3. Transaction theory

6 Serializability

3. Transaction theory

3. Transaction theory

3. Transaction theory

3. Transaction theory

3. Transaction theory

Serializability

- Introduction to the topic
- Formalization of schedules
- Serializability concepts
- Comparison of serializability concepts

Introduction to Serializability

$$
\begin{aligned}
T_1 \quad : \quad & \mathbf{read}(A);\ A := A - 10;\ \mathbf{write}(A);\ \mathbf{read}(B); \\
& B := B + 10;\ \mathbf{write}(B); \\
T_2 \quad : \quad & \mathbf{read}(B);\ B := B - 20;\ \mathbf{write}(B);\ \mathbf{read}(C); \\
& C := C + 20;\ \mathbf{write}(C);
\end{aligned}
$$

- Execution alternatives for two transactions:
  - Serial, e.g. T_1 before T_2
  - Interleaved, e.g. alternating steps of T_1 and T_2

Interleaved execution: Example

Execution 1		Execution 2		Execution 3	
T_1	T_2	T_1	T_2	T_1	T_2
read(A)		**read**(A)		**read**(A)	
$A - 10$			**read**(B)	$A - 10$	
write(A)		$A - 10$			**read**(B)
read(B)			$B - 20$	**write**(A)	
$B + 10$		**write**(A)			$B - 20$
write(B)			**write**(B)	**read**(B)	
	read(B)	**read**(B)			**write**(B)
	$B - 20$		**read**(C)	$B + 10$	
	write(B)	$B + 10$			**read**(C)
	read(C)		$C + 20$	**write**(B)	
	$C + 20$	**write**(B)			$C + 20$
	write(C)		**write**(C)		**write**(C)

Results of different executions

	A	B	C	$A+B+C$
Initial Value	10	10	10	30
After execution 1	0	0	30	30
After execution 2	0	0	30	30
After execution 3	0	20	30	50

Simplified model

Lock-Unlock-Model

$$T_1 \quad : \quad \texttt{lock}\ A; \quad \texttt{unlock}\ A; \quad \texttt{lock}\ B; \quad \texttt{unlock}\ B;$$
$$T_2 \quad : \quad \texttt{lock}\ B; \quad \texttt{unlock}\ B; \quad \texttt{lock}\ C; \quad \texttt{unlock}\ C;$$

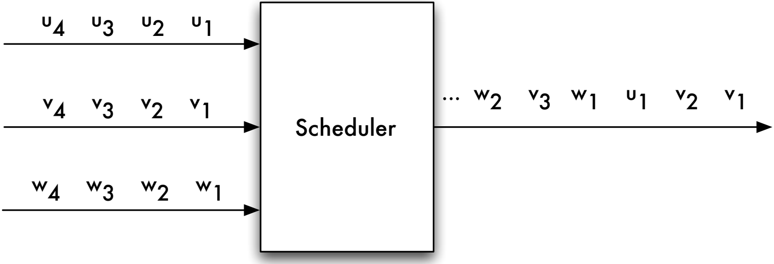
Interleaved Executions with Lock/Unlock: Example

Execution 1		Execution 2		Illegal execution	
T_1	T_2	T_1	T_2	T_1	T_2
lock A		**lock** A		**lock** A	
unlock A			**lock** B		**lock** B
lock B		**unlock** A		**unlock** A	
unlock B			**unlock** B	**lock** B	
	lock B	**lock** B			**unlock** B
	unlock B		**lock** C		**lock** C
	read C	**unlock** B		**unlock** B	
	unlock C		**unlock** C		**unlock** C

Serializability

An interleaved execution of a set of transactions is called **serializable**, if the result of the interleaved execution is identical with the result of a (randomly chosen) serial execution of the same set of transactions transaction.

