

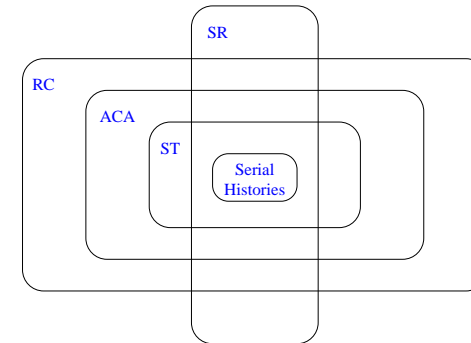
# CA420 Databases II

Basic Concepts – Recoverability

## Classes of Execution and Overall Approach

Approach:

1. define a formal model of transactions and transaction executions
2. define various classes of *correct* executions
3. study algorithms that allow these and only these “correct” executions



## Transaction Operations

Transaction operations:

**Begin:** begins a new transaction; all database operations take place within the context of a transaction; often implicit

**Commit:** the transaction is completed and the DBMS attempts to make all its effects permanent; may fail

**Abort:** the transaction is completed and all of its effects are removed from the database; must succeed

But what are a transaction’s “effects”?

## Notation 1 – Transactions, Data Items and Operations

Transaction identifiers:

- $T_i$  denotes a transaction, transaction  $i$  in this case

Data items:

- $x, y, z$ , etc.

Operations:

- $r_i[x], w_i[x], c_i, a_i$ , operations of  $T_i$

Simplification:

- no transaction reads or writes the same item more than once

## Notation 2 – Transactions

A transaction  $T_i$  consists of:

- a set of operations  $\Sigma_i$
- a partial order  $<_i$  over  $\Sigma_i$

Example  $T_1$ :

- $\Sigma_1 = \{r_1[x], w_1[y], c_1\}$
- $<_1 = r_1[x] <_1 w_1[y], w_1[y] <_1 c_1$

Aside:

A *partial order* is a binary relation (the ordering) that is reflexive, antisymmetric and transitive

Partial orders are always acyclic (by transitivity and antisymmetry)

## But this definition is too weak. . .

Nonsense example:

$$r_1[x] \rightarrow c_1 \rightarrow w_1[y] \rightarrow a_1$$

Some orderings simply don't make sense

## Notation 3 – Wellformed Transactions

Throughout, we only consider wellformed transactions

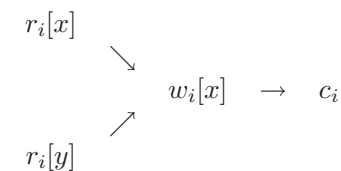
A transaction  $T_i = (\Sigma_i, <_i)$  is wellformed if:

1.  $a_i \in \Sigma_i$  iff  $c_i \notin \Sigma_i$
2. if  $t$  is  $c_i$  or  $a_i$  (whichever is in  $\Sigma_i$ ), then for all other operations  $p \in \Sigma_i$ ,  $p <_i t$
3. if  $r_i[x], w_i[x] \in \Sigma_i$ , then either  $r_i[x] <_i w_i[x]$  or  $w_i[x] <_i r_i[x]$

Or, in English:

1. every transaction either commits or aborts, but not both
2. all other operations are ordered *before* the commit or abort (whichever is present)
3. if there is an item  $x$  that the transaction both reads and writes, then those read and write operations are ordered

## Example Transaction



In this case, the transaction:

- issues  $r_i[x]$  before  $w_i[x]$
- issues  $r_i[y]$  before  $w_i[x]$
- issues  $w_i[x]$  before  $c_i$
- but  $r_i[x]$  and  $r_i[y]$  may be executed in either order

## Scheduling and Executions (or Schedules, Histories) – 1

A database application enforces its *transaction ordering* through the sequence in which it submits operations to the scheduler

- often this happens naturally as a result of the application's control flow

## An Example Execution

A simple example involving two transactions:

$$w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_1 \rightarrow c_2$$

In this case, there is no concurrency between operations:

- the scheduler submits  $r_2[x]$  only after  $w_1[x]$  is acknowledged
- the scheduler submits  $w_2[y]$  only after  $r_2[x]$  is acknowledged
- . . .

