ETH zürich

ENZIAN

Systems@ETH zürich

DINFK

Enzian Firmware Resource Interface

Pengcheng Xu
Daniel Schwyn, Prof. Dr. Timothy Roscoe
Systems Group @ ETH Zürich

Acknowledgements

- The speaker would like to thank the Enzian team, including (but not limited to):

  David Cock, Timothy Roscoe, Daniel Schwyn, Michael Giardino, Ben Fiedler, and many more

- For their great help:
  - Discussions and feedback on the core design
  - Practical help with implementation, development and debugging
  - Constructive comments on the project report
  - Access to previous slide decks for building this presentation
  - Feedback on the presentation dry-run
  - …

Outline

ENZIAN Systems@ETH zürich DINFK


Introduction & Motivation


Design


Implementation


Evaluation


Live Demo

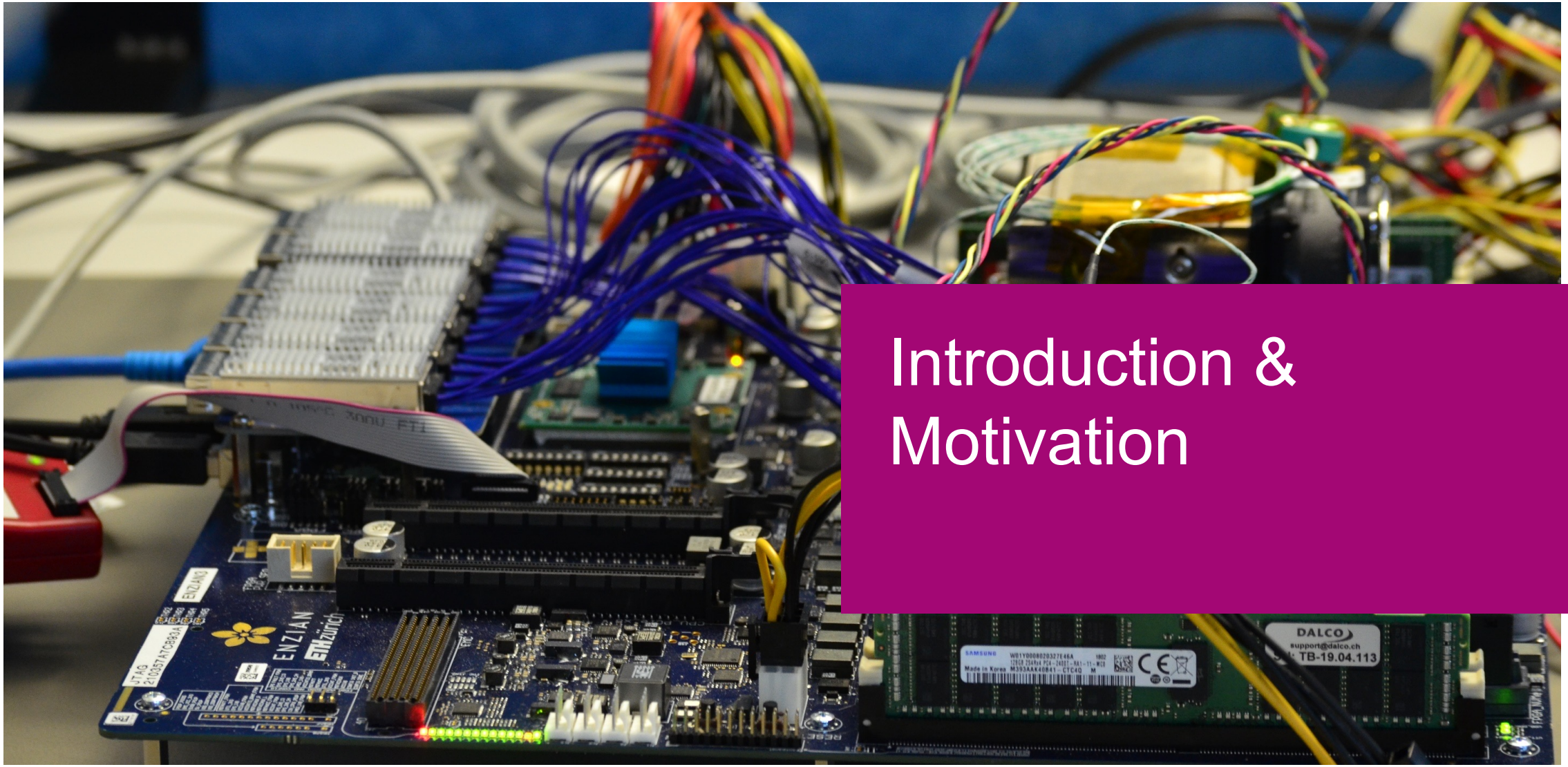

Discussion

ETH zürich

ENZIAN

Systems@ETH zürich

DINFK

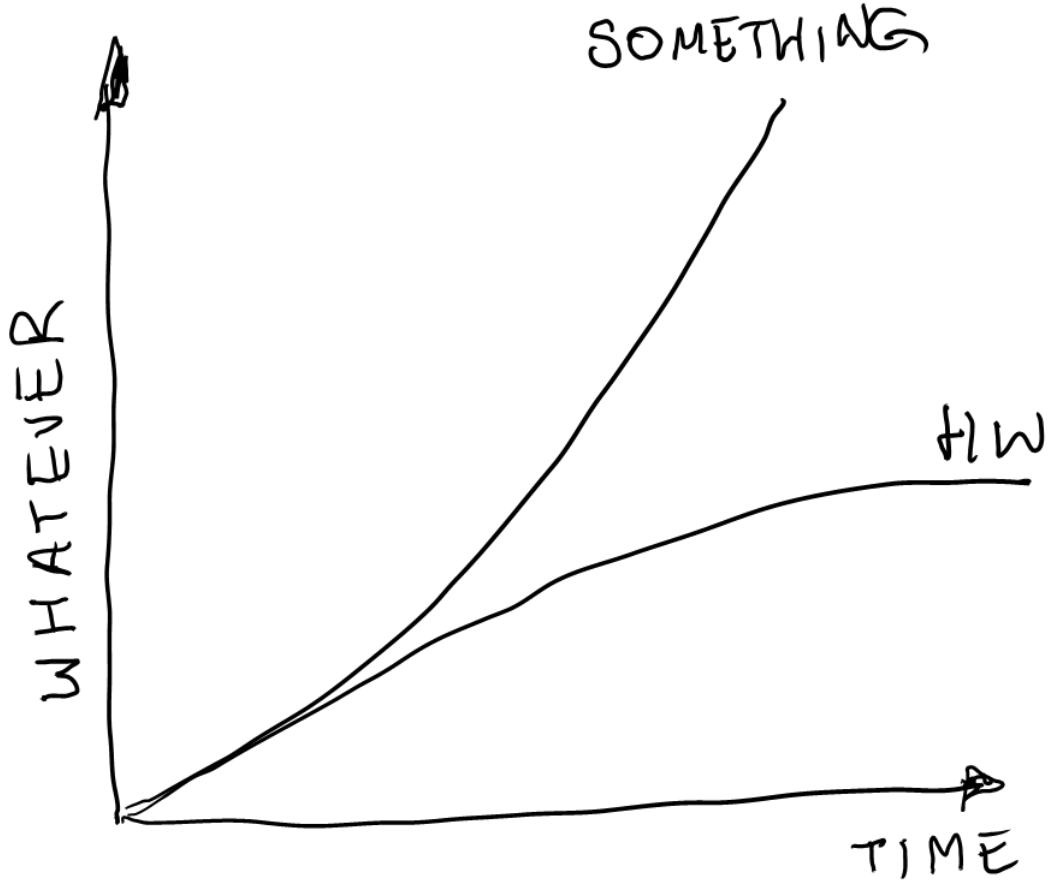


Introduction & Motivation

The usual bla, bla, bla …

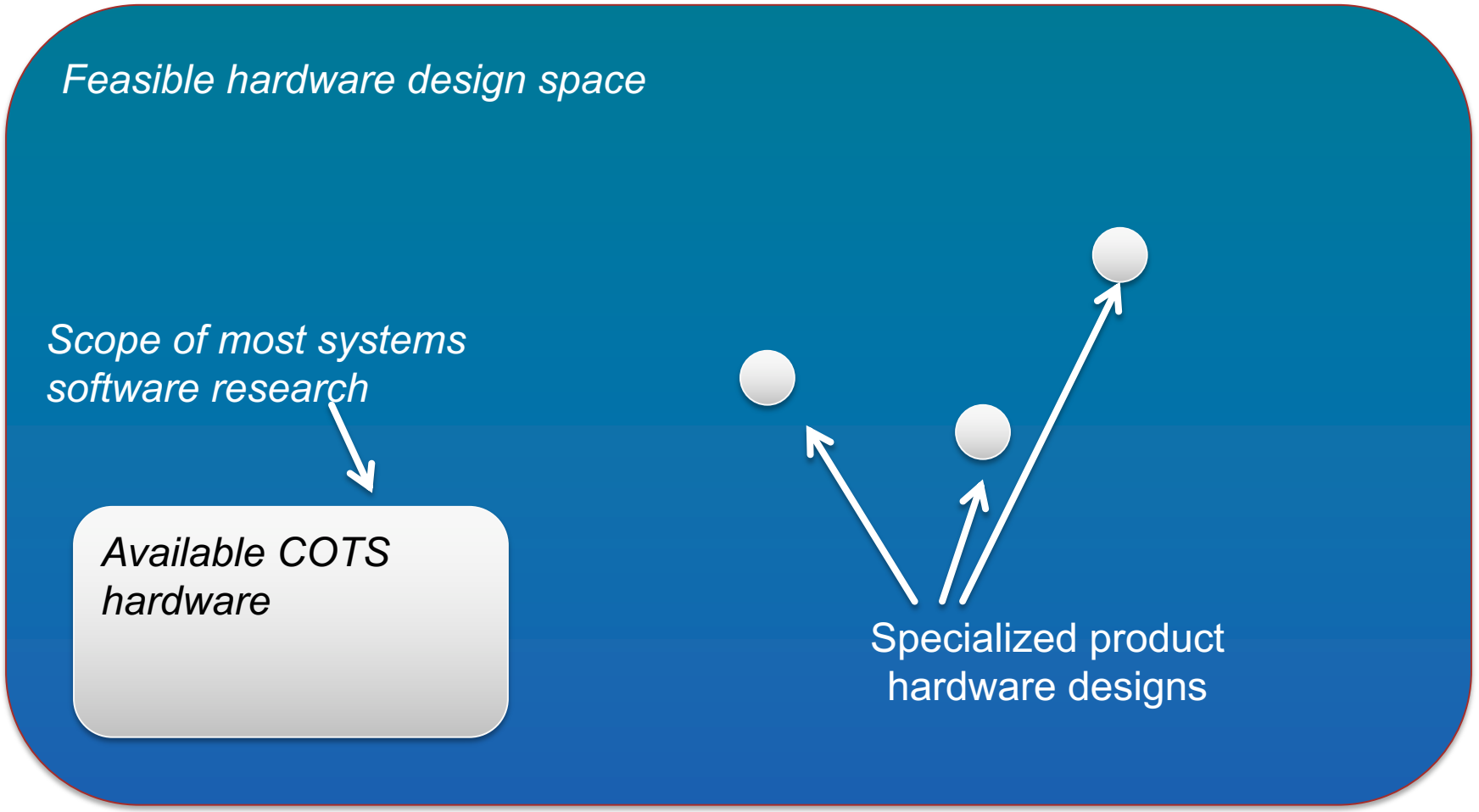
- Moore's law
- Dennar scaling, physical limits
- Multicore
- GPU, TPU, FPGA
- Data centers and the cloud
- …

- Corollary: Hardware is changing really fast (because they do not know what to do with the transistors)

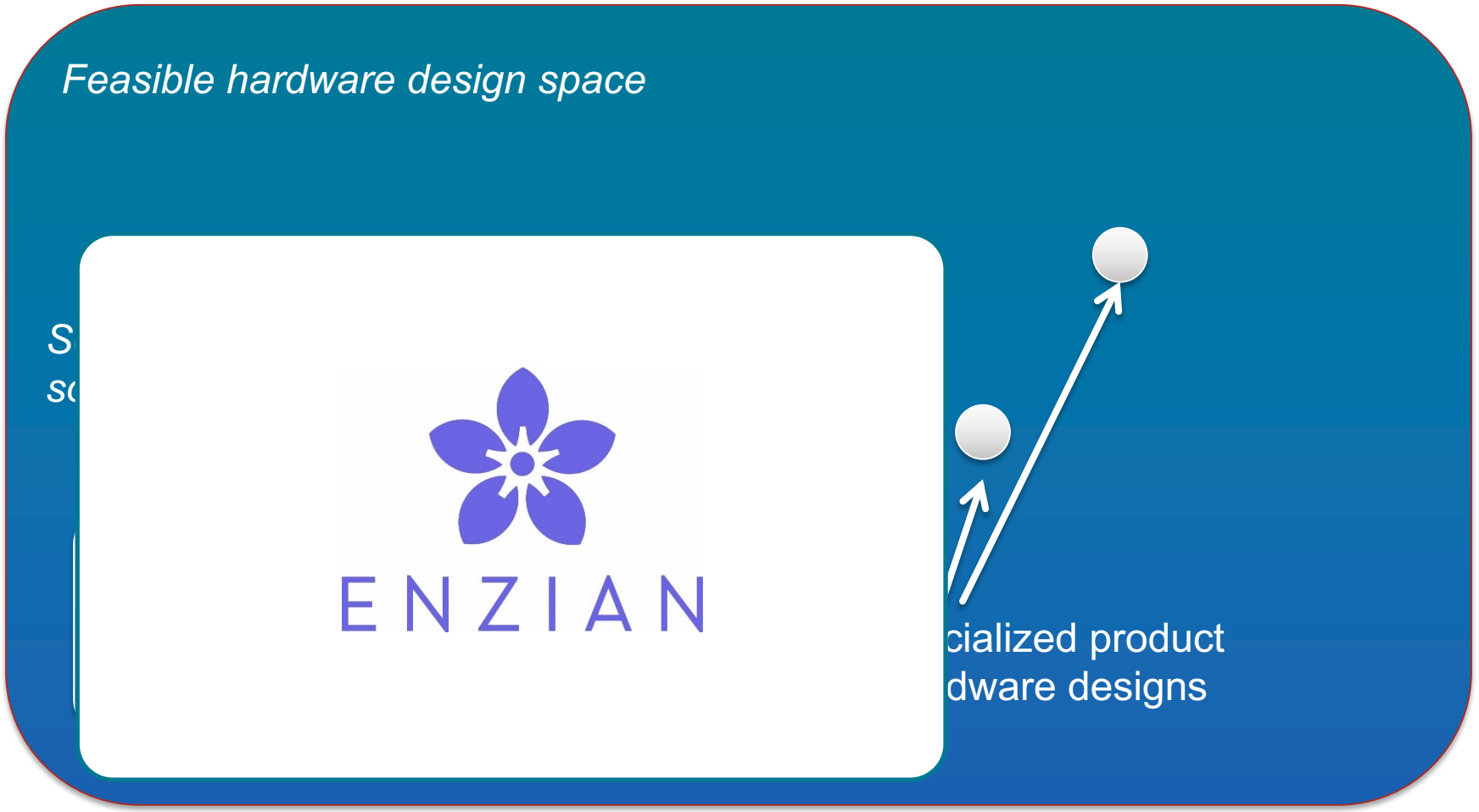


(courtesy Gustavo Alonso)

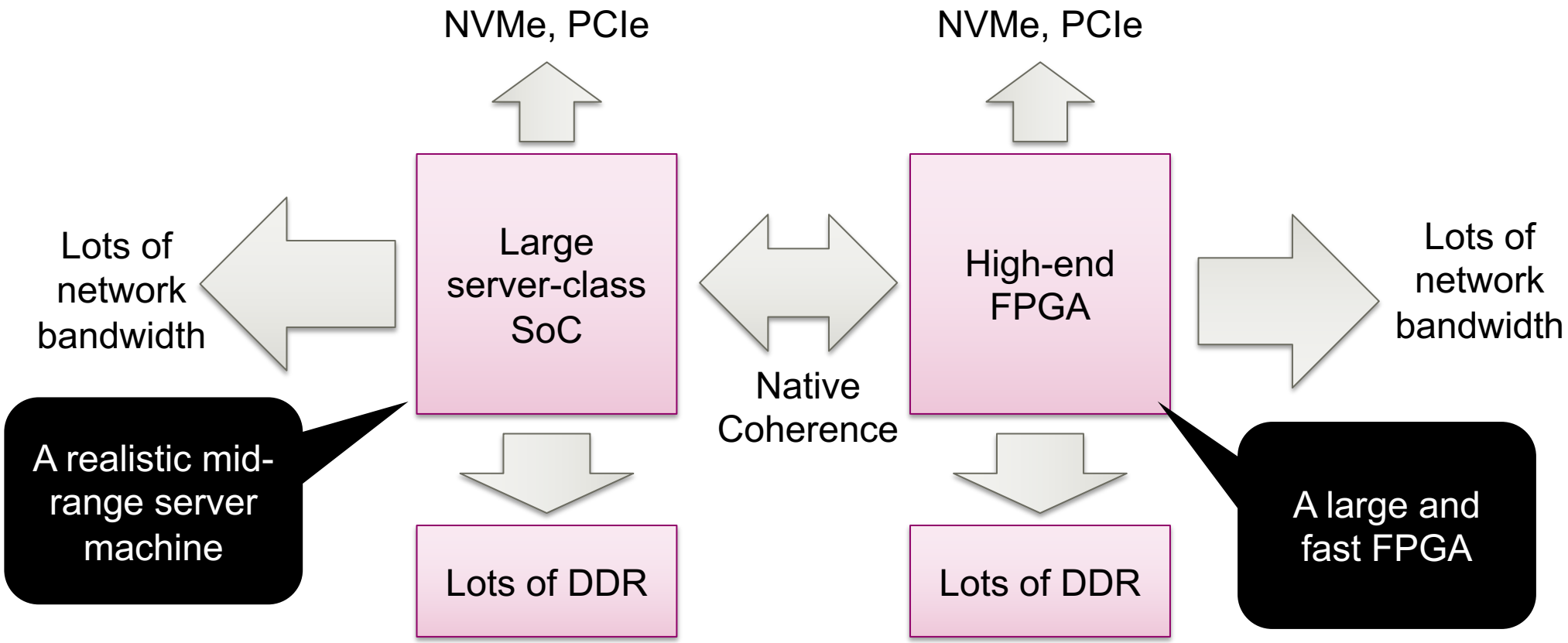
Challenge for systems software research



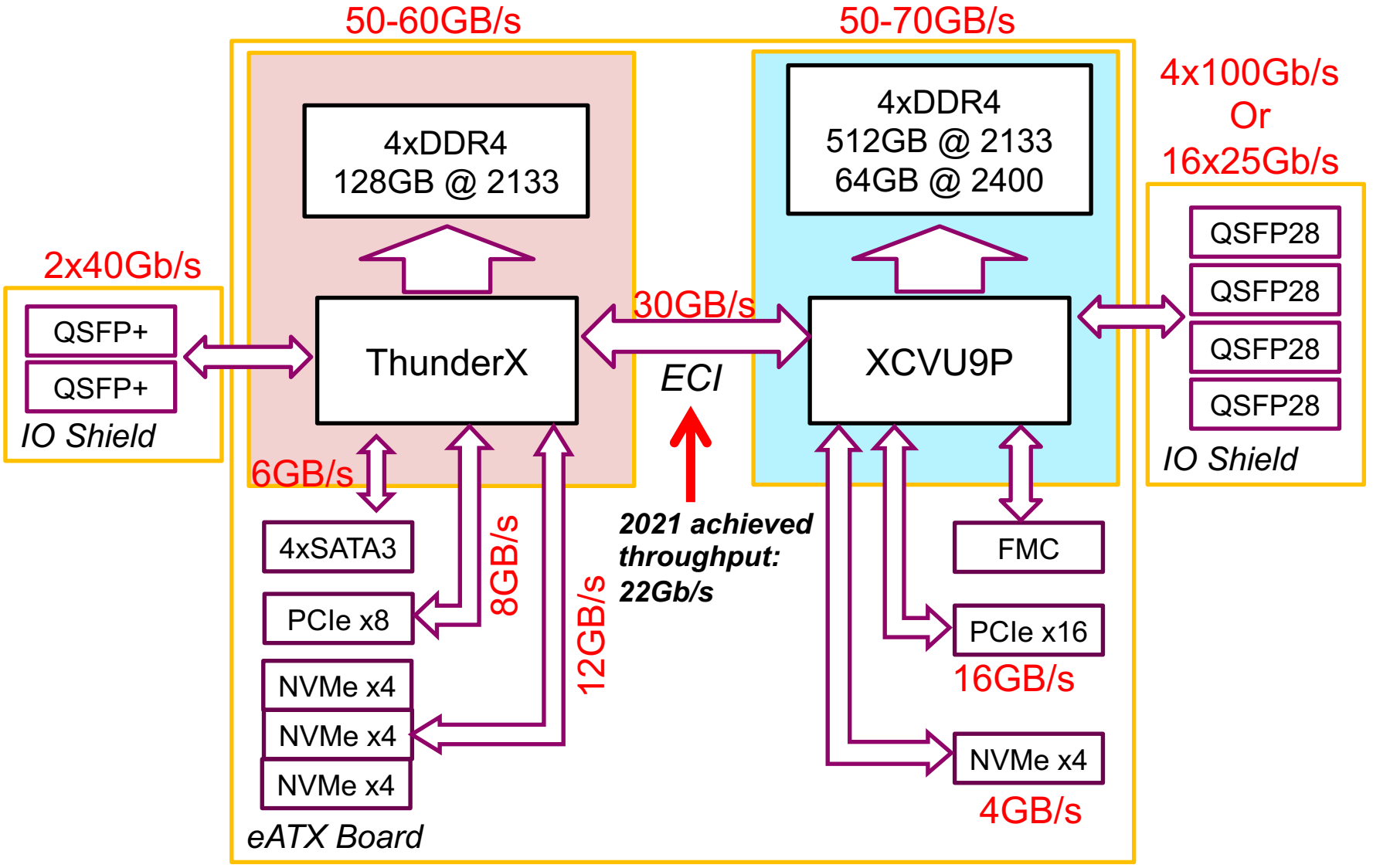
Challenge for systems software research

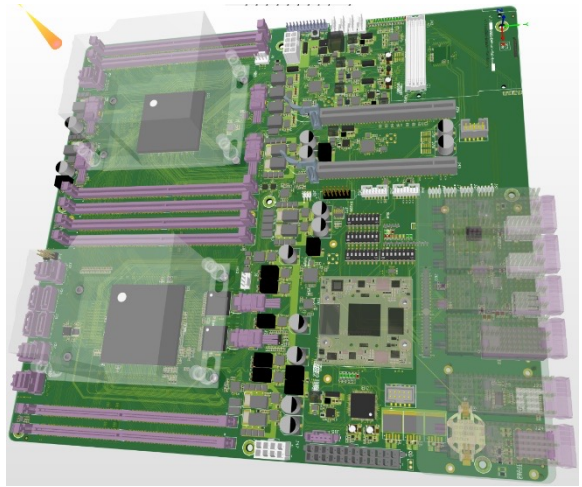


The Vision


NVMe, PCIe
NVMe, PCIe
Lots of network bandwidth
Large server-class SoC
Native Coherence
High-end FPGA
Lots of network bandwidth
A realistic mid-range server machine
Lots of DDR
Lots of DDR
A large and fast FPGA

