Erlang and RTEMS

Embedded Erlang, two case studies

Peer Stritzinger

Talk at Erlang Factory Light Munich 2013

Outline

- Case 1: Hydraprog-3, Reflashing system for Automotive "Embedded Control Units" (= ECUs)
    - Big picture
    - Erlang code structure
    - Lessons learnt
- Case 2: Intelligent RFID Antenna
    - Porting Erlang on RTEMS (work in progress)


hydraprog3
hydraprog3


hydraprog3


Hydraprog 3
Net 1
Net 2
Net 3
Main Pwr


X1
X2
HP3 GW-LCFM
X3

Automotive Protocols

- LIN ✓
- LS-CAN ✓
- HS-CAN ✓
- FlexRay ✓
- MOST ✓
- OABR (Ethernet with special Phys. Layer) (... in development)
- APIX1+2 ✓ - Ethernet via APIX (... in development)

Automotive Protocol Properties

- Flash process in most cases defined over Diagnostic protocol
    - Ping-Pong protocol => latency issues
- Mapping to lower level: usually bursty with timing/buffer negotiations
    - realtime requirements: semi-hard-realtime to really-hard
- Big variety of link and physical layers

Mainboard
Mini ITX
Intel Atom

FreeBSD, Erlang

Gateways
custom
Freescale
MPC5517

RTEMS

Ethernet

USB

USB

Ethernet Switch / USB Hub

FreeBSD

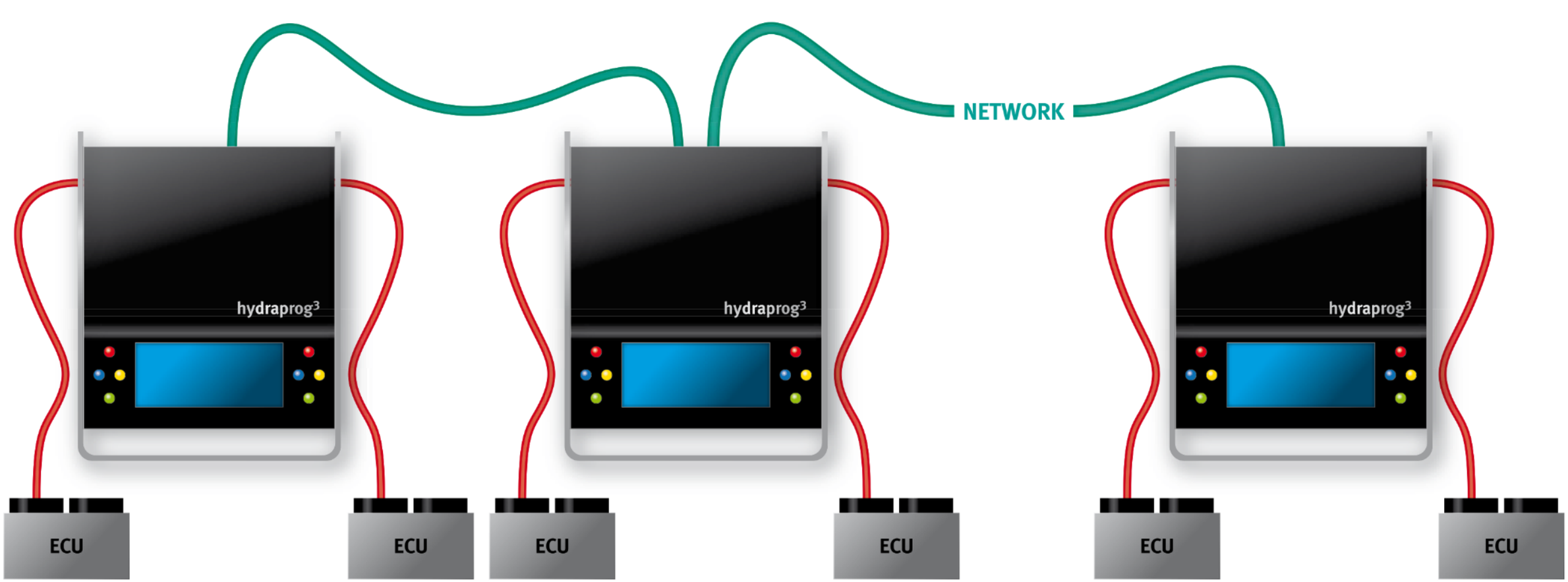
- Boots from read-only CF-card partition
- 2 Partitions for upgrade
- Partition for config storage copied to /etc in ramdisk
- Extra partition for logging
- Built with NanoBSD script

