

SCALING ERLANG WEB APPLICATIONS

100 TO 100K USERS AT ONE WEB SERVER

Fernando Benavides (*@elbrujohalcon*)

Inaka Labs

March 23, 2012



HELLO WORLD!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008



HELLO WORLD!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008



HELLO WORLD!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale



HELLO WORLD!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale



HELLO WORLD!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale



HELLO WORLD!

- I've built several dynamic web servers
 - Many of them with real-time updates
 - Most of them with high scale requirements
- I'll show you how I make them scale



INTRODUCTION

We will work 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.

Examples:

• Social media

• Chat apps

• Sports sites



INTRODUCTION

We will work 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.

Examples:

- *Websockets*
- *Long Polling*
- *Server-Sent Events*



INTRODUCTION

We will work 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.

Examples:

- Social sites

- Chatting

- Sports sites



INTRODUCTION

We will work 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.

Examples:

- Social sites
- Chat sites
- Gaming sites



INTRODUCTION

We will work 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.

Examples:

- Social sites
- Chat sites
- Sports sites



INTRODUCTION

We will work 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.

Examples:

- Social sites
- Chat sites
- Sports sites



INTRODUCTION

We will work 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.

Examples:

- Social sites
- Chat sites
- Sports sites



SCOPE

We will try to improve the way we use

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

*We will **not** deal with*

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



SCOPE

We will try to improve the way we use

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

*We will **not** deal with*

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



MATCH STREAM

GENERAL IDEA

A soccer match is played at some stadium



MATCH STREAM

GENERAL IDEA

Soccer fans are connected to the internet in their offices



MATCH STREAM

GENERAL IDEA

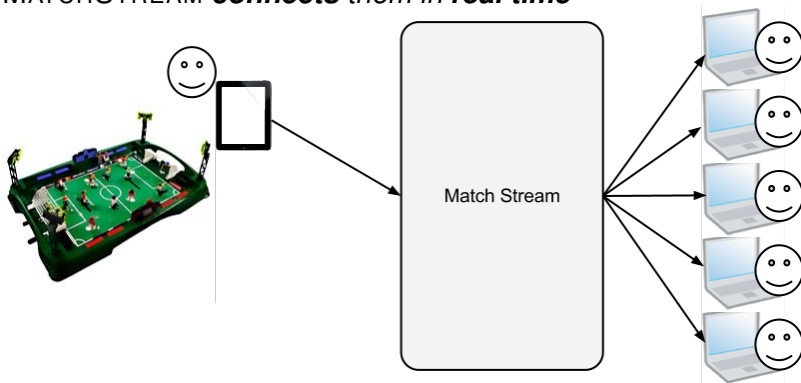
A reporter is at the stadium with his device



MATCH STREAM

GENERAL IDEA

MATCHSTREAM ***connects*** them in ***real time***



MATCH STREAM

REQUIREMENTS

SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

Erlang seems to be **the right fit for this**



MATCH STREAM

REQUIREMENTS

SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

Erlang seems to be **the right fit for this**



MATCH STREAM

REQUIREMENTS

SYSTEM CHALLENGES

- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

Erlang seems to be **the right fit for this**



MATCH STREAM

REQUIREMENTS

SYSTEM CHALLENGES

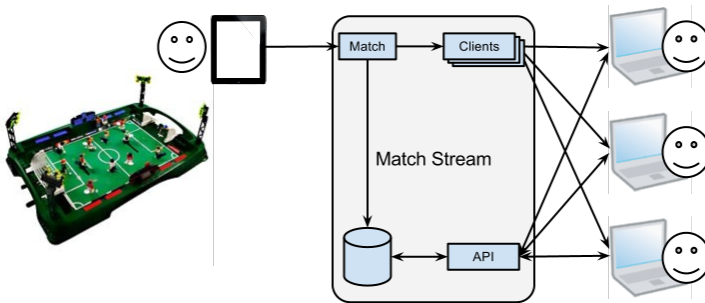
- Tons of concurrent users
- Two-hour-long bursts of connections followed by long periods of inactivity
- Real-time updates

