

CSC352

Week #7 — Spring 2017
Introduction to MPI

Dominique Thiébaud
dthiebaut@smith.edu

Introduction to MPI

D.Thiebaut

Inspiration & Reference

- MPI by Blaise Barney, Lawrence Livermore National Lab
<https://computing.llnl.gov/tutorials/mmpi>

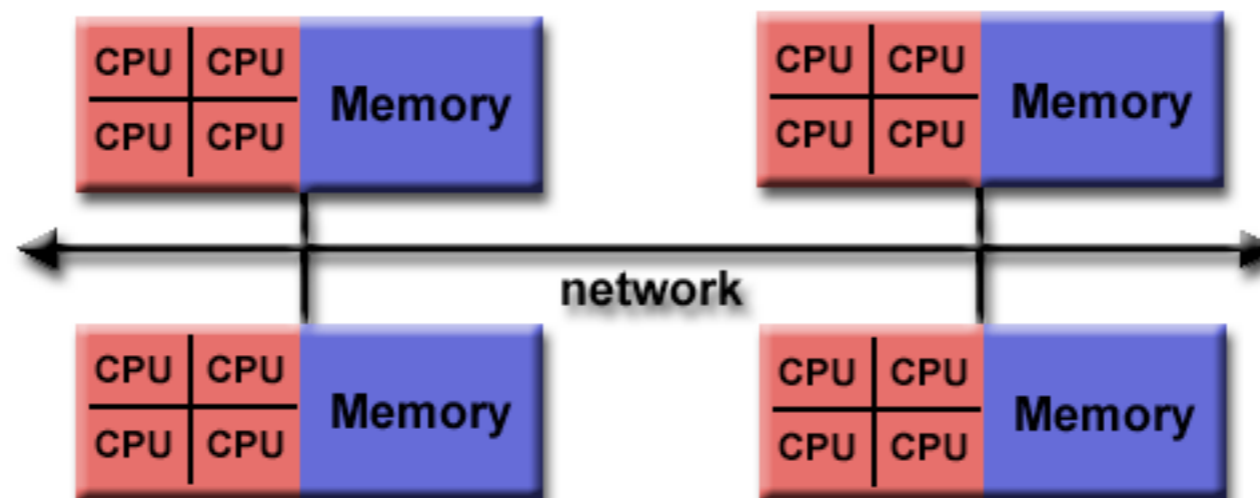


Some Background

- REVIEW: Flynn's taxonomy of computer architecture (1966): old, and faded, but everybody seems to know it!
- SISD (uniprocessor)
- SIMD (GPU)
- MISD (rare)
- MIMD (everything else!)

MIMD

- Multi-core, Many-core, Distributed systems, Clusters are all **MIMD**
- **SPMD: Single Process/Multiple Data:**
==> **MPI**



- **MPMD: Multiple Programs/Multiple Data:**

MPI: *Message Passing Interface*

- MPI is a *specification*. Not a library
- MPI libraries implement the specification
- *Processes* communicate with each other:
 - Synchronization (barriers)
 - Data exchange
- MPI is **large** (430 functions in MPI-3)
- MPI is **small** (~6 functions)

Old but Vibrant!



- **Top500: Great majority uses MPI**
- **From the Top500 Q&A:**

Q: Where can I get the software to generate performance results for the Top500?

A: There is software available that has been optimized and many people use to generate the Top500 performance results. This benchmark attempts to measure the best performance of a machine in solving a system of equations. The problem size and software can be chosen to produce the best performance. A copy of that software can be downloaded from:

<http://www.netlib.org/benchmark/hpl/>

In order to run this you will need MPI and an optimized version of the BLAS. For MPI you can see: <http://www-unix.mcs.anl.gov/mpi/mpich/download.html> and for the BLAS see: <http://www.netlib.org/atlas/> .

Advantages of MPI

- Supported in many languages (Mostly C, but not just C)
- Supports *heterogeneous* computer systems
 - Provides access to advanced parallel systems
 - Portable (install it on your Mac or Windows PC!)



THE
C
PROGRAMMING
LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

PRENTICE HALL SOFTWARE SERIES

C Tutorial

(See separate set of slides)

Hello world!

(version 1)

```
// minimalist hello
// world program
// D. Thiebaut
#include <mpi.h>
#include <stdio.h>

int main( int argc, char *argv[] ) {
    MPI_Init( &argc, &argv );
    printf( "Hello world!\n" );
    MPI_Finalize();
    return 0;
}
```

Compile & Run

```
[15:33:05] ~/mpi/352$: mpicc -o hello1 hello1.c
[15:33:41] ~/mpi/352$: mpirun -np 1 ./hello1
Hello world!
[15:33:48] ~/mpi/352$: mpirun -np 2 ./hello1
Hello world!
Hello world!
[15:33:53] ~/mpi/352$: mpirun -np 4 ./hello1
Hello world!
Hello world!
Hello world!
Hello world!
[15:33:57] ~/mpi/352$:
```

Different Syntaxes

- `mpirun -np 2 ./hello`
- `mpirun -n 2 ./hello`
- `mpiexec -n 2 ./hello`
- `mpiexec -np 2 ./hello`

Hello World!

(Version 2: more interesting)

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    printf("Process %d on %s out of %d\n", rank,
           processor_name, numprocs);

    MPI_Finalize();
}
```

Compile & Run

```
[15:41:51] ~/mpi/352$: mpicc -o hello2 hello2.c  
[15:42:00] ~/mpi/352$: mpirun -np 2 ./hello2  
Process 0 on MacDom2.local out of 2  
Process 1 on MacDom2.local out of 2
```



Exercise

Hello World on Aurora

- Create your first MPI *Hello World!* Program on Aurora (or your own laptop if you have installed MPI on it) compile it, and run it.

We Stopped Here Last Time



Next...

- Hello World on Cluster of 4 Servers
- MPI_Send/MPI_Recv/MPI_COMM_WORLD
- Potato (Exercise)
- Pi (Example)
- One-To-Many, Many-To-One Communication
- Load-Balancing and Scheduling
- MPI on **AWS**

Important Remarks

- MPI functions return values, either an error code or `MPI_SUCCESS`
- **An error causes all processes to stop**
- Two important MPI functions:
 - **`MPI_Comm_size`**: *# of processes enrolled*
 - **`MPI_Comm_rank`**: *rank of current process*



Tutorial

Hello World on Hadoop01

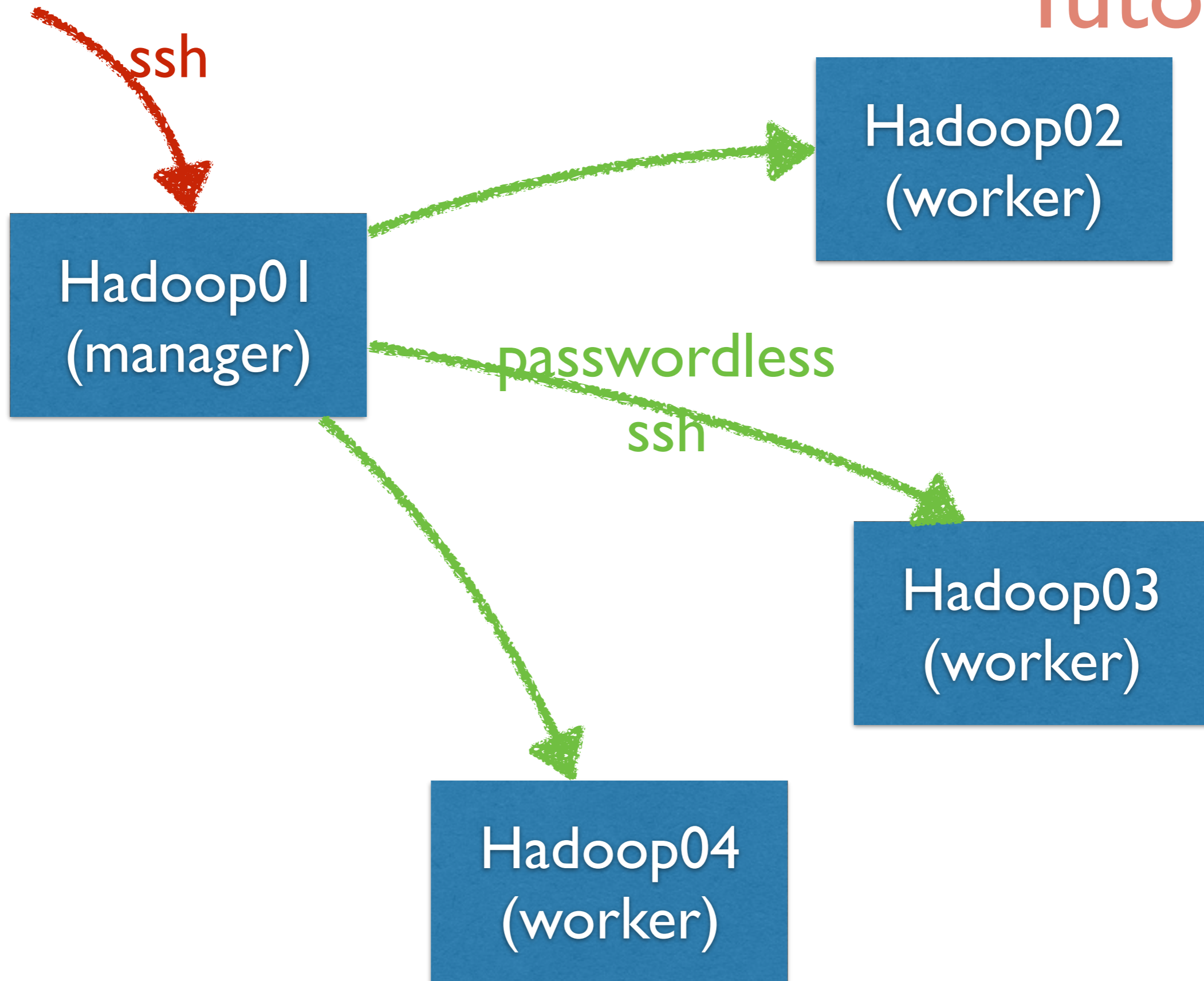
- Follow the tutorial at this URL

http://www.science.smith.edu/dftwiki/index.php/Setup_MPI_on_Hadoop_Cluster
to run MPI on a cluster of 4 servers

- See next slides for process overview

User

Tutorial



User



Hadoop01
(manager)

emacs
helloWorld.c

Tutorial

Hadoop02
(worker)

Hadoop03
(worker)

Hadoop04
(worker)

User



Hadoop01
(manager)

```
mpicc  
helloWorld.c  
-o hello
```

Tutorial

Hadoop02
(worker)

Hadoop03
(worker)

Hadoop04
(worker)

User

Tutorial

ssh

rsync

Hadoop02
(worker)

Hadoop01
(manager)

rsync

Hadoop03
(worker)

rsync

Hadoop04
(worker)

```
rsync -azv  
hello hadoop02  
rsync -azv  
hello hadoop03  
rsync -azv  
hello hadoop04
```

User

Tutorial

ssh

Hadoop01
(manager)

mpirun

Hadoop02
(worker)

mpirun

Hadoop03
(worker)

mpirun

Hadoop04
(worker)

