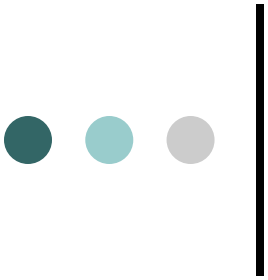
C=(FS)^2 : Cubing by Composition of Faceted Search

Ronny Lempel
Dafna Sheinwald

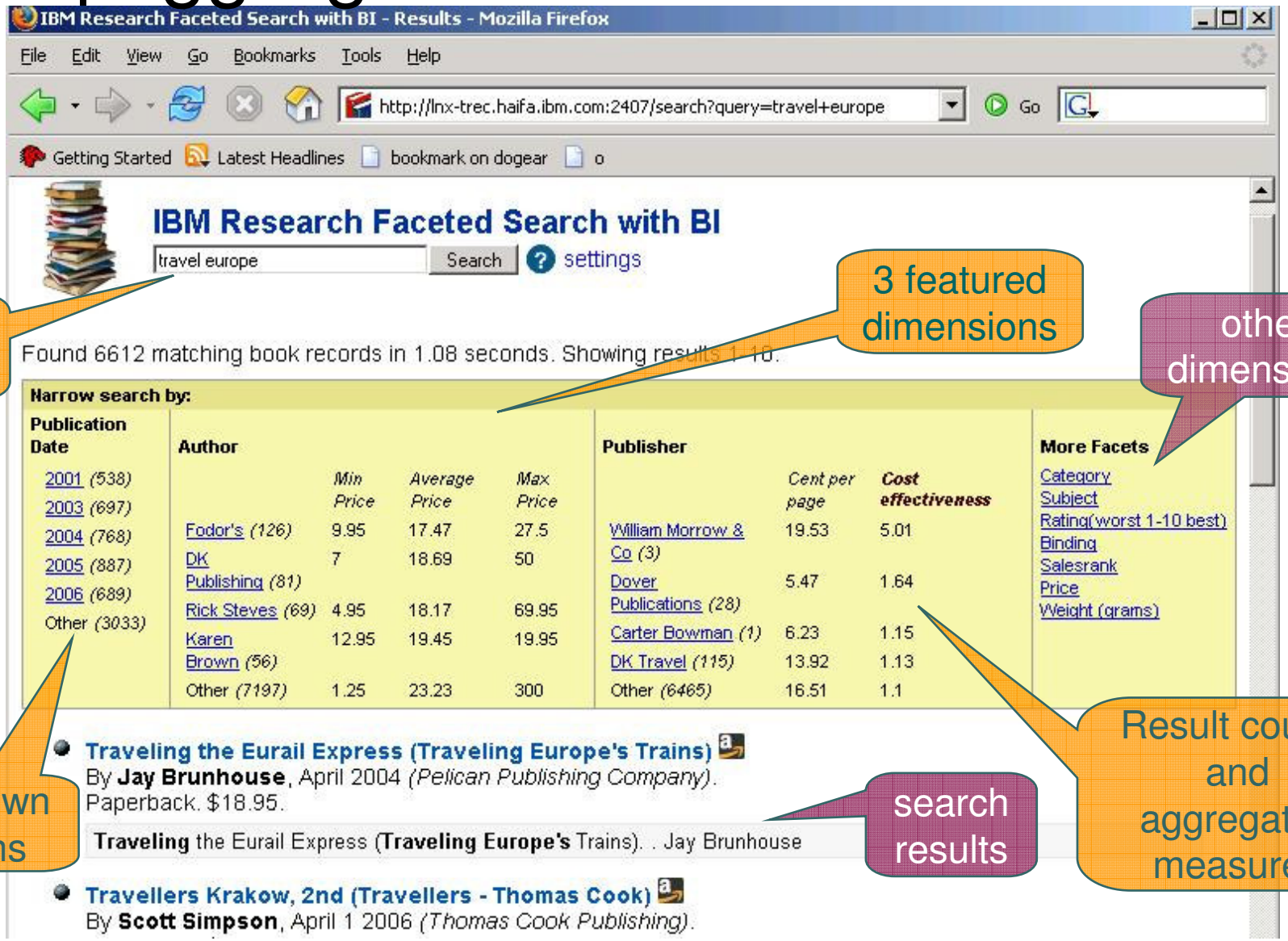
IBM Haifa Research Lab



Introduction to Multifaceted Search and to On-Line Analytical Processing (OLAP)

