FS19

RPC AND QUEUES

Consistency, Protocols/RPC, Message Queues

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Agenda

- **Distributed Systems in the News**
- **Consistency**
- **Message Queues / Apache Kafka**

Distributed Systems - In the *News*

- **27.04.2019: Verschlüsselung: Neue TLS-Empfehlungen vom BSI**
- **26.04.2019: Why We Can't Trust Network Time**
  - Not about consistency, but security
  - Validity period
    - Normal domain ownership change
    - Minimizing damages from hacker attacks
    - Technological upgrades → SHA1 to SHA256
  - IoT often offline, need to get time, NTP
  - NTP is often not secure → attack vector
- **26.4.2019: 6 Technical Challenges Developing a Distributed SQL Database**

1. Architecture: Amazon Aurora or Google Spanner?
   - Aurora: “Reads scale by using slave replicas which sacrifice consistency”
   - Spanner: “No foreign keys integrity”
2. SQL Protocol: PostgreSQL or MySQL?
   - PostgreSQL has a more permissive license
3. Distributed Transactions: Google Spanner or Percolator?
   - Perlocator: single timestamp oracle
   - Spanner: decentralized time tracking
4. Does Raft Work For Geo-Distributed Workloads?
   - Paxos:Spaner uses Paxos
   - Raft: easier to understand than Paxos, but latency high in certein use-cases
5. Can We Build a Software-Defined Atomic Clock?
   - Google: “The underlying source of time is a combination of GPS receivers and atomic clocks…”
   - Sync time with NTP with Lamport clocks to track causality
6. Rewrite or Reuse PostgreSQL Query Layer?
   - Reuse, after 5 month development

From the Exercises

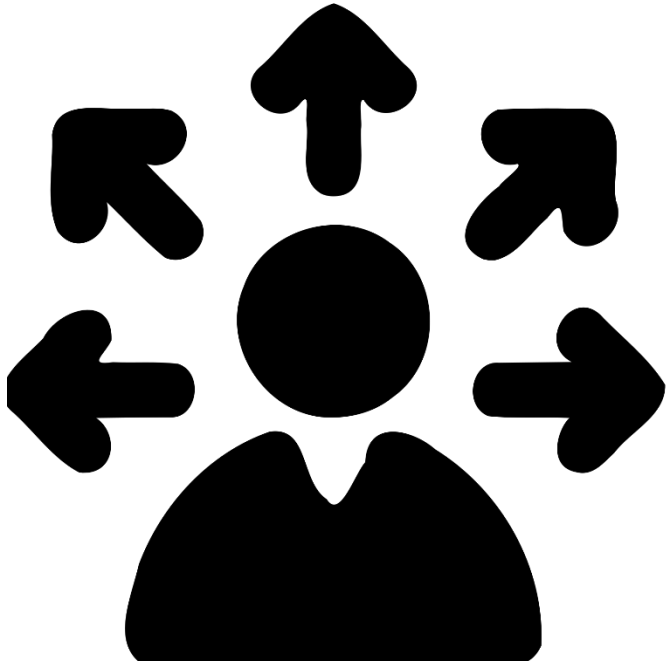
- **Attacking the DHT**
  - Create a key for a data item close to the target:
    Number160.createHash(data).xor(new Number160(0)) – distance 0, perfect match
    Number160.createHash(data).xor(new Number160(1)) – distance 1
    Number160.createHash(data).xor(new Number160(2)) – distance 2
  - Or create key of node close to the target
    new PeerBuilder(new Number160(RND)).ports(port).start(), where RND is
    Number160.createHash(data).xor(new Number160(0))
    Number160.createHash(data).xor(new Number160(1))
    …
  - Peer can then answer there is no data
  - For previously known values / peers (known public key)
    - Cannot change data, but make it disappear
- **Install vim on alpine:**
  - apk update
  - apk add vim

URL: b.socrative.com/student/

Room: HSR

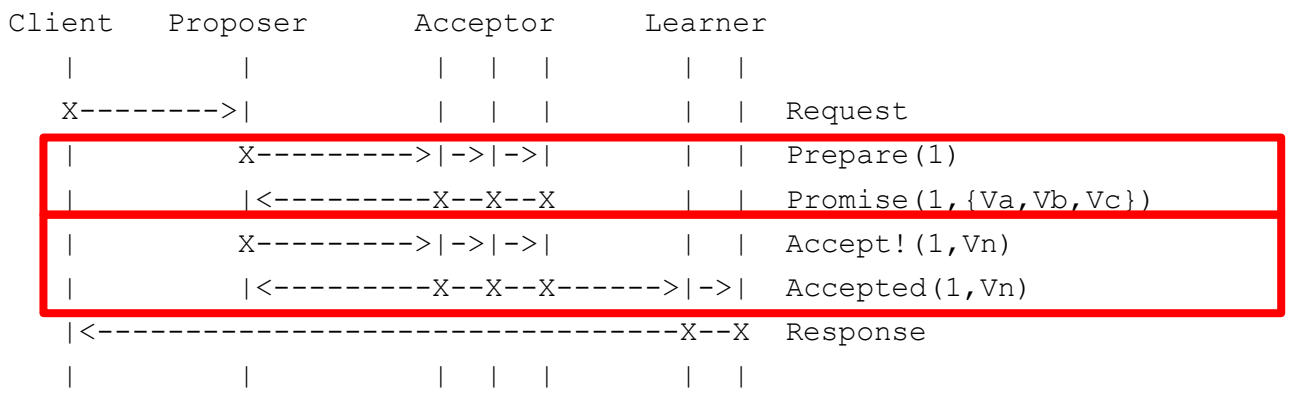
Consistency

- **DHTs have weak consistency**
  - Peer A put X.1, Peer B gets X.1 modifies it puts B.2
  - Same time: Peer C gets X.1 modifies it puts C.2
    - Which one is stored B.2 of B or C.2 of C?
- **Consistency generic issue in distributed systems**
  - No global time, coordinator required:
    - easy solution: centralized
    - hard solution: decentralized: nodes may fail
- **Coordinator needs to be defined**
  - Election, example Paxos



