CS150 Discussion 8

Transactions & Concurrency

Jiachun Jin
jinjch@shanghaitech.edu.cn

Motivation

- Many users work with the application at the same time

⇕

- Multiple clients running SQL at the same time, over one database
  - concurrency provides efficiency
    - improper algorithms introduces weird bugs
  - fault tolerant
  - reliable

Outline

- Transaction
- Concurrency

Transaction

- A transaction is a sequence of one or more operations (reads or writes) which reflects a single real-world transition
  - transfer money between accounts
- In a program, multiple statements (rd & wt) can be grouped together as a transaction:

```
START TRANSACTION
        UPDATE Bank SET amount = amount – 100
        WHERE name = ‘Bob’
        UPDATE Bank SET amount = amount + 100
        WHERE name = ‘Joe’
COMMIT
```

Property

- A tomicity: All actions in the transaction happen, or none happen
- C onsistency: If the DB starts out consistent, it ends up consistent at the end of the transaction
  - tables must always satisfy user-specified integrity constraints
- I solation: Execution of each transaction is isolated from that of others
  - concurrency relies on this property
- D urability: If a transaction commits, its effects persist

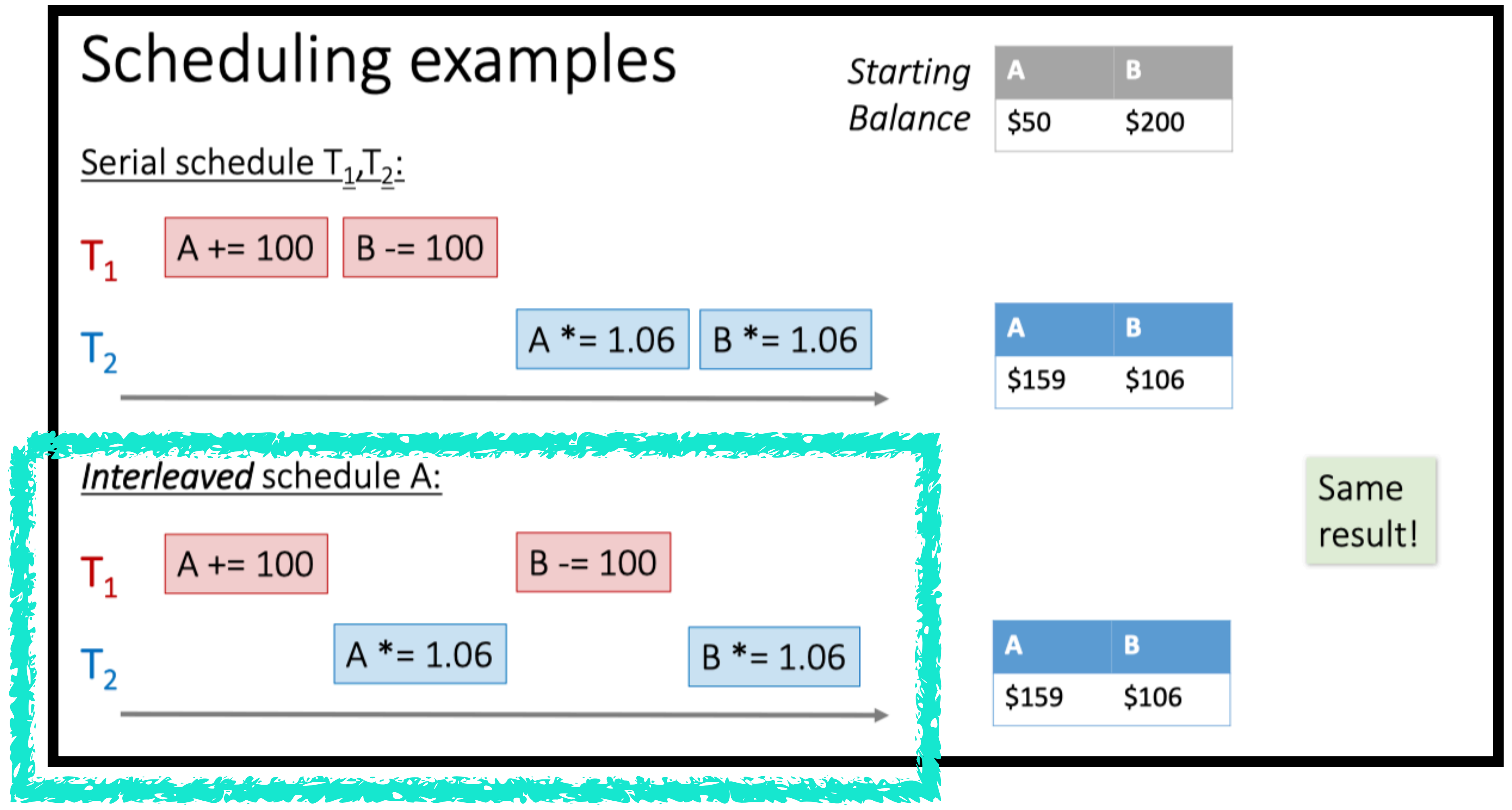
Outline

- Transaction
- Concurrency

Interleave transactions

- Individual transaction might be slow– don’t want to block other users during!
- We must pick an interleaving or schedule such that isolation and consistency are maintained

