# PASR Framework

## Saving the Power Consumption of the Unused Memory

**Maxime Coquelin**

**Loïc Pallardy**

# Content

- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

# Content

- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

# Context

- Trend: Increase of DDR size in embedded devices
  - From 1GB today
  - Up to 8GB tomorrow
- Major contributor to platform power consumption
  - About 25% of floor current with 512MB configuration on Novathor 8500
  - About 70% of floor current with 2GB configuration on Novathor 9540
- More and more DDR bandwidth requested
  - New chipset generation proposes DDR die interleaving.

**Control memory power consumption is now mandatory to reach Mobile Phone manufacturers power consumption requirements**

# Content

- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

# Partial Array Self-Refresh (PASR)

- Stop refreshing unused chunks of memory
- Four modes available
  - Single-ended
  - Double-ended
  - Bank-selective
  - Segment-selective
- Bank and segment selective modes are the best-adapted to Linux
- But depends on DDR
  - Bank-selective: Around 40µA @3.7V gain per 64MB bank masked

DDR power management mechanisms

# Deep Power-Down (DPD)

- Shutdown full DDR die and its internal controller
- Better power consumption gain
  - Around 400µA @3.7v saved per 4Gb die (8 banks) in DPD.
- Constraints:
  - Wake-up latency: 220µs
  - Minimum DPD duration: 500µs
- DPD and PASR can coexist

- Interesting for power saving
- But difficult with interleaving

# Content

- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

Existing concepts

# Linux Memory Hotplug

- Allows to insert/remove memory chunks in/from the allocator

+ Already available in mainline Kernel

+ Solution envisaged by Linaro for Memory PM

- No officially ARM architecture support

- Introduce high latencies

- No check of unmovable page presence before starting sequence

- Require governor to decide when to plug/unplug the memory

- No DDR PASR/DPD support

# Content

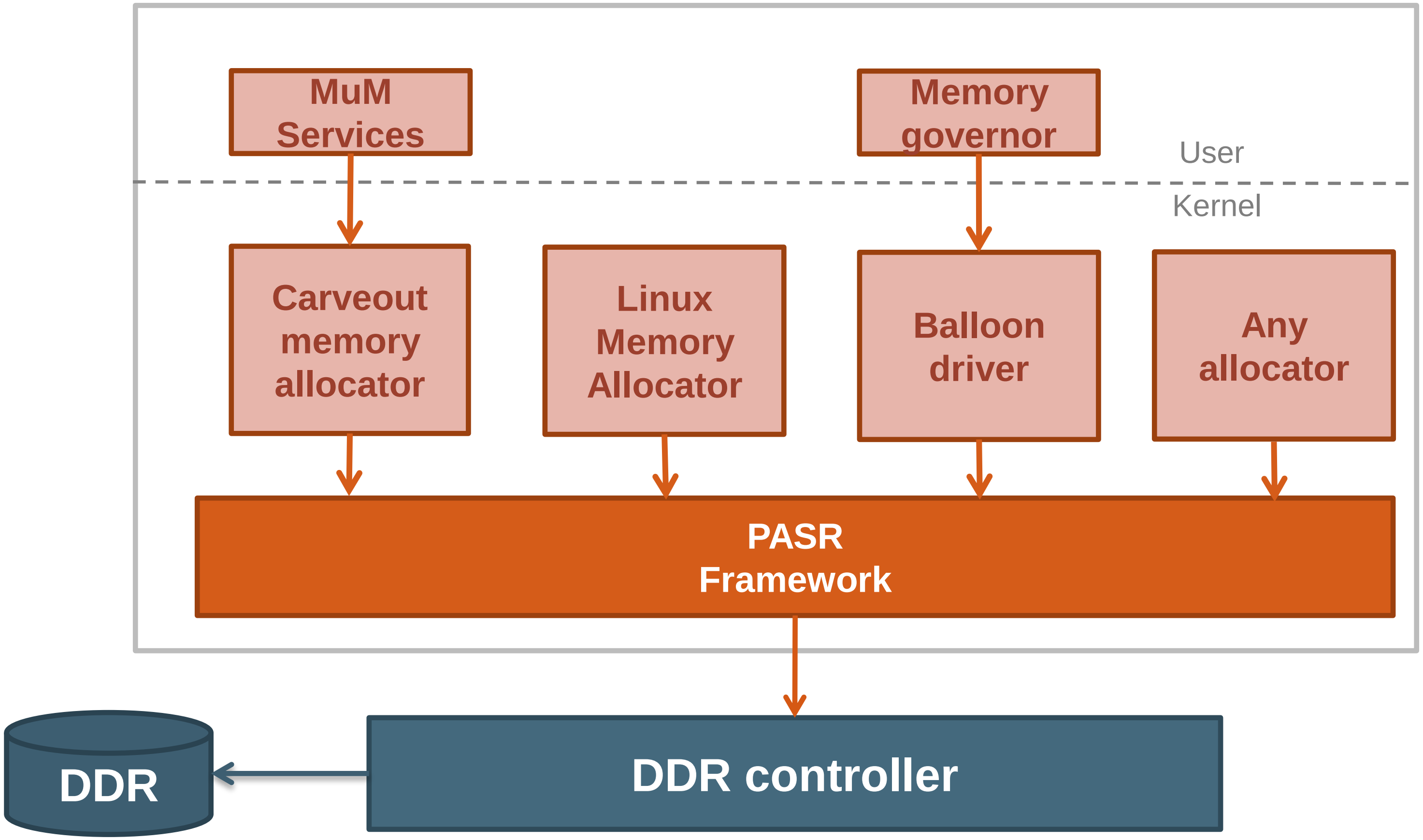
- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

# Description

- Add DDR PASR support on Linux platforms
- Characteristics:
  - Complete DDR memories topology
  - Bank and Segment configurations support
  - DDR die interleaving support
  - Compliant with DDR DPD
- Interface based on standard get/put mechanism
  - Get: Banks or segments refresh are unmasked when used
  - Put: Banks or segments refresh are masked when unused

