SEMINAR 1

P1. Bank account

```
transfer(Account * src, Account * dest, int amount)
{
    (*src) -= amount;
    (*dest) += amount;
}
```

```
Struct Account {
    int balance; // initial value
    mutex mtx; // balance is constant while mtx is unlocked
}
```

V1

```
mtxglobal.lock();
src->mtx.lock();
dest->mtx.lock();
mtxglobal.unlock();
src->balance -= amount;
dest->balance += amount;
src->mtx.unlock();
dest->mtx.unlock();
```

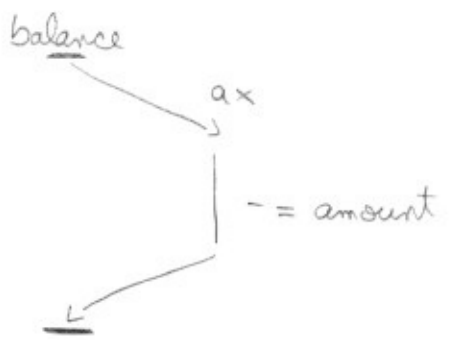
C++

- mutex: lock(), unlock()

C#/Java

```
lock(o){
...
}
```

-> every object is a mutex



Deadlock for V1

thread1	thread2
transfer(a, b, ..)	transfer(b, a, ..)
a lock	b lock
b lock	a lock

V2 Deadlock free -> cât mai puține operații ținute între lock/unlock

```
src->mtx.lock()
src->balance -= amount
src->mtx.unlock()
dest->mtx.lock()
dest->balance += amount
dest->mtx.unlock
```

Deadlock -> circular dependency, two threads are waiting one for the other, consuming same resources

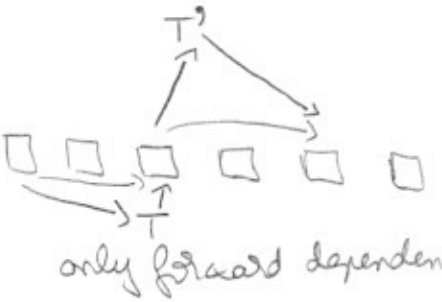
cycle in the graph

Dependency graph

mutexes;

```
if(src->id < dest->id){
    src->mtx.lock();
    dest->mtx.lock();
}
else{
    dest->mtx.lock();
    src->mtx.lock();
}
```

```
struct Account{
    int balance;
    mutex mtx;
    int id;
}
```



only forward dependencies

```
int sum(){
    int s=0;                                   vector
    for(int i=0; i<accounts.size(); i++){
        s+=accounts[i].balance  → accounts[i].mtx.lock();
    }                           → accounts[i].mtx.unlock();
    return s;
}
```

wrong

acc1 → 10
acc2 → 10

T1 → doing sum

s=0 sum()

acc[0].lock()

S+=acc[0].balance

=10 acc[0].unlock()

T2 → transfer

transfer(0,1,5)

acc[0].lock()
acc[1].lock()
acc[0]<-5
acc[1]<-15
acc[0].unlock()
acc[1].unlock()

acc[1].lock

S+=acc[1].balance

=25

5 units are counted twice → wrong value for the sum

CORRECT

```
int sum() {
    int s = 0;
    for(int i = 0; i < accounts.size(); i++) {
        accounts[i].mtx.lock();
        s += accounts[i].balance;
    }
    for(int i = 0; i < accounts.size(); i++) {
        accounts[i].mtx.unlock();
    }
    return s;
}
```

OR

the balance is constant for every version

Change transfer method so that a check for lock is done on resources:

try_lock -> in C++

```
src->mutex.lock()
while(!dest->mutex.try_lock()) {
    src->mtx.unlock();
    sleep(1);
    src->mtx.lock();
}
```

T1	T2
a.lock()	
	b.lock()
false <- b.try_lock()	
	a.try_lock() -> false
a.unlock()	
	b.unlock()
sleep()	
	sleep()
a.lock()	
	b.lock

then it repeats forever

livelock

it's unlikely to happen