Java App Servers are Dead!

Eberhard Wolff
Freelancer
Head Technology Advisory Board adesso
http://ewolff.com


Code

2003

EJB – Und jetzt?
Oder: Hat der Kaiser keine Kleider?

Eberhard Wolff

App Server =

Java EE or

Servlet

Container

An Application on a server needs an Application Server!

Why??

The Price We Pay

What now?

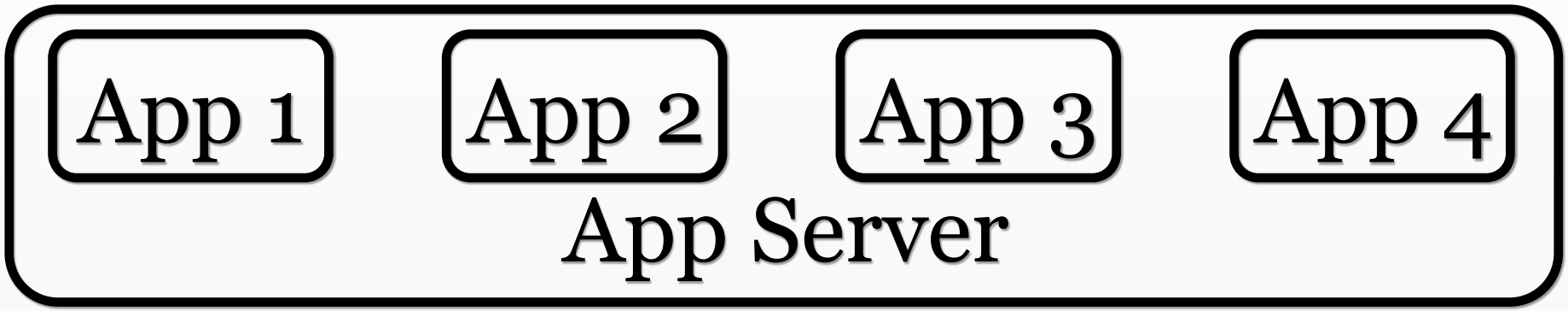
App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

Multiple Applications



- Isolation
- ClassLoader
- Can lead to non-trivial problems

Isolation

- ClassLoader is not enough
- CPU?
- Memory?
- Filesystems?
- Applications are not isolated
- Even individual parts are not isolated
- i.e. JMS might eat away resources from web requests

Isolation is Impossible

- Operating systems isolate processes from each other
- CPU, memory ...
- Resource allocation: #1 feature for operating systems
- Either the JVM becomes an operating systems
- ...or isolation won't be perfect

Multiple Applications

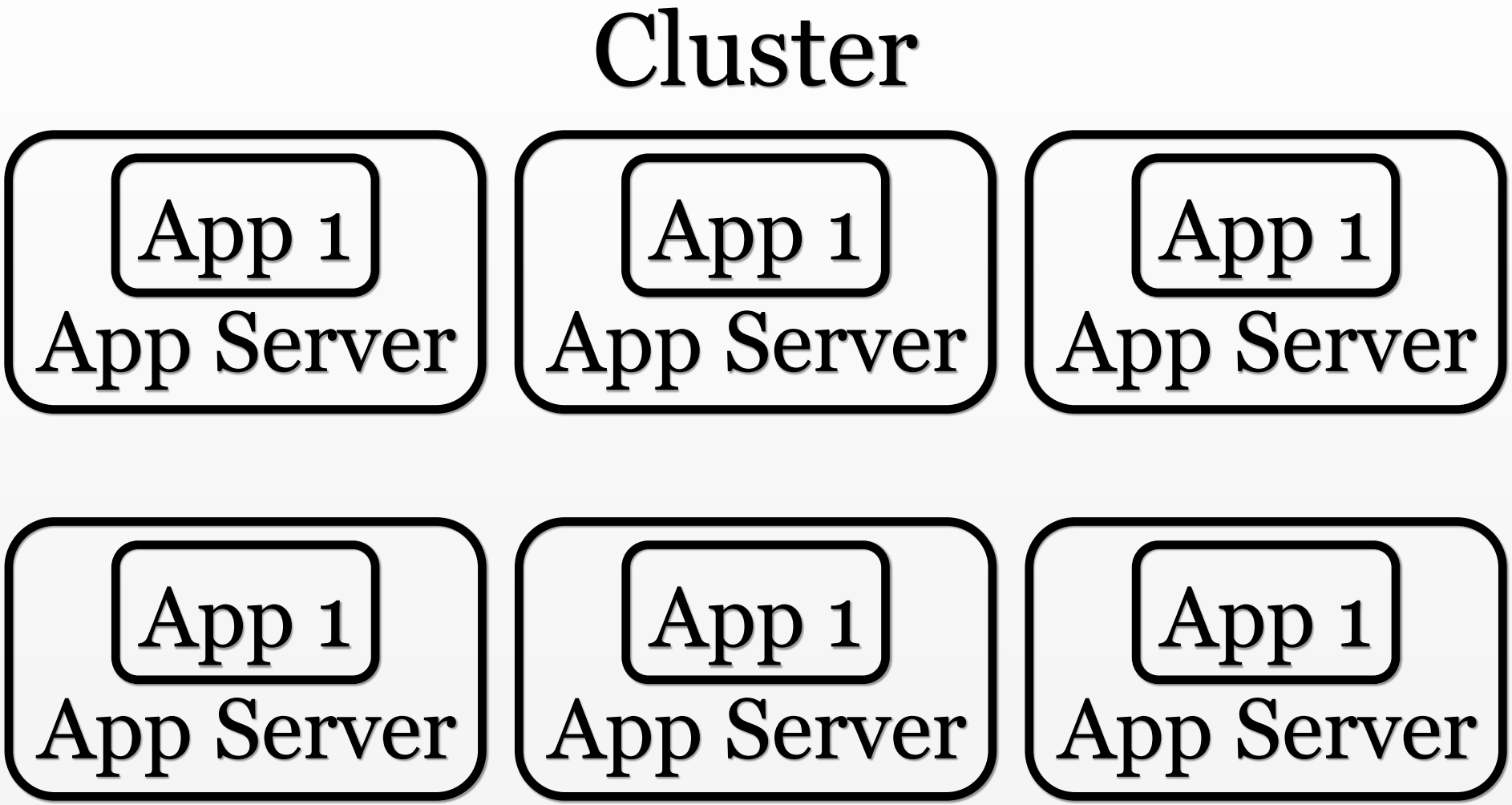
- Is that really what happens?
- Java EE spec talks about "components"
- not Apps