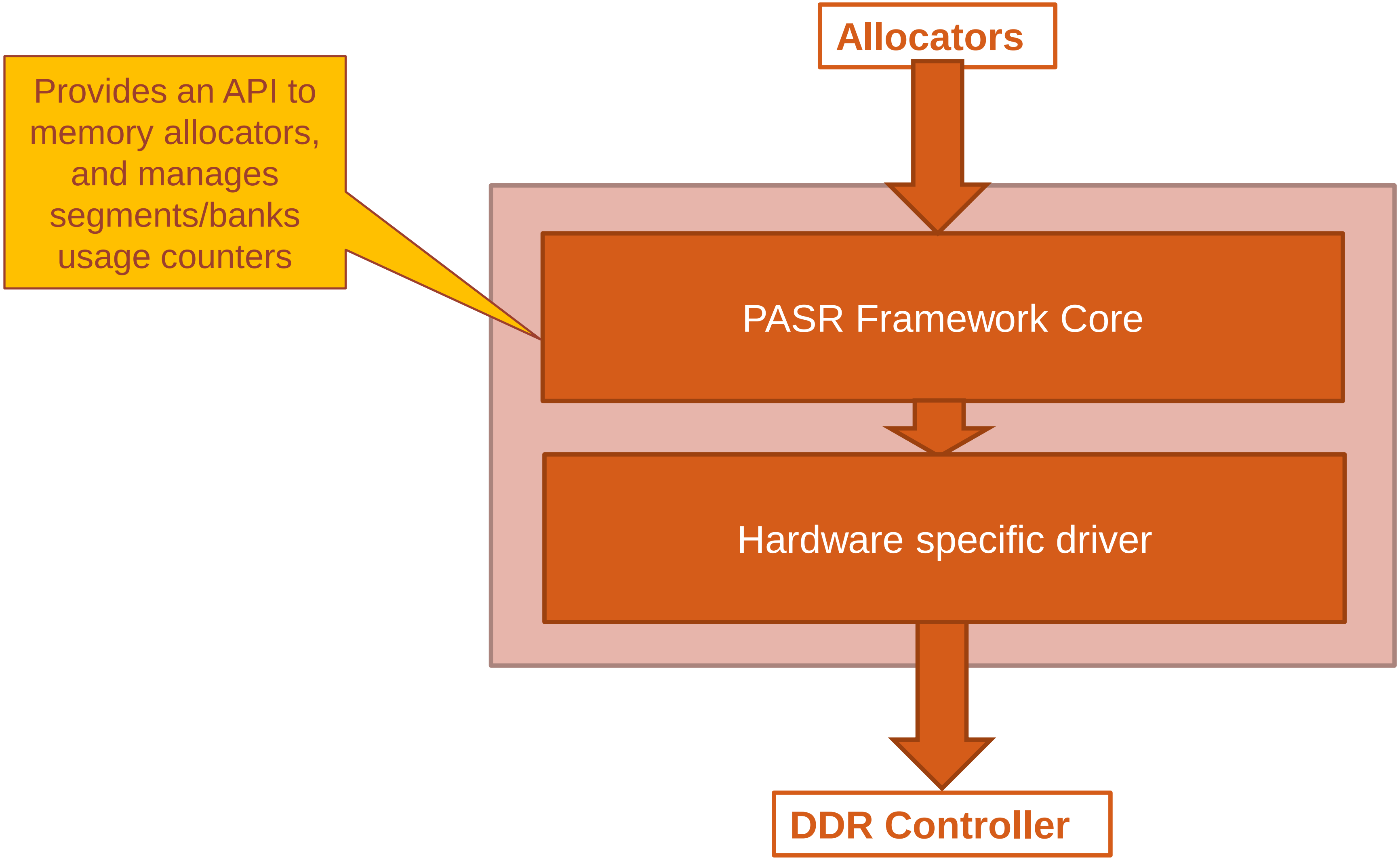
The PASR Framework

# Architecture Overview



## The PASR Framework

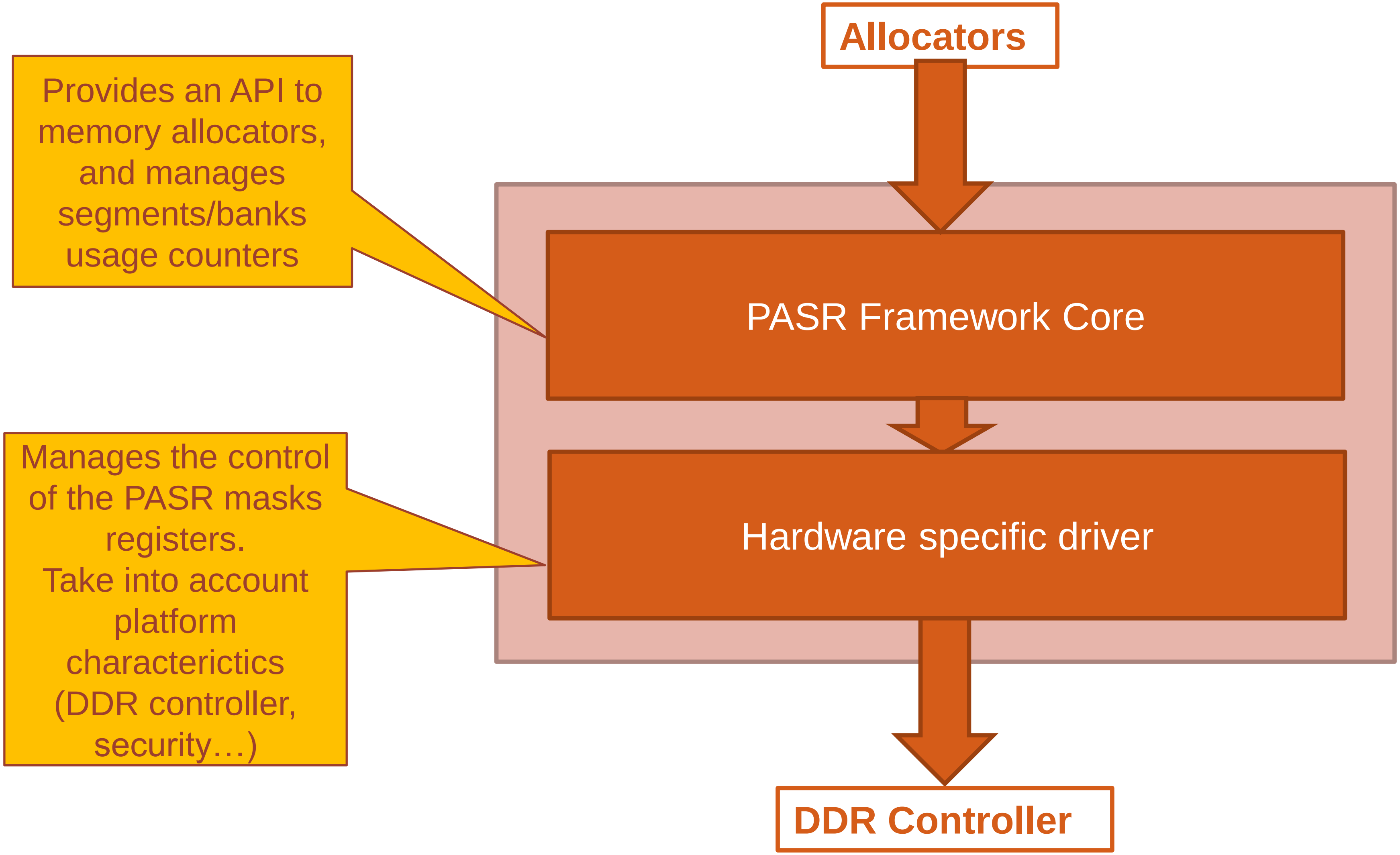
# Internal Architecture



Provides an API to memory allocators, and manages segments/banks usage counters

The PASR Framework

# Internal Architecture



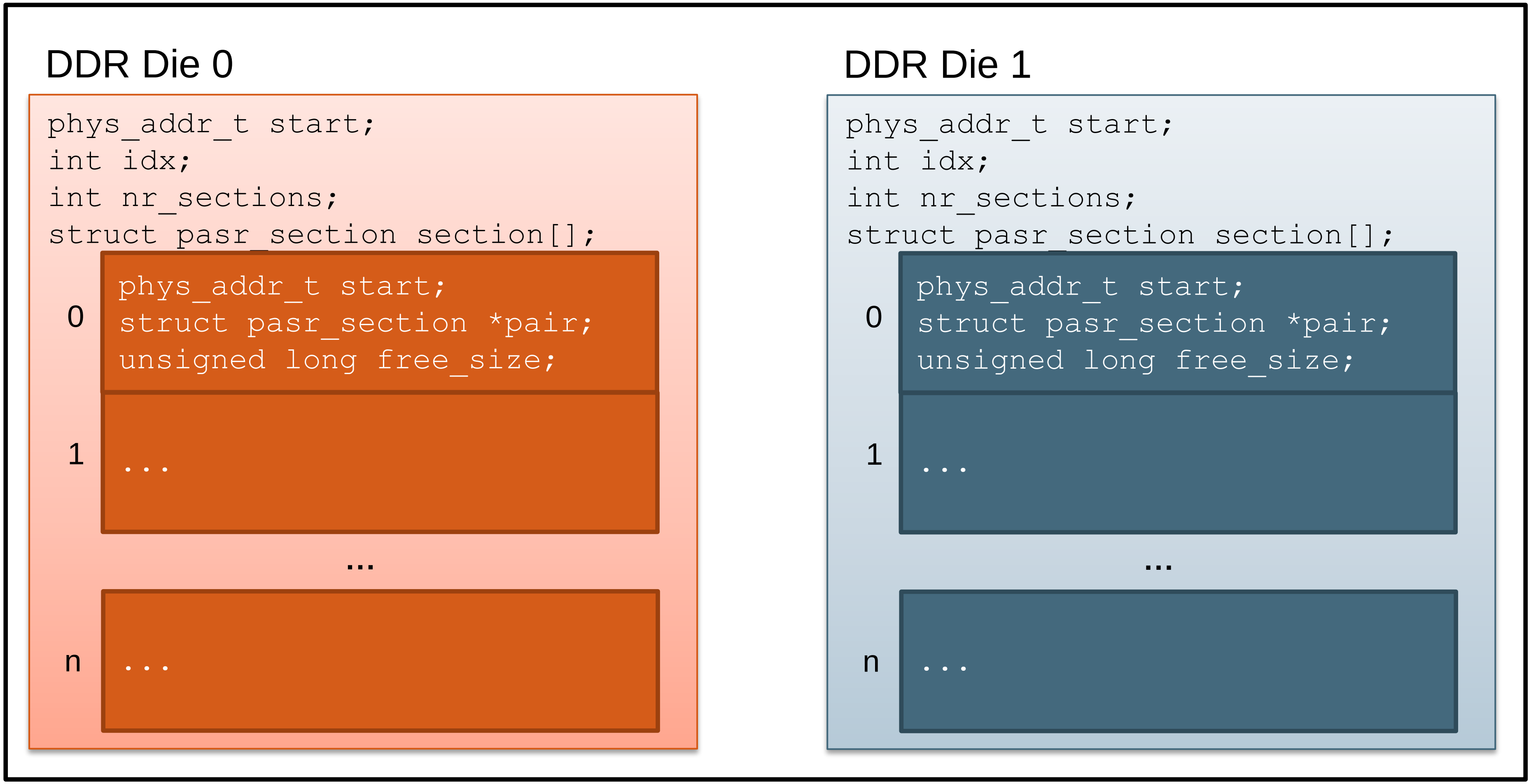
Provides an API to memory allocators, and manages segments/banks usage counters

Manages the control of the PASR masks registers.
Take into account platform characterictics (DDR controller, security…)

The PASR Framework

# Internal Structures

PASR map



DDR Die 0

```
phys_addr_t start;
int idx;
int nr_sections;
struct pasr_section section[];
```

0

```
phys_addr_t start;
struct pasr_section *pair;
unsigned long free_size;
```

1

```
...
```

...

n

```
...
```

DDR Die 1

```
phys_addr_t start;
int idx;
int nr_sections;
struct pasr_section section[];
```

0

```
phys_addr_t start;
struct pasr_section *pair;
unsigned long free_size;
```

1

```
...
```

...

