Scaling Erlang Web Applications

100 to 100K users at one web server

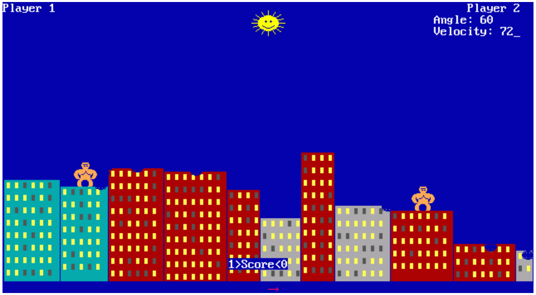
Fernando Benavides (*@elbrujohalcon*)

Inaka Labs

March 20, 2012

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you



Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

1. elbrujohalcon@Snow-Macky: ~/Projects/erlang/inaka/edis (gnumake)
elbrujohalcon@Sno... ⌘1
elbrujohalcon@Sno... ⌘2

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Hello World!

- I'm a developer since I was 10
- I'm an Erlang developer since 2008
- I've worked in many dynamic web sites
- Most of them with high scale requirements
- I'll share my experience with you

Hello World!

- I’m a developer since I was 10
- I’m an Erlang developer since 2008
- I’ve worked in many dynamic web sites
- Most of them with high scale requirements
- I’ll share my experience with you

Outline

Outline

Outline

Outline

Outline

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

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

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
- Underlaying 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
- Underlaying system configurations

*We will **not** deal with*

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

Match Stream

TODO: General system design graph

Requirements

This kind of system requires. . .

- Tons of users
- In two-hour-long bursts
- Real time updates

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

Requirements

This kind of system requires. . .

- Tons of users
- In two-hour-long bursts
- Real-time updates

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

Requirements

This kind of system requires. . .

- Tons of users
- In two-hour-long bursts
- Real-time updates

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

Requirements

This kind of system requires. . .

- Tons of users
- In two-hour-long bursts
- Real-time updates

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

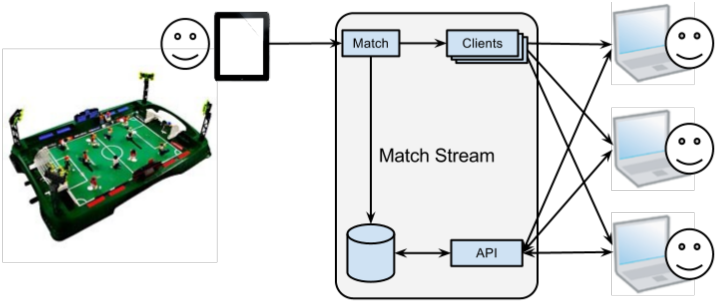
Requirements

This kind of system requires...

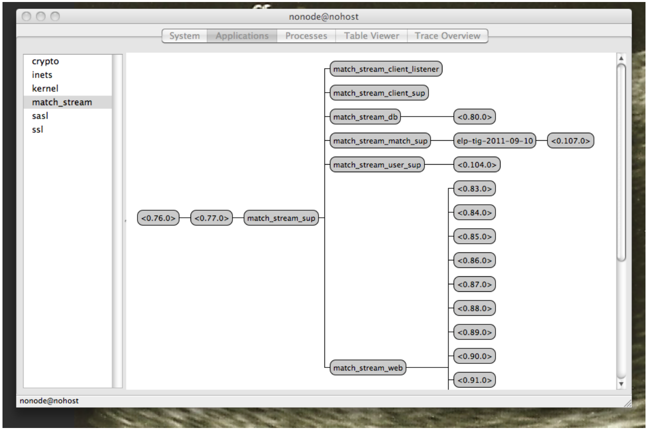
- Tons of users
- In two-hour-long bursts
- Real-time updates

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

Match Stream
General Design



Match Stream
Architecture



For the Clients

CLIENT_LISTENER
: `gen_server`. Handles a listening TCP port to receive clients

CLIENT_SUP
: `supervisor`. Supervises client processes

USER_SUP
: `supervisor`. Supervises user processes

WEB
: `mochiweb`. Listener for HTTP API calls

For the Clients

CLIENT_LISTENER
: `gen_server`. Handles a listening TCP port to receive clients

CLIENT_SUP
: `supervisor`. Supervises client processes

USER_SUP
: `supervisor`. Supervises user processes

WEB
: `mochiweb`. Listener for HTTP API calls

For the Clients

CLIENT_LISTENER
: `gen_server`. Handles a listening TCP port to receive clients

CLIENT_SUP
: `supervisor`. Supervises client processes

USER_SUP
: `supervisor`. Supervises user processes

WEB
: `mochiweb`. Listener for HTTP API calls

For the Clients

CLIENT_LISTENER
: `gen_server`. Handles a listening TCP port to receive clients

CLIENT_SUP
: `supervisor`. Supervises client processes

USER_SUP
: `supervisor`. Supervises user processes

WEB
: `mochiweb`. Listener for HTTP API calls

Lesson Learned

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

General Rules

Things you need to know when scaling your systems

- Start with a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Rules

Things you need to know when scaling your systems

- Start with a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Rules
Things you need to know when scaling your systems

