

3.– 6. September 2012  
in Nürnberg



# Herbstcampus

Wissenstransfer  
par excellence

## Darf's ein wenig größer sein?

Architekturmuster für hochskalierbare Systeme

Uwe Friedrichsen

codecentric AG

# ÜBER MICH ...

---

Name: Uwe Friedrichsen

Berufserfahrung: Relativ vielfältig und lang

Schwerpunkte:

- Teams, Projekte und Systeme zum Erfolg führen – mit einem speziellen Fokus auf Architektur und Agilität
- Ganzheitliches Denken, Ideen und Konzepte verknüpfen, Leute zum Nachdenken bringen
- Neue Konzepte & Technologien

Position: CTO bei codecentric AG



# AGENDA

---

## Basics of scalability

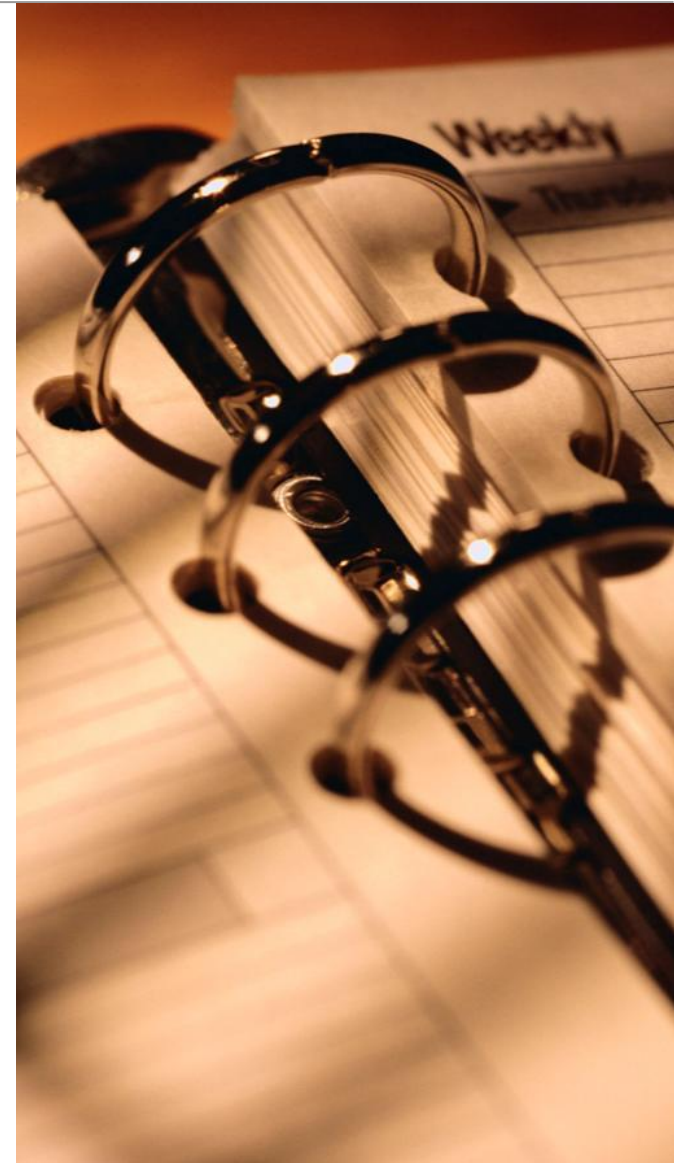
Dimensions of scalability

Design for high scalability

Choose the right tool

More stuff ...

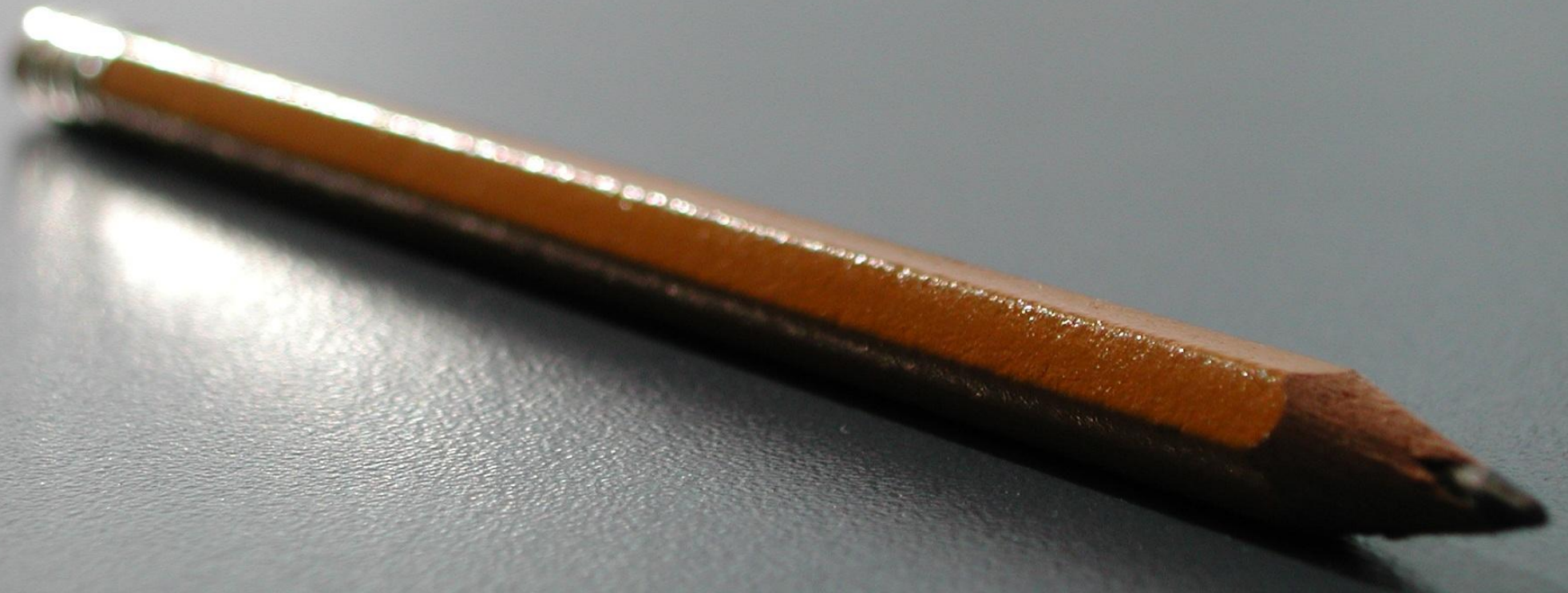
Summary





# Strive for Simplicity

The system should be made as simple as possible (- but no simpler)





Five identical green glass beer bottles are lined up on a metal wire rack. The bottles are covered in condensation droplets, suggesting they are cold. They are filled with a light-colored liquid, likely beer, and have black caps. A semi-transparent text box is overlaid across the middle of the bottles.