One Application

One Application

- Component e.g. WAR, EJB JAR ...
- Different ClassLoader isolation needed
- OSGi like
- Memory / CPU isolation still useful

What It Is More Like…



App Server: container for one application

App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

Infrastructure

- Two Phase Commit
- Net / Threads
- APIs

Java EE

Java EE+
Tomcat / Jetty

Java EE

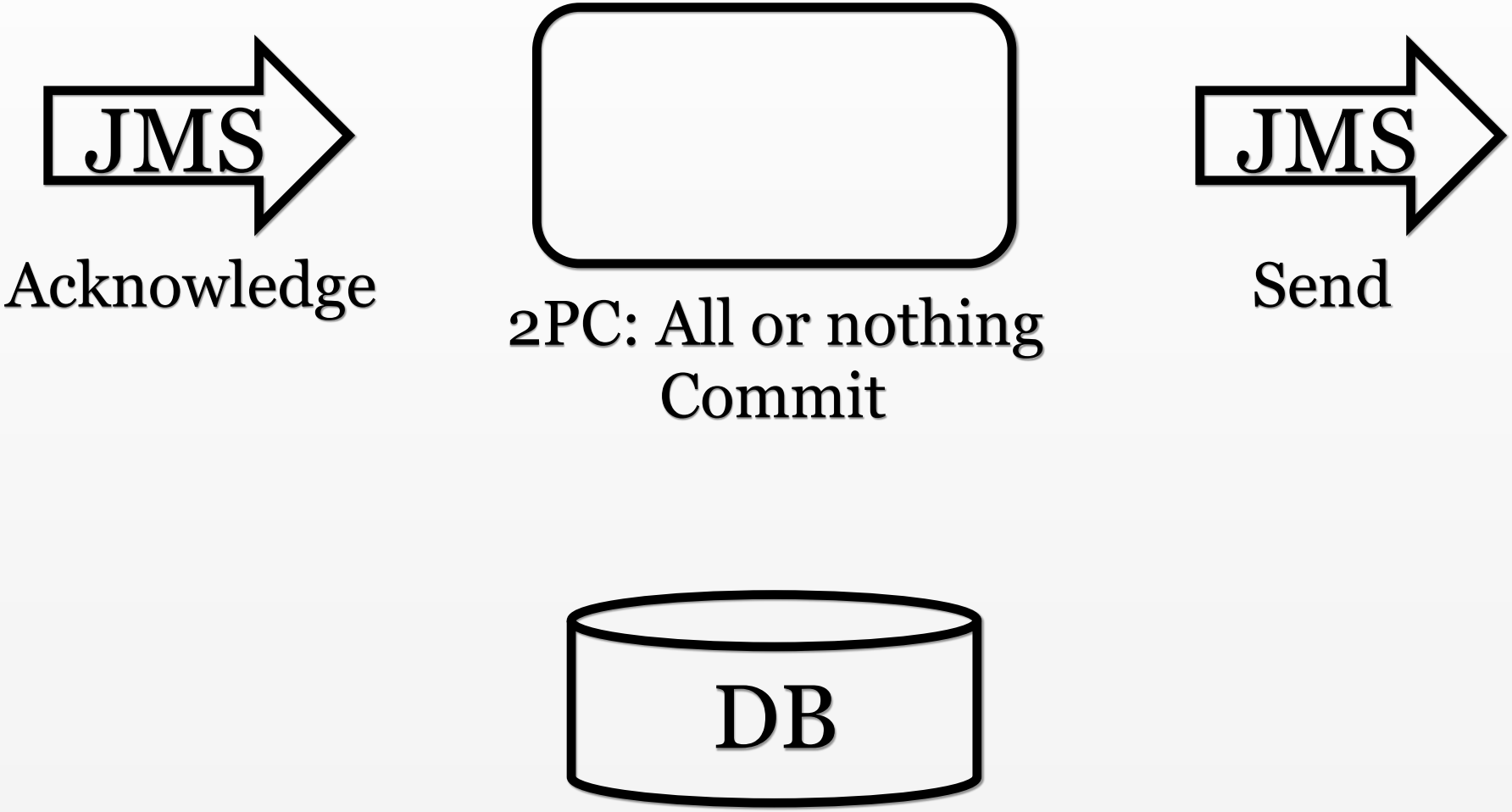
Infrastructure: Two Phase Commit

- Idea: Coordinate multiple transactional resources
- A talk in its own right
- 2 DBs

