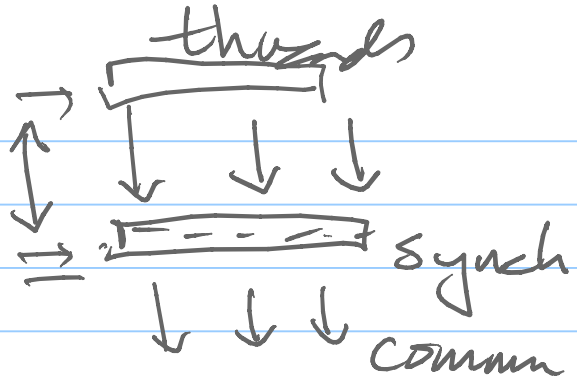
Parallel Processing —

Low level — [semaphores, monitors
threads, locks

High level

[language based
— Java
Open MP
MPI - ,

- fine grain parallelism
  - ↳ small computations / time taken



- Coarse grain parallelism
  - ↳ long computations

grain [time to compute
      " " synchronize / communicate

int

array → 4 elements] A, B, C

C = A + B

4 threads or sequential

Open MP

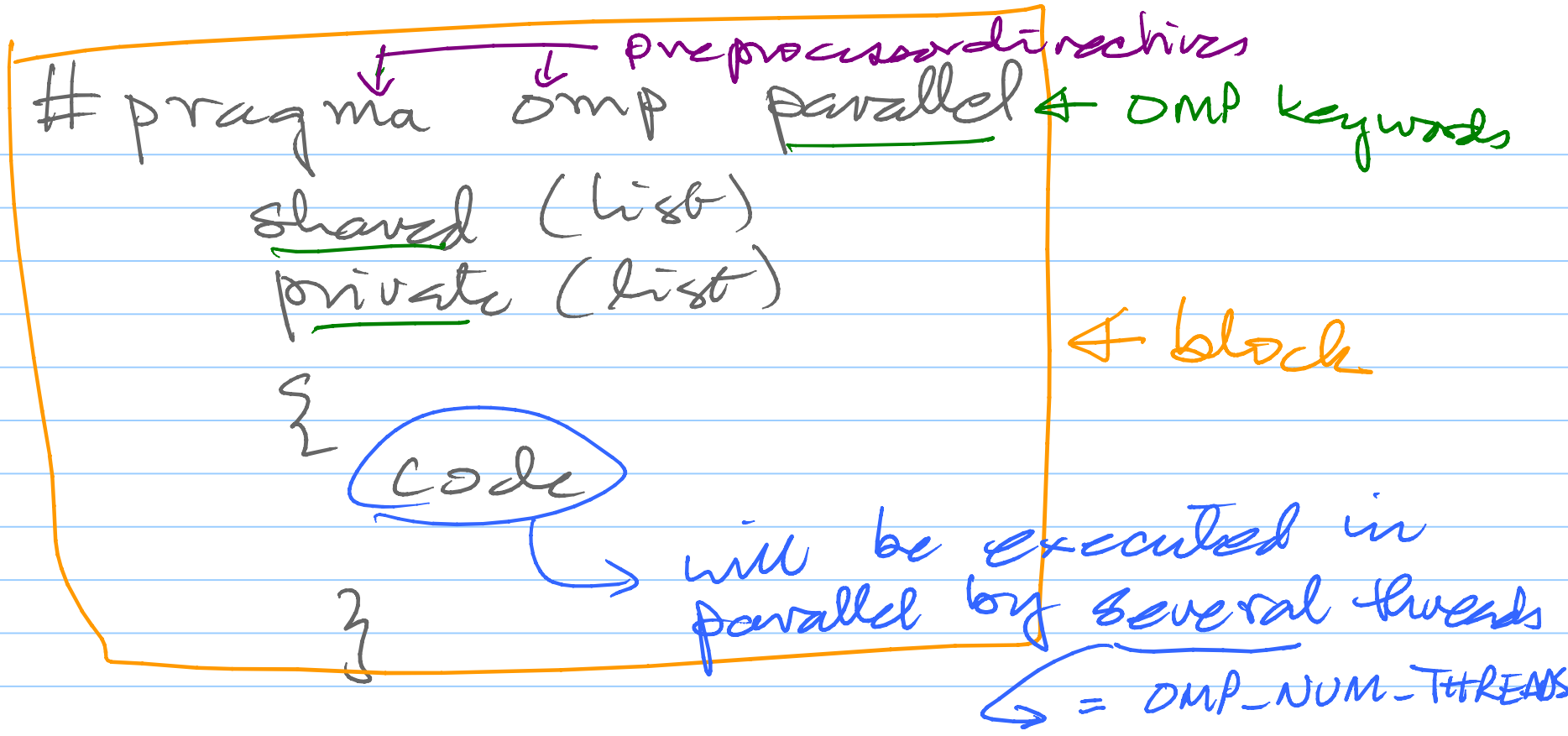
