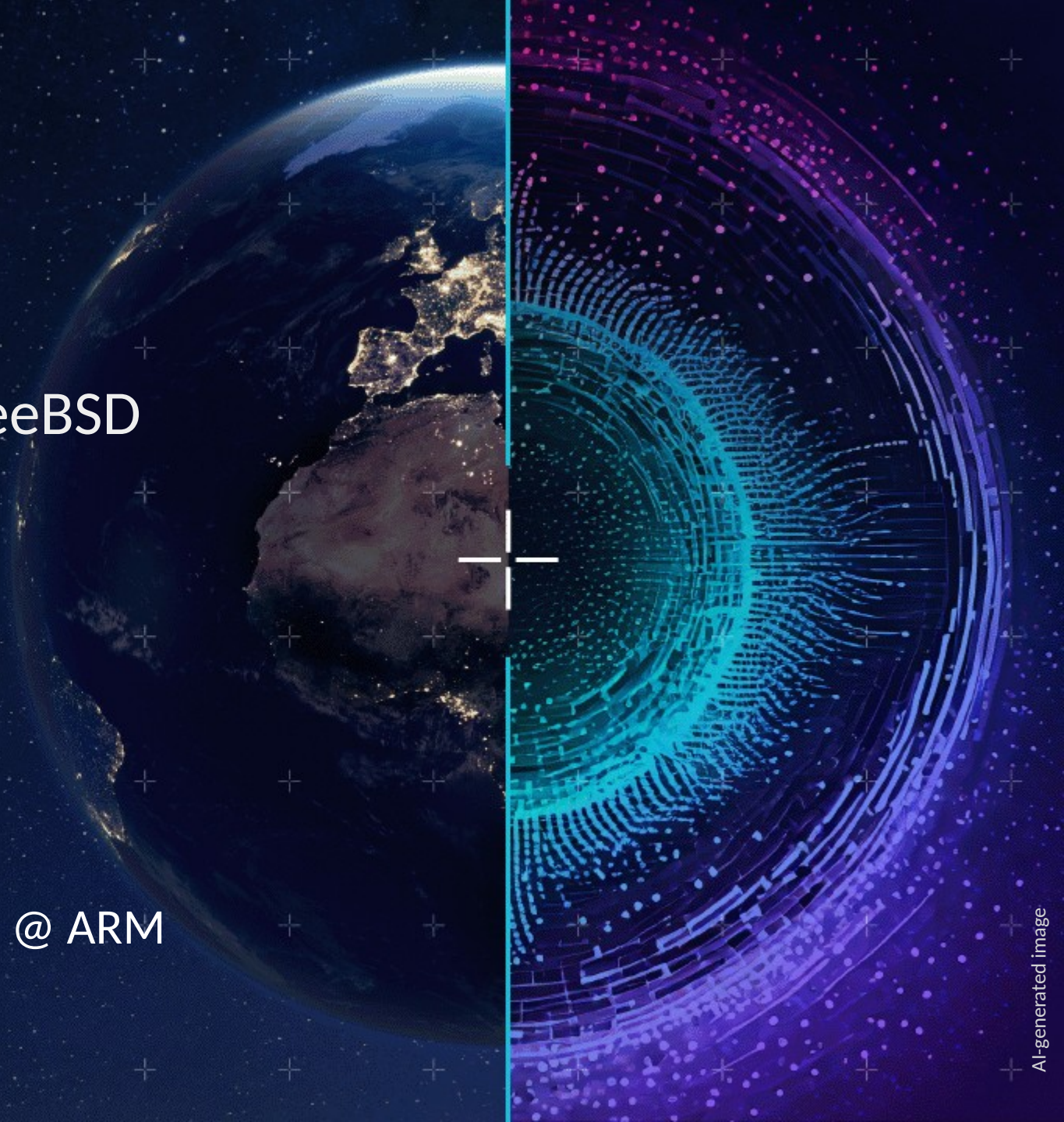
arm

Growing the SCMI support on FreeBSD



Cristian Marussi – Staff Software Engineer @ ARM

cristian.marussi@arm.com

AI-generated image

Agenda

- whoami (… with a “disclaimer”)
- What’s the problem(s) … why SCMI ?
- SCMI Overview
- Deployments Scenarios
- SCMI Messaging
- SCMI FastChannels
- Coexistence on ACPI systems
- Linux SCMI Stack (as a reference example)
- Current SCMI Support in FreeBSD
- Work in progress
- … Next

What's the problem ?

- Modern SoC are complex beasts …
  - … composed by a number of **different logic elements** providing a number of functionalities
  - … but, usually, **not all together** at the same time and **not at the same level** of performance
- … a well-known design **strategy to maximize efficiency** is to have it:
  - … **partitioned** in a number of distinct power/voltage **domain islands**
  - … so that such islands can be **selectively configured** based on actual runtime needs
- … such islands will definitely:
  - … have different on-chip users requiring, dynamically, **different configurations**
  - … with such users frequently using **different SW stacks** with different security needs
- … such users, potentially conflicting, configuration **requests will be served**
  - … by some sort of **central entity** who is in charge of policing

… BUT …

I - Why SCMI ?

- We don’t have a **common language** to express such requests so ….

  => a number of protocols have come into existence in the recent past:

  TI/SCI, QCOM/RPM, Nvidia/BPMP, ARM/SCPI (!)

SCMI – System Control and Management Interface [1]

