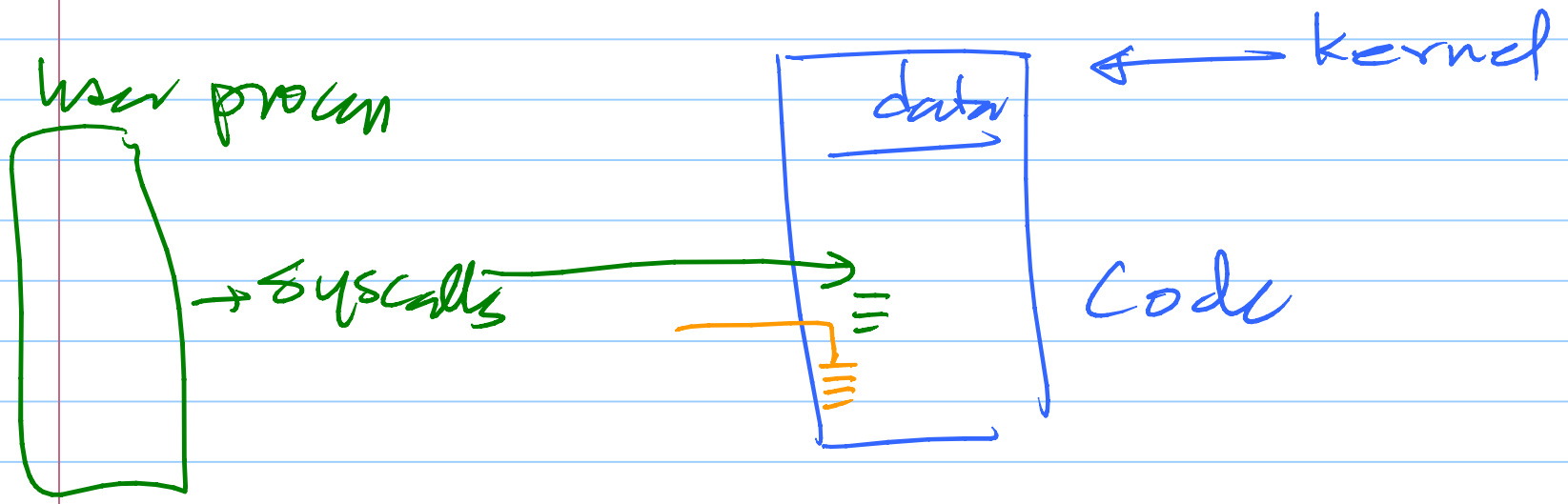
CSE 330: Operating Systems

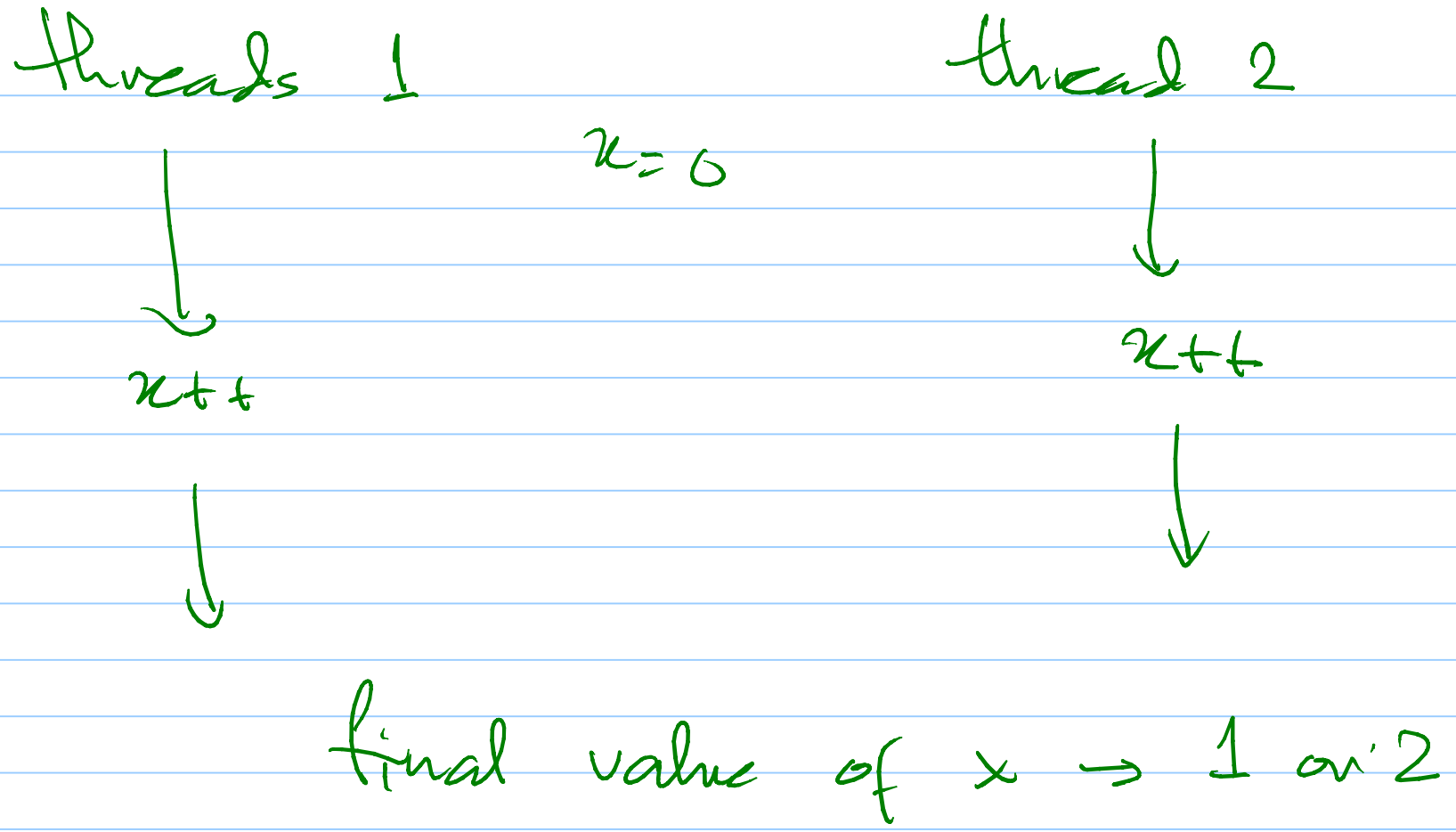
Class: 09 **Date:** 2/6

Spring 2018

Note Title

