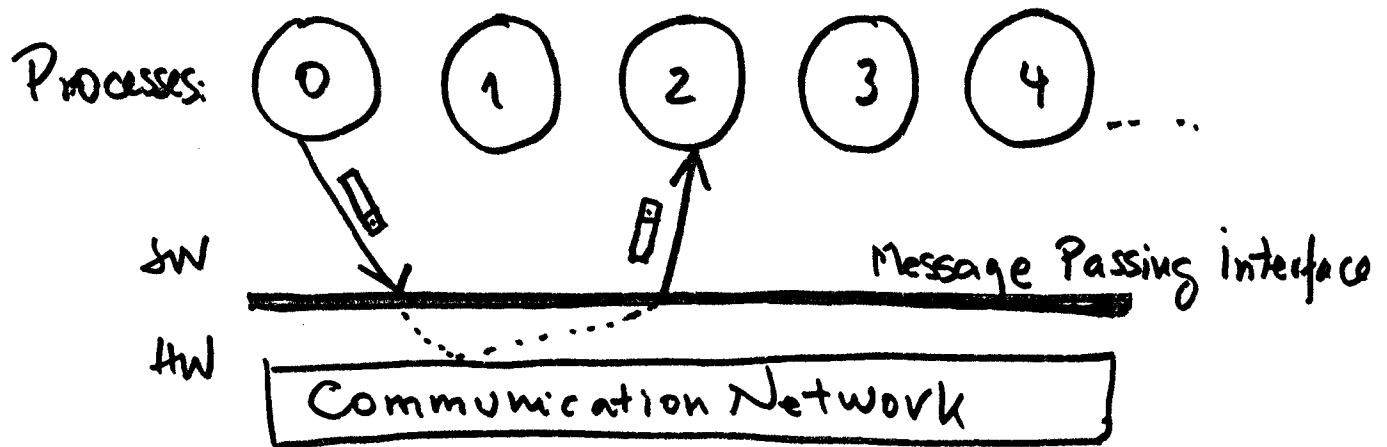


L6 - ECT 669 (not in textbook)

M P I = Message Passing Interface



# Message-Passing Programming Paradigm

- all variables are private
- processes communicate via subroutine calls
- typically written in a conventional sequential language (e.g., C, Fortran)
- Messages: "packets of data" moving between processes.

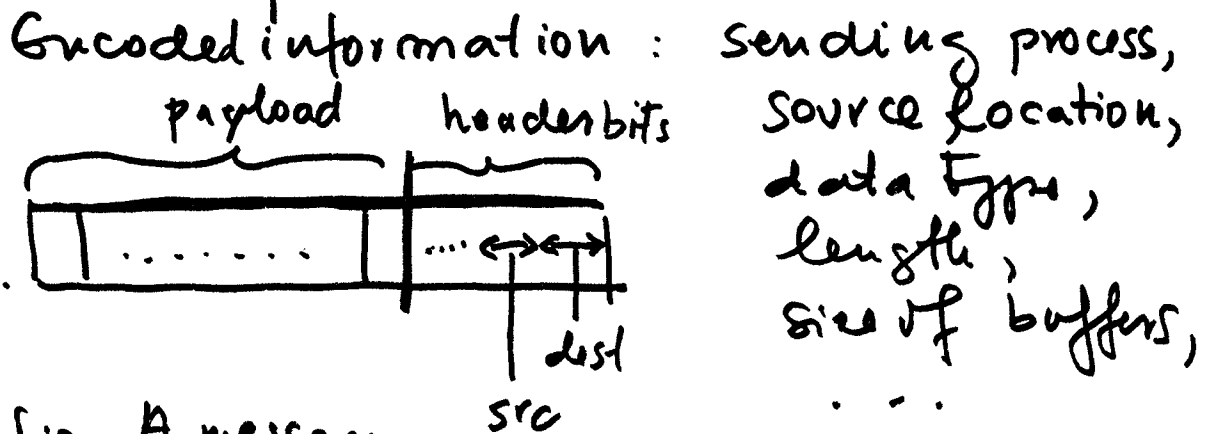


fig. A message

## MPI - Important Concepts

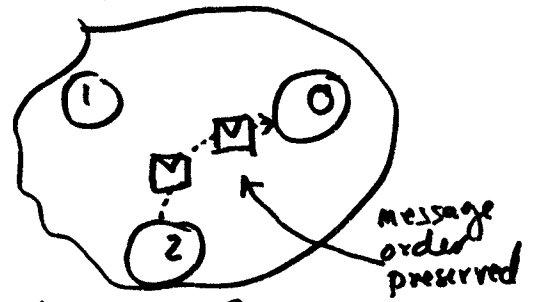
- Communicators ✓
- Point-to-Point Communication ✓
- Collective Communication ✓
- Virtual Topologies ✓

