Cloud Computing and Microservices

Sommerakademie in Leysin

AG 2 - Effizientes Rechnen

Fabian Wildgrube

Ludwig-Maximilians-Universität München

22. August 2016

Introduction: “The Cloud”

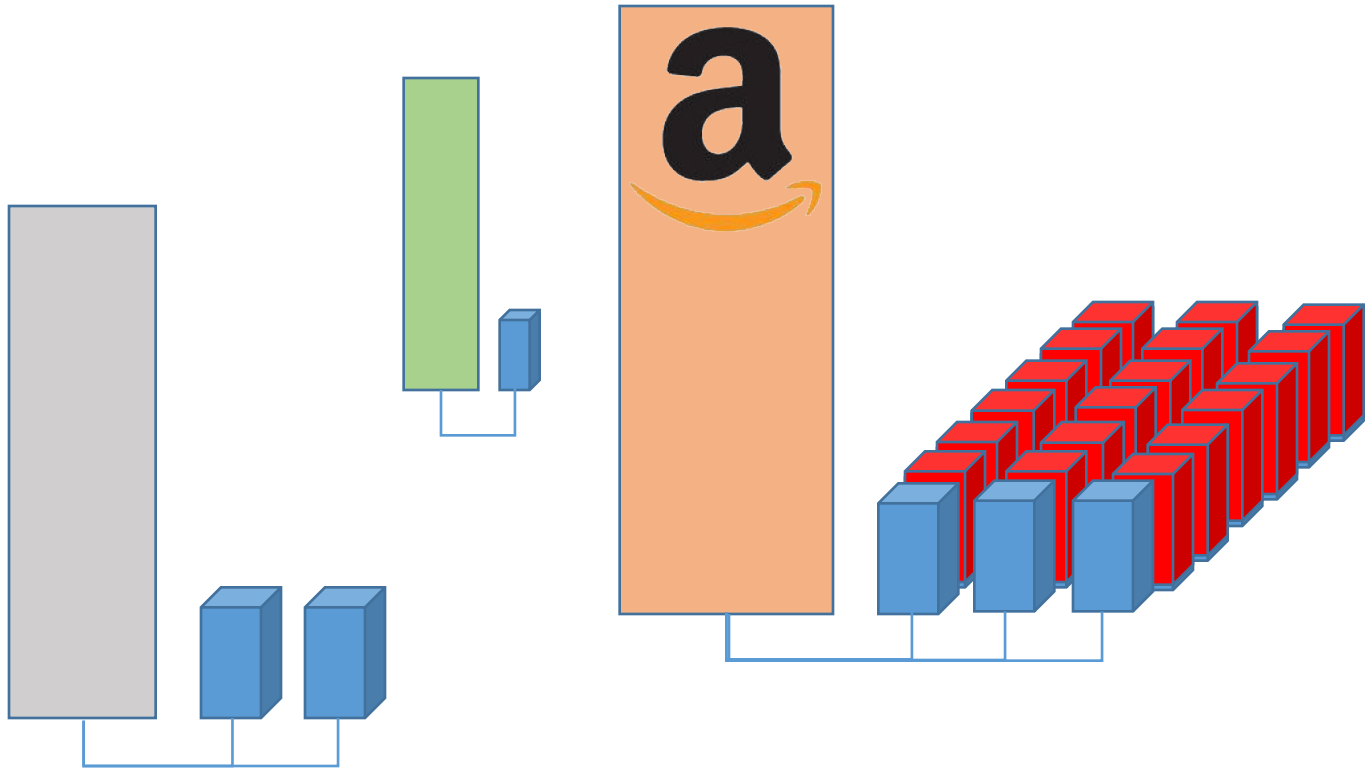
Fluffy cotton balls or what?