Erlang

- Runs as much of the automotive protocols as possible
- Protocol layers stackable by user config
- Uses libusb in a port to talk to the gateways
- USB bus is scanned regularly
- Supervised protocol stacks are started for each gateway

Distributed Device

- Multiple devices self network
- Behave like one device with more channels



RTEMS

- Hard-Realtime embedded executive
- Small core with several APIs
- Posix API
- TCP/IP stack
- Several schedulers

RTEMS cont.

- About the same age as Erlang
- Also very robust
- Runs in many satellites and planetary probes
- Industrial and Automotive applications
- Great support in Europe: www.embedded-brains.de

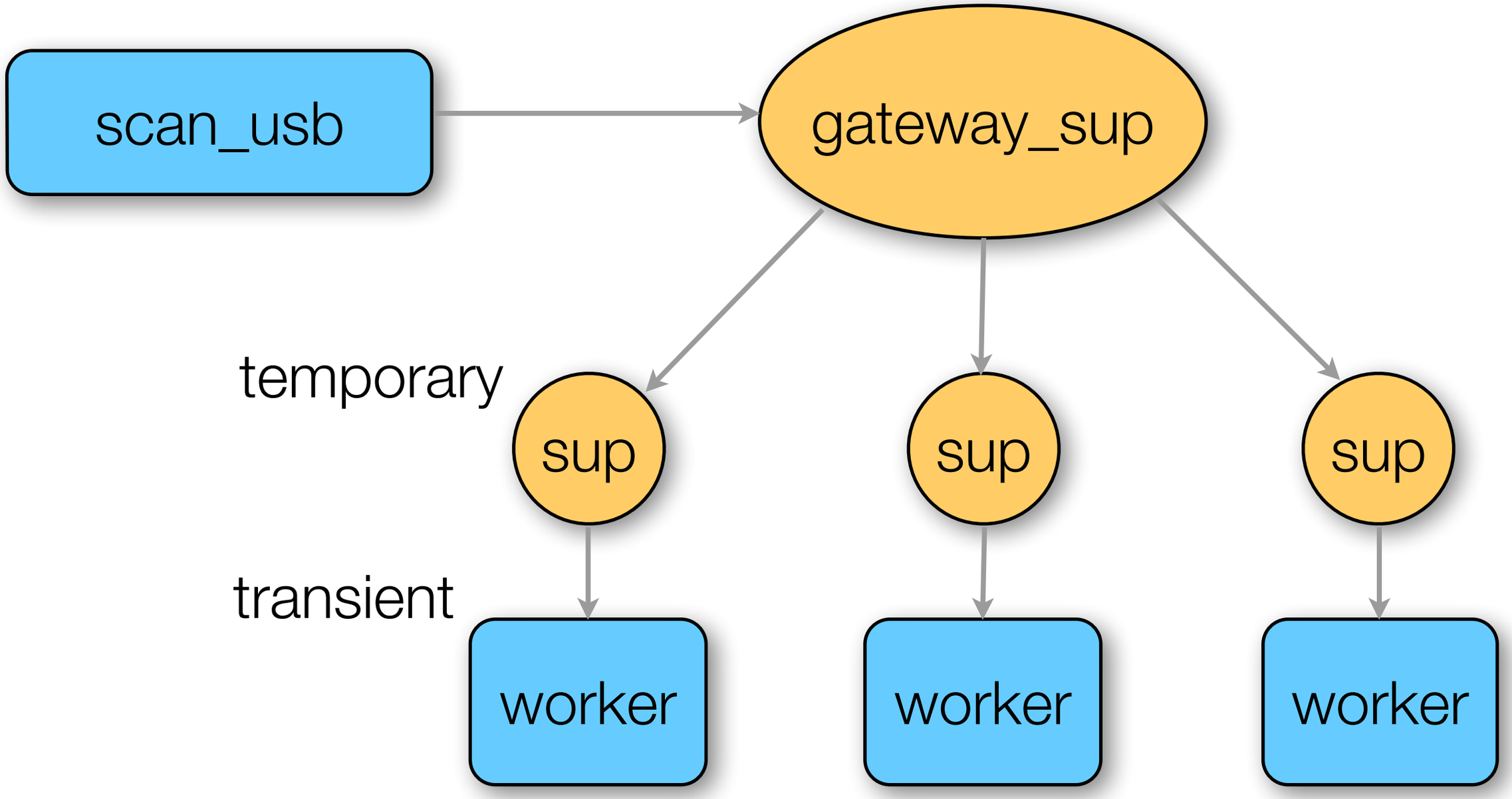
RTEMS in Hydraprog-3 Gateways

- Gets commands and data packets from USB
- Controls and switches power to the connected ECUs
- Protocol parts that require hard realtime or low latency
- Talks to Low-Speed CAN, High-Speed CAN, FlexRay, MOST, LIN
- Controls the display LEDs

