Updating XML Data & Transaction Support for XML

Saarland University

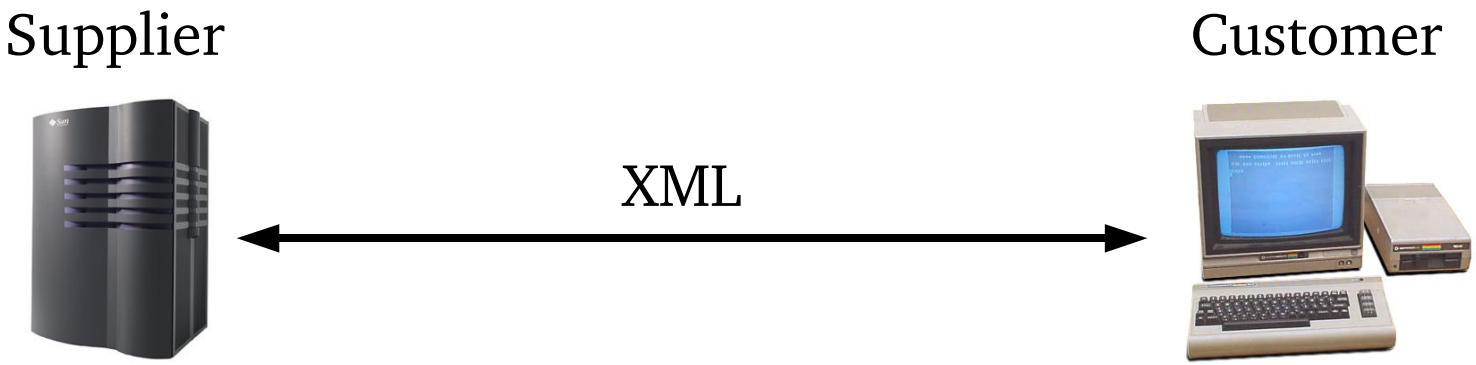
Stefan Bender
Christian Fuchs
Michael Schmidt

Jun, 03 2003

Overview

- Updating XML
- Transaction isolation in XML bases
- Detecting changes in XML documents

State of the art



- Exchange the whole document; delete the old one; insert the new one
- Exchange the whole document; detect the changes; update the old document
- Exchange only the differences between the new and old document; update the old document

Updating XML

- XML is the de facto data exchange standard
- XQuery
  - established as XML query language
  - is independent of the physical storage of the data
  - expressions are easy to understand
- But XQuery lacks the ability to change XML data
- Possible solution: Extend the XQuery language

Update operations for XQuery

- The extension should expand XQuery to a full featured query language like SQL
- The extension should not affect the properties of XQuery
- The proposed extensions to XQuery provide the following operations:
  - Delete
  - Insert / InsertBefore /InsertAfter
  - Rename / Replace
  - Sub-Update

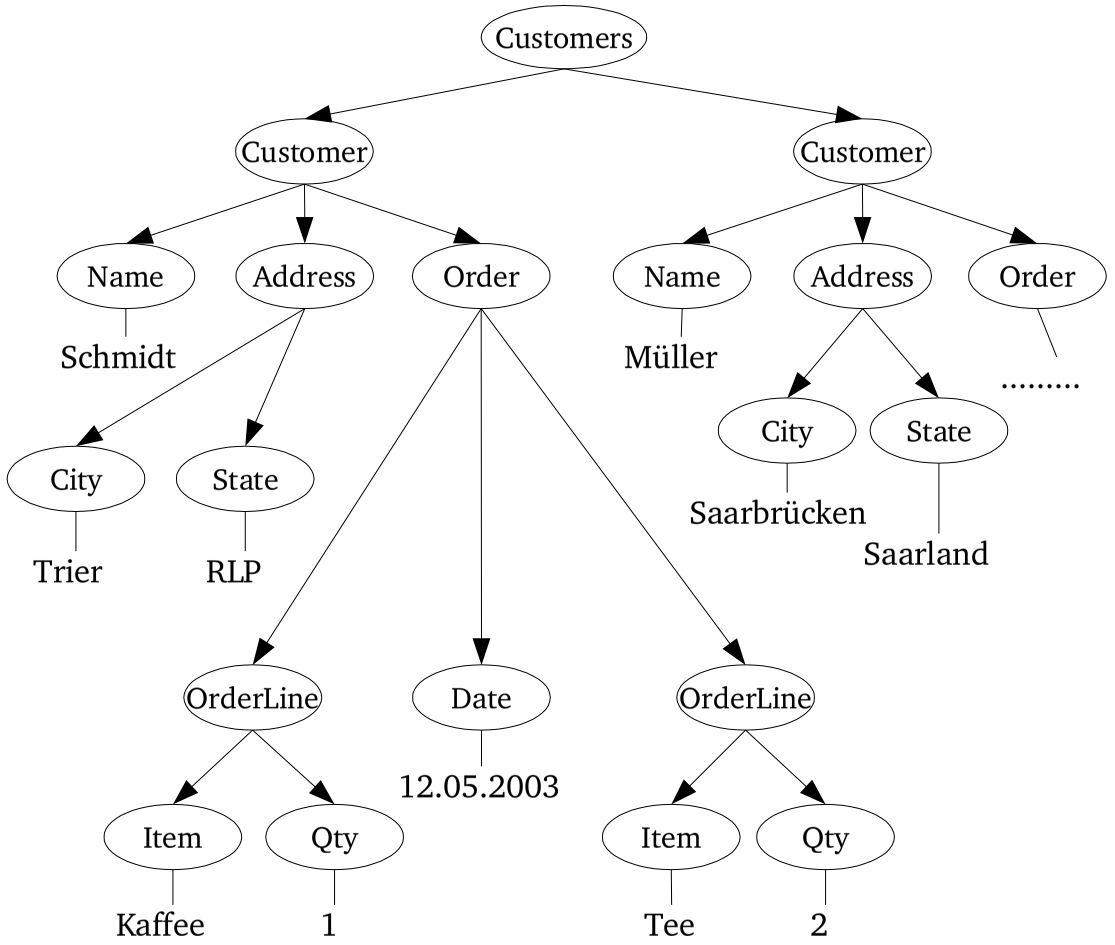
Basic structure of the XQuery extension

```
FOR
  $binding IN Xpath-expr,...

LET
  $binding := XPath-expr,...

WHERE predicate,...

UPDATE $binding { subOp {,subOP}*}
```

```
subOp:

DELETE $child |
RENAME $child |
INSERT $content [BEFORE | AFTER]$child |
REPLACE $child WITH $content |
FOR $binding IN Xpath-subexpr,...
    WHERE predicate,... subOP
```

An example document

```
<Customers>
    <Customer>
        <Name>Schmidt</Name>
        <Address>
            <City>Trier</City>
            <State>RLP</State>
        </Address>
        <Order>
            <Date>12.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
    <Customer>
        <Name>Müller</Name>
        <Address>
            <City>Saarbrücken</City>
            <State>Saarland</State>
        </Address>
        <Order>
            <Date>14.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
</Customers>
```



Two examples on XQuery extension

Delete State from Address

```
FOR
  $doc   IN document ("Customer.xml")
  $cust  IN $doc/customer,
  $addr  IN $cust/address,
  $state IN $addr/state

UPDATE $addr {

  DELETE $state

}
```

Replace every City that is called "Trier" with "Mainz"

```
FOR
  $doc  IN document ("Customer.xml")
  $cust IN $doc/customer,
  $addr IN $cust/address,
  $city IN $addr/city
WHERE $city = "Trier"

UPDATE $addr {

  REPLACE  $city
  WITH
    <appellation>
      Mainz
    </appellation>

}
```

Problems with XML updates

- There may be problems with references when an element is inserted or deleted
- A change on a node may result in an update of a whole subtree
- When dealing with a RDBS, a single update statement could result in multiple SQL statements
- After an update there may be a conflict between the document and its DTD

Problem with DTD

DTD

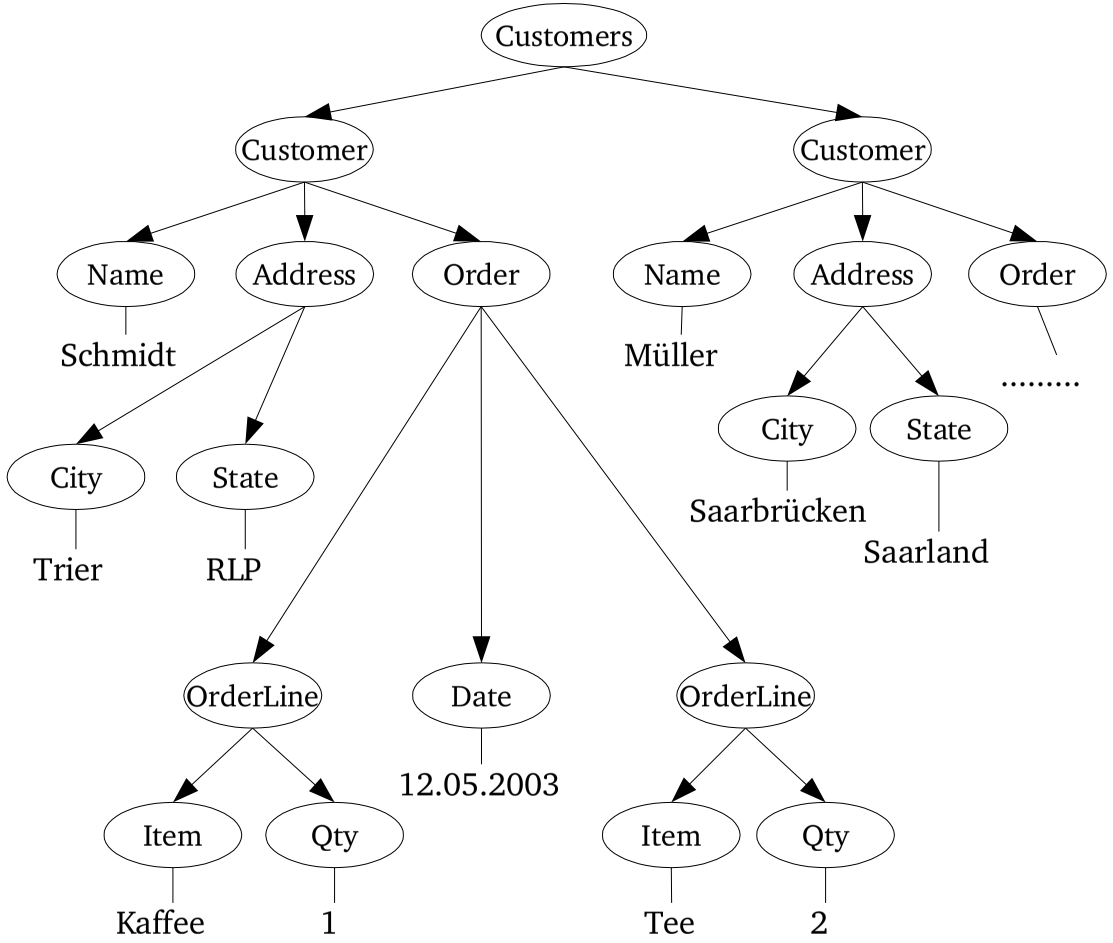
```
<!ELEMENT Customer
    (Name, Address, Order*)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (City, State)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
```

Delete State from Address

```
FOR
  $doc   IN document ("Customer.xml")
  $cust  IN $doc/customer,
  $addr  IN $cust/address,
  $state IN $addr/state

UPDATE $addr{

  DELETE $state

}
```

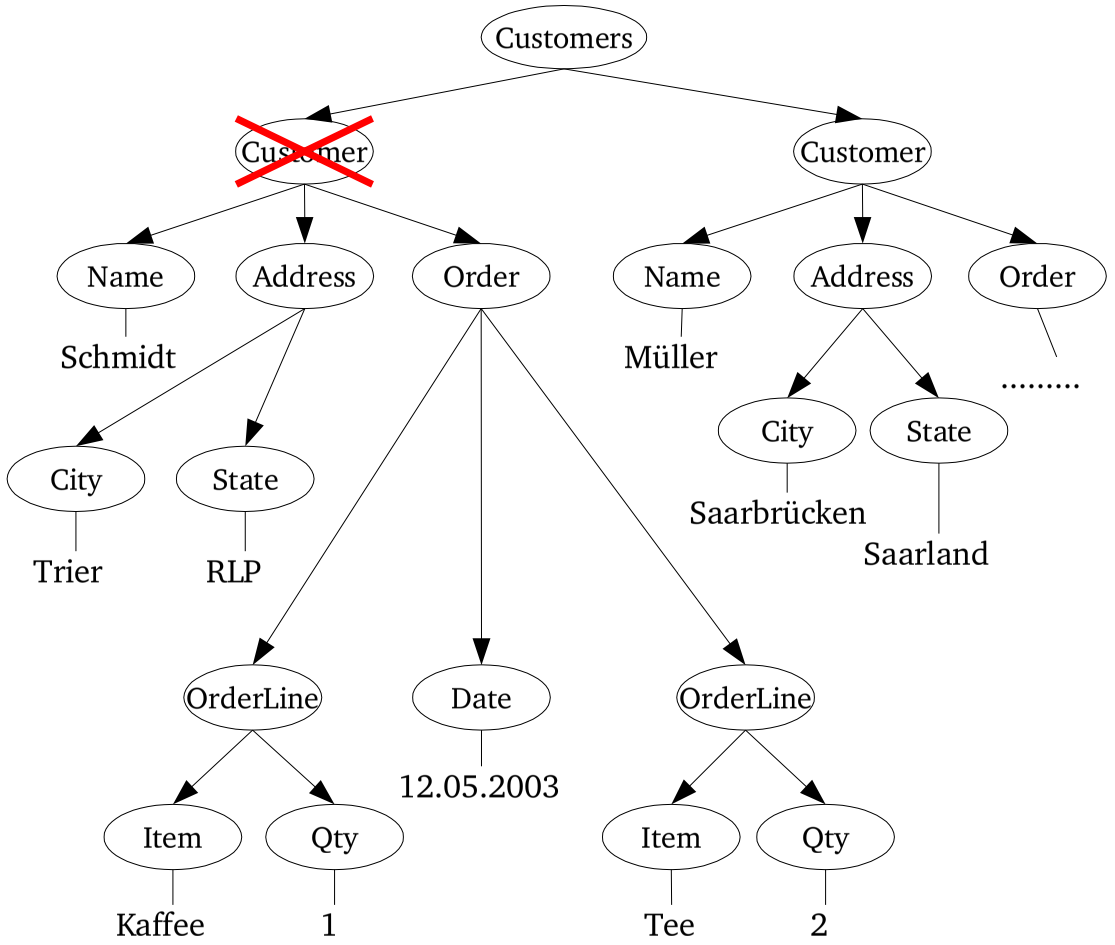
Effect of a single update

```
<Customers>
    <Customer>
        <Name>Schmidt</Name>
        <Address>
            <City>Trier</City>
            <State>RLP</State>
        </Address>
        <Order>
            <Date>12.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
    <Customer>
        <Name>Müller</Name>
        <Address>
            <City>Saarbrücken</City>
            <State>Saarland</State>
        </Address>
        <Order>
            <Date>14.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
</Customers>
```



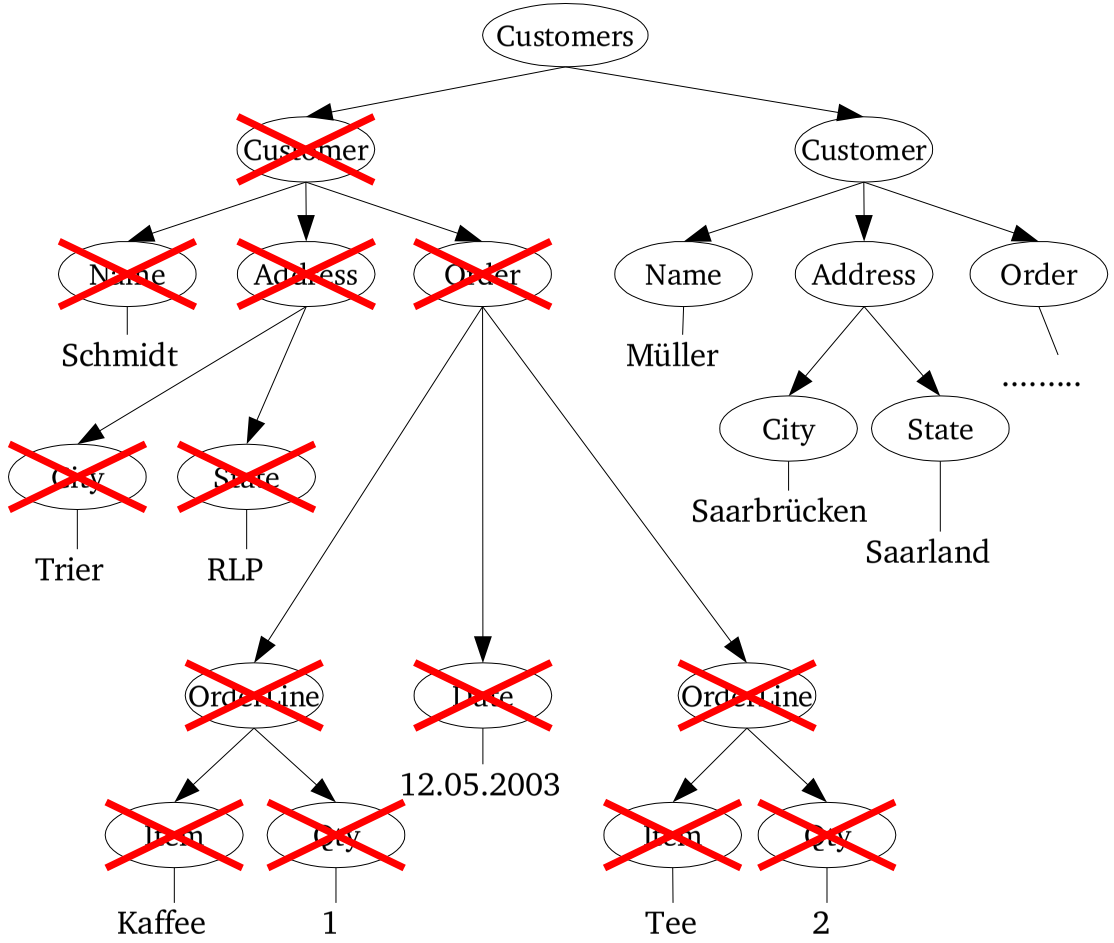
Effect of a single update

```
<Customers>
    <Customer>
        <Name>Schmidt</Name>
        <Address>
            <City>Trier</City>
            <State>RLP</State>
        </Address>
        <Order>
            <Date>12.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
    <Customer>
        <Name>Müller</Name>
        <Address>
            <City>Saarbrücken</City>
            <State>Saarland</State>
        </Address>
        <Order>
            <Date>14.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
</Customers>
```



Effect of a single update

```
<Customers>
    <Customer>
        <Name>Schmidt</Name>
        <Address>
            <City>Trier</City>
            <State>RLP</State>
        </Address>
        <Order>
            <Date>12.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
    <Customer>
        <Name>Müller</Name>
        <Address>
            <City>Saarbrücken</City>
            <State>Saarland</State>
        </Address>
        <Order>
            <Date>14.05.2003</Date>
            <OrderLine> ...
        </Order>
    </Customer>
</Customers>
```



Storing XML documents in a relational database system

Example DTD

```
<!ELEMENT Customer
    (Name, Address, Order*)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (City, State)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Order
    (Date, OrderLine*)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT OrderLine( Item, qty)>
<!ELEMENT Item (#PCDATA)>
<!ELEMENT qty (#PCDATA)>
```

Table: Customer

ID	*ParentID*	*Name*	*City*	*State*
1	0	Schmidt	Trier	RLP
2	0	Müller	Saarbrücken	Saarland

Table: Order

ID	*ParentID*	*Date*	*OrderLine*
1	1	12.05.03	1
2	1	12.05.03	2
3	2	14.05.03	3

Table: OrderLine

ID	*ParentID*	*Item*	*QTY*
1	1	Kaffee	1
2	1	Tee	2
3	2	Kaffee	3

XML update in RDBS

- A single update (like delete) can affect multiple relations in the database
- Problem
  - Correctness
  - Efficiency
- Solution
  - Cascading Delete
  - Trigger
  - Access Support Relations

Conclusion

- Updating XML is still immature
- Extending XQuery seems reasonable
- Further investigation needs to be done in this area
- What if two applications try to modify the same element?

Isolating in XML Bases

- Isolation is a vital part for the correctness of databases. Any action on the data must be prevented from working on intermediary data generated by other actions.
- Transaction:
  Sequence of Operations on one or more databases, with the following guaranteed (ACID) attributes

ACID attributes

- **Atomicity:** All changes of a transaction are commited entirely or not at all
- **Consistency:** A transaction transforms data from one valid state to another valid state
- **Isolation:** Every transaction works on the data as if there were no parallel transactions
- **Durability:** Commited changes are persistent, even in case of a hard- or software failure

Operations

- Classification

  By action: *reading or writing*

  By domain: *content* or *structure*

- Representative set of structure operations

  sd: select document root

  nthF: retrieve the n-th child in the child list

  nthB: same, but counting backwards

  del: delete node

  ins: insert node

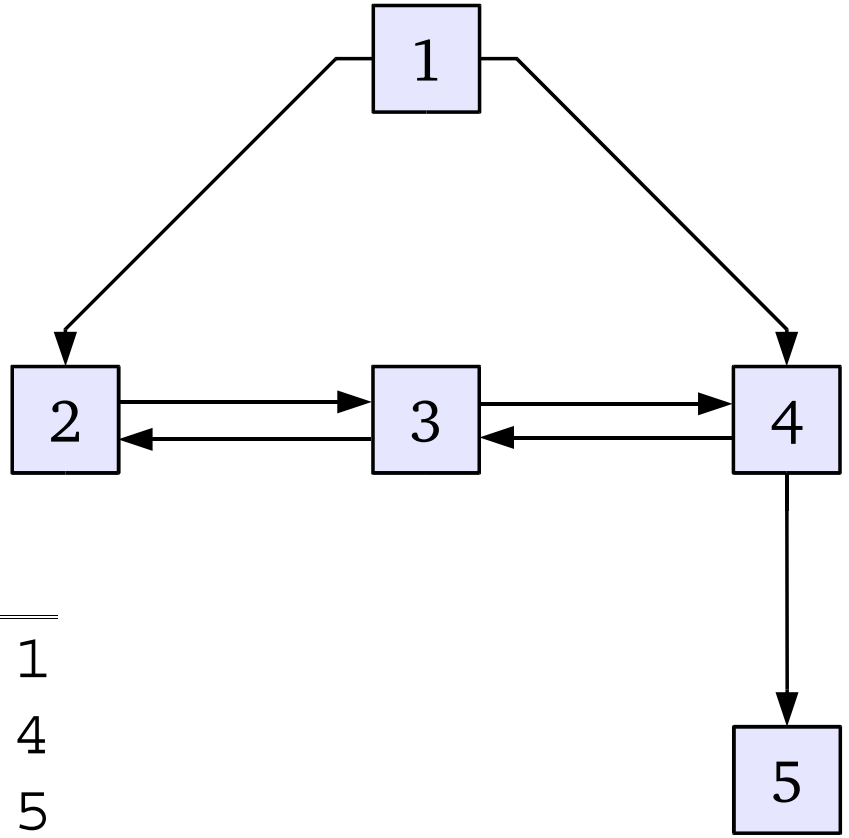
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1	T_2
`sd  => 1`	`sd  => 1`
`nthF(2) => 3`	`nthB(1) => 4`
`del  Delete 3`	`nthF(1) => 5`

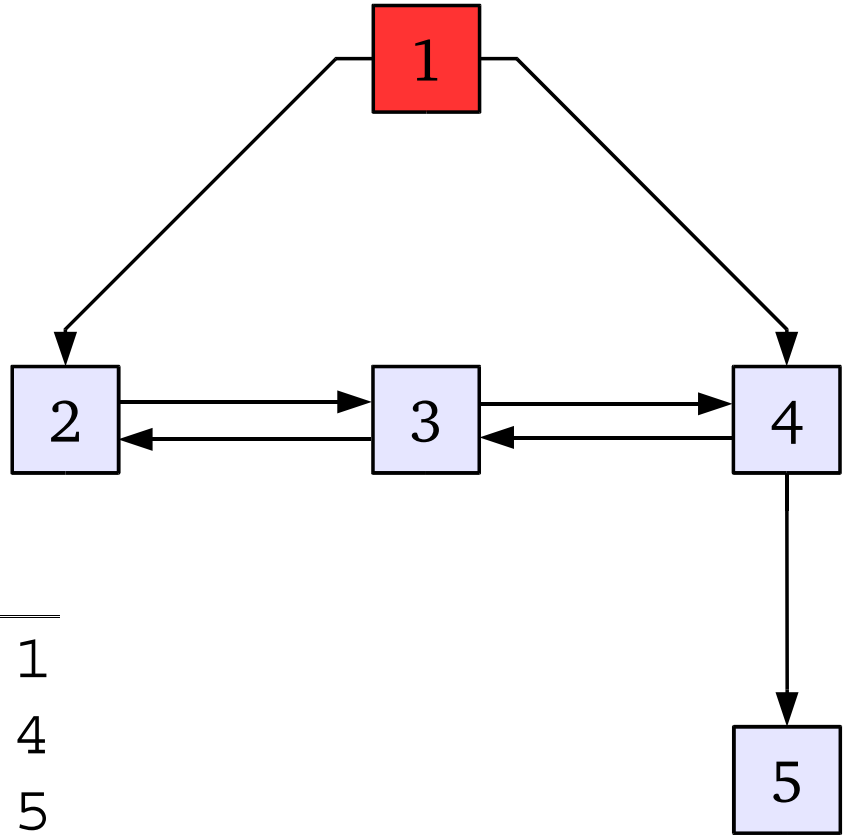
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1		T_2	
sd	=> 1	sd	=> 1
nthF(2)	=> 3	nthB(1)	=> 4
del	Delete 3	nthF(1)	=> 5

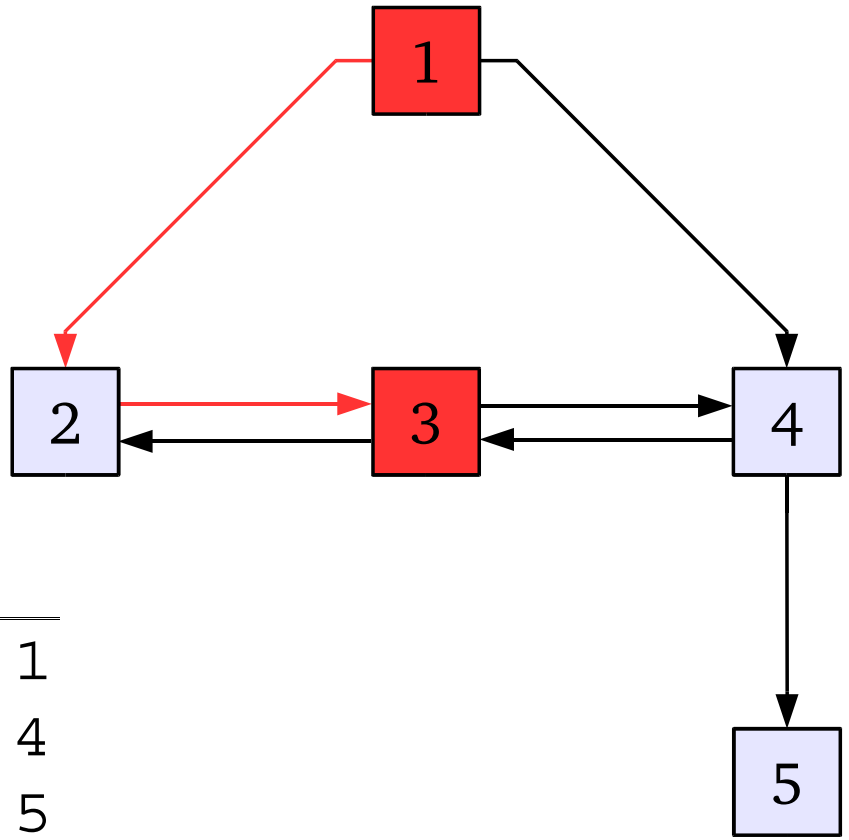
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1		T_2	
sd	=> 1	sd	=> 1
nthF(2)	=> 3	nthB(1)	=> 4
del	Delete 3	nthF(1)	=> 5

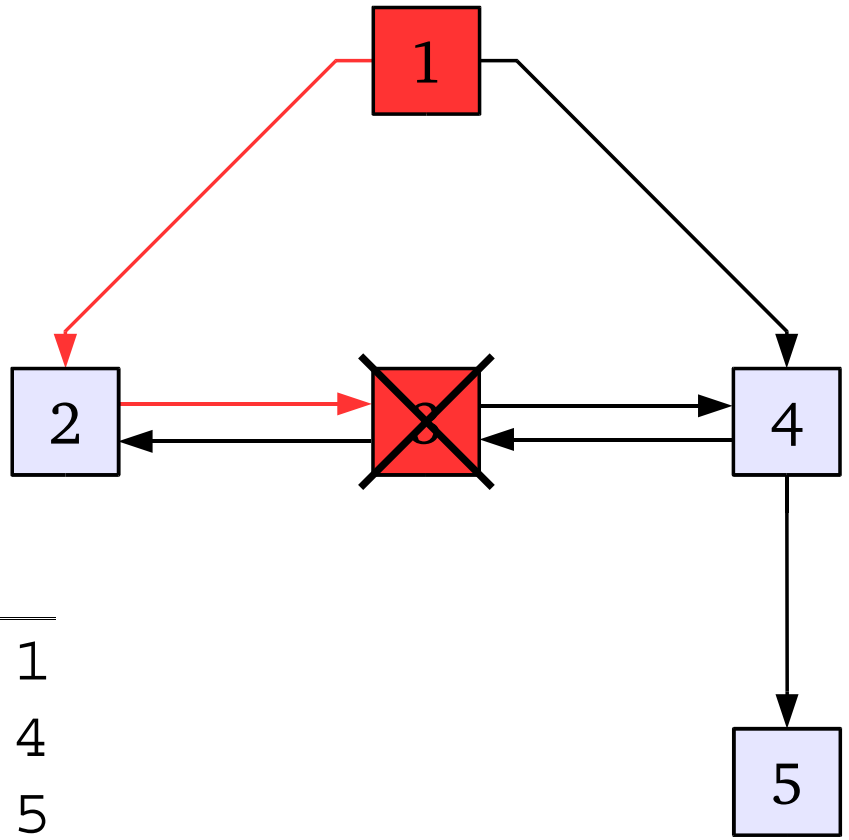
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

