CSE 330: Operating Systems

Fall 2016

Class: 08 **Date:** 9/13

Note Title

Threads

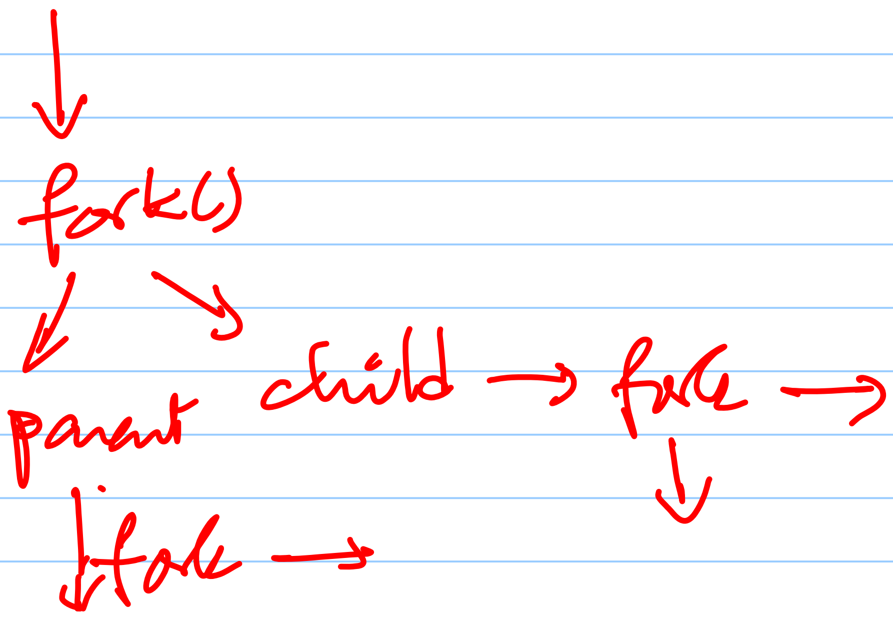
↳ activity (thread of control) + stack (private memory) + shared mem from parent

$\swarrow$

code, data, heap

Starting a thread

① fork (not unix fork)
+ join



$$\begin{tikzcd}
& X \arrow[dl] \arrow[dr, "fork"] & & \\
A \arrow[d] \arrow[dr] & & B \arrow[d] \arrow[dr] & \\
C & D \arrow[dr] & E & F \arrow[dl] \\
& & \text{join} \arrow[d] & \\
& & G &
\end{tikzcd}$$

← precedence graph.

A happens after X

B " " X

start thread, createthread

createthread (f)

