Micro Services – The New Architecture Paradigm

Eberhard Wolff

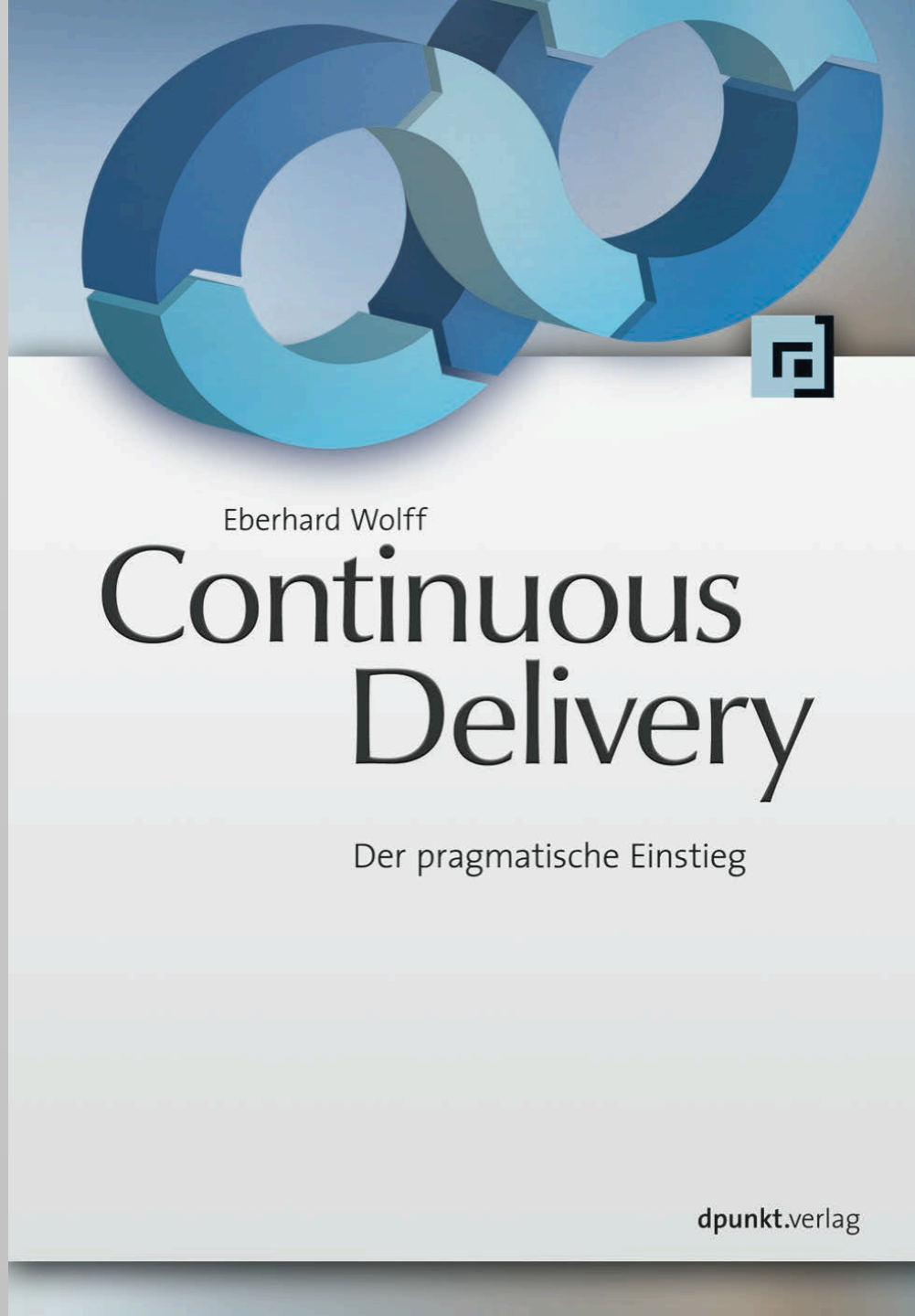
Freelancer

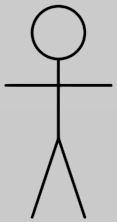
Head Technology Advisory Board

adesso AG

http://ewolff.com

Leseprobe:
http://bit.ly/CD-Buch

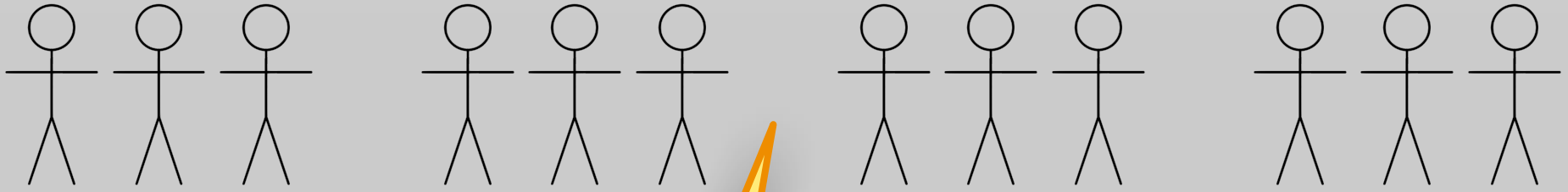




Change Order Process!

E Commerce Shop

Thou shalt not organize teams by technology!



Order Billing Search Catalog

E Commerce Shop

Deployment monolith

Common libraries & technical foundation

How to introduce e.g. Elasticsearch for Search?

Global code integration & coordination needed

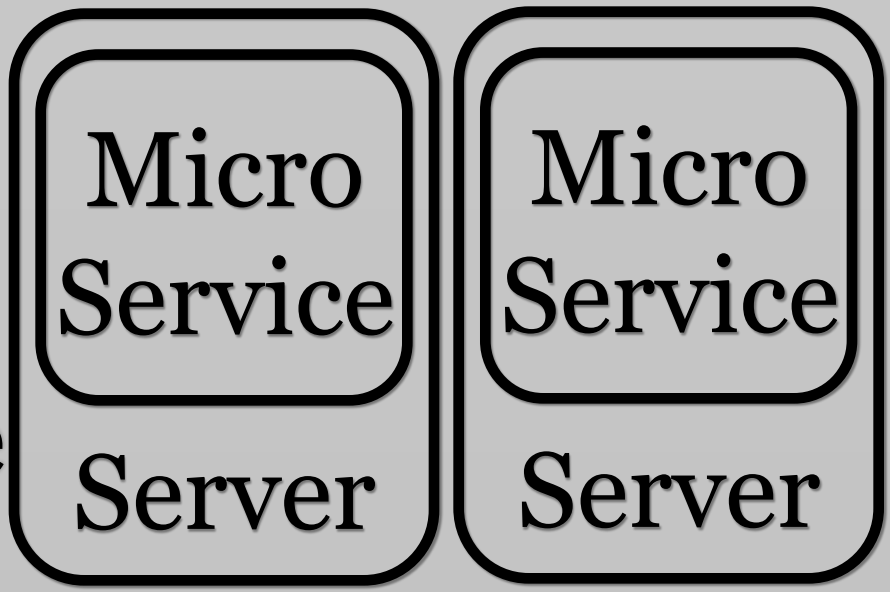
Code changes to production takes loooong

Architecture can be non-monolithic but dependencies might sneak in

Micro Services: Definition

- Small
- Independent deployment units
- i.e. processes

- Any technology
- Any infrastructure



Micro Services

- Component Model

- Component…
- Separate process
- Individual deployment unit
- GUI+Logic

Micro Services vs. SOA

- Micro Service:

1 service = 1 deployment unit

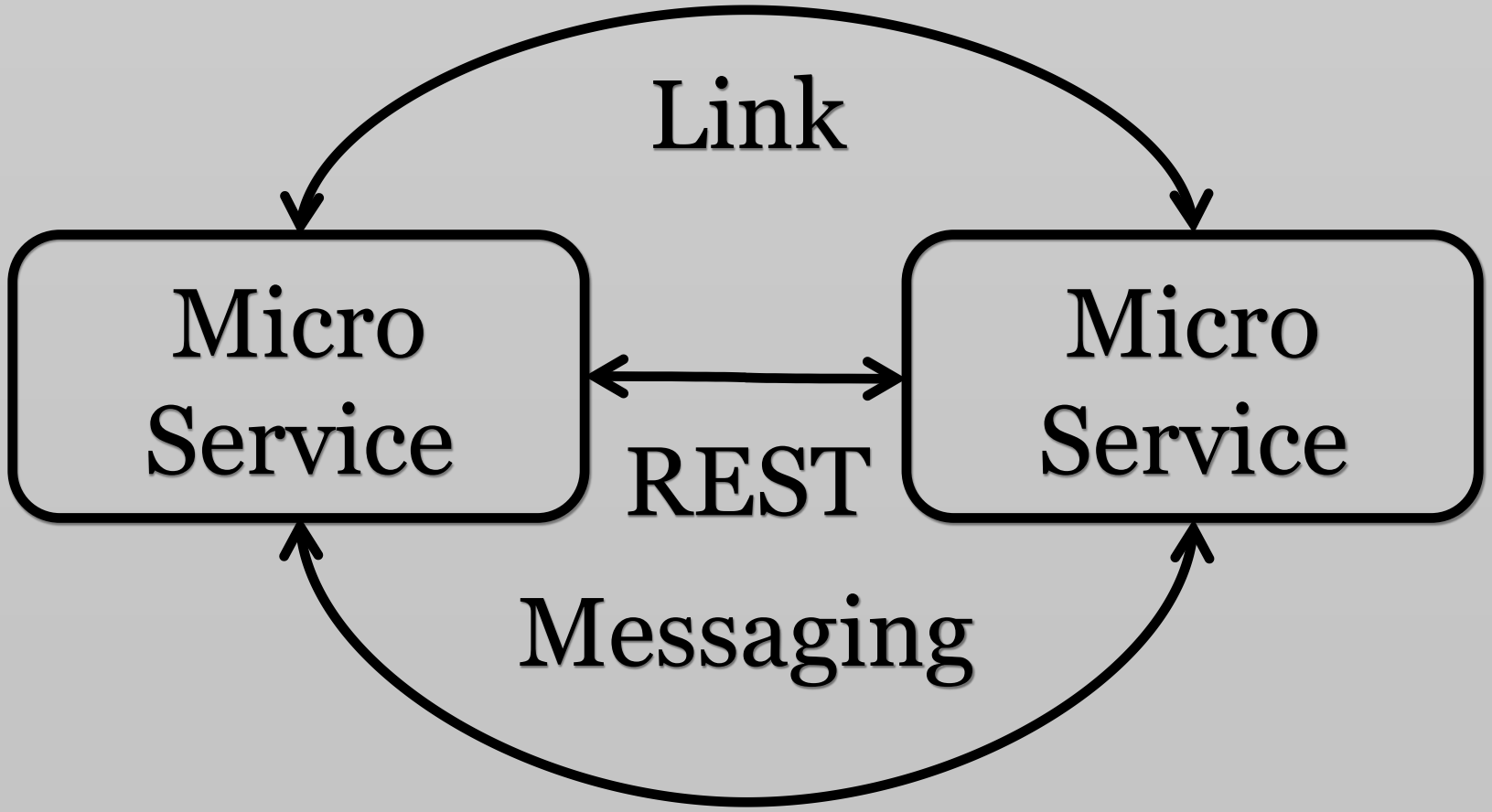
Service + GUI

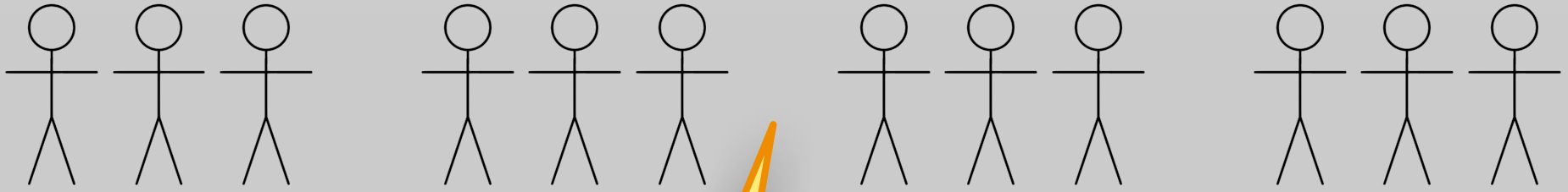
- SOA:

1 deloyment unit = n services

Service / GUI separate

Components Collaborate





E Commerce Shop

Deployment monolith

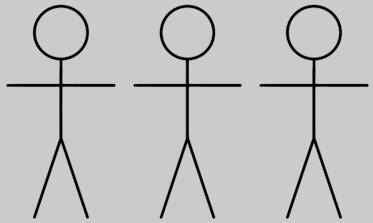
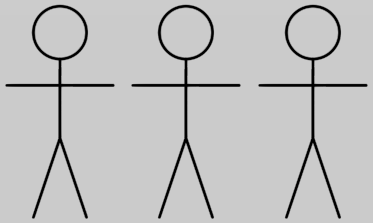
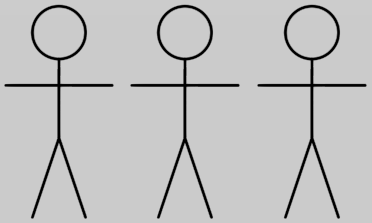
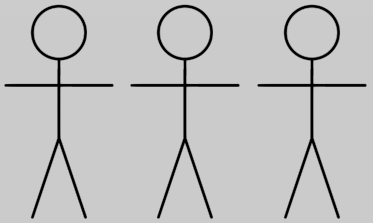
Common libraries & technical foundation

How to introduce e.g. Elasticsearch for Search?

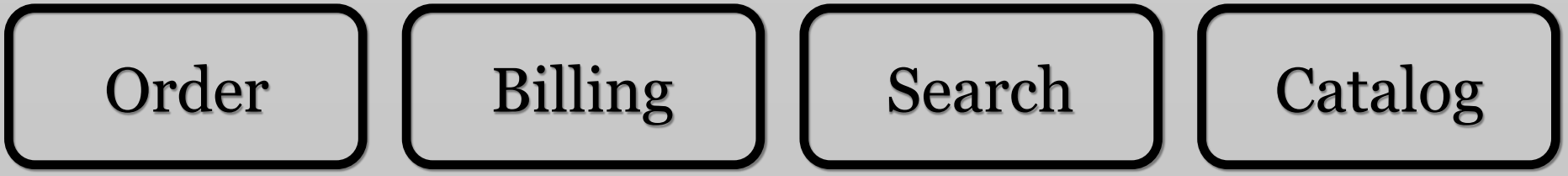
Global code integration & coordination needed

Code changes to production takes loooong

Architecture can be non-monolithic but dependencies might sneak in

Order Billing Search Catalog



Micro Services

Common libraries & technical Micro Services

One or many Micro Services per Team

How to introduce e.g. Elastic search for Micro Service

Technology stack per Micro Service?

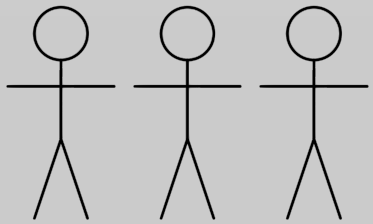
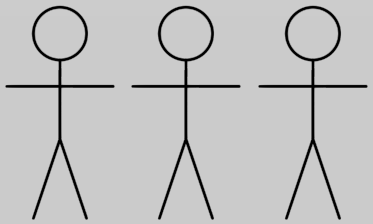
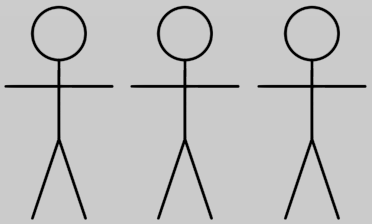
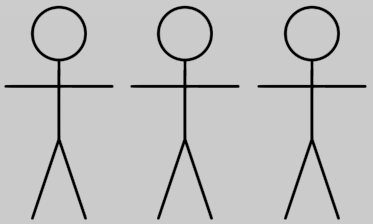
Global code integration & coordination needed

Team can deploy without integration

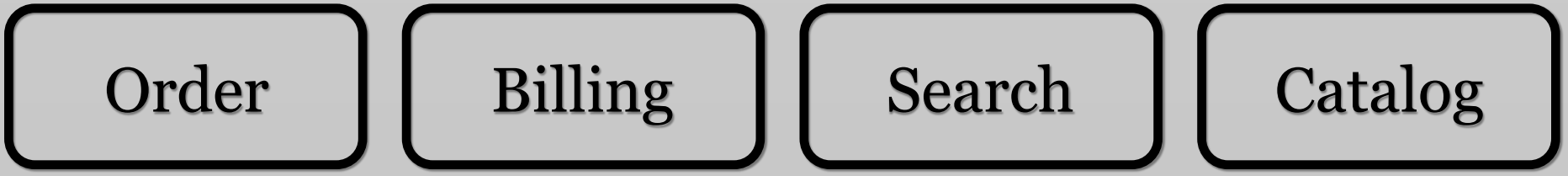
Code changes to product can be deployed independently & quickly