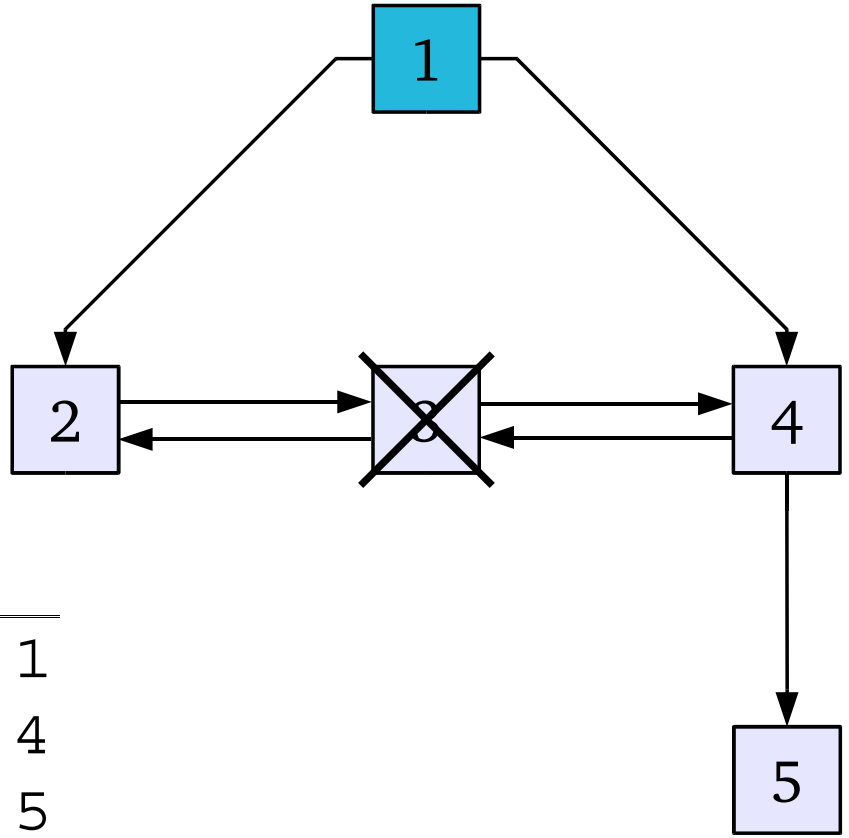
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1		T_2	
sd	=> 1	sd	=> 1
nthF(2)	=> 3	nthB(1)	=> 4
del	Delete 3	nthF(1)	=> 5

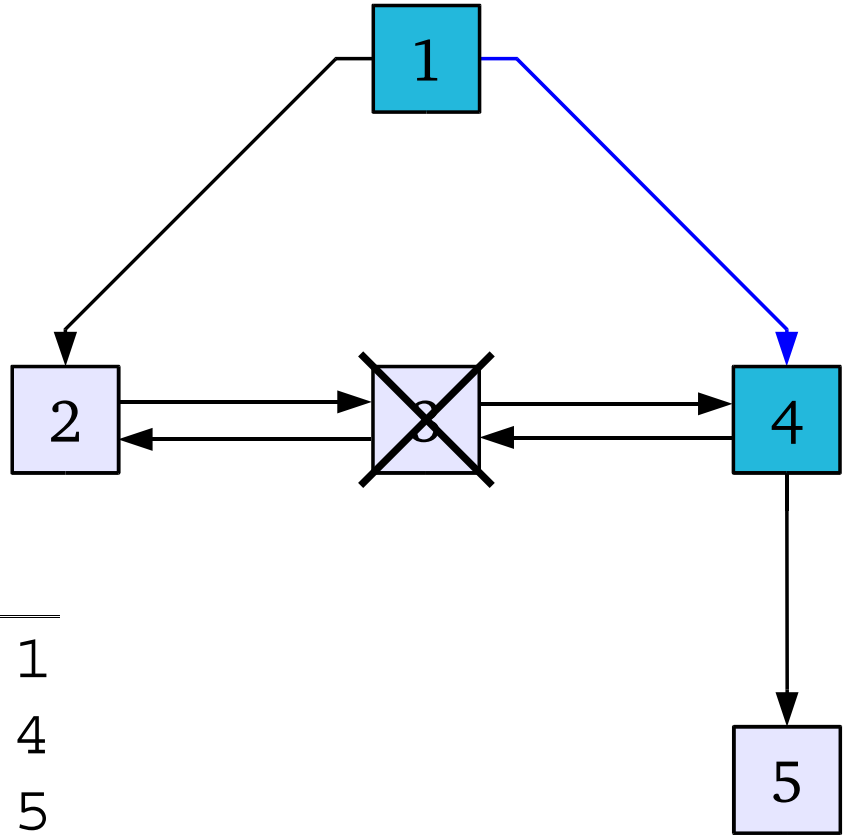
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1	T_2
`sd => 1`	`sd => 1`
`nthF(2) => 3`	`nthB(1) => 4`
`del Delete 3`	`nthF(1) => 5`

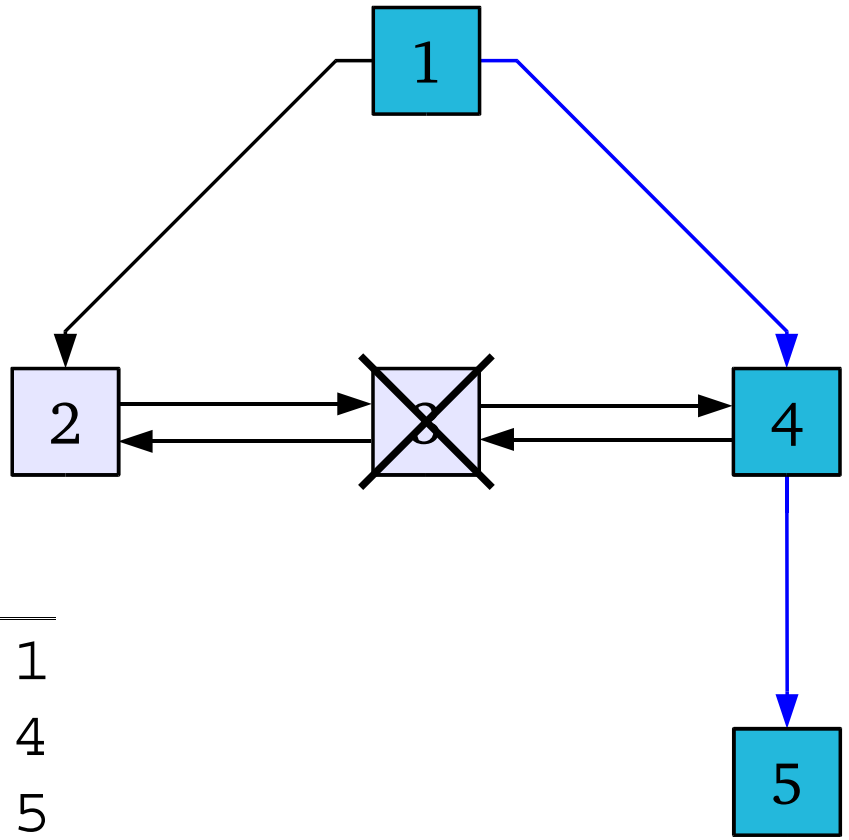
Example

- 4 Pointers per node

  First and last child

  Left and right sibling

- 2 Transactions



T_1	T_2
`sd => 1`	`sd => 1`
`nthF(2) => 3`	`nthB(1) => 4`
`del Delete 3`	`nthF(1) => 5`

Core protocols for synchronizing read and write operations

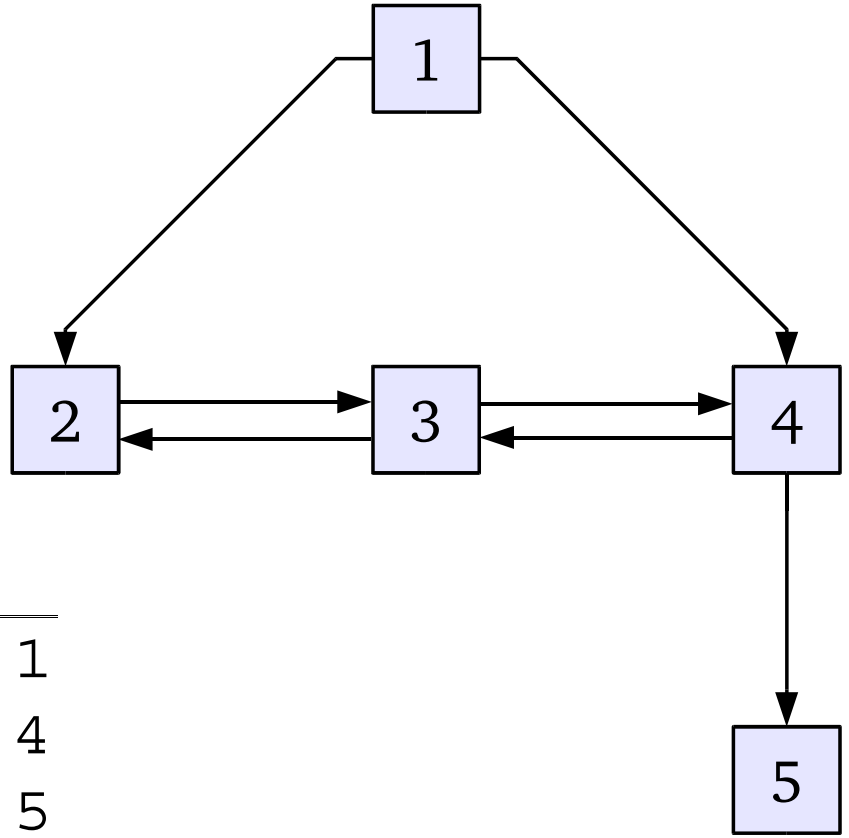
- Two phase locking based protocols
  - Doc2PL
  - Node2PL
  - NO2PL
  - OO2PL
- Timestamp-based protocol
- Dynamic commit ordering

Two phase locking

- Locks are associated with each data item
- A transaction has to acquire a *shared* (S) lock for reading a data item or an *exclusive* (X) lock to modify it
- An exclusive lock conflicts with all other locks
- All locks are acquired before any lock is released
- A transaction holds all locks until completion

Doc2PL

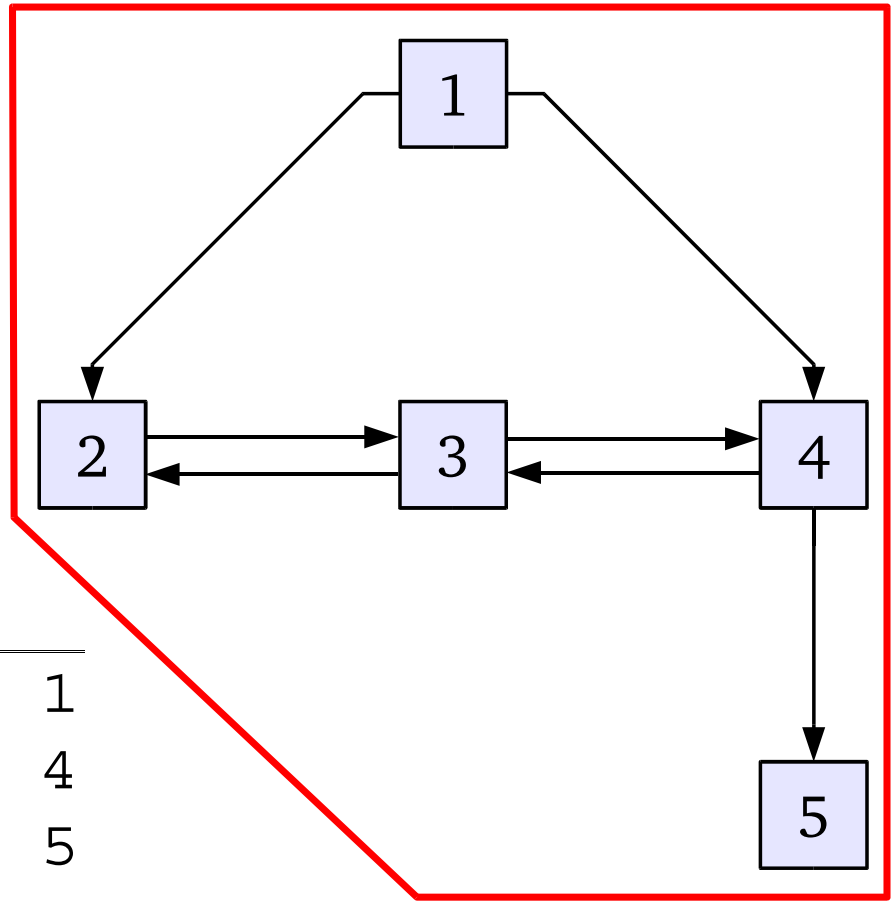
- lock the whole document



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

Doc2PL

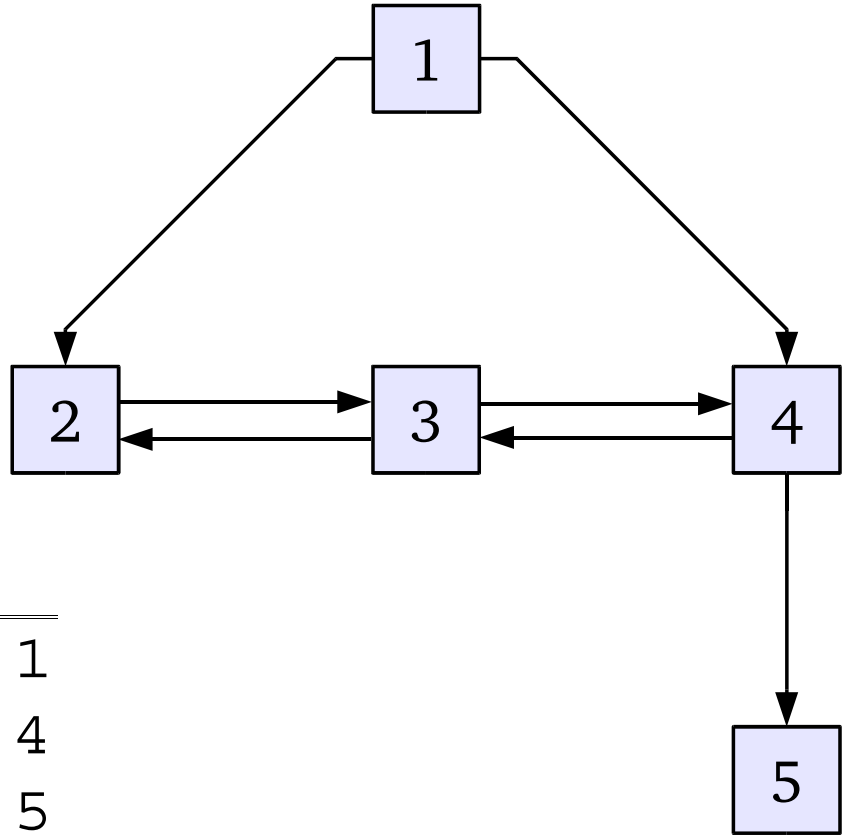
- lock the whole document
- T_2 has to wait until T_1 commits



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

Node2PL

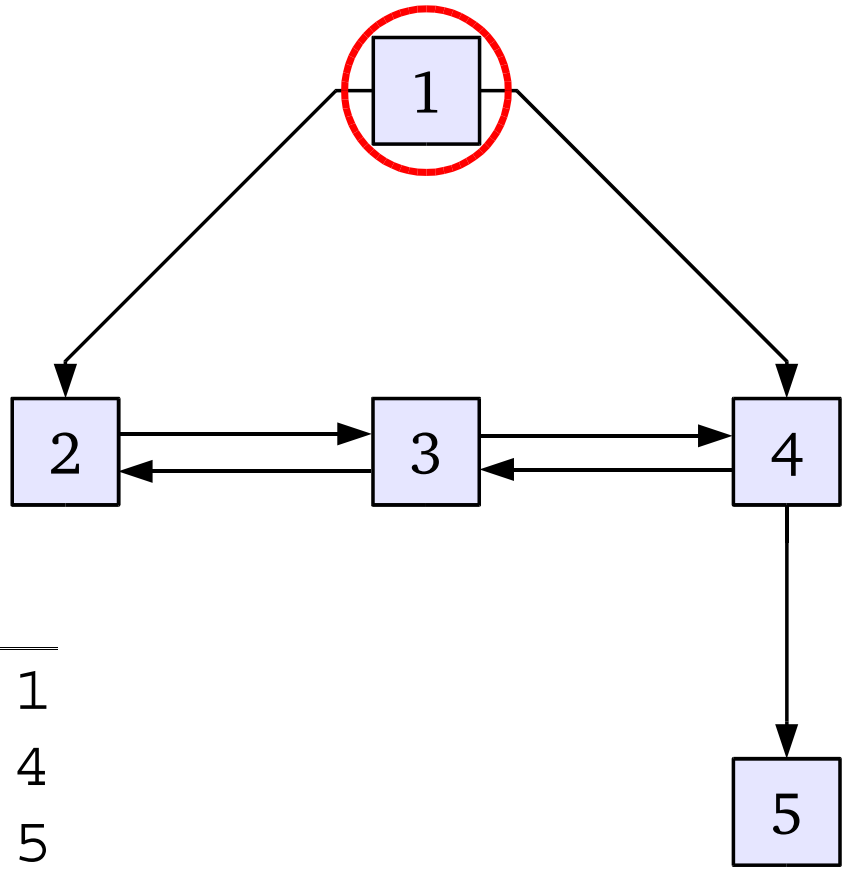
- lock parent nodes



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

Node2PL

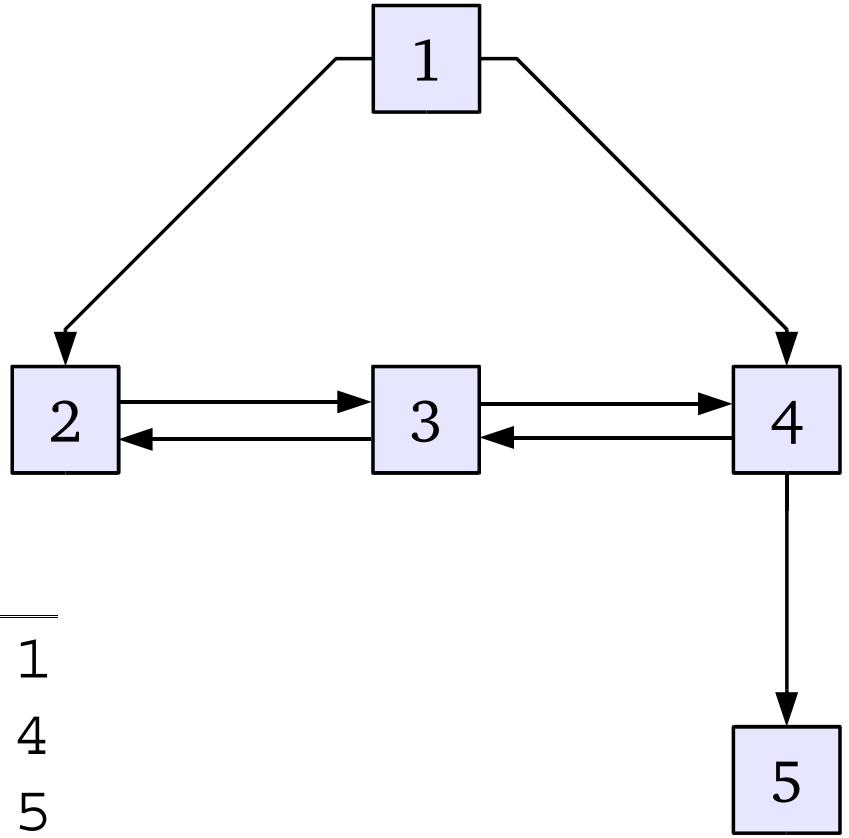
- lock parent nodes
- T_2 has to wait until T_1 commits



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

NO2PL

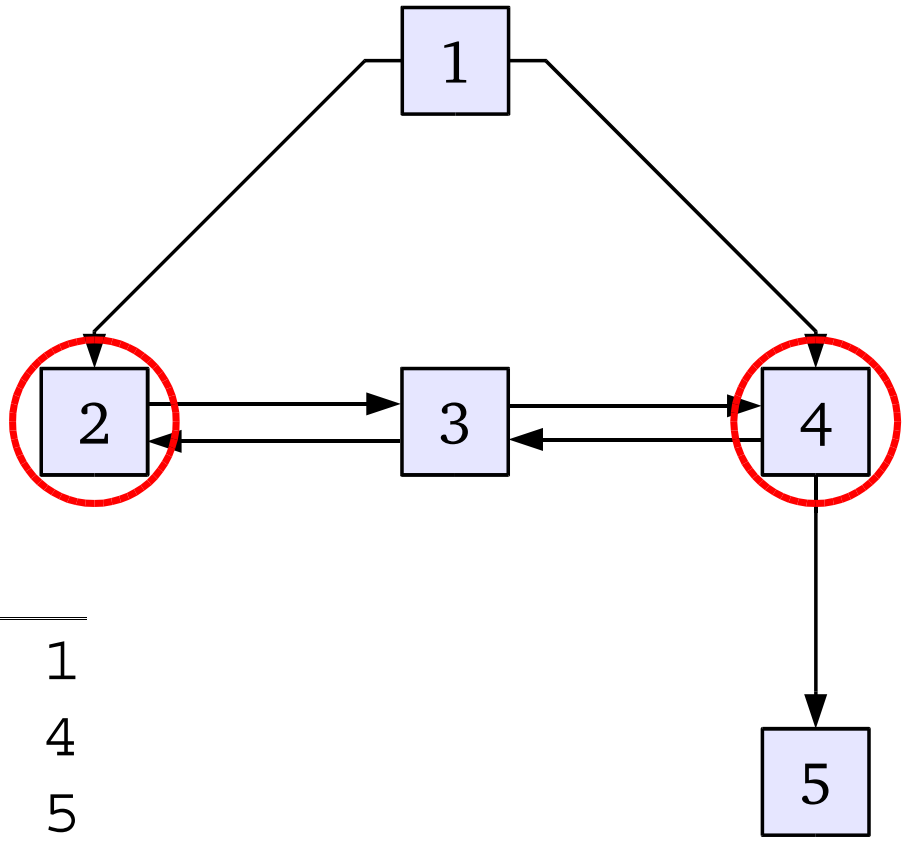
- lock nodes whose pointers are traversed or modified



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

NO2PL

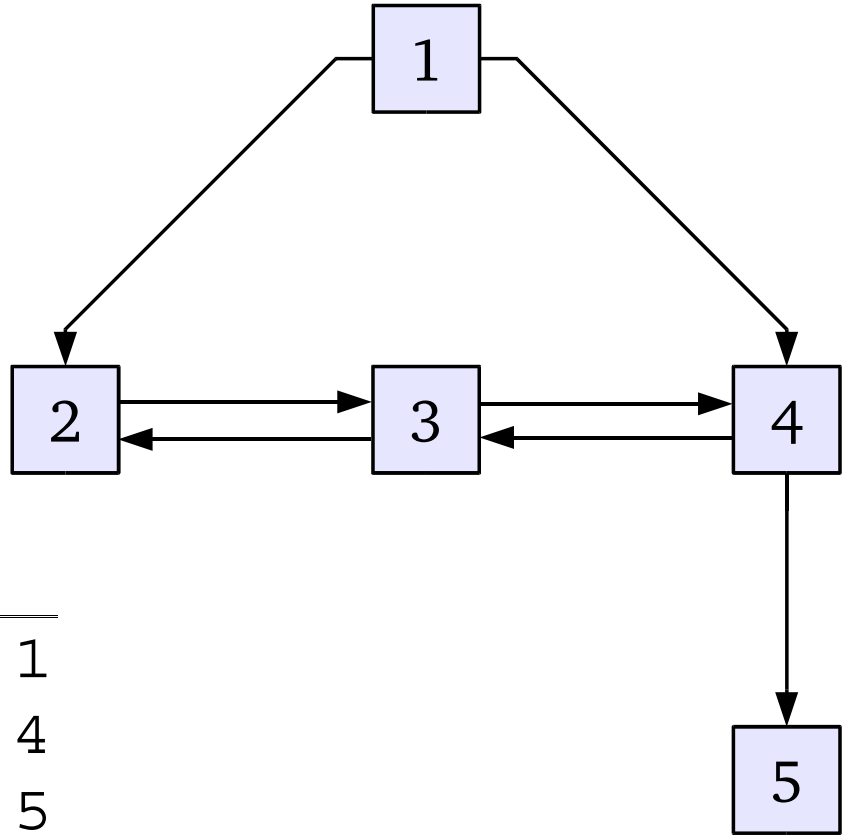
- lock nodes whose pointers are traversed or modified
- T_2 has to wait until T_1 commits



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

OO2PL

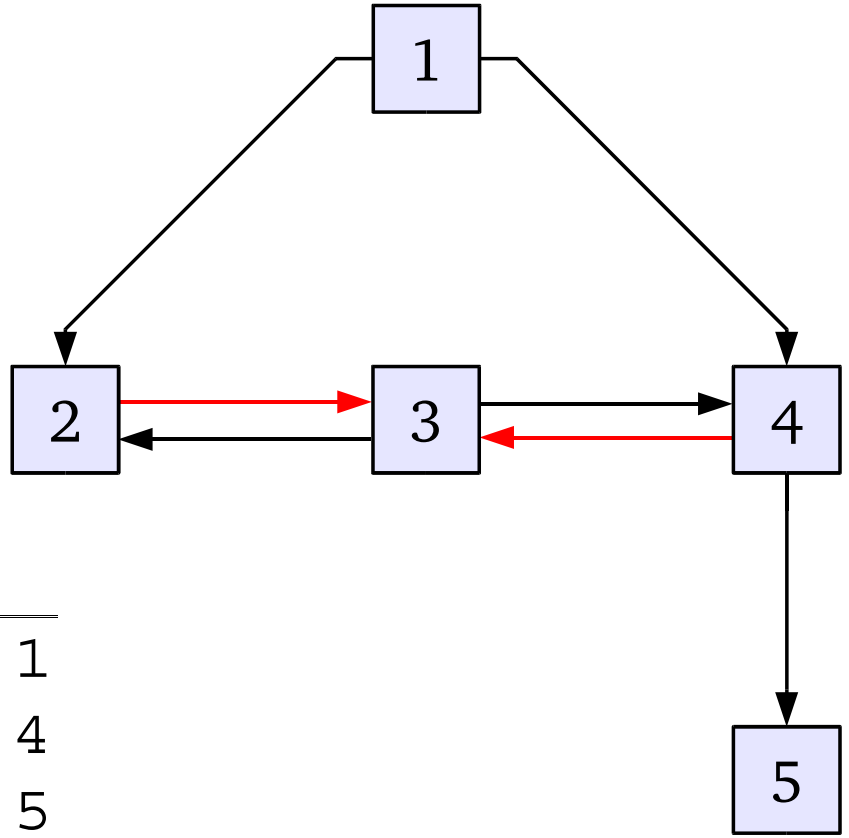
- lock only pointers



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

OO2PL

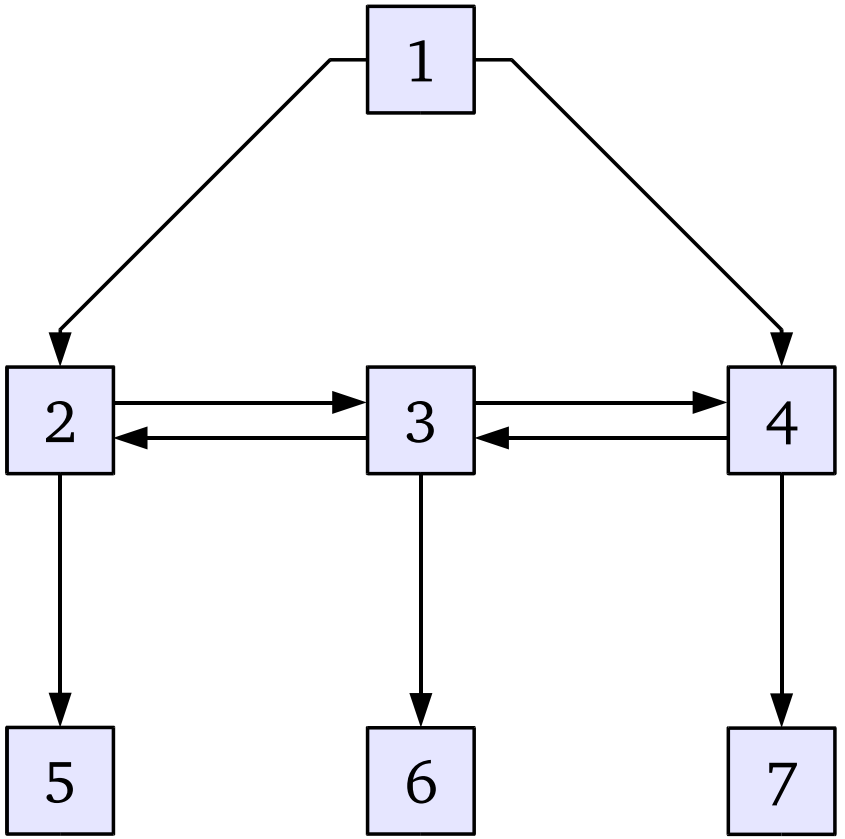
- lock only pointers
- T_2 and T_1 can be executed at the same time



T_1	T_2
sd => 1	sd => 1
nthF(2) => 3	nthB(1) => 4
del Delete 3	nthF(1) => 5

Another example

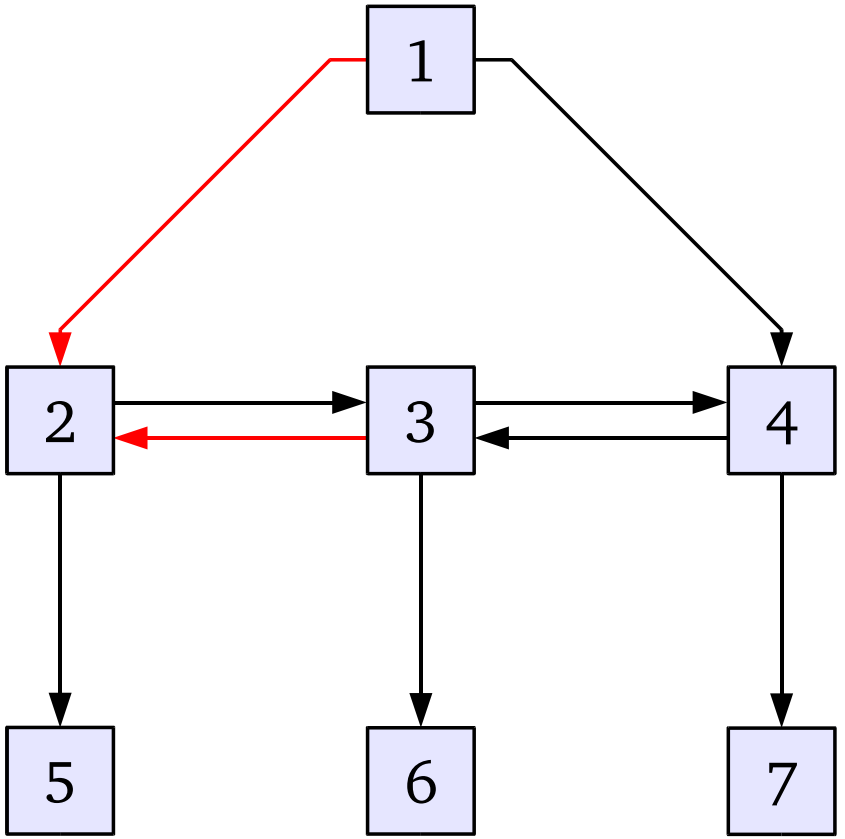
- Is OO2PL enough?