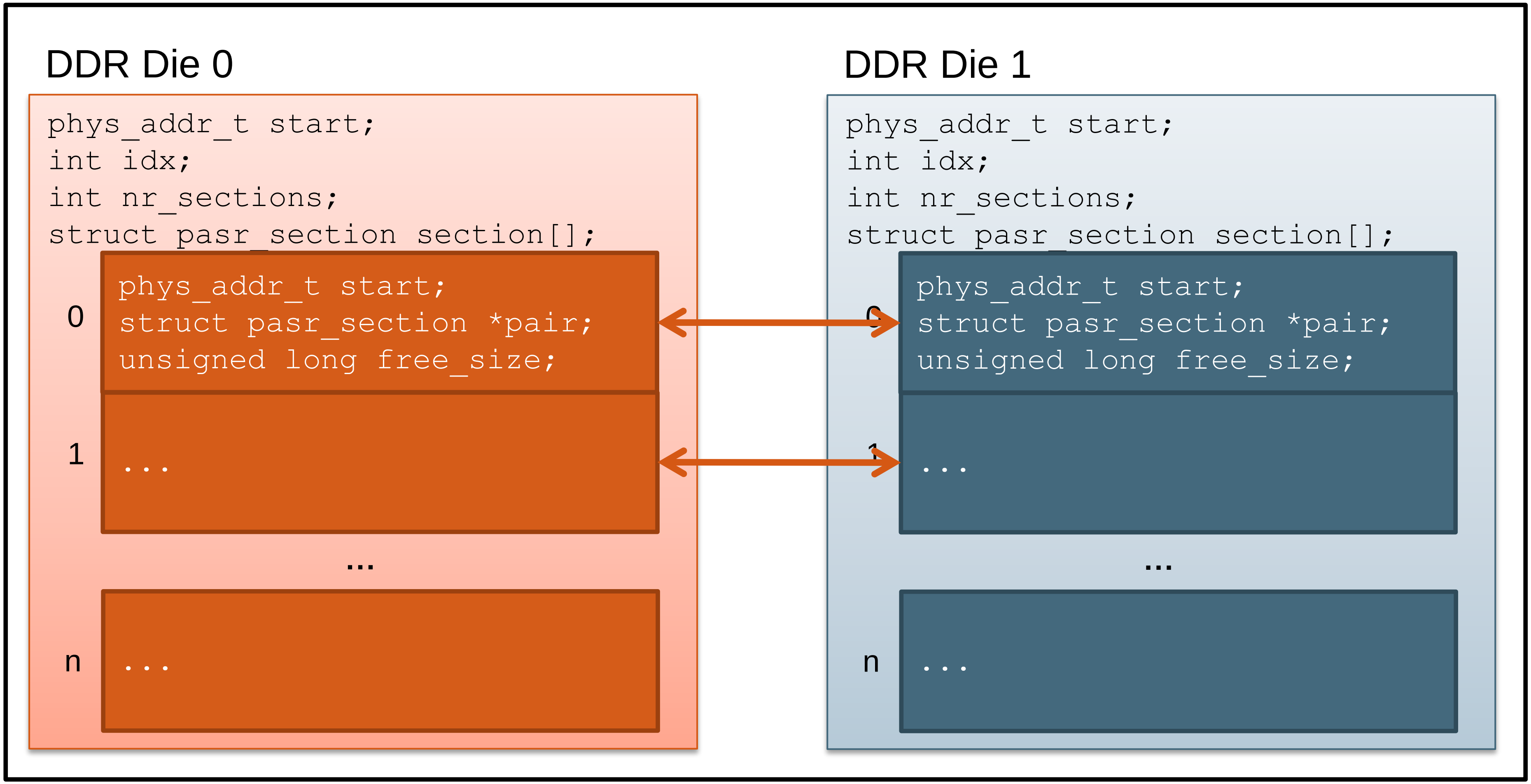
n

```
...
```

## The PASR Framework

# Internal Structures

PASR map



DDR Die 0

```
phys_addr_t start;
int idx;
int nr_sections;
struct pasr_section section[];
```

0

```
phys_addr_t start;
struct pasr_section *pair;
unsigned long free_size;
```

1

```
...
```

...

n

```
...
```

DDR Die 1

```
phys_addr_t start;
int idx;
int nr_sections;
struct pasr_section section[];
```

0

```
phys_addr_t start;
struct pasr_section *pair;
unsigned long free_size;
```

1

```
...
```

...

n

```
...
```

# Initialization

- PASR parameters passed via the Kernel command line
  - ddr_die=xxx[M|G]@yyy[M|G]
    - xxx : size of the DDR die
    - yyy : offset of the DDR die
    - E.g: `ddr_die=512M@0 ddr_die=512MB@3G` for 2x4Gb
  - interleave=xxx[M|G]@yyy[M|G]:zzz[M|G]
    - xxx : size if the interleaved section
    - yyy : offset of the section A interleaved with section B
    - zzz : offset of the section B interleaved with section A
    - E.g: `interleave=256M@0:3G` interleave 256first MB of die 0 with die 1
- Plan to support Device Tree

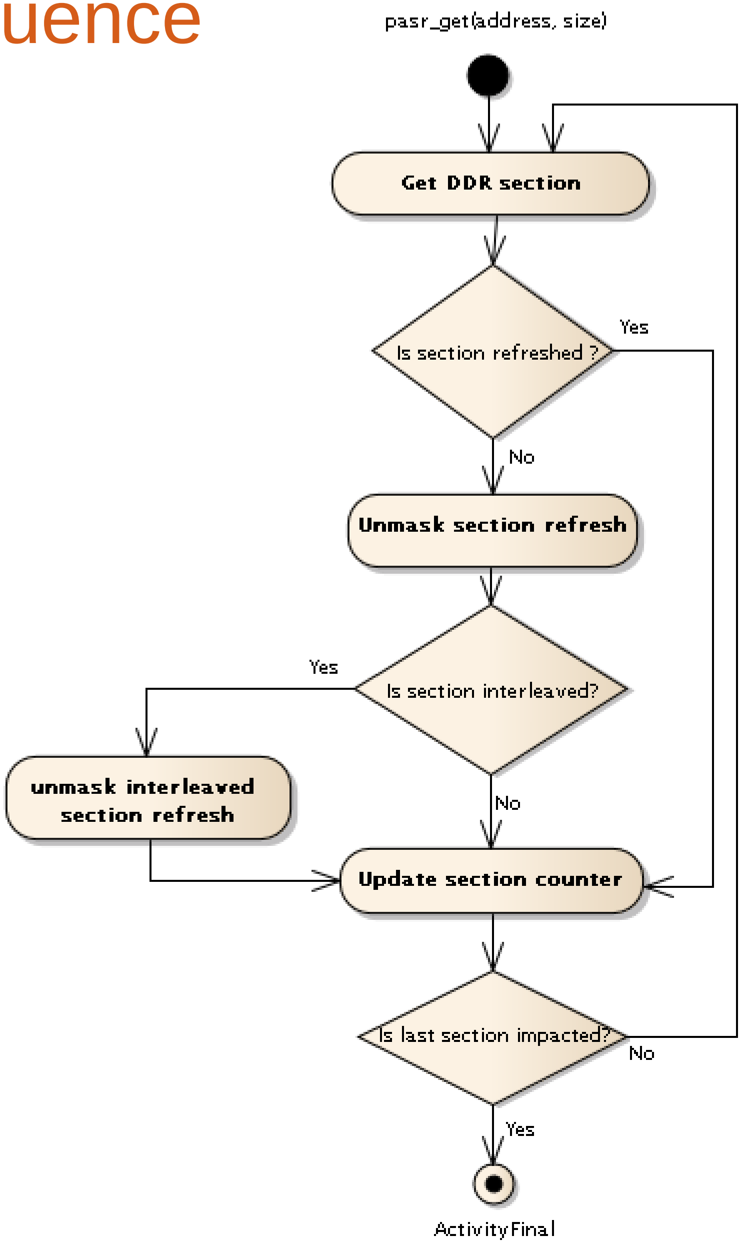
## The PASR Framework

# API description

- Generic interface (e.g. Carveout-style allocator):
  - *int **pasr_get**(phys_addr_t addr, phys_addr_t size)*
  - *int **pasr_put**(phys_addr_t addr, phys_addr_t size)*
- Page based interface:
  - *int **pasr_kget**(struct page *, int order)*
  - *int **pasr_kput**(struct page *, int order)*

