ELIXIRCONF™ 2017

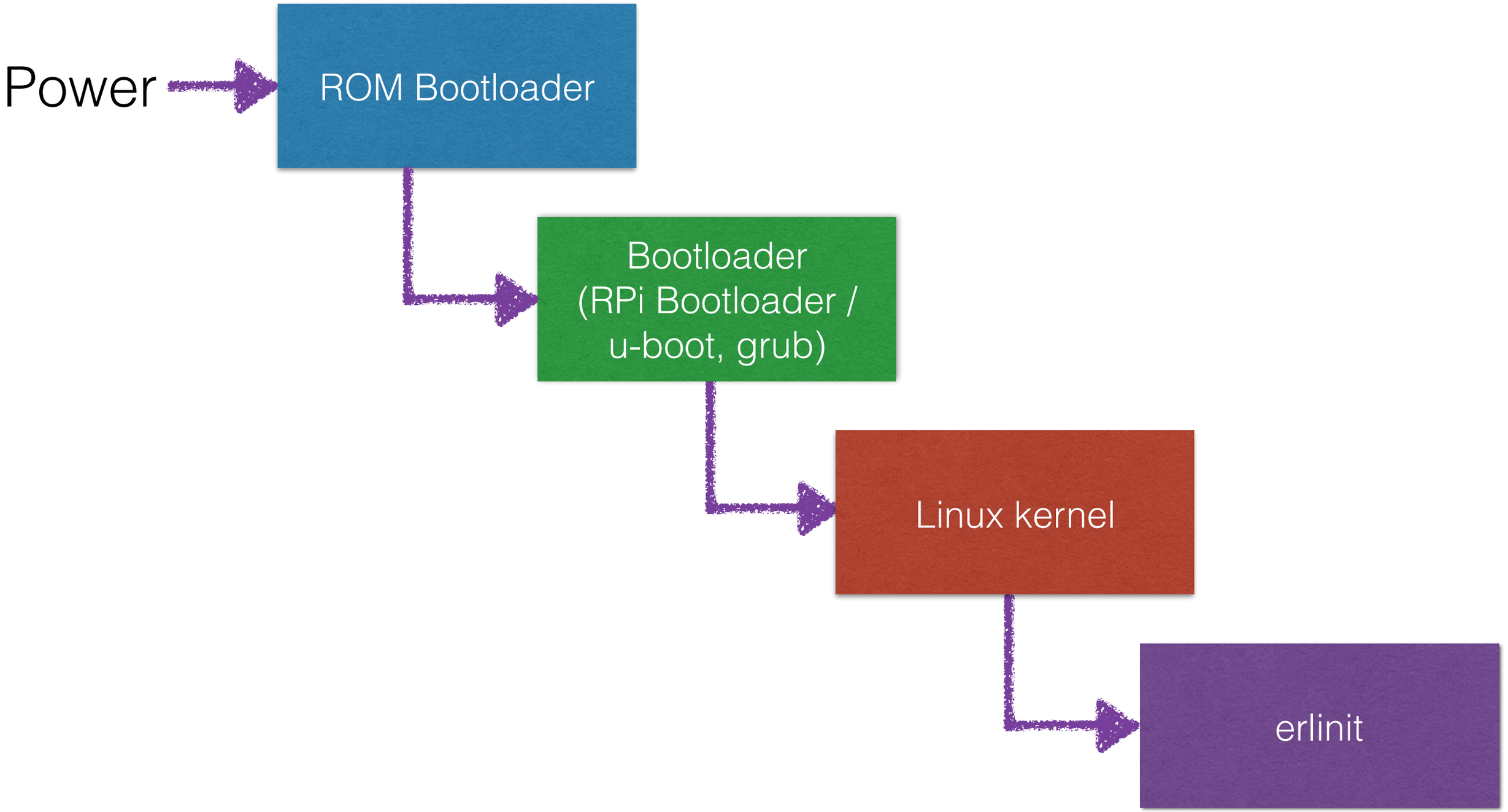
Nerves Training
Part 7: Advanced Topics

Topics

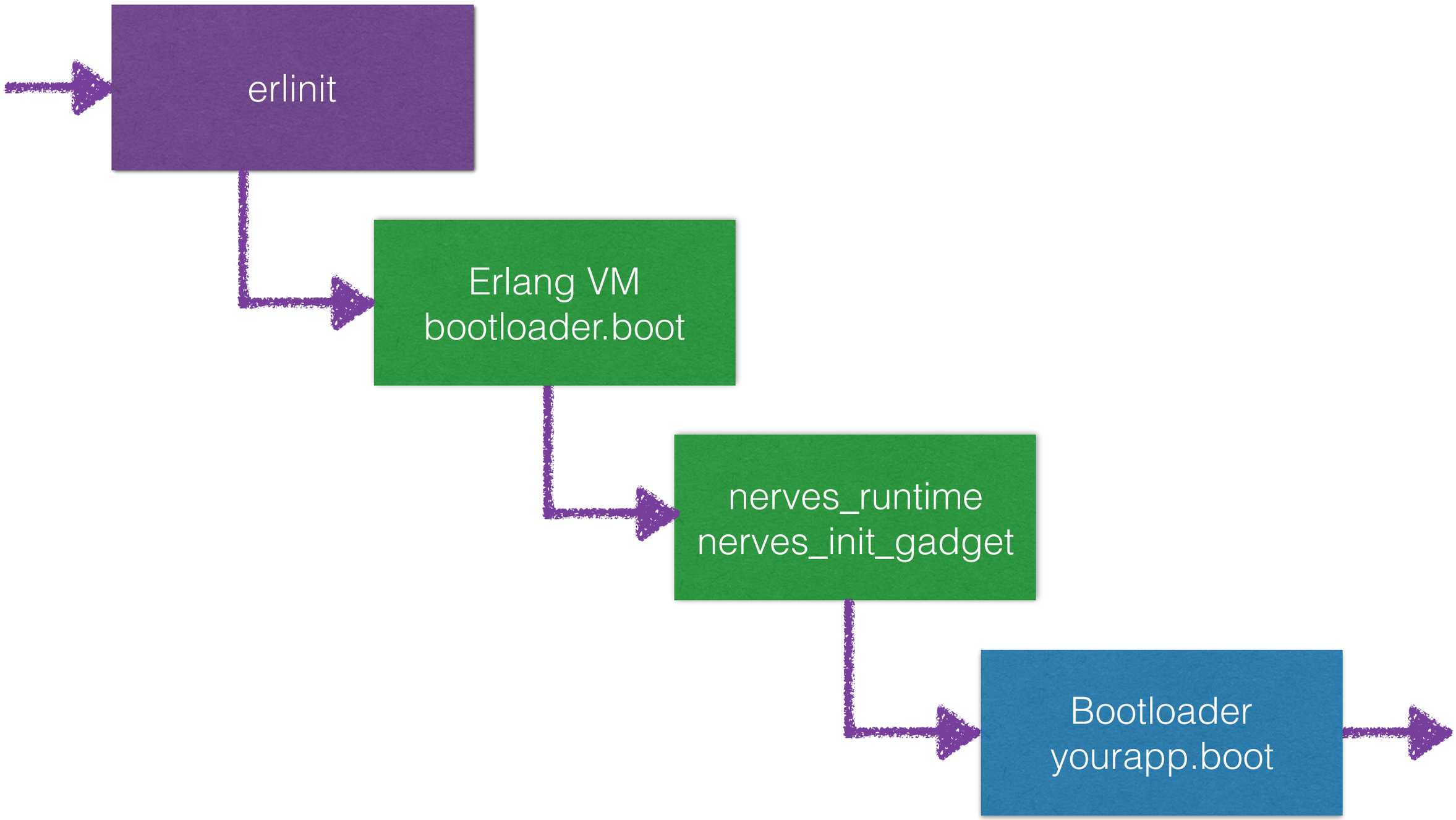
- Device initialization
- Storage layout and firmware updates
- Nerves and hard real-time
- Interfacing with C
- Security
- Going forward

Initialization

Initialization



Initialization Part 2



nerves_runtime

- Handles common initialization for all Nerves devices
- Initializes the application data partition if necessary
  - Formats as ext4
  - Mounts as /root by default
- Starts monitoring device insertion and removal

erlinit

- Replacement for /sbin/init that starts the Erlang virtual machine
- Basic initialization of the Linux user land
  - Loopback network connection
  - Mounts /tmp, /proc, /sys
  - Configures the tty
- Configuration stored in /etc/erlinit.conf
  - Can be overridden by passing parameters to the Linux kernel from the bootloader

erlinit debugging options

- --verbose
- --hang-on-exit
  - Useful to capture error messages when the VM exits
- --run-on-exit /bin/sh
  - Drop into a shell if the VM exits
  - Exiting the shell reboots or hangs
- --warn-unused-tty
  - erlinit will tell you what options to pass it to use the shell on the terminal you're looking at

erlinit features

- Mount filesystems
- Configure a unique hostname
  - --hostname-pattern and --uniqueid-exec
  - Nerves uses fhunleth/boardid to read serial numbers
- Wrap the launch of the Erlang VM in another program
  - --alternate-exec
  - Perform some custom system-specific initialization that can't be done in Erlang or Elixir
  - Capture the terminal with dtach to route it to a GUI
- Run the Erlang VM as a regular user

erlinit pitfalls

- Running shell scripts to initialize the system
  - Move initialization to Elixir to take advantage of OTP supervision
    - Be aware that Linux daemon processes don't supervise easily
- Assuming writable filesystems always can be mounted
  - Failures happen – must be handled in Elixir
  - erlinit can’t report errors so Elixir must check

Topics

- Device initialization
- Storage layout and firmware updates
- Nerves and hard real-time
- Security
- More resources