The concept of schedules



The Read-Write Model

- *Transaction T* is a finite sequence of operations (steps) p_i of the form r(x_k) or w(x_k):

$$T = p_1 p_2 p_3 \cdots p_n \text{ with } p_i \in \{r(x_k), w(x_k)\}$$

- A *complete transaction T* has as last step either an Abort *a* or a Commit *c*:

$$T = p_1 \cdots p_n a$$

or

$$T = p_1 \cdots p_n c.$$

Interleaved transactions

$\text{SHUFFLE}(\boldsymbol{T})$: set of all possible interleaved executions of the steps of all transactions T_i in the set $\boldsymbol{T}$

- All steps of the transaction T_i are included only once
- The relative order of the steps of a transaction is preserved

$$\begin{aligned} T_1 &:= r_1(x)w_1(x) \\ T_2 &:= r_2(x)r_2(y)w_2(y) \end{aligned}$$

$$\begin{aligned} \text{SHUFFLE}(\boldsymbol{T}) = \quad & \{r_1(x)w_1(x)r_2(x)r_2(y)w_2(y), \\ & r_2(x)r_1(x)w_1(x)r_2(y)w_2(y), \\ & \ldots\} \end{aligned}$$

Schedule

A **complete schedule** is a SHUFFLE-Element of a set of complete transactions.

A **schedule** is a prefix of a complete schedule.

$$\underbrace{\underbrace{r_1(x)\, r_2(x)\, w_1(x)}_{\text{a schedule}}\, r_2(y)\, a_1\, w_2(y)\, c_2}_{\text{a complete schedule}}$$

Serial schedule

- A **serial schedule** s for $\boldsymbol{T}$ is a complete schedule of the following form:

$$s := T_{\rho(1)} \cdots T_{\rho(n)} \qquad \text{for a permutation } \rho \text{ of } \{1, \ldots, n\}$$

- Resulting serial schedules for two transactions:
  $T_1 := r_1(x)w_1(x)c_1$ and $T_2 := r_2(x)w_2(x)c_2$:

$$s_1 := \underbrace{r_1(x)w_1(x)c_1}_{T_1}\underbrace{r_2(x)w_2(x)c_2}_{T_2}$$
$$s_2 := \underbrace{r_2(x)w_2(x)c_2}_{T_2}\underbrace{r_1(x)w_1(x)c_1}_{T_1}$$

Correctness criteria

A schedule s is **correct**, if its effect (the result of the execution of the schedule) is equivalent to the effect of a (randomly chosen) serial schedule s' regarding the same set of transactions ($s \approx s'$).

If a schedule s is equivalent to a serial schedule s', then s is **serializable** (to s').

- Open: How to define "being equivalent"?

View serializability

- Idea: Effect of a transaction only depends on what values were seen by the transaction.
- It reads the value that was written last
- Special treatment necessary for the initialization and for the final result of a schedule

View serializability: Preparations

- Artificial additional transactions:
  1. *Initial transaction* T_0: initially writes all involved objects
  2. *Terminal transaction* T_∞: reads all involved objects in the end

Reads-from-relation

- If $\rightarrow_s$ is the relation "temporally before in the schedule s", then "$r_j(x)$ reads x from T_i" if and only if:
  - $w_i(x) \rightarrow_s r_j(x)$ and
  - $\nexists k(w_i(x) \rightarrow_s w_k(x) \land w_k(x) \rightarrow_s r_j(x))$.
- Reads-from-relation $RF(s)$ for a schedule s:

$$RF(s) \;:=\; \{\, (T_i, x, T_j) \mid r_j(x) \text{ reads } x \text{ from } T_i \}$$

View equivalence

- Two schedules s and s' are **view equivalent**, if:
  1. $op(s) = op(s')$
     $op(s)$: the set of all steps occurring in s including a and c (Sets must be identical for both schedules)
  2. $RF(s) = RF(s')$
     Schedules s and s' have the same "Reads-from-relation"

View equivalence: Example I

- Given two complete schedules s_1 and s_2 consisting of two transactions T_1 and T_2 with:

$$\begin{aligned} s_1 &:= r_1(x) r_2(y) w_1(y) w_2(y) c_1 c_2 \\ s_2 &:= r_1(x) w_1(y) r_2(y) w_2(y) c_2 c_1 \end{aligned}$$