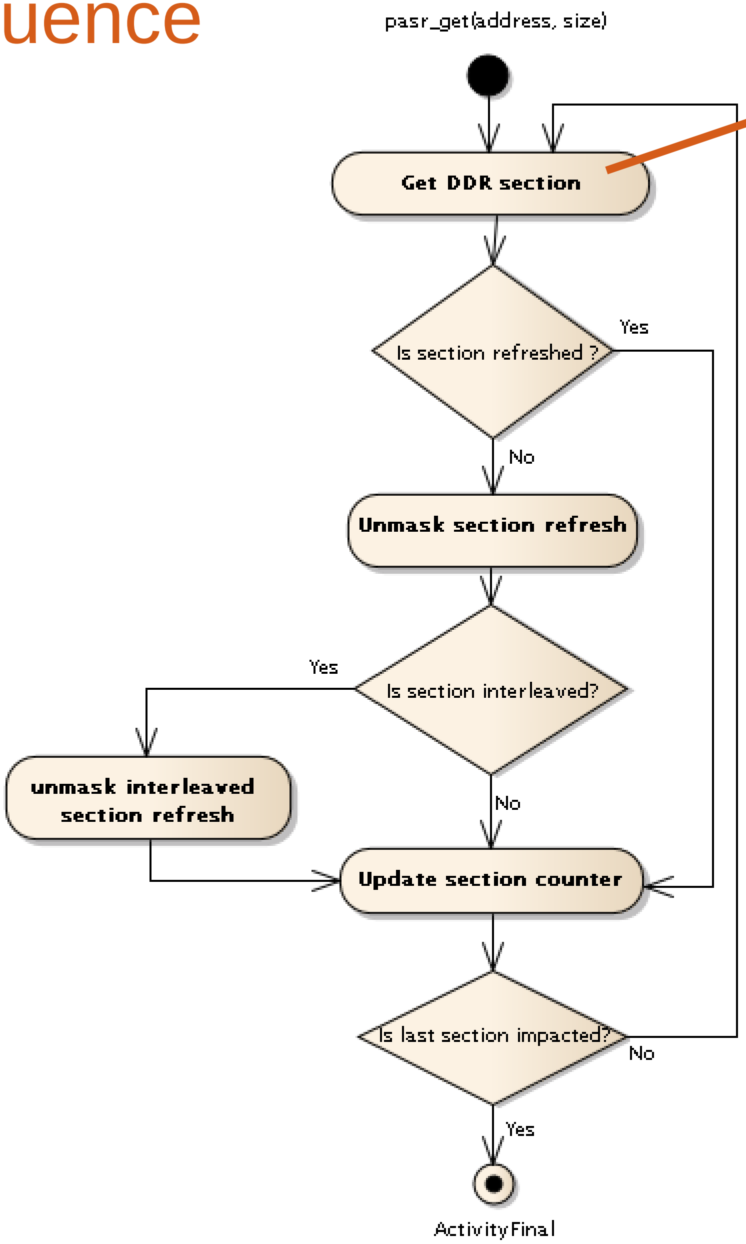
The PASR Framework

# Get sequence



The PASR Framework

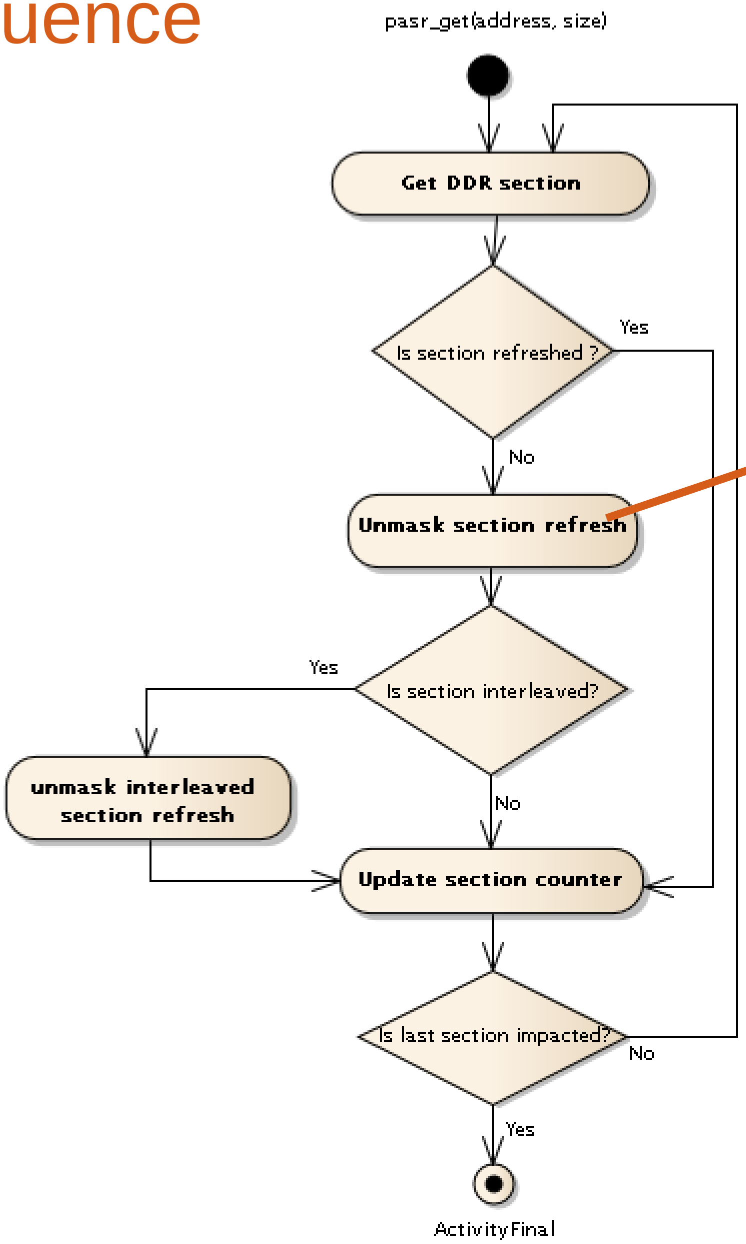
# Get sequence



Get DDR section

## The PASR Framework

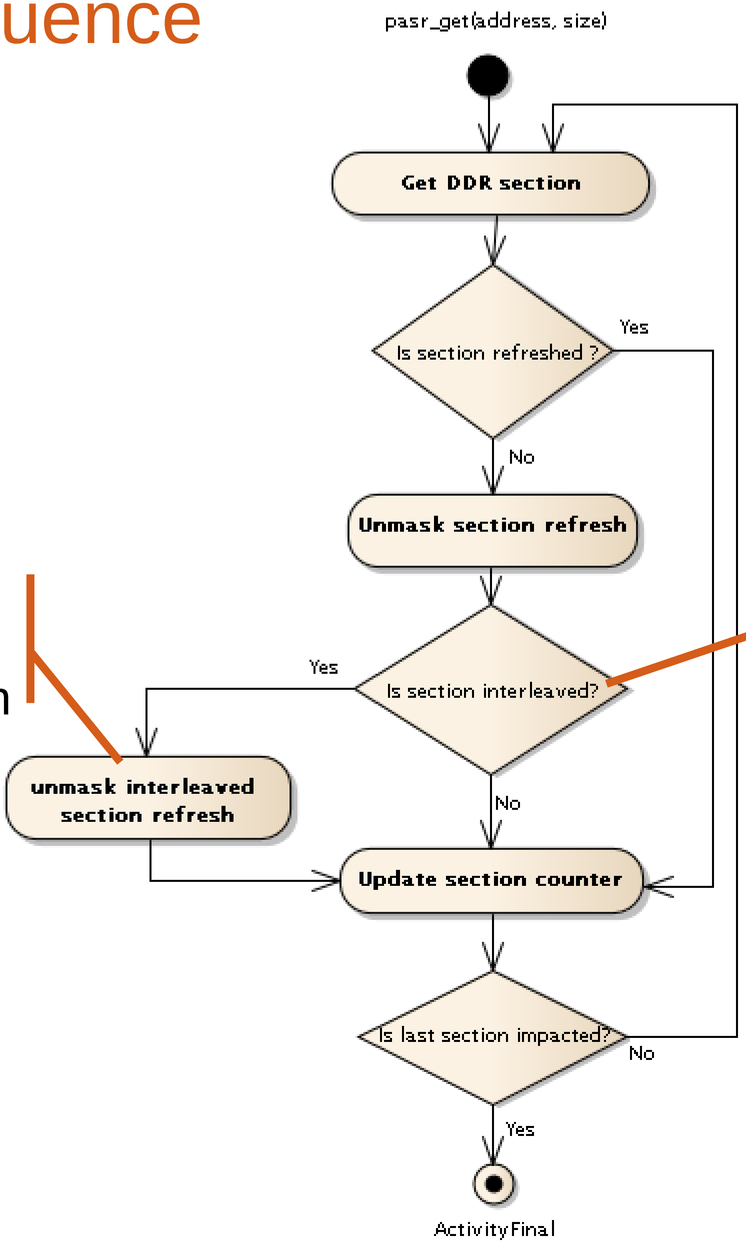
# Get sequence



Unmask section refresh

The PASR Framework

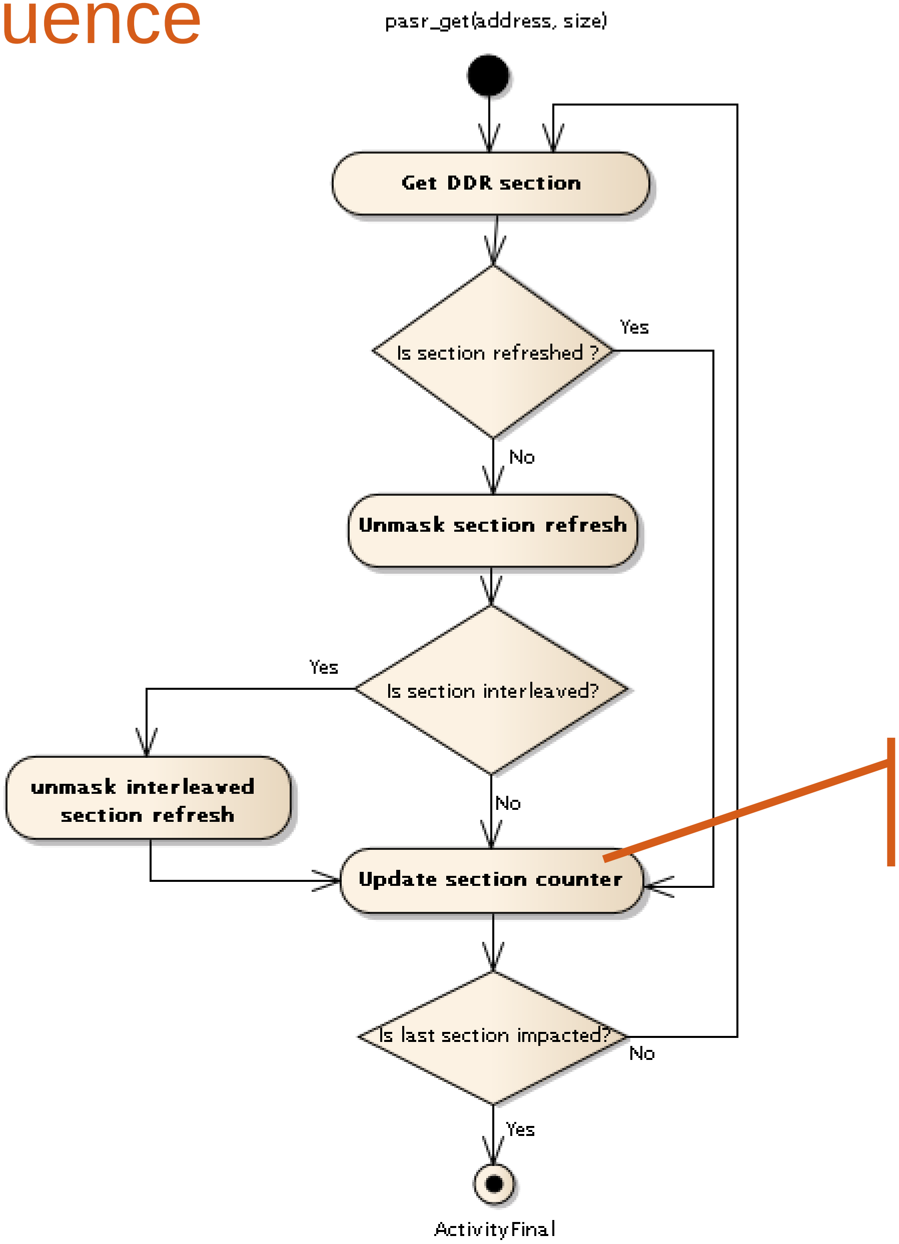
# Get sequence



Unmask interleaved section refresh

Verify interleaving

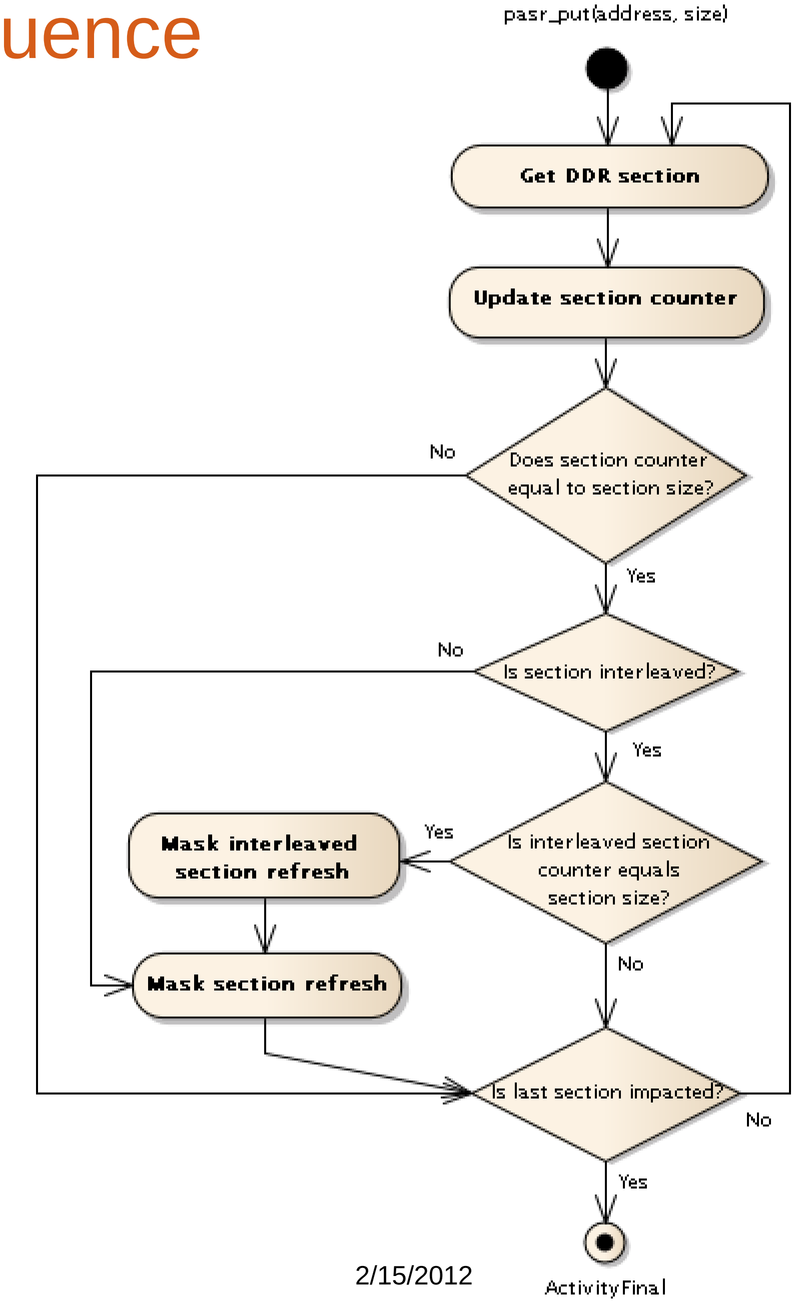