## Standard ?

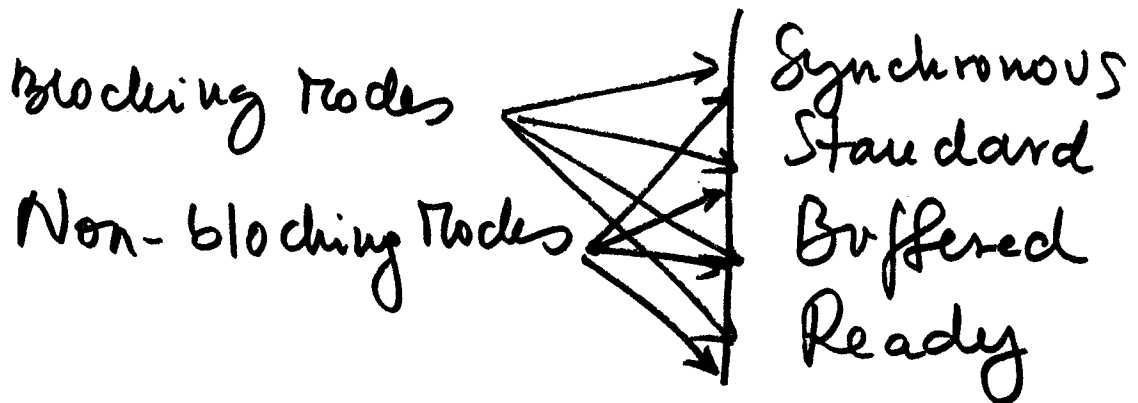
- Two years of proposal meeting
- Europe & US
- 60 people, 40 organisations
- First standard for message-passing
- "MPI document" produced

# Point-to-Point

- from process to process



- different semantics for send & receive(s)
  - when the operation terminates?
  - when can resources (e.g. buffers) be reused?

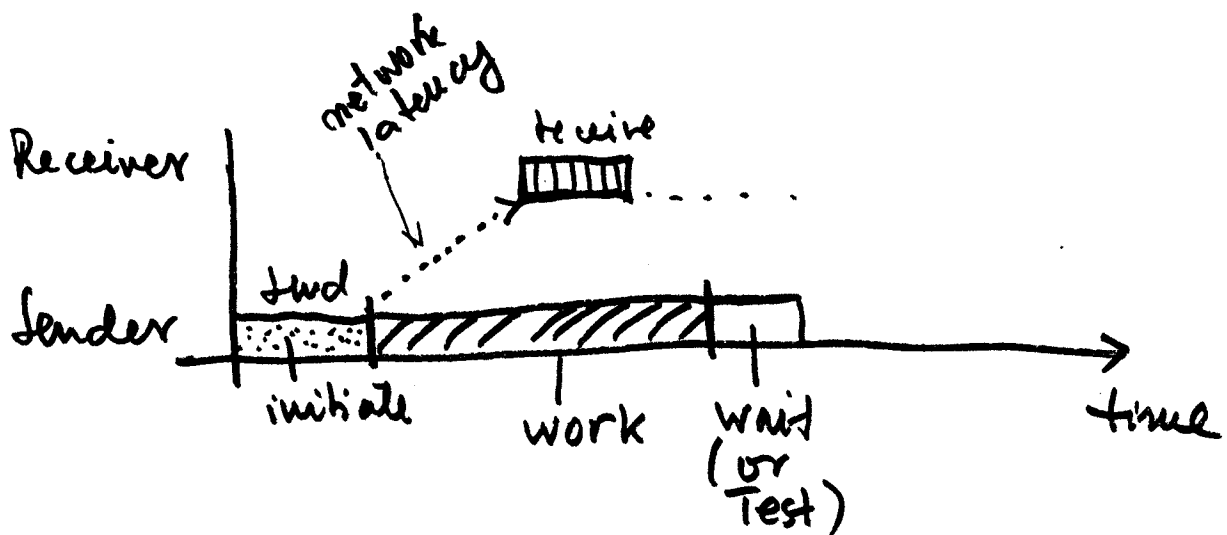


SEMANTICS?

