CSE 531: Distributed and Multiprocessor Operating Systems

Class: 14 Date: 10/12

Fall 2020

Note Title

Parallel Programming

- Application level
- Speedup / Engineering,
- Big Data → Map Reduce Hadoop]

Parallel program

- Low level → [thread creation, Semaphores / Locks, Monitors]

- High Level Language based

  - OpenMP
  - MPI
  - Linda
  - → Barriers
  - → [fork / join] PAR

Open MP

- programming language
- par construct (fork-join)
- barrier synch

→ embarassingly parallel applications

↳ speedup / scalability

Speedup

- N processors
- Seq. time = T
- parallel time → T/N

[Coarse grain]

Scalability

→ increase N

decrease total time

Open MP

Parallel proc on shared mem MP
(UMA)

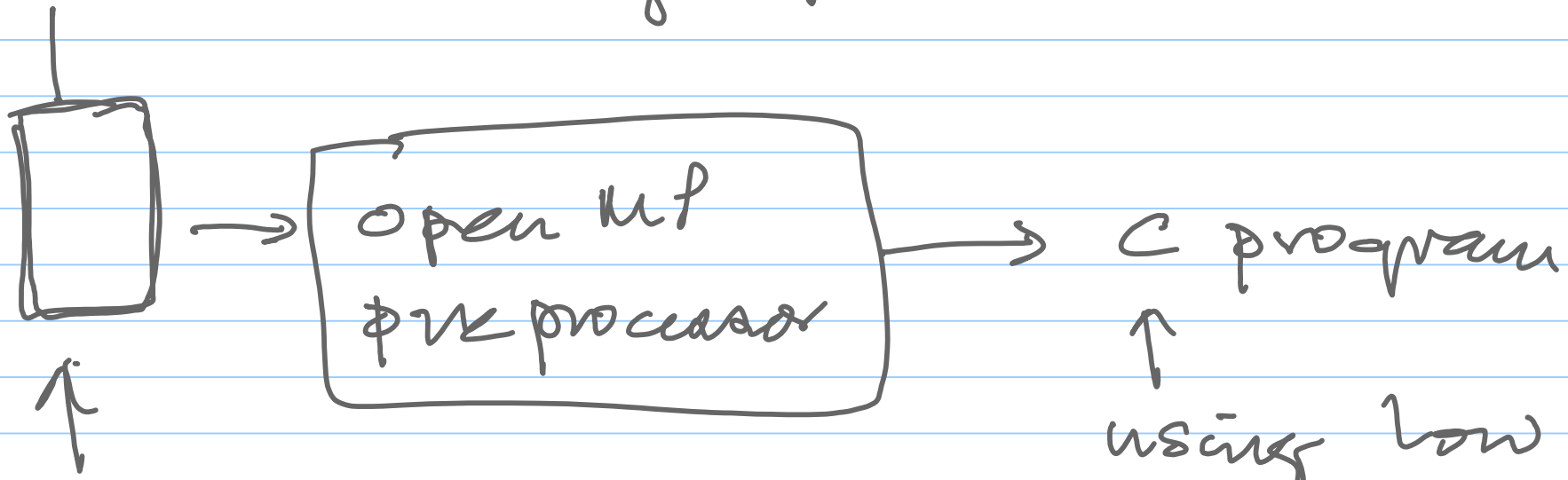
↓

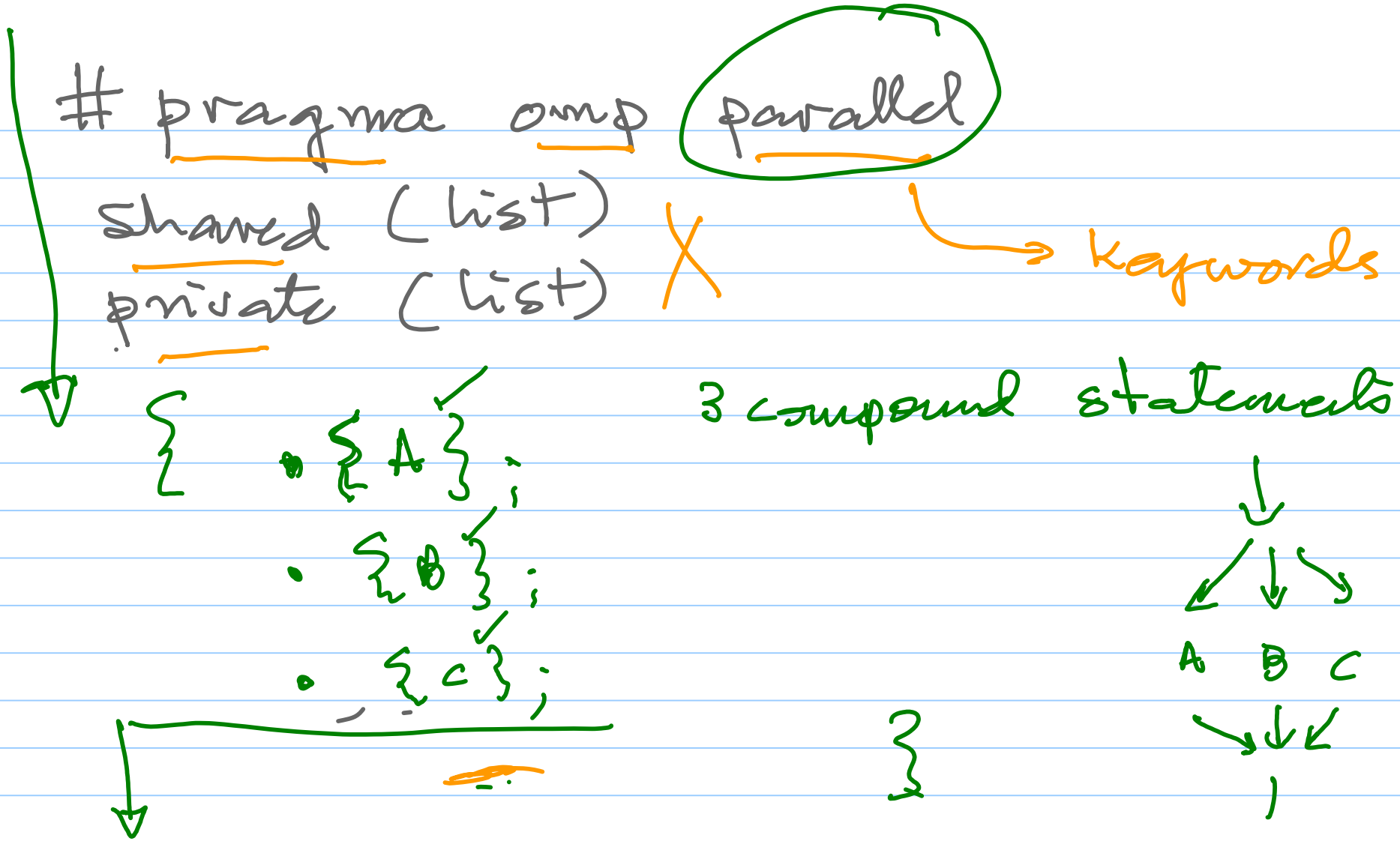
add on to a base language
C

↓

add keywords → pragma, etc

- write seq code in C
- parallelize using Open MP




pragma omp parallel
shared (list)
private (list)
keywords
{
• {A};
• {B};
• {C};
}
3 compound statements
A B C

pragma omp parallel for

```
{ for ( i=0, j< w, i++)
    {
    }
}
```

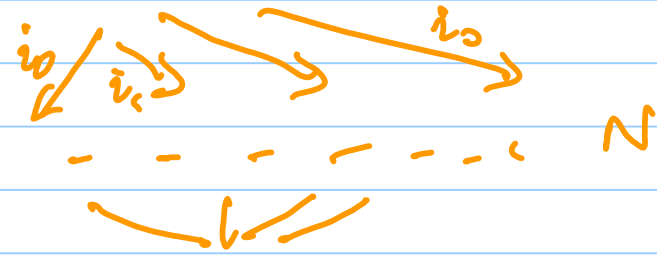
executed first → get set of values of i

Sequential for loop

Values of i

i_0 i_1 i_2 - - - i_N

- start N threads each with a diff value of i
- run all in parallel → N threads in parallel
- join them



