

# Four Variations of Matrix Multiplication

About 30 minutes - by Mike

# Matrix Multiplication Challenge

---

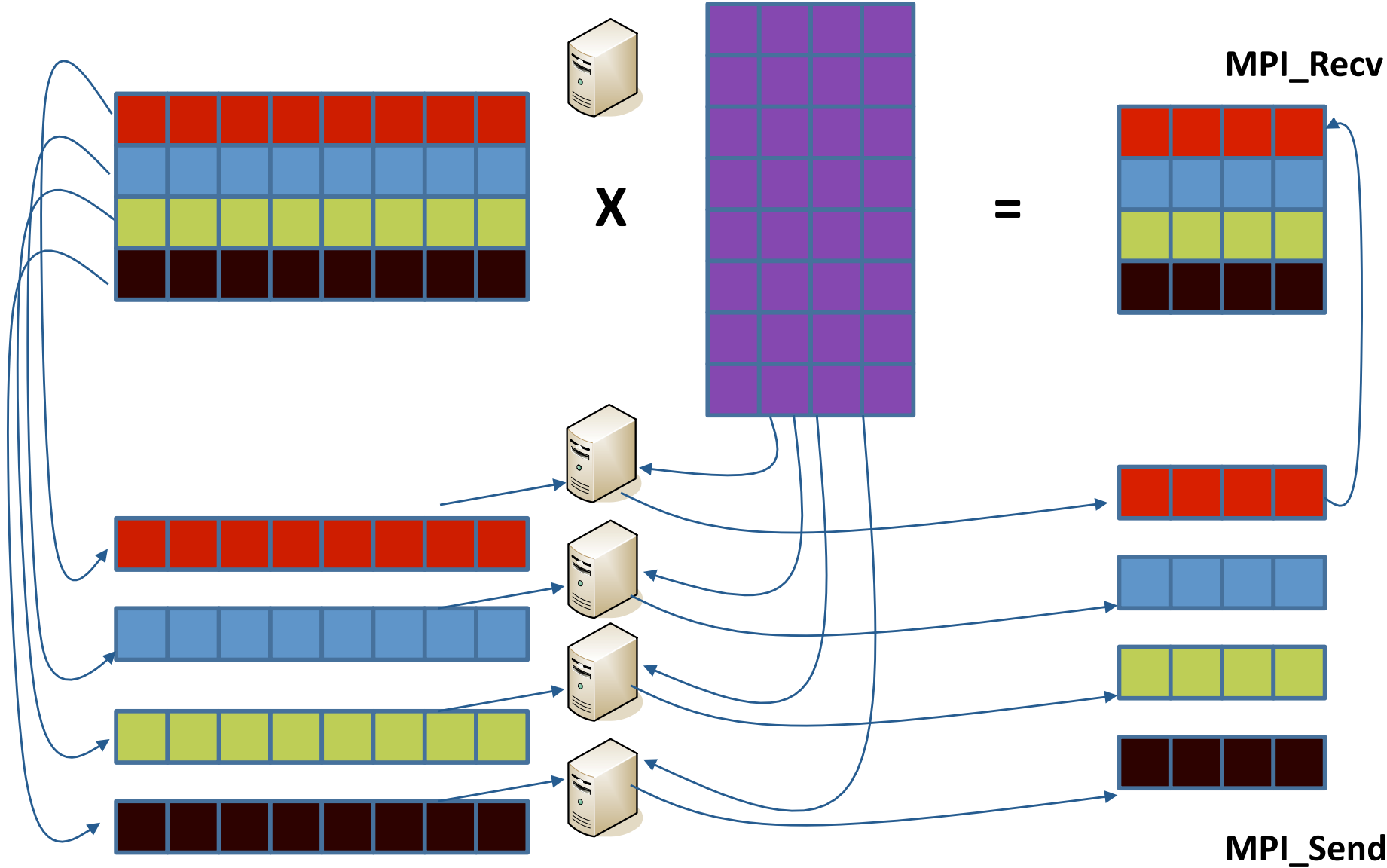
- See
  - [http://www.cs.utah.edu/formal\\_verification](http://www.cs.utah.edu/formal_verification)
  - Look under Concurrency Education
  - Look at MPI teaching resources
    - Many of these resources are due to Simone Atzeni
    - Many are due to Geof Sawaya
      - All the examples from Pacheco's MPI book!
      - This will soon be available as projects within our ISP Eclipse GUI!
- Matrix Challenge is due to Steve Siegel
  - See <http://www.cs.utah.edu/ec2>
  - Steve's challenge stems from an MPI book It is based on an example from the book

[Using MPI: Portable Parallel Programming with the Message-Passing Interface](#) by William Gropp, Ewing Lusk, and Anthony Skjellum.

