Supporting real-time hardware acceleration on dynamically reconfigurable SoC FPGAs

Marco Pagani

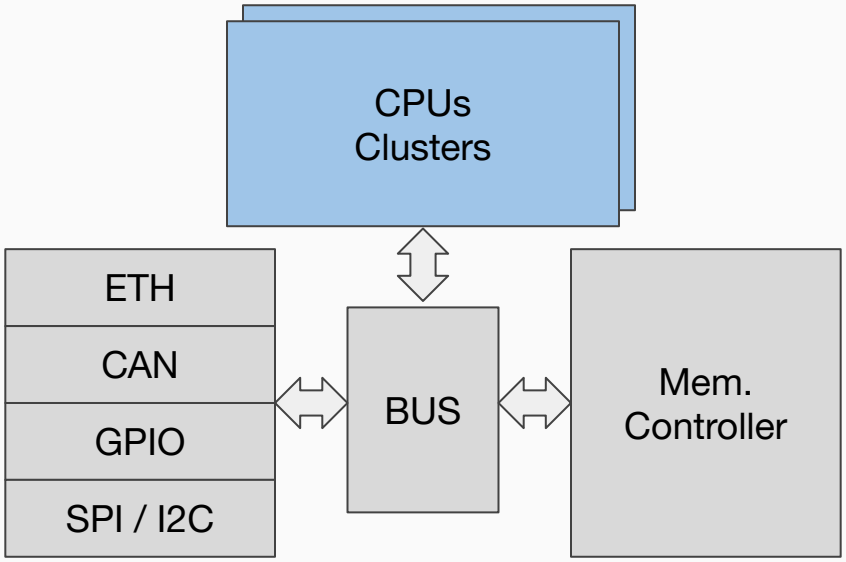
PhD candidate at
Scuola Superiore Sant'Anna and Université de Lille

Background: SoC-FPGAs platforms

Hardware-programmable platforms for embedded applications

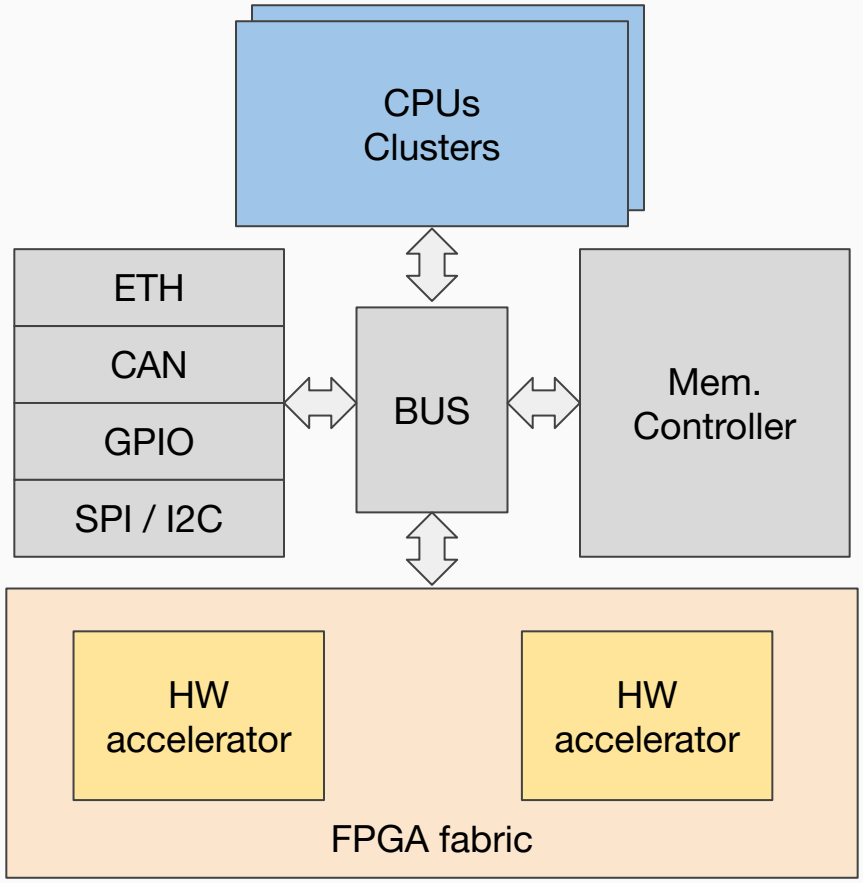
SoC-FPGAs platforms for embedded systems

- Traditional **SoCs** platforms typically includes one or more **GP cores clusters**, a **memory controller**, and a set of **peripherals**:



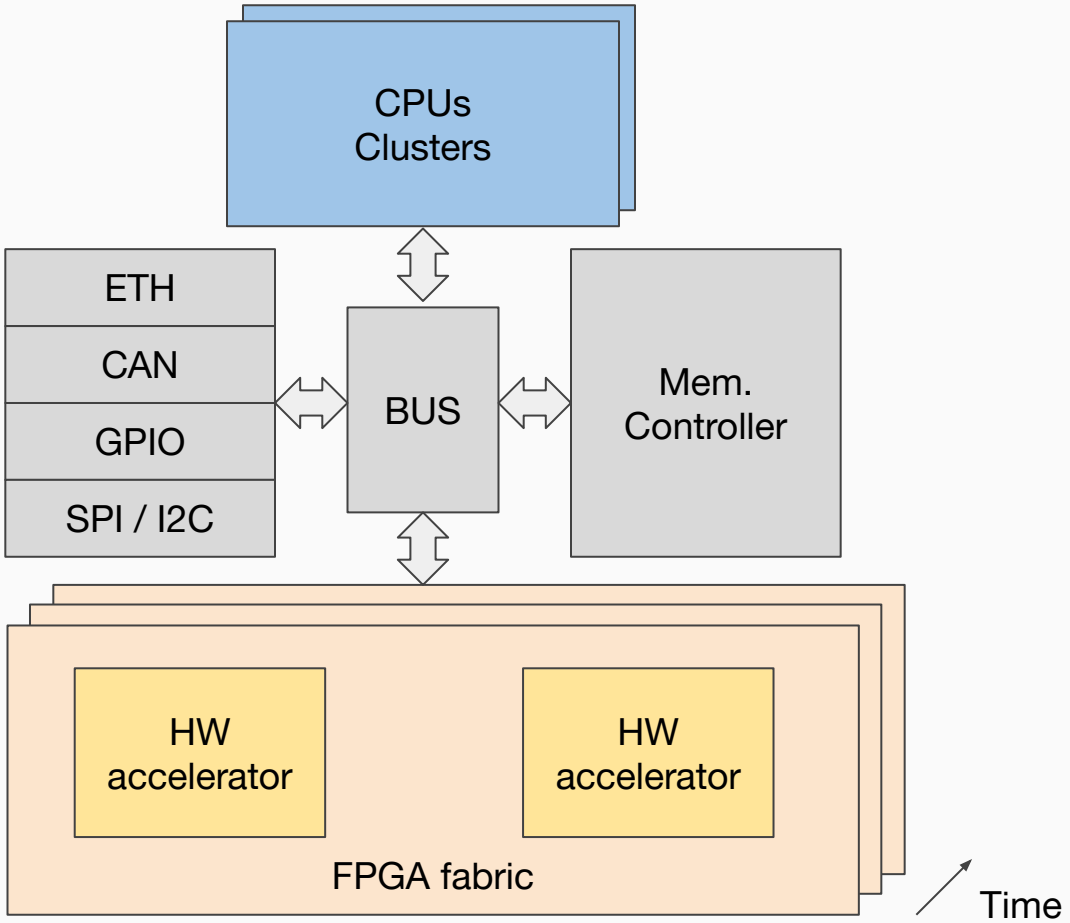
SoC-FPGAs platforms for embedded systems

- SoC-FPGAs platforms includes an **FPGA fabric** that can be used to ***extend*** the system with custom **HW-accelerators**.



SoC-FPGAs platforms for embedded systems

- With **dynamic partial reconfiguration** (DPR) it is possible to **dynamically** swap the set of **HW-accelerators** at run-time.



Why FPGA-based HW acceleration for real-time systems?

Pros

- **Very predictable** behavior. It is possible to **explicitly control** the HW-accelerators at **clock-level**;
- **High-performance** on SIMD / parallel operations with limited energy consumption;
- **Bus** and memory **contention** can be explicitly controlled at system level using **custom arbiters** and **bandwidth** shapers.

Why FPGA-based HW acceleration for real-time systems?

Pros

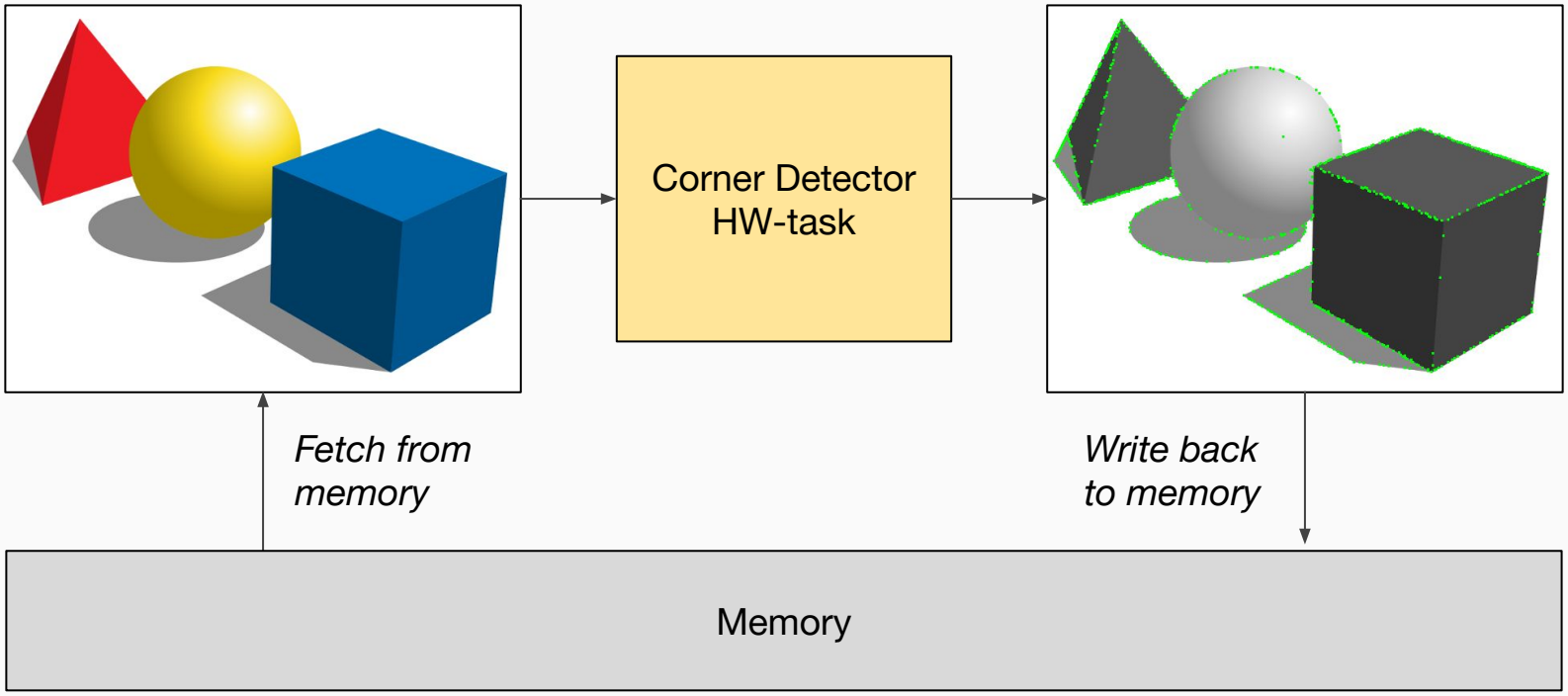
- **Very predictable** behavior. It is possible to **explicitly control** the HW-accelerators at **clock-level**;
- **High-performance** on SIMD / parallel operations with limited energy consumption;
- **Bus** and memory **contention** can be explicitly controlled at system level using **custom arbiters** and **bandwidth** shapers.

Cons

- Developing HW-accelerators for FPGAs is way more **time consuming** than developing software:
  - Even modern tools like **high-level synthesis** are **not so straightforward** to use;
  - **Less libraries** and development stacks are **available** with respect to other platforms (e.g., GPUs).

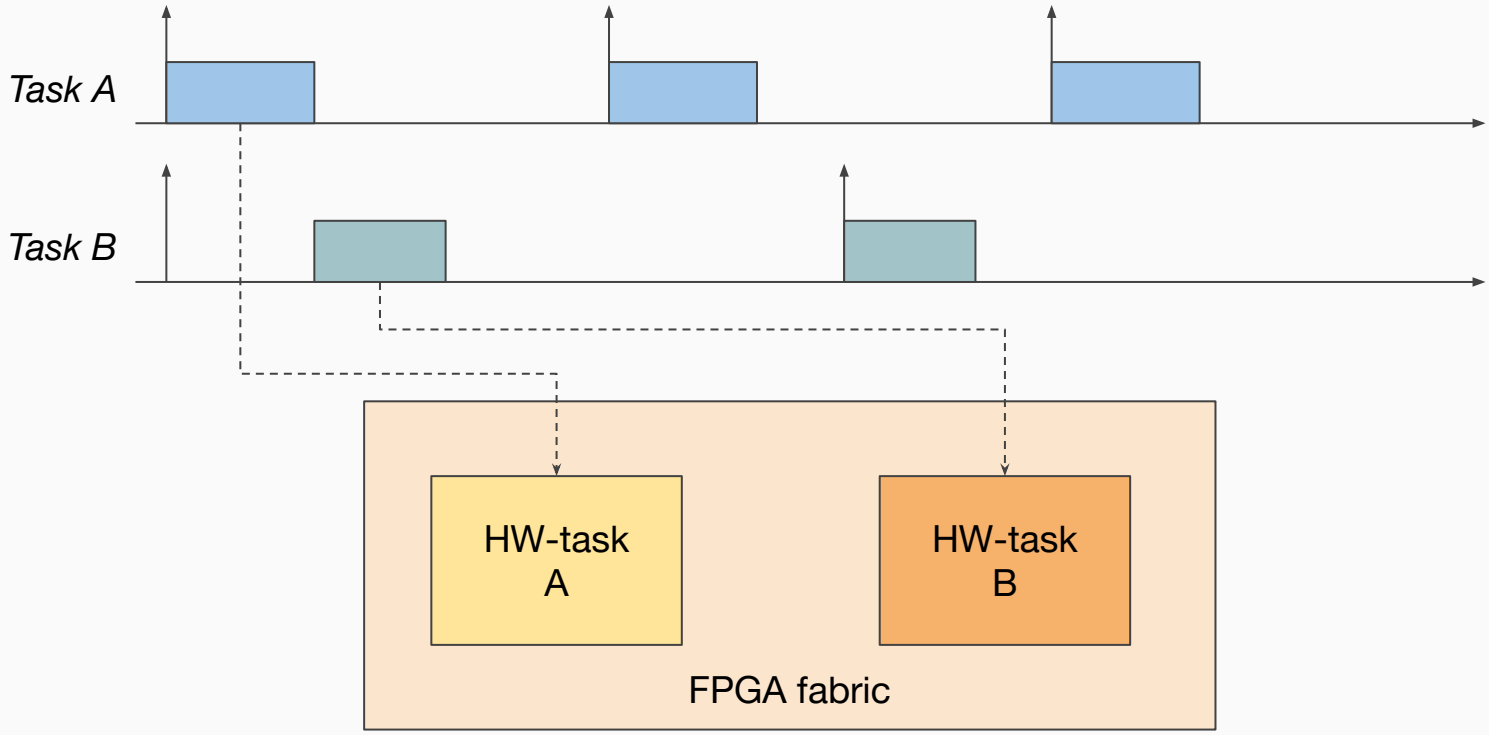
DMA / bus mastering accelerators

- **High-performances HW-accelerators** implement **bus mastering / DMA** to directly access data in the system memory;
- Let's consider **HW accelerators** performing the same **computational activity** (e.g. processing a frame) at each run (**HW-tasks**).



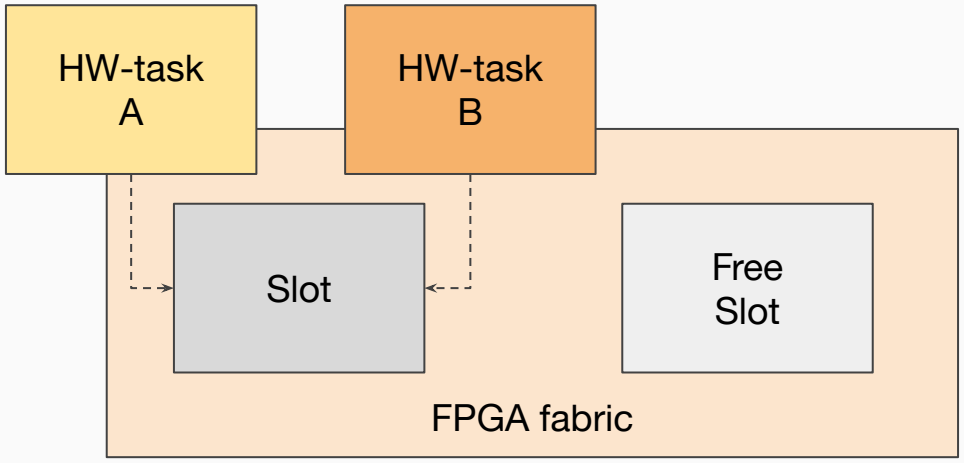
FPGA-based HW acceleration for real-time systems

- Many **real-time** systems use **periodic** or sporadic tasks:
  - Placing HW-accelerators **statically** may be inefficient;
  - FPGA's logic resources may **IDLE** most of the time resulting in **low utilization**.



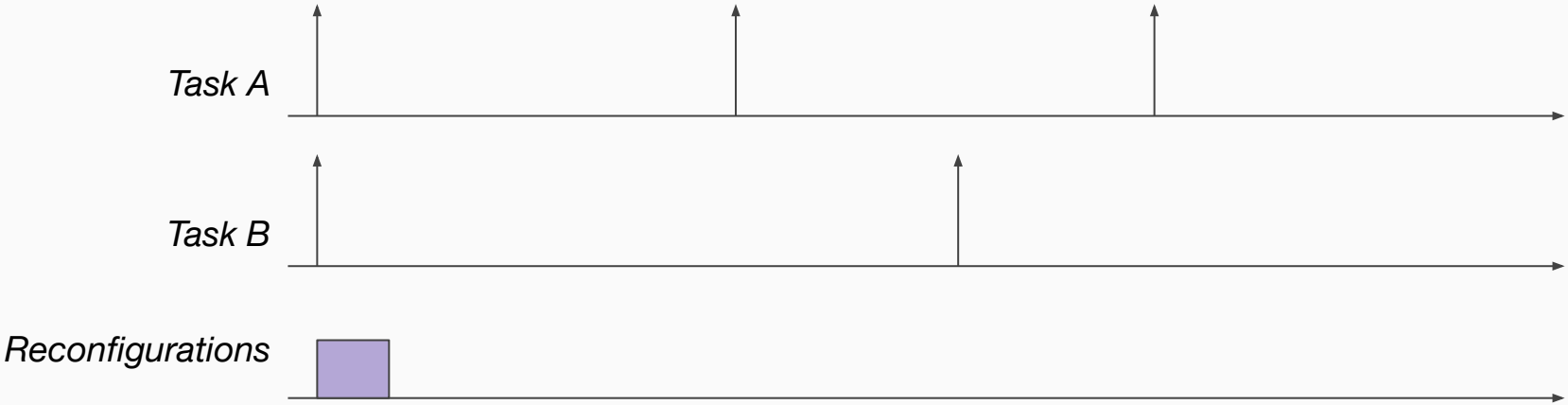
FPGA-based HW acceleration for real-time systems

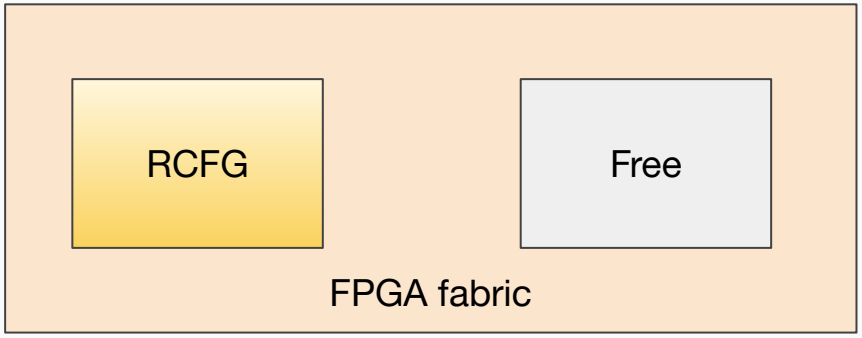
- With **DPR** it is possible to reconfigure only a **portion** of the **FPGA** while the **remainder** of the fabric continues to **operate**…
  - What if we **leverage DPR** to reconfigure the **HW-tasks** at **run-time** only when needed?



