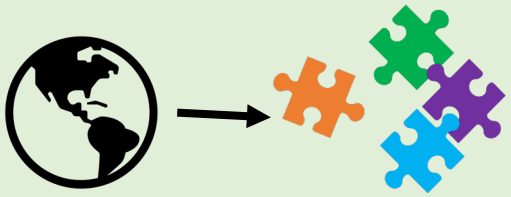
HPC



1)Algorithms

2)Architecture

Solutions

3)Shared Memory
OpenMP

4)Distributed Memory
MPI

5) Unified Engine
Spark

Create a strategy to communicate inside your group

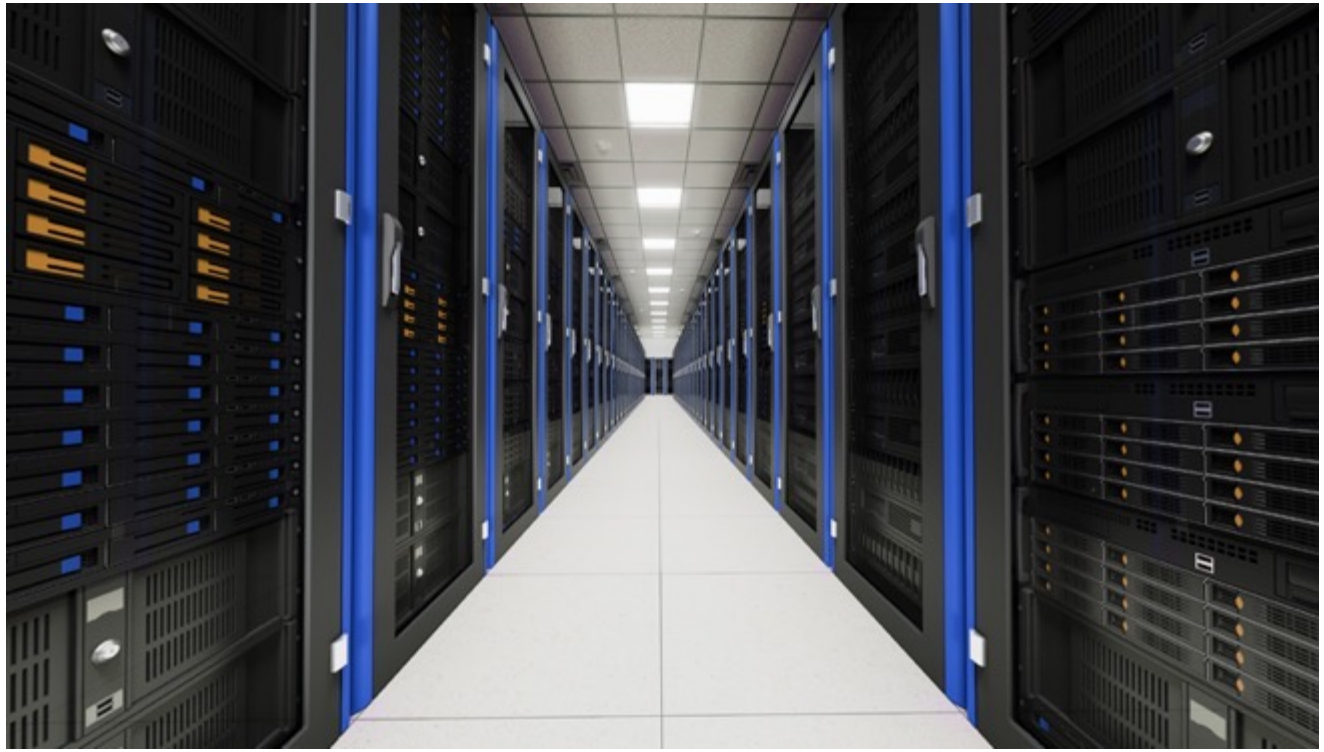
What problems did we encounter?

3)Distributed Memory

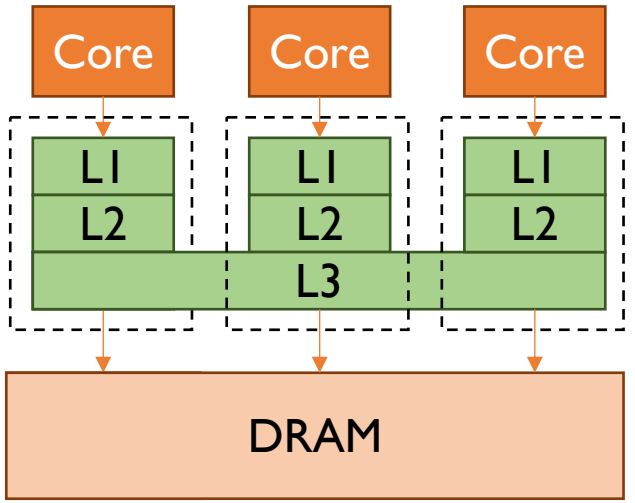
MPI

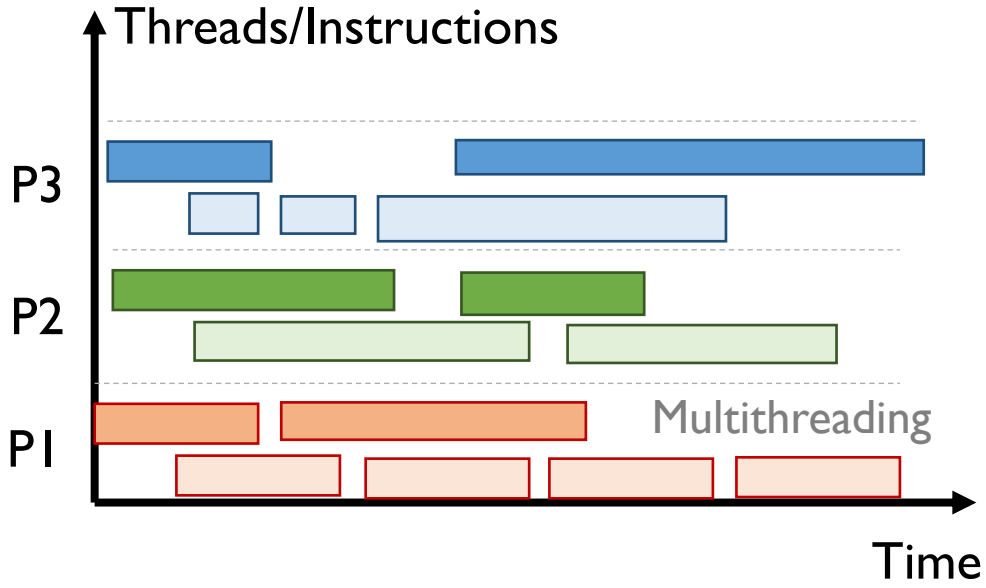
Communication decisions from Laptop to HPC cluster

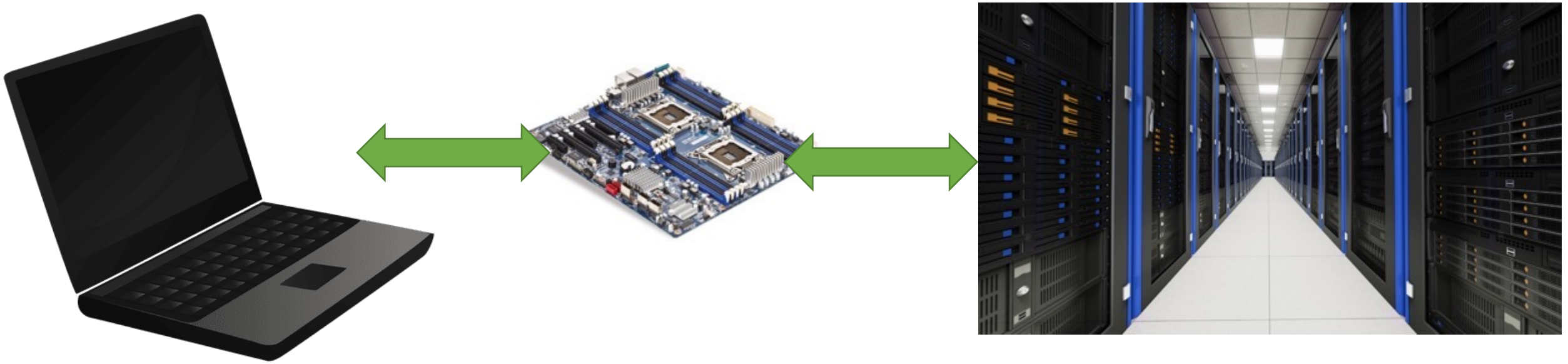
?



