AARHUS UNIVERSITET

Software Engineering and Architecture

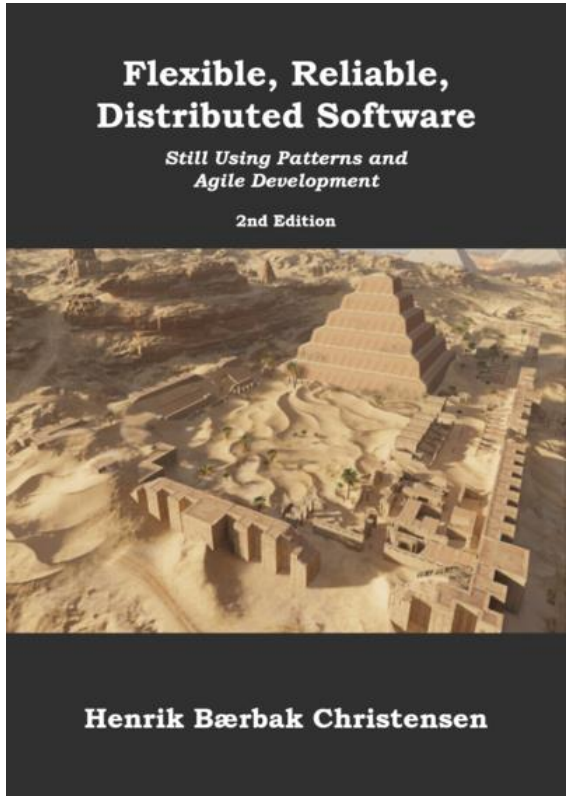
Distributed Systems

An Introduction

Intro…

- Distributed Computing is the last major SWEA topic
  - Our perspective: *Programming and Pattern Perspective*

- Curriculum: My second book ☺
  - Confusion: Looks much the the first…
  - Get it from **https://leanpub.com/frds**
    - For the price of a box of beer…
  - *Yes, I like pyramids !*



Distributed System

Definition: Distributed System

A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages. (Coulouris, Dollimore, Kindberg, and Blair 2012)

- Why?
  - To speed up computation
    - Google search, machine learning, and (a few) other cases
  - ***To share information***
    - Everything else! (Slight exaggeration!)

Limitations

- Distributed systems and distributed computing is a…

… vast subject area !!!

- We will limit ourselves to a ”niche”

Client-server Architectures using Remote Method Invocation

… this niche covers *a lot of* systems in practice ☺

And Limiting ourselves

- Even that is

… difficult to make!!!

- … because it must be
  - Highly available, performant, and secure
- And that is topics in advanced *software architecture*

We will only consider *happy path:*
All computers and networks are working;
Few users and none that are malicious

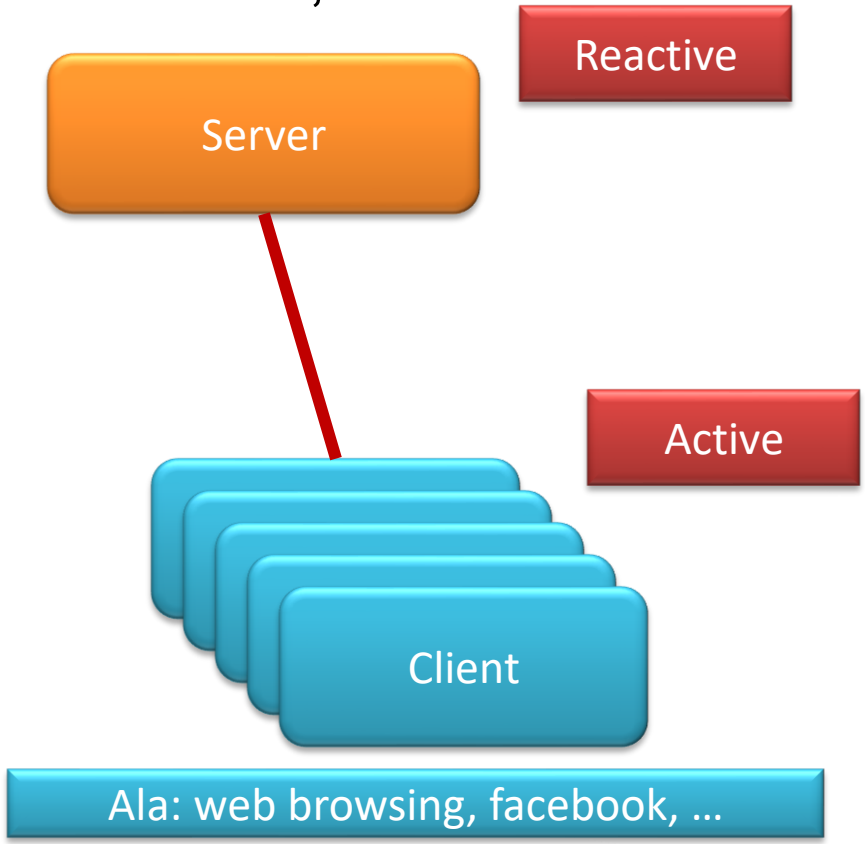
AARHUS UNIVERSITET

Client-Server

- You all know ‘client-server’ architectures, but…

Client-server architecture Two components need to communicate, and they are independent of each other, even running in different processes or being distributed in different machines. The two components are not equal peers communicating with each other, but one of them is initiating the communication, asking for a service that the other provides. Furthermore, multiple components might request the same service provided by a single component. Thus, the component providing a service must be able to cope with numerous requests at any time, i.e. the component must scale well. On the other hand, the requesting components using one and the same service might deal differently with the results. This asymmetry between the components should be reflected in the architecture for the optimization of quality attributes such as performance, shared use of resources, and memory consumption.

The CLIENT-SERVER pattern distinguishes two kinds of components: clients and servers. The client requests information or services from a server. To do so it needs to know how to access the server, that is, it requires an ID or an address of the server and of course the server’s interface. The server responds to the requests of the client, and processes each client request on its own. It does not know about the ID or address of the client before the interaction takes place. Clients are optimized for their application task, whereas servers are optimized for serving multiple clients[3].

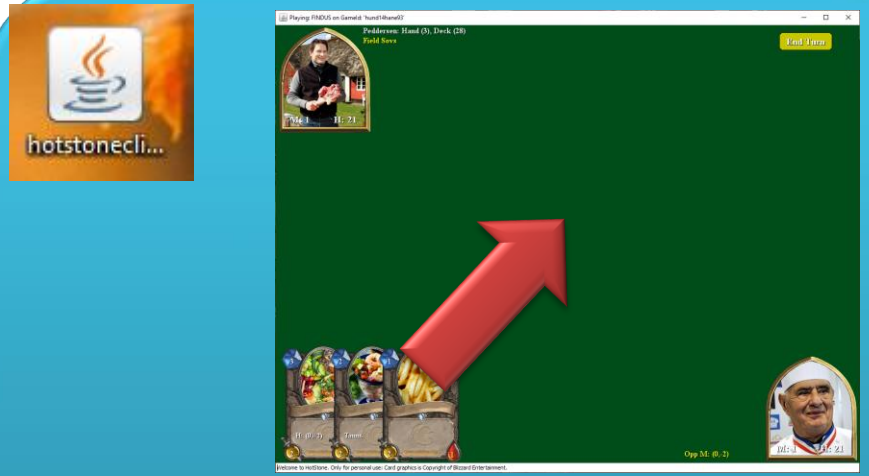


[3]Paris Avgeriou and Uwe Zdun, “Architectural patterns revisited - a pattern language”, In 10th *European Conference on Pattern Languages of Programs* (EuroPlop), Irsee, 2005.

