31.8.–3.9.2015
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Ich habe fertig!

Production-ready statt Feature-complete

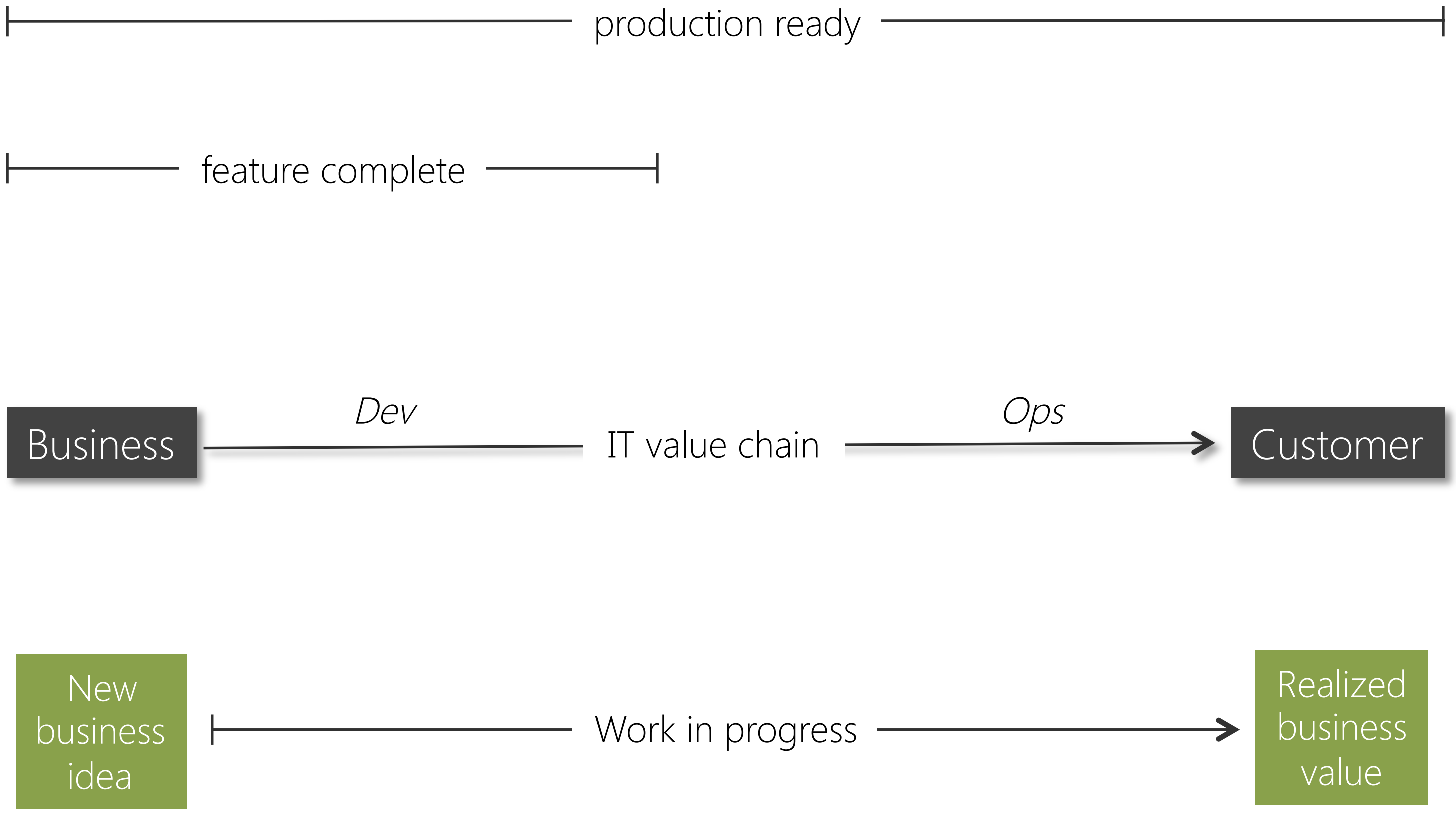
Uwe Friedrichsen

codecentric AG

@ufried

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

Why this talk?



production ready

feature complete

Business

Dev

IT value chain

Ops

Customer

New business idea

Work in progress

Realized business value

It’s all about production!

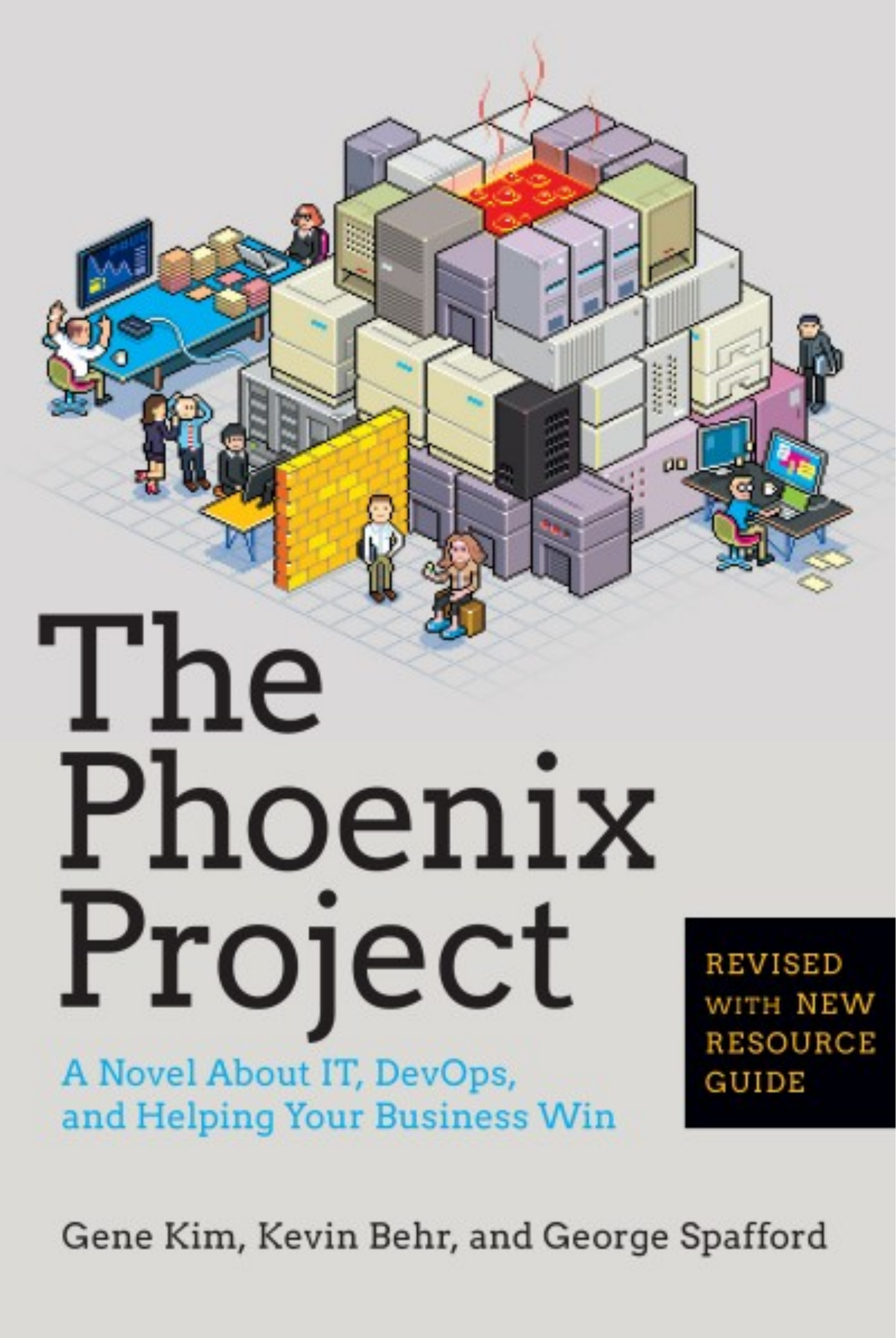
But before we talk about production …

… let’s talk about DevOps briefly

What is that DevOps thing anyway?