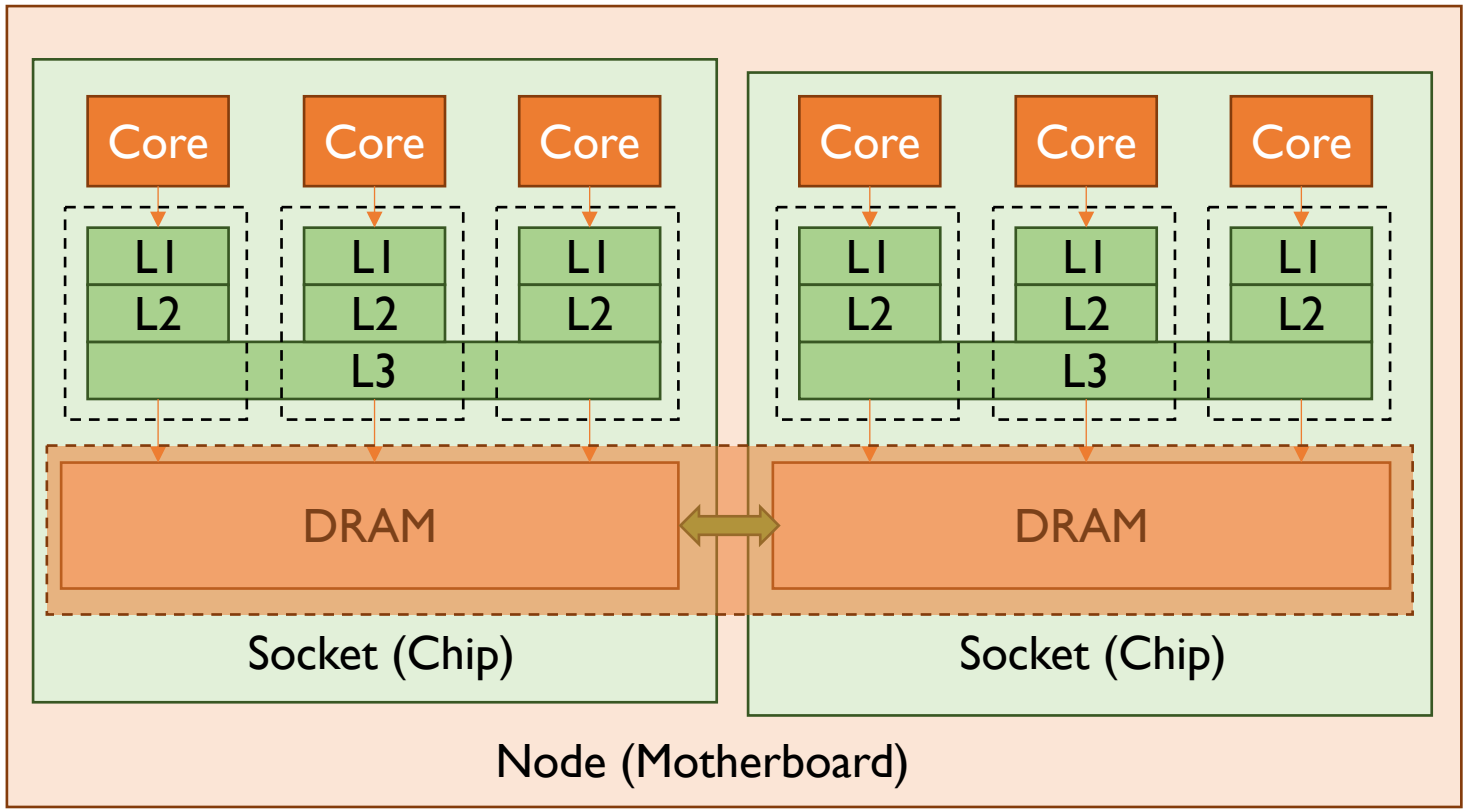
Multiple Cores



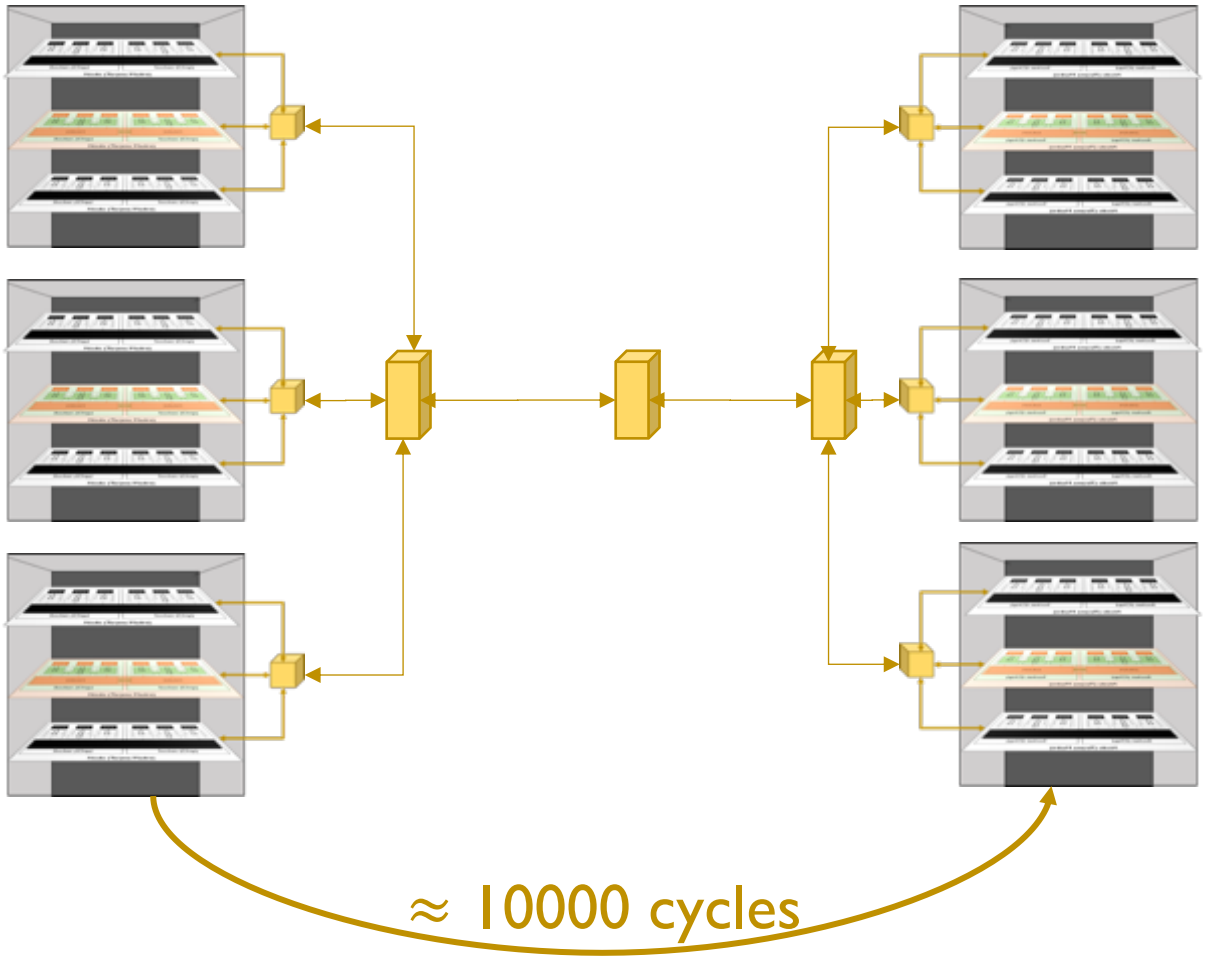




Non Uniform Memory Access(NUMA)



Distributed Memory



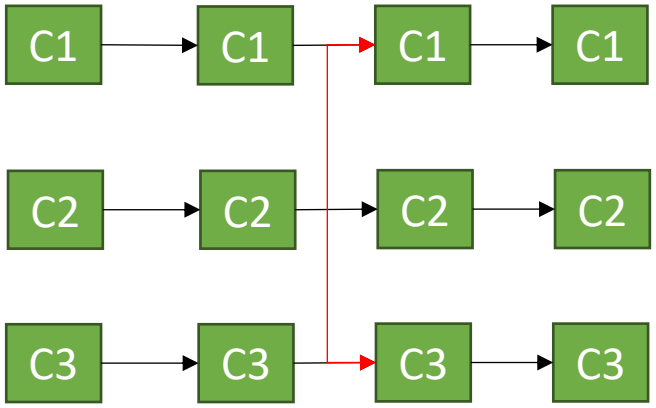
Spark

OpenMP

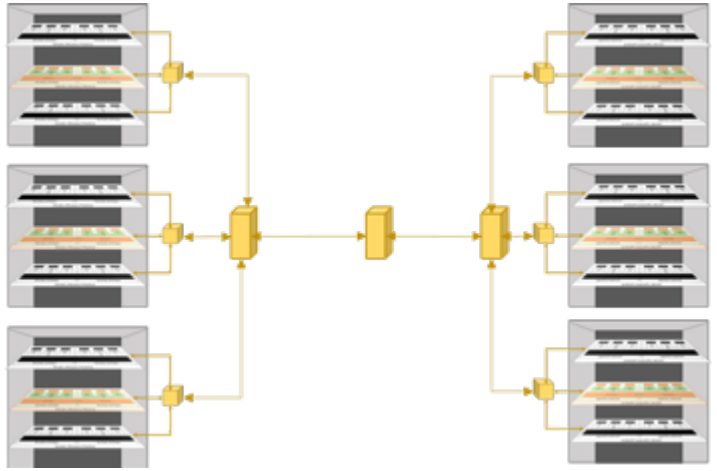
MPI

MPI + Scheduler (slurm)

Coarse granularity



Distributed Memory



The standard for communication

What do we need in a distributed memory system?