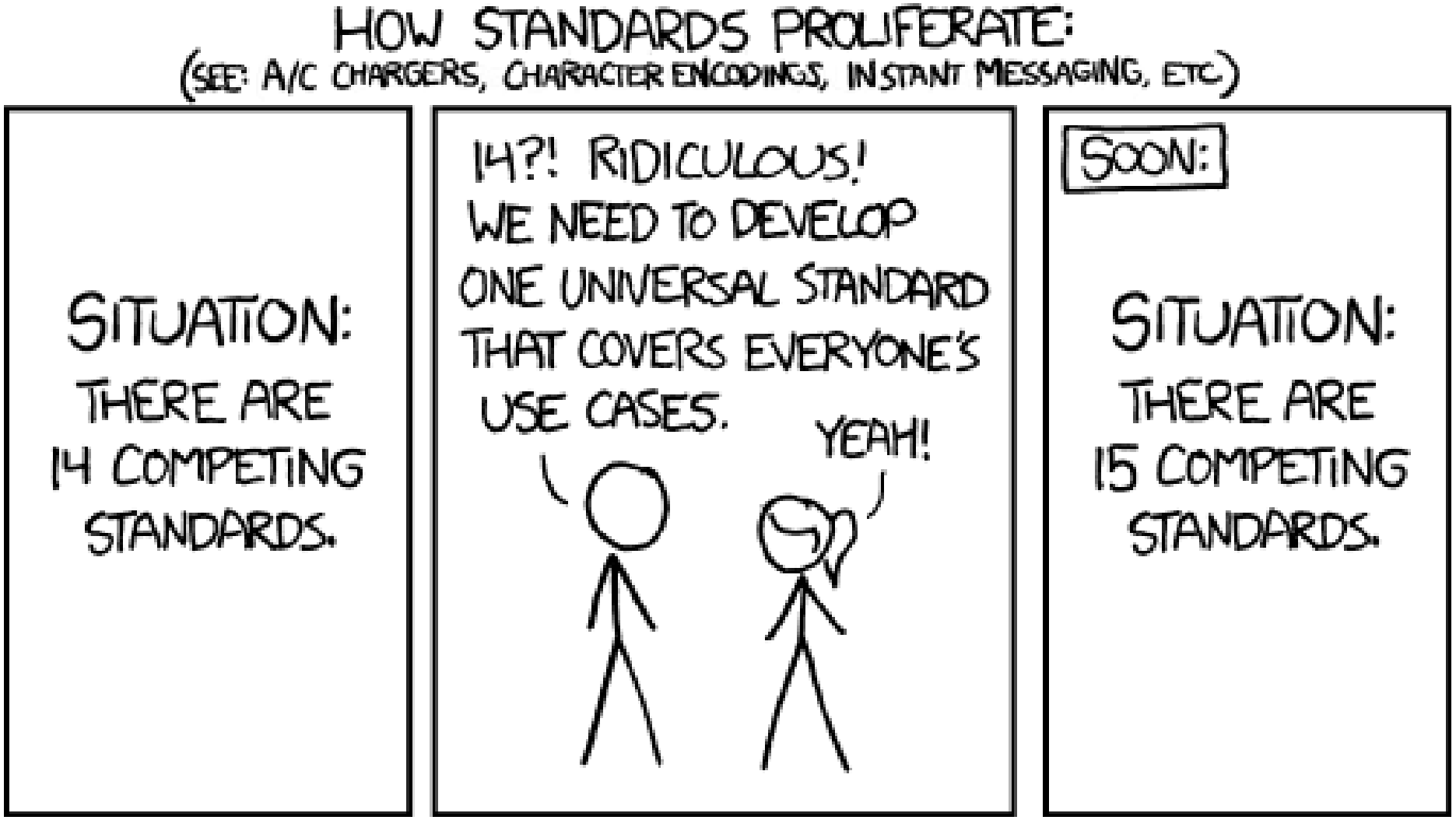
→ aims to unify this with a new **standard common protocol** abstraction …

“… System Control and Management Interface (SCMI), which is a set of operating system-independent software interfaces that are used in system management.”

… in these regards, of course, this could come to mind …

[1]: https://developer.arm.com/documentation/den0056/latest/

II - Why SCMI ? - https://xkcd.com/927/



arm

III - Why SCMI ?

… BUT …

since SCMI is more **flexible**, easily **extensible** and transport independent and …

… thanks to the efforts of Arm, Linaro and our partners …

→ an **increasing number of real-world systems have adopted SCMI** recently…

- … in a number of different **segments**: Mobile / Automotive / Embedded / PCs
- … even amongst the vendors that had previously developed their own protocol