Cloud Computing – a short definition

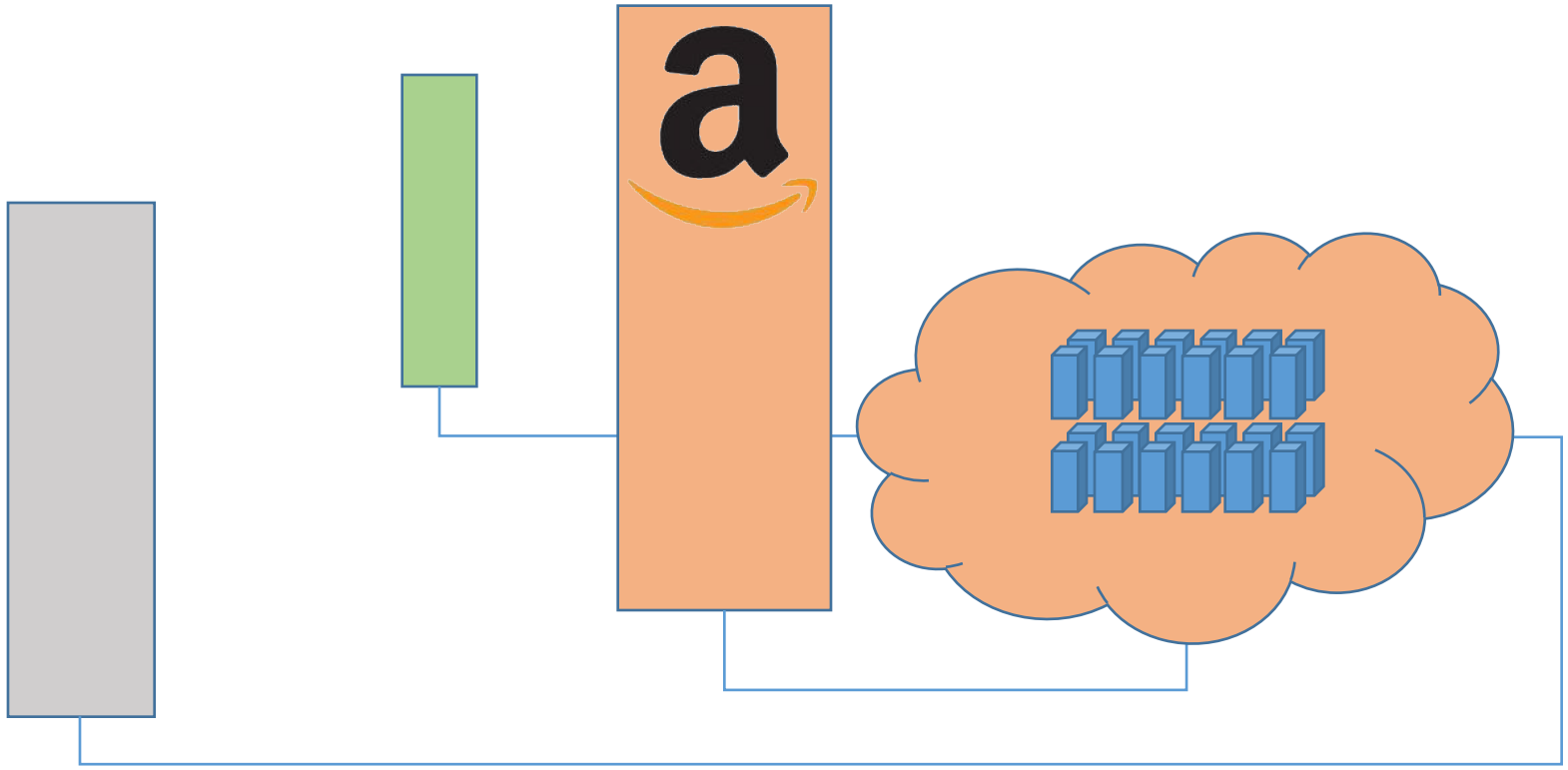
Cloud Computing describes the idea of accessing off-premise IT-infrastructure, platforms and ready-to-use applications via the network rather then owning and managing these on-premise.