T_1	T_2
sd => 1	sd => 1
nthF(1) => 2	nthB(1) => 4
del Delete 2	del Delete 4
sd => 1	sd => 1
nthB(1) => 4	nthF(1) => 3
nthF(1) => 7	nthF(1) => 6

Another example

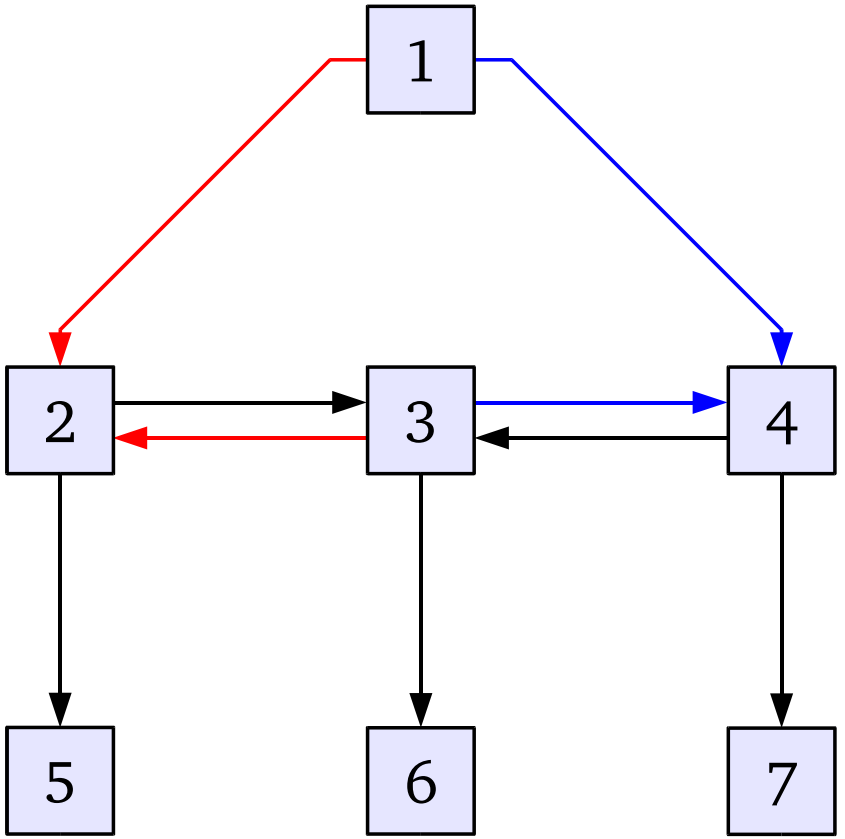
- Is OO2PL enough?



T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6

Another example

- Is OO2PL enough?
- In this situation OO2PL leads to a deadlock!



T_1	T_2
sd => 1	sd => 1
nthF(1) => 2	nthB(1) => 4
del Delete 2	del Delete 4
sd => 1	sd => 1
nthB(1) => 4	nthF(1) => 3
nthF(1) => 7	nthF(1) => 6

Timestamp-based protocol

- Basic idea is to implicitly keep multiple versions of the document
- Use marks to denote deleted respectively inserted nodes
- Eliminate marks and remove deleted nodes from document when the transaction commits
- Assign every transaction a unique timestamp
- Operations of a transaction inherit timestamp

Timestamp-based protocol

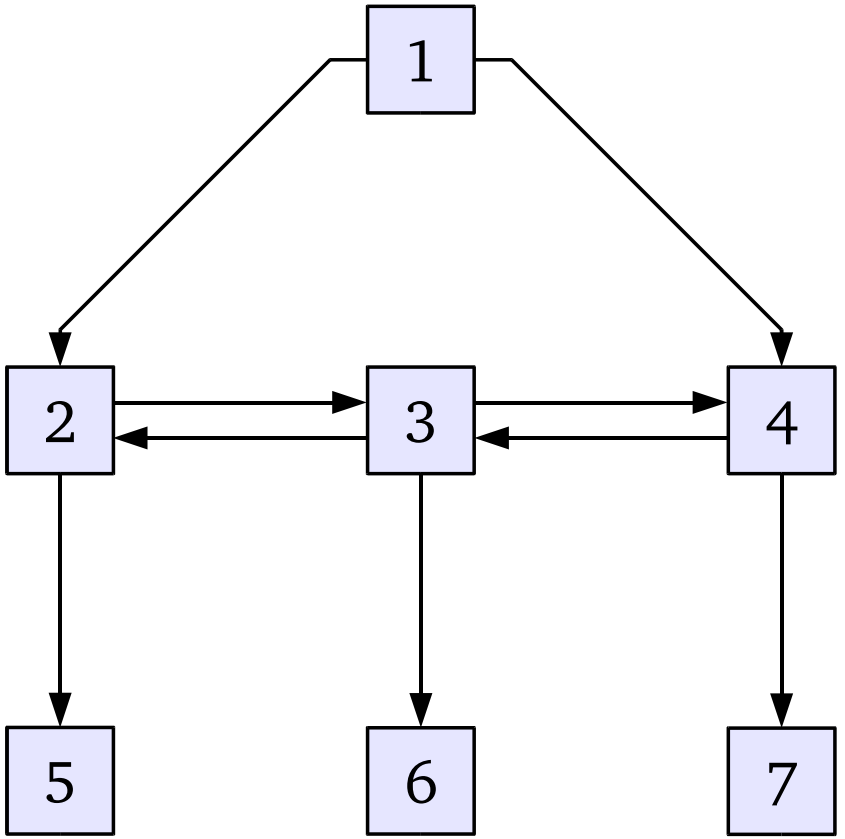
op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible
T	I	Not possible	Not possible
D	I	Not possible	Not possible
I	I	Not possible	Not possible

- Operations
  - op_F finished
  - op_S scheduled
- Actions
  - T Traverse
  - D Delete
  - I Insert
- Relation $<_{TS}$ denotes the timestamp order

Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

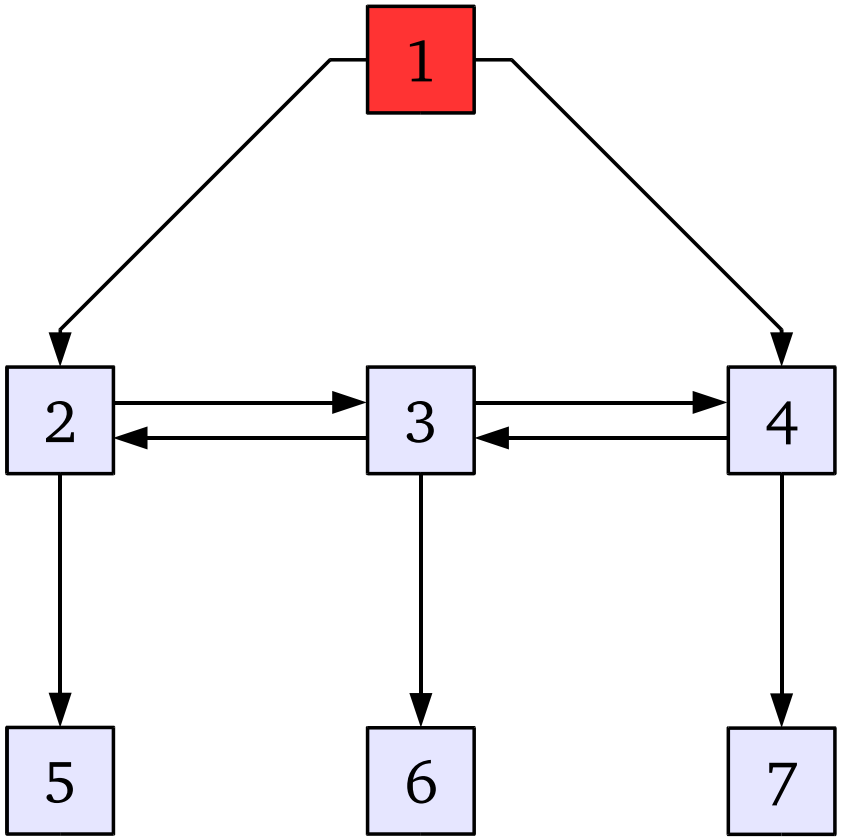
T_1	T_2
`sd => 1`	`sd => 1`
`nthF(1) => 2`	`nthB(1) => 4`
`del Delete 2`	`del Delete 4`
`sd => 1`	`sd => 1`
`nthB(1) => 4`	`nthF(1) => 3`
`nthF(1) => 7`	`nthF(1) => 6`



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

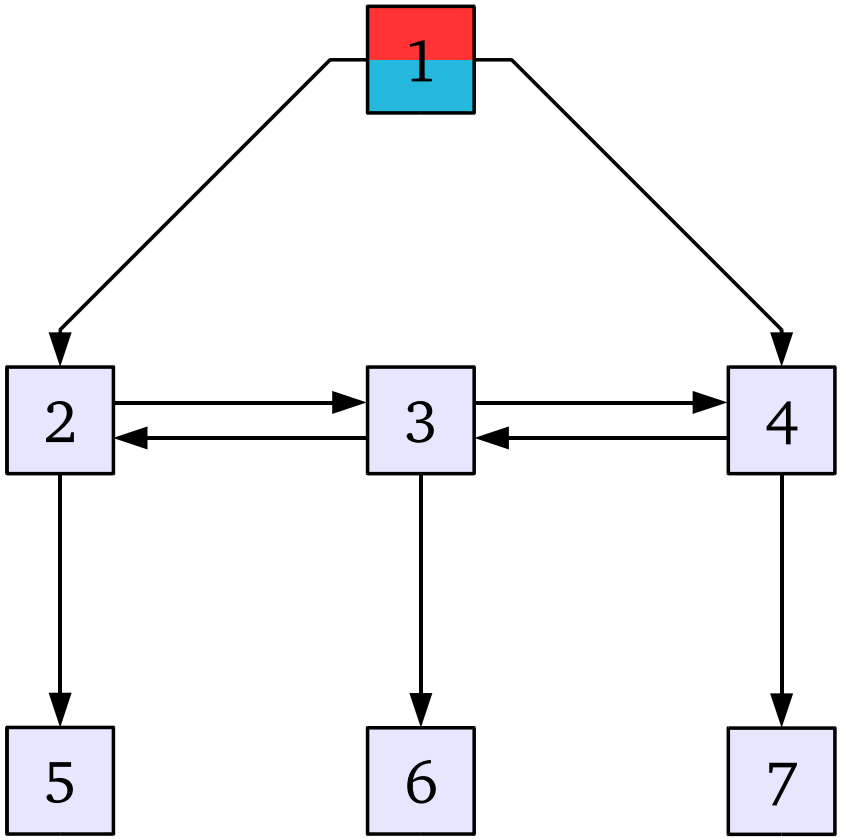
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

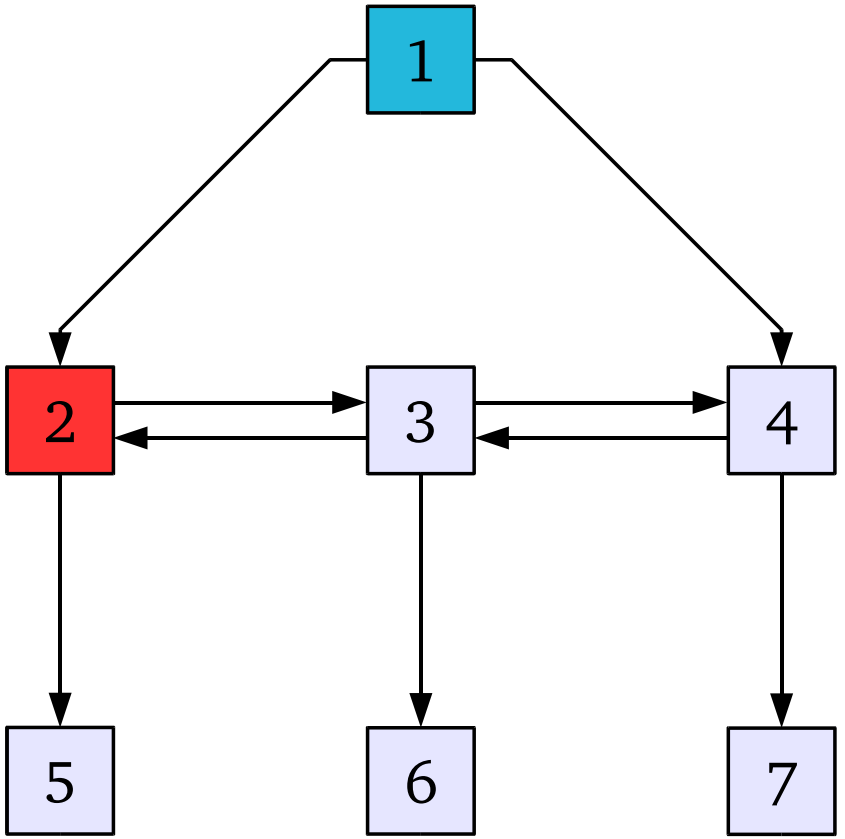
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

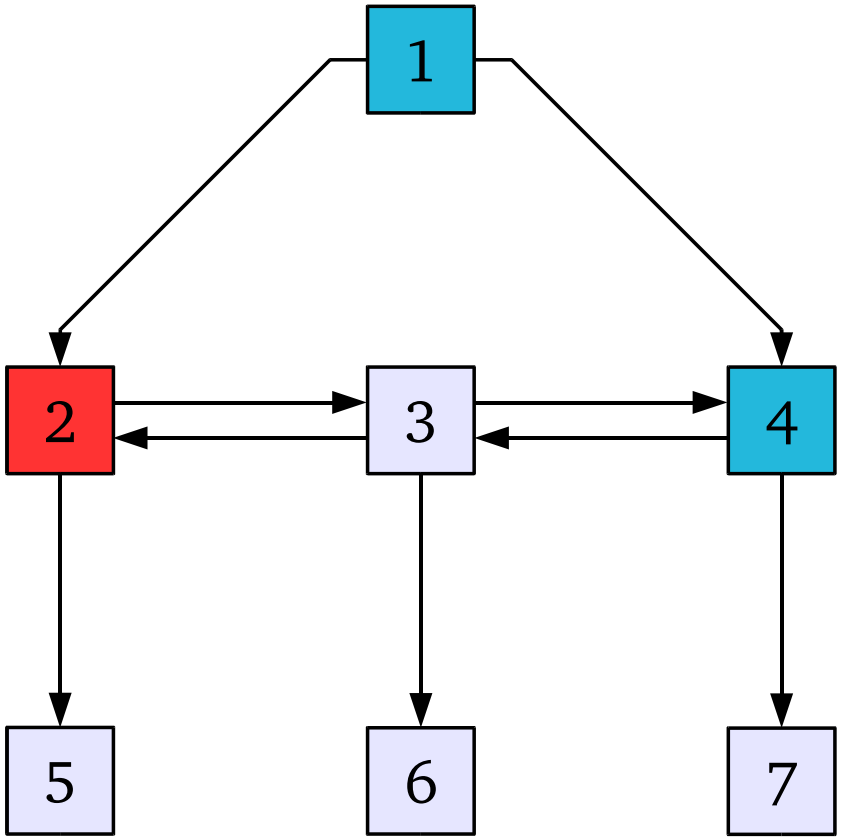
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

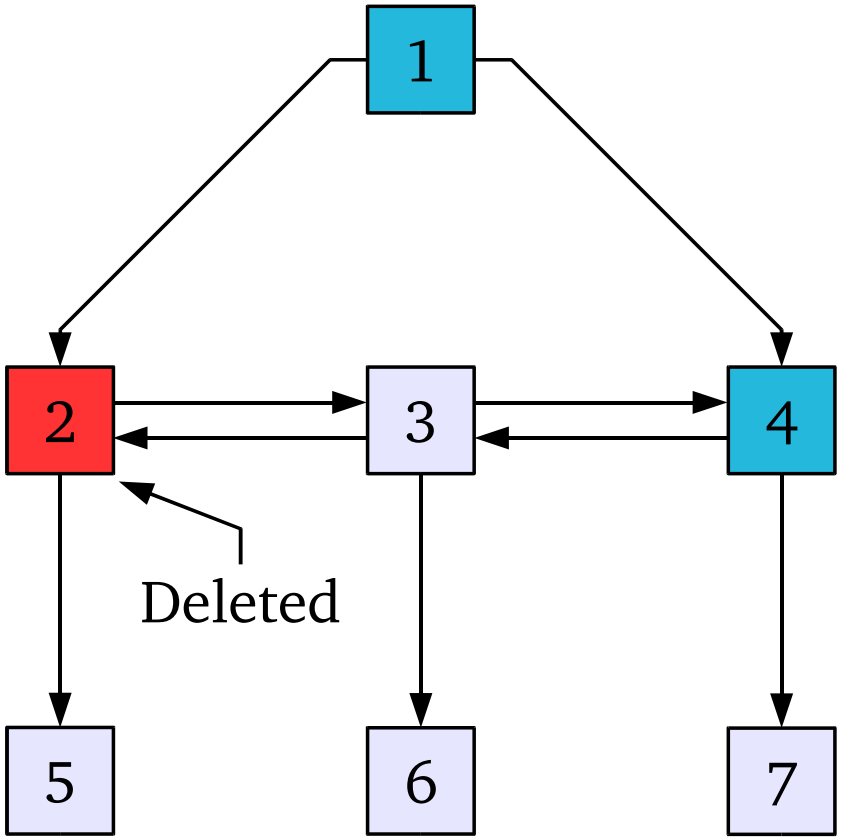
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

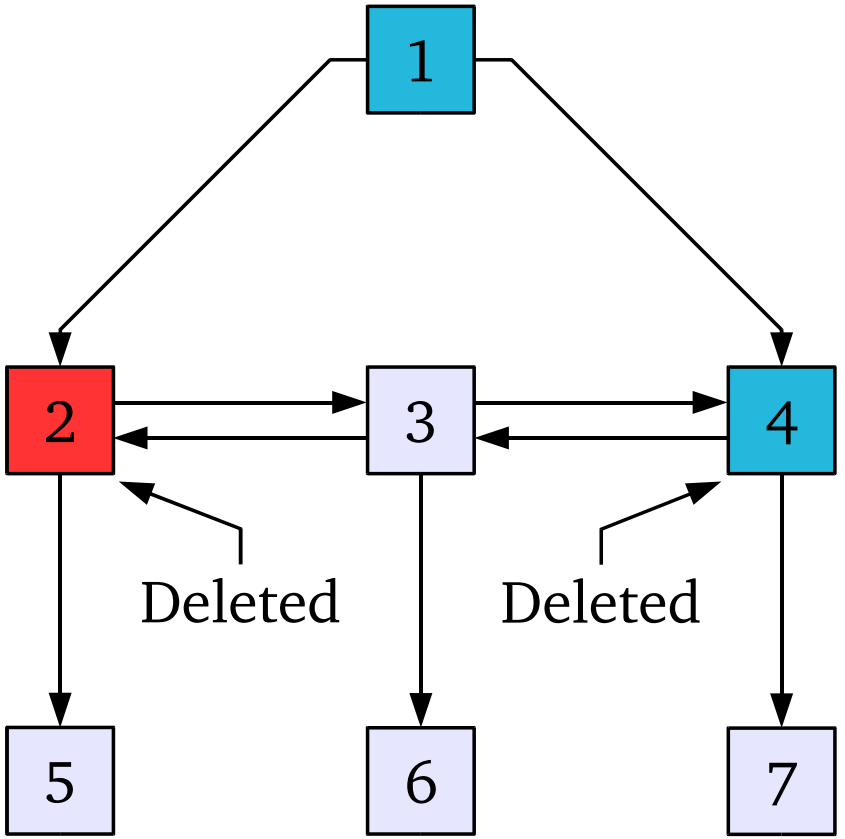
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

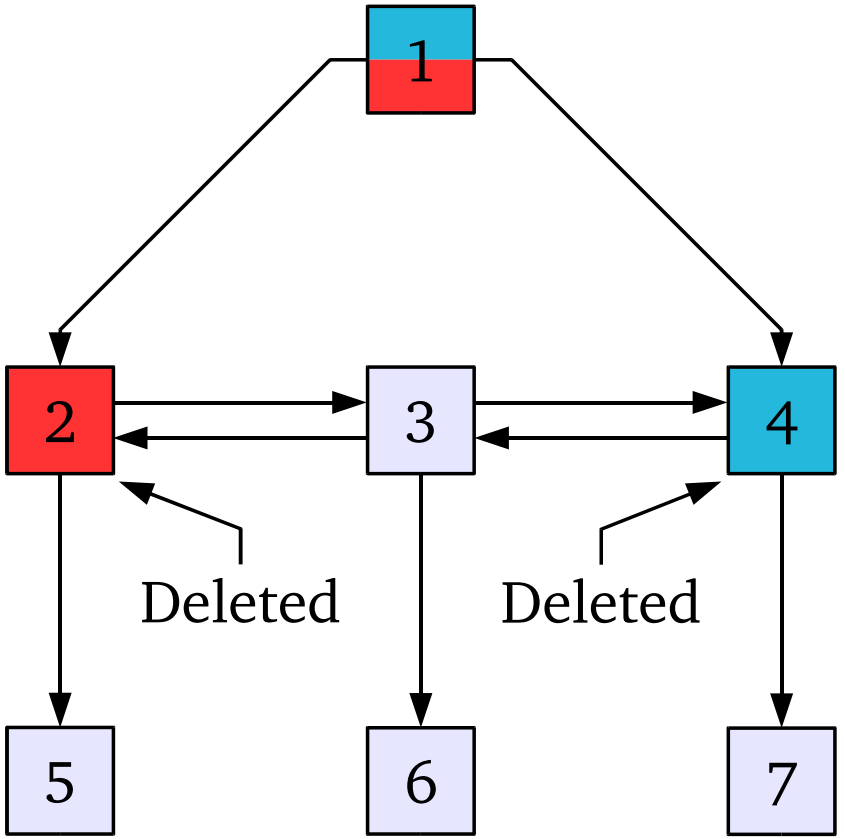
T_1	T_2
`sd  => 1`	`sd  => 1`
`nthF(1) => 2`	`nthB(1) => 4`
`del  Delete 2`	`del  Delete 4`
`sd  => 1`	`sd  => 1`
`nthB(1) => 4`	`nthF(1) => 3`
`nthF(1) => 7`	`nthF(1) => 6`



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

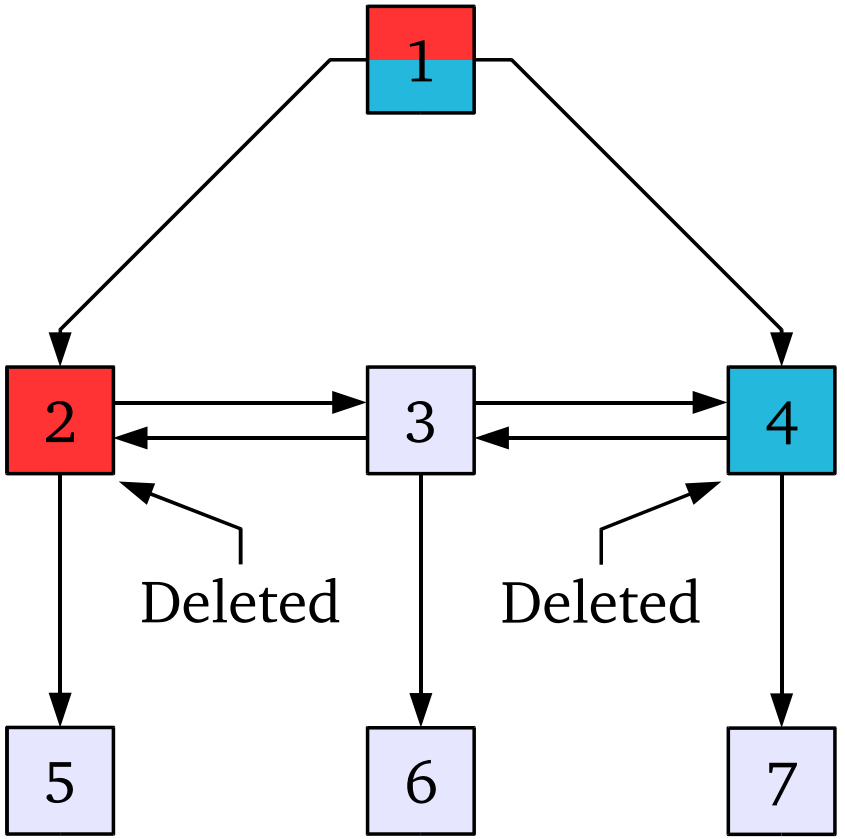
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

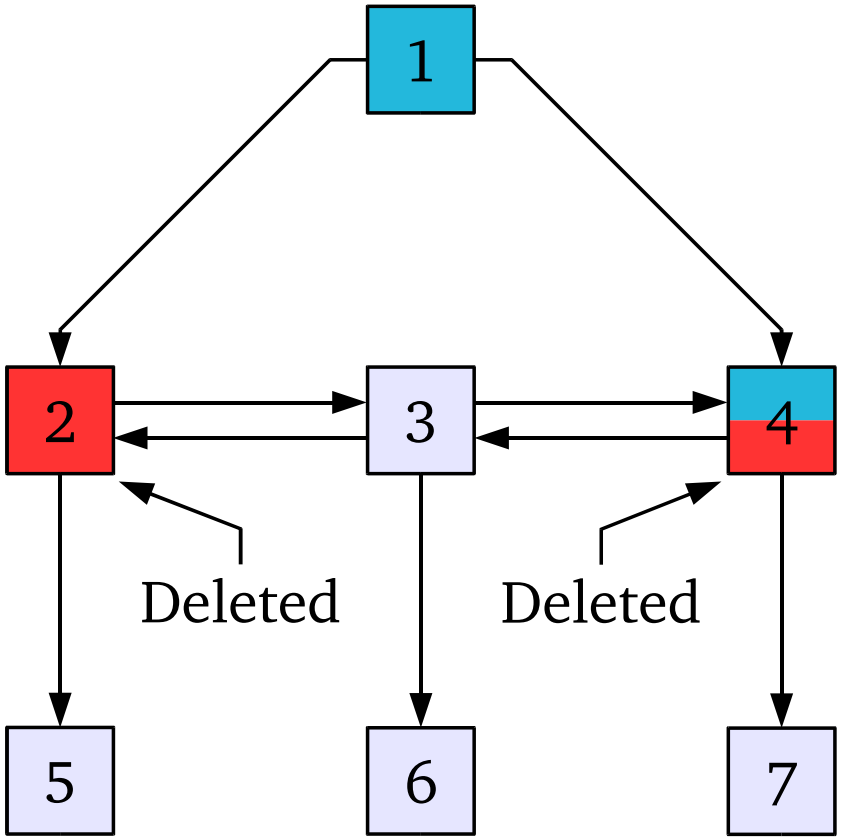
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

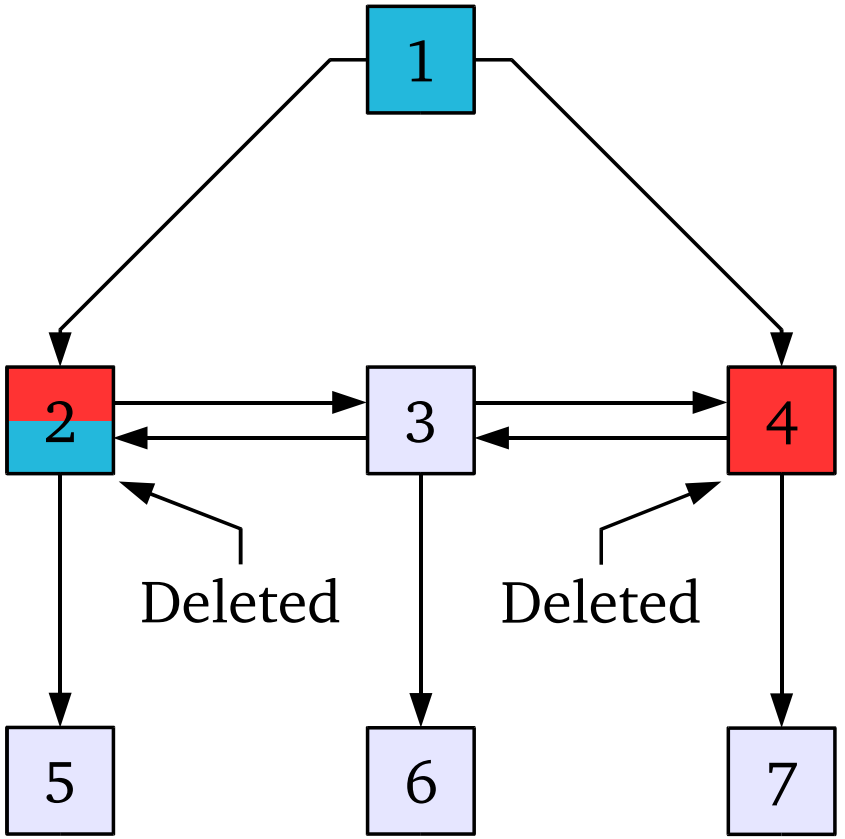
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

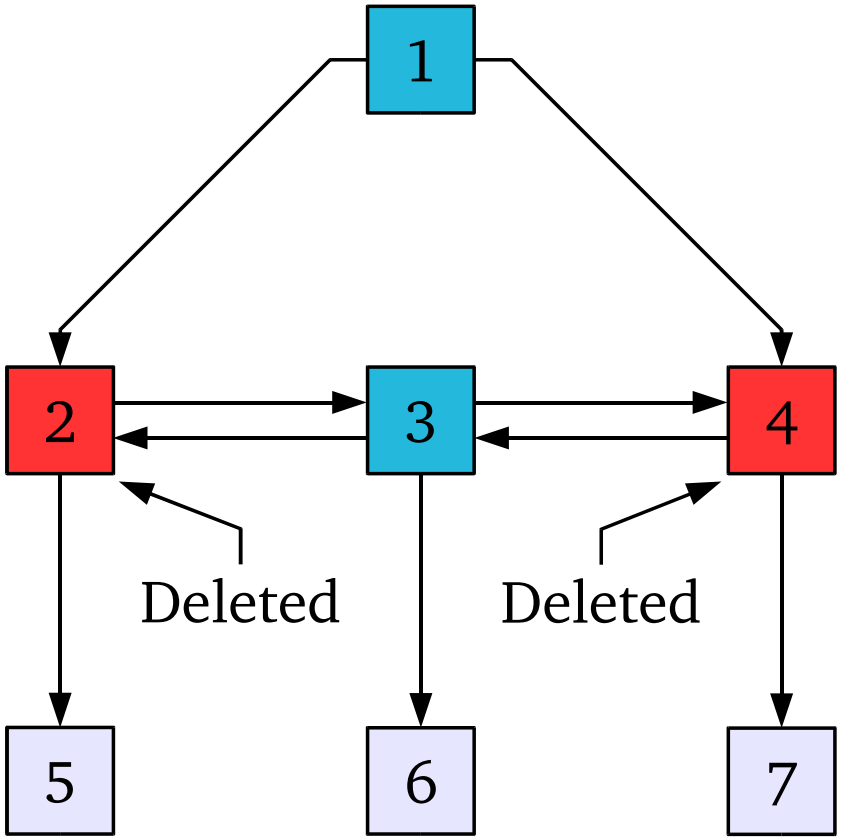
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