# Non-Blocking Communication

\* Three phases

- 1, initiate communication
- 2, do some work
- 3, wait for non-blocking communication to complete



Example

Non-blocking Synchronous send

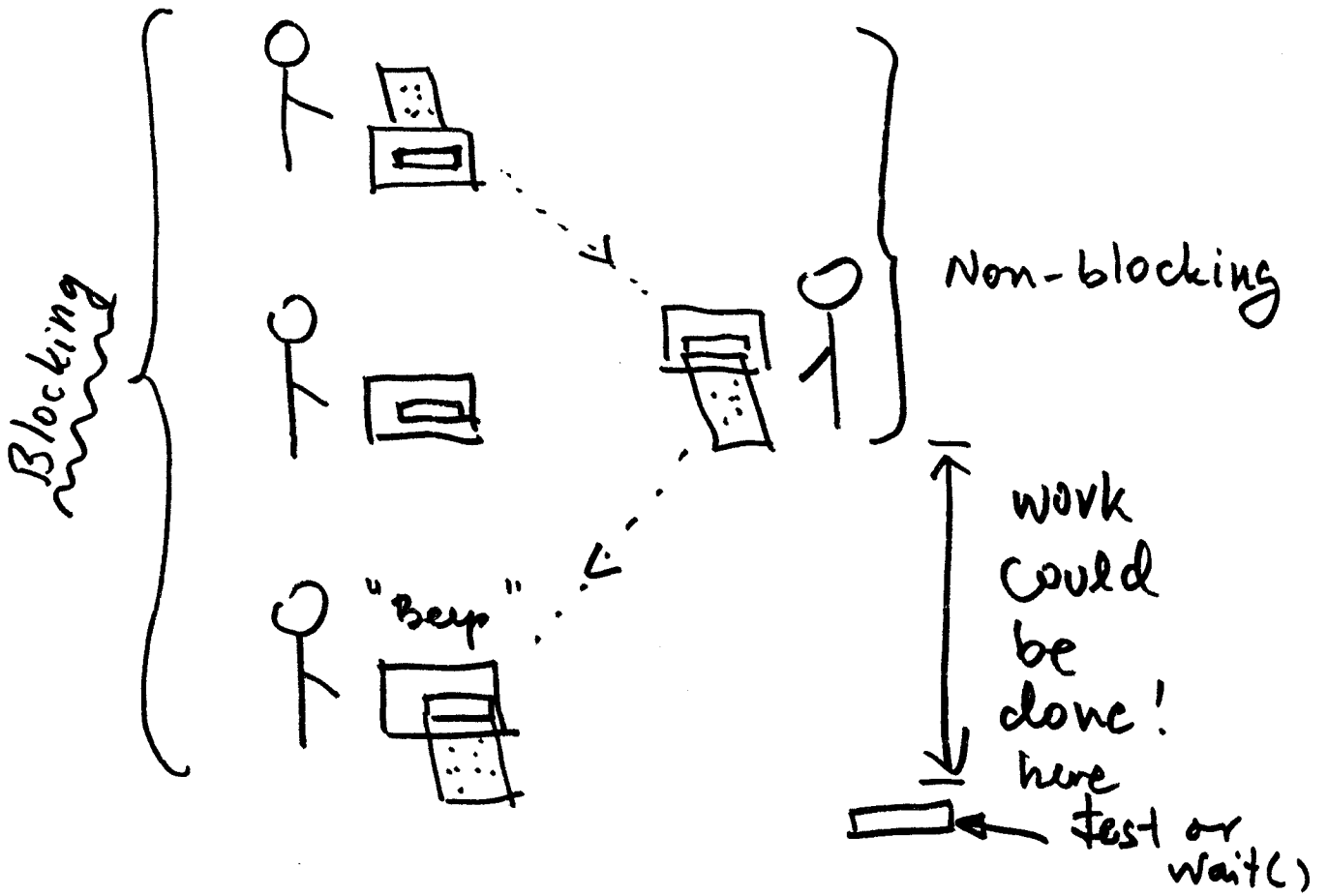
MPI\_Isend (buf, count, datatype, dest,  
tag, comm, handle)

MPI\_WAIT (handle, status, ierror)

MPI\_Irecv (buf, count, datatype, src,  
tag, comm, handle)

comm = Communicator = group of processes  
involved in  
communication.