```
mpirun -np 4 --hostfile  
hosts ./hello
```


GO!

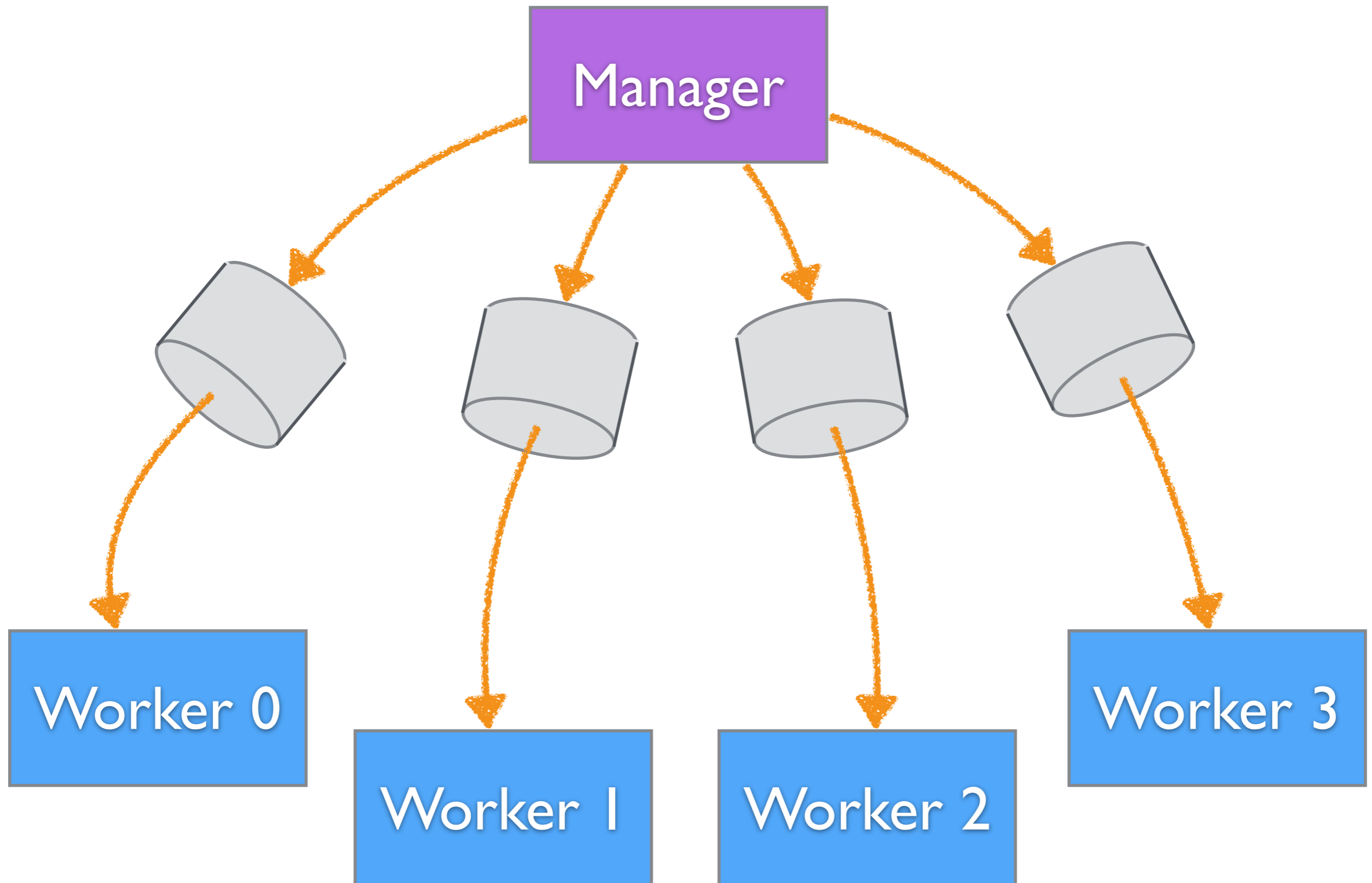
Deploy HelloWorld on
the Hadoop Cluster

We Stopped Here Last Time

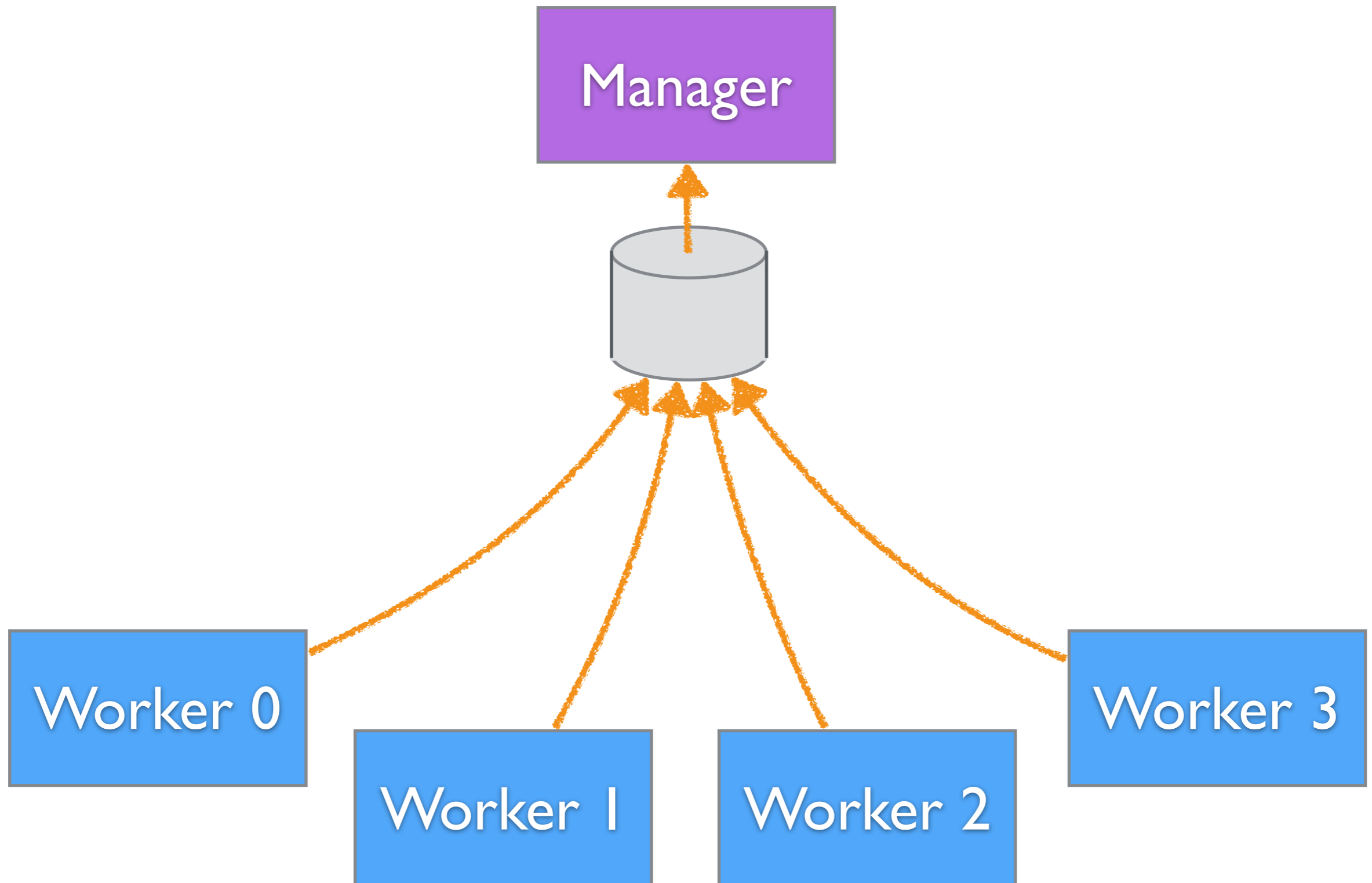


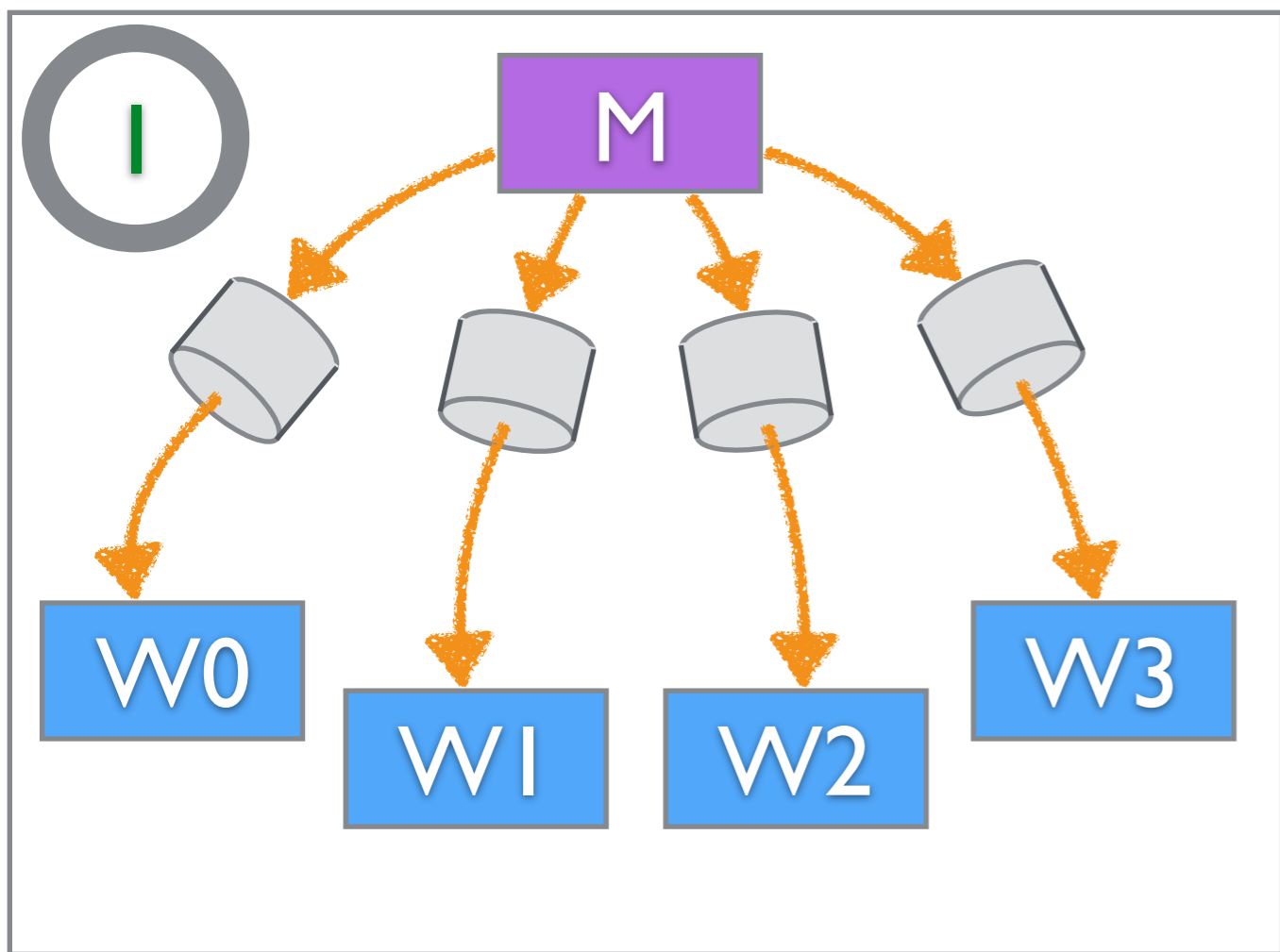
**Go Over
Java Multithreaded
Game of Life
Homework**

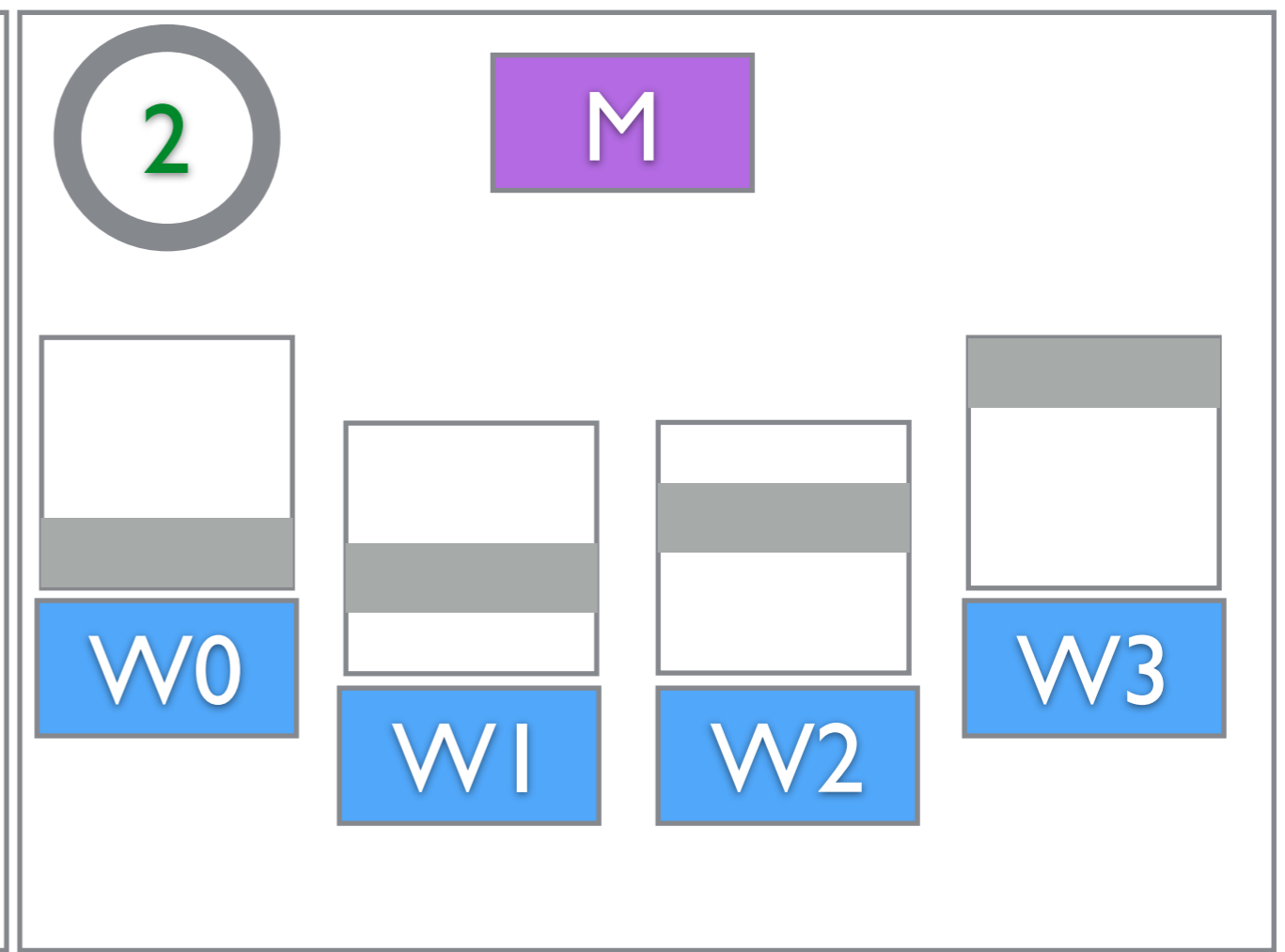
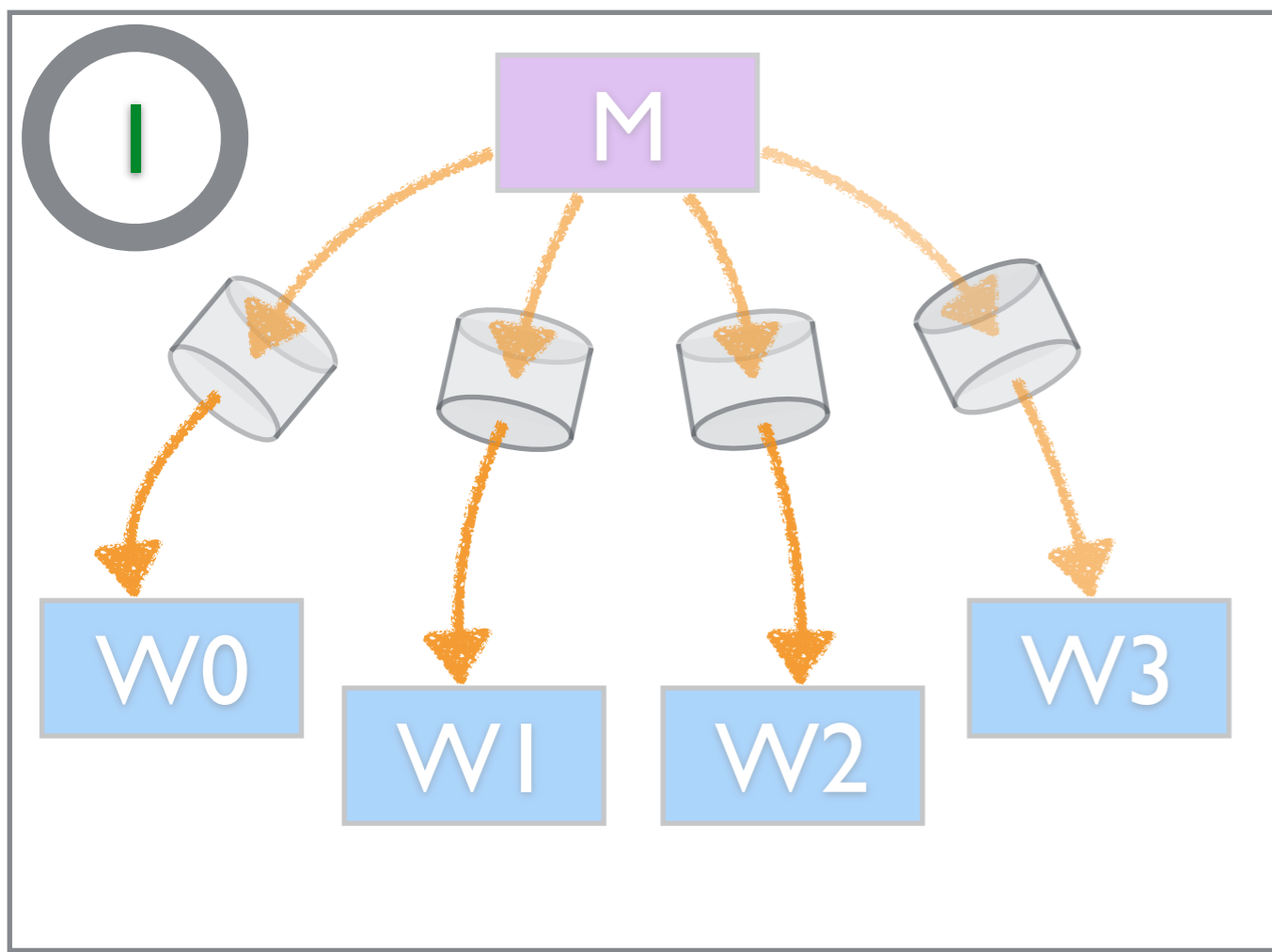
Communication Patterns

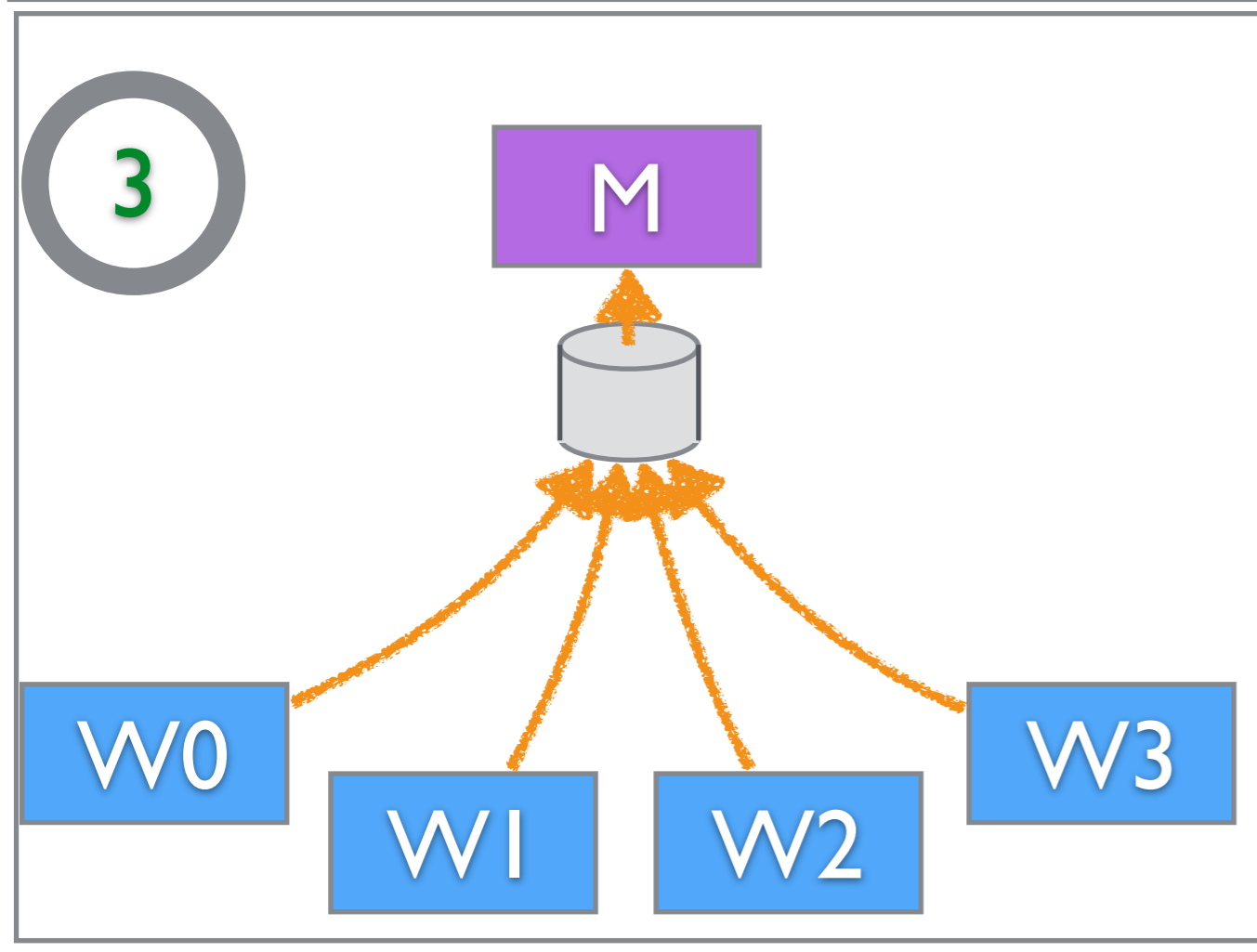
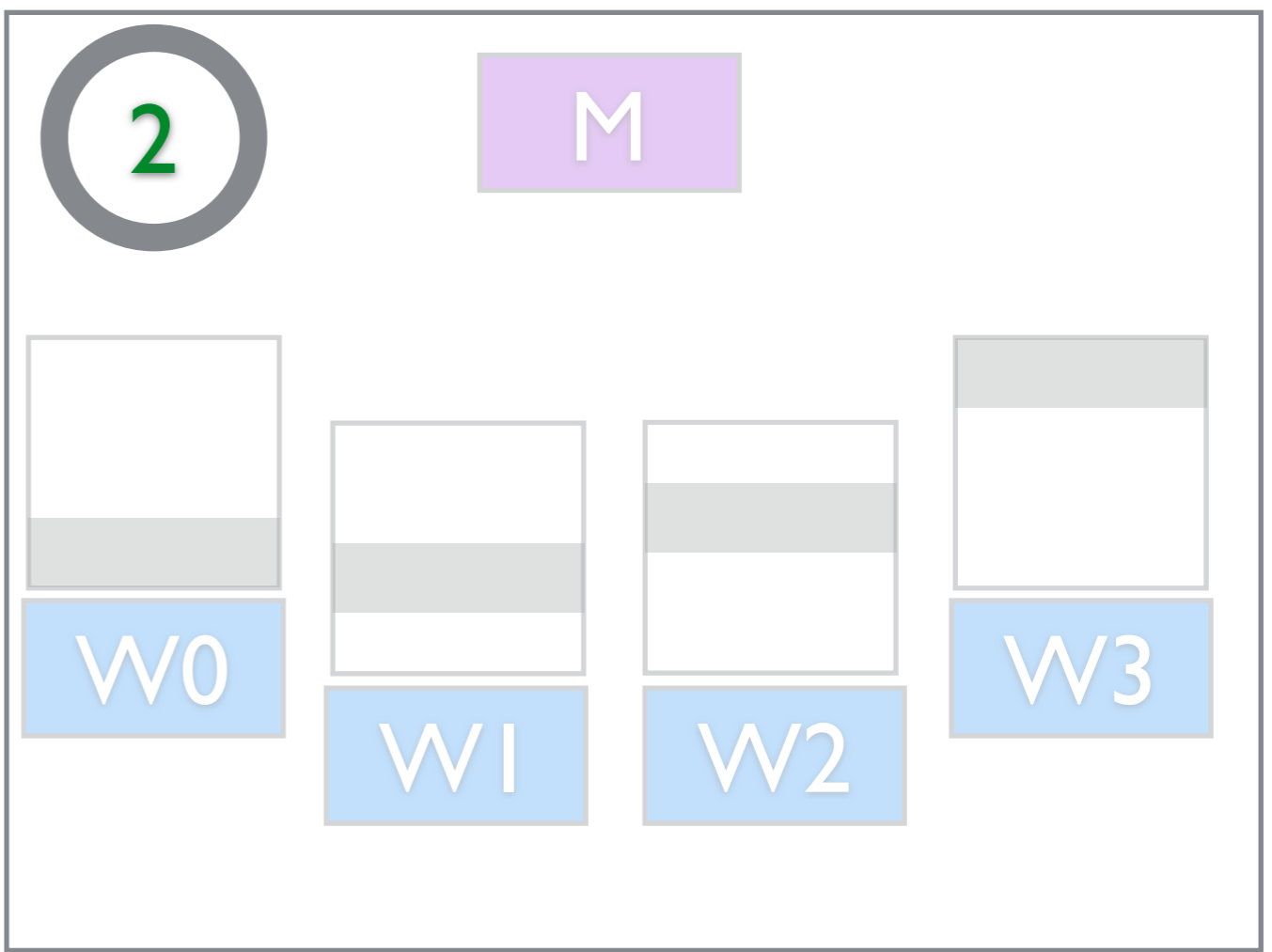
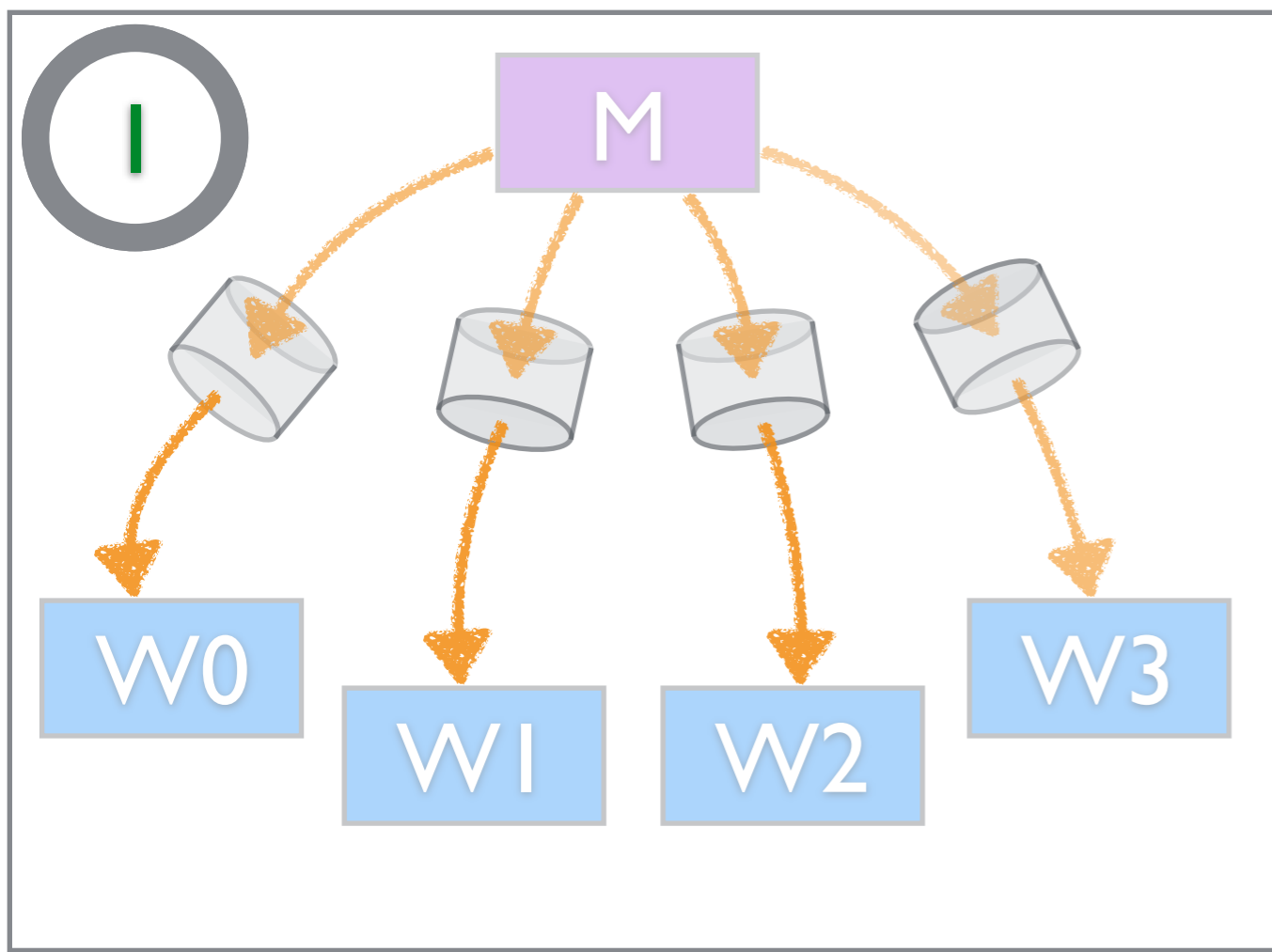


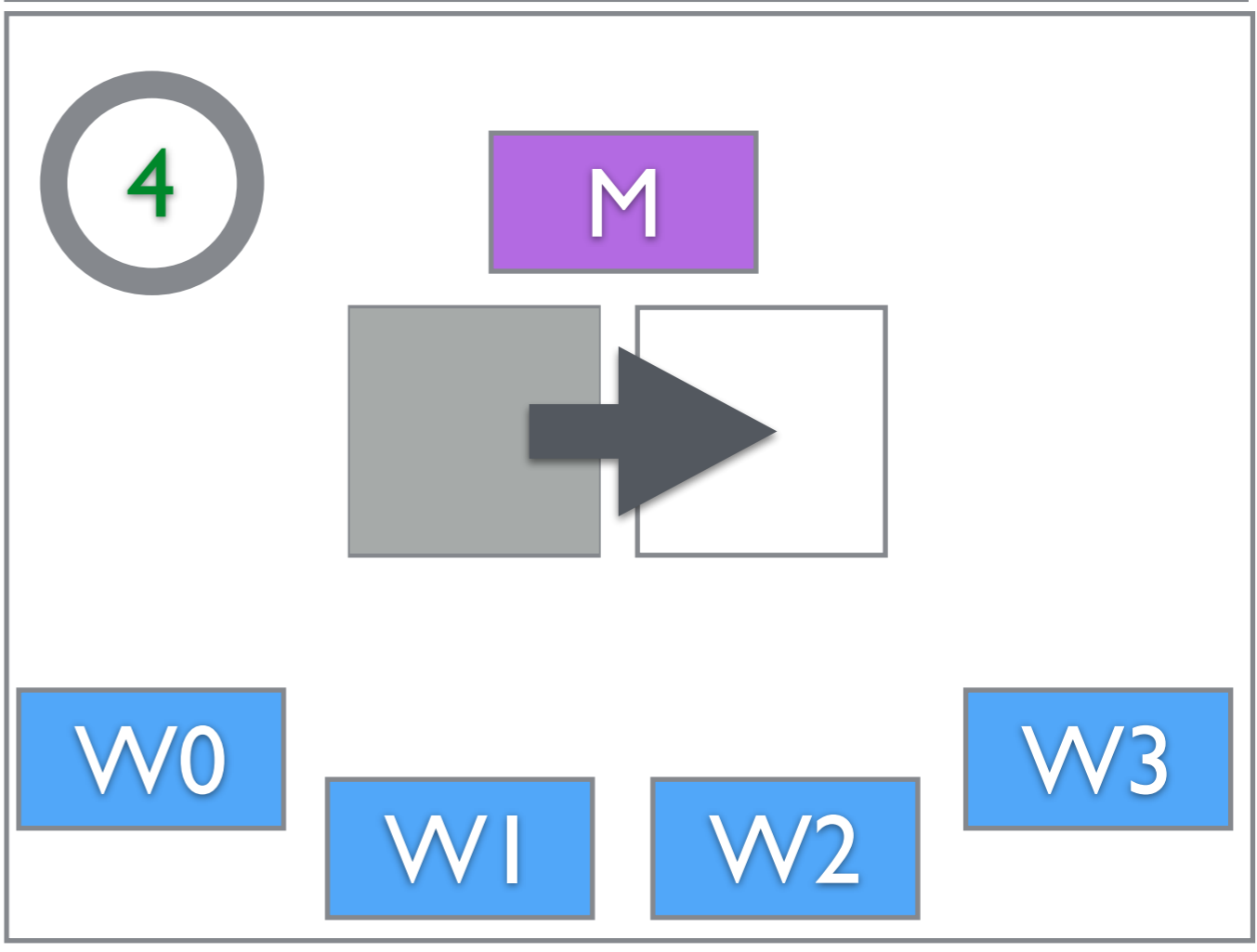
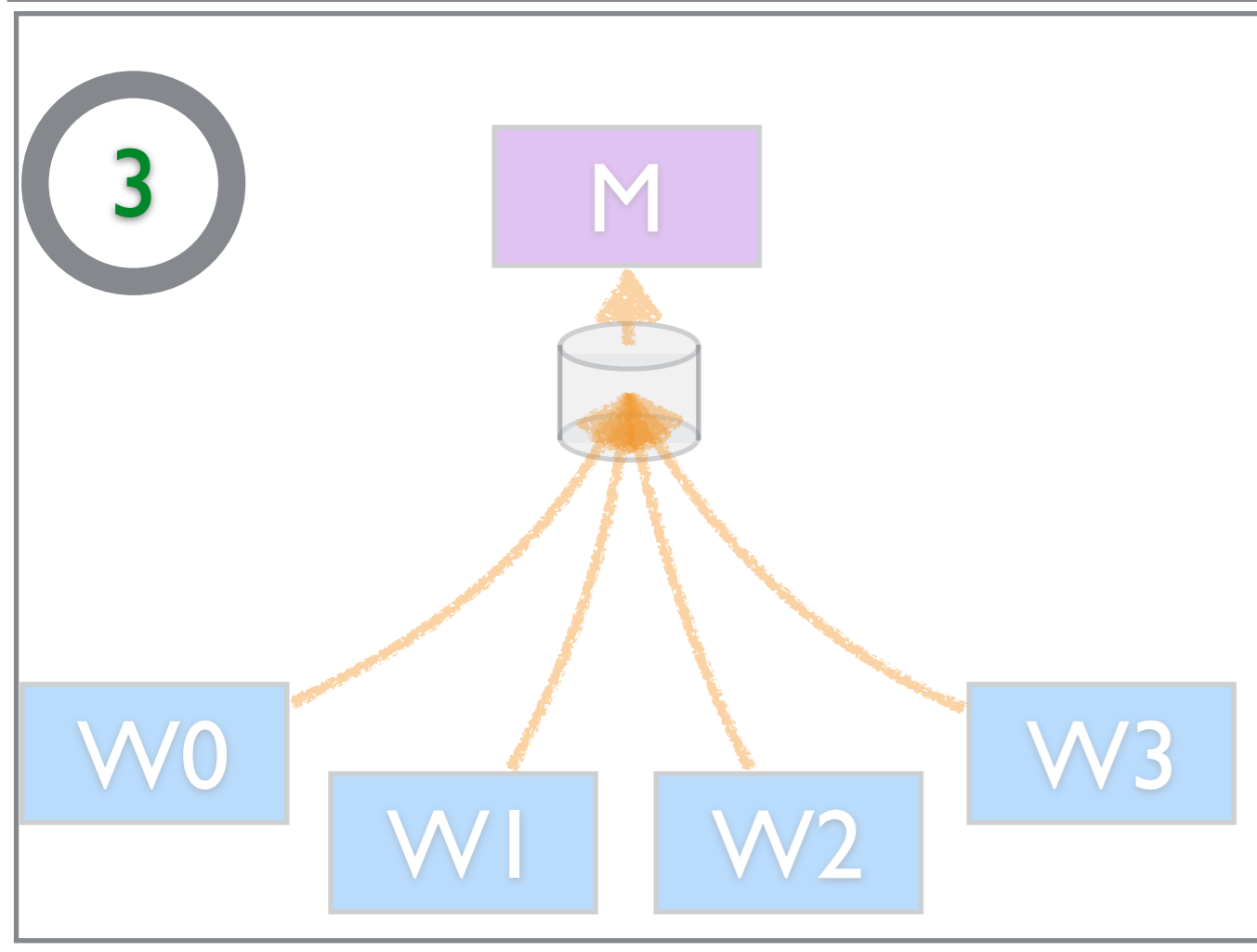
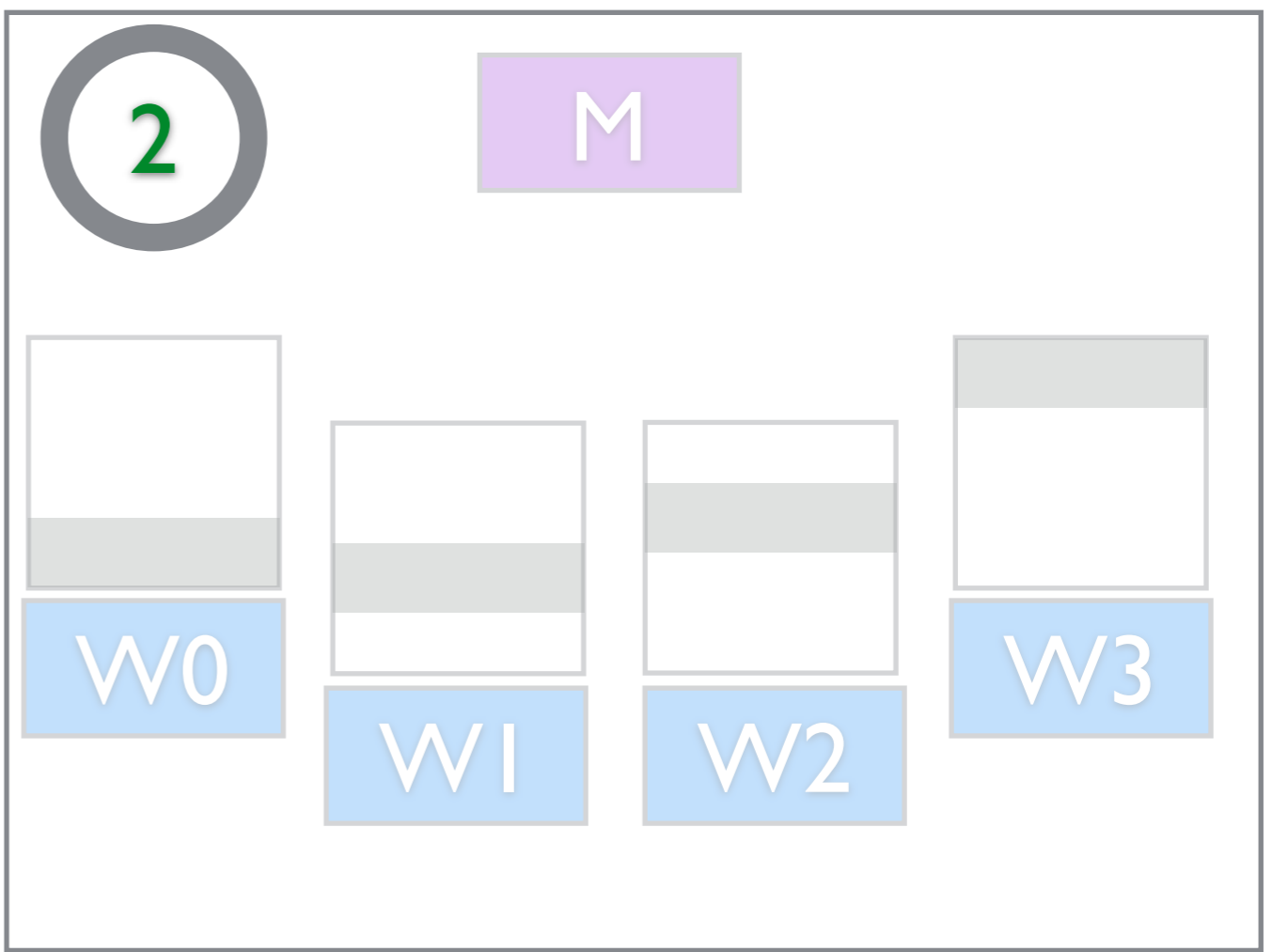
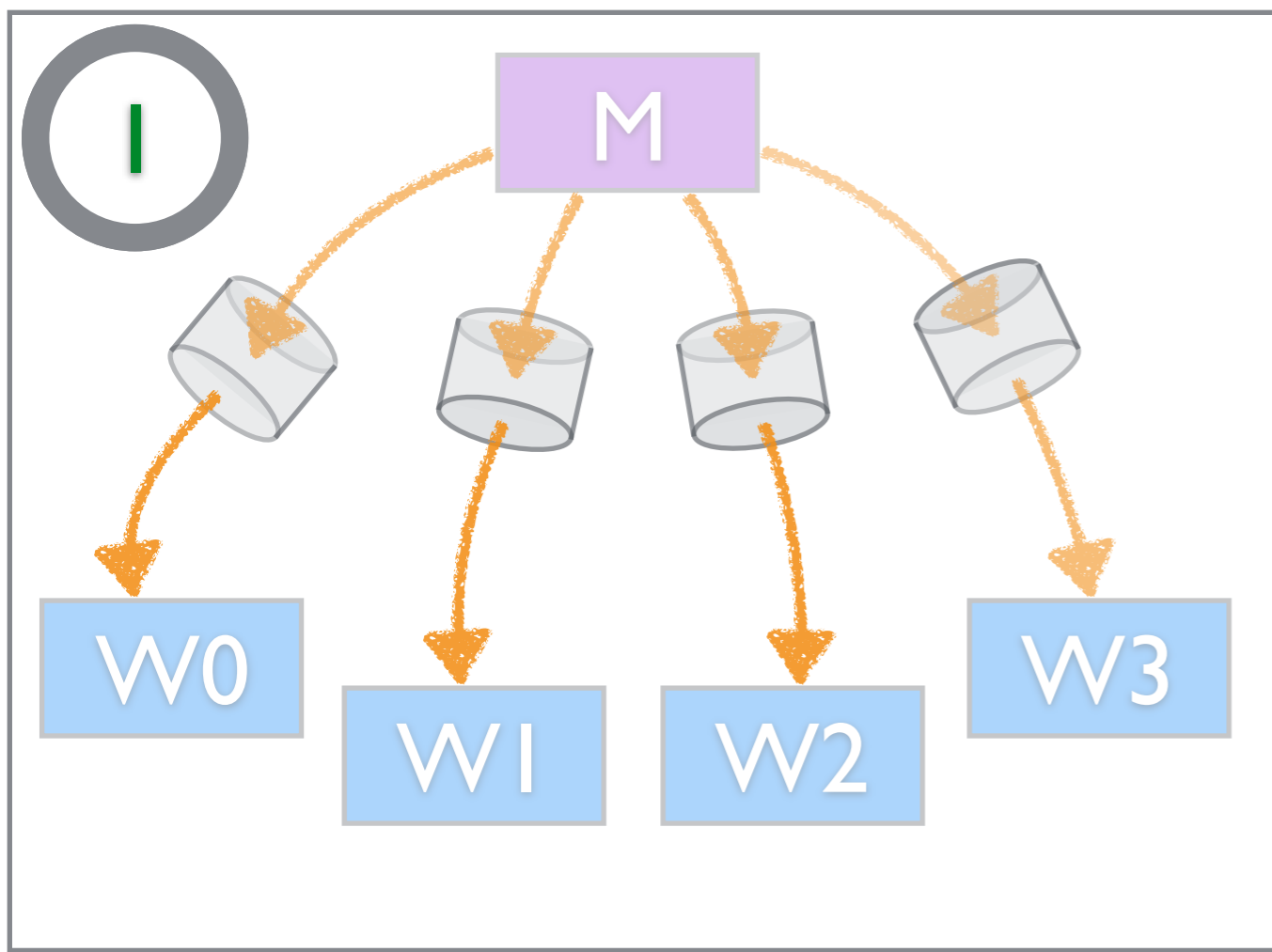
Communication Patterns

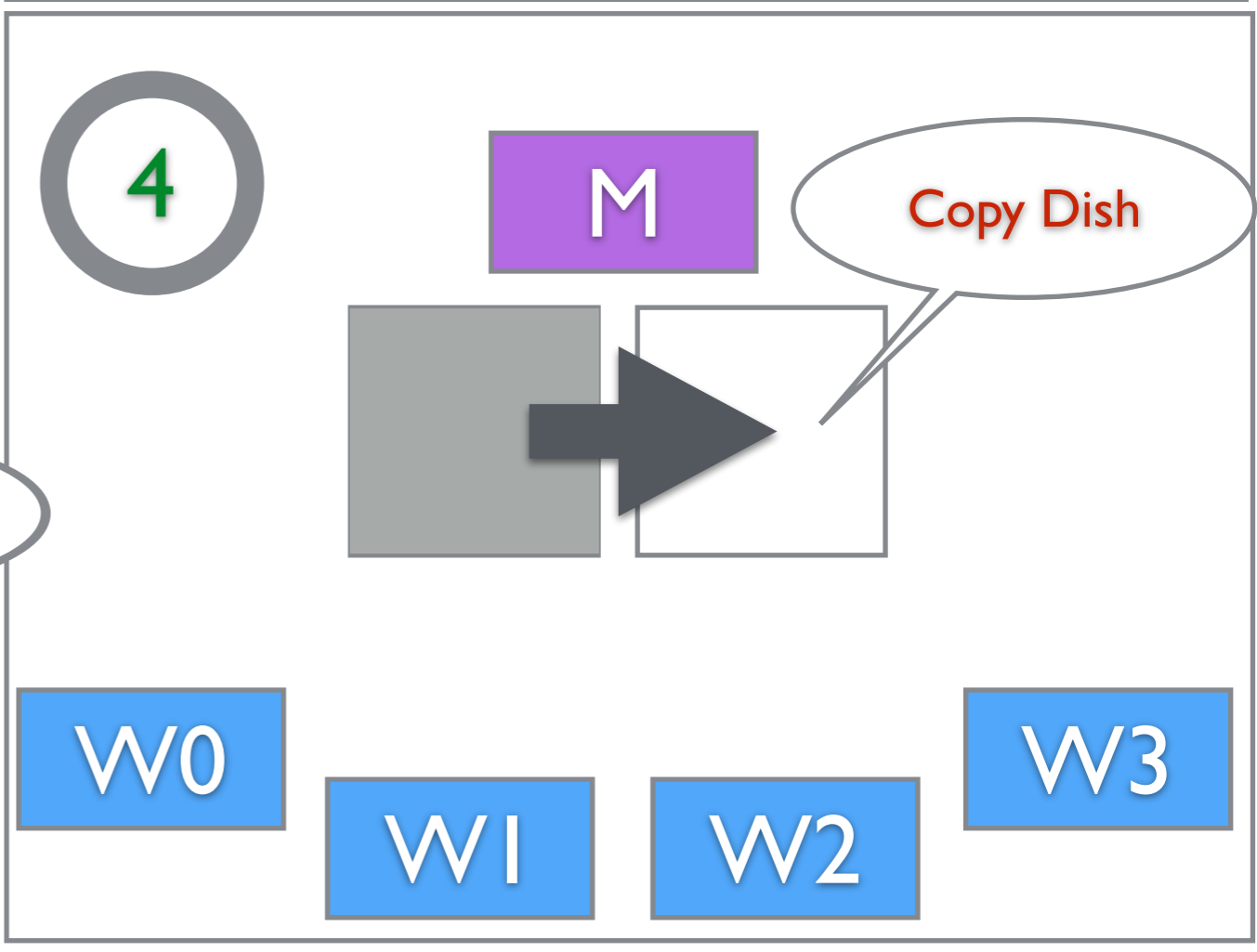
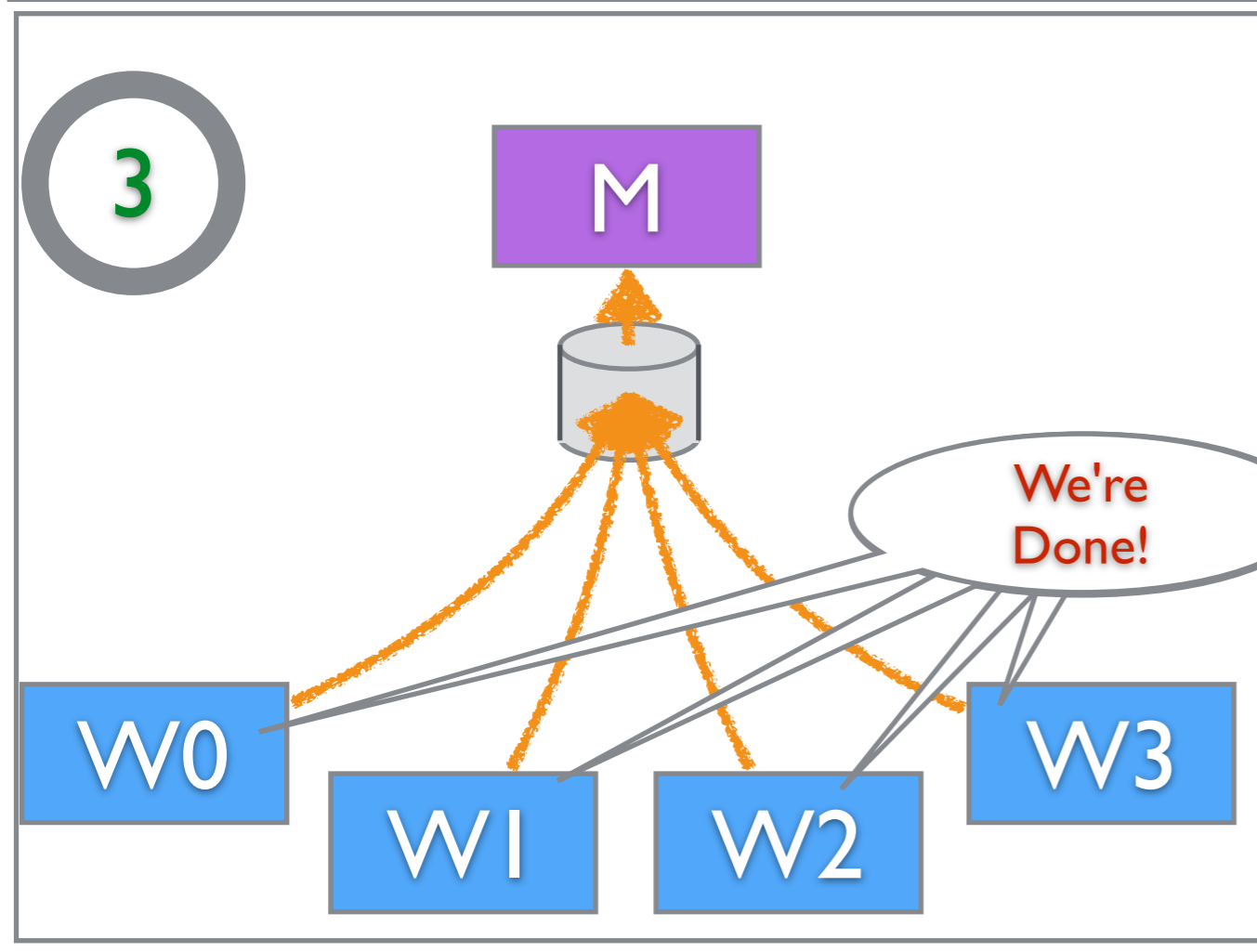
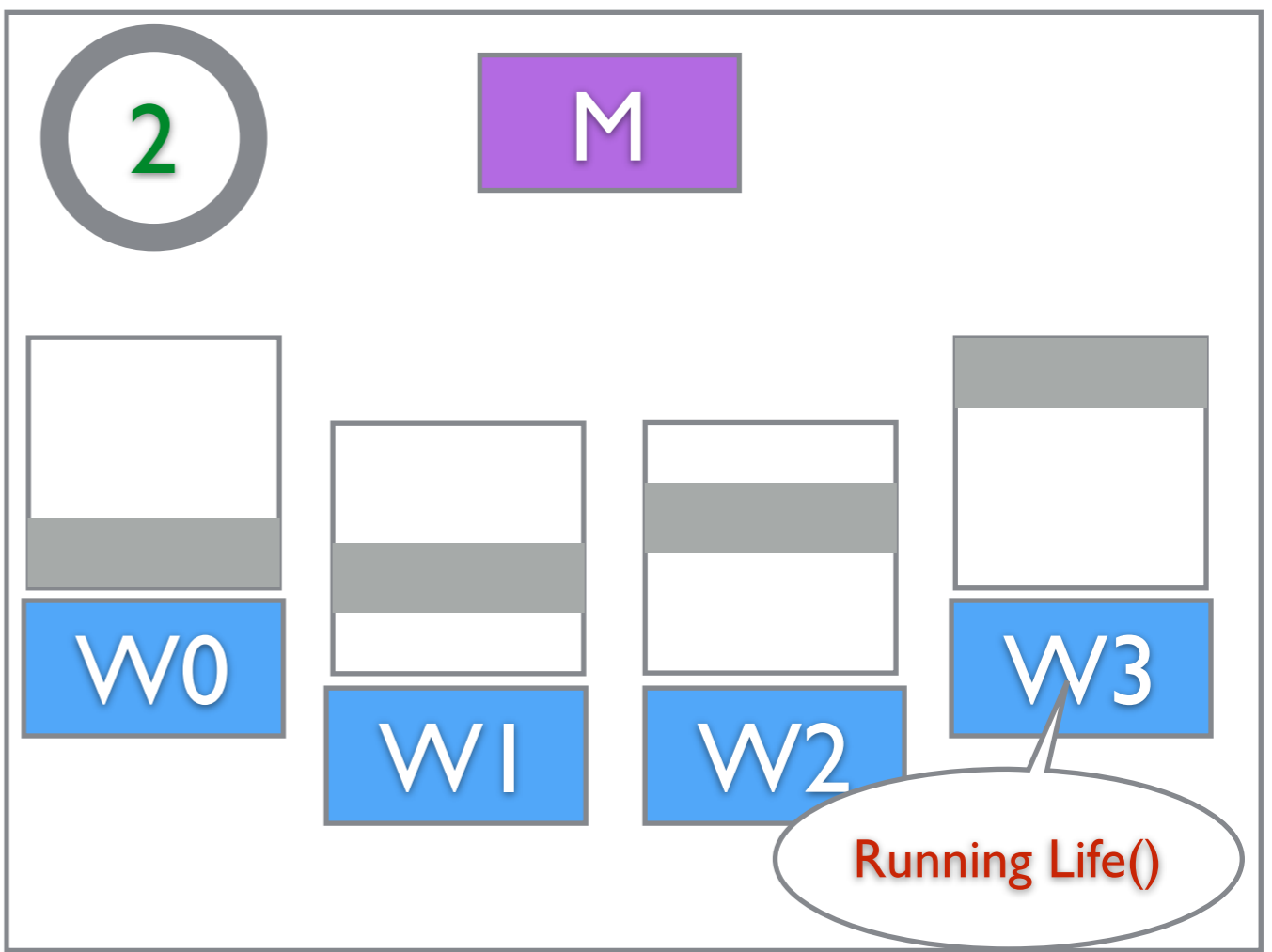
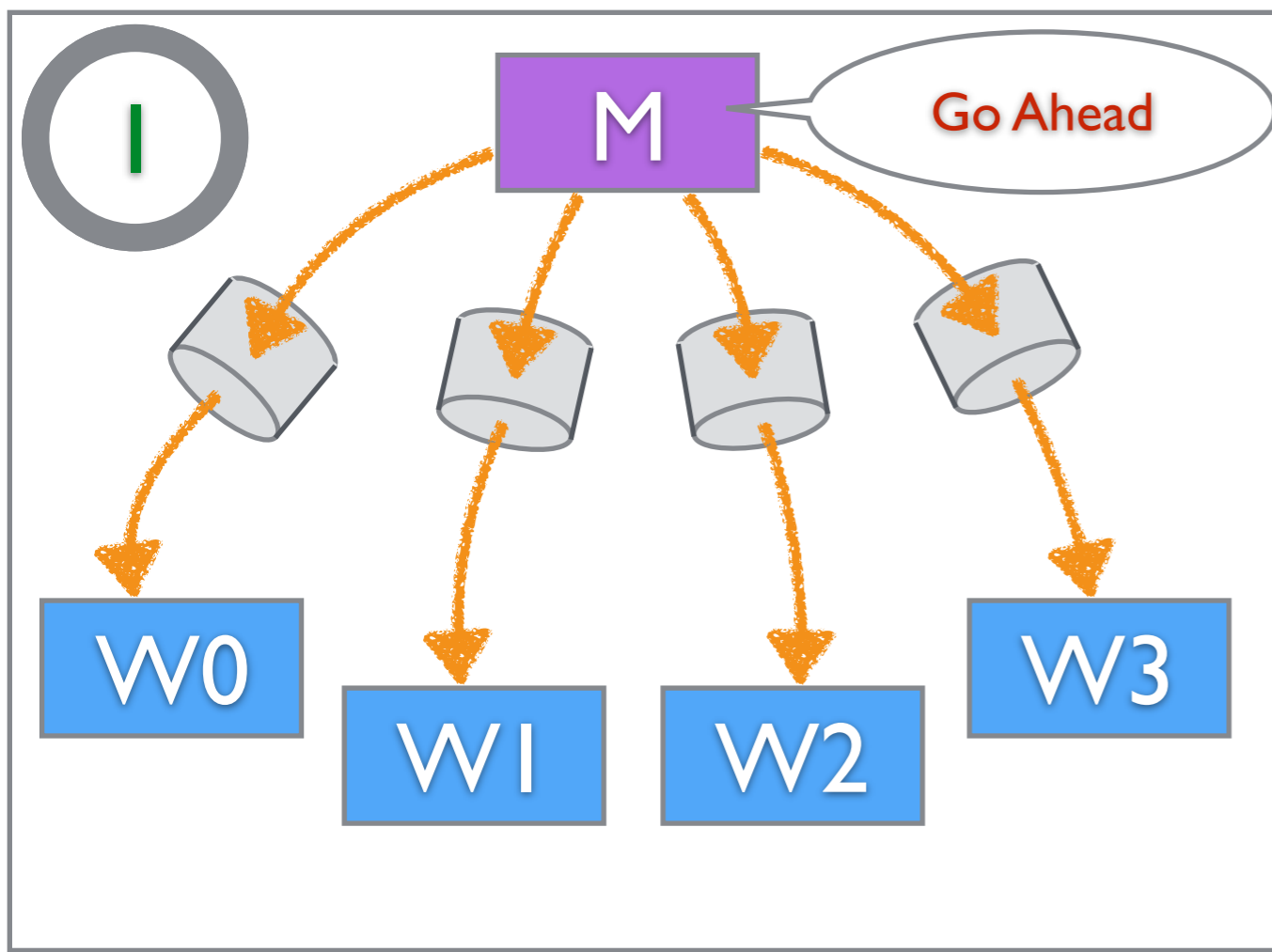












Thread Run()