→ shared memory MP

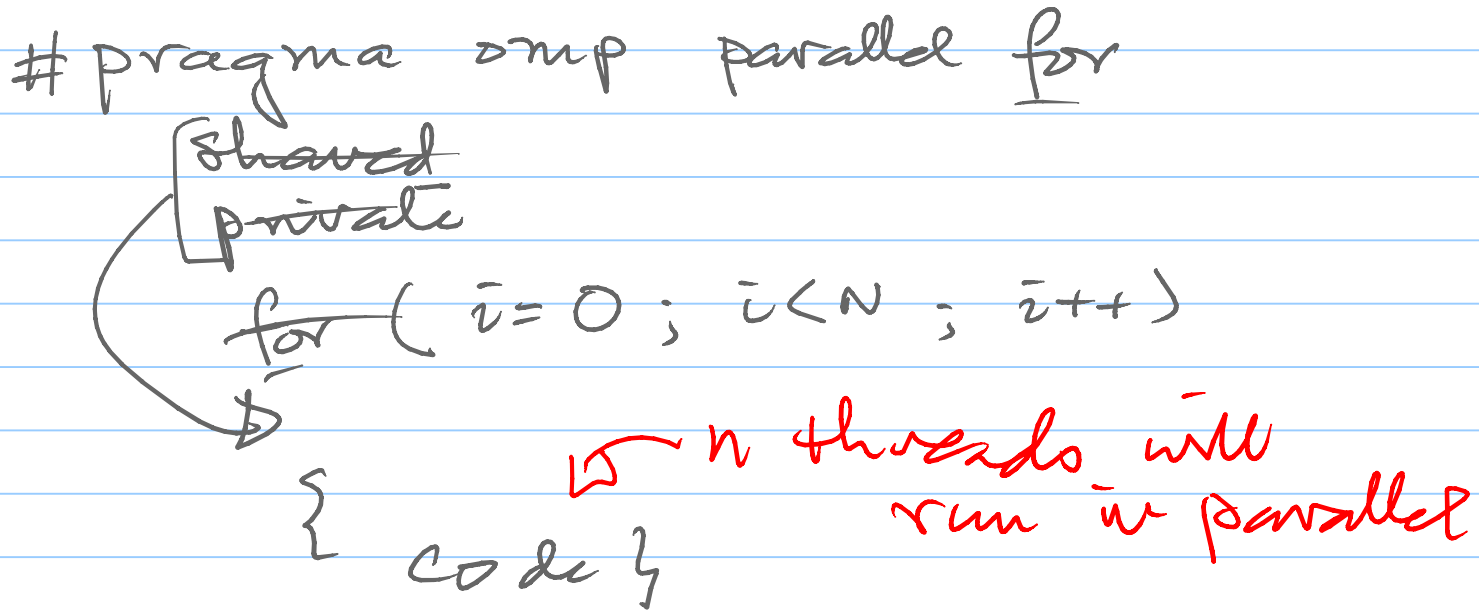
→ uses threads (implicitely)

& globals & locals

+ structured language

→ interfaces with C, C++ & others


preprocessor directives
#pragma omp parallel
OMP keywords
shared (list)
private (list)
block
{
code
}
will be executed in parallel by several threads
= OMP_NUM_THREADS


#pragma omp parallel for
shared
private
for (i=0 ; i<N ; i++)
{ code }
n threads will run in parallel

