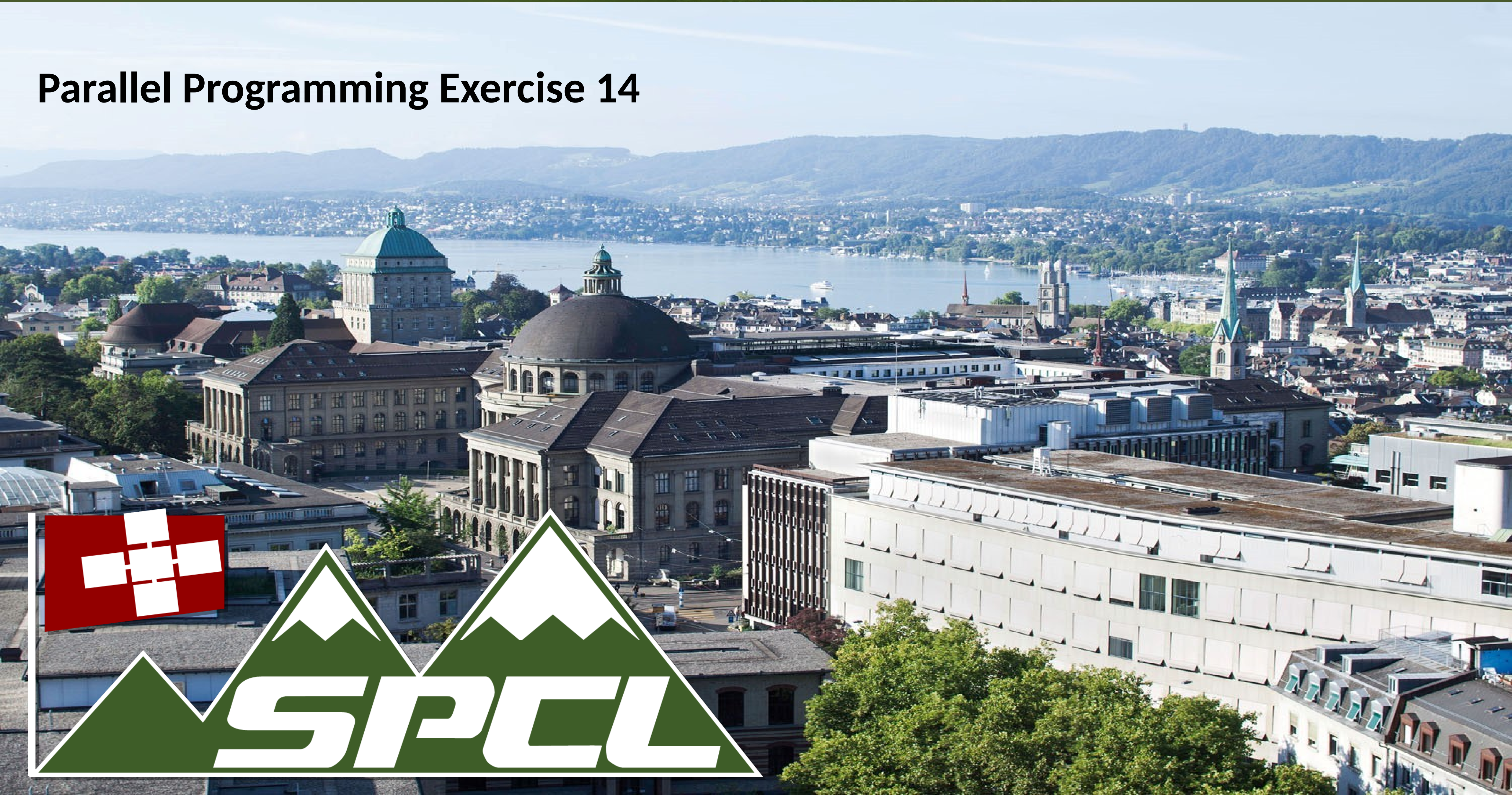
Parallel Programming Exercise 14



Feedback from Assignment 13

Feedback from Assignment 13

- **isFull and isEmpty – also STM!**

```
public boolean isEmpty() {
        return STM.atomic(new Callable<Boolean>() {
            @Override
            public Boolean call() {
                return count.get() == 0;
            }
        });
    }

    public boolean isFull() {
        return STM.atomic(new Callable<Boolean>() {
            @Override
            public Boolean call() {
                return count.get() == items.length();
            }
        });
    }
```

Feedback from Assignment 13

- **If vs while for STM.retry()**

```
public void put(final E item) {
        STM.atomic(new Runnable() {
            @Override
            public void run() {
                if (isFull())
                    STM.retry();
                items.update(putIndex.get(), item);
                putIndex.set(next(putIndex.get()));
                STM.increment(count, 1);
            }
        });
    }
```

Feedback from Assignment 13

- If and else

```
public E take() {
 return STM.atomic(new Callable<E>() {
   @Override
   public E call() {
     if (isEmpty())
       STM.retry();
     E item =
items.refViews().apply(takeIndex.get()).get();
     items.update(takeIndex.get(), null);
     takeIndex.set(next(takeIndex.get()));
     STM.increment(count, -1);
     return item;
   }
 });
}
```

```
public E take() {
 return STM.atomic(new Callable<E>() {
   @Override
   public E call() {
     if (isEmpty())
       STM.retry();
     else {
       E item =
  items.refViews().apply(takeIndex.get()).get();
       items.update(takeIndex.get(), null);
       takeIndex.set(next(takeIndex.get()));
       STM.increment(count, -1);
       return item;
  }
   }
 });
```

