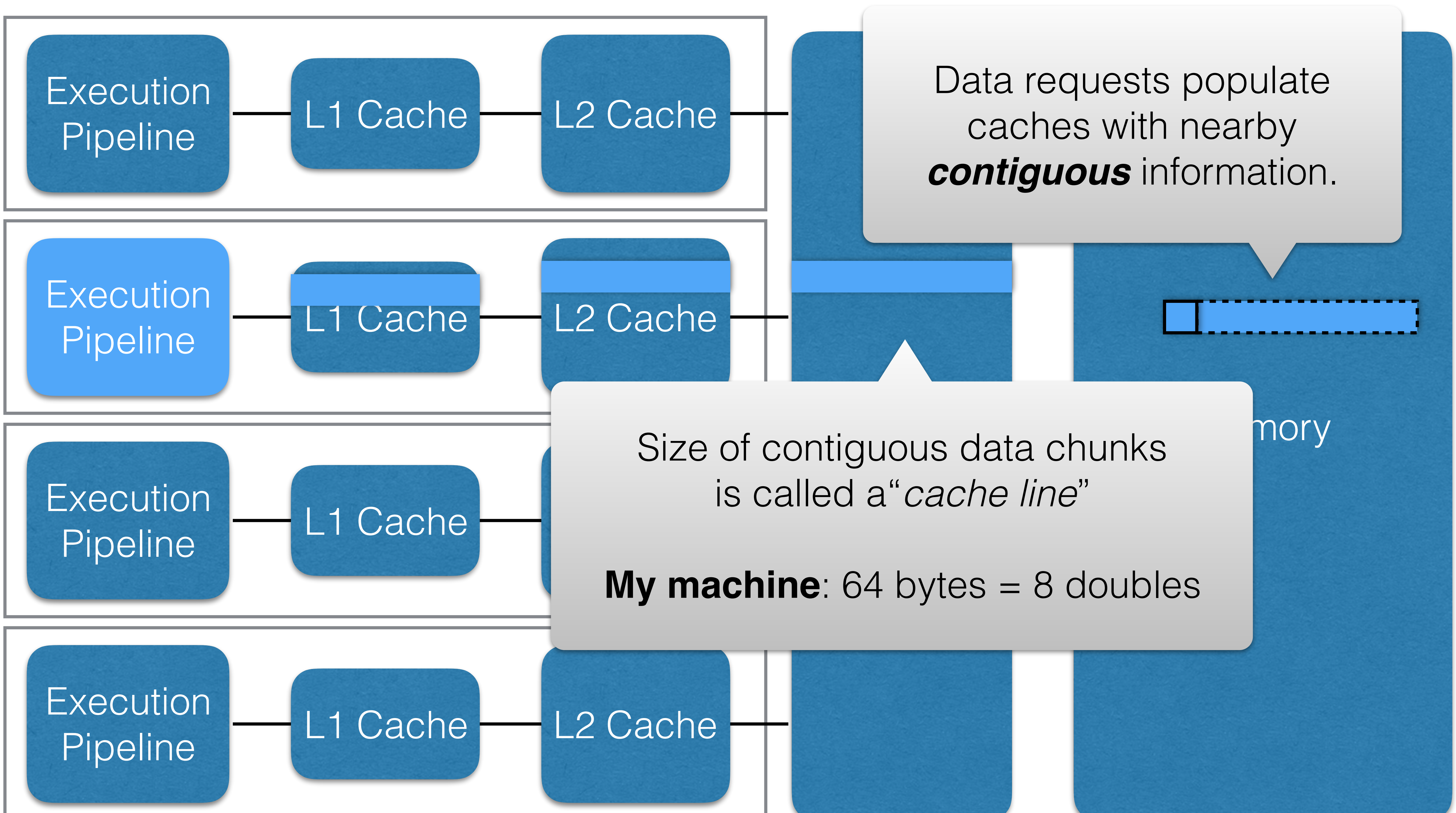
Lecture #15 - MPI Domain Decomposition

AMath 483/583

Announcements

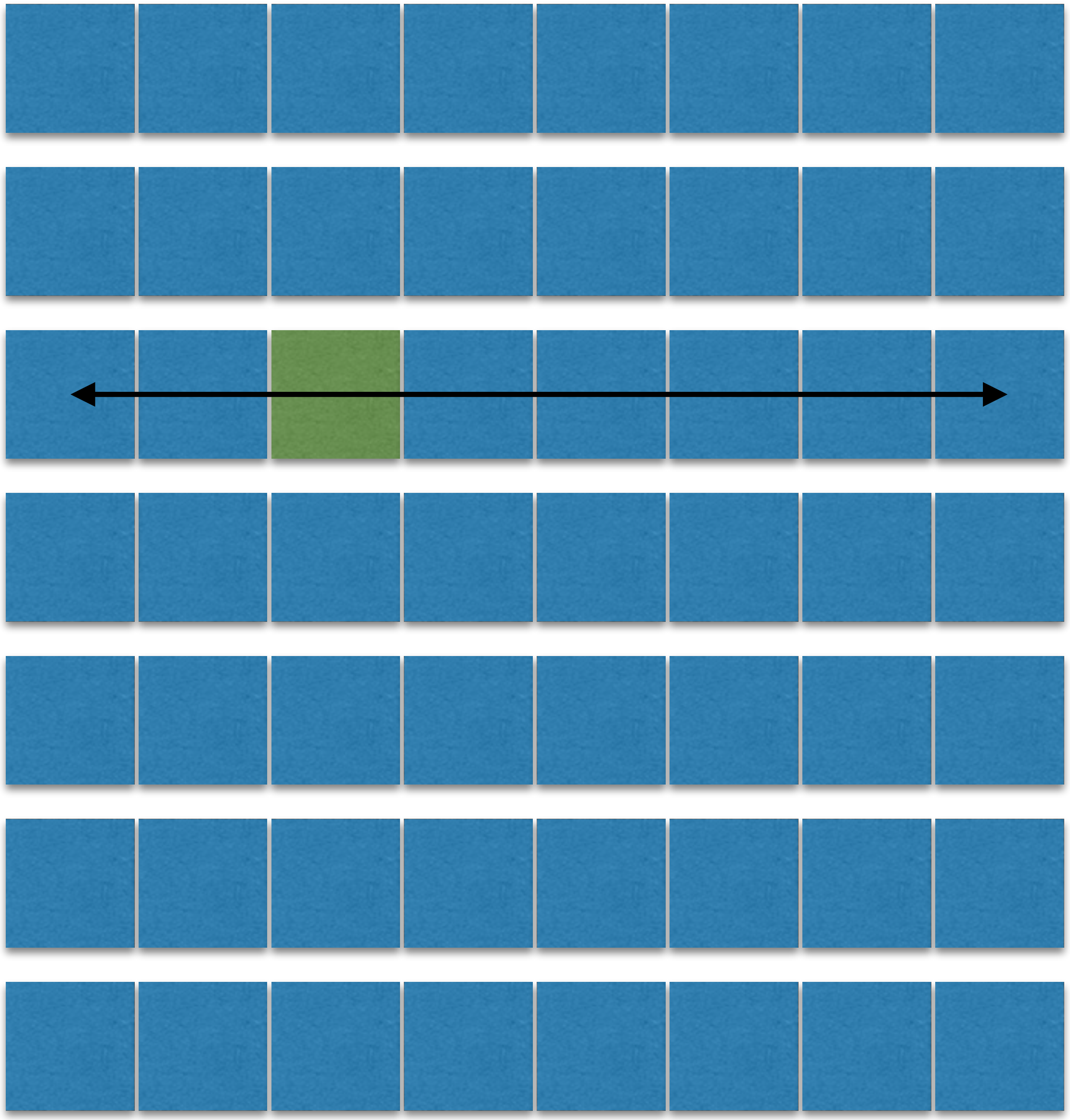
- Homework #3 — Due Friday *(get started if you haven't already! SMC loads are high before due date)*
- Today:
    - brief aside on cache lines
    - MPI (parallel) design patterns

