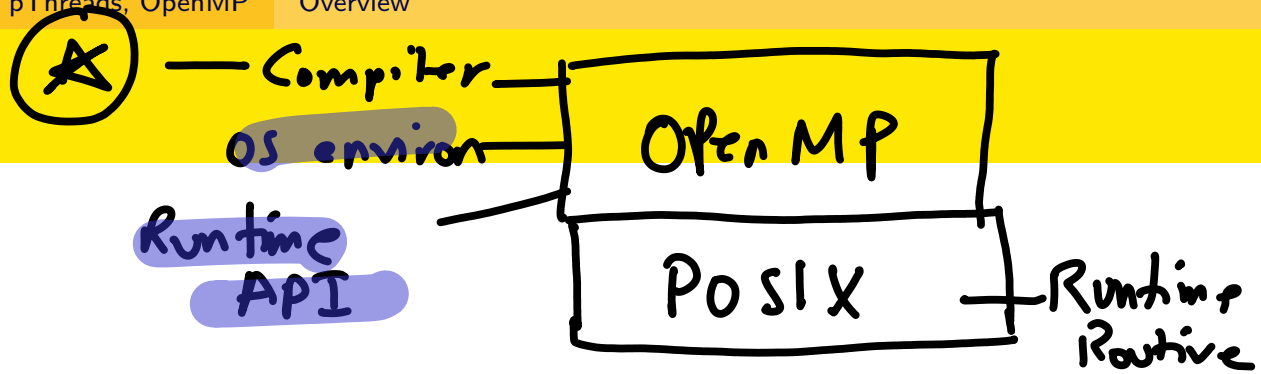
Overview



OpenMP

- Open Multi Processing
- Portable (Windows, Linux), shared-memory threading API for C/C++/Fortran (around 60 methods)
- Combines serial and parallel code in single source file
- Current specification: OpenMP 4.0
- Compiler directives (Native to C/C++/Fortran)
  - Runtime library routines (e.g. increase/decrease threads as required)
  - Environment variables

