



Distributed Systems - HS 2014

Assignment 3

Hông-Ân Cao

hong-an.cao@inf.ethz.ch

Outline

- Review of logical time and UDP
 - Causality
 - Lamport Timestamps
 - Vector Clocks
- Assignment 3
 - Task 1
 - Task 2
 - Task

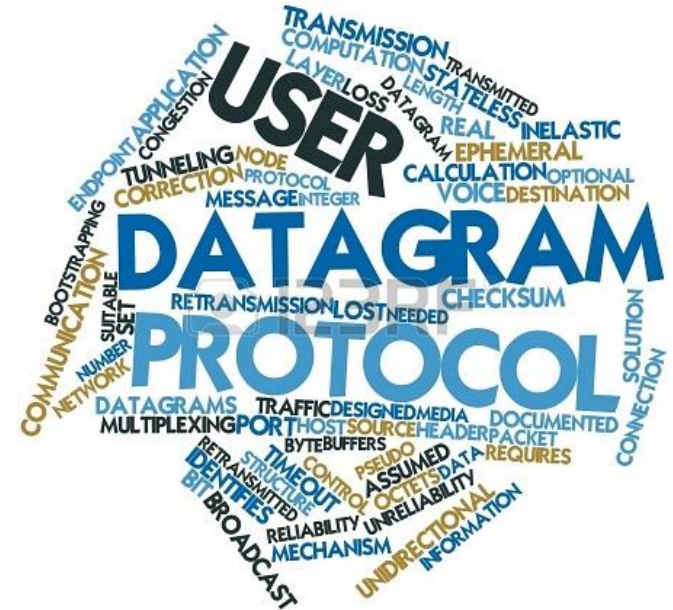
Dates:

Start: October 20, 2014

End: November 3, 2014 09:00 AM (CET)

The User Datagram Protocol

- Simple transmission model
 - No hand-shakes, ordering, data integrity
 - Datagrams delayed (out of order), duplicates, missing



31	00:06:23.432149000	10.33.47.177	10.40.4.44	DNS	72 Standard query 0xb220 A www.ietf.org
32	00:06:23.432569000	10.40.4.44	10.33.47.177	DNS	88 Standard query response 0xb220 A 12.22.58.30
33	00:06:23.471947000	10.33.47.177	208.64.200.203	UDP	126 Source port: 51099 Destination port: 27018
34	00:06:23.492935000	10.33.47.177	12.22.58.30	TCP	66 56033 > http [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SA
35	00:06:23.495665000	12.22.58.30	10.33.47.177	TCP	66 http > 56033 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=
36	00:06:23.495708000	10.33.47.177	12.22.58.30	TCP	54 56033 > http [ACK] Seq=1 Ack=1 win=65700 Len=0
37	00:06:23.495808000	10.33.47.177	12.22.58.30	HTTP	428 GET / HTTP/1.1

Frame 31: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface 0

Ethernet II, Src: Micro-St_01:58:35 (8c:89:a5:01:58:35), Dst: Cisco_ec:e9:3f (28:94:0f:ec:e9:3f)

Internet Protocol Version 4, Src: 10.33.47.177 (10.33.47.177), Dst: 10.40.4.44 (10.40.4.44)

User Datagram Protocol, Src Port: 49927 (49927), Dst Port: domain (53)

Source port: 49927 (49927)

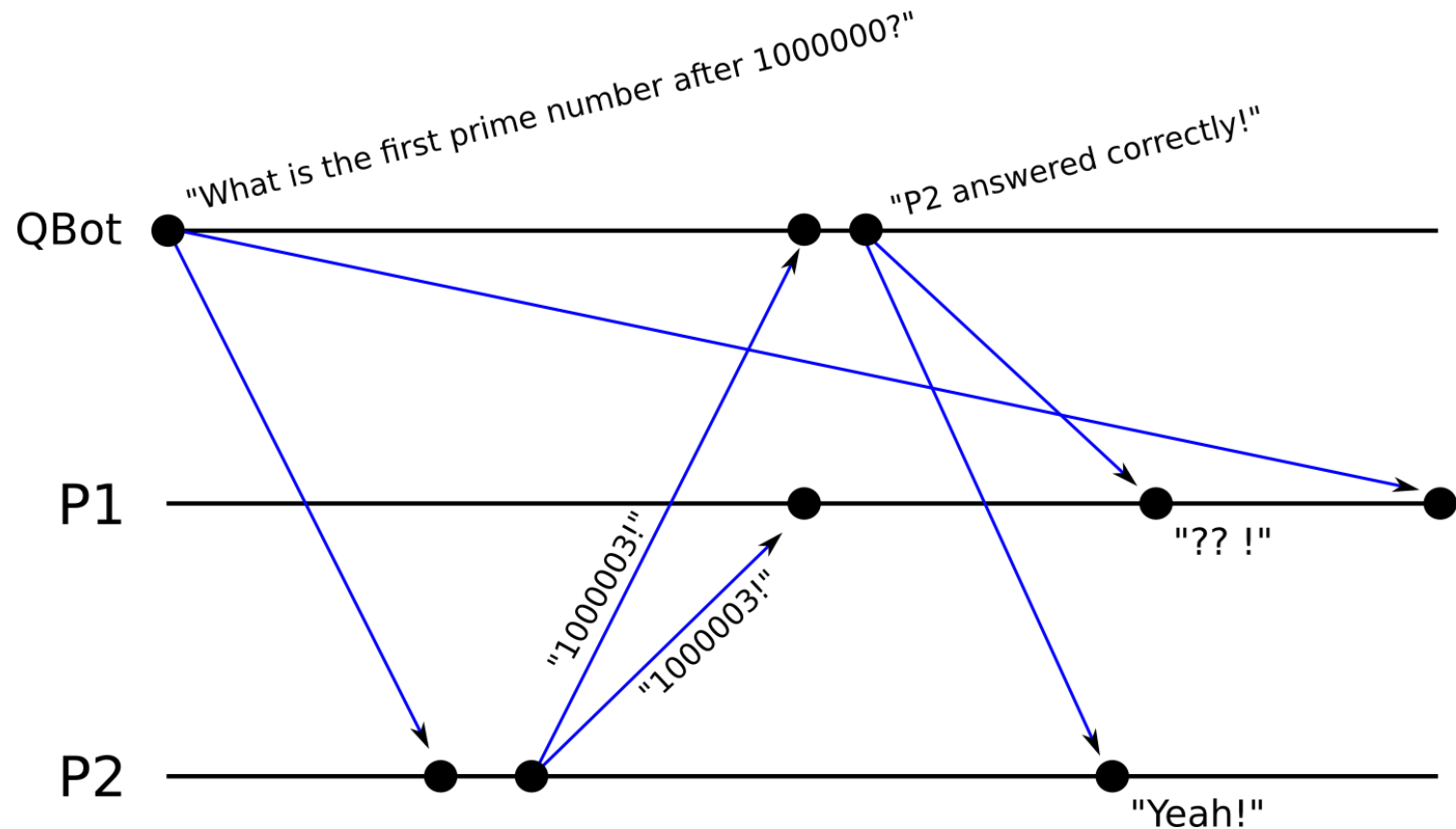
Destination port: domain (53)

