Kiel University
Christian-Albrechts-Universität zu Kiel

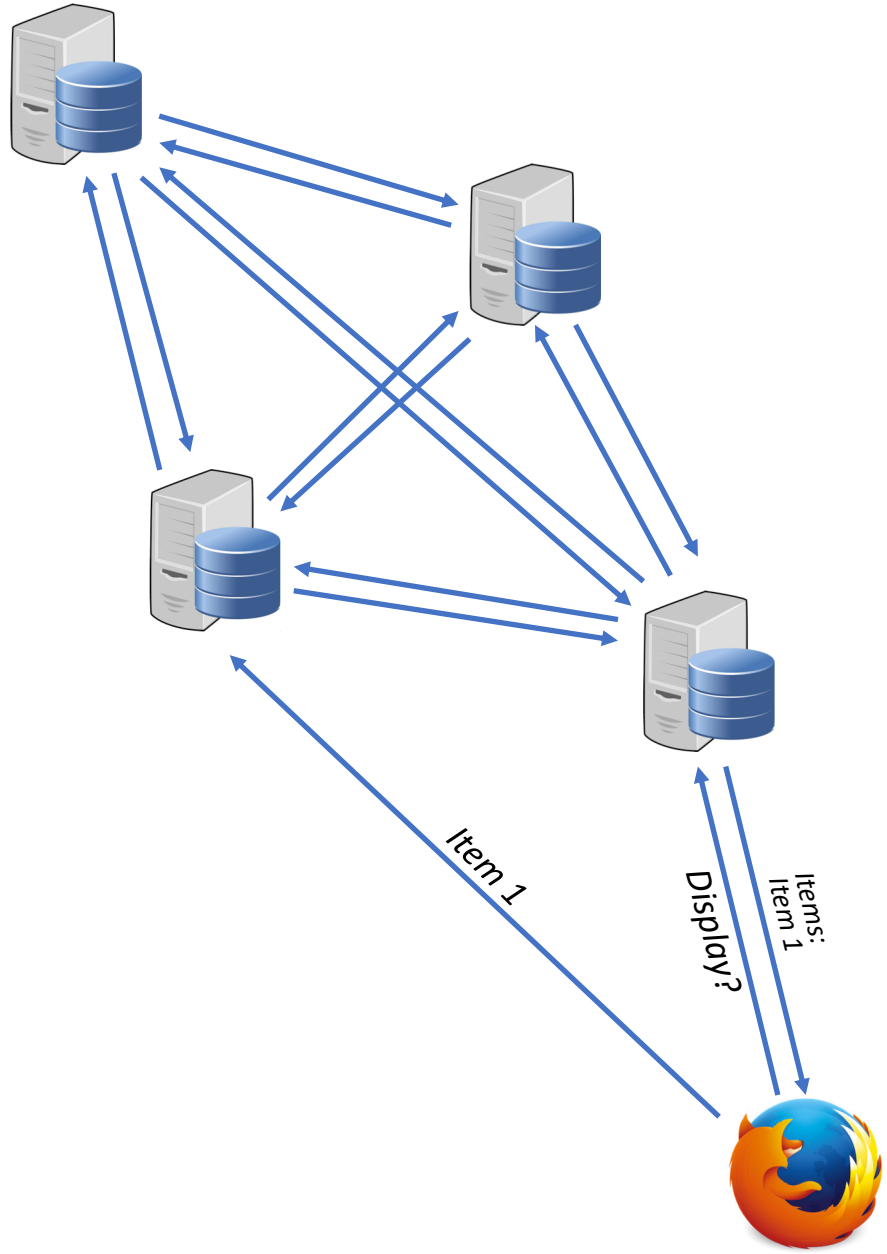
Distributed Systems
Research Group

Lab 3

Eventual Consistency and Vector Clocks

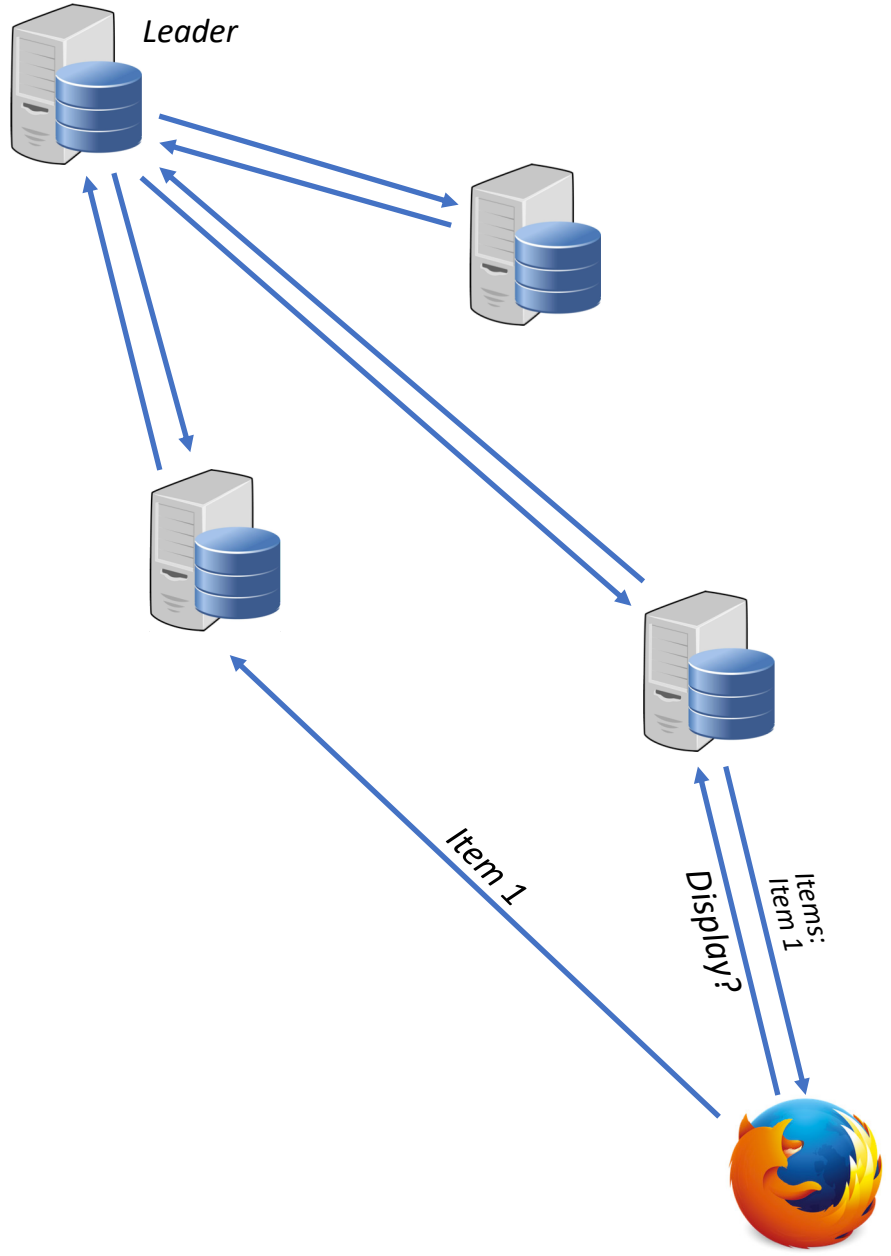
Wrap-up lab 1: An Overly Simple Distributed Blackboard?

- Blackboard *is* distributed
- But might get inconsistent!
- Consistent blackboard:
  - All entries are the same
  - Messages shown in the same order
- How can we have a consistent blackboard?
  - Let's try a centralized solution!



