**Assignment1 (Code aufs Cheat-Sheet)**

Exercise 1:

**Server**: socket, bind/listen, accept (socket, bind, receive, send, close), close;

**Client**: socket, bind/connect, send, receive, close;

//Server

class TimeServer {public void provideService() {int port = 8883;ServerSocket server = new ServerSocket(); server.listen(port); while(true){Socket client = server.accept(); HandleRequest hr = new HandleRequest(client); hr.start();}}}

class HandleRequest extends Thread {private Socket client; HandleRequest(Socket client) {this.client = client;} run() {String message = client.receive(); if(message == "getTime") {String result = getTime(); client.send(result); client.close();}}}

//Client

class Client {public void sendMessage() {String ipAddress='112.32.86.113'; int port = 8883; String message = "getTime"; Socket client = new Socket(); client.connect(ipAddress, port); client.send(message); String result = client.receive(); client.close(); print result;}}

Exercise 3: (wenn noch Platz)

Socket Implementations (synchrony):

(z.B. TimeServer, HandleRequest, Client)

And asynchrony

Exercise 5:

Advantages/Disadvantages of asynchronous to synchronous socket handling

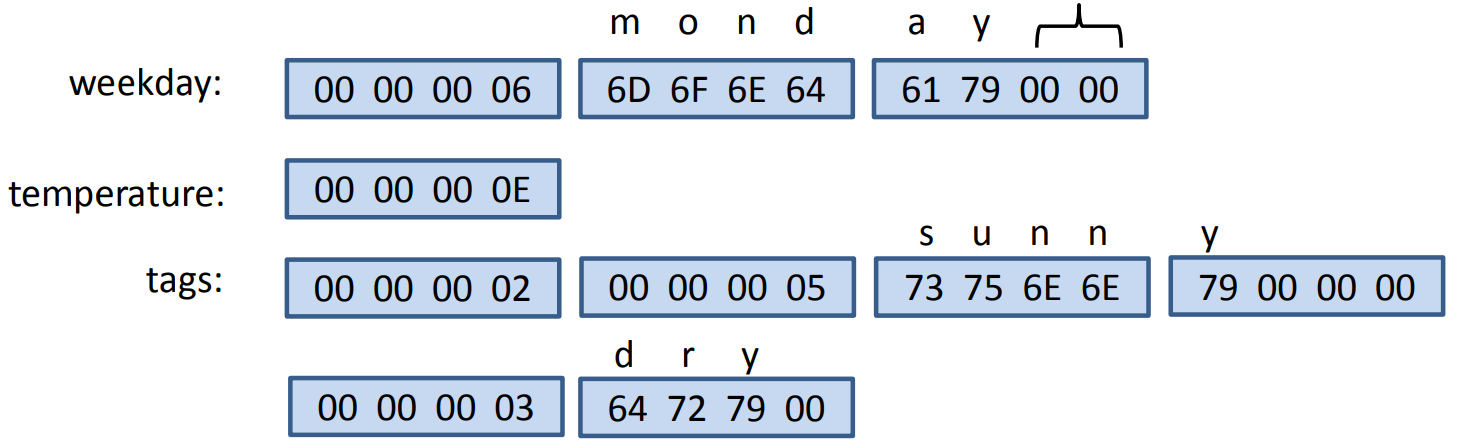
**Advantage**: Scalability, Slow consumers cannot block the server for a long time, one thread can handle multiple sockets, **Disadvantage**: Complex handling code, requires different kind of architecture, Eventloops

**Assignment2**

Exercise 1:

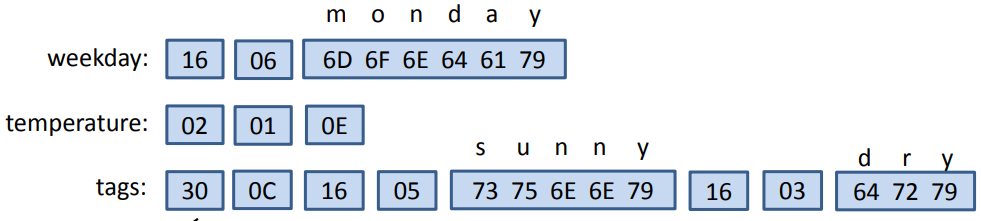
XDR:

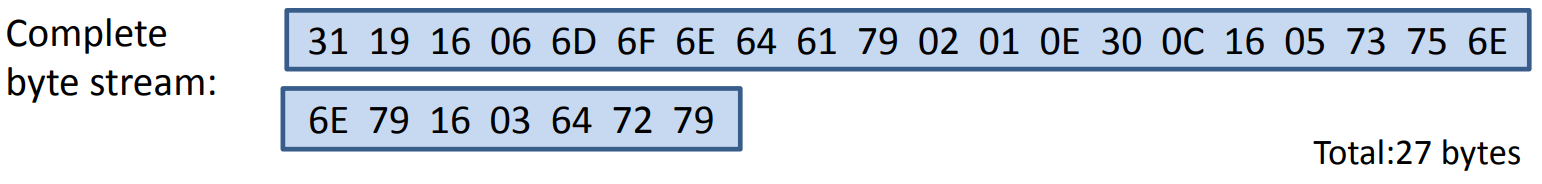
struct forecast{ string weekday; int temperature; string tags<>;}



