

## Chapter 4

---

# Isolation: Correctness in the read/write model

# Agenda

---

- Concurrent transactions
- Histories and schedules
- Correct histories and schedules

# Concurrent transactions

# Transaction model revisited (1)

## Definition 4.1 (Read/Write Model Transaction):

A **transaction**  $t_i$  is a partial order  $(op_i, <_i)$ , with  $op_i$  all operations in  $t_i$  and

1.  $op_i \subseteq \{r_i(x), w_i(x) \mid x \text{ is data element}\} \cup \{a_i, c_i\}$   
( $op'_i = op_i \setminus \{a_i, c_i\}$ )
2.  $\forall p \in op_i \ p = c_i \vee p = a_i \succ (\forall q \in op'_i \ q <_i p)$
3.  $r_i(x), w_i(x) \in op'_i \succ (r_i(x) <_i w_i(x) \vee w_i(x) <_i r_i(x))$
4.  $a_i \in op_i \succ c_i \notin op_i$  and  $c_i \in op_i \succ a_i \notin op_i$

# Transaction model revisited (2)

## Definition 4.2

- **Sequential transaction:**  $(op_i, <_i)$  is total order
  - ◆ Sufficient for the single-processor case.
- Transaction  $t_i$  is
  - ◆ **aborted**, if  $a_i \in t_i$
  - ◆ **committed**, if  $c_i \in t_i$
  - ◆ **completed**, if aborted or committed.

# Two steps to concurrency

## Step 1:

### Histories and schedules

- Find a formalism for describing the interleaved and concurrent execution of a set of transactions.

Required from the formalism:

- ◆ Include the formal characteristics of transactions.
- ◆ Operations that are in conflict can only be executed in sequence.

## Step 2:

### Serializability

- Ultimately we wish to know how to arrange the operations that are in conflict such that we obtain a correct ordering of the operations.
  - ◆ What definition of correctness?

# Histories and schedules

# Histories

## Definition 4.3 (Histories):

Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **history** for  $T$  is a pair  $h = (op(h), <_h)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  $c_i \in op(h) \Leftrightarrow a_i \notin op(h)$

(c)  $\cup_{i=1..n} <_i \subseteq <_h$

(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op'_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:  
 $p <_h q$  or  $q <_h p$

# Histories

## Definition 4.3 (Histories):

Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **history** for  $T$  is a pair  $h = (op(h), <_h)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  $c_i \in op(h) \Leftrightarrow a_i \notin op(h)$

(c)  $\cup_{i=1..n} <_i \subseteq <_h$

(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op'_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:  
 $p <_h q$  or  $q <_h p$

all operations from all transactions are included

# Histories

## Definition 4.3 (Histories):

Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **history** for  $T$  is a pair  $h = (op(h), <_h)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  $c_i \in op(h) \Leftrightarrow a_i \notin op(h)$

(c)  $\cup_{i=1..n} <_i \subseteq <_h$

(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op'_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:  
 $p <_h q$  or  $q <_h p$

each transaction includes **exactly one** completion operation

# Histories

## Definition 4.3 (Histories):

Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **history** for  $T$  is a pair  $h = (op(h), <_h)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  $c_i \in op(h) \Leftrightarrow a_i \notin op(h)$

(c)  $\cup_{i=1..n} <_i \subseteq <_h$

(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op'_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:

$$p <_h q \text{ or } q <_h p$$

the ordering within each transaction is preserved

# Histories

## Definition 4.3 (Histories):

Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **history** for  $T$  is a pair  $h = (op(h), <_h)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  $c_i \in op(h) \Leftrightarrow a_i \notin op(h)$

(c)  $\cup_{i=1..n} <_i \subseteq <_h$

(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op'_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:  
 $p <_h q$  or  $q <_h p$

the completion operation is the final operation in each transaction

# Histories

## Definition 4.3 (Histories):

Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **history** for  $T$  is a pair  $h = (op(h), <_h)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  $c_i \in op(h) \Leftrightarrow a_i \notin op(h)$

(c)  $\cup_{i=1..n} <_i \subseteq <_h$

(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op'_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:

$$p <_h q \text{ or } q <_h p$$

operations in conflict are strictly ordered

# Histories

## Definition 4.4 (Serial history):

A history  $h$  is **serial** if for any two transactions  $t_i$  and  $t_j$  in  $h$ , where  $i \neq j$ , all operations from  $t_i$  are ordered in  $h$  before all operations from  $t_j$  or vice versa.

## Definition 4.5 (Totally ordered history):

A history  $h$  is **totally ordered** if in  $h = (\text{op}(h), <_h)$ ,  $<_h$  is a total order, i.e., all operations from  $\text{op}(h)$  are ordered in sequence.

# Schedules

## Definition 4.6 (Schedules):

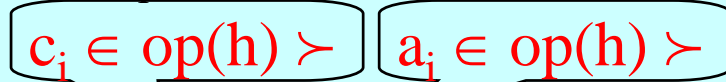
Let  $T = \{t_1, \dots, t_n\}$  be a set of transactions, where each  $t_i \in T$  has the form  $t_i = (op_i, <_i)$  with  $op_i$  denoting the operations of  $t_i$  and  $<_i$  their ordering.

