

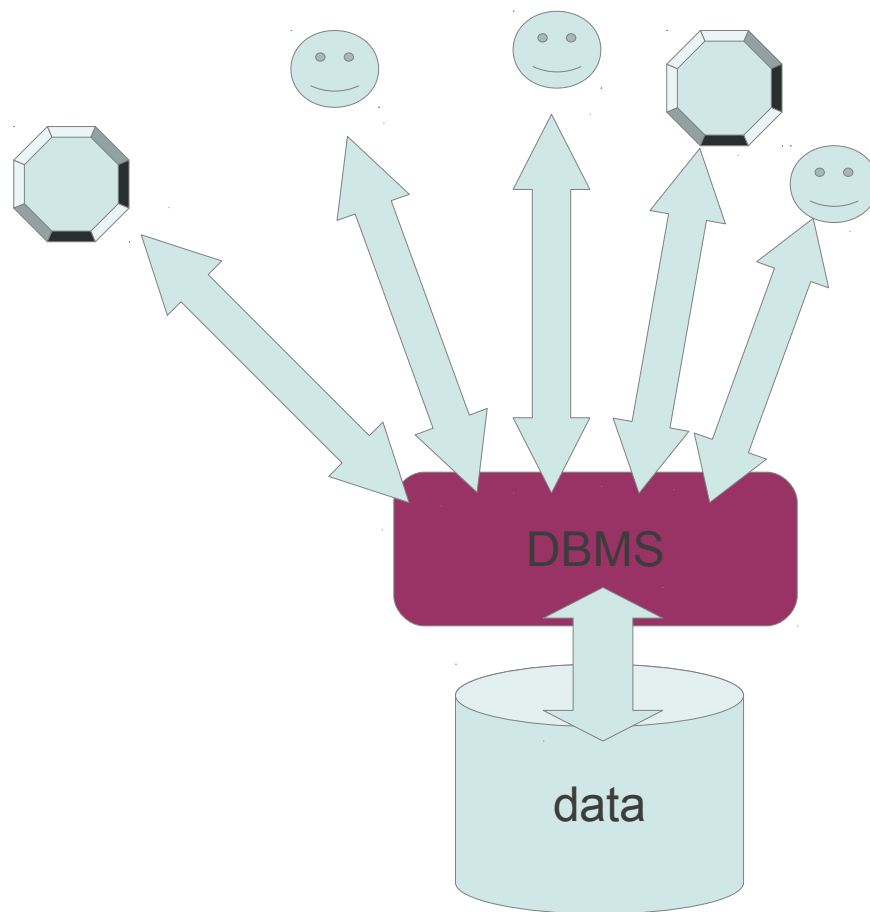
Databázové systémy

Transakcie

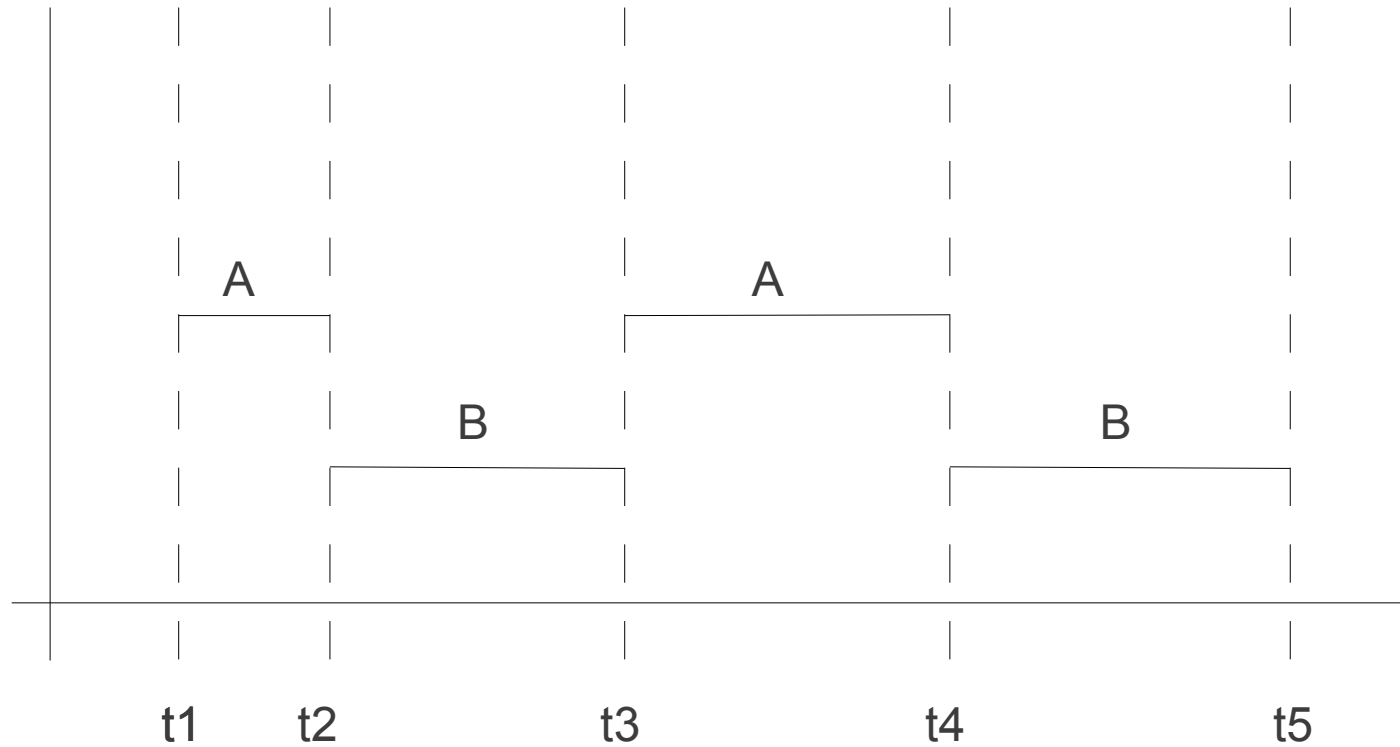
Motivácia

- paralelný prístup k databáze
- odolnosť databázy voči zlyhaniu

