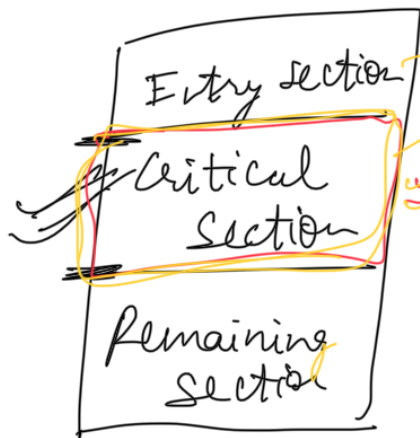
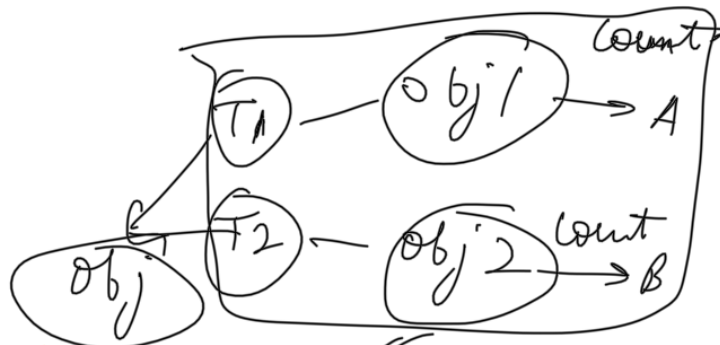


Critical Section



Shared resources are placed/accessed



```

{
    // ...
    int count = 0;
    {
        // ...
    }
    // ...
}
    
```

C.S.

Bal = 5000

T1

T2

R(Balance)

R(Balance)

5000

Withdraw(4000)

same

←

Parallel

W(Balance - 4000) W(Balance - 4000)

A	B
Read(x)	Read(x)
W(x + 800)	W(x - 300)
Read(x)	Read(x)

Depositing

Withdrawing

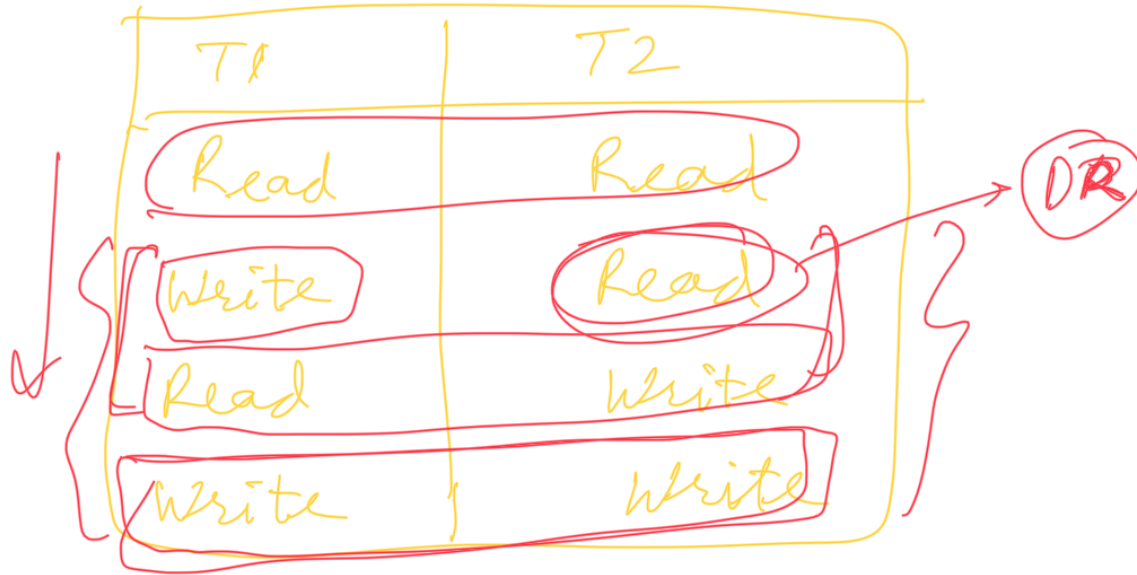
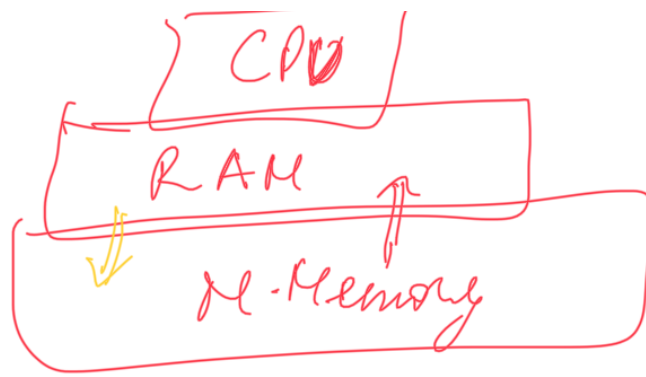
Bal X = 5000