Length: 38

Checksum: 0x485d [validation disabled]

Domain Name System (query)

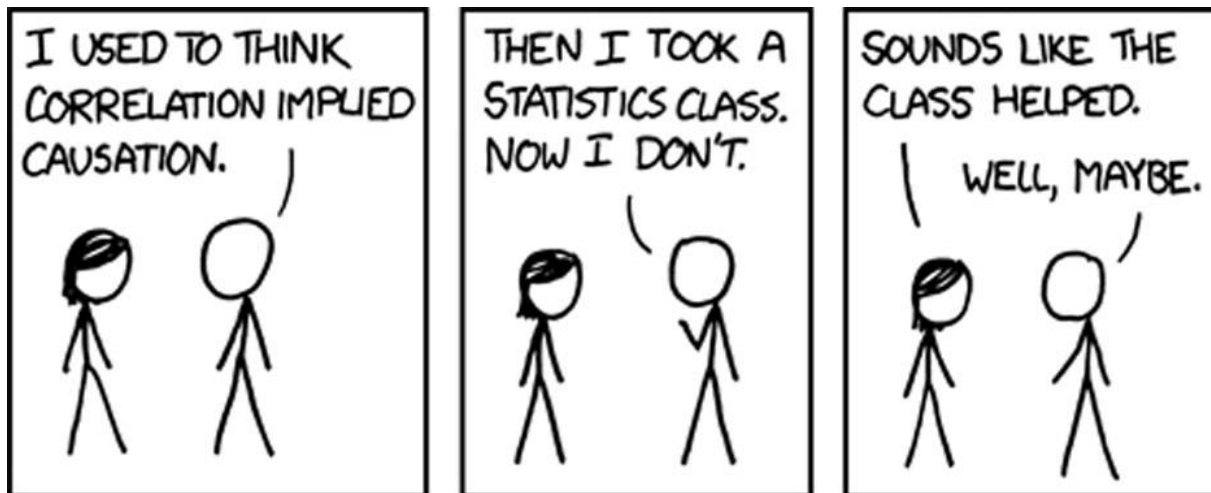
UDP Effects



Causality

- Interesting property of distributed systems
- Causal relationship \prec ("happened before")

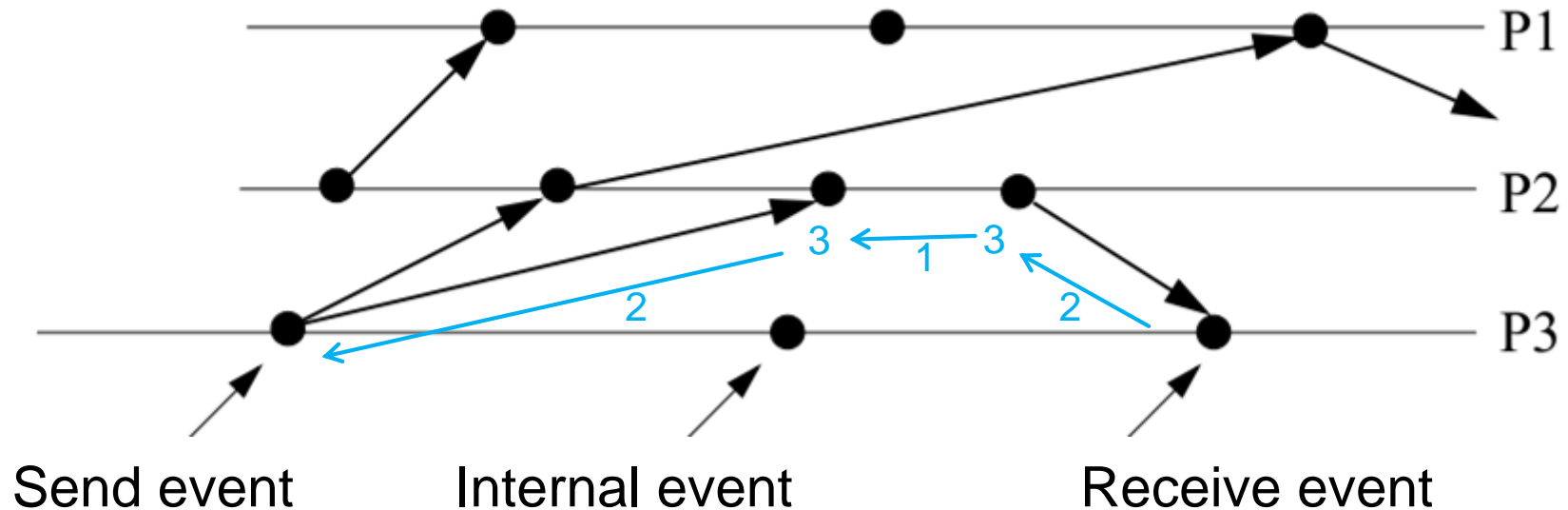
$x < y$ iff ($(x, y$ on same process, x happens before y) or
 $(x$ is sent and y is correspondingly received) or
 $(transitivity)$)



