A Fast Index for Semistructured Data

Brian F. Cooper	Department of Computer Science Stanford University
Neal Sample	Department of Computer Science Stanford University
Michael J. Franklin	Computer Science Division University of California
Gisli R. Hjaltason	RightOrder Incorporated
Moshe Shadmon	RightOrder Incorporated

VLDB Conference, 2001

2003.10.7 KINS Cho younggoo

Outline

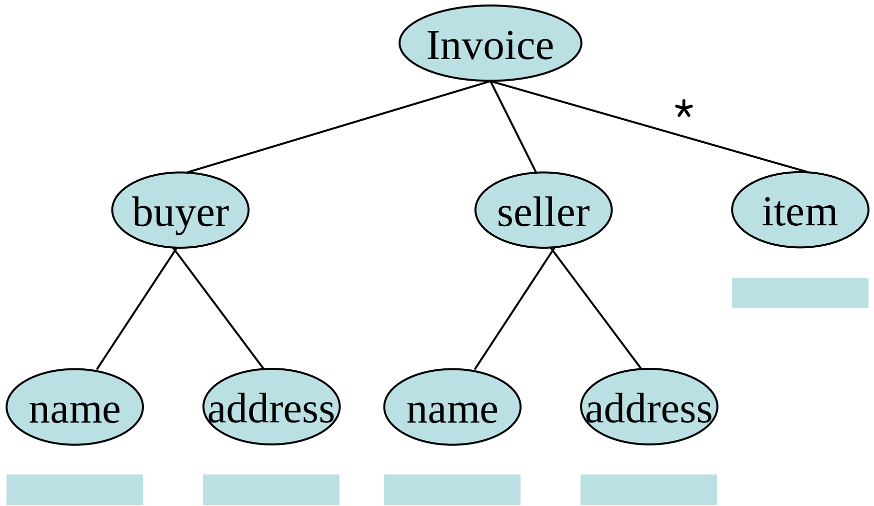
- Background
  - Simple Path Expression
  - General Path Expression
- Introduction
- Index Fabric
  - Overview
  - Trie, Patricia trie
  - Search, Insert, Split
  - Designator
  - Raw Path, Refined Path
- Experiment
  - Basic edge-mapping, STORED
  - Result

Simple Path Expression

A simple path expression specifies a sequence of tags starting from the root of the XML.

“Find invoices where the buyer is ABC Corp”

```
<invoice>
  <buyer>
    <name>ABC Corp</name>
    <address>1 Industrial Way</address>
  </buyer>
  <seller>
    <name>Acme Inc</name>
    <address>2 Acme Rd.</address>
  </seller>
  <item count=3>saw</item>
  <item count=2>drill</item>
</invoice>
```



The Query asks for XML Documents that contain the root-to-leaf path “invoice.buyer.name.’ABC Corp’.”

General Path Expression

$A.(B_1|B_2).C$

results in searches for $A.B_1.C$ and $A.B_2.C$

A.*.C

means find every <C> that has an ancestor <A>

General path expressions are vital for dealing with data that has irregular or changing structure because they allow for alternates, optional tags and wildcards.

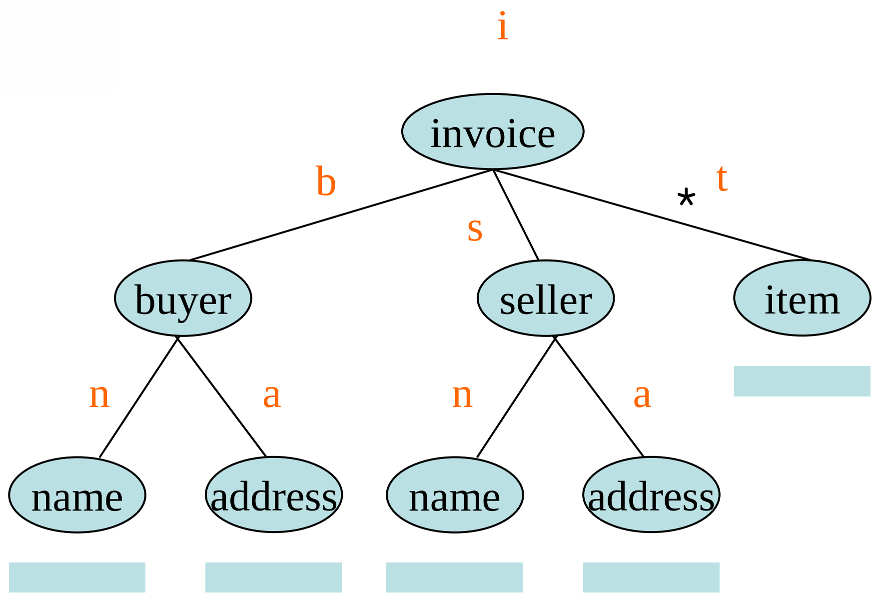
Introduction

- This paper suggests a method that encodes paths as strings, and inserts those strings into a special index that is highly optimized for long and complex keys.
  - *“Index Fabric”*

- This paper discusses how “raw paths” are used to optimize ad hoc queries over semistructured data, and how “refined paths” optimize specific access paths.

