



SCMI

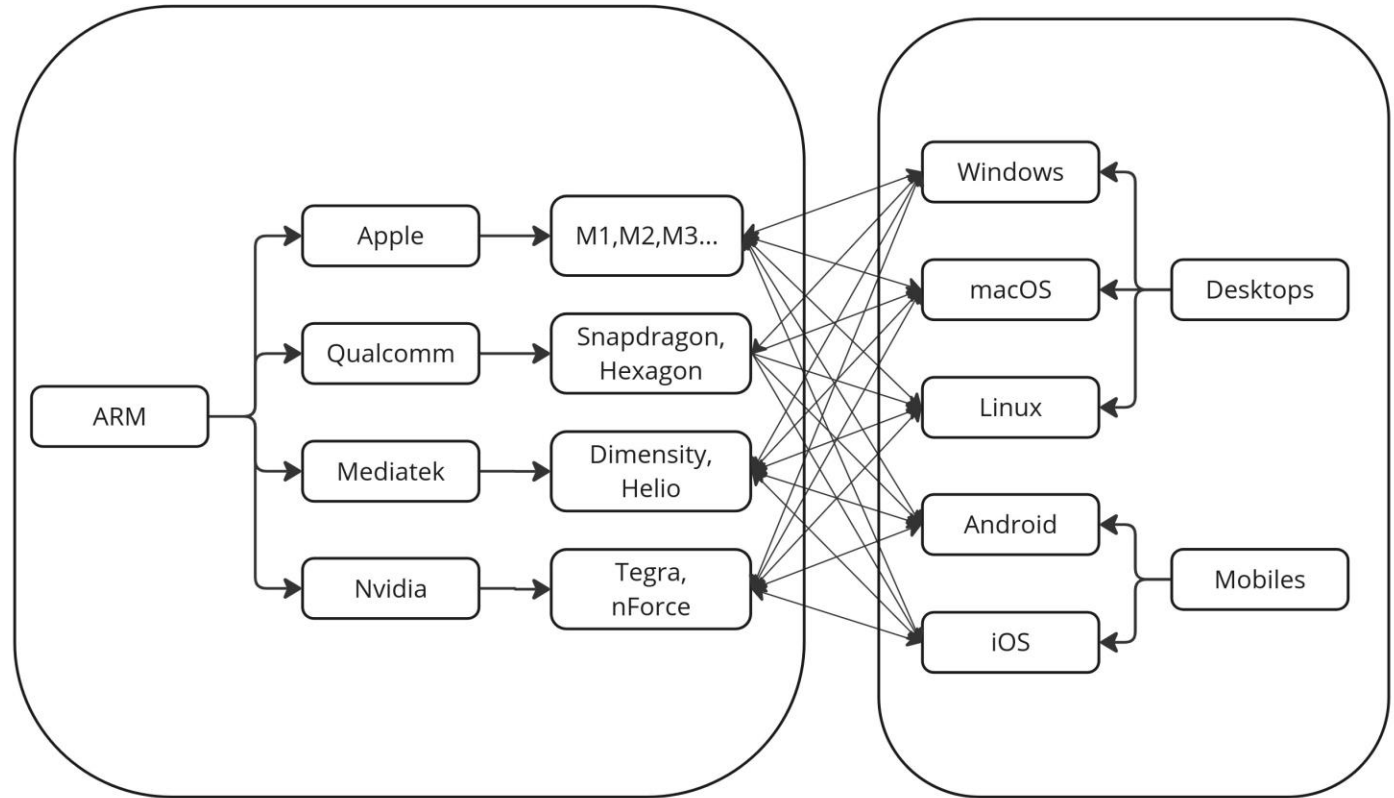
Presentation 1



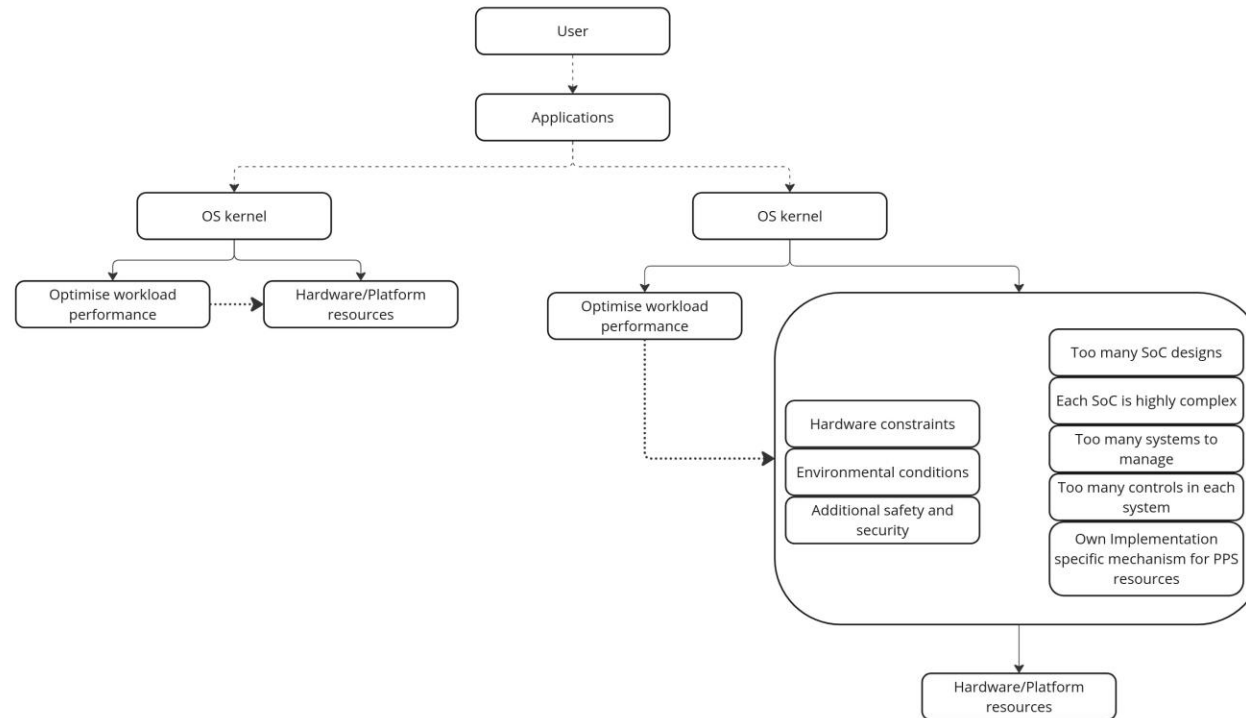
Terminology

- Agent : Entity which sends messages to SCP.
 - Requester : Agent who initiates transactions
 - Completer : Agent who responds and completes the transactions.
 - A2P -> Agent : Requester, Platform : Completer
 - P2A -> Platform : Requester, Agent : Completer
- Platform : Set of system components (SoC hardware) which interpret the messages send by agent and provide necessary functionality.
- Message : An individual communication from A2P or from P2A.
 - Command : Message from A2P
 - Notification : Message from P2A
- Channel : Transport link over which agent communicates to platform.
- Event : Desired job or status that is taking place in platform.

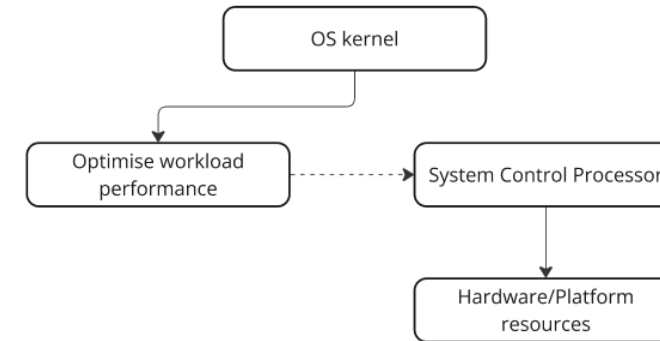
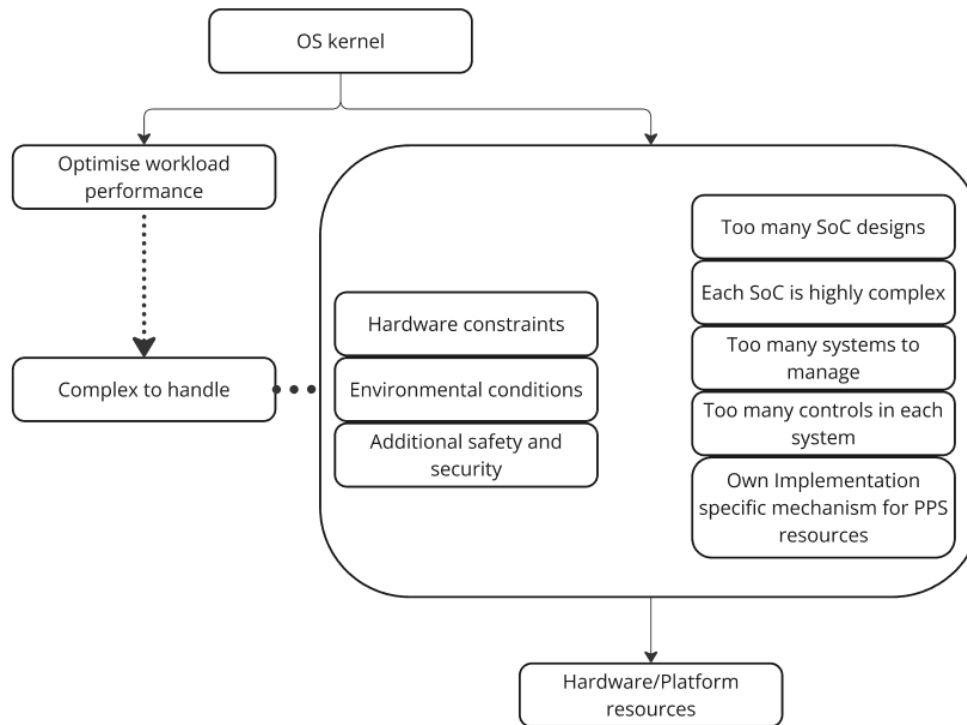
- Variety of ARM based chips are evolving in market.
- SoC's are becoming unique & complex in terms of design and PP management.
- Creates huge complexity for OS developer to manage everything in single, generic kernel.
- GPU's and accelerators further fuels complexity in both areas.
- But, bilateral support is expected from both Silicon vendors and OS vendors.



Traditional vs Modern duties of OS kernel



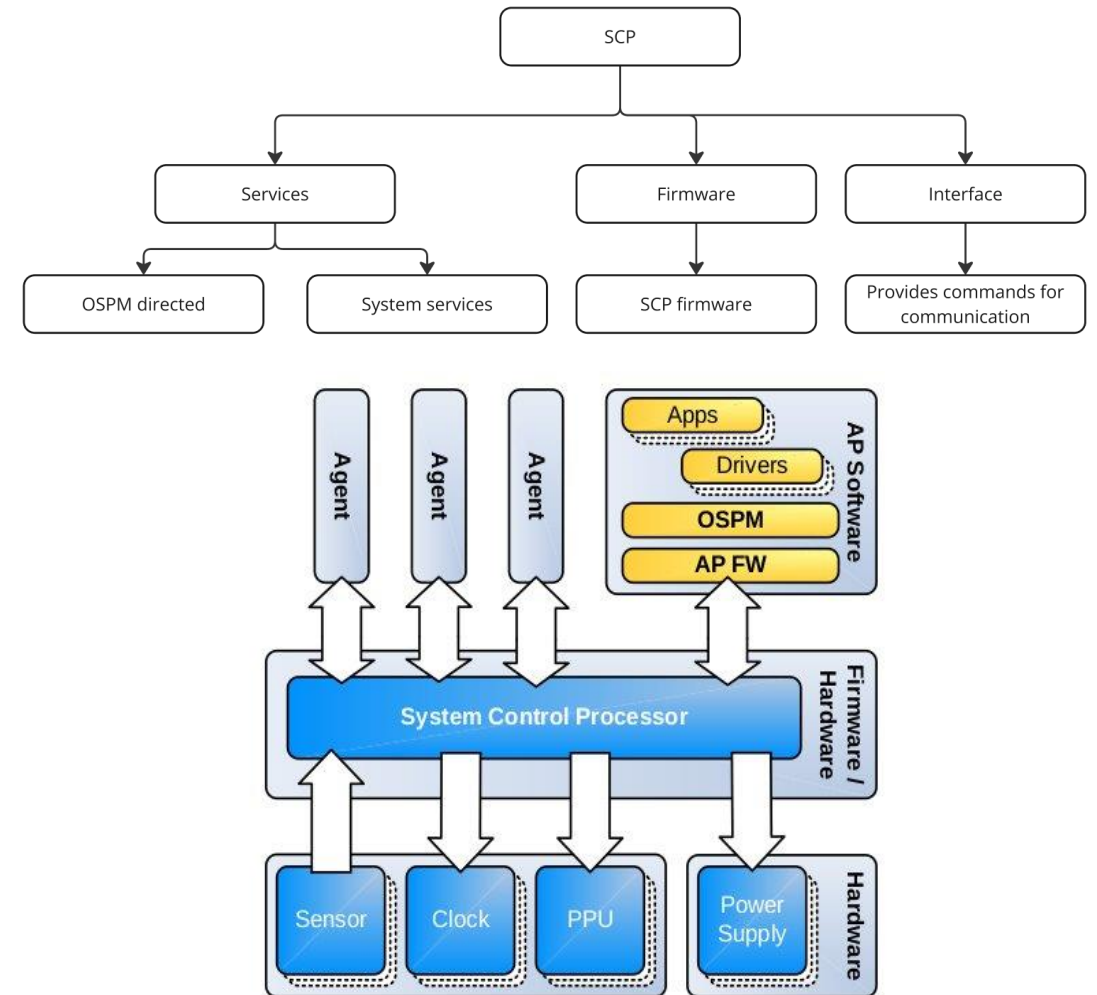
Highly complex for the OS to manage the requirements while simultaneously attempting to optimize workload performance.



