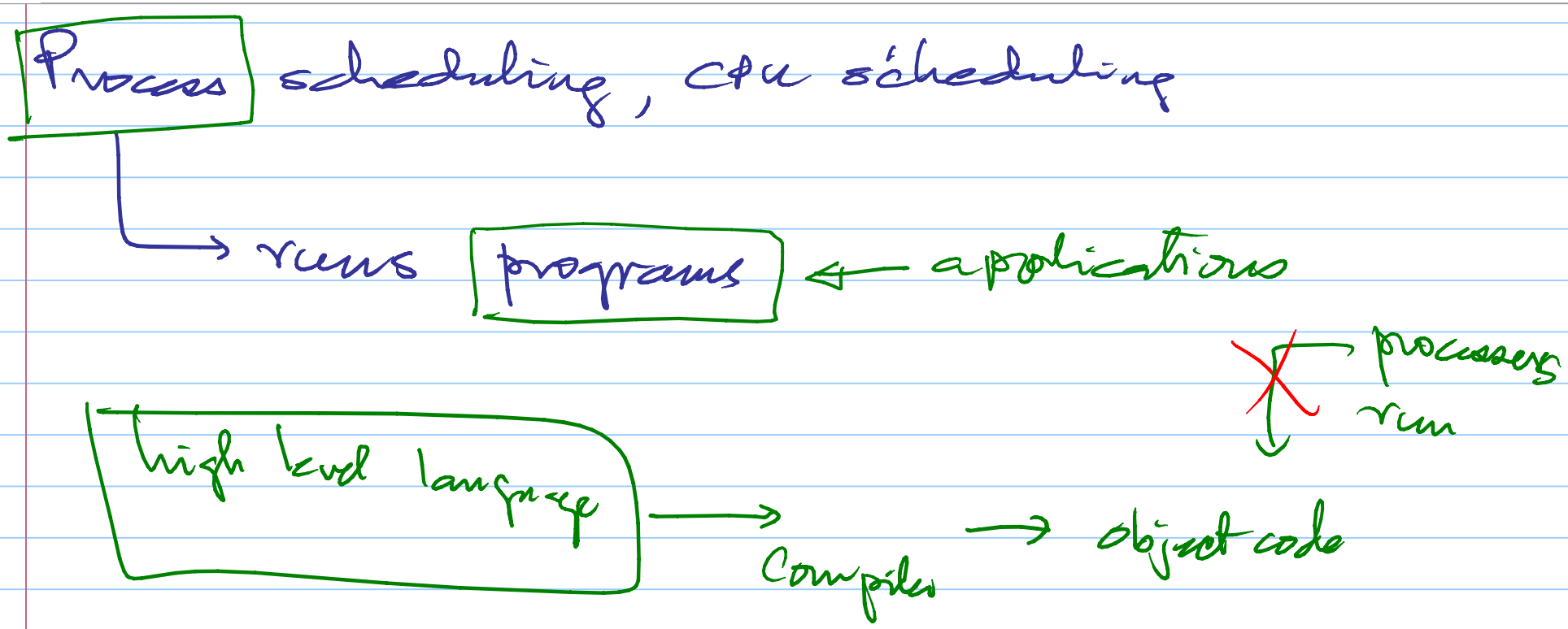
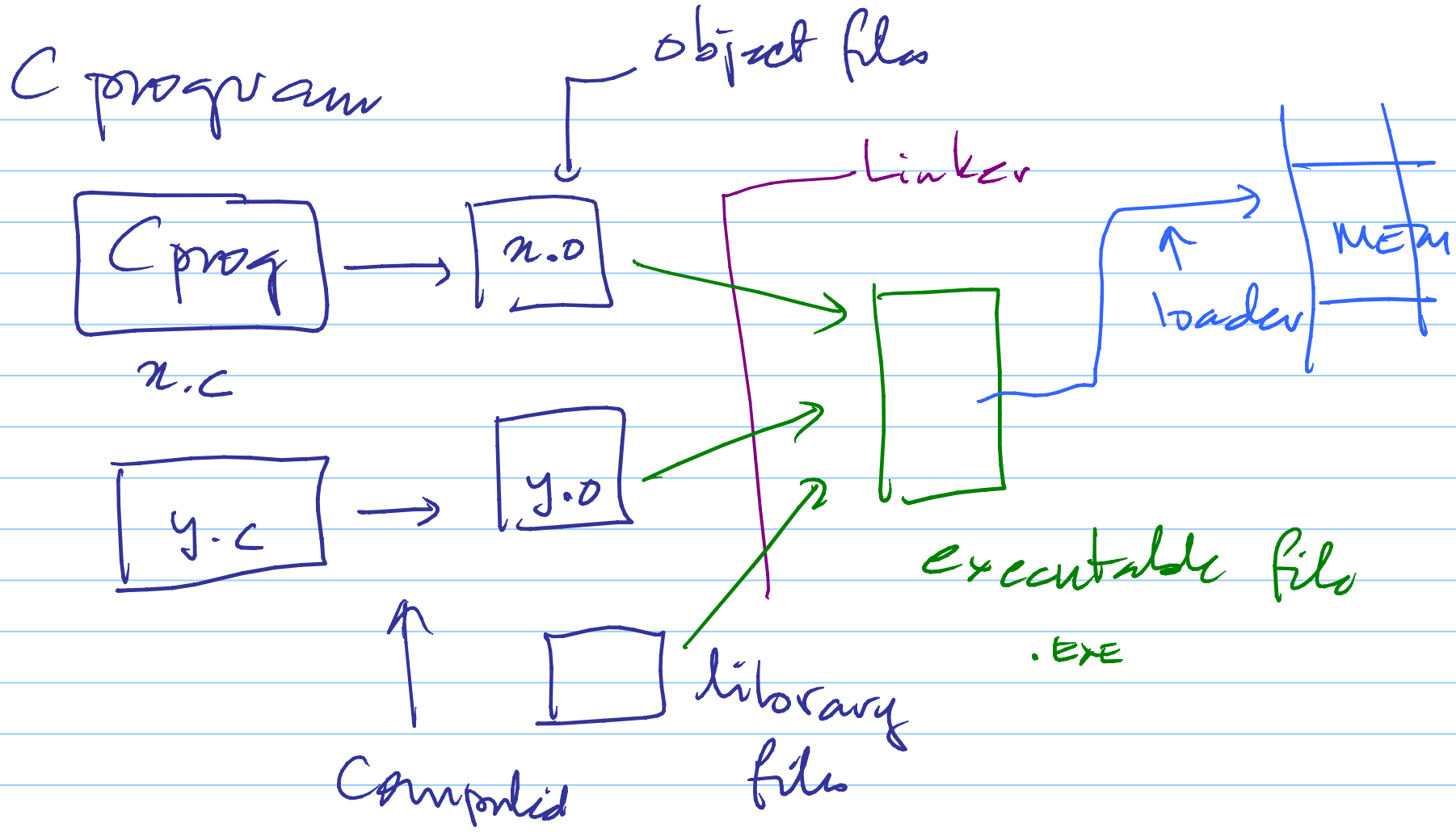
CSE 330: Operating Systems

