Fault tolerance made easy

A head-start to resilient software design

Uwe Friedrichsen (codecentric AG) – QCon London – 5. March 2014

@ufried

Uwe Friedrichsen | uwe.friedrichsen@codecentric.de | http://slideshare.net/ufried | http://ufried.tumblr.com

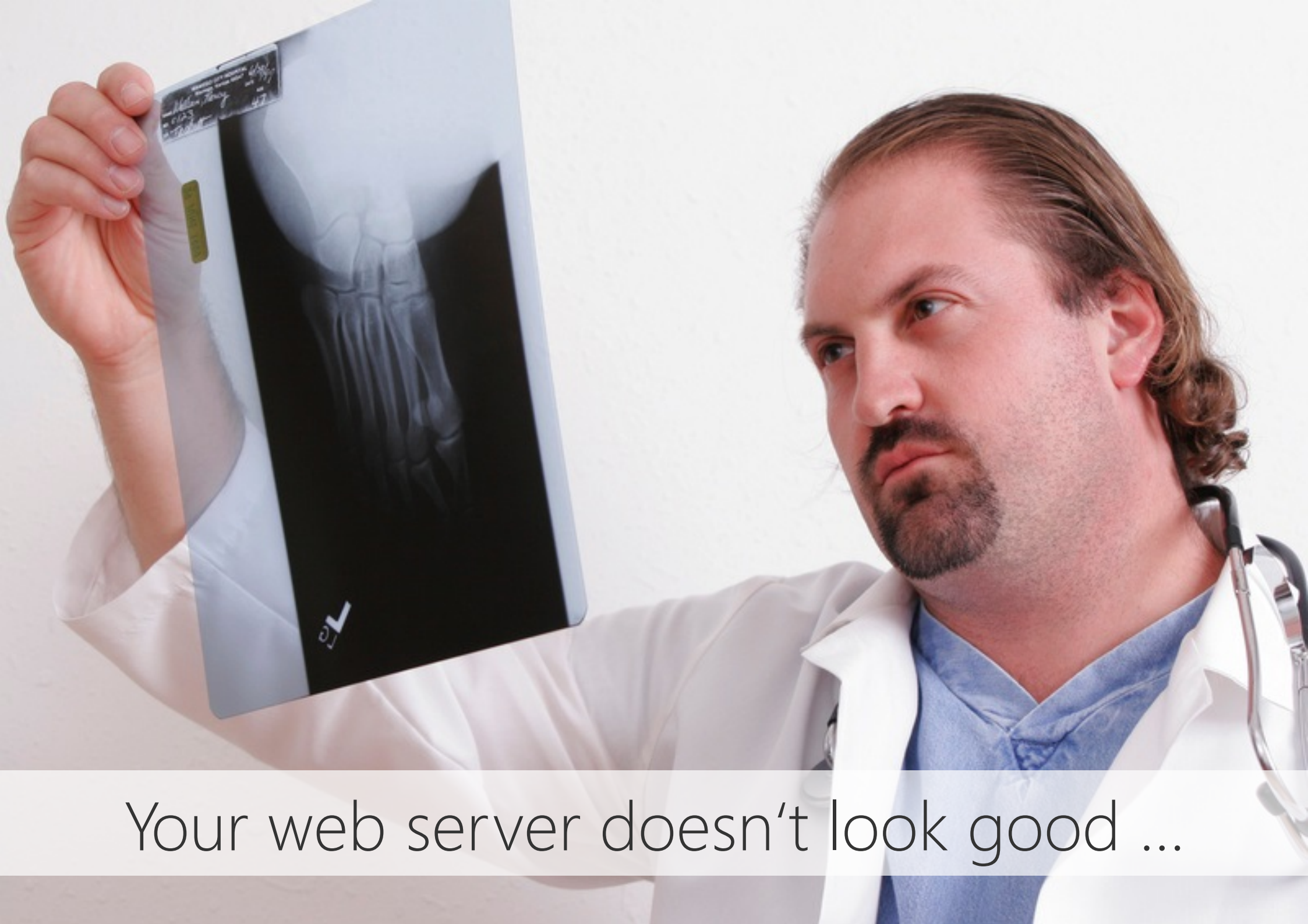

It’s all about production!

Production

Availability

Resilience

Fault Tolerance



Your web server doesn't look good …

Pattern #1

Timeouts

Timeouts (1)

```
// Basics
myObject.wait();  // Do not use this by default
myObject.wait(TIMEOUT);  // Better use this

// Some more basics
myThread.join();  // Do not use this by default
myThread.join(TIMEOUT);  // Better use this
```

Timeouts (2)

```
// Using the Java concurrent library
Callable<MyActionResult> myAction = <My Blocking Action>

ExecutorService executor = Executors.newSingleThreadExecutor();
Future<MyActionResult> future = executor.submit(myAction);
MyActionResult result = null;

try {
    result = future.get();  // Do not use this by default
    result = future.get(TIMEOUT, TIMEUNIT);  // Better use this
} catch (TimeoutException e) {  // Only thrown if timeouts are used
    ...
} catch (...) {
    ...
}
```

