XML Transactions

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

XQuery Update

- XQuery Update Facility 1.0
  - http://www.w3.org/TR/2008/CR-xquery-update-10-20080801/
- Primitive Updates: insert, delete, replace, rename
  - Extensions to other expr: FLWOR, TypeSwitch, ...

Examples

- delete nodes //book[@year lt 1968]

- insert node <author/> into //book[@ISBN eq "34556"]

-
  ```
  for $x in //book
    where $x/year lt 2000 and $x/price gt 100
    return replace value of node $x/price
    with $x/price-0.3*$x/price
  ```

-
  ```
  if ($book/price gt 200) then
  rename node $book as "expensive-book“
  ```

Update statements work on "node" or "nodes"

Insert

- Insert a new element into a document

- **insert node** UpdateContent **into** TargetNode
  - UpdateContent: any sequence of items (nodes, values)
  - TargetNode: Exactly one document or element
    - Otherwise ERROR

- Optionally, specify if to insert at the beginning or end
  - as last: Content becomes first child of Target
  - as first: Content becomes last child of Target
  - No position: no fixed position (honor other first/last inserts)
- Nodes in Content assume a new Id.

Insert

- Insert new book in the library

  insert node <book> <title>Die wilde Wutz</title> </book>
  into document("www.uni-bib.de")//bib

- Insert new book at the beginning of the library

  insert node <book> <title>Die wilde Wutz</title> </book>
  as first into document("www.uni-bib.de")//bib

- Insert new attribute into an element

  insert node (attribute age { 13 })
  into document("persons.xml")//person[@name = "KD"]

Insert

- Insert at a particular point in the document
- **insert node** UpdateContent **(after | before)** TargetNode
  - UpdateContent: No attributes allowed!
  - TargetNode: One Element, Comment or PI.
    - Otherwise ERROR
    - Must have parent
- Specify whether before or behind target
  - Before vs. After
- Nodes in Content assume new identity

Add an author to a book

insert node <author author>Florescu Florescu</ </author author>
before // article[title = "XL"]/ [author . = " Vincent"]

Delete

- Deletes nodes from a document

- **delete (node | nodes)** TargetNodes
  - TargetNodes: Any sequence of nodes

- Delete papers.
  delete node //article[header/keyword = "XML"]

Replace

- Replace a node

  replace node TargetNode **with** UpdateContent

- Replace the content of a node

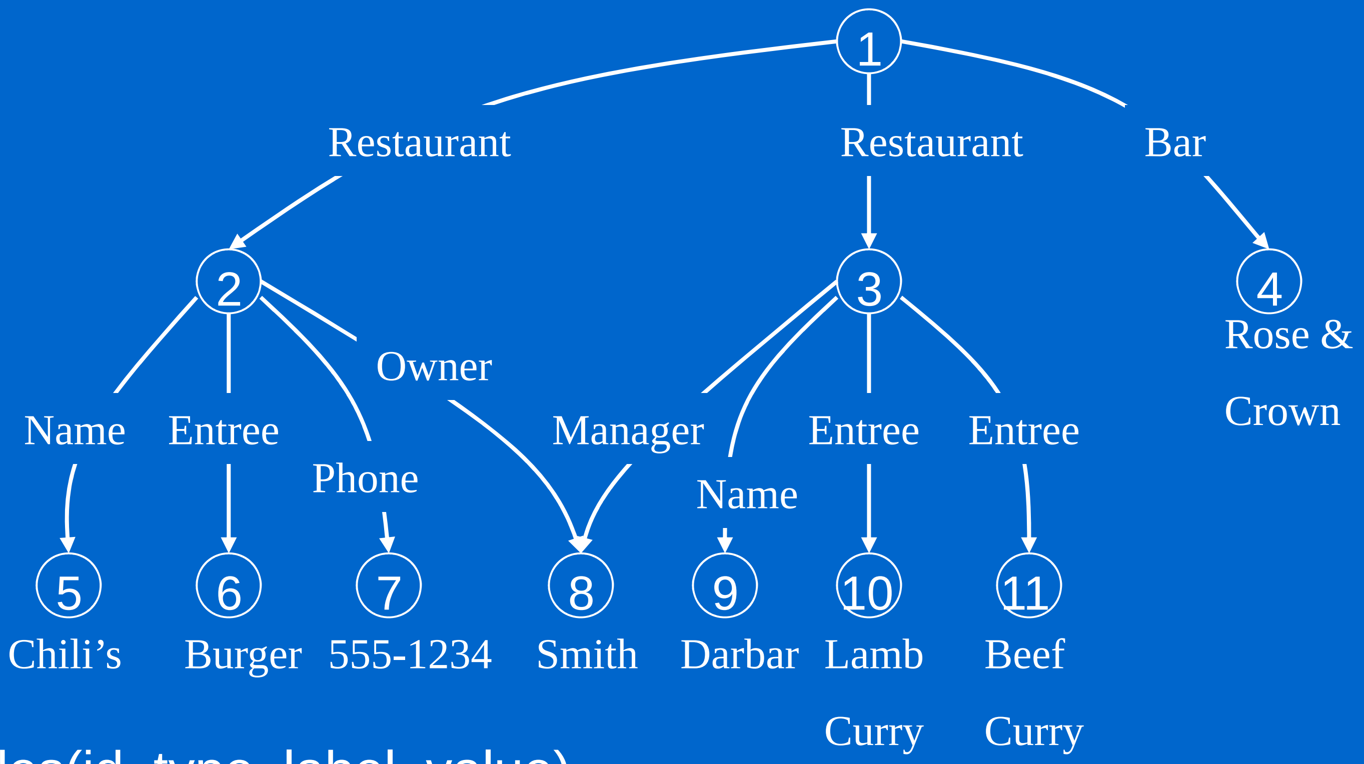
  replace value of node TargetNode **with** UpdateContent

  – TargetNode: One node (with Id)
  – UpdateContent: Any sequence of items
  – Keeps the node ID of TargetNode

Concurrency Control

- Concurrency control is a method used to ensure that database transactions are executed in a safe manner.
- ACID properties
  - A tomicity: All actions in the transaction happen, or none happen.
  - C onsistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent.
  - I solation: Execution of one transaction is isolated from that (the effect) of other transactions.
  - D urability: If a transaction commits, its effects persist.

