not possible");  
}
```



# DumbCircuitBreaker

```
public <V> V invoke(Callable<V> c) throws Exception {  
    if (!allowRequest()) {  
        throw new CircuitBreakerException();  
    }  
    try {  
        V result = c.call();  
        return result;  
    } catch (Throwable cause) {  
        handleFailure(cause);  
    }  
    throw new IllegalStateException("not possible");  
}
```

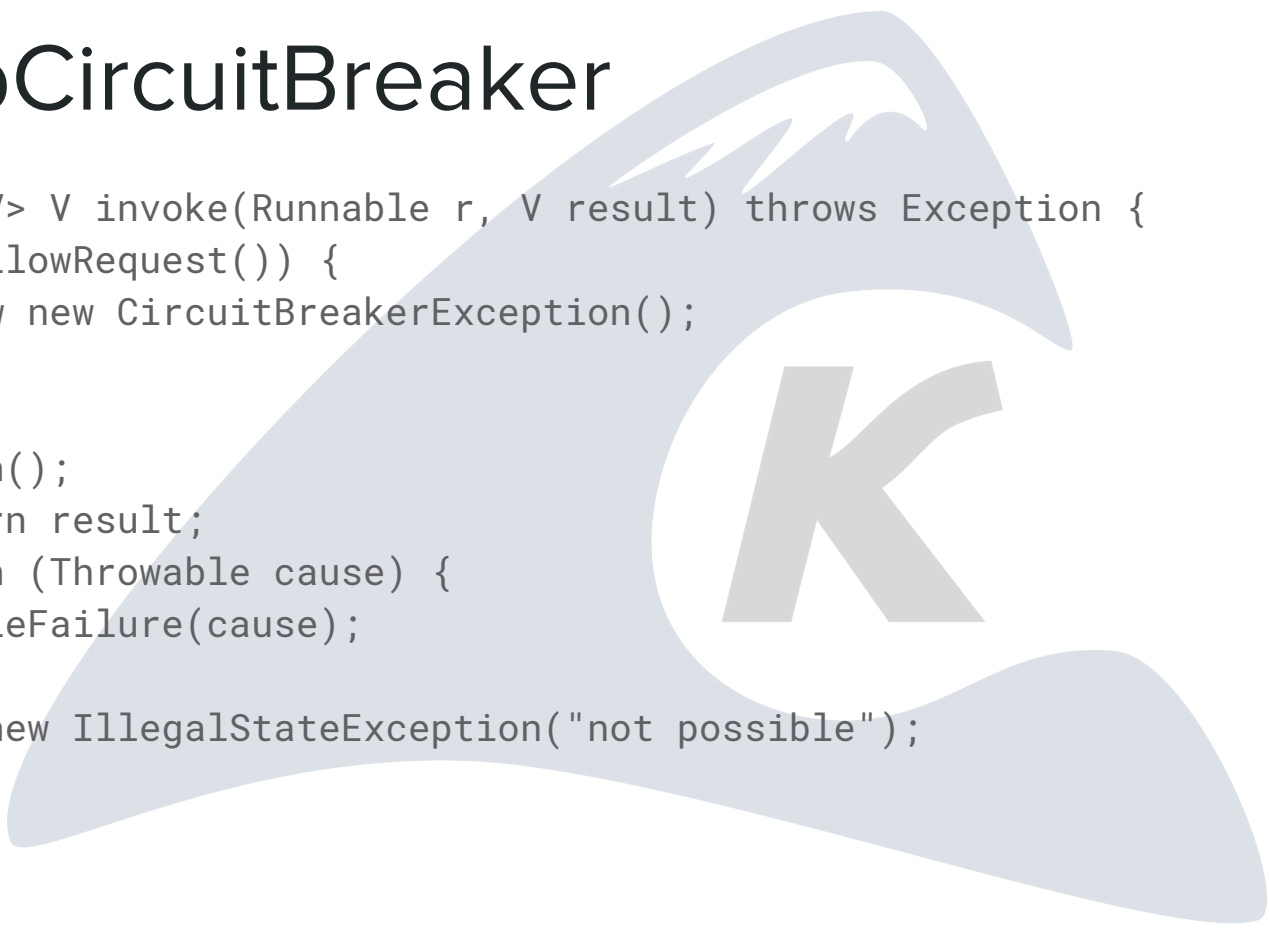


# DumbCircuitBreaker