5200

Central Memory

Thread Cache Memory

Synchronisation →



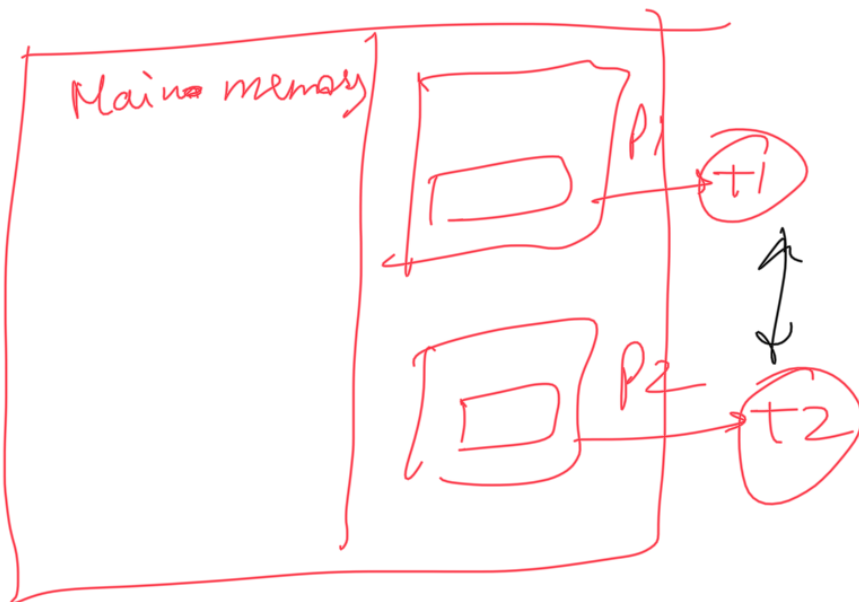
2 ways of making class thread safe

- ① Synchronized ~~key~~
- ② Using volatile keyword

Method or class instance can be used by multiple threads without any problem

shared object

static int count = 0



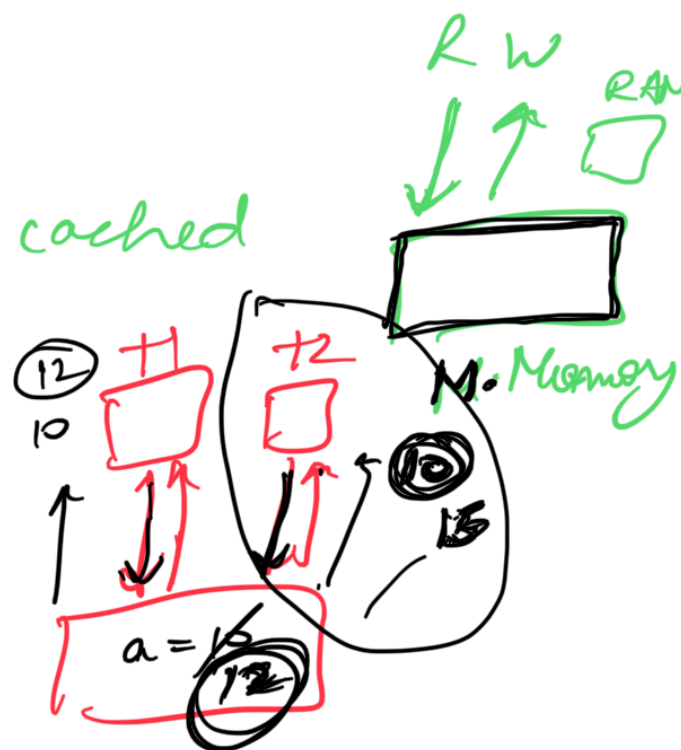
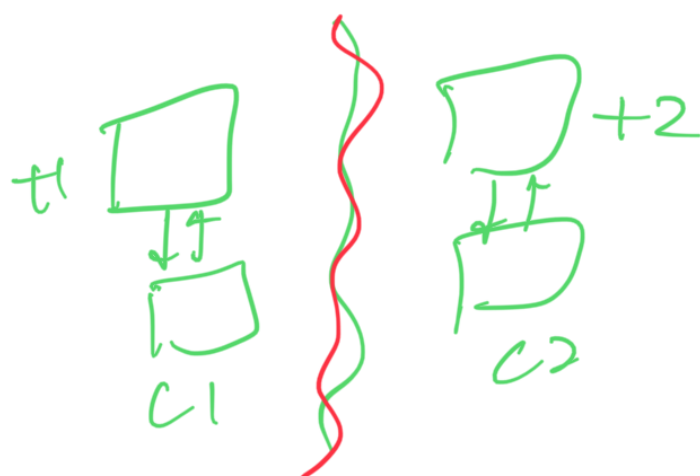
data inconsistency