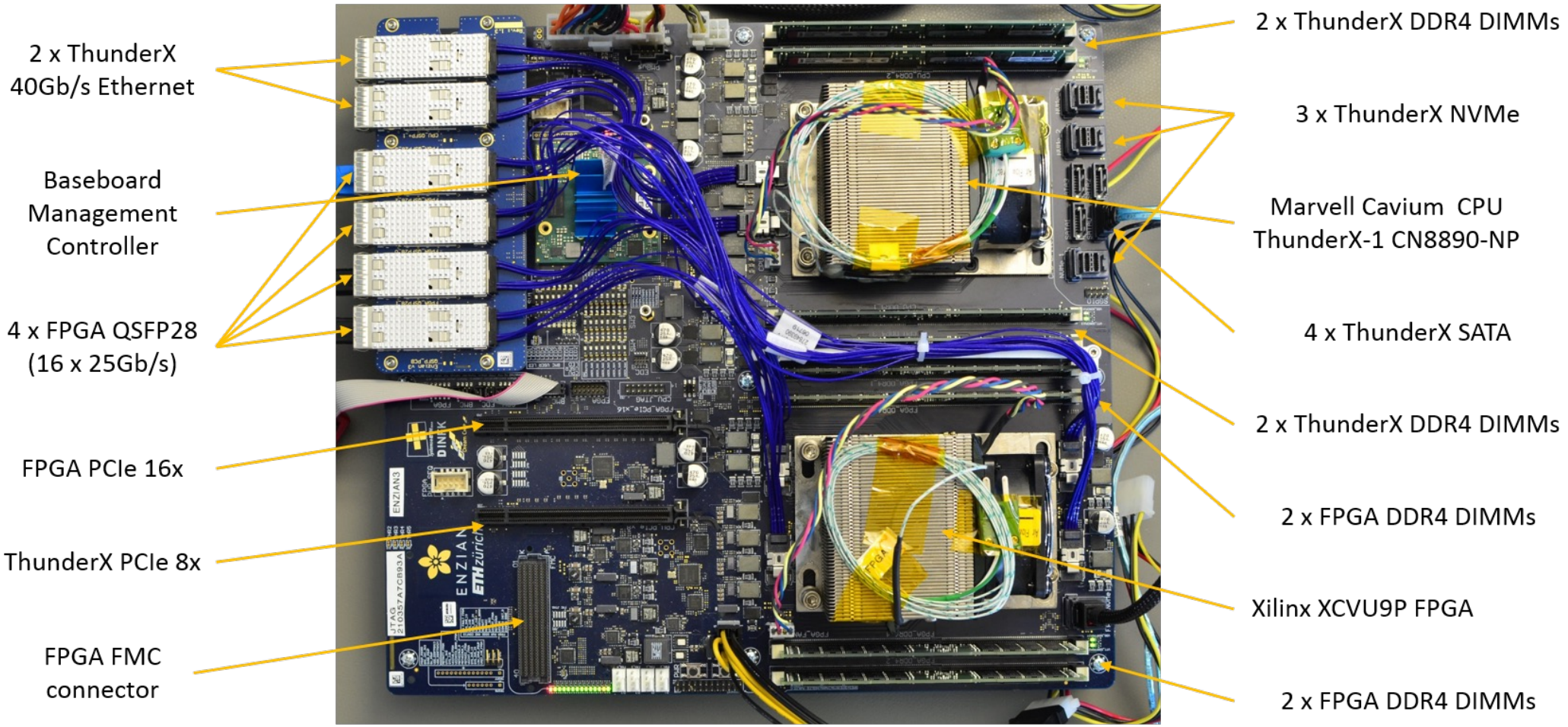
The Reality



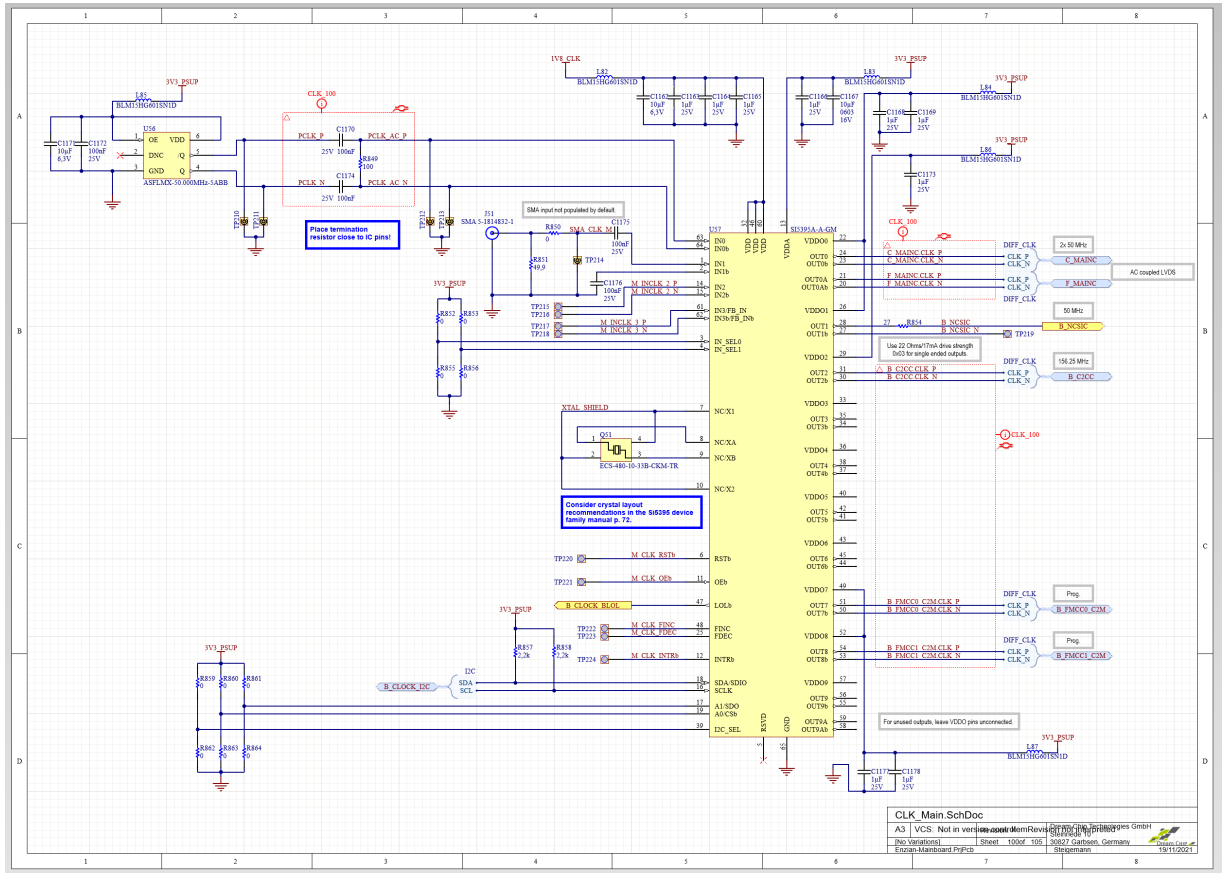


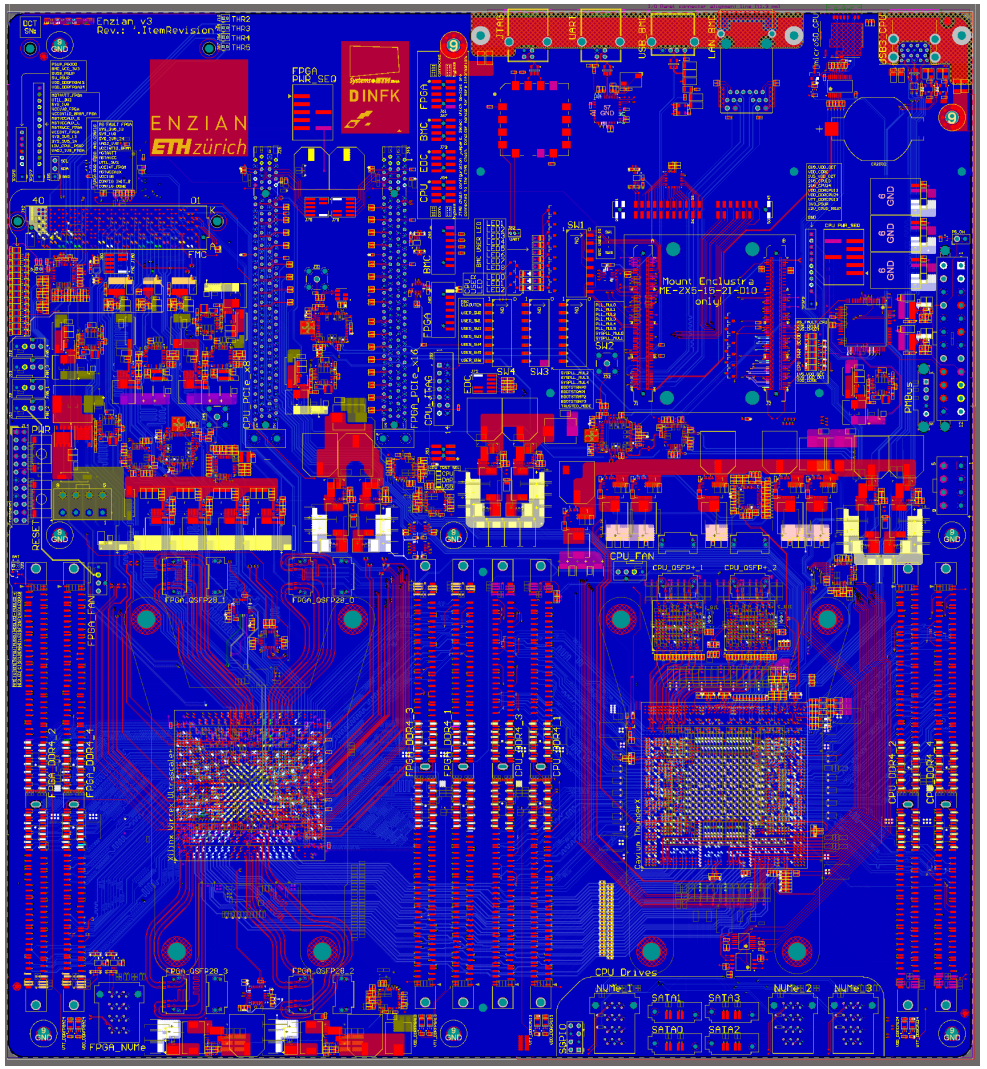
ENZIAN

So far we have 9 working machines (5 more in testing)

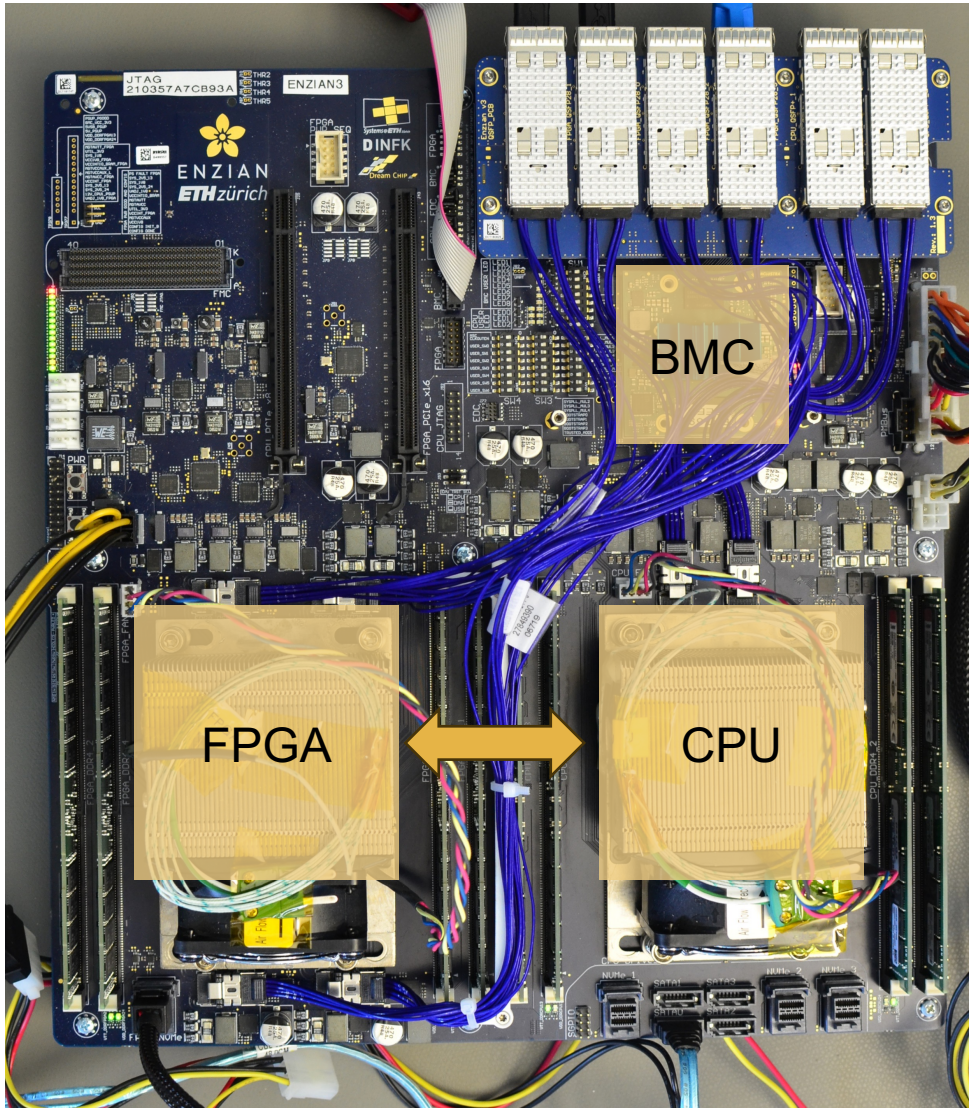


Fully Open-Source HW Design



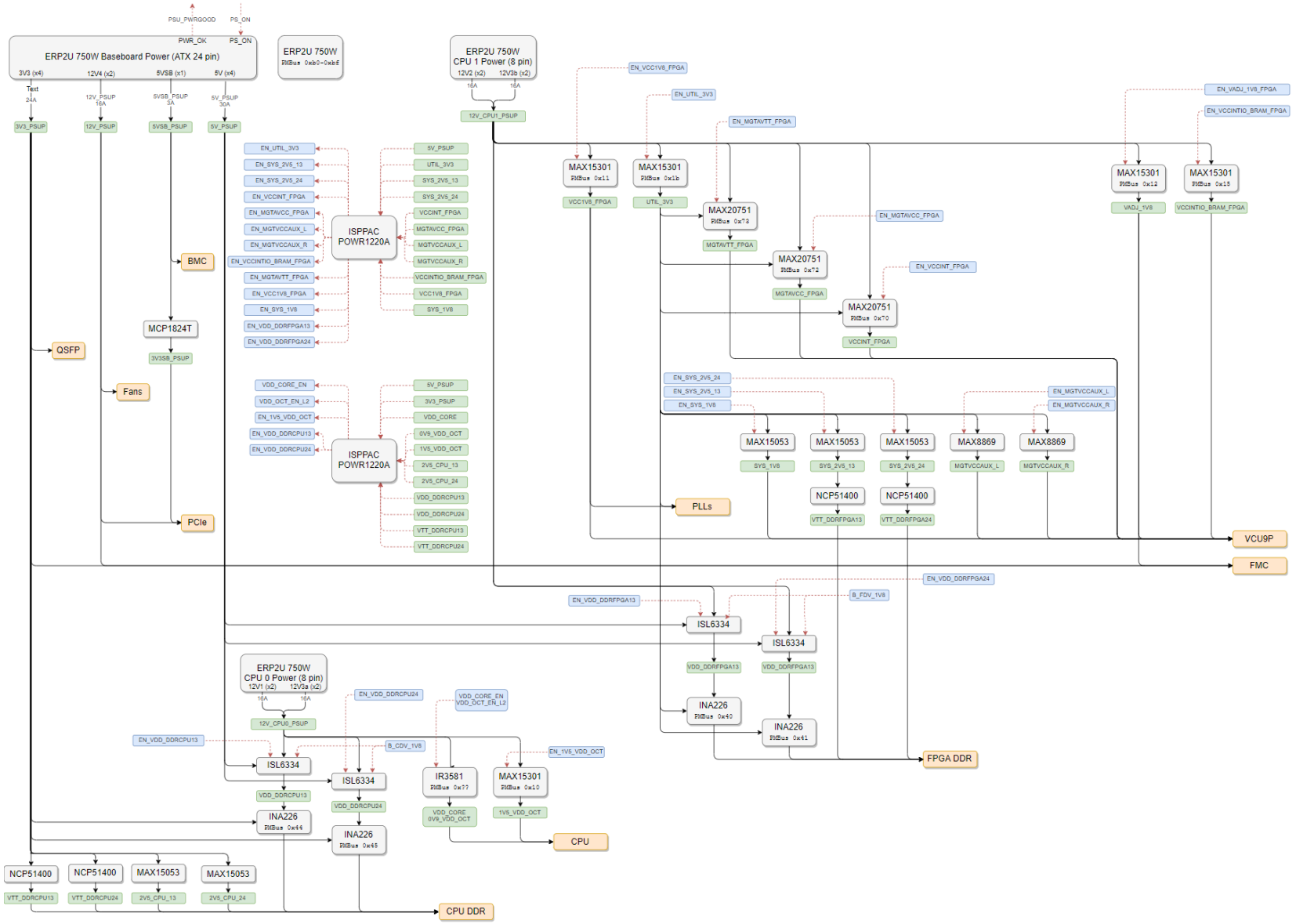


The Computer in the Computer

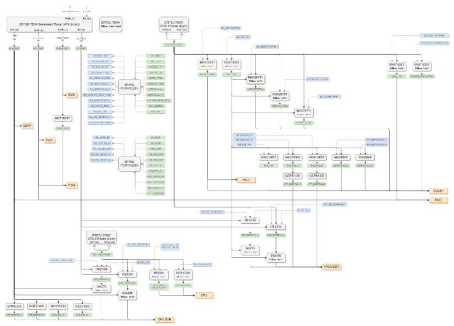


ENZIAN

Turning on computers is hard…

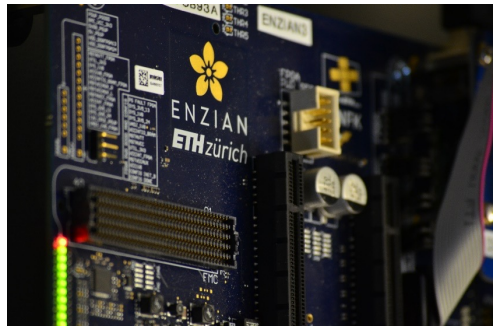


… and we need a more systematic approach!



SMT Solver

1. ▬▬
2. ▬▬
3. ▬▬
4. ▬▬
5. ▬▬

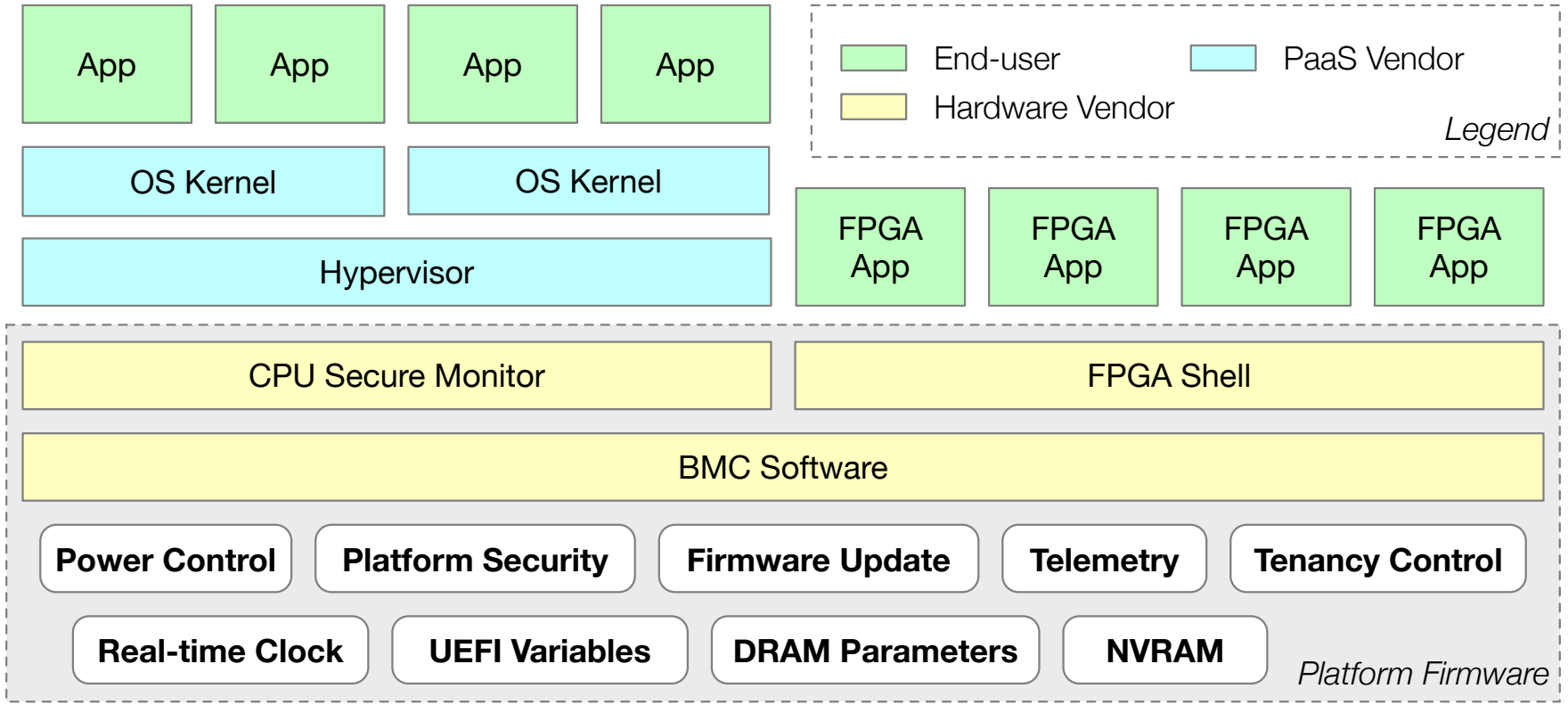


- We can derive sequences efficiently (< 10s)
- Works on real hardware
- More confidence in power-up sequences
- Basis for rigorous specification of correct behavior

Jasmin Schult, Daniel Schwyn, Michael Giardino, David Cock, Reto Achermann, and Timothy Roscoe. 2021. Declarative Power Sequencing. ACM Trans. Embed. Comput. Syst. 20, 5s, Article 84

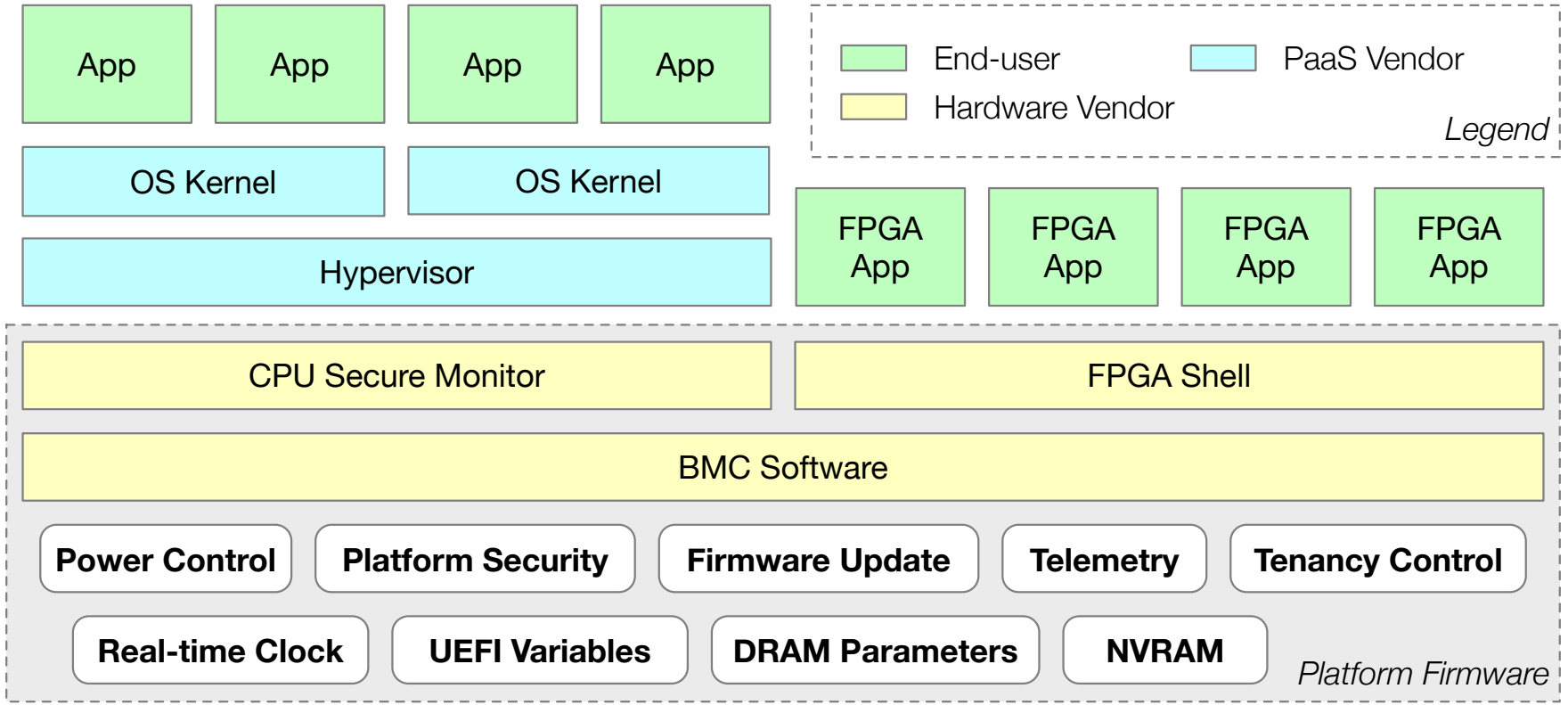
What is firmware?

- **Firmware**: a class of software that provides *low-level control* for a device's hardware
  - Provide services in the form of **actions** on **resources**



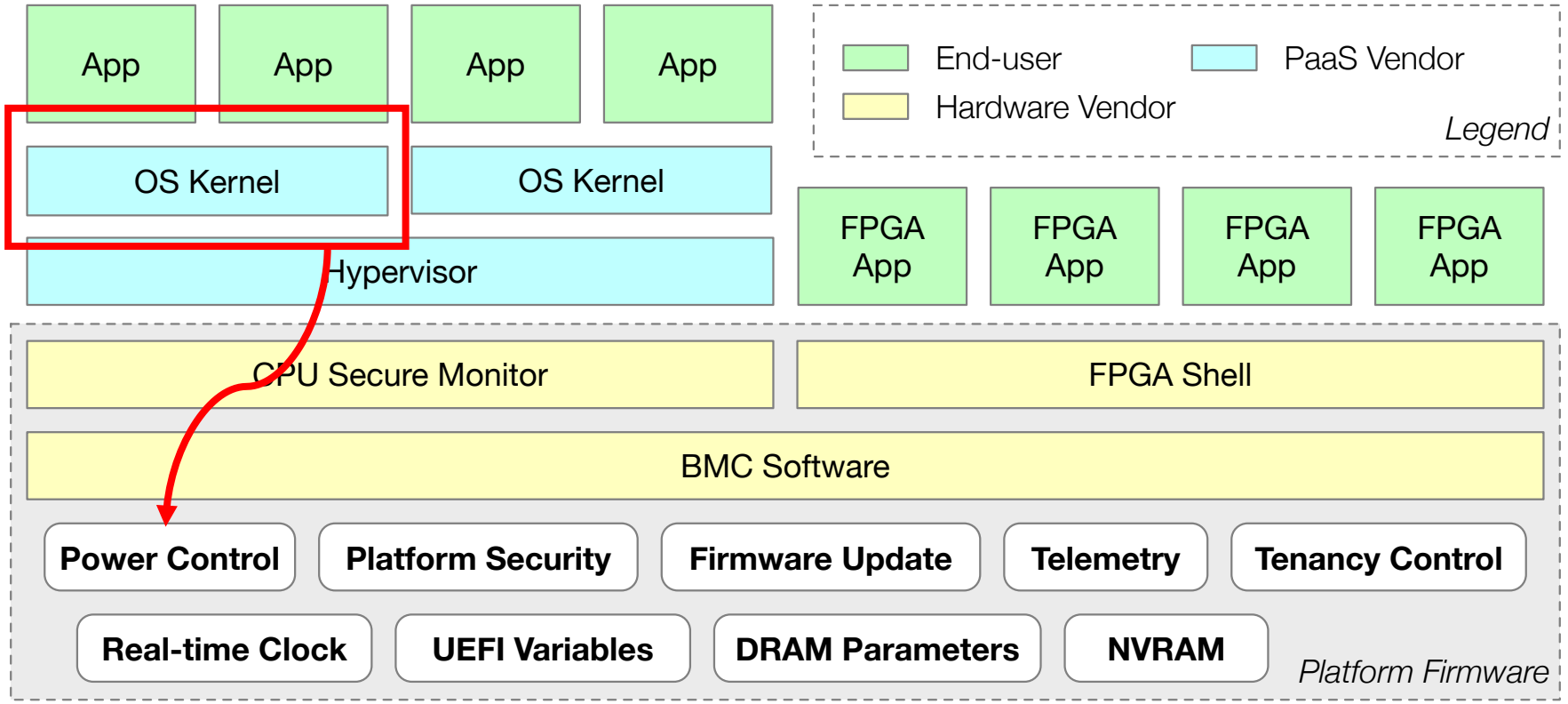
What is a firmware protocol?

- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system



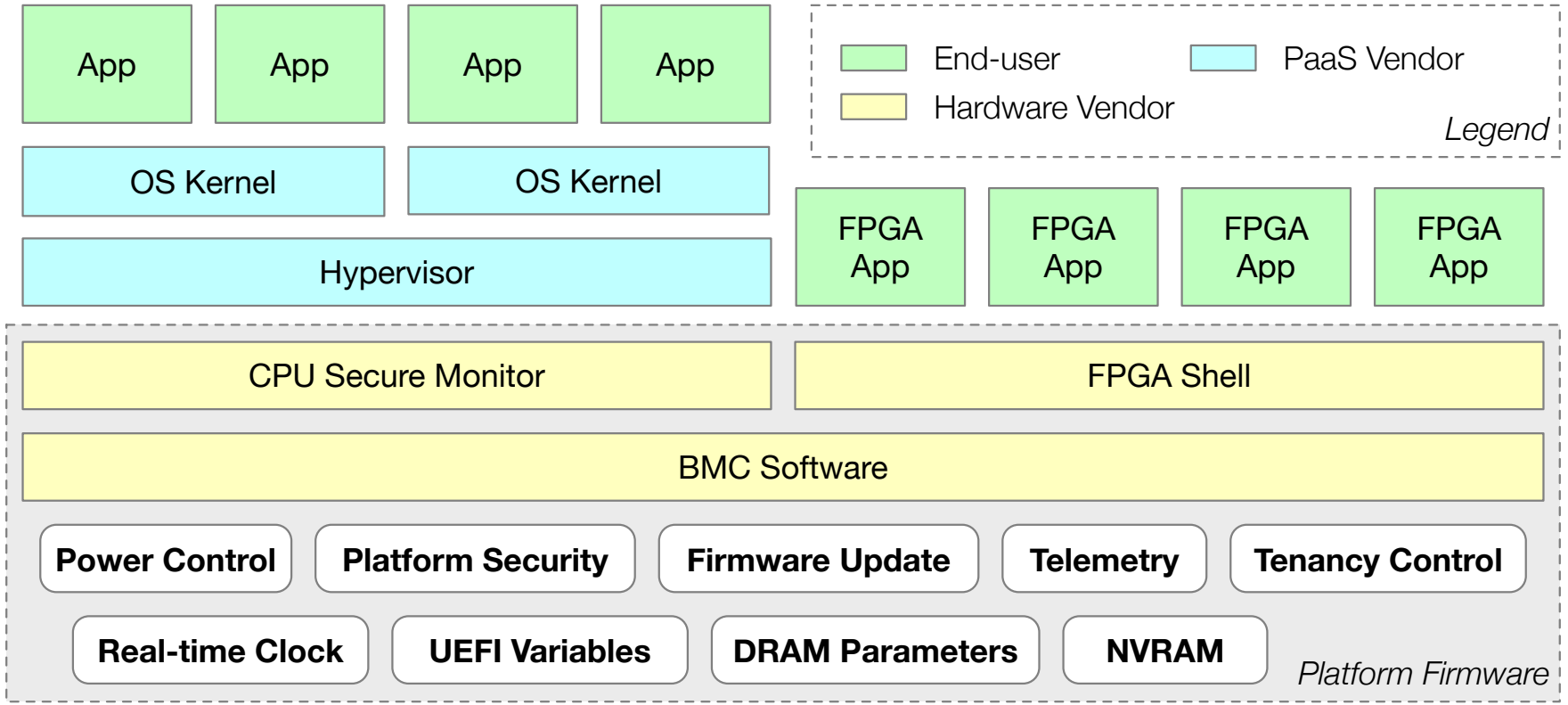
What is a firmware protocol?

- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system



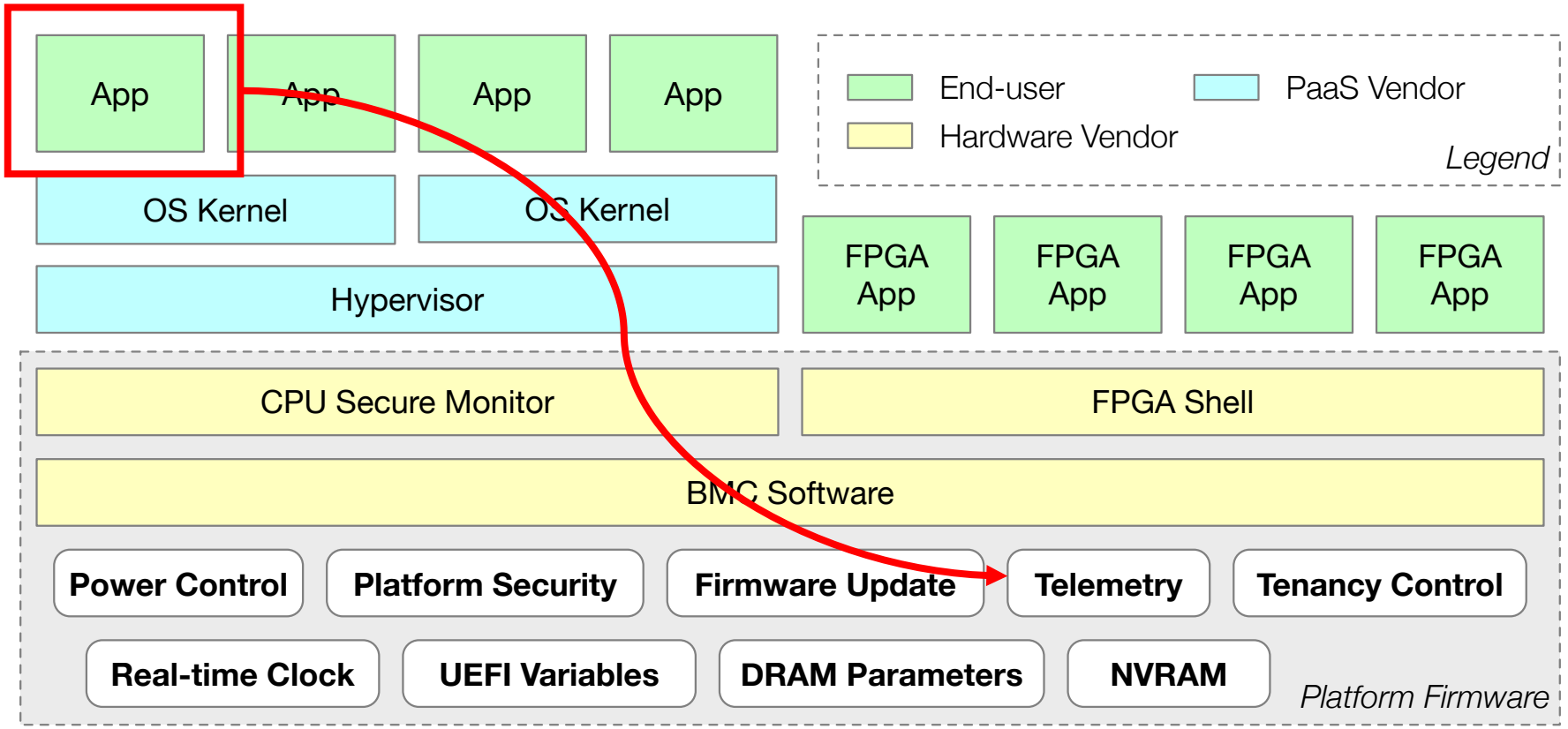
What is a firmware protocol?

- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system



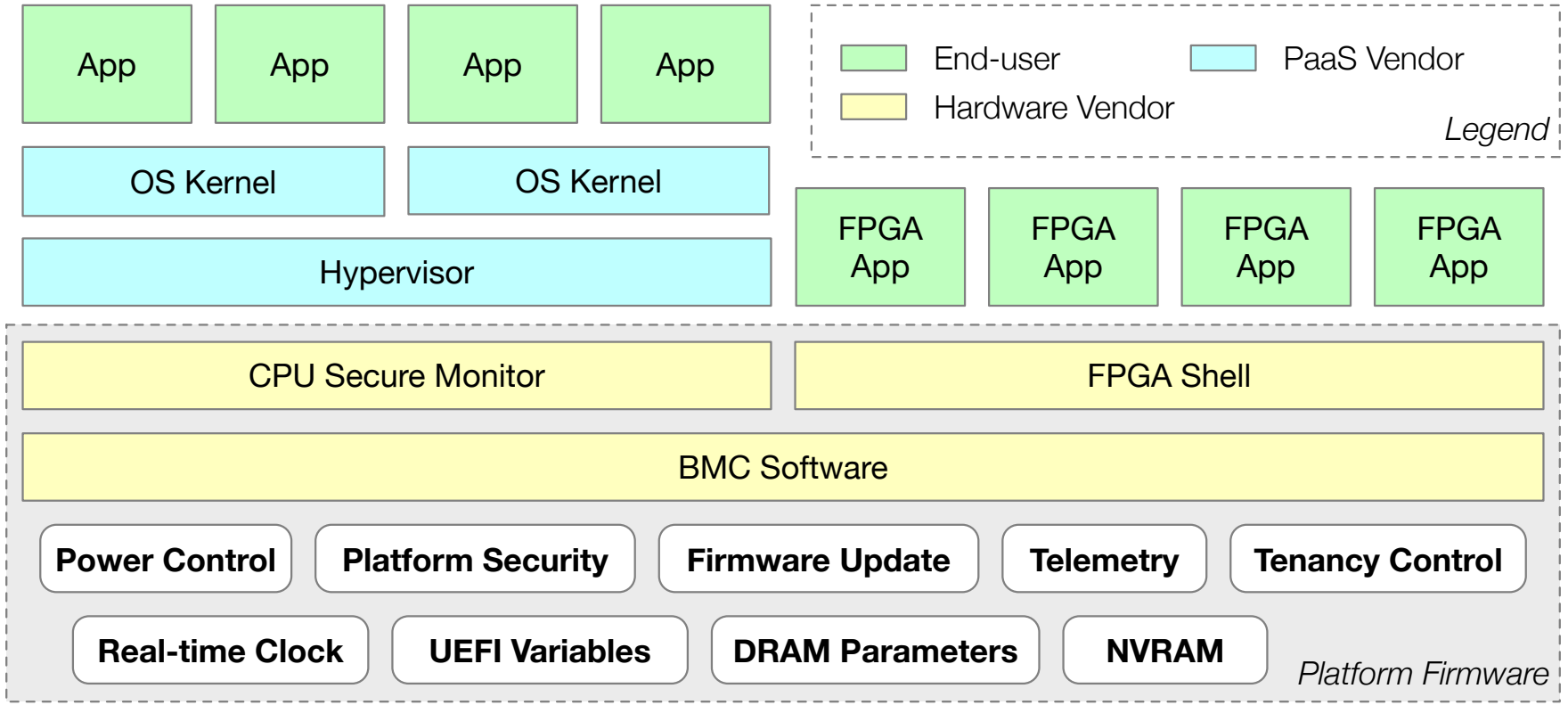
What is a firmware protocol?

- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system



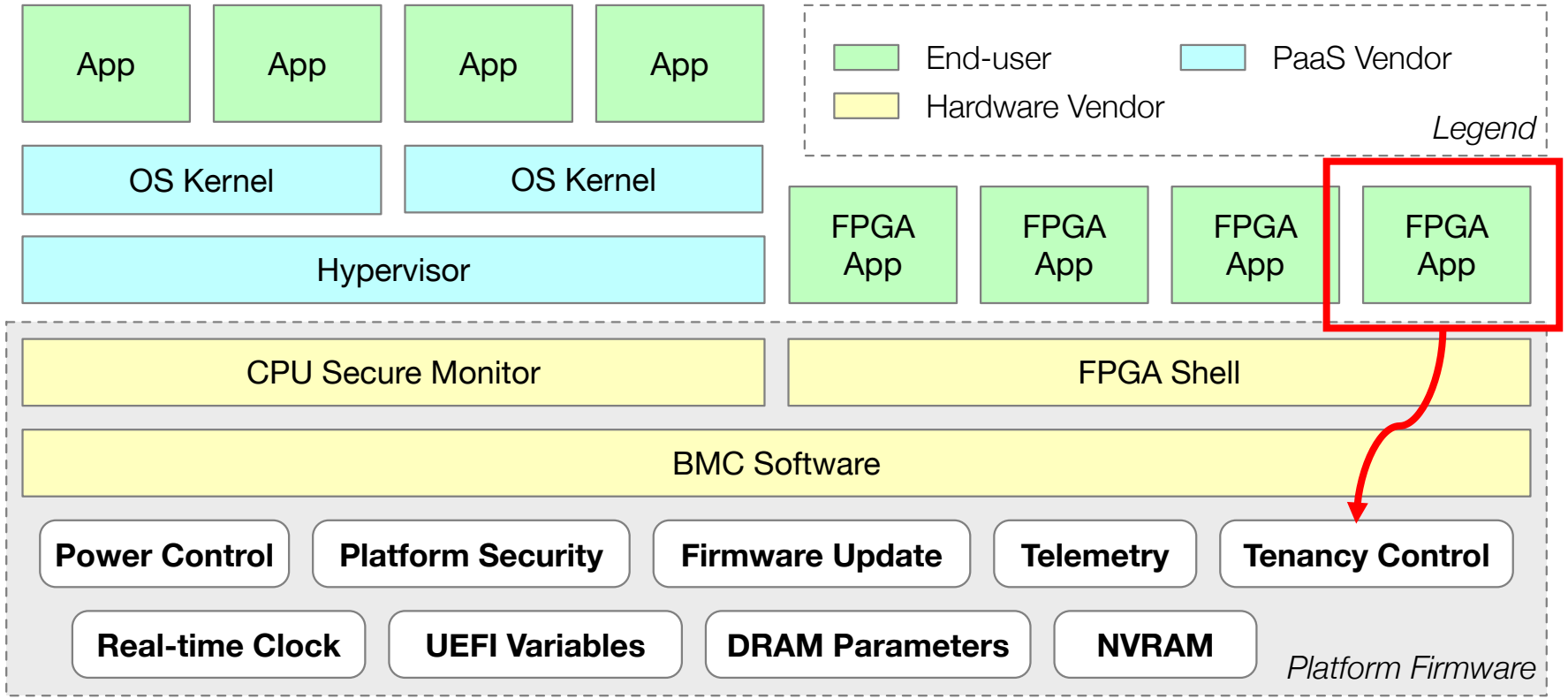
What is a firmware protocol?

- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system



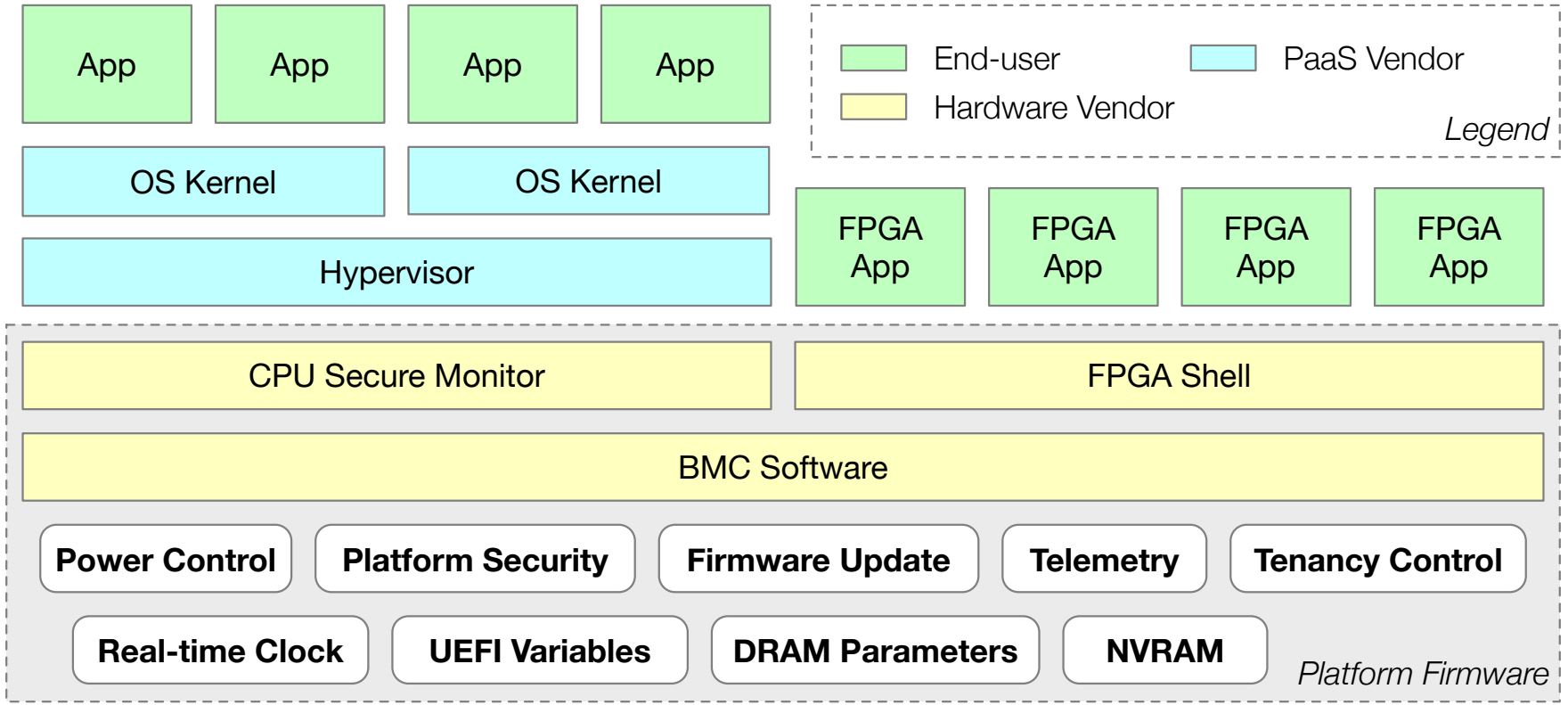
What is a firmware protocol?

- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system

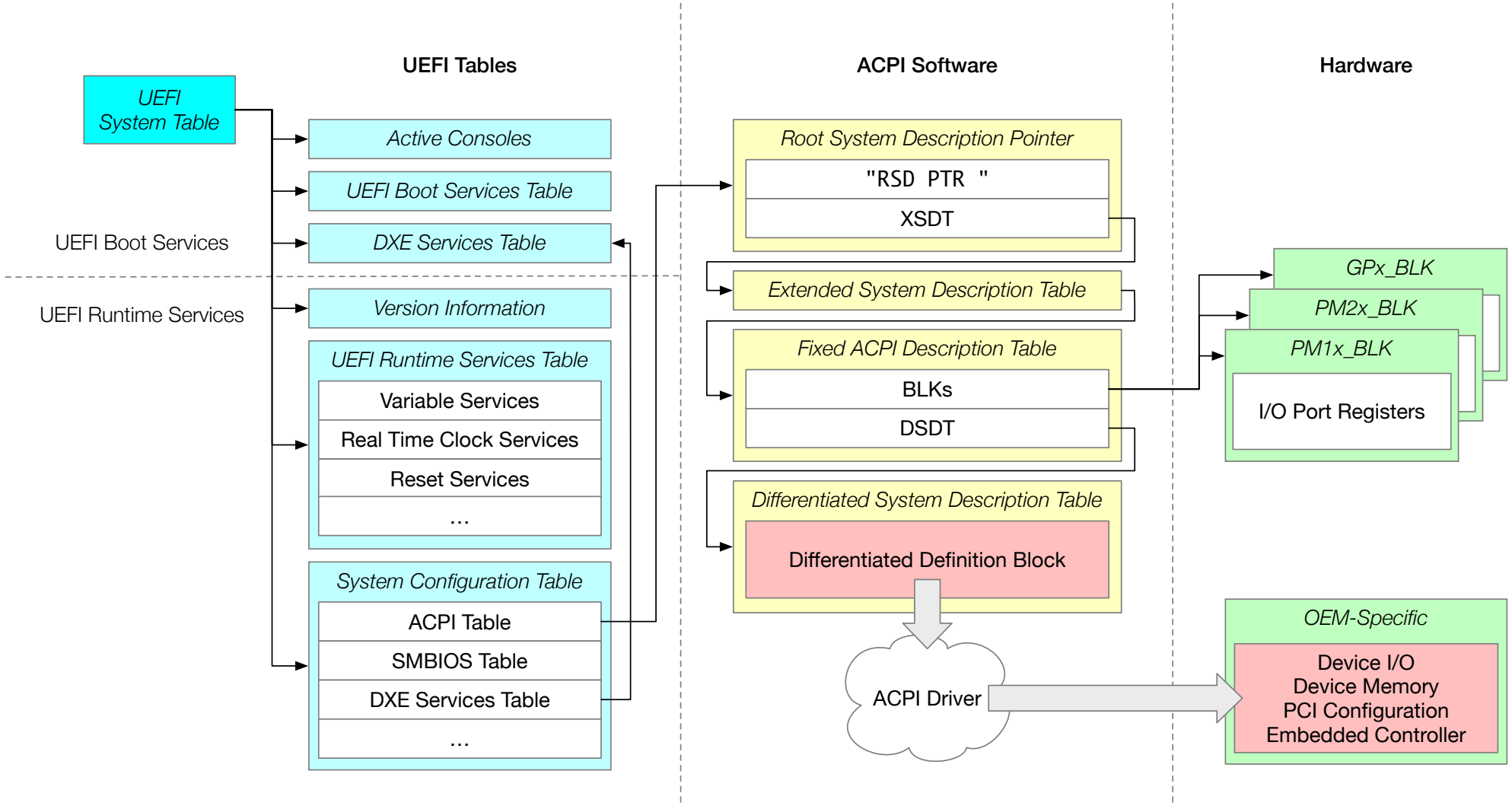


What is a firmware protocol?

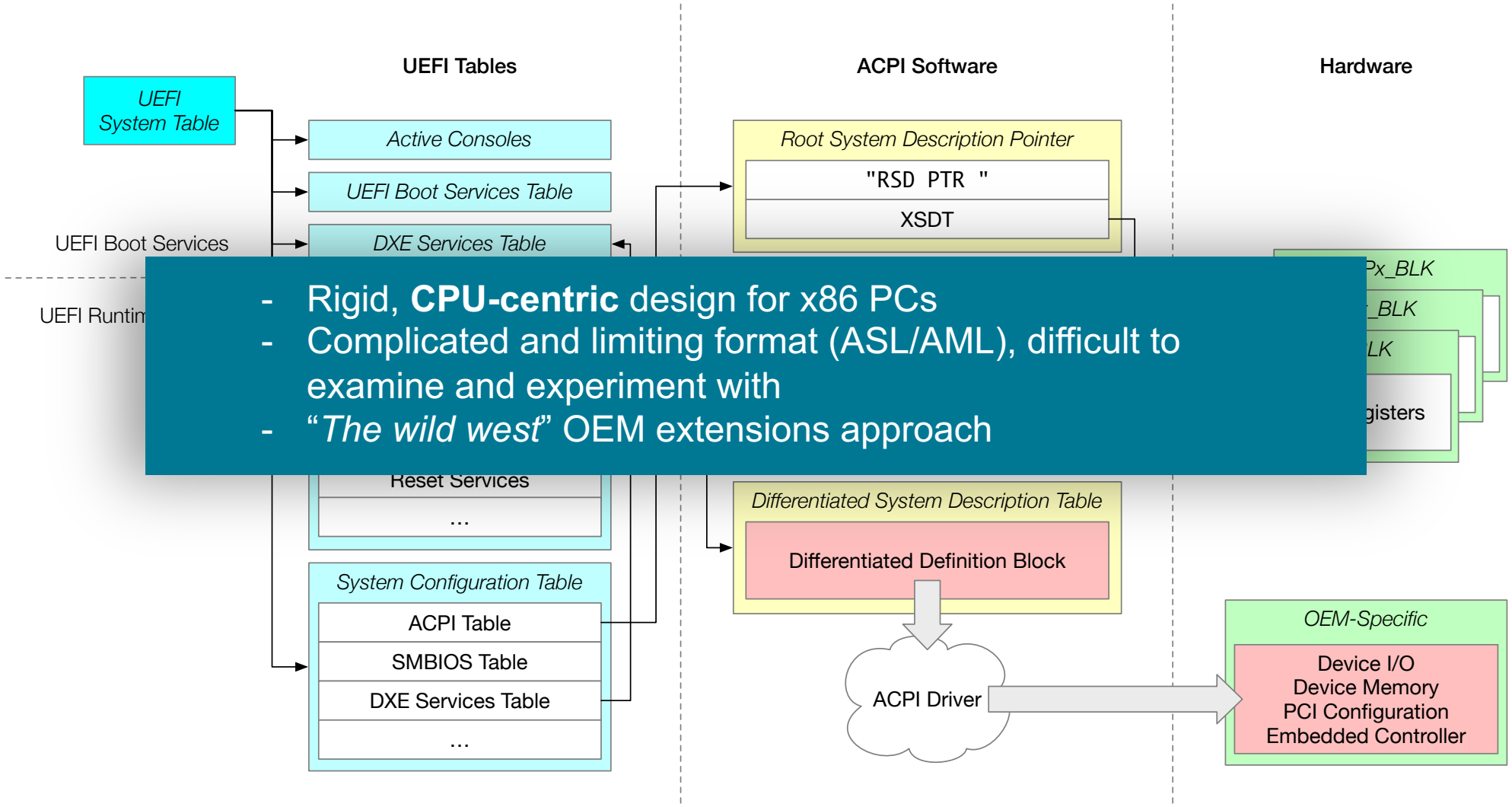
- **Firmware protocol**: semantics of how to access firmware resources
  - From different places in the system



Existing firmware protocols – UEFI/ACPI

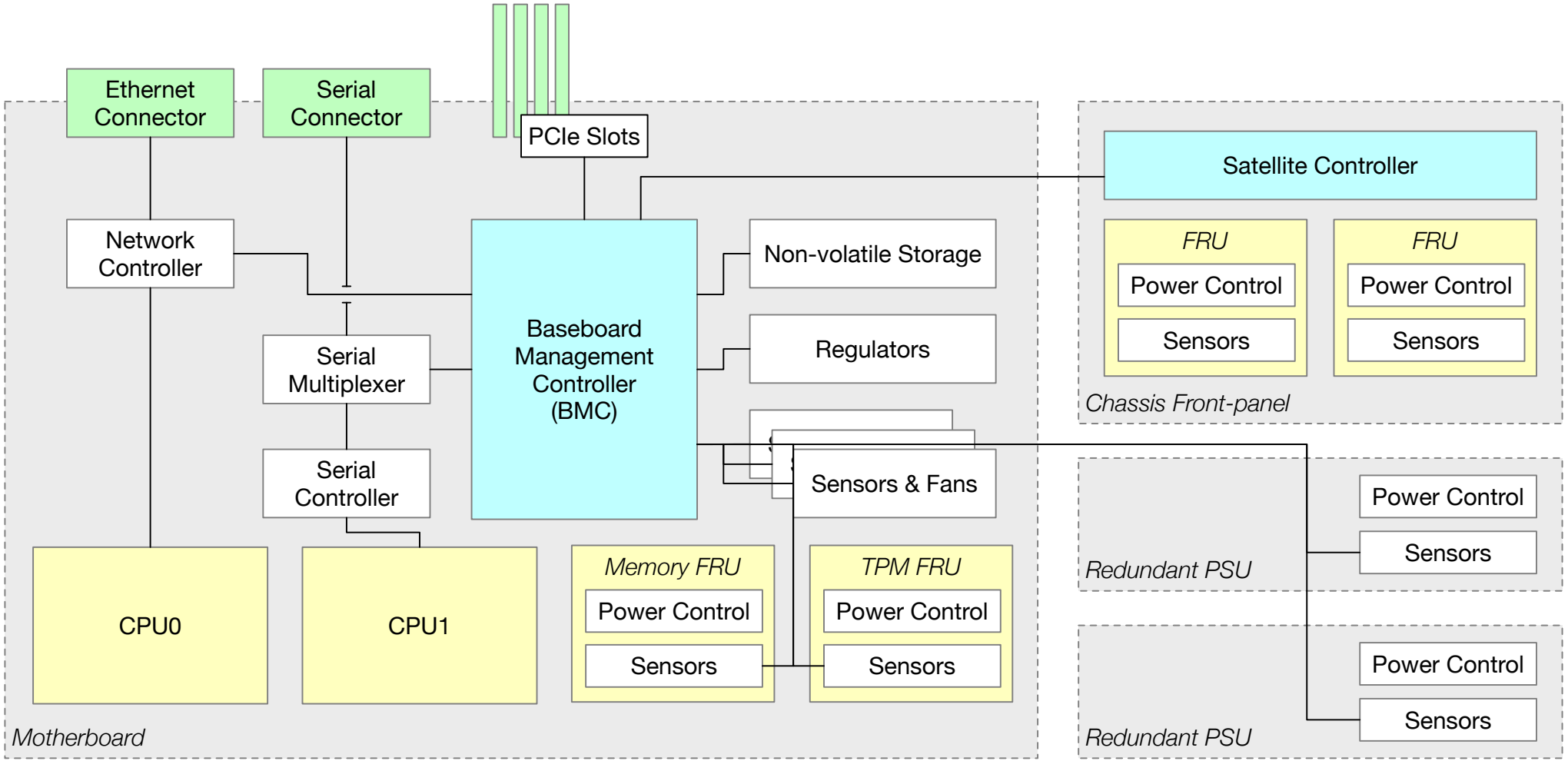


Existing firmware protocols – UEFI/ACPI

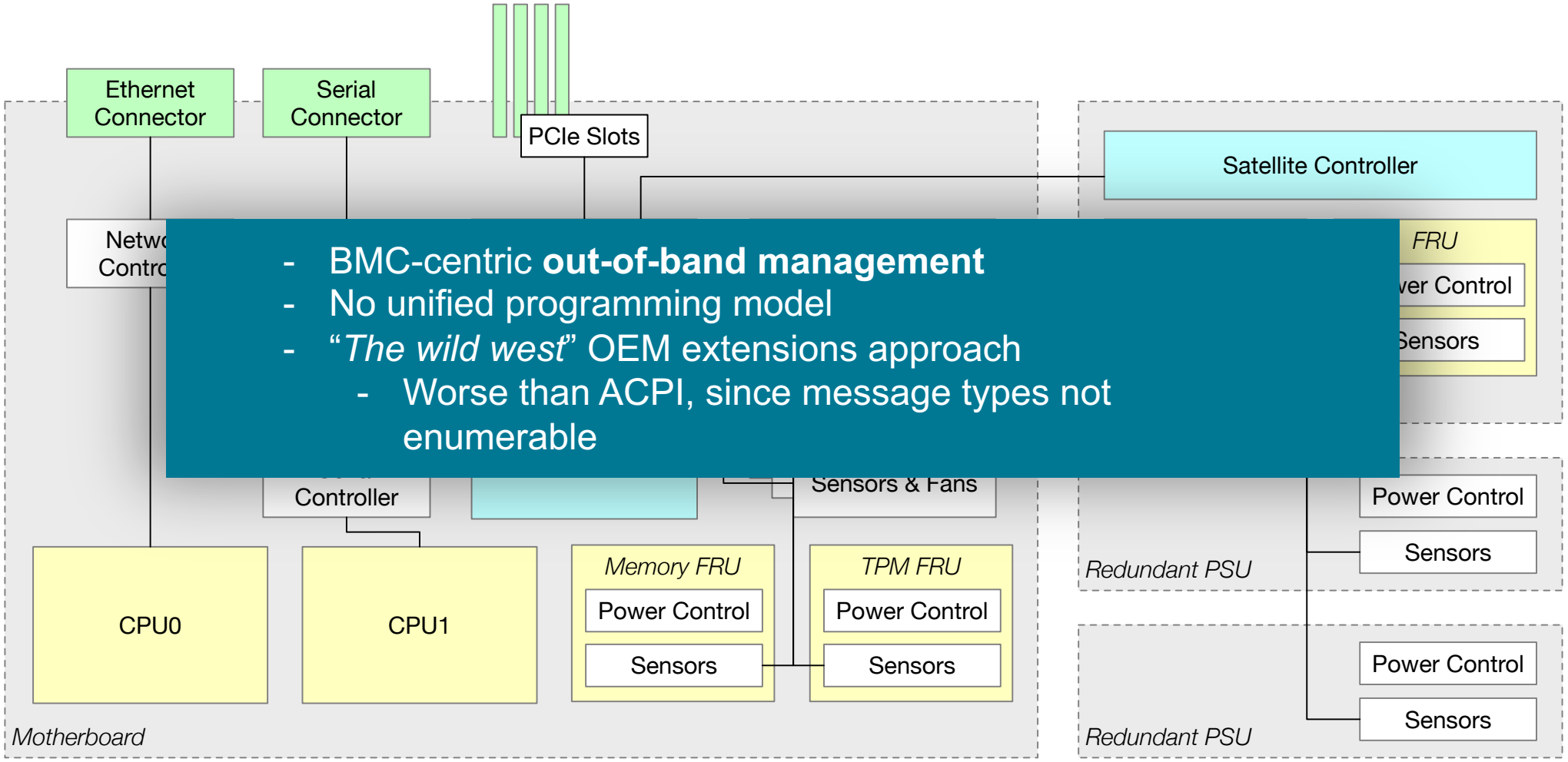


- Rigid, **CPU-centric** design for x86 PCs
- Complicated and limiting format (ASL/AML), difficult to examine and experiment with
- "*The wild west*" OEM extensions approach