Storage

Storage Basics

Raw NAND Flash

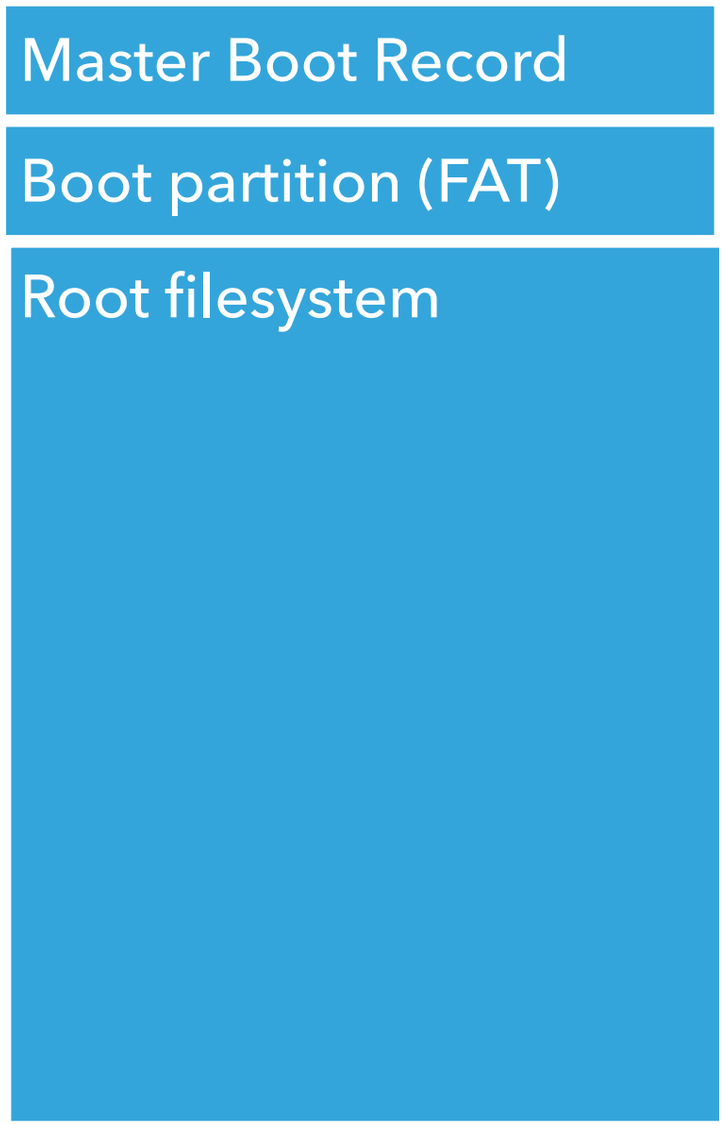
MicroSD

eMMC

- Almost all embedded devices use NAND Flash now since it's large and cheap
- MicroSD cards and eMMC contain microcontrollers that handle wear leveling, bad block handling, etc.

Basic Flash Layout

- Master Boot Record (MBR)
  - Table of contents
- Boot partition readable by ROM bootloader
  - Linux, device tree and other files
- Root filesystem
  - Almost everything under "/"
  - Normally writable