Erlang seems to be **the right fit for this**



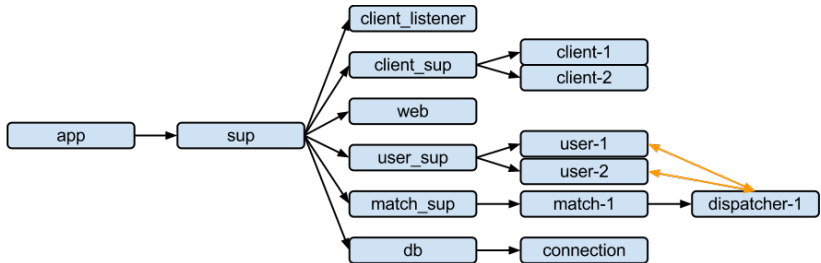
MATCH STREAM

GENERAL DESIGN



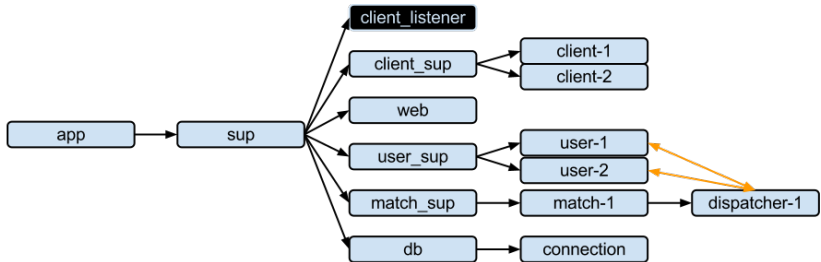
MATCH STREAM

ARCHITECTURE



MATCH STREAM

ARCHITECTURE

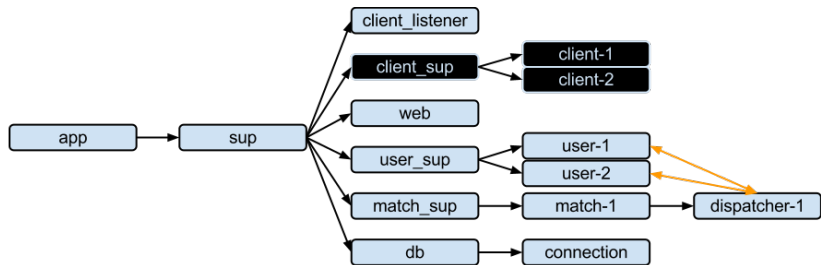


CLIENT_LISTENER `gen_server`. Listens on a TCP port to receive client connections



MATCH STREAM

ARCHITECTURE



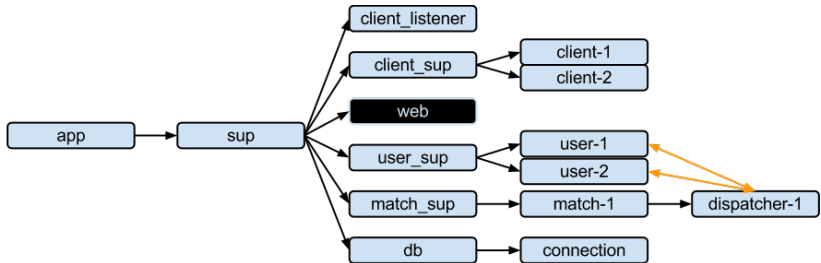
CLIENT_SUP supervisor. Supervises connection processes

CLIENT gen_fsm. Handles a TCP connection



MATCH STREAM

ARCHITECTURE

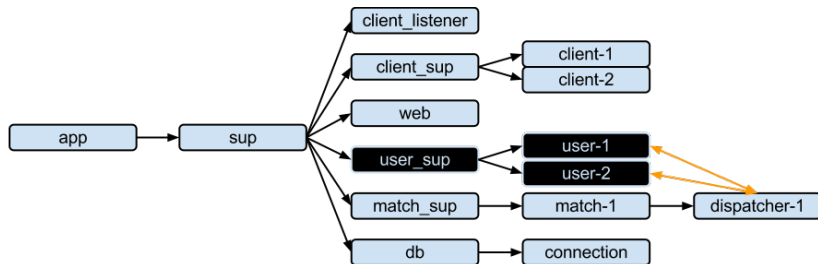


WEB mochiweb server. Listens for HTTP API calls



MATCH STREAM

ARCHITECTURE



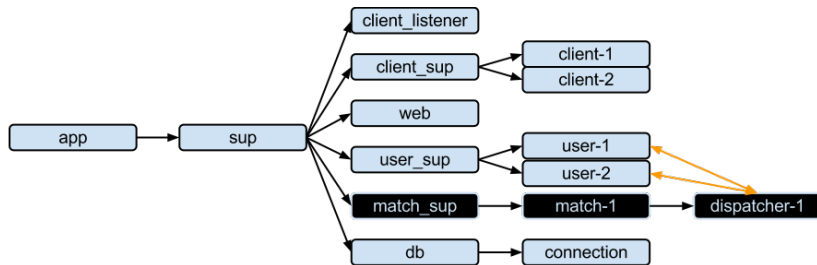
USER_SUP supervisor. Supervises user processes