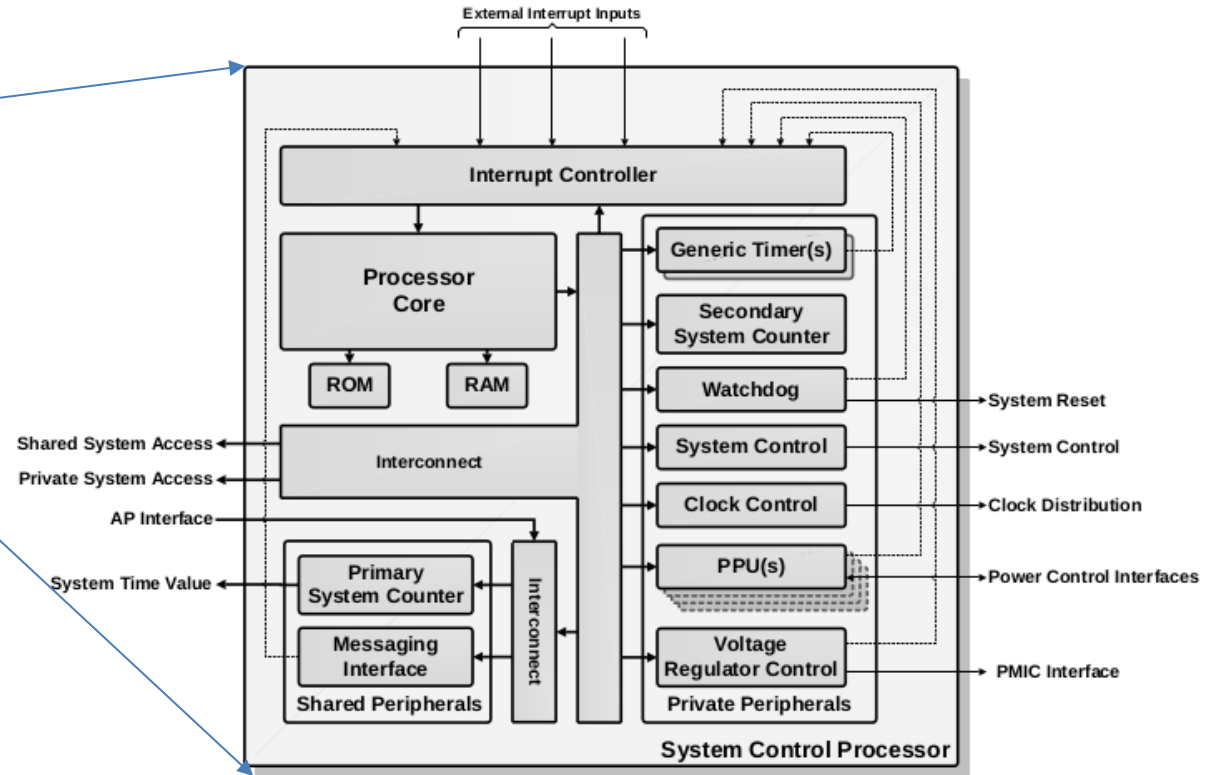
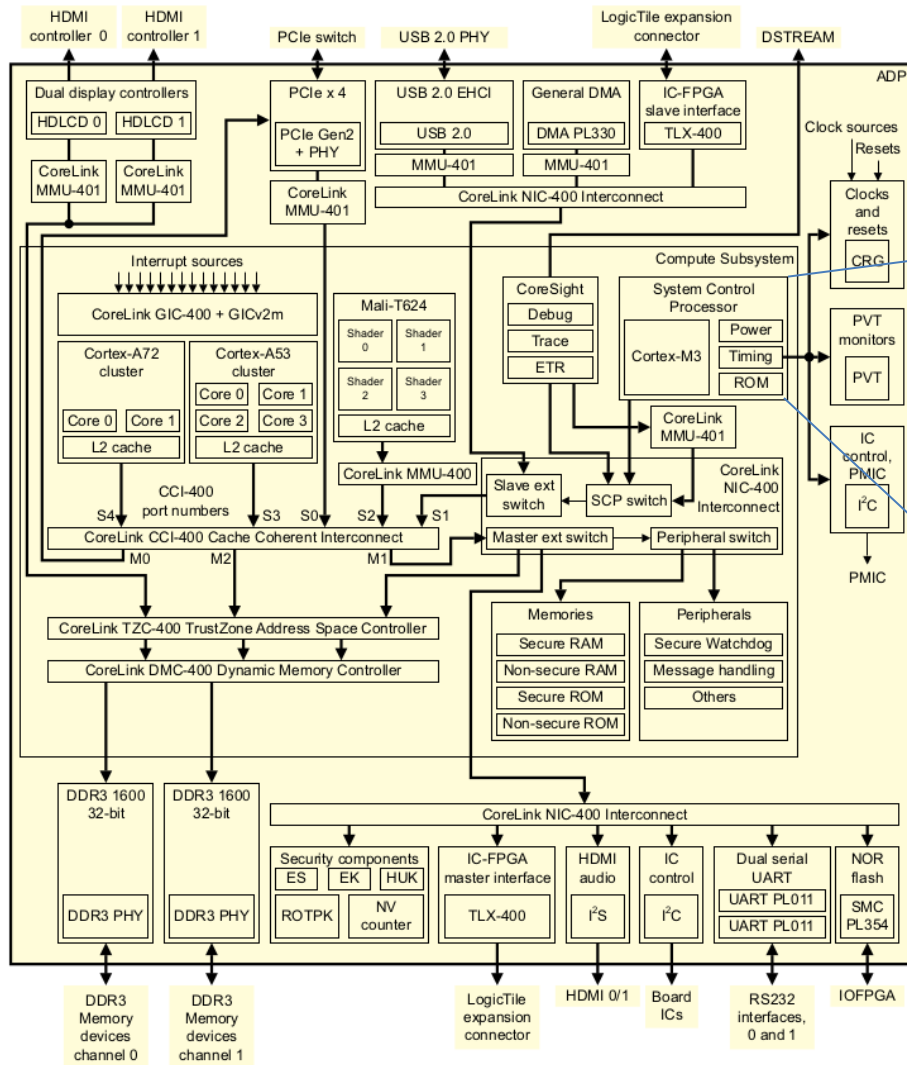
- It is clear that it is very complex for single OS kernel to manage everything.
- The motivation is to break away from traditional OS-based centralized control paradigms, in favor of delegation-based model.
- The work is delegated to SCP (System Control Processor).

System Control Processor

- SCP is a processor-based system running dedicated firmware controlling a set of hardware resources (SoC hardware).
- Used to abstract power and system management tasks away from AP's.
- It reconciles request from all agents, managing the availability of shared resources and power-performance limits according to all constraints.
- SCP is highly flexible and extensible in terms of functions and services.
- SCP provides some services and commands for services through interfaces.
- Using appropriate boot process SCP can be inherently trusted entity.



System Control Processor



Messaging interface

- Messaging interface provides hardware support for communication across agents and SCP.
- It is dependent upon type of transport.
- The requirements are
 - Should be simple to use.
 - Must be usable by all AP's and agents.

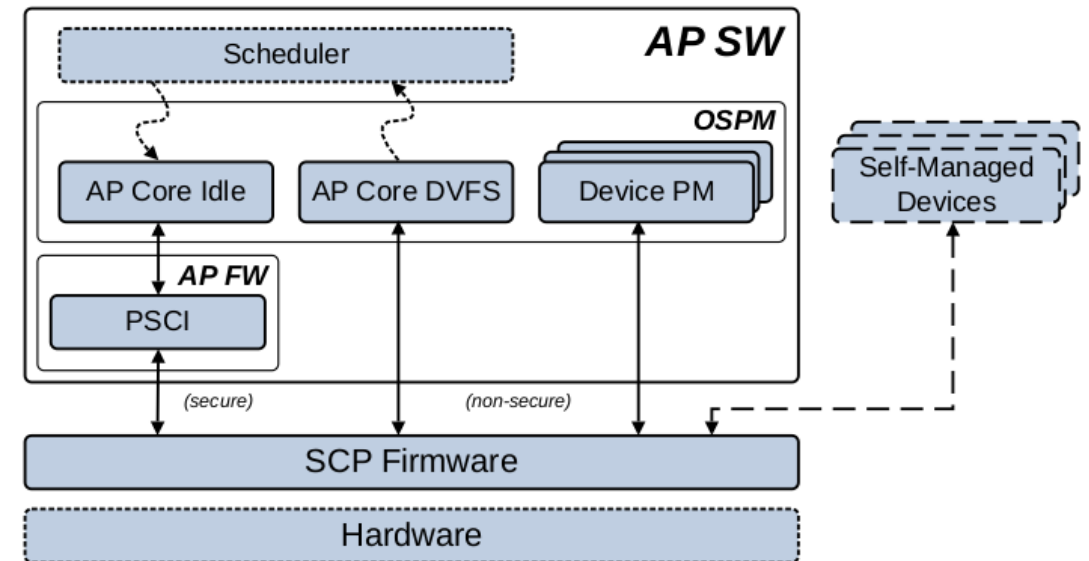
Primary System Counter

- System counters provide a consistent view of time for all AP's and debug infra.
- It is used to manage various activities like interrupt management, SoC sleep states etc.

- SCP services can be termed as basic responsibilities of SCP.
- At higher level SCP services are classified into two categories
 - OSPM directed
 - System services
- OSPM directed : Services which are requested by OS (OSPM) or by any capable agent to SCP.
- System services : Services provided by the SCP without OSPM direction.
- As per PCSA specification, only anticipation of SCP services are given.

OSPM directed services

- Under the direction of OSPM, SCP performs following
 - Voltage supply changes
 - Power control actions
 - Clock source management
- OSPM is divided into two parts
 - Core PM
 - Idle management
 - Scheduling
 - Hot-plugging
 - DVFS management
 - Device PM
 - OS based PM : when entire device and drivers depend on OSPM
 - Self-managed PM : some devices can message SCP independent of OSPM. Again these can be fully self-managed and partially self-managed.



- Services provided by SCP implicitly (without OS)
 - Response to system events : Respond to system events with appropriate resource control actions
 - Timer events : Triggering system wakeup and periodic actions like monitoring.
 - Wake events : GIC wake requests, interrupts to powered-off cores, access requests from other agents.
 - Debug access power control : Debug related controls and power management.
 - Watchdog events and system recovery actions : On watchdog timeout, SCP performs reset and re-init sequences.
 - System aware functions :
 - Reconcile requests from other agents and perform suitable operations.
 - Monitors sensor and measurement functions for operating point optimization and alarm conditions
 - For electrical and thermal protection if required overrides OSPM direction.
 - System initialization :
 - Does power-on reset init tasks, from power-on sequencing of the primary system and AP core power domains through to AP boot.

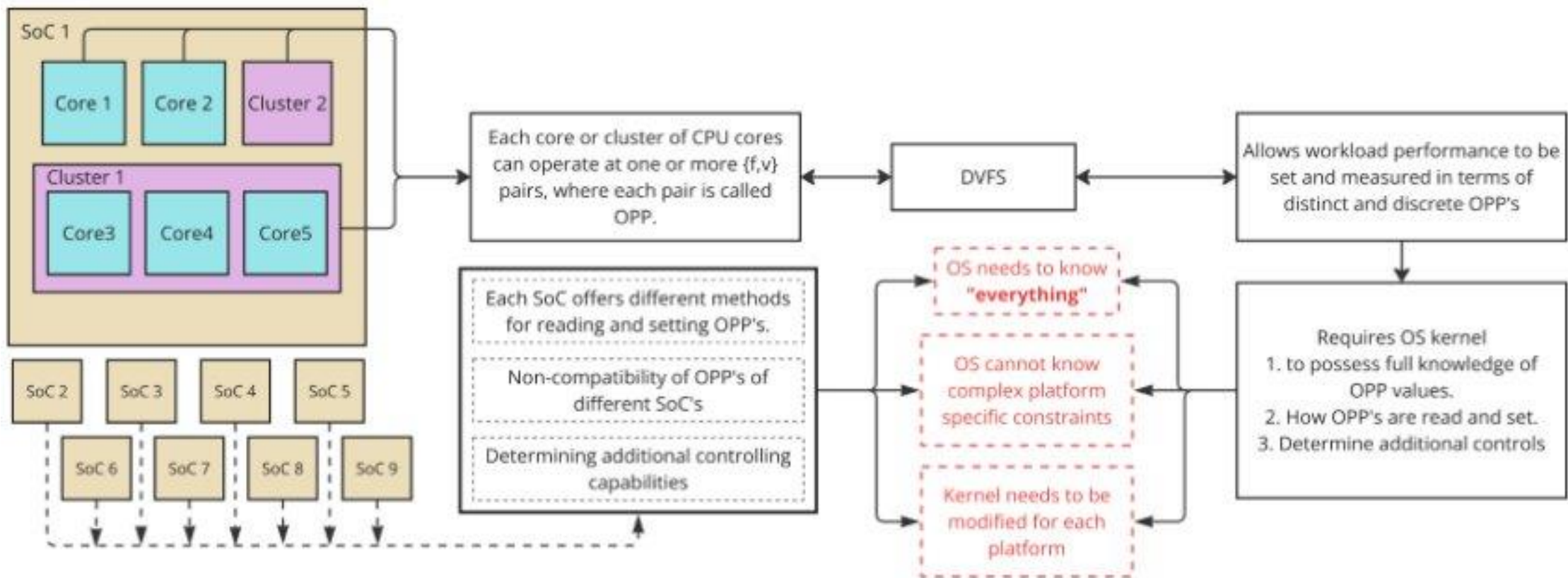
- Interface : Provides support to achieve the required functionality of services.
- SCP software interface is anticipated at minimum to support the following commands to request
 - Power states : setting of core, cluster, device and SoC power states.
 - DVFS (Dynamic Voltage Frequency Scaling) : changes the OP of DVFS capable to a desired performance point.
 - Voltage supply : changing the level of platform power supplies outside of DVFS domains.
 - Clock supply : Enabling and source freq of platform clocks outside of DVFS domains
 - Timer : Setting and cancelling of always-on wakeup timer events for specific AP core.
 - Boot : Boot time initialization.
 - Further extensions for interfaces is provided in SCMI.