- Start with a system that works
- Automate your clients
- Keep a human watching
- Be patient

General Rules

Things you need to know when scaling your systems

- Start with a system that works
- Automate your clients
- Keep a human watching
- Be patient

Stage 1
Finding The Initial Boundaries

Goals

- Test the system as it is
- Find N and C

Steps

- Create the automated testers
- Choose N and C
- Test the API and long-lived connections independently
- Test both together
- Repeat until you find the highest N and C

Stage 1
Finding The Initial Boundaries

Goals

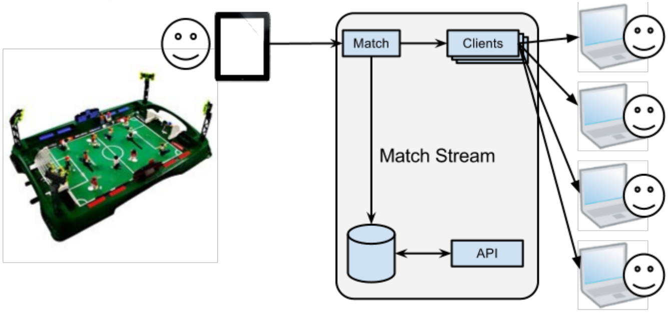
- Test the system as it is
- Find *N* and *C*

Steps

- Create the automated testers
- Choose *N* and *C*
- Test the API and long-lived connections independently
- Test both together
- Repeat until you find the highest *N* and *C*

Stage 1
Finding The Initial Boundaries

Results



N = 1024 / C = 4

Stage 2
Blackbox Tests

Goals

- Improve the system environment
- Find the highest *N* and *C* without altering the code

Steps

- Check kernel variables
- Check system limits
- Check Erlang VM parameters
- Repeat from Stage 1

STAGE 2
BLACKBOX TESTS

GOALS

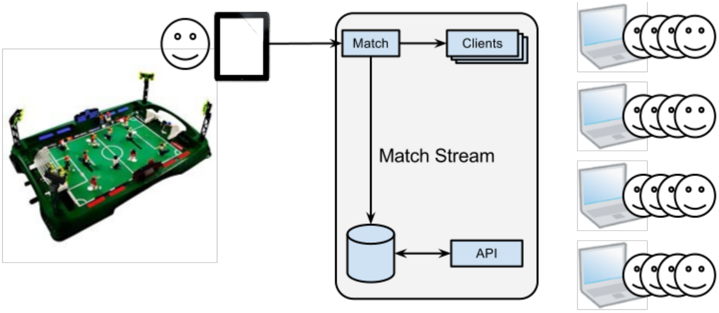
- Improve the system environment
- Find the highest *N* and *C* without altering the code

STEPS

- Check kernel variables
- Check system limits
- Check Erlang VM parameters
- Repeat from Stage 1

Stage 2
Blackbox Tests

Results



N = 4096 / C = 4

STAGE 3
ERLANG TUNING

GOALS

- Tune up your system
- Find the highest *N* and *C* 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
ERLANG TUNING

GOALS

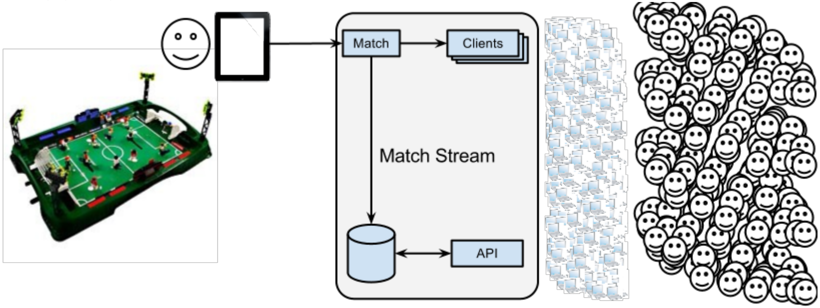
- Tune up your system
- Find the highest *N* and *C* for one node