```
public void invoke(Runnable r) throws Exception {  
    if (!allowRequest()) {  
        throw new CircuitBreakerException();  
    }  
    try {  
        r.run();  
        return;  
    } catch (Throwable cause) {  
        handleFailure(cause);  
    }  
    throw new IllegalStateException("not possible");  
}
```

# DumbCircuitBreaker

```
public <V> V invoke(Runnable r, V result) throws Exception {  
    if (!allowRequest()) {  
        throw new CircuitBreakerException();  
    }  
    try {  
        r.run();  
        return result;  
    } catch (Throwable cause) {  
        handleFailure(cause);  
    }  
    throw new IllegalStateException("not possible");  
}
```



# DumbCircuitBreaker

```
public class Service {  
    private DumbCircuitBreaker cb = new DumbCircuitBreaker();  
  
    public String doSomething(final Object arg) throws Exception {  
        return cb.invoke(new Callable<String>() {  
            public String call() {  
                // make the call ...  
            }  
        });  
    }  
}
```



# LessDumbCircuitBreaker

```
public void reset() {  
    state = BreakerState.CLOSED;  
    isAttemptLive = false;  
}  
protected enum BreakerState {  
    OPEN,  
    HALF_CLOSED,  
    CLOSED  
}  
protected AtomicLong lastFailure = new AtomicLong(0L);  
protected AtomicLong resetMillis = new AtomicLong(15 * 1000L);  
protected boolean isAttemptLive = false;
```

# LessDumbCircuitBreaker

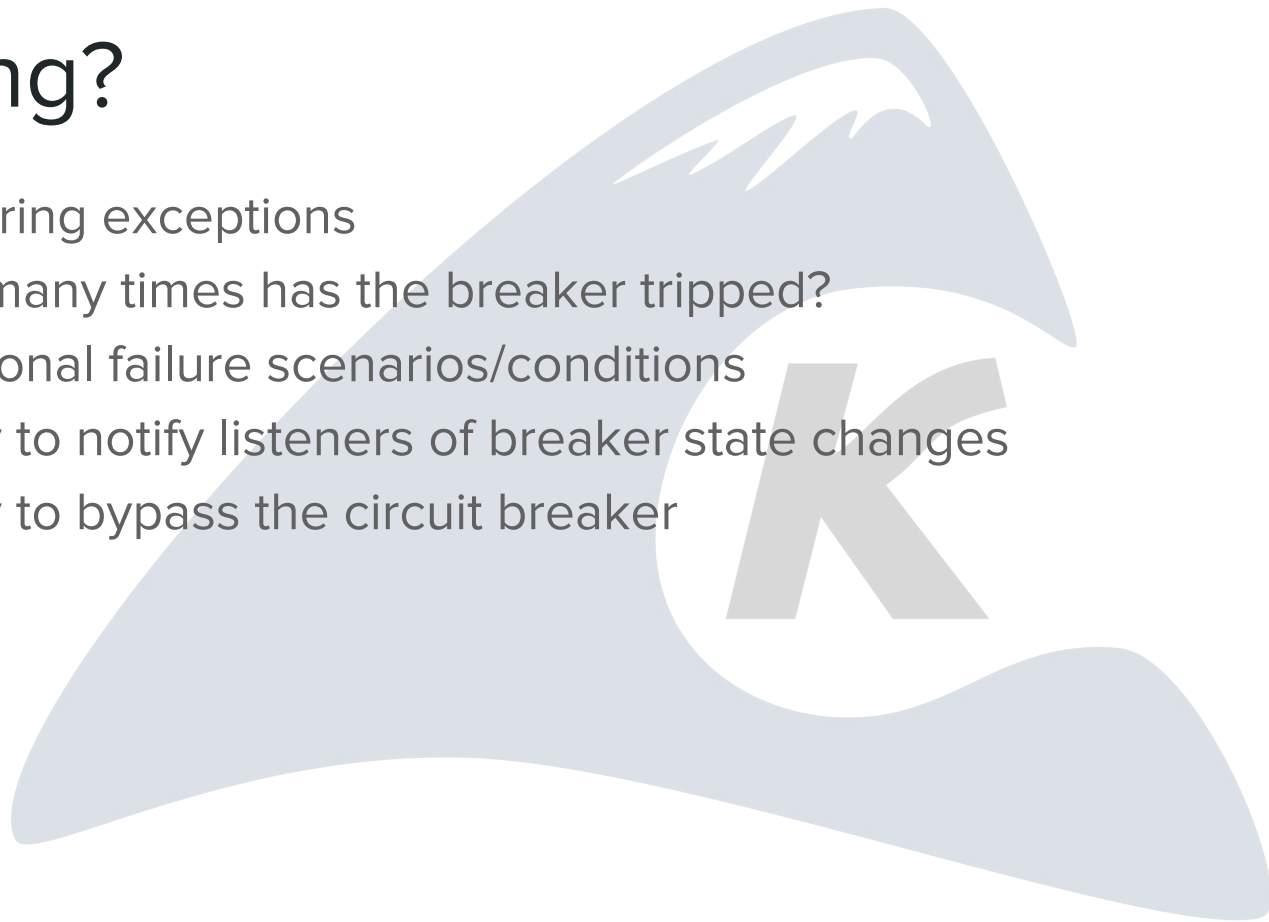
```
protected boolean allowRequest() {  
    if (BreakerState.CLOSED == state) {  
        return true;  
    }  
    if (BreakerState.OPEN == state &&  
        System.currentTimeMillis() - lastFailure.get() >= resetMillis.get()) {  
        state = BreakerState.HALF_CLOSED;  
    }  
  