Consistency – Paxos – Leader Election

- **Paxos**
  - Protocol family for consensus (multi, cheap, fast, generalized, …)
  - Roles: Client/Proposer (requester), Acceptor (voter), Leader (coordinator), Learner (responder)
    - Client sends requests to a proposer
    - Proposer send proposal acceptor, send back promise
    - If majority promises, send value to acceptor, acceptor sent to learner
    - Learner sent result to client
- **2 Phases**
  - Phase 1: prepare / promise
  - Phase 2: accept / accepted



http://en.wikipedia.org/wiki/Paxos_%28computer_science%29

Consistency – Raft/Other – Leader Election

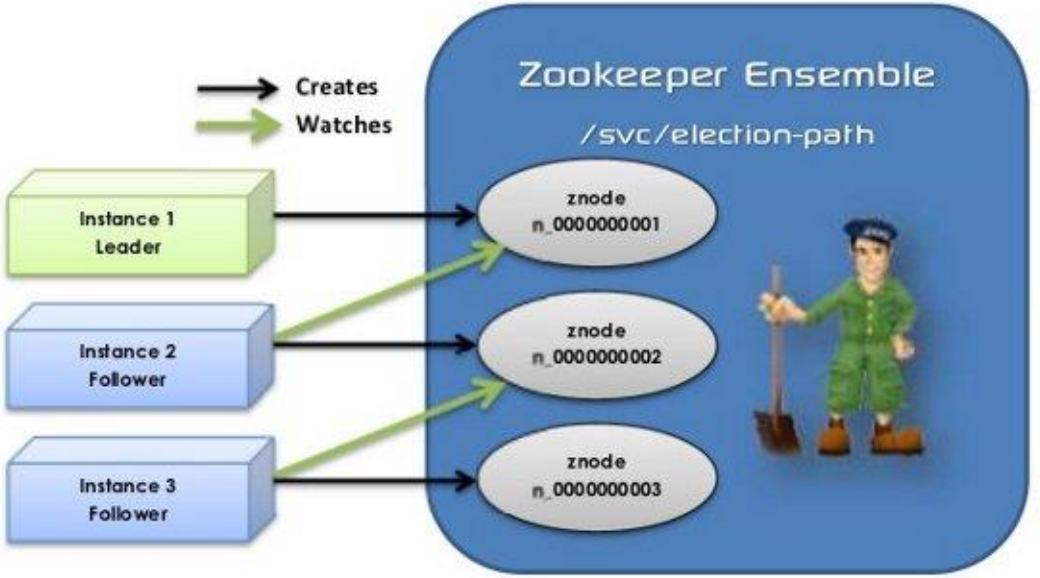
- **Raft – Alternative to Paxos (easier), three roles: leader, follower, candidate**
  - Paxos has a lot of roles: client, proposer, learner, acceptor, leader, follower. Implementation requires thorough understanding
  - In short: “Paxos has more complexity than it needs, and despite that, it needs to be tweaked to be really useful, and getting those tweaks right is hard.”
  - Raft applies randomized timers (between 150-300ms) to the heartbeat process to achieve leader election
    - If leader not alive, start leader election
    - Leader used to decide on consistency
- **etcd is built on top of Raft (using gRPC)**

		Usage Patterns								
Project	**Consensus System**	**SR**	**LR**	**SS**	**BO**	**SD**	**GM**	**LE**	**MM**	**Q**
GFS	Chubby			✓				✓	✓	
Borg	Chubby/Paxos	✓				✓		✓		
Kubernetes	etcd						✓		✓	
Megastore	Paxos		✓							
Spanner	Paxos	✓								
Bigtable	Chubby						✓	✓	✓	
Hadoop/HDFS	ZooKeeper	✓						✓		
HBase	ZooKeeper	✓		✓			✓		✓	
Hive	ZooKeeper			✓					✓	
Configerator	Zeus								✓	
Cassandra	ZooKeeper					✓		✓	✓	
Accumulo	ZooKeeper		✓	✓					✓	
BookKeeper	ZooKeeper						✓		✓	
Hedwig	ZooKeeper						✓		✓	
Kafka	ZooKeeper						✓	✓	✓	
Solr	ZooKeeper							✓	✓	✓
Giraph	ZooKeeper		✓		✓				✓	
Hama	ZooKeeper				✓					
Mesos	ZooKeeper							✓		
CoreOS	etcd					✓				
OpenStack	ZooKeeper					✓				
Neo4j	ZooKeeper			✓				✓		