function

f() {

start at this point in time

}

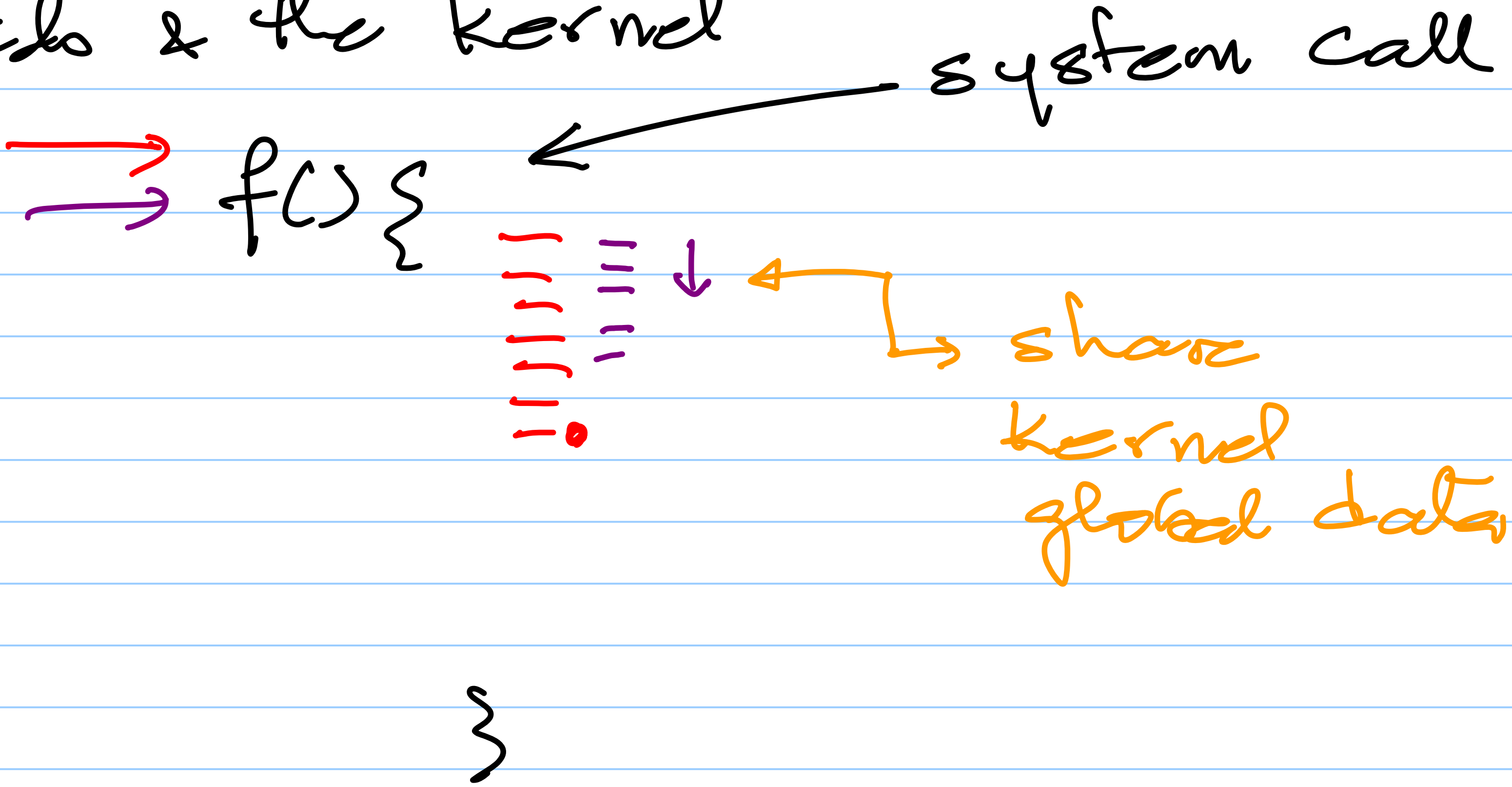
The 'par' construct

```
S0
par {   S1;
        S2;
        S3;   };

S4
```

$$\begin{tikzcd}
 & S_0 \arrow[dl] \arrow[d] \arrow[dr] & \\
S_1 \arrow[dr] & S_2 \arrow[d] & S_3 \arrow[dl] \\
 & S_4 &
\end{tikzcd}$$

Threads & the kernel



problem with threads...

→ (due to shared data)

race conditions

→ due to read-write conflicts

or write-write conflicts

```
f1() { int x;                          f2() {
  :                                       :
  :                                       :
  x = 5                                   x = 7
  • → what is the value of x?             :
}                                         :
                                        }
```

Both 5, and 7 are correct.

$\boxed{x=0}$

```
f1(){                f2() {
   ⋮                    ⋮
   x++                  x++
   ⋮                  }
}
```

