Five (easy?) Steps Towards Continuous Delivery

Eberhard Wolff
@ewolff
innoQ

http://continuous-delivery-buch.de/

Eberhard Wolff

Microservices

Grundlagen flexibler Softwarearchitekturen

dpunkt.verlag

http://microservices-buch.de/

Microservices

Flexible Software Architectures

Eberhard Wolff

http://microservices-book.com/

Eberhard Wolff

Microservices Primer

A Short Overview

FREE!!!!

innoQ

http://microservices-book.com/primer.html

What is CD?

Continuous Delivery = Deployment automation

Continuous Delivery = Deployment automation to increase speed

Production problem.
Boss screams at you.
How long to production?

‹15 min

‹30 min

‹60 min

›60 min

›1 day

OH “The application server takes 15 min to start.”

Why don’t we deliver software every day into production?

Hot fix

Old release

New release

Old release

Risk of bug fix small compared to keeping faulty release in production.

Would we deliver into production more often if it took e.g. 5 minutes?

Would we deliver software into production more often with Puppet, Chef, Docker?

Step One

Realize deployment automation is just a prerequisite for Continuous Delivery!

Continuous Delivery = Deployment automation to increase speed

Continuous Delivery =

Deployment automation

to increase speed

WRONG

What is CD?

Agile Manifesto Principles

Our highest priority is

to satisfy the customer

through early and

continuous delivery

of valuable software.

Build Pipeline

Many tests to minimize risk

Fast Feedback

Infrastructure automation

Why Continuous Delivery?

- Faster Deployment
- Higher Reliability
- Reproducibility

Why Continuous Delivery?

- Cost-savings hard to estimate
- CD is an important step towards better and faster results

Understand the Goal!

- Seemingly simple
- ...but often forgotten
- Not just about time-to-market
- Might also be reliability!

Reliability

- Errors hard to reproduce
- Software hard to install
- Invest in deployment automation
- Ensure environment is identical in development and production

Time-to-Market

- Optimize processing time!
- There is already a pipeline
- Goal: Constant flow of features through the pipeline
- Optimize throughput

Value Stream Mapping

- Analyze existing pipeline
- Optimize throughput

Delay

5 Days | 2 Days | 1 Day

Commit | Acceptance Testing | Capacity Testing | Release

1 day | 1 day | 1 day

Lead Time

Cycle Time = Delay + Lead Time

20 minutes | 20 minutes | 2 Day

Automated Acceptance Testing | Automated Capacity Testing | Testing, Sign Off & Release

1 hour | 30 minutes | 2 day

Delay

5 Days | 2 Days | 1 Day

Commit | Acceptance Testing | Capacity Testing | Release

11 days

1 day | 1 day | 1 day

Lead Time

Cycle Time = Delay + Lead Time

20 minutes | 20 minutes | 2 Day

Automated Acceptance Testing | Automated Capacity Testing | Testing, Sign Off & Release

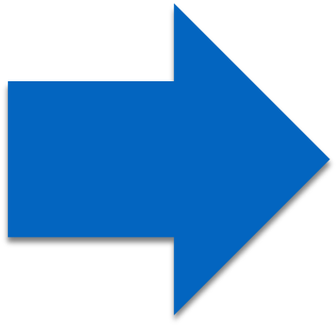
5 days

1 hour | 30 minutes | 2 day

Result

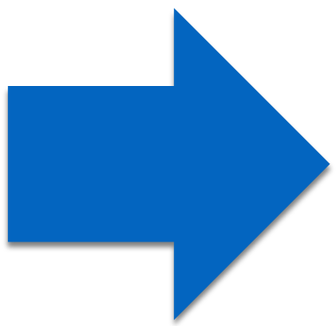
- Cycle time reduced: Automated tests faster
- ...and less effort
- Still manual sign-off & Release
- Feedback faster: Early 80% acceptance test