http://container-solutions.com/raft-explained-part-1-the-consenus-problem/

Consistency – ZooKeeper – Leader Election

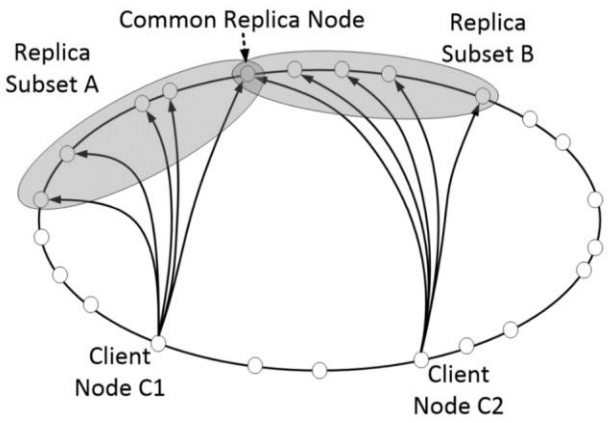
- **ZooKeeper - Leader election**
  - All nodes create a sequential node with path: /app/leader_election/guid_
  - Append the 10-digit sequence number to the path: /app/leader_election/guid_0000000001, /app/leader_election/guid_0000000002
  - Smallest number becomes the leader
  - Each follower node watches the next smallest number. E.g., the node /app/leader_election/guid_0000000008 will watch /app/leader_election/guid_0000000007
  - If leader goes down, corresponding /app/leader_election gets deleted
  - Next in line follower node will get the notification through watcher about the leader removal
  - Next in line follower node will check if there are any smaller number. If none, then it will assume the role of the leader.

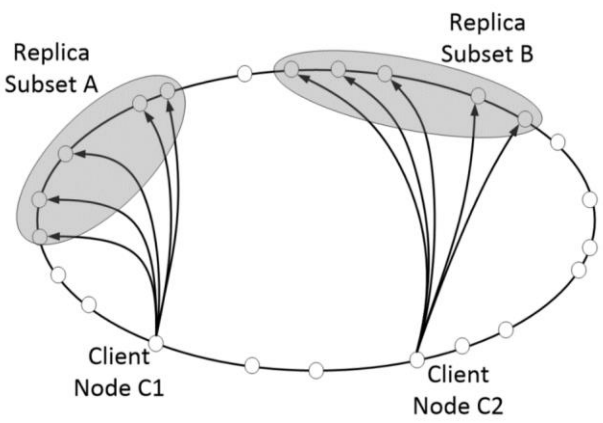


https://www.outbrain.com/techblog/2011/07/leader-election-with-zookeeper/

Consistency – DHT – Leader Election

- **Multi-Paxos, Raft, ZooKeeper → Leader Election**
- **Consistency in DHTs – vDHT**
  - Paxos and DHTs [1], [2]
  - vDHT: CoW, versions, 2PC, replication, software transactional memory (STM) → for consistent updates. Works for light churn
    - No locking, no timestamps (replication time may have an influence)
    - Every update – new version
      1. get latest version, check if all replica peers have latest version, if not wait and try again
      2. put prepared with data and short TTL, if status is OK on all replica peers, go ahead, otherwise, remove the data and go to step 1.
      3. put confirmed, don't send the data, just remove the prepared flag
  - **Leader** is the originator
- **In case of heavy churn, API user needs to resolv**





URL: b.socrative.com/student/

Room: HSR

Protocols

- **Protocols, lecture 7: layer 4**
  - TCP, UDP, (QUIC)
- **Protocols, in ex. 7: custom layer 7 protocol**
  - NAT-PMP
- **Writing custom protocols (e.g. TomP2P)**

  - Needs more time to develop / test

  + Can be more space efficient
- **Protocol generators (binary): Thrift / Avro / Protocol Buffers / (ASN1)**

  + Specification generates code

  + Standard

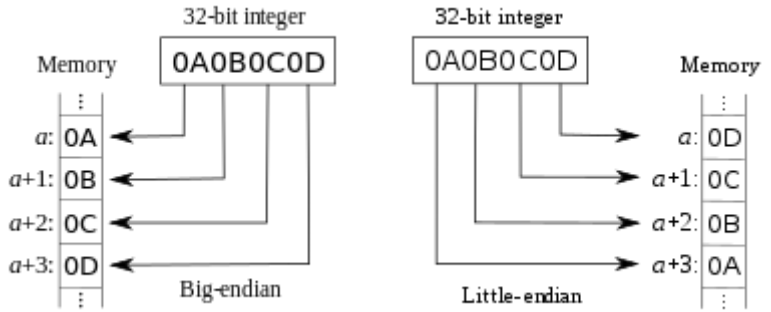
  - Has more overhead

```
//Avro IDL
@namespace("ch.hsr.dsl")

protocol MyProtocol{
  record AMessage {
     string request;
     int code;
  }
  record BMessage {
     string reply;
  }

  BMessage GetMessage(AMessage msg);
}
```

Protocols

- **Custom encoding/decoding**
  - You control every aspect
  - You send more time on it
- **Little-endian / Big-endian**
  - sequential order where bytes are converted into numbers
- **Networking, e.g. TCP headers: Big-endian**
- **Most CPUs e.g., x86: Little-endian**

```
115     public static boolean decodeHeader(final ByteBuf buffer, final InetSocketAddress recipientSocket,
116             final InetSocketAddress senderSocket, final Message message) {
117         LOG.debug("Decode message. Recipient: {}, Sender:{}.", recipientSocket, senderSocket);
118         final int versionAndType = buffer.readInt();
119         message.version(versionAndType >>> 4);
120         message.type(Type.values()[(versionAndType & Utils.MASK_0F)]);
121         message.protocolType(ProtocolType.values()[versionAndType >>> 30]);
122         message.messageId(buffer.readInt());
123         final int command = buffer.readUnsignedByte();
124         message.command((byte) command);
125         final Number160 recipientID = Number160.decode(buffer);
126
127         //we only get the id for the recipient, the rest we already know
128         final PeerAddress recipient = PeerAddress.builder().peerId(recipientID).build();
129         message.recipient(recipient);
130
131
132         final int contentTypes = buffer.readInt();
133         message.hasContent(contentTypes != 0);
134         message.contentTypes(decodeContentTypes(contentTypes, message));
```