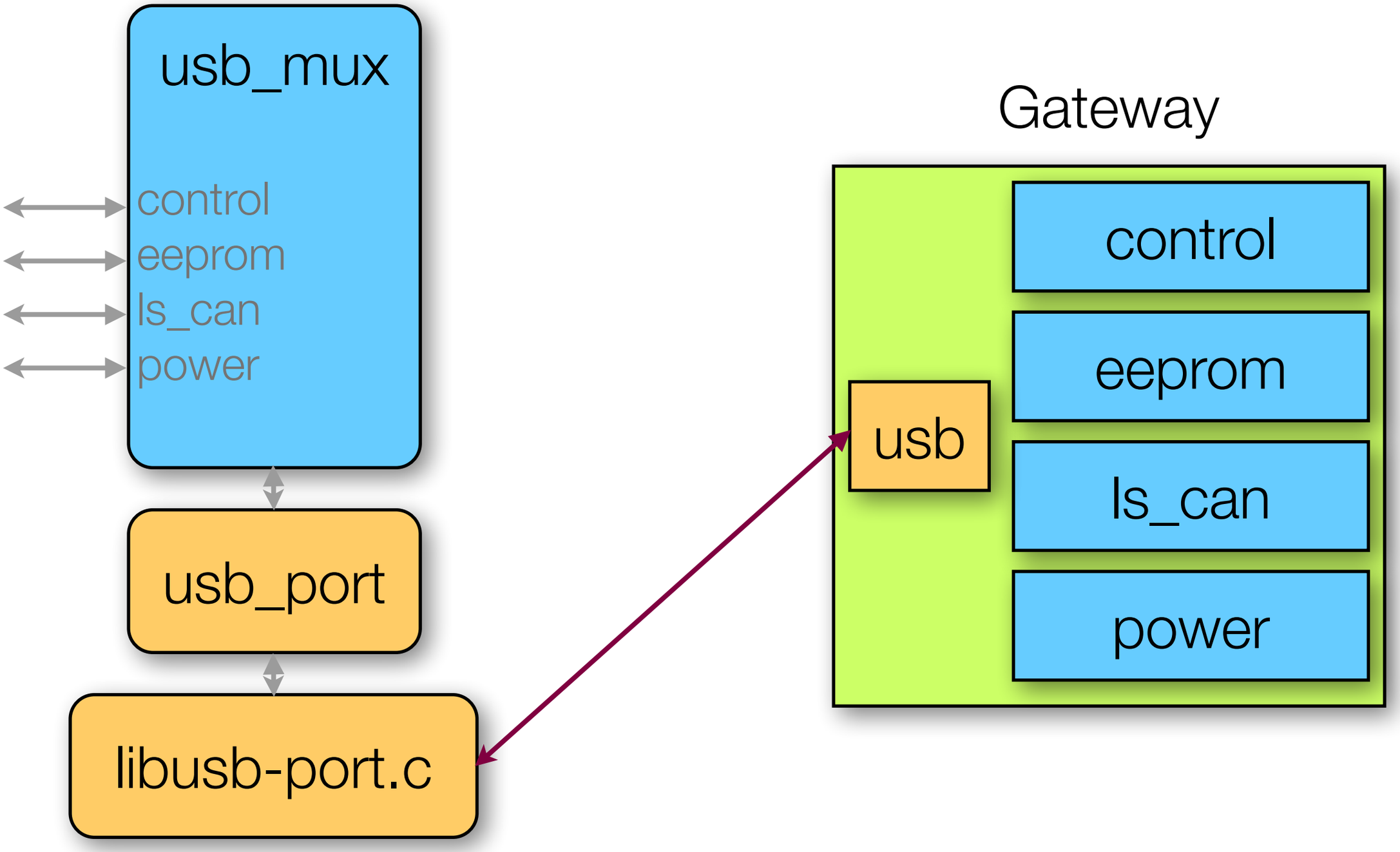
Talking to (internal) USB Devices

- C executable + libusb via port
    - Easy error handling
- Problem: USB device enumeration doesn't tell you what is where
    - Solution: Id + Serial number + Hw-Config file
- Also: USB is weird
    - But bulk endpoints are easy once they are established