Goal:
Reliability



Deployment
Automation

Goal:
Time-to-Market
Fast Feedback

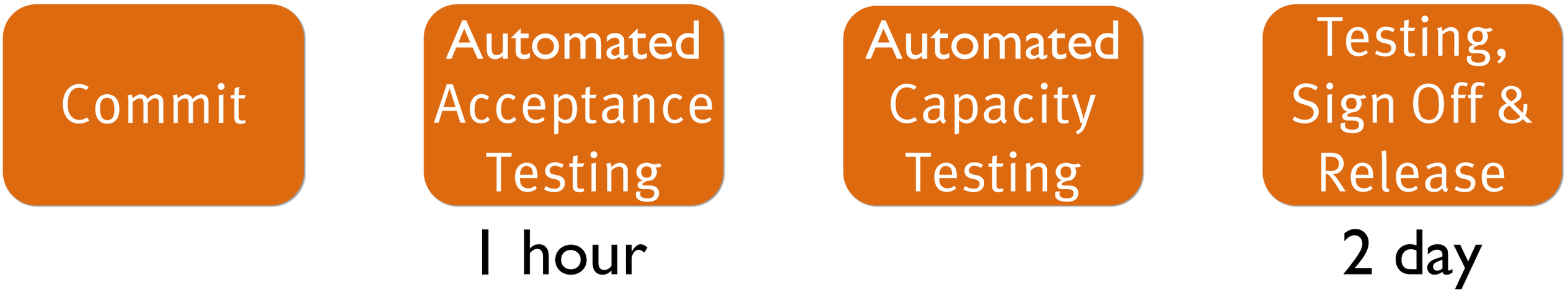


Value Stream
Mapping

Step Two

Understand goals!

Take pragmatic steps!



Acceptance test = software is accepted

Why Sign Off?

Why Sign-Off?

- Customer wants to check the final result.
- Understandable
- But: Slow
- But: Hard to reproduce

Eliminate manual Sign-off!

Sign-off->Automated Tests

- Automated tests are fast
- ...and easily reproducible
- ...and cheaper
- Obviously the better choice!

Why Sign-off?

- Risk of deploying the wrong software
- Lack of trust in tests
- Risk handling strategy

Handling risk

- Make it easier to resolve issues
- Make deployment easier and faster
- Problem in production easier to fix

- Deployment automation


Old release

New release

New release

New release

New release

Old release

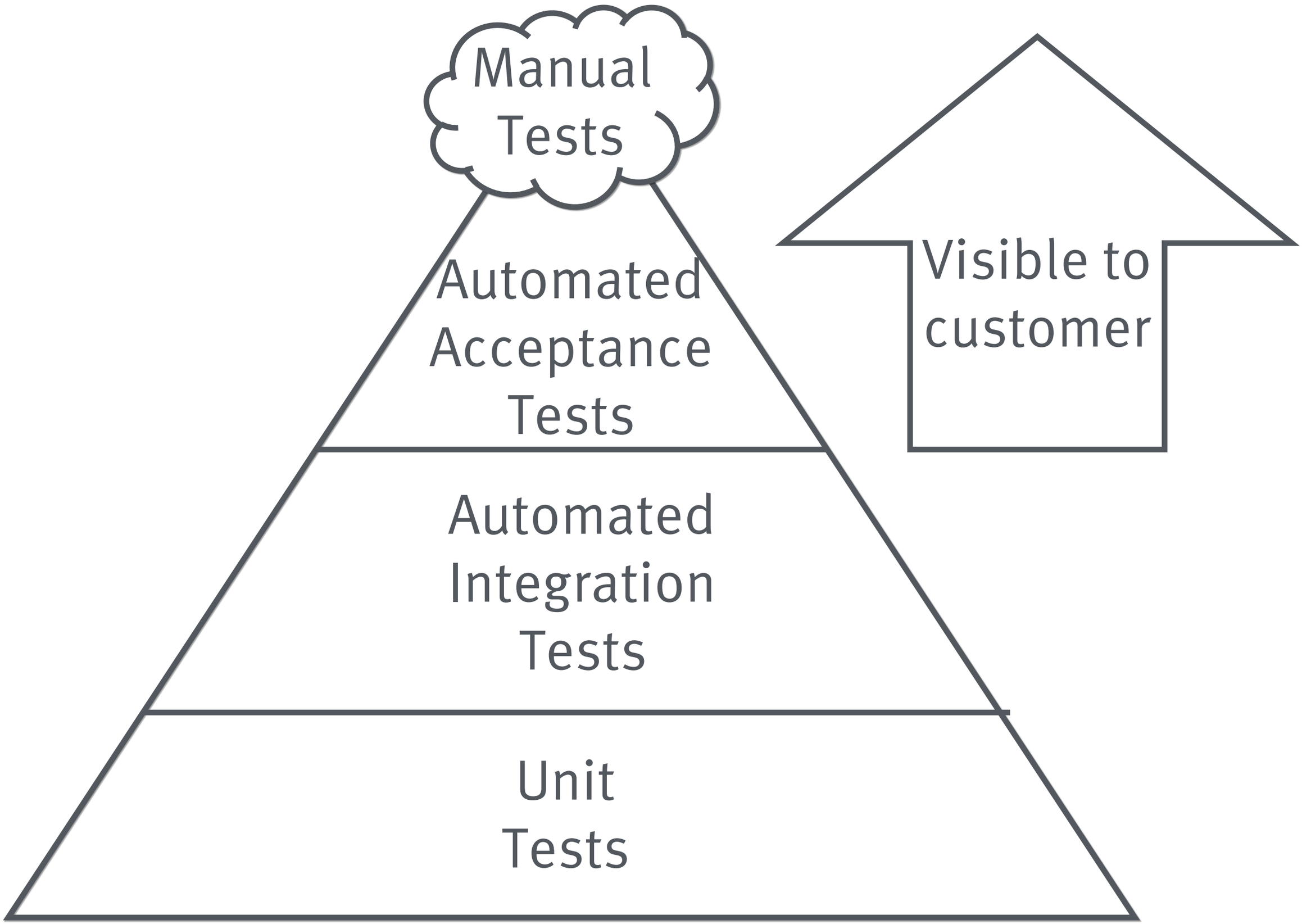
Handling risk

- Smaller deployments
- More frequent deployments
- Less risk that an error sneaks in
- Easier to reverse