Feedback from Assignment 13

- If and else

```
public E take() {
  return STM.atomic(new Callable<E>() {
      @Override
      public E call() {
          if (isEmpty())
              STM.retry();
          E item =
items.refViews().apply(takeIndex.get()).get();
          items.update(takeIndex.get(), null);
          takeIndex.set(next(takeIndex.get()));
          STM.increment(count, -1);
          return item;
      }
  });
}
```

```
public E take() {
  return STM.atomic(new Callable<E>() {
      @Override
      public E call() {
          if (isEmpty())
              STM.retry();
          else {
              E item =
    items.refViews().apply(takeIndex.get()).get();
              items.update(takeIndex.get(), null);
              takeIndex.set(next(takeIndex.get()));
              STM.increment(count, -1);
              return item;
    }
      }
  });
```

Lecture Recap: MPI

- **Maybe the most “relevant” part of the lecture if you do scientific computing**
- **The MPI Standard contains hundreds of functions, to use MPI you need to understand six of them**
- **We will use the C API when we talk about concepts**
  - since this is what you find in the MPI Standard and most other documentation
  - code examples will be in Java

Six-Function MPI

- MPI_Init() <- Call this before any other MPI function
- MPI_Finalize() <- Call this when you are done

- MPI_Send() <- Send a message to another process (blocking)
- MPI_Recv() <- Recv a message from another process (blocking)

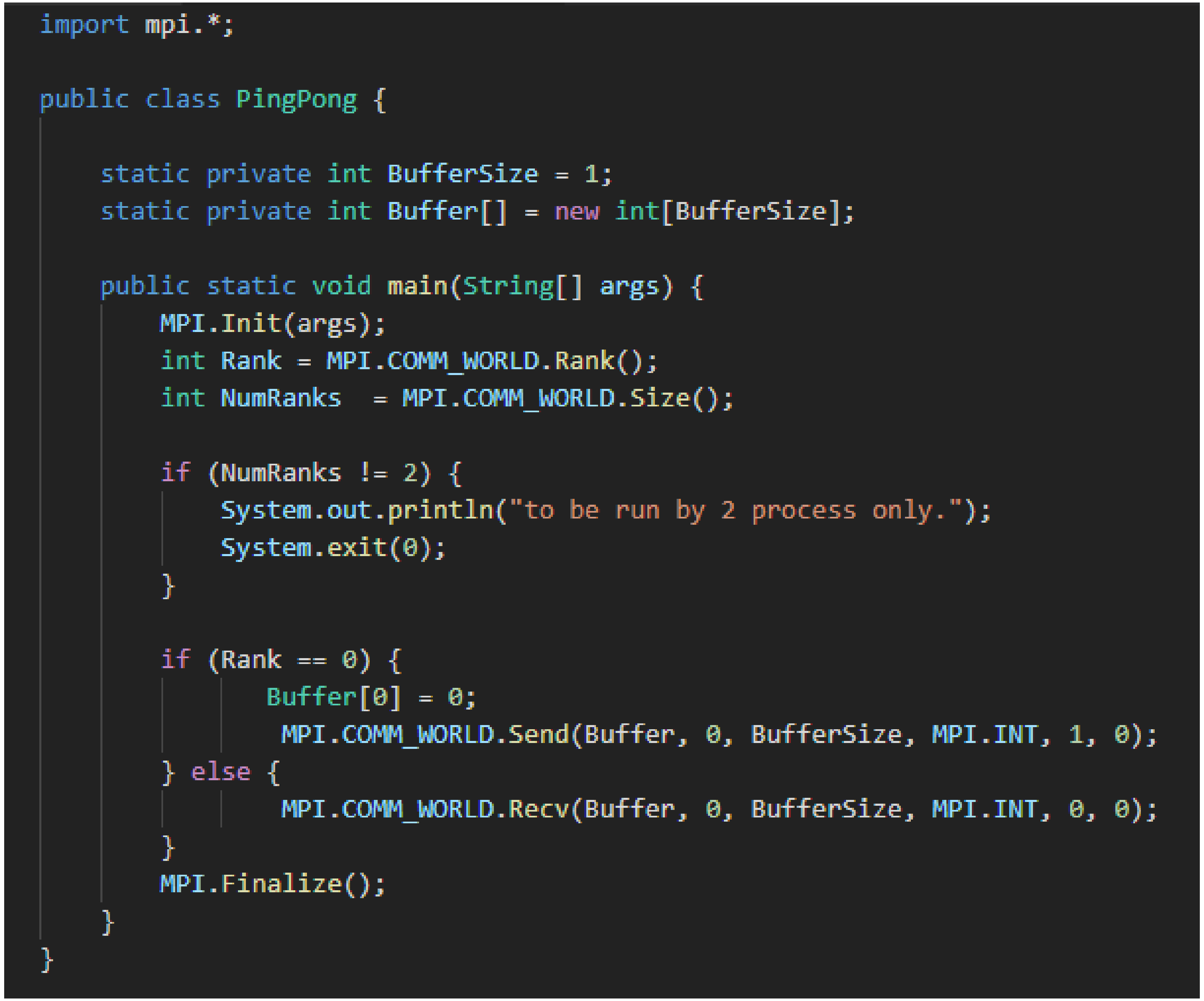
- MPI_Comm_rank() <- What is my ID in a communicator (i.e., MPI_COMM_WORLD)
- MPI_Comm_size() <- How many processes are in a communicator

Six-Function MPI in Java with MPJ

- Can be done in Eclipse directly (see exercise)
- Can be done on the command line (important for remote work on supercomputers)