Changes to product taken in pending

Architecture can be non-monolithic but dependencies might sneak in

Order Billing Search Catalog



Micro Services

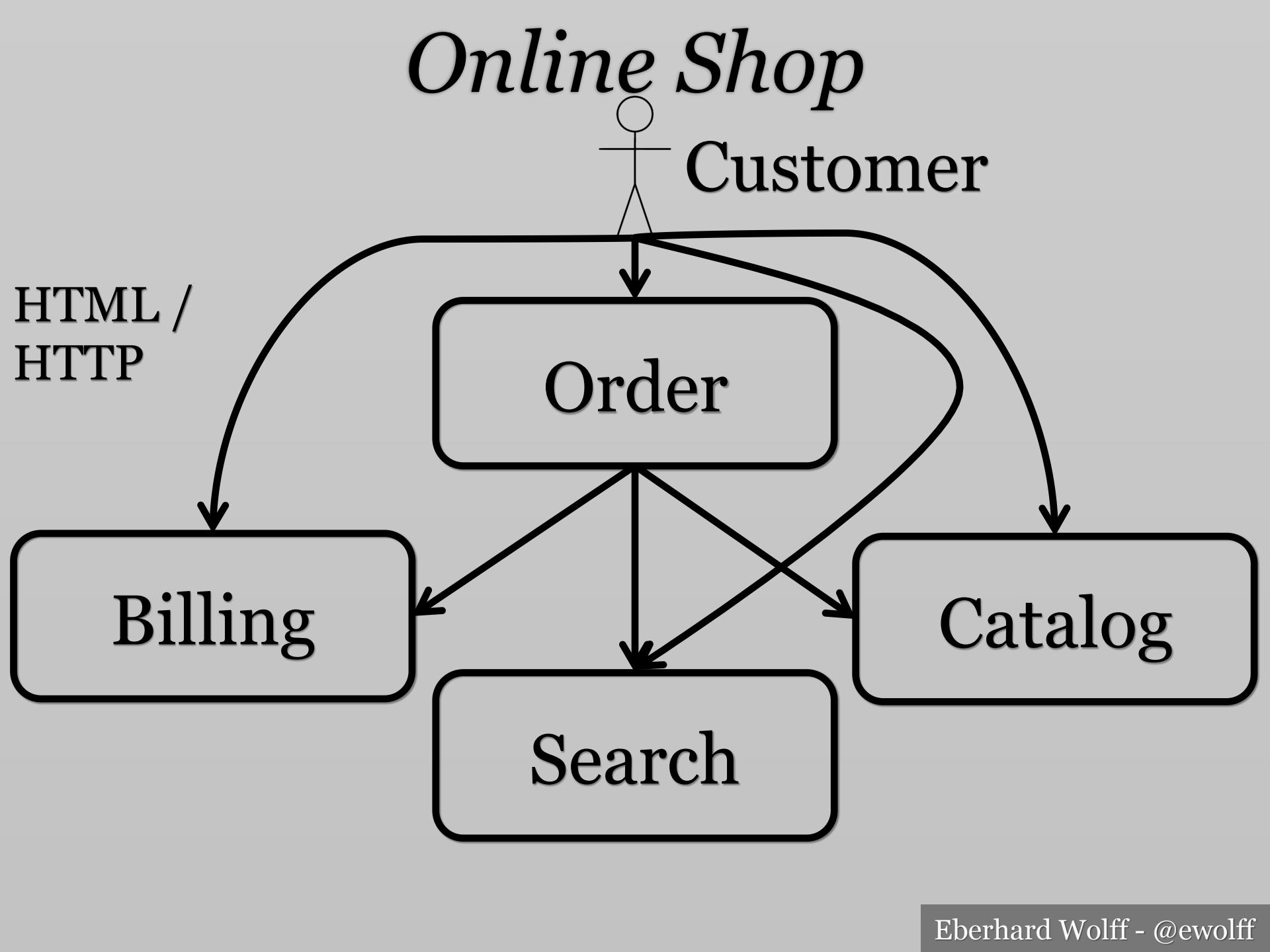
One or many Micro Services per Team

Technology stack per Micro Service

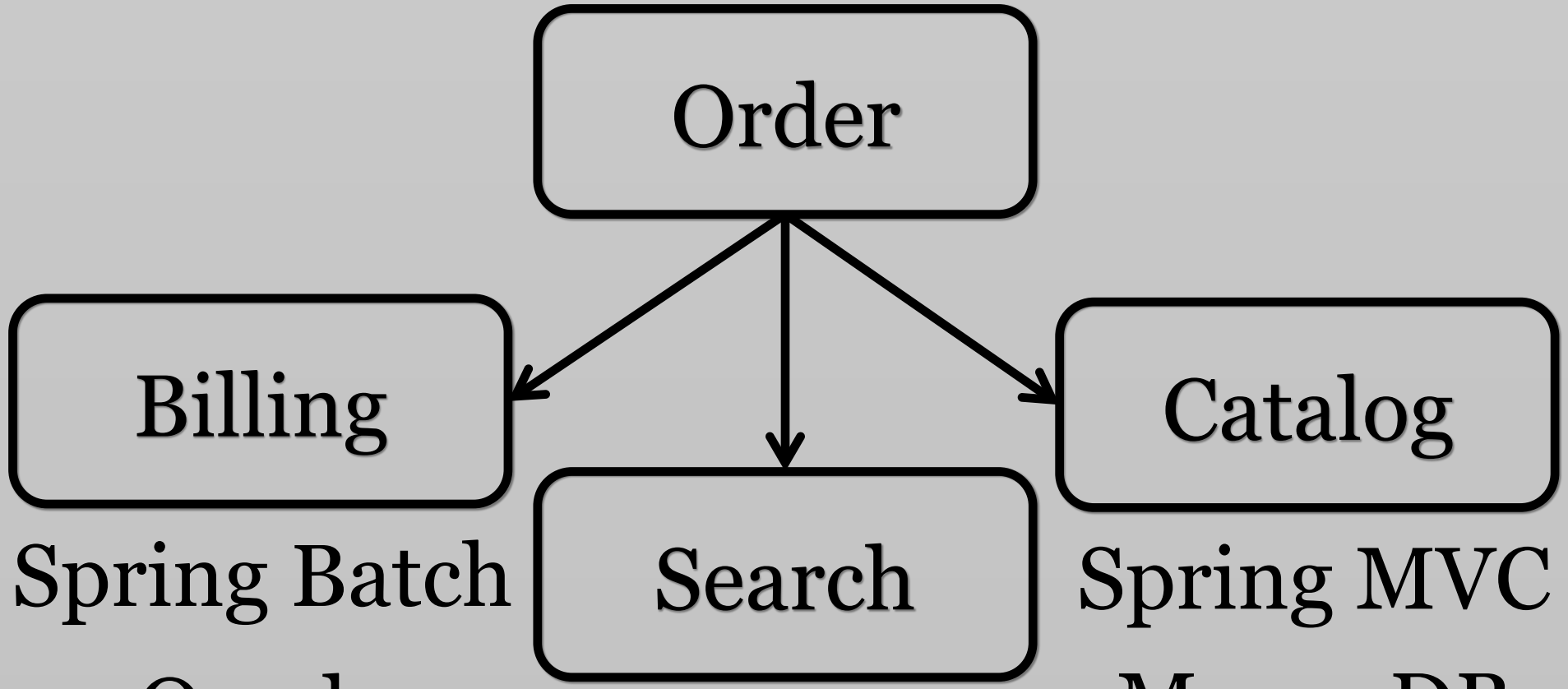
Team can deploy without integration

Changes can be deployed independently & quickly

Strong & enforced modularization


Online Shop
Customer
HTML /
HTTP
Order
Billing
Catalog
Search

Online Shop


Order
Billing
Search
Catalog
Spring Batch
Oracle
Elasticsearch
Spring MVC
MongoDB

Deploy & Operate?

Component Model

- No restriction on language etc
- Individual processes
- + infrastructure (database etc)
- JARs, WARs, EARs: No good fit

- Monitoring
- Logging

Possible Component Models

- Virtual machine
- Docker container
- Installable software (RPM, deb)
- + deployment / config scripts

Why Micro Services?

How to scale agile?

Implement more feature

Conway's Law

Architecture
copies
communication structures
of the organization

Conway's Law as a Limit

- Won't be able to create an architecture different from your organization

- I.e. mobile, GUI & database team
- Three technical artifacts

Conway's Law as an Enabler

- Desired architecture = project structure
- Team for each Micro Service
- Team should be responsible for meaningful features
- Ideal: Independent features

One team can build and deploy features independently!

