3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Wolkenschlösser

Architekturen für die Cloud

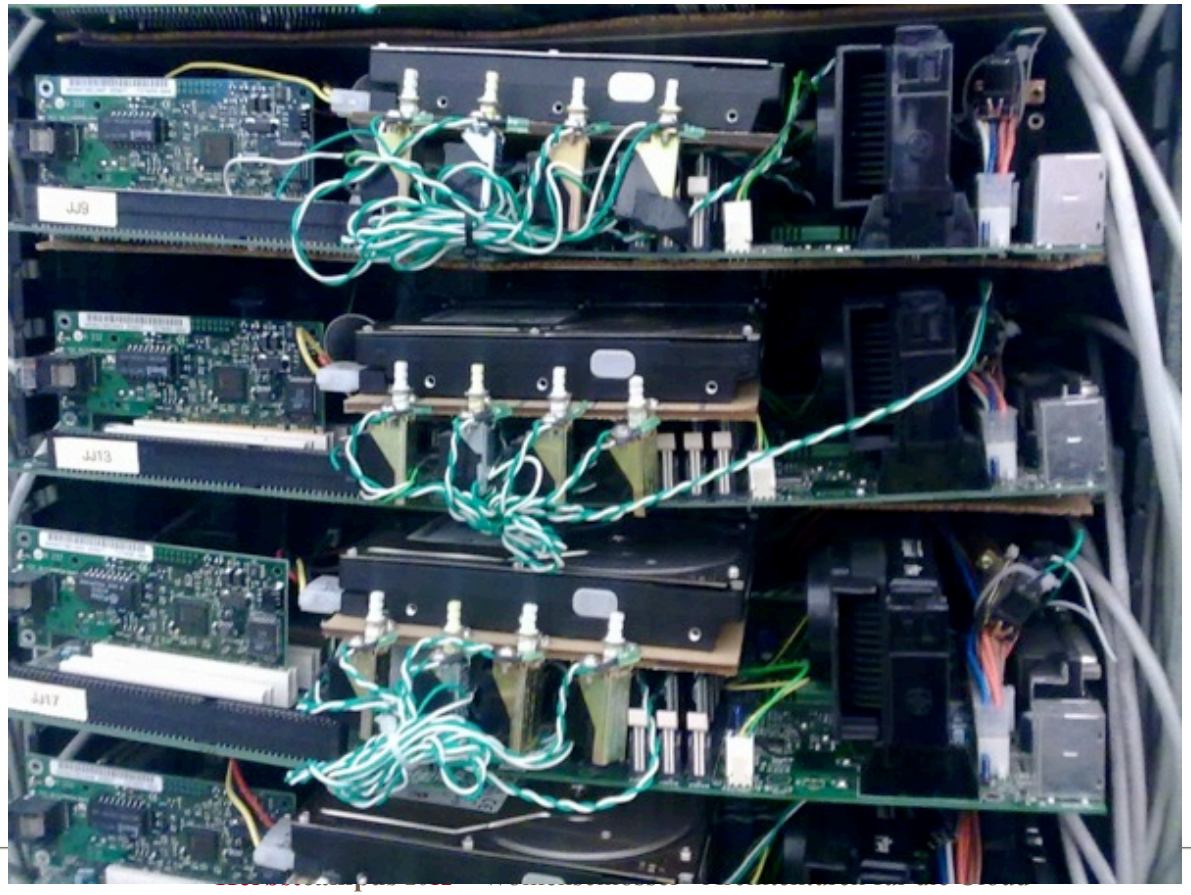
Eberhard Wolff

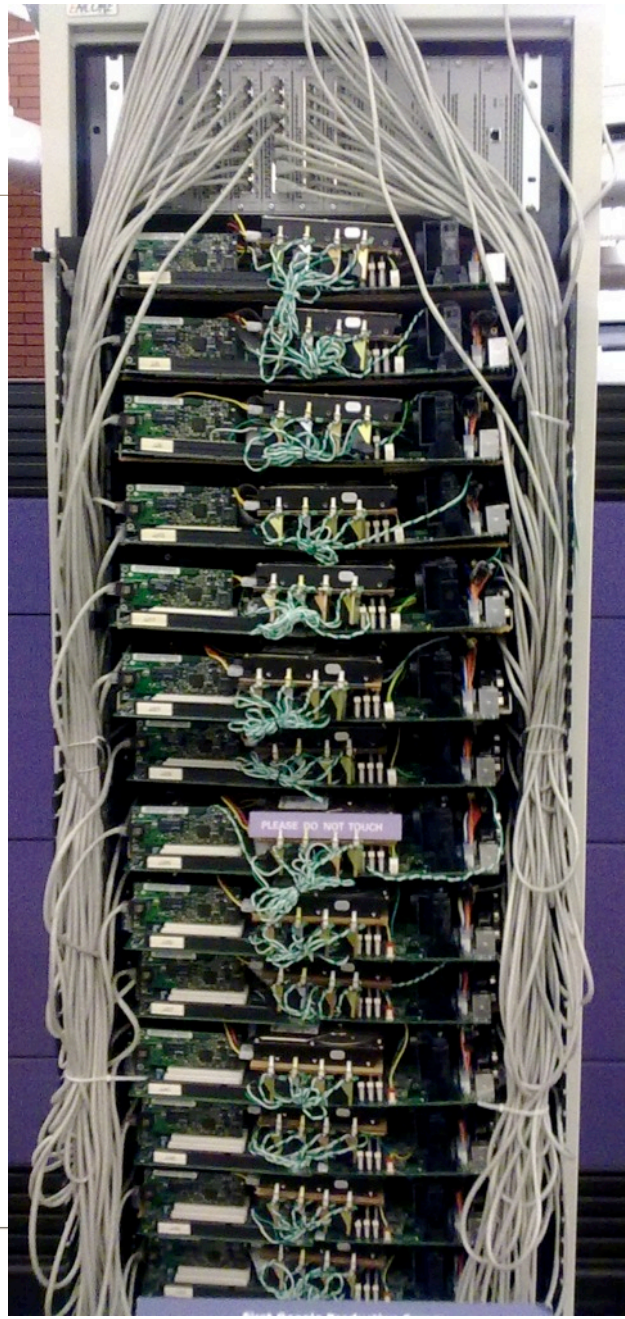
Architecture and Technology Manager, adesso AG

About me

- Eberhard Wolff
- Architecture & Technology Manager at adesso
- adesso is a leading IT consultancy in Germany
- Speaker
- Author (e.g. first German Spring book)
- Blog: http://ewolff.com
- Twitter: @ewolff
- http://slideshare.com/ewolff
- eberhard.wolff@adesso.de

How Is Cloud Different?