View equivalence: Example II

- For calculating the reads-from-relations $RF(s_1)$ and $RF(s_2)$, s_1 and s_2 are expanded by the initial transaction T_0 and the terminal transaction T_∞ in the following way:

$$\begin{aligned} s_1 &:= \underbrace{w_0(x)w_0(y)c_0}_{T_0} r_1(x)r_2(y)w_1(y)w_2(y)c_1c_2 \underbrace{r_\infty(x), r_\infty(y)c_\infty}_{T_\infty} \\ s_2 &:= \underbrace{w_0(x)w_0(y)c_0}_{T_0} r_1(x)w_1(y)r_2(y)w_2(y)c_2c_1 \underbrace{r_\infty(x), r_\infty(y)c_\infty}_{T_\infty} \end{aligned}$$

View equivalence: Example III

- Resulting *reads-from-relations*:

$$
\begin{aligned}
RF(s_1) &:= \{(T_0, x, T_1), (T_0, y, T_2), (T_0, x, T_\infty), (T_2, y, T_\infty)\} \\
RF(s_2) &:= \{(T_0, x, T_1), (T_1, y, T_2), (T_0, x, T_\infty), (T_2, y, T_\infty)\}
\end{aligned}
$$

- *View equivalence* of s_1 and s_2: Comparison of *reads-from-relations*
- Since $RF(s_1) \neq RF(s_2)$, s_1 and s_2 are *not* view equivalent

View Serializability

A schedule s is **view serializable**, only if s is view equivalent to a serial schedule.

- Set of all view serializable schedules: **VSR** (view serializability)
- For n transactions, there exist $n!$ serial schedules
- Problems:
  - Exponential complexity for testing
  - Testing requires complete schedules (blind writes and transaction aborts)

View Serializability: Example

- Obviously, schedule s_2 is view serializable, because s_2 is serial.
- Possible serial schedules s' and s'' for s_1:
  - $s' = T_1 T_2 = r_1(x) w_1(y) c_1 r_2(y) w_2(y) c_2$
    $RF(s') := \{(T_0, x, T_1), (T_1, y, T_2), (T_0, x, T_\infty), (T_2, y, T_\infty)\}$
    $\rightarrow \quad RF(s') \neq RF(s_1)$
  - $s'' = T_2 T_1 = r_2(y) w_2(y) c_2 r_1(x) w_1(y) c_1$
    $RF(s'') := \{(T_0, y, T_2), (T_0, x, T_1), (T_0, x, T_\infty), (T_1, y, T_\infty)\}$
    $\rightarrow \quad RF(s'') \neq RF(s_1)$

$RF(s') \neq RF(s_1)$ and $RF(s'') \neq RF(s_1)$: Schedule s_1 is not serializable!

Conflict Serializability

- Idea: Only the relative order of operations and not the actually read value is of importance in conflict situations
- It is not necessary to know which transaction has most recently written a value, only whether the value has been written before or after a transaction

Conflicts

T_1	T_2
read A	
	read A

order independent

T_1	T_2
read A	
	write A

order dependent

T_1	T_2
	write A
read A	

order dependent

T_1	T_2
	write A
write A	

order dependent

Conflict Serializability

- Conflict relation C of s:

$$C(s) := \{ \ (p, q) \ \mid p, q \text{ are in conflict and } p \rightarrow_s q\}$$

- Conflict matrix:

	$r_i(x)$	$w_i(x)$
$r_j(x)$	$\surd$	$-$
$w_j(x)$	$-$	$-$

Adjusted conflict relation

- conf(s) is an "adjusted" conflict relation, which includes no aborted transactions

  $$\text{conf}(s) := C(s) - \{\, (p, q) \mid (p \in t' \vee q \in t') \wedge t' \in \texttt{\textbf{aborted}}(s)\}$$