Intro

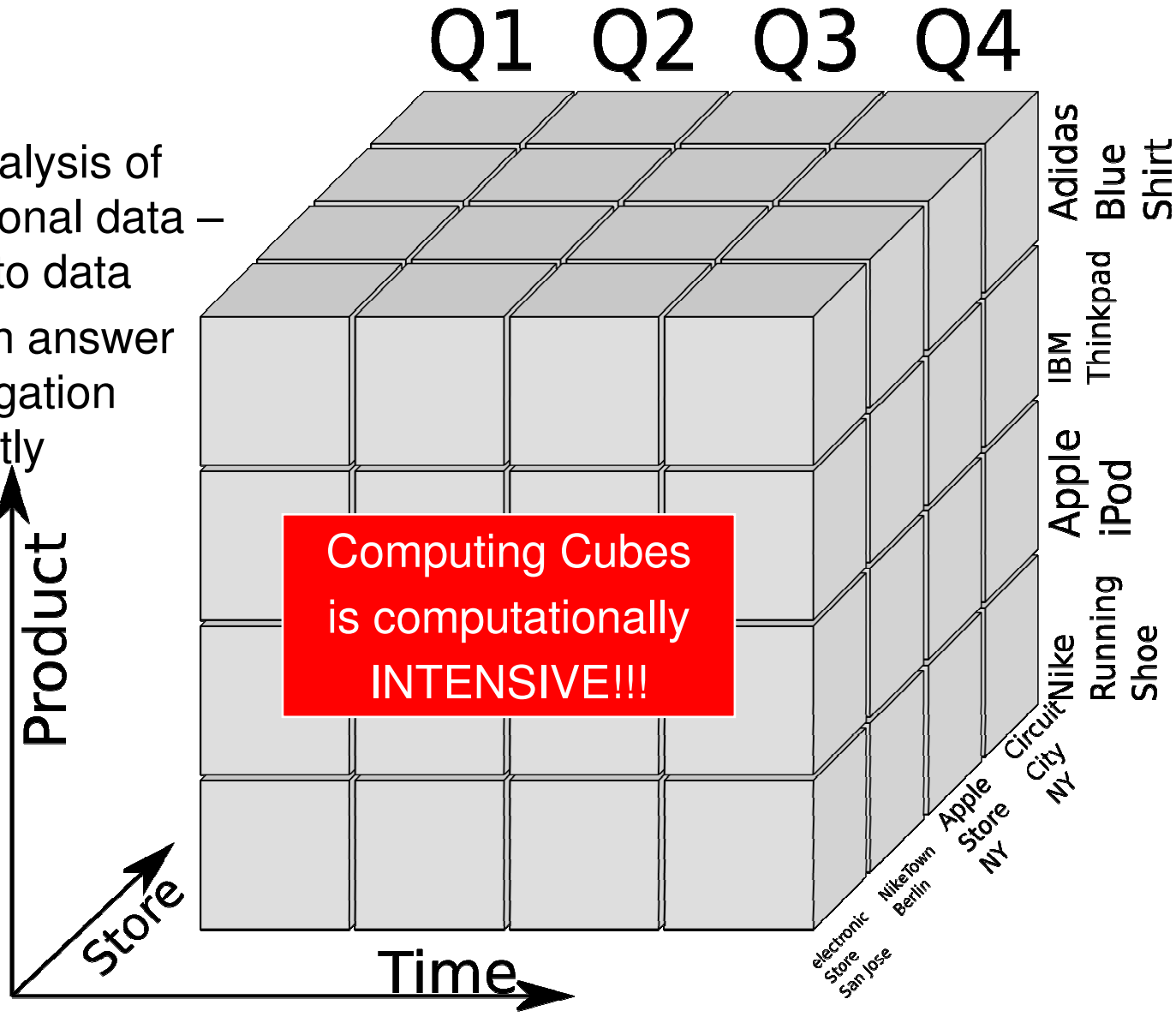
Multifaceted Search With Aggregated Measures



Intro

OLAP Cubes

- Primary tool for multidimensional analysis of structured and relational data – qualitative insight into data
- Once computed, can answer selection and aggregation queries very efficiently
  - Drill down
  - Roll up
  - Measures