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



MATCH STREAM

ARCHITECTURE



MATCH_SUP supervisor. Supervises match processes

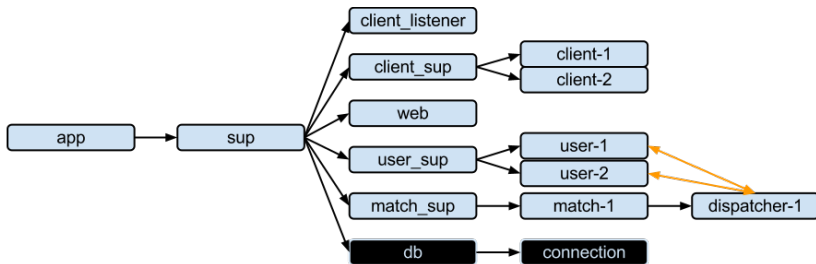
MATCH gen_server. Listens to match events, stores them

DISPATCHER gen_event dispatcher. Delivers match events



MATCH STREAM

ARCHITECTURE



DB `gen_server`. Processes database operations

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



LESSON LEARNED

*Using Erlang to build your system
is **not enough** to ensure
scalability*



STAGE 1

TESTING THE SYSTEM AS IT IS

GOALS

- Find how much the system can handle

STEPS

- Create automated testers
- Start the system on a *clean* machine
- Test repeatedly adjusting the number of connections
- Have a human trying the system himself



STAGE 1

TESTING THE SYSTEM AS IT IS

GOALS

- Find how much the system can handle

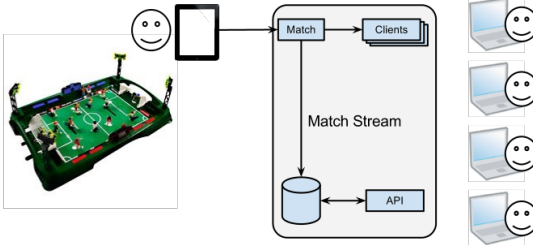
STEPS

- Create automated testers
- Start the system on a *clean* machine
- Test repeatedly adjusting the number of connections
- Have a human trying the system himself



STAGE 1

RESULTS



- **1024** users
- **4** at a time
- **10s** ART



STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

- Improve the system environment without altering the code

SETTINGS TO TUNE UP

- Concurrent TCP connections
- Open files limit
- Shared library cache
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

- Improve the system environment without altering the code

SETTINGS TO TUNE UP

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

- Improve the system environment without altering the code

SETTINGS TO TUNE UP

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

- Improve the system environment without altering the code

SETTINGS TO TUNE UP

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

- Improve the system environment without altering the code

SETTINGS TO TUNE UP

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

IMPROVING THE ENVIRONMENT

GOALS

- Improve the system environment without altering the code

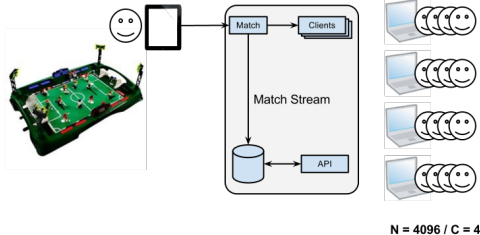
SETTINGS TO TUNE UP

- Concurrent TCP connections
- Open files limit
- TCP backlog size
- TCP memory allocation
- Erlang VM startup parameters



STAGE 2

RESULTS



- **4096** users
- **4** at a time
- **9s** ART



STAGE 3

IMPROVING MATCH STREAM

GOALS

- Tune up the system for **one node**

STEPS

- Find a problem
- Fix it using the list of *Tips and Tricks*
- If not there, add it
- Repeat from **Stage 1**



STAGE 3

IMPROVING MATCH STREAM

GOALS

- Tune up the system for **one node**

STEPS

- Find a problem
- Fix it using the list of *Tips and Tricks*
- If not there, add it
- Repeat from **Stage 1**