openmp

native

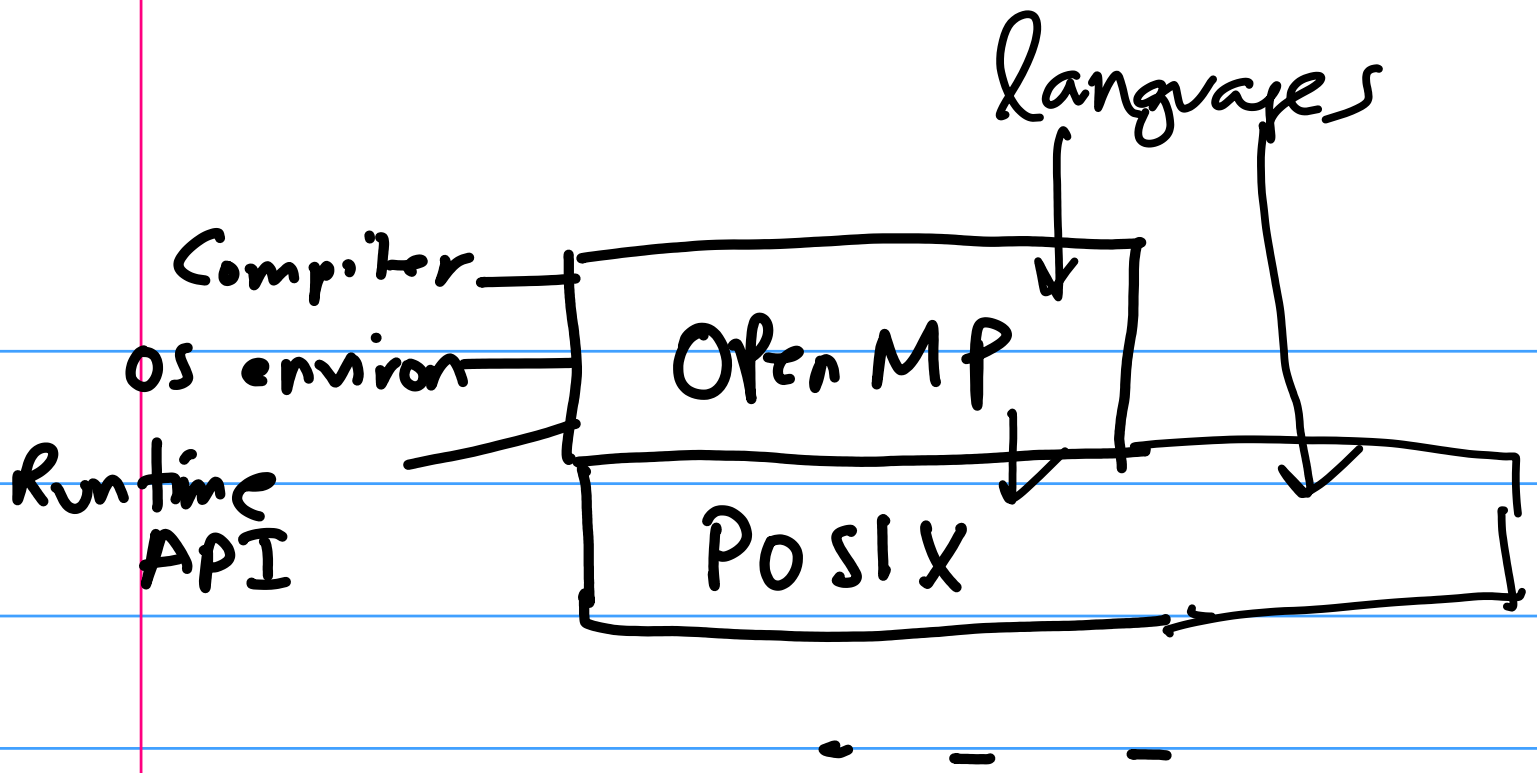
Compiler Directives

1 2 3 4

```
#pragma omp construct clauses
```

- **#pragma omp** Sentinel
- **construct** The construct, or directive Name
- **clauses** e.g., shared, schedule, etc.

#include
#define
#if
#ifndef
...


languages
Compiler
OS environ
Runtime API
OpenMP
POSIX

Overview (cont.)

Using

Runtime Routines

data collection.

- 61 routines
- Perform tasks such as Control number of threads, Query thread information, lock management, wall clock time monitoring, etc.
- Example:

```
omp_set_num_threads(8);
```

OS

Compiler X

Environment Variables

- Alternative to runtime routines

```
export OMP_NUM_THREADS=8 && source /etc/profile
```

X X

run time X

The Code (No. of Threads)

- Master thread spawns a team of threads as needed. (all thread creation work done transparently, developer saved from details)