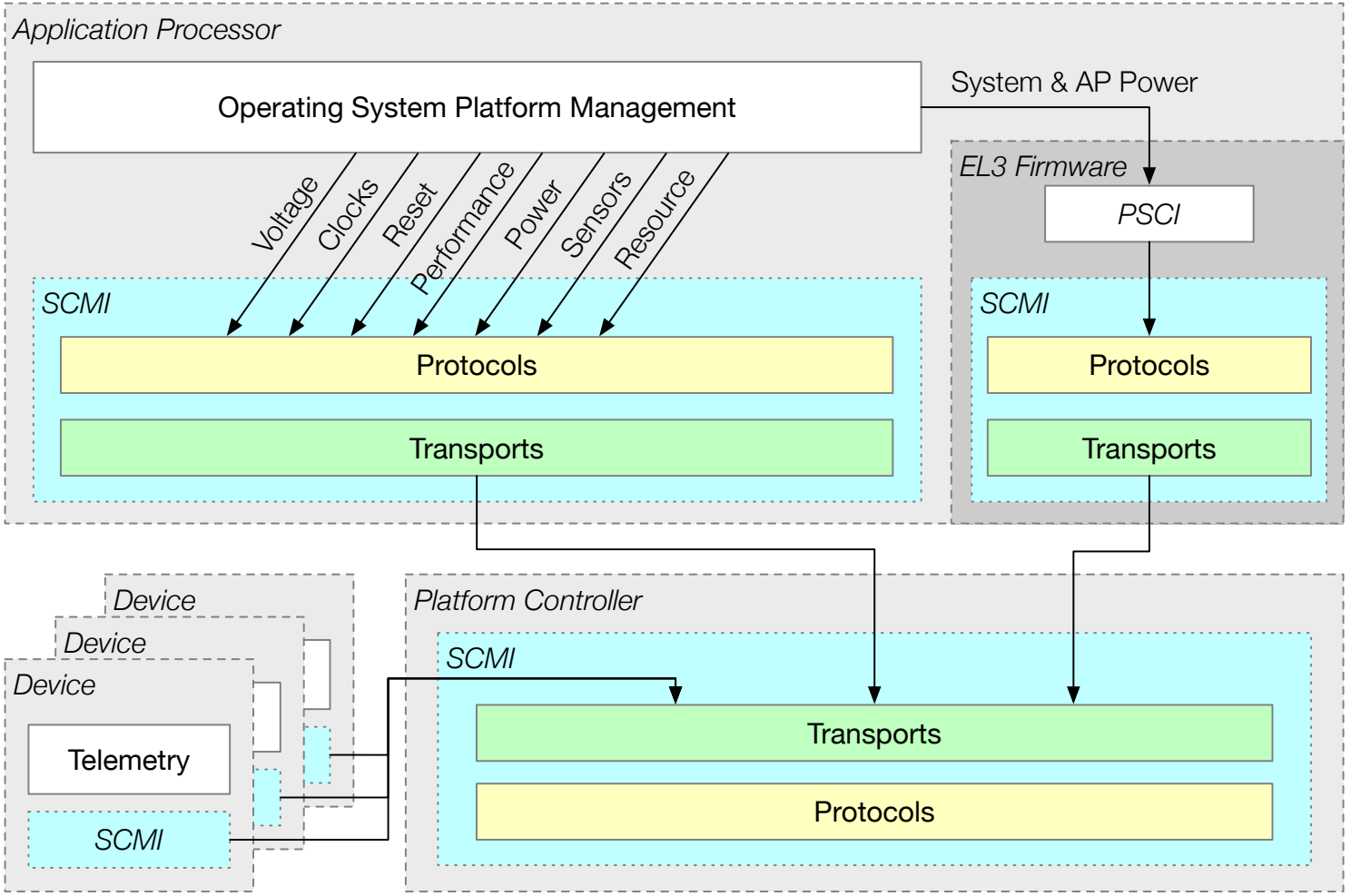
Existing firmware protocols – IPMI



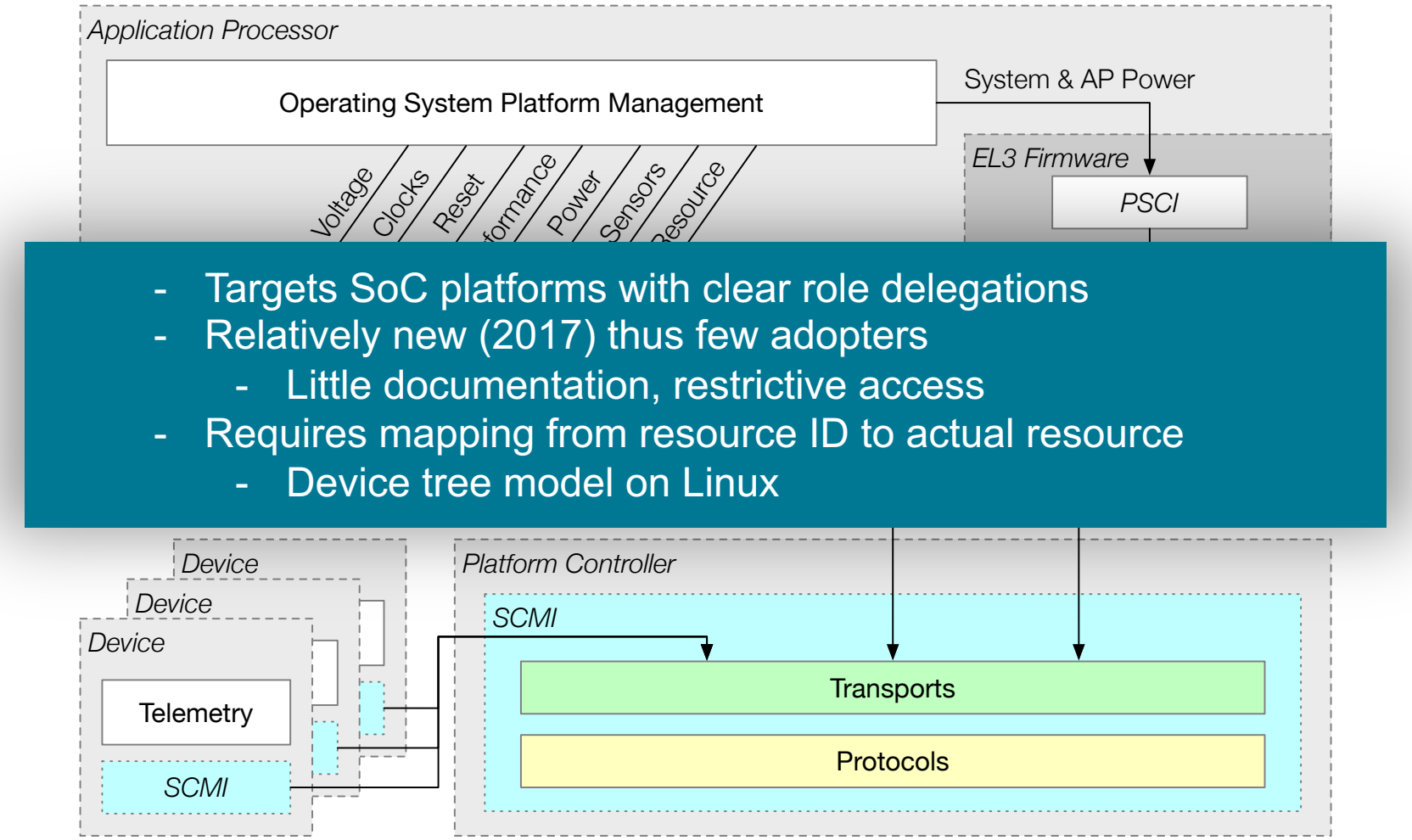
Existing firmware protocols – IPMI



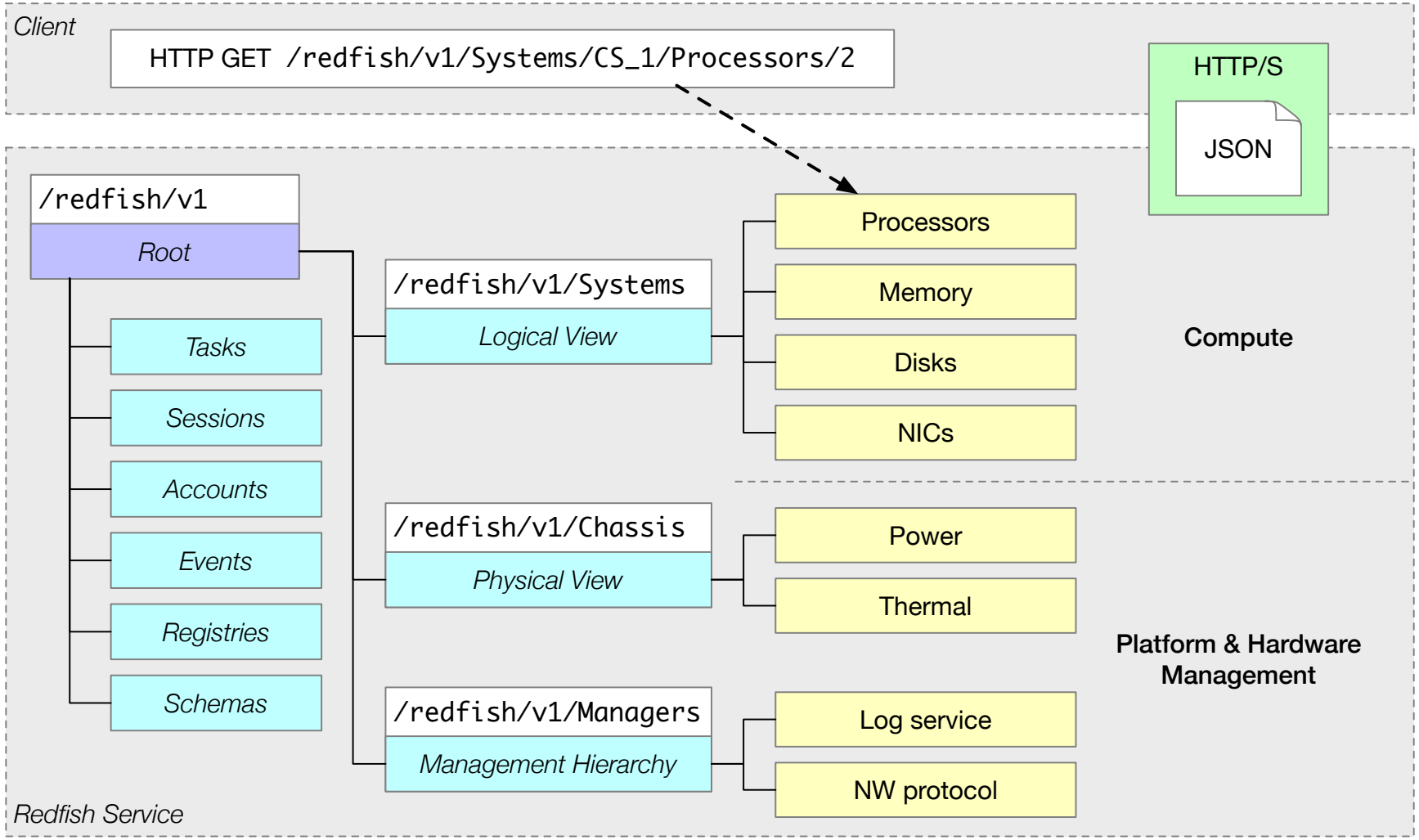
Existing firmware protocols – SCMI



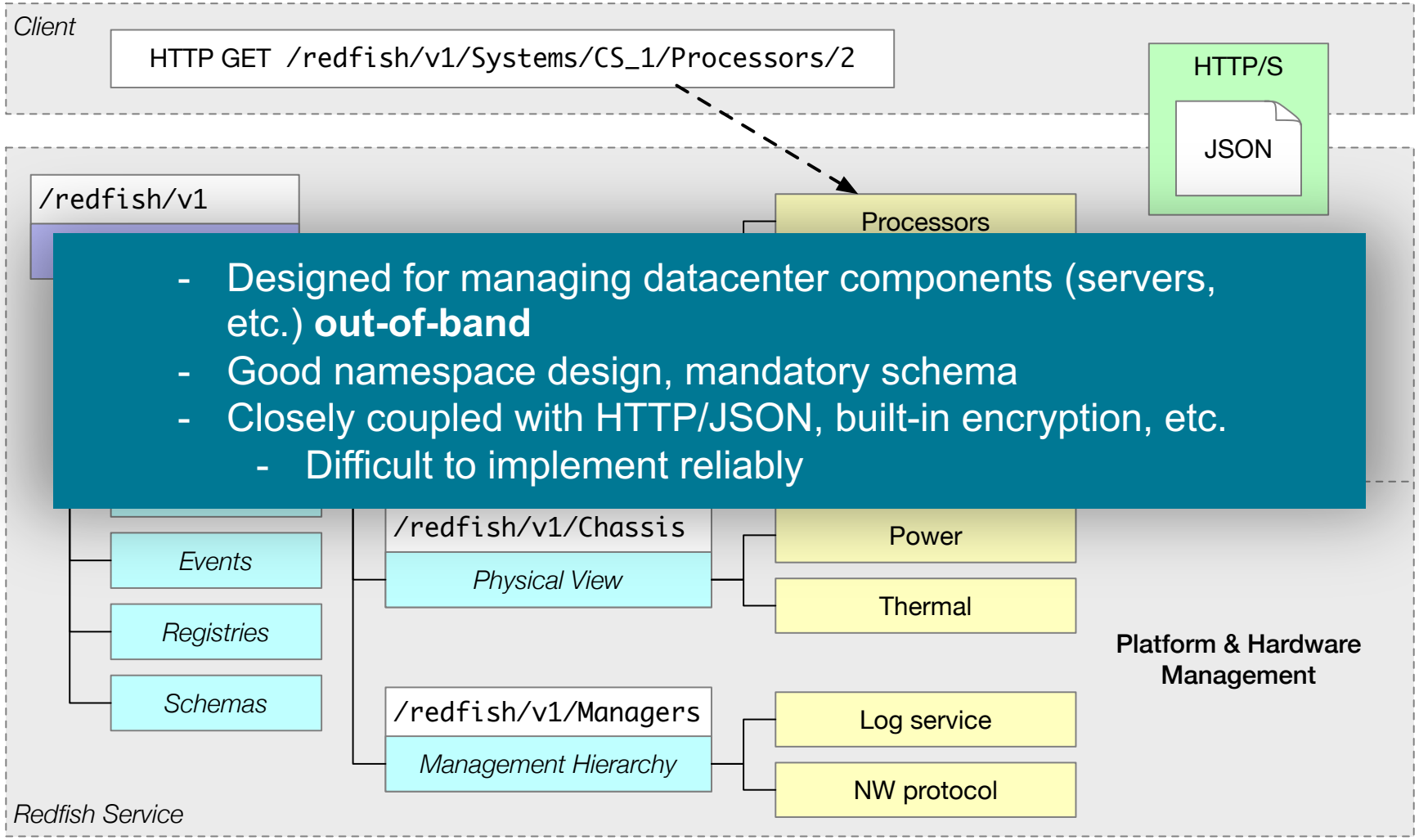
Existing firmware protocols – SCMI



Existing firmware protocols – DMTF Redfish



Existing firmware protocols – DMTF Redfish

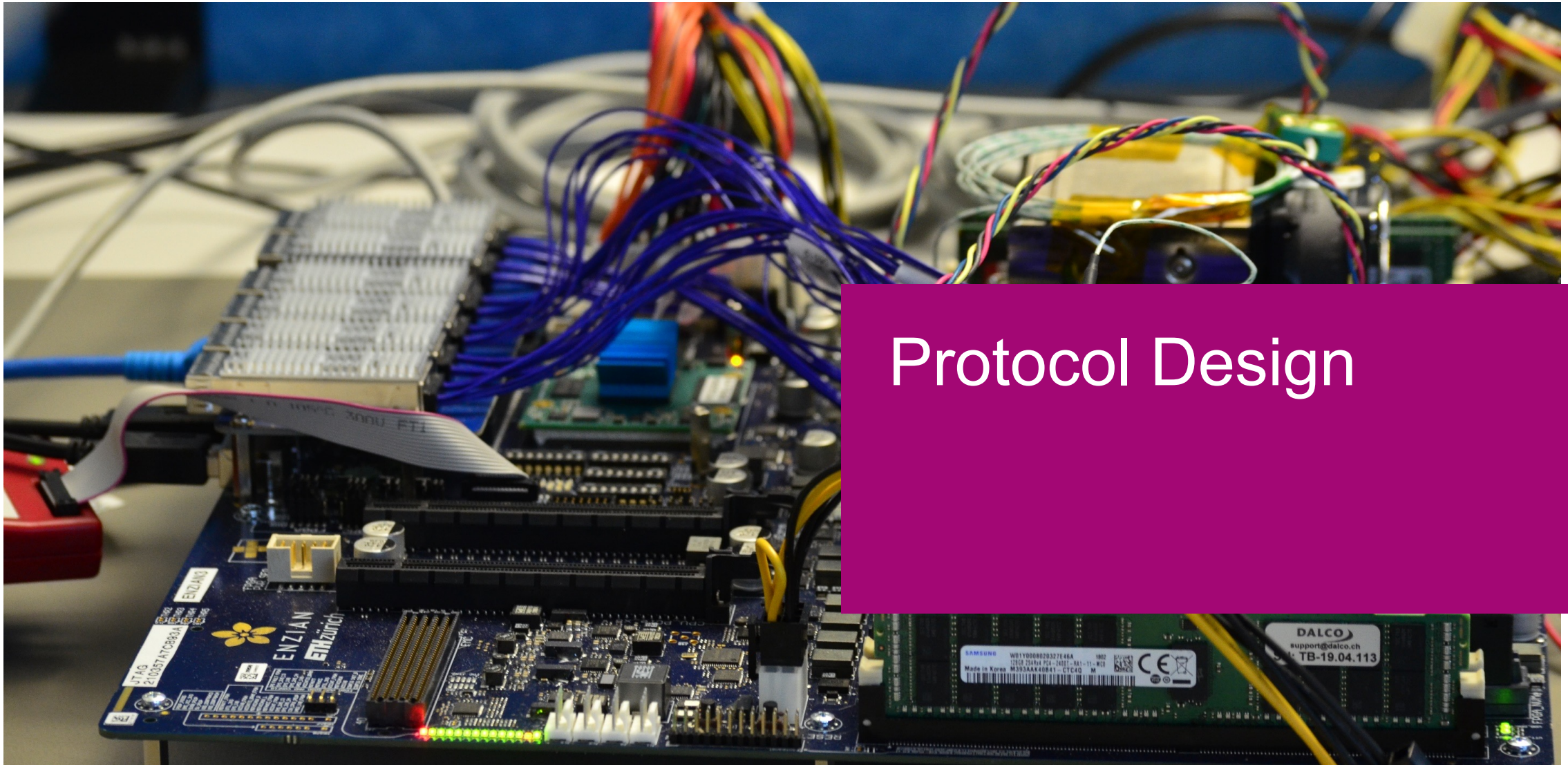


ETH zürich

ENZIAN

Systems@ETH Zürich

DINFK



Protocol Design

Requirements for a suitable firmware protocol **for Enzian**

Recall that Enzian is a **research** *heterogeneous* system…

Requirements for a suitable firmware protocol **for Enzian**

Recall that Enzian is a **research** *heterogeneous* system…

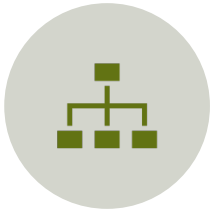
- Extensible
- Organised and discoverable
- Modular and composable
- Simple to implement
- Inspectable
- Usable

Overview – Enzian Firmware Resource Interface

Stateless RPC

Synchronous with async extension

Shared actions on resources



Hierarchical namespaces

Schema-based

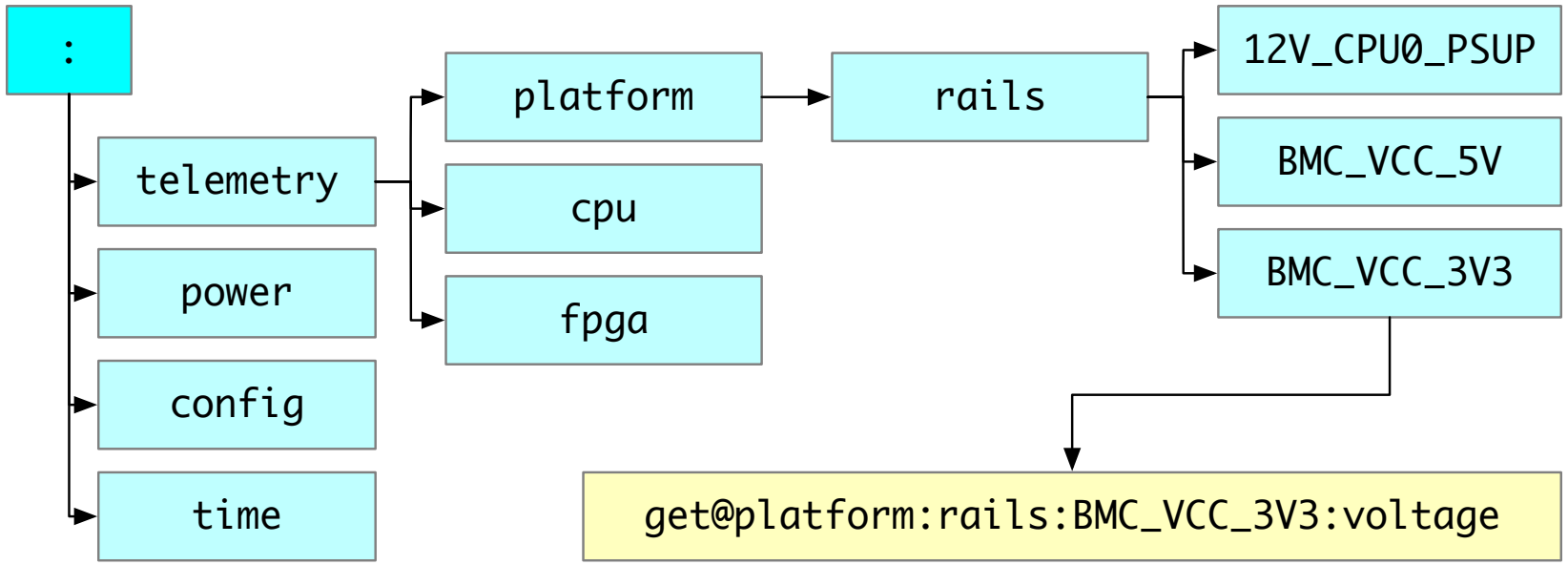
Symmetric client/actor model

Namespacing and Resource Discovery

- Firmware Resource Names (FRN)

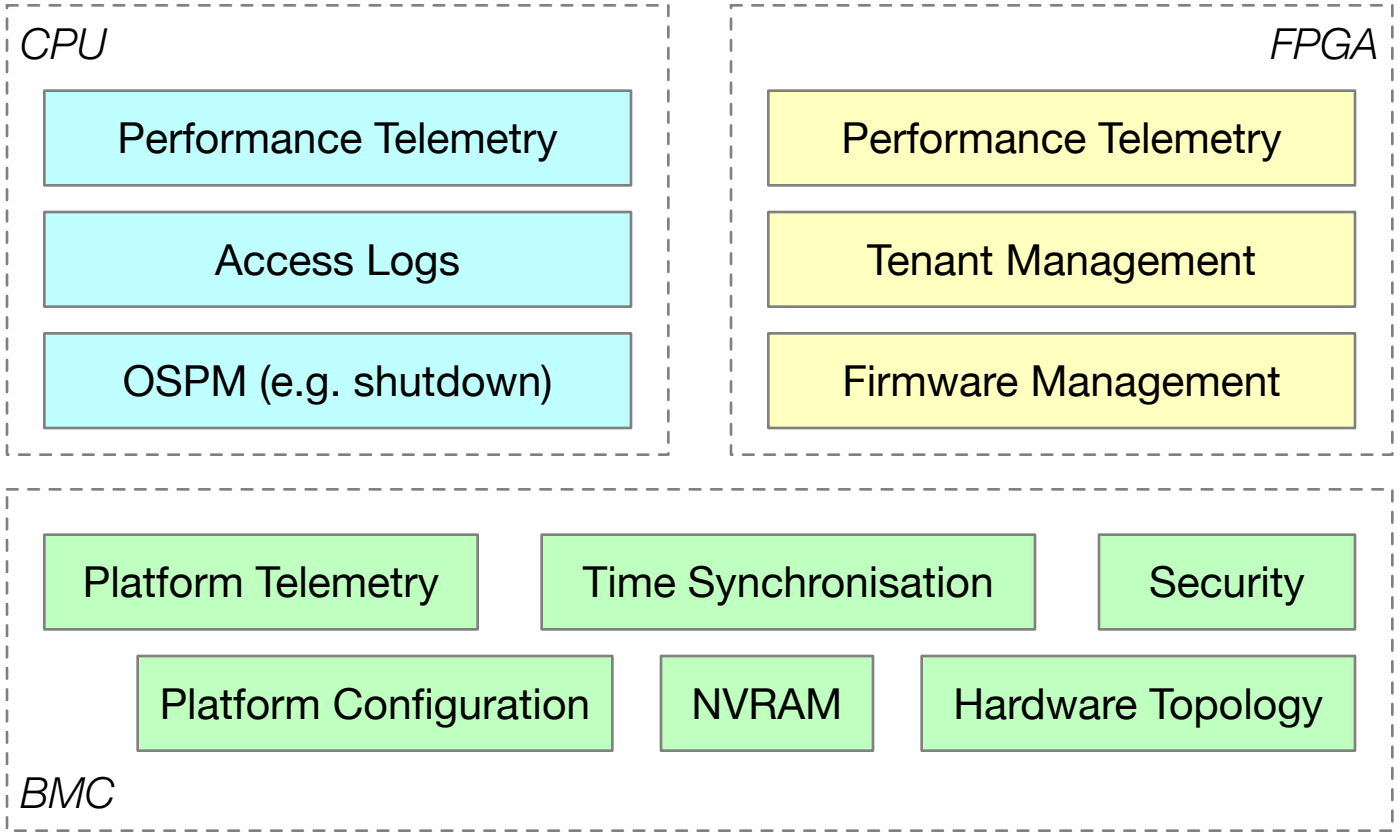
Namespace: `::telemetry` : `platform:rails:BMC_VCC_3V3:voltage`
(`::telemetry`: subsystem; `platform:rails:BMC_VCC_3V3:voltage`: topology)