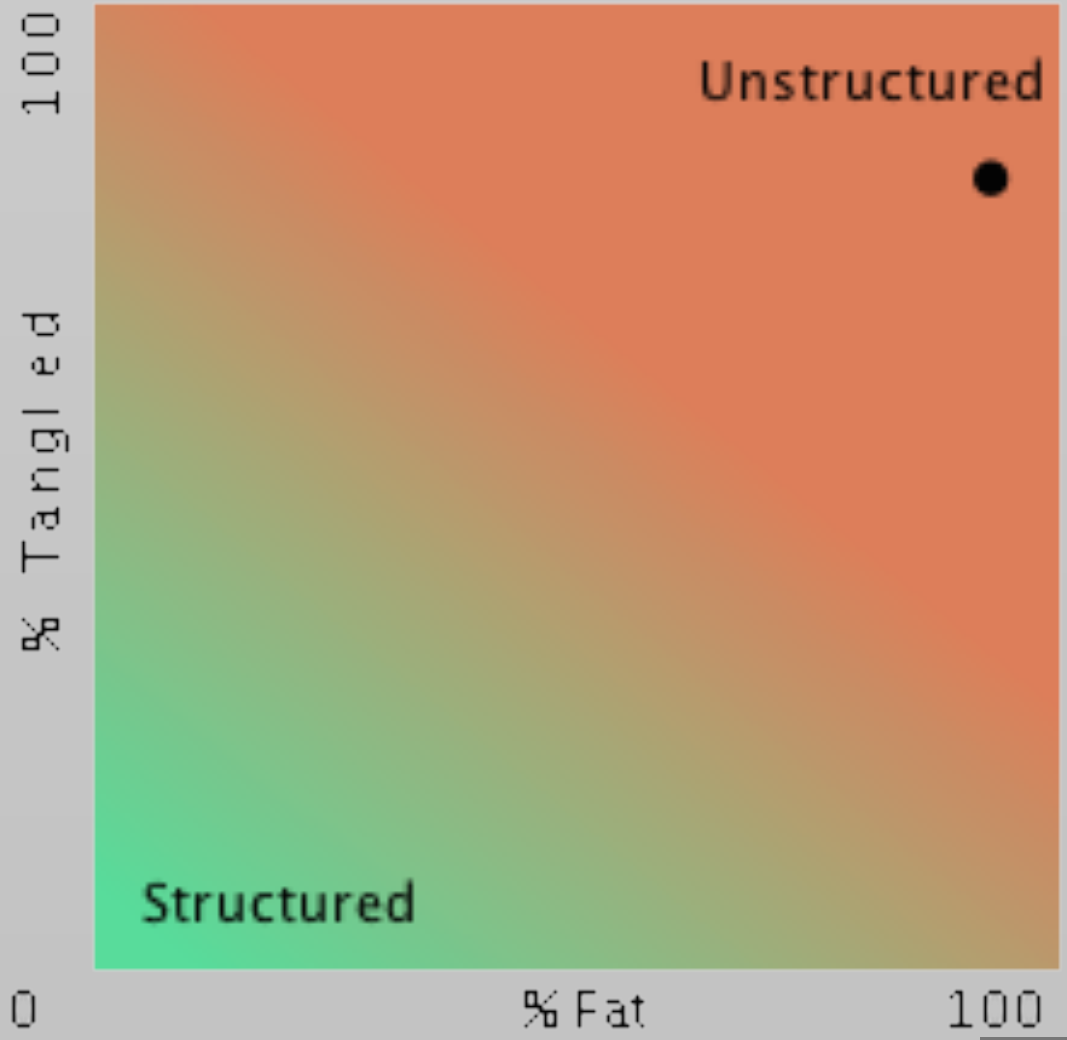
Team must be responsible for a sensible set of functionality

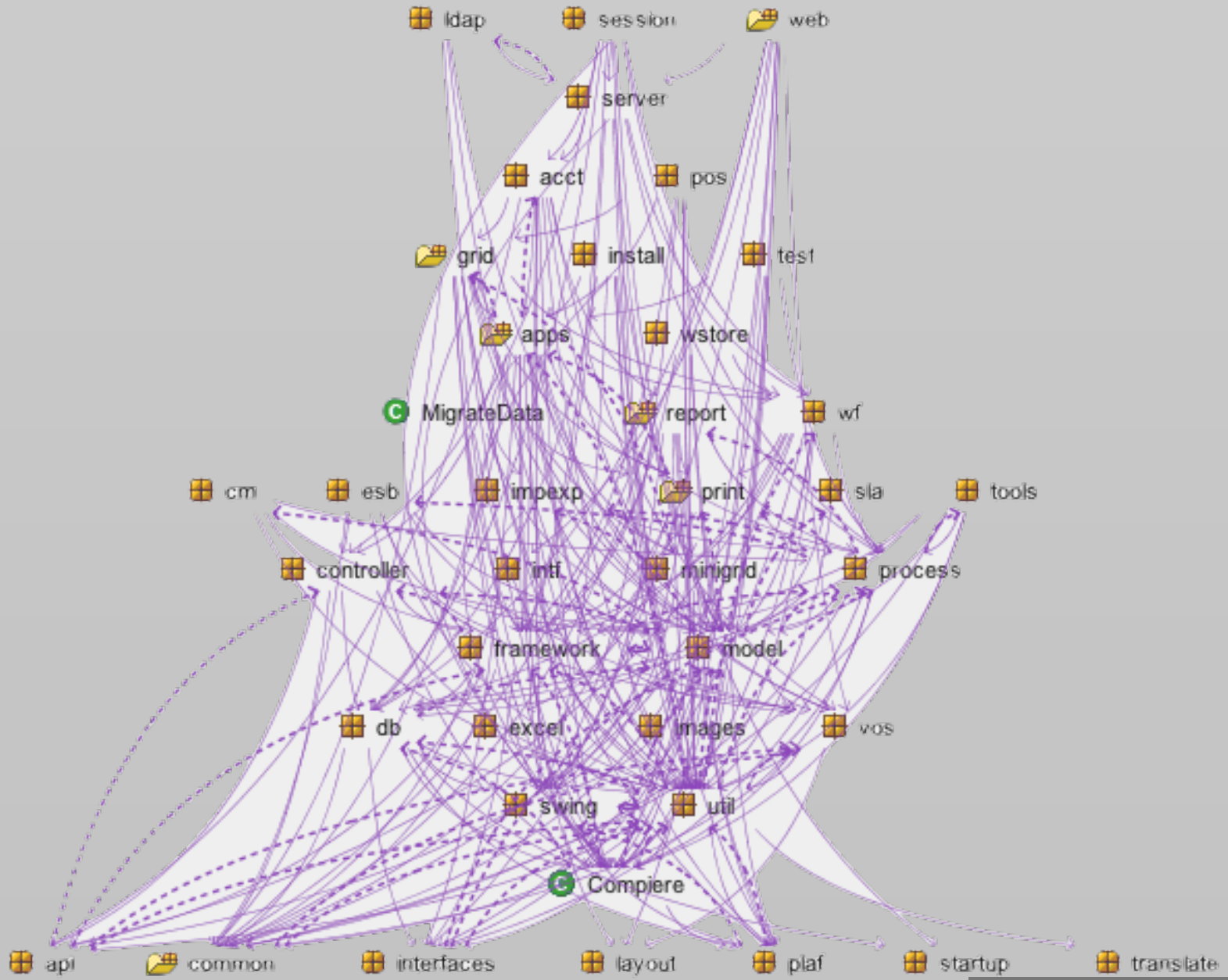
Conway's Law & Size

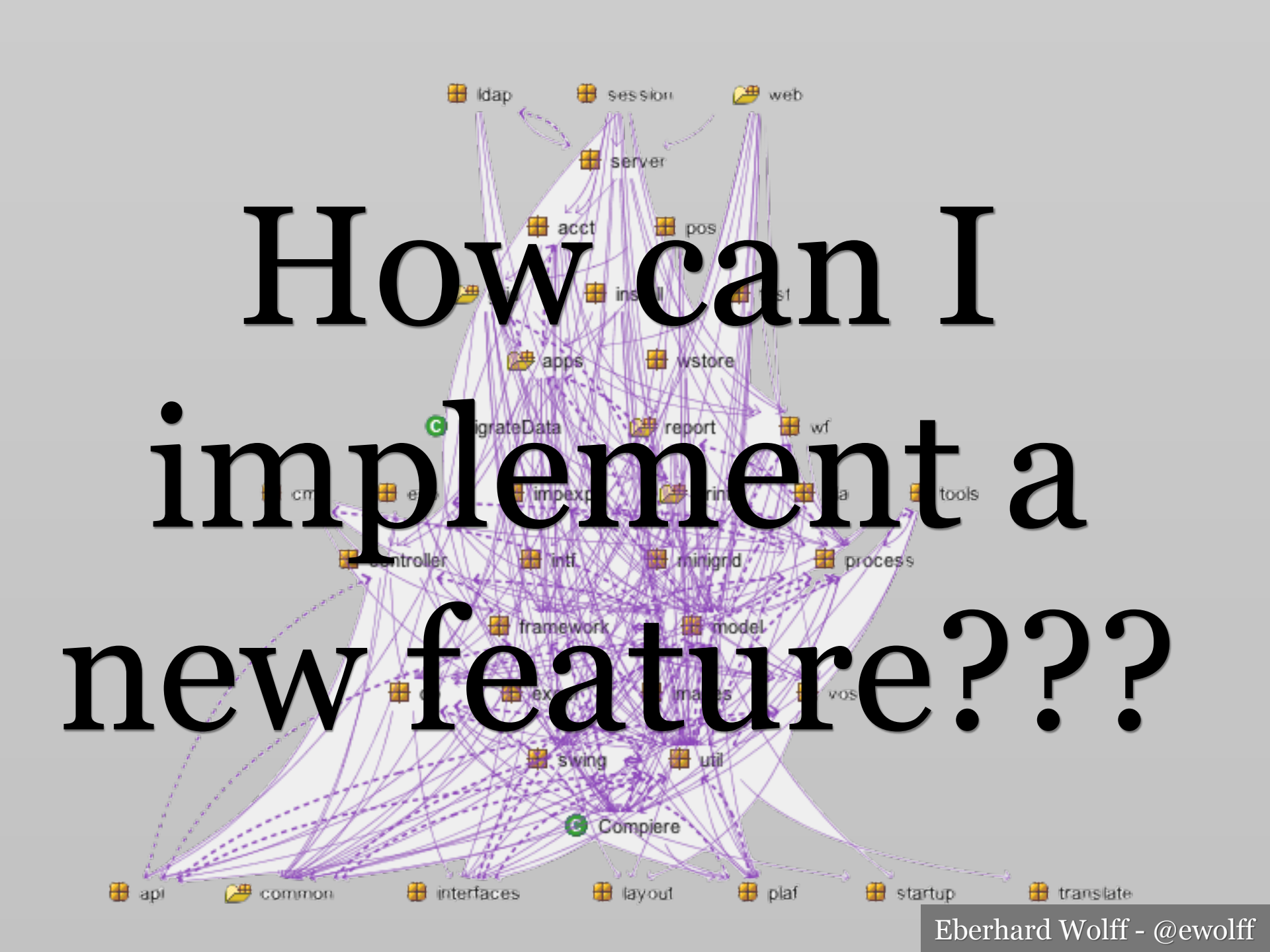
- Upper limit: What a (small) team can handle
- …and a meaningful set of features
- Probably not too small
- Smaller modules might be better
- Lower limit: Depends on overhead / technology

