CHALMERS
UNIVERSITY OF TECHNOLOGY

Distributed Computing and Systems
Chalmers university of technology

Distributed Systems I

Lab Introduction

TAs

Beshr Al Nahas

beshr

Charalampos (Babis) Stylianopoulos

chasty

Iosif Salem

iosif

@chalmers.se

Lab time: Monday 15:15-17:00, Thursday 08:00-9:45
Ask your questions during the lab time!

Lab Introduction

- Lab Introduction I (Nov. 3):
  - Seattle platform, Lab overview, Lab 1 introduction (1h)
  - Introduction to RESTful, Repy skeleton (1h)

- Lab Introduction II (Nov. 11):
  - Lab 2 introduction
  - Solution costs
  - Concurrency in Repy

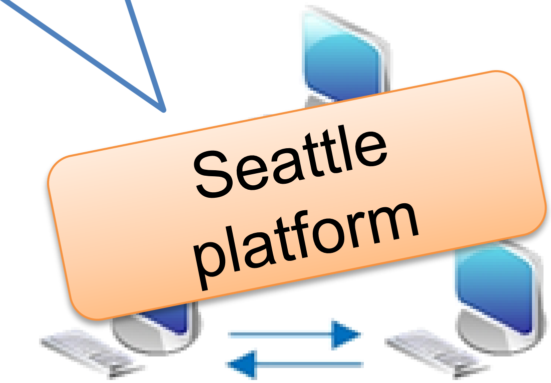
Labs in a nutshell

Lab overview

During the labs you will make this application:

- Reliable
- Consistent
- Efficient
- Fault-tolerant
- …

You will build an application over this network

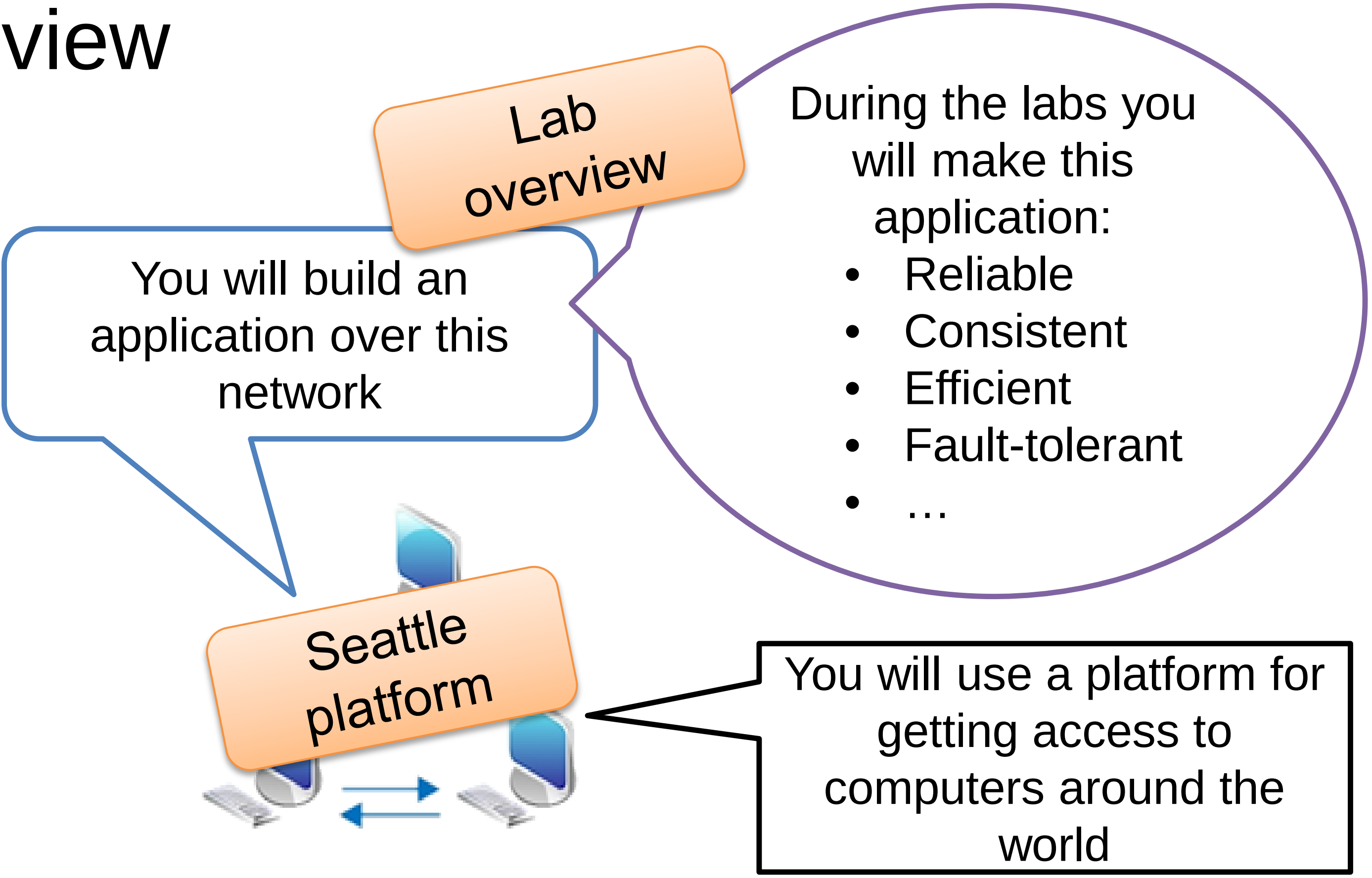


Seattle platform

You will use a platform for getting access to computers around the world

Agenda

- Seattle platform
- Lab overview
- Lab 1



Lab overview

During the labs you will make this application:

- Reliable
- Consistent
- Efficient
- Fault-tolerant
- …

You will build an application over this network

Seattle platform

You will use a platform for getting access to computers around the world

Seattle intro

- A platform that allows you to get access to computers around the world

- Each machine allows you to run your application in a **sandbox** with **limited resources**

- You will use this platform to study a distributed system in practice!