Memory Layout



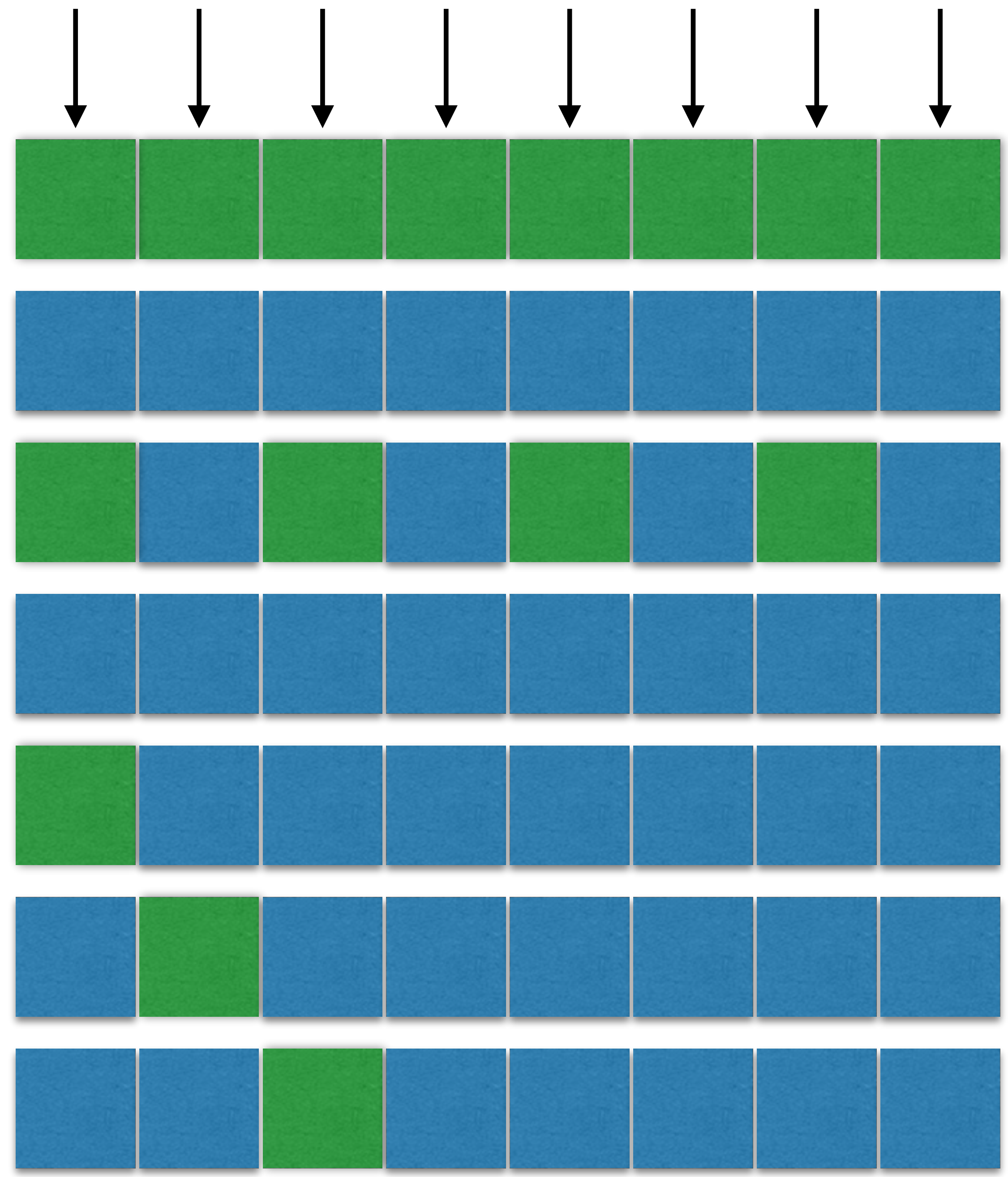
Cache Lines

- Imagine RAM laid out in rows of cache lines, each 8 doubles long.
- When you access a single piece of data the entire line is sent to the cache.



Cache Lines

- Why contiguity of access is good.
- Sub-optimal access patterns don't take advantage of cache lines.