## Scheduling and Executions (or Schedules, Histories) – 2

A scheduler enforces an *overall ordering* through the sequence in which it schedules operations on behalf of multiple transactions

When a scheduler submits  $o_1$  and  $o_2$  to the data manager:

- if  $o_2$  is submitted only after  $o_1$  has been acknowledged, then  $o_1$  executes strictly before  $o_2$
- if  $o_1$  and  $o_2$  are submitted concurrently, then they may be executed in either order

This overall ordering is known as an *execution* (or a *schedule*, or a *history*)

A scheduler ensures correctness by delaying (or rejecting) operations to control the execution order

## Recoverability – Example 1

What's *wrong* with the following execution?

$$w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow a_1 \rightarrow c_2$$

## Recoverability – Example 2

And what's wrong with this execution?

$$w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_2$$

## Something's Not Right?

“Consistency” from ACID:

- in isolation, transactions transform the stable database from one logically-consistent state to another

but here  $T_2$  is reading an intermediate state from  $T_1$

“Isolation” from ACID:

- transactions appear to execute in isolation

but here  $T_2$  is not isolated from  $T_1$

Next, we define the class of *recoverable* executions in which such situations cannot arise

## Recoverability – 1

Intuition:

*a transaction cannot commit until all transactions that it read from are guaranteed not to abort*

Reads-from:

- a transaction  $T_j$  *reads from* transaction  $T_i$  if:
  1.  $T_j$  reads  $x$  after  $T_i$  has written it
  2.  $T_i$  does not abort before  $T_j$  reads  $x$ , and
  3. every  $T_k$  that writes  $x$  between  $T_i$ 's write and  $T_j$ 's read aborts before  $T_j$  reads  $x$
- $T_j$  *reads from*  $T_i$  if  $T_j$  reads some data item from  $T_i$

Example:

- $w_i[x] \rightarrow w_k[x] \rightarrow a_k \rightarrow r_j[x]$   
 $T_j$  reads  $x$  from  $T_i$ , hence  $T_j$  reads from  $T_i$

## Reads-From – Examples

Does  $T_2$  read  $x$  from  $T_1$ ?

1.  $w_1[x] \rightarrow r_3[x] \rightarrow a_3 \rightarrow r_2[x]$
2.  $w_1[x] \rightarrow r_3[x] \rightarrow a_1 \rightarrow r_2[x]$
3.  $w_1[x] \rightarrow w_3[x] \rightarrow c_1 \rightarrow r_2[x]$
4.  $w_1[x] \rightarrow w_3[x] \rightarrow a_1 \rightarrow r_2[x]$
5.  $w_1[x] \rightarrow w_3[x] \rightarrow c_3 \rightarrow r_2[x]$
6.  $w_1[x] \rightarrow w_3[x] \rightarrow w_4[x] \rightarrow r_2[x]$

## Recoverability – 2

An execution is recoverable (RC) iff:

- for every transaction  $T$  that commits,  $T$ 's commit follows the commit of every transaction from which  $T$  read

Non-recoverable example:

- $w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_2$

What if  $T_1$  now aborts?