ASN:

Forecast ::= SET{ weekday IA5String temperature Integer tags SEQUENCE OF IA5String; }





**Types**: boolean 1, integer 2, bit string 3, octet string 4, sequence 10/16, set 11/17, IA5String 16/22

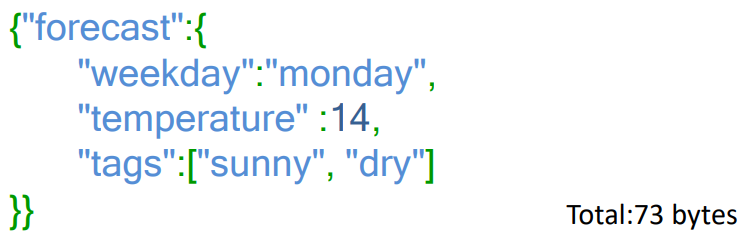
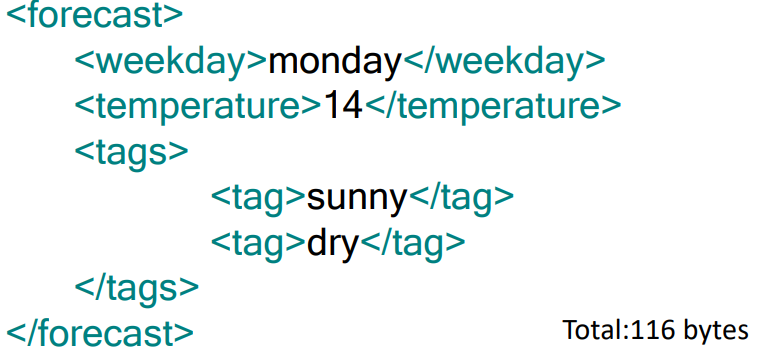
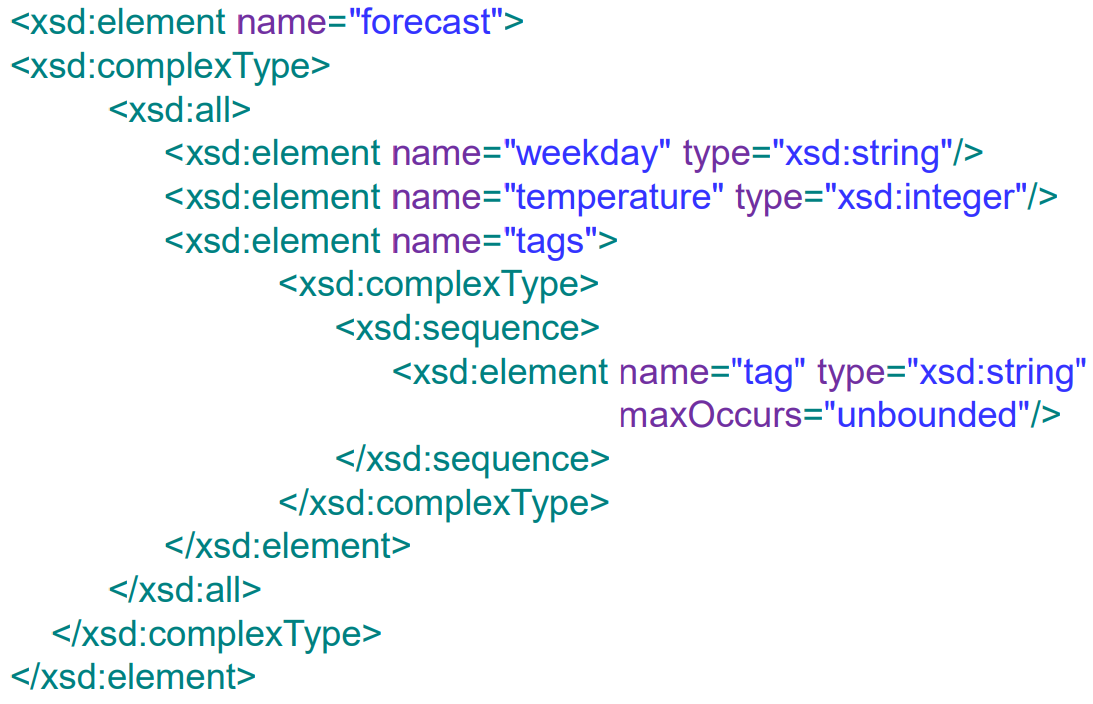
**Tag**: Class {universal 00, private 11}, f {primitive 0, constructed 1}, Number {von oben}

Disadvantages and Advantages

**ASN.1** encodes type information into the message. i. The receiver doesn’t need to know the data description. ii. The messages contain additional overhead. **XDR** uses blocks with multiples of 4 bytes. i. A lot of space/bandwidth is wasted with zero bits. ii. Fixed length of messages reduces the computational load. **XDR** doesn’t encode type information into messages i. Receiver needs to know the data description. ii. The message only contains data(no meta information).