Scale out, not up



Be stateless





Share nothing





Communicate  
asynchronously



# AGENDA

---

Basics of scalability

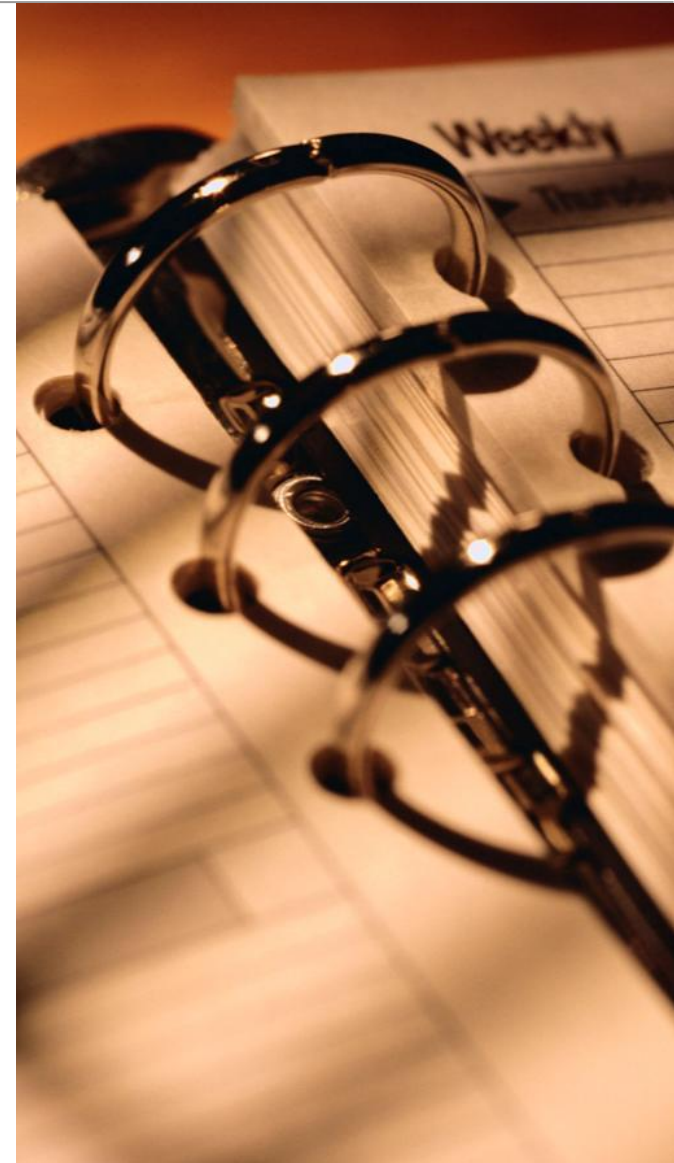
Dimensions of scalability

Design for high scalability

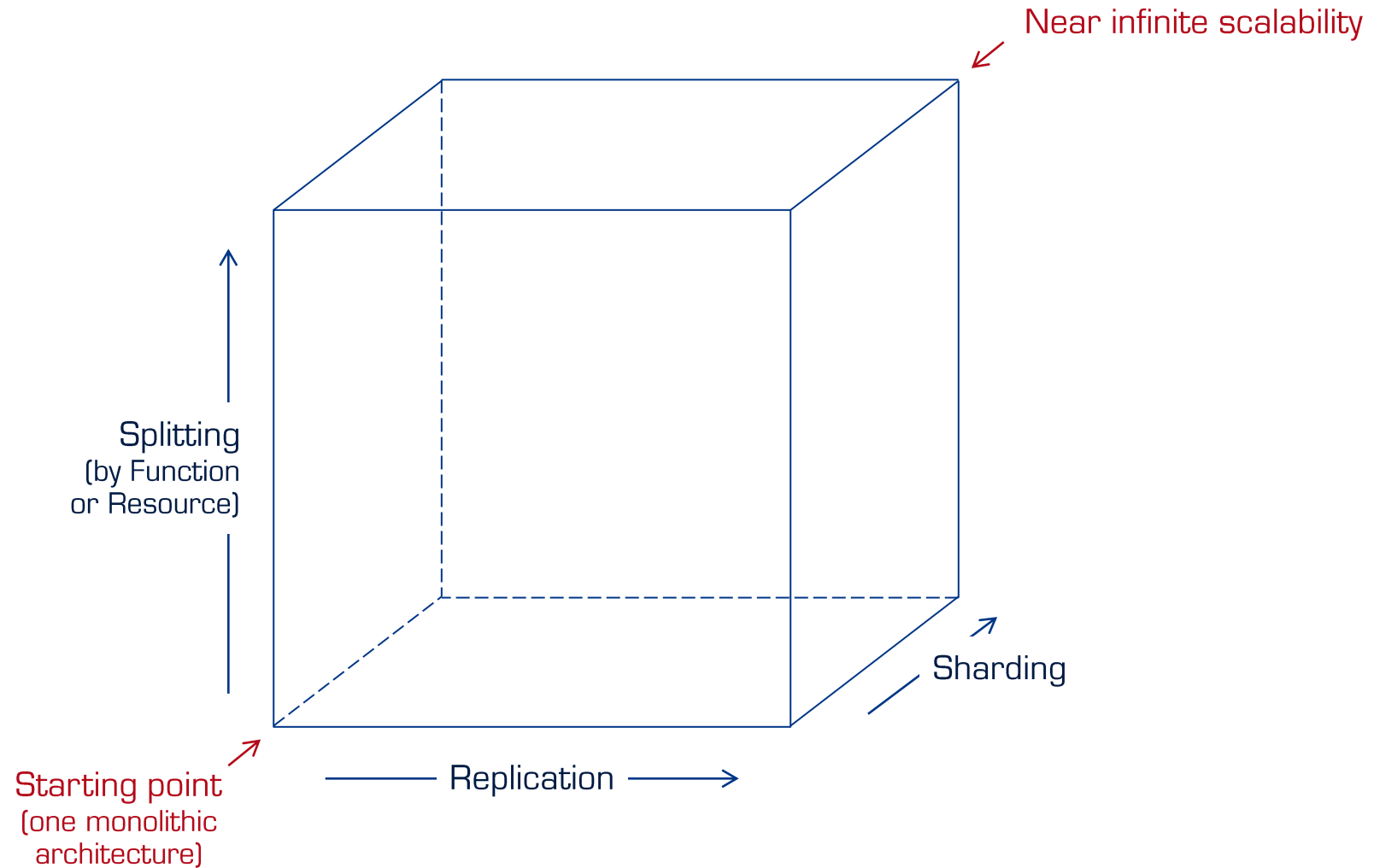
Choose the right tool

More stuff ...