Leveraging DPR for real-time systems

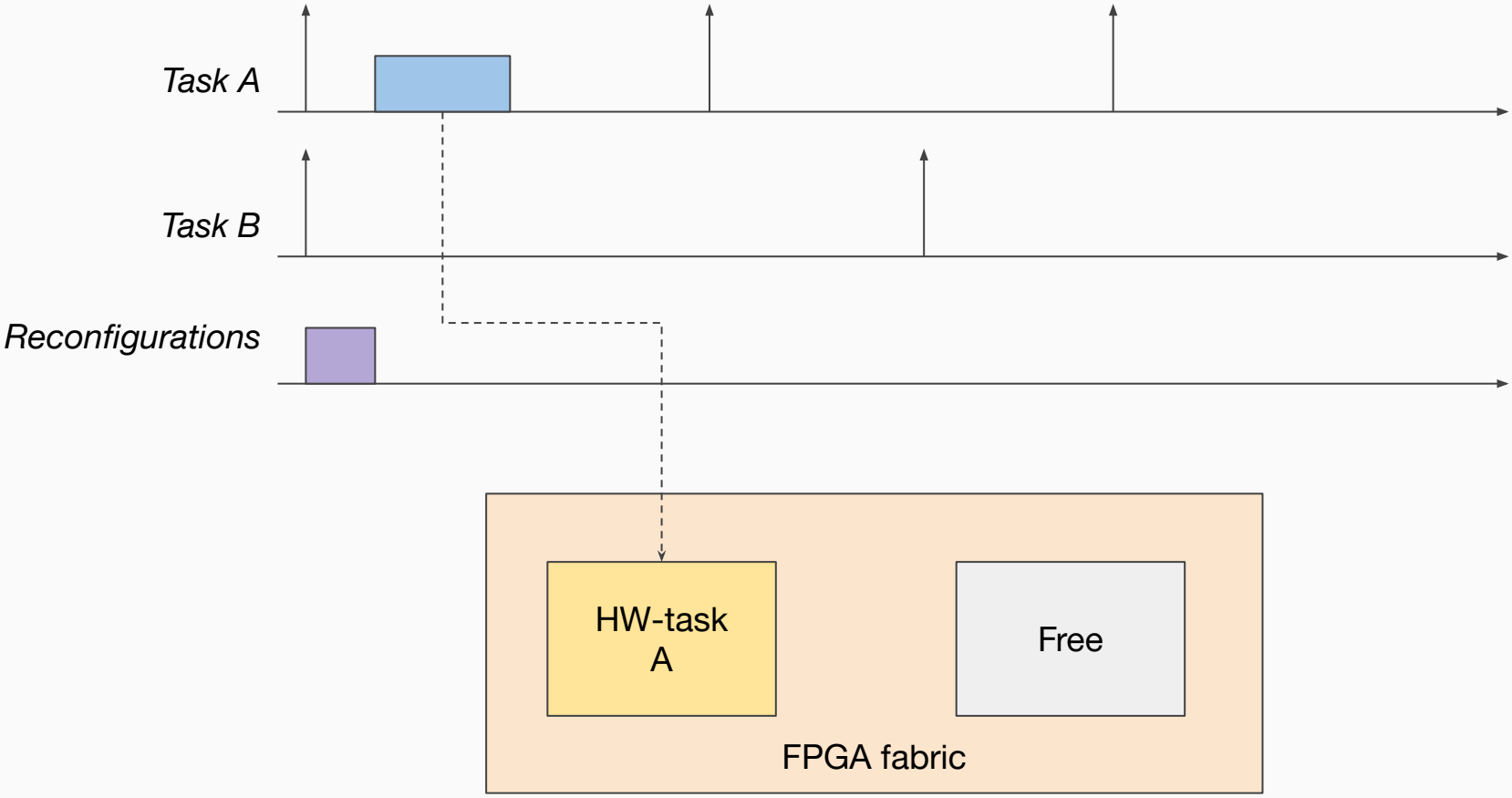
- Both **tasks A** and **B** are released and the **slot** is **reconfigured** for **HW-task A** utilized by **task A**





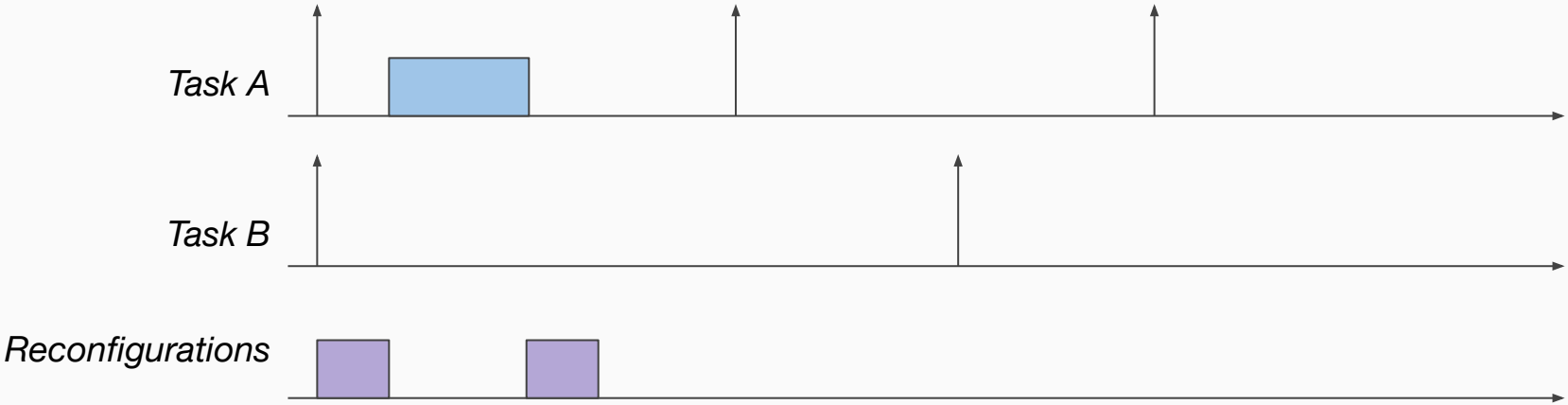
Leveraging DPR for real-time systems

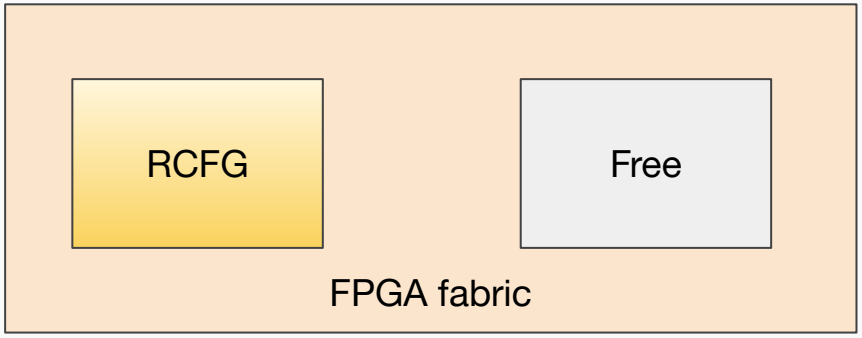
- **Reconfiguration** of **HW-task A** is completed and **task A** can **continue**.



Leveraging DPR for real-time systems

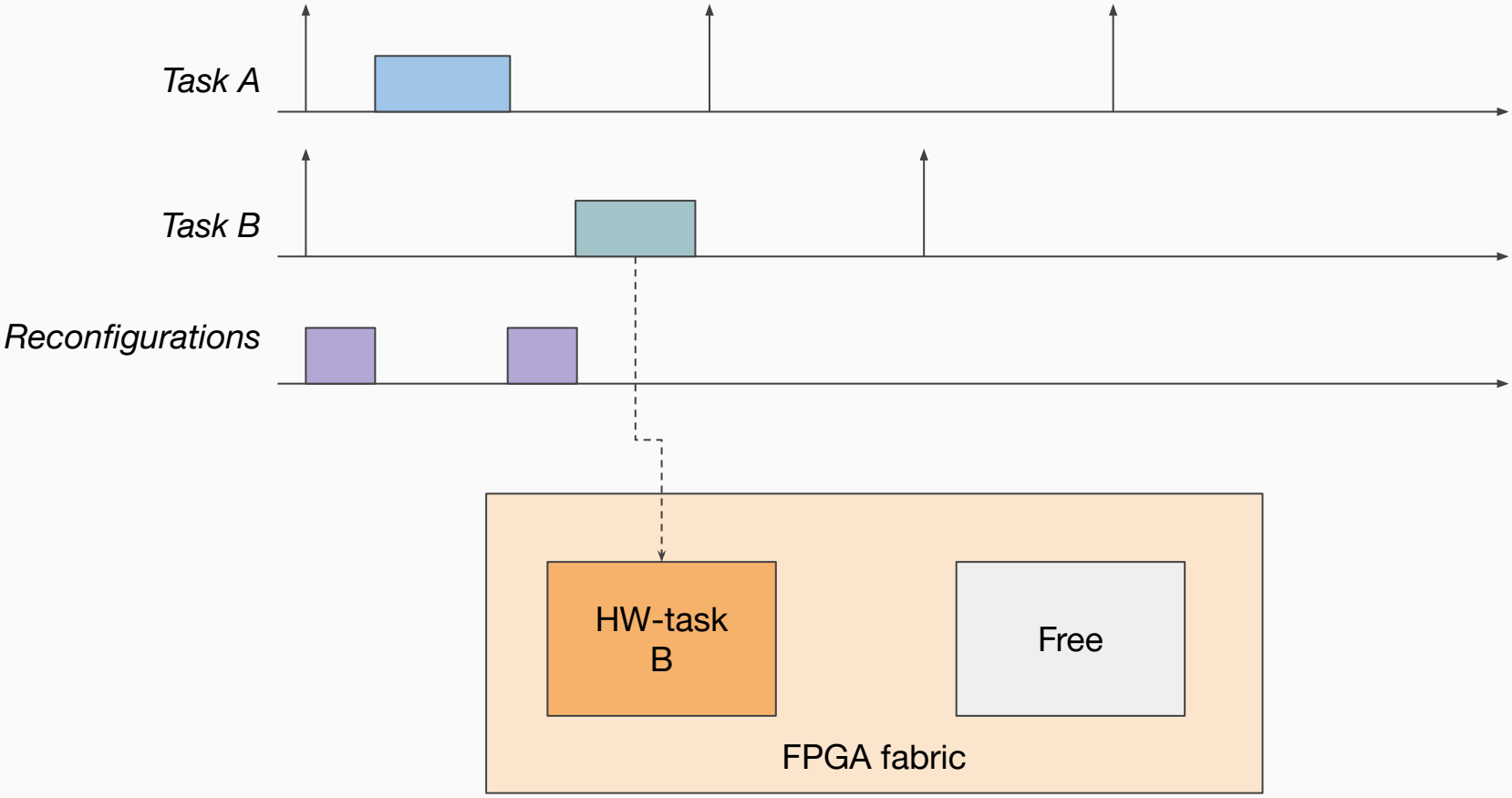
- **Task A completes** and the **slot** is **reconfigured** again for **HW-task B** required by **task B**.





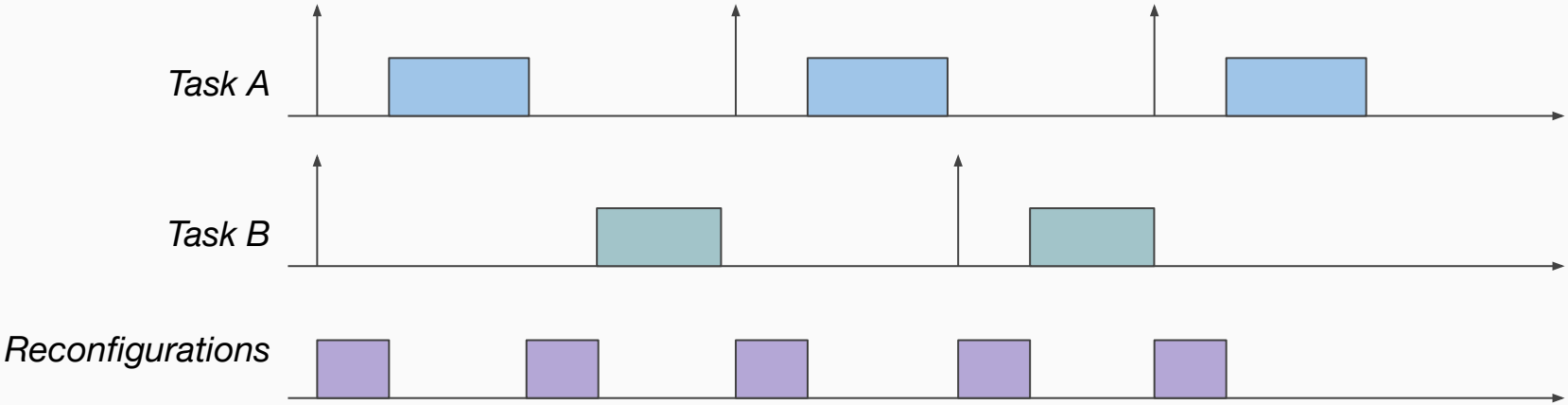
Leveraging DPR for real-time systems

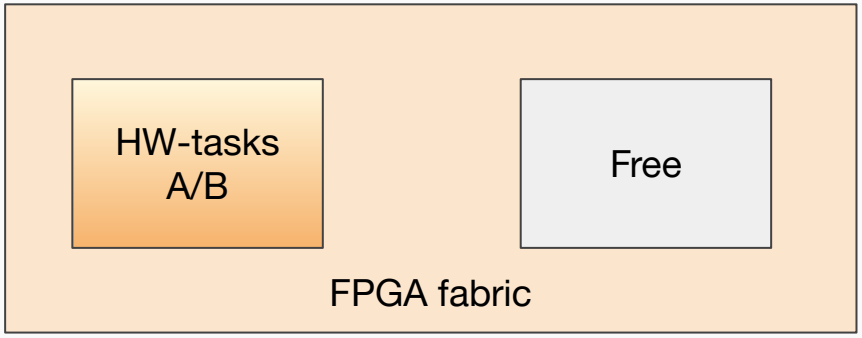
- **Reconfiguration** of **HW-task B** is **completed** and **task B** can **continue**



Leveraging DPR for real-time systems

- And the **cycle continues**...



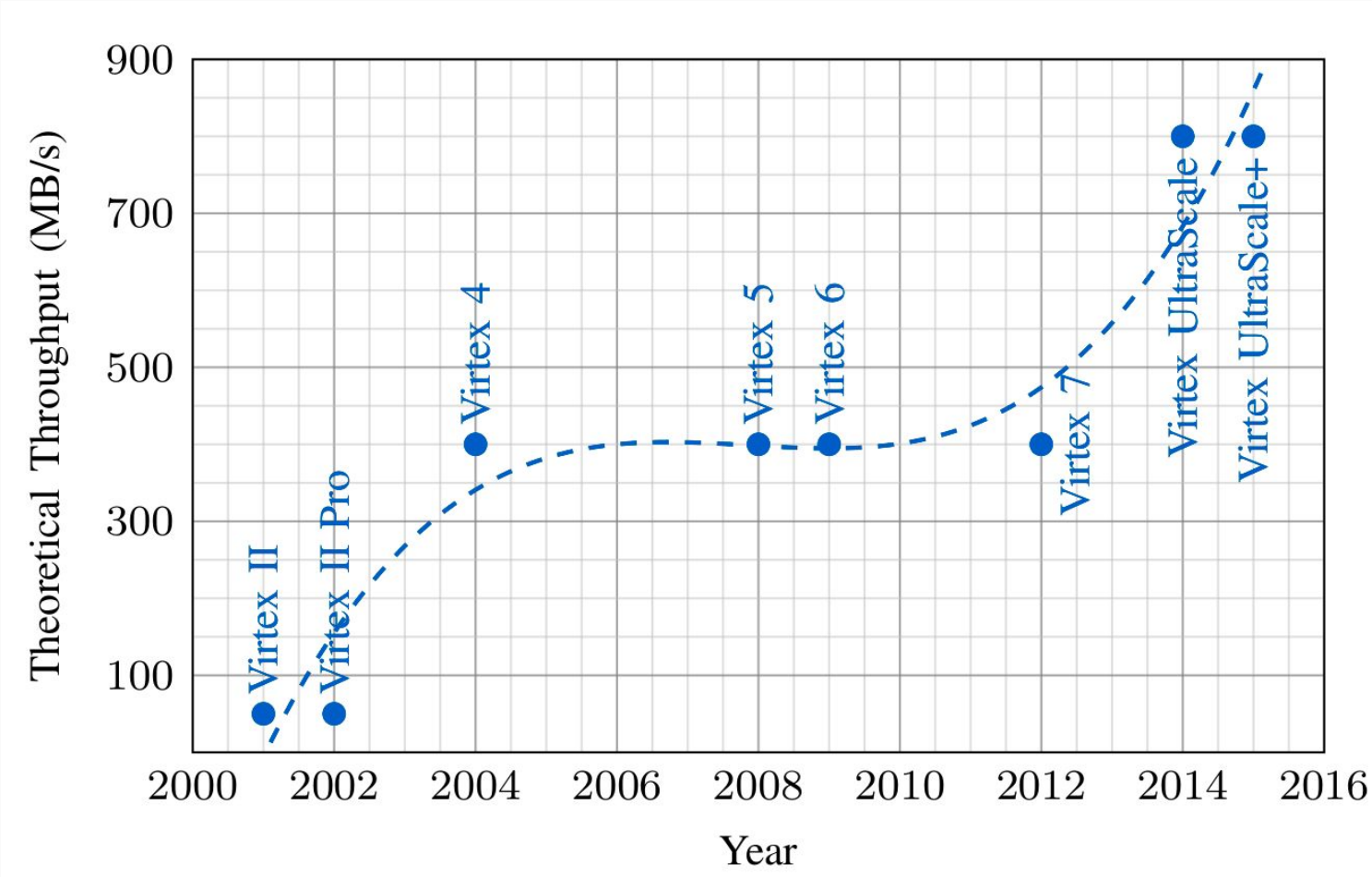


FPGA reconfiguration rate: the trend

- «*Well, this looks **nice in theory** but on **real-world FPGA** platforms the reconfiguration is **not fast enough** for this*»

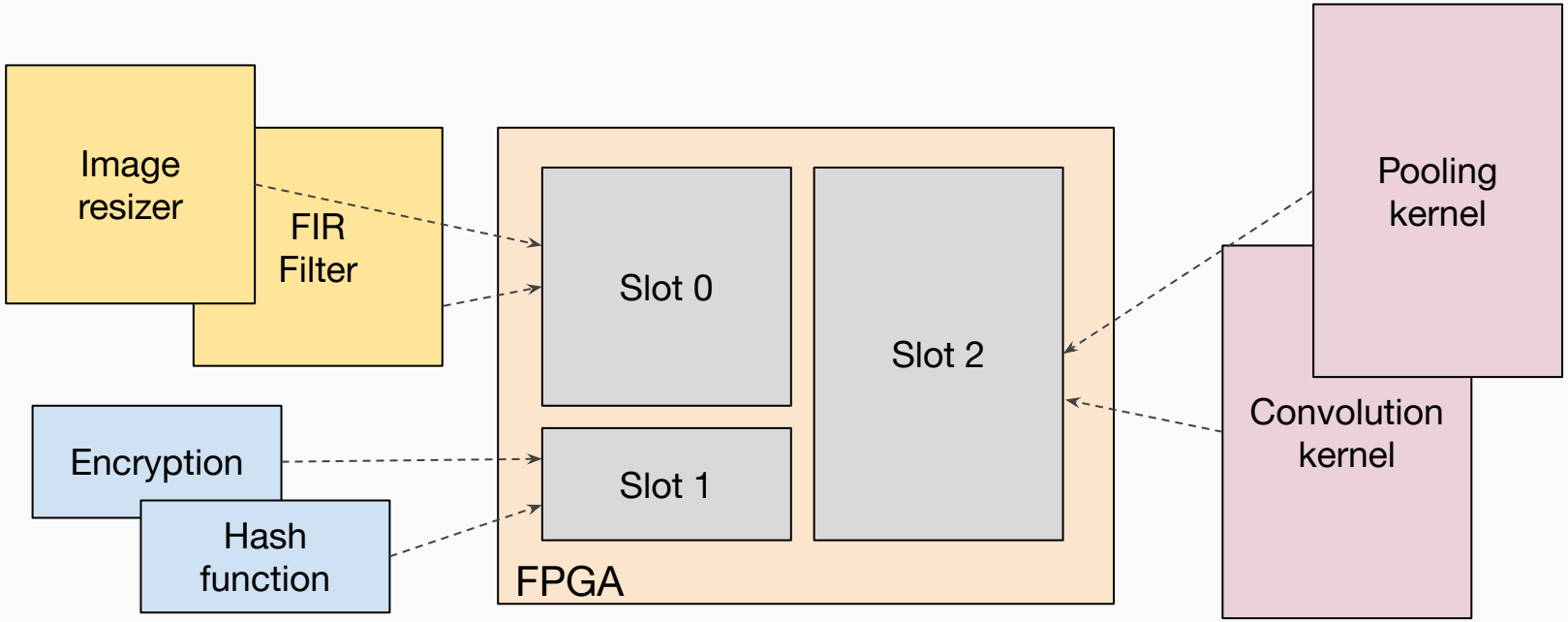
FPGA reconfiguration rate: the trend

- «*Well, this looks **nice in theory** but on **real-world FPGA** platforms the reconfiguration is **not fast enough** for this*»
  - Let's look at the **reconfiguration throughput** in the last years:



Leveraging DPR for real-time systems

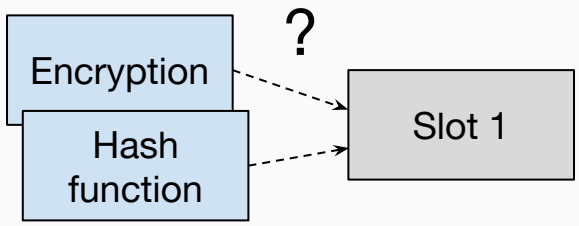
- **DPR** a **new scheduling dimension**!
  - Multiple HW-tasks can utilize the same logic resources in **time-sharing**;
  - Similar to **multitasking** but in the FPGA **area domain**;
  - Sort of **FPGA "virtualization"** since the FPGA can host more HW-tasks that what statically possible



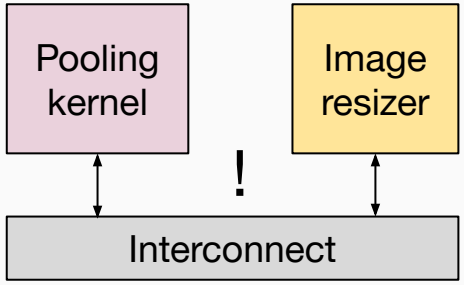
However, SoC FPGAs are complex platforms!

