Kiel University
Christian-Albrechts-Universität zu Kiel

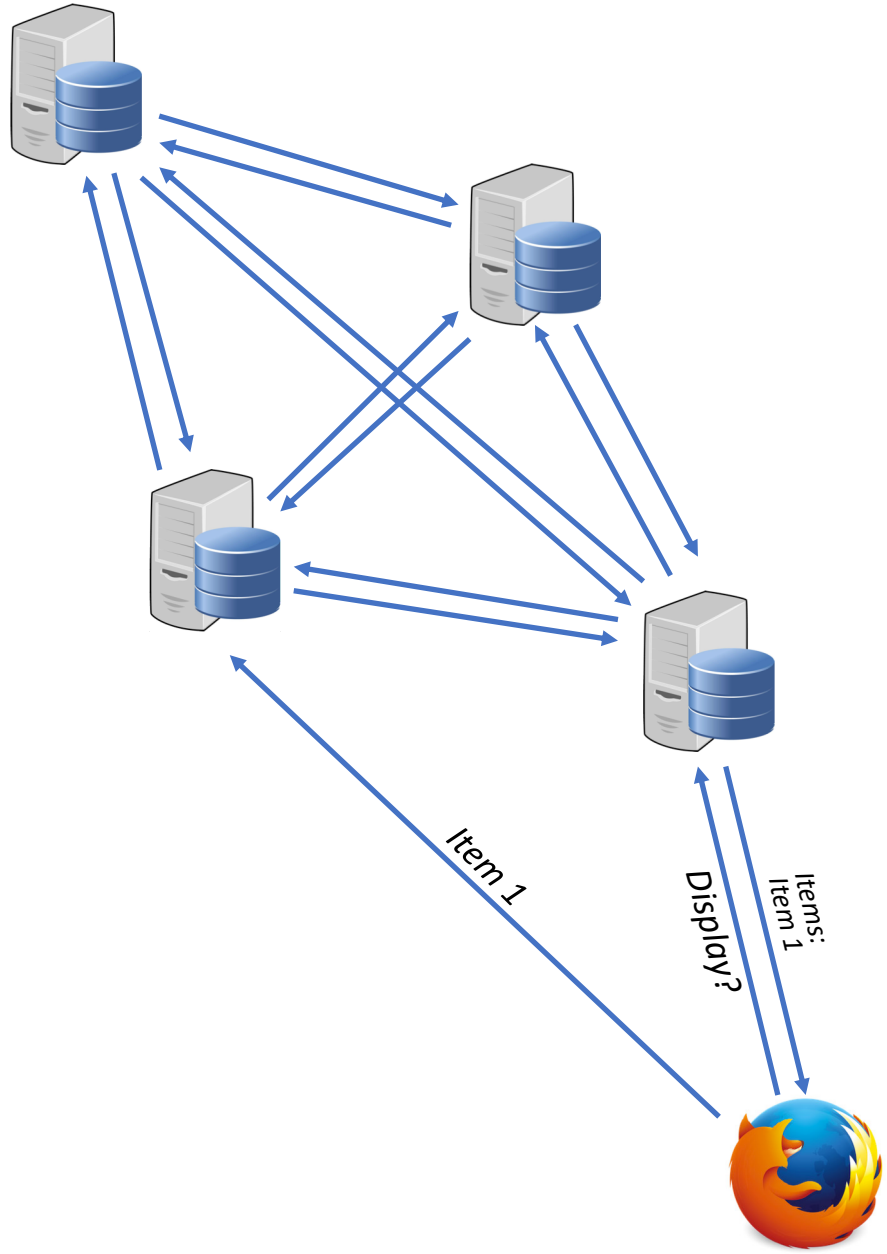
Distributed Systems
Research Group

Lab 4

Byzantine Agreement

