Final Project: Internet Radio Application

COMPUTER NETWORKS DESIGN LABORATORY

Internet Radio Application

Final project of this course!

More Advanced use of sockets.

Multithreaded programing.

Implement a simple server-client protocol.

Stream audio over UDP.

Runs on your own
GNS3 topology (from the multicast lab).

Internet Radio Application: Radio over Multicast

You will write two main programs:

1. Client
2. Server

We will provide binaries of them(Working examples) so you can test your programs step by step.

- The example programs are not perfect.
- Some errors may exist …

Client -“Controller”

- Two sockets, one TCP one UDP.
- User input, and the user’s interface with the server.
- Handshake with the server to learn the multicast address.
- Can ‘ask’ about songs.
- Receives audio packets from the server and plays music.
- Changes the multicast stream(station) to change songs.
- Can upload new songs to the server.
- Interacts with a server using a protocol.

Server

Many TCP and UDP sockets

Maintains control connections with clients.

Transmits stations via UDP to multicast

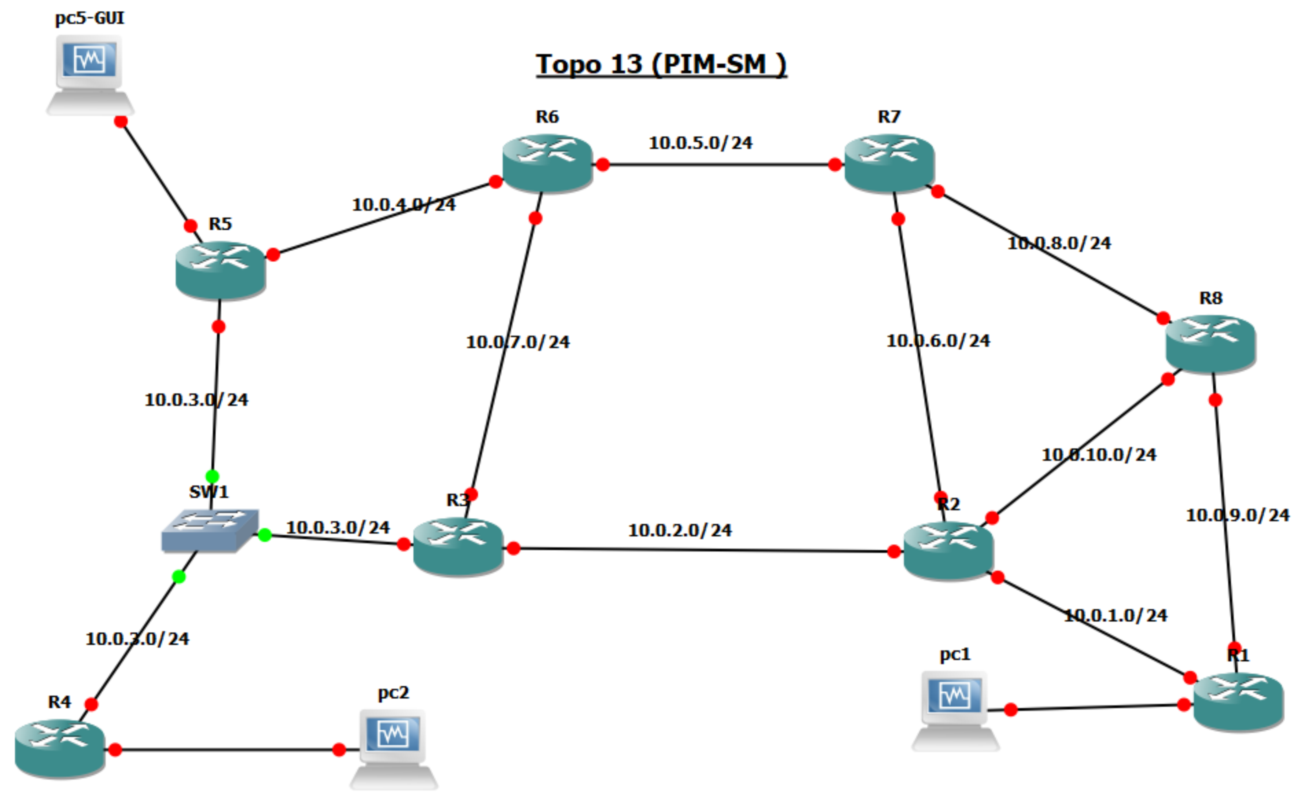
Maintains several stations

Notifies the clients about a new station.