… so this gives us some sort of **validation** …

- … and I am here pestering you about this SCMI thing :P

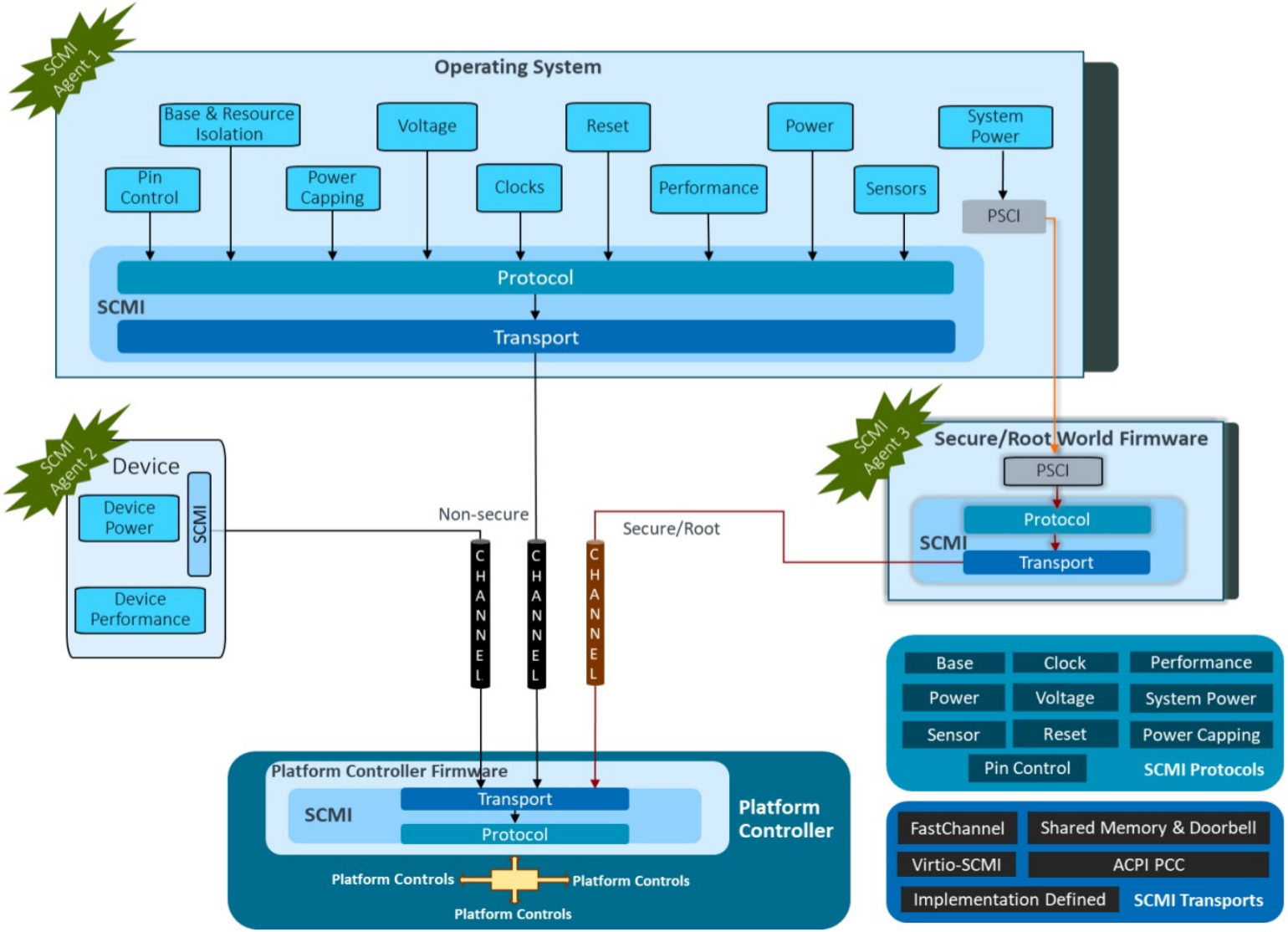
SCMI Overview - I

- Based on a **client-server model**
  - one SCMI server, called **platform**, which can live in a number of places
  - multiple SCMI clients, called **agents**, issuing possibly conflicting requests to the server
  - multiple possible transports are possible (in Linux MBOX/SMC/VIRTIO/FFA/OPTEE) [out-of-spec]
  - each agent has one or more dedicated communication channel
    - agents are identified by the server from the channel they speak from
- **OS agnostic** specification provides protocol interfaces dedicated to system management
- **Fully discoverable** around features, services and available resources (except transport characteristics)
  - a mandatory Base protocol to discover the list of implemented protocols (and other things…)
- **Extensible**
  - per-domain dedicated sub-protocols (voltage, power, performance) … external contributions possible
  - optional vendor protocols

SCMI Overview - II

- **Abstraction** and unification of accesses to managed resources
  - no need to have hw specific drivers kernel-side
    - …as long as you have them platform side … and you speak SCMI (not necessarily for everything)
- **Policy enforcing** on a per-client basis, SCMI server is the ultimate arbiter:
  - can deny, silently or not, any agents' request based on server configuration
- **Enhanced security** by delegation of policing and actions to a **smaller TCB living in a secure world**
  - CLKscrew https://www.youtube.com/watch?v=0tM2v2SZDxY … server can deny unsafe requests even from rooted OS
- **Virtualization friendly** design
  - server can expose a different **views** of the system to each agent (VMs)
    - same shared resource, different ID (ex. Clock) **OR** different resource, same ID (ex. sensor)
  - **harmonization** of potentially conflicting requests around shared resources
- transport independent…not part of the specification really….

SCMI Overview - III



SCMI Server Deployment Scenarios

Having a number of transports available means the SCMI server can be **deployed in a number of different places**:

- in a dedicated MCU, most probably in Secure world
- in a TrustedApplication which run within the context of a TEE OS (like OPTEE)
- in a SecurePartition running at S-EL2, S-EL1 or S-EL0 depending on arch support
- in a dedicated Virtual Machine
- embedded in EL3 TF-A (discouraged…)

A **reference SCMI Server** platform is implemented in **ARM SCP Firmware** [1], and a compliance test suite is available too [2] … but vendors can write **their own** servers …

[1] https://gitlab.arm.com/firmware/SCP-firmware
[2] https://gitlab.arm.com/tests/scmi-tests

SCMI Messaging - I

Agents/Platform communications use 2 kinds of channels:

- **A2P**: agent to platform channels carrying the **agent initiated** transactions, i.e. command issued by the agent and the immediate responses from the platform
- **P2A** (optional): platform to agent channels carrying the **platform initiated** transactions, i.e. delayed responses and notifications

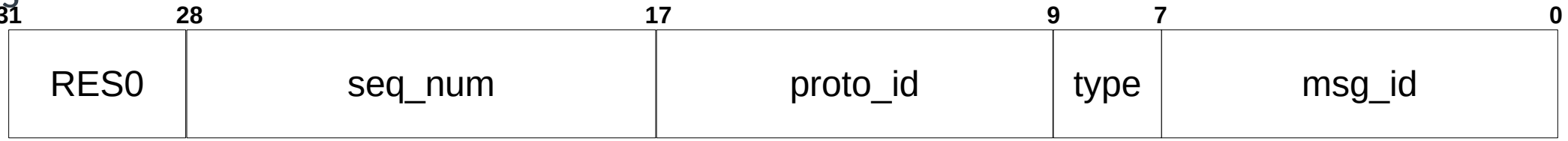
… and 2 kinds of commands:

- **Synchronous Commands** - agent issues a command on A2P and the response is immediately delivered on that same channel, which is ***kept busy*** until the request has been served and the reply received.

- **Asynchronous Commands** - agent issues a command on A2P and an immediate status-only response is delivered on that same channel: such immediate reply does NOT carry any effective payload but ONLY the status, so that the channel is ***freed immediately***; the full-response will be delivered **later** (once computed) using a **delayed response** sent on **P2A**

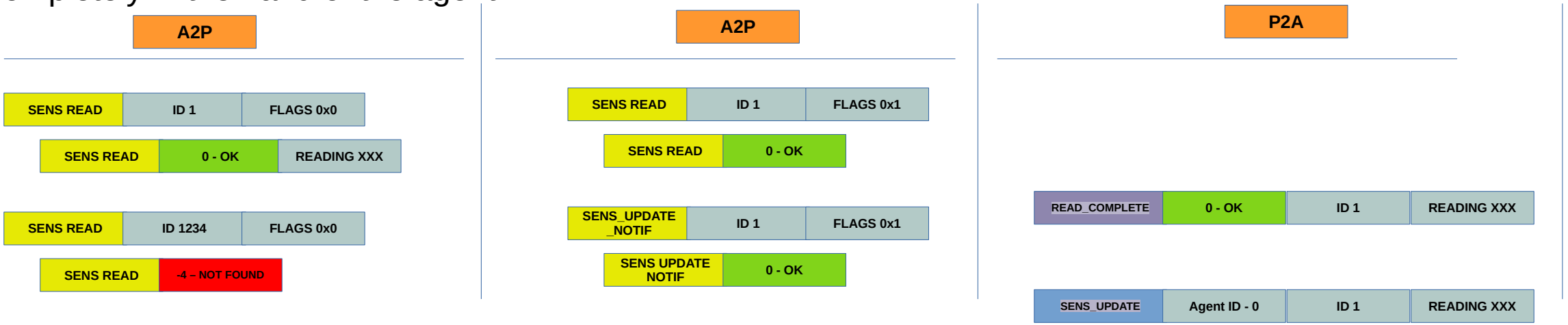
SCMI Messaging - II

Each message is composed by a 32-bit **header** and an optional payload of <N> 32-bit words



Types :: 0 – command/response 2 – delayed response 3 – notification

A response message is built by returning **exactly the same header**, unchanged, plus any optional payload. (except for delayed response where the type is changed). Policy around the usage of the ***seq_num*** field is completely in the hand of the agent.



