FEUERLABS

Erlang

A beacon language for concurrency programming

by Ulf Wiger, Co-Founder, Feuerlabs

Erlang/Linux Timeline

1956

CSLab

VAX/Unix

Erlang

Linux

RHL

Products w/ Erlang&Unix

Open Source Erlang

Ericsson

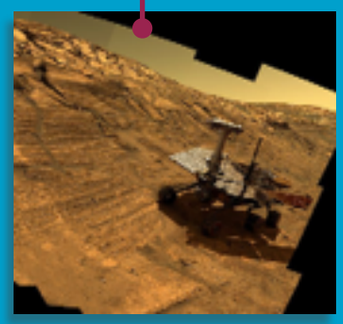
Linux at ///

1980 1985 1990 1995 2000 2005

Linux

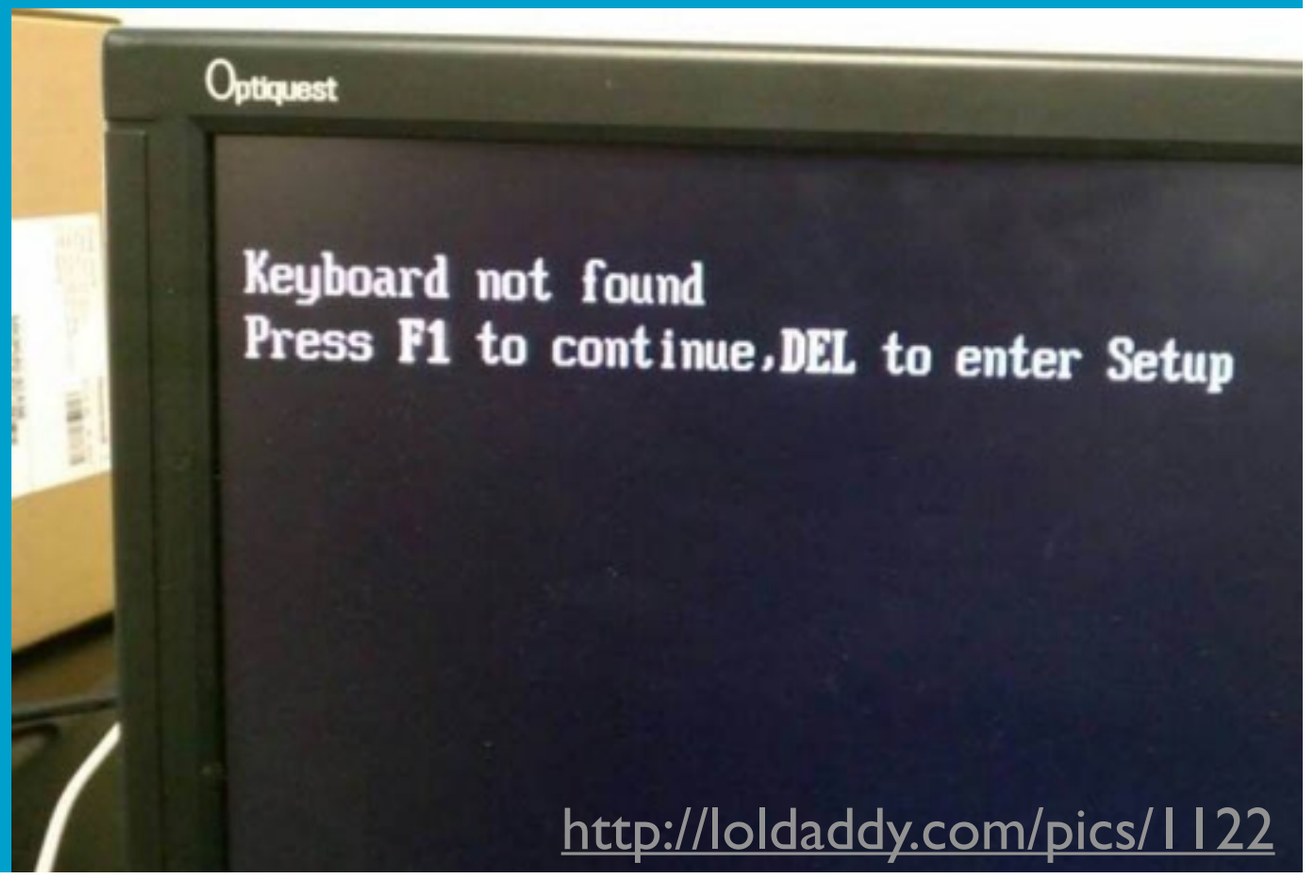
1969

Linux user base triples in 1998



Mid-90s—Early Days

- PC Hardware and UNIX not yet in embedded systems
- Many off-the-shelf boards had the old BIOS bug
- We were told UNIX unsuitable for 99.999% systems
- And Erlang was weird and slow!

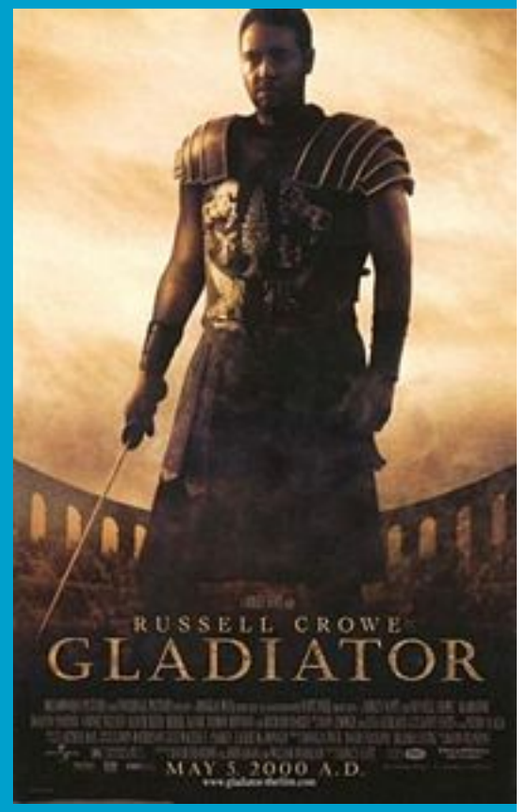


Now—Portability & Multicore

- CPU Architecture War revived
- Parallelism made explicit!
  - Saves power in devices
  - Gives speed on servers
- Erlang thrives on multicore

Great Names, Immortal Names

- **“You have a great name. He must kill your name before he kills you.”** (Juba, The Gladiator)



- “Robert Metcalf [the inventor of Ethernet] says that **if something comes along to replace Ethernet, it will be called “Ethernet”, so therefore Ethernet will never die.** Unix has already undergone several such transformations. (Ken Thompson, from “The Durability of Unix”)
- **Unix is immortal**

http://upload.wikimedia.org/wikipedia/ca/0/09/Gladiator_ver2.jpg

Erlang—Favoured by the Gods?

- “Erlang is going to be a very important language.
  It could be the next Java” (Ralph Johnson)
- “Erlang is a beacon language for concurrency programming”
  (Simon Peyton Jones)
- “If Erlang doesn't become the next great language,
  it will at least receive honourable mention

  When the [multi-core] revolution comes, you will be better prepared if you know Erlang”
  (Kresten Krab Thorup, GOTO CPH Keynote 2011)

FEUERLABS

What Erlang Was Made For

Agent-based services...