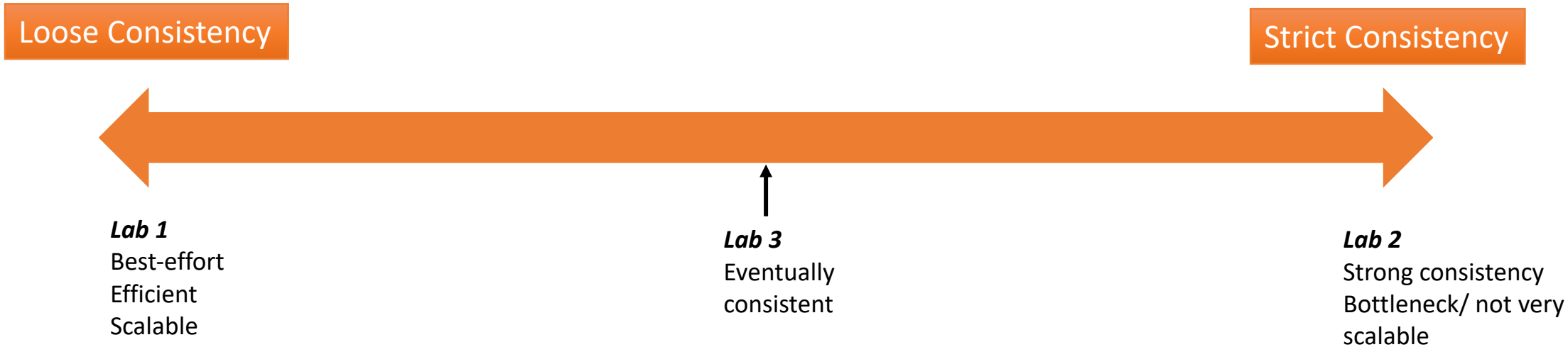
Wrap-up lab 2: Strong consistency and bottleneck

- Blackboard is centralized and strongly consistent
  - All entries are the same
  - Messages shown in the same order
- Disadvantages?
  - Time to recover upon leader failure
  - Leader becomes a bottleneck as system grows
  - Not very scalable?
- How to build a consistent, scalable system?
  - Let's move to a looser definition of consistency



Consistency tradeoff

- Balance between strictness of consistency and efficiency/scalability
  - How “much” consistency we need depends on the application
- Lab 3 will be somewhere in the middle
  - Messages might temporarily appear out of order
  - But will eventually (= after some time) be consistent (same order)



Eventual Consistency

- All replicas ***eventually converge*** to the same value

- A possible protocol for eventual consistency:
  - Writes are eventually applied in total order
    - Either with a history log (rollback to last correct value and reapply log)
    - Or with an ordered queue (apply write only if all previous writes were applied)
    - Or by considering only the last write (if history has no impact)
    - Eventually all writes are received and the value has converged
  - Reads might not see the most recent writes in total order
    - Value might be based on incomplete history log or outdated value

- Used in many applications like Google File System (GFS) and Facebook's Cassandra
- For more information, see lecture slides

Design considerations

- In *eventually consistent* (large) data-stores
  - Write-write conflicts are rare
  - But they are frequent in **our** system!
  - In real-life systems
    - 2 processes writing the same data concurrently is a rare case
    - It can be handled through simple mutual exclusion
- Read-write conflicts are more frequent
  - 1 process reading a value while another process is writing over this value
  - *Eventually consistent* solutions should efficiently resolve such conflicts
    - Based on the needs of the application

Requirements

- ***Distributed*** blackboard
- No leader/central entity
- No ring topology
  - Peer-to-peer communication, like lab 1
- **No global time available!**

Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Lab 3 -Tasks

What you NEED to implement

Task 1 – Eventually consistent blackboard

- Implement a leaderless blackboard that is *eventually consistent*
  - Add, modify and delete functions are present and working
- Choose between:
  - Choice 1: use logical clocks
  - Choice 2: use vector clocks
- Argue your choice, its pros & cons
- The logical time of each post should be displayed on the board

