

Banco

```
private Account[] contas
boolean depositar(int conta, double valor) {
    l.lock();
    contas[conta]+=valor;
    l.unlock();

    return true;
}

boolean levantar(int conta, double valor) {
    l.lock();
    contas[conta]-=valor;
    l.unlock();

    return true;
}

void transferir(int origem, int destino, double
valor){
    this.levantar(origem, valor);
    this.depositar(destino, valor);
}
```



contas[10]									
0	1	2	3	4	5	6	7	8	9
v	v	v	v	v	v	v	v	v	v

Lock da
instância Banco

```
void transferir(int origem, int destino, double valor) {  
    this.levantar(origem, valor);  
    this.depositar(destino, valor);  
}
```

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.levantar(conta1, 1000)

Cliente1

Cliente2

levantar(conta0, 1000)

saldo conta0 = 0; saldo conta1 = 0



levantar(conta1, 1000)

depositar(conta1, 1000)



```
void transferir(int origem, int destino, double valor) {
    l.lock();
    this.levantar(contaOrigem, valor);
    this.depositar(contaDestino, valor);
    l.unlock();
}
```

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.levantar(conta1, 1000)

Cliente1

Cliente2

lock(Banco)
 levantar (conta0, 1000)
 depositar (conta1, 1000)
unlock(Banco)

saldo conta0 = 0; saldo conta1 = 1000



levantar(conta1, 1000)



```
void transferir(int origem, int destino, double valor) {
    l.lock();
    this.levantar(contaOrigem, valor);
    this.depositar(contaDestino, valor);
    l.unlock();
}
```

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.levantar(conta1, 1000)

Cliente1

Cliente2

lock(Banco)

lock(Banco)

contas[0].levantar(1000)

unlock(Banco)

lock(Banco)

contas[0].depositar(1000)

unlock(Banco)

unlock(Banco)

saldo conta0 = 0; saldo conta1 = 1000



levantar(conta1, 1000)

```
void transferir(int origem, int destino, double valor) {
    l.lock();
    this.levantar(contaOrigem, valor);
    this.depositar(contaDestino, valor);
    l.unlock();
}
```

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.levantar(conta1, 1000)

Cliente 1

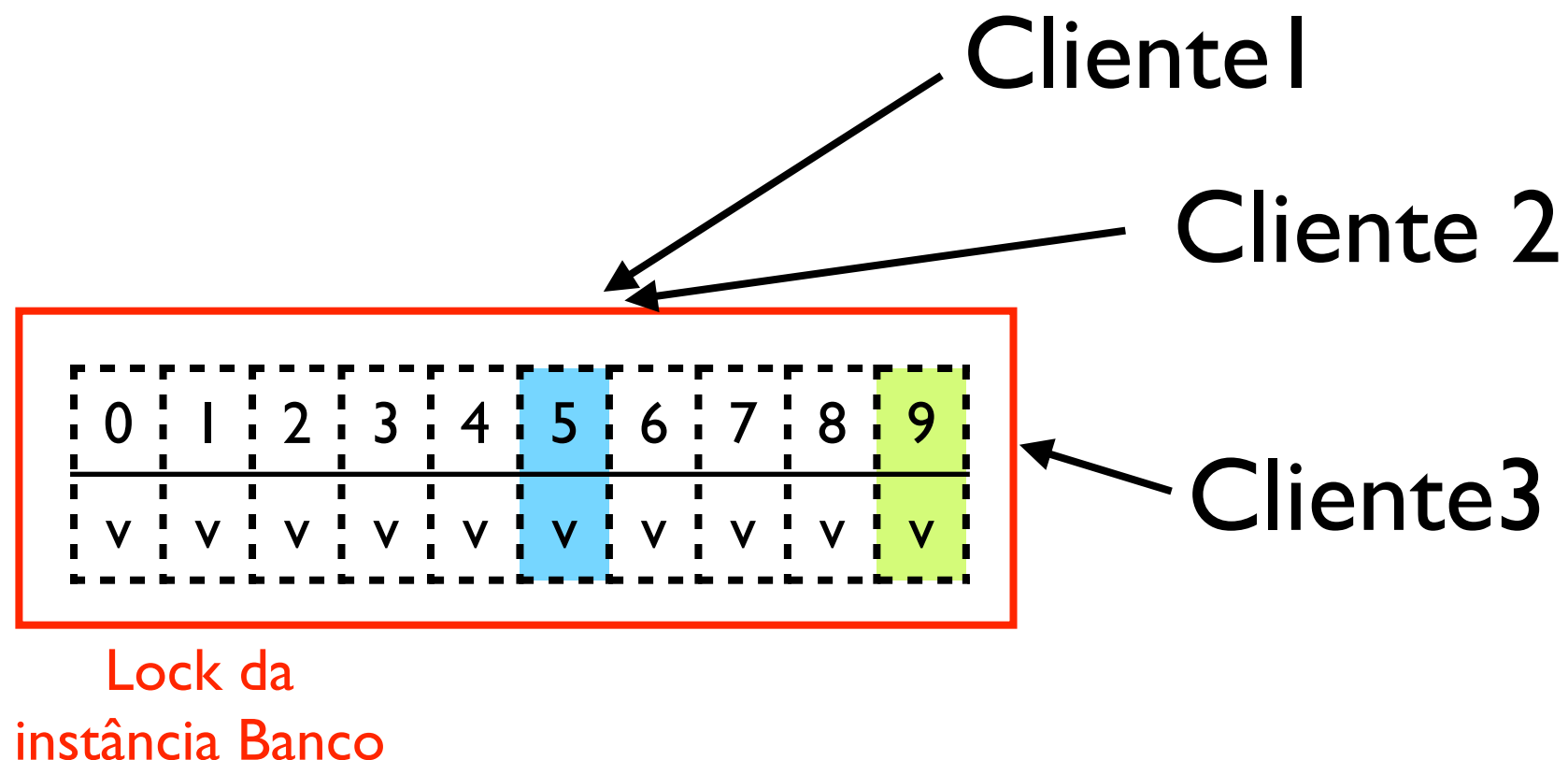
lock(Banco)
lock(Banco)
contas[0].levantar(1000)
unlock(Banco)

