TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software-Engineering Project, Summer 2017

Deepthi Akkoorath, Annette Bieniusa, Mathias Weber, Peter Zeller

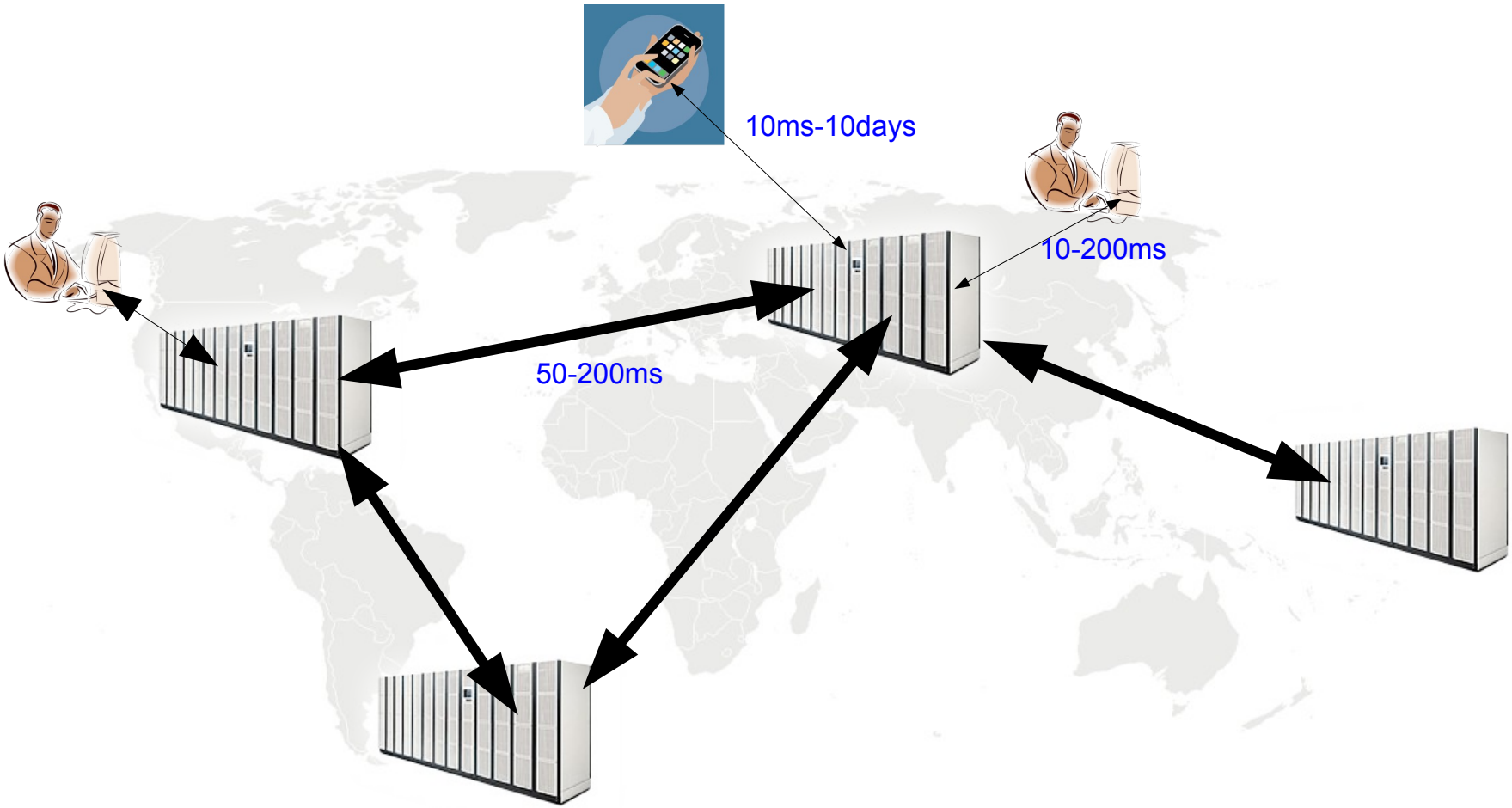
AG Softech
FB Informatik
TU Kaiserslautern