```
#include <omp.h>
```

// header

```
int main(int args, char *argv) {
  #pragma omp parallel num_threads(4)
  {
    tid = omp_get_thread_num();
    printf("Hello from %d\n", tid);
  }
}
```

Compiler

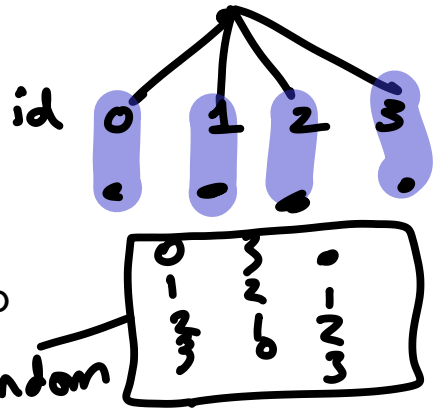
API

- Compilation (Intel)

  `icc file.c -openmp`

- Compilation (GNU)

  `gcc file.c -fopenmp`



- Common Runtime Routines:

```
omp_set_num_threads(int);
/* Who am I */
omp_get_thread_num();
omp_get_max_threads();
/* Are we in a parallel region?*/
omp_in_parallel();
/* How many processing cores*/
omp_get_num_procs();
/* Lock a thread */
omp_set_lock();
```

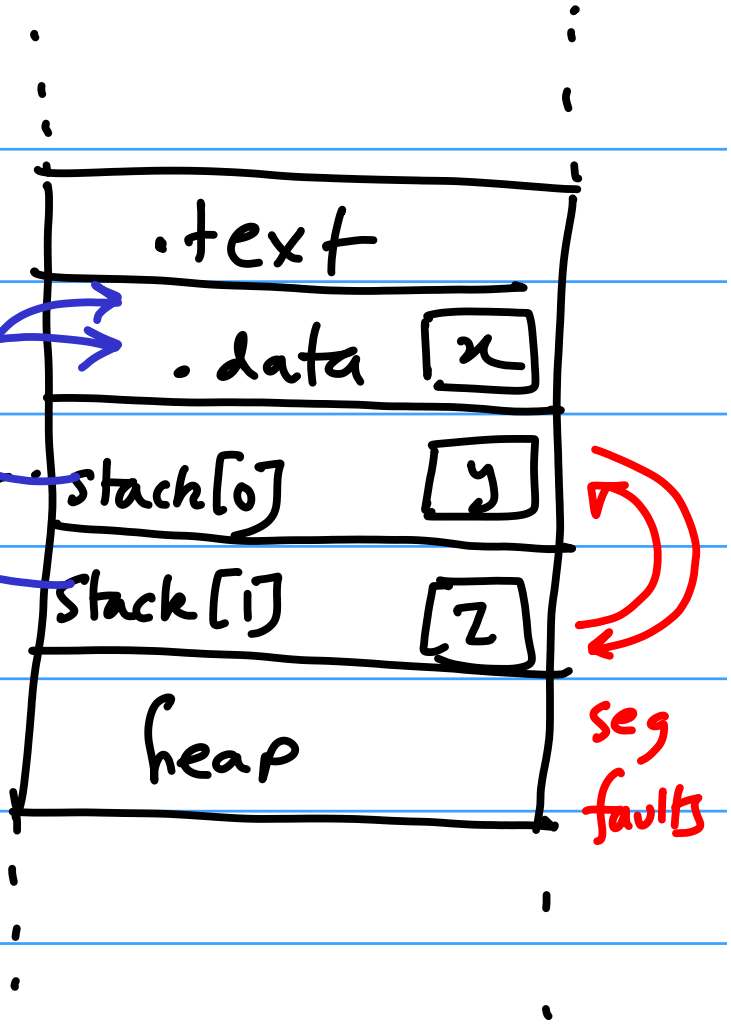
- Shared vs Private (What will happen below?)

```
int tid;
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num()
}
#pragma omp parallel shared(tid)
{
    tid = omp_get_thread_num()
}
```

