XML Concurrency Control
XML as Database

• Multiple Users
  – 1~1000, or more?
  – Querying and updating occur simultaneously

• Transaction Management
  – Atomicity of query and update operations
    • All-or-nothing execution
  – Consistency and Concurrency Control
    • Locking system
What is XUpdate

• XUpdate is a language for updating XML documents
  – It is defined within the XML:DB project by Andreas Laux and Lars Martin in year 2000
• The motif for introducing XUpdate was the lack of any update capability within XQuery XML query language
• An update in the XUpdate language is expressed as a well-formed XML document
• XUpdate documents use XPath expressions to precisely select the part of an XML document that is going to be updated
XUpdate

- New Expressions
  - Insert, delete, replace, rename, and transform
- Updated all Expressions to allow updates to model
- FLOWR and Typeswitch
  - Add a "do" clause
    - Goes in the place of a return
    - Is the only clause allowed in the expression construct to have updating expressions
- Update Primitives
  - insertBefore, insertAfter, insertInto, insertIntoAsLast, insertAttributes, delete, replaceValue, rename
<?xml version = "1.0" ?>
<xupdate:modifications version="1.0" xmlns:xupdate="http://www.xmldb.org/xupdate">
<!- - Here comes at least one of the following xupdate elements: 
  - xupdate:insert-after
  - xupdate:insert-before
  - xupdate:append
  - xupdate:remove
  - xupdate:variable
  - xupdate:value-of
-- >
</ xupdate:modifications>
Example XML Document

```xml
<?xml version="1.0" standalone="yes"?>
<CLASS name="COMP5323">
    <STUDENT stdno="100" program="ST">
        <NAME>Dang</NAME>
        <ASSIGNMENT no="1">50</ASSIGNMENT>
    </STUDENT>
</CLASS>
```
Inserts

• The XUpdate instructions
  – xupdate:insert-before and
  – xupdate:insert-after
insert a new node either before, or after the context node selected by the select attribute

• The new node can be
  – An element,
  – An attribute,
  – A text,
  – A comment, and
  – A processing instruction
Inserting a New Student Element

<xupdate:insert-after
    select="doc('/db/pavle/class.xml')/CLASS/STUDENT[NAME='Dang']">
    <xupdate:element name="STUDENT">
        <xupdate:attribute name="stdno">
            150
        </xupdate:attribute>
        <xupdate:attribute name="program">
            IT
        </xupdate:attribute>
        <NAME>Bill</NAME>
        <ASSIGNMENT no="1">29</ASSIGNMENT>
    </xupdate:element>
</xupdate:insert-after>
After Insert

<?xml version="1.0" encoding="UTF-8"?>
<!-- The class.xml document after inserting a complete new student element-->
<CLASS name="COMP5323">
  <STUDENT stdno="100" program="ST">
    <NAME>Dang</NAME>
    <ASSIGNMENT no="1">50</ASSIGNMENT>
  </STUDENT>
  <STUDENT stdno="150" program="IT">
    <NAME>Bill</NAME>
    <ASSIGNMENT no="1">29</ASSIGNMENT>
  </STUDENT>
</CLASS>
Append

- The `xupdate:append` element appends a child of the context (element) node selected by the `select` attribute.
- There is also an optional child attribute whose value specifies the position of the new child node:
  - The child expression has to evaluate to an integer value.
  - If not specified, the default is `last`.
- An `xupdate:append` may contain at least one of the following elements:
  - `xupdate:element`
  - `xupdate:attribute`
  - `xupdate:text`
  - `xupdate:comment`
  - `xupdate:processing-instruction`
Appending an ASSIGNMENT element

<! - - append a new ASSIGNMENT element without attributes as the last child of the student Ahmed element - - >

<xupdate:append select=
"doc('/db/pavle/class.xml')/CLASS/STUDENT[NAME='Dang']"
child="last()">
  <xupdate:element name="ASSIGNMENT">
    38
  </xupdate:element>
</xupdate:append>
After Append