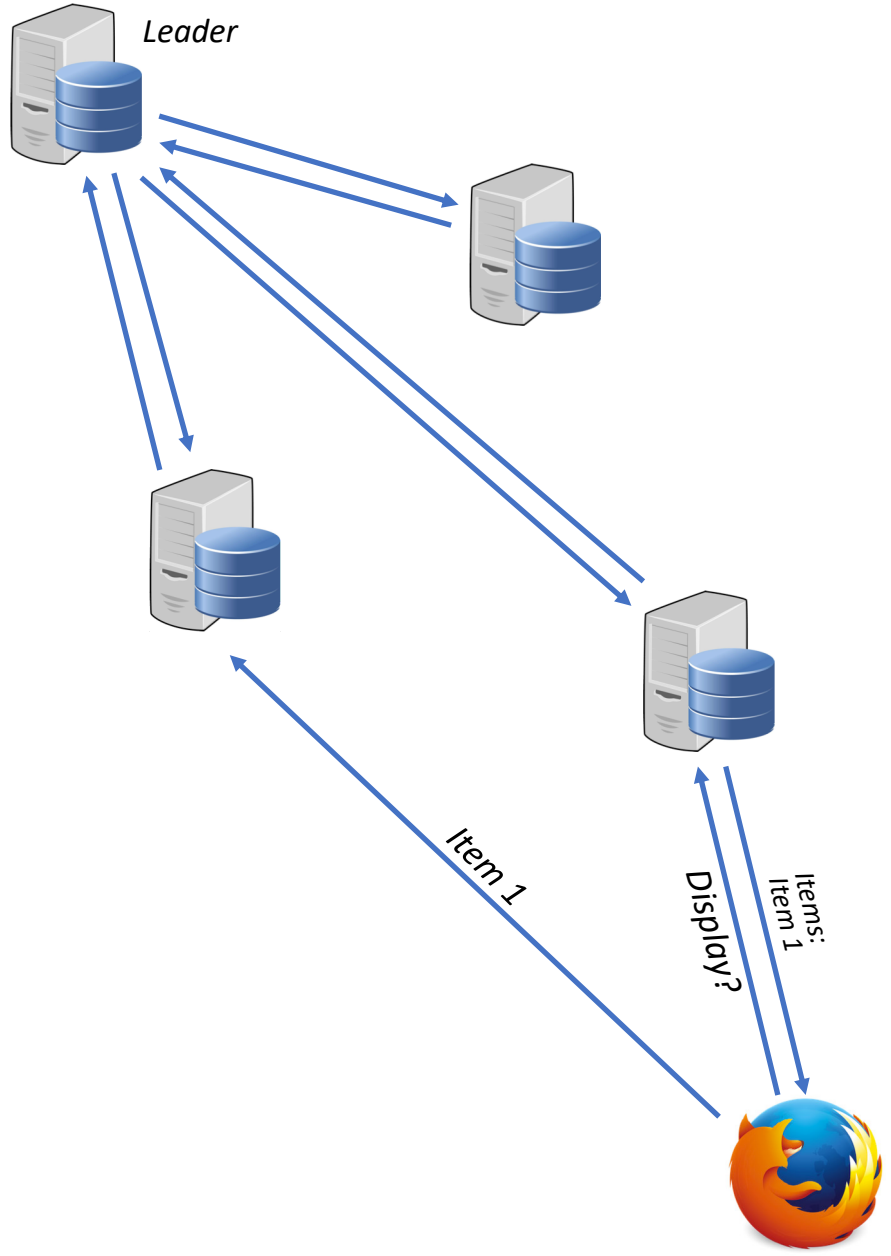
Wrap-up lab 1: An Overly Simple Distributed Blackboard?

- 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!