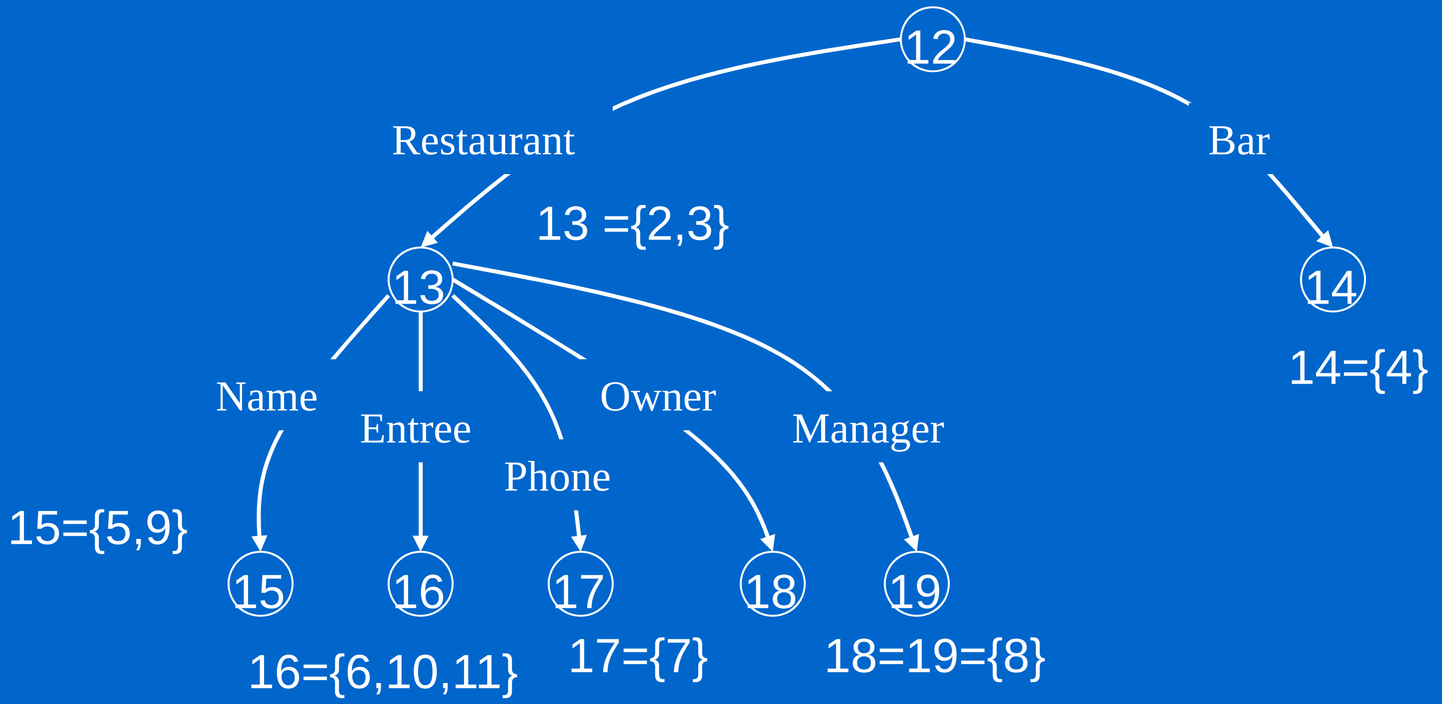
Dataguide model



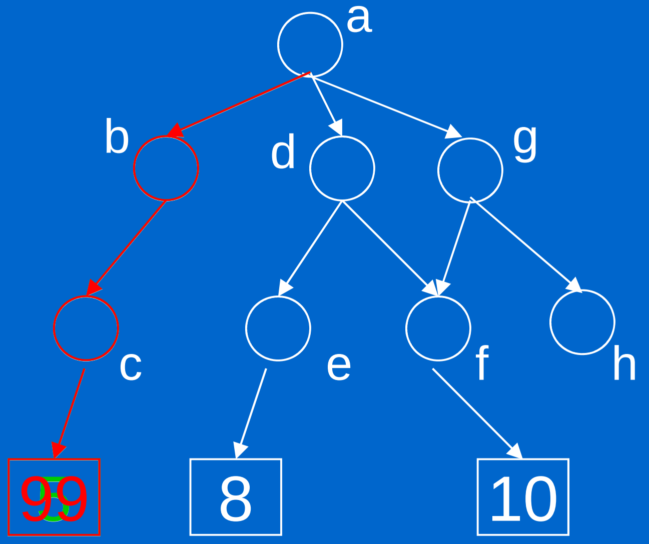
nodes(id, type, label, value)

child(child-id, parent-id)

Dataguide model



Data-value Conflict



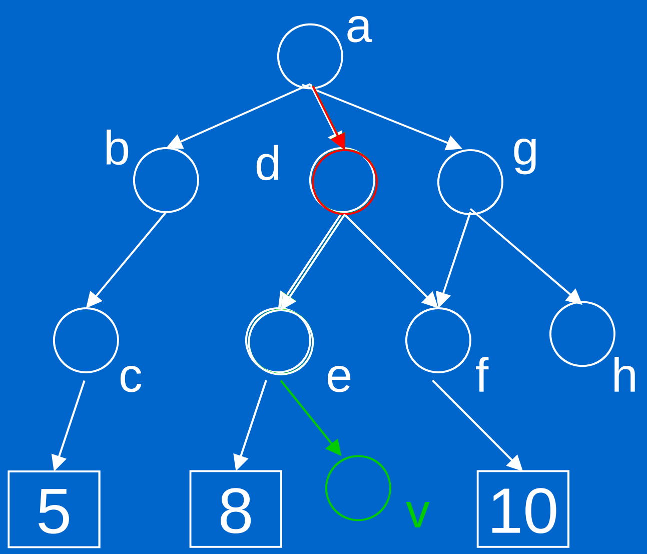
Access Path L=/a/b/c

T1: R(L) r(a)r(b)r(c)

T2: W(L) r(a)r(b)w(c)

r(a)r(b) r(a)r(b) w(c) r(c) abort

Structural Conflict



L1=/a/d/e

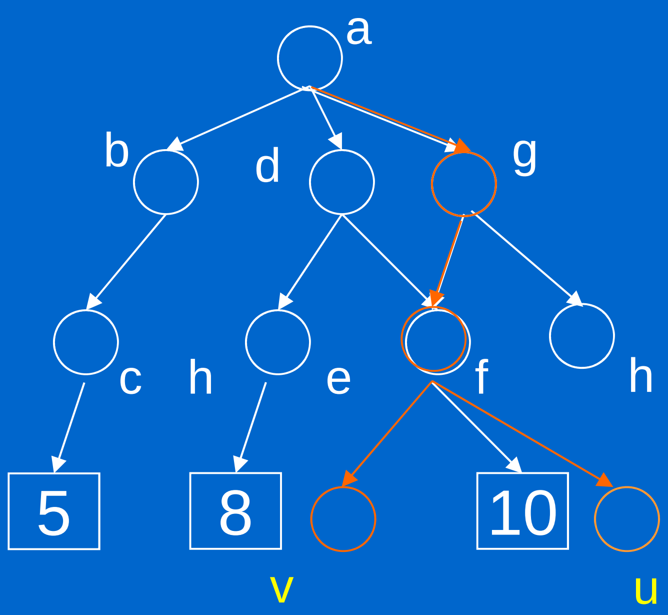
L2=/a/d

T1: Append(L1,v) - r(a)r(d)r(e)w(v)w(e)

T2: Remove(L2) - r(a)r(d)w(d)

r(a) r(a) r(d)r(e)w(v) r(d)w(d) w(e)

Element-order Conflict



L=/a/g/f

T1: Append(L, u) r(a)r(g)r(f)w(u)w(f)

T2: Append(L, v) r(a)r(g)r(f)w(v)w(f)

What is a possible problem?