Class: 08 **Date:** 2/1

Spring 2018

Note Title




C program
object files
Linker
Cprog
x.o
x.c
y.c
y.o
MEM
loader
executable file
.EXE
library
file
Compiler

x.c

```
int i
main()
{
    i=0
    i++
    printf("Hello\n")
}
```

y.c

addr

main → starts here

{ STO 0 → i ← addr

inc i

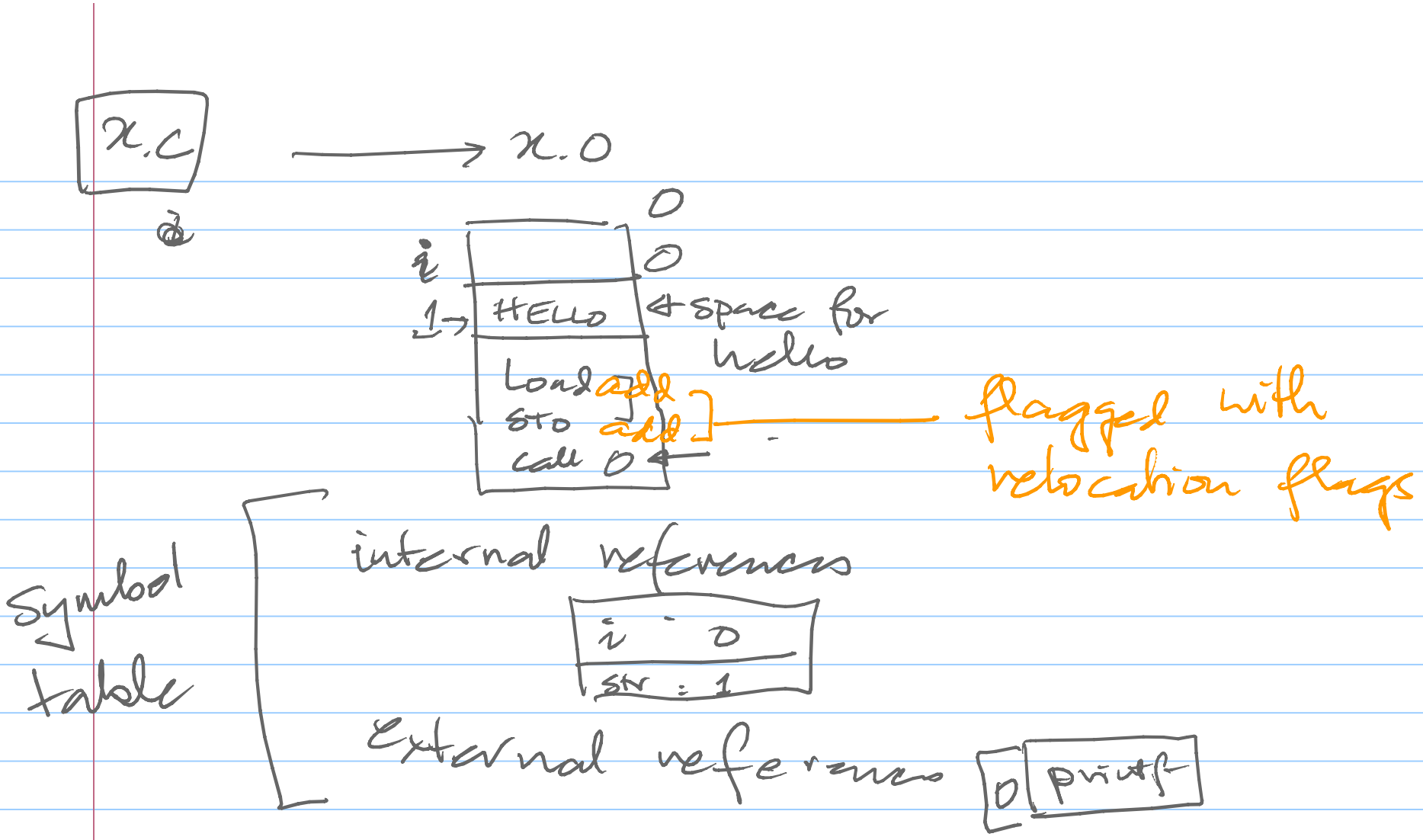
? → copy Hello → data memory

Call printf

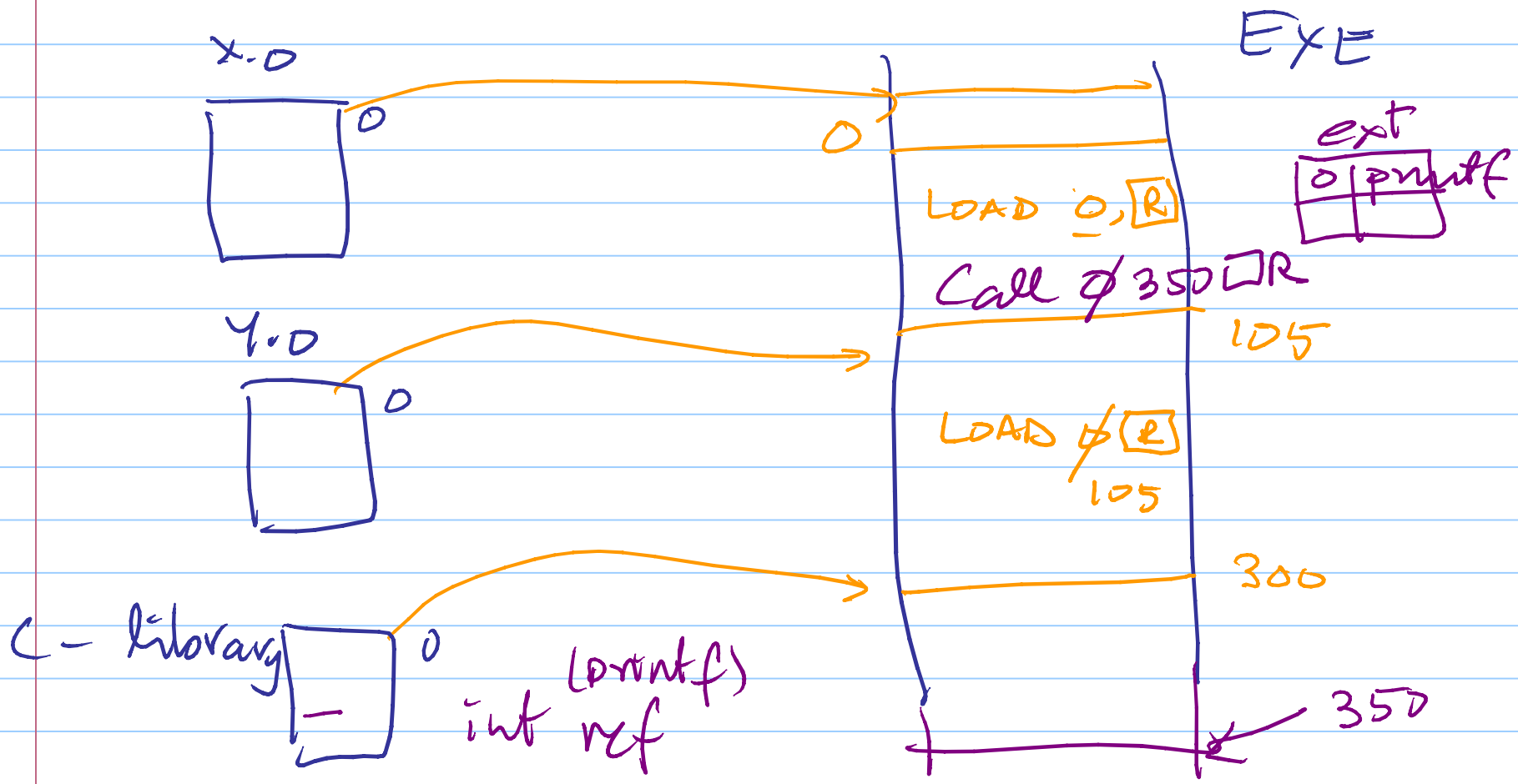
}

what?

where?