An example



Scheduling examples

Starting Balance

A	B
$50	$200

Serial schedule T_1,T_2:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

A	B
$159	$106

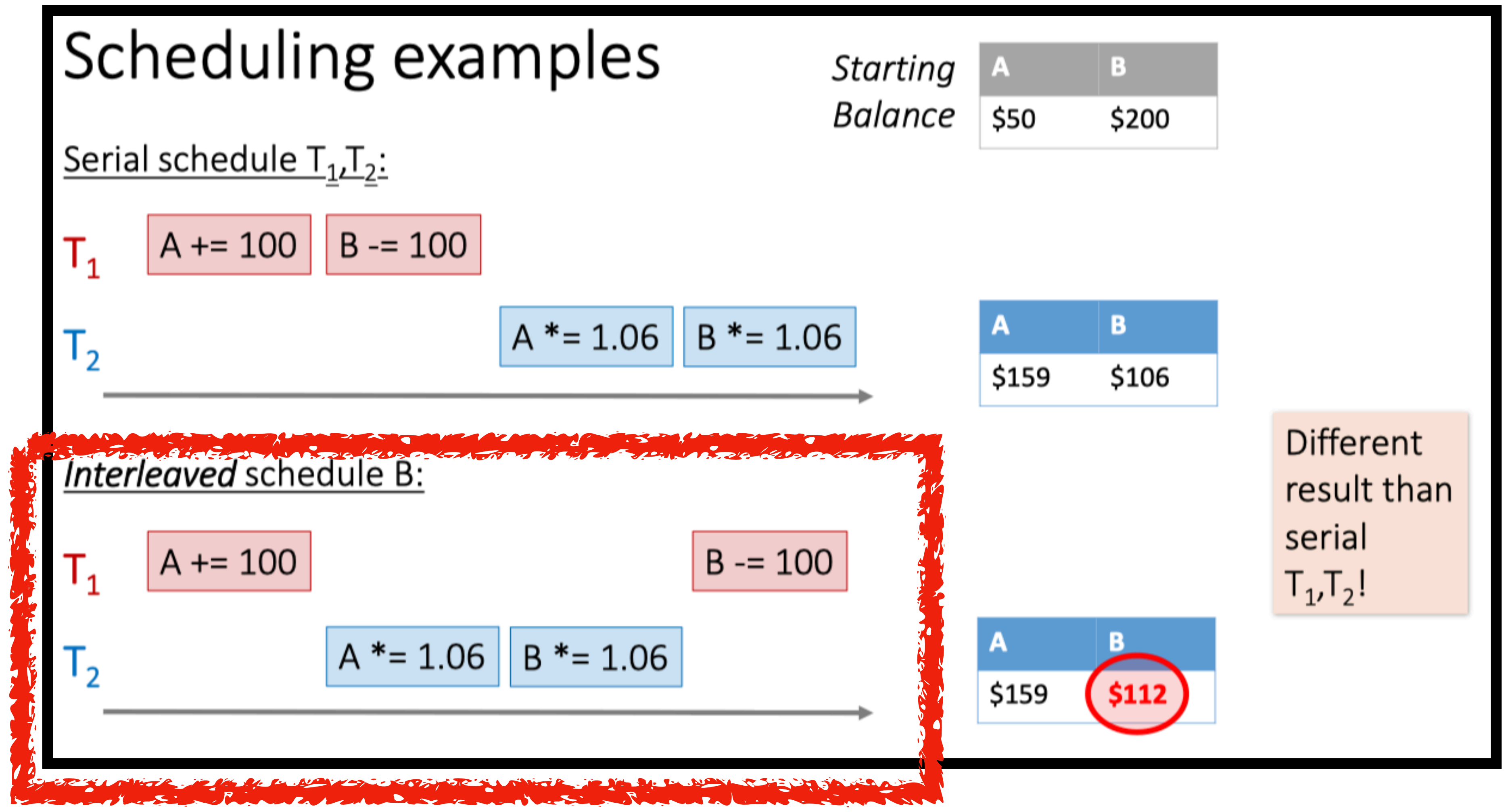
Interleaved schedule A:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

A	B
$159	$106

Same result!



Scheduling examples

Starting Balance

A	B
$50	$200

Serial schedule T_1,T_2:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

A	B
$159	$106

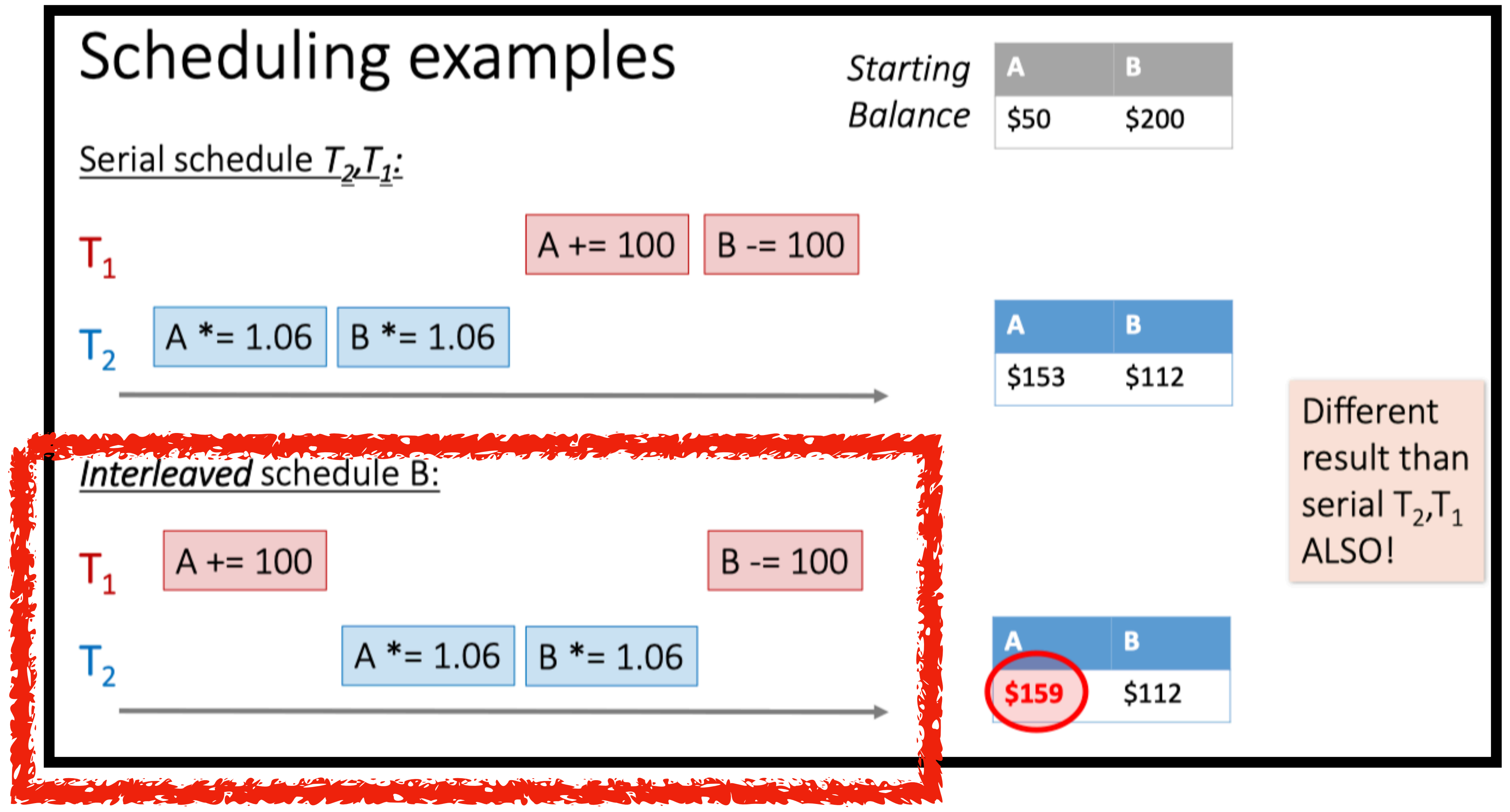
Interleaved schedule B:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