SCMI FastChannels

Beside the standard messaging channels, based on a command-reply pattern, FastChannels are provided as an **alternative** messaging mechanism but only for:

- some specific commands in a few protocols
- a specific resource

“A FastChannel is a lightweight unidirectional channel that is dedicated to a single SCMI message type for controlling a specific platform resource.”

In a nutshell the **server can advertise** (via regular messaging) some well defined memory **areas** where the agent can **read/write directly the payload** for a well defined command related to a well defined resource, avoiding the command-reply overhead.

SCMI on ACPI Systems

Most of the SCMI Agent support currently needed in Linux/FreeBSD is targeted at DeviceTree based systems…which is where most of the work is needed

ACPI-based implementations **can leverage SCMI protocols** to provide platform services using standard ACPI methods (SCMI specification is kept ACPI compatible by ATG)

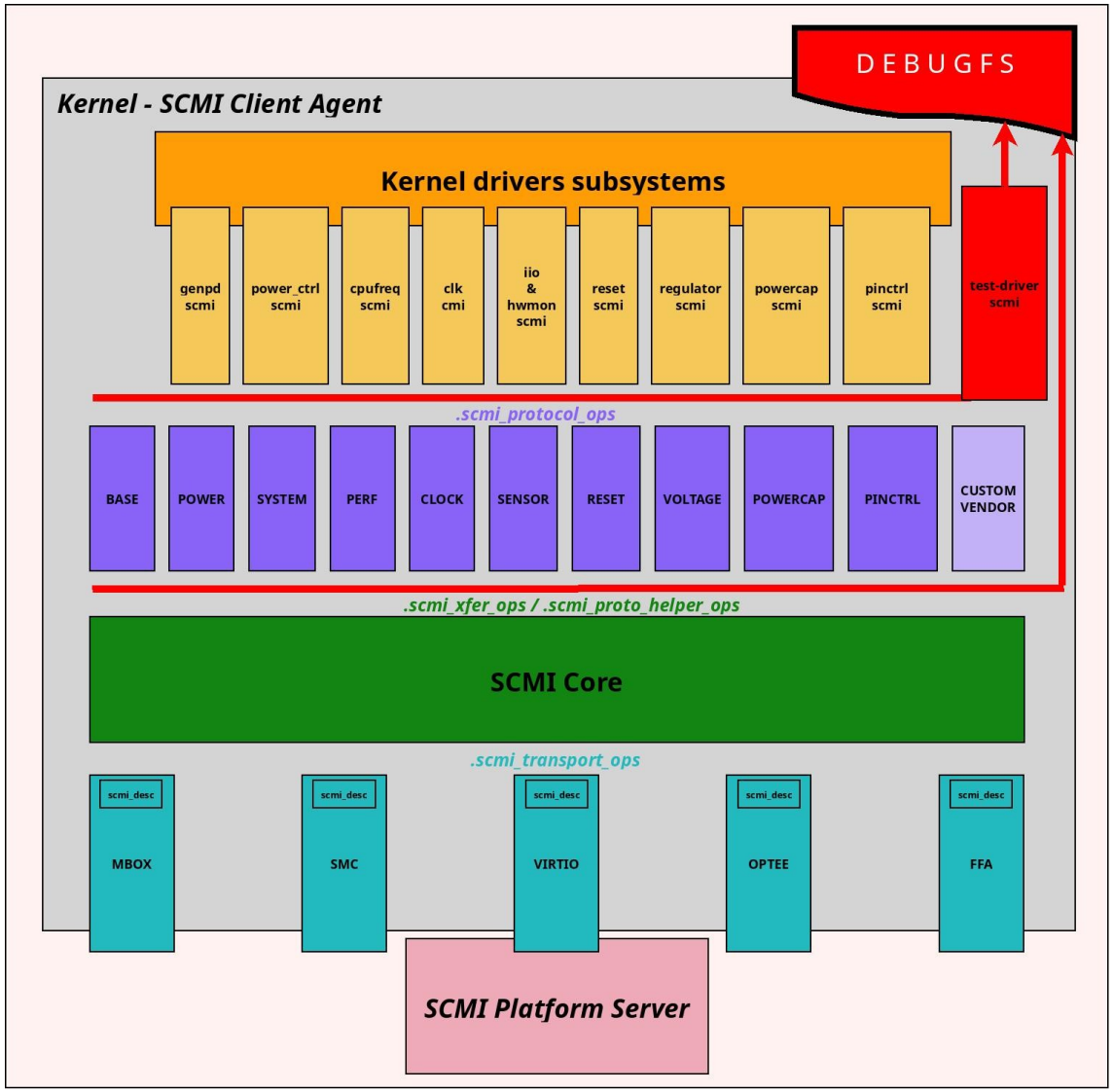
As an example, a device may be power managed by ACPI-aware OS using the **standard ACPI control** methods.

These ACPI methods can **send SCMI Power Management Protocol requests** to the platform to transition the power state of the device.

SCMI **MBOX/SHMEM transport** channels can be represented as an ACPI Platform Communications Channel **(PCC) of Type 3.**

The server on the other end does NOT really know if the request originated from a DT or an ACPI system…as long as you have some **glue** code in the ACPI stack (TBD)