Summary



# DIMENSIONS OF SCALABILITY



Source: [1]



# REPLICATION

---

## *When to use*

- Scaling of transactions, not data
- High read to write ratio

## *How to implement*

- Clone services and distribute calls via load balancer
- Use replication features of infrastructure components  
(database, app-server)

## *Related Concepts*

- Load balancer & shared nothing units
- Load balancer & stateless nodes & scalable storage
- Master slave replication
- Masterless replication

## *Tradeoffs*

- Doesn't scale well with high write rates
- Doesn't work for unpredictable data volumes



# SPLITTING

---

## *When to use*

- Very large data sets of loosely coupled data
- Large complex systems with loosely coupled functionality

## *How to implement*

- Split up by noun (data) or verb (function)
- Best implemented with shared nothing units and/or asynchronous communication

## *Related Concepts*

- Service oriented architecture (SOA)
- Resource oriented architecture (ROA)
- Pipe & Filter

## *Tradeoffs*

- Data and/or functions aren't always suitable for splitting
- No tool support (as for replication or sharding)
- Long functional chains make system less reliable





# SHARDING

---

## *When to use*

- Very large data sets of tightly coupled data
- Unpredictable data volume, rapidly growing data sets
- High write rates

## *How to implement*

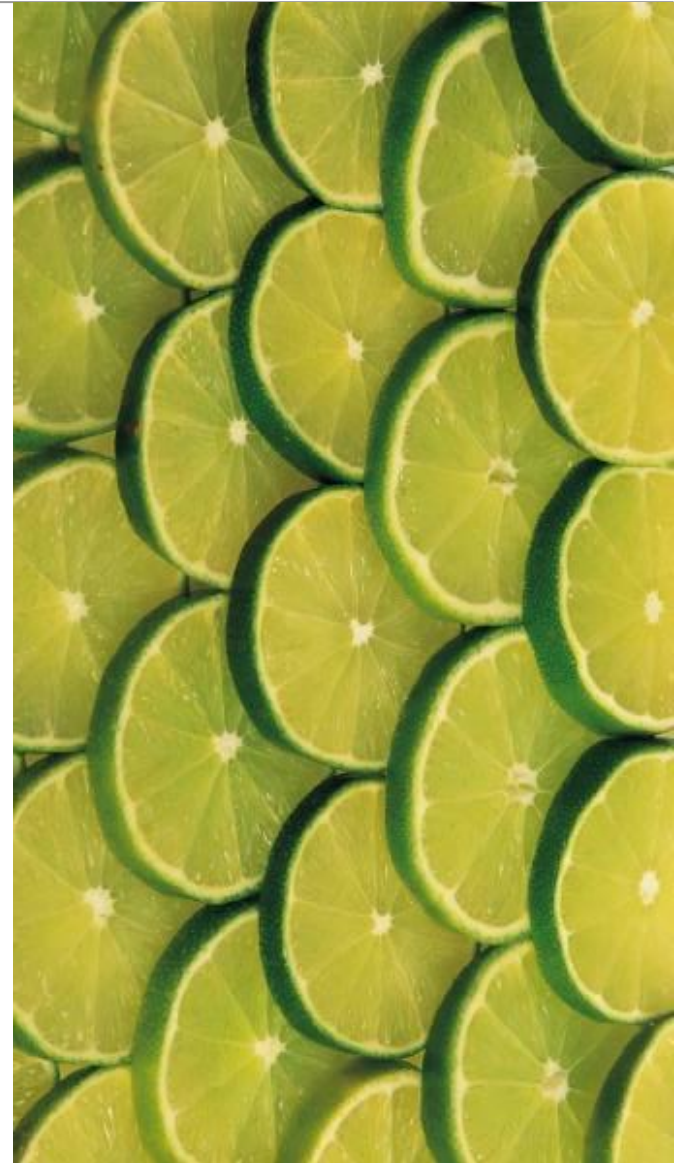
- Use sharding feature of your database component

## *Related Concepts*

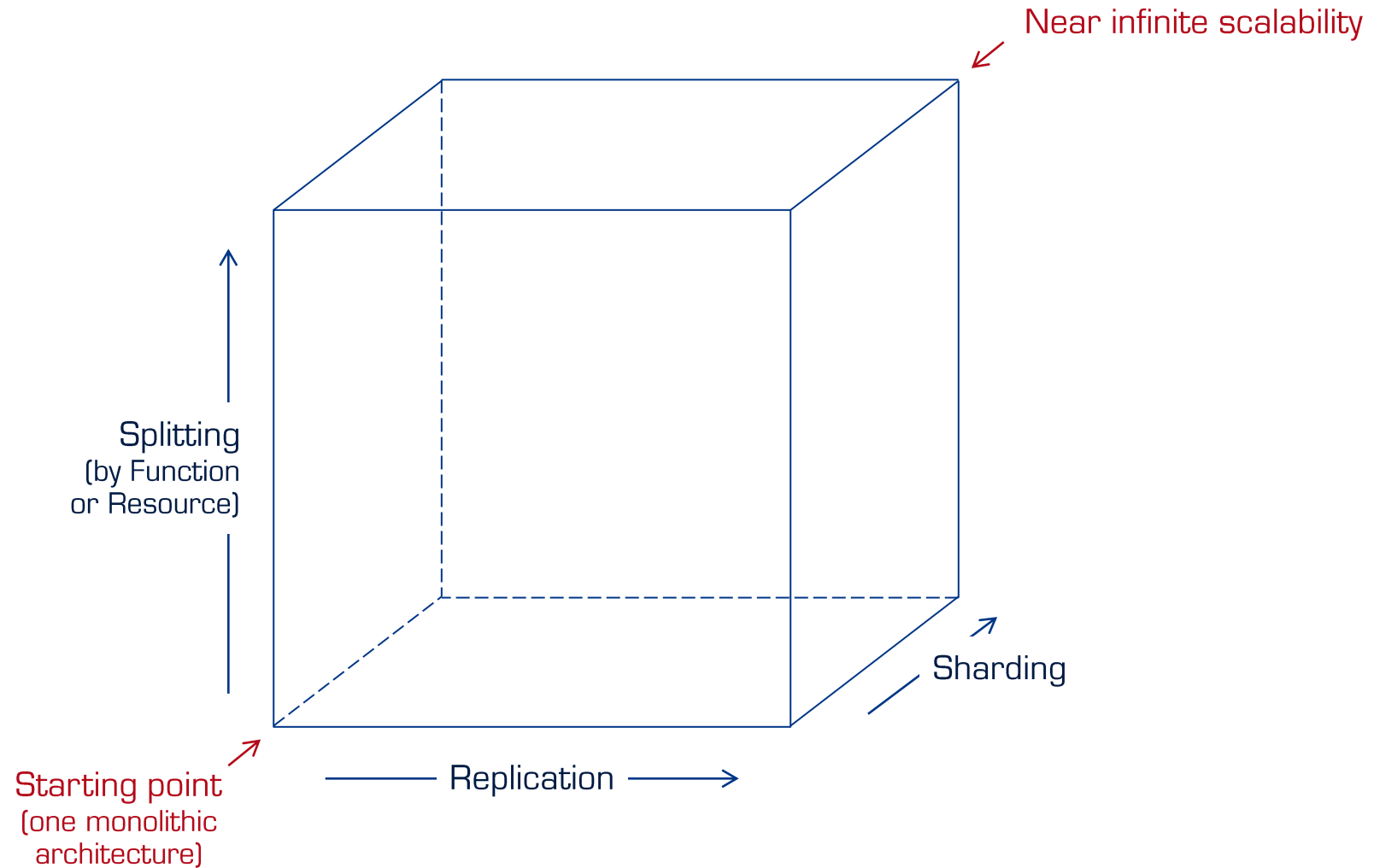
- Consistent Hashing
- Scatter & gather (MapReduce)

