SCALING ERLANG WEB APPLICATIONS
100 TO 100K USERS AT ONE WEB SERVER

Fernando Benavides (*@elbrujohalcon*)

Inaka Labs

March 27, 2012

HELLO WORLD!

- I'm a developer since I was 10
- I worked with Visual Basic, C#, .NET, Javascript . . .
- I switched to functional programming in 2008
- I wrote my thesis project in Haskell
- I'm an Erlang developer since then

HELLO WORLD!

- I'm a developer since I was 10
- I worked with Visual Basic, C#, .NET, Javascript . . .
- I switched to functional programming in 2008
- I wrote my thesis project in Haskell
- I'm an Erlang developer since then

HELLO WORLD!

- I'm a developer since I was 10
- I worked with Visual Basic, C#, .NET, Javascript . . .
- I switched to functional programming in 2008
- I wrote my thesis project in Haskell
- I'm an Erlang developer since then

HELLO WORLD!

- I'm a developer since I was 10
- I worked with Visual Basic, C#, .NET, Javascript . . .
- I switched to functional programming in 2008
- I wrote my thesis project in Haskell
- I'm an Erlang developer since then

HELLO WORLD!

- I'm a developer since I was 10
- I worked with Visual Basic, C#, .NET, Javascript . . .
- I switched to functional programming in 2008
- I wrote my thesis project in Haskell
- I'm an Erlang developer since then

INAKA

TODO: Chad's speach for this

Introduction
Match Stream
Scaling
Final Words

Description
Scope

INTRODUCTION

My talk is on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

It's a design pattern seen in many places:

- Chat Rooms
- Social Sites
- Sport Sites

Introduction
Match Stream
Scaling
Final Words

Description
Scope

INTRODUCTION

My talk is on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

It's a design pattern seen in many places:

- Chat Rooms

- Social Sites

- Sport Sites

Introduction
Match Stream
Scaling
Final Words

Description
Scope

INTRODUCTION

My talk is on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

It's a design pattern seen in many places:

- Chat Rooms

- Social Sites

- Sport Sites

Introduction
Match Stream
Scaling
Final Words

Description
Scope

INTRODUCTION

My talk is on the scalability of a *web* project that has an *HTTP API* and a component that keeps clients *connected* to the server for *long periods* of time.

It's a design pattern seen in many places:

- Chat Rooms
- Social Sites
- Sport Sites

Introduction
Match Stream
Scaling
Final Words

Description
Scope

SCOPE

We will improve the way we use

- OTP behaviours
- TCP and HTTP connections
- Underlaying system configurations

*We will **not** deal with*

- Multiple machines/nodes
- Database choices and/or implementations

Introduction
Match Stream
Scaling
Final Words

Description
Scope

SCOPE

We will improve the way we use

- OTP behaviours
- TCP and HTTP connections
- Underlaying system configurations

*We will **not** deal with*

- Multiple machines/nodes
- Database choices and/or implementations

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
GENERAL IDEA

A soccer match is played at some stadium

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
GENERAL IDEA

Soccer fans are connected to the internet in their offices

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
GENERAL IDEA

A reporter is at the stadium with his device

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
GENERAL IDEA

MATCHSTREAM *connects* them in *real time*
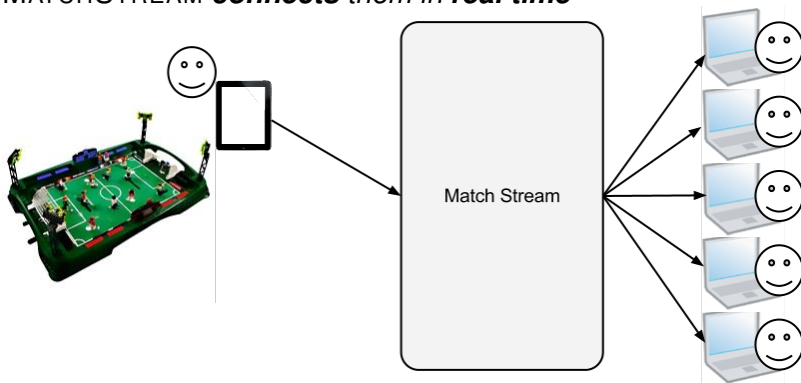
Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
REQUIREMENTS

SYSTEM CHALLENGES

- Many concurrent users connecting at the same time
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
REQUIREMENTS

SYSTEM CHALLENGES

- Many concurrent users connecting at the same time
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