A **schedule** for  $T$  is a pair  $s = (op(s), <_s)$  s.t.

(a)  $op(h) = \cup_{i=1..n} op_i$

(b) for all  $i, 1 \leq i \leq n$ :  ~~$a_i \in op_i \wedge c_i \notin op_i \triangleq c_i \in op_i \wedge a_i \notin op_i$~~   
 ~~$c_i \in op(h) \leftrightarrow a_i \notin op(h)$~~

(c)  $\cup_{i=1..n} <_i \subseteq <_h$



(d) for all  $i, 1 \leq i \leq n$ , and all  $p \in op_i$ :  $p <_h c_i$  or  $p <_h a_i$

(e) for all  $p, q \in op(h) \setminus \cup_{i=1..n} \{a_i, c_i\}$  s.t. at least one of them is a write and both access the same data item:  
 $p <_h q$  or  $q <_h p$

# Schedules and Histories

- A schedule is a prefix of a history.
- A history is a projection of a schedule on completed transactions.
- A *committed projection* of a schedule is a projection on committed transactions.

# History: example

**Example 4.7:** Suppose  $T = \{t_1, t_3, t_4\}$  with

$$t_1 = r_1(x) \rightarrow w_1(x)$$

$$t_3 = r_3(x) \rightarrow w_3(y) \rightarrow w_3(x)$$

$$t_4 = r_4(y) \rightarrow w_4(x) \rightarrow w_4(y) \rightarrow w_4(z)$$

One history  $h_1$  over  $T$  is:

$$r_3(x) \rightarrow w_3(y) \rightarrow w_3(x) \rightarrow c_3$$



$$h_1 = r_4(y) \rightarrow w_4(x) \rightarrow w_4(y) \rightarrow w_4(z) \rightarrow c_4$$

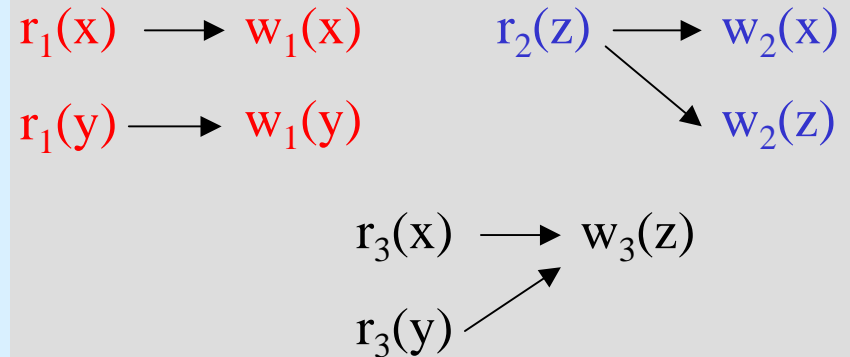


$$r_1(x) \rightarrow w_1(x) \rightarrow c_1$$



# History and schedule: example

## Example 4.8:



## totally ordered history:

h1 =  $r_1(x)$   $r_2(z)$   $r_3(x)$   $w_2(x)$   $w_1(x)$   $r_3(y)$   $r_1(y)$   $w_1(y)$   $w_2(z)$   $w_3(z)$   $c_1$   $c_2$   $a_3$

## totally ordered schedules (history prefixes):

s2 =  $r_1(x)$   $r_2(z)$   $r_3(x)$   $w_2(x)$   $w_1(x)$

s3 =  $r_1(x)$   $r_2(z)$   $r_3(x)$   $w_2(x)$   $w_1(x)$   $r_3(y)$   $r_1(y)$   $w_1(y)$   $w_2(z)$   $w_3(z)$   $c_1$

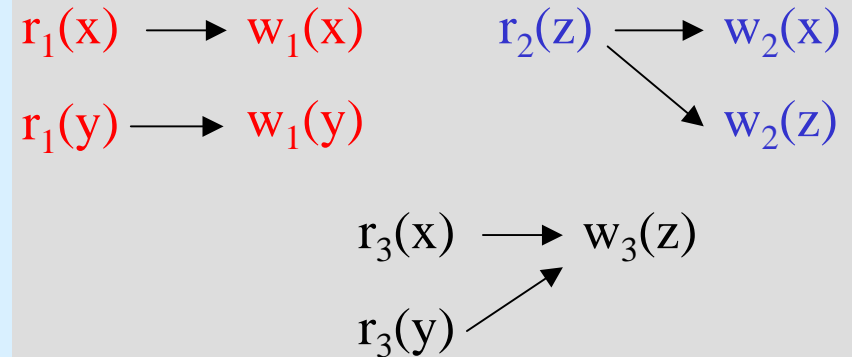
## serial history:

h4 =  $r_1(x)$   $w_1(x)$   $r_1(y)$   $w_1(y)$   $c_1$   $r_3(x)$   $r_3(y)$   $w_3(z)$   $a_3$   $r_2(z)$   $w_2(x)$   $w_2(z)$   $c_2$