Client-Server

- One big difference from all you have been doing up until now…
  - You have been building “programs” = all behavior in one ‘unit’
- A client-server system consists of **two programs**
- The *client* program: The one the user runs
  - Communicating with…
- The *server* program: Well hidden in some server room
  - WarpTalk – the server is provided by Clemens

Or Visually



game.playCard(Findus, ff);

Client program

game.playCard(Findus, ff);

Server program

Or Visually

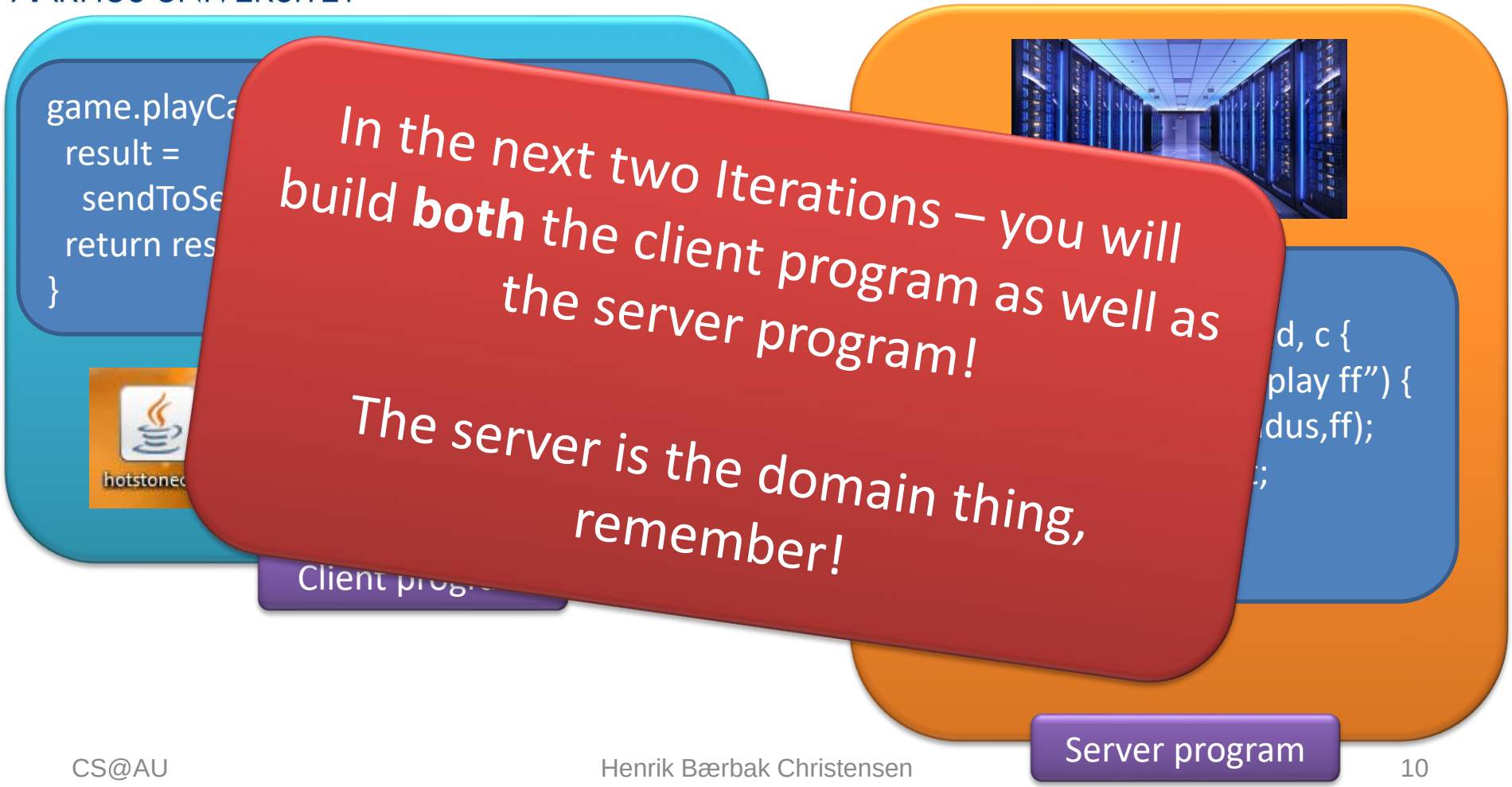
```
game.playCard(Findus, c) {
  result =
    sendToServer("Findus tries to play ff");
  return result;
}
```

Client program

```
Await incoming command, c {
  if (c == "Findus tries to play ff") {
    r = game.playCard(Findus,ff);
    send 'r' back to client;
  } else ...
```

Server program

Or Visually



In the next two Iterations – you will build **both** the client program as well as the server program!

The server is the domain thing, remember!

One Word of Caution

- We will ***happily disregard security !!!***

- Security is so important that we ignore it!
  - Because the real security techniques is one big set of *hard bindings and strong coupling*
    - You need certificates that tie you to a specific DNS name
      - Certificate stores, key pair generation, trust chains, yaga yaga
    - Quite a lot of extra coding and makes experiments difficult

- ***Morale: Add that stuff for real production usage !***

The History

- Birrell and Nelson, 1984:
  - “allow calling procedure on remote machines”
  - A calls procedure f on B means
    - A suspends, information on f is transmitted to B
    - B executes the f procedure
    - B sends the result back to A
    - A resumes

AARHUS UNIVERSITET

Grounding Example

TeleMed

Inspired by Net4Care:

https://baerbak.cs.au.dk/net4care/

Case

- Demographic challenges
  - 2009: 70% of public health expenditure goes to chronic diseases
  - 2040: 100% more elderly
- Geographical challenges
  - Larger, fewer hospitals
  - Fewer general practitioners
- Leads to a need for ***telemedical*** solutions
  - *ICT-supported healthcare services where some of the people participating in service delivery are not co-located with the receiver of the service*

