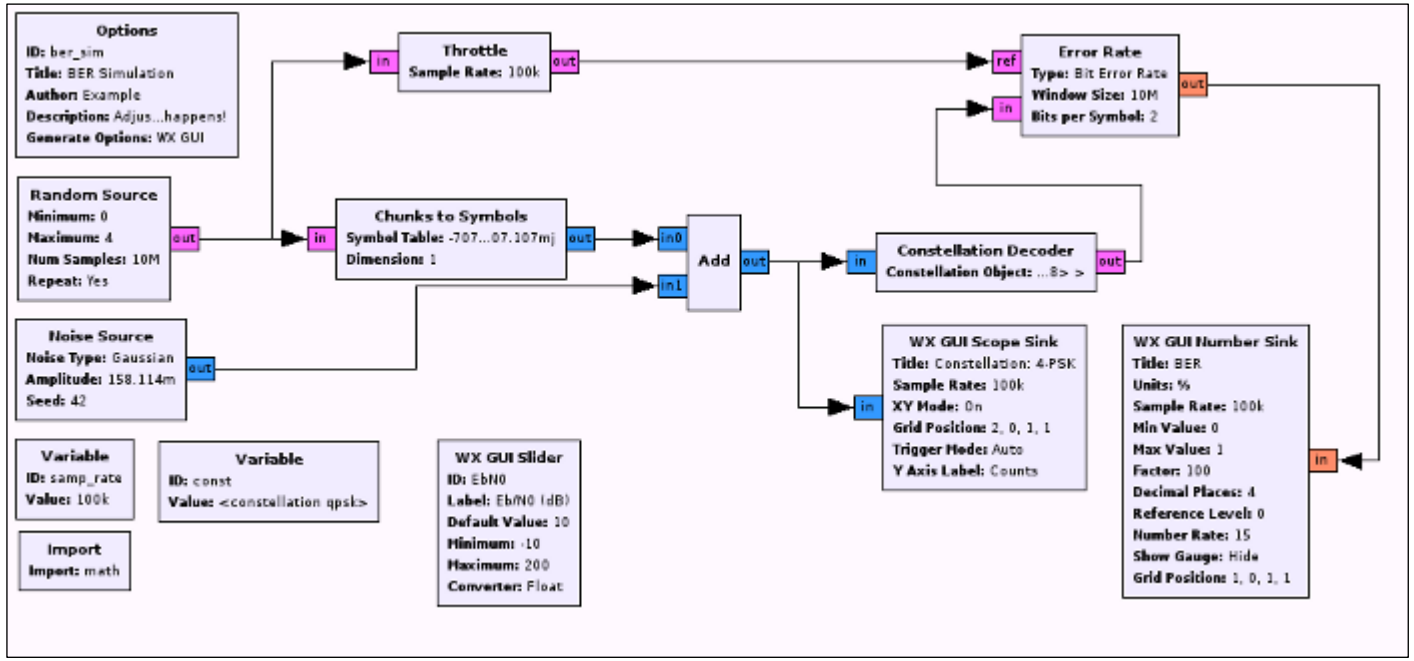
Improving GNU Radio Accelerator Device Dataflow

Presented by: David Sorber

The Problem: Intro

GNU Radio is a software-defined radio (SDR) framework that uses a block-based interface. It includes a library of processing blocks that can be interconnected in various ways to create signal processing flowgraphs.



