Kiel University
Christian-Albrechts-Universität zu Kiel

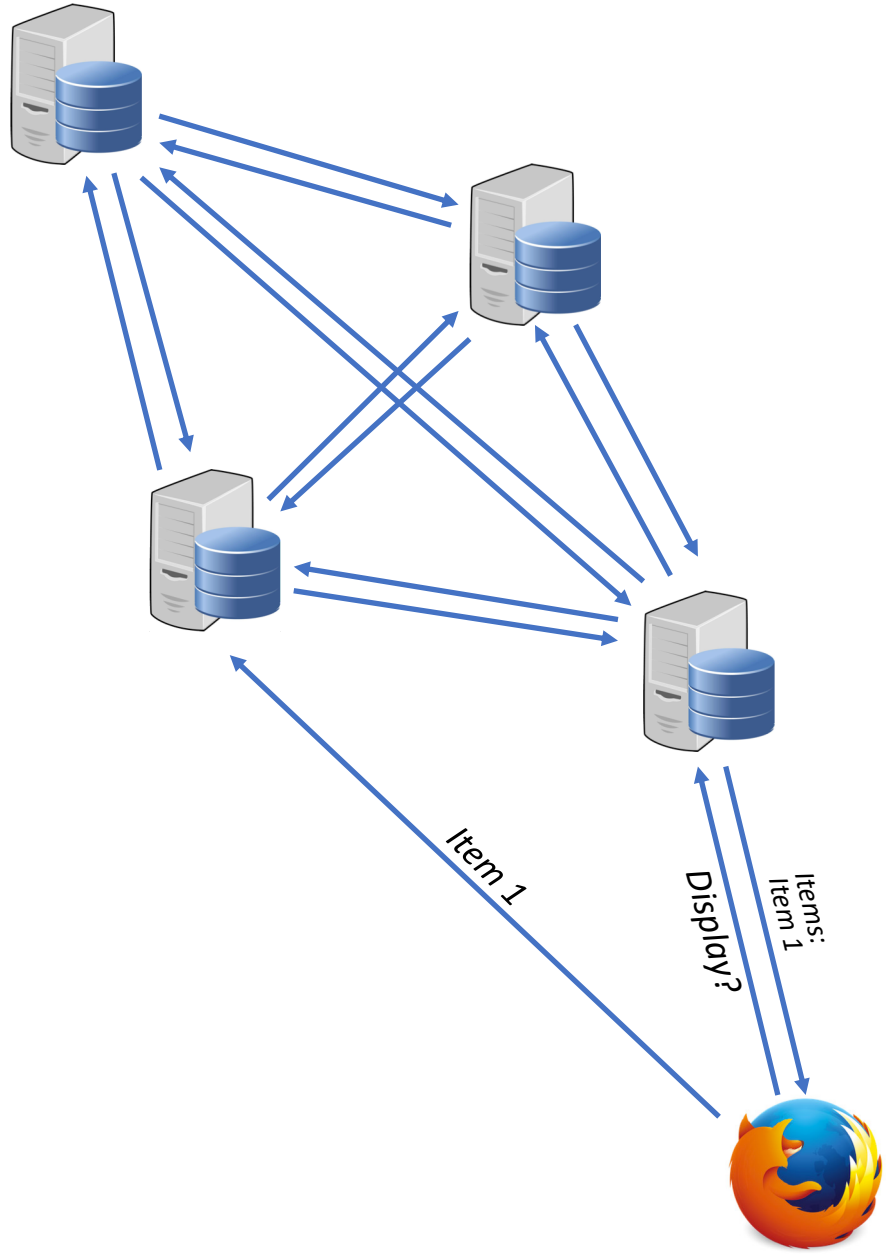
Research Group
Distributed Systems

Lab 3

Eventual Consistency and Vector Clocks

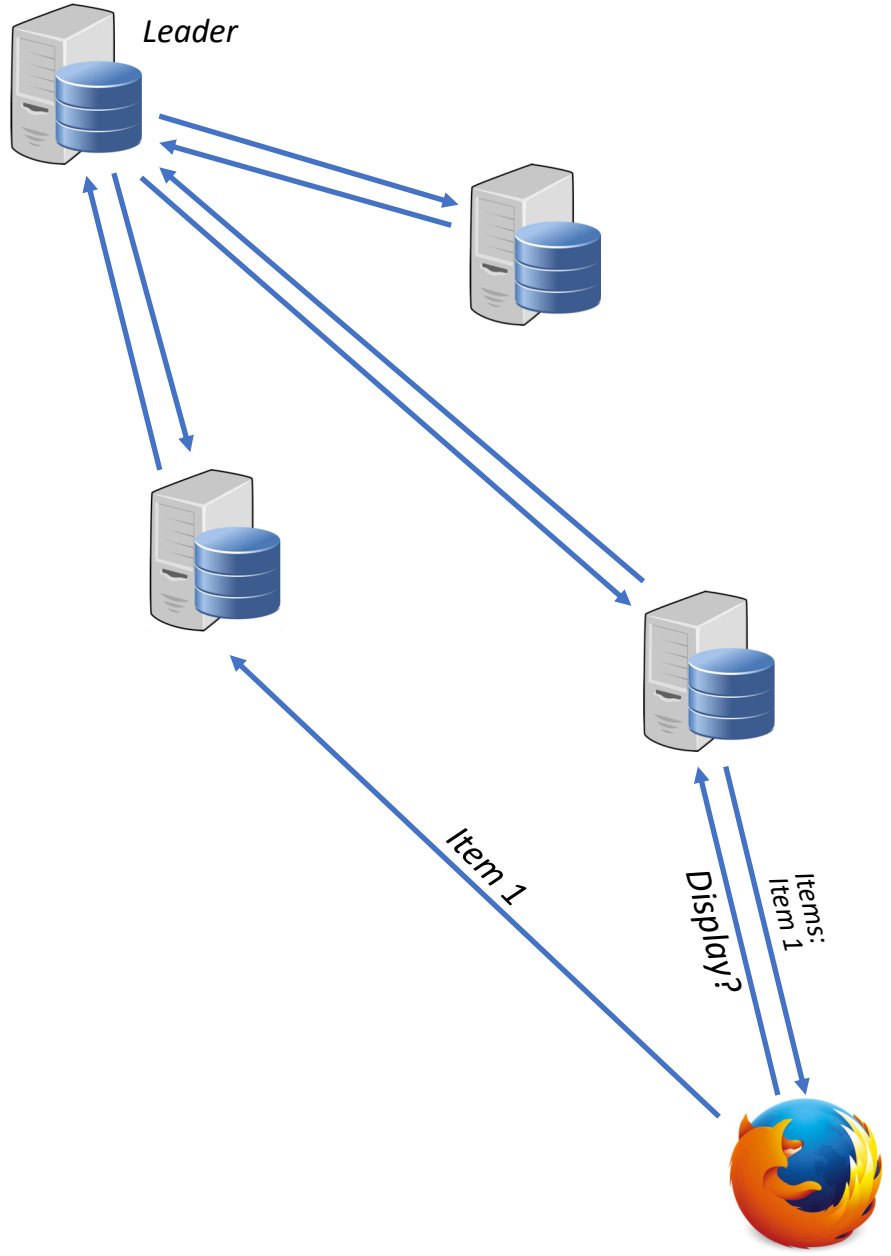