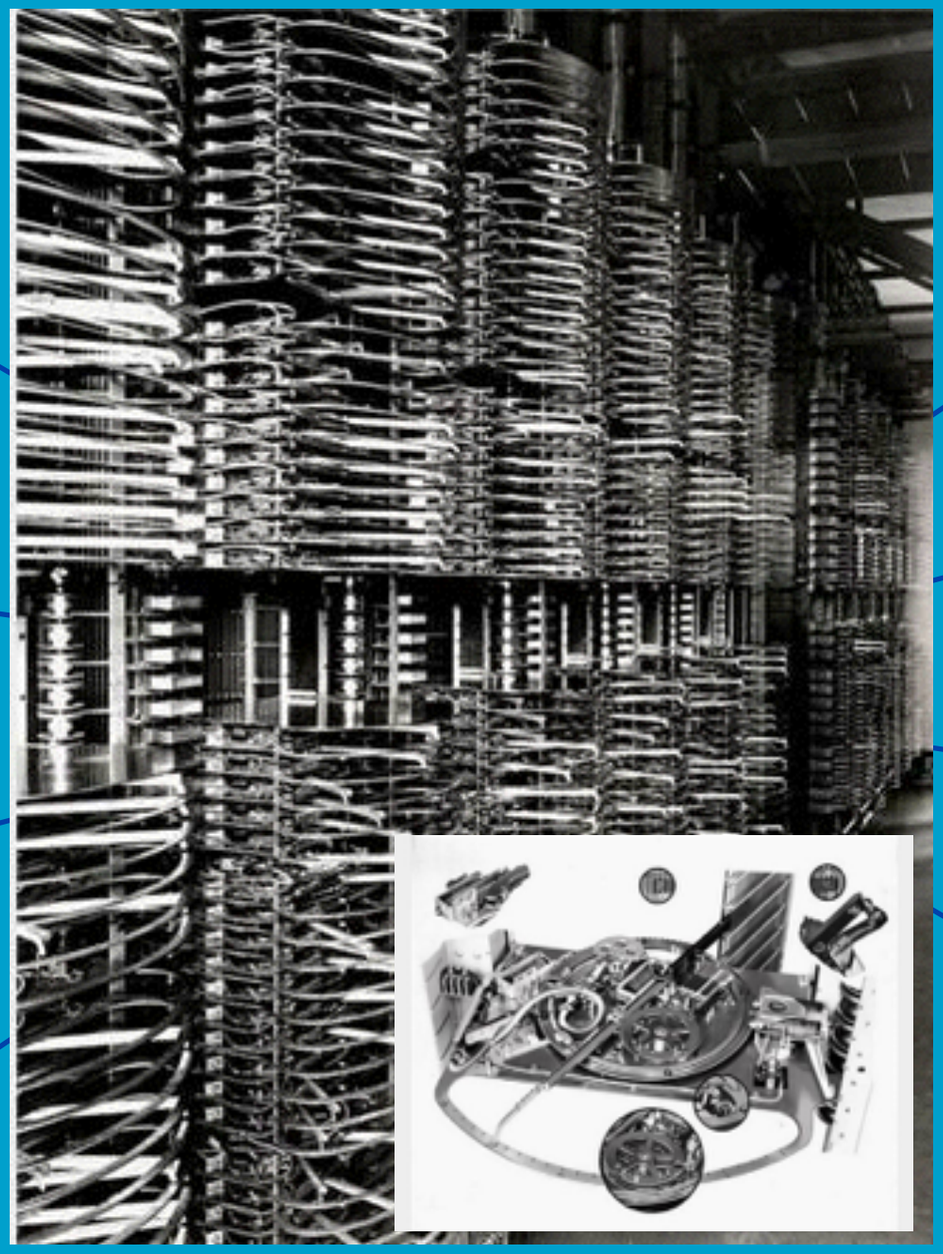
25-lines switchboard, Natal Province, South Africa 1897

Scalability through message passing...

SAT's main telephone exchange, Stockholm 1897 – 7000 lines
Human interaction required for cross-switchboard connections