amazon
web services

The basic idea

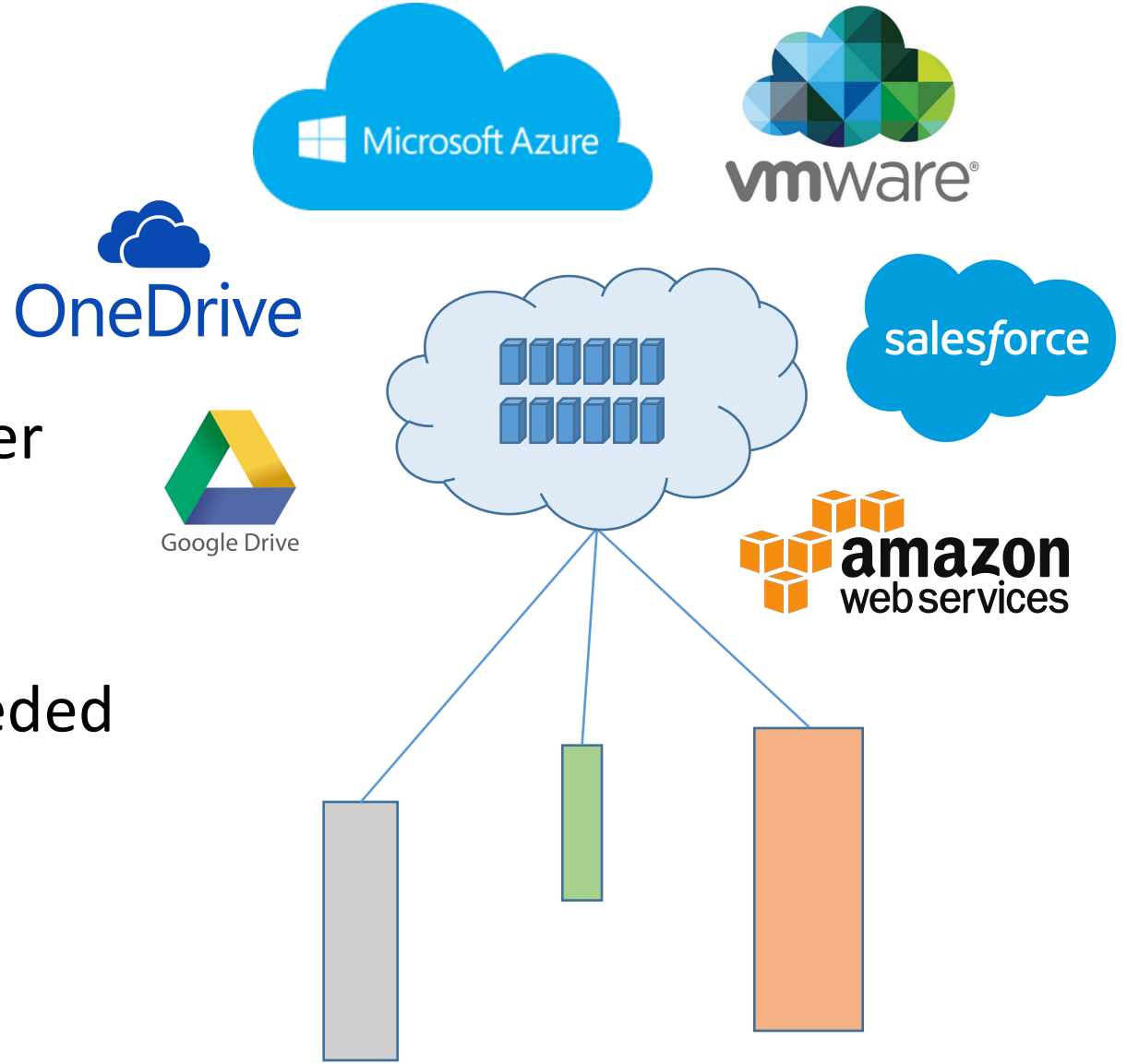


The basic idea



The Cloud =

- No on-premise IT-Infrastructure
- Several people/companies share the resources offered by one central provider
- Flexible, demand-based adjustments (Peak-Coverage)
- Subscription based – only as long as needed



› “Outsourcing“ to the Cloud

Lets get technical

How does the Cloud actually work?

The different Levels of Cloud Computing Solutions

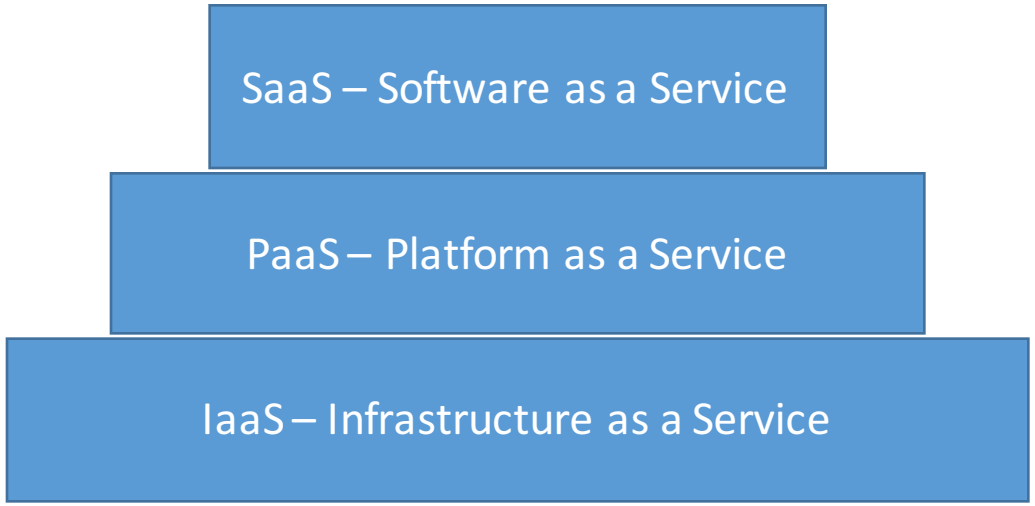
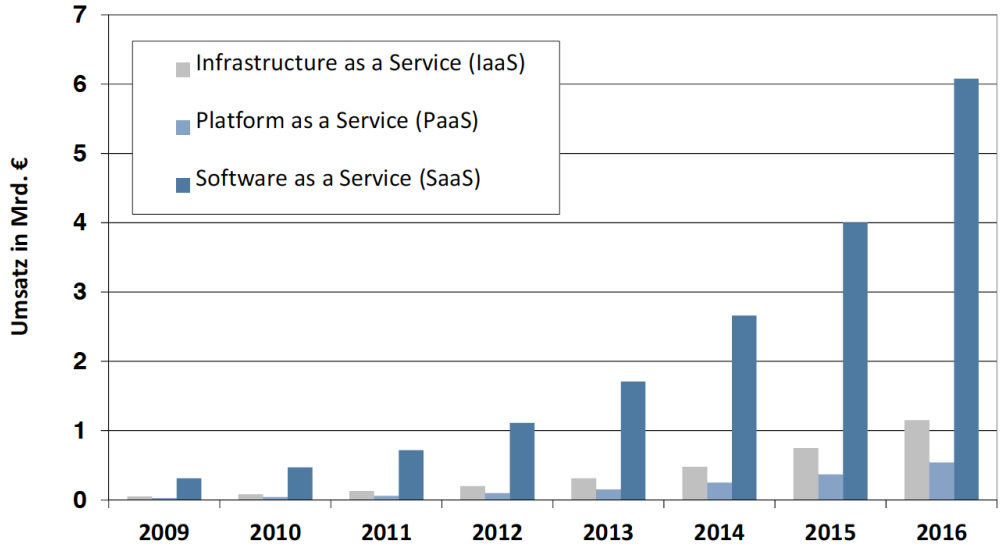