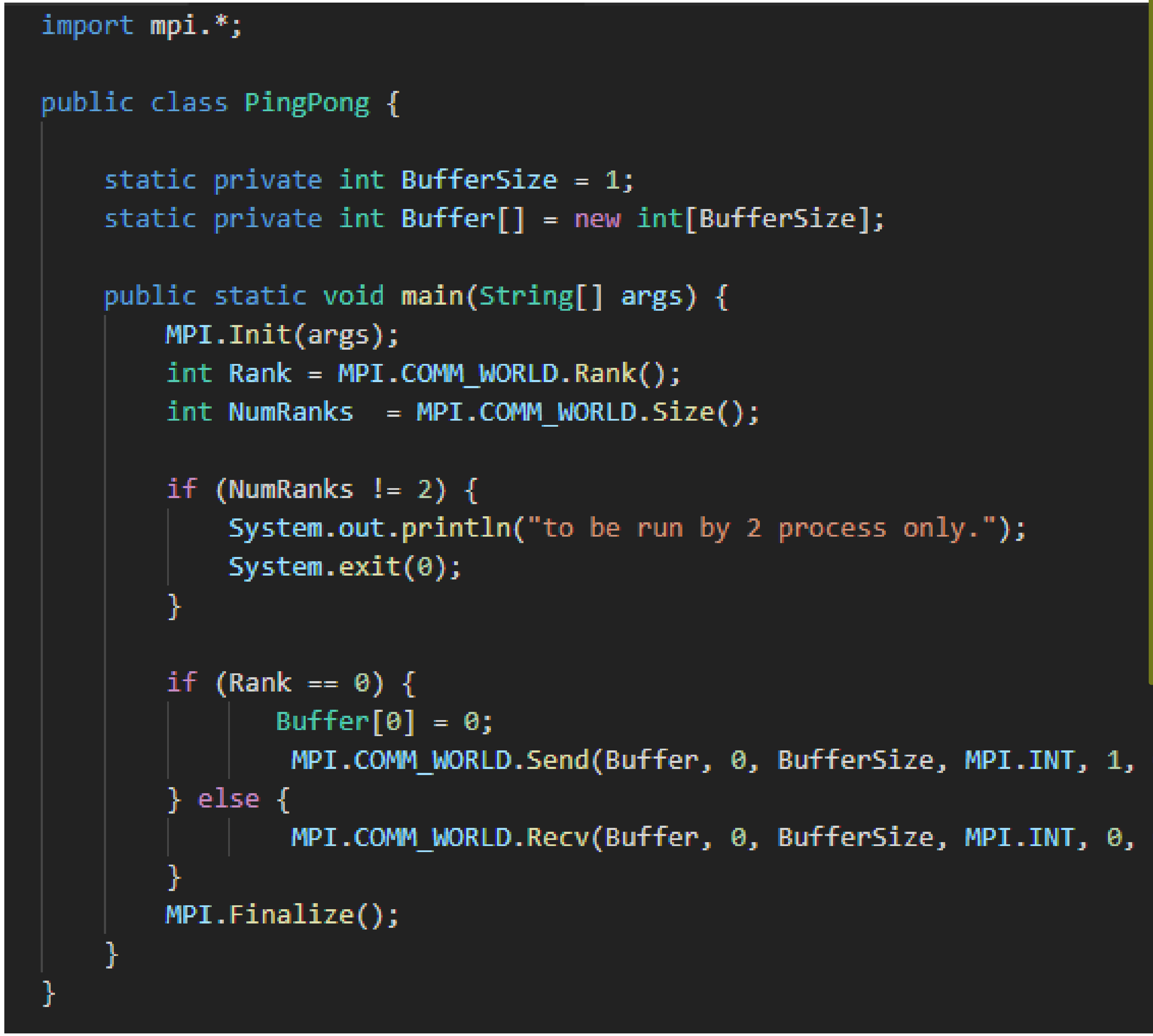
- Download MPJ and unpack it
- export MPJ_HOME=/home/youruser/path/to/mpj
- export PATH=$MPJ_HOME/bin:$PATH
- javac -cp .:MPJ_HOME/lib/mpj.jar YourCode.java
- mpjrun.sh -np 2 YourCode

