Building an IP Network Camera

Frank Hunleth
Twitter: @fhunleth
Erlang Factory 2014

Agenda

- **Introducing Erlang to a project**
- The camera
- Demo
- Embedded plumbing and development
- Performance
- Conclusion

Embedded Calibration Slide

Limited resource/
Specific purpose

General
purpose

Microcontrollers

RTOS

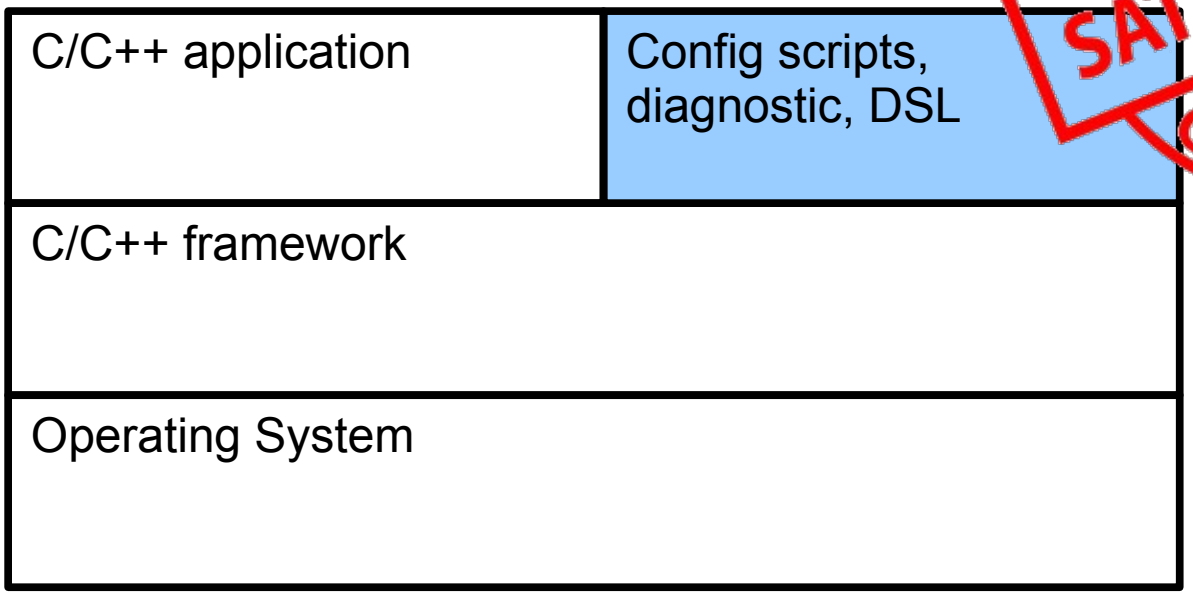
Embedded
Linux

Android, iOS,
Ubuntu, Windows

32-bit Microprocessors

This talk is focused in this area of embedded

How to use Language X in an Embedded Project



- Low risk - can always implement in C first
- Worked in the past for TCL, Lua, and Javascript
- My first attempt to introduce Erlang to an organization so that it could be used for real on the next project

Fail

What makes Erlang/OTP interesting is its focus on robust and scalable **systems**.

Try #3,4, or 5 - IP Camera

- Build a non-trivial embedded device
  - Didn't need to be a camera, but wanted a device with a hard real-time component and decent network usage
  - Make sure that SW infrastructure supports production use cases
  - Should benefit from in robustness or simplicity by using Erlang/OTP
- Questions
  - Will Erlang be too slow?
  - Will Erlang require too much DRAM and Flash memory?
  - Will developing Erlang in a cross-compiled environment be a pain?
  - Will I miss all of the libraries and frameworks available in C?
  - Will I have confidence in the Erlang/OTP platform?