Intro

Goal: General OLAP Cubing over Search Results

- Observation: faceted search with BI expressions is practically equivalent to having several “one-dimensional cubes” with measures
  - Equivalently, cubes can be seen as aggregations over the cross-product (Cartesian product) of several facets
- One cannot compute cubes accounting for free-text queries, since the space of possible queries is practically infinite
- Two natural approaches for combining free-text search and cubing:
  - Pipe the search engine’s results into a cubing engine
  - Extend a faceted search engine to aggregate by cross-products of facets
- We will actually do something different…

Intro

Interaction Paradigms

- Faceted search: ad-hoc discovery sessions of a multidimensional corpus using a mixture of keyword search and zoom-in/zoom-out browse operations
  - Typically few browsing operations per keyword query
  - Used by everyday users, in sessions lasting minutes
- OLAP Cubes: deep analysis of a fixed multidimensional corpus
  - A dataset is fixed, the cube is defined and built, and many cubing queries are executed
  - Used by professional analysts in sessions lasting days
- Mission: support cubing operations over subsets of a corpus defined by search-engine result sets

IDEA

Cubing Queries: Usage Scenario and Flow

Usage scenario: on the cube that was built over the documents and dimensions defined by some initial query, many subsequent slice-and-dice operations will be performed.

Submits selection query (text predicates + category and range constraints) that defines the set of result documents to be considered by the cube

1

Also defines the requested dimensions over which to build the cube

Faceted Search Index

User now submits numerous slice & dice queries to this "cube"

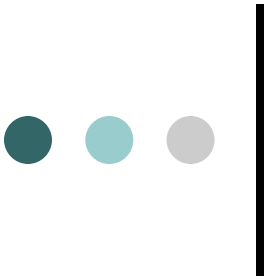
3

Cube

2

Search engine returns an object that supports cubing queries

What is the implementation behind this "cube"? We will show it to be just another Faceted Search index, built on altered and "massaged" input!