Abbildung 2: Umsatz und Wachstum bei Cloud Services



Quelle: Darstellung nach eco/Arthur D. Little 2013, 16

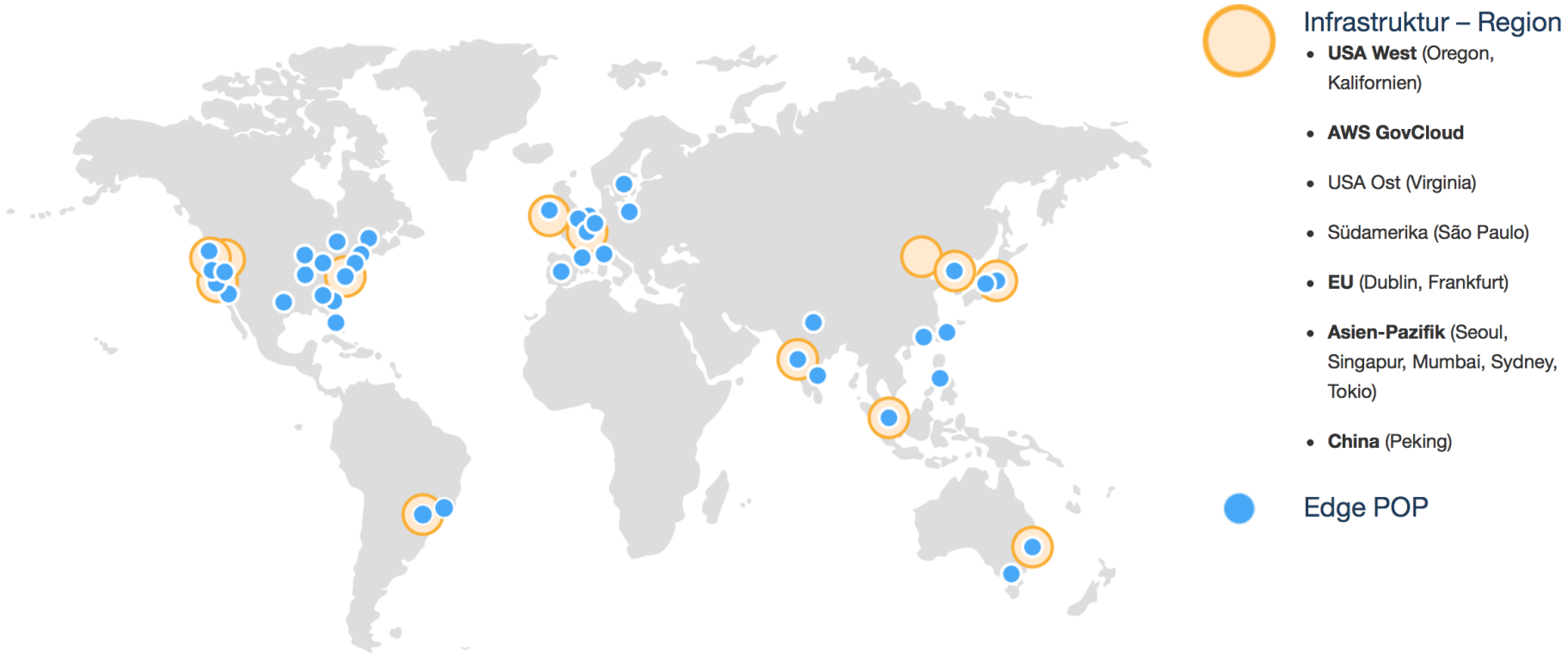
Looking into a Cloud Data Center

- Much wow, many servers...