    return canAttempt();  
}
```

# LessDumbCircuitBreaker

```
private synchronized boolean canAttempt() {  
    if (!(BreakerState.HALF_CLOSED == state) || isAttemptLive) {  
        return false;  
    }  
    return true;  
}  
  
public void trip() {  
    state = BreakerState.OPEN;  
    lastFailure.set(System.currentTimeMillis());  
    isAttemptLive = false;  
}
```

# Missing?

- Capturing exceptions
- How many times has the breaker tripped?
- Additional failure scenarios/conditions
- Ability to notify listeners of breaker state changes
- Ability to bypass the circuit breaker



# Agenda

Electricity	1
-------------	---

Software	2
----------	---

Implementation	3
----------------	---

Hystrix	4
---------	---

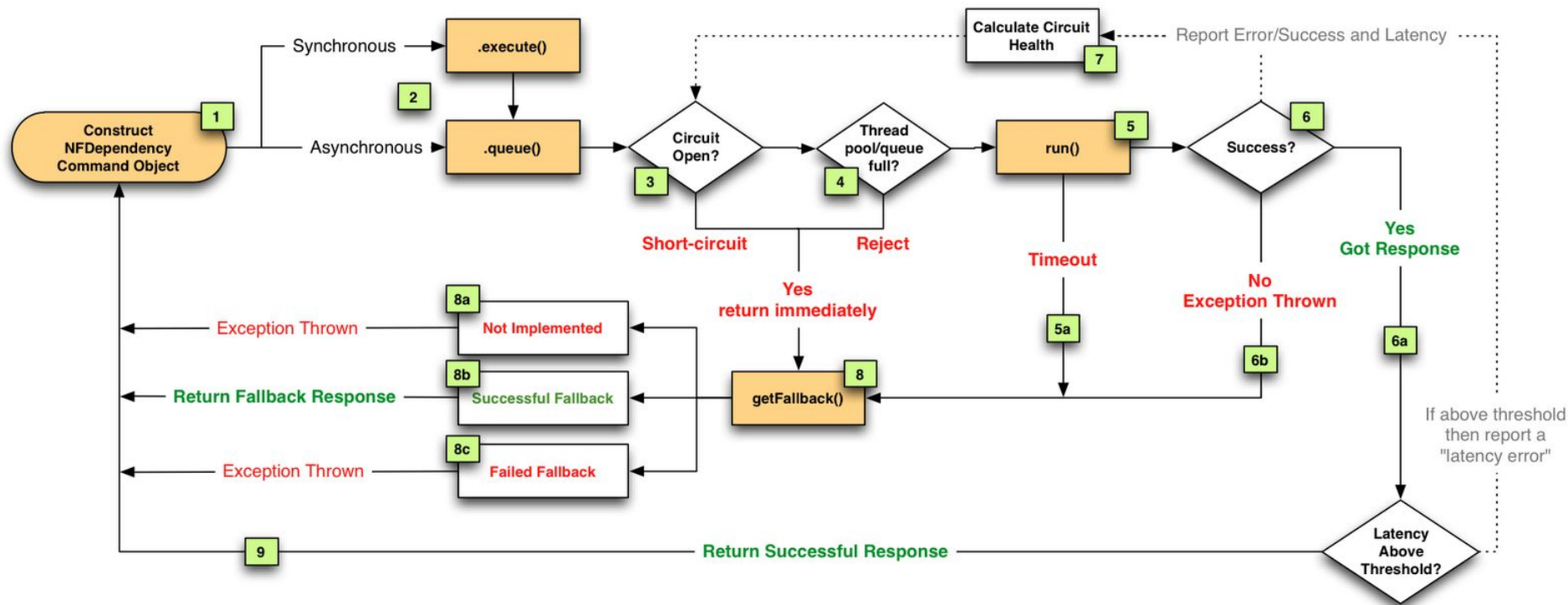
Summary	5
---------	---



# HYSTRIX

DEFEND YOUR APP

“Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.”



# How to Use

- Add to Maven

- `<dependency>`