# History and schedule: example

20

**Example 4.8:**



**totally ordered schedule:**

$s_5 = r_1(x) w_1(x) r_1(y) w_1(y) c_1 r_3(x) r_3(y) w_3(z) a_3 r_2(z) w_2(x) w_2(z)$

**history as a projected schedule:**

$h_6 = r_1(x) w_1(x) r_1(y) w_1(y) c_1 r_3(x) r_3(y) w_3(z) a_3$

**history as a committed projection:**

$h_7 = r_1(x) w_1(x) r_1(y) w_1(y) c_1$

# Transaction sets of a schedule (1)

## Definition 4. 9 (Transaction sets of a schedule)

- Transactions occurring partially or completely in  $s$ :  
 $trans(s) := \{t_i \mid s \text{ contains operations from } t_i\}$
- Transactions aborted in  $s$ :  
 $abort(s) := \{t_i \mid a_i \in s\}$
- Transactions committed in  $s$ :  
 $commit(s) := \{t_i \mid c_i \in s\}$
- Transactions still active in  $s$ :  
 $active(s) := trans(s) \setminus (commit(s) \cup abort(s))$
- Transactions completed in  $s$ :  
 $complete(s) := commit(s) \cup abort(s)$
- Operations in  $s$ :  
 $op(s) := \{op \mid op \in s\}$

# Transaction sets of a schedule (2)

## Example 4.10

$s1 = r_1(x) r_2(z) r_3(x) w_2(x) w_1(x) r_3(y) r_1(y) w_1(y) w_2(z) w_3(z) c_1 c_2 a_3$

$\text{trans}(s1) = \{t_1, t_2, t_3\}$

$\text{commit}(s1) = \{t_1, t_2\}$

$\text{abort}(s1) = \{t_3\}$

$\text{active}(s1) = \{\}$

# Transaction sets of histories and schedules

## Remarks 4.11:

For each schedule  $s$ :

- $\text{commit}(s) \cap \text{abort}(s) = \text{commit}(s) \cap \text{active}(s)$   
 $= \text{abort}(s) \cap \text{active}(s) = \emptyset$
- $\text{trans}(s) = \text{commit}(s) \cup \text{abort}(s) \cup \text{active}(s)$

For each history  $h$ :

- $\text{trans}(h) = \text{commit}(h) \cup \text{abort}(h)$
- $\text{active}(h) = \emptyset$

# **Correct histories and schedules**

# Correct schedules (1)

## Remember:

- Include the formal characteristics of transactions.
- Operations that are in conflict can only be executed in sequence.

## Definitions sufficient?

- Given a set of transactions  $t_1, \dots, t_n$ , there are many syntactically correct schedules.
- Which of these guarantee global consistency (semantic correctness)?
  - ◆ What do we mean by global consistency?

# Correct schedules (2)

**Find:** Correctness criterion  $\sigma : S \rightarrow \{0, 1\}$  for schedules  $s \in S$  where

- $correct(S) := \{ s \mid \sigma(s) = 1 \} \neq \emptyset,$
- $correct(S)$  large
- $s \in correct(S)$  is efficiently decidable.

To be useful the criterion must admit some schedules.

To be useful the criterion should admit a large number of schedules. (This would give us a choice among alternatives, and we could choose one with a high degree of parallelism.)

Choose the criterion s.t. an algorithm can quickly decide on the correctness of a schedule.

**conflict!**

And: The criterion should make (semantic) sense!

# Correct schedules (3)

**correct(S)  $\neq \emptyset$ :**

- That's easy: In serial schedules transactions are isolated.  $\Rightarrow$  Serial schedules are correct.
- However: Serial execution is extremely inefficient.  $\Rightarrow$  Find schedules that allow for something better than serial execution, but are **in some (which?) sense *equivalent*** to serial schedules. We call such schedules *serializable*.

**correct(S) large:**

**i.e., are semantically correct**

- Our hope: There are more serializable schedules than just serial schedules.
- Find a constructive definition of equivalence with a large set of serializable schedules.

**correct(S) efficiently decidable:**

- The definition should make equivalence efficiently decidable.

# Correct schedules (4)

**Schedules evolve**

**Histories have a known outcome**



**Define correctness on histories!**



**Then ensure somehow that schedules result in correct histories!**

# Final-state equivalence

## Definition 4.12

- **Final-state equivalence**: Two histories are final-state equivalent if they have the same operations and result in the same final state for any given initial state.
- **Final-state serializability**: A history  $h \in H$  is final-state serializable if there exists a final-state equivalent serial history.
  - Semantics?  $\Rightarrow$  Equivalence considers only the final outcome, not the intermediate states and not whether each individual transaction behaves the same in the two histories.
  - $\text{correct}(H)$  large: Presumably, because of the weak requirement: Much of the past can safely be ignored.
  - $\text{correct}(H)$  efficiently decidable: ??????