Introduction
Match Stream
Scaling
Final Words

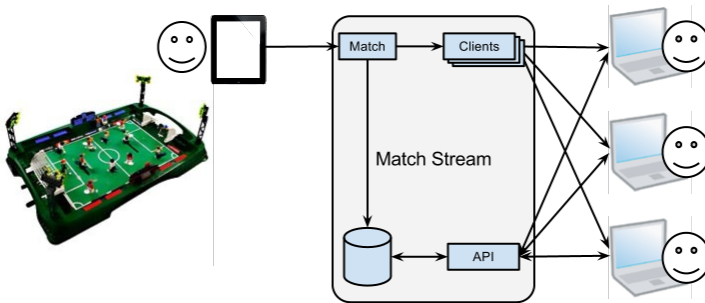**Idea**
Design

MATCH STREAM
REQUIREMENTS

SYSTEM CHALLENGES

- Many concurrent users connecting at the same time
- Two-hour-long bursts of connections followed by long periods of inactivity
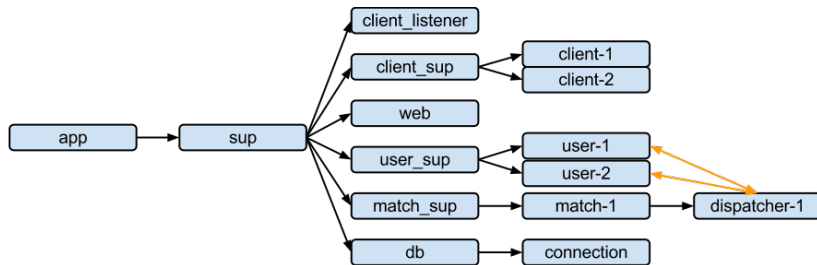- Real-time updates

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
GENERAL DESIGN

Introduction
Match Stream
Scaling
Final Words

Idea
Design

MATCH STREAM
ARCHITECTURE

Introduction
Match Stream
Scaling
Final Words

Idea
Design
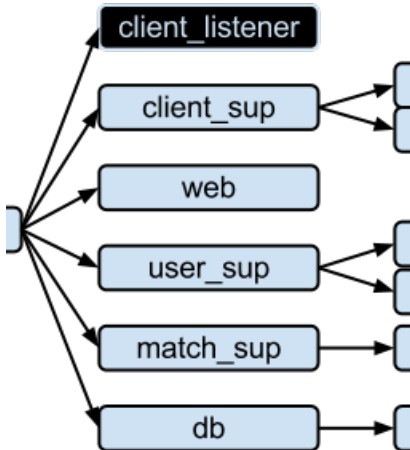
COMPONENTS

CLIENT_LISTENER gen_server.
Listens on a TCP
port to receive
client connections

Introduction
Match Stream
Scaling
Final Words

Idea
Design
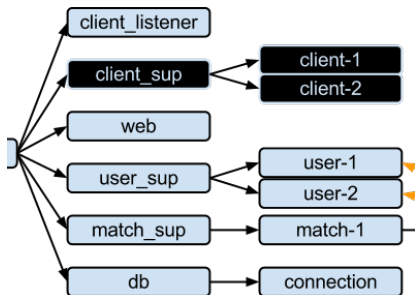
COMPONENTS

CLIENT_SUP supervisor.
Supervises
connection
processes

CLIENT gen_fsm.
Handles a TCP
connection

Introduction
Match Stream
Scaling
Final Words

Idea
Design

COMPONENTS

WEB mochiweb server.
Listens for HTTP
API calls

Introduction
Match Stream
Scaling
Final Words

Idea
Design
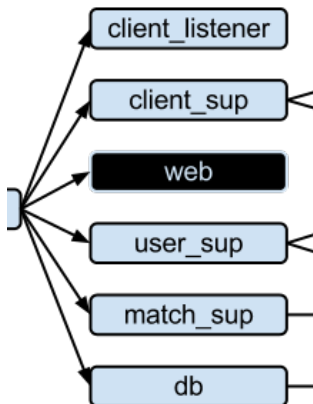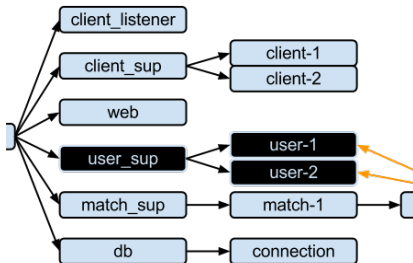
COMPONENTS