Protocols Examples with Golang

- **UDP**
  - Why is it failing?

```
func main() {
   fmt.Println("connecting...")
   conn, _ := net.Dial("udp", "127.0.0.1:7000")
   defer conn.Close()
   buf := make([]byte, 4)
   binary.LittleEndian.PutUint32(buf, 77)
   _, _ = conn.Write(buf)
}
```

```
func main() {
   fmt.Println("listening...")
   inet := &net.UDPAddr{net.IPv4zero, 7000, ""}
   udpConn, _ := net.ListenUDP("udp", inet)
   b := make([]byte, 4)
   n, b2, _ := udpConn.ReadFromUDP(b);
   fmt.Printf("connecting... read: %d, addr: %v, data: %v," +
     " decoded: %v\n", n, b2, b[:n], binary.BigEndian.Uint32(b))
}
```

- **TCP**
  - Custom serialization 5,Anybody there?
    - 15 bytes

```
func main() {
   fmt.Println("connecting...")
   conn, _ := net.Dial(“tcp", "127.0.0.1:7000")
   defer conn.Close()
   buf := make([]byte, 15)
   buf[0]=5
   copy(buf[1:],[]byte("Anybody there?"))
   _, _ = conn.Write(buf)
}
```

```
func main() {
   fmt.Println("listening...")
   tcpConn, _ := net.Listen("tcp", ":7000")
   conn, _ := tcpConn.Accept() //do this in a go routine
   b := make([]byte, 15)
   n, _ := conn.Read(b)
   fmt.Printf("connecting... read: %d, addr: %v, data: [% x], decoded: %v\n",
      n, conn.RemoteAddr(), b[:n], string(b[1:]))}
```

Protocols Example ASN1

```
30 13 02 01 05 16 0e 41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f
```

- **ASN1**
  - Standard interface description language (IDL)
  - Define data structures - can be serialized and deserialized
  - Used e.g., in: X.509 (hsr.ch)
  - Generic binary protocol, Golang package
  - Example: 21 bytes, XML: 48 bytes

```
30 — type tag indicating SEQUENCE
13 — length in octets of value that follows
  02 — type tag indicating INTEGER
  01 — length in octets of value that follows
    05 — value (5)
  16 — type tag indicating IA5String
     (IA5 means the full 7-bit ISO 646 set, including
variants,
      but is generally US-ASCII)
  0e — length in octets of value that follows
    41 6e 79 62 6f 64 79 20 74 68 65 72 65 3f — value
("Anybody there?")
```

```
type TestASN struct {
            X *big.Int
            Y string
}
…
func main() {
…
   var t TestASN
   _, err = asn1.Unmarshal(b, &t)
}
```

```
<code>5</code>
<message>Anybody there?</message>
```

URL: b.socrative.com/student/

Room: HSR

Protocols Example Avro

- **Avro: data serialization system**
  - Remote procedure call and data serialization framework
  - Use: Hadoop (Big-data framework)
- **LinkedIn go-avro library**
  - Define a message in JSON (benchmarks) or IDL (slide 12) – no code generation

```
func main() {
  schema, _ := ioutil.ReadFile("schema.avsc")
  codec, _ := goavro.NewCodec(string(schema))
  map1 := map[string]interface{}{
    "request": "Anybody there?",
    "code":  5,
  }
  binary, _ := codec.BinaryFromNative(nil, map1)
  conn, _ := net.Dial("tcp", "127.0.0.1:7000")
  defer conn.Close()
  n, _ := conn.Write(binary)
  fmt.Printf("wrote %d bytes: [% x]\n", n, binary)
}
```

- **Server**
  - Example 16 bytes, assuming both have the same IDL

```
{"namespace": "ch.hsr.dsl",
  "type": "record",  "name": "AMessage",
  "fields": [
      {"name": "request", "type": "string"},
      {"name": "code", "type": "int"}
  ]
}
```

```
func main() {
  schema, _ := ioutil.ReadFile("schema.avsc")
  codec, _ := goavro.NewCodec(string(schema))
  tcpConn, _ := net.Listen("tcp", ":7000")
  conn, _ := tcpConn.Accept() //do this in a go routine
  binary := make([]byte, 100)
  n, _ := conn.Read(binary)
  native, _, _ := codec.NativeFromBinary(binary[:n])
  fmt.Printf("read: %v\n", native)
}
```