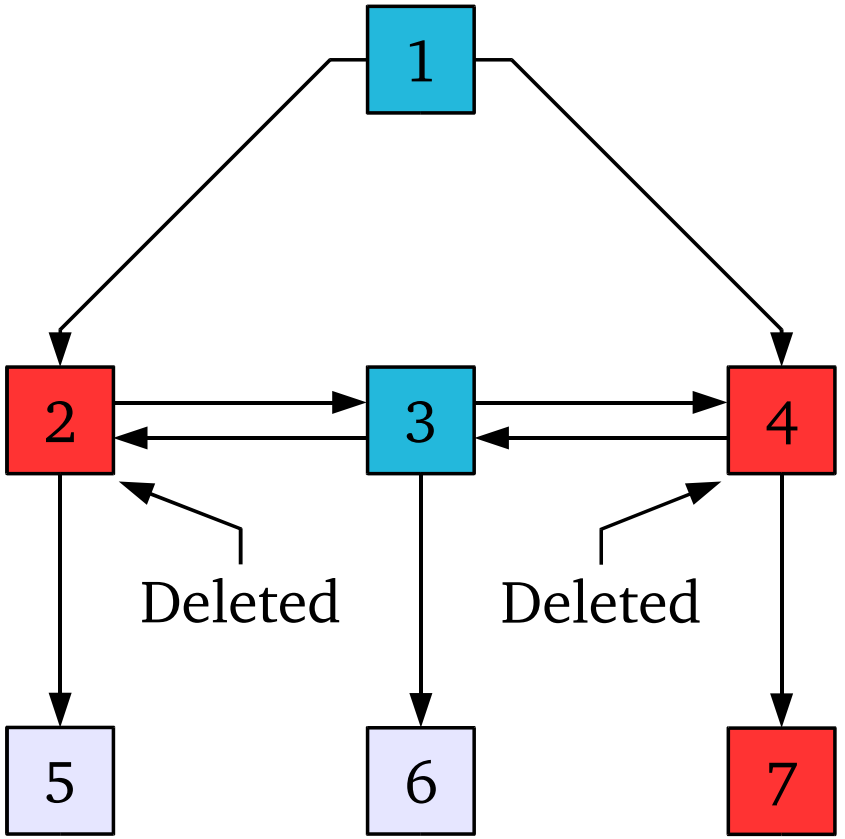
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

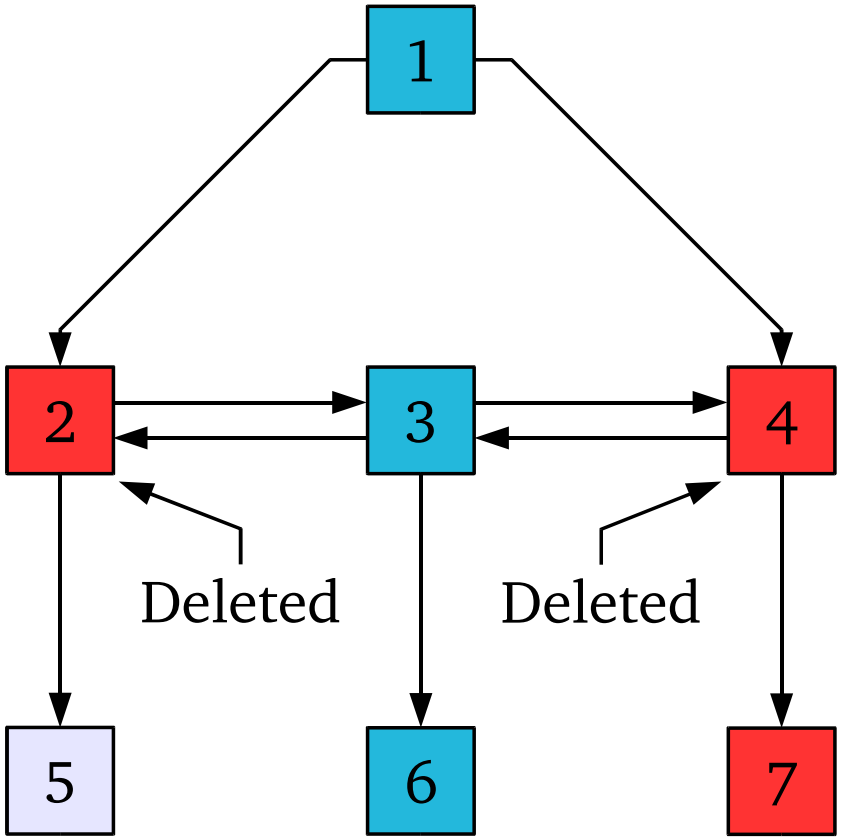
T_1		T_2	
sd	=> 1	sd	=> 1
nthF(1)	=> 2	nthB(1)	=> 4
del	Delete 2	del	Delete 4
sd	=> 1	sd	=> 1
nthB(1)	=> 4	nthF(1)	=> 3
nthF(1)	=> 7	nthF(1)	=> 6



Timestamp-based protocol

op_F	op_S	$op_F <_{TS} op_S$	$op_F >_{TS} op_S$
T	T	No conflict	No conflict
D	T	Ignore (c.a.)	Traverse
I	T	Traverse (c.a.)	Ignore
T	D	Mark deleted	Abort TS
D	D	Not possible	Abort TS
I	D	Block TS then mark deleted	Not possible

T_1	T_2
`sd  => 1`	`sd  => 1`
`nthF(1) => 2`	`nthB(1) => 4`
`del  Delete 2`	`del  Delete 4`
`sd  => 1`	`sd  => 1`
`nthB(1) => 4`	`nthF(1) => 3`
`nthF(1) => 7`	`nthF(1) => 6`



Conclusion

- What about *Dynamic commit ordering*?
- Level of parallelism depends on the granularity of locking
- The protocols shown are designed for native XML bases
- Traditional DBS could benefit from those protocols by e.g. introducing a specialized Transaction Manager
- Performance evaluation is still missing

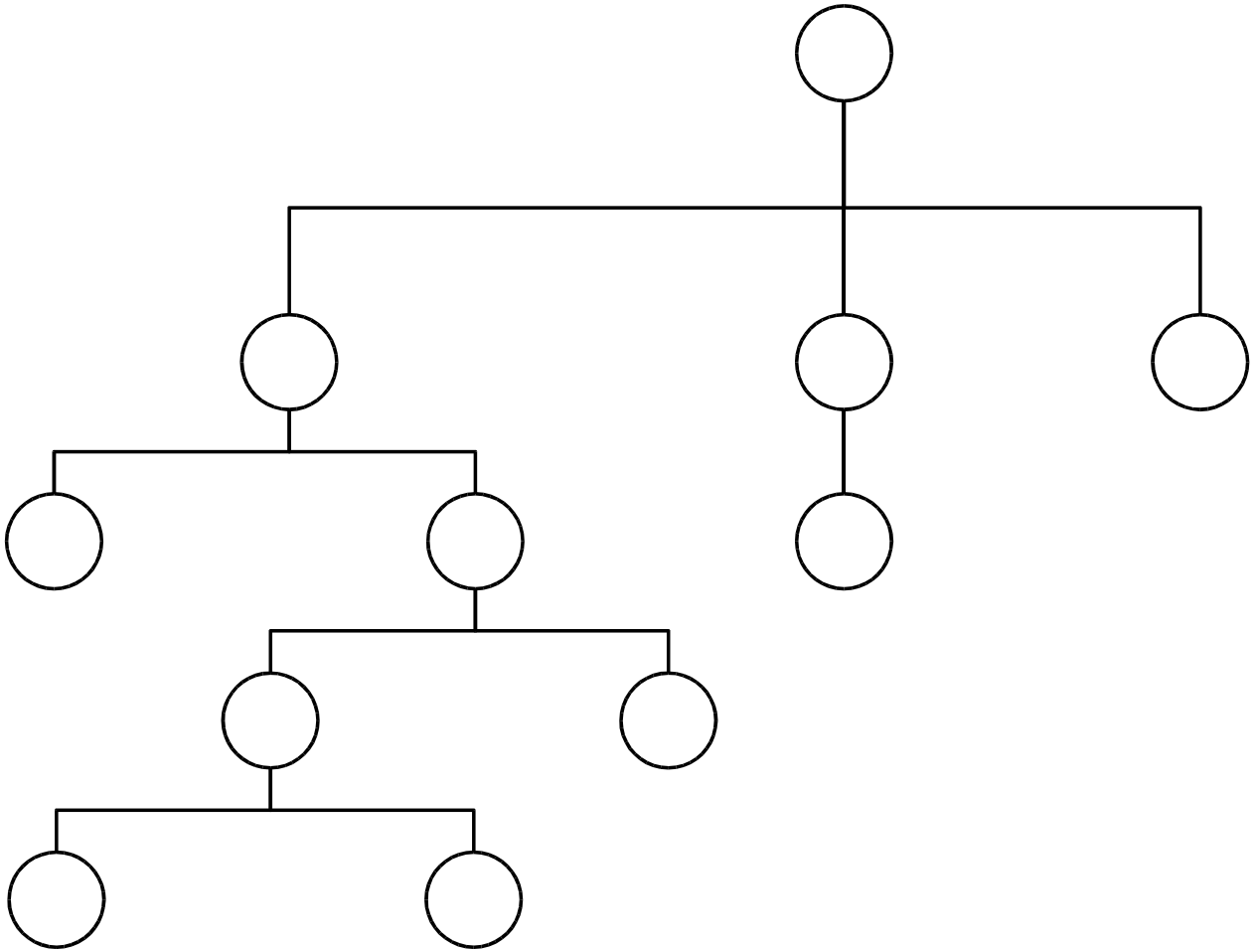
Detecting changes in XML documents

- Important for internet search engines
  - Frequent changes of online information
  - Probably don't want to rebuild whole index
- Continuous query systems (trigger systems)
  - React on the change of a document
- Versioning of XML documents
  (like CVS uses diff on text files)

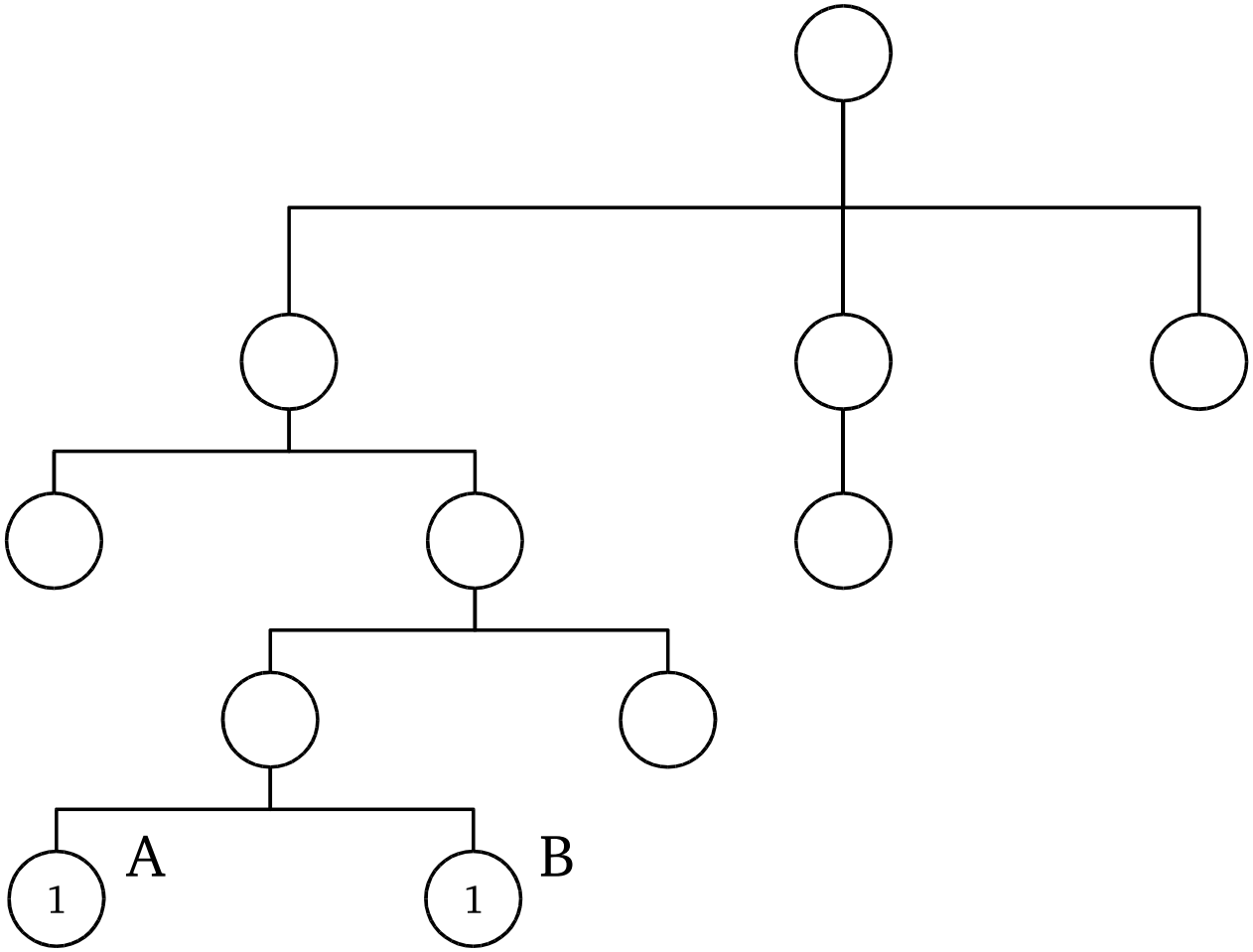
XY-Diff

- Developed as part of a web crawler
- Considers XML document trees ordered
- Designed to work on average in linear time using a greedy algorithm
- Resulting diffs are not always optimal