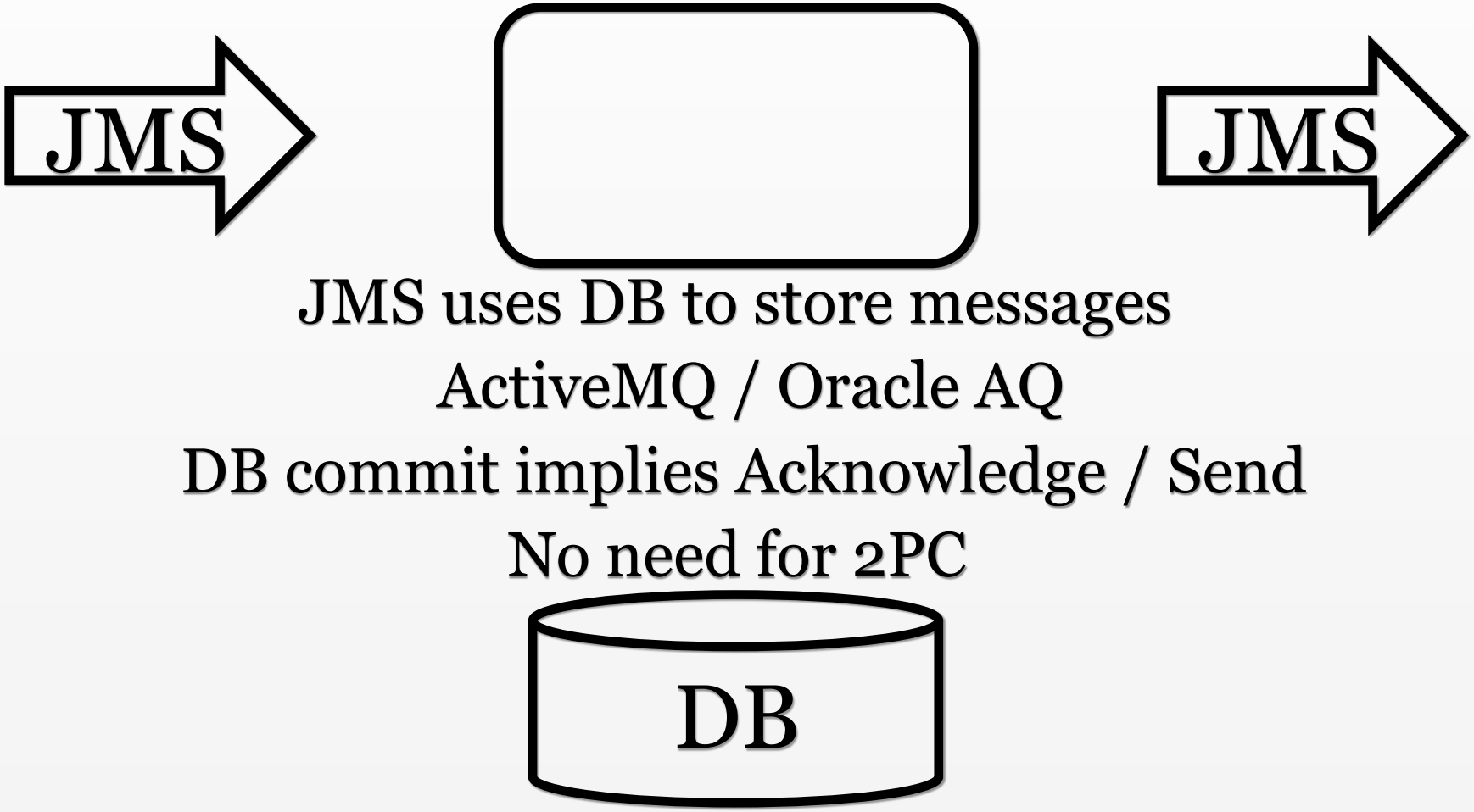
Technically valid

Consider a different architecture

2PC: JMS + DB



JMS Uses Same DB



JMS uses DB to store messages

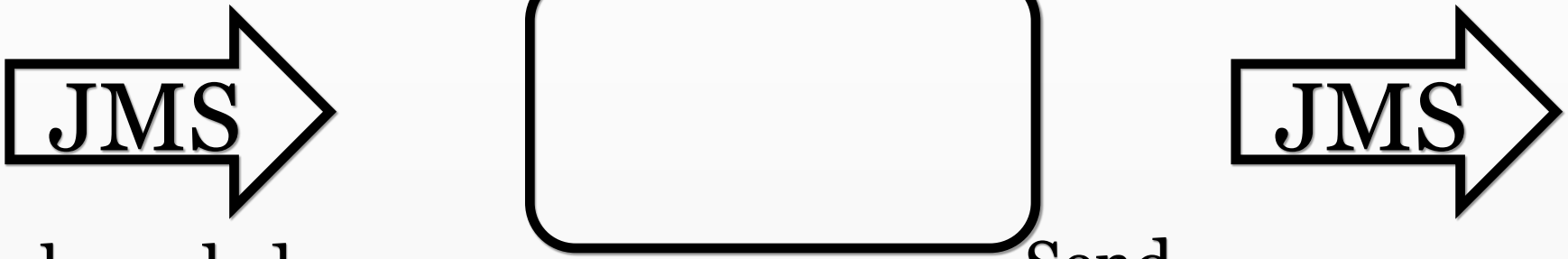
ActiveMQ / Oracle AQ

DB commit implies Acknowledge / Send

No need for 2PC

Synchronization: JMS + DB

Implemented in Spring



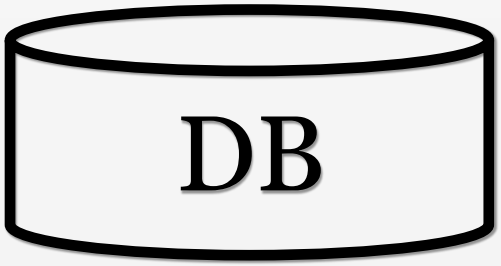
Acknowledge
if commit successful

Send
If commit successful

Commit DB

Need to handle duplicates if acknowledge fails

Might lose sent messages



Infrastructure: Two Phase Commit

- Slows down the good case in favor of the bad case
- No 100% - 2PC *can* fail
- Of limited use in distributed systems