<?xml version="1.0" encoding="UTF-8"?>
<!-- The class.xml document after appending a partial assignment element-->
<CLASS name="COMP5323">
  <STUDENT stdno="100" program="ST ">
    <NAME>Dang</NAME>
    <ASSIGNMENT no="1">50</ASSIGNMENT>
    <ASSIGNMENT>38</ASSIGNMENT>
  </STUDENT>
  <STUDENT stdno="150" program="IT ">
    <NAME>Bill</NAME>
    <ASSIGNMENT no="1">29</ASSIGNMENT>
  </STUDENT>
</CLASS>
Appending an Attribute

<! - - This update appends an attribute with the name no to the second ASSIGNMENT element of the student Ahmed - ->

<xupdate:append select="doc('~/db/pavle/class.xml')/CLASS/STUDENT[NAME='Dang']/ASSIGNMENT[2]">
    <xupdate:attribute name="no">2</xupdate:attribute>
</xupdate:append>
After Appending no="2"

<?xml version="1.0" encoding="UTF-8"?>
<!-- The class.xml document after appending a no="2" attribute to the ASSIGNMENT element-->
<CLASS name="COMP5323">
  <STUDENT stdno="100" program="ST ">
    <NAME>Dang</NAME>
    <ASSIGNMENT no="1">50</ASSIGNMENT>
    <ASSIGNMENT no="2">38</ASSIGNMENT>
  </STUDENT>
  <STUDENT stdno="150" program="IT ">
    <NAME>Bill</NAME>
    <ASSIGNMENT no="1">29</ASSIGNMENT>
  </STUDENT>
</CLASS>
Updating Values

<! - - Change @stdno from 100 to 120 for student Ahmed - - >

<xupdate:update select="doc('/db/pavle/class.xml')/CLASS/STUDENT[NAME='Dang']/@stdno">
  120
</xupdate:update>
After Updating

<?xml version="1.0" encoding="UTF-8"?>
<!-- The class.xml document after updating stdno attribute to 120 -->
<CLASS name="COMP5323">
  <STUDENT stdno="120" program="ST ">
    <NAME>Dang</NAME>
    <ASSIGNMENT no="1">50</ASSIGNMENT>
    <ASSIGNMENT no="2">38</ASSIGNMENT>
  </STUDENT>
  <STUDENT stdno="150" program="IT ">
    <NAME>Bill</NAME>
    <ASSIGNMENT no="1">29</ASSIGNMENT>
  </STUDENT>
</CLASS>
Renaming

<! - - rename @no of ASSIGNMNET into @number - - >

<xupdate:rename
    select="doc('/db/pavle/class.xml')/CLASS/STUDENT/ASSIGNMENT/@no">
    number
</xupdate:rename>
XML Concurrency Control

After Renaming

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- The class.xml document after renaming @no of all ASSIGNMENT elements into @number -->
<Class name="COMP5323">
  <Student stdno="100" program="ST">
    <Name>Dang</Name>
    <Assignment number="1">50</Assignment>
    <Assignment number="2">38</Assignment>
  </Student>
  <Student stdno="150" program="IT">
    <Name>Bill</Name>
    <Assignment number="1">29</Assignment>
  </Student>
</Class>
```
Deleting

<! - - Delete the first ASSIGNMENT element of the student Dang - - >

<xupdate:remove
   select="doc('/db/pavle/class.xml')/CLASS/STUDENT[NAME='Dang']/ASSIGNMENT[1]"/>
Possible Issues

• Concurrency control (XML)
• Conflict
  – data-value
  – structural
  – element-order
• Ensure conflict serializability
Concurrency Control

• Concurrency control is a method used to ensure that database transactions are executed in a safe manner.
• ACID properties
  – Atomicity: All actions in the transaction happen, or none happen.
  – Consistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent.
  – Isolation: Execution of one transaction is isolated from that (the effect) of other transactions.
  – Durability: If a transaction commits, its effects persist.
Dataguide model

```
Dataguide model

nodes(id, type, label, value)
child( child-id, parent-id)