# Synchronous feeds

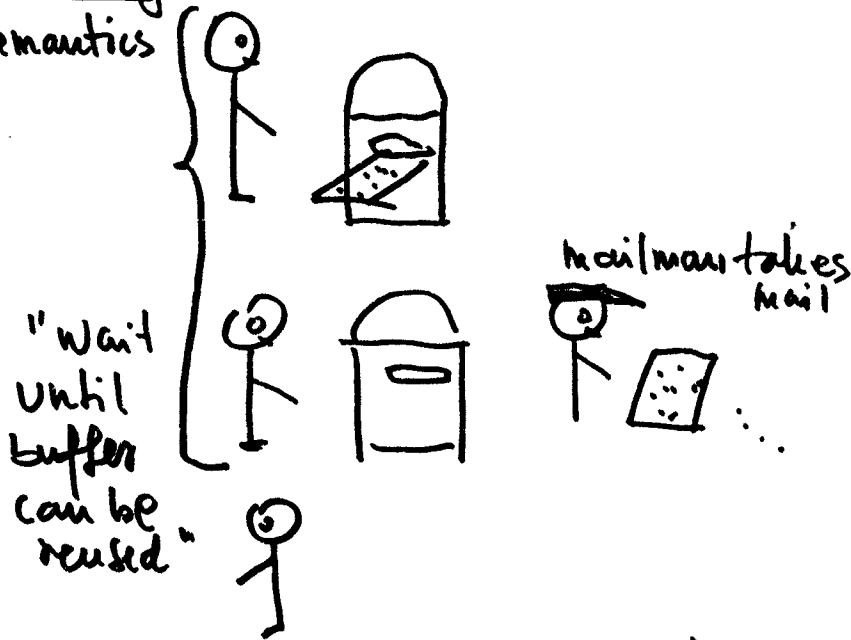


P7.

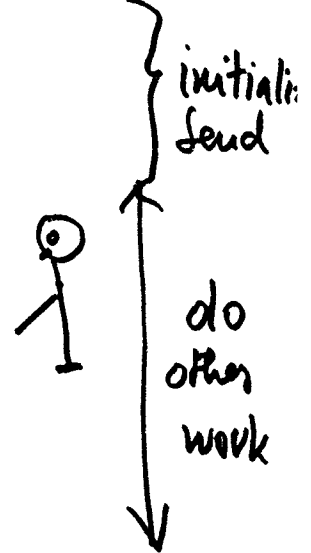
# Asynchronous Sends or Buffered

Analogy - just put it in Mailbox  
(- here is buffer!)

Blocking  
Semantics



Non-blocking



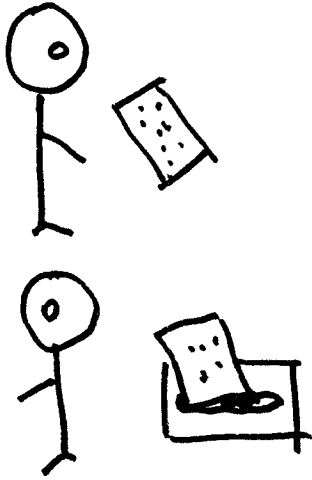
Note: independent of Receive!

## Standard sends in MPI

- either synchronous or buffered....
- depends on message size and resources available....
- can mean different things, i.e., implementation specific!
- if you expect certain behavior use synchronous or buffered modes!

