Database Administration

José Orlando Pereira

HASLab / Departamento de Informática
Universidade do Minho

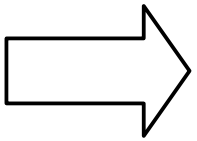
2020/2021

Transaction

- A unit of work composed by individual read and write database operations
- Can be committed or rolled back

ACID Transactions

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability



- The developer only cares about:
  - the logic of each transaction
  - running alone
  - in a perfect world

Data

- Tables:
  - Client: Id, Name, Address
  - Product: Id, Description, **Stock, Max, Min**
  - Invoice: Id, ~~**ProductId**~~, ClientId
  - **InvoiceLine: Id, InvoiceId, ProductId**
  - **Orders: Id, ProductId, Supplier, Items**

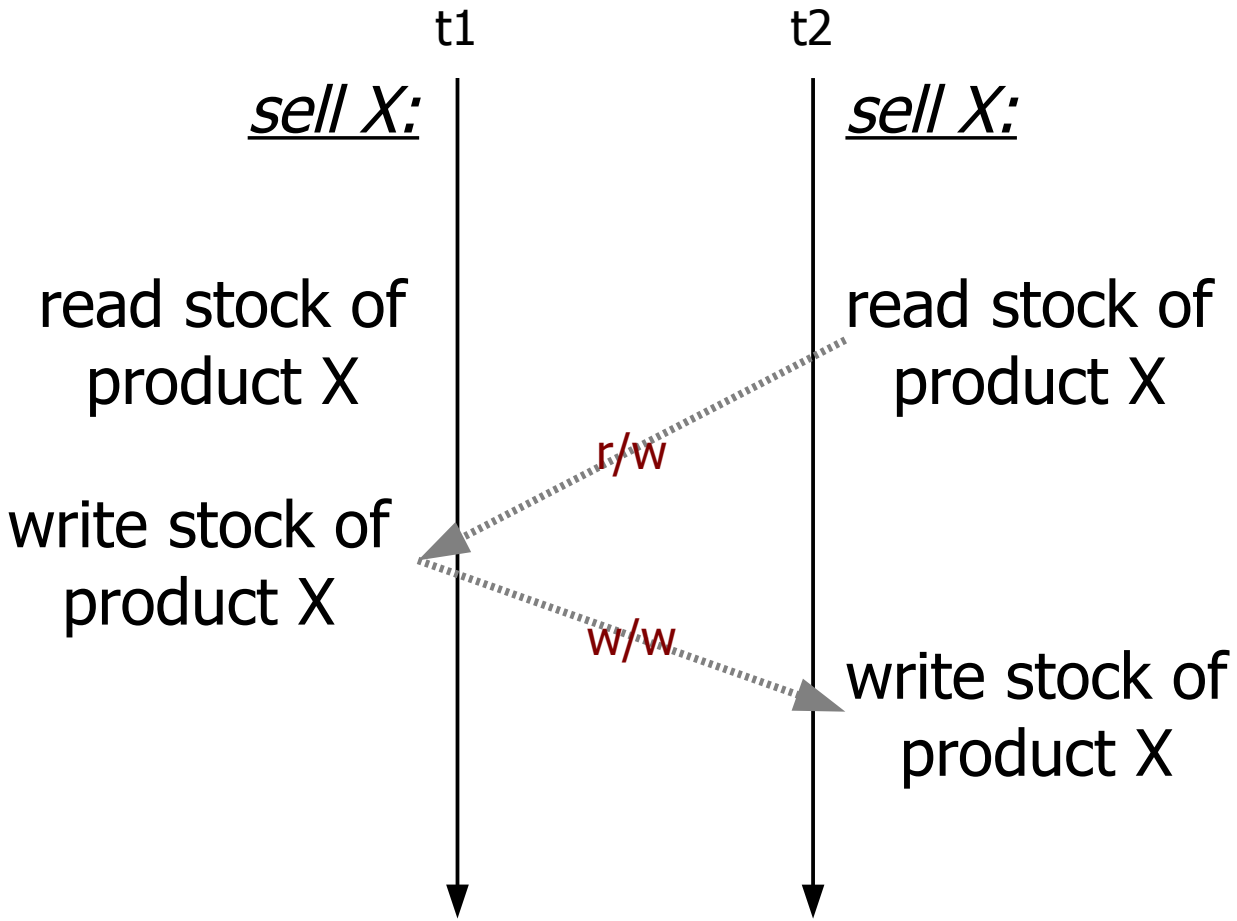
Operations

- Sell:
  - Add invoice record
  - Add multiple invoice lines
  - Decrease stocks
- Order:
  - If Stock < Min
  - Order items up to Max
- Delivery:
  - Add each ordered product to stock
  - Reset orders

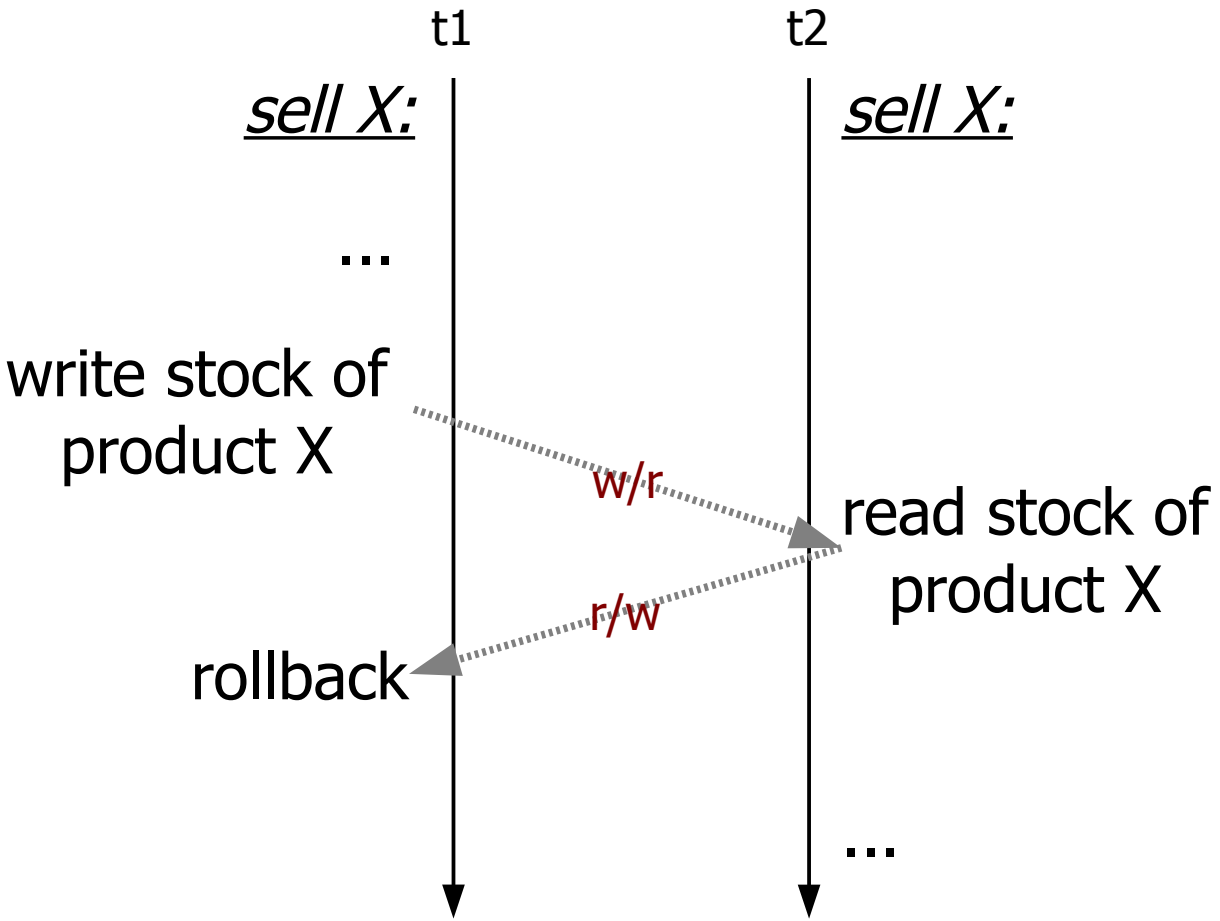
What can go wrong?

- First, implement each operation assuming that no two operations run concurrently
- Then, assume that any operations can run concurrently
- What can go wrong?

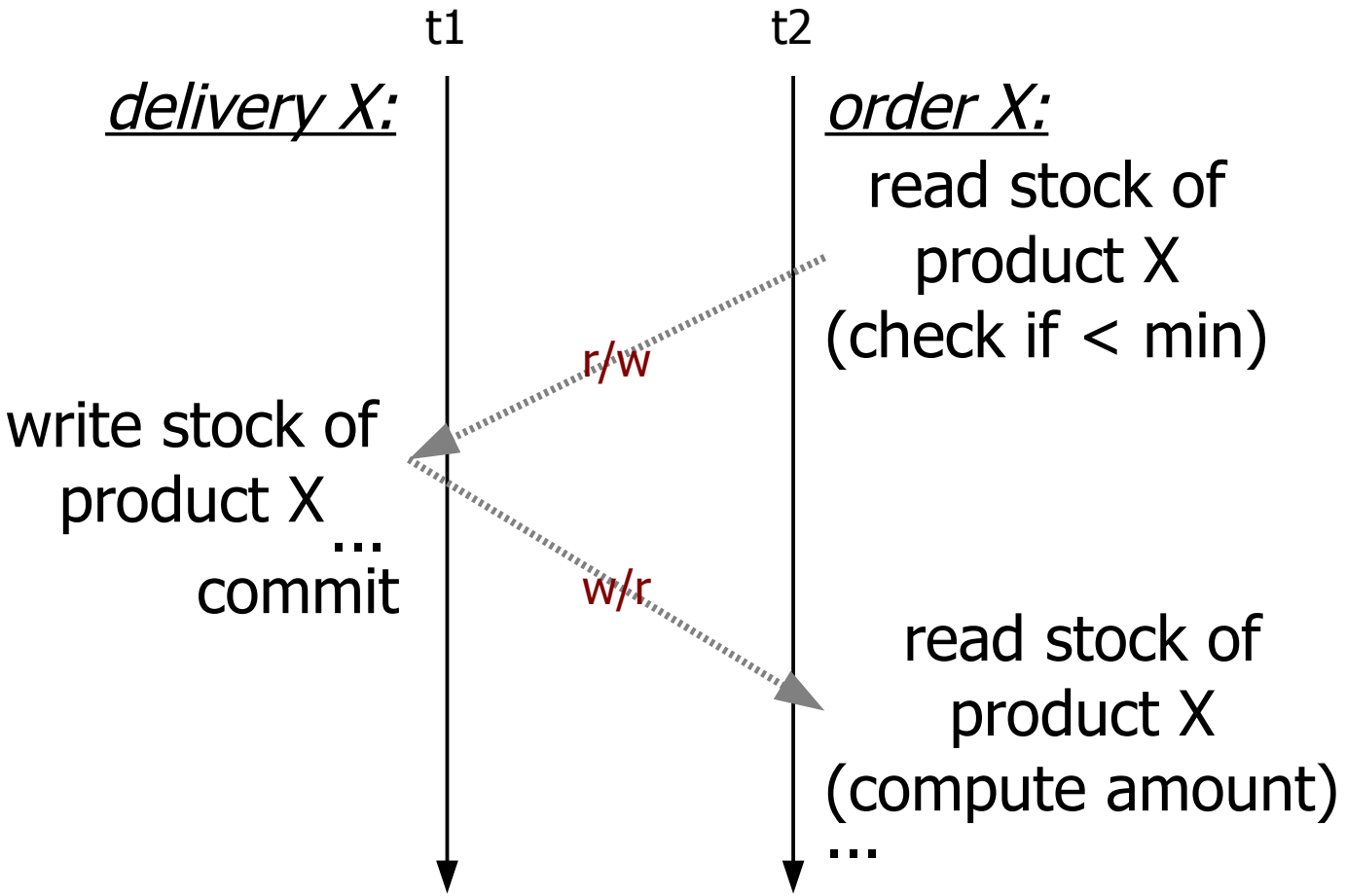
Lost update (RWW)



Dirty read (WRW)



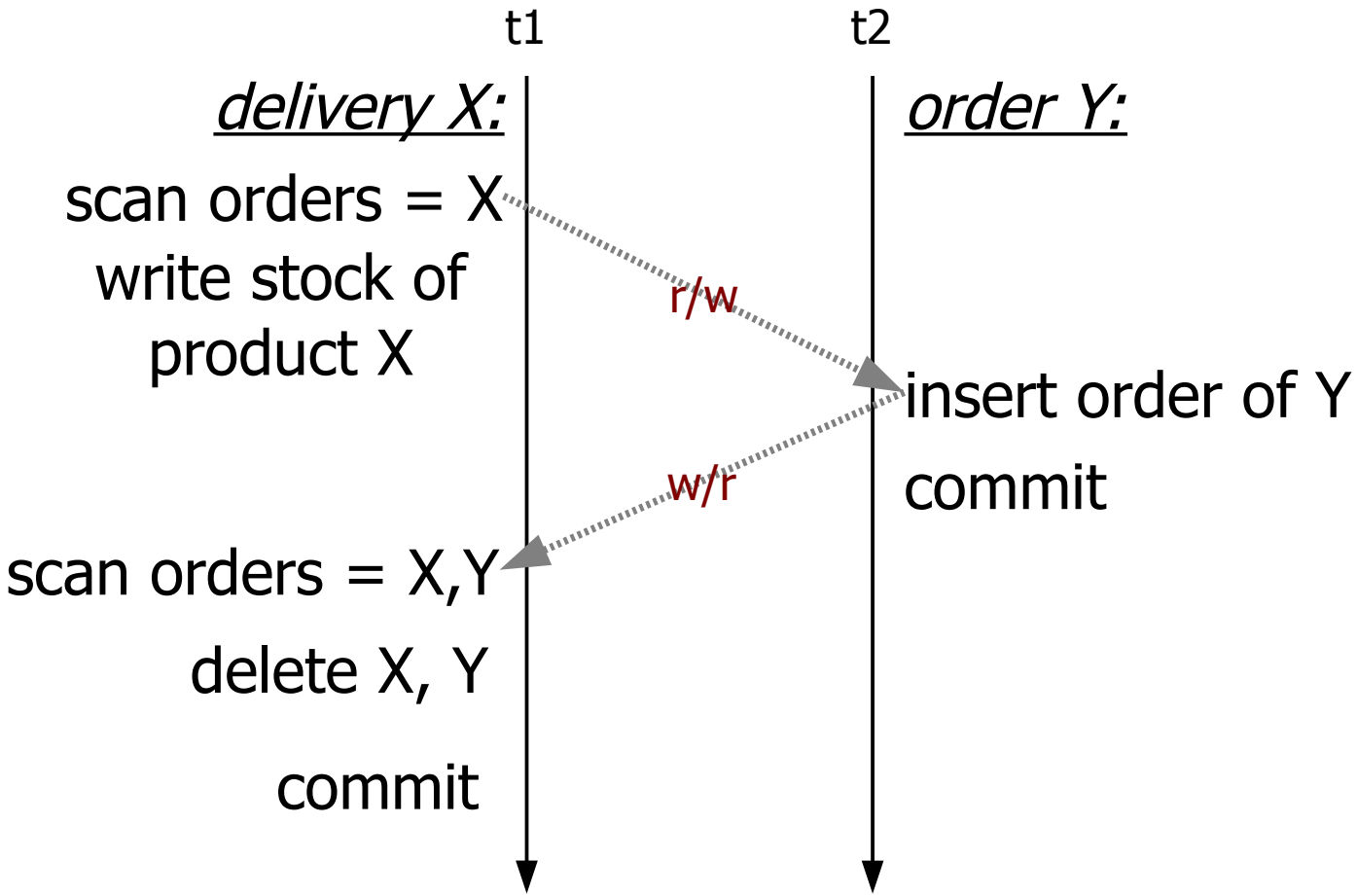
Non-repeatable read (RWR)



Other anomalies?

- Read after read is not a problem
  - Thus no RRW, WRR...
- Why no WWR?
  - Assume no "blind writes"
  - Actually a **R**WW**R** (lost update)

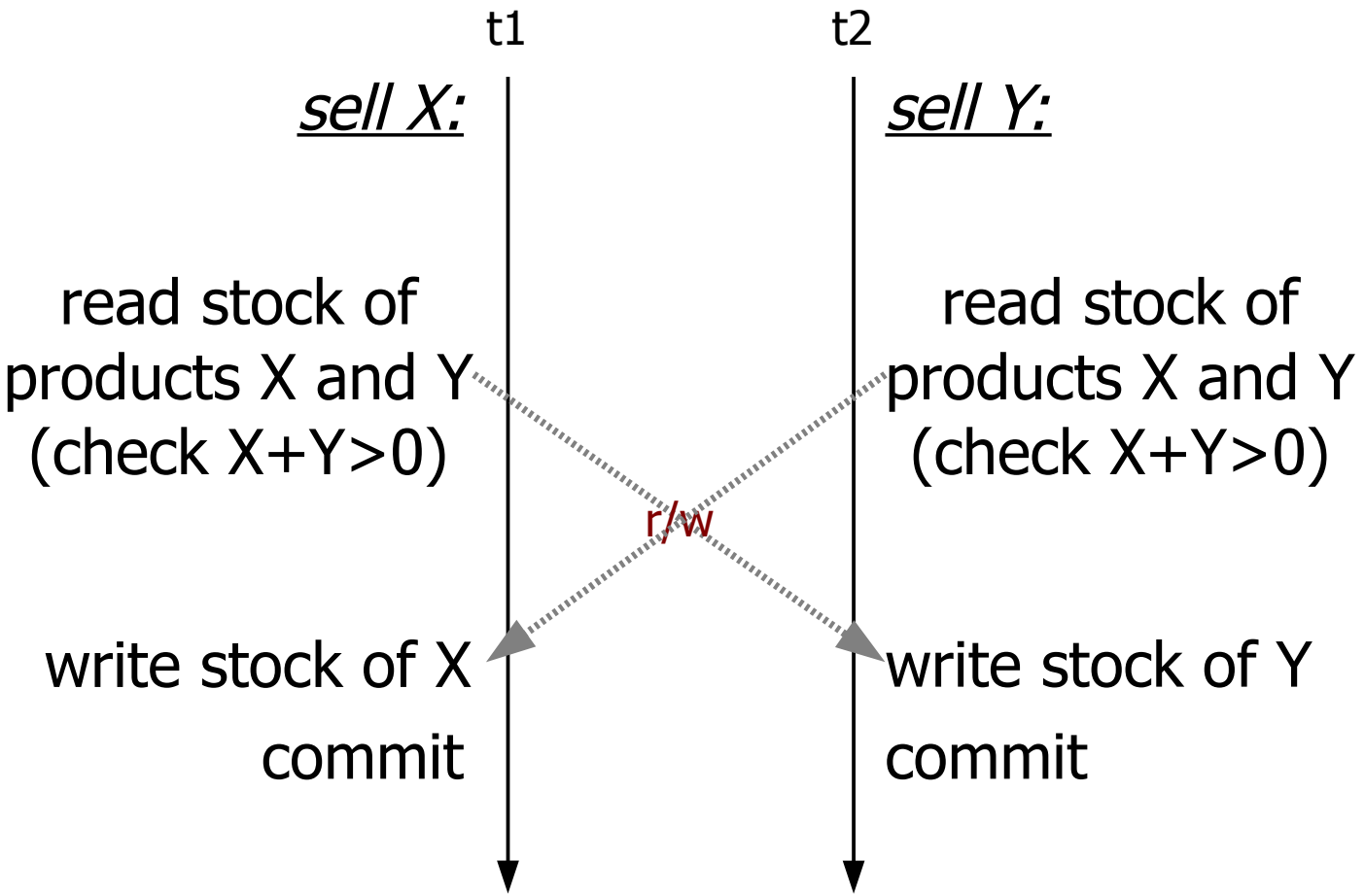
Phantoms



Phantoms

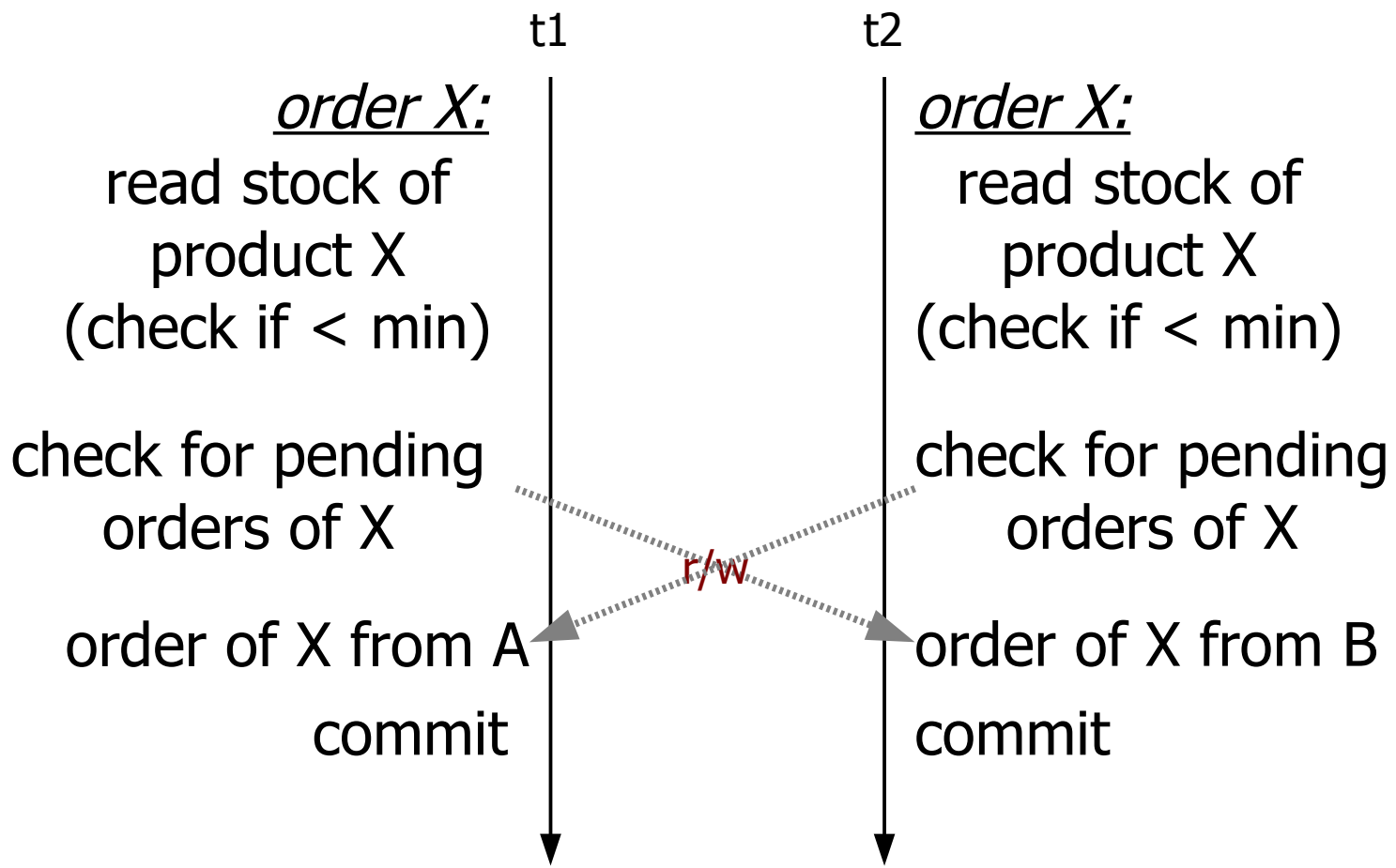
- It is actually a non-repeatable read (RWR) on the collection
- Why no dirty read (WRW) for collections?
  - Solved by having no dirty reads on the item
- Why no RWW (lost update) for collections?
  - Means allocating the same physical space for two records!
  - Very dangerous: corruption, etc...

Write skew



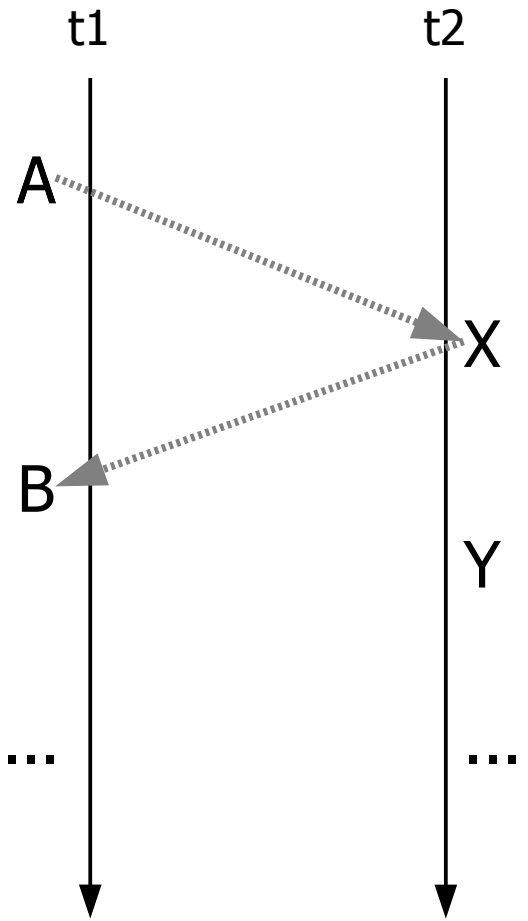
(assume X is backup for Y and vice-versa)

Write skew on collections



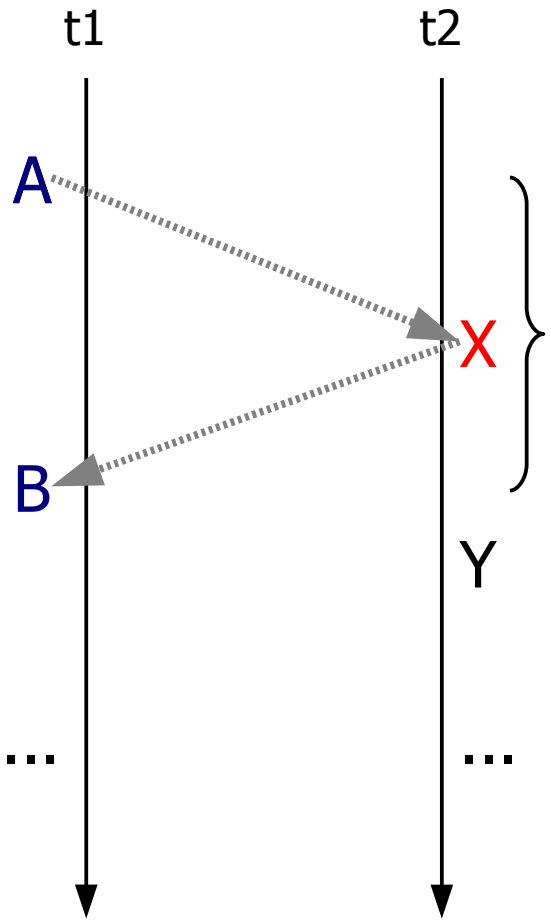
General problem

- No serial execution is conceivable:
  - Some t1 must be ordered after t2
  - But t2 must be ordered after t1
- The user cannot be fooled into thinking that transactions execute serially (i.e. serialized)



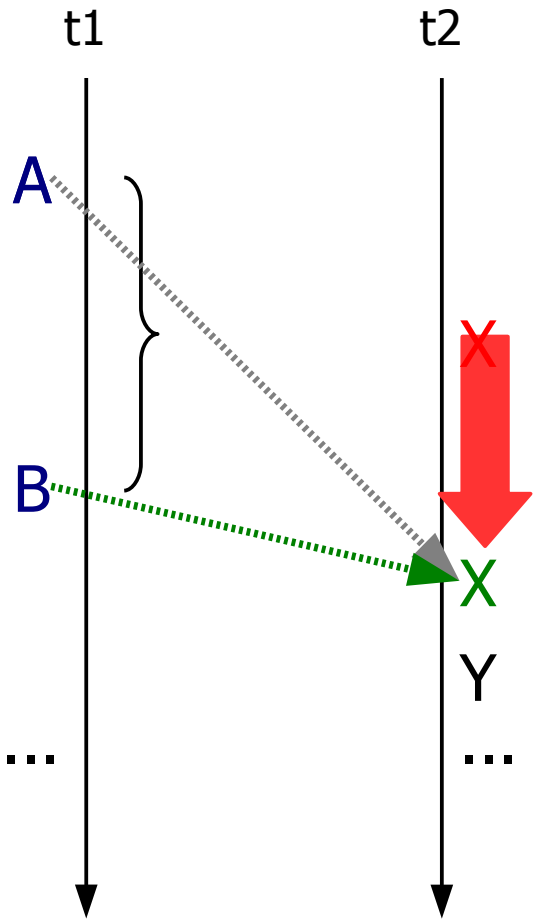
General problem

- In detail, some operation X should not be happening between A and B...



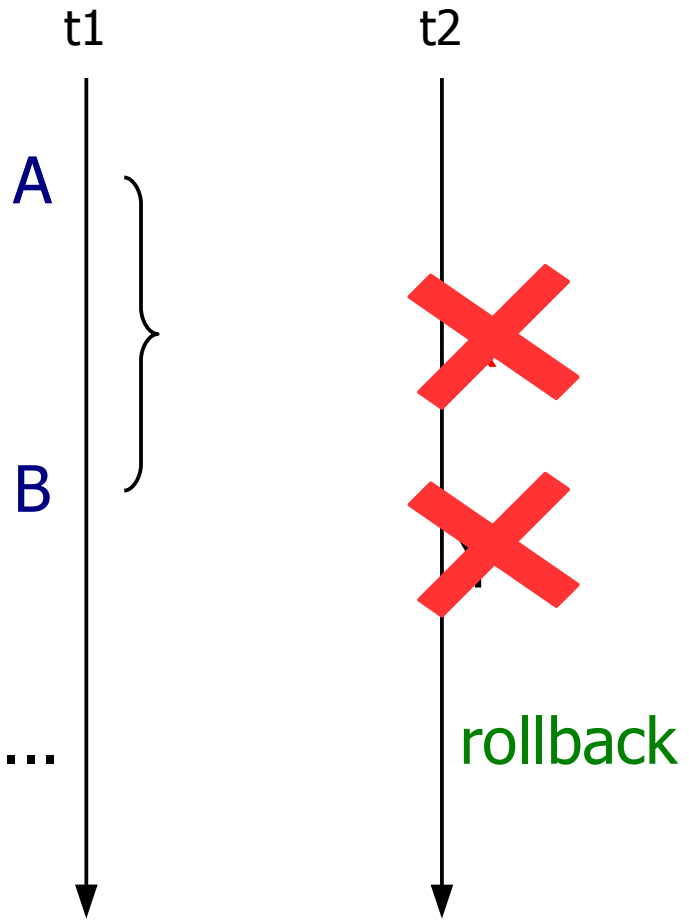
General approach

- Delay X (and all its consequences) until B
  - t1 precedes t2



General approach

- Remove X and related operations
  - t1 executes alone



General approach

- Anticipate X (i.e. execute X before the application has requested it!)
  - t2 precedes t1
- (Can you propose a mechanism to do this!?)