BEAM on the Edge

Innovation Through Problem Solving

Robert Virding
Frank Hunleth

Robert Virding

Erlang on the Edge

The Problem

- Ericsson’s “best seller” AXE telephone exchanges (switches) required large efforts to develop and maintain software.
- The problem for the CSLab to solve was how to make programming these types of applications easier, but keeping the same characteristics.

The problem domain

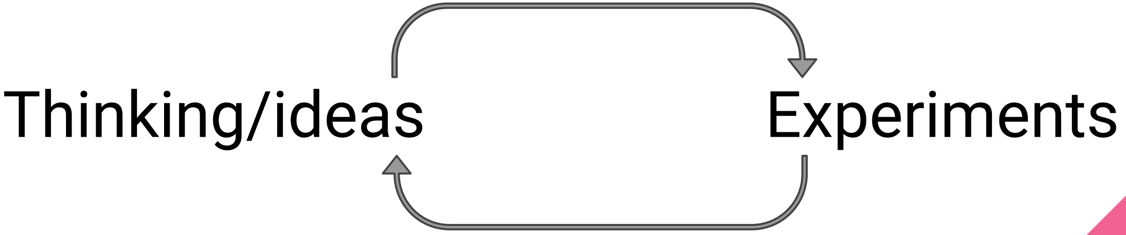
- Handling of very large numbers of concurrent activities
- Actions to be performed at a certain point in time or within a certain time
- System distributed over several computers
- Continuous operation over many years
- Software maintenance (reconfiguration, etc.) without stopping the system
- Fault tolerance both to hardware failures and software errors

Not just telecom

Bjarne Däcker, November 2000 - Licentiate Thesis

Internal development

- Many threads at the same time
- Understanding the problem domain
- Designing language and architecture which could be used in these systems
- Testing with our idea of how these systems should perform

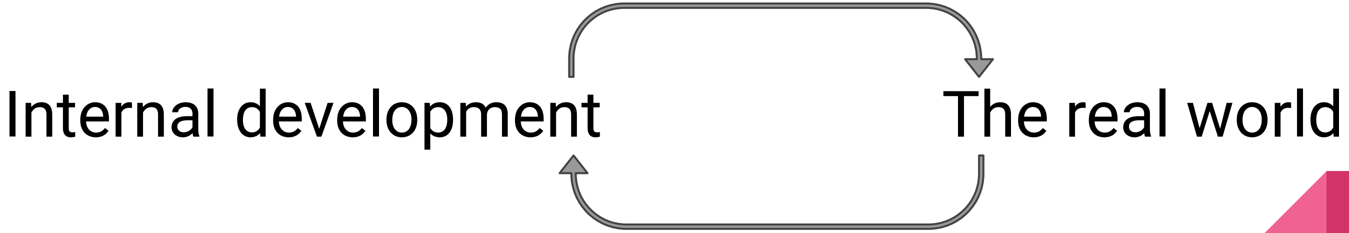


Thinking/ideas

Experiments

The real world

- We worked together with another project in Ericsson (ACS/Dunder) who tested our ideas and gave a lot of very good feedback.
- This allowed us to rethink and come with new ideas for which we then got feedback.
- They were the first users of Erlang in a real product.



Internal development

The real world

The solution: first principles

- Lightweight concurrency
  - Must handle a large number of processes
  - Process creation, context switching and inter-process communication must be cheap and fast.
- Asynchronous communication
- Process isolation
  - What happens in one process must not affect any other process.
- Error handling
  - The system must be able to detect and handle errors.
- Continuous evolution of the system
  - We want to upgrade the system while running and with no loss of service.
- Soft real-time, non-blocking

The solution: first principles

Also

- High level language to get real benefits.
- The language/system should be simple
  - Simple in the sense that there should be a small number of basic principles, if these are right then the language will be powerful but easy to comprehend and use. Small is good.
  - The language should be simple to understand and program.
- Provide tools for building systems, not solutions
  - We would provide the basic operations needed for building communication protocols and error handling

The language: sequential

- Simple functional language
  - With a “different” syntax
- Typical features of functional languages
  - Immutable data
  - Immutable variables
  - Extensive use of pattern matching
  - Recursion rules!
- Dynamically typed!
- No user defined data-types!

The language: concurrency

- Light-weight isolated processes
  - Millions of Erlang processes possible on one machine
- Asynchronous message passing
  - Only method of communication between processes
  - Necessary for non-blocking systems
  - Provide basic mechanism
  - Very cheap
- Selective receive mechanism
  - Allows us to ignore messages which are uninteresting now
- NO GLOBAL DATA!

