Kiel University
Christian-Albrechts-Universität zu Kiel

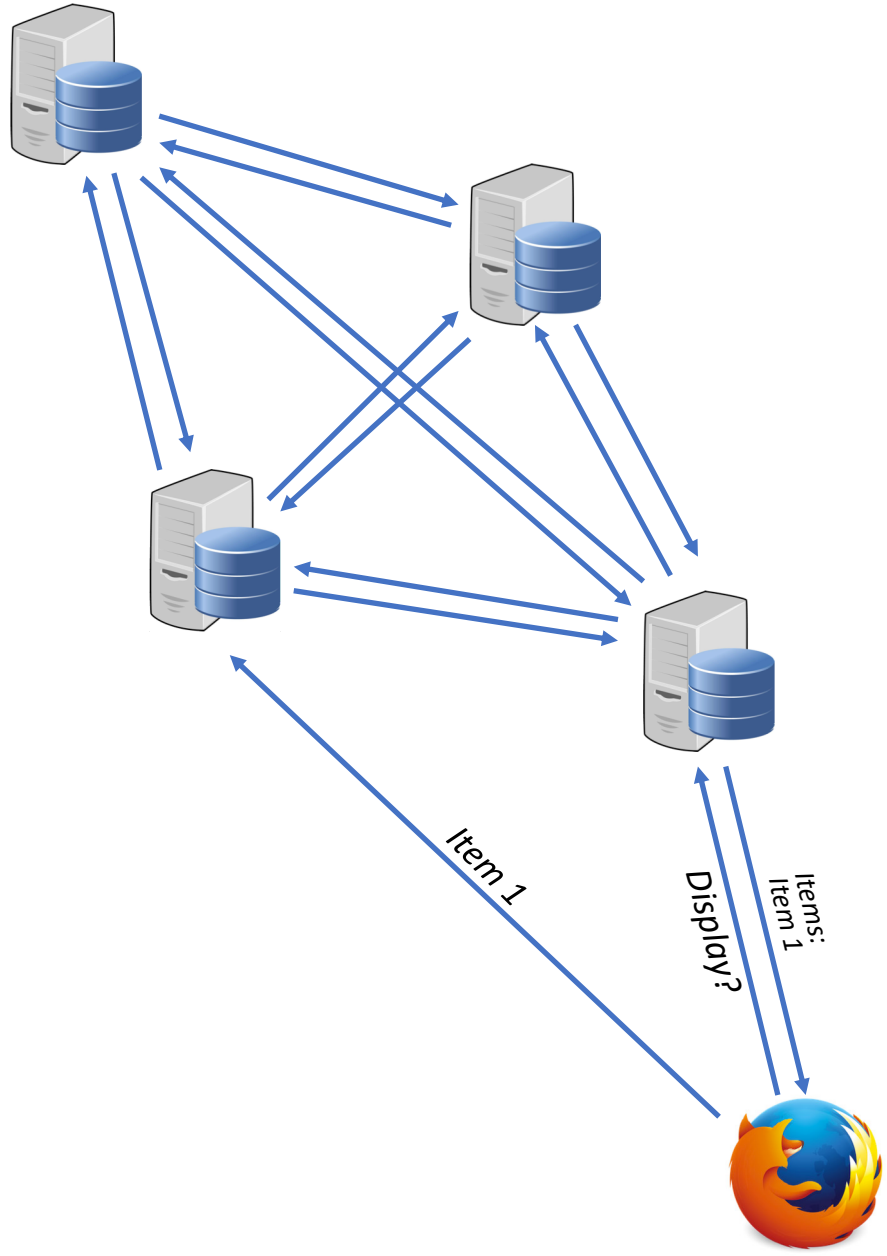
Distributed Systems
Research Group

Lab 2

Leader Election and Centralized Solution

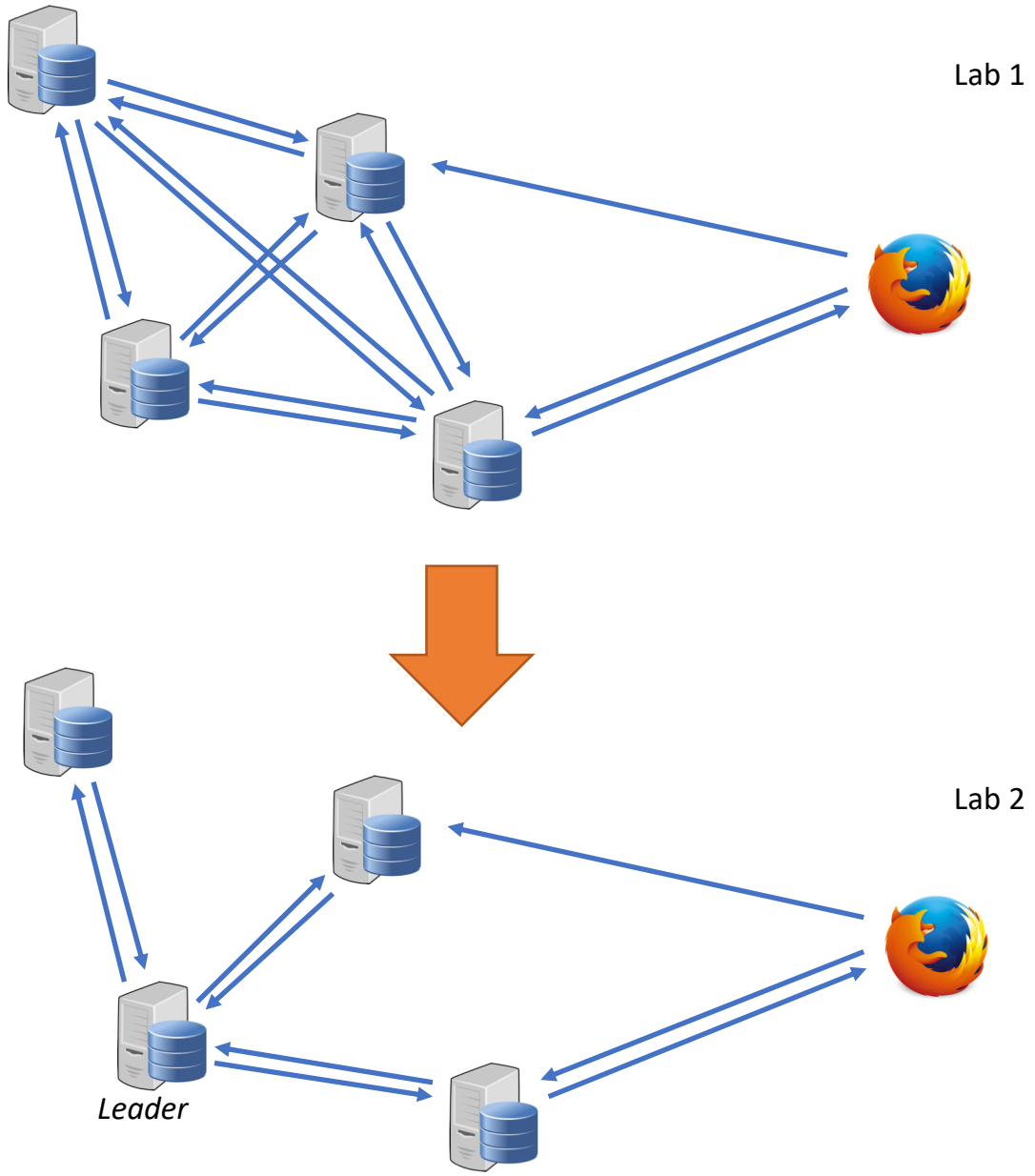
Wrap-up lab 1: An Overly Simple Distributed Blackboard?

- Our blackboard *is* distributed
- But might get inconsistent!
- Consistent blackboard:
  - All entries are the same
  - Messages shown in the same order
  - Order is somehow associated with a notion of time
- How can we have a consistent blackboard?
  - Let's try a centralized solution!