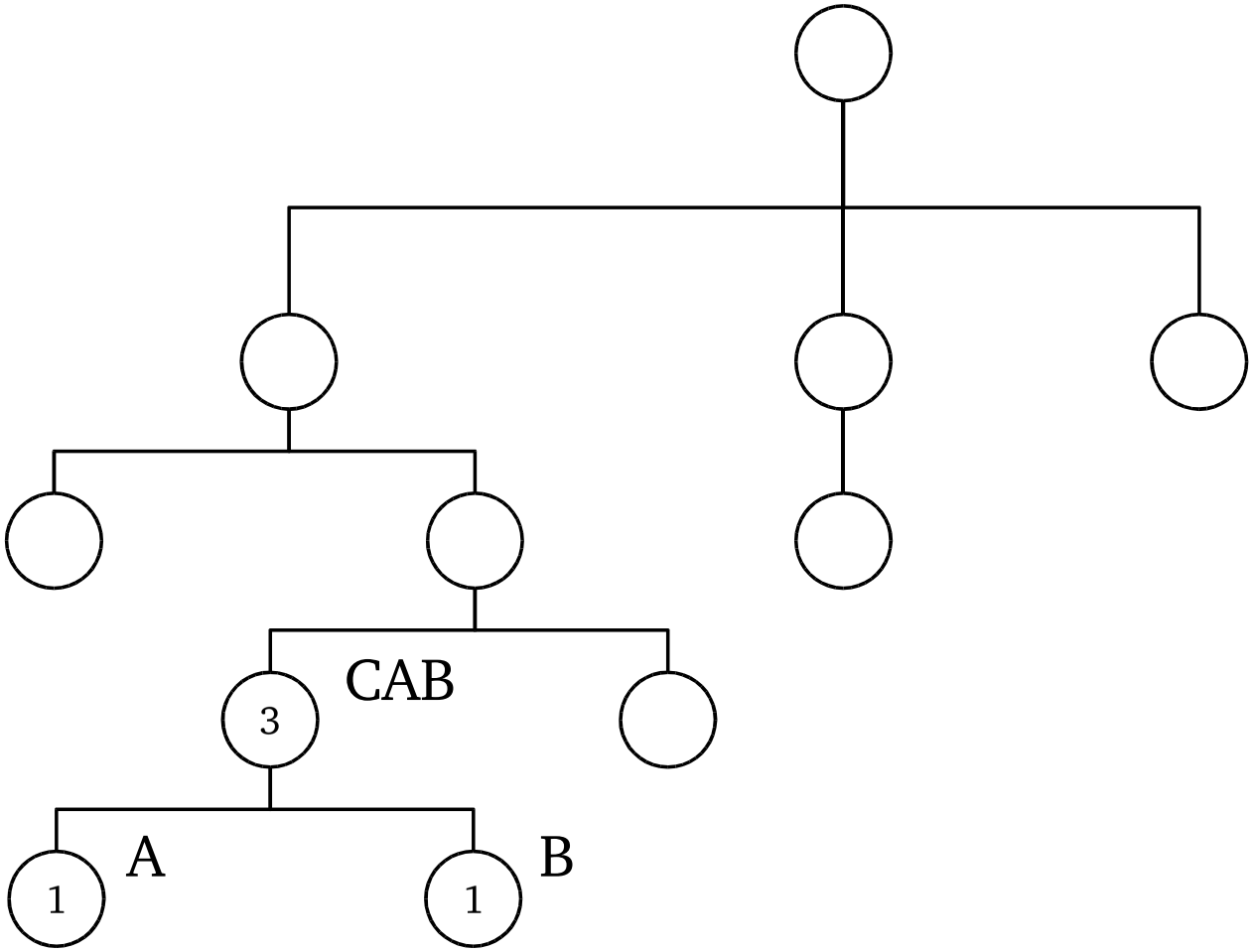
XY-Diff



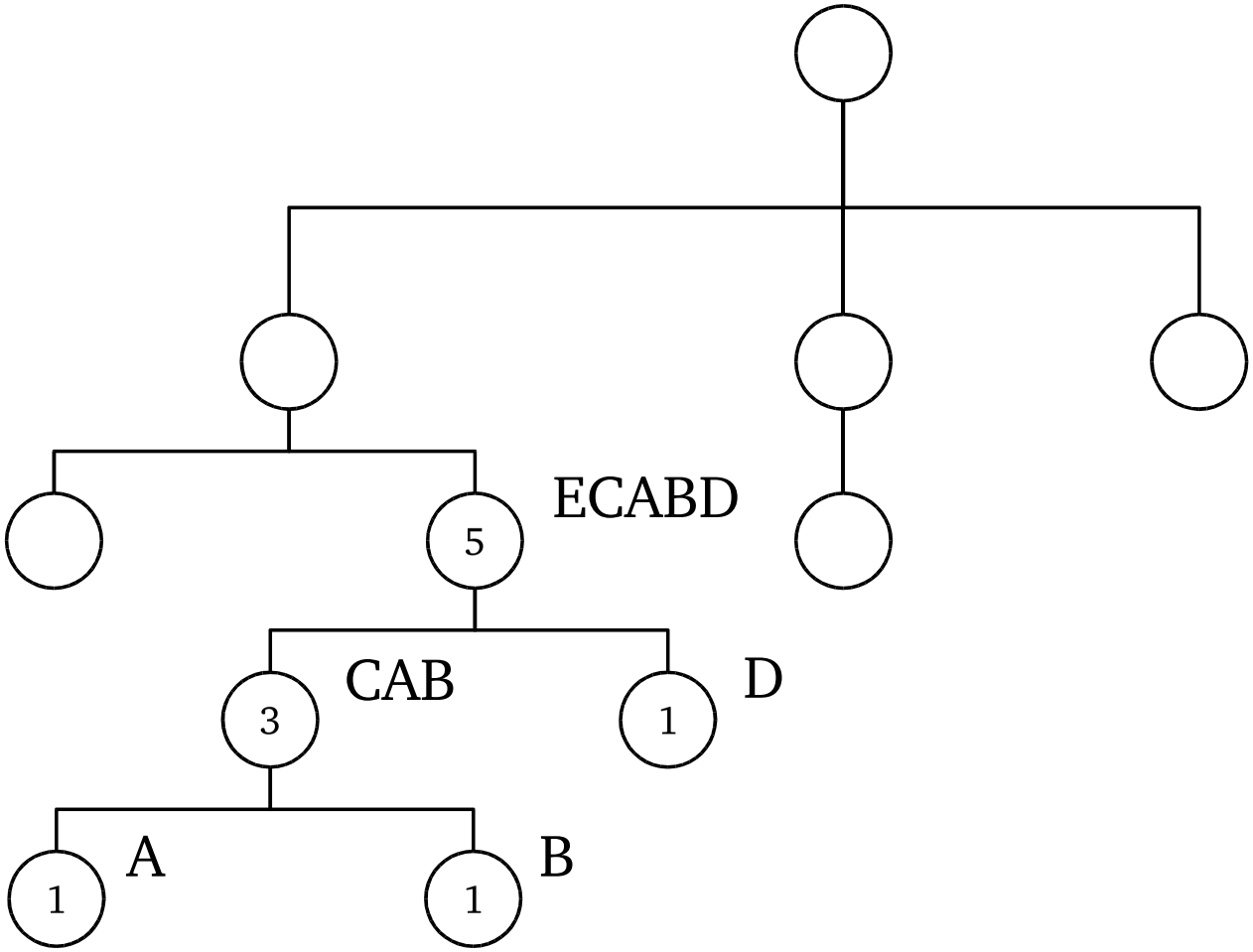
Hashing and weighting



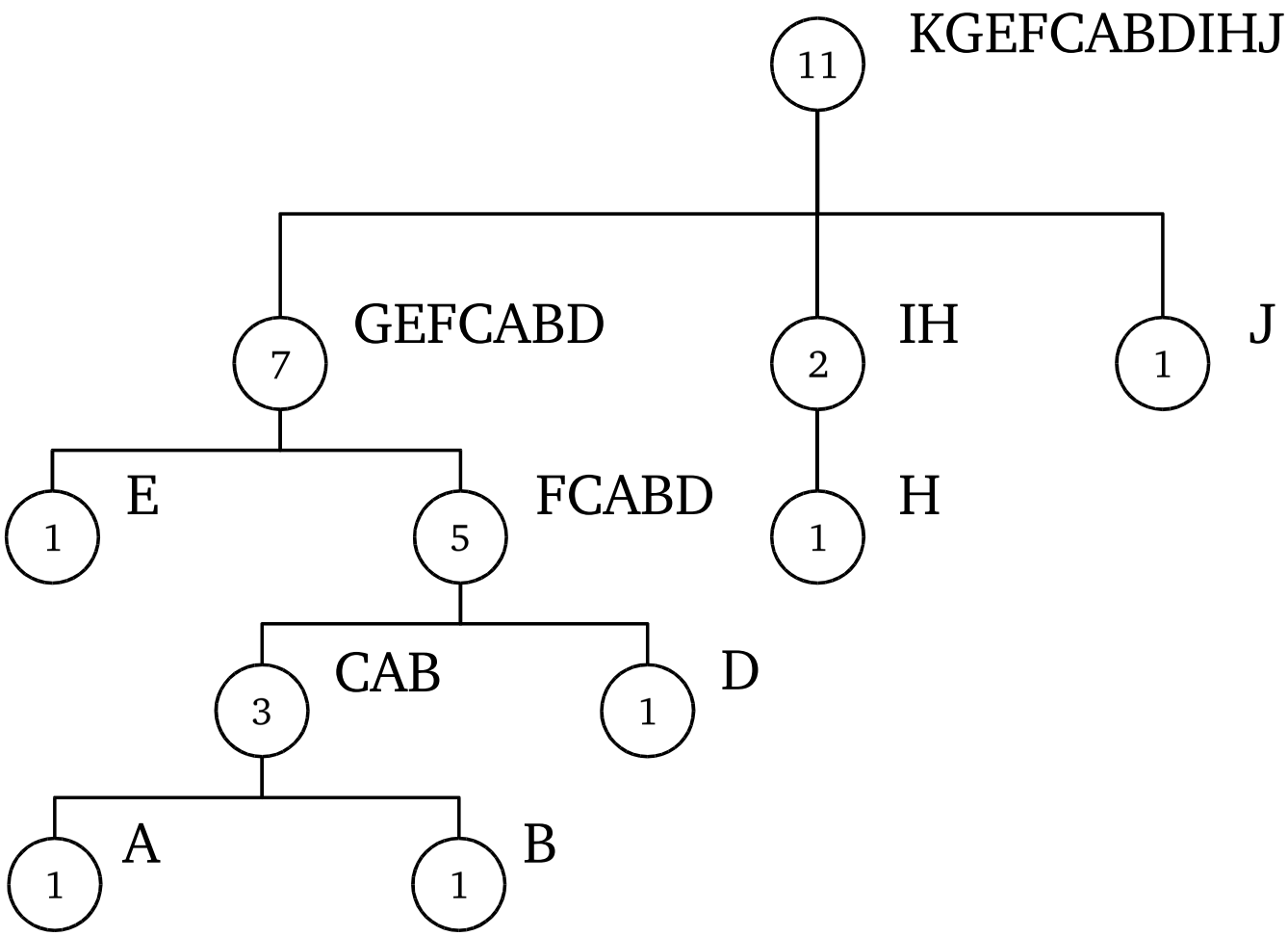
Hashing and weighting



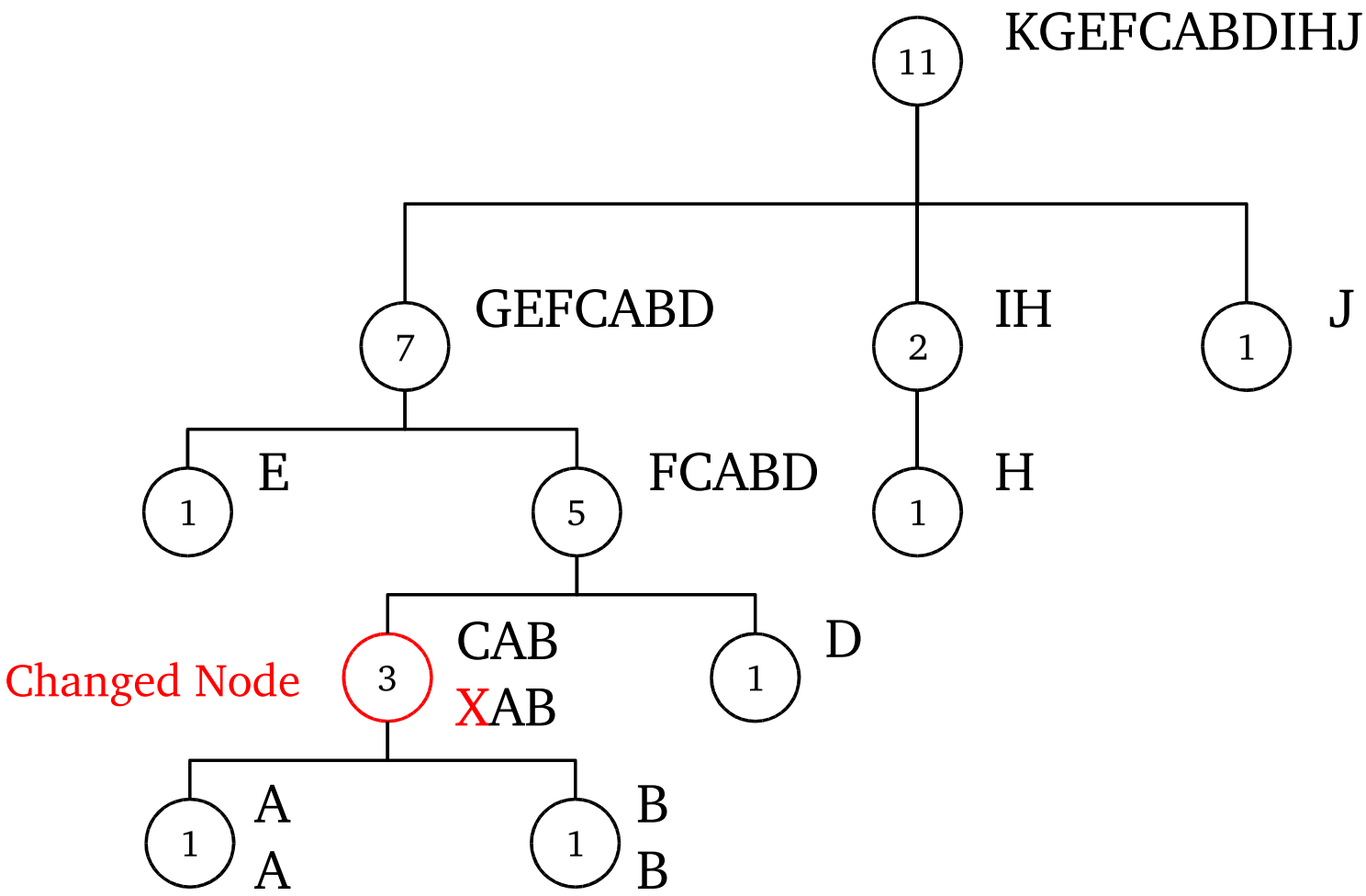
Hashing and weighting



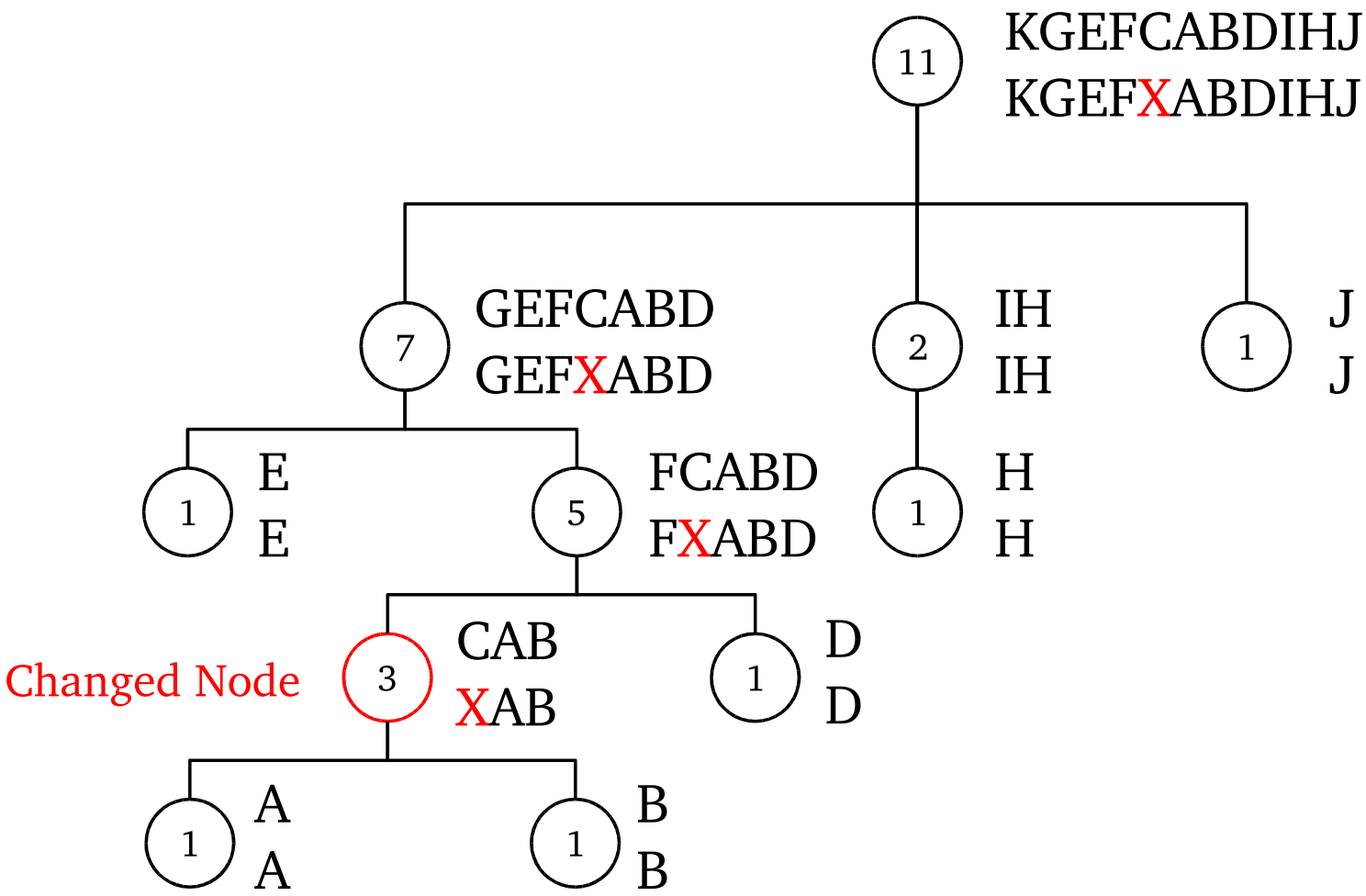
Hashing and weighting



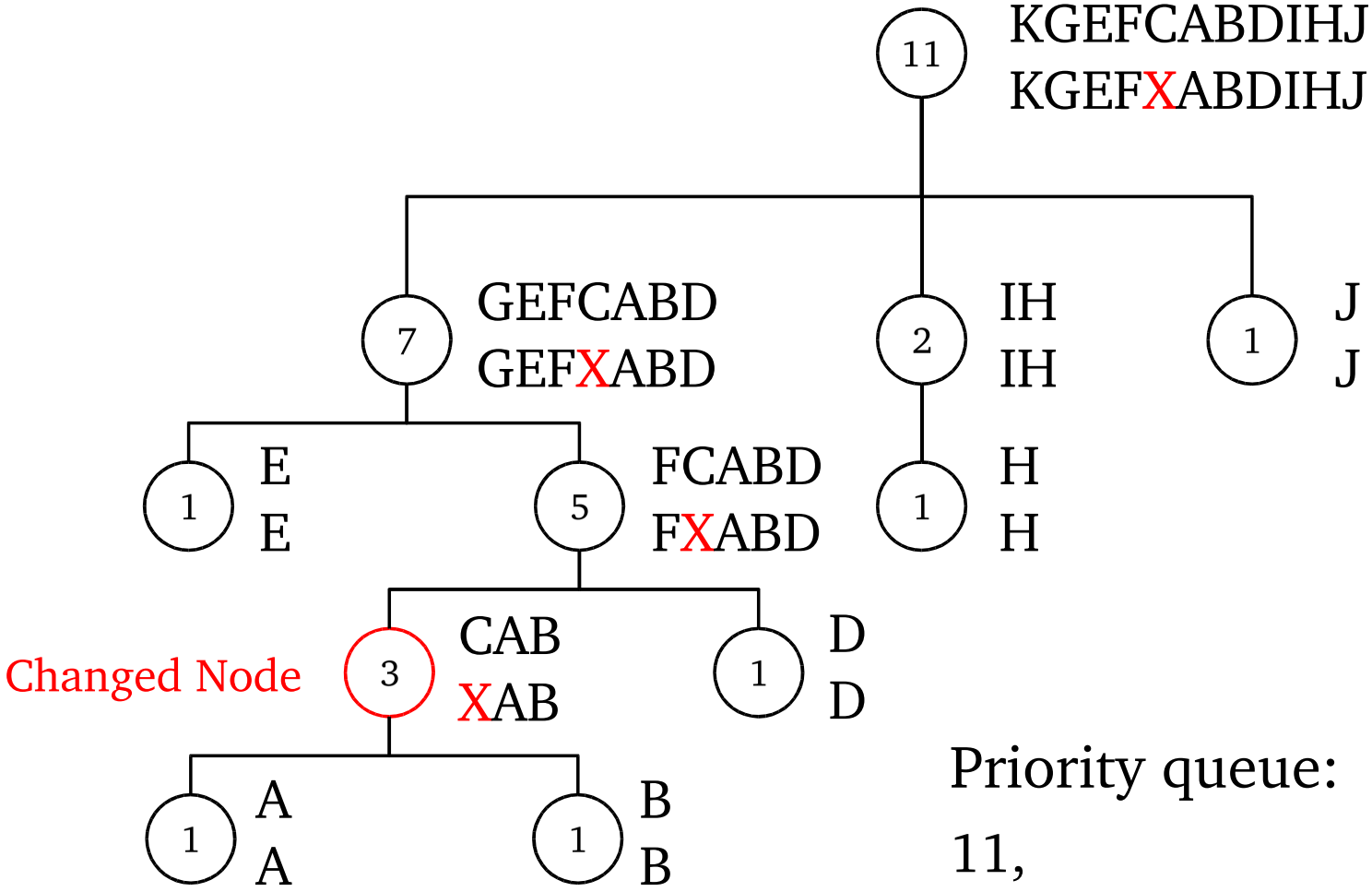
Hashing and weighting



Hashing and weighting



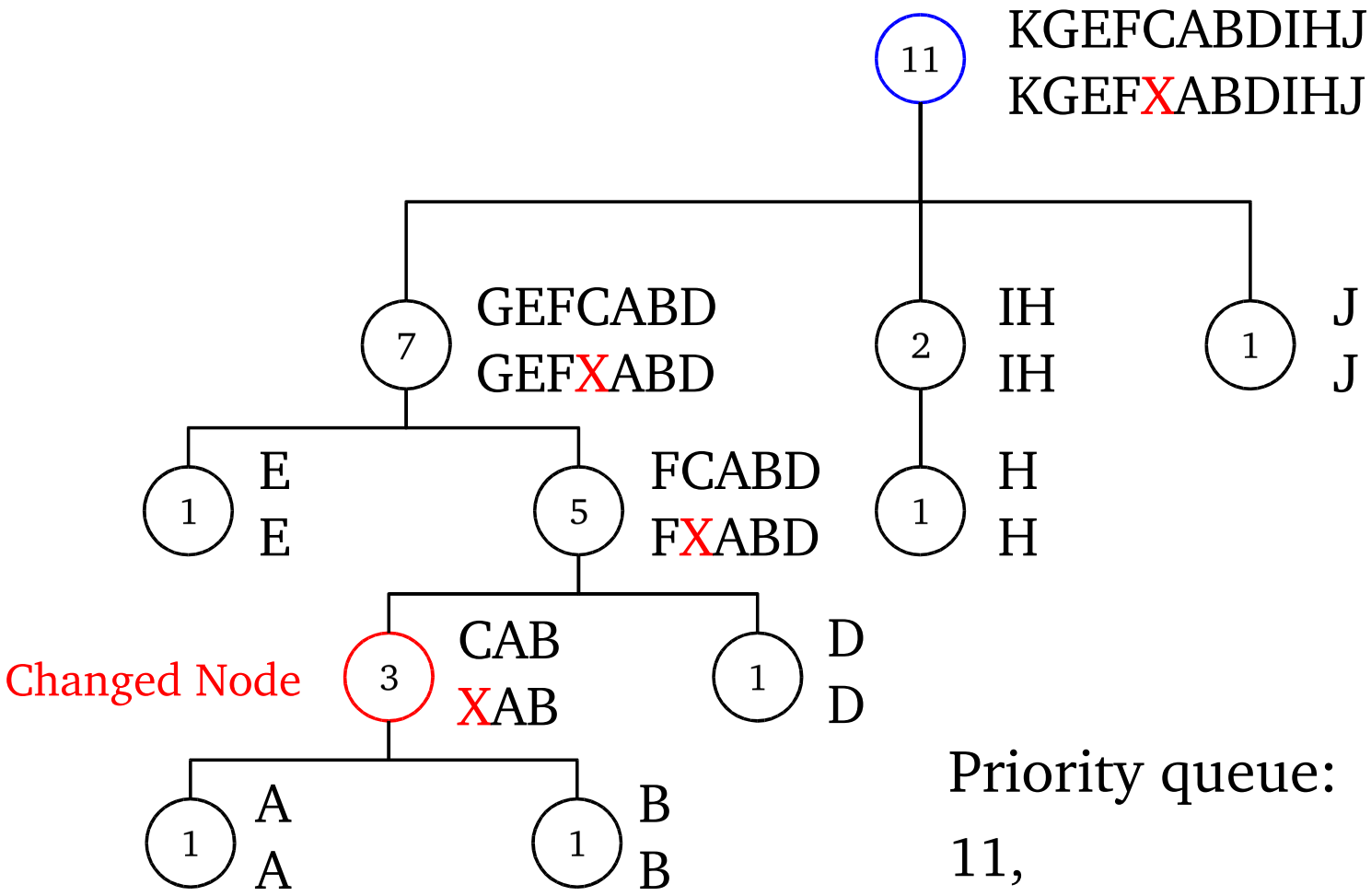
Create priority queue for the weights, start with the root node



Priority queue:

11,

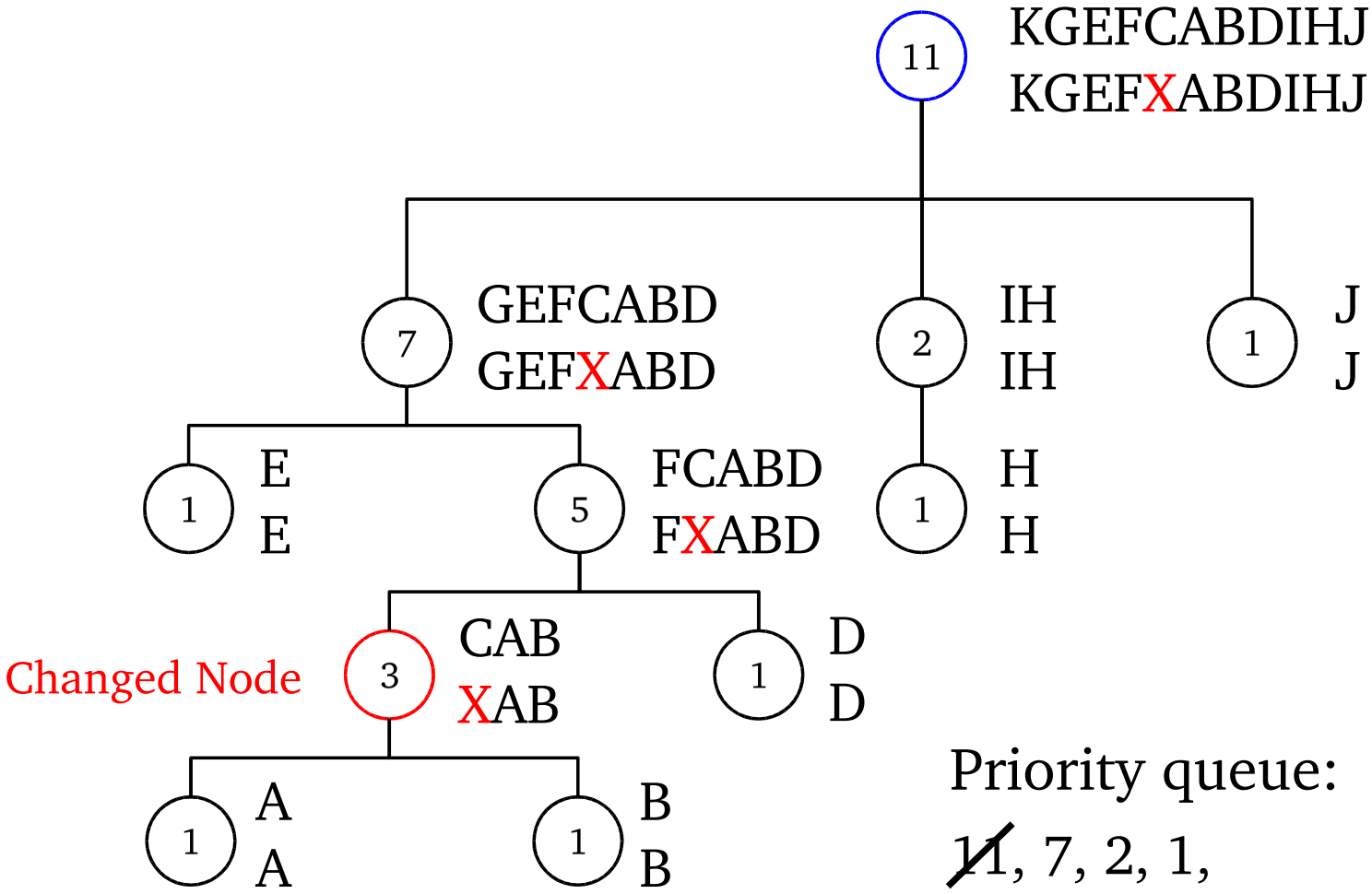
Compare nodes



Priority queue:

11,

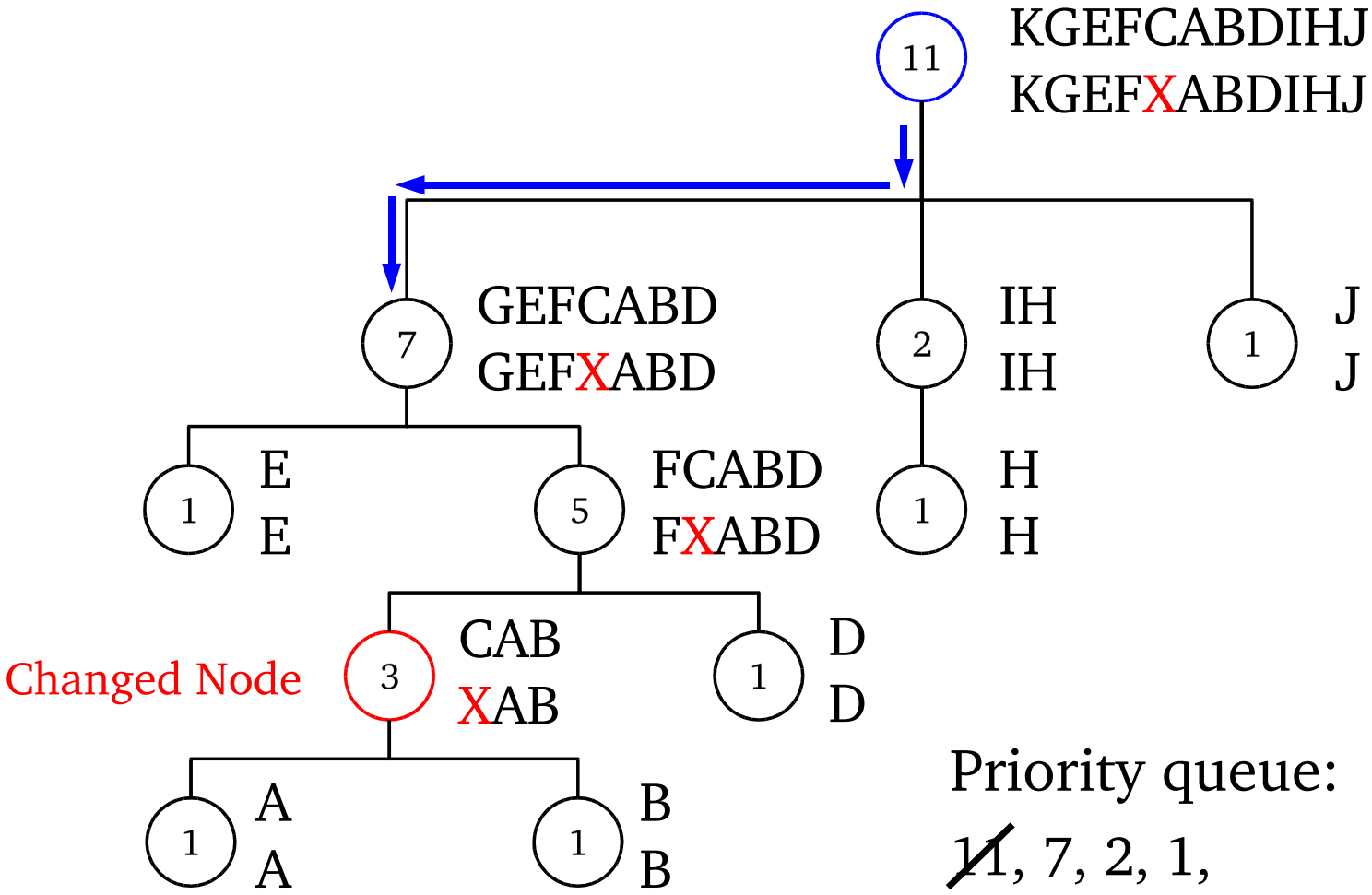
Hash value of the root node is different => add children of root to priority queue



Priority queue:

~~11~~, 7, 2, 1,

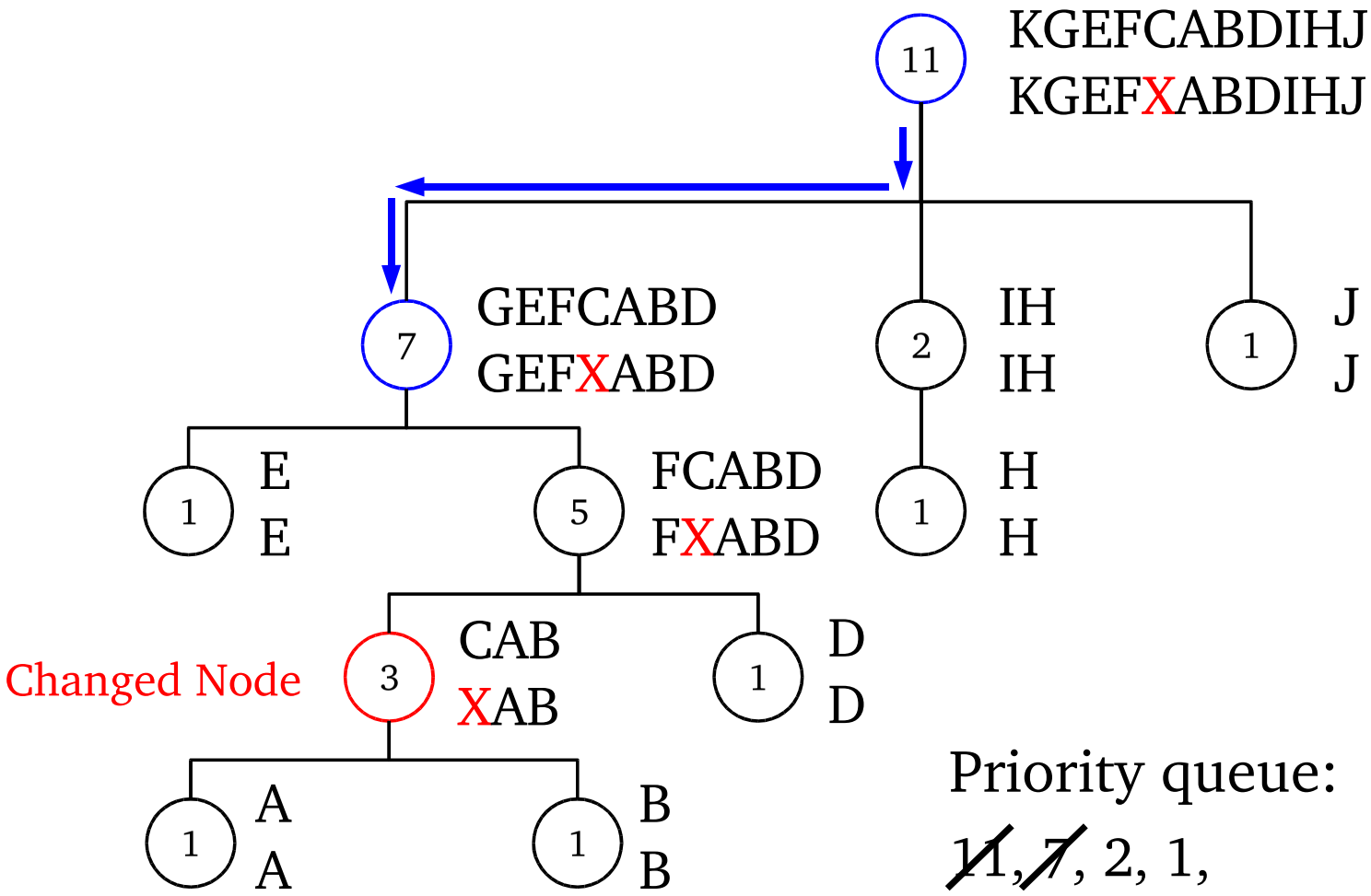
Hash value of the root node is different => add children of root to priority queue



Priority queue:

~~11~~, 7, 2, 1,

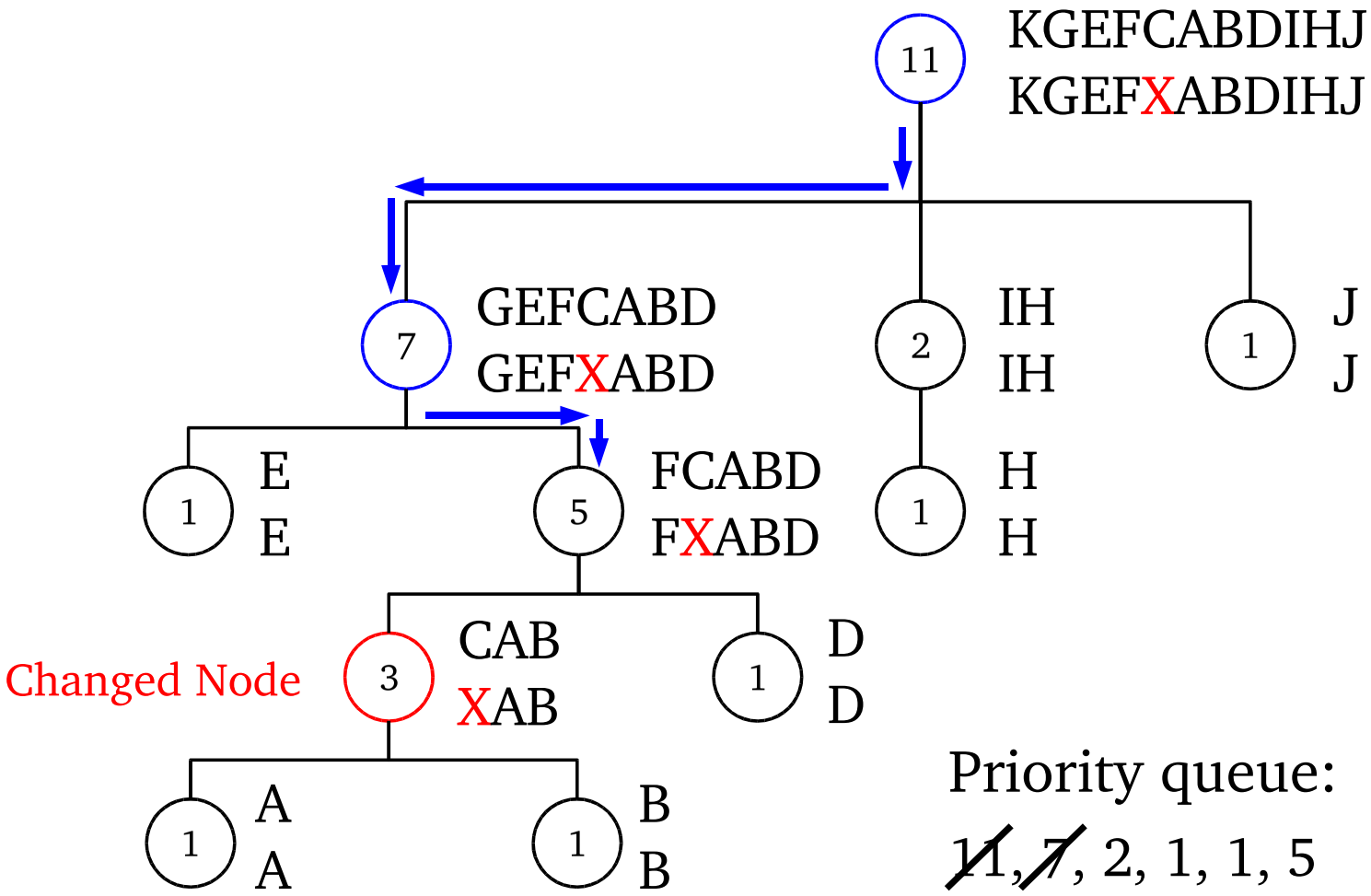
Repeat until differences are found



Priority queue:

~~11~~, ~~7~~, 2, 1,

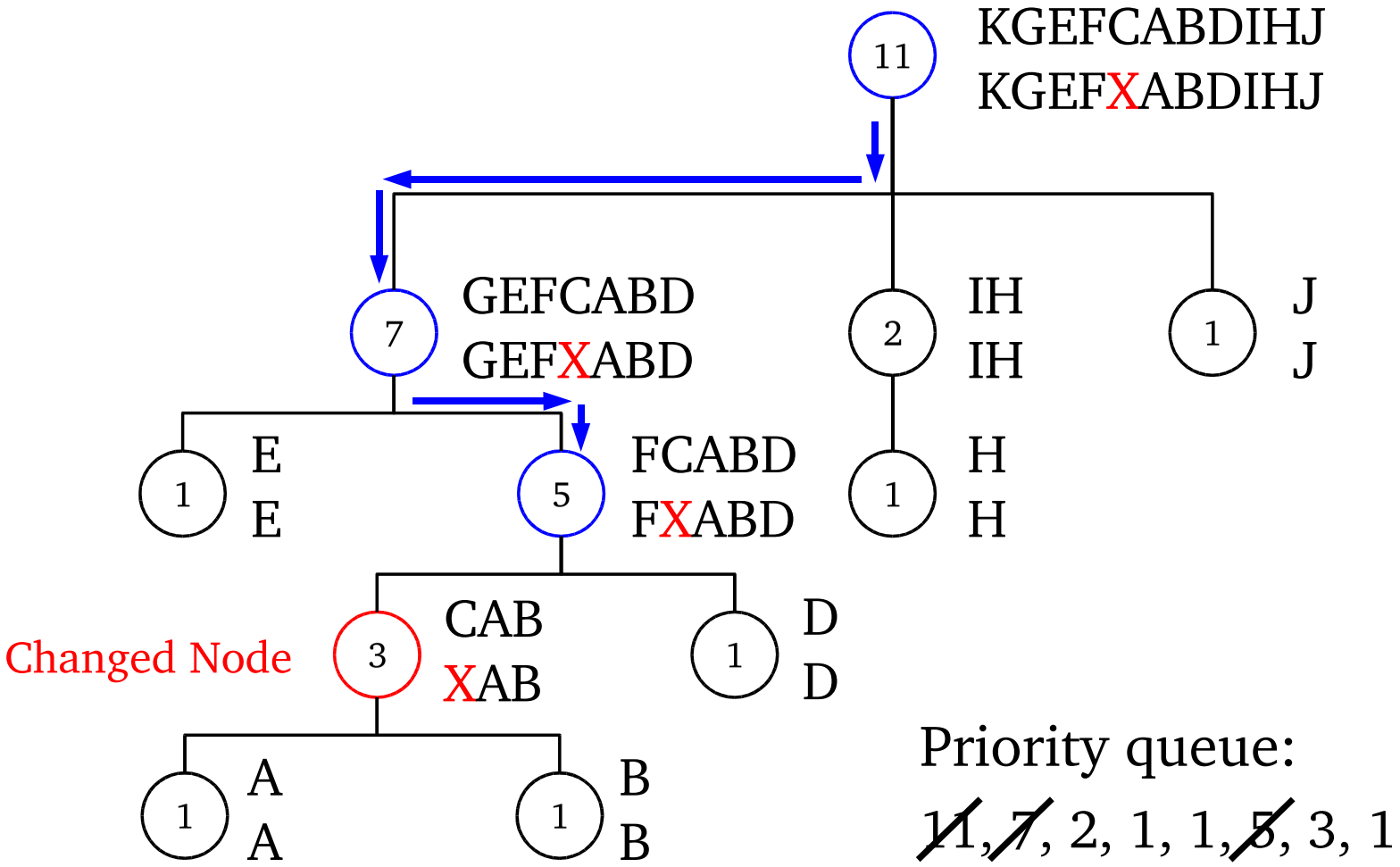
Repeat until differences are found



Priority queue:

~~11~~, ~~7~~, 2, 1, 1, 5

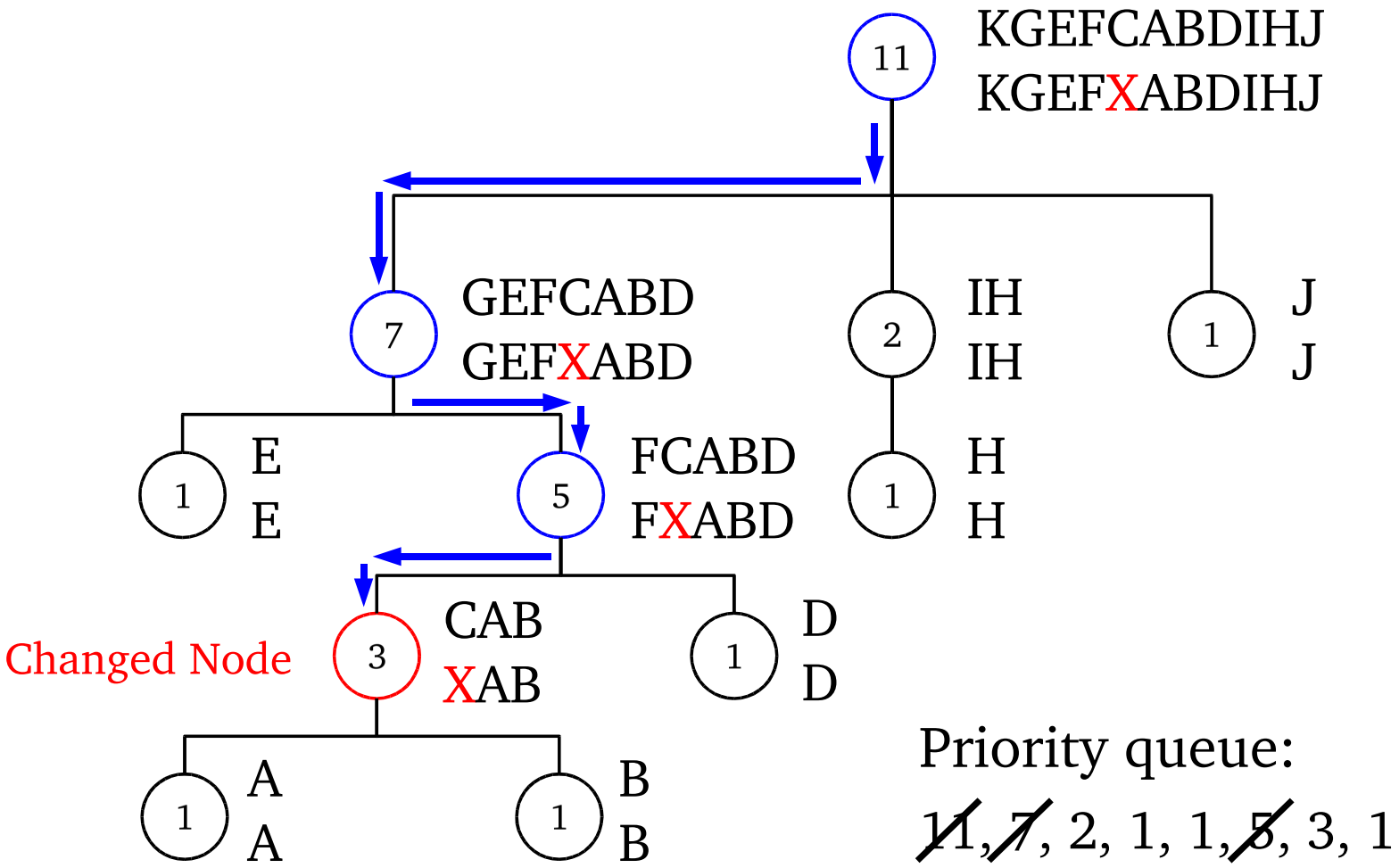
Repeat until differences are found



Priority queue:

~~11~~, ~~7~~, 2, 1, 1, ~~5~~, 3, 1

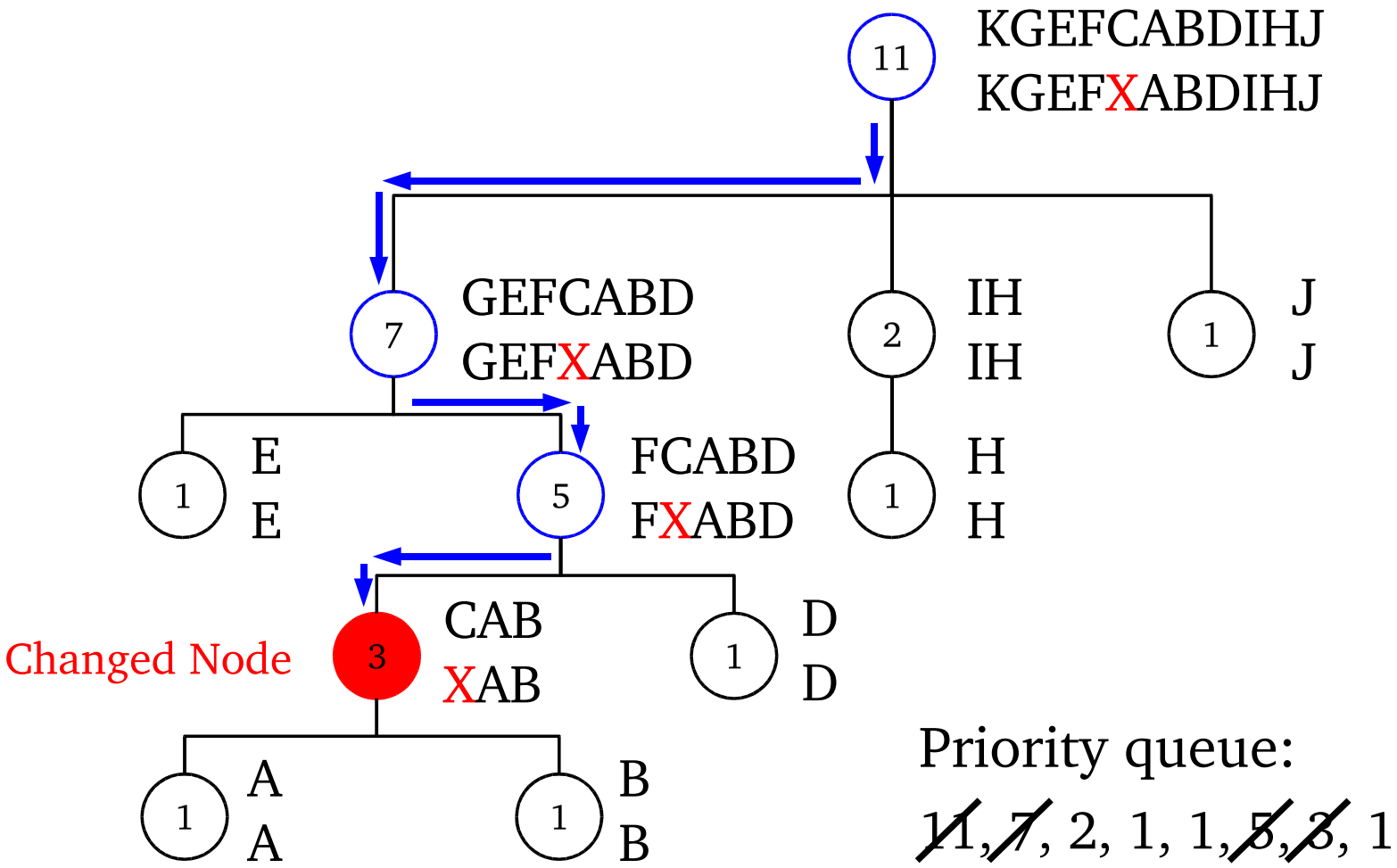
Repeat until differences are found



Priority queue:

~~11~~, ~~7~~, 2, 1, 1, ~~5~~, 3, 1

Repeat until differences are found

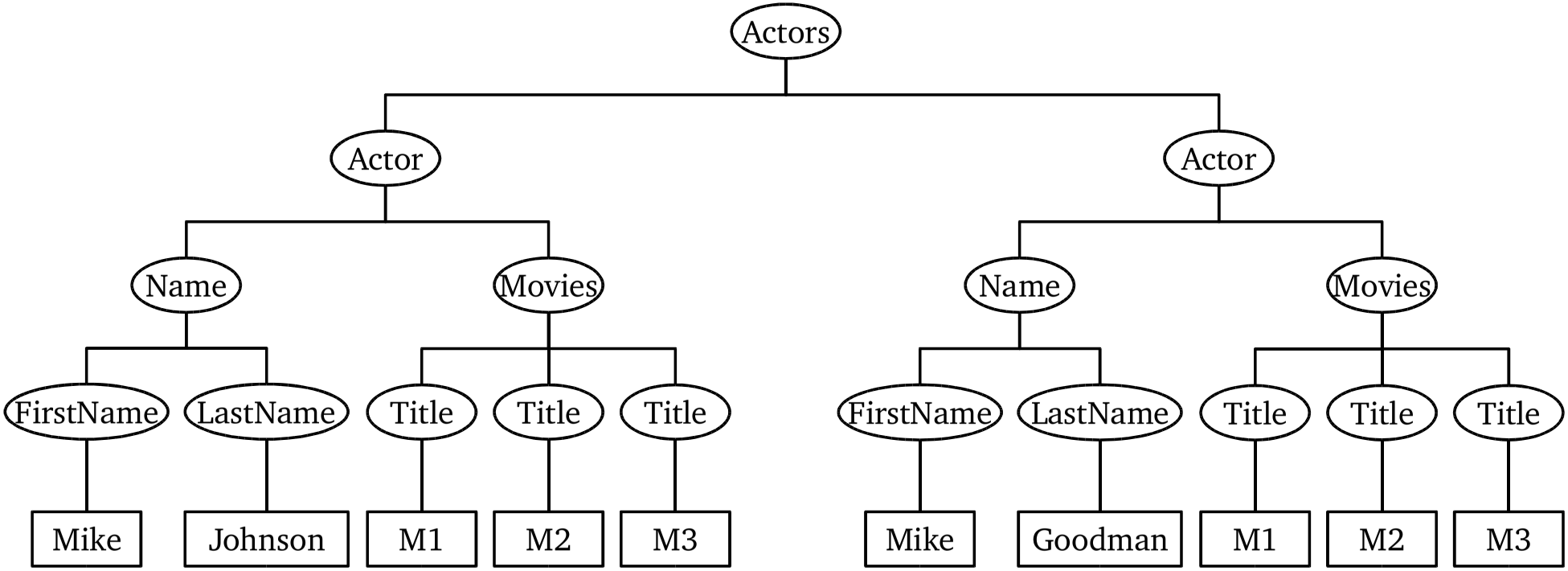


Priority queue:

~~11~~, ~~7~~, 2, 1, 1, ~~5~~, ~~3~~, 1

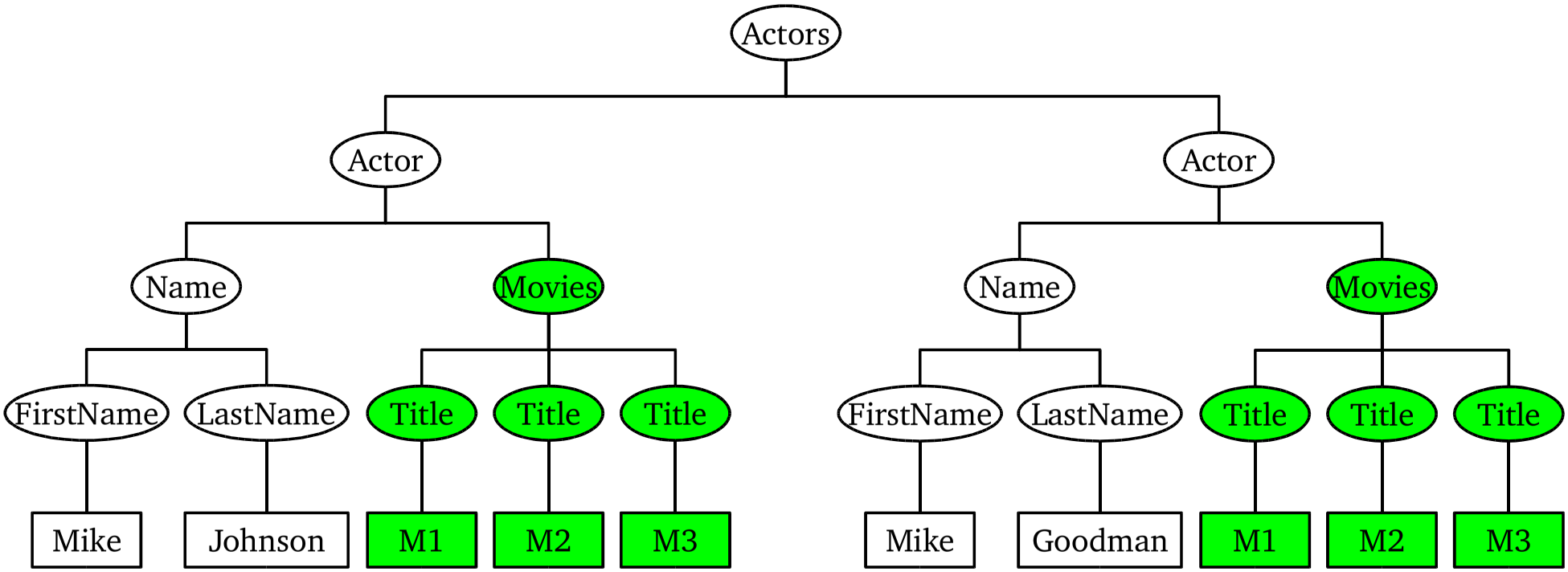
Disadvantage of XY-Diff

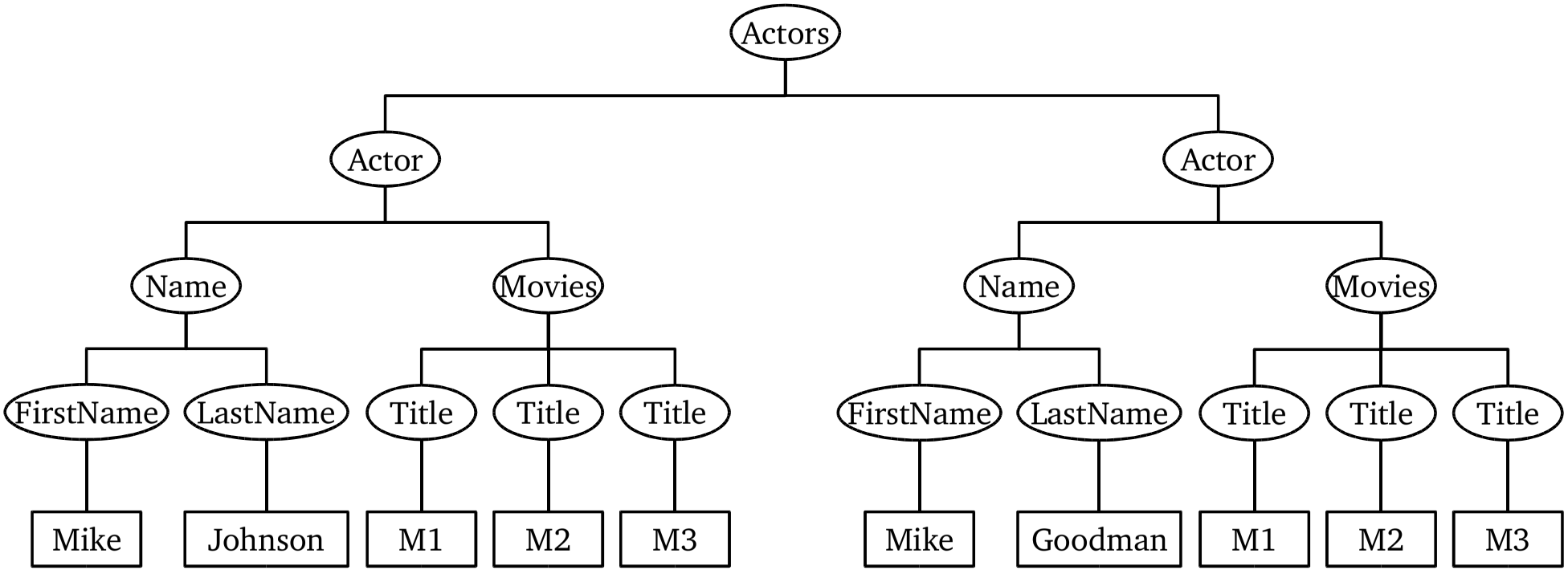
- Tendency to mismatch nodes when guided by its greedy matching rules

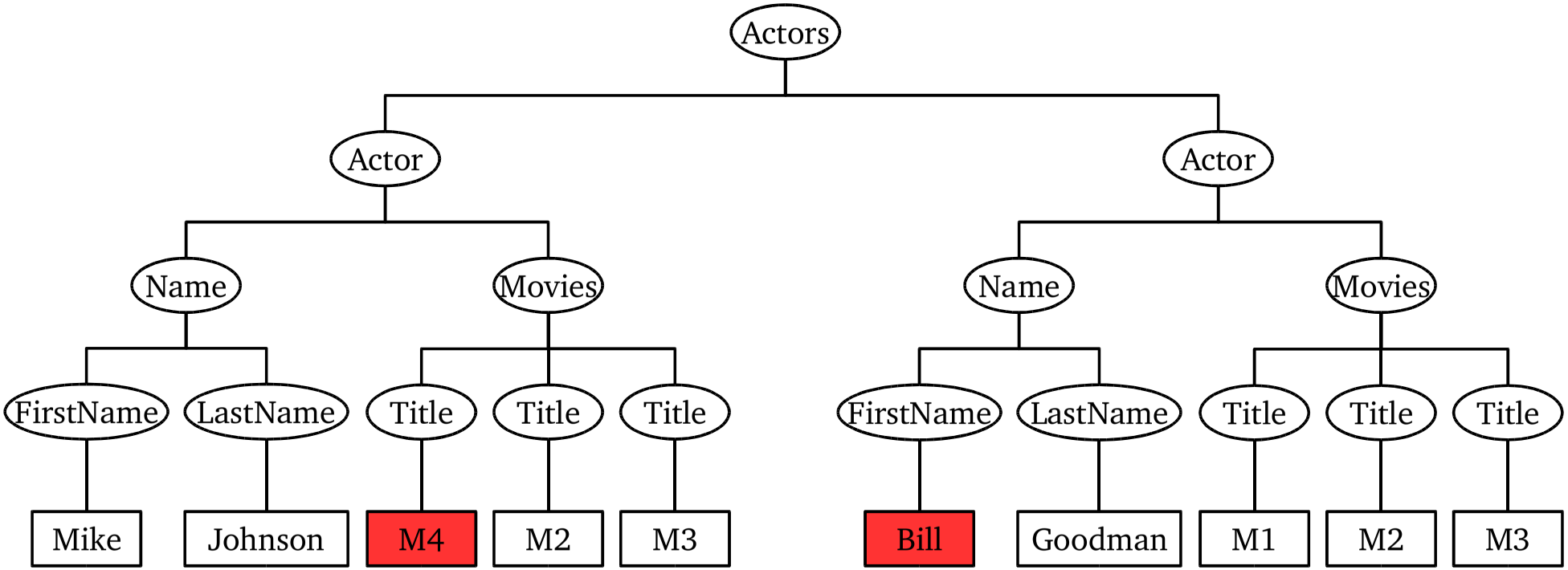


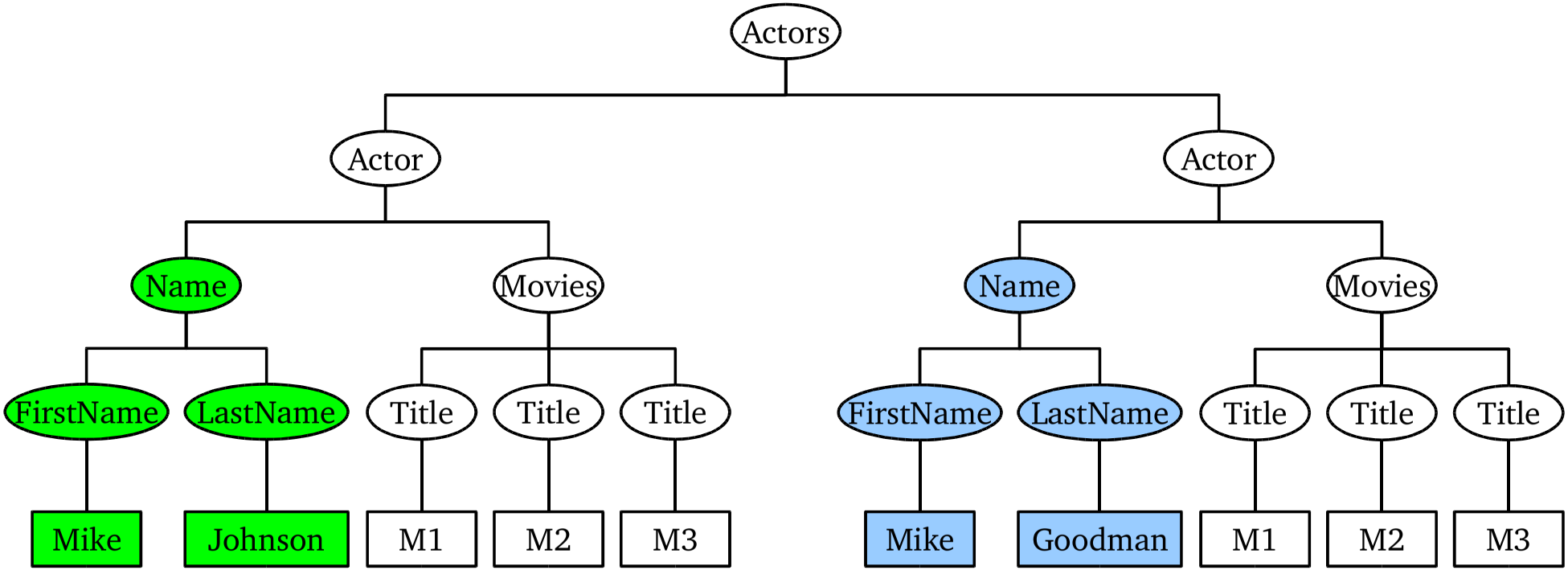
Disadvantage of XY-Diff

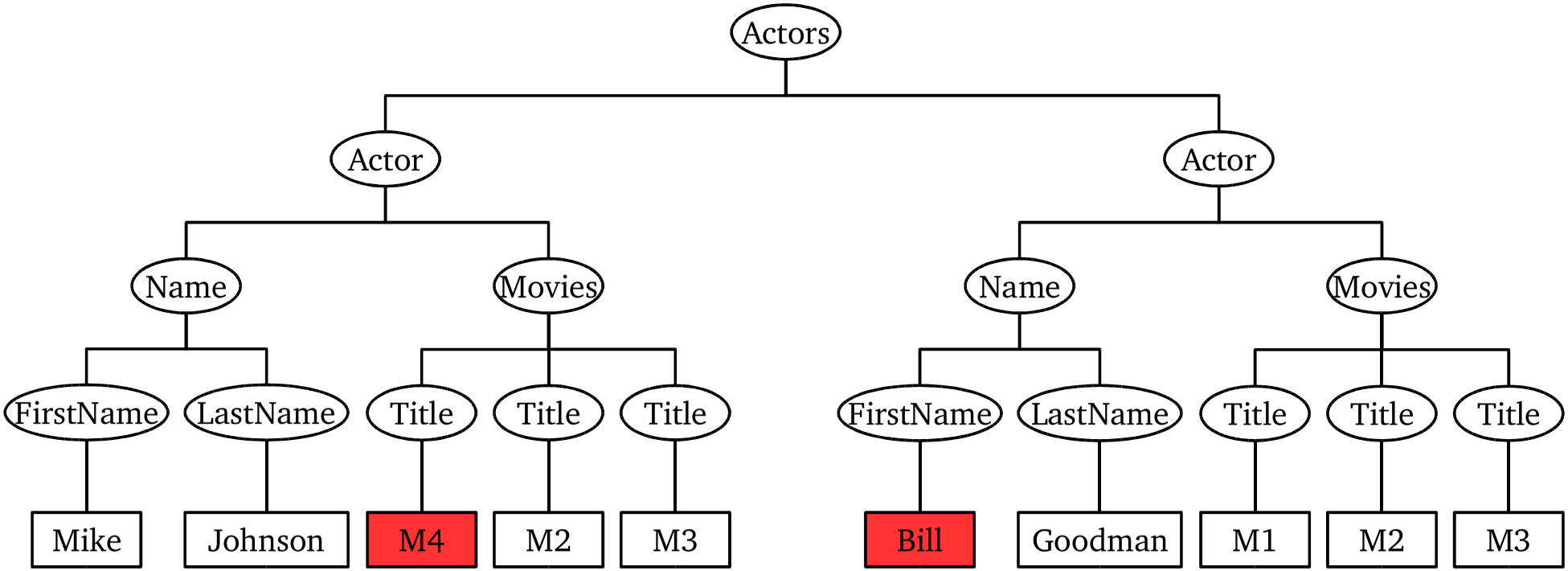
- Tendency to mismatch nodes when guided by its greedy matching rules

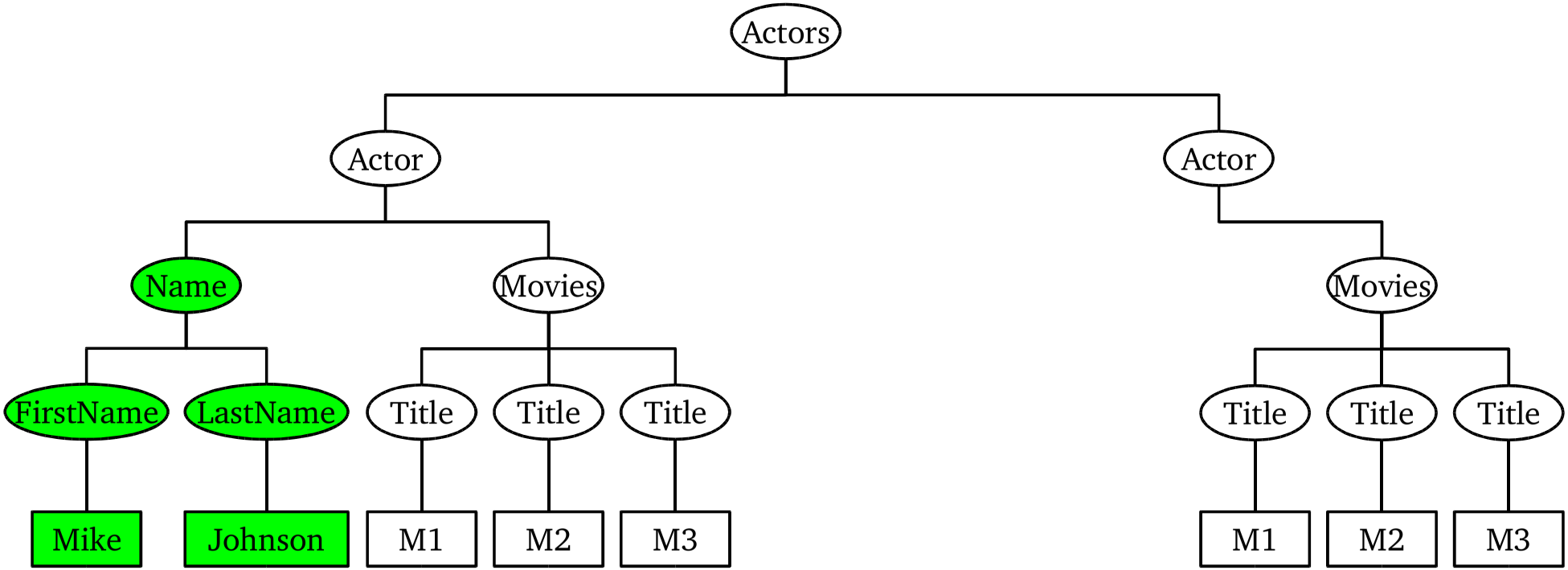


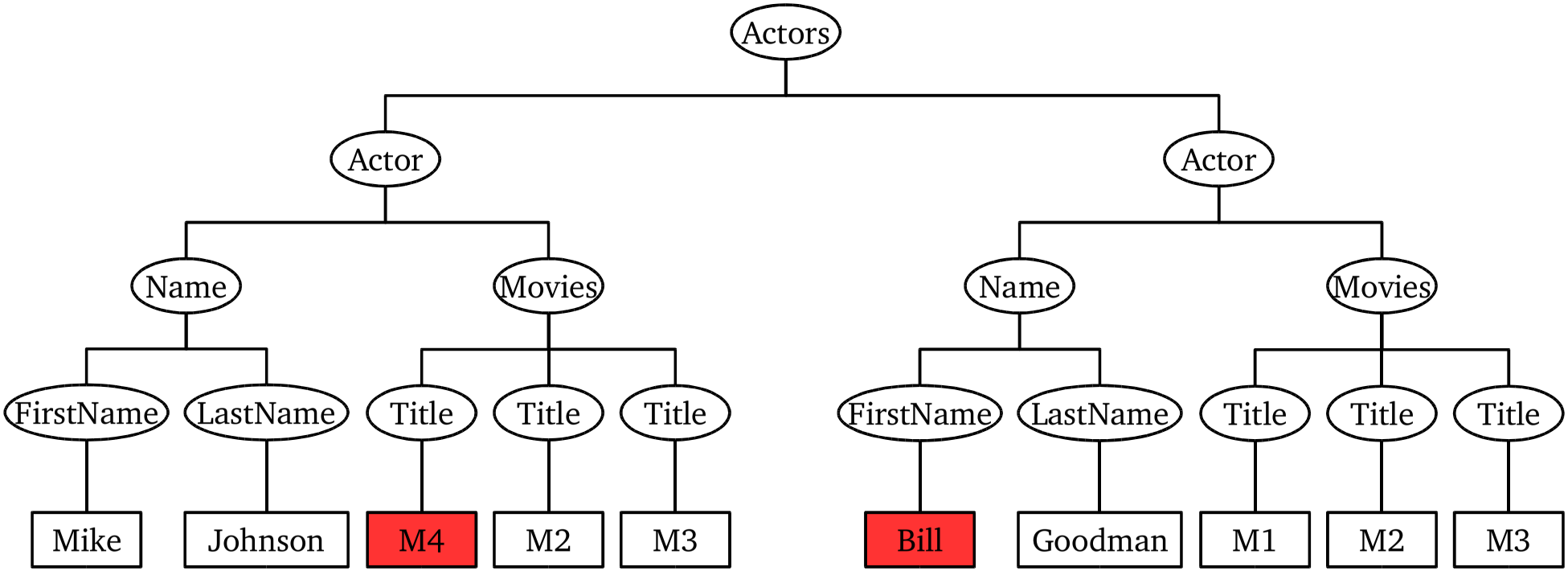

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M1
M2
M3
Mike
Goodman
M1
M2
M3

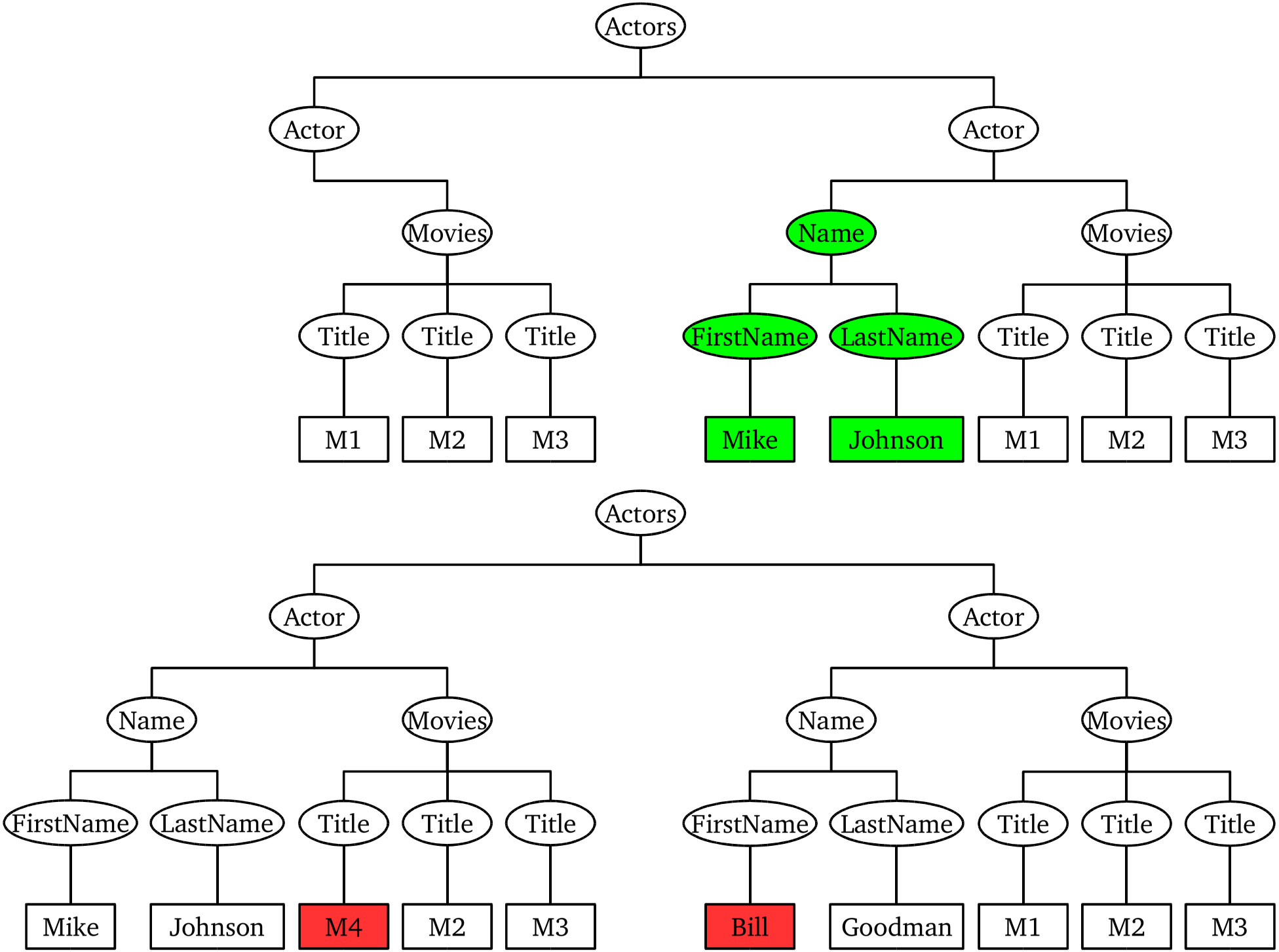

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

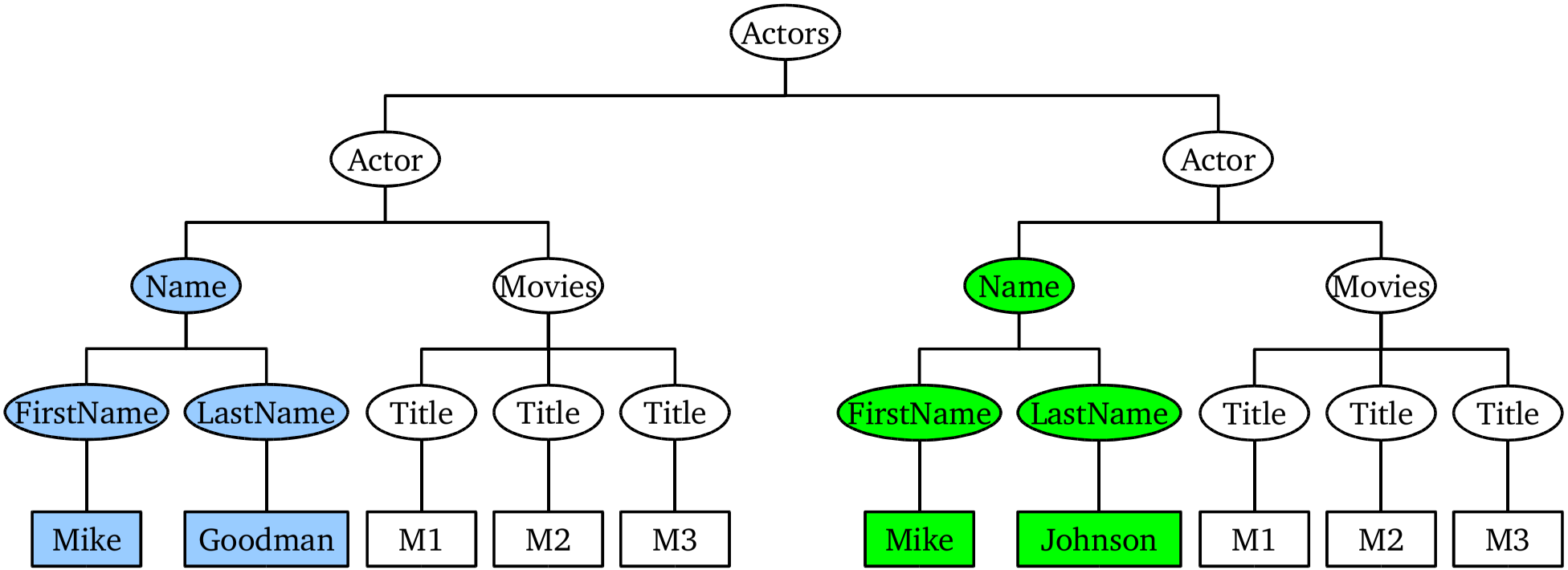

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M1
M2
M3
Mike
Goodman
M1
M2
M3