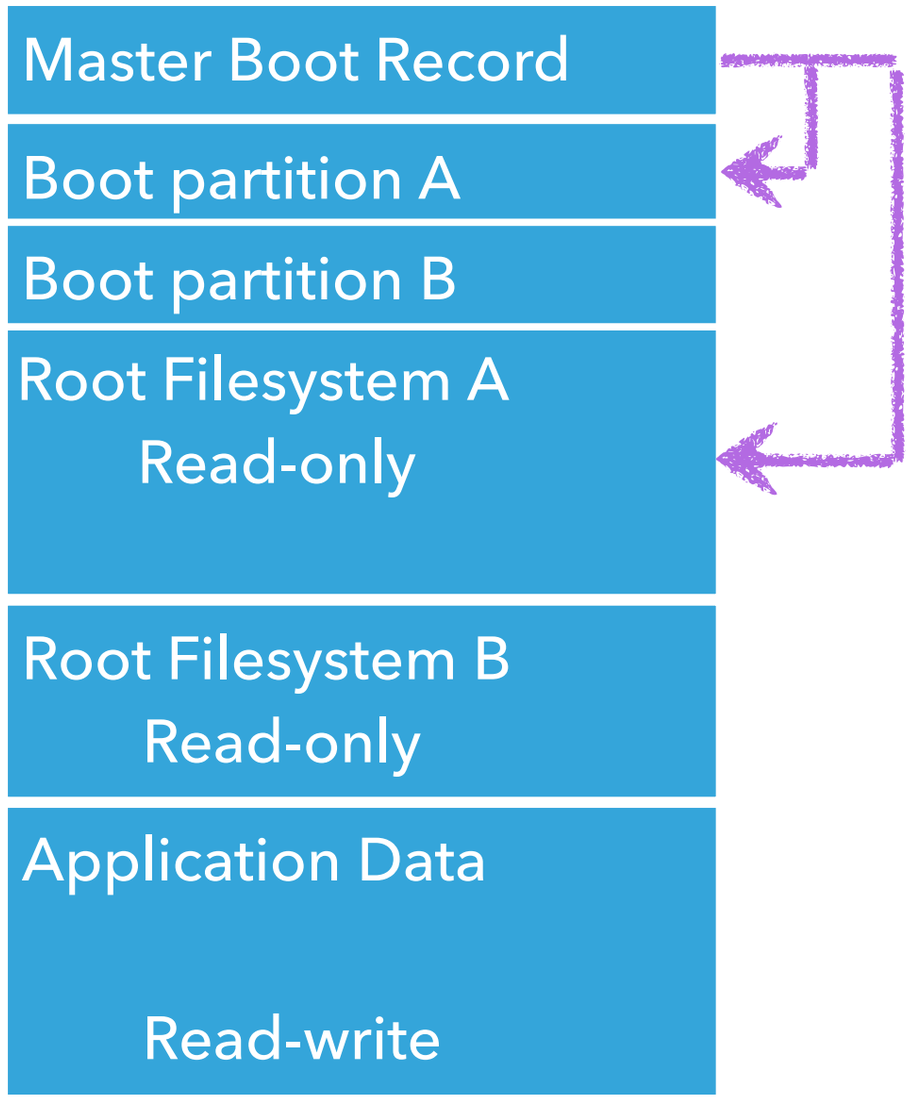


Common Filesystems

- FAT8, FAT16 and FAT32 - the filesystems that you know and love from DOS and Windows
  - Very simple and widely supported even though old tech
  - Get corrupted very easily
- ext4 - de facto Linux filesystem
  - Relatively fast and supports journals for failure recovery
  - Not so well supported outside of Linux world
- SquashFS - compressed read-only filesystem
  - Common in embedded systems and Live CDs
  - Often combined with UnionFS
- UBIFS - If you use raw NAND Flash

Flash Layout with Nerves

- Pair boot and root filesystems
  - Updates go to the unused one
  - Swap active when done
- Read-only root filesystem
- If data partition is corrupt reformatting it returns system to a usable state



Root filesystem overlay

- Needed to update files outside of Elixir
  - Linux program configuration files in /etc
    - erlinit.config
    - Workaround read-only filesystem with symlinks to the application partition
- See *config/config.exs* for example

Failure scenarios

- Root filesystem corruption due to ill-timed disk op
  - Difficult thanks to read-only filesystem
- Ill-timed power failure or crash on firmware update
  - Small window: A/B swap is final operation and nearly atomic
- Bad firmware applied
  - Cryptographic hashes to detect corruption
  - Verification of firmware metadata before applying it

Handling Faulty Firmware

- Development scenarios
  - Use Elixir bootloader to initialize everything needed for firmware updates
  - Take advantage of nerves_network configuration cleanup when processes die

Handling Faulty Firmware 2

- Verify and failback
  - Allow one boot of new firmware
  - If "valid" bit isn't set, then next reboot reverts to old firmware
- Safe mode
  - Dedicated partition that runs a firmware update application
  - Detect button press (or other input) at start to enter this mode

fwup

- Firmware update packaging and application
- Packages
  - Zip-formatted archives
  - Metadata
  - All data files protected by cryptographic hashes
  - Packages can be cryptographically signed
  - Very simple scripts supported (lack of functionality is a feature)

fwup processing

fwup processing

.fw file

on_init 1

rootfs.img 2

zImage 3

other files 4

on_finish 5

Master Boot Record

Boot partition A

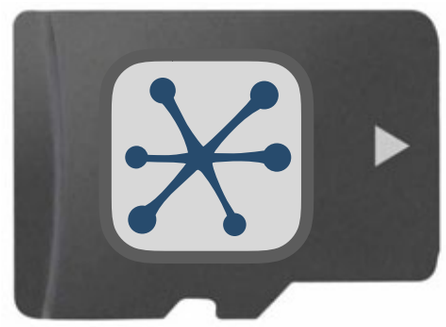
Boot partition B

Root Filesystem A
Read-only

Root Filesystem B
Read-only

Application Data

Read-write



Reasons to Modify fwup.conf

- Raspberry Pi bootloader configuration changes
  - New device tree overlays for HATs
    - Change UART config, GPU parameters, etc.
    - Change Linux kernel parameters
- Implement failback and safe boot
- Modify root filesystem and data partition sizes

Nerves Firmware and System Metadata

Nerves.Runtime.KV

- Simple key-value store for system and firmware metadata
- Not intended to be written frequently
- Requires Nerves system support
  - U-boot formatted kv store not in any partition
  - fwup.conf can modify keys
  - Requires uboot-tools (*fw_printenv* and *fw_setenv*)

Nerves.Runtime.KV Uses

- Information about running firmware
  - Useful for bug reports and new release detection
  - Verify firmware update compatibility (versions, architecture, product, etc.)
- Device provisioning information
  - Calibration
  - Manufacturing build configuration

Try it out

```
iex> Nerves.Runtime.KV.get_all_active
%{"nerves_fw_application_part0_devpath" => "/dev/mmcblk0p3",
  "nerves_fw_application_part0_fstype" => "ext4",
  "nerves_fw_application_part0_target" => "/root",
  "nerves_fw_architecture" => "arm",
  "nerves_fw_author" => "The Nerves Team",
  "nerves_fw_description" => "",
  "nerves_fw_misc" => "",
  "nerves_fw_platform" => "rpi0",
  "nerves_fw_product" => "starter",
  "nerves_fw_vcs_identifier" => "",
  "nerves_fw_version" => "0.1.0"}
```

Works on any of the training examples

Nerves and Hard Real-time

Real-time

- How critical is response time to the correctness of a program?
- Real-time systems provide guarantees
  - Hard real-time - Deadlines can never be missed
  - Soft real-time - Missed deadlines are undesirable, but not fatal
  - Not real-time - No timeliness guarantees are made