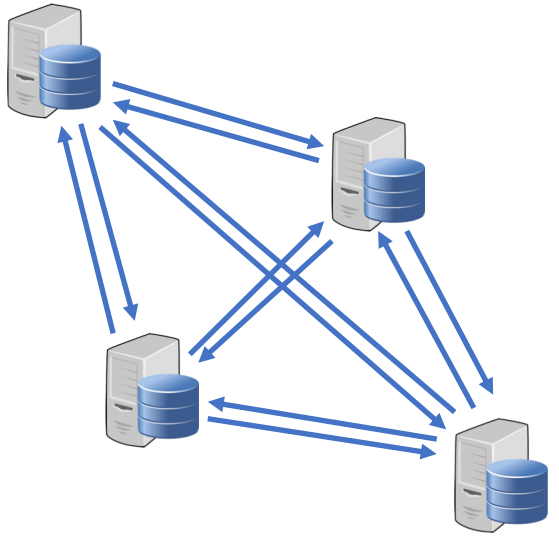


Idea: a centralized solution

- Each post is sent to the leader
- Leader decides what to do and distributes it to the network
- Leader should be able to handle messages from multiple nodes and come up with a unique ordering
- Who will be leader?
  - We must execute a Leader Election (LE)
  - See lecture slides for different LE algorithms

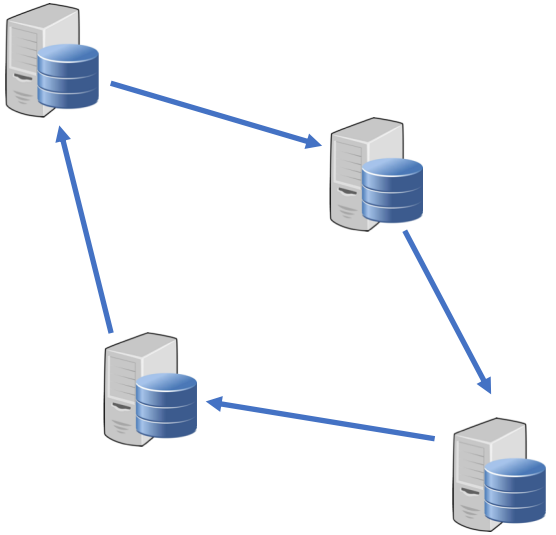


Leader Election



- Which LE algorithm should I use?
  - Bully algorithm?
  - Ring algorithm?
  - Your own LE algorithm?
  - You decide!

- Which election attribute?
  - Plenty of choices: random number, IP or MAC address, some uuid…
  - **This lab: random number, generated when the server boots up**
    - **Some servers might have the same number! You can use the server ID (or IP address) as tie breaker**



Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Lab 2 -Tasks

What you NEED to implement

Task 1 – Leader Election

- Implement a *Leader Election* in your distributed blackboard
  - **All servers should start an election at the beginning!**
  - **LE attribute: random number selected at bootup + server ID as tie breaker**

- Hint: LE should be started at the beginning, but requires the server to be started to work properly. Start LE as a thread, with a delay to let the server start. Use *do_parallel_task_after_delay* just before *bottle.run*, with a few seconds delay

- Display on each server their election attribute, as well as their status (leader, slave)

Task 2 – Centralized Blackboard

- Implement your *Centralized Blackboard*
  - Add, Modify and Delete should work

- Show the strong consistency of your solution
  - Order is the same on all servers
  - Concurrent modify do not break consistency

Task 3 – Dynamic reelection after failures

- During runtime, kill your leader (by closing the terminal running bottle, or shutting down the link within mininet)
- Your system must detect that the leader has failed, and start a new election
  - You can reuse the random number generated at bootup
- What happens if the old leader is rebooted? (with loss of previous state-information)

Task 4 – Argue for your choices

In a small text file uploaded with your code

- Discuss the solution cost of your LE
- Discuss whether your LE fulfills the properties of a correct LE algo
- Discuss the cost of adding messages to the board
- Briefly discuss pros&cons of centralized solutions

Kiel University
Christian-Albrechts-Universität zu Kiel

Distributed Systems
Research Group

Lab 2 - 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 9th, December

Good luck, and start coding!