A	B
$159	$112

Different result than serial T_1,T_2!



Scheduling examples

Starting Balance

A	B
$50	$200

Serial schedule *T_2,T_1*:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

A	B
$153	$112

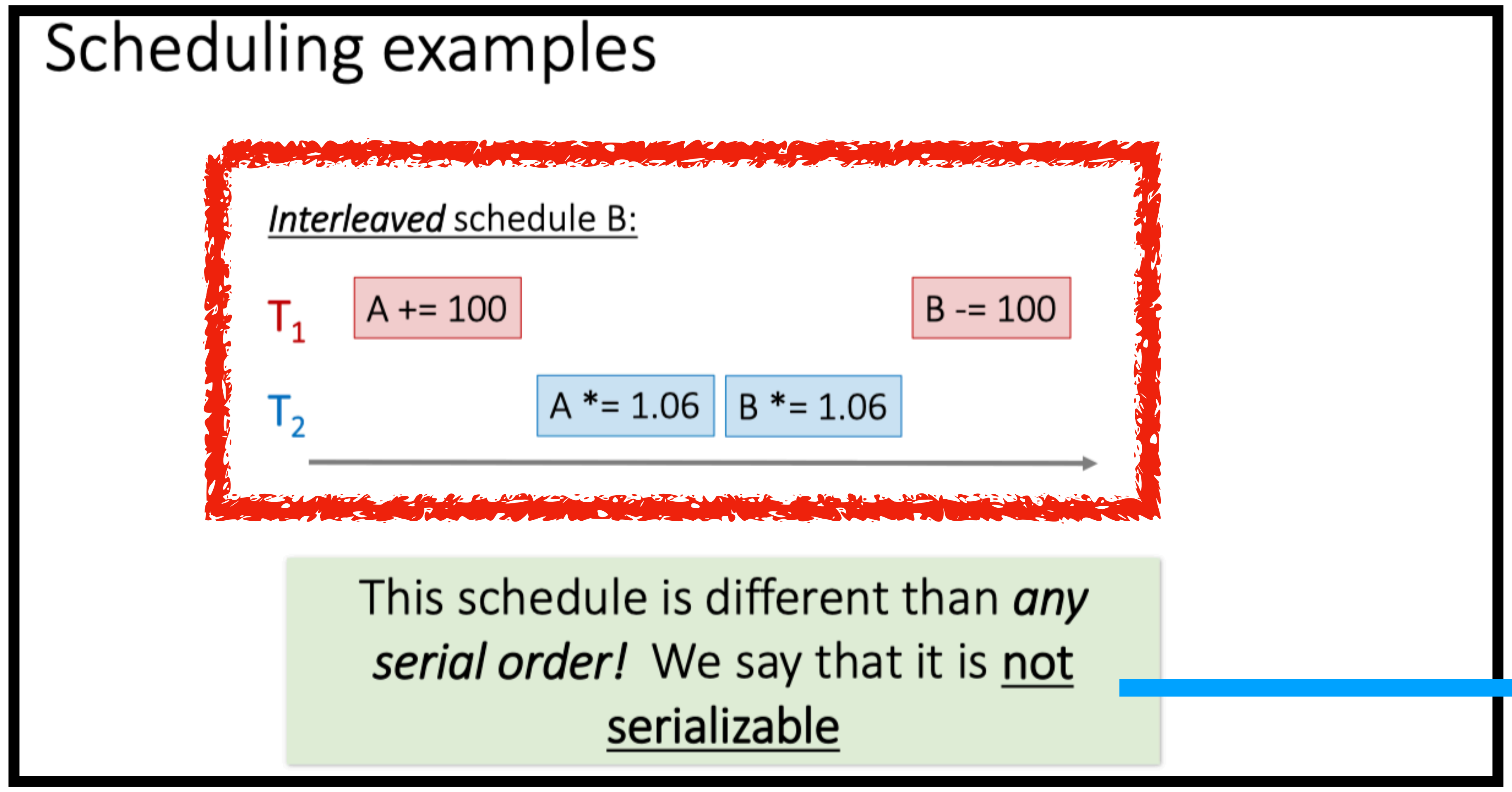
Interleaved schedule B:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

A	B
$159	$112

Different result than serial T_2,T_1 ALSO!