USB Hot-Plugging



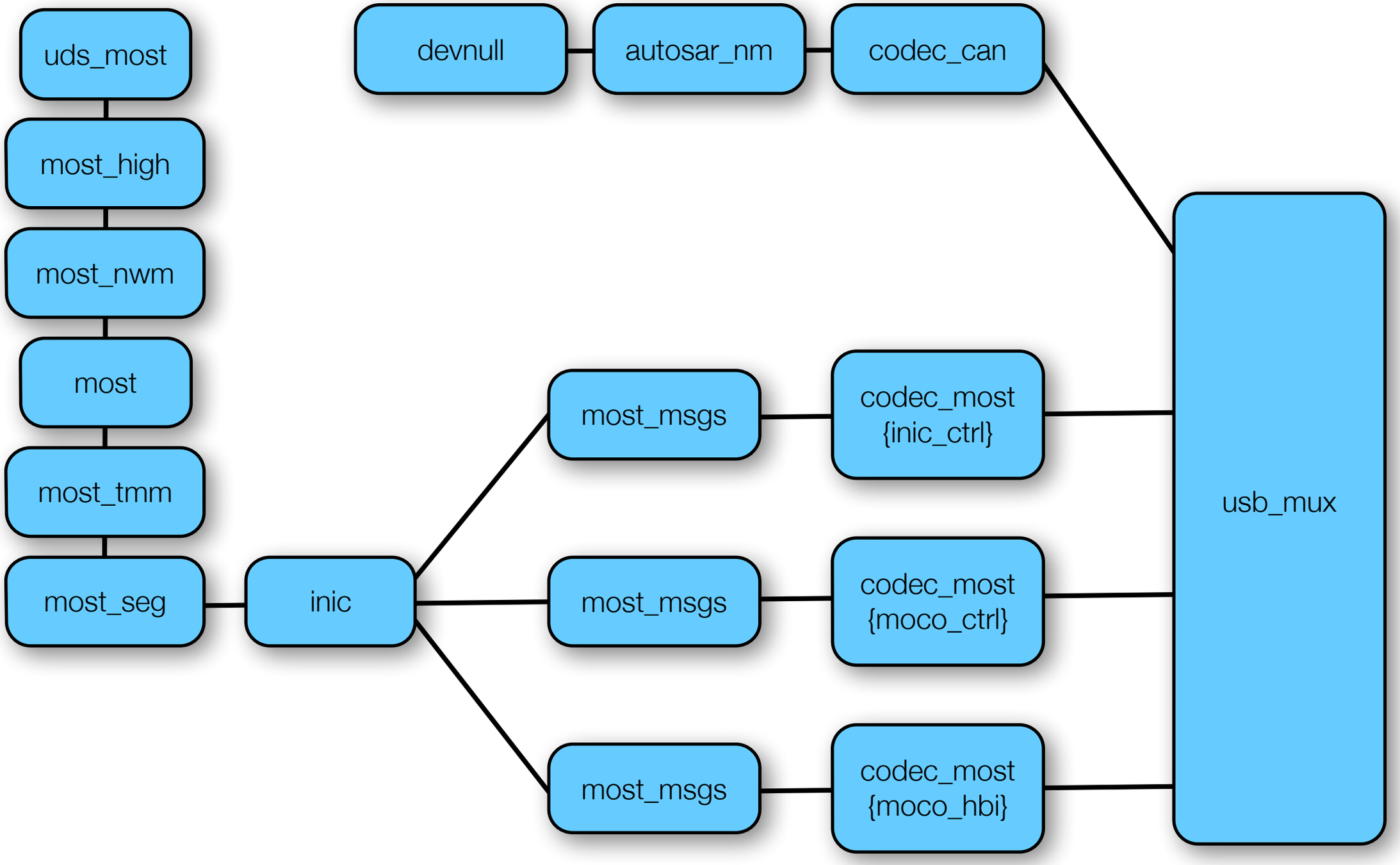
Gateway queue handling



Protocol Layer Plumbing

- Dynamic configuration during startup
- Minimize latency
- Differing state requirements
  - Stateless -- encoders/decoders
  - Configuration state
  - Runtime state