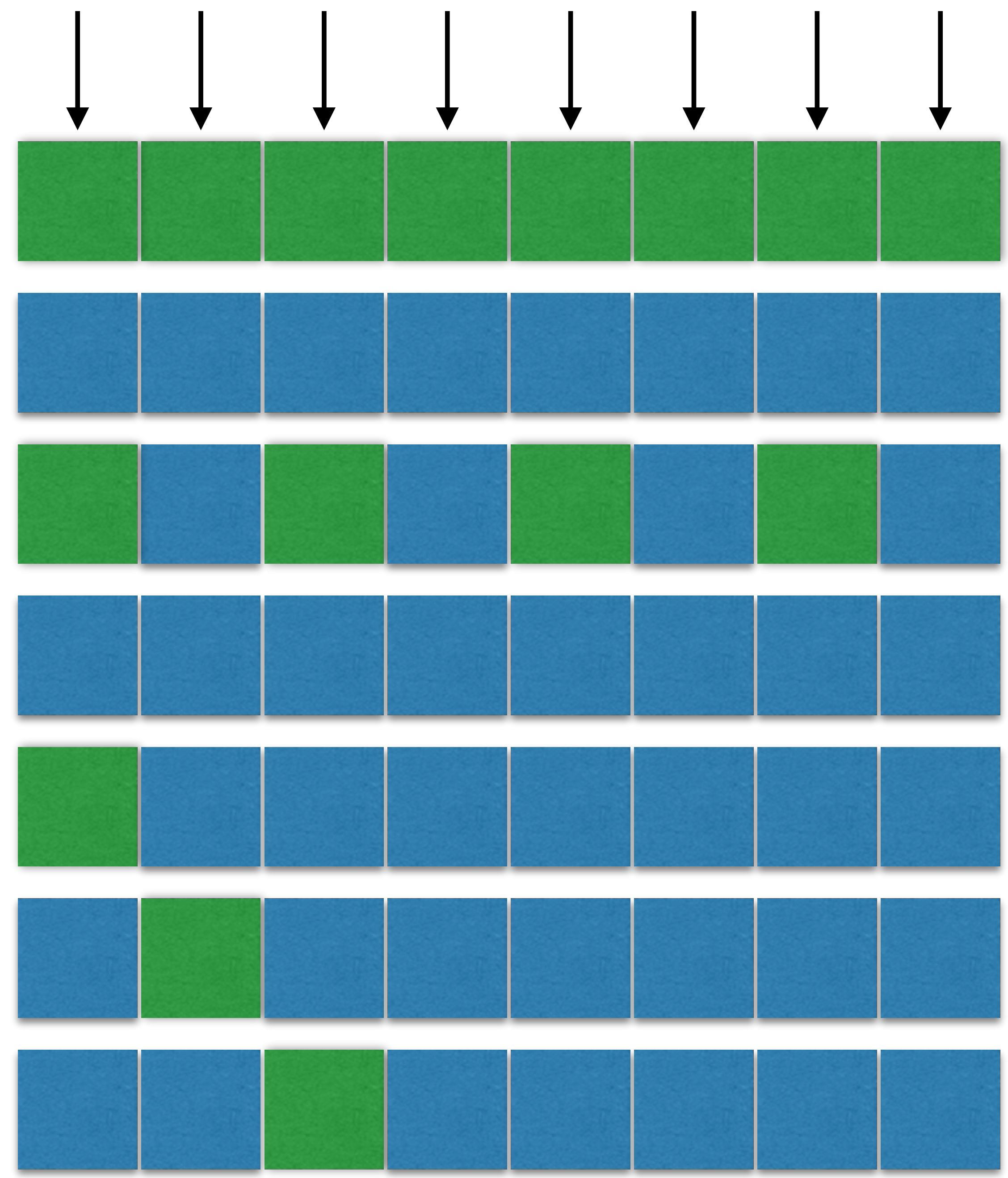


Cache Lines

```
for (size_t i=0; i<N; ++i)
  arr[i] = // input
```

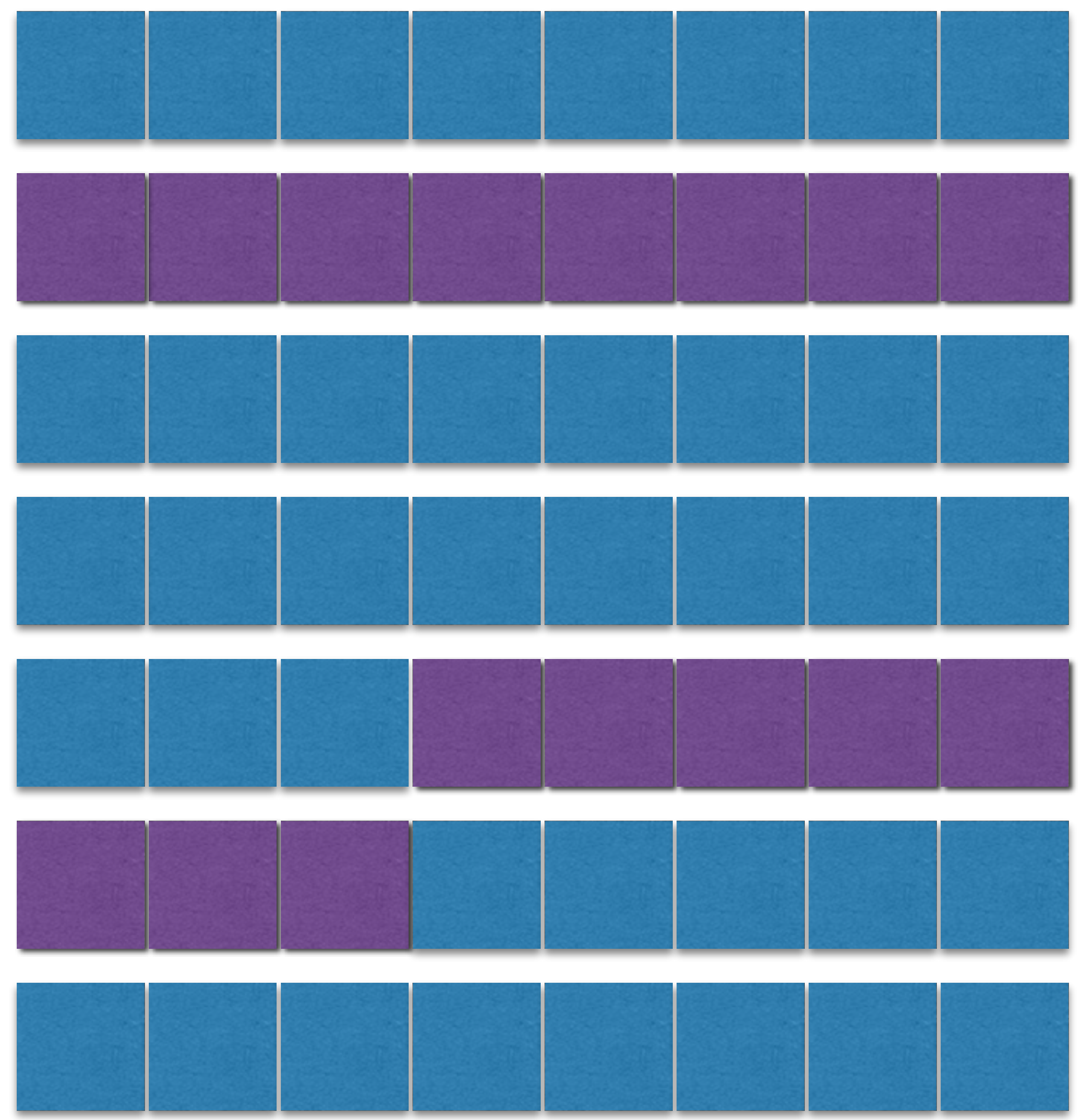
```
for (size_t i=0; i<N; i +=2 )
  arr[i] = // input
```

```
for (size_t i=0; i<N; ++i)
  arr[i*N+i] = // input
```



Cache Lines

- “Cache-alignment” — make most use of cache
  - `posix_memalign()`
  - `aligned_alloc()`
- C struct packing
- “Power of two” rule

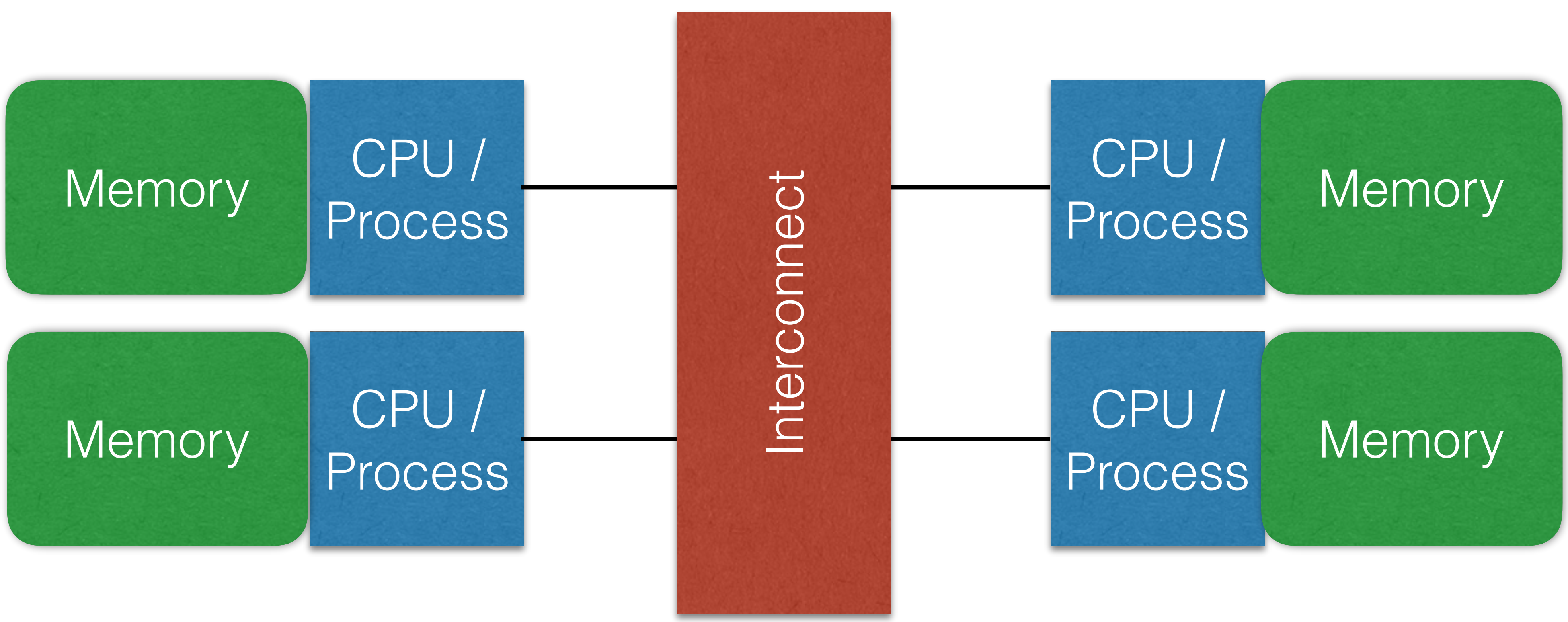


MPI and Domain Decomposition

(Back to regular lecture)