```
/*
 * This will apply life repeatedly to the half of the dish array that
 * belongs to this thread.
 *
 * @see java.lang.Thread#run()
 */
public void run() {
    int noRows    = dataN.dish.length;           // number of rows in dish
    int startRow  = Id * noRows / N;            // where thread starts computing life
    int endRow    = ((Id+1) * noRows / N) ;     // where thread stop computing life
    if ( Id == N-1 ) endRow = noRows;          // if last thread, make sure include last

    int command = 1;
    while ( command != 0 ) {
        life( startRow, endRow );
        // send a token, wait for token
        try {
            toManager.put( Id );
            command = fromManager.take();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

getcopy GameOfLifeNThreads.java

Manager Main Loop

```
// go through generations
for ( int i = 0; i < gens; i++ ) {
    if ( verbose ) System.out.println( "generation " + i );

    // wait for N threads to finish their computation of
    // life
    for ( int t=0; t<N; t++ ) {
        try {
            int c = QFromThreads.take();
            if ( verbose ) System.out.println( " Received \"Done\" message from Thread " + c );
        } catch (InterruptedException e) {
        }
    }

    if ( verbose ) System.out.println( " Heard from all threads..." );

    // copy newGen to dish
    for ( int row=0; row < dataN.dish.length; row++ )
        dataN.dish[row] = dataN.newGen[row];

    if ( verbose ) System.out.println( " dish copied" );
    //print( dataN.dish );

    // tell N threads to resume computation
    for ( int t=0; t<N; t++ ) {
        int command = 1;
        if (i==gens-1)
            command = 0;
        try {
            QToThreads[t].put( command );
        } catch (InterruptedException e) {
        }
    }

    if ( verbose ) System.out.println( " all threads alerted to continue..." );
}
}
```

getcopy GameOfLifeNThreads.java

Manager Main Loop

```
// go through generations
for ( int i = 0; i < gens; i++ ) {
    if ( verbose ) System.out.println( "generation " + i );

    // wait for N threads to finish their computation of
    // life
    for ( int t=0; t<N; t++ ) {
        try {
            int c = QFromThreads.take();
            if ( verbose ) System.out.println( " Received \"Done\" message from Thread " + c );
        } catch (InterruptedException e) {
        }
    }

    if ( verbose ) System.out.println( " Heard from all threads..." );

    // copy newGen to dish
    dataN.dish[row] = dataN.newGen[row];
    if ( verbose ) System.out.println( " dish copy" );
    //print( dataN.dish );

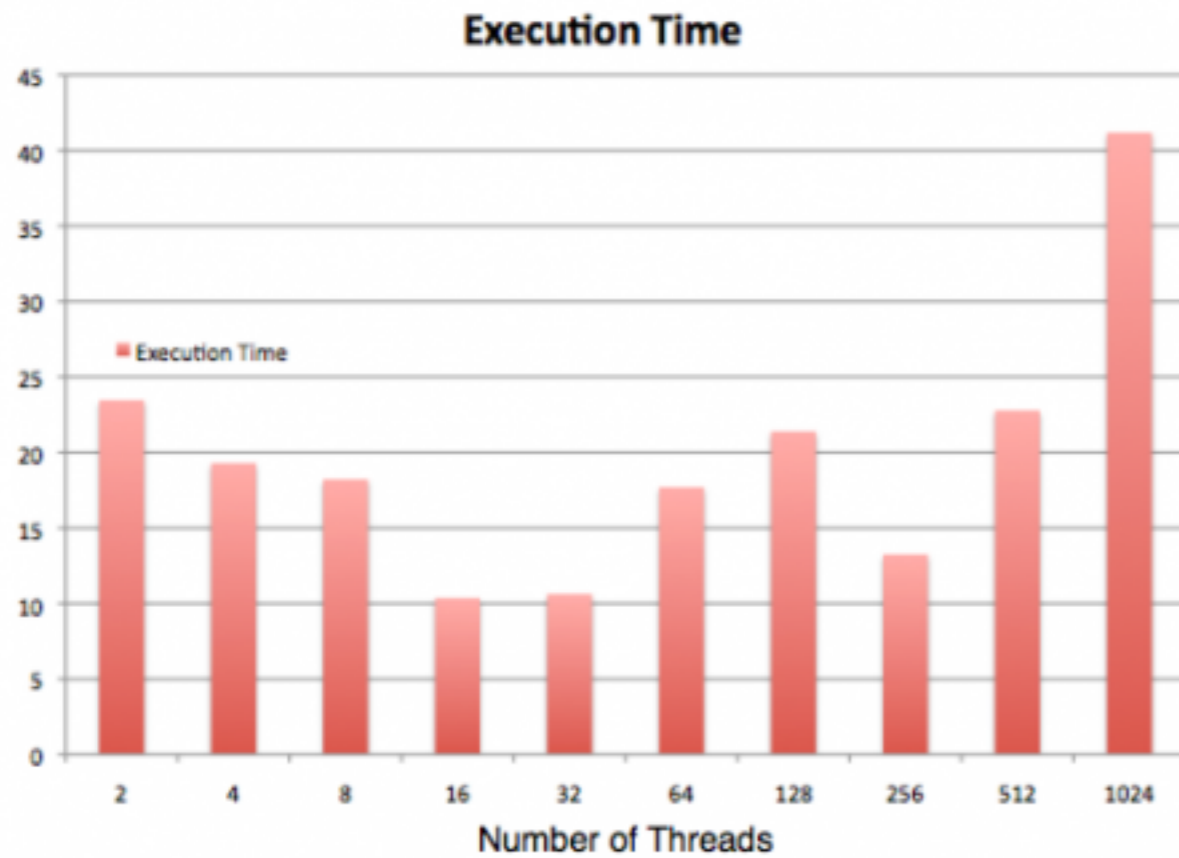
    // tell N threads to resume computation
    for ( int t=0; t<N; t++ ) {
        int command = 1;
        if ( i==gens-1 )
            command = 0;
        try {
            QToThreads[t].put( command );
        } catch (InterruptedException e) {
        }
    }

    if ( verbose ) System.out.println( " all threads alerted to continue..." );
}
}
```

http://www.science.smith.edu/dftwiki/index.php/CSC352_Game_of_Life_in_Java,_N_Threads#Source