- So far so good, but in practice things are a bit more complex for real-world systems:

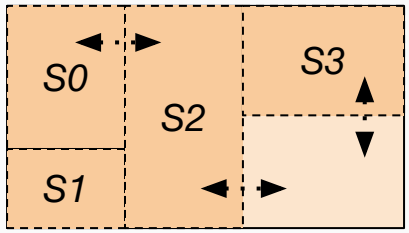
- How to **schedule reconfiguration** and **acceleration request** in order to achieve **bounded** response times?



- How to **manage bus** and **memory contention** ensuring **safe** and **predictable** hardware accelerators behaviour?



- How to efficiently **partition** the FPGA **fabric** to avoid wasting time and resources?

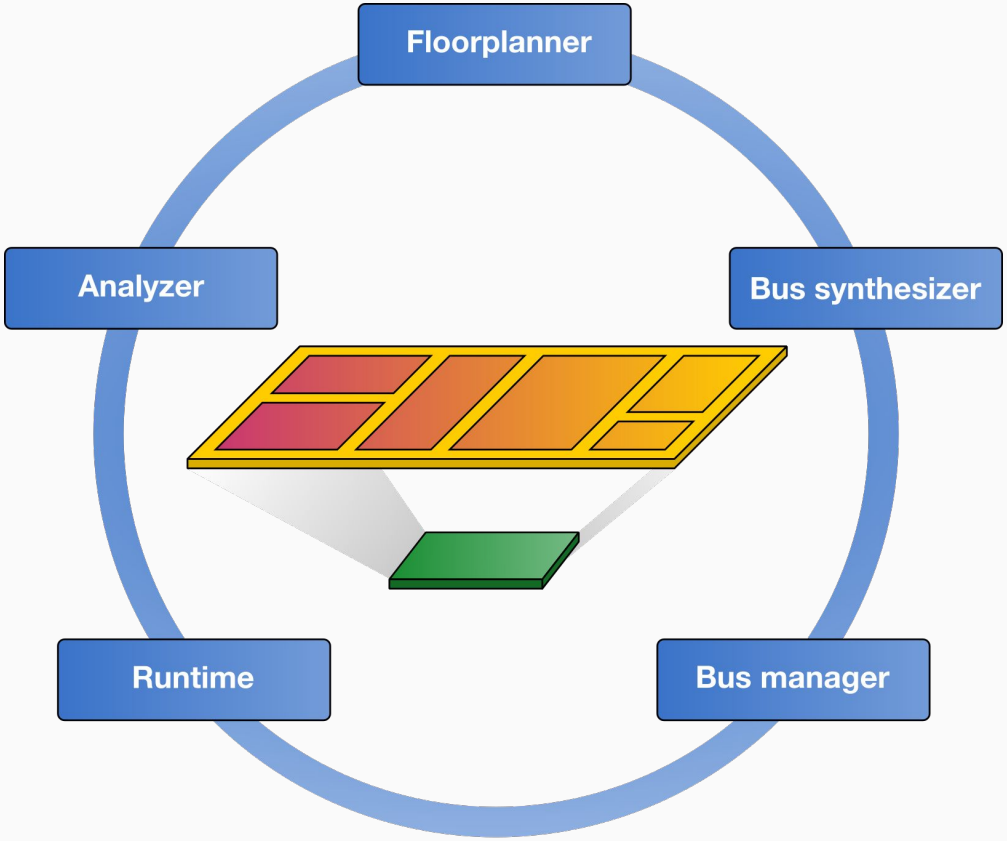


The FRED framework

A framework for leveraging dynamic partial reconfiguration and recurrent execution to predictably “virtualize” the FPGA fabric guaranteeing bounded delays by design.

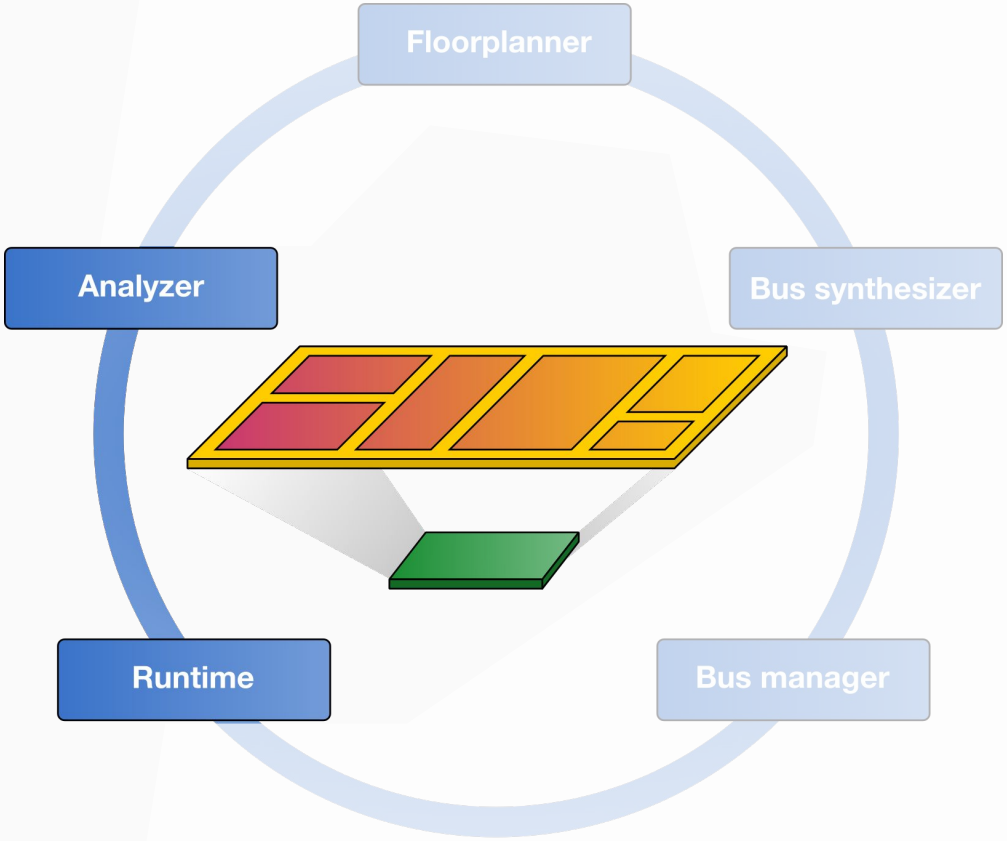
Overview

- **FRED** is a combination of several tools to tackle these issues:
  - **Platform model** + **real-time analysis** and **runtime** on **Linux**;
  - **Bus monitors** and **bandwidth shapers**;
  - **Automated** FPGA fabric **floorplanning**;



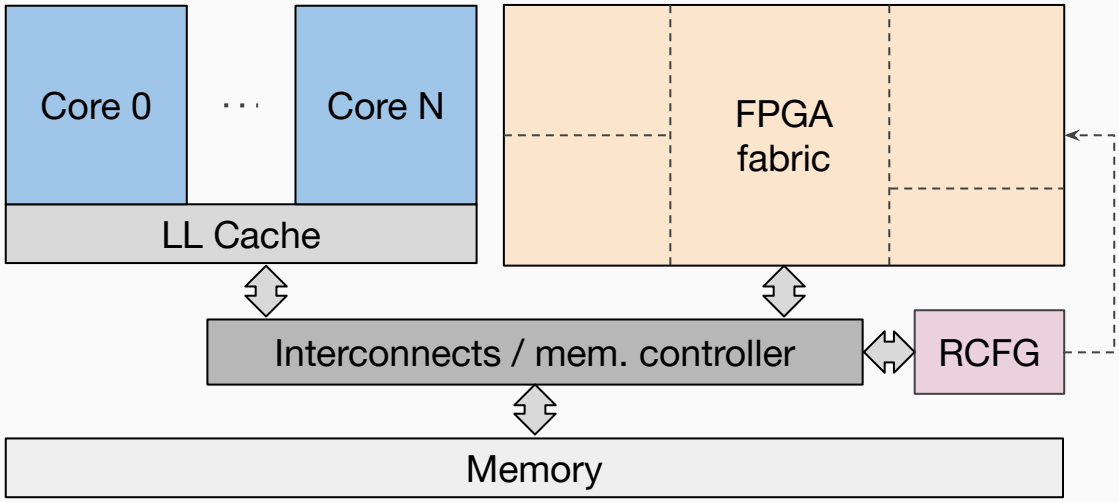
Overview

- This talk will focus on:
  - **Platform model** and **runtime** support on **Linux**;



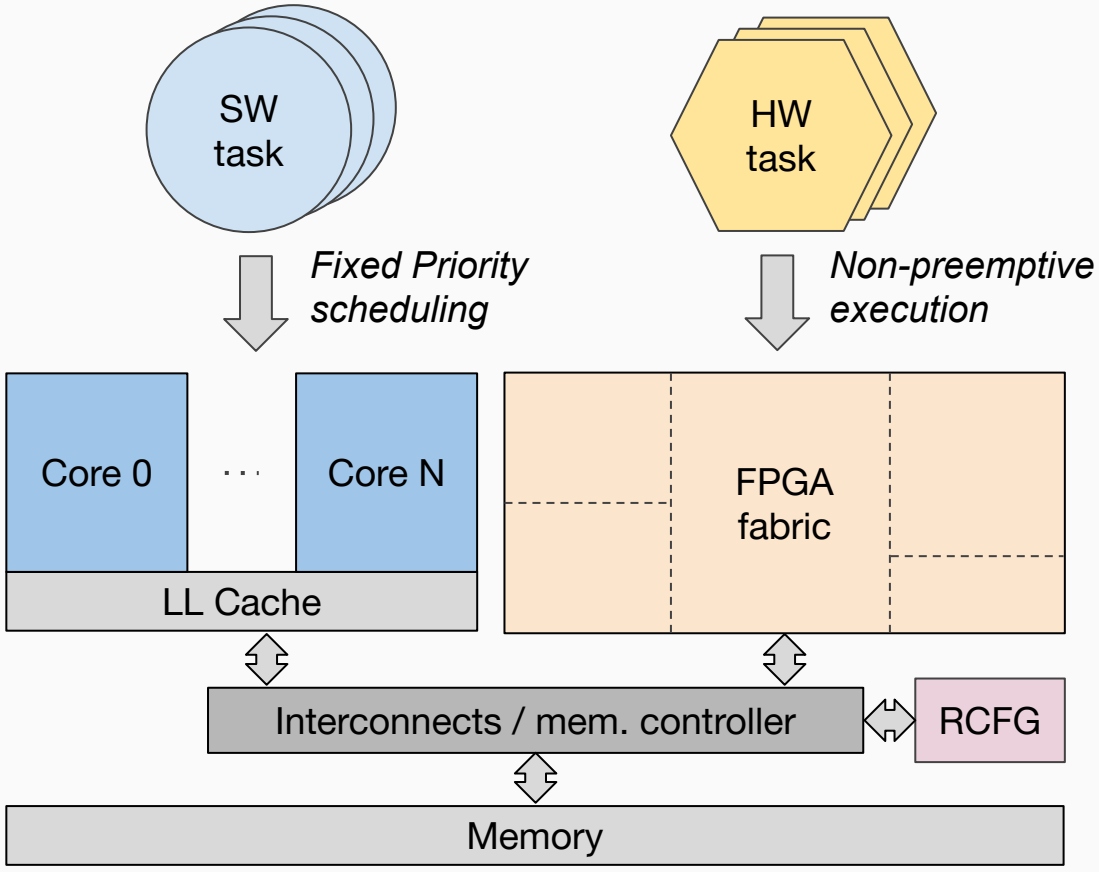
FRED runtime: platform model

- FRED's runtime **platform model** considers a system comprising:
  - One (or more) **GP cores**;
  - A dynamically reconfigurable **FPGA fabric**;
  - A **shared memory**;
  - An FPGA **reconfiguration DMA** engine.
    - One reconfiguration at a time.



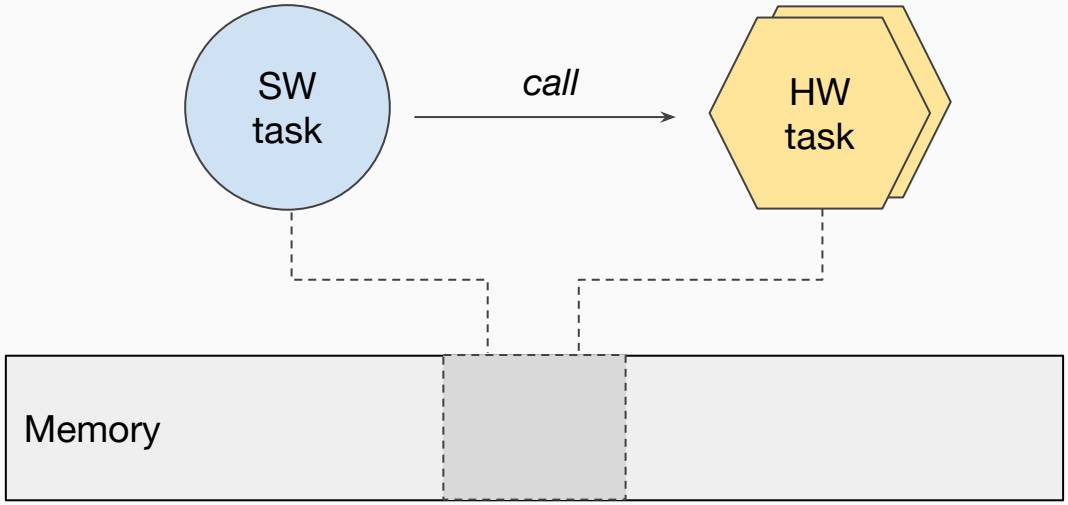
FRED runtime: platform model

- On top of this, **two kinds** of computational **activities**:
  - **SW-tasks**: **periodic/sporadic** task;
  - **HW-tasks**: instances of hardware accelerators.



FRED runtime: platform model

- Each **job** of a **SW-task** can **call** one (or more) **HW-task** to **accelerate** its execution:
  - The **HW-task(s)** will be **reconfigured** on the **FPGA fabric**.
  - **Data** are **exchanged** through shared **memory buffers**;

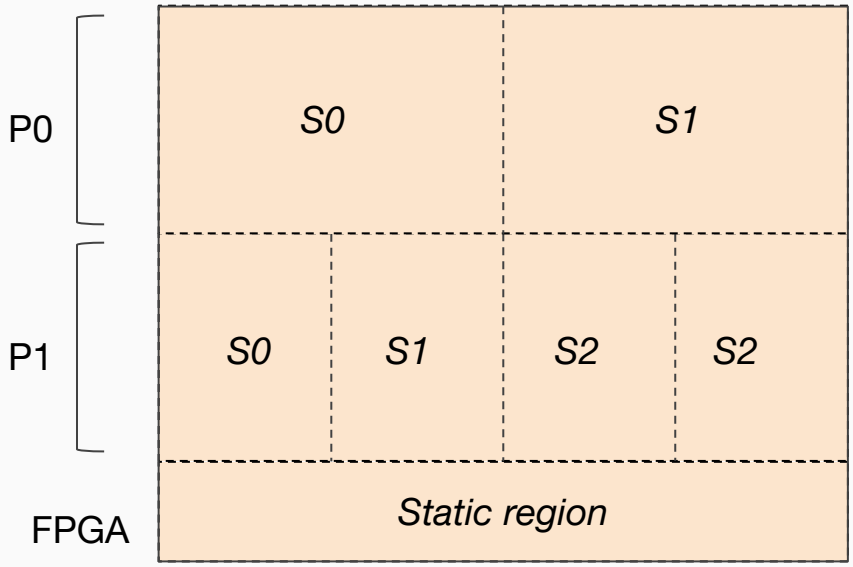


FRED runtime: FPGA partitioning

- **FRED** uses a slotted approach for **FPGA**:
  - The **FPGA** is divided into **partitions**;

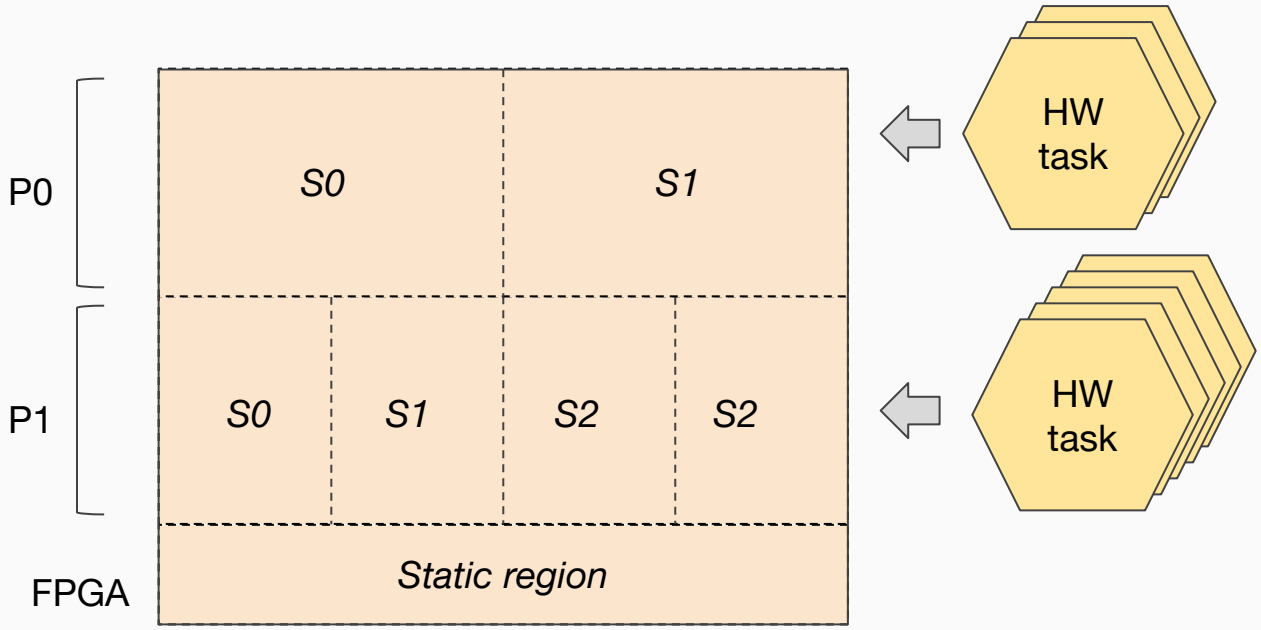
FRED runtime: FPGA partitioning

- **FRED** uses a slotted approach for **FPGA**:
  - The **FPGA** is divided into **partitions**;
  - Each **partitions** further is divided into **slots** of equal sizes;



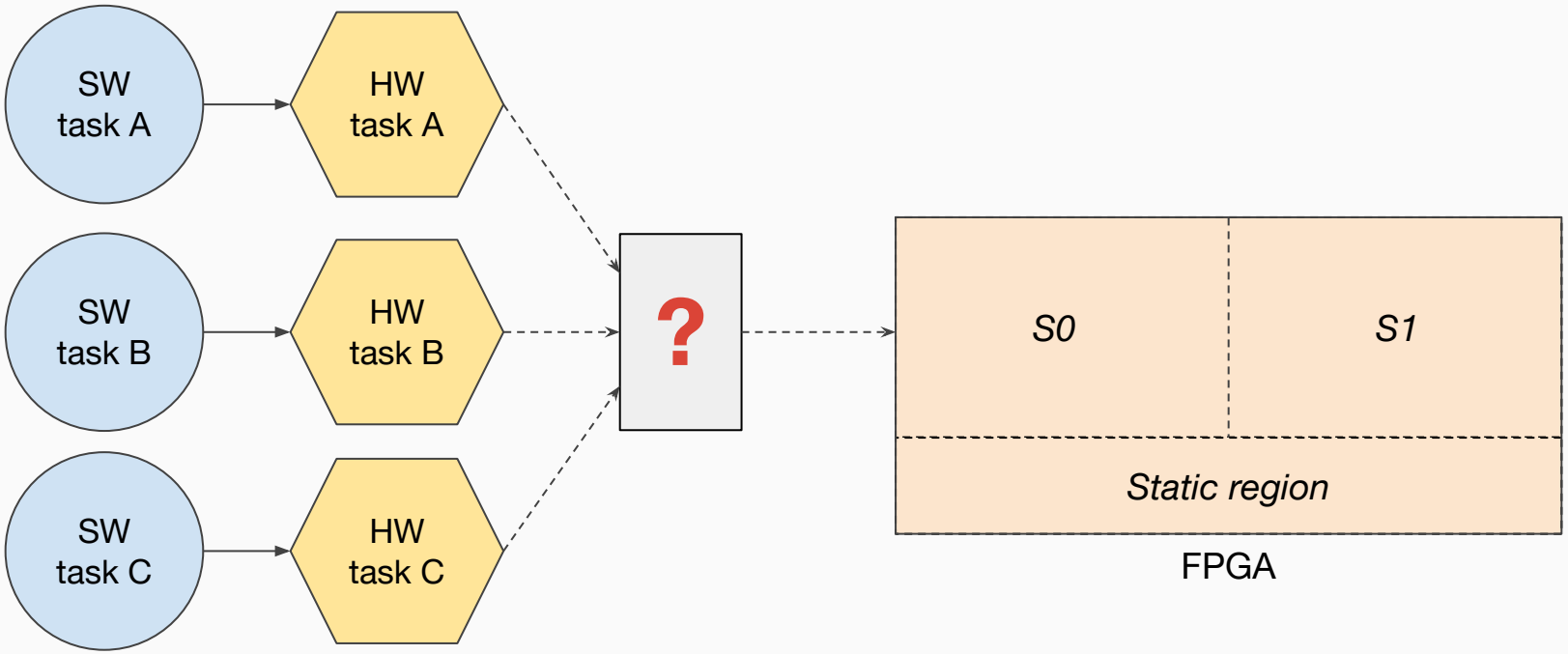
FRED runtime: FPGA partitioning

- **FRED** uses a slotted approach for **FPGA**:
  - The **FPGA** is divided into **partitions**;
  - Each **partitions** further is divided into **slots** of equal sizes;
  - **HW-tasks** are **associated** to a **single partition** (affinity).
    - Can be reconfigured in **any slot** of the partition.