Creating Trust

- Customer must understand the tests
- Reviews
- Use proper kind of tests

Test Pyramid



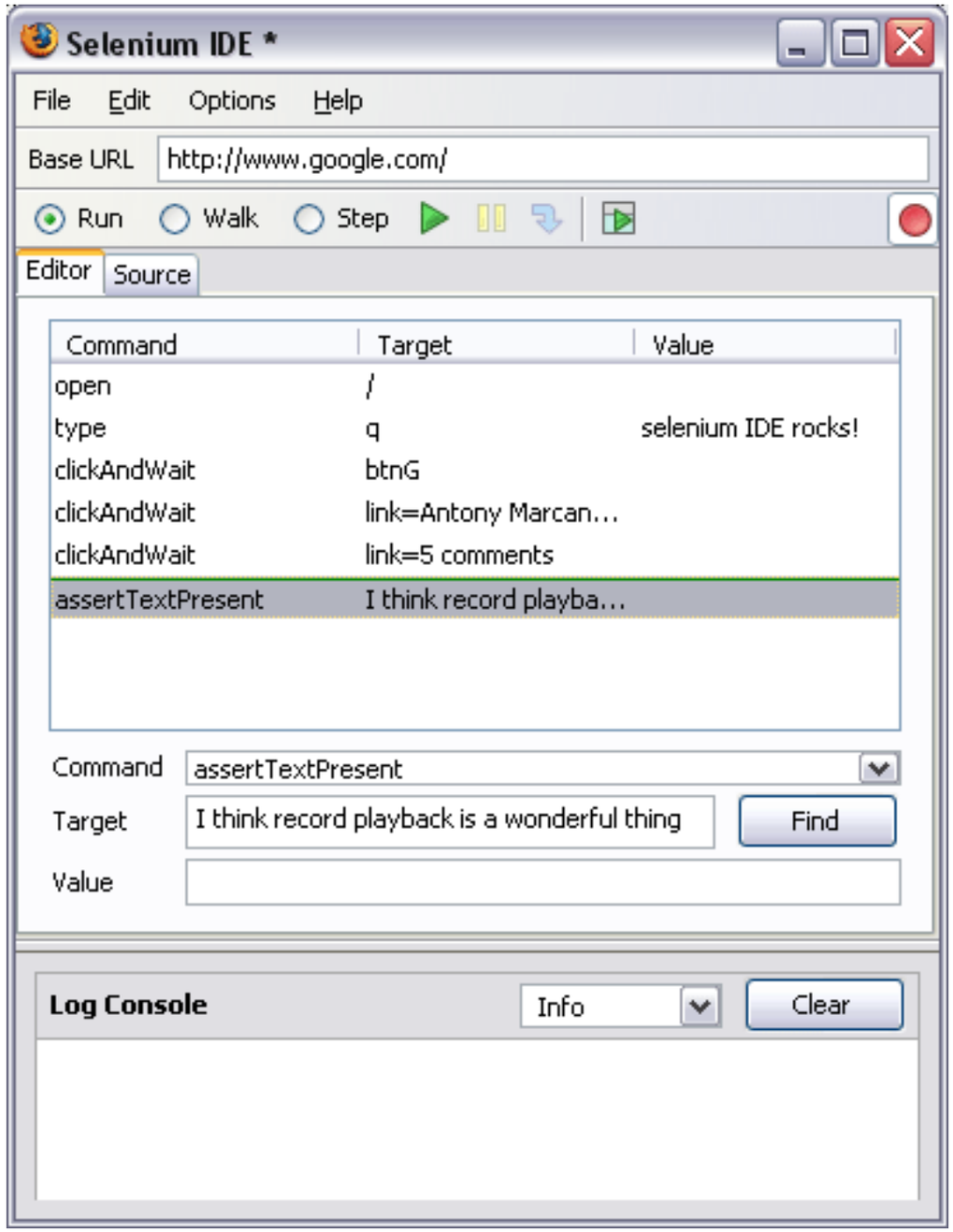
The ultimate requirement is an automated acceptance test.