Restaurant
  Owner
    Name
      Chili’s
    Entree
      Burger
        Phone
          555-1234
    Manager
      Smith
    Entree
      Darbar
      Lamb
        Curry
      Beef
        Curry
        Rose & Crown
```

XML Concurrency Control
Dataguide model

XML Concurrency Control
Data-value Conflict

Access Path L=/a/b/c
T1: R(L)
T2: W(L)
Structural Conflict

L1=/a/d/e
L2=/a/d

T1: Append(L1,v)
T2: Remove(L2)
Element-order Conflict

L=\langle a/g/f \\nh \\ne \\dh \\dv \\
5 \\8 \\10 \\vu \\
T1: Append(L, u) \\
T2: Append(L, v)
Data-value Conflict

L1=/a/c/d
L2=/a/b/d
T1: Read(L1)
T2: Write(L2)
Structural Conflict and Index Node Split

L1=/a/b/d

L2=/a/b

T1: Append(L1, v)

T2: Remove(L2)
www.xerial.org
- The name comes after XML and serializable
- Taro L. Saito. On concurrency and updatability of XML databases. Master thesis submitted to Department of Computer Science, Graduate School of Information Science and Technology, University of Tokyo, January 2004

Transaction Database for XML
- Concurrent Transactions
  - Serializable schedule
- Recoverability
  - Handling transaction aborts and system failures
- Updating XML
  - Node insertion, deletion, modification, etc.
- Transaction Language
  - Query and update notations
Xerial Overview

Transaction Requests

Multi-Thread

Query Compiler

actions

Transaction Scheduler

Serializable Schedule

Lock Requests

Lock Table

XML source

xml2db

XML Storage

Read & Write

Log

DB Access System

Outputs

XML Concurrency Control
Data Model

<customer id="J-001">
  <name>Jeffrey</name>
  <city>New York</city>
  <order oid="3">
    <item>Notebook</item>
    <date>2002/02/11</date>
    <num>50</num>
  </order>
  <order oid="1">
    <item>Blank Label</item>
    <date>2002/02/10</date>
    <num>100</num>
    <status>delivered</status>
  </order>
</customer>
Querying XML

• **XQuery**
  - W3C standard
  - Query Language for XML
  - Use of *Path expressions*
  - Bind elements to a *variable*

