transactions - 1

introduction

- distributed systems: shared data & transactions
- transaction: sequence of operations as individual unit
- critical section

  short duration, indivisible group of instructions
- atomic operation

  isolated and free of interference from other concurrent operations
- transaction: extend critical section / atomic operation

  may contain operations on different servers

  possibly long duration

ACID

- Atomicity

  all-or-nothing, commit or abort
- Consistency

  moves data from one consistent state to another

  transaction不会破坏数据库的完整性规则

  比如订单扣减库存，如果库存不够则transaction abort并回滚，不会出现负数库存
- Isolation

  2 parts:

  - serializbility

    concurrent transactions has the same effect as some serial ordering of the transactions
  - failure isolation:

    a transaction cannot see the uncommitted effects of another transaction
- Durability

preserved once a transaction commits

life histories

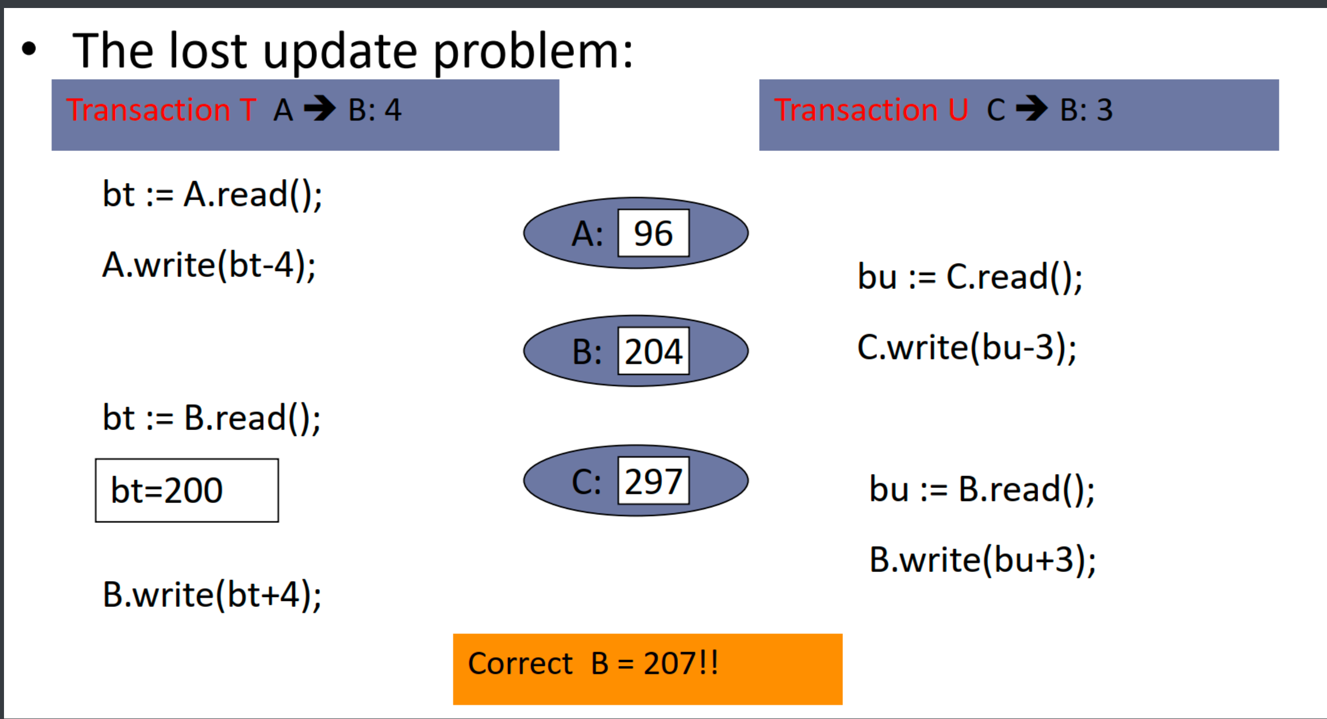
```
OpenTransaction() -> Trans
CloseTransaction(Trans) -> (Commit, Abort)
AbortTransaction(Trans)
```

- success
- abort by client `AbortTransaction(Trans)`
- abort by server (then error reported)

