university of koblenz
Computer Science

5. Persistence (Part III)

Engineering Web and Data-intensive Systems

Dr. Volker Riediger - Winter Term 2022/23

Institute for Software Technology

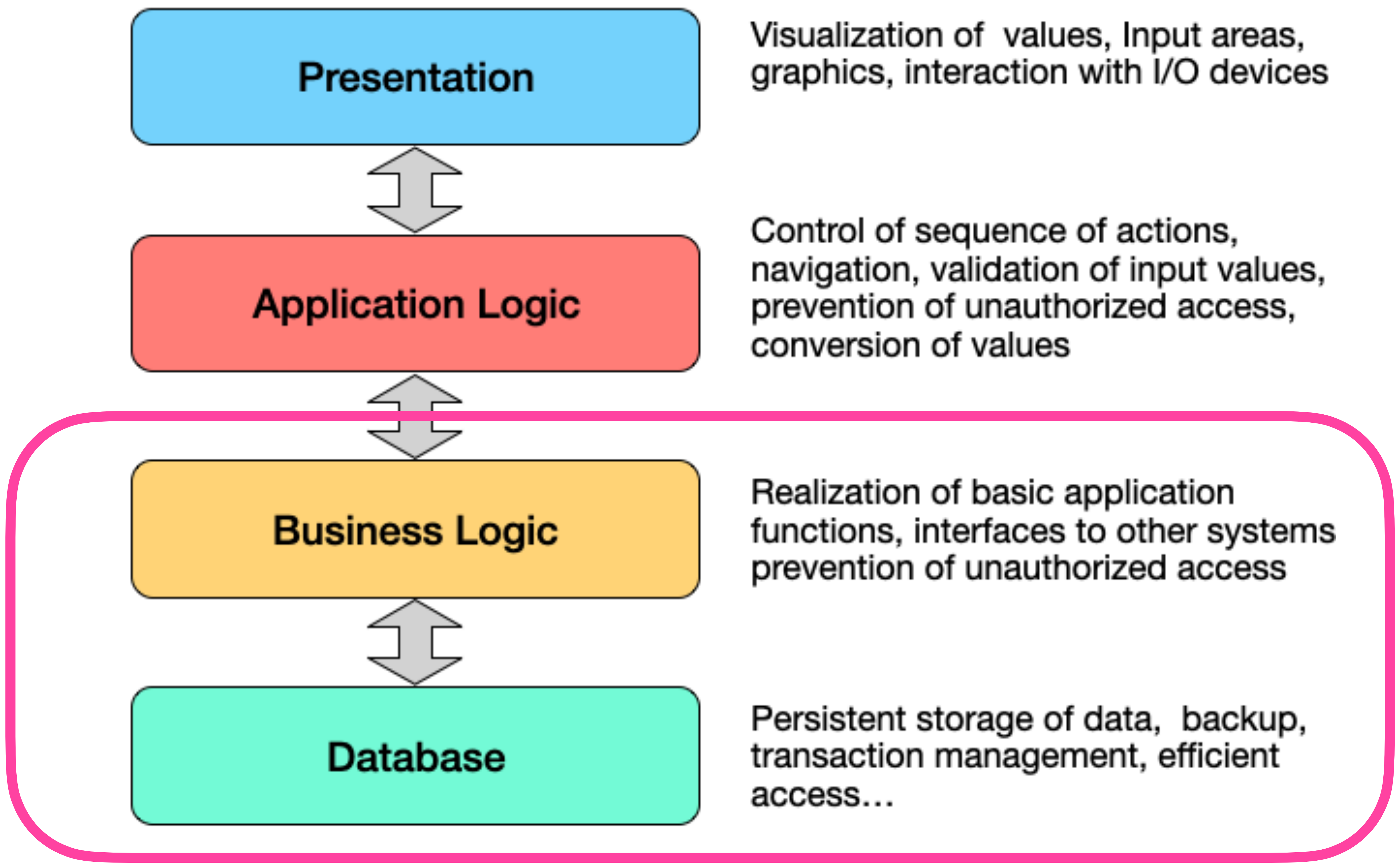
Persistence (Part III)

- Document Databases
- Data Mapping to Documents
- Distributed Data Storage
- Data Intensive Systems
- ACID, CAP and BASE

Image: colourbox.de

Persistence: Overview

- Data Properties (I)
- Persistence Tasks (I)
- Persistence vs. Scaling (III)
  - Data Intensive Systems
  - Distributed Storage
  - CAP, ACID, BASE
- Data Mappings
  - Relational (I)
  - Graph (II)
  - Document (III)



5.5 Data Mapping III - Documents

Document Databases

- Manage documents instead of individual objects
- Documents…
  - are a self-contained unit of interest
  - are identifiable (e.g., by UUID)
  - can have various formats, e.g. text document, image, …
  - are usually structured, e.g., as XML or JSON
- Engines support
  - efficient access by sophisticated index structures
  - distributed storage
  - REST access
  - versioning and automated replication

Document Databases

- Flexible schema
  - documents can have individual structure
  - relations between documents not explicitly represented
  - semantics up to application
- As with every schema-less technology: querying needs schema information
- Best suited for “document-like”, probably inhomogeneous, data

- In contrast to relational/and or graph databases
  - probably higher redundancy in data
  - no standardized query language
  - applications need to care for relations between documents
  - more programming required on the persistence layer
  - **no “recipes” for data mapping from OO models**

How to map OO models to document models?

A suggested approach for Object Document Mapping (ODM)

- Decide which objects/relations belong together and should form a document
- Decision can only be made from an application domain perspective
- Define required document types
  - Identify the relevant fraction of the domain model for each document type
  - For each document type:
    map the domain model excerpt to XML and/or JSON schemas
- Develop the persistence layer of the application

- As of 2021, virtually no support for automatic ODM
  - Maintain instances: Transformation of documents into/from domain objects mostly “manual work”
  - Keeping external and internal state in-sync: mostly “manual work”
  - Some initial Object-Document-Mappers available for PHP, unknown state and quality
    see Doctrine project (visited 2021/02/01)

CouchDB: A Document Store

- (See also demonstration session)
- Stores JSON documents
- Local ACID transactions (see later in this lecture)
- Optionally partitioned storage
- Easy replication (BASE-Style, see later in this lecture)
- Conflict detection based on document revisions,
  only on document level, no structural merge of individual attributes
- REST API
- Heavily relies on index structures
- Developers need JavaScript skills
- Recommended Reading:
  - CouchDB Documentation and Tutorials
  - https://www.dimagi.com/blog/what-every-developer-should-know-about-couchdb/

  (both visited 2021/02/01)

5.6 Distributed Data Storage

General commandment:

Don’t optimize when there’s no bottleneck!