#pragma omp barrier

↳ use inside the other pragmas

parallel for

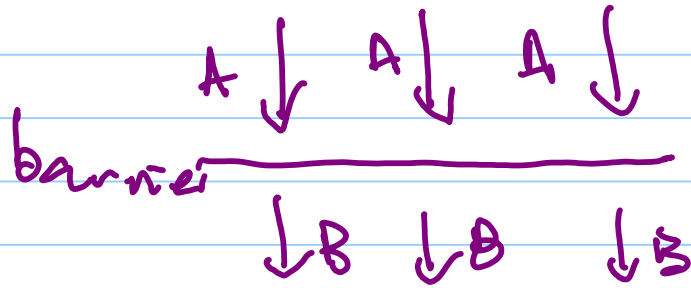
→ for ( )

{ ≡ A

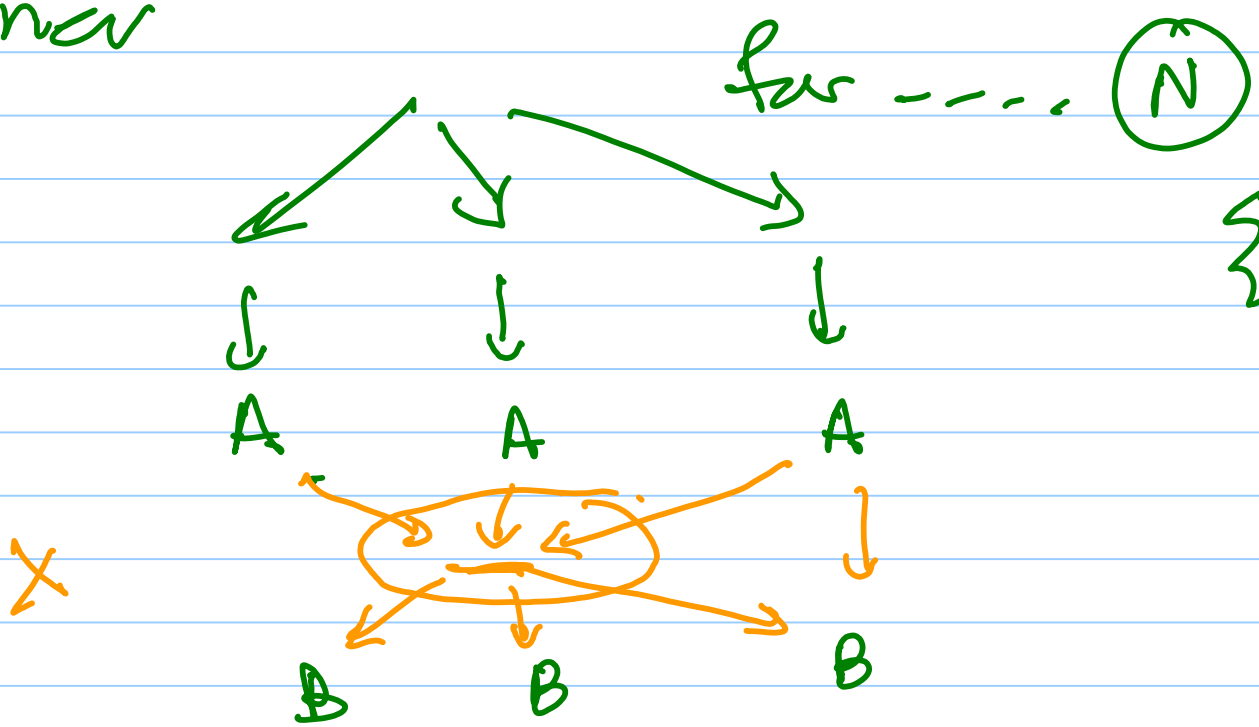
#pragma barrier

≡ B

}



Barrier



{ = Code A

barrier

= Code B