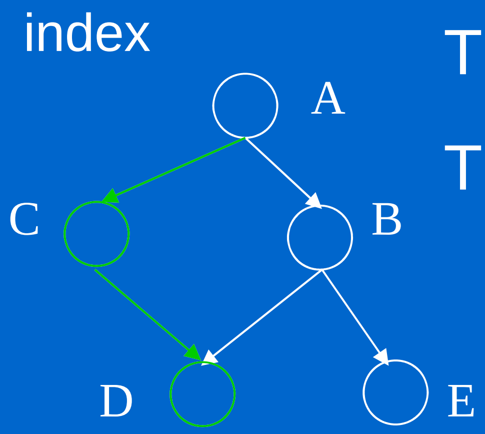
Data-value Conflict

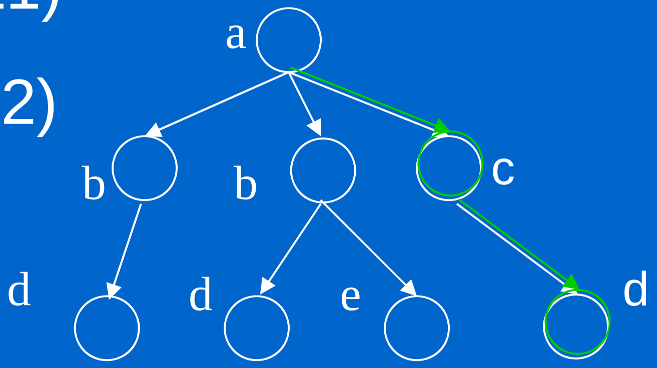
L1=/a/c/d

L2=/a/b/d

T1: Read(L1)

T2: Write(L2)





Problem in the index tree, common D node

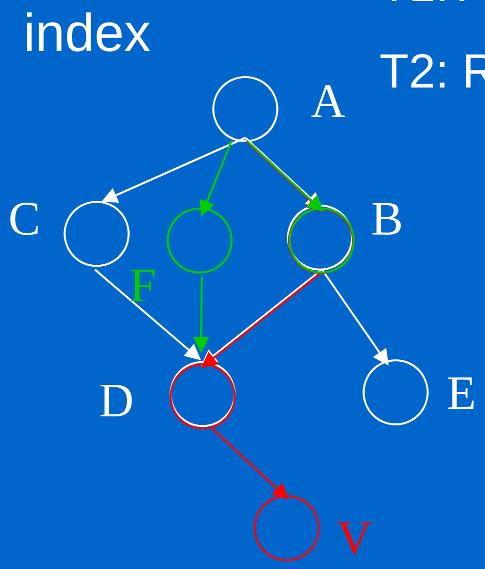
Structural Conflict and Index Node Split

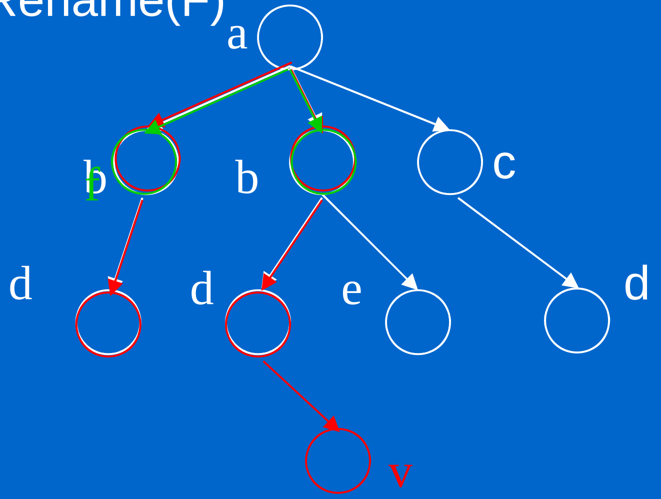
L1=/a/b/d

L2=/a/b

T1: Append(L1, v)

T2: Remove(L2) Rename(F)