STAGE 3.1

CONNECTION TWEAKS

BACKLOG

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

CONNECTIONS

- Don't use just one of them
- Check inbound and outbound connections



STAGE 3.1

CONNECTION TWEAKS

BACKLOG

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

CONNECTIONS

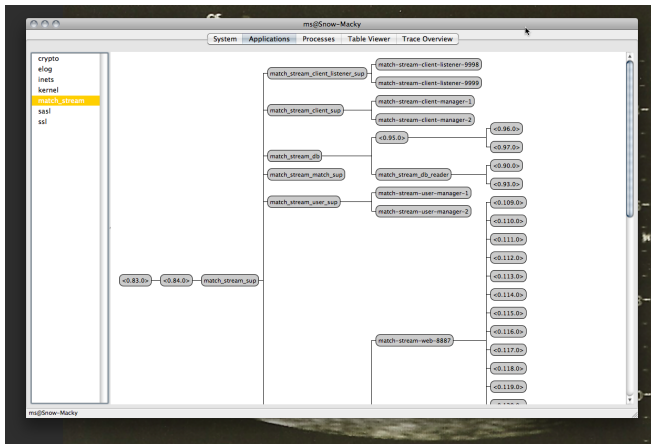
- Don't use just one of them
- Check inbound and outbound connections