- leave  $T_2$  alone:
  - violates semantics of  $r_2[x]$  operation
- abort  $T_2$ :
  - violates semantics of  $c_2$

Recoverability ensures that an aborting transaction does not violate the semantics of committed transactions' operations

## Scheduling and Recoverability – 1

Generally, recoverability requires a scheduler to delay commit operations until the commits of all transactions from which the committing transaction read have been successfully processed

Assume  $T_2$  reads from  $T_1$

When the scheduler receives  $c_2$ :

- if  $T_1$  has aborted, reject  $c_2$
- if  $T_1$  has successfully committed, schedule  $c_2$  immediately
- otherwise, delay  $c_2$  until:
  - $c_1$  has been successfully processed:
    - \* at which point  $c_2$  can be scheduled, or
  - it is apparent that  $T_1$  cannot commit:
    - \* at which point  $c_2$  must be rejected

## Recoverability – Examples

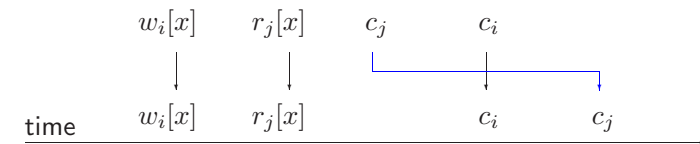
Which of the following are recoverable?

1.  $w_1[x] \rightarrow r_3[x] \rightarrow a_3 \rightarrow r_2[x] \rightarrow c_2$
2.  $w_1[x] \rightarrow r_3[x] \rightarrow a_1 \rightarrow r_2[x] \rightarrow c_2$
3.  $w_1[x] \rightarrow w_3[x] \rightarrow c_1 \rightarrow r_2[x] \rightarrow c_2$
4.  $w_1[x] \rightarrow w_3[x] \rightarrow a_1 \rightarrow r_2[x] \rightarrow c_2$
5.  $w_1[x] \rightarrow w_3[x] \rightarrow c_3 \rightarrow r_2[x] \rightarrow c_2$
6.  $w_1[x] \rightarrow w_3[x] \rightarrow w_4[x] \rightarrow c_1 \rightarrow r_2[x] \rightarrow c_2$
7.  $w_1[x] \rightarrow w_3[x] \rightarrow w_4[x] \rightarrow a_1 \rightarrow r_2[x] \rightarrow a_2$

## Scheduling and Recoverability – 2

Scheduler receives:

$w_i[x]$  then  $r_j[x]$  then  $c_j$  then  $c_i$

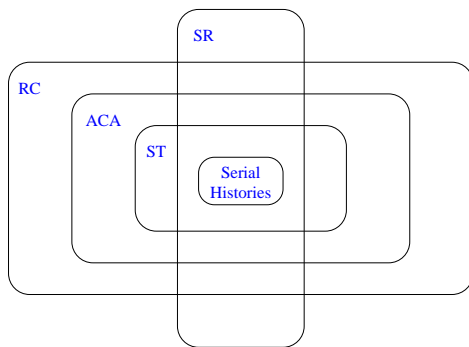


Scheduler delays  $c_j$  until after  $c_i$  has been successfully processed

Execution:

$w_i[x] \rightarrow r_j[x] \rightarrow c_j \rightarrow c_i$

## Classes of Execution



## Or what about this one?

$$w_1[x] \rightarrow r_2[x] \rightarrow w_2[x] \rightarrow r_3[x] \rightarrow w_3[x] \rightarrow r_4[x] \rightarrow w_4[x] \rightarrow a_1$$

## Another Problem Execution

What's wrong with the following execution?

$$w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow a_1$$

Is it recoverable?

## Cascading Aborts

Aborting one transaction may force an arbitrarily long chain of dependant transactions to be aborted

This situation is call *cascading aborts*

Cascading aborts are undesirable:

- require substantial bookkeeping
- lead to unpredictable behaviour from DBS

Schedulers and cascading aborts:

- do not schedule  $r[x]$  operations until all other transactions that previously wrote  $x$  have either committed or aborted
- schedulers that conform to this rule are said to *avoid cascading aborts* (ACA)

## ACA Examples

Example *RC but not ACA* execution:

- $w_1[x] \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_1 \rightarrow c_2$

Example *RC and ACA* execution:

- $w_1[x] \rightarrow c_1 \rightarrow r_2[x] \rightarrow w_2[y] \rightarrow c_2$

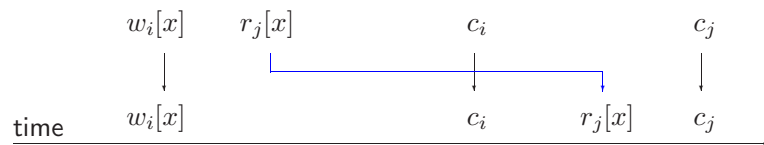
## ACA and RC

It is easy to see that all ACA executions are also RC:

- that is:  $ACA \subset RC$

The converse is not true

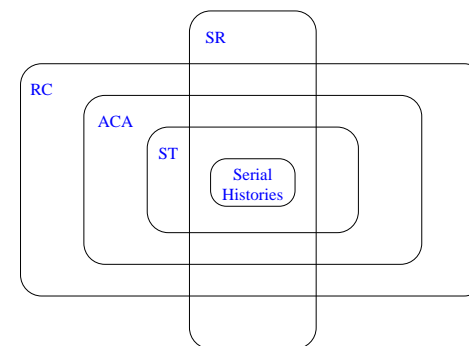
## Schedulers and ACA



When scheduler receives  $r_j[x]$ :

- if any transactions that previously wrote  $x$  are active, then delay  $r_j[x]$  until all such transactions have completed at which point submit  $r_j[x]$
- otherwise, submit  $r_j[x]$  immediately

## Classes of Execution



## Using Before Images for Recovery

Many database systems use before images of writes to implement abort and restart processing:

$$w_1[x] \rightarrow w_1[y] \rightarrow a_1$$

normal processing:

- before  $w_1[x]$ , take copy of current value of  $x$  (the *before image* of  $w_1[x]$ )
- before  $w_1[y]$ , take copy of current value of  $y$  (the *before image* of  $w_1[y]$ )

abort processing:

- install before image of  $w_1[y]$  into  $y$
- install before image of  $w_1[x]$  into  $x$

Before images might also be stored on a log to recover from system failure

## Unfortunately, reinstalling before images is not always correct

Example:

- assume that initially  $x = 1$
- $w_1[x, 2] \rightarrow w_2[x, 3] \rightarrow a_1$
- before image for  $w_1[x, 2]$  is 1
- *but reinstalling this would be incorrect*
- it is actually correct to do nothing in this case

Extending the same example:

- $w_1[x, 2] \rightarrow w_2[x, 3] \rightarrow a_1 \rightarrow a_2$
- *but reinstalling 2, the before image of  $w_2[x, 3]$ , would also be incorrect*
- the correct value to install to here is actually 1

Note:

these examples are both RC and ACA

## Before Images – Example

Consider:

- $w_1[x, 1] \rightarrow w_1[y, 3] \rightarrow w_2[y, 1] \rightarrow c_1 \rightarrow r_2[x] \rightarrow a_2$

The overall effect should be:

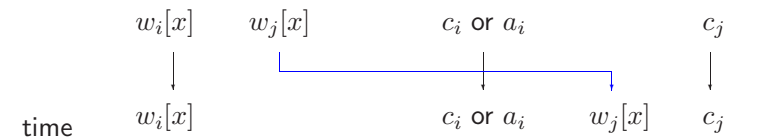
- $w_1[x, 1] \rightarrow w_1[y, 3] \rightarrow c_1$

Aborting  $T_2$  requires that 3, the *before image* of  $w_2[y, 1]$ , be reinstalled to remove the effects of  $T_2$

## Strict Executions (ST)

*Strict executions:*

- delay  $r[x]$  operations until all other transactions that previously wrote  $x$  have completed (i.e. ACA), and
- *delay  $w[x]$  operations all until other transactions that previously wrote  $x$  have completed*



For strict executions, abort and restart processing can always be implemented by reinstalling before images

## ST and ACA

It is easy to see that all ST executions are also ACA (hence also RC):