Interacts with clients using a protocol.

Server is stateful.

Demo


Topo 13 (PIM-SM)
pc5-GUI
R5
R6
R7
R8
SW1
R3
R2
R1
R4
pc2
pc1
10.0.4.0/24
10.0.5.0/24
10.0.8.0/24
10.0.7.0/24
10.0.6.0/24
10.0.3.0/24
10.0.3.0/24
10.0.3.0/24
10.0.2.0/24
10.0.10.0/24
10.0.9.0/24
10.0.1.0/24

Server – Client Protocol

HOW DO THINGS CONNECT

Server – Client Protocol

The server and client interact using a simple protocol.

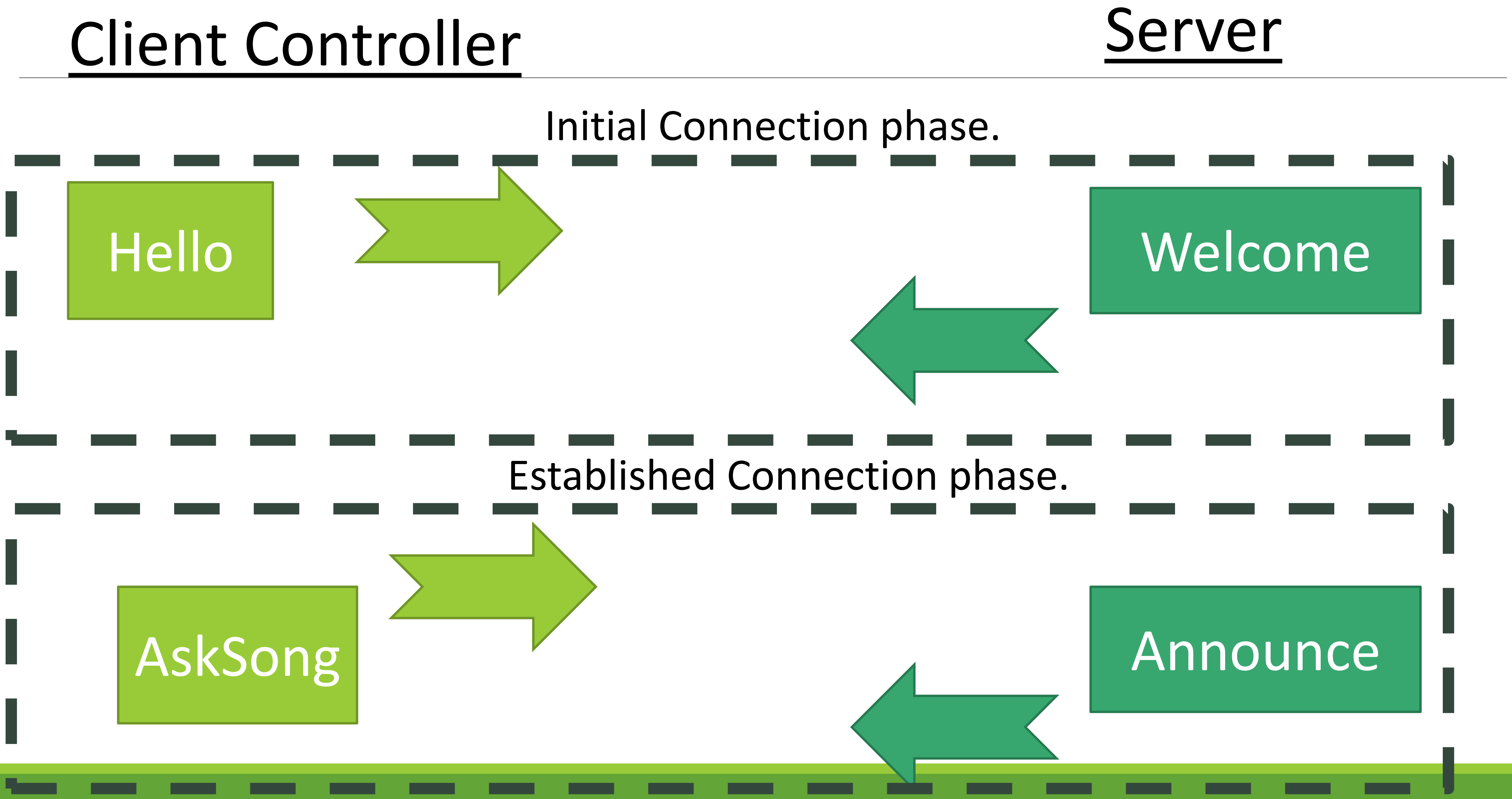
The protocol is a set of rules, messages, timeouts, etc, that both the server and client agree on.