Additional slides

Agenda

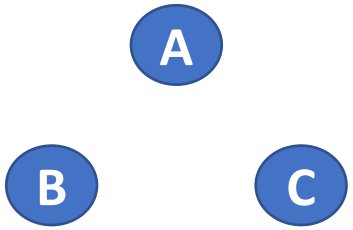
- Lab 2

- Solution cost for Lab 1
- Mininet Python API (optional)
- A small note on threads & CURL

Solution cost

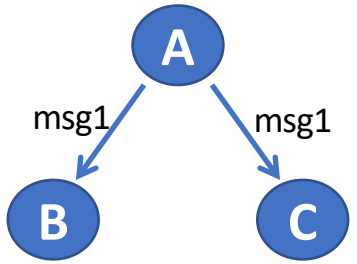
- We care about the solution cost in terms of communication
- We can measure the cost of a solution (e.g. in Lab 1) in terms of
  - number of nodes to which a new post is propagated
  - payload: number of blackboard entries per message

- For example, consider the case of three nodes A, B and C, and the following events:
  - Event 1: User posts “msg1” to node A.
  - Event 2: User posts “msg2” to node A.

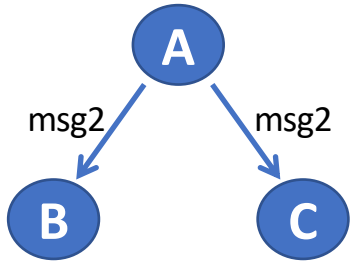


Example: a good scenario

- Propagate only the new post.
- Upon Event 1,
  Vessel A sends “msg1” to vessels B and C
  - Payload for each message = 1
  - Overall cost = 2



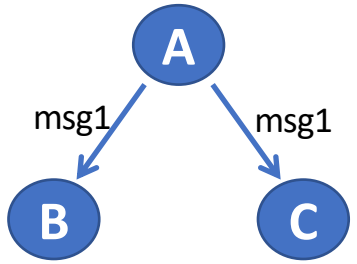
- Upon Event 2,
  Vessel A sends “msg2” to vessels B and C
  - Payload for each message = 1
  - Overall cost = 2



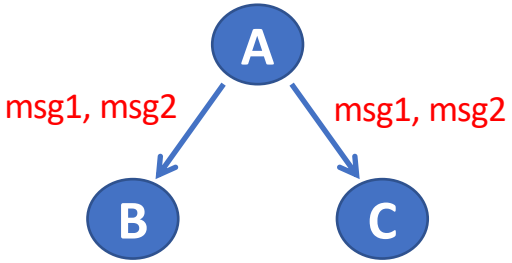
- Overall cost per post: #nodes-1

Example: a costly scenario

- Propagate the whole blackboard
- Upon Event 1, Vessel A sends “msg1” to vessels B and C
  - Payload for each message = 1
  - Overall cost = 2



- Upon Event 2, Vessel A sends “msg1, msg2” to vessels B and C
  - Payload for each message = 2
  - Overall cost = 4



- Overall cost per post: b(n-1)
  - board size *(#nodes-1)

Cost of Lab2

- We want you do the same simple communication cost analysis for Lab2:
  - First for the leader election part.
    - What is the cost of the whole leader election?
  - Then for the centralized blackboard.
    - What is the cost of a new post?

Agenda

- Lab 2
- Solution cost for Lab 1
- Mininet Python API (optional)
- A small note on threads & CURL

Mininet Python API

- Staff that are not really required for the labs, but it is good to know ☺
- You can use a Python API to:
  - Set up your own topologies.
  - Configure link properties e.g. bandwidth, delay etc.
  - Run arbitrary commands on the hosts.
  - much more…
- …all through python scripts.
- For example, *start_topology.py* uses that API.