Legacy Apps

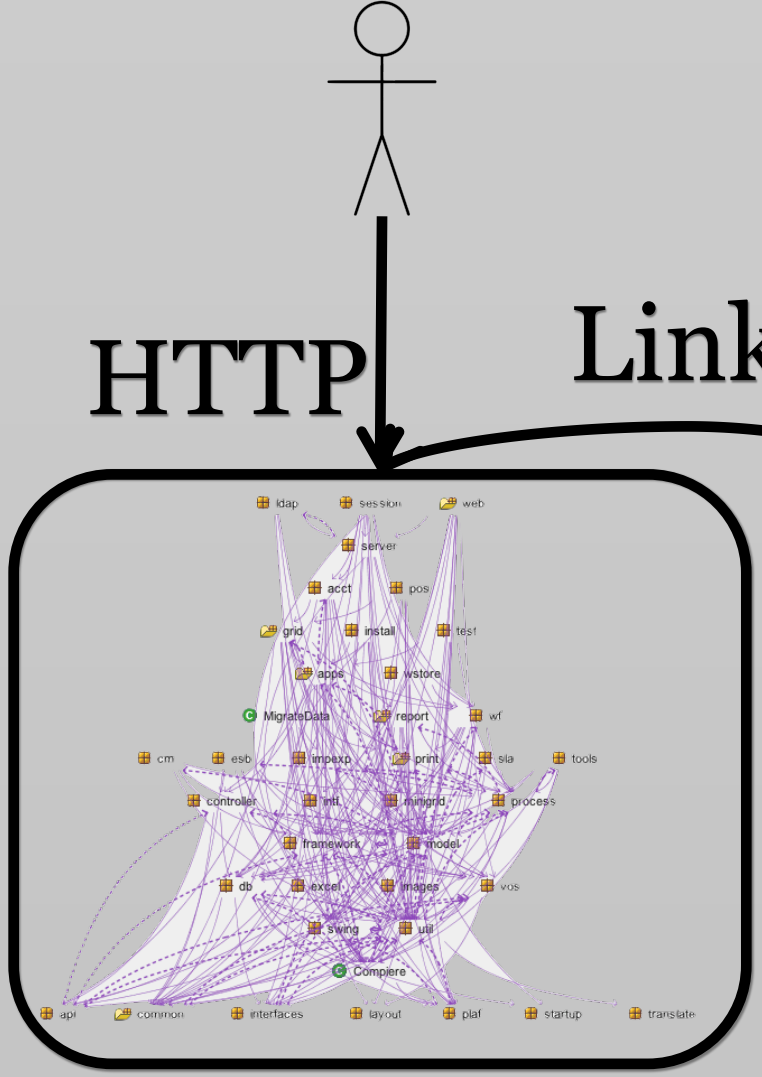
Top tangles and fat	Size	Problem
org.compiere	955.512	Tangled
org.compiere	1.080.0...	Fat




ldap
session
web
server
acct
pos
grid
install
test
apps
wstore
MigrateData
report
wf
cm
esb
impexp
print
sla
tools
controller
intf
minigrid
process
framework
model
db
excel
images
vos
swing
util
Compiere
api
common
interfaces
layout
plaf
startup
translate



How can I implement a new feature???



New Stuff

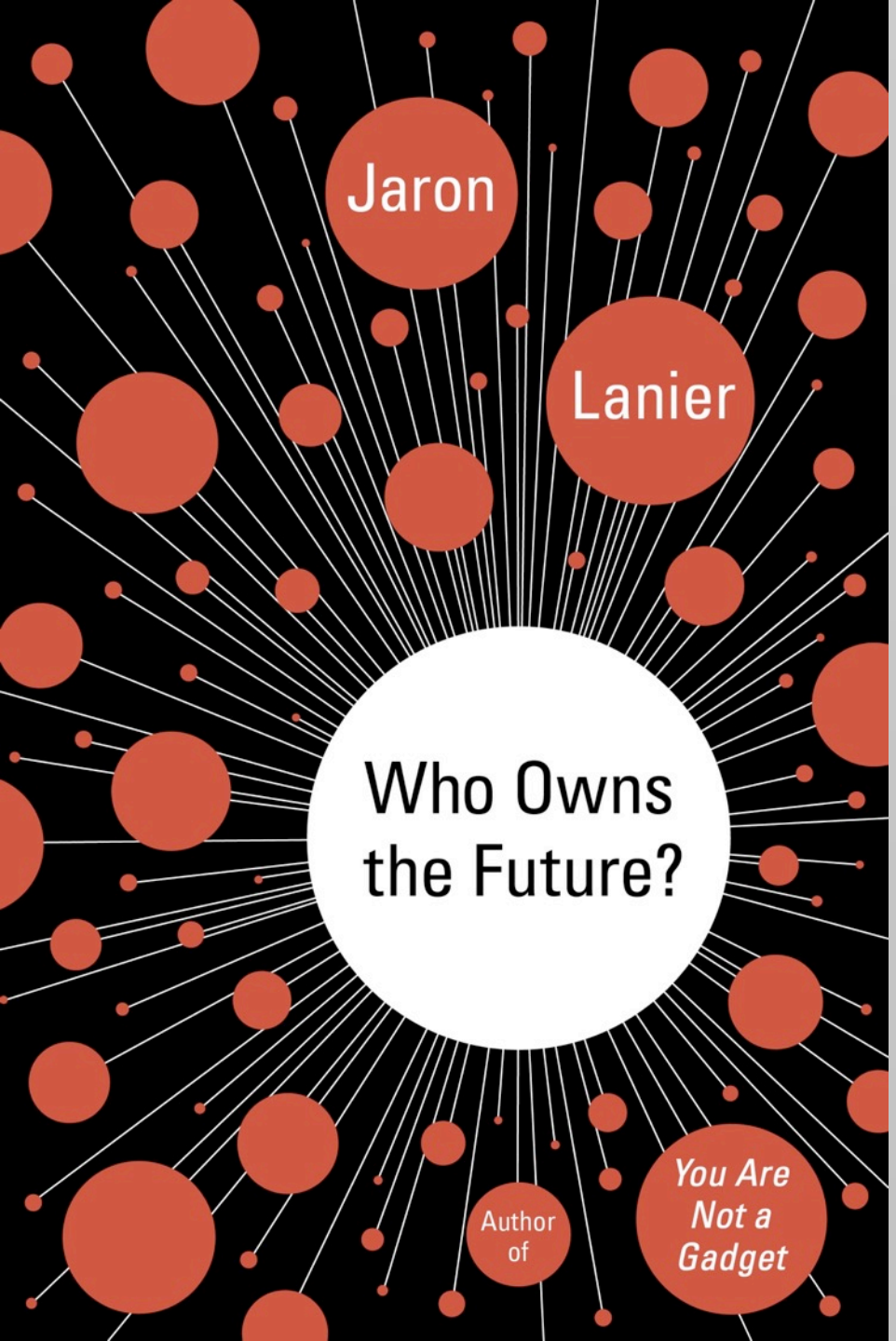
No legacy code

Any technology

Small code base

Little programs are delightful to write in isolation,
but the process of maintaining large-scale software is always miserable.