Open MP has other feature

barrier ⟶ #pragma omp barrier

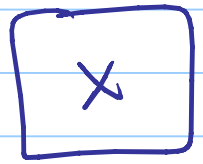
critical

atomic

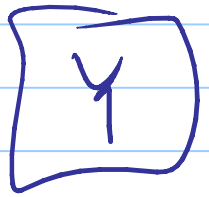
etc

[same # of threads

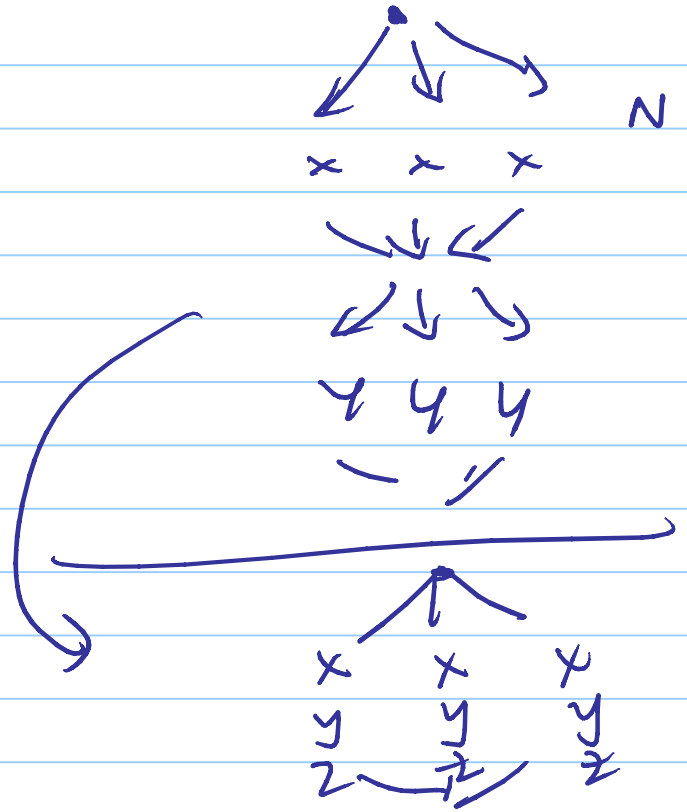
parallel for



parallel for



)) for



```
parallel for
{
    x
        barrier
    y
        barrier
    z
}
```

parallel programs

→ easy to subdivide computations into separate (non-conflicting) chunks → no need to lock

→ use barriers

→ "embarrasingly parallel"

PVM / MPI

↳ parallel virtual machine

↳ MPI

↳ message passing Interface

"standard"

MPI

- multiple processes
  - communicate via messages
  - master - worker architecture
    - (1)
    - (N) → fixed
  - environment variables (read only globals)
  - communication groups

MPI_Spawn

MPI_send

MPI_recv

MPI_scatter

MPI_gather

C program

main()
{ MPI_Spawn
(fn)

```
fn( )
{          <-- children
}

main()
{ [MPI_spawn (fn)

    main code parent

}
```

parent

IDs returned by MPI_spawn

MPI_send (dest1, data1)

(dest2, data2)

send

MPI_recv (data)

(data)

(data)

worker

{ MPI_rcv()