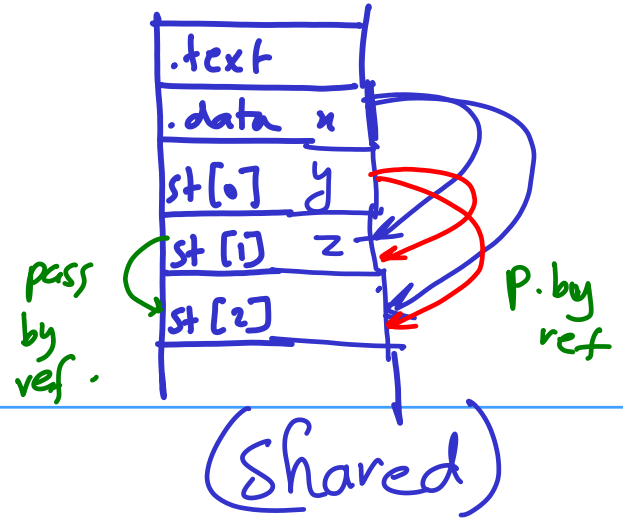
```
int x
void foo()
{
    int z
}
int main()
{
    int y
}
```

global.

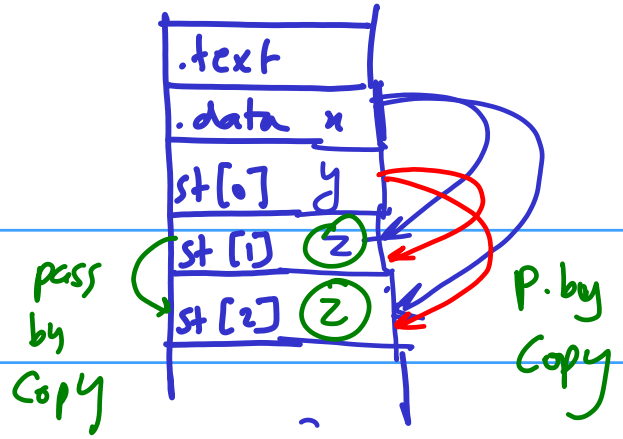
local

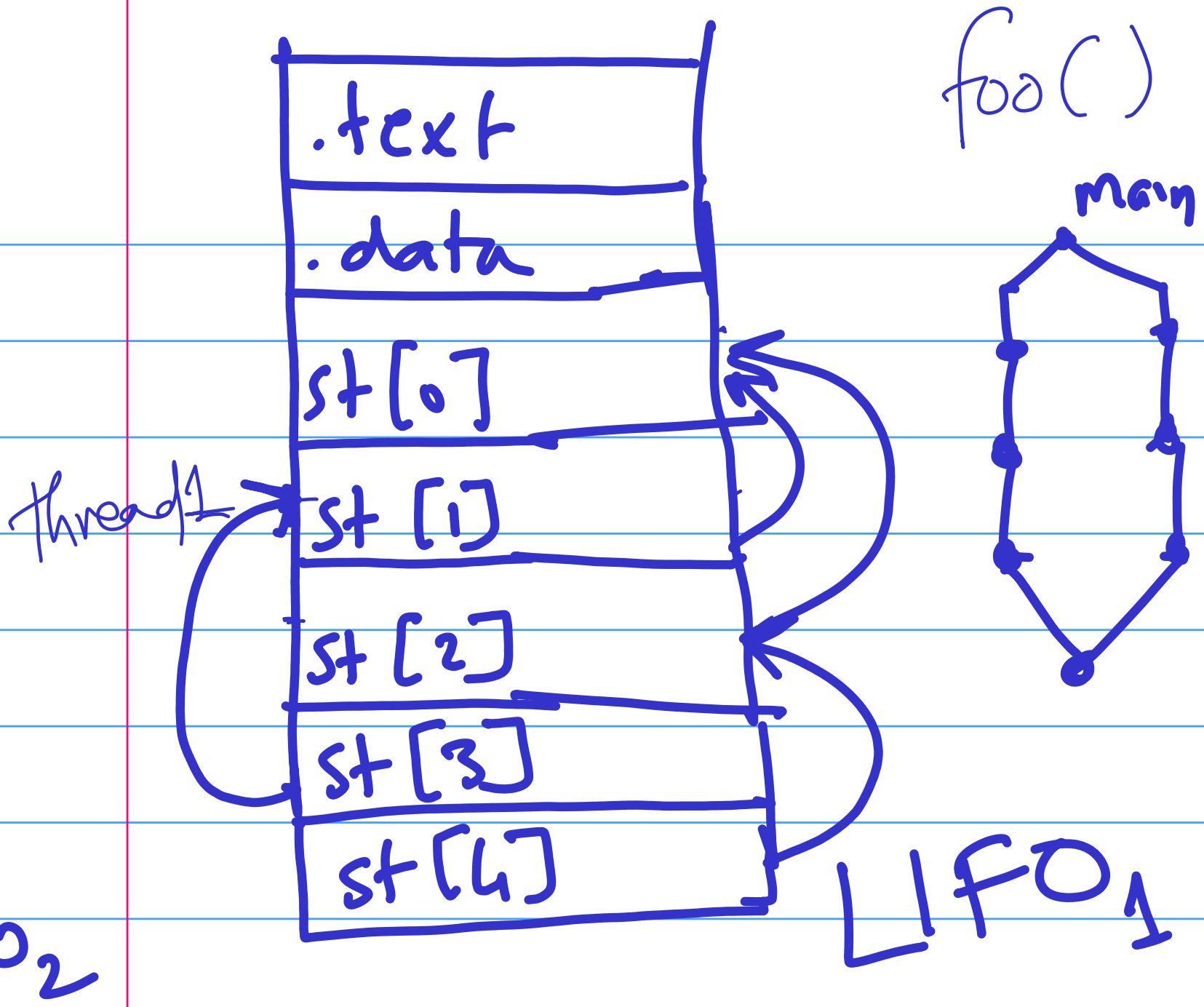


```
int x        → global

int main()
{
    int y    → local.

    #pragma omp parallel shared(z)
    {
        int z
    }
}
```



```
int x        → global

int main()
{
    int y    → local.

    #pragma omp parallel private(z)
    {
        int z
    }
}
```




.text
.data
st[0]
st[1]
st[2]
st[3]
st[4]
Thread1
foo()
main
LIFO2
LIFO1

The Code (No. of Threads)

- Master thread spawns a team of threads as needed. (all thread creation work done transparently, developer saved from details)

```
#include <omp.h>

int main(int args, char *argv) {
  #pragma omp parallel num_threads(4)
  {
    tid = omp_get_thread_num();
    printf("Hello from %d\n", tid);
  }
}
```

- Compilation (Intel)

  `icc file.c -openmp`

- Compilation (GNU)

  `gcc file.c -fopenmp`

info

algorithms parallel enable.

- Common Runtime Routines:

```
omp_set_num_threads(int);
/* Who am I */
omp_get_thread_num();
omp_get_max_threads();
/* Are we in a parallel region?*/
omp_in_parallel();
/* How many processing cores*/
omp_get_num_procs();
/* Lock a thread */
omp_set_lock();
```

- Shared vs Private (What will happen below?)

```
int tid;
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num()
}
#pragma omp parallel shared(tid)
{
    tid = omp_get_thread_num()
}
```

The Code (No. of Threads)

- Master thread spawns a team of threads as needed. (all thread creation work done transparently, developer saved from details)

```
#include <omp.h>

int main(int args, char *argv) {
  #pragma omp parallel num_threads(4)
  {
    tid = omp_get_thread_num();
    printf("Hello from %d\n", tid);
  }
}
```

Compiler

- Compilation (Intel)

  `icc file.c -openmp`

- Compilation (GNU)

  `gcc file.c -fopenmp`

#pragma omp parallel

- Common Runtime Routines:

```
omp_set_num_threads(int);
/* Who am I */
omp_get_thread_num();
omp_get_max_threads();
/* Are we in a parallel region?*/
omp_in_parallel();
/* How many processing cores*/
omp_get_num_procs();
/* Lock a thread */
omp_set_lock();
```

Runtime.