Let’s check the “DevOps bible”

http://itrevolution.com/books/phoenix-project-devops-book/


The Phoenix Project
A Novel About IT, DevOps, and Helping Your Business Win
REVISED WITH NEW RESOURCE GUIDE
Gene Kim, Kevin Behr, and George Spafford

DevOps in a nutshell

The 3 ways of DevOps

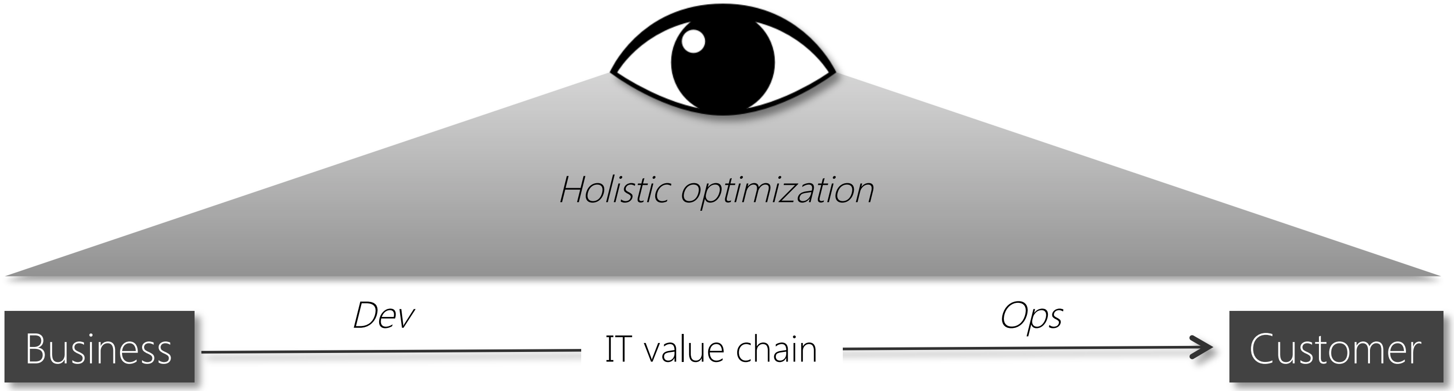
Systems thinking

Amplify feedback loops

Culture of continual experimentation & learning

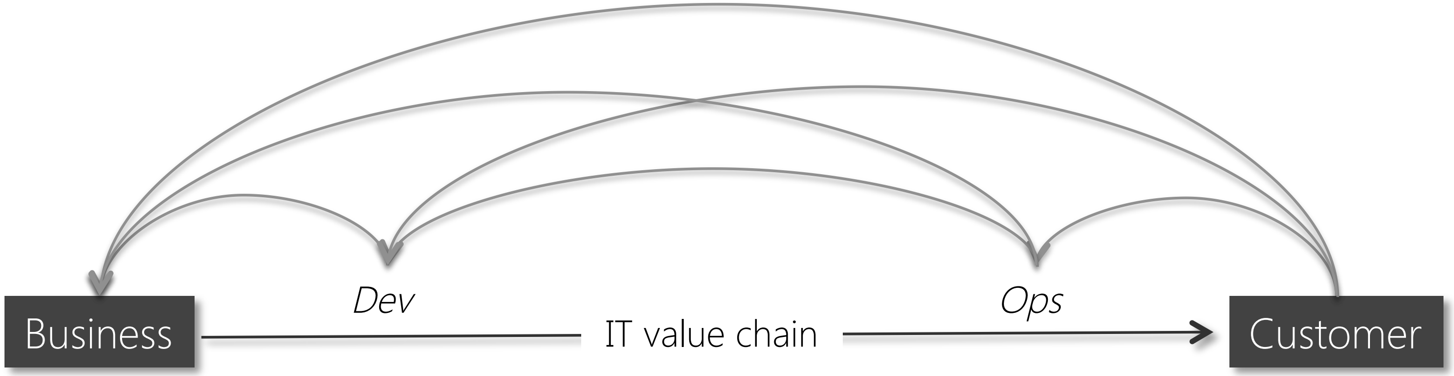
http://itrevolution.com/the-three-ways-principles-underpinning-devops/

Systems thinking



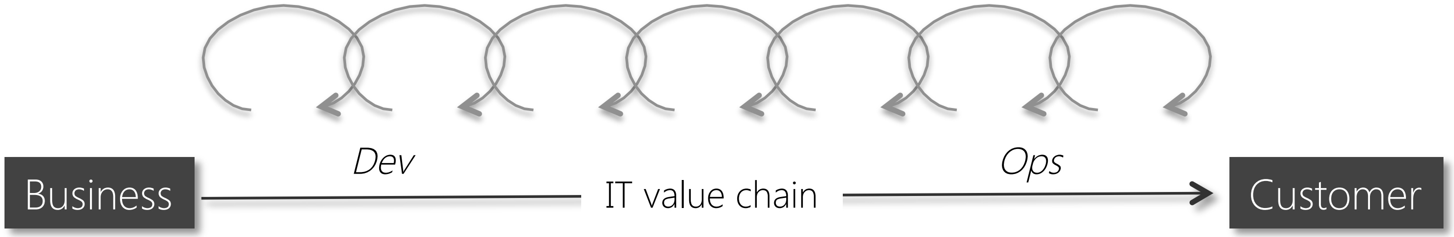
- Maximize flow (minimize cycle times)
- Optimize for global goals (holistic view)
- Never pass defects downstream
- Limit work in progress
- Build systems and organizations that are safe to change

Amplify feedback loops



- Facilitate constant flow of fast feedback from right-to-left
- Create quality at source (provide knowledge where needed)
- Create shared goals and shared pain between Dev and Ops
- Implement fast automated test suites
- Pervasively measure outcome (customer value), not output