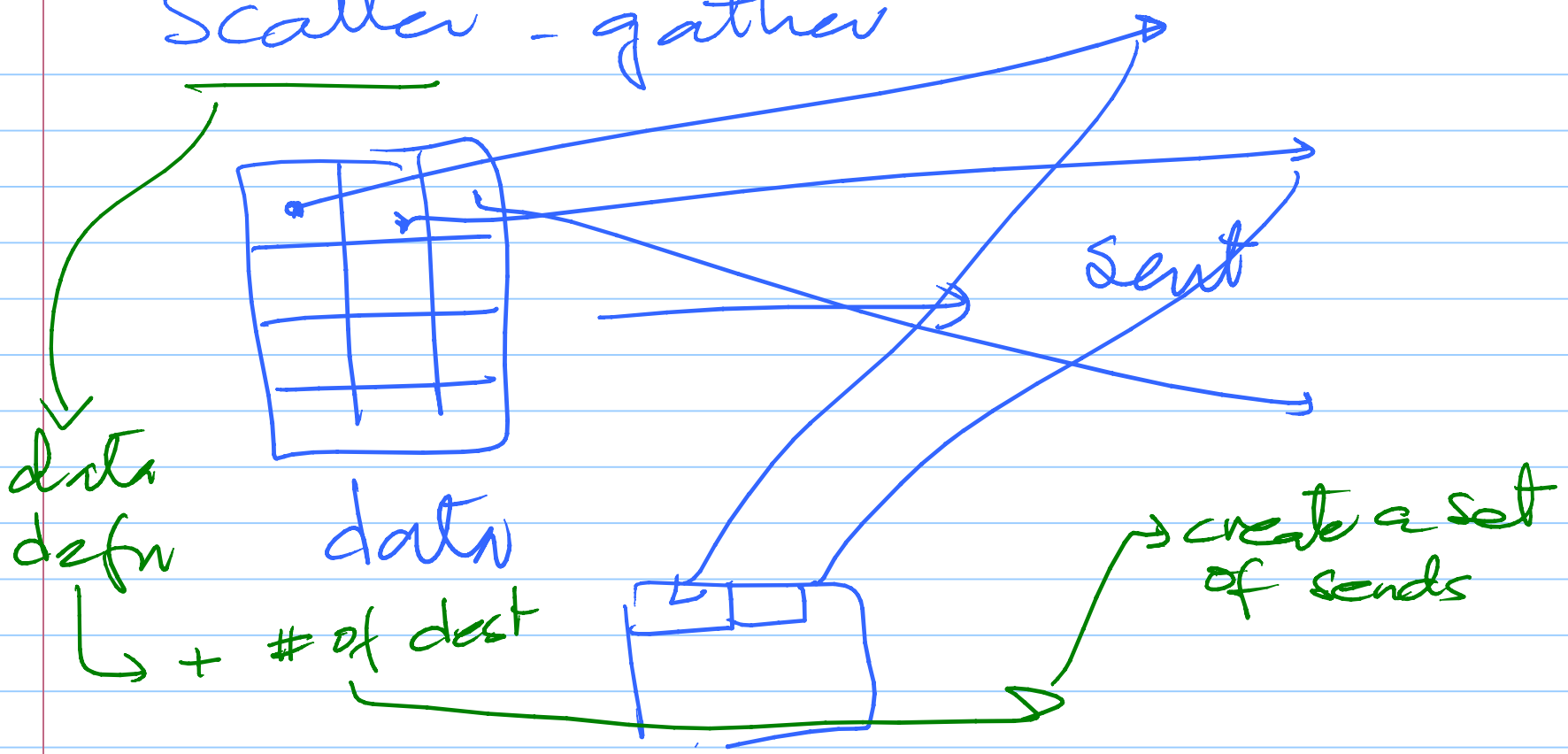
Compute

MPI_send (result)

