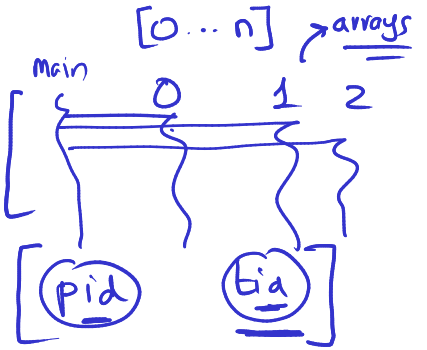
Posix Threads **pthreads**

- 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)

```c
#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);
}
```



Concurrency

Copy

race condition

locks = Concurrency

① More than two variables. → options

void *foo(void *x, void *y)

— posix library.

pthread_create(— , — , — , 4th param passing of variables.)

Overloading

[C++]

[C] —

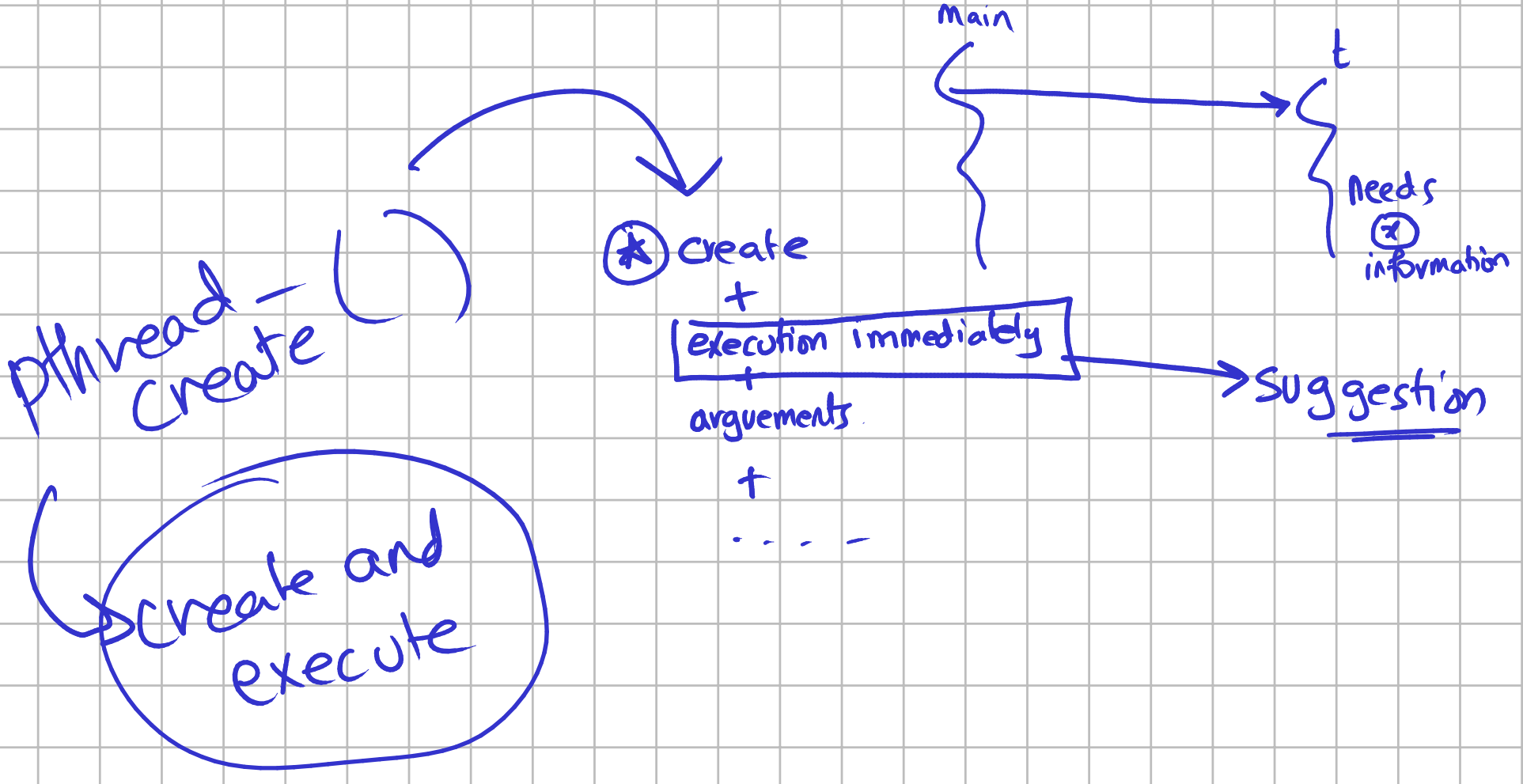
allowed

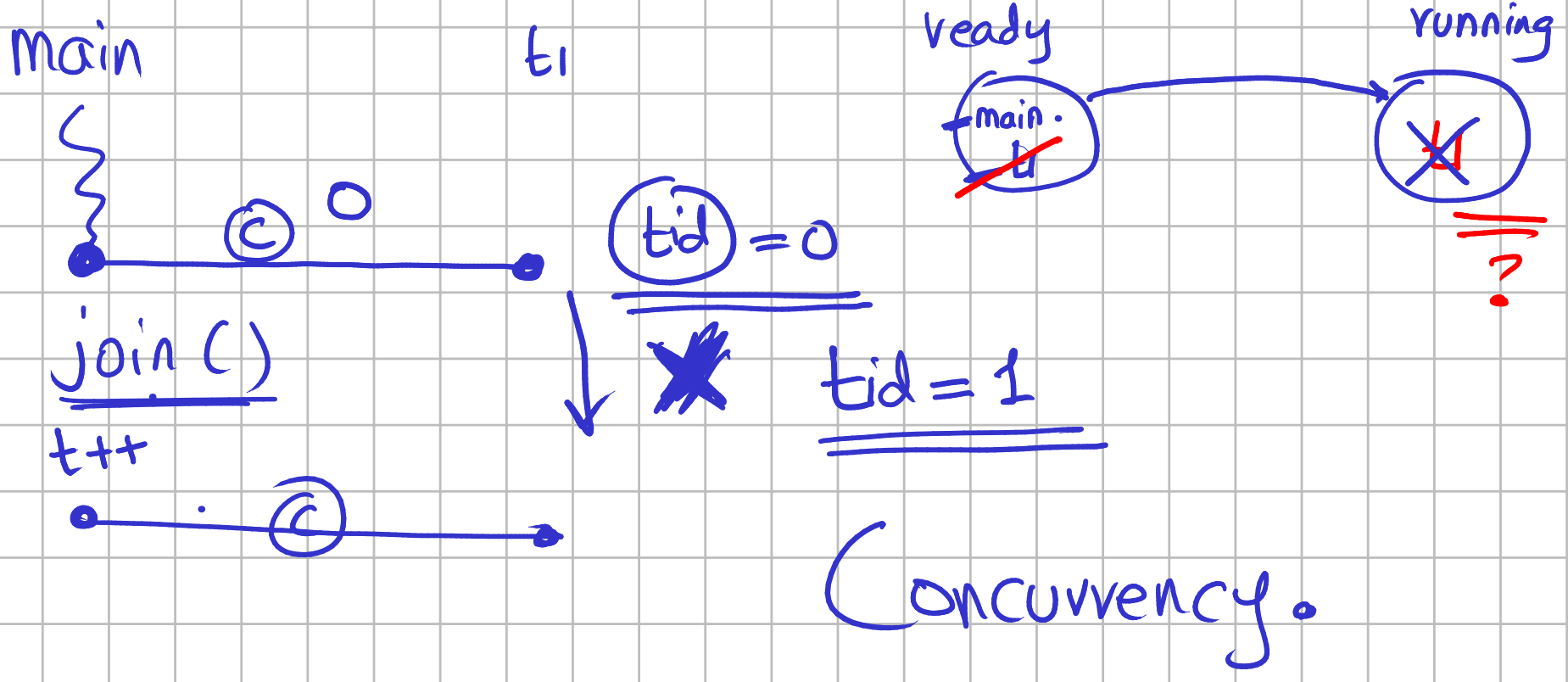
→ NULL

→ (void *) x

→ (void *)x (void *)y

5 parameter.


