Worker Controller Paradigm

Warwick RSE

Overview

- The next most common approach is the worker controller approach
- Also often called Master/Slave
- You have one processor handing out work packages to all the other processors

Overview

- You can also have your controller processor doing work
  - Your MPI gets a bit more complex
  - You need to make sure that the controller serves the workers with data fast enough or you'll lose more from workers lying idle than you'll gain from the controller doing work
- We won't consider that case here

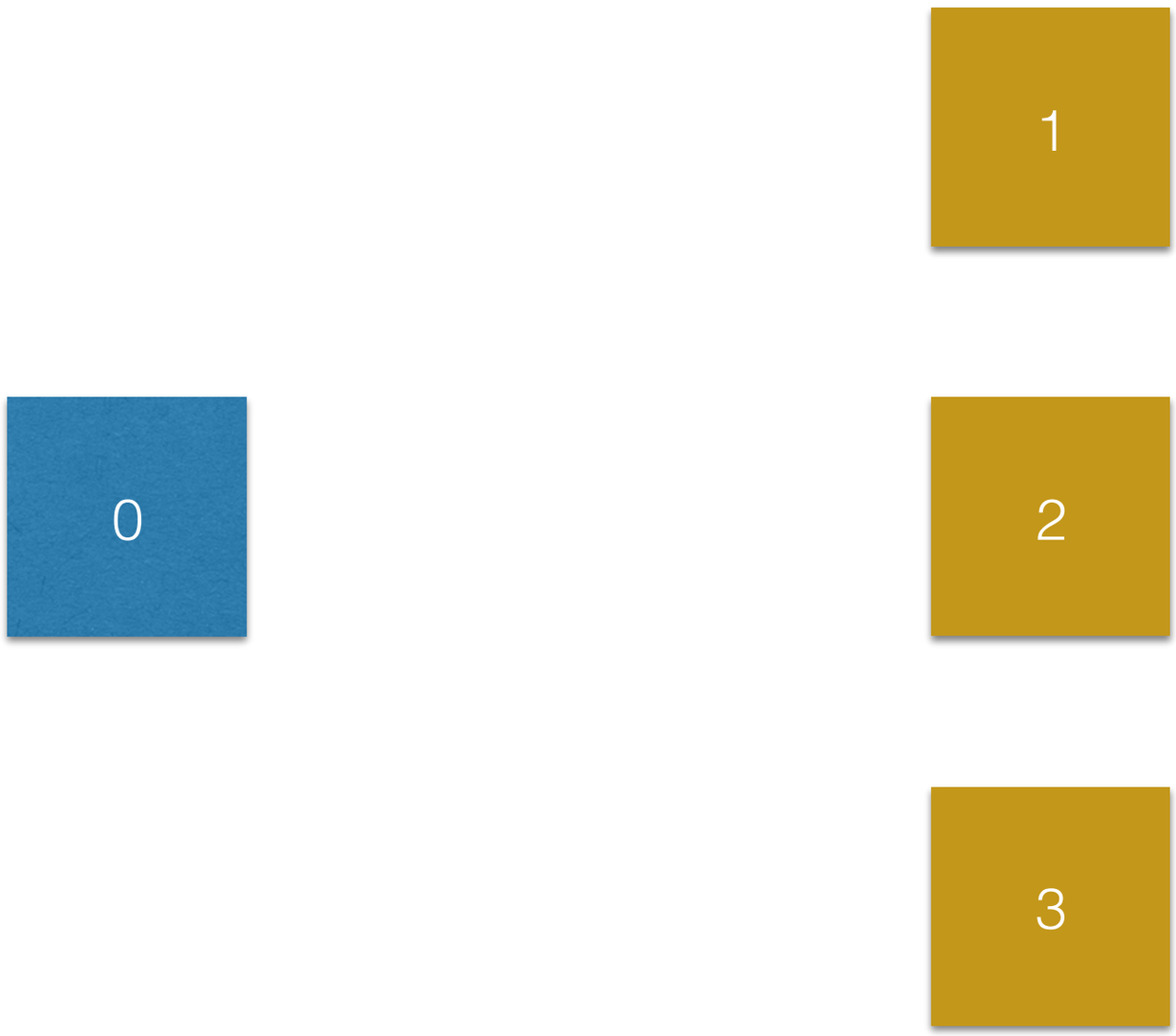
SPMD

- Important to remember MPI is (almost) a pure SPMD library
  - Single Program Multiple Data
- Have to write a single program that is both the controller and the worker
- Actually really easy, just pick a rank to be the controller and all others are workers