## Ready Mode

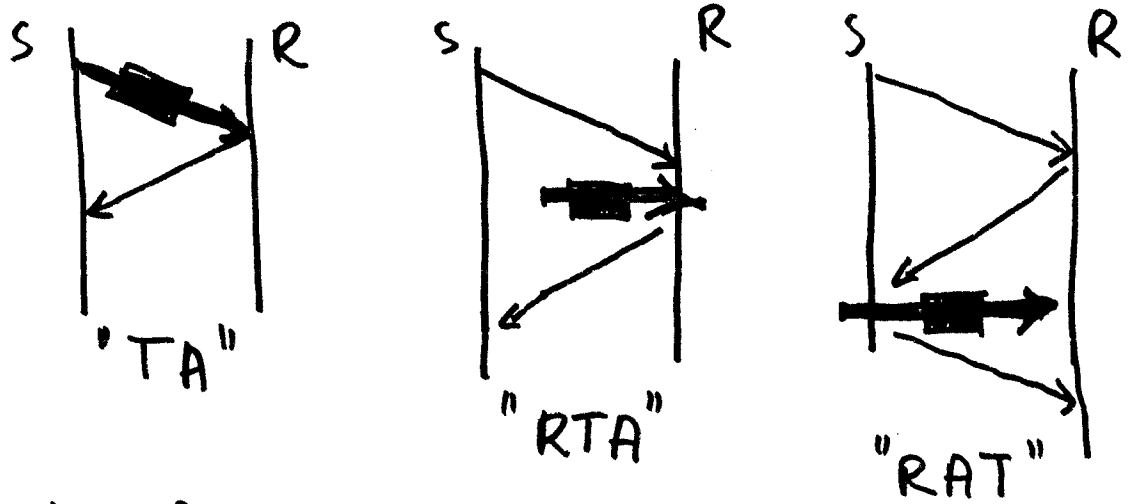
= immediate return!  
from MPI



- Not recommended!
- Useful when no other message is sent to a processor, and no information about resources/ completion time is needed!

Implementation issues:

Point-to-Point Data Transfer  
Protocols on CRAY T3D



Transfer - Acknowledge (TA)



Request - Transfer - Acknowledge (RTA)

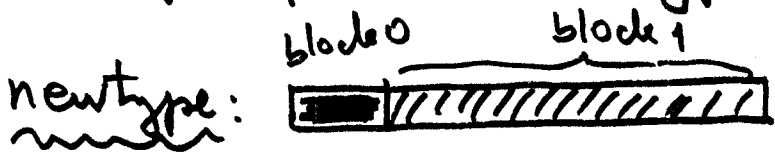
Request - Acknowledge - Transfer (RAT)

# Derived Data Types in MPI

Contiguous : (address, size)

Strided : (address, numFragments,  
fragment size,  
stride size)

Example of MPI Datatypes: } MPI\_INT:   
} MPI\_DOUBLE: 