Cooperative Scheduling

- Any system using an event loop
  - Node.js
  - Python Twisted
  - C++ Qt framework
- No guarantees that a long running computation doesn't stall the event loop
- Does that 3rd party package I just pulled in behave nicely?
- Not a real-time system

Scheduling in Elixir

- Preemptive multi-tasking
- Each process gets a fixed number of “reductions”
    - Function calls
    - Garbage collects
    - Sending messages
- Processes are scheduled round robin
- Four queues (:low, :normal, :high, :max) but priorities other than :normal are rarely used
- Soft real-time

Real-time Operating Systems

- Preemptive multi-tasking
- Often priority based schedulers, but other options available
- Can handle priority inversion (when a high priority process needs something from a low priority one)
- Run in limited resource environments
- Fewer features than general purpose operating systems
- Support hard real-time applications
- See RTEMS usage with GRiSP

Hard Real-time Tasks

- Collecting data from streaming sensors
- Sensor/actuator feedback loops
- Low level communication with hardware devices (e.g., LED strips)
- Example: Accelerometer
  - Acceleration data reported at 100+ Hz
  - If sample missed, it's lost - no retransmissions
- Example: PID motor controller
  - Feedback from RPM sensor used to control current to a motor
  - Control algorithm tuning depends on algorithm w/ precise timing

Mix of tasks

Hard real-time ⟷ Soft real-time

Touchscreen UI

Data collection

Web interface

Control logic

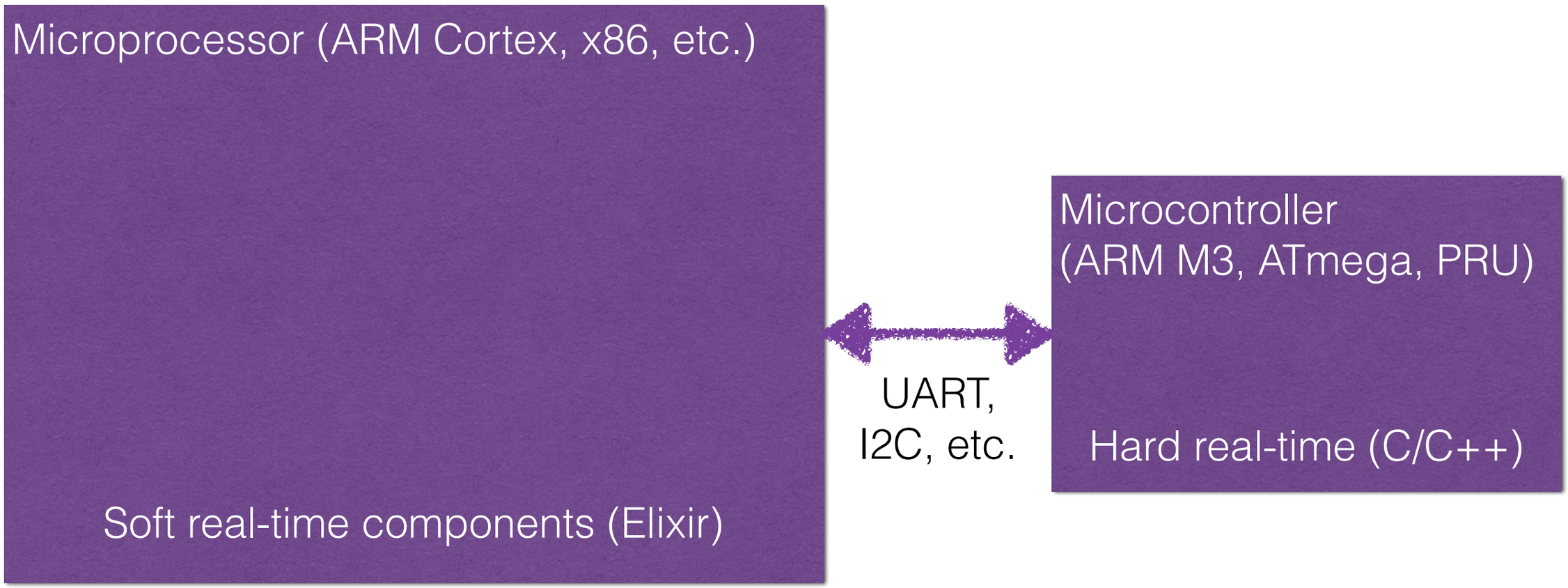
Software update

Actuator control

System configuration

Diagnostics

Common Architecture



Segregating hard real-time tasks can simplify both verification of real-time constraints and overall system complexity.