But when there's a bottleneck…

KISS principle: Keep it small and simple

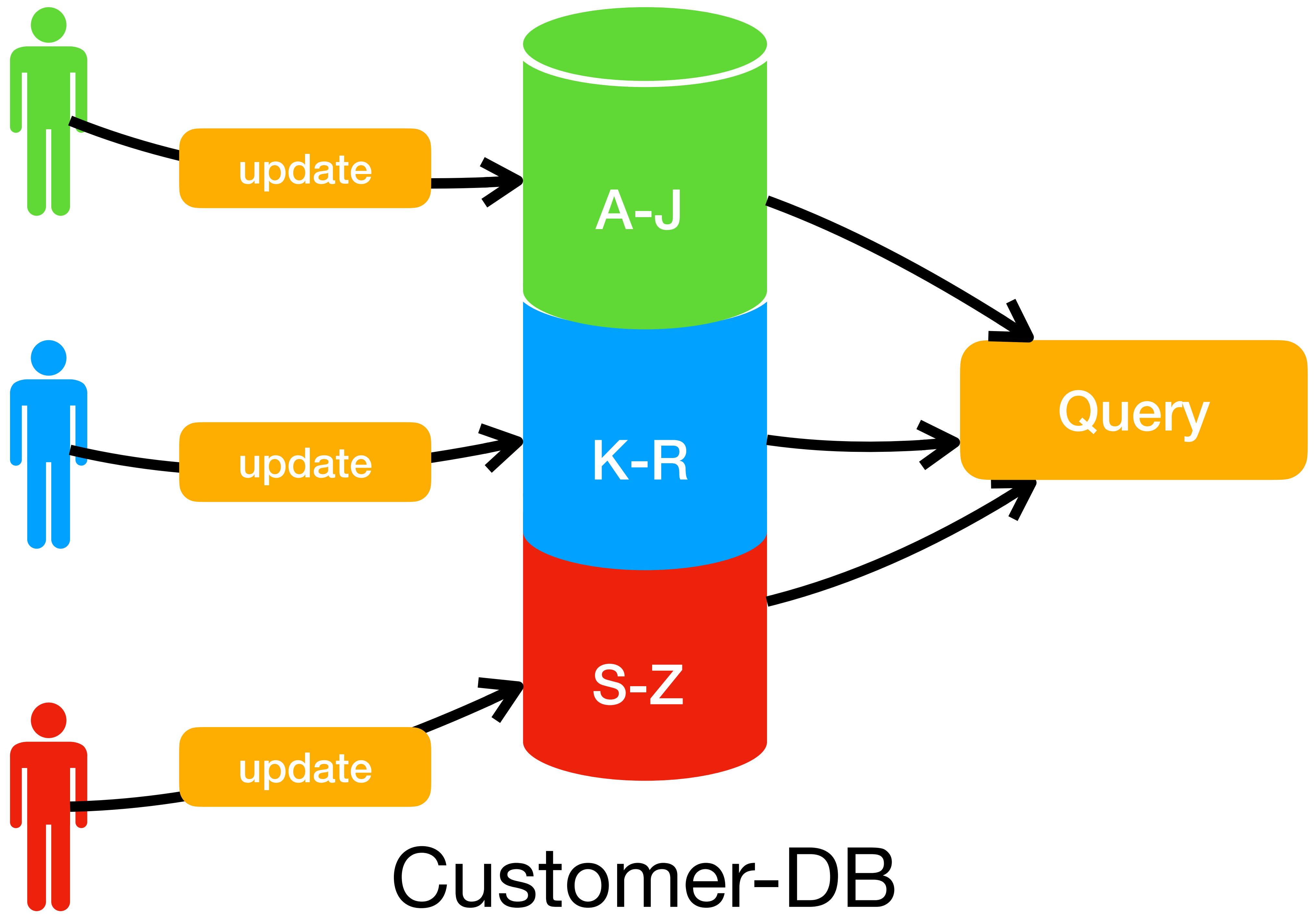
Analyze precisely the **cause** of the bottleneck, then take appropriate actions and countermeasures.

- **Local** measures
  - Vertical Scaling
  - Partitioning
  - Horizontal Scaling
  - Clustering
- **Non-local** measures
  - Distributed storage/processing
  - Replication
  - Sharding

Partitioning

Separate data at logical/organizational boundaries

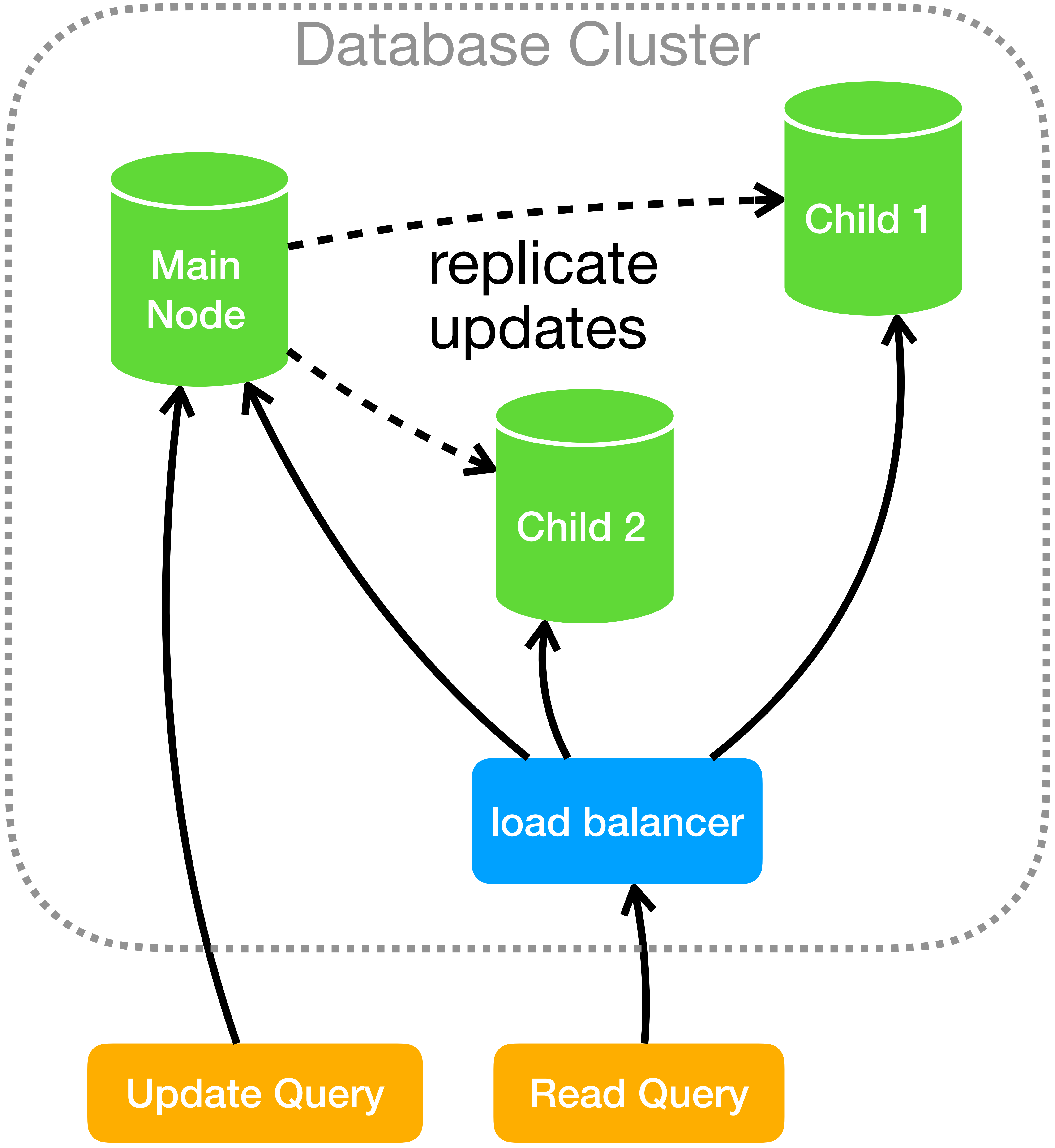
- e.g., distribute customers by name among account managers
- put data in partitions per manager
- concurrent changes most likely hit different partitions
- enables parallelized queries
- minimizes conflicts of write operations
- minimizes blocking due to locked pages