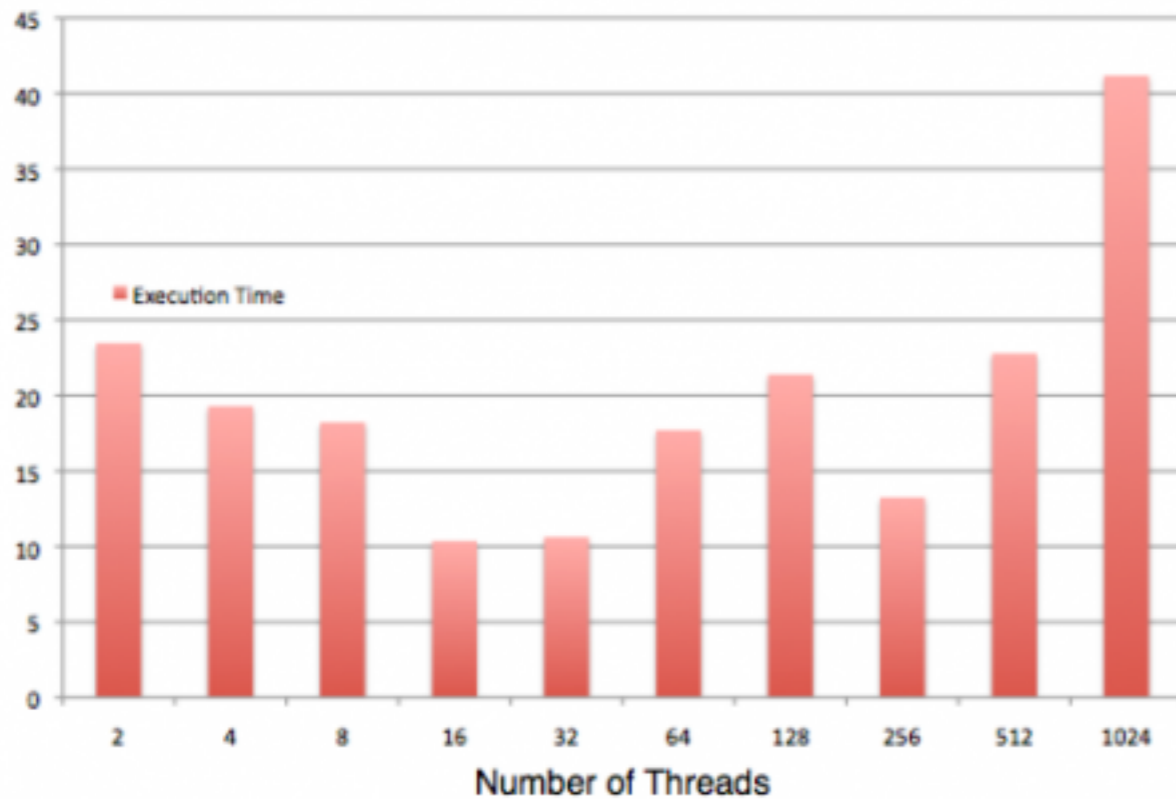
Performance

2048 rows, 80 cells wide
MacBook Pro 2014
4 cores, 2.8 GHz

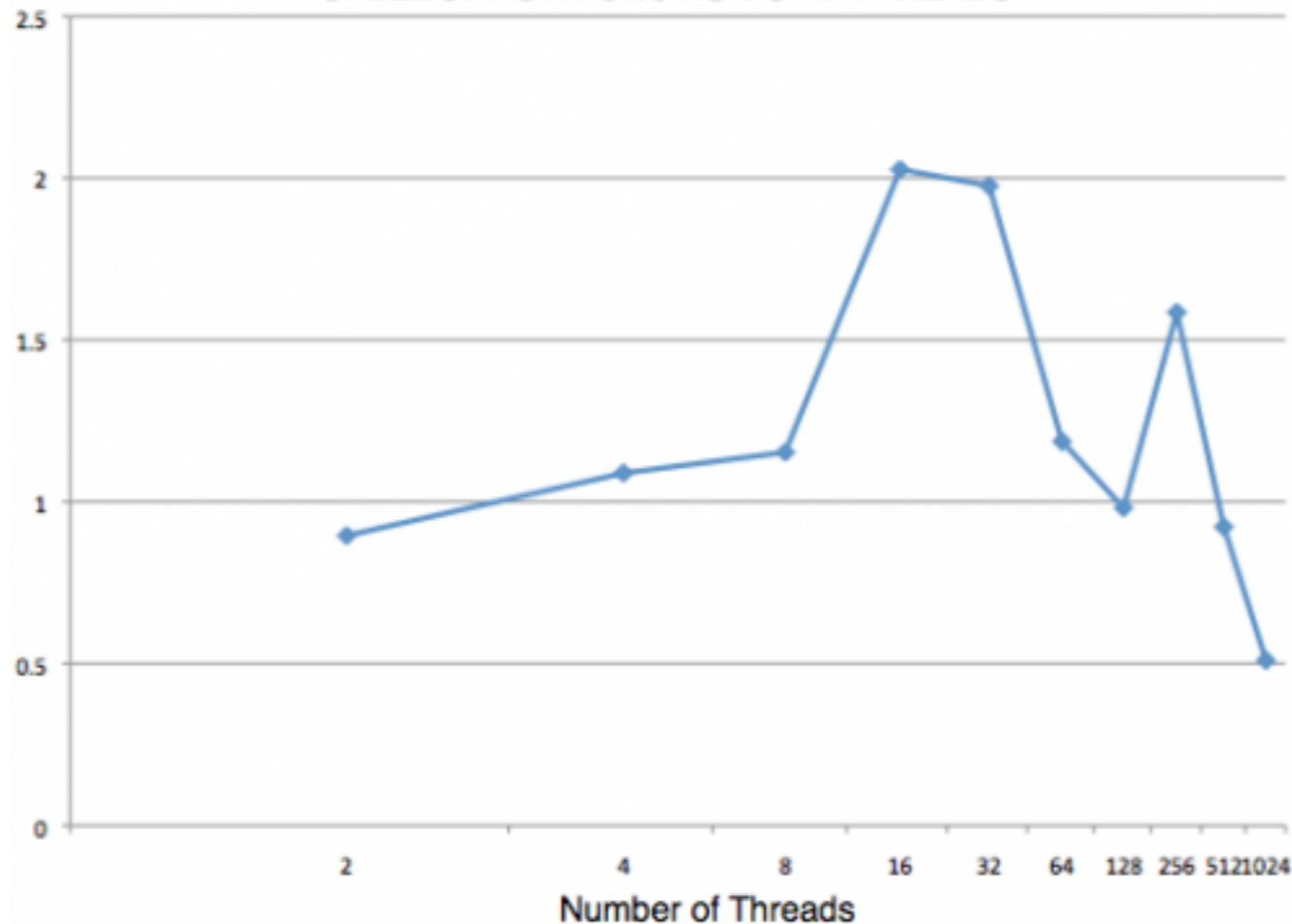


Performance

Execution Time



SPEEDUP AS A FUNCTION OF # THREADS



Threads	Speedup
2	0.895231676
4	1.088301802
8	1.152712505
16	2.026521961
32	1.976106799
64	1.186445303
128	0.981590412
256	1.584739771
512	0.921974566
1024	0.510000713



Questions

- What serial program should be used to compute the speedup?
- What does the speedup curve tell us?
- What is the general shape of the speedup curve? How could we make it "smoother"?
- What could be done to the Java program to make it faster? *Think hard...*

Back to MPI!

Point-to-Point Blocking Communication

MPI_Send

```
MPI_Send(&work,           // buffer
        1,                // number of items
        MPI_INT,          // type of items
        rank,             // Id of receiver
        tag,              // message tag (must match)
        MPI_COMM_WORLD); // the communicator's group
```

https://computing.llnl.gov/tutorials/mpi/#Derived_Data_Types

Basic blocking send operation. Routine returns only after the application buffer in the sending task is free for reuse. Note that this routine may be implemented differently on different systems.

<https://computing.llnl.gov/tutorials/mpi/>

```
MPI_CHAR
MPI_SHORT
MPI_INT
MPI_LONG
MPI_UNSIGNED
MPI_FLOAT
MPI_DOUBLE
```

MPI_Recv

```
MPI_Recv(&result,           // buffer
        1,                 // # items
        MPI_DOUBLE,        // item type
        MPI_ANY_SOURCE,    // receive from any sender
        MPI_ANY_TAG,       // any tag
        MPI_COMM_WORLD,    // default communicator
        &status);          // info about the received
                          // message
```

Receive a message and block until the requested data is available in the application buffer in the receiving task.

<https://computing.llnl.gov/tutorials/mpi/>

In C, status is a structure that contains three fields named MPI_SOURCE, MPI_TAG, and MPI_ERROR; the structure may contain additional fields. Thus, status.MPI_SOURCE, status.MPI_TAG and status.MPI_ERROR contain the source, tag, and error code, respectively, of the received message.

<http://www.mpi-forum.org/docs/mpi-1.1-html/node35.html#Node35>

Status Structure

```
int recvd_tag, recvd_from;
int recvd_count;
MPI_Status status;

MPI_Recv( ..., ..., ..., &status );

Recvd_tag = status.MPI_TAG;
Recvd_from = status.MPI_SOURCE;
MPI_Get_count( &status,
               datatypeOfbuffer,
               &recvd_count );
```



Definition

MPI_COMM_WORLD

- MPI_COMM_WORLD is a *Communicator*
- It contains ALL processes
- A communicator determines the scope and the "*communication universe*" in which a point-to-point or collective operation is to operate.
- It's the *universe* for most MPI programs

HOT POTATO



Worker 1



Manager

Worker 2

Worker N-1

Manager



Worker I

Worker 2

Worker N-1

Worker 1



Manager

Worker 2

Worker N-1

Worker 1

Manager

Worker 2

Worker N-1



Worker 1

Manager

Worker 2

Worker N-1



Worker 1

Worker 2

Worker N-1

Manager

!!!!





- Implement the Hot Potato game in MPI and run it on **Aurora**. Make the potato go once around. (What is a good data-type to use for the potato?)
- Make each node of the cluster increment the potato as it passes around. Make the manager print the value of the potato when it gets it.
- Modify the MPI program so that the number of rounds can be specified on the command line.

```
#include <stdlib.h>

int N;
N = atoi( argv[1] );
```

First program generated in class

```
/* hotPotato.c
D. Thiebaut
Hot potato passing around, with N processes.
```

The manager (Process 0) passes the potato to Worker 1, which passes it to worker 2, etc... until the manager gets it back again and quits.

```
*/
#include <stdio.h>
#include <mpi.h>

#define MANAGER 0
int main ( int argc, char *argv[] ) {
    int rank, size;
    int potato = 1213;
    MPI_Status status; // required variable for receive routines

    MPI_Init (&argc, &argv); // starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); // get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size); // get number of processes */

    if ( rank == MANAGER ) {
        //      buffer  #items  item-size src/dest      tag  world
        MPI_Send( &potato, 1,      MPI_INT,  1,          0,      MPI_COMM_WORLD );
        printf( "Manager sent potato!\n" );
        MPI_Recv( &potato, 1,      MPI_INT,  size-1,      0,      MPI_COMM_WORLD, &status );
        printf( "Manager got potato back!\n" );
    }
    else {
        MPI_Recv( &potato, 1,      MPI_INT,  rank-1,      0,      MPI_COMM_WORLD, &status );
        printf( "Worker %d got potato!\n", rank );
        MPI_Send( &potato, 1,      MPI_INT,  (rank+1)%size, 0,      MPI_COMM_WORLD );
    }

    MPI_Finalize();
    return 0;
}
```

First program generated in class

```
/* hotPotato.c
D. Thiebaut
Hot potato passing around, with N processes.
```

The manager (Process 0) passes the potato to Worker 1, which passes it to worker 2, etc... until the manager gets it back again and quits.

```
*/
#include <stdio.h>
#include <mpi.h>

#define MANAGER 0
int main ( int argc, char *argv[] )
{
    int rank, size;
    int potato = 1213;
    MPI_Status status; // required

    MPI_Init (&argc, &argv); // starts MPI */
    MPI_Comm_rank (MPI_COMM_WORLD, &rank); // get current process id */
    MPI_Comm_size (MPI_COMM_WORLD, &size); // get number of processes */

    if ( rank == MANAGER ) {
        //      buffer  #items  item-size src/dest      tag  world
        MPI_Send( &potato, 1,      MPI_INT,  1,          0,    MPI_COMM_WORLD );
        printf( "Manager sent potato!\n" );
        MPI_Recv( &potato, 1,      MPI_INT,  size-1,      0,    MPI_COMM_WORLD, &status );
        printf( "Manager got potato back!\n" );
    }
    else {
        MPI_Recv( &potato, 1,      MPI_INT,  rank-1,      0,    MPI_COMM_WORLD, &status );
        printf( "Worker %d got potato!\n", rank );
        MPI_Send( &potato, 1,      MPI_INT,  (rank+1)%size, 0,    MPI_COMM_WORLD );
    }

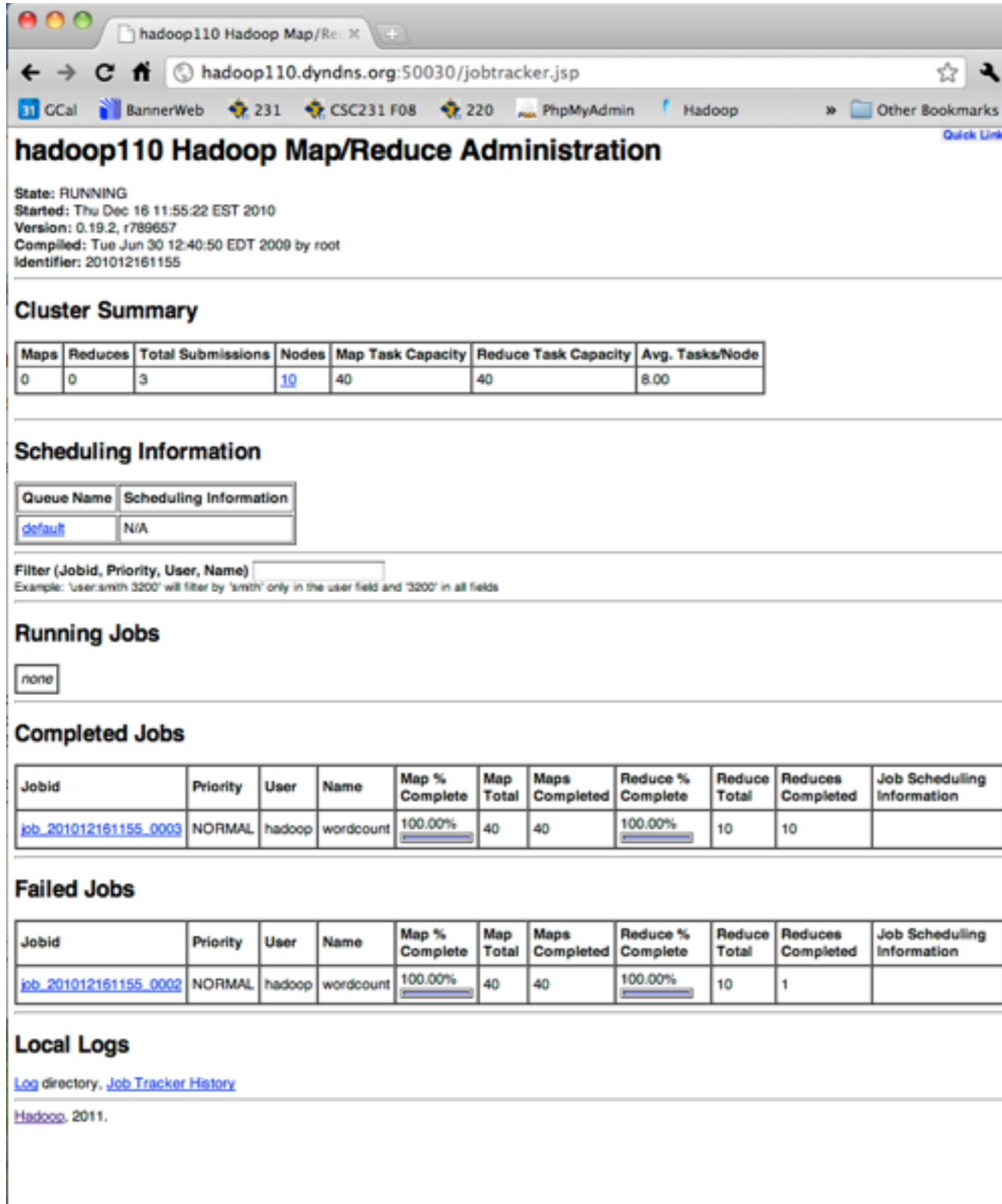
    MPI_Finalize();
    return 0;
}
```

```
352b@aurora ~/handout/mpi $ mpicc hotPotato.c -o hotPotato
352b@aurora ~/handout/mpi $ mpirun -np 2 ./hotPotato
Manager sent potato!
Worker 1 got potato!
Manager got potato back!
352b@aurora ~/handout/mpi $
```


We Stopped Here Last Time



Misc Notes on Paper Presentations



The screenshot displays the Hadoop Web Console interface. At the top, the browser address bar shows the URL `hadoop110.dyndns.org:50030/jobtracker.jsp`. The page title is "hadoop110 Hadoop Map/Reduce Administration".

Key information displayed includes:

- State:** RUNNING
- Started:** Thu Dec 16 11:55:22 EST 2010
- Version:** 0.19.2, r789657
- Compiled:** Tue Jun 30 12:40:50 EDT 2009 by root
- Identifier:** 201012161155

The **Cluster Summary** section contains the following table:

Maps	Reduces	Total Submissions	Nodes	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node
0	0	3	10	40	40	8.00

The **Scheduling Information** section shows a table with the following data:

Queue Name	Scheduling Information
default	N/A

Below this is a filter input field: "Filter (Jobid, Priority, User, Name) ". An example is provided: "Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields".

The **Running Jobs** section shows a single entry: "none".

The **Completed Jobs** section contains a table with the following data:

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201012161155_0003	NORMAL	hadoop	wordcount	100.00%	40	40	100.00%	10	10	

The **Failed Jobs** section contains a table with the following data:

Jobid	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information
job_201012161155_0002	NORMAL	hadoop	wordcount	100.00%	40	40	100.00%	10	1	

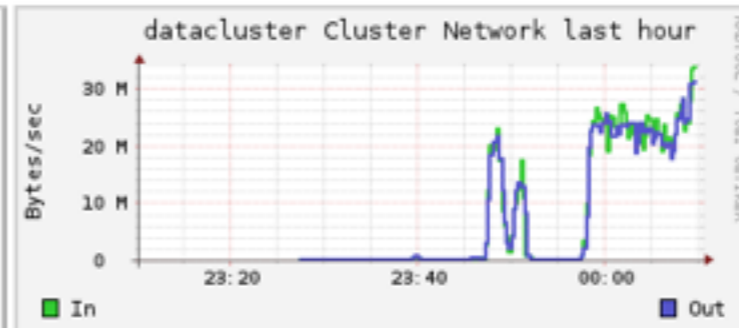
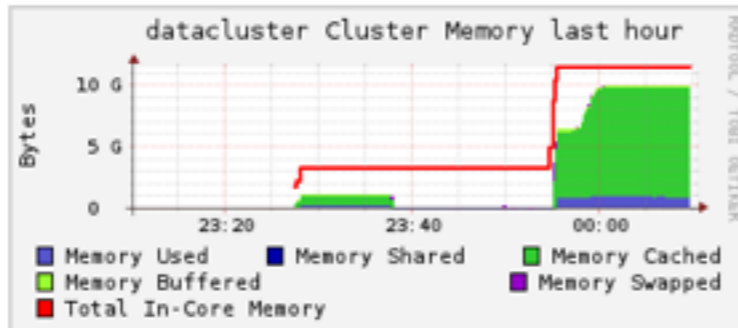
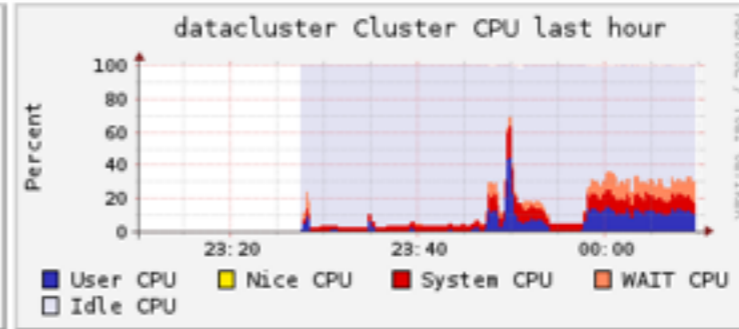
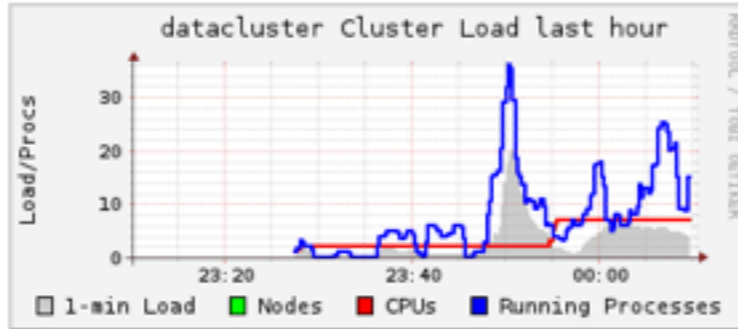
The **Local Logs** section includes links for "Log directory" and "Job Tracker History", and a footer note: "Hadoop, 2011."

Hadoop Web Console

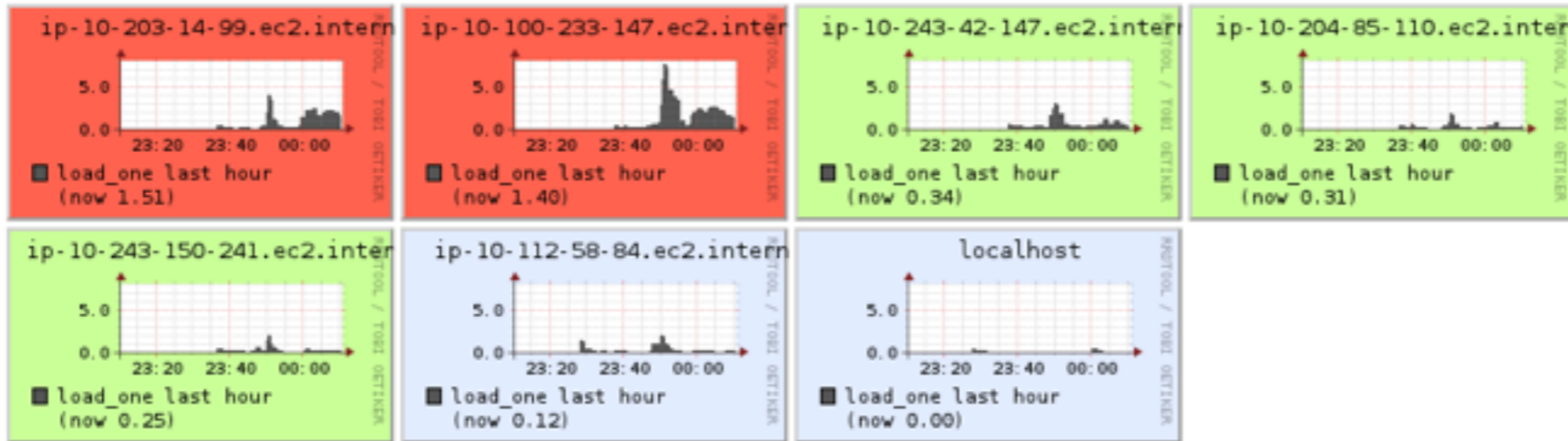
Overview of datacluster

CPU's Total: 7
Hosts up: 7
Hosts down: 0

Avg Load (15, 5, 1m): 58%, 66%, 56%
Localtime: 2011-09-03 00:09



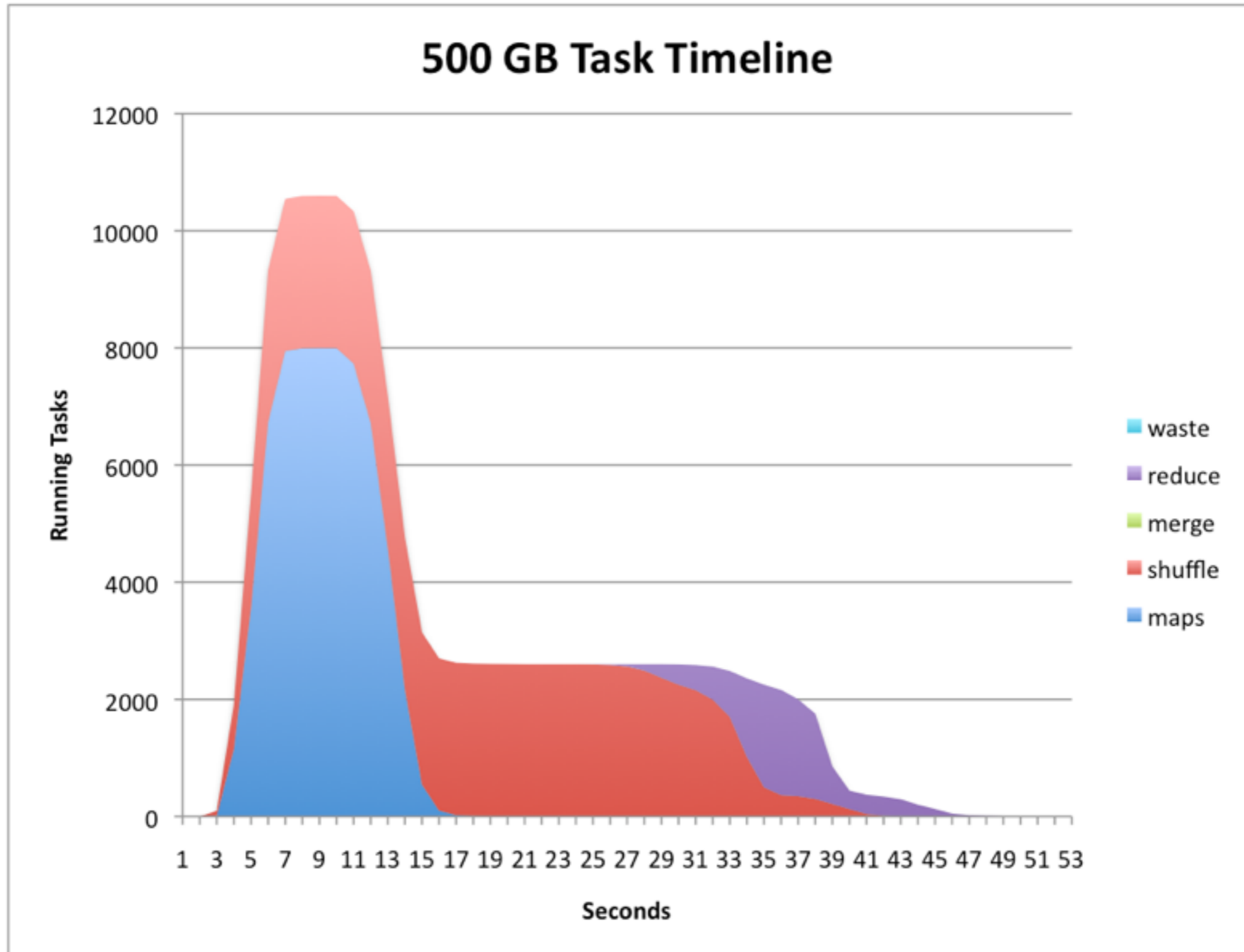
Show Hosts: yes no | datacluster load_one last hour sorted descending | Columns 4 | Size small



(Nodes colored by 1-minute load) | Legend

Ganglia Console for Hadoop

Typical Map-Reduce Job



In reference to “Nobody ever got fired...”

PROCESSING WIKIPEDIA DUMPS *A Case-Study comparing the XGrid and MapReduce Approaches*

Dominique Thiébaud, Yang Li, Diana Jaunzeikare, Alexandra Cheng, Ellysha Raelen Recto,
Gillian Riggs, Xia Ting Zhao, Tonje Stolpestad, Cam Le T Nguyen
Dept. Computer Science, Smith College, Northampton, MA, USA
{*dthiebau, yli2, djaunzei, acheng,erecto, griggs, xzhao,tstolpes,lnguyen*}@smith.edu

Keywords: Grid Computing, XGrid, Hadoop, wikipedia, data mining, performance .

Abstract: We present a simple comparison of the performance of three different cluster platforms: Apple’s XGrid, and Hadoop the open-source version of Google’s MapReduce as the total execution time taken by each to parse a 27-GBYTE XML dump of the English wikipedia. A local hadoop cluster of Linux workstation, as well as an Elastic MapReduce cluster rented from Amazon are used. We show that for this specific workload, XGrid yields the fastest execution time, with the local Hadoop cluster a close second. The overhead of fetching data from Amazon’s Simple Storage System (S3), along with the inability to skip the reduce, sort, and merge phases on Amazon penalizes this platform targeted for much larger data sets.