Seattle intro

- Seattle
  - Write code in a subset of Python (Repy = **Re**stricted **py**thon) and run on machines all over the world
  - These machines are provided by the community
  - Sandbox: **limits** the consumption of CPU, memory, storage space, network bandwidth, etc.
  - https://seattle.poly.edu/html/

- A perfect platform for teaching distributed systems
  - Experience distributed nature: machines all over the globe
    - Machines may disconnect, break, limited bandwidth, …
  - Use a modern language: Python
    - Fast prototyping

Seattle Resources

- **5 min video**
  - https://seattle.poly.edu/wiki/UnderstandingSeattle/DemoVideo
- Installation, tutorials, etc.
  - https://seattle.poly.edu/wiki/ProgrammersPage
  - Check python tutorial for **exception handling**
- Repy tutorial
  - https://seattle.poly.edu/wiki/RepyTutorial
- Repy API
  - https://seattle.poly.edu/wiki/RepyApi

Seattle Demo

How to start:

- Register on Seattle Clearinghouse
- Install demokit.zip from your profile page
- Generate your private/public key pair and put them in the demokit directory
- Put the restrictions.test file in the demokit directory

Seattle demo: Repy examples

- Seash Shell command
  - https://seattle.poly.edu/wiki/SeattleShell
  - https://seattle.poly.edu/wiki/RepyTutorial

- Hello World example:
  - example.1.1.repy
- Hello World to web users:
  - example.1.2.repy
  - example.1.3.repy: **no global in Repy**

Example 1.1: Hello world v1

```
if callfunc == 'initialize':
    print "Hello World"
```

Goal:
Print “Hello World”
on each machine’s log

Example 1.1: Hello world v1

```
if callfunc == 'initialize':
    print "Hello World"
```

Add this to execute your code only once (due to Repy’s architecture)

Example 1.1: Hello world v1

```
if callfunc == 'initialize':
    print "Hello World"
```

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header
htmlresponse = "<html><head><title>Hello World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"
sockobj.send("HTTP/1.1 200 OK\r\nContent-type: text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse), htmlresponse))
stopcomm(thiscommhandle)
# close my connection with this user (we could also do sockobj.close())

if callfunc == 'initialize':
          if len(callargs) > 1:
                    raise Exception("Too many call arguments")
          # Running remotely:
          # whenever this vessel gets a connection on its IPaddress:GENIport it'll call
hello
          elif len(callargs) == 1:
                    port = int(callargs[0]) ip = getmyip()
          # Running locally:
          # whenever we get a connection on 127.0.0.1:12345 we'll call hello
          else:
                    port = 12345
                    ip = '127.0.0.1'
          listencommhandle = waitforconn(ip,port,hello)
```

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header
htmlresponse = "<html><head><title>Hello World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"
sockobj.send("HTTP/1.1 200 OK\r\nContent-type: text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse), htmlresponse))
stopcomm(thiscommhandle)
# close my connection with this user (we could also do sockobj.close())

if callfunc == 'initialize':
          if len(callargs) > 1:
                    raise Exception("Too many call arguments")
          # Running remotely:
          # whenever this vessel gets a connection on its IPaddress:GENIport it'll call hello
          elif len(callargs) == 1:
                    port = int(callargs[0]) ip = getmyip()
          # Running locally:
          # whenever we get a connection on 127.0.0.1:12345
          else:
                    port = 12345
                    ip = '127.0.0.1'
          listencommhandle = waitforconn(ip,port,hello)
```

Goal:

Upon a connection to a machine I control through Seattle, show a “Hello World” webpage

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header
htmlresponse = "<html><head><title>Hello World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"
sockobj.send("HTTP/1.1 200 OK\r\nContent-type: text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse), htmlresponse))
stopcomm(thiscommhandle)
# close my connection with this user (we could also do sockobj.close

if callfunc == 'initialize':
          if len(callargs) > 1:
                    raise Exception("Too many call arguments")
          # Running remotely:
          # whenever this vessel gets a connection on its IPaddress:G
hello
          elif len(callargs) == 1:
                    port = int(callargs[0]) ip = getmyip()
          # Running locally:
          # whenever we get a connection on 127.0.0.1:12345 we'll cal
          else:
                    port = 12345
                    ip = '127.0.0.1'
          listencommhandle = waitforconn(ip,port,hello)
```

Callback function hello

Initialization: Set ip and port and wait for connections.

Call hello upon connection to ip:port

Example 1.2: Hello world v2

```
[…]
if callfunc == 'initialize':
        if len(callargs) > 1:
                raise Exception("Too many c
        # Running remotely:
        # whenever this vessel gets a connection on its
IPaddress:GENIport it'll call hello
        elif len(callargs) == 1:
                port = int(callargs[0]) ip = getmyip()
        # Running locally:
        # whenever we get a connection on 127.0.0.1:12345
we'll call hello
        else:
                port = 12345
                ip = '127.0.0.1'
        listencommhandle = waitforconn(ip,port,hello)
```

Run this once

Example 1.2: Hello world v2

```
[…]
if callfunc == 'initialize':
        if len(callargs) > 1:
                raise Exception("Too many call arguments")
        # Running remotely:
        # whenever this vessel gets a con
IPaddress:GENIport it'll call hello
        elif len(callargs) == 1:
                port = int(callargs[0]) ip = getmyip()
        # Running locally:
        # whenever we get a connection on 127.0.0.1:12345
we'll call hello
        else:
                port = 12345
                ip = '127.0.0.1'
        listencommhandle = waitforconn(ip,port,hello)
```

Too many arguments

Example 1.2: Hello world v2

```
[…]
if callfunc == 'initialize':
        if len(callargs) > 1:
                raise Exception("Too many call arguments")
        # Running remotely:
        # whenever this vessel gets a connection on its
IPaddress:GENIport it'll call hello
        elif len(callargs) == 1:
                port = int(callargs[0]) ip = getmyip()
        # Running locally:
        # whenever we get a connection o
we'll call hello
        else:
                port = 12345
                ip = '127.0.0.1'
        listencommhandle = waitforconn(ip,port,hello)
```

Set port and ip according to input arguments

Example 1.2: Hello world v2

```
[…]
if callfunc == 'initialize':
      if len(callargs) > 1:
            raise Exception("Too many call arguments")
      # Running remotely:
      # whenever this vessel gets a connection on its
IPaddress:GENIport it'll call hello
      elif len(callargs) == 1:
            port = int(callargs[0]) ip = getmyip()
      # Running locally:
      # whenever we get a connection on 127.0.0.1:12345
we'll call hello
      else:
            port = 12345
            ip = '127.0.0.1'
      listencommhandle = waitforconn(
```

Set local ip and port

Example 1.2: Hello world v2