The protocol is what makes communications possible! Any pair of client server program can successfully connect!

If either the client or the server behave in a way that is contrary to the protocol someone has to quit.

- If the server misbehaves, the client will disconnect itself
- If the client misbehaves the server will disconnect the client .

Server – Client Protocol: basics



Server – Client Protocol: messages

Lets take a closer look at the two most basic messages we saw earlier:

First, “Hello”:

```
uint8_t    commandType = 0;
uint16_t  reserved = 0;
```

Second, “Welcome”:

```
uint8_t       replyType = 0;
uint16_t     numStations;
uint32_t     multicastGroup;
uint16_t     portNumber;
```

Each message has a known size.

‘Simply’ fill a buffer with the right fields and send!

- Remember to use switch from host to network order when needed.

See “Internet Radio Application.pdf” for full details.

From the socket presentation: Socket API

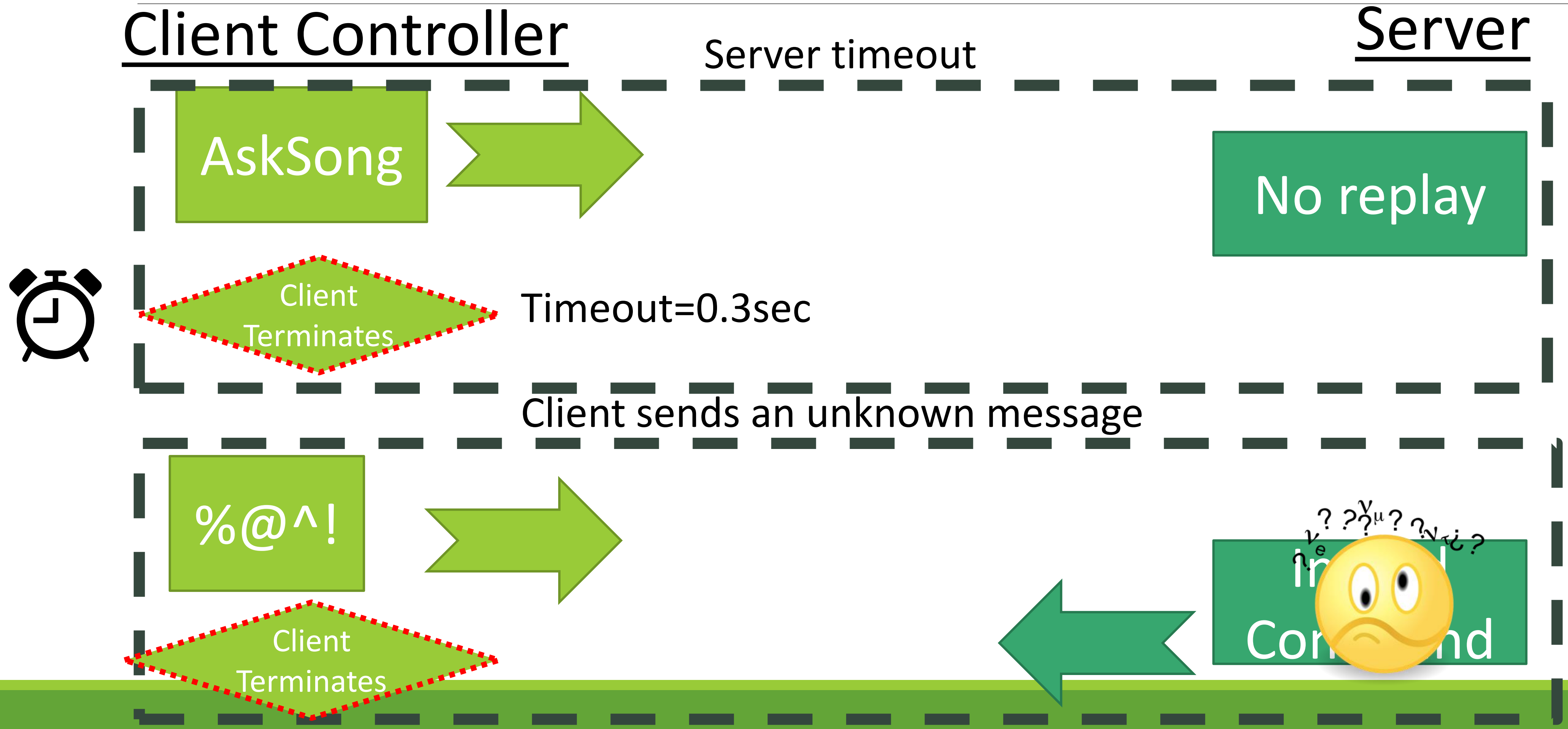
htons(), htonl(), ntohs(), ntohl(): Convert to or from host byte order to network byte order, i.e. big endian⇔ little endian.