The language: error handling

- Links
- Exit signals
  - Kill processes
- Trapping errors
  - Allow using links to monitor processes

The language: trivial code example

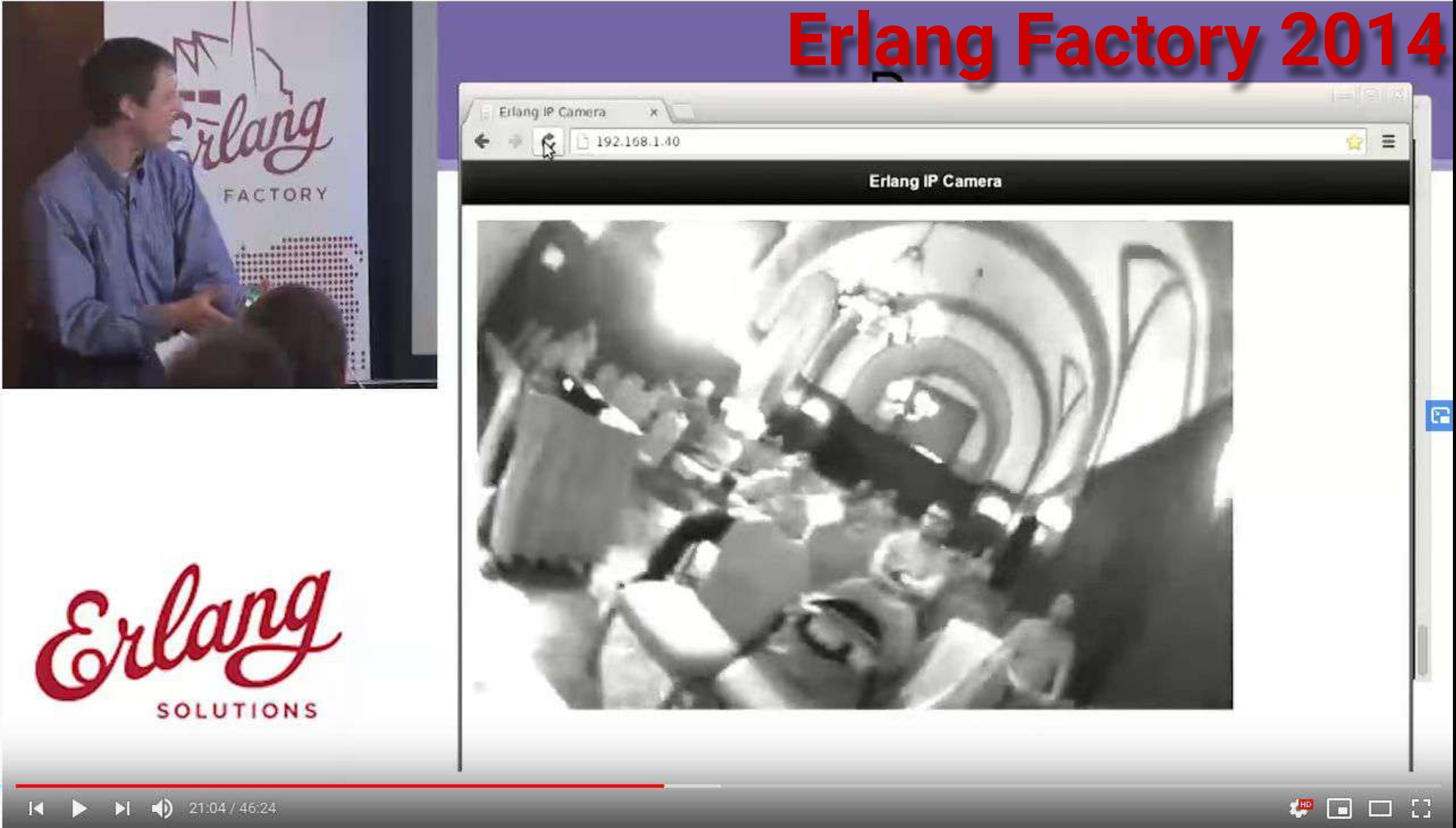
```
ringing_a_side(Addr, B_Pid, B_Addr) ->
  receive
    on_hook ->
      B_Pid ! cleared,
      tele_os:stop_tone(Addr),
      idle(Addr);
    answered ->
      tele_os:stop_tone(Addr),
      tele_os:connect(Addr, B_Addr),
      speech(Addr, B_Pid, B_Addr);
    {seize,Pid} ->
      Pid ! rejected,
      ringing_a_side(Addr, B_Pid, B_Addr);
    _ ->
      ringing_a_side(Addr, B_Pid, B_Addr)
  end.
```

```
ringing_b_side(Addr, A_Pid) ->
  Receive
    cleared ->
      tele_os:stop_ring(Addr),
      idle(Addr);
    off_hook ->
      tele_os:stop_ring(Addr),
      A_Pid ! answered,
      speech(Addr, A_Pid, not_used);
    {seize,Pid} ->
      Pid ! rejected,
      ringing_b_side(Addr, A_Pid);
    _ ->
      ringing_b_side(Addr, A_Pid)
  end.
```