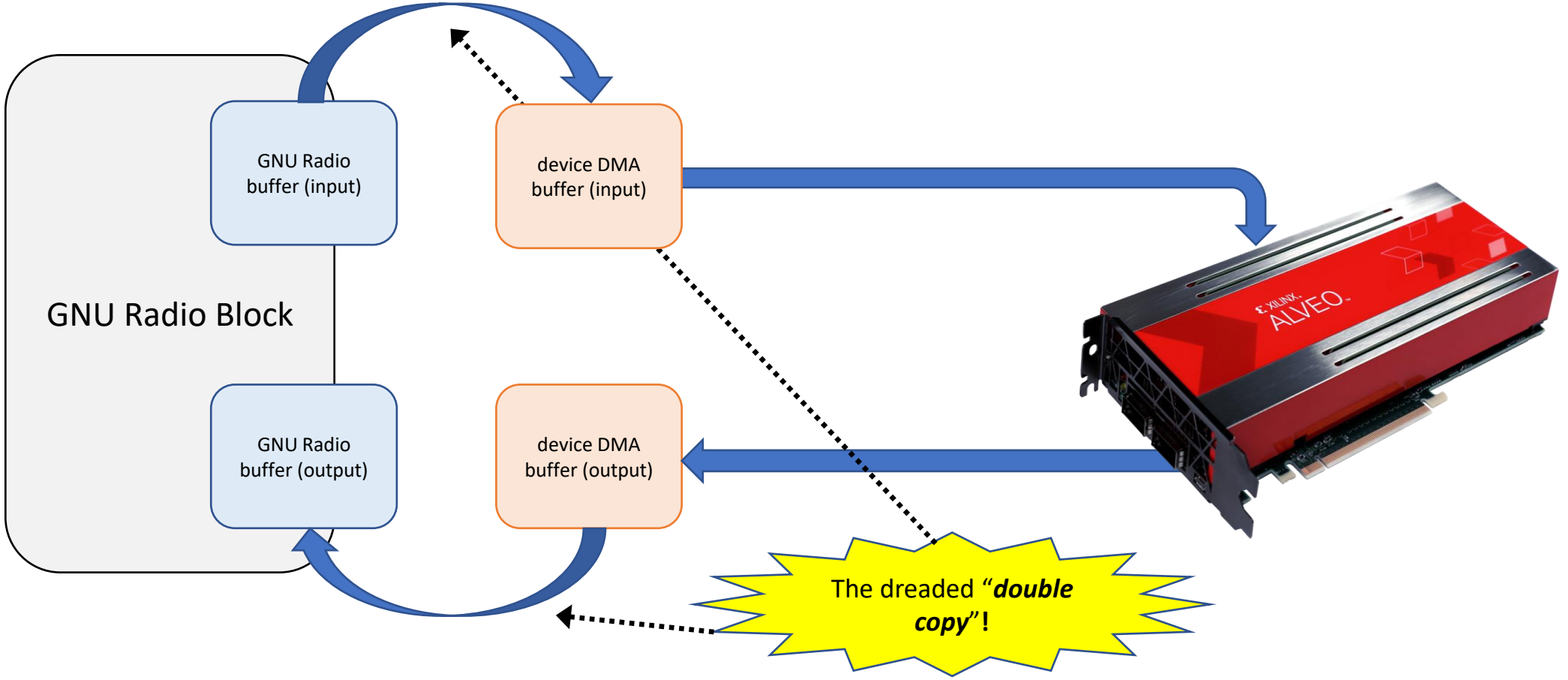
The Problem

Although not explicitly supported, the GNU Radio block-based interface allows signal processing blocks to offload their processing to external “accelerator” hardware including GPUS, FPGAs, and DSPs.

The efficiency of data transfer to and from accelerator devices is often suboptimal because the GNU Radio scheduler controls allocation of memory buffers.

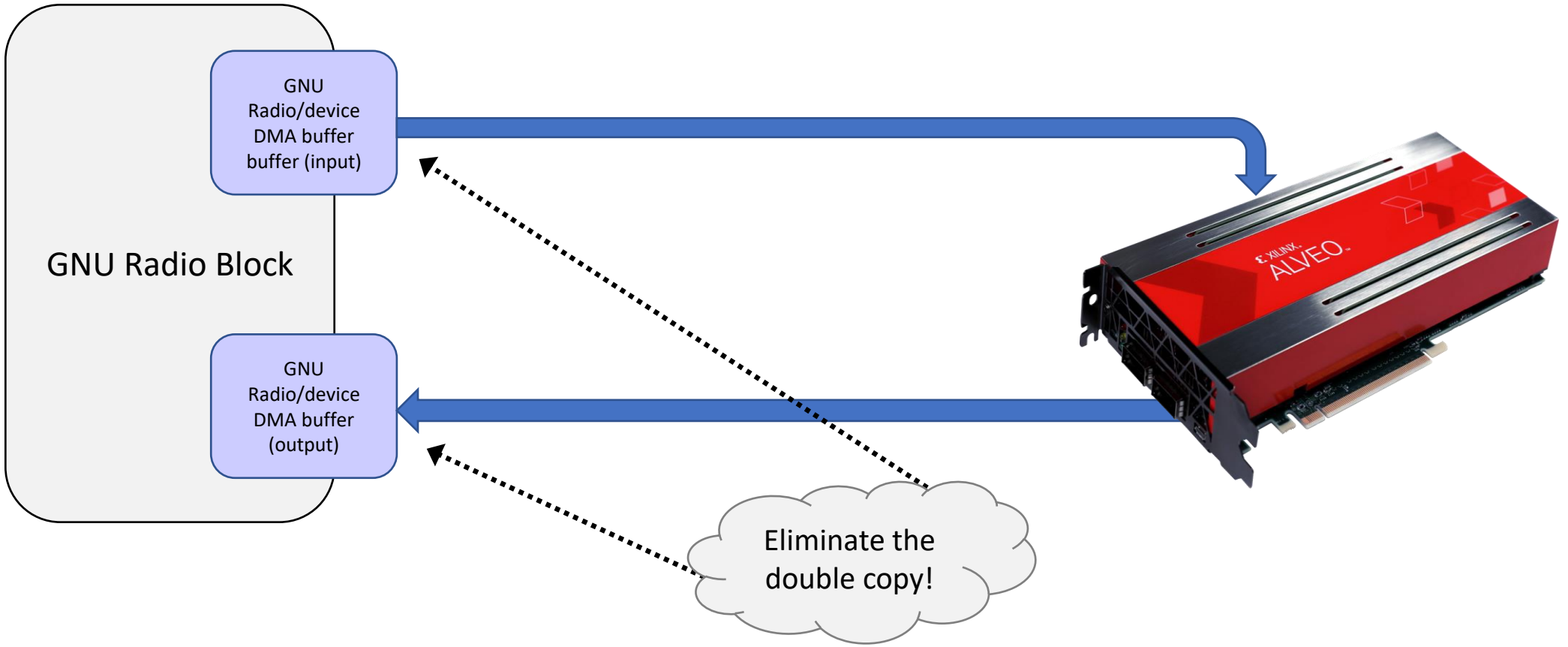
The Problem: Suboptimal

Many accelerator devices require dedicated memory buffers to transfer data; this leads to the double copy problem.