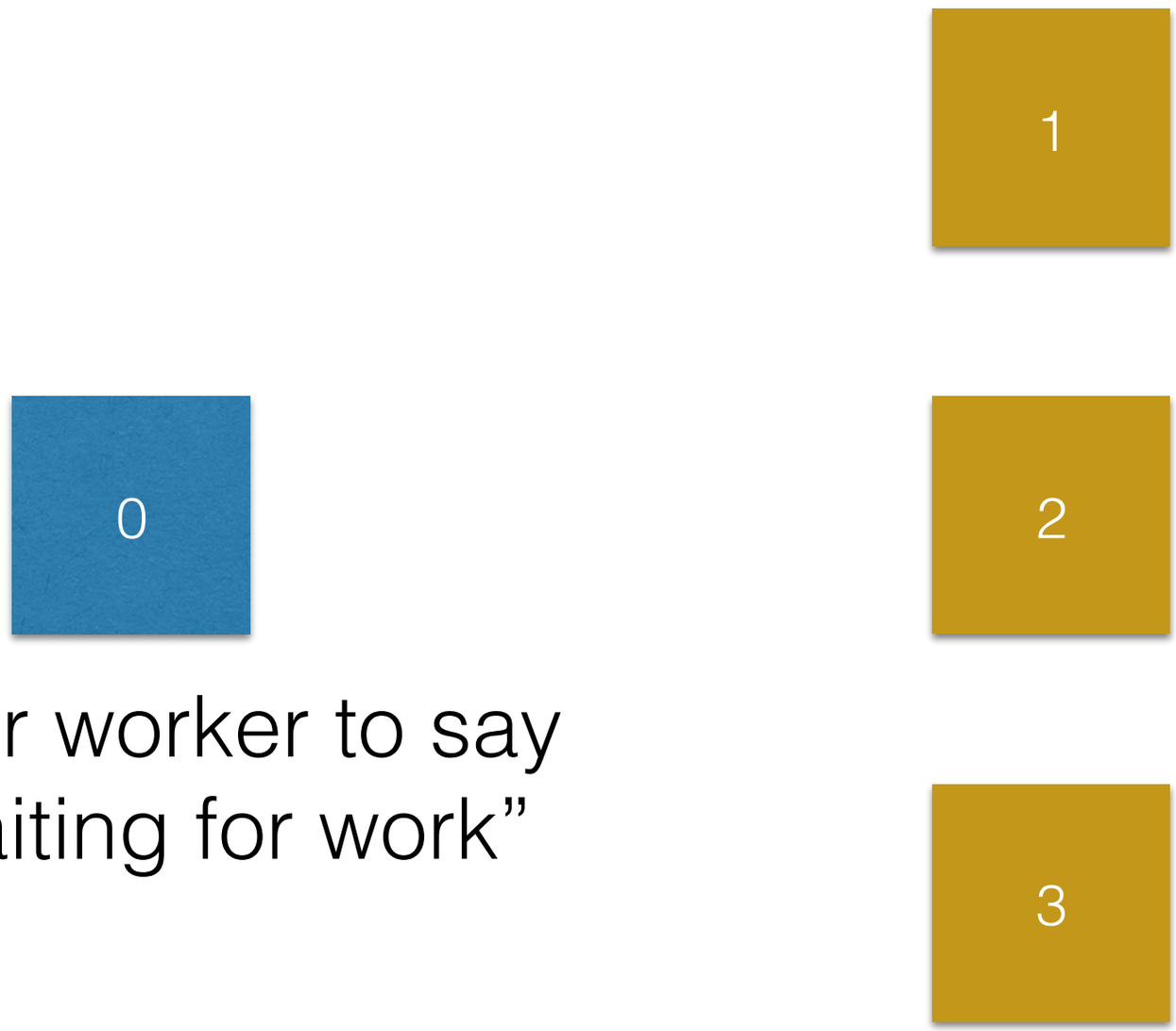
SPMD

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
  controller_fn();
} else {
  worker_fn();
}
```

Schematic

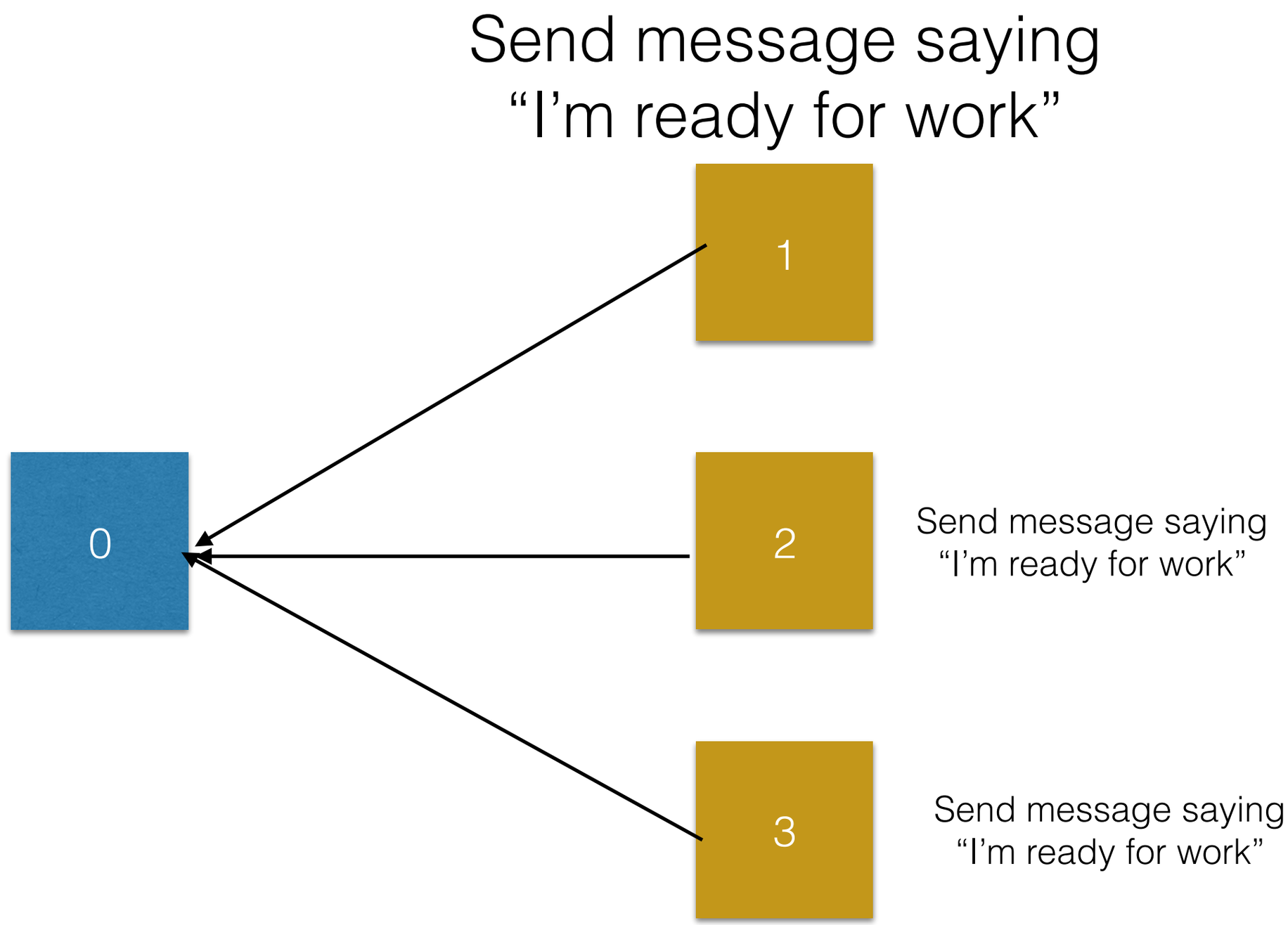


Schematic

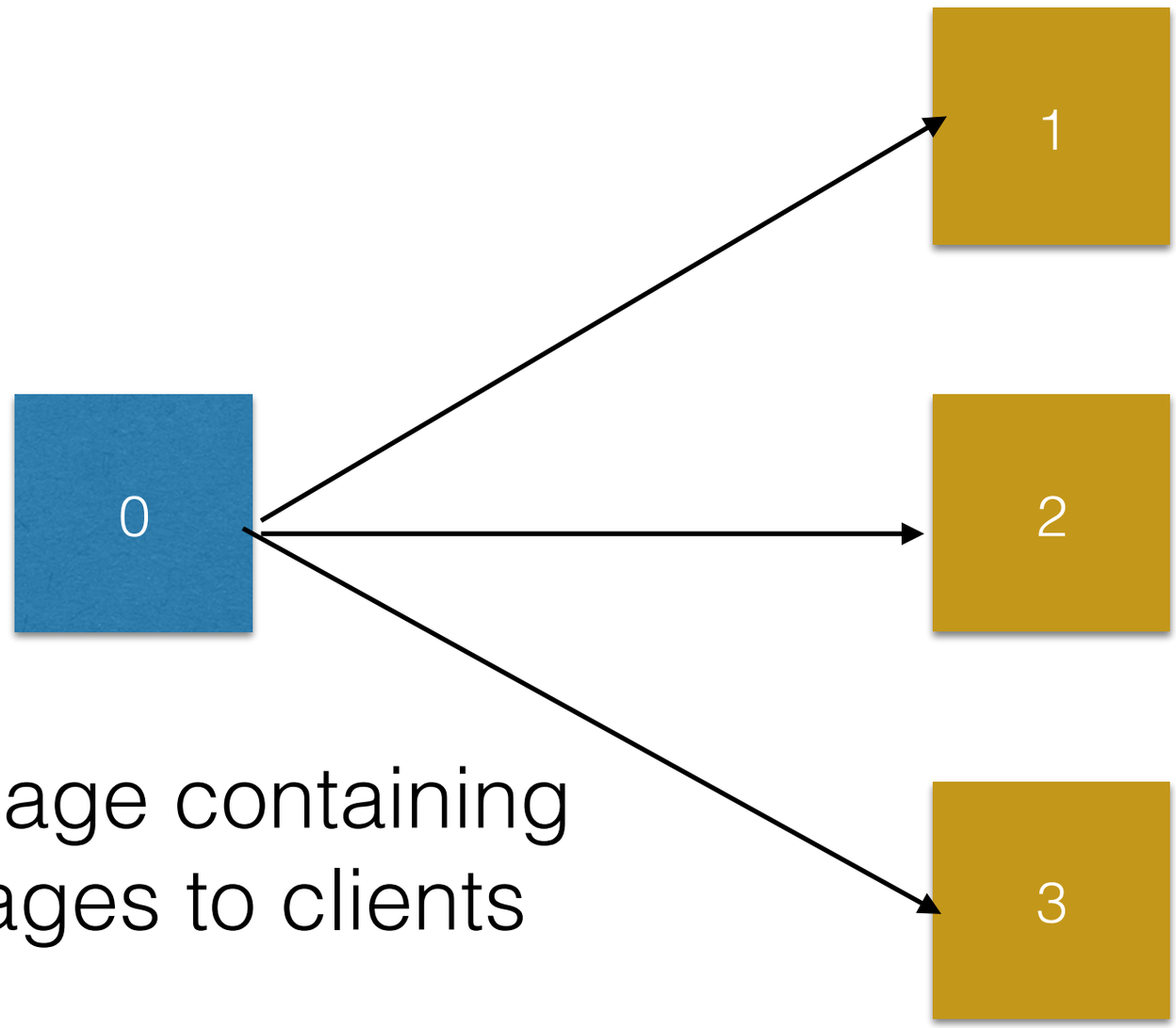


Waits for worker to say
