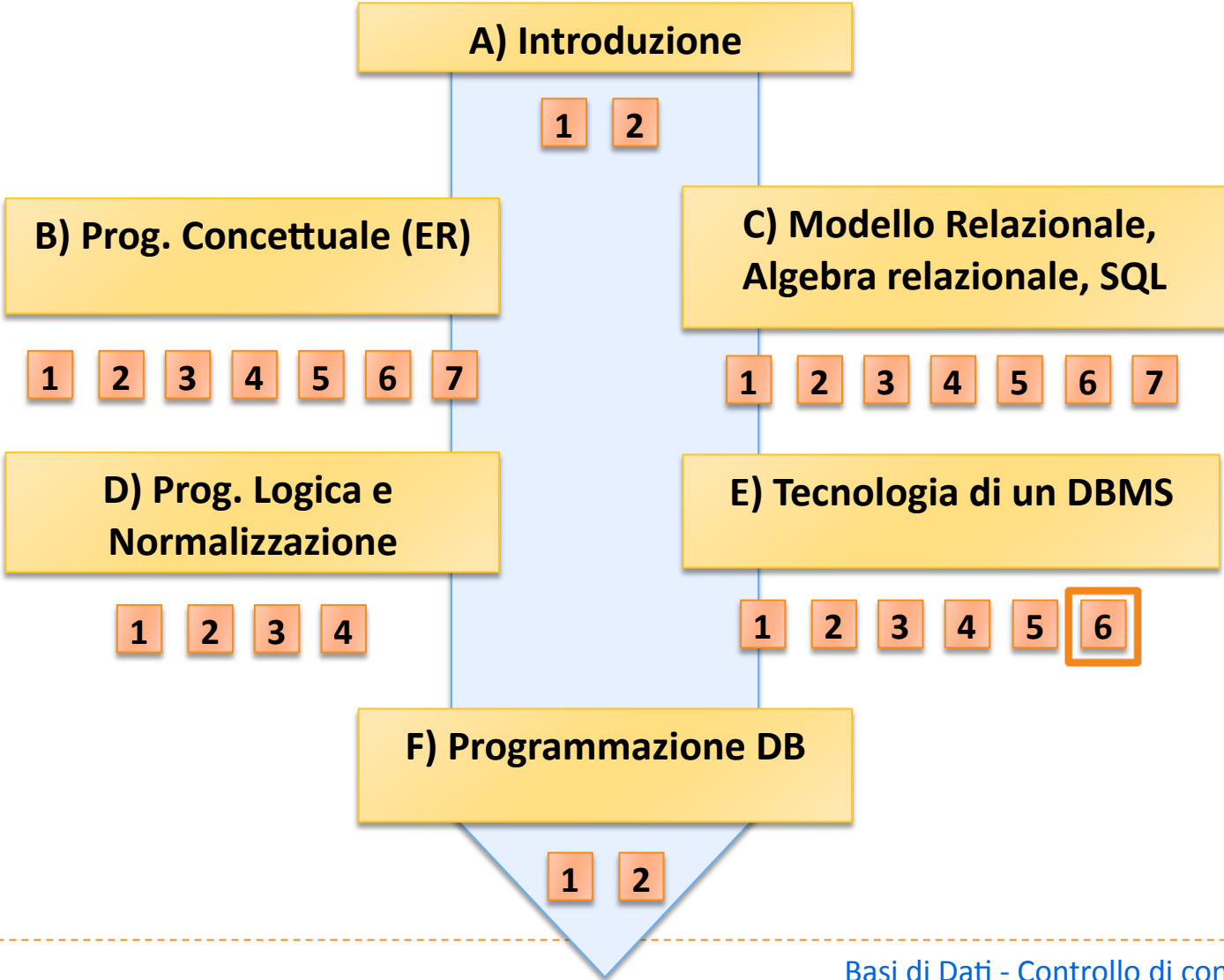


# Basi di Dati

## Controllo di Concorrenza

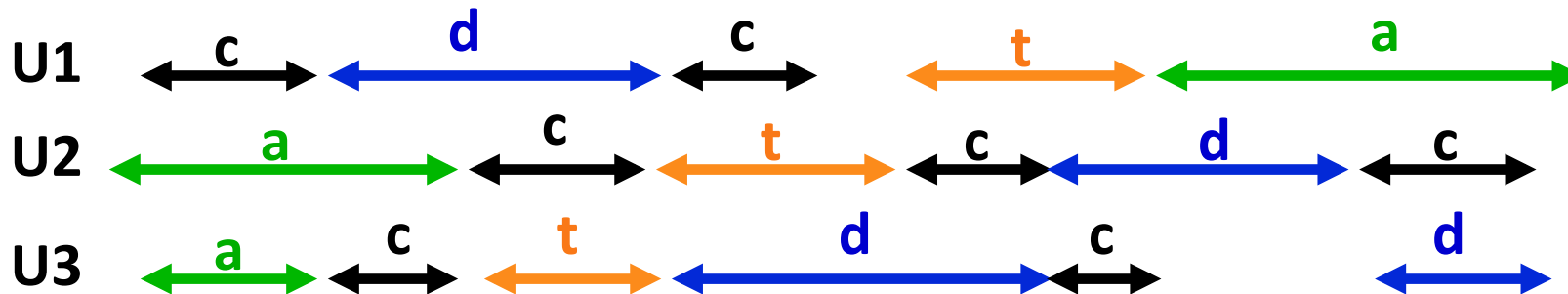
# Basi di Dati – Dove ci troviamo?



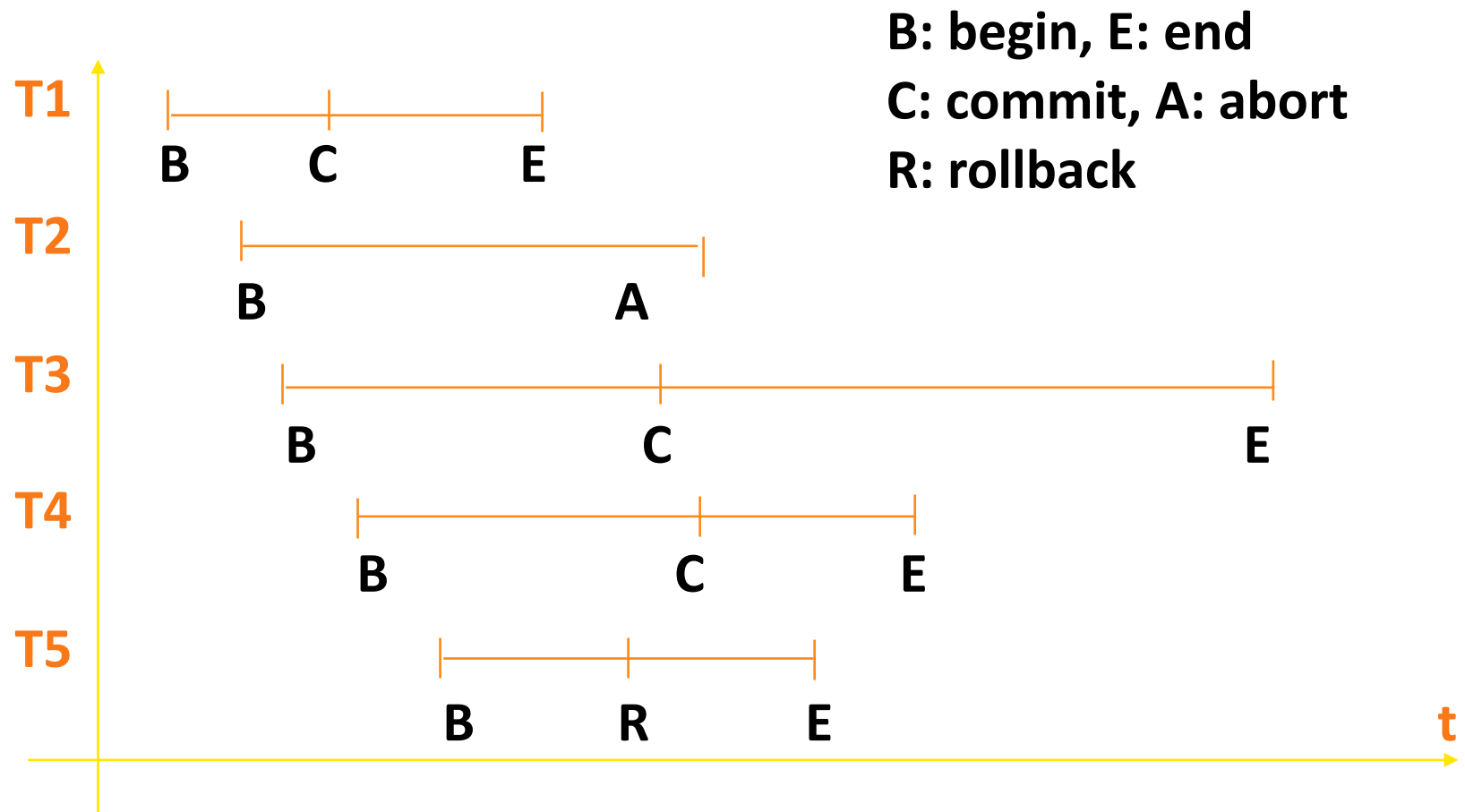
# I vantaggi della concorrenza

---

- ▶ sul server (1 cpu, 1 disco, 1 sistema di trasmissione) è possibile il parallelismo tra:
  - ▶ elaborazione: cpu (c)
  - ▶ operazioni di I/O: disco (d)
  - ▶ operazioni di trasmissione (t)
  - ▶ attesa dai client (a)