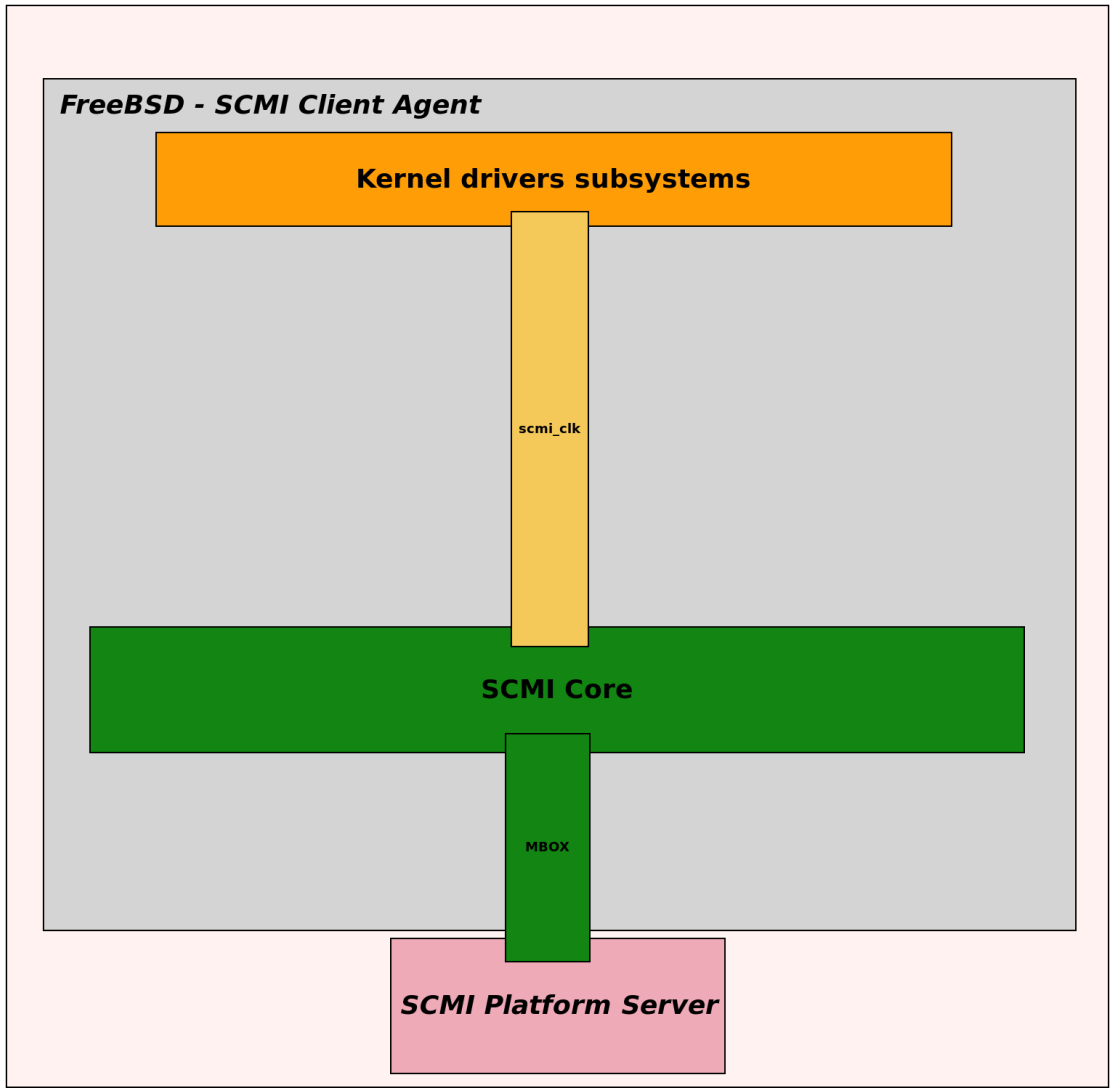
Linux SCMI Stack



- **Layered** design
  - ***transport:***
    - Abstract medium specific details
    - MBOX/SMC/OPTEE use a well defined SharedMemory area to carry the message
  - ***core*:**
    - Message tracking (*seq_num)*
    - Handles replies, timeout, errors, late replies
  - ***protocols*:**
    - Knows how to build the messages for a specific task
  - SCMI driver ***users:***
    - Plug into various kernel subsystems (clocks)
    - Call into protocol ops (clock_enable)

- 2 test and development facilities in DEBUGFS
  - **SCMI Raw** - inject/snoop messages to:
    - test the Server with the compliance suite
    - test the basic core messaging functionalities
  - **scmi-test-driver** – invoke scmi_ops to:
    - test the SCMI protocol APIs (not upstream)

FreeBSD SCMI Stack - I



Initial FreeBSD SCMI Support

- Initial commit in 2022

commit 54b96380f5774c1754a0fcf25212fa8e01db74f6
Author: Ruslan Bukin <br@FreeBSD.org>
Date: Mon Dec 19 20:16:18 2022 +0000

Add support for ARM System Control and Management Interface (SCMI) v3.1.

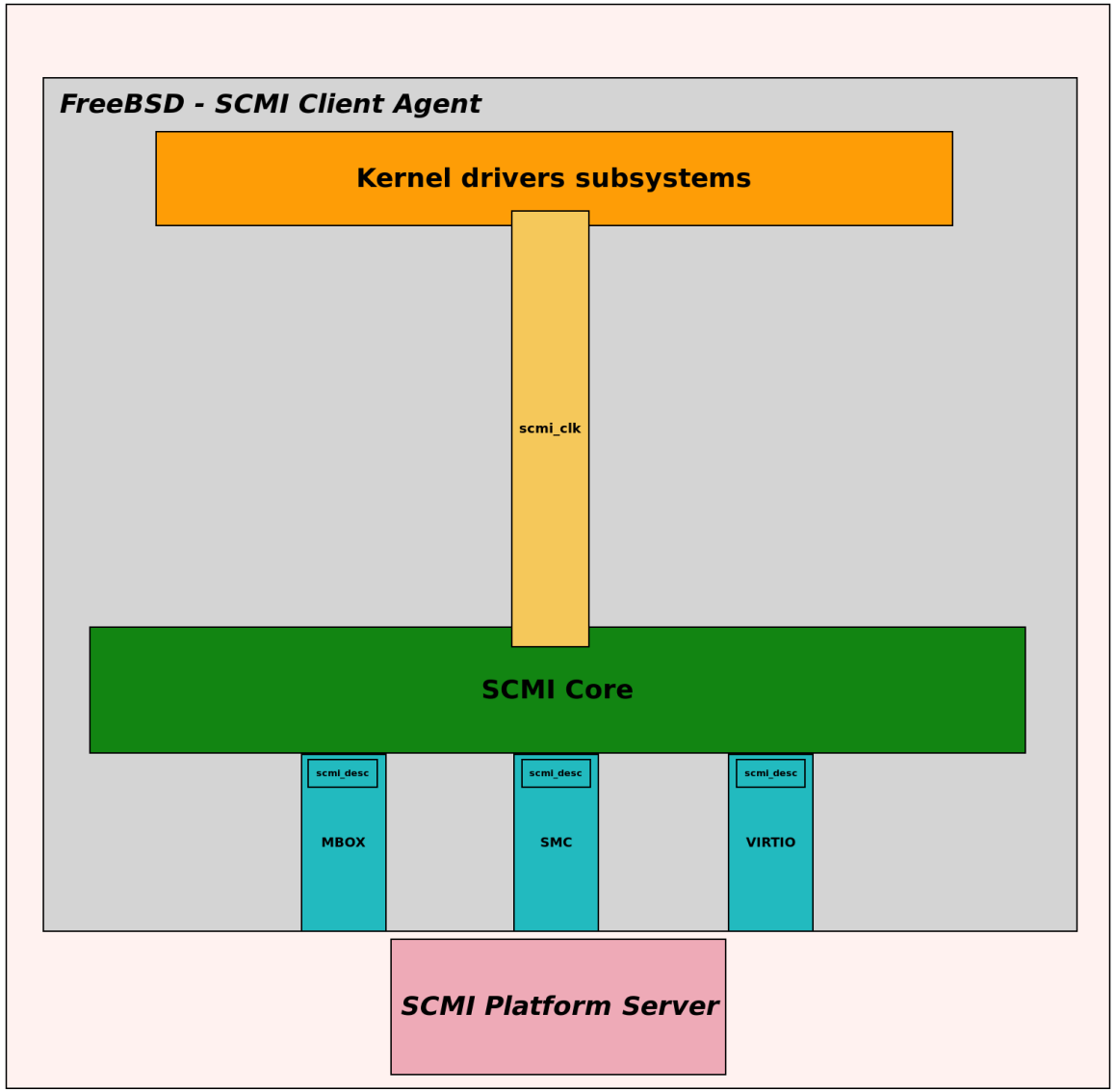
- added minimal support for
  - 1 transport (MBOX)
  - “Basic” core facilities (only sync-commands)
  - SCMI clocks