Jaron Lanier



Friedenspreis des deutschen Buchhandels

Sustainable Development

Monoliths

- Architecture rot
- …not maintainable any more
- …and can’t be rewritten / replaced

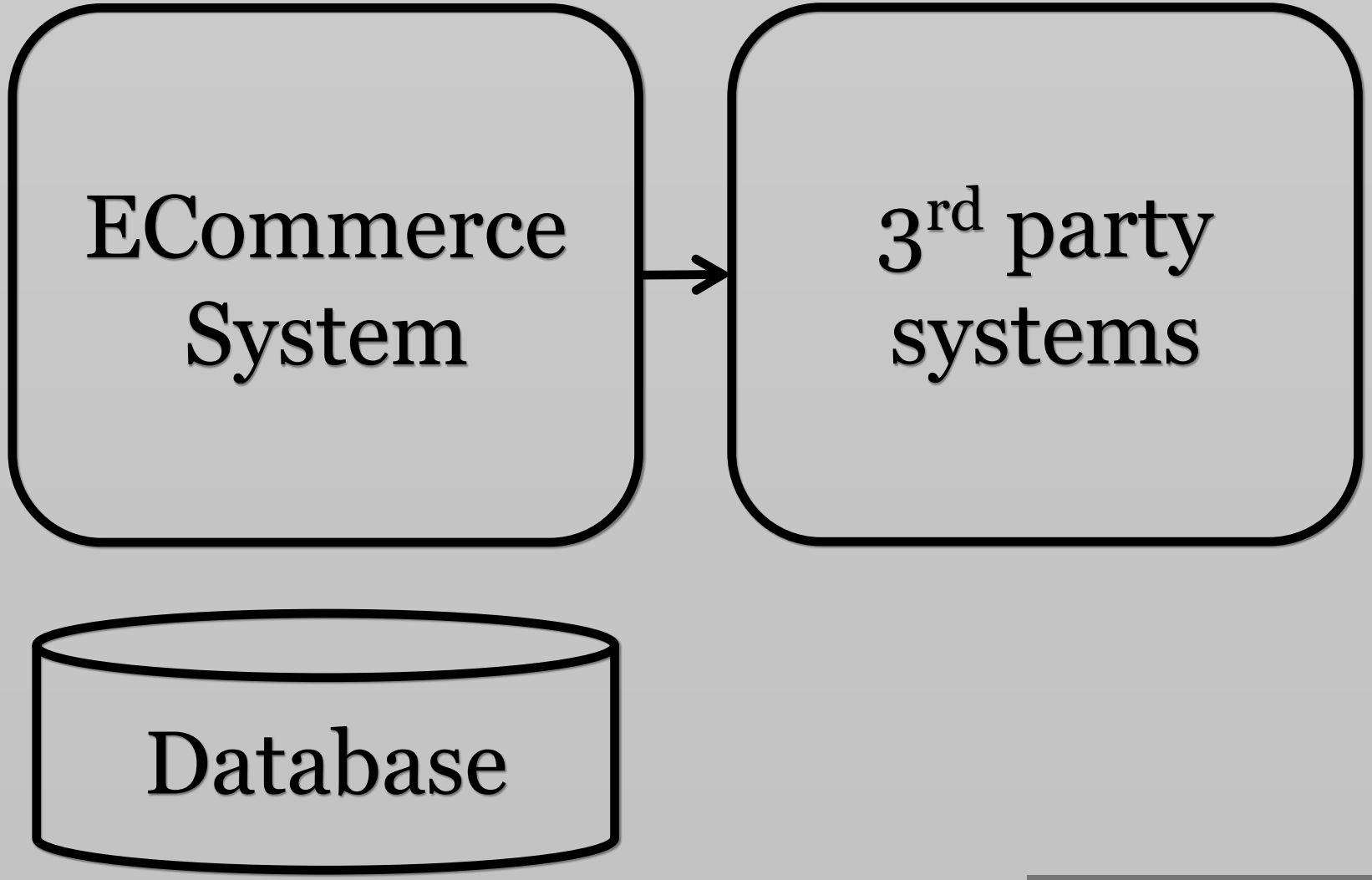
Micro Services

- Distributed system of small units
- Architecture violations harder

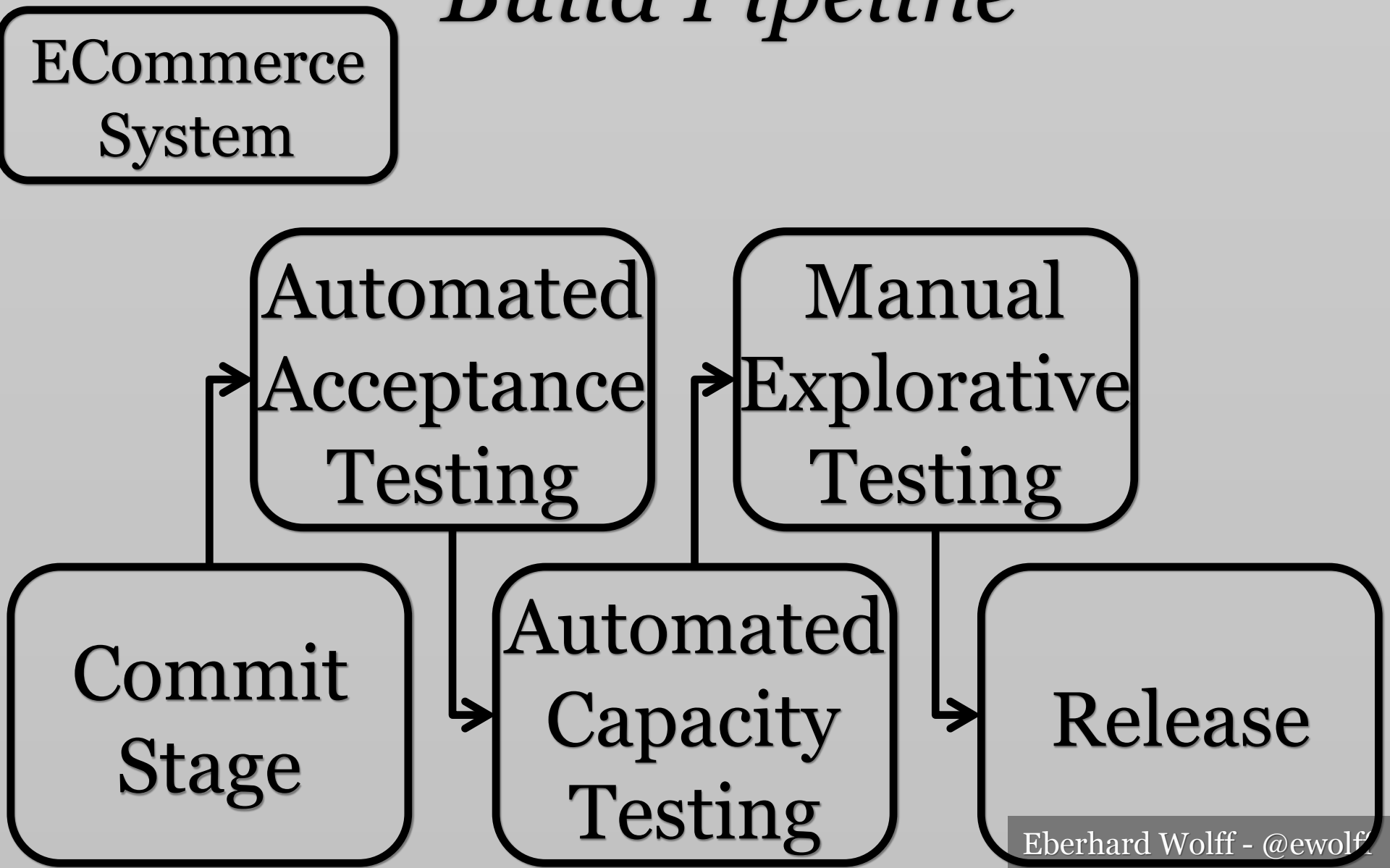
- Small units
- Easy to replace

Continuous Delivery

Monolith



Continuous Delivery: Build Pipeline

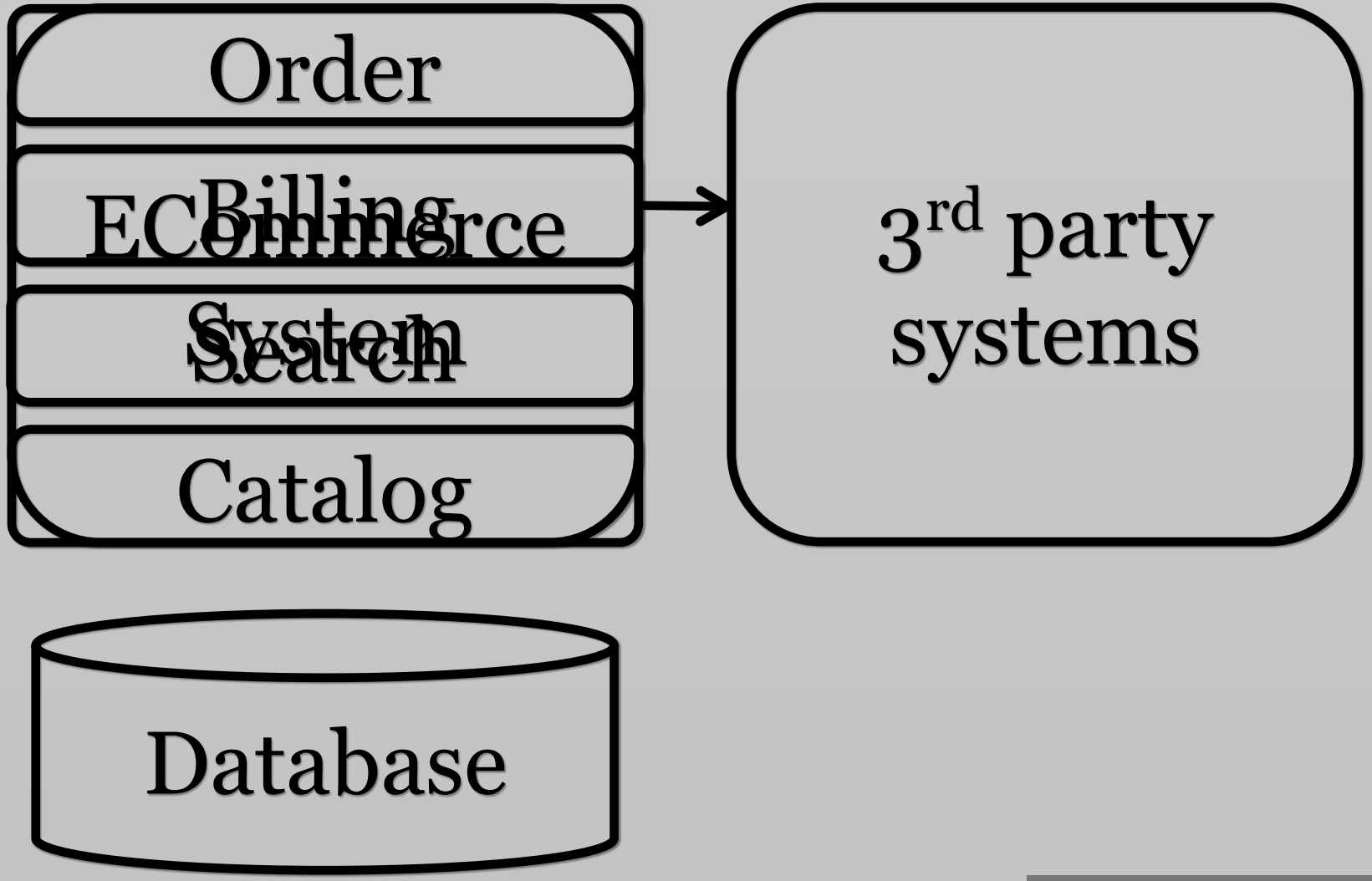


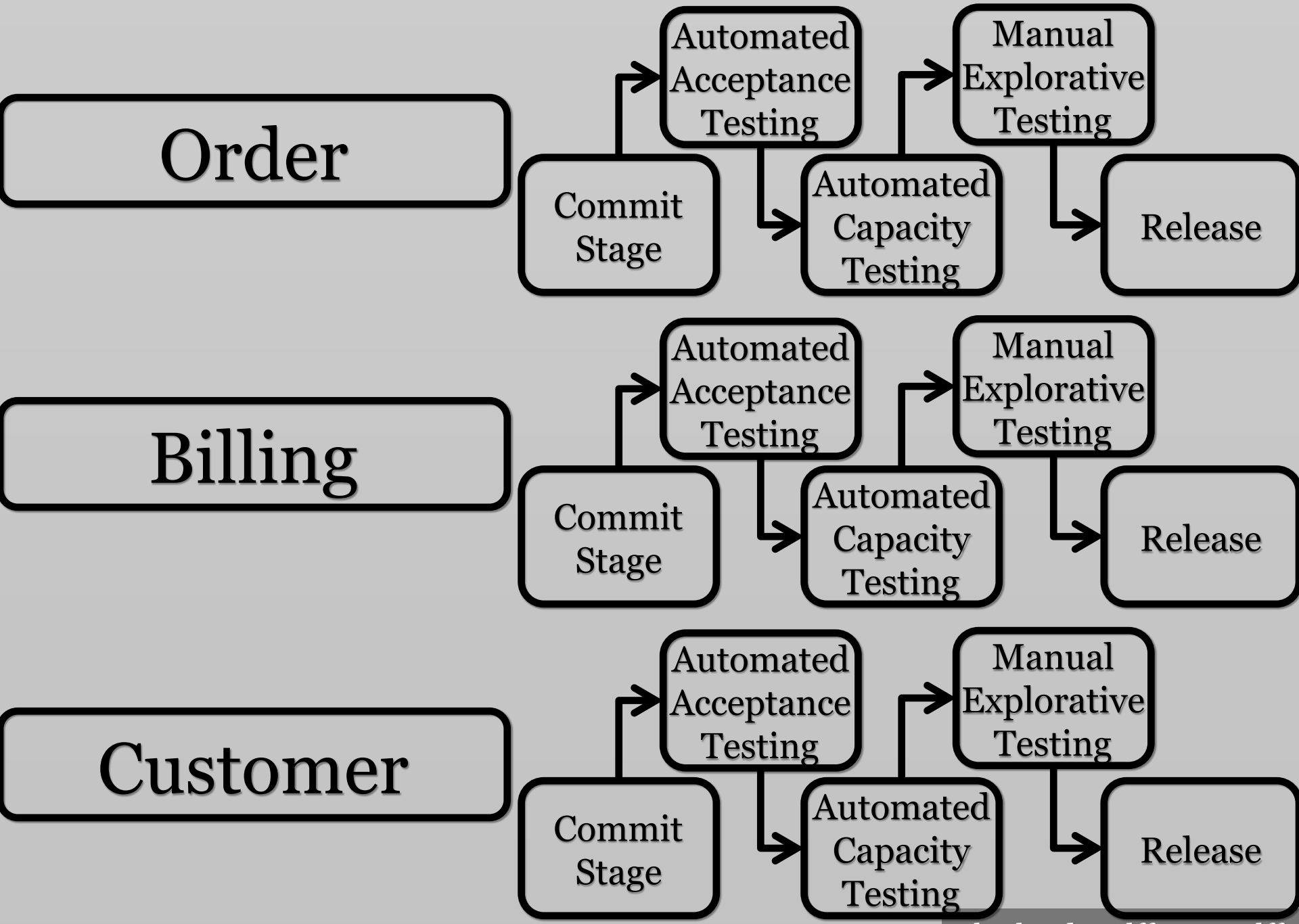
Build Pipeline: Problems

- Complex infrastructure
- Huge database
- 3rd party integration

- Slow feedback
- Test everything for each commit