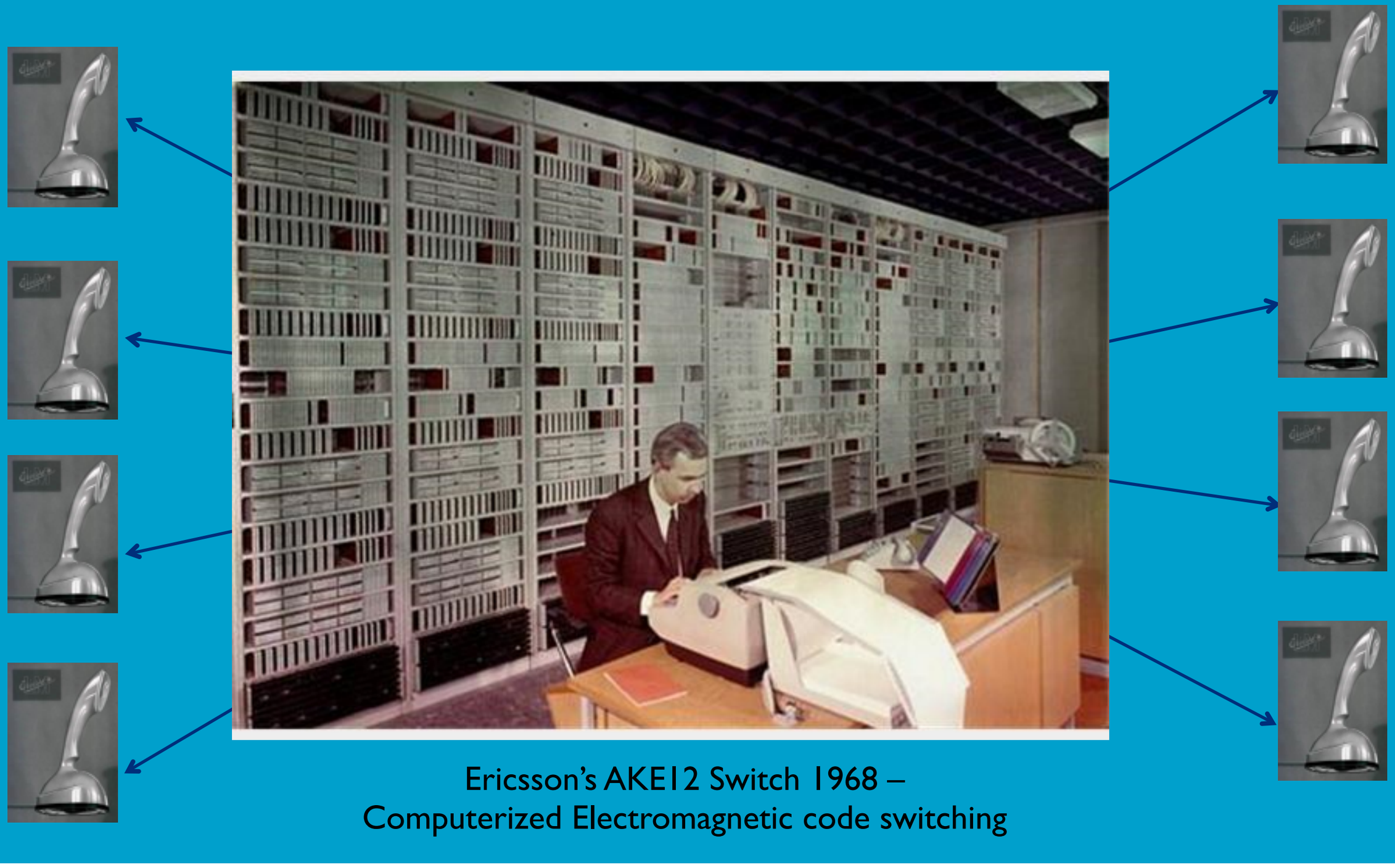
FEUERLABS

Automatic Switching – Machine-driven



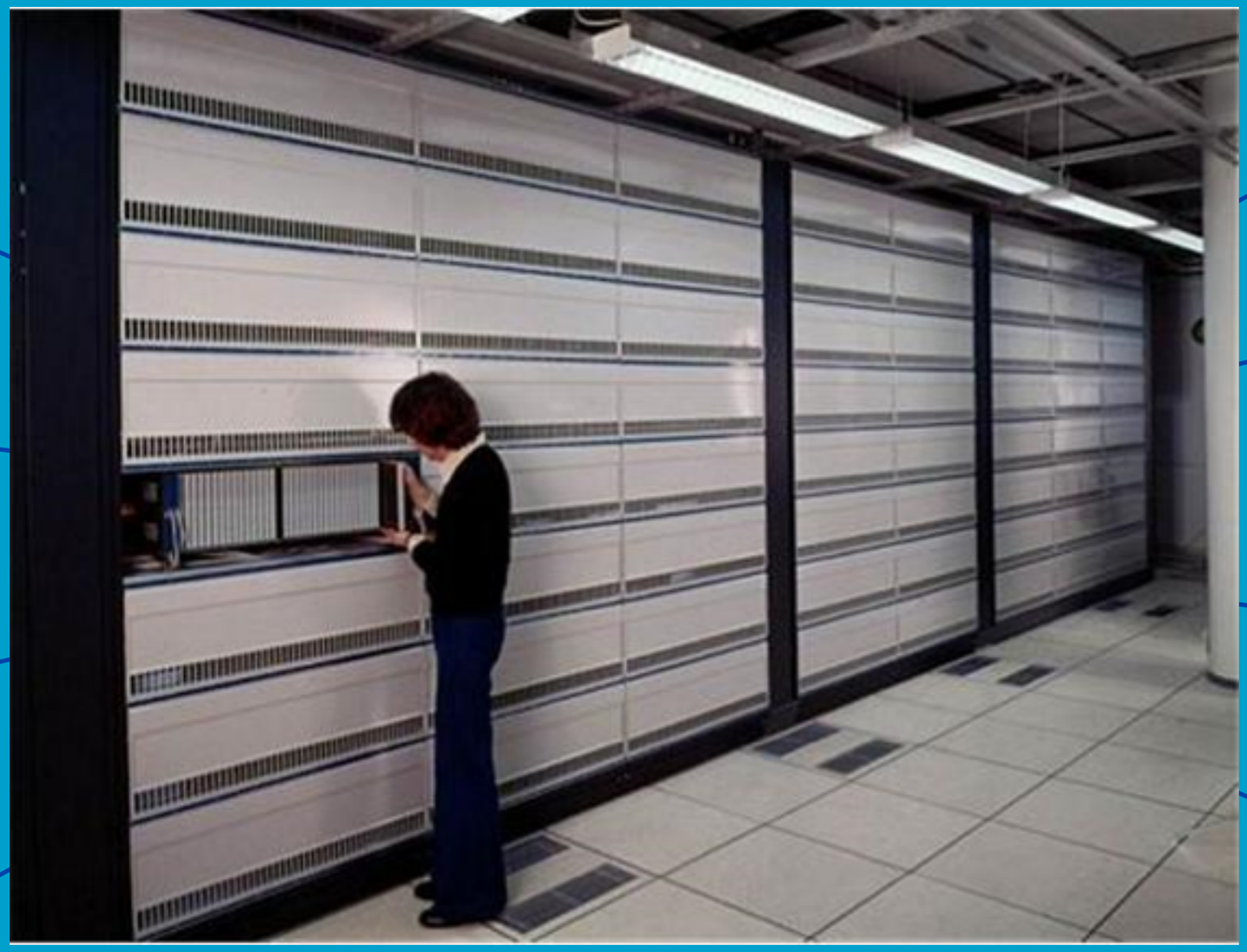
Ericsson's 500 Switch 1940s – 500 lines per stage

Stored Program Control – Bugs and all!!!



Ericsson's AKE12 Switch 1968 –
Computerized Electromagnetic code switching

Digital switching, modular SW design



Ericsson's AXE10 Switch 1975 – High-Level Language (PLEX)

Telephony-Realm Problems

- Soft Real-Time
- High Availability
  - In-service upgrades
  - Self-healing systems (fault-tolerance)
- Scalability
- Device Management
- Complexity

These aspects are reflected in the design of Erlang

FEUERLABS

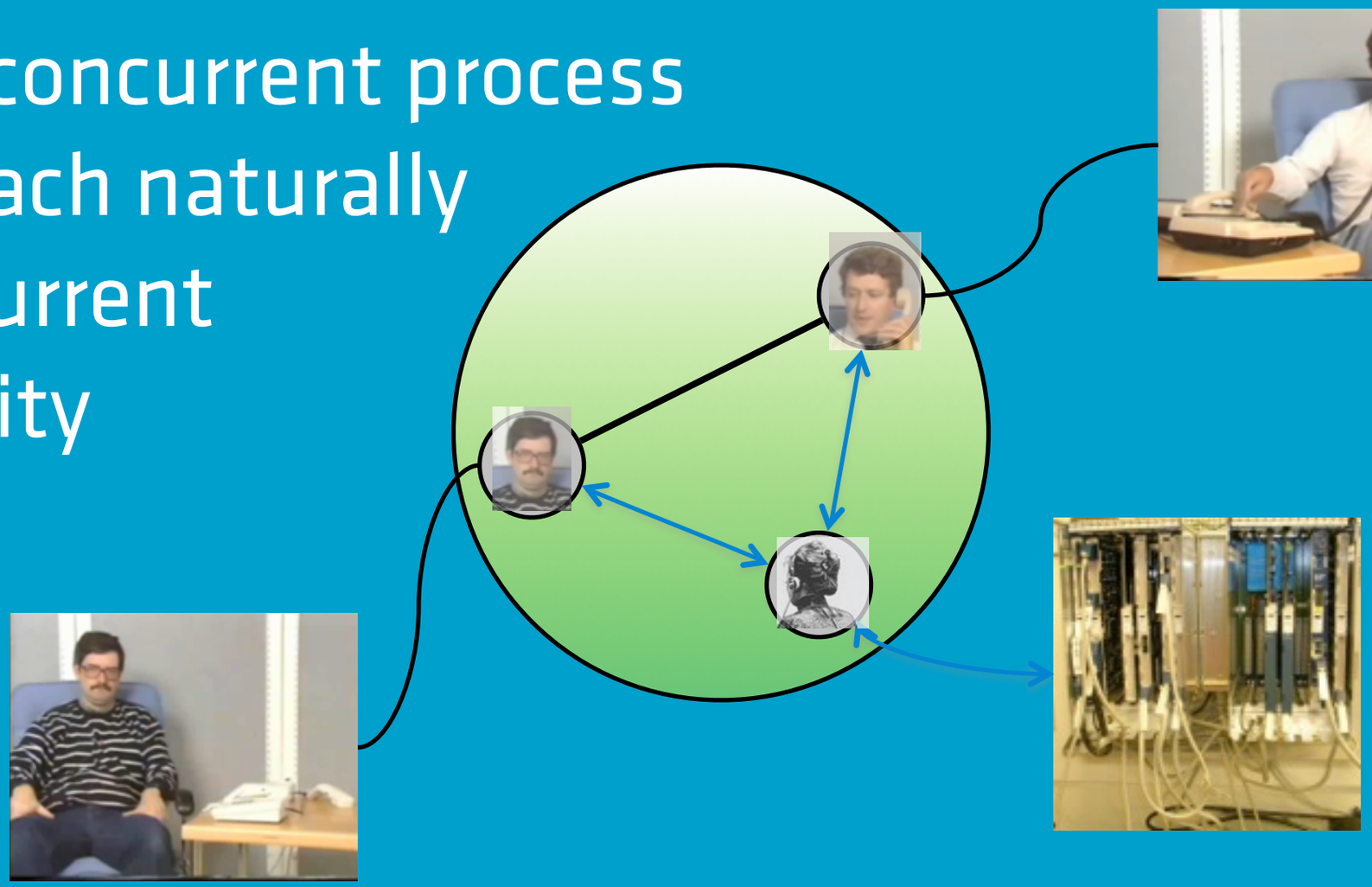
Erlang, Intuitively

http://www.youtube.com/watch?v=xrIjfIjssLE

FEUERLABS

Erlang, Intuitively

- One concurrent process for each naturally concurrent activity



FEUERLABS

Woah! Pthread Hell?!!

- Most people have a hard enough time understanding tasks, never mind "chopped up tasks" or threads.
- The first problem while programming is answering the question: "What can be threaded in my app?". That, in itself, can be very laborious (see section on "What kinds of things should be threaded/multitasked?").
- Another problem is locking. All the nightmares about sharing, locking, deadlock, race conditions, etc. come vividly alive in threads. Processes don't usually have to deal with this, since most shared data is passed through pipes. Now, threads can share file handles, pipes, variables, signals, etc.
- Trying to test and duplicate error conditions can cause more gray hair than a wayward child.