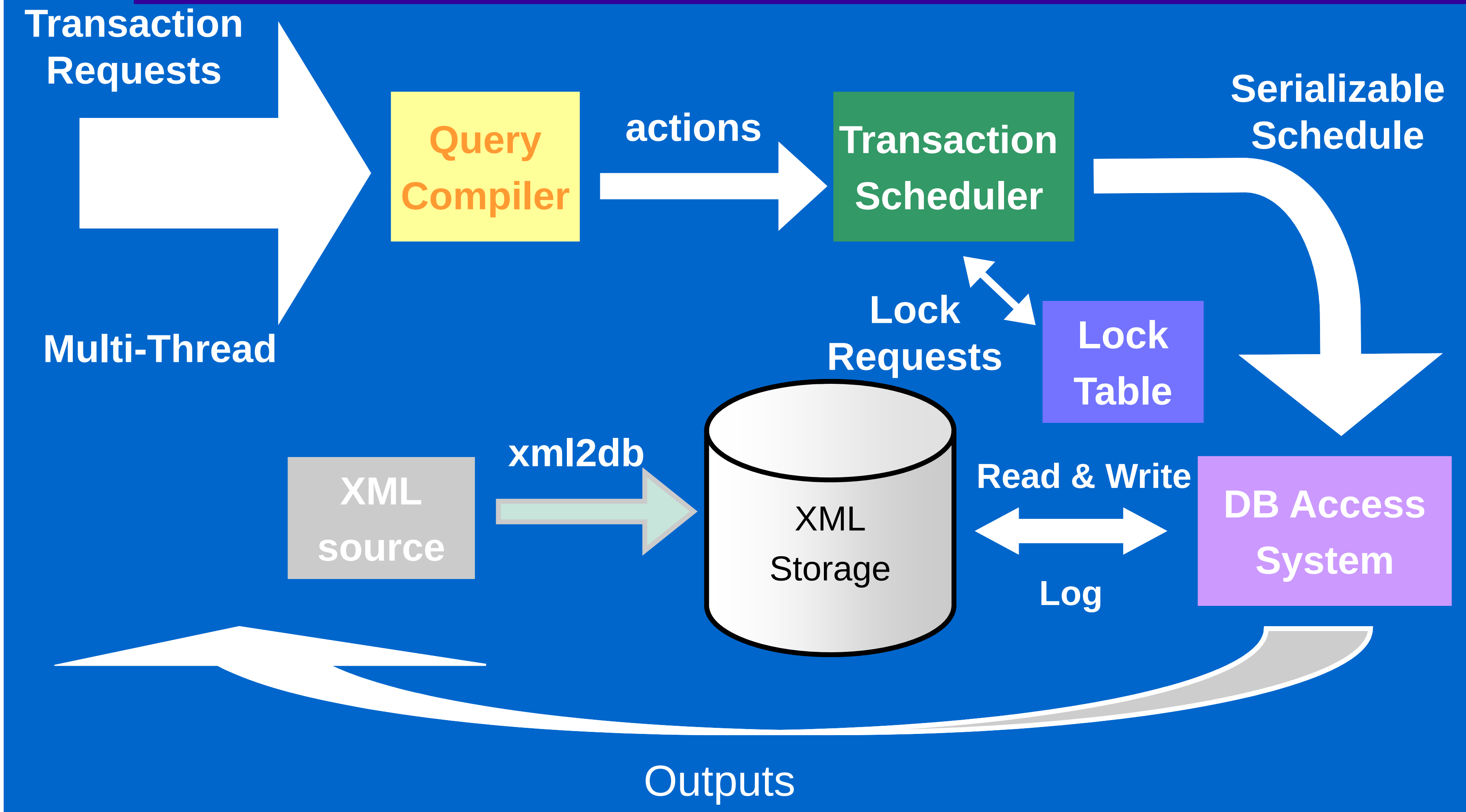




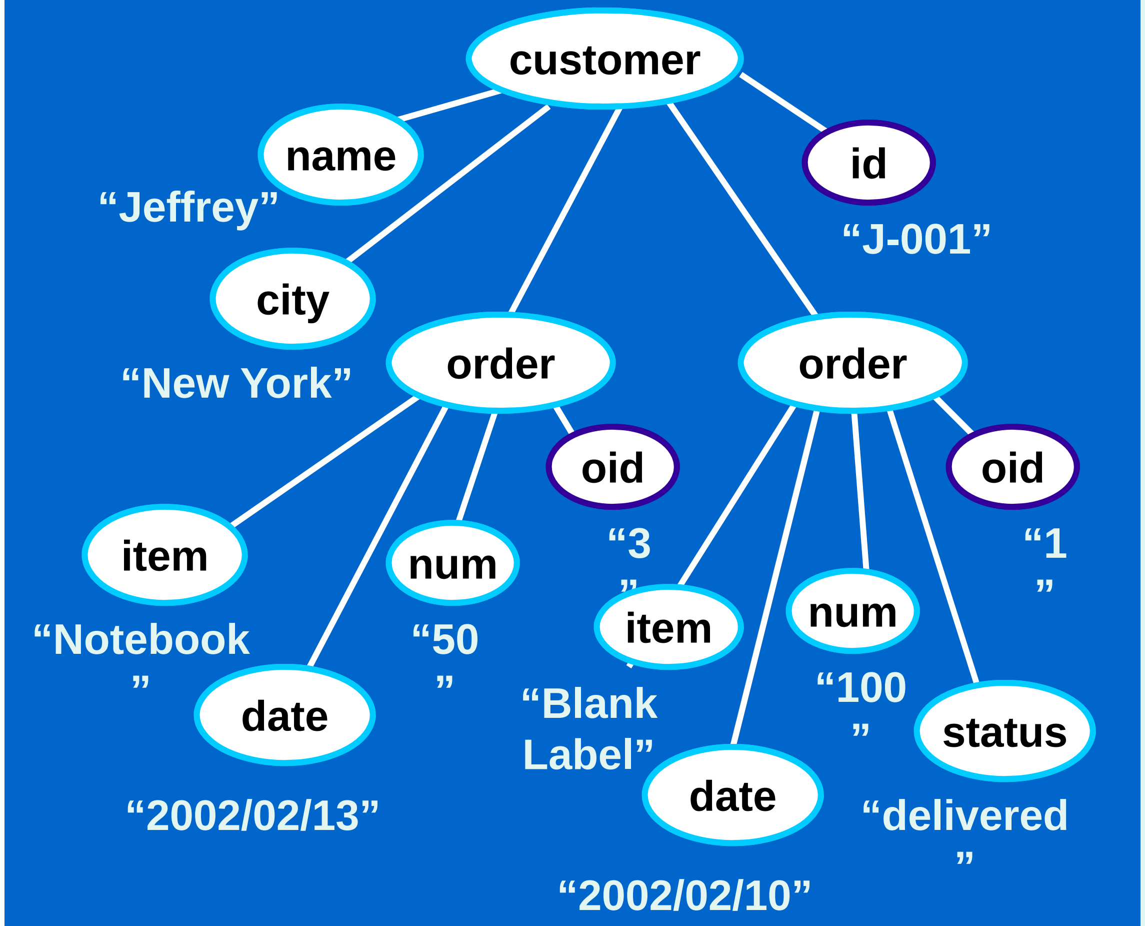
Xerial

- 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
- Transactional 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