# Last write in a schedule

## Definition 4.13 (Last write)

Let  $s$  be a schedule and  $x$  a data element. **Last write** of  $x$  in  $s$  is operation  $w_i(x) \in s$  where

- $a_i \notin s$
- $\forall w_j(x) \in s \quad i \neq j \succ w_j(x) <_s w_i(x) \vee a_j \in s$

We write  $w_i(x) = FIN_s(x)$

## Example 4.14

$s_{12} = w_1(x) w_2(x) w_2(y) c_2 w_1(x) c_1 w_3(x) w_3(y) c_3 w_4(x) a_4$

Then  $w_3(x) = FIN_{s_{12}}(x)$

$w_3(y) = FIN_{s_{12}}(y)$

# Final-state equivalence

## Definition 4.15 (final-state equivalence)

Let  $s$  and  $s'$  be schedules.  $s$  and  $s'$  are **final-state equivalent** ( $s \equiv_F s'$ )  $\Leftrightarrow$

- $op(s) = op(s')$
- $\forall x \in D \text{ } FIN_s(x) = FIN_{s'}(x)$  ( $D$  database)

Two schedules – and by extension, two histories – are final-state equivalent iff both include the same operations and result in the same final state.

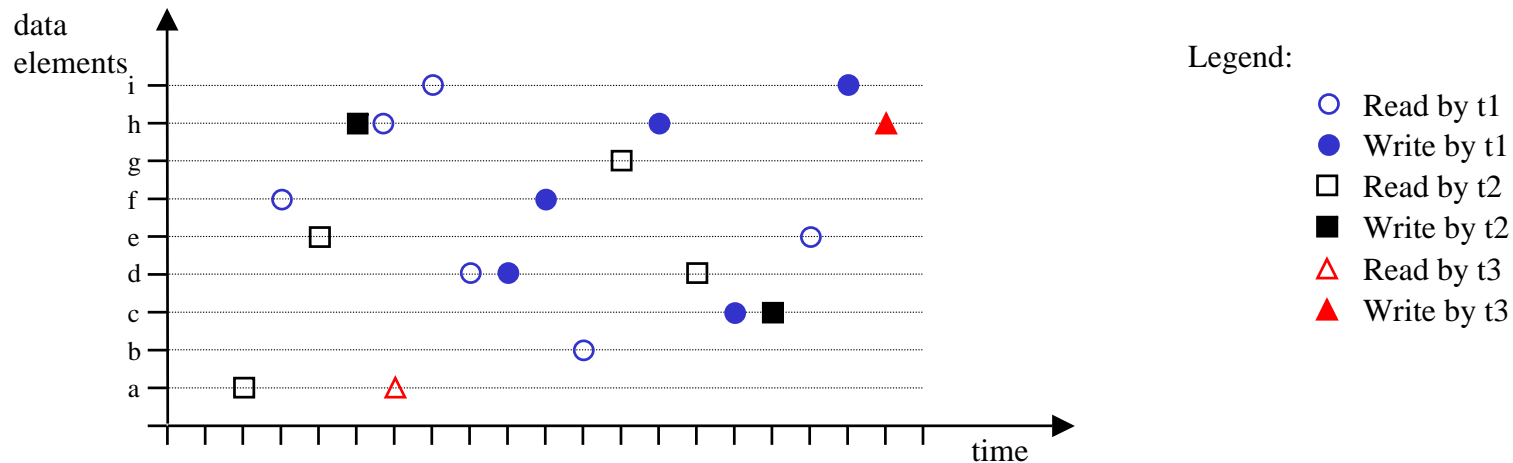
# Conceptual test for serializability

- Existence of a final-state equivalent serial history implies: For each transaction its operations can commute to a single point in time (equivalence time) without changing *FIN*. The order of equivalence times determines the serial order.

# Conceptual test for serializability

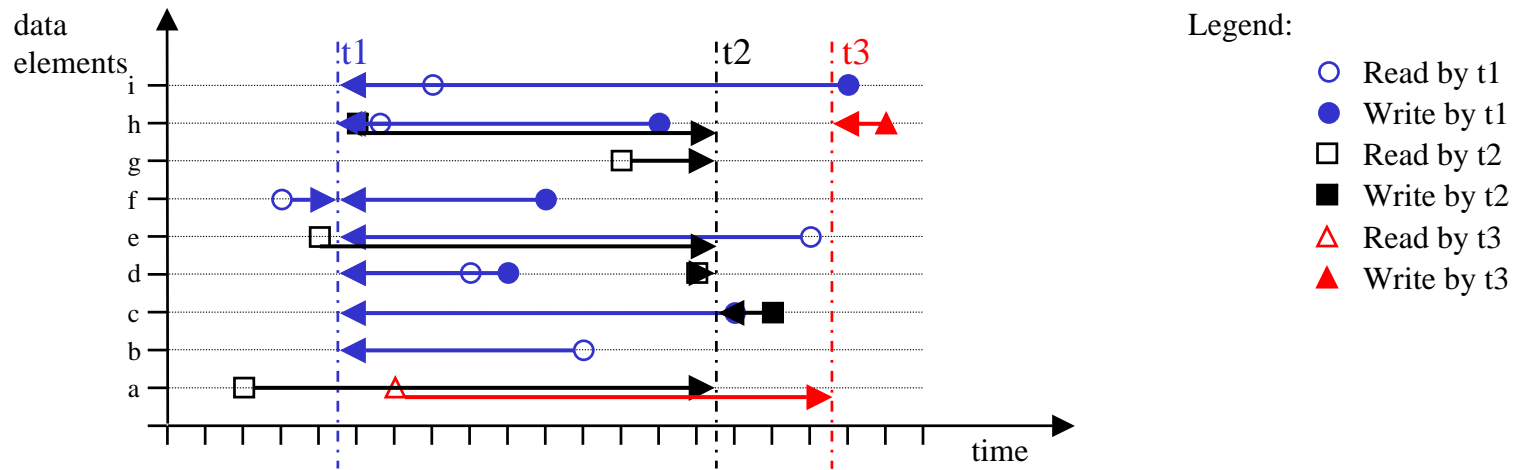
- Take history:

$r_2(a)$   $r_1(f)$   $r_2(e)$   $w_2(h)$   $r_1(h)$   $r_3(a)$   $r_1(i)$   $r_1(d)$   $w_1(d)$   $w_1(f)$   $r_1(b)$   
 $r_2(g)$   $w_1(h)$   $r_2(d)$   $w_1(c)$   $w_2(c)$   $r_1(e)$   $w_1(i)$   $c_1$   $w_3(h)$   $c_2$   $c_3$ .



# Conceptual test for serializability

## ■ Example:



# View equivalence

## Definition 4.16

- **View equivalence:** Two histories are view equivalent if they have the same operations, if each transaction behaves the same in both histories, i.e., its read and write operations have the same effect, and they result in the same final state for any given initial state.
- **View serializability:** A history  $h \in H$  is view serializable if there exists a view equivalent serial history.
  - Natural semantics!
  - $\text{correct}(H)$  large: Because of the stronger requirement smaller than for final-state serializability.
  - $\text{correct}(H)$  efficiently decidable: ??????

# Read and write in schedules (1)

## Definition 4.17 (reads from)

Let  $t_i$  and  $t_j$  be transactions and  $s$  a schedule where  $t_i, t_j \in s$ .  
 $t_j$  **reads x from**  $t_i$  if

- $\exists x w_i(x) <_s r_j(x)$
- $a_i \prec_s r_j(x)$
- $\forall k \neq i, j w_i(x) <_s w_k(x) <_s r_j(x) \Rightarrow a_k \prec_s r_j(x)$

A transaction  $t_j$  **reads from**  $t_i$  if there exists a data element  $x$  s.t.  $t_j$  reads  $x$  from  $t_i$ .

Notation:  $t_j \triangleright_s(x) t_i$  and  $t_j \triangleright_s t_i$ , respectively.

Relation ( $RF$ : „reads from“)

$$RF(s) := \{(t_i, x, t_j) \mid t_j \triangleright_s(x) t_i\}$$

# Read and write in schedules (2)

## Example 4.18

Consider

$$s_7 = w_1(x) w_1(y) r_2(u) w_2(x) r_2(y) w_2(y) c_2 w_1(z) c_1$$

Then:

$$t_2 \triangleright_{s_7(y)} t_1$$

$$t_2 \triangleright_{s_7} t_1$$

# View equivalence

## Definition 4.19 (View equivalence)

Two schedules  $s$  and  $s'$  are **view equivalent** ( $s \equiv_v s'$ ):  $\Leftarrow$

- $op(s) = op(s')$
- $RF(s) = RF(s')$
- $\forall x \text{ FIN}_s(x) = \text{FIN}_{s'}(x)$

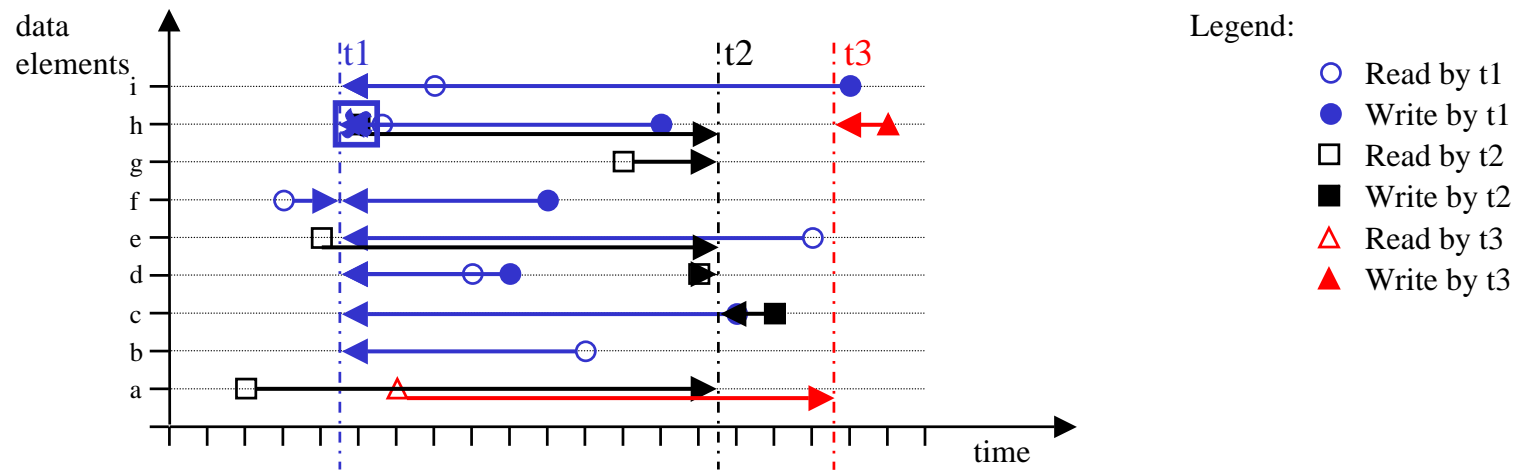
Two schedules – and by extension, two histories – are view equivalent iff both include the same operations, read the same states and result in the same final state.