# Matrix Multiplication Illustration

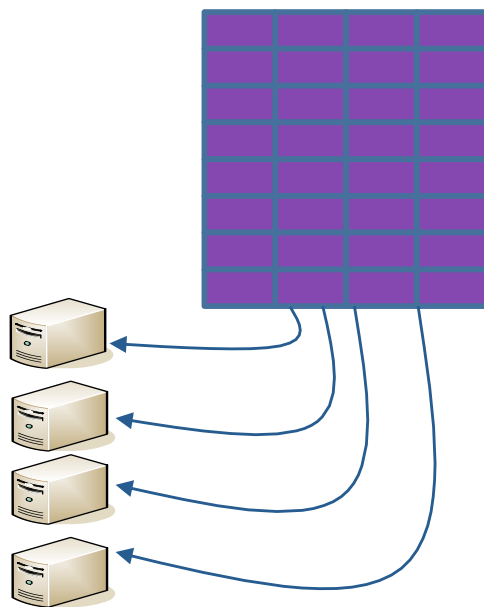
- For this tutorial, we include four variants
- These try various versions of mat-mult
- Includes one buggy version
- Also reveals one definite future work item
  - Detect symmetries in MPI programs
  - Avoid redundant searches
  - Very apparent when you run our fourth version

# Example of MPI Code (Mat Mat Mult)



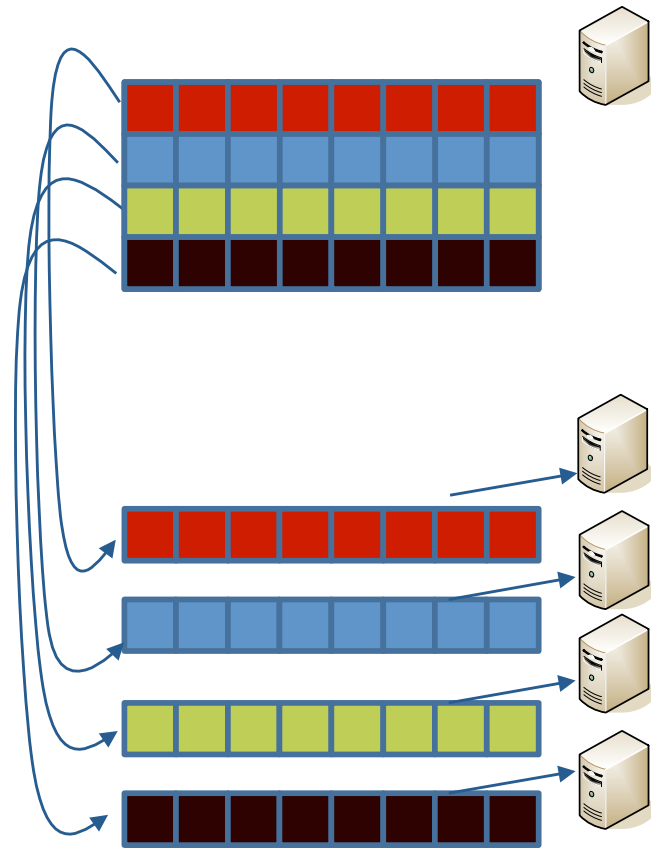
# Salient Code Features

---



```
if (myid == master) {  
    ...  
    MPI_Bcast(b, brows*bcols, MPI_FLOAT, master, ...);  
    ...  
}  
else { // All Slaves do this  
    ...  
    MPI_Bcast(b, brows*bcols, MPI_FLOAT, master, ...);  
    ...  
}
```

# Salient Code Features



```
if (myid == master) {
```

```
...
```

```
for (i = 0; i < numprocs-1; i++) {
```

```
for (j = 0; j < acols; j++) {
```

```
buffer[j] = a[i*acols+j];
```

```
}
```

```
MPI_Send(buffer, acols, MPI_FLOAT, i+1, ...);
```

```
numsent++;
```

```
}
```

```
}
```

*Block till buffer is copied  
into System Buffer*

System Buffer

```
else { // slaves
```

```
...
```

```
while (1) {
```

```
...
```