1 INTRODUCTION

The aim of this paper is to find the fastest parallel computer cluster available at Smith College for processing Wikipedia dumps and for generating word statistics. We implement the parsing of the XML dump of the English wikipedia, and the gathering of word usage on an Apple XGrid system, and on two Hadoop clusters, one local, the other hosted on Amazon. We report on the performance obtained from

paper we investigate whether hadoop is a contender, and under which conditions the execution time of the parsing process is comparable on both frameworks.

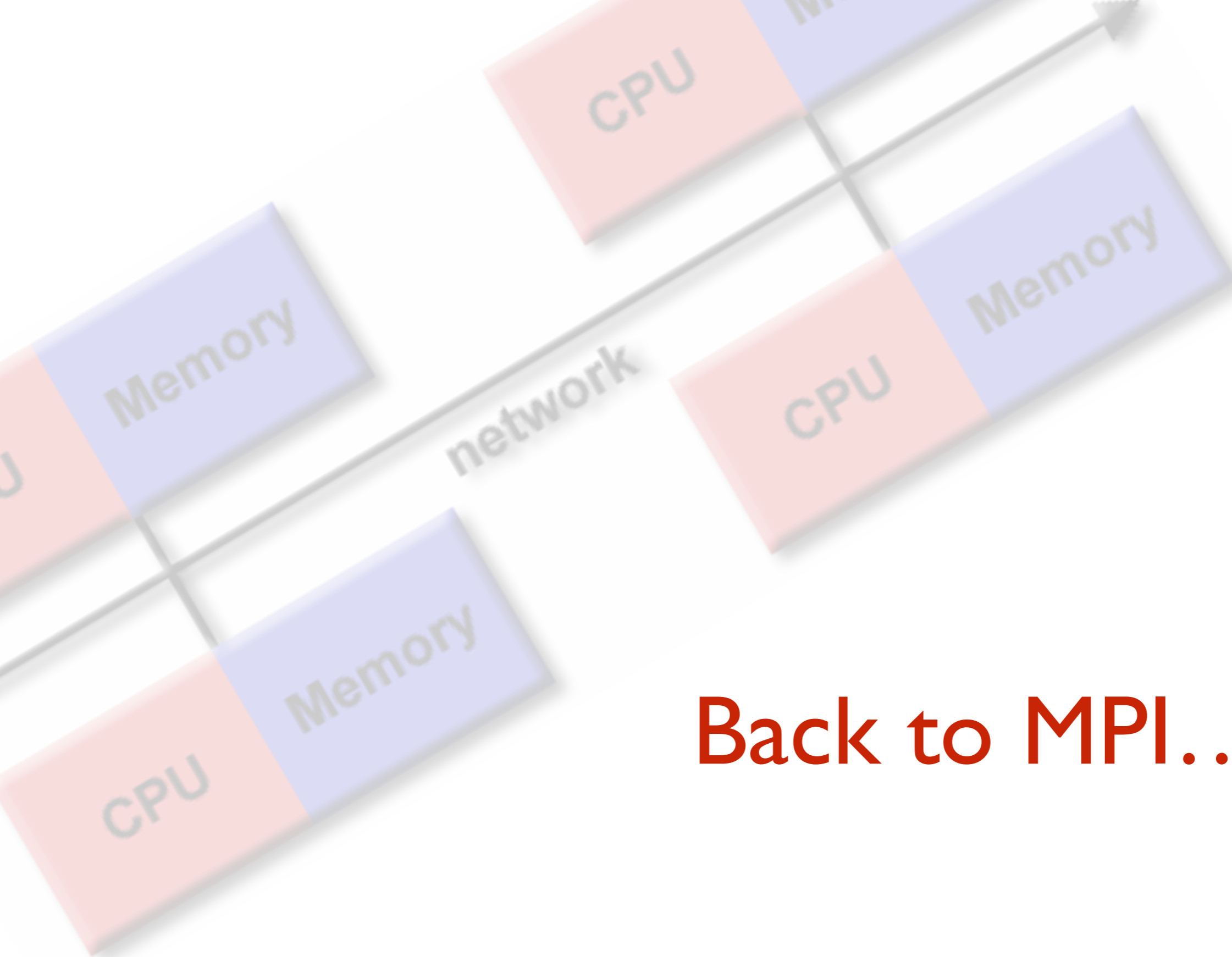
When low-cost, powerful, and easily accessible parallel computational platforms are available, it is important to better understand their source of performance and pick the best one for a solving a given problem.

AWS 2015 NYC



SC 16 Salt Lake City





Back to MPI...


```
3.141592653589793238462643383279
5028841971693993751058209749445923
07816406286208998628034825342117067
9821      48086      5132
823      06647      09384
46      09550      58223
17      25359      4081
          2848      1117
          4502      8410
          2701      9385
          21105     55964
          46229     48954
          9303      81964
          4288      10975
          66593     34461
          284756    48233
          78678     31652      71
          2019091   456485     66
          9234603   48610454326648
2133936   0726024914127
3724587   00660631558
817488    152092096
```

An Example: Computing Pi With MPI

$$\delta = 1/N$$

$$\pi = 4\delta(4 + 4/(1 + \delta^2) + 4/(1 + (2\delta)^2) + \dots + 4/(1 + ((N - 1)\delta)^2))$$

```
// pi.c
#include <stdlib.h>
#include <stdio.h>

double f( double x ) { return 4.0 / ( 1 + x*x );}

int main( int argc, char *argv[] ) {
    int N, i;
    double deltaX, sum;

    if ( argc < 2 ) {
        printf( "Syntax %s N\n", argv[0] );
        exit(1);
    }

    N = atoi( argv[1] );
    sum = 0;
    deltaX = 1.0/N;

    for ( i = 0; i < N; i++ )
        sum += f( i * deltaX );

    printf( "%d iterations: Pi = %.6f\n", N, sum*deltaX );
}
```

Computing Pi

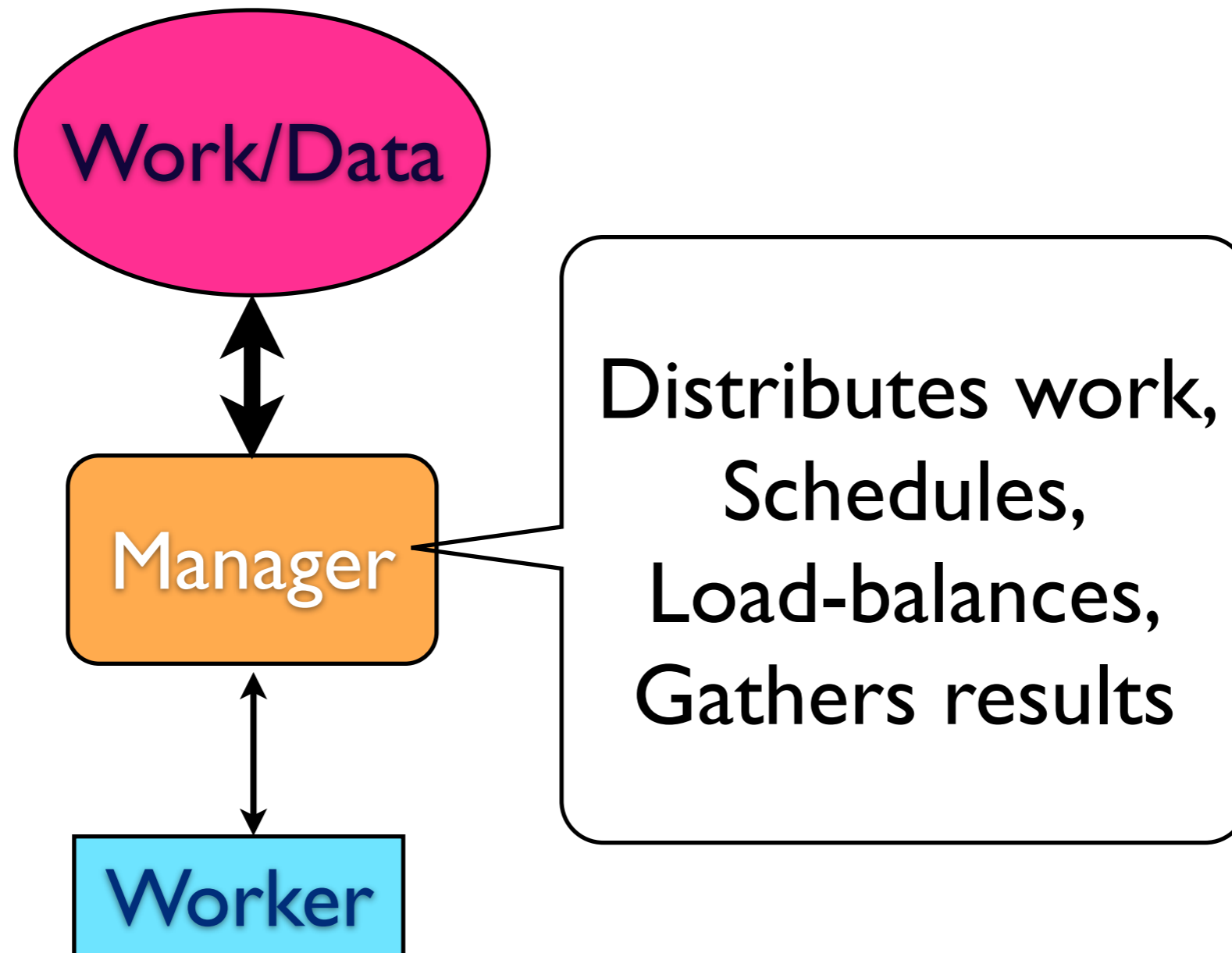
(serial version in C)

```
cc -o pi pi.c
./pi 100000
100000 iterations: Pi = 3.141603
```

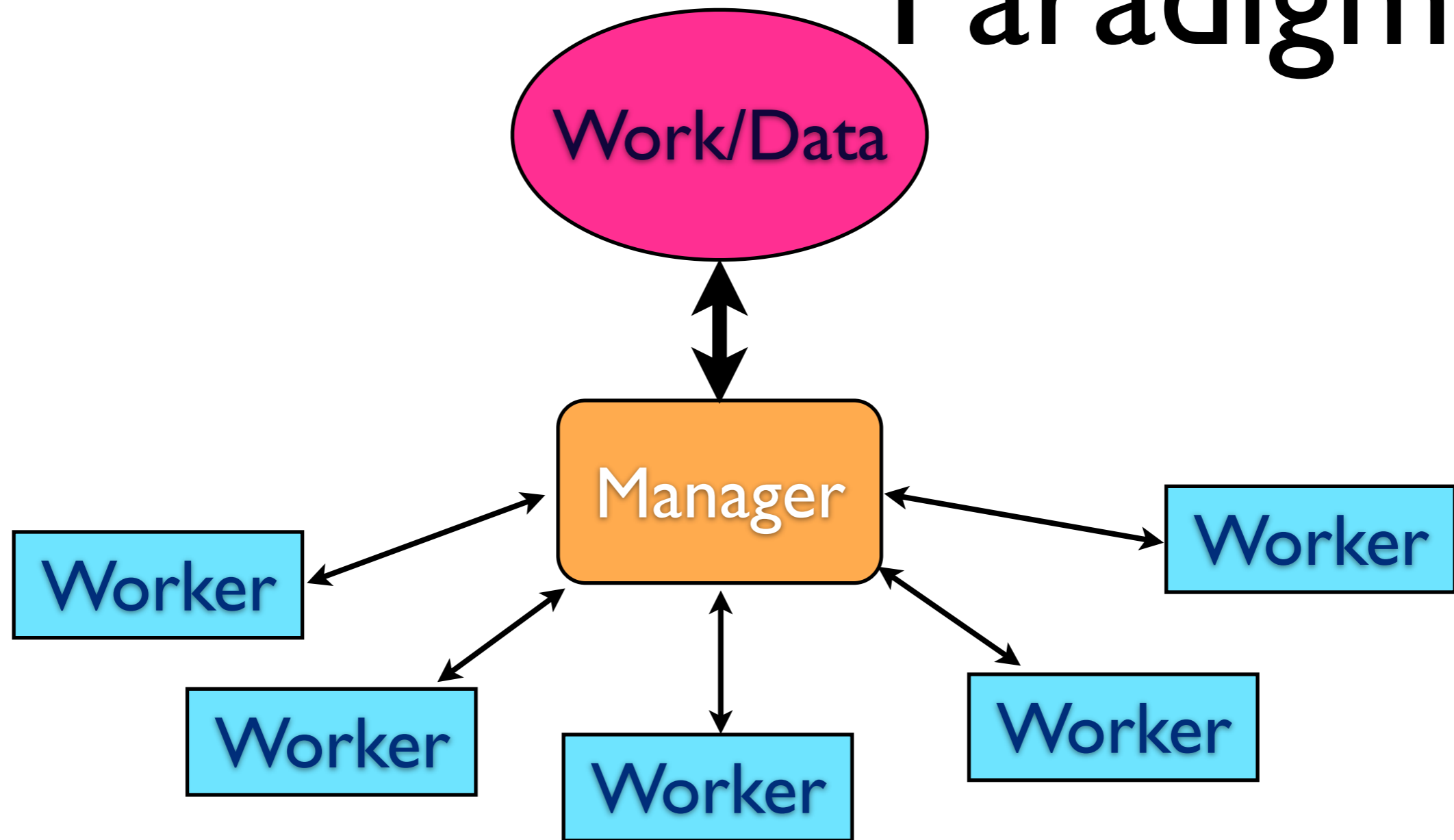
[getcopy mpi/pi.c](#)



Manager/Worker Paradigm



Manager/Worker Paradigm

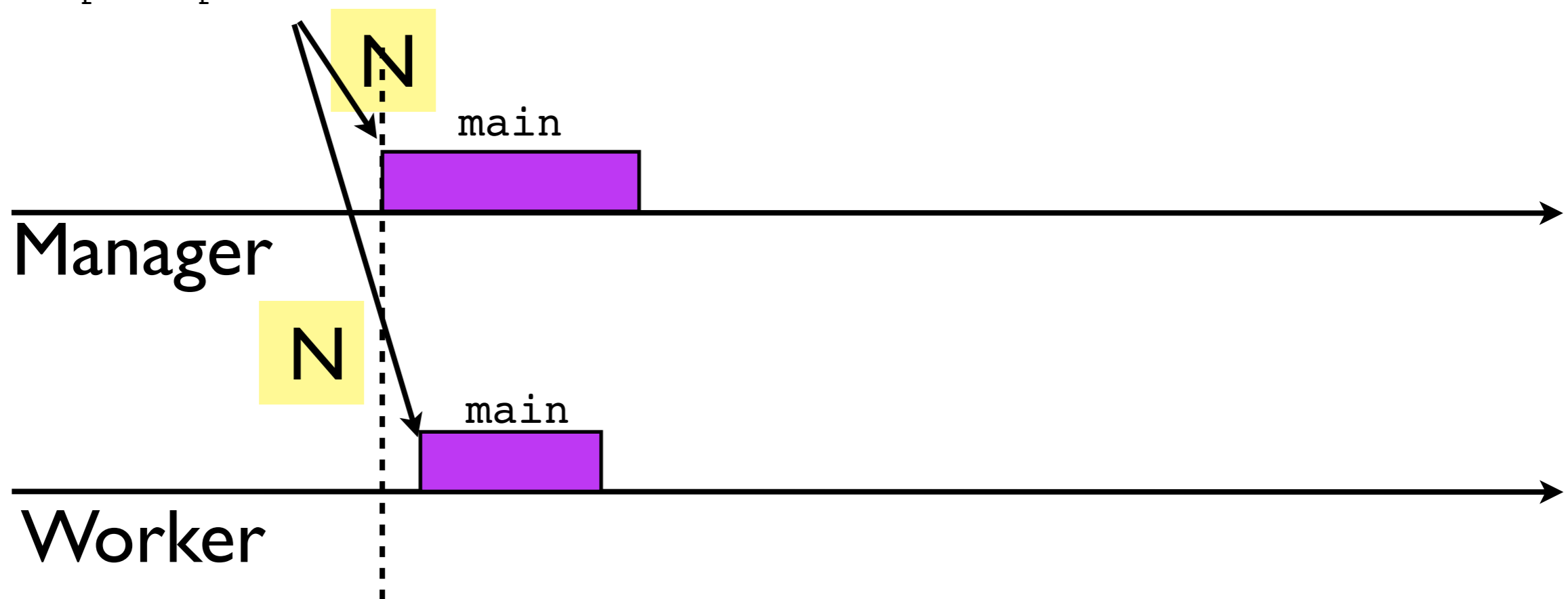


Computing Pi

(Parallel version in MPI)

- Manager/Worker setup

```
mpirun -np 2 ./pi2 N
```

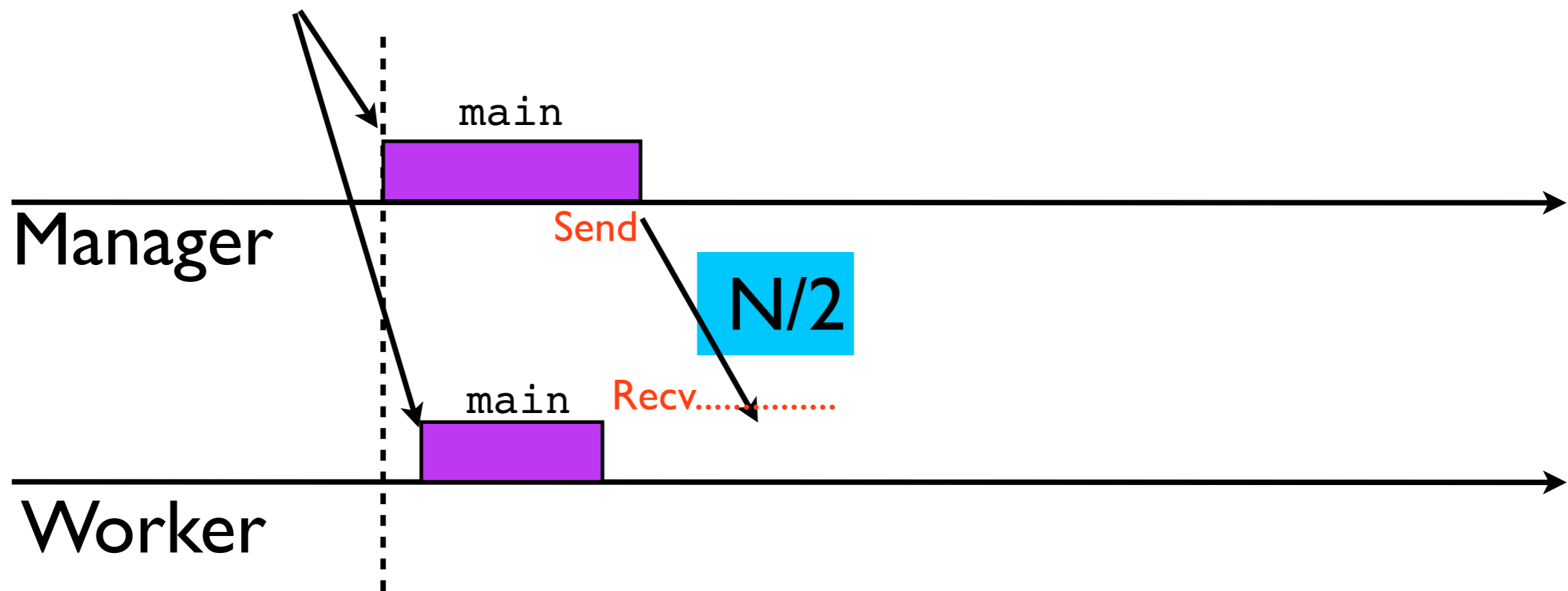


Computing Pi

(Parallel version in MPI)

- Manager/Worker setup

```
mpirun -np 2 ./pi2 N
```

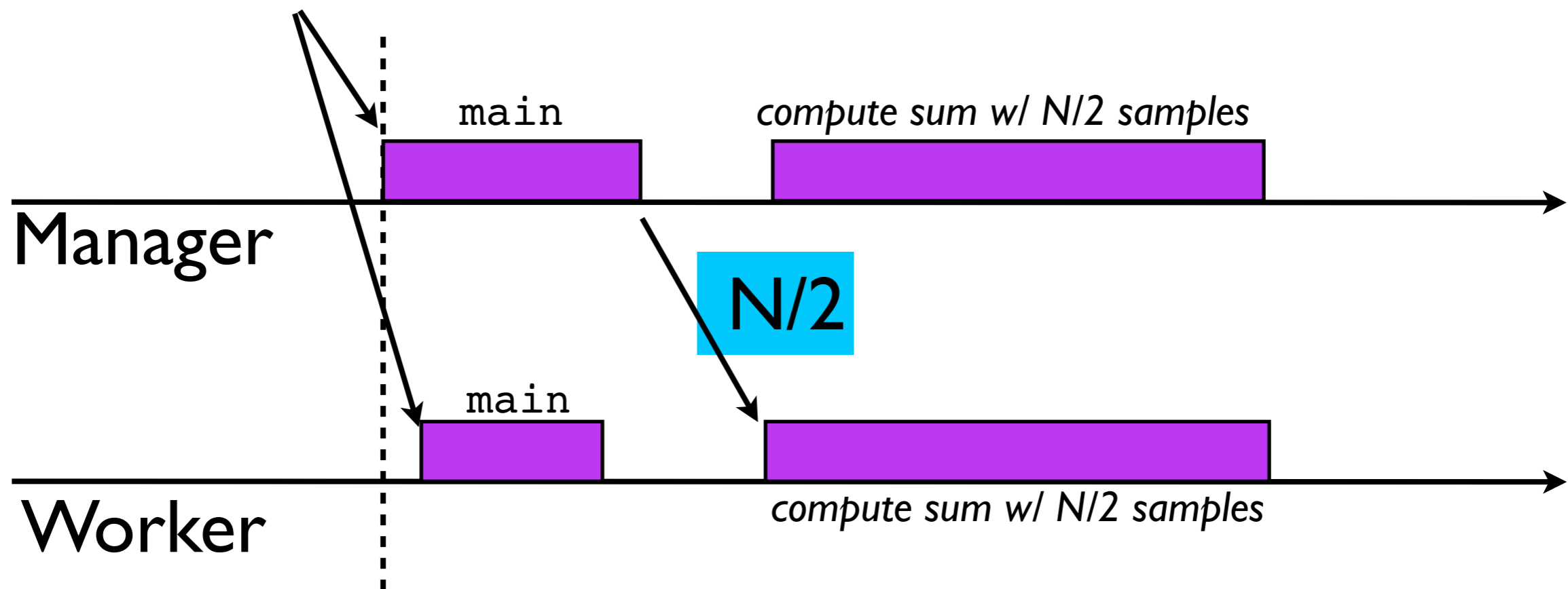


Computing Pi

(Parallel version in MPI)

- Manager/Worker setup

```
mpirun -np 2 ./pi2 N
```

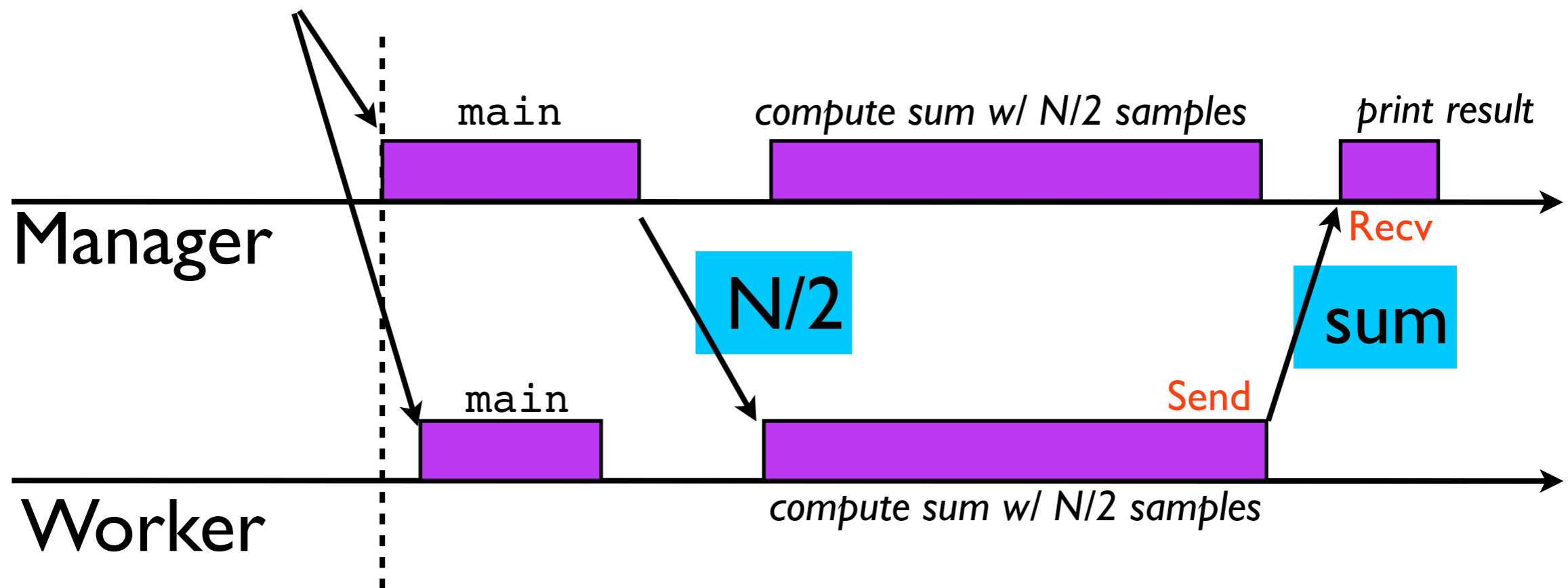


Computing Pi

(Parallel version in MPI)

- Manager/Worker setup

```
mpirun -np 2 ./pi2 N
```



Computing Pi

(Parallel version in MPI)

Main Function

```
int main(int argc, char *argv[]) {
    int N, myId, noProcs, nameLen, i;
    char procName[MPI_MAX_PROCESSOR_NAME];

    if ( argc<2 ) {
        printf( "Syntax: mpirun -np 2 pi2 N\n" );
        return 1;
    }
    N = atoi( argv[1] );

    //--- start MPI ---
    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &myId );
    MPI_Comm_size( MPI_COMM_WORLD, &noProcs );
    MPI_Get_processor_name( procName, &nameLen );
    printf( "Process %d of %d started on %s. N = %d\n",
           myId, noProcs, procName, N );
    //--- farm out the work: 1 manager, several workers ---
    if ( myId == MANAGER )
        doManager( N );
    else
        doWorker( );

    //--- close up MPI ---
    MPI_Finalize();
    return 0;
}
```

getcopy mpi/pi2.c



Computing Pi

(Parallel version in MPI)

Manager Function

```
//=== M A N A G E R ===  
void doManager( int n ) {  
    double sum0 = 0, sum1;  
    double deltaX = 1.0/n;  
    int i;  
    MPI_Status status;  
  
    //--- first, send n to worker ---  
    MPI_Send( &n, 1, MPI_INT, WORKER, 0, MPI_COMM_WORLD );  
  
    //--- perform 1st half of the work ---  
    for ( i=0; i< n/2; i++ )  
        sum0 += f( i * deltaX );  
  
    //--- wait for other half from worker ---  
    MPI_Recv( &sum1, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status );  
  
    //--- output result ---  
    printf( "%d iterations: Pi = %1.6f\n", n, ( sum0 + sum1 )*deltaX );  
}
```

Computing Pi

(Parallel version in MPI)

Worker Function

```
//=== W O R K E R ===
void doWorker( ) {
    int i, n;
    MPI_Status status;
    double sum = 0, deltaX;

    //--- get n from manager ---
    MPI_Recv( &n, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &status );

    //--- do (second) half of the work ---
    deltaX = 1.0/n;

    for ( i=n/2; i< n; i++ )
        sum += f( i * deltaX );

    //-- send result to manager ---
    MPI_Send( &sum, 1, MPI_DOUBLE, MANAGER, 0, MPI_COMM_WORLD );
}
```

Compile and Run

(on aurora)

```
mpicc -o pi2b pi2b.c
```

```
mpirun -np 2 ./pi2b 1000000
```

```
Process 0 of 2 started on MacDom2.local. N = 1000000
```

```
Process 1 of 2 started on MacDom2.local. N = 1000000
```

```
1000000 iterations: Pi = 3.141594
```

`getcopy mpi/pi2b.c`

We Stopped Here Last Time



Exercise

- Modify the MPI Pi computation program so that the Manager uses 2 Workers to do the computation. The Manager doesn't compute anything, except summing up the results. Worker 0 computes the sum from $[0, N/2)$, and Worker 1 computes the sum from $[N/2, N)$. Develop your code on your laptop or on aurora, for faster results.