The Problem: A better way

Eliminate the double copy be allowing GNU Radio to manage “custom” (device, DMA, etc.) buffer.

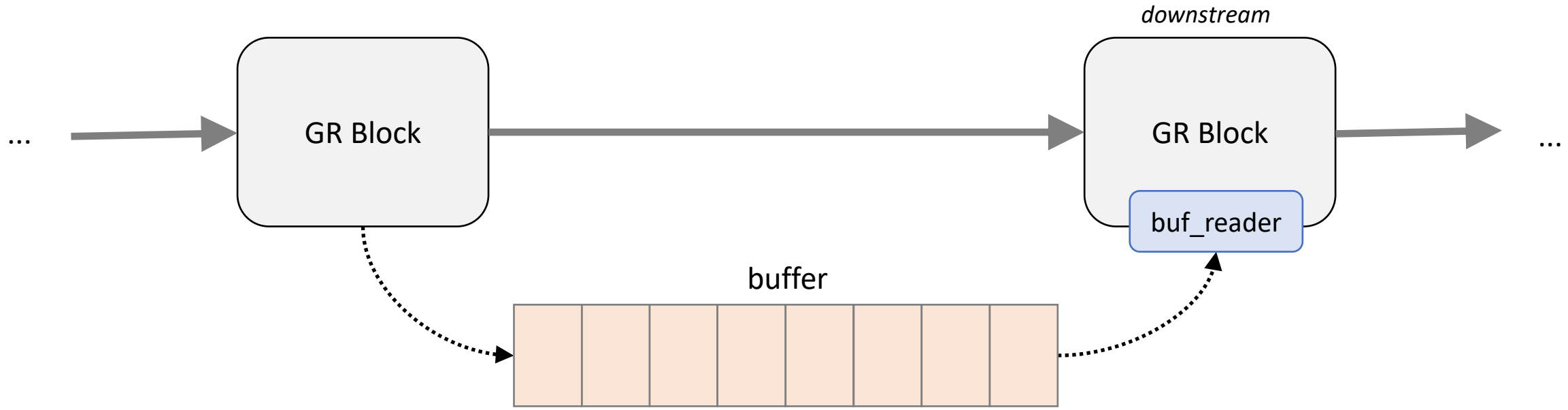


Background: Block interconnect

GR blocks are connected to create a flow graph.

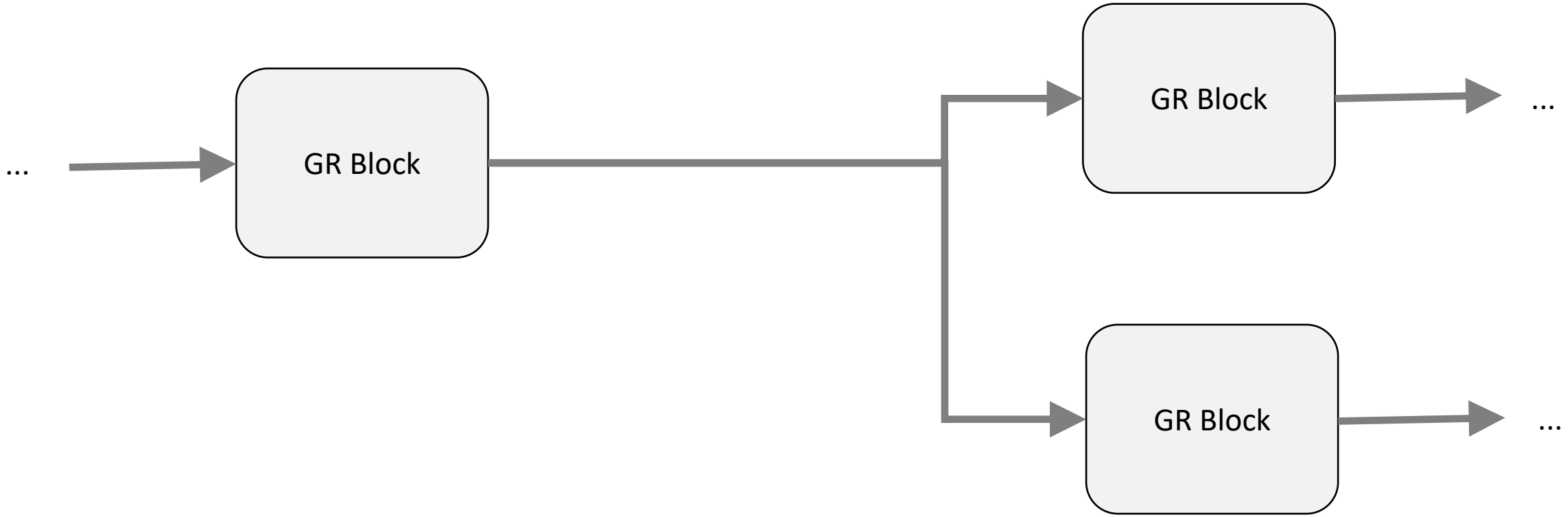
Background: Block interconnect pt. 2

"Under the hood" a block writes its output data to a buffer. A *downstream* buffer reader consumes (reads) data from the same buffer.