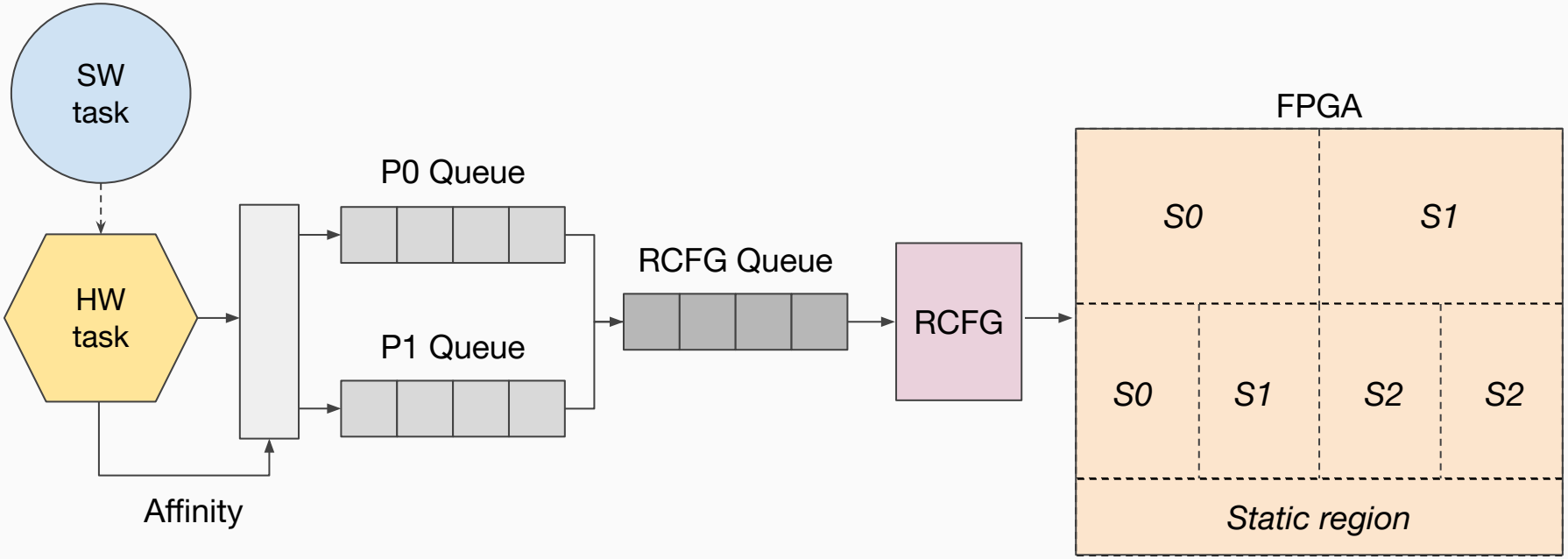
FRED runtime: HW-tasks scheduling

- How to **schedule** concurrent acceleration requests avoiding **unbounded delays**?
  - **Reconfigurations** must be **serialized**;
  - **More HW-tasks** than available **slots**.



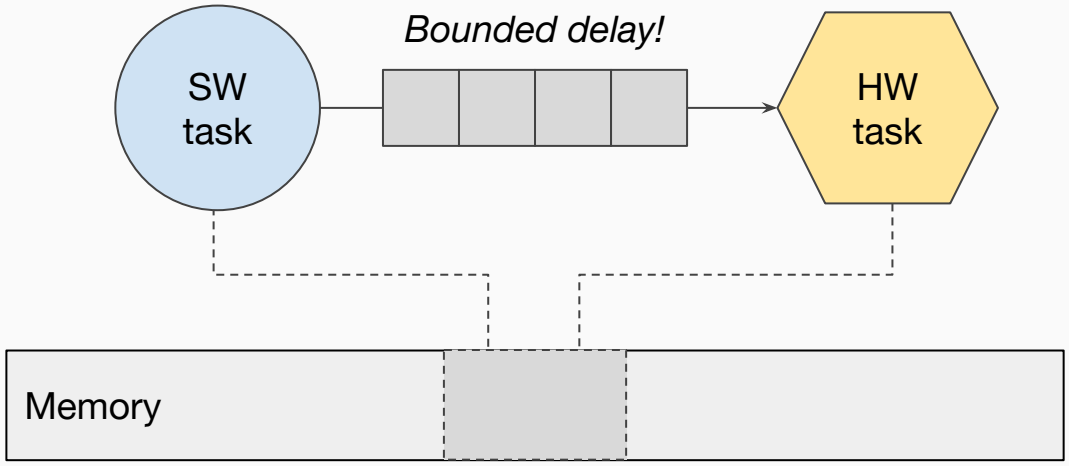
FRED runtime: HW-tasks scheduling

- **FRED** uses a custom **scheduling infrastructure** based on a **multi-level queue** structure:
  - One **queue** for each **partition** (FIFO policy);
  - Single **queue** for **reconfiguration** DMA.
    - Ordered by request timestamp (ticket-based).



FRED runtime: HW-tasks scheduling

- **This scheduling infrastructure** has been designed for **predictability**!
  - Analytical **upper-bounds** on the delay **incurred** by **SW-tasks** when requesting the execution of **HW-tasks**.



- A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, *"A Framework for Supporting Real-Time Applications on Dynamic Reconfigurable FPGAs"*, Proc. of the IEEE Real-Time Systems Symposium (RTSS 2016)
- M. Pagani, M. Marinoni, A. Biondi, A. Balsini, and G. Buttazzo, "*Towards Real-Time Operating Systems for Heterogeneous Reconfigurable Platforms"*, Proc. of the 12th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT16)

FRED runtime: life of a SW-task

- Each **SW-task** start like a **regular task** performing whatever kind of software computation.

```
sw_task_body(void)
{
    < first chunk >

    fred_accel(hw_task);

    < second chunk >
}
```

FRED runtime: life of a SW-task

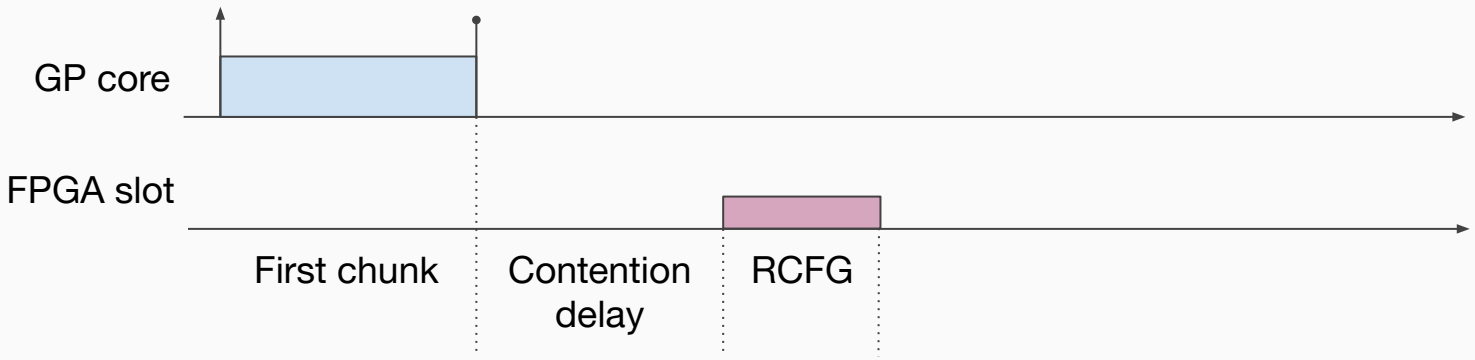
- Then, it performs an **acceleration request** calling a **HW-task** and **suspend**.

```
sw_task_body(void)
{
    < first chunk >

    fred_accel(hw_task);

    < second chunk >
}
```

FRED runtime: life of a SW-task

- After a **contention phase** (shared resources (area) and reconfiguration interface) the **HW-task** is **configured** on the FPGA

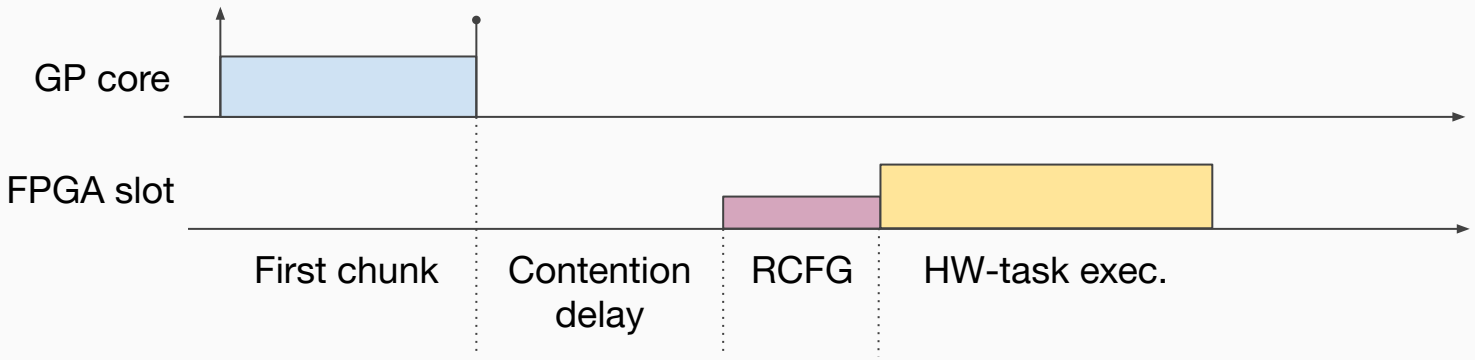
```
sw_task_body(void)
{
    < first chunk >

    fred_accel(hw_task);

    < second chunk >
}
```



FRED runtime: life of a SW-task

- And can start its **computing** phase.

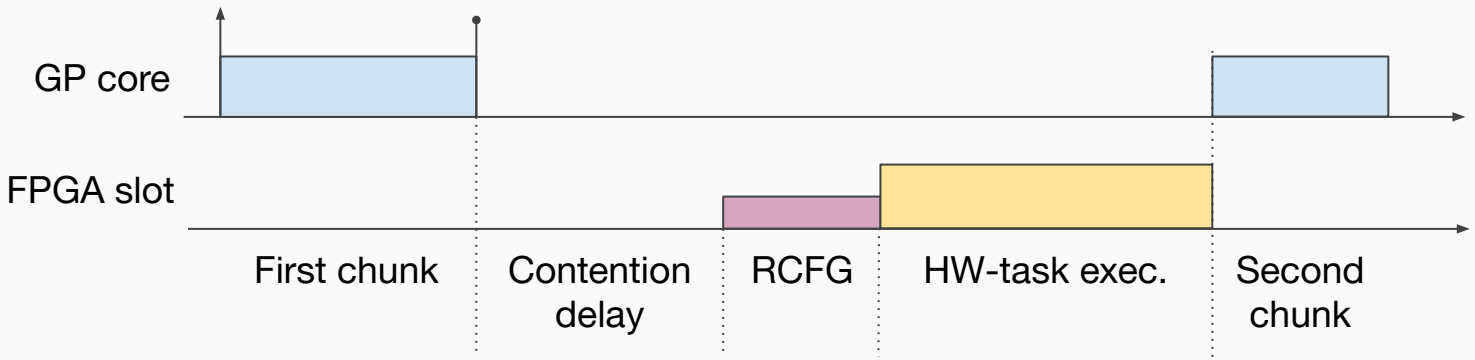
```
sw_task_body(void)
{
    < first chunk >

    fred_accel(hw_task);

    < second chunk >
}
```



FRED runtime: life of a SW-task

- Once the **HW-task finishes**, the **Sw-task unblocks** and can **continue** its software part

```
sw_task_body(void)
{
    < first chunk >

    fred_accel(hw_task);

    < second chunk >
}
```

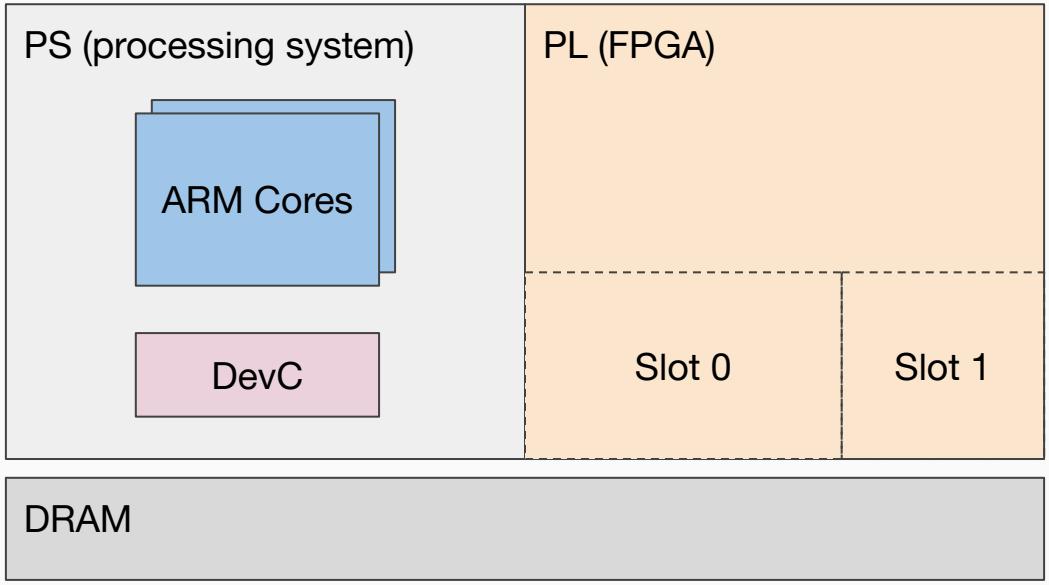


FRED on Linux

Implementing FRED on a feature-rich operating system

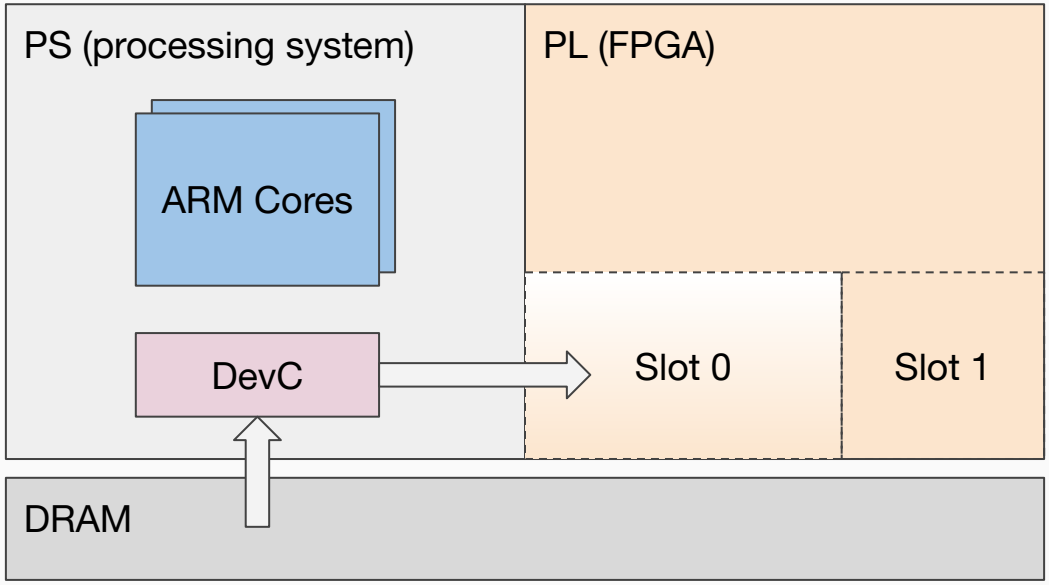
FRED on Linux: reference platforms

- The Linux implementation of FRED is based on the **Xilinx's MPSoCs** as the reference platforms:
  - A cluster of ARMv7 or v8 GP **cores**;
  - Reconfigurable **FPGA fabric**;
  - built-in **reconfiguration DMA** (DevC)



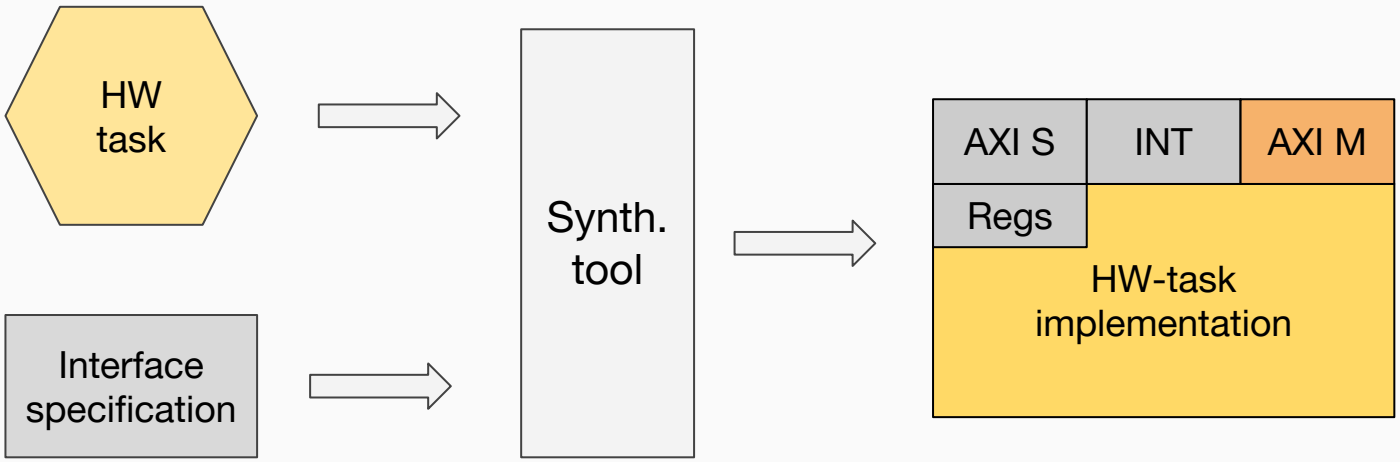
FRED on Linux: reference platforms

- The Linux implementation of FRED is based on the **Xilinx's MPSoCs** as the reference platforms:
  - A cluster of ARMv7 or v8 GP **cores**;
  - Reconfigurable **FPGA fabric**;
  - built-in **reconfiguration DMA** (DevC)