Scheduling examples

Interleaved schedule B:

T_1 A += 100 B -= 100

T_2 A *= 1.06 B *= 1.06

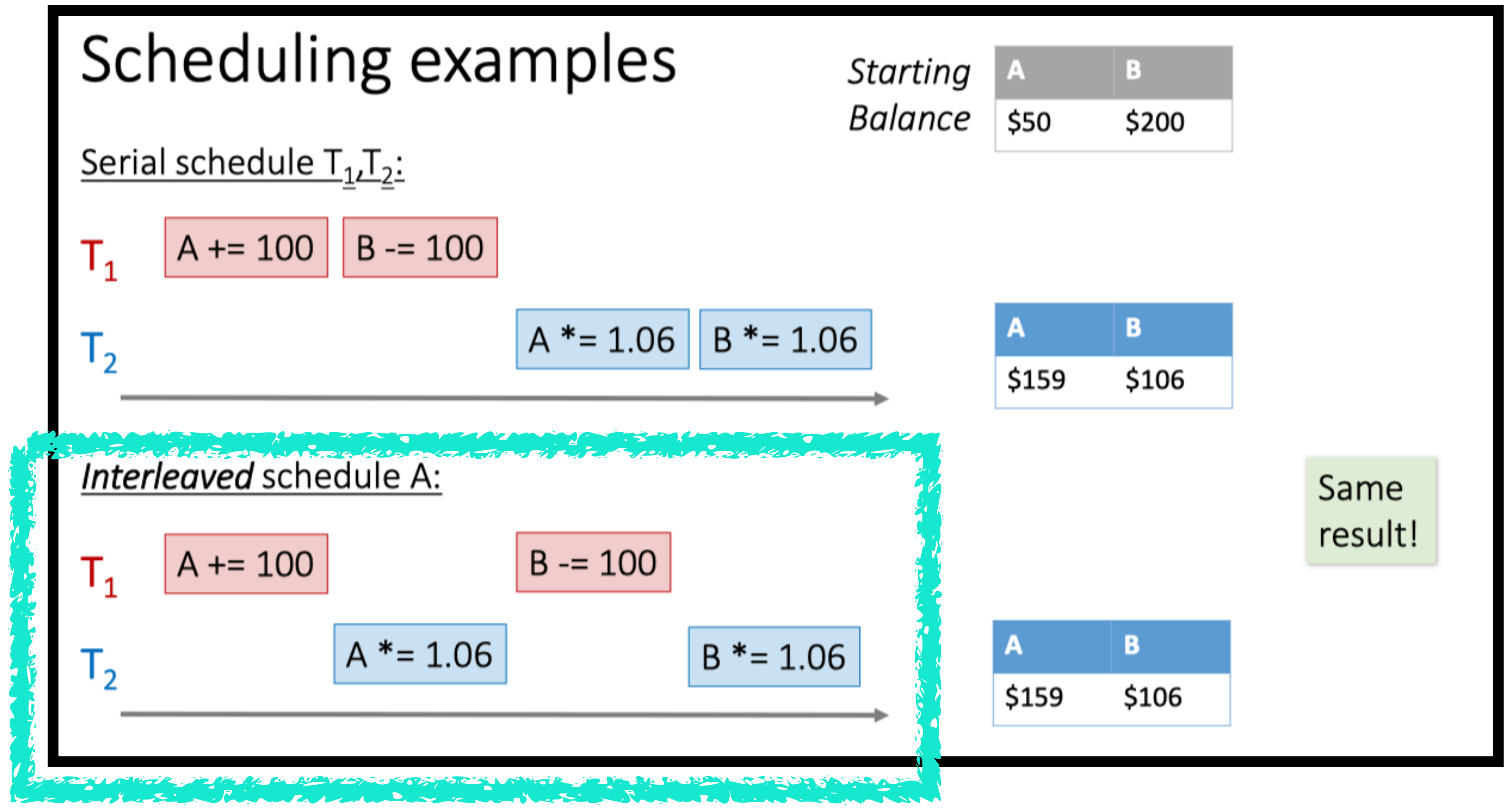
This schedule is different than *any serial order!* We say that it is not serializable

what does serializable mean?

Scheduling Definitions

- A serial schedule is one that does not interleave the actions of different transactions
- A and B are equivalent schedules if, for any database state, the effect on DB of executing A is identical to the effect of executing B
- A serializable schedule is a schedule $\mathbb{S}$ that is equivalent to some serial execution of the transactions $\mathbb{S}_{serial}$
  - revisit the above example

An example



Scheduling examples

Starting Balance

A	B
$50	$200

Serial schedule T_1,T_2:

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

A	B
$159	$106

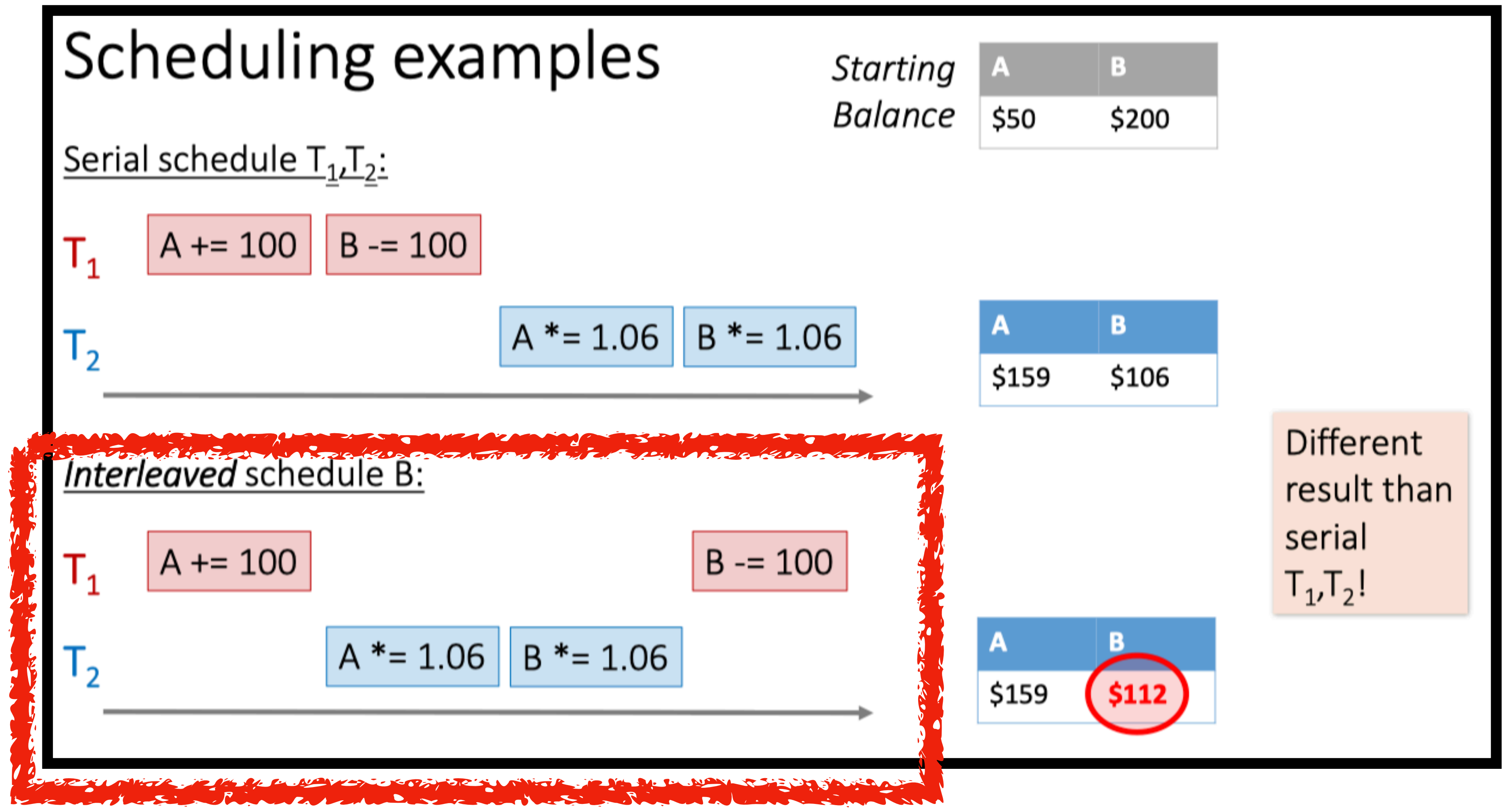
Interleaved schedule A:

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

A	B
$159	$106

Same result!



Scheduling examples

Starting Balance

A	B
$50	$200

Serial schedule T_1,T_2:

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

A	B
$159	$106

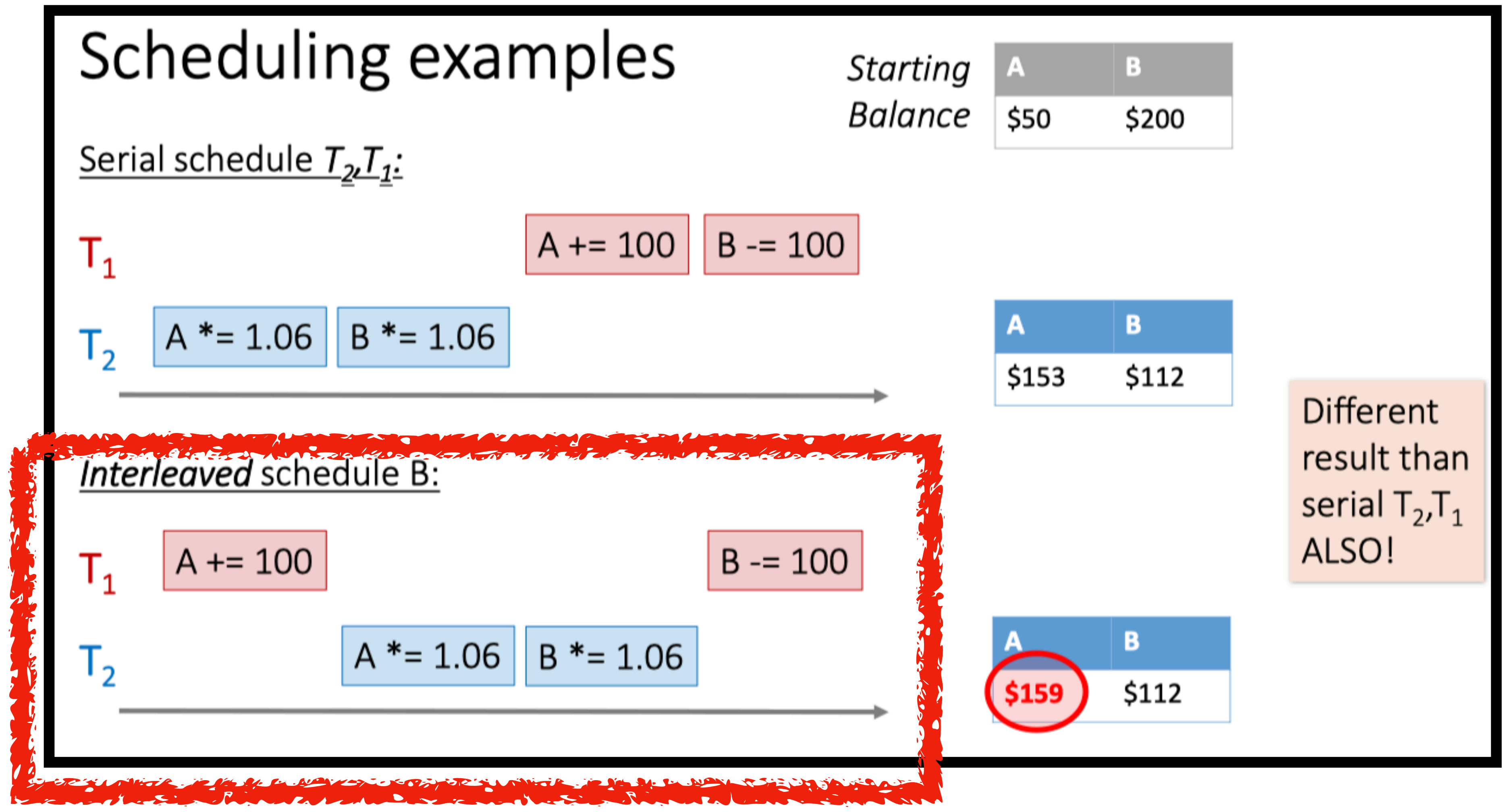
Interleaved schedule B:

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

A	B
$159	$112

Different result than serial T_1,T_2!