- htons(): host to network short.
- htonl(): host to network long.
- ntohs(): network to host short.
- ntohl(): network to host long.

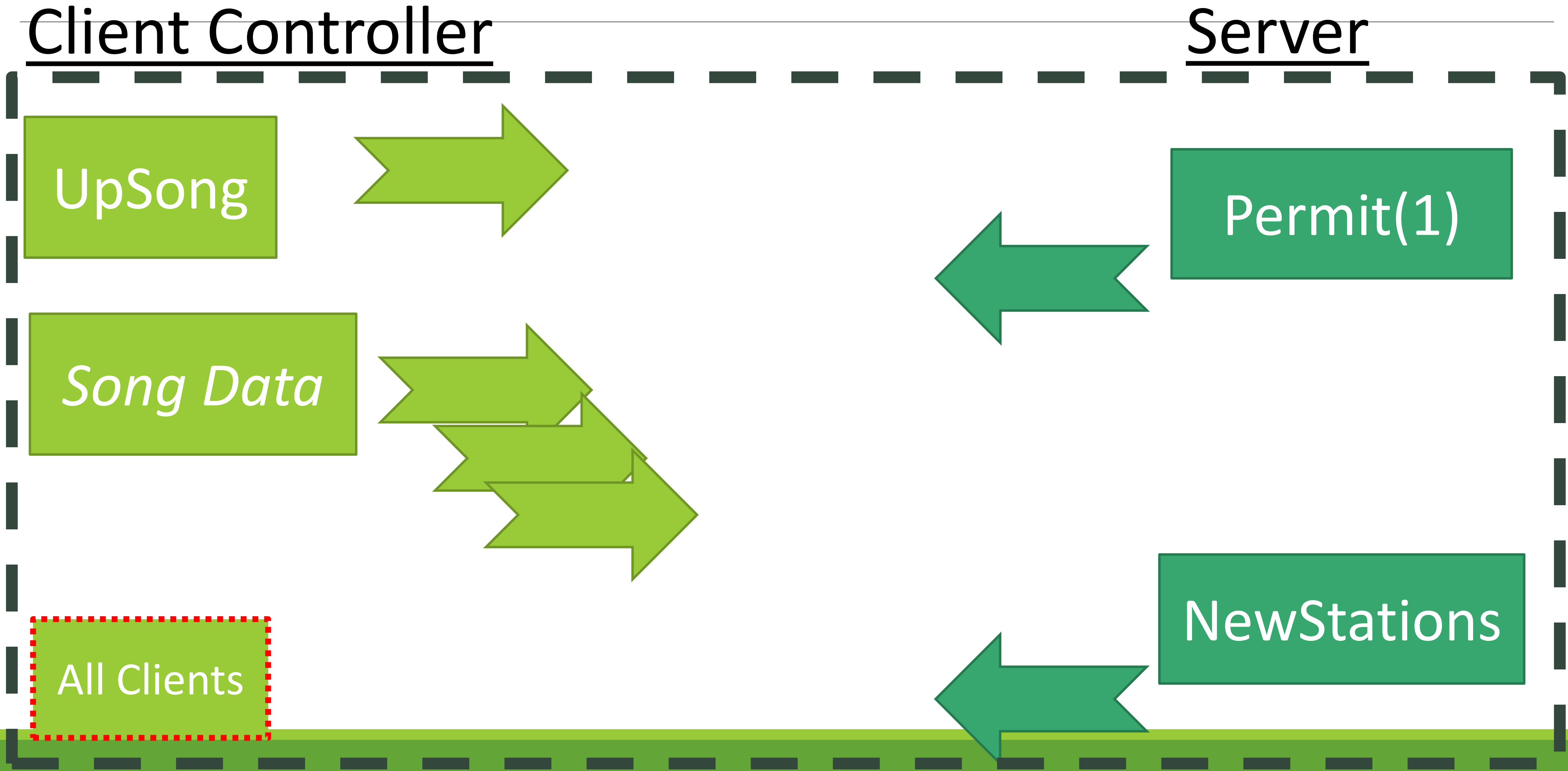