Exercise 2:

XML and JSON examples



Disadvantages and Advantages

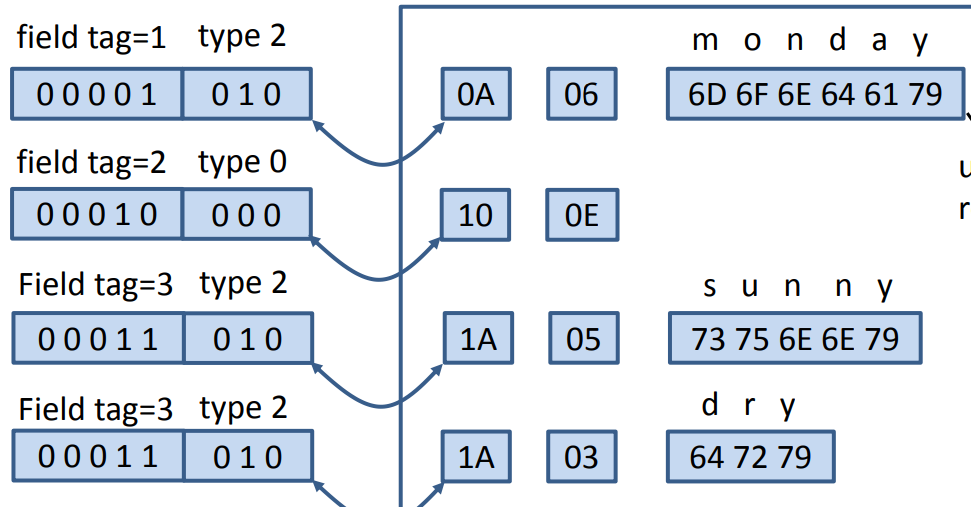
**XML** i. XML has large overhead. It is slow to write and parse. ii. Format is human-readable. Intuitive hierarchical structure, easy to read, also for non-programmers. iii. XML Builders/Parsers exist in every programming language. iv. XML is defined as standard (as well as JSON). **JSON** i. JSON has less overhead than XML, but still more than the binary data representations. ii. Still a human-readable format, not as structured as XML. iii. JavaScript support. JSON can be directly loaded into the Browser and deserialized into objects.

Exercise 3:

Protocol Buffers

message forecast{ required string weekday =1 required int32 temperature =2 repeated string tags =3}

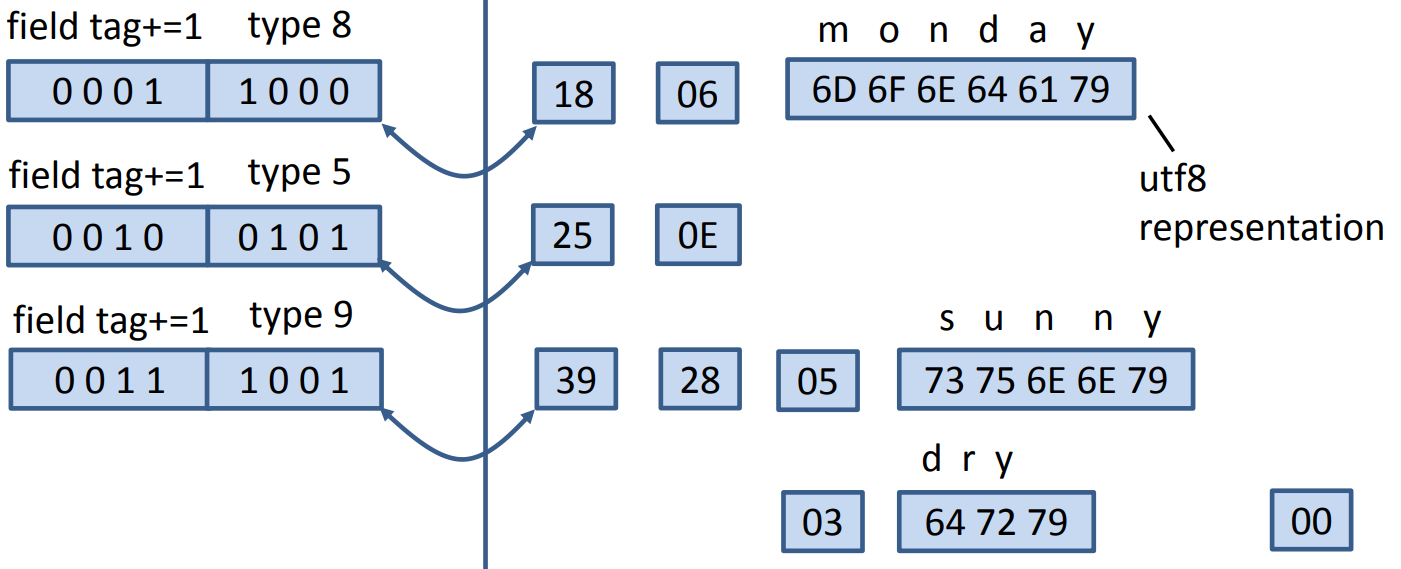
**Types**: Varint 0, 64-bit, 1, string/byte 2, start group 3 end group 4, 32-bit 5