How Is Cloud Different?

⛔	Amazon Elastic Compute Cloud (N. Virginia)	Instance connectivity, latency and error rates. more »
✅	Amazon Elastic MapReduce (N. California)	Service is operating normally.
✅	Amazon Elastic MapReduce (N. Virginia)	Errors starting job flows. more »
✅	Amazon Flexible Payments Service	Service is operating normally.
✅	Amazon Mechanical Turk (Requester)	Service is operating normally.
✅	Amazon Mechanical Turk (Worker)	Service is operating normally.
✅	Amazon Relational Database Service (N. California)	Service is operating normally.
⚠	Amazon Relational Database Service (N. Virginia)	Database instance connectivity and latency issues more »

Life of our patients is at stake - I am desperately asking you to conta

Posted by: md76040303317
Posted on: Apr 22, 2011 11:20 PM

This question is **answered**. Helpful answers available: **2**. Correct answers available: **1**

Sorry, I could not get through in any other way

We are a monitoring company and are monitoring hundreds of cardiac patients at home
We were unable to see their ECG signals since 21st of April

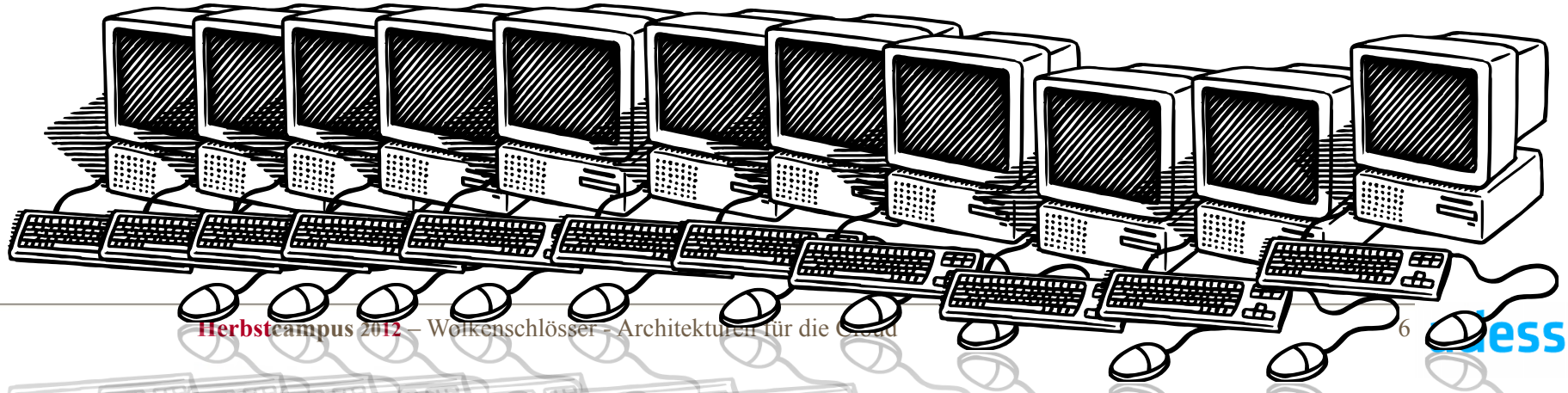
Could you please contact us?
Our account number is: 9252-9100-7360
Our servers IDs:

i-bb5c0fd0
i-8e6163e5
i-6589720f

Or please let me know how can I contact you more ditectly.
Thank you

How is Cloud Different?

- Can easily and cheaply add new resources
  - Prefer starting new instances over highly available instances
  - Prefer adding instances over using a more powerful instance
  - Might end up with lots of instances
- Prefer dealing with failure over providing a highly available network
- Lots of non powerful instances with unreliable network
- How can you end up with a reliable system then?



Chaos Monkey

- Test tool for Amazon cloud
- Part of the Simian Army project
- Originally developed by Netflix
  - Very large Amazon customer
  - Streaming TV provider
- Chaos Monkey randomly terminates systems in your Amazon Cloud

True High Availability

- Do not rely on the availability of your hardware!
- Therefore: Cloud architectures offer better availability

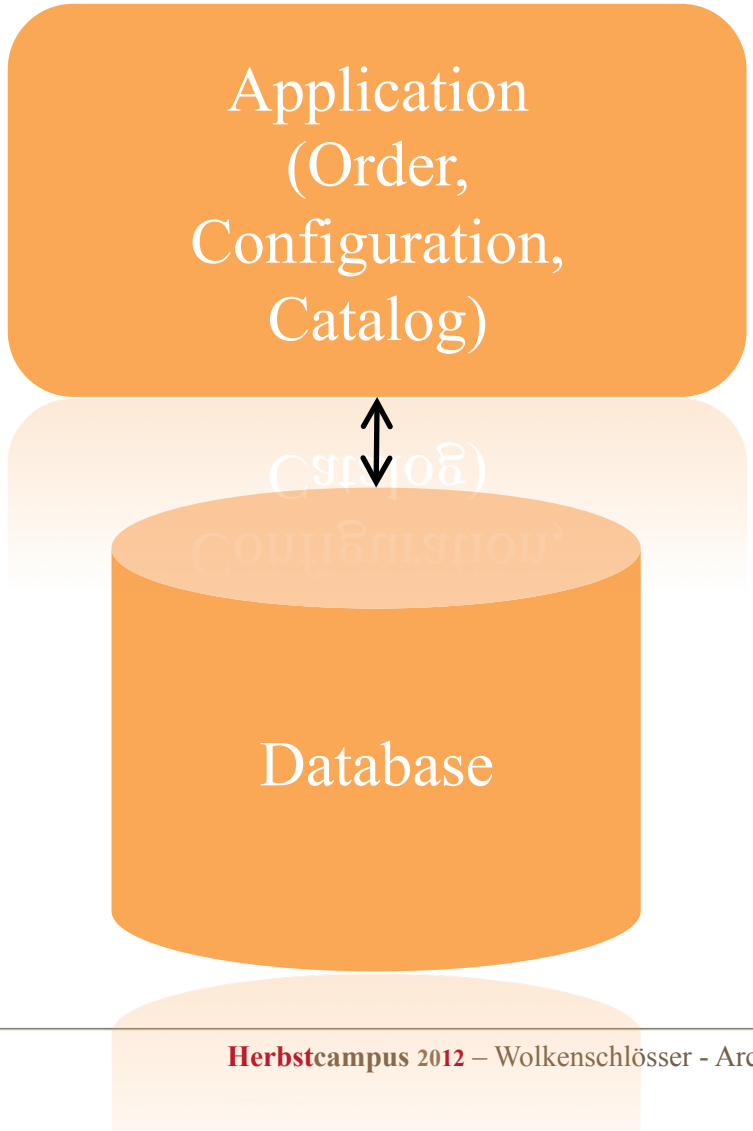
- Things to consider:
- How dependent are your non-cloud systems on individual servers?