Concurrency Complexity, from Linux Threads FAQ

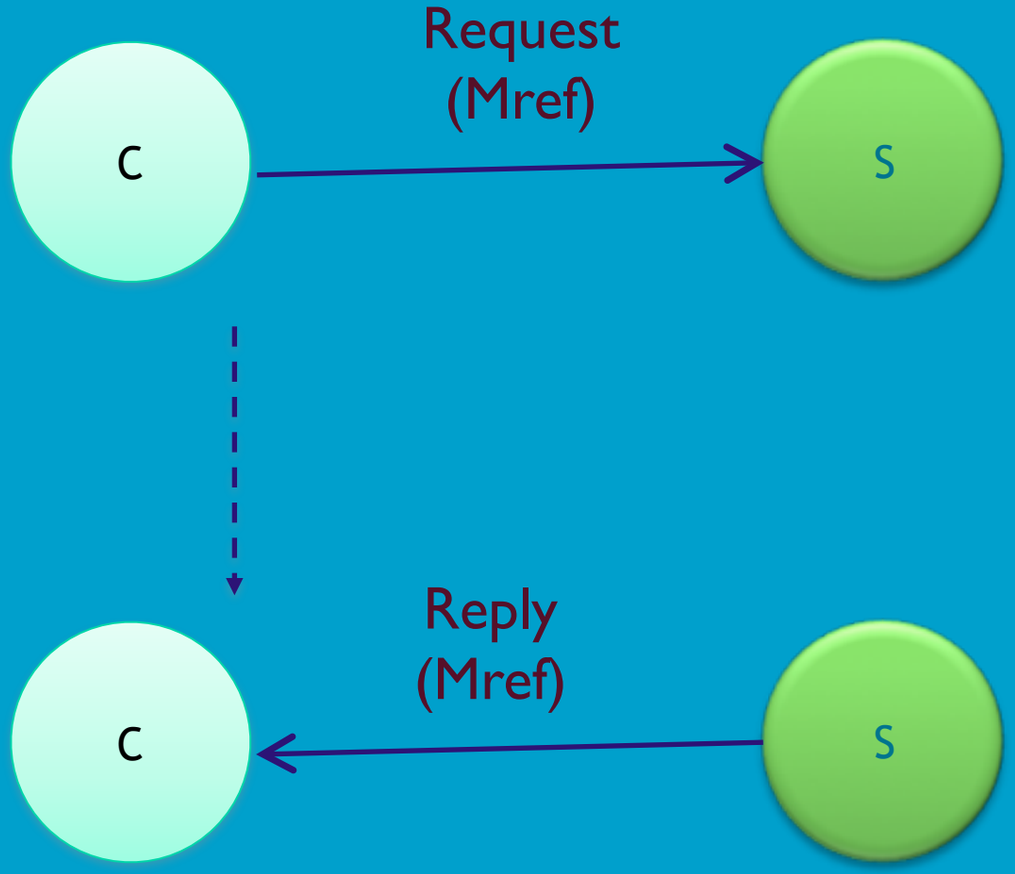
Erlang is Different

- Concurrency Done Right™
- No locking, no shared memory (mostly)
- Extremely lightweight processes
- Maps intuitively to the inherent concurrency of the problem space

FEUERLABS

Client-server in Erlang

1 Client monitors server



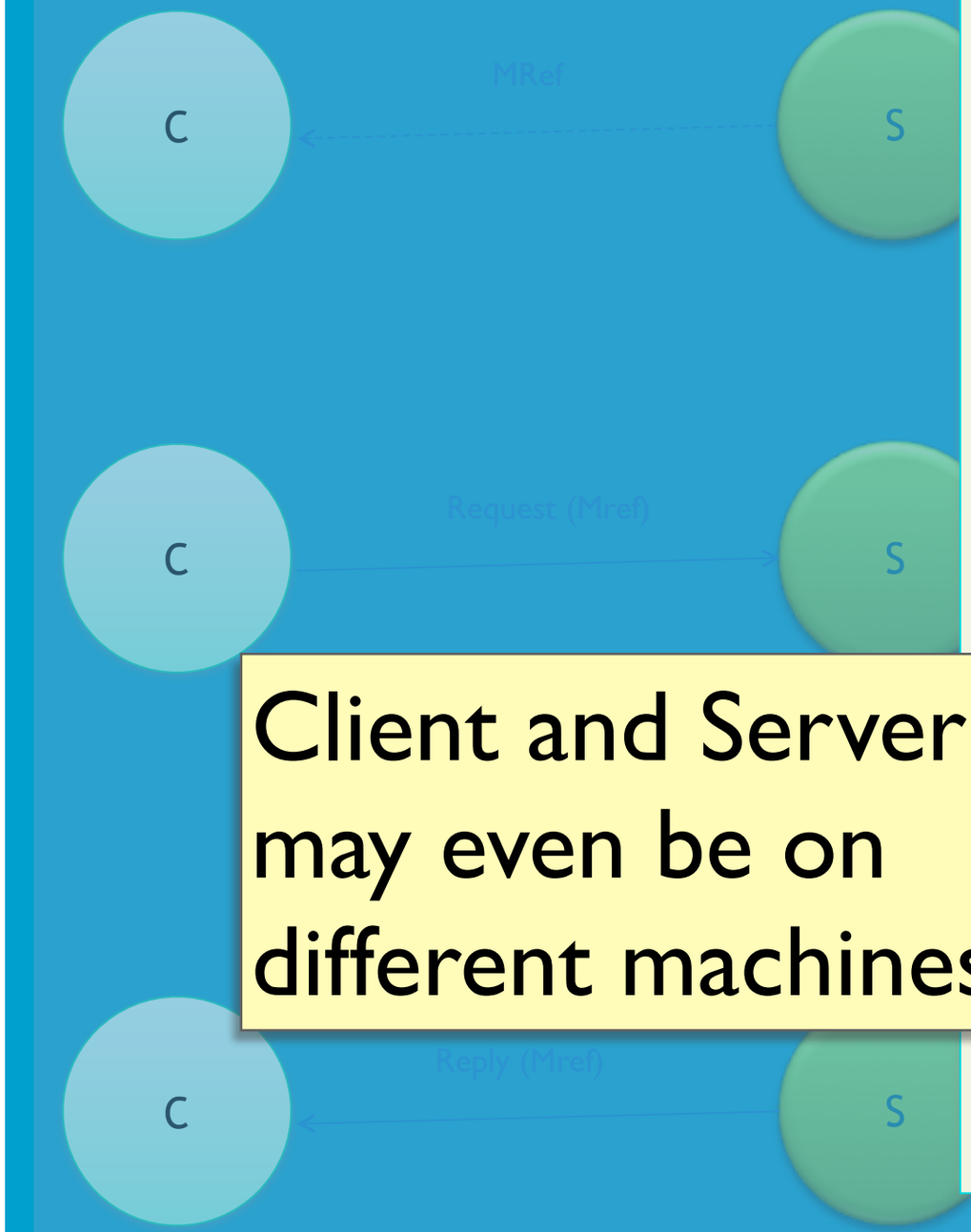
2 Client sends a request

3 (Blocks while waiting)