Distribute instructions

Manage communication over the network

Instructions -> Scheduler

Coarse Level of granularity

Done in the terminal (i.e. Slurm)

1. Resources allocation salloc / sbatch
2. Execute jobs srun
3. Manage queues squeue

Communication -> MPI

Distributed Memory

Done in the code (i.e. C/FORTRAN)

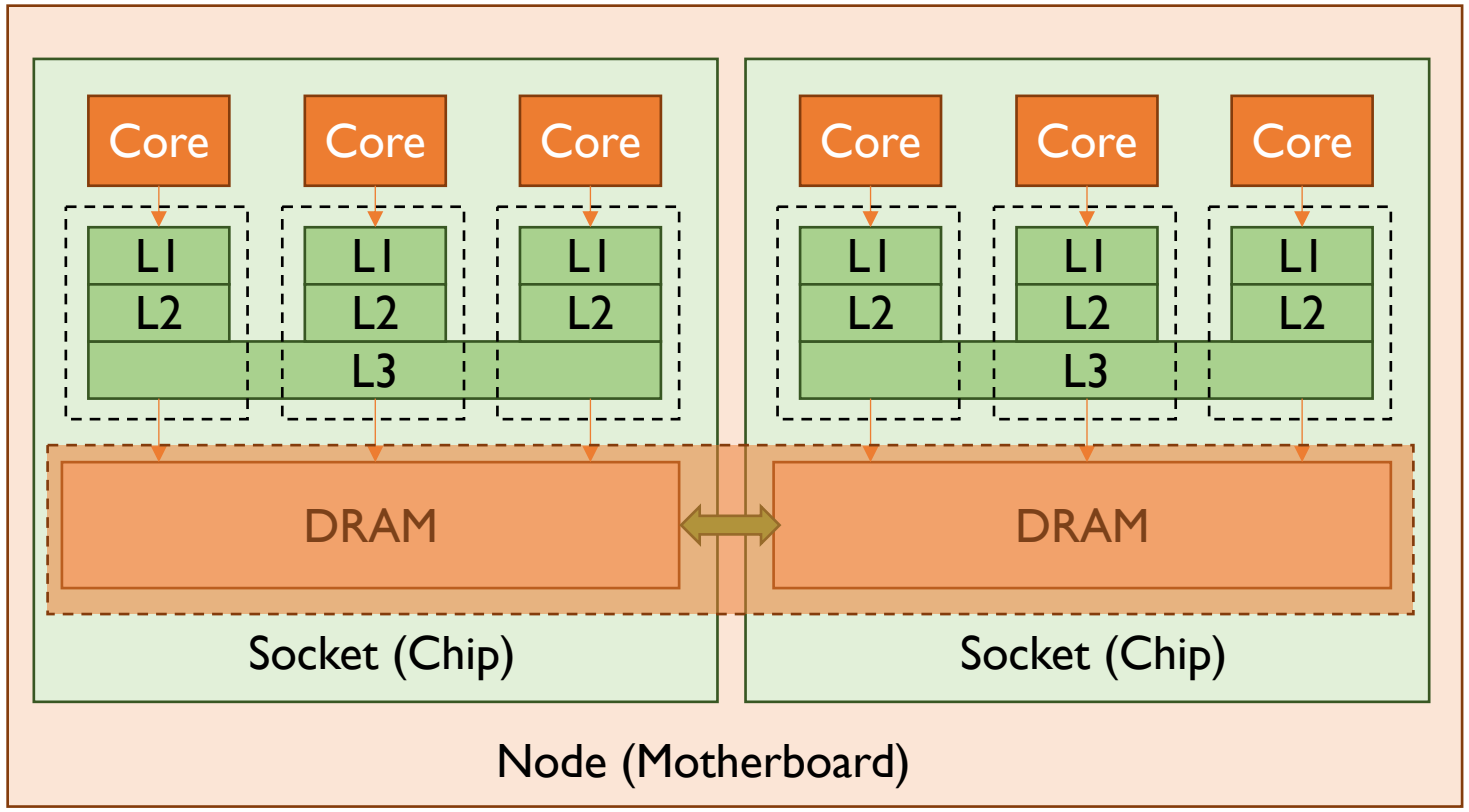
1. Communication channels communicators
2. Establish communication
    1. Point to point Send/Receive
    2. Collective Broadcast/Reduce/Gather/Scatter
    3. One sided (RMA) Put/Get
3. Synchronize Barrier/Fence/Wait
4. Data Types Vector/Topologies/Bytes/Int/…

slurm

Resource Allocation
salloc/sbatch

Run applications
srun

```
--nodes
--sockets-per-node
--cores-per-socket

--ntasks-per-node
```