USER_SUP supervisor.
Supervises user
processes

USER gen_server.
Subscribes to
match dispatchers
and sends events
to clients

Introduction
Match Stream
Scaling
Final Words

Idea
Design
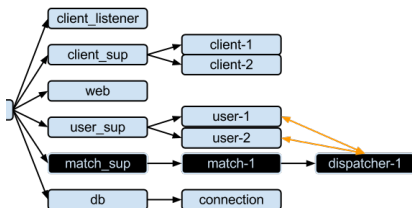
COMPONENTS

MATCH_SUP supervisor. Supervises match processes

MATCH gen_server. Listens to match events, stores them

DISPATCHER gen_event dispatcher. Delivers match events

Introduction
Match Stream
Scaling
Final Words

Idea
Design

COMPONENTS
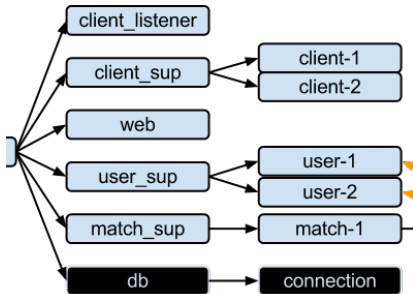
DB `gen_server`.
Processes database operations

CONNECTION `erldis client`.
Handles the connection to the database

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

LESSON LEARNED

*Simply using Erlang to build your system is **not enough** to ensure **scalability***

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

MEASURES

- N *Connections*. Number of connections the server can handle
- C *Concurrency*. Number of multiple connections starting at a time
- ART *Average Response Time*. How much does it take for the server to send an event

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

Tools

Test Client

We create our own test client for TCP connections

ApacheBench

To test API calls

entop

We use it to check server processes

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 0
ESTABLISHING A BASELINE

GOALS

- Find how much the system can handle

STEPS

- Create automated testers

- Install and start the system on a *clean* machine

- Run the tests on the server's local network

- Test repeatedly adjusting our parameters to maximize them

- Have a human using the system himself

STAGE 0
ESTABLISHING A BASELINE

GOALS

- Find how much the system can handle

STEPS

- Create automated testers
- Install and start the system on a *clean* machine
- Run the tests on the server's local network
- Test repeatedly adjusting our parameters to maximize them
- Have a human using the system himself

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 1
RESULTS

N **1000**
C **4**
ART **26s**

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 1
TUNE THE OS AND THE VM

GOALS

- Improve the underlying Operating System
- Improve the Erlang VM Configuration

SETTINGS TO TUNE UP

- Open files limit
- TCP connections limit
- TCP backlog size
- TCP memory allocation
- Number of Erlang processes

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 1
TUNE THE OS AND THE VM

GOALS

- Improve the underlying Operating System
- Improve the Erlang VM Configuration

SETTINGS TO TUNE UP

- Open files limit
- TCP connections limit
- TCP backlog size
- TCP memory allocation
- Number of Erlang processes

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 1
RESULTS

N **4000**
C **4**
ART **35s**

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning
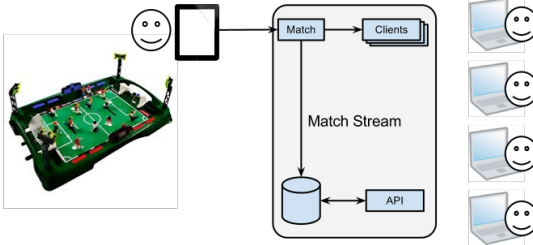
STAGE 2
IMPROVING MATCH STREAM

We can't blame the machine anymore, we need to improve our system

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.1
CONNECTION TWEAKS

BACKLOG

- Allow more concurrent connections
- Don't forget TCP tuning your HTTP server

STAGE 2.1
CONNECTION TWEAKS

CLIENT_LISTENER

```erlang
gen_tcp:listen(Port,
  [binary, {packet, line}, {keepalive, true},
   {active, false}, {reuseaddr, true},
   {backlog, 128000}, {send_timeout, 32000},
   {send_timeout_close, true}]).
```

WEB

```erlang
mochiweb_http:start(
  [{name, ?MODULE}, {loop, {?MODULE, loop}},
   {backlog, 128000}, {port, Port}]).
```

STAGE 2.1
CONNECTION TWEAKS

OUTBOUND CONNECTIONS

- e.g., database connections
- Don't use just one of them
- You may have separated connections for different purposes