Clustering

Add more servers locally

- local replication
- usually, updates go through a main node
- optionally, all nodes accept updates, more complicated commit protocol
- read queries can be handled by all nodes concurrently
- child nodes can serve as backup



Data Intensive Systems

- Data size and/or global access requirements make local optimization measures infeasible
- Huge amounts of data
  - e.g. physical experiments, earth observation data, social networks, global dictionaries, linked open data, …
- Arbitrary many locations
- Spatially distributed storage

- Balancing processing loads
  - Distributed parallel computing
  - Propagation of local results
- Goals
  - Minimize the need to move data
    "keep data close to it's users"
  - High availability
  - Reliability
  - … and more

Fallacies of Distributed Computing

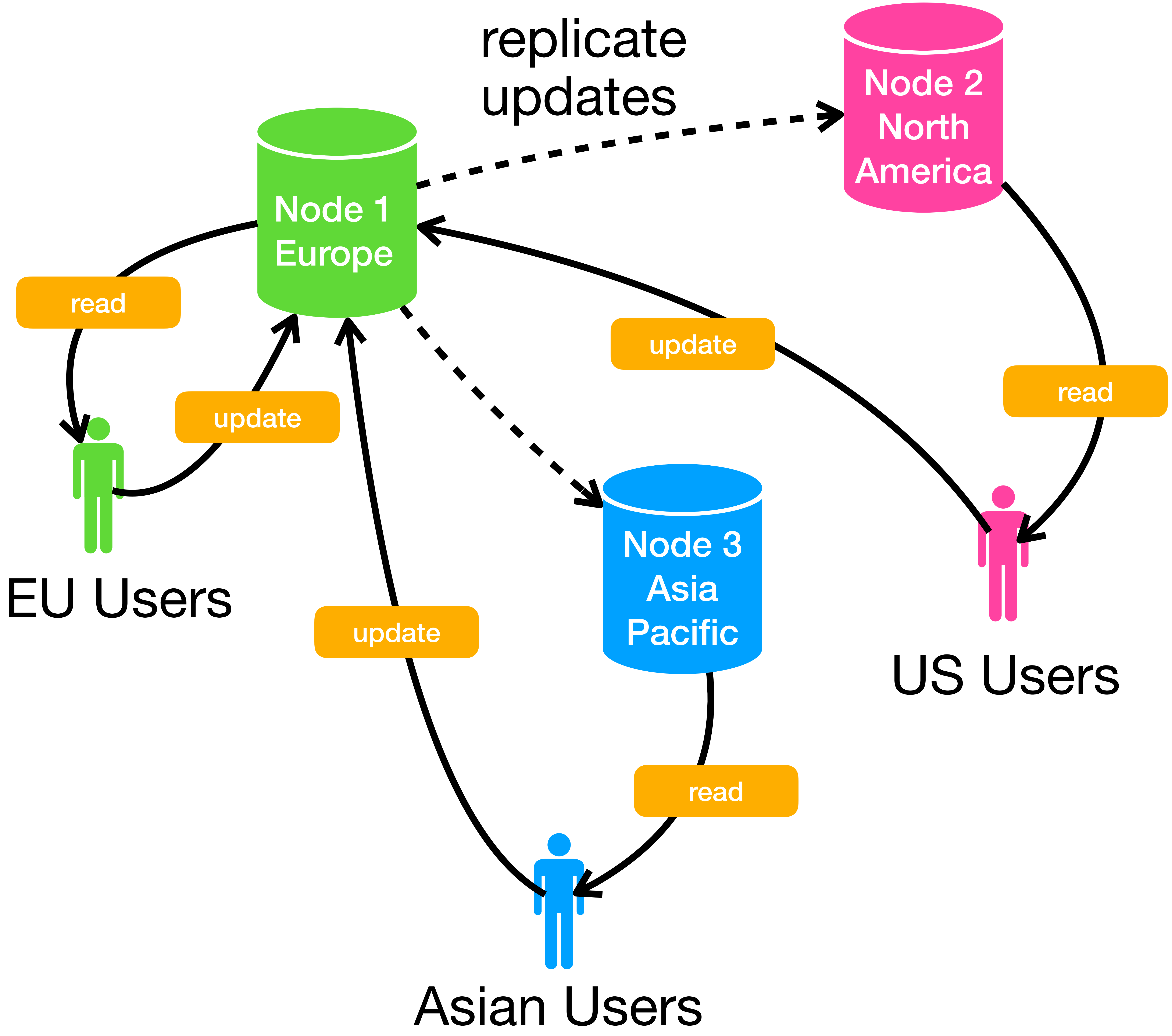
1. The network is reliable;
2. Latency is zero;
3. Bandwidth is infinite;
4. The network is secure;
5. Topology doesn't change;
6. There is one administrator;
7. Transport cost is zero;
8. The network is homogeneous;
9. We all trust each other.

Development of distributed systems has to take into account that any one or multiple of the assumptions can be false at any time.