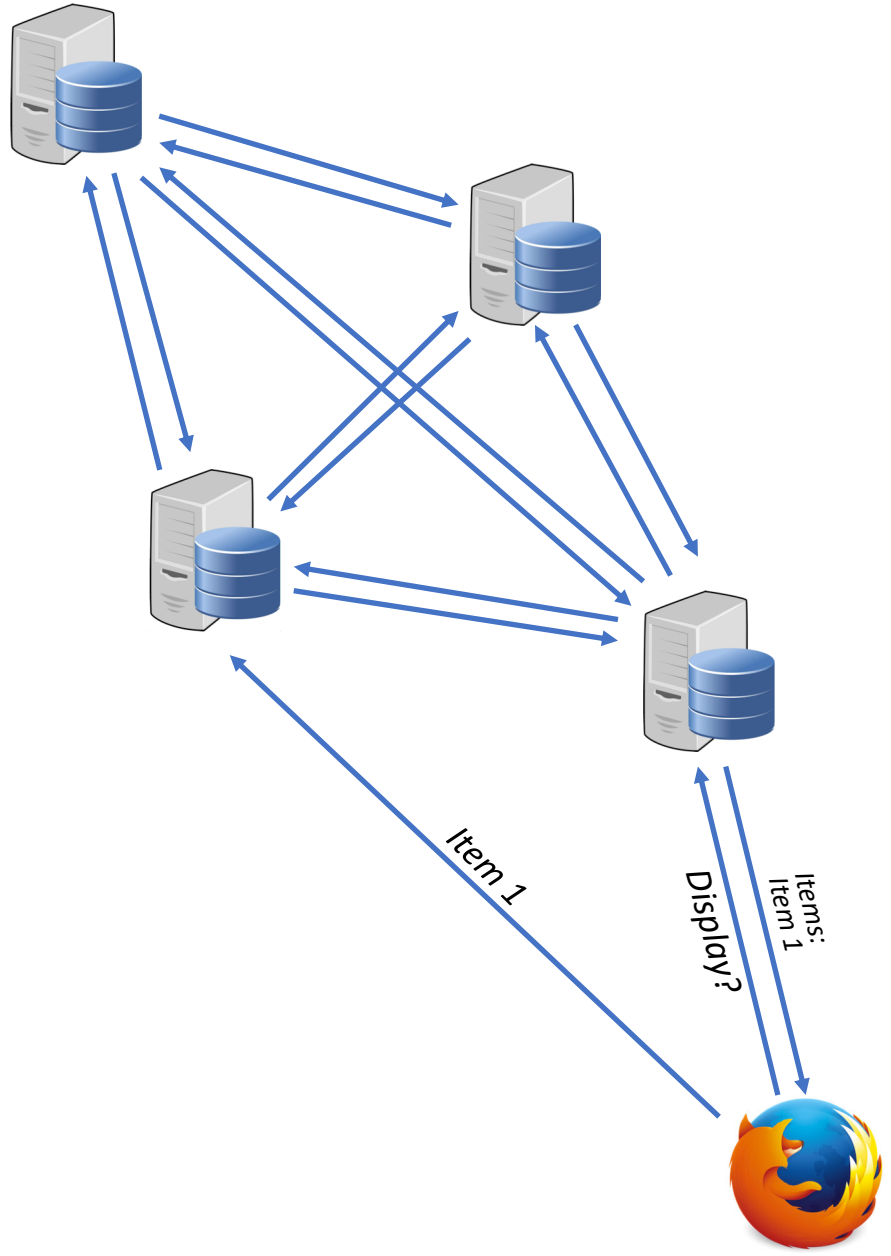
Wrap-up lab 2: Strong consistency and bottleneck