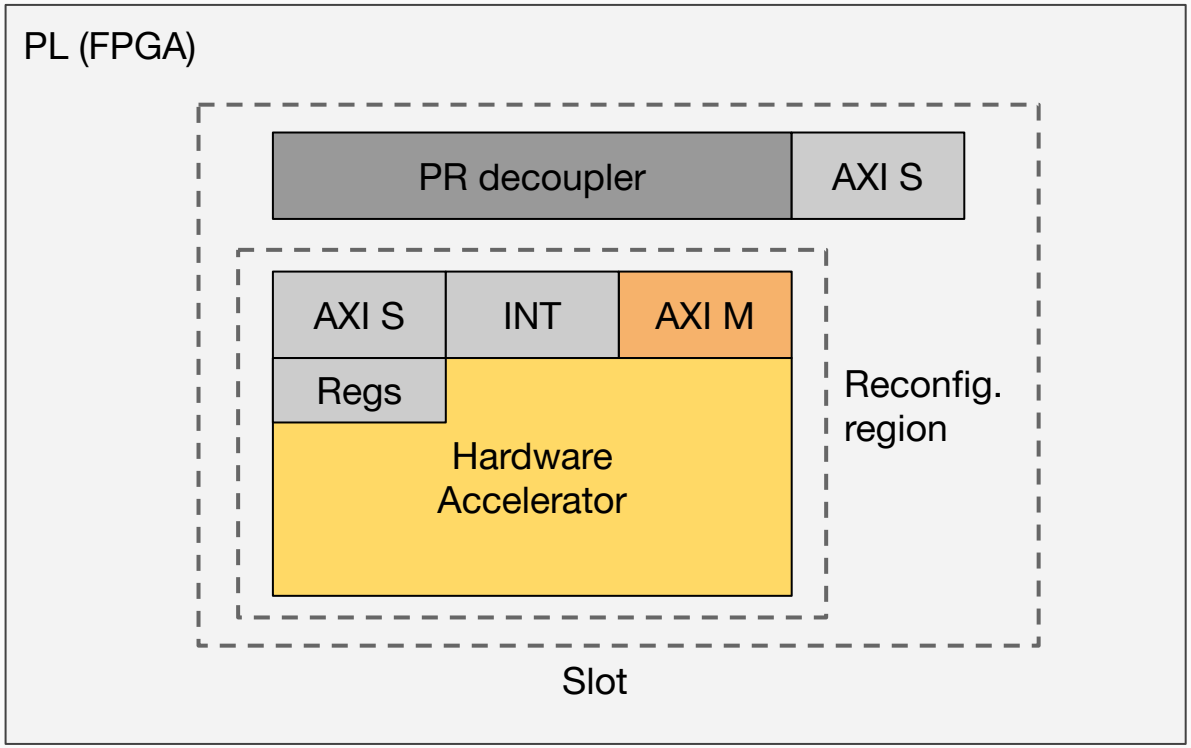
FRED on Linux: FPGA support design

- Each **slot** must be able to accommodate **any HW-task** belonging to its **partition**:
- It is necessary to define a **common interface**:
  - **AXI-MM master** for accessing **DRAM**;
  - **AXI-MM** lite **slave** for **control** and up to 8 **pointer registers**;
  - Done signal to notify the ARM cores through interrupt;

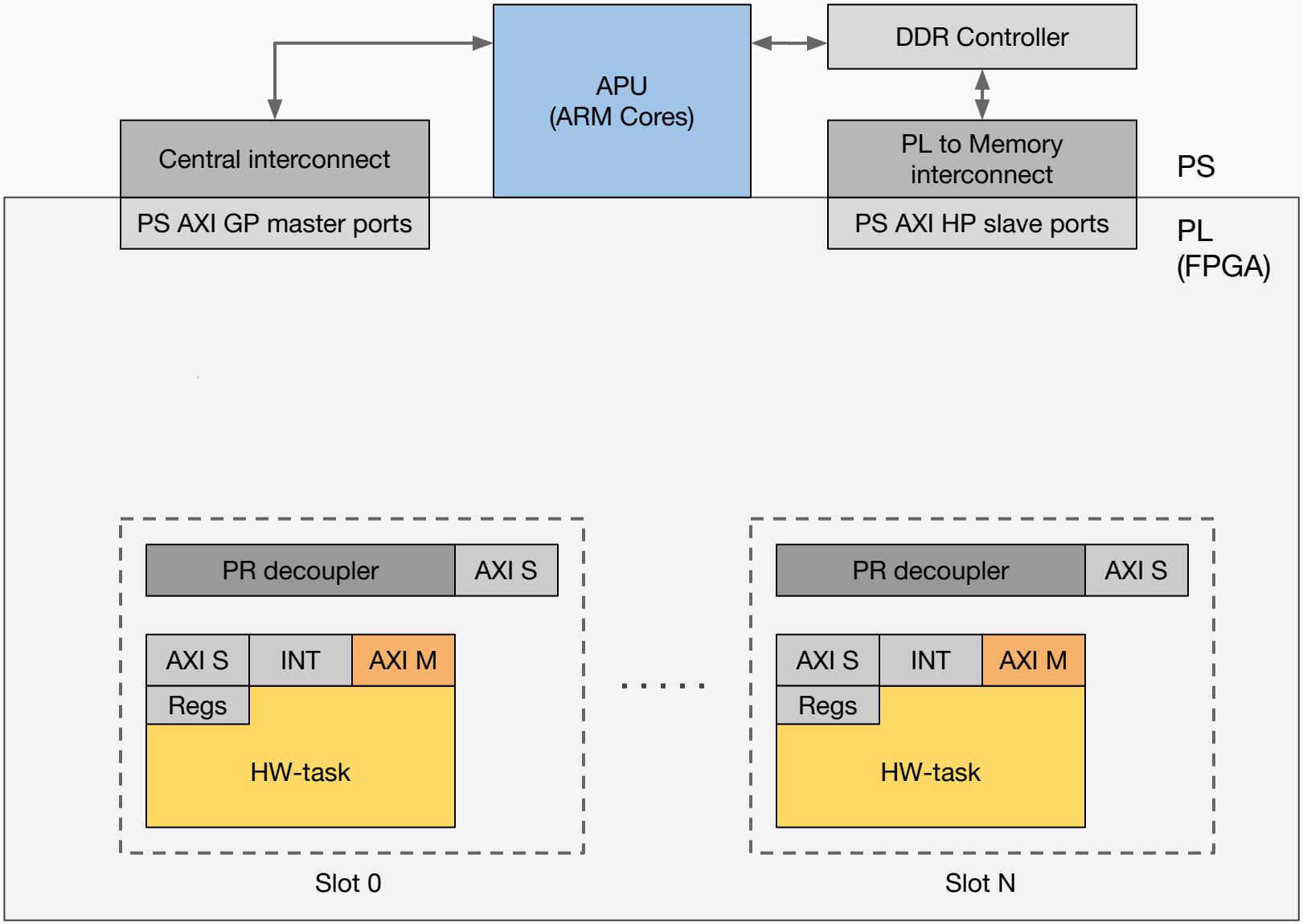


FRED on Linux: FPGA support design

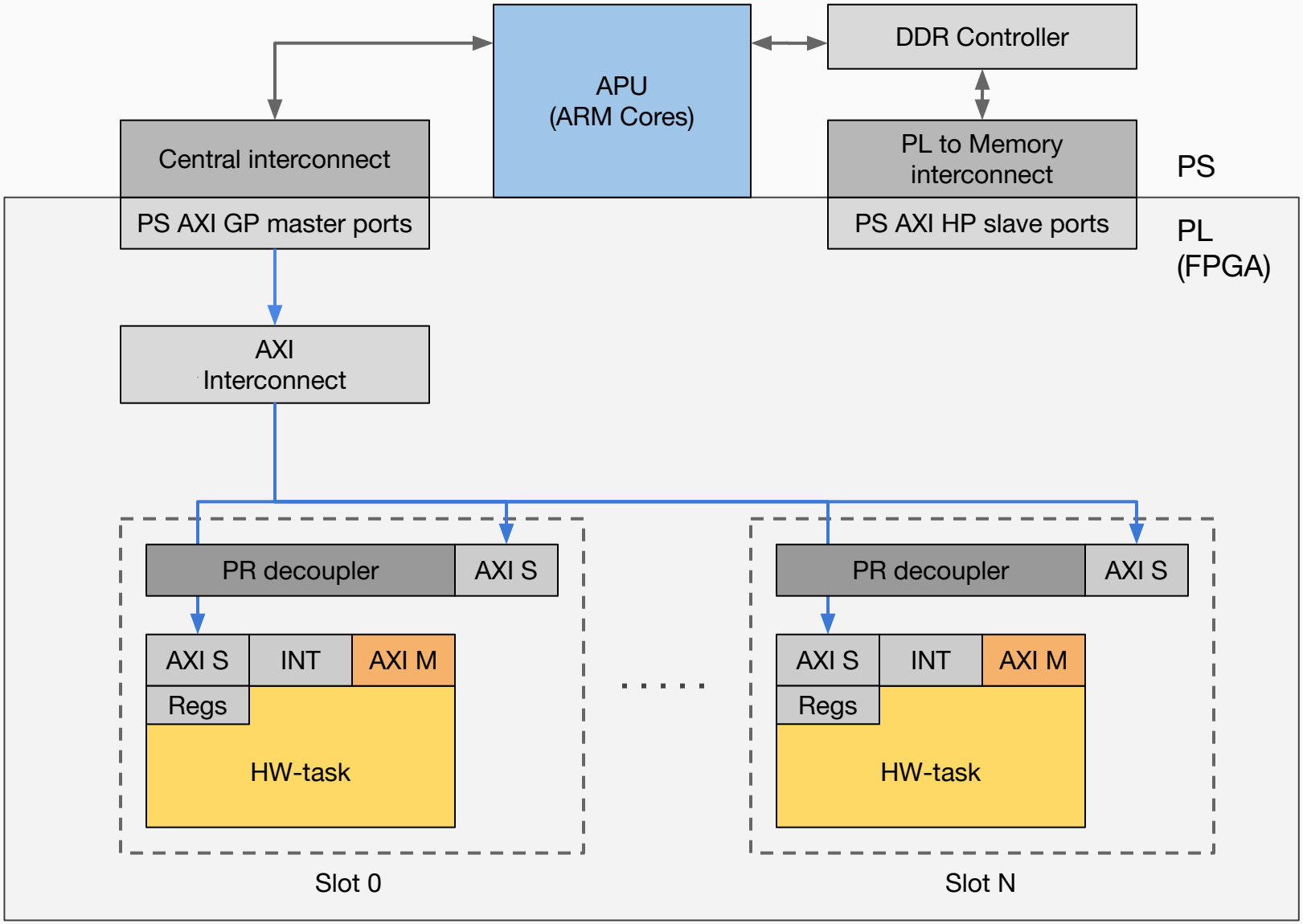
- Each **slot** is also associated to a **decoupler** (Xilinx IP) to suppress glitches during partial reconfiguration;
  - Controlled through AXI slave interface (single reg).



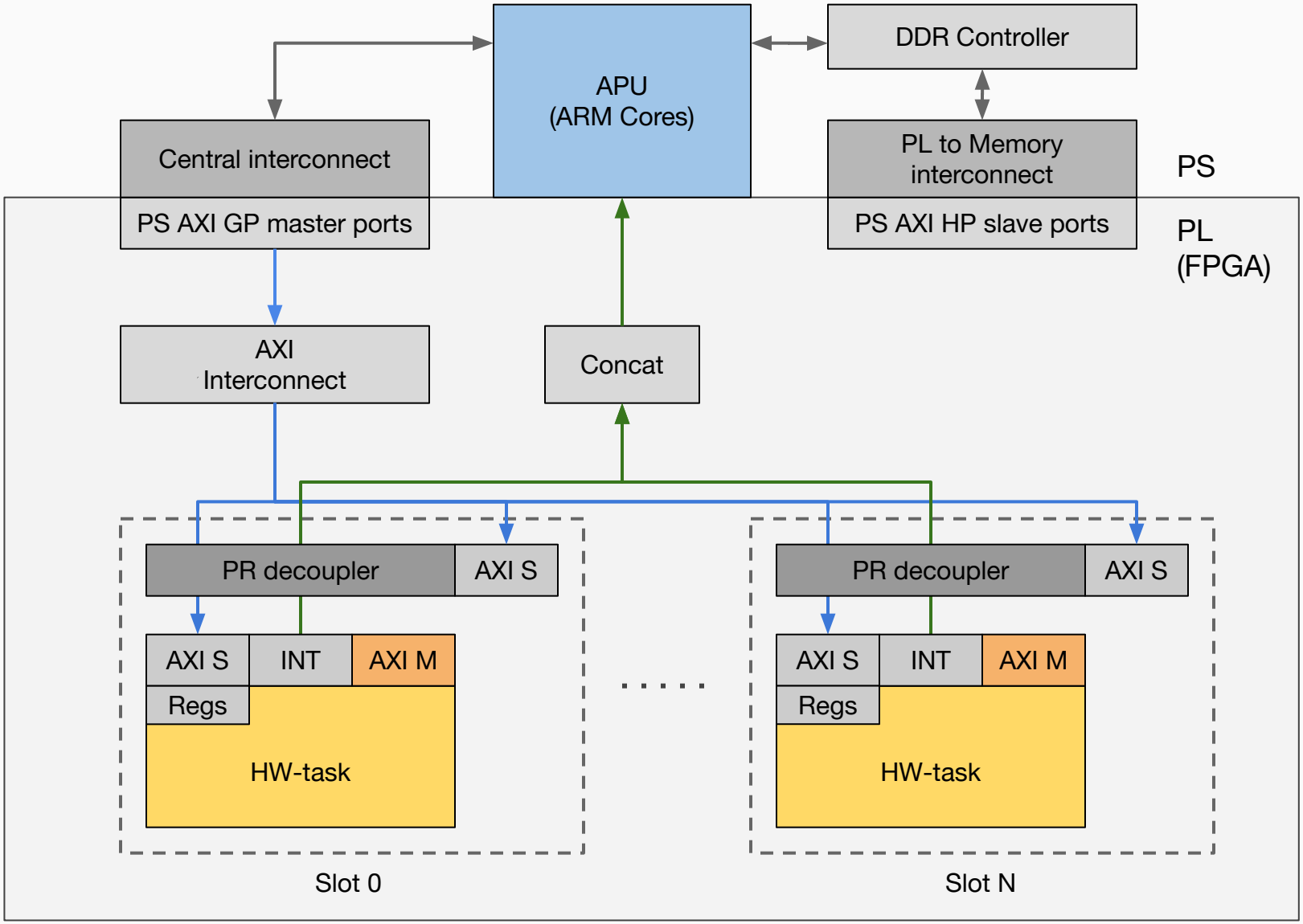
FRED on Linux: FPGA support design



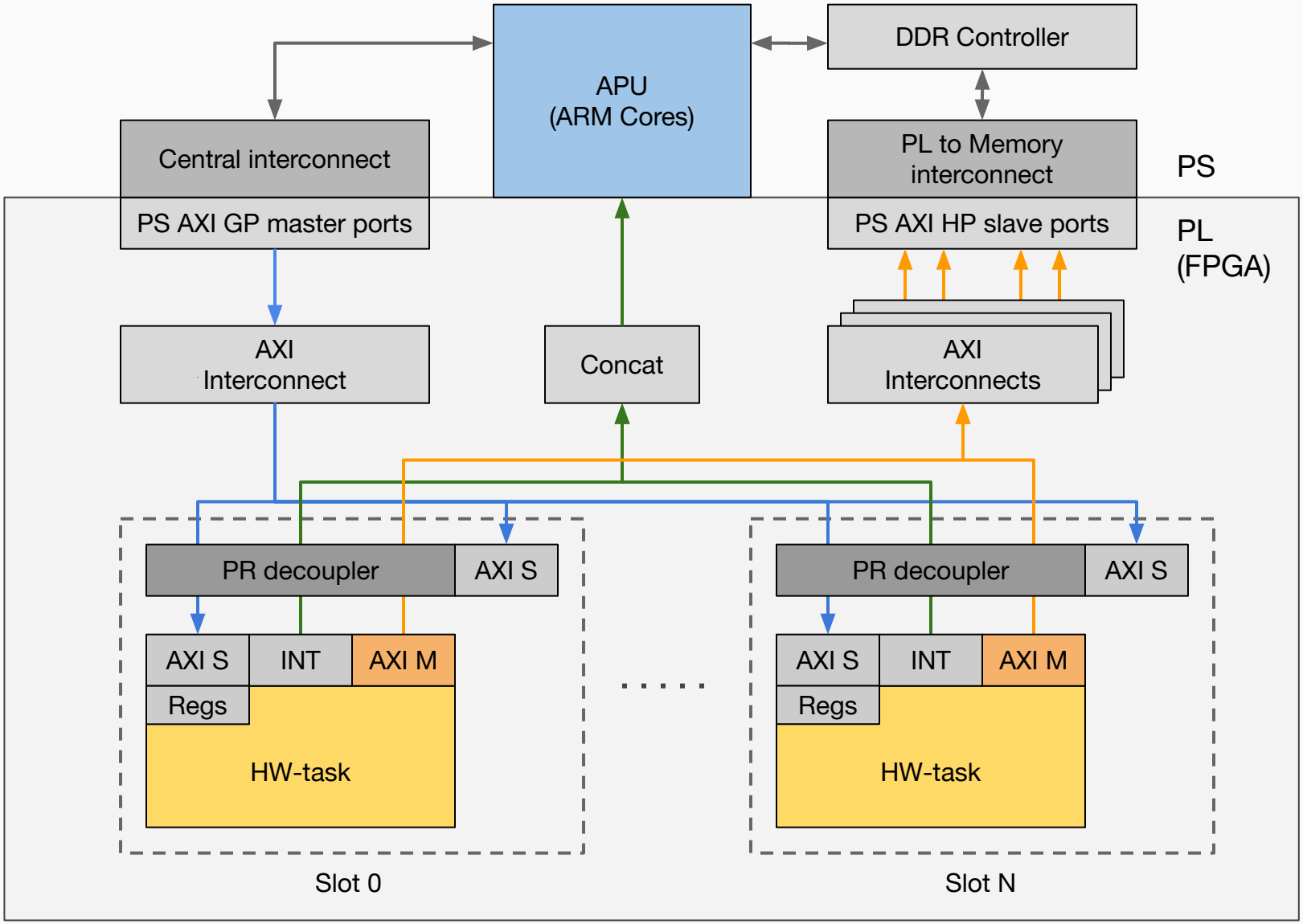
FRED on Linux: FPGA support design



FRED on Linux: FPGA support design



FRED on Linux: FPGA support design



FRED on Linux: challenges

- How to implement **FRED's** shared **memory buffers?**
  - **Linux** uses **virtual memory**!
    - ***SW-task*** *(processes/threads) uses* ***virtual addresses****;*
    - ***HW-tasks****, like other HW devices, use* ***physical addresses****;*
    - *How to handle cache coherence?*

- How to Implement the **FRED's scheduling policy?**
  - *Who is in charge of receiving and handling acceleration requests?*

- How to control **hardware resources?**
  - ***HW-tasks*** *modules;*
  - ***Reconfiguration DMA*** *and* ***decouplers****.*

FRED on Linux: design keypoints

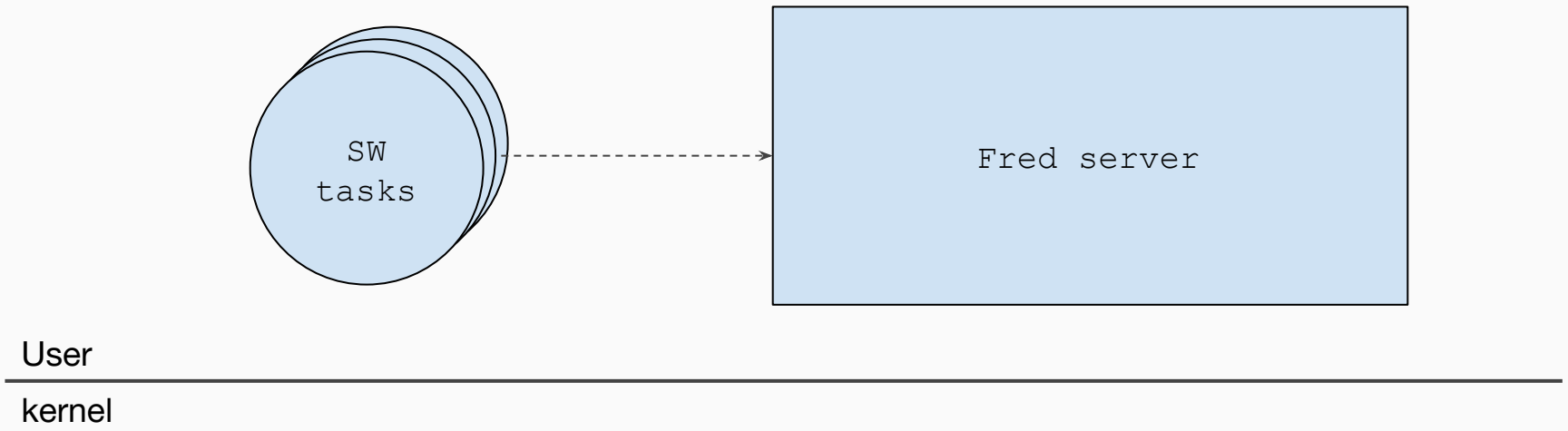
- FRED on Linux (**FredLinux**) had been **implemented**, as much as possible, in **user-space** to improve **maintainability** and **safety**:
  - *User-space server to handle and schedule acceleration requests;*
  - *Minimal kernel support.*