Participants

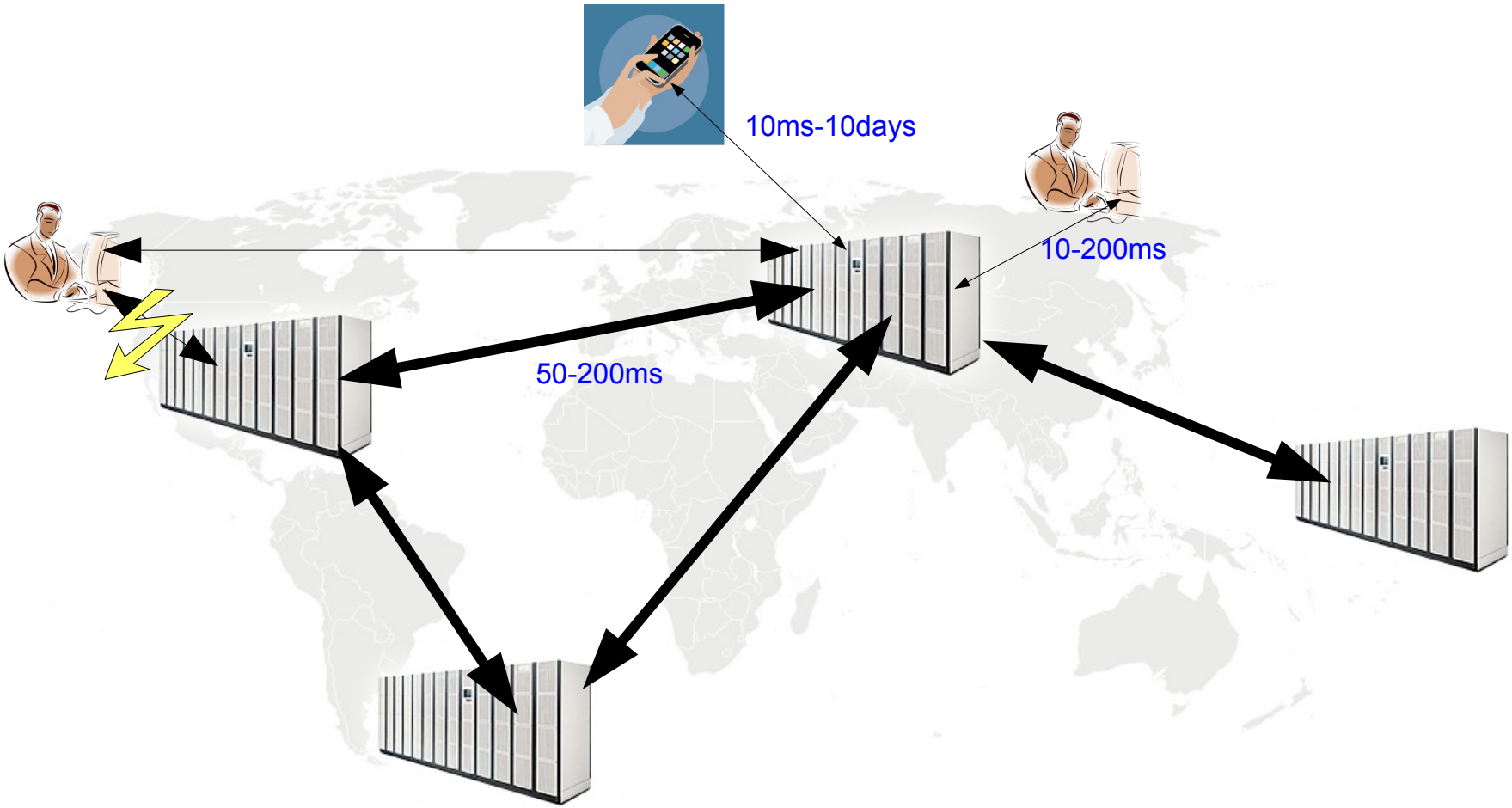
1. Farooq Arshad
2. Vishnu Vardhan Sundarrajan
3. Alka Scaria
4. Kevin Bartik
5. Rasha Abo Qasem
6. Tchangou Maxime
7. Frikha Yassine
8. Huzaifa Sabir
9. Revathi Balasubramanian
10. Samitha Jayathilake
11. ...
12. ...
13. ...
14. ...
15. ...