Interfacing with C

Elixir is not a Systems Programming Language

- Can't read or write to special files like device files
- No ioctl()
- No mmap()
- Low level OS parts of C standard library generally missing (e.g., signals)

Not really a problem, since providing this many of these features inside the Erlang VM might compromise soft real-time and stability

Interfacing to C (ports)

- Start a program and communicate via stdin/stdout
- OS process isolation from the Erlang VM
- Debug standalone or via calls from Elixir
- erl_initerface provides some help with marshaling and unmarshaling Erlang terms in C (optional)

Example Port Protocol

message_size | {request, args} → To port process

```
msg = {request, args}
send myport, {self, {:command, :erlang.term_to_binary(msg)}}
```

message_size | return_value ← From port process

```
receive do
  {_, {:data, return_value}} ->
    result = :erlang.binary_to_term(return_value)
    Logger.debug "Got #{inspect result}"
end
```

Queuing and Back Pressure

- Without back pressure, requests queue between Elixir code and the port process

- Easy solution: wait for a response for each request

Limitations

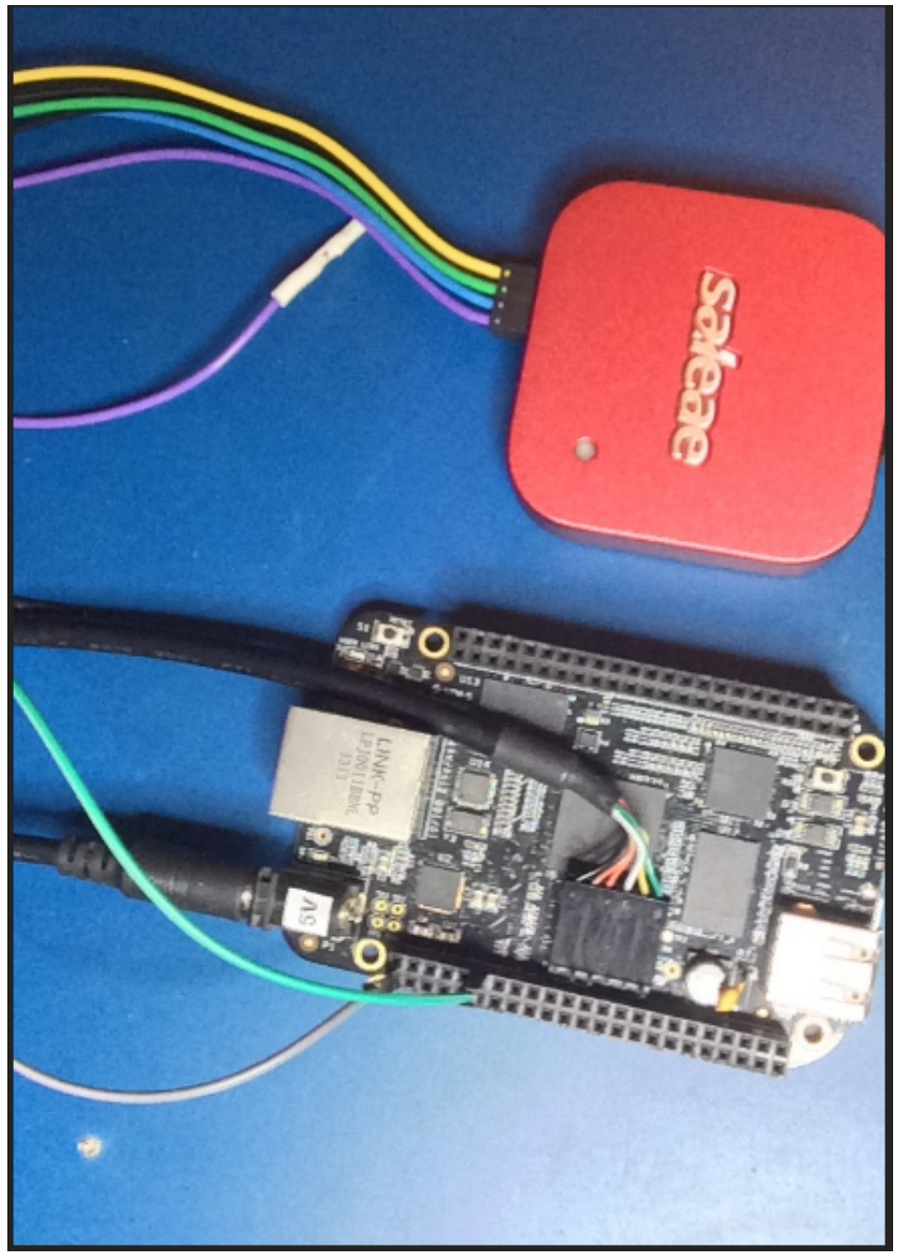
- Erlang ports lack common operations on child processes
  - Can't send EOF - affects nearly all Unix data processing programs
  - Can't send signals
- Processes scheduled by the OS scheduler (not Erlang's)
- Misbehaving C processes can easily stick around even when Erlang port