- **Zero-copy** design for **shared buffers** to **avoid** unnecessary **copy** operations overhead and related **BUS/memory traffic**;
  - *Linux DMA layer provides functions for allocating and mapping **large coherent** memory buffers (using CMA).*

- **Modular design** to allow reusability and future extensions:
  - *Core mechanisms are independent from the platform and hardware specific support.*

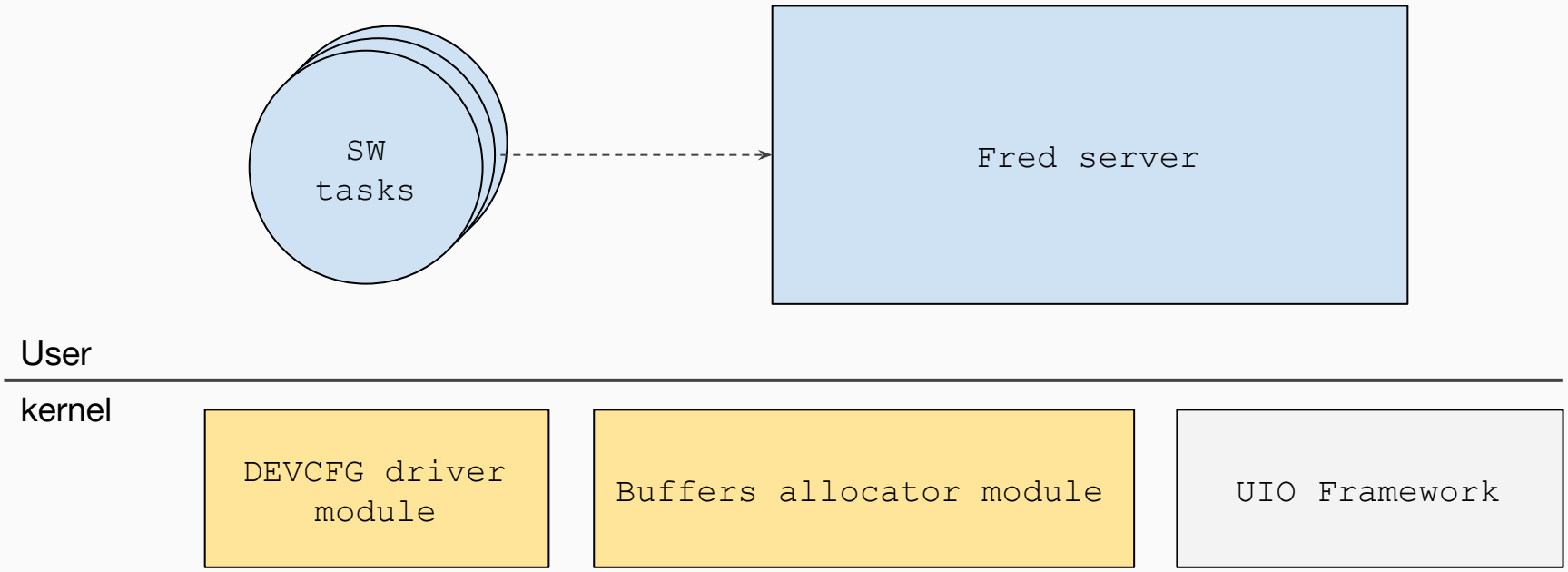
FRED on Linux: software architecture overview

- The **central component** of **FREDLinux** is a user-space server process named FRED server:
  - **Receives** and **manages** acceleration request from **SW-tasks**.



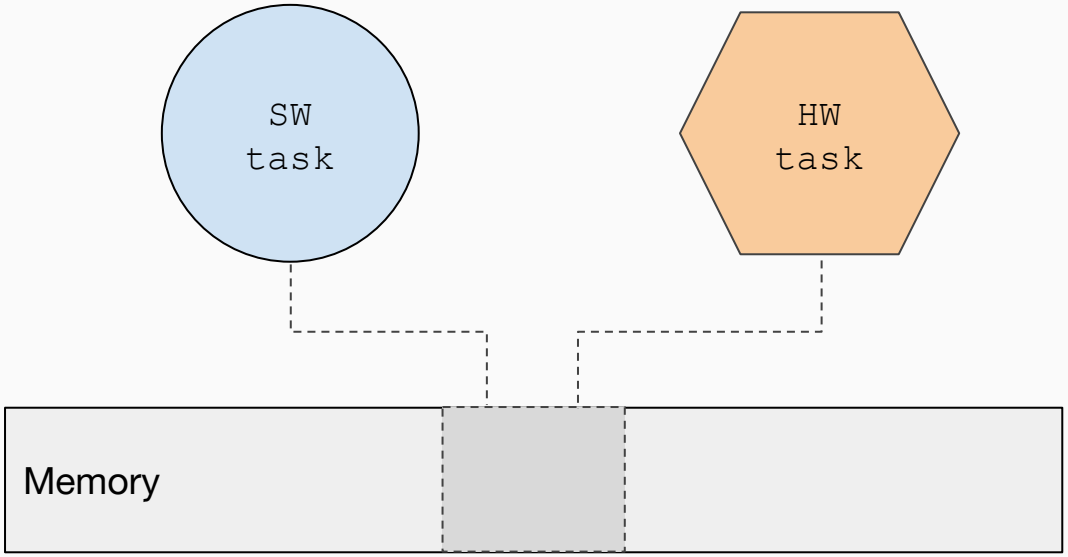
FRED on Linux: software architecture overview

- The **central component** of **FREDLinux** is a user-space server process named FRED server:
  - **Receives** and **manages** acceleration request from **SW-tasks**.
  - Relies upon two custom **kernel modules**, and the UIO framework, for low-level operations.



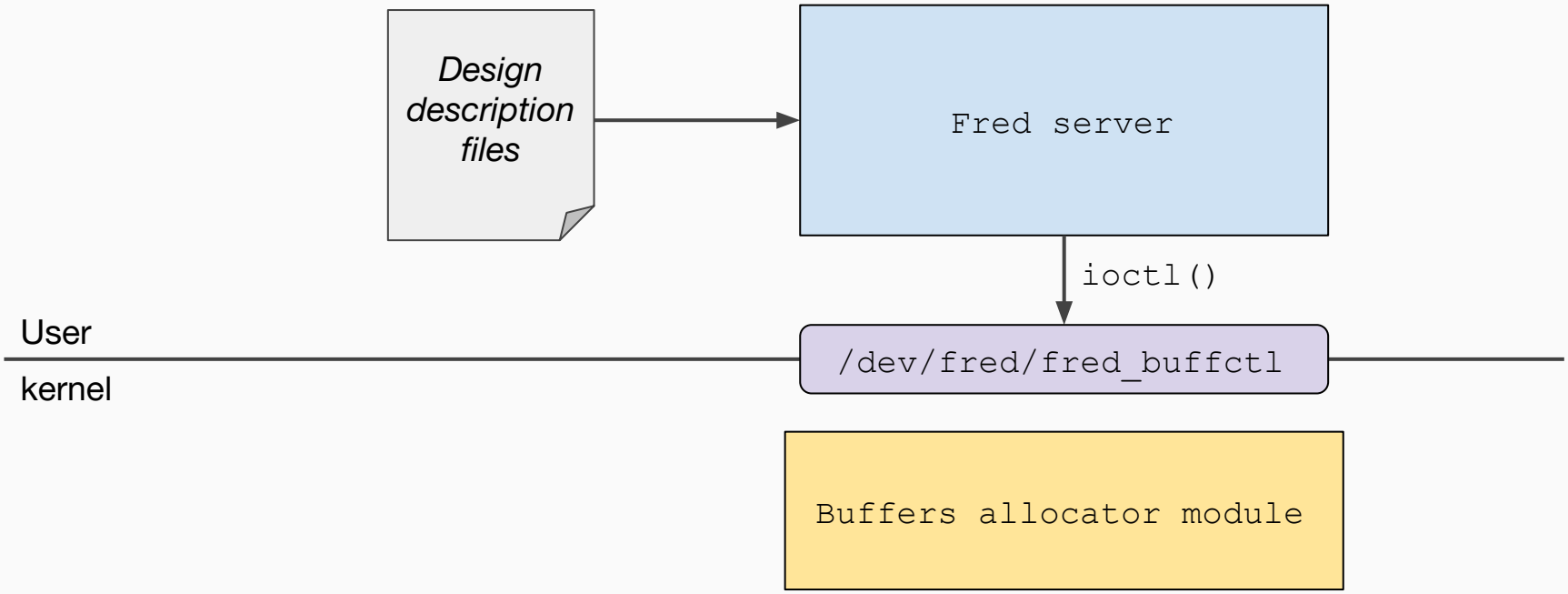
FRED on Linux: buffers allocator module

- The purpose of the **buffer allocator module** is to:
  - Allocate **physically contiguous**, **uncached**, memory **buffers**;
  - Provide the means by which such buffers can be **accessed efficiently** from user space by **SW-tasks**.



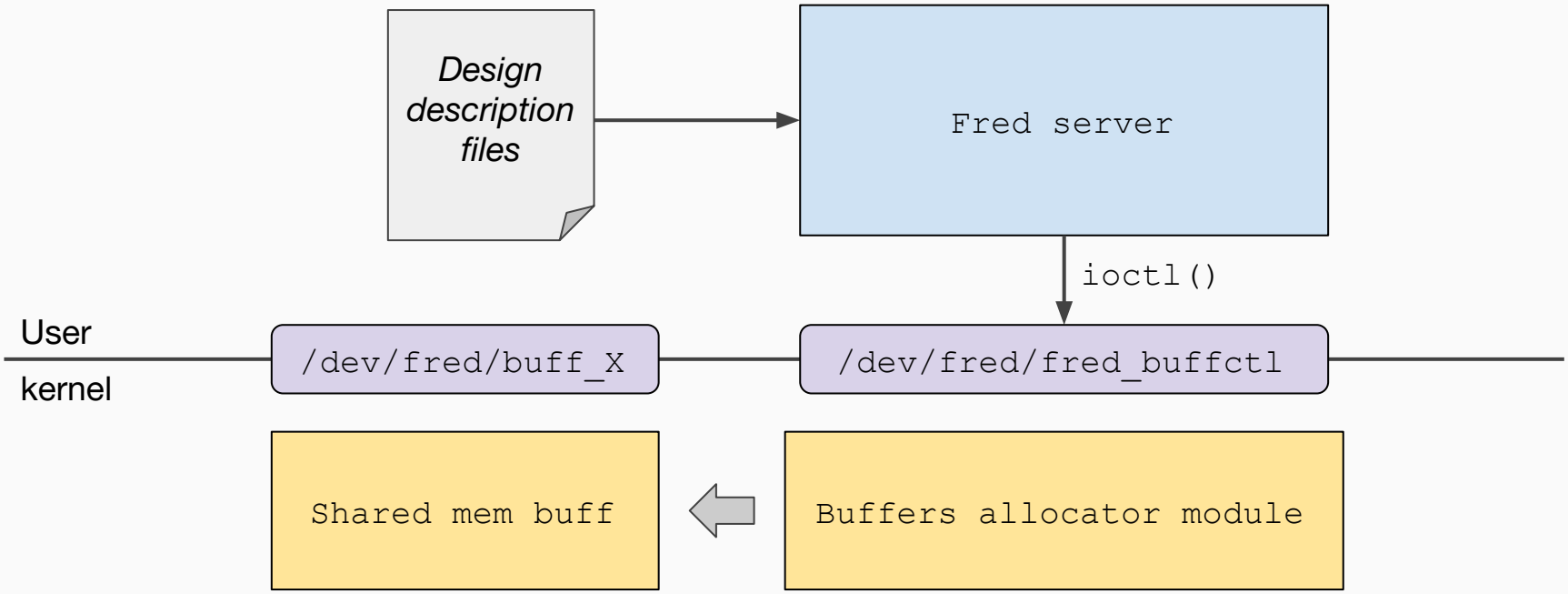
FRED on Linux: buffers allocator module

- When loaded, the **buffer allocator module** instantiates a new **character device** named `fred_buffctl`.
  - Buffers are allocated during the **`Fred server`** initialization according to two design description files.
  - Each **allocation request** is performed by an `ioctl()` syscall:



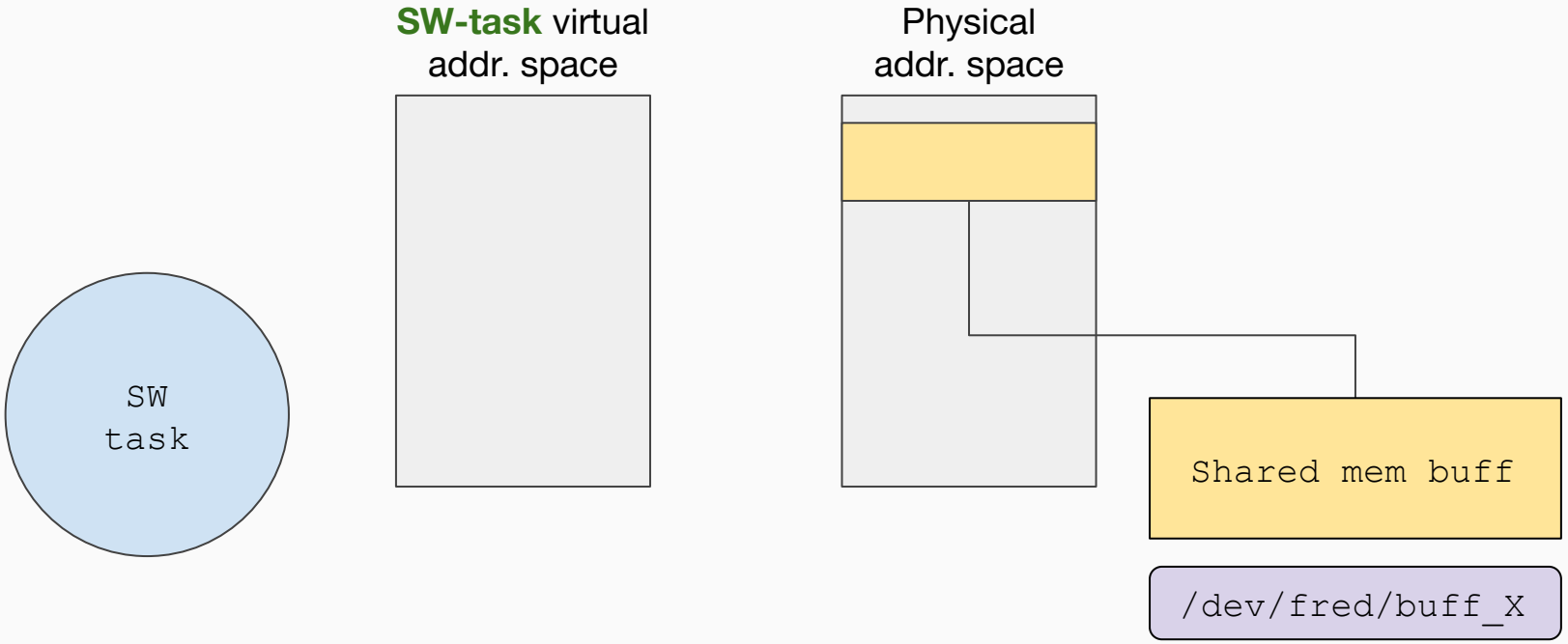
FRED on Linux: buffers allocator module

- On the kernel side, the buffer allocator module:
  - **Creates** a **new character device** named `buff<N>`;
  - **Allocates** a **new** contiguous **memory buffer**, associated with the `buff<N>` device, using the `dma_alloc_coherent()` function of the DMA layer.



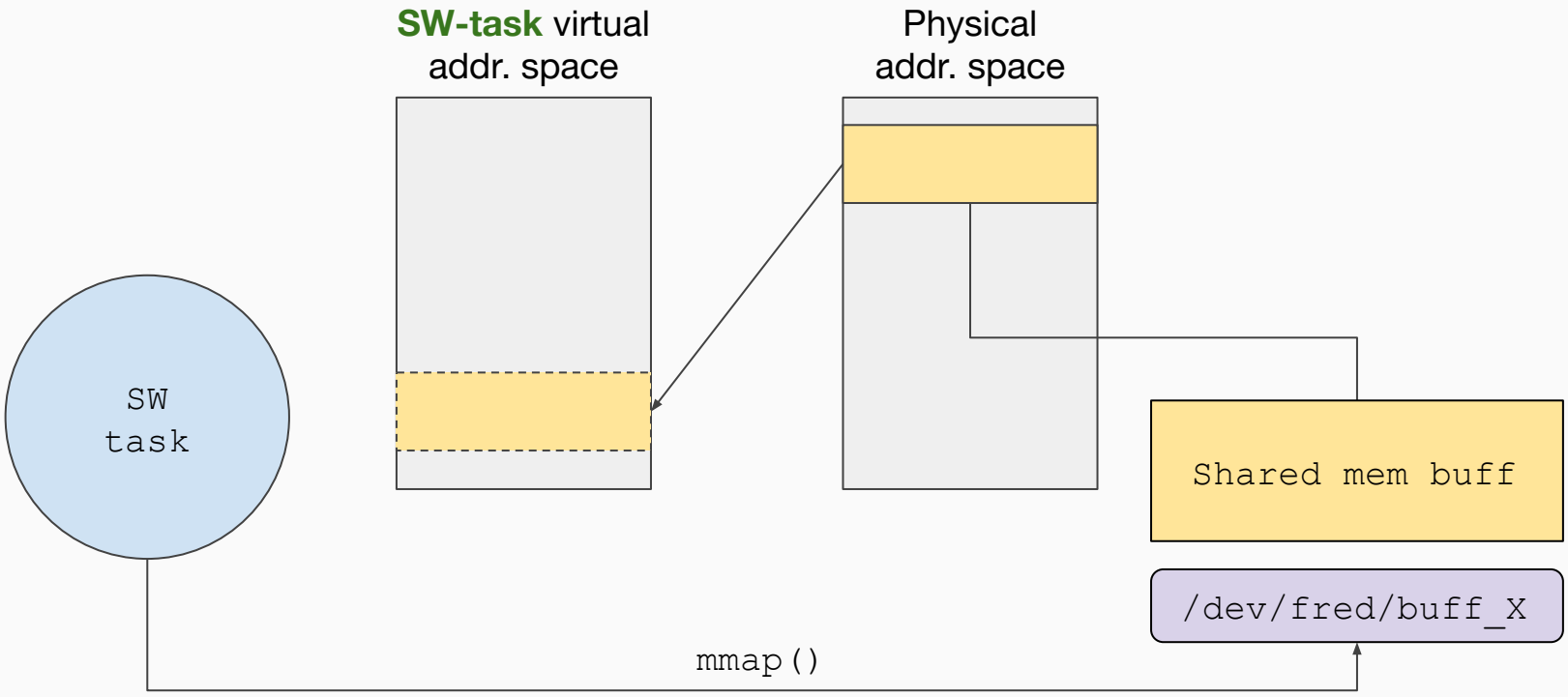
FRED on Linux: buffers allocator module

- The `buff<N>` char devices **implement** the `mmap()` method using the `dma_common_mmap()` function.