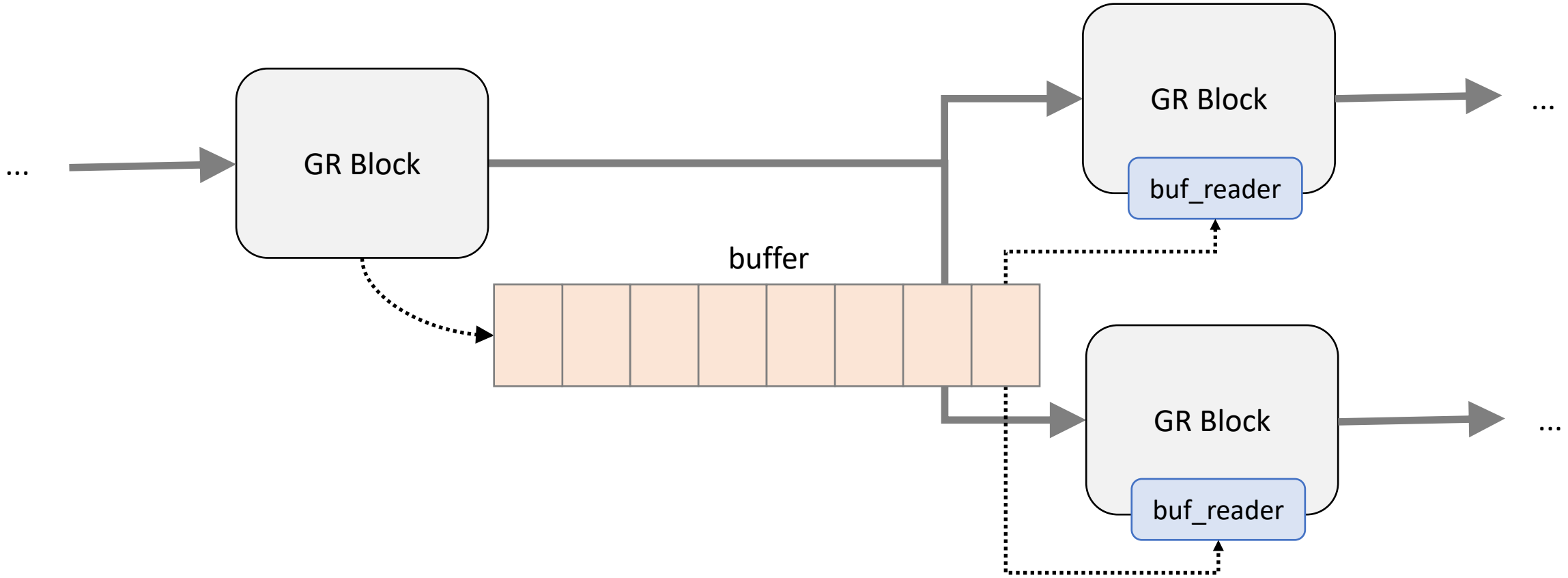
Background: Block interconnect pt. 3

One upstream block and feed multiple downstream blocks.



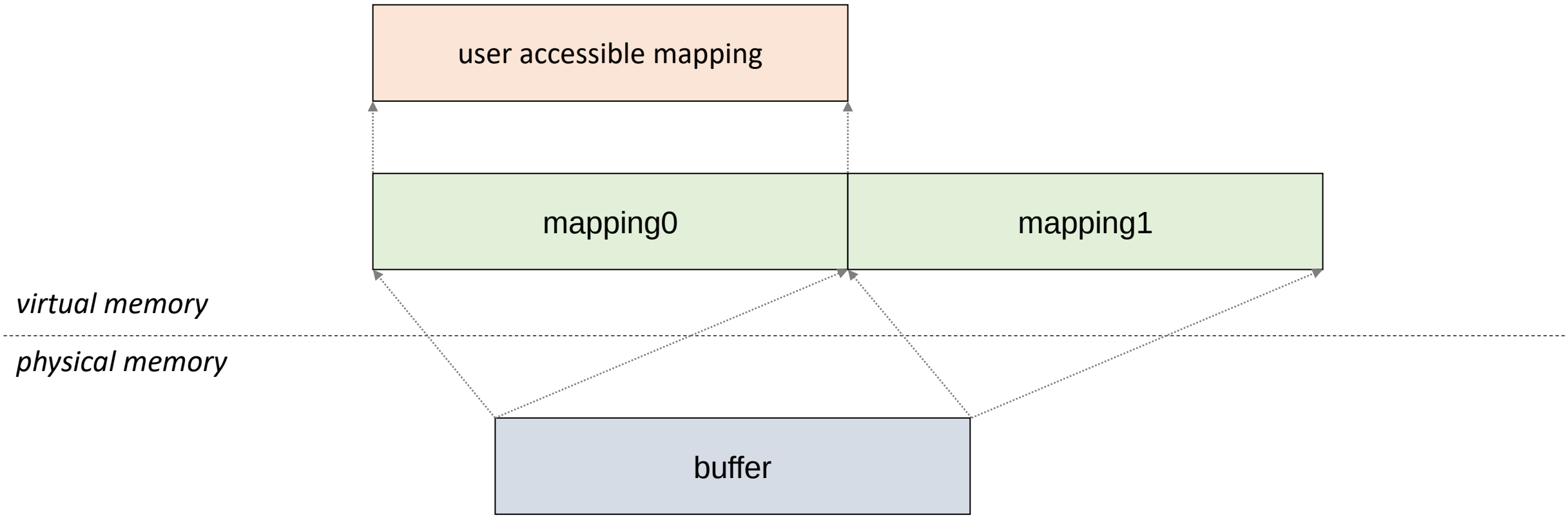
Background: Block interconnect pt. 4

Each downstream block has a separate read pointer.