Enter Spring Biking!

- The revolutionary web site to create customized bikes!
- We got a few million € Venture Capital

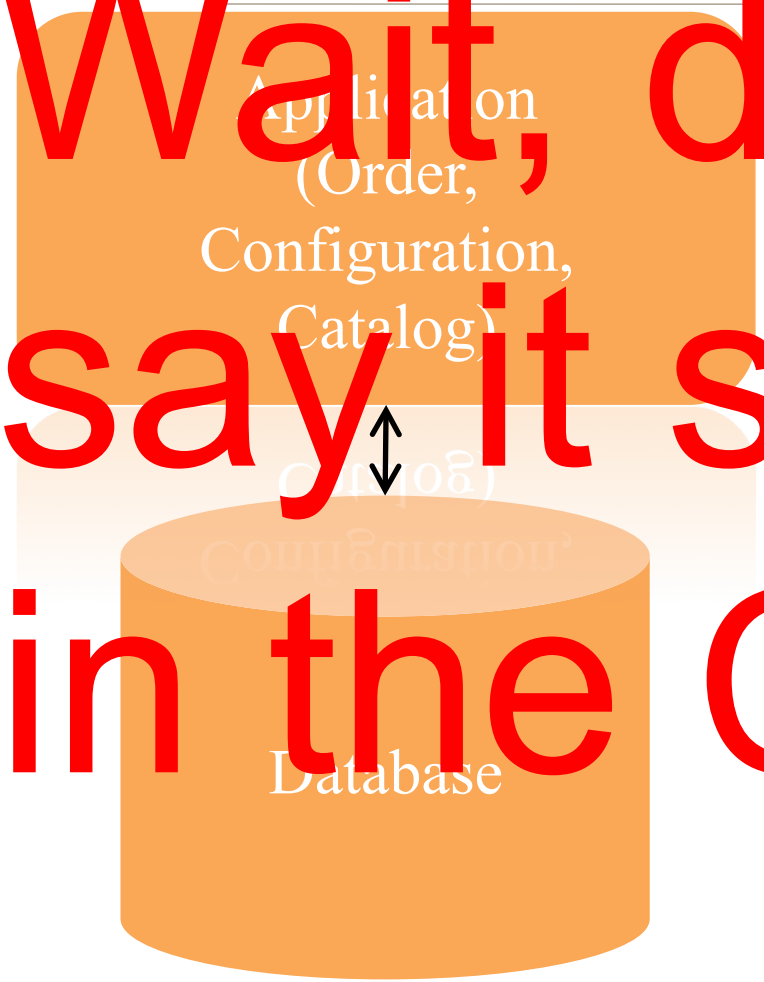
- We need...
  - Catalog of all Mountain Bike parts and bikes
  - System to configure custom Mountain Bikes
  - Order system
- Cloud good idea
  - No CapEx
  - Rapid elasticity -> easy to grow
- Focusing on German market

Spring Biking: Architecture



- Standard Enterprise Architecture
- Relational database

Spring Biking: Architecture

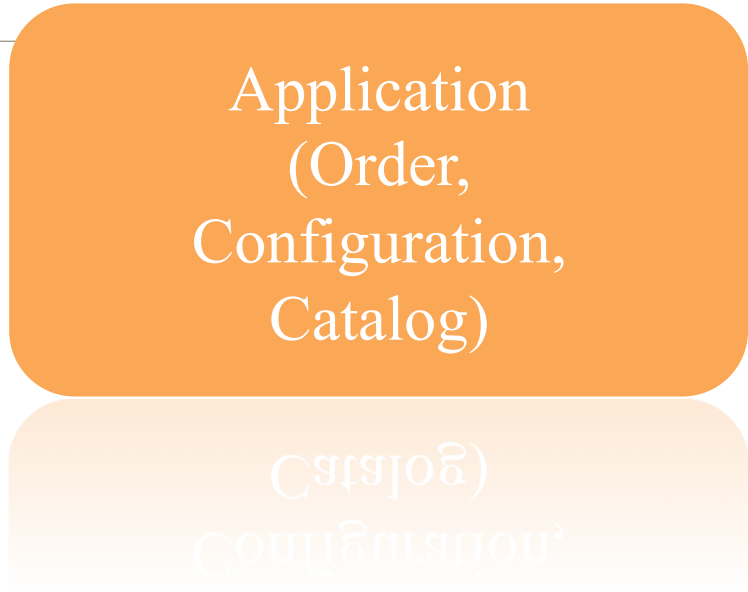


- Standard Enterprise Architecture
- Relational database

Wait, didn't you say it should run in the Cloud?

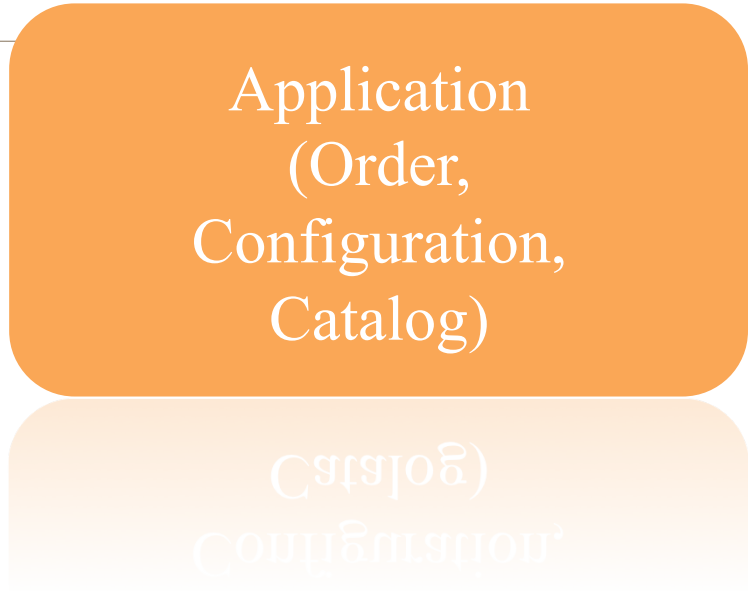
How Spring Biking Deals with Cloud Challenges