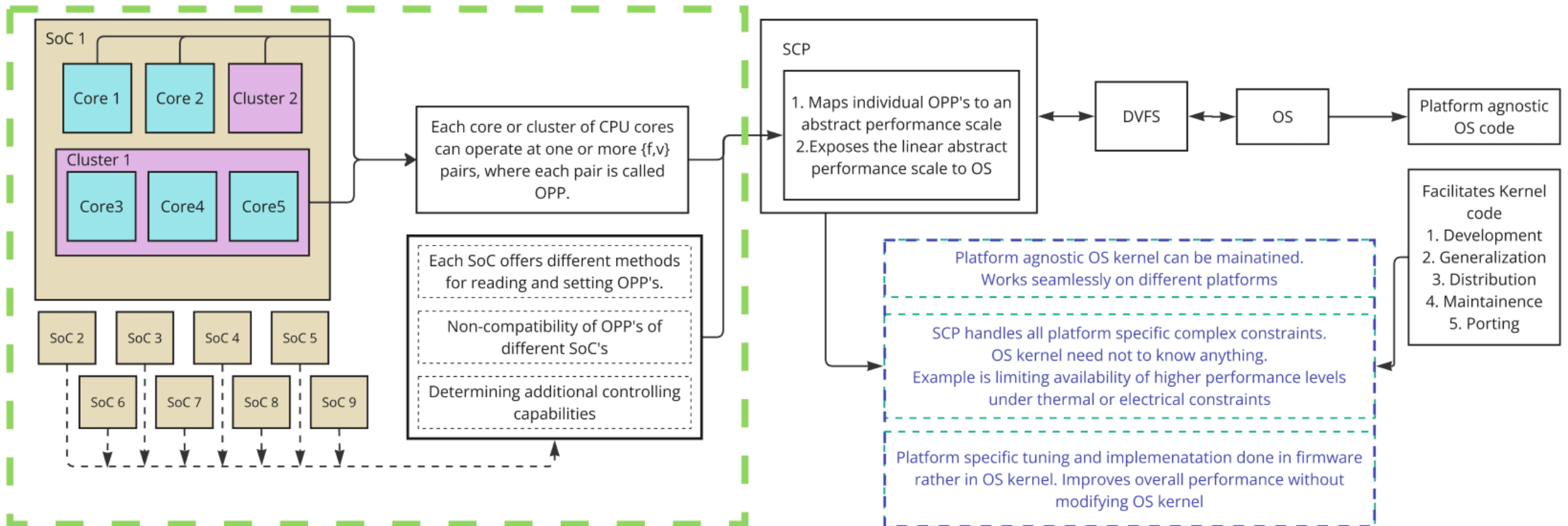
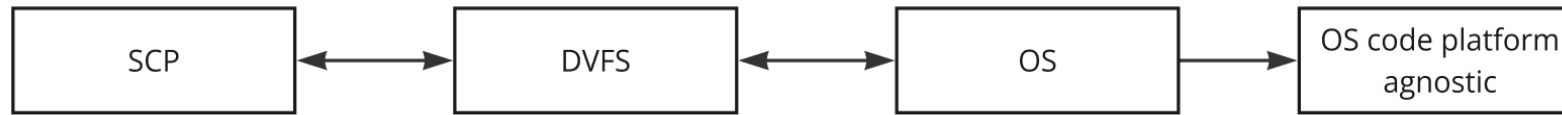


Advantages of delegation

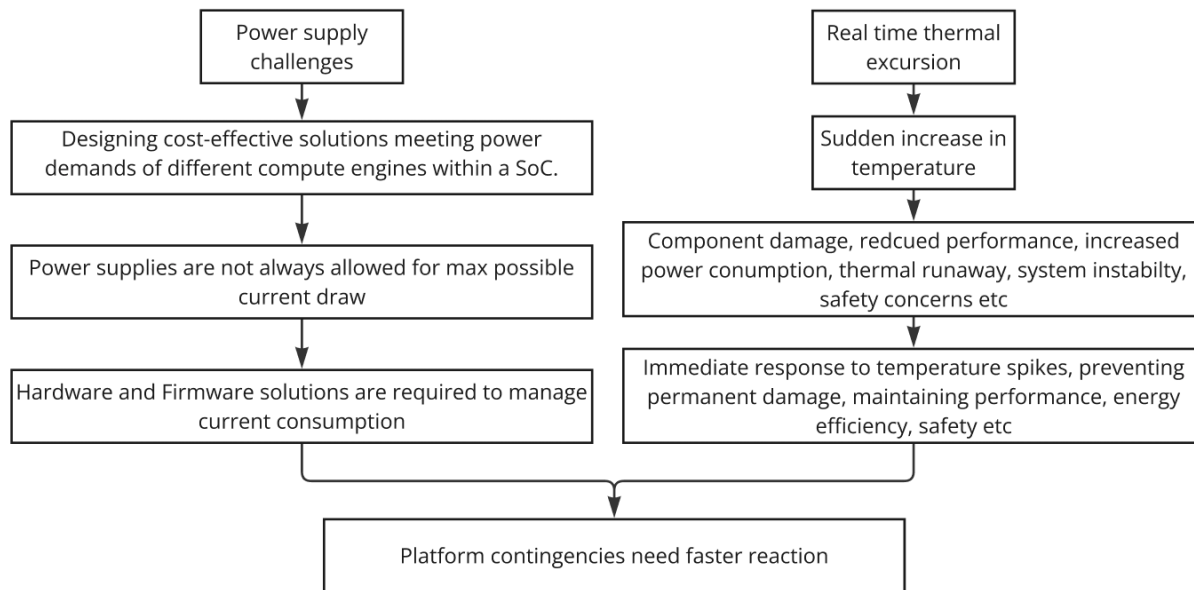
- Delegation provides the following features
 - Abstraction
 - Safe Hardware operation
 - Security

Abstraction

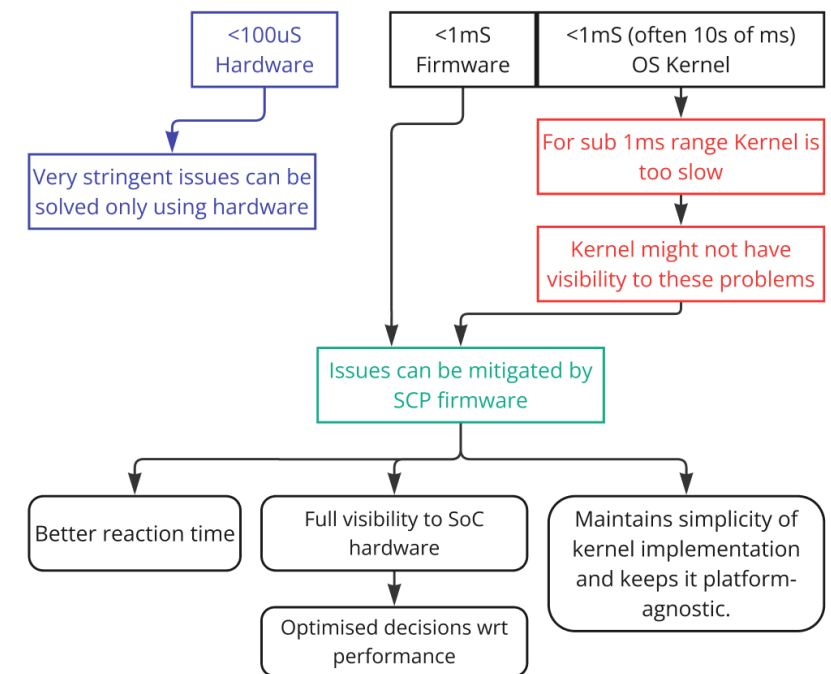




Power and thermal challenges

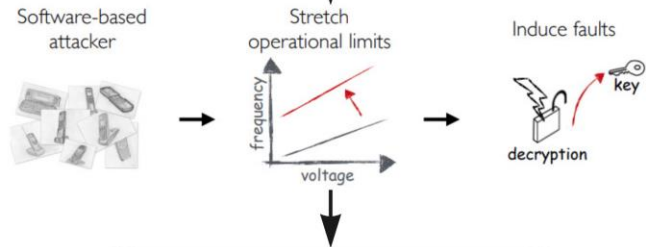


Typical reaction time and effects of hardware, firmware and kernel solutions

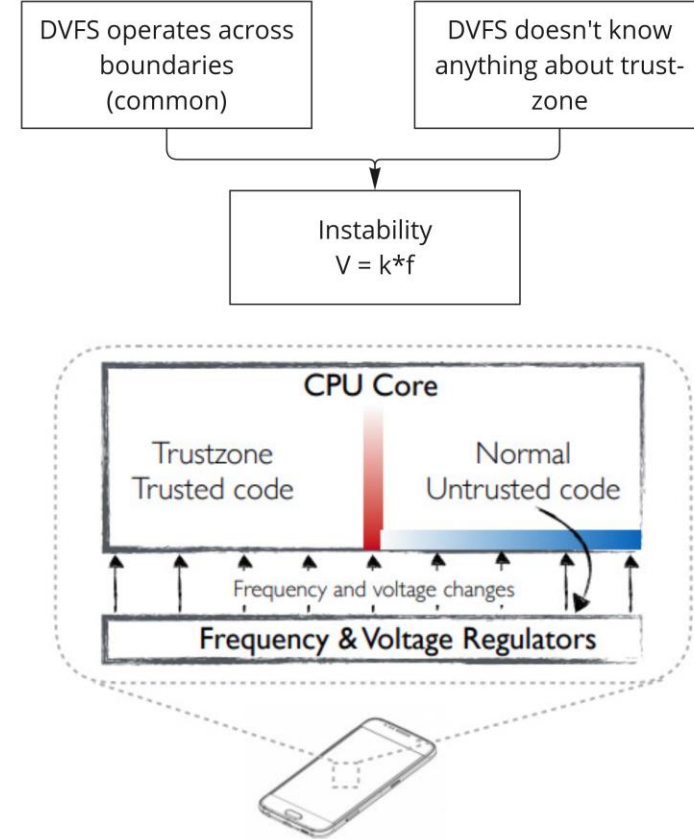
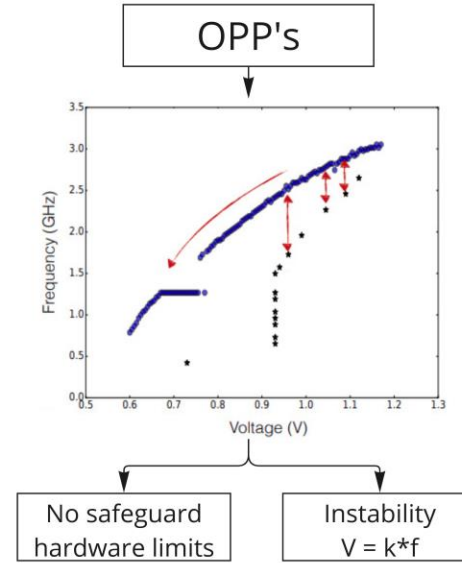
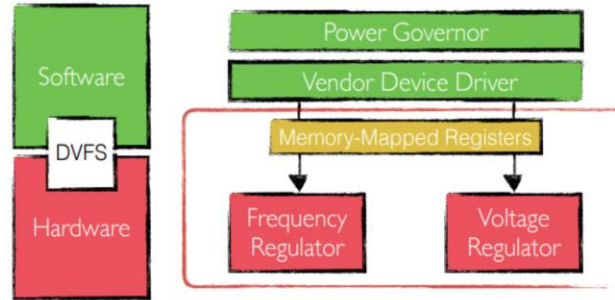
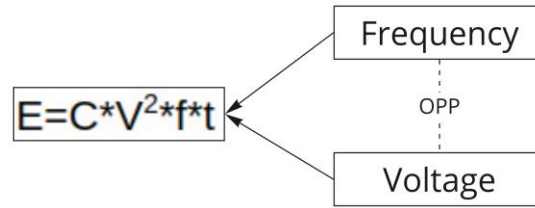


Today's systems cannot ~~exist without~~ ^{stay secure with} **Energy Management**

Exploiting software interfaces to energy management



1. New vector attack exploiting energy management
2. Practical attack on trusted computing ARM devices.
3. Impacts hundreds of million devices.



This attack is known as CLKsCREW attack



Conclusions on security

- New attack surface via energy management software interfaces.
- Not a hardware or software bug, rather fundamental design flaw in energy management mechanisms.
- Security of energy management must be taken into consideration. Two requirements for mitigating CLKsCREW attack.
 - Kernel must not have direct and independent control of clock and voltage.
 - Trusted entity such as SCP firmware, must be able to perform sanity check on requested performance levels, thereby preventing any attempted malicious programming.
- The above requirements can be met with kernel implementation that delegates the management of the control to platform through abstracted interface.



A quick recap

- We have seen the present market trend.
- The challenges in the implementation of modern single OS kernel.
- Motivation for new delegated based model : SCP
- Basic overview of SCP.
- High level understanding of SCP services and interfaces.
- Advantages of using SCP.