slurm scheduling scontrol/squeue

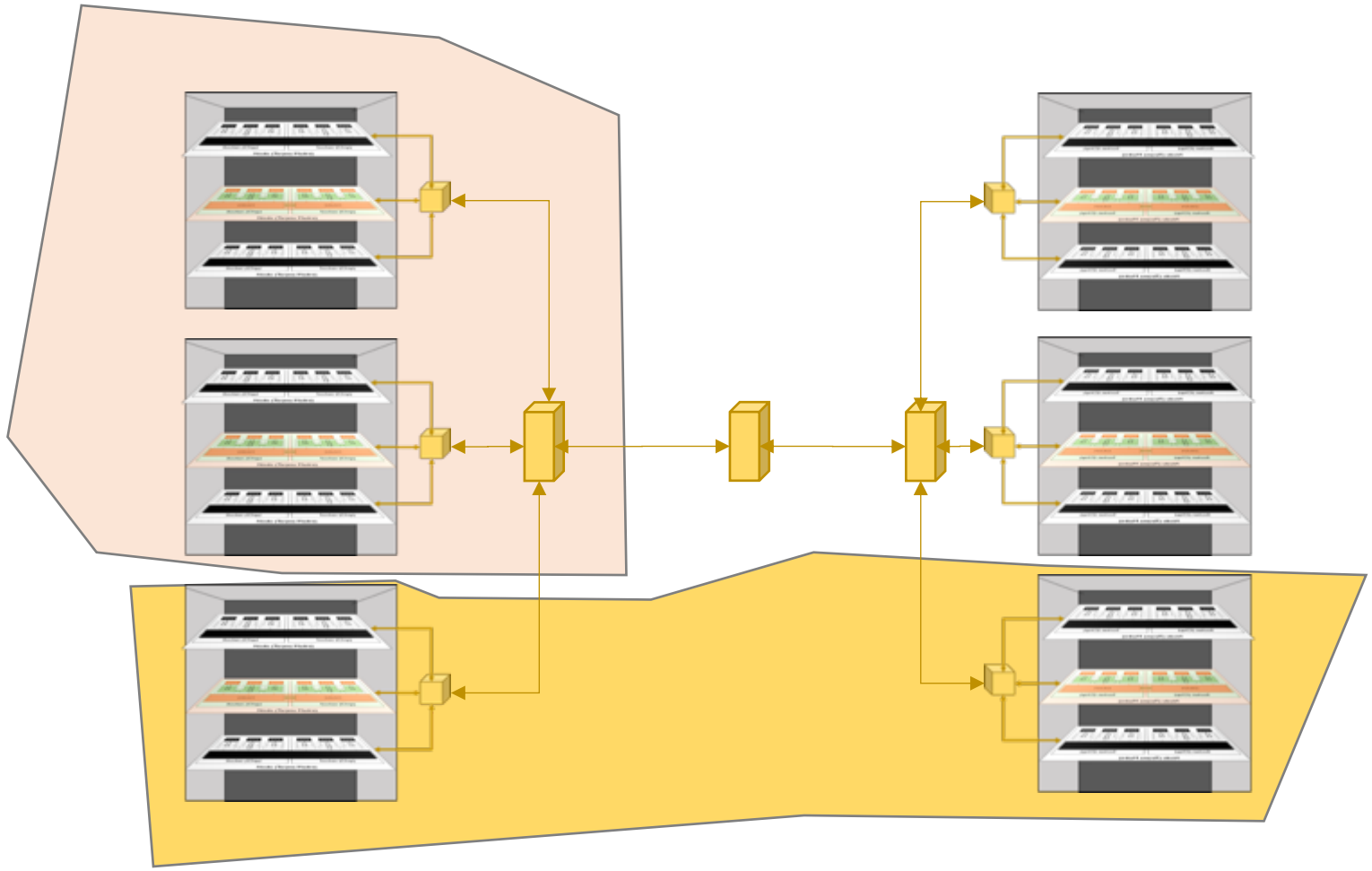
Partitions

Priority

Levels

Time

Amount of resources



MPI is The Standard for distributed memory

Standard for message-passing library interface

Vendor-dependent implementation

C/C++ & Fortran applications

https://www.open-mpi.org/

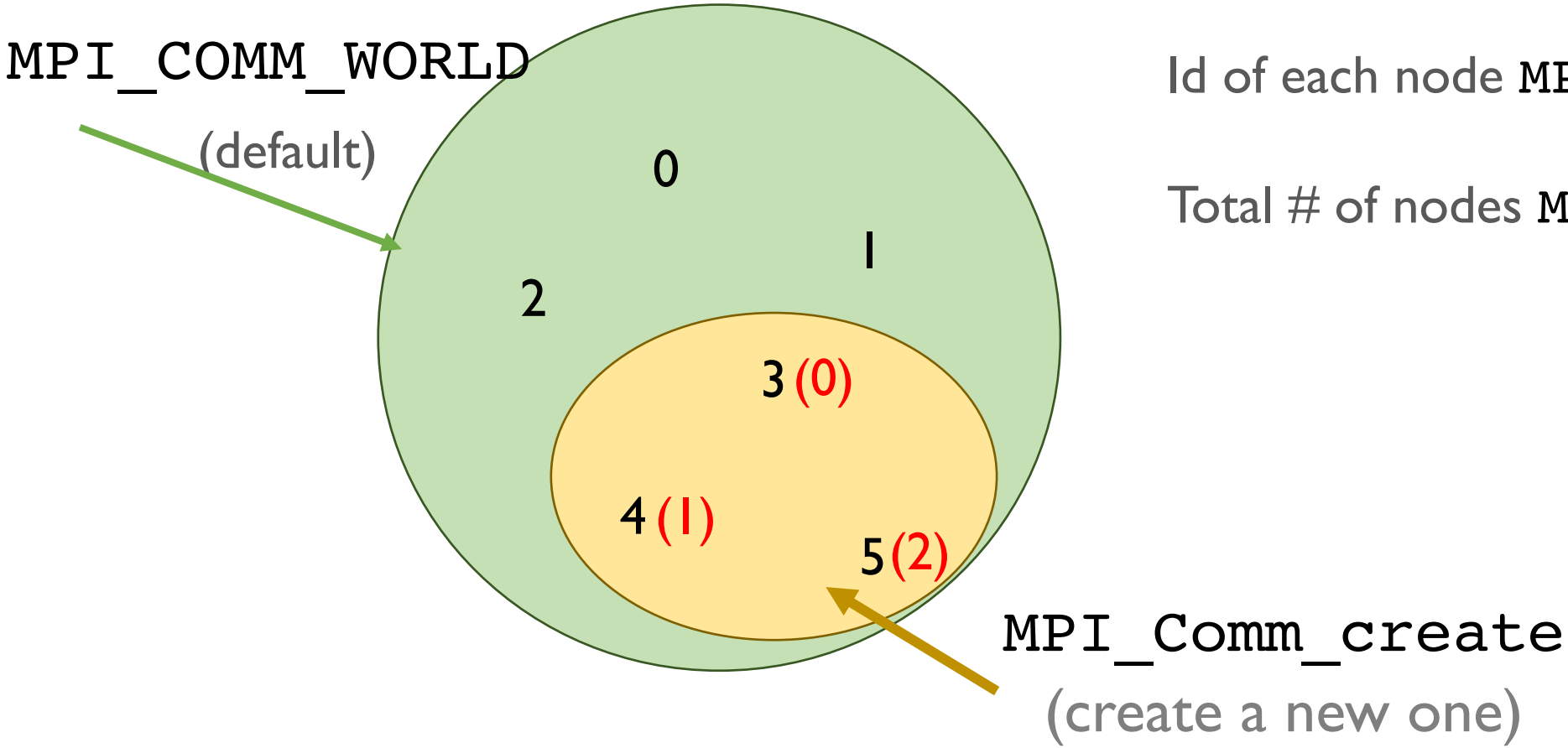
MPI in a nutshell

Control Start MPI application

Start	`MPI_Init(&argc, &argv)`
	`MPI_Init_thread(&argc,&argv,&required,&provided)`
	`MPI_Finalize()`

Environment Information	`MPI_Get_processor_name(&name,&resultlen)`
Timing	`MPI_Wtime()` `MPI_Wtick()`

Communicators = communication scope



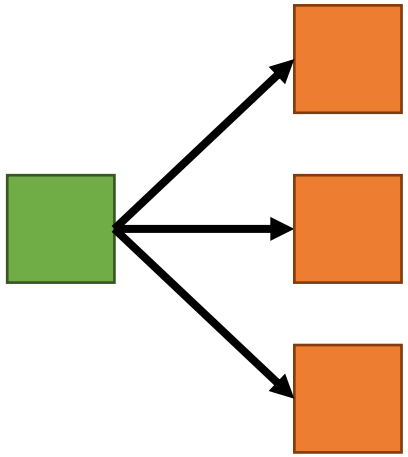
Id of each node `MPI_Comm_rank`

Total # of nodes `MPI_Comm_size`

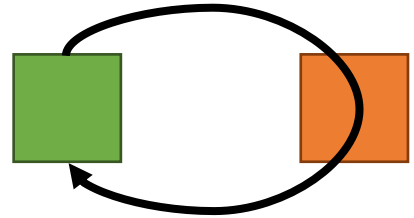
Communication Styles

Point-to-point

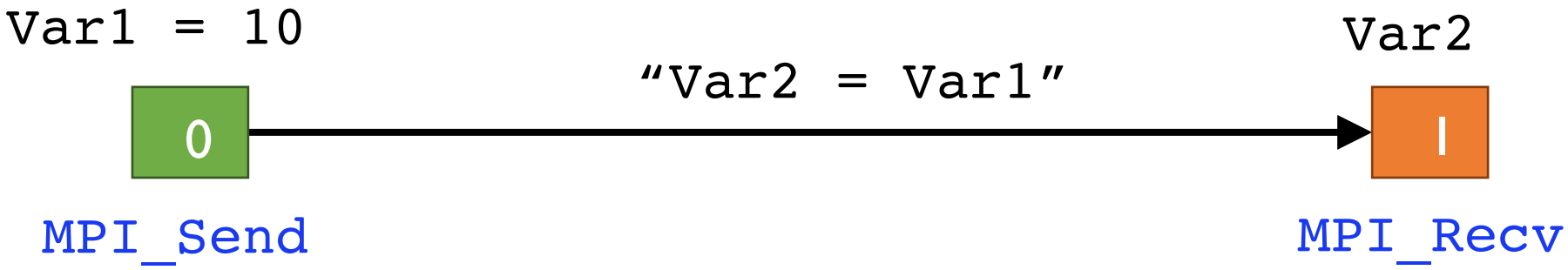
Collective



One sided



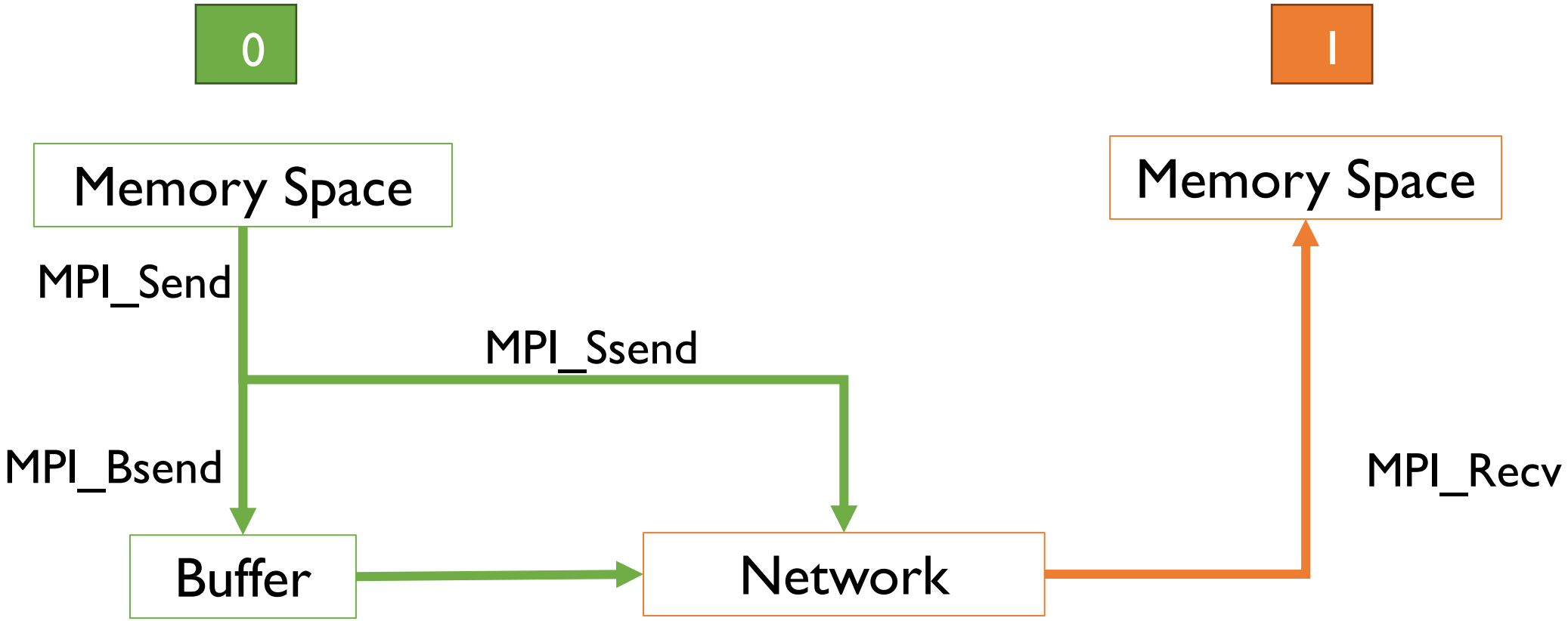
Point to Point Communication



local Ref | length | Type | To/From | Id | Communicator | Status

```
MPI_Send(&var1, 1 ,MPI_DOUBLE, 1 ,tag,MPI_COMM_WORLD);

MPI_Recv(&var2, 1 ,MPI_DOUBLE, 0 ,tag,MPI_COMM_WORLD,status);
```