Six-Function MPI in Java with MPJ

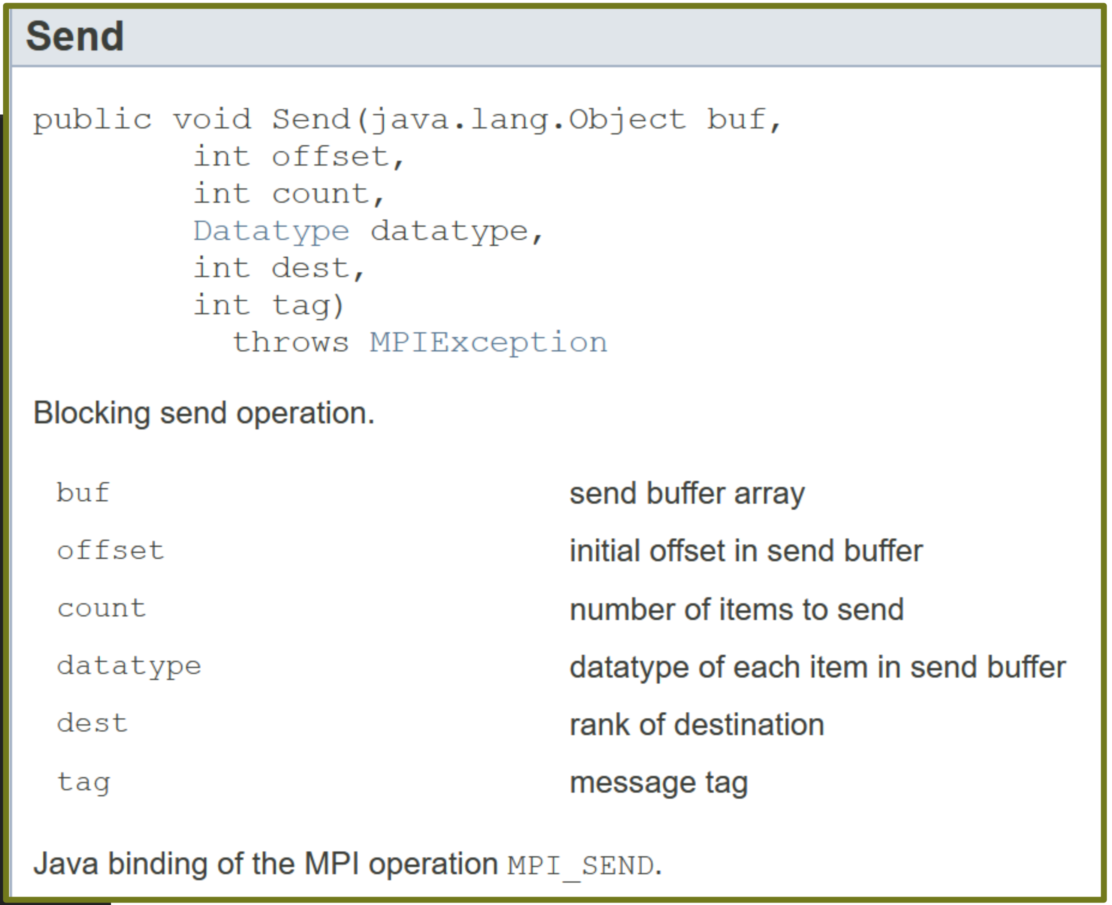


```java
import mpi.*;

public class PingPong {

    static private int BufferSize = 1;
    static private int Buffer[] = new int[BufferSize];

    public static void main(String[] args) {
        MPI.Init(args);
        int Rank = MPI.COMM_WORLD.Rank();
        int NumRanks  = MPI.COMM_WORLD.Size();

        if (NumRanks != 2) {
            System.out.println("to be run by 2 process only.");
            System.exit(0);
        }

        if (Rank == 0) {
               Buffer[0] = 0;
                MPI.COMM_WORLD.Send(Buffer, 0, BufferSize, MPI.INT, 1, 0);
        } else {
                MPI.COMM_WORLD.Recv(Buffer, 0, BufferSize, MPI.INT, 0, 0);
        }
        MPI.Finalize();
    }
}
```

Six-Function MPI in Java with MPJ



```
import mpi.*;

public class PingPong {

    static private int BufferSize = 1;
    static private int Buffer[] = new int[BufferSize];

    public static void main(String[] args) {
        MPI.Init(args);
        int Rank = MPI.COMM_WORLD.Rank();
        int NumRanks  = MPI.COMM_WORLD.Size();

        if (NumRanks != 2) {
            System.out.println("to be run by 2 process only.");
            System.exit(0);
        }

        if (Rank == 0) {
                Buffer[0] = 0;
                 MPI.COMM_WORLD.Send(Buffer, 0, BufferSize, MPI.INT, 1, 0);
        } else {
                 MPI.COMM_WORLD.Recv(Buffer, 0, BufferSize, MPI.INT, 0, 0);
        }
        MPI.Finalize();
    }
}
```



Send

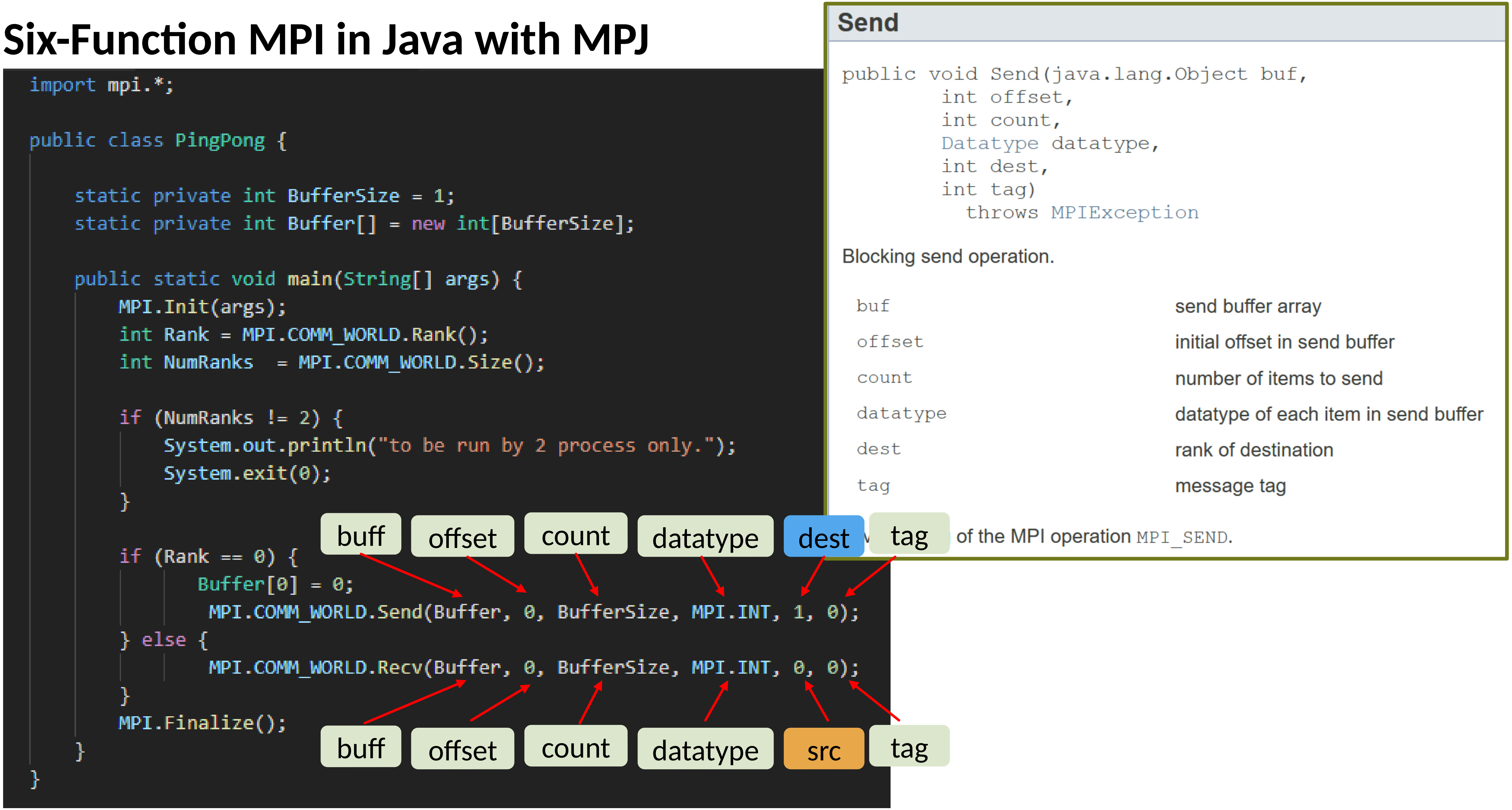
```
public void Send(java.lang.Object buf,
        int offset,
        int count,
        Datatype datatype,
        int dest,
        int tag)
          throws MPIException
```