Replication

Spatially distributed nodes

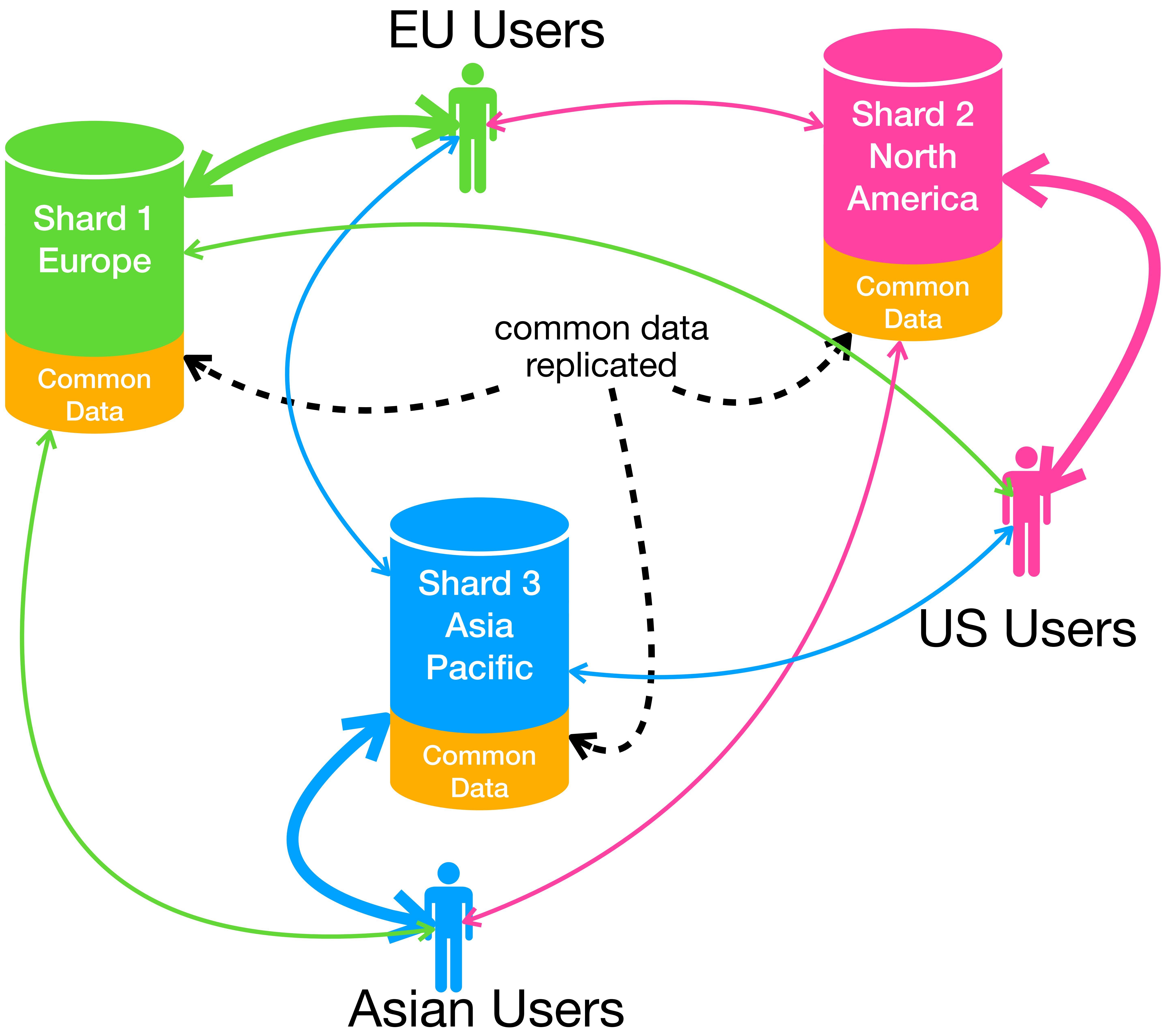
- usually, updates go through a main node
- optionally, all nodes accept updates; conflict detection and replication more complicated
- each node contains all data
- read queries handled by closest node
- child nodes can serve as backup



Sharding

When data gets too big

- nodes contain only local data and act as single source for it
- replication only for small fraction of common data that's often used across regions, usually few updates
- requests/updates for data from other regions slower, but (hopefully) less frequent



ACID, CAP, and BASE

ACID Transactions

Challenges from Concurrent Updates

- Concurrent write access by multiple clients
- Complex operations and integrity constraints that encompass many entities
- Very simple example:

  In an accounting system, the balance of all bookings has to be 0.

- Some possible sources of violations:
  - Dirty Read
  - Non-Repeatable Read
  - Lost Update

Dirty Read

User A	User B
Read B_1	
$B_1 = B_1 - 10$	
Write B_1	
	Read B_1
	$B_1 = B_1 + 20$
Read B_2	
$B_2 = B_2 + 10$	
	Read B_3
Rollback	
	$B_3 = B_3 - 20$
	Write B_1
	Write B_3
	Commit

BOOKING	NO	AMOUNT
	1	100
	2	-150
	3	200
	4	-150

OK: Balance=0 :-)

BOOKING	NO	AMOUNT
	1	110
	2	-150
	3	180
	4	-150

Inconsistent: Balance =-10 :-(

Non-Repeatable Read

User A	User B
Read B_1	
$B_1 = B_1 - 10$	
	Read B_1
	... do something...
Write B_1	
Commit	
	Read B_1
	Commit

User B reads value B_1 subsequently without changing it in between, but reads different values.

Lost Update

User A	User B
Read B_1	
$B_1 = B_1 - 10$	
	Read B_1
Write B_1	
	$B_1 = B_1 + 20$
	Write B_1
Read B_2	
$B_2 = B_2 + 10$	
Write B_2	
	Read B_3
	$B_3 = B_3 - 20$
	Write B_3
	Commit
Commit	

BOOKING	NO	AMOUNT
	1	100
	2	-150
	3	200
	4	-150

OK: Balance=0 :-)

BOOKING	NO	AMOUNT
	1	120
	2	-140
	3	180
	4	-150

Inconsistent: Balance=+10 :-(

Solution Strategies

- Single user operation (of course, often not feasible)
- Pessimistic strategy
  - Definition of critical resources, synchronization of access, e.g. by temporary locks on data
  - Read/Write locks
    multiple simultaneous reads permitted
    only a single write lock without any reads
  - Clients have to wait for locks, leads to limited concurrency
- Optimistic strategy
  - Permit concurrent modifications
  - Check for conflicts only when data is written
  - Challenges:
    Isolation of concurrent clients
    Detection of conflicts
  - Advantages:
    Database management system controls concurrency
    Higher degree of concurrency compared to locks

Transactions

- A **transaction** is a sequence of logically associated read and write operations.