PVM → parallel Virtual Machine

OLD

↓ clusters

replaced by MPI

→ Message Passing Interface

MPI → no shared memory

→ cluster of processors

- can be network — of computers

- can be shared memory machine

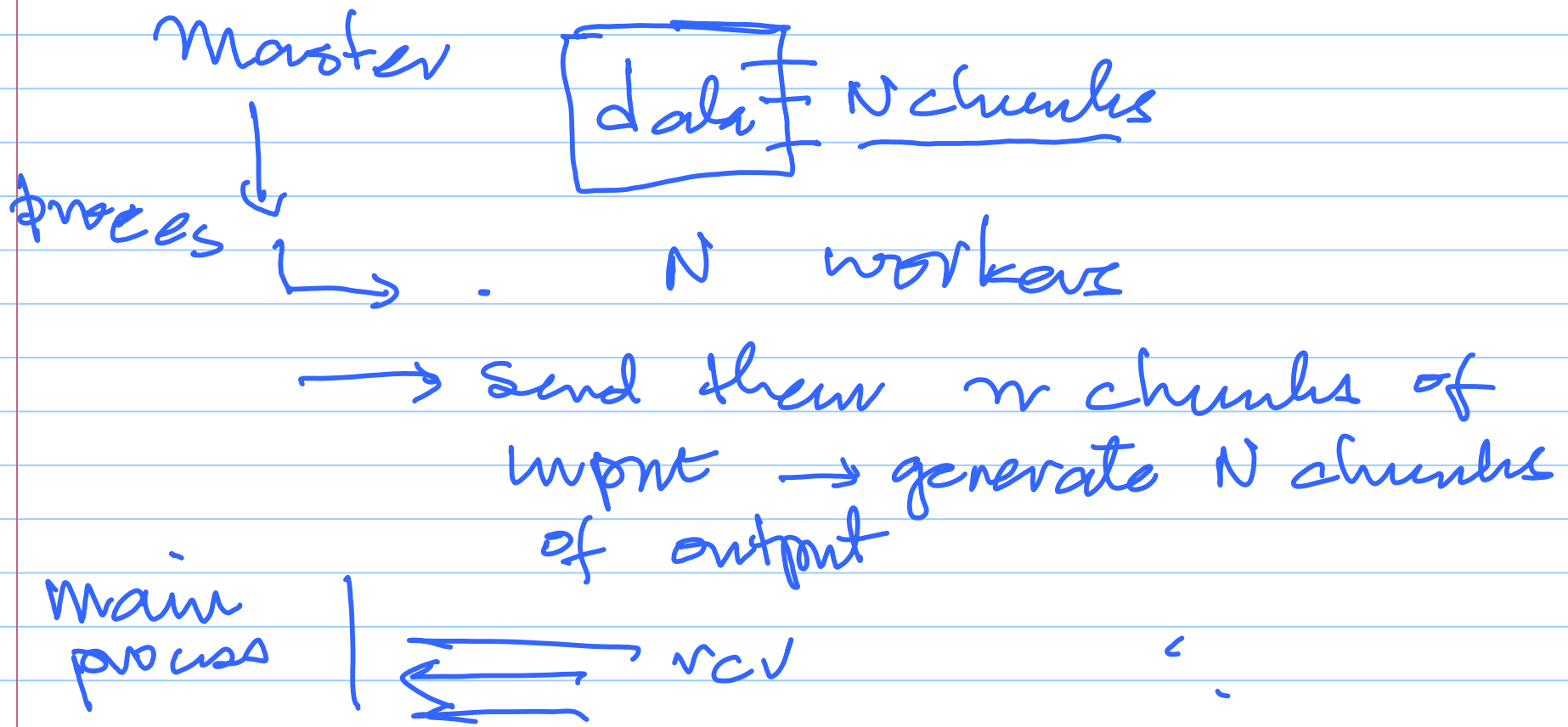