Background: Double-mapped buffers

GR circular buffers (`vmcircbuf`) use a very clever double mapping scheme.



Design: Goals and Plan

- Preserve compatibility with existing GR blocks (*important!)*
- Create new hardware-agnostic interface to allow users to define and use “custom buffers”
- Work with GNU Radio development team; upstream changes

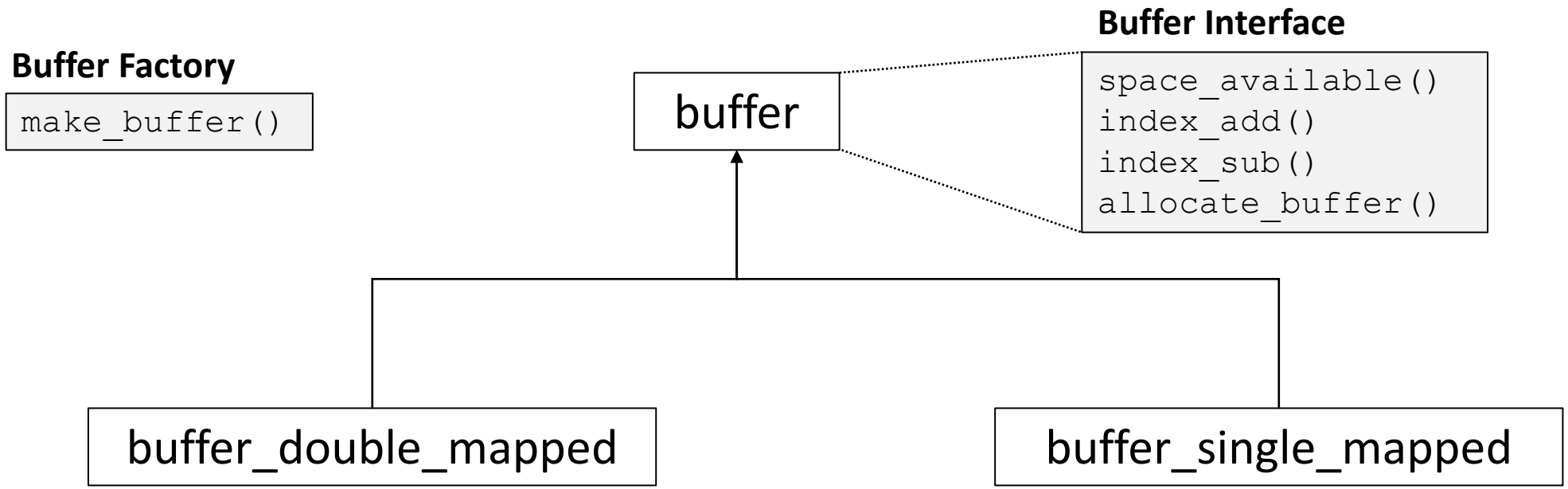
- High-level Development Plan:
  - **Milestone1** – custom buffer interface that eliminates double copy problem
  - **Milestone2** – extend custom buffer interface to support device-to-device (D2D) transfers

Design: Milestone1

- Currently working on milestone1
- 1st draft of changes posted in early December 2020 to ngsched github repo
  - https://github.com/gnuradio/gnuradio-ngsched/
  - Working on next draft, changes not posted yet

- Refactor existing buffer interface and create abstraction for single mapped buffers (complete)
- Create block buffer allocation interface (1st draft)
- Test single mapped buffer class implementation (ongoing)
- Test performance
- Test with many different devices (GPU cards, FPGA cards, embedded hardware, etc.)

Design: `buffer` Abstraction



- Original buffer class interface using `vmcircbuf`

- Abstraction for single mapped buffers
- Wraps custom buffer
- Tested using regular host buffer (i.e. `new char[size]`)