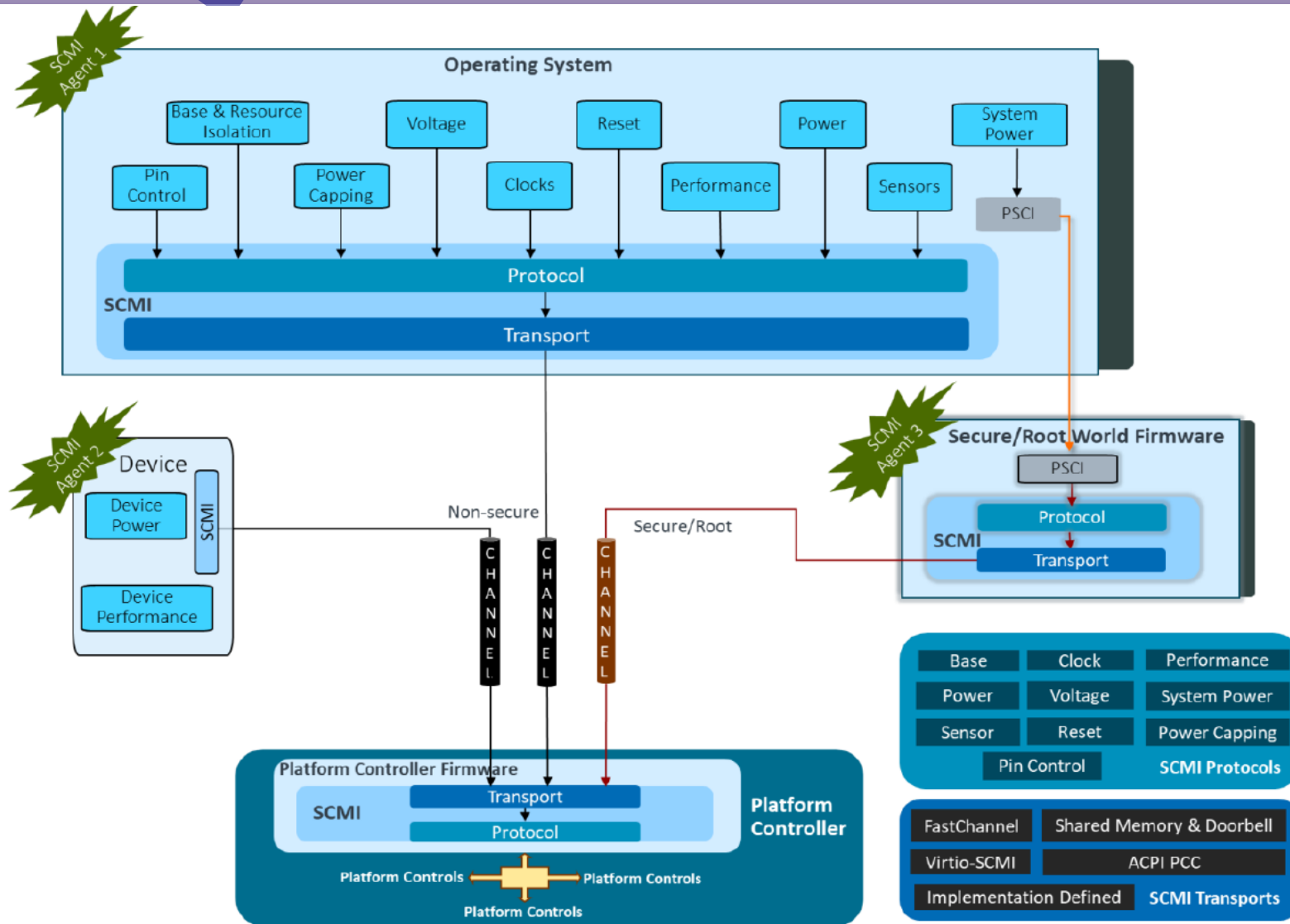
How do Application Processors communicate with SCP ?



System Control and Management Interface

- It is a set of standardized OS independent software interface that are used in system (SoC) management.
- SCP provides interfaces to its clients (agents) for
 - Discovery and self-description of the interface it supports
 - Different domains.
- SCMI interface provides two levels of abstraction
 - Protocols
 - Transports
- The interface is intended to be described in firmware using FDT or ACPI.

Background



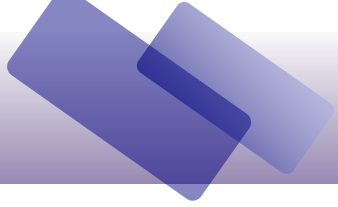
Protocol : Group of SCMI messages.

Agent : SCMI caller

Platform : Interprets SCMI messages. (SCP)

Resource : Hardware controlled using SCMI messages.

Transport : Method through which SCMI messages are communicated.

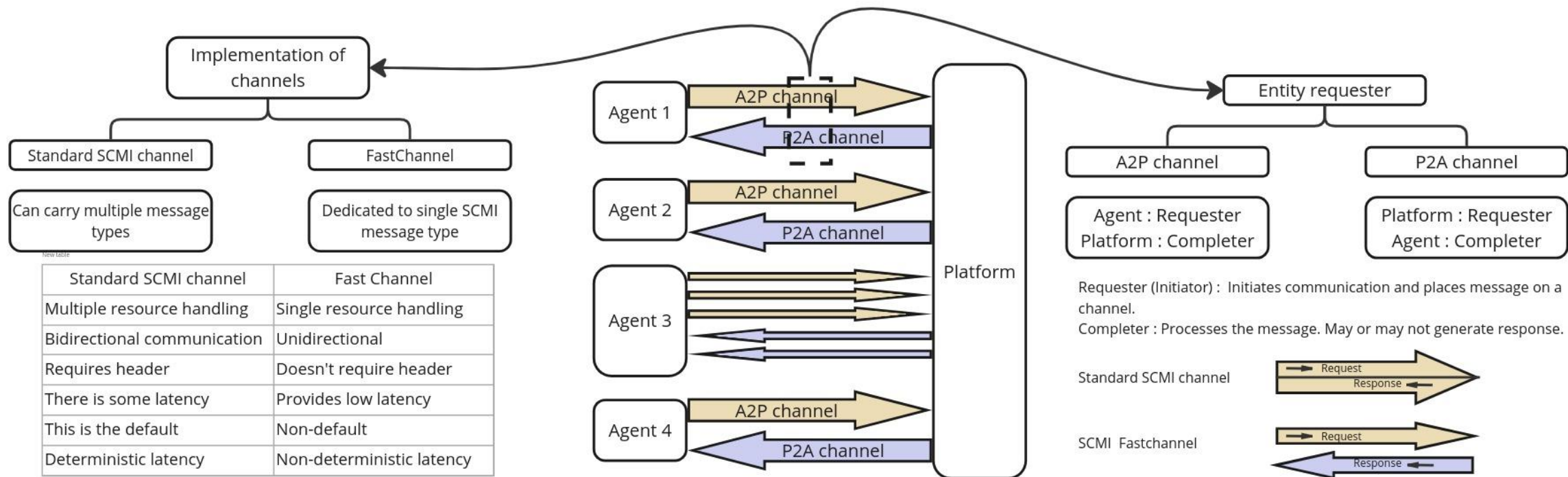


SCMI Protocols

The topics in the SCMI protocols are

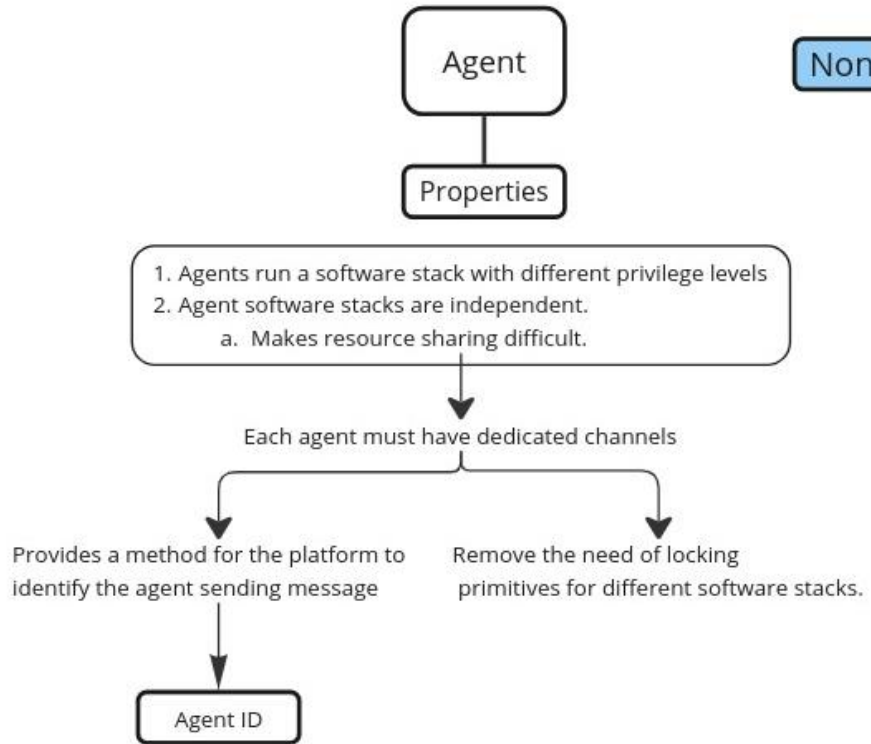
- Agents, messages and channels
- Message format
- Protocol discovery
- SCMI status codes

Channels



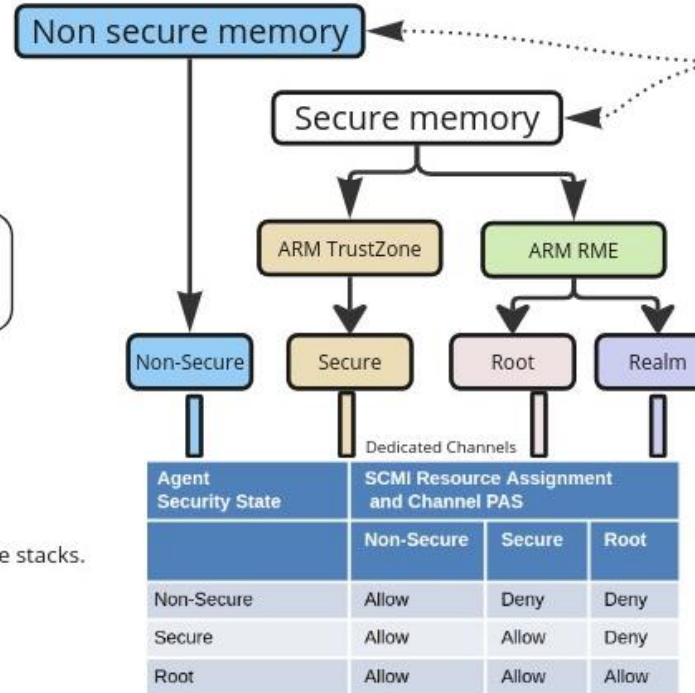
Agents

Agent basic properties

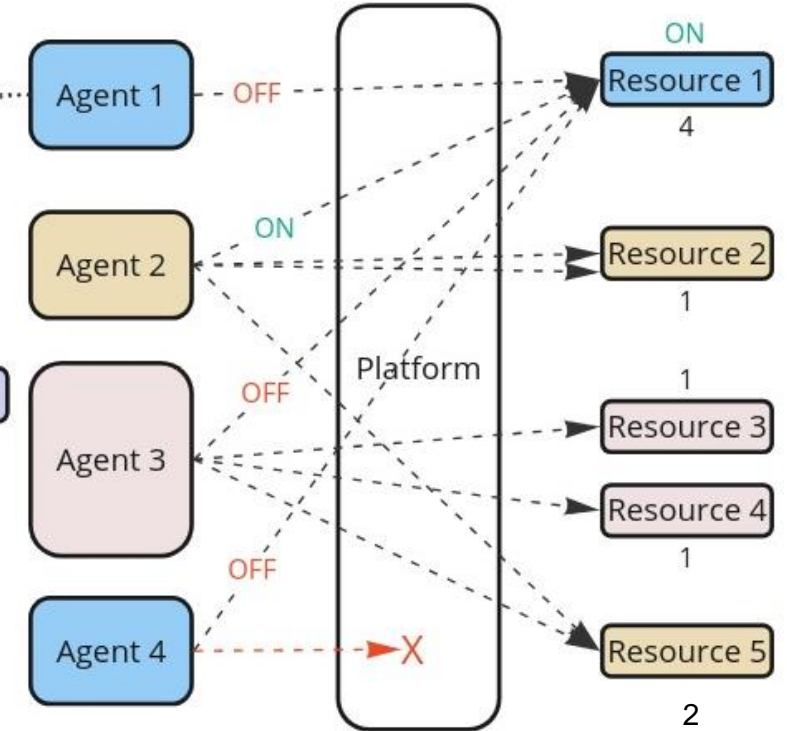


1. Every agent has a unique agent ID.
2. Used by platform to identify the agents communicating with it.
3. Assigned by platform statically.
4. Agent discovers the id by the channel it owns. It is done by base protocol.

Expected agent behaviour in different channels



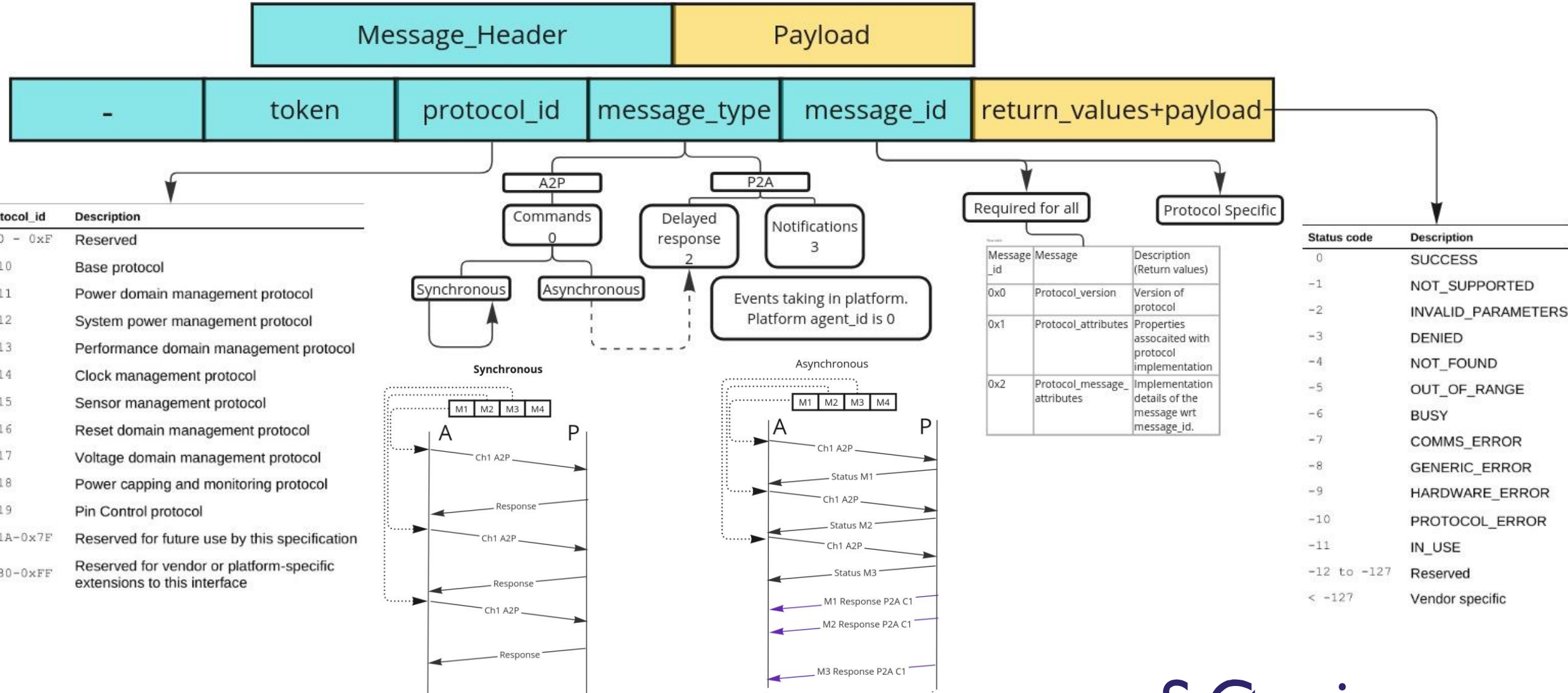
Multi agent- multi resource management



1. Resources that are shared between multiple agents should be in a state which meets the requirements of all agents sharing the resource.
2. Any agent that needs to use a shared resource should explicitly make a request for it.
3. The platform may disable a resource if no agent has requested to use that resource
4. For resource management, only inter-agent reference counting is used.
5. Resource assignment shall ensure the security guarantees provided by ARM TZ and RME are not violated.

Messages

Messages are used by agents to make requests to the platform





SCMI Protocols

Base Discovery Protocol

Discovers platform information. Such as protocol version, Vendor name, supported device/protocol permissions and agent configuration set

Power Domain Management

Put device/domain to different power saving states.

System Power Management Protocol

Intended for system shutdown, suspend and reset

Performance domain management protocol

Performance management of groups or devices or AP's that run in the same performance domain.

Sensor Management Protocol

Sensor readings, events and settings.

Reset Domain Management Protocol

Device/Domain reset handling and notifications.

Clock Management Protocol

Get/Set clocks, clock rate change notification

Power Capping and monitoring Protocol

Set power caps and monitor power consumptions.

Pin Control Protocol

Pin config set/get, pin function select.

Voltage Domain Management Protocol

Mode[On/Off] config, voltage level config.

- The only mandatory protocol.
- Provides commands for
 - Protocol version
 - Implementation attributes and vendor identification
 - List of implemented protocols.
 - Agents in the system.
 - Register for notifications for events and platform errors.
 - Configure the platform to control and modify an agent's visibility of platform resources and commands.
 - Trusted agent.



Base Protocol Commands

Protocol_Id	Command Name	Description
0x0	PROTOCOL_VERSION	Version of protocol (0x20001), code->0x20000
0x10	NEGOTIATE_PROTOCOL_VERSION	Agent negotiates when it doesn't support PROTOCOL_VERSION
0x1	PROTOCOL_ATTRIBUTES	Returns implementation details associated with this protocol.
0x2	PROTOCOL_MESSAGE_ATTRIBUTES	Returns implementation details associated with a specific message in this protocol.
0x3	BASE_DISCOVER_VENDOR	Provides vendor identifier ASCII string
0x4	BASE_DISCOVER_SUB_VENDOR	Provides sub-vendor identifier ASCII string
0x5	BASE_DISCOVER_IMPLEMENTATION_VERSION	Provides vendor-specific 32-bit implementation version
0x6	BASE_DISCOVER_LIST_PROTOCOLS	Allows the agent to discover which protocols it is allowed to access
0x7	BASE_DISCOVER_AGENT	Allows the caller to discover the name of agent up to 16 bytes.
0x8	BASE_NOTIFY_ERRORS	Provides notifications of errors in the platform to agent.
0x9	BASE_SET_DEVICE_PERMISSIONS	Allowed agent can change permissions of device for a given agent.
0xA	BASE_SET_PROTOCOL_PERMISSIONS	Allowed agent can change protocol permissions for a given agent.
0xB	BASE_RESET_AGENT_CONFIGURATION	Used to reset the platform resource settings to default settings

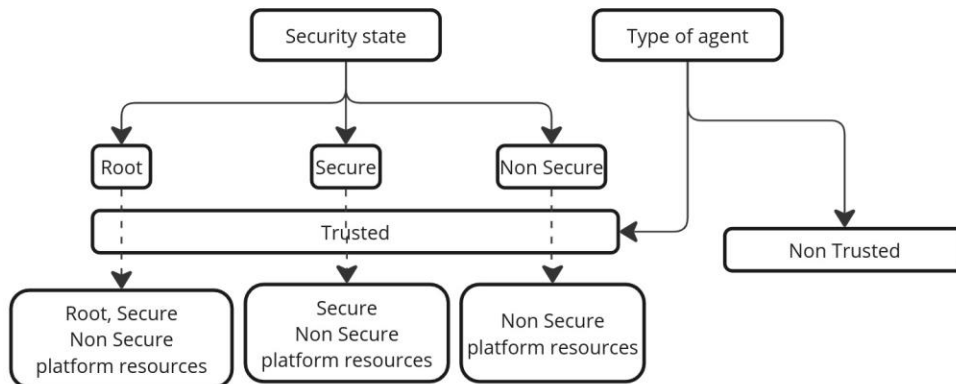
Trusted agent : Agent who has elevated privileges to configure and control the access rights of other agents in the system

Domain : A group of hardware resources that are managed collectively.

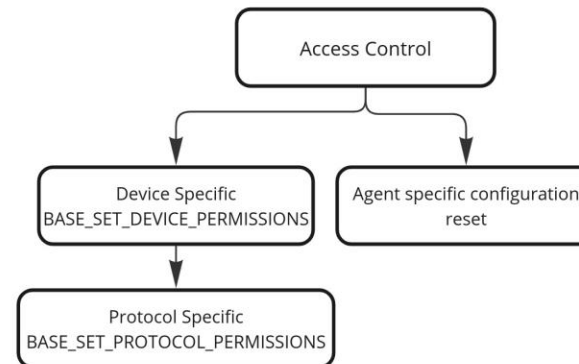
There are many types of domains depending upon hardware resources

Platform resources : Typically refer to power domains, performance domain, clocks, sensors, reset and voltage domains

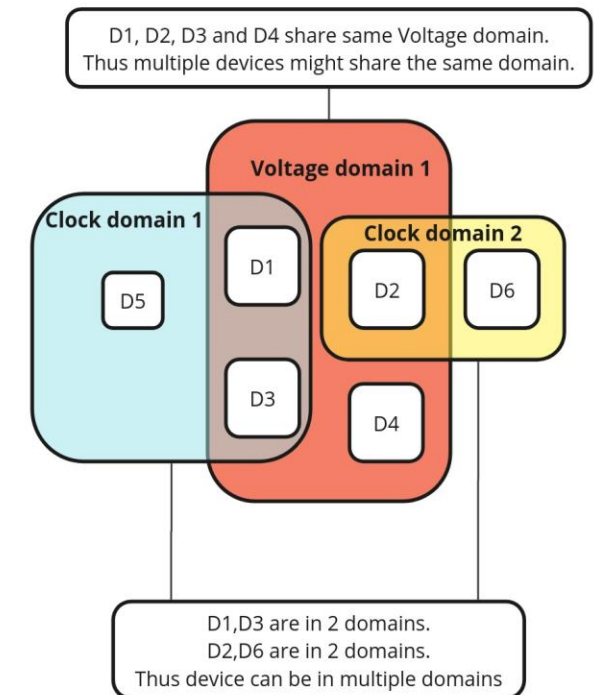
Agent permissions management w.r.t Agent security state

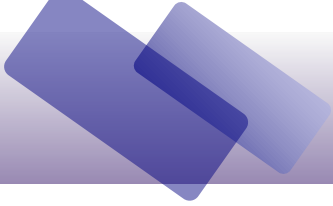


Fine grain permission control

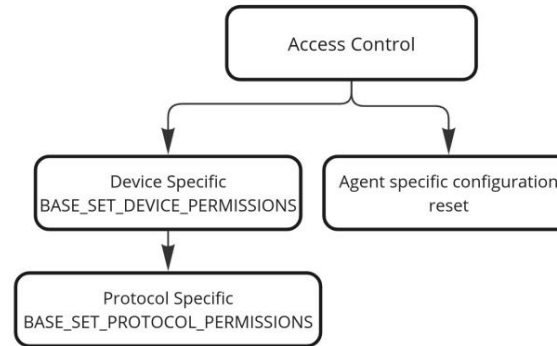


Devices and Domains

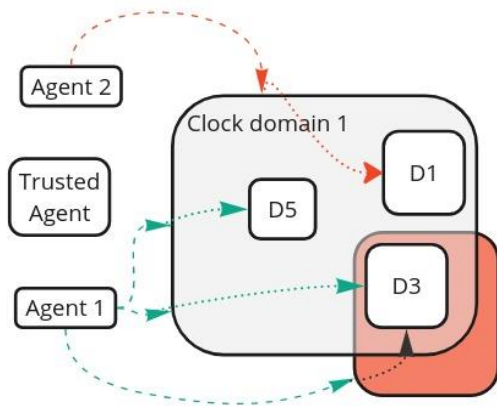




Fine grain permission control

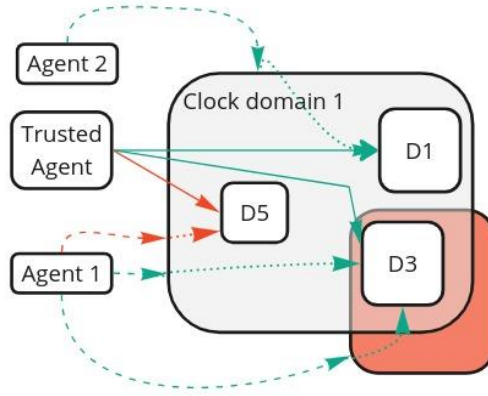


Initial Agent Permissions



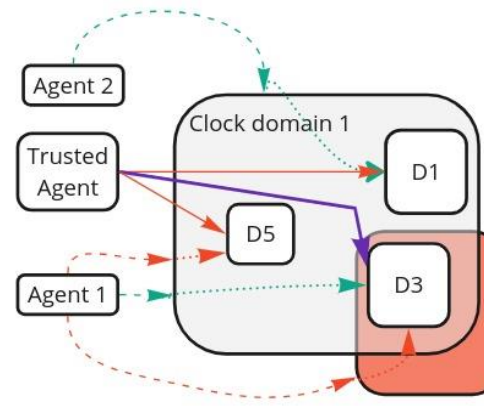
System boots with default permissions
Implementation defined

Device Specific BASE_SET_DEVICE_PERMISSIONS



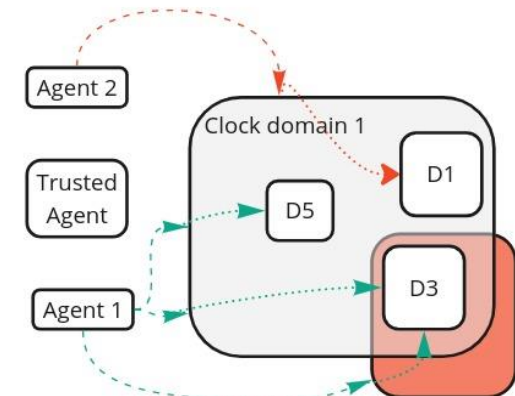
Device access -> Resource access.
Used to give access to a device to a specific agent
Recommended only trusted agent should do

Protocol Specific BASE_SET_PROTOCOL_PERMISSIONS



Fine-grain resource management.
Used to configure the protocol permissions
Recommended only trusted agent should do

Configuration reset BASE_RESET_AGENT_CONFIGURATION



Reset all platform resource configurations.
Follows rules of agent security states and resource access.
Used when all run-time permissions are removed or agent becomes unresponsive

- Transports provide a physical link through which communication occurs.
- Should allow the largest possible message described in SCMI specification.
- Transport might support multiple channels.
- SCMI provides the following transports
 - Shared memory and doorbell.
 - Fastchannel
 - ACPI PCC -> SCMI transport channels can be represented as ACPI Platform Communication Channel of Type 3.
 - Virtio-SCMI -> Supports Virtio channels to use SCMI in Virtualization concept
 - Implementation defined



Shared memory based transport

- Uses shared memory for transports.
- Each channel in the transport includes
 - Shared memory area
 - Area of memory that is shared between the caller and callee.
 - Doorbell
 - Mechanism of caller to alert the callee of presence of new message.
 - If callee is in interrupt mode -> caller generates interrupt. Else doorbell is optional.
 - If callee is in secure world / hypervisor -> doorbell implementation is done using SMC or HVC.
 - Completion interrupt
 - Optional mechanism where callee raises interrupt to caller upon processing message.
- The shared memory, doorbell and completion interrupt details are passed to OS through FDT or ACPI.

Message communication flow

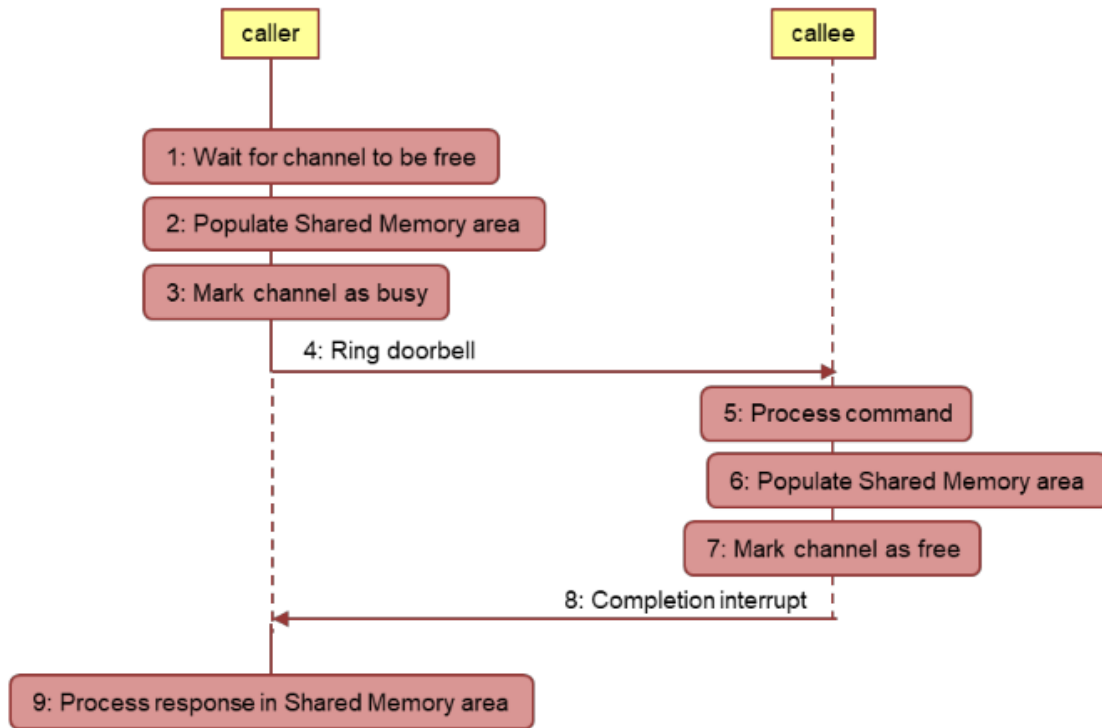


Figure 8 Interrupt-driven Communications flow

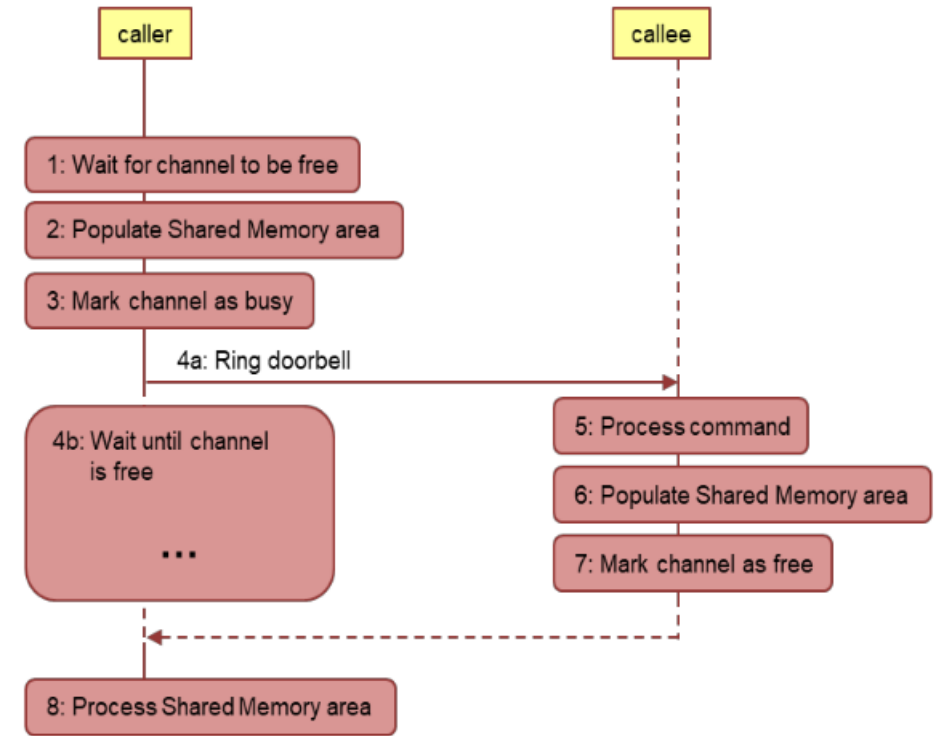


Figure 9: Polling based Communication Flow

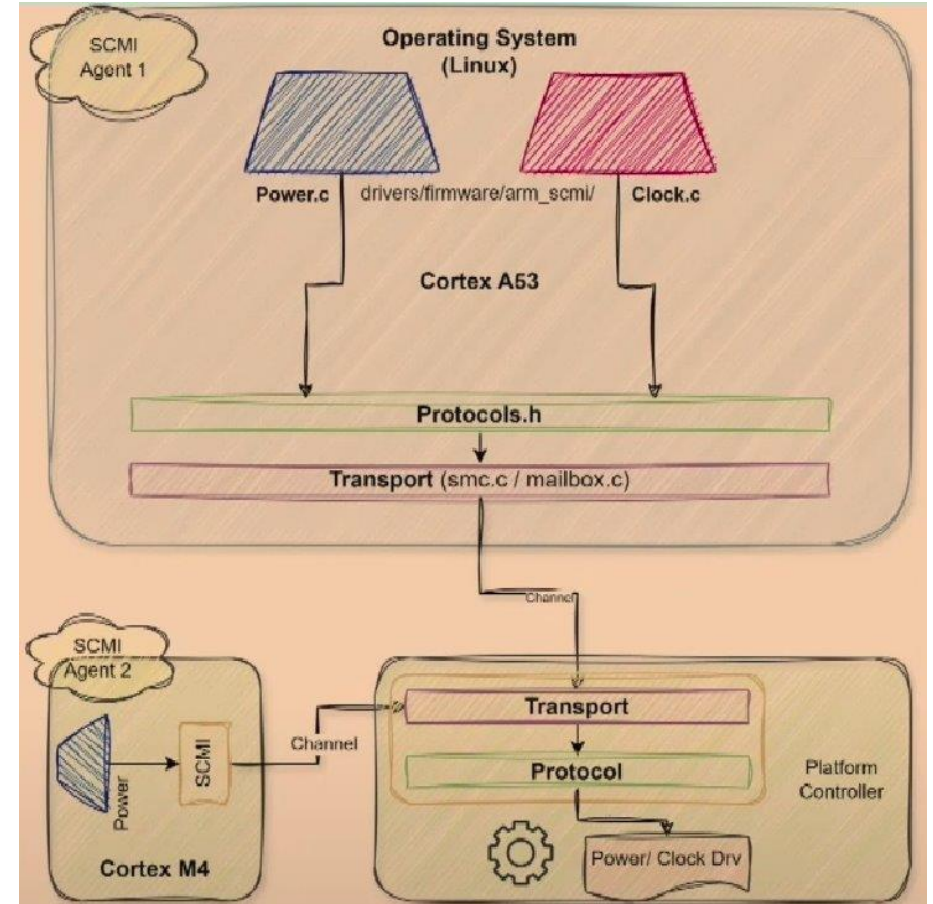


SCMI client-server connections

- We have primarily seen that SCP firmware is deployed on a dedicated SCP.
- There are several supportive mechanisms for SCMI client-server connections. They are
 - Client to server that is handled by Trusted OS
 - Client to server handled by EL3 firmware
 - SCMI sessions going through hypervisor
 - SCMI sessions via virtio
- But SCP firmware on dedicated SCP its own advantages compared to others with fewer drawbacks.

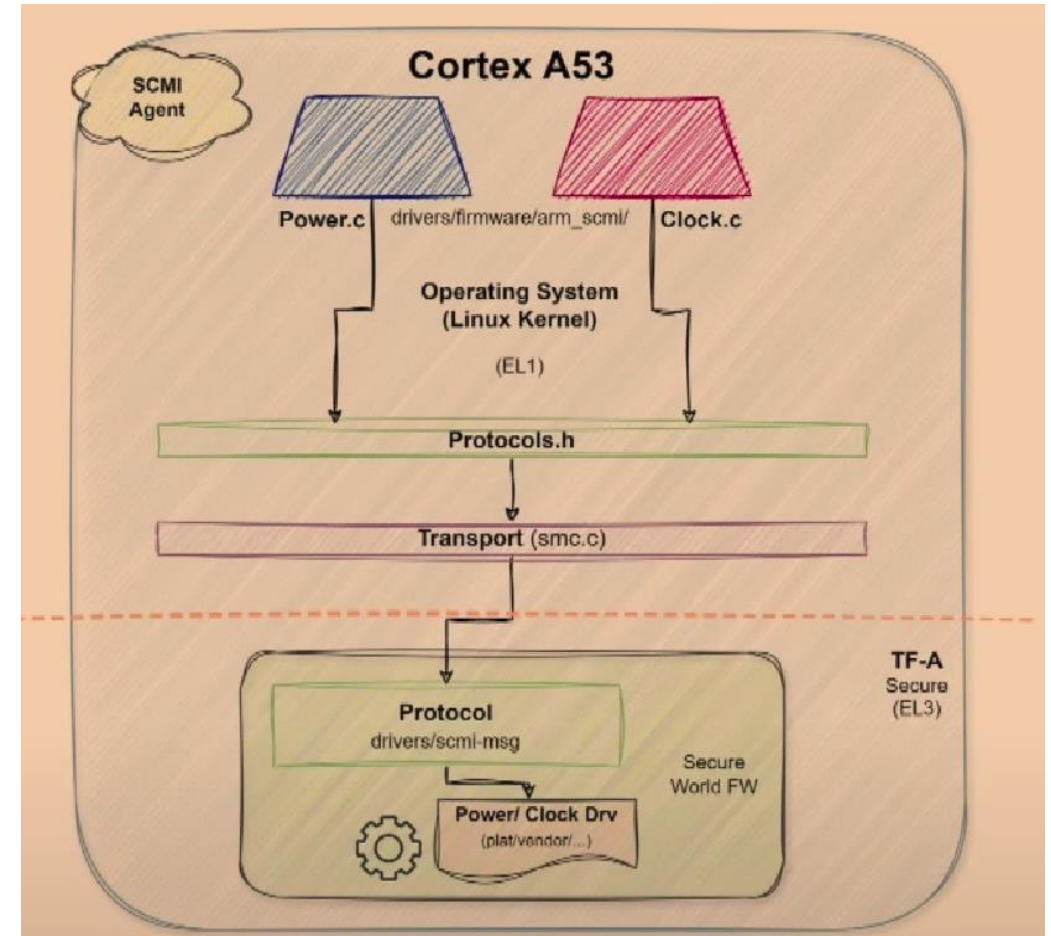
SCP firmware on a dedicated SCP

- Advantages
 - It has intelligence capability
 - Flexible , extensible and autonomous.
 - SCP consumes low power and AP's can work at full potential.
 - Simple management of platform resources for SMP's and AMP's.
 - Real time working possible
- Drawbacks
 - Dedicated processor -> cost is high
 - Heavily based on use-case.
 - Once tape-out no modifications to SCP.



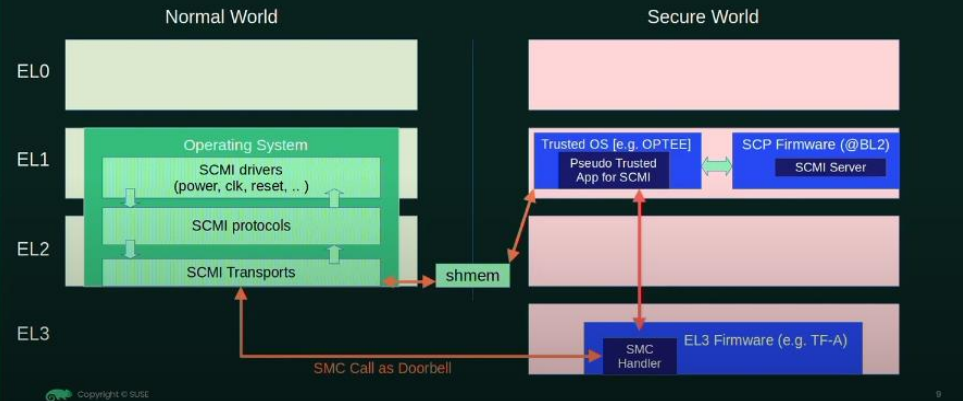
SCP firmware in Secure World

- Advantages
 - SCP can be implemented without dedicated SCP
-> low cost.
 - Relatively easy to implement.
 - Is okay for single processor and less number of peripherals.
- Drawbacks
 - Limited/no support to SMP's and AMP's
 - Performance overhead
 - Limited scalability and highly complex.



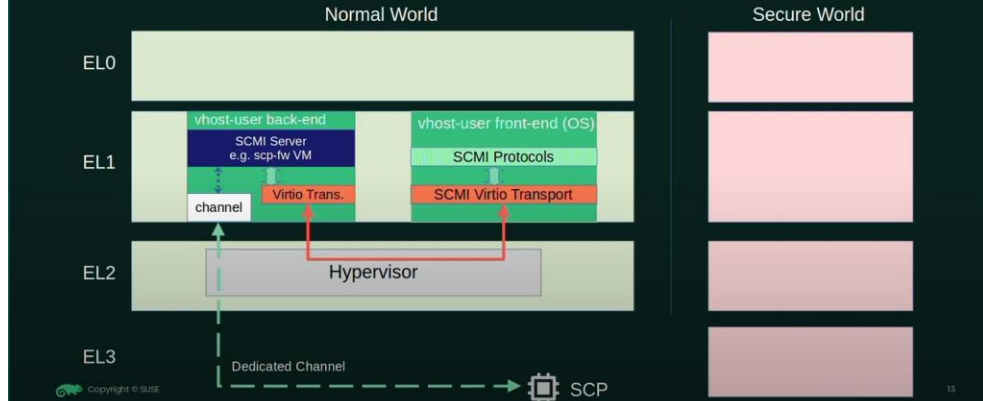
SCMI client-server connections

Client to server that handled by Trusted OS



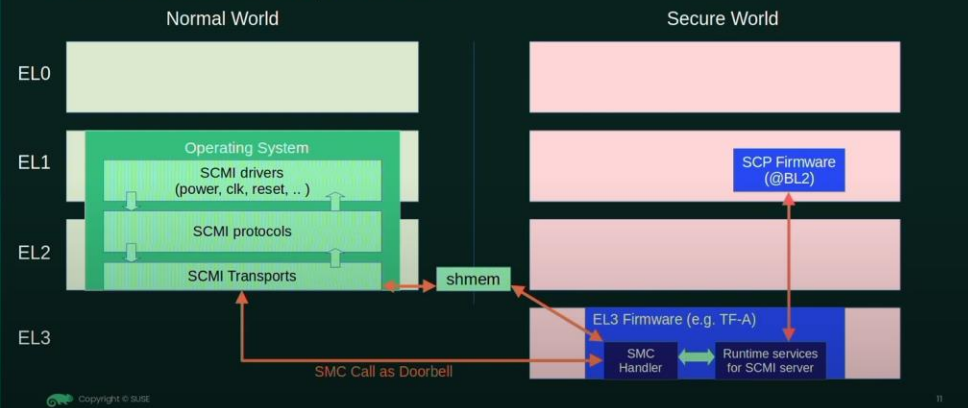
SCMI client-server connections

SCMI sessions via virtio



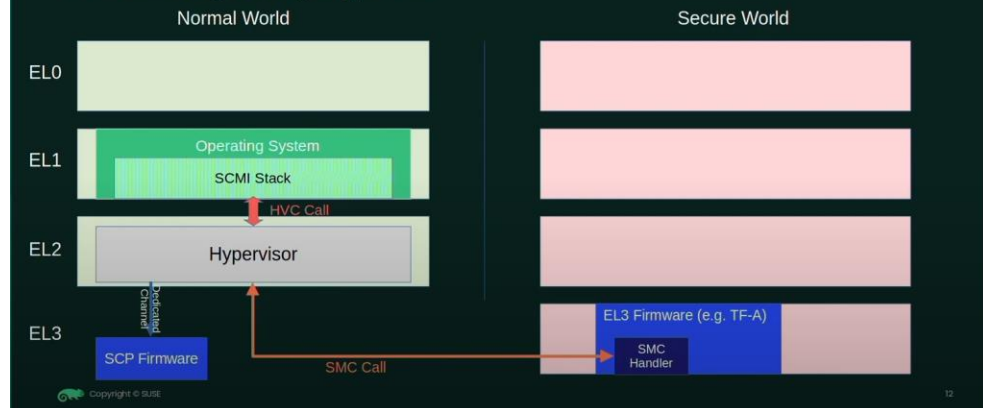
SCMI client-server connections

Client to server handled by EL3 firmware



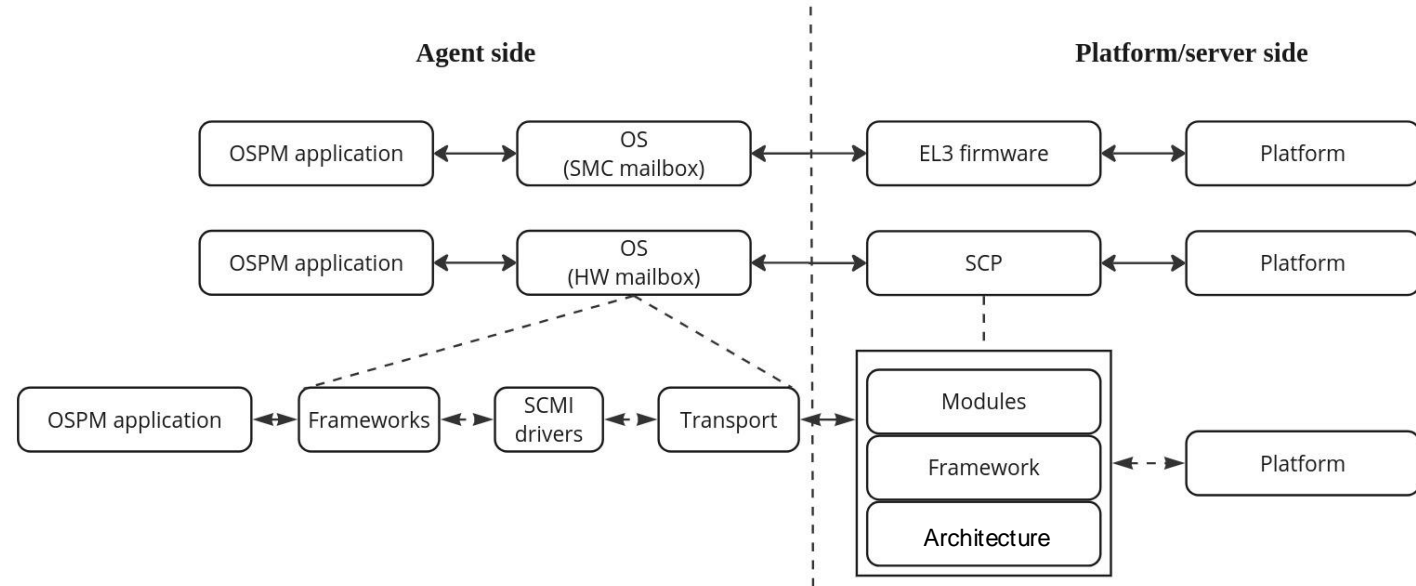
SCMI client-server connections

SCMI sessions go through hypervisor



SCP implementation

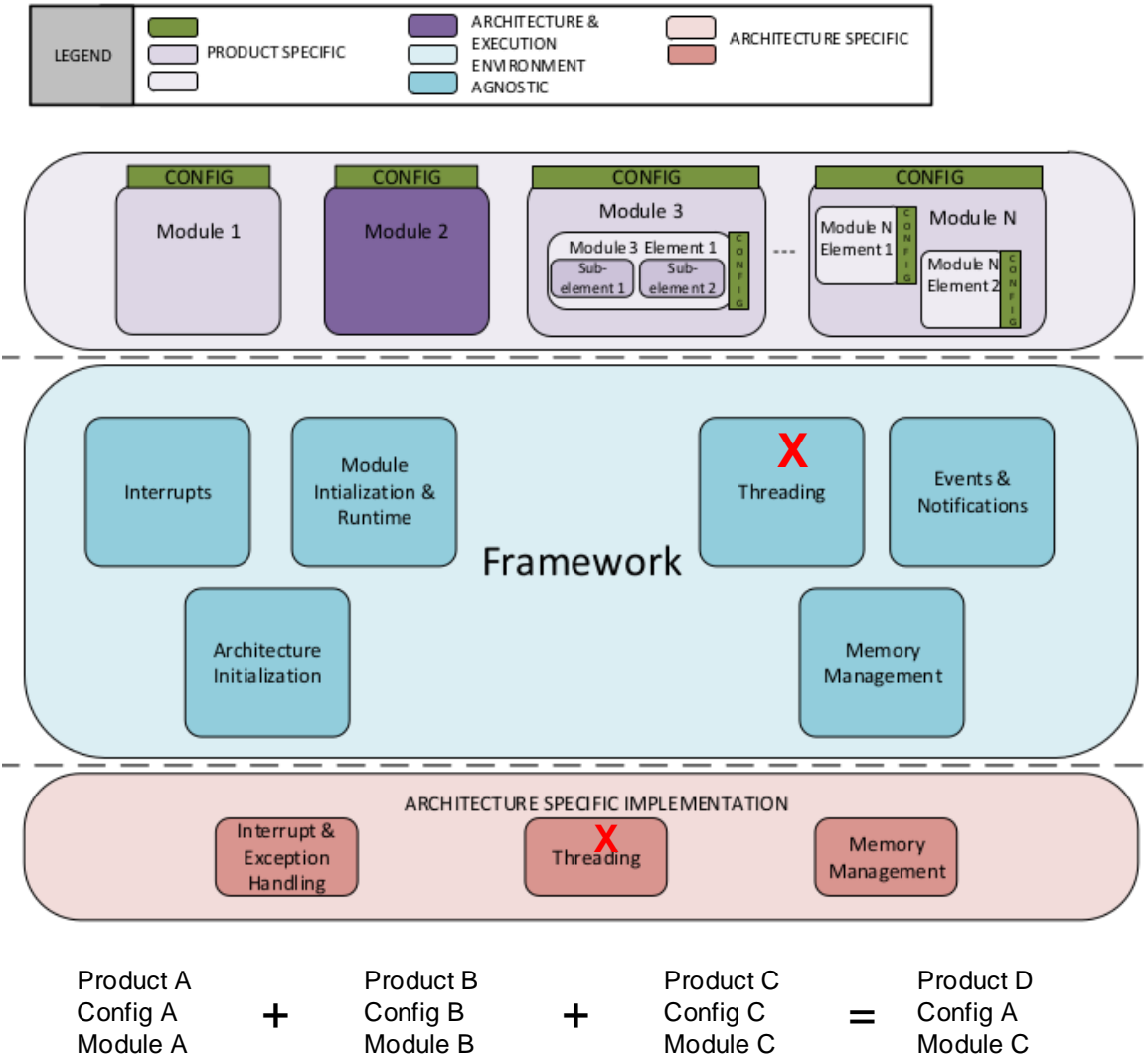
- SCMI is OS agnostic and platform agnostic.
- The implementation is dependent upon "agent" and "platform".
- SCP platform side code is very simple to understand and similar implementation for any platform.
- Agent side implementation is complex particularly if the agent is OS.
- Thus, for a complete transaction we need to see both agent and platform side code.
- In this presentation our major focus lies on SCP platform code.

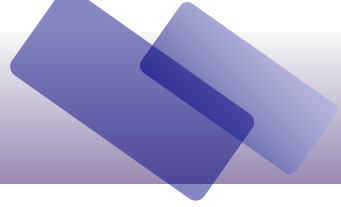


Basic building blocks

Three major layers

- Architecture:
 - Provides execution environment dependent functionality like interrupts, memory management etc.
- Framework:
 - Provides common services to all modules like Initialization, Event, Notification & Interrupt handling.
 - Depends on architecture layer for execution environment dependent services
 - Architecture and execution environment agnostic
 - Drives the initialization, and manages proper coordination & interactions between modules
- Modules:
 - Architecture agnostic
 - A module performs a well-defined set of operations.



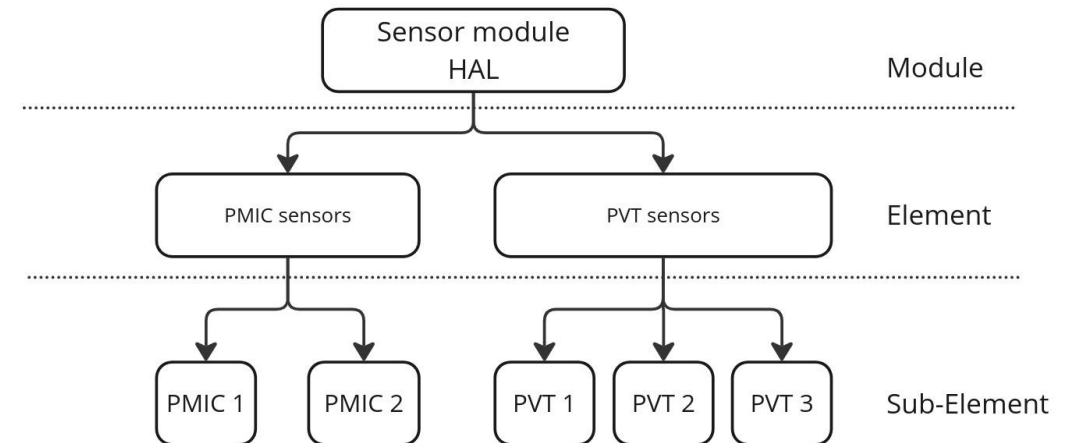


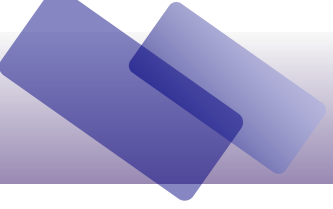
Modules

- Everything is a module in SCP firmware.
- Each modules provides a single piece of functionality.
- Combination of modules lead to overall functionality.
- There are various types of modules based on operations. (informative for programmer)
 - HAL :
 - Module that provides functionality for one or more types of hardware device through standardized interfaces that abstract away differences between the supported devices.
 - Depends on other modules at lower level.
 - Driver :
 - Controls a specific device/ class of device.
 - Implements the API's defined by HAL.
 - Driver module can be standalone without an associated HAL module.
 - Protocol :
 - Provides one or more APIs so that other modules can make use of it.
 - Services :
 - Performs work and/or Provides non-hardware device functionality.
- Every module must declare and define a structure of type "**struct fwk_module**"
- The name of the module should be "**module_modulename**".

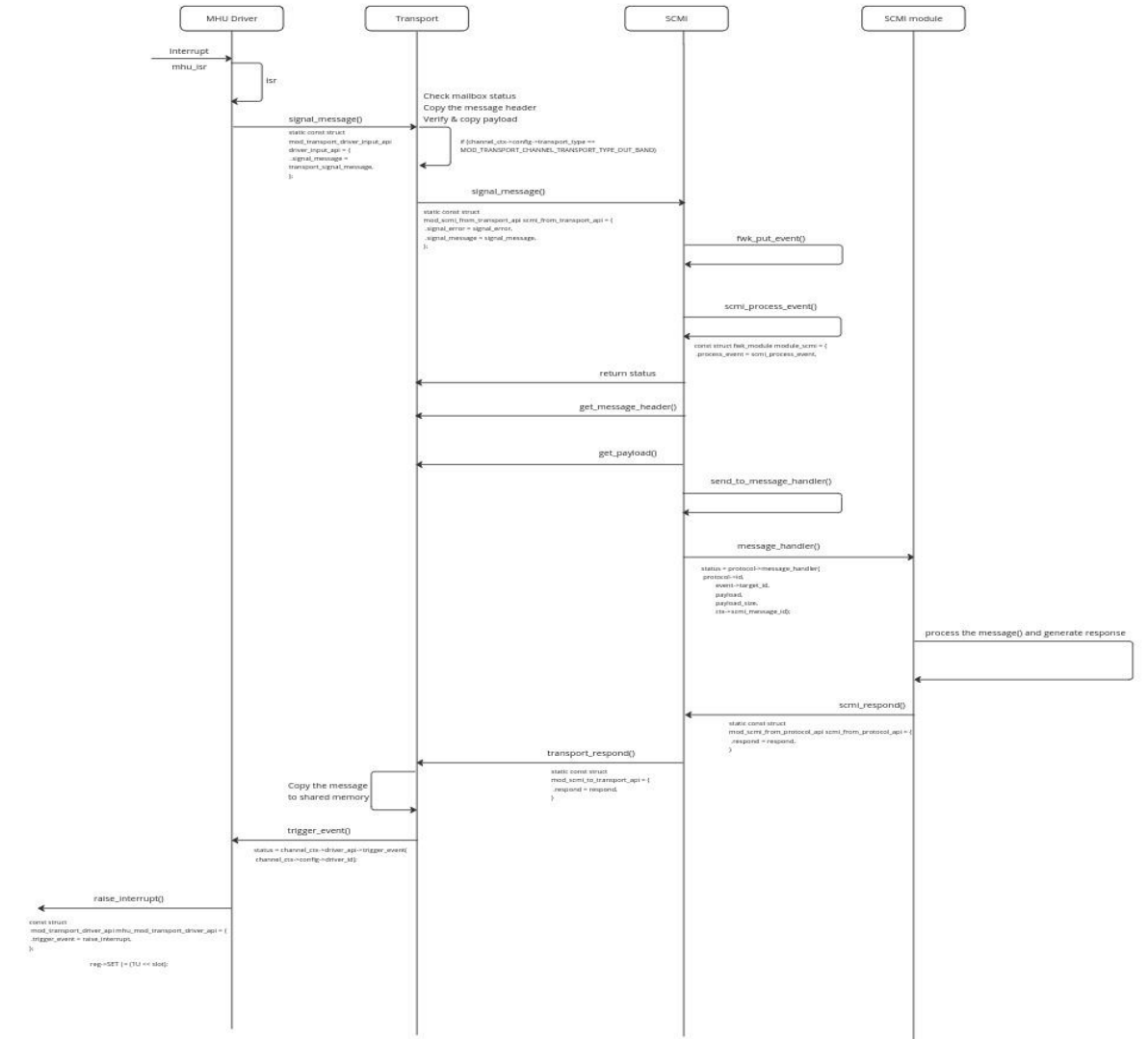
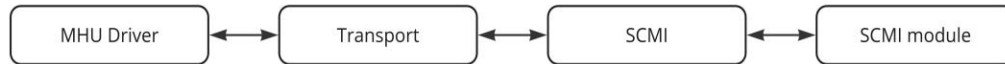
Elements and Sub-elements