AARHUS UNIVERSITET

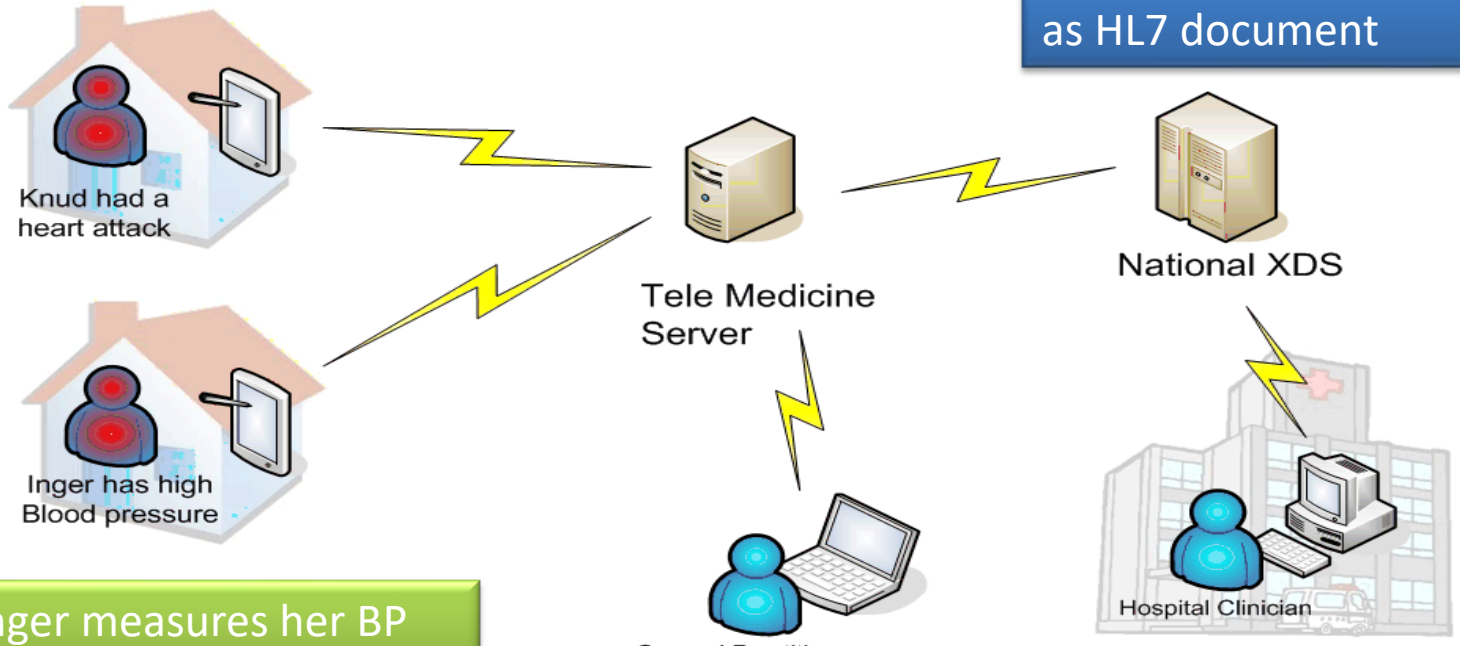
Vision

- Vision
  - Replace out-patient visits by measurements made by patients in their home
  - *Move data from home to regional/national storage so all health care personal can view them...*
- Motivation
  - Reduce out-patient visits
    - Better quality of life
    - Cost savings
    - Better traceability and visibility

AARHUS UNIVERSITET

Story 1

2) BP measurement stored as HL7 document

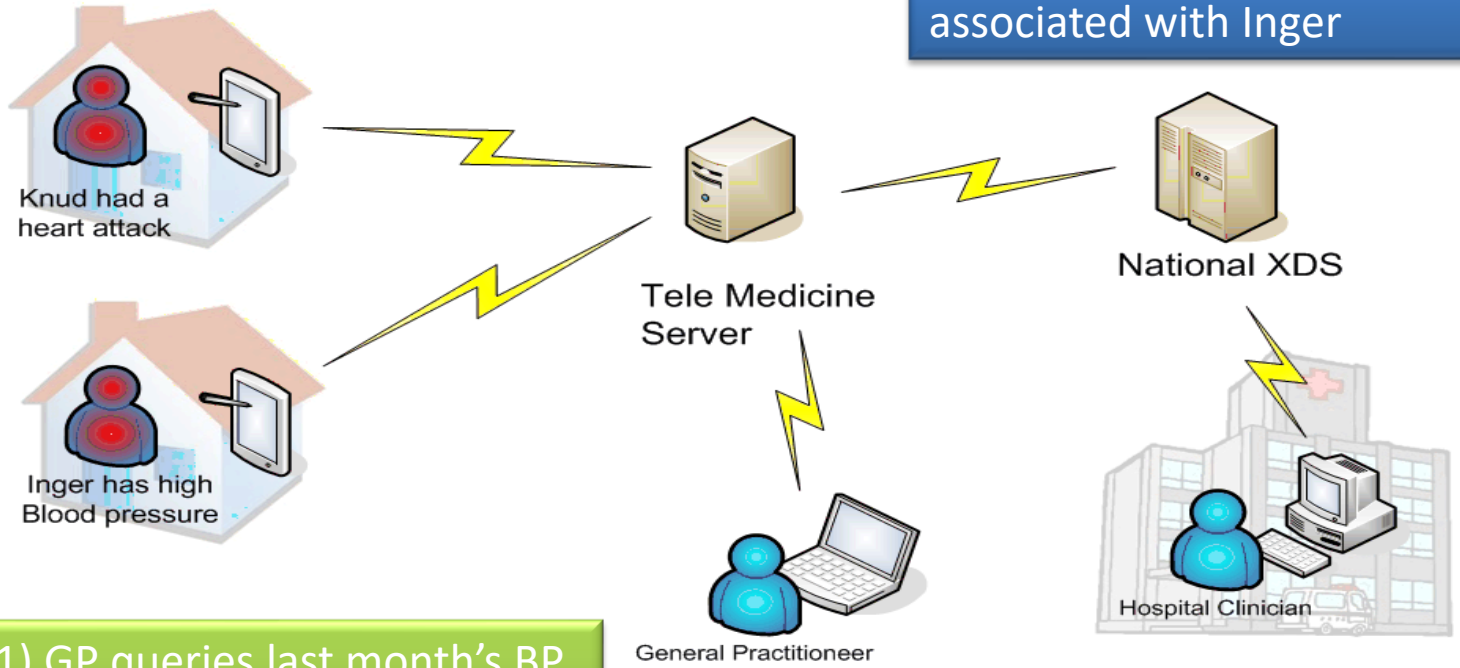


1) Inger measures her BP using her TeleMed terminal

AARHUS UNIVERSITET

Story 2

2) Query for all BP documents associated with Inger



1) GP queries last month's BP measurements for Inger using web browser

AARHUS UNIVERSITET

(What is XDS)

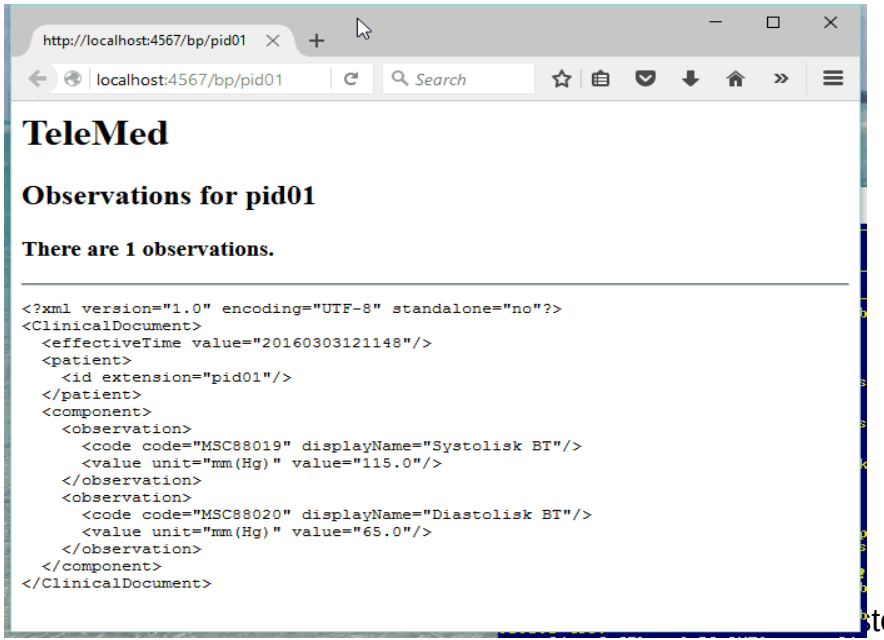
- Cross-Enterprise Document Sharing
  - One Registry + Multiple Repositories
  - Repository: Stores *clinical documents*
    - (id,document) pairs
  - Registry: Stores *metadata* with document *id*
    - Metadata (cpr, timeinterval, physician, measurement type,...)
    - Id of associate document and its repository

- Think
  - Registry = Google (index but no data)
  - Repository = Webserver (data but no index)

AARHUS UNIVERSITET

(What is HL7)

- HL7 is a standard (complex!) for clinical information storage and exchange.
  - Version 3 loves XML!
- Our version:

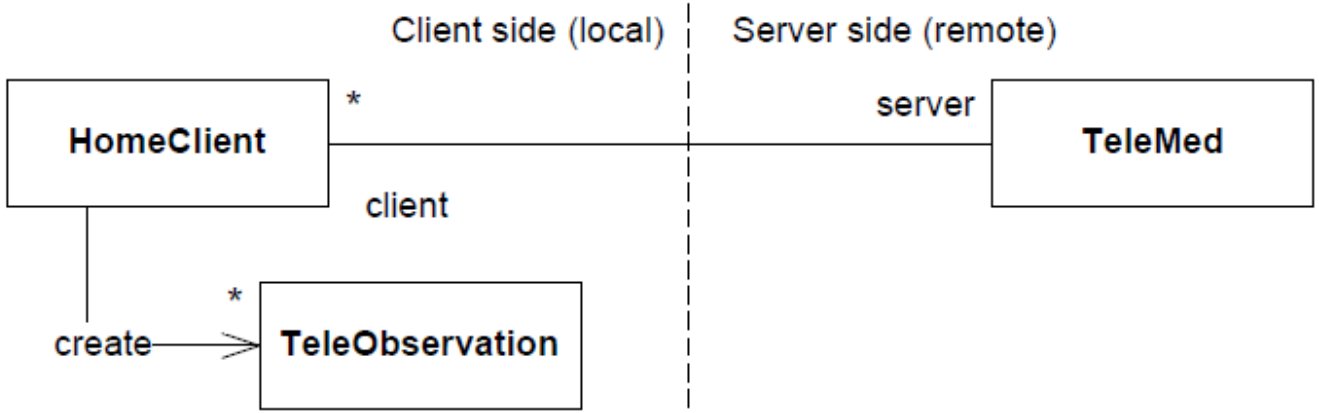


Real version:

tensen

TeleMed Design

- Roles involved



- TeleObservation: Represents a measurement
- HomeClient: Responsible for measuring + uploading
- TeleMed: Responsible for storage and queries

AARHUS UNIVERSITET

Demo

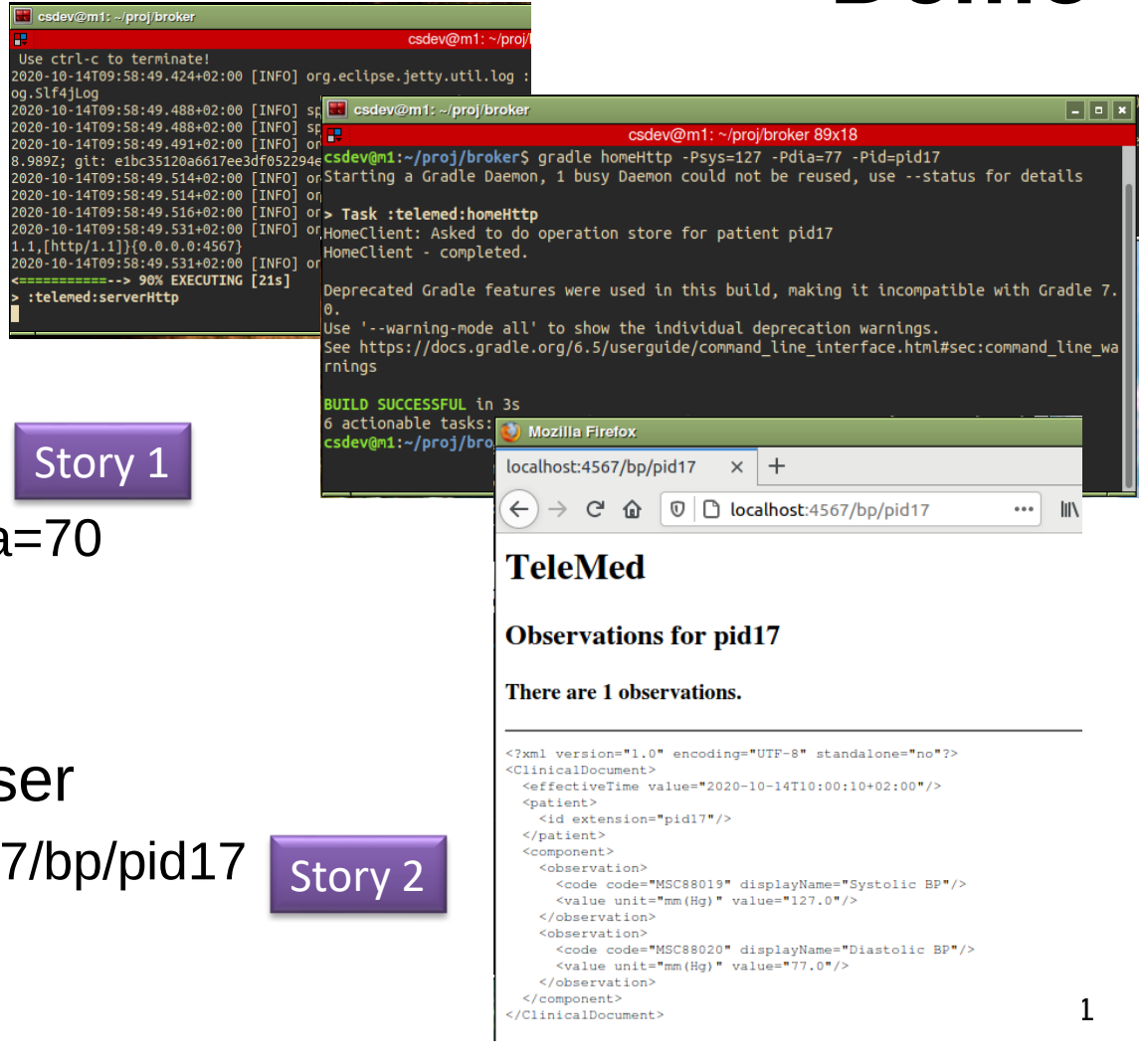
- Start a server
  - gradle serverHttp

- Send an obs.
  - gradle homeHttp
  - … -Psys=126 -Pdia=70 -Pid=pid17

Story 1

- GP review in browser
  - http://localhost:4567/bp/pid17

Story 2

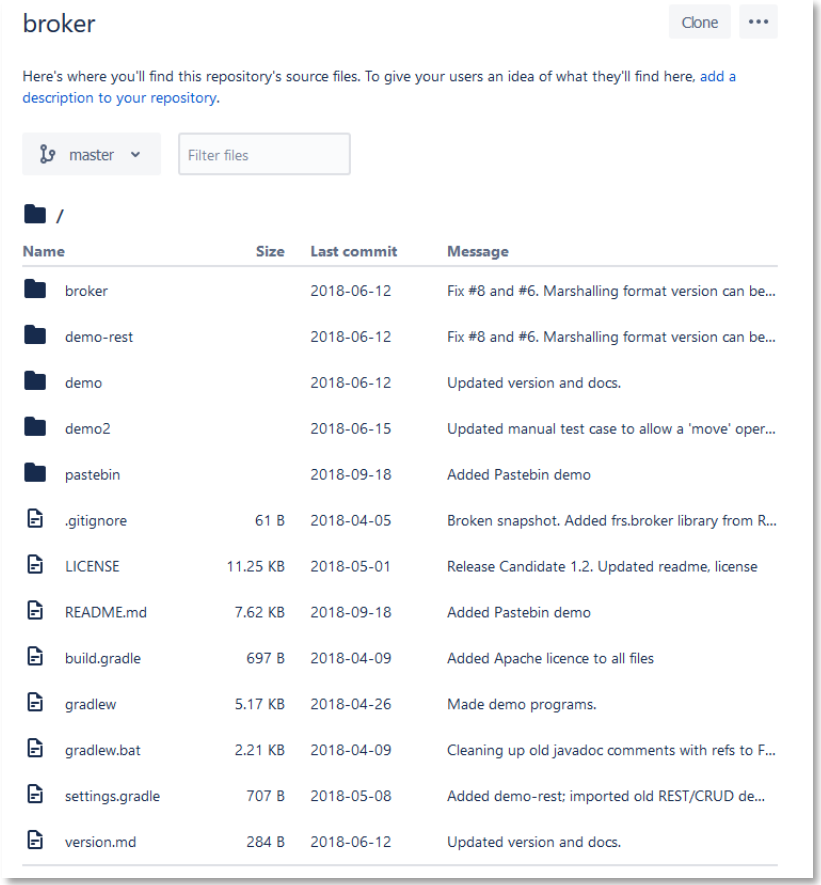


AARHUS UNIVERSITET

Source Code

- The source code is open source at
  - https://bitbucket.org/henrikbaerbak/broker/
  - Download or Fork