STAGE 3.1

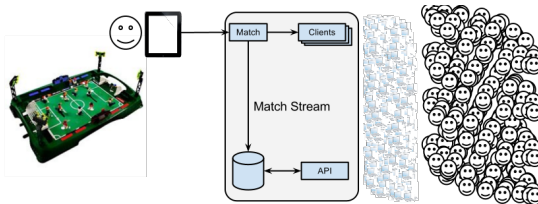
CONNECTION TWEAKS

SYSTEM ARCHITECTURE



STAGE 3.1

RESULTS



N = 65536 / C = 8192

- **8192** users
- **256** at a time
- **16s** ART



STAGE 3.2

GEN_EVENT

SUP_HANDLER

- Don't use it
- Monitor the processes instead

LONG DELIVERY QUEUES

- Use *repeaters*



STAGE 3.2

GEN_EVENT

SUP_HANDLER

- Don't use it
- Monitor the processes instead

LONG DELIVERY QUEUES

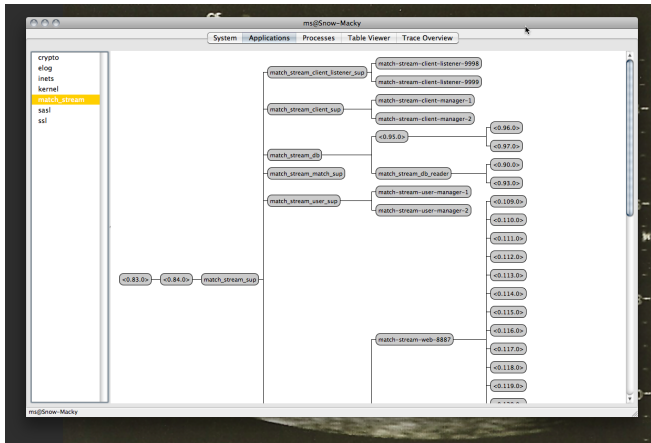
- Use *repeaters*



STAGE 3.2

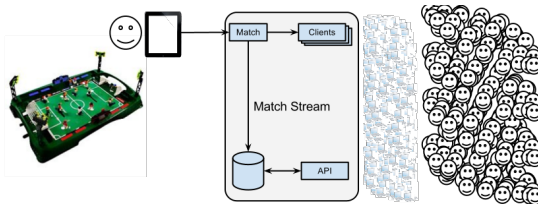
GEN_EVENT

SYSTEM ARCHITECTURE



STAGE 3.2

RESULTS



N = 65536 / C = 8192

- **8192** users
- **256** at a time
- **8s** ART



STAGE 3.3

GEN_SERVER

CALL TIMEOUTS

Remember `gen_server:reply/2`

MEMORY FOOTPRINT

Remember `hibernate`

LONG INIT/1

Use 0 timeout



STAGE 3.3

GEN_SERVER

CALL TIMEOUTS

Remember `gen_server:reply/2`

MEMORY FOOTPRINT

Remember `hibernate`

LONG INIT/1

Use 0 timeout



STAGE 3.3

GEN_SERVER

CALL TIMEOUTS

Remember `gen_server:reply/2`

MEMORY FOOTPRINT

Remember `hibernate`

LONG INIT/1

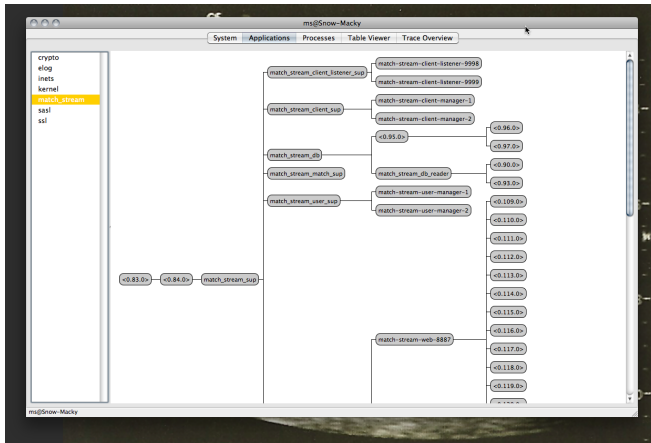
Use 0 timeout



STAGE 3.3

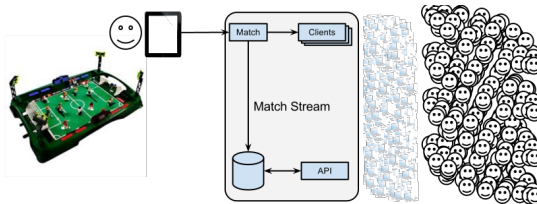
GEN_SERVER

SYSTEM ARCHITECTURE



STAGE 3.3

RESULTS



N = 65536 / C = 8192

- **32768** users
- **1024** at a time
- **1s** ART



STAGE 3.4

SUPERVISORS

- Sometimes `simple_one_for_one` supervisors get **overburdened** because they have too many children
- Try a supervisor hierarchy with several managers below the main supervisor
- Turn `supervisor:start_child/2` calls into something like

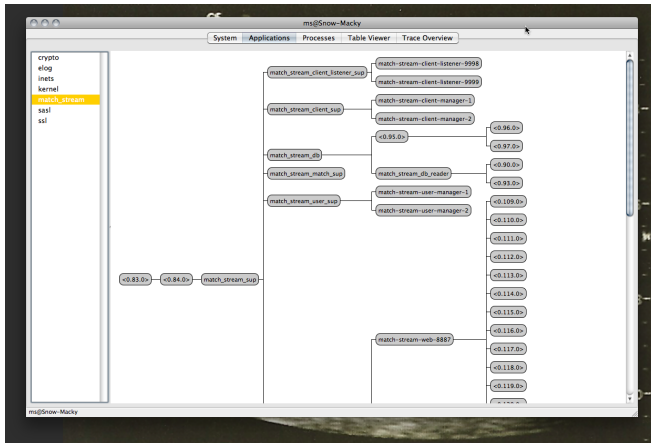
```
supervisor:start_child(  
    list_to_atom("module-name_" ++  
                integer_to_list(random:uniform(#ofSupervisors)))
```



STAGE 3.4

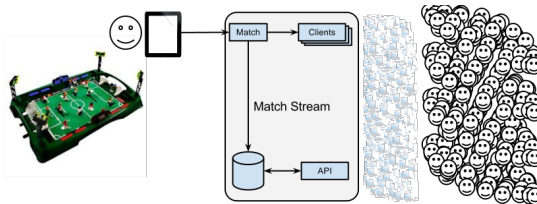
SUPERVISORS

SYSTEM ARCHITECTURE



STAGE 3.4

RESULTS



$N = 65536 / C = 8192$

- **65536** users
- **2048** at a time
- **1s** ART



STAGE 3.5

OTHER PROCESSES

TIMERS

- Don't use the `timer` module
- Use `erlang:send_after`

LOGGING

- Don't log too much
- Use a good logging system

REGISTRATION

- Sometimes it's better to register processes instead of keeping track of their pids manually
- You can always register processes **both** locally and globally



STAGE 3.5

OTHER PROCESSES

TIMERS

- Don't use the `timer` module
- Use `erlang:send_after`

LOGGING

- Don't log too much
- Use a good logging system

REGISTRATION

- Sometimes it's better to register processes instead of keeping track of their pids manually
- You can always register processes **both** locally and globally