# Conceptual test for serializability

- Existence of a view equivalent serial history implies: For each transaction its operations can commute to a single point in time (equivalence time) without changing *RF* and *FIN*. The order of equivalence times determines the serial order.

# Conceptual test for serializability

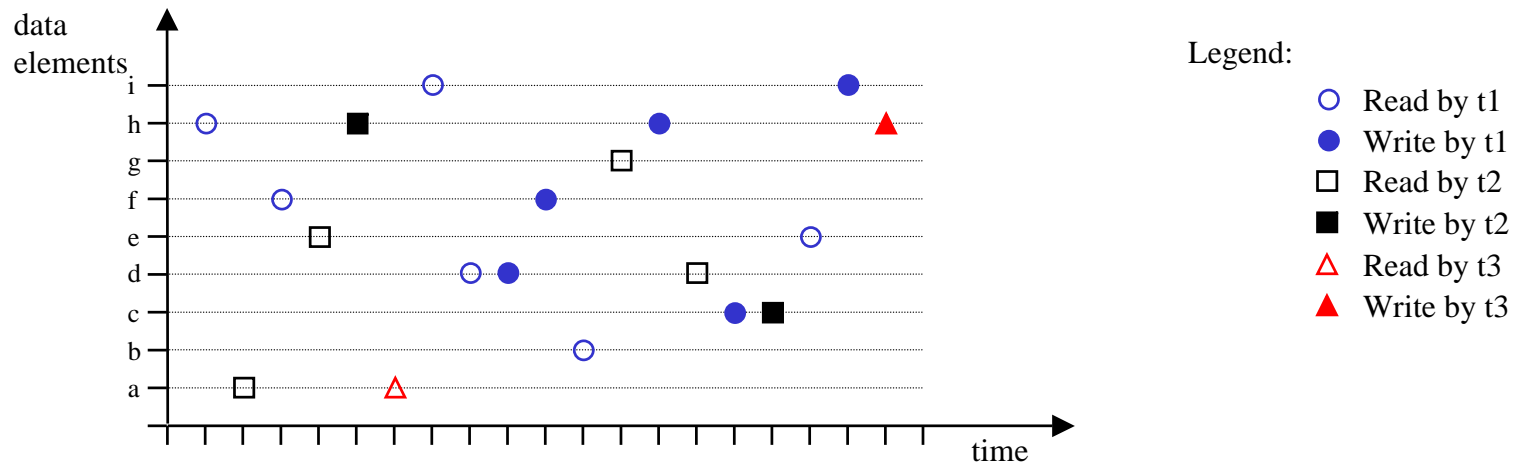
## ■ Example:



# Conceptual test for serializability

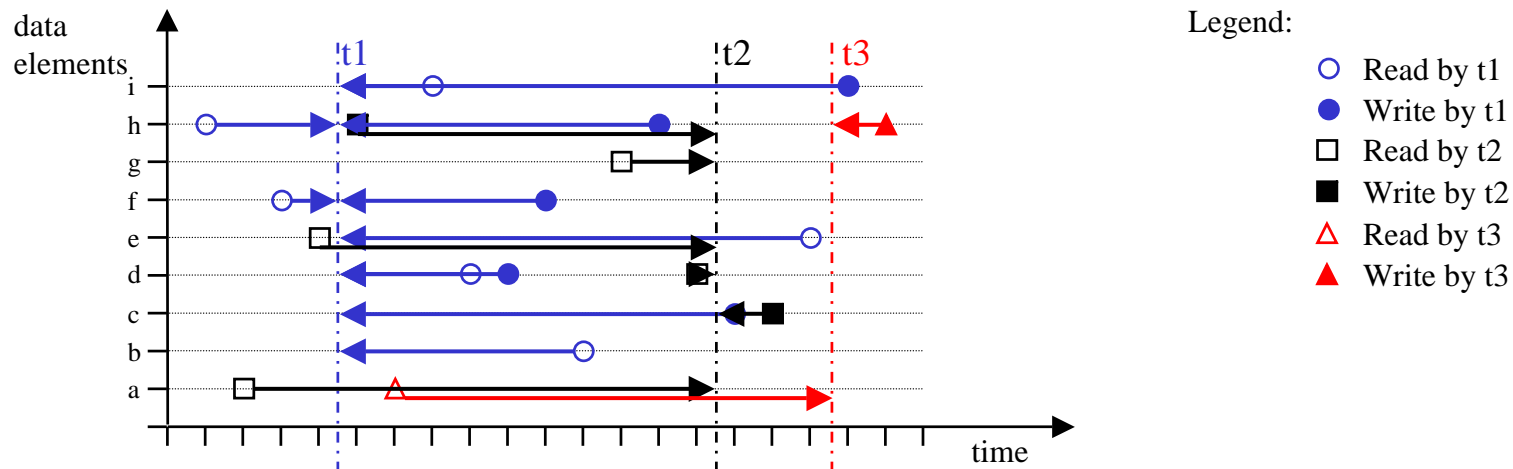
## Take history:

$r_2(a)$   $r_1(f)$   $r_2(e)$   $w_2(h)$   $r_1(h)$   $r_3(a)$   $r_1(i)$   $r_1(d)$   $w_1(d)$   $w_1(f)$   $r_1(b)$   
 $r_2(g)$   $w_1(h)$   $r_2(d)$   $w_1(c)$   $w_2(c)$   $r_1(e)$   $w_1(i)$   $c_1$   $w_3(h)$   $c_2$   $c_3$ .



# Conceptual test for serializability

## ■ Example:



# Conflicts (1)

## Definition 4.20 (in conflict)

Two operations  $p$  and  $q$  in the same or different transactions are **in conflict (do not commute)** if both access the same data element and at least one of them is a write operation.

## Remarks

- Notation:  $p \not\parallel q$ .
- *in conflict* is symmetric.

# Conflicts (2)

time	Miller	Smith	db value	comment
1	read(R)		34	
2	update(R)		34	Miller sells 1 box
3	write(R)		33	34-1
4		read(R)	33	Smith reads the changed value
5		update(R)	33	Smith sells 2 boxes
6	abort		34	Miller cancels his sale
7		write(R)	31	33-2

$w_M(R) \not\parallel r_S(R)$

$w_M(R) \not\parallel w_S(R)$

but

$r_M(R) \parallel r_S(R)$

# Conflict equivalence

## Definition 4.21

- **Conflict equivalence:** Two histories are conflict equivalent if they have the same operations, and operations that are in conflict are ordered the same in both histories.
- **Conflict serializability:** A history  $h \in H$  is conflict serializable if there exists a conflict equivalent serial history  $h'$ :  $\exists h' \text{ serial: } h \equiv_C h'$

# Conflict relation

## Definition 4.22 (conflict relation):

Let  $s$  be a schedule. Then the conflict relation  $conf(s)$  is

$$conf(s) = \{ (p_i, q_j) \mid p_i \not\ll q_j, p_i <_s q_j, a_i, a_j \notin s \}$$

$conf$  describes which operations of the transactions in a schedule are in conflict.

# Conflict equivalence

## Definition 4.23 (Conflict equivalence)

Two schedules  $s$  and  $s'$  are **conflict equivalent** ( $s \equiv_C s'$ )  $\Leftrightarrow$

- $op(s) = op(s')$
- $conf(s) = conf(s')$

Two schedules – and by extension, two histories – are conflict equivalent iff both include the same operations, and identically order their conflicting operations.