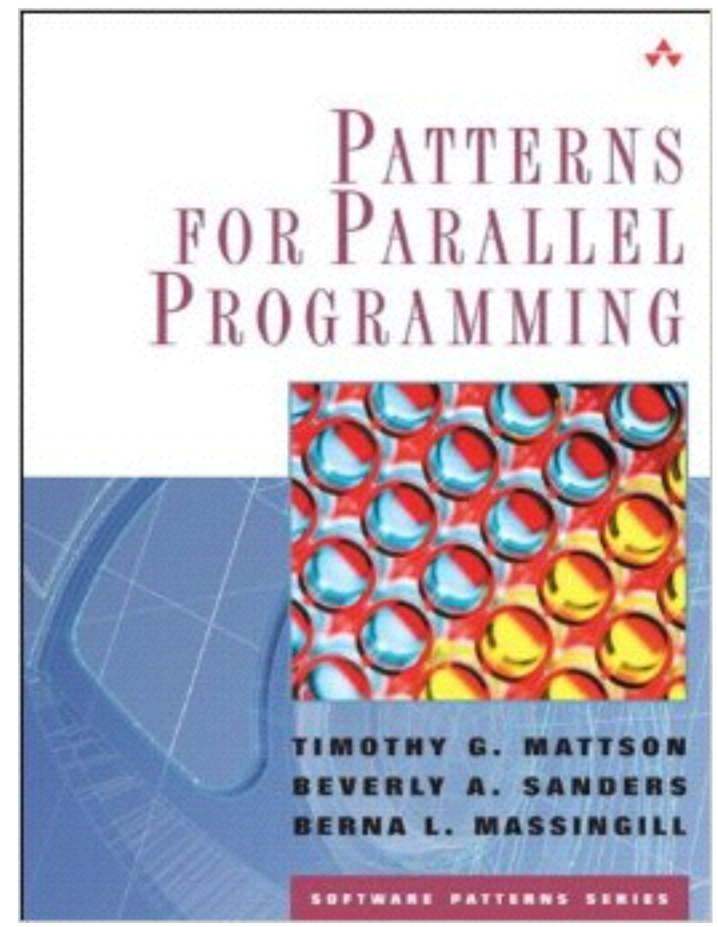
Reminder

- MPI is used in distributed memory environments
- If working in shared memory environment, probably better off using OpenMP



Domain Decomposition

- Common “design strategy” for MPI programs
- Each process only computes over subdomain
- May need to communicate on “boundaries”

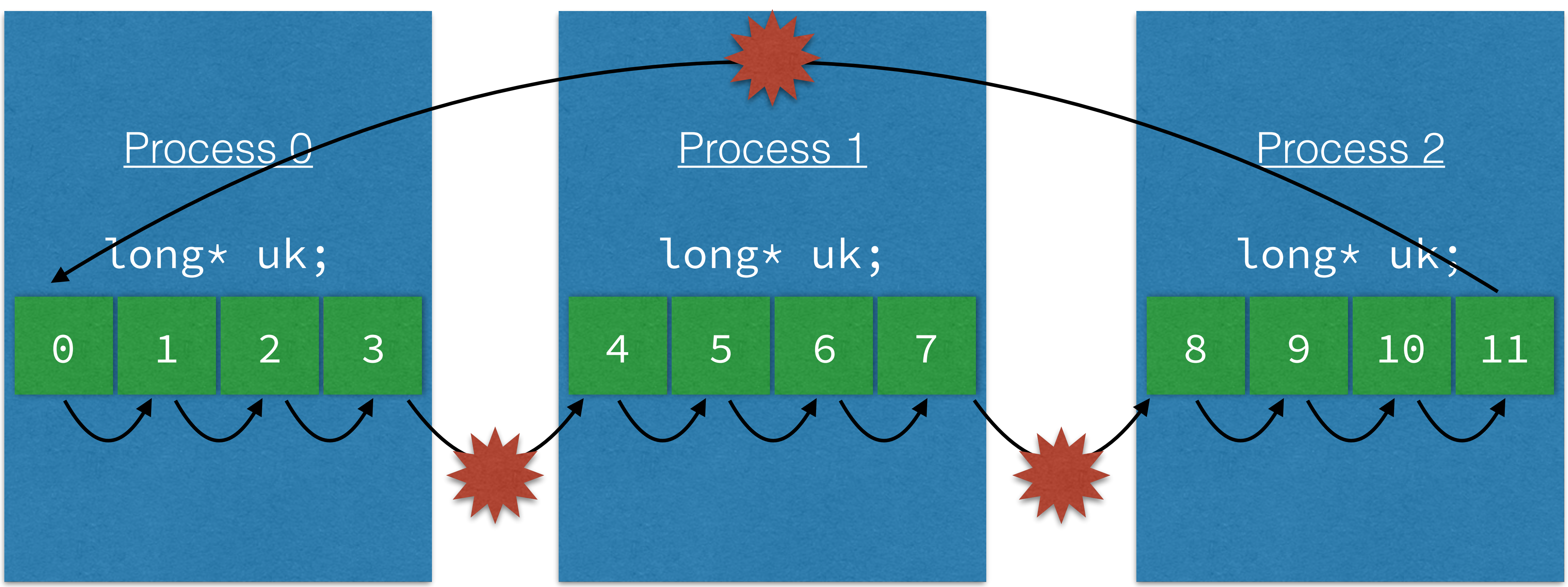


Problem Statement

- Each process allocates an array of data
- Want to “shift” the data right

Problem Statement

- Explicit Communication:
  - Process `k` receives data from Process `k-1`
  - Process `k` sends to Process `k+1`