- E.g. REST + 2PC?
- NoSQL + 2PC?
- Limits scalability

Infrastructure: Net / Threads

- Support i.e. for HTTP and thread pooling
- Connection pooling
- Can be done inside the application
- Embedded servers (Tomcat, Jetty)

Infrastructure: APIs

- EJB, CDI, JPA, JSF...
- Version tied to App Server version
- App depends on Application Server
- New APIs can't be used until new App Server in production
- Version conflicts might arise

Infrastructure: APIs

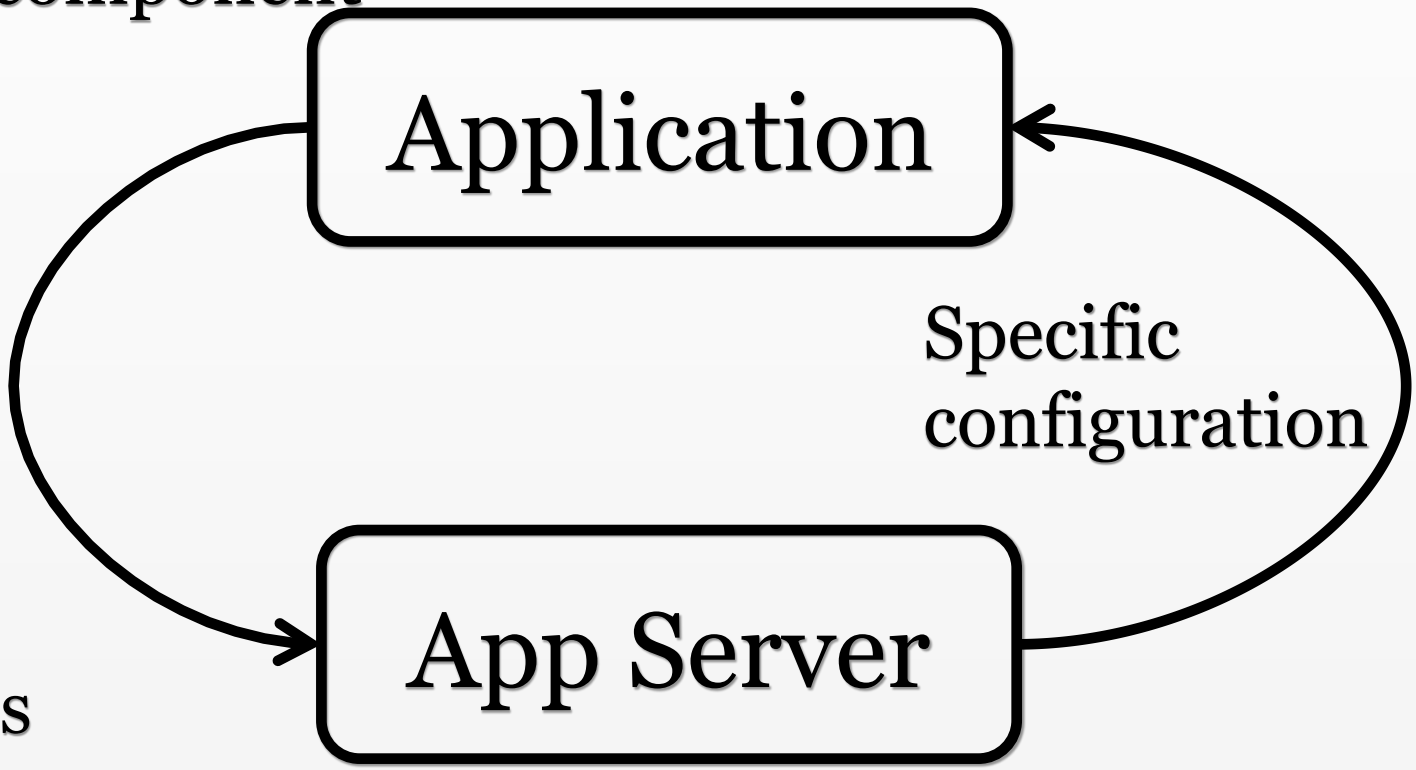
- Usually not every need covered
- ...so additional libraries are used
- App Server APIs can be replaced by libraries
- Makes application more portable