“I’m waiting for work”

Schematic


Send message saying
“I’m ready for work”
1
0
2
Send message saying
“I’m ready for work”
3
Send message saying
“I’m ready for work”

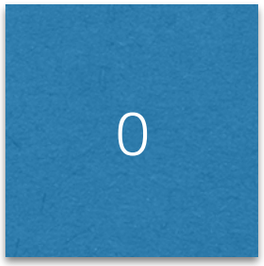
Schematic



Sends message containing work packages to clients

Schematic

Processing!

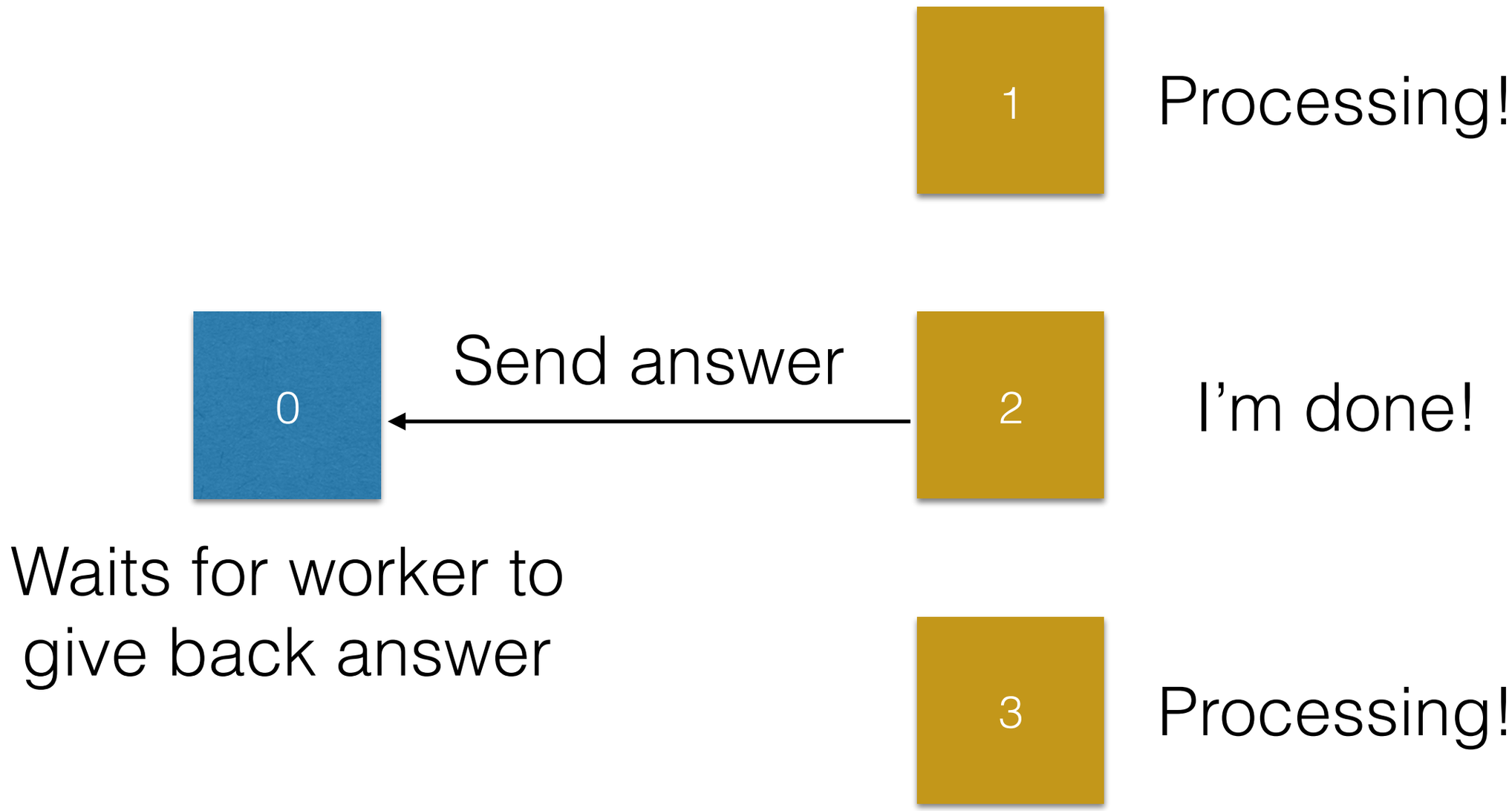


Waits for worker to give back answer

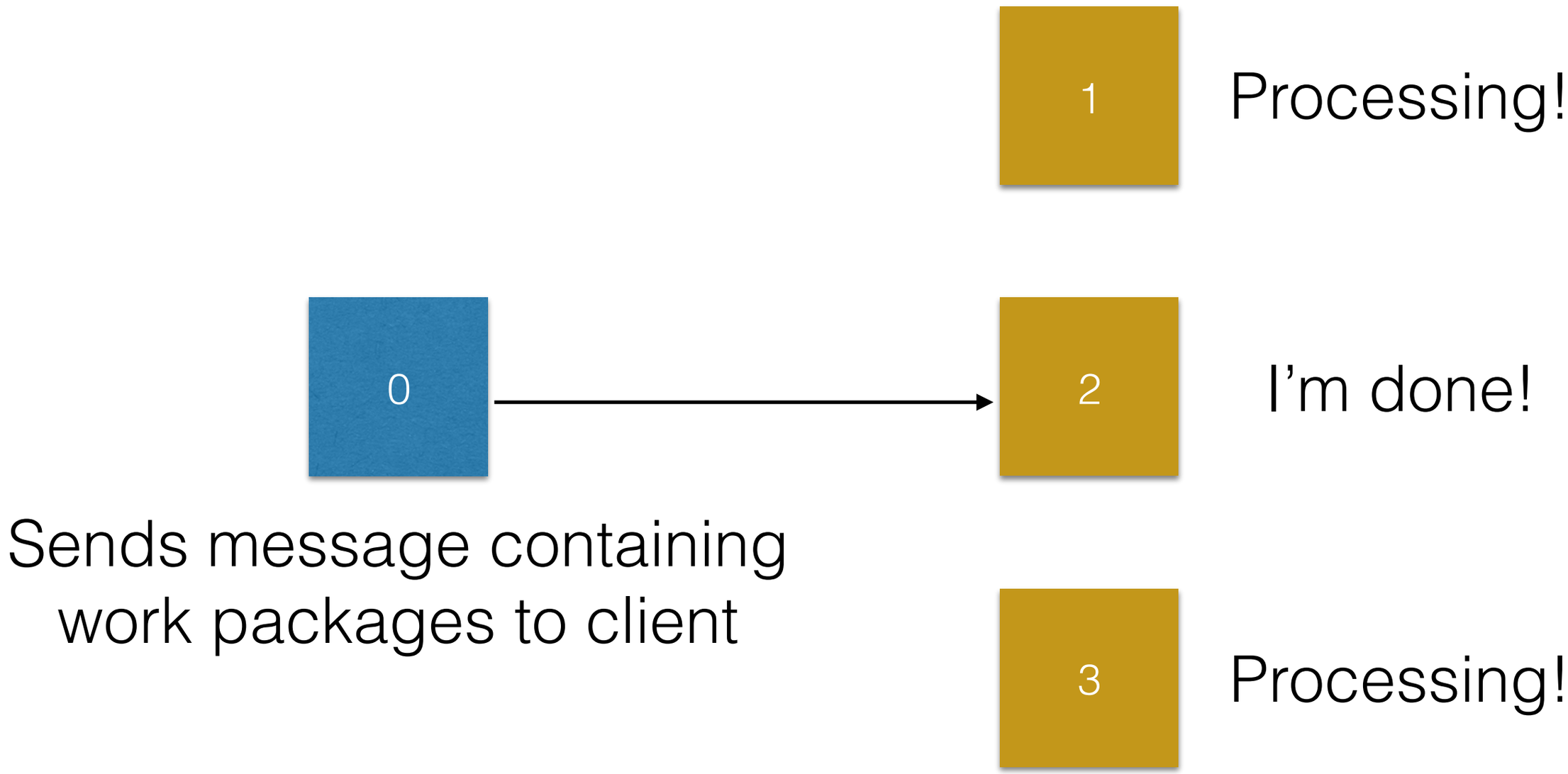
Processing!

Processing!

Schematic

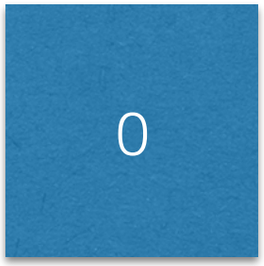

1
Processing!
0
Send answer
2
I'm done!
Waits for worker to give back answer
3
Processing!