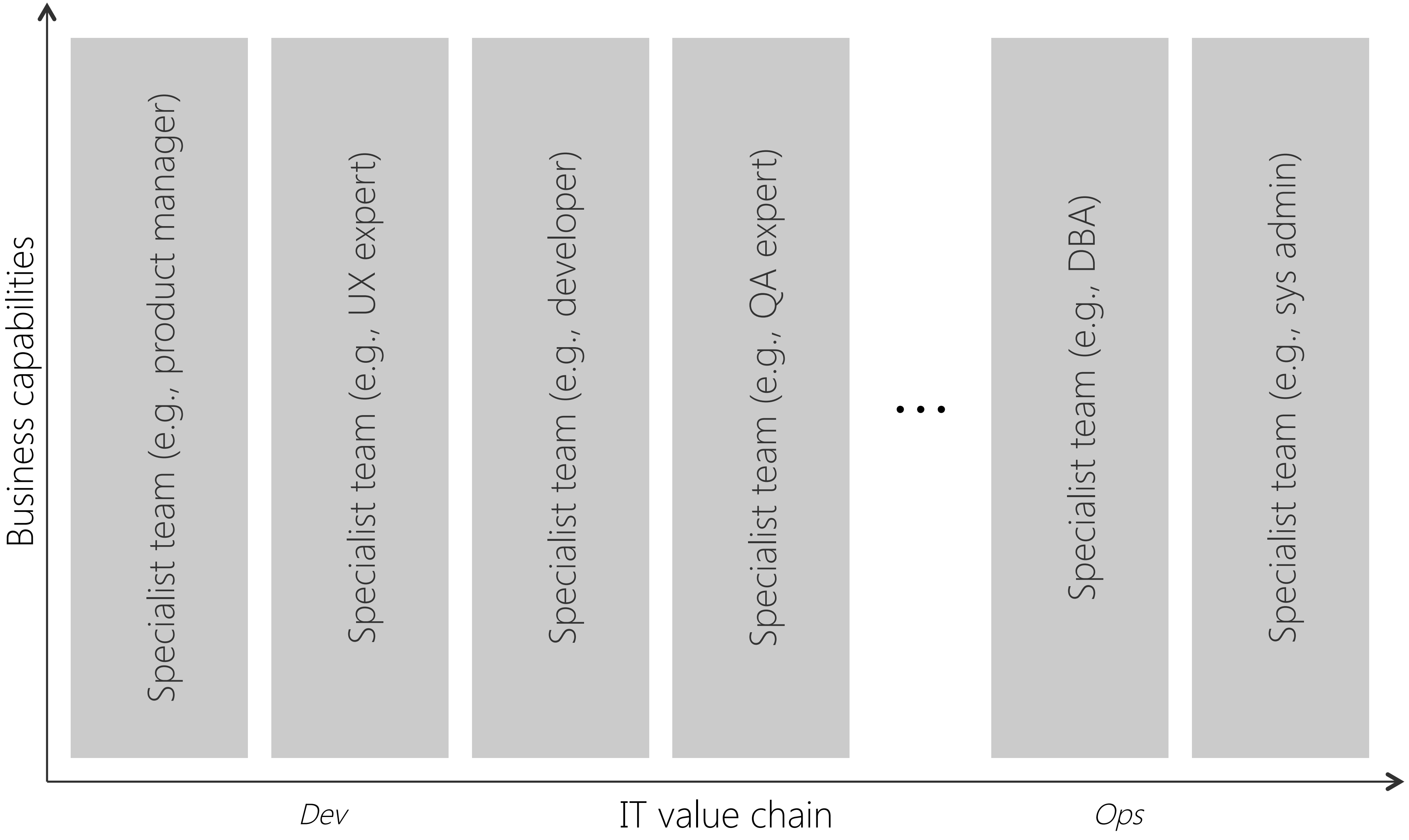
Continual experimentation and learning



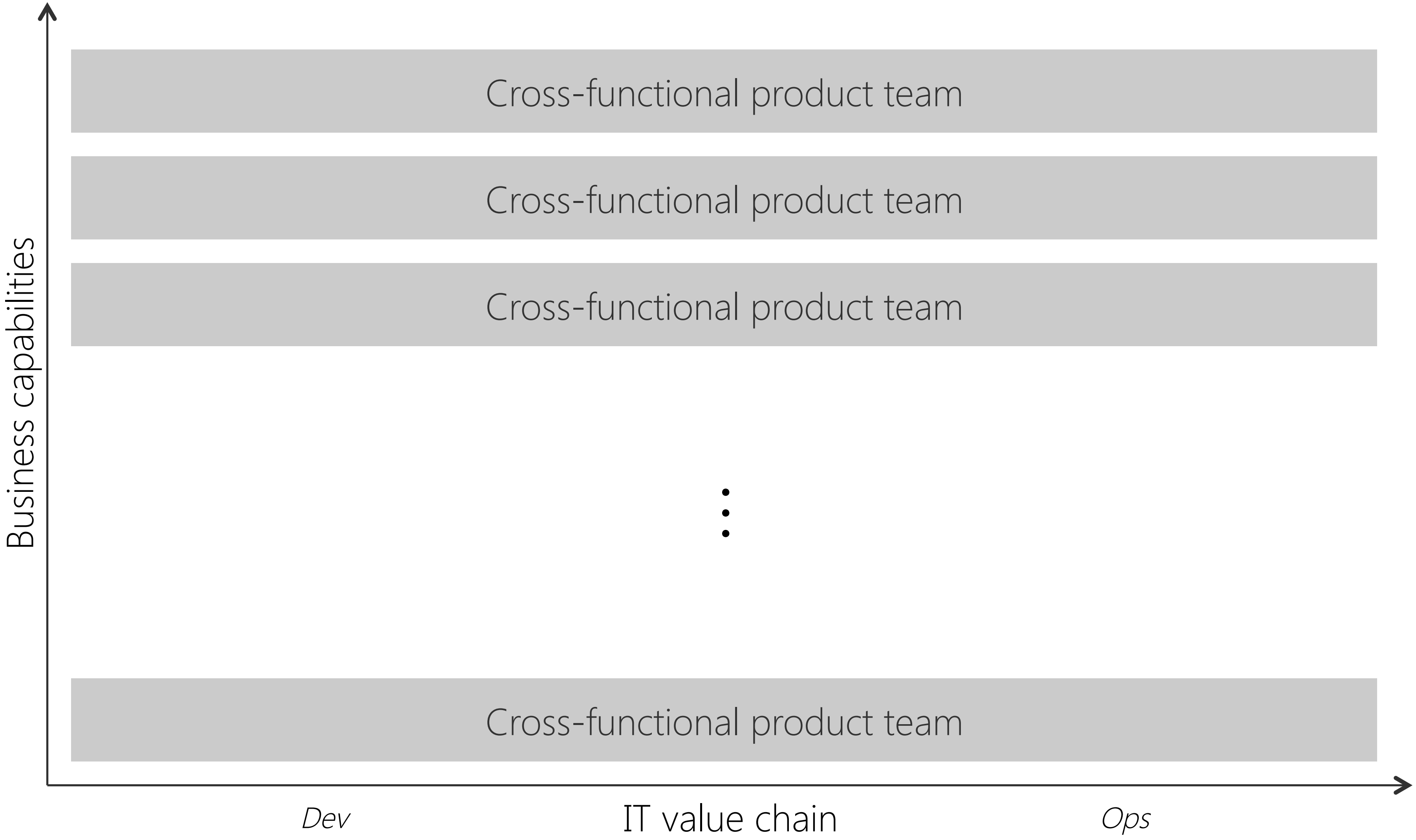
- Create a culture that fosters two things
  - Continual experimentation, taking risks and learning from success and failure
  - Understanding that repetition and practice is the prerequisite to mastery
- Allocate at least 20% of Dev and Ops cycles to NFRs
- Constantly reinforce that improvements are encouraged & celebrated

If taken seriously DevOps will eventually rotate your IT organization by 90°

Traditional IT organization


Business capabilities
Specialist team (e.g., product manager)
Specialist team (e.g., UX expert)
Specialist team (e.g., developer)
Specialist team (e.g., QA expert)
…
Specialist team (e.g., DBA)
Specialist team (e.g., sys admin)
Dev
IT value chain
Ops