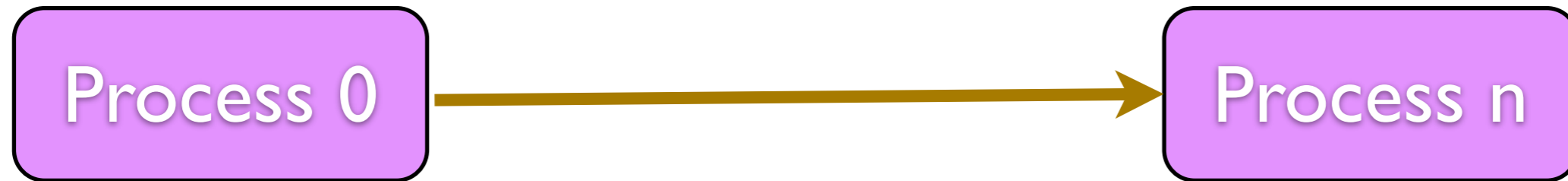
- Run your program for different values of N and compare its execution time to the execution time of the original version. Faster? Slower?

One-to-Many, Many-to-One Communication

Communication

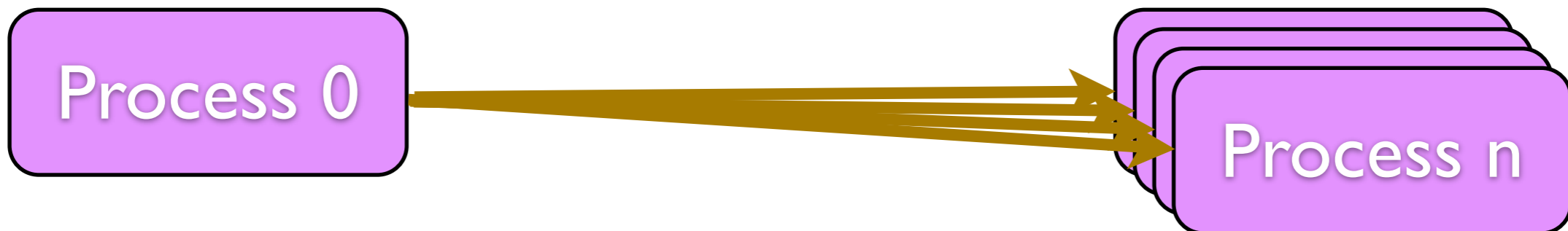
`MPI_Send(__, __, __, n, __, __)`

`MPI_Recv(__, __, __, 0, __, __, __)`



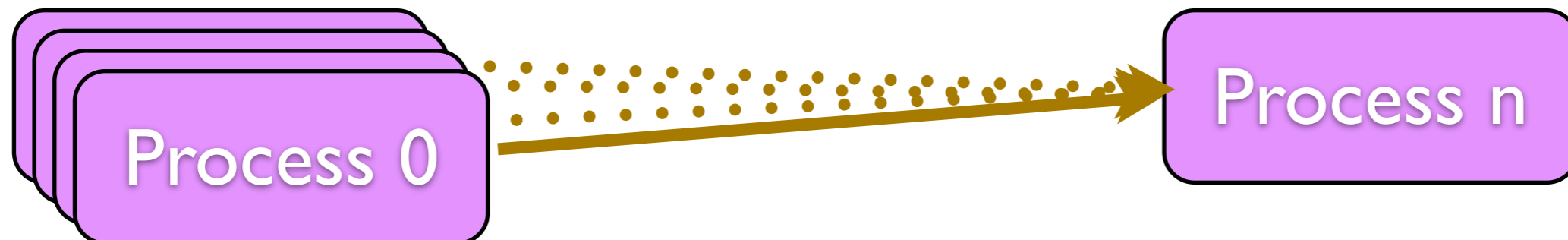
`MPI_Bcast(__, __, __, __, __)`

`MPI_Recv(__, __, __, 0, __, __, __)`



`MPI_Send(__, __, __, n, __, __)`

`MPI_Recv(__, __, __, MPI_ANY_SOURCE, __, __, __)`





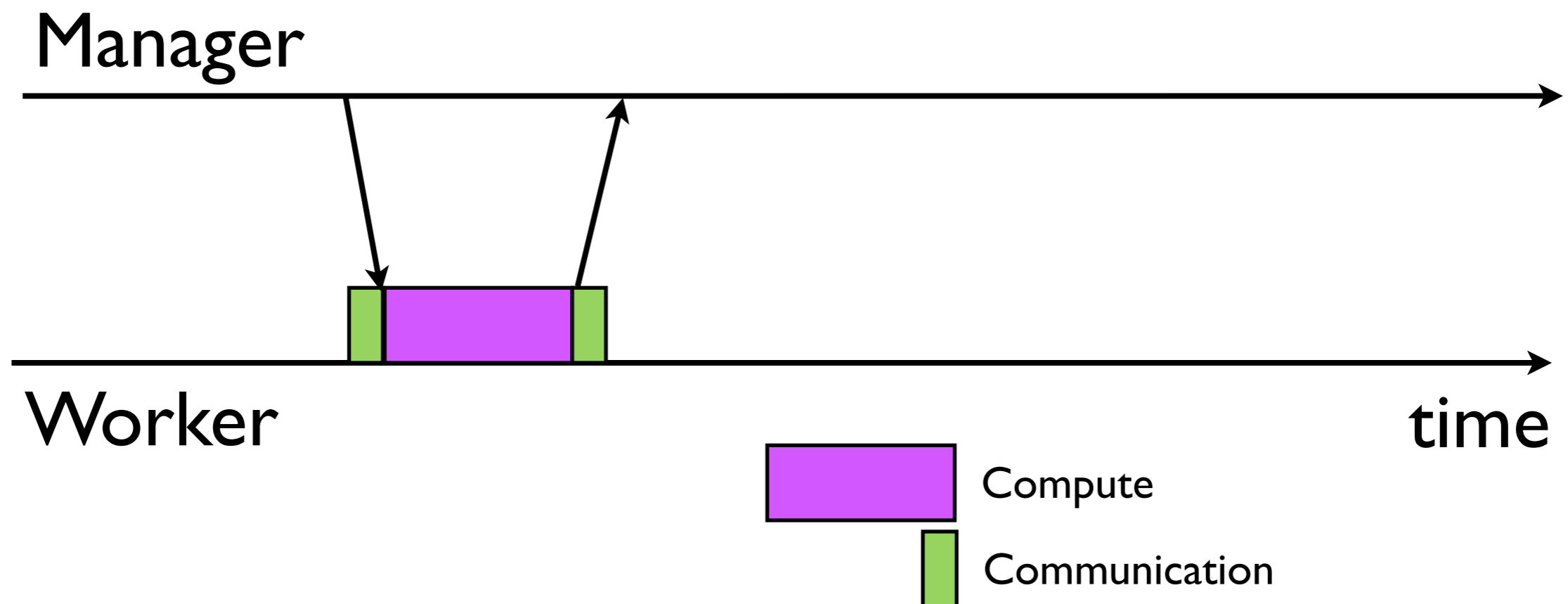
Exercise

- Run your version of the Pi program with 2, 10, 20 Processes:
 - On the hadoop cluster, using 4 processors.
 - On your laptop or Aurora
- Compare the execution times...

Scheduling/ Load-Balancing

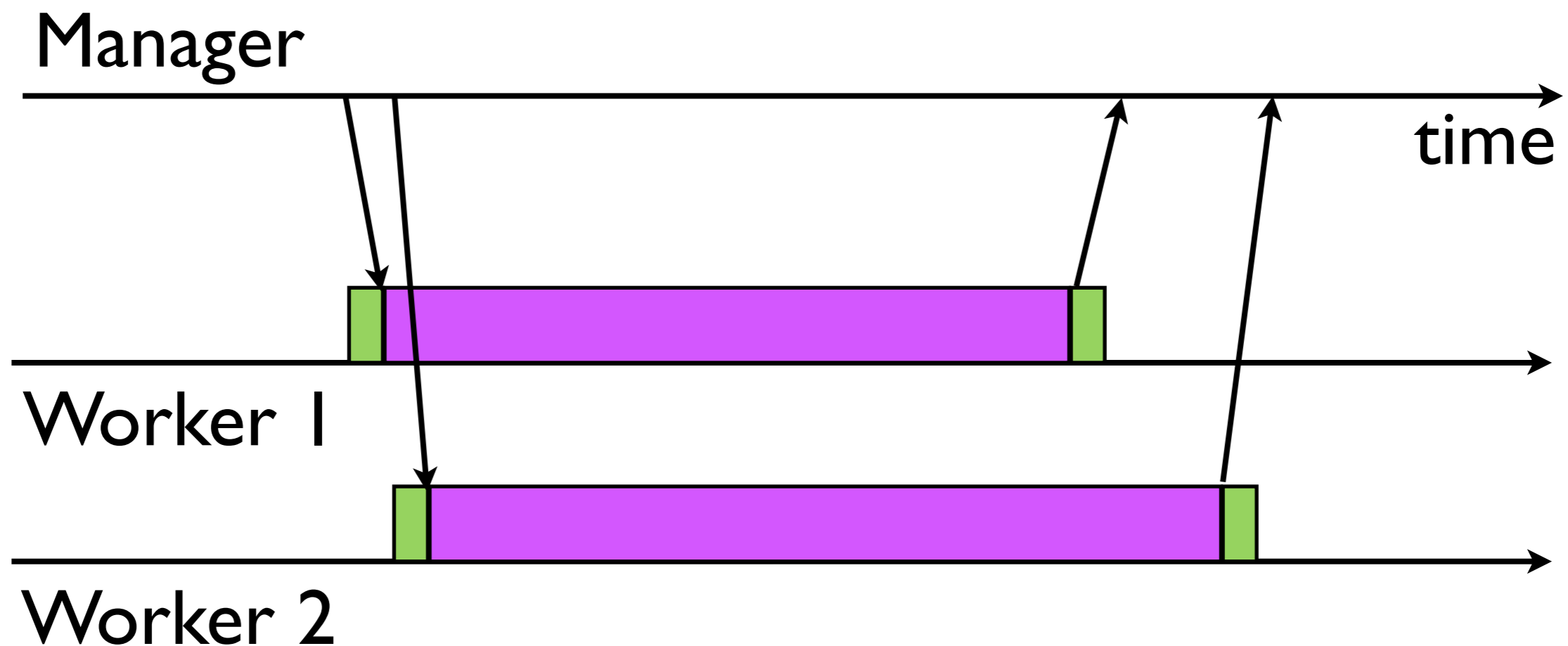
- Similar concepts
- Goal: Maximize performance by transferring tasks from busy to idle processors
- How: Determine parallel tasks + assign tasks to processors

Communication vs. Computation

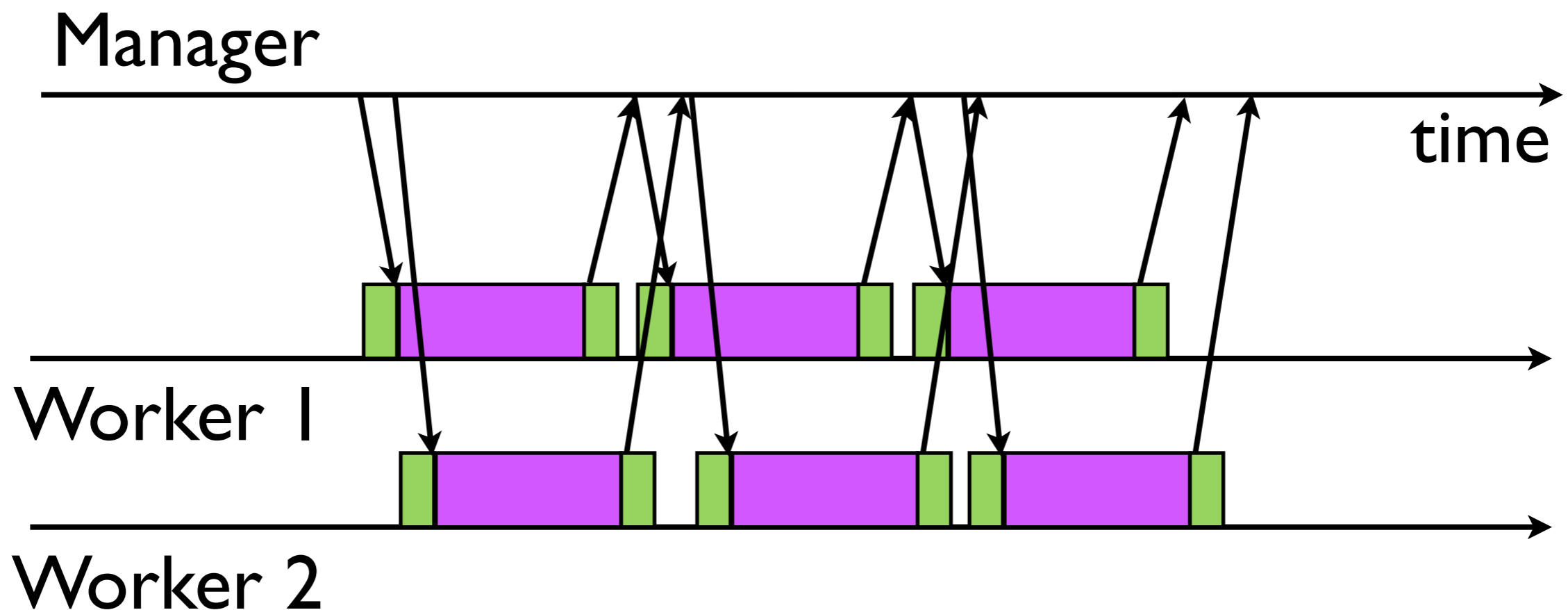


Communication vs. Computation

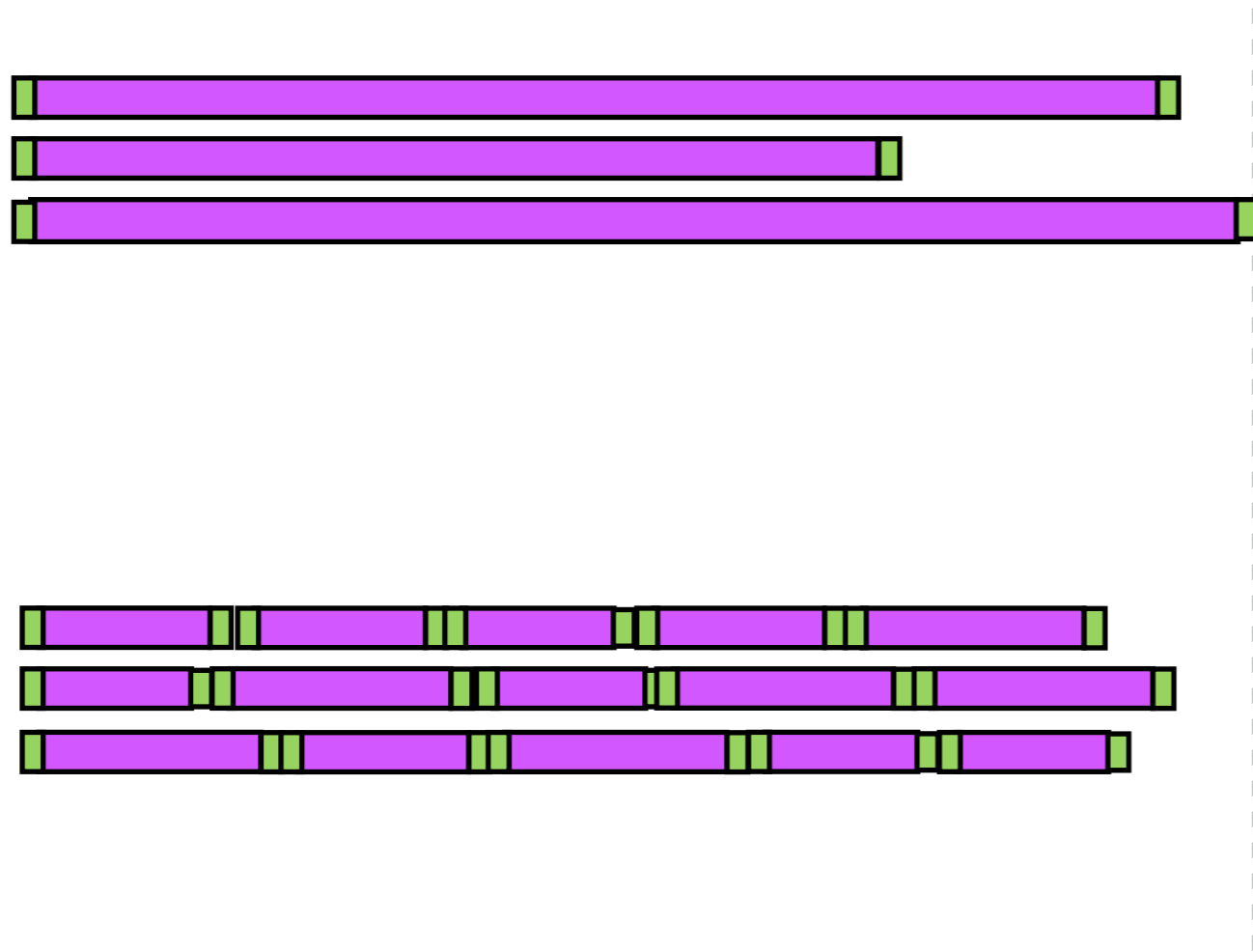
Coarse-grain parallelism



Communication vs. Computation



Communication vs. Computation



All-Important Graph: Communication vs Computation

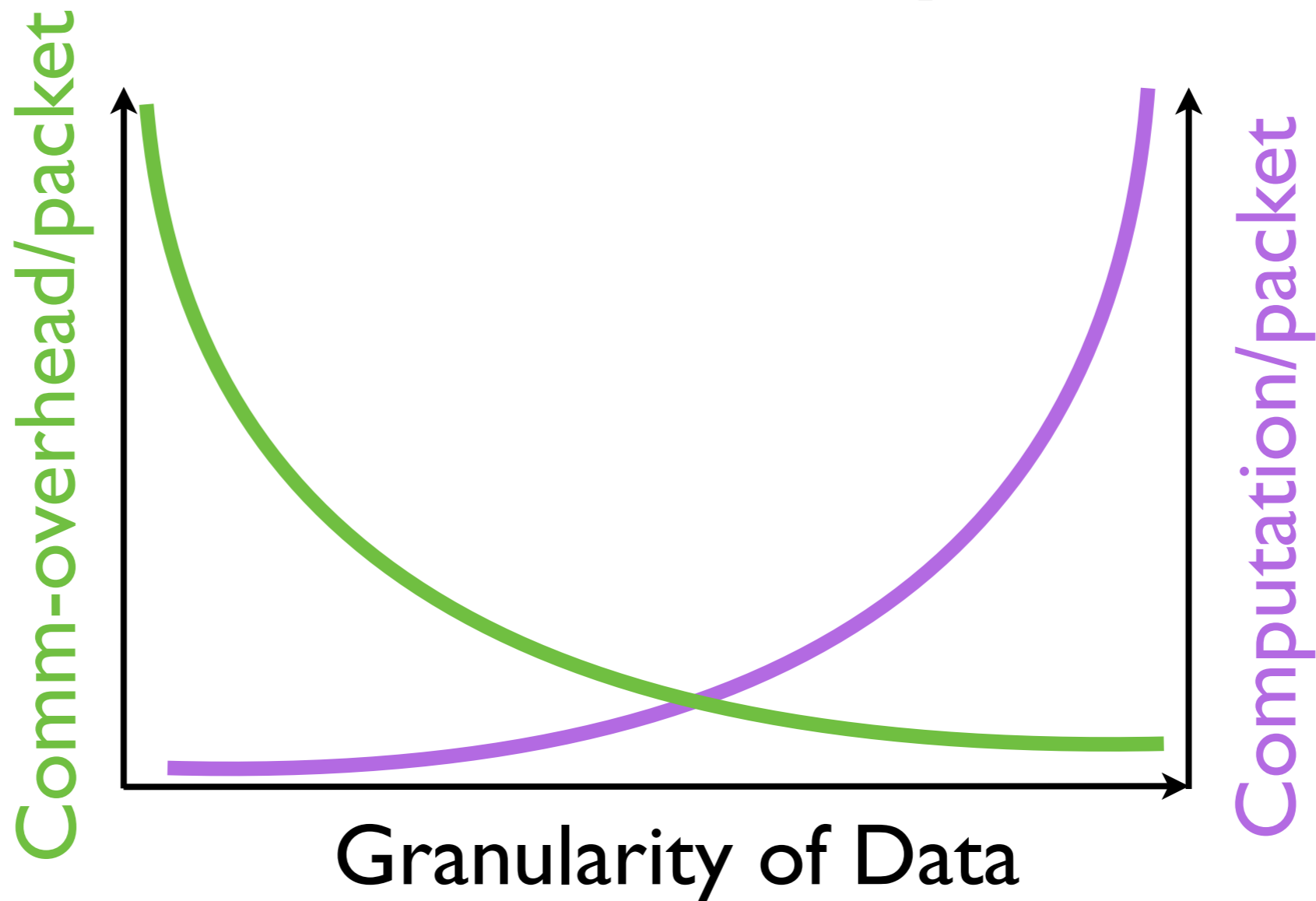
Fixed # Procs



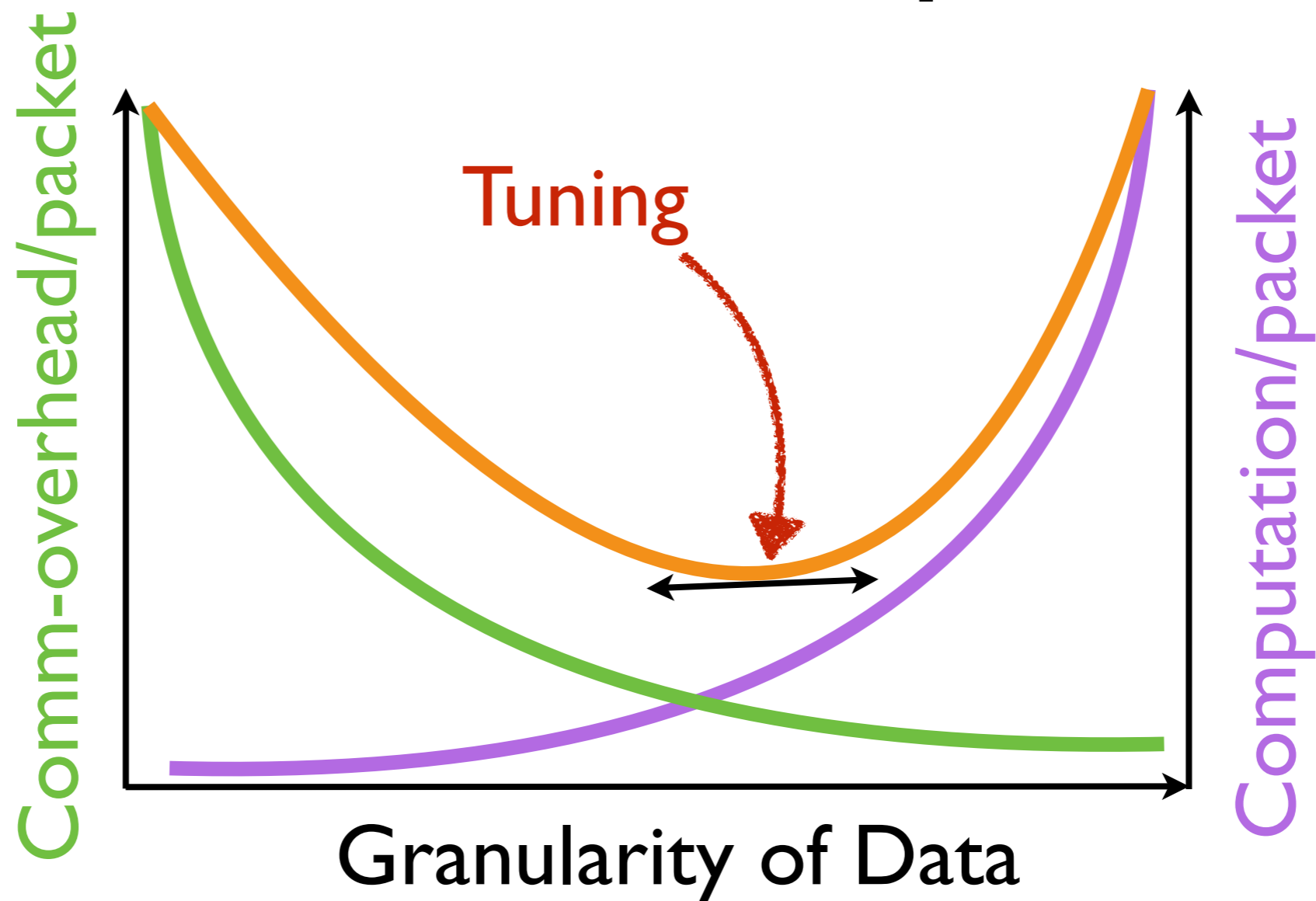
Granularity of Data



All-Important Graph: Communication vs Computation



All-Important Graph: Communication vs Computation



Scheduling & Load-Balancing w/ Manager & Workers

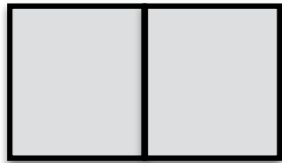
- Generate algorithm that solves the problem with **Manager** distributing work as N blocks to N **Workers**.
- Find solution for M blocks ($M > N$) and N workers. Worker asks for additional block when done.
- Pick M that yields best average performance.

