CSE 330: Operating Systems

Class: 07 **Date:** 9/8

Fall 2016

Note Title

processes run programs

Program → composed of many source files.

f1.c

```
int i
p()
```

f2.c

```
int x
q()
```

f2.c

```
int y
r()
{ }
```

← data

← code

→ separate compilation → object file

data

code

int i

while (i < 20) { A; i--; }

f3( ) → call addr of f3

x:

A;

decrement i

goto x

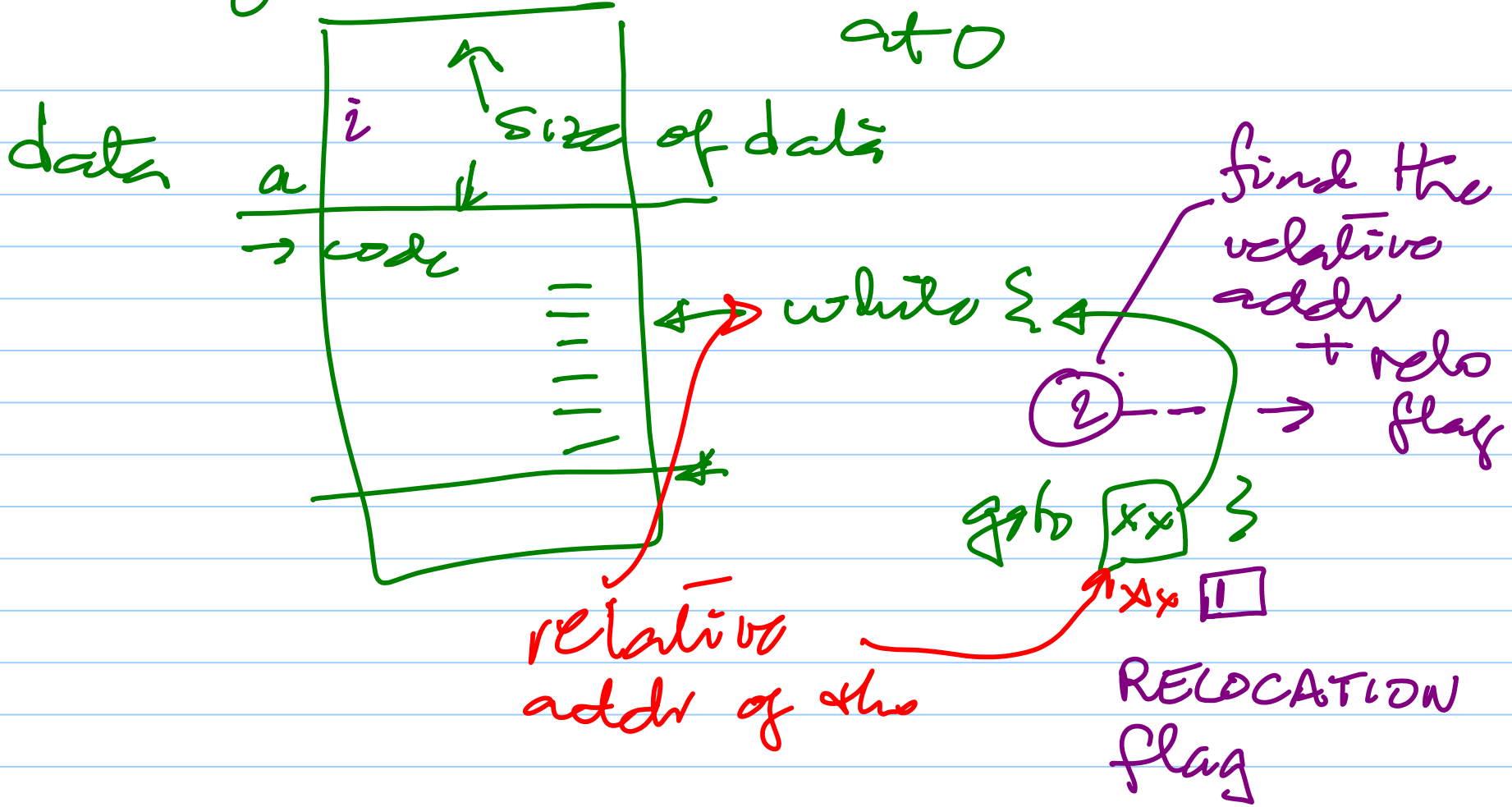
external reference

internal reference

addresses

not in source file

0 ← assume the module starts at 0



external references → ext1, ex2

references that external modules may use

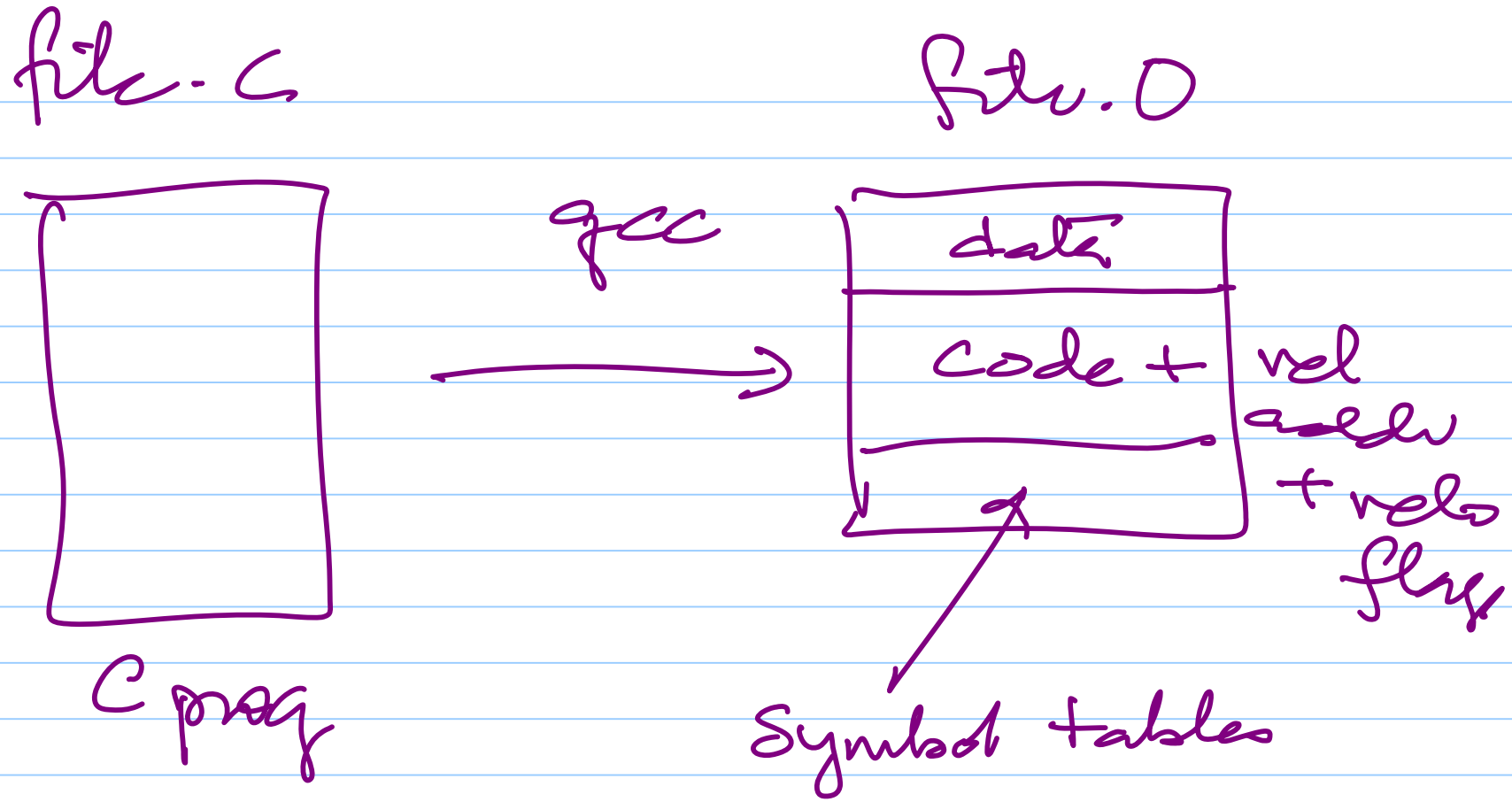
internal₁ $\boxed{\text{ext}}$ →