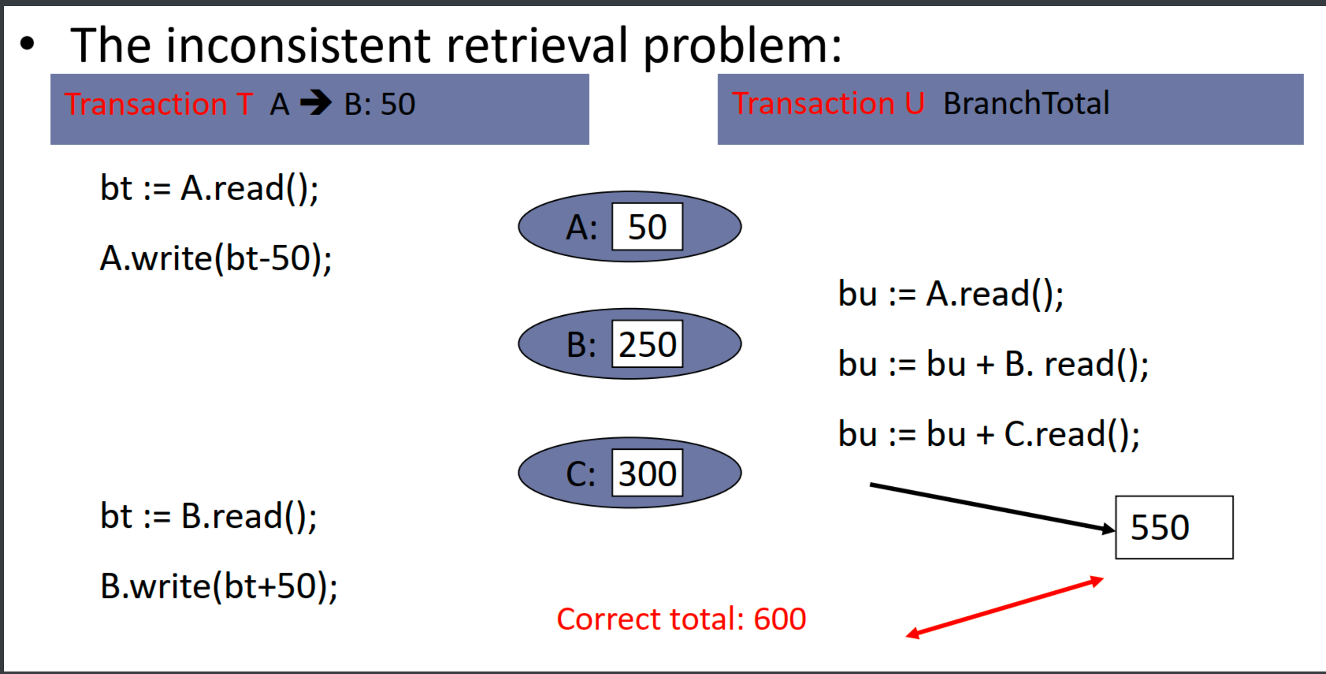
concurrency

- the lost update problem

  interleaved execution of operations on B?



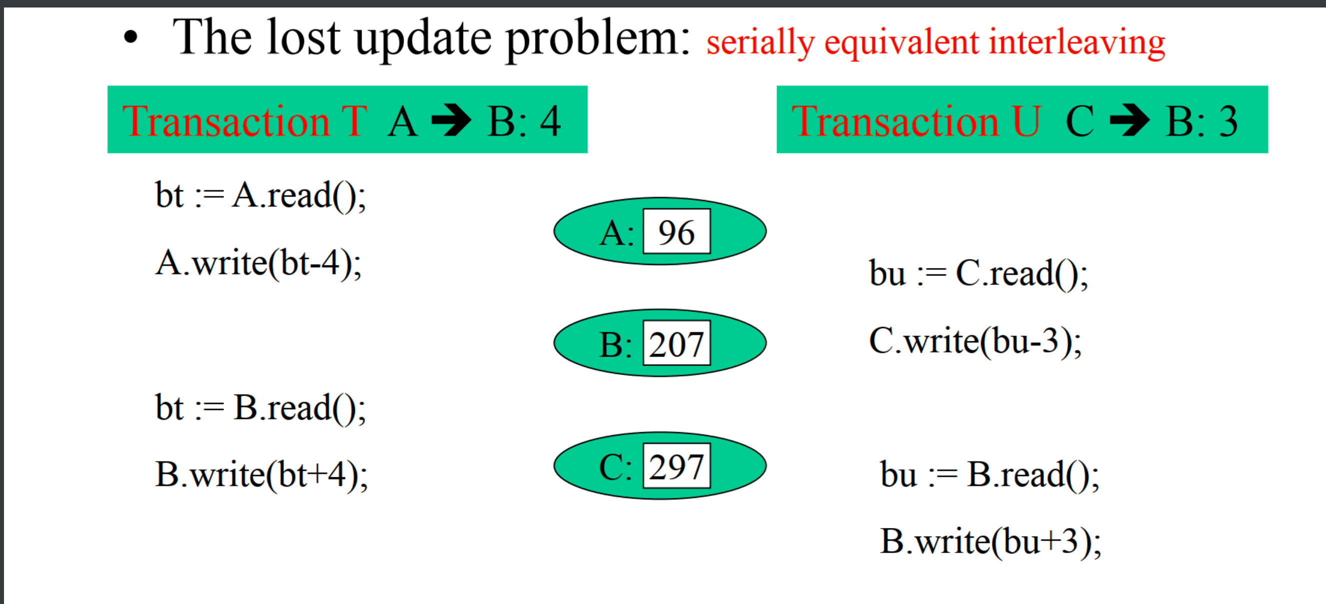
- inconsistent retrievals



- solutions?

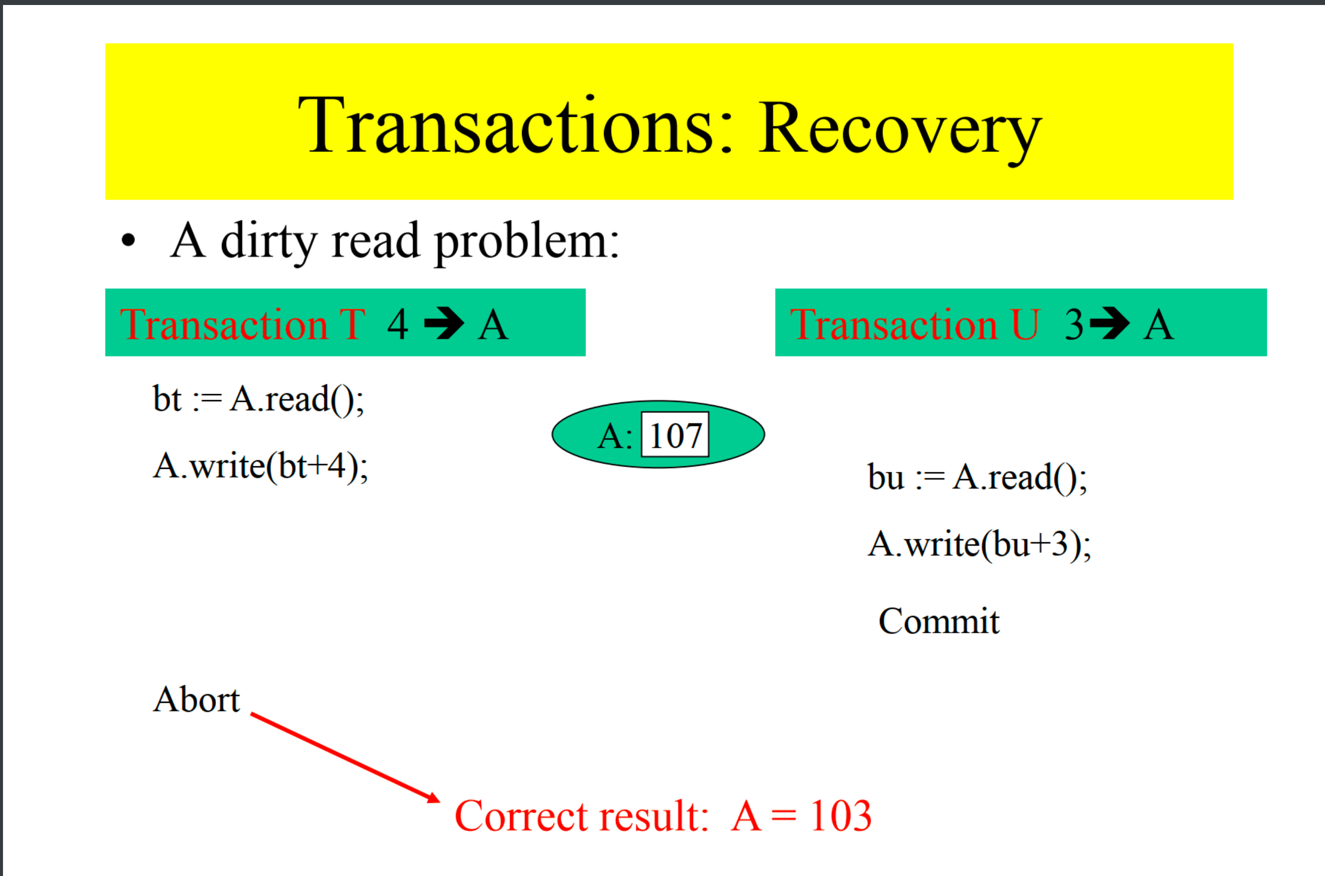
concurrency control: execute transactions in such a way that overall execution is equivalent with some serial execution

-> **serially equivalent interleaving**



recovery

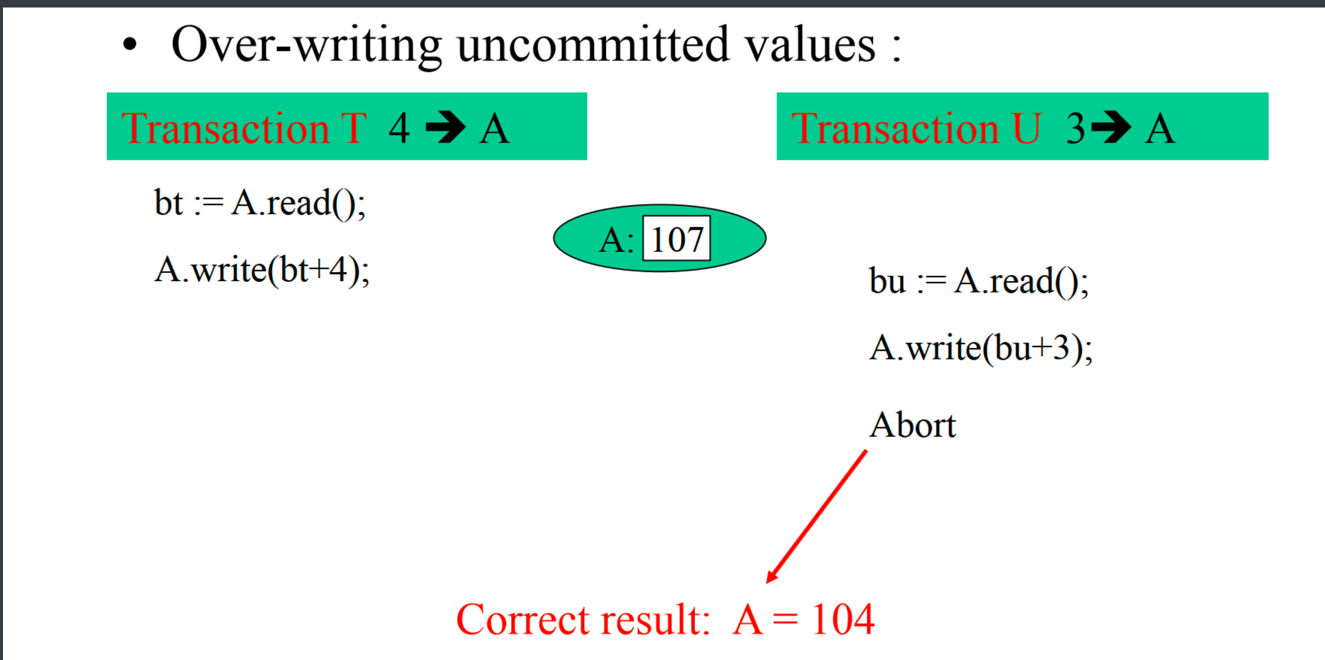
- a dirty read problem



T修改了数据但未提交，U读取了T修改过的数据；

如果T回滚，U读取的值会成为无效的脏数据

- premature write (over-writing uncommitted values)



事务 U 回滚后，其修改无效，但它已经覆盖了事务 T 的未提交值，导致事务 T 的更新丢失；

- solutions

- **cascading aborts**: a transaction reading uncommitted data must be aborted if the transaction that modified the data aborts
- to avoid cascading aborts (too many rollbacks)

  transactions can only read data written by committed transactions
- how to preserve data despite subsequent failures

  <u>stable storage</u>

  2 copies of data stored in separate parts of disks && not decay related

  <u>decay related:</u> probability of both parts corrupted is small

nested transactions

- transactions composed of **sub-transactions**
- subtransactions commit / abort independently
- effect of sub-transaction becomes durable only when top-level transaction commits

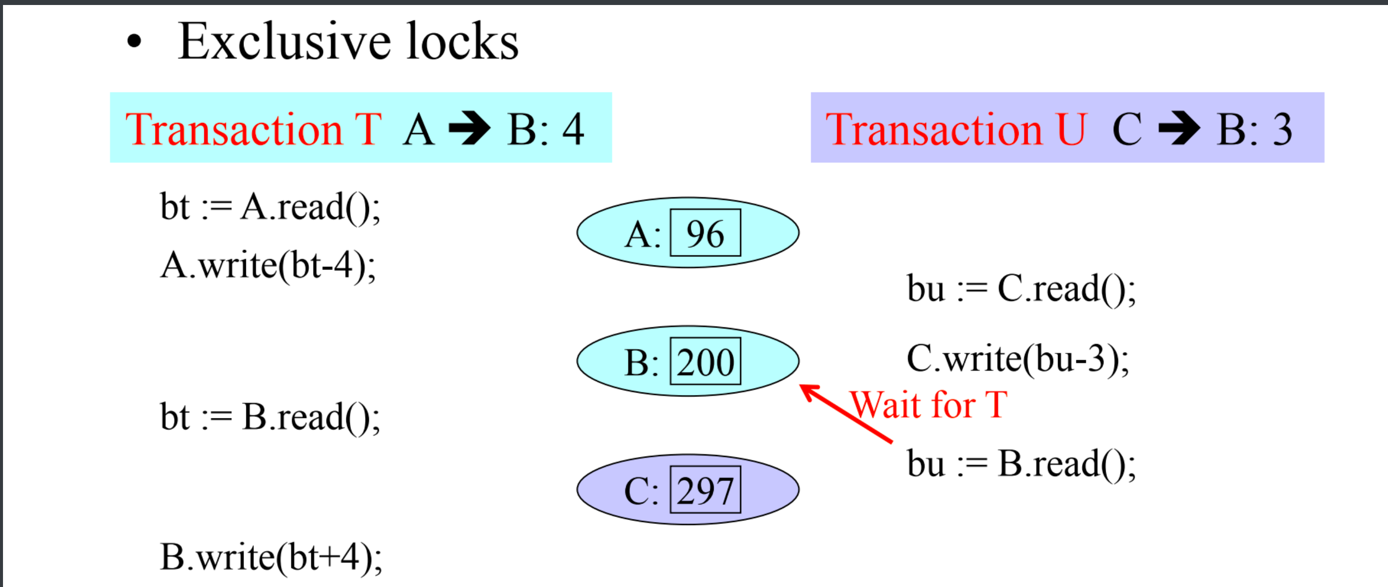
concurrency control: locking

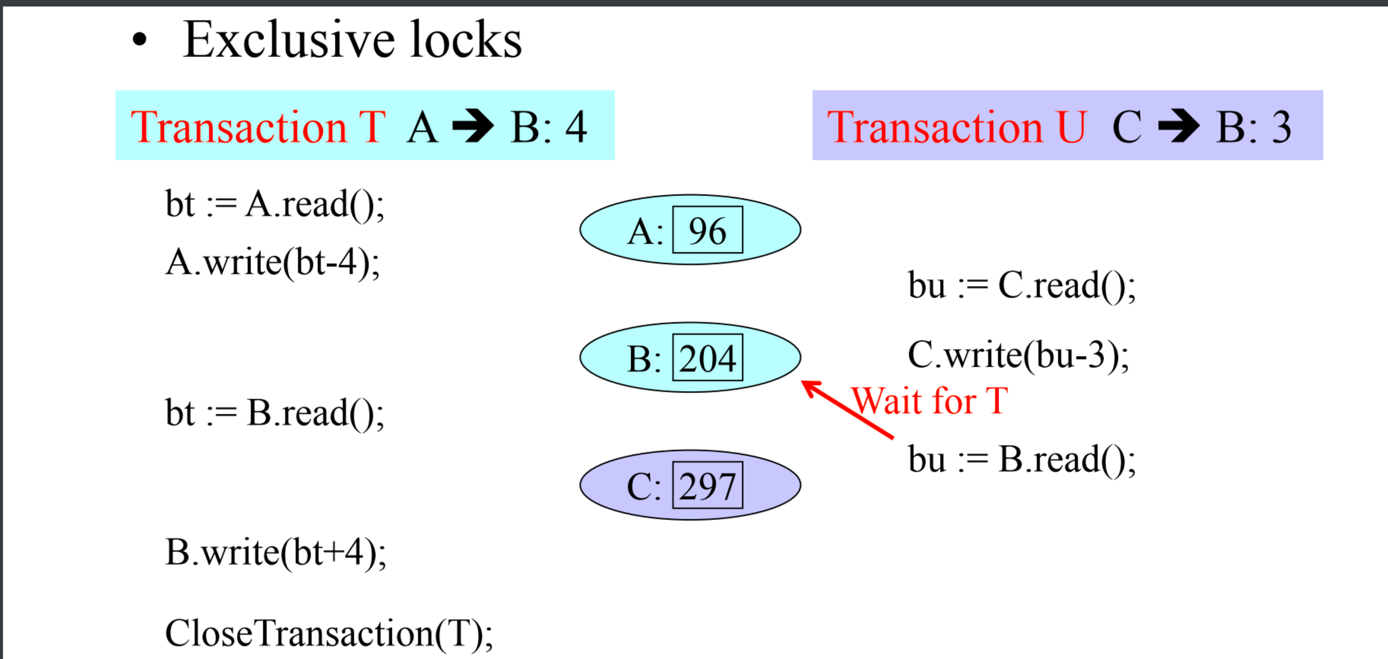
- <u>environment</u>:

  shared data in a <u>single</u> server, competing clients
- <u>goal:</u>

  <u>transactions && maximizing concurrency</u>
- <u>solution:</u>
- <u>serial equivalence</u>

<u>exclusive locks</u>





- basic elements of protocol
  1. serial equivalence -> *2-phase locking*
     - growing phase

       加锁阶段，拥有了锁之后可以继续加锁

       确保transaction锁住所有需要的资源
     - shrinking phase

       一旦transaction开始释放任何锁，就进入了释放锁阶段，不能再加锁
     - lock compatability for 2-phase locking

lock state of the target data	action
not locked yet	lock set & operation proceeds
conflicting lock set by another transaction	wait till release
non-conflicting lock set by another transaction	lock shared & operation proceeds
locked by itself	lock promoted if necesary (read -> write) & operation proceeds

2. hide intermediate results

   problem:

   如果一个transaction在commit / abort之前释放锁，其他transaction可能访问到其未完成的中间结果

   solution: strict 2-phase locking

   better release of locks only at commit / abort, which means that locks held till end of transaction

- how to increase concurrency & preserve serial equivalence
  - granularity of locks 锁的粒度（数据范围）