## *Tradeoffs*

- Consistency is relaxed (especially for indices)
- Simple sharding does not work well for rapidly growing data sets
- Expensive in terms of required computing resources



# DIMENSIONS OF SCALABILITY



Source: [1]



# AGENDA

---

Basics of scalability

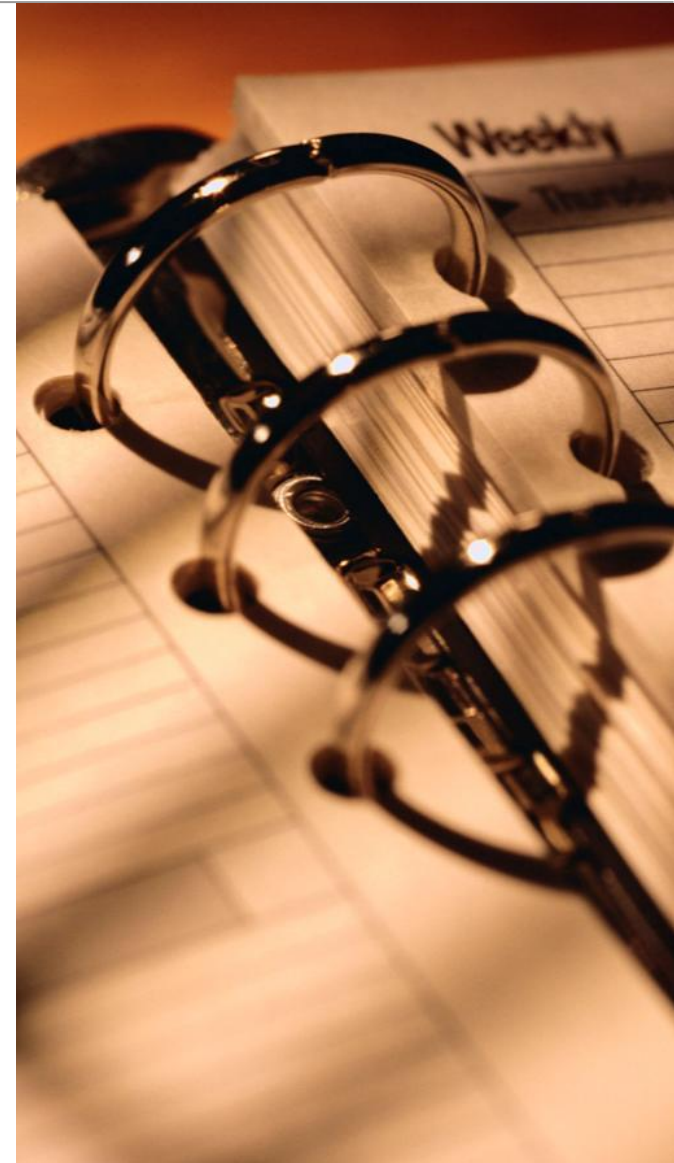
Dimensions of scalability

Design for high scalability

Choose the right tool

More stuff ...

Summary



The image features a 3x4 grid of 12 identical analog clocks. Each clock has a white face with black numbers from 1 to 12, black hour and minute hands, and a thin orange second hand. The clocks are arranged in three rows and four columns. In the center of the grid, there is a white rectangular box containing the text "Relax temporal constraints" in a dark blue, serif font. The clocks show various times: the top row shows approximately 1:50, 2:10, 6:00, and 12:10; the middle row shows approximately 10:10, 10:50, 6:00, and 3:10; the bottom row shows approximately 9:10, 4:10, 2:10, and 7:10.

Relax temporal  
constraints



# RELAX TEMPORAL CONSTRAINTS

## *When to use*

Distributed data sets and the CAP theorem gets into the way  
Availability is more important than immediate consistency

## *How to implement*

Consider carefully your availability and consistency requirements  
Use a datastore that supports your requirements

## *Related Concepts*

Quorum

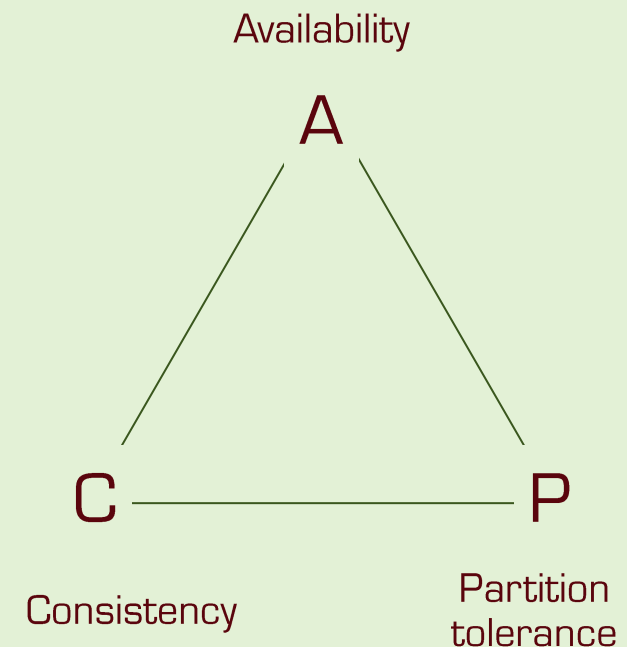
Harvest & Yield

Hinted handoffs

## *Tradeoffs*

BASE consistency, no ACID consistency

Application might need to handle temporal inconsistencies itself





# Question:

Why do people insist in ACID transactions even though the real world is always BASE?