Concurrent executions





final value of x → 1 or 2

Race condition

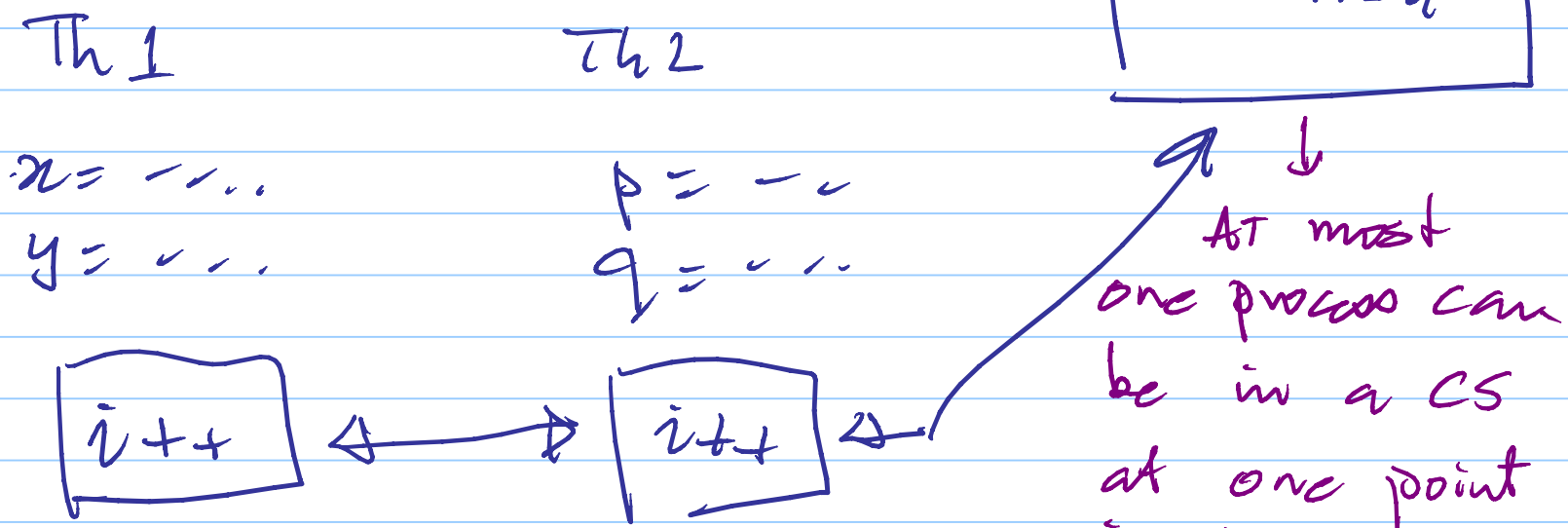
→ a condition where the speed of threads cause different results