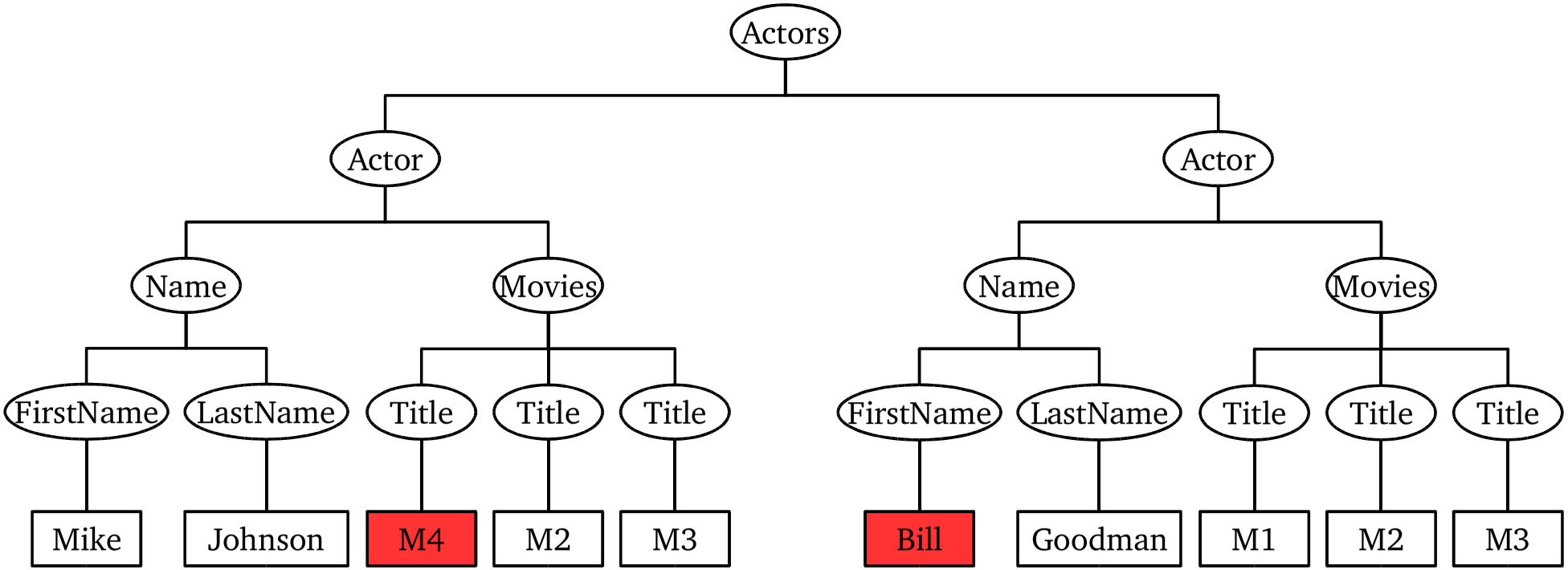

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

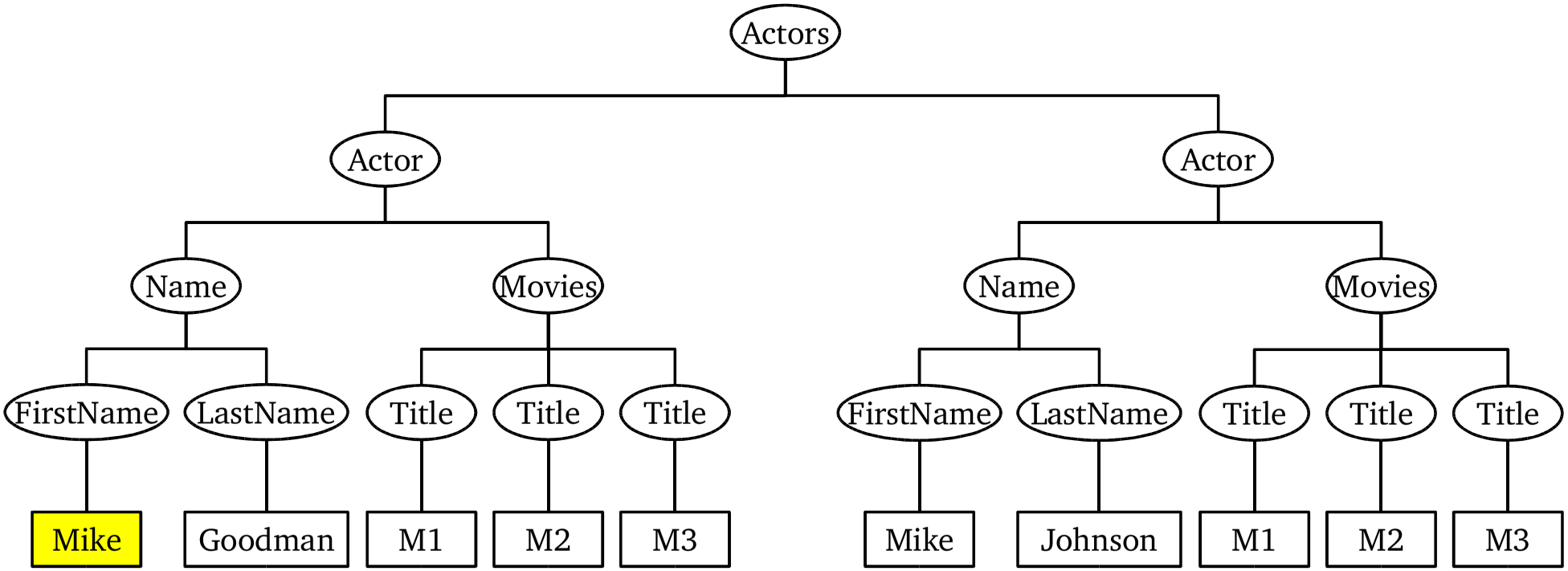

Actors
Actor
Actor
Name
Movies
Movies
FirstName
LastName
Title
Title
Title
Title
Title
Title
Mike
Johnson
M1
M2
M3
M1
M2
M3

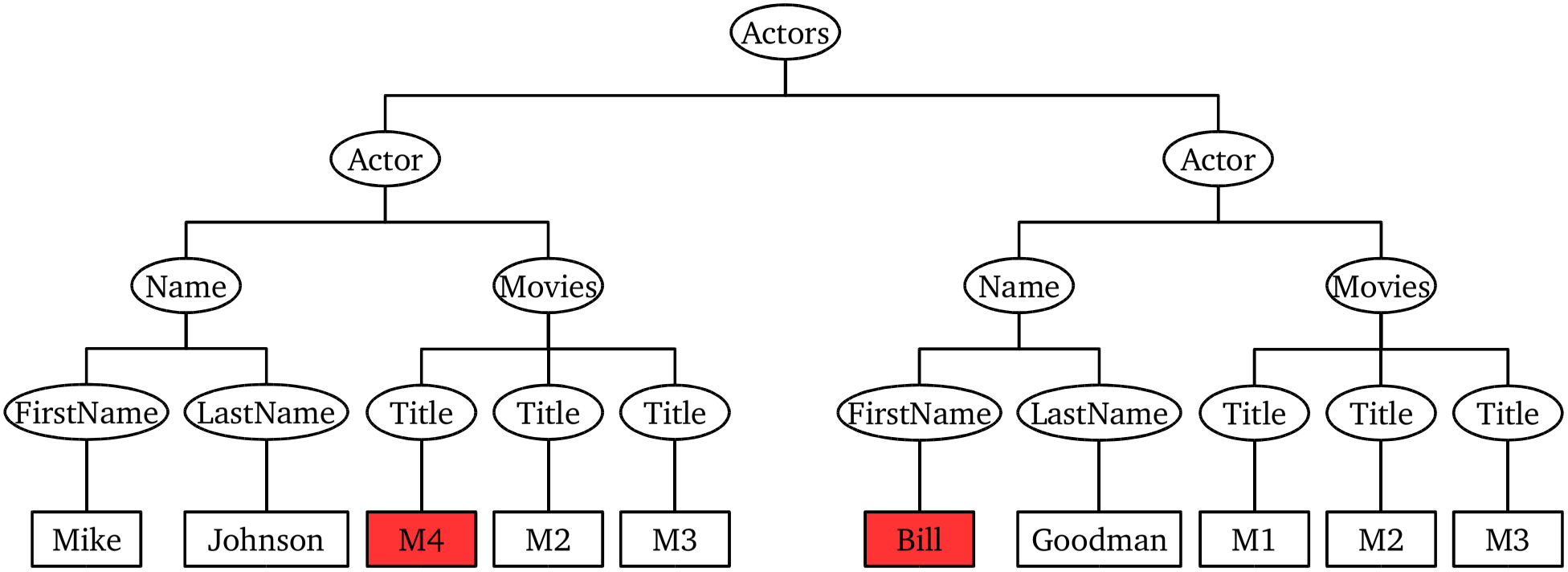

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

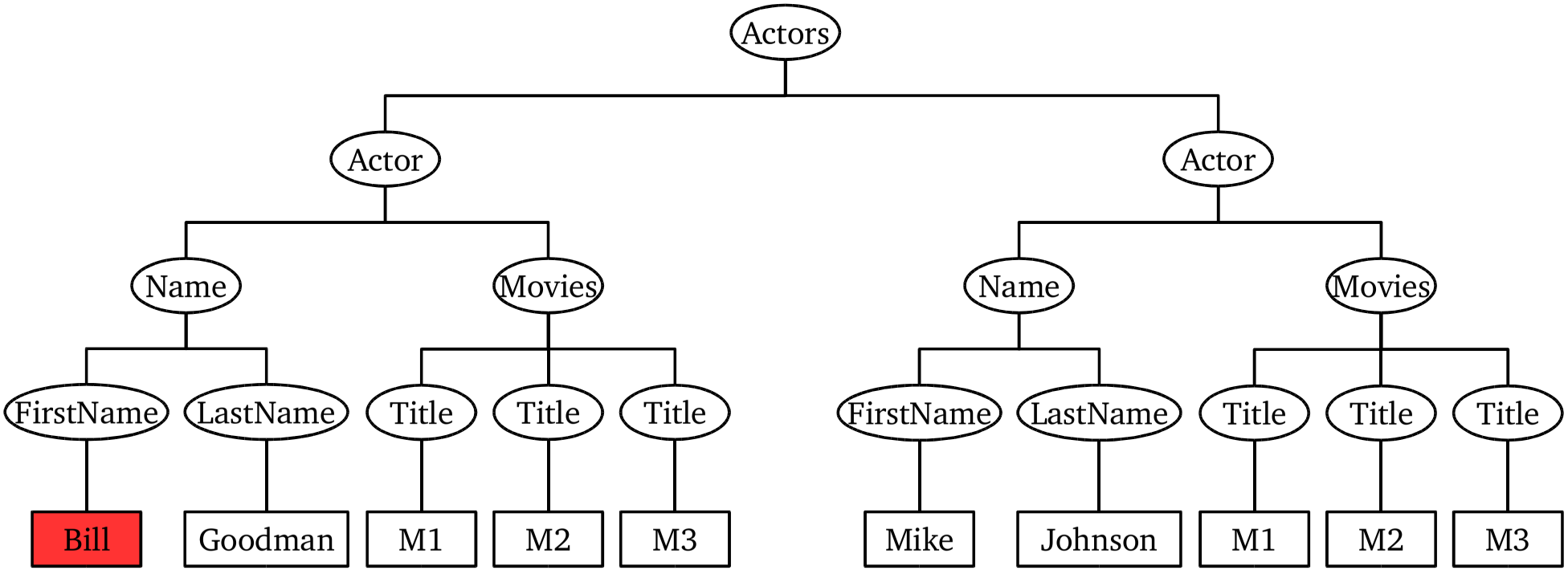

Actors
Actor
Actor
Movies
Name
Movies
Title
Title
Title
FirstName
LastName
Title
Title
Title
M1
M2
M3
Mike
Johnson
M1
M2
M3
Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

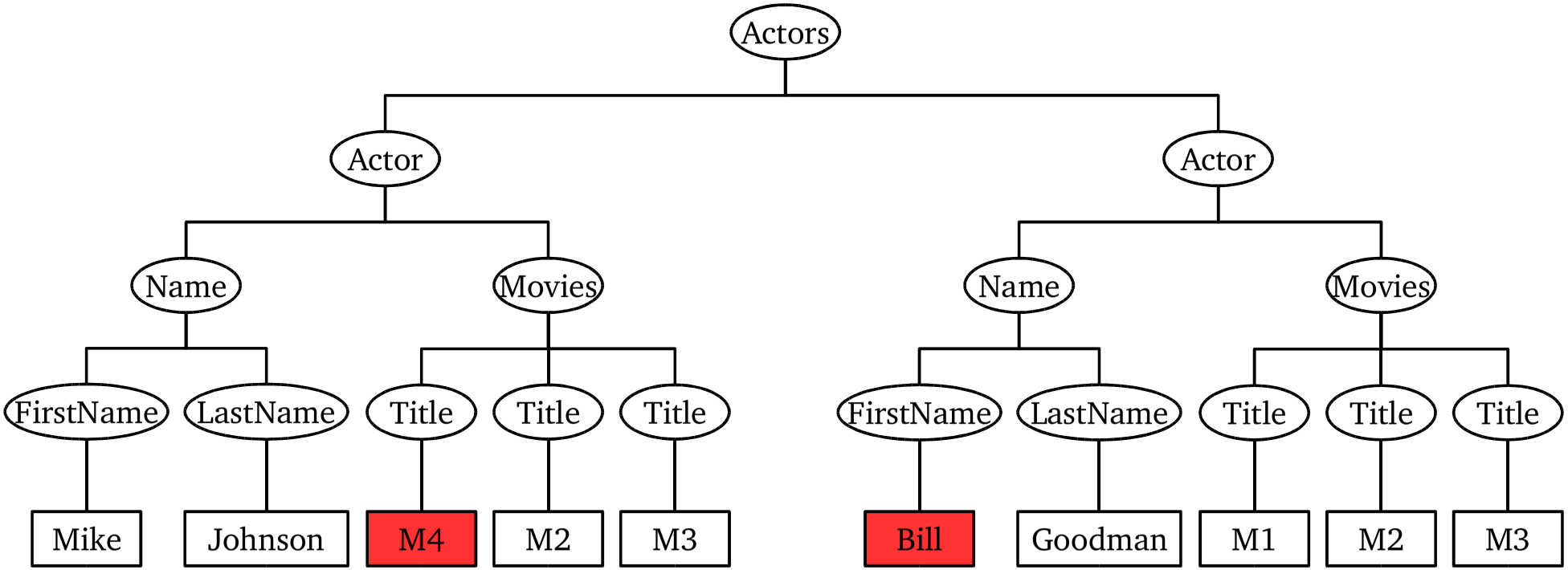

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Goodman
M1
M2
M3
Mike
Johnson
M1
M2
M3

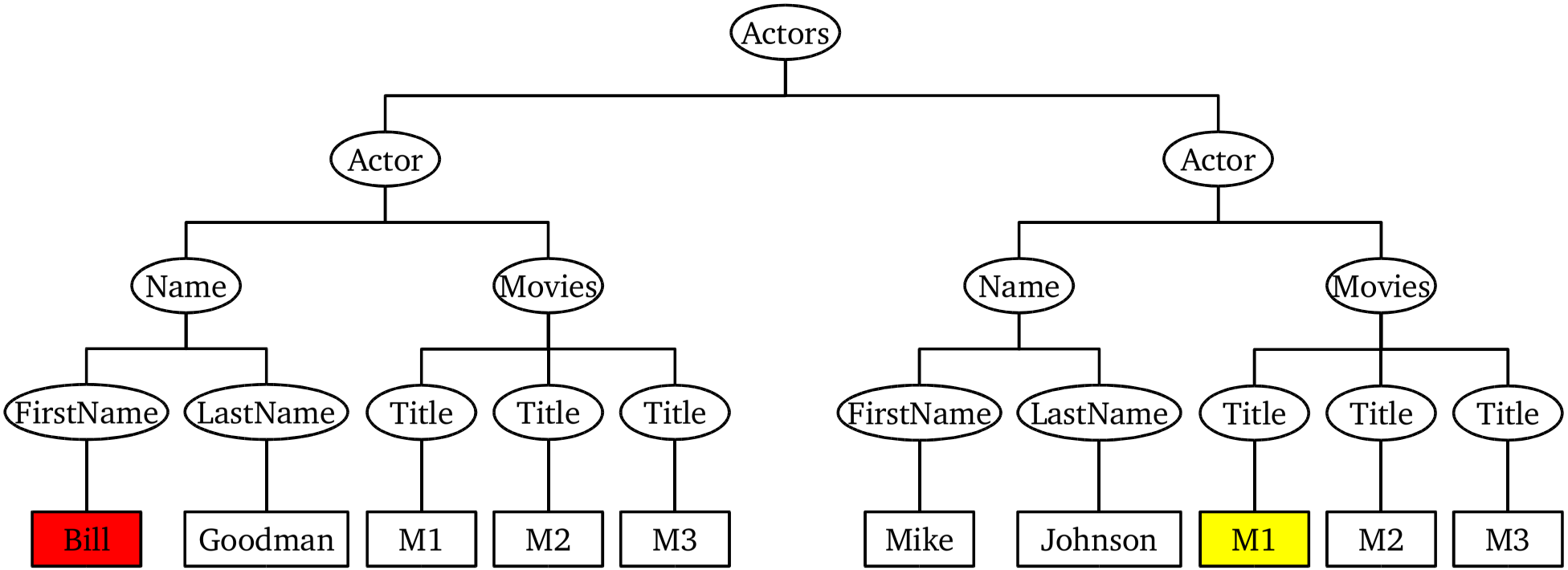

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

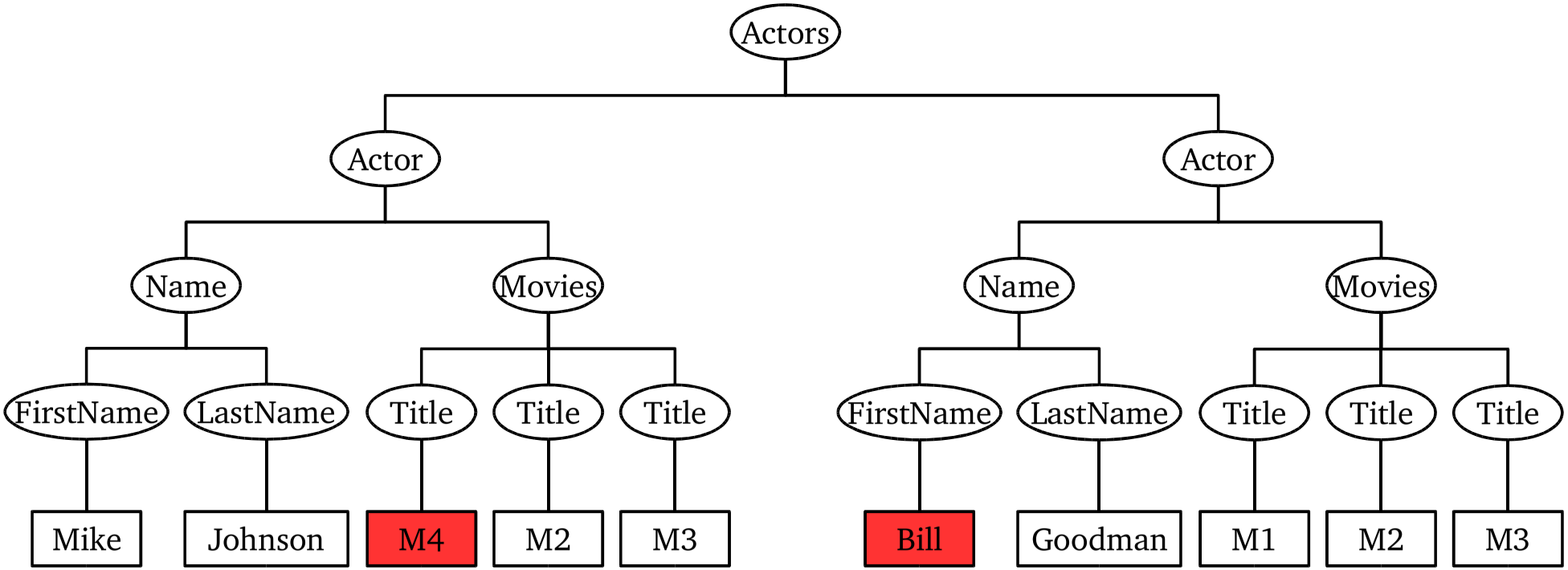

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Goodman
M1
M2
M3
Mike
Johnson
M1
M2
M3

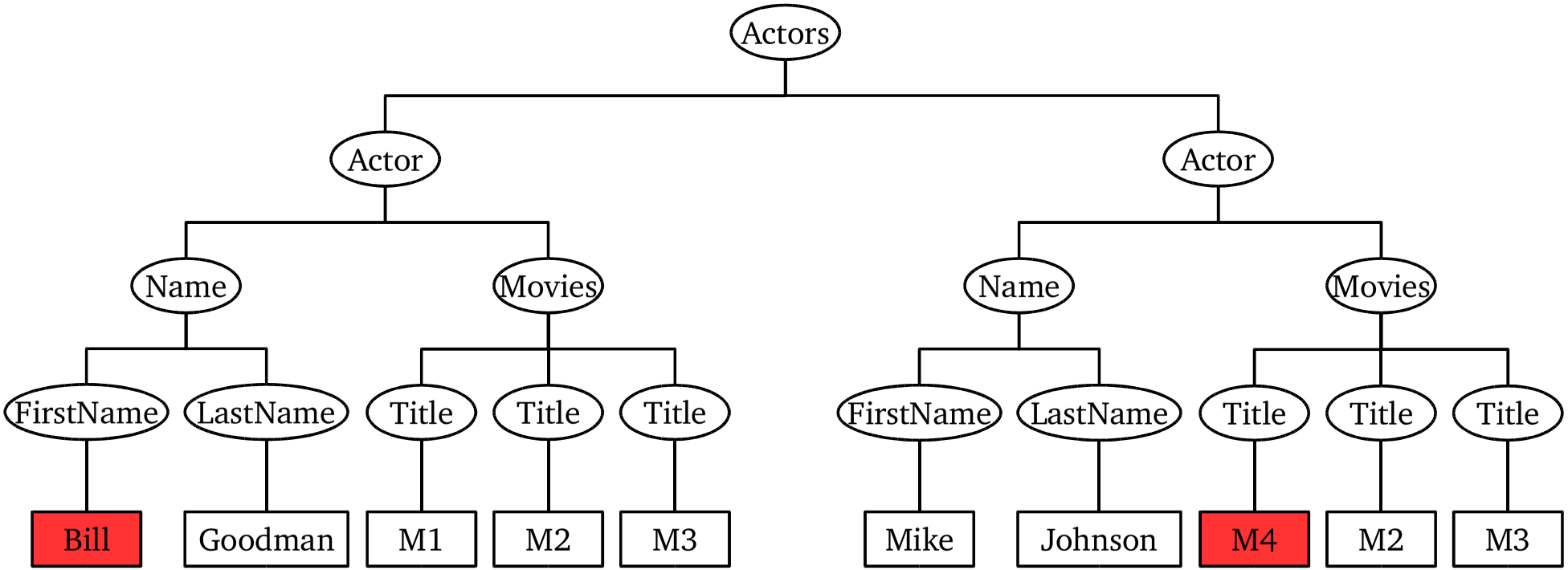

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

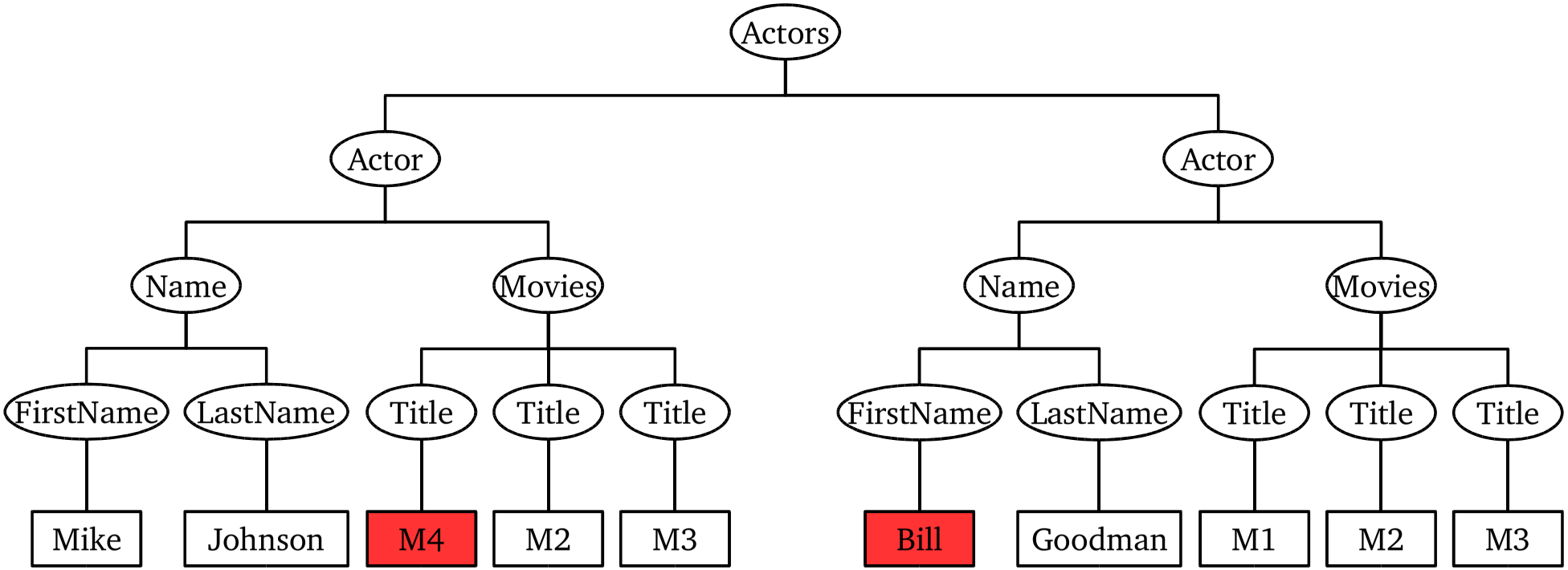

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Bill
Goodman
M1
M2
M3
Mike
Johnson
M1
M2
M3