x.c
x.o
0
i
0
1
HELLO
space for hello
Load add
STO add
call 0
flagged with relocation flags
Symbol table
internal references
i 0
STR 1
External references
0 printf

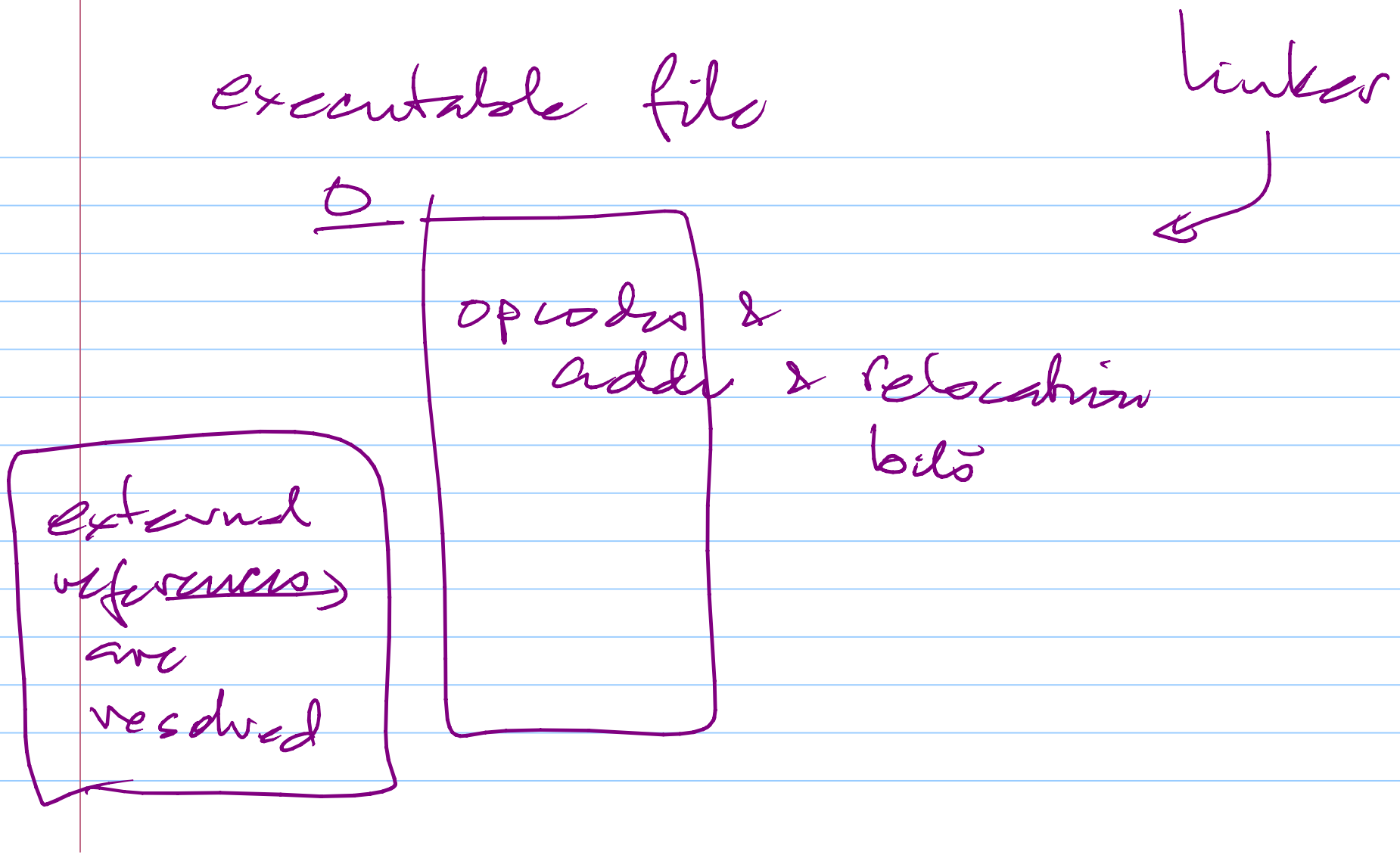
Linker (called ld in Linux)