# Get sequence



Update section counter

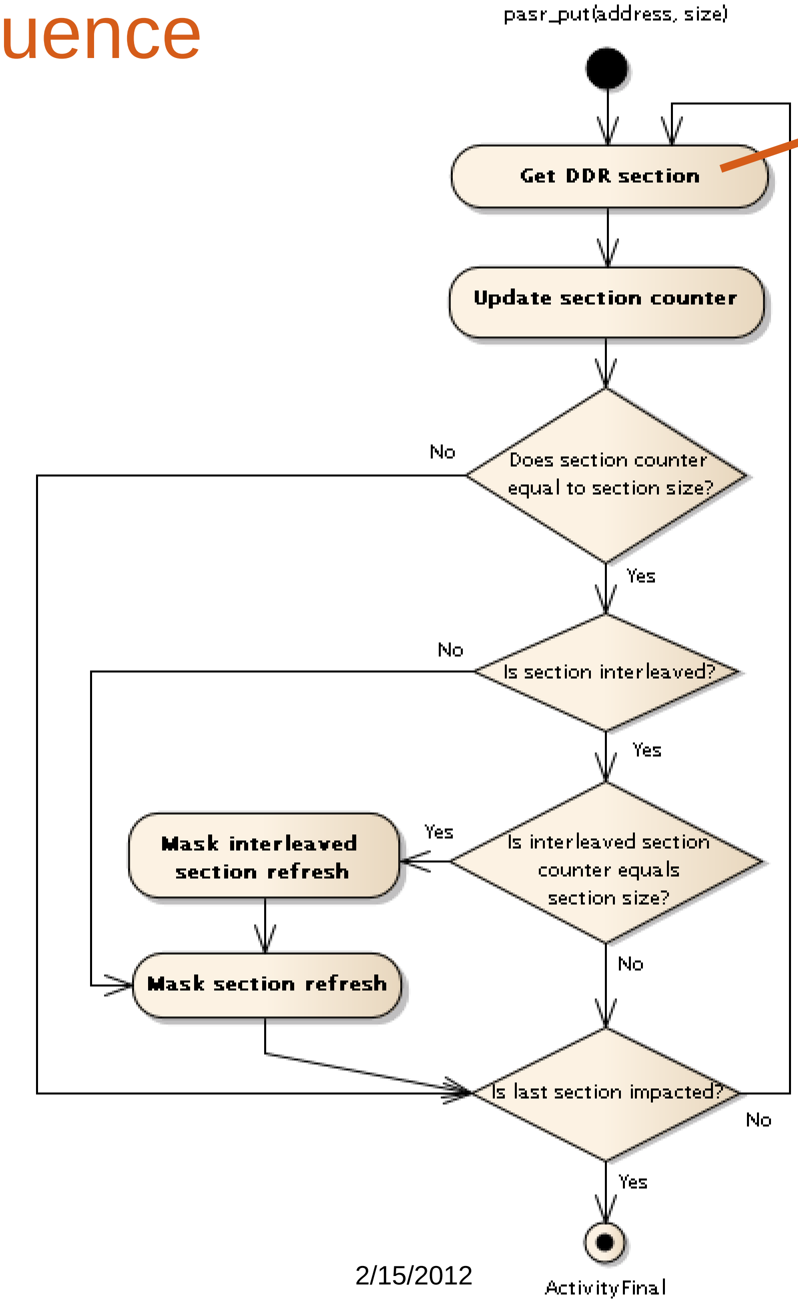
## The PASR Framework

# Put sequence


pasr_put(address, size)
Get DDR section
Update section counter
Does section counter equal to section size?
No
Yes
Is section interleaved?
No
Yes
Is interleaved section counter equals section size?
Yes
No
Mask interleaved section refresh
Mask section refresh
Is last section impacted?
No
Yes
ActivityFinal