Frank Hunleth

Embedded systems on the Edge

Erlang Factory 2014



IIoT

"The Industrial Internet of Things is the use of smart sensors and actuators to enhance manufacturing and industrial processes."

Datacenter UPS

Embedded Systems
Self-contained and single purpose

Embedded systems for me going into the 00s

- Large low level C/C++ codebases
- Increasingly networked
- Transitioning from all in-house development

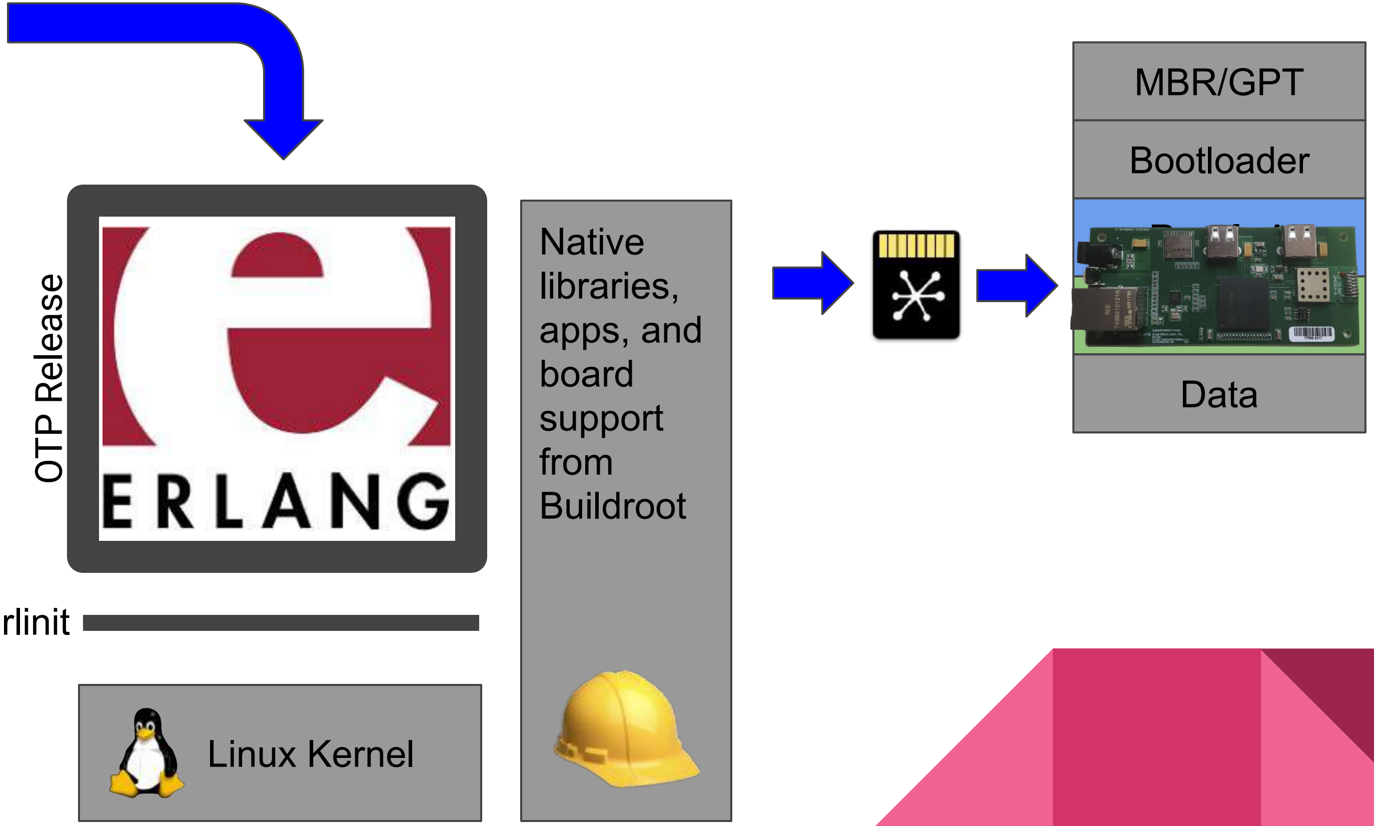
Embedded systems for me going into the 00s