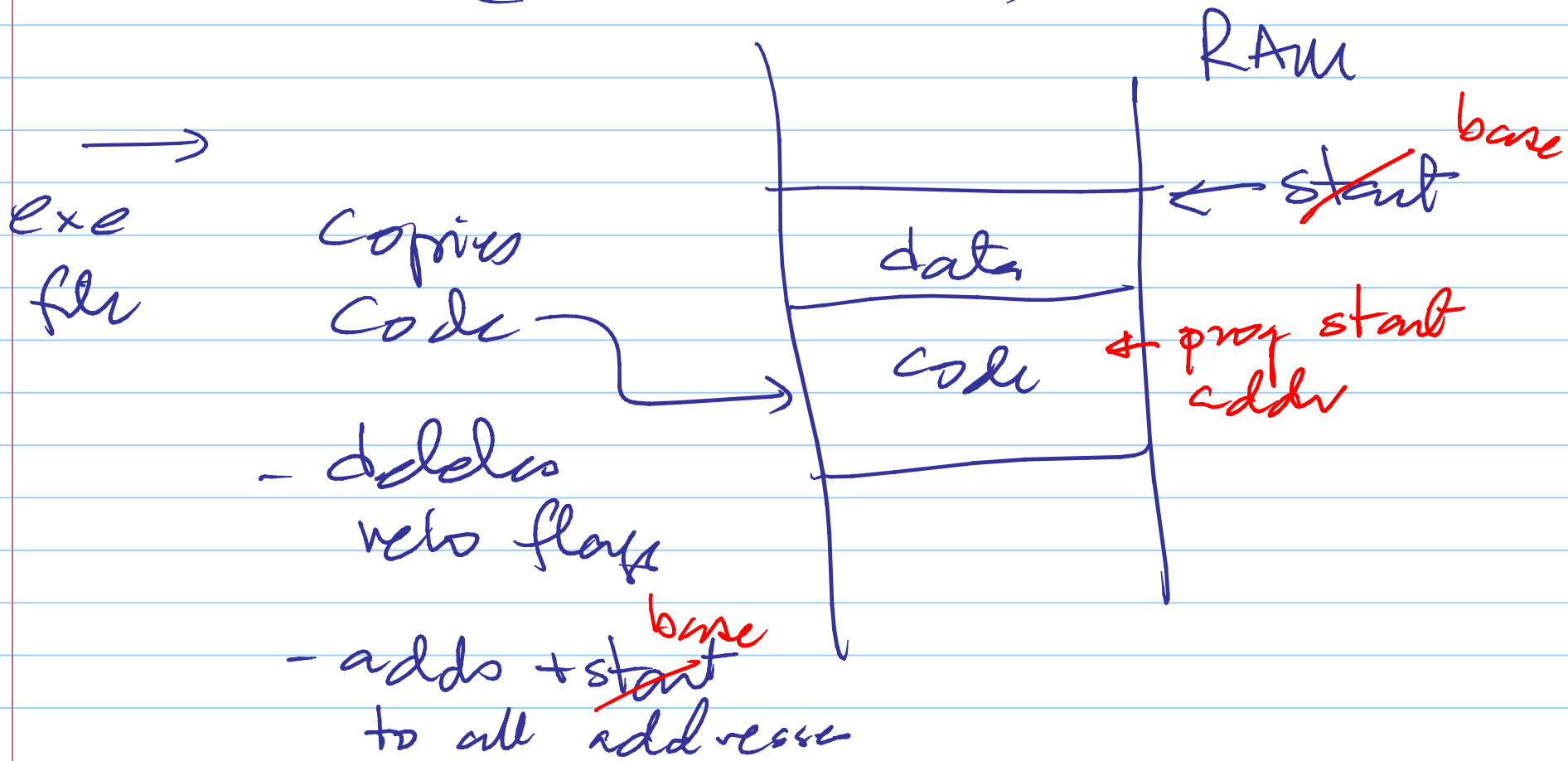
Object files contain

Compile

- code
  - addresses (relocation flag)
  - external references
- int ref
  - contains symbols & addresses names.
- ext ref
  - contains index & symbol names


executable file
linker
0
opcodes &
addr & relocation
bits
external
references
are
resolved

Loader (in kernel)