- No state on the web tier
  - i.e. no session
  - State stored in database
- No CAP issues on the web tier – no data



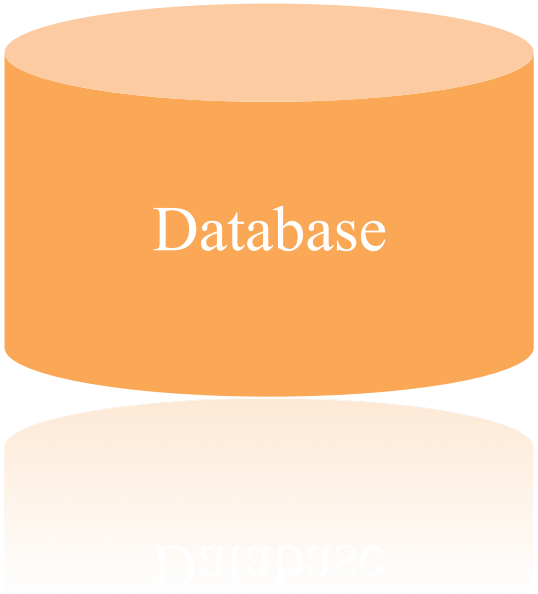
How Spring Biking Deals with Cloud Challenges

- Easy to automatically start new instances if load increases
- Every PaaS should deal with elastic scaling
- Example: Amazon Elastic Beanstalk
  - Takes a standard Java WAR
  - Deploys it
  - Add elastic scaling
- Could build something similar yourself with an IaaS
  - Automated deployment
  - Elastic scaling and load balancing available from Amazon IaaS offerings



How Spring Biking Deals with Cloud Challenges

- Relational database fine for now
  - Example: Amazon RDS (Relational Database Service)
  - MySQL and Oracle
  - MySQL: Multi data center replication
  - Can deal with failure of one data center



Benefits for the Development Process

- Trivial to get a new version out
- Easy to create a production like environment for test or staging
  - Take snapshot from production database
  - Set up new database with snapshot
  - Create a new environment with a different release of the software
  - Automated for production
  - Production-like sizing acceptable: You pay by the hour
- Some details might make it hard (e.g. 3rd party systems)

Benefits for the Development Process

- This can also be done using Virtualization / Private Clouds!
- Can be more important than cost reduction
- Business Agility is a major driver for (private) Cloud!
- …and the next step for virtualization.

Next step: Spring Biking Goes Global!

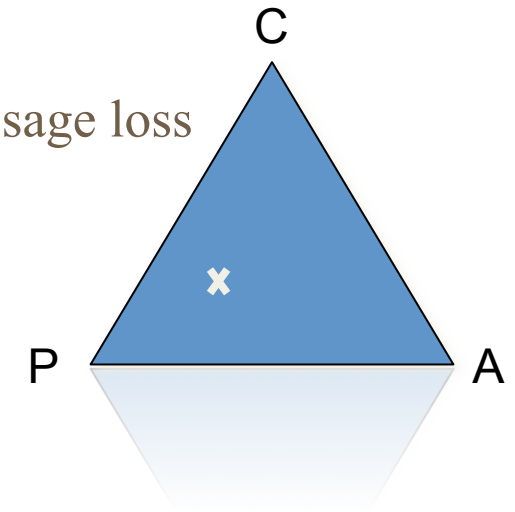
- Global demand for bikes is on all time high!
- We need to globalize the offering
- A central RDBMS for the global system is not acceptable
  - Amazon RDS offers databases for a Region (e.g. US-East, EU-West)
  - Need a different solution for a global system

- Just an example
- Traditional Enterprise architectures scales to a certain limit

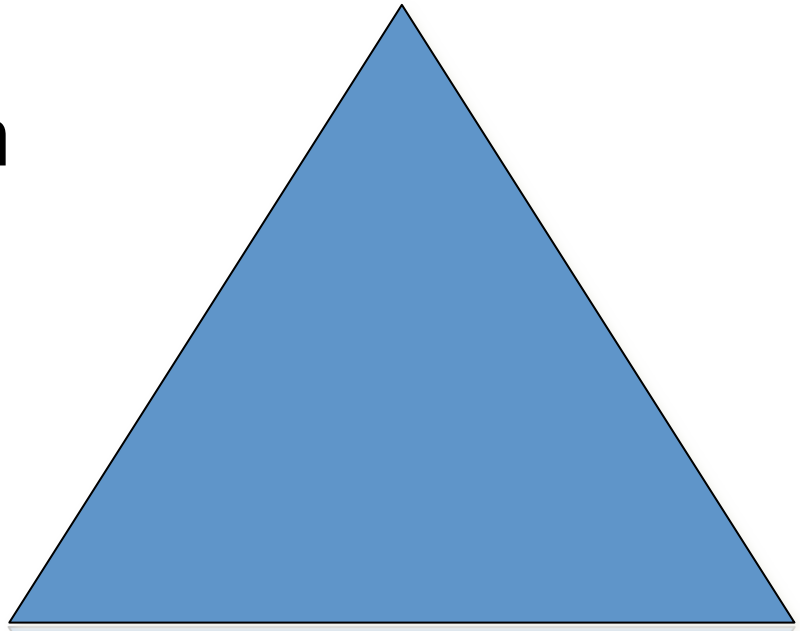
- We are not all going to build Twitter or Facebook

CAP Theorem

- Consistency
  - All nodes see the same data
- Availability
  - Node failure do not prevent survivors from operating
- Partition Tolerance
  - System continues to operate despite arbitrary message loss
- Can at max have two
- Or rather: If network fail – choose A or P.