- ...About 50,000 - 80,000
- Power capacity between 25 and 30 megawatts.

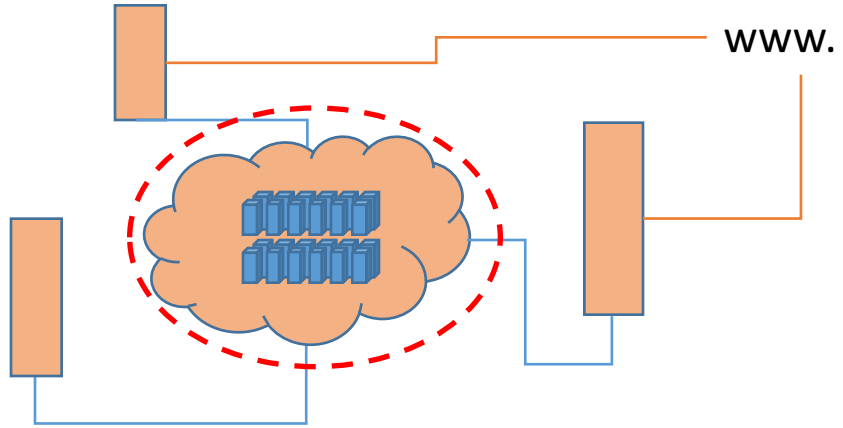
Locations of big Data Centers

- Example: Amazon Web Services:



Source: https://aws.amazon.com/de/about-aws/global-infrastructure/

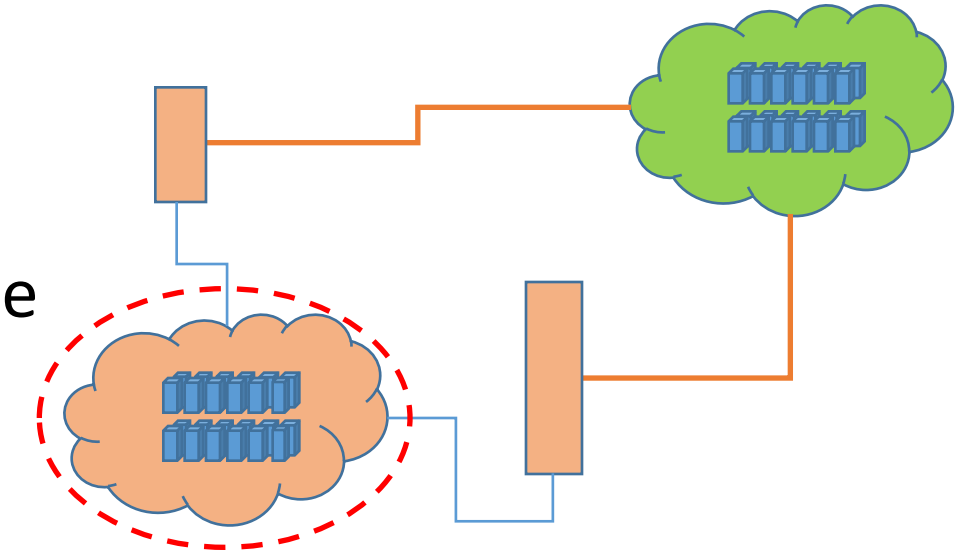
Private / Hybrid / Public Clouds



- Private Cloud Solutions
  -> on-premise cloud-like Infrastructure

- Public Cloud Solutions
  -> off-premise services offered to multiple customers (AWM, Azure,…)

- Hybrid Solutions
  -> Partly on-premise, partly off-premise



But is it worth it?

Pros and Cons

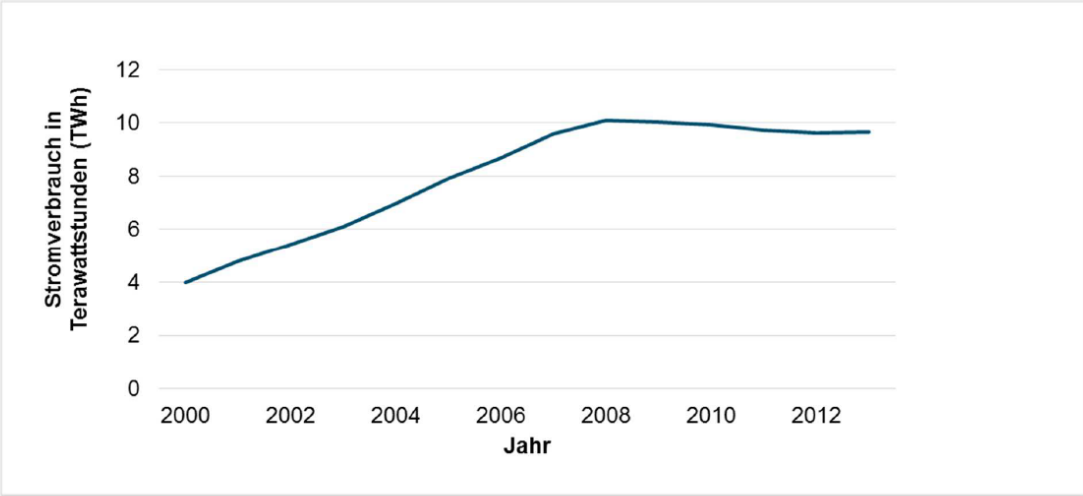
(Potentially) Negative Aspects of Cloud Computing