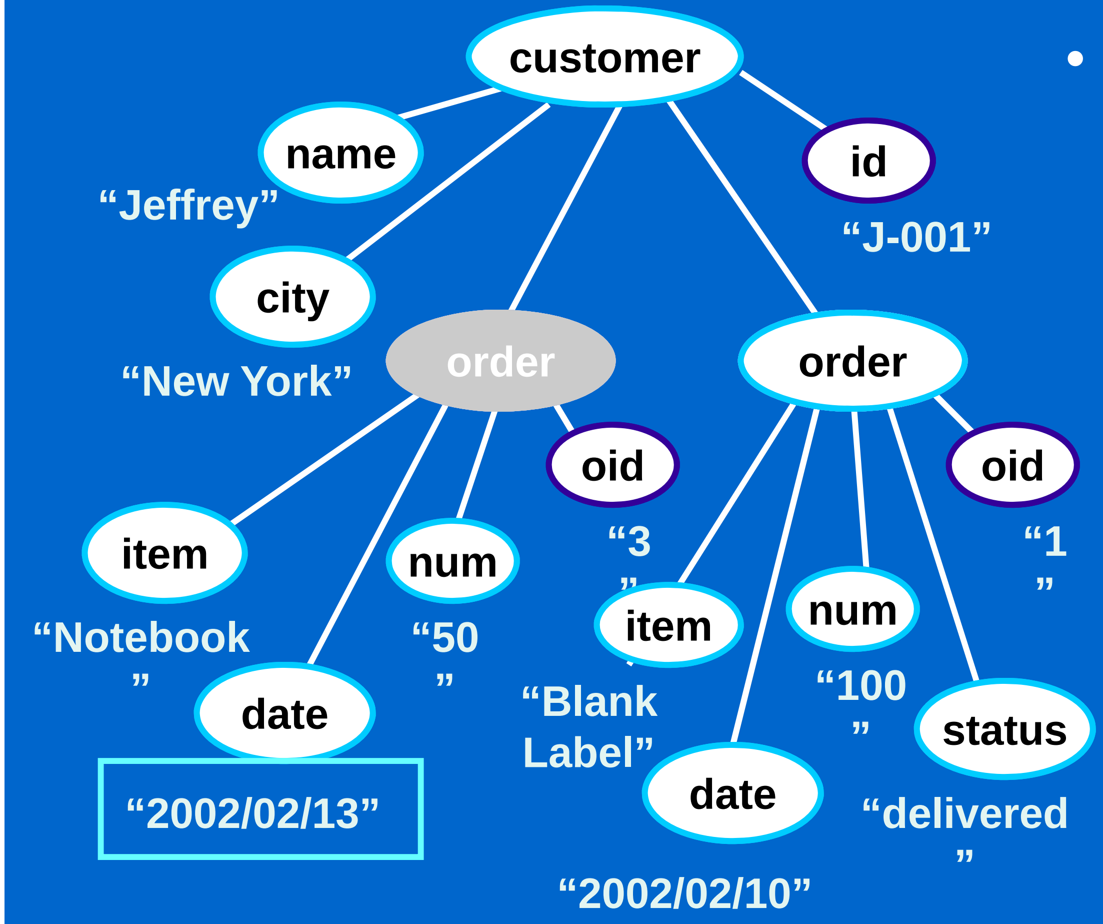


Data Model



```
<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>
```

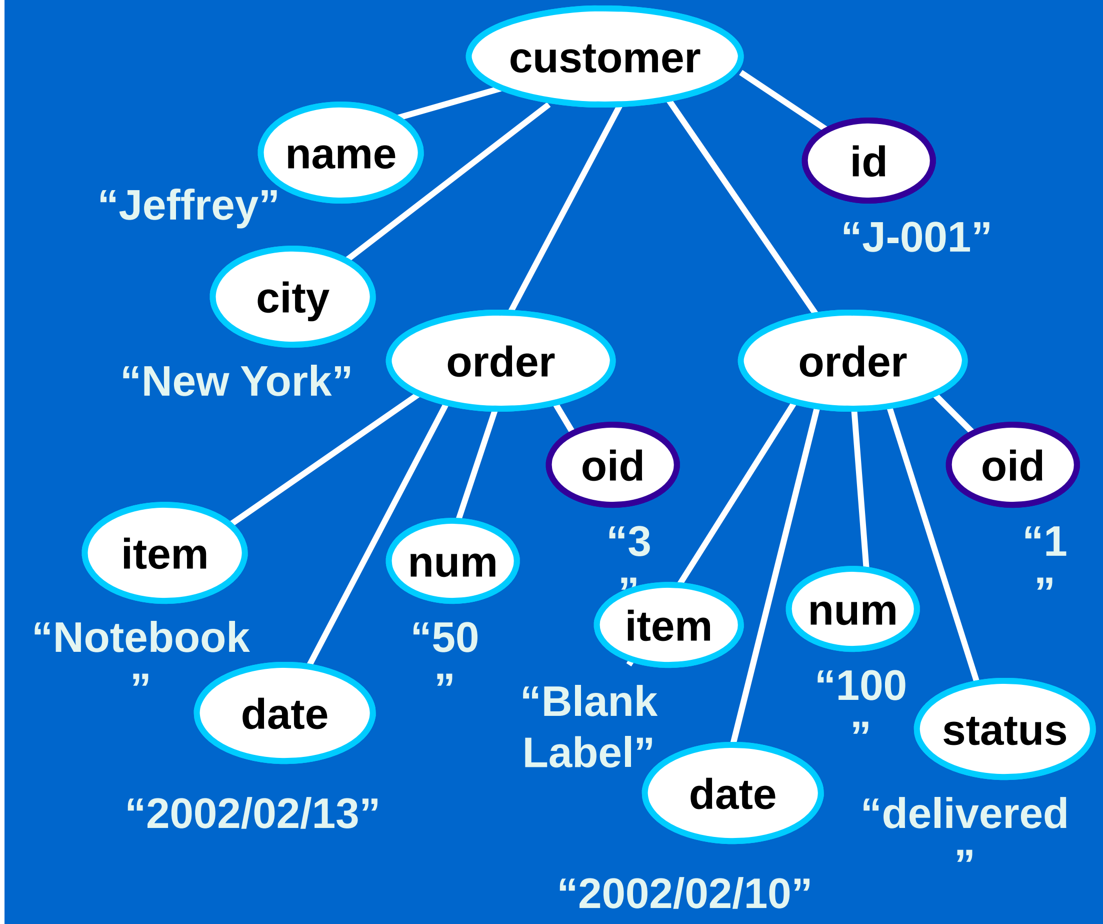
Querying XML



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

```
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

t1/t2	IS	IX	S	X
IS	Yes	Yes	Yes	No
IX	Yes	Yes	No	No
S	Yes	No	Yes	No
X	No	No	No	No

Compatibility Matrix

Ordinal Locks

•*S* Shared Lock (read)

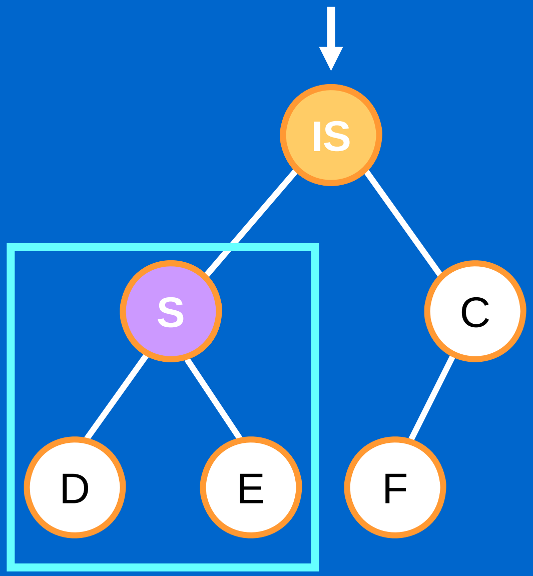
•*X* Exclusive Lock (write)

Warnings

•*IS* Intention to Share

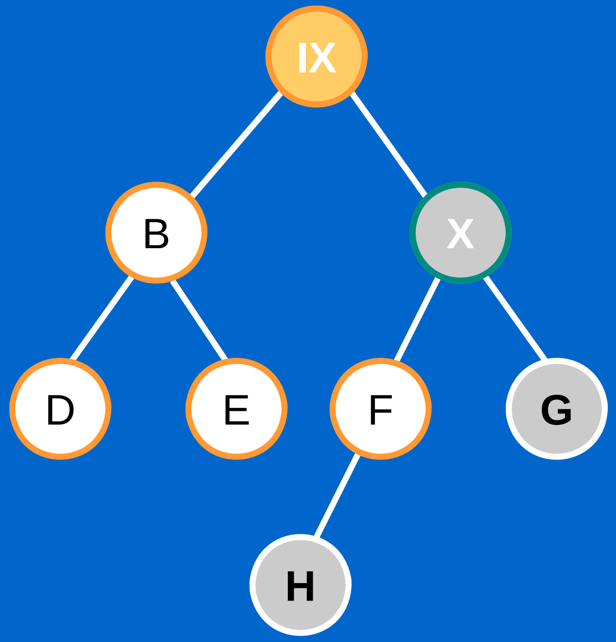
•*IX* Intention to Exclusive

Warning Protocol



- Jim Gray et al, 1975.
- 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

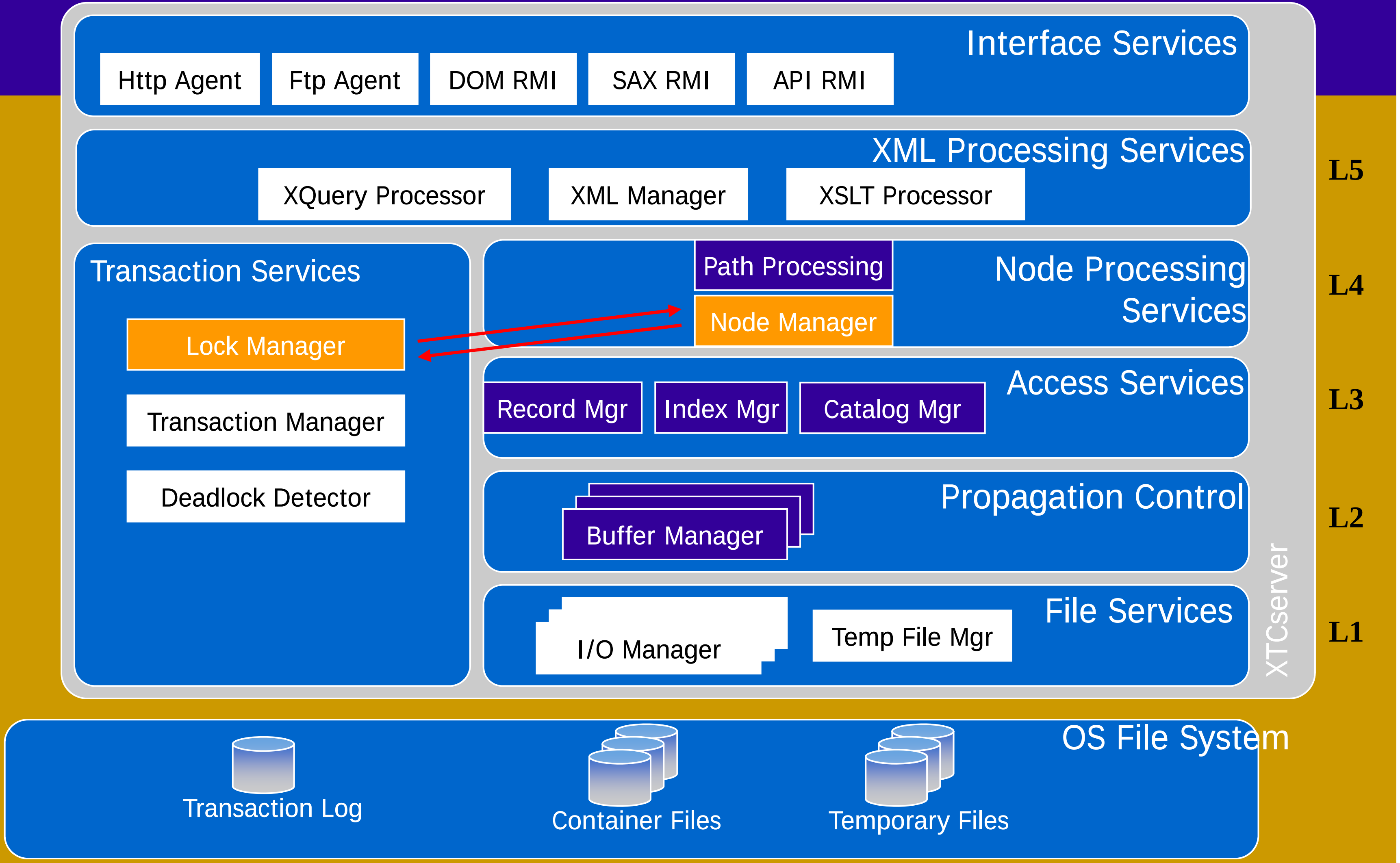


- **Extension**
  - When we insert or delete nodes, we must obtain *X* lock on the *parent* of the destination
  - Until we place a warning on a node, we cannot trace its pointers to the children
  - A transaction never releases locks or warnings until it finishes
    - **2 phase locking**

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
  - Michael P. Haustein, Theo Haerder: Twig Query Processing Under Concurrent Updates, in ICDE. 2006.
  - Michael P. Haustein, Contest of XML Lock Protocols, VLDB 2006 (www.vldb.org/conf/2006/p1069-haustein.pdf)

XTC – Architectural Overview



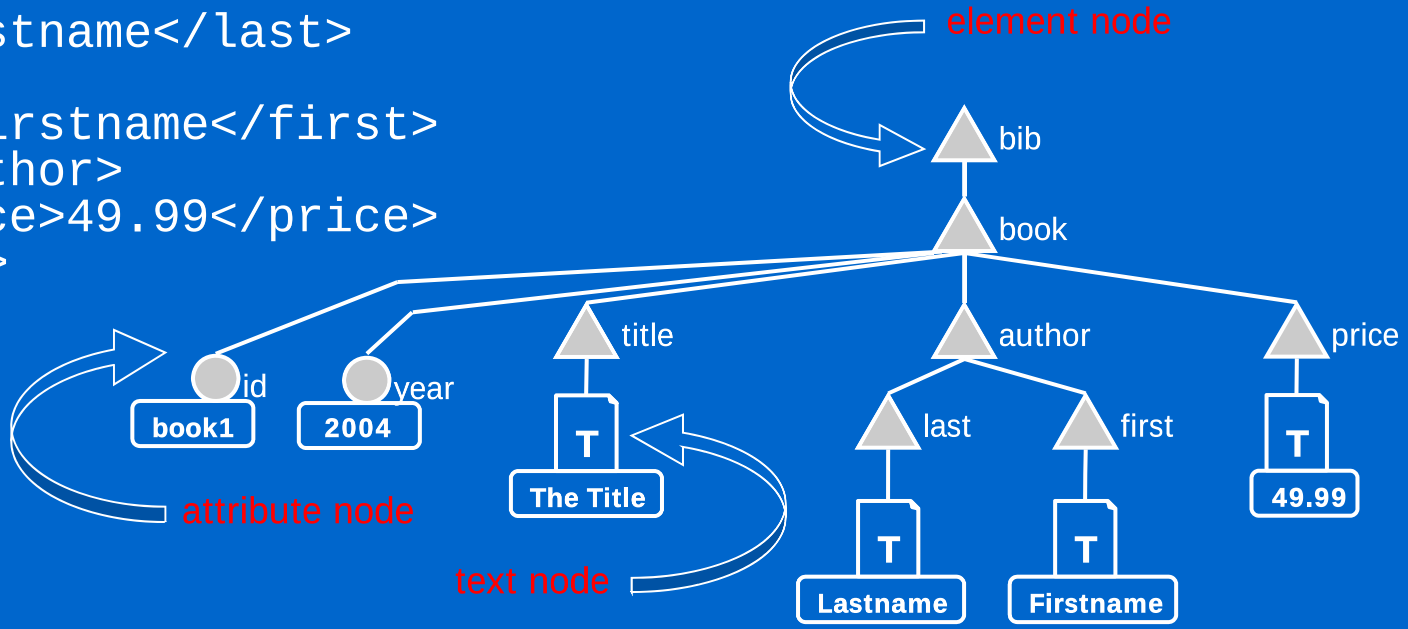
determination of ancestor node IDs are of outmost importance
for any locking protocol

XML Document