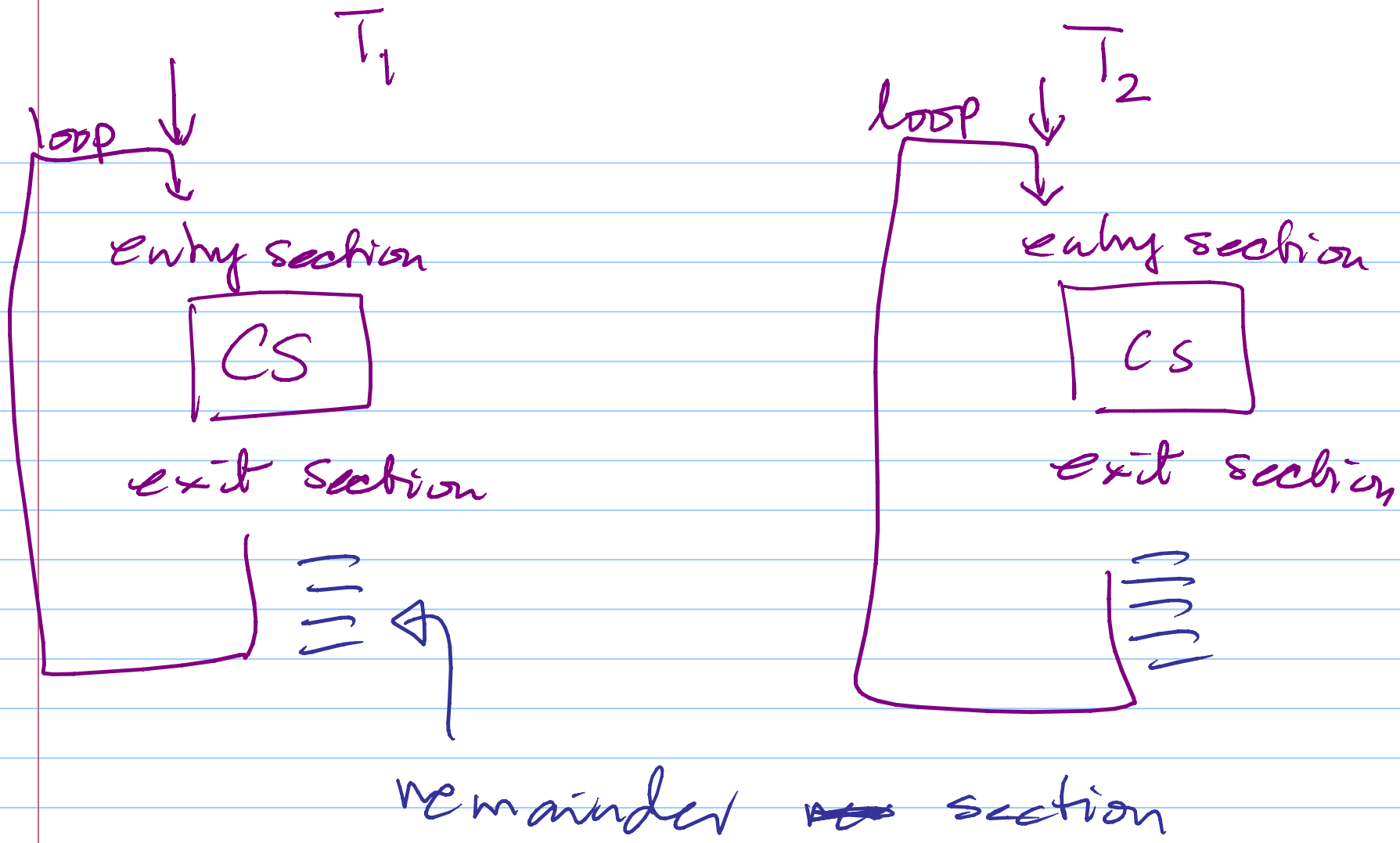
read - write conflict

write - write conflict

- Avoid race conditions

- Invent the concept of a **Critical Section**