- ***You will want its code to learn the Broker pattern…***
  - But your HotStone mandatory only needs to fetch the Broker library using gradle…
    - As with MiniDraw library



broker

Clone

Here's where you'll find this repository's source files. To give your users an idea of what they'll find here, add a description to your repository.

master

Filter files

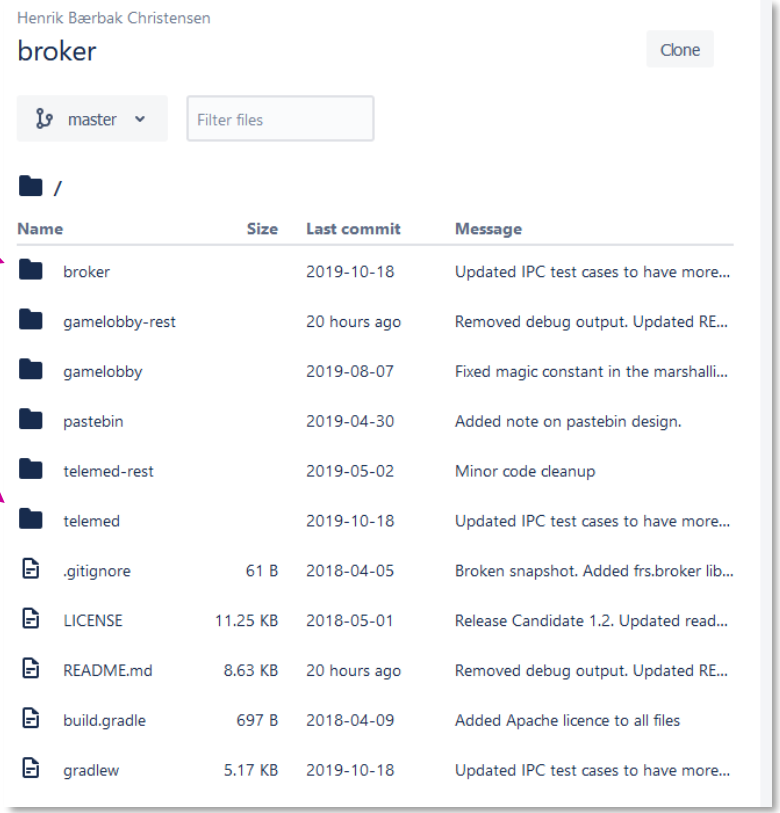
/

Name	Size	Last commit	Message
broker		2018-06-12	Fix #8 and #6. Marshalling format version can be...
demo-rest		2018-06-12	Fix #8 and #6. Marshalling format version can be...
demo		2018-06-12	Updated version and docs.
demo2		2018-06-15	Updated manual test case to allow a 'move' oper...
pastebin		2018-09-18	Added Pastebin demo
.gitignore	61 B	2018-04-05	Broken snapshot. Added frs.broker library from R...
LICENSE	11.25 KB	2018-05-01	Release Candidate 1.2. Updated readme, license
README.md	7.62 KB	2018-09-18	Added Pastebin demo
build.gradle	697 B	2018-04-09	Added Apache licence to all files
gradlew	5.17 KB	2018-04-26	Made demo programs.
gradlew.bat	2.21 KB	2018-04-09	Cleaning up old javadoc comments with refs to F...
settings.gradle	707 B	2018-05-08	Added demo-rest; imported old REST/CRUD de...
version.md	284 B	2018-06-12	Updated version and docs.

AARHUS UNIVERSITET

Source Code

- Subprojects
  - *Broker*: Core roles + default implementation of *some*
  - *TeleMed:* The TeleMed code including tests of *broker* code
  - *Others:* We will return to these next…



AARHUS UNIVERSITET

Issues in Distribution

Why is it hard?

Challenge

- How guys like me like to code:

Definition: Object-orientation (Responsibility)

An object-oriented program is structured as a community of interacting agents called objects. Each object has a role to play. Each object provides a service or performs an action that is used by other members of the community.

- Which is then something like this on the client:

```
public void makeMeasurement() {
  TeleObservation teleObs;
  teleObs = bloodPressureMeterHardware.measure();
  TeleMed server = new RemoteTeleMedOnServer(...);
  String teleObsId = server.processAndStore(teleObs);
}
```

Challenge

- However - networks *only support two asynch functions!*

```
void send(Object serverAddress, byte[] message);
byte[] receive();
```

- Which is *not* exactly the same as

```
public void makeMeasurement() {
  TeleObservation teleObs;
  teleObs = bloodPressureMeterHardware.measure();
  TeleMed server = new RemoteTeleMedOnServer(...);
  String teleObsId = server.processAndStore(teleObs);
}
```

Issues (at least!)

```
public void makeMeasurement() {
  TeleObservation teleObs;
  teleObs = bloodPressureMeterHardware.measure();
  TeleMed server = new RemoteTeleMedOnServer(...);
  String teleObsId = server.processAndStore(teleObs);
}
```

- Send/receive is a too low level programming model
- Send() does not wait for a reply from server (Asynch)
- Reference to object on *my* machine does not make sense on *remote* computer (memory address)
- Networks does not transfer objects, just bits
- Networks are slow
- Networks and Remote computers may fail
- Networks are insecure, others may pick up our data

Security QA

Availability QA

Performance QA

Architectural Issues: Not SWEA stuff. (Follow my EVU course once you are in a job ☺)

Performance

- Just how much slower is a network call compared to a local in-JVM memory call?

Configuration	Average time (ms)	Max time (ms)	Factor
Local call	1,796	3,366	1.0
Localhost	9,731	12,806	5.4
Docker	17,091	35,873	9.5
On switch	22,817	29,427	12.7
Frankurt	494,966	513,411	275.6

- Imagine that your next trip to the supermarket for a soda was 275 times slower???
  - 10 minutes walk versus 46,8 hours walking ☺

Elements of a Solution

- On the ‘happy path’, we need to
  - Make the HomeClient invoke a synchronous method call on a remote TeleMed object using only network send/receive
  - Keep our OO programming model: *telemed.processAndStore(to);*
    - *That is invoke specific method on remote object*
  - Convert TeleObservation object into bits to send it, and convert it back again
  - Locate the remote TeleMed object

Elements Overview

- Solutions are
  - Request/Reply Protocol
    - Simulate synchronous call (solves (partly) concurrency issue)
  - Marshalling
    - Packing objects into bits and back (solves data issue)
  - Proxy Pattern
    - Simulate method call on client (solves programming model issue)
  - Naming Systems
    - Use a registry/name service (solves remote location issue)
- Bundled together these constitute
  - The **Broker** pattern