Timeouts (3)

```
// Using Guava SimpleTimeLimiter
Callable<MyActionResult> myAction = <My Blocking Action>

SimpleTimeLimiter limiter = new SimpleTimeLimiter();
MyActionResult result = null;

try {
    result =
        limiter.callWithTimeout(myAction, TIMEOUT, TIMEUNIT, false);
} catch (UncheckedTimeoutException e) {
    ...
} catch (...) {
    ...
}
```

Determining Timeout Duration

Configurable Timeouts

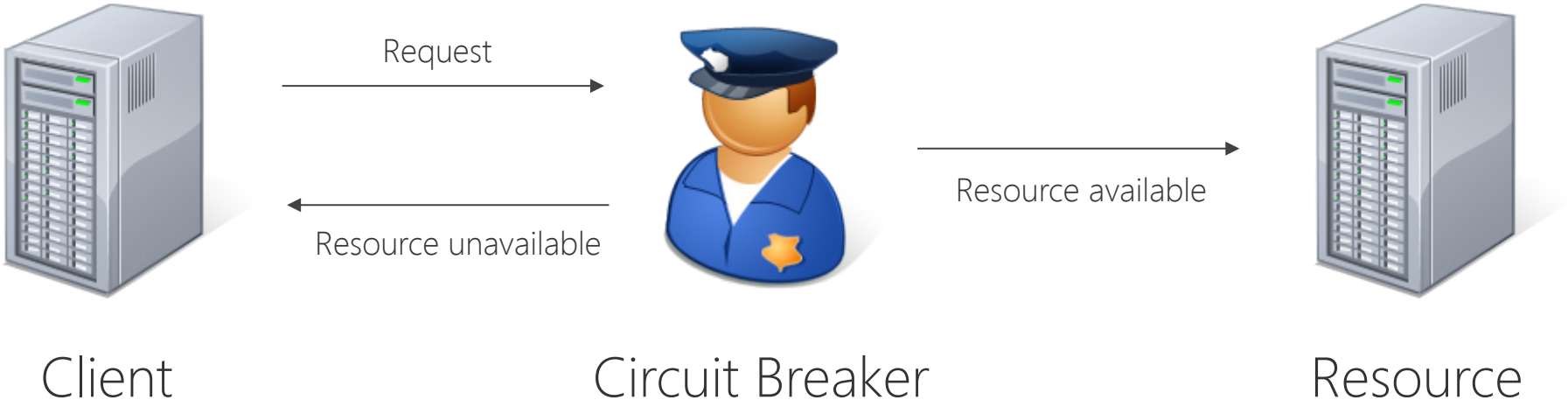
Self-Adapting Timeouts

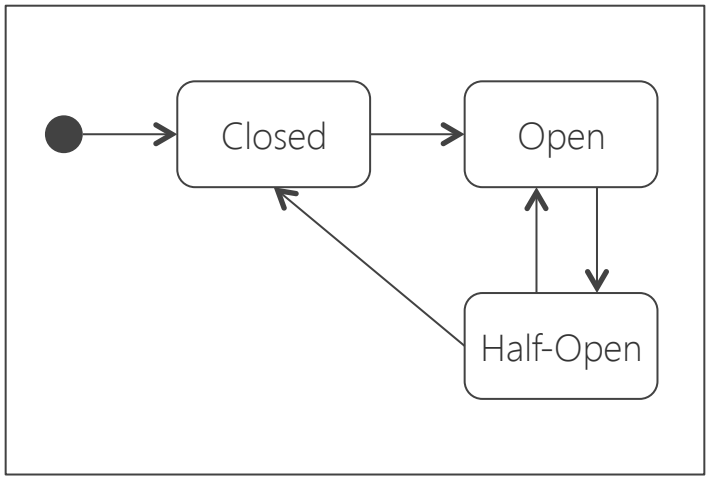
Timeouts in JavaEE Containers

Pattern #2

Circuit Breaker

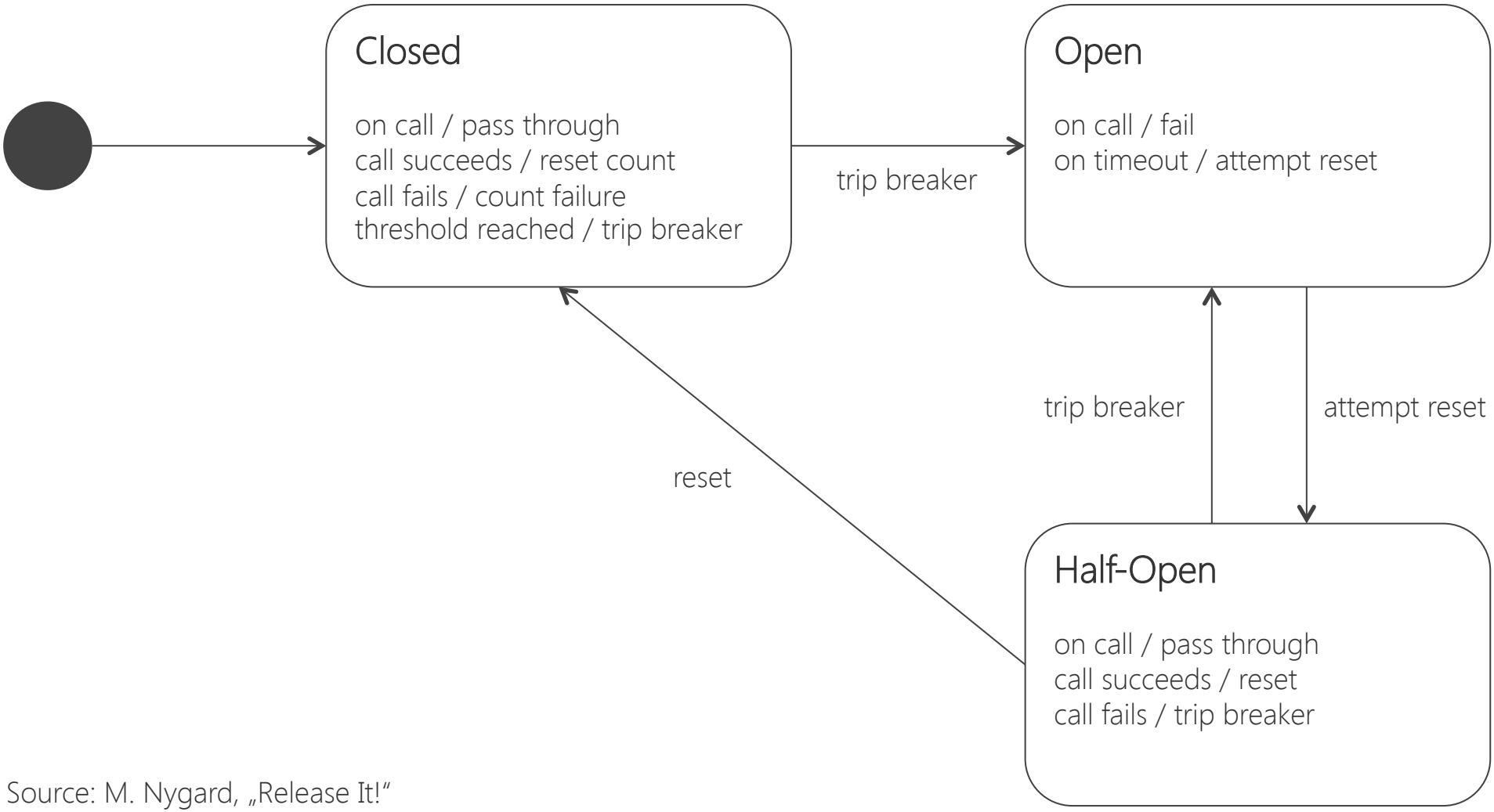
Circuit Breaker (1)





Lifecycle

Circuit Breaker (2)



Source: M. Nygard, „Release It!“

Circuit Breaker (3)

```
public class CircuitBreaker implements MyResource {
    public enum State { CLOSED, OPEN, HALF_OPEN }

    final MyResource resource;
    State state;
    int counter;
    long tripTime;

    public CircuitBreaker(MyResource r) {
        resource = r;
        state = CLOSED;
        counter = 0;
        tripTime = 0L;
    }

    ...
```

Circuit Breaker (4)

```
...
public Result access(...) {  // resource access
    Result r = null;

    if (state == OPEN) {
        checkTimeout();
        throw new ResourceUnavailableException();
    }

    try {
        r = resource.access(...);  // should use timeout
    } catch (Exception e) {
        fail();
        throw e;
    }
    success();
    return r;
}
...
```

Circuit Breaker (5)

```
...

private void success() {
    reset();
}

private void fail() {
    counter++;
    if (counter > THRESHOLD) {
        tripBreaker();
    }
}

private void reset() {
    state = CLOSED;
    counter = 0;
}

...
```

Circuit Breaker (6)

```
    ...

    private void tripBreaker() {
        state = OPEN;
        tripTime = System.currentTimeMillis();
    }

    private void checkTimeout() {
        if ((System.currentTimeMillis - tripTime) > TIMEOUT) {
            state = HALF_OPEN;
            counter = THRESHOLD;
        }
    }

    public State getState()
        return state;
    }
}
```