```
[…]
if callfunc == 'initialize':
    if len(callargs) > 1:
        raise Exception("Too many call arguments")
    # Running remotely:
    # whenever this vessel gets a connection on its
IPaddress:GENIport it'll call hello
    elif len(callargs) == 1:
        port = int(call
    # Running locally:
    # whenever we get a c
we'll call hello
    else:
        port = 12345
        ip = '127.0.0.1'
    listencommhandle = waitforconn(ip,port,hello)
```

Wait for connections to ip:port.
Upon connection call hello

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(      Receive HTTP header
```

hello(IP address, Port number, Socket Object, commhandle1, commhandle2)

```
"<body><h1> Hello World! </h1></body></html>"


sockobj.send("HTTP/1.1 200 OK\r\nContent-type:
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse),
htmlresponse))


stopcomm(thiscommhandle)
# close my connection with this user (we could also do
sockobj.close())
[…]
```

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header

htmlresponse = "<html><head><title>Hello
World</title></head>" + \
"<body><h1> Hello World! </h1></body></h

sockobj.send("HTTP/1.1 200 OK\r\nContent-type:
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse),
htmlresponse))

stopcomm(thiscommhandle)
# close my connection with this user (we could also do
sockobj.close())
[…]
```

Receive http header
save to sockobj

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header

htmlresponse = "<html><head><title>Hello
World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"

sockobj.send("HTTP/1.1 200 OK\r\nContent-
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htr
htmlresponse))

stopcomm(thiscommhandle)
# close my connection with this user (we could also do
sockobj.close())
[…]
```

Make html page for response

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header

htmlresponse = "<html><head><title>Hello
World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"

sockobj.send("HTTP/1.1 200 OK\r\nContent-type:
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse),
htmlresponse))

stopcomm(thiscommhandle)
# close my connection with this user (we
sockobj.close())
[…]
```

Respond to connection

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header

htmlresponse = "<html><head><title>Hello
World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"

sockobj.send("HTTP/1.1 200 OK\r\nContent-type:
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse),
htmlresponse))

stopcomm(thiscommhandle)
# close my connection with this user (we could also do
sockobj.close())
[…]
```

Close connection
(resources are limited!)

Example 1.2: Hello world v2

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header

htmlresponse = "<html><head><title>Hello
World</title></head>" + \
"<body><h1> Hello World! </h1></body></html>"

sockobj.send("HTTP/1.1 200 OK\r\nContent-type:
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse),
htmlresponse))

stopcomm(thiscommhandle)
```

takeaway

Check the Repy API for each of these connection handling functions!

Many of your bugs will related to them!

Example 1.3: adding a hit counter

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header
mycontext['pagecount'] = mycontext['pagecount'] + 1
htmlresponse = "<html><head><title>Hello World</title></head>" + \
"<body><h1> Hello World!</h1><p>You are visitor " +  str(mycontext['pagecount']) +
"</p></body></html>"
sockobj.send("HTTP/1.1 200 OK\r\nContent-type: text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse), htmlresponse))
stopcomm(thiscommhandle) # close my connection with this user
if callfunc == 'initialize':
          mycontext['pagecount'] = 0
          if len(callargs) > 1:
                    raise Exception("Too many call arguments")
          # Running remotely:
          # whenever this vessel gets a connection on its IPaddress:GENIport it'll call
hello
          elif len(callargs) == 1:
                    port = int(callargs[0])
                    ip = getmyip()
          # Running locally:
          # whenever we get a connection on 127.0.0.1:12345 we'll call hello
          else:
                    port = 12345
                    ip = '127.0.0.1'
          listencommhandle = waitforconn(ip,port,hello)
```

Goal:
Add hit counter
to Example 1.2

Example 1.3: adding a hit counter

```
def hello(ip,port,sockobj, thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header
mycontext['pagecount'] = mycontext['pagecount'] + 1
htmlresponse = "<html><head><title>Hello World</title></head>" + \
"<body><h1> Hello World!</h1><p>You are visitor " +  str(mycontext['
"</p></body></html>"
sockobj.send("HTTP/1.1 200 OK\r\nContent-type: text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse), htmlresponse))
stopcomm(thiscommhandle) # close my connection with this user
if callfunc == 'initialize':
          mycontext['pagecount'] = 0
          if len(callargs) > 1:
                    raise Exception("Too many call arguments")
          # Running remotely:
          # whenever this vessel gets a connection on its IPaddress:G
hello
          elif len(callargs) == 1:
                    port = int(callargs[0])
                    ip = getmyip()
          # Running locally:
          # whenever we get a connection on 127.0.0.1:12345 we'll cal
          else:
                    port = 12345
                    ip = '127.0.0.1'
          listencommhandle = waitforconn(ip,port,hello)
```

Callback function hello

Initialization: Set ip and port and wait for connections.

Call hello upon connection to ip:port

Example 1.3: adding a hit counter

```
def hello(ip,port,sockobj,thiscommhandle,listencommhandle):
sockobj.recv(512) # Receive HTTP header

mycontext['pagecount'] = mycontext['pagecount'] + 1

htmlresponse = "<html><head><title>Hello
World</title></head>" + \
"<body><h1> Hello World!</h1><p>You are
str(mycontext['pagecount']) + "</p></bo

sockobj.send("HTTP/1.1 200 OK\r\nConten
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(h
htmlresponse))

stopcomm(thiscommhandle) # close my conne
user
[…]
```

No global variables in Repy! (even though they are supported in Python)

Use this dictionary instead!

Example 1.3: adding a hit counter

Create html response, send it, close connection

```
def hello(ip, port, sockobj, thiscommhandle, listencommhandle):
so

my

htmlresponse = "<html><head><title>Hello 
World</title></head>" + \
"<body><h1> Hello World!</h1><p>You are visitor " + 
str(mycontext['pagecount']) + "</p></body></html>"

sockobj.send("HTTP/1.1 200 OK\r\nContent-type: 
text/html\r\n" + \
"Content-length: %i\r\n\r\n%s" % (len(htmlresponse), 
htmlresponse))

stopcomm(thiscommhandle) # close my connection with this 
user
[…]
```