```
<?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

lock	effect
IR	intention read lock on a node
NR (node read)	locks only a context node
LR (level read)	read lock on a context node and all direct-child nodes
SR (subtree read)	read lock on an entire subtree

Write locks

lock	effect
IX (intent. exclusive)	intention of a write lock on a non-direct child node
X (exclusive)	write lock on an entire subtree
CX (child exclusive)	indicates existence of a write lock on a direct child node
SU (update option)	read lock for intended update operations on an entire subtree

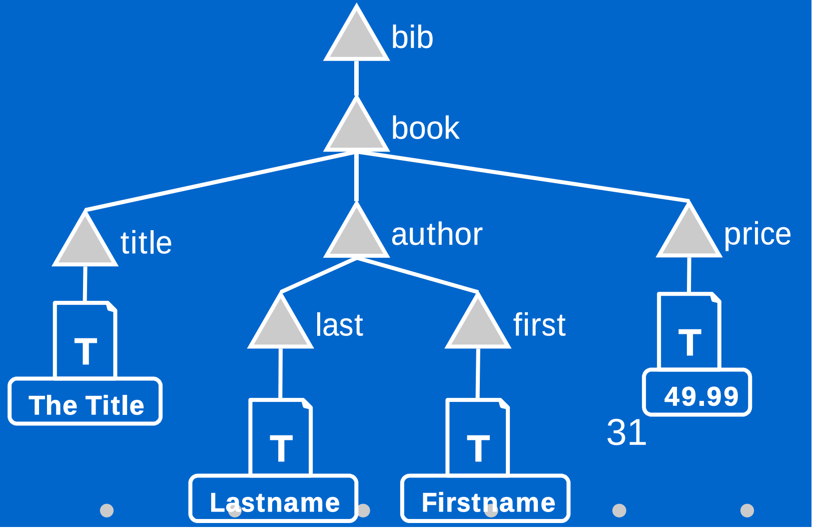
Tailored Node Locks for XML – taDOM2

Compatibility matrix

	-	IR	NR	LR	SR	IX	CX	SU	X
IR	+	+	+	+	+	+	+	-	-
NR	+	+	+	+	+	+	+	-	-
LR	+	+	+	+	+	+	-	-	-
SR	+	+	+	+	+	-	-	-	-
IX	+	+	+	+	-	+	+	-	-
CX	+	+	+	-	-	+	+	-	-
SU	+	+	+	+	+	-	-	-	-
X	+	-	-	-	-	-	-	-	-