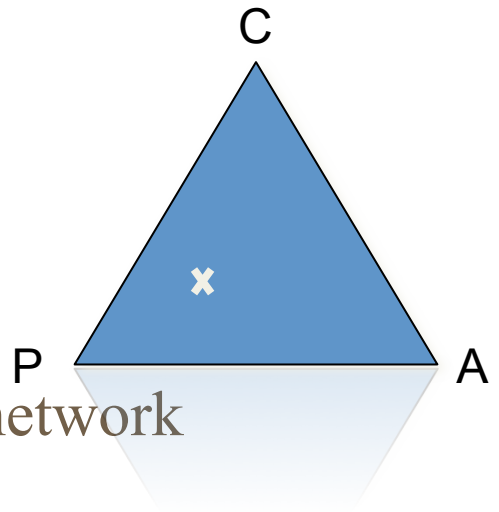


CAP Theorem


Consistency
RDBMS
Quorum
2 Phase
Commit
Partition
Tolerance
Replication
DNS
Availability

CAP Theorem in the Cloud

- Need A – Availability
  - A system that is not available is usually the worst thing
  - Shutting down nodes is no option
- Need P – Partition Tolerance
  - Network is not under your control
  - Lots of nodes -> partitioning even more likely
- No chance for C – Consistency
  - Because we can't
- CA used to be OK with a highly available network and a few nodes

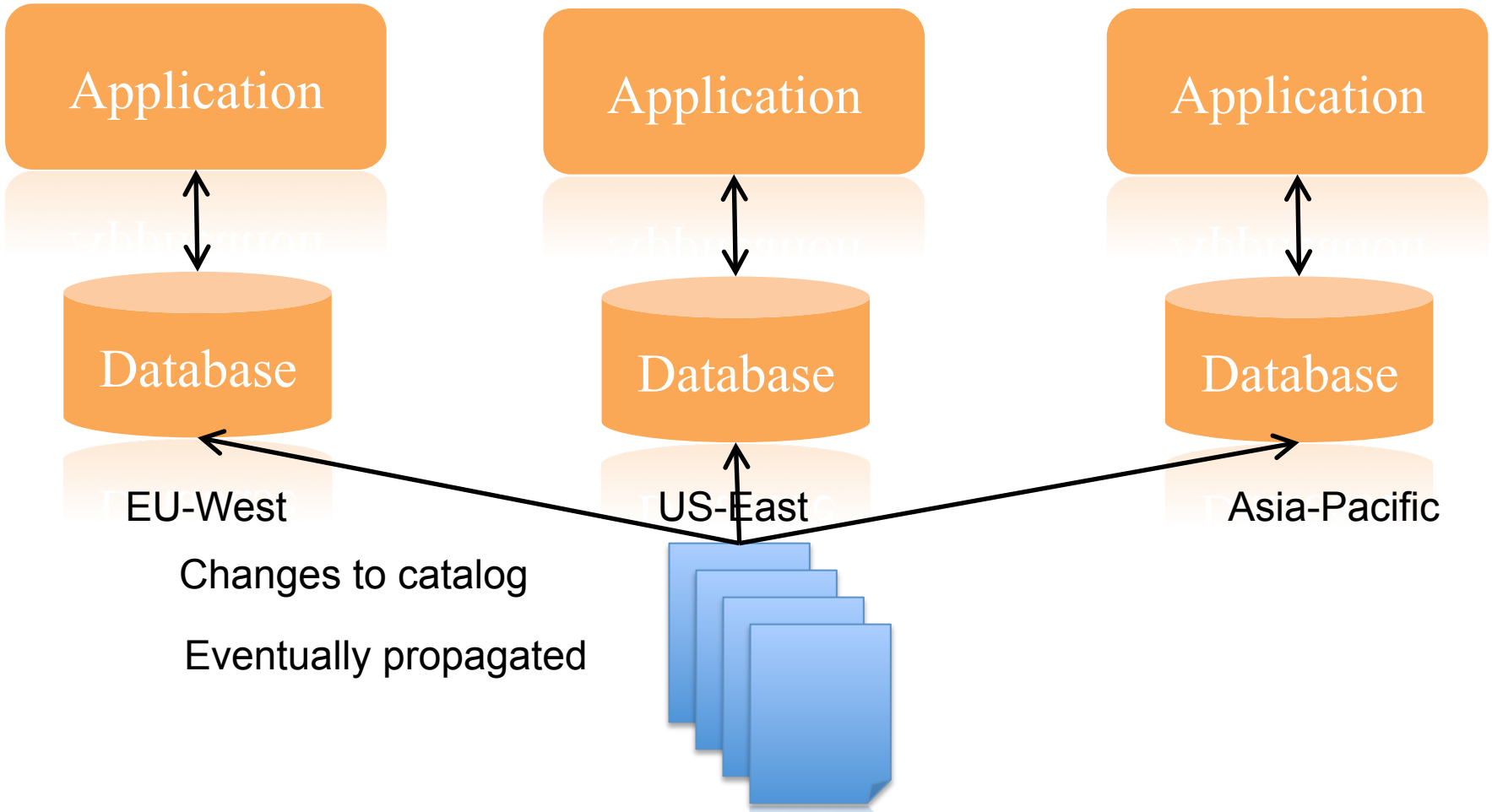


BASE

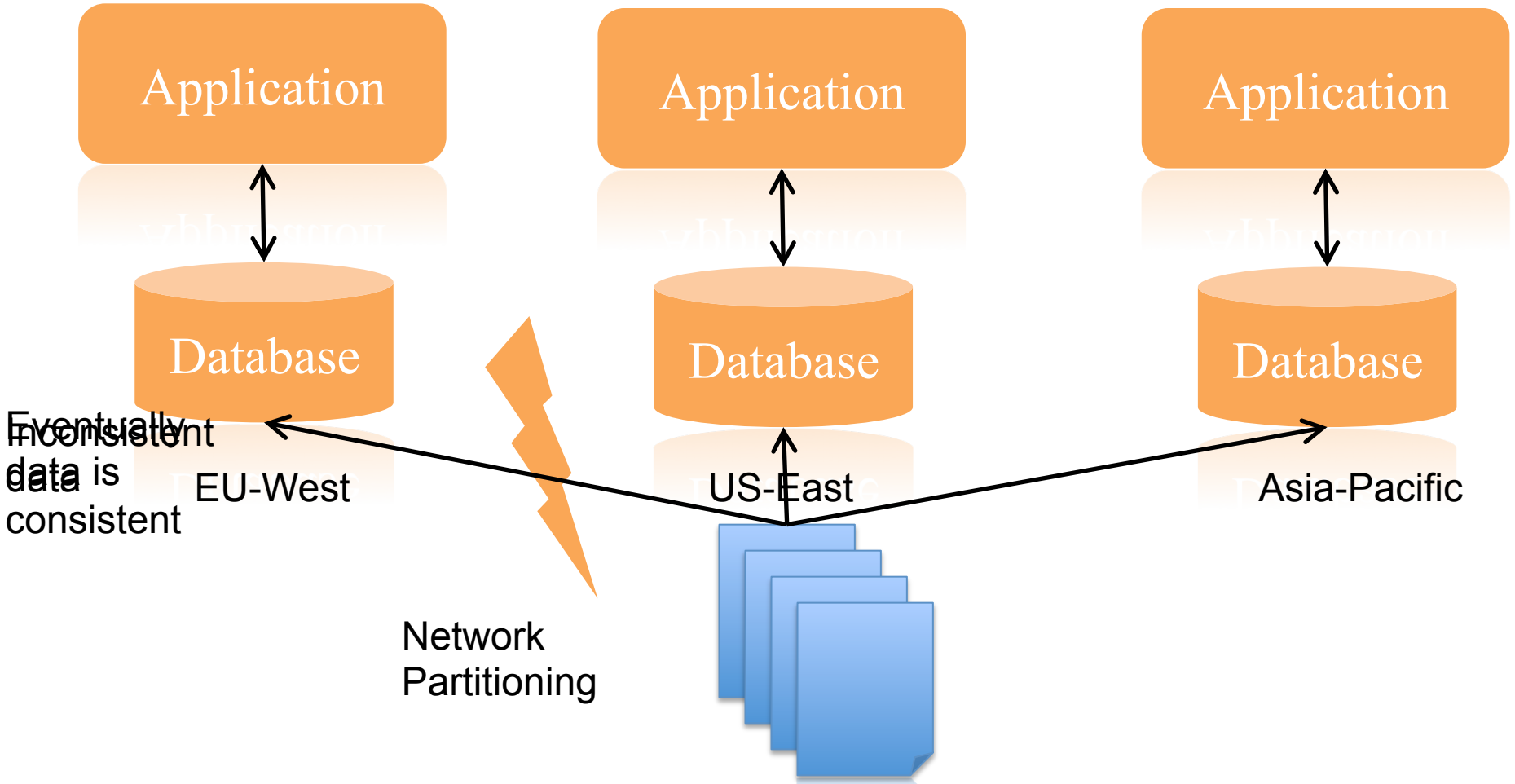
- Basically Available Soft state Eventually consistent
- I.e. trade consistency for availability
- Eventually consistent
  - If no updates are sent for a while all previous updates will eventually propagate through the system
  - Then all replicas are consistent
  - Can deal with network partitioning: Message will be transferred later
- All replicas are always available