Blocking send operation.

buf	send buffer array
offset	initial offset in send buffer
count	number of items to send
datatype	datatype of each item in send buffer
dest	rank of destination
tag	message tag

Java binding of the MPI operation MPI_SEND.

Six-Function MPI in Java with MPJ



```java
import mpi.*;

public class PingPong {

    static private int BufferSize = 1;
    static private int Buffer[] = new int[BufferSize];

    public static void main(String[] args) {
        MPI.Init(args);
        int Rank = MPI.COMM_WORLD.Rank();
        int NumRanks  = MPI.COMM_WORLD.Size();

        if (NumRanks != 2) {
            System.out.println("to be run by 2 process only.");
            System.exit(0);
        }

        if (Rank == 0) {
            Buffer[0] = 0;
             MPI.COMM_WORLD.Send(Buffer, 0, BufferSize, MPI.INT, 1, 0);
        } else {
             MPI.COMM_WORLD.Recv(Buffer, 0, BufferSize, MPI.INT, 0, 0);
        }
        MPI.Finalize();
    }
}
```

Send

```
public void Send(java.lang.Object buf,
        int offset,
        int count,
        Datatype datatype,
        int dest,
        int tag)
          throws MPIException
```

Blocking send operation.

buf	send buffer array
offset	initial offset in send buffer
count	number of items to send
datatype	datatype of each item in send buffer
dest	rank of destination
tag	message tag

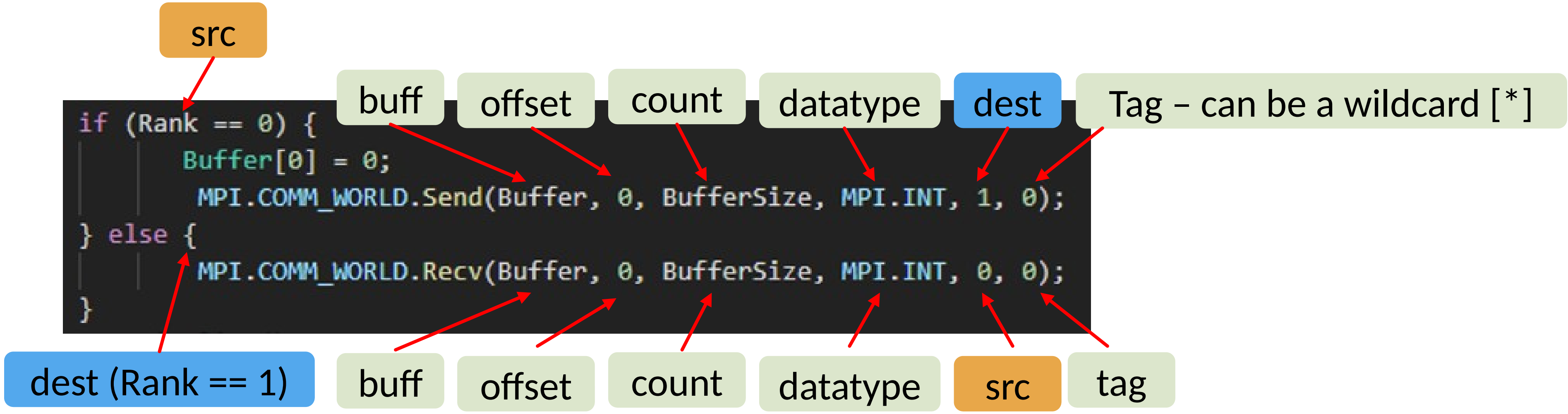
of the MPI operation MPI_SEND.