# I vantaggi della concorrenza



# Problemi dovuti alla concorrenza

---

**Lost update** (dipendenza **write** → **write**)

perdita di modifiche da parte di una transazione a causa di un aggiornamento effettuato da un'altra transazione

**Dirty read** (dipendenza **write** → **read**)

dipendenza di una transazione da un'altra non completata con successo e lettura di dati inconsistenti

**Unrepeatable read** (dipendenza **read** → **write**)

analisi inconsistente di dati da parte di una transazione causata da aggiornamenti prodotti da un'altra

# Esempi

---

**T1 : UPDATE CC**

**SET SALDO = SALDO + 3**

**WHERE CLIENTE = Tiberio**

**T2 : UPDATE CC**

**SET SALDO = SALDO + 6**

**WHERE CLIENTE = Tiberio**

# Esecuzione con perdita di update (lost update problem)

---

**S(tib)=100**

**1 R(T1): S(tib) -> B1**

**2 B1 = B1 + 3**

**3 R(T2): S(tib) -> B2**

**4 B2 = B2 + 6**

**5 W(T1): B1 -> S(tib) S(tib)=103**

**6 W(T2): B2 -> S(tib) S(tib)=106 (finale), oppure:**

**5' W(T2): B1 -> S(tib) S(tib)=106**

**6' W(T1): B2 -> S(tib) S(tib)=103 (finale)**

# Lost update: Sequenza che produce l'errore

---

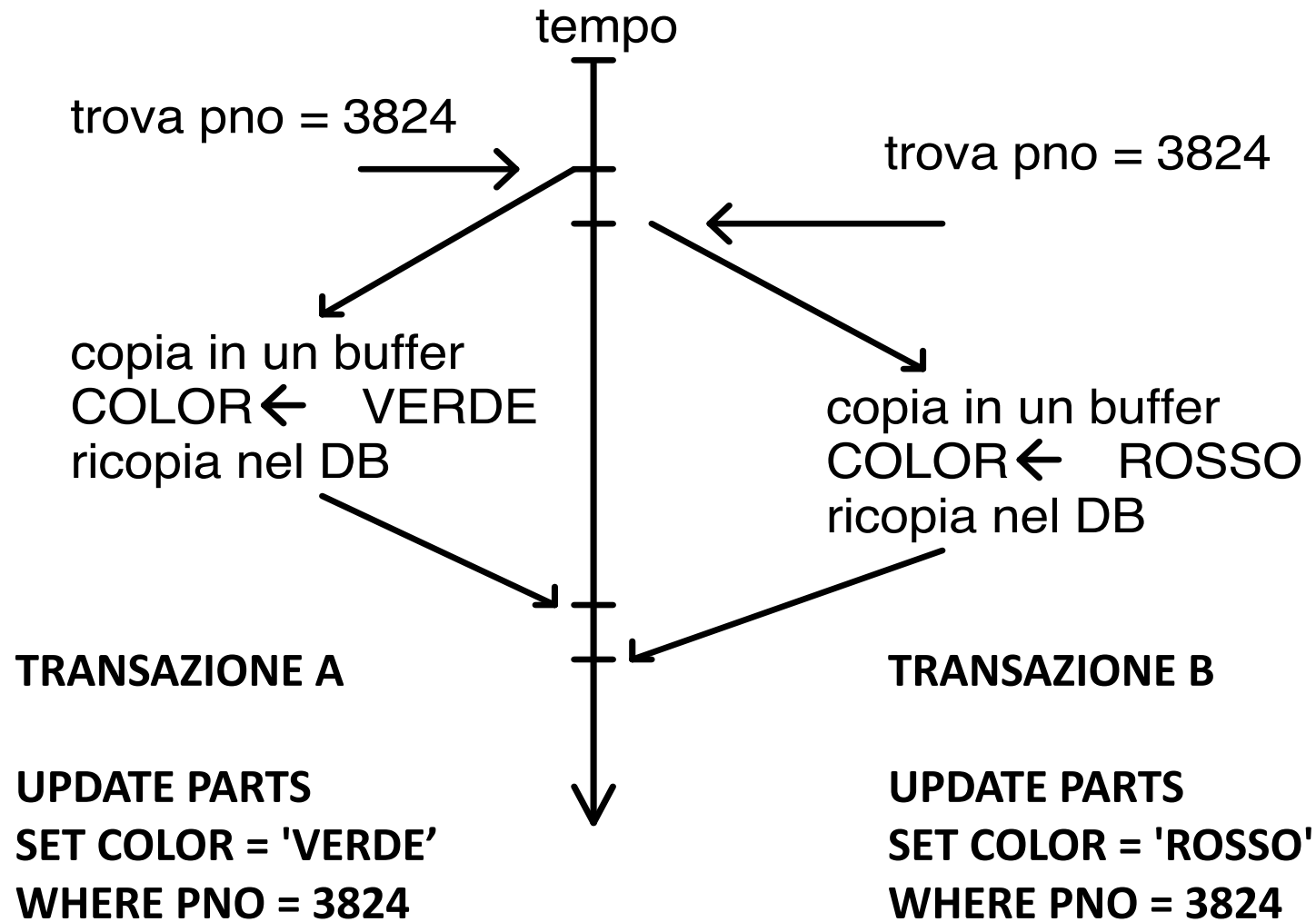


oppure





# Lost update: altro esempio



# Lettura “sporca” (dirty read)

---

**S(tib)=100**

**1 R(T1): S(tib) -> B1**

**2 B1 = B1 + 3**

**3 W(T1): B1 -> S(tib) S(tib)=103**

**4 R(T2): S(tib) -> B2**

**5 ROLLBACK(T1)**

**6 B2 = B2 + 6**

**7 W(T2): B2 -> S(tib) S(tib)=109**

# Unrepeatable read

---

**S(tib)=100**

**1 R(T1): S(tib) = 100**

**2 R(T2): S(tib) -> B2**

**3 B2 = B2 + 100**

**4 W(T2): B2 -> S(tib) S(tib)=200**

**5 R(T1): S(tib) = 200**

# Controllo di concorrenza

---

Si vuole garantire una esecuzione concorrente equivalente alla esecuzione seriale:

**serializzabilità**

**TECNOLOGIA PREVALENTE :**

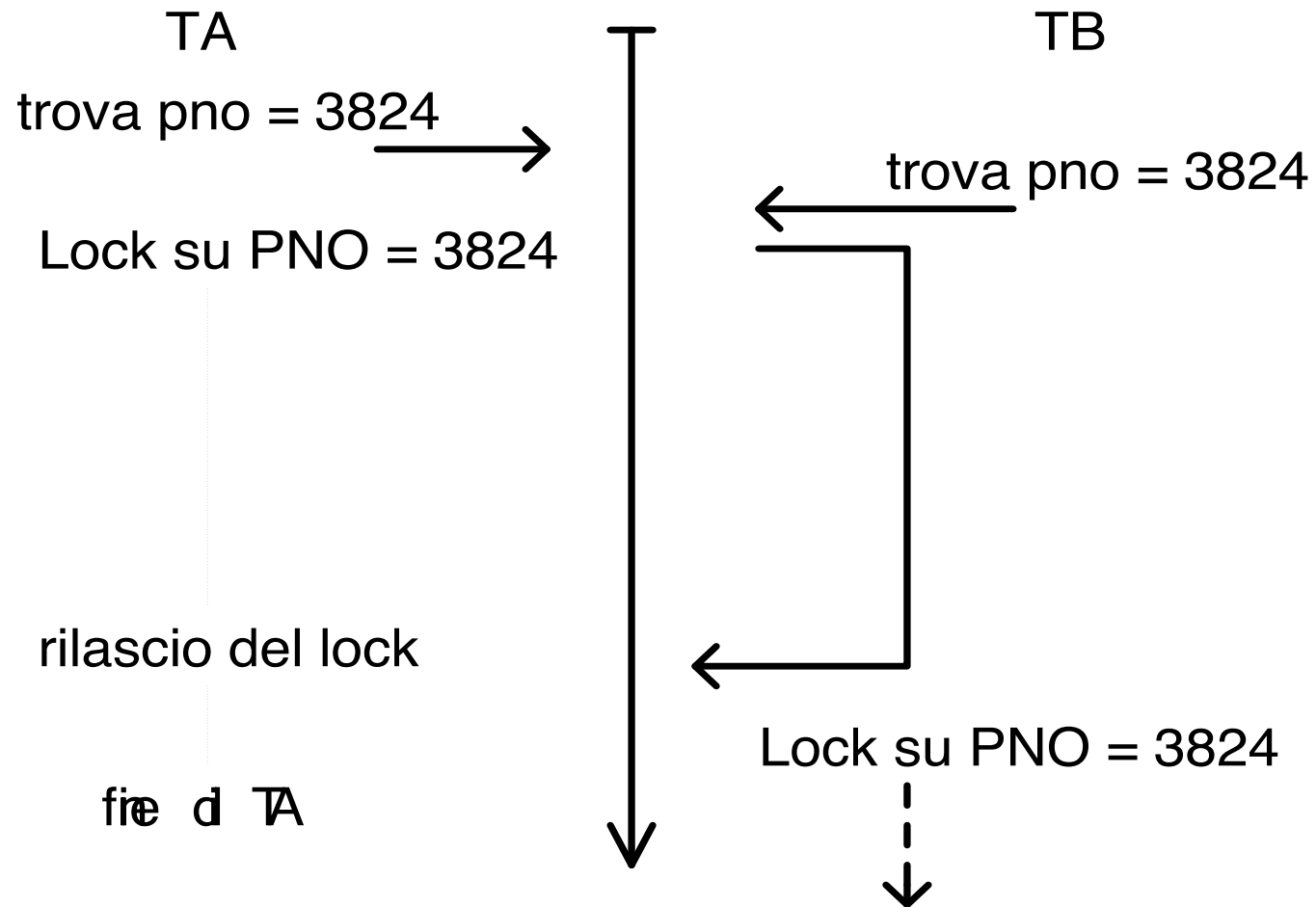
**LOCKING**

Per evitare il problema, TA acquisisce un "**lock**" sul record PNO = 3824 (oppure sulla pagina che contiene il record, oppure sulla relazione che contiene il record). TB entra in "**stato di wait**".

Quando TA rilascia il lock TB può visitare il dato.

# Locking

---



# Primitive di lock

---

**r-lock:** lock in lettura (shared)

**w-lock:** lock in scrittura (exclusive)

**unlock**

**STATO DI UN OGGETTO:**

**libero**

**r-locked** (bloccato da un lettore)

**w-locked** (bloccato da uno scrittore)

# Transazioni ben formate

---

- ogni **read** di un oggetto è preceduto da r-lock ed è seguito da unlock
- ogni **write** di un oggetto è preceduto da w-lock ed è seguito da unlock

Un r-lock entra in conflitto con un w-lock (e viceversa), mentre non entra in conflitto con un altro r-lock (cioè due transazioni di lettura non interferiscono)

Un w-lock entra in conflitto con un w-lock (e viceversa)

# Tabella dei conflitti

	r-locked	w-locked
r-lock	OK	NO
w-lock	NO	NO

**OK** : blocco della risorsa,  
il programma procede  
**NO** : il programma  
va in attesa che  
la risorsa venga sbloccata

