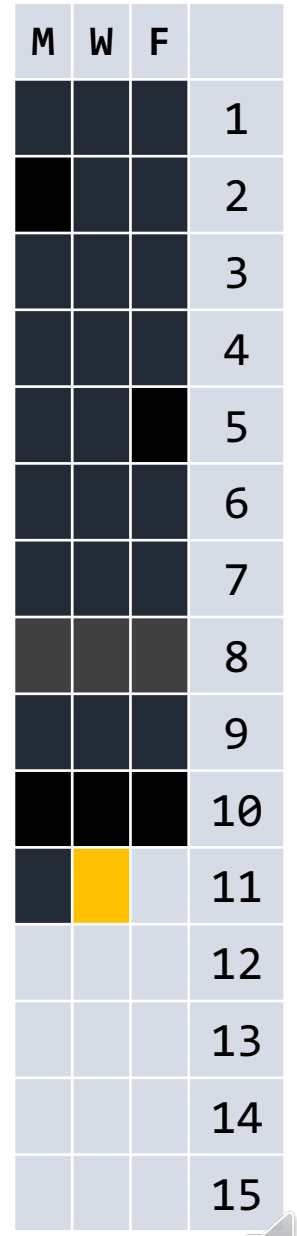
CMOR 421/521:

What we haven’t covered



Topics

- Cartesian and graph topologies
- MPI One-sided communication
  - MPI Windows
  - How to use them
  - When to use them

Recall…

- OpenMP Tasks were the last topic for the Non-Communicating Parallelism section
- They were the tool to use when the computation pattern was complex or random
- **MPI One-sided communication is similar:**
  - It is the tool to use when the *communication* pattern is complex or random

Other MPI topologies

- Cartesian topology
  - N-dimensional grid
  - Periodicity
  - Lower-dimensional slicing
- Graph topology
  - Each “vertex” is an MPI process with “edges” between communicating processes
  - Edge weights to represent how much information is passed

One-Sided Communication

- One-sided communication allows for one rank to get data from other ranks without their needing to be involved *in the communication*
  - They still have to participate, so it is not truly one-sided
- Since other ranks do not need to participate, the communication pattern:
  - Does not need to be known beforehand
  - Does not run the risk of communication-based deadlocks

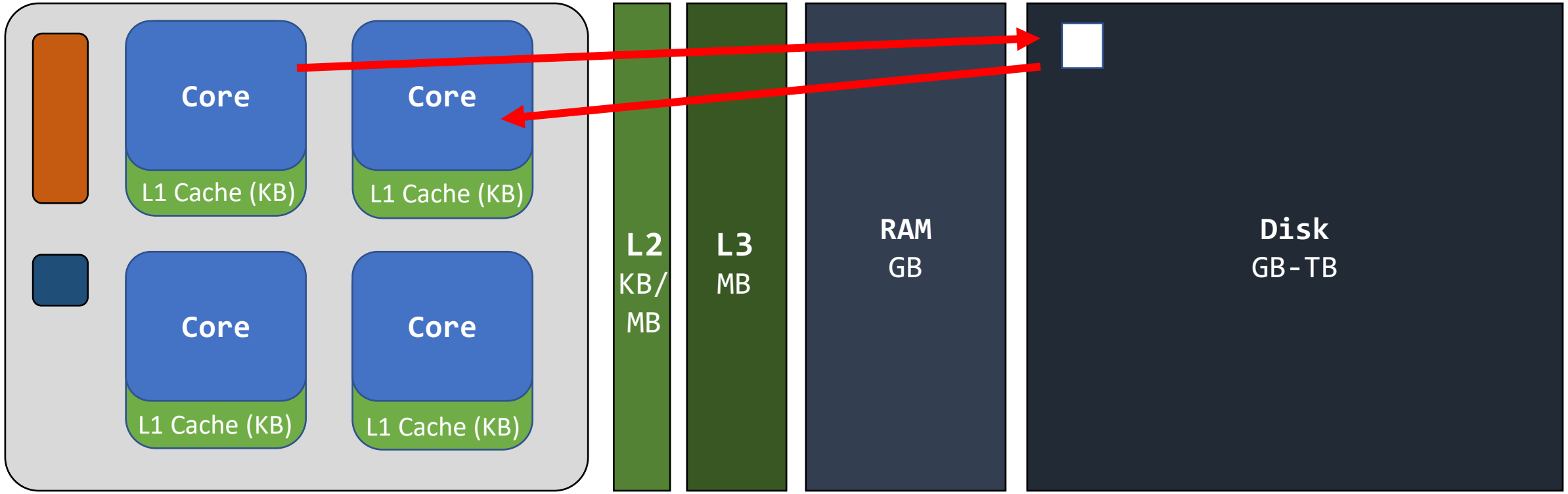
What is One-Sided Communication?

