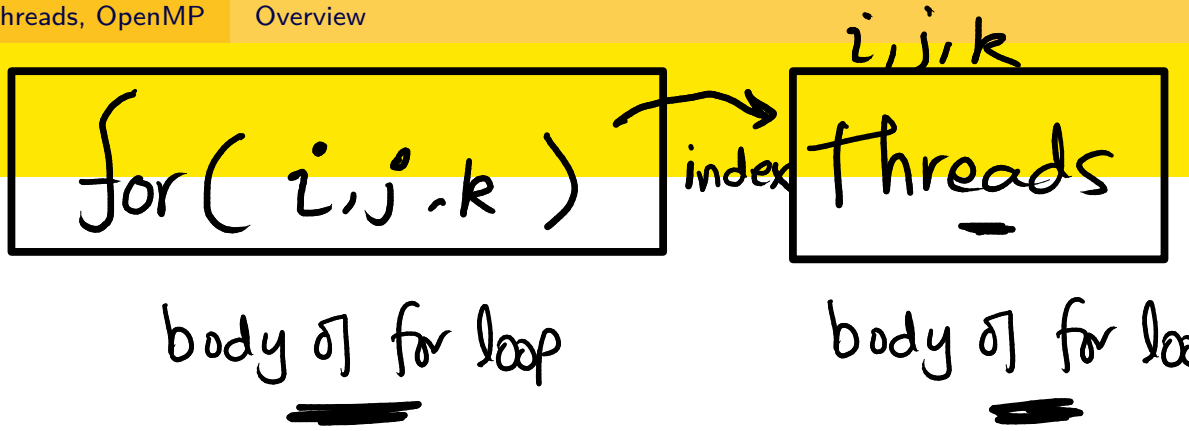
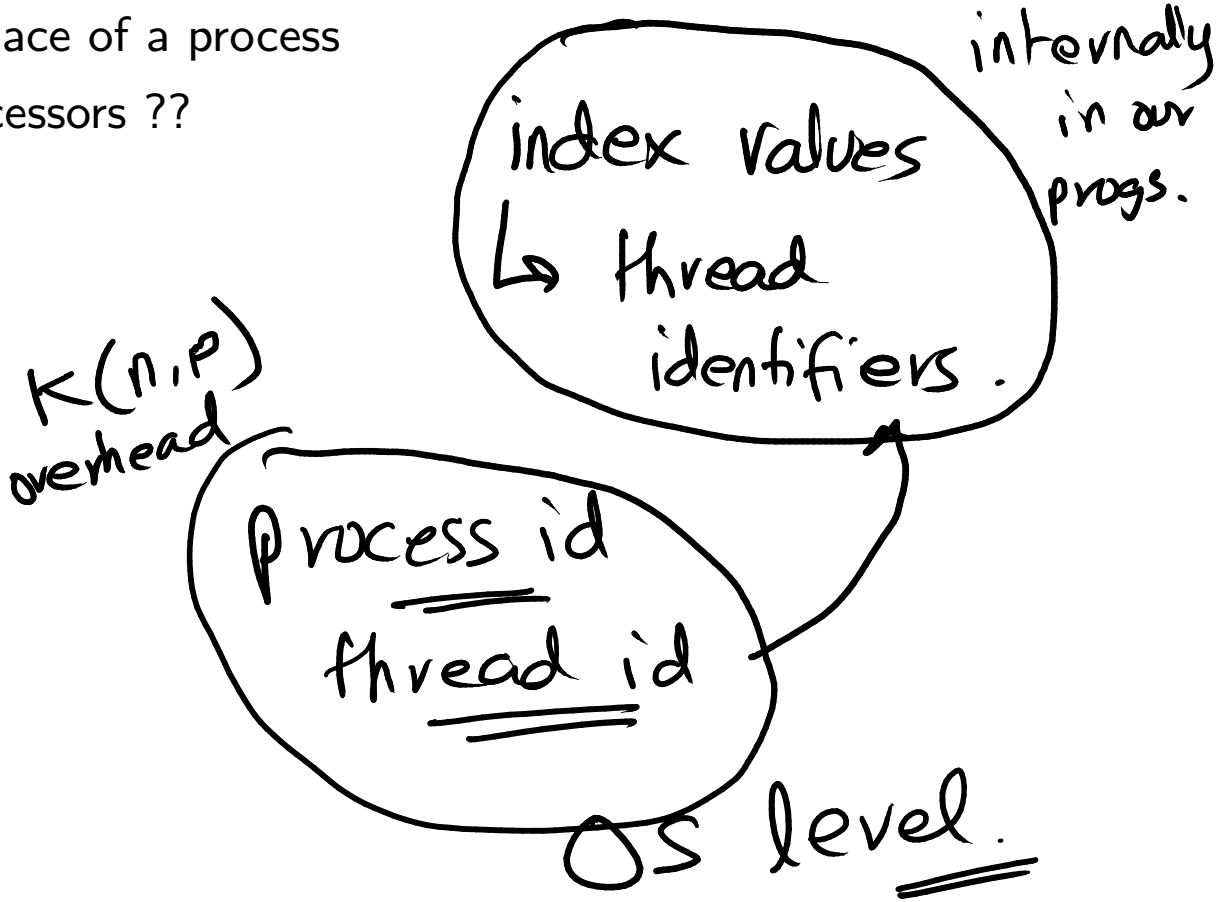