- Pun concerning ACID…
- Not the same C, however!

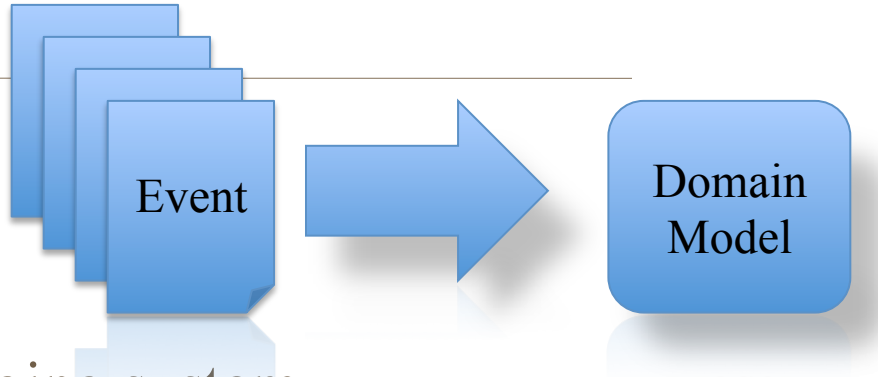
BASE in Spring Biking



Network Partitioning / Inconsistency



Implementing BASE Using Event Sourcing



- Do it yourself using a messaging system
  - JMS (ActiveMQ …)
  - RabbitMQ
  - Amazon Simple Queue Service (SQS)
  - Amazon Simple Notification Server (SNS)
  - Easy to duplicate state on nodes
  - Fail safe: Message will eventually be transferred
  - …and high latency is acceptable

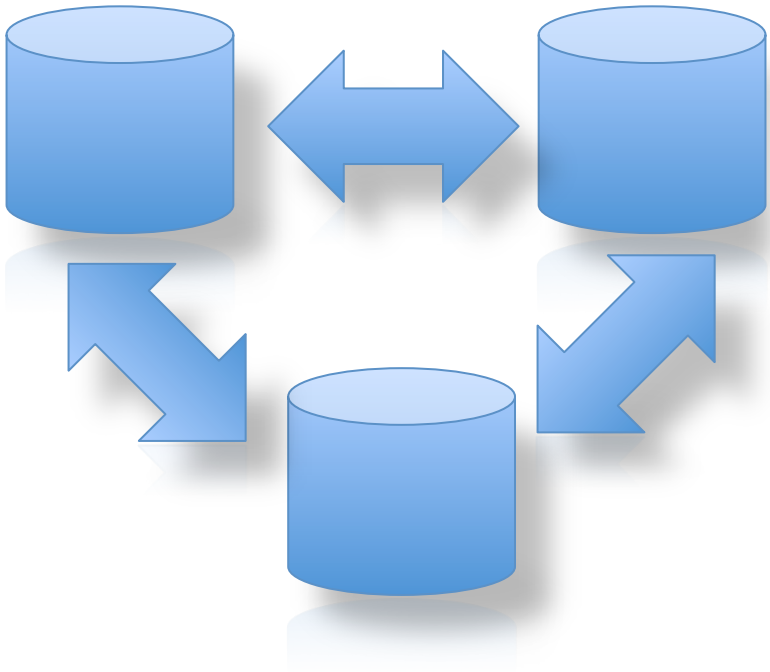
Implementing BASE Using Event Sourcing

- Other reason to use Event Sourcing
  - Capture all changes to an application state as a sequence of events
  - Originates in Domain Driven Design
  - Also used as a log of actions (to replay, reverse etc)
- Might end up with an Event-driven Architecture
  - Might add Complex Event Processing etc.

Implementing BASE Using NoSQL

- Some NoSQL databases include replication

- Example: MongoDB
  - Replication between nodes
  - Master-slave replication
  - Master automatically elected
  - Easy to set up
  - All nodes have the same data
  - Sharding also possible