# Cache as cache can





# CACHE AS CACHE CAN

---

## *When to use*

Short response times and/or high throughput are important  
High read to write ratio

## *How to implement*

Use caches built-in to products currently in use  
Insert dedicated cache layers into your architecture  
Plan – Do – Check – Act

## *Related Concepts*

Content Delivery Networks  
HTTP Caching – Expires and ETag headers  
Distributed caches

## *Tradeoffs*

Caches may become stale  
Balance between response time and additional resources



A close-up, top-down view of a person's hands sorting through a large metal filing cabinet. The cabinet is filled with numerous folders and papers in various colors, including yellow, white, and brown. The person's hands are visible, with one hand resting on a folder and the other reaching in to adjust or retrieve a document. The text is overlaid on a semi-transparent white rectangular box in the center of the image.

Keep dynamic data closer to the compute  
and static data closer to the end-user

# DYNAMIC AND STATIC DATA

---

## *When to use*

Optimized response times with large heterogeneous data sets  
Network bandwidth and/or transfer costs are an issue

## *How to implement*

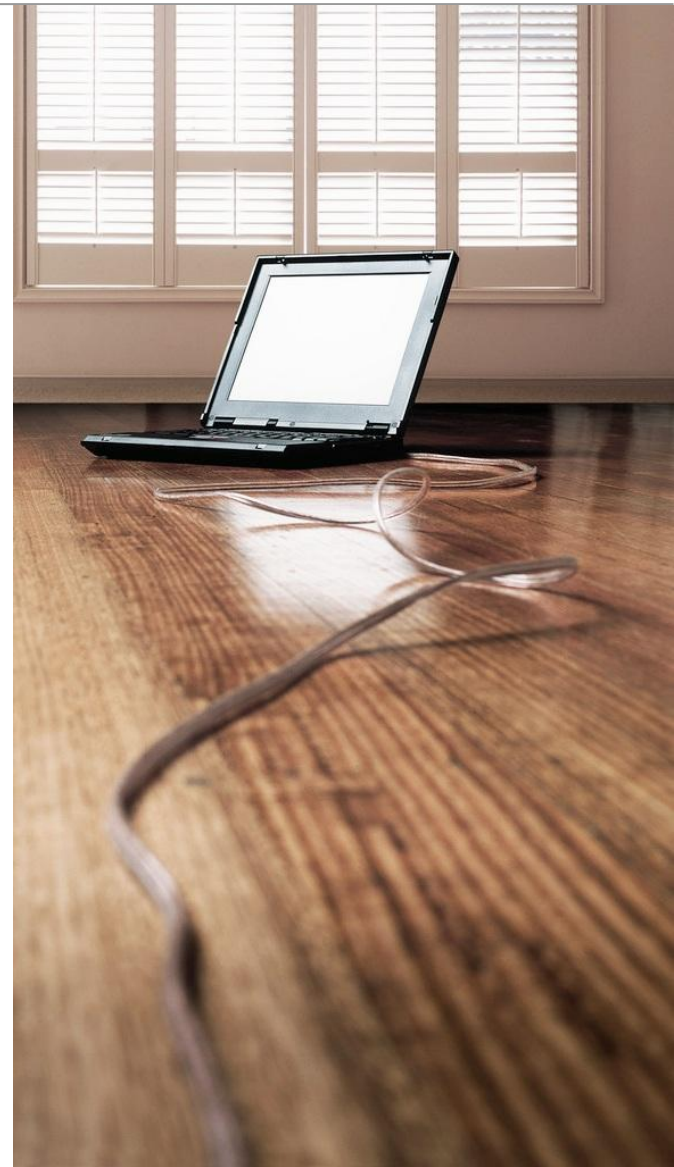
Analyze your data and identify static and dynamic portions  
Store dynamic data on same nodes where processing logic is  
Move processing logic to the data storage  
Use frontend caches for static data

## *Related Concepts*

Shared nothing  
Scatter & gather  
Content Delivery Networks  
HTTP Caching – Expires and ETag headers