loader

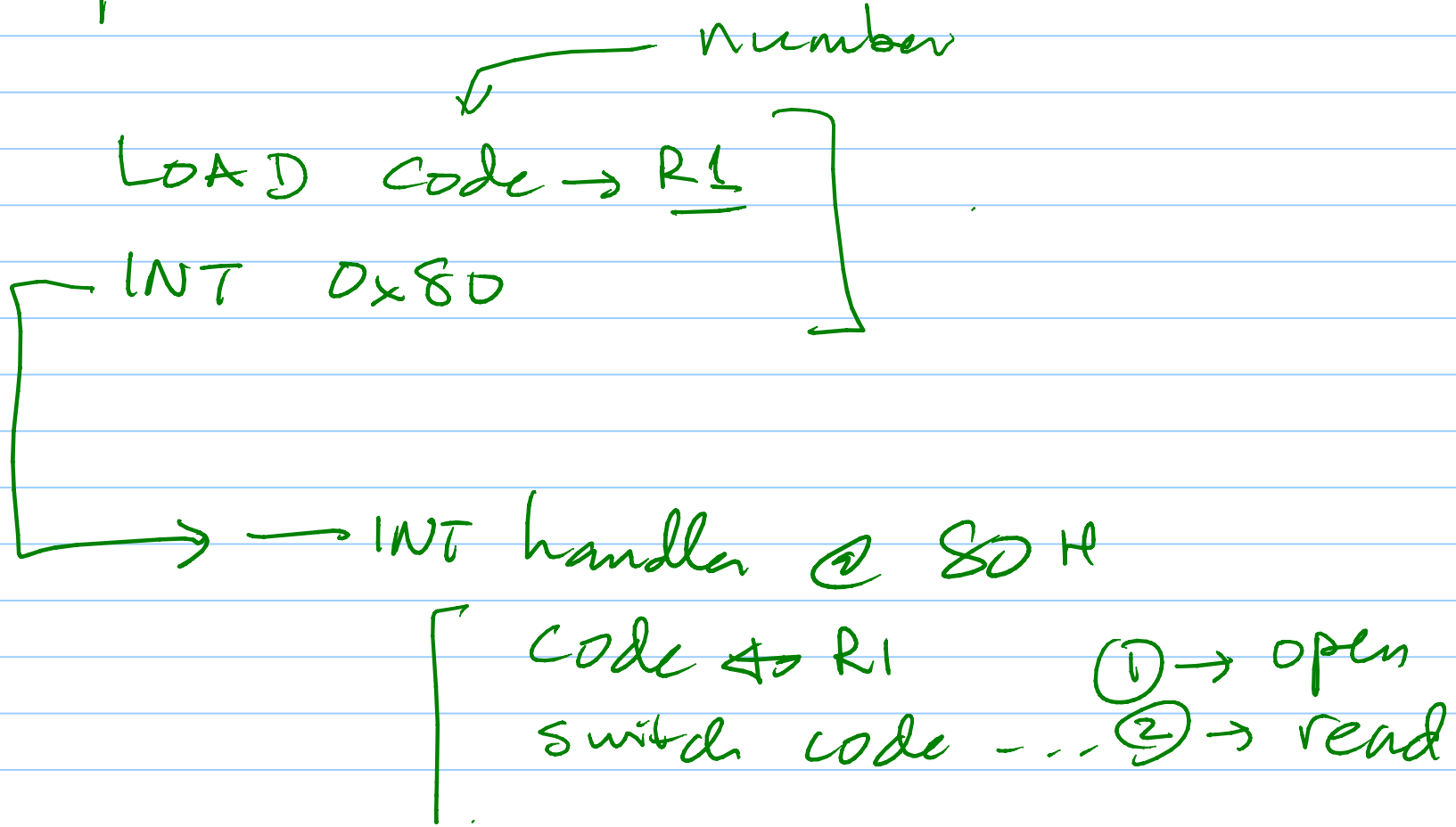
→ allocate enough memory
(base, limit)