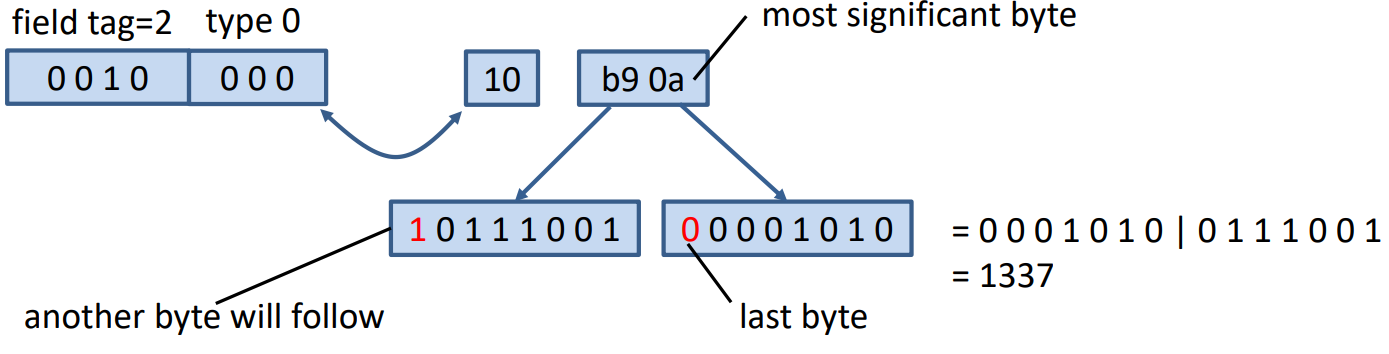
Apache Thrift

struct Forecast{ 1: string weekday 2: i32 temperature 3: list tags }

**Type**: Boolean 1, binary string 2, byte 3, i16 4, i32 5, i64 6, double 7, string 8, list 9



**int32, int64:**



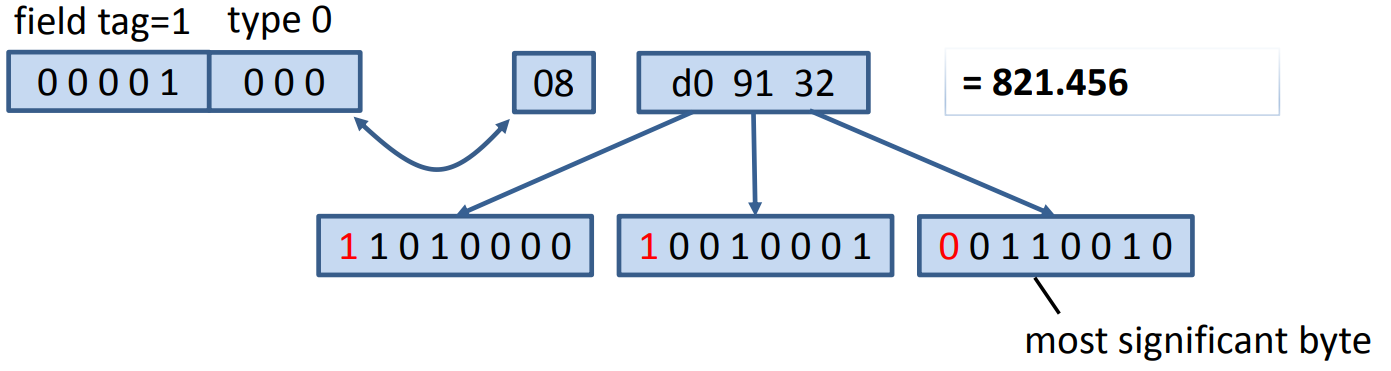
28 🡪 2 list items type 8 (string) 🡪 0010 1000

Disadvantages and Advantages

**Protocol Buffers** i. Highly efficient in writing/parsing data. ii. Well documented project. iii. Versioning of data structures possible. iv. No RPC implementation. **Apache Thrift** i. Multiple protocols to serve different purposes(binary, JSON). ii. Extensions for many programming languages. iii. RPC libraries included. iv. Versioning of data structures possible. v. Open source project, widely used.