Infrastructure: Application independent?

- Each application has its own infrastructure
- E.g. database connections
- + specific configuration
- Might even add its own libraries to the AppServer
- Big no-no if the App Server should be application independent

Dependencies
App Server / Application

Cyclic dependency
i.e. one component



Libraries
Infrastructure

Application
Servers
are just
another part of
the Application

You Don‘t Agree?

- Can you deploy your application on a different server?
- On a different version of the same server?
- Without modifications to the server?
- Do you deploy other applications on the App Server?
- Could you?
- Is the application server or an installation script in your version control?

Application Server: Just One Kind of Infrastructure

- App Server focus on interactive (web) applications
- Other types of application:
- Batches
- Integration
- Map / Reduce
- App Servers are no universal infrastructure

App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

Deployment

- Deployment Format: WAR, EAR, JAR...
- No way to define dependencies outside Application
- i.e. App Server version, database etc
- Operations usually work with deb, RPM...
- Completely different tool chain
- Also: Usually Unix services to start applications

App Server...

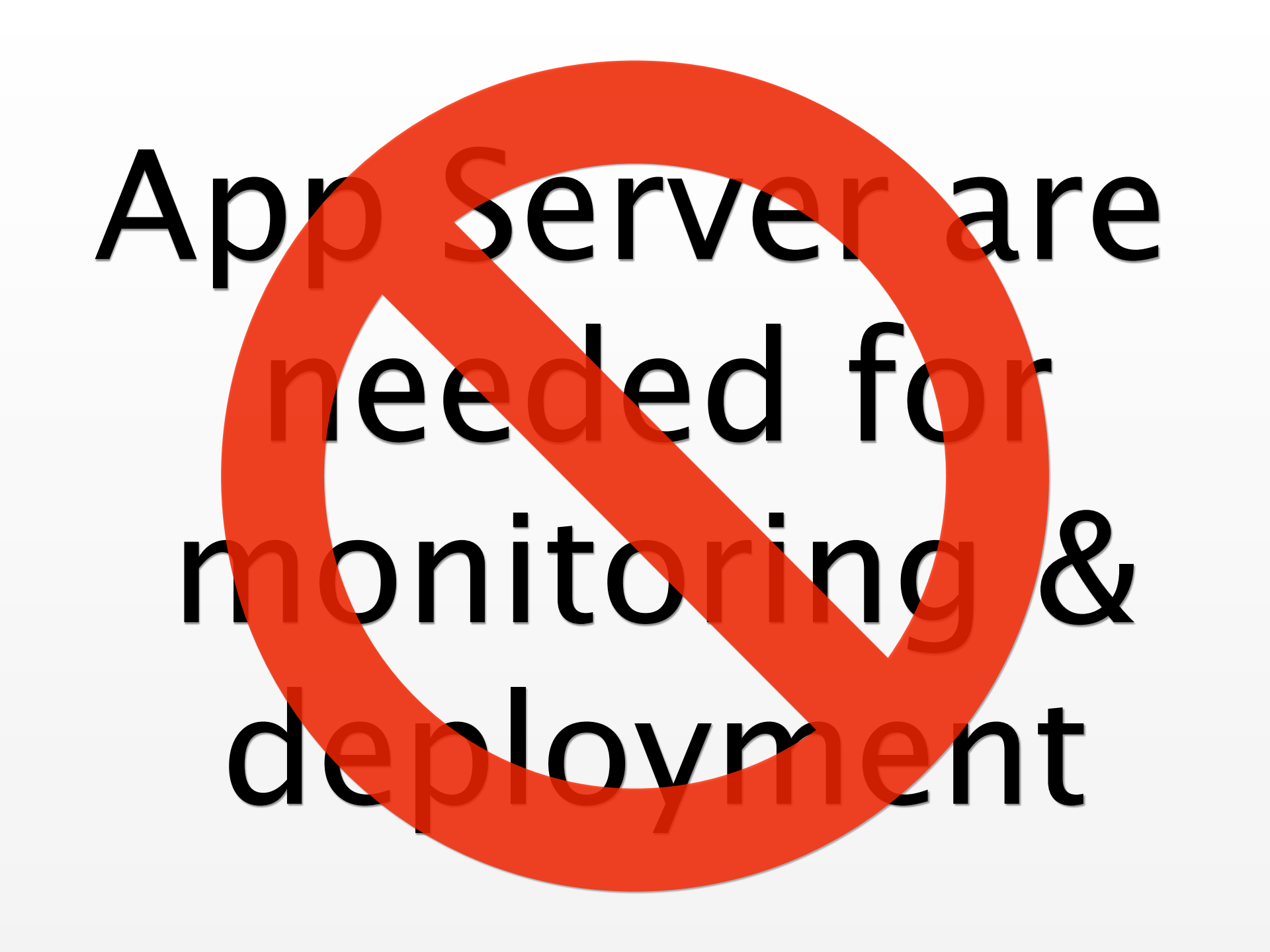
- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