- that is:  $ST \subset ACA$

The converse is not true

## Recoverability Issues – Summary

Recoverability (RC):

- a correctness consideration with regard to the semantics of the commit operation

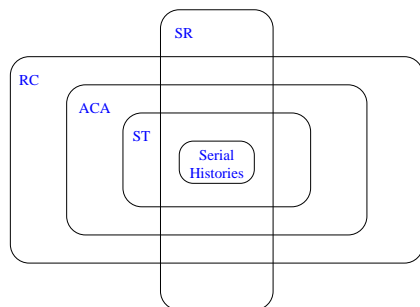
Avoiding cascading aborts (ACA):

- a practical consideration with regard to the implications of admitting cascading aborts

Strictness (ST):

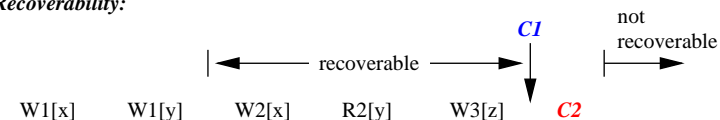
- an implementation consideration with regard to the implementation of abort and restart processing using before images

## Classes of Execution

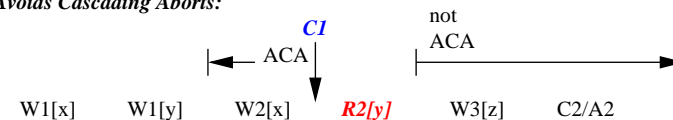


## RC, ACA, ST Illustration

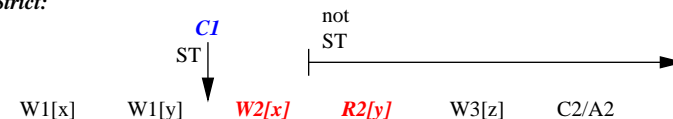
**Recoverability:**



**Avoids Cascading Aborts:**



**Strict:**



## ST, ACA, RC Examples

Which, if any, of the following are *recoverable*?

1.  $r_1[x] \rightarrow w_1[x] \rightarrow r_2[x] \rightarrow w_2[x] \rightarrow a_2 \rightarrow r_3[x] \rightarrow w_3[x] \rightarrow c_3 \rightarrow c_1$
2.  $r_1[x] \rightarrow w_1[x] \rightarrow r_2[x] \rightarrow w_2[x] \rightarrow c_2 \rightarrow r_3[x] \rightarrow w_3[x] \rightarrow c_3 \rightarrow c_1$

Which, if any, of the following *avoid cascading aborts*?

1.  $r_1[x] \rightarrow w_1[y] \rightarrow r_2[x] \rightarrow w_2[z] \rightarrow r_3[x] \rightarrow w_3[z] \rightarrow c_1 \rightarrow c_3 \rightarrow c_2$
2.  $r_1[x] \rightarrow w_1[y] \rightarrow r_2[x] \rightarrow w_2[z] \rightarrow r_3[y] \rightarrow w_3[z] \rightarrow c_1 \rightarrow c_2 \rightarrow c_3$

Which, if any, of the following are *strict*?

1.  $r_1[x] \rightarrow w_1[y] \rightarrow r_2[x] \rightarrow w_2[z] \rightarrow r_3[y] \rightarrow w_3[x] \rightarrow c_1 \rightarrow c_2 \rightarrow c_3$
  2.  $r_1[x] \rightarrow w_1[y] \rightarrow r_2[x] \rightarrow w_2[z] \rightarrow r_3[x] \rightarrow w_3[x] \rightarrow c_1 \rightarrow c_2 \rightarrow c_3$
  3.  $r_1[x] \rightarrow w_1[y] \rightarrow r_2[x] \rightarrow w_2[z] \rightarrow r_3[x] \rightarrow w_3[y] \rightarrow c_1 \rightarrow c_2 \rightarrow c_3$
-