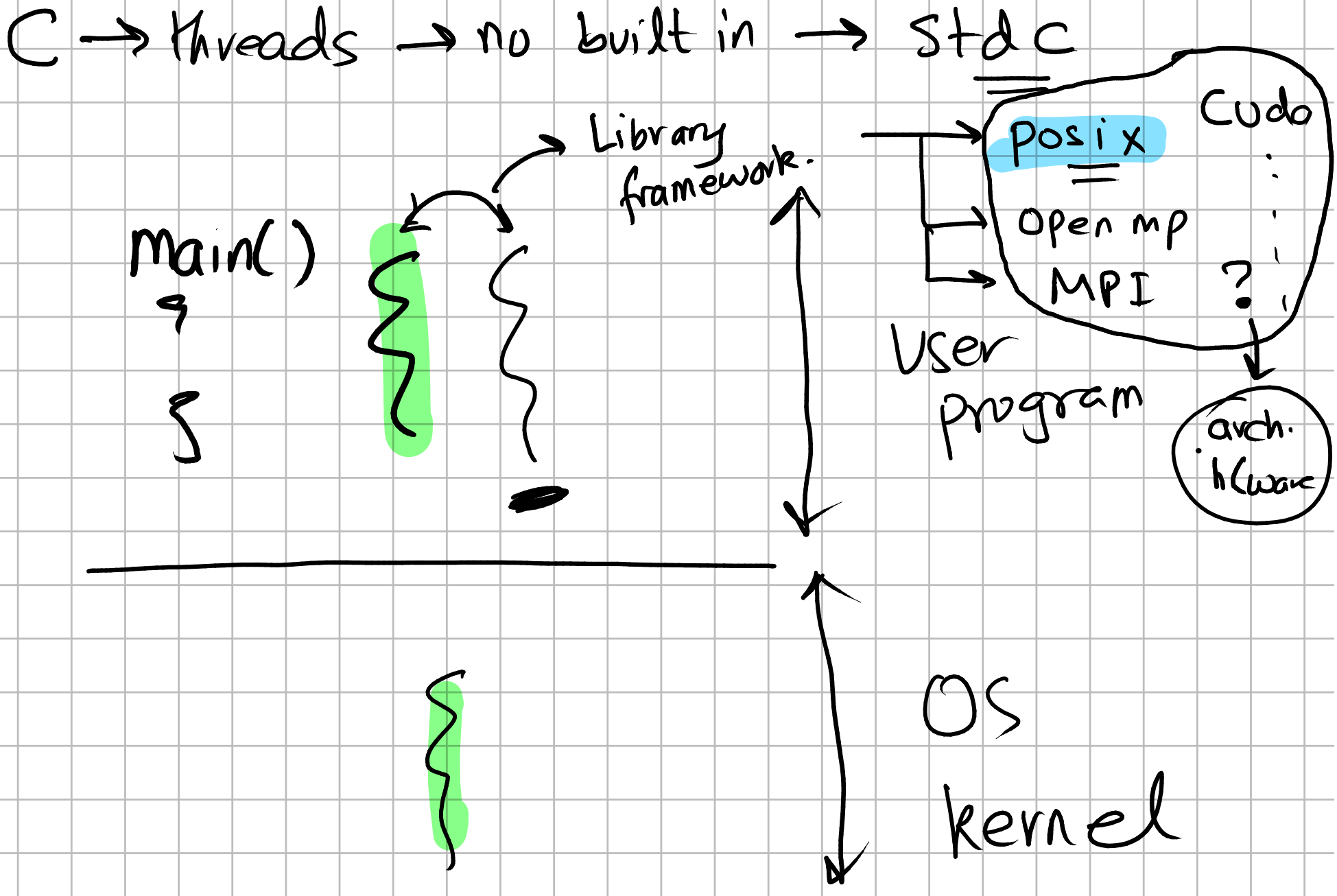
Threads