Basic Idea

```
<invoice>
  <buyer>
    <name>ABC Corp</name>
    <address>1 Industrial Way</address>
  </buyer>
  <seller>
    <name>Acme Inc</name>
    <address>2 Acme Rd.</address>
  </seller>
  <item count=3>saw</item>
  <item count=2>drill</item>
</invoice>
```



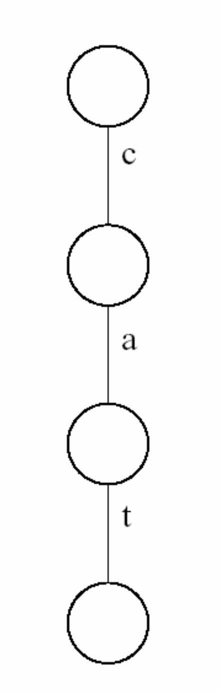
Overview of the index fabric

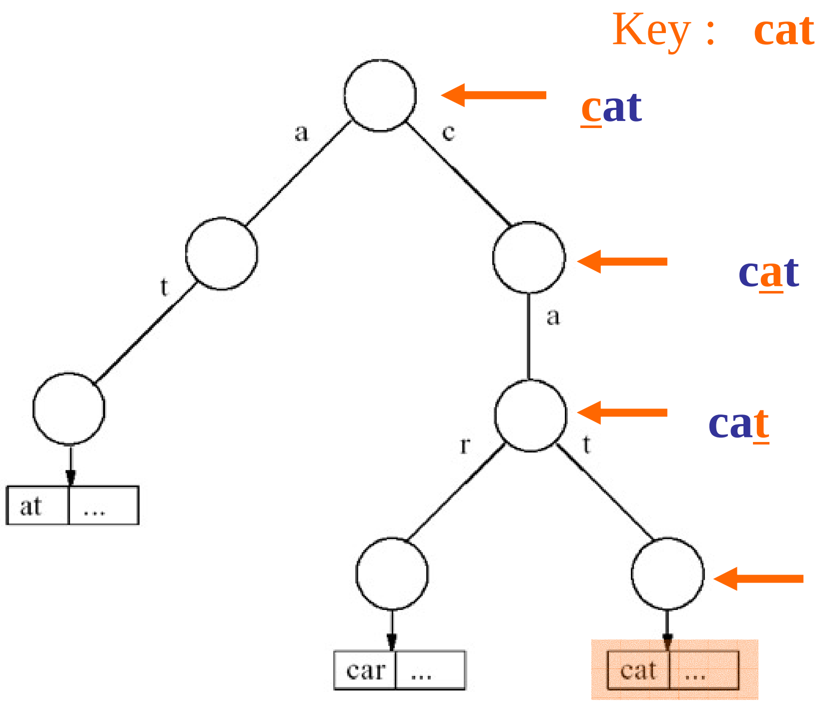
- The Trie is a tree for storing strings
  - There is one node for every common prefix.
  - The strings are stored in extra leaf nodes.

- The Patricia trie is a compact representation of a trie
  - All nodes with one child are merged with their parents.

- The Index Fabric is based on Patricia tries.