- Huge deployment unit
- Deployment slow

Micro Services



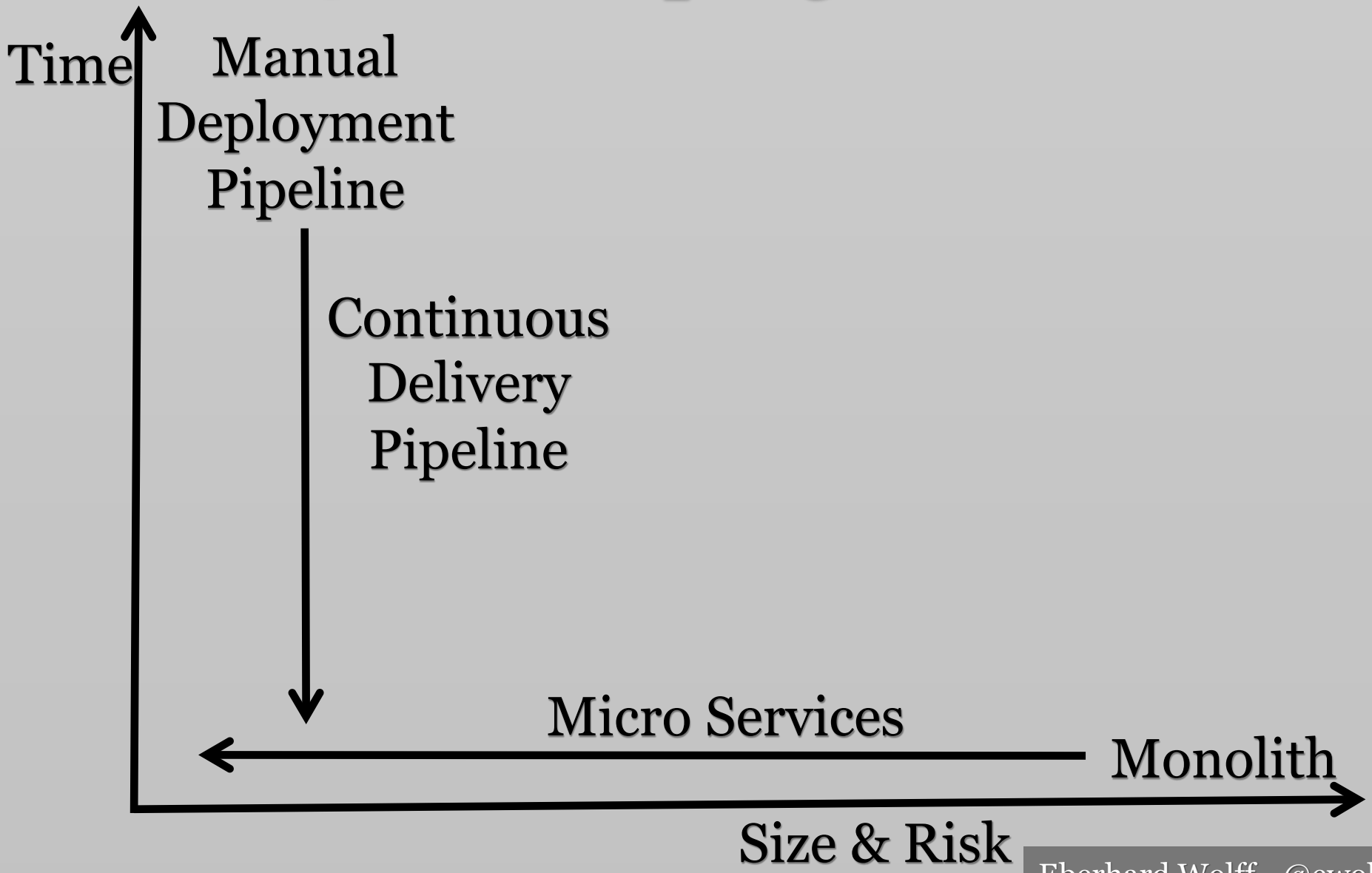

Order
Commit Stage
Automated Acceptance Testing
Automated Capacity Testing
Manual Explorative Testing
Release
Billing
Commit Stage
Automated Acceptance Testing
Automated Capacity Testing
Manual Explorative Testing
Release
Customer
Commit Stage
Automated Acceptance Testing
Automated Capacity Testing
Manual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

Build Pipeline for Micro Services

- Independent deployment
- Build pipeline per Micro Service

- Smaller
- Easier to set up
- Less features (3rd party systems)
- Faster Feedback: Less tests

Quick Deployments



Why Micro Services?