## The PASR Framework

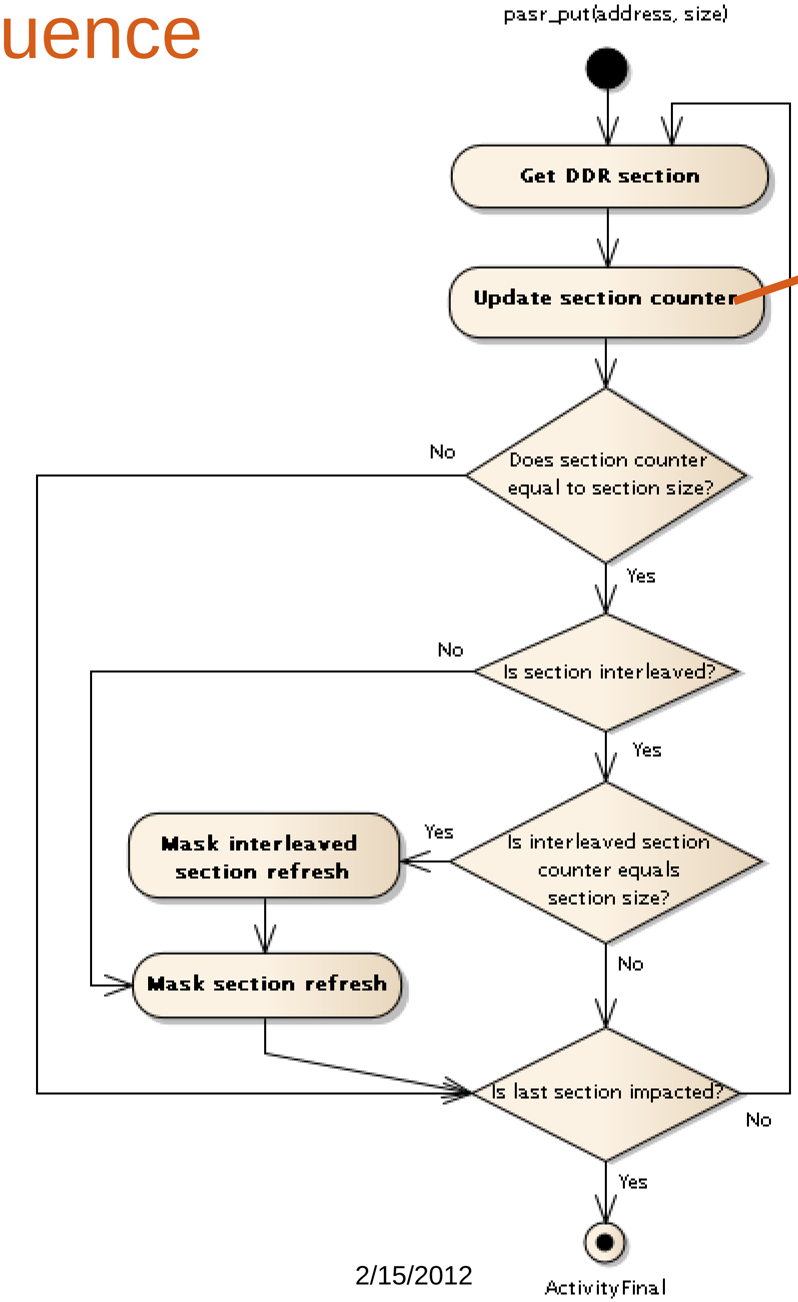
# Put sequence


pasr_put(address, size)
Get DDR section
Update section counter
Does section counter equal to section size?
No
Yes
Is section interleaved?
No
Yes
Is interleaved section counter equals section size?
Yes
No
Mask interleaved section refresh
Mask section refresh
Is last section impacted?
No
Yes
ActivityFinal

Get DDR section

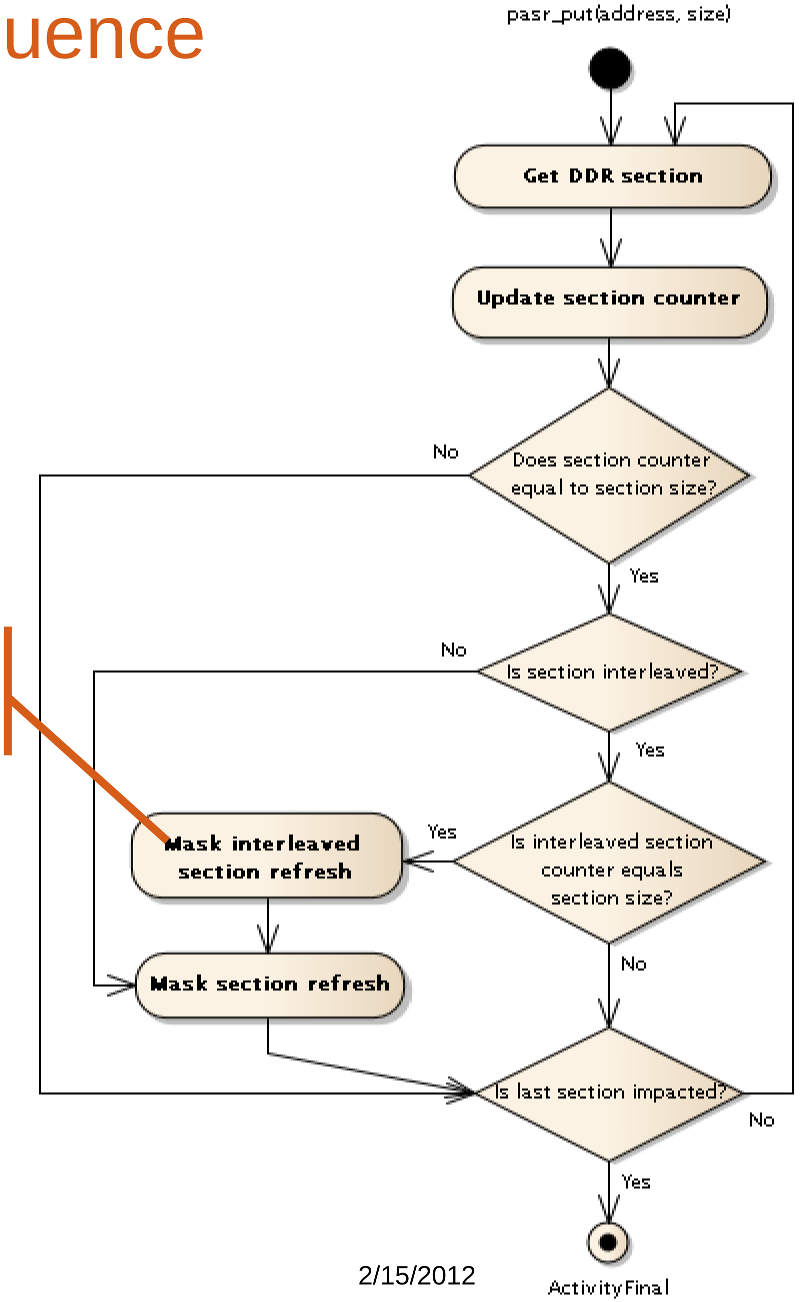
# The PASR Framework

# Put sequence


pasr_put(address, size)
Get DDR section
Update section counter
Does section counter equal to section size?
No
Yes
Is section interleaved?
No
Yes
Is interleaved section counter equals section size?
Yes
No
Mask interleaved section refresh
Mask section refresh
Is last section impacted?
No
Yes
ActivityFinal