Scheduling examples

Starting Balance

A	B
$50	$200

Serial schedule *T_2,T_1:*

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

A	B
$153	$112

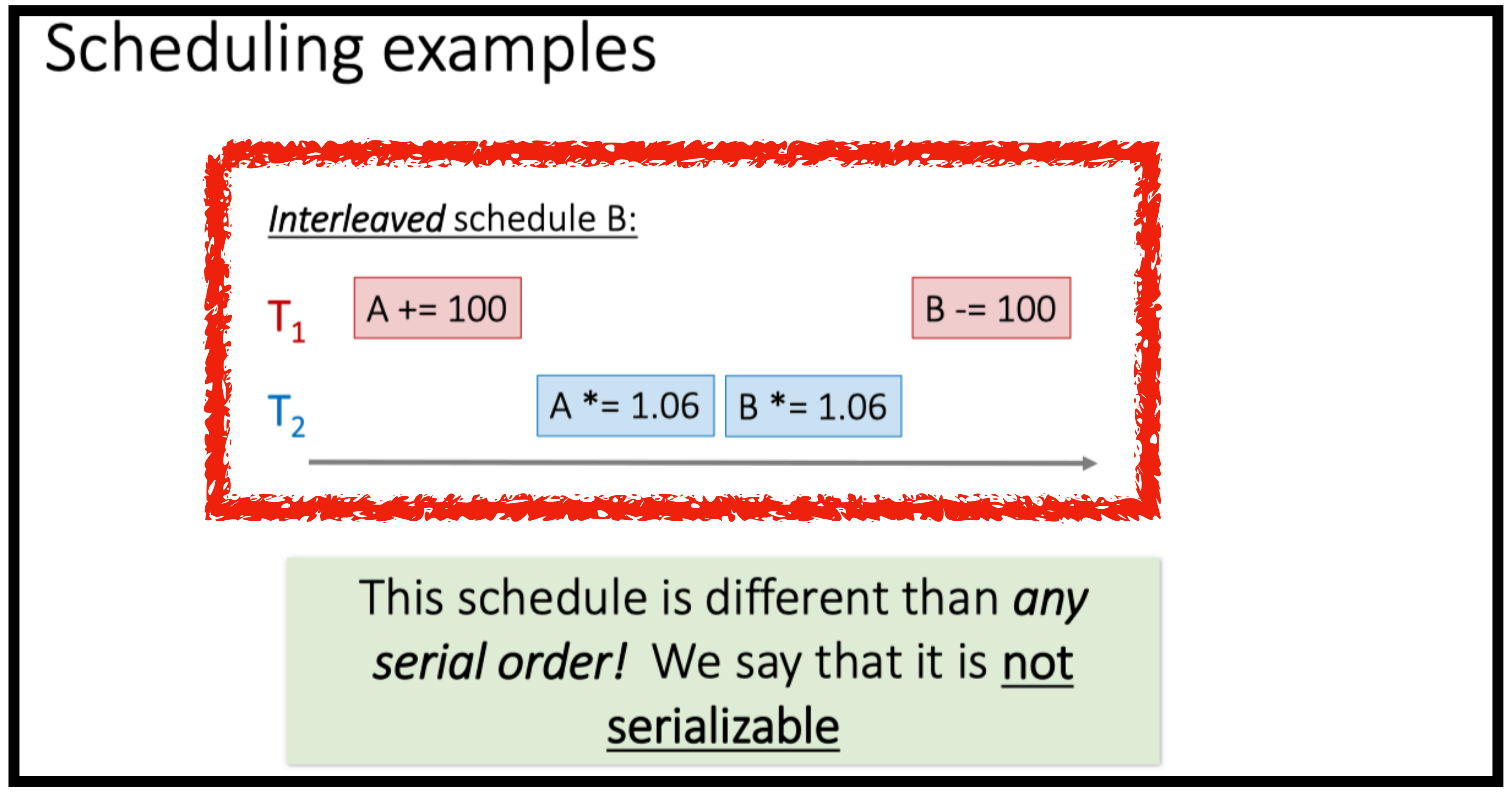
Interleaved schedule B:

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

A	B
$159	$112

Different result than serial T_2,T_1 ALSO!



Scheduling examples

Interleaved schedule B:

T_1: A += 100 | B -= 100

T_2: A *= 1.06 | B *= 1.06

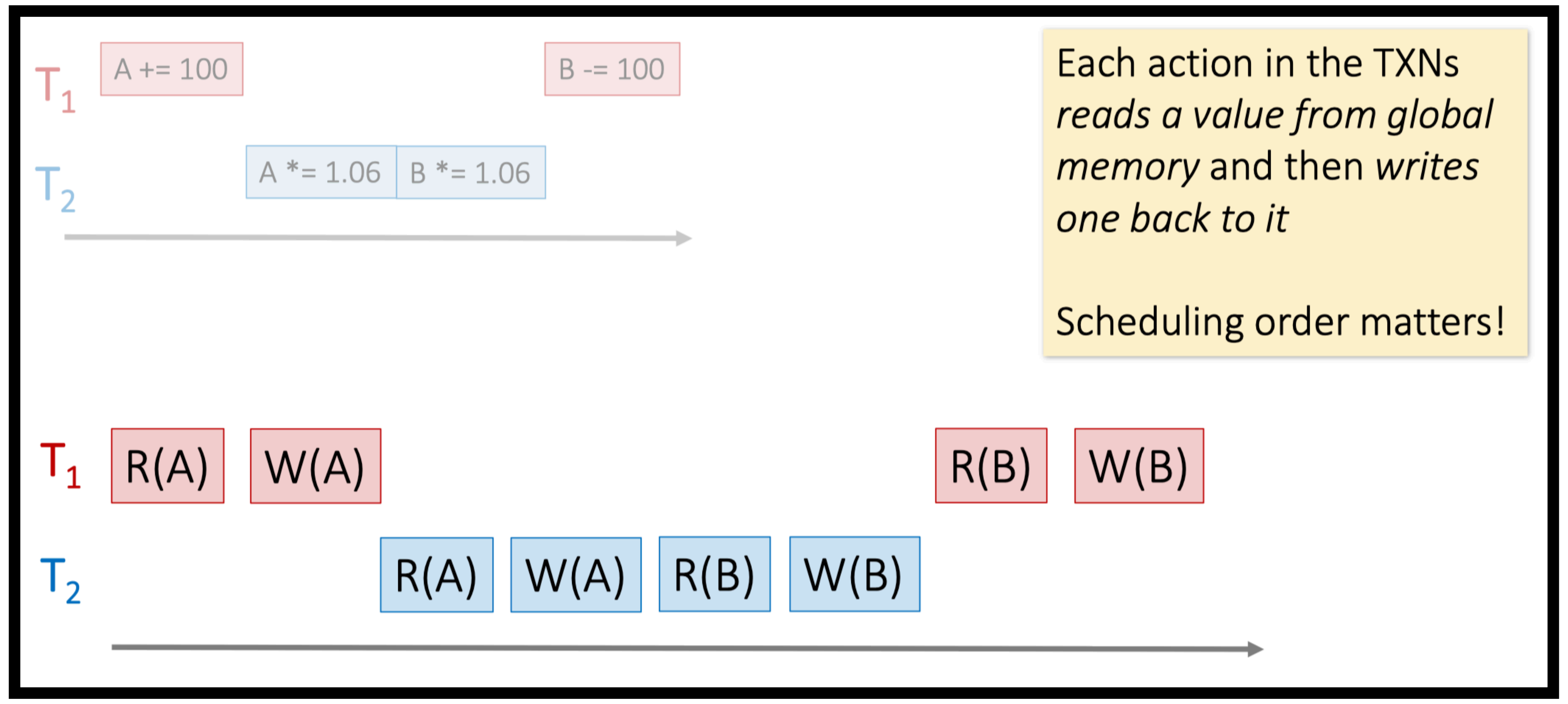
This schedule is different than *any serial order!* We say that it is not serializable

□ : serializable

□ : not serializable

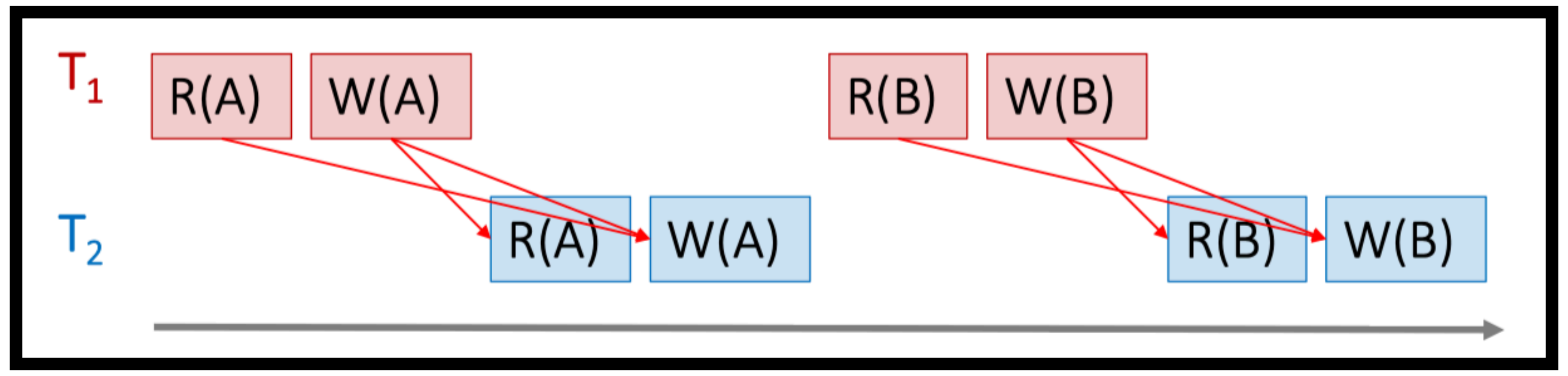
Conflicts

- Goal: discerning "good" vs. "bad" schedules
  - serializable schedule will maintain isolation & consistency (to some extend "good")
  - a stricter, but very useful variant: conflict serializability
- The DBMS's view of the schedule



Each action in the TXNs *reads a value from global memory* and then *writes one back to it*

Scheduling order matters!



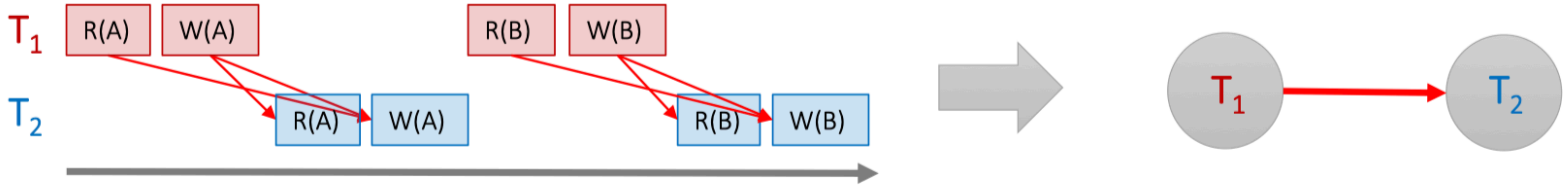
- Two actions conflict if they are part of different transactions, involve the same variable, and at least one of them is a write