Action: `get` @ `platform:rails:BMC_VCC_3V3:voltage`
(`get`: name; `platform:rails:BMC_VCC_3V3:voltage`: FRN)



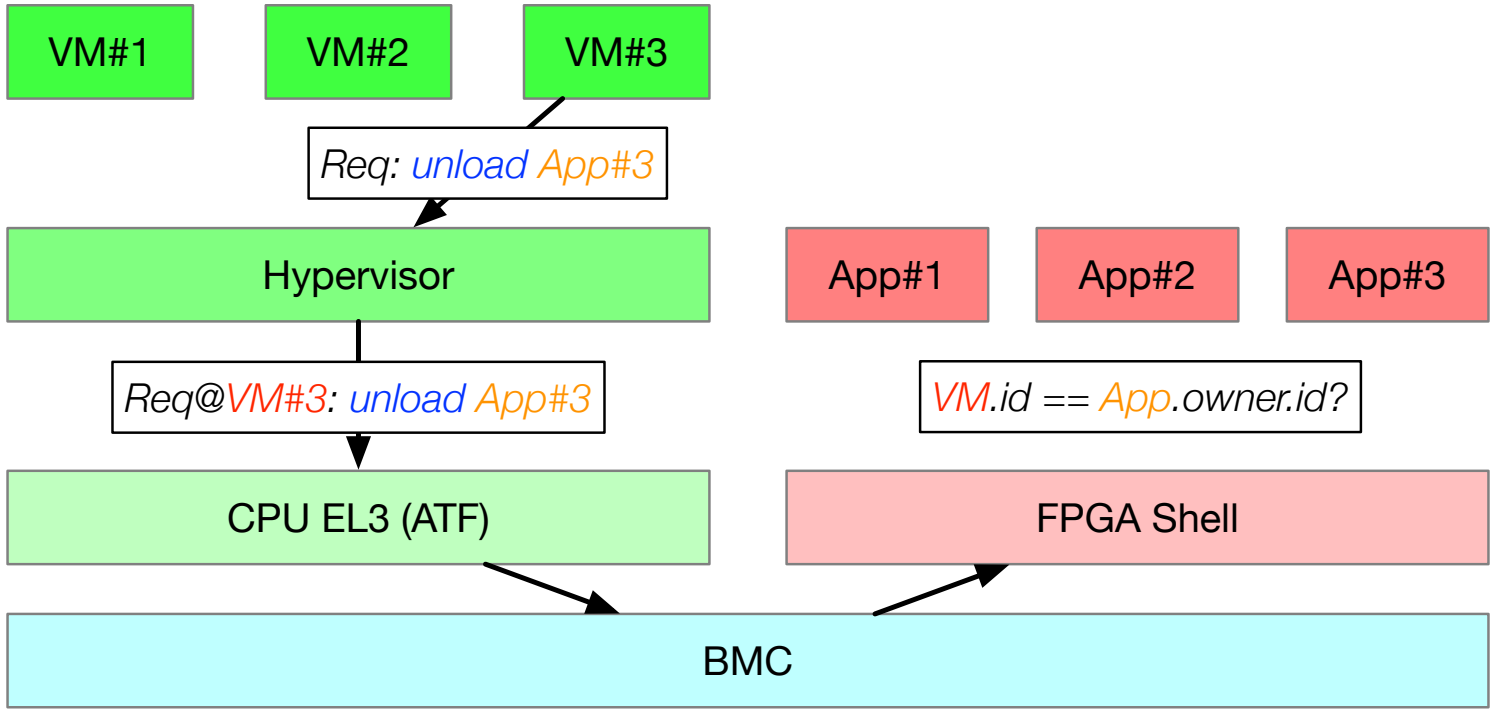
Symmetric Client/Actor Design

- Every hardware component in the system can be both a client and an actor



Security and Virtualisation (WIP)

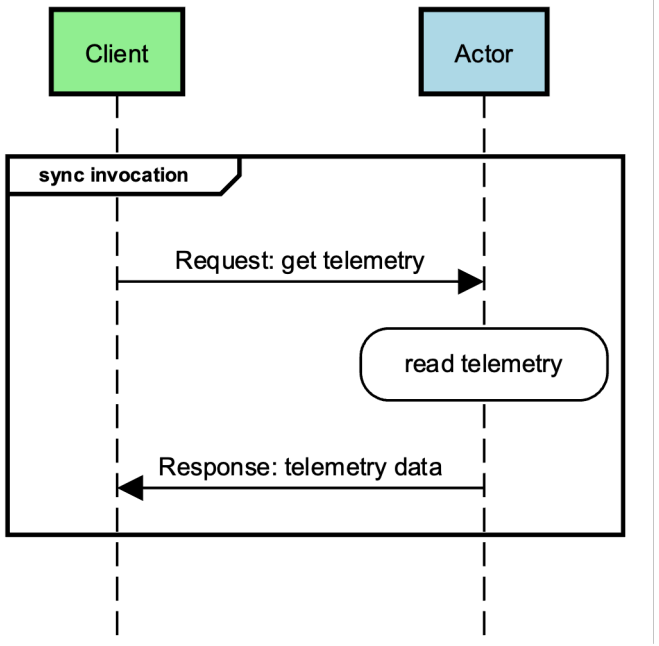
- Attack model: trust all **forwarding agents** along the way a request traverses
  - Permission check happens along this route



Asynchronous Event Mechanism (WIP)

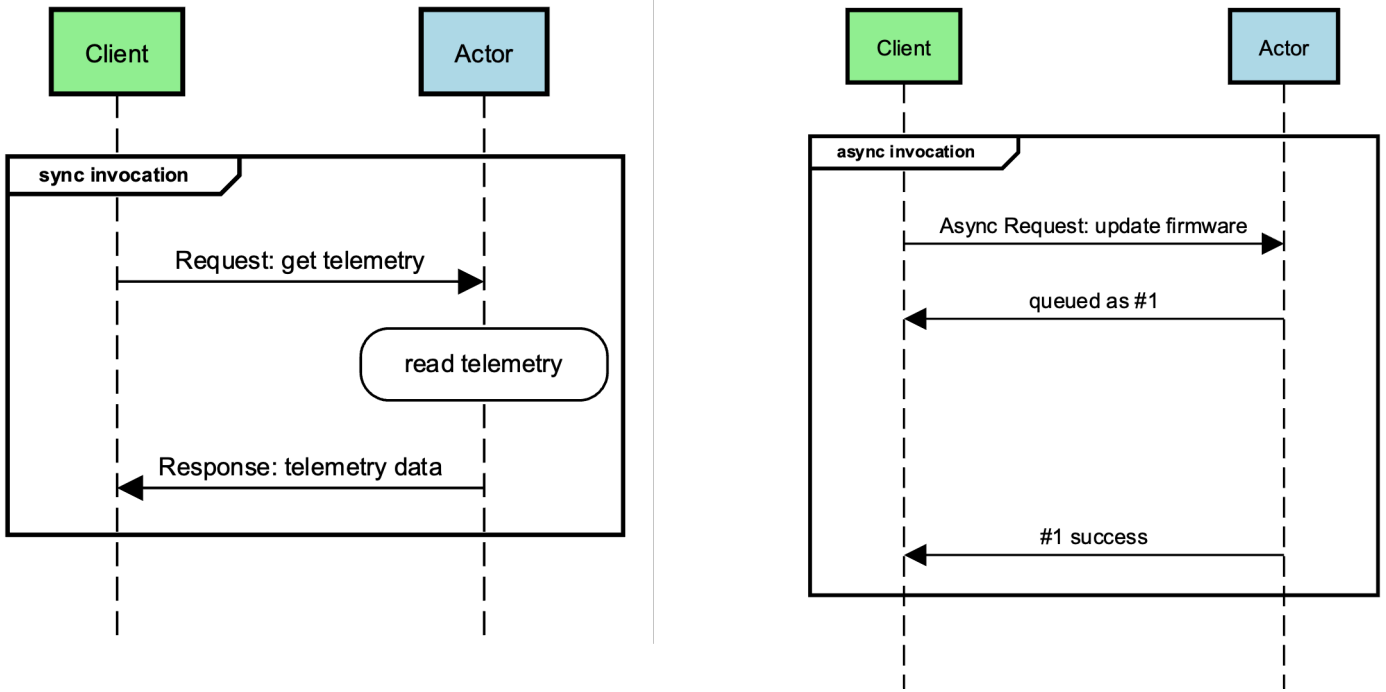
- Default synchronous due to in-band nature
  - Asynchrony useful for **slow** or **periodic** requests
  - Eliminates protocol overhead to generate request

Asynchronous Event Mechanism (WIP)



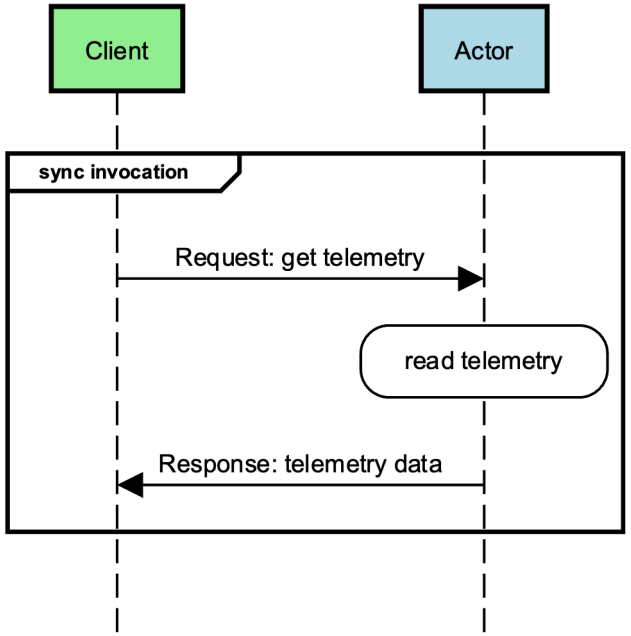
- Default synchronous due to in-band nature
  - Asynchrony useful for **slow** or **periodic** requests
  - Eliminates protocol overhead to generate request

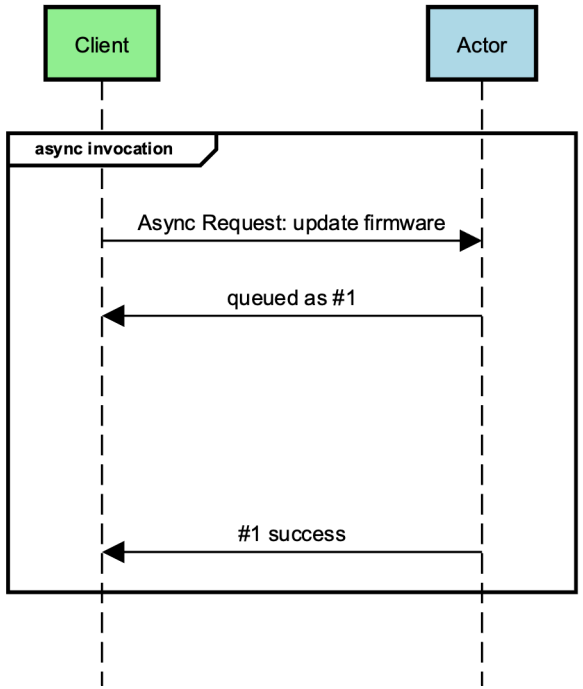
Asynchronous Event Mechanism (WIP)

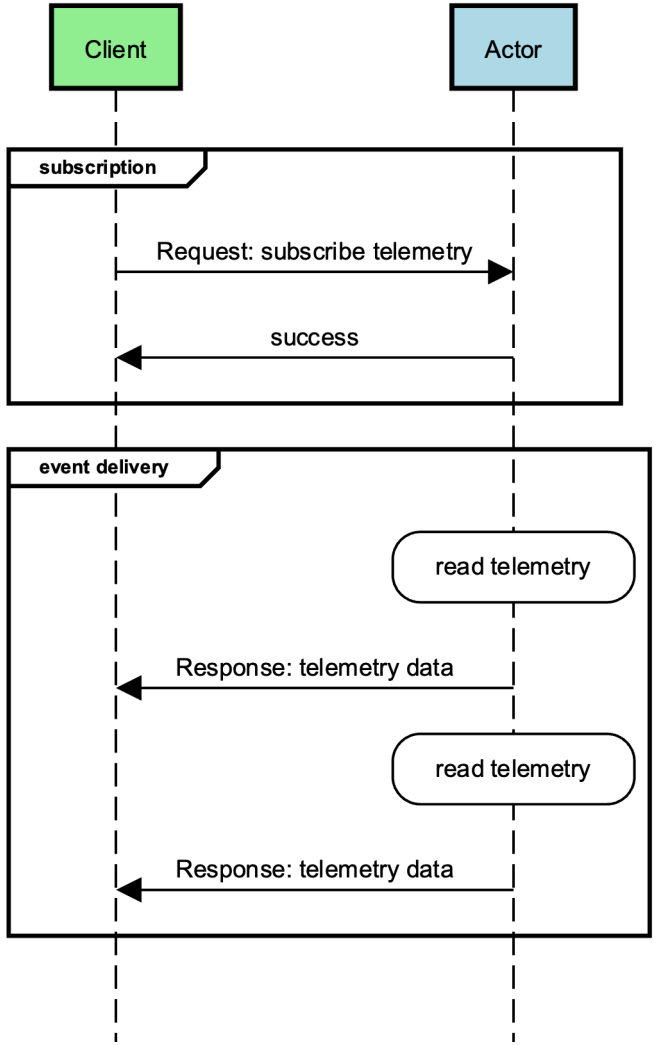


- Default synchronous due to in-band nature
  - Asynchrony useful for **slow** or **periodic** requests
  - Eliminates protocol overhead to generate request

Asynchronous Event Mechanism (WIP)







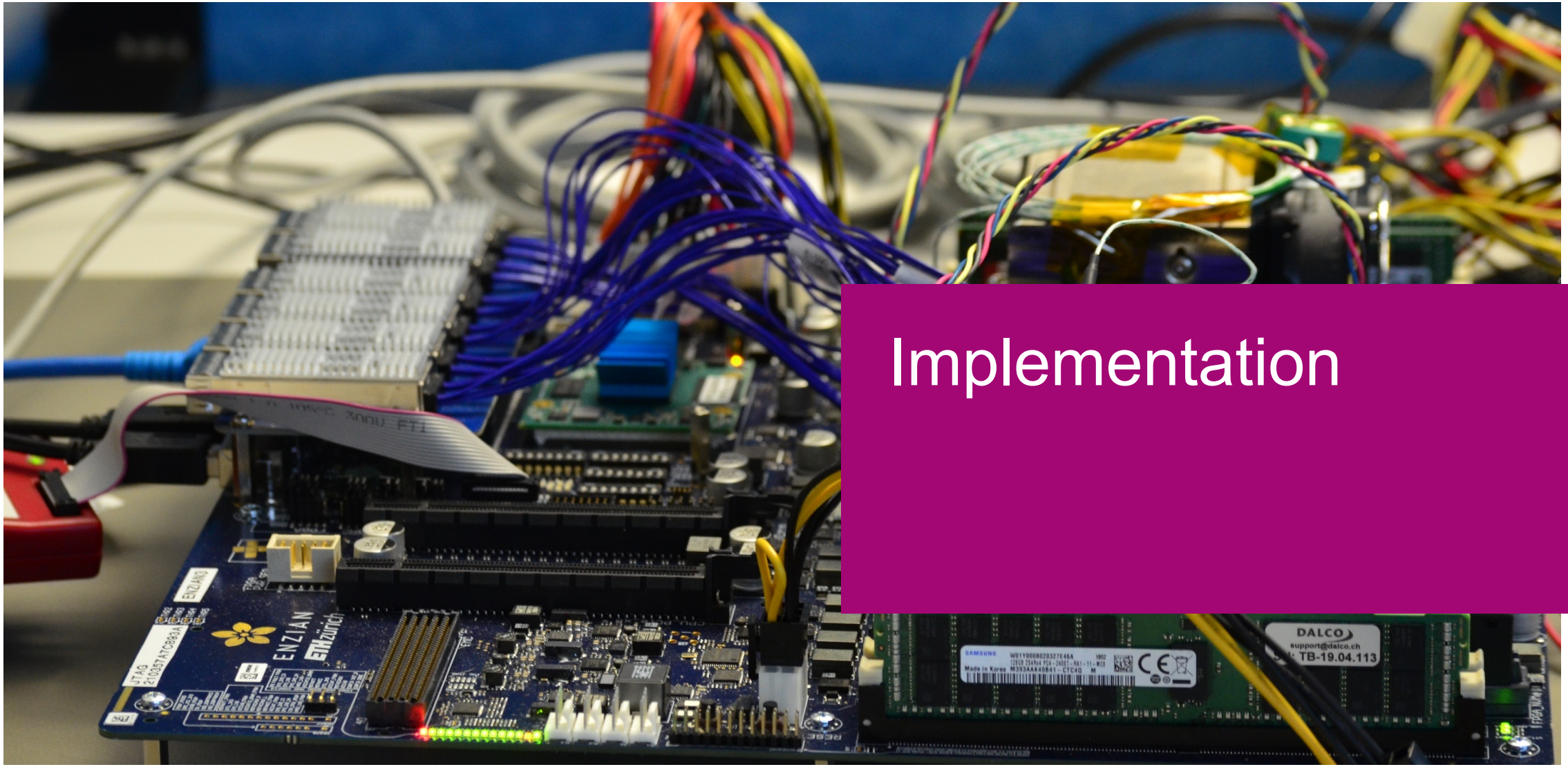
- Default synchronous due to in-band nature
  - Asynchrony useful for **slow** or **periodic** requests
  - Eliminates protocol overhead to generate request

ETH zürich

ENZIAN

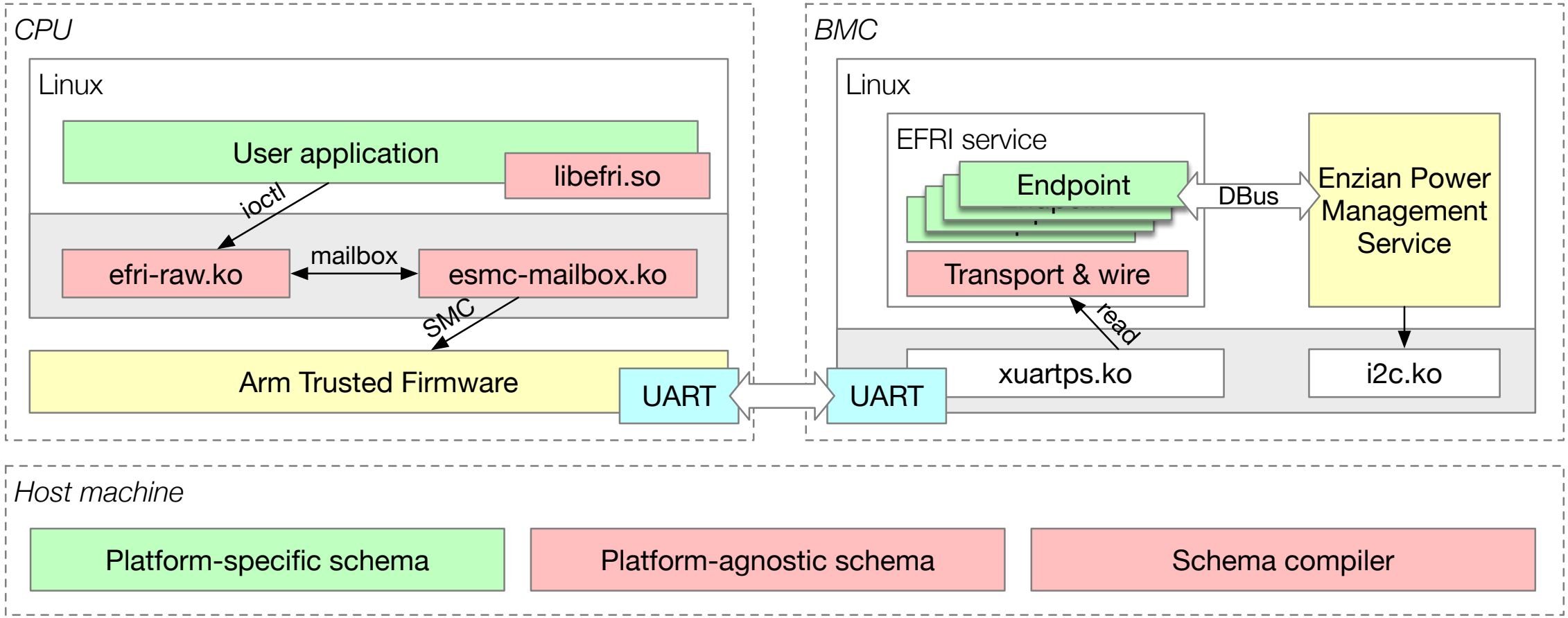
Systems@ETH zürich

DINFK



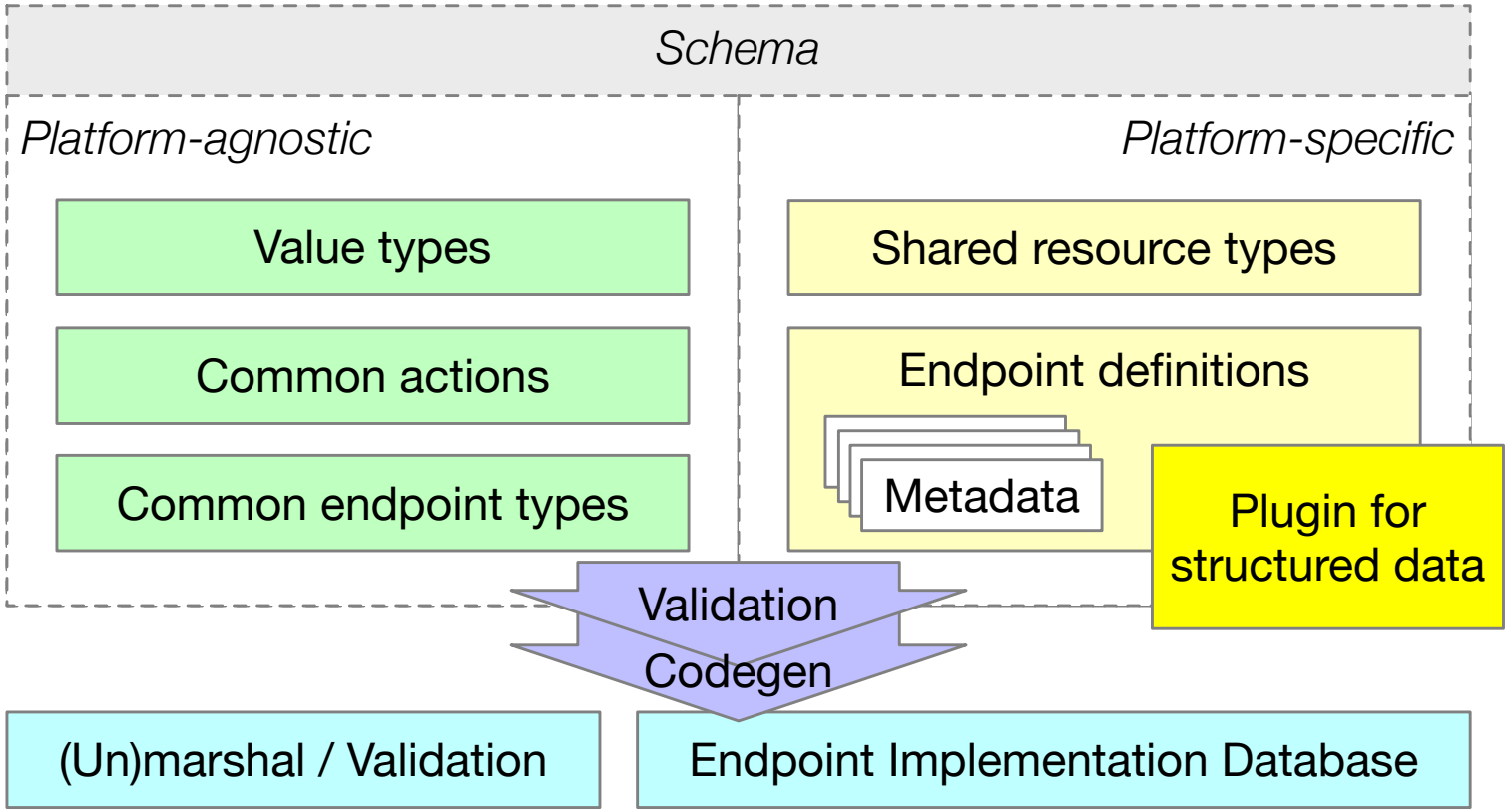
Implementation

Reference implementation overview

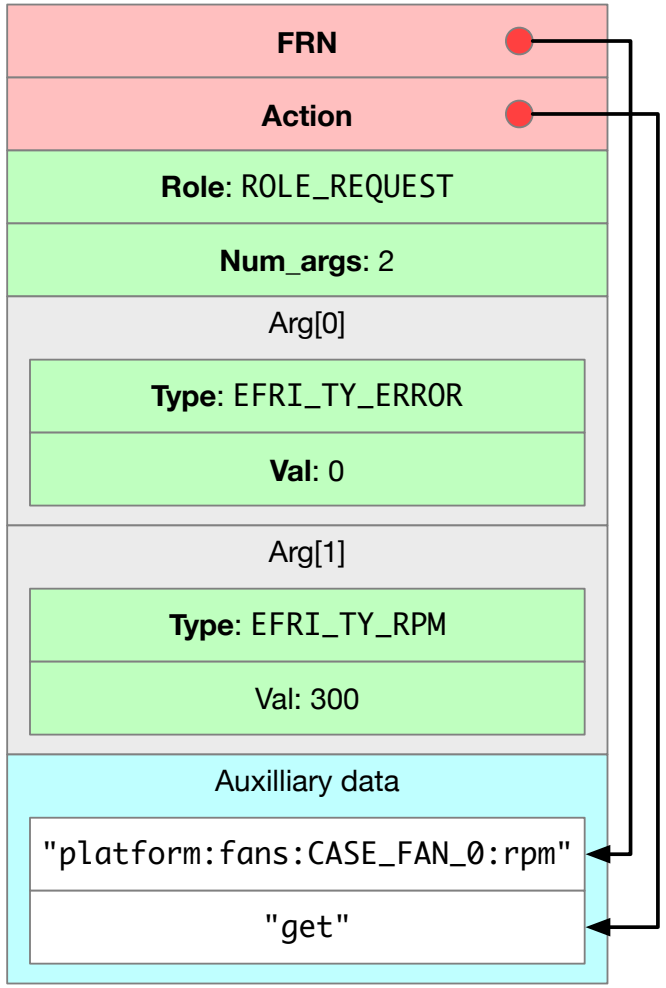


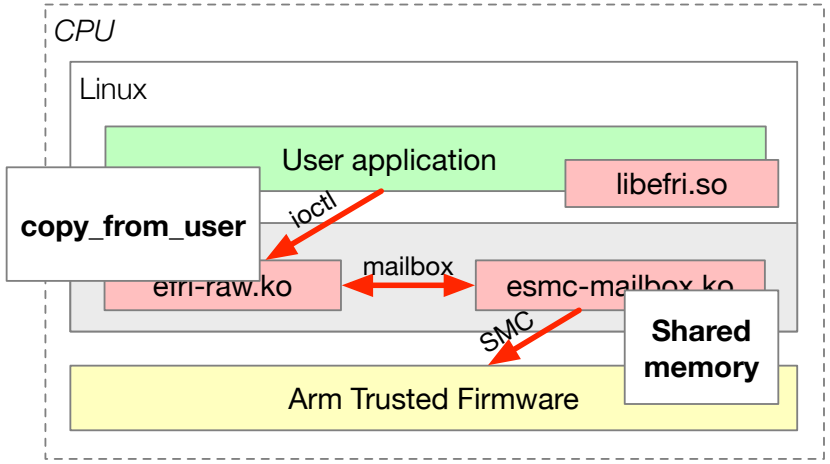
The schema compiler

- Generate boilerplate code from YAML schema



Implementation on the CPU

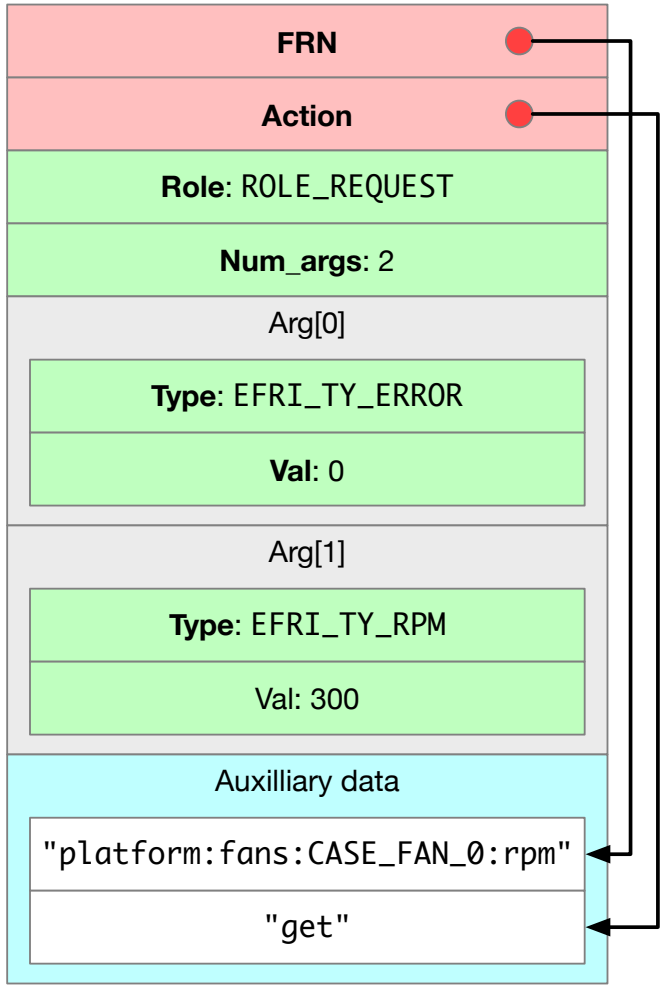


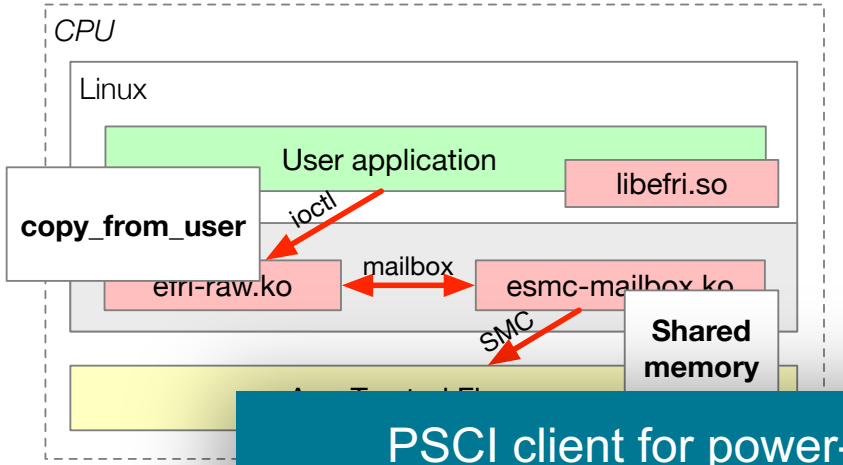



```
==EFRI== =>  platform:fans:CASE_FAN_0:rpm  get \n
request preamble        FRN                action

==EFRI== <=  platform:fans:CASE_FAN_0:rpm  get  Error$0  Rpm$300#RPM  \n
response preamble       FRN                action error code  value
```