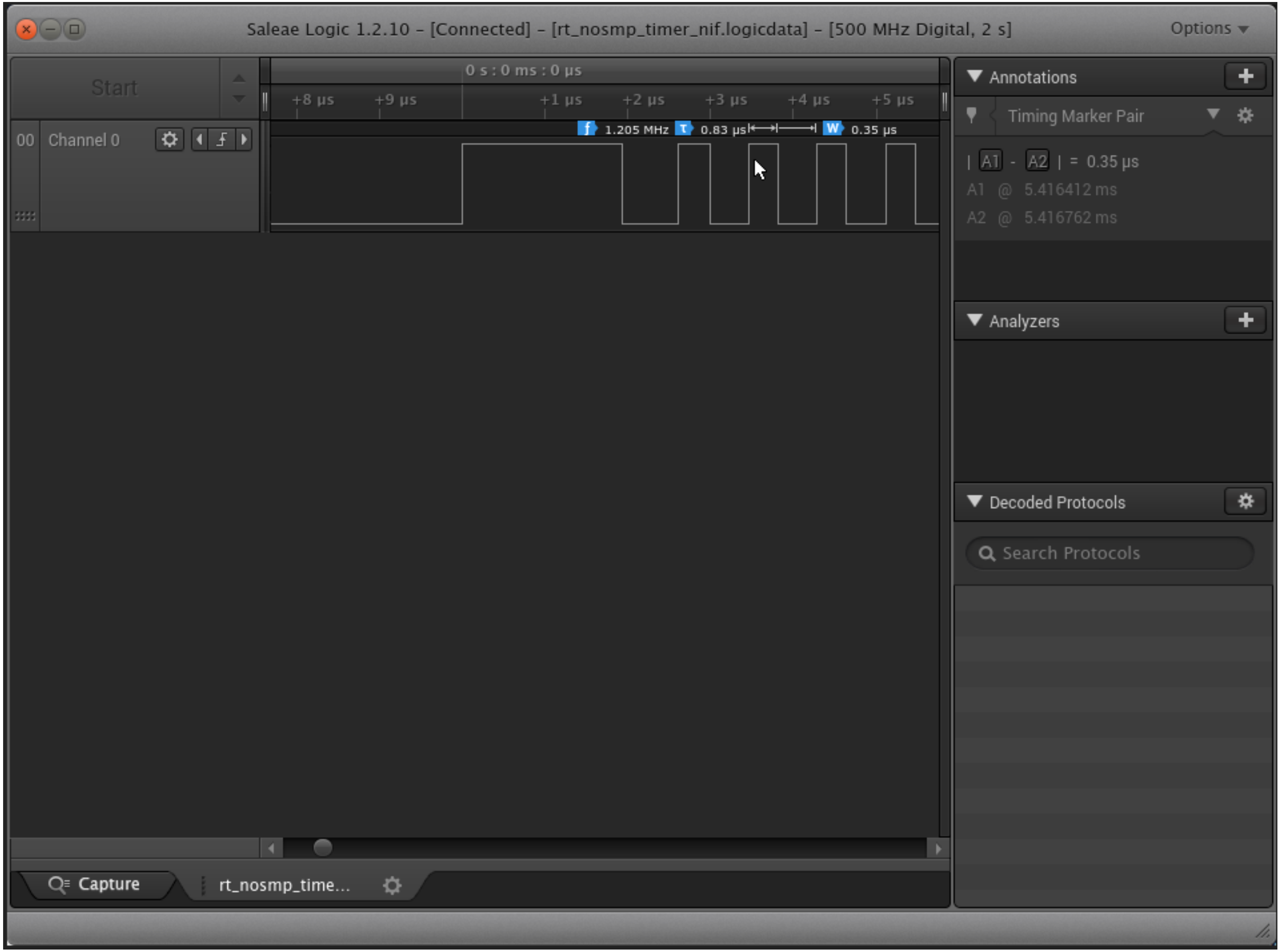
Interfacing to C with NIFS

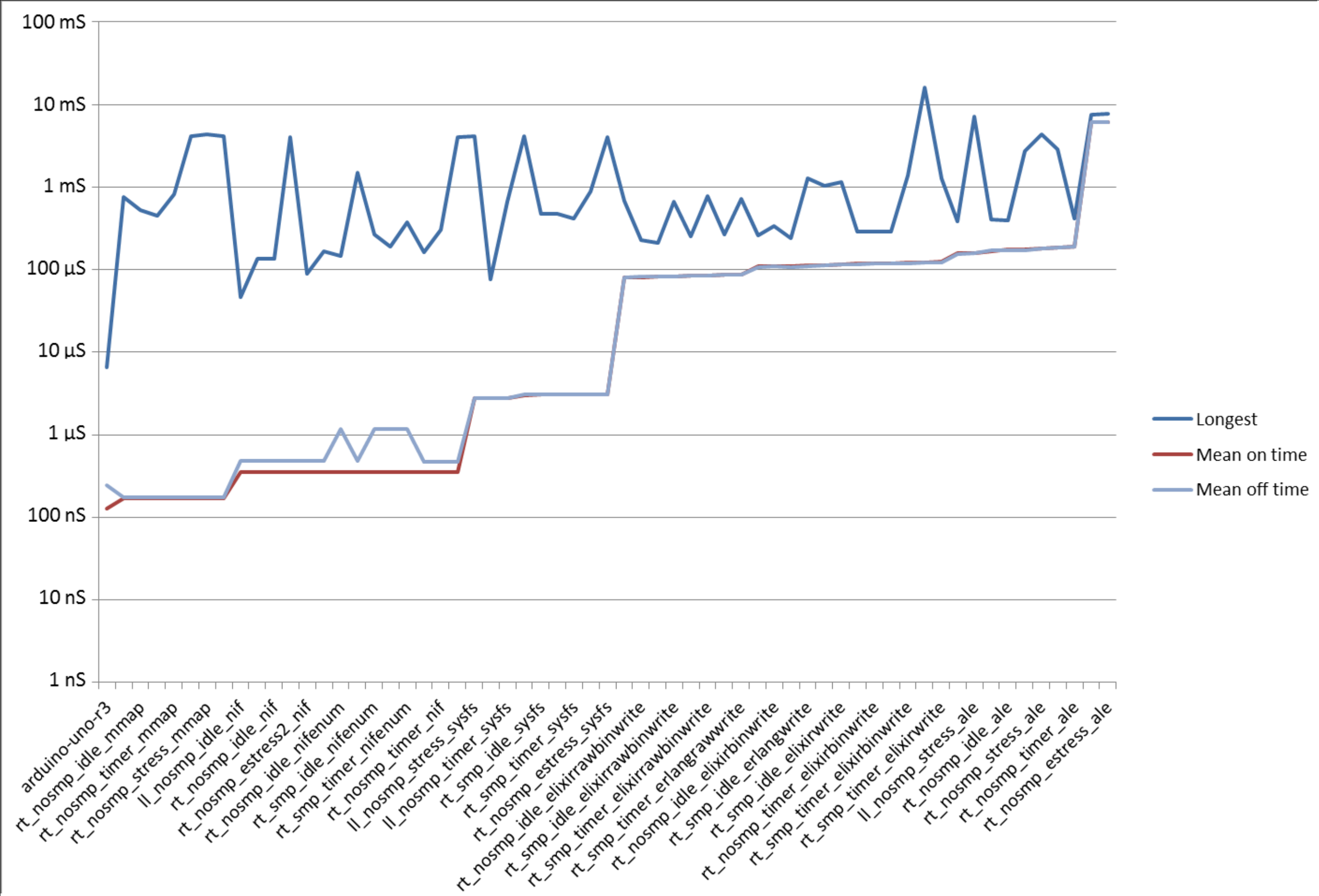
- Dynamically link a shared library into the Erlang VM
- Function callback interface in C
- Much faster than port processes
- Scheduled by the Erlang VM and run in Erlang's process environment