Conflict serializable

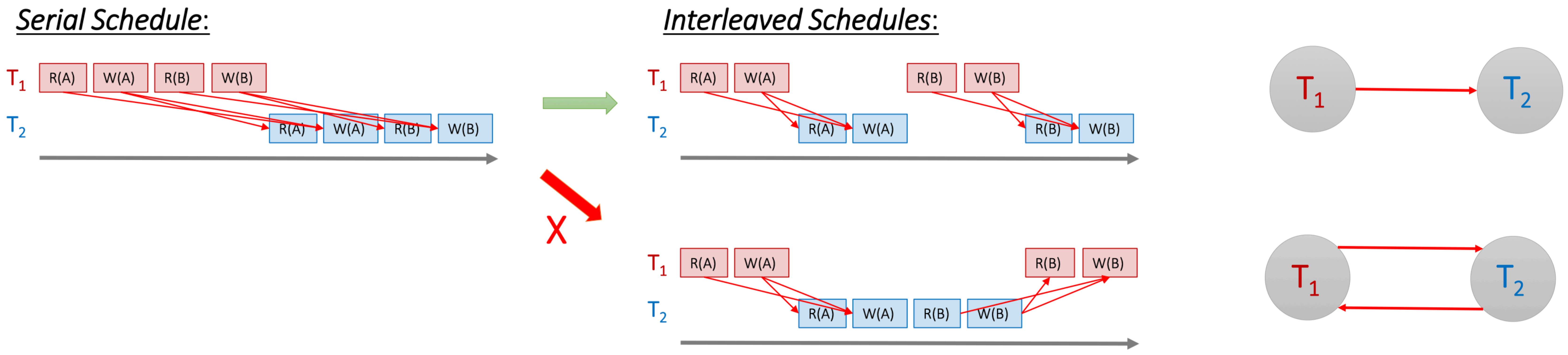
- The order of non-conflicting operations has no effect on the final state of the database
- We need more definitions
  - two schedules are conflict equivalent if:
    - they involve the same actions of the same transactions
    - every pair of conflicting actions of two transactions are ordered in the same way
  - schedule $\mathbb{S}$ is conflict serializable if $\mathbb{S}$ is conflict equivalent to some serial schedule
    - we like conflict serializable schedules

Conflict graph

- Looking at conflicts at the transaction level
  - there is an edge from $T_i \to T_j$ if an action in T_i precede and conflict with an action in T_j

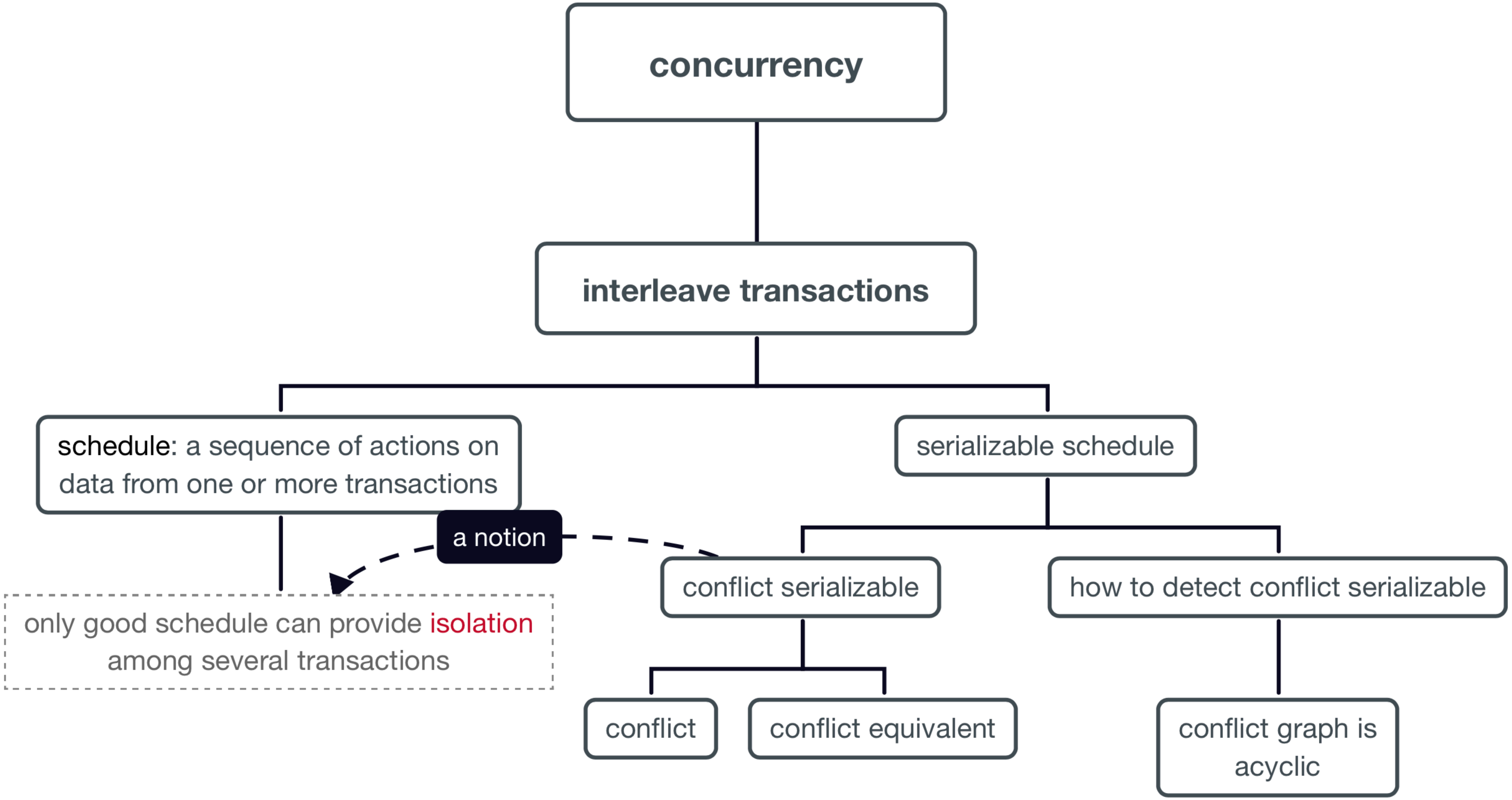


- Theorem: schedule is conflict serializable ⇔ its conflict graph is acyclic


Serial Schedule:

Interleaved Schedules:

A mindmap



concurrency

interleave transactions

schedule: a sequence of actions on data from one or more transactions

serializable schedule

a notion

only good schedule can provide isolation among several transactions

conflict serializable

how to detect conflict serializable

conflict

conflict equivalent

conflict graph is acyclic