- Message-based communication
- Failure recovery by restarting subsystems
- Rapidly falling processor prices

Nerves

Tools, runtime, and libraries for creating robust embedded systems using Elixir

Nerves



OTP Release

erlinit

Linux Kernel

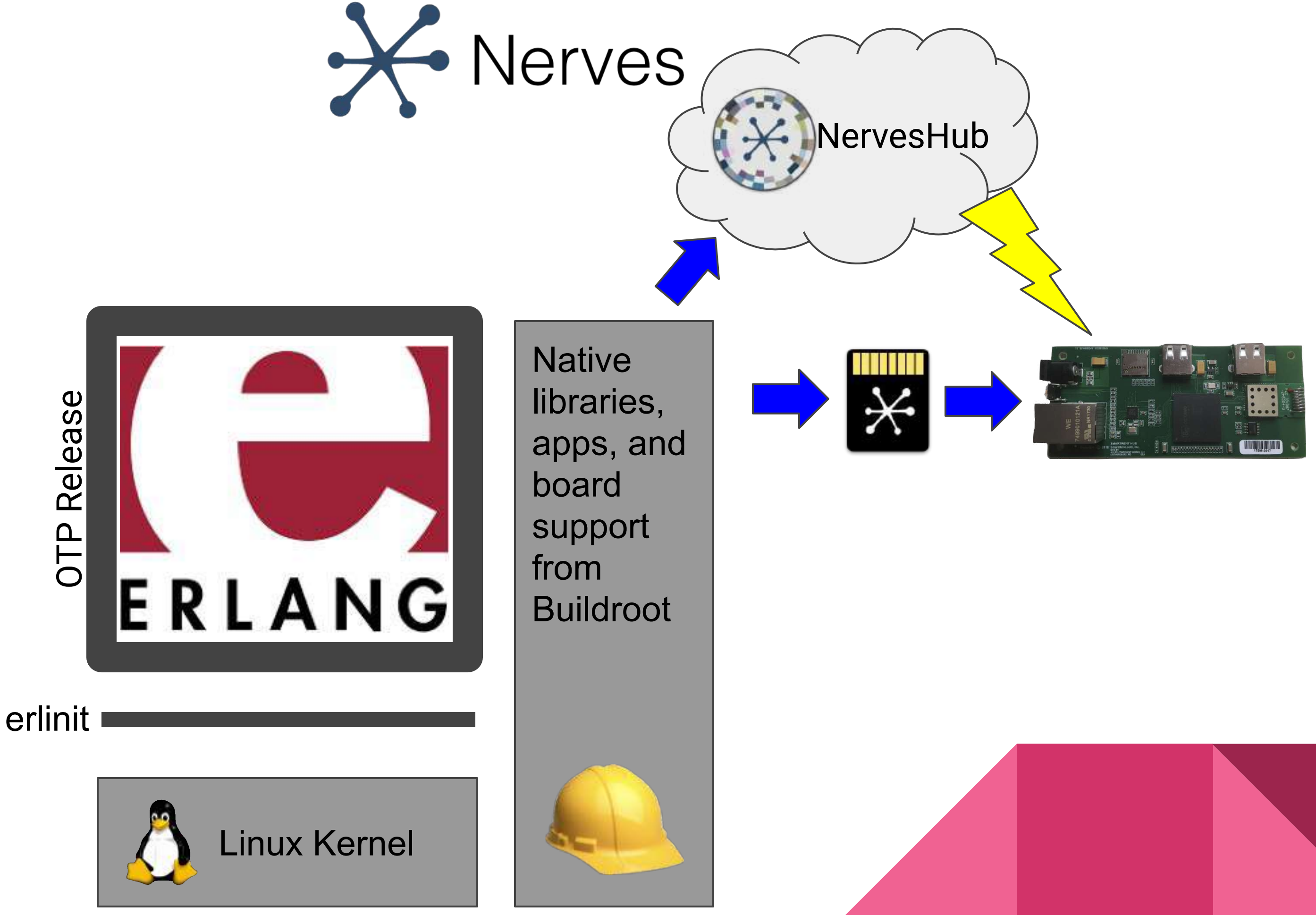
Native libraries, apps, and board support from Buildroot

MBR/GPT

Bootloader

Data

Nerves



Native libraries, apps, and board support from Buildroot

Reviewing the path so far...

Datacenter UPS

BEAM on the Edge

Innovation Through Problem Solving

Robert Virding
Frank Hunleth