Update section counter

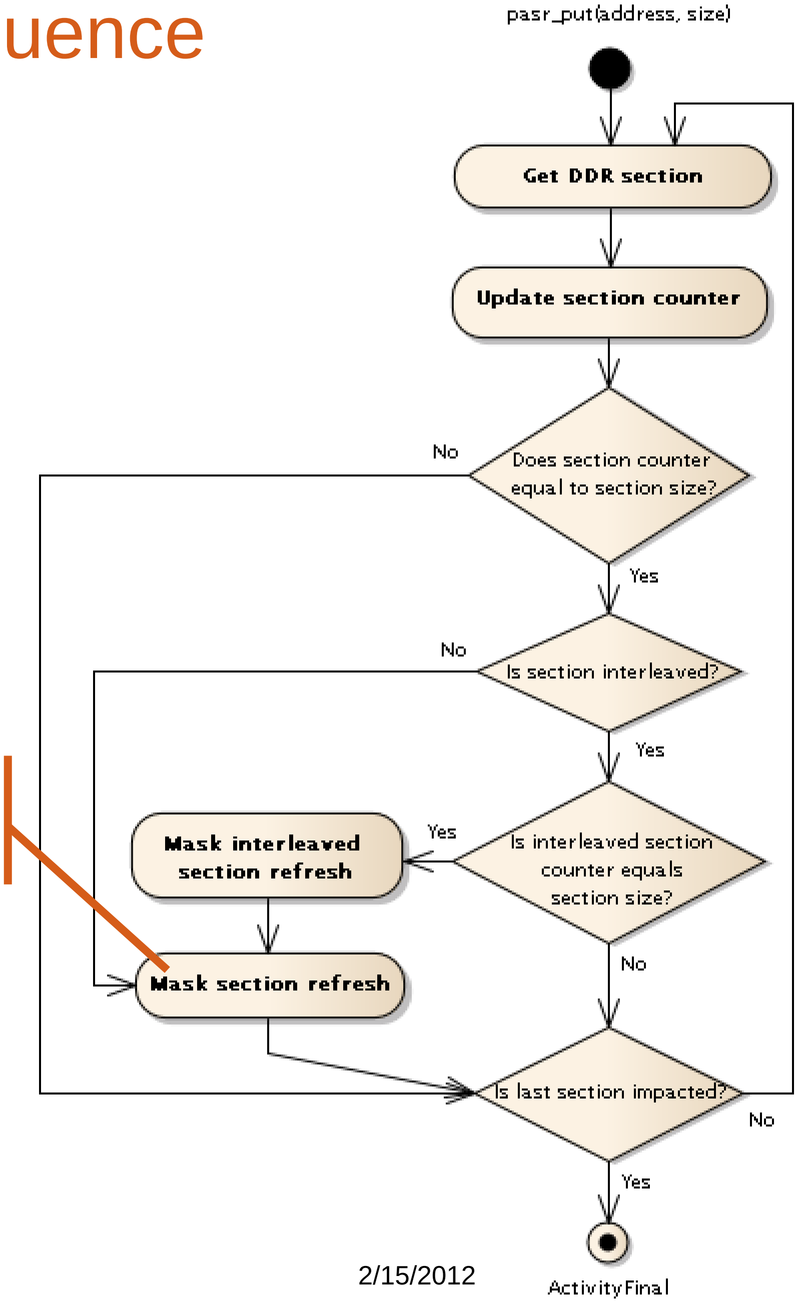
# Put sequence


pasr_put(address, size)
Get DDR section
Update section counter
Does section counter equal to section size?
No
Yes
Is section interleaved?
No
Yes
Is interleaved section counter equals section size?
Yes
No
Mask interleaved section refresh
Mask section refresh
Is last section impacted?
No
Yes
ActivityFinal

Mask interleaved section refresh

## The PASR Framework

# Put sequence


pasr_put(address, size)
Get DDR section
Update section counter
No
Does section counter equal to section size?
Yes
No
Is section interleaved?
Yes
Mask interleaved section refresh
Yes
Is interleaved section counter equals section size?
Mask section refresh
No
Is last section impacted?
No
Yes
ActivityFinal

Mask section refresh

# Content

- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

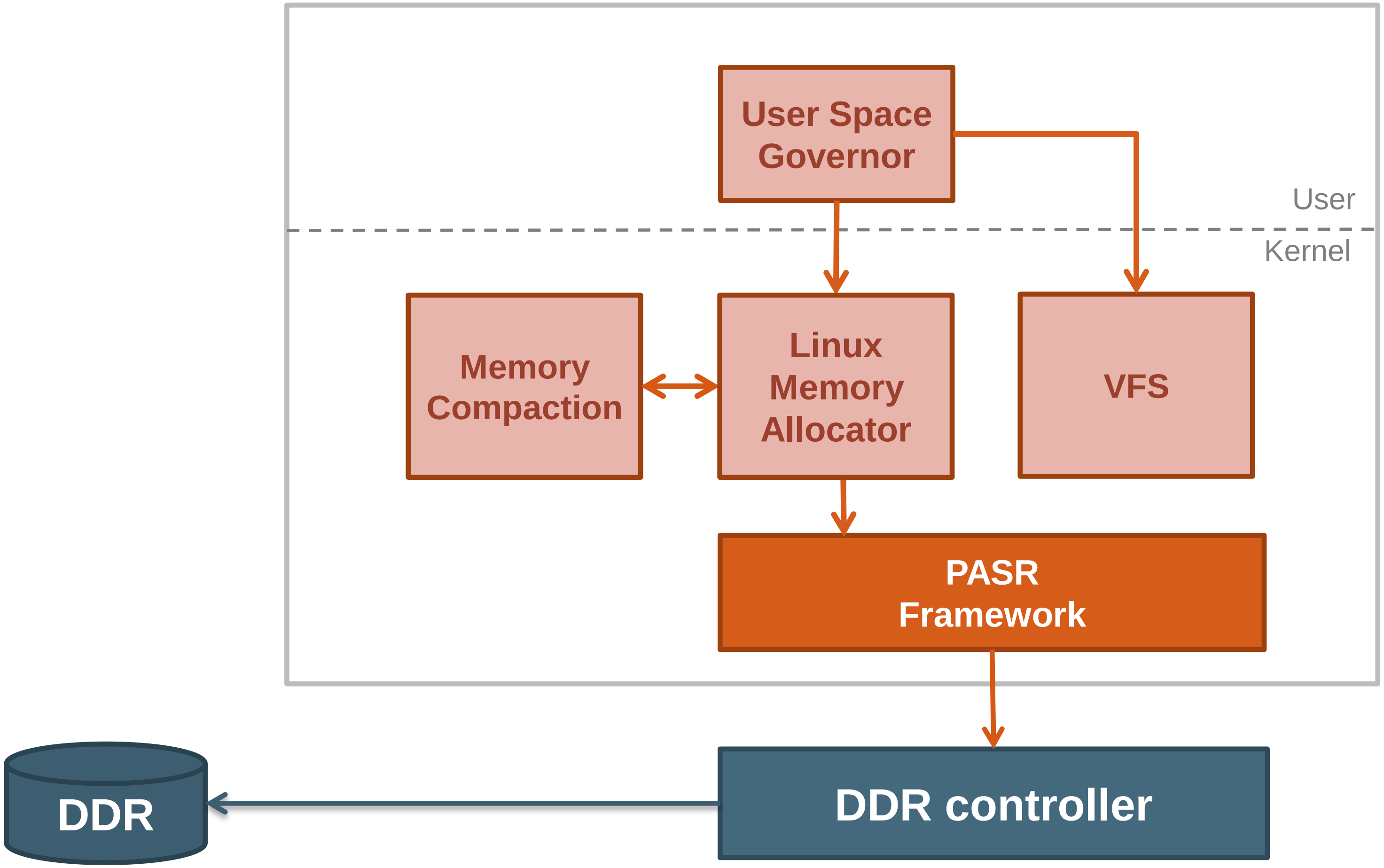
## How to use PASR framework?

# First approach

- Concern: Have as much as possible a “simple solution”
- Minimize kernel modifications
- Based on current kernel services:
  - Linux Memory Allocator
  - Memory compaction

How to use PASR framework?

# First approach - Overview



How to use PASR framework?

# Linux allocator specific optimization

- Buddy allocator
- Notification at page allocation/free level too heavy
- Useless regarding Buddy allocator principles!

→ Notifications only on “MAX_ORDER” pagebloc

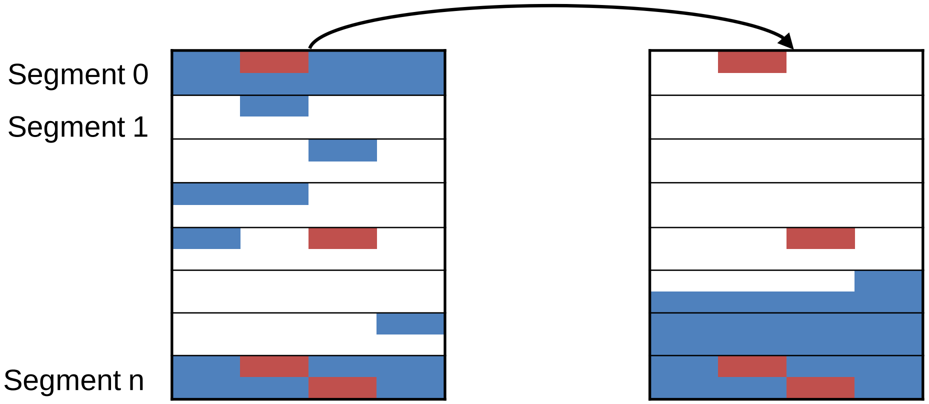
→ Remove from free list → call pasr_kget

→ Add in free list → call pasr_kput

## How to use PASR framework?

# Memory Compaction

- Based on standard Memory Compaction feature
- Defragment memory
- Optimization possible for higher PASR efficiency



**Before compaction**
**1 segment in PASR**

**After "normal" compaction**
**3 segments in PASR**

🟦 Movable page

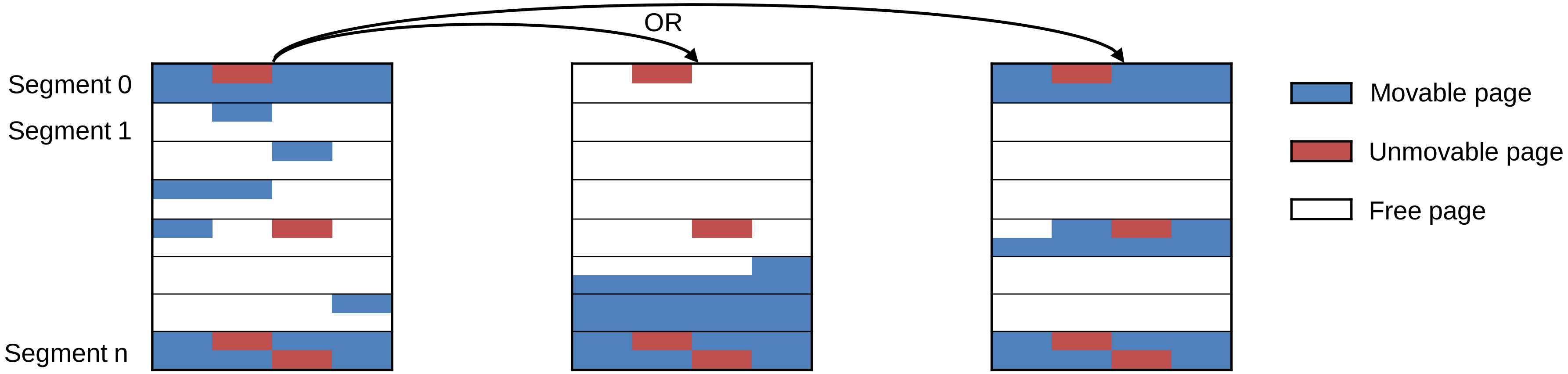
🟥 Unmovable page

□ Free page

## How to use PASR framework?

# Memory Compaction

- Based on standard Memory Compaction feature
- Defragment memory
- Optimization possible for higher PASR efficiency