STAGE 3.5

OTHER PROCESSES

TIMERS

- Don't use the `timer` module
- Use `erlang:send_after`

LOGGING

- Don't log too much
- Use a good logging system

REGISTRATION

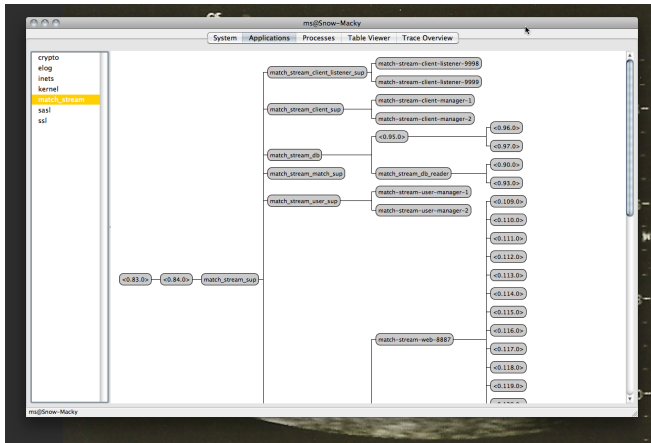
- Sometimes it's better to register processes instead of keeping track of their pids manually
- You can always register processes **both** locally and globally



STAGE 3.5

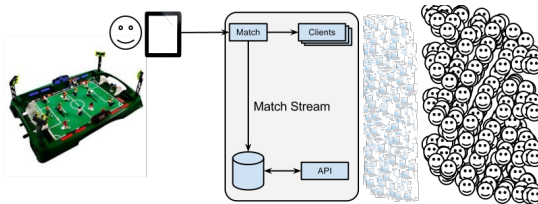
OTHER PROCESSES

SYSTEM ARCHITECTURE



STAGE 3.5

RESULTS



$N = 65536 / C = 8192$

- **65536** users
- **8192** at a time
- **10ms** ART



STAGE 4

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



STAGE 4

ADDING NODES

GOALS

- Find the best system topology

STEPS

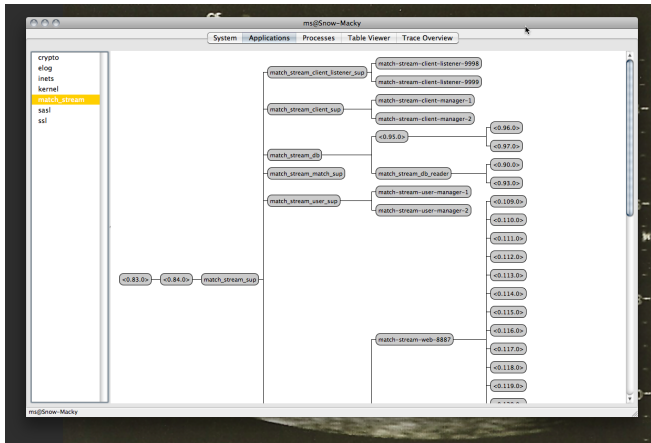
- Prepare the system to run in more than one node
- Decide if nodes should be connected or independent
- Decide if nodes should be on the same machine or not



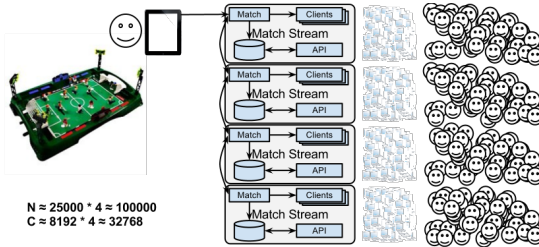
STAGE 4

ADDING NODES

SYSTEM ARCHITECTURE



STAGE 4 RESULTS



- **100K** users
- **32768** at a time
- **10ms** ART



SUMMARY

- This is an **iterative** process
- It worked awesomely for us in both experimental and real-life systems
- It's no **silver bullet**
- The list of *Tips and Tricks* grows **constantly** over time



SUMMARY

- This is an **iterative** process
- It worked awesomely for us in both experimental and real-life systems
- It's no **silver bullet**
- The list of *Tips and Tricks* grows **constantly** over time



SUMMARY

- This is an **iterative** process
- It worked awesomely for us in both experimental and real-life systems
- It's no **silver bullet**
- The list of *Tips and Tricks* grows **constantly** over time



SUMMARY

- This is an **iterative** process
- It worked awesomely for us in both experimental and real-life systems
- It's no **silver bullet**
- The list of *Tips and Tricks* grows **constantly** over time



SCALING TOPICS

THAT WEREN'T COVERED ON THIS PRESENTATION

- Managing many nodes
- Choosing databases
- System specific improvements
- Measuring tools



QUESTIONS



Thanks!