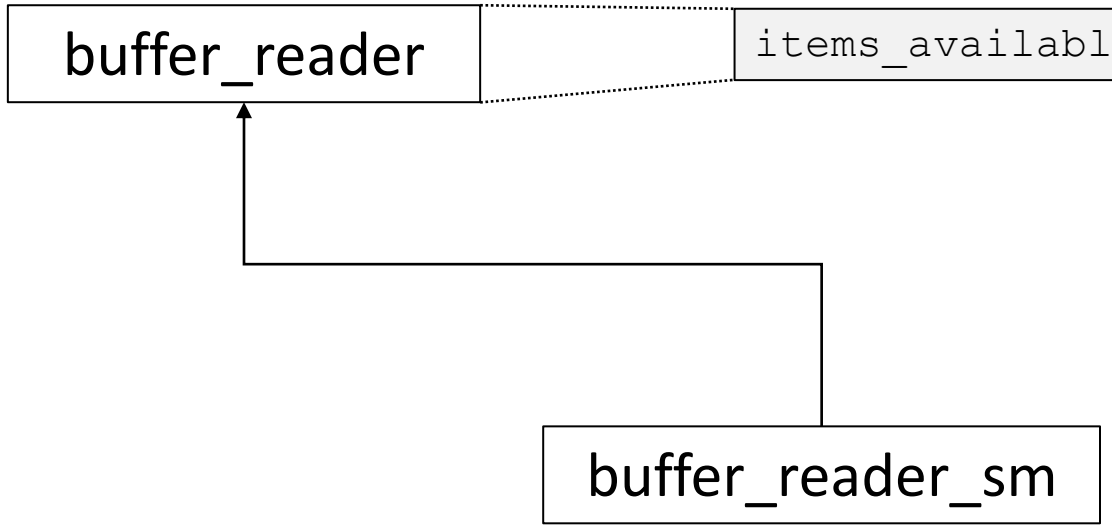
Design: `buffer_reader` Abstraction

Buffer Reader Factory

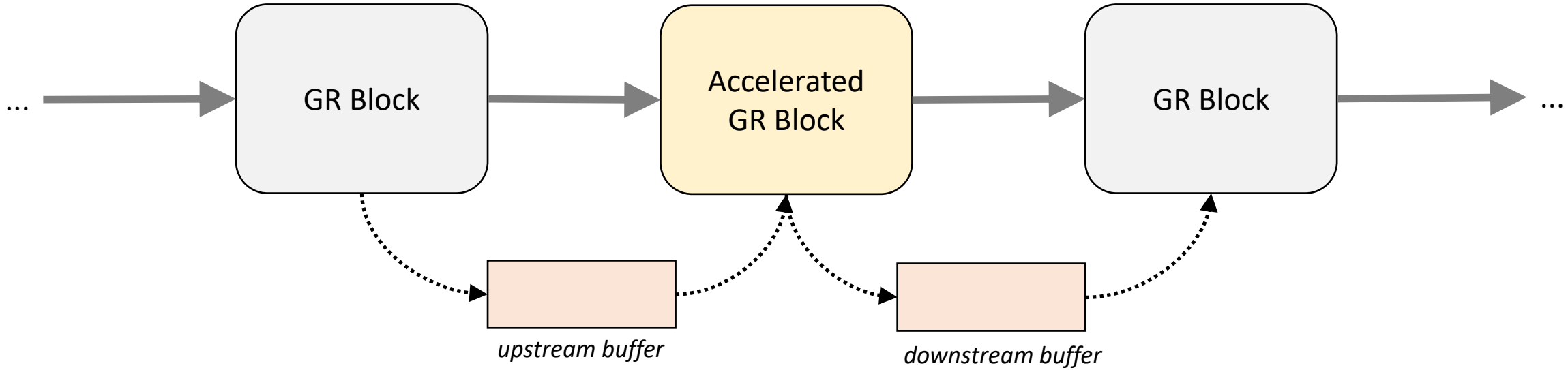
`buffer_add_reader()`

Buffer Reader Interface

`items_available()`



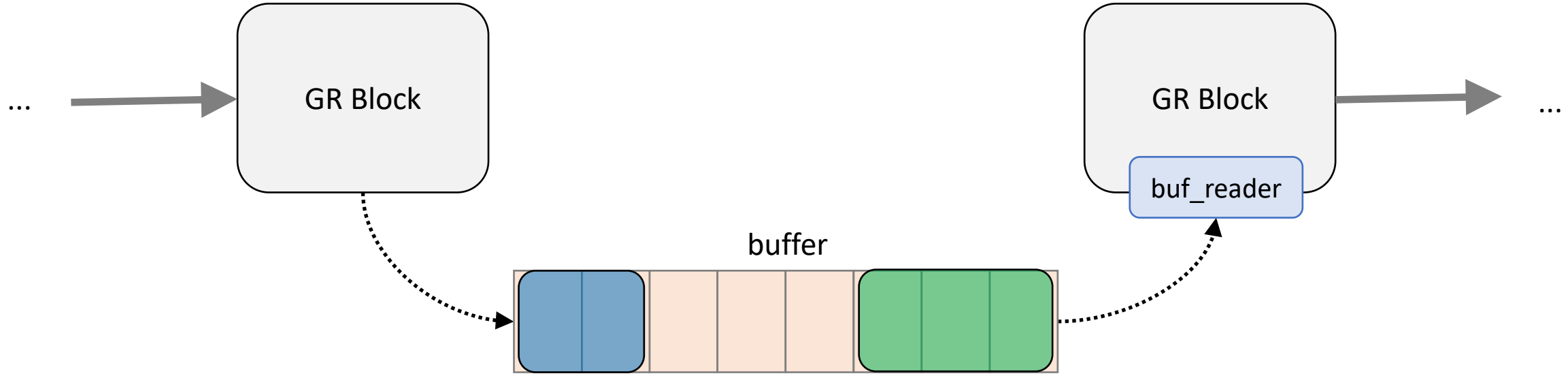
Design: Upstream buffer allocation



- Blocks normally allocate their downstream buffer. Accelerated blocks also need to allocate their upstream buffers.
- Add function to block class to selectively replace its downstream buffer:
  - `replace_buffer(uint32_t out_port, block_sptr block_owner)`
  - Calls `make_buffer()` with downstream block as buffer's block owner
  - Called by `flat_flowgraph::connect_block_inputs()` only if needed

Design: Single-mapped buffer

The single-mapped buffer must manage wrapping explicitly. Size alignment between producer and downstream consumer blocks becomes important.