XP 1999

UI Tests: Selenium

- Easy to start
- Natural for testers

- Fragile
- Loose semantics



Behavior-driven development: Example

Scenario: User registers successfully

Given a new user with email eberhard.wolff@gmail.com
firstname Eberhard
name Wolff

Context

When the user registers

Event

Then a customer with email eberhard.wolff@gmail.com should exist

And no error should be reported

Expected outcome

Creating Trust

- UI Tests are overused and have drawbacks
- BDD is designed for customers

- But most important:
- Choose whatever the customer understands!
- UI tests might be OK

Step Three

Eliminate manual sign-offs!
Create trust in tests!

CD & DevOps

- Usually Dev wants Continuous Delivery (CD)
- Dev wants easier and faster releases
- Ops not supportive
- However, they should aim for less risky deployments...

Ops

- Optimized for costs
- Caught in a local optimum
- i.e. standardized environments
- ...but manual deployment
- Large investment for full automation
- Continuous Delivery problem to be expected

Dev

Commit

Automated Acceptance Testing

Automated Capacity Testing

Testing, Sign Off & Release

Ops

CD & DevOps

- No need to create DevOps teams
- Collaboration needed, though
- Deployment is a joined Ops / Dev effort
- Good news: No reorganzation

CD & DevOps

- Seek feedback from Ops early on
- Try to leverage Ops experience

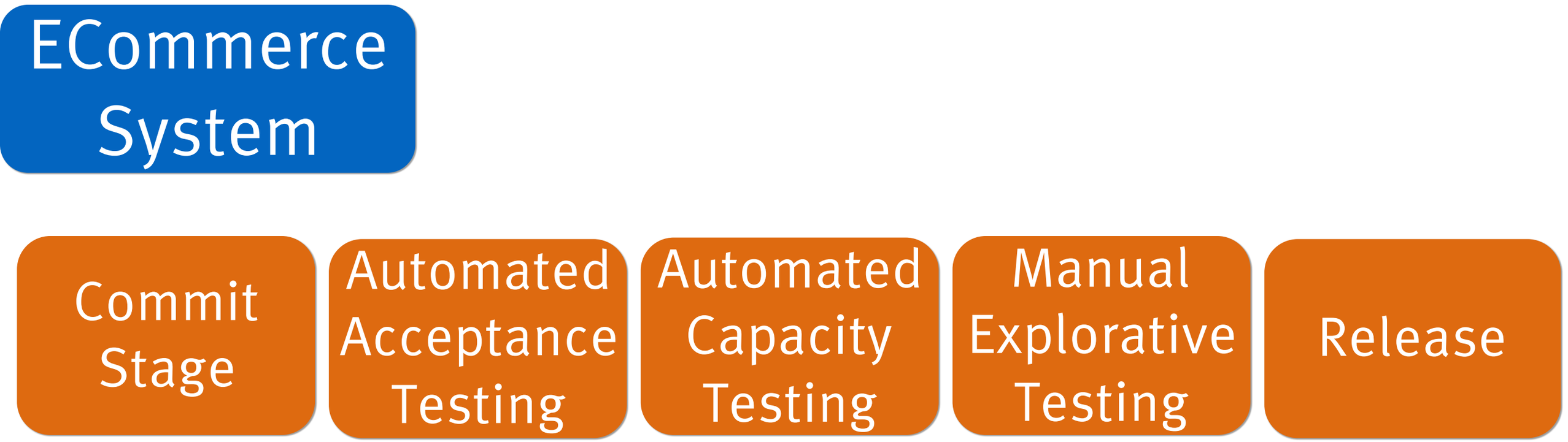
DevOps is no organization.