Paralelný prístup k databáze



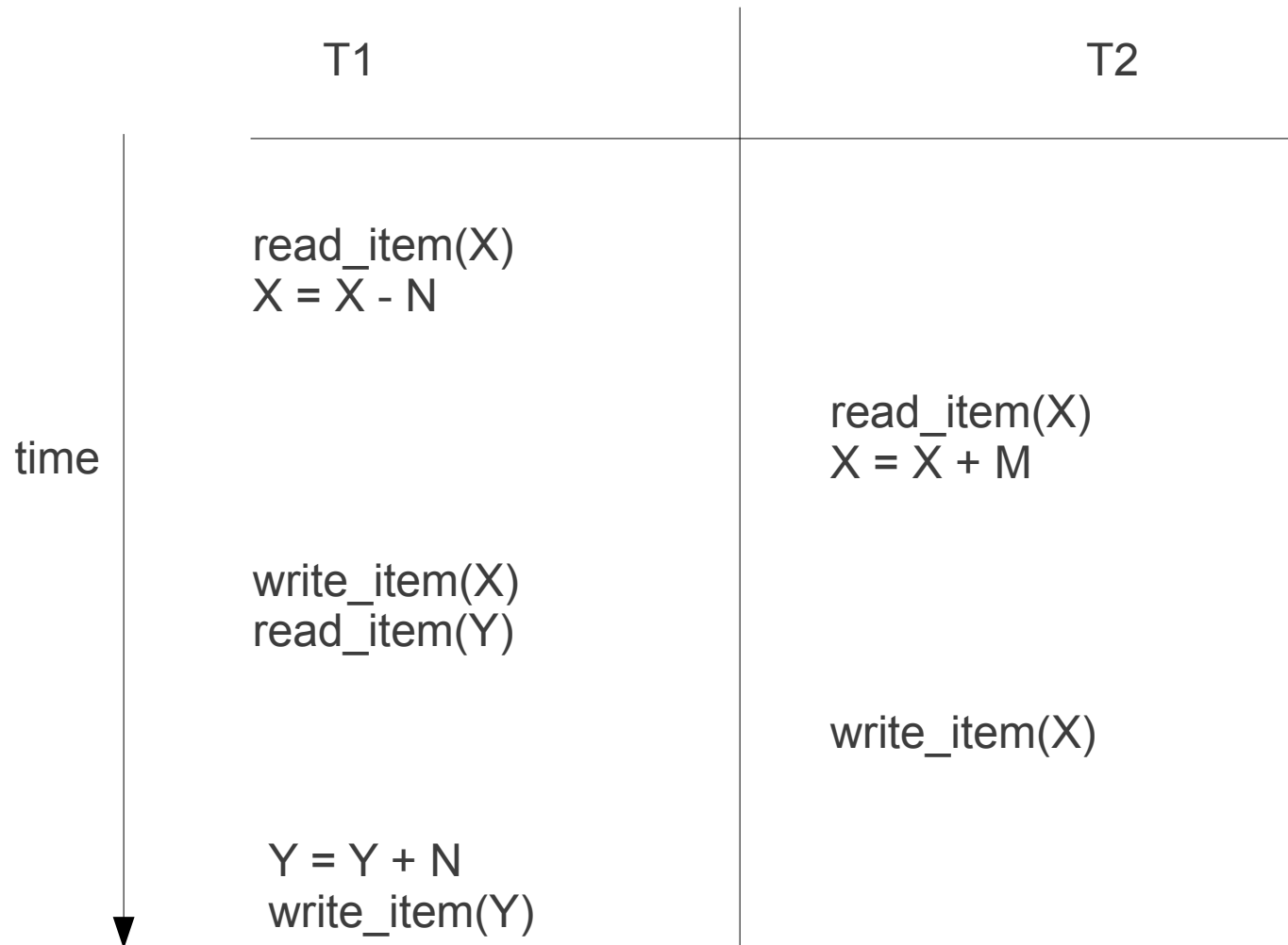
“Paralelizmus” na CPU



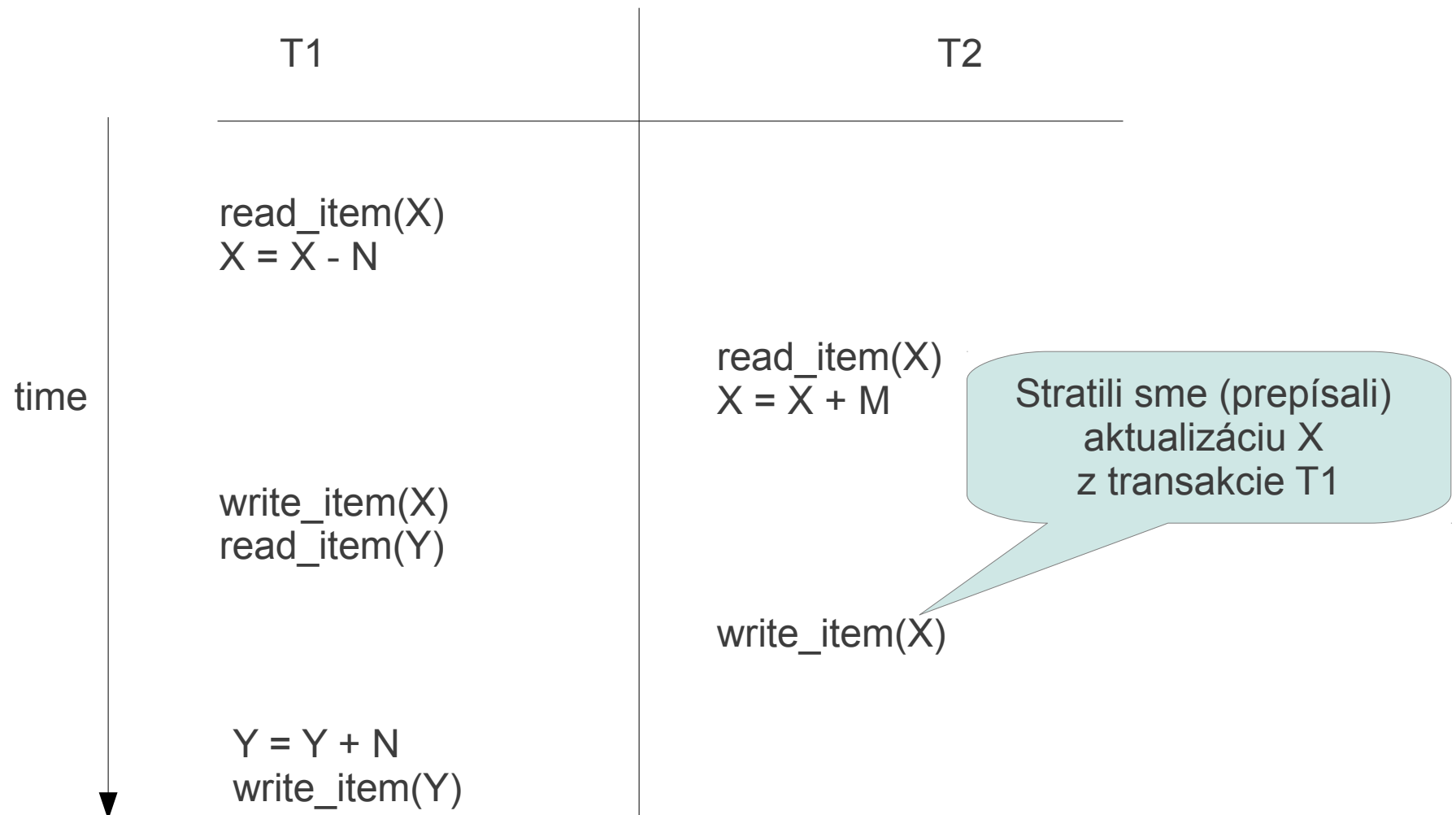
Prečo potrebujeme riadiť súbežnosť? (concurrency)