Experiment

- Turn on and off an GPIO as fast as possible on a BeagleBone Black (1 GHz)
- Use a logic analyzer to measure performance
- Source code at https://github.com/fhunleth/gpio_twiddler



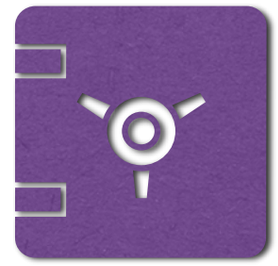

Saleae Logic 1.2.10 - [Connected] - [rt_nosmp_timer_nif.logicdata] - [500 MHz Digital, 2 s]
Options
Start
0 s : 0 ms : 0 µs
+8 µs
+9 µs
+1 µs
+2 µs
+3 µs
+4 µs
+5 µs
00
Channel 0
1.205 MHz
0.83 µs
0.35 µs
Annotations
Timing Marker Pair
| A1 - A2 | = 0.35 µs
A1 @ 5.416412 ms
A2 @ 5.416762 ms
Analyzers
Decoded Protocols
Search Protocols
Capture
rt_nosmp_time...



See https://github.com/fhunleth/gpio_twiddler

Security

IoT Security



- Incredibly important aspect of projects today
- Developer convenience at odds with security
- Nerves is not immune to vulnerabilities

Helpful Nerves features

- Starts small
  - Official Nerves systems have few Linux applications
  - Distillery only includes necessary OTP applications
- Read-only root filesystem*
- Image-based firmware updates
- Erlang VM and Elixir language
- Tracks Buildroot for security patches

*SquashFS supports appending to filesystems to add files

Be aware

- Everything in Nerves runs as root (change in *erlinit.config*)
- IEx console available if you have physical access (change in *erlinit.config* or *vm.args*)
- Erlang distribution is not secure by default
- `os:cmd/1` and ports let you get to C and Busybox easily
- No segregation of code in an Erlang VM
- Do not store secrets in firmware bundles (provision on devices)
- No support for processors that have a secure boot feature

Security Takeaways

- Elixir and Nerves aren't magic bullets on security
- Standard recommendations apply
  - Reduce attack surface
  - Review library and application dependencies
  - Keep up-to-date with security patches
  - Secure your firmware update mechanism
  - Be mindful of transmitting and storing secrets

Going forward

Nerves @ ElixirConf 2017

Nerves-day (aka Thursday)

9:10 AM - 10:10 AM	Justin Schneck	Keynote
Evergreen E 10:40 AM - 11:20 AM	Boyd Multerer	Elixir Native UI
Grand 11:25 AM - 12:05 PM	Tim Mecklem	Building an Artificial Pancreas with Elixir and Nerves
Evergreen E 1:30 PM - 2:10 AM	Chris Coté	Embedded Elixir for Monitoring the Built Environment
Grand 4:10 PM - 4:50 PM	Jeff Smith	Keep an Eye on the Sky with Nerves and Phoenix

Stay in touch!

- I'm @fhunleth on Slack, Twitter, Github and pretty much everywhere else
- Twitter: @NervesProject or #NervesProject
- Stop by say hi to the Nerves team in between talks
- Commercial support? Email me at fhunleth@troodon-software.com

ELIXIRCONF™ 2017

Thanks!

ELIXIRCONF™ 2017

Nerves Training
Part 7: Advanced Topics