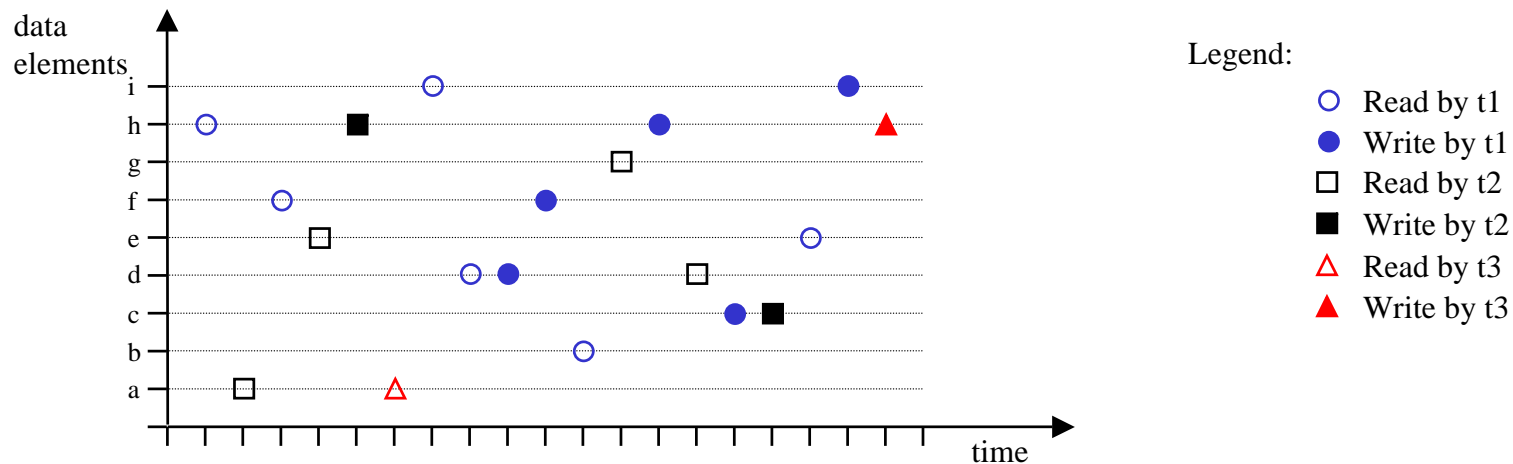
# Conceptual test for serializability

- Existence of a conflict equivalent serial history implies: For each transaction its operations can commute to a single point in time (equivalence time) without exchanging operations that are in conflict. The order of equivalence times determines the serial order.

# Conceptual test for serializability

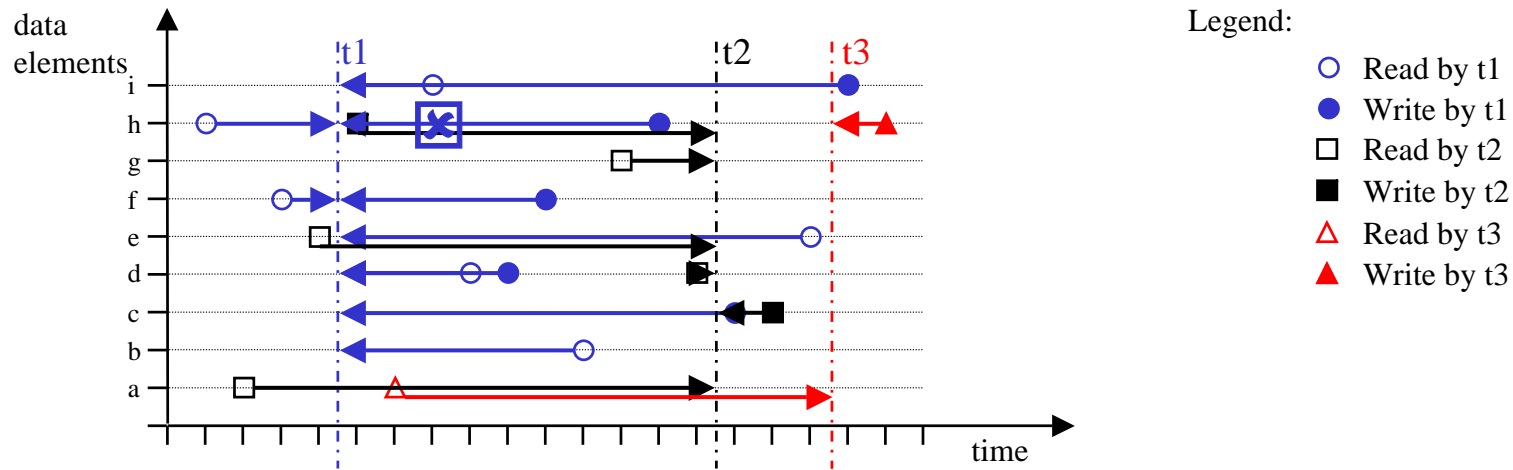
## ■ Take history:

$r_1(h)$   $r_2(a)$   $r_1(f)$   $r_2(e)$   $w_2(h)$   $r_3(a)$   $r_1(i)$   $r_1(d)$   $w_1(d)$   $w_1(f)$   $r_1(b)$   
 $r_2(g)$   $w_1(h)$   $r_2(d)$   $w_1(c)$   $w_2(c)$   $r_1(e)$   $w_1(i)$   $c_1$   $w_3(h)$   $c_2$   $c_3$ .



# Conceptual test for serializability

## ■ Example:



# Serializability relationships

Without proof:

$$\text{correct}_F(H) \supseteq \text{correct}_V(H) \supseteq \text{correct}_C(H)$$

where

- ◆ Final-state serializability:  $F$
- ◆ View serializability:  $V$
- ◆ Conflict serializability:  $C$

# From histories to schedules

- Serializability is defined on histories: Only for completed transactions we know their outcome.
- Moreover, serializability ignores aborted transactions because *FIN* (and *RF*, *conf*) eliminate them from further consideration.
- Serializability can be extended to schedules if we consider only those transactions that have already committed.

## ◆ Definition 4.24 (committed projection $CP(s)$ )

Let  $s$  be a schedule over  $T = \{t_1, \dots, t_n\}$ . Then

$$CP(s) := s|_{t_i \in \text{commit}(s)}$$

$CP(s)$  is a history.

- ◆ A schedule  $s$  is  $X$ -serializable if there exists a serial history  $h_s$  s.t.  $\exists h_s \text{ serial: } CP(s) \equiv_X h_s$