Causality

$x < y$ iff ($(x, y$ on same process, x happens before y) or $(x$ is sent and y is corresponding receive) or $(transitivity)$)

1
2
3



Software Clocks

- Ideal real time → Transitive, dense, continuous, etc.
- Logical time → Cheap version of real time
 - **Lamport Timestamps**
 - **Vector Clocks**
 - Matrix Clocks

Lamport Timestamps

- Using a single clock value

- Local Event:

Local clock tick

- Send Event:

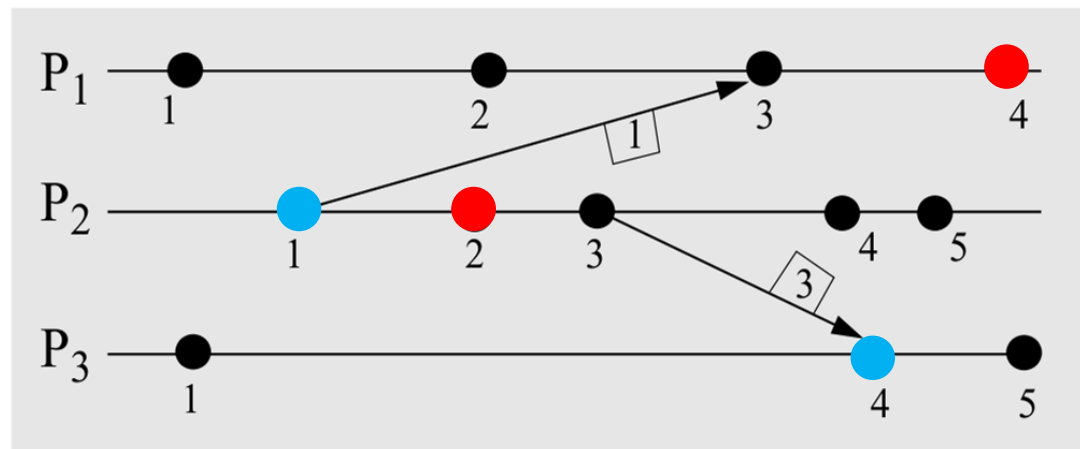
Attach local clock value

- Receive Event:

$\max(\text{local clock, message clock})$

- Satisfies clock consistency condition:

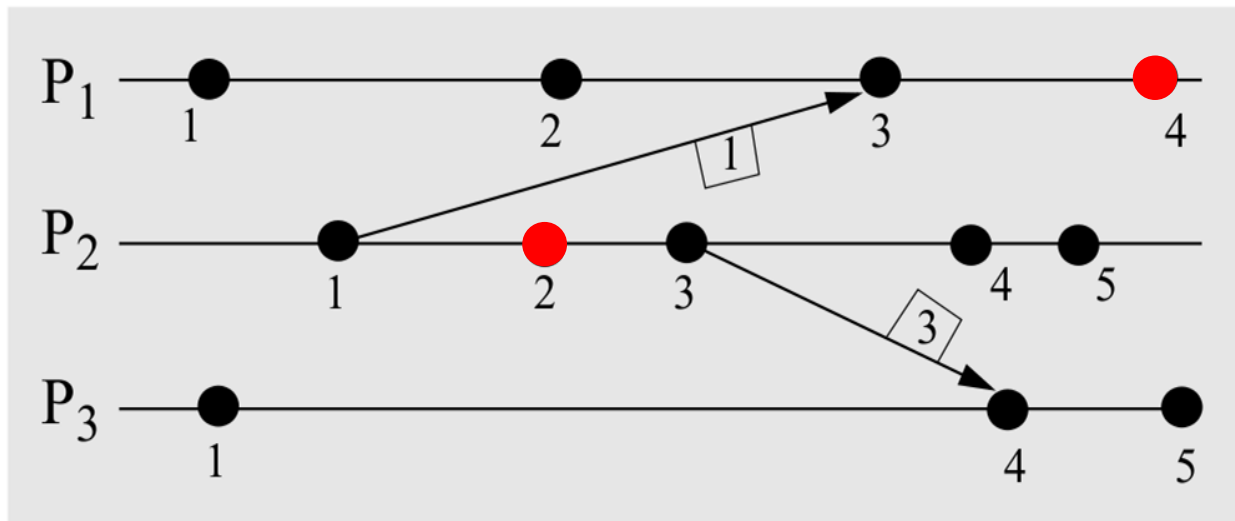
$$e < e' \rightarrow C(e) < C(e')$$



Lamport Timestamps

- Lamport Timestamp does **not** satisfy **strong clock consistency condition**

$$e < e' \leftrightarrow C(e) < C(e')$$

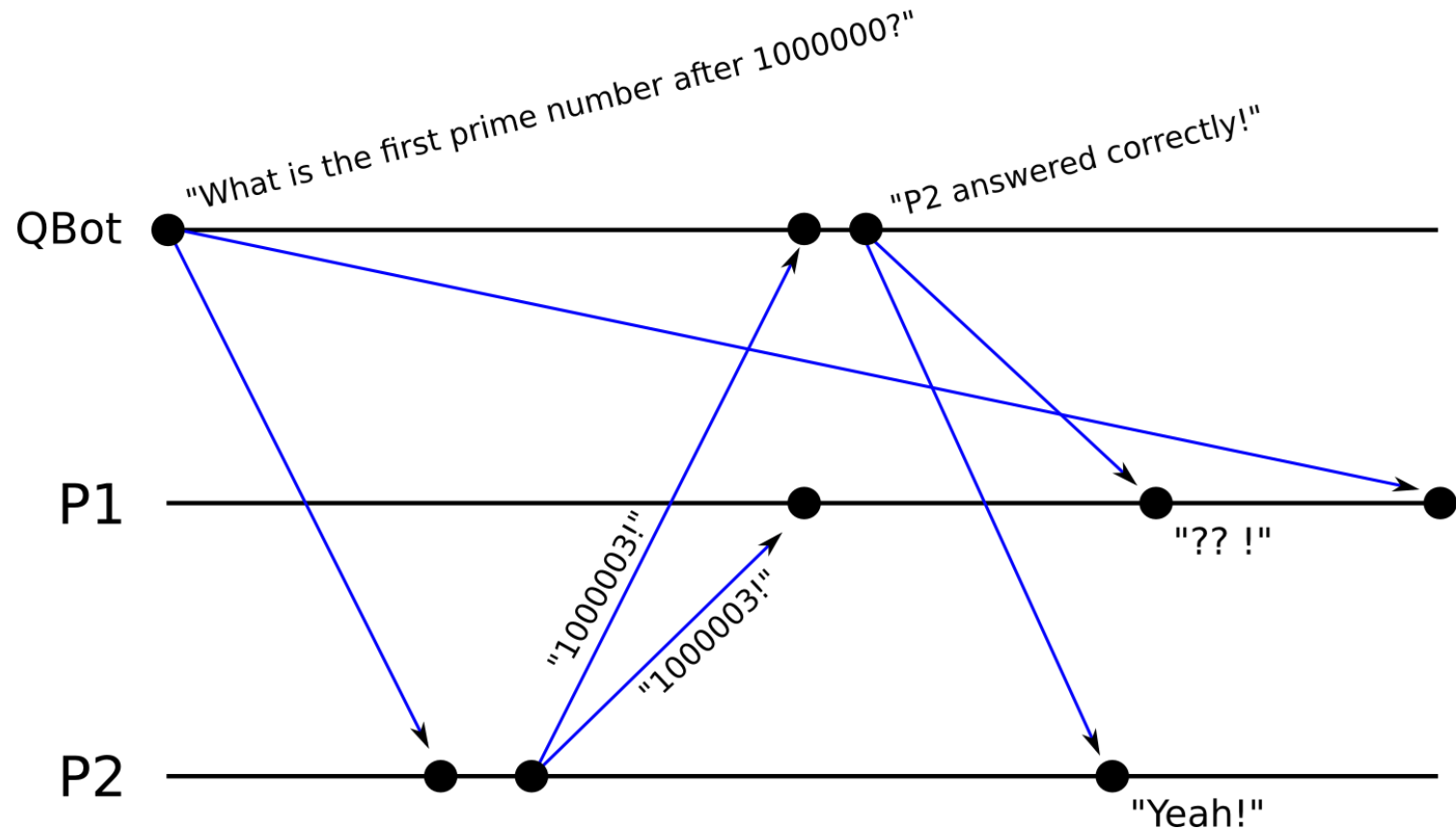


Vector Clocks

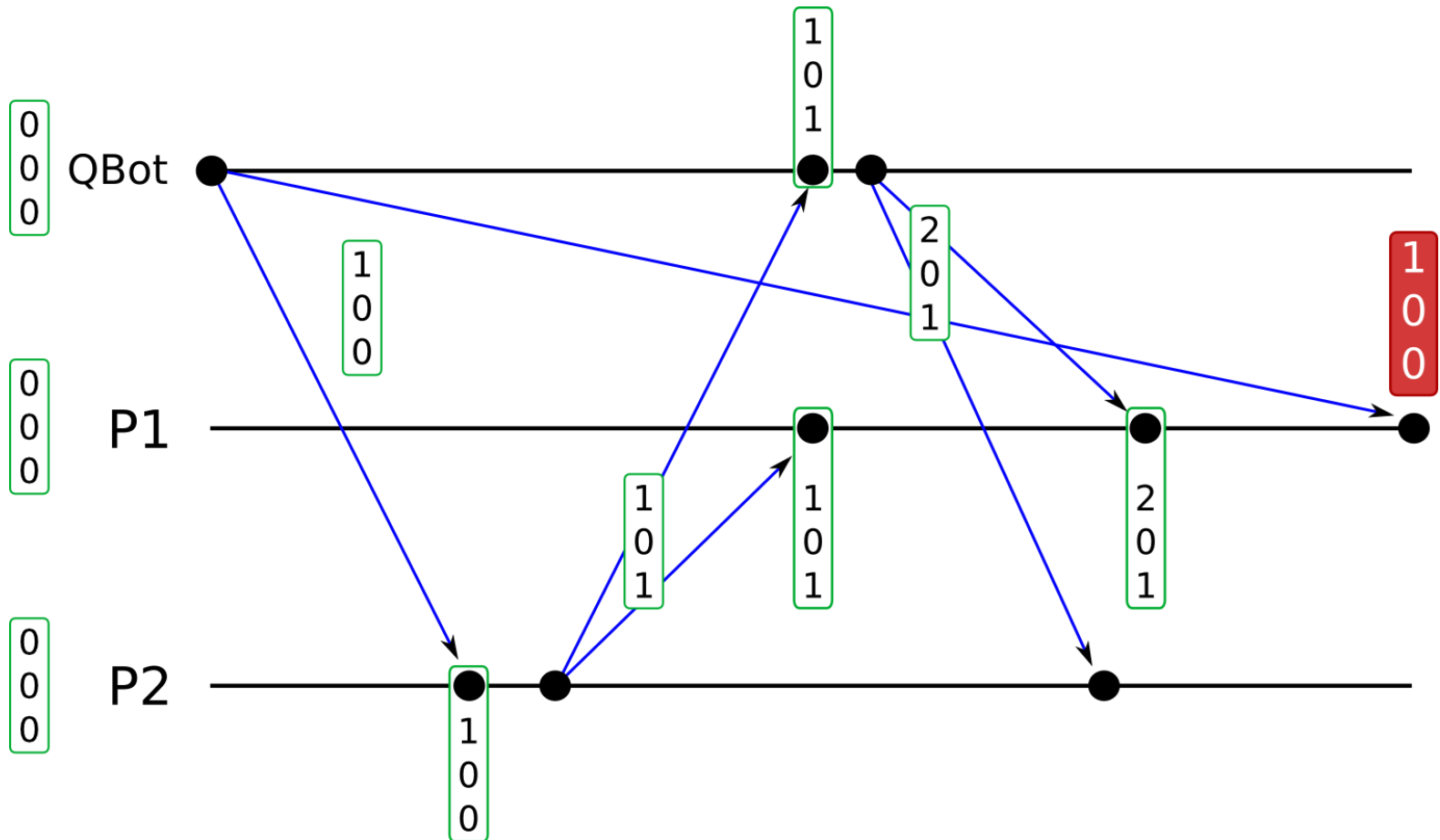
- Refining Lamport Timestamps → Processes keep one counter per process
- Does satisfy strong clock consistency condition!