- Lost Update problem
- Dirty Read problem
- Incorrect Summary problem

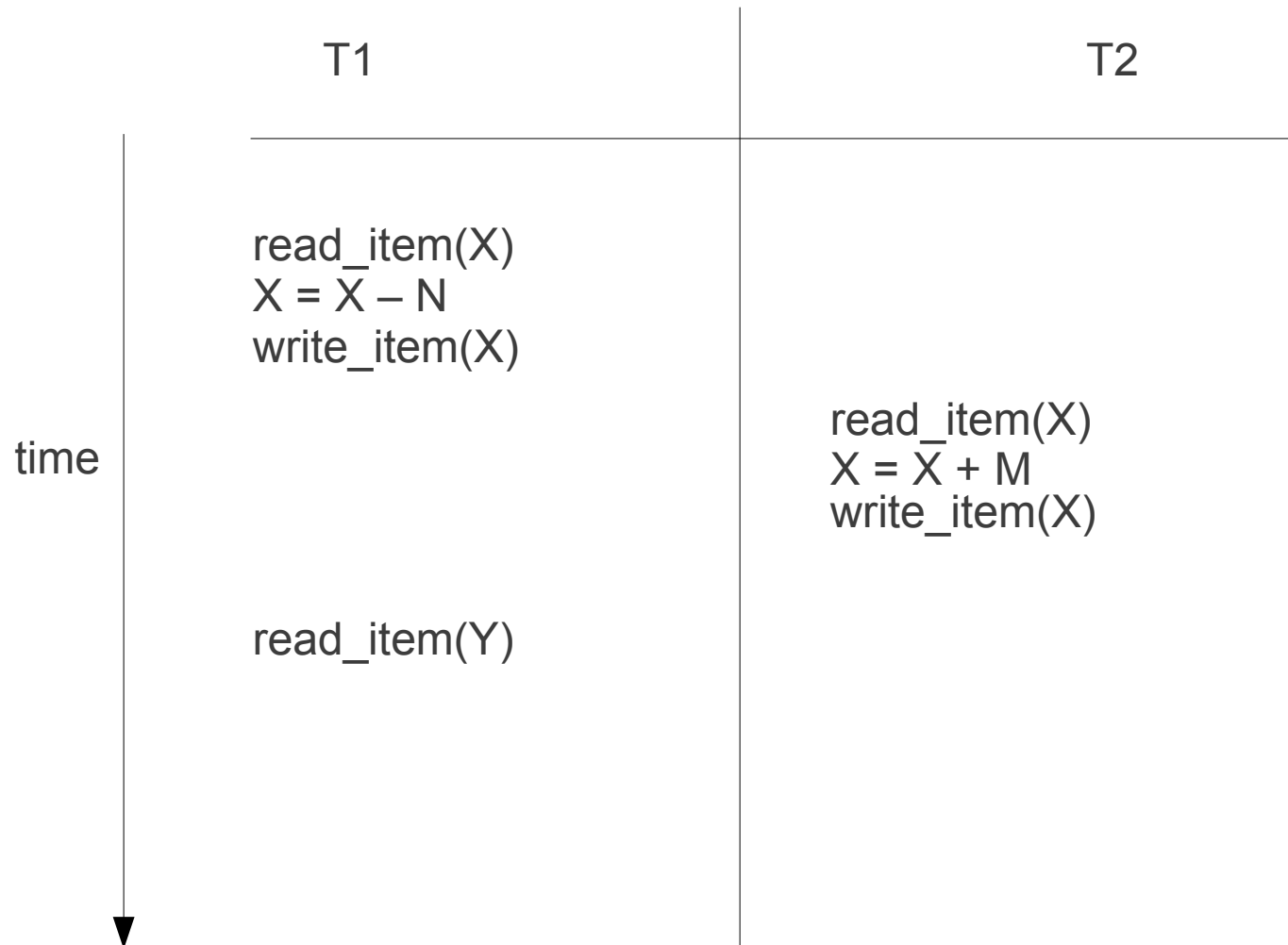
Lost Update problem



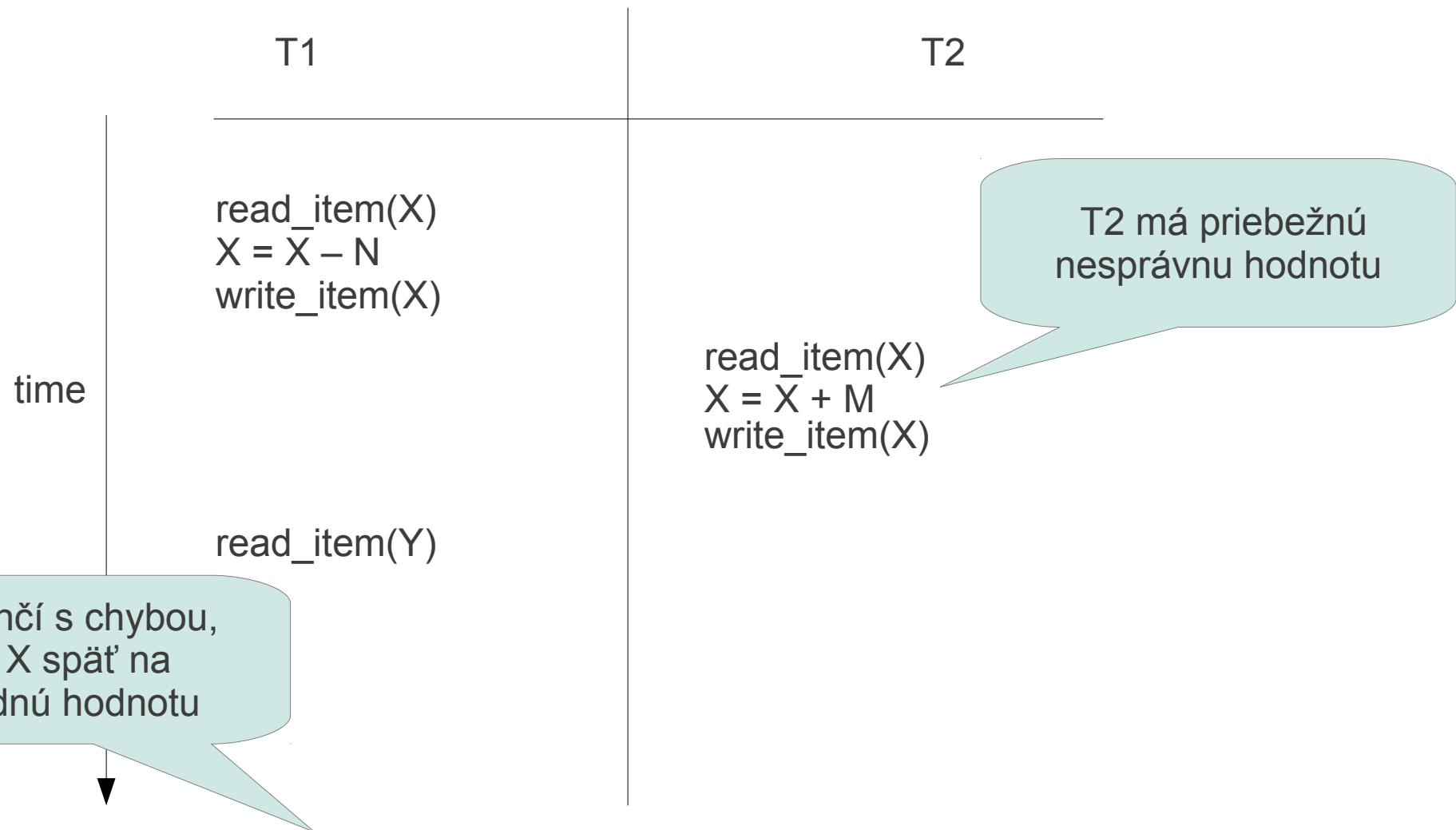
Lost Update problem



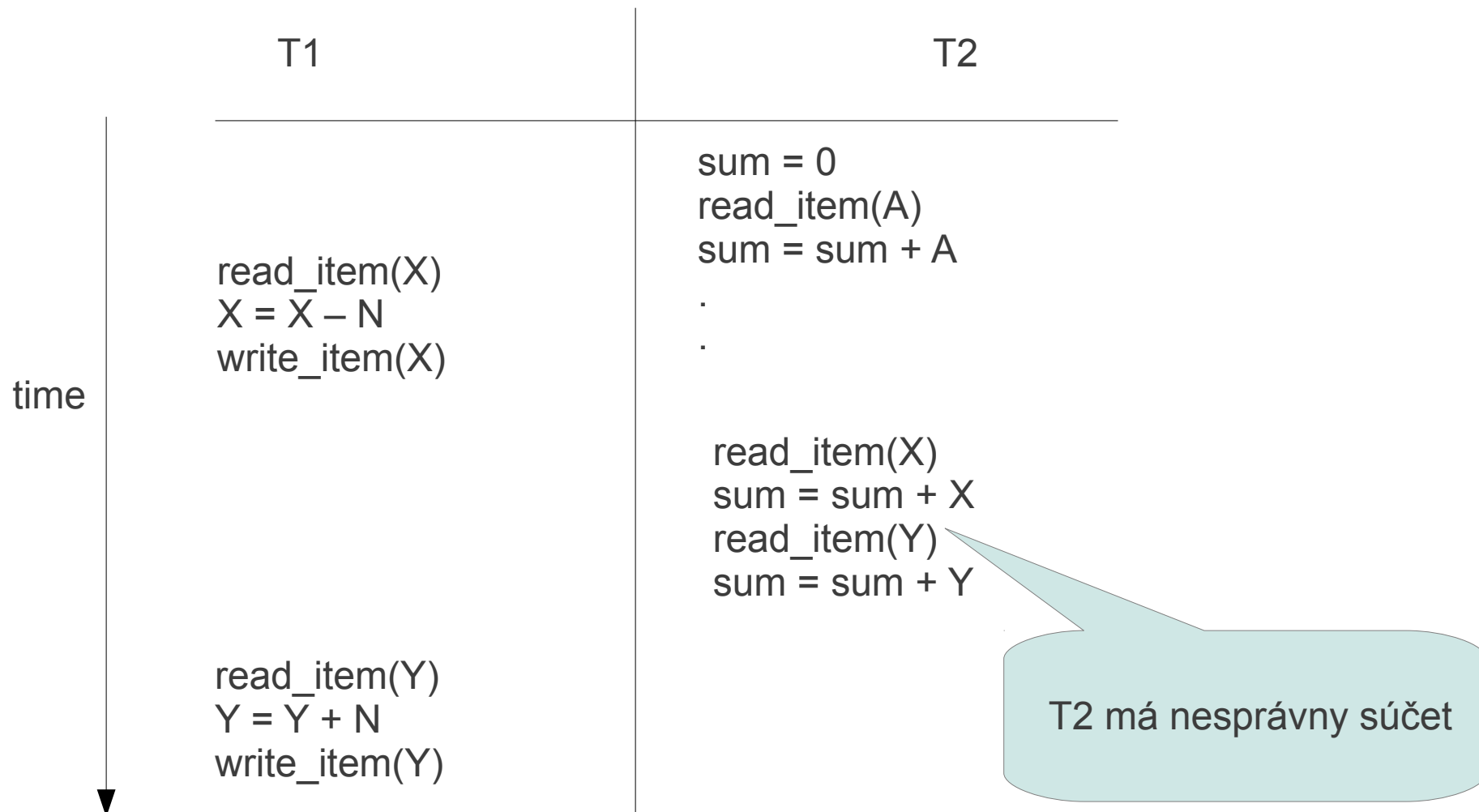
Dirty Read problem



Dirty Read problem



Incorrect Summary problem



Nekonzistencia na úrovni atribútov

```
UPDATE restaurants  
SET capacity = capacity + 100  
WHERE name = 'horna';
```

```
UPDATE restaurants  
SET capacity = capacity + 150  
WHERE name = 'horna';
```

- `get()`; `modify()`; `put()`

Nekonzistencia na úrovni n-tice

```
UPDATE restaurants  
SET capacity = capacity + 100  
WHERE name = 'horna';
```

```
UPDATE restaurants  
SET location = 'fakulty'  
WHERE name = 'horna';
```

Nekonzistencia na úrovni tabuľky

```
UPDATE lunches  
SET was_tasty = true  
WHERE student_in IN  
(SELECT id FROM students  
WHERE vsp < 2);
```

```
UPDATE students  
SET vsp = (1.1) * vsp  
WHERE name LIKE 'Michal%';
```