- 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

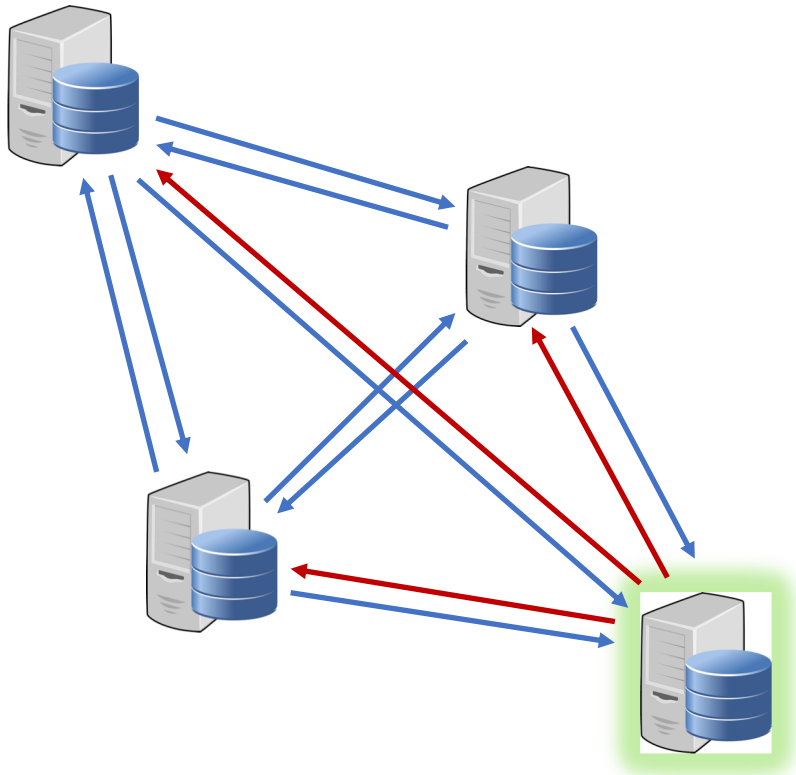


Wrap-up lab 3: Temporary inconsistency but eventually consistent

- Blackboard is *eventually* consistent
  - All entries are the same (eventually)
  - Messages shown in the same order (eventually)
- Advantages?
  - Scalable
  - Can survive network segmentation
- Disadvantages?
  - Boards will appear inconsistent for some time
  - Not ideal from the client perspective



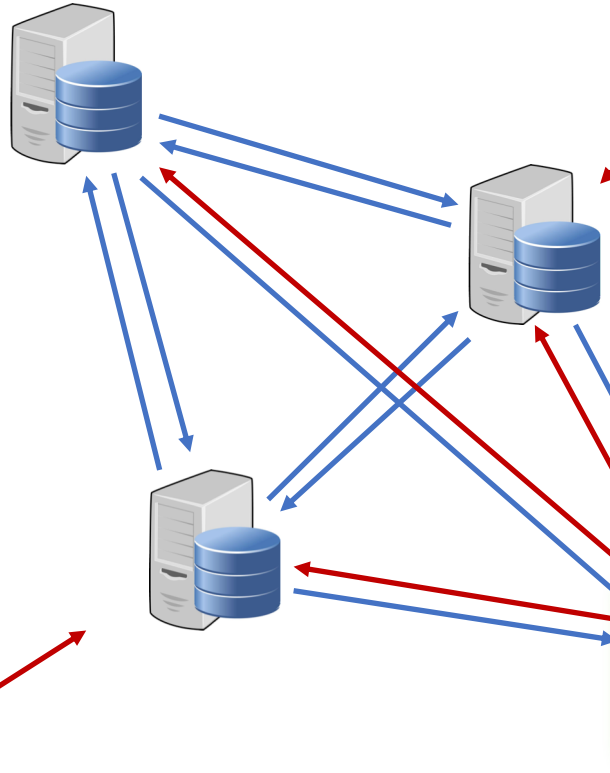
Lab 4 Idea: Some infected servers are working against us!