- Element is a resource that is owned and governed by a module.
- An abstraction to refer to a device, a protocol or a service instance.
 - E.g., elements of a driver type module may represent each hardware device instance it controls.
- Elements are optional.
- Element description.
 - One per element.
 - Contains element configuration data.
- Elements are defined by
 - A structure containing a pointer to a name string
 - The number of sub-elements associated with the element
 - A void pointer to data that is in a module-defined format
- Sub-Elements
 - Resources owned and governed by an element
 - These do not have descriptors



- 
- Modules communicate themselves using events and notifications.
 - There are mainly two queues that are maintained
 - Event queue
 - ISR queue
 - Until version 2.9 there used to be multi thread support in SCP firmware. But from 2.10 version multi-threading was completely removed.
 - Every event must be put in the single event/ISR queue with `put_event()`.

A typical SCMI transaction





Firmware execution stages

- Few questions
- How does the framework know which modules are to be called ?
- How does the module know it's configurations for a given product ?
- How a module binds with another module ?
- How the events and data are passed between modules for a complete transaction ?

Firmware execution stages

- There are 5 pre-runtime stages in fixed order.
- All the stages are called by framework.
- Module initialization : For module configuration data
- Element initialization : For element configuration data. Only valid if module has elements.
- *Post initialization : Done if needed additional initialization.
- ***Bind** : Binding between modules and elements.
- *Start : Modules can use resources of other modules to complete initialization.

Runtime Stage

Normal execution flow directed primarily by interaction between modules.

Event, notifications and responses generated and processed.

Module structure

```
struct fwk_module {
    enum fwk_module_type type;
    unsigned int api_count;
    unsigned int event_count;

#ifdef BUILD_HAS_NOTIFICATION
    unsigned int notification_count;
#endif
    struct fwk_io_adapter adapter;

    int (*init)(fwk_id_t module_id, unsigned int element_count,
               const void *data);
    int (*element_init)(fwk_id_t element_id, unsigned int sub_element_count,
                       const void *data);
    int (*post_init)(fwk_id_t module_id);
    int (*bind)(fwk_id_t id, unsigned int round);
    int (*start)(fwk_id_t id);
    int (*stop)(fwk_id_t id);
    int (*process_bind_request)(fwk_id_t source_id, fwk_id_t target_id,
                               fwk_id_t api_id, const void **api);
    int (*process_event)(const struct fwk_event *event,
                        struct fwk_event *resp_event);
    int (*process_notification)(const struct fwk_event *event,
                               struct fwk_event *resp_event);
};
```

SCP-firmware-master_copy

- > .github
- > .gitlab
- > .vscode
- > arch
- > build
- > cmake
- > CMakeFiles
- > contrib
- > debugger
- > doc
- > docker
- > framework
- > html
- > interface
- > **module**
- > product
- > tools
- > unit_test
- ⚙️ .armclang.cppcheck.cfg
- ⚙️ .armv6m.cppcheck.cfg
- ⚙️ .armv7m.cppcheck.cfg
- ⚙️ .armv8m.cppcheck.cfg

- > amu_mmap
- > amu_smc_drv
- > apcontext
- > armv7m_mpu
- > armv8m_mpu
- > atu
- > bootloader
- > cdns_i2c

- > mhu
- > mhu2
- > mhu3
- > mock_clock

- > **scmi**
- > scmi_apcore
- > scmi_clock
- > scmi_perf
- > scmi_power_capping
- > scmi_power_domain
- > scmi_reset_domain
- > scmi_sensor
- > scmi_sensor_req
- > scmi_system_power
- > scmi_system_power_req
- > scmi_voltage_domain

- > transport
- > ut
- > voltage_domain
- > xr77128

./framework/CMakeLists.txt

```
string(APPEND SCP_MODULE_EXTERN_GEN
"extern const struct fwk_module module_${SCP_MODULE};
\n")
```

./module/name/src/mod_name.c

```
const struct fwk_module module_mhu = {
const struct fwk_module module_scmi = {
const struct fwk_module module_scmi_clock = {
```

./module/scmi/src/mod_scmi.c

```
/* SCMI module definition */
const struct fwk_module module_scmi = {
    .api_count = (unsigned int)MOD_SCMI_API_IDX_COUNT,
    .event_count = 1,
#ifdef BUILD_HAS_NOTIFICATION
    .notification_count = (unsigned int)MOD_SCMI_NOTIFICATION_IDX_COUNT,
#endif
    .type = FWK_MODULE_TYPE_SERVICE,
    .init = scmi_init,
    .element_init = scmi_service_init,
    .bind = scmi_bind,
    .start = scmi_start,
    .process_bind_request = scmi_process_bind_request,
    .process_event = scmi_process_event,
#ifdef BUILD_HAS_NOTIFICATION
    .process_notification = scmi_process_notification,
#endif
};
```



Code walkthrough

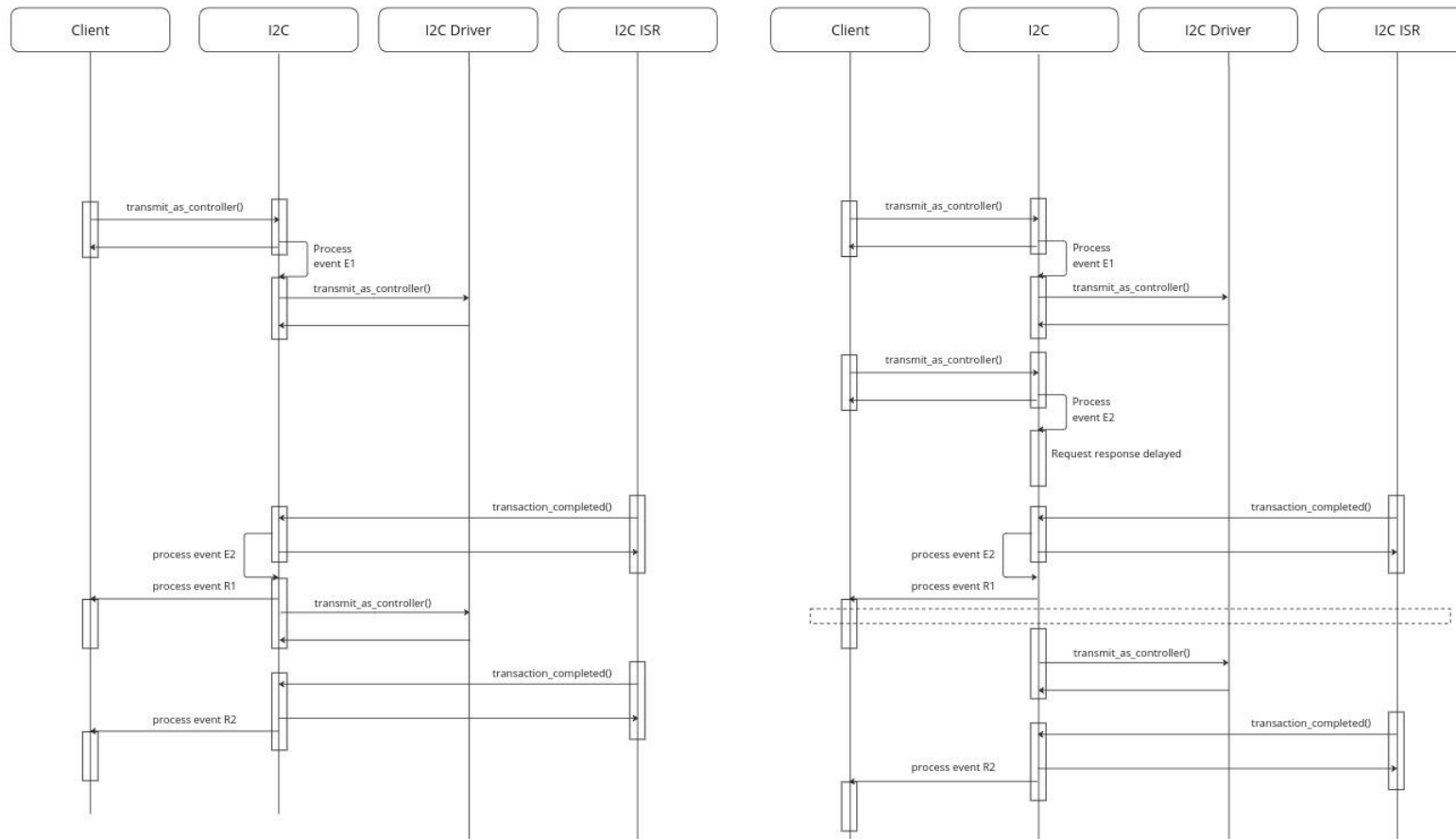


Modules design in SCP firmware

- No hard and fast rules while developing modules in SCP.
- SCP firmware allows the flexibility to design and implement the modules, elements and sub-elements as per requirement and flexibility that you want to provide.
- For example, the I2C data communication between IP's is different in "Juno" and in "Morello".
- We only need proper "binding" across modules.

I2C in SCP firmware

- From OS to SCP, SCMI is used for communication.
- Inside SoC, platform resources communicate using various protocols. I2C is one of them.





I2C implementation in SCP firmware

Juno

- I2c.c contains the functional view of i2c transaction (higher level).
- Dw_apb_i2c.c contains the actual register level transaction.
- Transaction parameters are written in dw_apb_i2c.h and are hardcoded.
- Address of the device is written in config file.
- Provides more abstraction, clients call becomes easy and effective.
- Effective delegation is required.

Morello

Entire transaction is handled by cdns_i2c.c

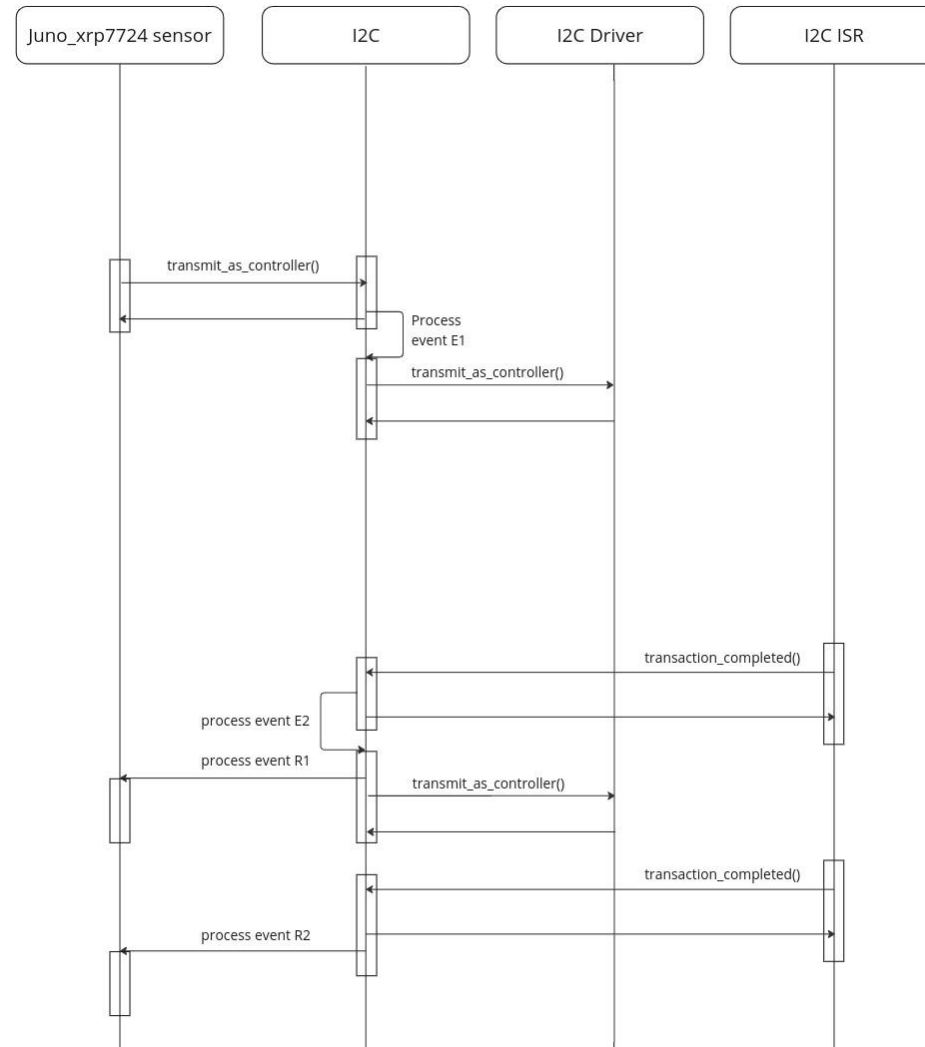
No separation of functional and register view.

All transaction parameters including address are written in the config_cdns_i2c.c file (customized).

Very straightforward, thus implementation becomes easy.

There is little to no abstraction.

I2C implementation in JUNO





Thank you