Implementation on the CPU



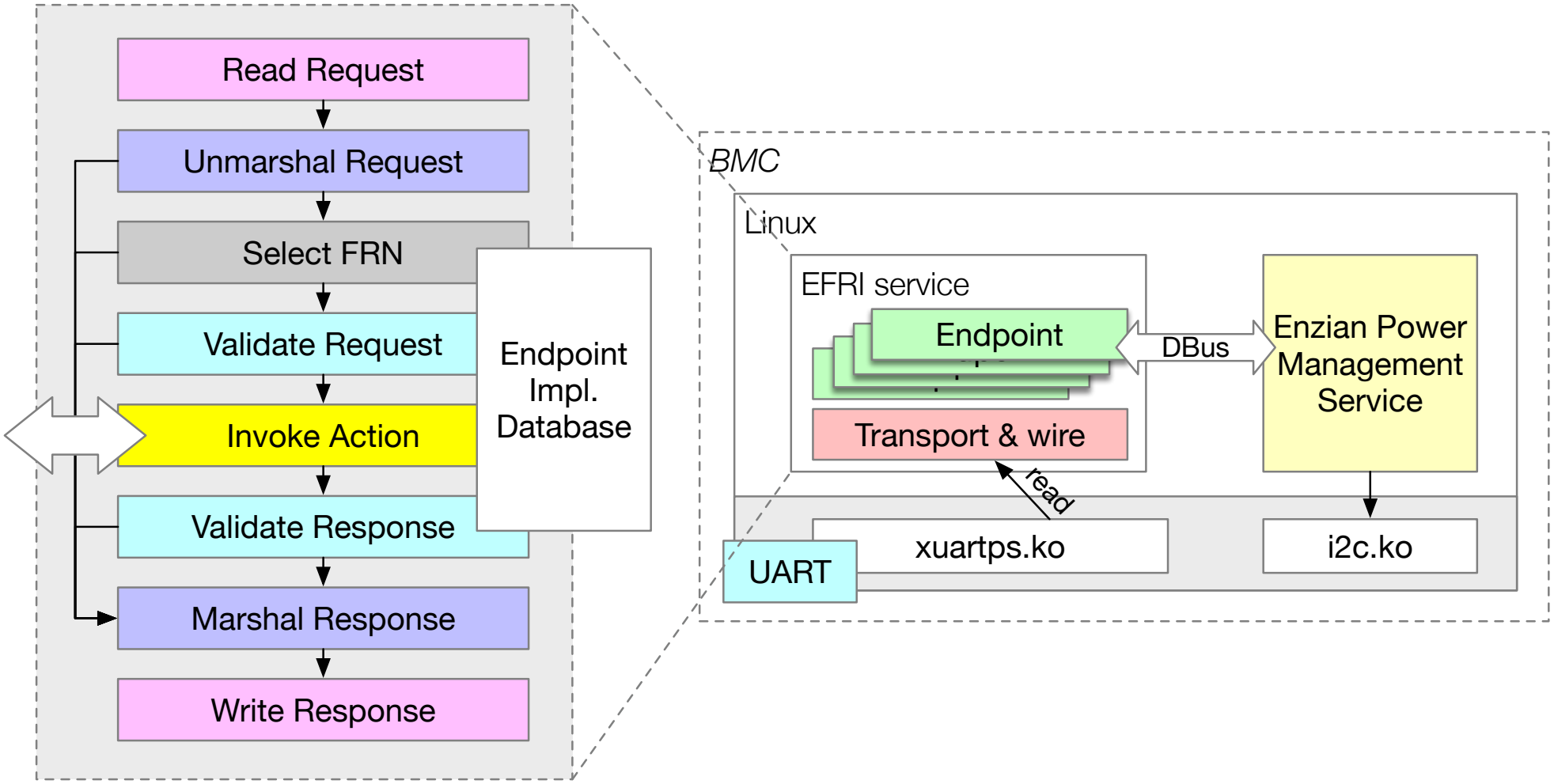


PSCI client for power-off!
DRAM parameters, etc.

```
==EFRI== =>  platform:fans:CASE_FAN_0:rpm  get \n
request preamble  FRN  action

==EFRI== <=  platform:fans:CASE_FAN_0:rpm  get  Error$0  Rpm$300#RPM  \n
response preamble  FRN  action  error code  value
```

Implementation on the BMC

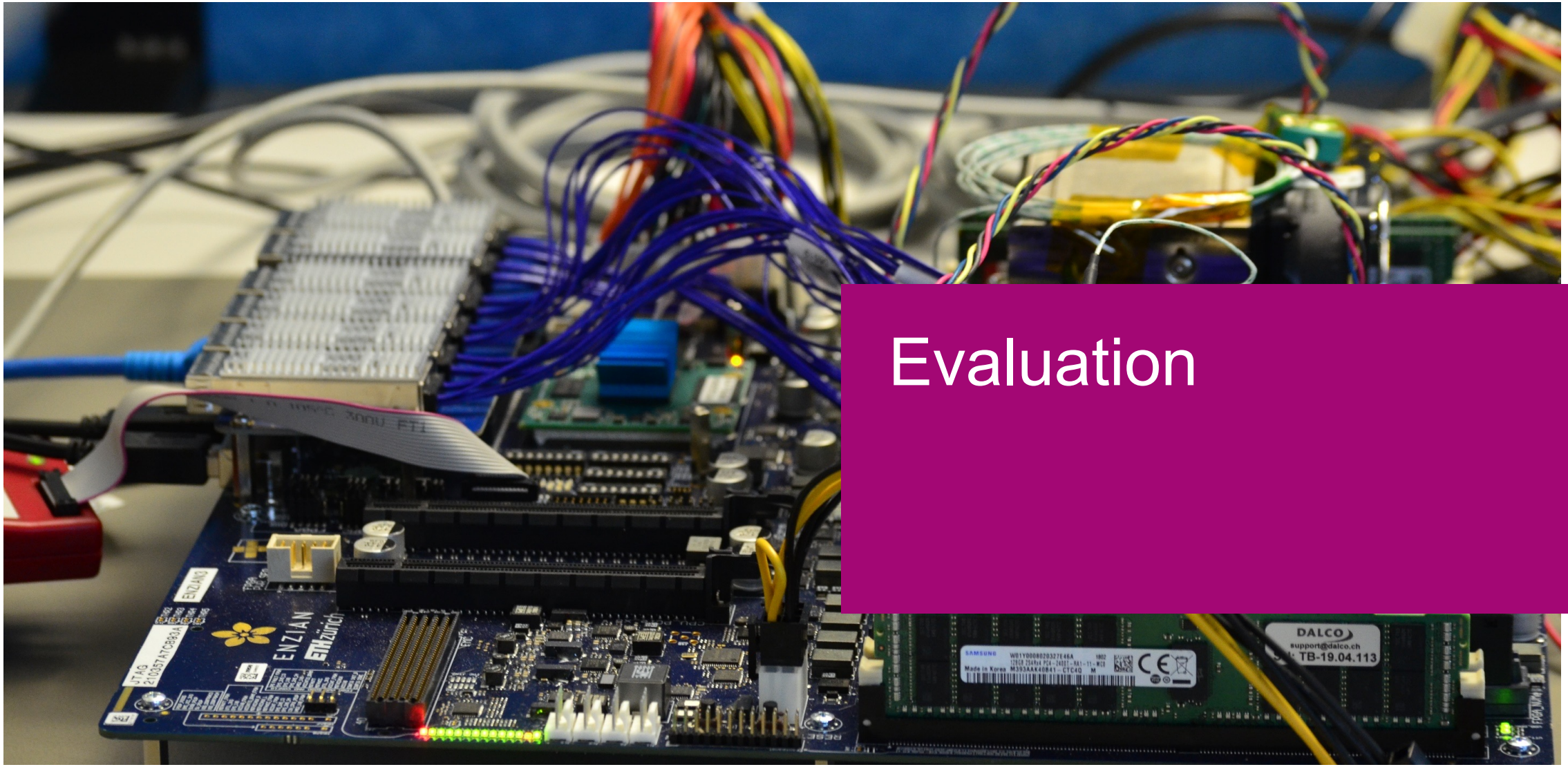


ETH zürich

ENZIAN

Systems@ETH zürich

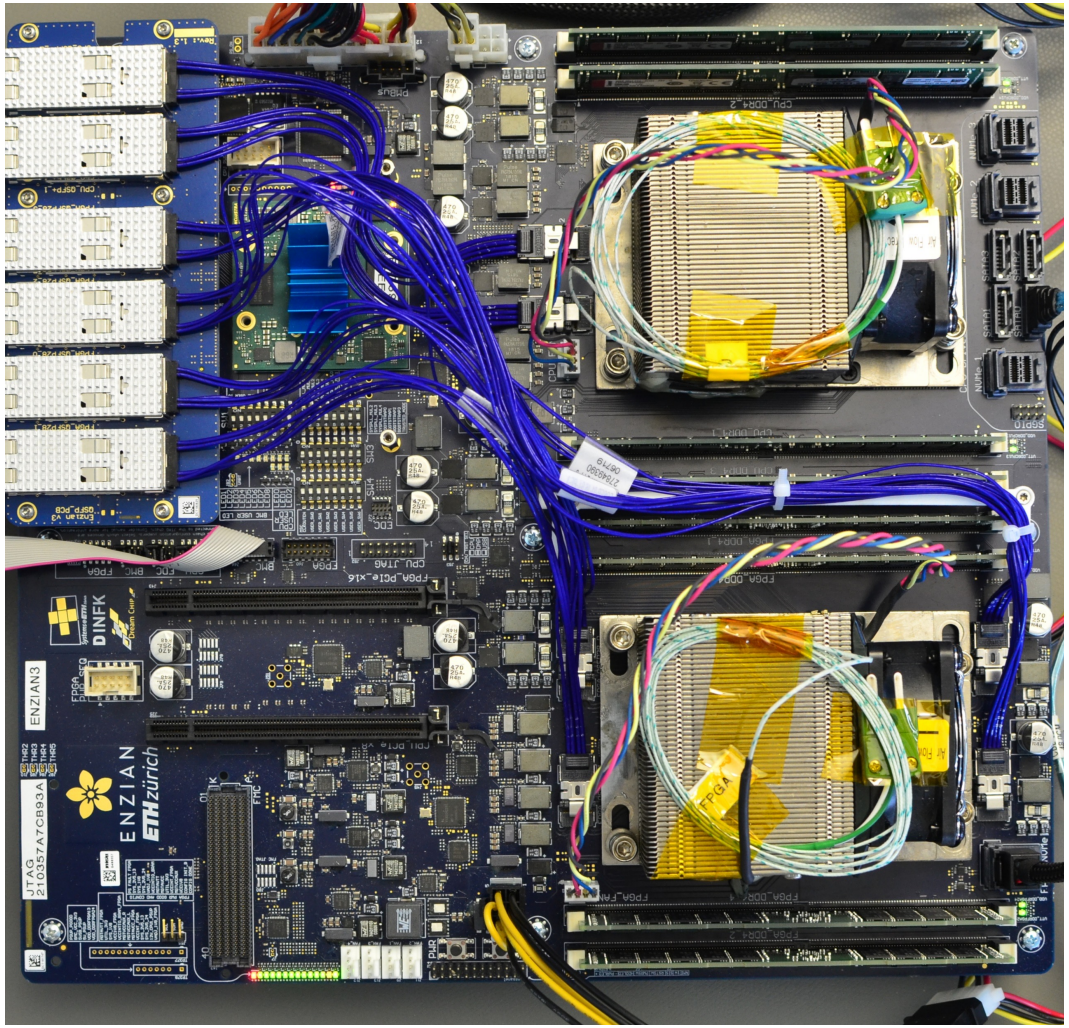
DINFK



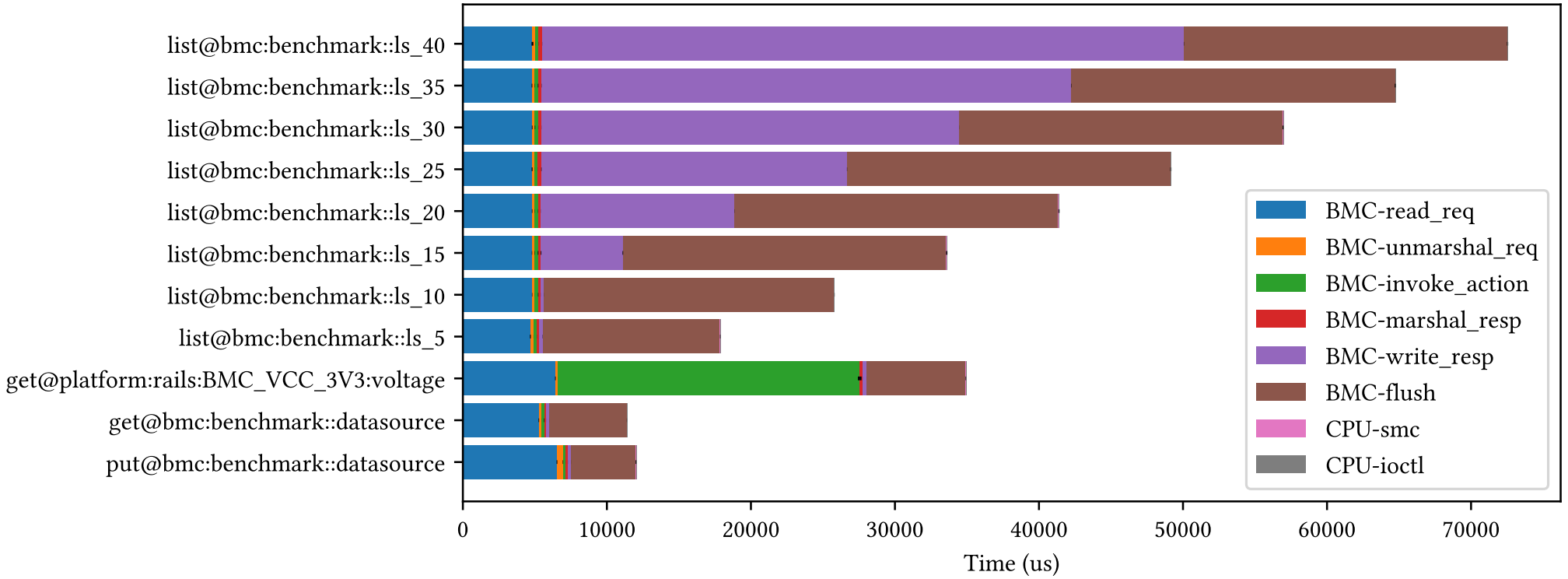
Evaluation

Experiments Platform

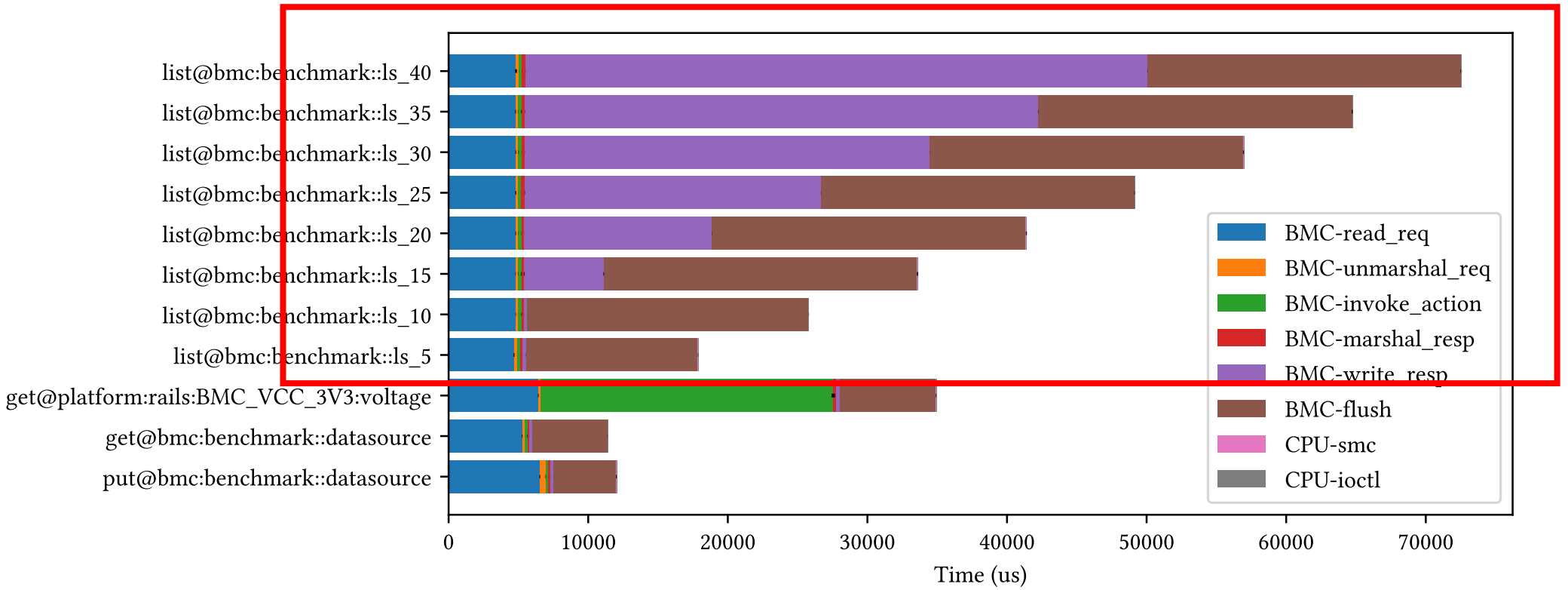
- Enzian v3 “zuestoll”
  - ThunderX-1, Zynq-7000 BMC
- 115200 8n1 for the serial link
- Toolchain:
  - Schema compiler generates ATF and BMC code
  - OpenBMC + Python 3
  - ATF codebase version unknown (from Cavium)
- Codebase augmented to collect timestamps at various stage of processing