http://linux.die.net/man/3/htons

1. TCP & UDP : htons(1234);//1234=port number

Server – Client Protocol: Errors



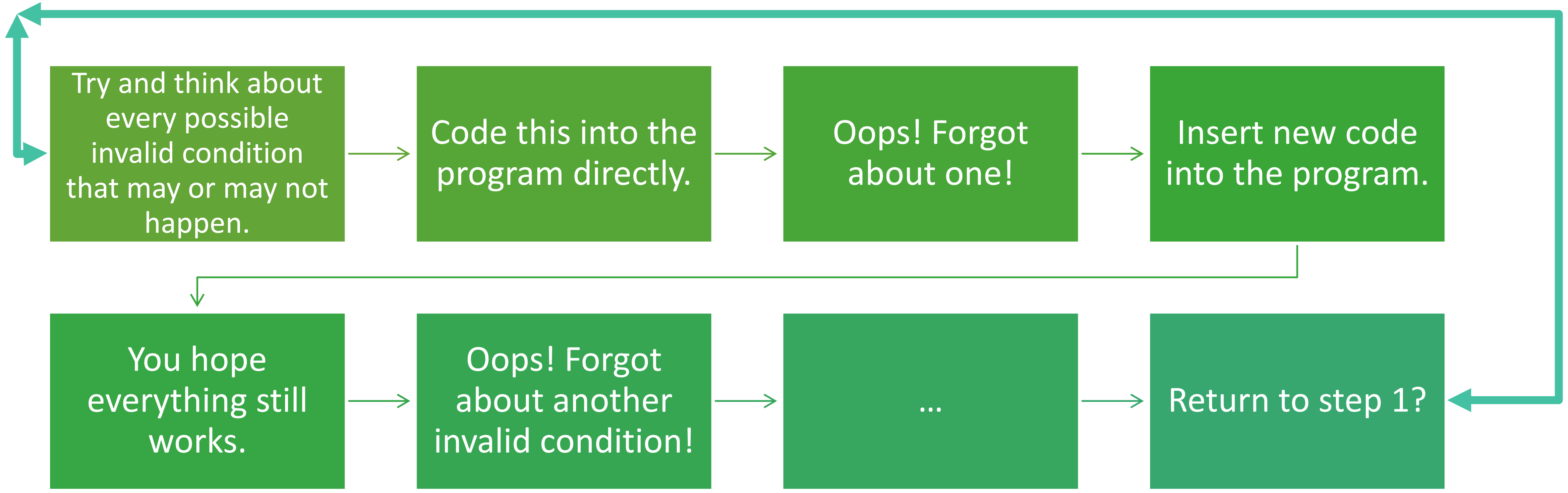
Server – Client Protocol: upload procedure