- Total dependance on the provider
  - Data Integrity
  - Loss of Data
  - Uptime
  - Switching/Going out of business
  - Security?!
  - Data Privacy -> Legal issues (especially in Germany)
  - Physical location of data centers might not be transparent

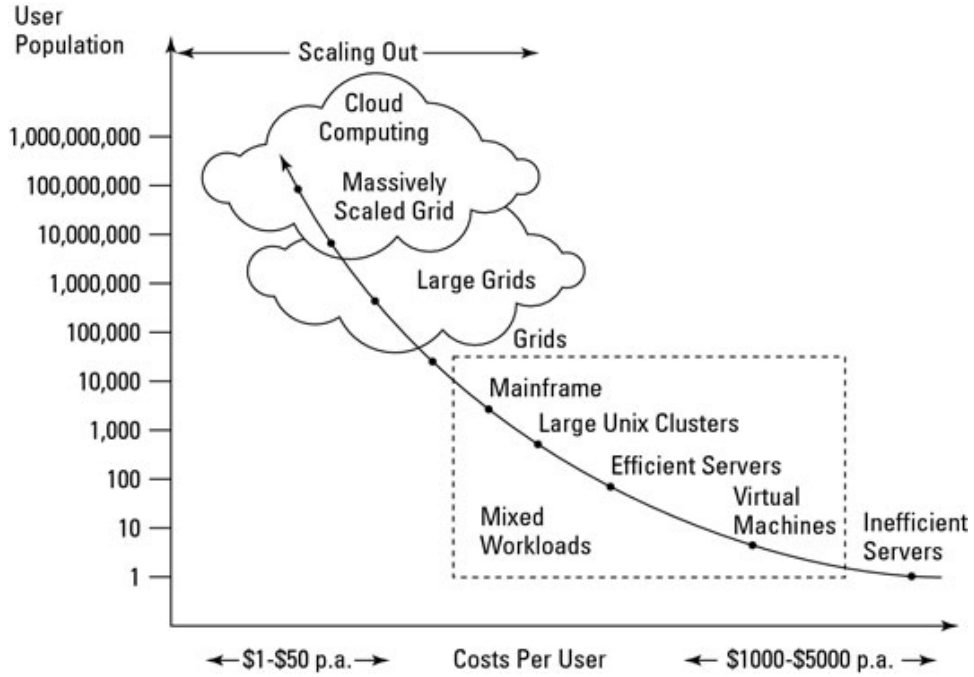
Positive Impact of Cloud Computing

- Less cost
- Productivity/Scalability
- Efficiency

Abbildung 15: Entwicklung des Stromverbrauchs der Server und Rechenzentren in Deutschland