① Mutual Exclusion - Only one thread at a time can execute a piece of code [C-Section]

② Visibility - Changes made by \uparrow are visible to other threads

Volatile keyword.

① Variable won't be cached

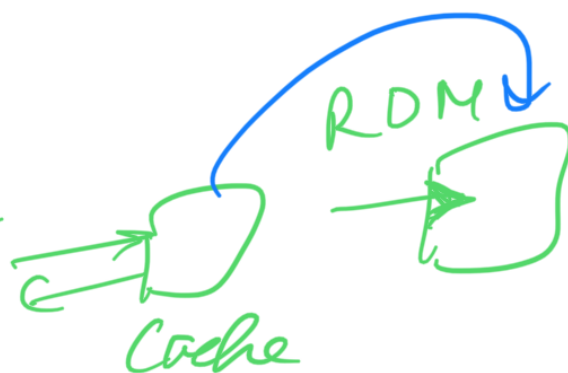


Tells the compiler that variable must not be cache

device drivers/
embedded
systems

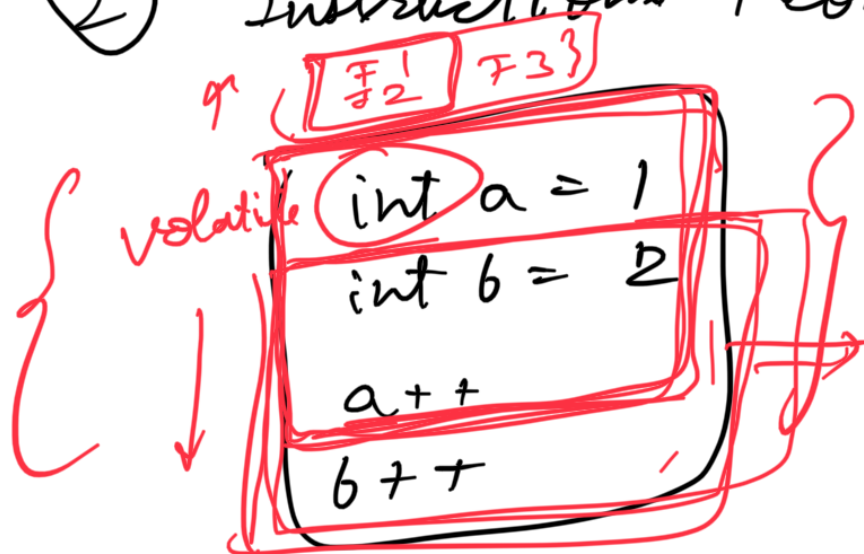
Cache - Optimisation technique

Transaction
RL



Don't do the optimisation

② Instructions Reordering



$\{F6, F7, F8\}$

JVM and CPU

`int a = 1`
`a++`

`int b = 1`
`b++`