Protocol Implementation

HOW TO MAKE IT WORK

Protocol Implementation: First steps...


Try and think about every possible invalid condition that may or may not happen.
Code this into the program directly.
Oops! Forgot about one!
Insert new code into the program.
You hope everything still works.
Oops! Forgot about another invalid condition!
…
Return to step 1?

Protocol Implementation: Finite State machine

1. Find all possible “states” and “transitions” that are expected to happen.
2. Code this into the program directly.
3. Oops! Something unexpected happened!
4. Just go to the default state.

Protocol Implementation: Finite State machine

This is a very wide topic, we will try to use a very basic and intuitive approach.

We can think of the Finite State machine as a graph where each vertex is a state and each edge is a transition

What is a state?

- A state is any condition where our state machine **waits** for a **specific** input.

What is a transition

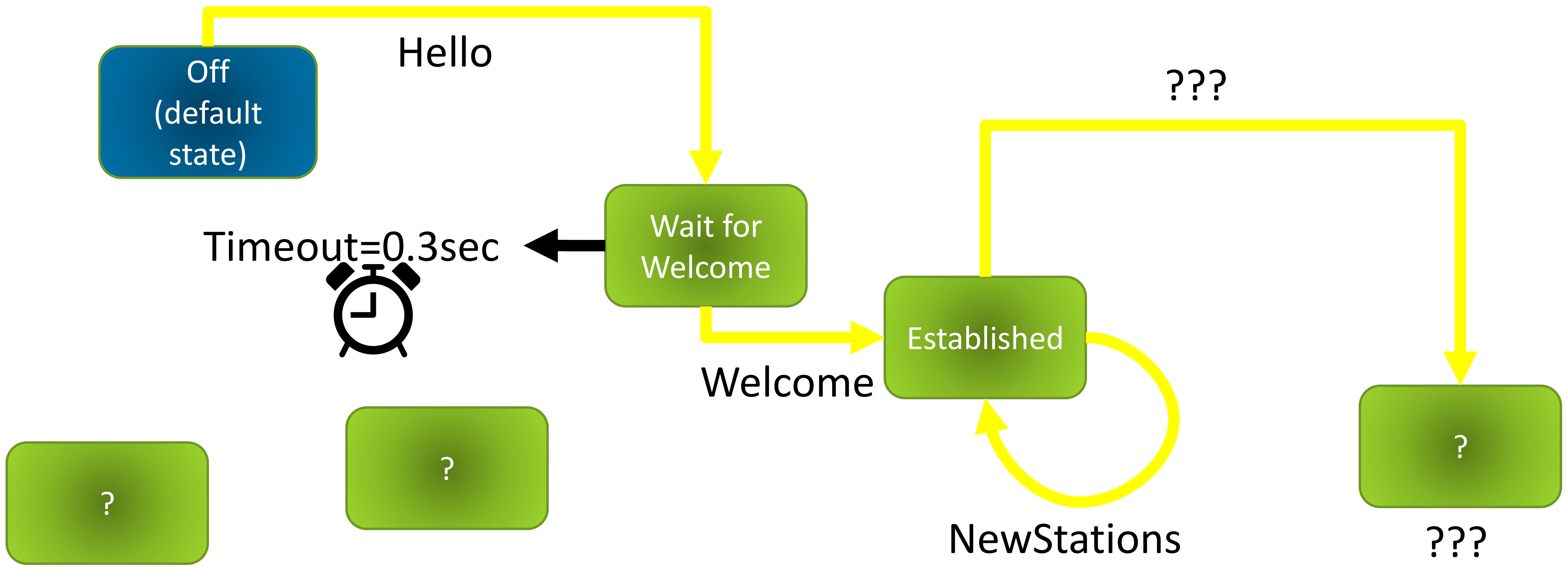
- A transition is any event which causes our state machine to change states.
- Lets assume the transitions are our input.