Thread-Safe Circuit Breaker

Failure Types

Tuning Circuit Breakers

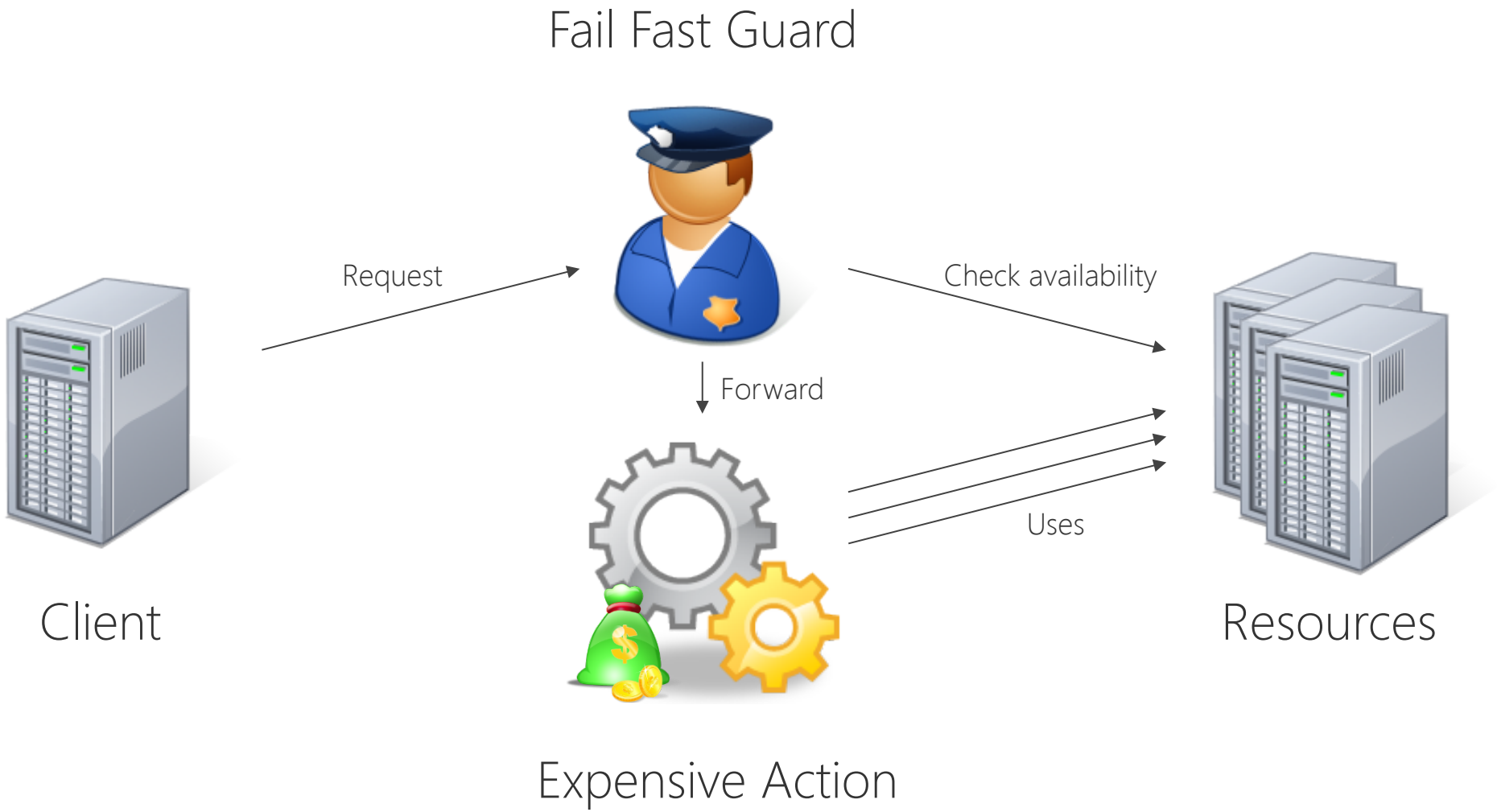
Available Implementations

Pattern #3

Fail Fast

Fail Fast (1)

Fail Fast (2)


Fail Fast Guard
Request
Check availability
Forward
Uses
Client
Resources
Expensive Action

Fail Fast (3)

```
public class FailFastGuard {
    private FailFastGuard() {}

    public static void checkResources(Set<CircuitBreaker> resources) {
        for (CircuitBreaker r : resources) {
            if (r.getState() != CircuitBreaker.CLOSED) {
                throw new ResourceUnavailableException(r);
            }
        }
    }
}
```

Fail Fast (4)

```
public class MyService {
    Set<CircuitBreaker> requiredResources;

    // Initialize resources
    ...

    public Result myExpensiveAction(...) {
        FailFastGuard.checkResources(requiredResources);

        // Execute core action
        ...
    }
}
```