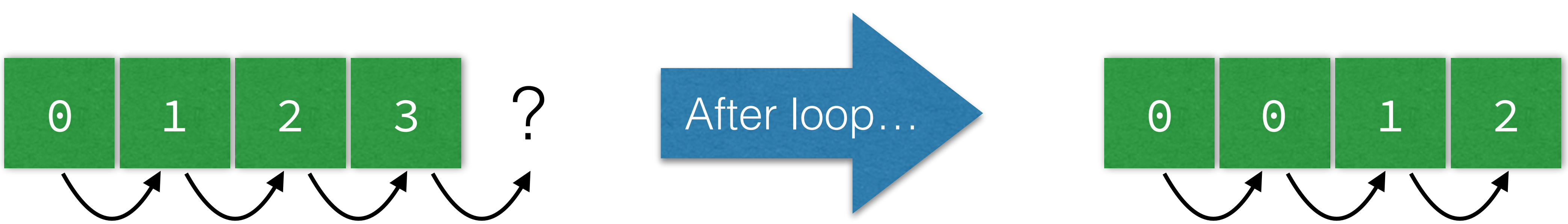
Problem Statement

- Desired result:
- *(Optionally: shift right* `num_shifts` *times.)*

Immediate Issues

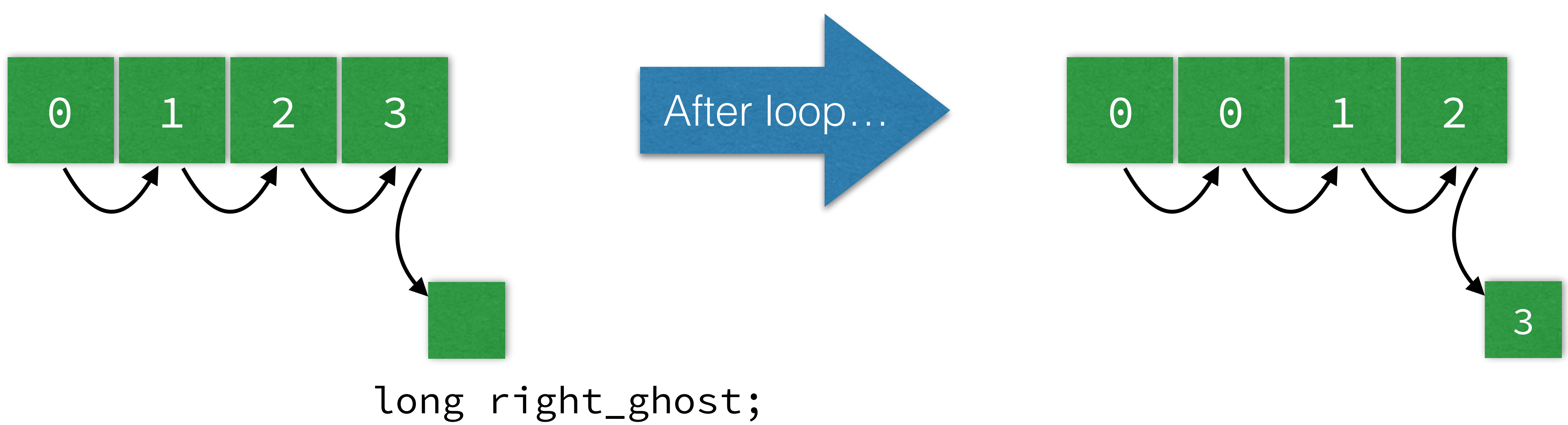
- Within each data array we risk losing information:

```
for (size_t i=N-1; i>0; --i)
  uk[i] = uk[i-1];
```



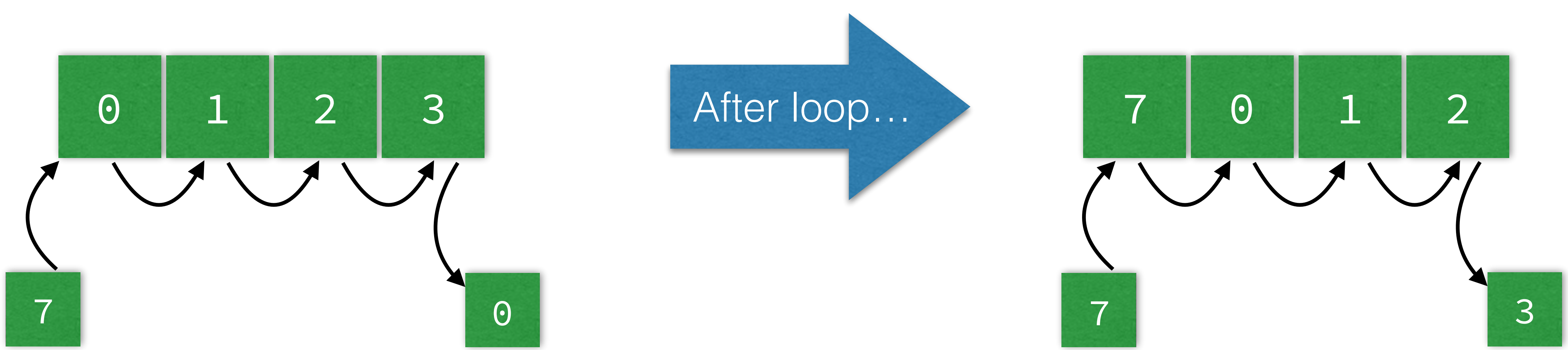
Ghost Cells

- Common domain decomposition strategy: create “ghost cells”
  - temp storage
  - data to be communicated



Ghost Cells

- Looking ahead: data incoming from "left" process
  - temp store result in `left_ghost`
  - **local "comms"**: `uk` **global comms**: ghost cells



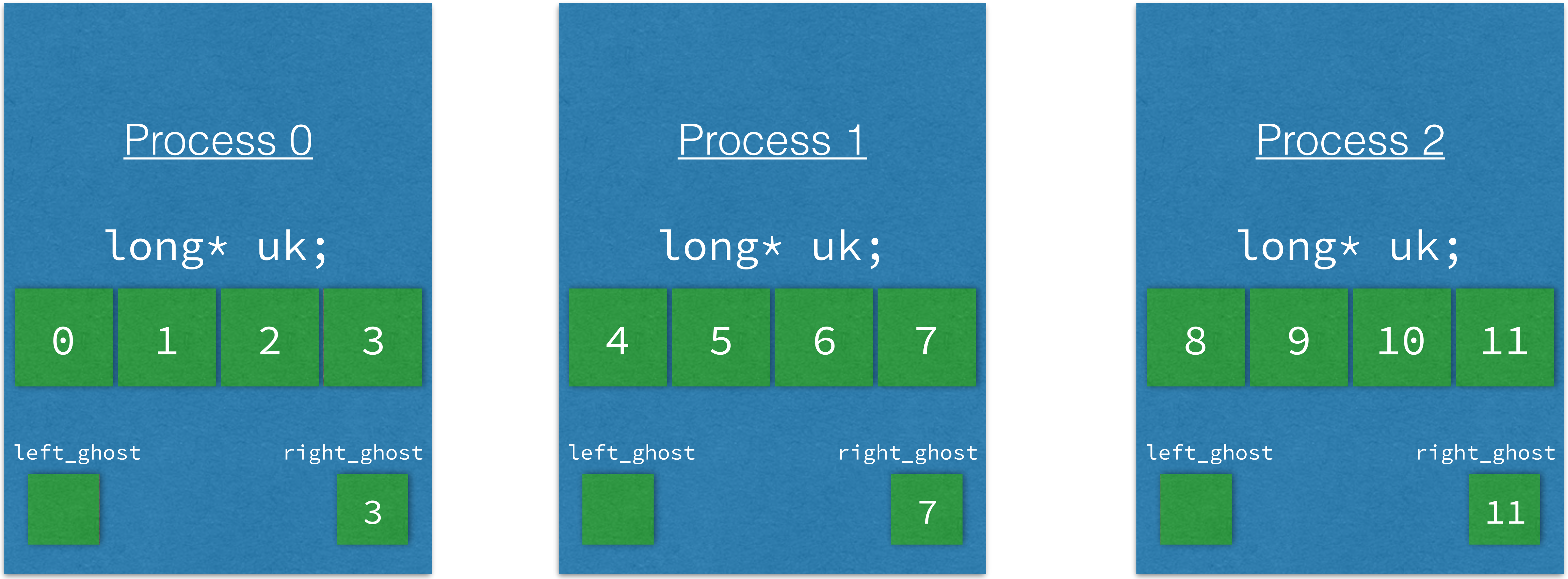
`long left_ghost;`
(received from left process)

`long right_ghost;`

(sent to "right" process)