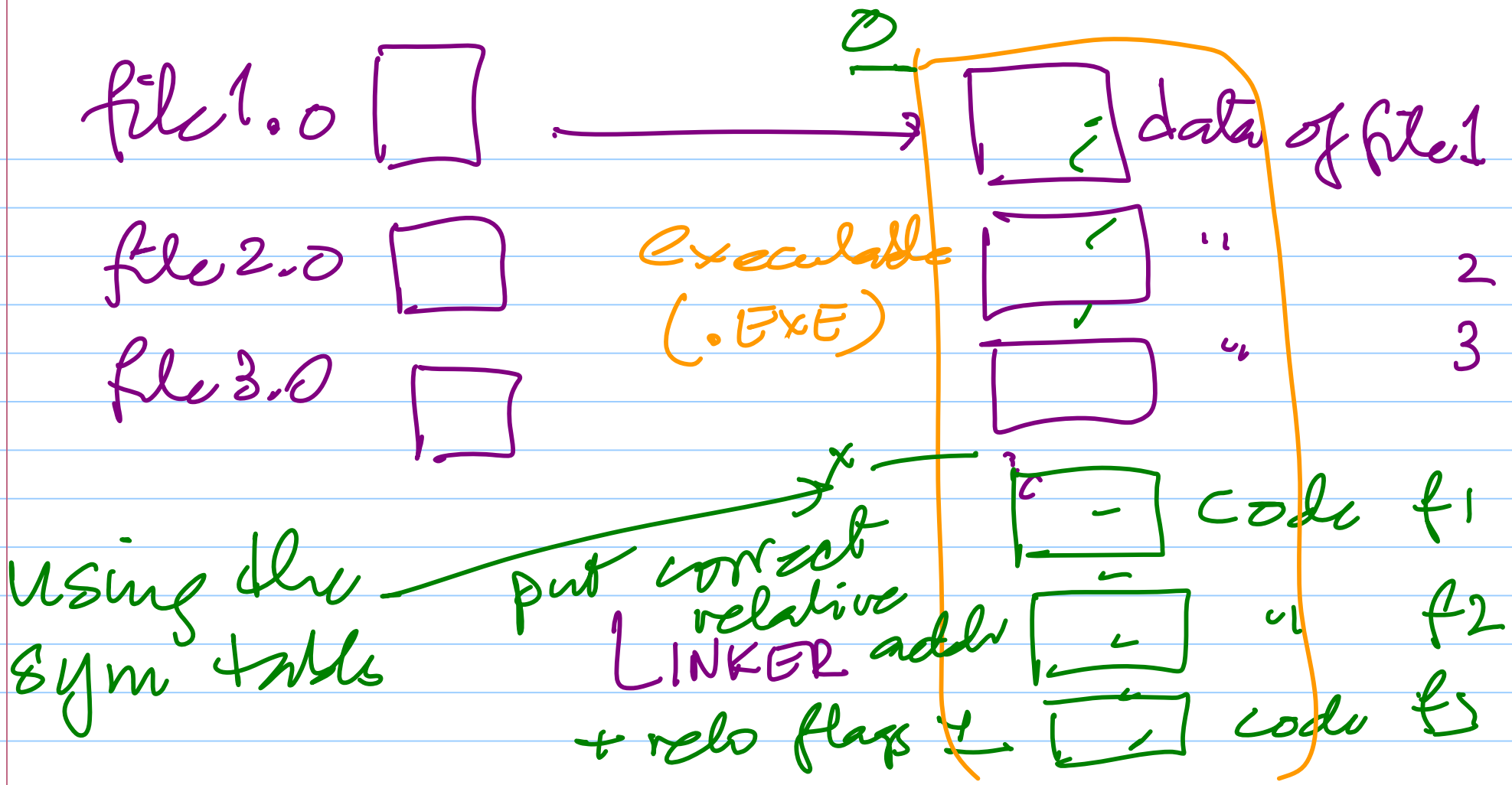
internal₂ → INTERNA

Symbol tables

ext1	—
—	—
1	add₁
2	add₂

addresses where ext1 is referenced

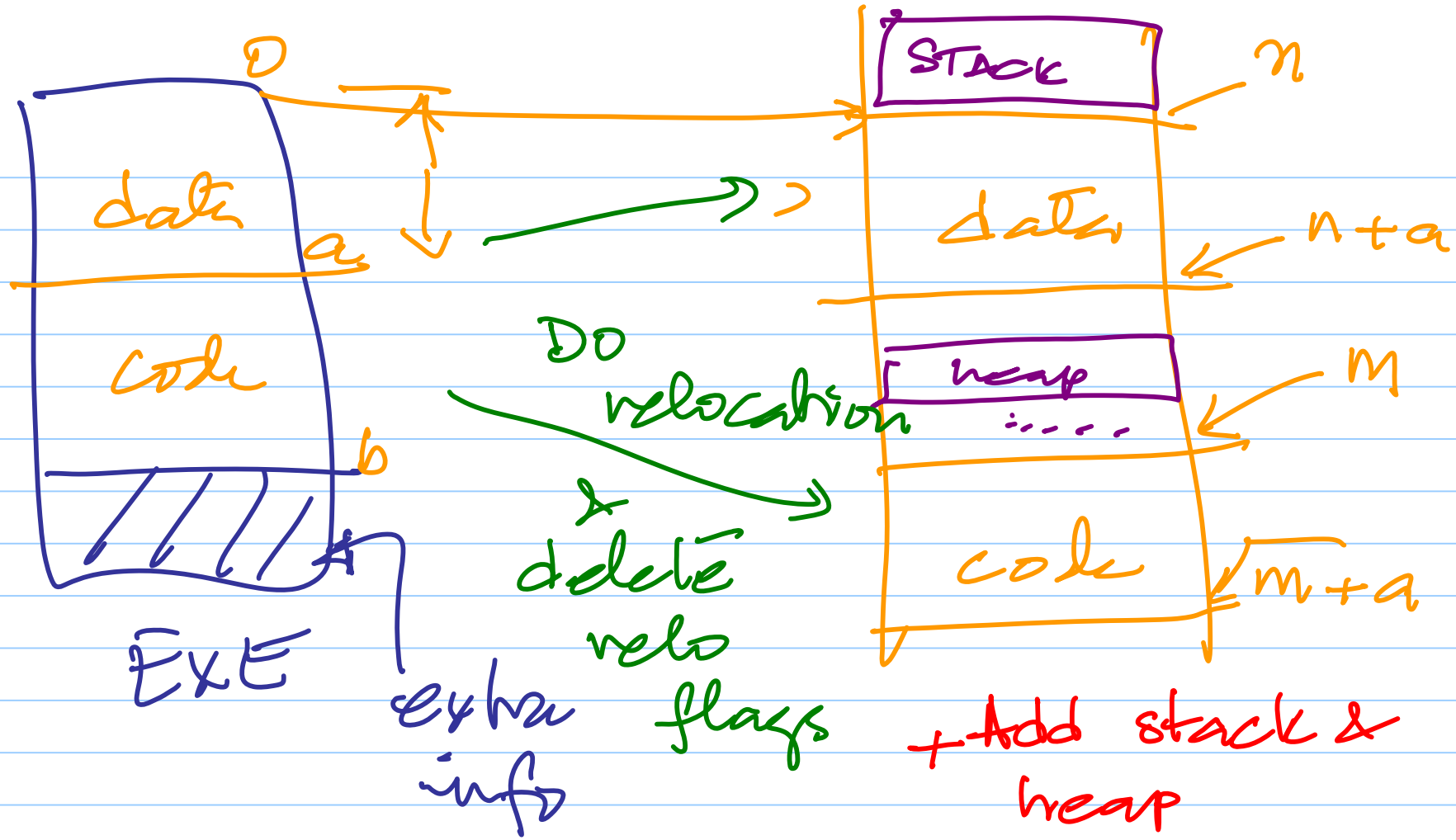

file.c
file.o
gcc
data
code +
reloc
addr
+ reloc
flags
C prog
Symbol tables


file1.o
file2.o
file3.o
executable
(.EXE)
0
data of file1
2
3
code f1
f2
f3
using the sym table
put correct relative
LINKER adds
+ reloc flags

C-prog → compile → .o files → link → .exe

.exe → used by OS to run a process

→ another program called the LOADER


0
data
a
code
b
EXE
extra info
DO relocation & delete reloc flags
STACK
n
data
n+a
heap
m
.....
code
m+a
+Add stack & heap