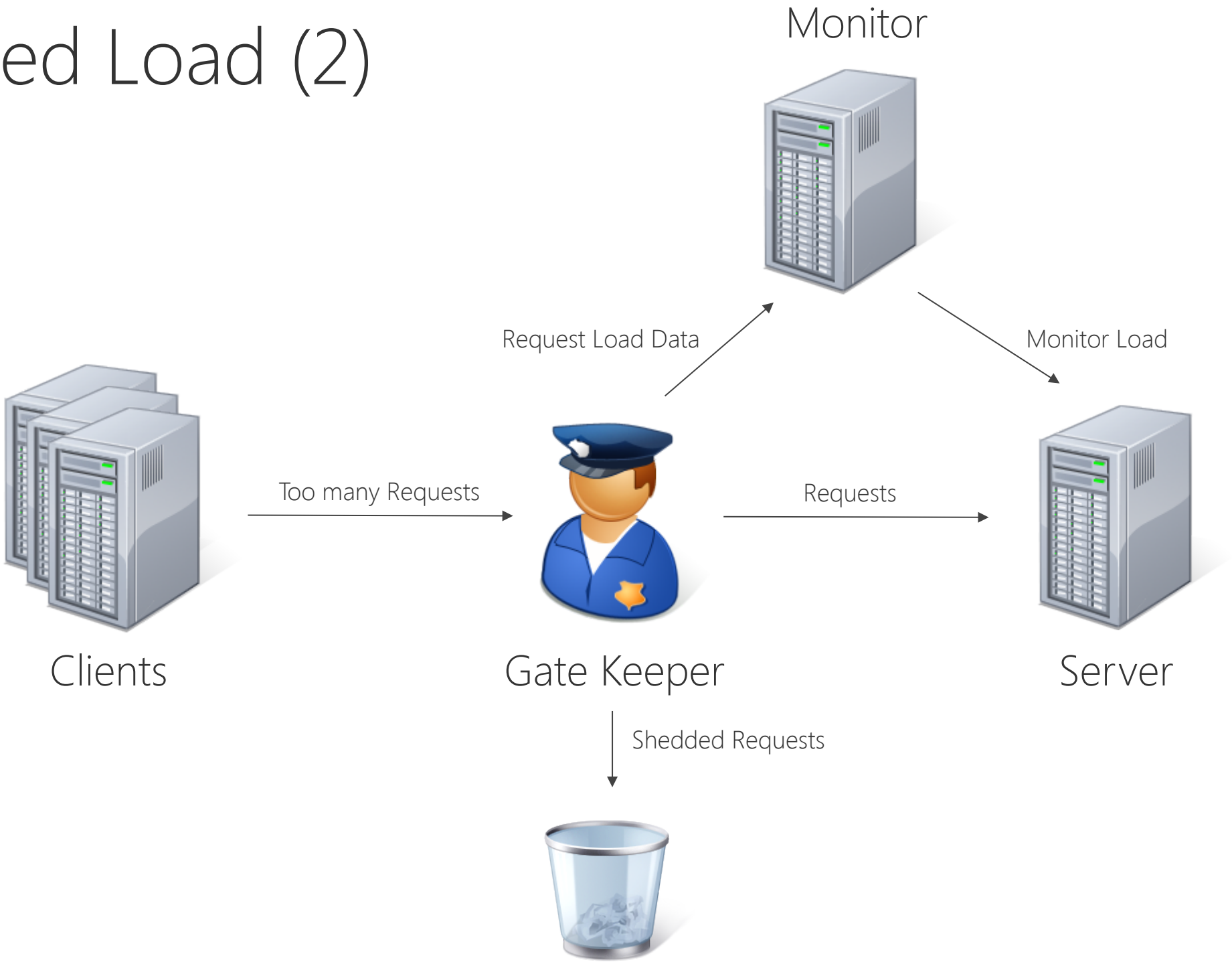
The dreaded SiteTooSuccessfulException ...

Pattern #4

Shed Load

Shed Load (1)

Shed Load (2)


Monitor
Request Load Data
Monitor Load
Too many Requests
Requests
Clients
Gate Keeper
Server
Shedded Requests

Shed Load (3)

```
public class ShedLoadFilter implements Filter {
    Random random;

    public void init(FilterConfig fc) throws ServletException {
        random = new Random(System.currentTimeMillis());
    }

    public void destroy() {
        random = null;
    }

    ...
```

Shed Load (4)

```
...

public void doFilter(ServletRequest request,
                     ServletResponse response,
                     FilterChain chain)
                     throws java.io.IOException, ServletException {
    int load = getLoad();
    if (shouldShed(load)) {
        HttpServletResponse res = (HttpServletResponse)response;
        res.setIntHeader("Retry-After", RECOMMENDATION);
        res.sendError(HttpServletResponse.SC_SERVICE_UNAVAILABLE);
        return;
    }
    chain.doFilter(request, response);
}

...
```

Shed Load (5)

```
    ...

    private boolean shouldShed(int load) {  // Example implementation
        if (load < THRESHOLD) {
            return false;
        }
        double shedBoundary =
            ((double)(load - THRESHOLD))/
            ((double)(MAX_LOAD - THRESHOLD));
        return random.nextDouble() < shedBoundary;
    }
}
```

Shed Load (6)


www.catonmat.net/http-proxy-in-nodejs/
Programming
55 Comments
April 28, 2010
A HTTP Proxy Server in 20 Lines of node.js Code
Microsoft Cloud Computing
Virtualisieren Sie jetzt: Mit der Microsoft Private Cloud! Infos hier
microsoft.com/virtualisierung
AdChoices
var http = require('http');
http.createServer(function(request, response) {
var proxy = http.createClient(80, request.headers['host'])
var proxy_request = proxy.request(request.method, request.url, request.h
proxy_request.addListener('response', function (proxy_response) {
proxy_response.addListener('data', function(chunk) {
response.write(chunk, 'binary');
});
proxy_response.addListener('end', function() {
response.end();
});
response.writeHead(proxy_response.statusCode, proxy_response.headers);
});
request.addListener('data', function(chunk) {
proxy_request.write(chunk, 'binary');
});
request.addListener('end', function() {
proxy_request.end();
});
}).listen(8080);
This is just amazing. In 20 lines of node.js code and 10 minutes of time I was able to write a HTTP proxy. And it scales well, too. It's not a blocking HTTP proxy, it's event driven and asynchronous, meaning hundreds of people can use simultaneously and it will work well.
Peteris Krumins' blog about programming, hacking, software reuse, software ideas, computer security, google and technology.
Reach me at:
peter@catonmat.net
Or meet me on:
Twitter
Facebook
Plurk
more
Subscribe to my posts:
Subscribe through an RSS feed:
15383 readers
BY FEEDBURNER
(what is rss?)
Subscribe through email:
Enter your email address:
Subscribe
Delivered by FeedBurner
Server Sponsors
I am being sponsored by Syntress since 2007! They bought me an amazing dedicated server to run catonmat on. If you're looking web services in Chicago

Shed Load (7)

nginx.org/en/docs/http/ngx_http_limit_conn_module.html

Module ngx_http_limit_conn_module

The `ngx_http_limit_conn_module` module allows to limit the number of connections per defined key, in particular, the number of connections from a single IP address.