Trie



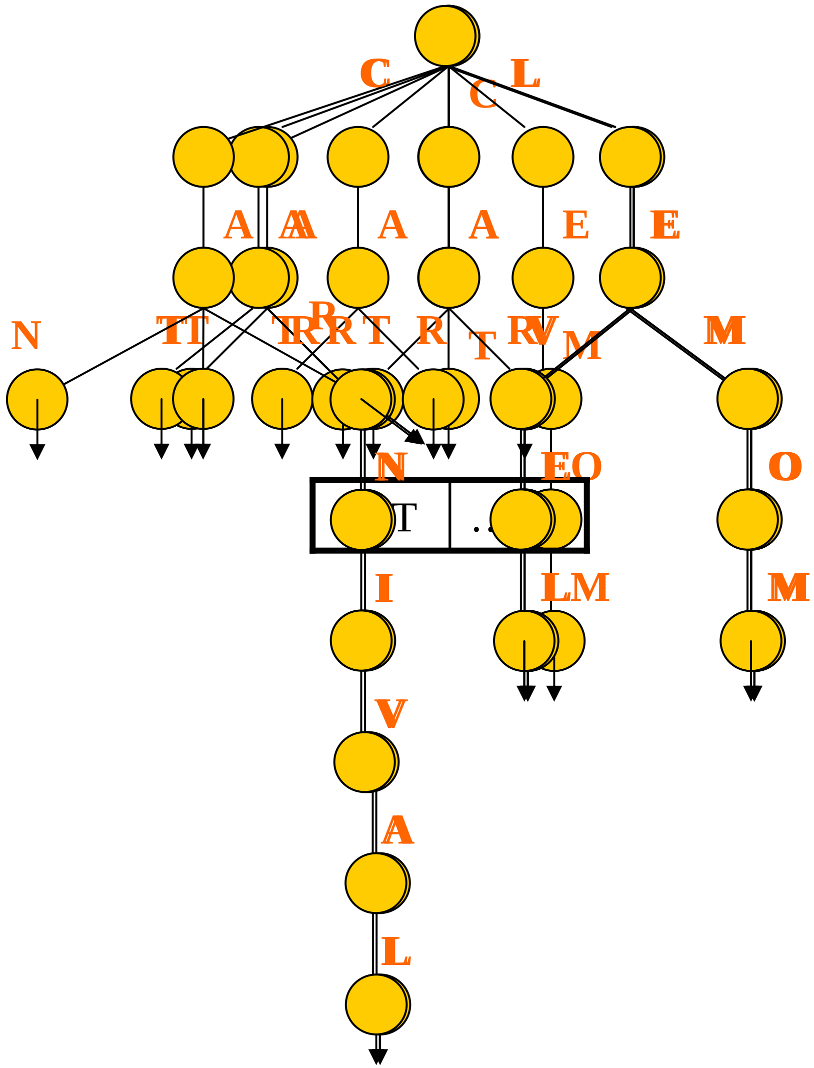


A trie is a tree that stores strings by representing each character in the string as an edge on the path from the root to a leaf.

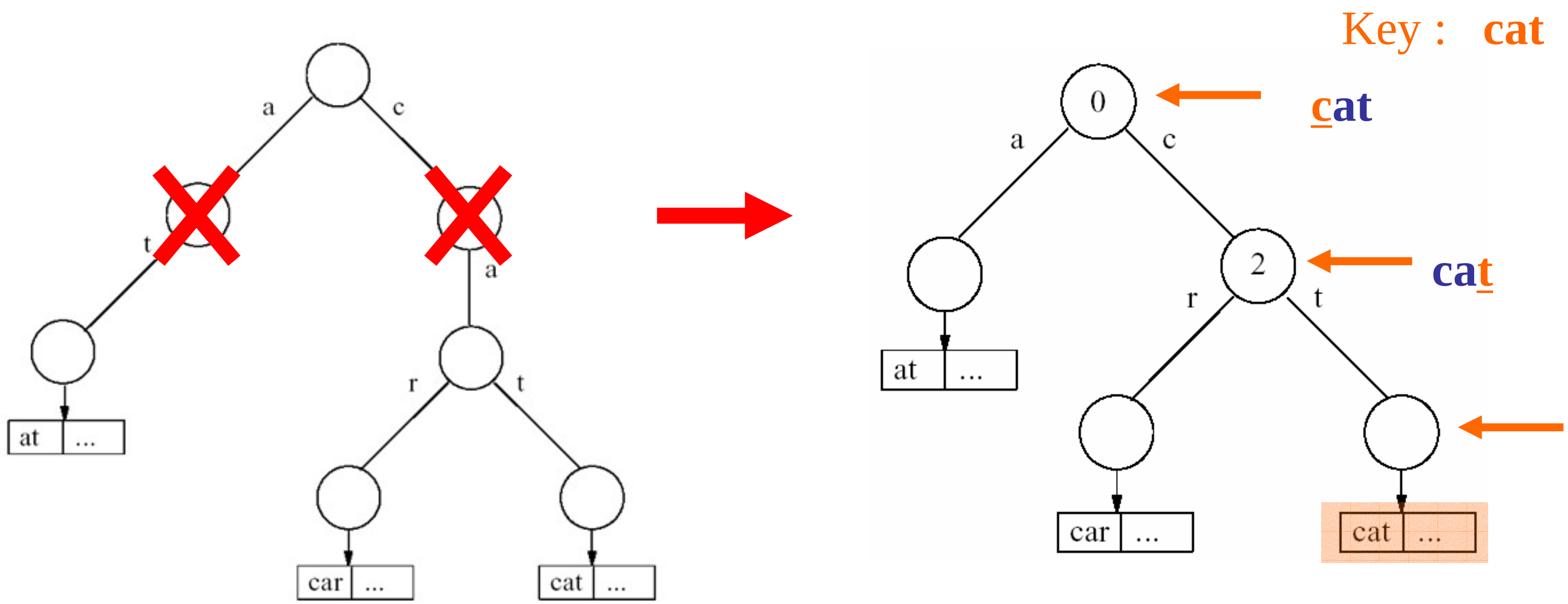
Tries are searched by starting at the root of the tree, and following the edges that correspond to the characters of the search key.