4 Server sends reply

FEUERLABS

Client-server in Erlang

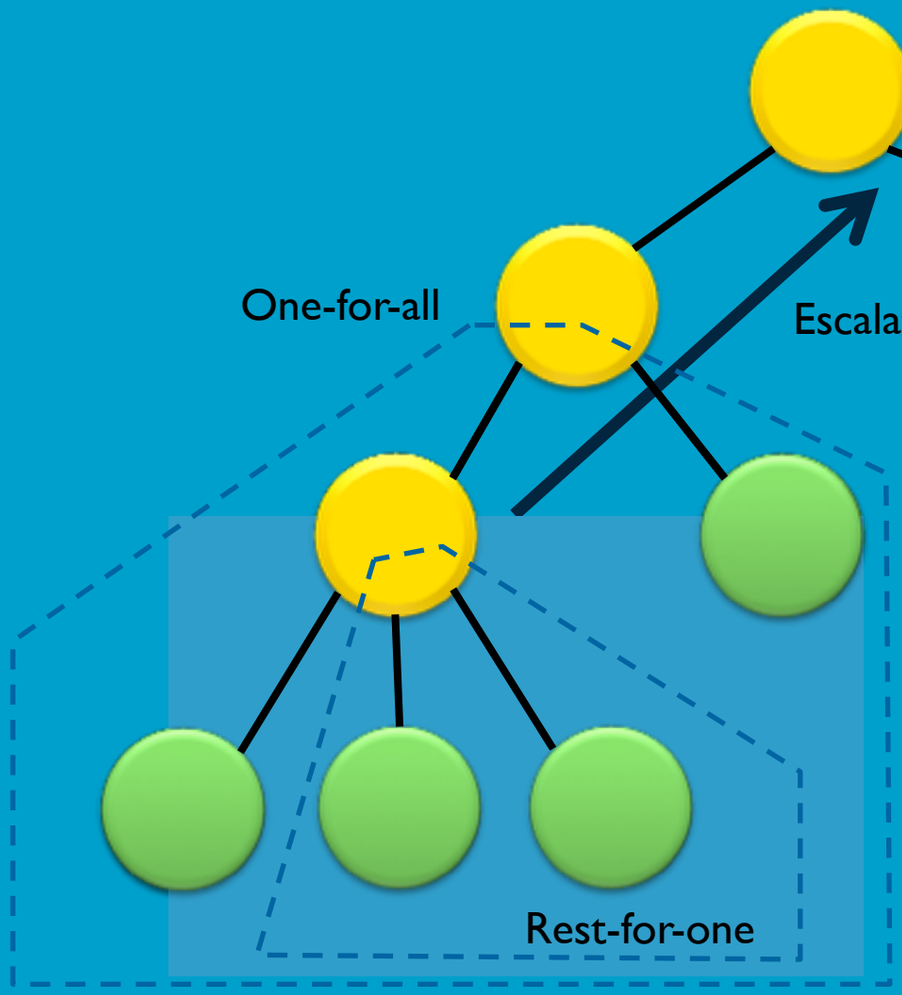


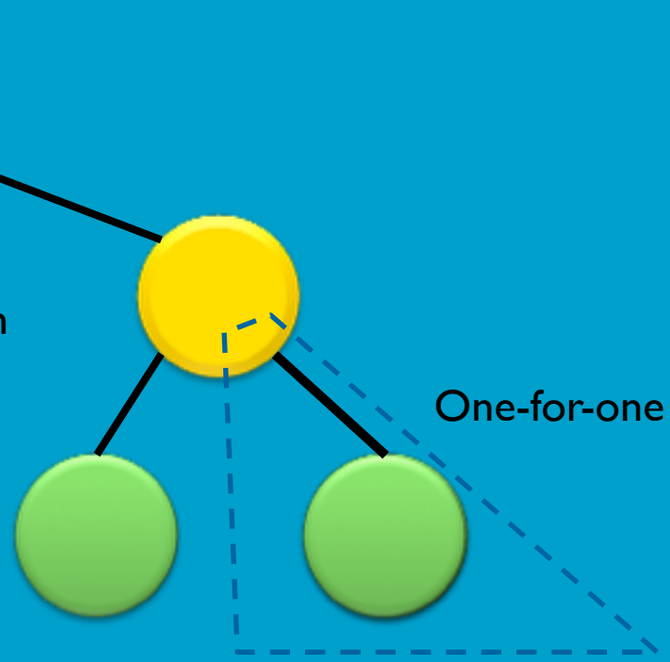
Client and Server may even be on different machines

```
call(S, Request, Timeout) ->
    Mref = monitor(process, S),
    S ! {call, Mref, Request},
    awaiting_reply(Mref, Timeout).

awaiting_reply(Mref, Timeout) ->
    receive
        {Mref, Reply} ->
            Reply;
        {'DOWN', Mref, _, _, Reason} ->
            error(Reason)
    after Timeout ->
        error(timeout)
    end.
```