- Shared vs Private (What will happen below?)

```
int tid;
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num()
}
#pragma omp parallel shared(tid)
{
    tid = omp_get_thread_num()
}
```

The Code (No. of Threads)

- Master thread spawns a team of threads as needed. (all thread creation work done transparently, developer saved from details)

```
#include <omp.h>

int main(int args, char *argv) {
  #pragma omp parallel num_threads(4)
  {
    tid = omp_get_thread_num();
    printf("Hello from %d\n", tid);
  }
}
```

- Compilation (Intel)

  `icc file.c -openmp`

- Compilation (GNU)

  `gcc file.c -fopenmp`

- Common Runtime Routines:

```
omp_set_num_threads(int);
/* Who am I */
omp_get_thread_num();
omp_get_max_threads();
/* Are we in a parallel region?*/
omp_in_parallel();
/* How many processing cores*/
omp_get_num_procs();
/* Lock a thread */
omp_set_lock();
```

id

total

- Shared vs Private (What will happen below?)

```
int tid;
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num()
}
#pragma omp parallel shared(tid)
{
    tid = omp_get_thread_num()
}
```

The Code (No. of Threads)

- Master thread spawns a team of threads as needed. (all thread creation work done transparently, developer saved from details)

```
#include <omp.h>

int main(int args, char *argv) {
  #pragma omp parallel num_threads(4)
  {
    tid = omp_get_thread_num();
    printf("Hello from %d\n", tid);
  }
}
```

- Compilation (Intel)

  `icc file.c -openmp`

- Compilation (GNU)

  `gcc file.c -fopenmp`

- Common Runtime Routines:

```
omp_set_num_threads(int);
/* Who am I */
omp_get_thread_num();
omp_get_max_threads();
/* Are we in a parallel region?*/
omp_in_parallel();
/* How many processing cores*/
omp_get_num_procs();
/* Lock a thread */
omp_set_lock();
```

- Shared vs Private (What will happen below?)

```
int tid;
#pragma omp parallel private(tid)
{
    tid = omp_get_thread_num()
}
#pragma omp parallel shared(tid)
{
    tid = omp_get_thread_num()
}
```

0 dependence.

identification ✓

iterations ✓

```
for(i=0; i<5; i++)
{
    printf("hello %d", i);
    ?
}
```

hello	0
"	1
"	2
"	3

5

```
#pragma omp parallel num_threads()
{
    int i = omp_get_thread_num();
    printf("hello %d", i);
}
```

for loop ⟹ Omp parallel construct.

Scenario 1

for (i _____)

A[i-1] + A[i-2]

→ Special needs.

reduction → reduce.

Scenario 1 = 2nd example

operators.

+=

-=

*=

```
sum = 0
for(i=0; i<10; i++)
    sum += a[i]
```

operator induced

new. previous

sum = sum + a[i]

X i-1, i-2

```
sum = 0
for(i=0; i< 10; i++)
    sum += a[i]
```

① iterations ✓

② i index ✓

option ①

```
sum = 0
#pragma omp parallel num_threads(10) shared(sum)
{
    int i = omp_get_thread_num()
    sum += a[i]
}
```

```
load r1, a[i]
load r2, sum
add r1, r2, r3
store r3, sum
```

Nature of execution → Race condition

```
sum = 0
for(i=0; i< 10; i++)
    sum += a[i]
```

① iterations ✓

② i index ✓

option ②

```
sum = 0
#pragma omp parallel num_threads(10) private(sum)
{
    int i = omp_get_thread_num()
    sum += a[i]
}
```

```
load r1, a[i]
load r2, sum
add r1, r2, r3
store r3, sum
```

```
sum = 0
for(i=0; i< 10; i++)
    sum += a[i]
```