Example: MOST Stack

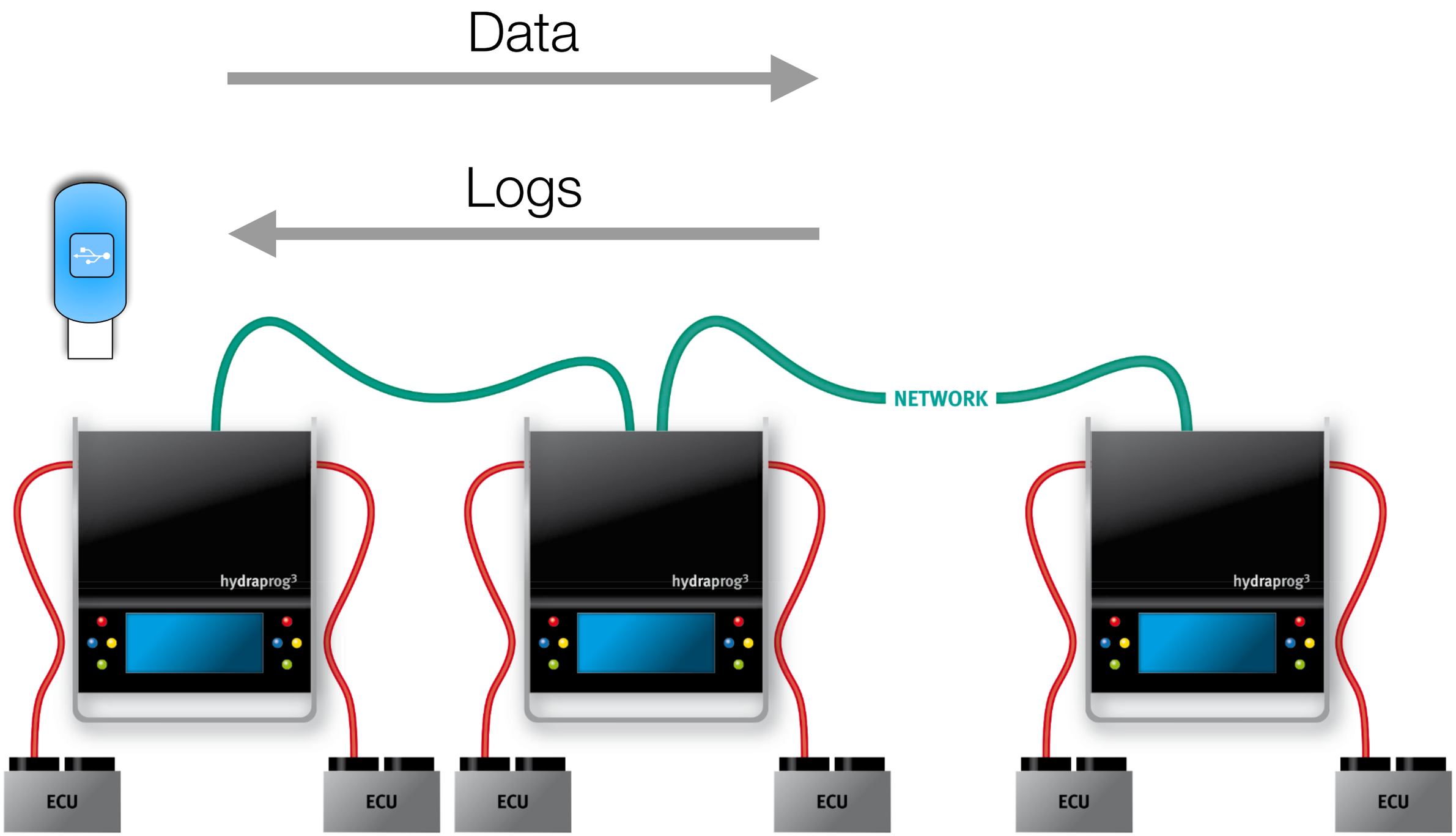


Example: MOST Stack

- Dynamic config: DAG like combination of modules

```
[[[uds_most, most_high, most_nwm, most, most_tmm,
                most_seg, inic,
                [[most_msgs, {codec_most, [inic_ctrl, simple]},
                  {usb_mux, ['Usb_mux', inic_ctrl]}],
                 [most_msgs, {codec_most, [moco_ctrl, simple]},
                  {usb_mux, ['Usb_mux', moco_ctrl]}],
                 [most_msgs, {codec_most, [moco_hbi, simple]},
                  {usb_mux, ['Usb_mux', moco_hbi]}]]],
               [devnull, autosar_nm, {codec_can, []},
                {usb_mux, ['Usb_mux', hs_can]}]]]}
```

Protocol-Stack config and Flash-Data Distribution



Signed Autoexec Erlang App

*.h3x

signed & encrypted

Erlang app config

Auth keys

extra modules

*.zip

User Data
+ Script

Scripting "language"

- Re-used Erlangs parser
- Interpreting the abstract syntax
- Different, domain specific semantics

Reducing Latency

- Send - Expect Engine in Gateway
    - Complicates protocol implementation
    - Enables streaming + optimal performance
- Libusb support as port driver
    - High stability requirements
    - Initially on separate nodes

Lessons Learnt

- Use gproc next time right away
    - Startup was messier than it needed to be without it
- Use Quickcheck for testing right away
- Start doing OTP Releases earlier