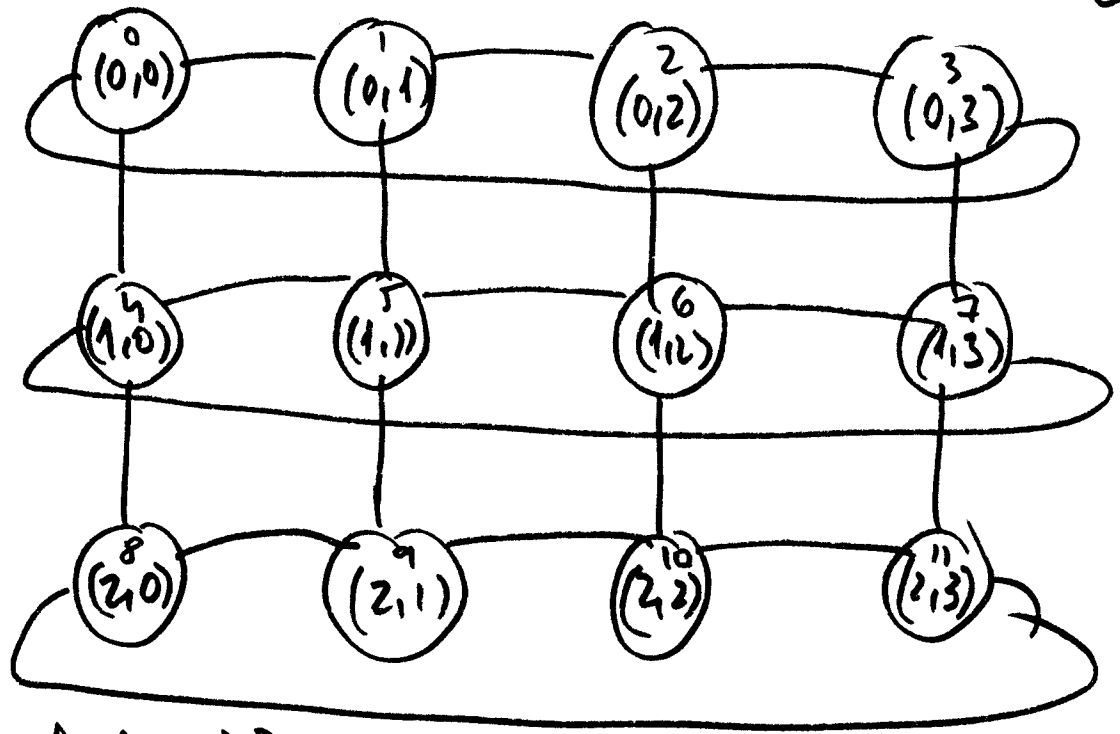


count = 2 ; array\_of\_blocklengths[0] = 1 ;  
array\_of\_types[0] = MPI\_INT ;  
array\_of\_types[1] = MPI\_DOUBLE ;

# Virtual Topologies

- Convenient process naming
- Simplifies writing of code
- Allows MPI to do optimizations
- Types
  - Cartesian
    - virtual grid
    - Processes id'd with cartesian coordinates
  - Graph (not covered here)
- How to use it?
  - Create a topology → new communicate
  - MPI mapping functions compute processor ranks
  - Partitioning?

Example - A cylinder  
- MPI provides the mapping



Rank of (2,1)?  $\xrightarrow{\text{MPI}}$  9

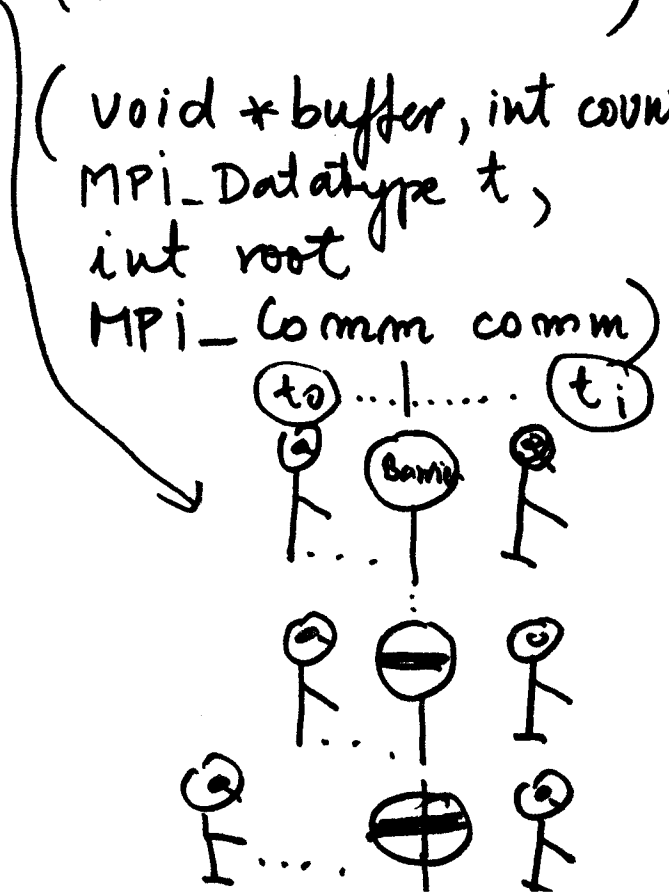
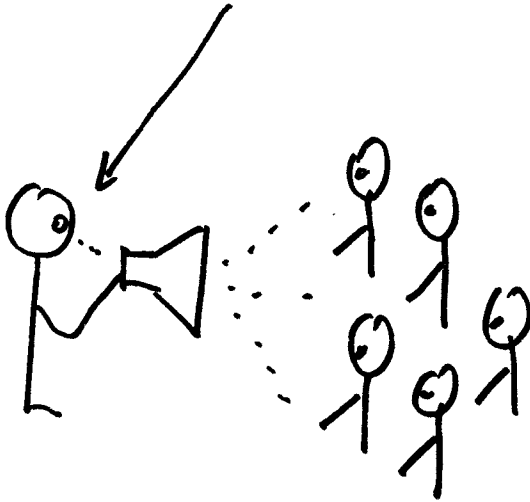
## Collective Communications

- Involving a group (a "communicator")
- Called by ALL in the communicator involved
- Types:
  - Barrier
  - Broadcast, scatter, gather
  - Global sum, etc
- All MPI collective communications are blocking!