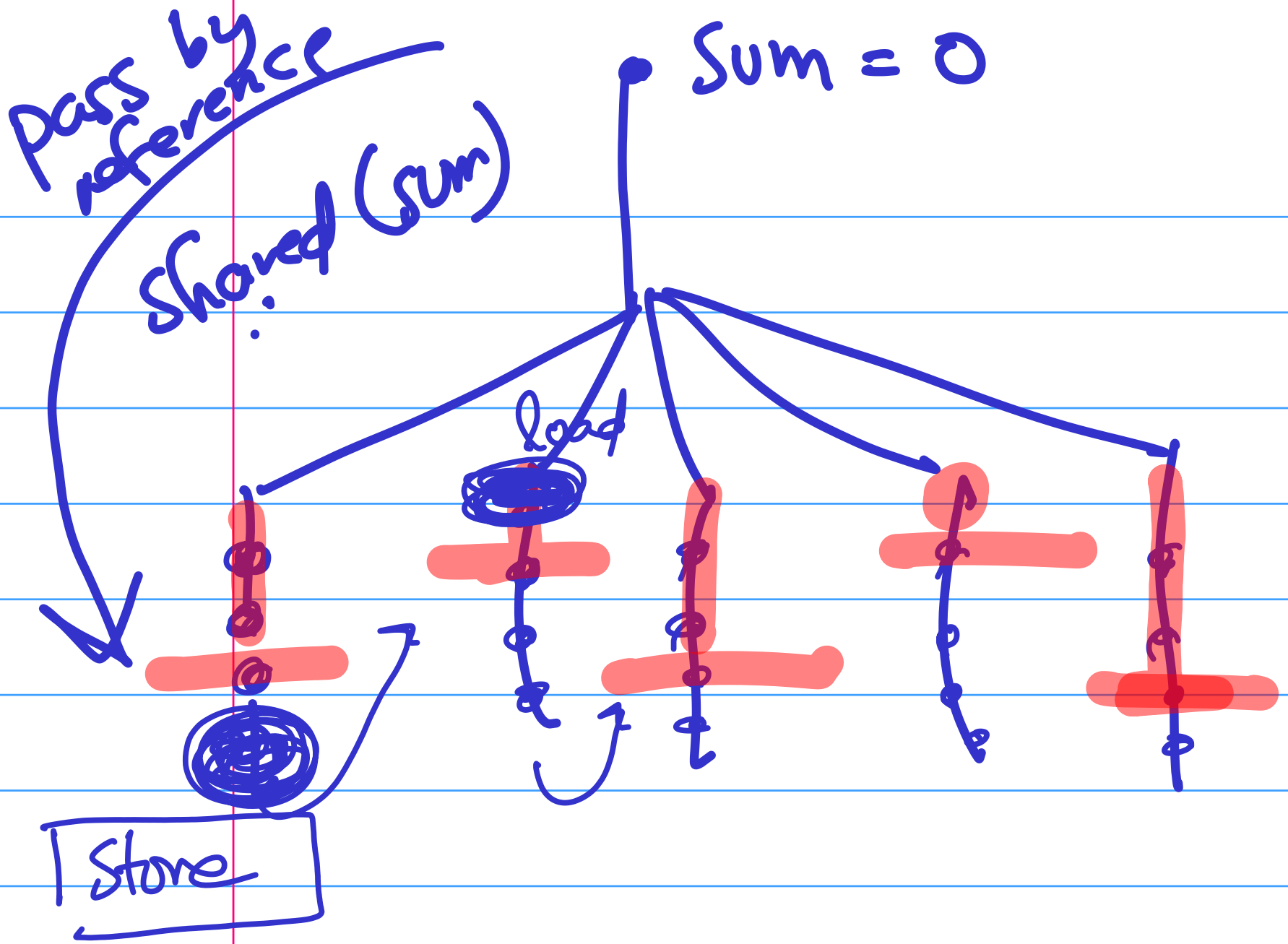
① iterations ✓

② i index ✓

option ③

```
sum = 0
#pragma omp parallel num_threads(10) reduction(sum:+)
{
    int i = omp_get_thread_num()
    sum += a[i]
}
```

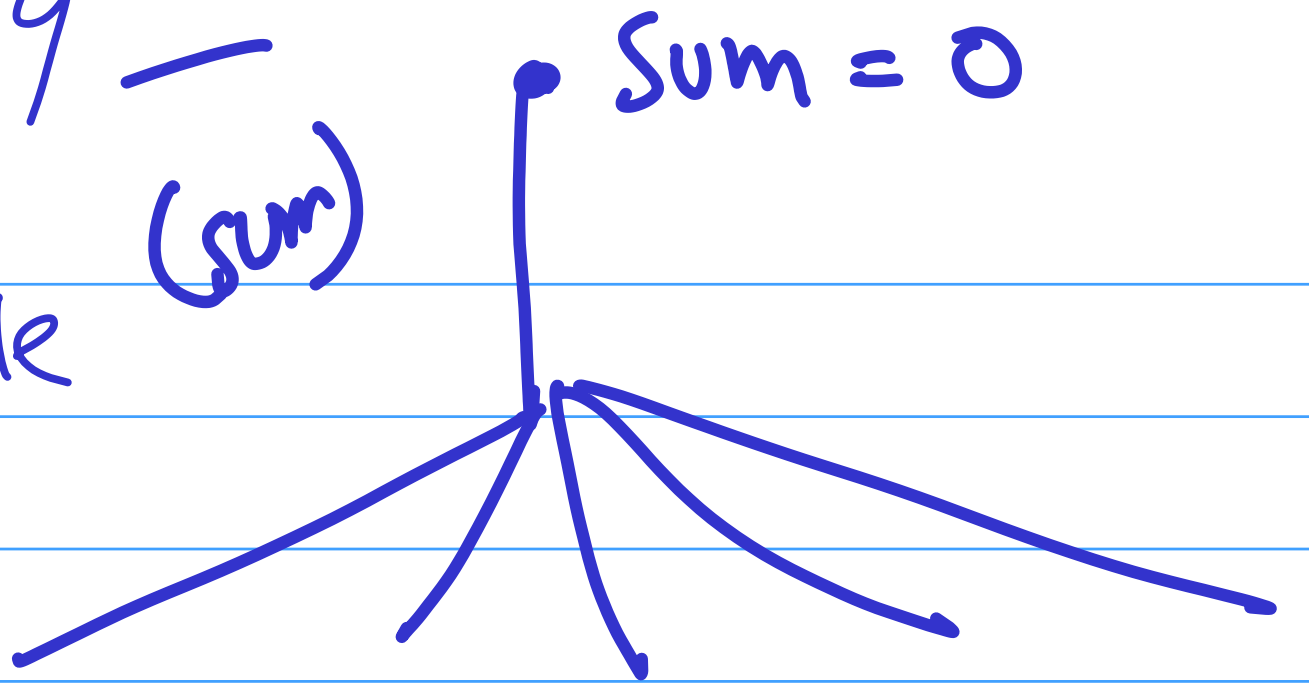
```
load r1, a[i]
load r2, sum
add r1, r2, r3
store r3, sum
```