- Database systems provide guarantees that either all operations of a transaction are executed, or none of them.

- During a transaction, clients get a consistent view on the data where changes of others are hidden.

- Transactions are controlled by the following operations:
  - `begin` mark the start of the operation sequence
  - `commit` try to store all changes, may succeed or fail
  - `rollback` undo all changes (e.g., on failed commit)

ACID Properties of Transactions

- Atomicity
  All (write) operations in a transaction are executed completely or not at all.

- Consistency
  Before and after execution of a transaction, all integrity constraints hold.
  This holds for both outcomes, successful `commit`, and `rollback`.

- Isolation
  Concurrent transactions are executed independently, during a transaction, changes of others are invisible.

- Durability
  Changes of successfully `committed` transactions are stored permanently, even in case of hazardous events.

Challenges

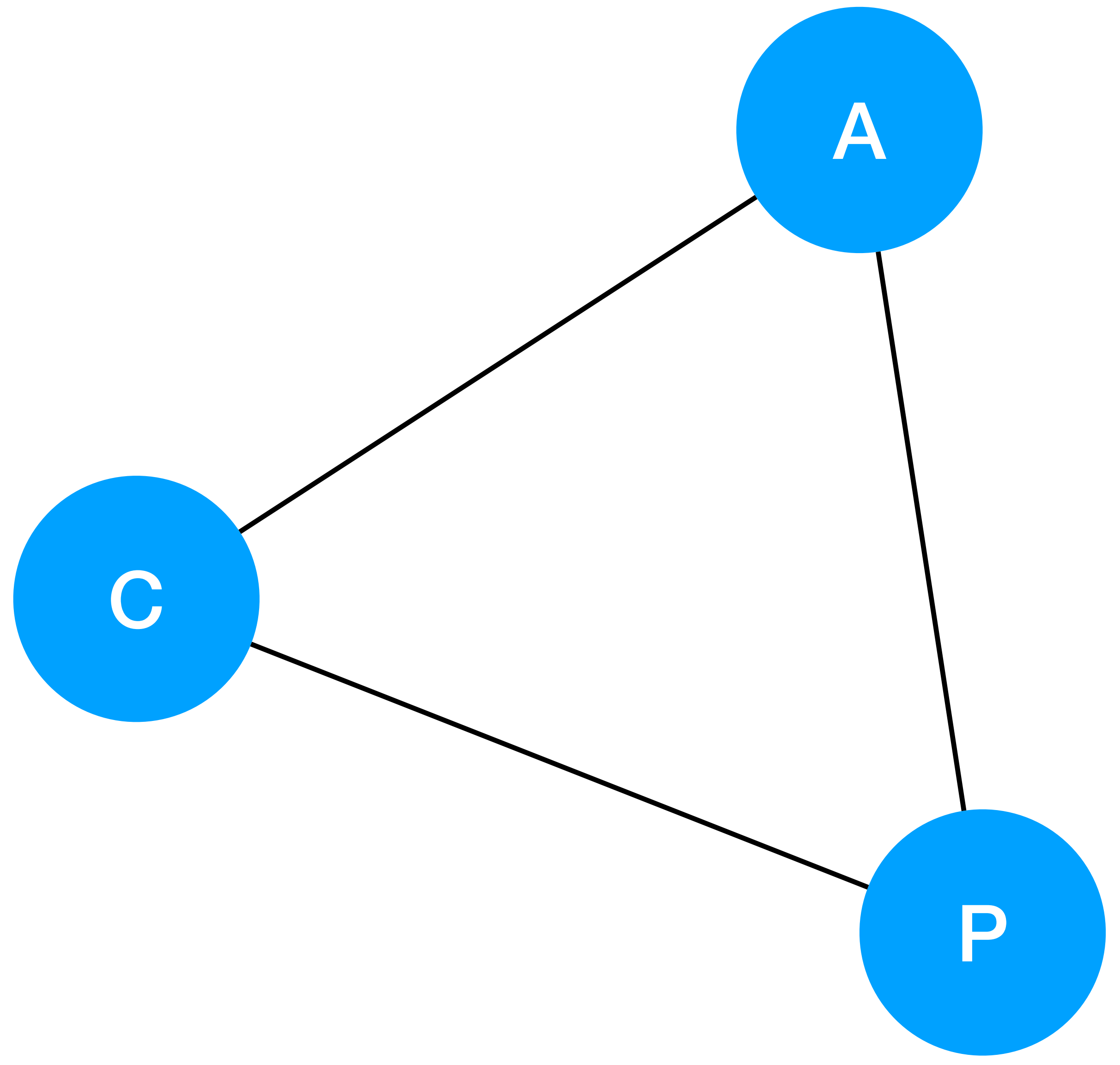
When conflicting changes are to be committed…

- What is a conflict?
- What does “consistent” mean?
- What to do if a conflict is detected?
- When does a transaction begin and end?
- What about distributed systems?
- When do we regard a system as “distributed”?

The CAP Theorem

The CAP - Theorem

- It is **impossible** for a distributed data store to **simultaneously** provide more than two out of the following three guarantees
  - Consistency: Every read receives the most recent write or an error
  - Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
  - Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



Original reference:

Armando Fox and Eric Brewer, "Harvest, Yield and Scalable Tolerant Systems", *Proc. 7th Workshop Hot Topics in Operating Systems (HotOS 99)*, IEEE CS, 1999, pg. 174–178

Heads up: same terms with specific meaning

Consistency

- “C” in ACID Transactions
- Means that before and after each transaction all database and domain constraints hold
- Independent of transaction success or fail

- “C” in CAP
- Means that all nodes in a distributed data store (e.g., in a set of replication nodes) have the same data, and hence give the same answers to queries
- Every read receives the most recent write or an error