Example 1.3: adding a hit counter

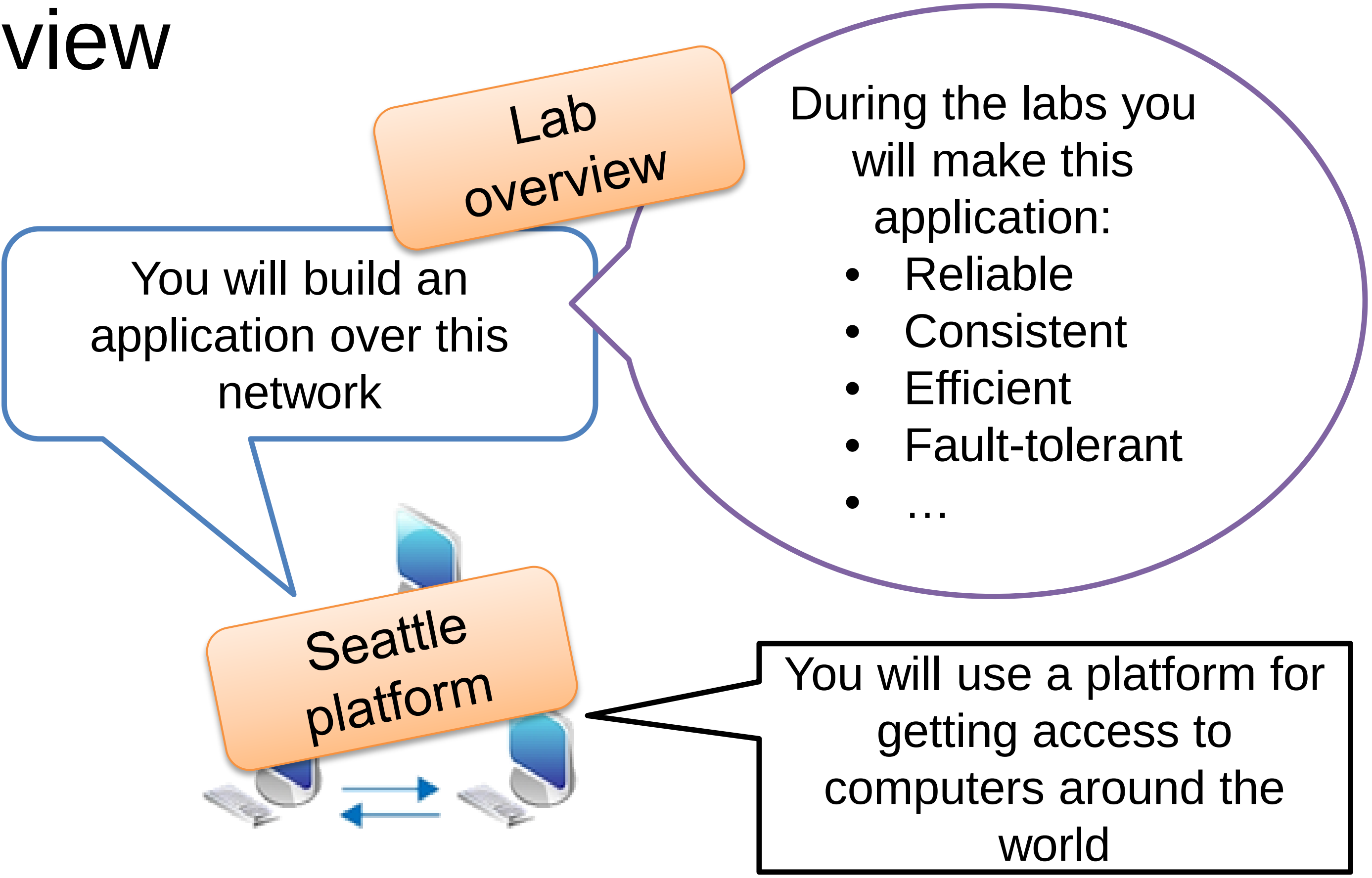
```
[…]
if callfunc == 'initialize':
    mycontext['pagecount'] = 0
    if len(callargs) > 1:
        raise Exception("Too m
    # Running remotely:
    # whenever this vessel gets
IPaddress:GENIport it'll call hello
    elif len(callargs) == 1:
        port = int(callargs[0])
        ip = getmyip()
    # Running locally:
    # whenever we get a connection on 127.0.0.1:12345
we'll call hello
    else:
        port = 12345
        ip = '127.0.0.1'
    listencommhandle = waitforconn(ip,port,hello)
```

Similar code to Example 1.2:

1. Checks arguments to set port and ip
2. Callback hello to be called upon a connection to ip:port

Agenda

- Seattle platform
- Lab overview
- Lab 1



Lab overview

During the labs you will make this application:

- Reliable
- Consistent
- Efficient
- Fault-tolerant
- …

You will build an application over this network

Seattle platform

You will use a platform for getting access to computers around the world

Lab logistics

- 4 Labs, 10 points each
- PASS = at least 31/40 points

- Late submissions:
  - within 1 week after the deadline
    → -1 point from your score on that lab

  - within 2 weeks after the deadline
    → -3 points from your score on that lab

  - No submissions accepted 2 weeks after the deadline

Lab deadlines

- **Pre-assignment**: November 7, 23:59
- **Lab 1**: November 14, 23:59
  - Demo: Nov. 10 & 14
- **Lab 2**: November 24, 23:59
  - Demo: Nov. 21 & 24
- **Lab 3**: December 8, 23:59
  - Demo: Dec. 5 & 8
- **Lab 4**: January 8, 23:59
  - Demo: Dec. 15

Use the Lab slots to demo your solutions to us before submission!

Deadlines are hard, but you can submit afterwards with a penalty.

Hand in

- Code
  - Well structured
  - Well documented

- Results
  - In a video
    - 1-2 minutes screencast to demonstrate your results
    - Some screencast software: http://en.wikipedia.org/wiki/Screencast
    - Screencast software in Lab rooms: RecordMyDesktop
  - **Or** in a report (as a pdf file)

If you choose report instead of video

- Include the same information as in the video

- That is
  - Document that your solution works with screenshots and explanations
  - **All stages of the execution** should be included in your documentation

Grading

- Solution: 4 points
- Code structure: 2 points
- Code documentation: 2 points
- Video or report: 2 points

Code Structure and Documentation

- *“Code is written primarily to be read by humans. It has to be acceptable to the compiler too, but the compiler doesn’t care about how it looks or how well it is written.”*

- In your professional life, you will mostly use, fix and improve code that already exists...

Code Structure and Documentation: Guidelines

- Descriptive variable names
  (‘a’, ‘b’, ‘apa’ are not descriptive…)

- Comment on any blocks of code that are doing something subtle or that is not immediately obvious.