Pass by reference
Shared (sum)
Sum = 0
load
Store

Pass copy

(sum)

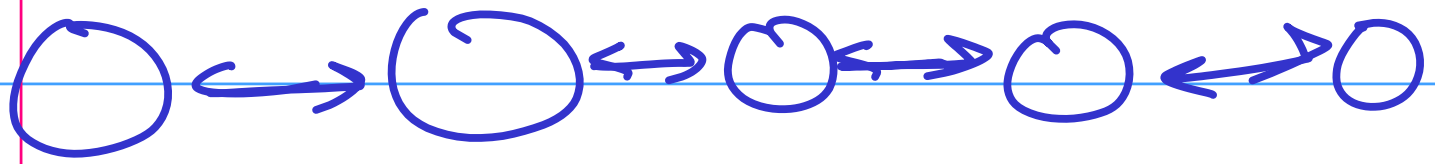
pivale



Sum 0 0 0 0 0

t =

a[i]



Option 3

reduction

pass by copy

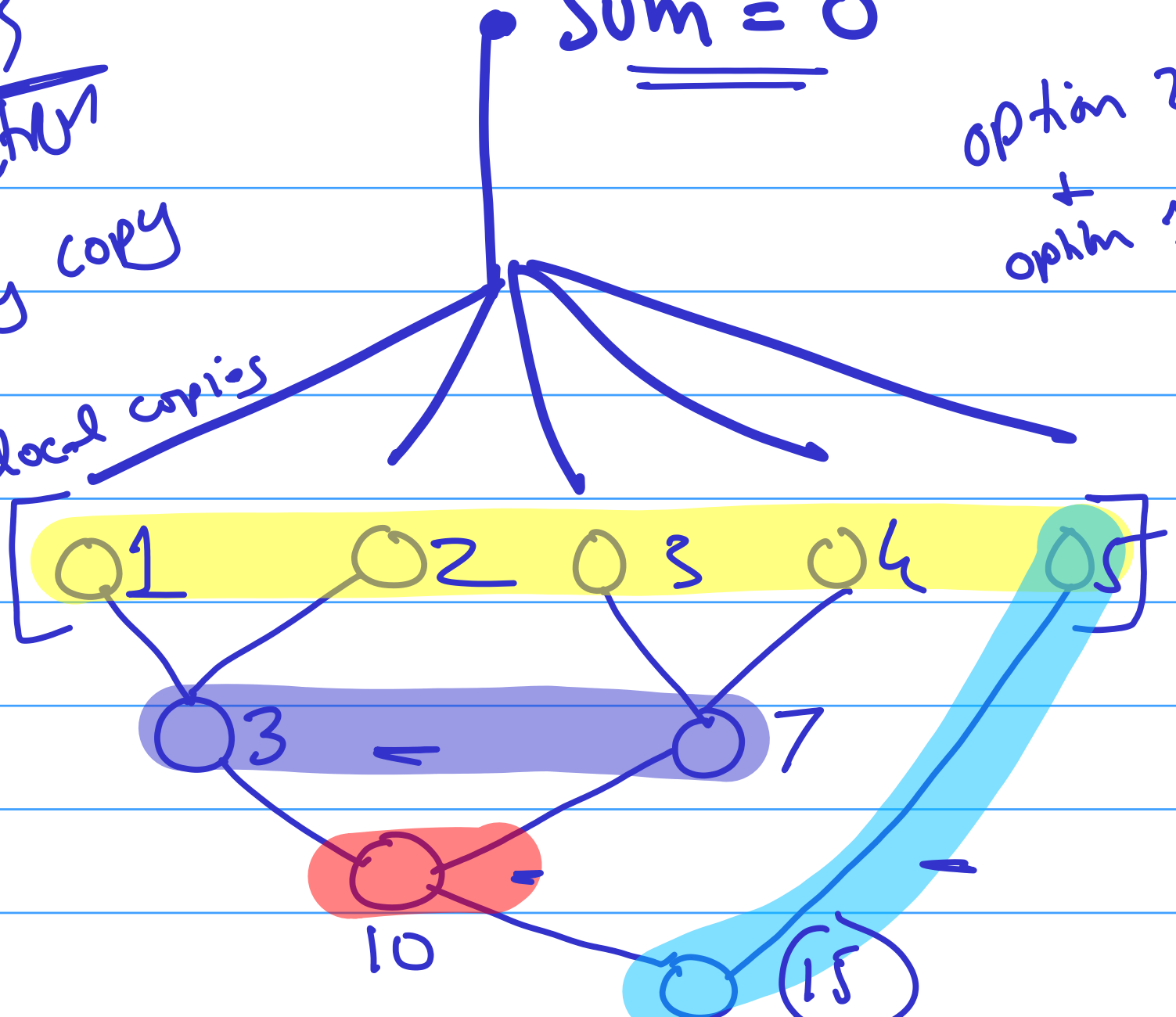
Sum = 0

option 2 + option 1

5 local copies

5 threads

additional threads



for ? ⟹ threads

Scenario 2

?

```
while (1)
{
    [recursive calls
    fork()]
}
```

do/while

Scenario 3

fibonacci

factorial (int a)
{

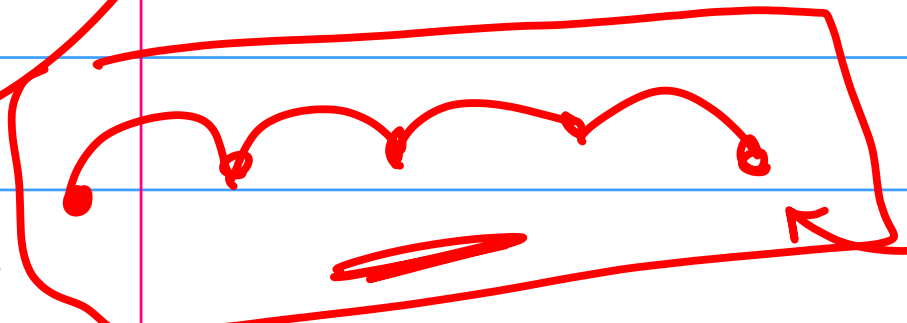
base condition

ret. factorial(a-1);
}

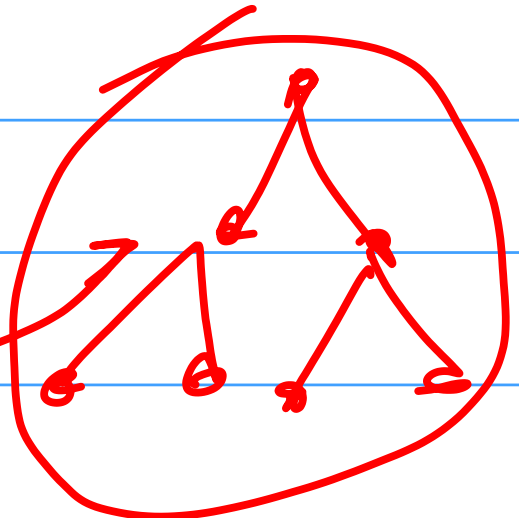
iterations

recursion

factorial (n);



thread



The Code (Reduction)

```
#include <omp.h>
#include <stdio.h>

int main(int args, char *argv)
{
 int tid, N = 1000, sum = 0;
 int array[N];

 for (i = 0; i < N; i++) {
  array[i] = i+1;
 }

 #pragma omp parallel num_threads(N) private(tid) reduction(+:sum)
 {
  tid = omp_get_thread_num();
  sum += array[tid];
 }

 printf("Sum: %d\n", sum);