↳ process

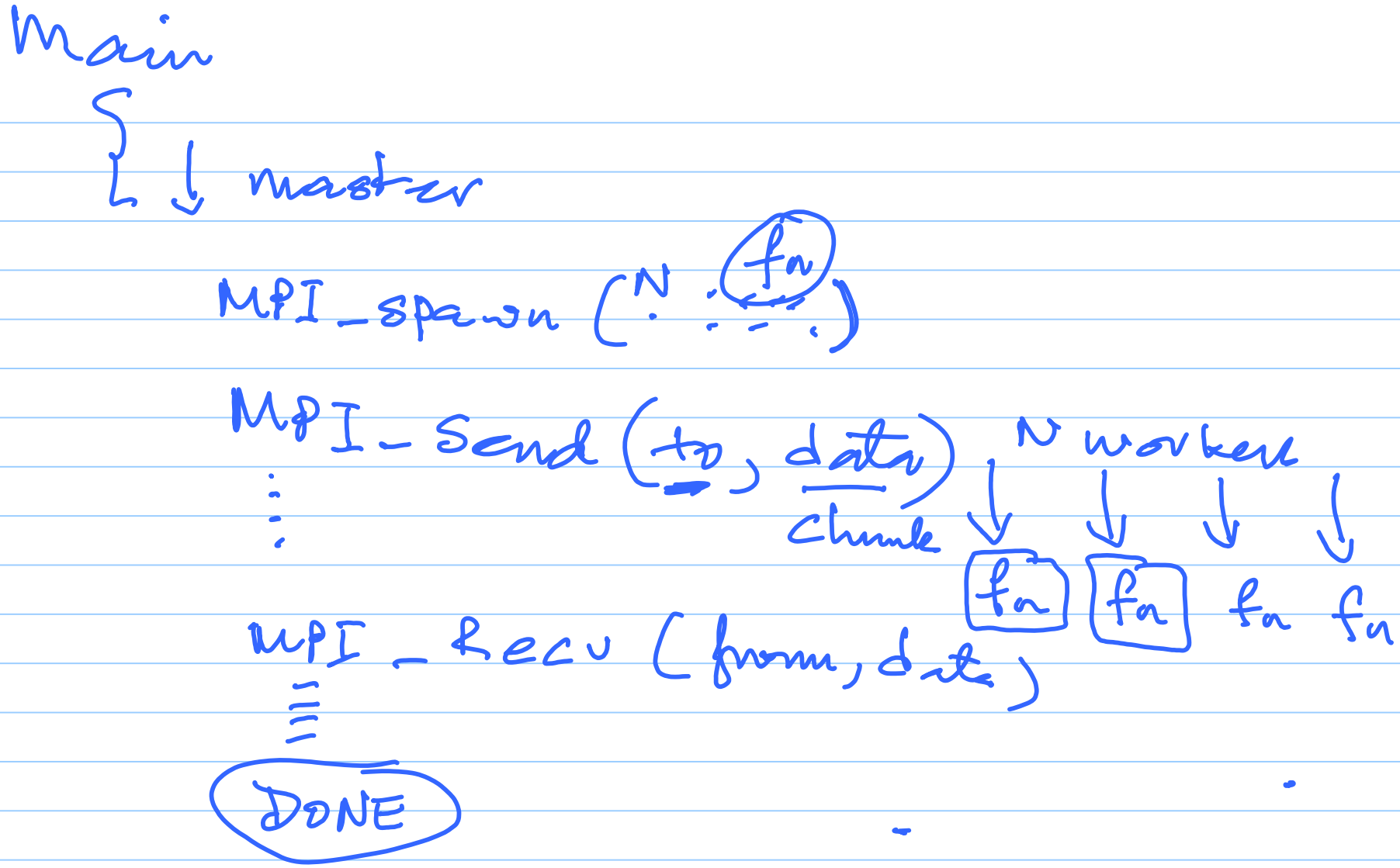
program ⟹ N processes ⟹ → end

↓

run anywhere

master - worker model




main
master
MPI_spawn (N : fn)
MPI_Send (to, data)
chunks
N workers
fn
fn
fn
fn
MPI_Recv (from, data)
DONE