```xml
FOR $x IN /customer/order
WHERE $x/date = "2002/02/13"
```
Locks for Tree-Structure

- **Subtree Level Locking**
  - Query to entire subtree is frequent in XML
  - Reduce the number of locks

- **Performance Factor**
  - The number of locks
    - Load of lock manager
  - Granularity of locks
    - Concurrency
## Locks

**Compatibility Matrix**

**Ordinal Locks**
- *S*  Shared Lock (read)
- *X*  Exclusive Lock (write)

**Warnings**
- *IS*  Intention to Share
- *IX*  Intention to Exclusive

<table>
<thead>
<tr>
<th></th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IX</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>S</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Warning Protocol

- Original Rules
  - All transactions must enter from the root
  - To place a lock or warning on any element, we must hold a warning on its parent
  - Never remove a lock or warning unless we hold no locks or warnings on its children
Warning Protocol for XML

- **Extension**
  - When we insert or delete nodes, we must obtain X lock on the parent of the destination
  - Until we place a warning on a node, we cannot trace its pointers to the children
  - A transaction never releases locks or warnings until it finishes
    - **2 phase locking**
**XTC**

- XTC is used as a test vehicle for empirical DB research
  - effectiveness of XML concurrency control
    - fine-granular locking on nodes and edges
    - meta-synchronization allows comparison of different compatibilities
  - taDOM* protocols
    - multiplicity of lock modes
    - intention locks are important
    - indexed document access is frequent
    - ancestor path locking without accessing the storage engine desirable

- References
determination of ancestor node IDs are of outmost importance for any locking protocol.
<?xml version="1.0"?>
<bib>
    <book year="2004" id="book1">
        <title>The Title</title>
        <author>
            <last>Lastname</last>
            <first>Firstname</first>
        </author>
        <price>49.99</price>
    </book>
</bib>
**Tailored Node Locks for XML – taDOM2**

### Read locks

<table>
<thead>
<tr>
<th>lock</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>intention read lock on a node</td>
</tr>
<tr>
<td>NR (node read)</td>
<td>locks only a context node</td>
</tr>
<tr>
<td>LR (level read)</td>
<td>read lock on a context node and all direct-child nodes</td>
</tr>
<tr>
<td>SR (subtree read)</td>
<td>read lock on an entire subtree</td>
</tr>
</tbody>
</table>

### Write locks

<table>
<thead>
<tr>
<th>lock</th>
<th>effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX (intent. excl.)</td>
<td>intention of a write lock on a non-direct child node</td>
</tr>
<tr>
<td>X (exclusive)</td>
<td>write lock on an entire subtree</td>
</tr>
<tr>
<td>CX (child excl.)</td>
<td>indicates existence of a write lock on a direct child node</td>
</tr>
<tr>
<td>SU (update option)</td>
<td>read lock for intended update operations on an entire subtree</td>
</tr>
</tbody>
</table>
Tailored Node Locks for XML – taDOM2

Compatibility matrix

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>IR</th>
<th>NR</th>
<th>LR</th>
<th>SR</th>
<th>IX</th>
<th>CX</th>
<th>SU</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>NR</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LR</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SR</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IX</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CX</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SU</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>X</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Node Locks

- Node read lock (NR)
  - requires IR locks on the ancestor path
- Level read lock (LR)
  - requested for reading the context node and all nodes located at the level below (all direct-child nodes)
- Child exclusive lock (CX)
  - indicates an X lock on a child
  - defined, in addition to IX, to detect conflicts with LR
Node Locks

Transaction $T_1$ is reading $<price>$

Transaction $T_2$ is reading $<book>$ and all direct-child nodes ($<title>$, $<author>$, and $<price>$)

Transaction $T_3$ is modifying the book title
Node Locks

- Locking subtrees exclusively: intention exclusive lock (IX), child exclusive lock (CX), and exclusive lock (X)
  - requested for updating the context node's content or deleting the context node and its entire subtree
  - requires a CX lock on the parent and IX locks on the ancestors
Node Locks

Transaction $T_1$ is deleting the `<first>` node and its content

Transaction $T_2$ is deleting the `<last>` node and its content

Transaction $T_3$ is reading the `<author>` node

Transaction $T_4$ is reading all direct-child nodes of `<book>` but is blocked when reading all child nodes of `<author>`
Tunable Lock Depth

- Goal
  - reduce the number of locks held by using coarser lock granularity
  - may decrease concurrency
  - when nodes deeper than lock depth are accessed: lock modes SR and X are used at the lock depth level
Tunable Lock Depth

Transaction $T_1$ is reading the author's last name.

Transaction $T_2$ is updating the author's first name.

**using lock depth 2**

Transaction $T_1$ would have to acquire an SR lock on author.

Transaction $T_2$ would have to acquire an X lock on author.

would therefore have to wait on author.
Sequential

- very slow, although supported by on-demand indexes
- determination of parent ID and ancestor IDs, however, is very frequent
Identifying Nodes – Node Numbering Schemes

SPLIs (DeWeysl Ds)

1 bib
1.3 book
1.3.1 title
1.3.3 author
1.3.5
1.3.5.3
1.3.5.3.3
1.3.5.5
1.3.5.5.3
1.3.7 price
1.3.7.3

1.3.1.1 id
1.3.1.3
1.3.1.1.1
1.3.1.3.1
1.3.3.3
1.3.3.3.1
1.3.3.3.1.1
1.3.3.3.1.3
1.3.3.3.1.3.1
1.3.5.3
1.3.5.3.3
1.3.5.3.3.1
1.3.5.5
1.3.5.5.3
1.3.5.5.3.1
1.3.7.3
1.3.7.3.1
OptiX/SnaX

- Locking overhead, especially for read operations, can be tremendous.