Example 1: Building a custom topology (from the Mininet tutorial)

Defines a topology with n nodes connected to a single switch

```
class SingleSwitchTopo(Topo):

        def build(self, n=2):
                switch = self.addSwitch('s1')
                 # Python's range(N) generates 0..N-1
                 for h in range(n):
                        host = self.addHost('h%s' % (h + 1))
                                                self.addLink(host,
switch)
```

Example 1: Building a custom topology (from the Mininet tutorial)

Defines a topology with n nodes connected to a single switch

```
class SingleSwitchTopo(Topo):

        def build(self, n=2):
                switch = self.addSwitch('s1')
                 # Python's range(N) generates 0..N-1
                 for h in range(n):
                        host = self.addHost('h%s' % (h + 1))
                                        self.addLink(host,
switch)
```

Inherit class *Topo* and override method *build*

add a switch

create a host

link it to the switch

Example 1: Building a custom topology (from the Mininet tutorial)

Defines a topology with n nodes connected to a single switch

```
class SingleSwitchTopo(Topo):

        def build(self, n=2):
                switch = self.addSwitch('s1')
                 # Python's range(N) generates 0..N-1
                 for h in range(n):
                        host = self.addHost('h%s' % (h + 1))
                                                self.addLink(host,
switch)
```

Starts Mininet with the specified topology

```
def simpleTest():

        topo = SingleSwitchTopo(n=4)
        net = Mininet(topo)
        net.start()
        print "Dumping host connections”
        dumpNodeConnections(net.hosts)
        print "Testing network connectivity"
        net.pingAll()
        net.stop()
```

Example 2: Configuring link parameters (from the Mininet tutorial)

```
class SingleSwitchTopo(Topo):

    def build(self, n=2):
        switch = self.addSwitch('s1')
        # Python's range(N) generates 0..N-1
        for h in range(n):
            host = self.addHost('h%s' % (h + 1))
                self.addLink( host, switch, bw=10, delay='5ms', loss=2 )
```

configure link properties

bandwidth in Mbs

delay in ms

loss rate %

Example 2: Configuring link parameters (from the Mininet tutorial)

```
class SingleSwitchTopo(Topo):

        def build(self, n=2):
                switch = self.addSwitch('s1')
                 # Python's range(N) generates 0..N-1
                 for h in range(n):
                        host = self.addHost('h%s' % (h + 1))
                                self.addLink( host, switch, bw=10, delay='5ms', loss=2 )
```

Test the bandwidth between two nodes

```
h1, h4 =  net.get(  'h1',  'h4'  )
net.iperf( (h1, h4) )
```

or run any command on a node

```
result = h1.cmd('ifconfig')
print result
```

Mininet Python API

- If you want to know more:
  1. here is walkthrough of the API https://github.com/mininet/mininet/wiki/Introduction-to-Mininet
  2. repo with many example scripts https://github.com/mininet/mininet/tree/master/examples
  3. Lots of videos about Mininet, making custom topologies etc. https://github.com/mininet/mininet/wiki/Videos
  4. or look at the script that we gave you for lab 1 (lab1.py).

Agenda

- Lab 2
- Solution cost for Lab 1
- Mininet Python API (optional)
- A small note on threads & CURL

A small note about threads(1)