# Example API:

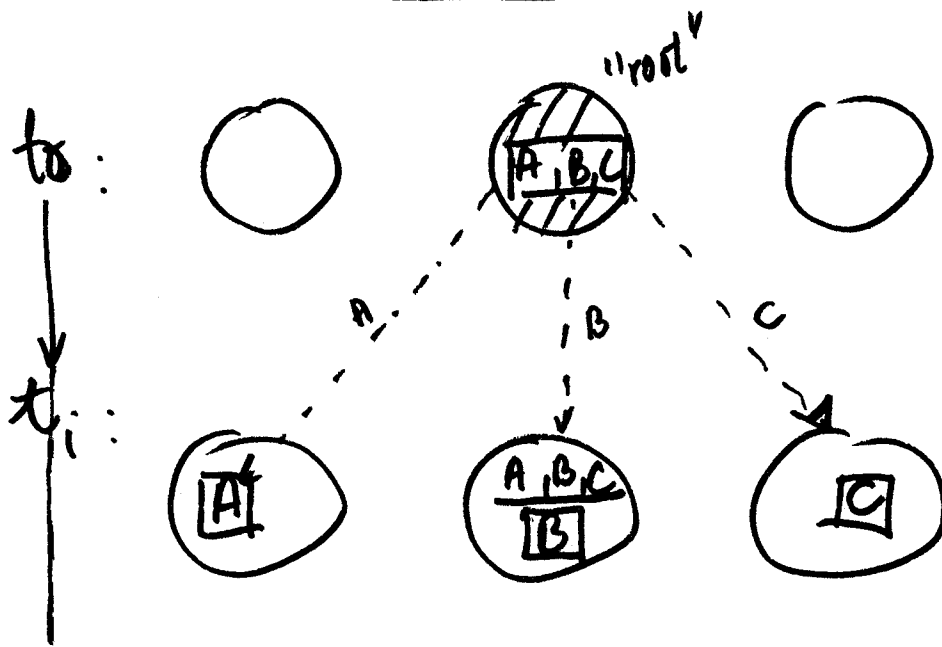
int MPI\_Barrier (MPI\_Comm comm)

int MPI\_Bcast (void \*buffer, int count,  
MPI\_Datatype t,  
int root,  
MPI\_Comm comm)

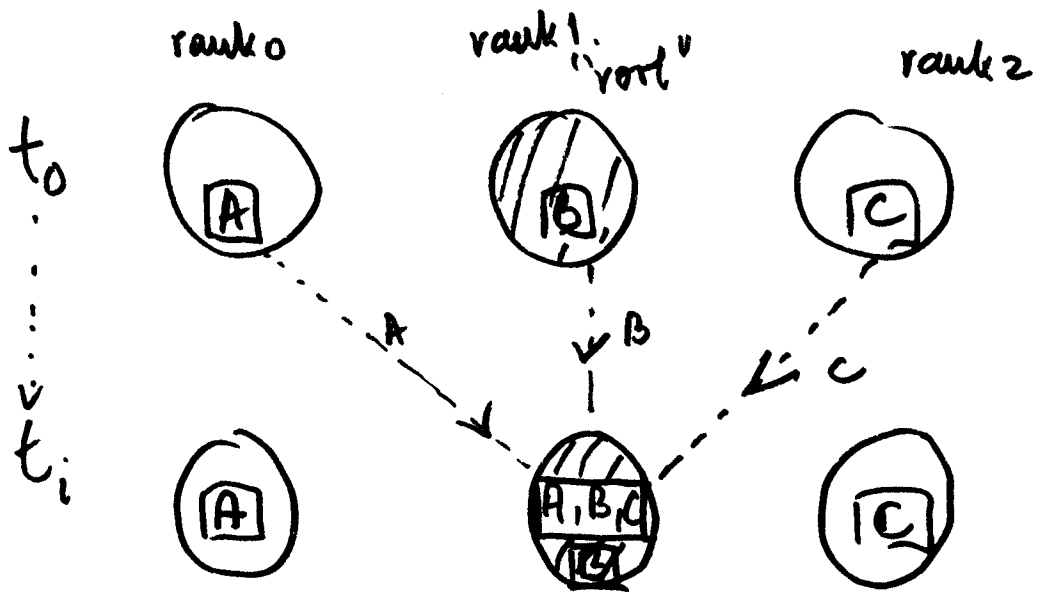


P16.