## *Tradeoffs*

Must be implemented explicitly





# AGENDA

---

Basics of scalability

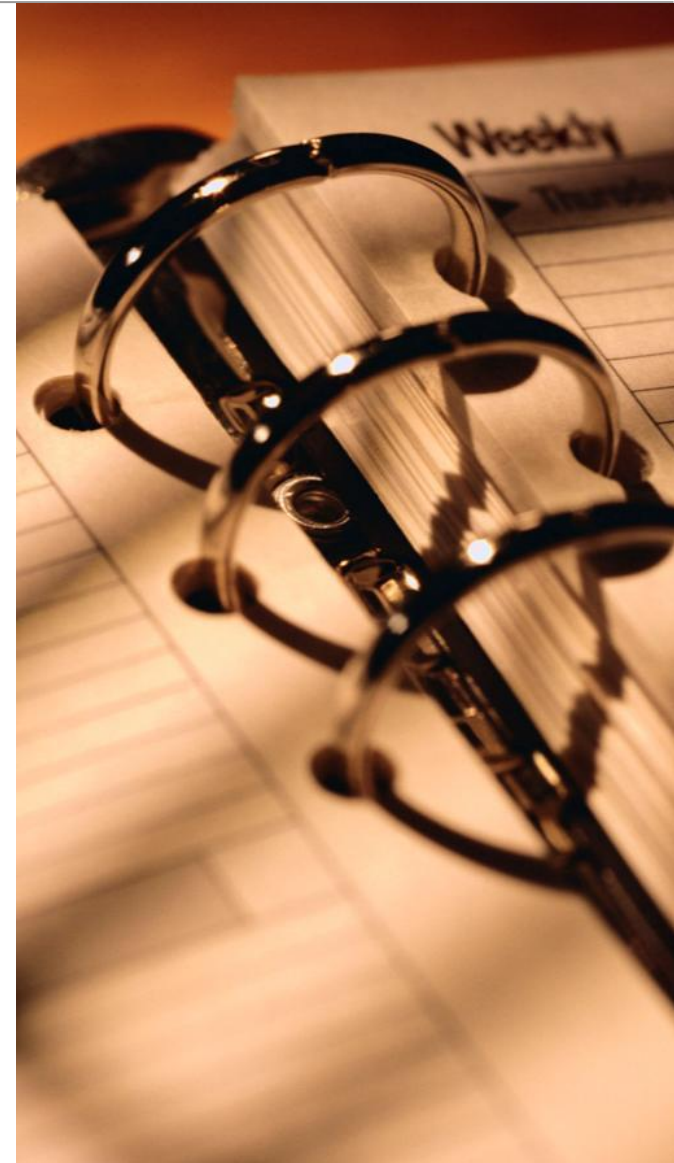
Dimensions of scalability

Design for high scalability

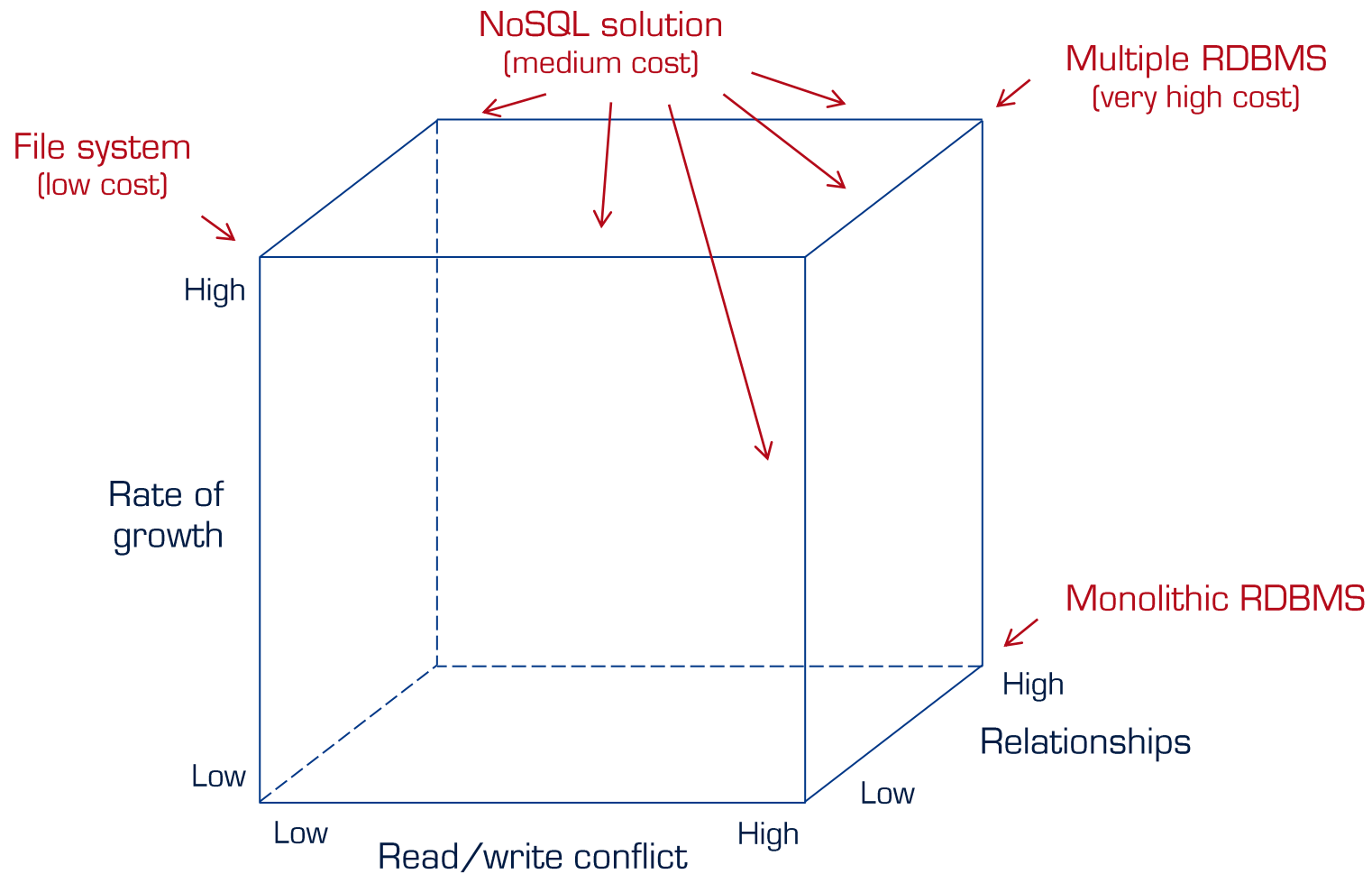
Choose the right tool

More stuff ...

Summary



# STORAGE TECHNOLOGY DECISION CUBE



Source: [1]

# STORAGE TECHNOLOGIES

---

## *File system*

- Easy to use, scales extremely well
- Can handle large entries well
- Poor support for relations and concurrency

## *NoSQL*

- Fills the gap between file systems and RDBMS
- Very diverse technologies: KV, Wide column, Document, Graph
- Different strategies w.r.t. CAP and scalability support
- Searching is often an issue
- Relatively young technology (stability, tools, documentation)