- … added support was **minimal for a very good reason**:
  - **ONLY 1 SCMI platform** (server) available on FreeBSD
    → Morello implementing only Clock protocol
- ...**today** → **not so many more** FreeBSD/SCMI platforms

BUT...

- the Server **can now be deployed in a number of places**
  → including in some VM or in Userspace

FreeBSD SCMI Stack - II



Current FreeBSD SCMI Support

...so the **first addition from Arm** around mid-2024 was:

- Splitting out the transport layer and adding:
  - **VirtIO transport** [1]
  - SMC transport
- Restructure a bit of the SCMI Core in preparation of more complex messaging support (still only sync-cmd…)

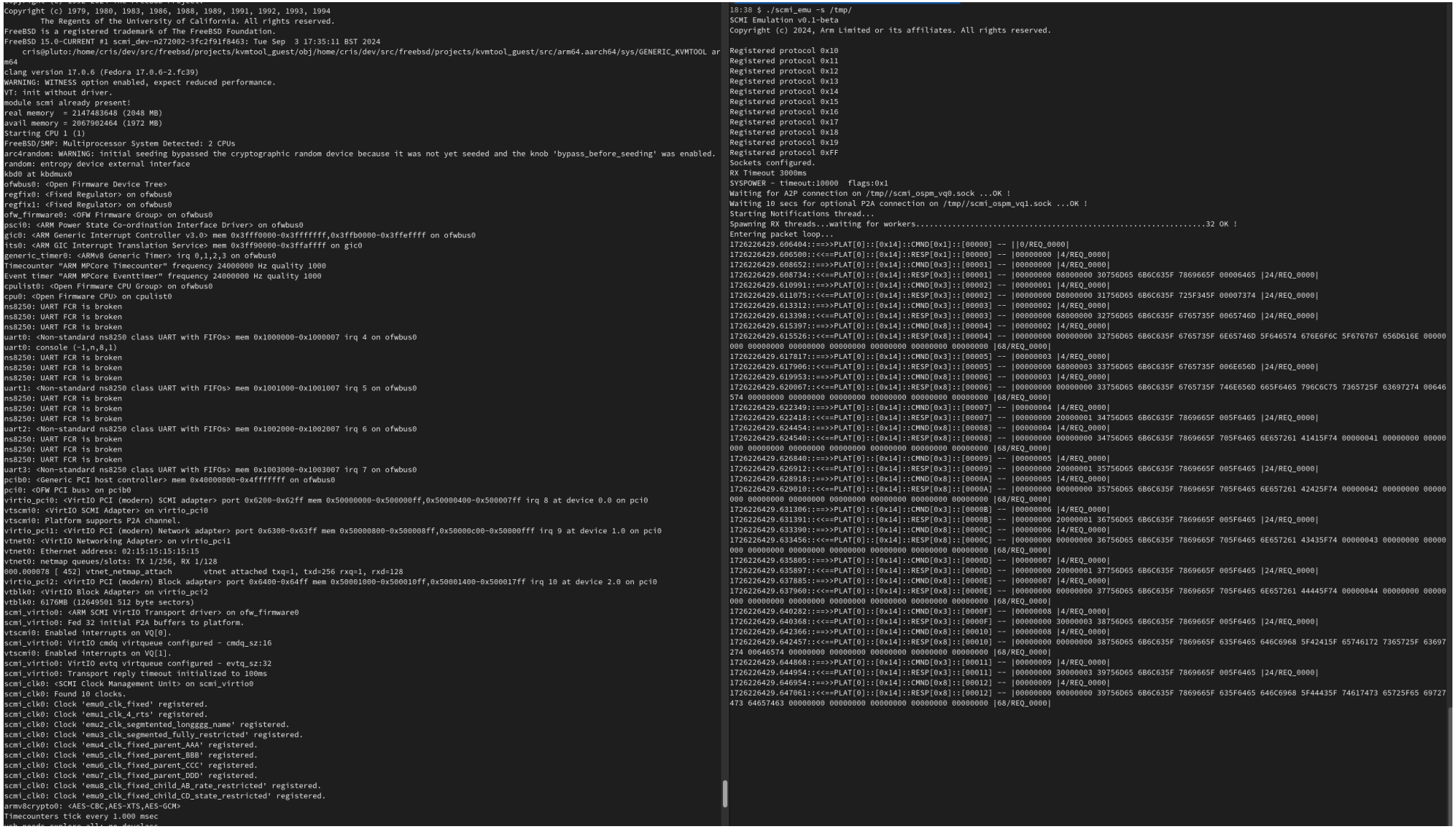
Since enabling the use of a **virtualized SCMI Server**:

→ allows for more complex SCMI development on FreeBSD SCMI Agent capabilities **without real HW** available