**contatore dei lettori**

**r-lock**

**r-counter** <- r-counter + 1

**unlock**

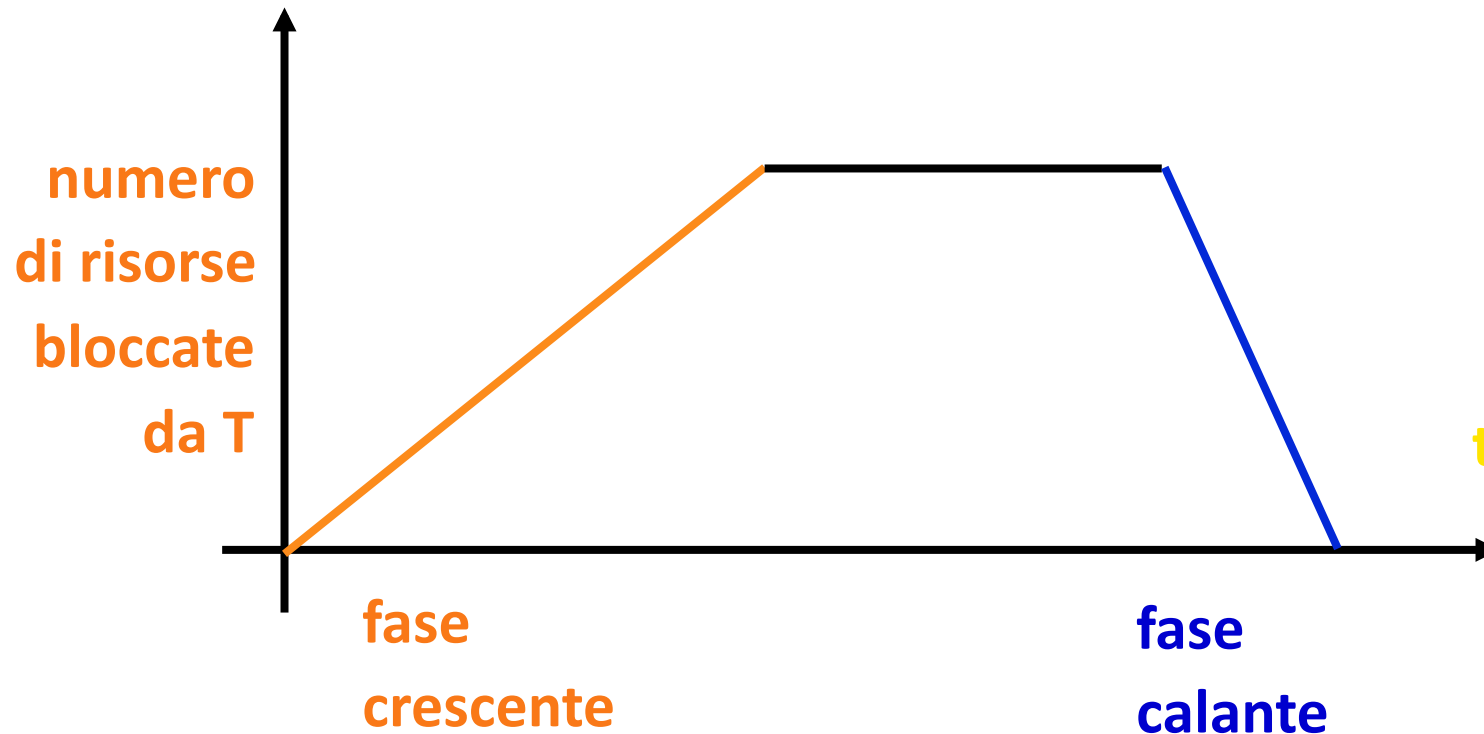
**r-counter** <- r-counter - 1



## Locking a due fasi

---

Per una transazione T una azione di unlock non può precedere una azione di lock



# Conseguenze

---

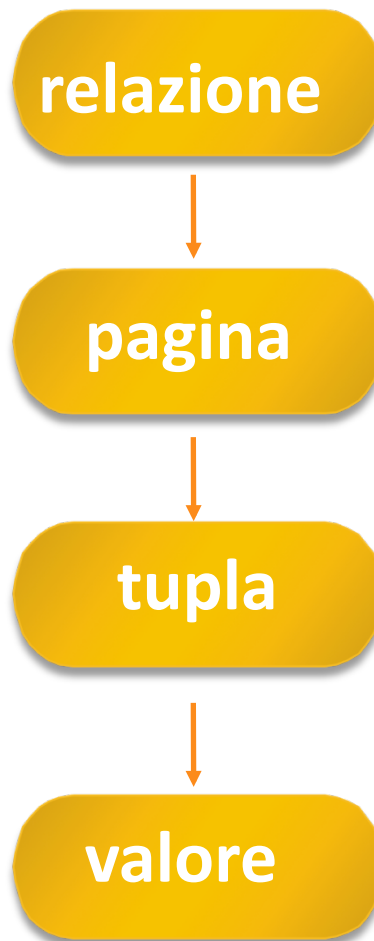
- a** transazioni ben formate
- b** politica dei conflitti come da tabella
- c** locking a due fasi

↓  
**serializzabilità**

# Granularità del locking

---

lock  
a livello di:  
**relazione**  
**pagina**  
**tupla**  
**valore** di un  
attributo  
di una tupla



(più la  
granularità  
è ridotta  
più è  
elevata la  
concorrenza)

# Sistema di locking

---

I **LOCK** vengono acquisiti in caso di modifica dei dati, oppure quando si vogliono leggere dati "**consistenti**".

Ad esempio, supponiamo che A voglia leggere la relazione EMP senza che avvengano modifiche durante la lettura e che B voglia modificare alcune n-ple di EMP.

Quando un utente vuole operare in modo esclusivo su un oggetto, acquisisce un **INTENTION LOCK**; potrà ottenere un **W-LOCK** solo quando il precedente **W-LOCK** verrà rilasciato e verrà il suo turno tra coloro che hanno posto **INTENTION LOCK**

# Sistema di locking

---

DEFINIZIONE di **UNCOMMITTED DATA** (dati non definitivi):  
dati soggetti a modifica, dati che sono stati modificati da  
una transazione ancora in funzione (e che potrebbe  
"ABORTIRE").

## REGOLA PER LA MODIFICA:

**Nessuna transazione può modificare dati uncommitted.**

Altrimenti, si potrebbe verificare la perdita delle  
modifiche, se una transazione che stava modificando gli  
stessi dati abortisce.