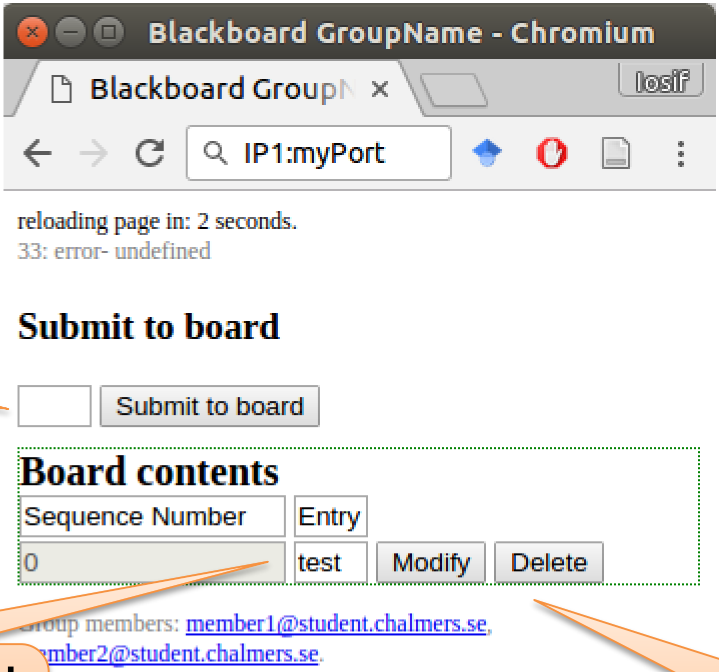
Code Structure and Documentation: Guidelines

- Document **what** each function does (not how), arguments, returned values, side effects etc (see Repy API).

- If you can’t do this in a few sentences, that suggests that you may need to rethink your abstraction.

Lab scheme

A (basic) blackboard should look like this:



reloading page in: 2 seconds.
33: error- undefined

Submit to board

□ Submit to board

Board contents

Sequence Number	Entry		
0	test	Modify	Delete

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

A form to submit text

Some blackboard entries

Options for each entry

Lab scheme

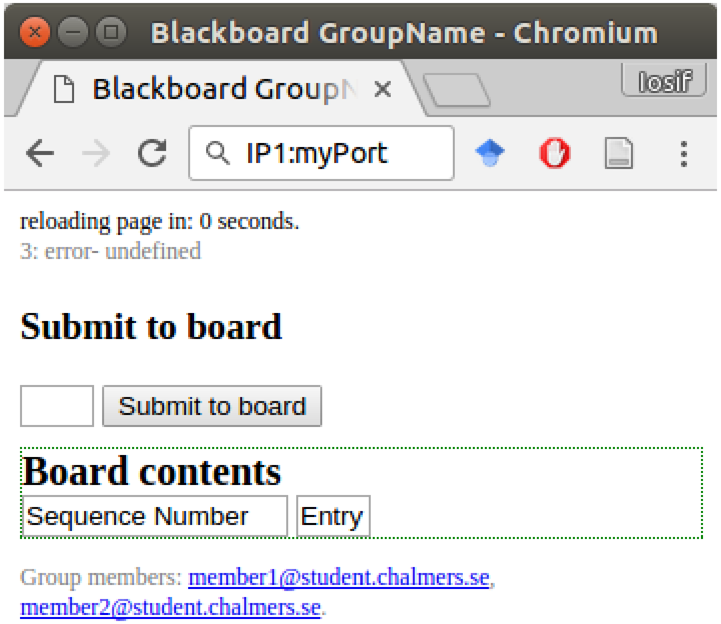
This lab: ***A RESTful* distributed blackboard!**

- A number of machines around the world, each one having each own blackboard

- When a message is written in one board, it should be also propagated to all other boards

Lab scheme

Initially: empty blackboard on all vessels



reloading page in: 0 seconds.
3: error- undefined

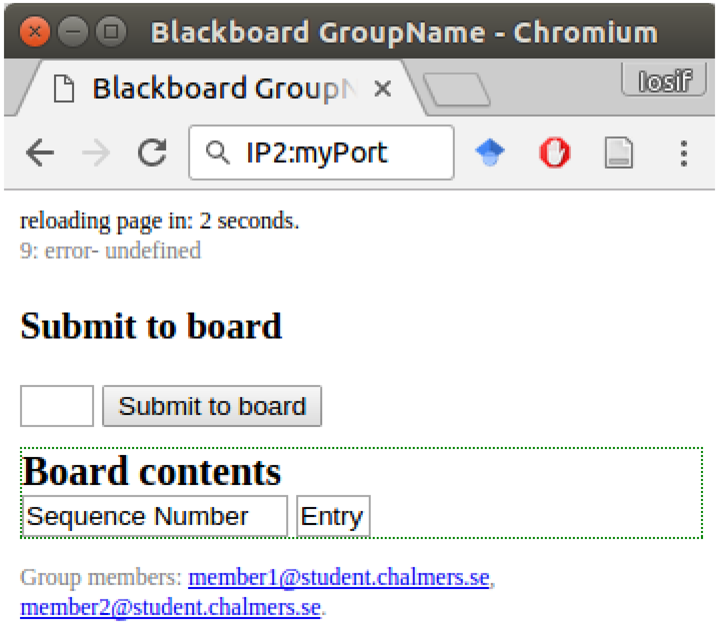
Submit to board

□ Submit to board

Board contents

Sequence Number	Entry

Group members: member1@student.chalmers.se, member2@student.chalmers.se.