same serial execution produces same results

not linearizable

incorrect correct

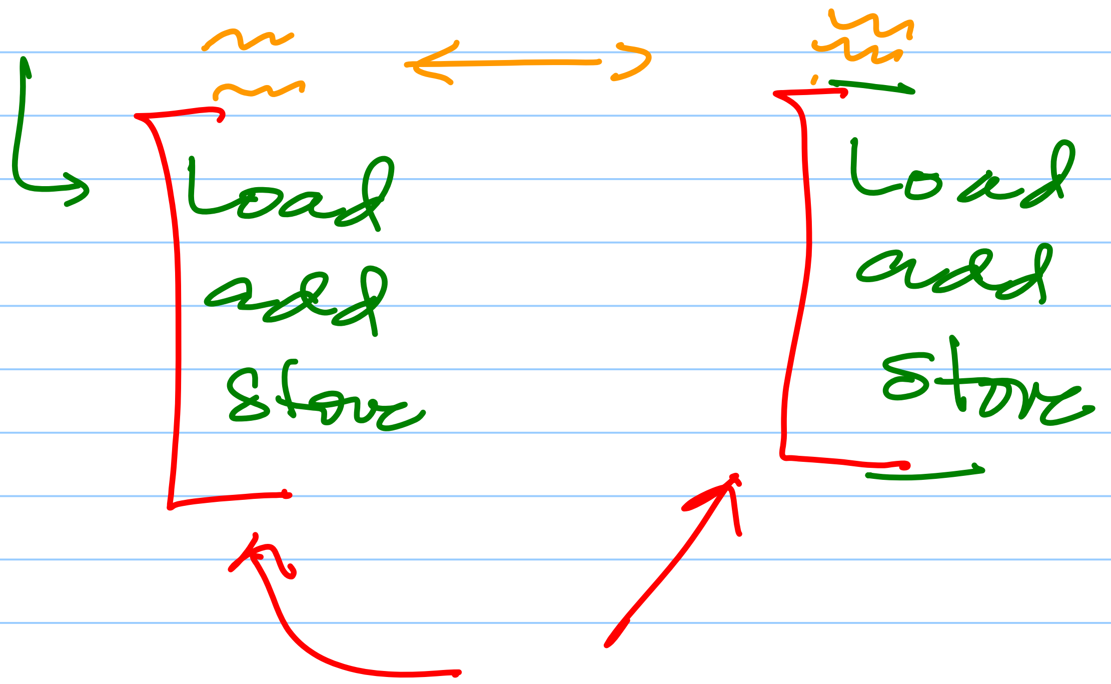
$x = 1$ or 2

$x{+}{+}$

not shared

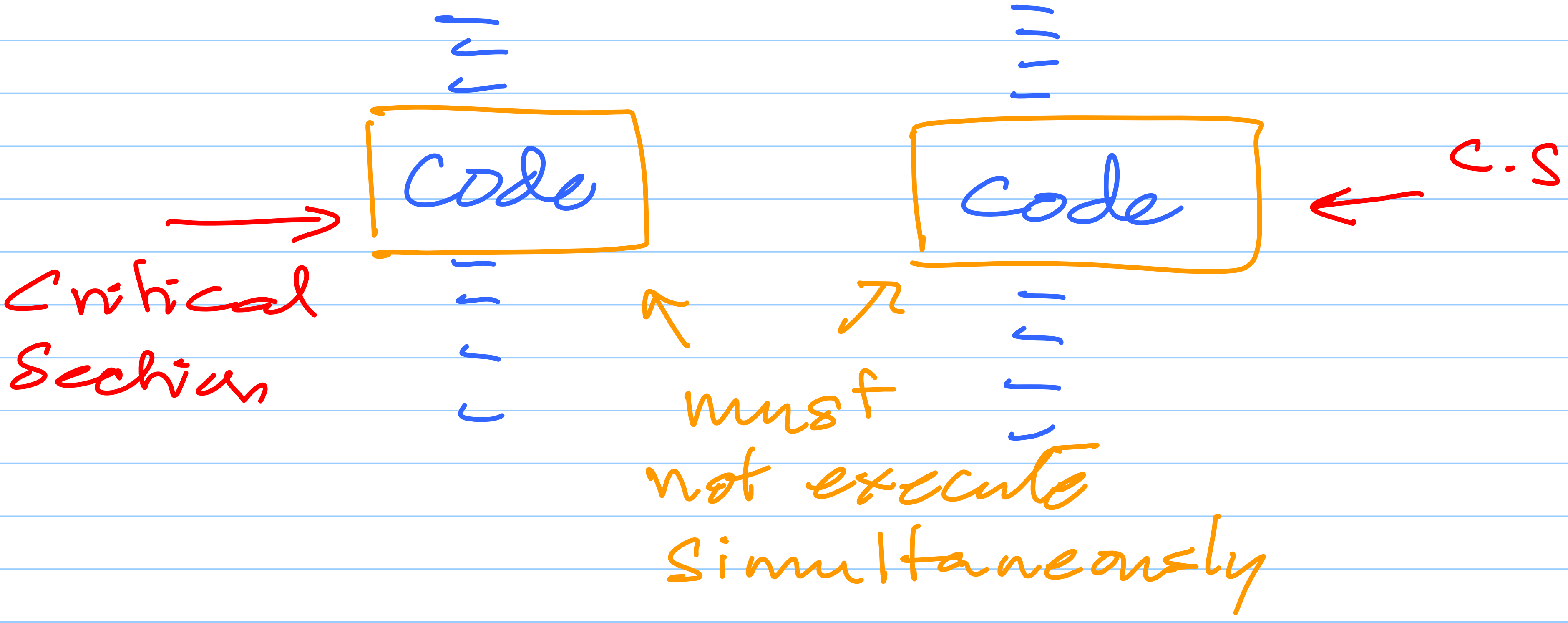
f1	f2
① LOAD $R_1 \leftarrow x$	② LOAD R_1
⑤ ADD $R_1, 1$	③ ADD R_1
⑥ STO $R_1 \rightarrow x$	④ STO R_1
$x=1$	$x=1$