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.1
CONNECTION TWEAKS

```
-define(REDIS_CONNECTIONS, 200).
-record(state, {redis :: [pid()] }).
...
init([]) ->
  ...
  Redis =
    lists:map(
      fun(_) ->
        {ok, Conn} = erldis_client:start_link()
        Conn
      end, lists:seq(1, ?REDIS_CONNECTIONS) ),
  {ok, #state{redis = Redis}}.
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.1
CONNECTION TWEAKS

```erlang
handle_call(Request, From, State) ->
    [RedisConn|Redis] = State#state.redis,
    proc_lib:spawn_link(
        fun() ->
            Res = handle_call(Request, RedisConn),
            gen_server:reply(From, Res)
        end),
    {noreply, State#state{redis =
                        Redis ++ [RedisConn]}}.
```

STAGE 2.1
CONNECTION TWEAKS

LISTENERS

- You can listen to more than one port
- For unified urls, use *nginx* in front of the server

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.1
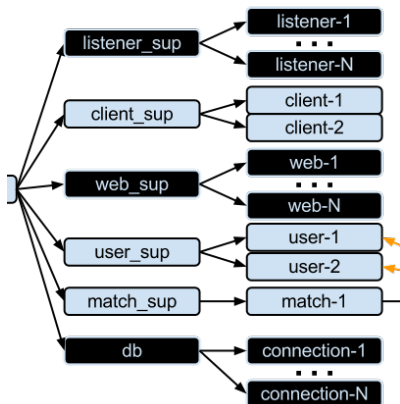## CONNECTION TWEAKS

```erlang
init([]) ->
  ...
  Listeners =
    [{list_to_atom("client-listener-" ++
                    integer_to_list(I)),
      client_listener, start_link, [I],
      permanent, brutal_kill, worker,
      [client_listener]}
    || I <- lists:seq(MinPort, MaxPort)],
  {ok, {{one_for_one, 5, 10}, Listeners}}.
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.1
CONNECTION TWEAKS

LISTENER `gen_server`.
Listens on a TCP port to receive client connections

WEB `mochiweb server`.
Listens for HTTP API calls on a particular port

CONNECTION `erldis client`.
Handles the connection to the database

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.1
RESULTS

N **8000**

C **500**

ART **15s**

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning
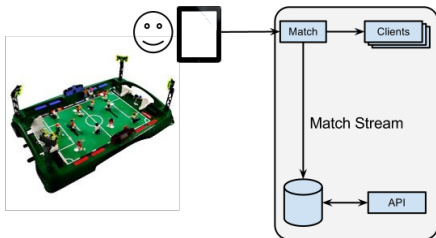
STAGE 2.2
GEN_EVENT

SUP_HANDLER

- Don't use it
- Monitor the processes instead

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.2
GEN_EVENT

```
EvtMgr =
  match_stream_match:event_manager(MatchId),
ok =
  gen_event: add_handler (EvtMgr,
    {?MODULE, {MatchId,UserId,Client}}, self()),
MgrRef = erlang:monitor(process, EvtMgr),
ClientRef = erlang:monitor(process, Client),
{reply, ok,
  State#state{matches =
    [{Client, MatchId, ClientRef, MgrRef}
      | State#state.matches]}}
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.2
GEN_EVENT

```erlang
handle_info({'DOWN', Ref,_,Client,_}, State) ->
  ...
  case lists:keytake(Ref, 4, State#state.matches) of
    {value, {Client,_,CRef,Ref}, OtherMatches} ->
      ...
```

STAGE 2.2
GEN_EVENT

LONG DELIVERY QUEUES

- Distribute the work
- Use *repeaters*

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.2
GEN_EVENT