Workload: 8CP = 200-240h
over 14 weeks = 14-17h/week

Replication – Delays

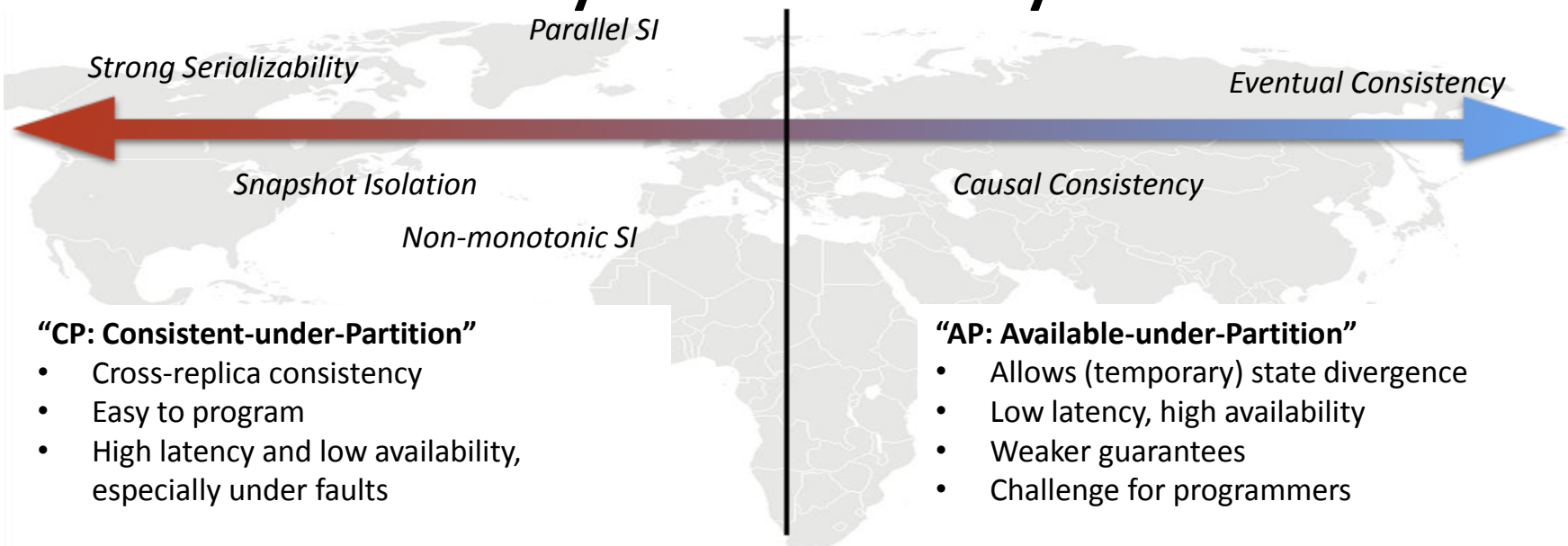


Replication – Delays



CAP-Theorem

Consistency-Availability trade-off



"CP: Consistent-under-Partition"

- Cross-replica consistency
- Easy to program
- High latency and low availability, especially under faults

"AP: Available-under-Partition"

- Allows (temporary) state divergence
- Low latency, high availability
- Weaker guarantees
- Challenge for programmers