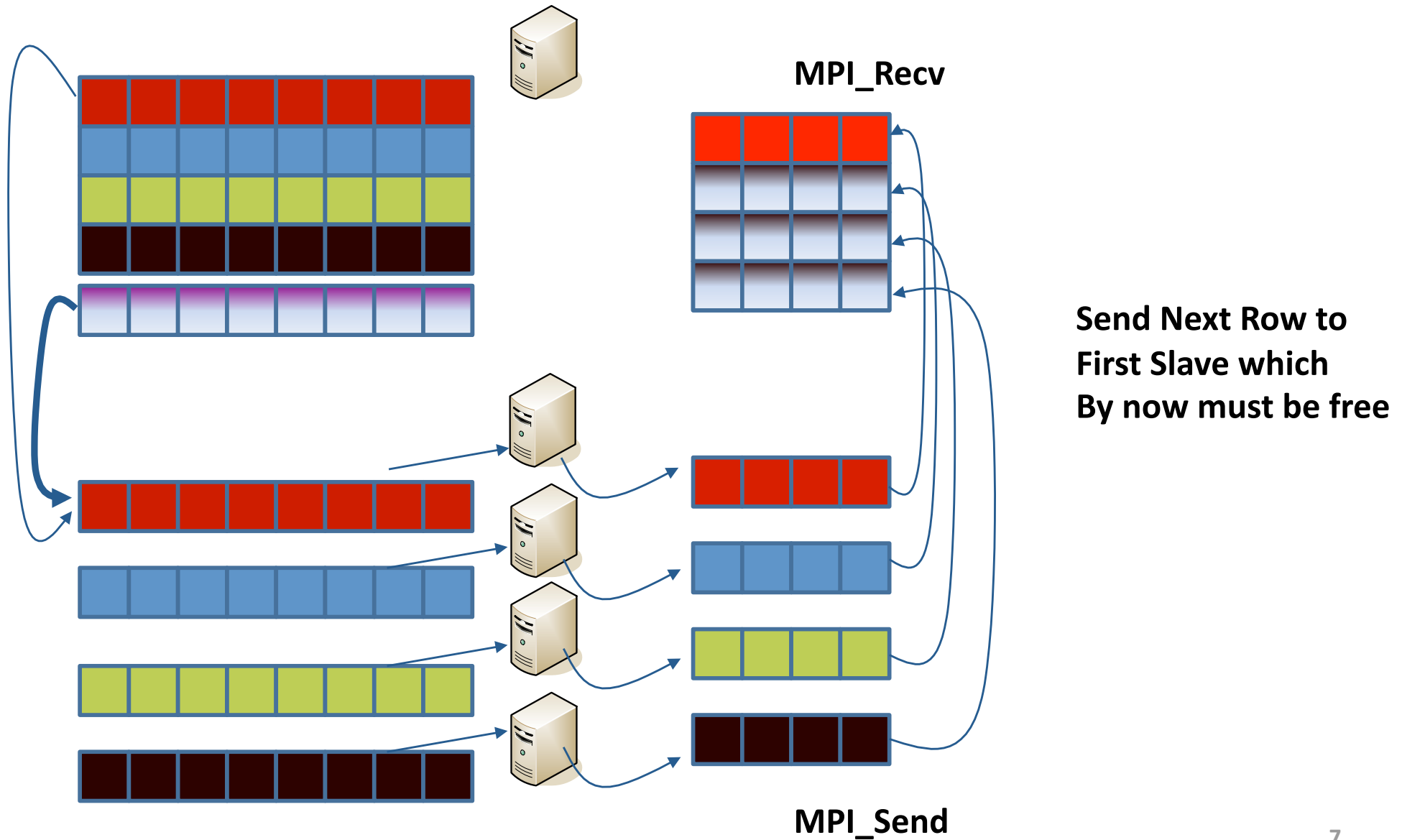
```
MPI_Recv(buffer, acols, MPI_FLOAT, master, ...);
```

```
...
```