- One-sided communication is also called RMA
  - RMA = Remote Memory Access
- It is one rank directly accessing another rank's memory rather than exchanging buffers of data
  - Particularly powerful on a system that has shared memory
  - Low-latency, high-bandwidth "communication" in that case

Computer Architecture

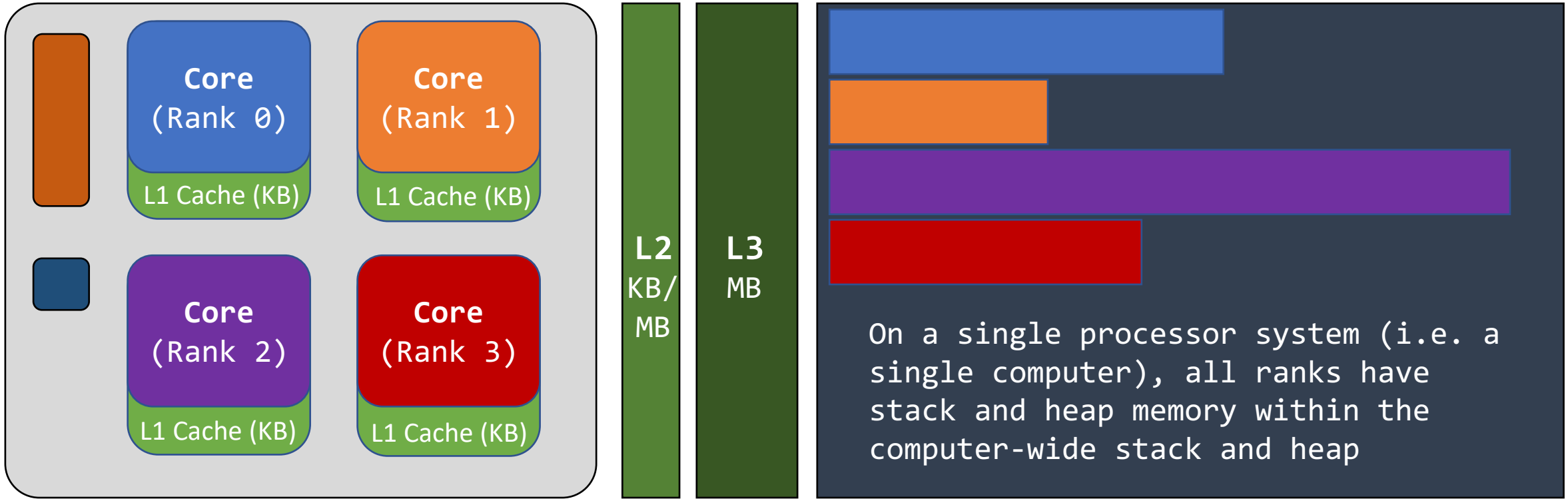
Example architecture of a single processor (single computer)

With OpenMP/shared memory, threads communicated by changing (shared) variable values