Partition

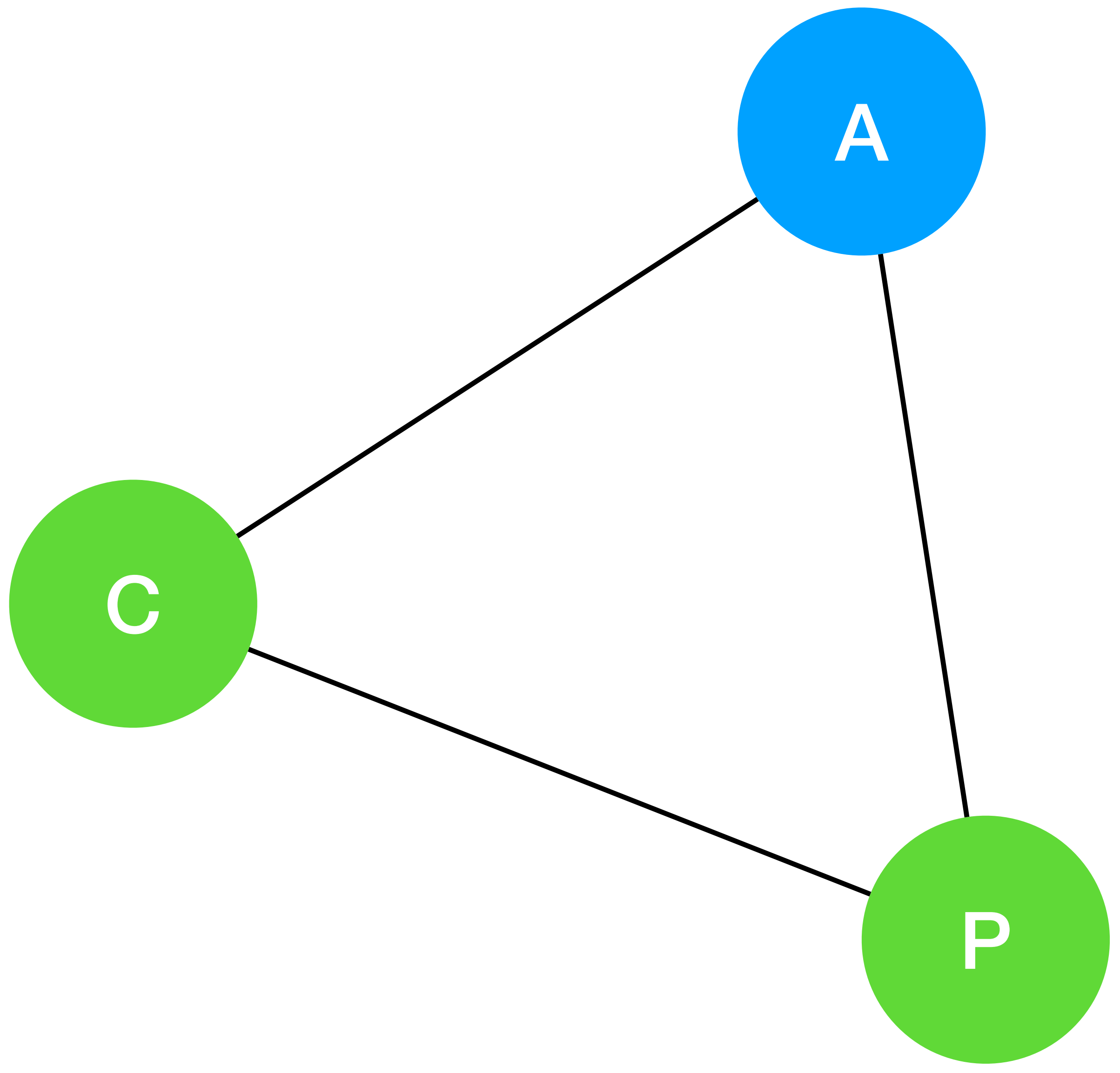
- Partition in databases
- Way to split data (e.g. relational tables) based on data properties to minimize lock conflicts and/or to enable parallel processing

- Partition in CAP
- Means that data is distributed over several nodes connected via potentially unreliable connections. Either the network or individual nodes can fail and become unavailable

CAP - Theorem

C-P System: Forfeiting Availability

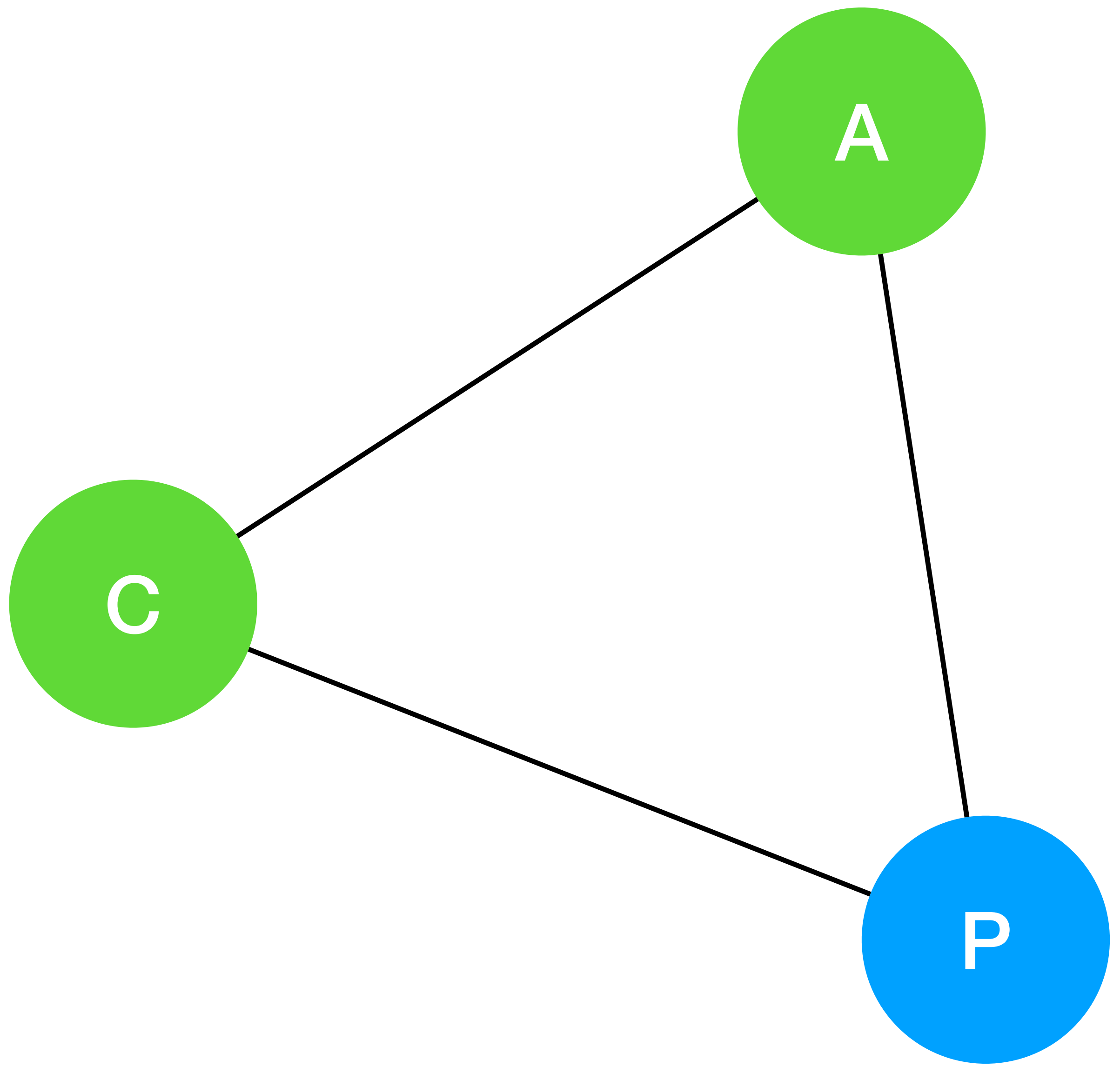
- Rather than continuing to provide the service in case of partitioning, consistency is favored.
- That can mean that (some) operations might be unavailable.
  - Reads might still work, depending on the actual data location
  - Writes are prohibited
  - Rather than providing a result, an error is reported