Exercise 4:

Variable number encoding – protobuf protocol



(signedint) bei negativen: even 🡪 821.456\*2-1, odd 🡪 821.456\*2

**Assignment3**

Exercise 1:

RPC (Remote Procedure Calling)

1. c → s(k1,5), s Ꝺ (k1,5), s → c(k1,5)

2. c → s(k1,7), s Ꝺ (k1,12), s → c(k1,12)

Exercise 2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Maybe** | **at least once** | **at most once** | **exactly once** |
| **c → s** | no reactions | c repeats msg | c repeats msg | c repeats msg |
| **c → s(k3,8),**  **c → s(k3,8)** |  | s add key once,  c send msg again | s add key once | s add key once |
| **s crash after receiving msg** | s inform c about crash | s inform c about crash, c send msg again, s adds same key again | s inform c about crash, c sends msg again, s notice the duplicate and does not add key | s inform c about crash, rollback and answer c |

Exercise 3:

**Service interface in a .proto-file:**

message SearchRequest {required String attribute =1; required String value =2;} message CustomerList{repeated Customer customers =1;} message CategoryList{repeated Category categories =1;}

service AdministrationService {rpc CreateCustomer (Customer) return (Status); rpc DeleteCustomer(Customer) return (Status); rpc SearchCustomer(SearchRequest) return (CustomerList); rpc CreateCategory(Category) return (Status); rpc DeleteCategory(Category) return (Status); rpc SearchCategory(SearchRequest) return (CategoryList);}

Exercise 4:

RPC Implementation (zu viel Code)

downside of using delimiters to build/parse object properties

The values of object properties should never include one of the delimiters, If a class is changed, parsing the old classes will probably fail, Object meta information can be stored into the message → XML, Object properties can be serialized in bytes, fixed length intervals can be used → Protocol Buffers

**Assignment4**

Exercise 1:

Web services with SOAP, wsdl (focuses on processes)

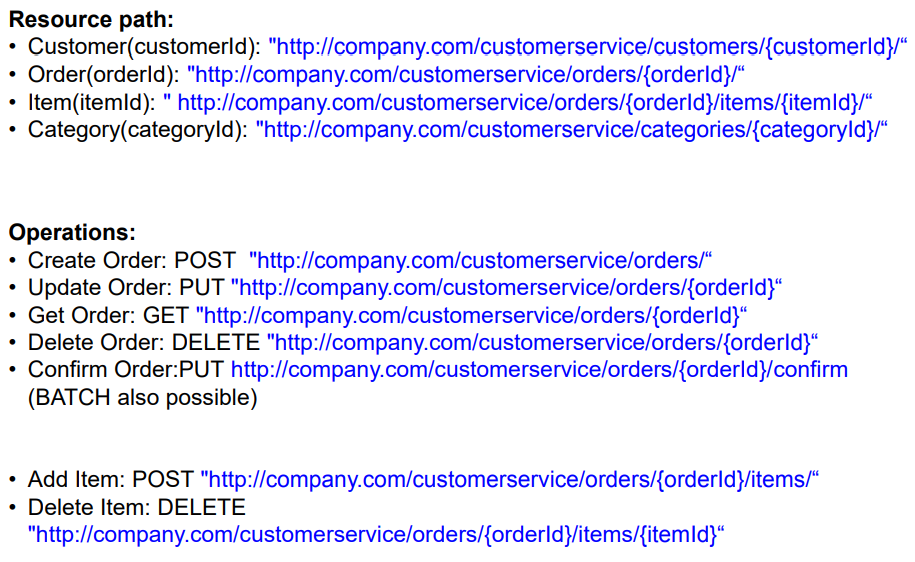


CRUD heavy Applications are a bad use case for RPC/SOAP: We have to define a lot of (CRUD) operations, Repeatedly, we have to define the same (CRUD) operations for different object types, We have to define a lot of messages, that contain no parameters or only one primitive, SOAP is depending on XML; large overhead for complex objects

Exercise 2:

Webservices with REST (REST focuses on resources)

* Rest verbs (GET, UPDATE, POST, DELETE)
* REST file: define paths and operations



REST is data centric, therefore a nice Use Case for CRUD heavy applications, Resources are identified by URL, unified schema, Access always by the same operations POST, GET, PUT, DELETE, Results in clear APIs; REST is independent of data representation

Exercise3:

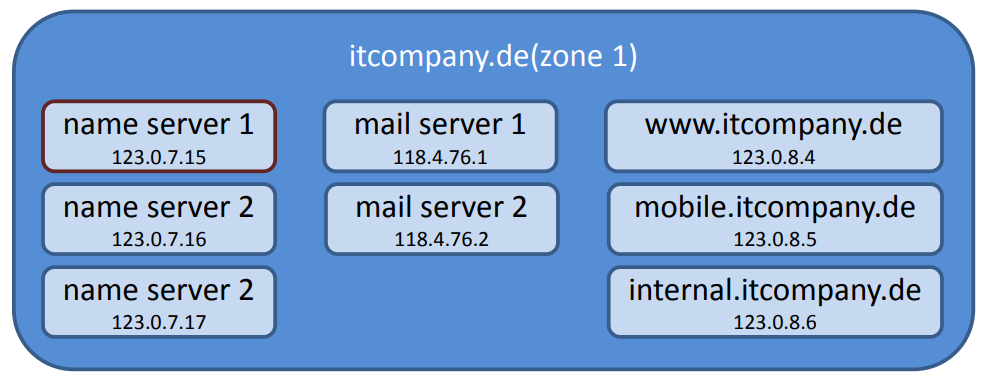
DNS resolution

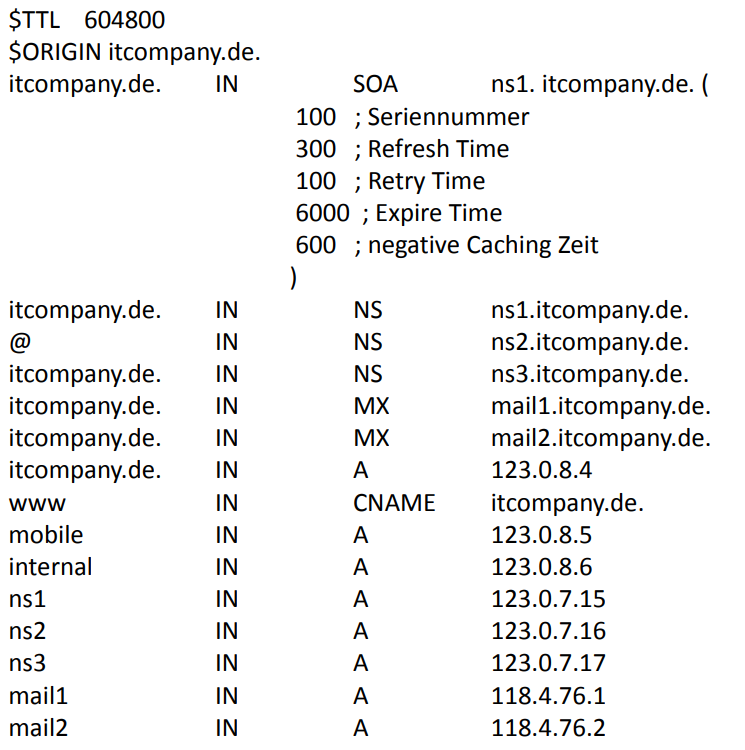
**DNS (iterative vs recursive)**

Bsp.: iterative: c1 → dns1(URLw1), dns1 → c1(IPr1)

recursive: dns2 → r1(URLw6), r1 → dns2(IPz2)

**DNS zone records:**





**Assignment5**

Exercise 1

**Message queuing**

enum Durability {Persistent, InMemory} class Producer {public void enqueue(byte[] message) {...}} class Consumer {public byte[] dequeue() {...}} class ChannelFactory { public void setHost(String broker) public void queueDeclare(String queuename, Durability durability) Producer newProducer(String queuename) Consumer newConsumer(String queuename)}

//Client

ChannelFactory cf = new ChannelFactory(); Producer producer; Consumer consumer; public void init(){cf.setHost("broker.tum.de"); cf.queueDeclare("request\_queue", Durability.Persistent); cf.queueDeclare("response\_queue", Durability.Persistent); producer = cf.newProducer("request\_queue"); consumer = cf.newConsumer("response\_queue");} public void send(byte[] msg){producer.enqueue(msg);} public byte[] receive(){byte[] response = consumer.dequeue(); System.out.println!("received message back" + response.toString()); return response;}