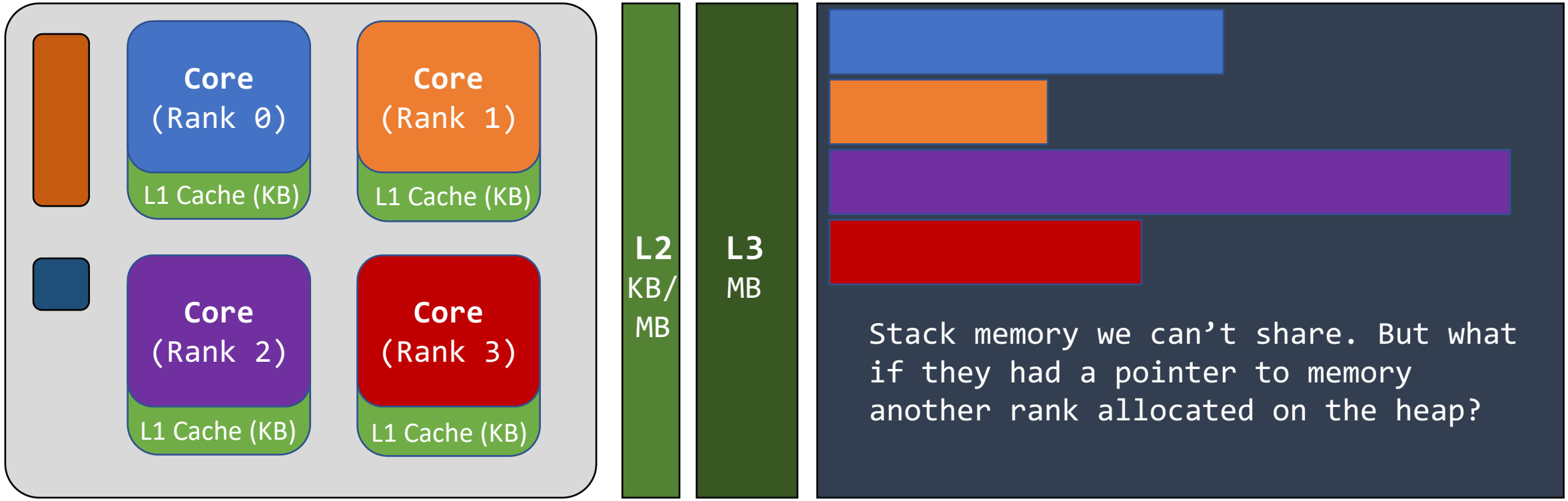
Computer Architecture & RMA

Assume: Num ranks = num cores



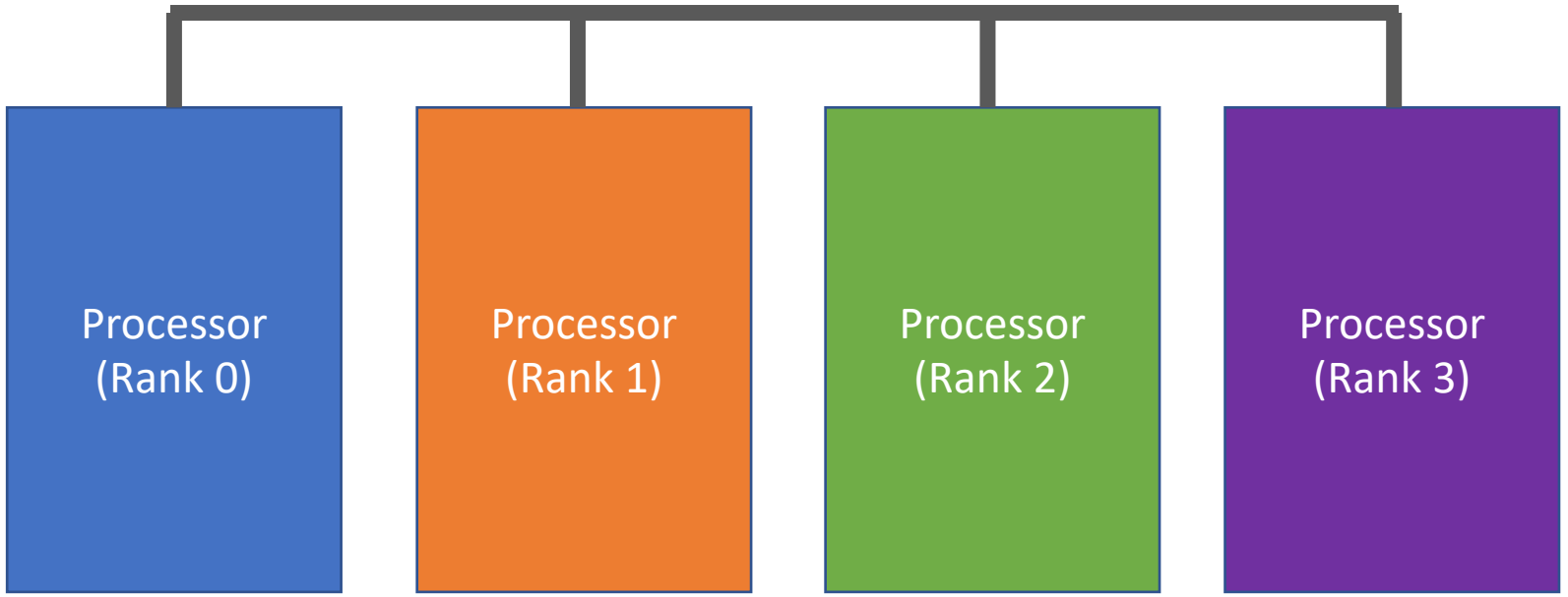
Computer Architecture & RMA

Assume: Num ranks = num cores



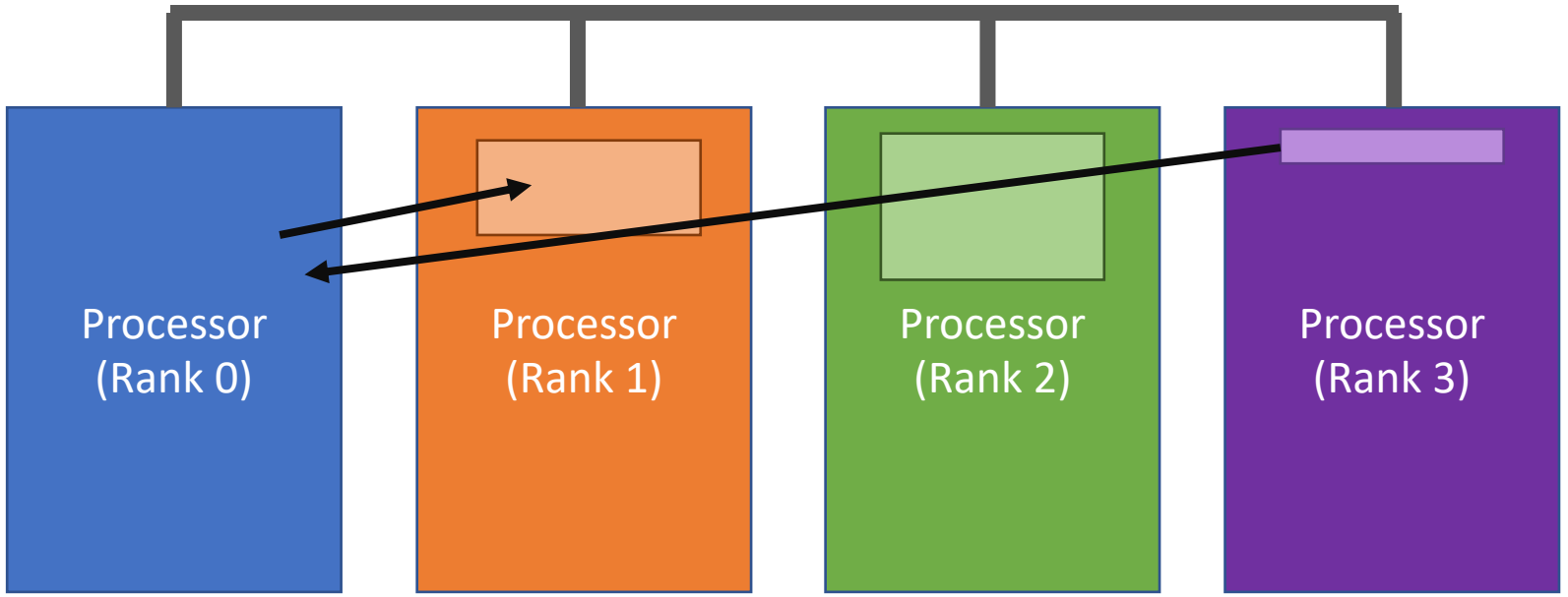
Computer Architecture & RMA

What if we had multiple processors? (No shared memory)



Computer Architecture & RMA

Processors can expose part of their memory to others for read and write access



One-Sided Communication

- One-sided communication lets ranks “expose” a portion of their memory for other ranks to read from and write to
  - **Danger:** wherever there is parallel read/writes, there is the possibility of race conditions
  - Nothing is ever free
- Other ranks can then read and write as they please (subject to synchronization as needed)

One-Sided Communication

- Exposing a section of memory is why “one-sided” communication isn’t truly one-sided
  - Ranks have to participate by exposing memory even if they don’t read or write to another rank’s memory (i.e. they don’t communicate themselves)
- Synchronization: At what point did the memory get exposed?
  - Not only can we get race conditions, the memory may not even be available yet!