pthread_create ( )
create and execute
create
+
execution immediately
+
arguements
+
. . . .
Main
t
needs
information
Suggestion

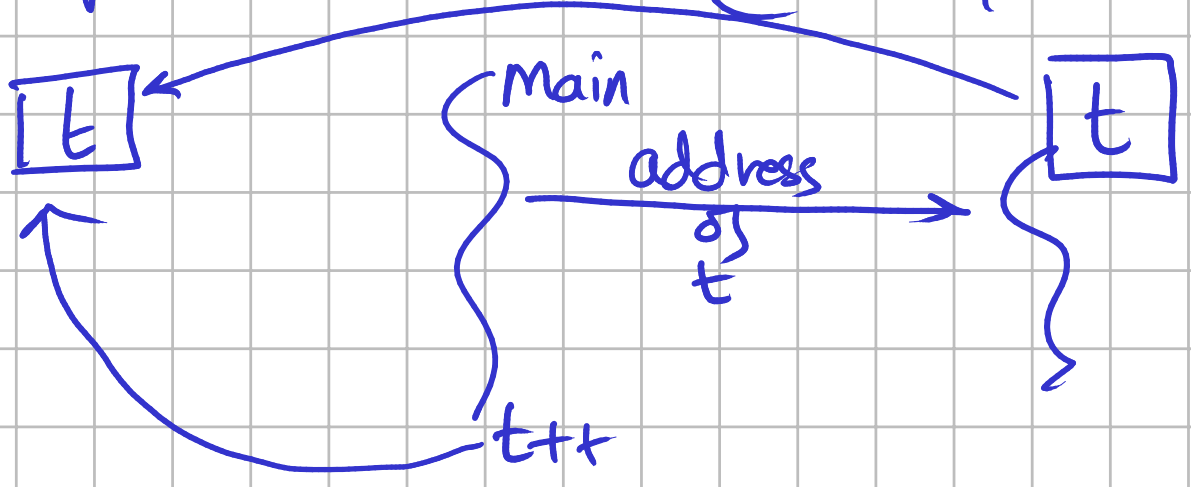


Concurrency.