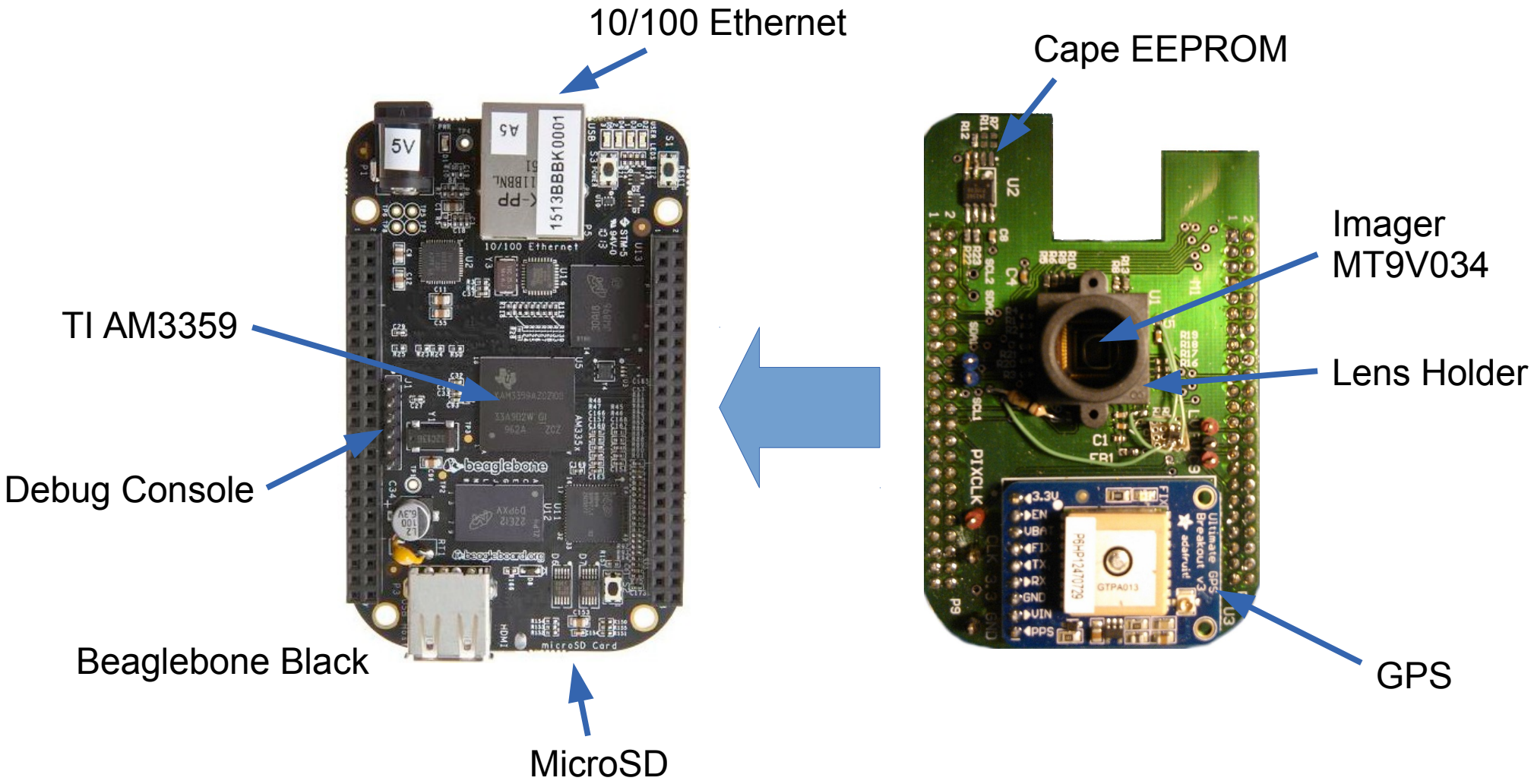
Constraint

- If a feature exists in both Erlang and an embedded Linux environment, use the Erlang version
- Examples
  - No shell scripts
  - No SysV init, systemd, upstart (these provide process initialization and supervision)

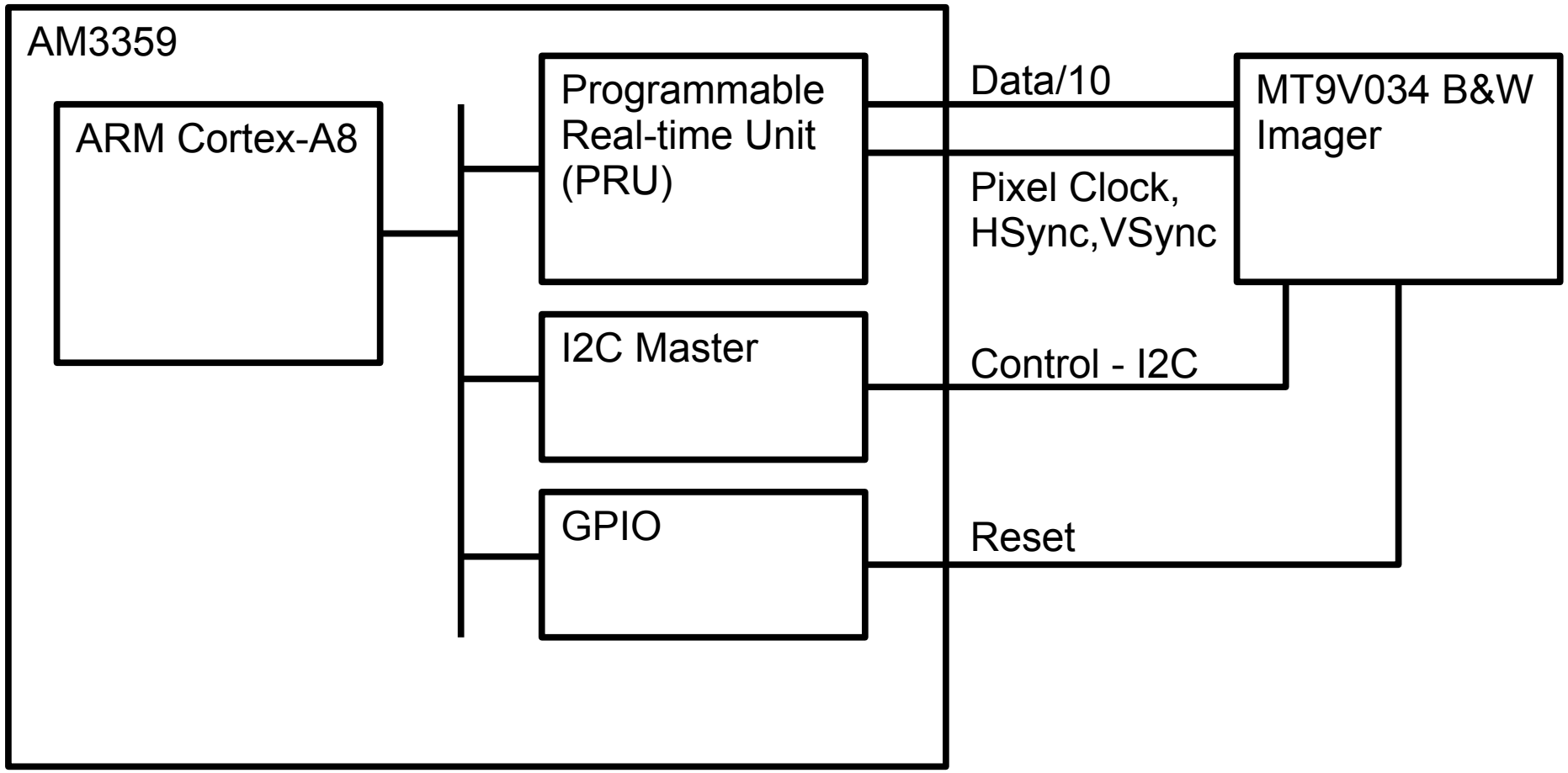
Agenda

- Introducing Erlang to a project
- **The camera**
- Demo
- Embedded plumbing and development
- Performance
- Conclusion