Supervisors – Out-of-Band Error Handling

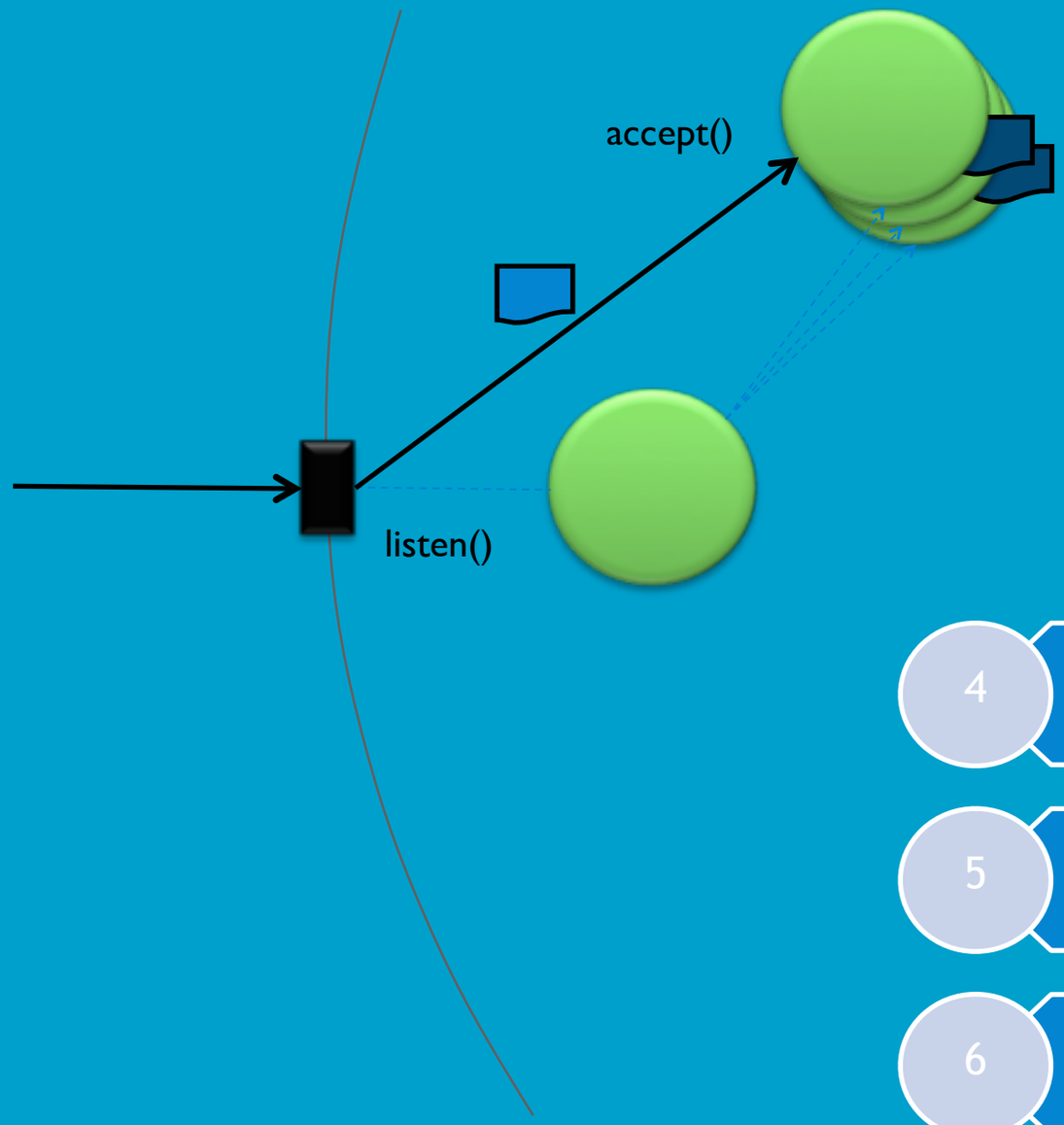




- Robust systems can be built using layering
- Program for the correct case

FEUERLABS

Handling sockets in Erlang

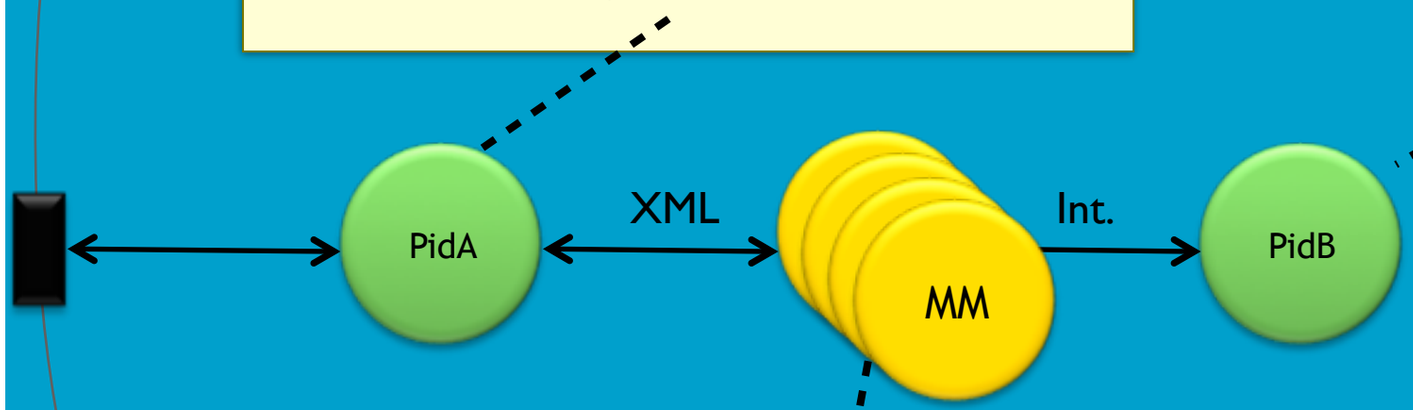


1. Static process opens listen socket
2. Spawns an acceptor process
3. Acceptor receives incoming
4. Acks back to socket owner
5. New acceptor is spawned
6. Replies sent directly to socket

FEUERLABS

Middle-man Processes

```
spawn_link(PidA, PidB) ->
    spawn_link(fun() ->
                   loop(#state{a_pid= PidA,
                               b_pid = PidB})
               end).
```




```
loop(#state{a_pid = PidA, b_pid = PidB} = State) ->
    receive
       {PidA, MsgBin} when is_binary(MsgBin) ->
           {simple_xml, _} = Msg = vccXml:simple_xml(MsgBin),
           PidB ! {self(), Msg},
           loop(State);
       {PidB, {simple_xml, _} = Msg} ->
           Bin = vccXml:to_XML(Msg),
           PidA ! {self(), Bin},
           loop(State)
    end.
```

```
await_negotiation(State) ->
    receive
        {From,
         {simple_xml,
          [{"offer", Attrs, Content}]}} ->
            HisOffer =
                inspect_offer(Attrs, Content),
            Offer = calc_offer(HisOffer, State),
            From ! {self(), Offer};
        …
    end.
```

- Practical because of light-weight concurrency
- Normalizes messages
- Main process can pattern-match on messages
- Keeps the main logic clear

FEUERLABS

Language Model Affects our Thinking

Example: RFC 3588 – DIAMETER Base Protocol

state	event	action	next state
	...		
I-Open	Send-Message	I-Snd-Message	I-Open
	I-Rcv-Message	Process	I-Open
	I-Rcv-DWR	Process-DWR, I-Snd-DWA	I-Open
	I-Rcv-DWA	Process-DWA	I-Open
	R-Conn-CER	R-Reject	I-Open
	Stop	I-Snd-DPR	Closing
	...		

Transport FSM

Watchdog FSM

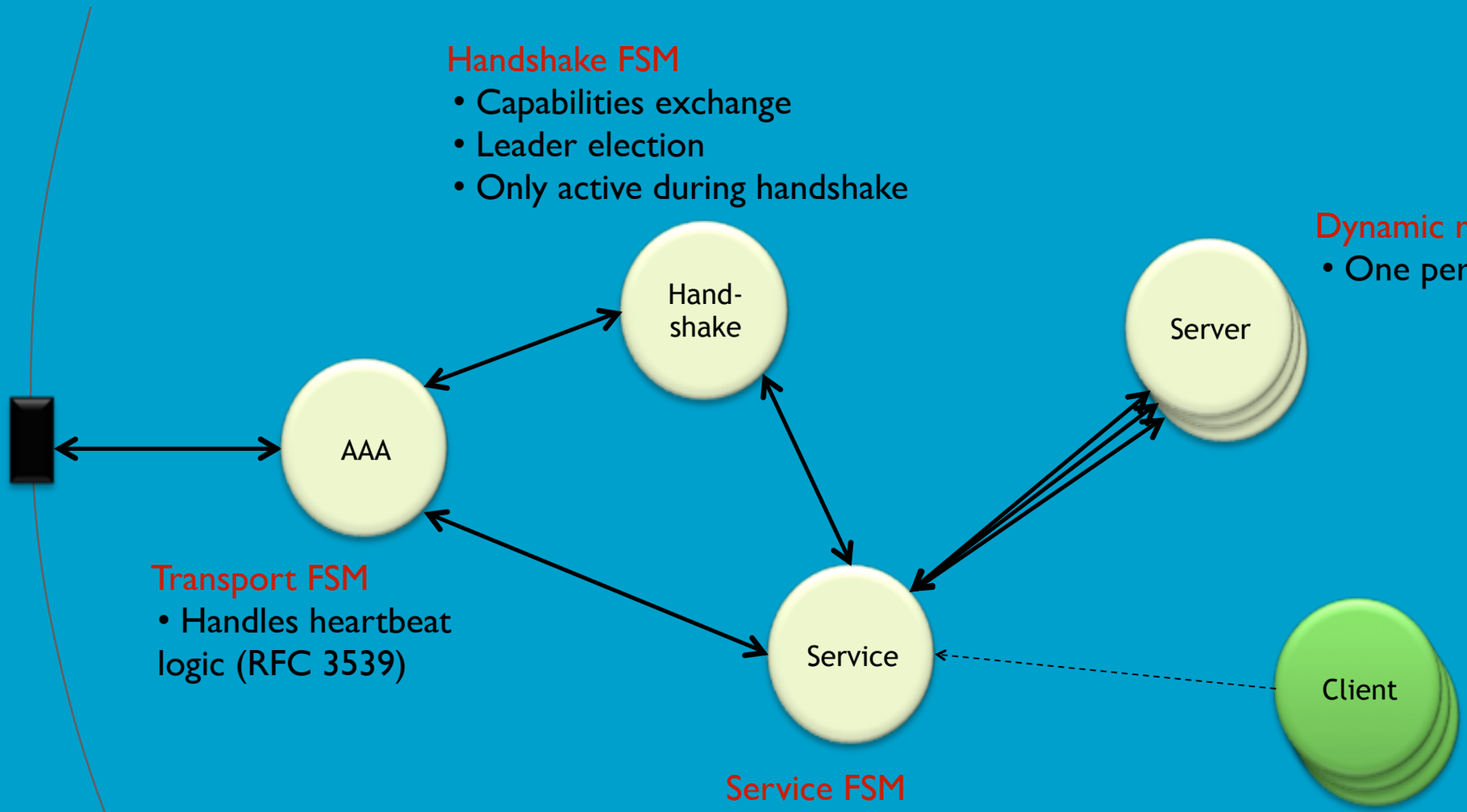
Handshake FSM

- Three state machines described as one
- Implies a single-threaded event loop
- Introduces accidental complexity

Use processes to separate concerns

Handshake FSM
- Capabilities exchange
- Leader election
- Only active during handshake

Dynamic request handler
- One per request



Transport FSM
- Handles heartbeat logic (RFC 3539)

Service FSM
- Request routing
- Failover
- Retransmission

Soft Upgrade

- Atomic per-module, per-process code switch
- Plus high-level support for system upgrade

```
9> setup:reload_app(gproc).
[gproc vsn "0.2.12-20-gc60b9b4"] soft upgrade from "0.2.7"
{ok,[]}
```

```
7> os:putenv("ERL_LIBS", "/Users/uwiger/FL/git").
true
8> setup:find_app(gproc).
[{"0.2.7","/Users/uwiger/ETC/git/gproc/ebin"},
 {"0.2.12-20-gc60b9b4","/Users/uwiger/FL/git/gproc/ebin"}]
```

```
9> setup:reload_app(gproc).
[gproc vsn "0.2.12-20-gc60b9b4"] soft upgrade from "0.2.7"
{ok,[]}
```

From the Tar Pit

- Complexity is the single major difficulty in the successful development of large-scale software systems.
- Following Brooks we distinguish accidental from essential difficulty, but disagree with his premise that most complexity remaining in contemporary systems is essential.