Protocol Buffers Example

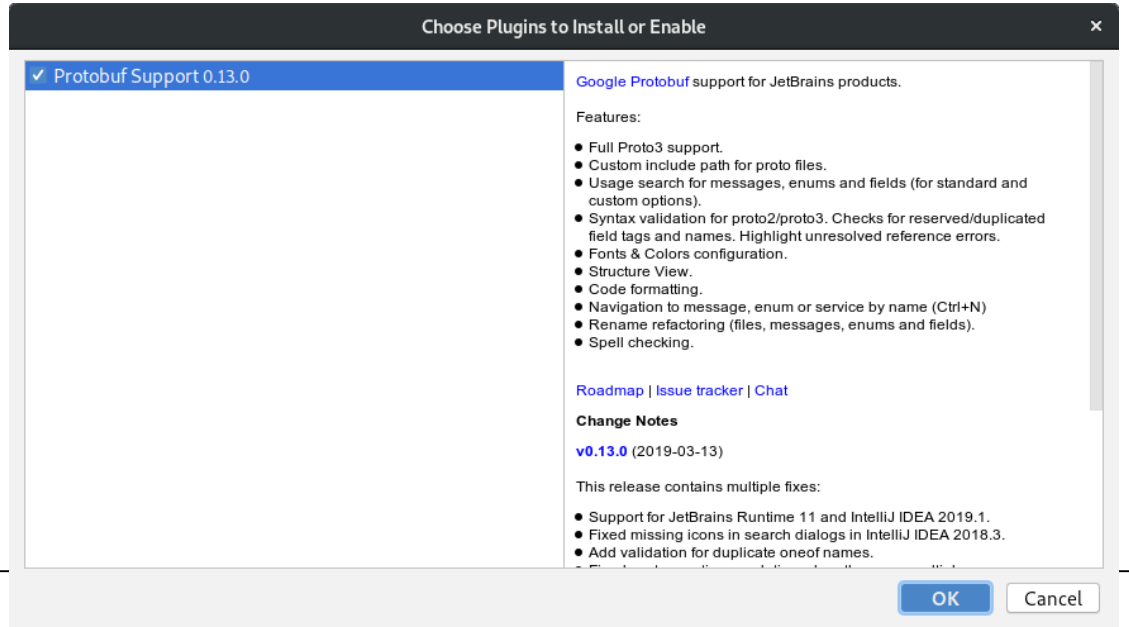
- **Protobuf: data serialization system from Google**
  - Design goals: smaller and faster than XML
  - Use: nearly all inter-machine communication at Google
- **Generate 1 go file**
  - protoc schema.proto --go_out go-gen2
- **18 bytes**

```
m := &pb.AMessage{Id: 5, Message: "Anybody there?"}
out, err := proto.Marshal(m)
```

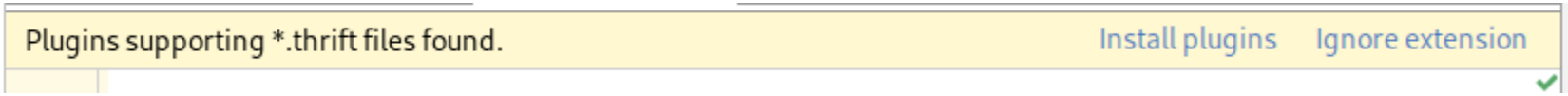
```
m := &pb.AMessage{}
if err := proto.Unmarshal(binary[:n], m); err != nil {
  panic(err)
}
```

- **Not self-describing, but has gzipped description**

```
syntax = "proto3";
message AMessage {
  int32 id = 1;
  string message = 2;
}
```



RPC Example Thrift



- **RPC Framework from Facebook**
  - IDL and binary protocol
  - For building cross-platform application in ActionScript, C, C++, C#, Cappuccino, Cocoa, Delphi, Erlang, Go, Haskell, Java, Node.js, Objective-C, OCaml, Perl, PHP, Python, Ruby, Rust, Smalltalk, and Swift
- **Installation**
  - sudo apt install thrift-compiler
  - Did not work (0.9), needed to compile manually
  - thrift -r --gen go schema.thrift
  - Creates go files (client)

```
service TestService {
    void AMessage(1:i32 int, 2:string message)
}
```

- **Example generic server**
  - go run simple-gen-srv.go
  - go run simple-thrift.go -p 7000
  - go run simple-thrift.go -p 7000 AMessage 5 'Anybody there?'
- **49 bytes transferred**
- **Thrift also encodes which function to call, larger size**

RPC Example gRPC

- **gRPC**
  - Uses HTTP/2 for transport
  - Uses Protocol Buffers as IDL
  - Features: authentication, bidirectional streaming and flow control, blocking or nonblocking bindings, and cancellation and timeouts
  - Installation
    - go get -u google.golang.org/grpc
    - go get -u github.com/golang/protobuf/protoc-gen-go
    - protoc schema-grpc.proto --go_out=plugins=grpc:go-gen3
  - 198 / 135 (wireshark)

- **Define services and message**
  - Generate 1 file (larger)

```
syntax = "proto3";

service MessageService {
  rpc SendMessage (AMessage) returns(Empty);
}

message AMessage {
  int32 code = 1;
  string message = 2;
}

message Empty {}
```

```
grpcServer := grpc.NewServer()
s := Server{}
schemagrpc.RegisterMessageServiceServer(grpcServer, &s)
grpcServer.Serve(tcpConn)
```

```
var conn *grpc.ClientConn
conn, _ := grpc.Dial(":7000", grpc.WithInsecure())
if err != nil {
  log.Fatalf("did not connect: %s", err)
}
defer conn.Close()
c := schema_grpc.NewMessageServiceClient(conn)
response, _ := c.SendMessage(context.Background(),
     &schema_grpc.AMessage{Code:5,Message:"Anybody there?"})
```

JSON example

- **JSON + REST**
  - Human-readable text to transmit data
  - Often used for web apps
- **196 bytes**