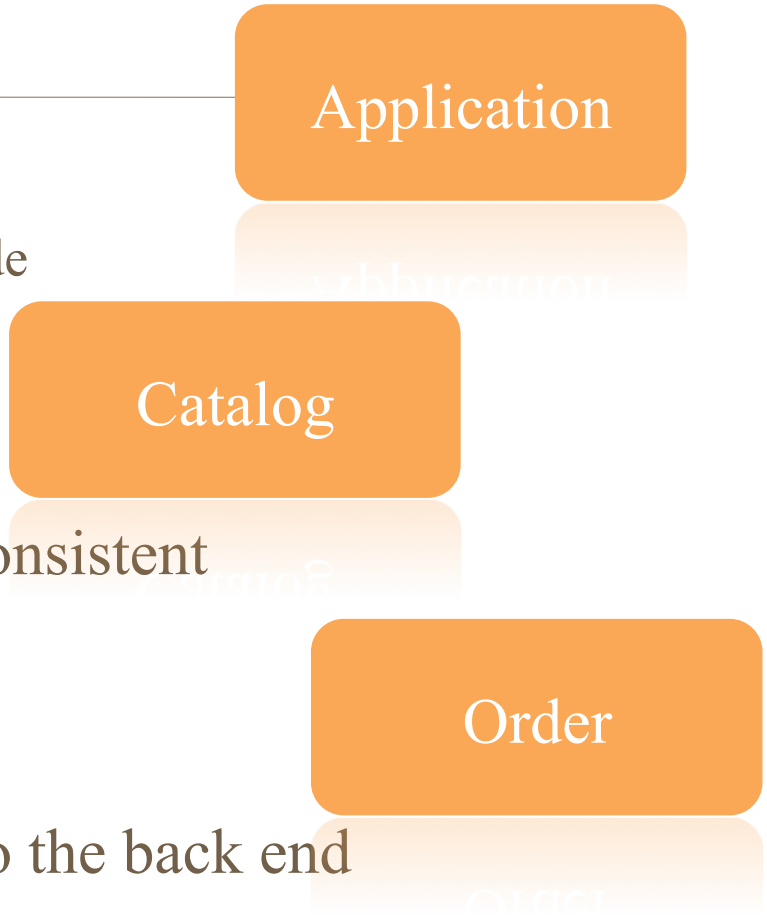


More Sophisticated

- Writes must be acknowledge by N nodes
- …or nodes in each data center
- Data is read from master
- …or also slaves are OK

- Replication done automatically
- Clustering built in
- Tuneable CAP

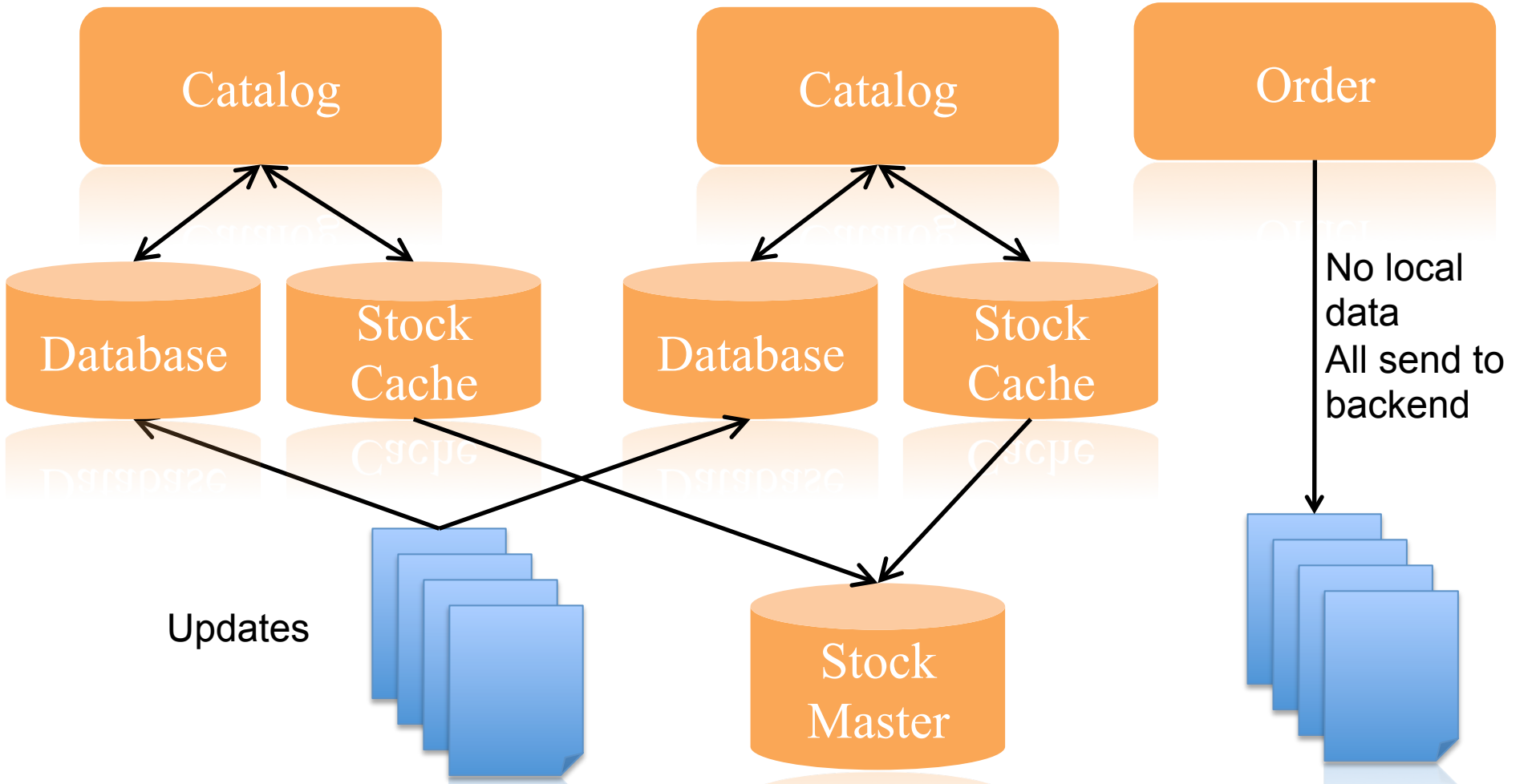
Different Parts Require Different Architecture



- So far: Catalog
  - Data must be available on each node
  - Slight inconsistencies are OK
  - i.e. new item added to catalog
- Stock information should be consistent
  - So customers are not disappointed
  - Might use caching-like structure
- Orders are immediately send to the back end
  - No local storage at all
- A lot more catalog browsing than ordering

More load on catalog ->
More instances

Less load on order ->
Less instances



Applications vs. Services

- Applications are decomposed into services
- Benefits
  - Unit of failures can be aligned to services
  - And: Service failure can be dealt with
  - Can scale services independently
  - Can use infrastructure specifically designed for the servers
- Remember the First Law of Distributed Objects: Don’t Distribute Your Objects!
- E.g. provide HTML pages
- Fits DevOps approach: Align operations to services