$$e < e' \leftrightarrow C(e) < C(e')$$

Vector Clocks



Vector Clocks

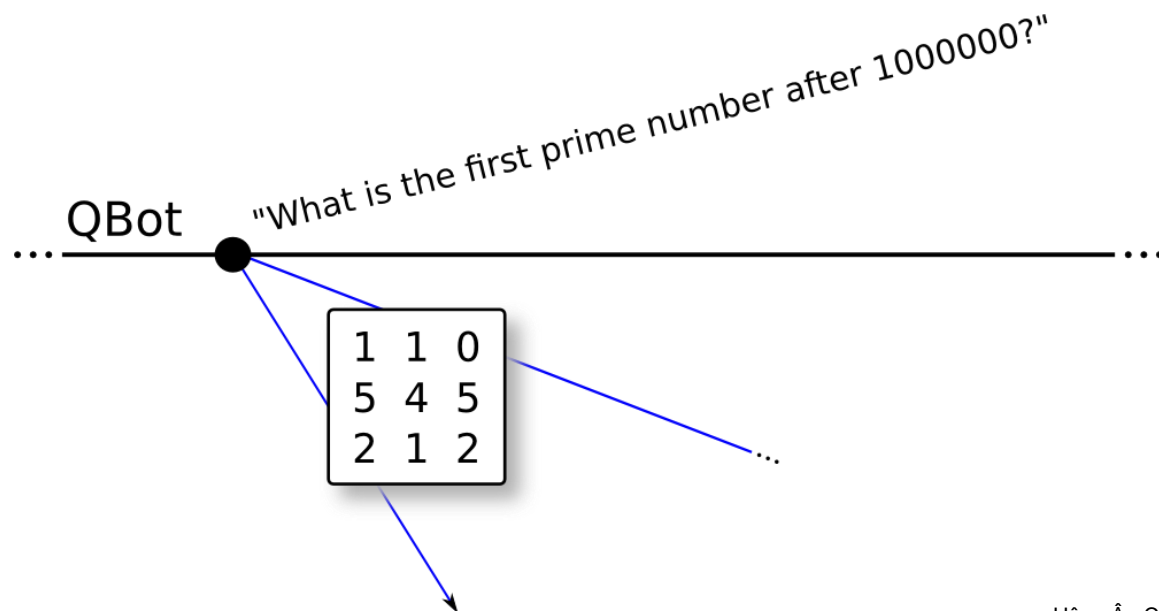


Vector Clocks

“Process i stores information on what it thinks about the local time of processes $(1, \dots, n)$.”

Matrix Time (not in the assignment)

- Refining Vector Clocks → Processes keep n counters per process
- “Process i stores information on what it believes that processes $(1, \dots, n)$ think about the local time of processes $(1, \dots, n)$.”