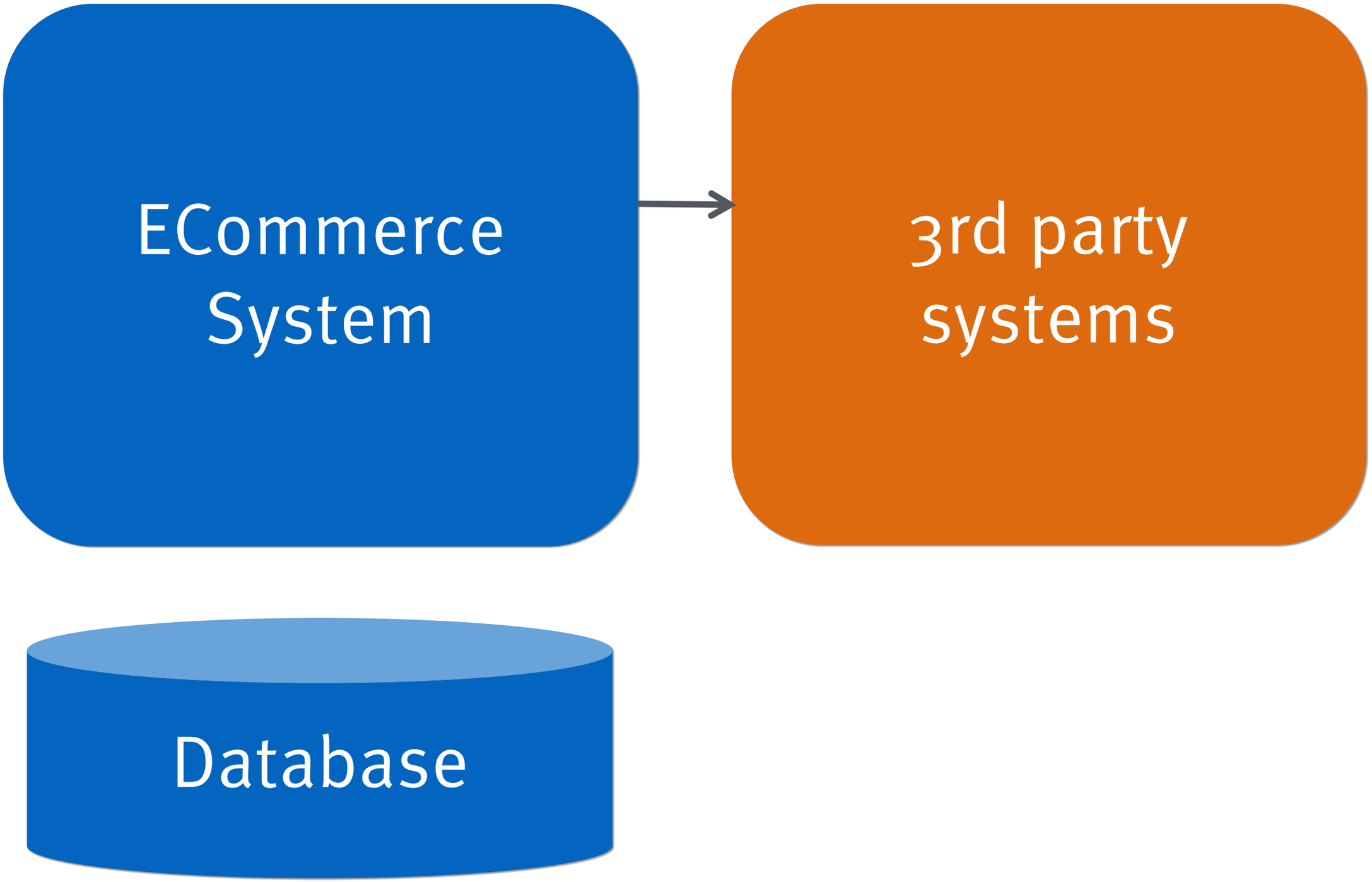
DevOps is collaboration.

Step Four

Deal with the gap between Dev and Ops!

Continuous Delivery: Build Pipeline




ECommerce System
3rd party systems
Database

Challenges

- Dependencies on 3rd party systems
- Must provide test systems
- ...or mocks
- Large database
- Must provide test data

Challenges

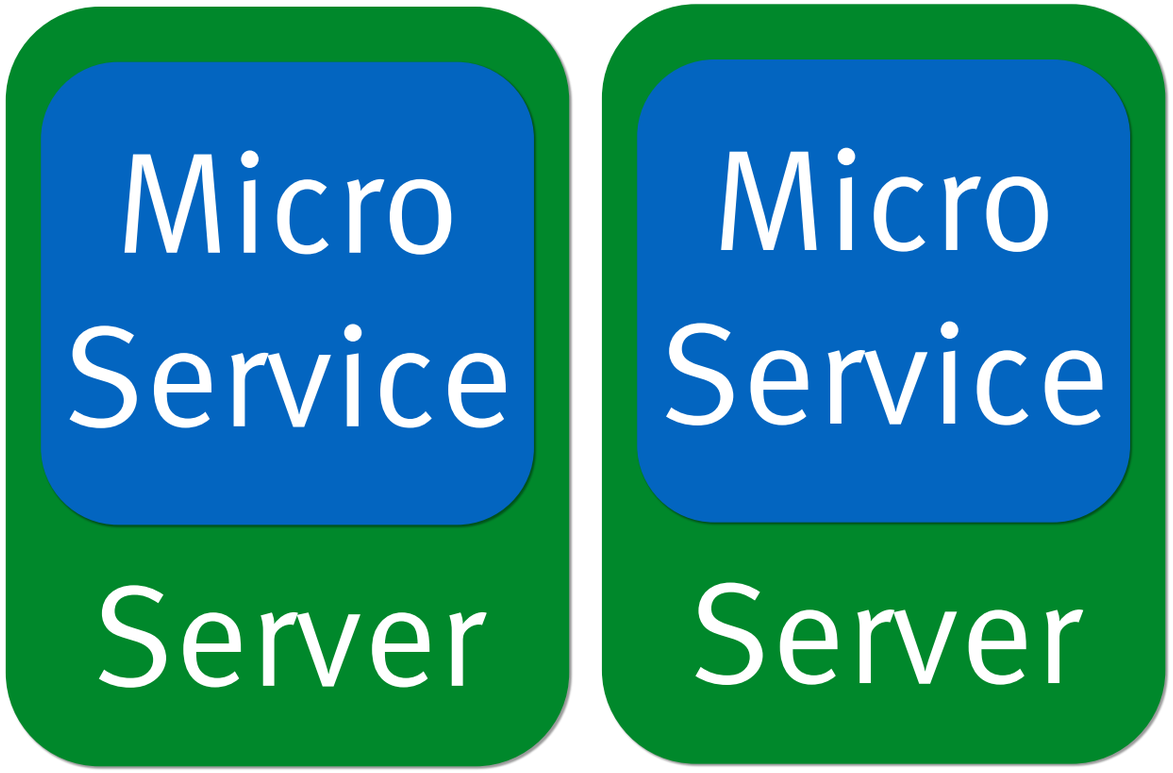
- Tests take too long
- Deployment takes too long
- Continuous Delivery pipeline takes far too long