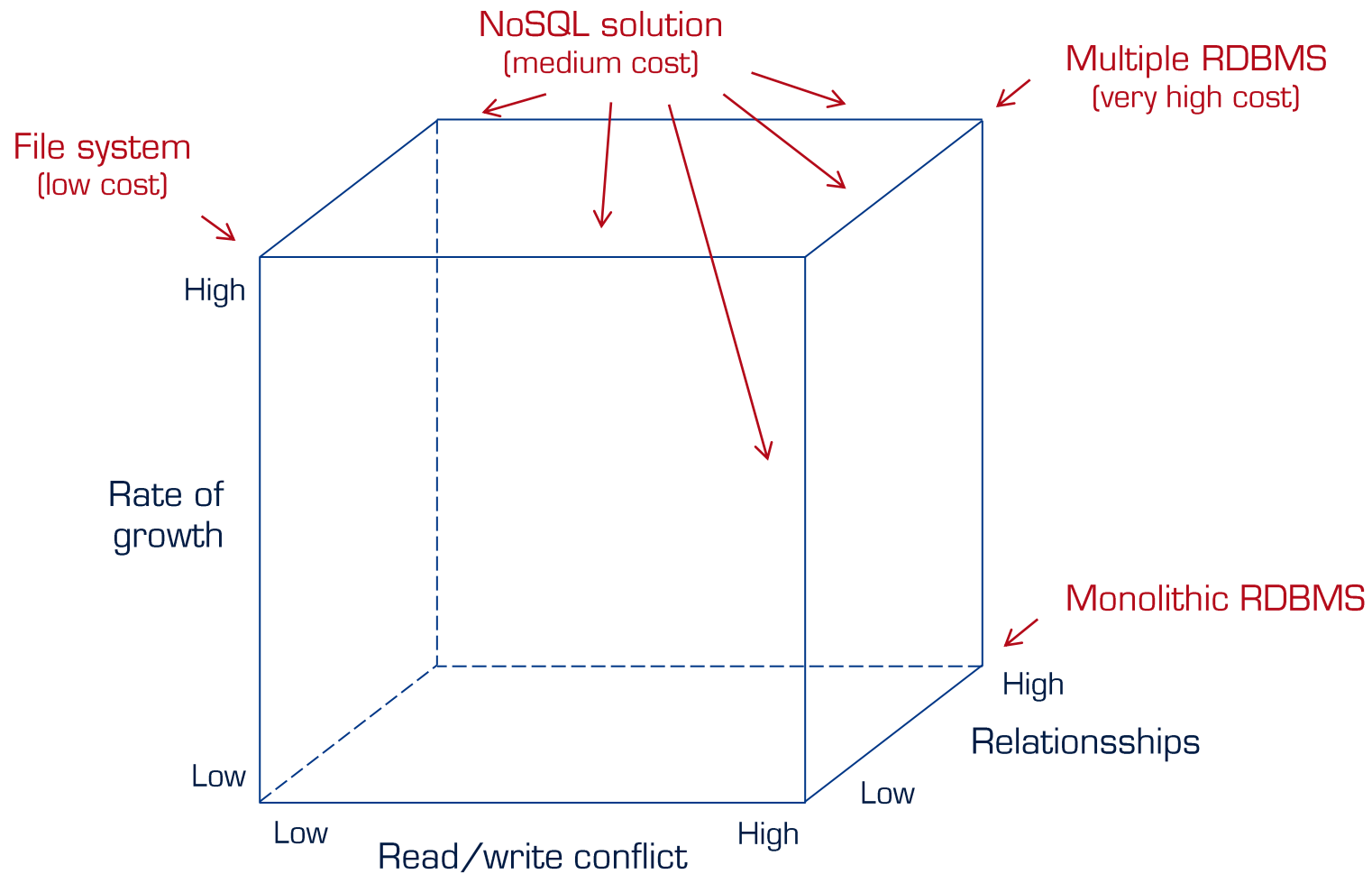
## *RDBMS*

- Very strong at relations and concurrency handling
- Don't scale well beyond a certain boundary
- Mature technology, very good tool support





# STORAGE TECHNOLOGY DECISION CUBE



Source: [1]

A beach vendor cart is parked on a sandy beach under a clear blue sky. The cart is a metal frame with a blue roof. It is heavily laden with various items for sale. On the top shelves, there are several large, colorful, circular jewelry pieces, possibly medallions or pendants, and many long, thin necklaces hanging from the sides. Below these, there are more necklaces and bracelets. In the foreground, several woven straw hats are displayed on the cart, some with decorative ribbons. The cart is positioned on the sand, and the ocean is visible in the background.

Be wary of vendor solutions

# AGENDA

---

Basics of scalability

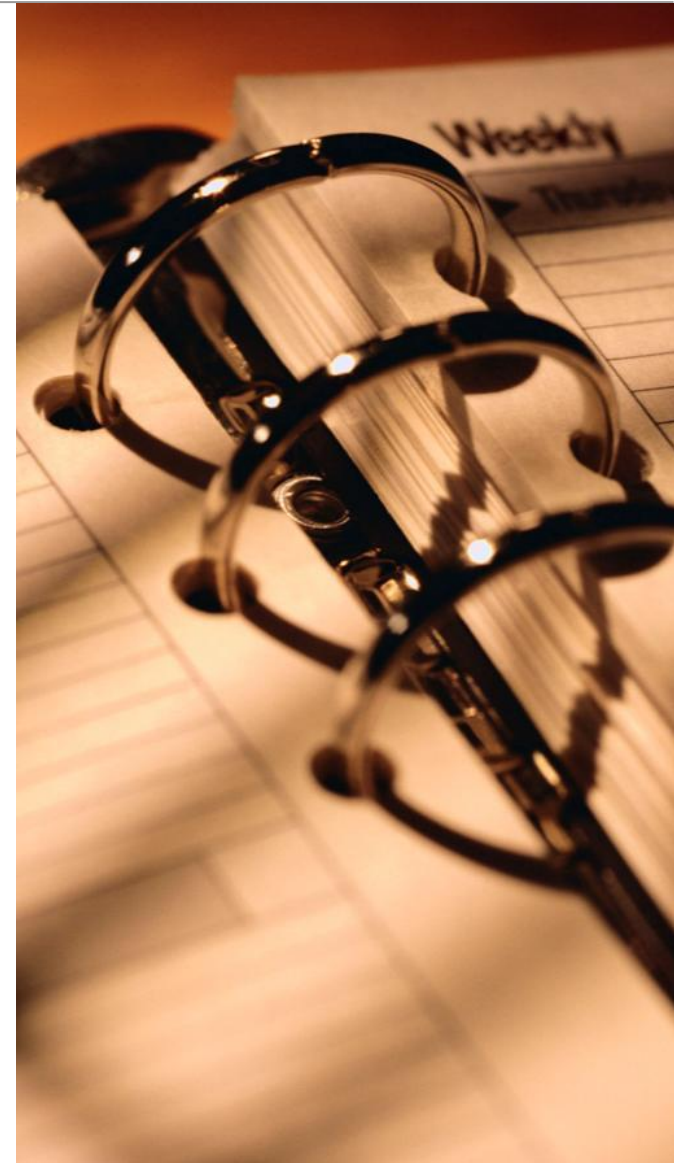
Dimensions of scalability

Design for high scalability

Choose the right tool

More stuff ...

Summary





# MORE PATTERNS AND PRINCIPLES ...

---

## *Fault tolerance*

- Be prepared for things to crash
- Deal with inconsistencies

## *Database handling*

- Use the right type of locking
- Don't use '\*' in select statements

## *Using web technologies best*

- Unleash the potential of HTTP

## *Extreme scalability*

- Automation and elasticity
- Big disjoint entities with message based communication

## *And many more ...*

- Monitoring the system
- Designing and learning recommendations
- Never down system



## MORE TO READ ...

---

- [1] Michael T. Fisher, Martin L. Abbott,  
Scalability Rules: 50 Principles for Scaling Web Sites,  
Addison-Wesley Longman, 2011
- [2] Theo Schlossnagle, Scalable Internet Architectures,  
Sams, 2005
- [3] Jinesh Varia, Architecting for the Cloud: Best Practices,  
Amazon Web Services 2010
- [4] Jinesh Varia, Cloud Architectures,  
Amazon Web Services 2008
- [5] Pat Helland, Life beyond Distributed Transactions,  
3rd Conference on Innovative DataSystems Research  
(CIDR) 2007
- [6] <http://highscalability.com/>
- [7] [http://www.slideshare.net/jboner/  
scalability-availability-stability-patterns](http://www.slideshare.net/jboner/scalability-availability-stability-patterns)