→ si evita il problema del **LOST UPDATE**

# Sistema di locking

---

Quando una transazione vuole *modificare* i dati deve acquisire **W-LOCK** (exclusive).

Quando una transazione vuole solo *leggere dati consistenti* deve acquisire **R-LOCK** (shared).

Nei sistemi commerciali sono possibili 3 livelli di isolamento per la lettura:

## LIVELLO 1

Una transazione T vuole leggere qualsiasi dato, anche uncommitted, quindi non chiede r-lock. Rileggendo due volte lo stesso dato, T può trovarlo cambiato perché:

- 1 un'altra transazione T3, che lo aveva cambiato in precedenza, abortisce (**DIRTY READ**).
- 2 un'altra transazione T2 lo ha cambiato (**UNREPEATABLE READ**)

# Sistema di locking

---

## LIVELLO 2

Una transazione T chiede un lock sul record da leggere, quindi non legge mai dati uncommitted.

Rilascia il lock dopo la lettura e prima di concludere la propria operazione sul DB. Però in un secondo tempo può trovare il record cambiato per la causa 2), mentre non si può verificare la 1).

## LIVELLO 3

Una transazione pone lock su tutti i record che legge e li rilascia solo dopo aver terminato. Non si verificano mai nè 1), nè 2).

# Livelli di isolamento

---

Riassumendo...

Tipi di problemi:

- **Lost update**
- **Dirty read**
- **Unrepeatable read**

Livelli di isolamento:

- **0 (Chaos)** → si presentano tutti i tre problemi
- **1 (Browse)** → non si hanno lost update
- **2 (Cursor stability)**  
→ non si hanno lost update e dirty read
- **3 (Repeatable reads)**  
→ nessun problema (isolation “pura”)



# Problema del deadlock

---

Situazione che si verifica quando due o più transazioni sono in stato di wait (attesa) per attendere il rilascio di oggetti da parte di altre transazioni in stato di wait.