reloading page in: 2 seconds.
9: error- undefined

Submit to board

□ Submit to board

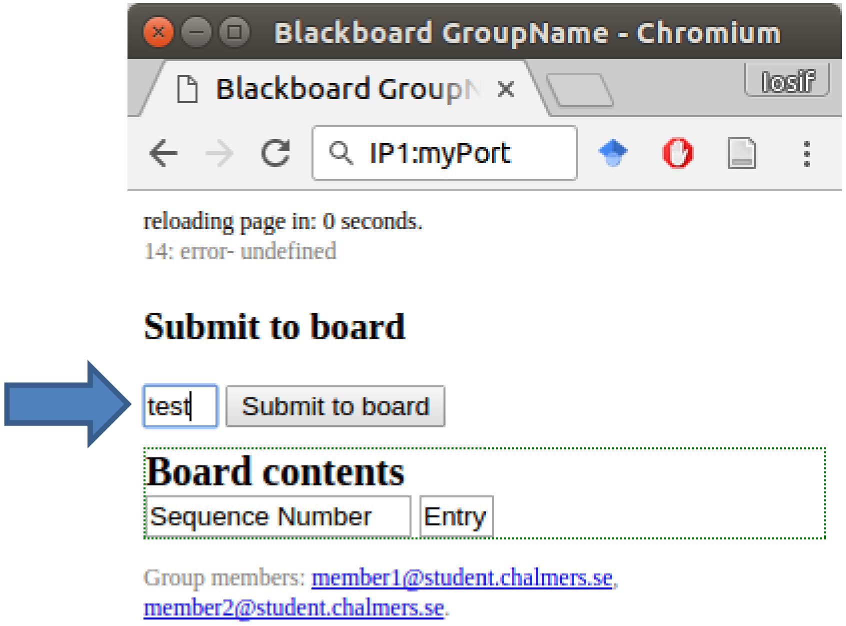
Board contents

Sequence Number	Entry

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

Lab scheme

On vessel 1:
Write text and submit



reloading page in: 0 seconds.
14: error- undefined

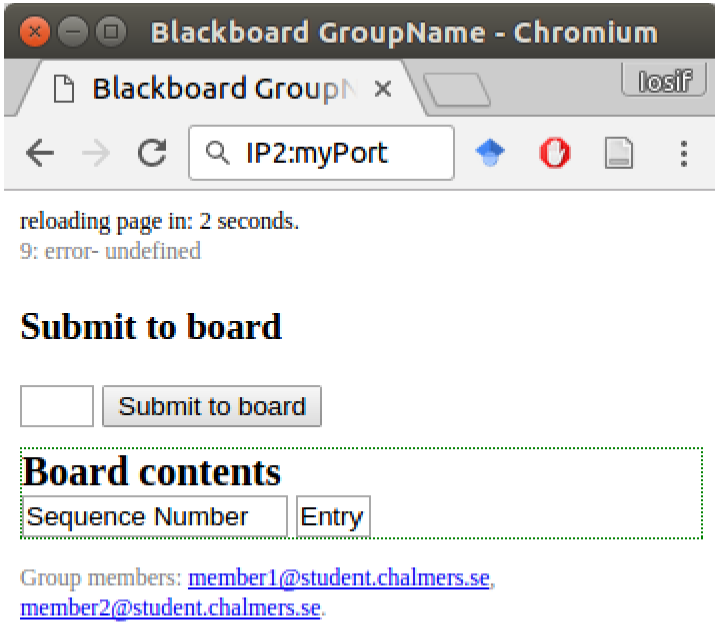
Submit to board

Board contents

Sequence Number	Entry

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

On vessel 2:
No action on the browser



reloading page in: 2 seconds.
9: error- undefined

Submit to board

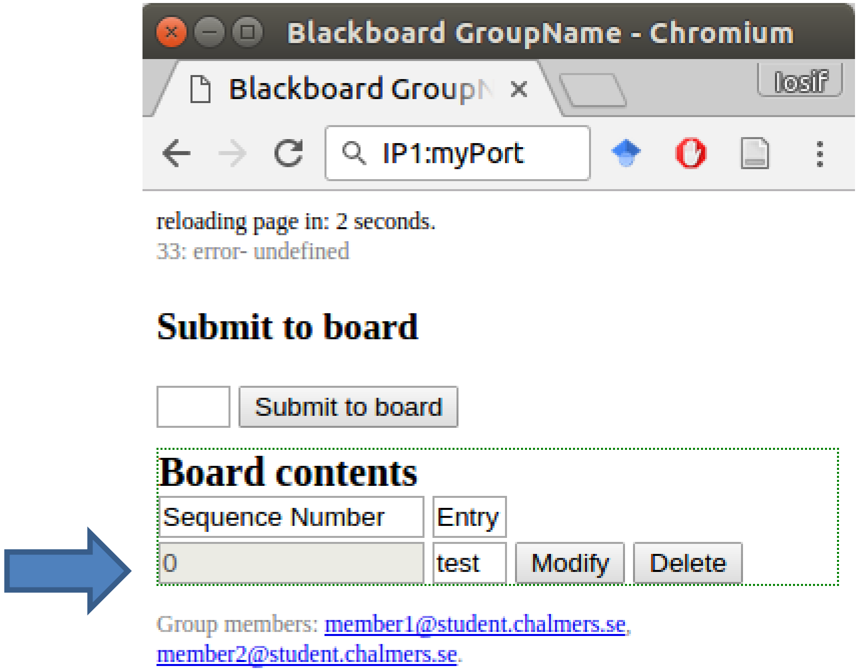
Board contents

Sequence Number	Entry

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

Lab scheme

On vessel 1:
Text appears on the board



reloading page in: 2 seconds.
33: error- undefined

Submit to board

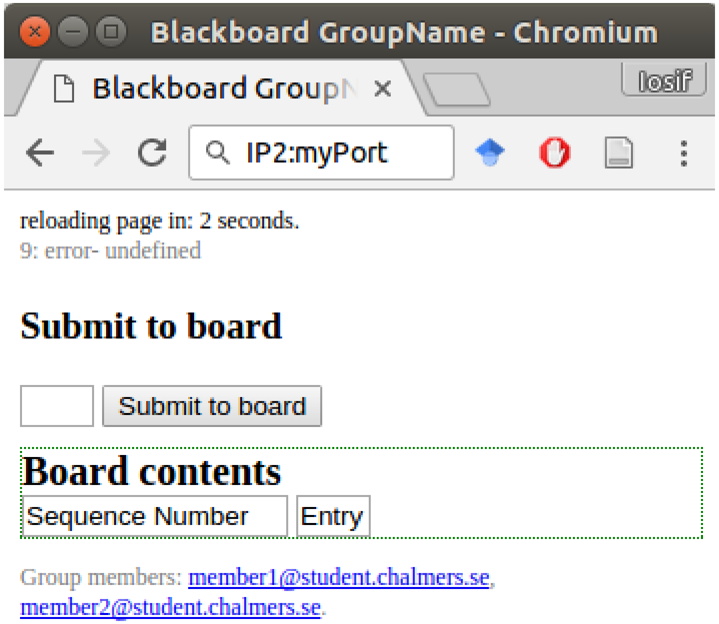
[ ] Submit to board

Board contents

Sequence Number	Entry		
0	test	Modify	Delete

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

On vessel 2:
No action on the browser



reloading page in: 2 seconds.
9: error- undefined

Submit to board

[ ] Submit to board

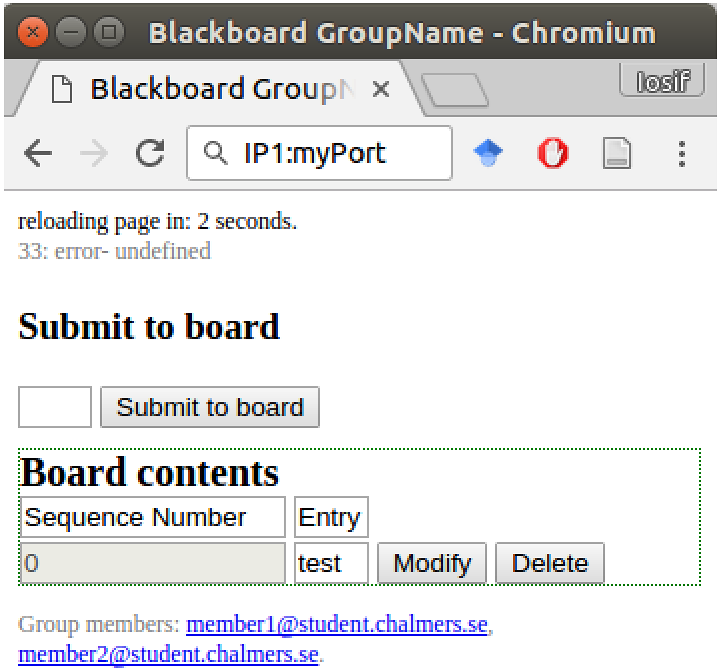