Message Matching

- **Which receive gets which data?**
- **Sender sends the message to the receiver rank**
- **When it arrives we check all the unmatched, posted receives in the order they were posted**
  - Source, Comm, and Tag must “match” with what the receiver specified – wildcards exist for source and tag
  - If we found a match we are done
- **If no match is found we put the message in a “unexpected messages” queue**
  - When a receive is posted, we check messages in this queue first

Message Matching

- **Which receive gets which data?**
- **Sender sends the message to the receiver rank**
- **When it arrives we check all the unmatched, posted receives in the order they were posted**
  - Source, Comm, and Tag must “match” with what the receiver specified – wildcards exist for source and tag
  - If we found a match we are done
- **If no match is found we put the message in a “unexpected messages” queue**
  - When a receive is posted, we check messages in this queue first



```
if (Rank == 0) {
        Buffer[0] = 0;
         MPI.COMM_WORLD.Send(Buffer, 0, BufferSize, MPI.INT, 1, 0);
} else {
         MPI.COMM_WORLD.Recv(Buffer, 0, BufferSize, MPI.INT, 0, 0);
}
```

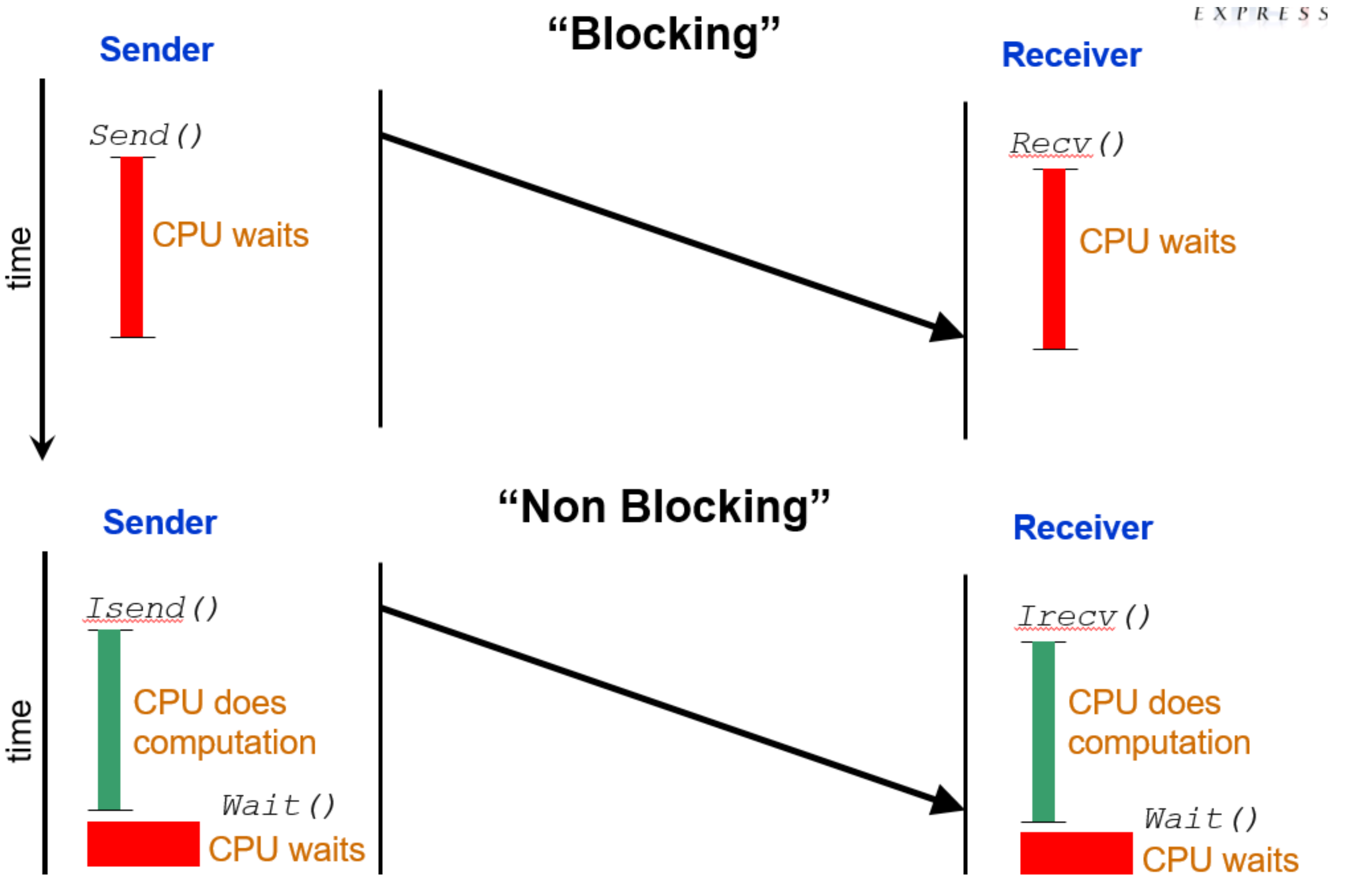
Synchronous / Asynchronous

- **Apart from blocking and intermediate, there is also Asynchronous and Synchronous send:**
- **When a synchronous send completes, you know**
  - You can overwrite the send buffer (same like “normal” send)
  - The receiver has received the message – Huh? What else could happen?

  - In asynchroneous send MPI can copy your message to an internal buffer! Now you can reuse the send buffer, but you don’t know anything about the receiver.

Blocking vs Non-blocking / Immediate

- The Send/Recv in our six-function MPI are blocking
- Meaning: When they return we can overwrite the send buffer / read the receive buffer
- This means we are wasting time! – Use Isend/Irecv + Wait to overlap “waiting” with doing something useful!