AARHUS UNIVERSITET

Request/Reply

The Protocol

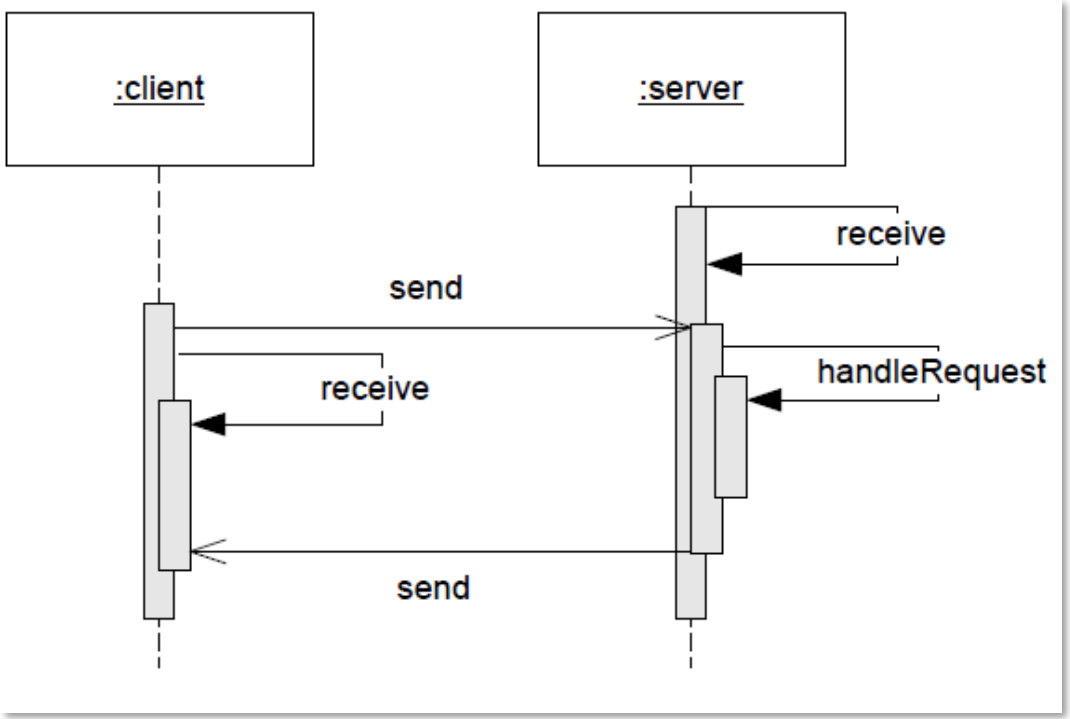
Definition: Request-Reply Protocol

The request-reply protocol simulate a synchroneous call between client and a server objects by a pairwise exchange of messages, one forming the request message from client to server, and the second forming the reply message from the server back to the client. The client sends the request message, and waits/blocks until the reply message has been received.

- Known from every WWW access you have ever made…
  - Firefox will *block* until a web page has been received

Pairing Send/Receives

- Client does
  - Send() and receive
- Server does
  - Receive() and send()
- Roles
  - Client is *active* – initiate action
  - Server is *reactive* – awaits actions and then reacts



Marshalling

Or Serialization

Definitions

Marshalling is the process of taking a collection of structured data items and assembling them into a byte array suitable for transmission in a network message.

Unmarshalling is the process of disassembling a byte array received in a network message to produce the equivalent collection of structured data items.

Two Basic Approaches

- There are two approaches
  - Binary formats
    - Google ProtoBuf, proprietary
  - Textual formats
    - XML, JSON, proprietary
- Exercise: Costs? Benefits?

JSON blood pressure

```
{
  patientId:   ''251248-1234'',
  systolic:    128.0,
  diastolic:   76.0
}
```

And we need more

- As we can send only bits, we also need to marshal information about the method and object id!

```
{
  methodName : "processAndStore_method",
  parameters : [
    {
      patientId:  ''251248-1234'',
      systolic:   128.0,
      diastolic:  76.0
    }
  ]
}
```

Note

- Marshalling is fine for atomic datatypes (int, double, char, array, …) but…

- What about *object references?*
  - *inventory.addCustomer(c)* *where c is Customer object?*

  - *Issue: the 'c' is an object reference but how to a use 'c' on the client if the object is located on the server?*

- Actually, it sort of depends on *parameter passing…*

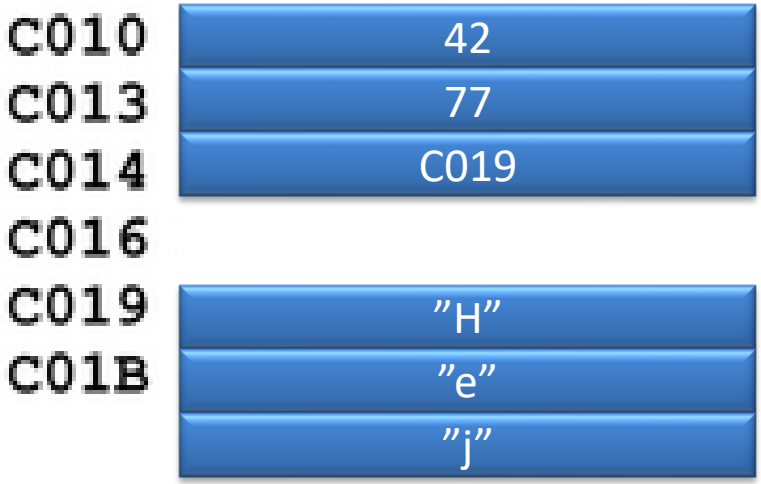
Parameter Passing

- **Pass by reference**
  - The Java style for all objects
  - *You do not get the Customer value, you get a **reference** to it!*
  - public void addCustomer(Customer c);

- **Pass by value**
  - Java does this for primitive types, like int and double
  - You do get the **value** itself
  - public void deposit(double amount);

At Machine Level

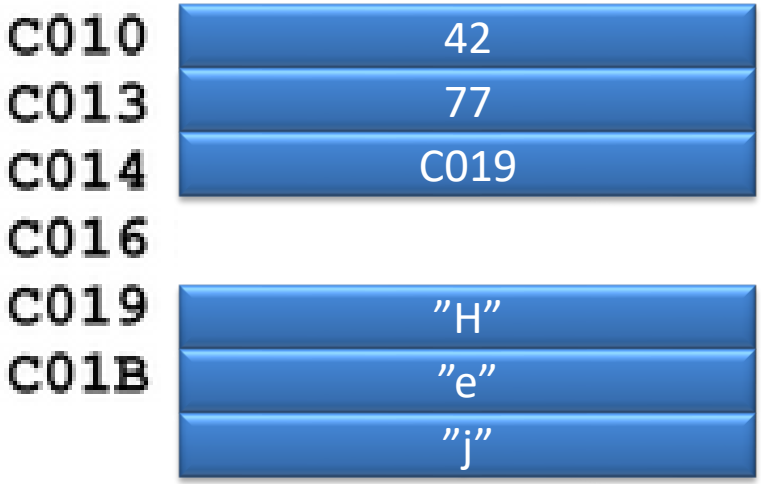
- All values are stored in memory on chunks of 64 bits
  - int value = 42;
  - int fish = 77;
  - String a = “Hej”;
- Internally, any variable/ object reference is a reference/pointer = *a memory address.*



- In C and C++ you can get that memory address, using &
  - value == 42 but &value = C010; and &fish = C013

At Machine Level

- All values are stored in memory on chunks of 64 bits
  - String a = “Hej”;
- An object reference in Java is not the “Hej” characters but the memory address of it!
  - a = C019 (reference to string)



- String a = null; *// C014 = 0L*
- a = new String(“Hej”); *// allocate space for string, put address of that space into C014*

… Which is why…

- You are taught to use .equals() instead of ==
  - If (x == 7) correct – value equality
  - If (s == “Hej”) incorrect – reference equality
  - If (s.equals(“Hej”) **correct – *values match***
- (Most of the time ‘s == “Hej”’ actually works in Java, because the compiler treats Strings in an intelligent way (which makes it confusing, sigh ☹))

… And Which is Why…

- You in HotStone should compare cards using *reference equality*
  - *If (card1 == card2)*
  - *Because if you do*
  - *If (card1.equals(card2))*
  - Then you get it wrong! Why?
    - Hint: You may have two Uno cards on the hand/field, right.
      - Are they the same card?