Board contents

Sequence Number	Entry

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

Lab scheme

On vessel 1:
No action on the browser



reloading page in: 2 seconds.
33: error- undefined

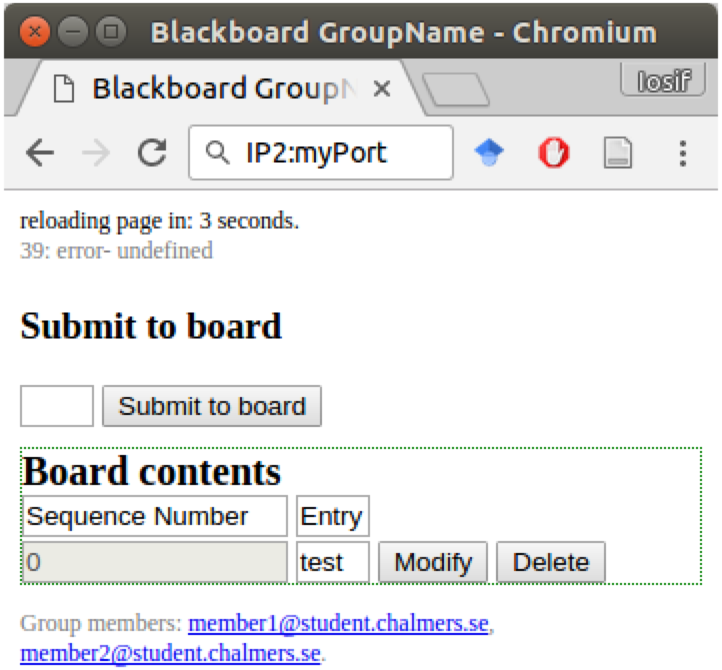
Submit to board

Board contents

Sequence Number	Entry		
0	test	Modify	Delete

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

On vessel 2:
Hit refresh and see post!



reloading page in: 3 seconds.
39: error- undefined

Submit to board

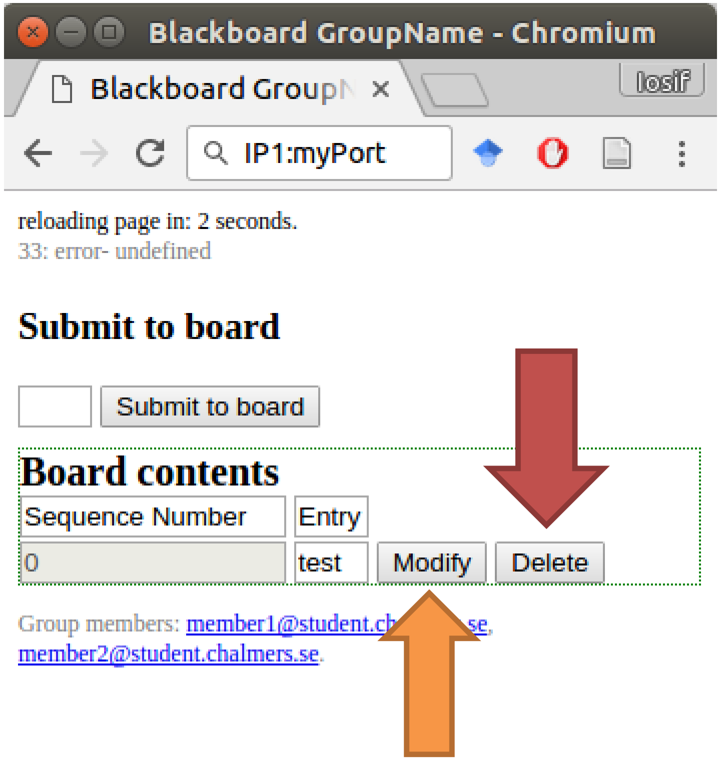
Board contents

Sequence Number	Entry		
0	test	Modify	Delete

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

Lab scheme

A user should also be able to **modify** and a **delete** each post.



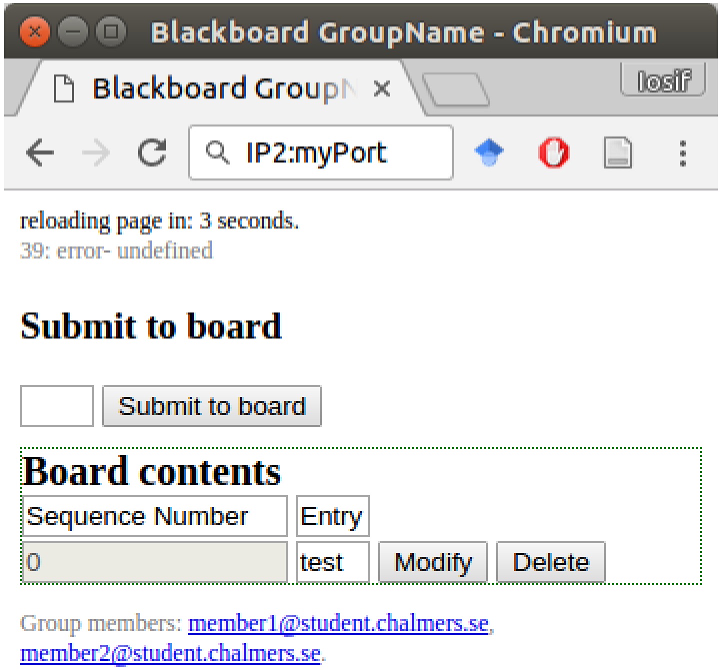
reloading page in: 2 seconds.
33: error- undefined

Submit to board

Board contents

Sequence Number	Entry		
0	test	Modify	Delete

Group members: member1@student.chalmers.se, member2@student.chalmers.se.



reloading page in: 3 seconds.
39: error- undefined

Submit to board

Board contents

Sequence Number	Entry		
0	test	Modify	Delete

Group members: member1@student.chalmers.se, member2@student.chalmers.se.