- Two snapshot based concurrency control mechanisms that avoid locking.
  - OptiX is a variation of optimistic concurrency control adjusted to use snapshots and work on XML data.
  - SnaX provides the isolation level of snapshot isolation and has similar semantics as the concurrency control mechanisms implemented
Snapshot based Concurrency Control

- Each transaction
  - Working phase, validation phase, update phase
- Two transactions Ti, Tj
  - Concurrent if Ti started the working phase before Tj finished the update phase and committed
- Validation
  - OptiX: WriteSet(Ti) not overlaps with ReadSet(Tj)
  - SnaX: WriteSet(Ti) not overlaps with WriteSet(Tj)
    - Snapshot isolation as in many RDBMS
    - Assuming that many applications are read only
ReadSets

• XML structure
  – RR(Ti) : roots of subtree
  – ER(Ti) : nodes that are explicitly read as part of a predicate or path constraint, but not return as part of result
  – ERa(Ti) : nodes that are read for the insertion of a sibling after them
WriteSets

• **XML structure**
  – $D(T_i)$: roots of subtree deleted or replaced
  – $R_n(T_i)$: nodes to be renamed
  – $I(T_i)$: the immediate parents of any nodes inserted by $T_i$
  – $I_{a}(T_i)$: the same set of nodes as in $E_{r}(T_i)$
OptiX Validation

• Suppose Ti wants to validate, and concurrent transaction Tj validated before Ti.

• When ReadSet(Ti) overlaps with WriteSet(Tj)
  – Instead of immediately inducing a conflict, we look at the subsets of ReadSet(Ti) and WriteSet(Tj) of which p is a member.
  – YES indicates compatibility, while NO shows a conflict leading to an abort of Ti.

• A conflict means that if Ti had executed serially after Tj, then it would have possibly read a different value for p than in the concurrent execution.

• In the table q is a descendant of p.
## OptiX Validation

Tj already validated on p

<table>
<thead>
<tr>
<th>Ti validating</th>
<th>D</th>
<th>Rn</th>
<th>I</th>
<th>Ia</th>
</tr>
</thead>
<tbody>
<tr>
<td>On p RR</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>On p ER</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>On p ERa</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>On q RS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

XML Concurrency Control
SnaX Validation

- In SnaX, remember snapshot isolation does not guarantee that there is an equivalent serial execution.
  - ignore the read sets and only consider write/write conflicts.
- Generally, delete and replace conflict with most other update types, while inserts conflict only with few.
### SnaX Validation

Tj already validated on p

<table>
<thead>
<tr>
<th>Ti validating</th>
<th>D</th>
<th>Rn</th>
<th>I</th>
<th>Ia</th>
</tr>
</thead>
<tbody>
<tr>
<td>On p D</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>On p Rn</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>On p I</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>On p Ia</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>On q WS</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Motivation: Decision support queries that read large amounts of data have concurrency conflicts with OLTP transactions that update a few rows
  – Poor performance results

Solution 1: Give logical “snapshot” of database state to read only transactions, read-write transactions use normal locking
  – Multiversion 2-phase locking
  – Works well, but how does system know a transaction is read only?

Solution 2: Give snapshot of database state to every transaction, updates alone use 2-
Extra slides
Snapshot Isolation

A transaction T1 executing with Snapshot Isolation
- takes snapshot of committed data at start
- always reads/modifies data in its own snapshot
- updates of concurrent transactions are not visible to T1
- writes of T1 complete when it commits
- First-committer-wins rule:
  - Commits only if no other concurrent transaction has already written data that T1 intends to write.

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W(Y := 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Commit</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Start</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R(X) → 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R(Y) → 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W(X := 2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>W(Z := 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R(Z) → 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R(Y) → 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>W(X := 3)</td>
<td>Commit-Req</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Abort</td>
</tr>
</tbody>
</table>

Concurrent updates not visible
Own updates are visible
Not first-committer of X
Serialization error, T2 is rolled back

XML Concurrency Control
Benefits of SI

- Reading is *never* blocked,
  - and also doesn’t block other txns activities
- Performance similar to Read Committed
- Avoids the usual anomalies
  - No dirty read
  - No lost update
  - No non-repeatable read
  - Predicate based selects are repeatable (no phantoms)
- Problems with SI
  - SI does not always give serializable executions
    - Serializable: among two concurrent txns, one sees the effects of the other
    - In SI: neither sees the effects of the other
  - Result: Integrity constraints can be violated
Snapshot Isolation Anomalies

- SI breaks serializability when transactions modify *different* items, each based on a previous state of the item the other modified.
- SI can also cause a read-only transaction anomaly, where read-only transaction may see an inconsistent state even if updaters are serializable.
SI In Oracle

- **Warning**: SI used when isolation level is set to serializable by Oracle
  - Oracle implements “first updater wins” rule (variant of “first committer wins”)
    - concurrent writer check is done at time of write, not at commit time
    - Allows transactions to be rolled back earlier
  - Does not support true serializable execution