Continuous Delivery with large deployment units is hard.

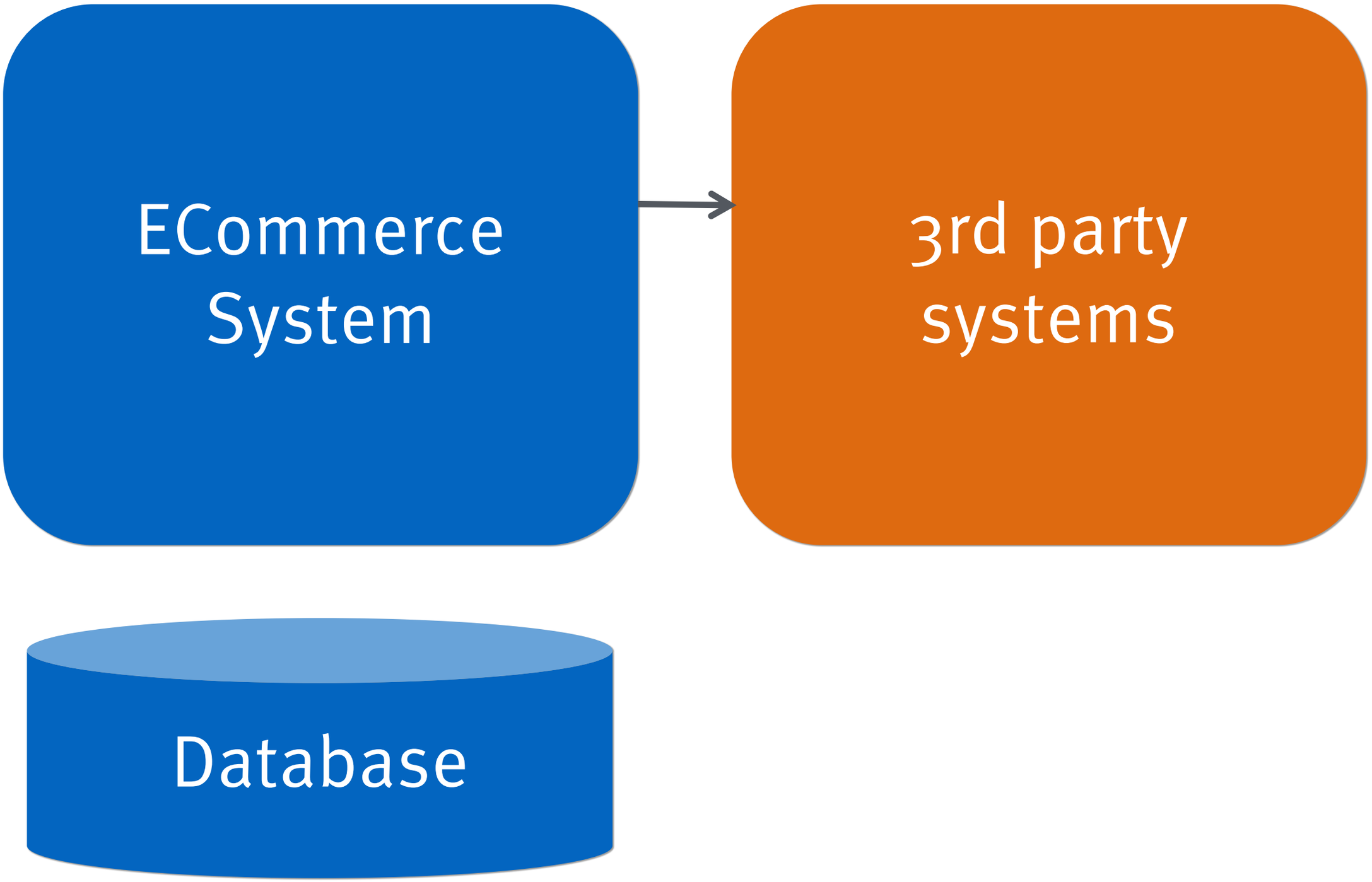
Like real hard.