→ MPI_Spawn → ↓ ↓ ↓

fn

MPI_scatter

MPI_gather

done

break up data
and send to N

Worker

```
{ MPI_ Rcv (data)
   fn
   MP_ Send (result)
}
```

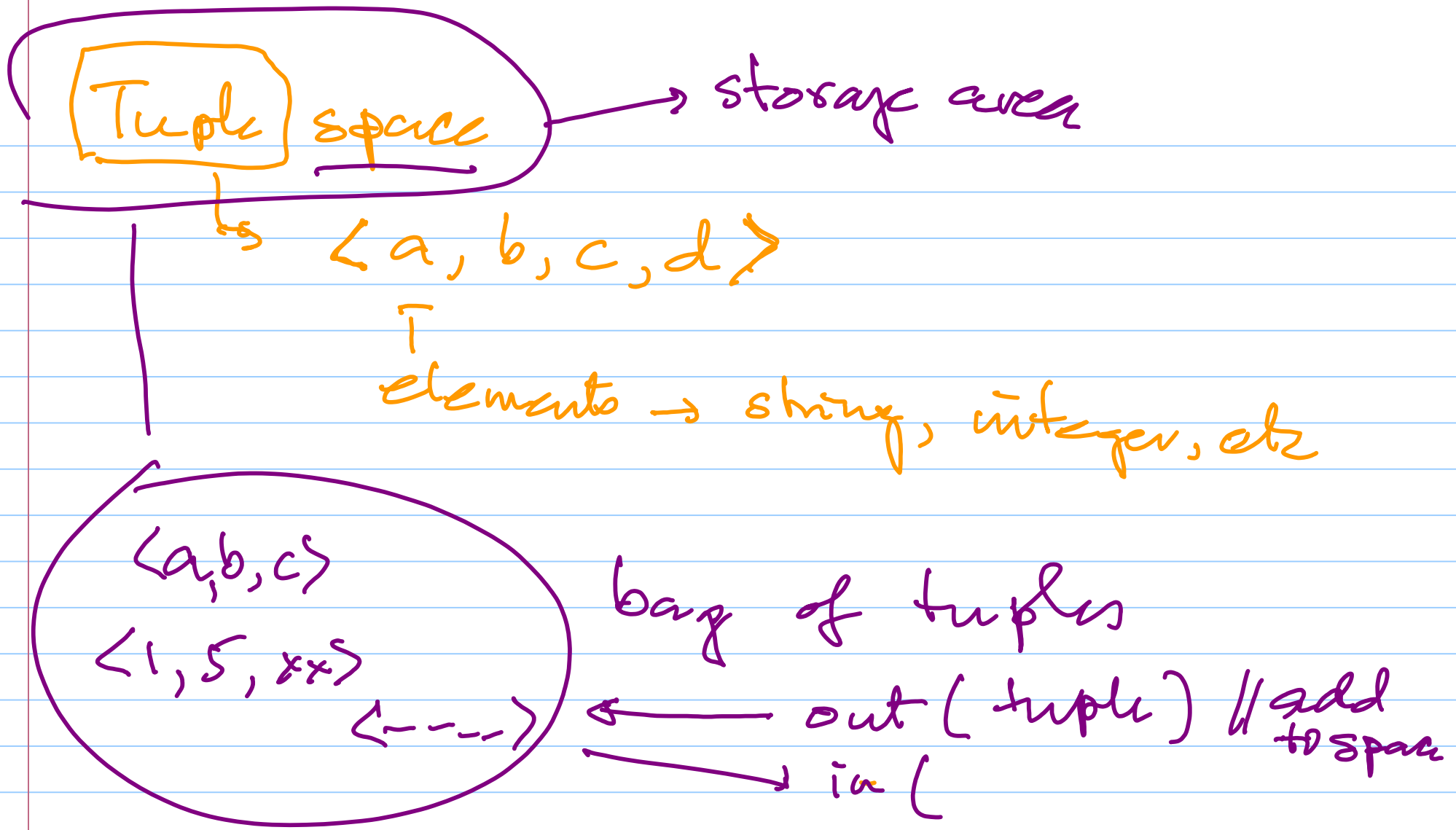
Linda → prog Language

→ data base

→ distributed tuple space

↓

parallel processing on dist machines


Tuple space
storage area
<a, b, c, d>
elements → string, integer, etc
<a,b,c>
<1, 5, xx>
bag of tuples
out (tuple) // add to space
in (