Communicators

- **All processes are a part of the MPI_COMM_WORLD communicator**
- **MPI_COMM_WORLD exists automatically**
- **Messages do not “match” across communicators (good to provide isolation)**
- **Communicators can be created for arbitrary subsets of processes**
  - MPI_Comm_dup() -- create a copy of a communicators (same procs in it but messages do not cross-match)
  - MPI_Comm_split() – divide a communicator in two according two colors given to processes
  - …
- **For this lecture, we only care about MPI_COMM_WORLD**

Collectives

- **When using MPI, a couple of patterns always repeat:**
  - I have some data on one rank, but I want all ranks to have it
  - I want to sum up data from all ranks and have the result on rank 0
  - I want to sum up data from all ranks and have the result on all ranks

- **With our six function MPI this is easy to solve!**
- **Just a for-loop from 0..P-1 with some sends and receives…**
  - This is slow (you learned about tree-based reductions in the lecture)
  - It would be really annoying to write this for every bigger MPI code

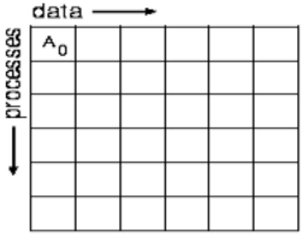
Collectives

- **When using MPI, a couple of patterns always repeat:**
  - I have some data on one rank, but I want all ranks to have it broadcast
  - I want to sum up data from all ranks and have the result on rank 0 reduce
  - I want to sum up data from all ranks and have the result on all ranks all-reduce

- **With our six function MPI this is easy to solve!**
- **Just a for-loop from 0..P-1 with some sends and receives…**
  - This is slow (you learned about tree-based reductions in the lecture)
  - It would be really annoying to write this for every bigger MPI code

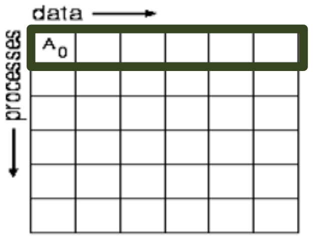
Collectives

- **MPI defines these patterns for us!**



Collectives

- **MPI defines these patterns for us!**

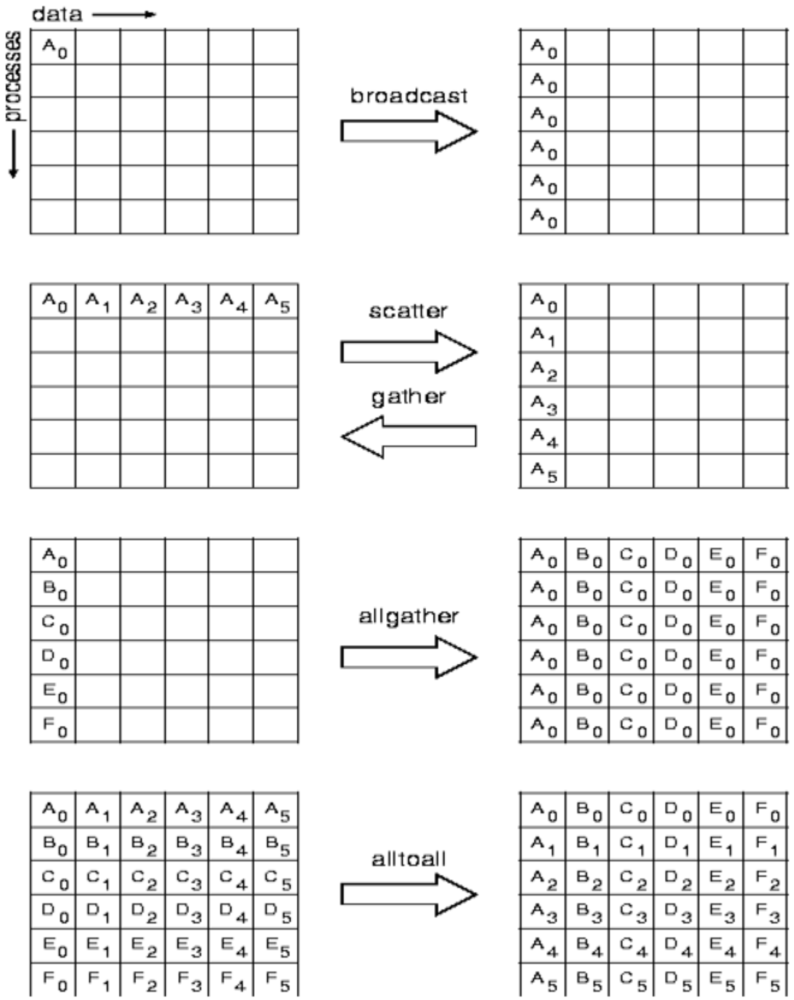


Local buffer of rank 0 can contain up to 6 elements.