spawn, send, rcv

- uniproc
- multicore
- cluster of machines (network)
  - UNI
  - multi

Scatter - gather

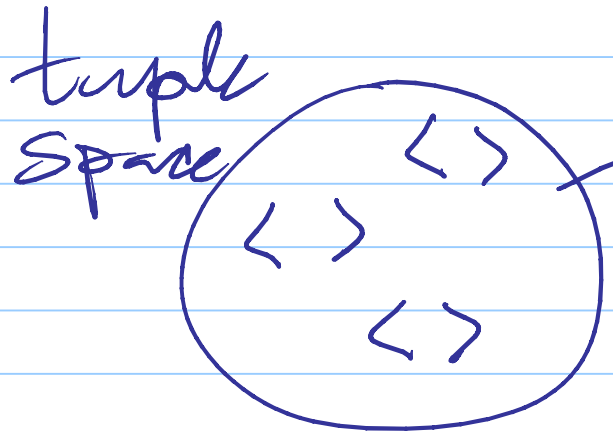


Linda

Variable # of items

tuple – < a, b, c, d, ... >

type of each item – int string struct



containing tuples

persistent

operations

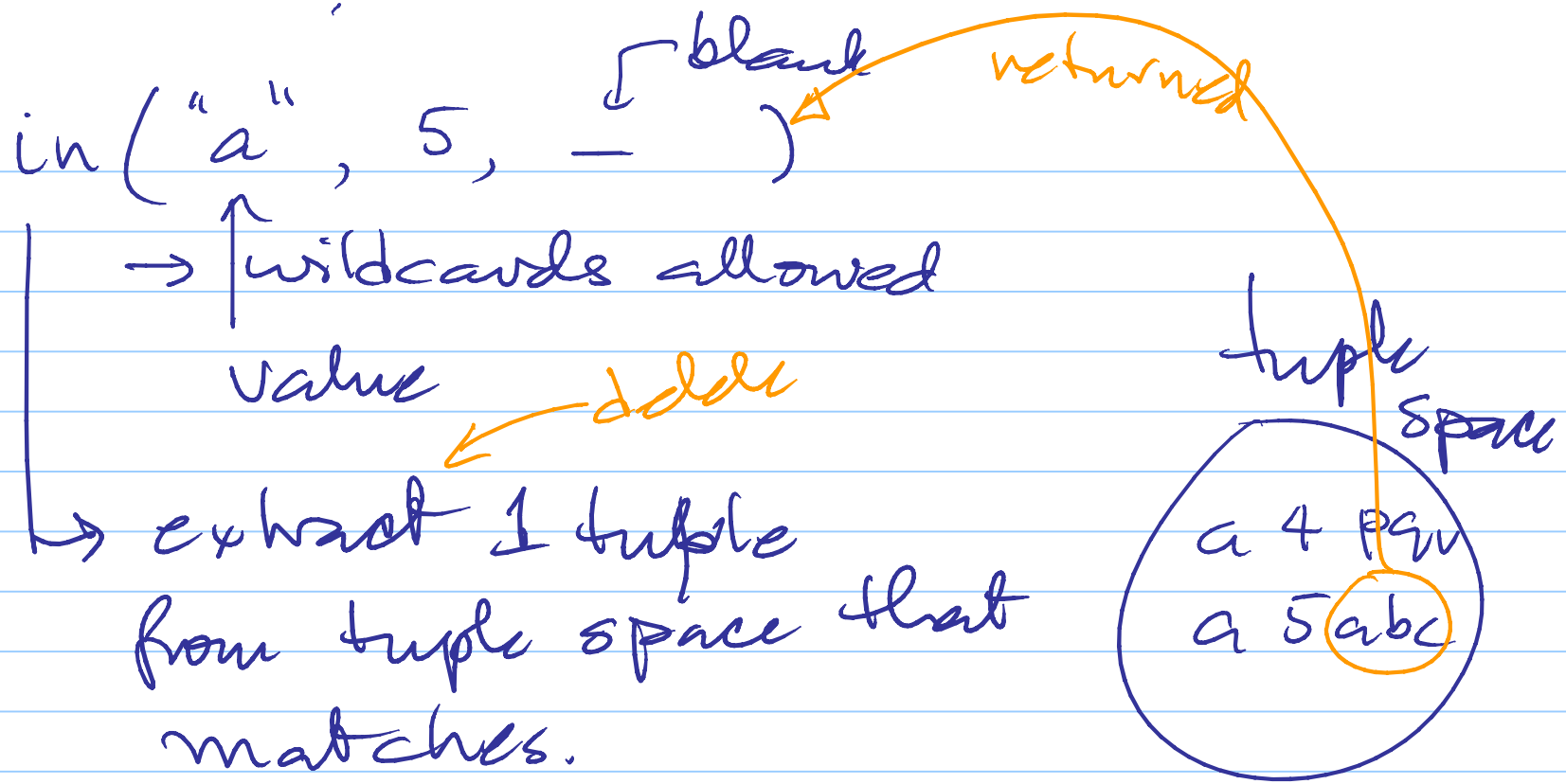
values

out("a", 1, "bcd")

# of tuples

add to tuple space

<a, 1, bcd>


in("a", 5, _)
blank
returned
wildcards allowed
value
added
tuple space
a 4 pqr
a 5 abc
exhaust 1 tuple from tuple space that matches.

in → if multiple matches
↳ any 1
if no matches → block
→ wait till a match
happens