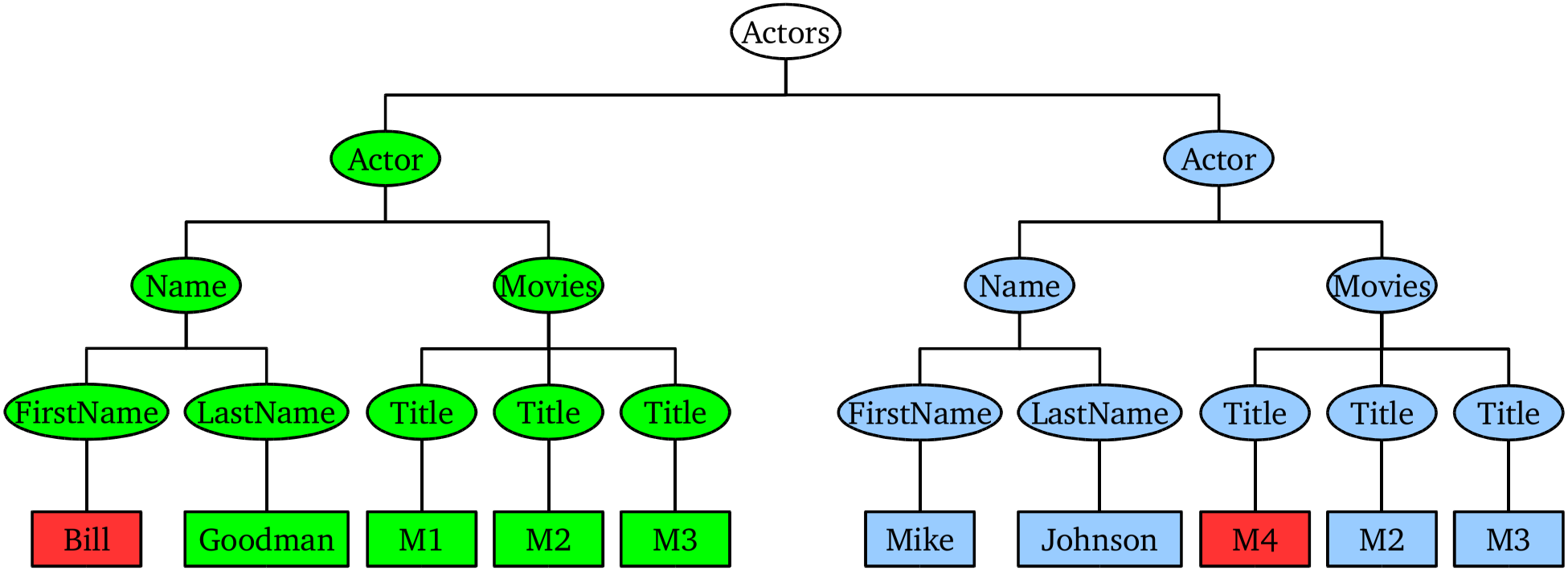

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

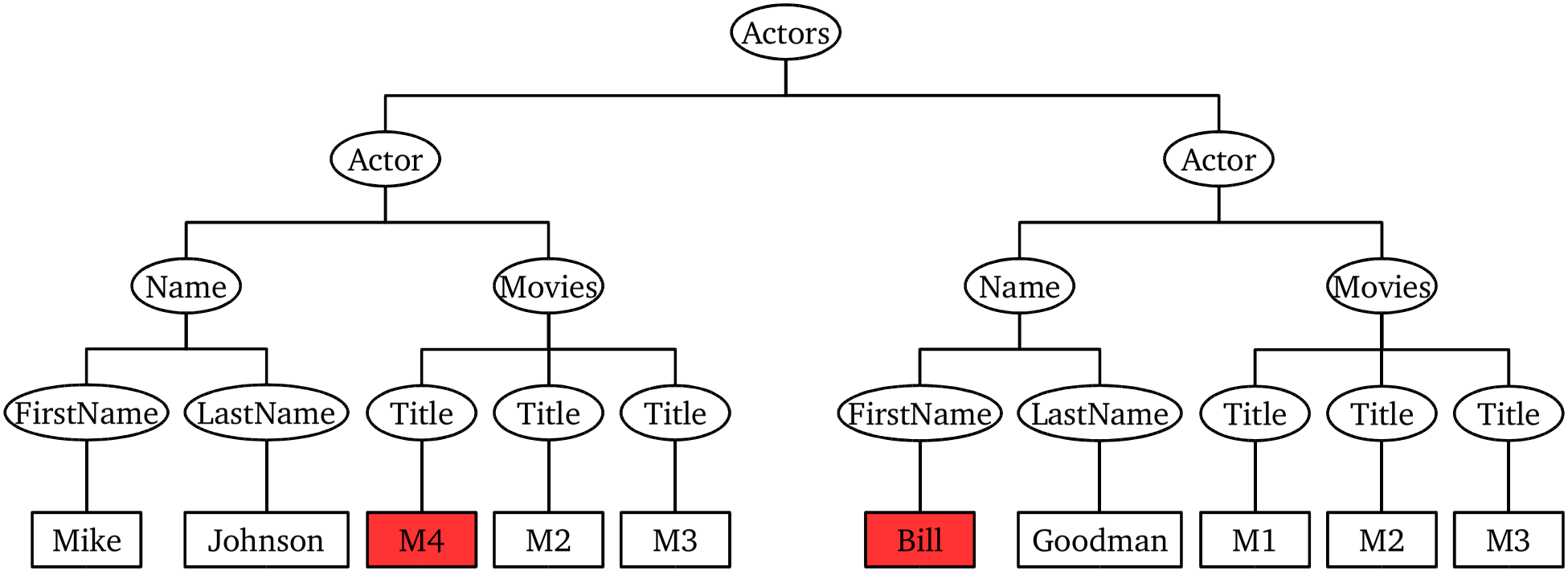

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Bill
Goodman
M1
M2
M3
Mike
Johnson
M1
M2
M3

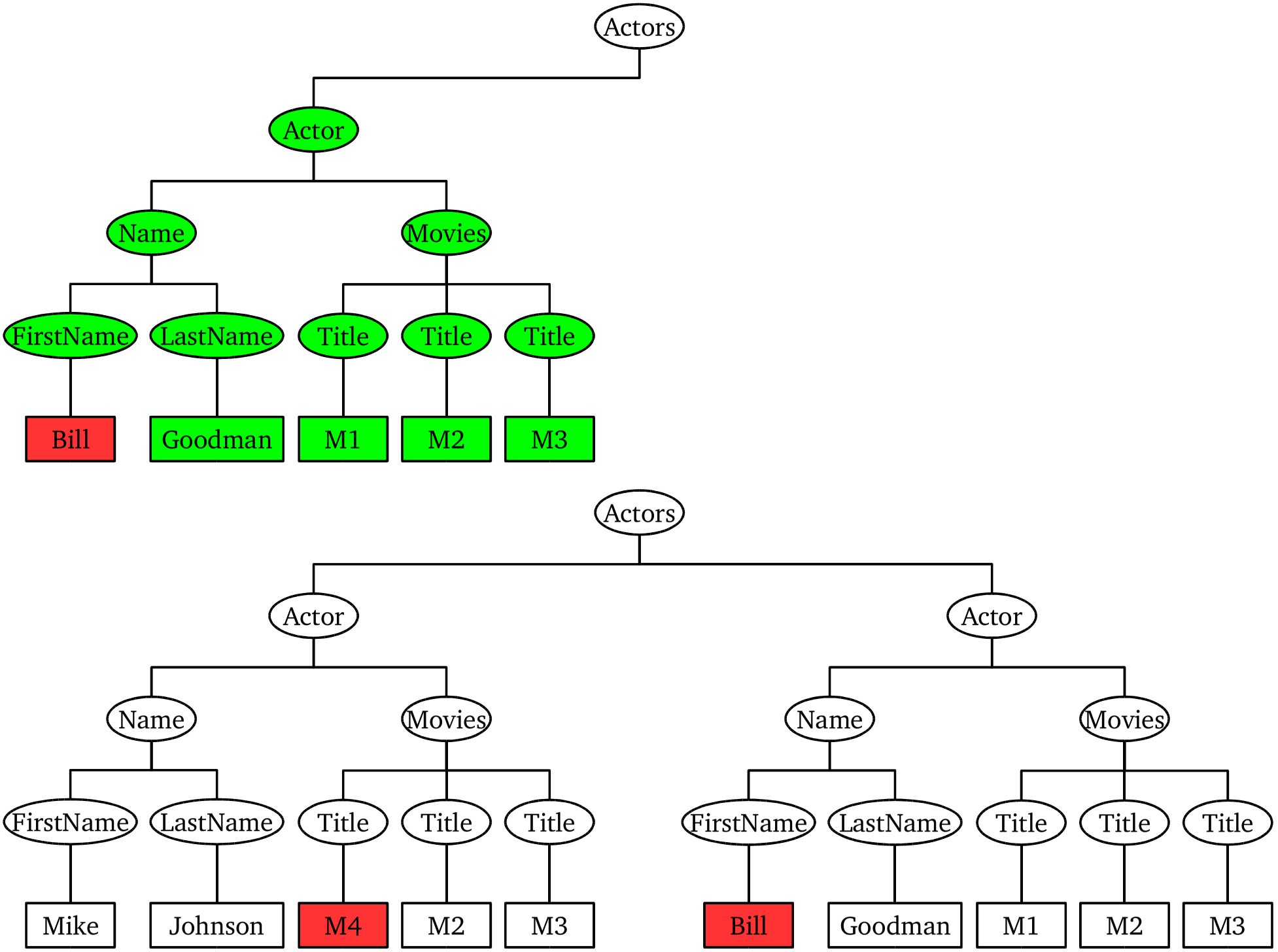

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

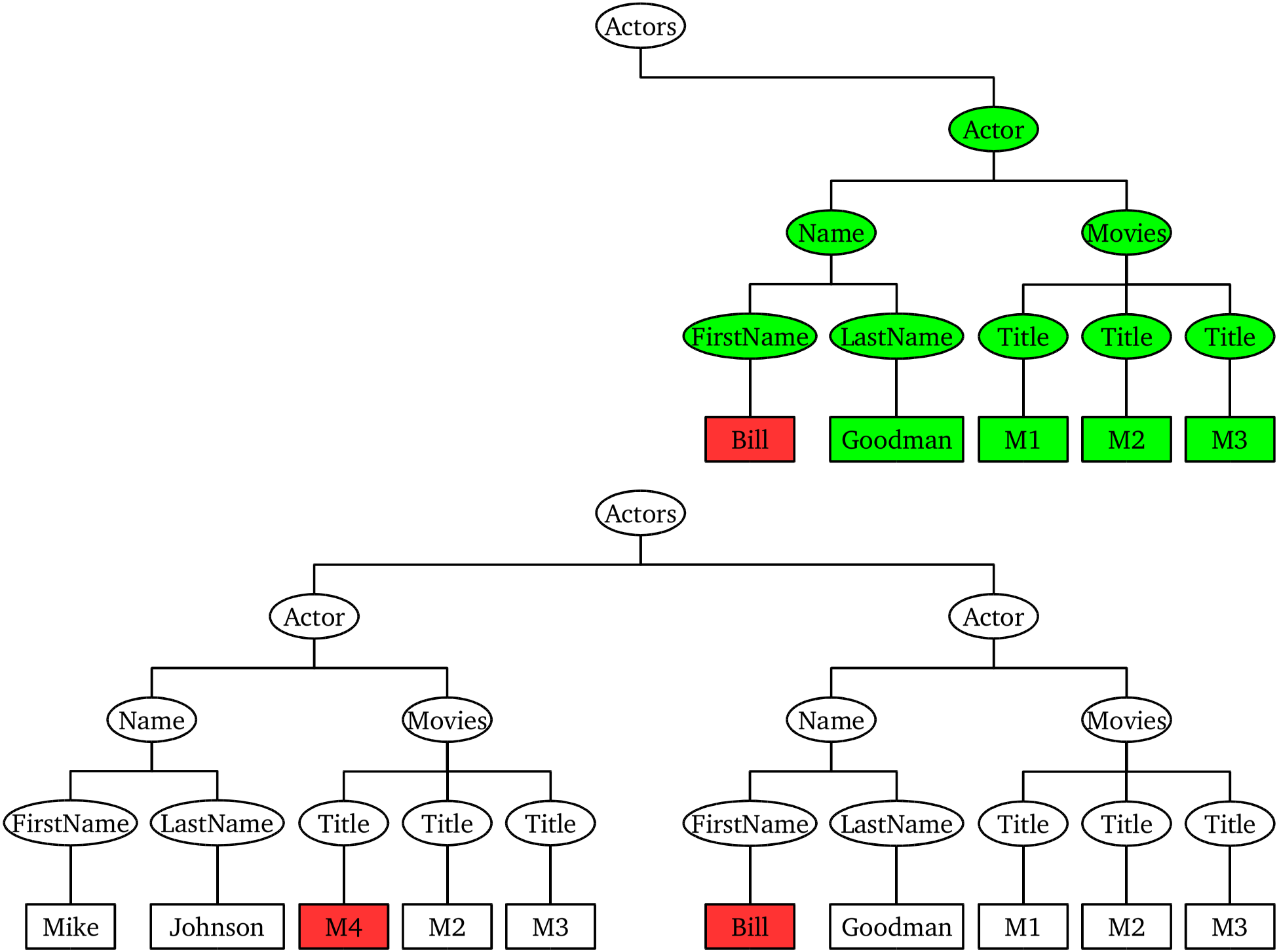

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Bill
Goodman
M1
M2
M3
Mike
Johnson
M4
M2
M3

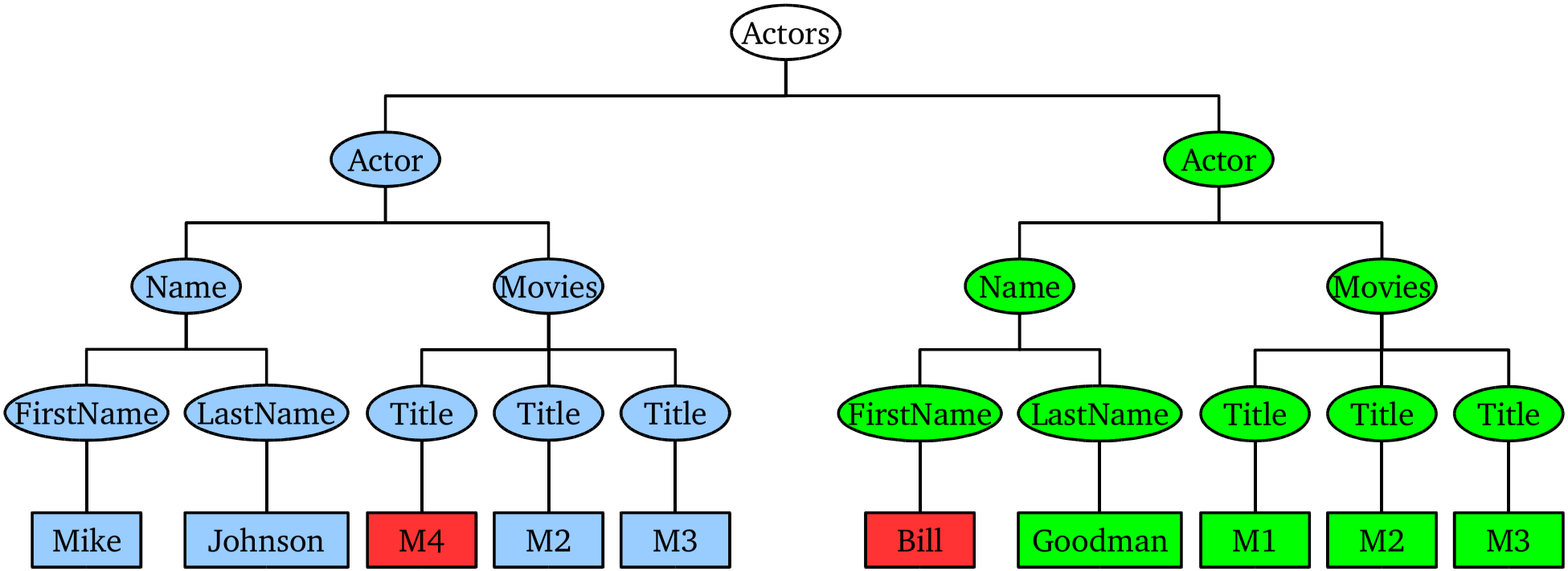

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

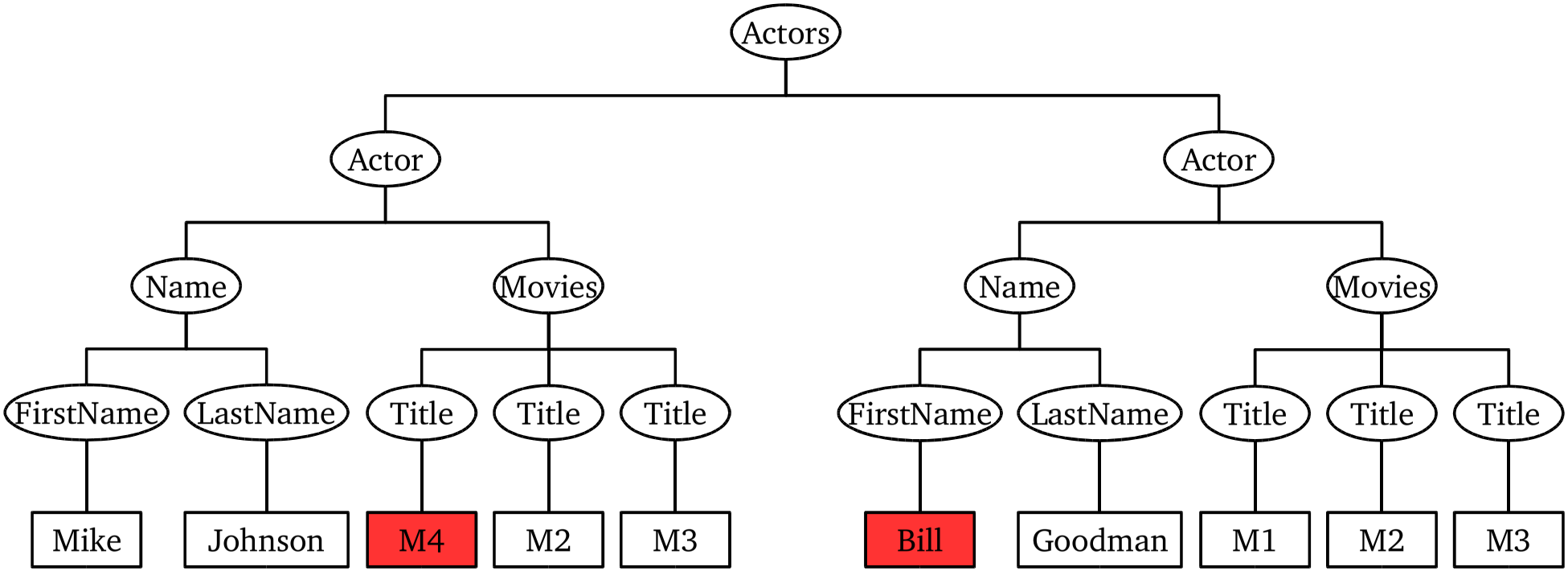

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Bill
Goodman
M1
M2
M3
Mike
Johnson
M4
M2
M3

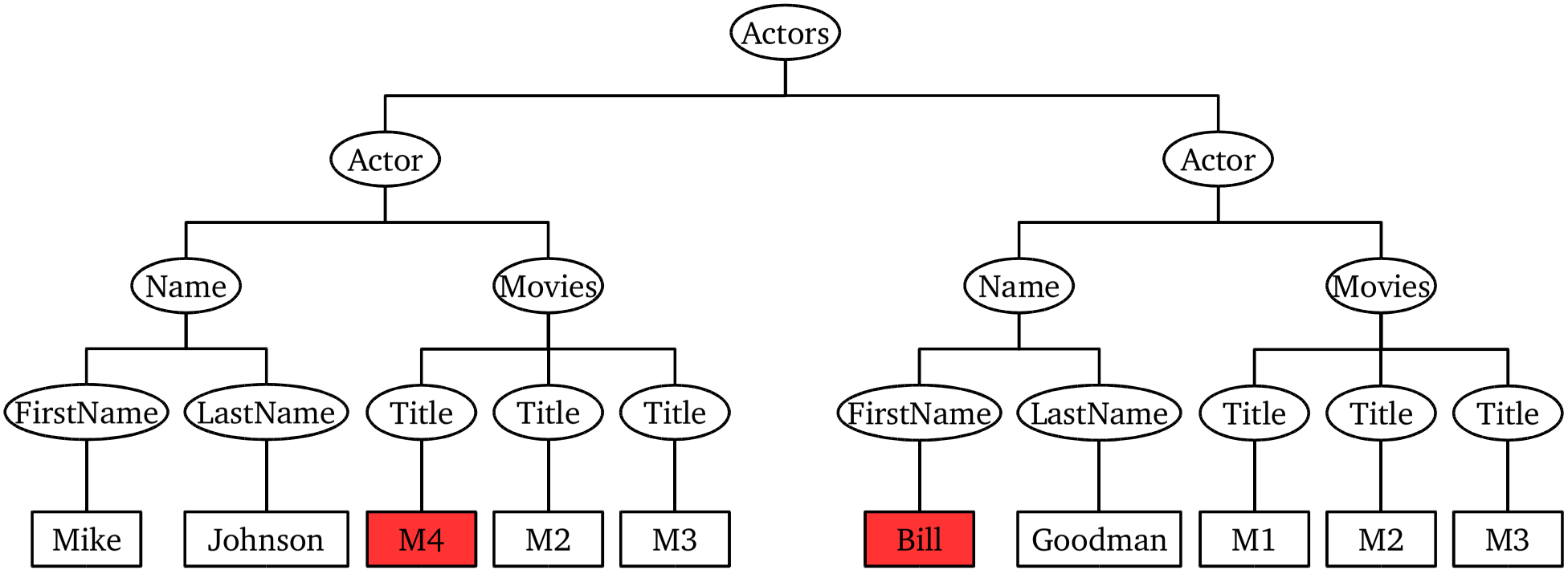

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

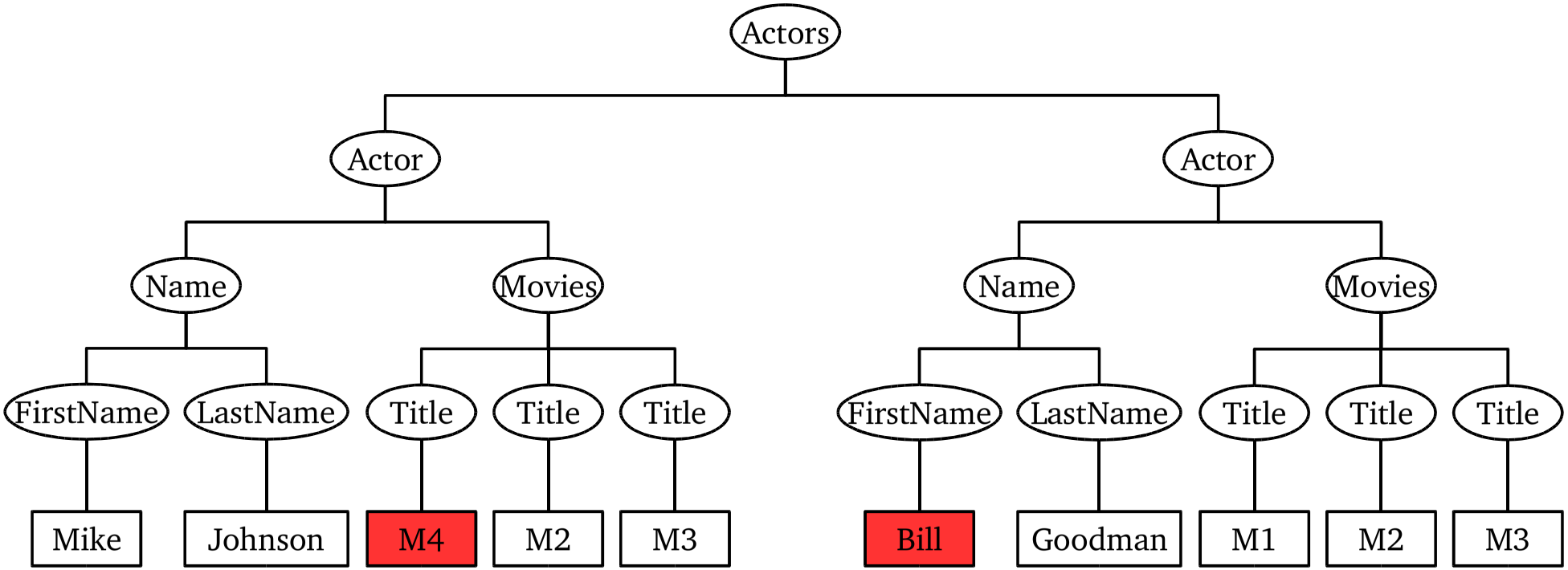

Actors
Actor
Name
Movies
FirstName
LastName
Title
Title
Title
Bill
Goodman
M1
M2
M3
Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3


Actors
Actor
Name
Movies
FirstName
LastName
Title
Title
Title
Bill
Goodman
M1
M2
M3
Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3


Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3


Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

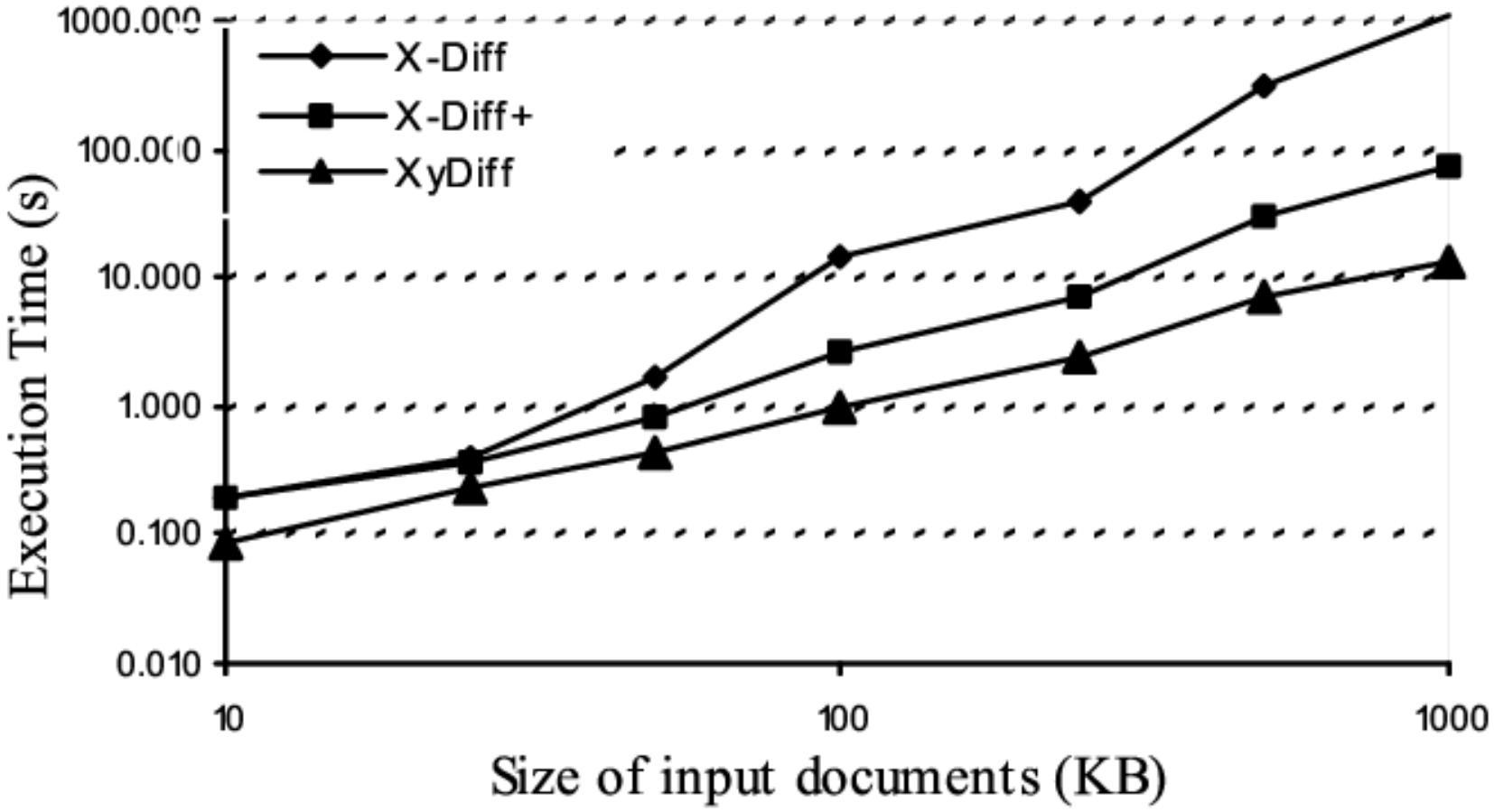

Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3


Actors
Actor
Actor
Name
Movies
Name
Movies
FirstName
LastName
Title
Title
Title
FirstName
LastName
Title
Title
Title
Mike
Johnson
M4
M2
M3
Bill
Goodman
M1
M2
M3

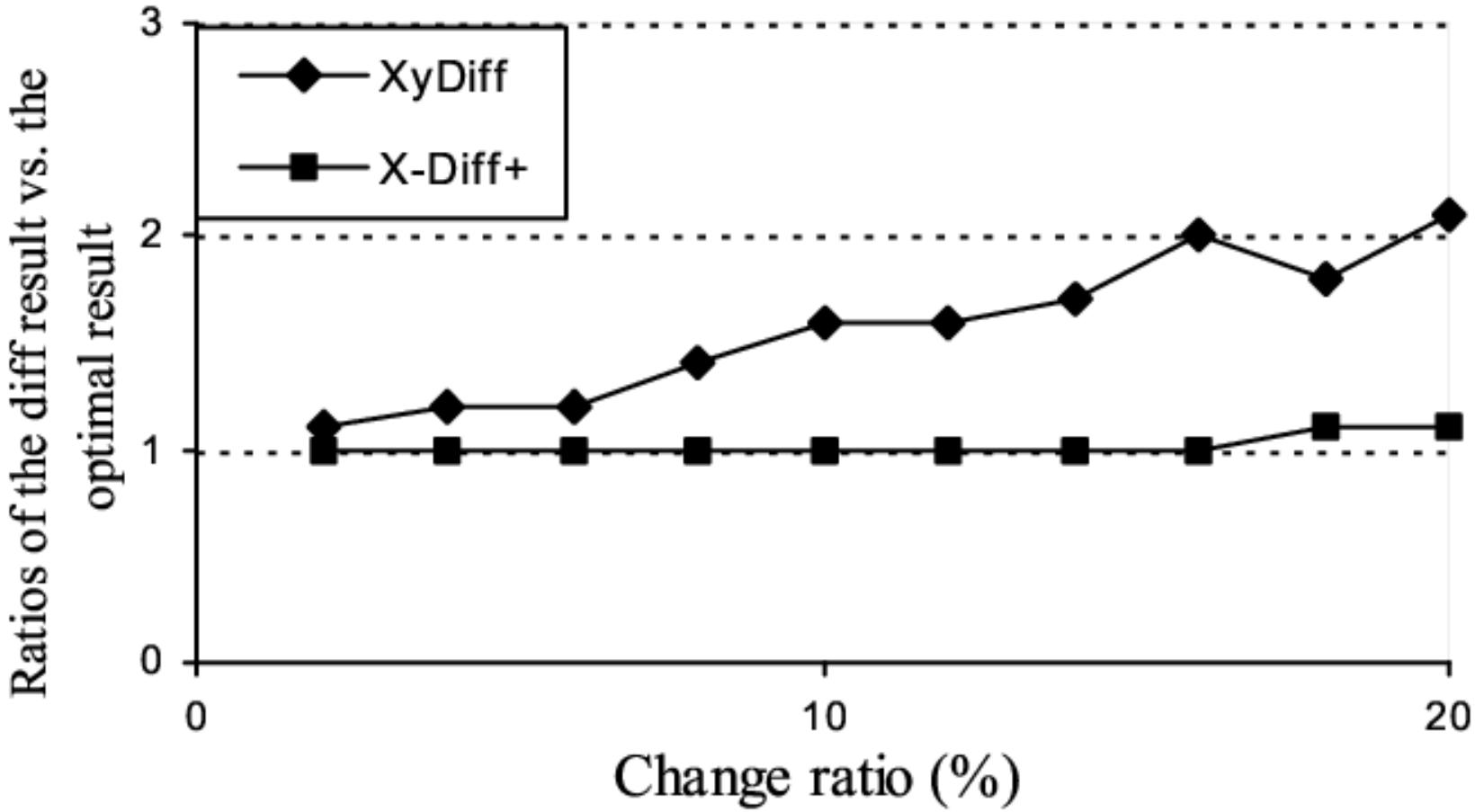
X-Diff

- Developed for database applications
- Considers XML document trees unordered
- Designed to produce optimal diffs
- Change detection on unordered trees is NP-complete
- Needs to compare every node in the old document with every node in the new one
- Significantly slower than XY-Diff

Performance



Quality


Ratios of the diff result vs. the optimal result
3
2
1
0
XyDiff
X-Diff+
0
10
20
Change ratio (%)

Conclusion

- Use of heuristics to save time
- Use of XML structure
- Two diff algorithms for different purposes
- More appropriate than unix diff

EOF