$x{+}{+}$



make these atomic

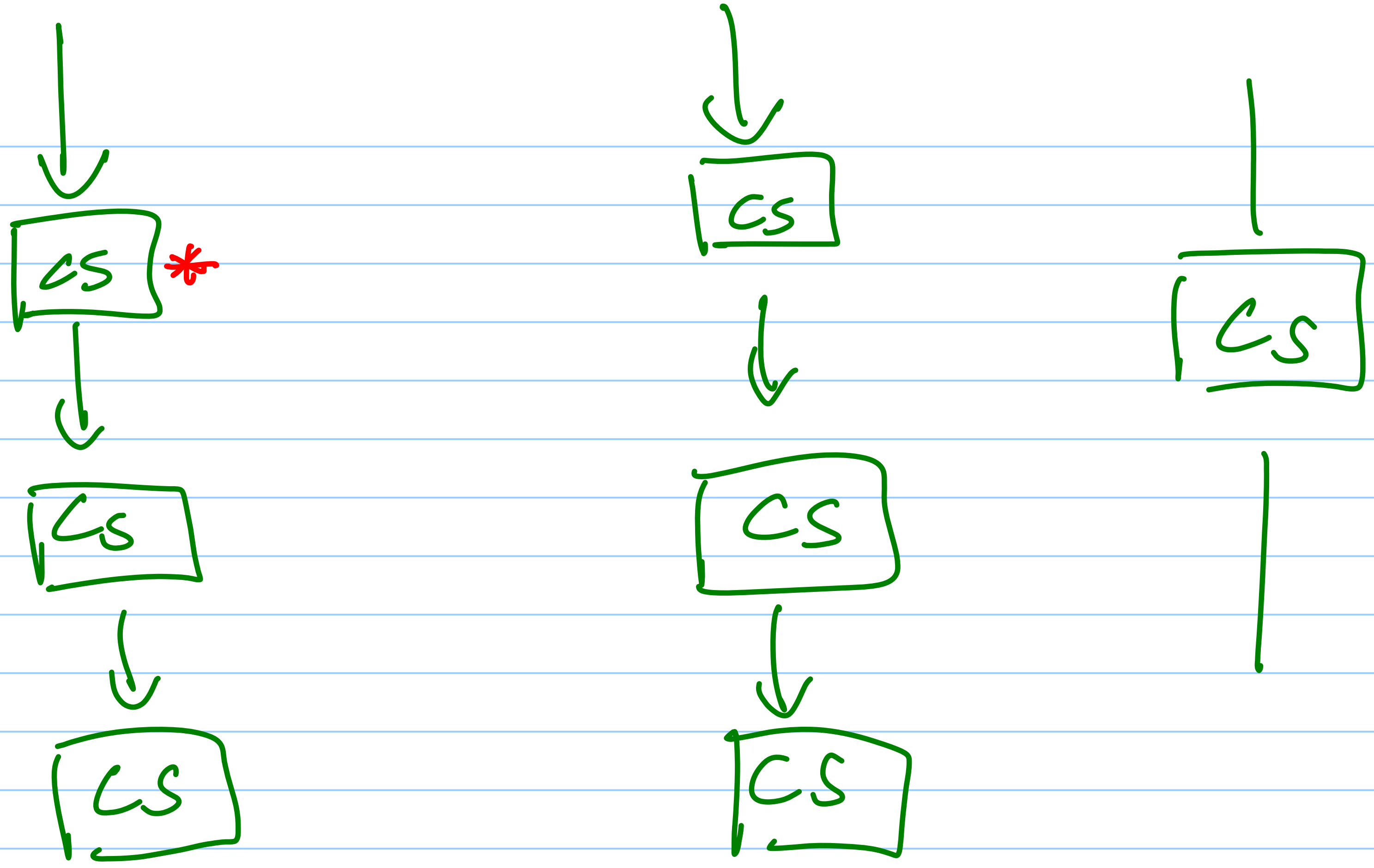
2-process (thread) Critical section problem.



properties of critical sections

1. mutual exclusion (mutex)

   when a process (thread) is executing in a c.s. no other process may execute inside any critical section


CS
*
CS
CS
CS
CS
CS
CS

- Progress
  - if a process wants to enter a critical section it must be allowed to enter if the section is not in use. ....

Bounded waiting

→ if a process wants to enter a CS it must no other process can be allowed to enter the CS more than $\underline{N}$ times.

$\uparrow$ fixed

2 process software solution

- entry section

    CS

- exit section

flag → 0, 1   initially 0

entry section → while(flag==1);
flag = 1

CS

exit section → flag = 0

```
flag[$\overset{I}{i}$] = 1                              flag[j]=1
while(flag[j] == 1){ flag=0 ;
                     flag=1 };
   CS

flag[i] = 0                                             flag[j]=0
```