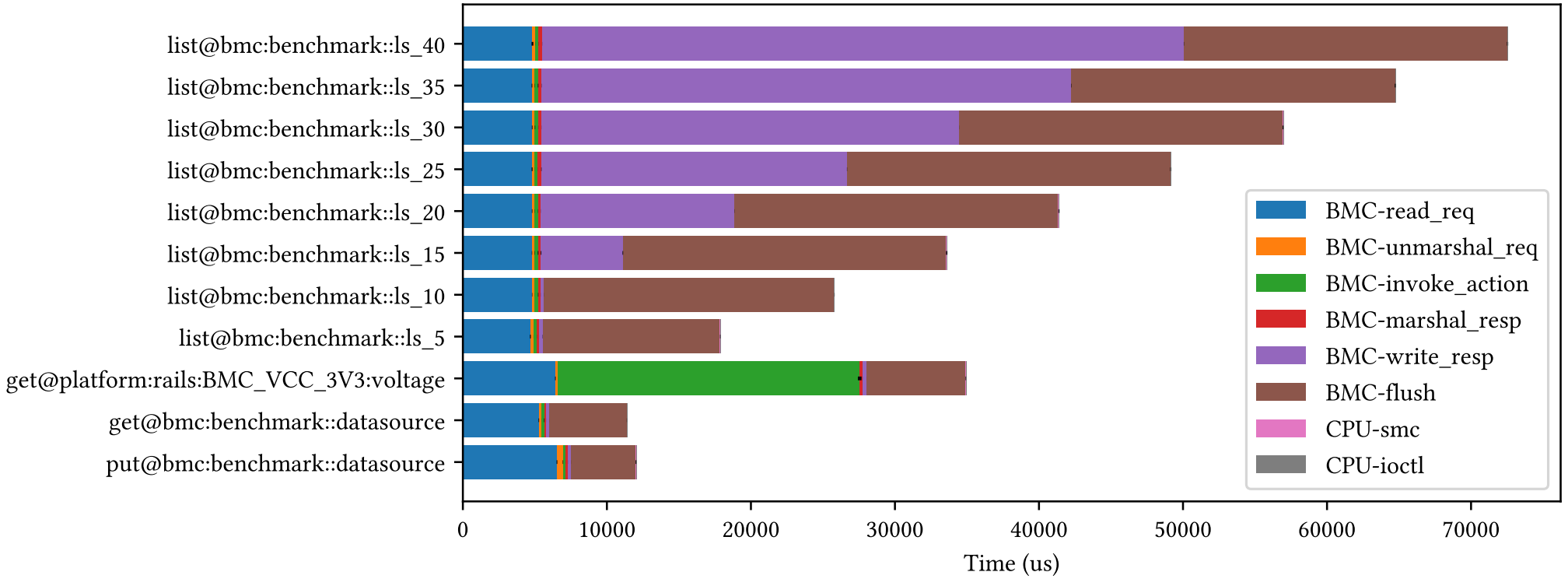
Performance – latency breakdown



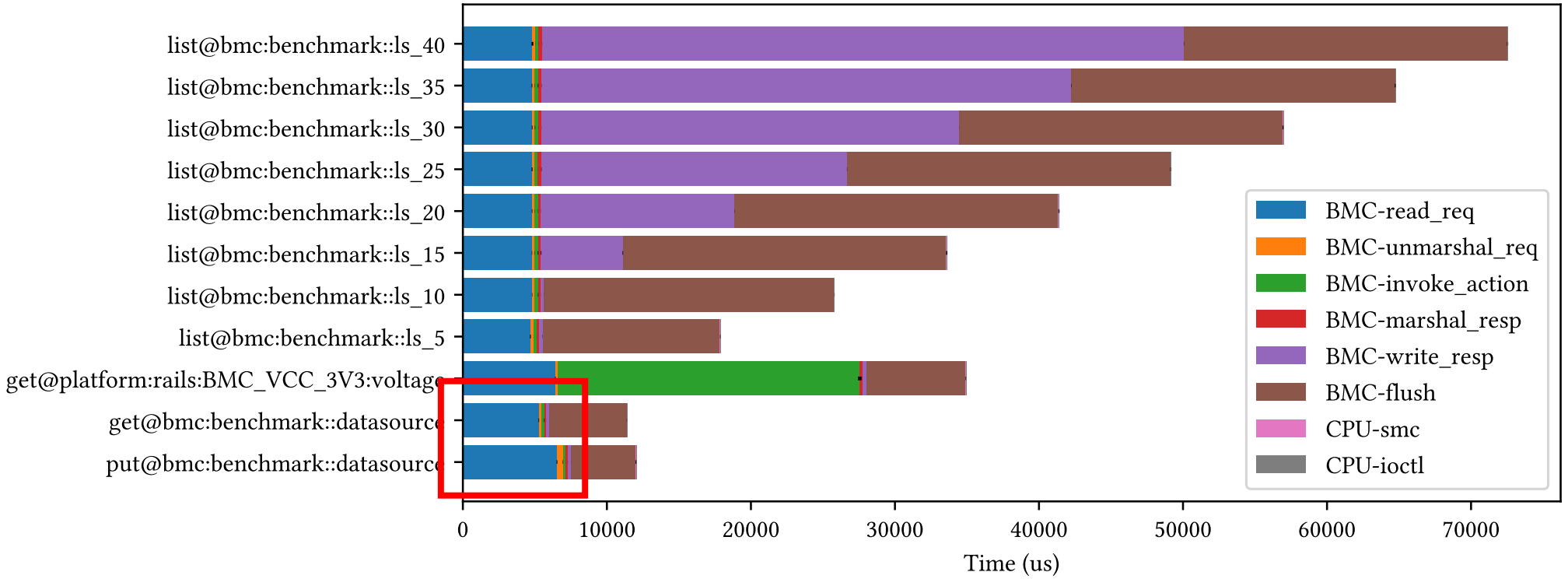
Performance – latency breakdown



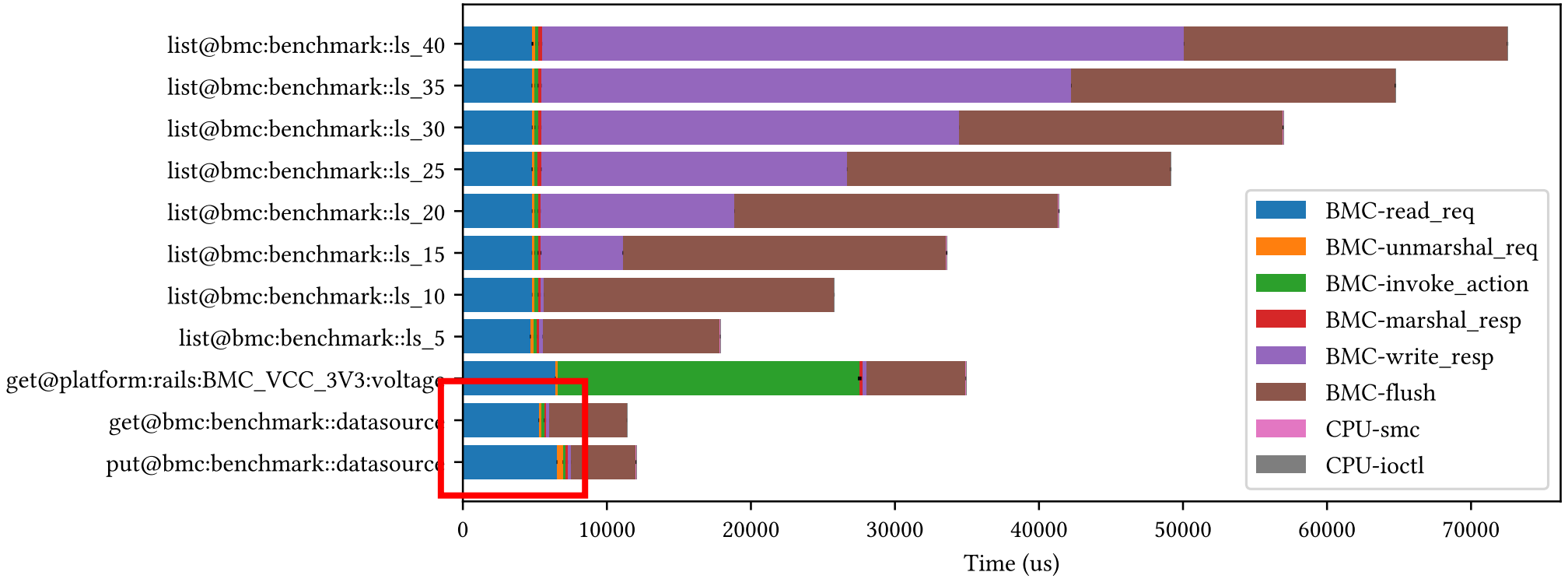
Performance – latency breakdown



Performance – latency breakdown

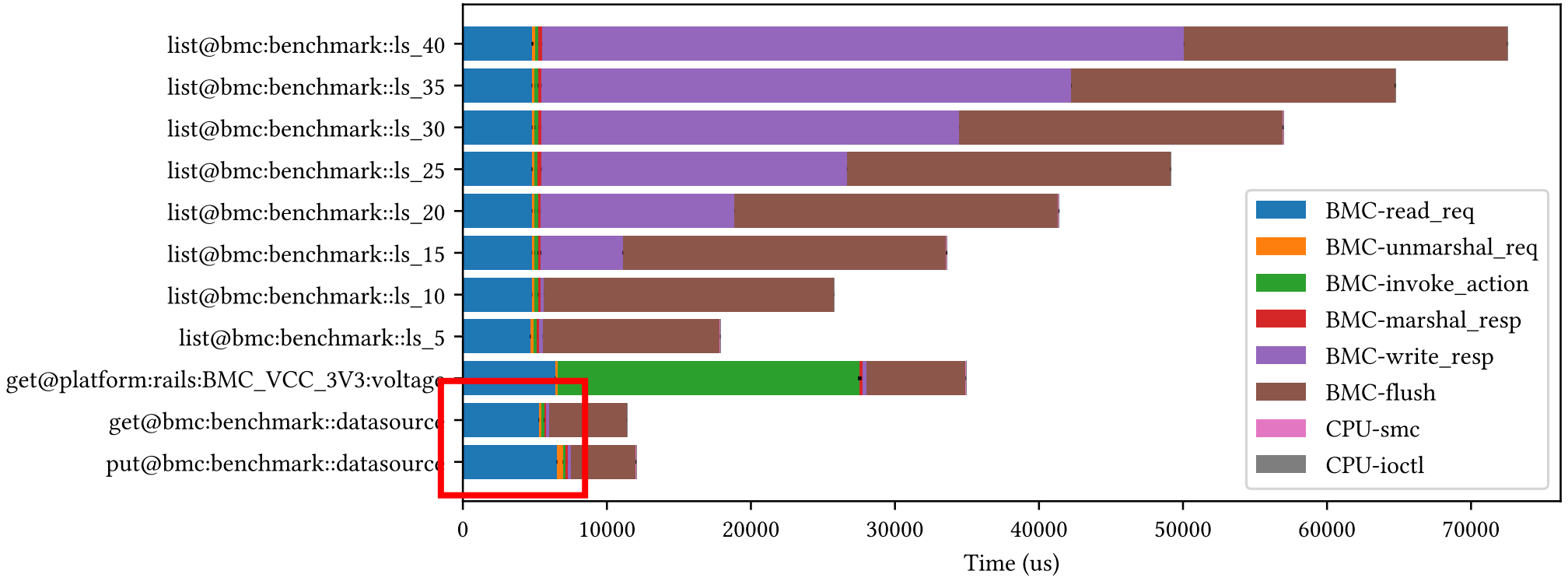


Performance – latency breakdown




```
==EFRI== => bmc:benchmark::datasource get\n
==EFRI== => bmc:benchmark::datasource put Rpm$5#RPM\n
```

Performance – latency breakdown

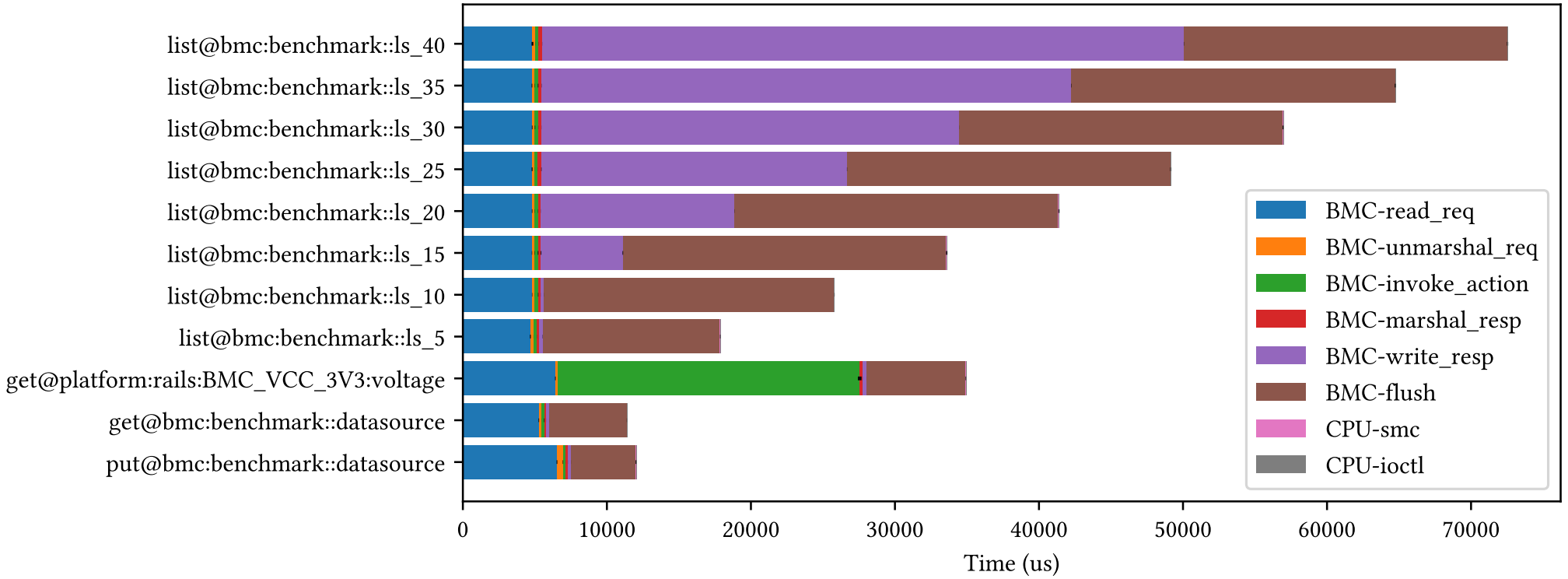



```
==EFRI== => bmc:benchmark::datasource get\n
==EFRI== => bmc:benchmark::datasource put Rpm$5#RPM\n
```

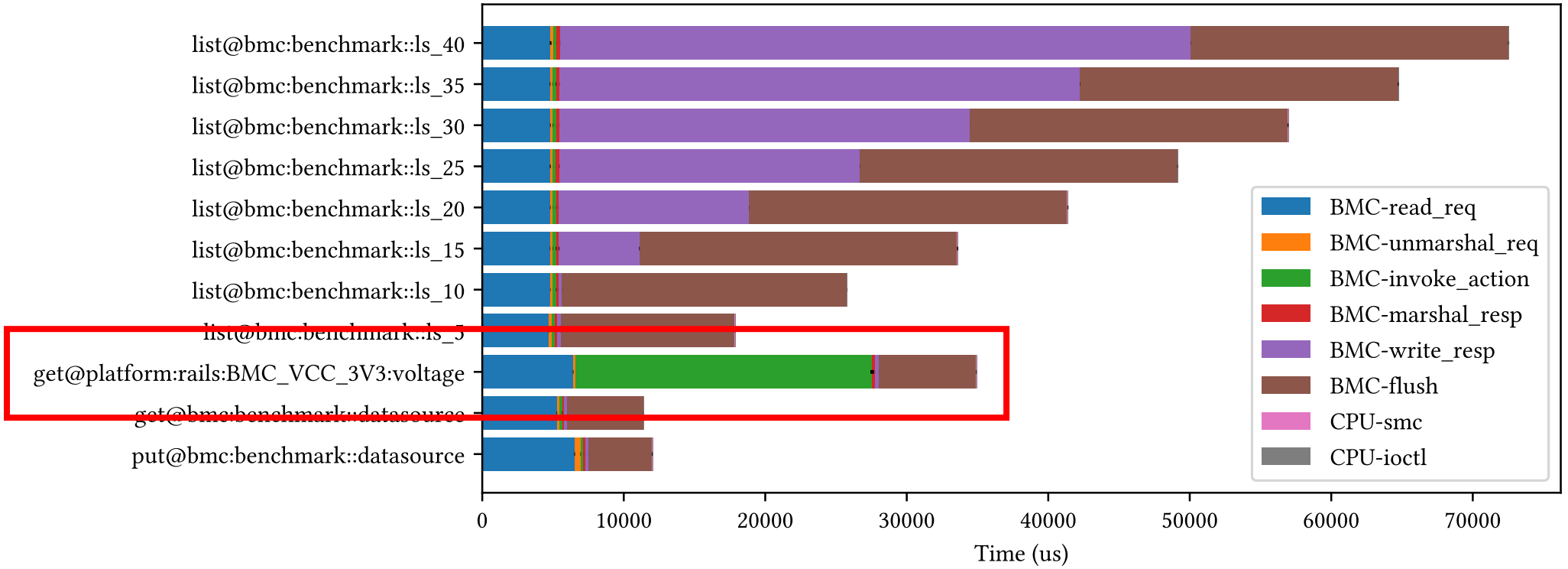
42 vs 52 bytes

120us/byte (UART: 86us/byte)

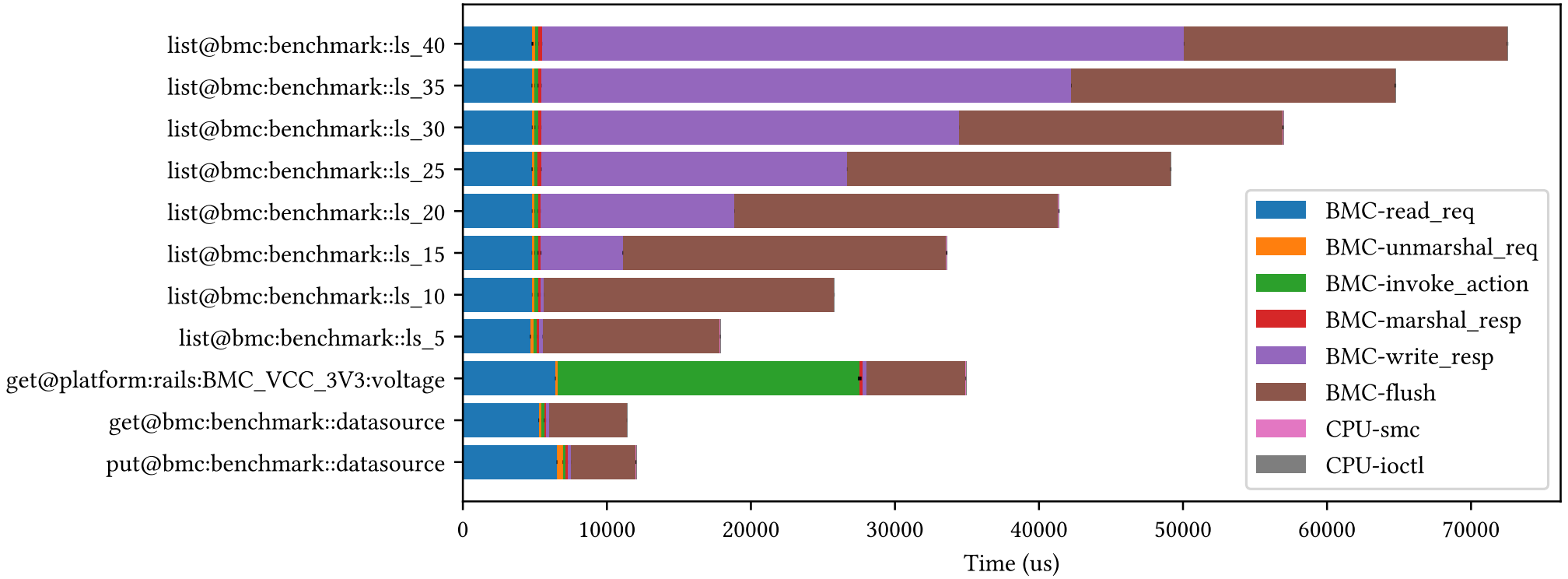
Performance – latency breakdown



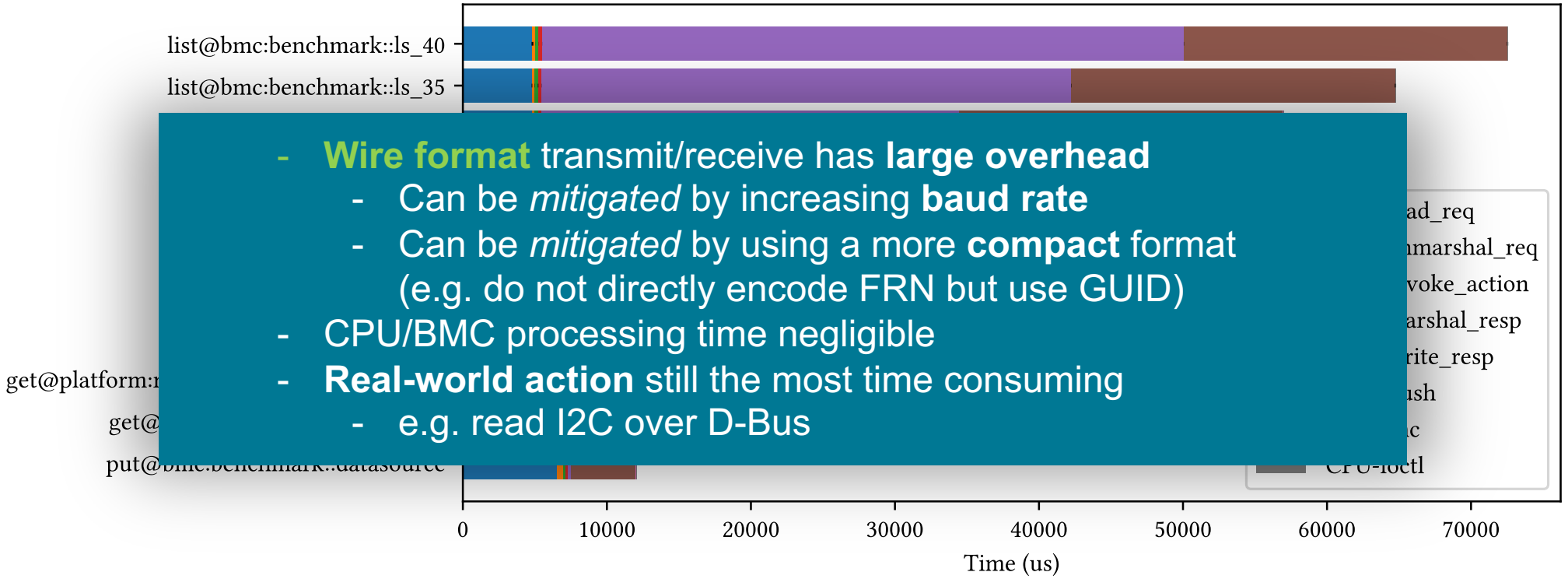
Performance – latency breakdown



Performance – latency breakdown



Performance – latency breakdown



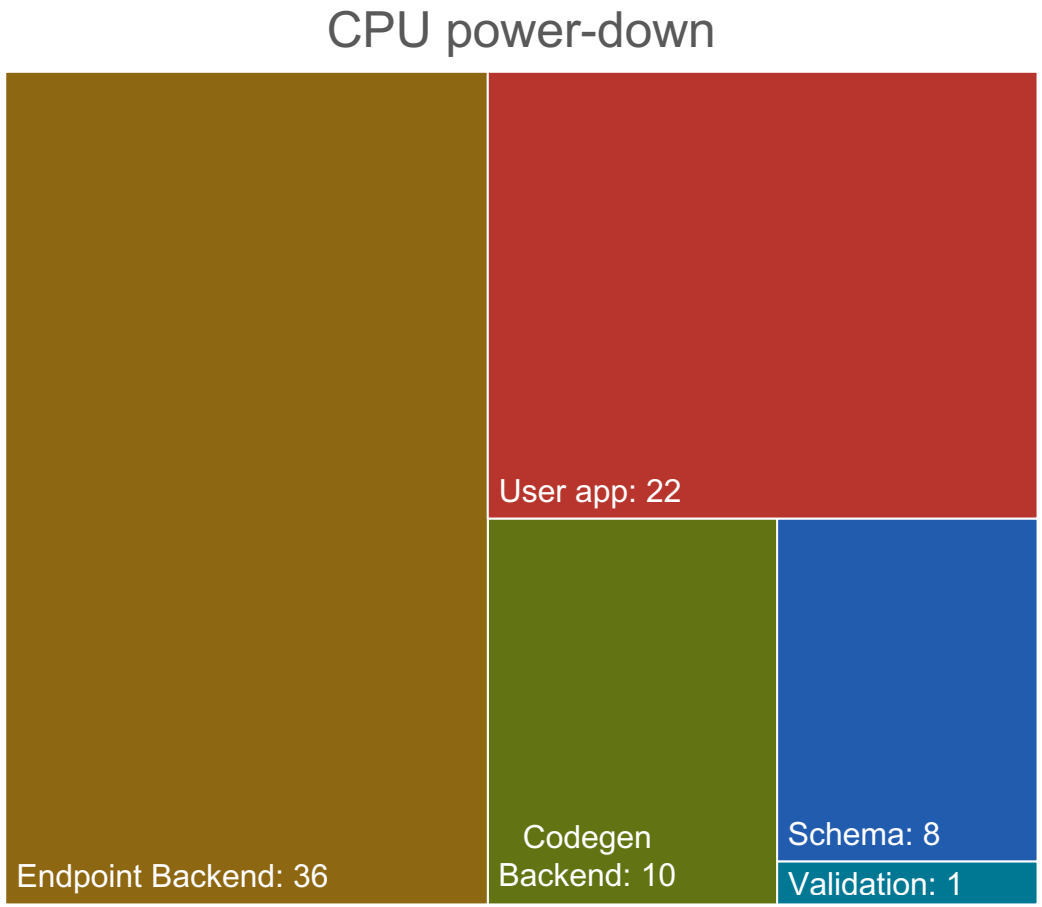
- **Wire format** transmit/receive has **large overhead**
  - Can be *mitigated* by increasing **baud rate**
  - Can be *mitigated* by using a more **compact** format (e.g. do not directly encode FRN but use GUID)
- CPU/BMC processing time negligible
- **Real-world action** still the most time consuming
  - e.g. read I2C over D-Bus

Case studies for ease of extension

- Measure LOC changes for implementing a specific firmware feature

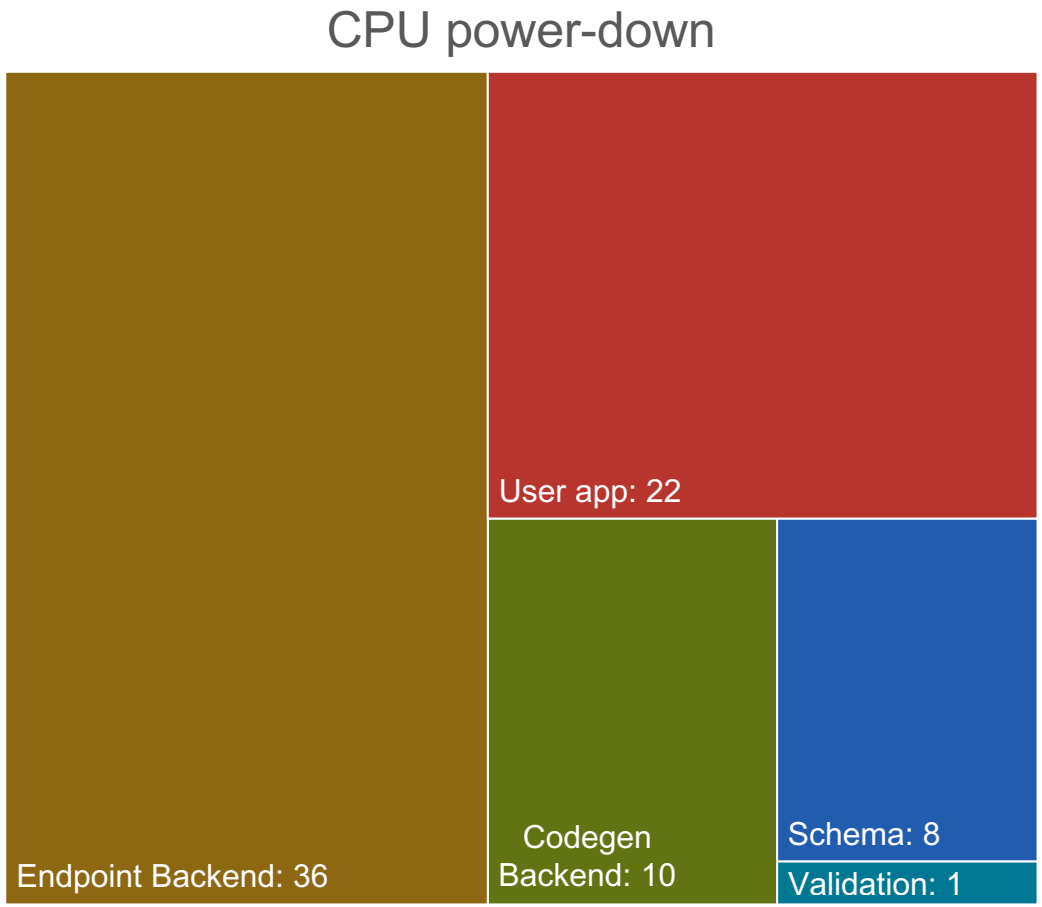
Case studies for ease of extension

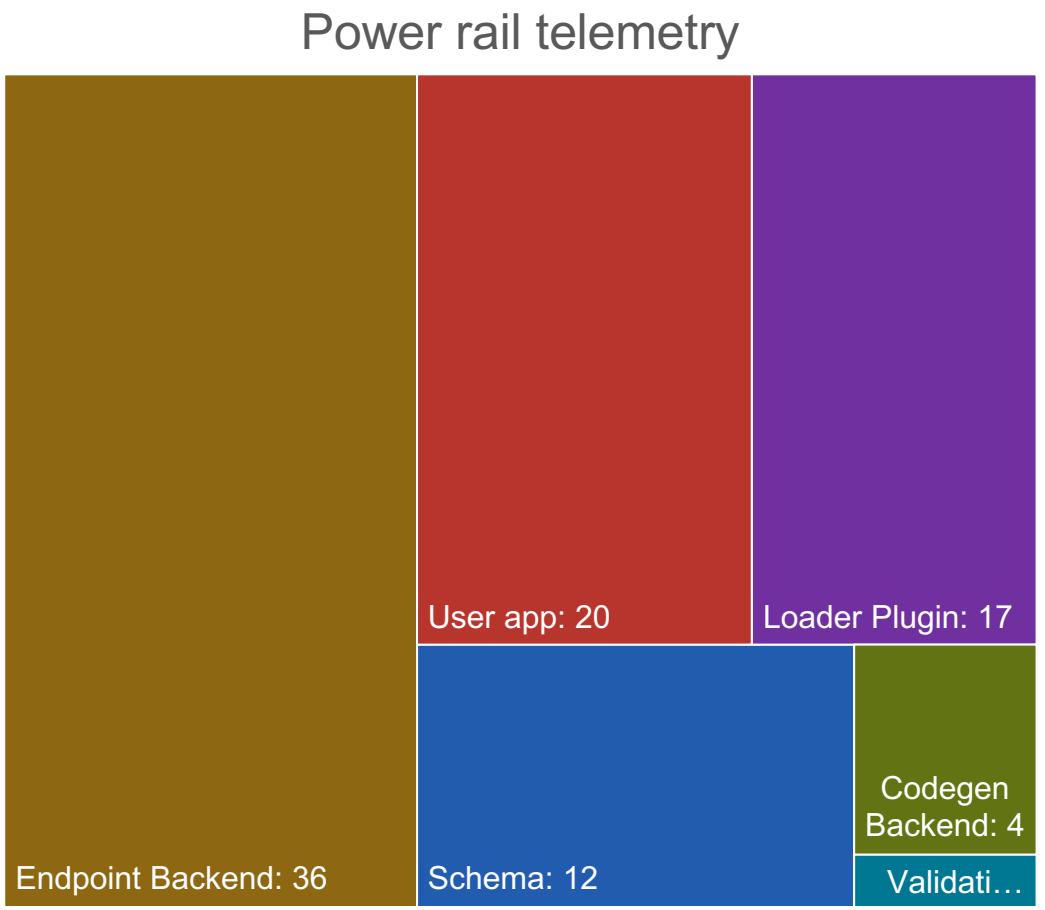
- Measure LOC changes for implementing a specific firmware feature



Case studies for ease of extension

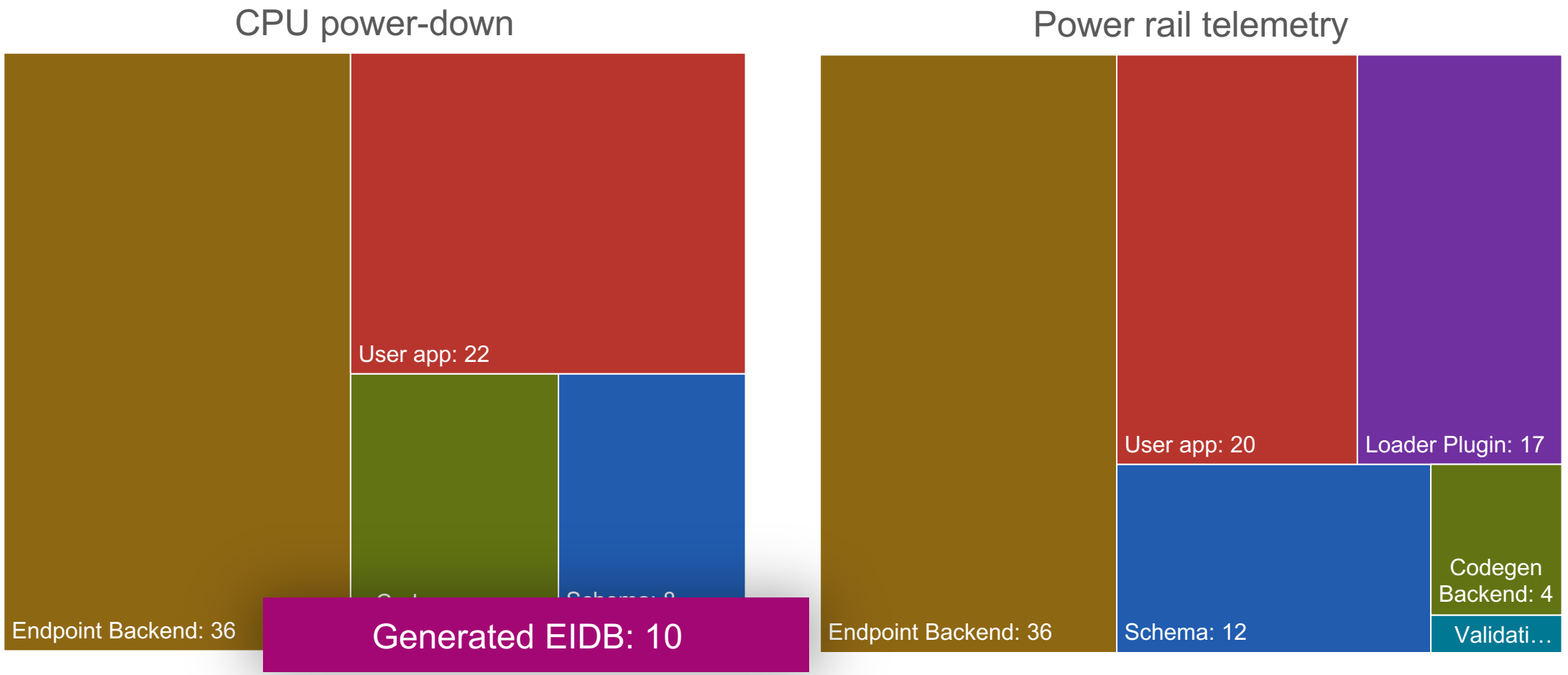
- Measure LOC changes for implementing a specific firmware feature