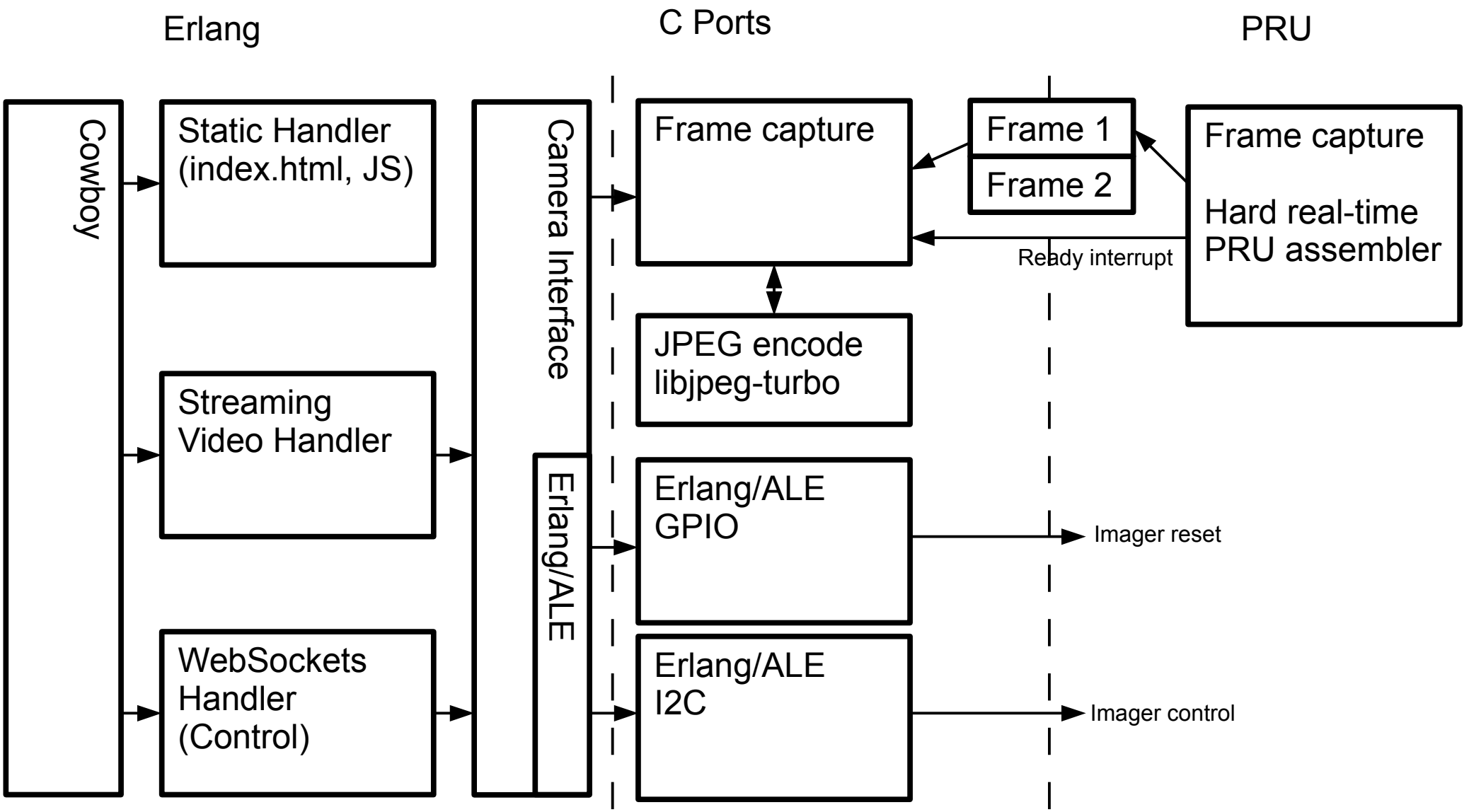
Hardware Setup



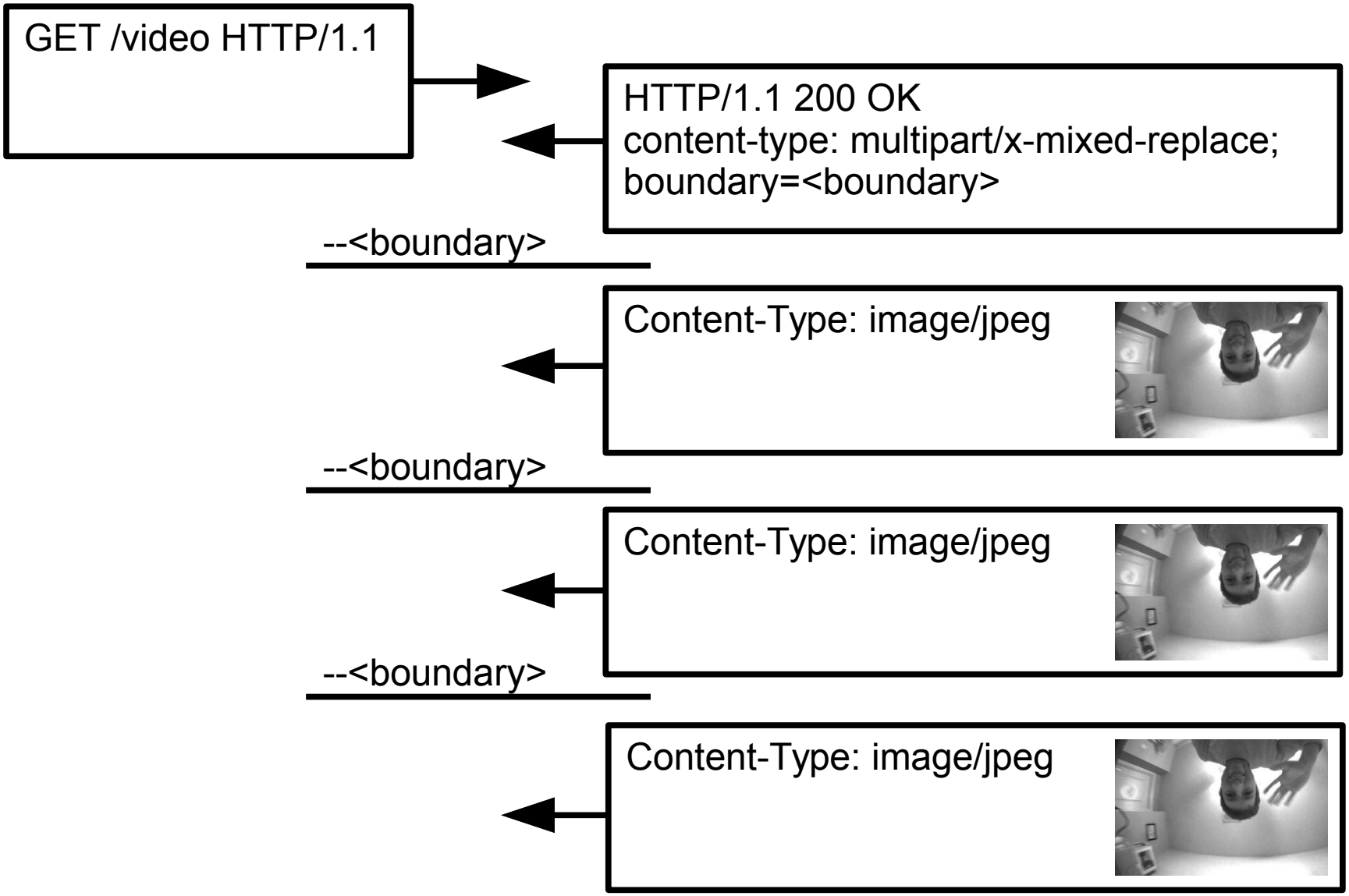
Hardware Components



High-level Software Components



Motion JPEG Streaming

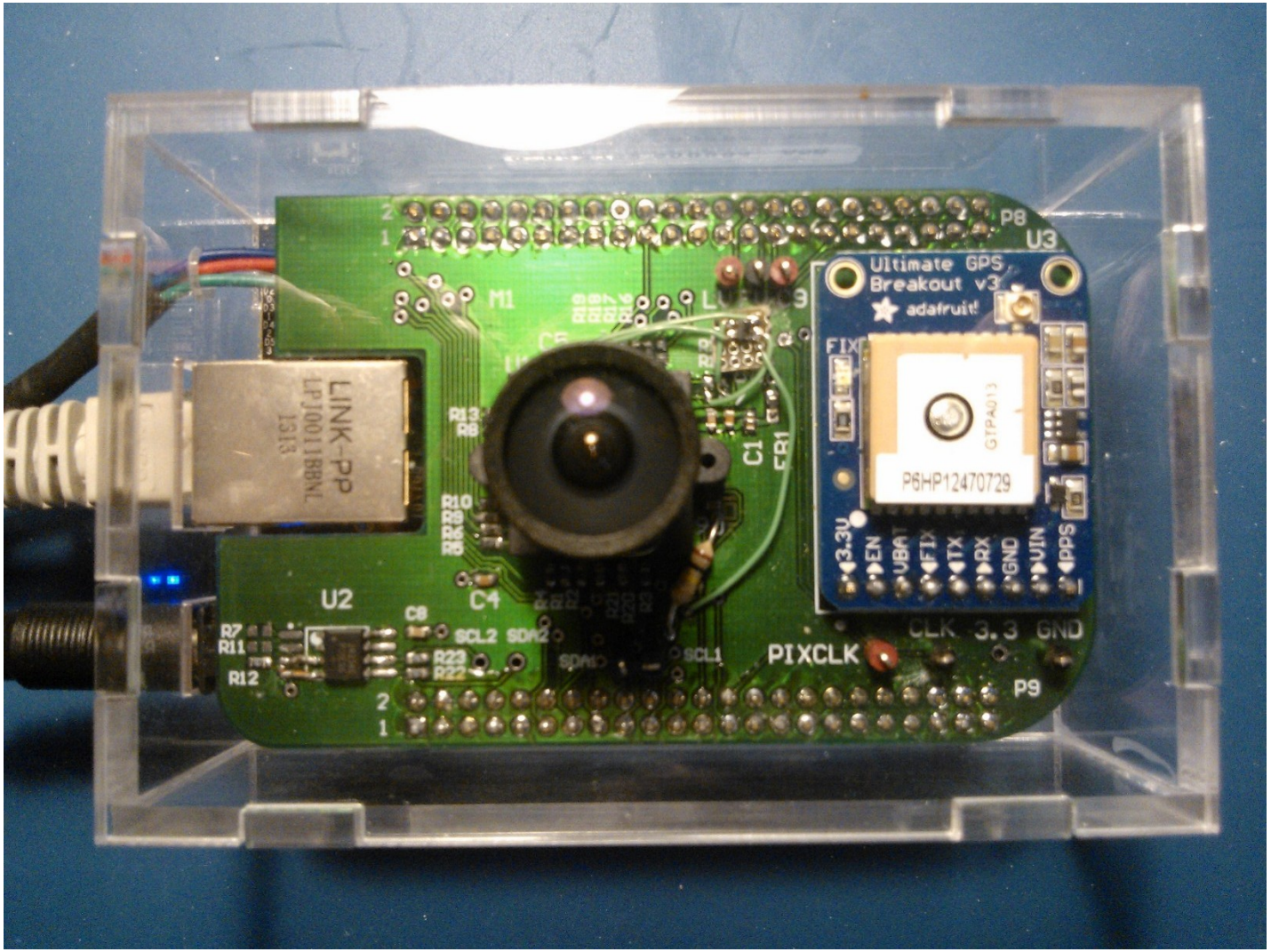


Streaming Code

```
handle(Req, _State) ->
    Boundary = boundary(),
    Headers =
        [{<<"MIME-Version">>, <<"1.0">>},
         {<<"content-type">>, <<"multipart/x-mixed-replace; ",
                                "boundary=", Boundary/binary>>}],
    {ok, Req2} = cowboy_req:chunked_reply(200, Headers, Req),
    send_first_picture(Req2),
    send_pictures(Req2).

send_pictures(Req) ->
    Pic = troodon_cam:get_next_picture(),