The read and write granularity of the upstream and downstream blocks may be different! This causes problems when managing wrap around case.

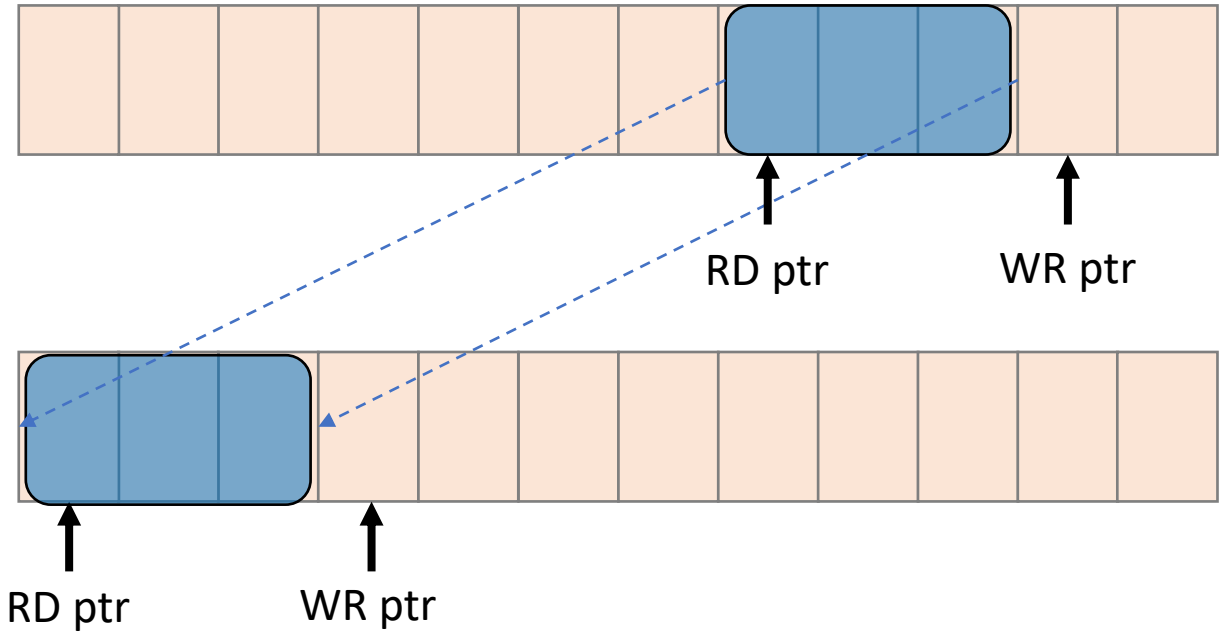
Design: Single-mapped buffer revisited

- It proved too difficult to determine and reconcile differing read and write granularities for a few edge cases.

- **New approach:** handle misaligned wrapping cases explicitly
  - Make reasonable attempt at size alignment up front
  - Add explicitly callback functions, called by the scheduler (block executor), to handle input and output blocked wrapping cases
    - Added `output_blocked_callback()` to `buffer` class
    - Added `input_blocked_callback()` to `buffer_reader` class