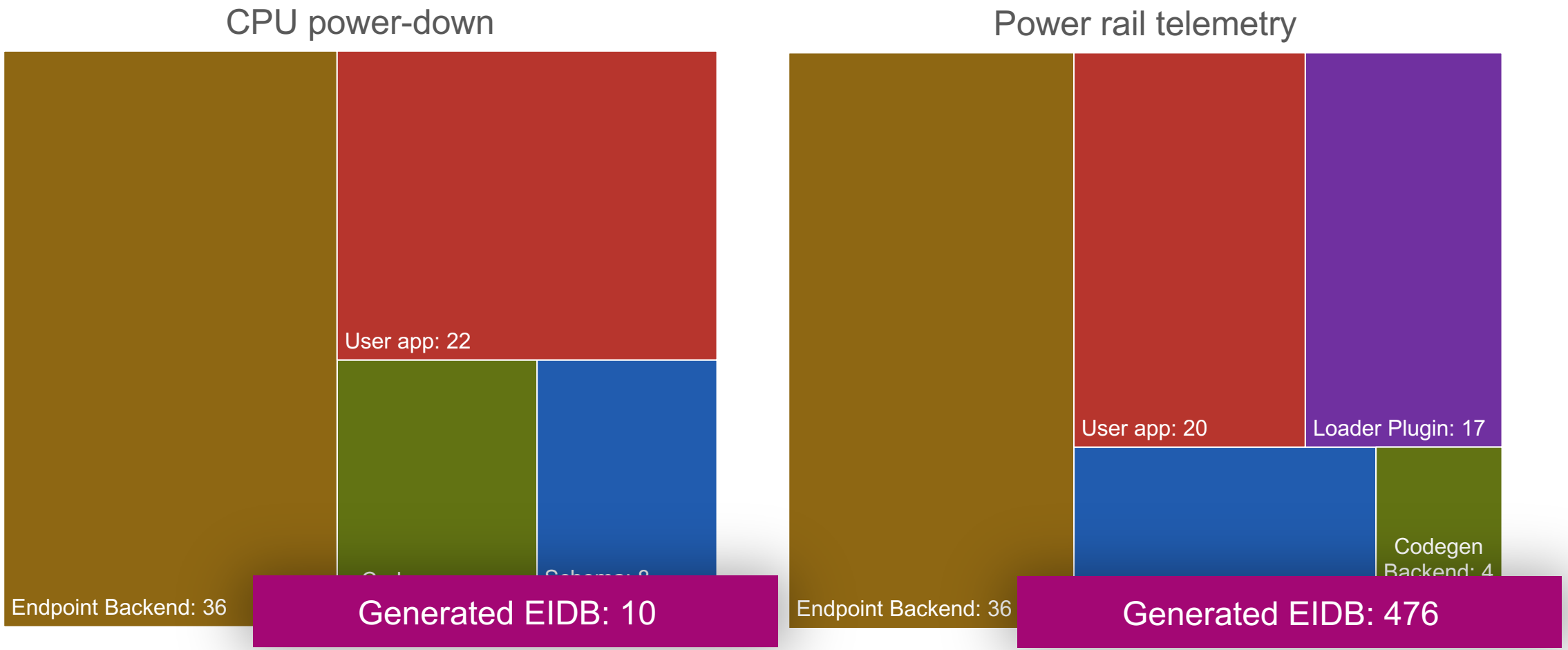
Case studies for ease of extension

- Measure LOC changes for implementing a specific firmware feature



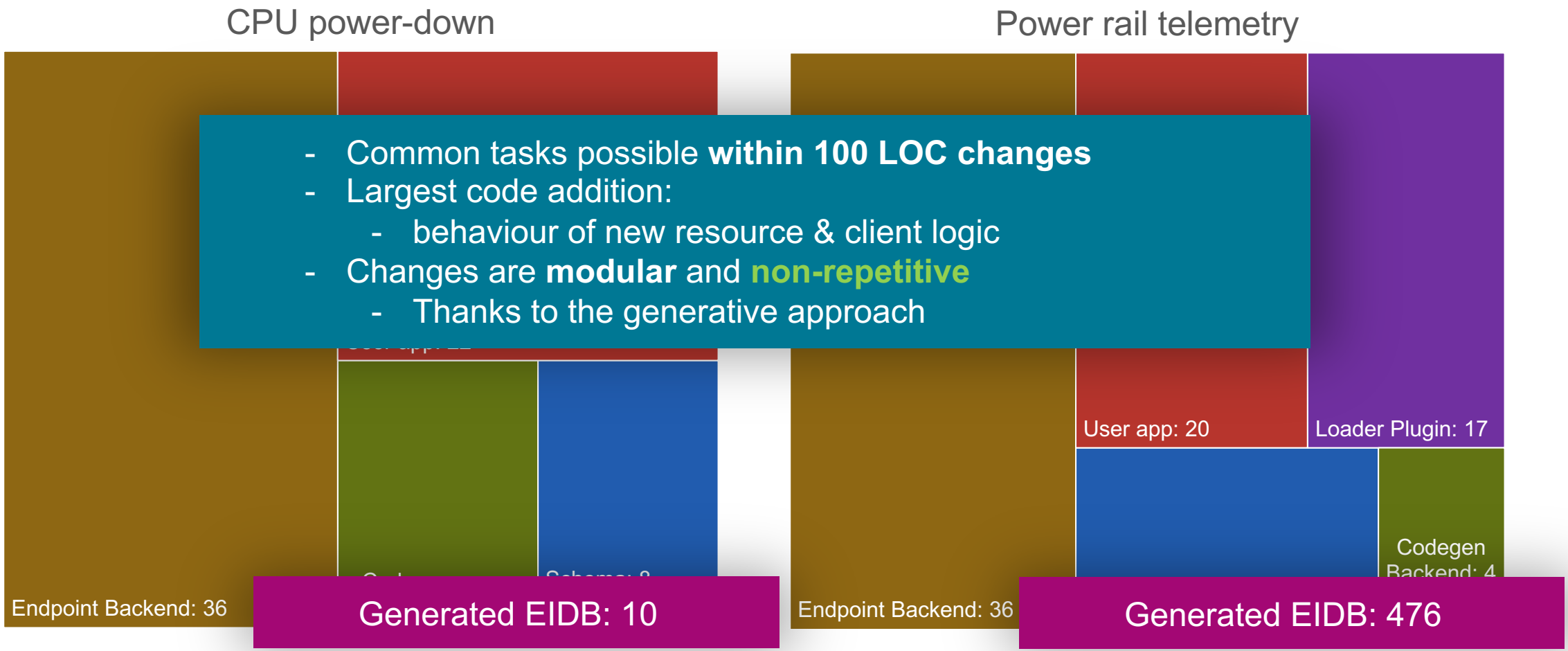
Case studies for ease of extension

- Measure LOC changes for implementing a specific firmware feature



Case studies for ease of extension

- Measure LOC changes for implementing a specific firmware feature



ENZIAN

DINFK

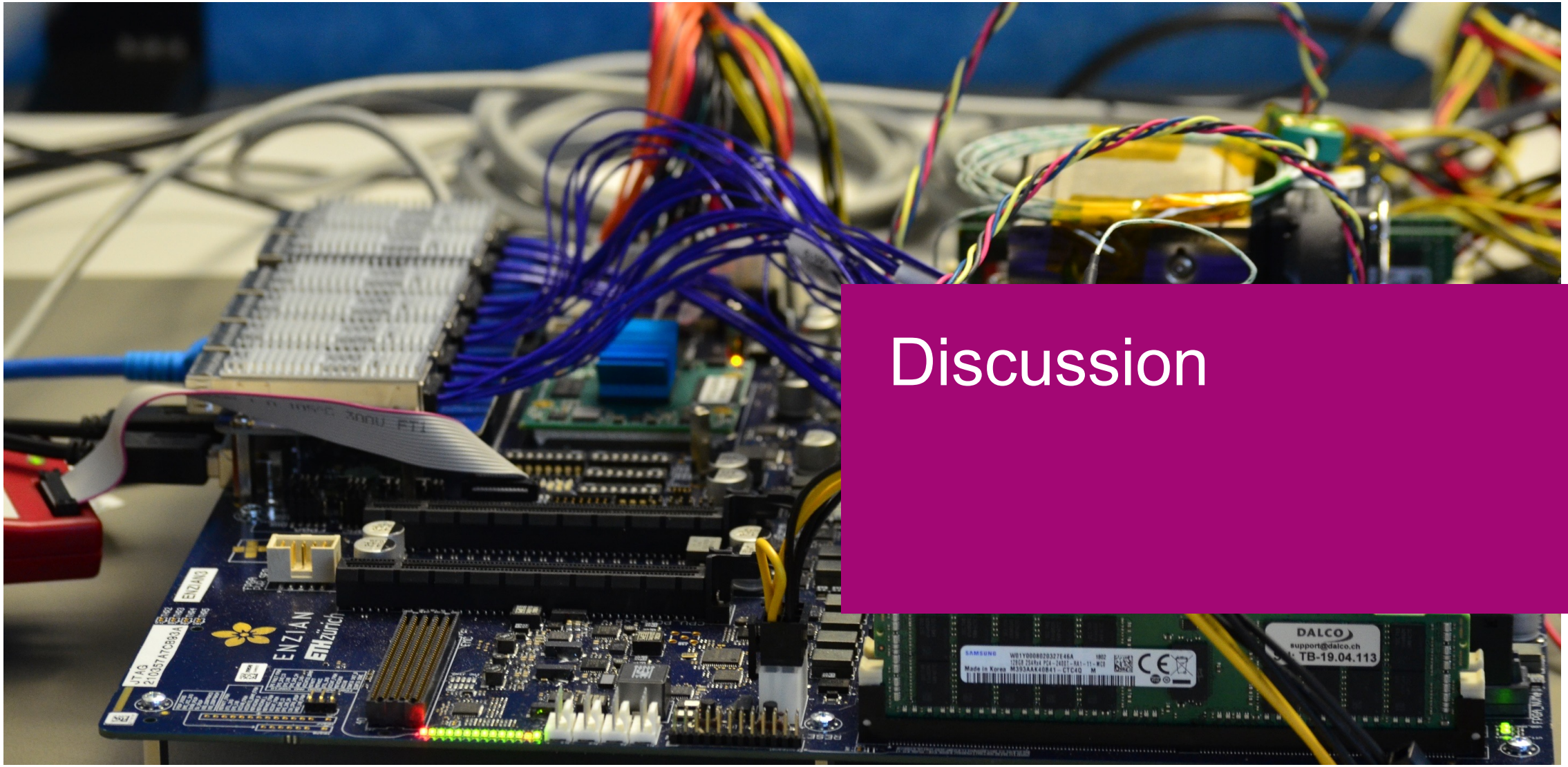
Live Demo

ETH zürich

ENZIAN

Systems@ETH zürich

DINFK

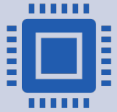


Discussion

Future work

- Better CPU-side API & libraries, language support, etc.
  - Upper EFRI kernel module now exposes argspage to userspace
- Better integration of benchmark framework into base protocol
  - For more predictable time estimation *in-vivo*
  - Important for cross-source profiling
- Asynchronous Event Mechanism implementation
- Security Model implementation
- Explore FPGA-side operation with EnzianShell
- Expand the use-case horizon: implement more services
- …

Conclusion



A *better* firmware protocol for heterogeneous systems

Industrial standards aren't *bad*, they are designed **for a different problem**

EFRI is…

Practical to implement

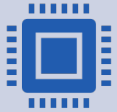
Easy to extend

Acceptably performant

A future where…

EFRI is the *common substrate* of heterogeneous platforms and applications

Conclusion



A *better* firmware protocol for heterogeneous systems

Industrial standards aren't *bad*, they are designed **for a different problem**

EFRI is…

Practical to implement

Easy to extend

Acceptably performant

A future where…

EFRI is the *common substrate* of heterogeneous platforms and applications

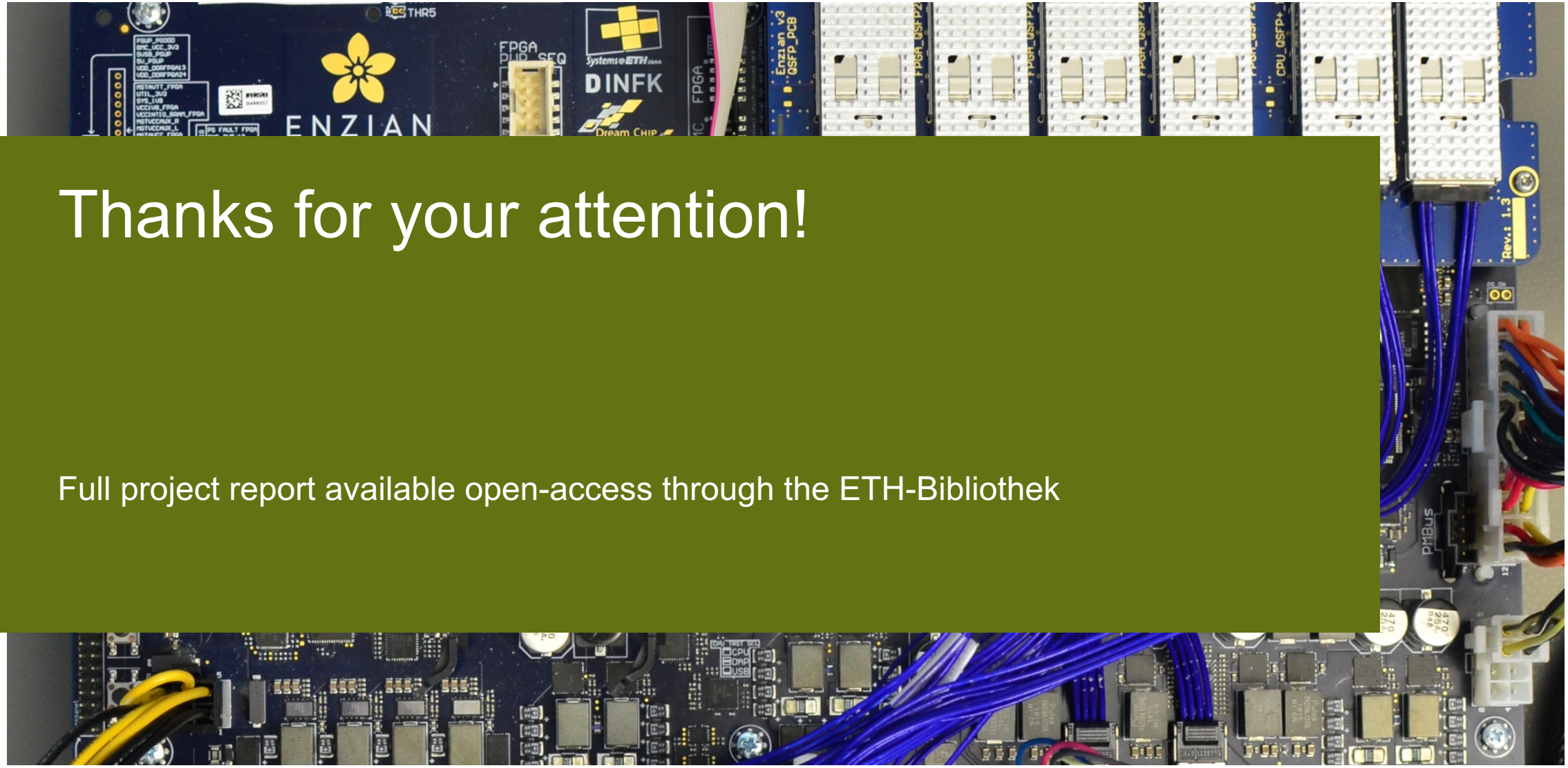
Questions?

ETH zürich

ENZIAN

Systems@ETH Zürich

DINFK



Thanks for your attention!

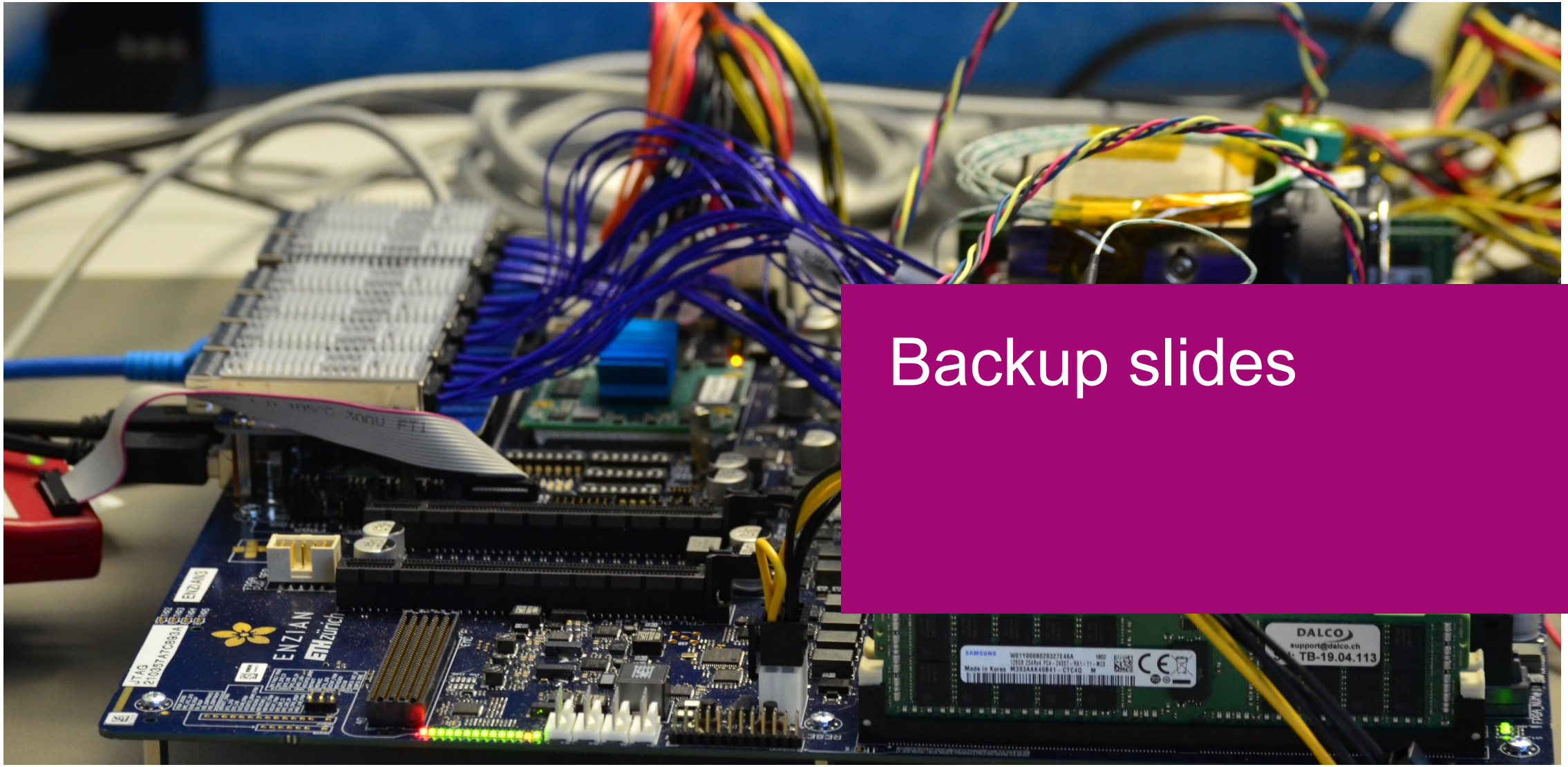
Full project report available open-access through the ETH-Bibliothek

ETH zürich

ENZIAN

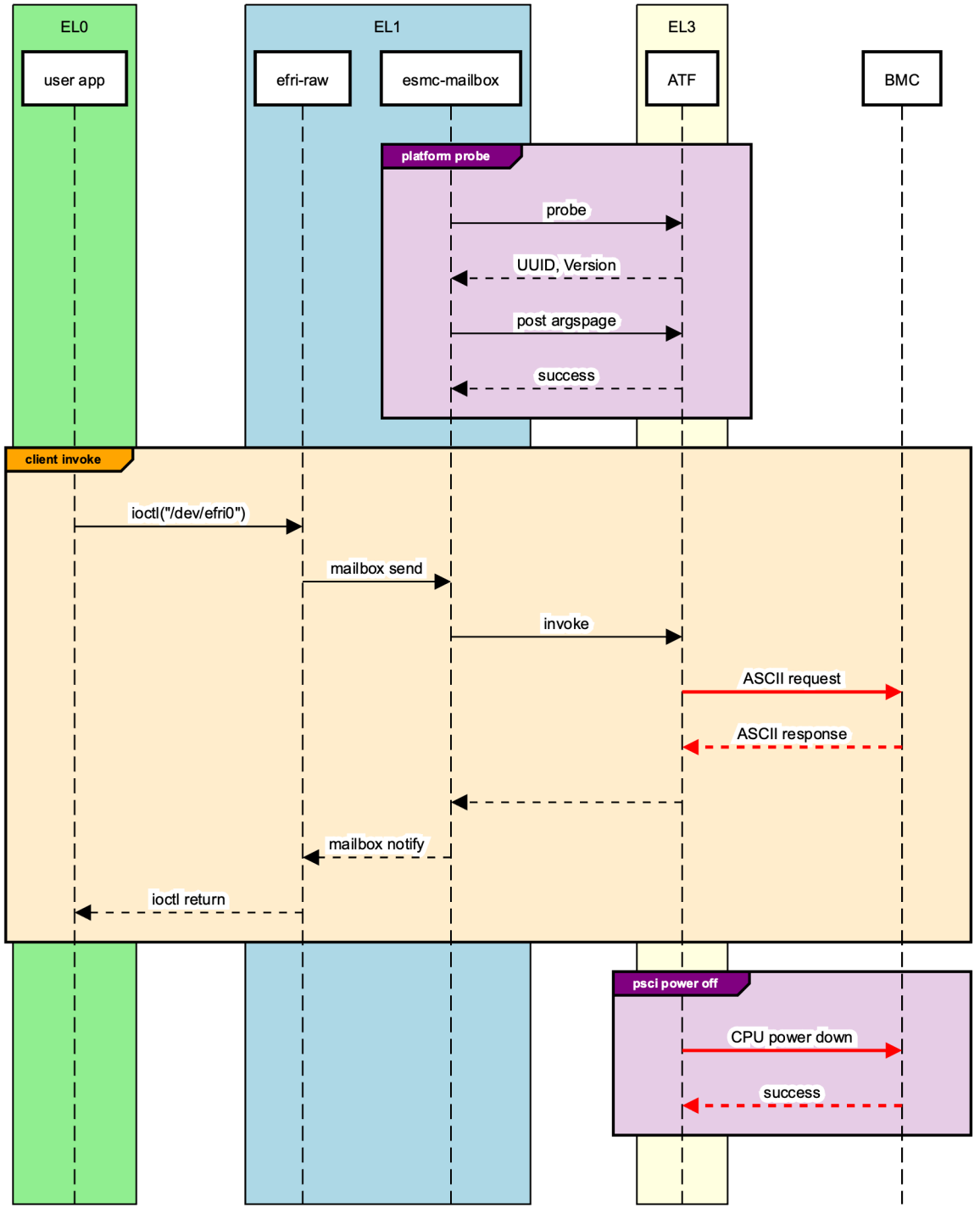
Systems@ETH zürich

DINFK



Backup slides

Global sequence diagram for client invocations



Implementation breakdown for power rail telemetry

Schema definitions

```yaml
actions:
    put: &put
        name: put
        args: [p]
        retval: [err]
    get: &get
        name: get
        args: []
        retval: [err, T]
aliases:
    rw: &rw [*put, *get]
    ro: &ro [*get]
```

```yaml
power_rail: &power_rail
    actions: *ro
    impl: datasource-dbus-power
    class: [platform, rails]
    subsystem: telemetry
rail_voltage: &rail_voltage
    name: voltage
    type: voltage
    <<: *power_rail
rail_current: &rail_current
    name: current
    type: current
    <<: *power_rail
```

Implementation breakdown for power rail telemetry

Codegen backend

```
1  def gen_ep(ep: Endpoint):
2      ...
3      if ep.impl.startswith('datasource'):
4          ds_common = f'"{frn}", {ep.type.name.capitalize()}, {actions}'
5          if 'dummy' in ep.impl:
6              ep_impl = f'DummyDataSource({ds_common}, readonly={"ro" in
               ↪  ep.impl})'
7          elif 'dbus-power' in ep.impl:
8              # ep should have device and monitor properties set
9              assert hasattr(ep, 'device') and hasattr(ep, 'monitor'), \
10                 'datasource-dbus-power requires device and monitor set'
11             ep_impl = f'DBusPowerDataSource({ds_common}, "{ep.device}",
               ↪  "{ep.monitor}")'
12         else:
13             assert False, 'unknown datasource impl ' + ep.impl
14     ...
```

Implementation breakdown for power rail telemetry

Endpoint implementation database

```
1  FRN_EP_MAP["platform:rails:VDD_DDRCPU13:voltage"] =
   ↪  DBusPowerDataSource("platform:rails:VDD_DDRCPU13:voltage", Voltage, [
2      ActionDesc("get", 0, 2),
3      ActionDesc("subscribe", 2, 1)
4  ], "pac_cpu", "VMON9")
5  FRN_EP_MAP["platform:rails:VDD_DDRCPU13:current"] =
   ↪  DBusPowerDataSource("platform:rails:VDD_DDRCPU13:current", Current, [
6      ActionDesc("get", 0, 2),
7      ActionDesc("subscribe", 2, 1)
8  ], "ina226_ddr_cpu_13", "CURRENT")
```

Implementation breakdown for power rail telemetry

Endpoint backend

```
1  E_DBUS = TypedValue(Error, 6) # EFRI_INTERNAL_ERROR
2  class DBusPowerDataSource(DataSource):
3      @classmethod
4      def _get_service(cls):
5          return dbus.SystemBus().get_object('systems.enzian.Power',
           ↪  '/systems/enzian/Power')
6      def get(self):
7          try:
8              raw_val = self._get_service().read_device_monitor(self.device,
               ↪  self.monitor)
9          except dbus.DBusException:
10             return E_DBUS, None
11         return E_OK, TypedValue(self.ty, raw_val)
```

Implementation breakdown for power rail telemetry

Userspace application

```c
#include "libefri.h"
int main(int argc, char *argv[]) {
    int fd = open("/dev/efri_raw0", O_RDWR);

    // fill argspage
    struct efri_raw_invoke_msg msg;
    efri_frame_t *frame = &msg.frame.frame;
    char *cur = msg.frame.aux;
    frame->frn = push_string(msg.buf, &cur,
↪   "platform:rails:BMC_VCC_3V3:voltage");
    frame->action = push_string(msg.buf, &cur, "get");
    frame->role = ROLE_REQUEST;
    frame->num_args = 0;
    msg.dest = EFRI_DEST_BMC;
    msg.size = cur - msg.buf;

    // invoke and print result
    ioctl(fd, EFRI_RAW_INVOKE, &msg);
    describe_frame(frame);
}
```