Trie Example

CAT
CAR
LEMON
LEVEL
CARNIVAL
CAN



Patricia Trie

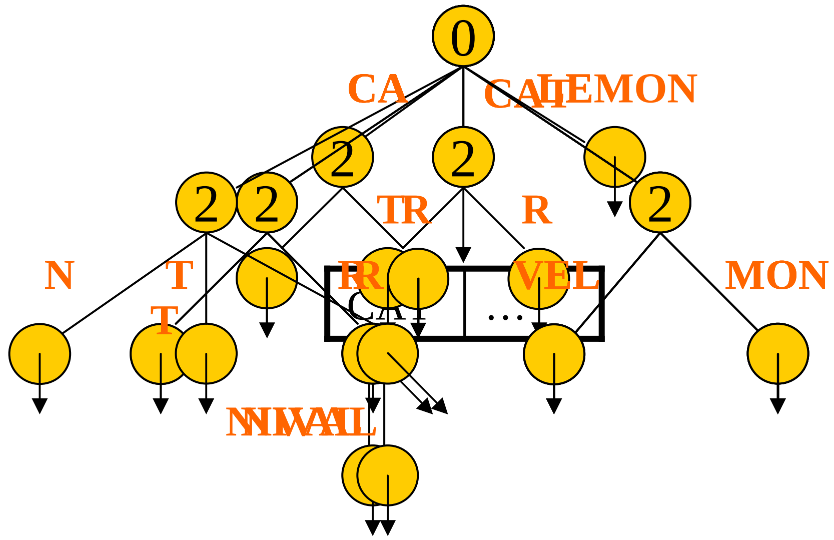


- Patricia tries are a compact form of tries
  - retain the same ability to search for strings.
  - nodes with only one child have been removed.

- The length of keys do not affect the size of the trie.

Patricia Trie Example

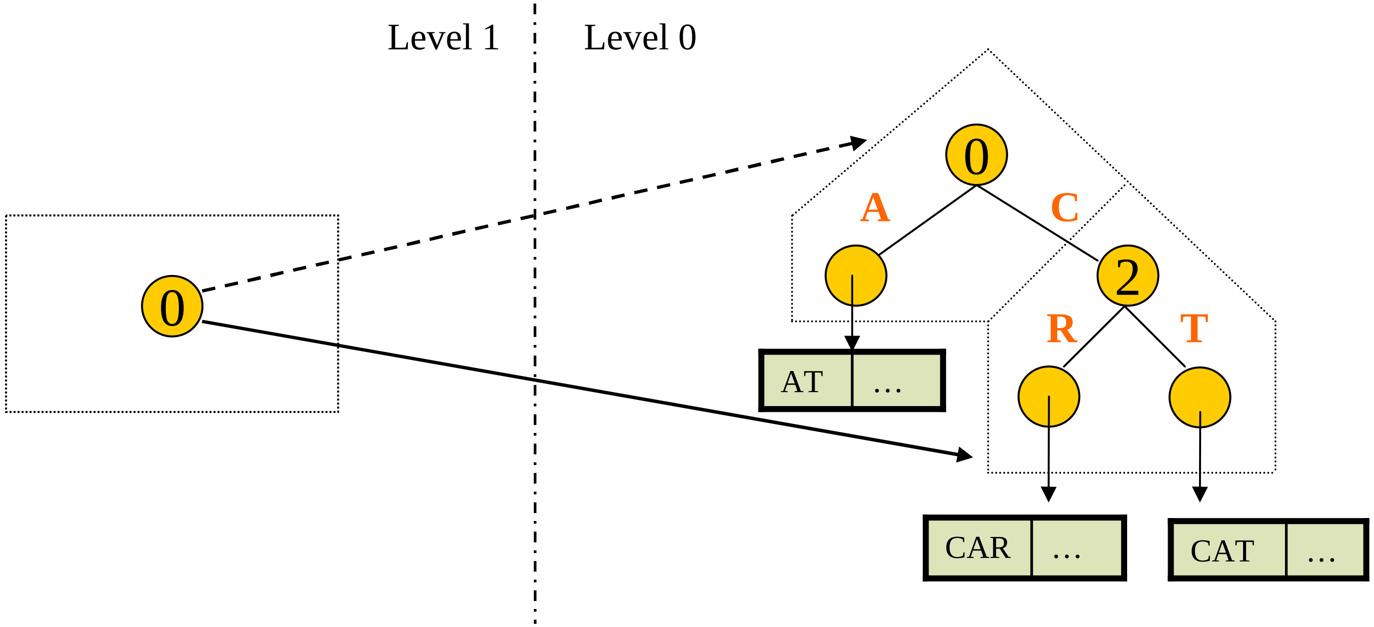
CAT
CAR
LEMON
LEVEL
CARNIVAL
CAN



Index Fabric

Patricia Tries are unbalanced structures. In real databases, this unbalance can become large, and result in performance degradation.