lock(Banco)
contas[0].depositar(1000)
unlock(Banco)
unlock(Banco)

ReentrantLock permite que uma thread entre em várias **secções críticas protegidas com o mesmo Lock.**

Granularidade de Exclusão Mútua

- Exclusão mútua ao nível do objecto Banco pode ser ineficiente:
 - Cliente 1 e 2 alteram o valor da conta 5, o mesmo recurso.
 - Cliente 3 altera o valor da conta 9, um recurso diferente.
 - Cliente 3 acede ao mesmo lock que Cliente 1 e 2.



Exclusão Mútua ao nível da Conta

- Tentativa: Lock em ambas as contas na classe Banco

```
public void transferir(origem, destino, valor){  
    contas[origem].lock();  
    contas[destino].lock();  
  
    try {  
        contas[origem].levantar(valor);  
        contas[destino].depositar(valor);  
    } finally {  
        contas[origem].unlock();  
        contas[destino].unlock();  
    }  
}
```

Exclusão Mútua ao nível da Conta

- Tentativa: Lock em ambas as contas na classe Banco

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.transferir(conta1, conta0, 1000)

Cliente1

```
lock(conta0)
lock(conta1)
levantar(conta0, 1000)
depositar(conta1, 1000)
unlock(conta1)
unlock(conta0)
```

Cliente2

```
lock(conta1)
lock(conta0)
levantar(conta1, 1000)
depositar(conta0, 1000)
unlock(conta0)
unlock(conta1)
```


Exclusão Mútua ao nível da Conta

- Tentativa: Lock em ambas as contas na classe Banco

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.transferir(conta1, conta0, 1000)

Cliente1

Cliente2

lock(conta0)

lock(conta1)

lock(conta1) *bloqueia...*

lock(conta0) *bloqueia...*

levanta (conta0, 1000)
deposita(conta1, 1000)
unlock(conta1)
unlock(conta0)

Deadlock!

levanta (conta1, 1000)
deposita(conta0, 1000)
unlock(conta0)
unlock(conta1)

Exclusão Mútua ao nível da Conta

- Tentativa: Lock em ambas as contas **com ordem** na classe Banco

```
public void transferir(origem, destino, valor){  
    contaMenorId = Math.min(origem, destino)  
    contaMaiorId = Math.max(origem, destino)  
    contas[contaMenorId].lock();  
    contas[contaMaiorId].lock();  
  
    try {  
        contas[origem].levantar(valor);  
        contas[destino].depositar(valor);  
    } finally {  
        contas[contaMaiorId].unlock();  
        contas[contaMenorId].unlock();  
    }  
}
```

Exclusão Mútua ao nível da Conta

- Tentativa: Lock em ambas as contas **com ordem** na classe Banco

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.transferir(conta1, conta0, 1000)

Cliente1

```
lock(conta_min)
lock(conta_max)
levanta (conta0, 1000)
deposita(conta1, 1000)
unlock(conta_max)
unlock(conta_min)
```

Cliente2

```
lock(conta_min)
lock(conta_max)
levanta (conta1, 1000)
deposita(conta0, 1000)
unlock(conta_max)
unlock(conta_min)
```

Exclusão Mútua ao nível da Conta


- Tentativa: Lock em ambas as contas **com ordem** na classe Banco

Cenário: conta0 = 1000, conta1 = 0

Cliente1: banco.transferir(conta0, conta1, 1000)

Cliente2: banco.transferir(conta1, conta0, 1000)

Cliente1



```
lock(conta_min) lock(conta0)
lock(conta_max) lock(conta1)
levanta (conta0, 1000)
deposita(conta1, 1000)
unlock(conta_max)
unlock(conta_min)
```

Cliente2

```
lock(conta_min) lock(conta0)
lock(conta_max) lock(conta1)
levanta (conta1, 1000)
deposita(conta0, 1000)
unlock(conta_max)
unlock(conta_min)
```