Wrap-up lab 1: An Overly Simple Distributed Blackboard?

- 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!



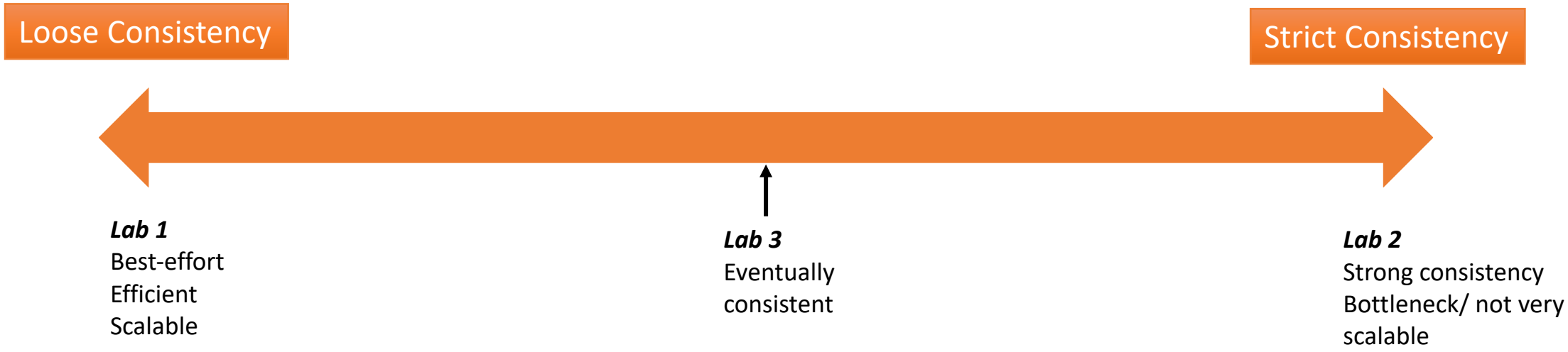
Wrap-up lab 2: Strong consistency and bottleneck