Schematic


1
Processing!
0
2
I'm done!
Sends message containing work packages to client
3
Processing!

Schematic

Processing!

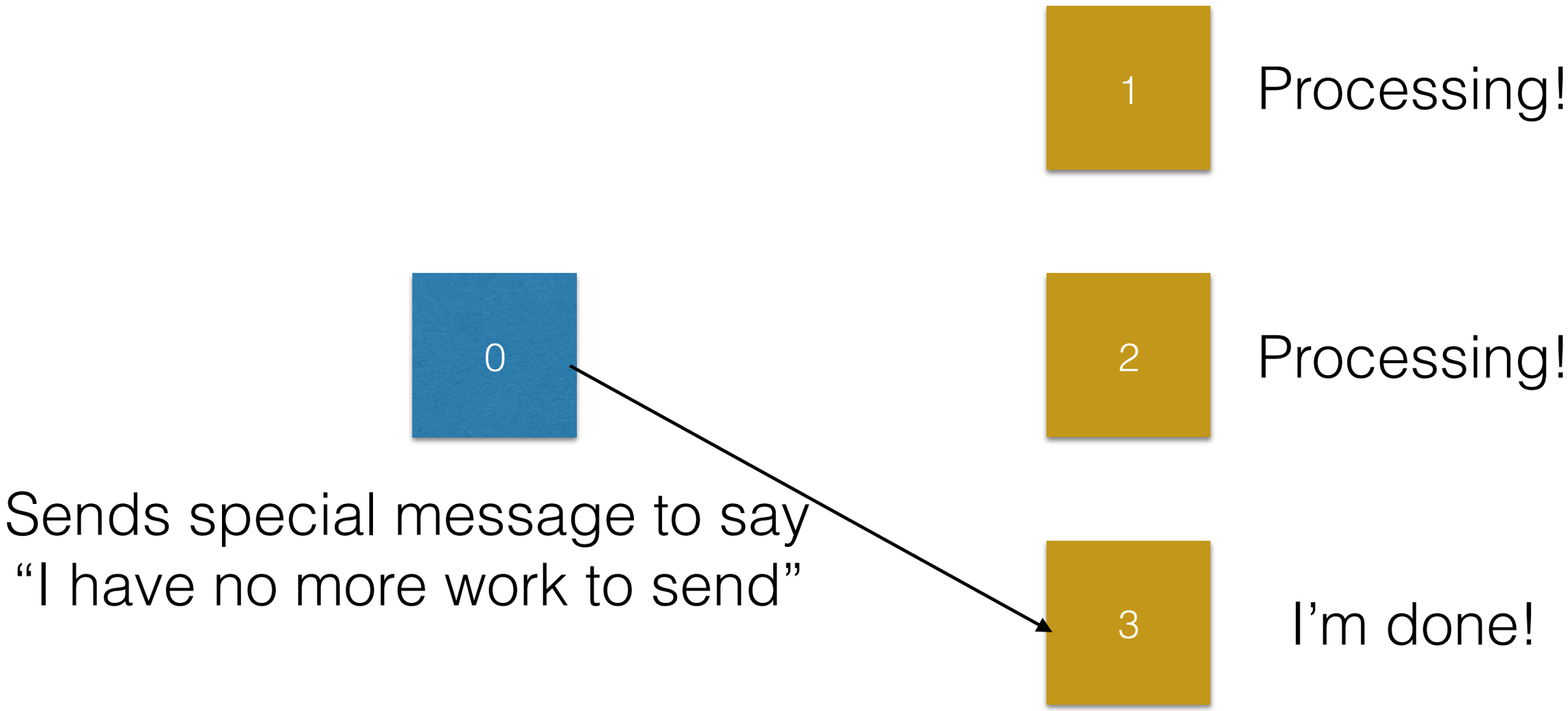


Waits for worker to say
“I’m waiting for work”

Processing!

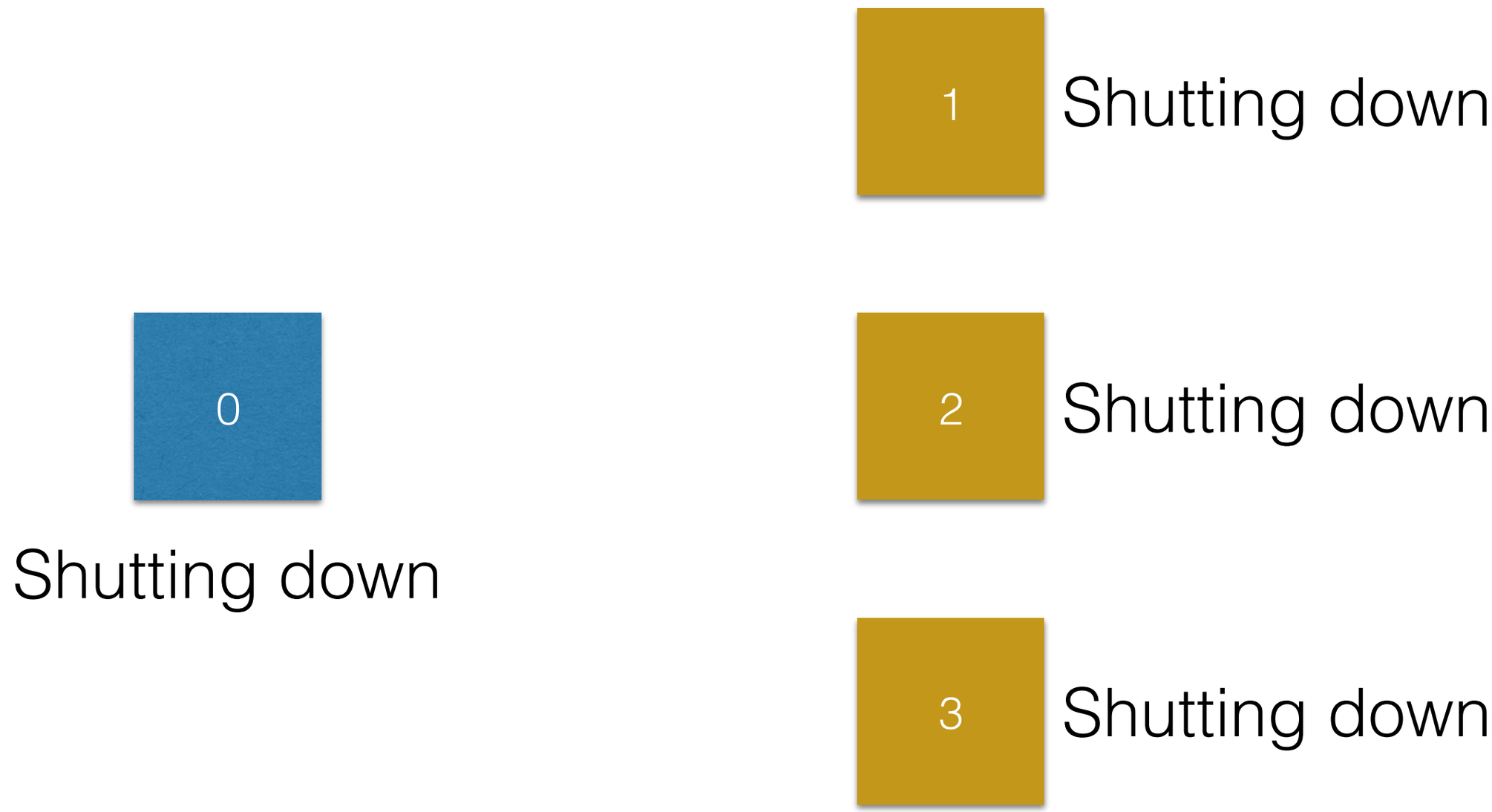
I’m done!

Schematic


1
Processing!
0
2
Processing!
Sends special message to say
“I have no more work to send”
3
I’m done!

Schematic

Schematic


1
Shutting down
0
Shutting down
2
Shutting down
3
Shutting down

Controller

- The controller basically just sits waiting
- Normal MPI_Recv
- Use MPI_ANY_SOURCE for rank to wait for a message from any worker
- Example code also uses MPI_ANY_TAG since we use the tag to tie up which work package I'm sending back the answer to
- ```
  int data;
  MPI_Recv(data, 1, MPI_INT, MPI_ANY_SOURCE,
  MPI_ANY_TAG, MPI_COMM_WORLD, stat);
  ```
  ```