```
<groupId>com.netflix.hystrix</groupId>
```

```
<artifactId>hystrix-core</artifactId>
```

```
<version>1.5.2</version>
```

```
</dependency>
```



# How to Use

- Extend HystrixCommand

- ```
public class CommandHelloWorld extends HystrixCommand<String> {  
  
    private final String name;  
  
    public CommandHelloWorld(String name) {  
        super(HystrixCommandGroupKey.Factory.asKey("ExampleGroup"));  
        this.name = name;  
    }  
  
    @Override  
    protected String run() {  
        return "Hello " + name + "!";  
    }  
}
```

# How to Use

- Use in code

- `String s = new CommandHelloWorld("Bob").execute();`
- `Future<String> s = new CommandHelloWorld("Bob").queue();`
- `Observable<String> s = new CommandHelloWorld("Bob").observe();`

# Spring Boot

- Add `@EnableCircuitBreaker` to Application class

- `@SpringBootApplication`

- `@EnableCircuitBreaker`

- `public class Application {`

- `public static void main(String[] args) {`

- `new SpringApplicationBuilder(Application.class).web(true).run(args);`

- `}`

- `}`

- Add `@HystrixCommand` to methods that might fail

# Spring Boot

@Component

```
public class StoreIntegration {
```

```
    @HystrixCommand(fallbackMethod = "defaultStores")
```

```
    public Object getStores(Map<String, Object> parameters) {
```

```
        //do stuff that might fail
```

```
    }
```

```
    public Object defaultStores(Map<String, Object> parameters) {
```

```
        return /* something useful */;
```

```
    }
```

```
}
```

# Defaults

- Isolation: Thread
- Timeout: 1 s, interrupt
- Max concurrent requests: 10 (both command and fallback)
- Request Volume Threshold: 20 (in 10 s window)
- Sleep Window: 5 s
- Error Threshold: 50%
- 20 failures in 5 s

# Properties

```
@HystrixCommand(commandProperties = {
    @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "500")
},
    threadPoolProperties = {
        @HystrixProperty(name = "coreSize", value = "30"),
        @HystrixProperty(name = "maxQueueSize", value = "101"),
        @HystrixProperty(name = "keepAliveTimeMinutes", value = "2"),
        @HystrixProperty(name = "queueSizeRejectionThreshold", value = "15"),
        @HystrixProperty(name = "metrics.rollingStats.numBuckets", value = "12"),
        @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "1440")
    })
public User getUserById(String id) {
    return userResource.getUserById(id);
}
```

# Dashboard



# Agenda

Electricity

1

Software

2

Implementation

3

Hystrix

4

Summary

5



# Sources

- Nygard, Michael T. *Release It!* 30 Mar 2007
- Timms, Simon. *Mastering JavaScript Design Patterns*. 21 Nov 2014
- Newman, Sam. *Building Microservices*. 10 Feb 2015
- Paramount Pictures. *Wayne's World*. 14 Feb 1992
- Fowler, Martin. CircuitBreaker. 6 Mar 2014. <http://martinfowler.com/bliki/CircuitBreaker.html>
- Comcast jrugged source code. <https://github.com/Comcast/jrugged>
- Netflix Hystrix Readme. <https://github.com/Netflix/Hystrix>

# Sources

- Christensen, Ben. Fault Tolerance in a High Volume, Distributed System. 29 Feb 2012. <http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html>
- Schmaus, Ben. Making the Netflix API More Resilient. 8 Dec 2011. <http://techblog.netflix.com/2011/12/making-netflix-api-more-resilient.html>
- Spring Cloud Netflix documentation. <http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html>

# Summary

- Like a physical circuit breaker, the circuit breaker pattern allows a subsystem to fail without destroying the system
- Failure is inevitable, be prepared for it
- Don't implement your own, use a trusted implementation