MPI Windows

- MPI supports RMA via the MPI_Window object
- The window reflects the section of exposed memory plus tools to check on its status
  - i.e., whether it is valid to read/write from a window
- MPI provides methods for creating, reading from, writing to, synchronizing, and destroying windows

MPI Window Functions

- Creating: MPI_Win_create
  - Note: there are other, more complex ways to create MPI windows; we'll just use this
- Destroying: MPI_Win_free
- Reading: MPI_Get
- Writing: MPI_Put
- Synchronizing: MPI_Win_fence, MPI_Win_lock/unlock,

Creating a MPI Window

```
int MPI_Win_create(
    void* base,
    MPI_Aint size,
    int displ_unit,
    MPI_Info info,
    MPI_Comm comm,
    MPI_Win* win
);
```

Argument	Description
base	A pointer to the memory to expose Must already be allocated!
size	How many BYTES to expose NOT ELEMENTS
displ_unit	How many bytes per element Use: sizeof(<datatype>)
info	
comm	The communicator to create the window in
win	A pointer to a window object (to be initialized by this function)

Destroying/Freeing a MPI Window

```
int MPI_Win_free(MPI_Win* win);
```

Argument	Description
win	A pointer to a window object (to be freed by this function)

Example Window Creation

```
int main() {
    MPI_Init(NULL, NULL);
    int n = 100;
    int* window_mem = new int[n];
    MPI_Win window;

    MPI_Win_create(window_mem, n*sizeof(int), sizeof(int),
         info, MPI_COMM_WORLD, &window);

    MPI_Win_free(&window);
    delete[] window_mem; // Delete after the window!
    return 0;
}
```

Who Has to Create a Window?

- **MPI_Win_create is a collective operation!**
- All ranks must participate
- If a rank doesn't have memory to expose, it can use base = NULL and size = 0

- **Note:** we never said how many elements to use
  - We always had to specify this before with Send/Recv & Co.
  - This is just for creating a window; it does no communication yet

Reading from a MPI Window

```
int MPI_Get(
    void* origin_addr,
    int origin_count,
    MPI_Datatype origin_type,
    int target_rank,
    MPI_Aint target_displ,
    int target_count,
    MPI_Datatype target_type,
    MPI_Win win
);
```

Argument	Description
origin_addr	Pointer to where to copy the data **to** on the calling rank
origin_count	How many entries to copy to the calling rank
origin_type	Data type of the data at the calling rank
target_rank	The rank to access
target_displ	Where in the target rank's exposed memory to start reading from IN BYTES
target_count	How many elements to read from the target
target_type	Data type of the data at the target rank
win	The window to use

Writing to a MPI Window

```
int MPI_Put(
    const void* origin_addr,
    int origin_count,
    MPI_Datatype origin_type,
    int target_rank,
    MPI_Aint target_displ,
    int target_count,
    MPI_Datatype target_type,
    MPI_Win win
);
```

Argument	Description
origin_addr	Pointer to where to copy the data **from** on the calling rank
origin_count	How many entries to copy to the calling rank
origin_type	Data type of the data at the calling rank
target_rank	The rank to access
target_displ	Where in the target rank's exposed memory to start reading from IN BYTES
target_count	How many elements to write to the target
target_type	Data type of the data at the target rank
win	The window to use

Synchronizing MPI Windows

```
int MPI_Win_fence(
    int assert,
    MPI_Win win
);
```

Argument	Description
assert	Condition to assert before moving forward For us: just use 0
win	The window to synchronize

RMA Synchronization with Fences

- **You should call MPI_Win_fence BEFORE and after any RMA communication (and before deletion)**
- Two calls to MPI_Win_fence start and end a RMA epoch
  - Access epoch: ranks reading/writing from/to a target rank's memory
  - Exposure epoch: the target rank exposing its memory for access
  - Whether it is an access or exposure epoch depends on the rank you are referring to
- The goal is to ensure the window remains valid

RMA Synchronization

- **There are several ways to synchronize windows**
- Using MPI_Win_fence
- Also have locks:
  - Locks can be customized to lock just some ranks but not others -> more flexible
  - Can have more overhead
- Also have “itemized” synchronization functions:
  - post->start->complete->wait
- Also have flushing: single rank “synchronization”

RMA vs Send/Recvs

- **RMA is optimal for many situations**
  - e.g., when you actually have shared memory available
- Optimality is a blessing and a curse
  - Optimal *for who?*
  - Optimality often means a lack of generalizability
- What if you code need to run on a system without shared memory? What if the communication pattern is sparse?
  - Send/Recvs will be better/more portable in these situations

Next Up

This is the end of Communicating Parallelism

Next: Multi-Node Parallelism

- Programming on multiple, multi-core processing units
- In the next few weeks:
  - GPU programming
  - MPI* and Measuring Performance
  - Example applications, quick peek at parallelism in other languages