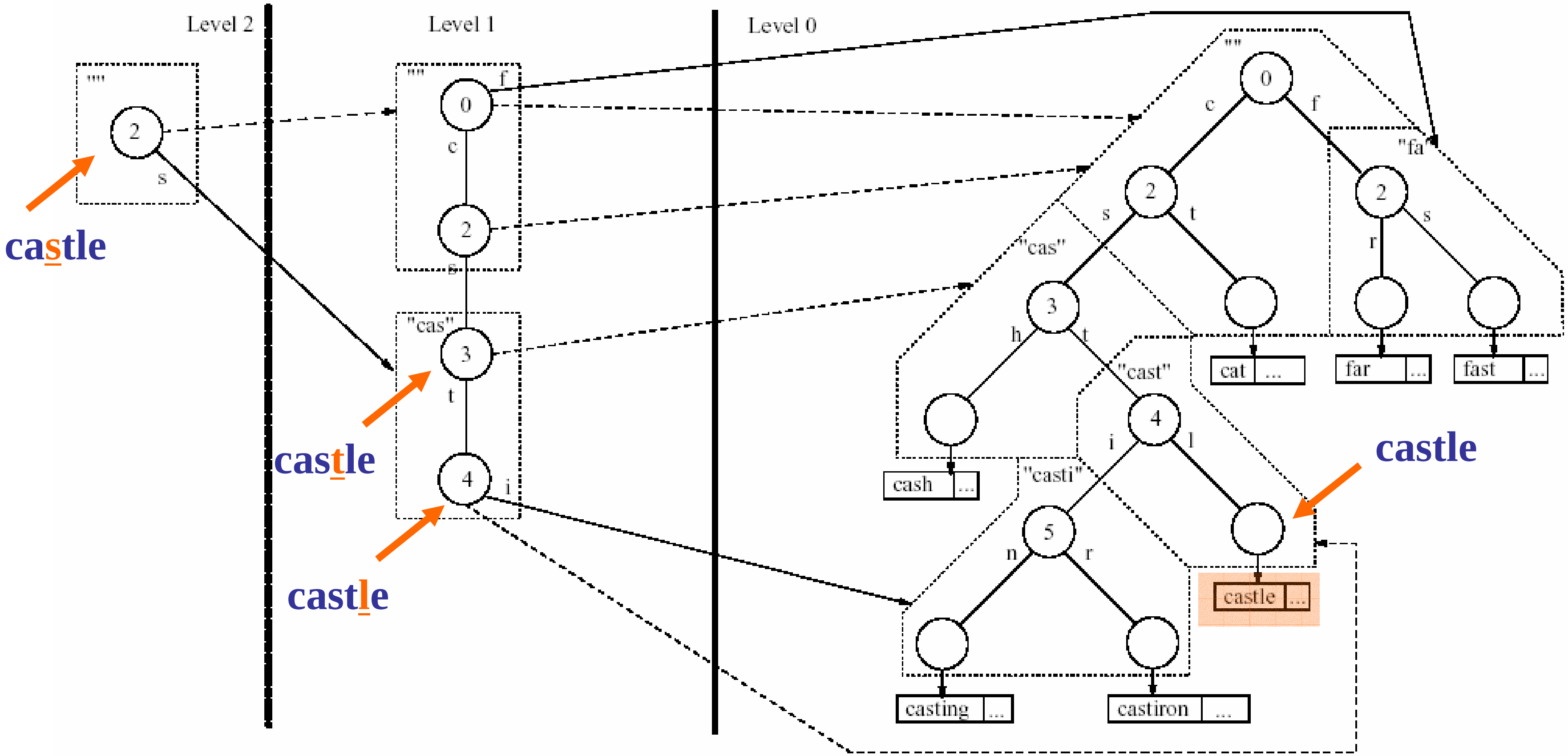
→ Multiple layers into the Patricia trie.



A search over the multilayer index requires one “block read” per layer, the search is “balanced”.