DevOps IT organization



Cross-functional product team

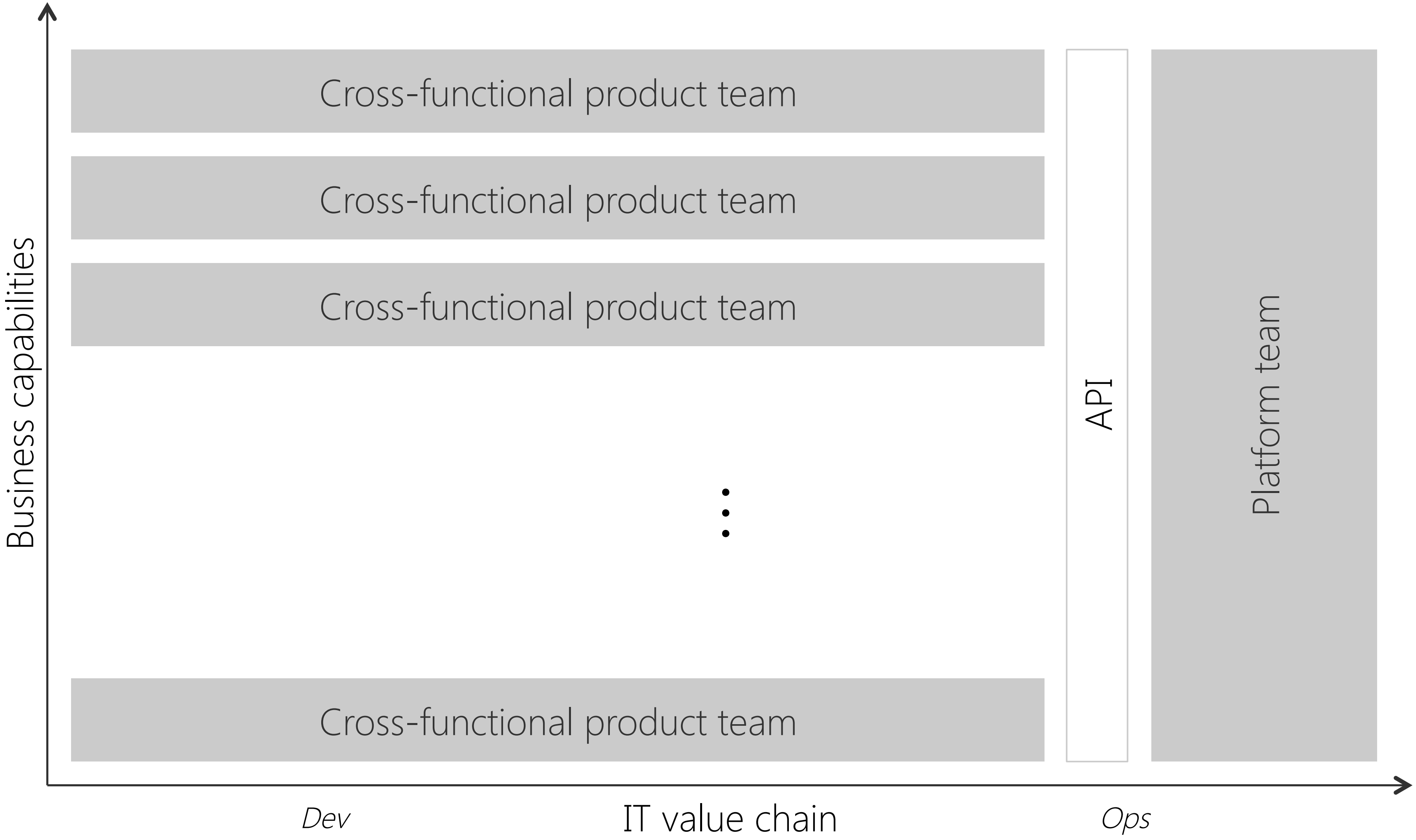
Cross-functional product team

Cross-functional product team

⋮

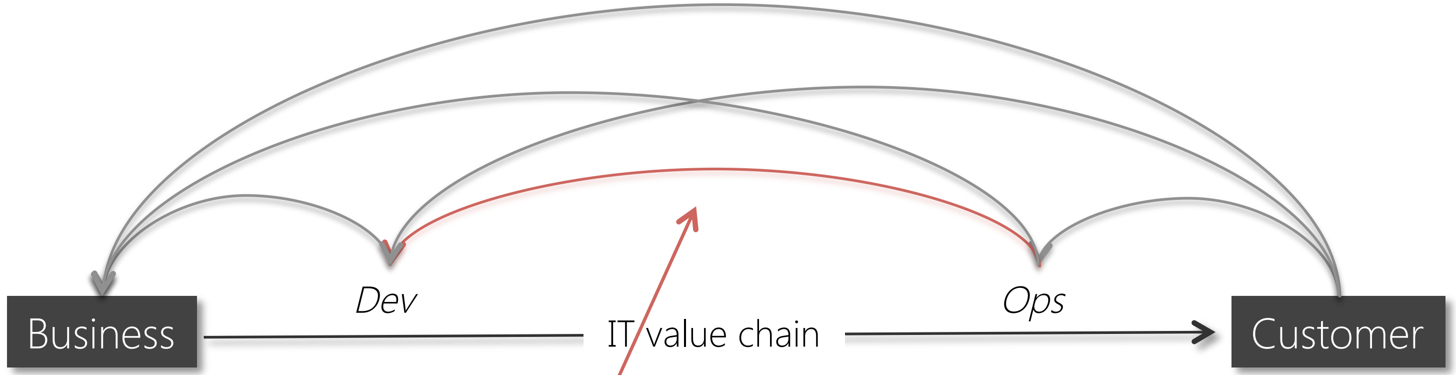
Cross-functional product team

DevOps IT organization (optimized)


Business capabilities
Cross-functional product team
Cross-functional product team
Cross-functional product team
⋮
Cross-functional product team
API
Platform team
Dev
IT value chain
Ops

But that’s still a long way to go for many organizations …

Amplify feedback loops



- Facilitate constant flow of fast feedback from right-to-left

Thus, we'll focus on this feedback loop

- Create quality at source (provide knowledge where needed)
- Create shared goals and shared pain between Dev and Ops
- Implement fast automated test suites
- Pervasively measure outcome (customer value), not output