At the beginning, it holds only element A_0

Collectives

- **MPI defines these patterns for us!**



A_0					

A_0					
A_0					
A_0					
A_0					
A_0					
A_0					

A_0	A_1	A_2	A_3	A_4	A_5

A_0					
A_1					
A_2					
A_3					
A_4					
A_5					

A_0					
B_0					
C_0					
D_0					
E_0					
F_0					

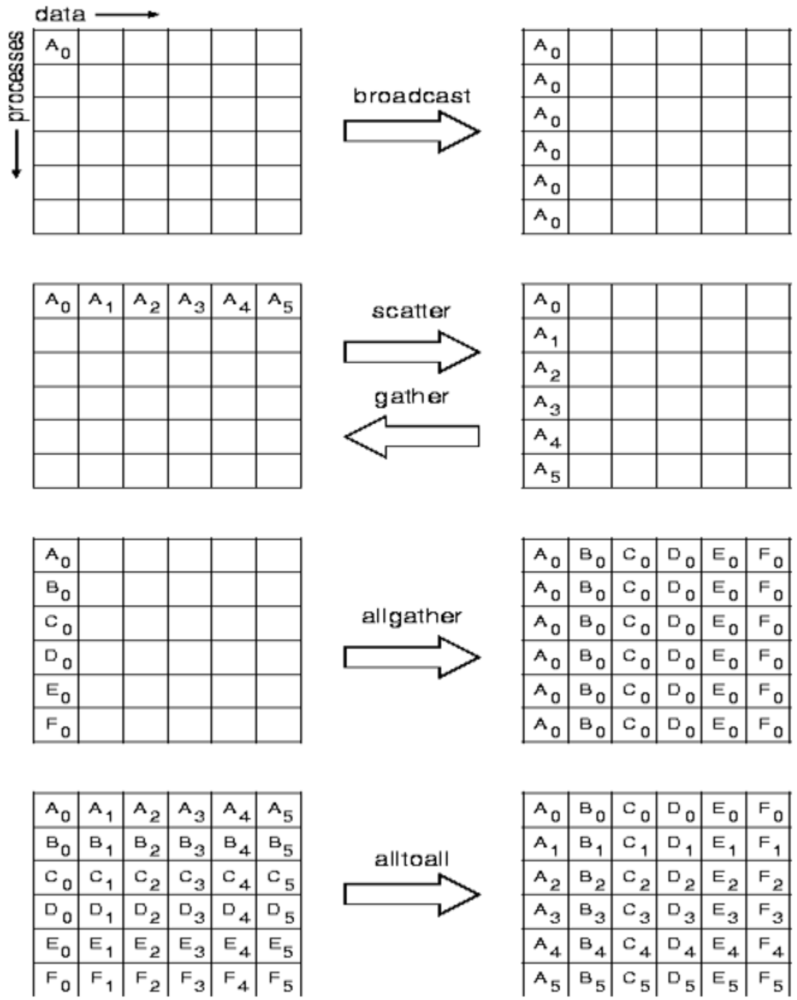
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0
A_0	B_0	C_0	D_0	E_0	F_0

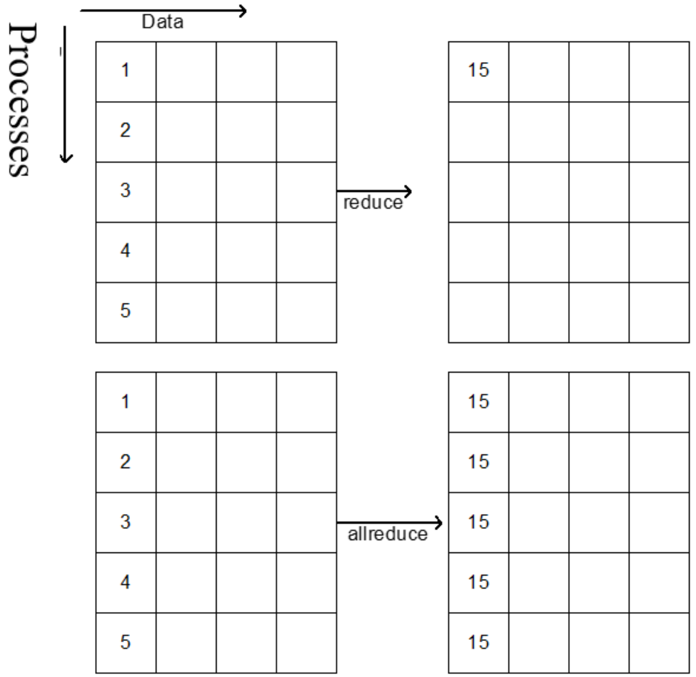
A_0	A_1	A_2	A_3	A_4	A_5
B_0	B_1	B_2	B_3	B_4	B_5
C_0	C_1	C_2	C_3	C_4	C_5
D_0	D_1	D_2	D_3	D_4	D_5
E_0	E_1	E_2	E_3	E_4	E_5
F_0	F_1	F_2	F_3	F_4	F_5

A_0	B_0	C_0	D_0	E_0	F_0
A_1	B_1	C_1	D_1	E_1	F_1
A_2	B_2	C_2	D_2	E_2	F_2
A_3	B_3	C_3	D_3	E_3	F_3
A_4	B_4	C_4	D_4	E_4	F_4
A_5	B_5	C_5	D_5	E_5	F_5

Collectives

- **MPI defines these patterns for us!**





- ➢ MPI.PROD
- ➢ MPI.SUM
- ➢ MPI.MIN
- ➢ MPI.MAX
- ➢ MPI.LAND
- ➢ MPI.BAND
- ➢ MPI.LOR
- ➢ MPI.BOR
- ➢ MPI.LXOR
- ➢ MPI.BXOR
- ➢ MPI.MINLOC
- ➢ MPI.MAXLOC

Exercise 1

- **Set up MPJ in Eclipse and Run a “Hello World” example, i.e., print the rank of each process in MPI_COMM_WORLD.**

Exercise 2

- How can we time how long a message takes to be delivered?
- We do not have synchronized timers across processes!

Idea: Send a message back and forth, so we can time on one process how long this takes and divide by two.

Exercise 3

- Implement a parallel prime sieve, each process works on different data
- Use collective communication where it makes sense

Exercise 4

- Implement your own reduce for the operator + on MPI_COMM_WORLD
- Use send/recv (or variants) to implement all communication
- Do not use more than O(P*log(P)) messages in total (for P processes)