App Server...

- ...container for multiple applications
- ... infrastructure
- ...deployment
- ...monitoring

Monitoring

- Provided by JMX
- Can be integrated in SMNP etc
- Again: different tool chain
- New tools arise
- Logs + Logstash / Kibana or Splunk
- REST based monitoring resources
- Scripts for monitoring


App Server are
needed for
monitoring &
deployment

App Server come with their own Ops Tool Set

App Server...

- ...container for multiple applications

  1 App per server ..or per cluster

- ... provide infrastructure

  App Server part of the application

- ...support deployment

  Deployment & monitoring OK
  but different tools

- ...support monitoring

The Price We Pay

Slower Turn Around

- Code
- Package a WAR
- Install it
- Have it unpacked
- Test

Why??

- Solutions: JRebel, Spring Loaded
- But: Why is it done at all??

App Server: Complex Deployment

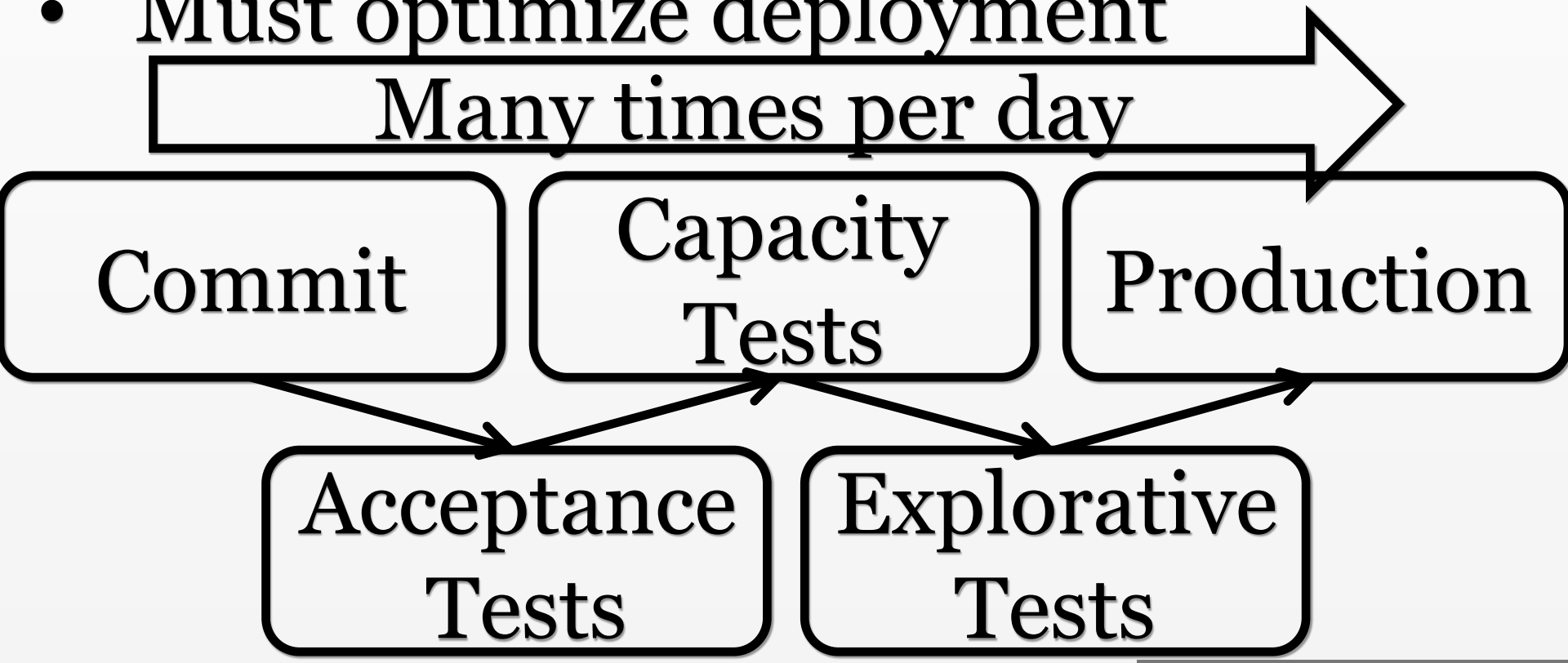
- Deployment: not just an application
- But also an Application Server
- App Server configuration more complex than Application configuration
- Look at automation scripts with Puppet / Chef etc