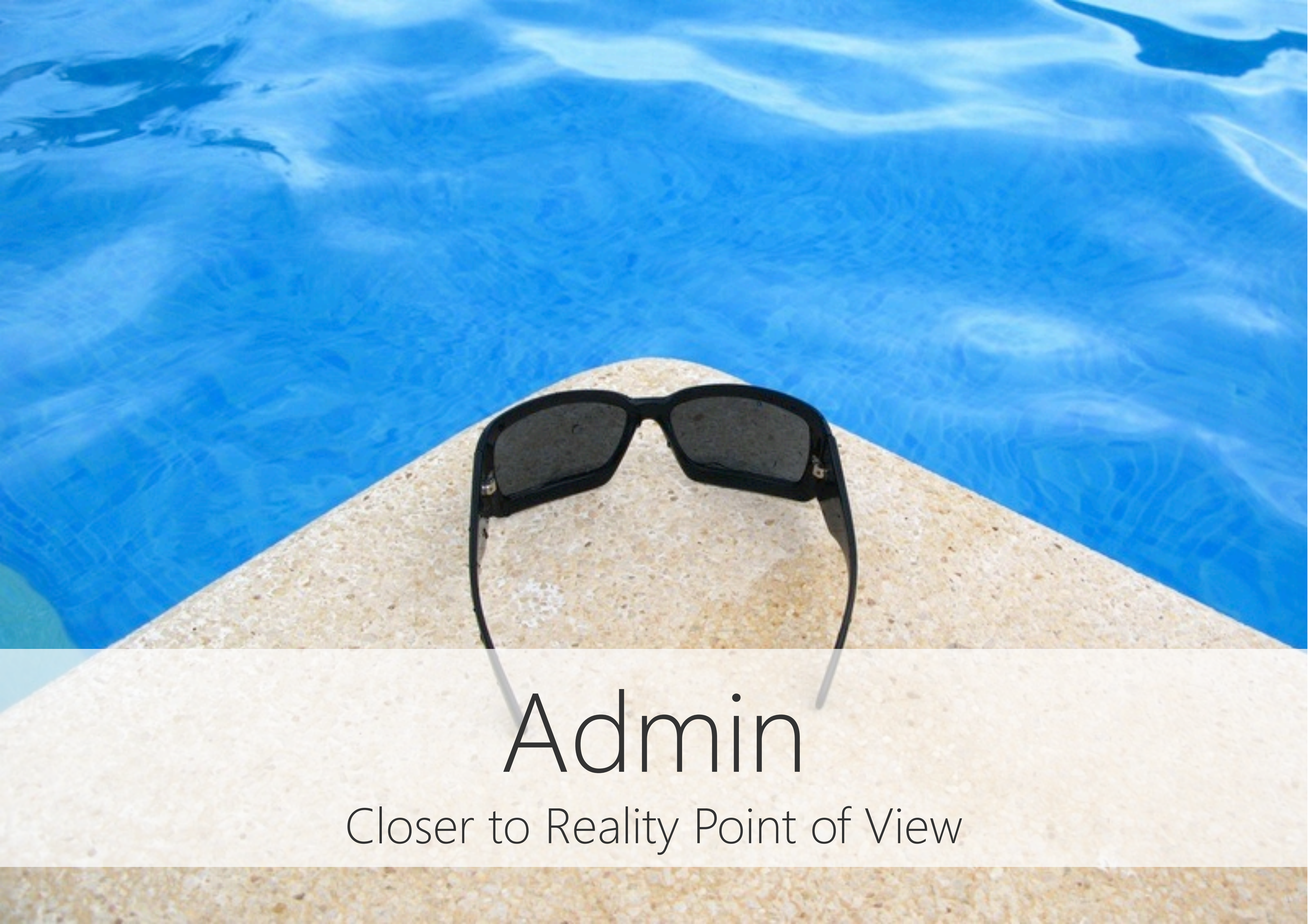
Let’s talk about operations …

Operations

Developers Point of View

Admin

Developers Point of View



Admin

Closer to Reality Point of View

Top 5 Needs of an Admin

1. Give me my peace and quiet!
2. Don‘t make me think!
3. Let me see that everything is fine!
4. Show me the problem – now!
5. Tell me what to do!

Top 5 Needs of an Admin (translated)

1. Give me my peace and quiet!

   (The application should just run smoothly)

2. Don‘t make me think!

   (Rollout, setup and operation of the application should be easy)

3. Let me see that everything is fine!

   (The application should show its state)

4. Show me the problem – now!

   (The application should provide concise error messages and enable easy root cause drilldown)

5. Tell me what to do!

   (The application should be documented properly – including error handling instructions)

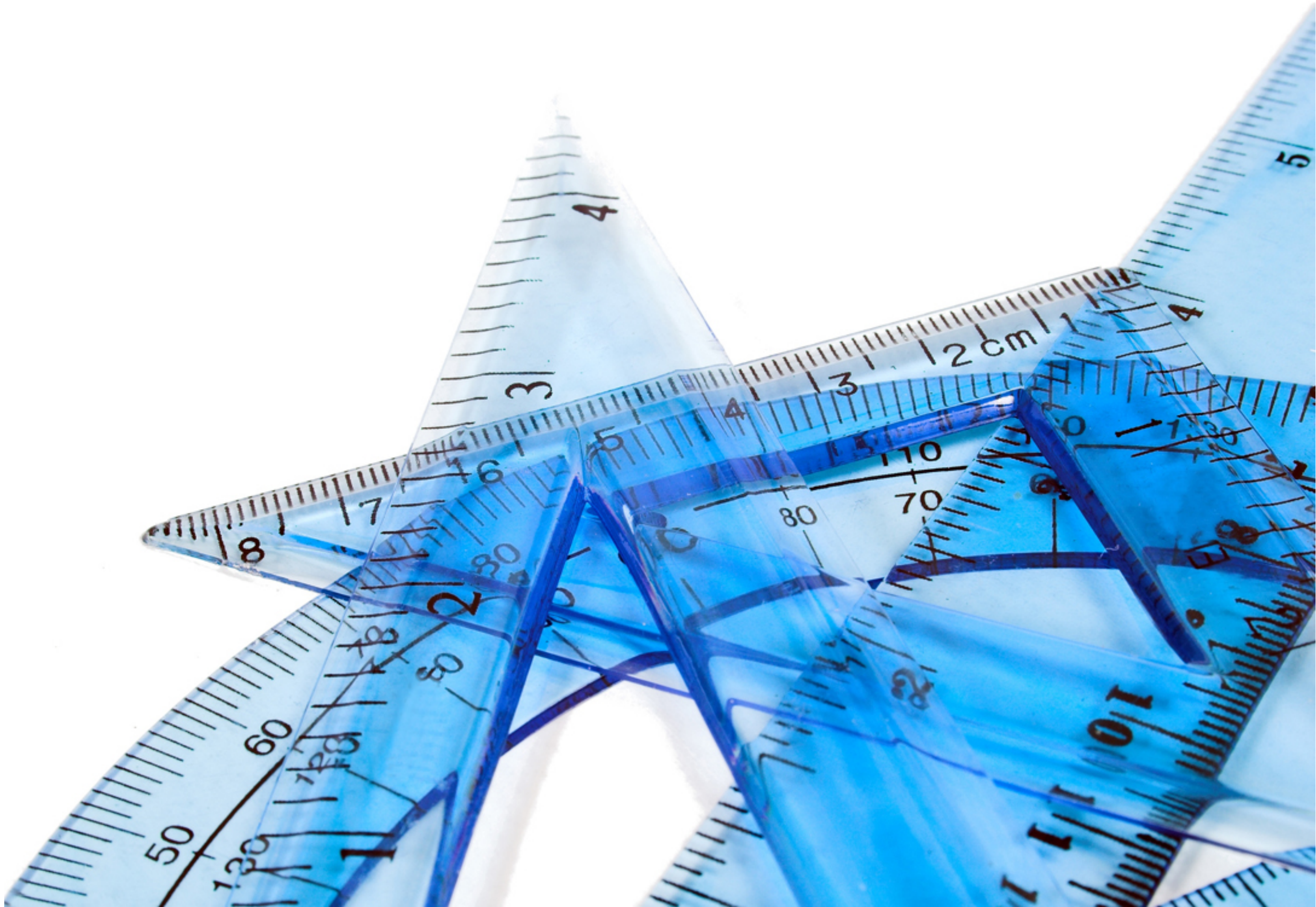
Top 3 Dev Challenges

1. Manageability
2. Resilience
3. Transparency
4. Documentation

11 Design Principles

For production-ready Applications

- Manageability (4)
- Resilience (5)
- Transparency (2)



Manageability

Deployment

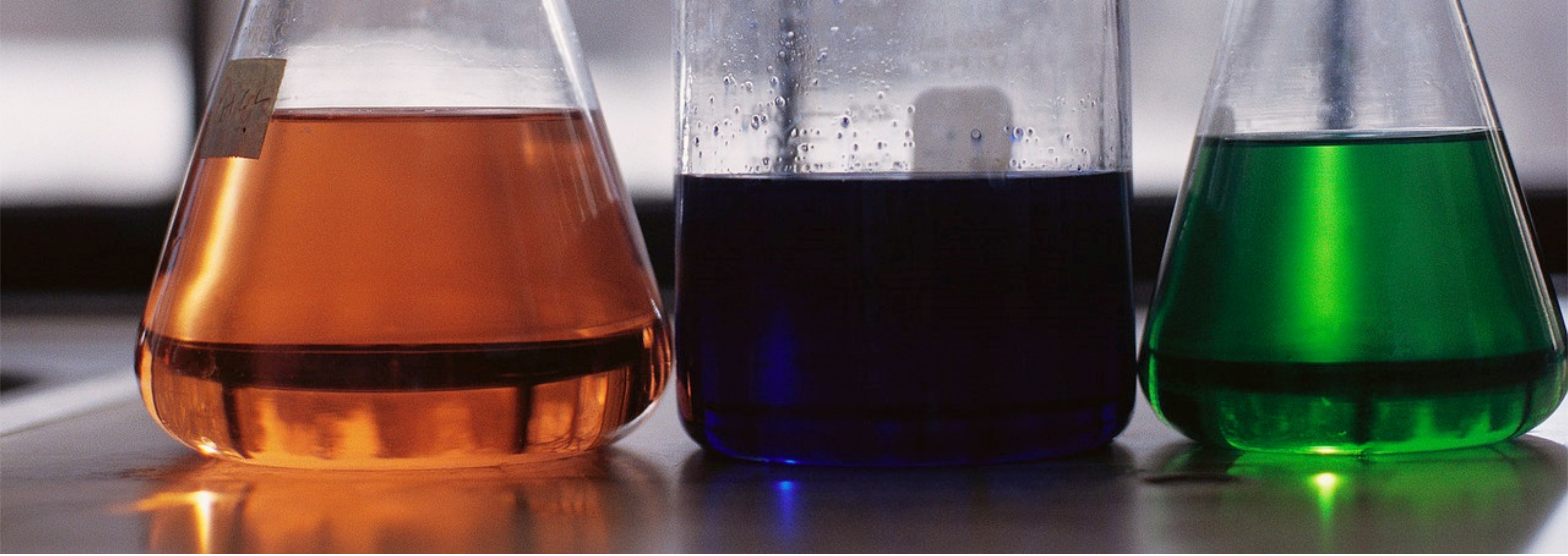
(Manageability)