- **`aborted`**(s): Set of aborted transactions in schedule s

Adjusted conflict relation: Example

- Given schedule s:

$$s = r_1(x)w_1(x)r_2(x)r_3(y)w_2(y)c_2a_1c_3$$

- Conflict relation for s:

$$C(s) := \{(w_1(x), r_2(x)), (r_3(y), w_2(y))\}$$

- Removing aborted transaction T_1 from s:

$$\text{conf}(s) := \{(r_3(y), w_2(y))\}$$

Conflict equivalence

- Two schedules s and s' are **conflict equivalent** ($s \approx_c s'$) if:
  1. $op(s) = op(s')$
  2. $\mathrm{conf}(s) = \mathrm{conf}(s')$

Conflict serializability

A schedule s is **conflict serializable**, if and only if s is conflict equivalent to a serial schedule.

- Class of all conflict serializable schedules: **CSR** (conflict serializability)

Conflict serializability: Example I

- Given two schedules s and s':

$$s = r_1(x)r_1(y)w_2(x)w_1(y)r_2(z)w_1(x)w_2(y)$$
$$s' = r_1(y)r_1(x)w_1(y)w_2(x)w_1(x)r_2(z)w_2(y)$$

- Question:
  Are schedules s and s' conflict equivalent?
- Step 1:
  $op(s) = op(s')$ applies, since all database operations occurring in s occur in s' as well; also applies vice versa

Conflict serializability: Example II

- Step 2: Adjusted conflict relations

$$\begin{aligned}\text{conf}(s) &= \{(r_1(x), w_2(x)), (w_2(x), w_1(x)), (r_1(y), w_2(y)),\\ &\quad (w_1(y), w_2(y))\}\\ \text{conf}(s') &= \{(r_1(x), w_2(x)), (w_2(x), w_1(x)), (r_1(y), w_2(y)),\\ &\quad (w_1(y), w_2(y))\}\end{aligned}$$

  - $\text{conf}(s) = \text{conf}(s')$ applies; so the conflict relations are equal and therefore s and s' are conflict equivalent

Conflict serializability: Example III

- Test for conflict serializability by comparison with serial schedules
- Given schedule s:

$$s = r_1(x)r_1(y)w_2(x)w_1(y)r_2(z)w_1(x)w_2(y)$$

Conflict serializability: Example IV

- Adjusted conflict relation for s:

$$\begin{aligned}\text{conf}(s) \quad = \quad & \{(r_1(x), w_2(x)), (w_2(x), w_1(x)), (r_1(y), \\ & w_2(y)), (w_1(y), w_2(y))\}\end{aligned}$$

- Possible serial schedule s_1:

$$s_1 = T_1 T_2 = r_1(x) r_1(y) w_1(y) w_1(x) c_1 w_2(x) r_2(z) w_2(y) c_2$$

  - Conflict relation of s_1 is *not* equal to the conflict relation of s:

$$\begin{aligned}\text{conf}(s_1) \quad = \quad & \{(r_1(x), w_2(x)), (w_1(x), w_2(x)), \\ & (r_1(y), w_2(y)), (w_1(y), w_2(y))\}\end{aligned}$$

Conflict serializability: Example V

- Possible serial schedule s_2 as a candidate:

$$s_2 = T_2 T_1 = w_2(x) r_2(z) w_2(y) c_2 r_1(x) r_1(y) w_1(y) w_1(x) c_1$$

  - Conflict relation of s_2 is *not* equal to the conflict relation of s:

$$\begin{aligned} \text{conf}(s_2) \quad = \quad & \{(w_2(x), r_1(x)), (w_2(y), r_1(y)), \\ & (w_2(y), w_1(y)), (w_2(x), w_1(x))\} \end{aligned}$$

- Therefore: $s \notin$ **CSR**, which means that schedule s is not conflict serializable