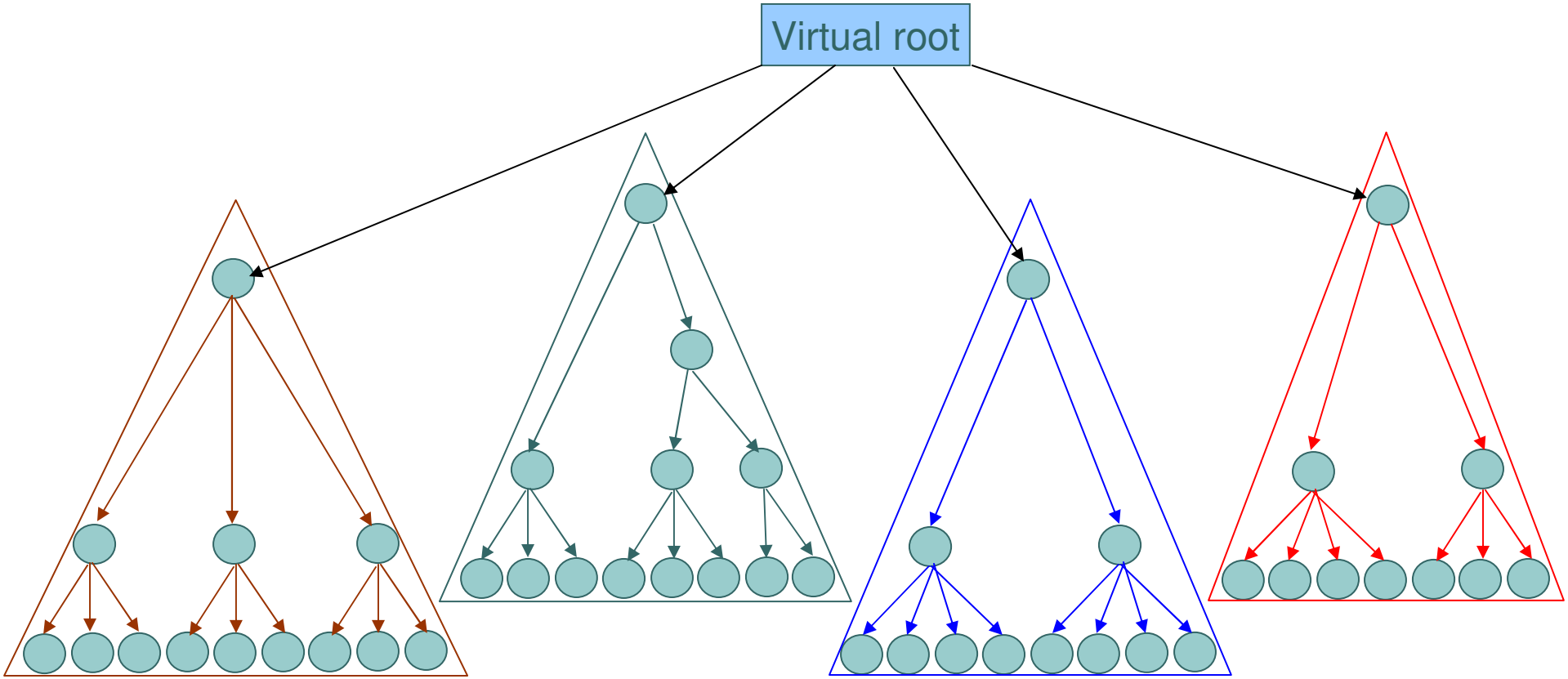


Multifaceted Search Implementation: Data Model, Indexing and Runtime

Data Model

Data Model (1)

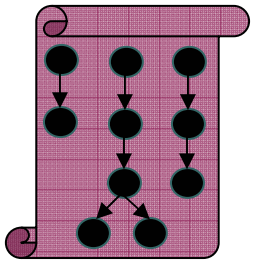
- There's some background (latent) multidimensional hierarchy – four dimensions in this example

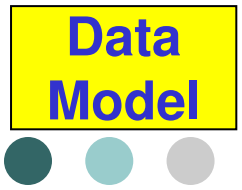


Data Model

Data Model (2)

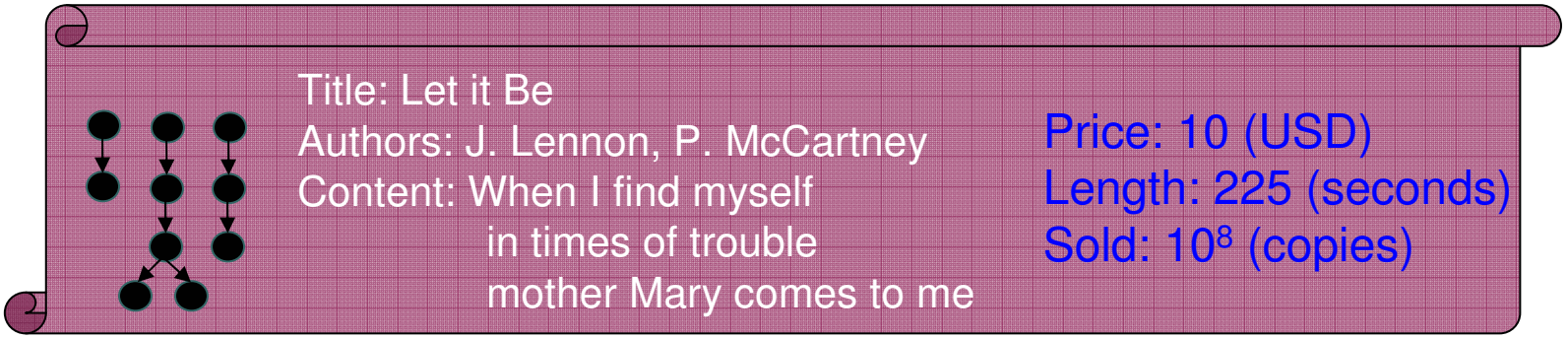
- Each document is associated with a set of paths
  - Perhaps several paths in some dimensions
  - Perhaps no path in some dimensions
  - Path may end not at a leaf





Data Model (3)

- Actually, each document can be thought of logically as composed of three parts:
  1. The collection of paths of categories/facets it is associated with
  2. Some free text, in one or more fields
  3. Several numeric attributes/fields that are associated with it