Application vs. Services

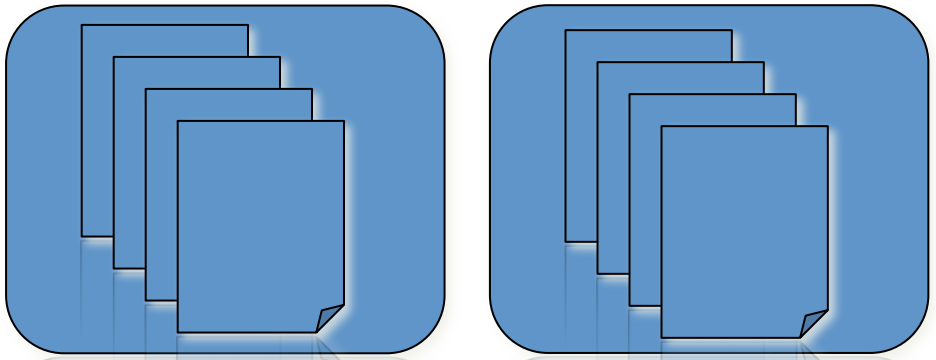
- Very different from centralized web server, db server etc
- Instead: to each service its own environment

- Very different from monolithic EAR style deployment
- Smaller services and deployment models

- So: Enterprise Java will need to adjust

Handling Log Files

- Business requirements
  - Need to measure hits on web pages
  - Need to measure hits for individual products etc.
- Sounds like a batch
  - File in, statistics out



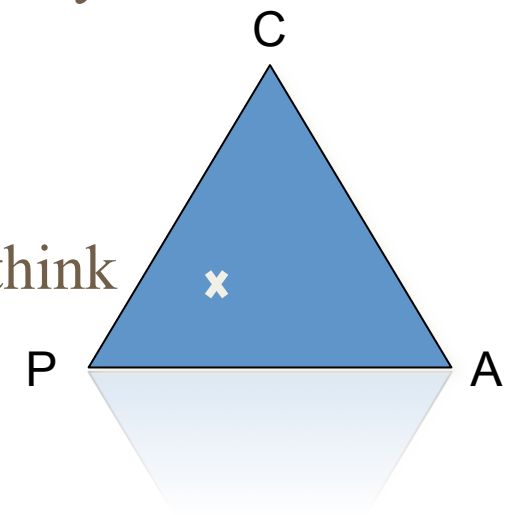
- But: Data is globally distributed
- Lots of data i.e. cannot be collected at a central place
- Data should stay where it is
- Some nodes might be offline or not available
- Prefer incomplete answer over no answer at all

More Than CAP

- CAP Theorem again
- Consistency, Availability, Network Partitioning
- You can only have two

- But: We want Availability
- …and a flexible trade off between Consistency and Network Partitioning
- Like Casssandra

- I.e. CAP theorem is not the proper way to think about this



Harvest and Yield

- Yield: Probability of completing a request
- Harvest: Fraction of data represented in the result

- Harvest and Yield vs. CAP
- Yield = 100% -> Availability
- Harvest = 100% -> Consistency

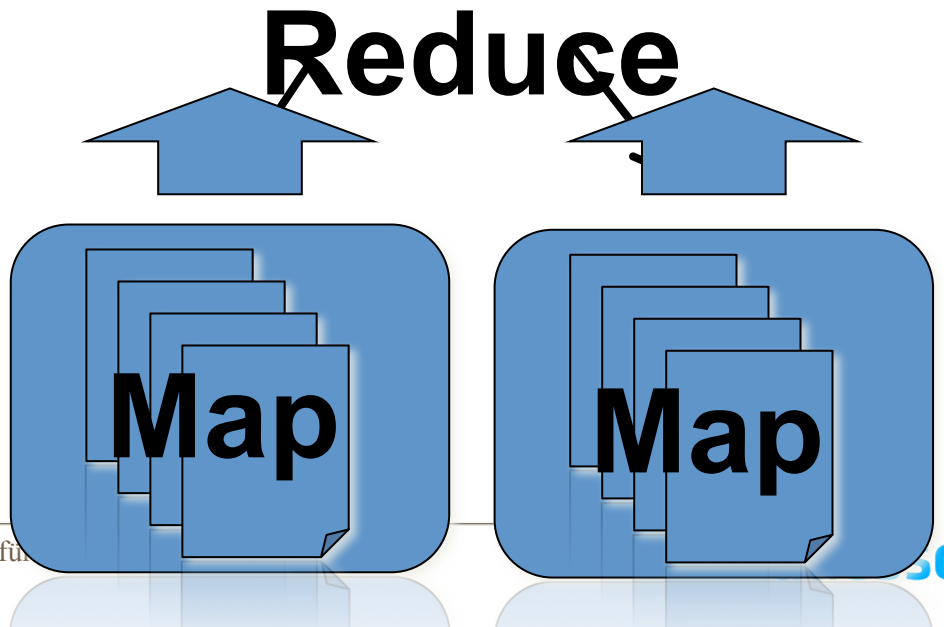
- Can also be used to execute some logic on all data
- …and wait until enough harvest is there to answer a query

- So: Send out a query to all log files
- …and collect the results

Map / Reduce

- Map: Apply a function to all data
  - Emit (item name, 1) for each log file line
- Master sorts by item name
- Reduce: Add all (item name, 1) to the total score

- Map can be done on any node
- Master collects data



Another Case Study

- Financials

- Build a Highly Available, High Throughput System, Low Latency System on Standard Hardware!
- Just like Google and Amazon

- Driver: Standard infrastructure – cheap and stable
- Driver: Even more availability, throughput, scalability and lower latency

- You will need to consider CAP, BASE, Harvest & Yield etc.
- Very likely with virtualization / Private Cloud

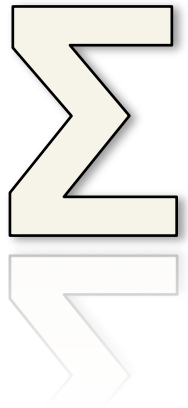
Another Case Study

- Random Project

- Make deployment easier!
- Make it easy to create test environment!
- Driver: Business Agility and Developer Productivity
- Will need to use automated installation + IaaS or PaaS
- Might be in a Public or Private Cloud

- Example: adesso Mobile Solutions

Conclusion



- Better and cheaper high availability
  - by welcoming hardware failure
- Better and cheapter scalability
  - by horizontal scaling

- Current PaaS run Enterprise applications unchanged

- Keep in mind:
- CAP: Consistency, Availability, Partition Tolerance
- You will need to relax C to get A and P

- Architecture will prefer small services
- …as units of failure

Vielen Dank!

Eberhard Wolff

Architecture and Technology Manager, adesso AG