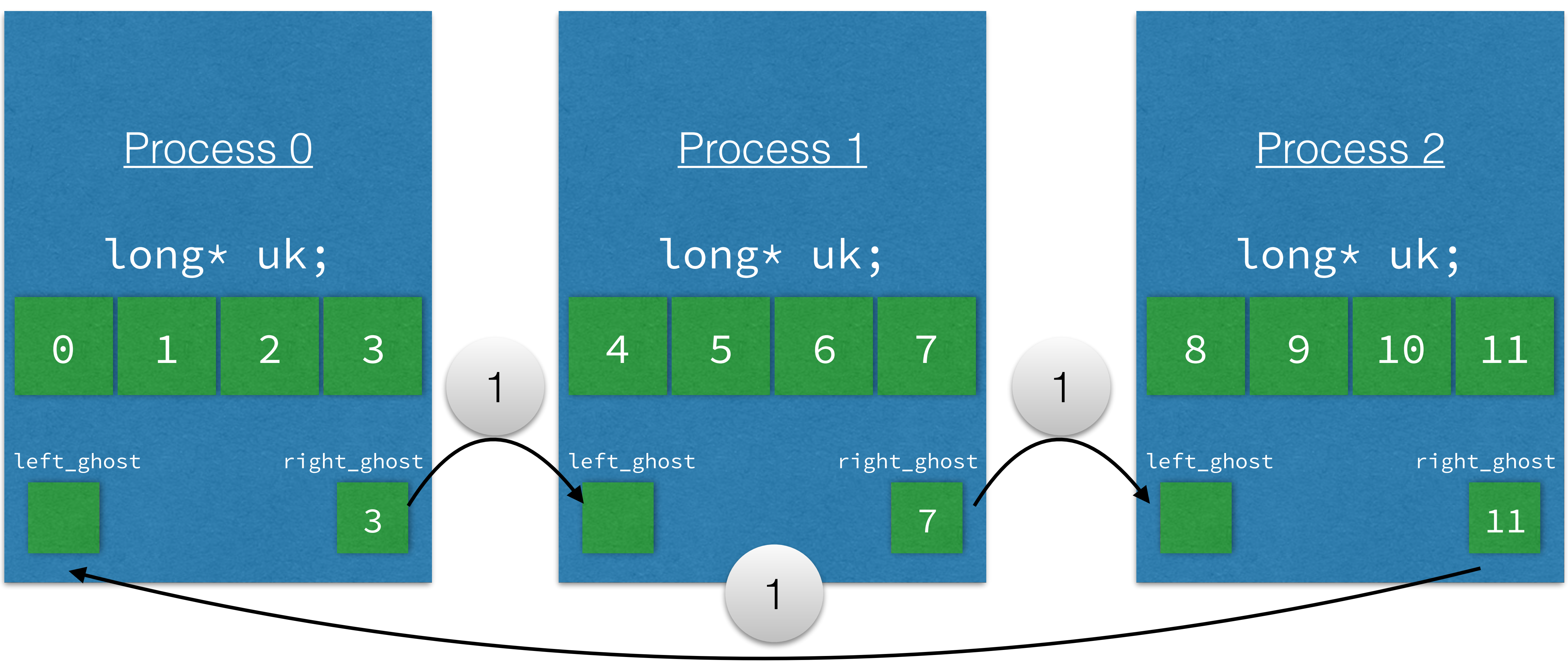
Revisiting Problem Statement

- Rewrite using ghost cells:



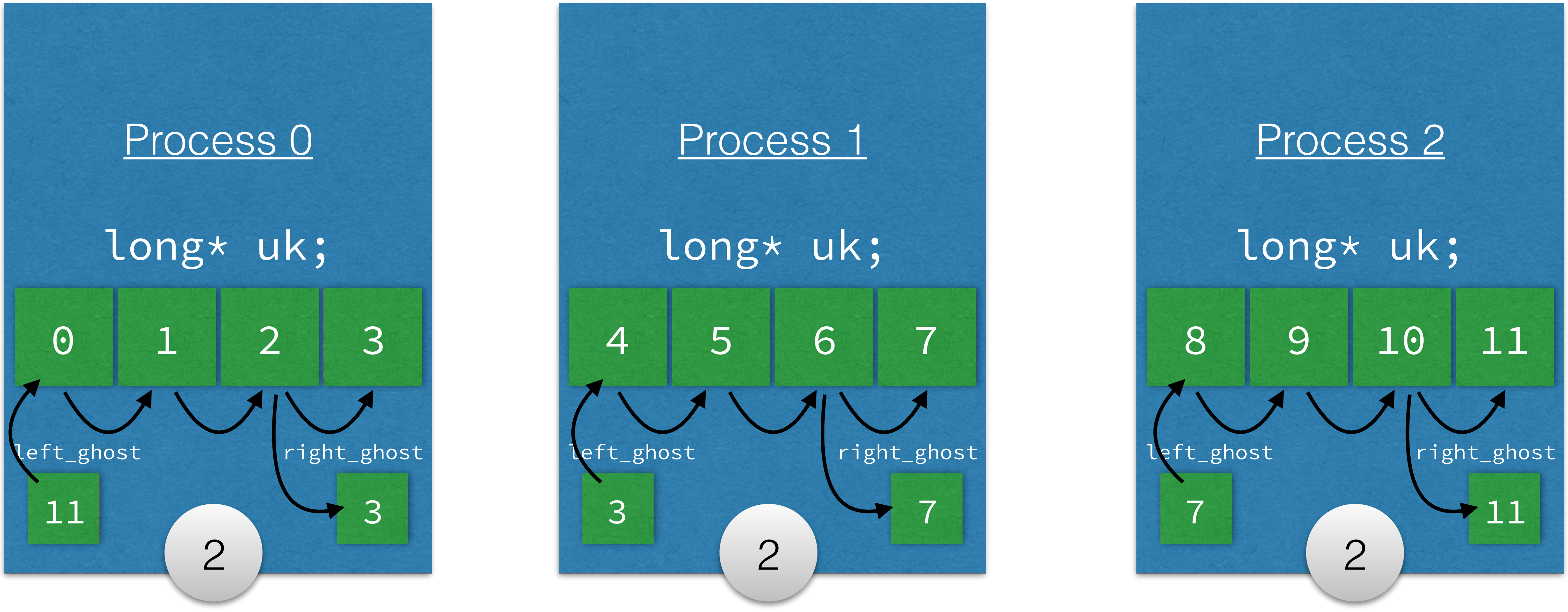
Revisiting Problem Statement

- 1) Communicate boundary data.
- 2) Perform shift within own data array.