```erlang
start_link(Name, Source) ->
  {ok, Pid} = gen_event:start_link(Name),
  ok = gen_event:add_handler(
         Source, {?MODULE, Pid}, Pid),
  {ok, Pid}.
...
init(Repeater) ->
  Ref = erlang:monitor(process, Repeater),
  {ok, #state{mgr = Repeater, ref = Ref}}.
...
handle_event(Event, State) ->
  gen_event:notify(State#state.mgr, Event),
  {ok, State}.
```
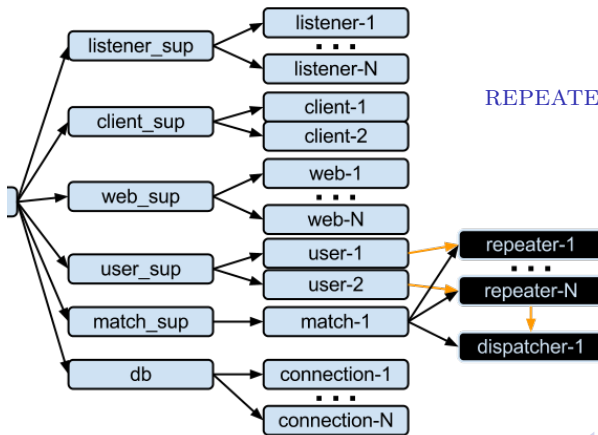
Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.2
GEN_EVENT

REPEATER gen_event dispatcher. It's subscribed to dispatcher and *repeats* the received events to its subscribers

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

Stage 2.2
Results

N **8000**
C **1000**
ART **2s**

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning
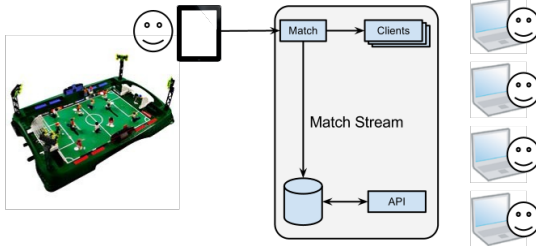
STAGE 2.3
GEN_SERVER

CALL TIMEOUTS
Remember gen_server:reply/2

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.3
GEN_SERVER

```erlang
handle_call(Request, From, State) ->
  [RedisConn|Redis] = State#state.redis,
  proc_lib:spawn_link(
    fun() ->
      Res = handle_call(Request, RedisConn),
      gen_server:reply(From, Res)
    end),
  {noreply, State#state{redis =
                        Redis ++ [RedisConn]}}.
```

STAGE 2.3
GEN_SERVER

MEMORY FOOTPRINT
Remember `hibernate`

Puts the calling process into a wait state where its memory allocation has been reduced as much as possible, which is useful if the process does not expect to receive any messages in the near future.

(Erlang Docs)

STAGE 2.3
GEN_SERVER

```erlang
handle_cast(Event, State) ->
  ...
  {noreply, State, hibernate}.

...

handle_call(Request, _From, State) ->
  ...
  {reply, Reply, State, hibernate}.
```

STAGE 2.3
GEN_SERVER

LONG STARTUP TIME

- Initialize your gen_servers in a 0 timeout
- Move initialization code to handle_info

STAGE 2.3

GEN_SERVER

```erlang
init(UserId) ->
  {ok, #state{user = UserId}, 0}.

...

handle_info(timeout, State) ->
  case match_stream_db:user(State#state.user) of
  ...
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.3
RESULTS

N **40K**

C **5K**

ART **25ms**

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning
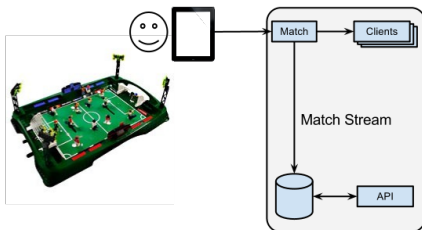
STAGE 2.4
SUPERVISORS

SIMPLE ONE FOR ONE SUPERVISORS

- Sometimes `simple_one_for_one` supervisors get overburdened because they have too many children
- Use a supervisor hierarchy

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.4
SUPERVISORS

```erlang
init([]) ->
  _ = random:seed(erlang:now()),
  Managers =
    [{list_to_atom("user-manager-" ++
                    integer_to_list(I)),
      {user_mgr, start_link, [I]},
      permanent, brutal_kill, supervisor,
      [user_mgr]}
     || I <- lists:seq(1, ?MANAGERS) ],
  {ok, {{one_for_one, 5, 10}, Managers}}.
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning
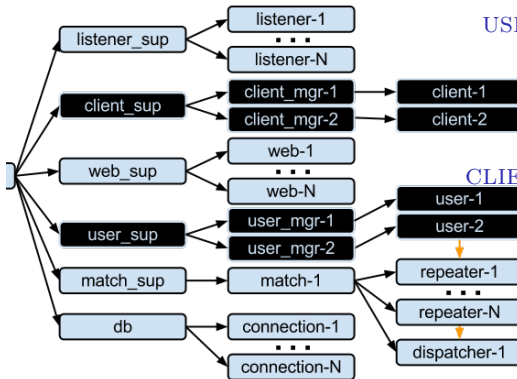
STAGE 2.4
SUPERVISORS