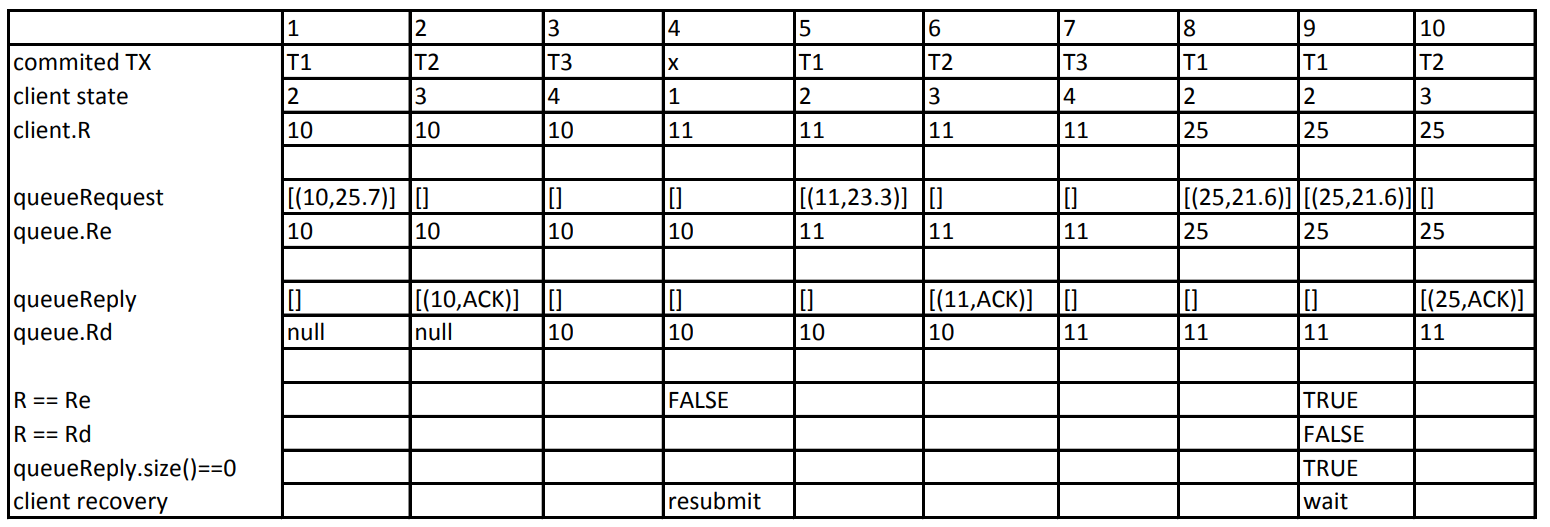
//Server

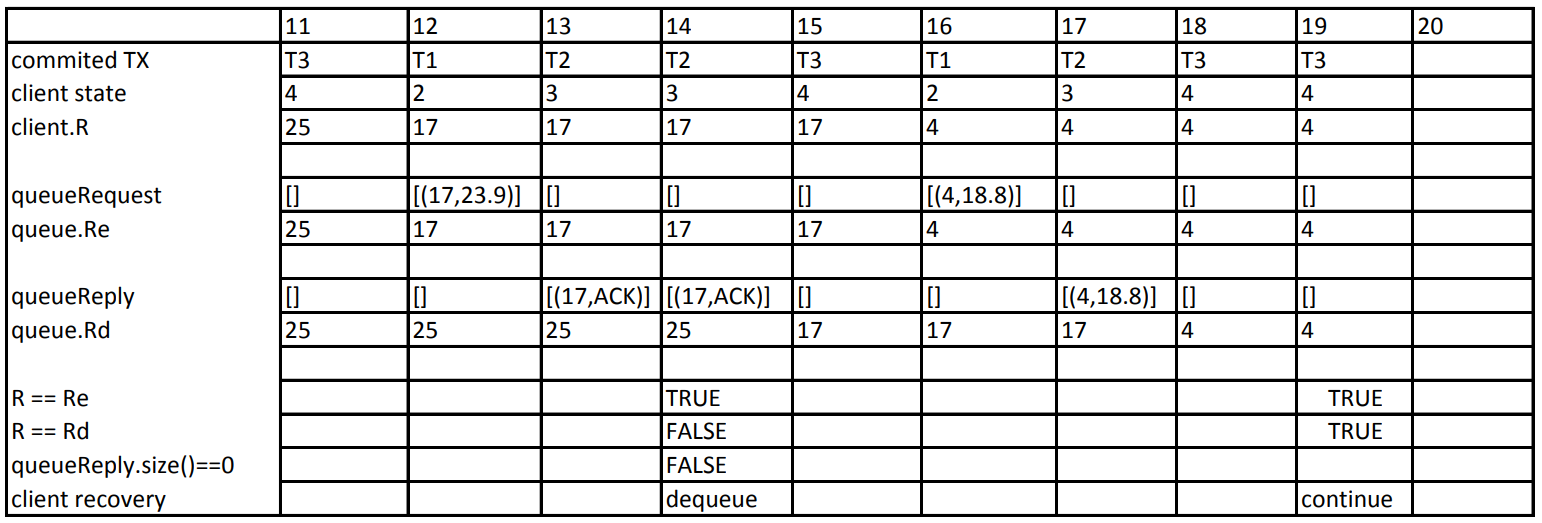
ChannelFactory cf = new ChannelFactory() Producer producer; Consumer consumer; public void init(){cf.setHost("broker.tum.de") cf.queueDeclare("request\_queue", Durability.Persistent); cf.queueDeclare("response\_queue", Durability.Persistent); producer = cf.newProducer("response\_queue"); consumer = cf.newConsumer("request\_queue");} public void run(){while(true) {byte[] request = consumer.dequeue(); println!("received msg" + request) try{Database.store(request); producer.enqueue("ACK".toBytes())}catch(Exception e){producer.enqueue("ERROR".toBytes())}}}