}
```

reduction(sum : +)

[+, −, *, /
||, &&, ^]

++
−−

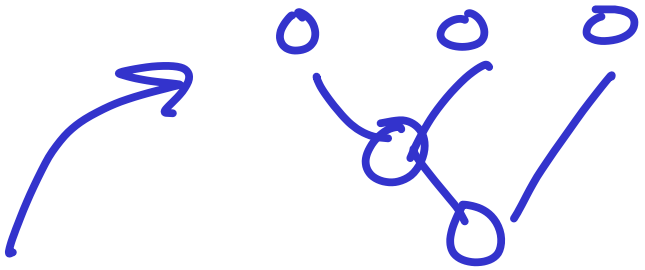
The Code (Reduction)

```
#include <omp.h>
#include <stdio.h>

int main(int args, char *argv)
{
 int tid, N = 1000, sum = 0;
 int array[N];

 for (i = 0; i < N; i++) {
  array[i] = i+1;
 }

 #pragma omp parallel num_threads(N) private(tid) reduction(+:sum)
 {
  tid = omp_get_thread_num();
  sum += array[tid];
 }

 printf("Sum: %d\n", sum);
}
```

3 || 5 = 7

true true

O(1)



3 — 0 0 1 1

5 — 0 1 0 1 OR

0 1 1 1

algorithms v.v. fast

The Code (Reduction)

load balancing.

```
#include <omp.h>
#include <stdio.h>

int main(int args, char *argv)
{
 int tid, N = 1000, sum = 0;
 int array[N];

 for (i = 0; i < N; i++) {
  array[i] = i+1;
 }

 #pragma omp parallel num_threads(N) private(tid) reduction(+:sum)
 {
  tid = omp_get_thread_num();
  sum += array[tid];
 }

 printf("Sum: %d\n", sum);
}
```

tree structure of threads

complicated example