Java References

- If you create a class, and do *not* override ‘toString()’ you get a glimpse of that memory address
  - The JVM does some trickery so it is not a clean/real memory address, but anyway…

```
public class PrintAddr {
  public static void main(String[] args) {
    System.out.println("=== Java References ===");
    Point p = new Point();
    System.out.println("Value of p is: " + p );
  }
  private static class Point { public int x; public int y; }
}
```

```
csdev@m51:~/tmp$ java PrintAddr
=== Java References ===
Value of p is: PrintAddr$Point@372f7a8d
```

In C and C++ (and Go)

- In C and C++ you can actually choose…

value += 10;

```
void fooByValue(int value) { ... }
void fooByRef(int* value) { ... }
```

  – If the *first* call adds 10 to value, what happens to ‘value’ at the *call site?*
    - *int v = 7; fooByValue(v); print(v);*

  – If the *second* call adds 10 to value, what happens to value at the *call site?*
    - *int v = 7; fooByRef(&v); print(v);*

In Our Broker

- The semantics *change* in our Broker Pattern
  - localObject.say(”Hello”) localObject *pass by reference*
  - remoteObject.say(”Hello”) remoteObject *pass by value*
- Exercise: Why?

Our Broker *only* supports *pass by value!*

(next week we introduce a trick to *simulate* pass by ref)

Consequences

- If my *client* sends an object – pass-by-value
  - Person { String name; int age} with value { “Mikkel”, 29 }
- And the *server* receives this object and then change
  - Person { String name; int age} to value { “Magnus”, 26 }

- Then what happens in the *client’s* person object ???

AARHUS UNIVERSITET

JSON Libraries

Libraries

- Every distributed system in the world needs to marshall!
- Thus – lots of marshalling libraries around ☺
  - Do NOT code it yourself!!! You *will* end reimplementing one!
    - ~~String json = ”{ name: ”+ object.name+ ”}…~~

- JSON I have used many libraries
  - Json-simple
  - Jackson JSON
  - Gson

Gson

- Gson is the most compact I have used
  - (But have had trouble with ‘date’ objects that marshall incorrectly!)
- It allows easy marshalling of **record types**
  - Also known as
    - PODO: Plain Old Data Objects,
    - DTO: Data Transfer Object
- Record type (Pascal) / ‘struct’ (C) / record (java 17+)
  - No complex methods, only set/get methods with no side effects
  - **Must** have a default constructor
- That is: A pure data object, just storing information
  - Akin a ‘resource’ in REST terminology, by the way

Example:

```java
@Test public void shouldMarshallTeleObservation() {
  // This is a learning test, showing Gson marshalling
  Gson gson = new Gson();
  String json = gson.toJson(to);

  assertThat(json, containsString( substring: "\"patientId\":\"251248-0000\"")); 

  TeleObservation copy = gson.fromJson(json, TeleObservation.class);
  assertThat( copy.getPatientId(), is(HelperMethods.NANCY_ID));
  assertThat( copy.getSystolic().getValue(), is( value: 120.0));
  assertThat( copy.getDiastolic().getValue(), is( value: 70.0));
  assertThat( copy.getSystolic().getUnit(), is( value: "mm(Hg)") );
}
```

```json
{
  "patientId": "251248-0000",
  "systolic": {
    "value": 120,
    "unit": "mm(Hg)",
    "code": "MSC88019",
    "displayName": "Systolic BP"
  },
  "diastolic": {
    "value": 70,
    "unit": "mm(Hg)",
    "code": "MSC88020",
    "displayName": "Diastolic BP"
  },
  "time": {
    "date": {
      "year": 2017,
      "month": 6,
      "day": 30
    },
    "time": {
      "hour": 11,
      "minute": 7,
      "second": 26,
      "nano": 0
    }
  }
}
```

- toJson(obj)
  - Marshall
- fromJson(str, type.class)
  - Demarshall, using given type

AARHUS UNIVERSITET

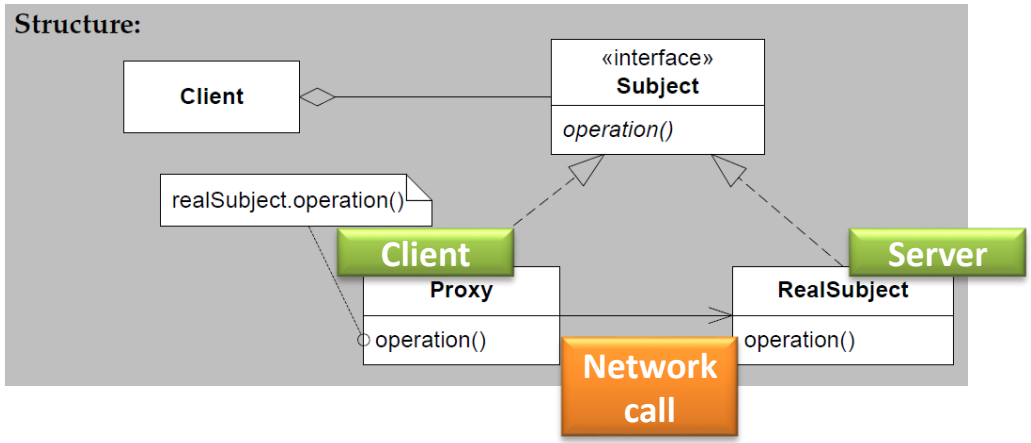
Proxy

You know that one…

Example

- TeleMed**Proxy**

```
public String processAndStore(TeleObservation teleObs) {
  byte[] requestMessage = marshall(teleObs);
  send(server, requestMessage);
  byte[] replyMessage = receive();
  String id = demarshall(replyMessage);
  return id;
}
```



Note

- The algorithm of *all methods in the proxy* will be the same
  - Marshall parameters, send, await reply, demarshall, return

- Can be auto generated – this is what RMI does

- We will *hand-code* it, because
  - … it is the learning goal of this course ☺
  - And it actually makes sense if you want very strict control of architectural attributes like performance and availability
    - *And, if you do not, you are in trouble ☺*
      - *Find more info in*

Teaching Distributed Programming – Revisiting the Broker Pattern

Author: Henrik Bærbak Christensen Authors Info & Claims

ECSEE '23: Proceedings of the 5th European Conference on Software Engineering Education • June 2023 • Pages 162–168 • https://doi.org/10.1145/3593663.3593674

Why QAs

- Why? *One Example:*
- You have a lot of accessor methods, and a single mutator
  - i.e. *state only changes when mutator is invoked!*
- RMI will autogenerate proxy (send/receive) for every method
- That is, every accessor method call will generate network traffic!
- **Performance Antipattern: Chatty interface**
- Pattern: Chunky interface
  - All accessors just return cached state in the proxy instance itself!