Blocking Communication = Memory space can be reused

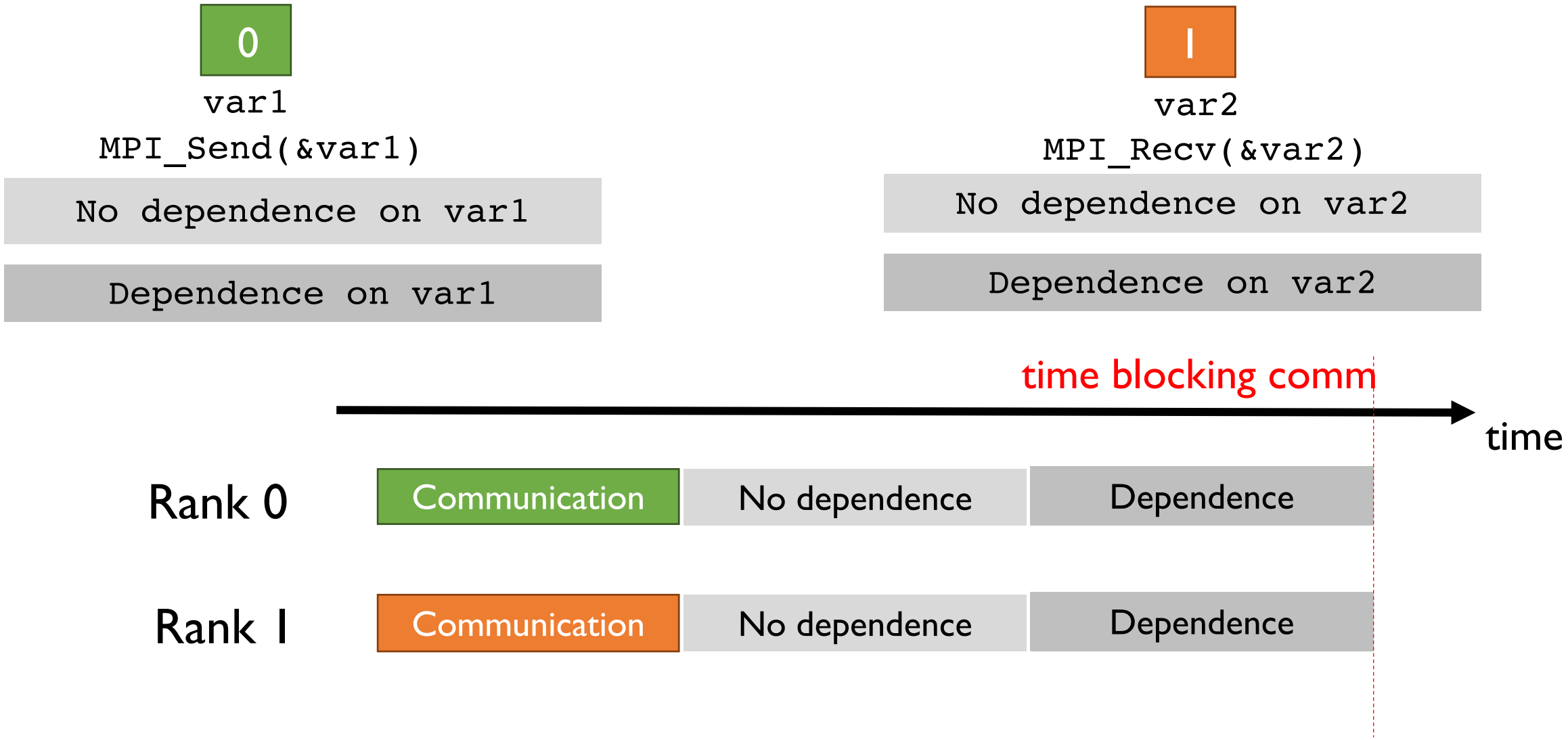


S = Synchronous, B = Buffered

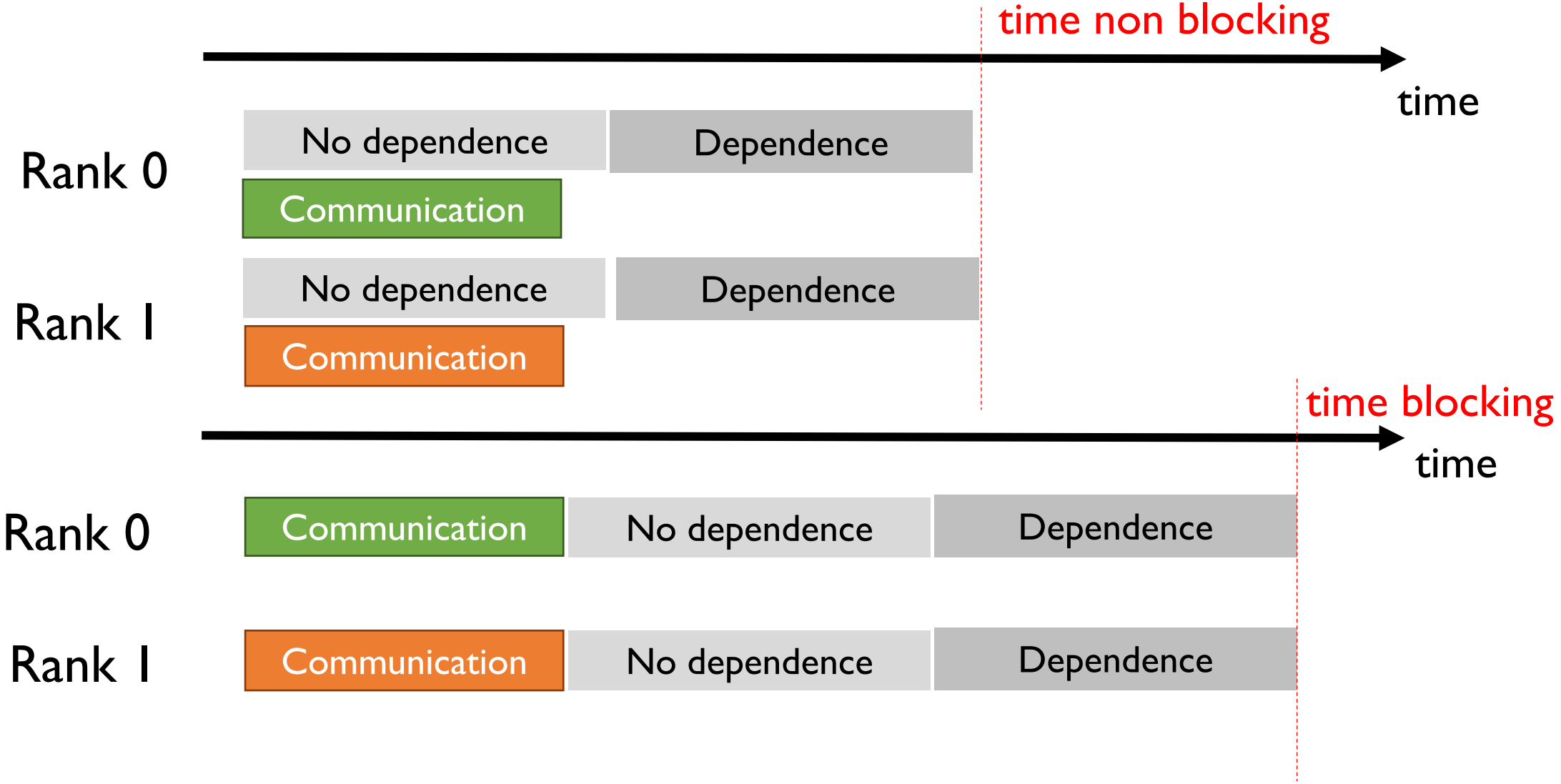
Deadlock


```
var1                    var3
var2                    var4

MPI_Send(&var1,...)     MPI_Send(&var3,...)

MPI_Recv(&var2,...)     MPI_Recv(&var4,...)
```