**Before compaction**
**1 segment in PASR**

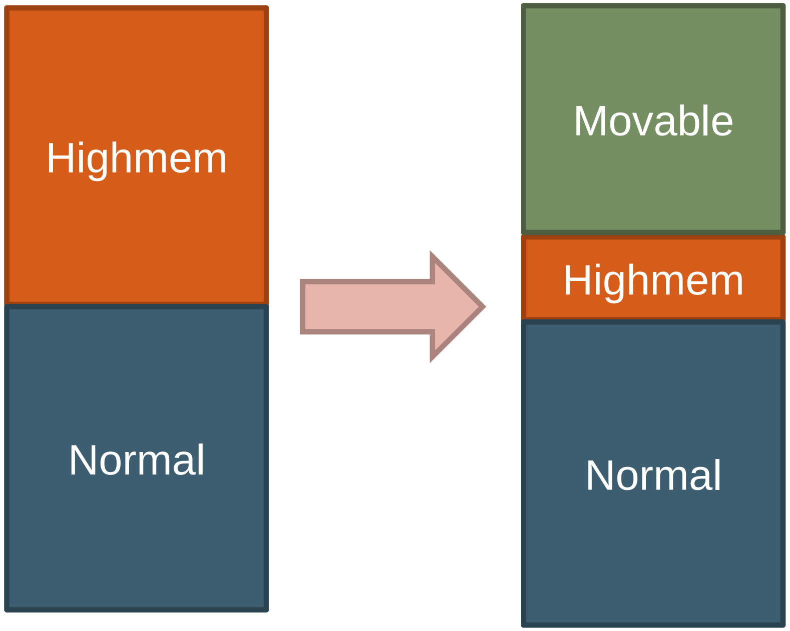
**After “normal” compaction**
**3 segments in PASR**

**After “PASR-Aware” compaction**
**5 segments in PASR**

How to use PASR framework?

# Movable Zone

- Unmovable allocations possible in Highmem zone
- Movable zone → only Movable pages allocation
- Not implemented in mainline ARM Kernel
- Improve defragmentation



How to use PASR framework?

# Carveout-style allocator

- 2 different cases:
- Dedicated chunk of memory →apply  approach 1
  - On Allocation: call pasr_get
  - On Free: call pasr_put
- Based on CMA → no impact
  - Linux Allocator handles PASR sequence
  - When allocator resquest buffer to CMA, pages are removed from Linux allocator free list
  - When buffers are no more used, they return back to Linux allocator thanks to CMA.

How to use PASR framework?

# Current Results

- Approach 1 has been integrated on Novathor L9540 platform
  - 4x4Gb DDR fully interleaved
  - Kernel 3.0
  - Android ICS
  - No compaction

- After platform start-up (Android Idle screen):
  - More than 1.2GB of free memory
  - 10 sections (over 32) masked

  → Results are promising

How to use PASR framework?

# First approach status

+ “Make It Simple” solution

+ Based on existing development

+ Easy to put in place

+ The complete memory always available from system point of view

- Impact in Linux Buddy Allocator
  - Request hook insertion which can slow down the page allocation
- No memory pressure control

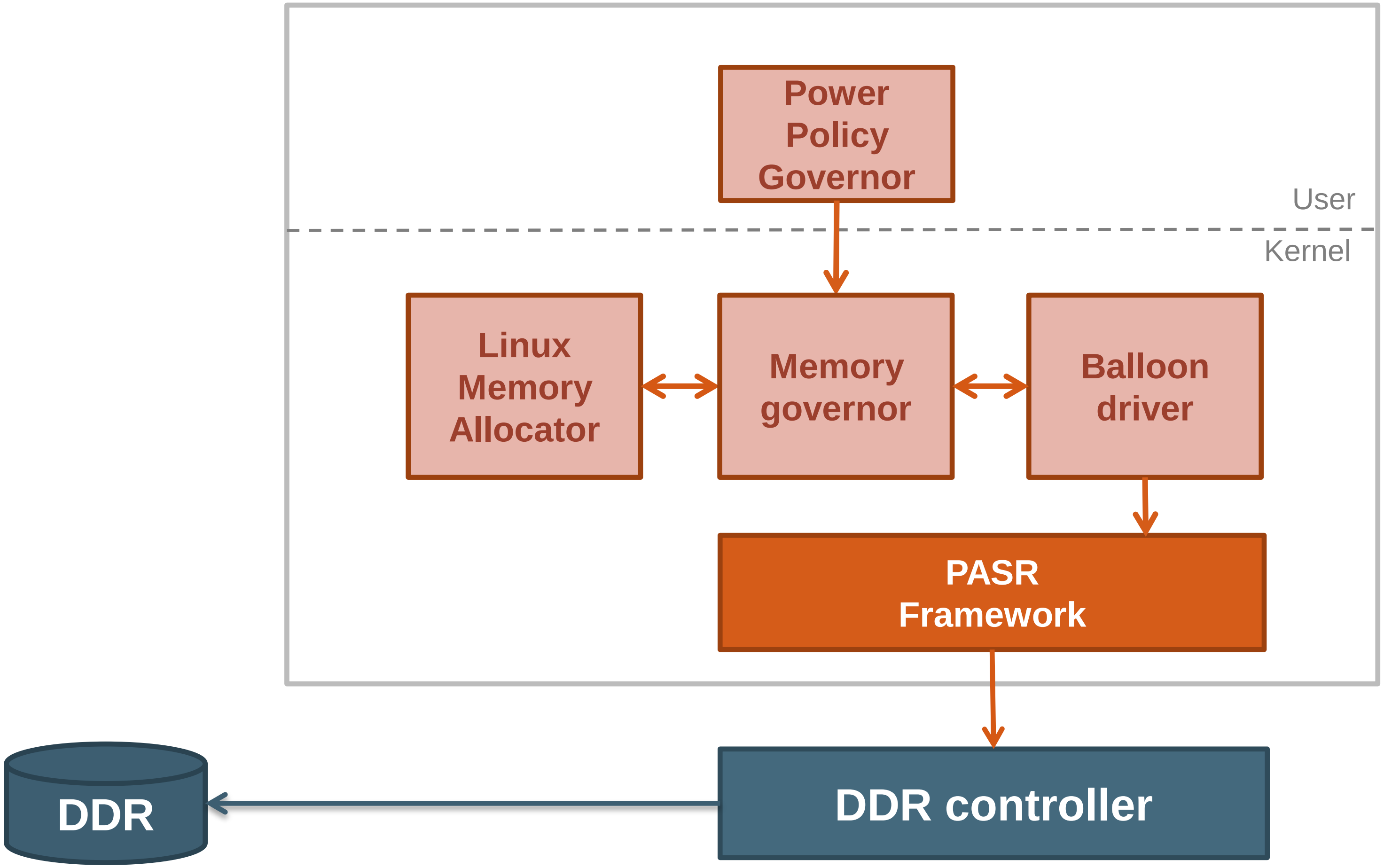
## How to use PASR framework?

# Second approach

- Based on a Balloon driver
  - Inflating: allocate large contiguous memory buffer
  - Deflating: release memory
  - Based on page reclaim/migration
  - Or CMA
  - Connected to PASR framework
- Associated to a memory governor
  - Memory pressure notification
  - Memory pressure strategy
  - Connected to user space for use case association

## How to use PASR framework?

# Second approach - Overview



How to use PASR framework?

# Second approach status

- Solution under investigation

+ Not intrusive in Linux memory allocator

+ Cost only when entering in idle or under memory pressure

+ Possible to control system memory pressure

+ Future proof

- More complex to set up

- No existing Balloon driver

- Memory pressure strategy definition

# Content

- Context
- DDR power management mechanisms
- Existing concepts
- The PASR Framework
- How to use PASR framework?
- Next steps

# Next Steps

- Patch series 2 release soon
- Integrate DPD support
- Add PASR debugfs
- Deliver PASR test suite
- Develop approach 2

# References

- PASR patch :
  - https://lkml.org/lkml/2012/1/30/146

- Movable zone support for ARM
  - git://codeaurora.org/kernel/msm.git
  - Branch msm-3.0

# QUESTIONS ?

# THANK YOU

**Contacts:**

**maxime.coquelin@stericsson.com**

**loic.pallardy@stericsson.com**