int tid[5] = {0, 1, 2, 3, 4}

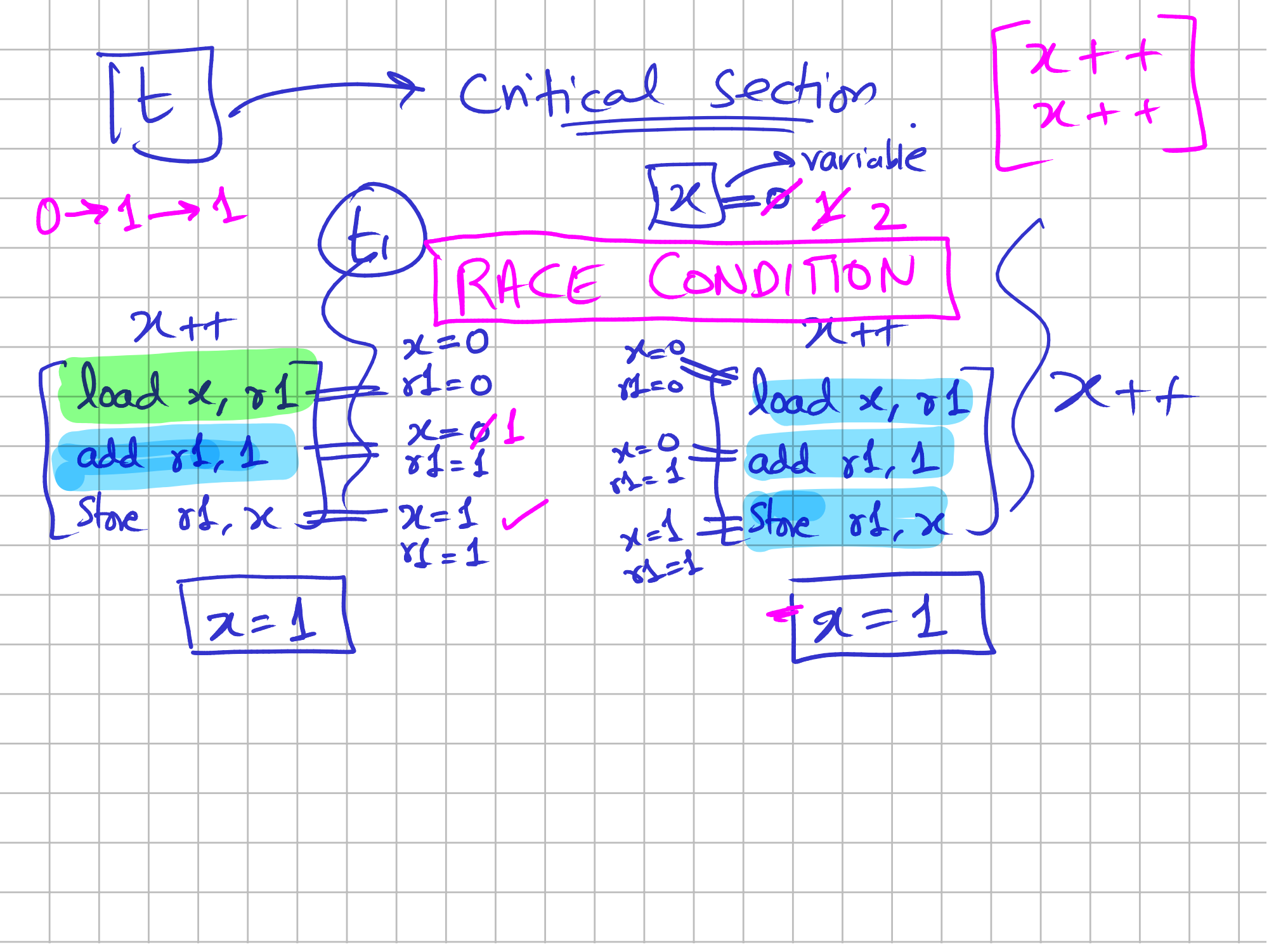
int for

pthread_create(—— , —— , —— , *tid[t])



t

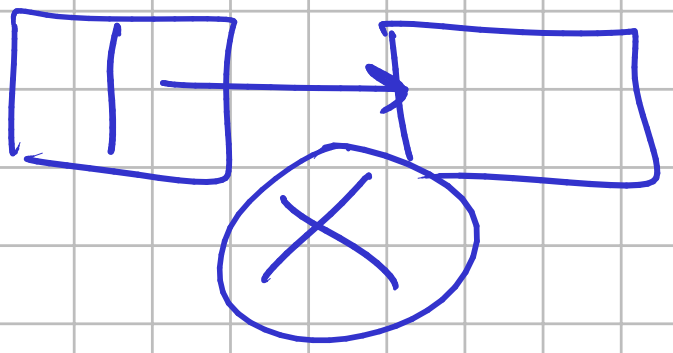
for(t=0; ——
tid[t] = t


t
Critical Section
x++
x++
variable
x = 0 1 2
0 → 1 → 1
t1
RACE CONDITION
x++
load x, r1
add r1, 1
Store r1, x
x=0
r1=0
x=0 1
r1=1
x=1
r1=1
x=0
r1=0
x=0
r1=1
x=1
r1=1
x++
load x, r1
add r1, 1
Store r1, x
x++
x = 1
x = 1

Pass by reference.

```
Void *foo ( Void * M)
{
        M[0] ---> x
        M[1] ---> y
}
```

```
Struct myObj
{
    char x
    int y
    int z
    float z
};
```