The Tar Pit (Moseley, Marks, 2006)

Complex State Machines

What makes FSMs Complex?

- Multi-way messaging
- History-dependent states
- Failures

FEUERLABS

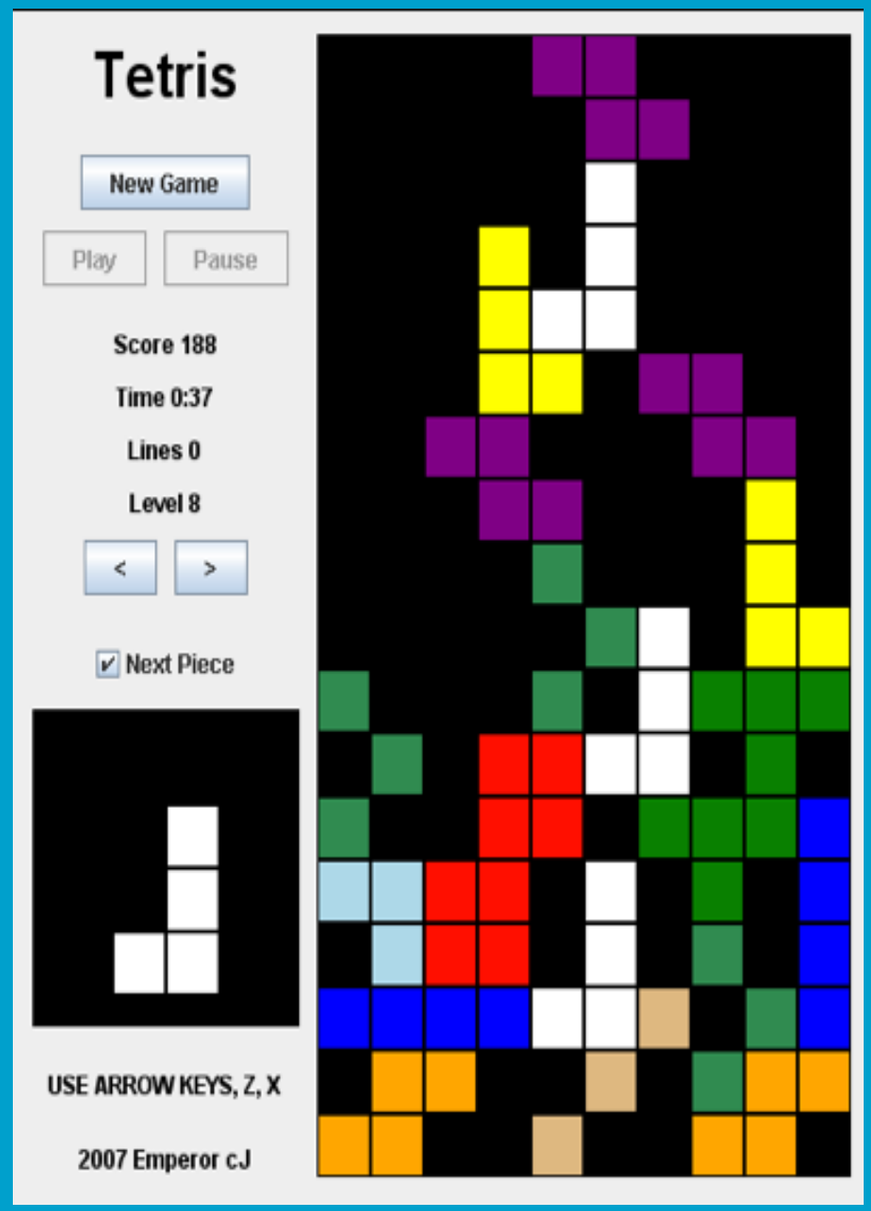
Tetris Management

- The age-old classic has coined a new time management method
- The idea: learn how to keep the pile small

FEUERLABS

Tetris Management

- Used in a derogatory sense at a major software development project
- As in ”reactive management without a plan”
- Basically, don’t let your project become a tetris game



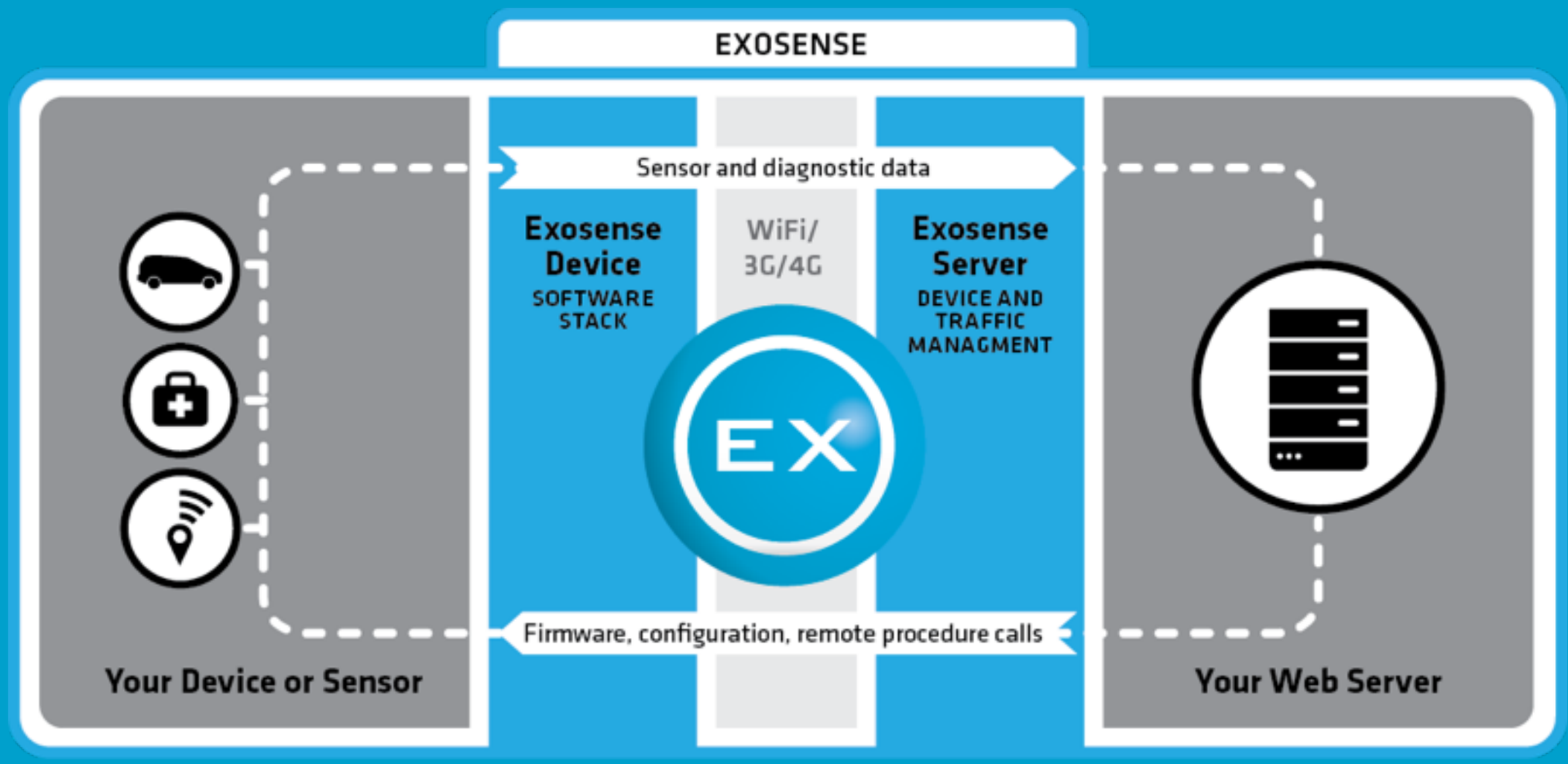
A different kind of puzzle

- What if your problem more resembles this?
- Would you attack this problem with a tetris approach?