CAP

Antidote

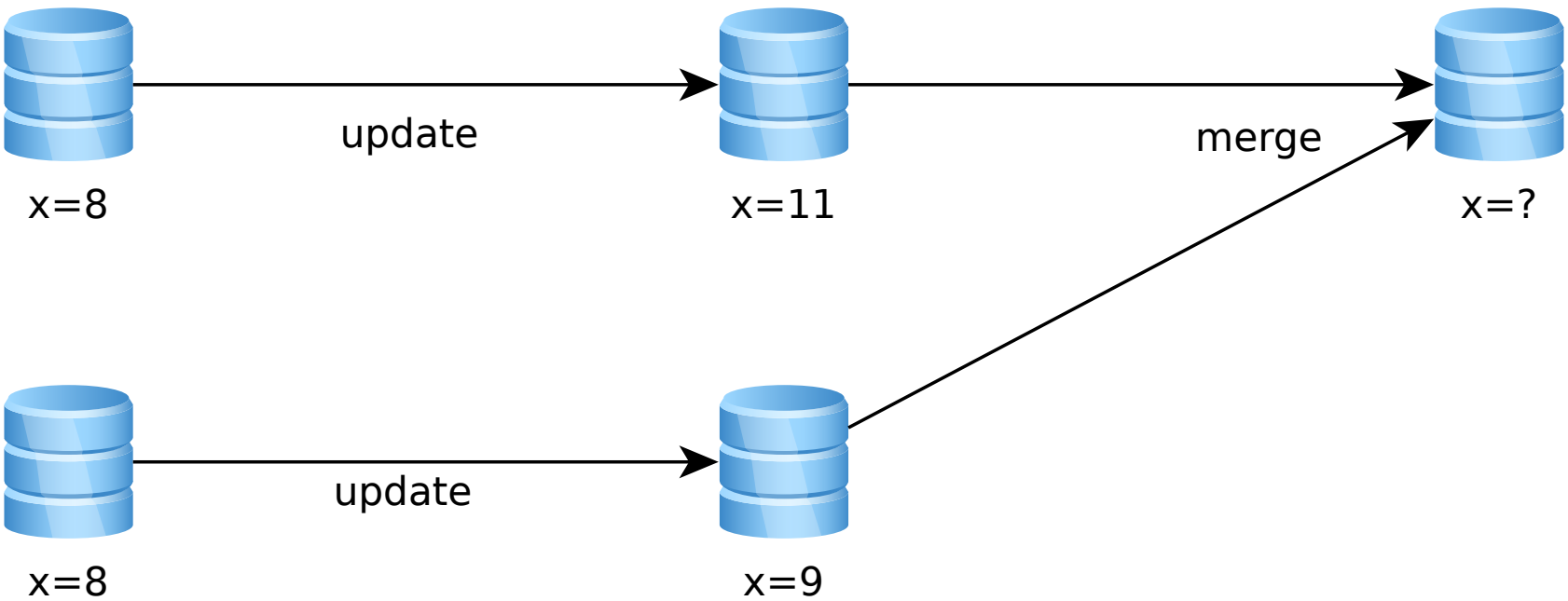
A planet-scale, available, transactional database with strong semantics

Traditional databases provide strong guarantees but are **slow** and **unavailable under failures and network partition**. Hence, they are not suitable for geo-replication.
The alternatives are NoSQL-style databases which are **fast and available even under network partition**. They provide a low-level key-value interface and expose data inconsistencies due to asynchronous communication among the servers. It takes significant effort and expertise from programmers to deal with these inconsistencies and develop correct applications on top of these databases.