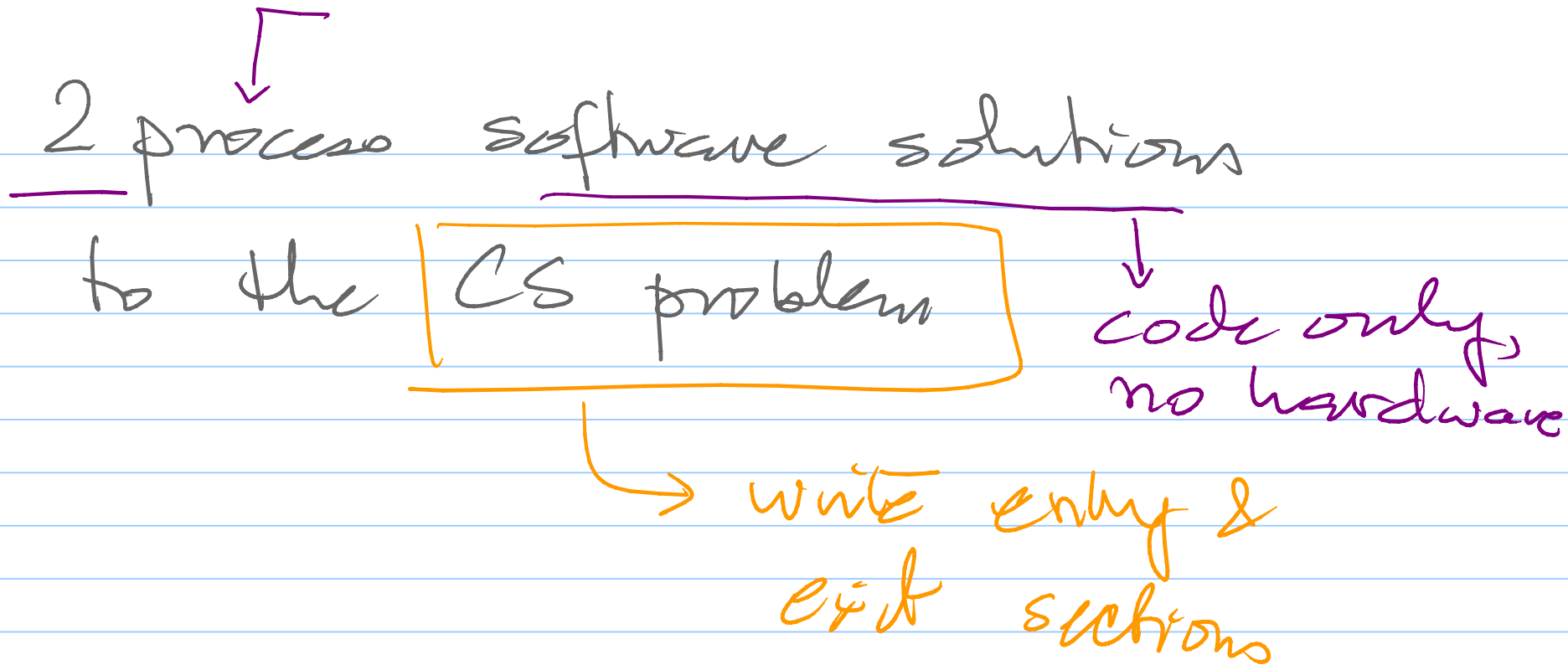

T1
loop
entry section
CS
exit section
remainder section
T2
loop
entry section
CS
exit section