Struct myObj M. → M.x ←
M.y ←

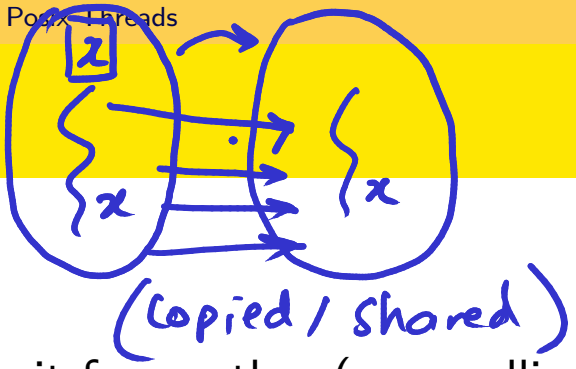
pthread_create(___ , ___ , (void *) M)

void *foo(void * m)

struct myobj *ptr = (struct myobj *) m;

(ptr→x , ptr→y)

Posix Threads **pthreads** (cont.)

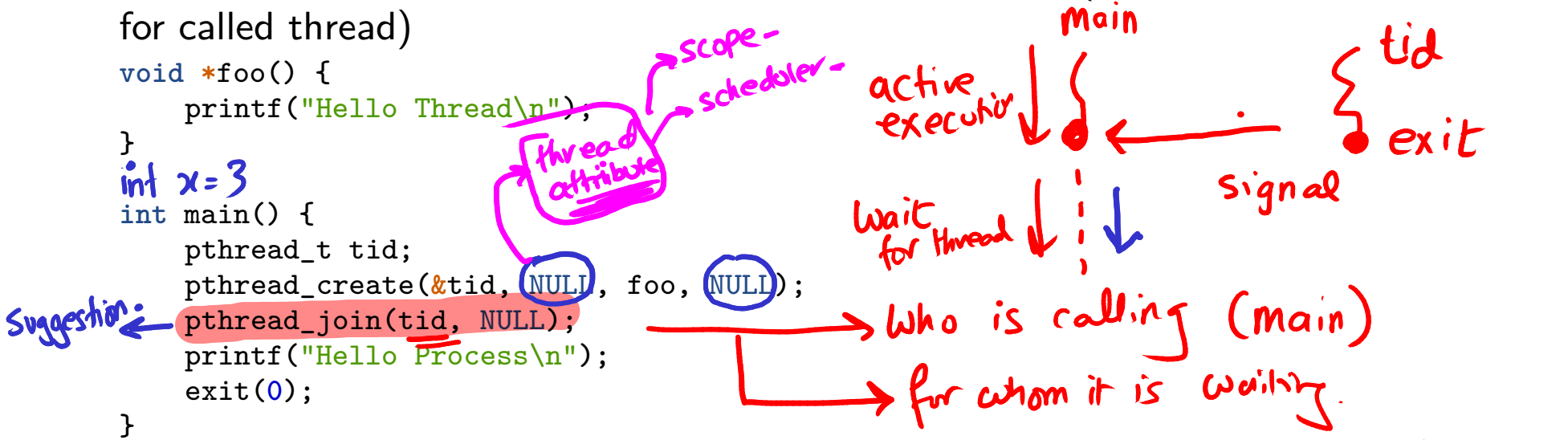


- Compiled as

  `gcc filename.c -lpthread`

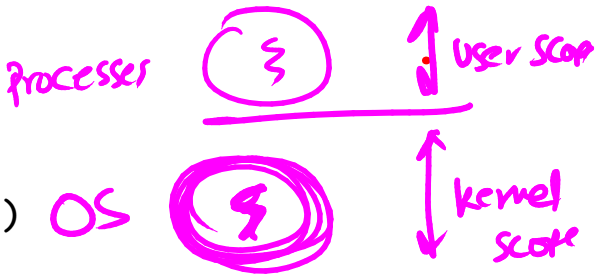
- **Joining a Thread**: Making one thread wait for another (e.g., calling thread waiting for called thread)

```c
void *foo() {
    printf("Hello Thread\n");
}
int x=3
int main() {
    pthread_t tid;
    pthread_create(&tid, NULL, foo, NULL);
    pthread_join(tid, NULL);
    printf("Hello Process\n");
    exit(0);
}
```