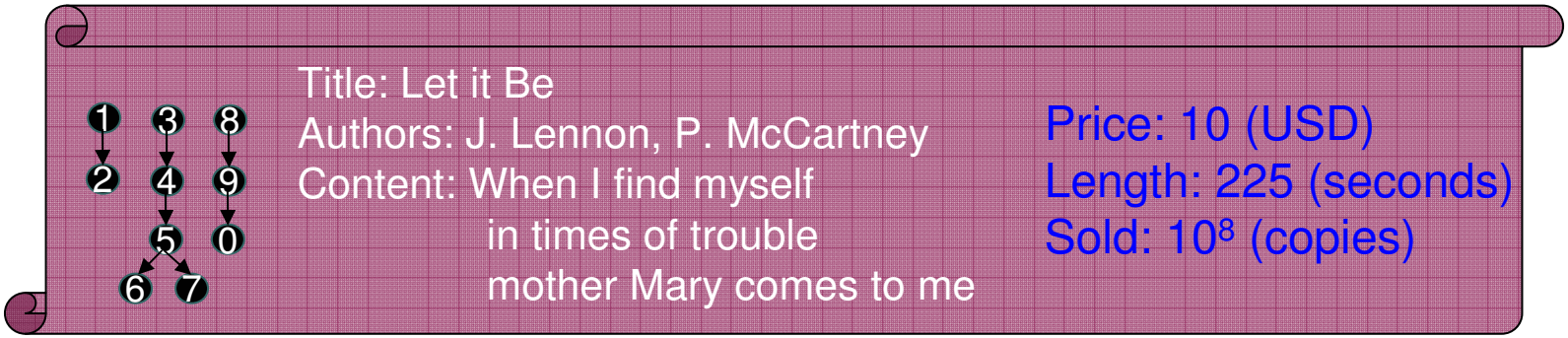
The Taxonomy Index

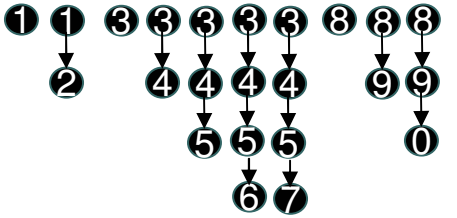
- Documents are ingested by the index one by one
- The taxonomy index component unions the set of paths associated with each document to form the overall taxonomy implied by the documents
- The taxonomy index provides the following services:
  - Maps human readable labels of each taxonomy node to some internal ID
  - Maps nodes to their ancestors
  - Maps nodes to their children

Faceted Indexing

The Inverted Index (1)

- Keeps postings lists per each text token
- Keeps postings lists per each path prefix, listing all documents that are associated with the path prefix
- Keeps numeric postings lists to support range-constraint queries



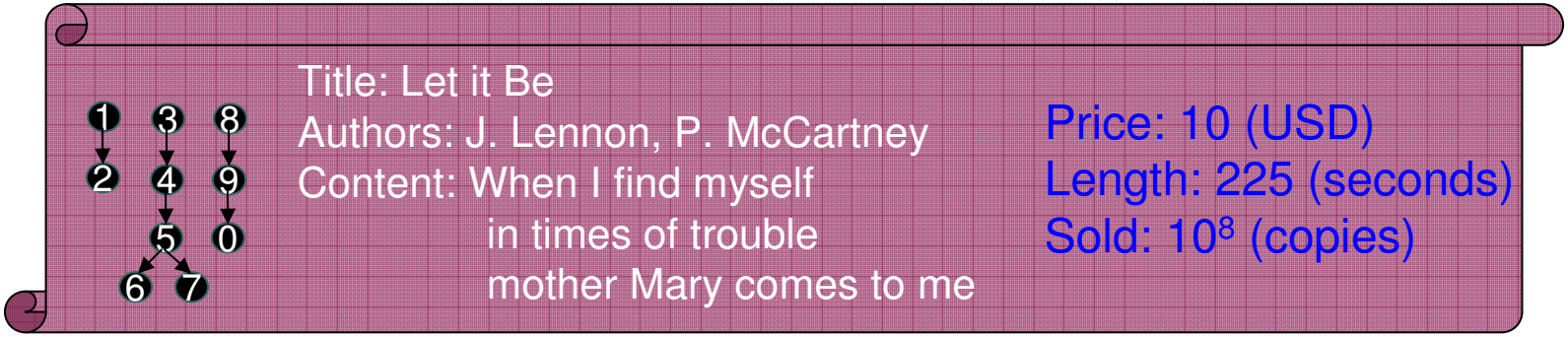


This document will be listed in the postings lists corresponding to these path prefixes

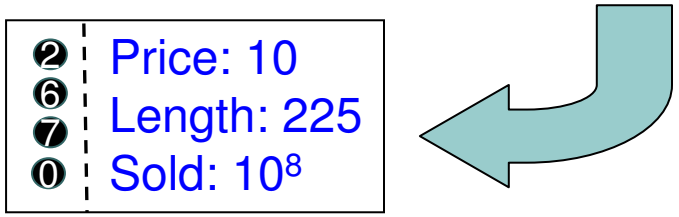
Faceted Indexing

The Inverted Index (2)

- Keeps a special postings list that allows efficient access to the paths and numeric attributes associated with each document



The payload associated with the posting element of this document:



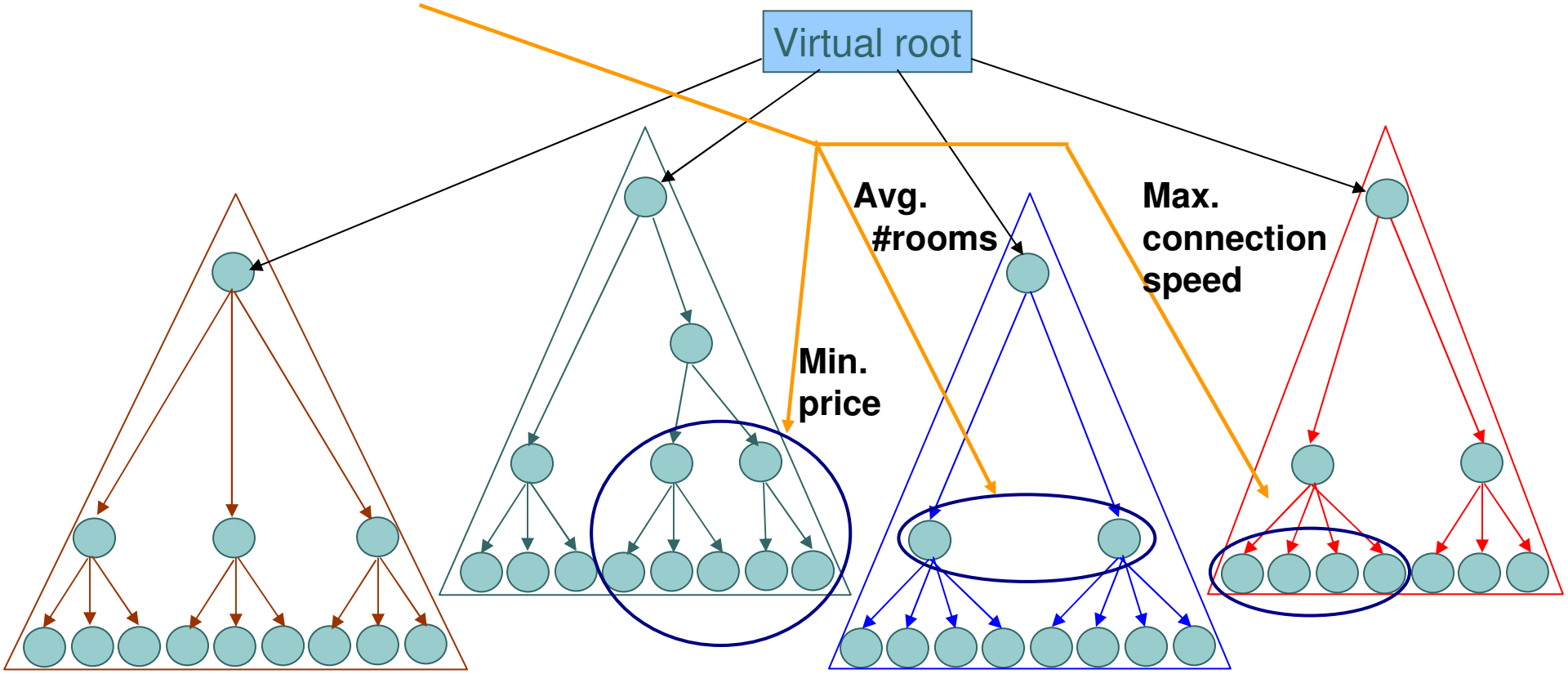
Note that only path tails are stored – higher-level nodes can be inferred using the taxonomy index