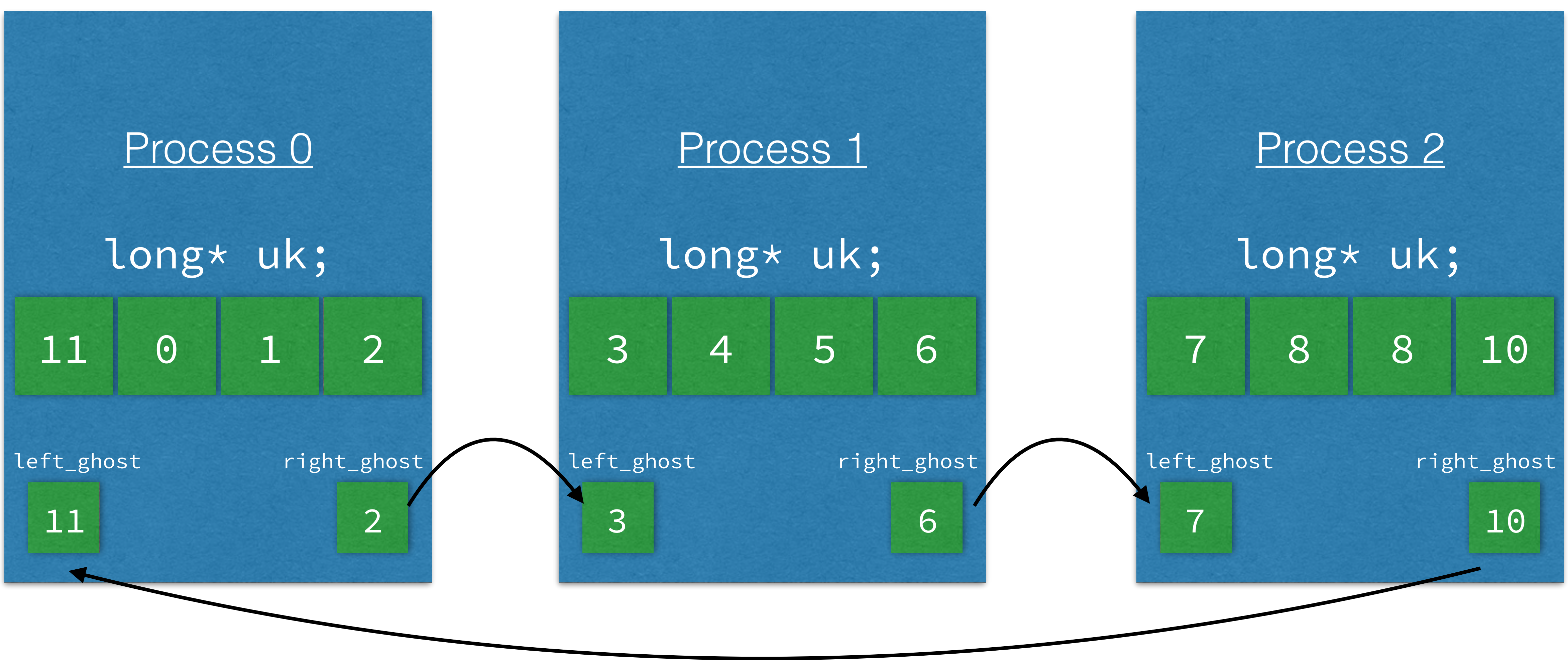
Revisiting Problem Statement

- 1) Communicate boundary data.
- 2) Perform shift within own data array.



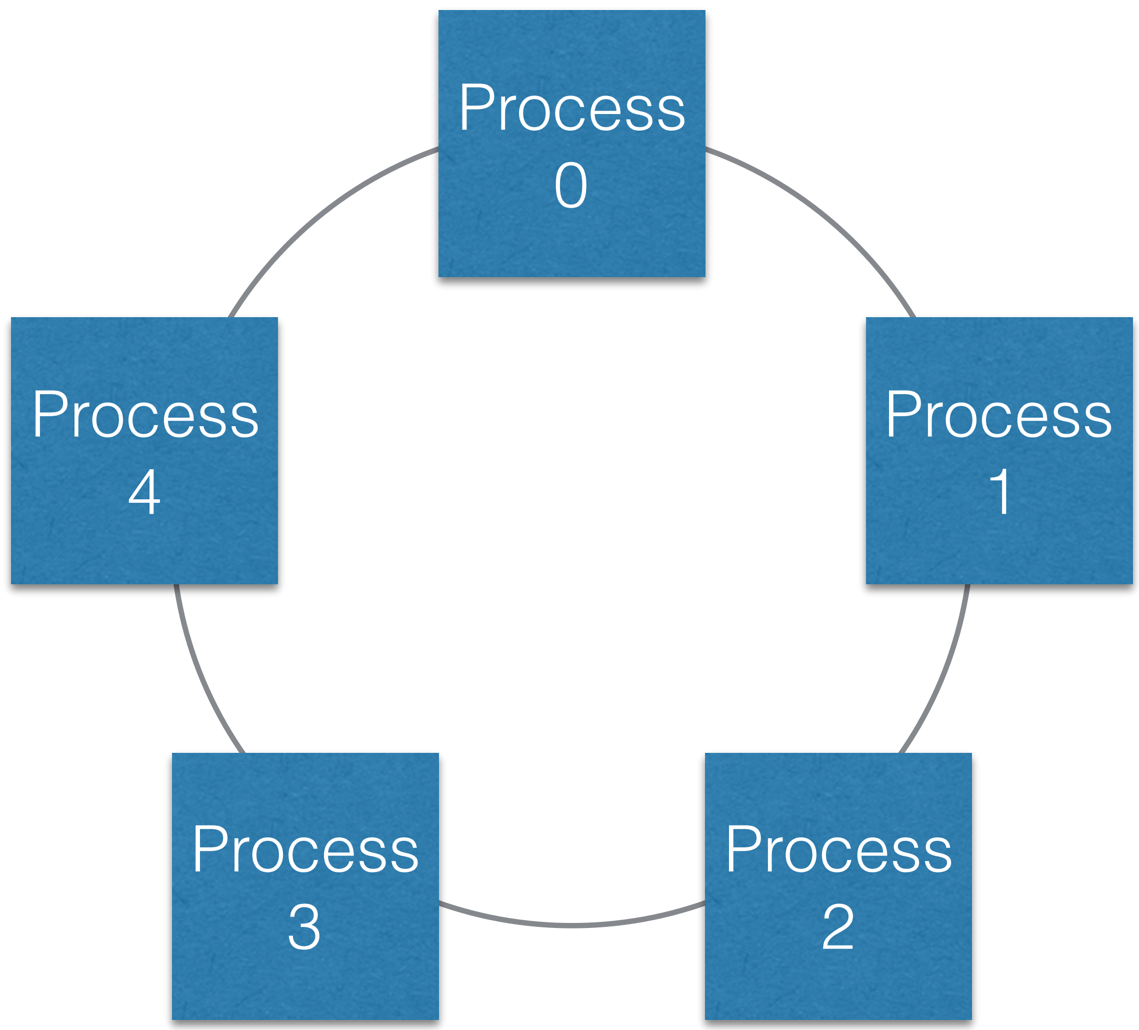
Revisiting Problem Statement

- **Key Design Idea**: separate the local computation with the necessary communication.
  - (sometimes communication latency can be hidden)



Aside: Process Topology

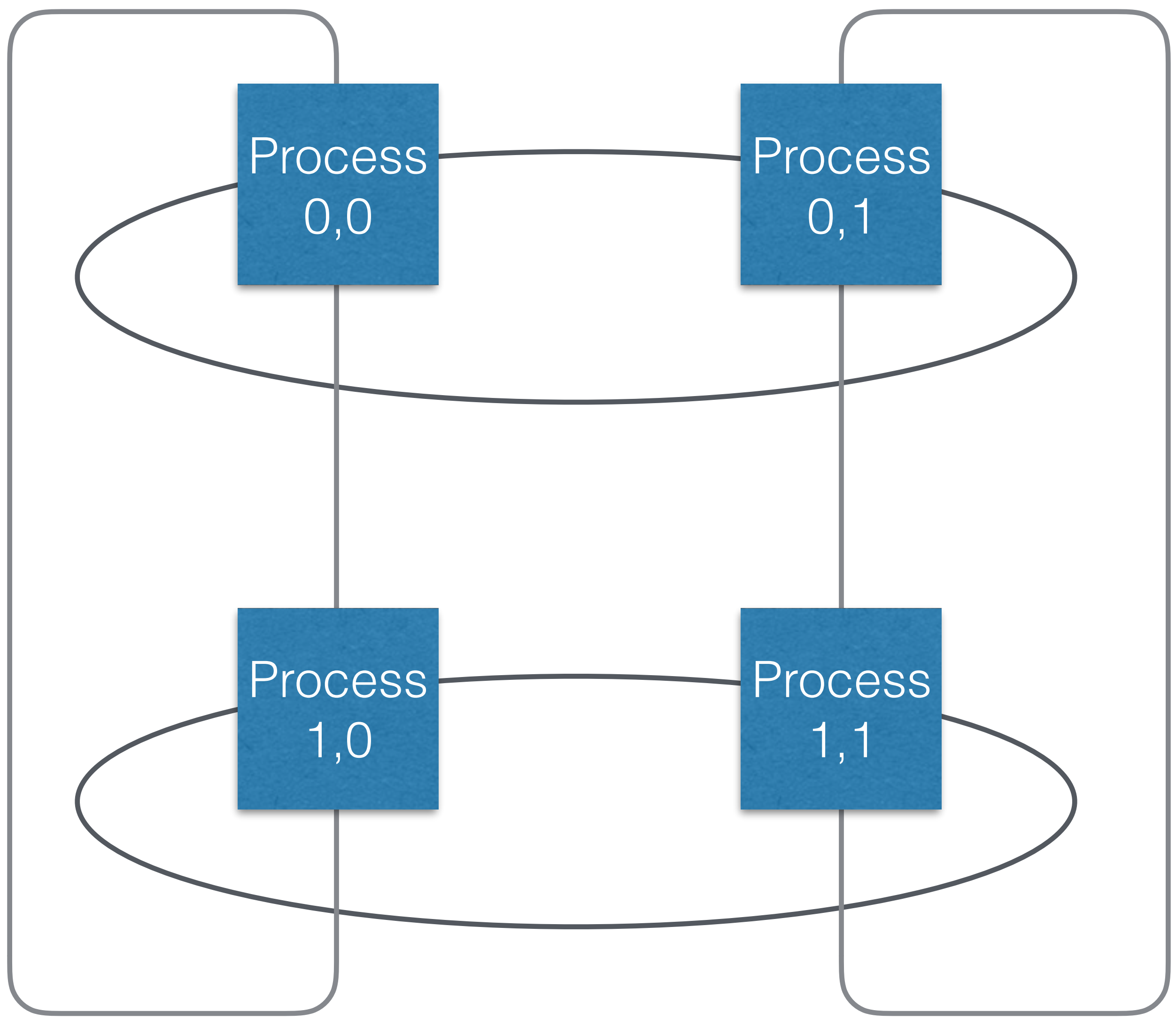
- Process communication laid out in ring:



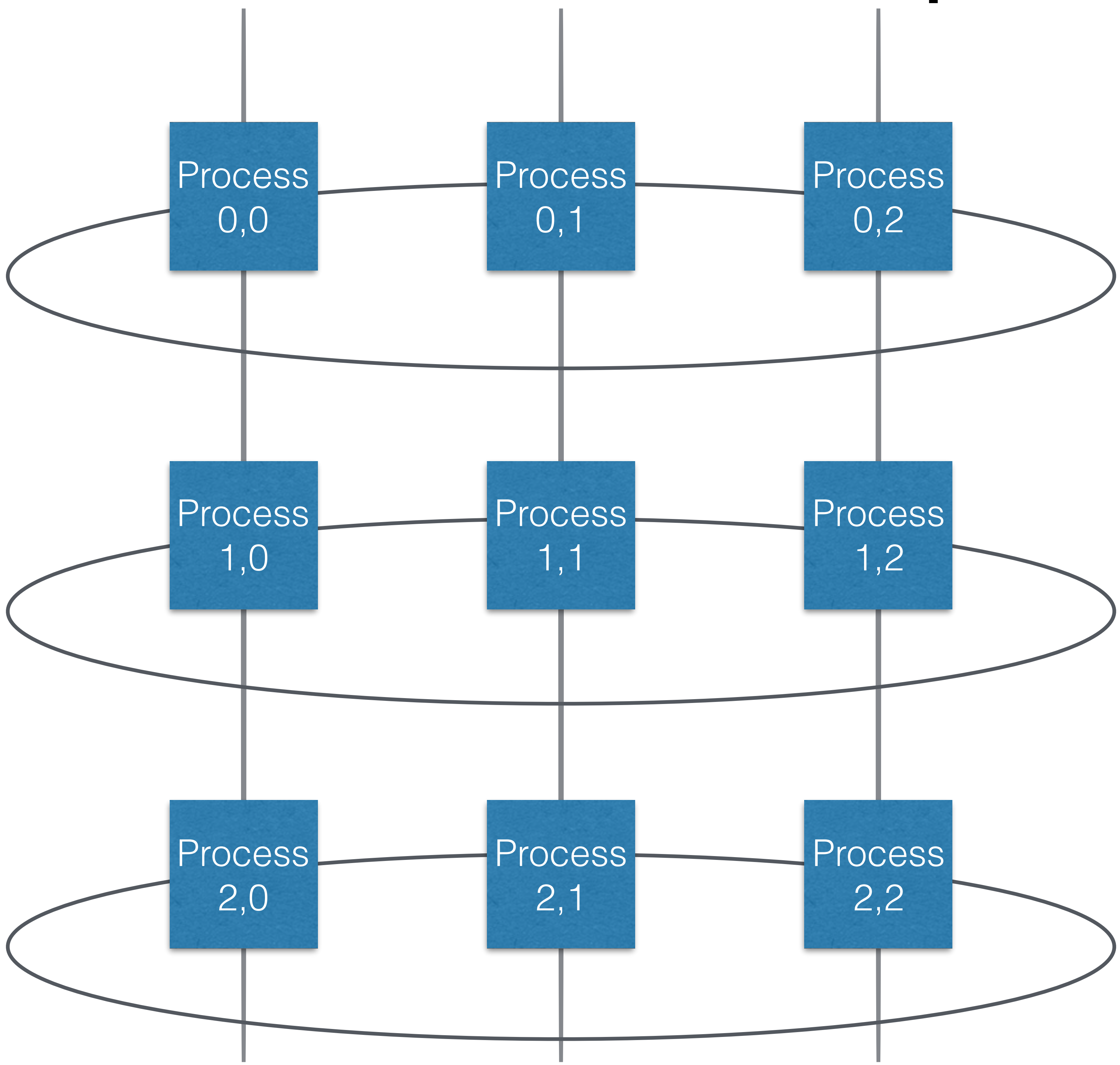
- Topology at hardware level as well. *(Certain servers connected to others in certain way.)*

Aside: Process Topology

- “Toric topology” — 2D periodic



Aside: Process Topology



Code Setup

- **Each process**:
  - gets `rank` and `size`
  - identifies "`left`" and "`right`" processes
  - allocates and initializes data, `long* uk`
  - creates ghost cell storage

Demo

Start with `shift-setup.c` and fill in “the good stuff”.
(Finished and fully documented version in `shift.c`)

Refactoring for Organization

- Suppose we want to use this perform_shift() functionality in another code.
  - Useful to create shared library.
  - Callable from C binaries / Python!
- Step 1: refactor within one file.

Quick Demo

`shift-refactored.c`

libshift.so

- Now, source files: `shift.h` / `shift.c`

```
void peform_shift(
    long* uk, size_t N,
    long num_shifts, MPI_Comm comm);
```

- Compile as usual, but with mpicc

```
$ mpicc -shared -o libshift.so shift.c
```

main.c

- Program: `main.c`

```
#include "shift.h"
int main(void) {
  // MPI setup, create data
  perform_shift(data, length, 2, MPI_COMM_WORLD)
  // MPI teardown
}
```

Number of shifts

- Compile and link:

```
$ mpicc -lshift -L. -I. main.c
$ mpiexec -n 2 ./a.out
```

Quick Demo

Contents of: `shift-library/`

mpi4py

- Miraculously, you can run MPI library code from Python using ctypes
- **Package**: mpi4py

  ```
  $ pip install mpi4py
  ```

  (Install may appear to freeze. Actually compiling in the background.)
- Homework #4. *(I will write wrappers for you.)*

mpi4py

- Workflow:

  1. Compile MPI C library code

  2. Link Python code via ctypes

  3. Create an mpi4py communicator object and pass to C library function

  4. Execute Python script using mpiexec:

```
$ mpiexec -n 4 python my_script.py
```

Demo

Code in `lecture15/mpi4py/` directory.