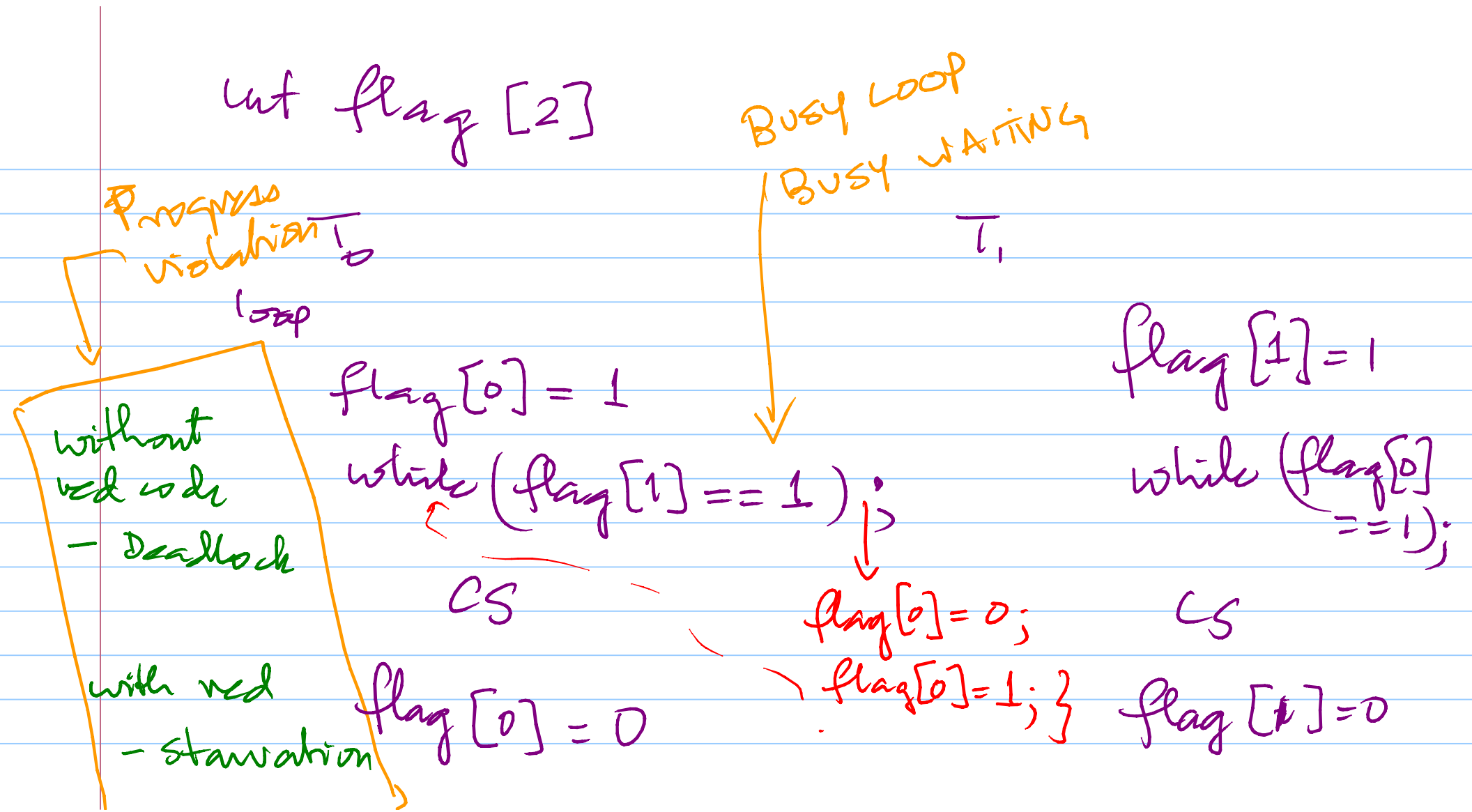
Critical Section properties

1. Mutual Exclusion → at most 1 thread may be executing in a CS
2. Progress – If no thread in CS and some threads want to enter then CS then those threads that do no want to enter should not participate in the decision — and the decision cannot be postponed indefinitely

③ Bounded waiting

if T_1 wants to enter & T_2 already in CS then there must be a bound on the number ~~time~~ of times T_2 can reenter before T_1 enters


2 process software solutions
to the CS problem
code only
no hardware
write entry & exit sections


int flag [2]
Progress violation
loop
BUSY LOOP
BUSY WAITING
T_0
T_1
flag[0] = 1
while (flag[1] == 1);
CS
flag[0] = 0
flag[0] = 0;
flag[0] = 1; }
flag[1] = 1
while (flag[0] == 1);
CS
flag[1] = 0
without sed code
- Deadlock
with sed
- starvation

while (flag[1] == 1);