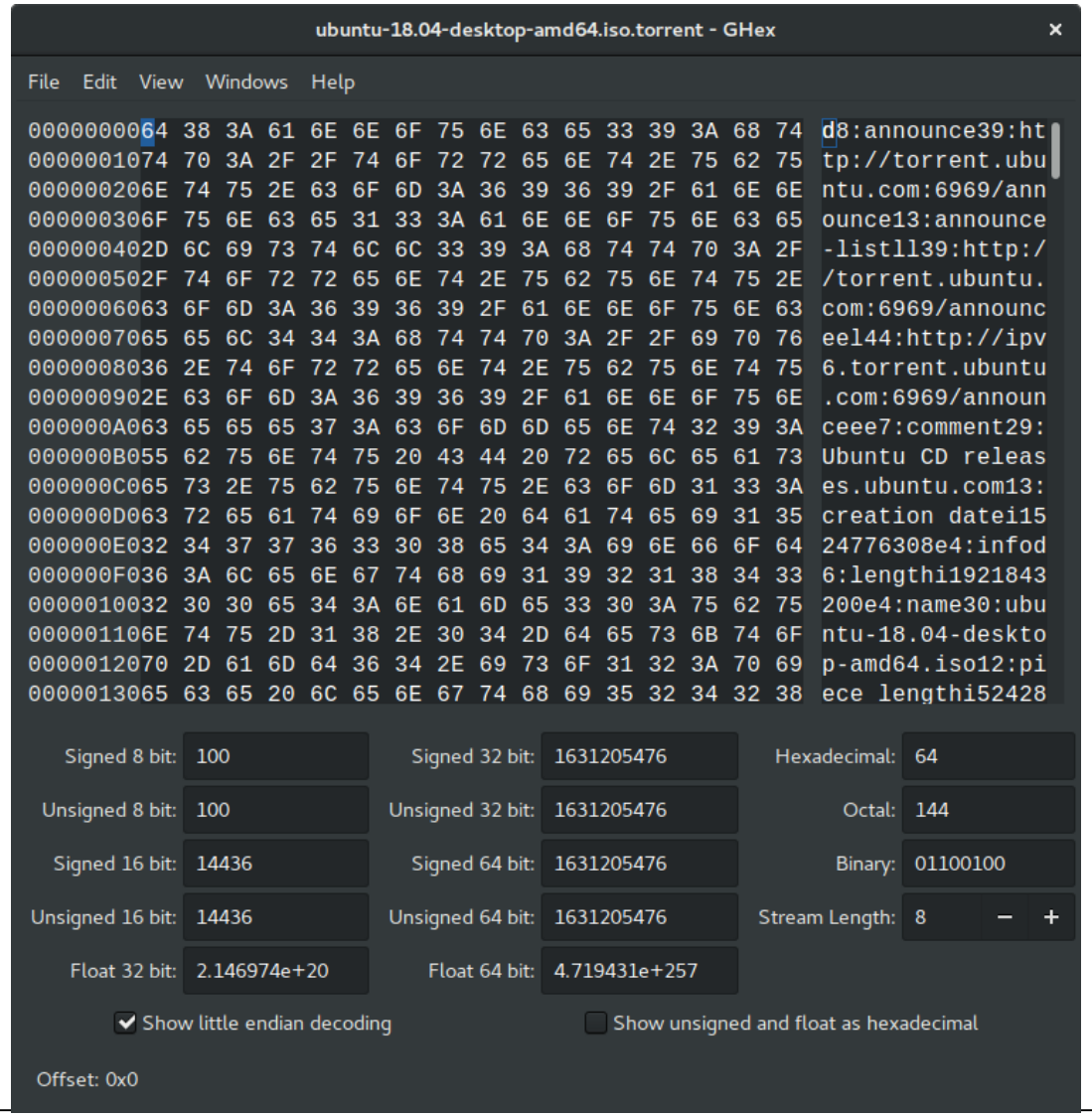
```
func main() {
  fmt.Println("Connecting...")
  req, _ := http.NewRequest("POST", "http://localhost:7000",
    strings.NewReader(`{"code": 5,"message": "Anybody there?"}`))
  req.Header.Set("Content-Type", "application/json")
  client := &http.Client{}
  resp, err := client.Do(req)
  if err != nil {
    panic(err)
  }
  defer resp.Body.Close()
  fmt.Printf("wrote request")
}
```

- **Parsing overhead, JSON slower than binary protocol - benchmarks**

```
[
    {
        "id": "bitcoin",
        "name": "Bitcoin",
        "symbol": "BTC",
        "rank": "1",
        "price_usd": "9324.08",
        "price_btc": "1.0",
        "24h_volume_usd": "9039300000.0",
        "market_cap_usd": "158560288125",
        "available_supply": "17005462.0",
        "total_supply": "17005462.0",
        "max_supply": "21000000.0",
        "percent_change_1h": "0.46",
        "percent_change_24h": "-0.27",
        "percent_change_7d": "4.5",
        "last_updated": "1525011874"
    }, ...
]
```

Protocols Bencoding and Others

- **Benconding**
  - Integers: i42e
  - Byte string: 4:test
  - List: l4:testi42ee
  - Map/dictionary: d4:test3:hsr3:tomi42ee
- **Use: BitTorrent**
- **Cap'n Proto**
  - Do not serialize, just copy, little-endian
- **Apache Arrow**
  - Do not serialize, copy, and optimally layout for memory access
- **… and many others**
- **Benchmarks, benchmarks, benchmarks**