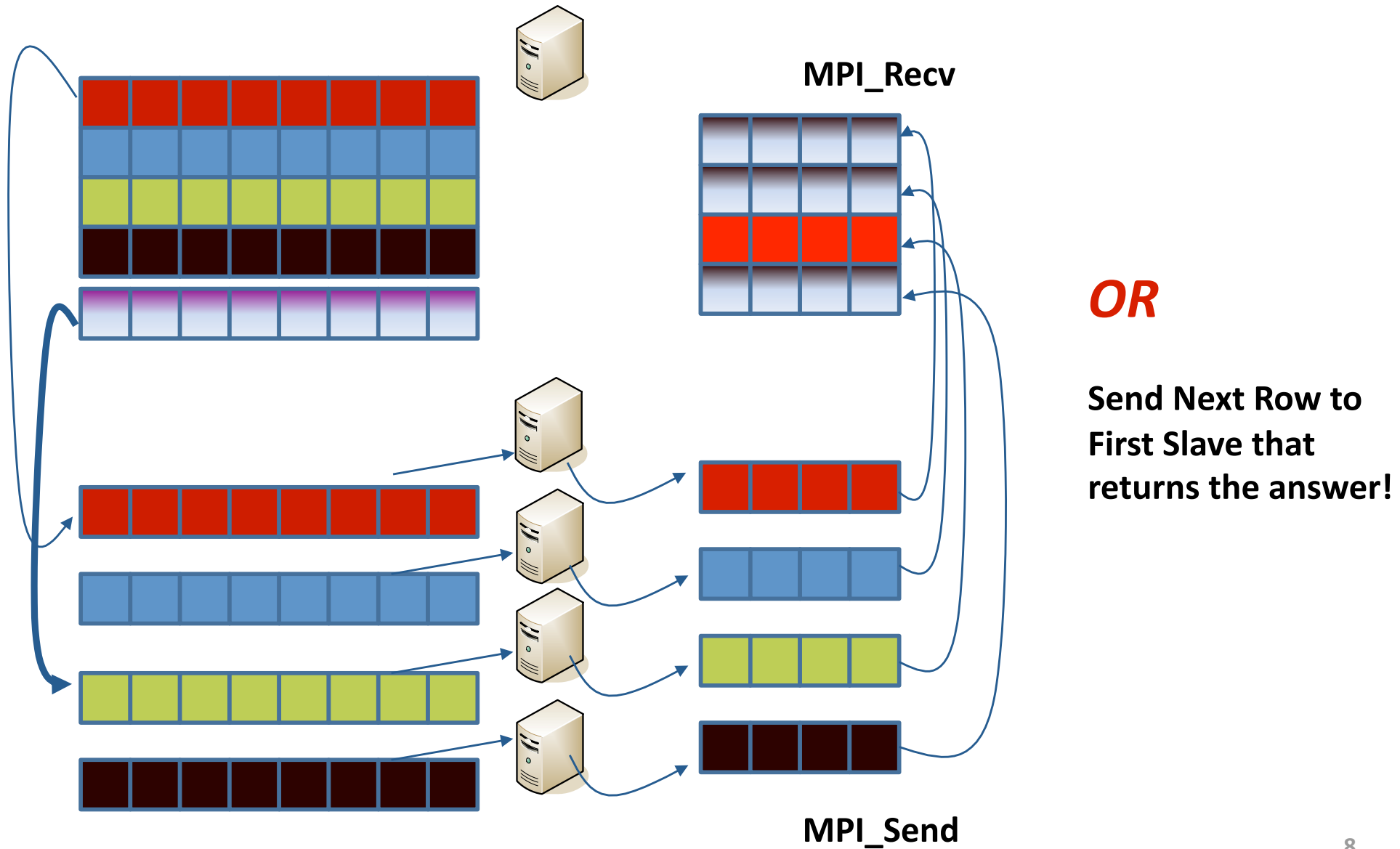
```
}
```

```
}
```

# Handling Rows $\gg$ Processors ...



# Handling Rows >> Processors ...



# Optimization

---

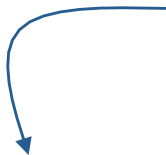
```
if (myid == master) {  
    ...  
    for (i = 0; i < crows; i++) {  
        MPI_Recv(ans, ccols, MPI_FLOAT, FROM ANYBODY, ...);  
        ...  
        if (numsent < arows) {  
            for (j = 0; j < acols; j++) {  
                buffer[j] = a[numsent*acols+j];  
            }  
            MPI_Send(buffer, acols, MPI_FLOAT, BACK TO THAT BODY, ...);  
            numsent++;  
        }  
    }  
}
```

# Optimization

---

```
if (myid == master) {  
  ...  
  for (i = 0; i < crows; i++) {  
    MPI_Recv(ans, ccols, MPI_FLOAT, FROM ANYBODY, ...);  
    ...  
    if (numsent < arows) {  
      for (j = 0; j < acols; j++) {  
        buffer[j] = a[numsent*acols+j];  
      }  
      MPI_Send(buffer, acols, MPI_FLOAT, BACK TO THAT BODY, ...);  
      numsent++;  
    }  
  }  
}
```

Shows that wildcard receives  
can arise quite naturally ...



# Further Optimization

---

```
if (myid == master) {  
    ...  
    for (i = 0; i < crows; i++) {  
        MPI_Recv(ans, ccols, MPI_FLOAT, FROM ANYBODY, ...);  
        ...  
        if (numsent < arows) {  
            for (j = 0; j < acols; j++) {  
                buffer[j] = a[numsent*acols+j];  
            }  
            ... here, WAIT for previous Isend to finish  
            (software pipelining) ...  
            MPI_Isend(buffer, acols, MPI_FLOAT, BACK TO THAT BODY, ...);  
            numsent++;  
            ...  
        }  
    }  
}
```

# Summary of Some MPI Commands

- `MPI_Irecv(source, msg_buf, req_struct, ..)`
- This is a non-blocking receive call
- `MPI_Wait(req_struct)` awaits completion
- Source could be
  - “wildcard” or \* or ANY\_SOURCE
- Receive from any eligible (matching) sender

End of E