Nekonzistencia pri viacerých operáciách

```
INSERT INTO archive  
    SELECT * FROM lunches  
    WHERE was_tasty = true;  
DELETE FROM lunches  
WHERE was_tasty = true;
```

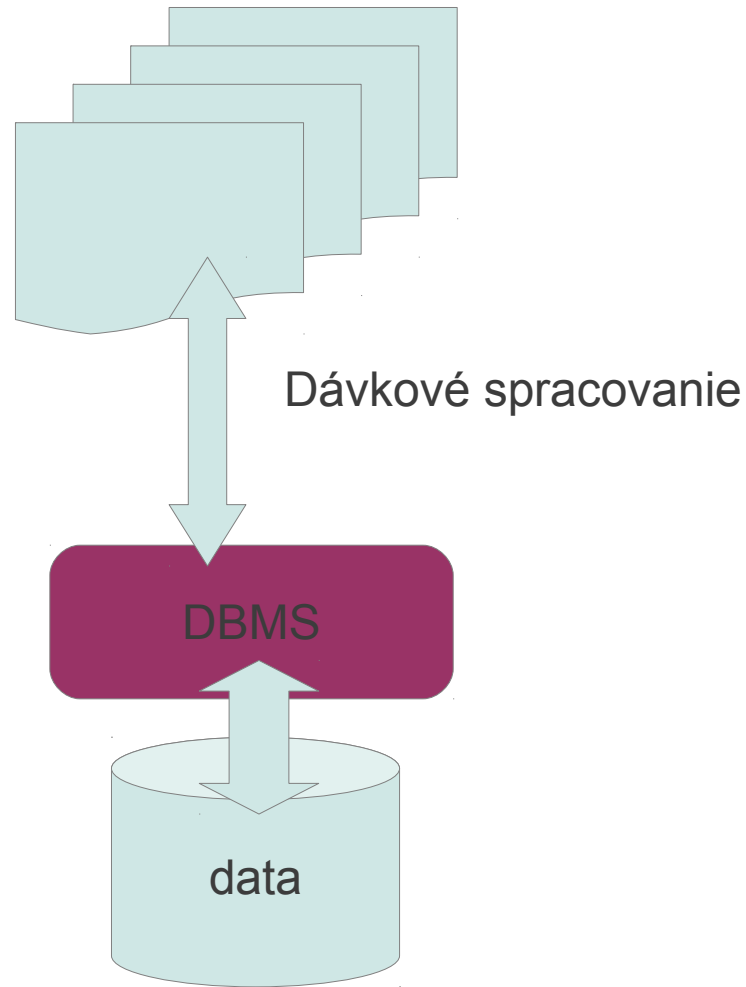
```
SELECT count(*) FROM archive;  
SELECT count(*) FROM lunches;
```

Cieľ riešenia súbežnosti

Vykonať jednotlivé SQL príkazy tak, aby výsledný efekt bol rovnaký ako keby jednotlivé transakcie bežali izolovane

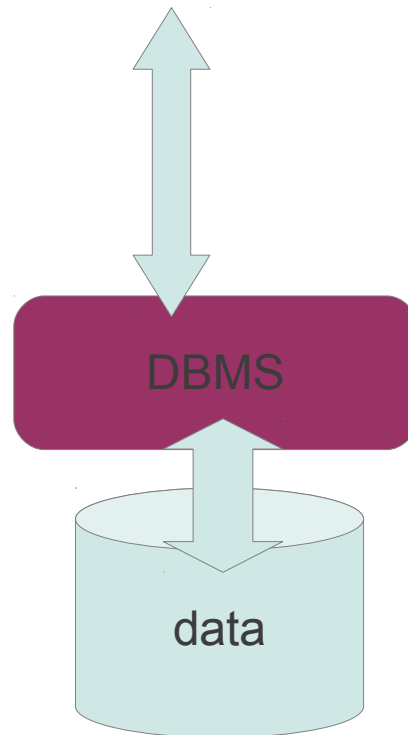
- Obvious riešenie: spustiť ich izolovane
 - My ale chceme súbežnosť
 - máme viacero CPU, viacero vlákien – ostali by nevyužívané
 - aj tak čakáme na drahé I/O operácie

Odolnosť voči zlyhaniu



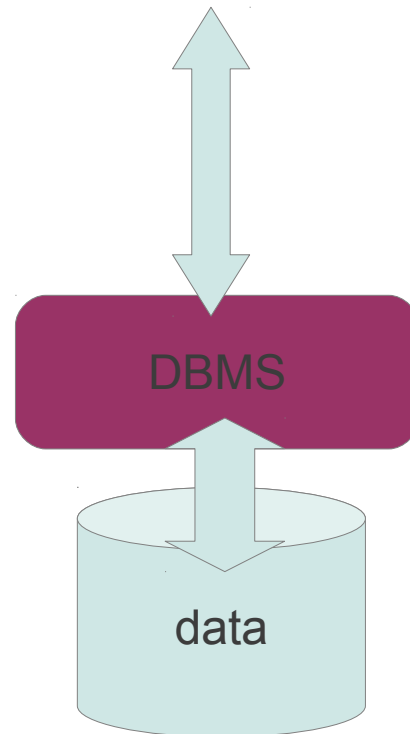
Odolnosť voči zlyhaniu

```
INSERT INTO archive  
  SELECT * FROM lunches  
  WHERE was_tasty = true;  
DELETE FROM lunches  
WHERE was_tasty = true;
```