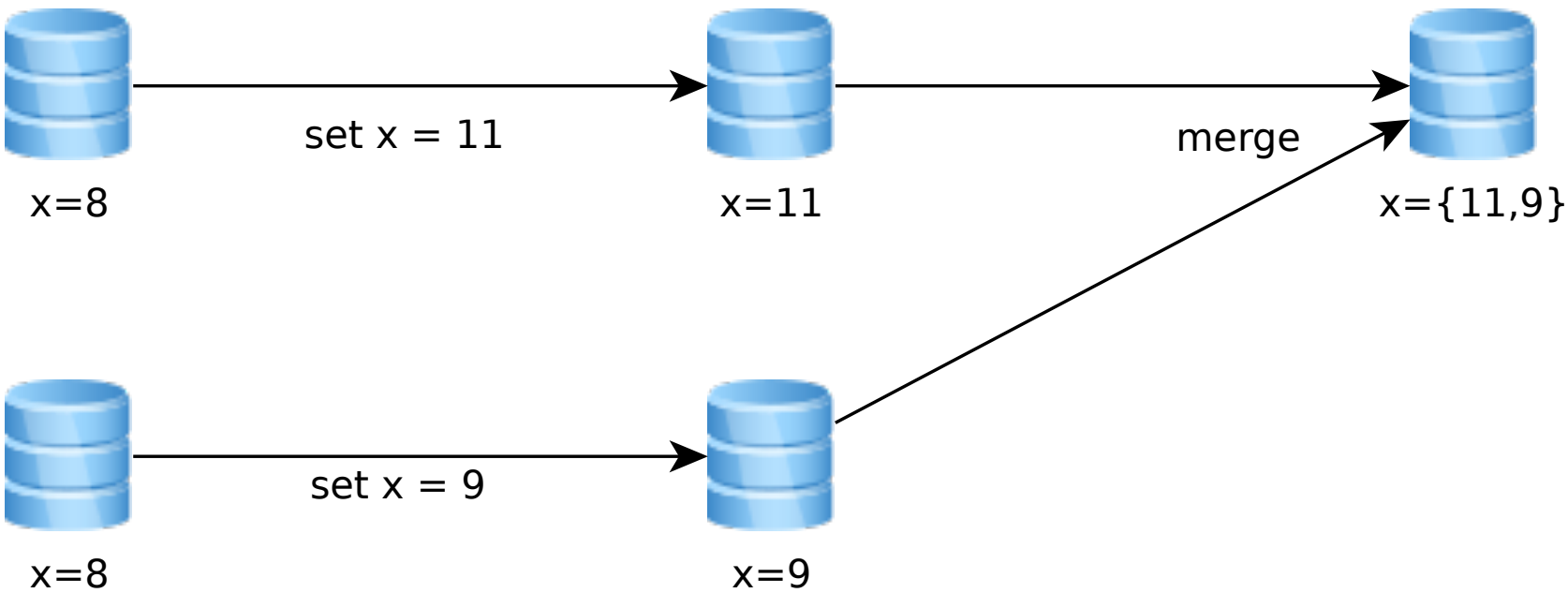
https://syncfree.github.io/antidote/

- Cloud-scalable CRDT database
  - High performance:
    exploit parallelism via sharding
  - Widely geo-replicated
- Supports transactional causal+ consistency
  - Atomic update
  - Consistent snapshot
- Written in Erlang with client drivers for several languages
  - Erlang, JavaScript, Java, Haskell