    large granularity -> limits concurrent access

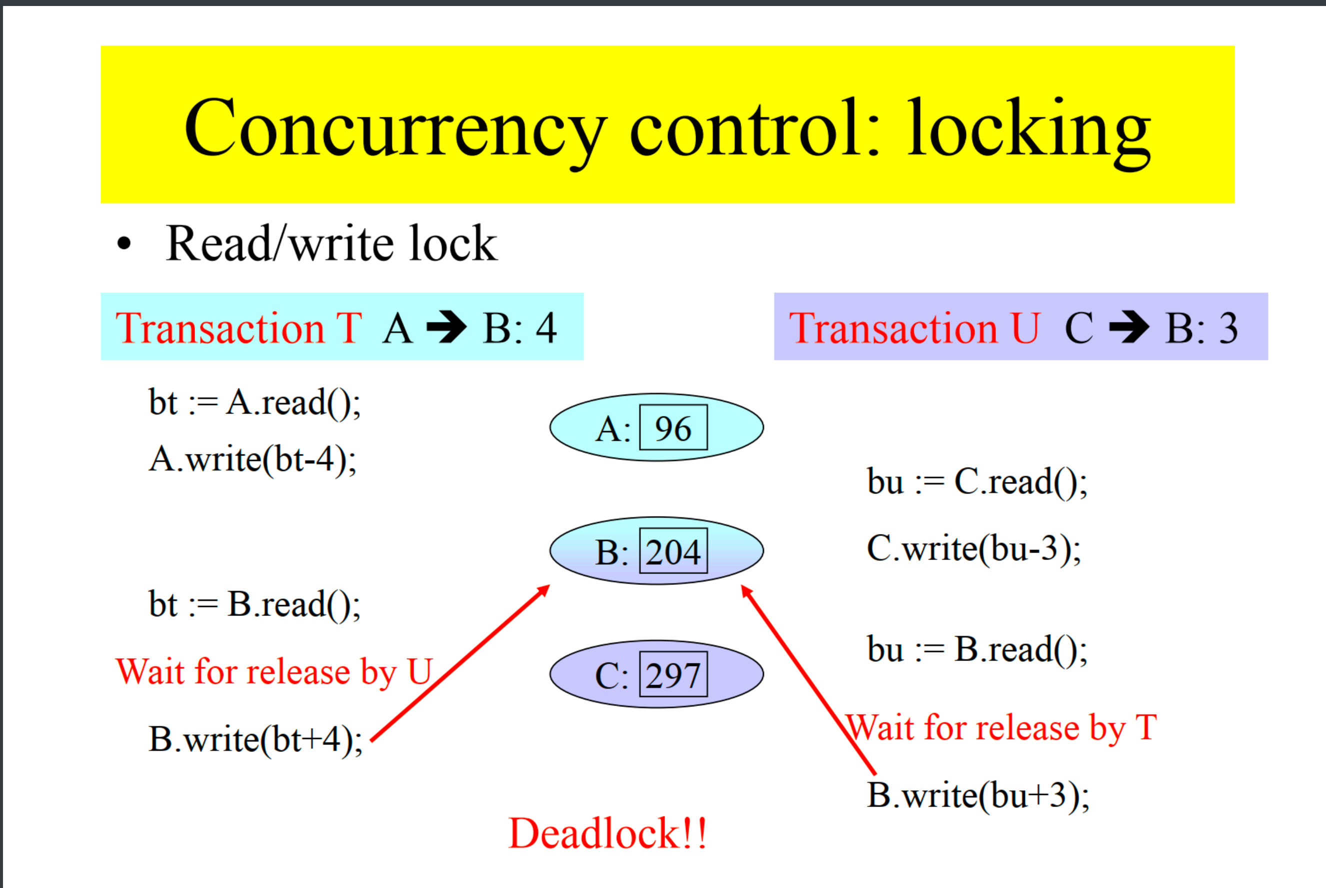
    small granularity -> higher managing overhead
  - appropriate locking rules

    lock compatibility

For one data item		Lock requested	
		Read	Write
Lock already set	None	OK	OK
	Read	OK	Wait
	Write	Wait	Wait

- lock implementation
  - lock manager
  - managing table of locks

    (which contains transaction identifiers, data item identifiers, lock type [shared / exclusive], condition variable)
- deadlocks



  - prevention
    - locking all data items used by a transaction when it starts
    - requesting locks on data items in a predefined order

    but impossible for interactive transactions, (在interactive环境下用户会动态决定transaction的行为，很难预先知道transaction将访问哪些数据) && reduction of concurrency
  - detection

    server keeps track of a wait-for graph,

    lock - edge added,

    unlock - edge removed

checks **cycles** when an edge is added

- solution

  1. once a deadlock detected, server selects a transaction and aborts it (**to break the cycle**)

     and the choice? factors:

     age of transaction, （年龄较小的transaction重启成本较低）

     number of cycles the transaction is involved in（优先终止涉及更多deadlock的transaction）

  2. timeouts

     locks granted for a limited period of time

     within period: lock invulnerable (**non-preemptive**)

     after period: lock vulnerable (**preemptive**)

     存在中断有效transaction的风险，适合实时性要求高，transaction复杂度低的场景