atomic

master

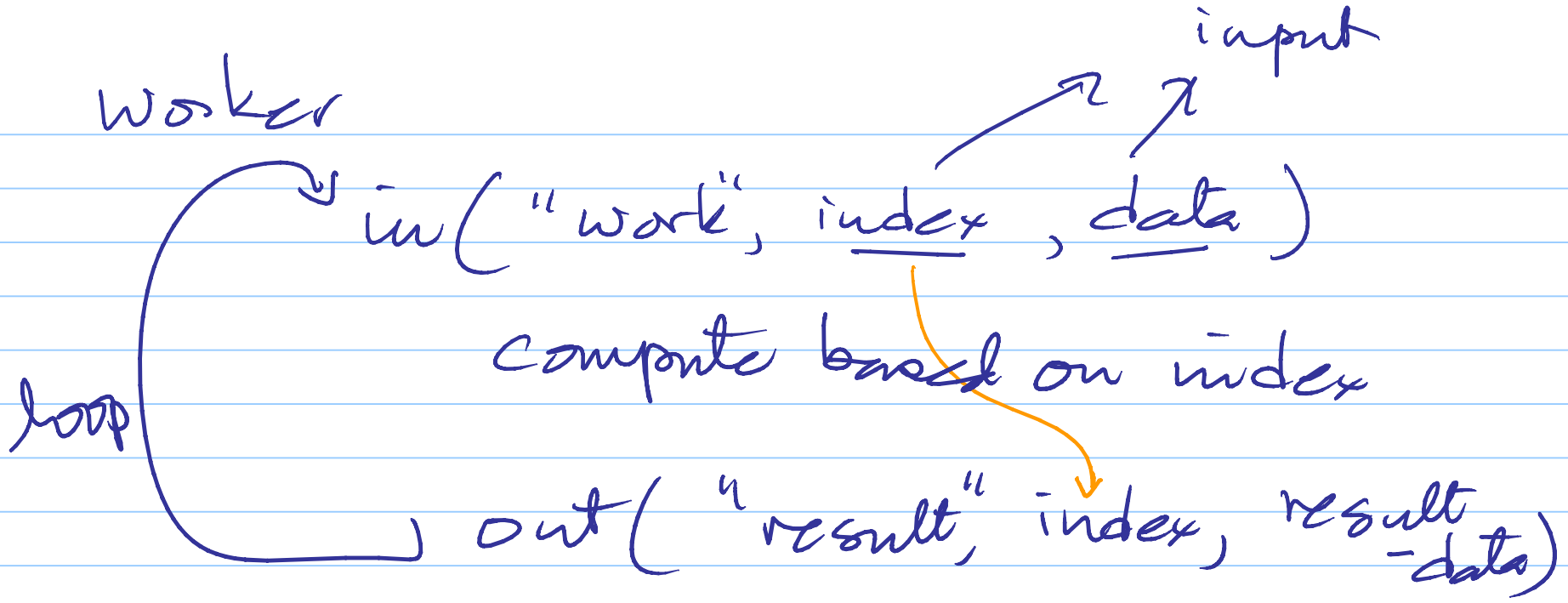
out ("work", index, data)

multiple

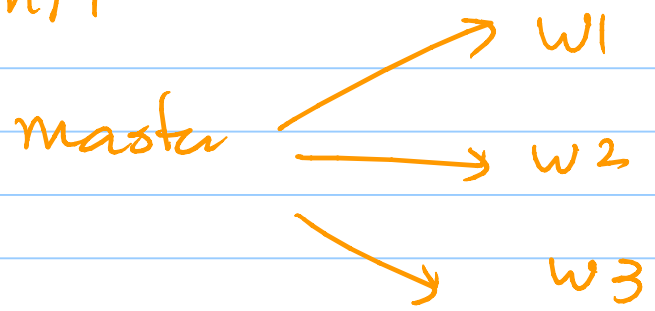
different

in ("result", index, data)

input


worker
loop
in("work", index, data)
input
compute based on index
out("result", index, result_data)

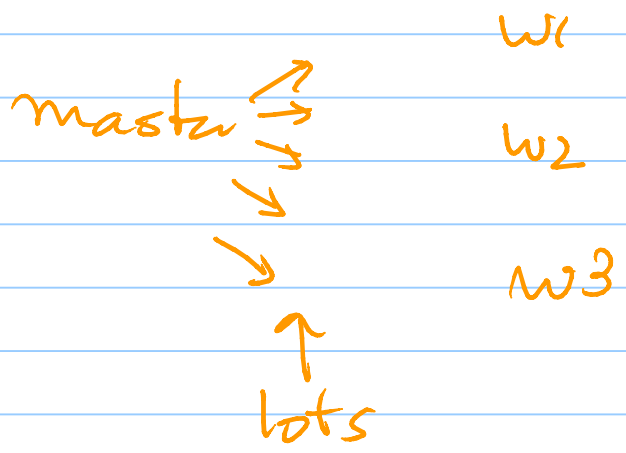
MPI



time taken

= slowest worker

Linda



time → from worker