Conflict serializability: Example VI

- Schedule:

$$s_3 = r_1(x) r_2(x) w_2(y) c_2 w_1(x) c_1$$

is obviously conflict serializable, since only one conflict occurs

Graph-based Test

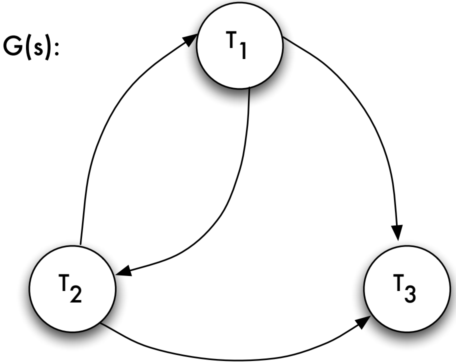
- Conflict graph $G(s) = (V, E)$ for schedule s:
  1. Set of vertices V includes all transactions occurring in s
  2. Set of edges E includes all directed edges between two conflicting transactions:
     $(t, t') \in E \Leftrightarrow t \neq t' \wedge (\exists p \in t)(\exists q \in t')$ with $(p, q) \in \text{conf}(s)$

Chronological sequence of three transactions

T_1	T_2	T_3
r(y)		
		r(u)
	r(y)	
w(y)		
w(x)		
	w(x)	
	w(z)	
		w(x)

$$s = r_1(y)r_3(u)r_2(y)w_1(y)w_1(x)w_2(x)w_2(z)w_3(x)$$

Conflict graph



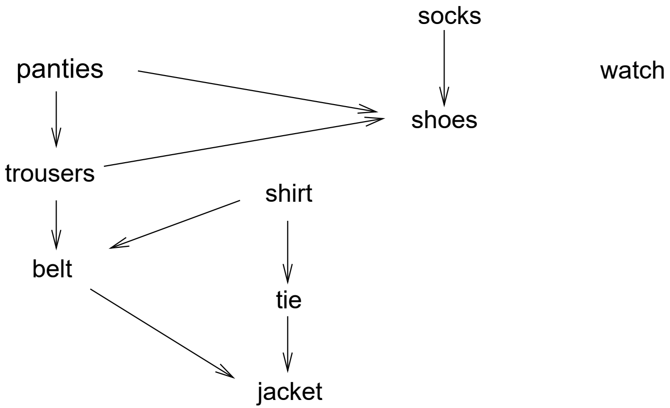
Properties of conflict graph $G(s)$

1. If s is a serial schedule, the given conflict graph is an acyclic graph
2. For every acyclic graph $G(s)$, a serial schedule s' can be constructed in order to make s conflict serializable to s' (Test e.g. by *topological sorting*)
3. If a graph contains cycles, the respective schedule is not conflict serializable

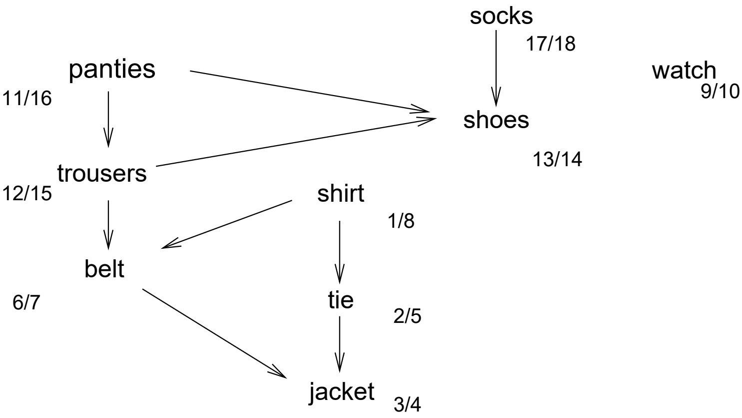
Topological Sorting

- (Recursive) iteration through the graph can detect the absence of cycles
  - Each node is visited once
  - processing status per node and time stamp for entering and leaving is noted
  - repeated reentry on arbitrary unprocessed nodes
- Topological sorting can be done in the same run
  - Times of leaving provides the order
- sorted sequence defines a conflict-equivalent serial schedule
- see example of Saake/Sattler: Algorithmen und Datenstrukturen (the absent-minded professor ...)