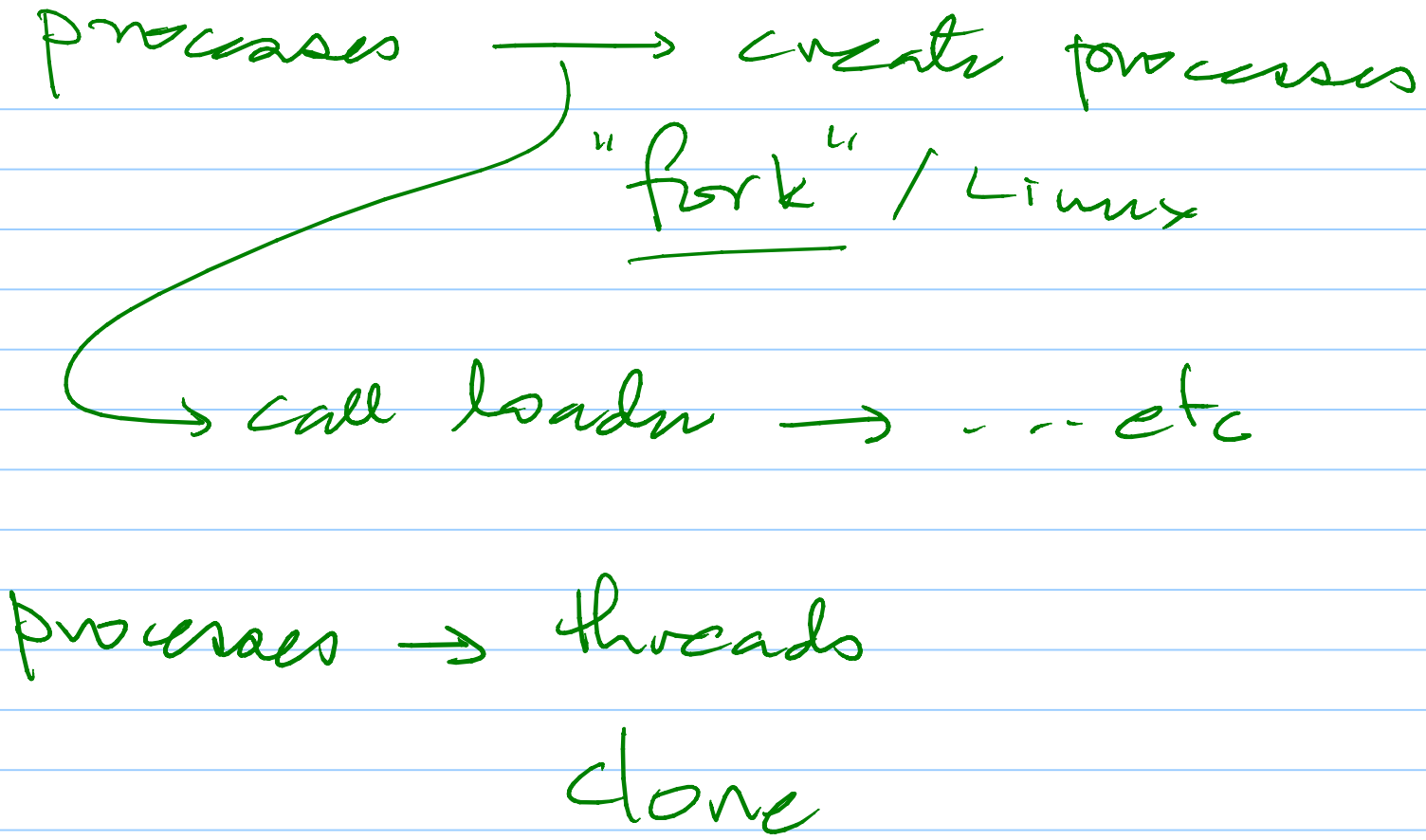
→ copies & relocates

→ creates PCB with stack ptr
base, limit & other info

→ init the stack → put PCB in sched Q

System call


number
LOAD code → R1
INT 0x80
INT handler @ 80 H
code ⇔ R1
switch code ... ① → open
② → read


processes
create processes
"fork" / Linux
call loader
... etc
processes → threads
clone

threads in Linux

→ library call pthread

→ function pthread_create ( )

{ ≡

clone(—·)

}

clone (f) → no loader needed

↳ create a PCB

set up stack & other info

↳ init stack

start addr = addr of func

Process & threads.

own mem space
- code
- data // global
- stack
- heap

uses parent process mem space
- share code
- share data
- private stack (new)
- share heap

main
start a thread in f10
f1
f2
f3
f10
f11

kernel level threads aka kernel threads

↳ created by syscall

→ has PCBs in scheduler Qs

→ is like a process

User level threads — aka user thread

↓

- A process can create a "user level thread" without calling a system call

- the process has a scheduler & TCBs for its threads

Behavior of threads.

```
int i=0

f1 ()
    int j=0
  {  ∞ loop
          j++
          i++
          print ("f1", i, j)
  }

f2 ()
    int j = 0
  {    j++
       i++
       print ("f2", i, j)
  }
```