**T1 :**

**W-LOCK(D1)**

**W-LOCK(D2)**

**D1 <- D1 - 100**

**D2 <- D2 + 100**

**UNLOCK(D1)**

**UNLOCK(D2)**

**T2 :**

**W-LOCK(D2)**

**W-LOCK(D1)**

**D1 <- D1 + 30**

**D2 <- D2 - 50**

**UNLOCK(D2)**

**UNLOCK(D1)**

# Insorgenza del deadlock

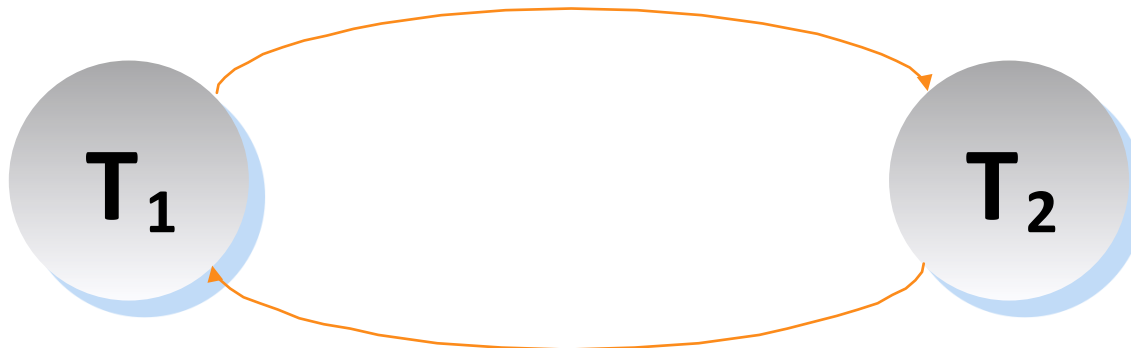
---

**T1 esegue W-LOCK(D1)**

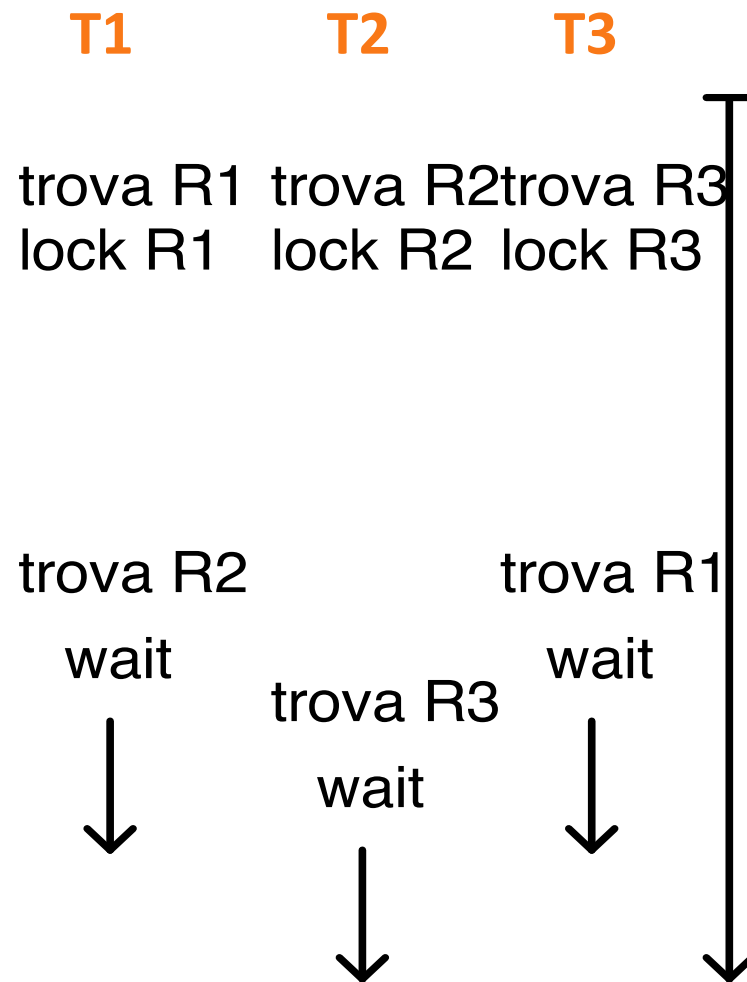
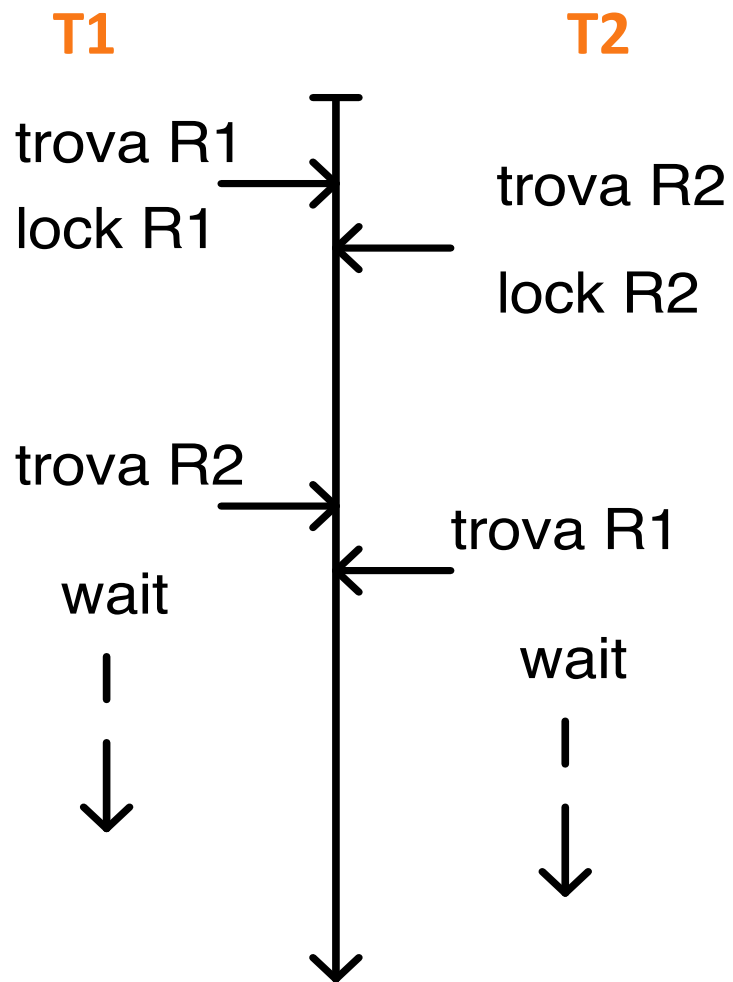
**T2 esegue W-LOCK(D2)**

**T1 attende una risorsa controllata da T2**

**T2 attende una risorsa controllata da T1**



# Esempi di deadlock



## Tecnica del Time-out

---

La tecnica di risoluzione più usata è quella del **TIME-OUT**.

**Quando una transazione entra in stato di wait si attiva un TIME-OUT :**

un'attesa eccessiva è interpretata come deadlock, dopo un certo tempo in attesa (scadenza del timeout) la transazione viene abortita.

La probabilità di avere DEADLOCK è influenzata dalla *granularità* del LOCK (RELAZIONE, PAGINA, TUPLA).