out (tuple) → add

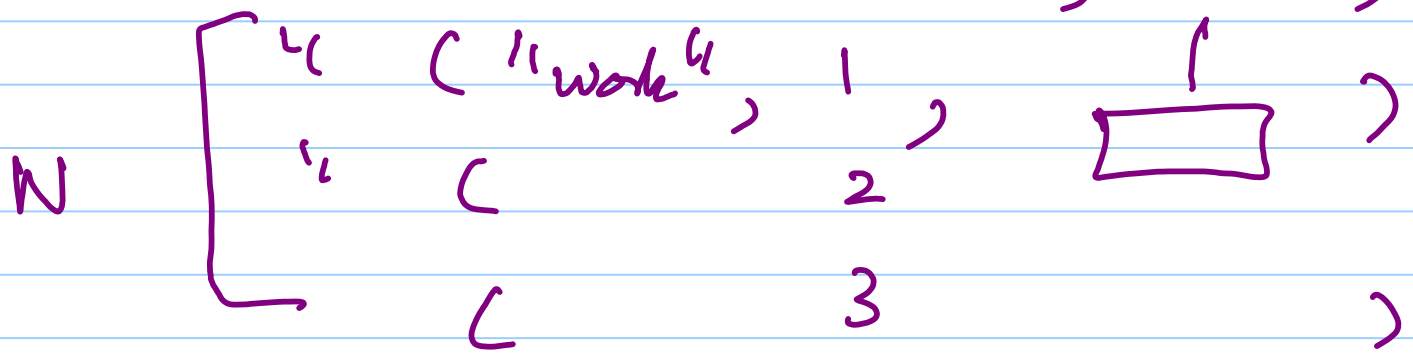
in (tuple) → delete

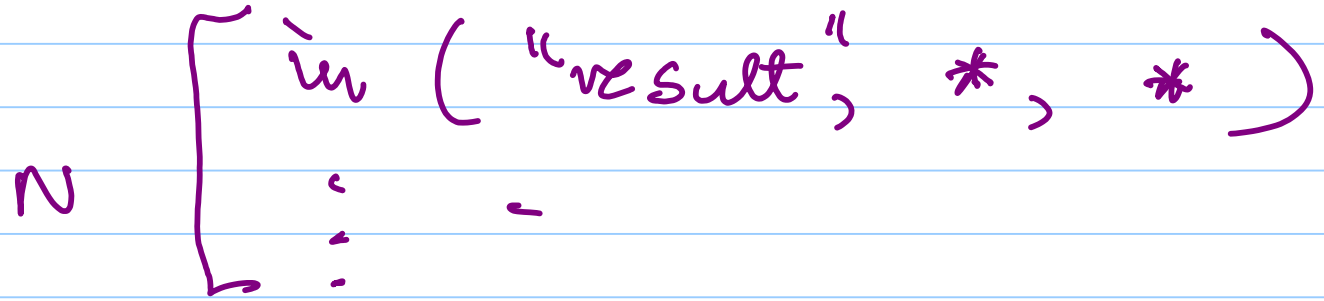
in⟨ 1, 2, 3 ⟩ → delete (1, 2, 3)