http://www.worldslargestpuzzle.com/hof-008.html

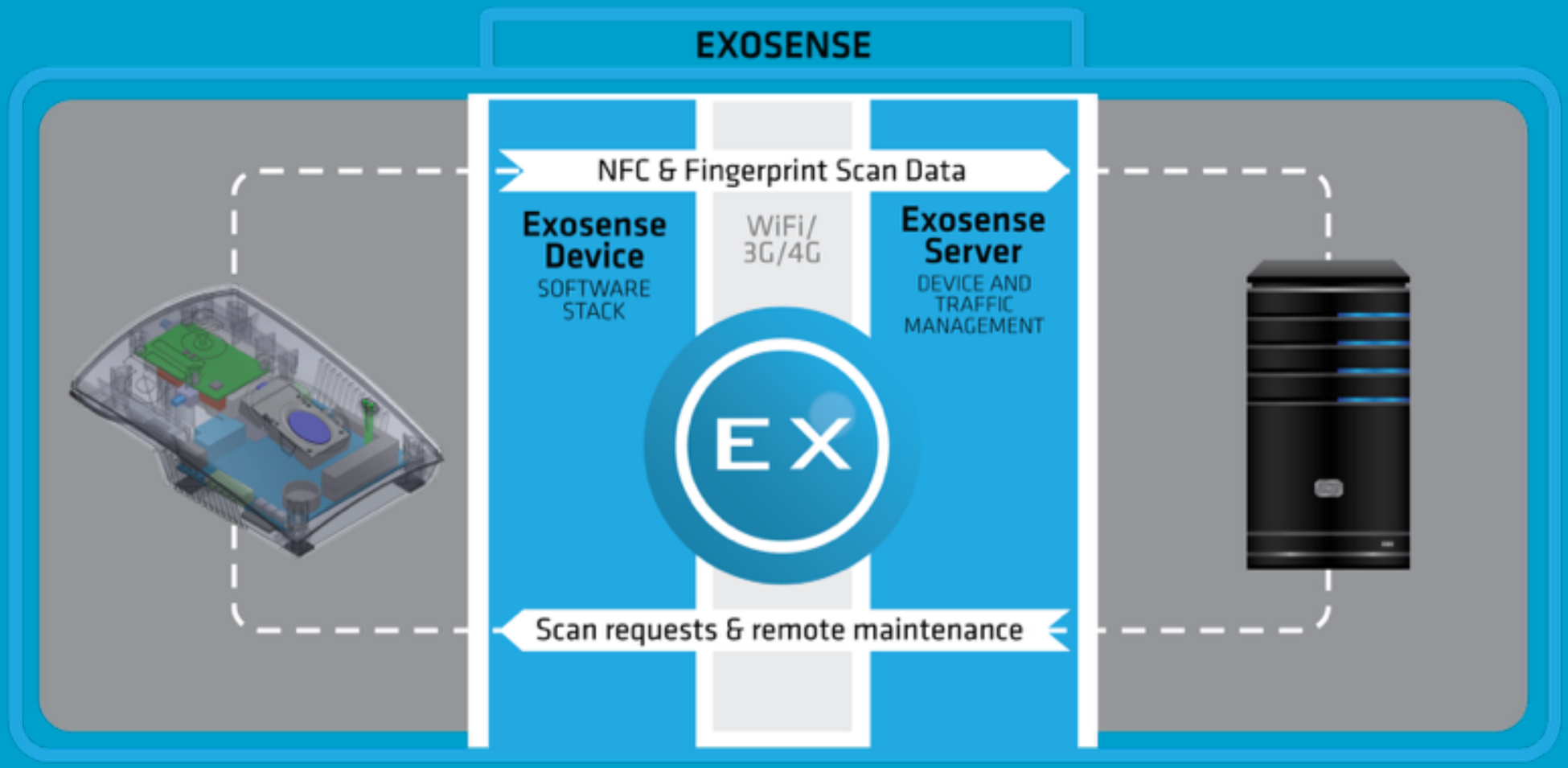
FEUERLABS

Shameless Pitch—Feuerlabs



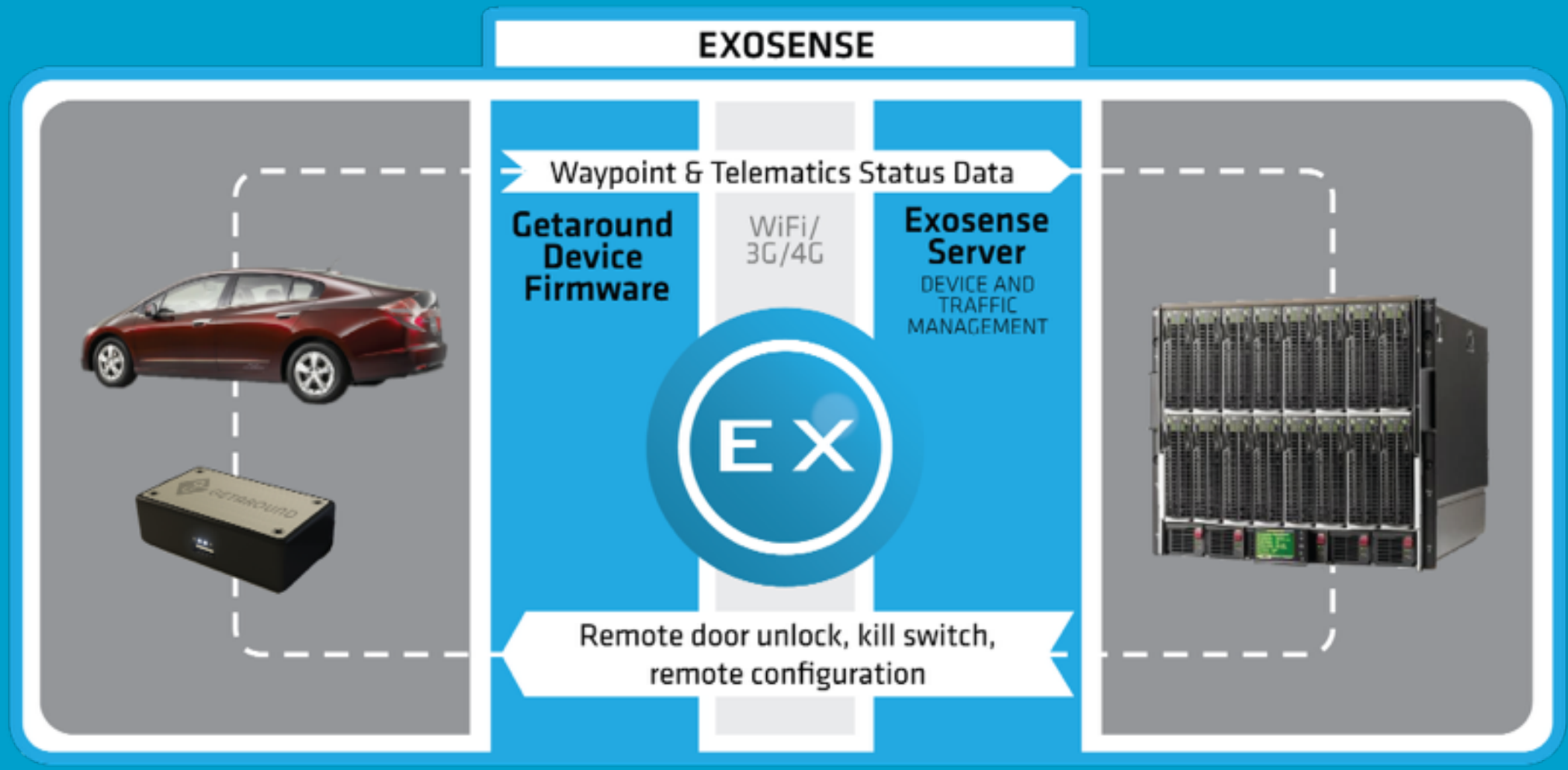
FEUERLABS

Connected Health



FEUERLABS

Peer-to-Peer Car Sharing



FEUERLABS

Demo