Exercise 4 b)

**Messaging – R/R with queued transactions**

1. c1(10,25.7) 2. c1(11,23.3) Client crashes during Transaction T X1 3. c1(25,21.6) Client crashes after Transaction T X1 4. c1(17,23.9) Client crashes after Transaction T X2 5. c1(4,18.8) Client crashes after Transaction T X3





**Assignment6**

Exercise1:

Two Phase Commit Protocol (2PC)

* 2PC execution (Vote Phase, Completion Phase)
* Coping with failures
  + **Timeouts** are used to cope with failures
  + Beispiele für Timeout-Aktionen, die keine Rücksprache mit anderen Parteien erfordern
    - Coordinator aborts TX when it timeouts in initial or wait state
    - Coordinator retransmits commit/abort message to participants when it timeouts in commit/abort state
    - Participant aborts TX when it timeouts in initial state
  + when a participant timeouts in the prepared state, consultation is required
  + The period from when a participant has voted to commit to the moment it knows the global decision, is called **uncertainty period**
  + Uncertain participant is blocked, until it becomes certain
  + **It cannot unilaterally abort because it cannot revoke its vote**
  + **It can also not unilaterally commit because the global decision may be to abort**
  + It can try to contact other participants to find one which is certain (that either voted abort or that already knows the global decision)
  + If it can only contact uncertain participants, it is blocked (reason may be communication failure or failure of all other sites)

Exercise2:

Queuing theory example

* Customers arrive at a theatre ticket counter in a Poisson fashion at the rate of 4 per hour. The time to serve a customer is distributed exponentially with mean 10 minutes. Calculate the following values:
  + **arrival rate:** λ = 4/h = 1/15min
  + **service rate:** µ = 1/10min
  + ρ = λ/µ = 10/15 = 2/3
  + N = λ/(µ - λ) = 2 (mean number of customers in the system)
  + Tsys = 1/(µ - λ) = 30min (mean time spent in the system, only if ρ < 1)
  + Tqueue = Tsys – x = 30min – 10 min = 20min
  + Pn = ρn(1 - ρ)
  + P0 = (1 - ρ) = 1/3 (probability that a customer arrives to find the ticket counter empty)
  + **Utilization**: λ \* µ

Exercise3/4:

Queuing theory - Parking space

* Consider the ticket counter again. A parking space should be build in front of the ticket counter such that the customers can park their cars while waiting for the tickets. The requirement is that 90% of the time, the customers should find parking.
* general formula for P(k ≤ n) 🡪 **P(k ≤ n) = 1−ρn+1**
* general formula for n 🡪 **n = ln(1−P(k≤n))/ln(ρ) − 1**

Queuing theory – Cost evaluation

* determine cost function
* bsp. : Cs = 10 µ/h, employee waiting in the system (queue and also getting served) is 50/h, λ = 8/h

**Assignment7**

Exercise1:

Queuing Theory - Traffic flows

* Arrival and departure rate
* Is a queue stable?
  + service time < arrival time, µ < λ
* **Achtung**: wenn Schleife zwischen Nodes dann Gleichungssystem

**Assignment8**

Exercise1/2:

Publish/subscribe

* Publish/subscribe – content-based routing
  + Bsp: {p2,b2,b1,{b4,s3}||{b3,s4}} → s3,s4
* Publish/subscribe - composite subscriptions (CS) 🡪 with AND and OR
  + Advantage with **AND** expression (…)

Exercise3:

topic based routing

* Examples:
  + p2 connects to b2 and advertises sports

[l1,=,sports][l2,=,∗][n,=,∗]

* + s3 connects to b3 and subscribes to football and tennis

[l1,=,sports][l2,=, football] AND [l1,=,sports][l2,=,tennis]

* + Publisher p3 publishes news n2 under politics

**Message**:[l1, politics][l2,**null**][n,n2]

**Path**:{p3,b4,b1,b0,s2} → s2

**Assignment9**

Exercise1:

State machine verification

* Draw message exchanges between two processes
* Draw the composition of the state-machine until the end or deadlock from a given combination

t5 = (s4,true,a2,s5), t6 = (s5,true, ?m5,s6), t7 = (s6,true,a3,s7), t’5 = (s’4 ,true,a2,s’5 ) t’6 = (s’5 ,true,a3,s 0 6 ) t’7 = (s’6 ,true,!m5,s’7 )