Odolnosť voči zlyhaniu

veľa operácií nabuffrovaných v pamäti



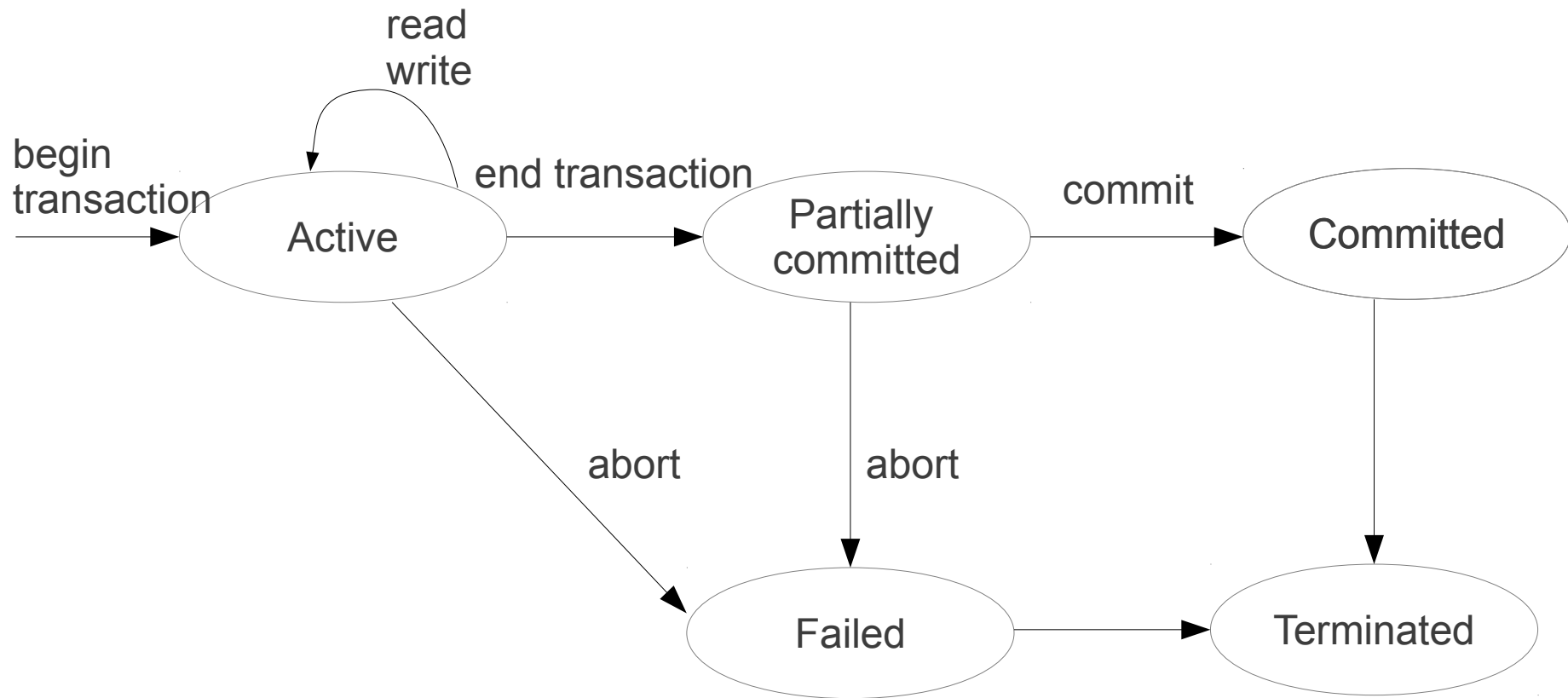
Transakcie

- sekvencia jedného alebo viacerých SQL príkazov
 - ktoré sú vnímane ako jedna jednotka
- zdá sa, ako keby bežali izolovane
- ak nastane zlyhanie tak sú zmeny z transakcie buď kompletne zachované alebo nie sú vôbec

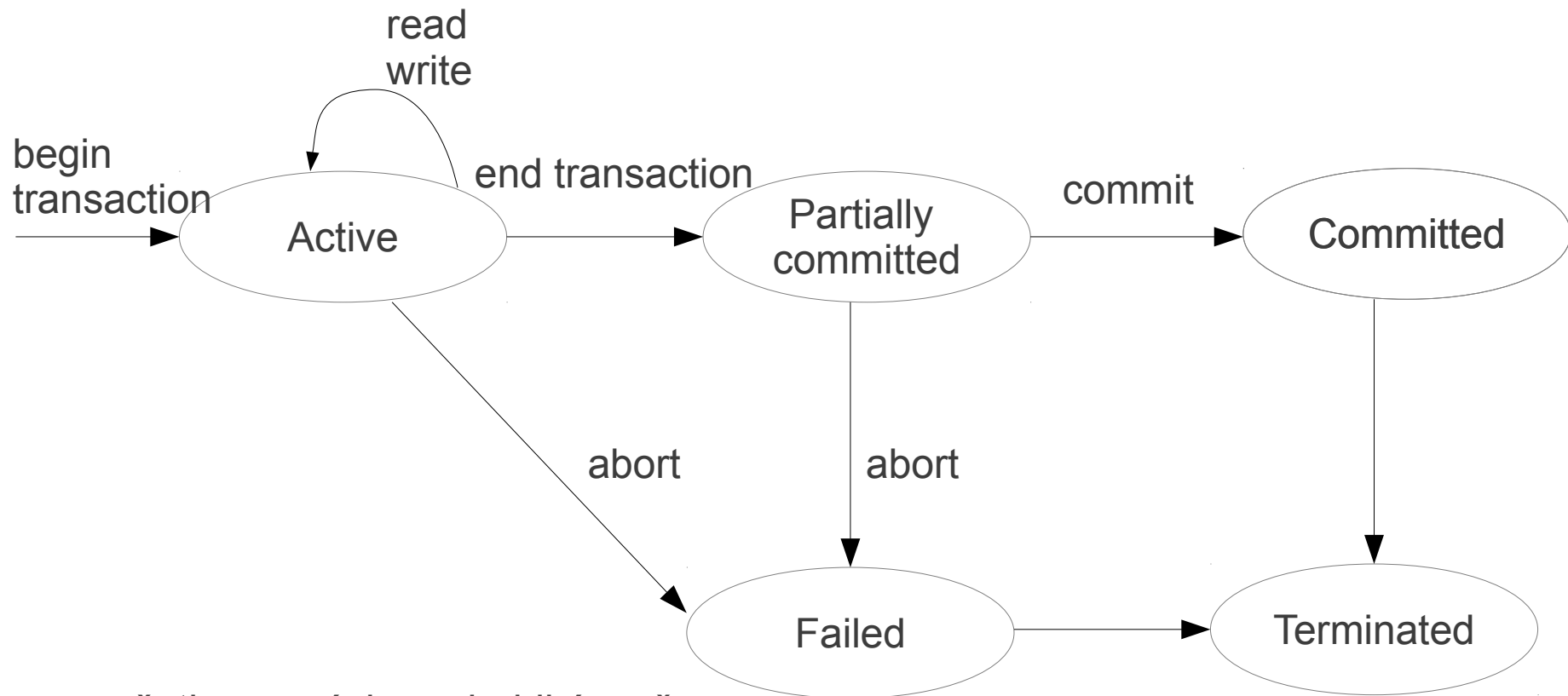
Transakcie

- sekvencia jedného alebo viacerých SQL príkazov
 - ktoré sú vnímané ako jedna jednotka
- Transakcia
 - začína automaticky na prvom SQL príkaze
 - `commit` končí transakciu a začína novú
 - `session.close` končí transakciu
 - `autocommit` mód mení každý príkaz na transakciu