Parsing the status

- In the domain decomposition code the MPI_Status objects aren't used at all

- Here we read out the source and the tag from the objects

- ```
  MPI_Status s;
  printf(“source %i\n”,s.MPI_SOURCE);
  printf(“tag %i\n”,s.MPI_TAG);
  ```

- ```
  INTEGER :: s(MPI_STATUS_SIZE)
  PRINT *, ‘Source ‘, S(MPI_SOURCE)
  PRINT *, ‘Tag ‘, S(MPI_TAG)
  ```

Parsing the status

- The code for the workers is much simpler
- Simple MPI_Send to rank 0 to send results back to the controller
- Simple MPI_Recv from rank 0 to get your next work package
  - Or the shutdown message

Final notes on code

- Have to be careful that you make sure that your code will terminate correctly under all conditions
- Important to make sure that you keep listening on the controller even after the last work package is sent out
- Only stop when the last worker **finishes**
- If the controller stops receiving before the last worker has sent back it's result then your code won't terminate properly
  - Usually killed by wall time on a cluster
- Might cost you a lot of money without need

Non blocking

- In a real working worker/controller code you will probably see the commands

  - MPI_Isend
    MPI_Irecv

- These are **non-blocking** MPI send and receive commands

- When you call them they return straight away but hand you an **MPI_Request** object so that you can check on the status of the send or receive

Non blocking

- Matched with them are the following commands
  - MPI_Wait - Wait until a given MPI_Request completes
  - MPI_Waitall - Wait until all of the MPI_Requests in an array have completed
  - MPI_Waitany - Wait until any MPI_Request in an array of MPI_Requests has completed
- There are also **Test** versions (MPI_Test, MPI_Testall, MPI_Testany) that return a logical flag about the current state of an MPI_Request (completed or not) rather than waiting for completion

Non blocking

- They also all return MPI_Status objects (just like MPI_Recv) or arrays of MPI_Status objects
  - Allows you to check the same things (tag, source etc.)
- This allows you to have rank 0 do work as well as the other ranks since it can post a receive (using MPI_Irecv), do work and then test if any of the workers has completed (using MPI_Testany normally)

Non blocking

- Not mentioned here because non blocking is quite a bit harder
  - Can’t use temporary arrays as easily (the array mustn’t be deallocated until the receive or send has completed)
  - Fortran array subsections are right out
  - Generally paired with custom MPI types
- Covered in intermediate MPI course