Note:

- We have a state Machin before in this course before. E.g. TCP and BGP
- An FSM is not a flow chart for your code.

Client's Finite State machine

To simplify things lets ignore the "invalid command" message type.
Just remember any time we get an "invalid command" we go to the "off" state



State machine of ‘client’ within server

This refers to the states of a client connection

Should have three states.

Which?

Finite State machine

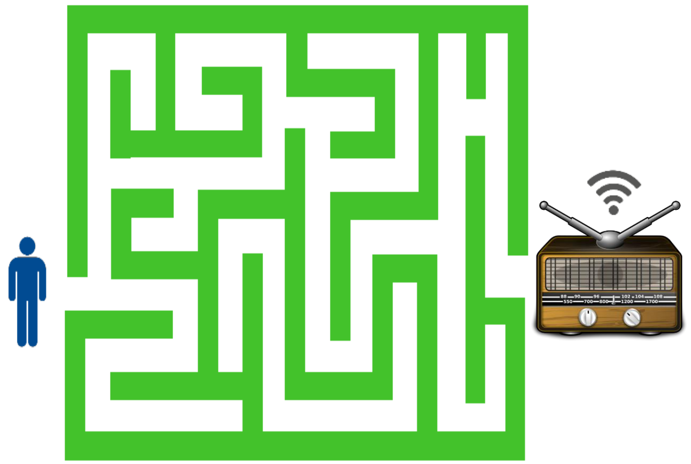
Let see how we can use our state machine in the program.

This is merely an example! you may choose to implement this very differently.

```
//Each time we get a message from the server we use this function//
int handleMsgFromServer( int CurrState , char* msg)
int newState;
switch (CurrState){
	case OFF:{newState =offStateMsgHandler(msg)}
	case WelcomeWait:{newState = WelcomeWaitMsgHandler(msg)}
	case Established:{newState = EstablishedMsgHandler(msg)}
…
	default: {terminate() /**/basically go to the “off state” /**/ }
…
```

Finite State machine cont.

```
//each time we get a welcome from the server we go to this function//
int WelcomeWaitMsgHandler( char* msg)
int newState, msgType=msg[0];
switch (msgType){
        case WELLCOME:{
                handleWelcom(msg);
                newState =ESHTABLISHED;
                        }
        case OTHER???:{newState = ?}
…
                default: {newState =OFF/**/basically go to the "off state" /**/ }
Return newstate;
```



Where to start

OR: THE DEMAND PAPER IS TOO LONG!!!

Design paper

You need to submit a **short** design paper for both the server and client.

For the client you will also need to submit the complete FSM we started to construct in this presentation.

We will go over each design paper with each pair of students.

Full requirements are listed in “Programing project Labs Planning.pdf”.

The design paper is almost free but mandatory five points out of your grade in this project

Demand paper

The demand paper, i.e. *"Internet Radio Application 2022.pdf"* contains **everything** you'll need to know about the project.

Use the handbook for a shorter read.

- It contains the same text, ordered with the same numbers for easier reference .
- Doesn't contain some specifics, examples, longer explanations.
- However if you'll still have to read the full demand paper to make sure you comply with all demands…

Start small, the Client.

You could start with the basic protocol, without the upload procedure, just remember it's going to be there in the end.

The server is more complex.

A word about plagiarism

DON’T.