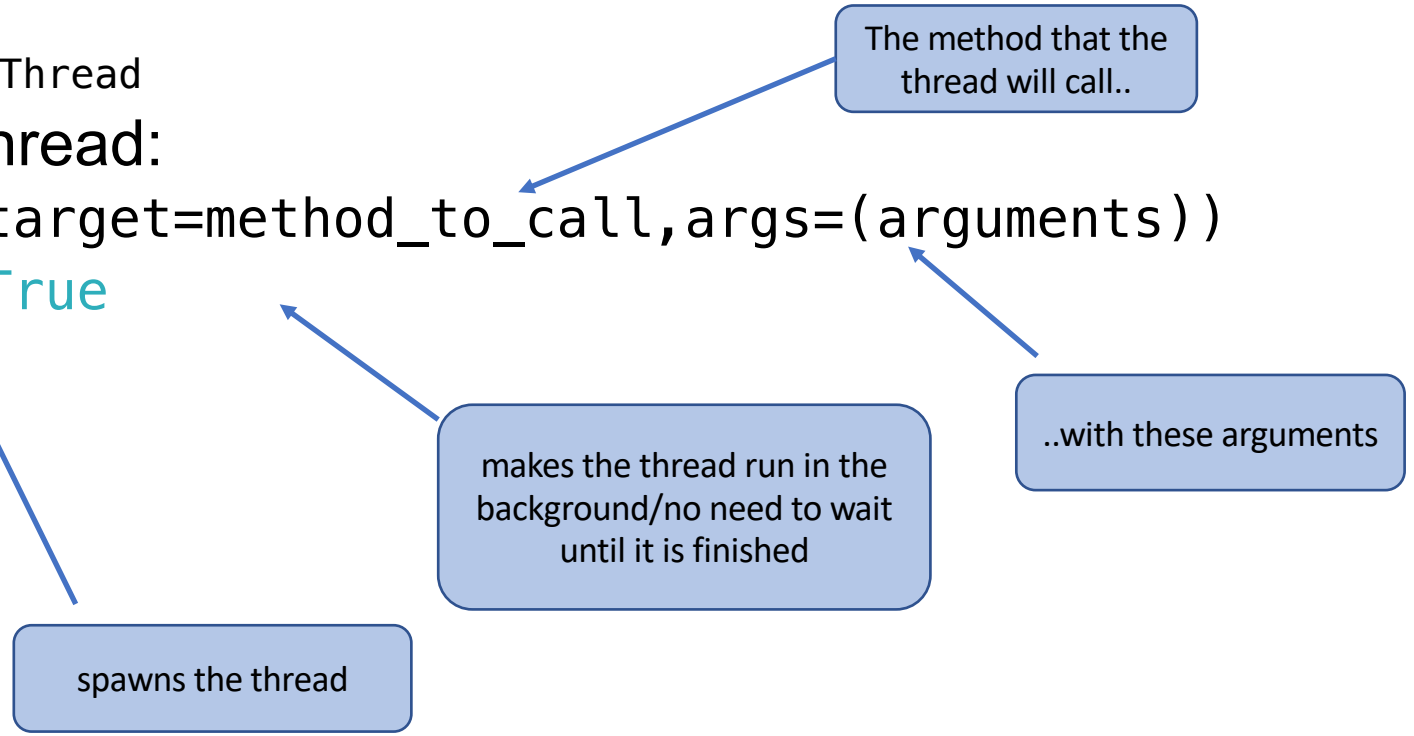
- At the *server.py* that we give you
  - Inside method *do_POST*

```
thread = Thread(target=self.server.propagate_value_to_vessels,args=("action", "key", "value") )

thread.daemon = True

thread.start()
```

- The idea is to offload the propagation of messages to a different thread.
- So that do_POST can return immediately.

A small note about threads(2)

- Threads in python:
  - `from threading import Thread`
  - How to spawn a Thread:
    - 1) `t = Thread(target=method_to_call,args=(arguments))`
    - 2) `t.daemon = True`
    - 3) `t.start()`



A small note about threads(3)

- No need to worry much about it in the labs.
- Use the code as it is when you want to propagate.

… but if you want to know more…

- the threading library:
  - https://docs.python.org/3/library/threading.html
- An interesting presentation (advanced):
  - https://www.slideshare.net/dabeaz/an-introduction-to-python-concurrency

CURL

- There is more than Firefox
  - HTTP POST on the command line

```
curl -d 'entry=t'${i} -X 'POST' 'http://ip:port/entries'
          done
```

This will post entry=t1, entry=t2, ..., entry=t20 to http://ip:port/entries

General notes about the labs

- Mininet is used as an infrastructure to test your servers on complex topologies with many nodes.
- Servers are “plain” Python, that can run everywhere.
- You can test them locally, on your own machine.
  - Might need to change the default port and how to contact the neighbors.
  - E.g. Start two servers on localhost, one on port 8080 and one on 8081.
- Then switch to Mininet to test with a bigger topology.