Performance enhancing Technique: Double Buffering

- Each Worker maintains 2 buffers for 2 blocks of data
- Worker requests new block when done with first block, and directly moves on to 2nd buffered block.
- Result: **overlap** of CPU & Communication

Step 0

Manager



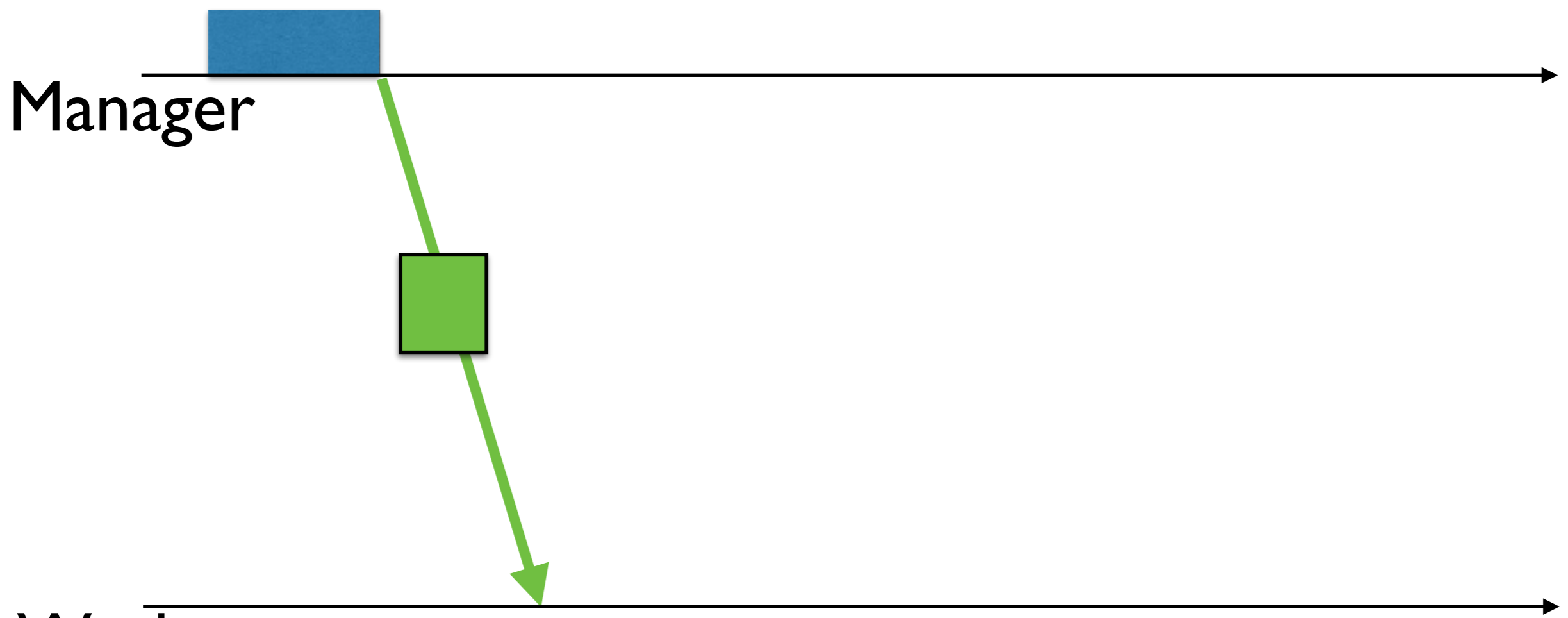
2 empty buffers



Worker



Step 1: Manager sends Block 0



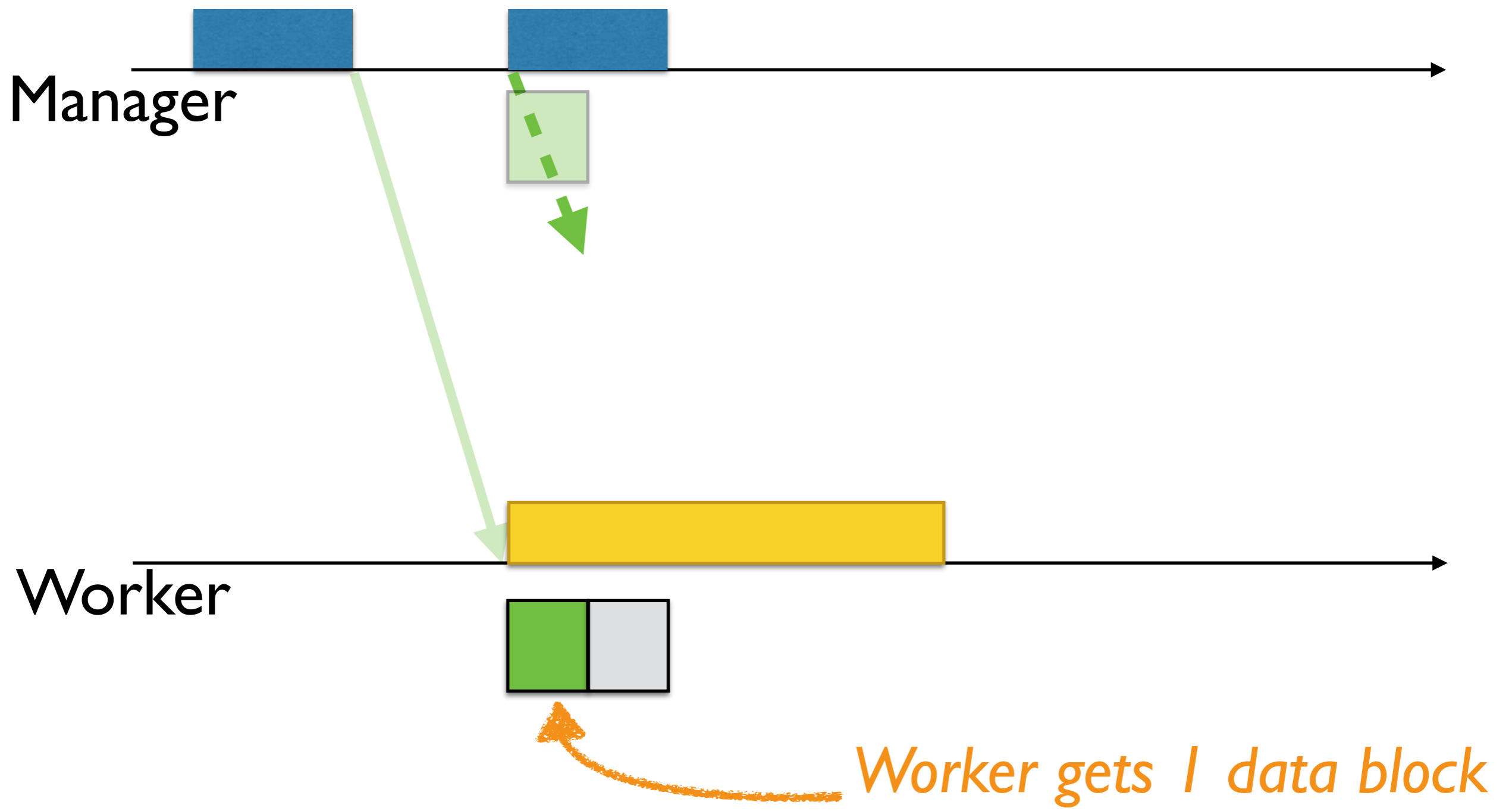
Manager

Worker

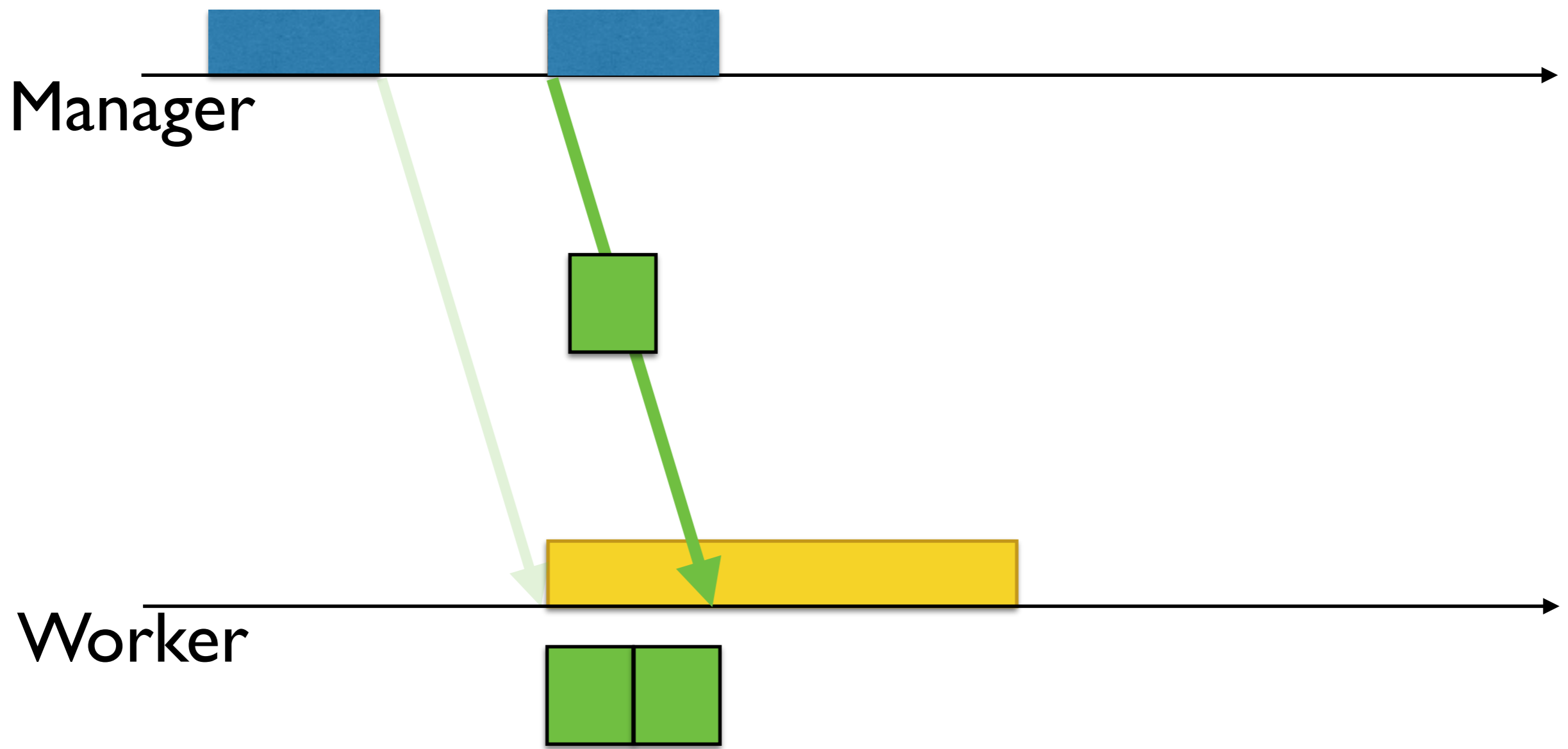


Worker gets 1 data block

Step 2: Worker starts working on Block 0
Manager starts sending Block 1

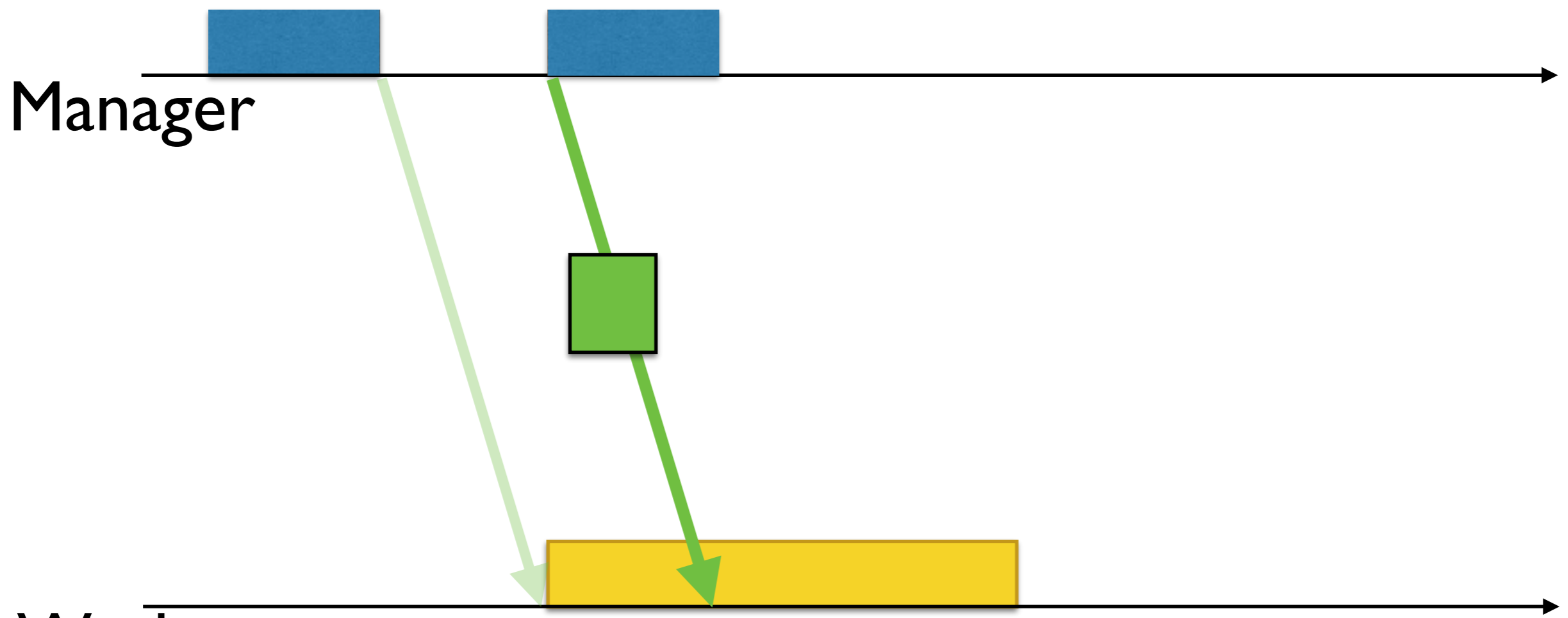


Step 2: Worker starts working on Block 0
Manager starts sending Block 1



Worker has 2 data blocks

Step 2: Worker starts working on Block 0
Manager starts sending Block 1

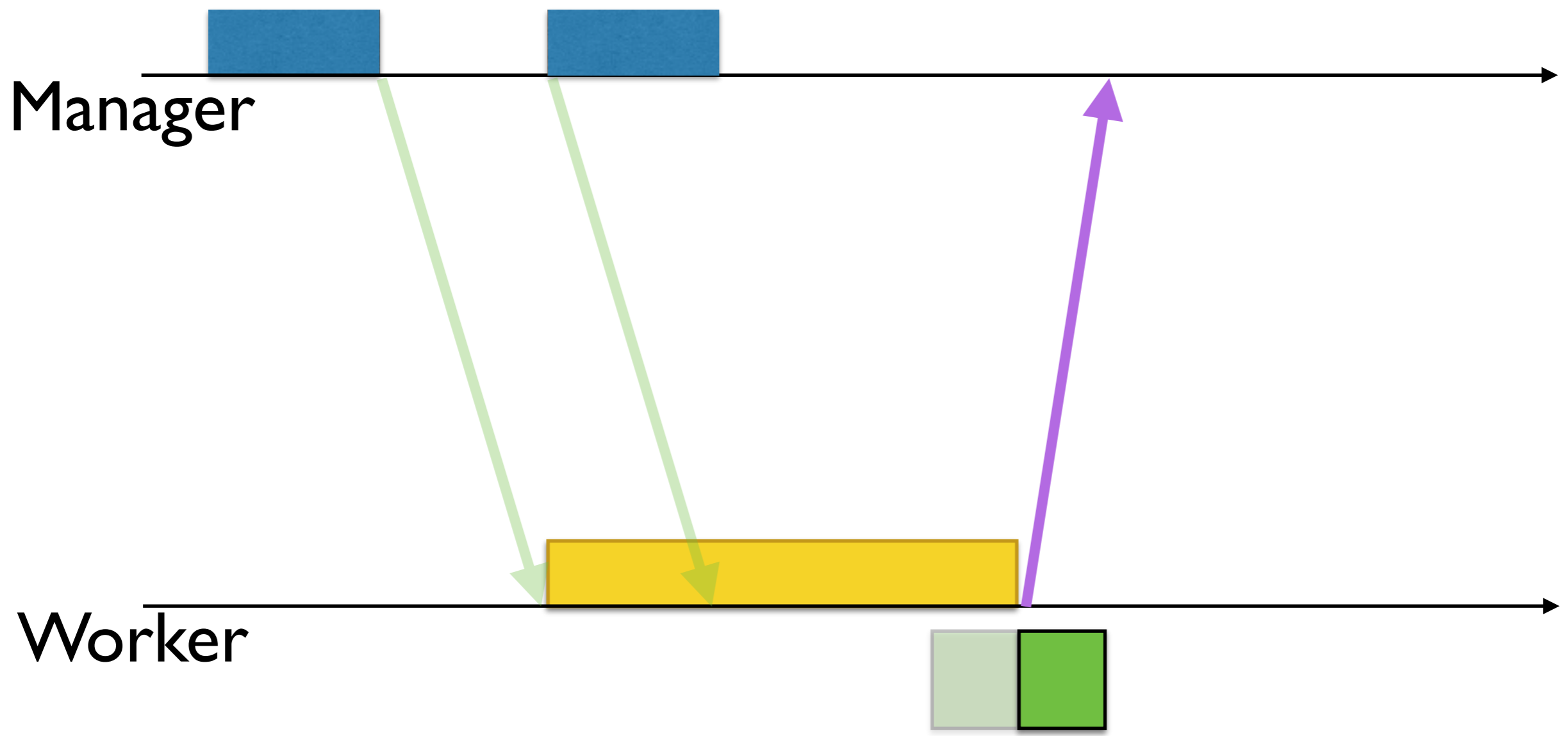


Manager

Worker

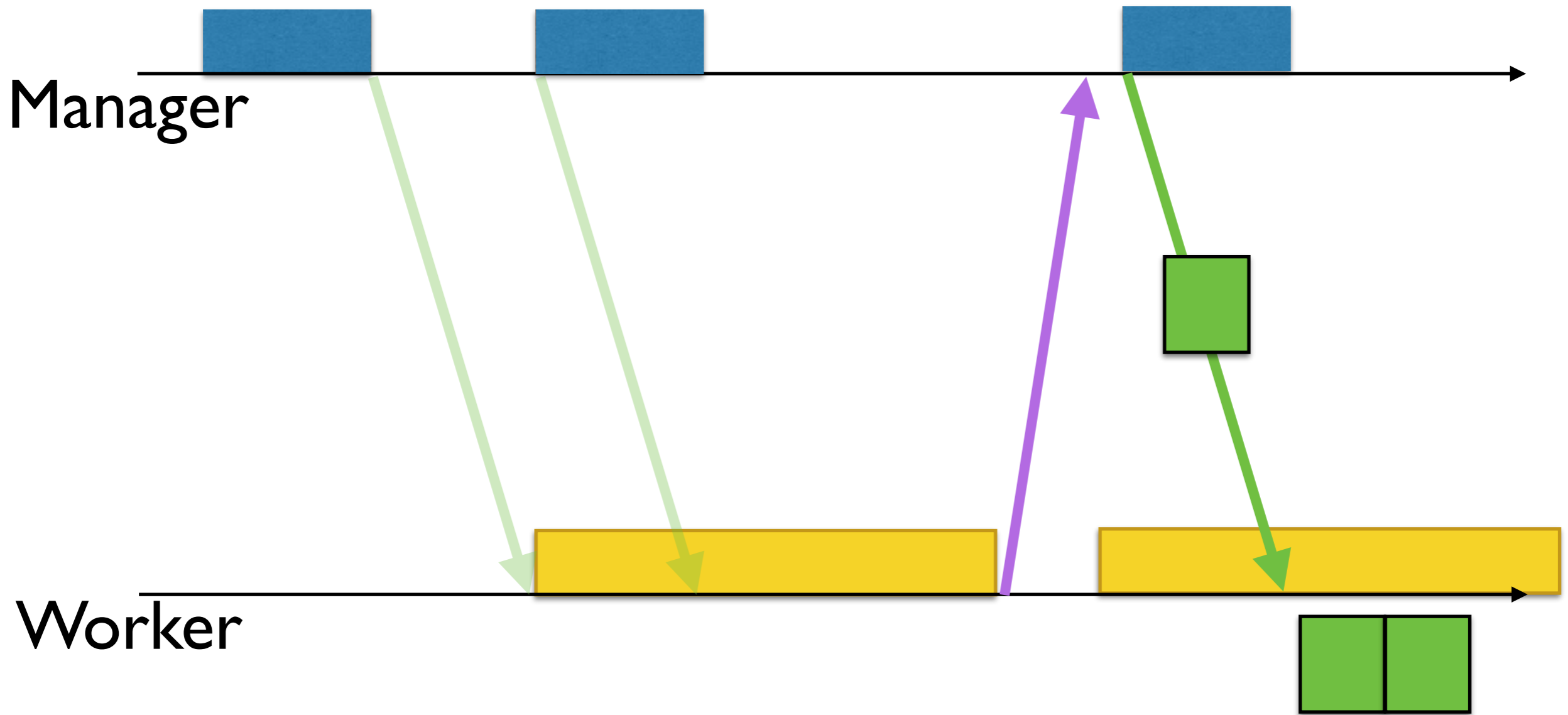
Worker has 2 data blocks

Step 3: Worker done with Block 0
Worker flags Manager



Worker done with first block

Step 4: Worker starts on Block 1
Manager sends Block 2



Worker got new block

Observations

- Double-Buffering requires **non-blocking communication**
- Double-Buffering can easily be generalized to **K-Buffering**, where worker keeps buffer of K blocks.

Exercise



- Modify the Pi-approximation program so that the manager distributes computation in several small chunks to its n workers.
- See if you can find a combination of n , the size and the chunk size that reduces the execution time.

Reference

- List of all MPI functions
<http://www.mcs.anl.gov/research/projects/multiplatform/mpi/mpi3/>



MPI on **AWS**

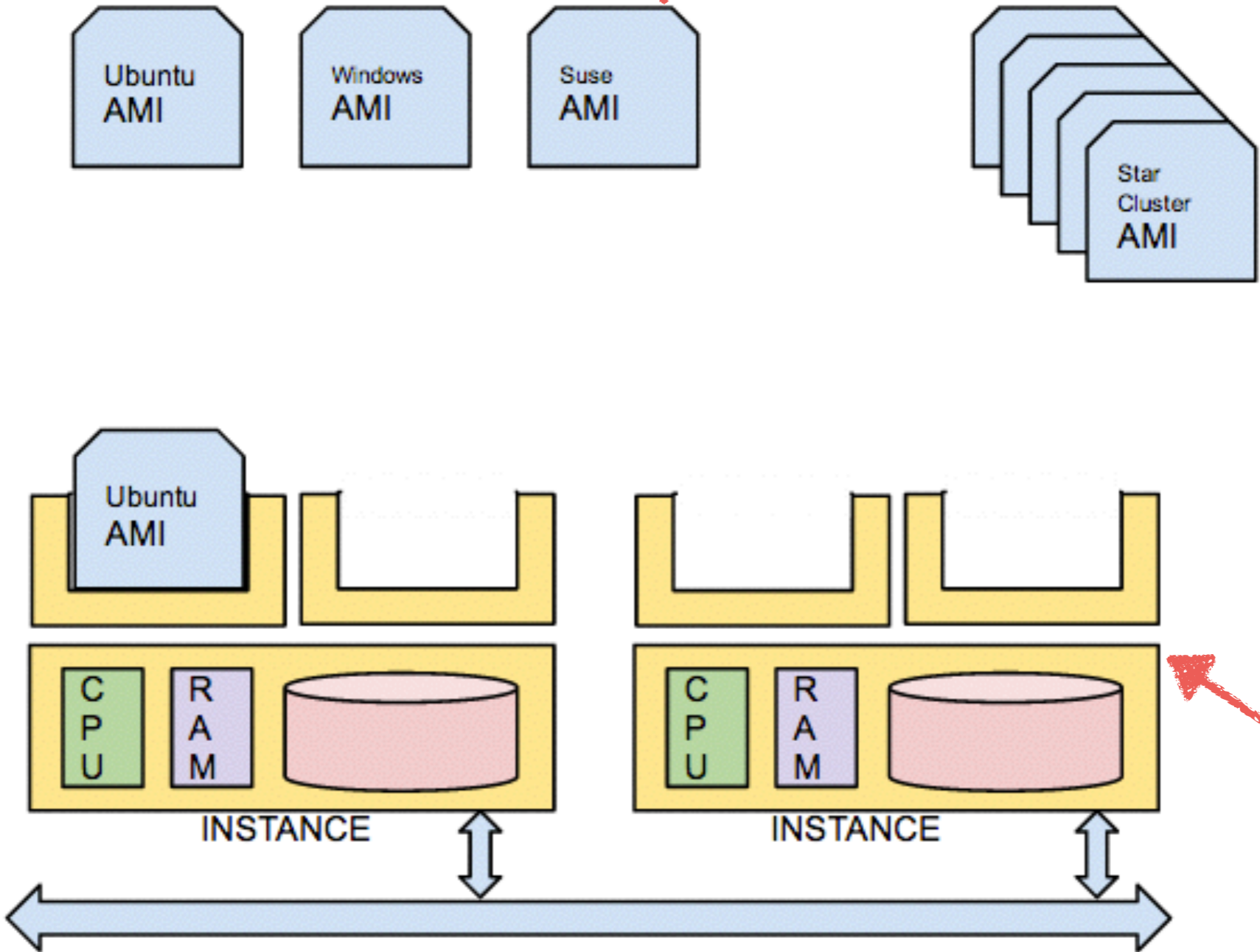
Installation and Tutorials

- Running MPI on AWS
- Tutorial 1 (Install StarCluster)
- Tutorial 2 (Compute Pi on AWS/MPI)

(See next slide...)