run the process

child

a process is started by

a process

parent

process

ask OS to run a process

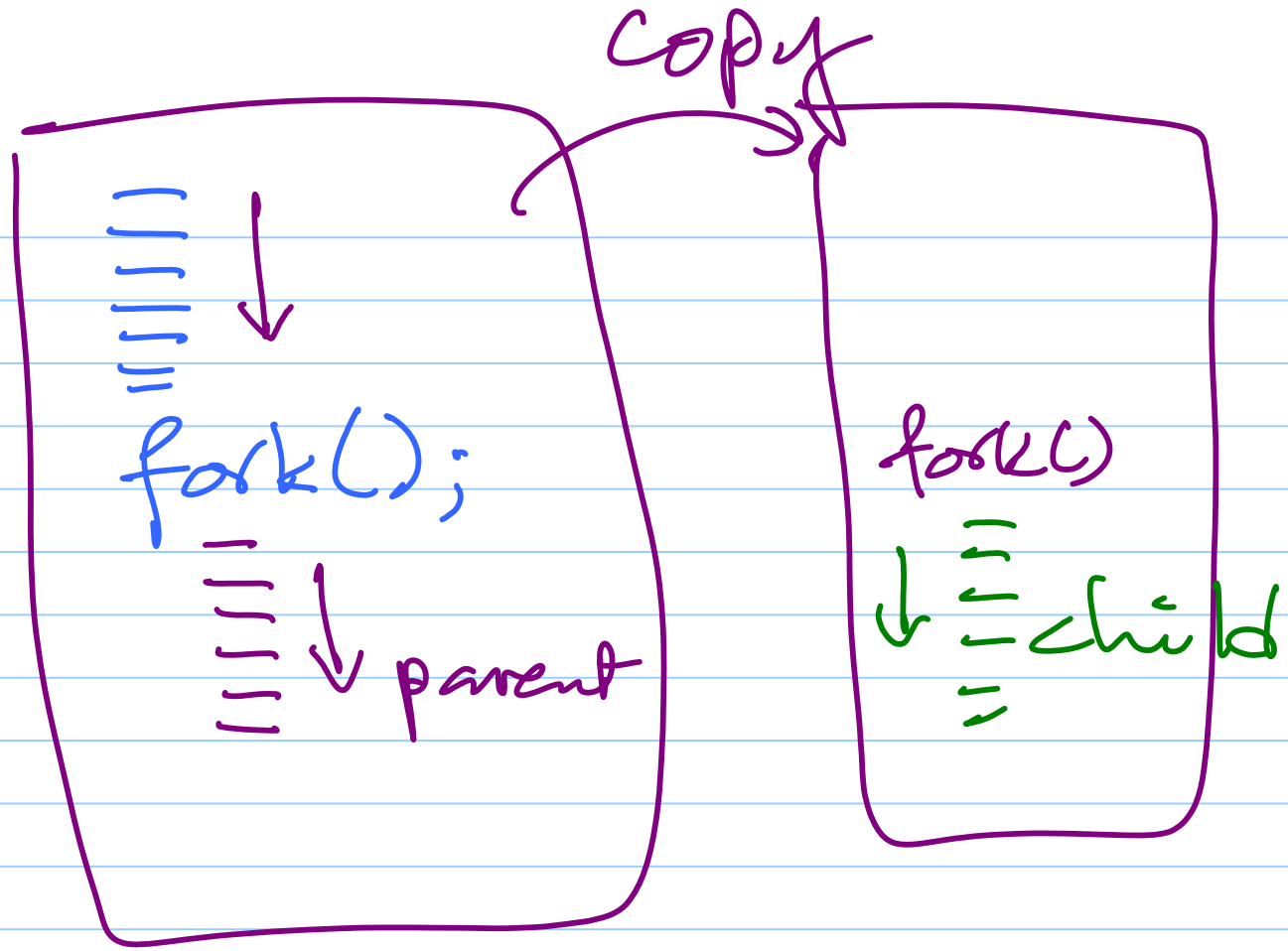
load memory

create a process control Block (PCB)

put the PCB into the [ready Q]

start addr = addr of main

UNIX



```
main()
{ printf("start");
  fork()
    printf("Hello world");
}
```

output → start
parent ;
child

```
main()
{ start
    if (fork==0) { f1(); }
    f2();
}
```

f1 → printf("child");
exit();

f2 → printf("parent");

fork creates processes

separate execution

UNIX → no threads

LINUX → thread creation done by clone()

WINDOWS → StartThread(...)

processes → [separate activity, share nothing]

threads → [cooperative activity share code, data & heap (not stack) with parent]

threads are useful

↳ one application has multiple concurrent activity

- computation speedup → overlap computation & I/O + multicore
- modularity
- responsiveness ↳ user interface
- economy

sharing memory

problem with threads

- race conditions → makes execution inconsistent or incorrect

dont step on me

- mutual exclusion or locking needed

related

speed control

- synchronization needed

also called lightweight processes

User threads & Kernel threads

→ Note terminology fail

User thread → a thread created at application level by a scheduler in the application

Kernel thread → a OS created thread

kernel thread → treat this as a CPU

↓

Build more threads out of 1 thread via a scheduler

↑

CPU allocated by the OS kernel

user threads execute when the kernel threads execute

1 → kernel thread blocks
→ all user threads block

2 → 1 CPU can be used.

1 kernel thread → n user threads

use multiple kernel threads

↳ m kernel threads → n user threads

$m \leq n$