Search

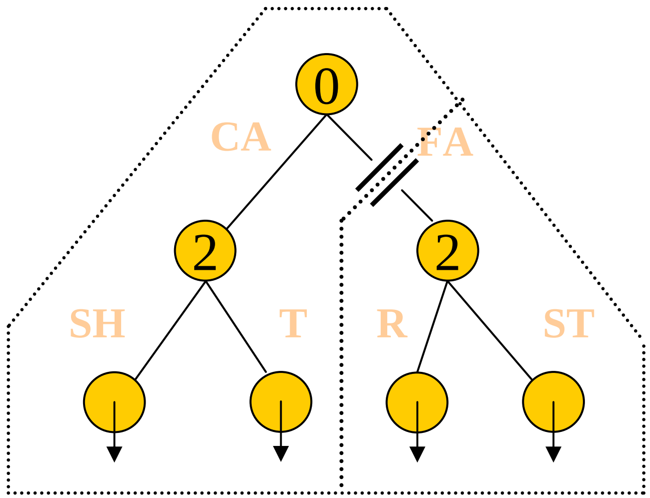
Key : **castle**



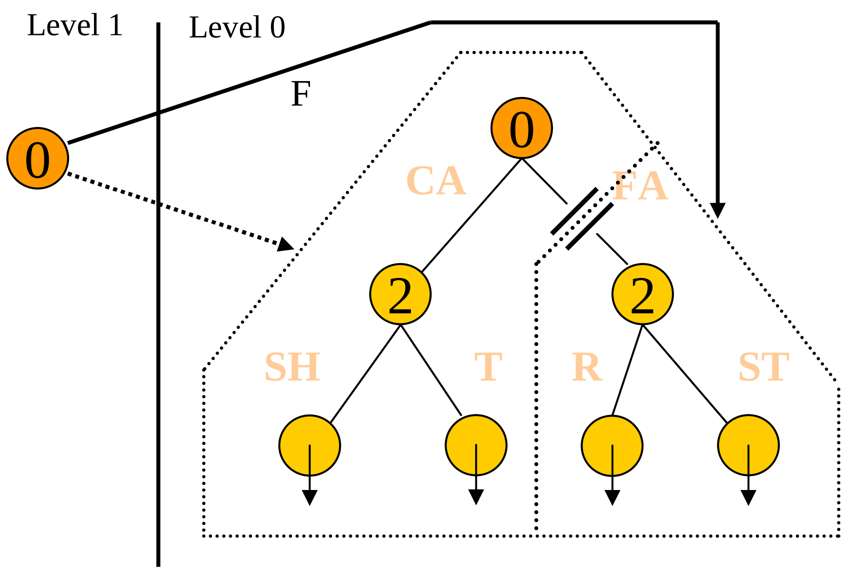
Insertion into index fabric

- Keys are inserted into the multilayer trie using a two step process
  - The key is inserted in the lowest layer PT using the normal PT insertion algorithm.
  - If after creating new nodes there are now too many nodes to fit in the block, *the block must be split*

Splitting a block



Both subtries resulting from the split should be of approximately equal size to maintain good space utilization of the disk blocks.



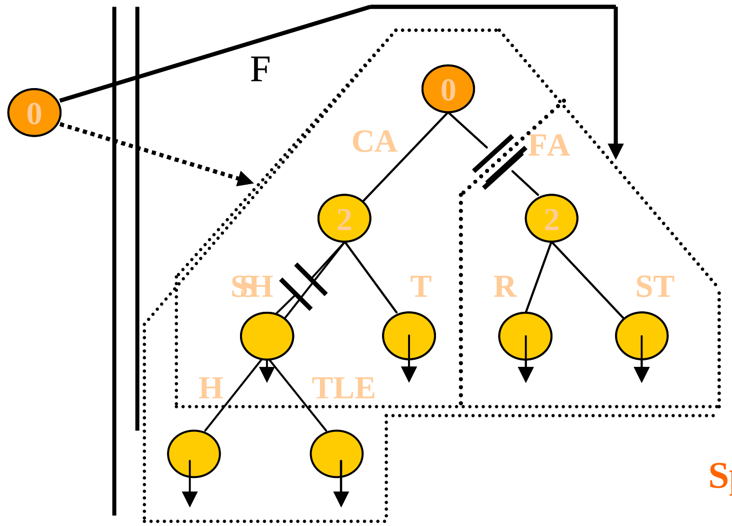
A new layer, level 1 is created.

label

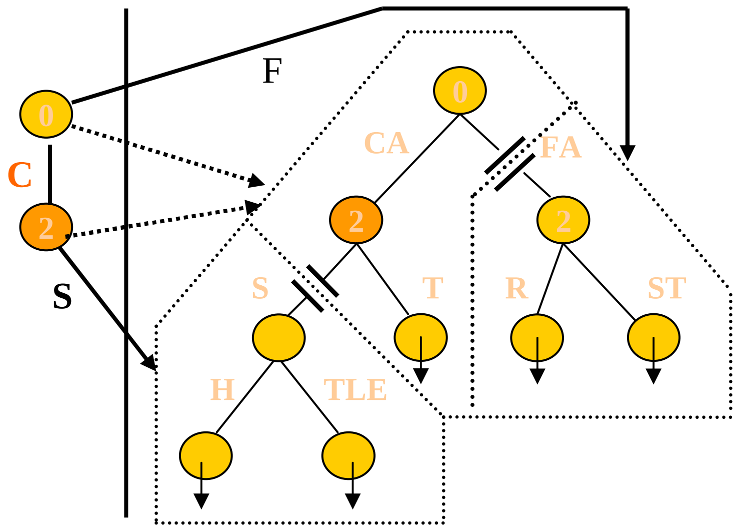
A far pointer (with the same level as the split edge) points to the new block.