- 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

Research Group
Distributed Systems

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
- 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 – Centralized vs distributed

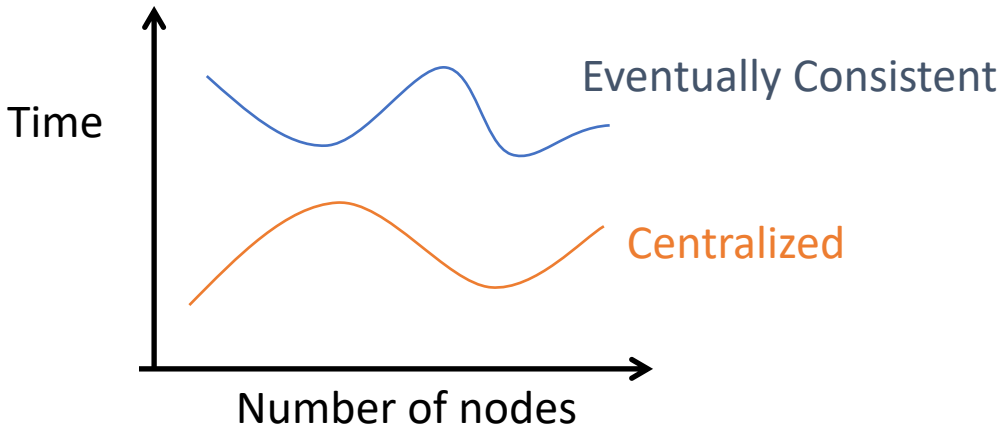
- Evaluate the performance of your system vs lab 2's system
- Evaluate time required to reach consistency
  - Defined as MAX(duration between the first and last message received by one server) over all servers
  - Create a script that *concurrently* sends 10 messages to each server
    - If you use curl, use & at the end of the command!
    - If you use python, create a thread for each request!
- Show its evolution for a varying number of servers
  - 4, 8, 12 servers (change start_topology.py)
  - A correct evaluation averages over multiple trials!

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:



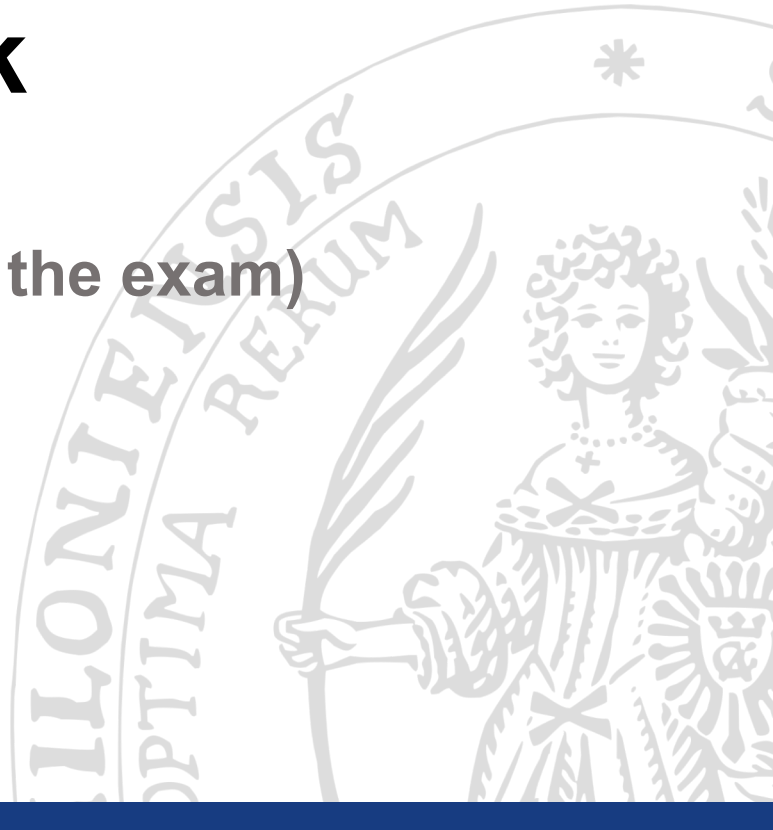
Kiel University
Christian-Albrechts-Universität zu Kiel

Research Group
Distributed Systems

Lab 3 - Optional Task

This is not mandatory to do

It gives bonus points (for the lab score, not the exam)