Project schedule

28/12/23

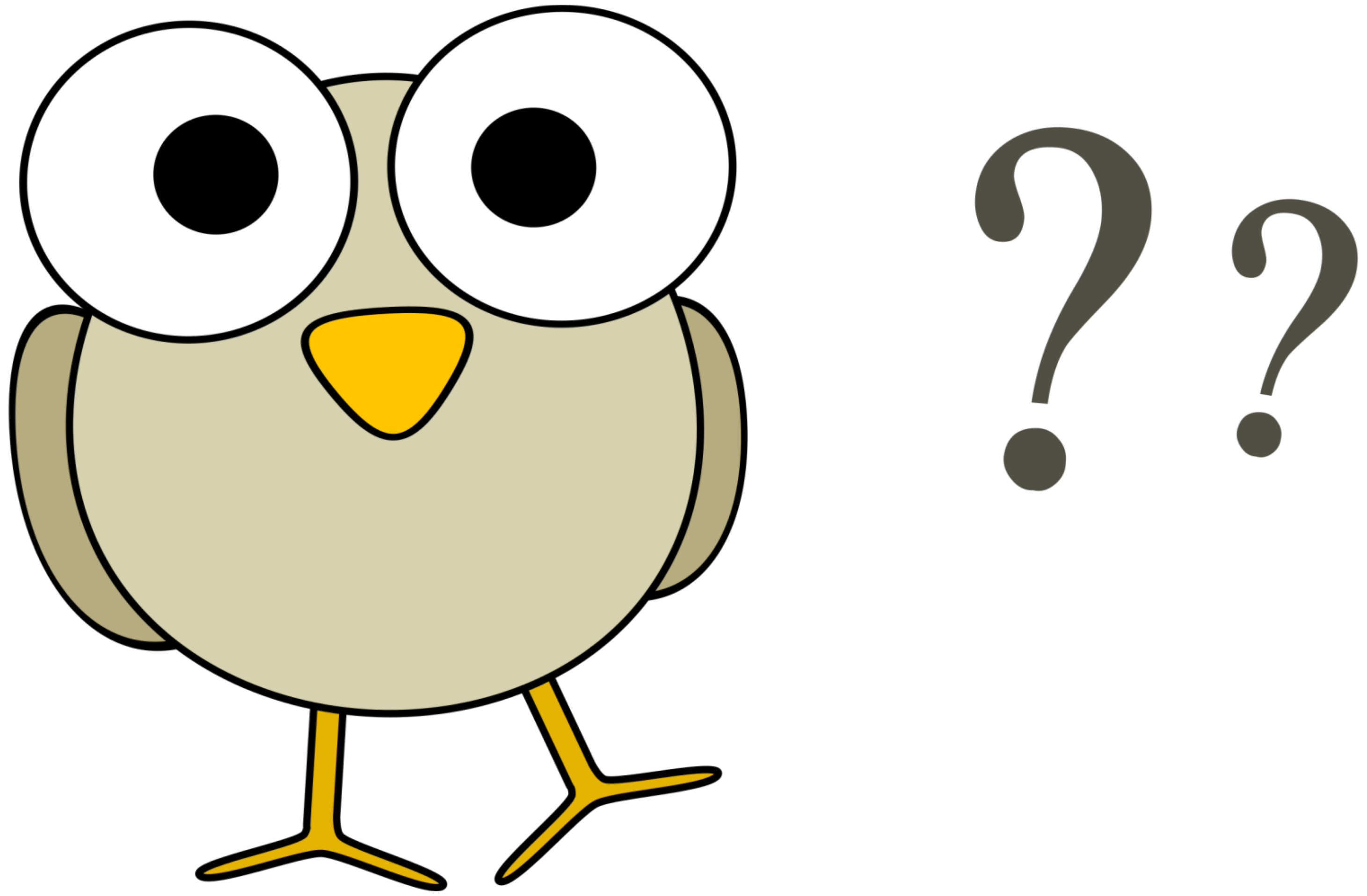
- Design paper submission.

15?/1/23

- Finale project submission.

15/1/23

- Project defense.



Queries?