Microservices

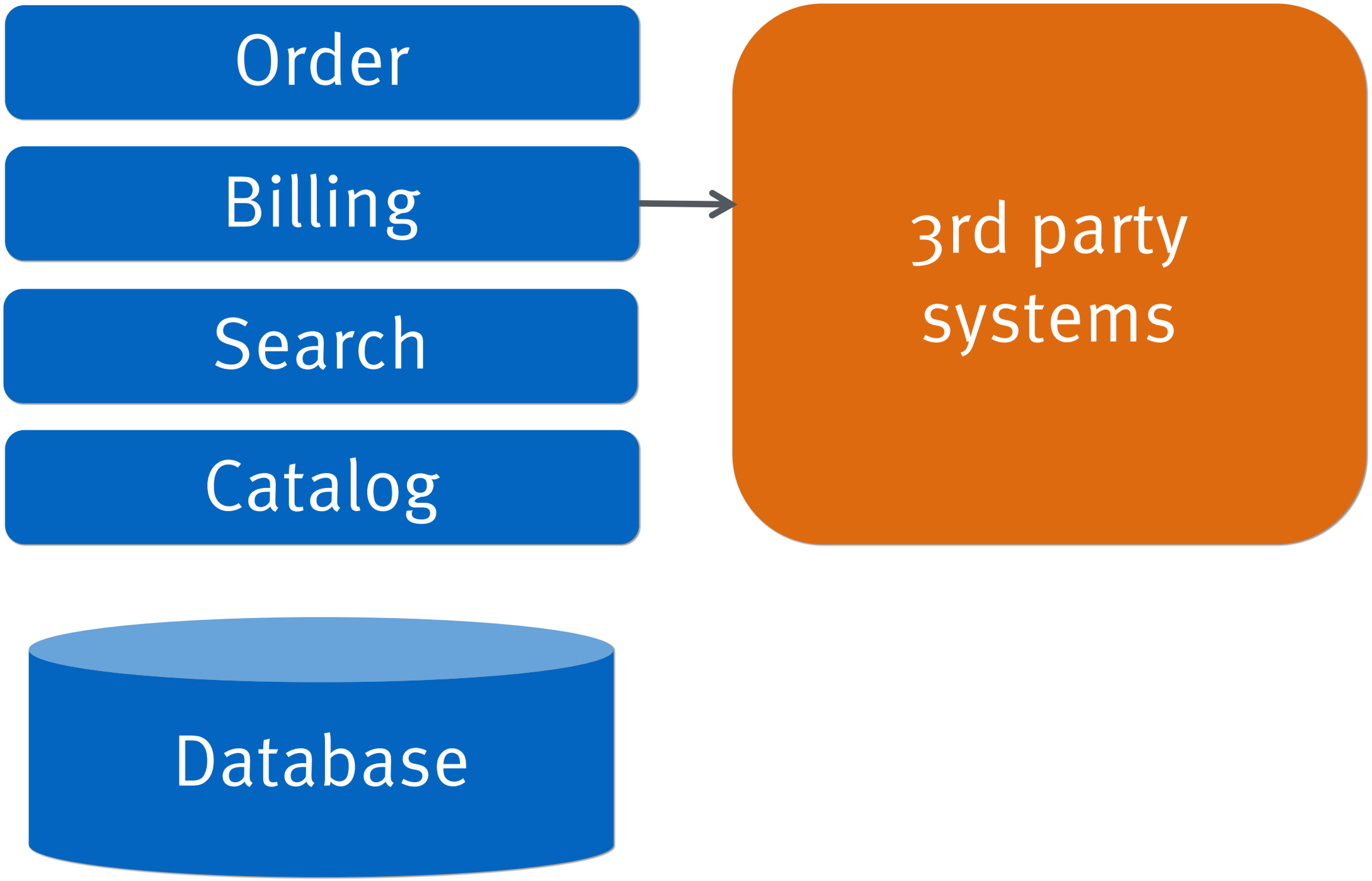
- Independent deployment units
- E.g. process, VMs, Docker containers
- Any technology
- Any infrastructure

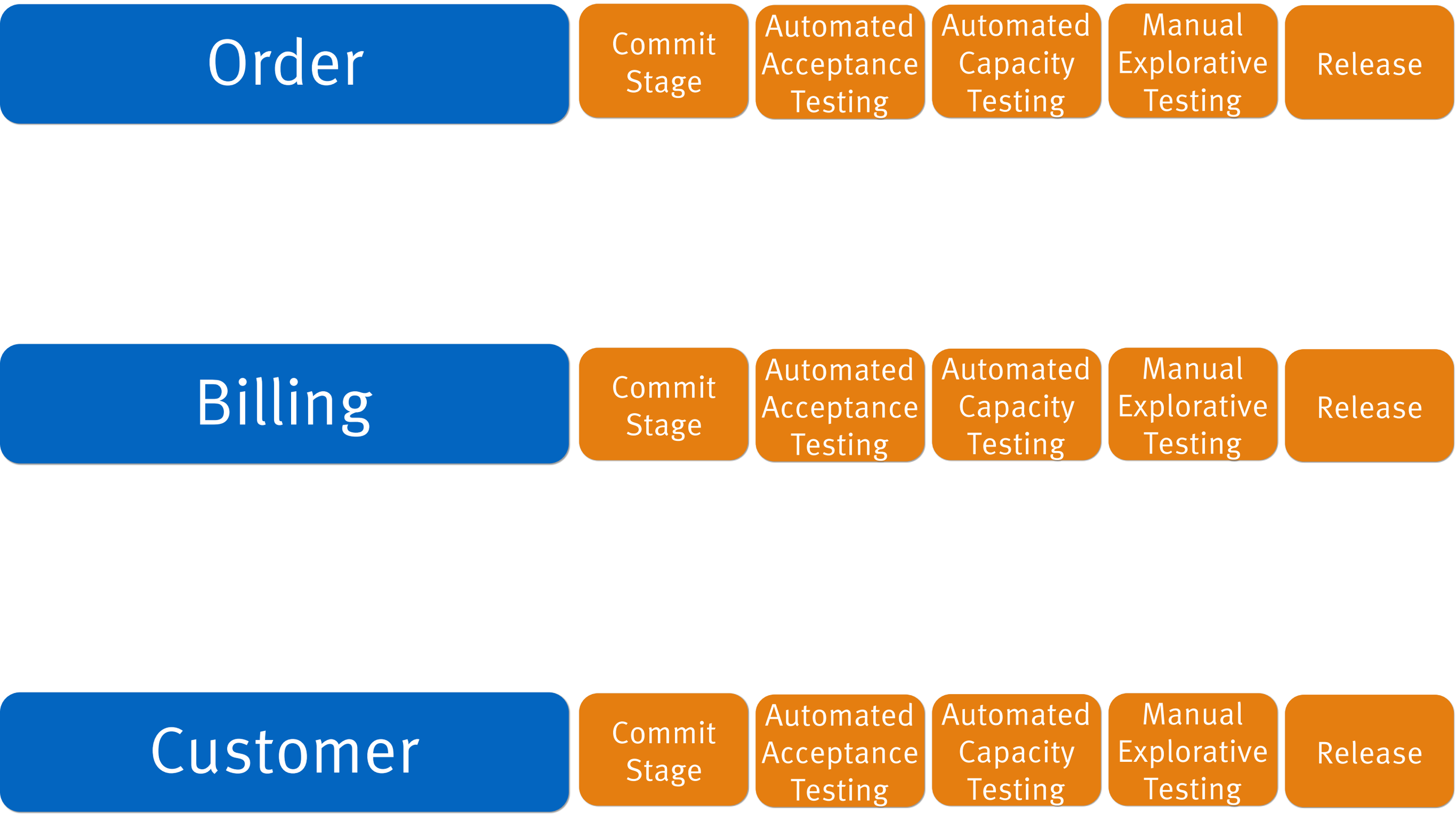


Microservices



Microservices




Order
Commit Stage
Automated Acceptance Testing
Automated Capacity Testing
Manual Explorative Testing
Release
Billing
Commit Stage
Automated Acceptance Testing
Automated Capacity Testing
Manual Explorative Testing
Release
Customer
Commit Stage
Automated Acceptance Testing
Automated Capacity Testing
Manual Explorative Testing
Release

Microservice Pipeline

- Build pipeline per Microservice
- Small
- Easy to set up
- Simpler (3rd party systems)
- Faster Feedback: Less tests
- Separate critical and less critical parts

Migrate to Microservices

- Add Microservices to existing system
- Implement new features in Microservices
- No need to redo the full application

Continuous Delivery doesn’t mean Microservices.

But can you do
Continuous Delivey
without supporting it
in the architecture?

Step Five

Adjust the architecture!

Conclusion

Final words

- Do no underestimate the effort!
- This is not about automation to save cost.
- It is about increasing quality!

Conclusion

- Deployment automation is just a prerequisite
- Understand the goals! Take pragmatic steps!
- Eliminate sign-offs! Create trust in tests!
- Deal with the gap between Dev and Ops!
- Adjust the architecture!

Thank You!

@ewolff