Stavy transakcie



Commit point



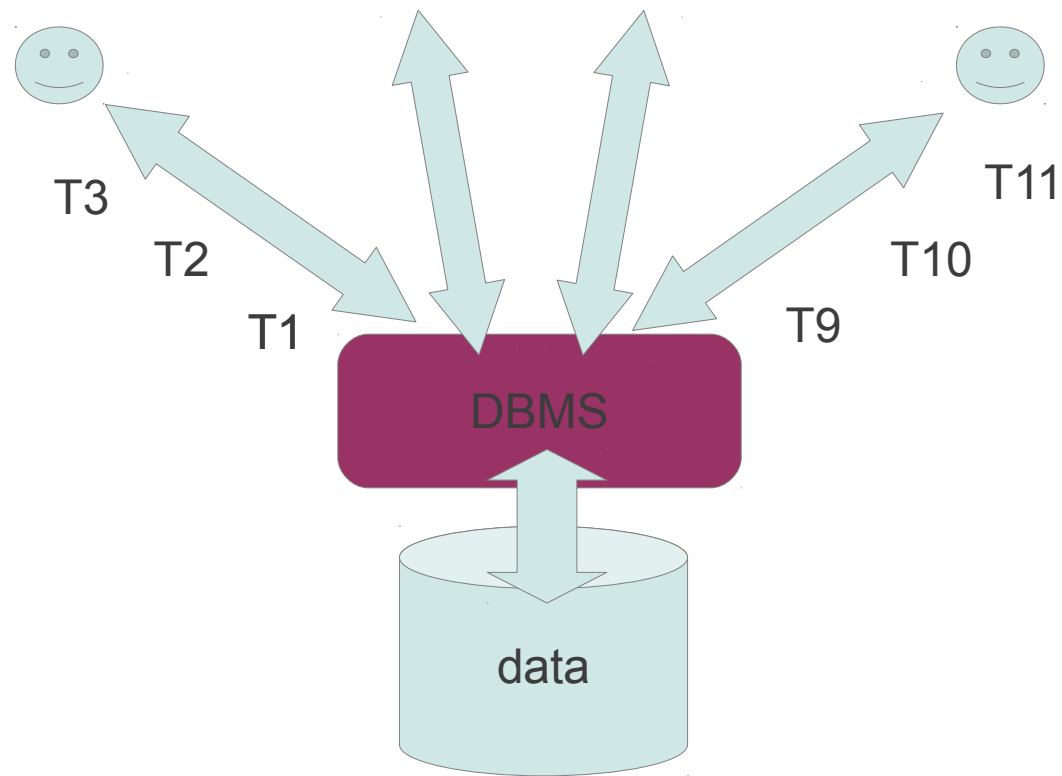
všetky operácie prebehli úspešne
ich dopad bol zaznamenaný do logu

zapíšem zmeny na disk
potom môžem do logu zapísať commit

ACID

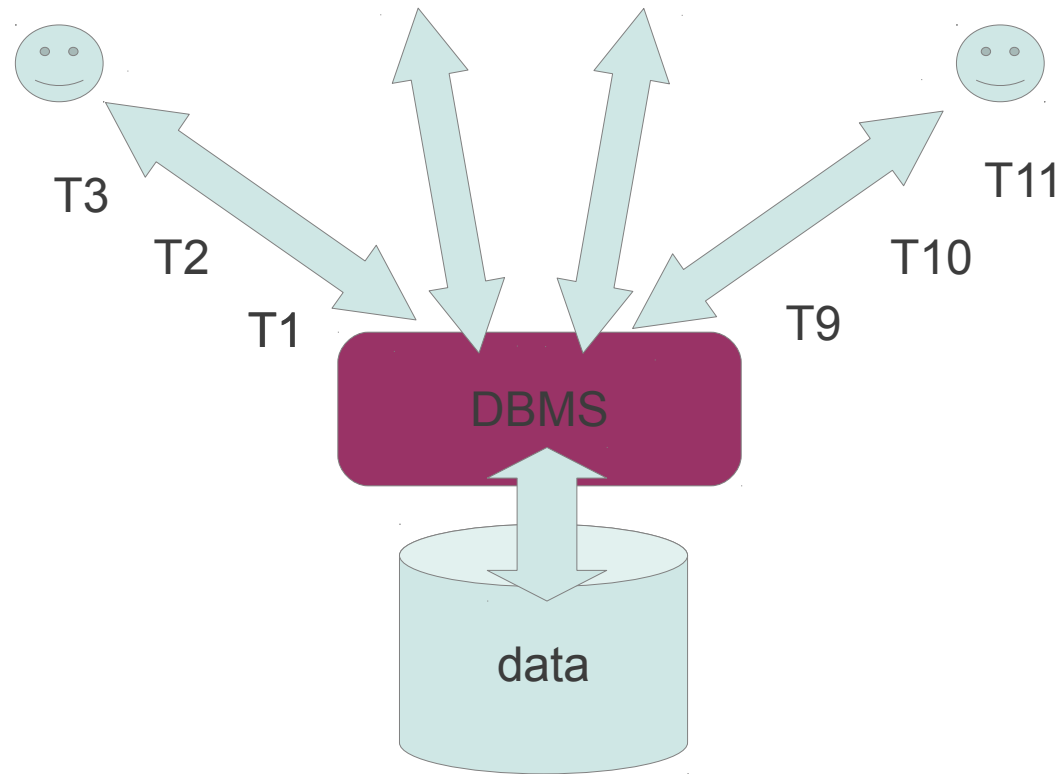
- Atomicity
- Consistency
- Isolation
- Durability

Isolation



Isolation

- Serializovateľnosť
 - Operácie môžu byť prekladané, ale vykonanie musí byť ekvivalentné *niektorej* sekvencii vykonania všetkých transakcií



Isolation – ako?