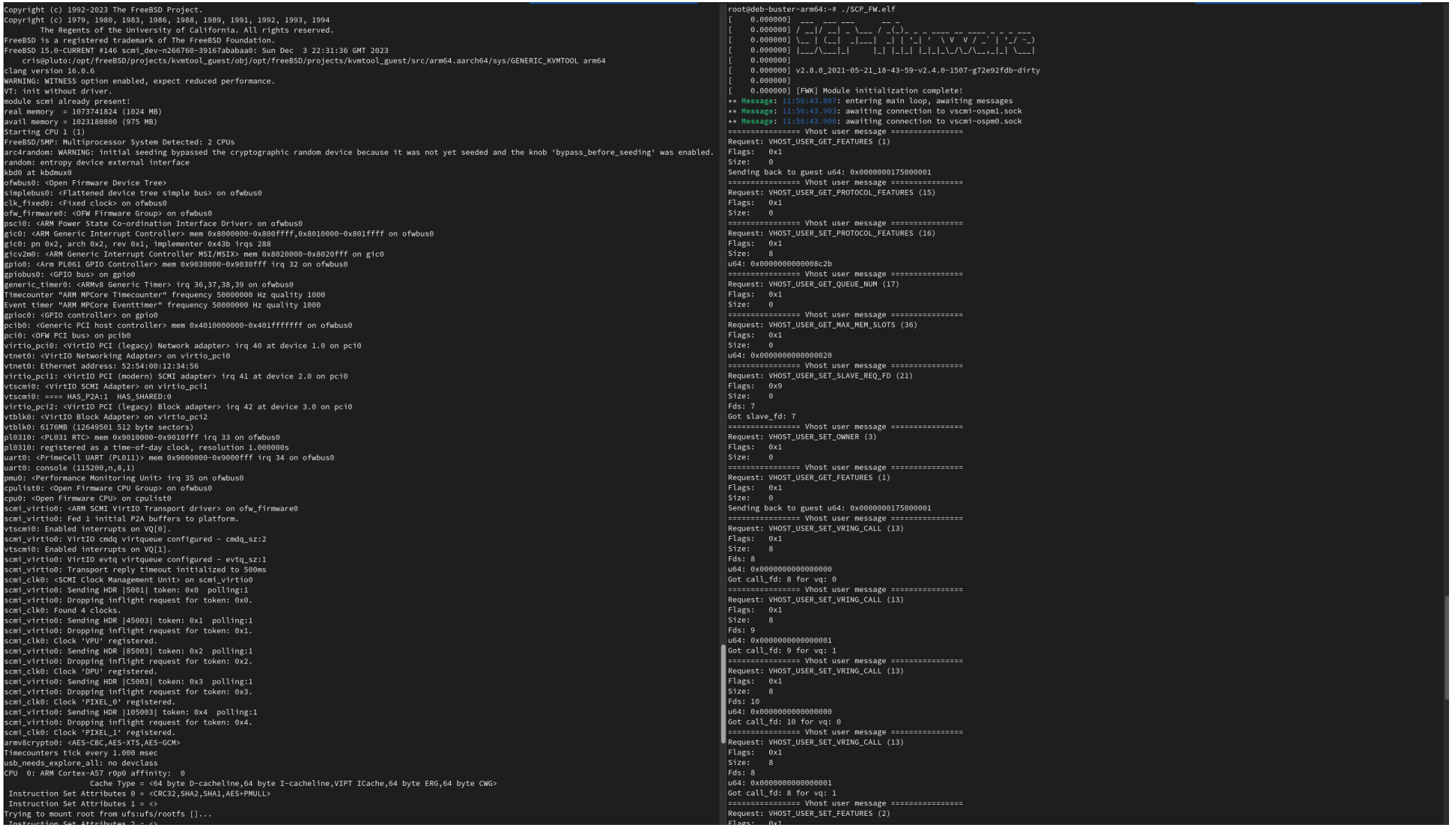
→ allows anyway the FreeBSD SCMI Agent to be used in an existing **virtualized deployment**

[1] https://docs.oasis-open.org/virtio/virtio/v1.2/csd01/virtio-v1.2-csd01.html#x1-60400017

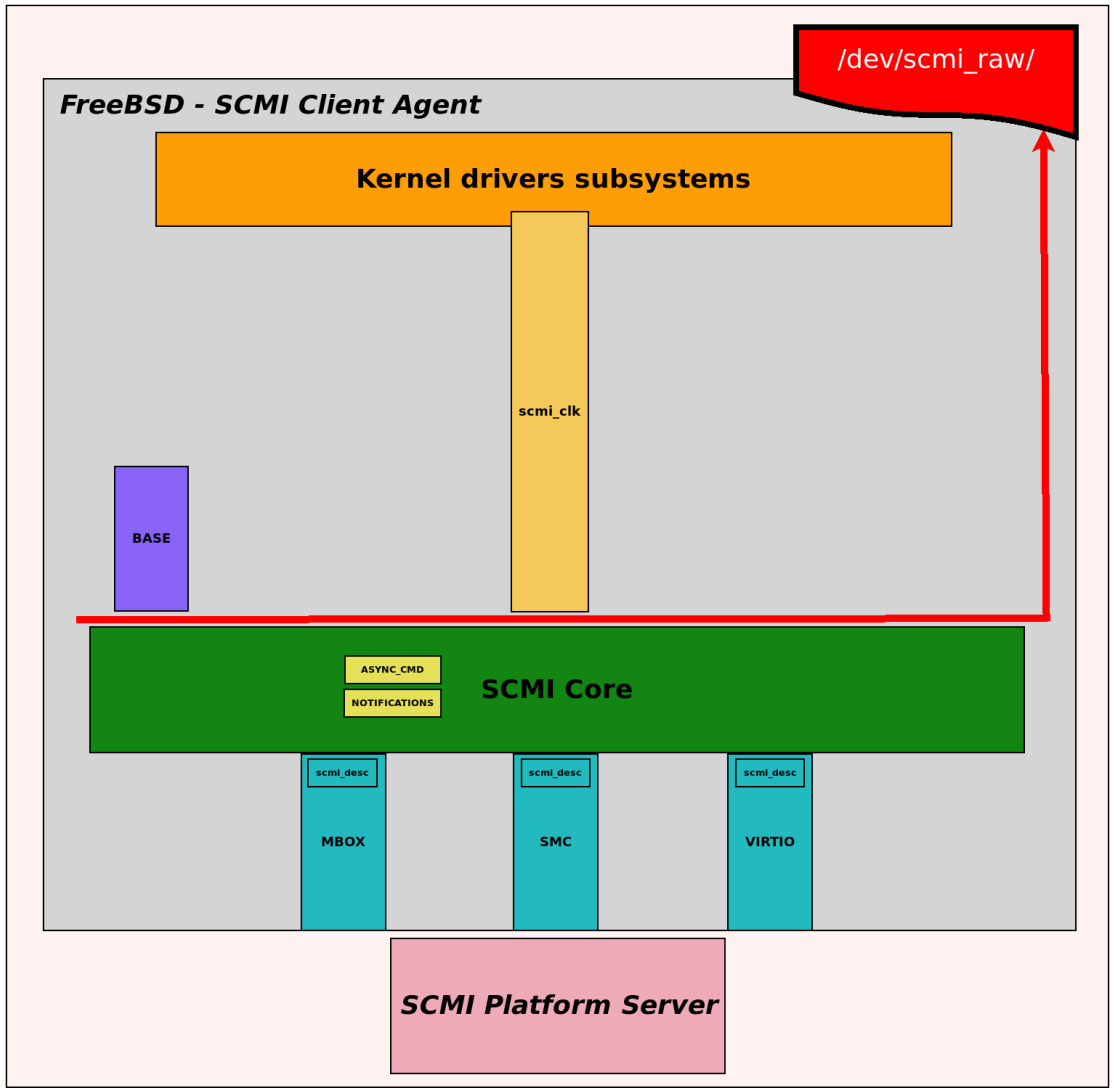
FreeBSD - Guest SCMI Agent with KVMtool/SCMI_EMU