    Msg = [multipart_header(Pic), Pic, delimiter()],
    ok = cowboy_req:chunk(Msg, Req),
    send_pictures(Req).
```

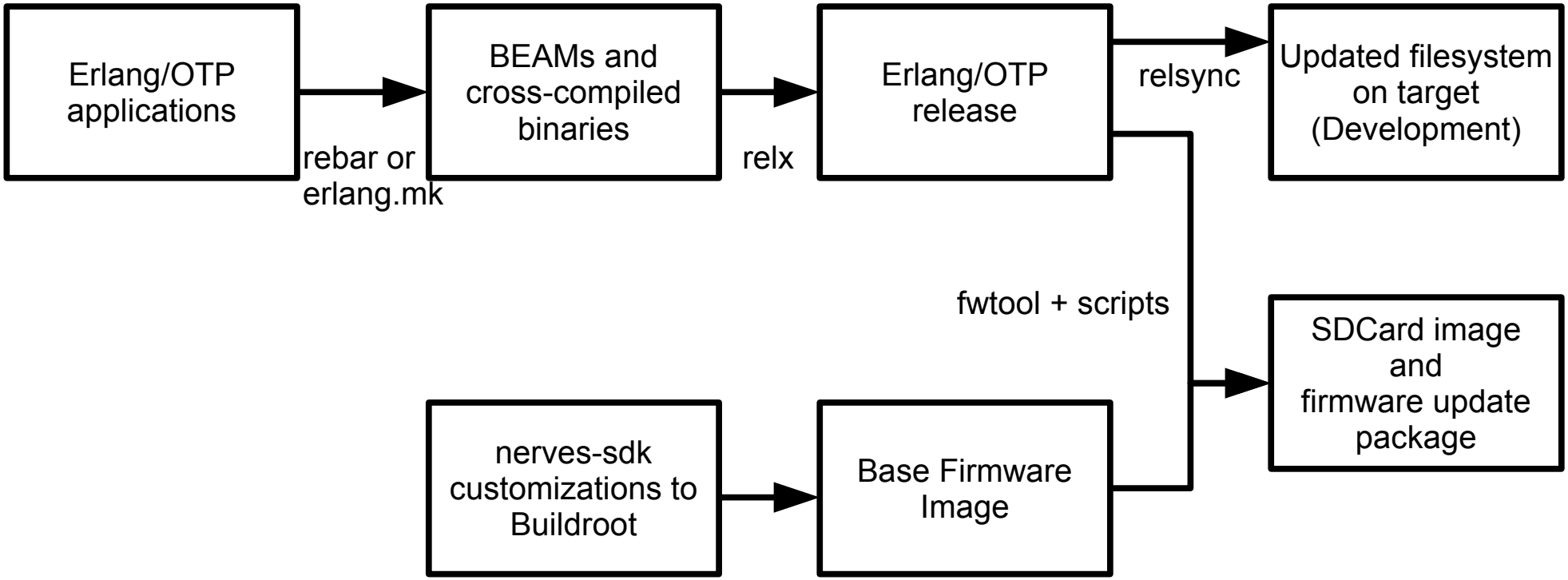
Demo


Ultimate GPS
Breakout v3
adafruit!
P6HP12470729
LINK-PP
U2
U3
P8
P9
C4
C1
M1
R7
R11
R12
R13
R10
SCL2
SDA2
SCL1
PIXCLK
CLK
3.3
GND
3.3V
EN
VBAT
FIX
TX
RX
GND
VIN
PPS

Agenda

- Introducing Erlang to a project
- The camera
- Demo
- **Embedded plumbing and development**
- Performance
- Conclusion

Nerves-Project Work Flow



Raw SDCard Image

Master Boot Record

Bootloaders

Root Filesystem A

Read-only

Root Filesystem B

Read-only

Application Data Partition

Read-write

Linux kernel
Erlang
Libraries
Main application

Ping-pong location used when upgrading the firmware

Majority of nonvolatile memory for use by the application

Raw images are only needed for initial code load and bulk device programming

relsync

- Reprogramming SDCards gets old quickly!!!
- relsync synchronizes the files in the generated Erlang/OTP release directory with corresponding ones on the target
- Like *rsync* except
  - Communicates via the Erlang distribution protocol
  - Reloads modules that changed
  - Runs scripts pre and post sync to stop and start Erlang/OTP applications (needed to update ports)
- Limitations
  - Target must have writable FS (currently using a union FS)
  - NIFs and linked-in port drivers can't be updated

Initialization - Booting to Erlang

- erlinit
  - Replacement for /sbin/init that starts an Erlang/OTP release
  - Similar to a release start script, but in C
  - Supports remounting root fs with unionfs for development
  - Configurable via Linux kernel command line

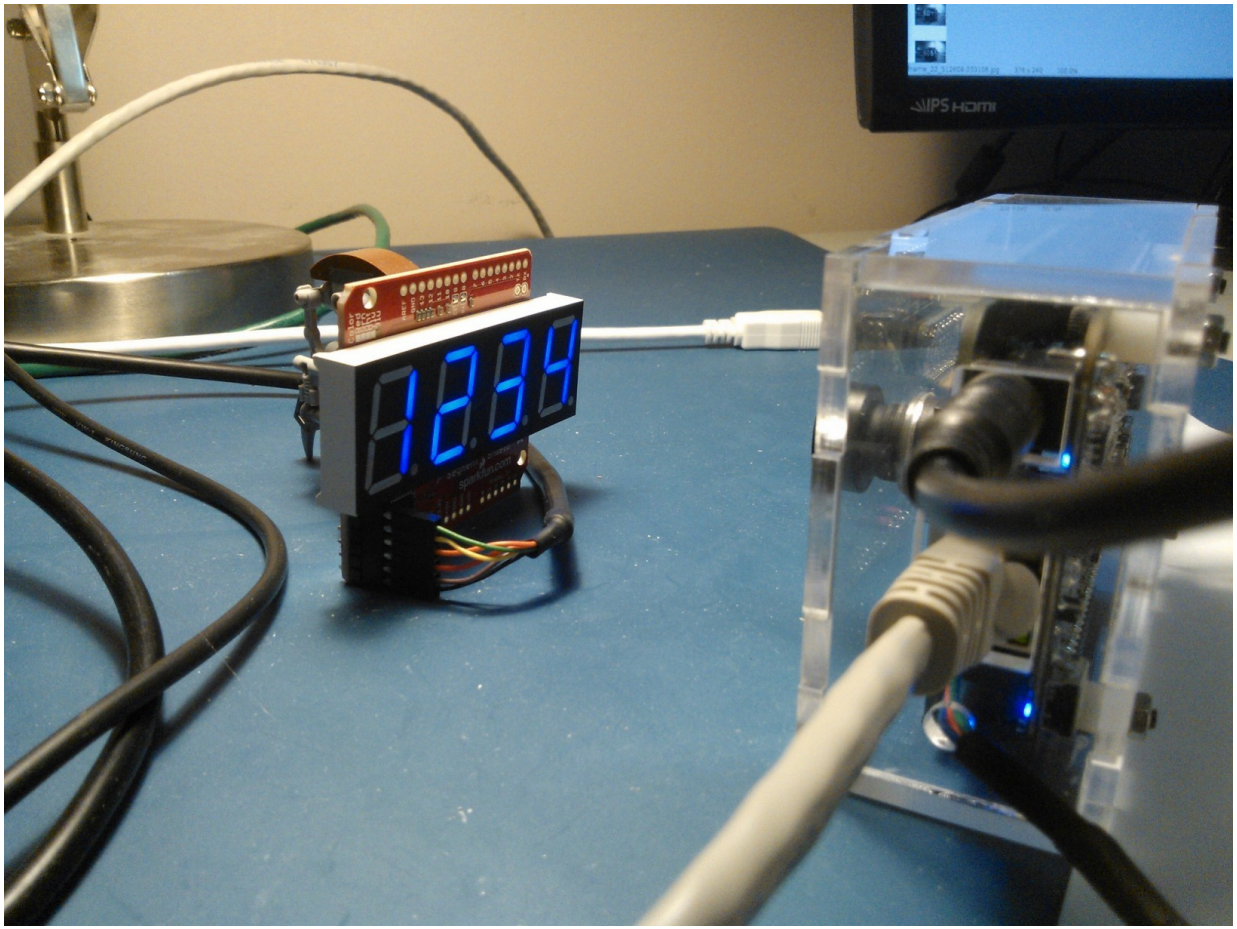
Agenda

- Introducing Erlang to a project
- The camera
- Demo
- Embedded plumbing and development
- **Performance**
- Conclusion

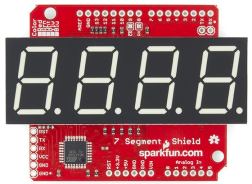
Performance - Throughput

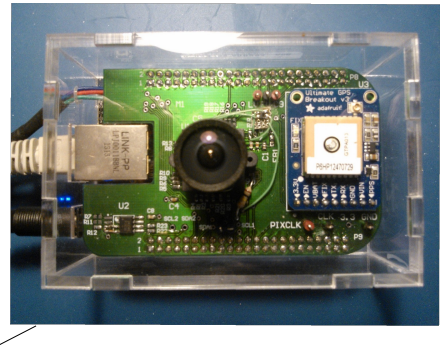
- Imager set to capture at 45 fps (22 ms/frame)
- Browser stats
  - Average FPS ~43.5 fps (23 ms/frame)
  - About 1 frame dropped per second
- Profiling revealed JPEG encode time taking 22-23.5 ms
  - AM3358 doesn't support HW encoding
  - Did not investigate tuning JPEG Turbo library
- Take away: Erlang is not a bottleneck

End-to-end Latency Measurements



Software Setup





UART
via erlang-serial

Latency
Measurement App
(Erlang, of course)

HTTP
via httpc

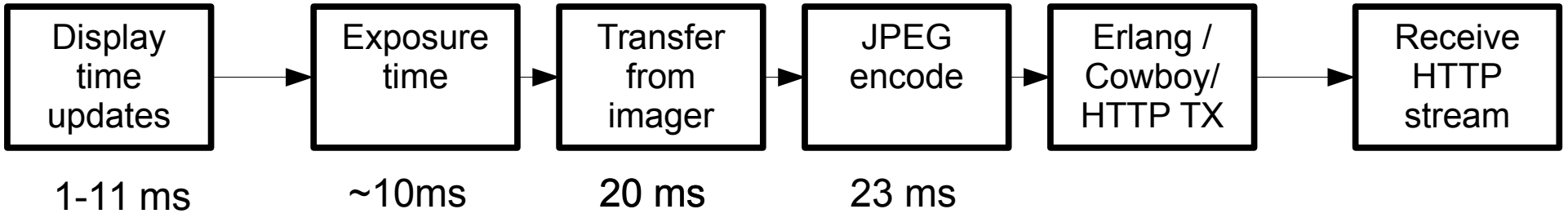
Timestamped
Frames

Received: 0.481 (~71 ms)

More info: https://github.com/fhunleth/cam_latency

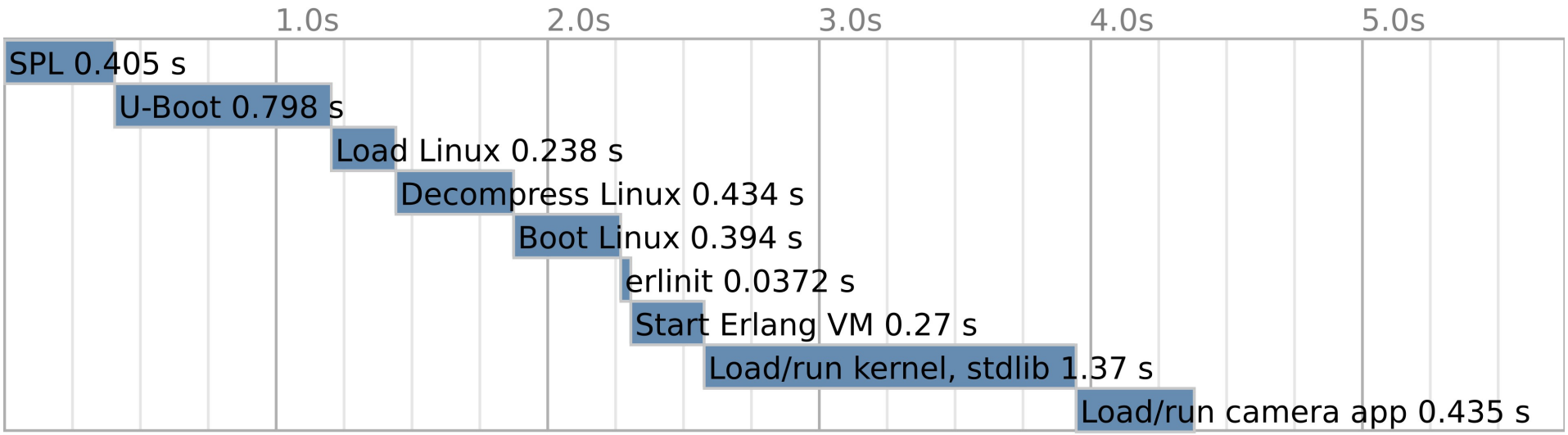
Latency Results

- Average frame latency 78 ms (best 66 ms, worst 94 ms)