Problem of Blocking Communication



Hide Communication with Non Blocking Communication

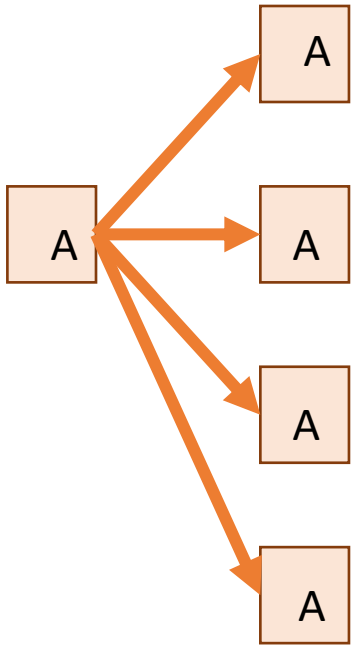


Non Blocking Communication

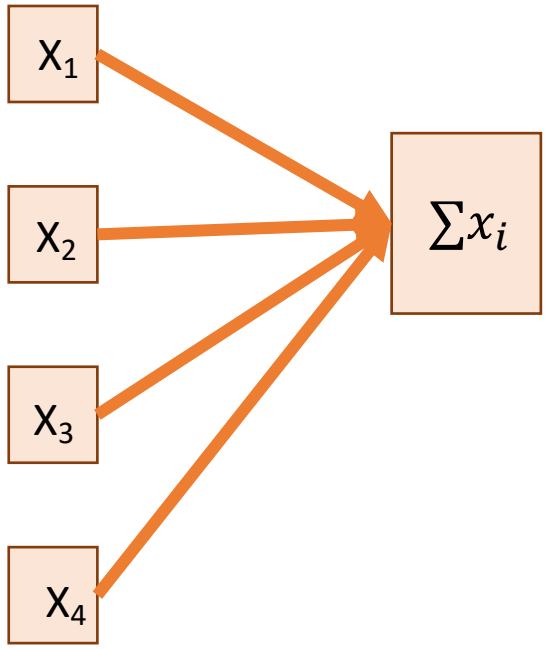
Start	Complete
`MPI_Irecv`	`MPI_Test(request,flag,…)`
`MPI_Isend`	
`MPI_Ibsend`	`MPI_Wait(request,…)`
`MPI_Issend`	

Collective Communication

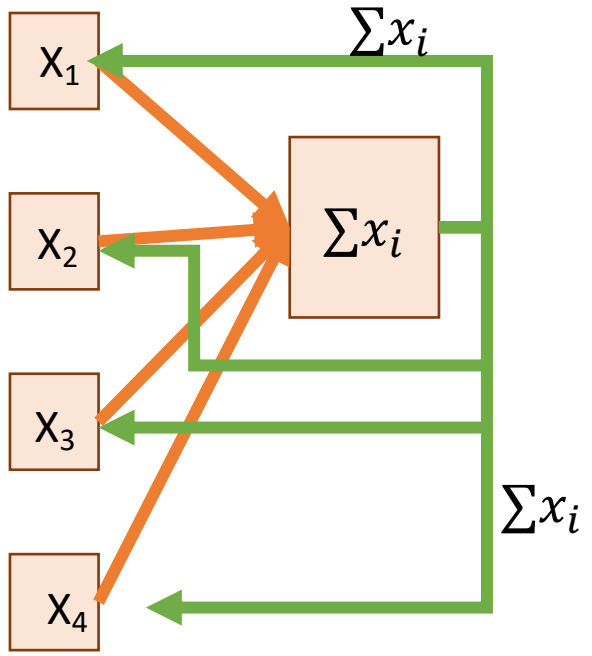
MPI_Bcast



MPI_Reduce



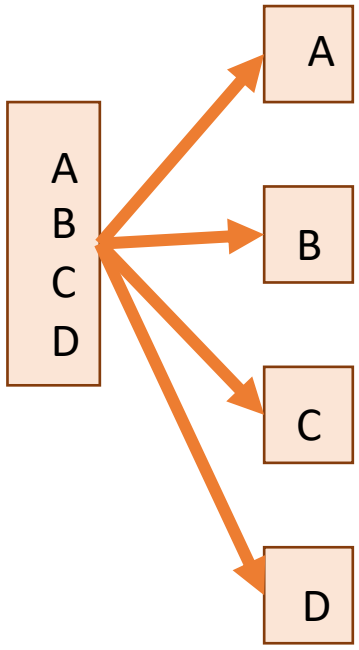
MPI_Allreduce



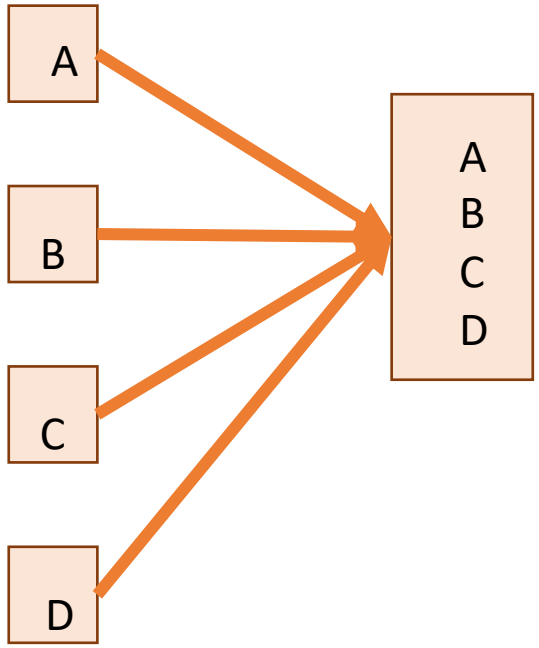
* By default are blocking, non blocking alternative adding I i.e. MPI_Ireduce

Collective Communication

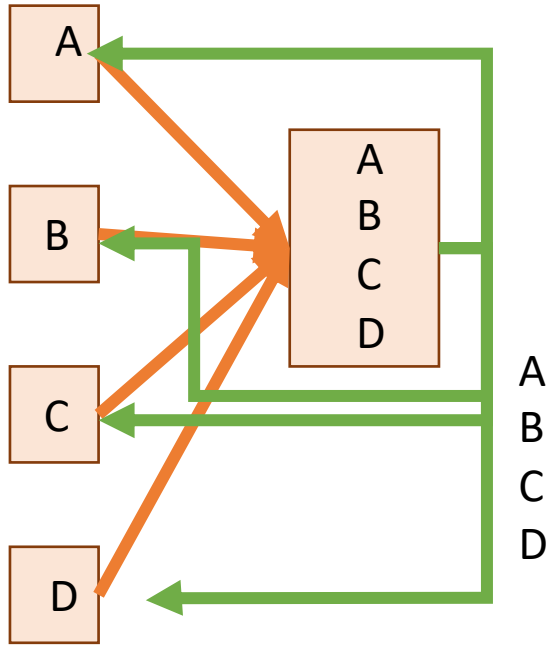
MPI_Scatter



MPI_Gather

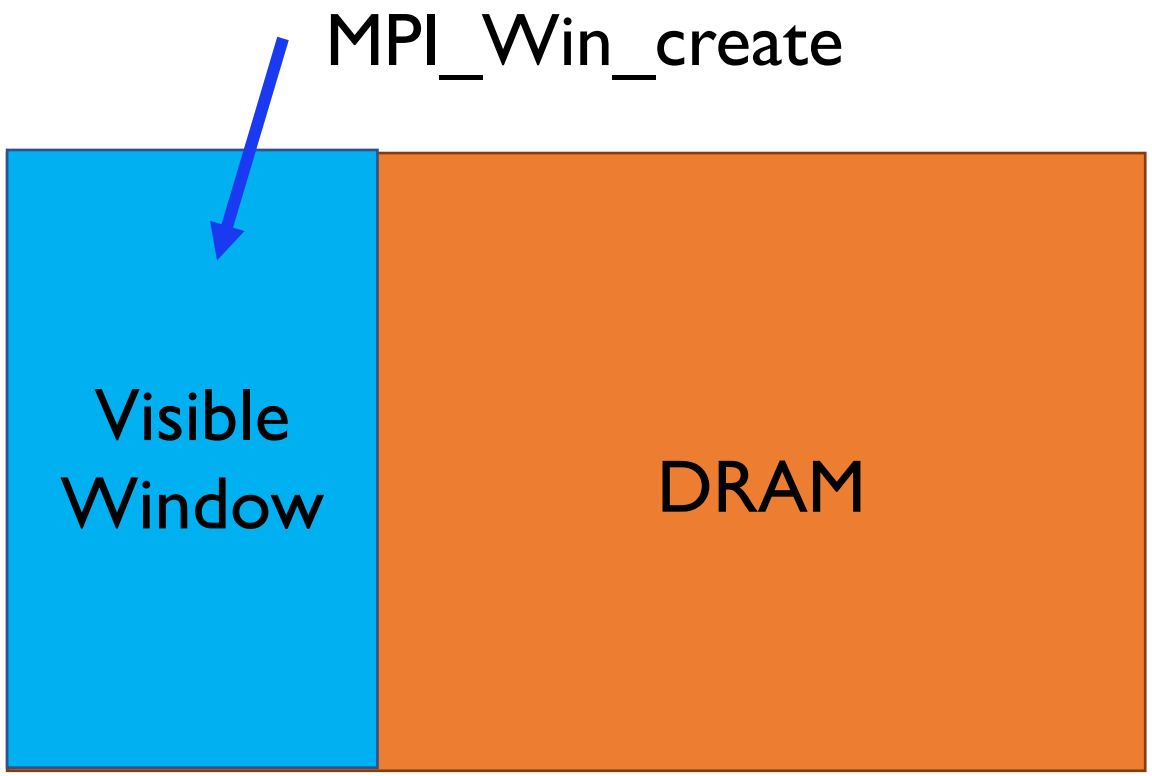


MPI_Allgather



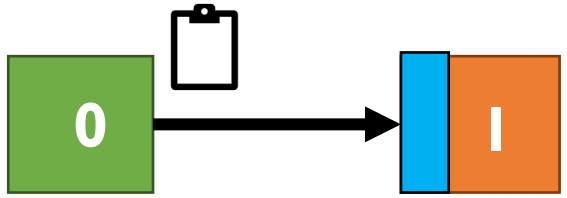
* By default are blocking, non blocking alternative adding I i.e. MPI_Iscatter