Lab 4 Idea: The system should still agree on actions

(honest, vote$_1$)

(honest, vote$_2$)



(honest, vote$_3$)

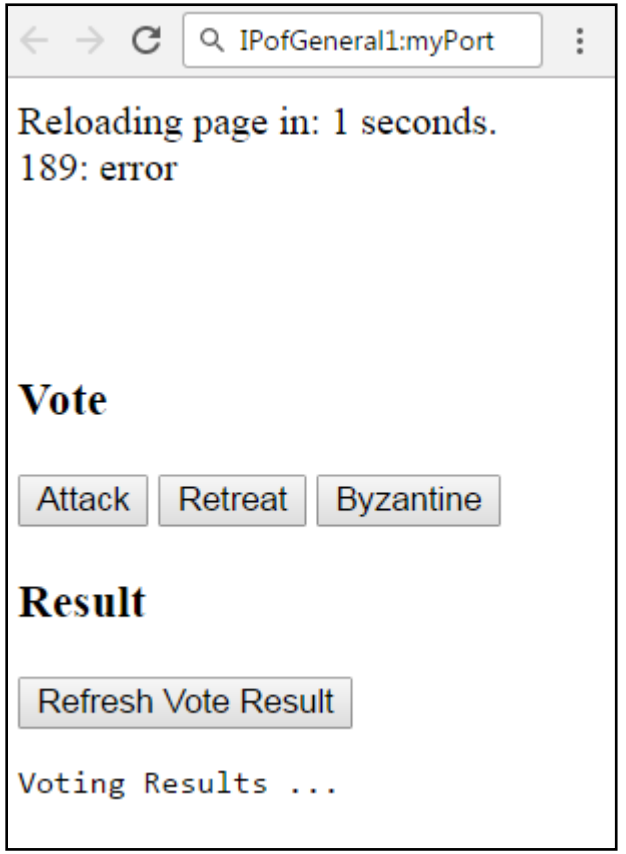
Byzantine

General Setup

- *N* servers, each general controls one server
- For each general (*server*)
  - Decide on a profile: ***Honest*** or ***Byzantine***
  - For honest generals:
    - Decide on a vote: ***Attack*** or ***Retreat***
  - Once all honest generals have voted, start the Byzantine agreement algorithm (see lecture "Fault Tolerance I") on **all** generals
- When the algorithm ends, each server displays its **final result AND result vector**
  - See slide "Byzantine Agreement Problem (10)" on lecture "Fault Tolerance I"
  - Honest generals should agree on the same votes among them
  - Byzantine generals don't need to agree or display the agreement result

Server Interface

- Required web interface:
  - Three buttons:
    - Attack
    - Retreat
    - Byzantine
- By clicking on *Attack* or *Retreat*, the general becomes honest
  - And starts the byzantine agreement algorithm
- By clicking on *Byzantine*, the general becomes byzantine (infected)
  - And starts the byzantine behavior once the agreement algorithm is executed
- We give you some code (see appendix)



What you will do

- Use the web interface to initialize the Byzantine agreement algorithm
  - for each general (server)
  - Recall that initially generals are unaware of each others votes
  - Each of the "*Attack*", "*Retreat*", and "*Byzantine*" buttons should call a function (e.g., through a POST request) that initiates the Byzantine agreement algorithm on that general
- When the function is called with argument "*Attack*" or "*Retreat*", your code should handle the behavior of an honest general that votes attack or retreat, respectively
- When the function is called with argument "*Byzantine*" your code should handle the behavior of a Byzantine general (we provide some code for that – see Appendix)
- All servers are aware of the total number of generals. **Only the Byzantine generals** know which servers are Byzantine (i.e., their IPs).

What you will do (II)

Byzantine agreement algorithm runs in two steps (see lecture “Fault Tolerance I”):

- (step 1) When voting starts:
  - honest servers start by sending out their votes
  - Byzantine generals wait until they collect all the honest votes and send out different votes to the honest servers in order to break agreement (if possible).