Amazon Machine Image

AWS



Instance

Some Additional Words on FPGA and Apache Spark

Introduction to Apache Spark

CSC352—Spring 2017

Dominique Thiébaud
dthiebaut@smith.edu

References

- Apache Spark: A Unified Engine for Big Data Processing, Zaharia Matei et al, *CACM*, Nov 2016, Vol. 59, No. 11.
[http://cs.smith.edu/dftwiki/images/3/3d/](http://cs.smith.edu/dftwiki/images/3/3d/ApacheSparkUnifiedEngine.pdf)
[ApacheSparkUnifiedEngine.pdf](http://cs.smith.edu/dftwiki/images/3/3d/ApacheSparkUnifiedEngine.pdf)



The image shows a screenshot of a Vimeo video player. The video frame displays a man with glasses speaking in front of a window. The video player interface includes a play button, a progress bar showing 01:19, and a volume icon. On the right side of the video frame, there are icons for heart, clock, list, and share. Below the video frame, the title "Apache Spark: A Unified Engine for Big Data Processing" is displayed in a large, bold font. Underneath the title, it says "from CACM PRO 6 months ago". To the right of the title, there is a section titled "More from CACM" with a toggle switch for "Autoplay next video" which is currently turned on. Below this section, there is a small thumbnail of another video.

<https://vimeo.com/185645796>

What is Verilog? How Do FPGAs Work?

CSC352 — Spring 2017

Dominique Thiébaud
dthiebaut@smith.edu

XiLinx CoolRunner Kit



References



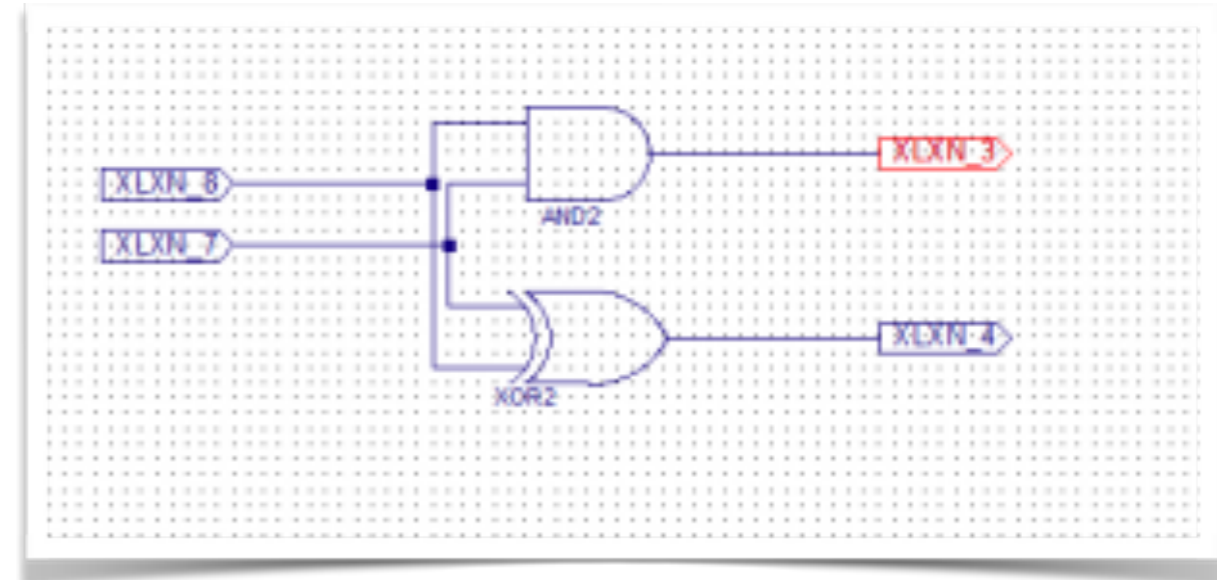
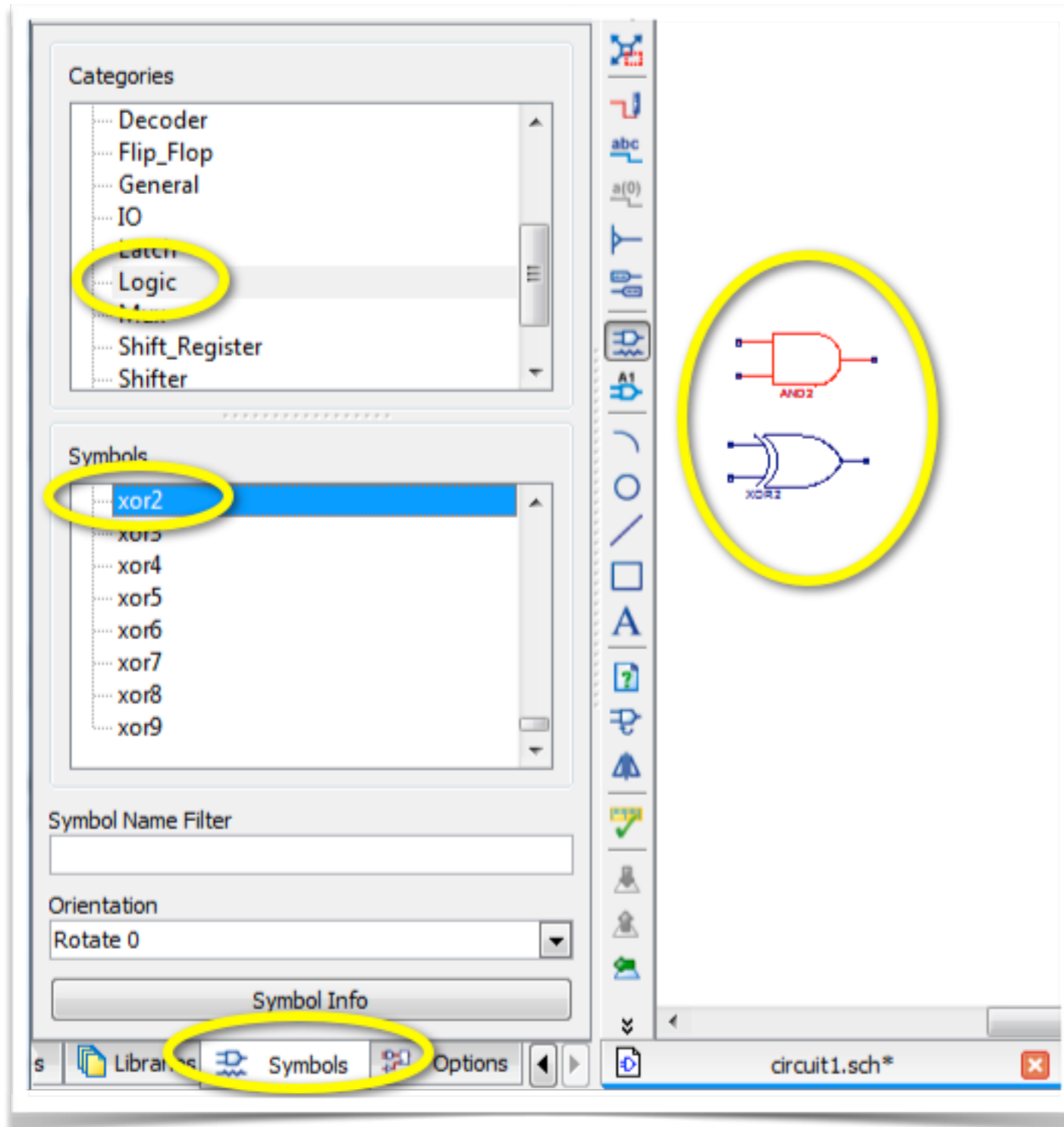
Tiffany Liu

- [Verilog/Xilinx Tutorial #1](#)
- [Verilog/Xilinx Tutorial #2](#)
- [Verilog/Xilinx 4-Bit Adder](#)
- [Sequential Circuit in Verilog](#)

Example 1: A 2-Bit Adder

2-Bit Adder

Step 1: Design



2-Bit Adder

Step 2: Verilog

```
//Design Name: circuit1
//Device: xbr
//Purpose:
// This verilog netlist is translated from an ECS schematic.It can be
// synthesized and simulated, but it should not be modified.
//
`timescale 1ns / 1ps

module circuit1(A,
               B,
               Carry,
               Sum);

    input A;
    input B;
    output Carry;
    output Sum;

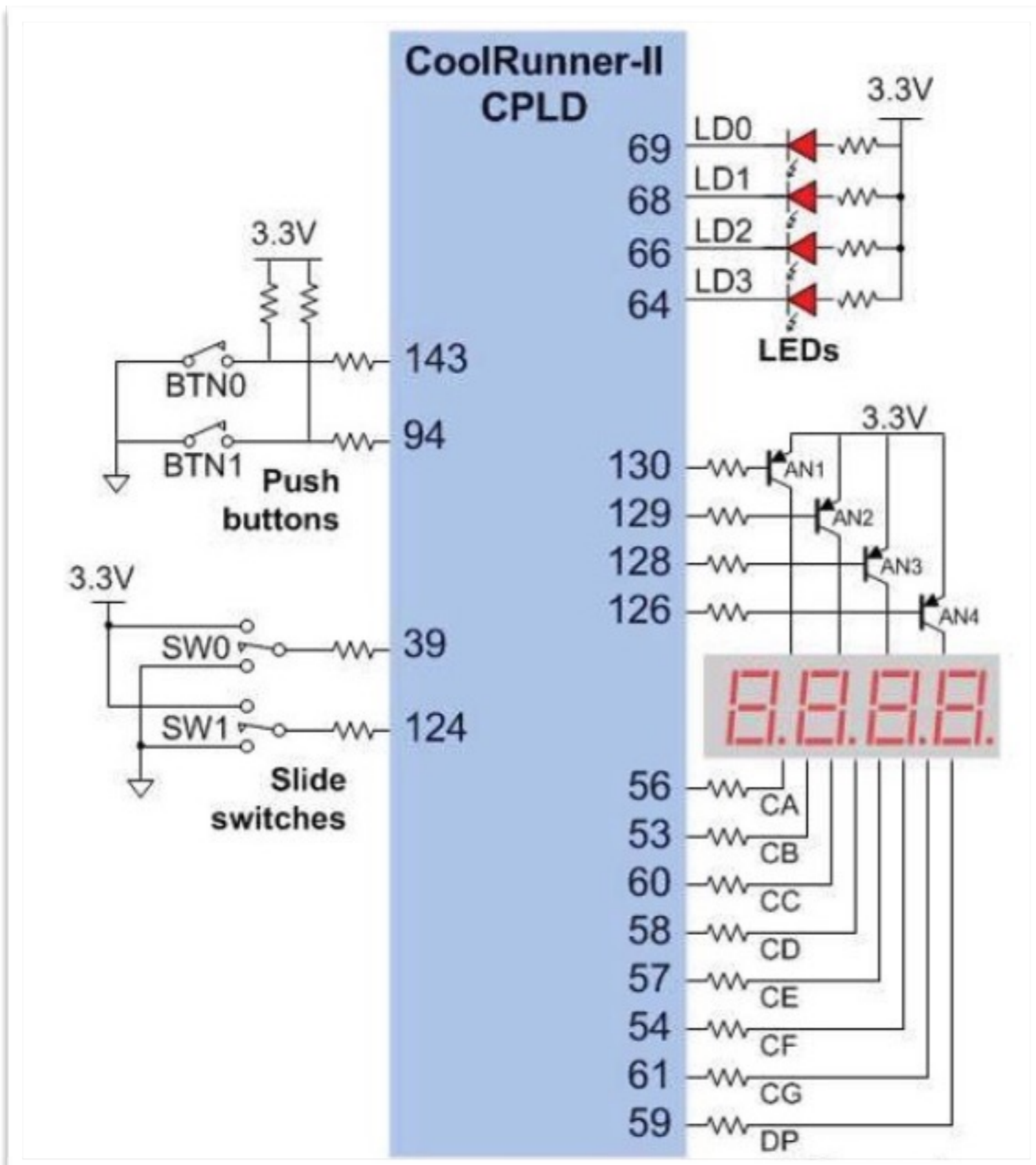
    AND2  XLXI_1 (.I0(B),
                .I1(A),
                .O(Carry));
    XOR2  XLXI_2 (.I0(A),
                .I1(B),
                .O(Sum));

endmodule

// Verilog instantiation template [...] - Tue Apr 24 16:12:42 2012
//
// Notes:
// To use this template to instantiate this component, cut-and-paste and then edit
//
// Instantiate the UUT
circuit1 UUT (
    .Carry(    ),
    .Sum(      ),
    .B(        ),
    .A(        )
);
```

2-Bit Adder

Step 3: I/O



(Hand Testing)

2-Bit Adder

Step 4: Simulation

```
// Verilog test fixture created from schematic [...] - Mon Apr 16 14:48:13 2012
`timescale 1ns / 1ps
module circuit1_circuit1_sch_tb();

// Inputs
reg B;
reg A;

// Output
wire Carry;
wire Sum;

// Bidirs

// Instantiate the UUT
circuit1 UUT (
    .Carry(Carry),
    .Sum(Sum),
    .B(B),
    .A(A)
);

// Initialize Inputs

    initial begin
        B = 0;
        A = 0;

        // wait 100 ns
        #100;

        // after 10 ns, set B to 1
        #10 B = 1;

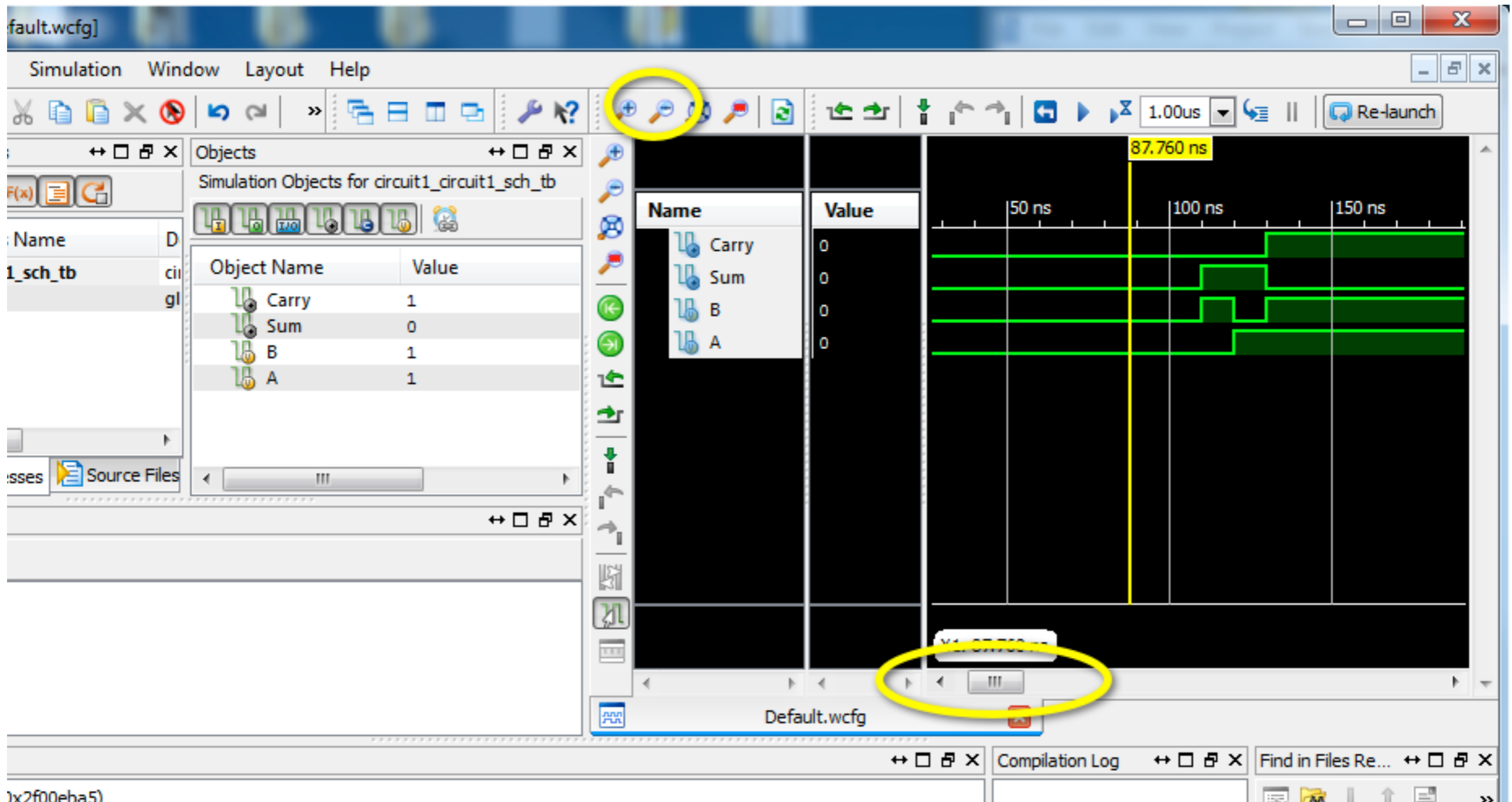
        // after 10 ns, set A to 1, reset B to 0
        #10 A = 1;
            B = 0;

        // after 10 ns, set B to 1
        #10 B = 1;
    end

endmodule
```

2-Bit Adder

Step 4: Simulation Output



Example 2:
Green-Yellow-Red
Sequencer

GYR Sequencer

Step 2: Tesing

```
// Verilog test fixture [...] - Tue Apr 24 15:25:45 2012
`timescale 1ns / 1ps
module Sequencer_Sequencer_sch_tb();

// Inputs
reg Cmd;
reg Clock;

// Output
wire Y;
wire R;
wire G;

// vars
integer i;

// Instantiate the UUT
Sequencer UUT (
    .Cmd(Cmd),
    .Y(Y),
    .R(R),
    .Clock(Clock),
    .G(G)
);

// setup the clock to switch every 10 ns
initial begin
    Cmd = 0;
    Clock = 0;
end

initial begin
    // wait 100 ns
    #100

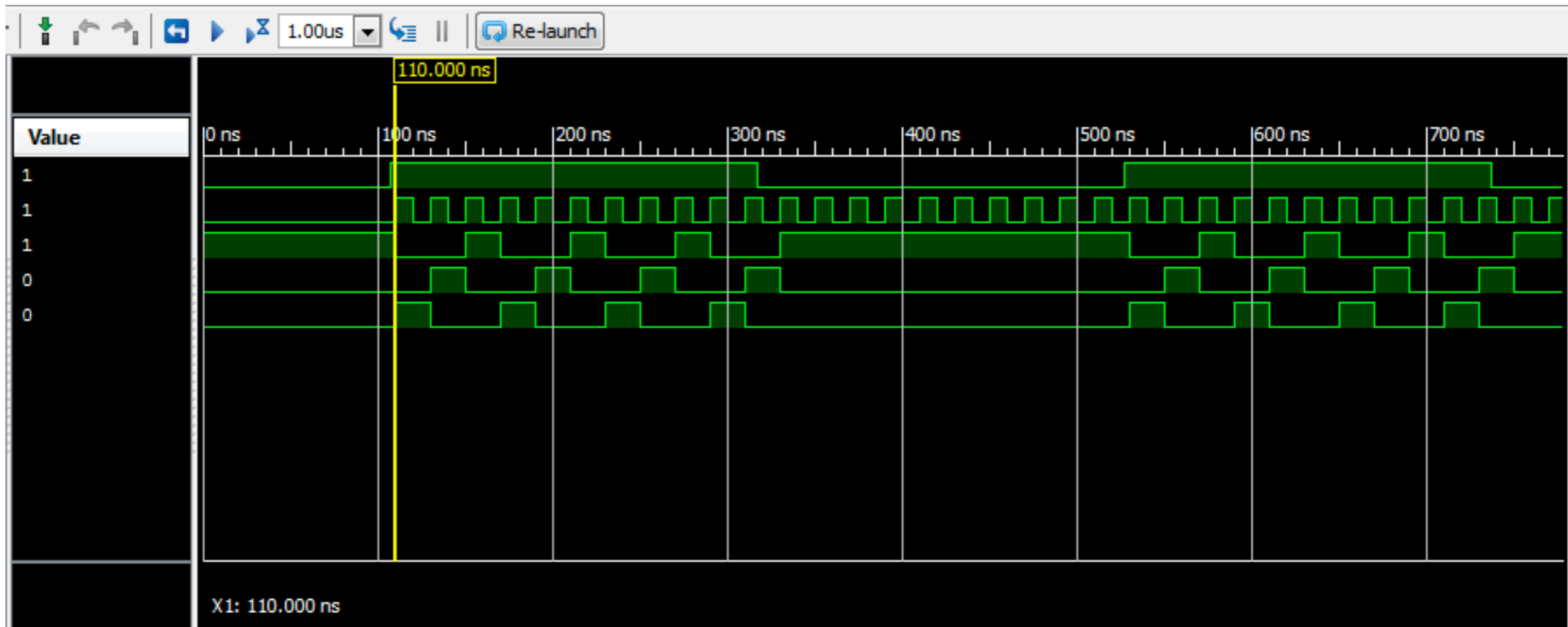
    // flip clock every 10 ns
    forever begin
        #10 Clock = ~Clock;
    end
end

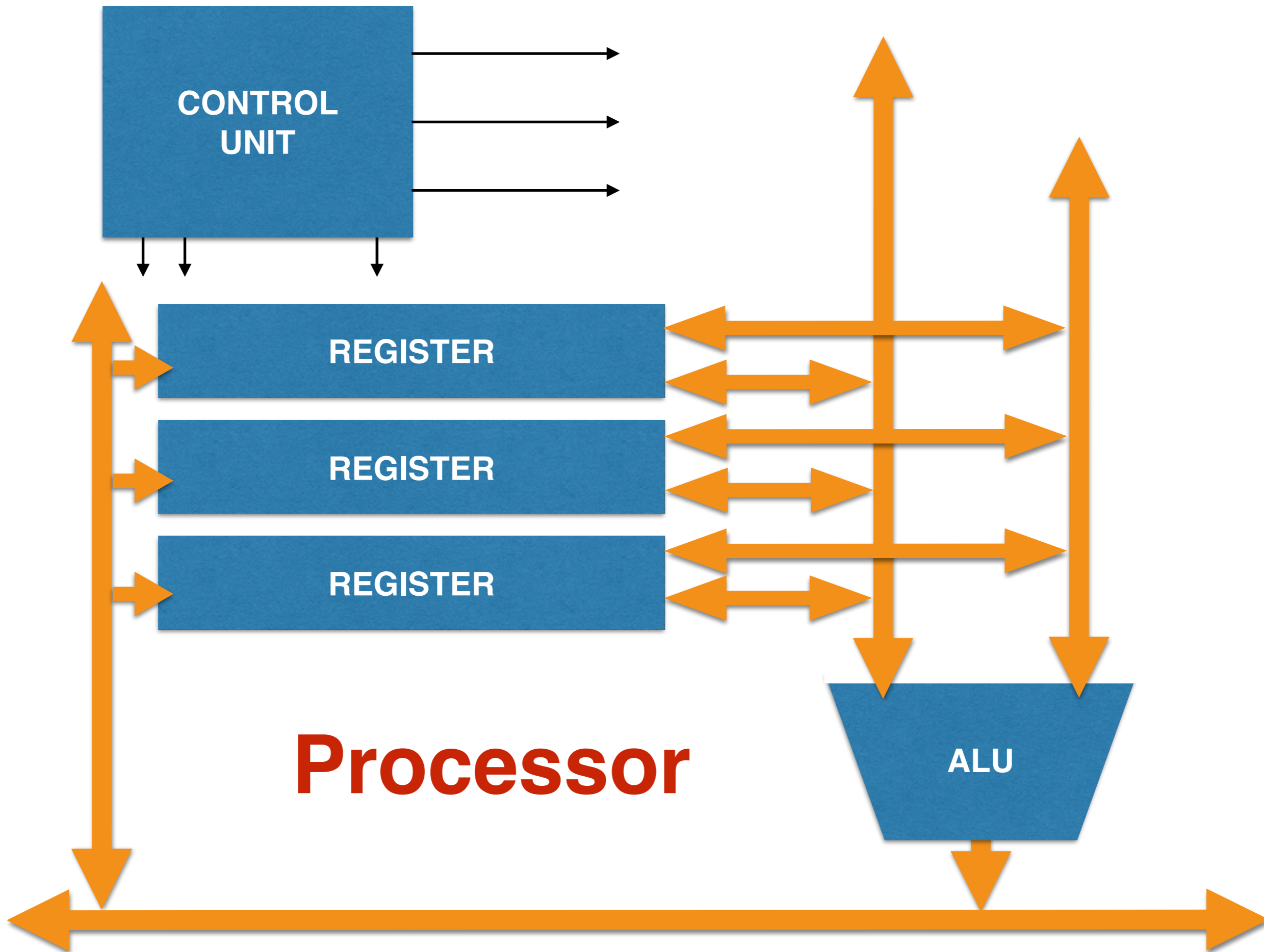
// activate the cmd signal regularly
always @ ( Cmd or Clock )
    #100
    for ( i = 0; i < 200; i = i+1 )
        begin
            #7 if ( i % 30 == 0 ) Cmd = ~Cmd;
        end
end

endmodule
```


GYR Sequencer

Step 3: Simulation





Amazon Into FPGAs as Well!



The image shows a presentation slide for Amazon F1 Instances. On the left, there is a circular graphic containing a red square with a white clock icon, surrounded by white vertical bars representing a chip. A red starburst with the word "NEW!" is positioned above the graphic. Below the graphic, the text "Preview Available Today" is displayed. To the right of the graphic, the title "F1 Instances" is written in a large, bold, white font. Below the title, the text "New Instance Family With Customizable Field Programmable Gate Arrays" is written in a smaller white font. At the bottom of this section, the text "Run Your Custom Logic On EC2" is displayed in a bold white font. A person in a dark suit and blue shirt is standing on a stage in front of the slide.

NEW!

F1 Instances

New Instance Family With Customizable
Field Programmable Gate Arrays

Run Your Custom Logic On EC2

Preview Available Today

New Homework Assignment

http://cs.smith.edu/dftwiki/index.php/CSC352_Homework_3_2017