Example 1 – with logical clocks

- 3 servers A, B, and C
  - Each with their clock T_A,T_B,T_C
- Client sends a *message* on server A
  - T_A=1,T_B=0,T_C=0
- Server A propagates to B and C
  - T_A=1,T_B=2,T_C=2
- Client modifies its message on server B
  - T_A=1,T_B=3,T_C=2
- Server B propagates to A and C
  - T_A=4,T_B=3,T_C=4

Example 1 – with vector clocks

- 3 servers A, B, and C
  - Each with their clock T_A,T_B,T_C
- Client sends a *message* on server A
  - T_A=[1,0,0],T_B=[0,0,0],T_C=[0,0,0]
- Server A propagates to B and C
  - T_A=[1,0,0],T_B=[1,1,0],T_C=[1,0,1]
- Client modifies its message on server B
  - T_A=[1,0,0],T_B=[1,2,0],T_C=[1,0,1]
- Server B propagates to A and C
  - T_A=[2,2,0],T_B=[1,2,0],T_C=[1,2,2]

Example 2 – with logical clocks

- 3 servers A, B, and C
  - Each with their clock T_A,T_B,T_C
- Client 1 sends a *message* on server A
  - T_A=1,T_B=0,T_C=0
- Server A propagates to B and C
  - T_A=1,T_B=2,T_C=2
- Client 2 modifies on B & Client 3 modifies on C concurrently!
  - T_A=1,T_B=3,T_C=3
- Server B propagates to A and C & C propagates to A and B
  - How to deal with this case?
  - Causality is maintained, but how to ensure consistency?

Task 2 – Concurrent & Causal modifications

- Show that the causality of two (non-concurrent) Modify events is preserved in your system

- Show the non-causality of two (concurrent) modify events, and explain how your system deals with concurrency

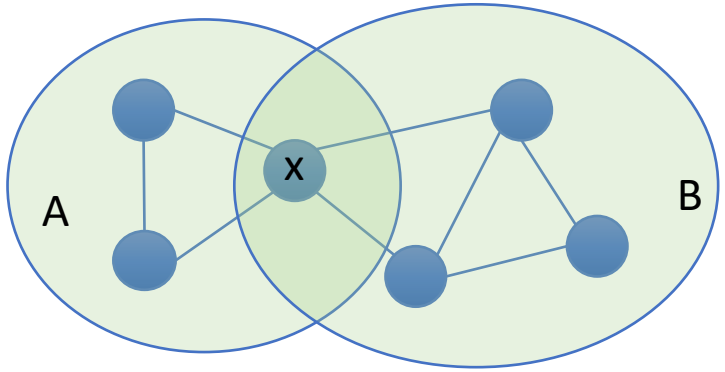
Task 3 – Network Segmentation

- Show that your system is eventually consistent in the presence of **network segmentation**

- What is network segmentation?
  - Some links are inactive in such a way that your network is now composed of smaller, disjoint networks
  - Topology is not all-to-all anymore
  - See next slide