Quelle: Borderstep



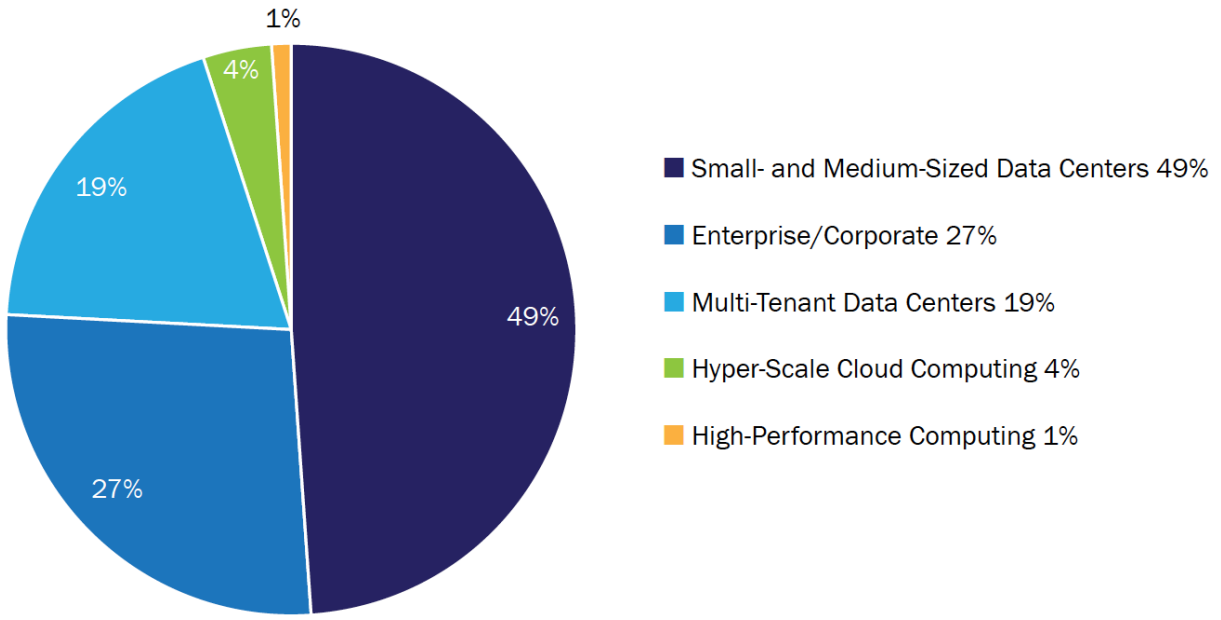
A Closer Look at Efficiency

- Average servers run at only 12-18% capacity

Lets do a little math:

On premise: 15% utilization v/s **65% in the Cloud**
On premise: **29% less efficient than cloud**

=> In the Cloud: **84% less energy** than on premise



- Server rooms of small companies (in the US) are responsible for *half of all US server electricity consumption*

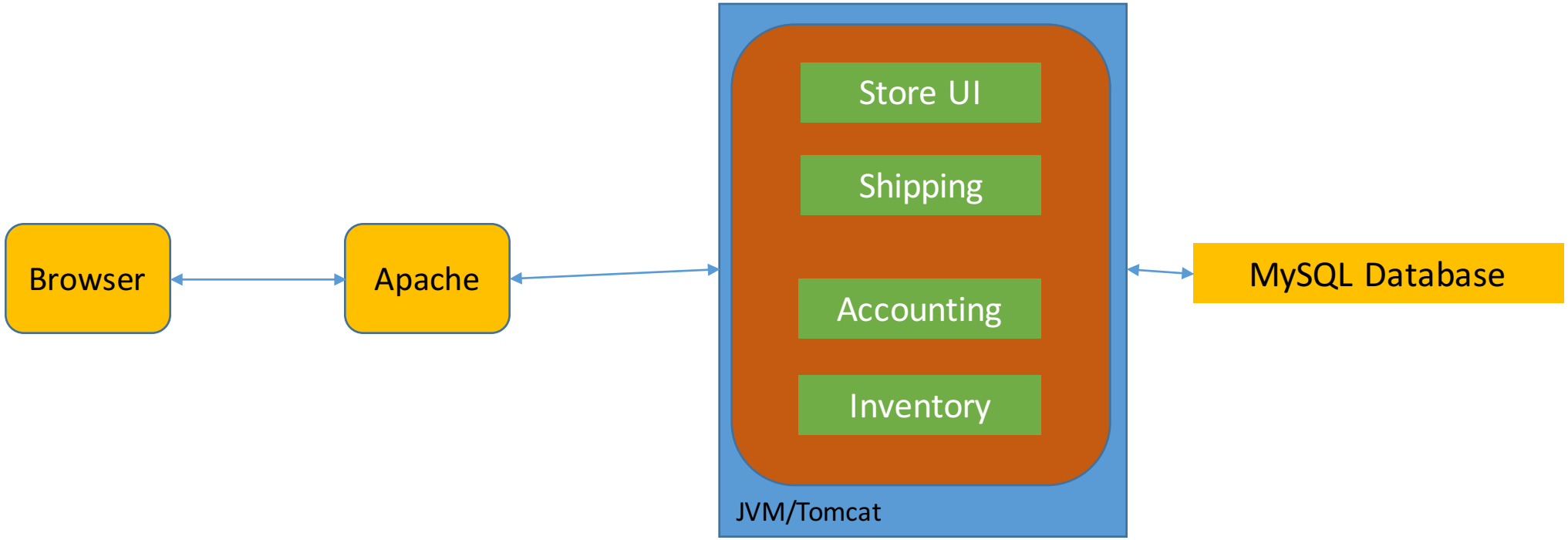
Microservices

They might be small, but they are pretty cool

Prelude: Monolithic Architecture

= building a program as one big application that handles everything the program should handle

For example: *A simple e-commerce website*



Prelude: Monolithic Architecture

Pros

+ Simple deployment: Only one program file (.war in example)
+ Fast and comfortable development (existing IDEs are tailored for monoliths)
+ Easily scalable for more traffic (single dimension scaling)