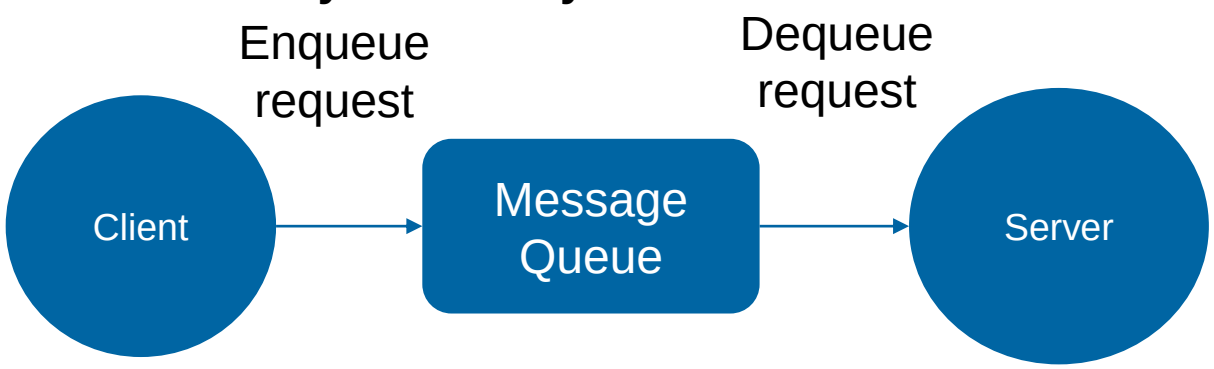


URL: b.socrative.com/student/

Room: HSR

Message Queues

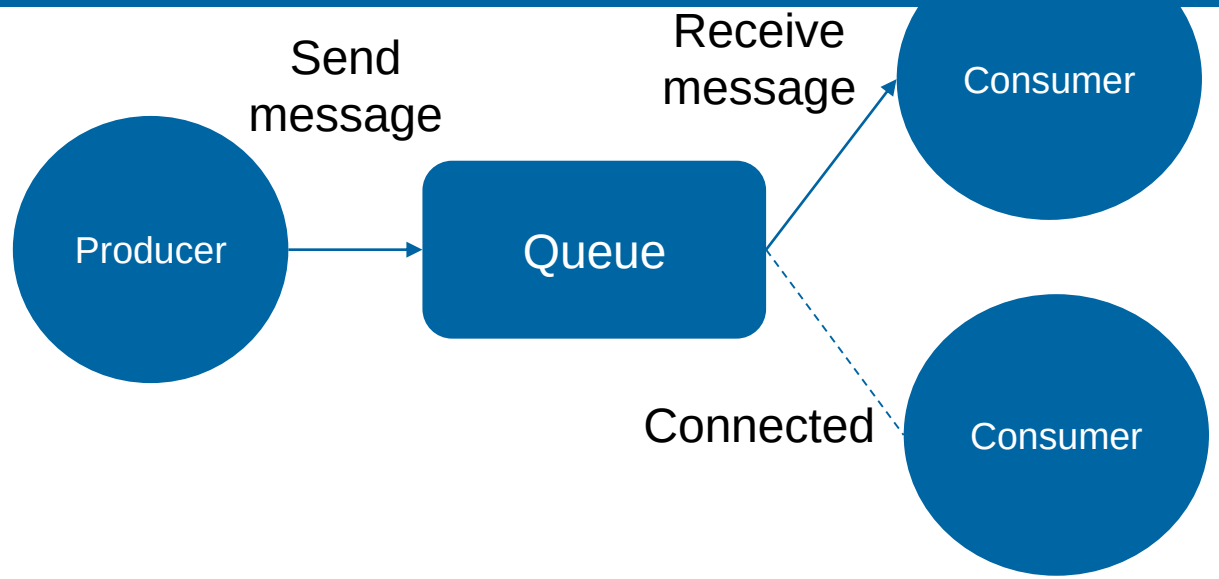
- **Message Queues in Distributed Systems**
  - Communicating without continuous connection
  - Decoupling: hardware is not reliable
  - Systems can be heterogeneous
  - Some systems “guarantee” message delivery
- **Messages placed on queue by one systems, taken by another system**