- Smallest execution path
- Runs within scope/address space of a process
- No. of threads = No. of processors ??
- Fork-Join Model

```
main()
{
  // serial region
  fork();
  // parallel region
  join();
  // serial region
}
```




C → threads → no built in → stdc
Library framework.
Posix
Open mp
MPI
Cuda
?
User program
arch. hware
main()
OS kernel

main()

→ pthread_t t1 → number type

→ data type
unsigned long int ?

k(n,p) →

→ Pthread_create (&t1,
NULL, foo, NULL);

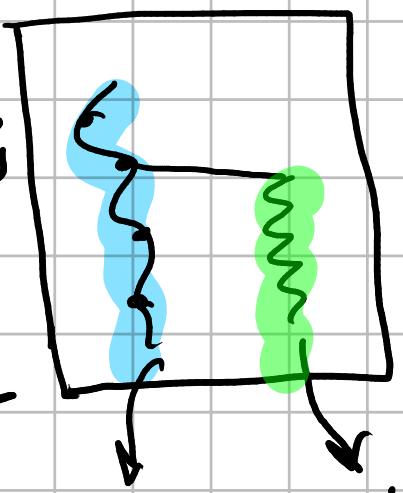
Pthread_join(&t1);

}

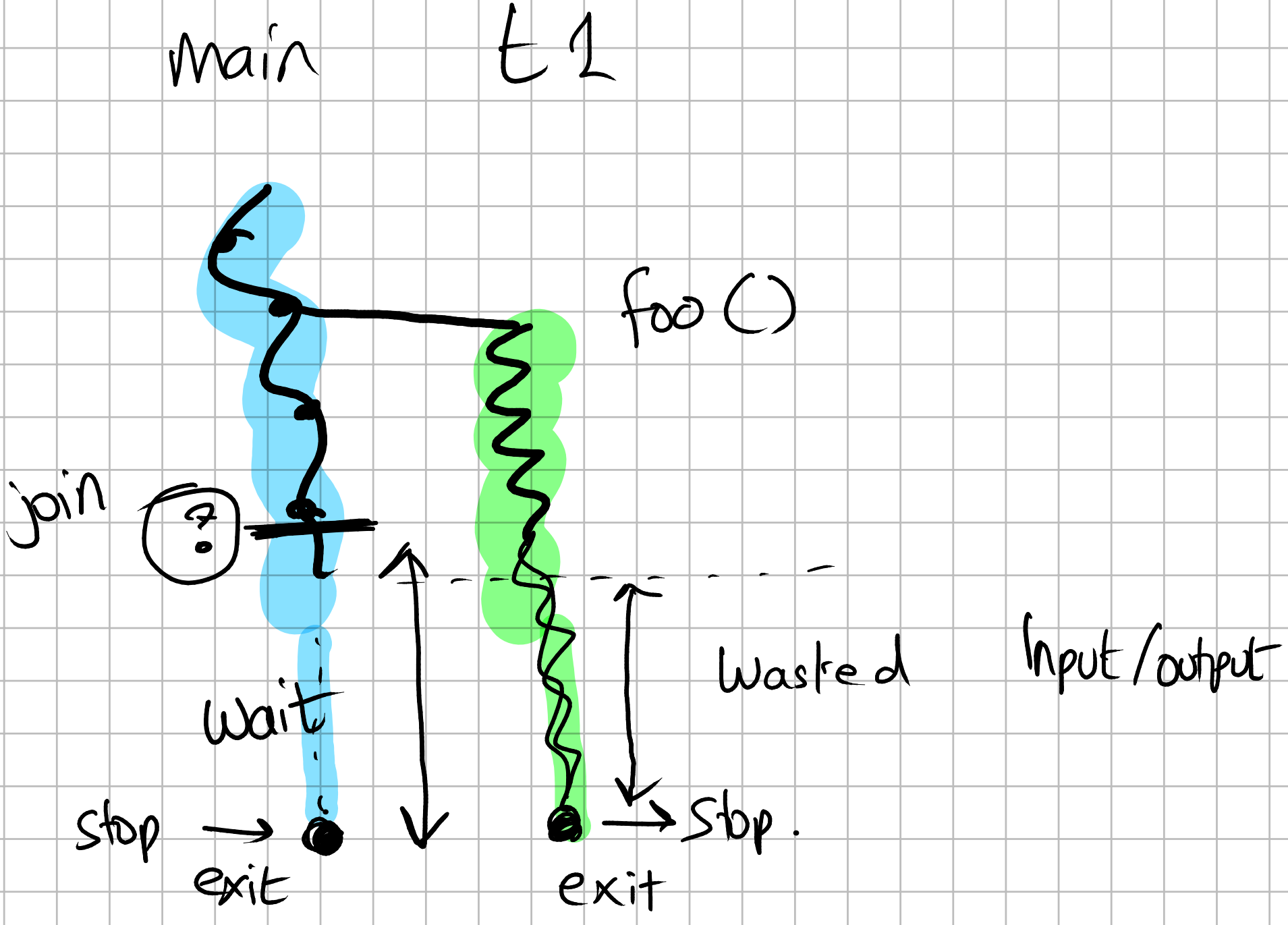
Void *foo()

{

}



simple
tricky.


main
t1
foo()
join
?
wait
wasted
Input/output
stop
exit
Stop.
exit

Posix Threads **pthreads**

- Before there was OpenMP, common approach to support parallel programming was(is) **pthreads**
- **P**ortable **O**perating **S**ystem **I**nterface for UNI**X**
- Originally for UNIX and Linux, but meant for all operating systems that are POSIX standard compliant (Windows did not fall down this way)

```
#include <pthread.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid) {
    printf("\n%d: Hello World!\n", threadid);
    pthread_exit(NULL);
}

int main() {
    pthread_t threads[NUM_THREADS];
    int rc, t;
    for(t=0;t < NUM_THREADS;t++) {
        printf("Creating thread %d\n", t);
        rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
        if (rc) printf("ERROR; return code from pthread_create() is %d\n", rc);
    }
    pthread_exit(NULL);
}
```