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

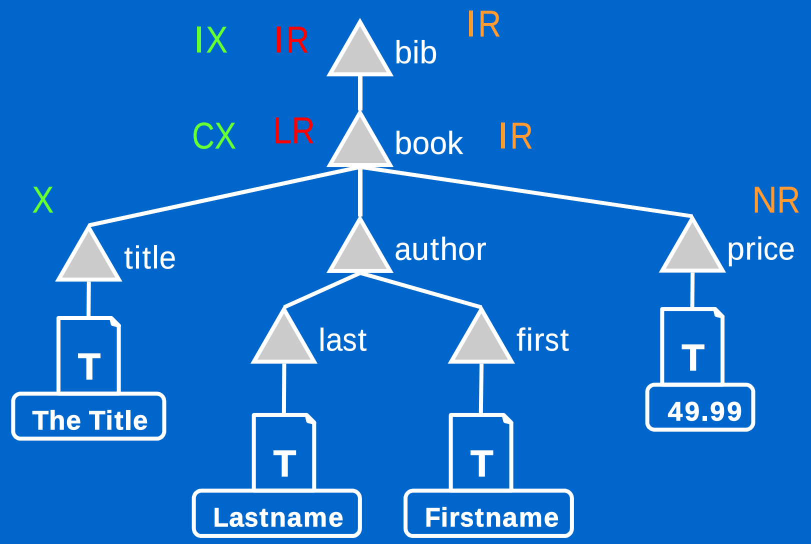


Node Locks

Transaction T_1 is reading <price>

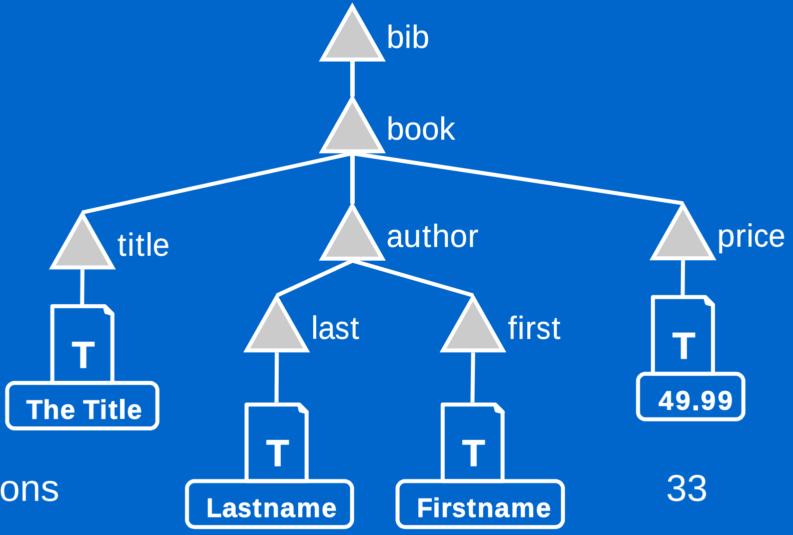
Transaction T_2 is reading <book> and all direct-child nodes (<title>, <author>, and <price>)

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)
- 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

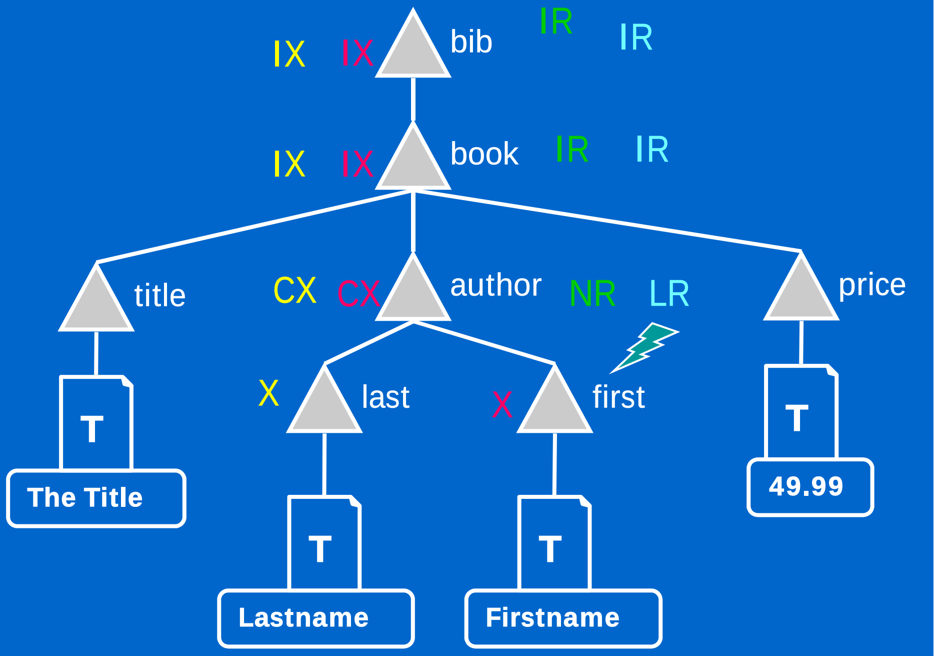
Transaction T_1 is deleting the <first> node and its content

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

Transaction T_3 is reading the <author> node

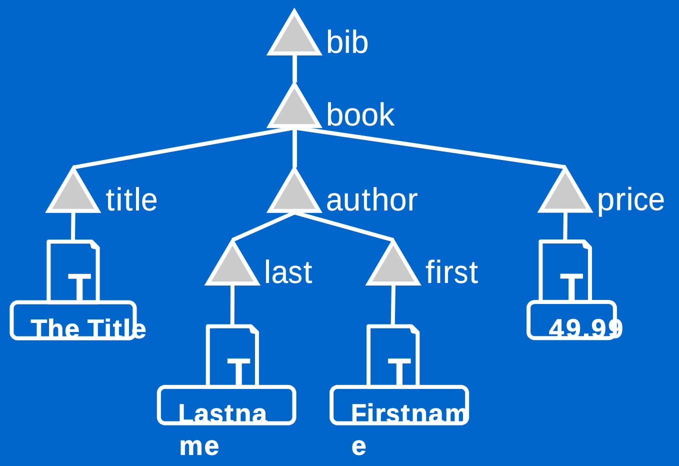
Transaction T_4 is reading all direct-child nodes of <author>

but is blocked when reading all child nodes of <author> by T_1



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 types SR and X are used at the lock depth level



Tunable Lock Depth

Transaction T_1 is reading the author's last name

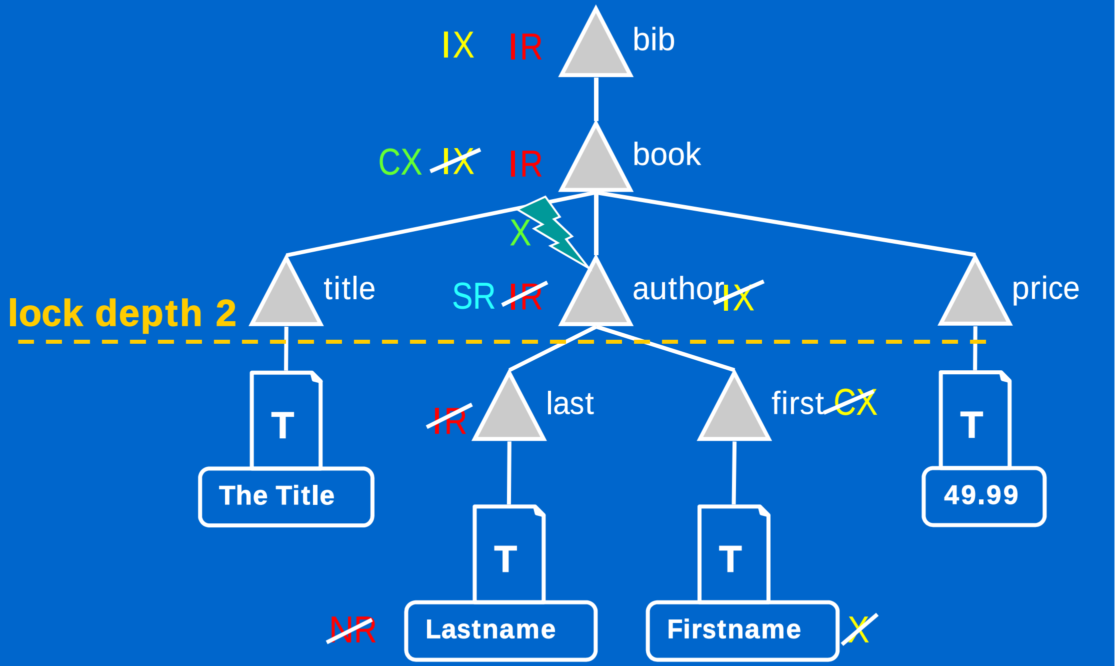
Transaction T_2 is updating the author's first name