STEPS

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

Stage 3
Erlang Tuning

Results



N = 65536 / C = 8192

STAGE 4
ADDING NODES

GOALS

- Find the best system topology
- Find *N* and *C* per node

STEPS

- Add a node
    - connected; or
    - independent
- Repeat from Stage 1

Stage 4
Adding Nodes

Goals

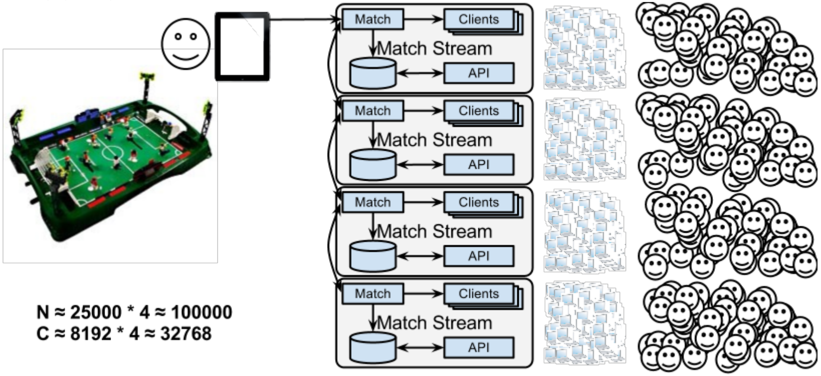
- Find the best system topology
- Find *N* and *C* per node

Steps

- Add a node
    - connected; or
    - independent
- Repeat from Stage 1

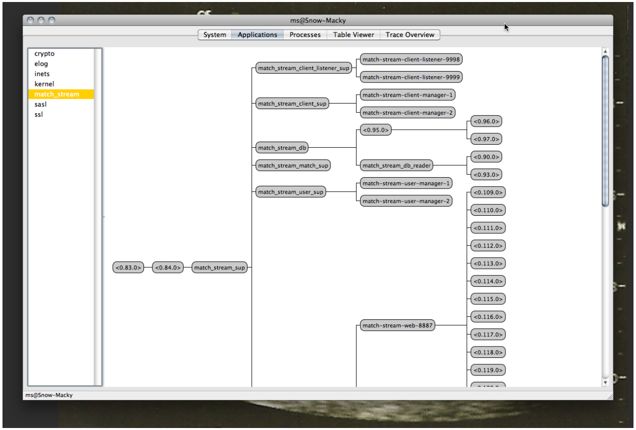
Stage 4
Adding Nodes

Results



$N \approx 25000 * 4 \approx 100000$
$C \approx 8192 * 4 \approx 32768$

Match Stream
Final Architecture



Tips and Tricks

OS TWEAKS

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```

Open Files Limit

```
ulimit -n 999999
```

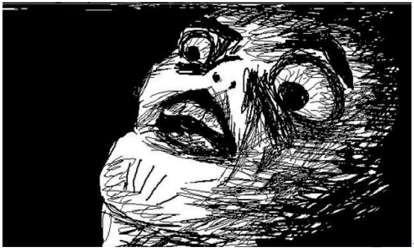
Erlang VM tweaks

+P	Number of Processes
+K	Kernell Polling
-SMP	SMP Support

OS TWEAKS

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```



OS TWEAKS

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```

Open Files Limit

```
ulimit -n 999999
```

Erlang VM tweaks

+P	Number of Processes
+K	Kernell Polling
-SMP	SMP Support

OS tweaks

Kernel Variables

```
sysctl -w net.ipv4.ip_local_port_range="1024 65535"
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_rmem="4096 87380 16777216"
sysctl -w net.ipv4.tcp_wmem="4096 65536 16777216"
sysctl -w net.ipv4.tcp_syncookies=1
sysctl -w net.ipv4.tcp_mem="50576   64768   98152"
sysctl -w net.core.netdev_max_backlog=2500
sysctl -w net.netfilter.nf_conntrack_max=1233000
```

Open Files Limit

```
ulimit -n 999999
```

Erlang VM tweaks

+P	Number of Processes
+K	Kernell Polling
-SMP	SMP Support

Connection tweaks

Backlog

- Allow more concurrent connections
- Remember HTTP *runs on* TCP

Connections

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

Connection tweaks

Backlog

- Allow more concurrent connections
- Remember HTTP *runs on* TCP

Connections

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

GEN_EVENT

SUP_HANDLER

- Don’t use it
- Monitor the processes instead

Long Delivery Queues

- Use *repeaters*

GEN_EVENT

SUP_HANDLER

- Don't use it
- Monitor the processes instead

Long Delivery Queues

- Use *repeaters*

GEN_SERVER

Call Timeouts
Remember `gen_server:reply/2`

Memory Footprint
Remember `hibernate`

Long init/1
Use 0 timeout

GEN_SERVER

Call Timeouts
Remember `gen_server:reply/2`

Memory Footprint
Remember `hibernate`

Long `init/1`
Use 0 timeout

GEN_SERVER

Call Timeouts
Remember `gen_server:reply/2`

Memory Footprint
Remember `hibernate`

Long `init/1`
Use 0 timeout

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))).
```

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

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

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

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

- Adding nodes
- Choosing databases
- System specific improvements
- Measuring tools

Questions

Thanks!