- Strong modularization
- Small deployment units
- Faster & easier deployment
- Sustainable development speed
- Continuous Delivery
- Less risk in deployment
- Best technology for each service

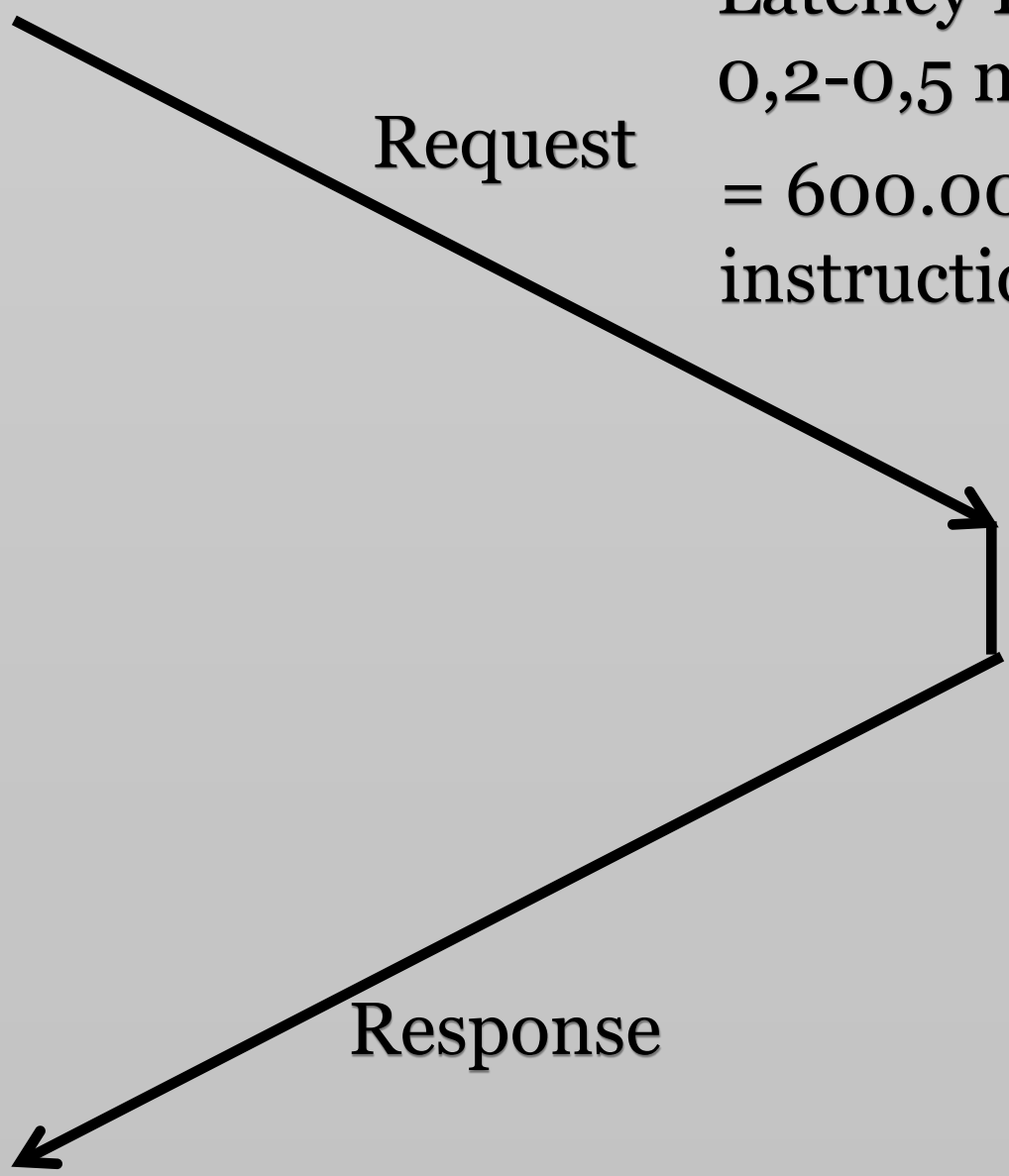
Challenges

Distributed System

1st Law of Distributed Objects

- Don’t Distribute Your Objects!
- Too much remote communication & overhead
- Lesson learned from CORBA etc
- Micro Service should include a GUI

- http://martinfowler.com/bliki/FirstLaw.html



Latency Round Trip:
0,2-0,5 ms

= 600.000-1.500.000 instructions@3GHz

1st Law of Distributed Objects & Micro Services

- Small Micro Services mean a lot of communication
- Violate the 1st Law
- Seems to work, though

- http://martinfowler.com/articles/distributed-objects-microservices.html

Too small = too much communication

Code Reuse

- Reuse across technology stacks hard
- Code dependencies are evil!
- Deployment dependency
- No more independent deployment
- Update hell

- Avoid code reuse!
- Or make it Open Source projects (Netflix)
- Service reuse is fine

Global Refactorings?

- Move code from service to service
- Might be a port to a different language
- Separate in a new service?
- More services = more complex

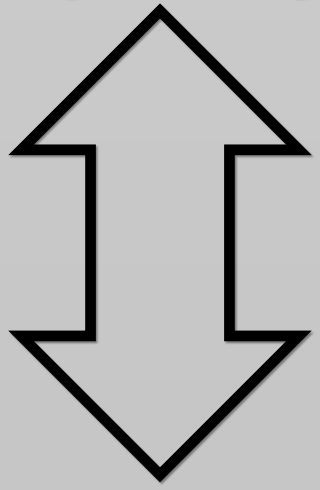
- Very hard

Functional Architecture

- Teams should be independent
- i.e. one team = one functionality
- Otherwise: Coordination hard

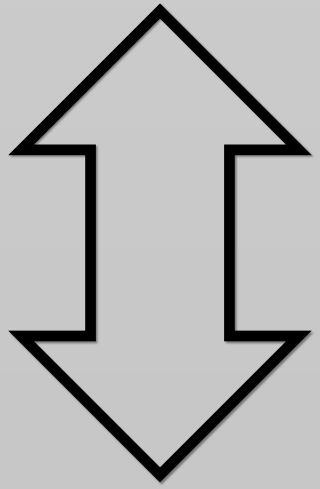
- Functional architecture much more important

Refactoring & Code Reuse



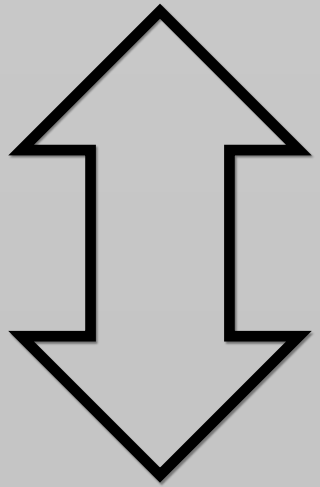
Individual Technology Stacks

Refactoring hard



Functional architecture much more important

Need to get architecture right first time



Architecture evolves

Start BIG

- Won't have too much code at the start anyway
- Refactoring easier
- Can build architecture as you go

- Let the functional architecture grow!

Handling Interfaces

- Independent deployments = backwards compatibility
- Different versions of the same service run at the same time
- Only to allow updates
- i.e. transient

- Minor issue

Managing Dependencies Between (>100) Services?

- Monoliths can be easily analyzed
- Micro Services?
- With heterogeneous technology stacks?

- Need to come up with your own solution

Global Architecture?

- Manages dependencies
- Which service/team does what?
- Defines common communication infrastructure
- Optional: Common Ops

- Very different from usual architecture

Network?

- Micro Services = Distributed Systems
- Services & network might fail
- Need to deal with failure

- Resilience: System must survive failure of parts
- Makes system highly available

Deployment?

- One Deployment Pipeline per services
- Should be fully automated
- Too many services for manual steps

- Infrastructure investment needed
- Common deployment?

Infrastructure?

- >50 server per system
- Resource consumption probably high

- Virtualization must be fully automated
- Docker might save ressources

Conclusion

Conclusion: Micro Services

- Micro Services are a new way of modularization
- More technological freedom
- Easier, faster and less risky deployment

Use If...

- Time to market is important
- Legacy systems must be modernized
- Sustained development speed
- Continuous Delivery should be implemented
- Large enough project

Don't Use If...

- Infrastructure complexity cannot be handled
- Architectural complexity cannot be handled

Thank You!