```erlang
start_user(User) ->
  Manager =
    list_to_atom(
      "user-manager-" ++
        integer_to_list(random:uniform(?MANAGERS))),
  supervisor:start_child(Manager, [User]).
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.4
SUPERVISORS

USER_MGR supervisor.
Supervises a group of users processes

CLIENT_MGR supervisor.
Supervises a group of connection processes

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning
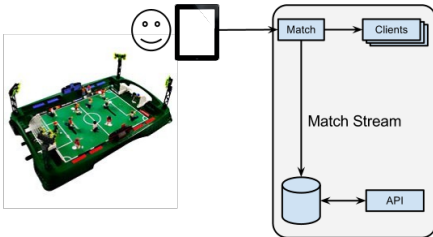
STAGE 2.4
RESULTS

N **50K**
C **8K**
ART **25ms**

STAGE 2.5
OTHER PROCESSES

LOGGING

- Use a good logging system
- Choose carefully what to log

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.5
OTHER PROCESSES

REGISTRATION

- If everybody agrees on a name, nobody has to find where it is
- You can always register processes both locally and globally

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

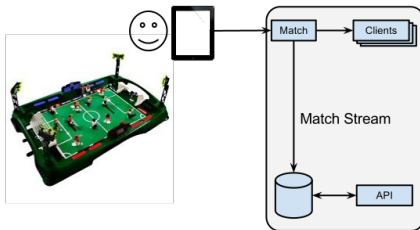# STAGE 2.5
OTHER PROCESSES

```erlang
event_manager(MatchId) ->
  binary_to_atom(<<"event_mgr@", MatchId/binary>>, utf8).

init(MatchId) ->
  {ok, EventMgr} =
    gen_event:start_link(
      {local, event_manager(MatchId)} ),
  ...
```

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 2.5
RESULTS

N **50K**
C **8K**
ART **10ms**

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 3
ADDING NODES

GOALS

- Find the best system topology

STEPS

- Prepare the system to run in more than one node

- Decide if nodes should be connected or independent

- Decide if nodes should be on the same machine or not

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 3
ADDING NODES

GOALS

- Find the best system topology

STEPS

- Prepare the system to run in more than one node
- Decide if nodes should be connected or independent
- Decide if nodes should be on the same machine or not

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 3
ADDING NODES

TODO: Draw this!
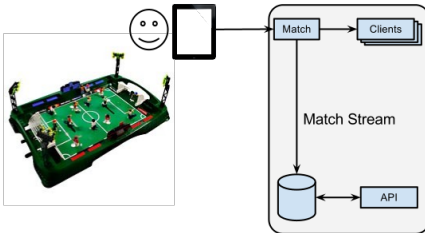
DB_READER gen_server. One per node. Processes db read operations

DB_WRITER gen_server. One per system. Processes db write operations

Introduction
Match Stream
Scaling
Final Words

Stage 0: Baseline
Stage 1: OS Tune
Stage 2: Tune the Code
Stage 3: Multi-Node Tuning

STAGE 3
RESULTS

With four nodes in
the same machine:

N **100K**

C **32K**

ART **10ms**

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

SUMMARY

With started with:
- N **1K**
- C **4**
- ART **26s**

We scale up to:
- N **100K**
- C **32K**
- ART **10ms**

Our improvements:
- N **100x**
- C **8000x**
- ART **2600x**

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

SUMMARY

With started with:
- N **1K**
- C **4**
- ART **26s**

We scale up to:
- N **100K**
- C **32K**
- ART **10ms**

Our improvements:
- N **100x**
- C **8000x**
- ART **2600x**

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

SUMMARY

With started with:
N **1K**
C **4**
ART **26s**

We scale up to:
N **100K**
C **32K**
ART **10ms**

Our improvements:
N **100x**
C **8000x**
ART **2600x**

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

SUMMARY

- This is an iterative process

- It proved itself useful in both experimental and real-life systems

- It gets improved with every system we scale

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

SUMMARY

- This is an iterative process
- It proved itself useful in both experimental and real-life systems
- It gets improved with every system we scale

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

SUMMARY

- This is an iterative process
- It proved itself useful in both experimental and real-life systems
- It gets improved with every system we scale

Introduction
Match Stream
Scaling
Final Words

Summary
Questions

QUESTIONS

Thanks!