Outline

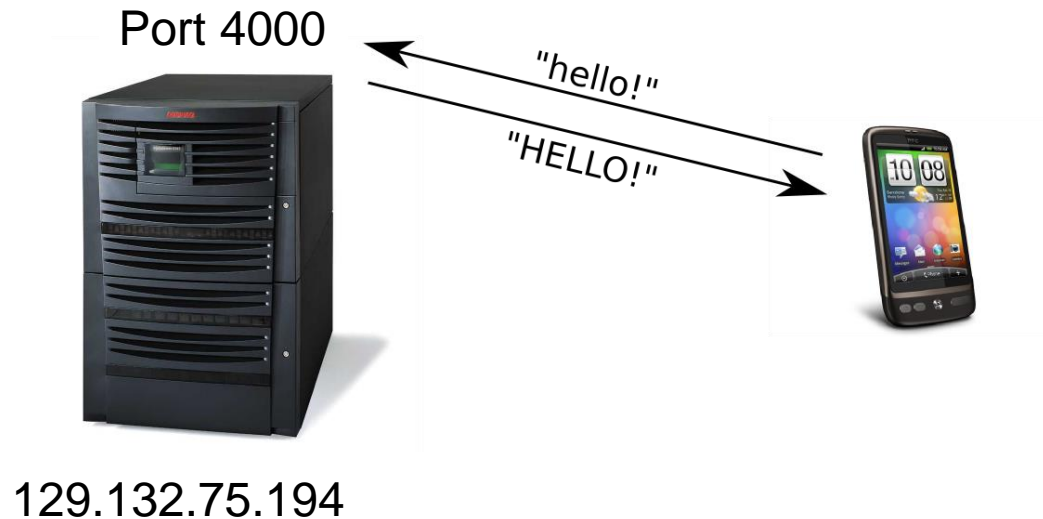
- Review of logical time and UDP
 - Causality
 - Lamport Time
 - Vector Time
- Assignment 3
 - Task 1
 - Task 2
 - Task 3

A Mobile, Causal, UDP-based Chat-Application

- Task 1: Getting familiar with datagrams
- Task 2: Starting the conversation + (Lamport Timestamps + Vector Clocks) to overcome the desequencer
- Mini-Test

1. Getting familiar with datagrams

- Communicate with server at 129.132.75.194:4000 using UDP
- Provides "capitalization" service



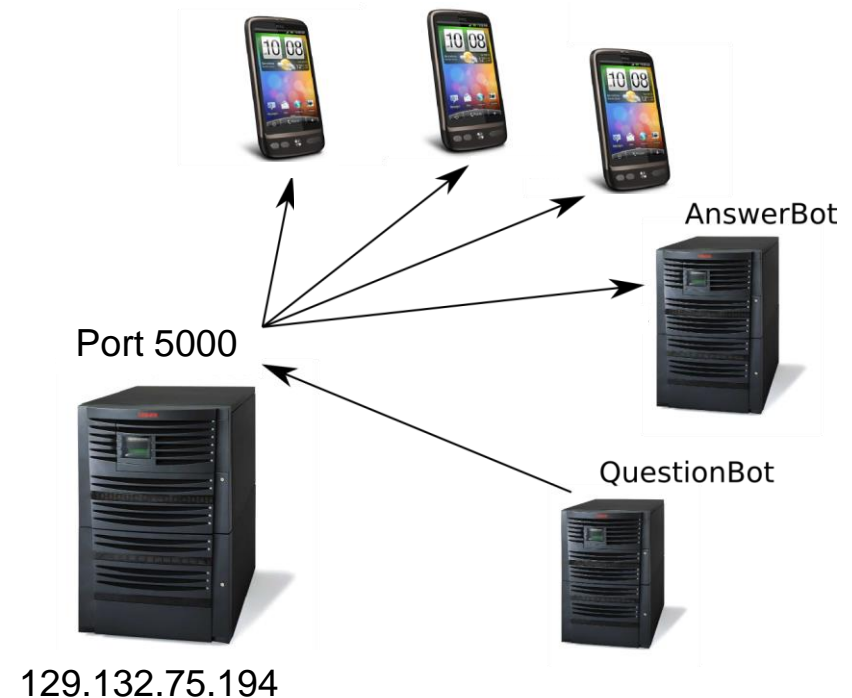
2. This is not a chat server...