A direct (unlabeled) pointer points to the old block.

Splitting a block



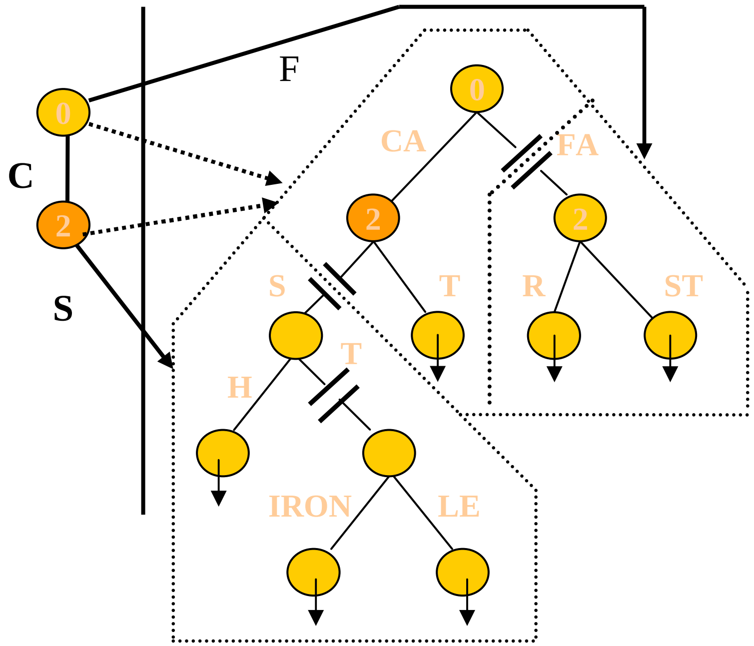
Splitting a block



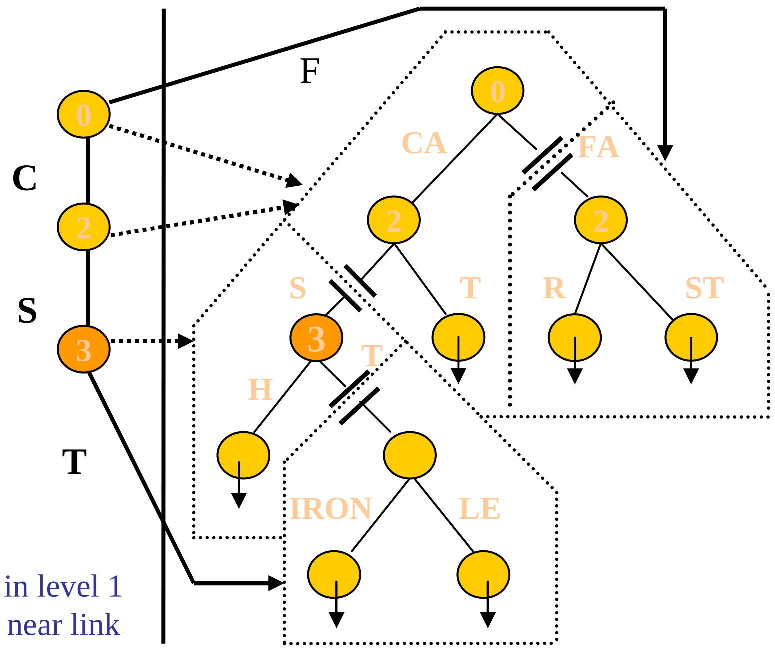
New node in level 1
Near link added

Splitting a block

Castiron added



Splitting a block



New node in level 1
Far link to near link

Designator

The designator encode path expressions as strings. The designator-encoded String is inserted into the Index Fabric.

Data paths are encoded using designators

```
<invoice>
        <buyer>
                <name>ABC Corp</name>
        </buyer>
</invoice>
```

<invoice> ~> I <buyer> ~> B <name> ~> N

The string “**IBNABC Corp**” has the same meaning as the XML fragment.

A mapping is maintained between designators and element tags, called the *designator dictionary*.

Raw Path

Raw paths index the hierarchical structure of the XML by encoding root-to-leaf paths as strings.