FreeBSD: Guest SCMI Agent with QEMU/SCP



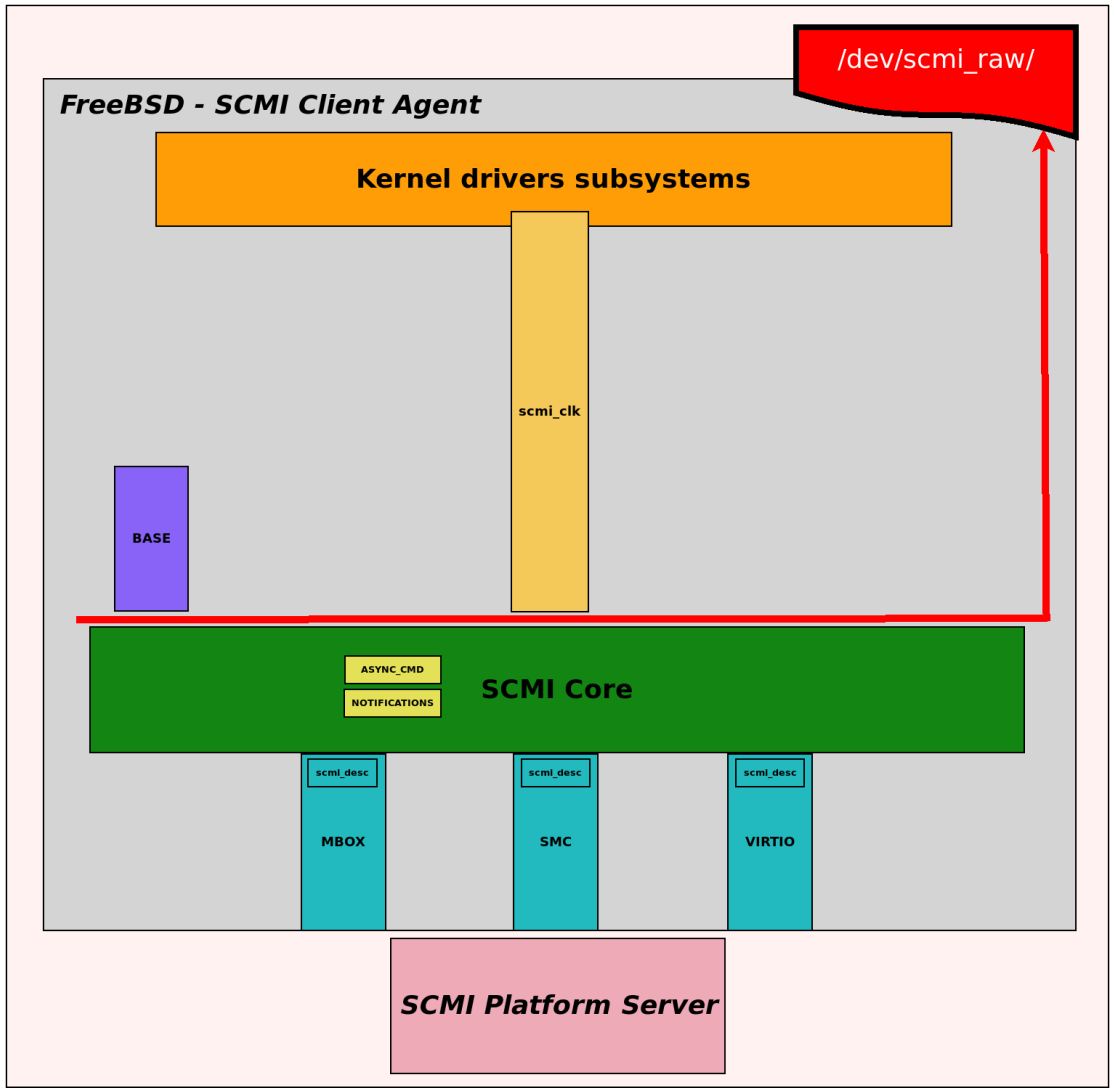
FreeBSD SCMI Stack - III



FreeBSD SCMI Support – WIP or planned

- Adding **SCMI RAW** for message injection to enable:
  - **testing** of the SCMI Core stack
  - **compliance** testing of a real Platform server
- Adding SCMI Core support for:
  - **Asynchronous commands**
  - **Notifications**
- Adding **Base** protocol
- Some **SDT(?) traces** to dump ongoing SCMI transactions

FreeBSD SCMI Stack - IV



FreeBSD SCMI Support - NEXT

The current WIP will enable a more complete development and test environment for SCMI on FreeBSD...so…

What's NEXT ?

→ Per-protocol transport channels (where possible)
→ Protocol layer abstraction (maybe)
→ Vendor Protocols support
→ SCMI Test Driver
→ More transports (if really required by new HW)

Enabling more SCMI Protocols and related drivers ?
Depends on the availability of related Kernel frameworks on FreeBSD:

- Power
- SysPower
- Performance: cpufreq(4) / scmi_perf ?
- Sensors ?
- Reset ?
- Voltage ?
- Powercap ?
- Pincontrol ?

arm

Questions ?

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు