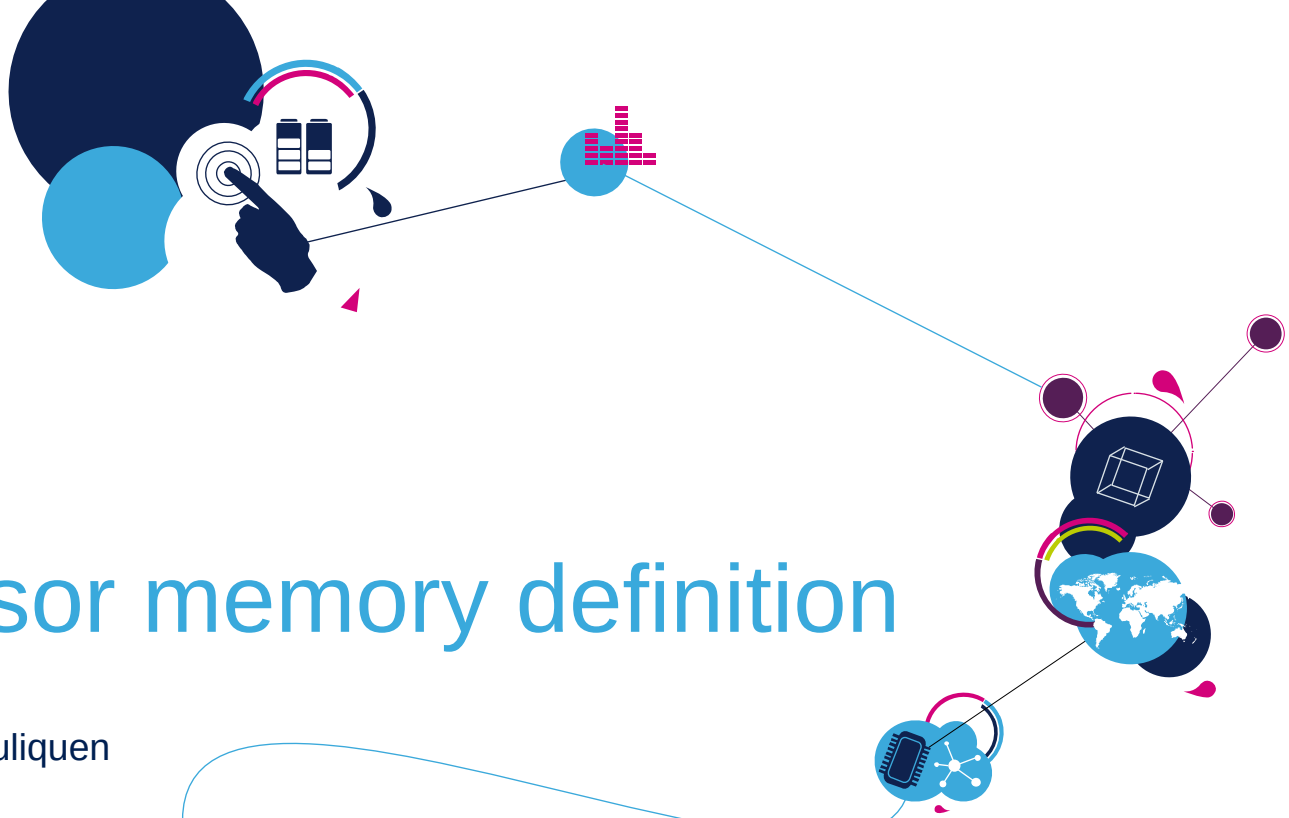


Coprocessor memory definition

Loic Pallardy / Arnaud Pouliquen

life.augmented

Objective

- The goal of following slides is to sum up on-going discussion in OpenAMP weekly about Remoteproc/Rpmsg memory allocation.
- Following proposal is based on resource table used as central element for information sharing between the two processors.
- Different use cases are taken into account (in priority order)
  - Master main OS is loading and starting slave processor. Master allocates resources and initiates communication (current mainly)
  - Slave starts before Master main OS, Master allocates resources and initiates IPC communication
  - Whatever boot order, each processor allocates its own resources and is able to initiate communication once link ready (peer to peer mode).
- For each use case
  - Memory resource can be dynamically defined at runtime
  - Memory resource can be fixed at code generation taking into account project constraints (SoC mapping, security…)

Current situation

- Remoteproc is in charge of 2 allocations
  - Carveout: a memory region dedicated to coprocessor accessible by both processors
    - Each region declared in resource table is allocated thanks to dma_alloc_coherent call and managed in a list by rproc
    - Rsc table updated by rproc for information sharing with slave coprocessor (pa only)
    - Carveout region are used mainly for firmware and log buffer
    - Region is accessible thanks to rproc_da_to_va interface
    - No fixed definition support by remoteproc framework (means if carveout resource defined in rsc table, rproc try to allocate it)
    - **BUT** hook exit at platform driver level to manage a platform local carveout list
  - Vrings: buffers used to establish communication at virtio level
    - The two vrings are dynamically allocated thanks to dma_alloc_coherent according to rsc table definition (nb element, alignment…)
    - Rsc table is updated by rproc for information sharing with slave coprocessor
    - No fixed vring definition supported
    - Nb of vring per vdev fixed to 2.
    - Nb of vdev per rproc fixed to 1.

Current situation

- Rpmsg is in charge of shared buffer allocation
  - Only dynamic allocation supported based on dma_alloc_coherent
  - Relies on rproc platform device dma memory pool (grand-father)
  - Doesn't update any information in resource table as buffer link done at vring level
  - Considers that coprocessor has complete memory access (as master processor) (or that memory access has been grant/defined by another way before)

Different memory regions configuration

- Memory region: generic term to address carveout, vring and rpmsg shared buffers
- Current implementation : No constraint
  - master could perform dynamic allocation in system memory (external RAM in general).
  - Master has the responsibility to provide all needed information to slave to handle memory mapping on its side
- Use of specific internal SoC memory
  - Master could perform dynamic allocation but in one internal memory using a dedicated allocator
  - Master has the responsibility to provide all needed information to slave to handle memory mapping on its side (via rsc table)
- Use of fixed memory region
  - All or part of coprocessor memory regions are defined at product definition level to take into account SoC, security, product requirements
  - Master and slave should rely on address defined in rsc table to enable memory access
  - Master must not change pre-defined value in rsc table

Proposal overview

- To cover all memory configurations for coprocessor firmware, vrings and buffers, following items should be covered:
  - Use carveout resource in rsc table to define all memory regions shared between master and slave
  - Add support of fixed memory region in rproc carveout handler
  - Add platform driver specific memory allocation support (covered by carveout region management)
  - Add support of fixed memory region for vring allocation
  - Provide dedicated DMA memory pool to virtio_rpmsg device for buffer allocation
- Peer to peer use case to be added on the top

Carveout resource management

- On master side
  - Need to allocate requested memory region
  - Need to grant CPU access to this memory region
  - Need to map it on device memory domain if supported (sMMU)
- On slave side
  - Enable memory access on defined carveout region (MPU/MMU configuration)
- At the end of master and slave remoteproc initialization, both are able to access the different carveout regions
- All carveout resources should be processed before other resources
- Common reference between master and slave is physical address.

How to define a fixed memory region

- Physical mapping is the reference at project/SoC level
- PA field in the different resource description is a good candidate to differentiate the different type of memory (fixed or not).
- Rule could be the following one

```
if PA == FW_RSC_ADDR_ANY then
        dynamic allocation ( = current implementation)
else
        memory fixed => just map it
```

- In both case, memory region should be added in carveouts list
- A new flag is needed in struct rproc_mem_entry to identify how carveout access is managed (DMA allocation, mem_map, other ?). Maybe priv field could be used?

How to rely on specific memory allocator

- Rely on platform driver carveout registering for dedicated allocation management
- Free callback associated to each carveout to keep remoteproc core generic

Vring allocation

- Need to add fixed vring location support in vring allocation function.
- Differentiation between fixed and dynamic Vring could be done thanks to PA field (same as carveout):

```
if PA == FW_RSC_ADDR_ANY then
        dynamic allocation ( = current implementation)
else
        memory fixed => find match between fixed vring and
                        carveout list → get associated
                        VA address
```

- Need a new helper function "rproc_pa_to_va" to parse carveouts list with physical address
- Need to update rproc_free_vring function to only free vring if doesn't belong to a carveout region
  - Flag needed in struct rproc_vring?

Rproc client buffer allocation (virtio_rpmsg or virtio_console)

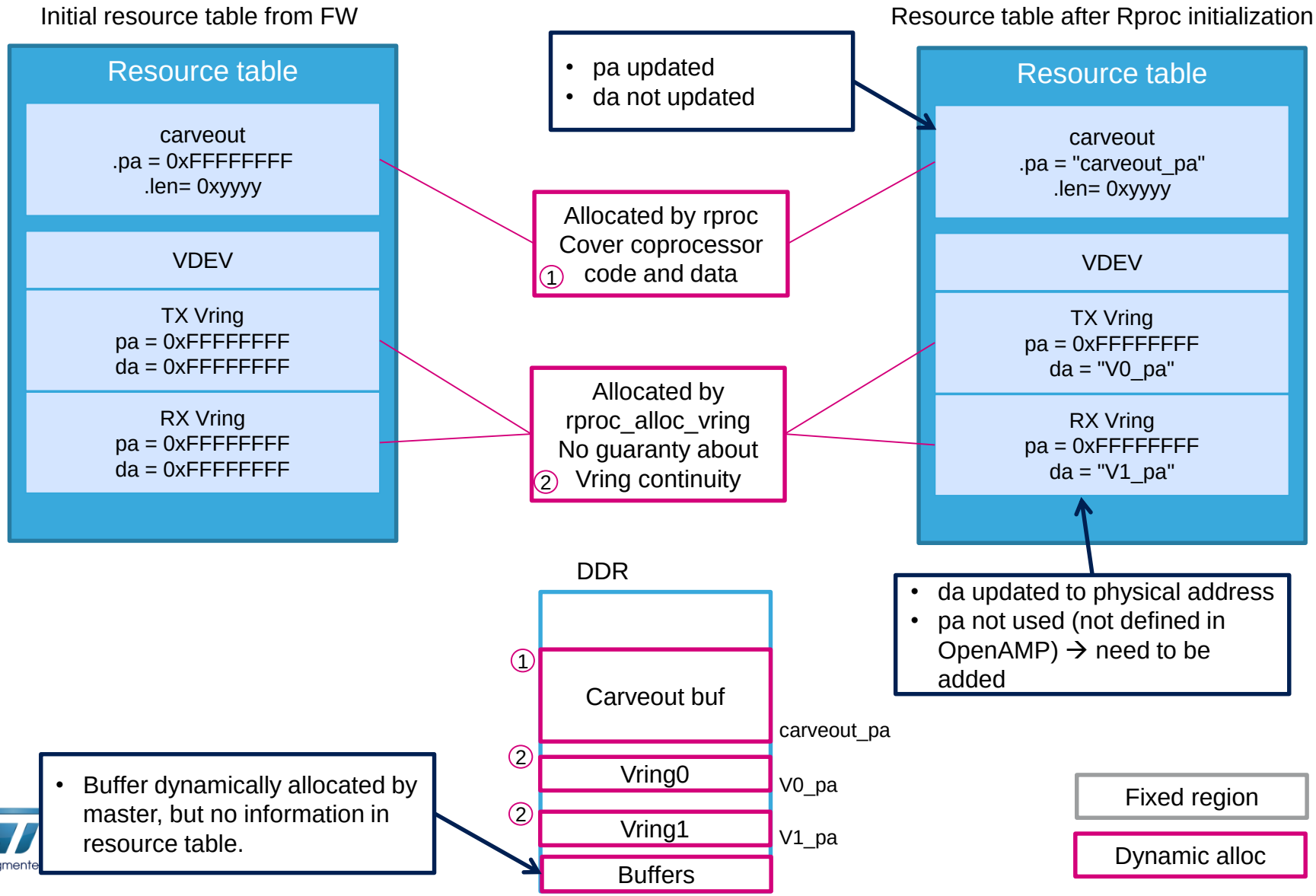
- Virtio based clients buffers are dynamically allocation using dma_alloc_coherent function
- Device used for allocation is client grand-father device, i.e. platform remoteproc driver
- No direct link between client and remoteproc. Isolation has to be preserved.
- Possibility to access cfg extension field of struct fw_rsc_vdev of the rsc table to get information (see virtio get and set ops)
  - BUT should be used for custom/optional configuration, not for memory definition which is already present in carveout resource

Rproc client buffer allocation (virtio_rpmsg or virtio_console)

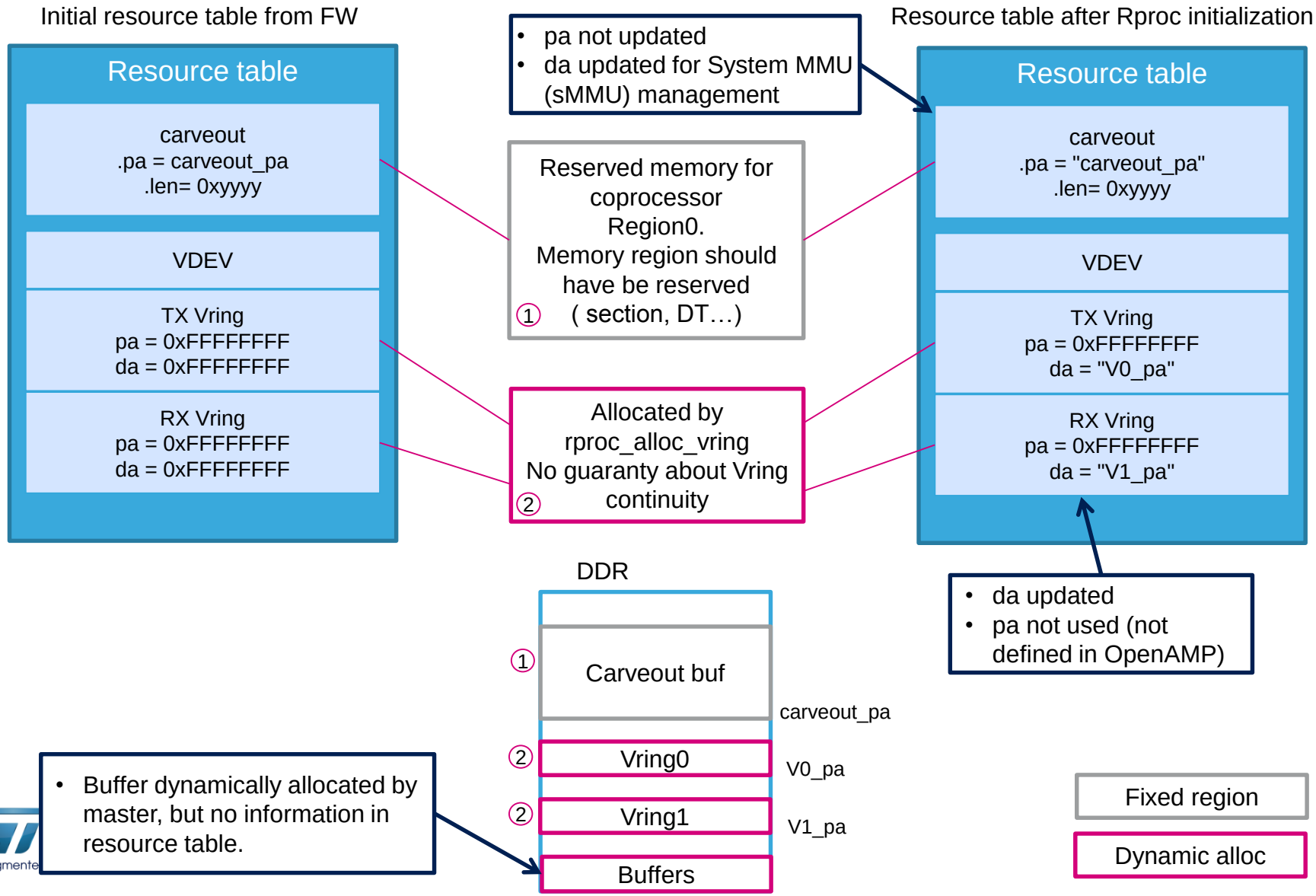
- Proposal is to associate a dedicated memory pool to virtio client for its allocations
  - Either assign DMA memory pool to virtio device at its creation and change all client to rely on father (virtio device) instead of grand-father (platform rproc device)
  - Or Introduce a sub-device per vdev at remoteproc level which will be provided as parent of virtio device. No change at virtio client level.
  - Association done thanks to dma_declare_coherent_memory function
- Master and slave will have access to buffers as access granted during carevout resources handling
  - No need to exchange more information about shared buffer address and size
- Need to define a way to get information about shared buffer location (and associated memory region)
  - Rely on carveout resource name ?
  - Specific platform driver helper function ?
  - New field in struct fw_rsc_vdev providing index of carveout resource to use?
  - …

Examples

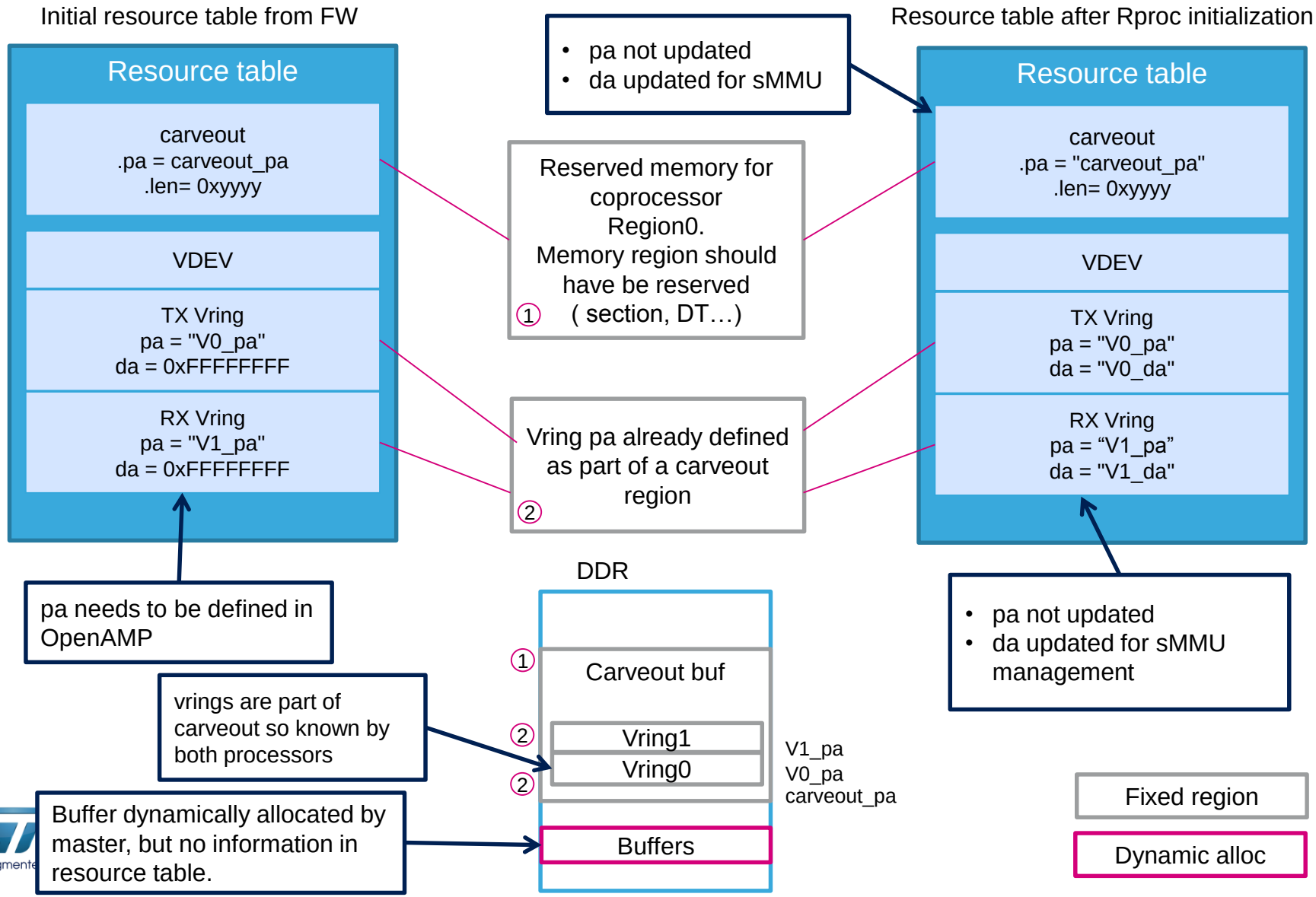
Full dynamic allocation ("current" Linux support)



Static code/data carveout definition
Dynamic vring/rpmsg buffer allocation

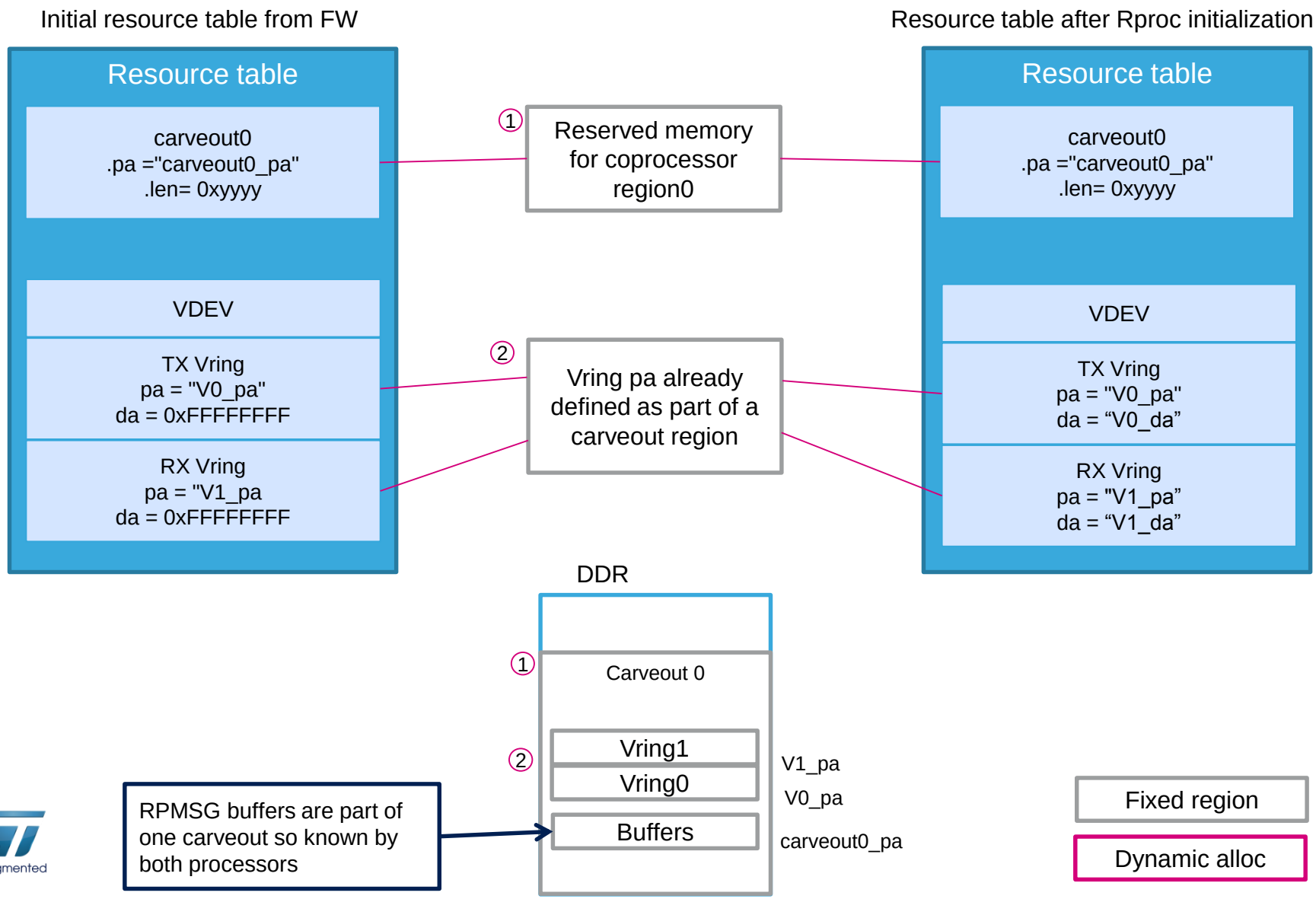


Static code/data carveout & vring definition
Dynamic rpmsg buffer allocation



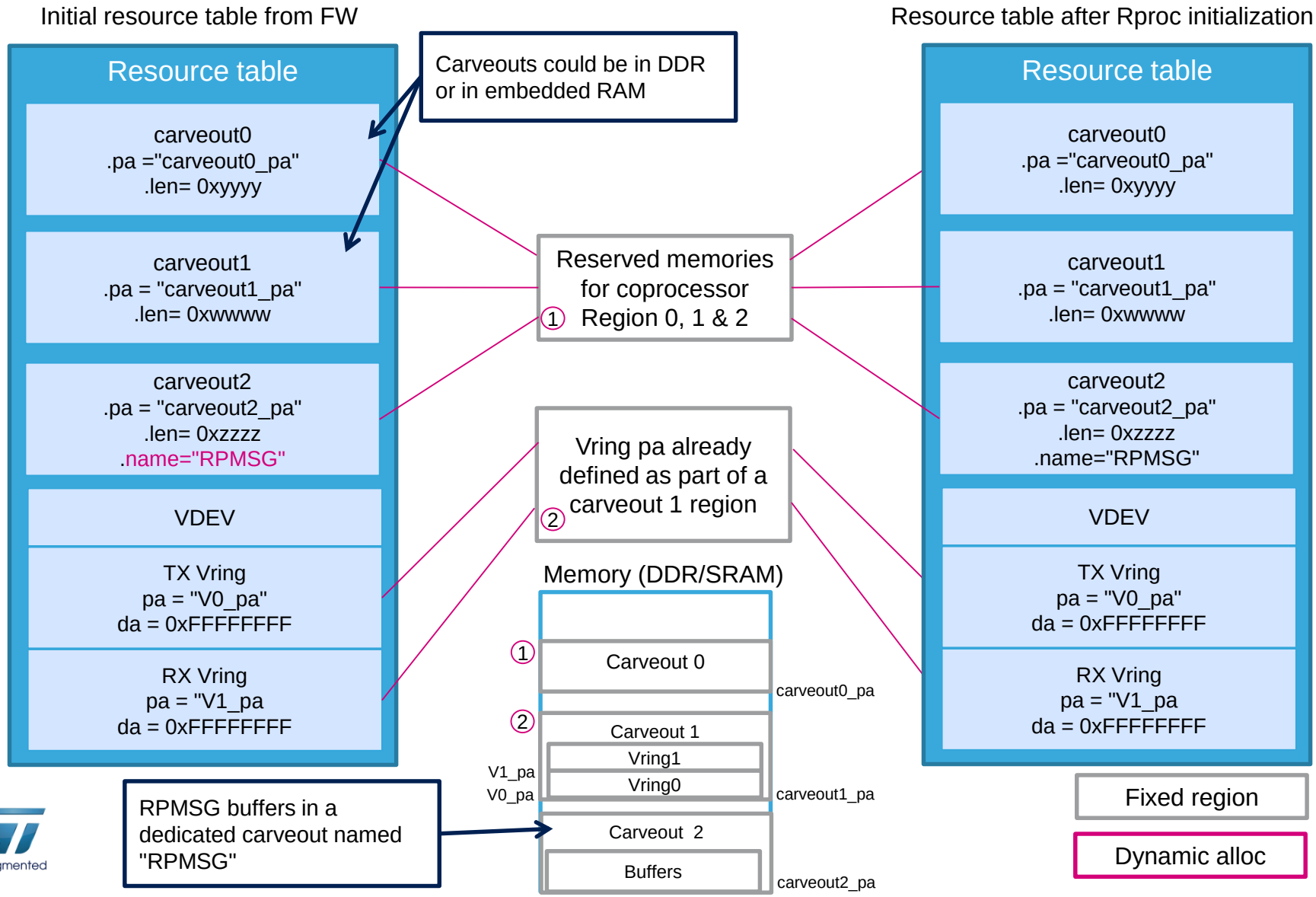
Full static definition With one Carevout

Not applicable if rmsg buffer get by name



life.augmented

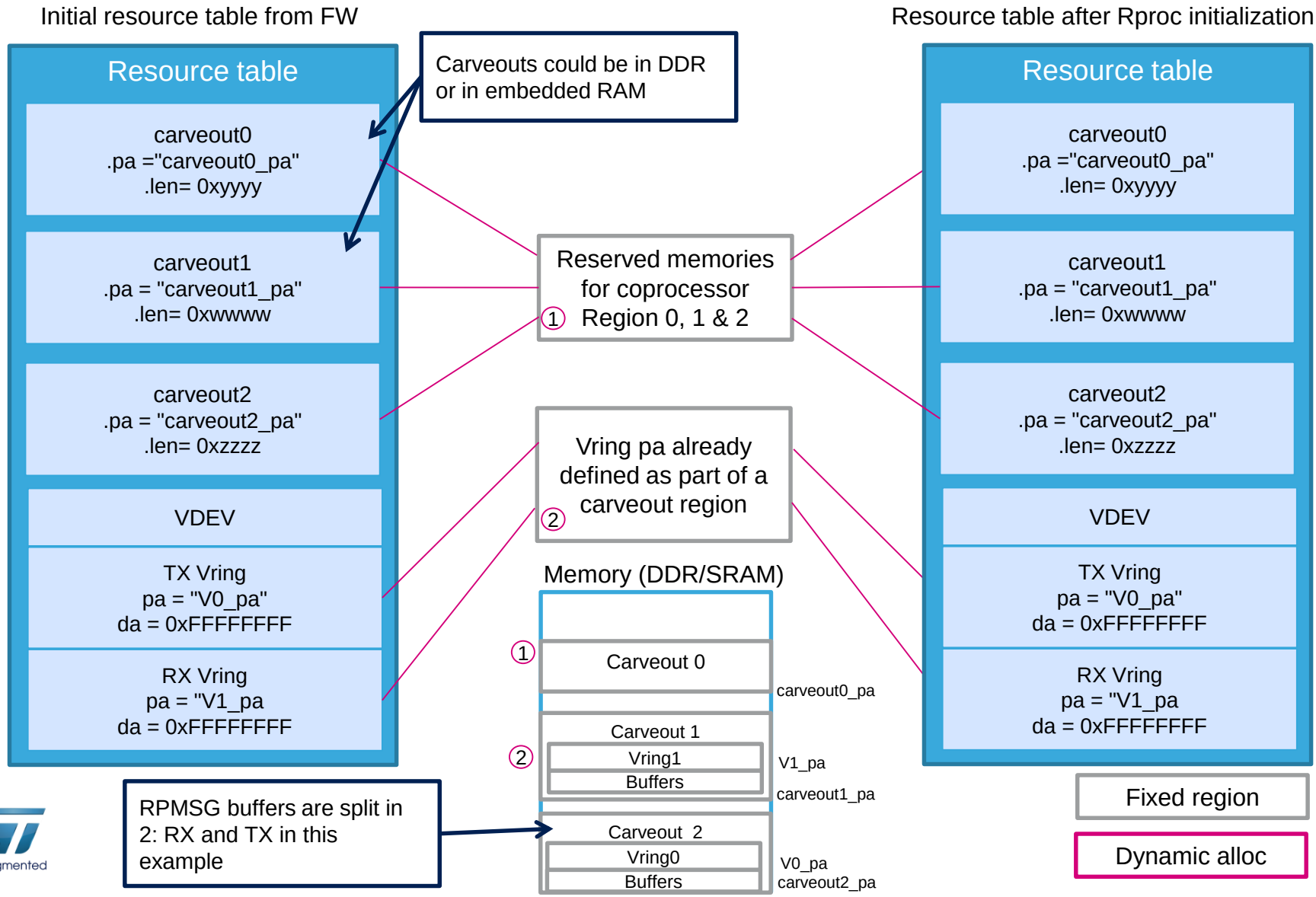
Full static definition
With several Carevouts



life.augmented

Not applicable if rmsg buffer get by name
Need one carveout by RPMSG buffer RX and TX

Full static definition With several Carevouts

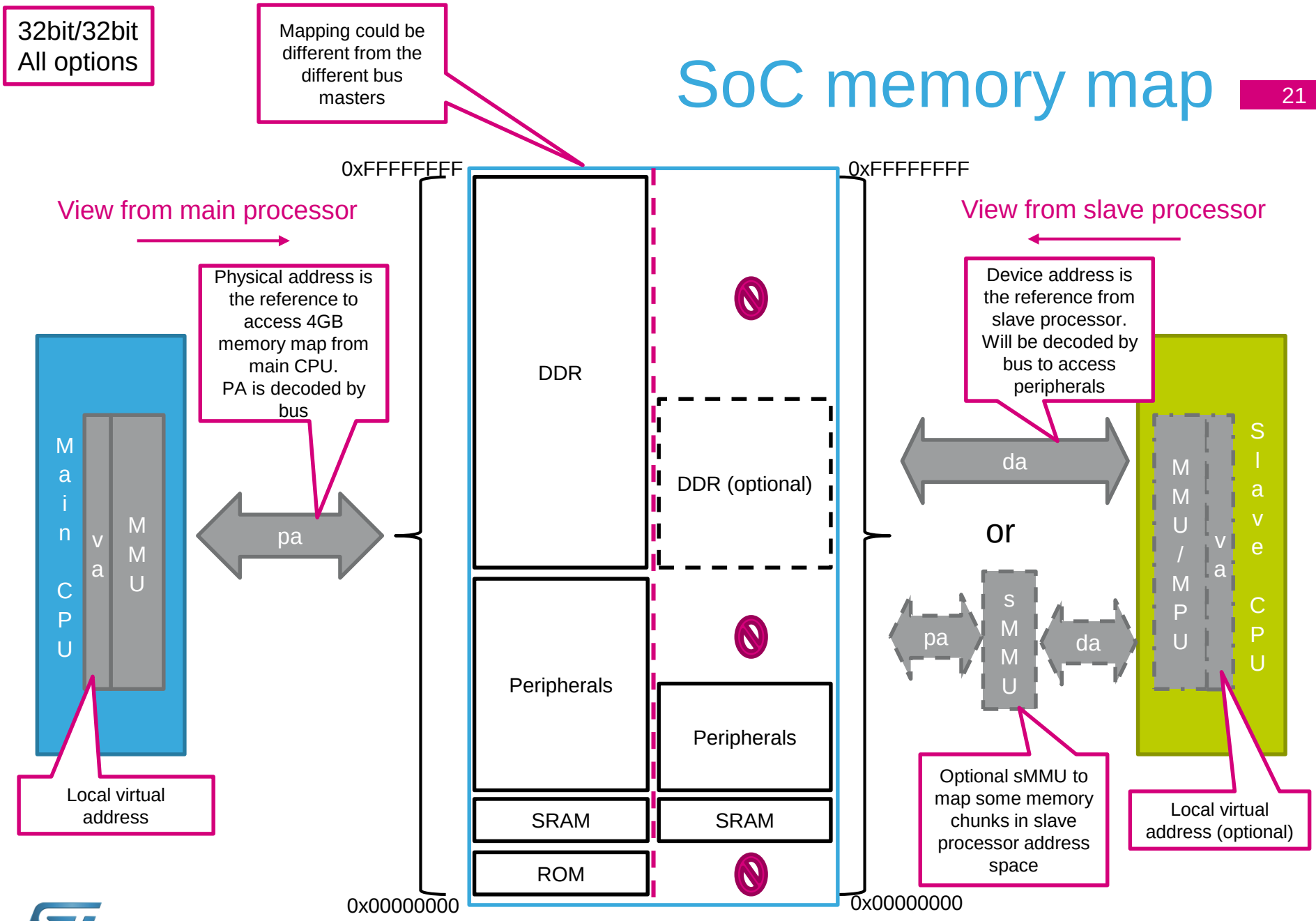


life.augmented

Memory address definition

32bit/32bit
All options

SoC memory map

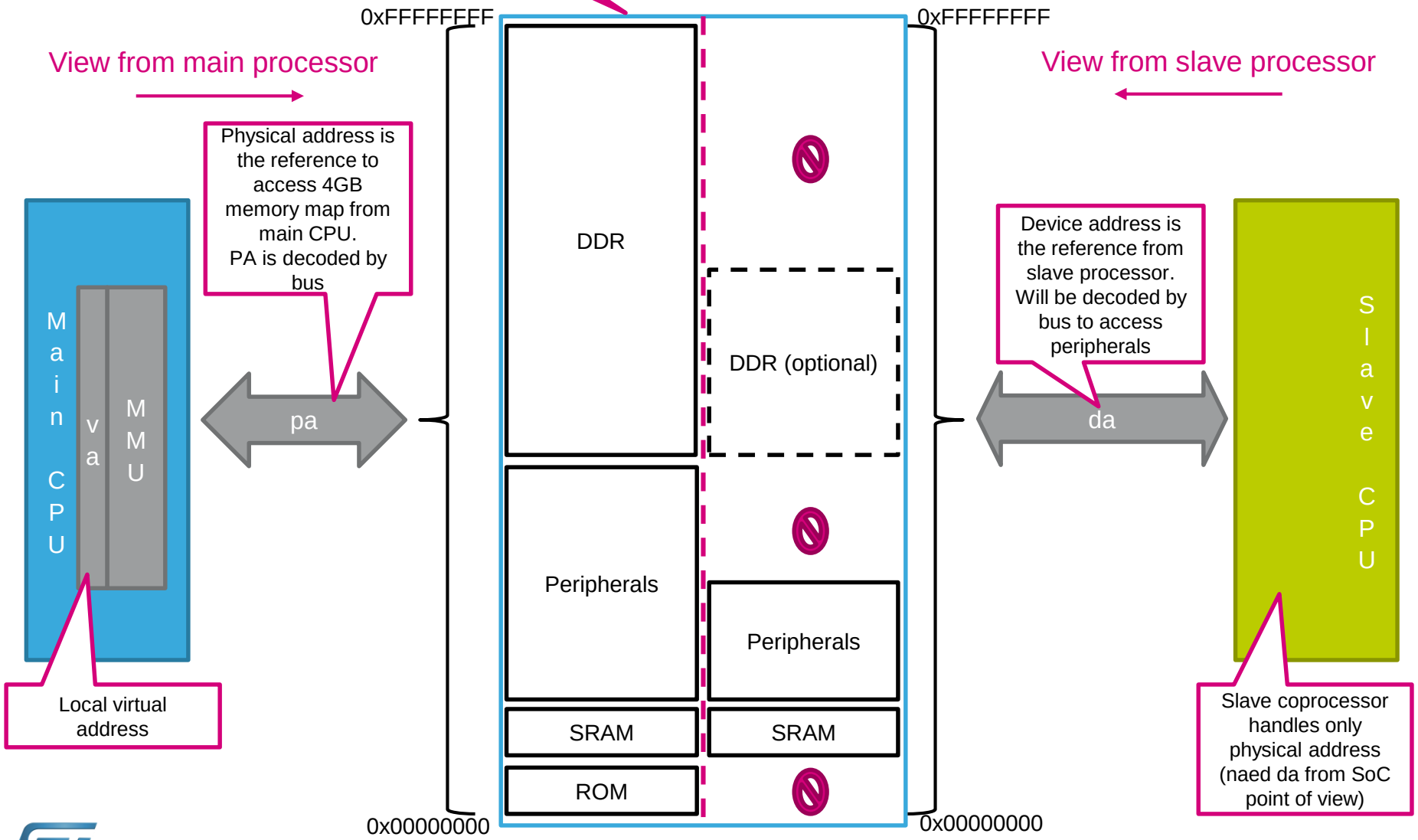


See https://en.wikipedia.org/wiki/Input%E2%80%93output_memory_management_unit

SoC memory map

32bit/32bit
Option1: coprocessor
Without MMU or sMMU

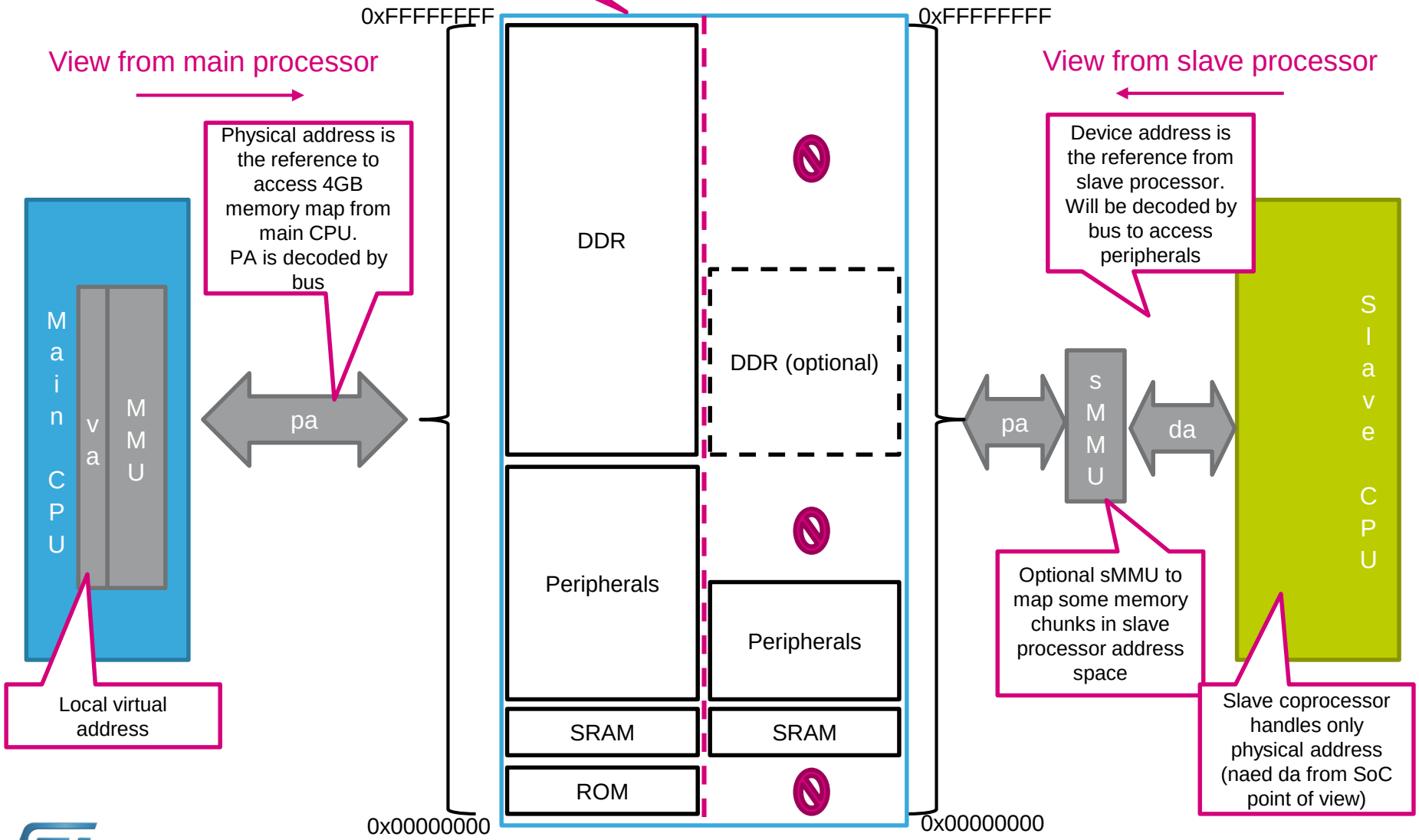
Mapping could be different from the different bus masters



SoC memory map

32bit/32bit
Option1: coprocessor
Without MMU
with sMMU

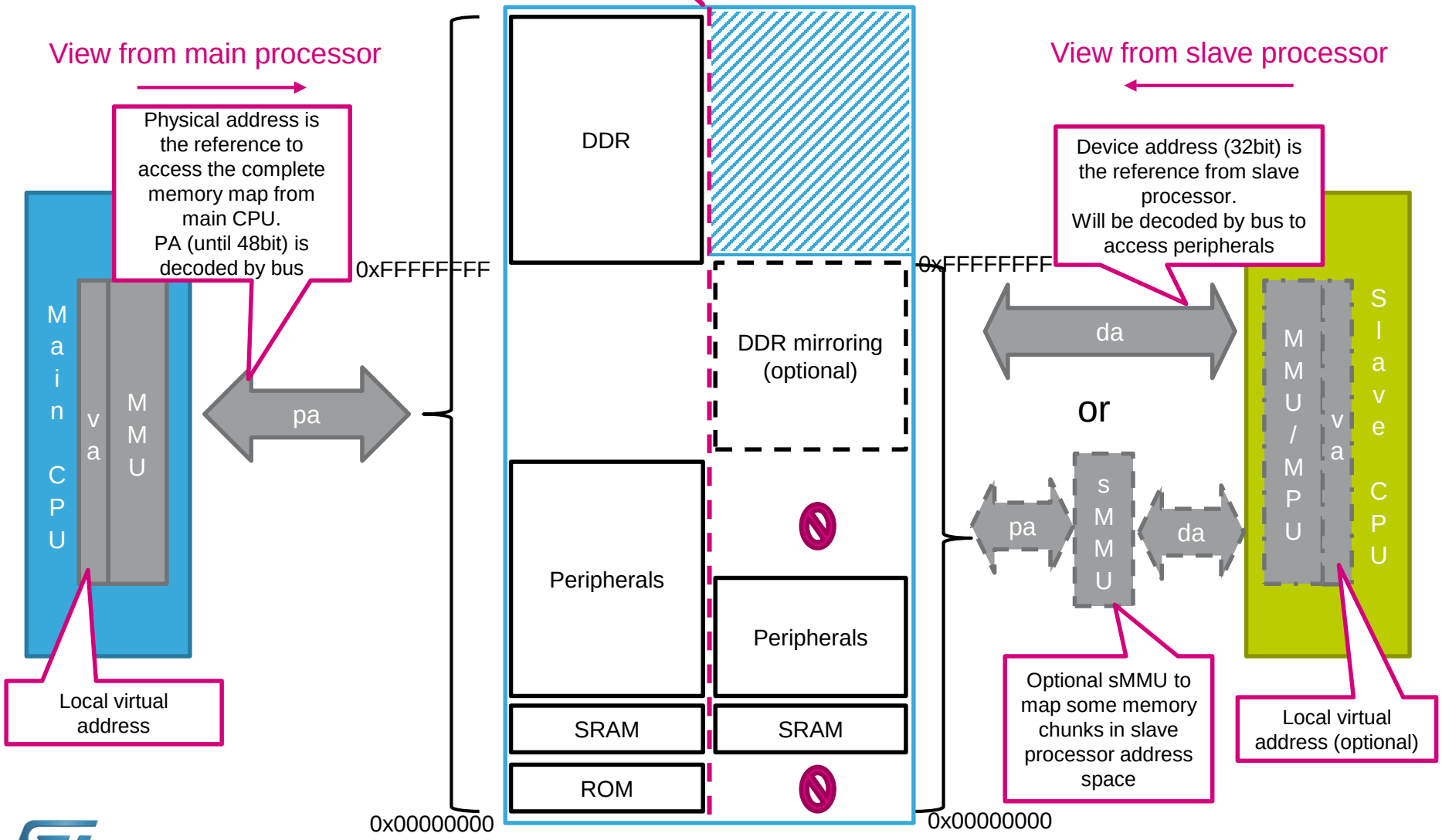
Mapping could be different from the different bus masters



SoC memory map

64bit main /32bit coprocessor
All options from copressor pov

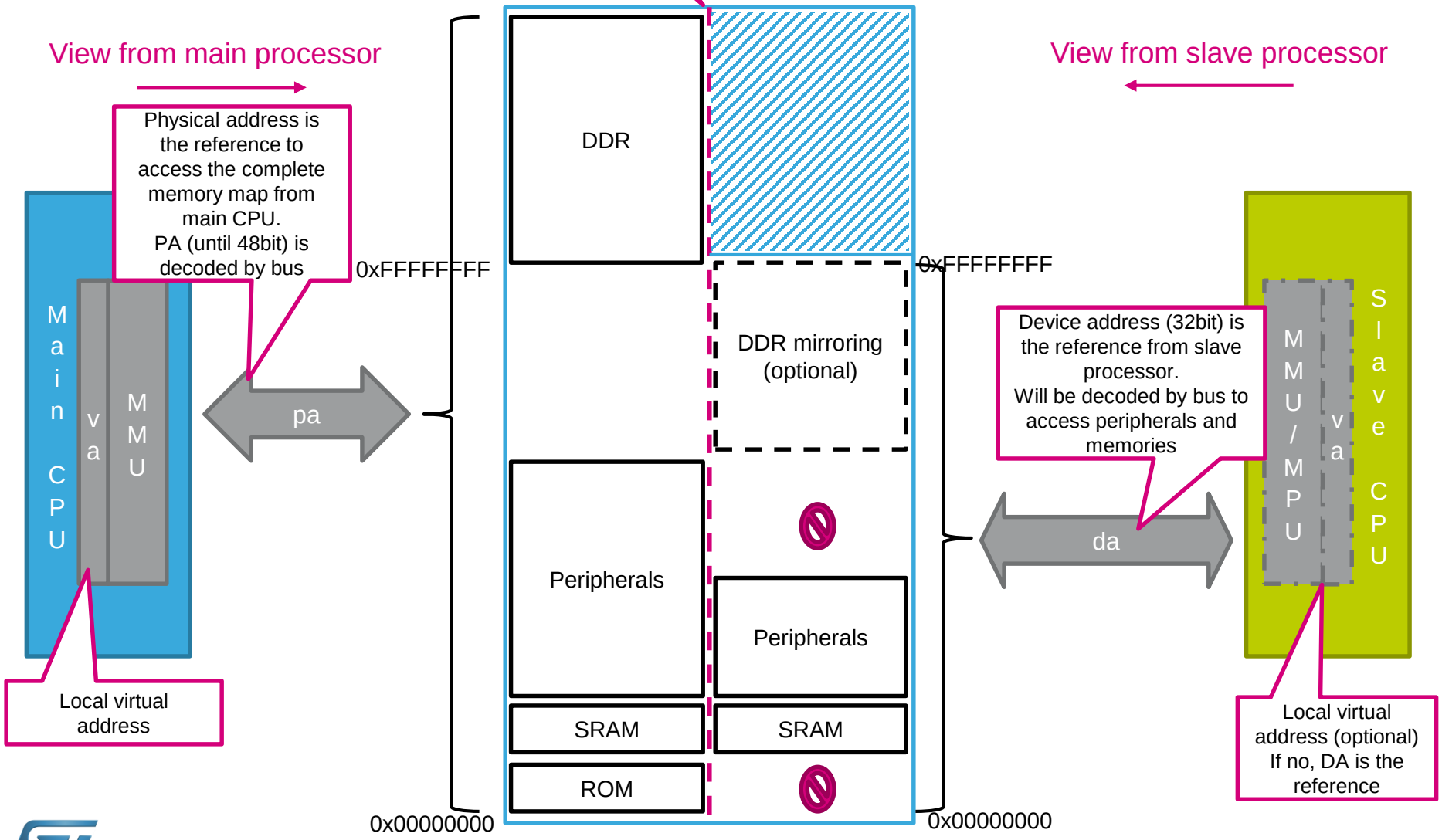
Mapping is different as processors don't have the same address space



SoC memory map

64bit main /32bit coprocessor
Option1: coprocessor
Without MMU or sMMU

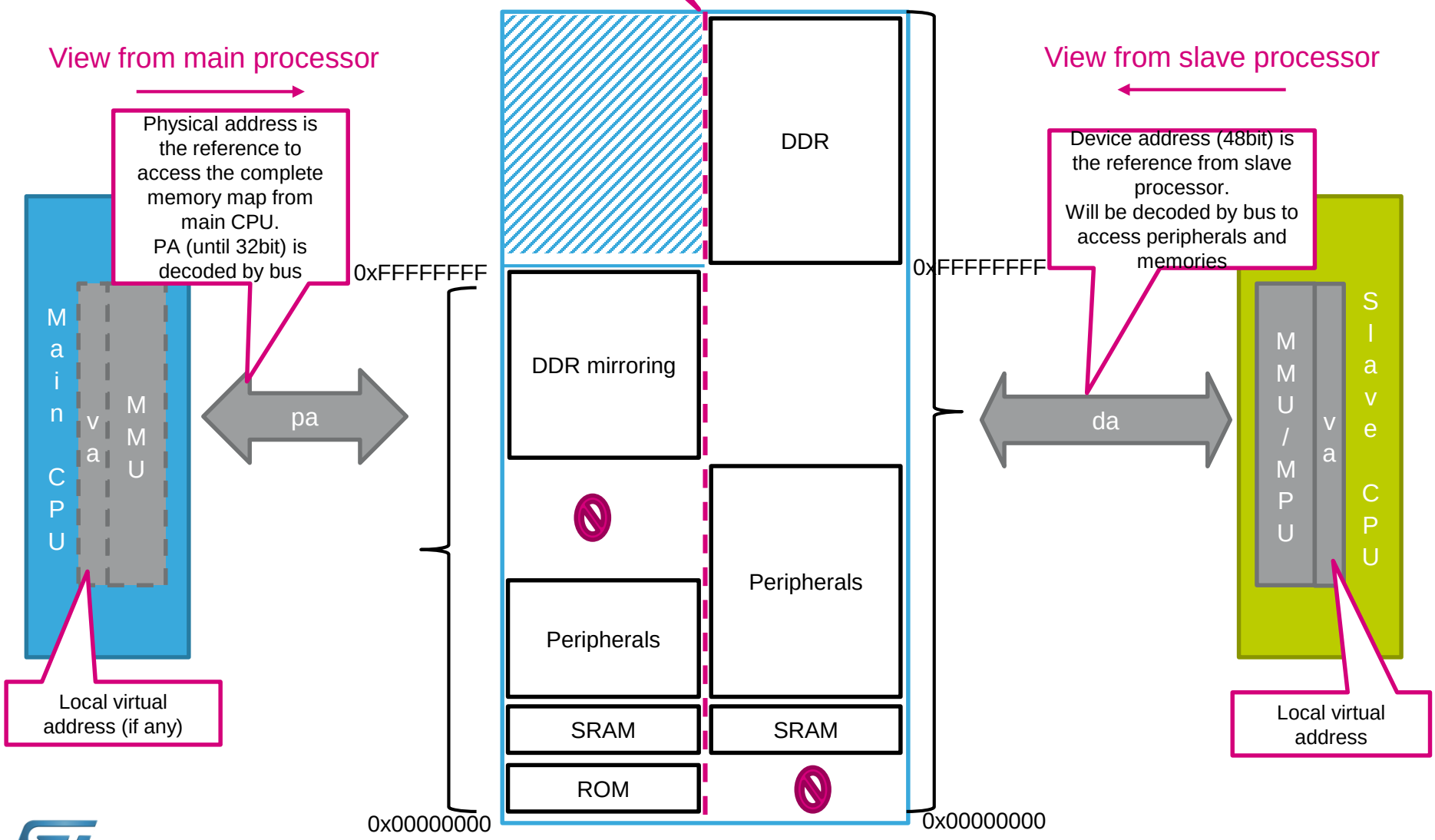
Mapping is different as processors don't have the same address space



SoC memory map

32bit main /64bit coprocessor
CortexM considered as main processor
In charge of loading and starting Cortex-A Application coprocessor.
Linux slave rpmsg use case

Mapping is different as processors don't have the same address space



Carveout handling according to PA and DA definition

Value filed during firmware generation

Use cases	PA	DA	Actions
Case 1	= 0xFF…FFFF (-1)	= 0xFF…FFFF (-1)	Main CPU to allocate carveout buffer and fill PA field. If sMMU → main cpu to program it to bind PA on a DA and fill DA field. If no sMMU → main CPU should know how to convert PA in DA (part of platform driver) and fill DA field
Case 2	= 0xFF…FFFF (-1)	= defined address	Two different cases: - Carveout memory has been reserved (pre-allocated) as this area is defined/fixed due to same remote processor constraints. Main CPU to get PA via platform driver (carveout list). Access/allocation and sMMU configuration (if any) under platform driver responsibility. - DA not found in carveout list, Main CPU need to allocate carveout buffer and fill PA field. If sMMU → main cpu to program it to bind PA on specified DA
Case 3	= defined address	= defined address	Carveout memory should be reserved (pre-allocated and registered in carveout list). Main CPU to enable its access If sMMU → main cpu to program it to bind PA on specified DA If no sMMU → nothing to do

Resource table definition update

- To be future proof, all resources should support versioning
- To support peer to peer mode, carveout resource owner should be defined.
- Carveout resource need to specify dedicated memory attributes like cached/coherency…
- Address should be 32bit and 64bit platform compliant
- Status may be needed to indicate region is ready to access

Resource header update

-
```
/**
 * struct fw_rsc_hdr - firmware resource entry header
 * @type: resource type
 * @data: resource data
 *
 * Every resource entry begins with a 'struct fw_rsc_hdr'
header providing
 * its @type. The content of the entry itself will
immediately follow
 * this header, and it should be parsed according to the
resource type.
 */
struct fw_rsc_hdr {
        u32 type;
        u8 data[0];
} __packed;
```

All bytes of type field are not used today, Use union to insert versionning

```
struct fw_rsc_hdr {
        union {
                u32 type;
                struct {
                        u16 type;
                        u8 ver;
                        u8 ed;
                } s;
        } u;
        u8 data[0];
} __packed;
```

Carveout resource definition update

- Current fw_rsc_carveout struct is:

```
struct fw_rsc_carveout {
        u32 da;
        u32 pa;
        u32 len;
        u32 flags;
        u32 reserved;
        u8 name[32];
} __packed;
```

32bit address not compliant with 64bit product

flags field is reserved to IOMMU configuration

Carveout resource definition update

- Update proposal according to our discussion for fw_rsc_carveout struct :

```
struct fw_rsc_carveout {
        u64 da;
        u64 pa;
        u32 len;
        u16 mem_flags;
        u16 iommu_flags;
        u8  owner_id;
        u8  status;
        u8  pad[2];
        u32 reserved;
        u8 name[32];
} __packed;
```

Is 32bit lenght enough in case of 64bit coprocessor?

DA field during firmware generation or during IOMMU configuration

Need for peer to peer allocation

Should we keep a reserved as versionning exists now?

Vring resource definition update


```
struct fw_rsc_vdev_vring {
        u64 da;
        u32 align;
        u32 num;
        u32 notifyid;
        u64 pa;
} __packed;
```

Different use cases:

- Fixed during firmware generation, should belong to a carveout area
- Filled by remoteproc core during vring allocation; da = iommu configuration or pa translation done platform driver

PA maybe needed by remote processor for DMA transfer programming

Trace buffer update


```
struct fw_rsc_trace {
        u64 da;
        u32 len;
        u32 reserved;
        u8 name[32];
} __packed;
```

Different use cases:

- Fixed during firmware generation, should belong to a carveout area
- Filled by remoteproc core during vring allocation; da = iommu configuration or pa translation done platform driver

How to create link between vdev and RPMsg buffer carveout

- 2 possibilities
  - No information in resource table: simply rely on carveout rsc name to find carveout associated to RPMsg buffer
  - Use an additional "resource index" to refer to carveout dedicated to RPMsg buffer.
    - Add carveout index in struct fw_rsc_vdev if only one memory pool is needed for RX and TX buffers. Valid in current configuration in which master allocates all buffers
    - Add carveout index in struct fw_rsc_vdev_vring to associate memory pool with vring. Will be compliant with peer to peer.

Remoteproc initialization

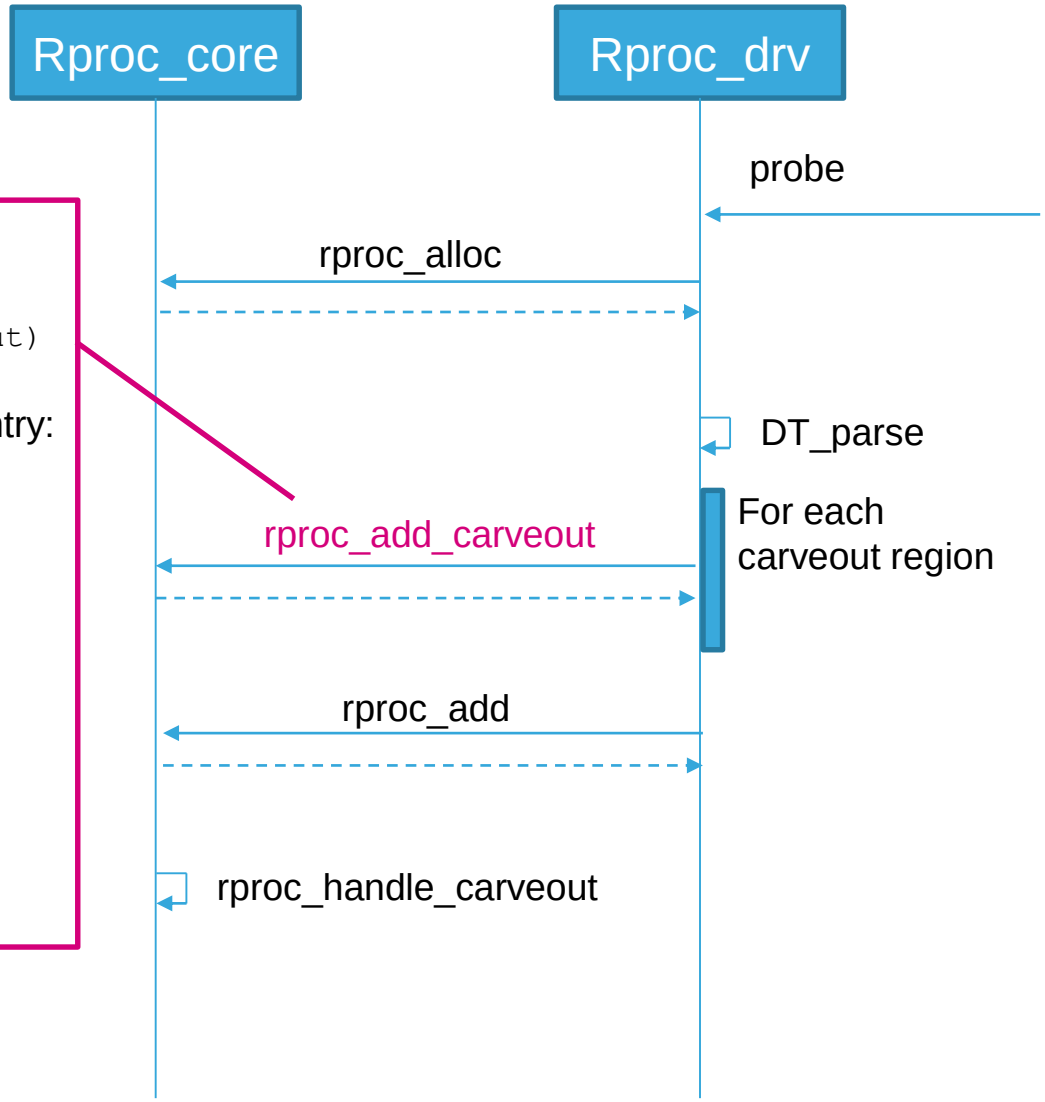
New API to declare carveout

```
int rproc_add_carveout(struct rproc
*rproc, struct rproc_mem_entry *carveout)
```

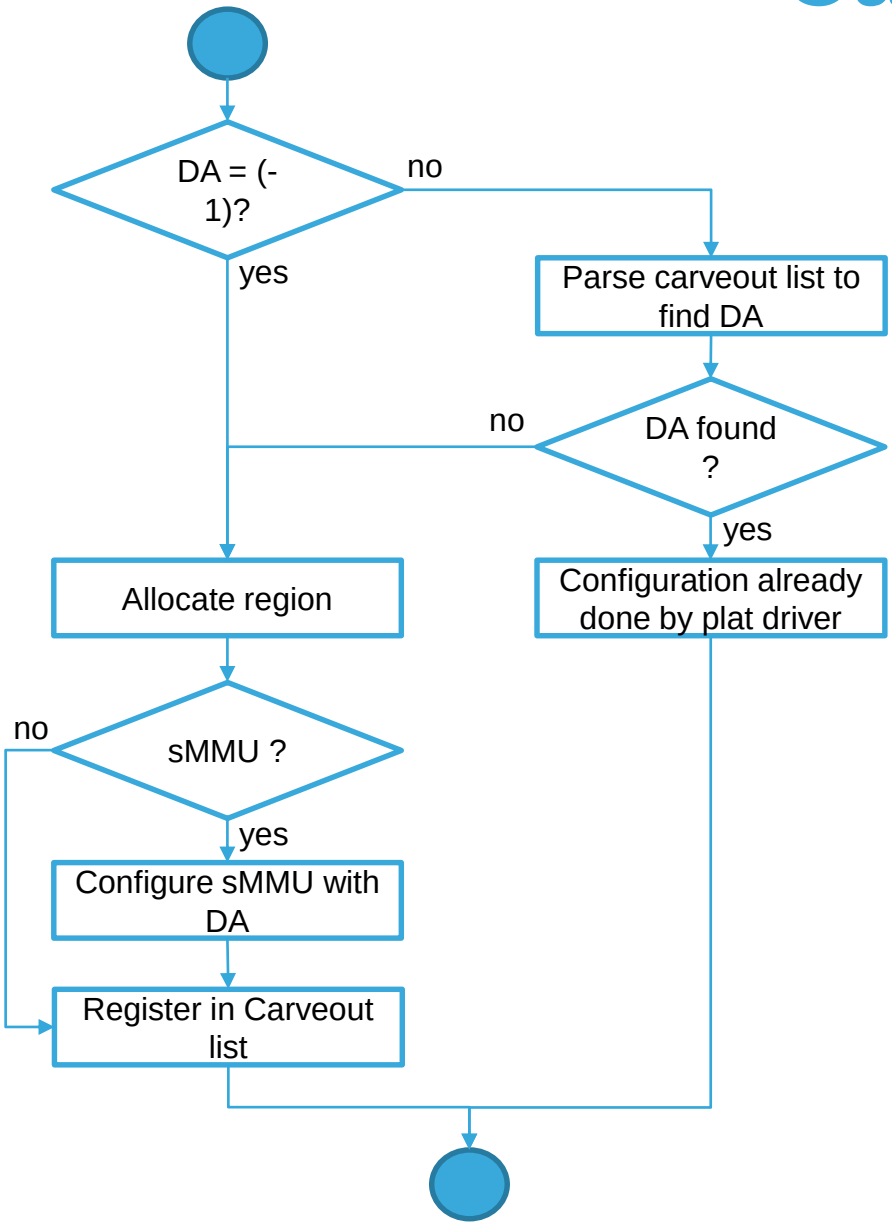
With following change in struct rproc_mem_entry:

```
struct rproc_mem_entry {
          void *va;
          dma_addr_t dma;
          int len;
          u64 da;
          void *priv;
          void (*free)(struct rproc
*rproc, struct rproc_mem_entry *mem);
          struct list_head node;
};
```

Could priv field be used ?



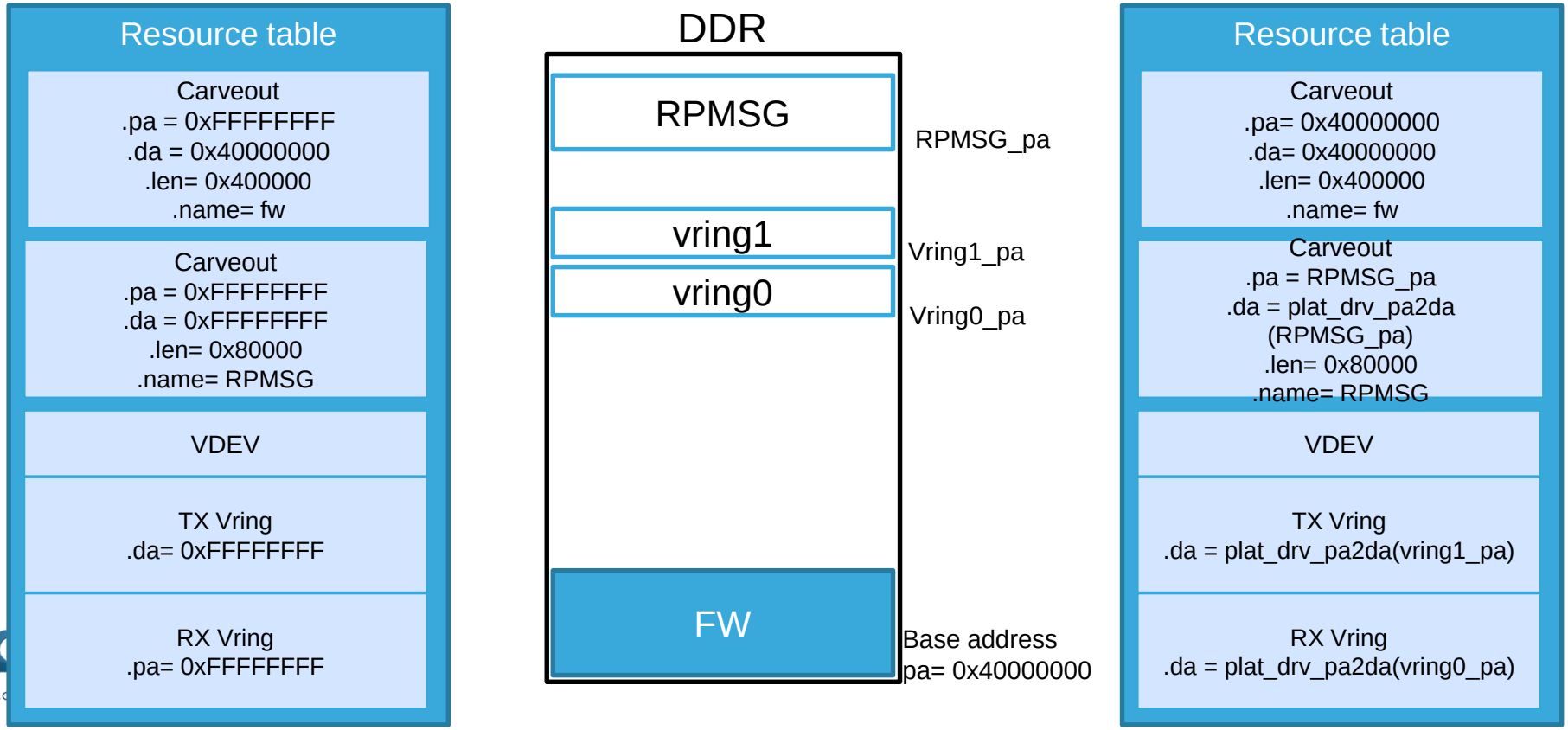
Carveout handling



life.augmented

Example for ST platform B2260

- Need 2 carveouts
  - 1 carveout with fixed address for code and data (optional as platform driver could register carveout based on DT definition)
  - 1 carveout RPMsg buffer location, dynamically allocated by master



History

Date	Version	Comment
12/07/17	V1	Creation
10/08/17	V2	Add memory address definition
17/08/17	V3	Complete memory definition according to 10/08/17 OpenAMP weekly Add fw_rsc_carveout struct evolution proposal
31/08/17	V4	Add resource evolution proposal Add B2260 resource table example
24/09/17	V5	Update resource table structures
04/10/17	V6	Update after review with Wendy Add carveout handler flowchart