Source: <http://www.vulgart.be/?tag=surrealisme>

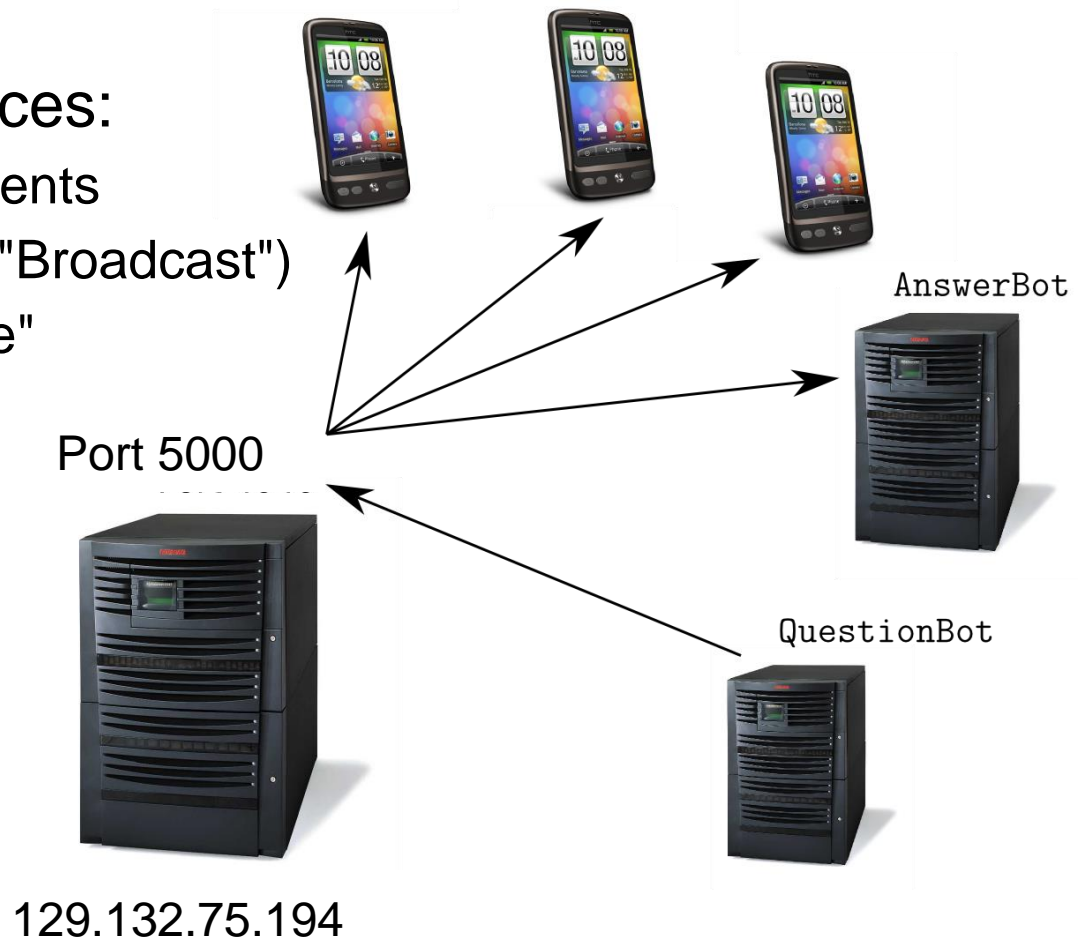
2. Side Note: Encoding Time

- Lamport Timestamps → Need to encode single timestamp
- Vector Time → Need to encode multiple timestamps
- You will find the VectorClock class that uses the underlying `HashMap<int, int>` or dictionary to identify vector times.
- An underlying int is associated to the lamport timestamps in the Lamport class.



2. Side Note: System Setup

- 129.132.75.194 services:
 - (De-) Registration of clients
 - Distributes messages ("Broadcast")
 - De-sequencing "service"



2. JSON Protocol on 129.132.75.174:5000

→ {"cmd": "register", "user": "caohl"}

← {"index": 2, "init_time_vector": {"2": 0, "1": 70, "0": 71}, "init_lamport": 74, "cmd": "register", "status": "success"}

→ {"cmd": "get_clients"}

← {"cmd": "get_clients", "clients": {"0": "QuestionBot", "1": "AnswerBot", "2": "caohl"}}

→ {"cmd": "info"}

← {"cmd": "info", "text": "I am an advanced UDP server that is running at port 5000 to provide a de-sequencing service for Android UDP chatting programs..."}

→ {"text": "hello", "cmd": "message", "time_vector": {"2": 1, "1": 70, "0": 71}, "lamport": 75}

← {"cmd": "message", "status": "success"}

→ {"cmd": "deregister"}

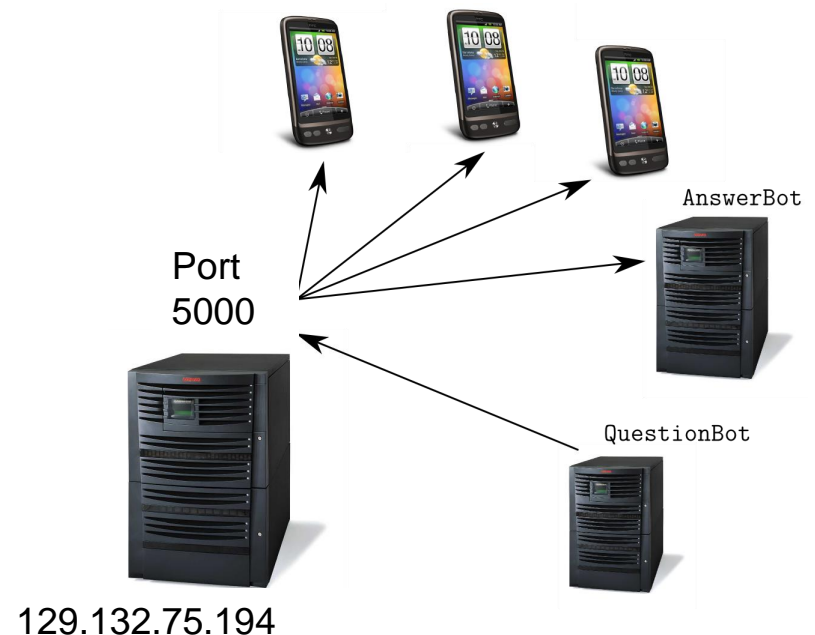
← {"cmd": "deregister", "status": "success"}

Everyone else receives:

← {"cmd": "notification", "text": "caohl has left (index 2)"}

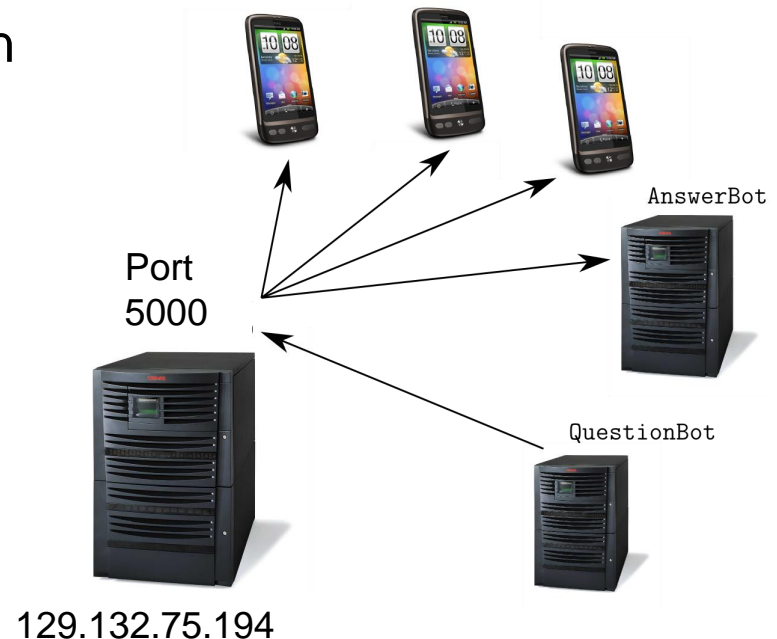
2. Overcoming the Desequencer

- UDP chat with server port 5000 (Use port 4999 for testing)
- Causality preservation via Lamport Timestamps
- Lamport Timestamp stored in integer in field "Lamport"



