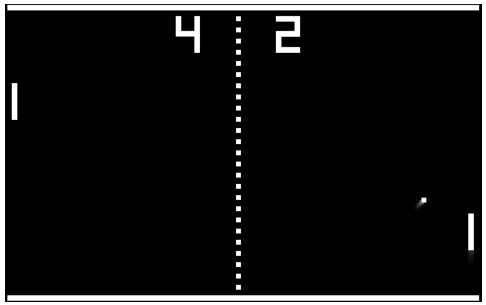
A game of Pong

Johan Montelius

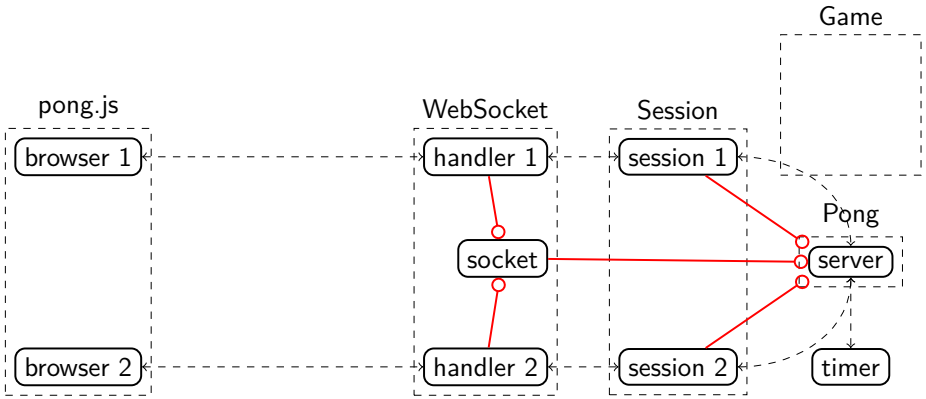
KTH

VT23

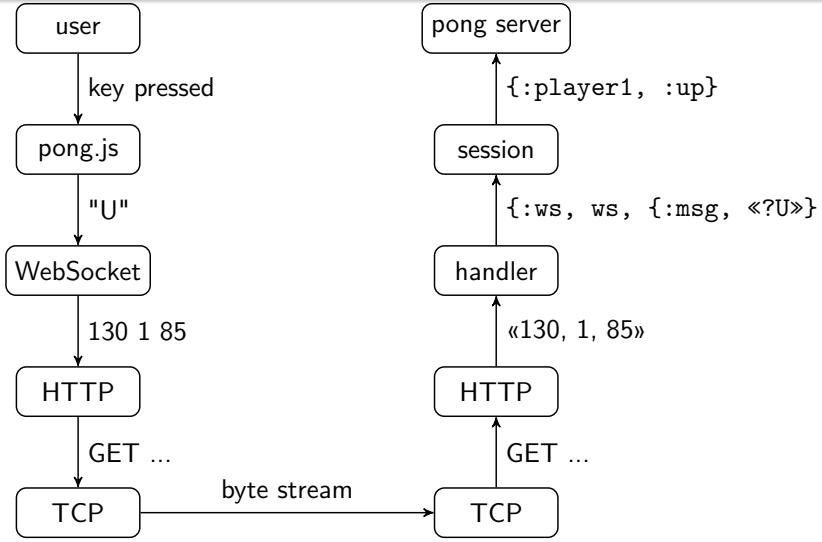
the classical game of Pong



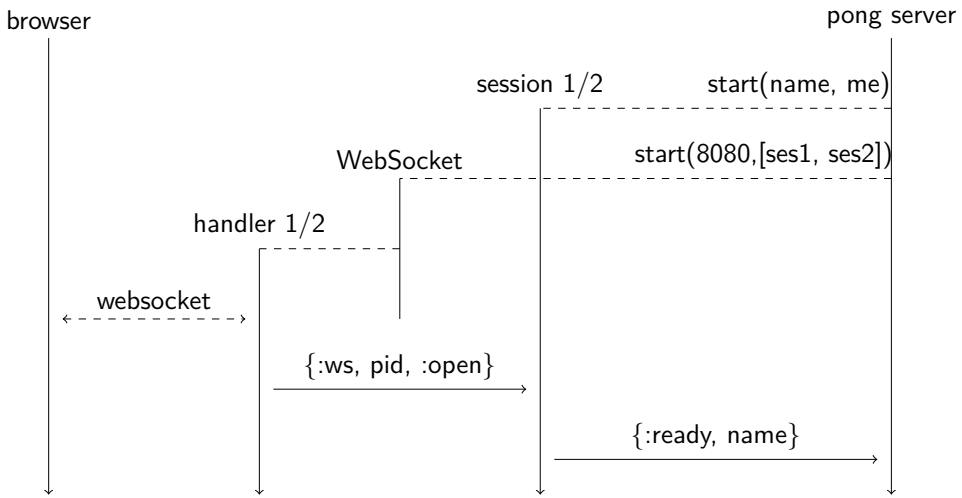
architectur



layers of communication

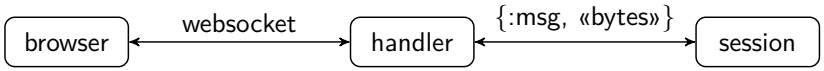


sequence diagram



WebSocket interface

The Javascript client communicate using a websocket interface. After the initial HTTP handshake, a bidirectional message channel is created. Each message consist of sequence of bytes.



The handler process will take care of the WebSocket internals and deliver a stream of messages to the session process.

Each client is connected through a unique handler process that is communicating with a single session process.

Session handler lower interface

Messages from the websocket handler to the session handler:

- `{:ws, pid, :open}` : a connection was establishes
- `{:ws, pid, :close}` : the connection was closed by the client
- `{:ws, pid, {:msg, «byte encoded message»}}` : message from the client

Messages from the session handler to the websocket handler:

- `{:frw, «byte encoded message»}` : encode and send message to client
- `:stop` : time to close the connection

The session handler

Works as a decoder/encoder of byte-encoded messages and Elixir messages.

Messages from the client forwarded to the pong game server:

- `«?U»` : player pressed up - `{name, :up}`
- `«?D»` : player pressed down - `{name, :down}`

session handler

Messages from pong server forwarded to the websocket handler:

- `{:player1, :up}` : player moved up - `«?P,?U»`