Prelude: Monolithic Architecture

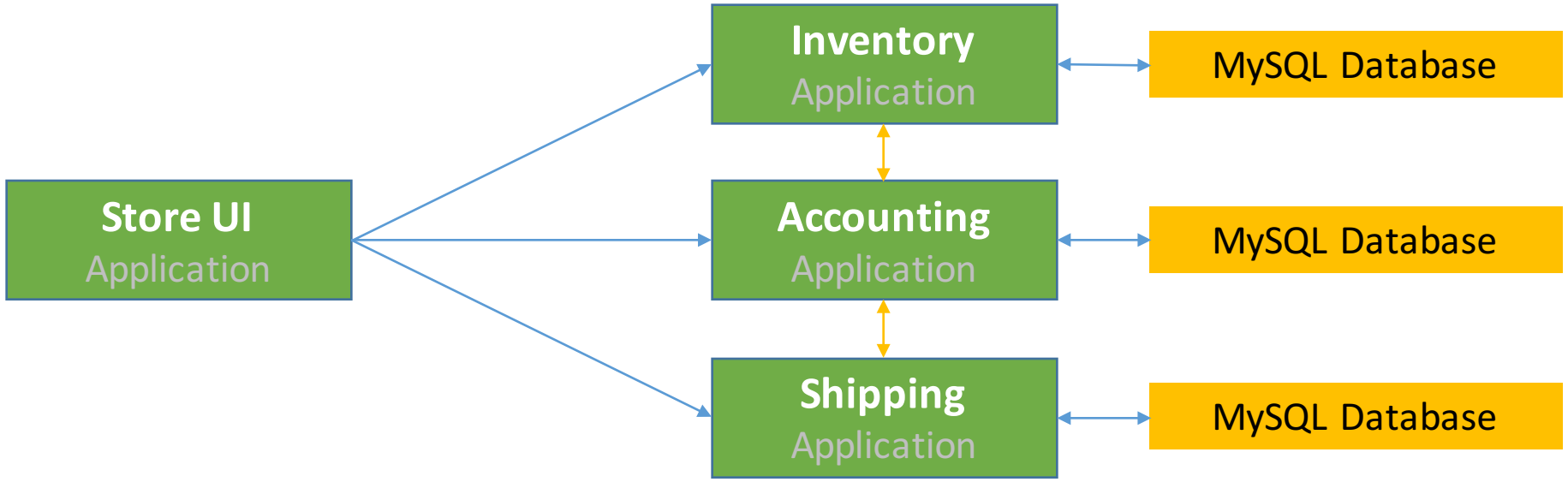
Cons:

- Huge code base -> intimidating for new developers joining the team
  -> IDE becomes slower, starting the app becomes slower
  -> Deployment of new versions becomes slower
- Loss of modularity over time (lazy "shortcuts" will be used)
- Updating changes of one aspect (e.g. UI) results in redeploying entire application
- Teams can't work independently on parts of the application
- Scaling becomes very difficult
- Adapting new technologies becomes impossible

The alternative approach: *Microservice Architecture*

= splitting up the application into a number of independent services, each running in its own process, having only one responsibility/task

-> Messages betw. services through a simple protocol (HTTP resource API)



Key Aspects of the Microservice Architecture

- **independent services**
- **Dumb pipes - Smart Ends**
- Each service is only responsible for a **very narrow (related) set of functions**

  -> *Single repsonsibility principle*
- Each service has **its own database**

  -> consistency between data bases has to be realized

Pros

+ Small code base per service -> easier for new developers
+ Service failures won’t affect the entire System
+ Deploying new versions is fast and (almost) painless
+ One (independent) team per service speeds up development
+ Multiple Platforms/Programming Languages can be use
+ Modularity is preserved
+ Scaling services individually (via Cloud-Platforms)

Cons

- Organizing and testing becomes very complex
- Design for failure
- Overhead for automated tests and deployment
- Fast switch from monolithic architecture almost impossible
- Finding good modularity
- Updating a service API without breaking ”customer services”

Microservices in Real Life

You like Netflix? Good, because we're gonna look at Netflix now.

Netflix

- One of the first companies to switch to Microservices
- 2009: Movie encoding as a separate service
- 2010: Sign-Up, Movie selections, TV-selections, device configuration
- By December 2011 all of Netflix is in the cloud (500+ microservices)

Why Microservices – and why in the cloud?

- In 2008 a single missing semicolon crashed Netflix for several hours
- Availability
- Adaptive Scaling (Each service can be scaled independently!)
- Testing and deploying new services on a global scale

Back to the Future

How will Cloud Computing evolve?

Interesting Parallels

"I think there is a world market for maybe five computers."

Thomas Watson, president of IBM, 1943