- (step 2) When a server has received all votes
  - if honest, it sends to other servers a vector of all votes received
  - if Byzantine, it sends to other servers a vector of Byzantine votes
    - We provide the Byzantine behavior (see Appendix)

- Voting and outcome:
  - when a server has received all the messages from step 2, it computes the (majority vote) result vector and the result
  - Then, it adds the **result vector and the result** to the webpage (to be seen upon refresh)

What you will do (III)

- A general should have different behavior when honest or Byzantine
- Byzantine code (`byzantine_behavior.py`) can be found in Lab 4's page on iLearn and its explanation in the Appendix. The same code works for all the tasks of this lab, but feel free to edit/improve the code (and the interface)
- Recall the "3k+1 rule":
  - when having a total of N = 3k+1 servers, agreement can be reached only if at most k out of them are Byzantine

Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Lab 4 -Tasks

What you NEED to implement

Task 1 – 4 servers (N=4, k=1)

- Select 4 nodes for this subtask. Using the interface set:
  - 3 honest nodes and 1 Byzantine (N=4, k=1)

- Demonstrate that agreement is reached. That is, no matter what the honest servers vote for, agreement can always be reached
  - the result vectors of the honest generals should match on the entries corresponding to the honest generals
  - Hence, the honest generals agree on the same result.

- Note that the Byzantine general must respect the agreement protocol, but it can change the votes to be sent to the honest nodes (e.g. it cannot sent garbage – in this implementation, not in general).

Examples

- On tie, retreat
- Different scenarios:
  - Attack, Attack, Attack, and Byzantine
  - Byzantine, Retreat, Retreat, and Attack
  - Attack, Retreat, Attack, and Byzantine
- Final outcome?

Task 2 – 3 servers (N=3, k=1)

- Select 3 nodes for this subtask. Using the interface set:
  - 2 honest nodes and 1 Byzantine (N=3, k=1)

- Set different votes for the two honest servers. The Byzantine general must be able to convince one server to attack and another one to retreat. cf. example on the ”Fault Tolerance I” lecture slides.

- As in task 1, the Byzantine general can only change the votes to be sent to other nodes, but always respects the agreement protocol.

Task 3 – A general solution?

- **Explain** how you can extend your solution for task 1 to handle more than one Byzantine general

- **Explain** if coordination between the Byzantine generals is needed to break a (non-Byzantine) coordination algorithm, and **why**

Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Lab 4 - 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

Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Deadline:
Wednesday 27th, January

Good luck, and start coding!

Appendix

The code we give you

Extra reading

- Here is a link to the original paper by Lamport et al.: http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf

- There are many more resources about Byzantine agreement on the web (for various versions of the problem)

- This lab is tied to the Byzantine agreement problem as it is described on the lecture slides

Web interface & API – one among many possibilities

- Three buttons
- Show results as simple text
- Page reloads periodically

Action	URI	Explanations	Parameters
POST	/vote/attack /vote/retreat /vote/byzantine	Sets behavior and vote	
GET	/vote/result	Shows end result & result vector	
GET	/	Displays webpage	

HTML Template

- Two files:
  - Main page:
    - vote_frontpage_template.html
  - Result template: (really nothing)
    - vote_result_template.html

- Get template on iLearn

Byzantine behavior

- We provide you the simple functions that implement the logic of the byzantine nodes, on a .py file on iLearn.
  - You just put them in your code and call them as described next.

- On round 1, byzantine node calls:
  - compute_byzantine_vote_round1(#loyal_nodes, #total_nodes, tie-braker)
    - tie-braker is a boolean variable (True or False) indicating Attack or Retreat respectively.
  - Result: A list with the votes to send to the loyal nodes e.g. [True,False,True,….]

Byzantine behavior (II)

- On round 2, byzantine node calls:
  - compute_byzantine_vote_round2(#loyal_nodes, #total_nodes, tie-braker)
  - Result: A list, where every element is the vector to send to each loyal node.
  - e.g. [[True,False,…] , [False,True,…] , …]
  - Again, True stands for Attack and False stands for Retreat.