Cyclic Dependency Application – App Server

- Application Server and App Server must fit each other
- Configuration must be compatible
- For each developer and each testing stage
- Old configuration e.g. for bug fixes
- Hard to get right

Deployment Is Important

- Continuous Delivery means a lot more deployment
- Must optimize deployment



Continuous Delivery

- Applications deployed more frequently
- ...in many different stages
- Simple deployment even more important
- App Servers become bigger headache

Continuous Delivery increases demand for simple infrastructure.

Different Ops Mindset

- App Server administrator?
- Deployment, monitoring etc. have been solved already!
- Package manager
- Ops Monitoring
- Why not stick to general solutions?

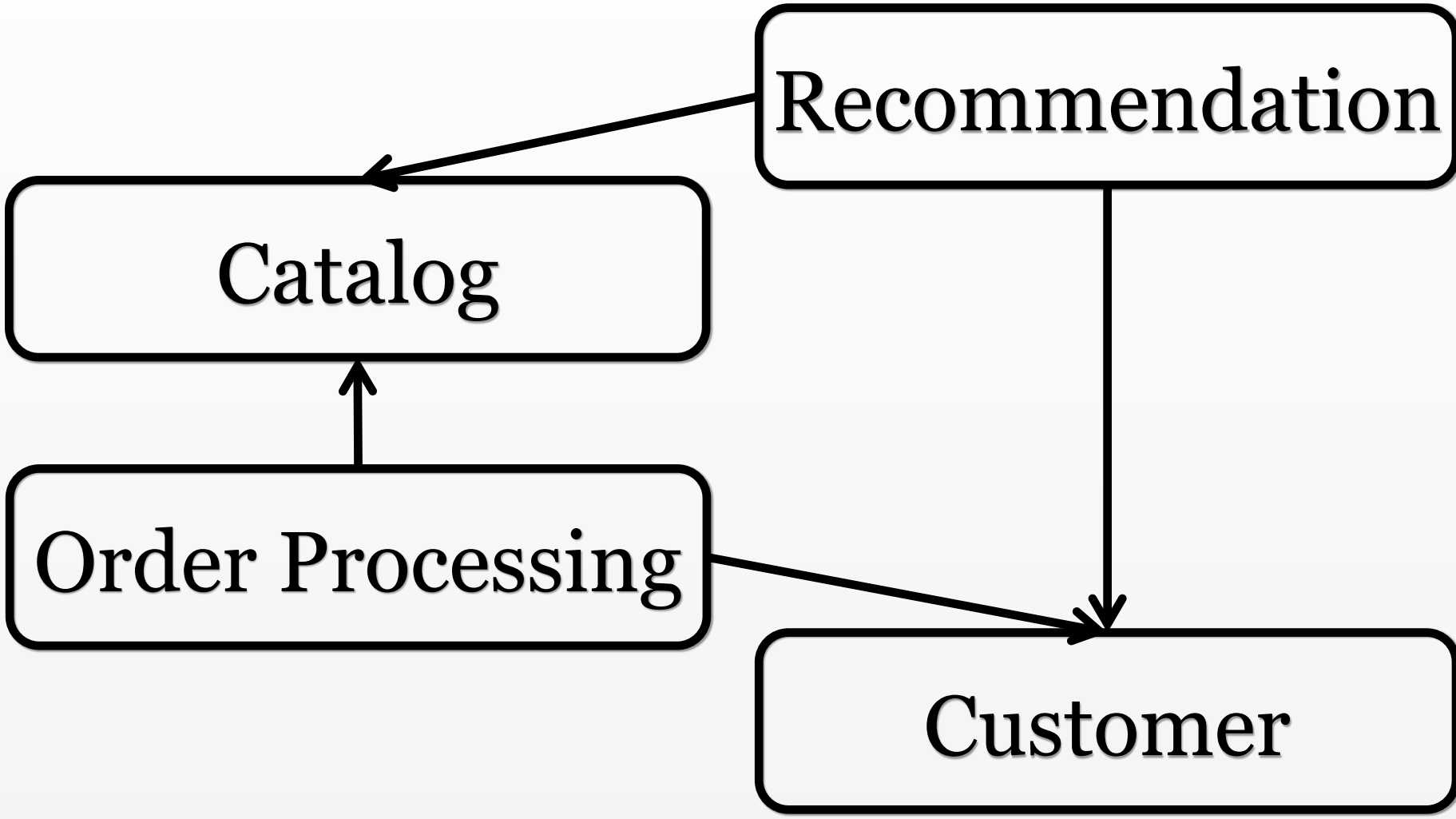
DevOps

- DevOps and Continuous Delivery: focus on “normal” tools and approaches
- Dev will see more than only Java Apps and App Servers
- Need different tools