One sided communication (RMA)

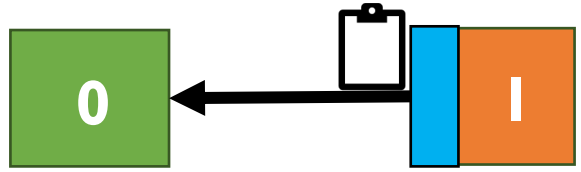


One sided communication (RMA)

MPI_Put



MPI_Get



MPI_ Acummulate

MPI_Compare_and_swap

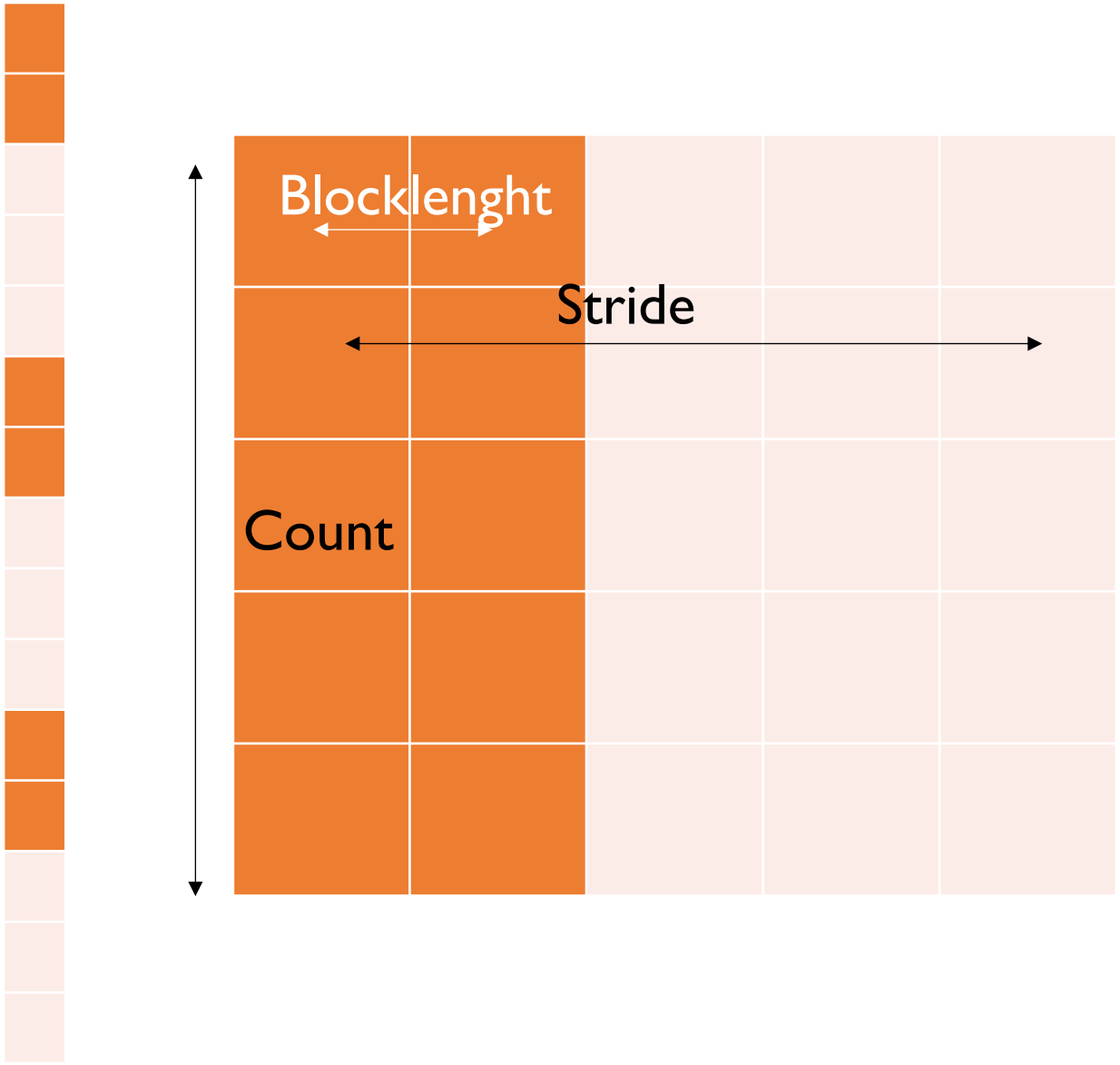
MPI_Win_fence()

Unified Data Types

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG (as a synonym)	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_BYTE	
MPI_PACKED	

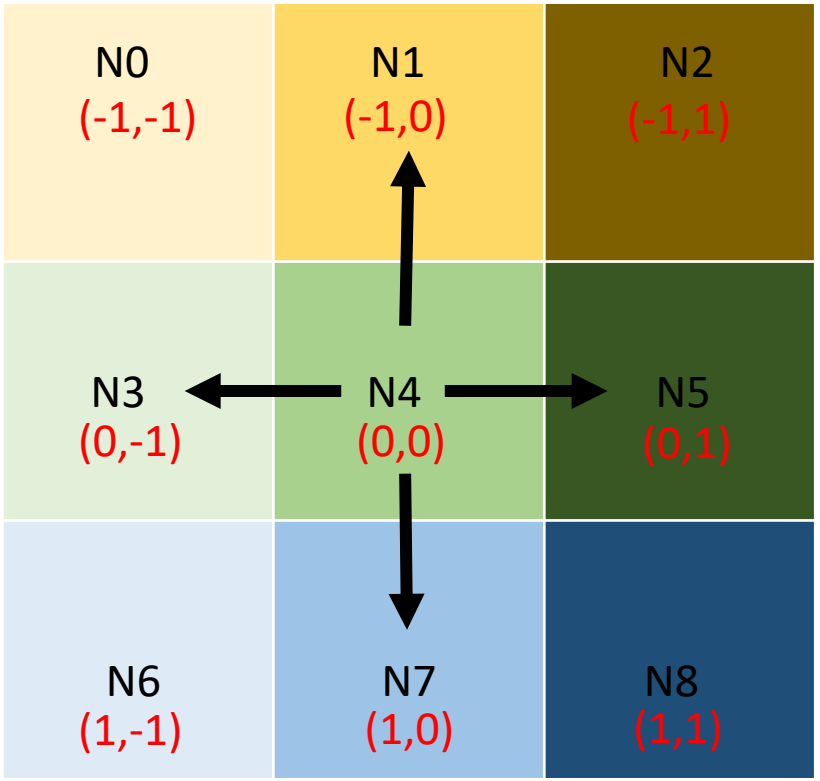
Custom Data Types

```
MPI_Type_vector

MPI_Type_commit

MPI_Type_free
```



Topologies

`MPI_Cart_create`



MPI is The Standard for distributed memory

Standard for message-passing library interface

Vendor-dependent implementation

C/C++ & Fortran applications

https://www.open-mpi.org/

How to run my application?