Erlang and RTEMS in Hydraprog-3

- Collaborates on different CPUs
- Best of both worlds
- Solution for larger devices
- More on http://www.stritzinger.com

What about Hard Realtime?

- Erlang has very good soft realtime responsiveness
- Embedded applications often need hard realtime
- Erlang alone can't do this
- Especially on "normal" operating systems
- Usually only small part required to be hard realtime

Erlang Running on RTEMS

- Small embedded system
- Only a small communication controller
- Freescale MPC8309
- Still want best of both worlds
- RTEMS Posix API has almost everything Erlang needs

Porting Erlang to RTEMS

- Work in progress, at the moment (Feb 2013)
- Erlang shell on RS232 console
- Linked in drivers
- Cross-built with otp_build using RTEMS build tools
- RTEMS gets linked as a library
- File access on target via NFS

In Progress

- Get distribution working (epmd needs porting or faking)
- Build Erlang as loadable application with new RTEMS Linker
- Threads for async I/O
- Support for NIFs with new dynamic linking support
- Adding kqueue support to RTEMS

Future

- Support for some RTEMS primitives from Erlang
- Get all necessary patches back to RTEMS and OTP
- Have a documented standard way to build Erlang/RTEMS
- Once RTEMS gets stable SMP, support it from Erlang

Questions

- Contact: peer@stritzinger.com
- Twitter: @peerstr
- IRC: peerst