XML

```
<invoice>
  <buyer>
    <name>ABC Corp</name>
    <address>1 Industrial Way</address>
  </buyer>
  <seller>
    <name>Acme Inc</name>
    <address>2 Acme Rd.</address>
  </seller>
  <item count=3>saw</item>
  <item count=2>drill</item>
</invoice>
```

Designator Dictionary

<invoice> = **I**
<buyer> = **B**
<name> = **N**
<address> = **A**
<seller> = **S**
<item> = **T**
<phone> = **P**
<count> = **C**
count (attribute) = **C'**

Raw Path

I B N ABC Corp
I B A 1 Industrial Way
I S N Acme Inc
I S A 2 Acme Rd.
I T drill
I T C' 2
I T saw
I T C' 3

Refined Path

We can create a refined path that is tuned for a frequently occurring query over the XML.

Such as "find the invoices where company 'ABC Corp' sold to company 'Acme Inc'." Answering this query involves finding <buyer> tags that are siblings of a < seller> tag within the same <invoice> tag.

```
<invoice>
  <buyer>
    <name>ABC Corp</name>
    <address>1 Industrial Way</address>
  </buyer>
  <seller>
    <name>Acme Inc</name>
    <address>2 Acme Rd.</address>
  </seller>
  <item count=3>saw</item>
  <item count=2>drill</item>
</invoice>
```

First, we assign a designator, such as "Z," to the path and create a key of the form "Z ABC Corp Acme Inc."

Experimental Setup

- All experiments used
  - the same installation of the RDMBS,
  - Pentium III 866MHz, 512MB
- To evaluate performance, an XML data set is indexed using both the Index Fabric and the DBMS’s native B-tree.
- Two different methods of indexing the XML via the RDMBS are used.
  - Basic edge mapping
  - STORED

Basic edge-mapping

The basic edge-mapping treats the XML as a set of nodes and edges.
The database has two tables, roots(id, label) and edges(parentid, childid, label).

<book><author>Jane Doe</author></book>

id	label
0	book

parentid	childid	Label
0	1	Author
1	NULL	Jane Doe

The following key-compressed B-tree indexes are created for experiments.

- An index on roots(id), and an index on roots(label).
- An idex on edges(parentid), and index on edges(childid), and an index on edges(label).

STORED

STORED system uses data mining to extract schemas from the data based on frequently occurring structures.

The extracted schemas are used to create "*storage-mapped tables*".

Most of the data can be stored in the *storage-mapped tables*, while more irregularly structured data must be stored in *overflow buckets*, similar to the edge mapping.

STORED

The *SM tables* identified for the DBLP data

- *inproceedings*, *conference papers*, *articles*, *journal papers*.

Conference and *journal paper* that does not fit into the SM tables is stored in *overflow buckets* along with other types of publications.

The following key-compressed B-tree indexes are created.

- An index on each of the author attributes in the inproceedings and Articles SM tables.
- An index on the booktitle attribute in the inproceedings table.
- An index on the id attribute of each SM table.

Data set & Query

The data set is the DBLP.

over 180,000 documents, 72Mb of data

grouped into eight classes (journal, article, book, etc.).

```
<article key="Codd70">
        <author>E. F. Codd</author>
        <title>A Relational Model of Data for Large Shared Data Banks.</title>
        <volume>13</volume>
                …
        <ee>db/journals/cacm/Codd70.html</ee>
        <cdrom>CACMs1/CACM13/P377.pdf</cdrom>
</article>
```

Query	Description
A	Find books by publisher
B	Find conference papers by author
C	Find all publications by author
D	Find all publications by co-authors
E	Find all publications by author and year

48 different publisher
7,000 different author
10,000 different author
10,000 different pair
10,000 different pair

Result

Query	Description
A	Find books by publisher
B	Find conference papers by author
C	Find all publications by author
D	Find all publications by co-authors
E	Find all publications by author and year

A: book.publisher.X
B: inproceeding.author.X
C: *.author.X
D: *.author.X & author.Y
E: *.author.X & year.Y

I/O - Blocks

	Edge Map		STORED		Raw path		Refined path	
	value	Δ	value	Δ	value	Δ	value	Δ
A	416	1.0	370	1.1	13	**32.0**	-	-
B	68788	1.0	26490	2.6	6950	**9.9**	-	-
C	69925	1.0	61272	1.1	34305	2.0	20545	**3.4**
D	353612	1.0	171712	2.1	89248	4.0	17337	**20.4**
E	327279	1.0	138386	2.4	113439	2.9	16529	**19.8**

Time - Seconds

Edge Map		STORED		Raw path		Refined path	
value	Δ	value	Δ	value	Δ	value	Δ
6	1.0	4	1.5	0.83	**7.2**	-	-
1017	1.0	293	3.5	81	**12.6**	-	-
1056	1.0	649	1.6	397	2.7	236	**4.5**
5293	1.0	2067	2.6	975	5.4	208	**25.4**
4835	1.0	1382	3.5	1209	4.0	202	**23.9**