Design: Single-mapped buffer output blocked

Write granularity: 3
Read granularity: 1

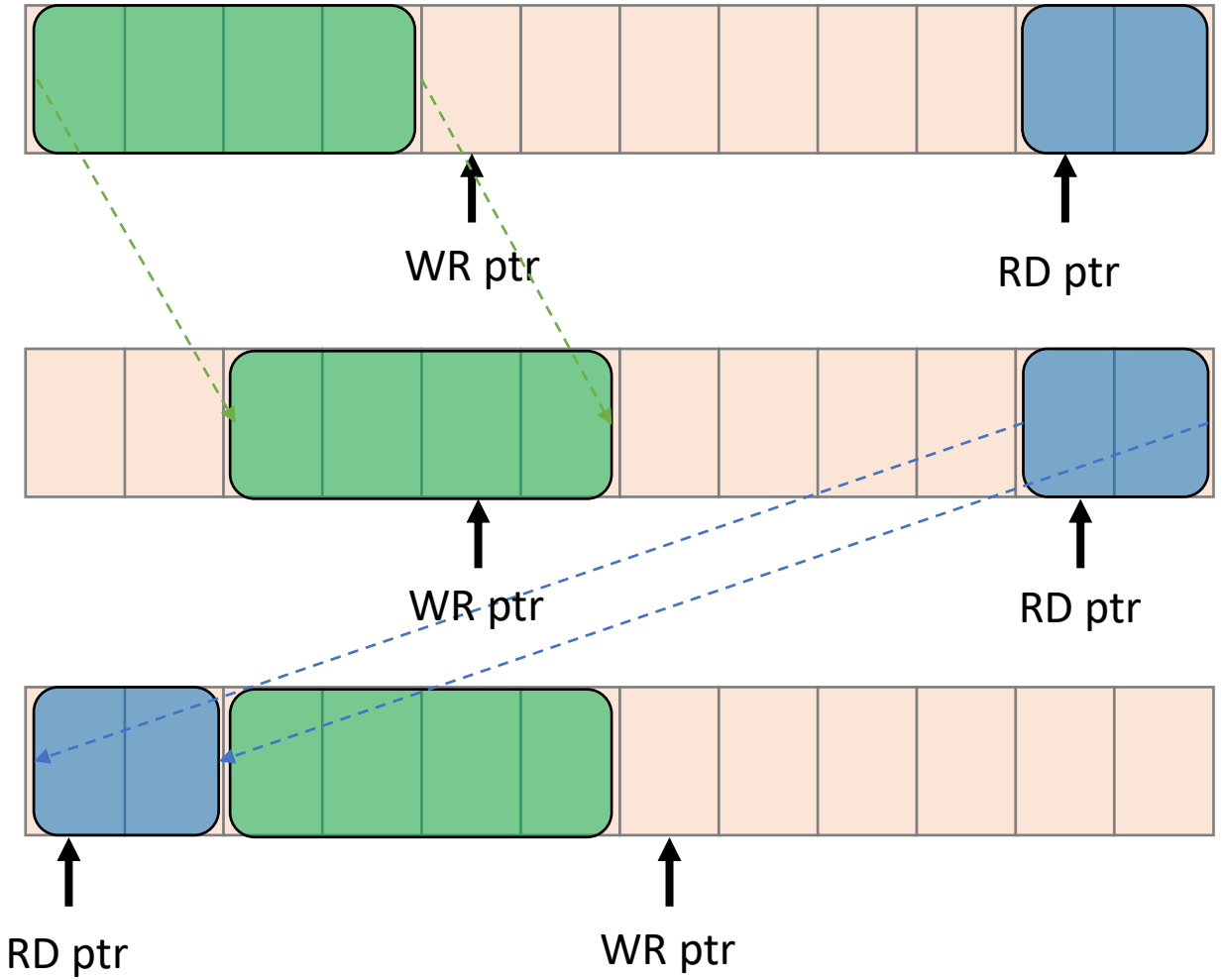


Output blocked!

Copy “to read” data back to beginning of buffer. Adjust pointers. Output unblocked!

Design: Single-mapped buffer input blocked

Write granularity: 1
Read granularity: 3



Input blocked!

Move written data down.

Copy “tail” data back to beginning of buffer. Adjust pointers. Input unblocked!

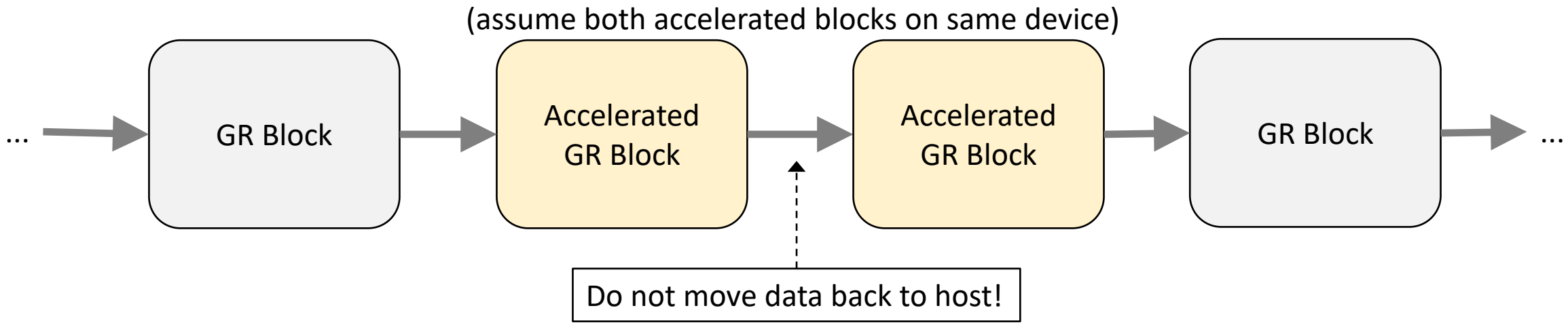
Design: Testing ongoing

- Testing of the single-mapped buffer implementation is ongoing
  - Next draft should be available soon

Design: Buffer allocation interface

- Created simple interface for 1st draft
  - 3 virtual functions added to the block class
    - return buffer type function
    - allocate buffer function
    - free buffer function
  - Simple interface but lacked flexibility
- Received good feedback on 1st draft asking for additional flexibility
- Working on next draft to balance simple interface while adding flexibility

Milestone2: The zerocopy future



- Milestone1 will support back-to-back accelerated blocks (for the same device) but will require data transfer to and from the host between each
- The custom buffer concept can be extended to support device-to-device (D2D) transfers
- Ultimately make accelerated blocks more modular

Conclusion

- Thank you for listening
- David Sorber
- ngsched: https://github.com/gnuradio/gnuradio-ngsched/

- Milestone1 - work is ongoing
- Milestone2 - beginning later this year