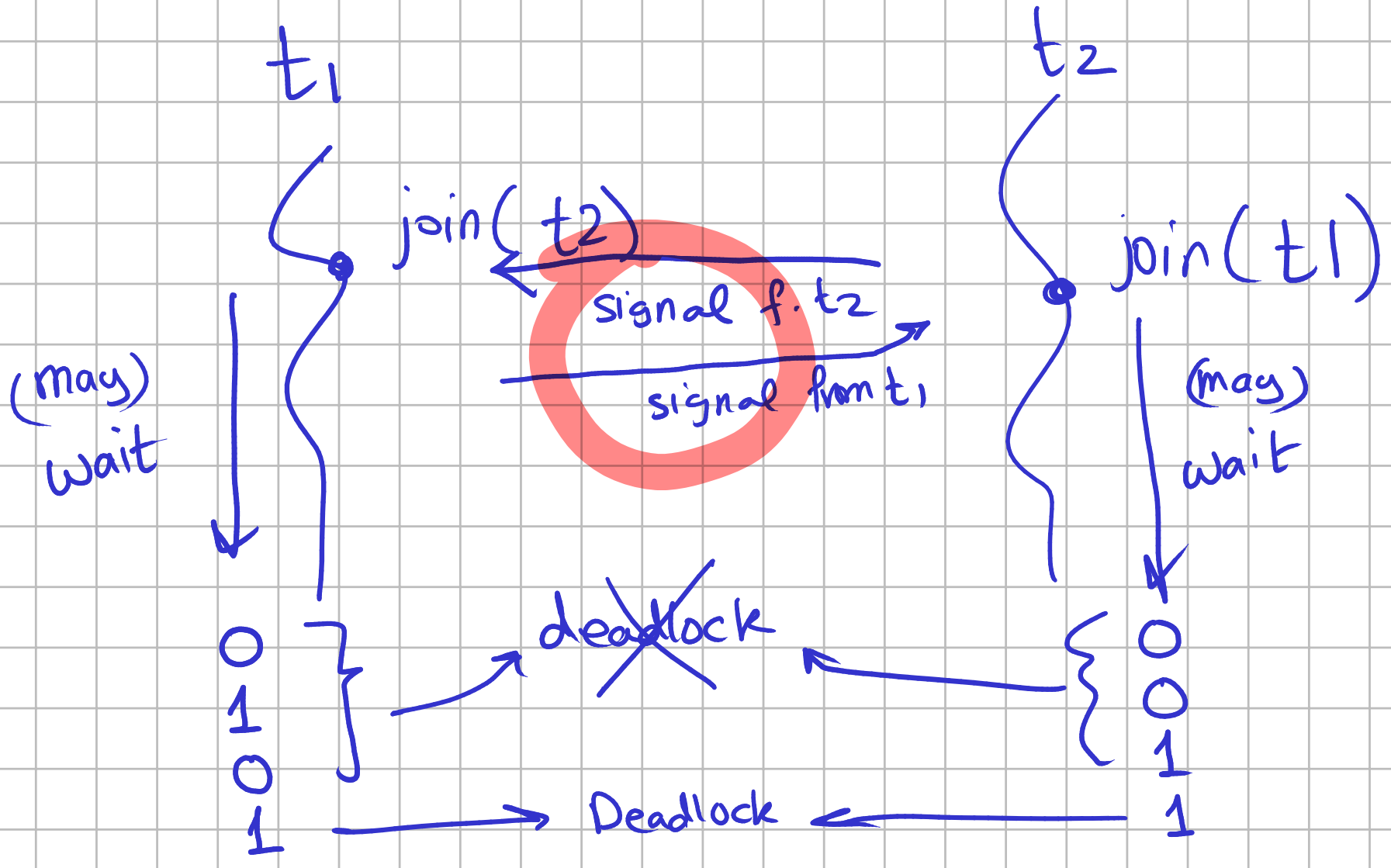


- Many programmers find posix to be hard, cumbersome
  - Function pointers
  - Crypted functions calls such as:

    `pthread_create(), pthread_exit(), pthread_join()`



- Low chances that a compiler may optimize automatically for the above code
- Code is dependent on Posix compatible platforms (operating systems) only.
- Not designed for data-parallelism (scientific computing)



∞ => Deadlock b/w threads.

join

Banker