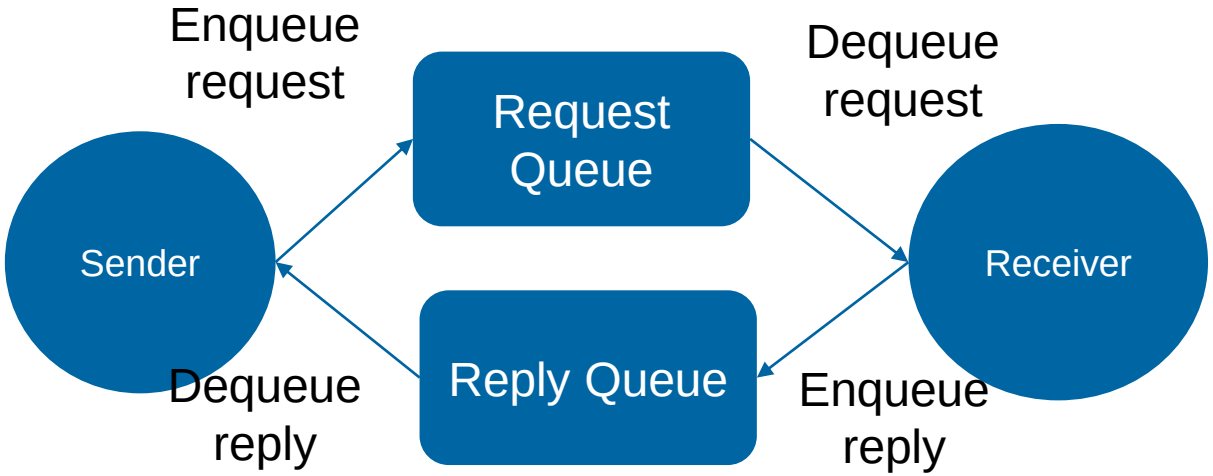


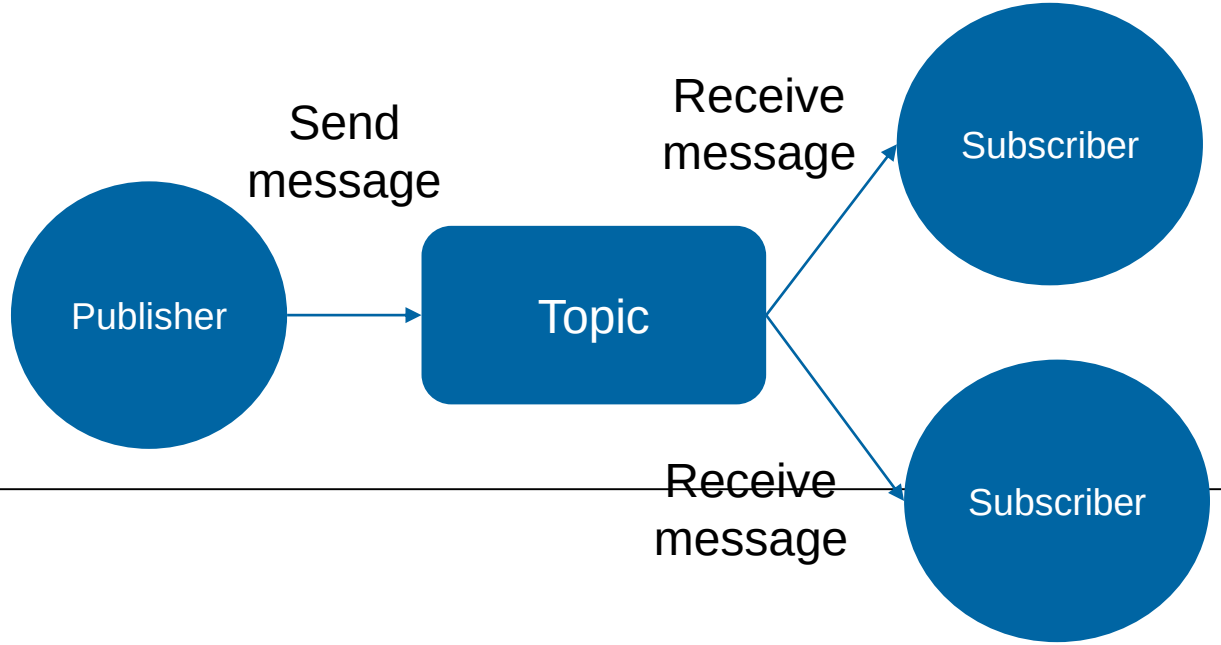
- **Channels in Go (synchronization of goroutines)**
- **Message queues and mailboxes**
  - Similar to Email
  - Software-engineering components used IPC
  - Asynchronous communications protocol
    - no need to interact with the message queue at the same time
- **Standardization**
  - Sun Microsystems' JMS specification – early attempt
    - Advanced Message Queuing Protocol (AMQP) – feature-rich message queue protocol (HTTP)
    - Streaming Text Oriented Messaging Protocol (STOMP) – simple, text-oriented message protocol (HTTP)
    - MQTT (formerly MQ Telemetry Transport) - lightweight message queue protocol especially for embedded devices (TCP/IP)

Message Queues

- **Topologies (Terminology!)**
  - Point-to-point,
  - Publish-Subscribe
  - Bidirectional Queues
- **Terminology**
  - one-to-many, many-to-one, fanout, …

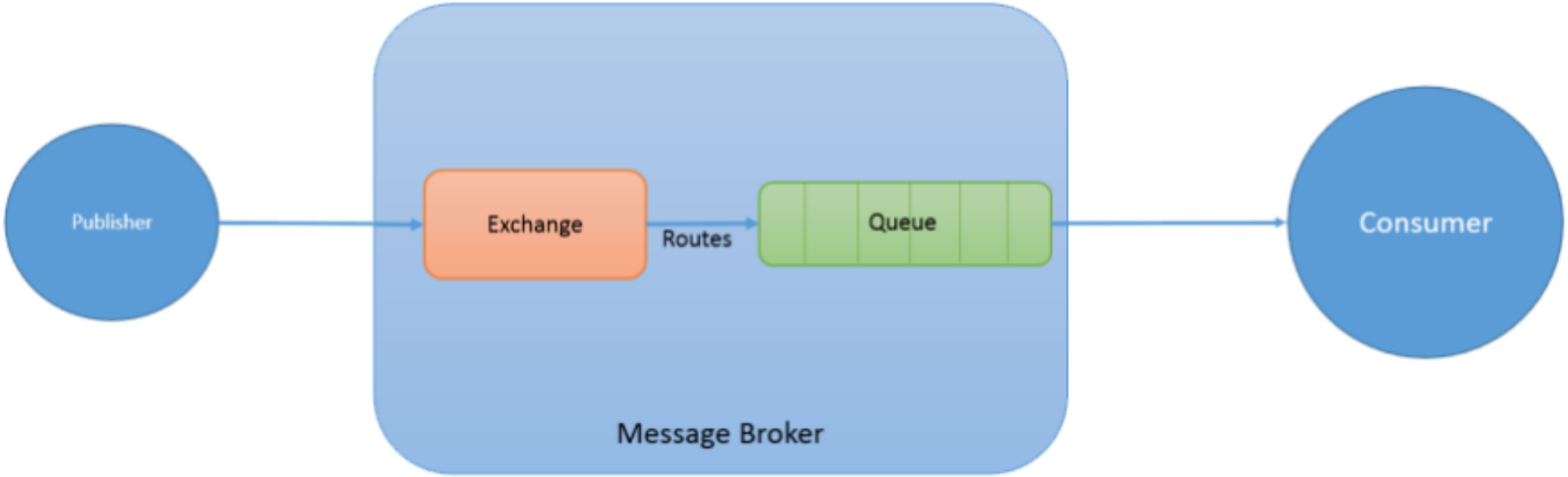






Message Queue / Apache Kafka

- **RabbitMQ (AMQP, STOMP, MQTT), ActiveMQ (AMQP, STOMP, MQTT), QPID (AMQP), ZeroMQ (ZMTP, runs without broker)**
  - Message Brokers, e.g. in AMQP
  - Exchange: routing keys selects relevant queue
  - Queue: messages are stored, consumer can get the message (durable or transient)



URL: b.socrative.com/student/

Room: HSR

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch