TIMBER:
A Native XML Database

H. V. Jagadish

Tree-structured native XML database

Implemented at the University of

Michigan by

Bright (and sometimes even brilliant)

Energetic

Researchers

WORK IN PROGRESS!!

Acknowledgements

- Project funded by NSF
- Many collaborators on various pieces
  - Students
    - Shurug Al-Khalifa
    - Andrew Nierman
    - Stelios Paparizos
    - Keith Thompson
    - Melanie Wu
  - Others
    - Nick Koudas
    - Laks Lakshmanan
    - Jignesh Patel
    - Divesh Srivastava

Outline

- Introduction
- Logical basis – Tree algebra
- TIMBER system overview
- Selected specifics

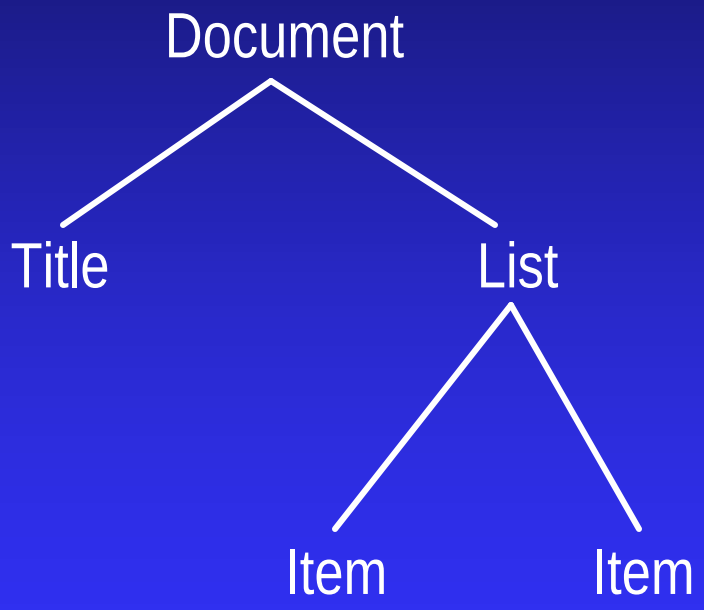
What is XML?

- eXtensible Markup Language.
- ASCII file.
- Structural content over the web.
- Similar to HTML.
  - XML tags represent semantics not style.
- Similar to SGML.
  - XML is simpler.

XML has a tree structure

A Structured XML Fragment

```
<doc>
  <title> Pooh's List </title>
  <list>
    <item> eeyore </item>
    <item> tigger </item>
  </list>
</doc>
```



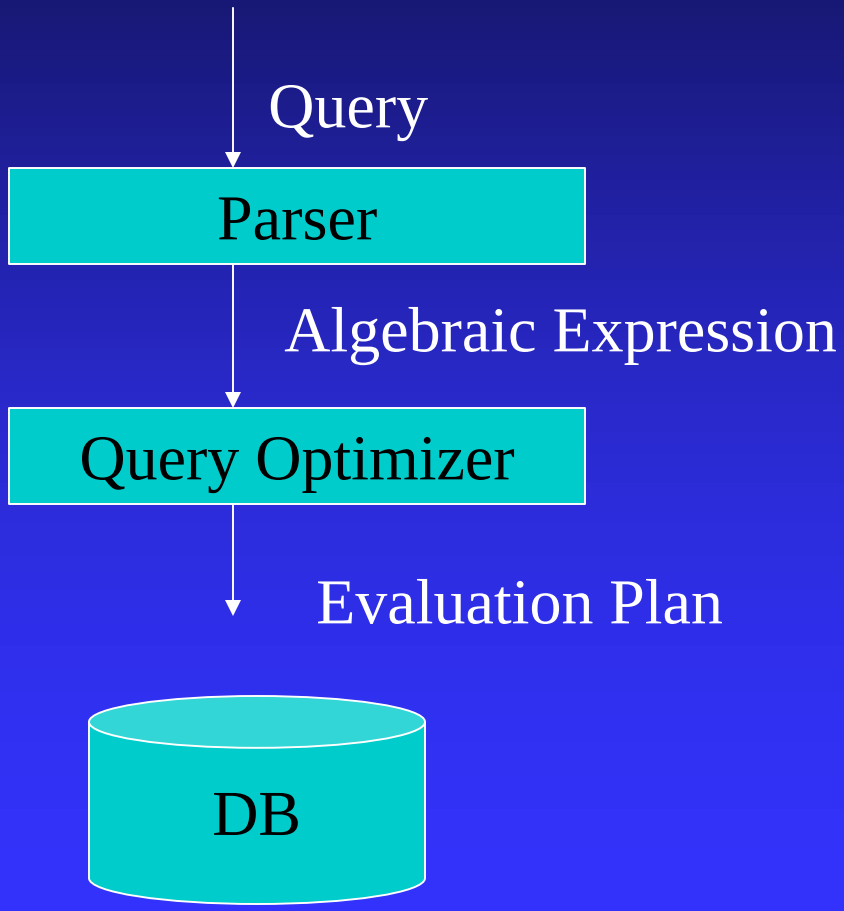
XML in Databases

- Storing XML
  - Layer on existing products.
  - Native implementation.
    - TIMBER.

Outline

- Introduction
- Logical basis – Tree algebra
  - Desiderata
  - TAX overview
- TIMBER system overview
- Selected specifics

Database Architecture



Importance of Algebra

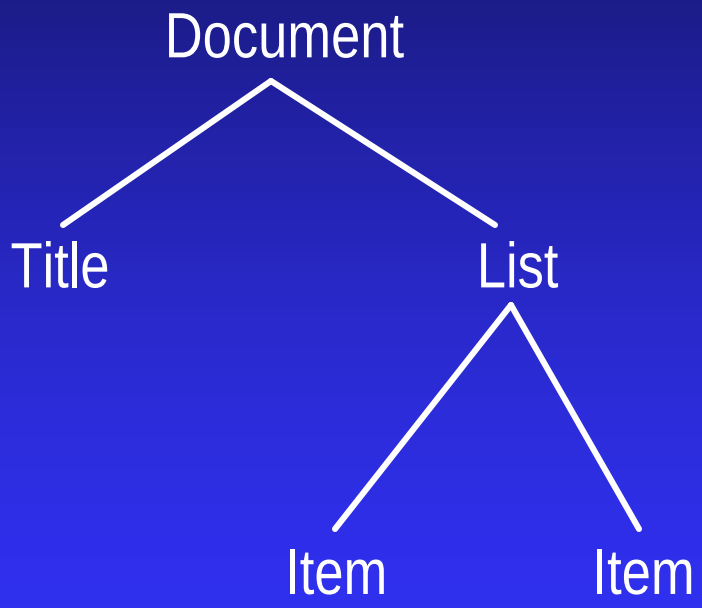
- Accesses must be specified “set-at-a-time” for purposes of efficiency
- Choice of evaluation plan can make orders of magnitude difference in query cost

=> Set-oriented algebra essential

Choosing unit of Manipulation

Can be

- A tree (document or document fragment)
- A node in the tree (an XML element)



Pros and Cons of Choices

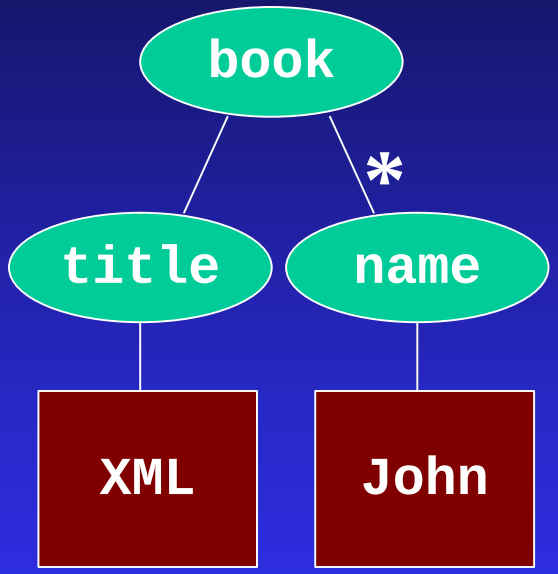
- Manipulating trees is messy
  - Trees have variations in structure
  - Will frequently need parts of a tree
  - Some trees can be very large
- Manipulating nodes is insufficient
  - Loses tree structure

No Choice: Manipulate Trees

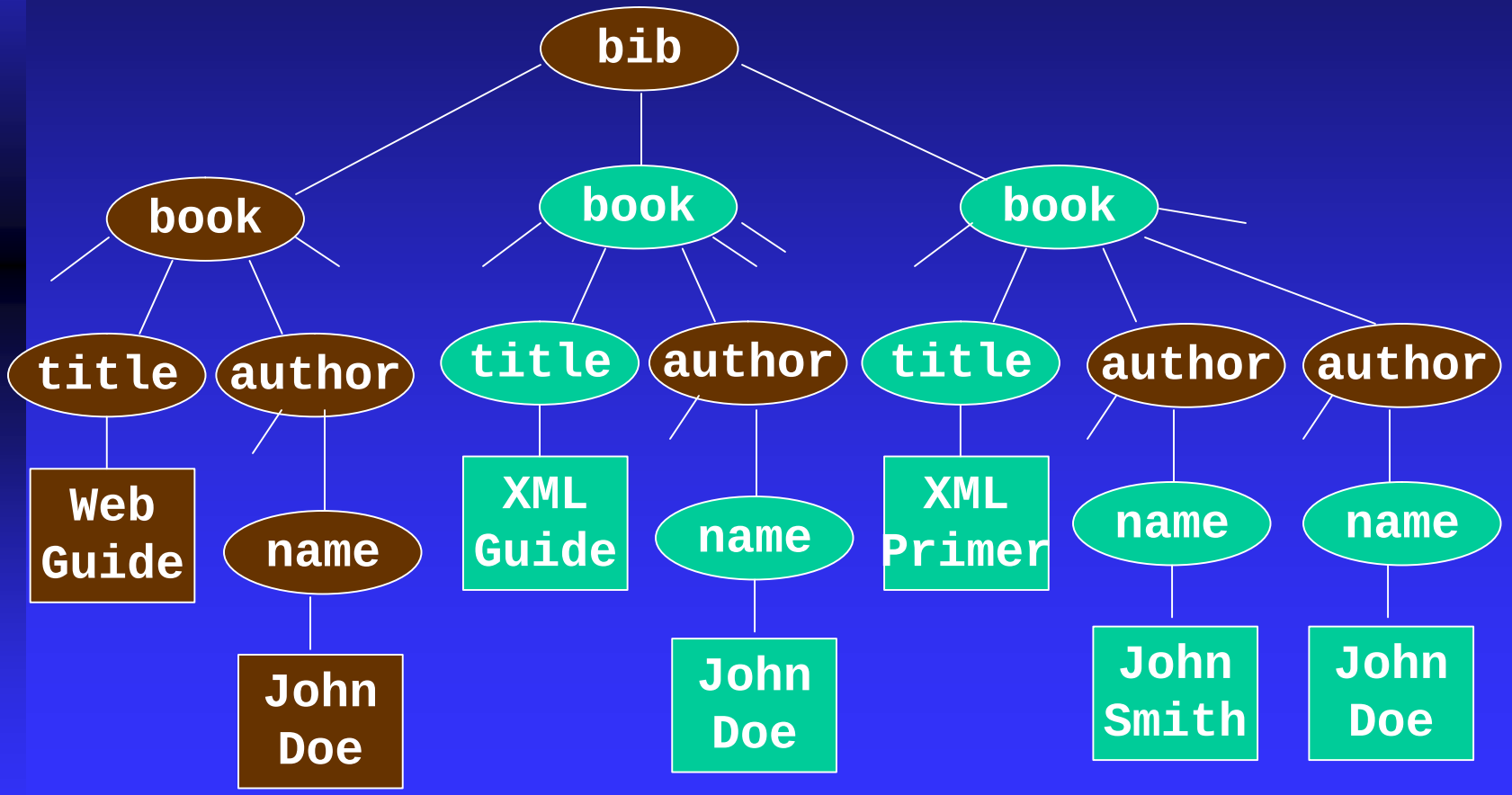
- Address Challenges
  - Trees have variations in structure
    - Pattern Tree Specification
  - Will frequently need parts of a tree
    - Witness Tree for Selection
  - Some trees can be very large
    - Process identifiers

Pattern Tree

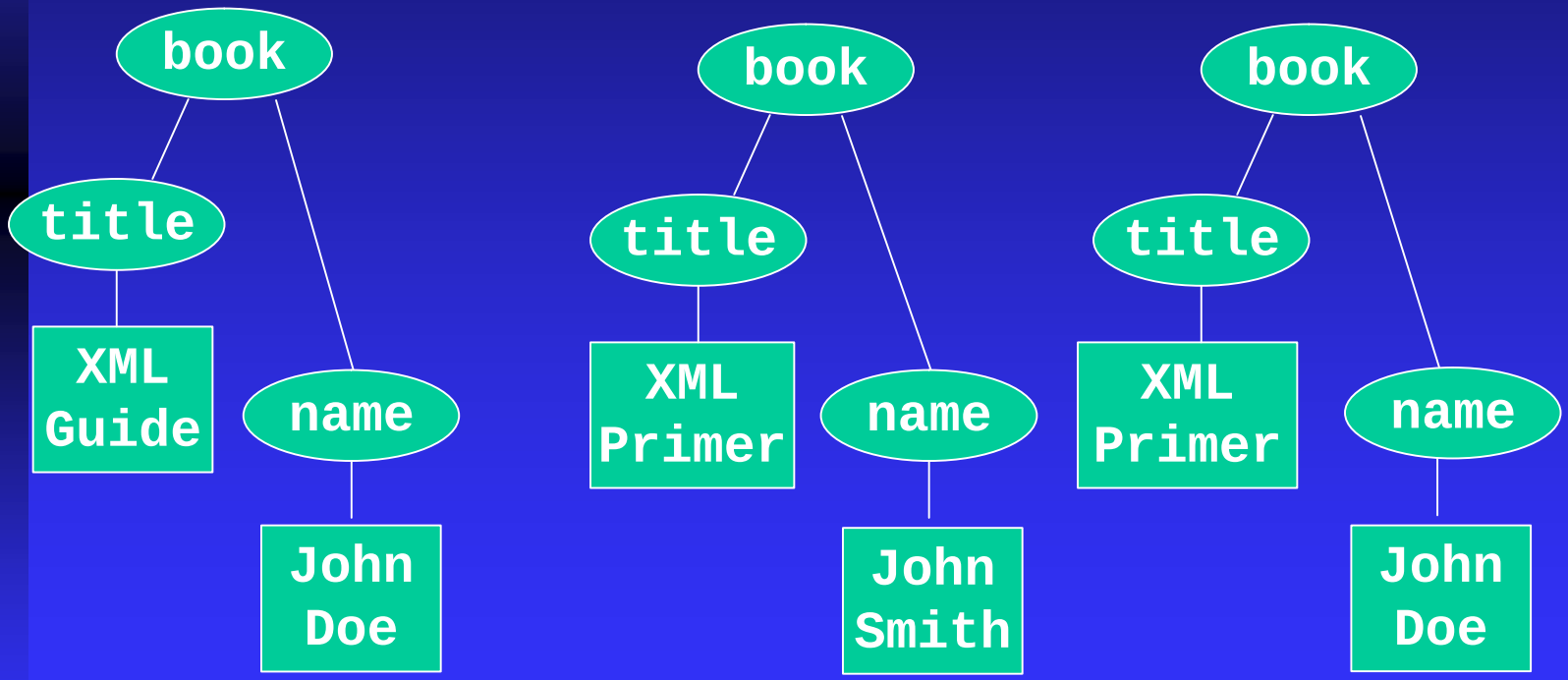
- A set of trees may not be homogeneous in structure. How to identify nodes of interest?
- Define a structural pattern tree, identifying all nodes of interest. Match this pattern against all input trees.



Sample Data


bib
book
book
book
title
author
title
author
title
author
author
Web Guide
name
XML Guide
name
XML Primer
name
name
John Doe
John Doe
John Smith
John Doe

Witness Trees



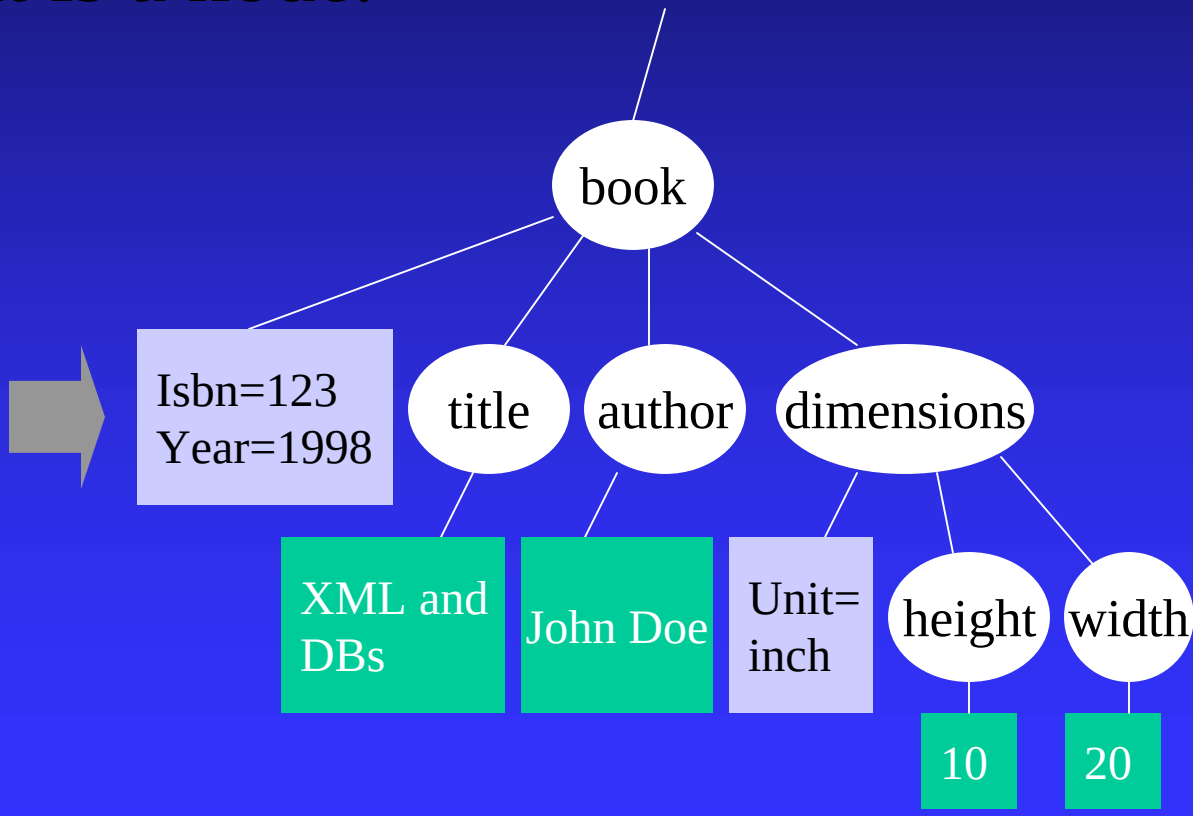
Outline

- Introduction
- Logical basis – Tree algebra
- TIMBER system overview
  - Physical Design
  - System Architecture
- Selected specifics

XML Storage

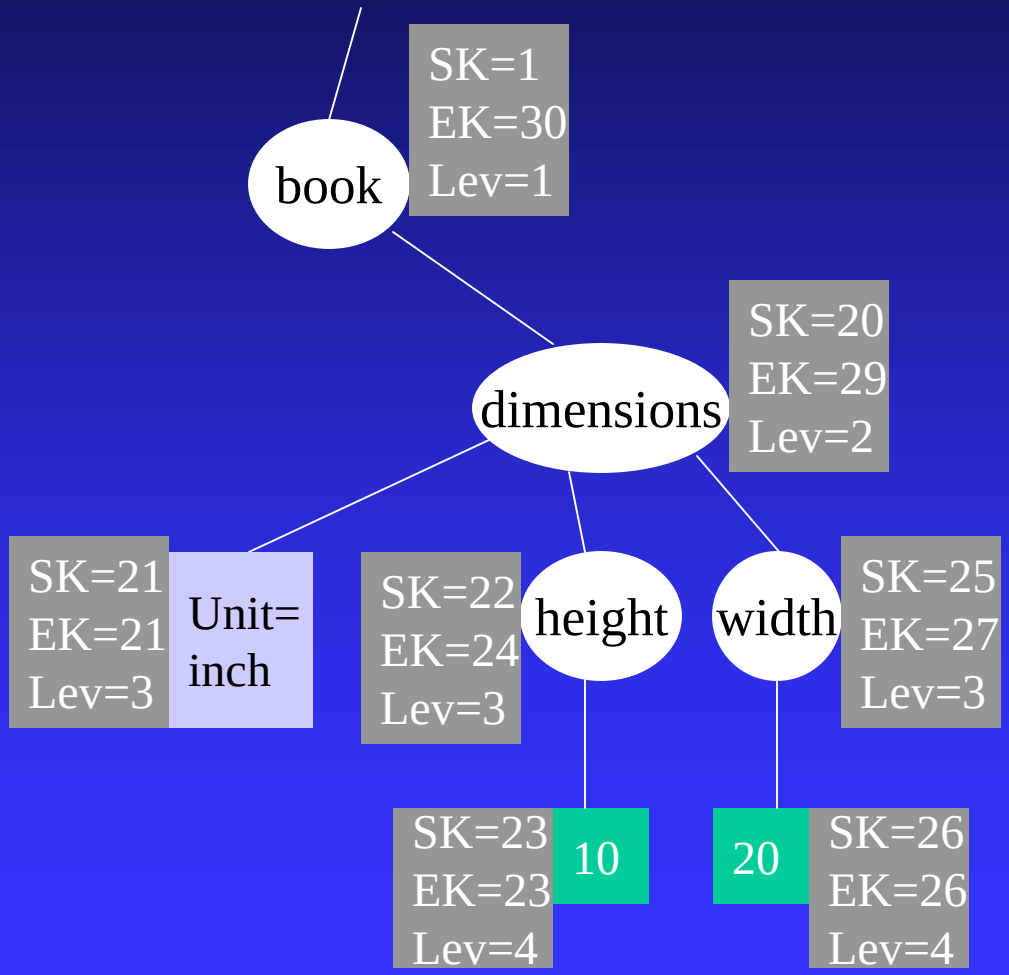
- Storage unit is a node.

```
:
:
:
<book isbn="123" year="1998">
  <title>XML and DBs</title>
  <author>John Doe</author>
  <dimensions unit="inch">
        <height>10</height>
        <width>20</width>
  </dimensions>
</book>
:
:
```

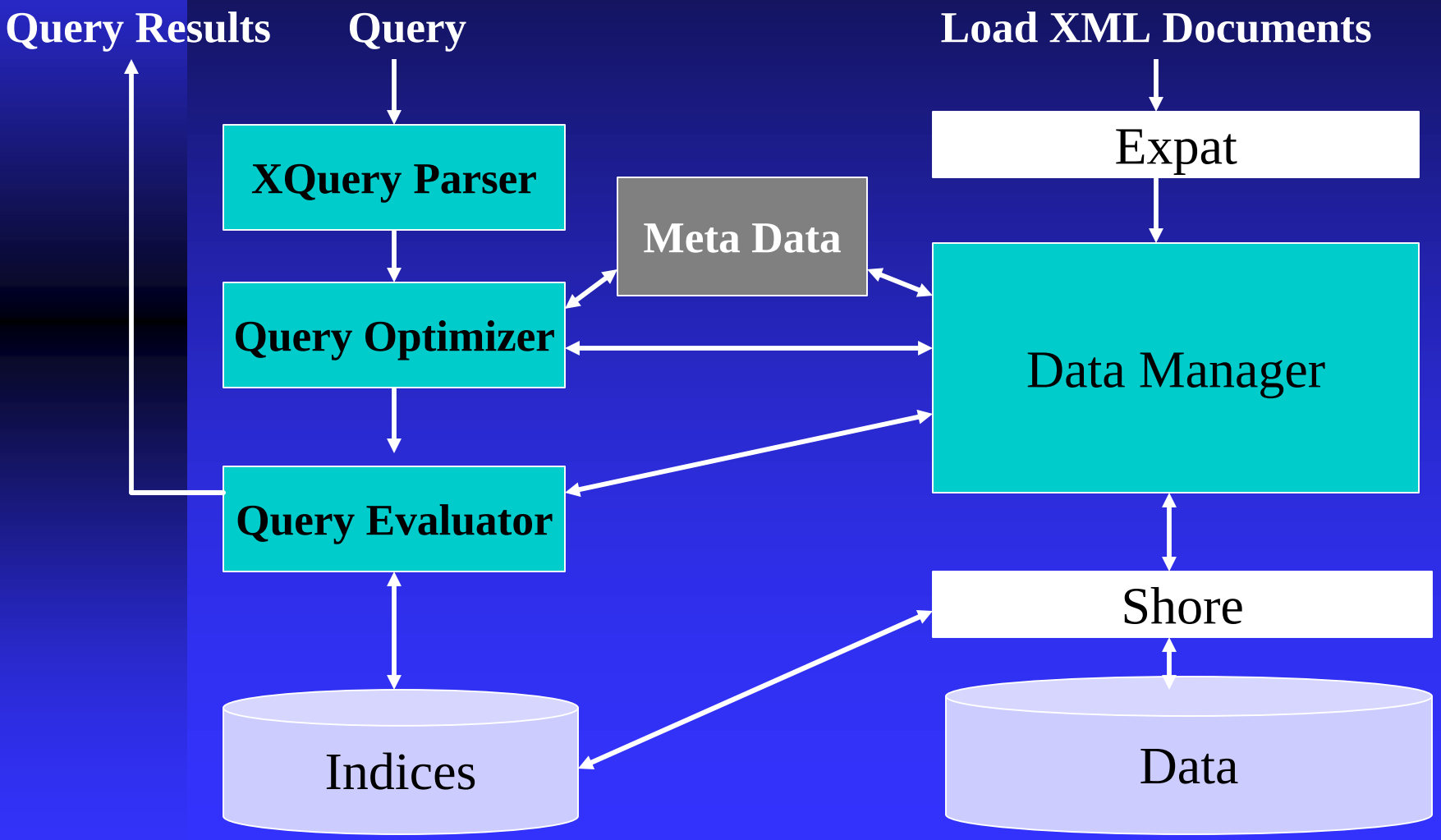


Node Key

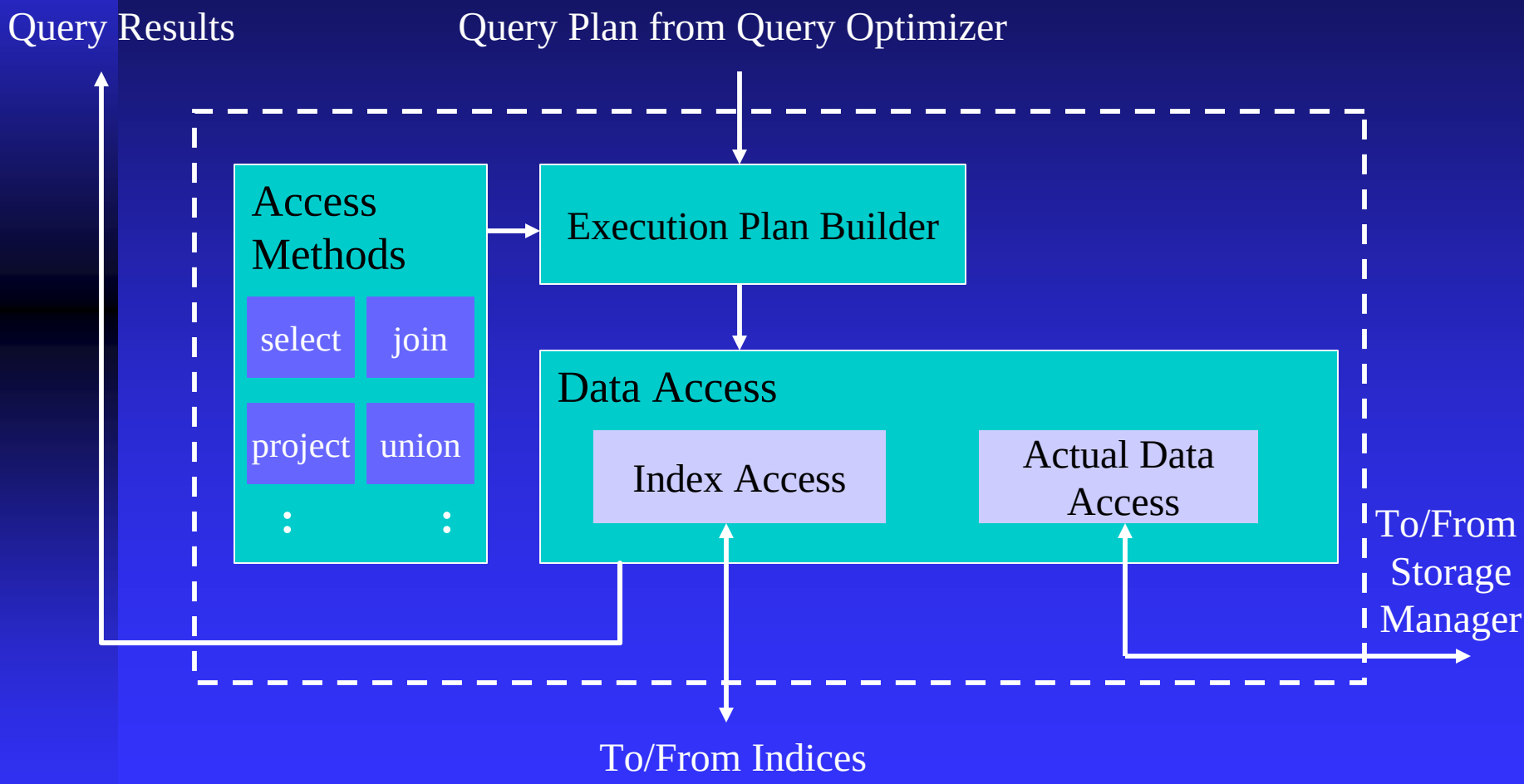
- Each node has
  - Start Key.
  - End Key.
  - Level.



TIMBER Structure



The Query Evaluator



Process Identifiers

- Indices return node identifiers.
- Continue computations with identifiers as long as feasible
  - actual data much larger than identifier
  - postpone access to the actual data
  - carefully stage full materialization of result data

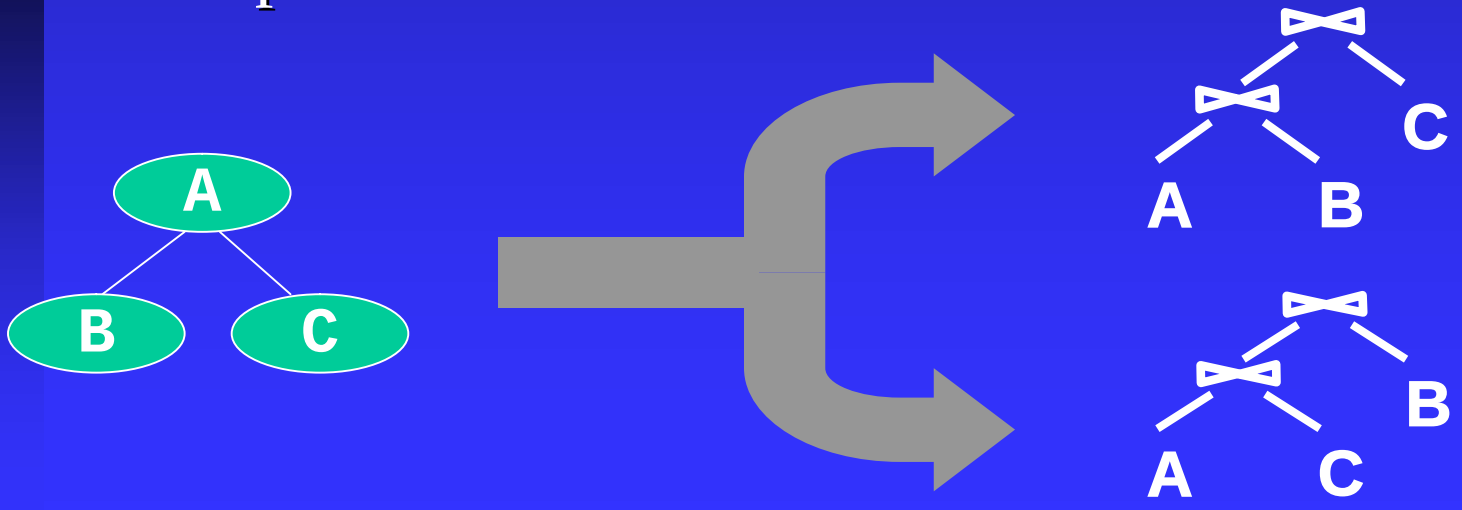
Outline

- Introduction
- Logical basis – Tree algebra
- TIMBER system overview
- Selected specifics
  - One key access method – Structural join
  - Query optimization – Cost estimation

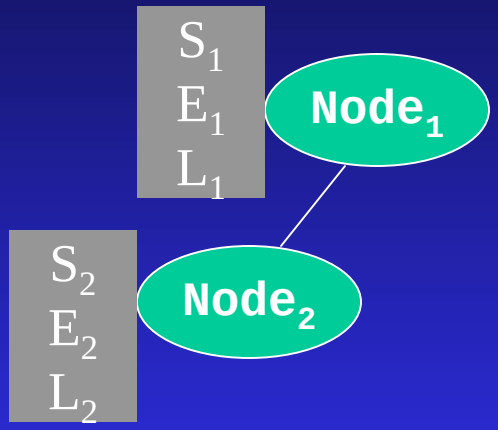
Decomposing Pattern Trees

- Steps in obtaining a pattern tree match
  1. Match individual nodes → indices.
  2. Establish relationships between nodes matching indices → structural joins.

Example.



Ancestor-Descendant Join



S_i = Start Key of element i
E_i = End Key of element i
L_i = Level of element i

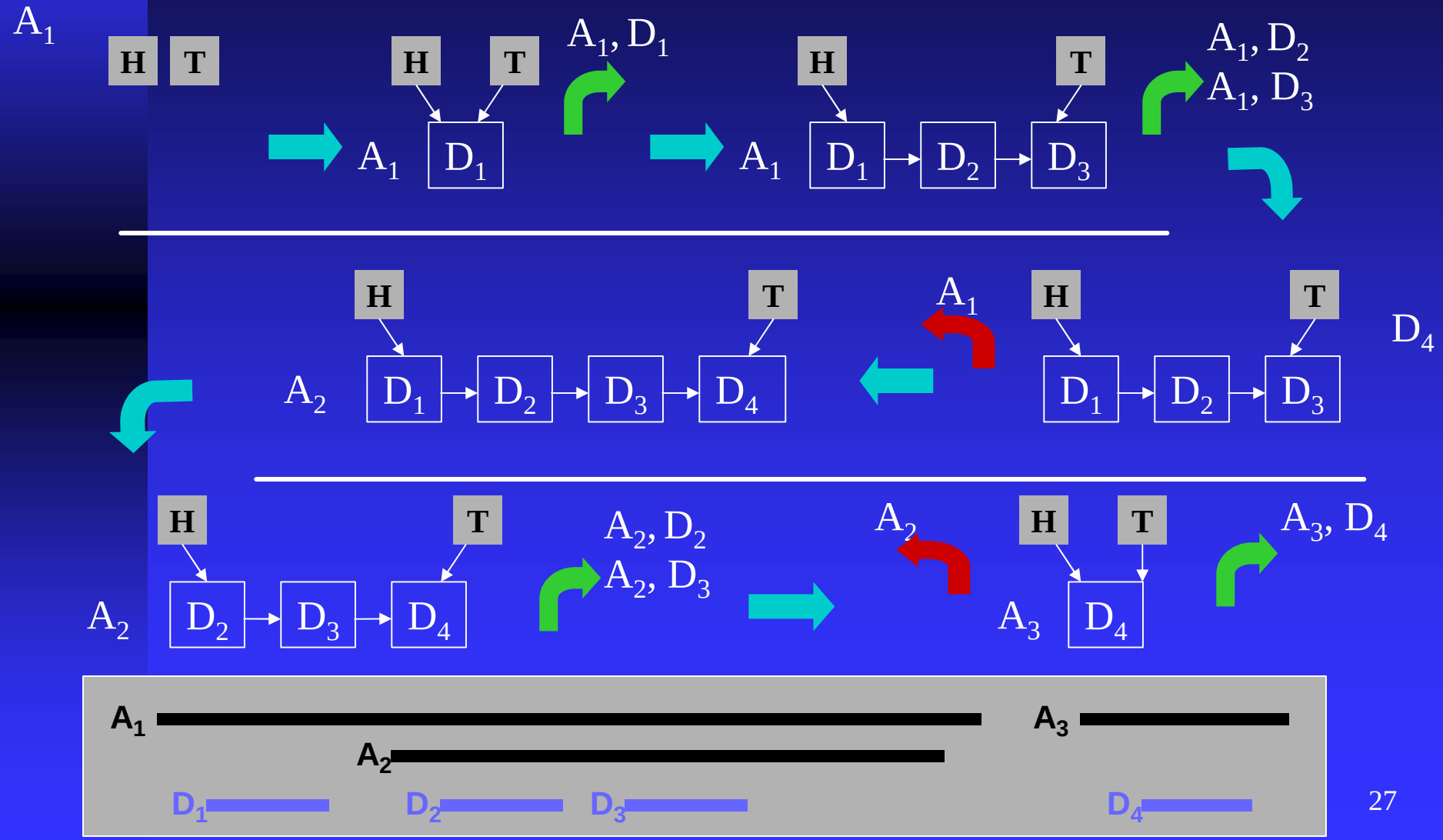
- Join condition: $S_1<S_2$ & $E_2<E_1$
- Special case: Parent-Child.
  - Add the level condition: $L_2=L_1+1$

Structural Join Algorithms

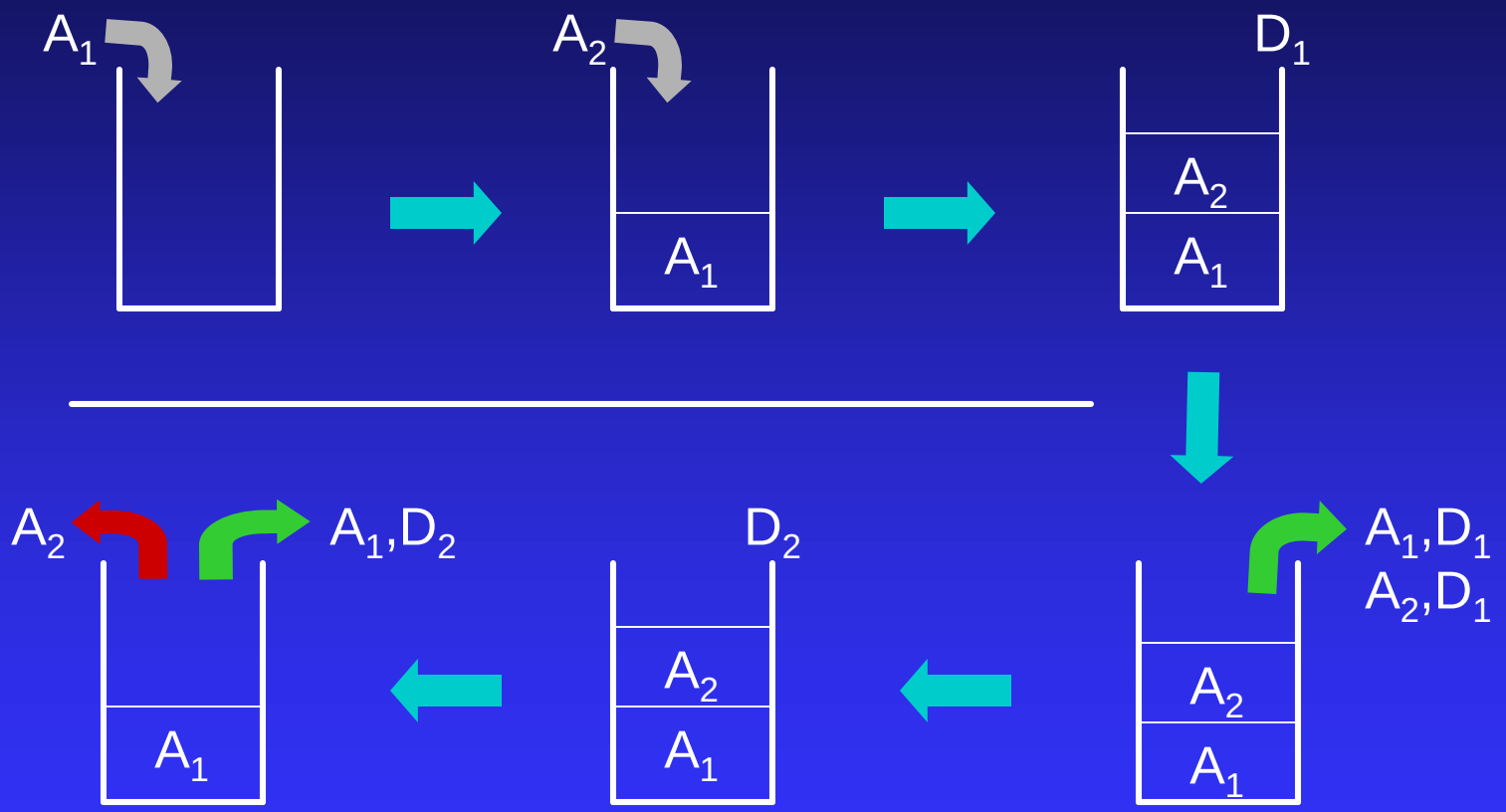
- Two classes of algorithms
  - Tree-merge algorithms.
  - Stack-based algorithms.

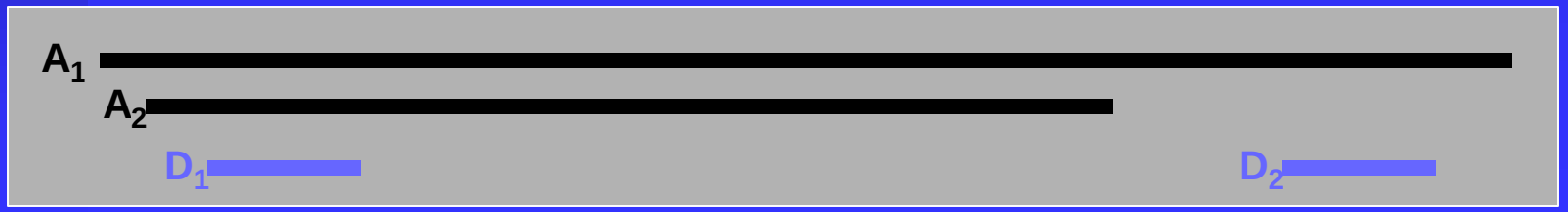
- Ordering of output is very important.

Tree-merge Algorithms

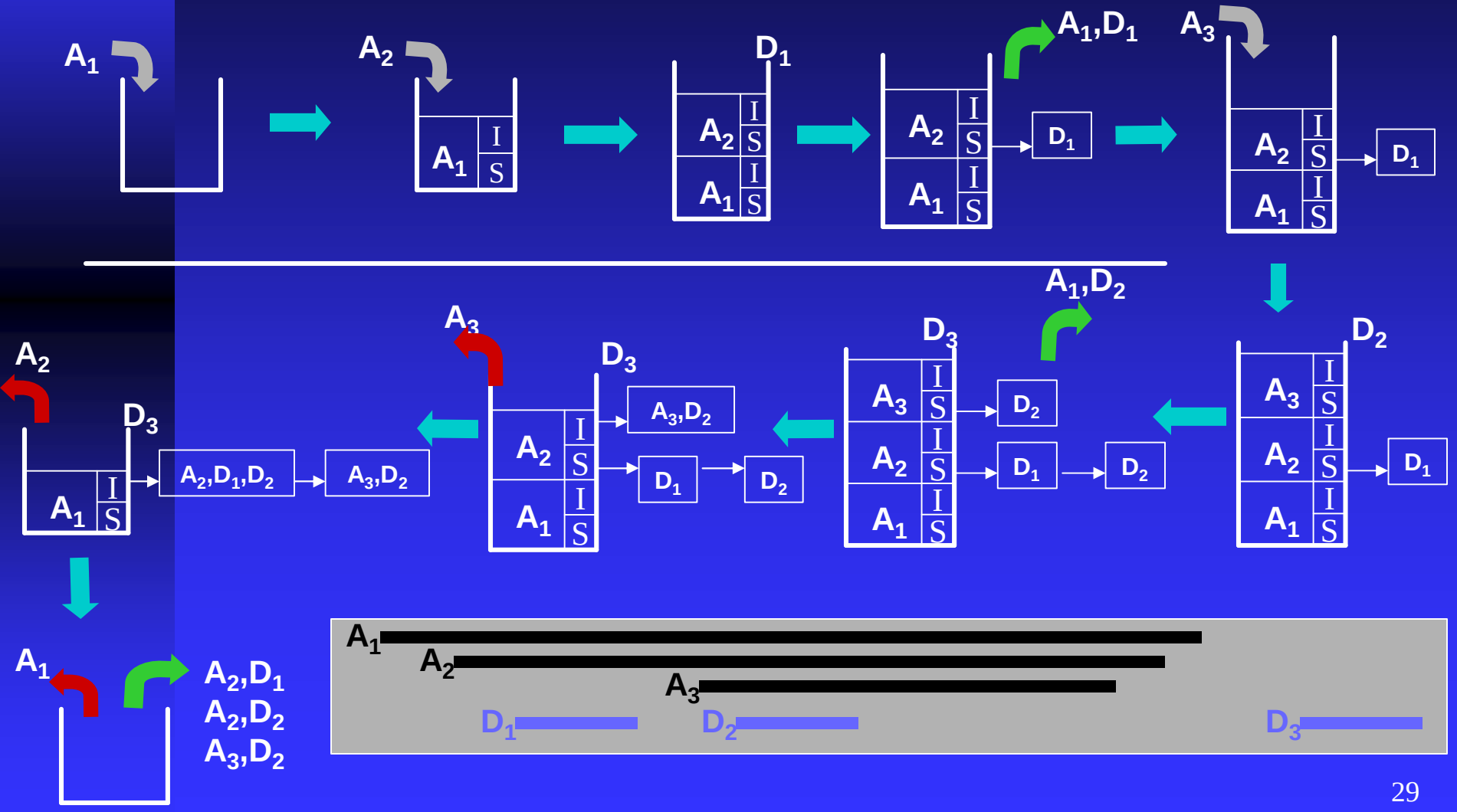


Stack-based Descendant

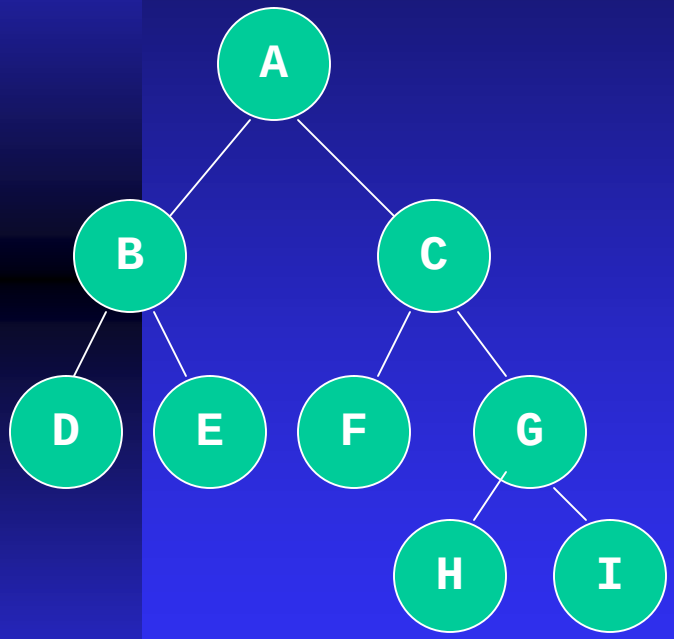




Stack-based Ancestor



An Example Join Plan

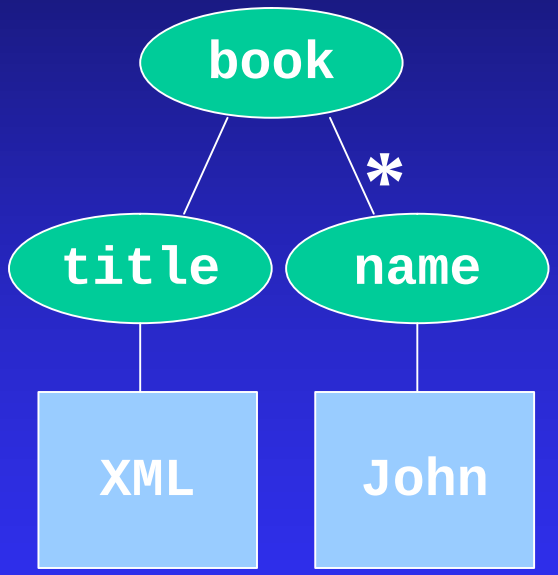


B join E	Sort by B
BE join D	Sort by B
BDE join A	Sort by A
ABDE join C	Sort by C
ABCDE join F	Sort by C
ABCDEF join G	Sort by G
ABCDEFG join I	Sort by G
ABCDEFGI join H	Sort by ??

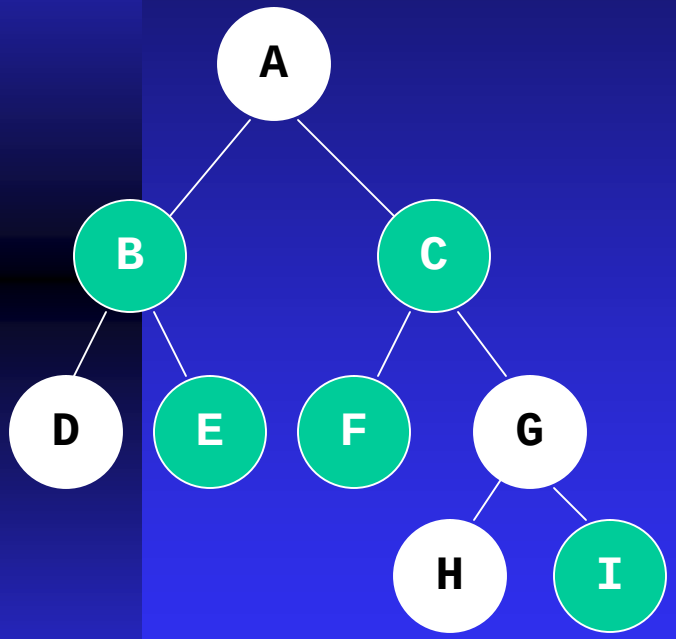
Variants of Joins are Valuable

We may only wish to return the book for this pattern tree match, rather than the entire witness tree.

We can use new variants of the join access methods to exploit such a “projection” applied to the results of a pattern match.



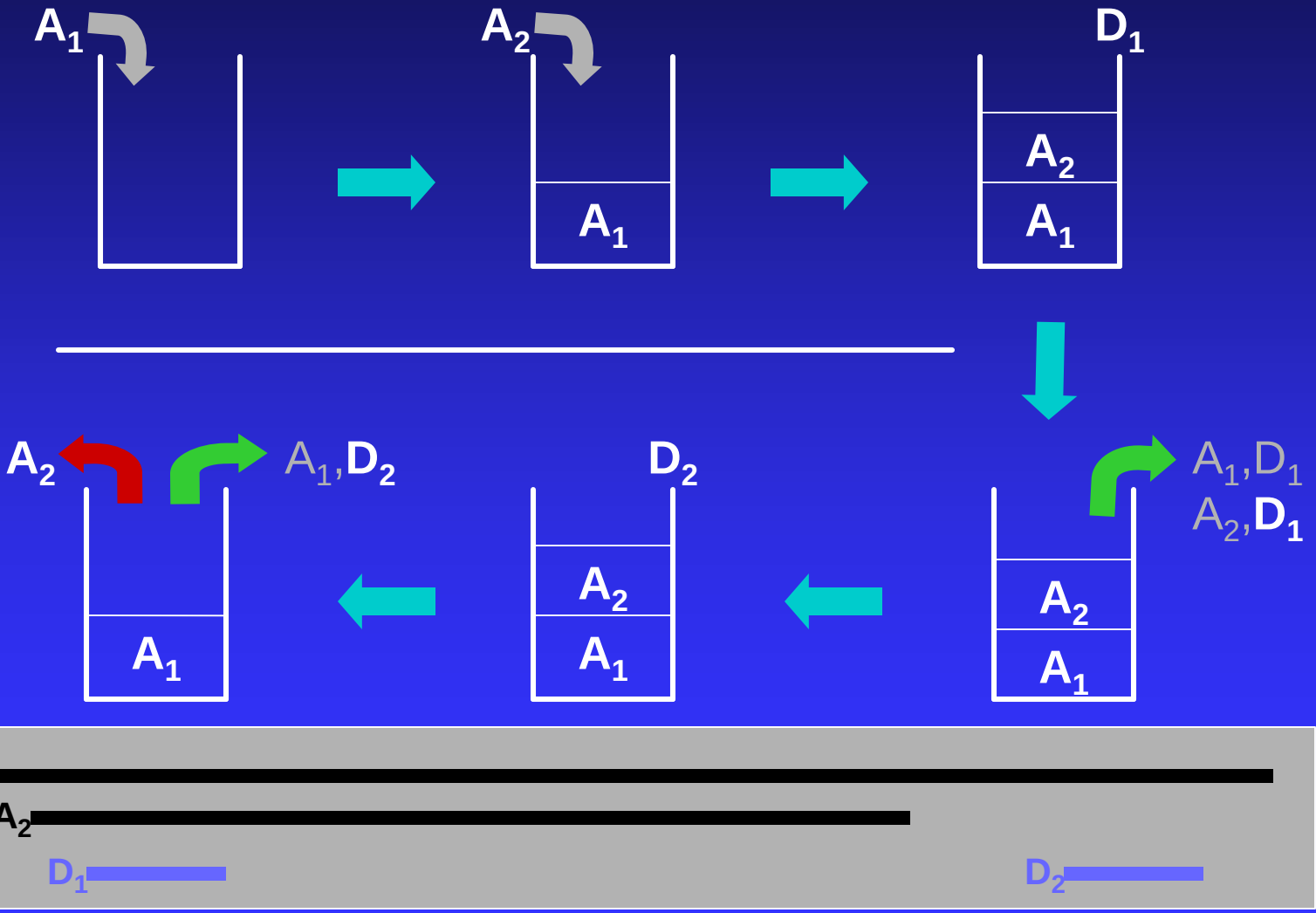
Example of Projection Push-In



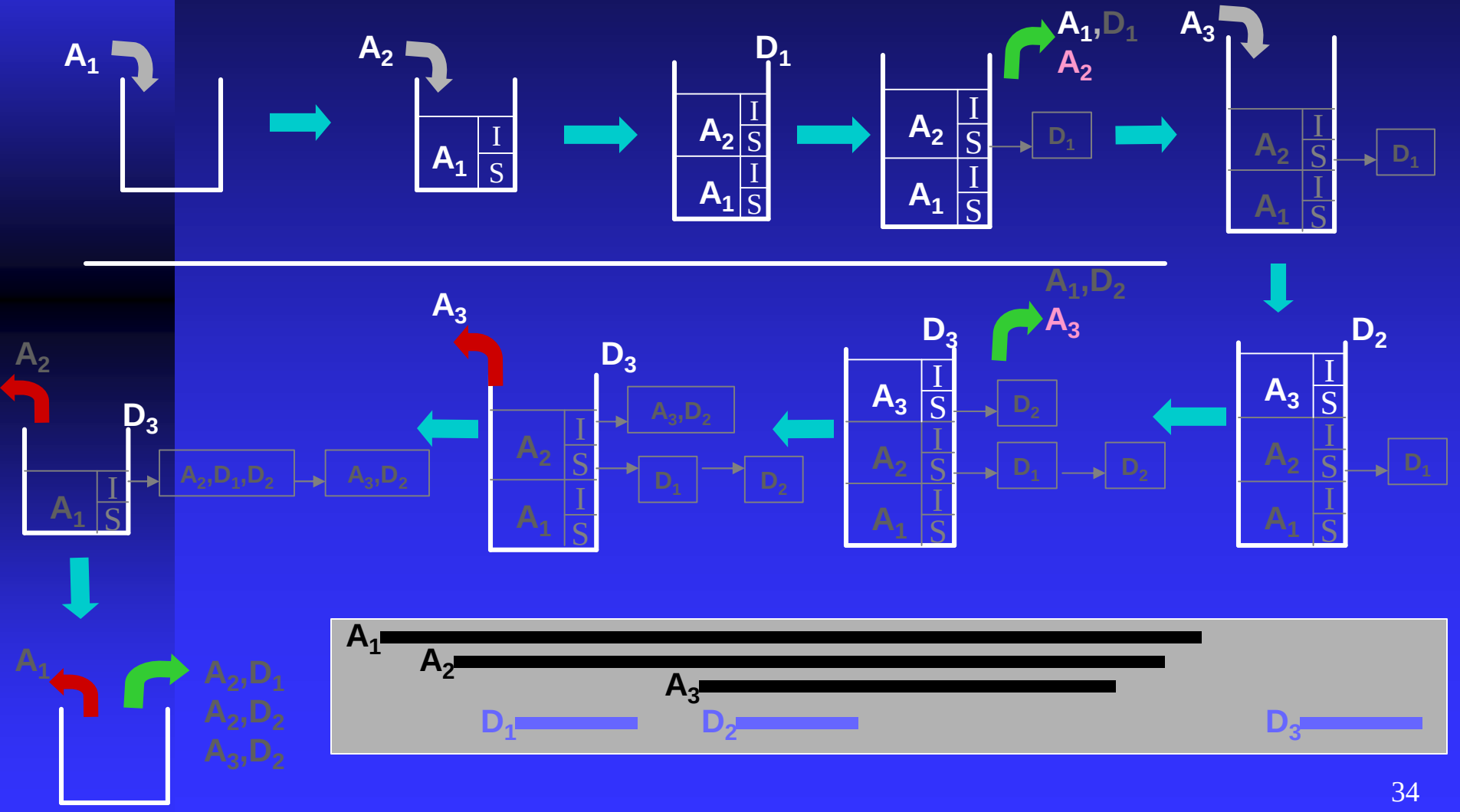
- PL = {D, A, G, H}
- DAB (Distinguished Ancestor Bag) = {B, B, C, C}

B join E → B	DAB={B,C,C}
B join D → BD	DAB={C,C}
BD join A→AD	DAB={C,C}
AD join C → ADC	DAB={C}
ADC join F → ADC	DAB={}
ADC join G → ADG	DAB={}
ADG join I → ADG	DAB={}
ADG join H → ADGH	DAB={}

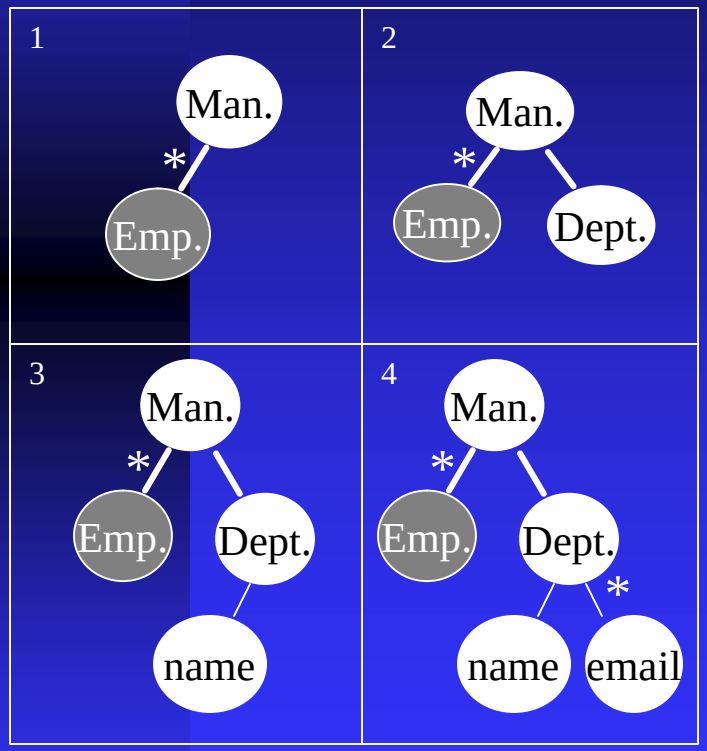
Partial Join Stack-based Descendant

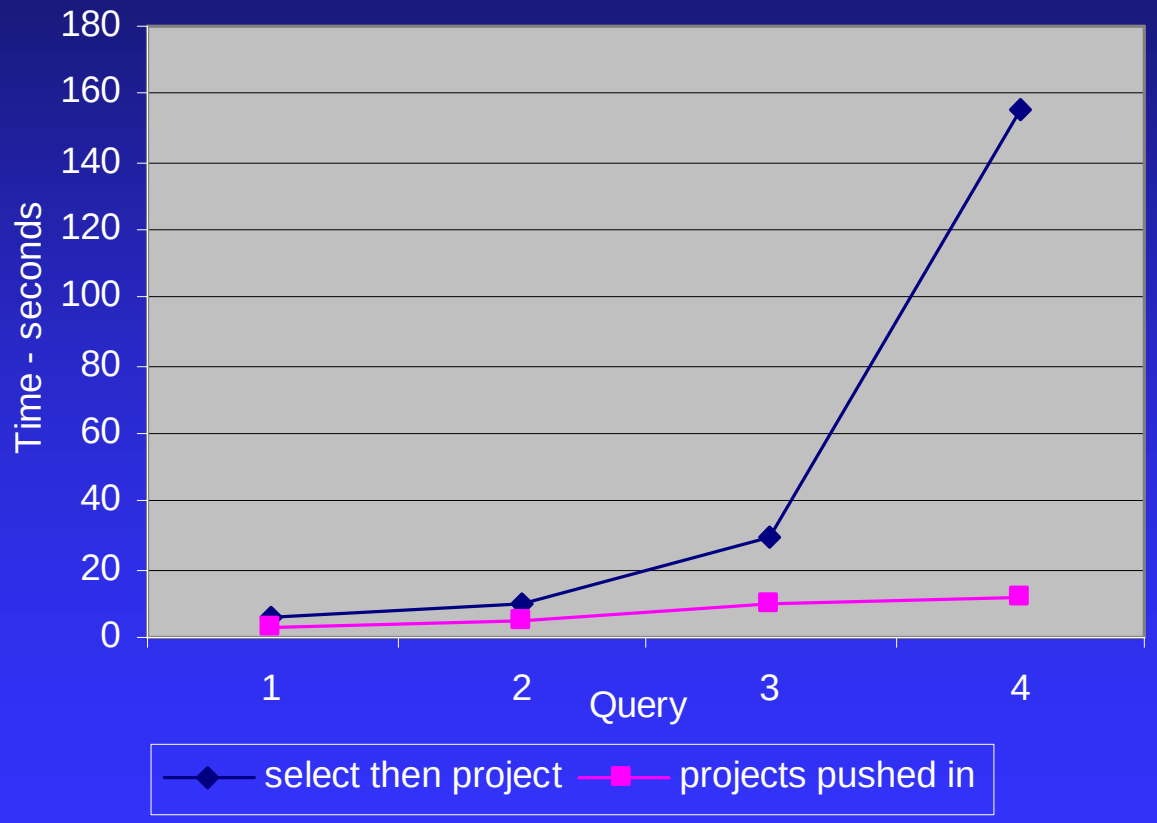


Partial Join Stack-based Ancestor

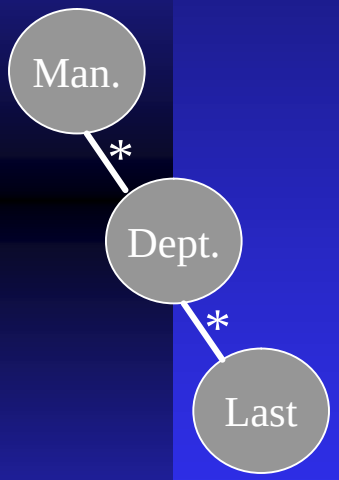


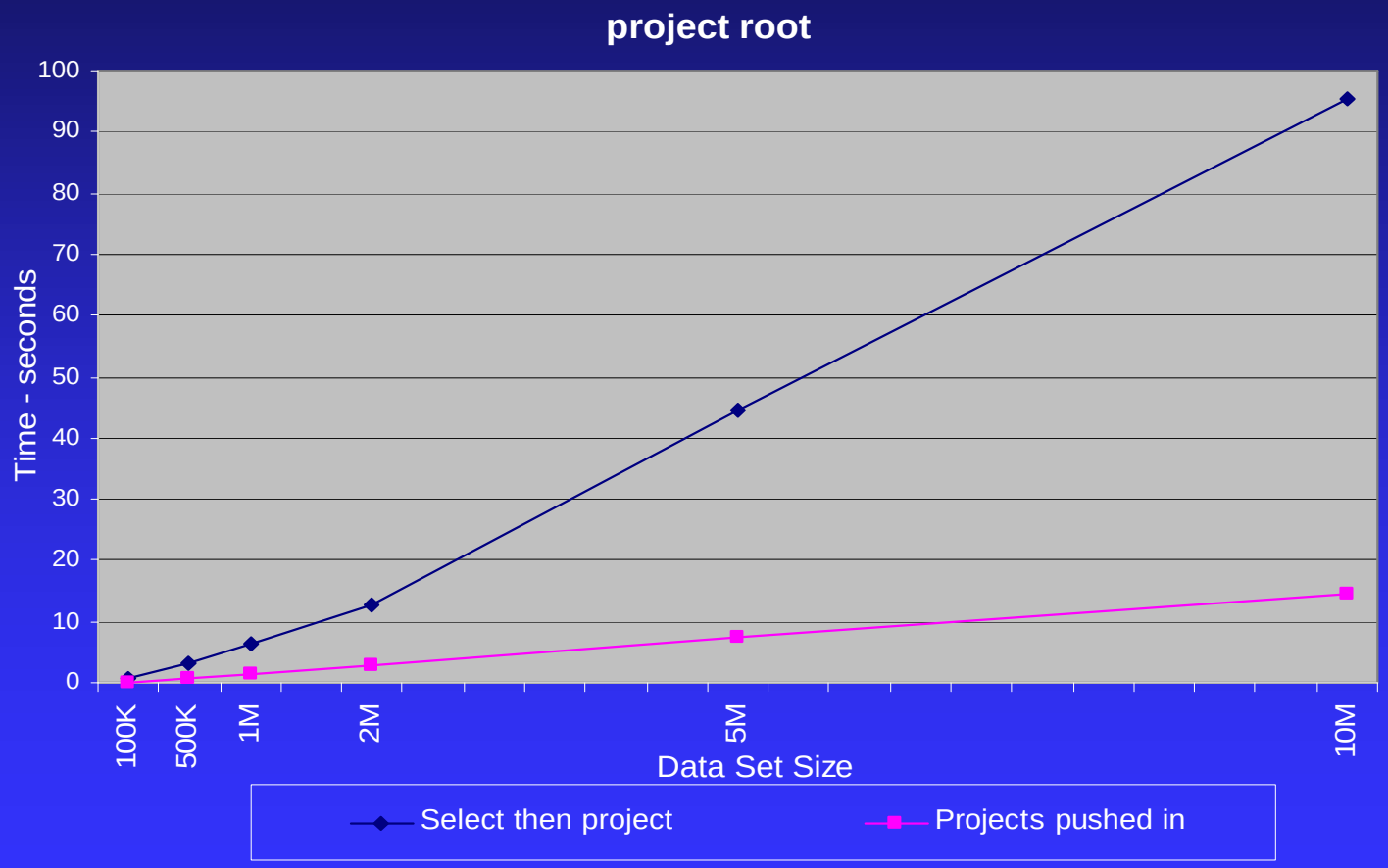
Increasing Query Complexity





Increasing Size of Data Set



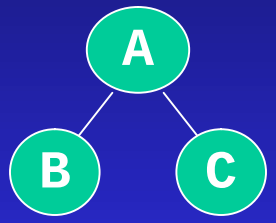


Outline

- Introduction
- Logical basis – Tree algebra
- TIMBER overall system architecture
- Selected specifics
  - One key access method – Structural join
  - Query optimization – Cost estimation

Structural Binary Join Order

exp.

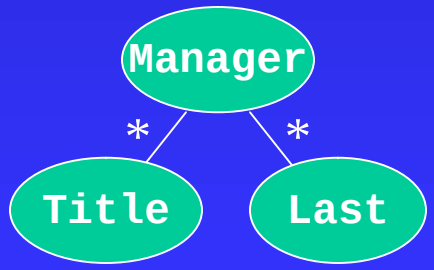


A join B→AB join C→B

A join C→AC join B→B ←better

Experimentally:

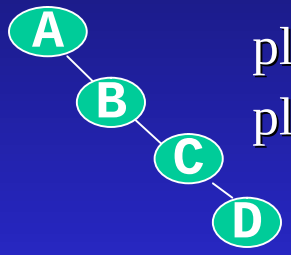
Depth is 13 levels and size is 5M nodes.



Plan	Time
Man. Title first	20.363
Man. Last first	10.305

Rules of Thumb

"Never start from the middle."

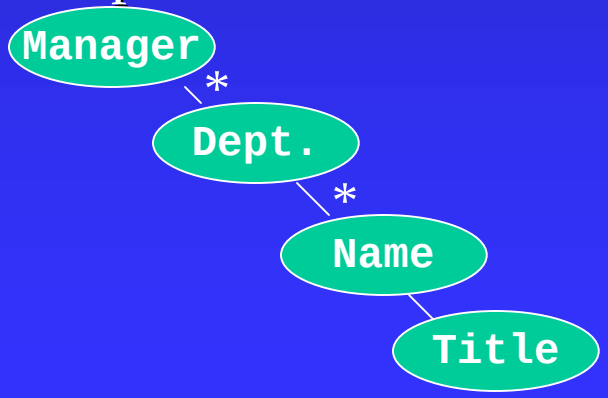


plan 1 = B join C→BC join D→BCD join A→

plan 2 = C join D→CD join B→BCD join A→ ←better

Experimentally:

Depth is 13 levels and size is 5M nodes.



Plan	Time
Plan 1	80.716
Plan 2	18.593

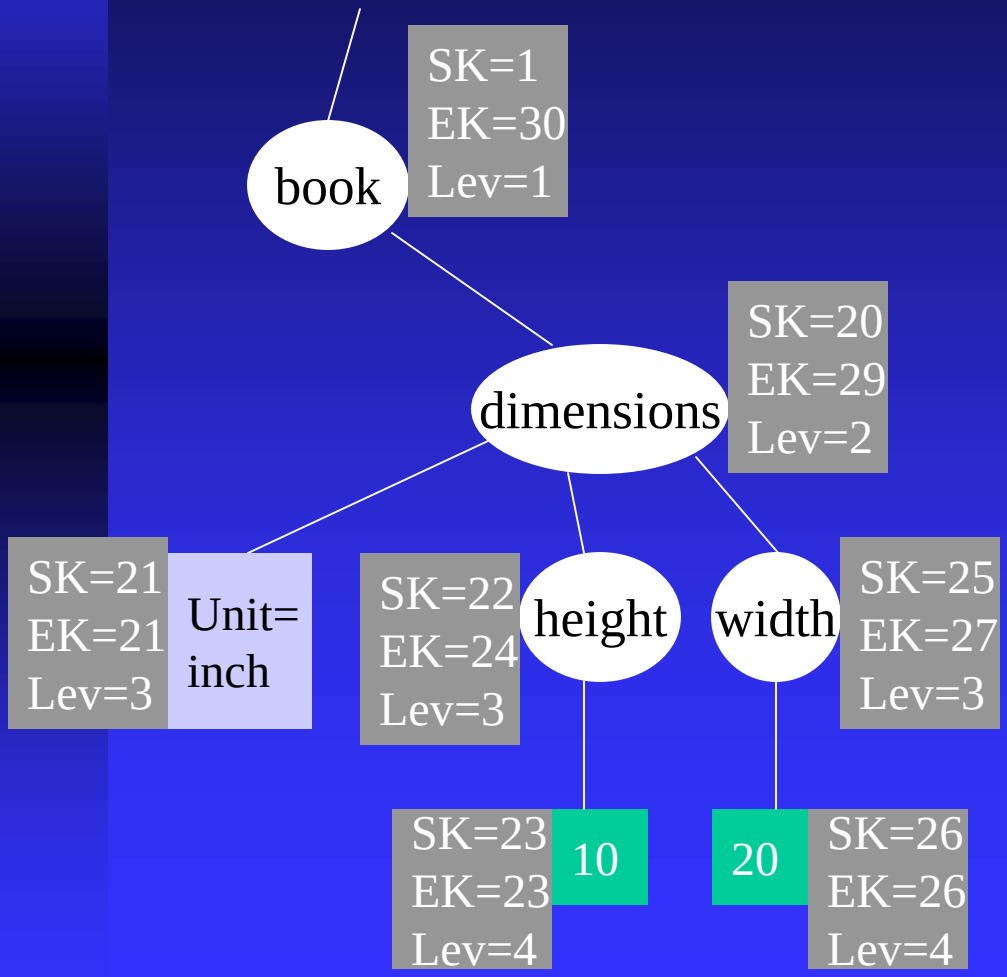
Importance of Size Estimation

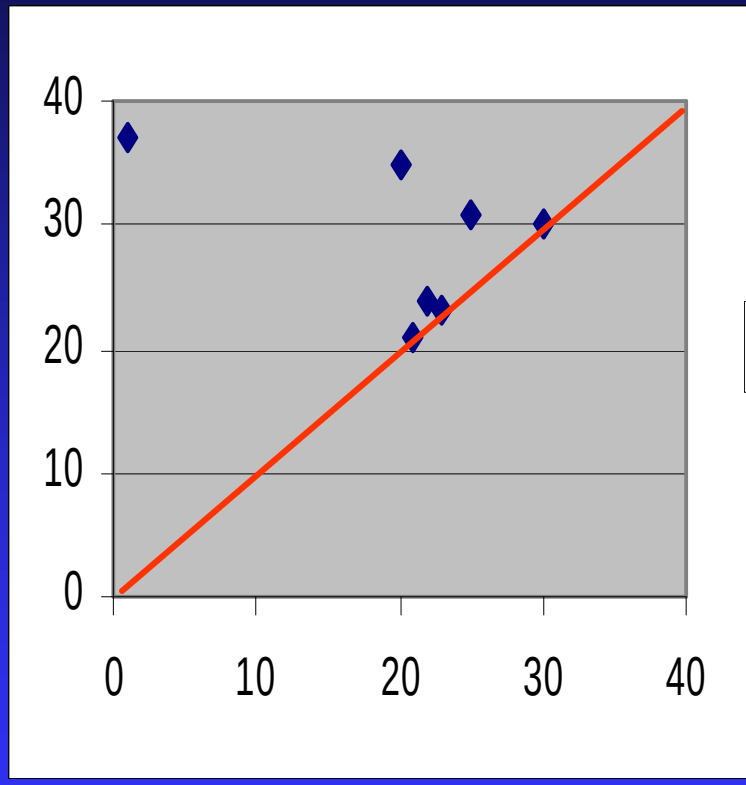
- Query optimizer needs reliable estimate of (intermediate) result sizes to be able to choose between query plans.
- How can we estimate the size of the answer set without computing the answer?
- Use pre-computed summary data statistics.

Difficulty of Size Estimation

- Straightforward to keep statistics regarding individual XML elements.
- Can attempt to estimate pattern match selectivity by assuming independence between nodes in the pattern.
- But strong correlations typically exist between nodes in the pattern.
- How to record these cheaply?

Two-Dimensional Key Space





Position Histograms

- Exploit locality to store aggregate population information regarding points in two-dimensional key space.
- Divide key space into g X g grid. Store occupancy information for each cell.
- Only O(g) storage required!!!
- Relative placements of grid cells provide the needed estimates.

Coverage Histograms

- Many embellishments possible.
- Can even store key-space distribution of descendants of points in a particular grid cell.
- Though this may point towards $O(g^4)$ storage, can show it only requires $O(g)$.

Result Size Estimation for Three Node Twig Queries on the DBLP Data Set

Node A	Node B	Node C	AB Est	AB Real	Naïve ABC Est	ABC Est	ABC Real
article	author	cite	14,627	14,644	2,357,210,770	11,619	14,259
article	title	author	7,323	7,366	8,042,378,602	15,341	14,644
book	title	year	610	408	5,085,086,419	471	408
book	url	cdrom	493	33	8,469,868,800	3	3
proc	title	url	19	32	0,013,818,800	11	28

Outline

- Introduction
- Logical basis – Tree algebra
- TIMBER overall system architecture
- Selected specifics
- Done!!!

Status

- Basic architecture of Timber is in place.
- Initial software release planned for spring.

But …

- We are nowhere close to done.

Summary

- XML is the data representation format of the future.
- Managing XML data effectively raises a number of challenges.
- The TIMBER project has addressed several of these, in a principled way.
- Leads the way towards a new generation of Database Management Systems.

Timber is good for building the next generation data warehouse