in⟨ (*), 2, 3 ⟩ → returned

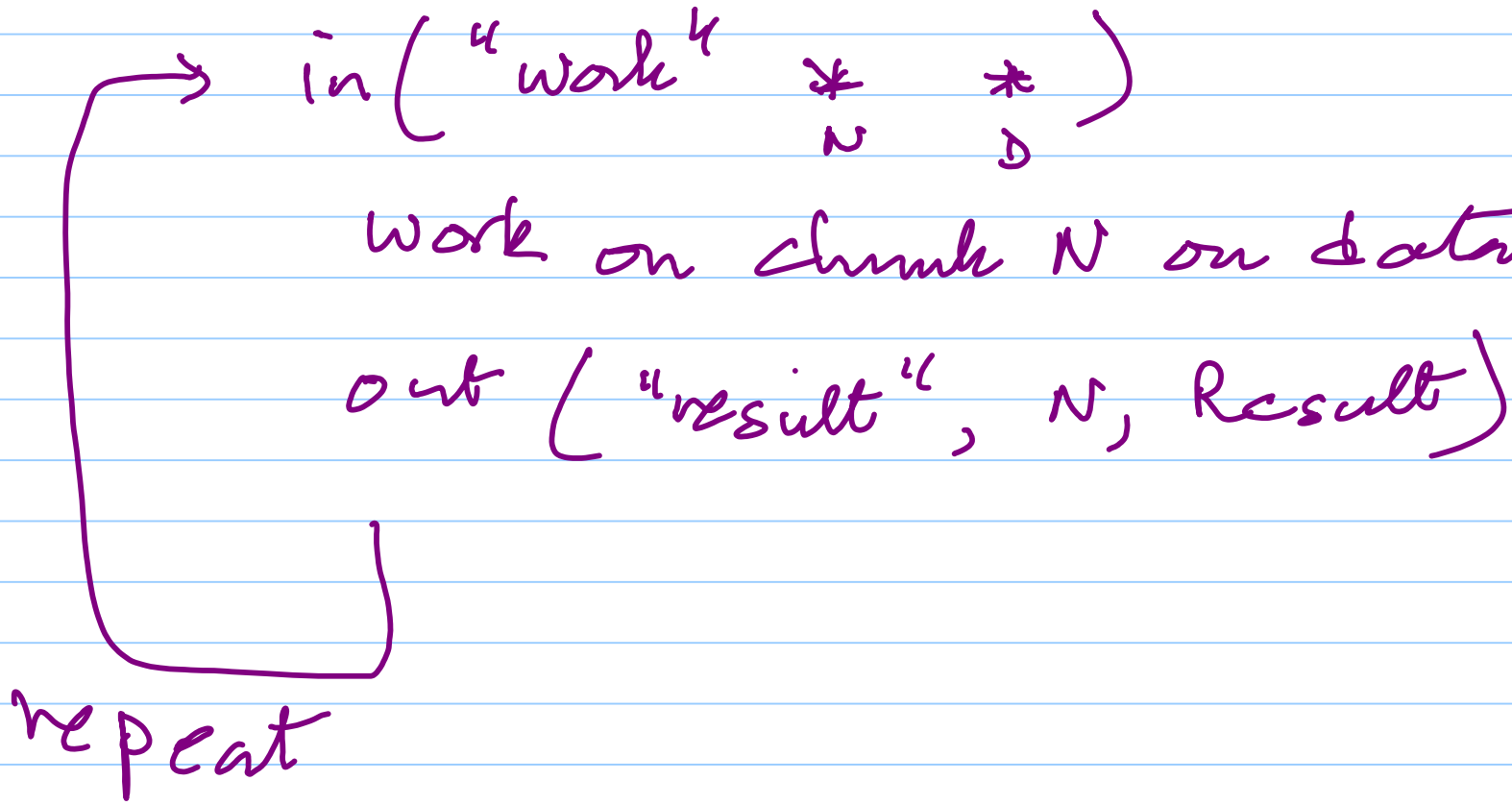
Parallel proc MASTER

→ out("work", chunk#, chunk)

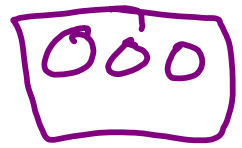




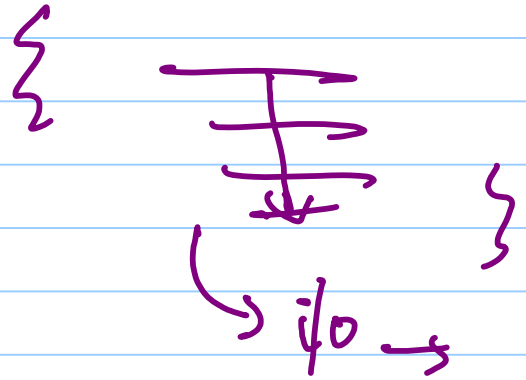
Worker



Open MP



shared mem



Master does I/O


Nucleus