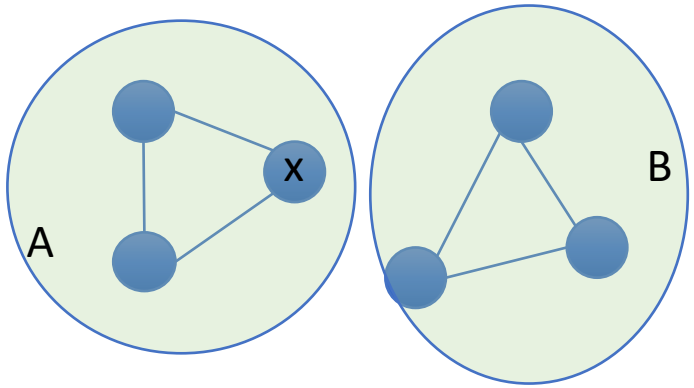
Network segmentation – an example

- Phase 1



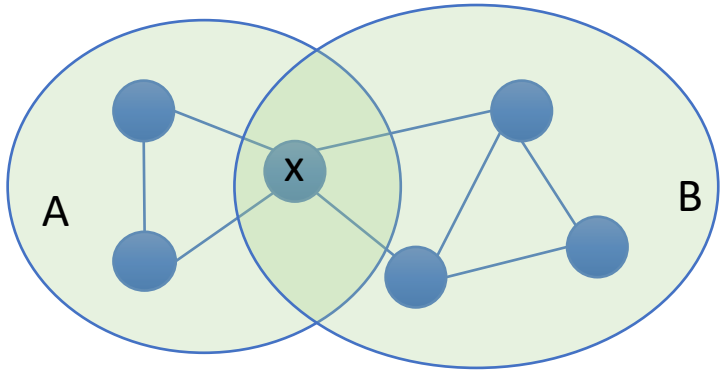
The network is a single segment. Node X communicates with both groups.

- Phase 2



Node X stops communicating with group B. Now there is consistency only within each group.

- Phase 3



Node X starts communicating with group B again. Groups A and B will have inconsistent blackboards unless they exchange their history.

Network segmentation in mininet

- You can use the two_clusters_topology.py script
- It creates 8 servers separated into two clusters
  - Servers 1 to 4 are connected to switch 0
  - Servers 5 to 8 are connected to switch 1
  - Switch 0 and 1 are connected together

- Disconnect the two switches
  - In mininet ”link s0 s1 down”
  - Link s0 s1 up to reconnect

- Firefox will not have access to the second cluster
  - Run a script from a machine (xterm server5, then run your script)

Task 4 – Performance evaluation

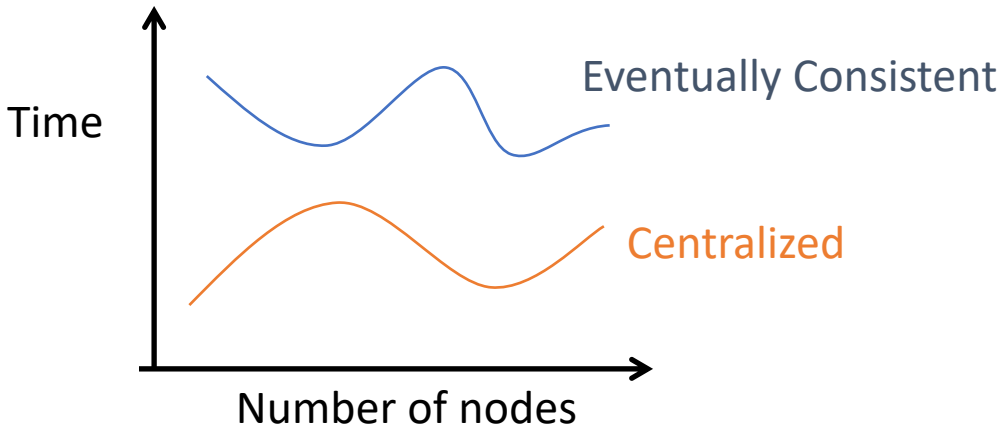
- A. Evaluate the time required to reach consistency
- B. Compare it to the performance of Lab 2
- C. Evaluate the time it takes to merge two networks after segmentation

Hints

- Your script should look like this:

```
for i in `seq 1 10`; do
        for ip in `seq 1 8`; do
                curl -d 'entry=t'${i} -X 'POST' 'http://10.1.0.${ip}:80/board' &
        done
done
```

- Your plot should (not) look like this:



Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Lab 3 - deliverables

What you NEED to give/show us

Deliverables

- Your implementation
  - Including any script you used to test consistency!
  - Upload it on iLearn

- A video of your work
  - ~ 5 min (max 10)
  - Upload on the CAU-cloud
    - Everybody will be able to watch your video!
  - Should include (at least) Design – Demo – Evaluation
    - see Lab presentations for details

Deadline:
Wednesday 13th, January

Good luck, and start coding!