Transaction T_1 would have to acquire an SR lock on author

Transaction T_2 would have to acquire an X lock on author

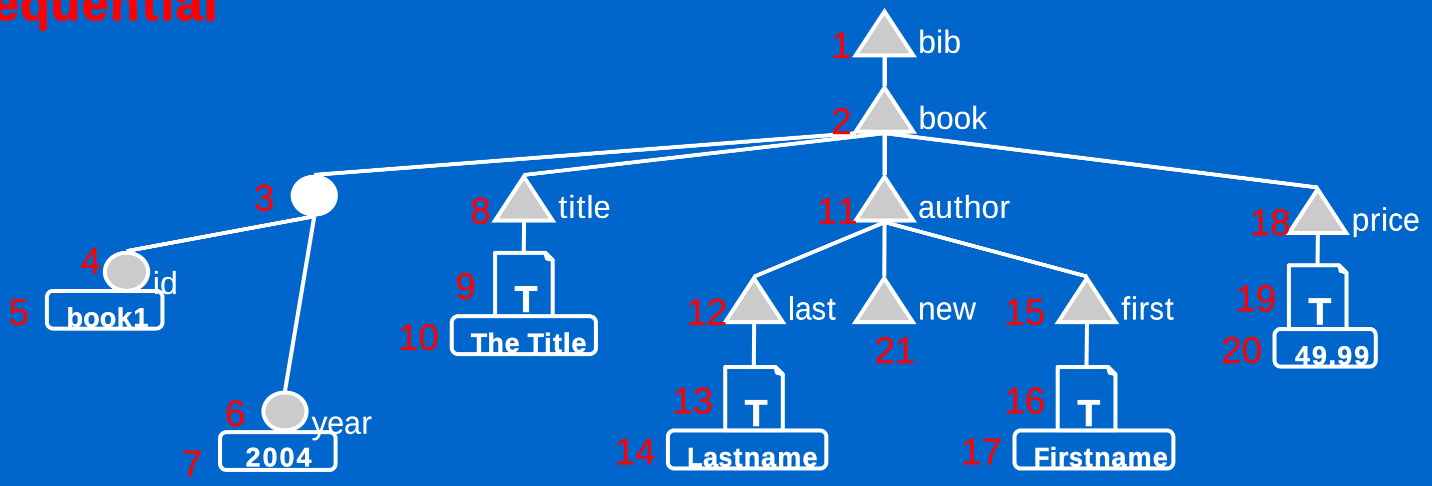
using

would therefore have to wait on author



Identifying Nodes – Node Numbering Schemes

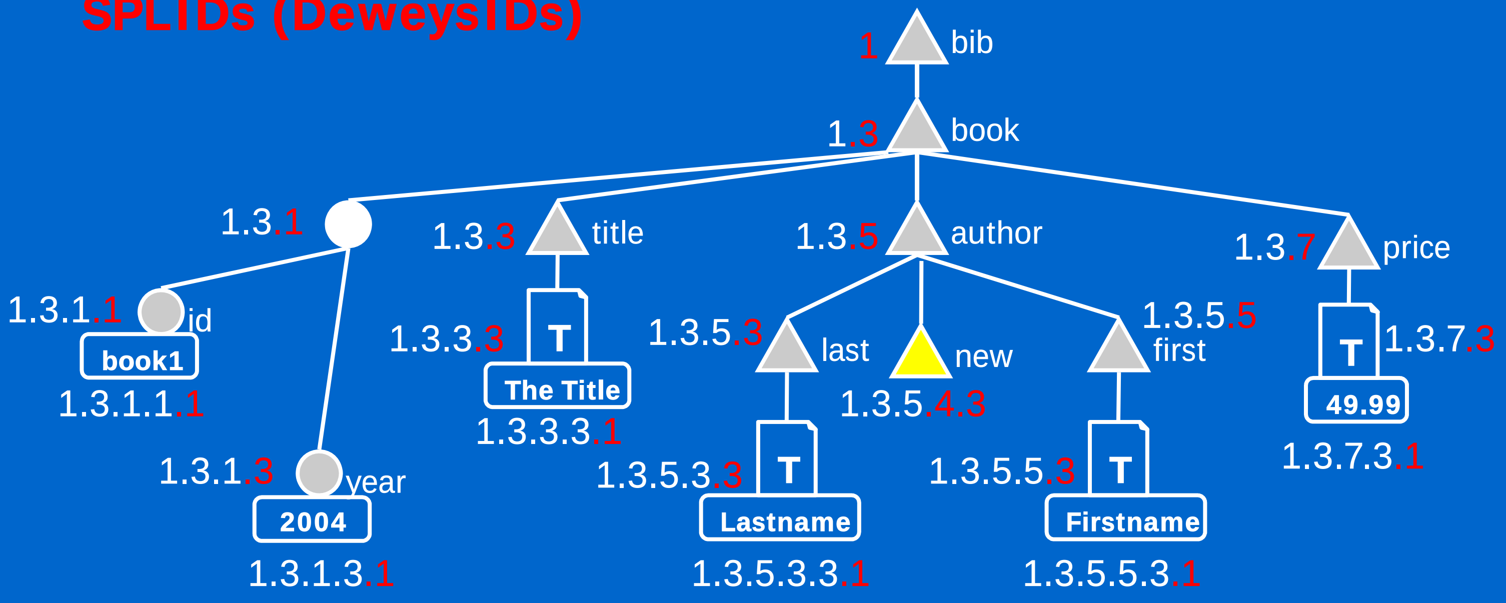
Sequential



- very slow, although supported by on-demand indexes
- determination of parent ID and ancestor IDs, however, is very frequent

Identifying Nodes – Node Numbering Schemes

SPLIDs (DeweysIDs)



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
- Zeeshan Sardar, Bettina Kemme. “Don’t be a Pessimist: Use Snapshot based Concurrency Control for XML”, ICDE 2006.

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(Tj) not overlaps with ReadSet(Ti)
  - SnaX: WriteSet(Tj) not overlaps with WriteSet(Ti)
    - Snapshot isolation as in many RDBMS
    - Assuming that many applications are read only

ReadSets

- XML structure
  - RR(Ti) : roots of subtree returned as part of the query result
  - 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(Ti) : roots of subtree deleted or replaced
  - Rn(Ti) : nodes to be renamed
  - I(Ti) : the immediate parents of any nodes inserted by Ti
  - Ia(Ti) : the same set of nodes as in Era(Ti)

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

Ti validating	D	Rn	I	Ia
On p RR	**No**	**No**	**No**	Yes
On p ER	**No**	**No**	Yes	Yes
On p ERa	**No**	**No**	Yes	**No**
On q ReadSet	**No**	Yes	Yes	Yes

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

Ti validating	D	Rn	I	Ia
On p D	**No**	**No**	**No**	Yes
On p Rn	**No**	**No**	Yes	Yes
On p I	**No**	Yes	Yes	Yes
On p Ia	Yes	Yes	Yes	**No**
On q WriteSet	**No**	Yes	Yes	Yes