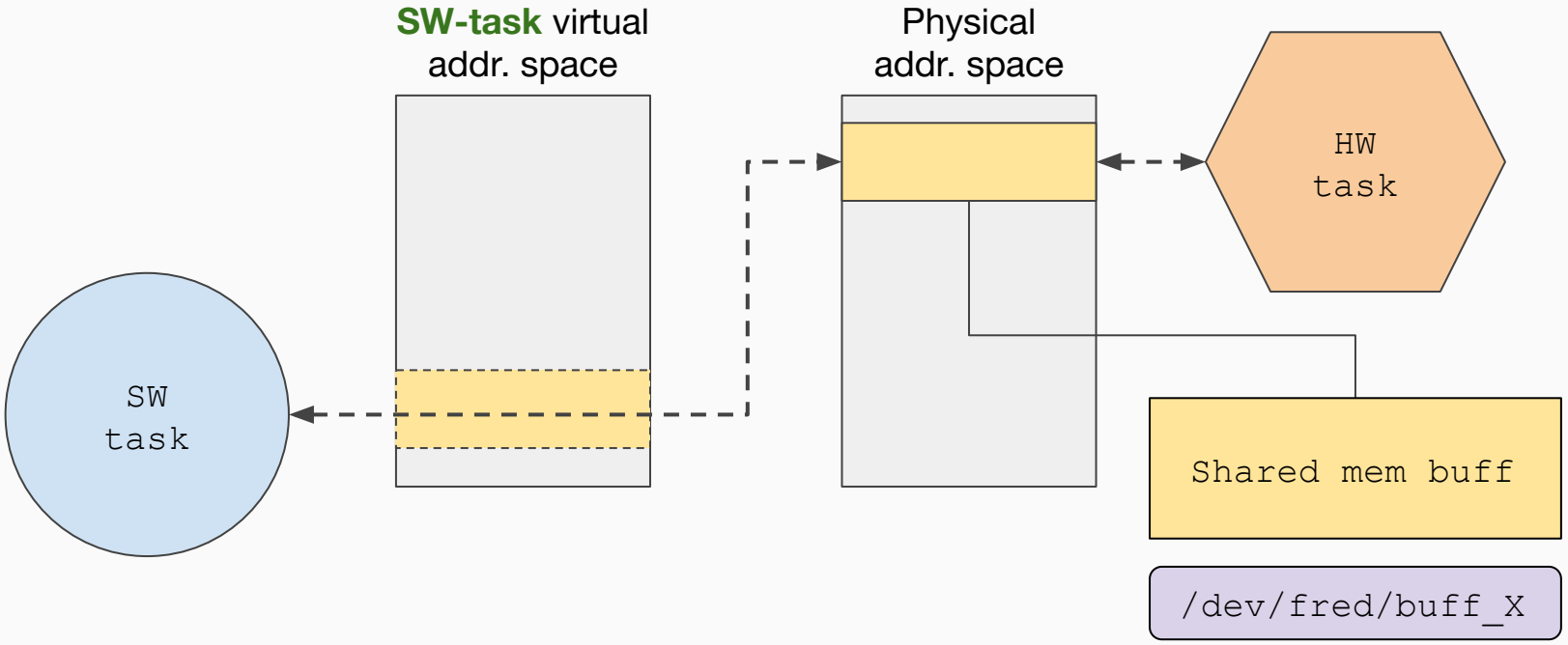
FRED on Linux: buffers allocator module

- The `buff<N>` char devices **implement** the `mmap()` method using the `dma_common_mmap()` function.
- When a **SW-task** calls (from userspace) the `mmap()` on the char device the **buffer** gets **mapped** into its virtual memory space.



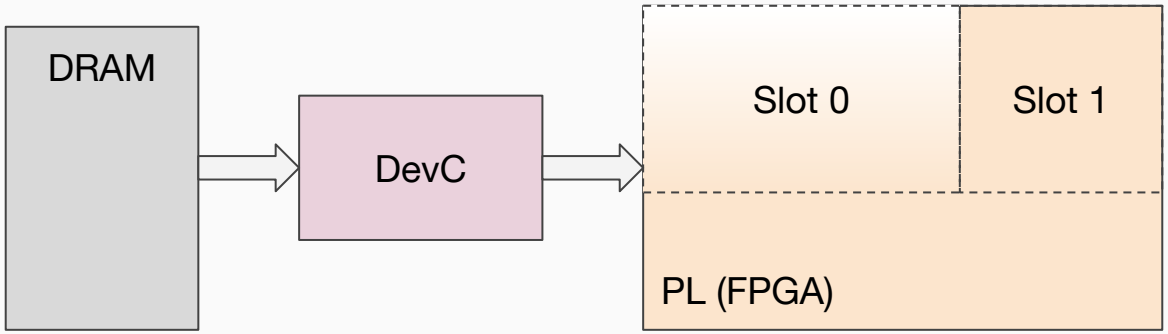
FRED on Linux: buffers allocator module

- Once the buffer is mapped, it can be read and write data **without any overhead**. **No copy or flush** are needed.



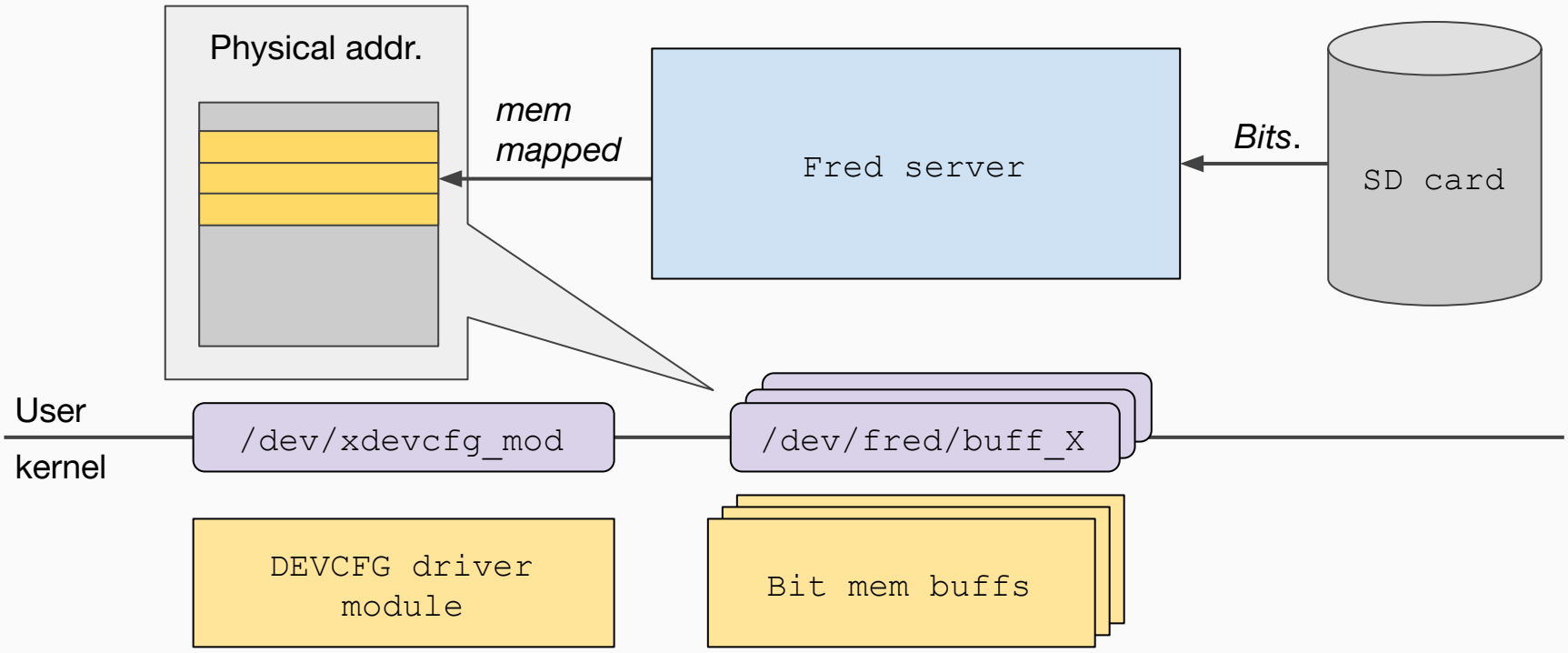
FRED on Linux: reconfiguration module

- Xilinx's original **reconfiguration driver** (DevC) was designed to be safe and **easy** to use, **not for efficiency**:
- For **each** reconfiguration:
  - **Allocates** a **new** contiguous memory **buffer**;
  - **Copies** the whole bitstream from **userspace to kernel**;
  - **Busy wait** until completion.
- **Unsuitable** for the **intensive use** of partial reconfiguration required by **FRED**!



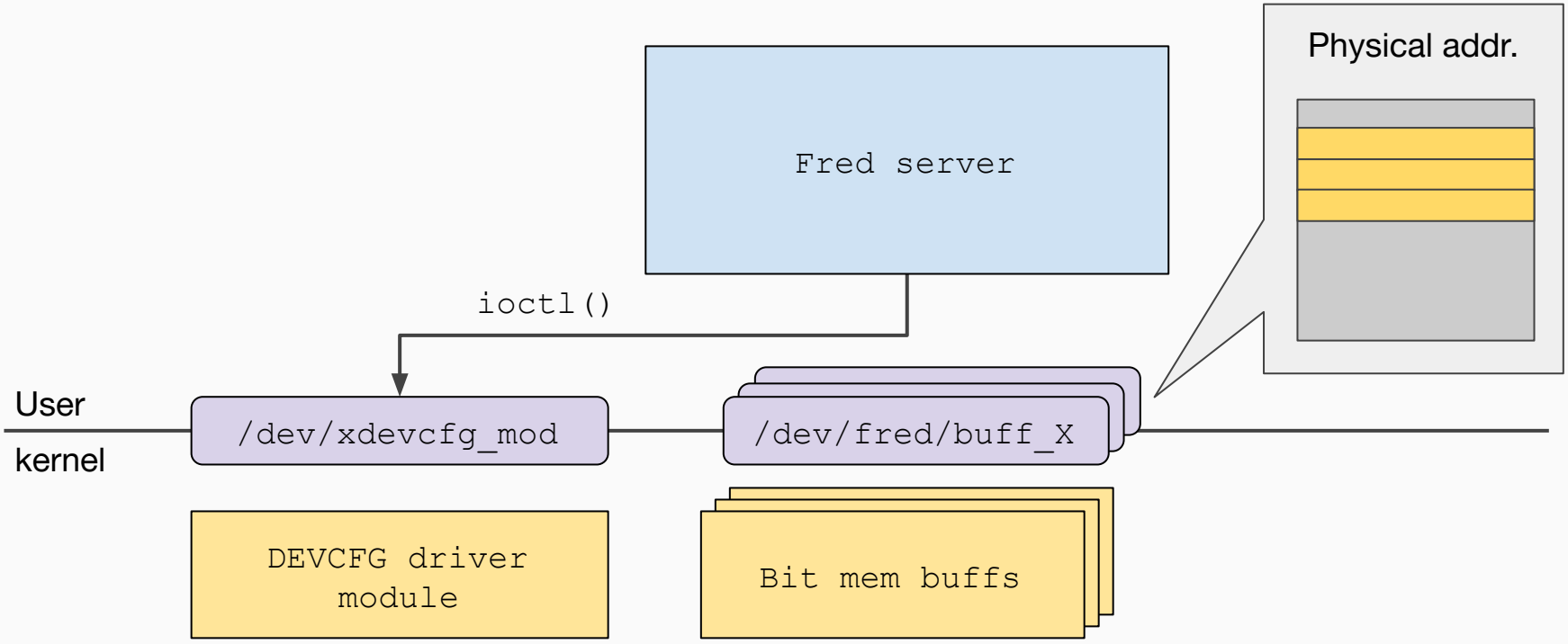
FRED on Linux: reconfiguration module

- To overcome those issue the **DevC** driver has been modified:
  - **Preload** all the **bitstreams** (HW-tasks images) into set of physically contiguous buffers.



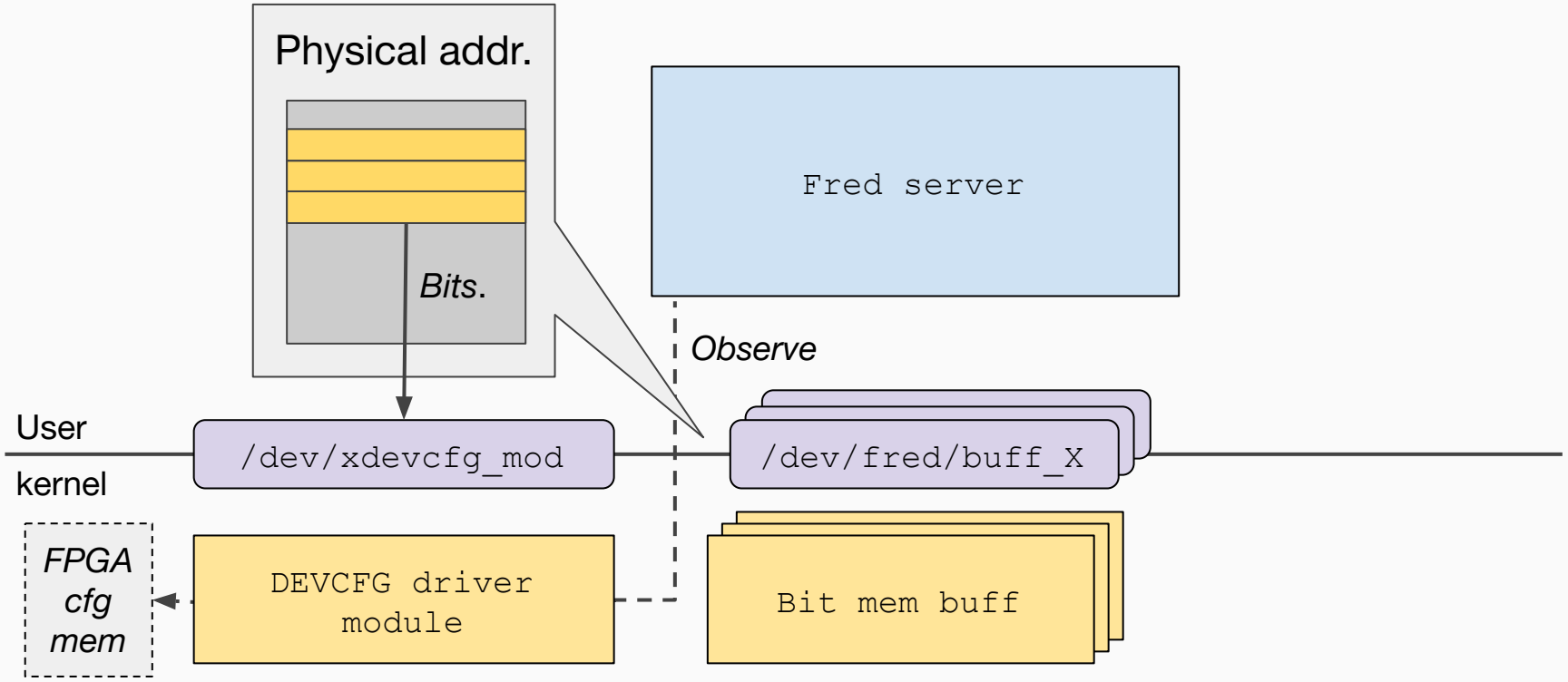
FRED on Linux: reconfiguration module

- Now the **reconfiguration** can also be **initiated** by an `ioctl()` call passing, as argument, a **reference to the buffer**;
  - `write()` method untouched for legacy compatibility.



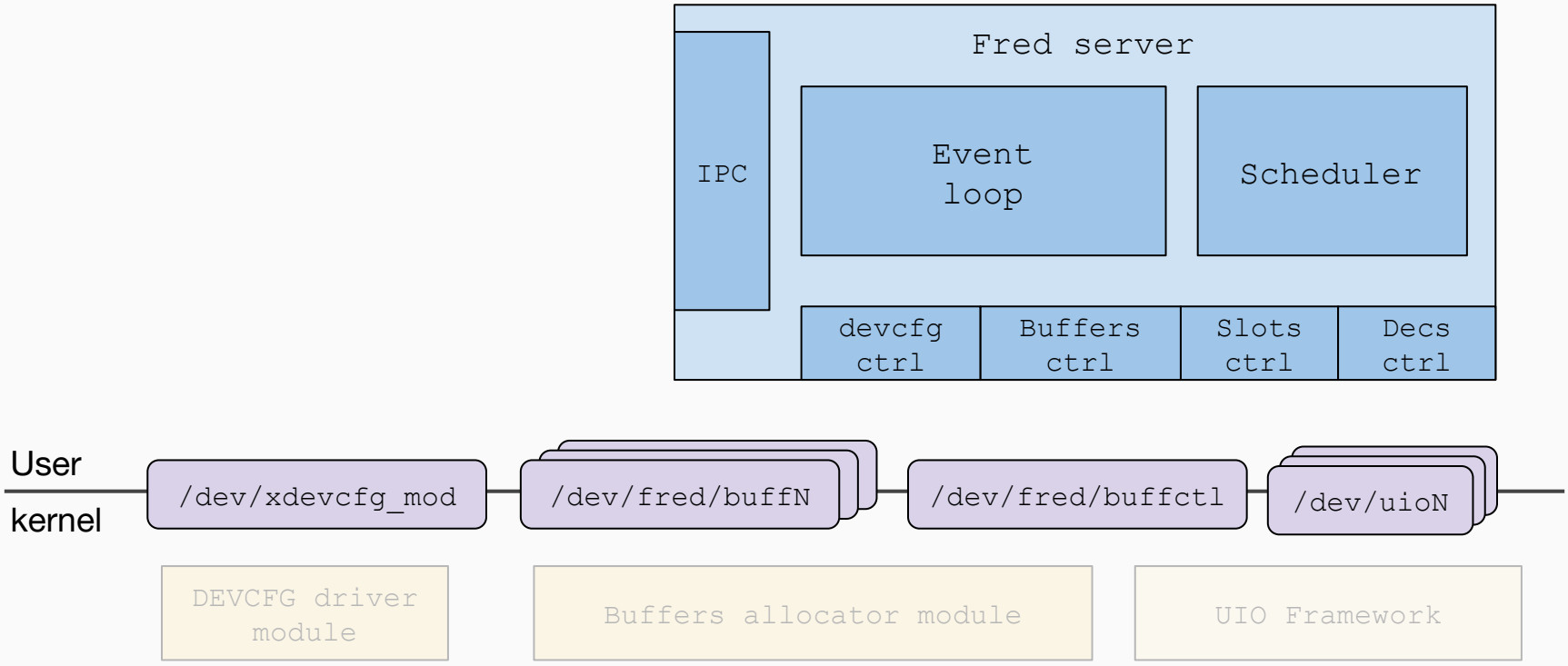
FRED on Linux: reconfiguration module

- To **avoid busy-waits** and allow I/O multiplexing, the driver has been **enhanced** with the `poll()` method.
  - The `ioctl()` **returns immediately** after the reconfiguration has been initiated;
  - Once **reconfiguration is complete** char device **fd** becomes **ready**.



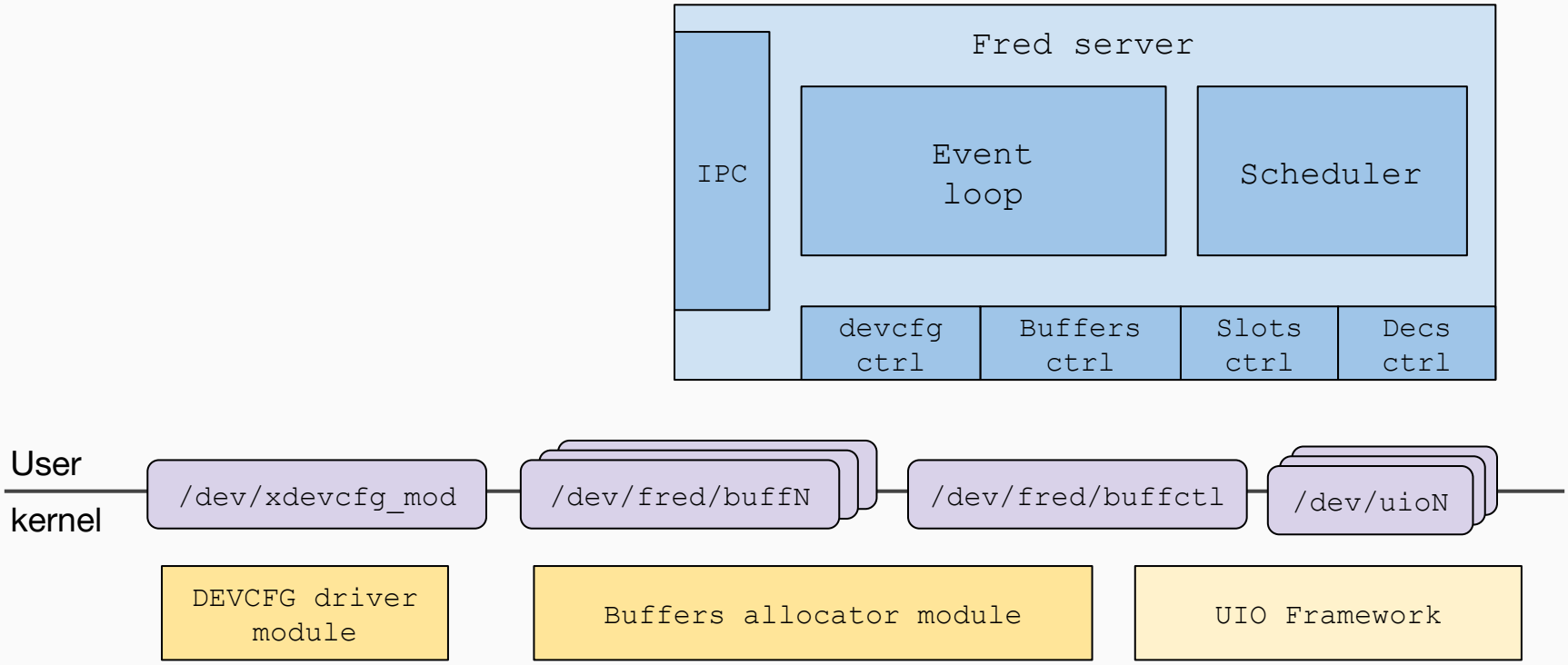
FRED on Linux: server internals

- **FRED Server** is organized as an **event-driven system**:
  - Organized as a **state machine** driven by an **event loop**;
    - **Monitors** the **file descriptors** using *`epoll()`* or *`poll()`*;
    - **Sleep until** an **event occurs**.
  - The **HW-tasks scheduler** is the core component;