CAP - Theorem

C-A System: Forfeiting Partition Tolerance

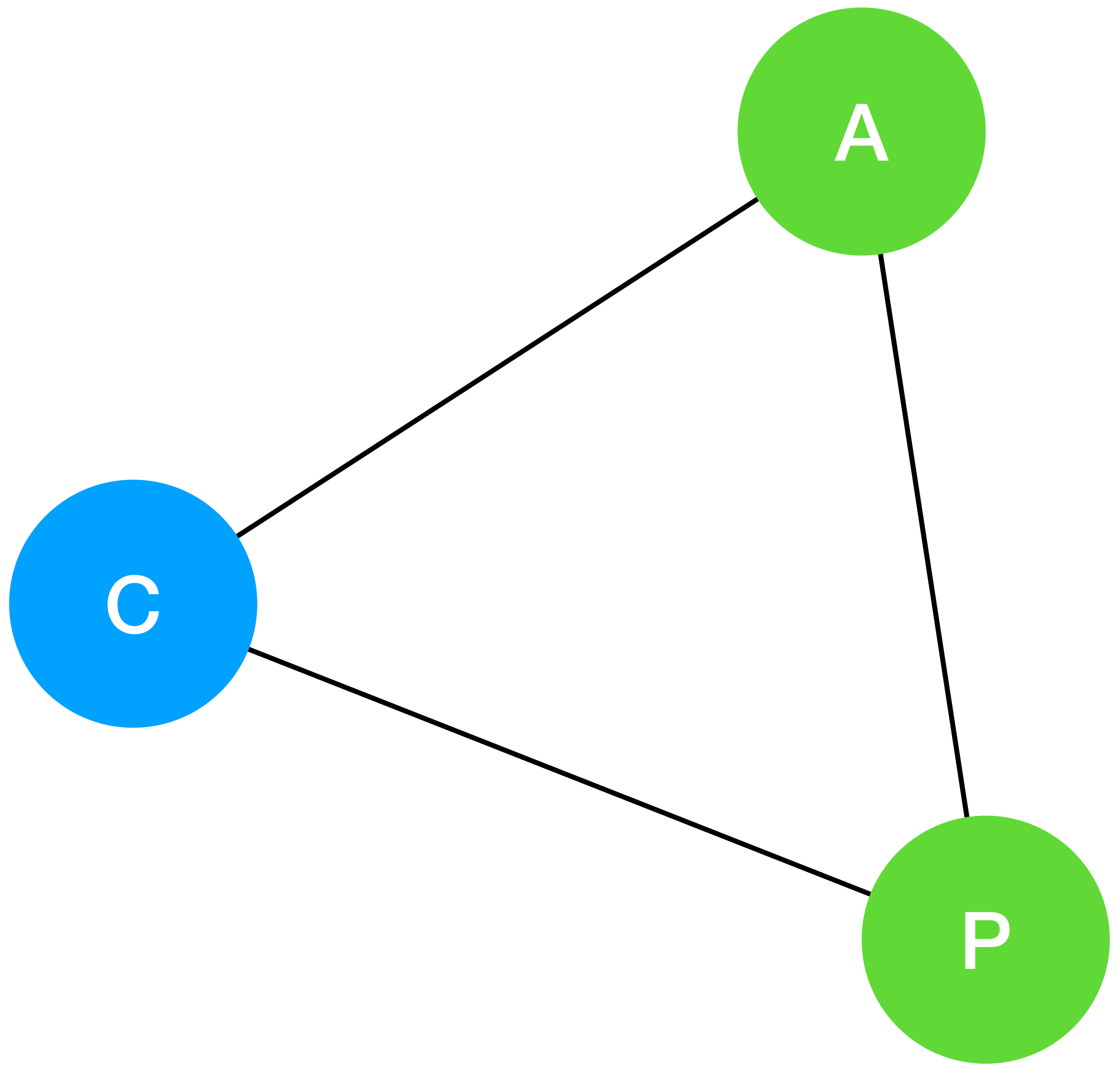
- Only possible when the system is not distributed, since failures can occur at any time.
- Even with a single nodes, this node may still fail and become unavailable.