- One-click deployment
- Preserve settings
- Provide rollbacks or roll-forward
- Go for containers

Configuration

(Manageability)

- Avoid multiple configuration procedures
- Define default value handling
- Organize change traceability
- Notification about new parameters



Configuration Parameter Types

(Manageability)

- Context-related parameters

  Do not stage – managed by stage admin

- Application-related parameters

  Must be staged – managed by application admin

- Business-related parameters

  Must be staged – managed by business admin

Backup

(Manageability)

- Think about backup purpose
- Define backup strategy
- Provide tooling
- What about cloud backup?

Resilience

Isolation

- System must not fail as a whole
- Divide system in failure units (a.k.a. bulkheads)
- Avoid error propagation by isolating failure units
- Define fallback strategy

Redundancy

- Elaborate use case

  Minimize MTTR / avoid latency / handle response errors / …

- Define routing & distribution strategy

  Round robin / master-slave / fan-out & quickest one wins / …

- Consider admin involvement

  Automatic vs. manual / notification – monitoring / …

Loose Coupling

- Isolate failure units (complements bulkheads)
- Go asynchronous wherever possible
- Use timeouts & circuit breakers
- Make actions idempotent

Implementation Example #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 (...) {
    ...
}
```

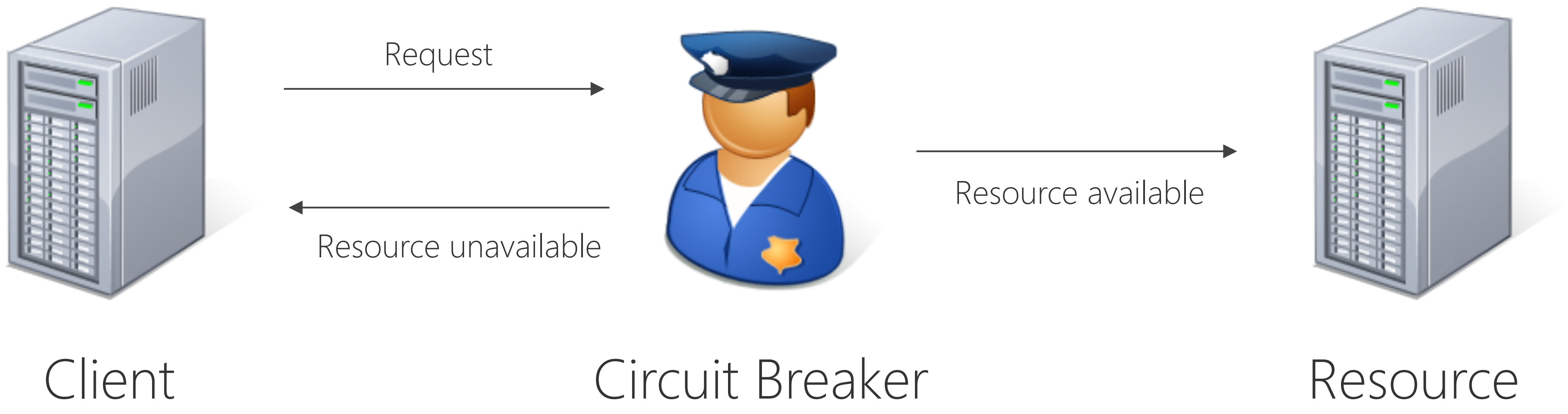
Timeouts (3)