Micro Services

- Build software composed of services
- Service has business meaning
- i.e. Order, Catalog etc
- Services (re)deployed independently
- ...instead of deployment monolith
- ...and communicate e.g. via REST

Micro Services: eCommerce

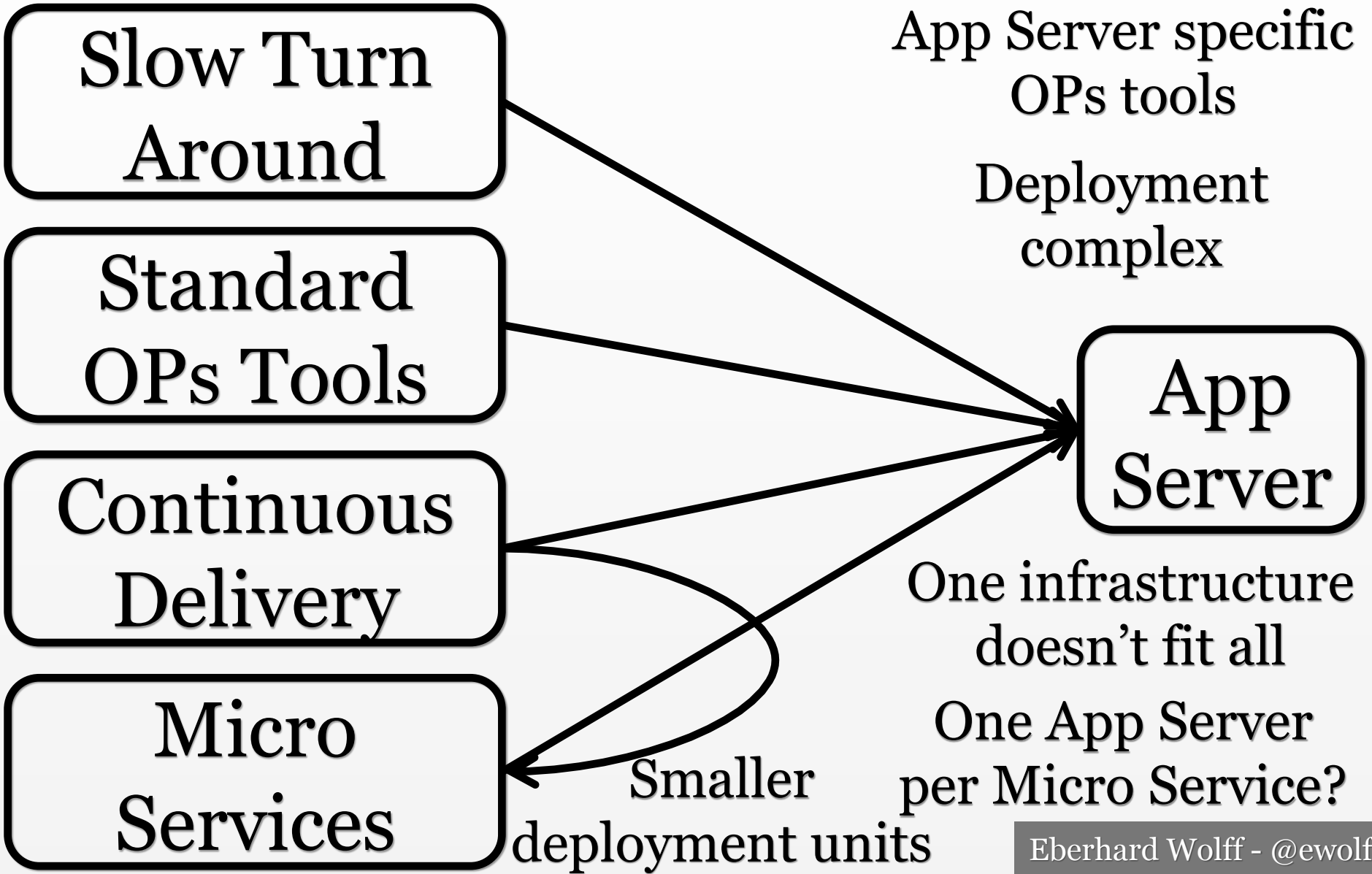


Install and configure App Server for each Microservice??

Micro Services

- Service might have different non-functional requirements
- So different infrastructure might be needed
- E.g. asynchronous applications
- Traditional Servlets
- Batches
- Map / reduce
- ....
- App Server just provide one kind of infrastructure

The Price We Pay



RIP Application Server!

Eberhard Wolff - @ewolff

What now??

The Re-Rise of the Applications

- Create a JAR files
- ...that contains a main class
- Custom infrastructure
- E.g. HTTP server
- Or Batch
- ...

Monitoring & Deployment

- Rely on standard Ops deployment and monitoring tools
- REST based monitoring URLs
- Evaluate log files

Application: Benefit

- Easier to Deploy: Just a JAR
- + command line
- + config file
- Debug & run in IDE
- Acceptance tests etc much easier
- Ensured: Infrastructure compatible with application

Technologies

- Spring Boot
- see my talk on Friday

- Dropwizard
- See Felix Braun‘s talk on Friday

Thank You!!