- `{:player1, :down}` : player moved down - `«?P,?D»`
- `{:player1, :score, score}` : player scored - `«?P,?S, score»`
- `{:player2, ... }` : same messages for opponent - `«?O, ... »`
- `{:ball, x, y }` : ball moved to new position - `«?B, x::16, y::16 »`
- `{:frw, msg}` : raw message to client - `msg`

the game engine

The game server:

- create two session handlers with unique names
- create a WebSocket process, give session handlers as arguments
- wait for session handlers to report
- start the game

the game engine

The pong engine keeps a state consisting of:

- Two players (`player1` and `player2`).
- A ball.
- The current score.
- Two *session pids* to send messages to the players.

The pong engine is defined by two modules:

- The Pong module that describes the server as a communicating process.
- The Game module that describe the rules as functions.

Rules of the game

A player is represented as a tuple `{name, x-pos, y-pos, dir}`.

- `player1(name)` : return a named `player 1`
- `player2(name)` : return a named `player 2`
- `up(player)` : return either `{:ok, player}` or `:no`
- `down(player)` : return either `{:ok, player}` or `:no`

A ball is represented a as tuple `{:ball, x, y, dx, dy}`.

- `serve(player)` : return `{pos, ball}`
- `move_ball(player1, player2, ball)` : return either
  - `{:bounce, pos, ball}` (increase speed)
  - `{:moved, pos, ball}`
  - `{:score, name}`

the Game module

The Game module provides only functions; it does not keep a state.

The Game module knows how large the court is.

The state (apart from the score) is held by the three data structures: player1, player2 and the ball.