```
// 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 (...) {
    ...
}
```

Implementation Example #2

Circuit Breaker

Circuit Breaker – concept



Request

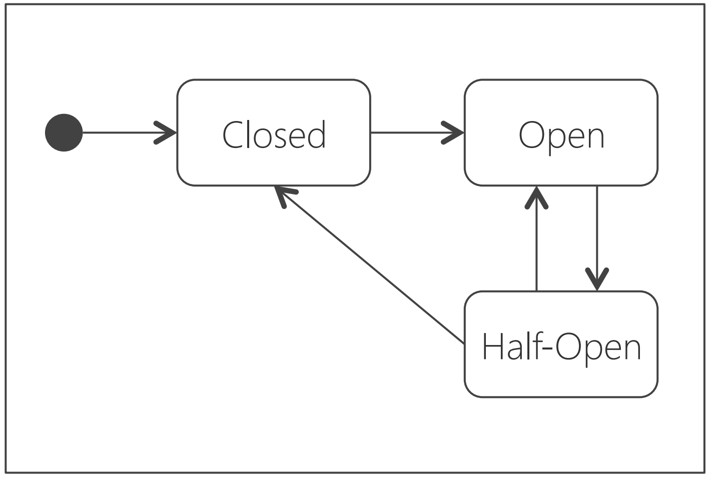
Resource unavailable

Resource available

Client

Circuit Breaker

Resource

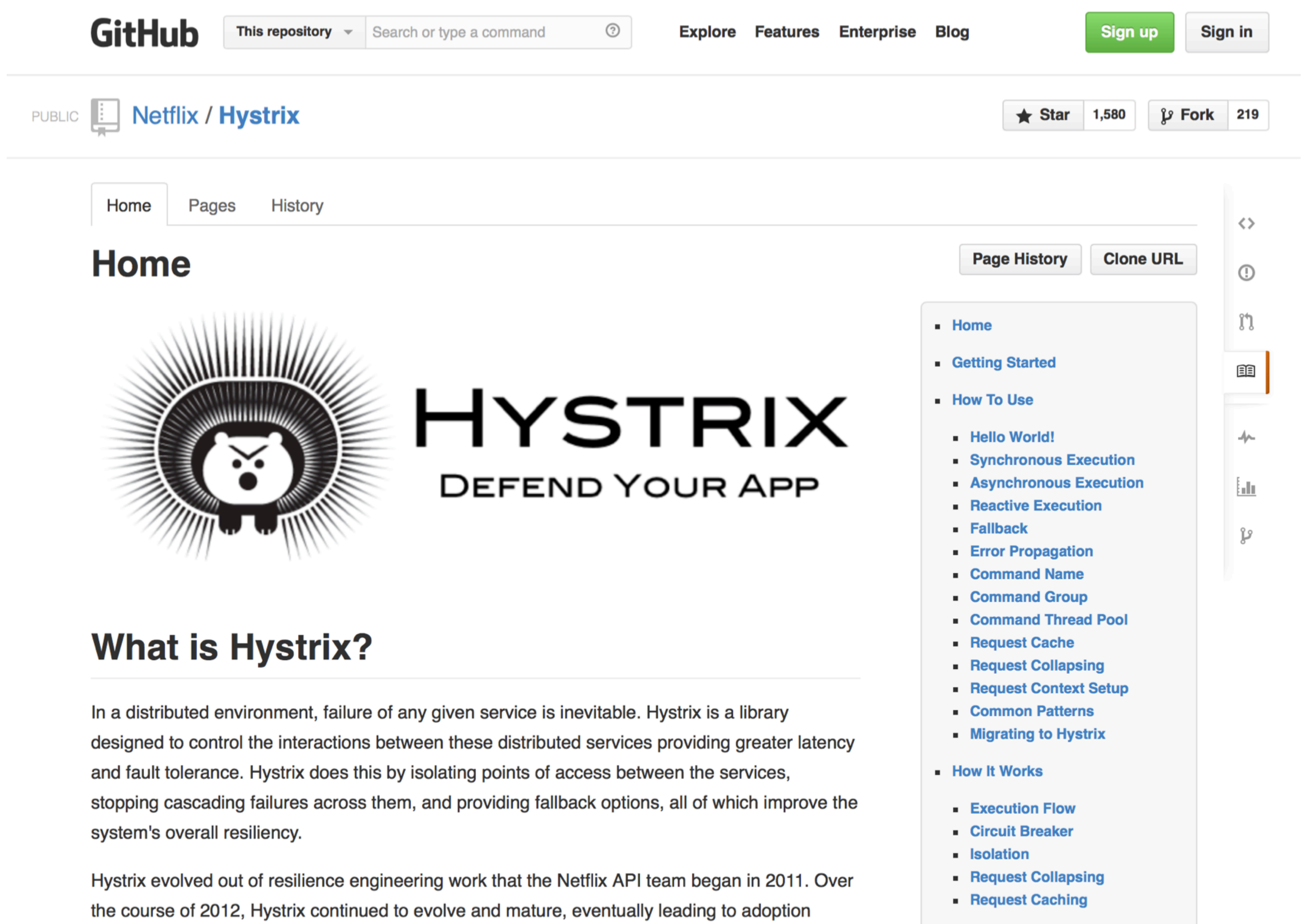


Closed

Open

Half-Open

Lifecycle



Home

- Home
- Getting Started
- How To Use
  - Hello World!
  - Synchronous Execution
  - Asynchronous Execution
  - Reactive Execution
  - Fallback
  - Error Propagation
  - Command Name
  - Command Group
  - Command Thread Pool
  - Request Cache
  - Request Collapsing
  - Request Context Setup
  - Common Patterns
  - Migrating to Hystrix
- How It Works
  - Execution Flow
  - Circuit Breaker
  - Isolation
  - Request Collapsing
  - Request Caching

What is Hystrix?

In a distributed environment, failure of any given service is inevitable. Hystrix is a library designed to control the interactions between these distributed services providing greater latency and fault tolerance. Hystrix does this by isolating points of access between the services, stopping cascading failures across them, and providing fallback options, all of which improve the system's overall resiliency.

Hystrix evolved out of resilience engineering work that the Netflix API team began in 2011. Over the course of 2012, Hystrix continued to evolve and mature, eventually leading to adoption

Implemented patterns

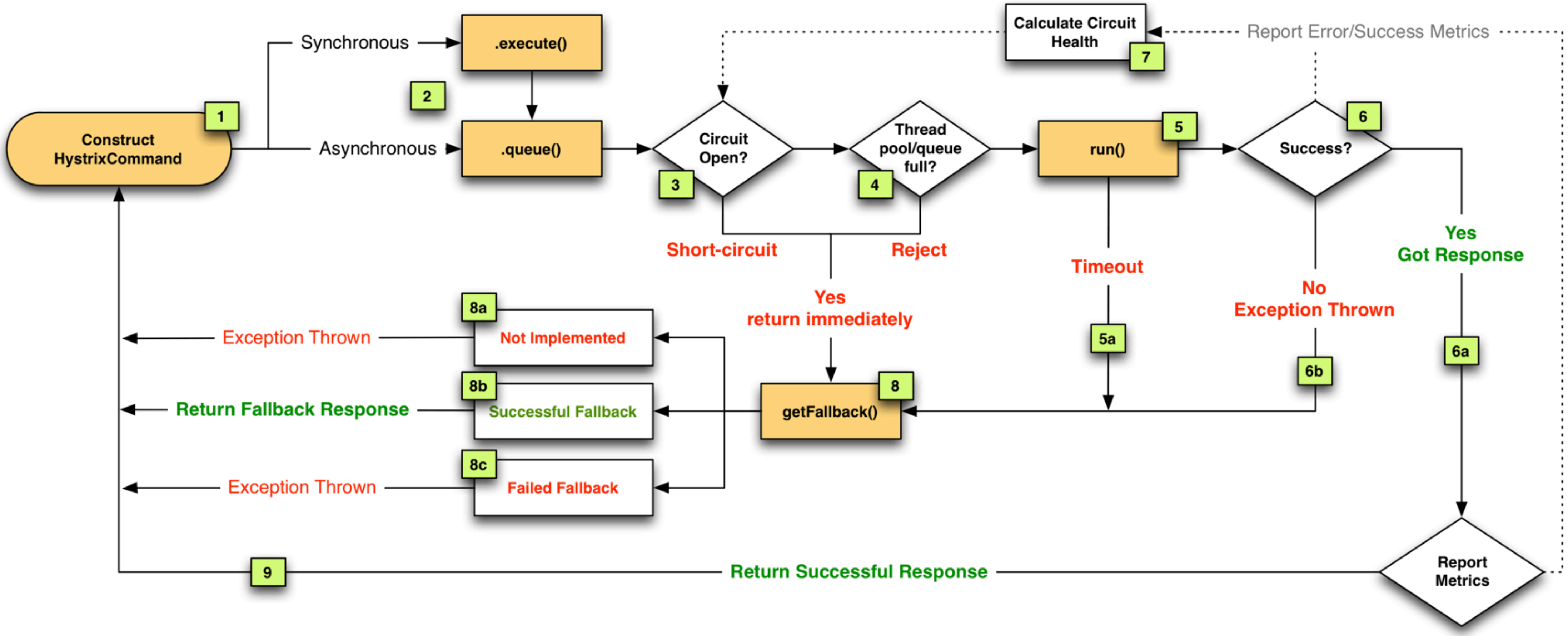
- Timeout
- Circuit breaker
- Load shedder
- Fallback

Supported patterns

- Bulkheads
  (a.k.a. Failure Units)
- Fail fast
- Fail silently
- Graceful degradation of service
- Failover
- Escalation
- Retry
- …

Hello, world!

```
// Hystrix “Hello world”

public class HelloCommand extends HystrixCommand<String> {
    private static final String COMMAND_GROUP = ”Hello”; // Not important here
    private final String name;

    // Request parameters are passed in as constructor parameters
    public HelloCommand(String name) {
        super(HystrixCommandGroupKey.Factory.asKey(COMMAND_GROUP));
        this.name = name;
    }

    @Override
    protected String run() throws Exception {
        // Usually here would be the resource call that needs to be guarded
        return "Hello, " + name;
    }
}