# AGENDA

---

Basics of scalability

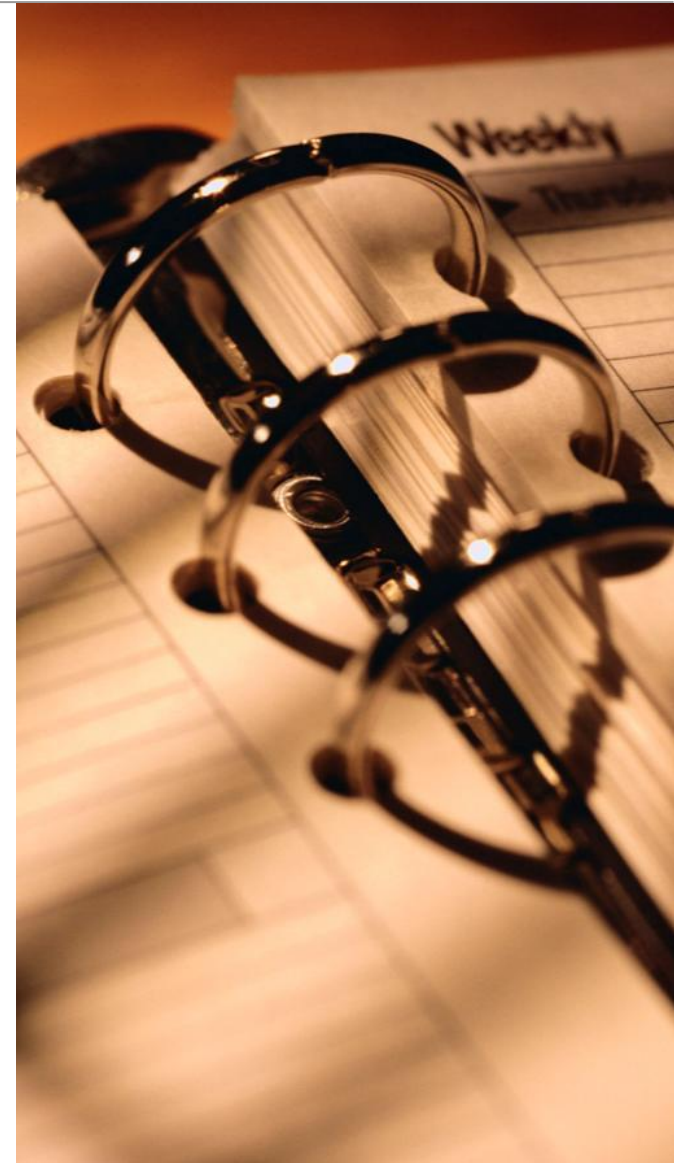
Dimensions of scalability

Design for high scalability

Choose the right tool

More stuff ...

Summary



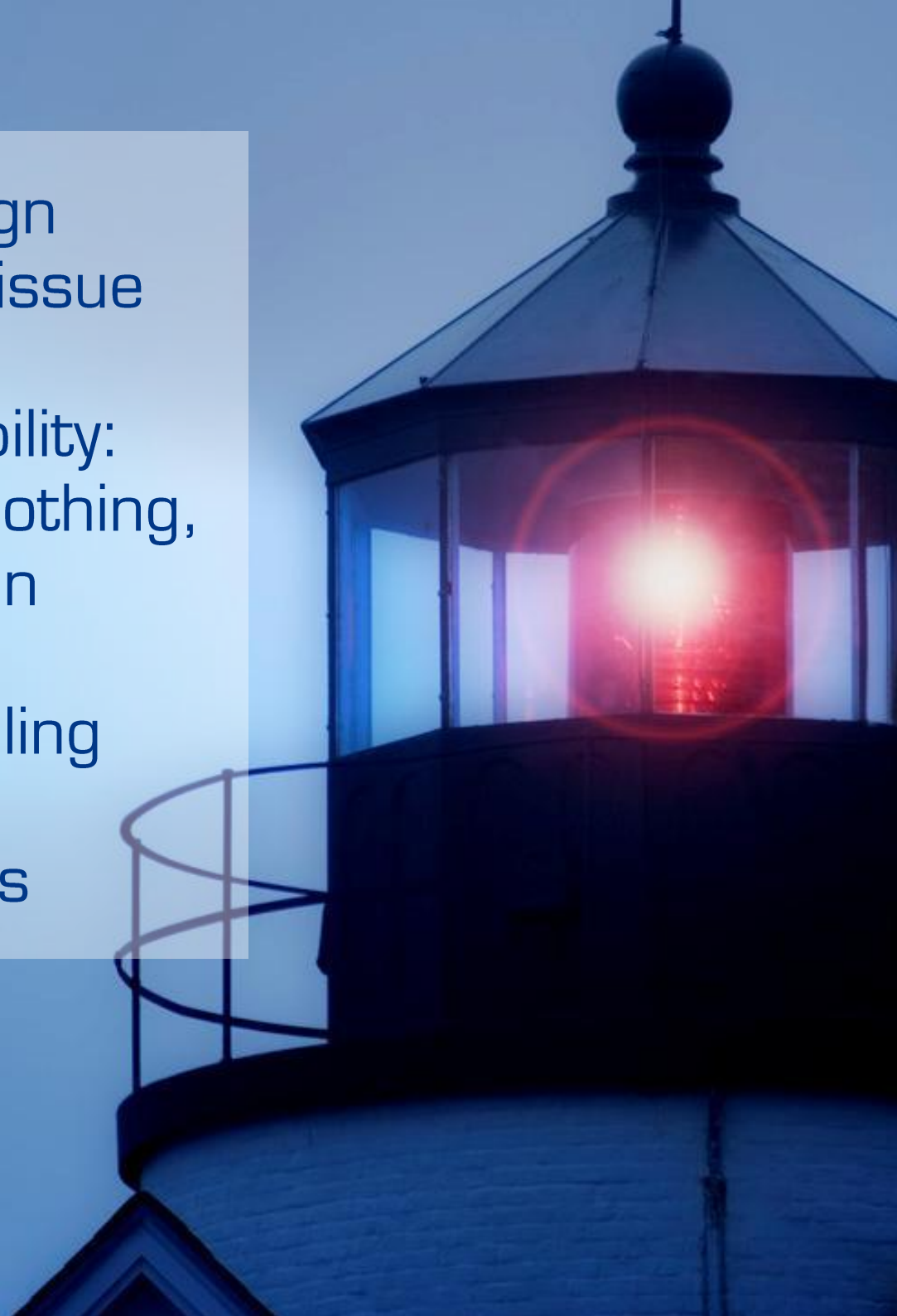


Scalability is primarily a design issue, not a tool or product issue

The core principles of scalability: simplicity, scale out, share nothing, asynchronous communication

The three dimensions of scaling

The different storage choices



# VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!

---

Uwe Friedrichsen  
CTO

codecentric AG  
Merscheider Straße 1  
42699 Solingen

uwe.friedrichsen@codecentric.de  
tel +49 (0) 212 . 23 36 28 10  
fax +49 (0) 212 . 23 36 28 79  
mobil +49 (0) 160 . 90 62 66 00

[www.codecentric.de](http://www.codecentric.de)  
[blog.codecentric.de](http://blog.codecentric.de)  
[www.meettheexperts.de](http://www.meettheexperts.de)



# DISKUSSION & FRAGEN

---