Not all connections are counted; only those that have requests currently being processed by the server, in which request header has been fully read.

Example Configuration

```
http {
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    ...

    server {

        ...

        location /download/ {
            limit_conn addr 1;
        }
```

Directives

syntax: `limit_conn` *`zone number;`*
default: —
context: `http`, `server`, `location`

Sets a shared memory zone and the maximum allowed number of connections for a given key value. When this limit is exceeded, the server will return error 503 (Service Temporarily Unavailable) in reply to a request. For example, the directives

Shedding Strategy

Retrieving Load

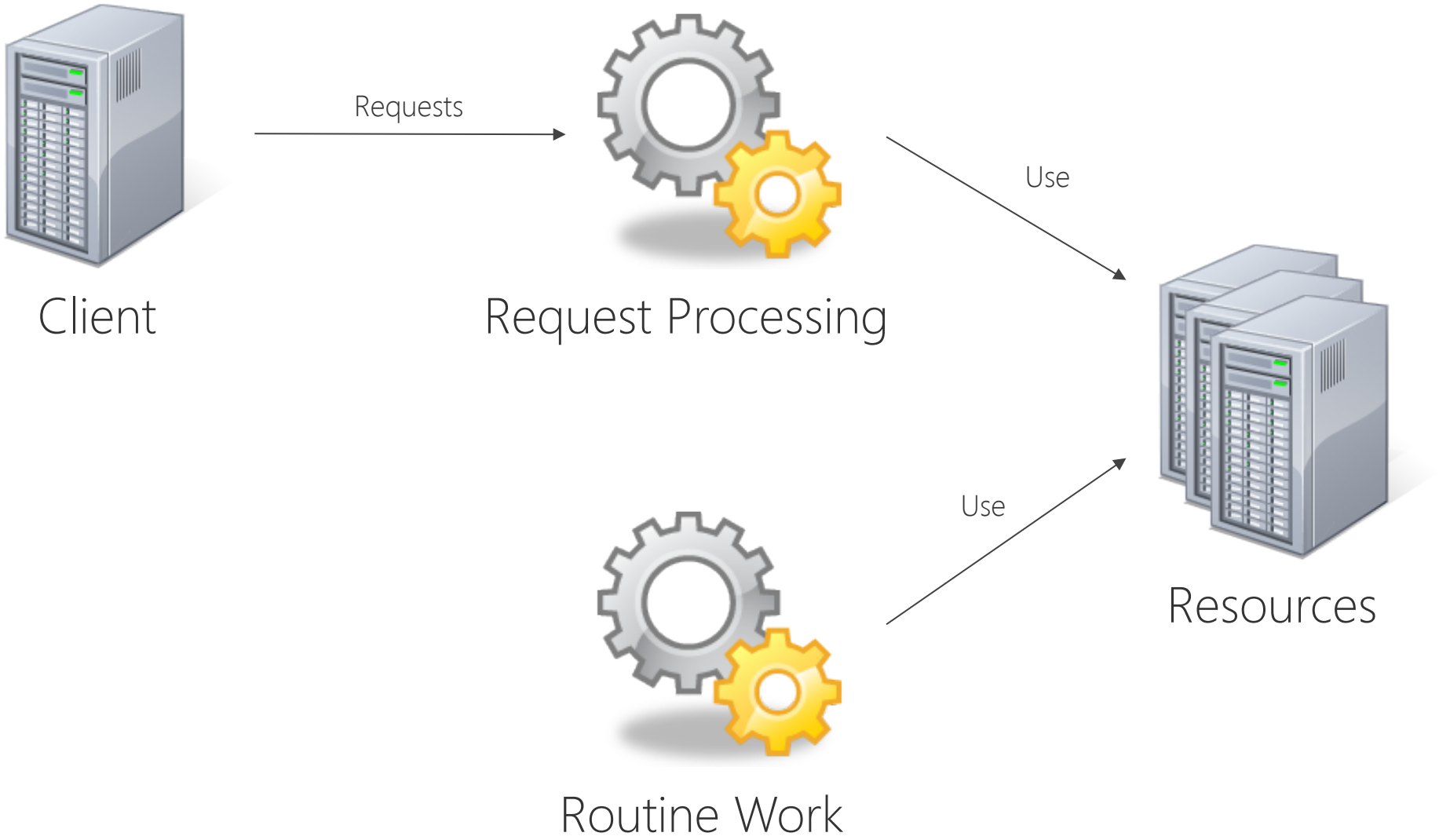
Tuning Load Shedders

Alternative Strategies

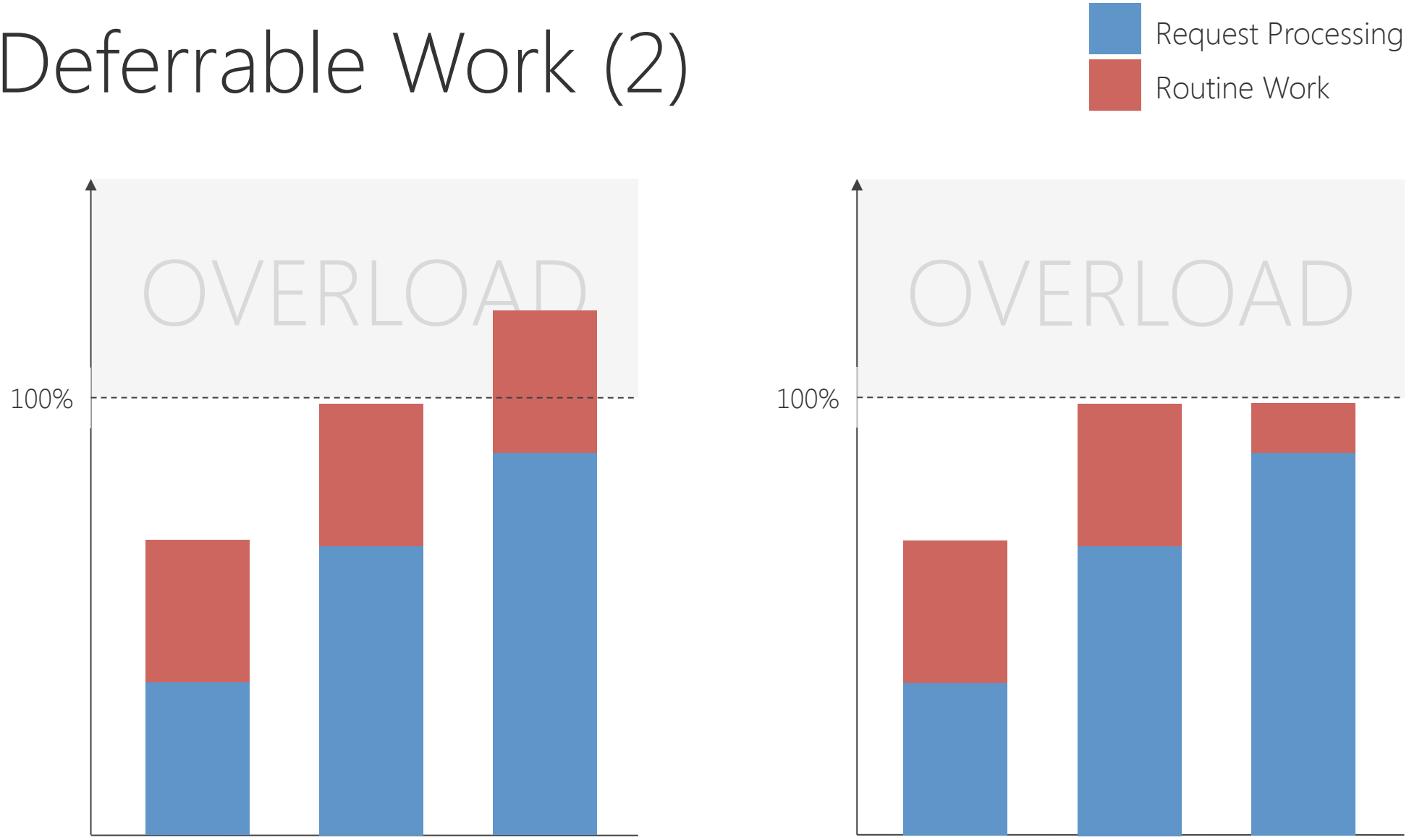
Pattern #5

Deferrable Work

Deferrable Work (1)



Deferrable Work (2)



Without
Deferrable Work

With
Deferrable Work

Deferrable Work (3)

```
// Do or wait variant
ProcessingState state = initBatch();
while(!state.done()) {
    int load = getLoad();
    if (load > THRESHOLD) {
        waitFixedDuration();
    } else {
        state = processNext(state);
    }
}

void waitFixedDuration() {
    Thread.sleep(DELAY);  // try-catch left out for better readability
}
```

Deferrable Work (4)

```
// Adaptive load variant
ProcessingState state = initBatch();
while(!state.done()) {
    waitLoadBased();
    state = processNext(state);
}

void waitLoadBased() {
    int load = getLoad();
    long delay = calcDelay(load);
    Thread.sleep(delay);  // try-catch left out for better readability
}

long calcDelay(int load) {  // Simple example implementation
    if (load < THRESHOLD) {
        return 0L;
    }
    return (load - THRESHOLD) * DELAY_FACTOR;
}
```