2 Overcoming the Desequencer

- UDP chat with server on port 5000
- Causality preservation via **Vector Clocks**
- Own timestamp in **i^{th} time vector index**
 - i assigned by server upon registration



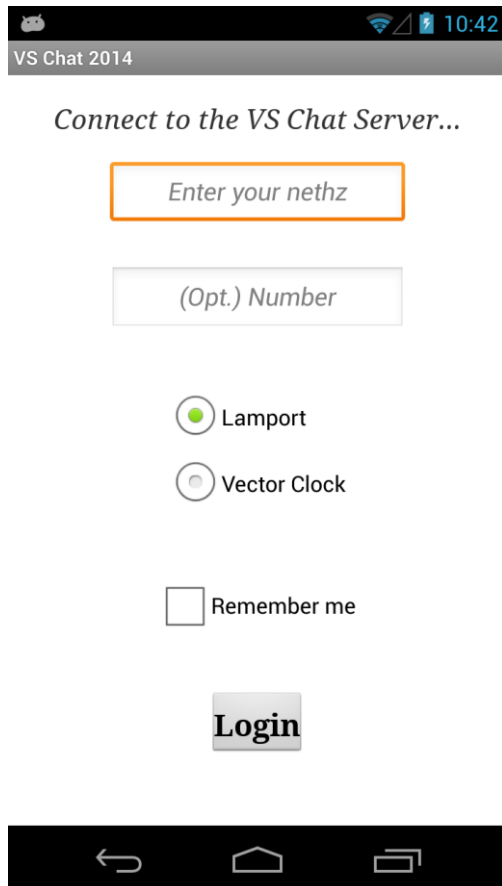
2. Send/Receive/Tick policies

- Multiple ways to implement vector clock ticking
 - Tick only when sending, after sending [vs. before sending]
 - Tick when receiving and sending, after sending [vs. before sending]
- QuestionBot's and AnswerBot's policy:
 - Tick only when sending, before sending
 - Example: Message from process 2 with timestamp [4,5,1] means:
"Before receiving me, you should already have received and delivered 4 messages from process 1, 4 (!) from process 2 and 1 message from process 3!"
"If you did not receive these, wait before delivering me!"
 - What if a message is lost?

2. Issues/Considerations

- **Maybe try it in pure Java first...**
 - Better debugging... (e.g. exceptions are actually displayed)
 - Faster and more convenient
- **Forward port to emulator**
<http://stackoverflow.com/questions/5064304/how-can-i-forward-my-localhost-ip-address-to-an-android-emulator>
- **Use the VPN (you need to be on the ETH 129.132.0.0/16 subnet!)**
- Lots of groups interact via the chat server use the server at 4999 first
 - Potential problem → some groups non-compliant
 - Results could be → Everyone's code crashes...
 - Solution → Tag your messages (e.g. using your group's number) and/or only consider your own messages

2. Design



VS Chat 2014

Connect to the VS Chat Server...

Enter your nethz

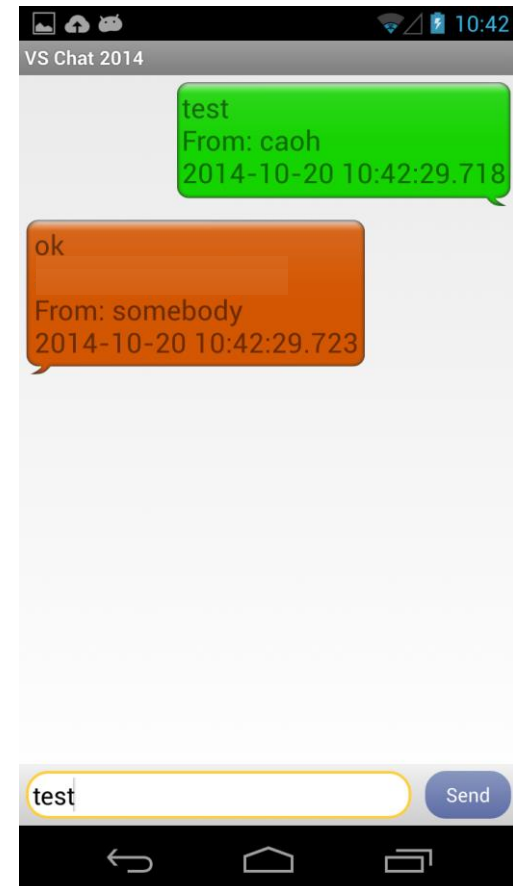
(Opt.) Number

☒ Lamport

☐ Vector Clock

☐ Remember me

Login



VS Chat 2014

test
From: caoh
2014-10-20 10:42:29.718

ok
From: somebody
2014-10-20 10:42:29.723

test

Send

3. Mini-Test

- When exactly are 2 Vector Clocks causally dependent?
 - Does your application allow "purely local" events? Do they trigger a clock tick?
 - Does a local clock tick happen before or after sending a message?
 - How are receive events handled? Do they trigger local clock ticks?
- Dynamically joining/leaving clients
 - Read the paper "Dynamic Vector Clocks"
 - Think about the approach described there
- **Cover this in your answers!**

The End