Optional Task A – Network Segmentation

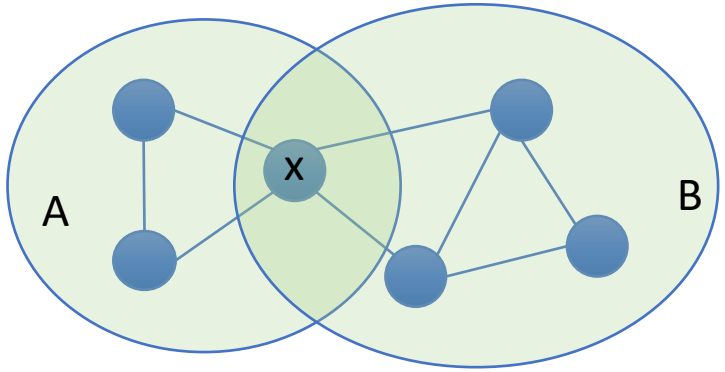
- Show that your system is eventually consistent
  - Even in the presence of network segmentation

- 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

- This task gives up to 4 bonus points

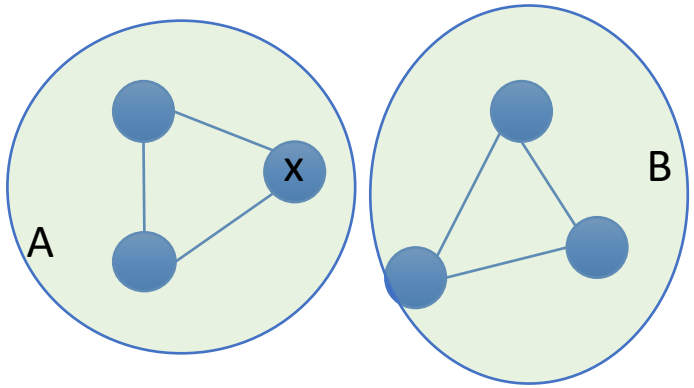
Network segmentation – an example

- Phase 1



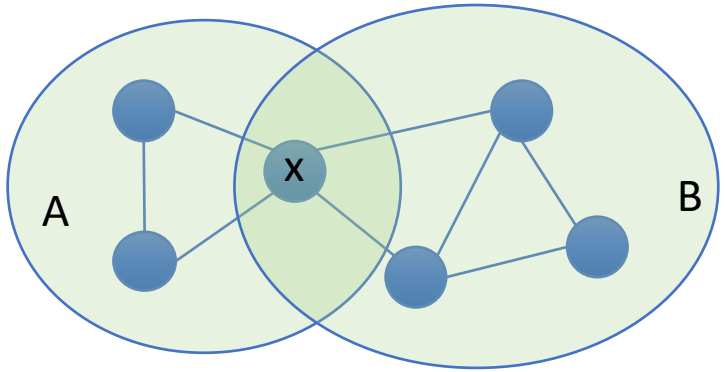
The network is a single segment. Node X communicates with both groups.

- 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.

Optional Task B – Segmentation recovery evaluation

- Evaluate the time it takes to merge two networks after segmentation
  - As the number of nodes in the system grows

- This task gives up to 1 bonus point

Kiel University
Christian-Albrechts-Universität zu Kiel

Research Group
Distributed Systems

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
  - Plot for task 2

- A presentation of your work
  - 3 minutes max per group!
  - Use slides (you can use figures, screenshots, videos…)
  - Focus on the *focus* given to your for this lab (see next slides)
  - Followed by a discussion

Focus list

- You have been given one of the following focuses:
  - Design 
  - Corner Cases 
  - Implementation 
  - Demo 
  - Evaluation 

Focus - Design

- Explain how you designed your system
- What is your algorithm?
- What complexity (message, time)?
- Advantages/Cons of your design vs what we saw in lectures?

Focus - Corner Cases

- Sometimes, protocols need to handle special cases
  - Also known as corner cases
- What corner cases could break your design?
- How did you deal with them?
- Concurrent *Modify* often causes many corner cases…

Focus - Implementation

- Explain how you have implemented your system
- You can focus on some interesting parts
- Did you have to do any tweaks/tricks to make it work?

Focus - Demo

- Show us your system!
- You can record a video or do a live show
  - But live experiments often fail… ☺
- Don’t use screenshots only!
  - We might believe you are hiding a broken system…

Focus - Evaluation

- Evaluate the performance of your system
- Time to convergence?
  - (total amount of time from the first request until all servers have their final board)
- Impact of the number of servers?
  - You can modify start_topology.py for that
- Number of messages?
- This is the hardest focus, but evaluation is a key element for your master thesis/research, so you should learn how to evaluate!

Kiel University
Christian-Albrechts-Universität zu Kiel

Research Group
Distributed Systems

Deadline:
Thursday 5th, December 2019

Good luck, and start coding!