flag[0] = 1

CS

flag[0] = 0

Mutual exclusion violation

Mutex → OK

progress → No

takes turns

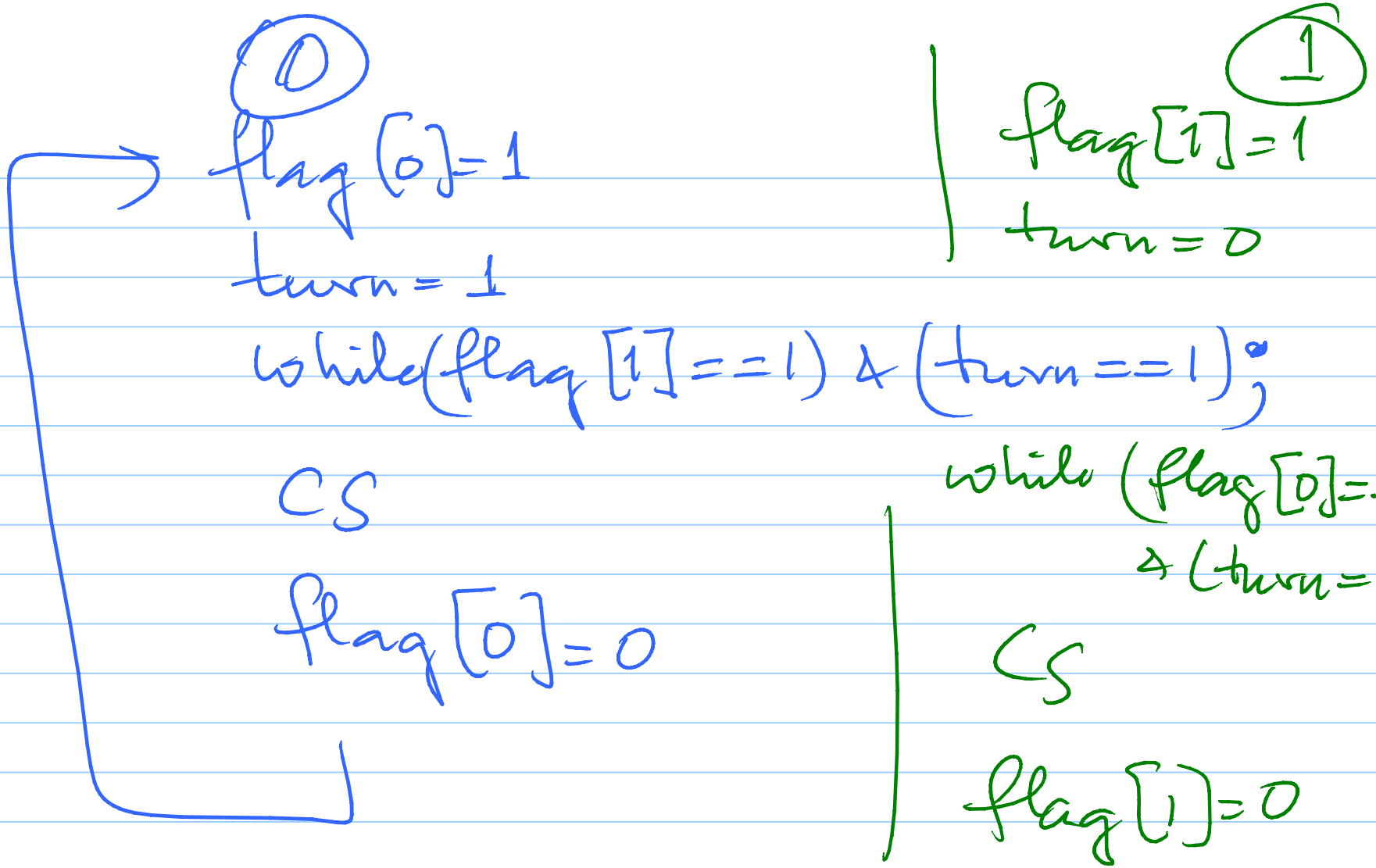
```
int turn = 0;

while (turn == 1);

CS

turn = 0;
```

```
while (turn == 0)

CS

turn = 1
```

- About 20 yrs → lots of wrong solutions

≈1965 Dekkers Solution
— 1st correct solution

1981 Petersons Solution


0
flag[0] = 1
turn = 1
while(flag[1] == 1) & (turn == 1);
CS
flag[0] = 0
1
flag[1] = 1
turn = 0
while (flag[0] == 1)
& (turn == 0);
CS
flag[1] = 0