# Scatter



# Gather



# Global Reduction

MPI - Reduce

Rank

0

A, B, C, D

A, B, C, D

"ROOT"

1

E, F, G, H

E, F, G, H

A, E, G, H

2

I, J, K, L

I, J, K, L

3

M, N, O, P

M, N, O, P

Operations:

min,

max,

sum,

AND,

OR,

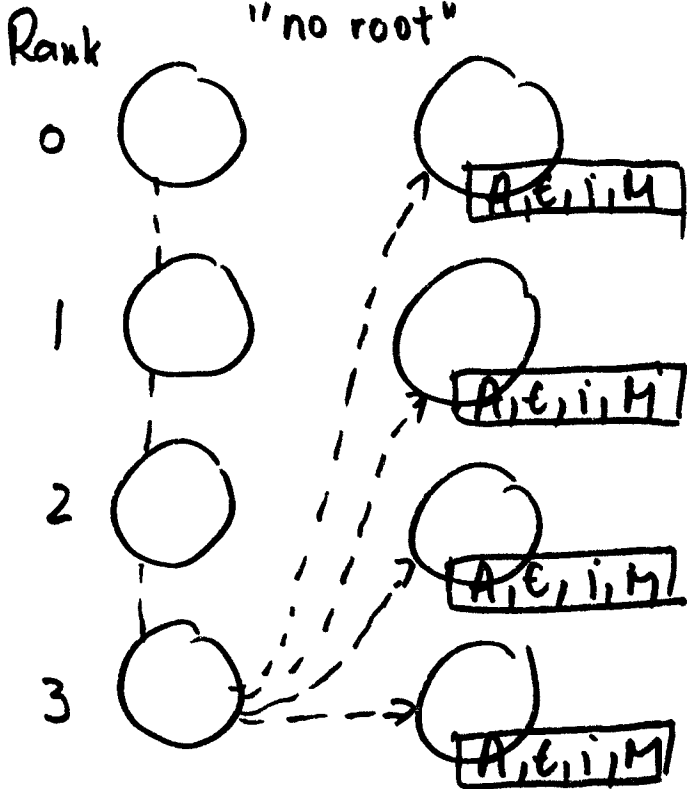
...

user defined!

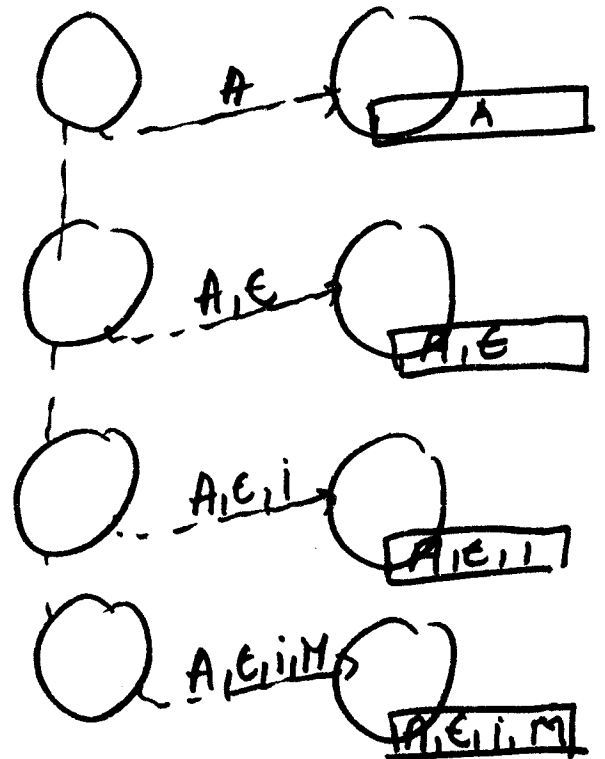
P.19.

Variants (Same example)

MPI-ALLREDUCE  
"no root"



MPI-SCAN  
"parallel prefix"



## Problem for home

- ① Rewrite the "equation solver" from textbook in MPI
- ② Rewrite the "matrix multiply" with MPI

Note: you will need the MPI manual for this

P21.