// Usage of a Hystrix command – synchronous variant
@Test
public void shouldGreetWorld() {
    String result = new HelloCommand("World").execute();
    assertEquals("Hello, World", result);
}
```



Source: https://github.com/Netflix/Hystrix/wiki/How-it-Works

Alarm
Betätigung führt im
Bahnhofsbereich zum Halt
Missbrauch strafbar
Notruf
wenn Alarm
betätigt

Fallbacks

- *What will you do if a request fails?*
- Consider failure handling from the very beginning
- Supplement with general failure handling strategies

Scalability

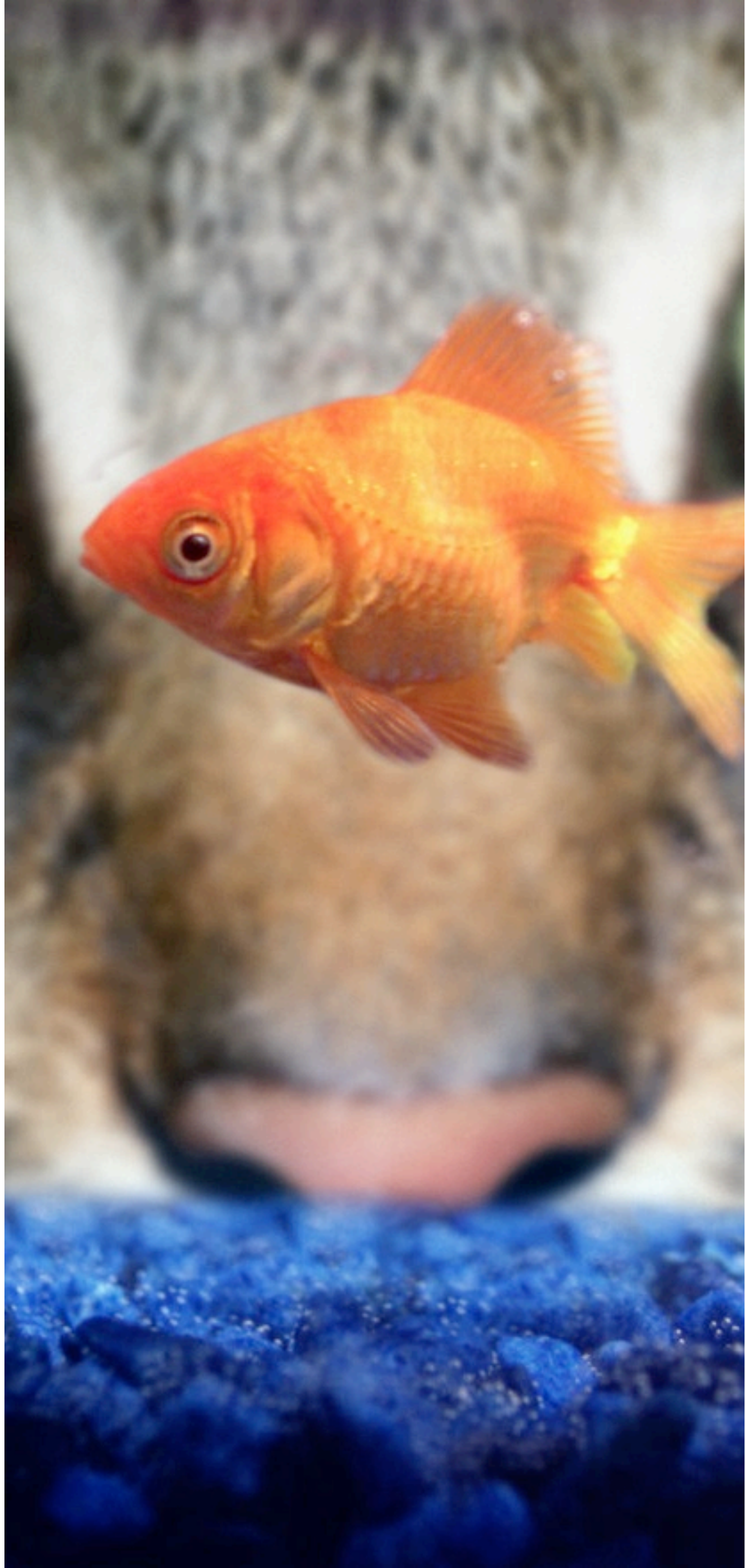
- Define scaling strategy
- Think full stack
- Design for elasticity
- At least apply D-I-D rule

Transparency

Monitoring

(Transparency)

- Think about required metrics
- Design hook or event mechanism
- Plan for changing metrics
- Consider event sourcing



Logging

(Transparency)

- Consider log message structure

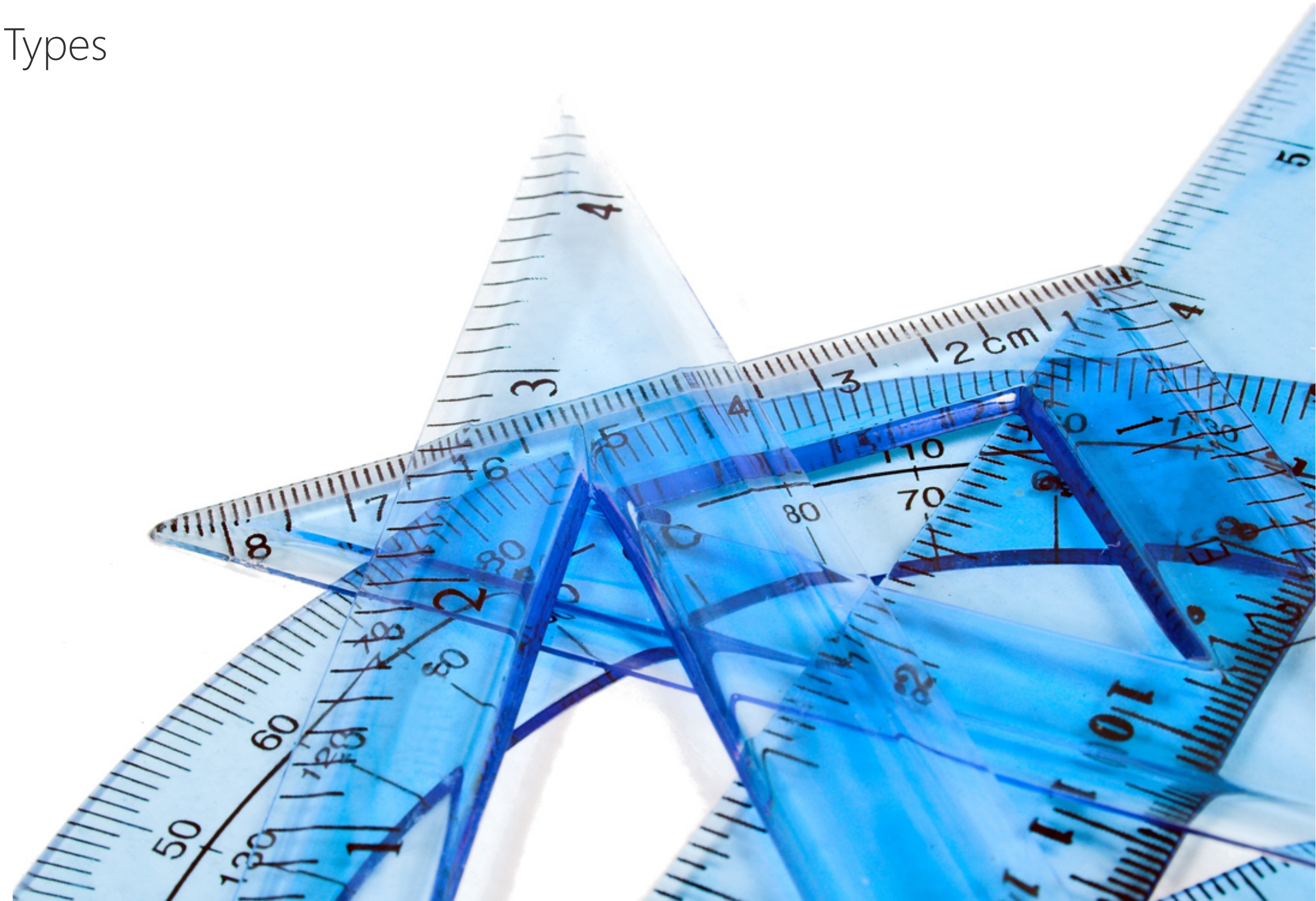
  Assume centralized logging: required information / machine readable / human readable

- Define logging policy

  Debug and less: developers perspective / Info and more: operations perspective

11 Design Principles

- Manageability
  - Deployment
  - Configuration
  - Configuration Parameter Types
  - Backup
- Resilience
  - Isolation
  - Redundancy
  - Loose Coupling
  - Fallbacks
  - Scalability
- Transparency
  - Monitoring
  - Logging



Don’t forget to read the “bible” of production-ready software …

https://pragprog.com/book/mnee/release-it

The Pragmatic Programmers

Release It!

Design and Deploy Production-Ready Software

Michael T. Nygard

Wrap-up

- The importance of “production readiness”
- The 3 ways of DevOps
- The needs of Ops
- The resulting challenges for Dev
- Design principles to support the needs
    - Manageability
    - Resilience
    - Transparency

It’s all about production!

@ufried

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