



# Guide to incorporating CHERI at the SoC Level

CHERI SoC Working Group

Version 1.0  
Date 14/04/2025

The purpose of this document is to guide system-on-chip designers to find out how to integrate hardware with varying degrees of CHERI support. It can also be used as a base to help those who design data interfaces on or off chip, with high-level requirements and ideas to help their standardization effort.



This work © 2025 by [CHERI Alliance](#) is licensed under CC BY-SA 4.0  
(Creative Commons Attribution-ShareAlike 4.0 International) –  
<https://creativecommons.org/licenses/by-sa/4.0/>

## Content

---

|  |    |
|--|----|
| Content .....  | 2  |
| What is the impact of CHERI at the system level? .....   | 2  |
| Composition of CHERI hardware .....  | 3  |
| CHERI CPU using CHERI-aware internal memory.....   | 3  |
| CHERI CPU using CHERI-aware external memory.....   | 5  |
| CHERI CPU and Accelerator using CHERI-aware external memory .....                                      | 7  |
| All IPs support the same CHERI features, and use the same capability format.....                       | 9  |
| All IPs support the same CHERI features, but use differing capability formats.....                     | 9  |
| IPs in the system support different sets of CHERI features, and use differing capability formats. .... | 10 |
| CHERI CPU and Accelerator using CHERI-aware external memory, alongside a CHERI Root-of-Trust.....      | 11 |
| Levels of CHERI support .....  | 12 |
| Levels of CHERI integration .....  | 12 |
| Levels of CHERI Support vs Integration .....   | 14 |

## What is the impact of CHERI at the system level?

---

CHERI is a way of providing hardware-based memory safety. In a CHERI system, pointers are replaced by capabilities. A capability not only provides the memory address being pointed at, but also a set of permissions and bounds that the hardware must operate within. The permissions say what the hardware accessing the memory address is allowed to do (for example, read data from the memory location, execute code held at the memory location, etc.), whilst the bounds specify the window of memory that can be accessed using this specific capability.

The use of capabilities means that unauthorized access to memory (and use of the data found there) can be avoided, but the system only works if capabilities are unforgeable. Therefore, it is not enough to just define the format of a capability. There must also exist a mechanism to confirm the validity of that capability. This is done using a tagged memory system.

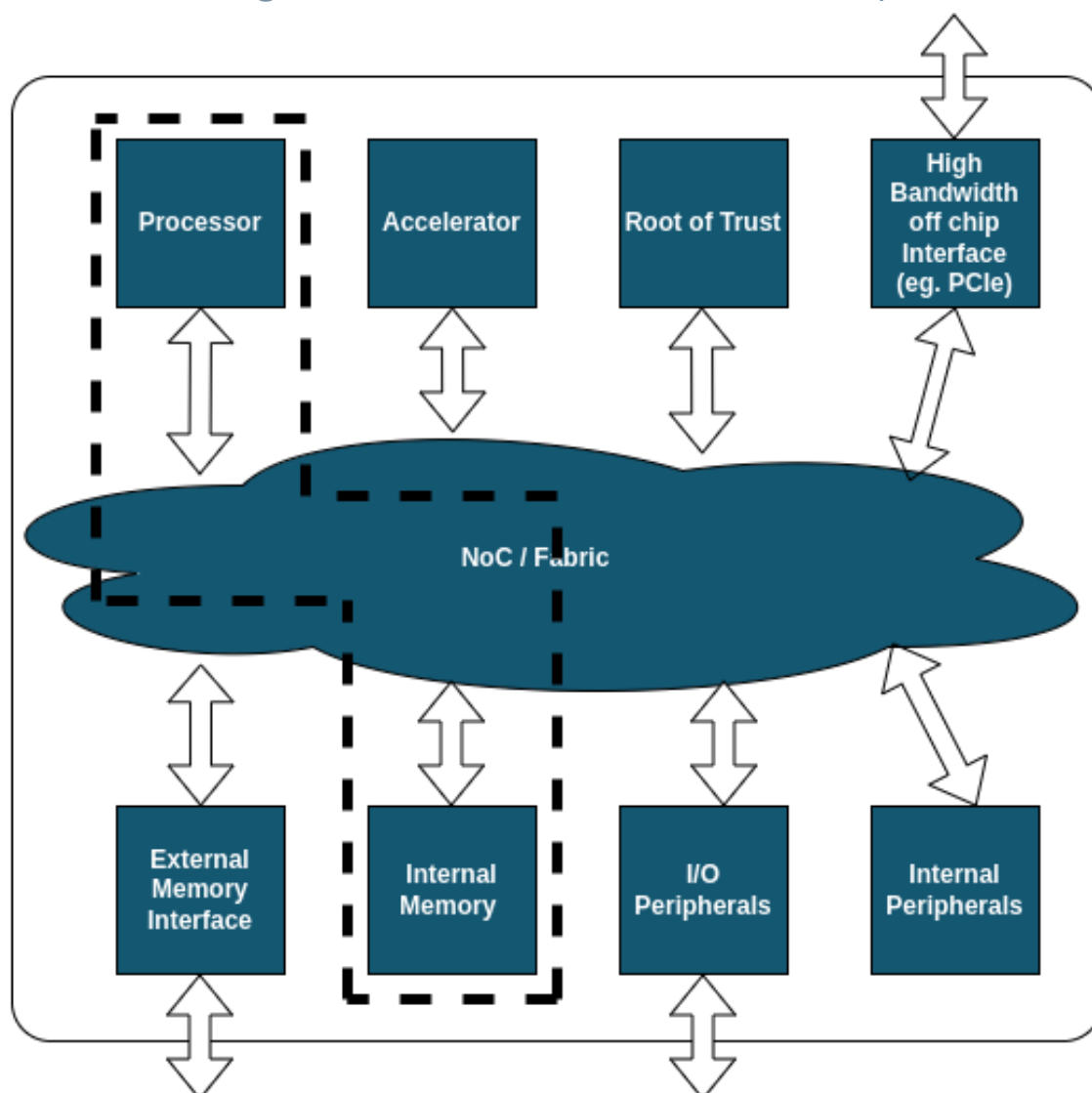
The exact mechanics of a CHERI system may vary somewhat (for example, with the underlying machine architecture), but the basic concepts will remain the same. The fundamental principles are:

- Capabilities must be stored in an aligned fashion (based on the width of the capability).
- Each aligned capability must have a tag bit associated with it.
- Tag bits exist outside of the normal addressable space of the system.
- A tag bit being set indicates that its associated data is a valid capability.
- A tag bit being clear shows that its associated data is not a valid capability.
- Tag bits and their associated data must be treated atomically at all times.
- Only a CHERI-enabled bus manager can set a tag bit.
- Any partial write to a capability must result in the associated tag bit being cleared.
- Any write to a capability from a non-CHERI-enabled bus manager must clear the tag bit.

## Composition of CHERI hardware

Let us consider some example systems including some IPs with different levels of CHERI support and integration.

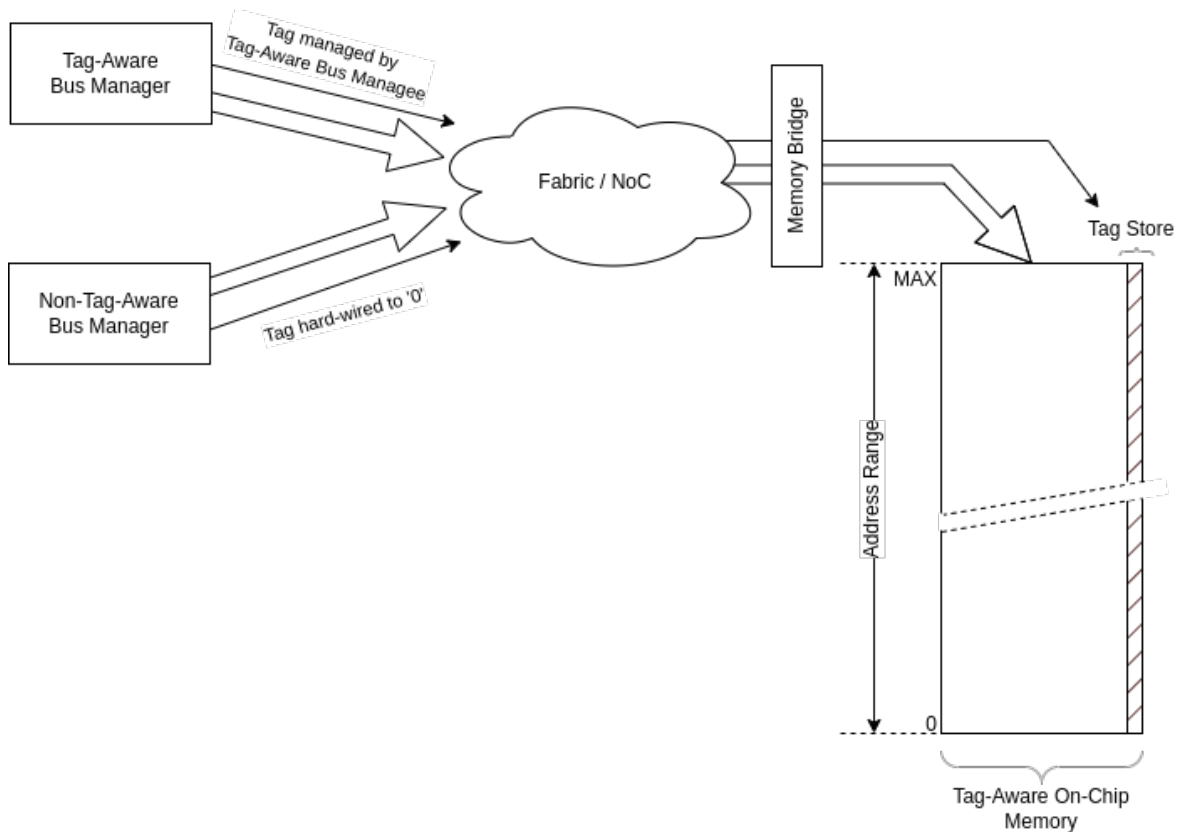
### CHERI CPU using CHERI-aware internal memory



In this system the CPU is the only functional unit within the system to feature any CHERI support, and can store capabilities and tags directly to an internal RAM. Capabilities can therefore not be shared between the CPU and any other functional units.

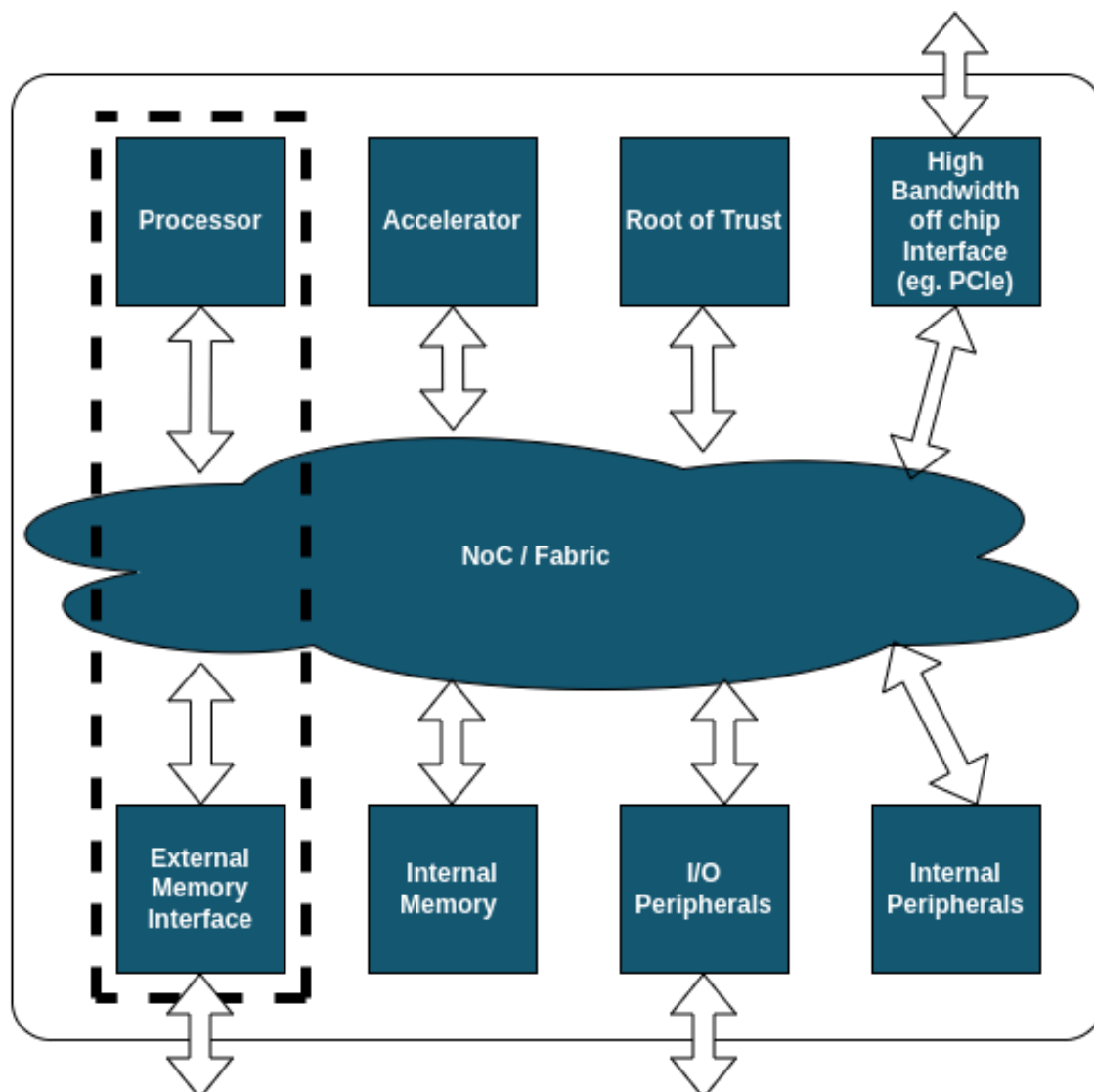
Capabilities written to the RAM will need to have their tags read and written atomically alongside the associated data. If a non-CHERI-aware bus manager writes to a location in the internal RAM the associated tag bit must be cleared to ensure that capabilities cannot be forged.

A possible hardware structure to achieve this is shown below.



Here data is read and written to the RAM from the CPU (or other bus manager) via some fabric / network on chip and a RAM bridge (to bridge between the relevant bus protocol and the RAM itself). Tag bits (and other required meta data) are carried on out-of-band signalling. The RAM is wide enough to store the tag bits alongside the associated data. The RAM bridge is responsible for managing the storage of tag bits (ensuring that if a non-CHERI-aware bus manager writes to some location within the RAM the associated tag bit will be cleared in all situations).

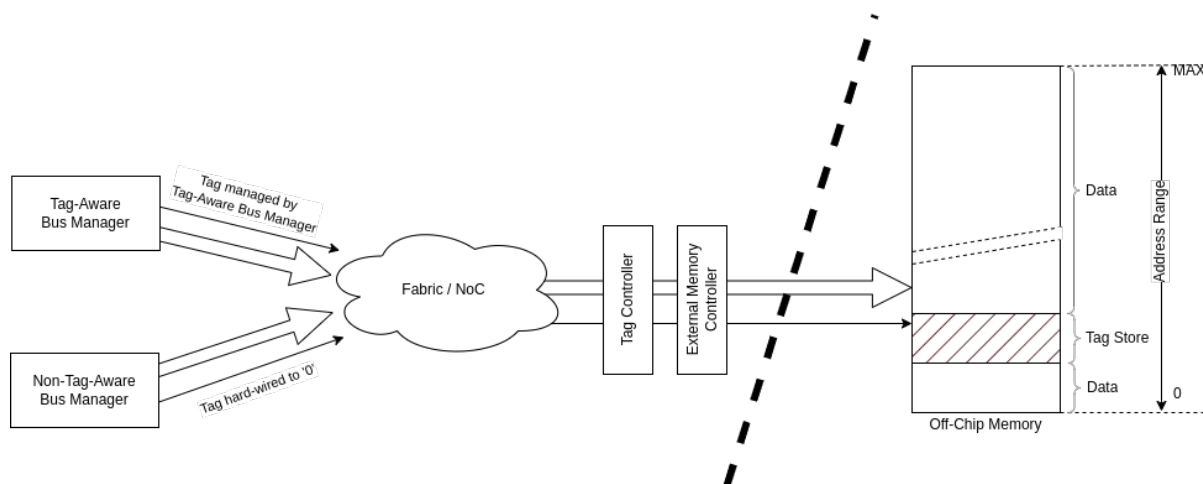
## CHERI CPU using CHERI-aware external memory



In this system, the CPU is once again the only functional unit within the system to feature any CHERI support, and stores capabilities and tags to an external memory. Capabilities cannot be shared between the CPU and any other functional units.

Capabilities written to the external memory will need to have their tags stored atomically alongside the associated capabilities. If a non tag aware bus manager writes to a location in the external RAM the associated tag bit should be cleared to ensure that capabilities cannot be forged.

A possible hardware structure to achieve this is shown below.

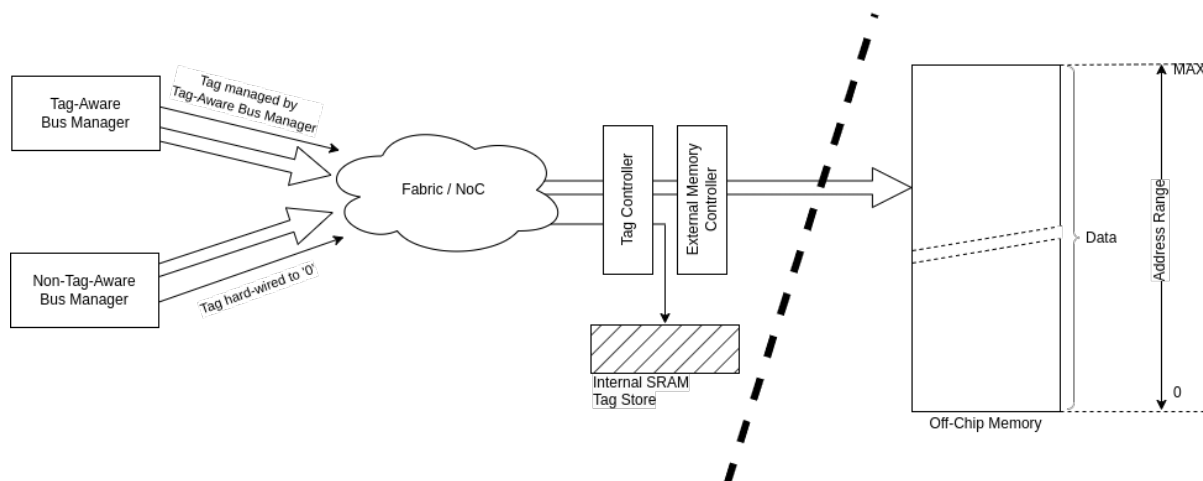


Here there exists a tag controller that is able to receive bus transactions from the Fabric / network on chip (including tags and any other required metadata passed in out-of-band signalling) and then store the data and tags atomically in the external memory. This would likely be via some memory controller.

External memory parts typically do not support any provision for storing tag bits. Therefore some other strategy is required. In this case, a region of the memory is defined as a tag store, and all tag bits are stored in this region, separately from their associated data.

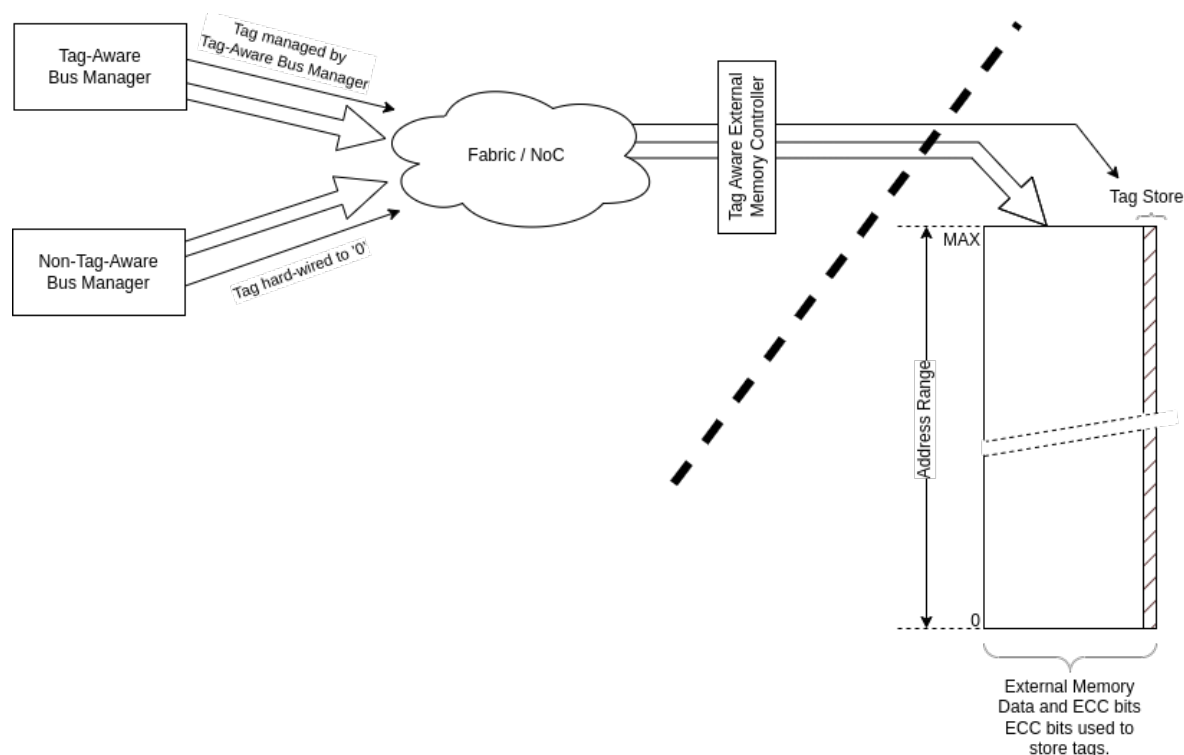
Therefore, the Tag Controller hardware must ensure that the atomicity of tag and data access is preserved, as well as ensuring that tags are cleared if written by a non-tag aware bus manager.

Of course, it is also possible to have an external memory to hold data, and an internal memory used to store the associated tag bits. A possible hardware structure that could support this approach is shown below.



In this case the total amount of tag aware external memory space is limited by the size of the internal RAM used to store the tags. The tag controller remains responsible for maintaining the atomicity between the internal tag store and the external data store.

Another possible hardware structure that could be imagined is shown below.



Here an external memory part has been selected that supports additional bits alongside the data - these bits are typically used to store ECC bits, for example. In this example the additional bits are not used to store ECC bits, but rather to store the tag bits themselves. This approach requires the use of a memory controller that:

- is tag aware,
- receives tag bits and other required metadata via out-of-band signalling,
- is able to enforce the atomicity of the tag bits and the associated data,
- can utilize the ECC bits of the external memory for tag storage without enforcing ECC functionality.

## CHERI CPU and Accelerator using CHERI-aware external memory

The more interesting behaviour comes when you try to compose a complex SoC from multiple hardware IP blocks with possibly varying levels of CHERI support, and the intent that it should be possible to pass capabilities, tags and revocation tags between the different CHERI-enabled IPs.

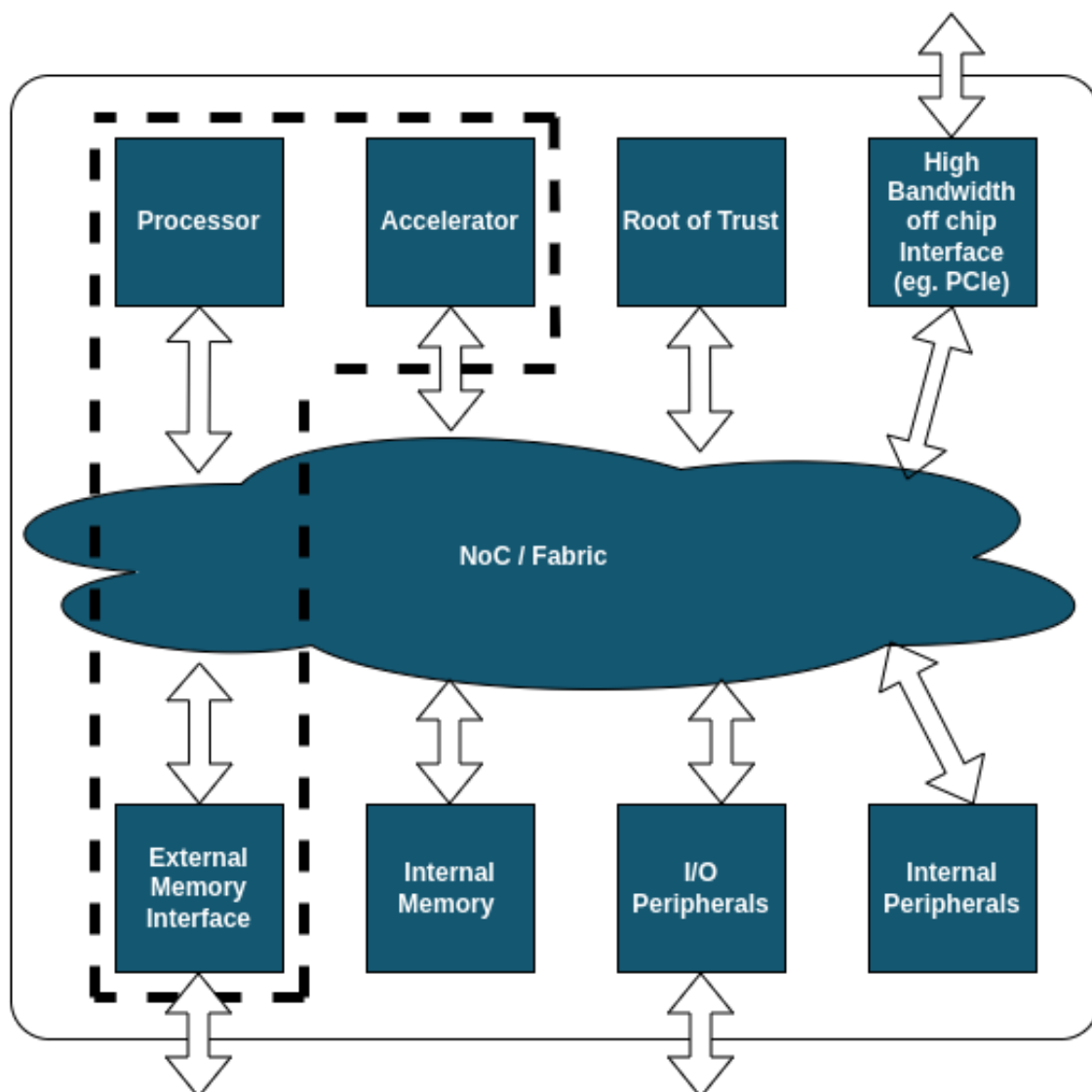
In a system where multiple CHERI-enabled IPs should work together, we develop the concept of a CHERI-domain. Within a CHERI-domain all CHERI-enabled IPs are able to share capabilities, tags and revocation tags.

Interfaces between CHERI-enabled IPs within a CHERI-domain, or between CHERI-domains, will likely require more consideration to support the more complex interactions between the IPs (such as sharing and revoking capabilities between different IPs). There may also be non-CHERI-domains within a system.

Depending on which choices you make in your SoC design, we can classify interfaces as having different properties.

- **None:** In this case, there is no integration to be done with the wider system, capability and revocation tags are both cleared when going off the interface, and the interface can operate as usual.
- **Raw:** Raw just means that all IP blocks use the same capability format and understand tags in the same way. This means that the capability tags and meta-data can be transported raw over the interface.
- **Translation:** Translation is the case where different capability formats exist within the same system or between systems. An example use-case is when a system that uses compressed capabilities talks to a system that uses uncompressed capabilities. Different capability formats are quite likely when CPUs and GPUs start interacting. The translation layer must be able to translate between these two domains and deal with cases when capabilities may be representable in one domain but not in the other.

In the example system below, the CPU is no longer the only functional unit within the system to feature some level of CHERI support. There is also some kind of Accelerator that is also CHERI-enabled. Both of these IPs use an external memory to hold their capabilities and associated tags, and should be able to share capabilities as desired.



There are four possibilities that exist in respect of the compatibility of the CHERI-enabled IPs within the system, and the resulting impact on the integration of those IPs into the system.

1. All IPs support the same CHERI features, and use the same capability format.



2. All IPs support the same CHERI features, but use differing capability formats.
3. IPs in the system support different sets of CHERI features, but use the same capability format.
4. IPs in the system support different sets of CHERI features, and use differing capability formats.

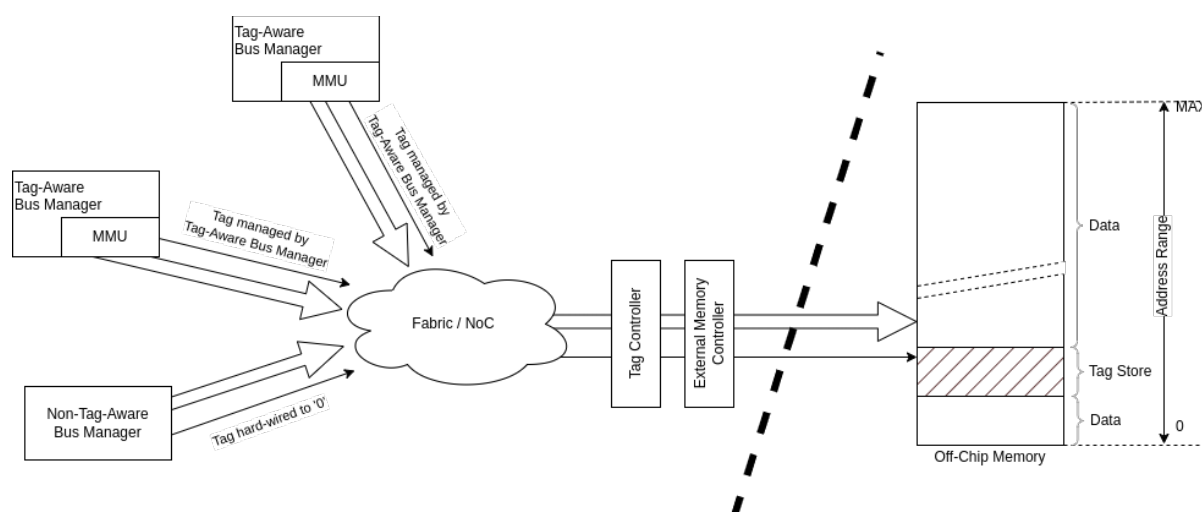
These four cases are considered in more detail below.

## All IPs support the same CHERI features, and use the same capability format.

It is possible that each of the CHERI-enabled IPs share the same capability format, and support the same set of CHERI features. This is the most straightforward situation (from a systems point-of-view). In this case, capabilities are able to flow between the CHERI-enabled IPs (via memory) with no technical difficulty or extra complexity.

Note that there may be situations where a capability should not be shared between the CHERI-enabled IPs (in line with the security policy of the system). Such limitations can be imposed using standard memory management techniques, which can co-exist alongside CHERI.

A possible hardware structure that could be used to implement such a system is shown below.



As system complexity, number of CHERI-enabled IPs and the number of IP suppliers in a given system all increase, so too does the likelihood of having multiple differing capability formats and IPs supporting differing combinations of CHERI features.

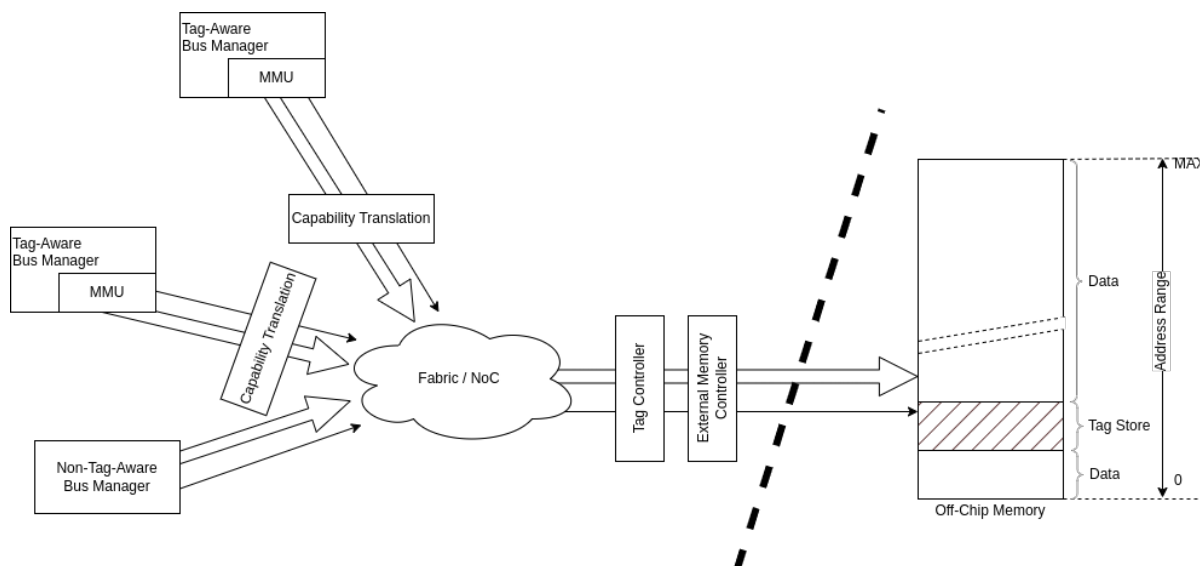
## All IPs support the same CHERI features, but use differing capability formats.

Let us consider a system in which the various CHERI-enabled IPs all support the same CHERI features, but use differing capability formats to represent them. (Note that while this scenario is possible, it is unlikely to be commonplace).

If such a system was constructed, then some translation would be required in order to allow the IPs to share their capabilities. As the IPs support the same CHERI features, it is possible to select a common capability format that can fully represent all capabilities within the system. Capabilities are all stored in memory in this common format.

When an IP in the system writes a capability out to main memory, it will be translated to the common capability format before it reaches the memory. When an IP reads a capability from main memory, it will be translated back to the IPs specific capability format.

A possible hardware structure that could be used to implement such a system is shown below.



IPs in the system support different sets of CHERI features, but use the same capability format.

Now let's consider a system in which the various CHERI-enabled IPs support differing sets of CHERI features, but use the same capability format to represent them. As the IPs do not all support the same CHERI feature set, this implies that (for at least some IPs) there are aspects of the capability format that are not supported. It may be possible for one IP to create a capability that another IP cannot fully support. In this case, it is important to have a clear and defined behaviour so that the system does not operate in an unpredictable, unexpected or insecure way.

For example, consider a crypto engine. Capabilities could be used to provide memory safety for data being transferred into or out of the crypto engine. From the crypto engine's perspective it has no concept or ability to execute input data as code on its internal CPU, therefore it can effectively ignore any associated permission bits. However, if the crypto-engine is creating a derived capability, all permission bits (even those that hold no relevance to the crypto-engine) must be preserved.

## IPs in the system support different sets of CHERI features, and use differing capability formats.

Lastly, let's consider a system in which the various CHERI-enabled IPs support different sets of CHERI features, and also use differing capability formats to represent them. As discussed previously, this implies a requirement for translation between the various capability formats, as well as suitable policies being put in place to bridge between the varying levels of functionality offered by each of the IPs, and gaps in what can be represented in each of the differing capability formats.

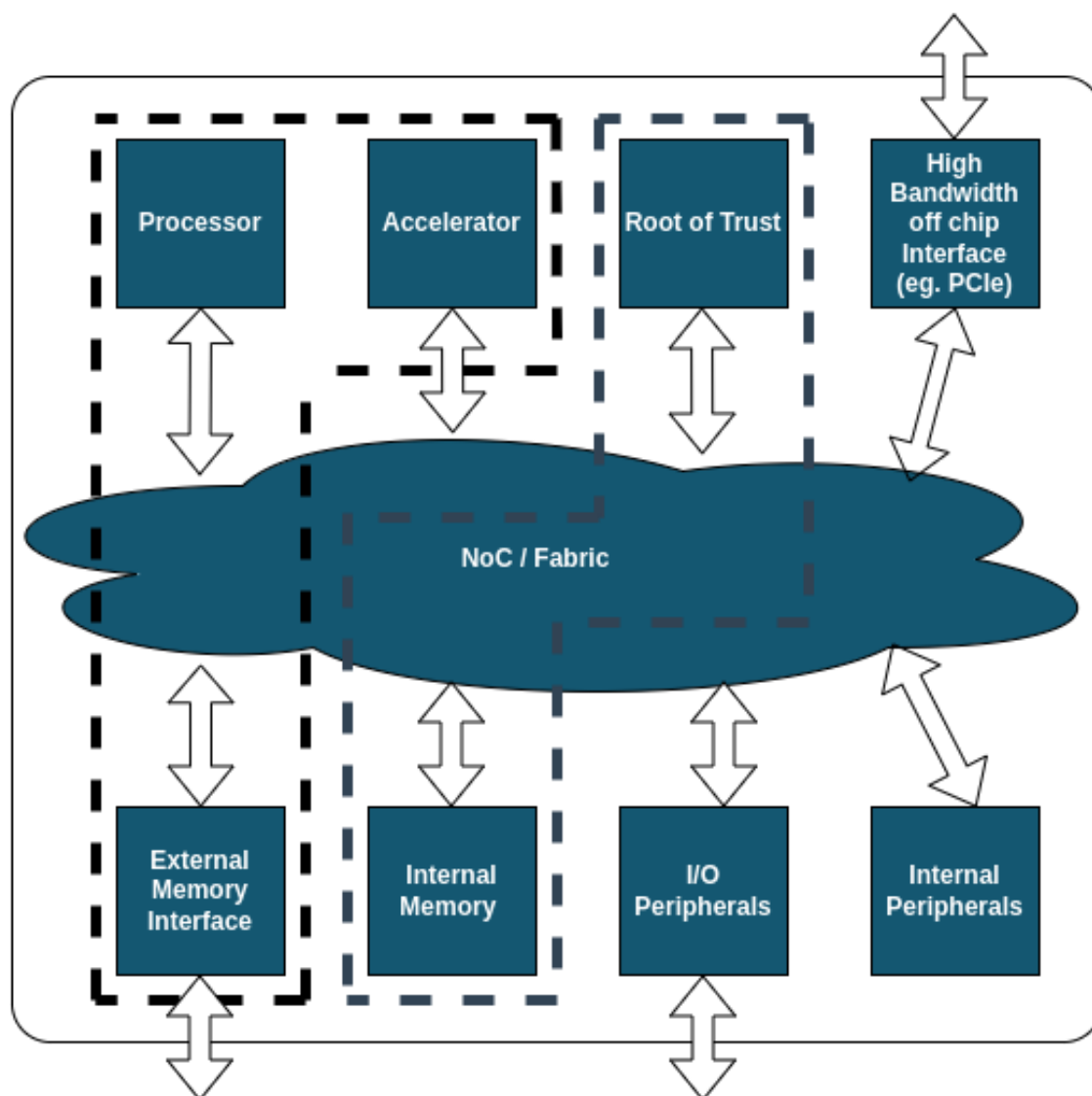
Such policies can be implemented in many ways, however it is vital that capabilities are never relaxed.

Translation units could be used (as in one of the previous examples). However, in this case, there may be situations where a capability cannot be represented in the target capability format. In these cases, one option is to clear the tag bit, so that the capability is not usable after translation. This is perhaps the most secure approach.

As CHERI technology becomes more mainstream and more IP suppliers integrate CHERI into their portfolios, this situation is likely to become very common when developing an SoC of reasonable complexity.

## CHERI CPU and Accelerator using CHERI-aware external memory, alongside a CHERI Root-of-Trust

It is also possible to have several CHERI regions within a system existing independently.



In this example we see a CHERI-enabled CPU and CHERI-enabled Accelerator forming a single CHERI region which includes some external memory. Alongside this region exists a second region, which is made up of a Root of Trust and some internal memory.

As these two regions do not interact (from a CHERI perspective) there is no requirement to consider any interoperability with respect to capability formats or CHERI feature sets.

## Levels of CHERI support

---

It is useful to be able to broadly specify what level of CHERI support a given IP offers. This will help facilitate conversations and comparisons between different IPs, and also help clear communication of what guarantees a specific IP can offer.

Therefore 4 levels have been specified and are listed below (in order from most sophisticated support down to no support).

Each level builds on the features of the lower levels.



- **Level 3:** Spatial and temporal safety (with revocation accelerated via hardware to allow immediate revocation). Use-after-free deterministically trapped.



- **Level 2:** Concurrent Temporal Safety. Protecting against use-after-reuse can be done without a stop-the-world phase. Software is assisted or accelerated by specific hardware. Revocation happens concurrently with normal execution. Revocation is not immediate, use after free is *not* guaranteed to trap, but an allocator can prevent use after reuse.



- **Level 1:** Basic spatial safety support. Spatial memory safety with bounds and permissions is supported by the IP. Temporal safety is possible only with a stop-the-world phase.



- **Level 0:** No CHERI support (i.e. legacy hardware). This means all memory operations are done in the regular way. The compiler may still make some memory-safety guarantees but these are not visible nor enforced by the hardware.

## Levels of CHERI integration

---

It is also useful to be able to broadly specify what level of CHERI integration exists within a system (e.g. product, network, PCB, SoC, chiplet, sub-system, CHERI region, or group of IPs).

This will help facilitate conversations and comparisons between different systems, and also help clear communication of what guarantees a specific system can offer.

Each level builds on the features of the lower levels.

 Level A

- **Level A - Immediate revocation integration:** In addition to Level B: Revocation is possible across the IP-block barrier. Immediate revocation is possible at the hardware level. Deterministic use-after-free protection.

 Level B

- **Level B - Hardware accelerated revocation integration:** In addition to Level C: Software managed revocation is possible, with some hardware acceleration. Concurrent revocation is possible without a stop-the-world phase on any IPs

 Level C

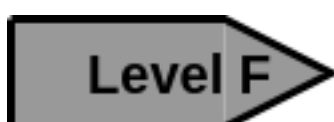
- **Level C - Software revocation integration:** In addition to Level D: Spatial and temporal memory safety is available between multiple IPs. Capabilities can be shared between various IPs. Capability tags are visible across the IP-block barrier. Software managed revocation is possible, with no specific hardware support. Likely to need to stop-the-world (at least for some IPs) to perform revocation.

 Level D

- **Level D - Local integration:** In addition to Level E: Spatial and/or temporal memory safety is available for isolated IPs. Capabilities cannot be shared between any IPs within the system. The system's main CPU and accelerators (where the end user will be running their application code) will support spatial and/or temporal memory safety.

 Level E

- **Level E - No integration:** The system's main CPU and accelerators (where the end user will be running their application code) does not support spatial or temporal memory safety. Some firmware in the system is running on CHERI enabled hardware.

























 Level F

- **Level F - CHERI not supported:** No code running on CHERI enabled hardware.

## Levels of CHERI Support vs Integration

Not all combinations of levels of support and integration are sensible. For example, you could not build a level A system (a system which supports the most sophisticated CHERI features and offers the highest guarantees of memory safety) created from IPs with only level 0 support (i.e. the IPs do not support any CHERI features, and are not CHERI-aware in any way).

The grid below shows which combinations of support and integration level are feasible.

|   | 0   | 1   | 2   | 3  |
|---|---|---|---|--|
| A |    |    |    |    |
| B |    |    |    |    |
| C |    |    |    |    |
| D |   |   |   |   |
| E |  |  |  |  |
| F |  |  |  |  |