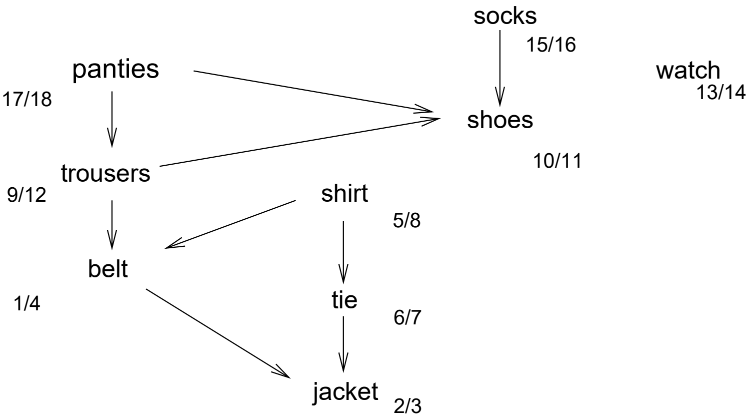
Topological Sorting: Input



Topological Sorting: Result



Topological Sorting: Alternative result



Conflict Graphs and Serializability

For every schedule s applies:

$$G(s) \text{ acyclical} \Leftrightarrow s \in \textbf{CSR}$$

Problems during run-time

- Verification of serializability properties during run-time
  - Only incomplete schedules available during run-time $\rightsquigarrow$ Observation of incomplete schedules is necessary
  - Transactions that have not performed a **`commit`** yet can still be aborted anytime
- How does this affect the accuracy of the verification?

Closure Properties

1. **Prefix Closure**
   If property E applies to a schedule s, E also applies to any prefix of s. If E is fulfilled at the end of a schedule, E must not have been violated before.
2. **Commit Closure**
   If E applies to s, E also applies to $CP(s)$ ("**c**ommitted **p**rojection"). If E applies to a set of transactions, it still applies if some of them are aborted.
3. **Prefix-commit Closure** (PCC)
   Prefix-commit closure is the conjunction.

Closure Properties

- **VSR**-Schedules are **not** *prefix-commit closed*
  - not prefix closed: blind writes can repair
  - not commit closed: last writes can be changed by Abort
- **CSR**-Schedules are *prefix-commit closed*
  - prefix closed: cycles in the graph remain cycles
  - commit closed: Aborts can not generate any cycles

CSR vs VSR: Example I

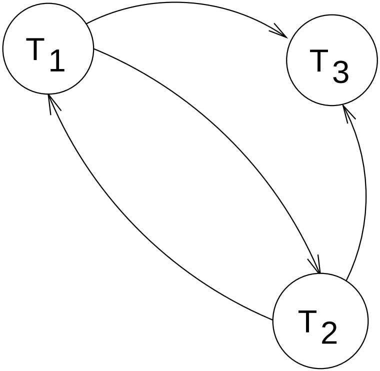
Does **CSR** $\subset$ **VSR** or **VSR** $\subset$ **CSR**?

- Given: schedule s:

$$s = r_1(y) r_3(w) \underbrace{r_2(y) w_1(y)}_{T_2 \rightarrow T_1} \underbrace{w_1(x) w_2(x)}_{T_1 \rightarrow T_2} w_2(z) w_3(x) c_2 c_1 c_3$$

CSR vs VSR: Example II

Schedule *s* is not conflict serializable, since conflict graph $G(s)$ contains a cycle.



CSR vs VSR: Example III

Is *s view serializable*?