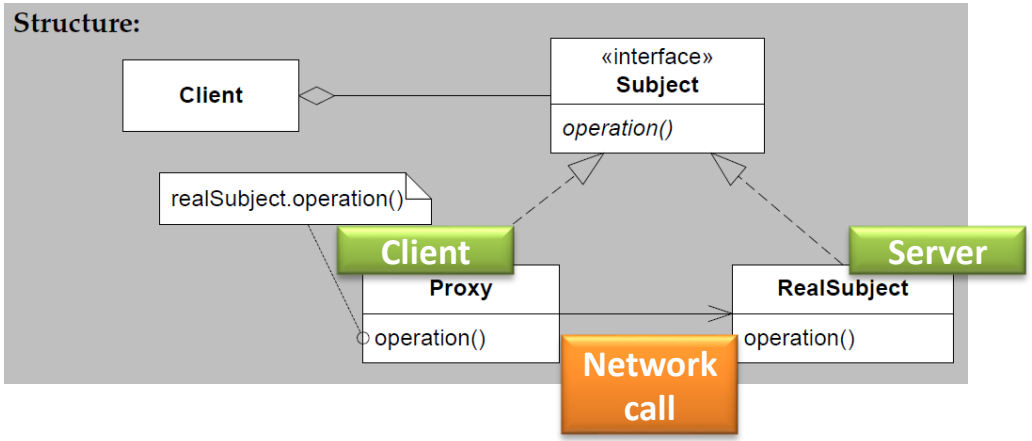
AARHUS UNIVERSITET

Name Services

Finding the Object to Talk to

Coupling Proxy and ‘server-side’

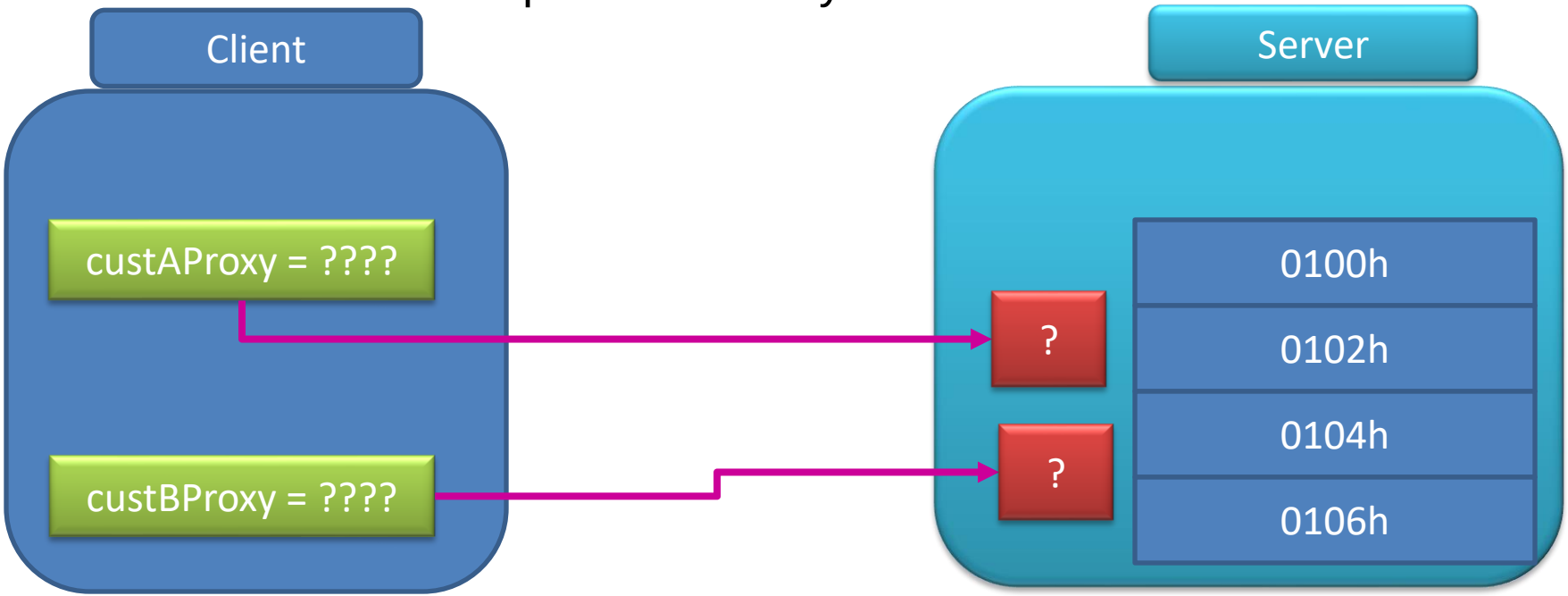
- Ok a Proxy “plays the client side role” of the real object on the server side…



- But what if there are many ‘RealSubjects’?
  - Like 10.000 instances of ‘Customer’ on the server?

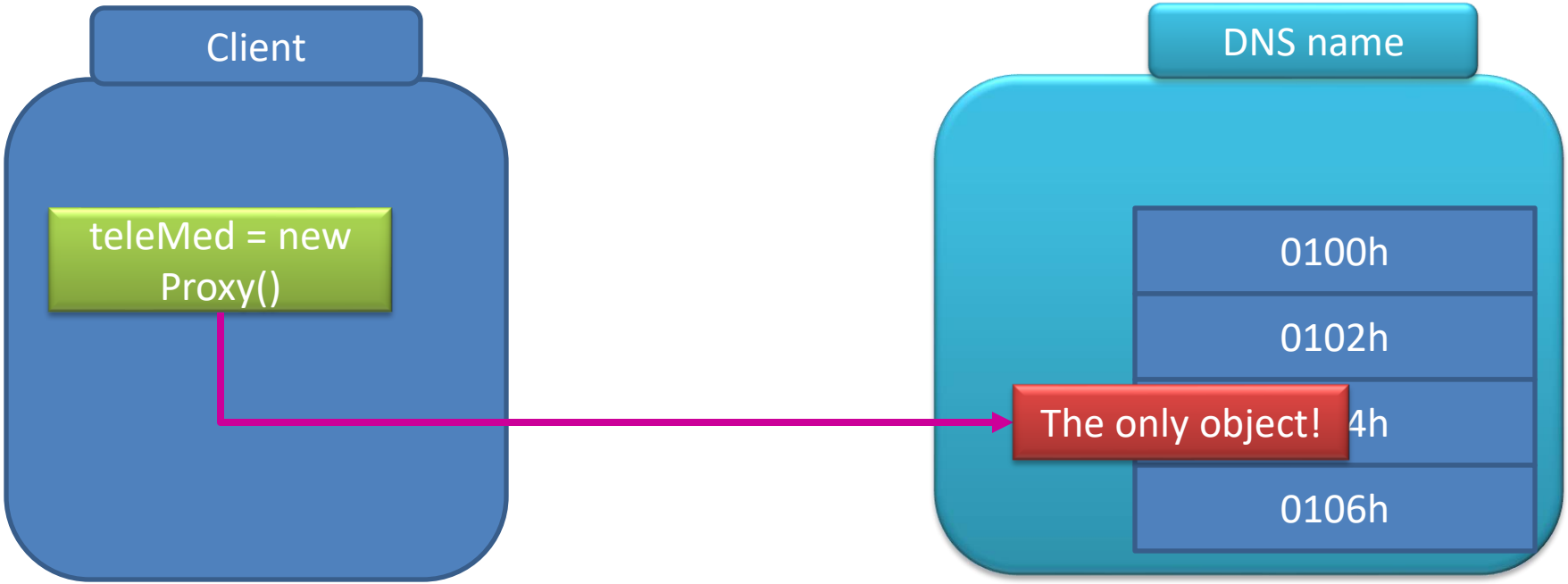
Now What?

- In distributed computing this is tricky
  - How to refer to a specific memory address in a remote server???



This Week (only!)

- We will solve this issue (partly) next week, but for now…
  - We just have *one object on the server* making it easier
    - If we know the IP of the machine – we can access that object ☺



Summary

- The Broker Pattern combines
  - Request/Reply protocol
  - Marshalling
  - Proxy pattern
  - Naming Systems (next week)
- … to produce something that (on happy days)

- Allows an Object Oriented Programming model to apply to distributed computing