Faceted Search

Faceted Queries

Faceted queries are comprised of two parts:

1. Selection criteria: text(hotels with internet access), categories(geo:US, chain=Hilton), numerics(price<200)
2. Facet Context: sub-trees where counts/aggregations are required



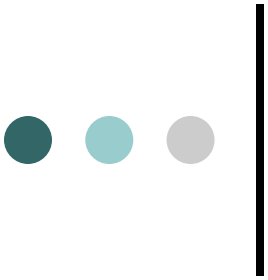
Faceted Query Evaluation

1. Given the selection portion of the faceted query, identify and rank the matching documents using the index
2. Access the special per-document entry (payload) corresponding to each matching document
3. For any category included in the facet context, aggregate counts/expressions

❷ ❻ ❼ ❿ Price: 10 Length: 225 Sold: 10^8	❷ ❻ ❽ ❿ Price: 3 Length: 215 Sold: 10^7	❹ ❺ ❼ Price: 18 Length: 185 Sold: 10^6	❺ ❽ ❾ ❿ Price: 12 Length: 220 Sold: 10^8

Category:2	Count: 2	Average price: 6.5
Category:0	Count: 3	Max sold: 10^8

Note that per query, computation time scales linearly with size of result set (number of matching docs)

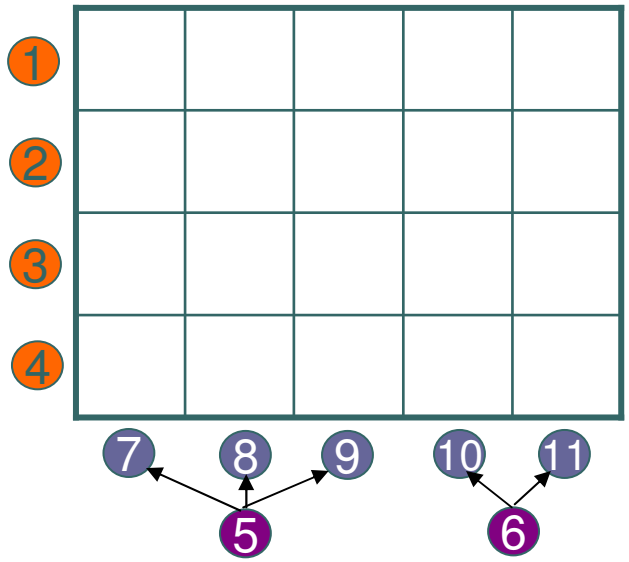


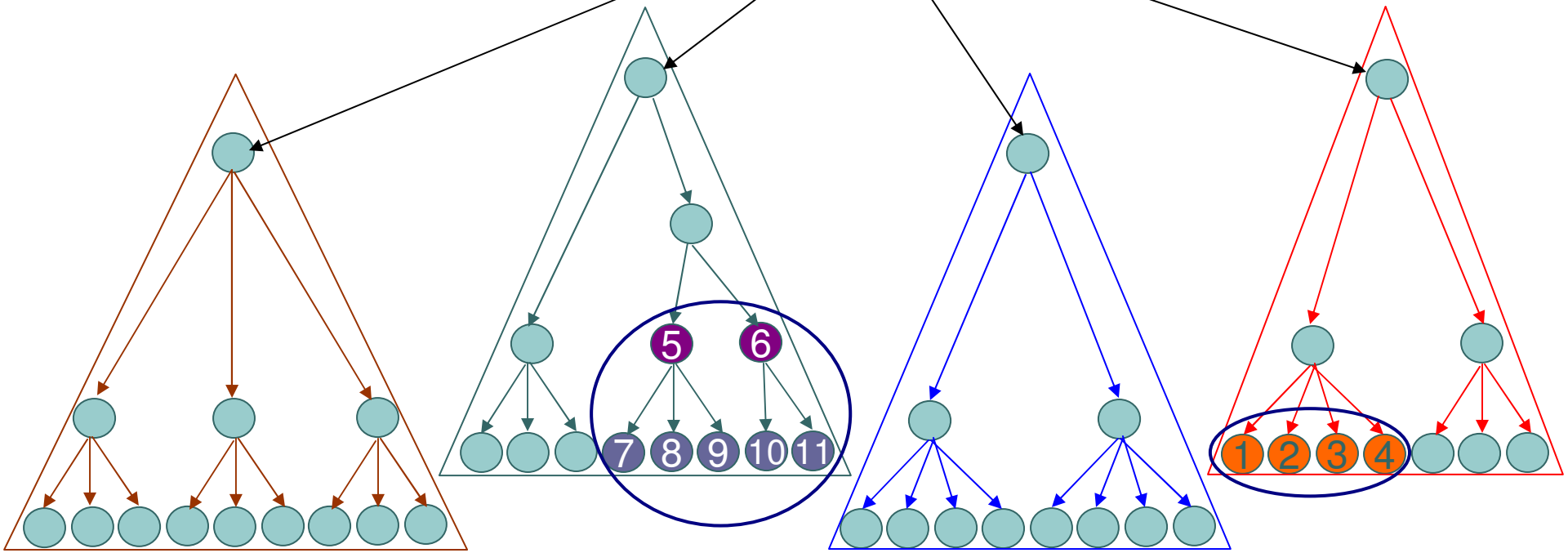
Building a Cubing Engine by a Recursive Application of Faceted Indexing