- Determining of the reads-from-relation *RF*

$$\begin{aligned} RF(s) &= \{(T_0, y, T_1), (T_0, w, T_3), (T_0, y, T_2), (T_3, x, T_\infty), \\ &\quad (T_1, y, T_\infty), (T_2, z, T_\infty), (T_0, w, T_\infty)\} \end{aligned}$$

- Serial schedule $s' = T_2\ T_1\ T_3$:

$$s' = r_2(y)w_2(x)w_2(z)c_2r_1(y)w_1(y)w_1(x)c_1r_3(w)w_3(x)c_3$$

- Reads-from-relation for schedule s':

$$\begin{aligned} RF(s') &= \{(T_0, y, T_1), (T_0, w, T_3), (T_0, y, T_2), (T_3, x, T_\infty), \\ &\quad (T_1, y, T_\infty), (T_2, z, T_\infty), (T_0, w, T_\infty)\} \end{aligned}$$

CSR vs VSR: Example IV

- $RF(s) = RF(s')$ applies, therefore, schedule s is view serializable
- Therefore: Conflict serializability is more restrictive than view serializability

$$\neg(\mathbf{CSR} \supset \mathbf{VSR})$$

- Generally:

$$\mathbf{CSR} \subset \mathbf{VSR}$$

- Cause: blind writes can repair violations caused by conflicts

Fault Tolerance

In terms of fault tolerance, the following schedule s is not acceptable:

$$s = r_1(x) w_1(x) r_2(x) a_1 w_2(x) c_2$$

. . . though it is serializable in **VSR** *and* **CSR**!

Recoverability RC

- s is *recoverable*, if the following condition is fulfilled:

$$(T_i \text{ reads from } T_j \text{ in } s) \wedge (c_i \in s) \Rightarrow (c_j \rightarrow_s c_i)$$

Recoverability: Example

$$s_1 = w_1(x)w_1(y)r_2(u)w_2(x)r_2(y)w_2(y)c_2w_1(z)c_1$$

In s_1, T_2 reads data object y from T_1 but c_2 comes before c_1 $\rightsquigarrow$ s_1 is not recoverable

$$s_2 = w_1(x)w_1(y)r_2(u)w_2(x)r_2(y)w_2(y)w_1(z)c_1c_2$$

s_2 is recoverable

But: Problems when aborting T_1 instead of c_1 (dirty read)!

Avoiding cascading aborts

- Schedule *s avoids cascading aborts* **ACA**, if the following condition is fulfilled:

$$(T_i \text{ reads } x \text{ from } T_j \text{ in } s) \Rightarrow (c_j \rightarrow_s r_i(x))$$

$\rightsquigarrow$ A transaction may only read data that has last been written by an already committed transaction.

Avoiding cascading aborts: Example

- Schedule s_2 from the last example does not belong into the class **ACA**
- However, s_3 avoids cascading aborts:

$$s_3 = w_1(x)w_1(y)r_2(u)w_2(x)w_1(z)c_1r_2(y)w_2(y)c_2$$

- Therefore: $s_3 \in$ **ACA**

Problems with Before-Images

DB Content	Operation
$x = 1$ (initial value)	
$x = 2$	$w_1(\mathrm{x} \leftarrow 2)$ [$BF_{x,T_1} = 1$]
$x = 3$	$w_2(\mathrm{x} \leftarrow 3)$ [$BF_{x,T_2} = 2$]
	a_1 rollback of $w_1(x \leftarrow 2)$ with $BF_x := 1$.
	Overwriting of T_2 has to be maintained
$x = 3$	
	a_2 rollback of $w_2(x \leftarrow 3)$ with $BF_x := ??$

Strictness ST

- Schedule s is **strict**, if the following applies:

$$(w_j(x) \rightarrow_s p_i(x) \quad \wedge \quad j \neq i) \Rightarrow$$
$$(a_j \rightarrow_s p_i(x) \vee c_j \rightarrow_s p_i(x), (p \in \{r, w\}))$$

$\leadsto$ No "written" object of an incomplete transaction may be read or overwritten

Strictness: Example

- $s_3 \notin \mathbf{ST}$
- s_4 is strict, therefore $s_4 \in \mathbf{ST}$:

$$s_4 = w_1(x) w_1(y) r_2(u) w_1(z) c_1 w_2(x) r_2(y) w_2(y) c_2$$

Relation between the concepts