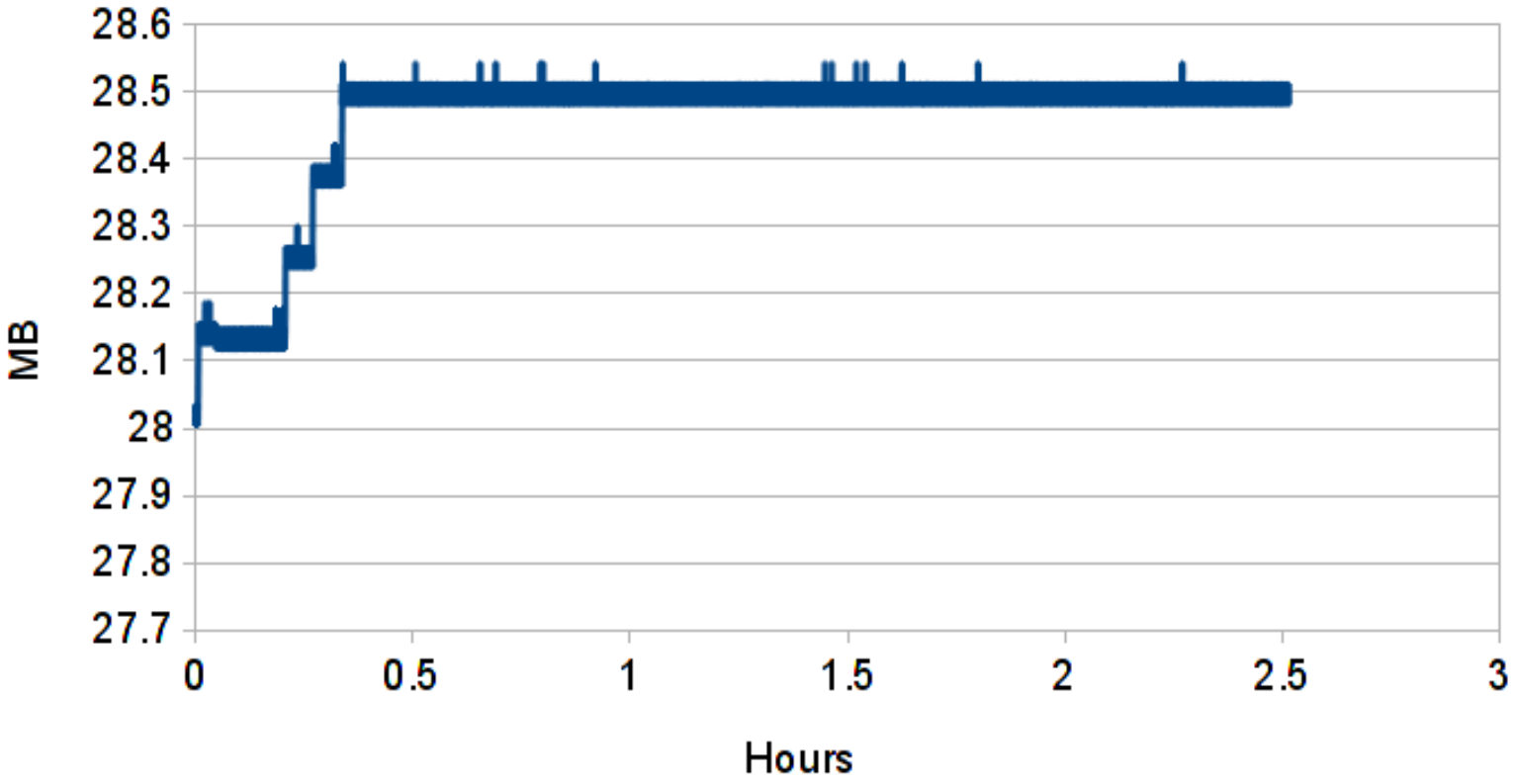


- Easily accounted for latencies - 54-66 ms
- Remaining latencies - 12-30 ms
  - Not unexpected; requires more instrumentation
  - Doubt that Erlang overhead is significant

Boot Time (4.38 seconds)

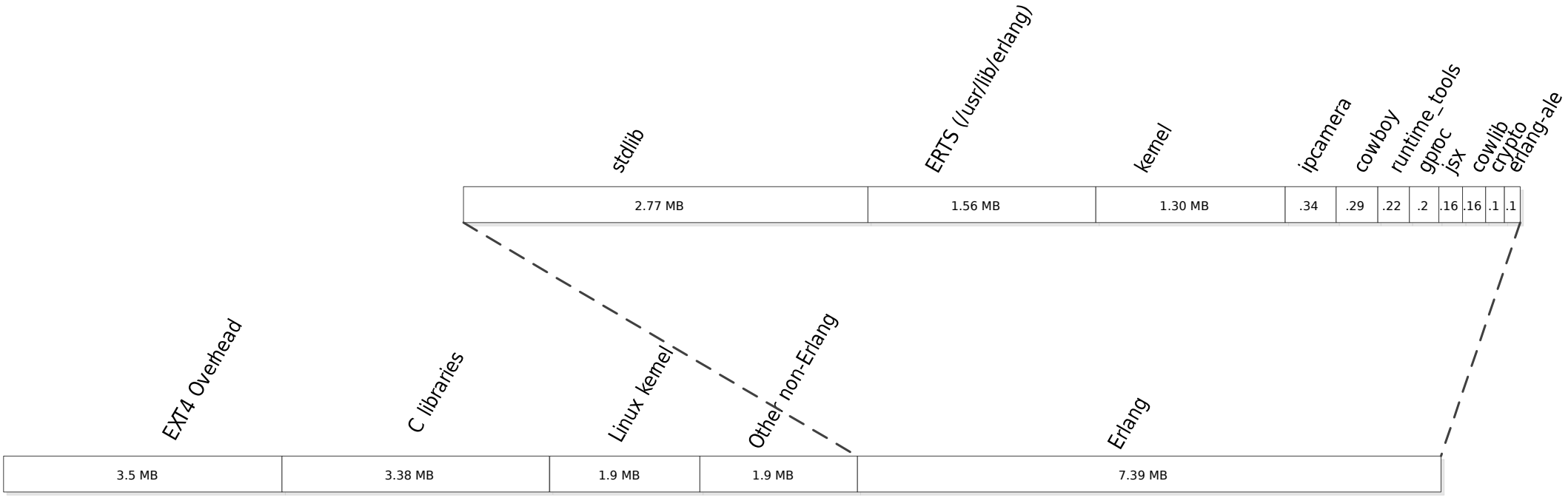


DRAM Usage (/proc/meminfo)



- Connect, stream for 5 seconds, disconnect, repeat
- erlang:memory/0 reports 7.7 MB at steady state

Footprint - 18 MB RootFS



- Erlang/OTP release tools significantly trim footprint
- Usage on par with C++ frameworks like Qt

Conclusions

- Will Erlang be too slow?
  - No. Lack of HW compression is the bottleneck as it should have been.
- Will Erlang require too much DRAM and Flash memory?
  - No. Flash footprint was on par with C/C++ frameworks. DRAM usage stable.
- Will developing Erlang in a cross-compiled environment be a pain?
  - Erlang shell + relsync can be pretty nice
- Will I miss all of the libraries and frameworks available in C?
  - Cowboy is far superior to anything I had used in C/C++
  - Still hit or miss when searching for Erlang libraries
- Will I have confidence in the Erlang/OTP platform?
  - Yes - platform feels very robust (no unexplained crashes)
  - Interface to C very easy
  - Rough edges look easily fixed with time and effort

Nerves-Project

- All source code is open source
  - Mostly MIT licensed
  - Buildroot and build scripts are GPLv2
- Upcoming
  - Hobby → Real products
  - Documentation
  - Network configuration improvements
  - Elixir!!
- http://nerves-project.org

Building an IP Network Camera

Frank Hunleth
Twitter: @fhunleth
Erlang Factory 2014