CAP - Theorem

A-P System: Forfeiting Consistency

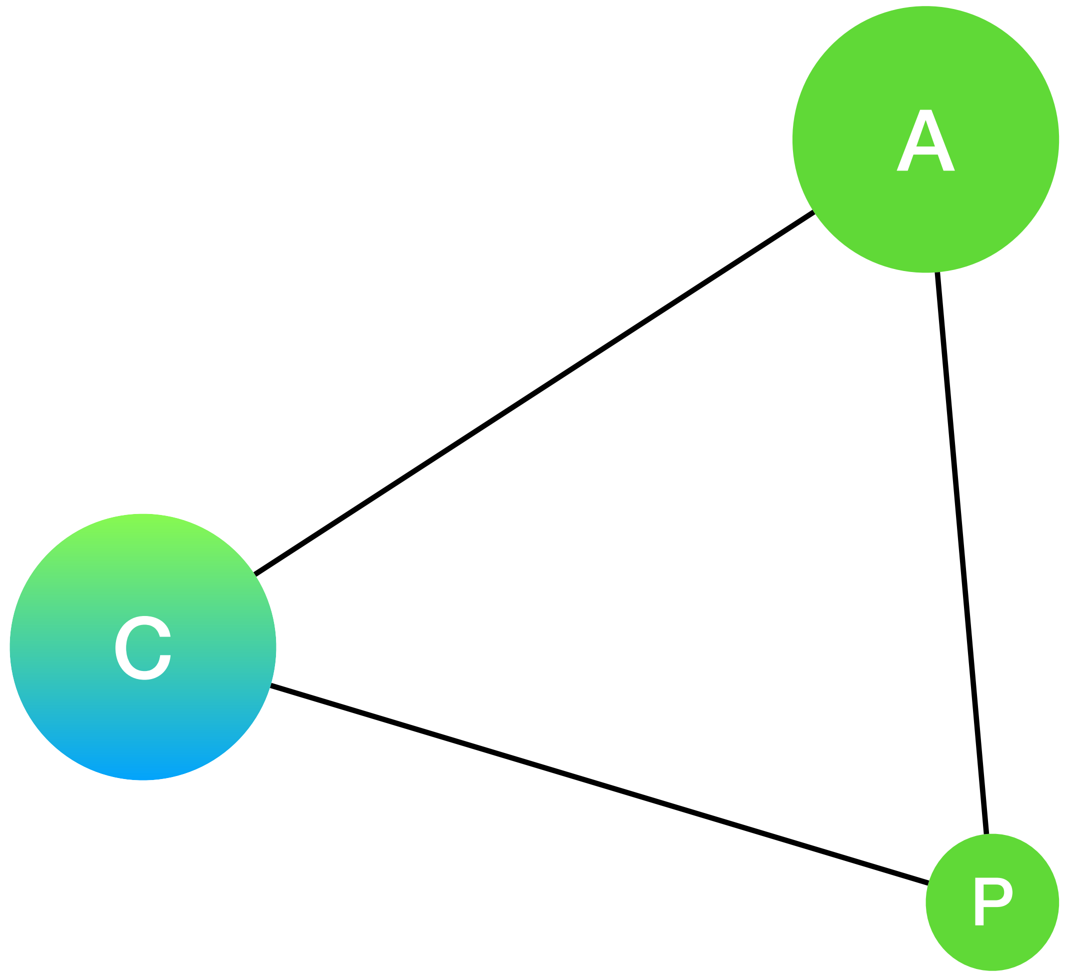
- The system provides (non error) responses even in case of unavailable nodes or dropped messages.
- The responses might not contain the latest changes and hence give a “false” result.
- Write operations are still provided, but can lead to inconsistent nodes.



BASE

Basically Available, Soft State, Eventually Consistent

- More useful/realistic variant of an A-P system.
- Focus on high availability, taking into account that partitioning can occur but is rather unlikely
- Soft state means that a node might not know the overall state at any time, there's a only a certain probability that the state is known
- The responses might not contain the latest changes and hence give a "false" result.
- Write operations are still provided at any node at any time, but can lead to inconsistent nodes.
- When connectivity is restored, nodes eventually receive delayed writes (optimistic replication)
- Conflicts can occur and must be handled



Recommended Reading:
[CAP Twelve Years Later: How the "Rules" Have Changed]
(visited 2021/02/01)

What we have learned…

Persistence (Part III)

✓ Document Databases

✓ Data Mapping to Documents

✓ Distributed Data Storage

✓ Data Intensive Systems

✓ ACID, CAP and BASE

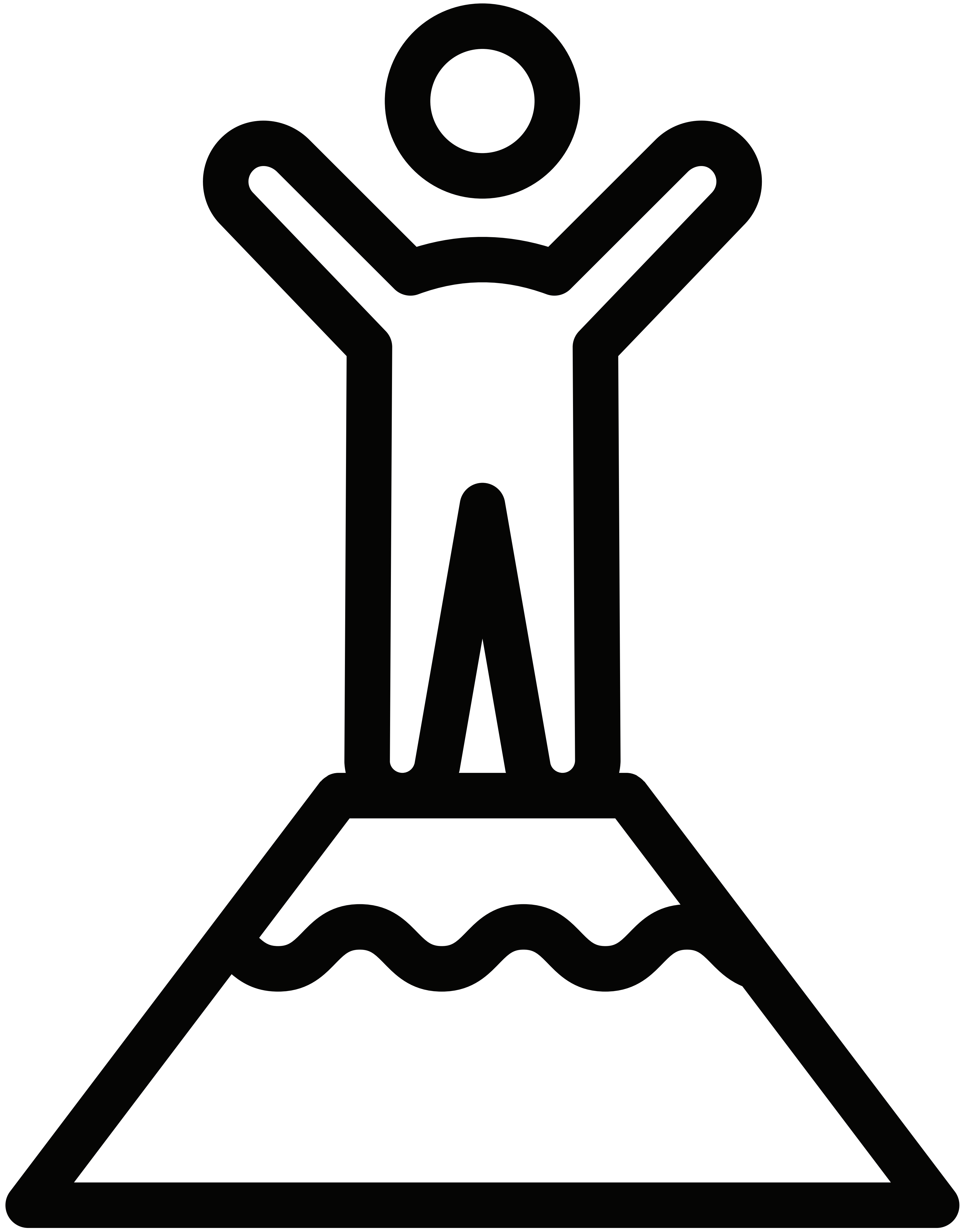


Image: colourbox.de