Lab scheme

Over the course we will make this blackboard more

- Reliable
- Consistent
- Efficient
- Fault-tolerant
- …

Agenda

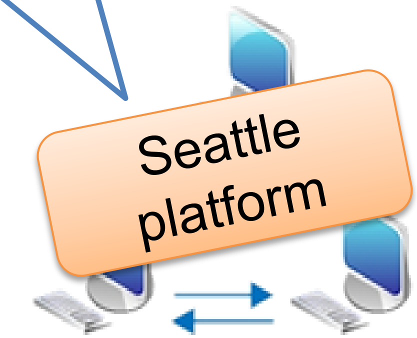
- Seattle platform
- Lab overview
- Lab 1

Lab overview

During the labs you will make this application:

- Reliable
- Consistent
- Efficient
- Fault-tolerant
- …

You will build an application over this network



Seattle platform

You will use a platform for getting access to computers around the world

Lab 1: Make it work

Some hints:

- Keep a list of all vessels in each vessel
  - (We know, this is not a scalable design, you will work on this aspect in the following labs)
- Upon a post
  - Send the update to all other vessels
  - To be shown (along with the previous board content) when a user visits again the page from the browser
- Note
  - You can use TCP connections between the vessels to send the updates
- Roughly 80 lines of code (depends on HTML)

Task 1: make it work

- Demonstrate that your distributed blackboard works by submitting a 1-2 minutes video or a report
  - Document: Do 2 or 3 posts and show them appearing on the other blackboards
  - Use at least 8 vessels/blackboards
  - Hint: show the browser windows and (optionally consoles) next to each other on your screen
  - Record your screen and document what is happening by using your mouse and your voice
    - No video editing, cutting, etc. required

Task 2: delete/modify

- A user must be able to delete or modify a post (we provide the HTML interface)
- Once a post is either modified or deleted, a vessel should propagate this change to other vessels

- Submit a 1-2 minutes video/report that demonstrates this functionality

Task 3: Consistency?

- Can it happen that two vessels show different blackboards?
  - Even when all data was reliably send to all vessels, and then we hit refresh afterwards

  - Hint (optional): Use a script to do “concurrent posts” to your blackboards! (use the curl command)

  - Submit max 1-2 minutes video or a report
    - Explaining your thoughts
    - If you can make it happen: document it in your video or report

Hand in

- Code
  - Well structured
  - Well documented
- Task 1
  - Video or report
- Task 2
  - Video or report
- Task 3
  - Video or report

What you will get from us

- A sample HTML file
  - Change it if you want (totally optional)
- A skeleton Repy file
  - Based on Hello world from Repy tutorial
- Answers to your questions in the labs
- Demo slots

Pitfalls & Hints

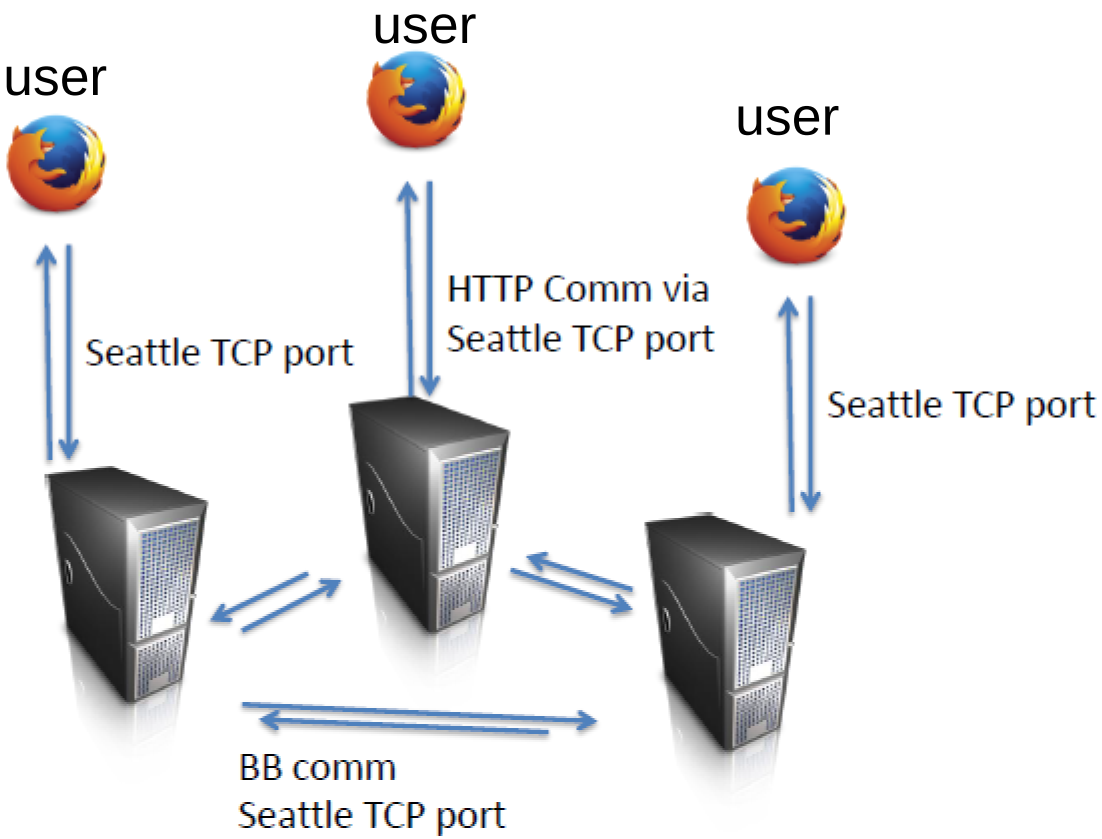
- Repy & Python
  - No global variables across functions!
    - There is a solution, cf Repy API.
  - Python has excellent built-in string manipulation functions.
  - Python is Dynamically & Strongly typed. Objects have types, not the variables.

    ```
    x = 10
    x = “Hello”
    ```

    The above sequence is valid.
- HTTP
  - We don’t care if the board content is encoded: “Hello+world” is considered a valid entry

Pitfalls & Hints

- “User interface” vs Communication



Questions?