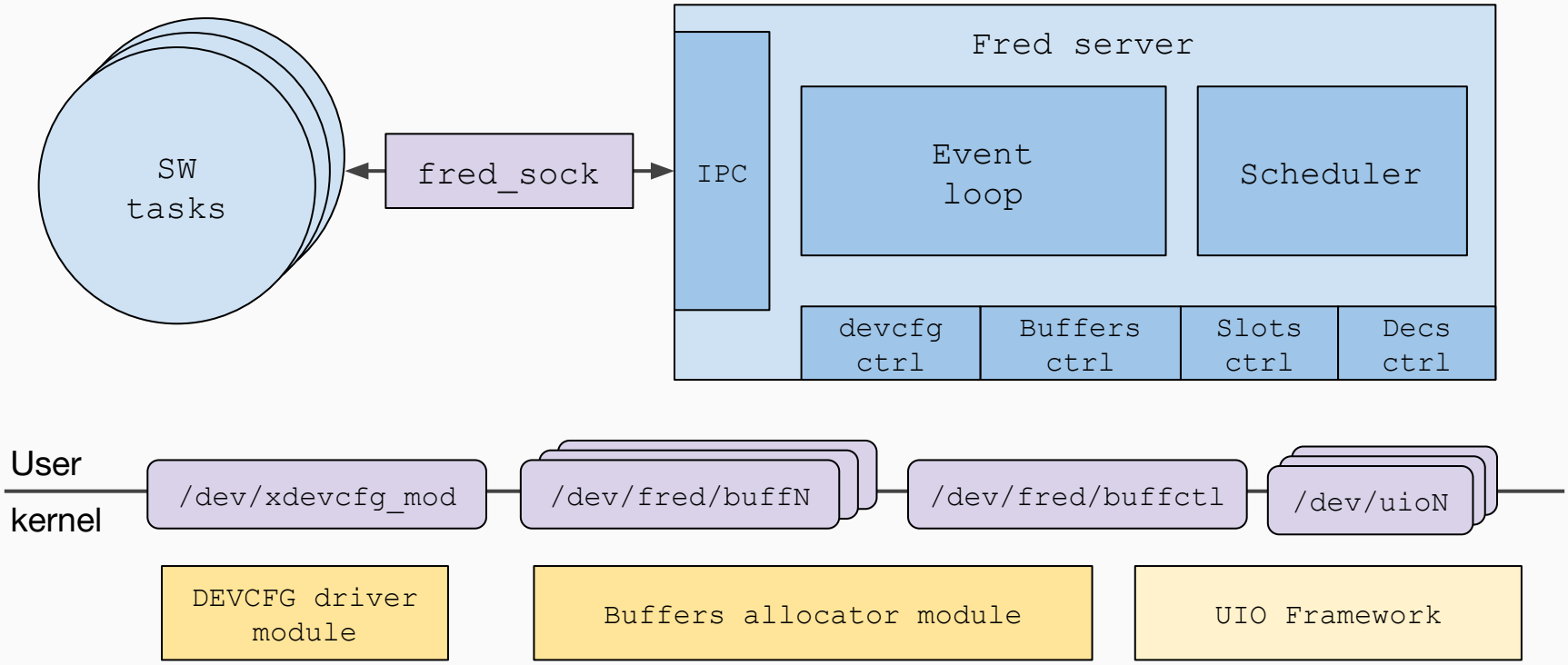
FRED on Linux: server internals

- **Low-level** components for interacting with **kernel support**;



FRED on Linux: server internals

- **Low-level** components for interacting with **kernel support**;
- **Inter-process** communication with **SW-tasks** to receive requests using **unix domain sockets**.



FRED on Linux: SW-task API

```
struct fred_data;
struct fred_hw_task;

/* ---------------------------------------------------------------------------- */

int fred_init(struct fred_data **self);

int fred_bind(struct fred_data *self, struct fred_hw_task **hw_task, uint32_t hw_task_id);

int fred_accel(struct fred_data *self, const struct fred_hw_task *hw_task);

void fred_free(struct fred_data *self);

/* ---------------------------------------------------------------------------- */

void *fred_map_buff(const struct fred_data *self, struct fred_hw_task *hw_task, int buff_idx);

void fred_unmap_buff(const struct fred_data *self, struct fred_hw_task *hw_task,
                     int  buff_idx);
```

FRED on Linux: SW-task pseudocode example

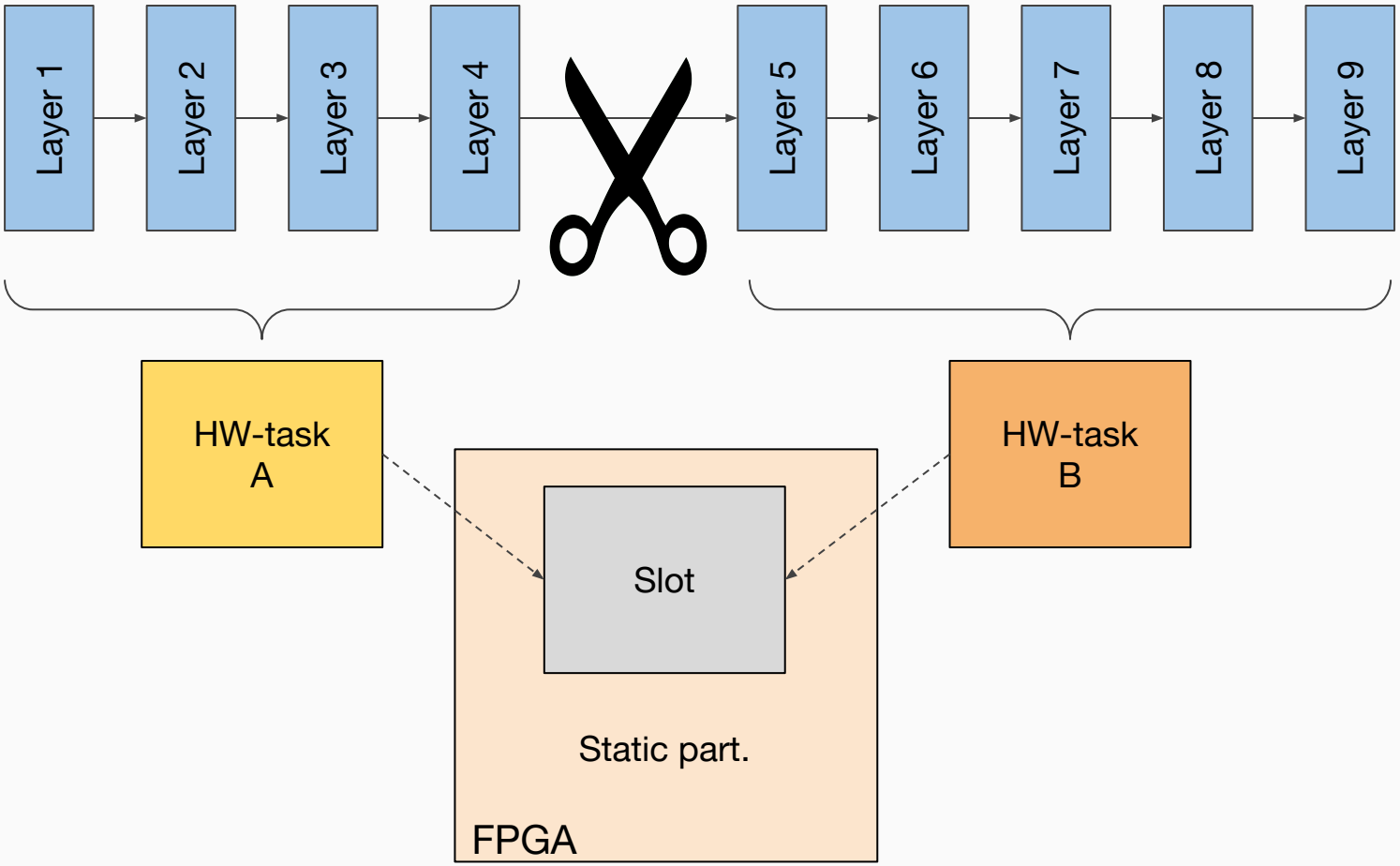
```
struct fred_data *fred;
struct fred_hw_task *hw_task;
uint32_t hw_task_id = 100;

void sw_task(void)
{
     void *buff_in = NULL;
     void *buff_out = NULL;

     /* Init communication and bind a HW-task */
     fred_init(&fred_data);
     fred_bind(fred_data, &hw_task, hw_task_id);

     /* Map the buffers */
     buff_in = fred_map_buff(fred, hw_task, 0);
     buff_out = fred_map_buff(fred, hw_task, 1);

     while (done) {
          fred_accel(fred_data, hw_task);
          < wait for the next period >
     }
}
```

FRED on Linux: use cases

- **Image processing** and **matrix multiplication** on the Zybo Board
  - **2 slots**, **4 HW-tasks** (Sobel, FAST, Gmap, and Mult);
  - More than **50** partial **reconfigurations** per **second**.

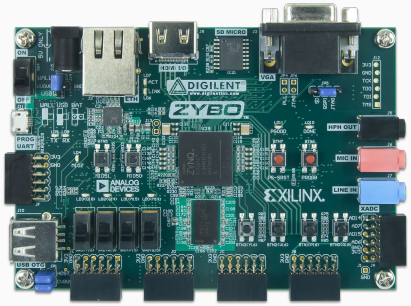
FRED on Linux: use cases

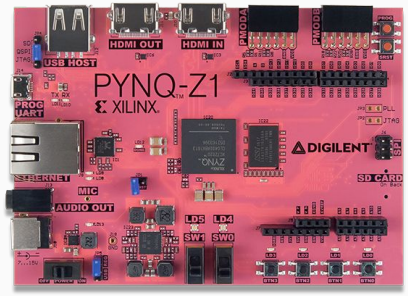
- **Deep learning** on **PYNQ** with FINN:
  - **Splitting** large a convolutional (quantized) **neural network**.



FRED on Linux: supported platforms

Zynq-7000 series SoC





Zynq UltraScale+
MPSoC (in progress)

`fred.santannapisa.it`

The FRED framework:

- TODO:
  - *Update the reconfiguration driver and the fred server for the new manufacturer agnostic* `FPGA Manager;`
  - *Extend support to other platforms;*
  - *Support HW-task to HW-task communication (waring: model and real-time analysis should be updated);*
  - *I'm here to collect suggestions and advices to improve the runtime!*

Thank you for your attention

marco.pagani@sssup.it

Contributors:

Alessandro Biondi, Francesco Restuccia, Biruk Seyoum, Giuseppe Lipari, Enrico Rossi, Alessio Balsini, Sara Balleri, Lorenzo Molinari

Project Coordinators:

Alessandro Biondi, Mauro Marinoni, Giorgio Buttazzo

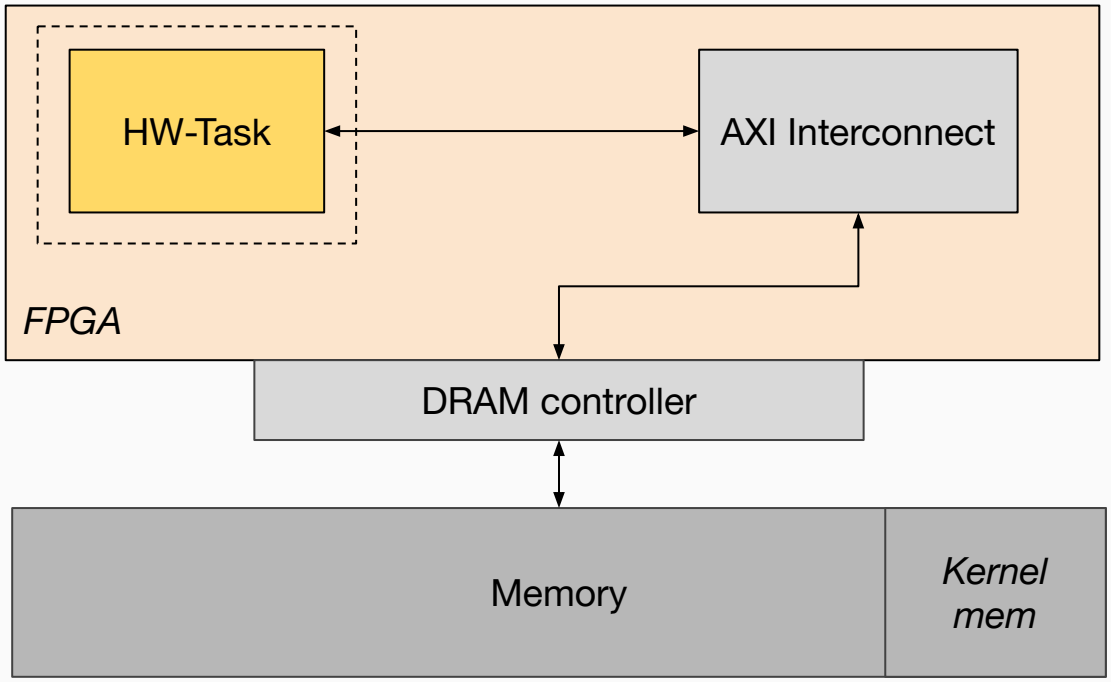
Questions?
Suggestions?

marco.pagani@sssup.it

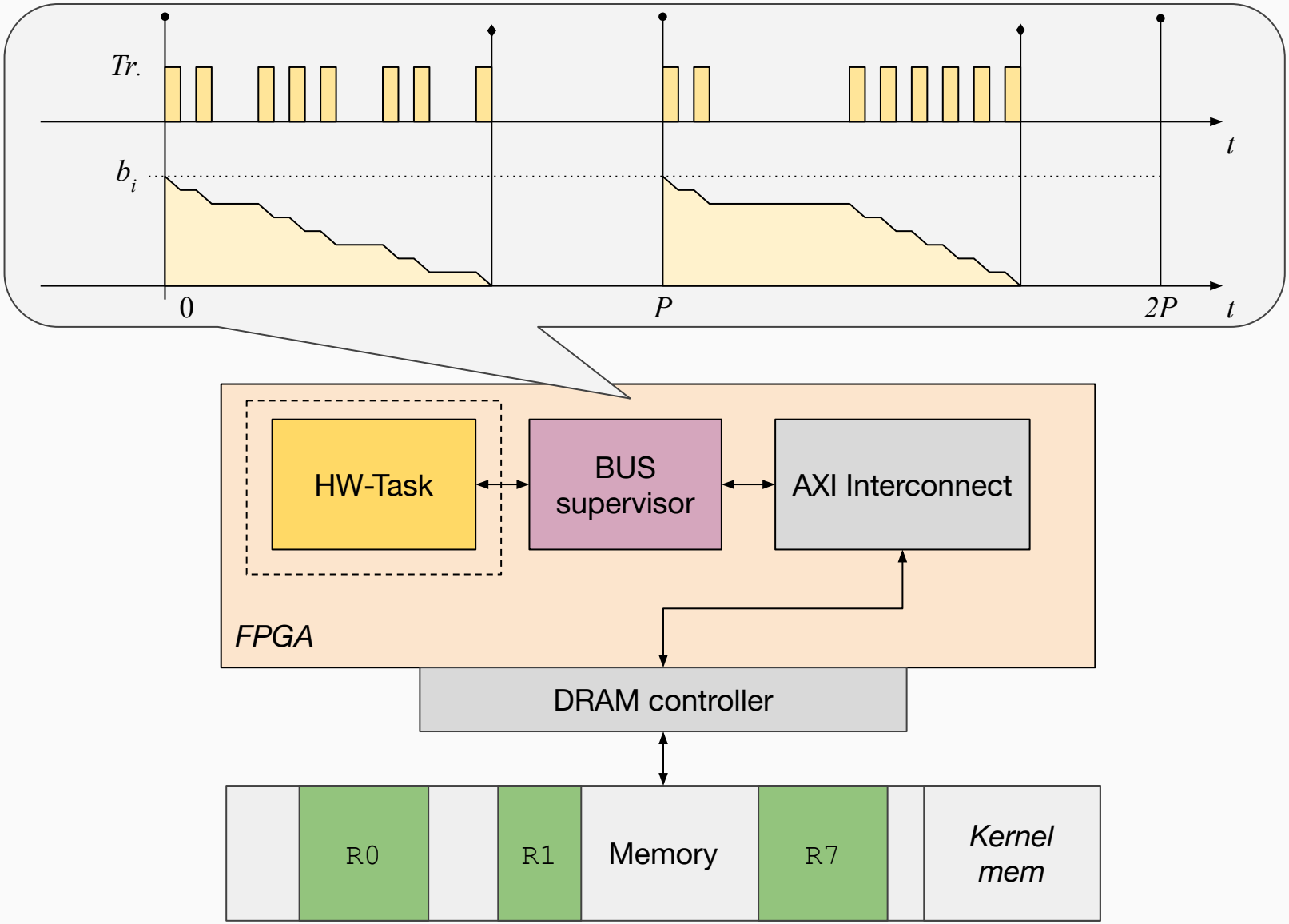
Beyond the runtime

Brief overview on the other parts of the FRED framework

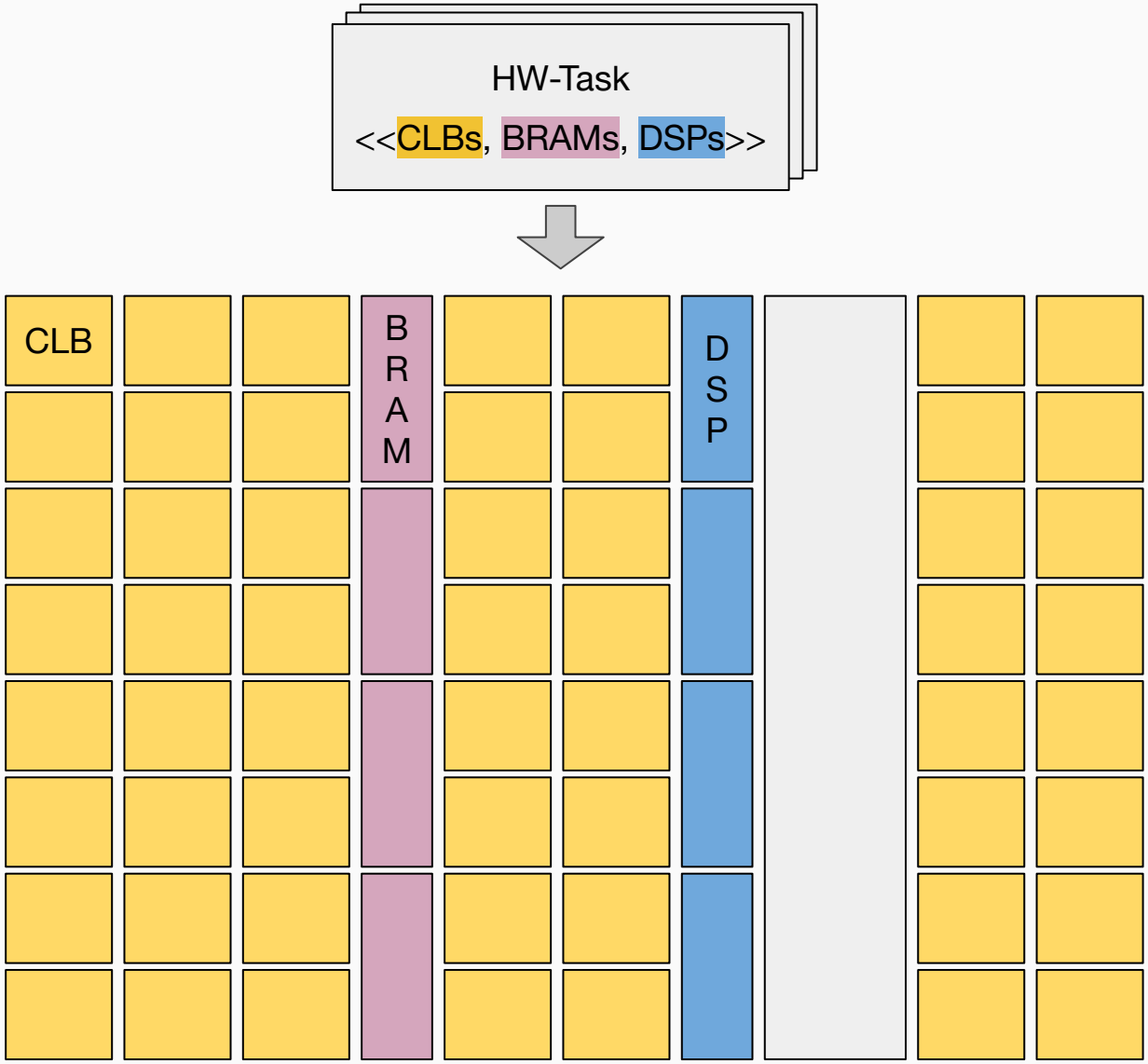
FRED bandwidth regulators / access control overview



FRED bandwidth regulators / access control overview



The FLORA floorplanner overview



The FLORA floorplanner overview