main()
start a thread
in f1
" f2

Γ1 thread

f1 1 1

f2 2 1

f1 3 2

f2 4 2

f1 3 3

f2 2 3

f1 4 4

f1 1 1

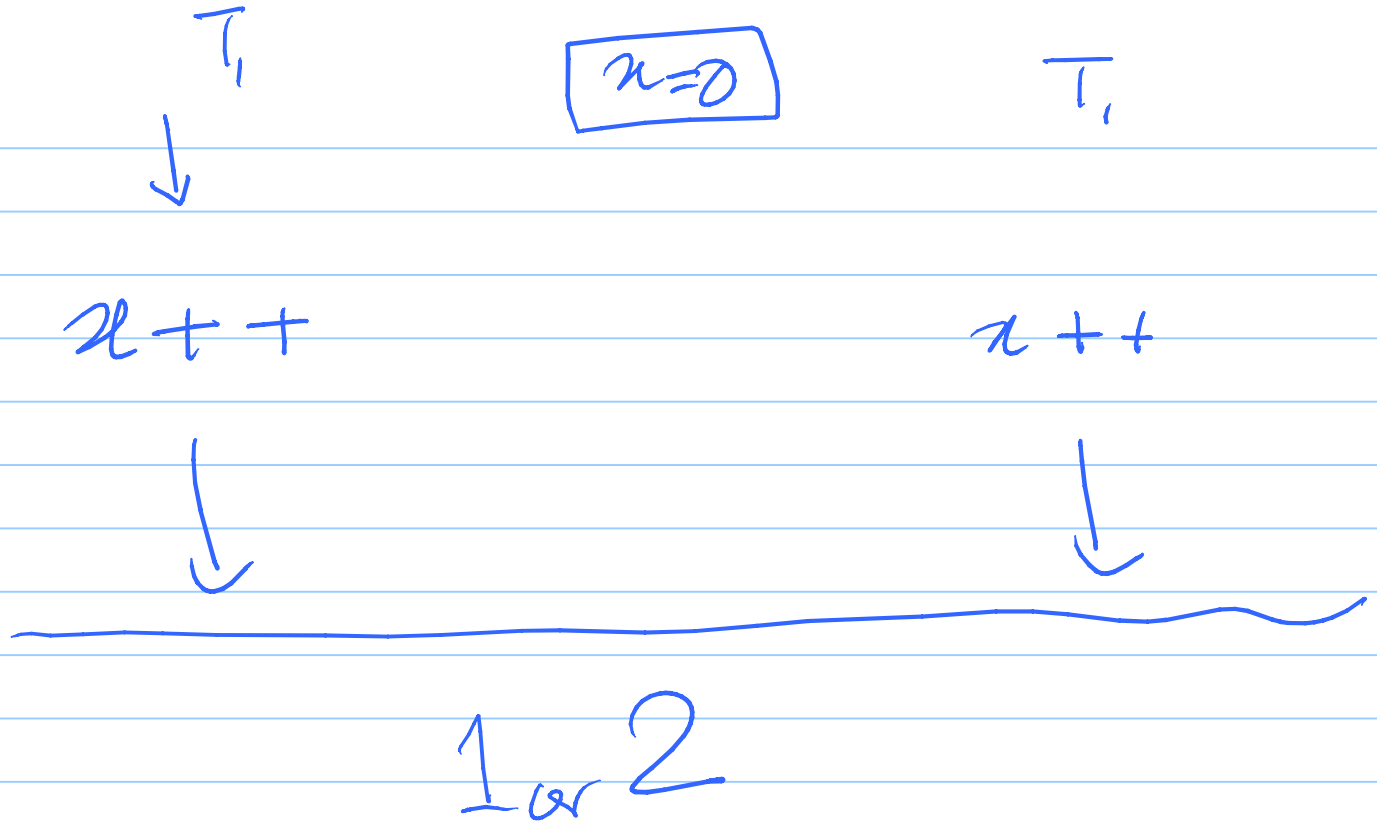
f1 2 2

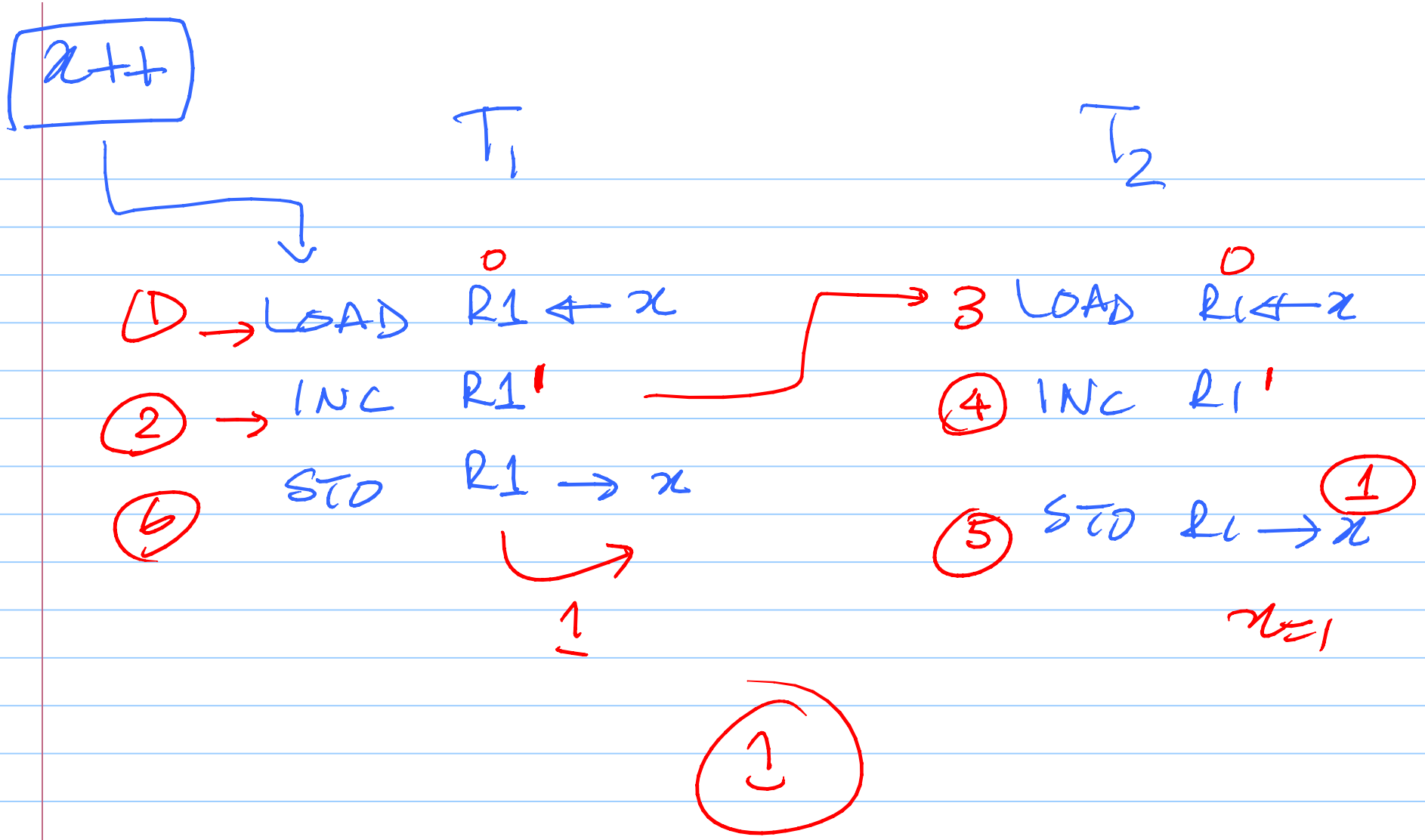
f1 3 3

f2 4 1

f2 5 2

f1 6 4


T1
x=0
T1
x++
x++
1 or 2


x++
T1
T2
0
1 LOAD R1 ← x
2 INC R1
3 STO R1 → x
0
3 LOAD R1 ← x
4 INC R1
5 STO R1 → x
1
x=1
1