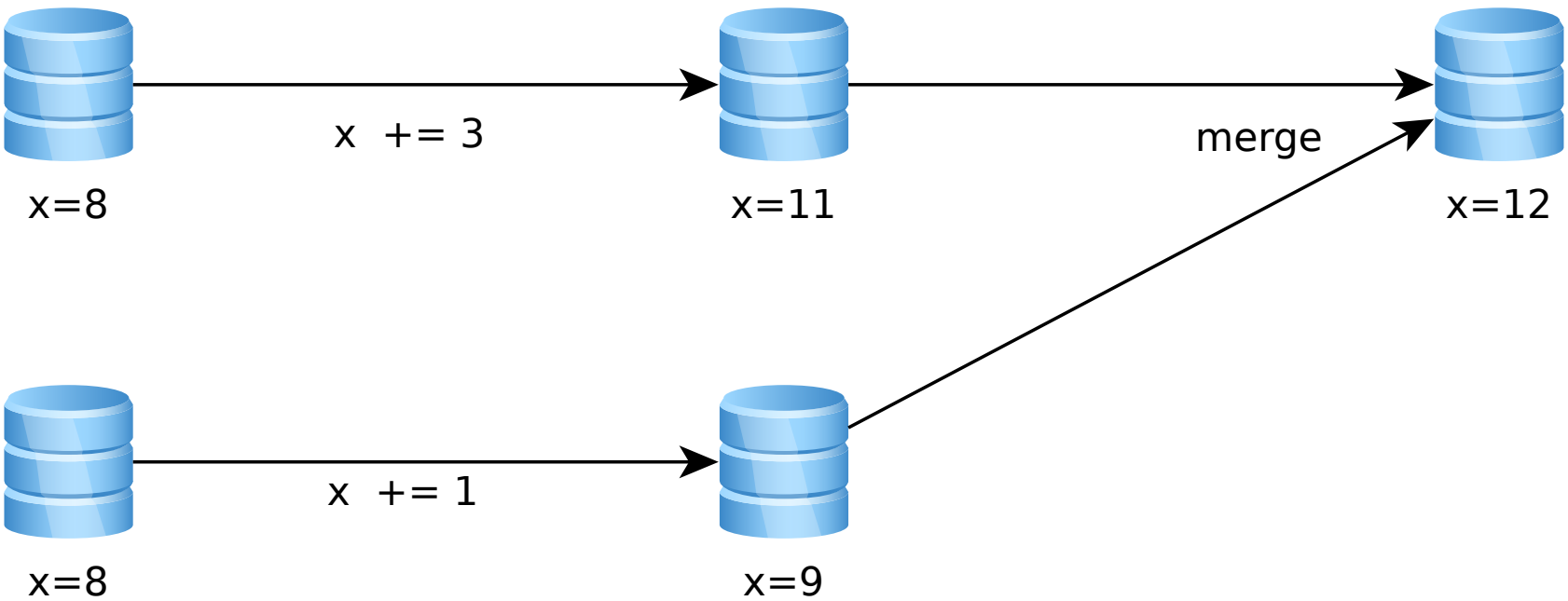
Eventual Consistency



Multi-Value Register



Counter



CRDTs

(Conflict-free/Commutative/Convergent) Replicated Data Types

- Contain data
- Have a built-in strategy to handle concurrent updates

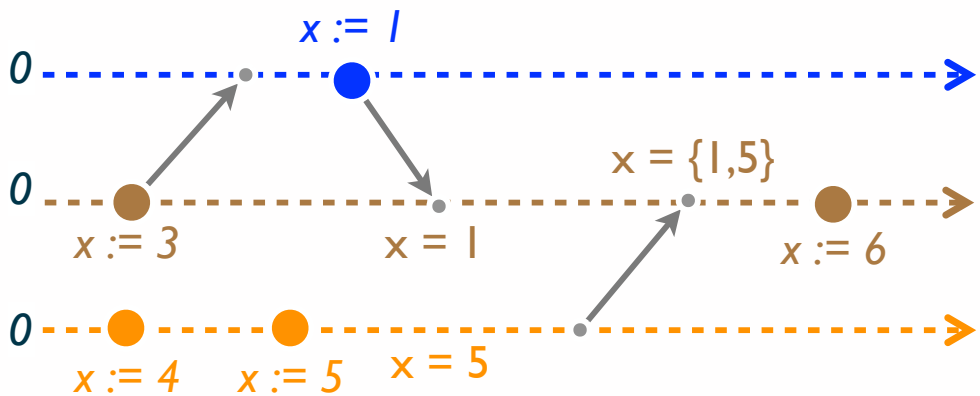
Data types in Antidote:

- Last-writer-wins Register (LWW-Register)
- Multi-value Register (MV-Register)
- Number Types (Counter, Fat Counter, Integer)
- Flags (enable-wins and disable-wins)
- Maps (Grow-only Map, Add-wins Map, Remove-Resets Map)
- Sets (Grow-only Set, Add-wins Set, Remove-wins Set)

Topics

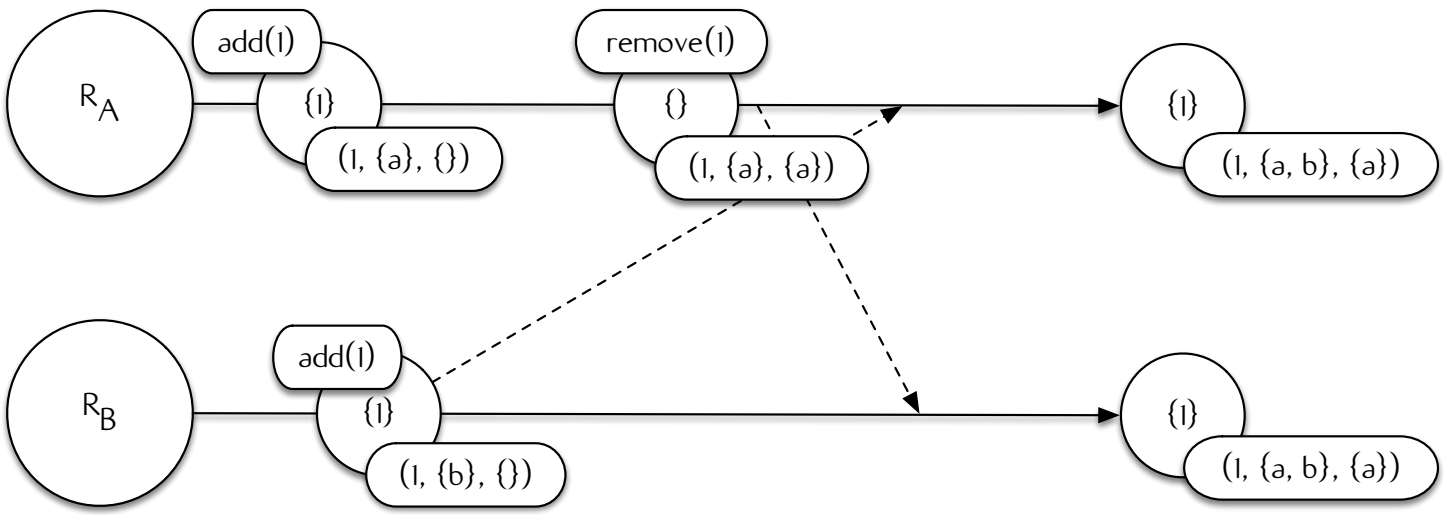
1. An interactive tool for demonstrating CRDTs
2. Antidote Jupyter Notebook
3. Managing a causally consistent materializer
4. A Performance Benchmark for Antidote

An interactive tool for demonstrating CRDTs



- Interactive visualization of CRDT executions
- Must run in the browser
- Used for Antidote documentation (embed in Markdown document)

An interactive tool for demonstrating CRDTs



Antidote Jupyter Notebook

Example:

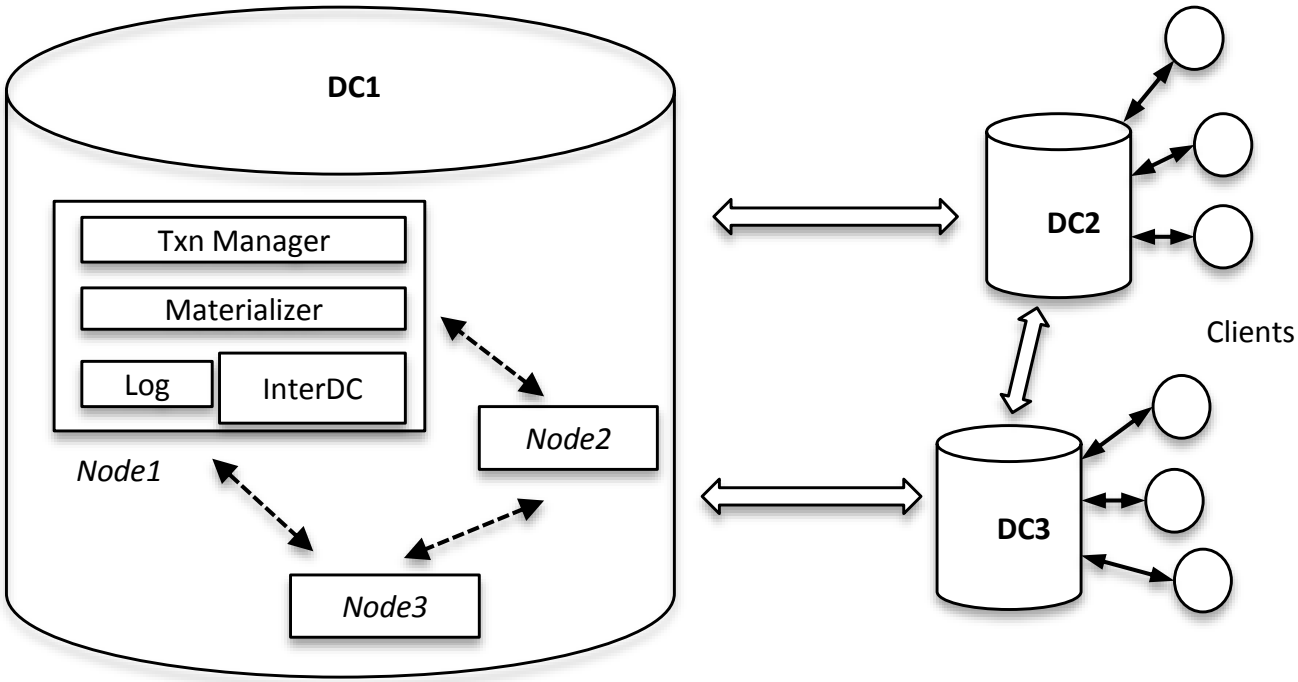
```
docker run -it --rm -p 8888:8888 jupyter/datascience-notebook
```

Goal: Writing interactive Antidote tutorials using Jupyter Notebook

Controlling Datacenters Users should see, how Antidote behaves in the case of partitions and how concurrent updates are resolved by Antidote. Therefore users should be able to artificially disconnect data centers and reconnect them again with commands in the Jupyter notebook.

Deployment It should be easy to run a notebook locally. Ideally users just have to run `docker-compose up` to get the notebook running in Docker containers.

Managing a causally consistent materializer



The persistent storage in Antidote is an '**append-only log**' that records each update operation that is executed. A materialized view, called **snapshot**, of a key can be generated by applying all updates to that key, starting from an initial snapshot. The materialized view is generated by a component called **Materializer**.

Managing a causally consistent materializer

Goal: Implement prototypes for efficient Materializers
Challenges:

- Several versions can be requested
- Operations have to be applied in right order
- No sequential order of operations

Implementation

- in Java or C/C++

A Performance Benchmark for Antidote

Goals:

- Building a comprehensive benchmark suite for Antidote

Implementation:

- in Java or Erlang

Use cases:

- Comparing performance of pull request to master
- Monitoring performance development over time.
  For example: `https://arewefastyet.com/`

Topic-Assignment

1. An interactive tool for demonstrating CRDTs
   3-4 students
2. Antidote Jupyter Notebook
   2-3 students
3. Managing a causally consistent materializer
   2*3 students
4. A Performance Benchmark for Antidote
   3-5 students

First Steps - next 2 weeks

- Clarify requirements with customer
  - Requirements document
  - Prioritize requirements
- Create a plan for the project
- Research, get familiar with technology