1) Have a MPI distribution

Open MPI, MPICH, IBM MPI, Intel MPI, MVAPICH

2) In file include library: `#include <mpi.h>`

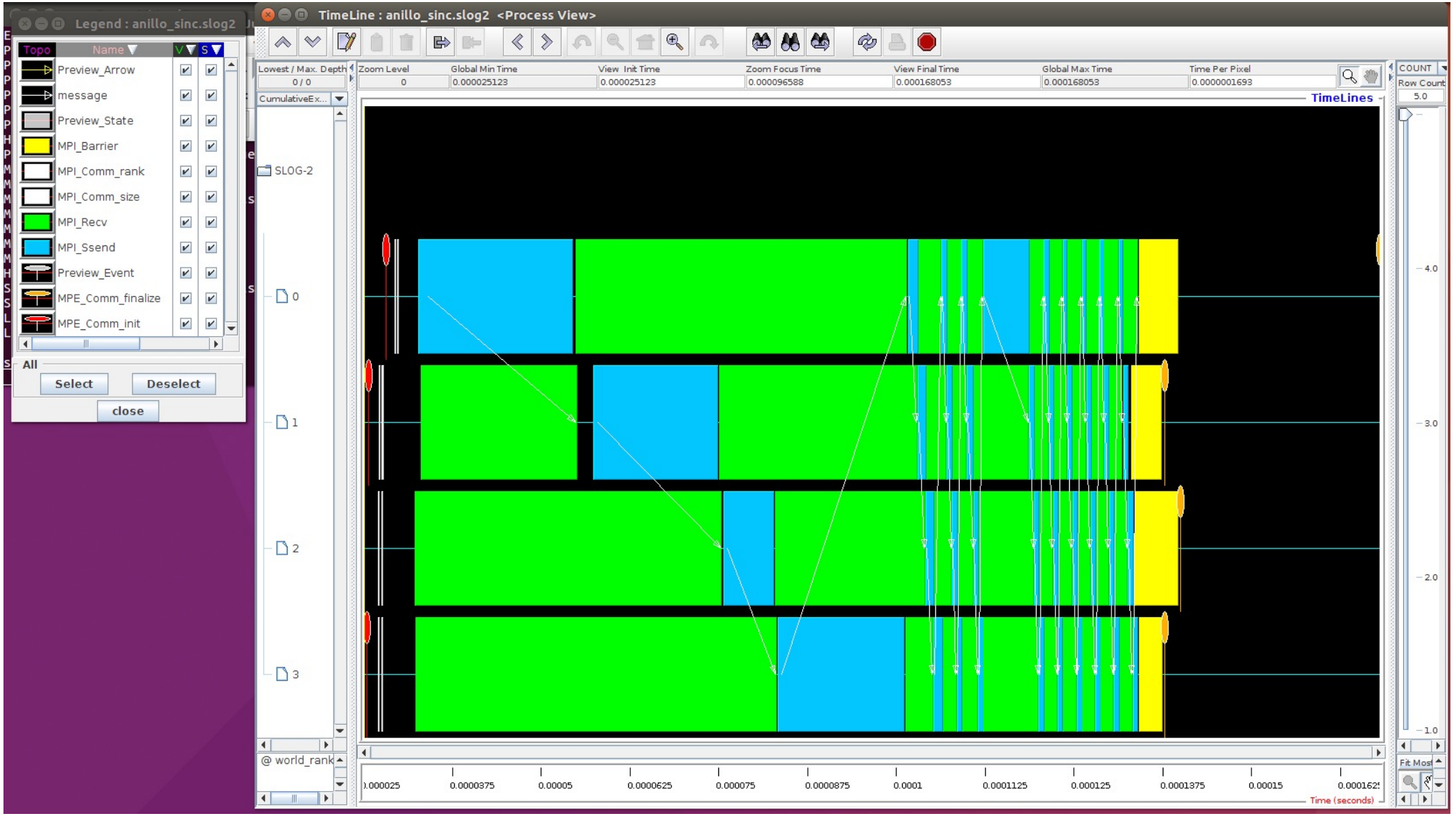
3) Compile using distribution-specific compiler wrapper

```
mpicc hello_world_mpi.c -o hello
```

4) Execute defining number of nodes

```
mpirun –n 2 hello
```

Measure Performance: Tracing with Jumpshot



To use MPE

1) Compile with MPI

```
mpicc -g -c ring.c
```

2) Linking with MPE

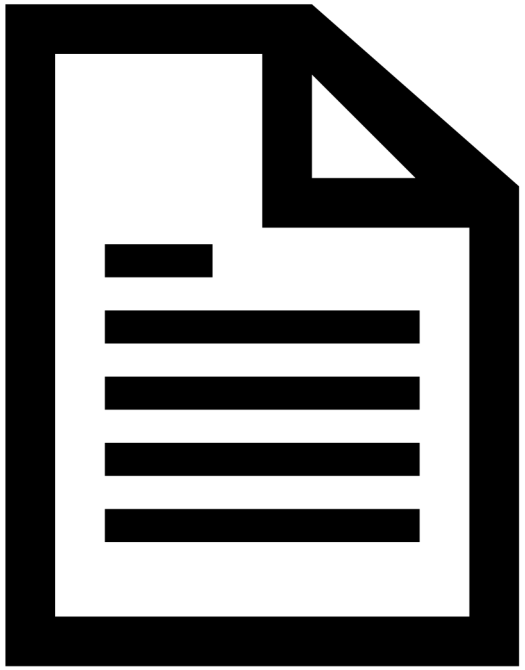
```
mpecc -lm -mpilog ring.o -o ring
```

3) Execute as standard MPI

```
mpirun –n 2 ring
```

4) Run `jumpshot` and convert `ring.clog2` into `ring.slog2`

References



MPI API https://www.open-mpi.org/doc/current/

MPI 3.1 Standard https://www.mpi-forum.org/docs/

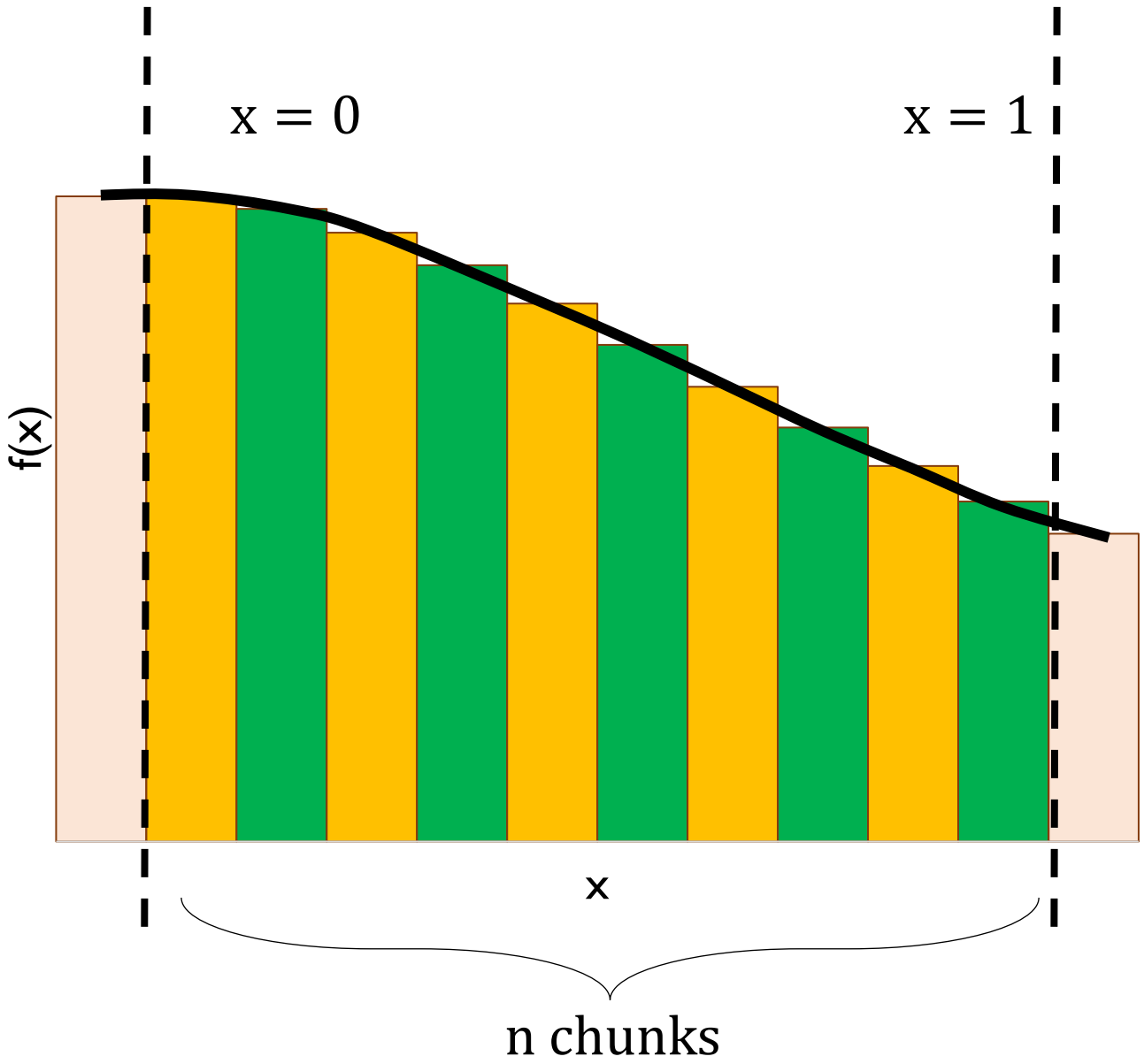
Jumpshot http://www.mcs.anl.gov/research/projects/perfvis/software/viewers/index.htm

Example: Compute Pi ($\pi = 3.14159\ldots$) in parallel

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx$$

P 0 $= \sum_{i=0,2,\ldots}^{n-1} f(x_i) * \frac{1}{n}$

P 1 $= \sum_{i=1,3,\ldots}^{n-1} f(x_i) * \frac{1}{n}$



Example: Solve Laplacian equation