Delay Strategy

Retrieving Load

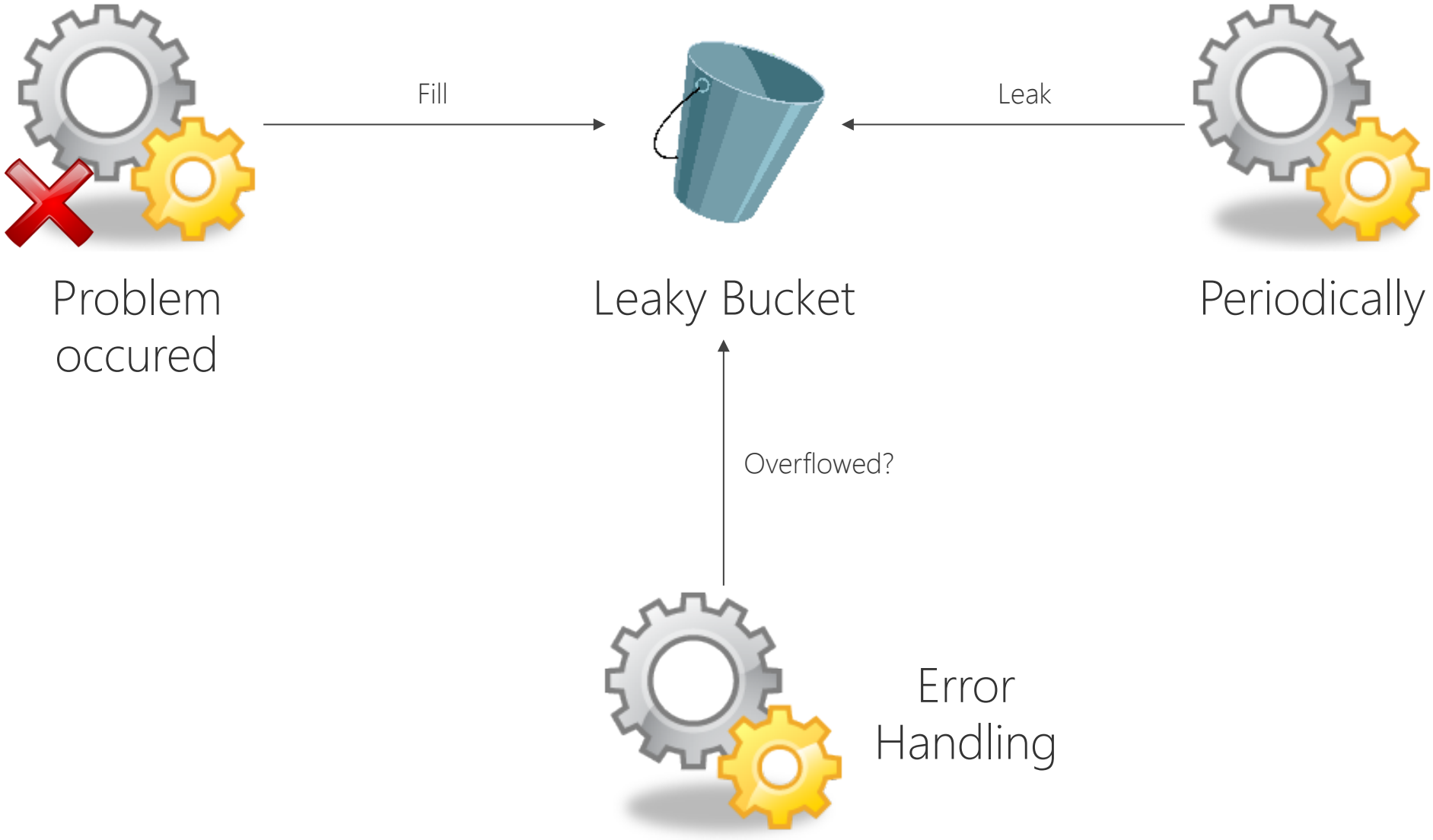
Tuning Deferrable Work

I can hardly hear you …

Pattern #6

Leaky Bucket

Leaky Bucket (1)



Leaky Bucket (2)

```
public class LeakyBucket { // Very simple implementation
    final private int capacity;
    private int level;
    private boolean overflow;

    public LeakyBucket(int capacity) {
        this.capacity = capacity;
        drain();
    }

    public void drain () {
        this.level = 0;
        this.overflow = false;
    }

    ...
```

Leaky Bucket (3)

```
    ...

    public void fill() {
        level++;
        if (level > capacity) {
            overflow = true;
        }
    }

    public void leak() {
        level--;
        if (level < 0) {
            level = 0;
        }
    }

    public boolean overflowed() {
        return overflow;
    }
}
```

Thread-Safe Leaky Bucket

Leaking strategies

Tuning Leaky Bucket

Available Implementations

Pattern #7

Limited Retries

Limited Retries (1)

```
// doAction returns true if successful, false otherwise

// General pattern
boolean success = false
int tries = 0;
while (!success && (tries < MAX_TRIES)) {
    success = doAction(...);
    tries++;
}

// Alternative one-retry-only variant
success = doAction(...) || doAction(...);
```

Idempotent Actions

Closures / Lambdas

Tuning Retries

More Patterns

- Complete Parameter Checking
- Marked Data
- Routine Audits

Further reading

1. Michael T. Nygard, Release It!,
   Pragmatic Bookshelf, 2007

2. Robert S. Hanmer,
   Patterns for Fault Tolerant Software,
   Wiley, 2007

3. James Hamilton, On Designing and
   Deploying Internet-Scale Services,
   21st LISA Conference 2007

4. Andrew Tanenbaum, Marten van Steen,
   Distributed Systems – Principles and
   Paradigms,
   Prentice Hall, 2nd Edition, 2006

It’s all about production!

@ufried

Uwe Friedrichsen | uwe.friedrichsen@codecentric.de | http://slideshare.net/ufried | http://ufried.tumblr.com