`read_lock()`

`write_lock()`

`unlock()`

- Two-Phase Locking
 - všetky `lock()` operácie sú vykonané pred prvou `unlock()` operáciou
- Deadlock, starvation

Durability

- Ak niečo zlyhá potom, ako bol vykonaný `commit`, všetky efekty spôsobené transakcie zostanú zachované
- transaction logging

Atomicity

- Každá transakcia buď dobehne celá (až po commit) alebo sa jej efekt v databáze vôbec neprejaví
- opäť vďaka transakčnému logu
- Transaction Rollback (Abort) - undo
 - môže byť vyvolaný aj aplikáciou

Príklad

- Začni transakciu
- <získaj vstupy od používateľa>
- SQL príkazy na základe vstupov
- <spýtaj sa používateľa, či je OK s výsledkom>
 - Ak je OK, tak `commit()`
 - Ak nie je OK, tak `rollback()`

Zlý příklad

- Začni transakciu
- <získaj vstupy od používateľa>
- SQL príkazy na základe vstupov
- <spýtaj sa používateľa, či je OK s výsledkom>
 - Ak je OK, tak `commit()`
 - Ak nie je OK, tak `rollback()`



Nemení stav sveta...(napr. bankomat)

Consistency

- Všetky constraints sú splnené pred začiatkom transakcie
- Všetky constraints sú splnené po ukončení transakcie
 - či skončila commitom alebo rollbackom

Ešte k isolation

- Serializovateľnosť
 - réžia
 - redukcia súbežného využitia databázy
 - pamätáte si ... zámky

Úrovne izolácie

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializovateľnosť

slabšie

silnejšie

- Nižšia réžia
- Vyššia súbežnosť
- Nižšia garancia konzistentnosti

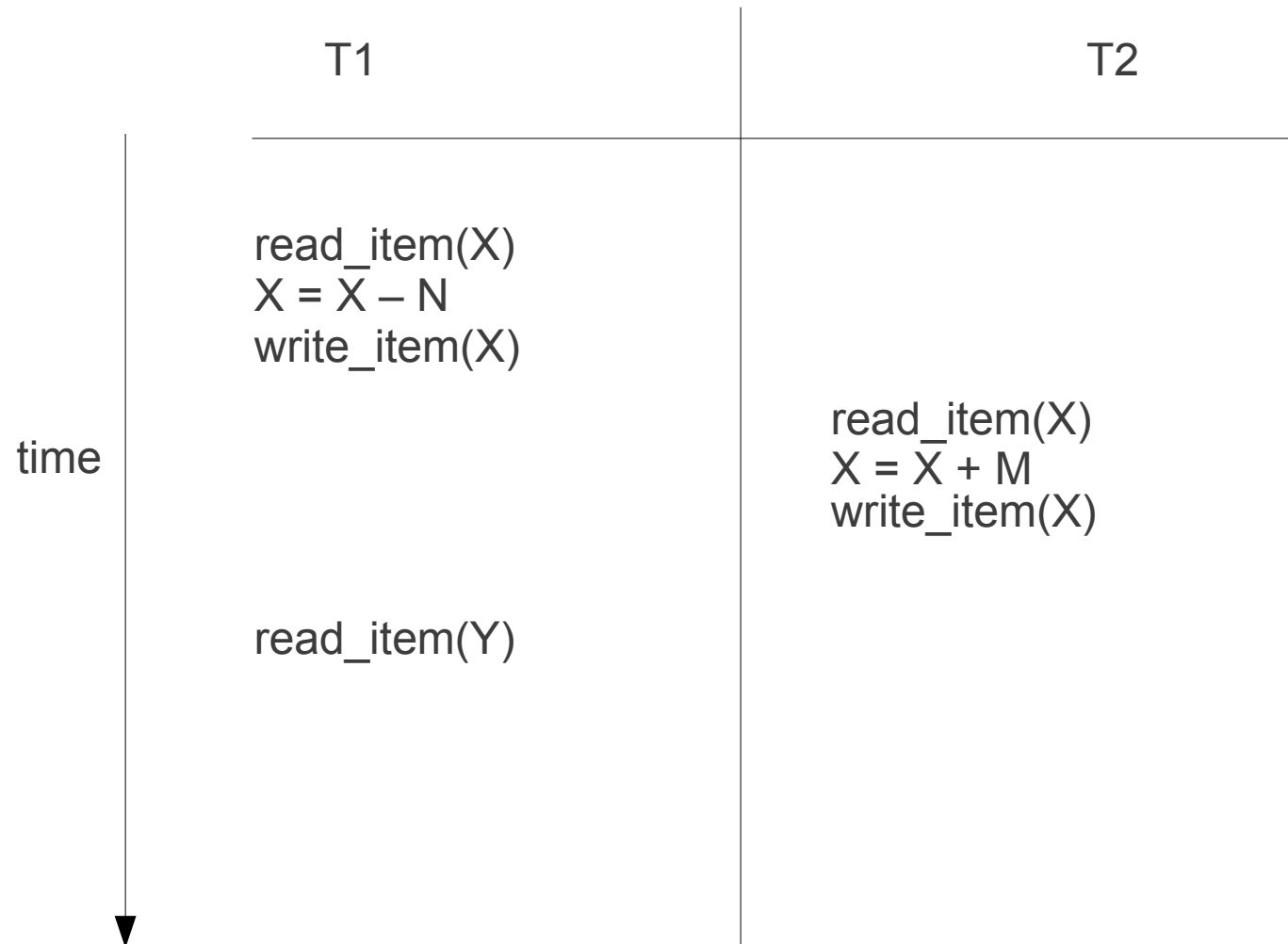
Úroveň izolácie

- je vždy vlastnosť transakcie
- určuje si ju klient
- neovplyvňuje úroveň izolácie iných transakcií

Read uncommitted

- povere mi DirtyReads

Dirty Read problem



Read uncommitted

- povoľte mi DirtyReads
- Napr. rátam priemery a nevadí mi, ak to nebude úplne presné

Read Committed

- Transakcia môže čítať iba také údaje, ktoré boli committed
 - takže žiadne DirtyRead
 - napriek tomu nie je zabezpečená serializovateľnosť

```
UPDATE students SET vsp = 1.1 * vsp;
```

```
SET TRANSACTION ISOLATION LEVEL READ  
COMMITTED
```

```
SELECT avg(vsp) FROM students;
```

```
SELECT max(vsp) FROM students;
```

Repeatable Reads

- Transakcia nemôže robiť Dirty Read
- Viackrát načítaná hodnota sa nemôže zmeniť

```
UPDATE students SET vsp = 1.1 * vsp;  
UPDATE students SET credits = 60 where  
name LIKE 'Michal%'
```

```
SET TRANSACTION ISOLATION LEVEL  
REPEATABLE READ  
  
SELECT avg(vsp) FROM students;  
SELECT avg(credits) FROM students;
```

Repeatable Reads

- fantómové n-tice

```
INSERT INTO students [100 n-tíc]
```

```
SET TRANSACTION ISOLATION LEVEL  
REPEATABLE READ  
SELECT avg(vsp) FROM students;  
SELECT avg(credits) FROM students;
```


Repeatable Reads

- fantómové n-tice
 - toto by už neprešlo

```
DELETE FROM students [100 n-tíc]
```

```
SET TRANSACTION ISOLATION LEVEL  
REPEATABLE READ  
SELECT avg(capacity) FROM students;  
SELECT avg(credits) FROM students;
```

Zhrnutie

- ACID
- Transakcie
 - commit
 - rollback
- Isolation levels