Cubing

Cubing Queries

Same as faceted queries, except that the output counts and aggregations are to be computed for the Cartesian product of the given sub-trees



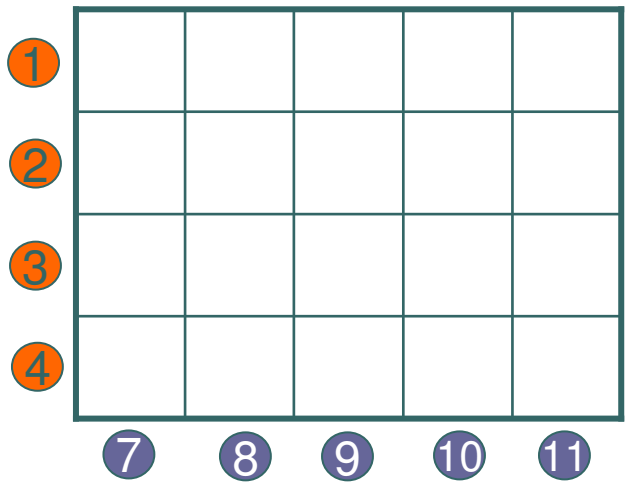


Cubing

Cubing Queries

Naïve solution:

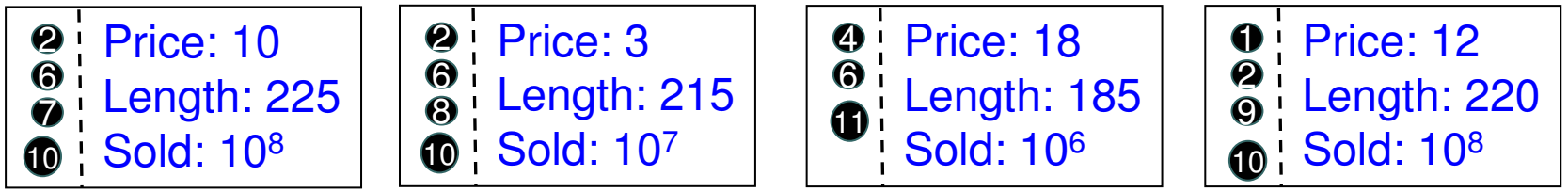
- Every cell can be computed by a single regular query, with the appropriate category constraints
- Every line can be computed by a single faceted query, given constraints on the other dimensions
- Problem: each such query runs in a time that is linear in the size of the result set, which is not acceptable for cubing queries
- So let's look at a solution that trades-off some of the runtime cost with some preprocessing



Cubing

Cubing Query Evaluation

1. Given the selection portion of the initial query, identify the matching documents
2. Access the special per-document payload corresponding to each matching document
3. Extract the Cube-DNA (CDNA) of each document, i.e. the sequence of categories to which the document belongs and which are part of the cube
   - Example: if the cubed sub-trees are {1,2,3,4} x {7,8,9,10,11}, the CDNA sequences of the sample documents are as follows:

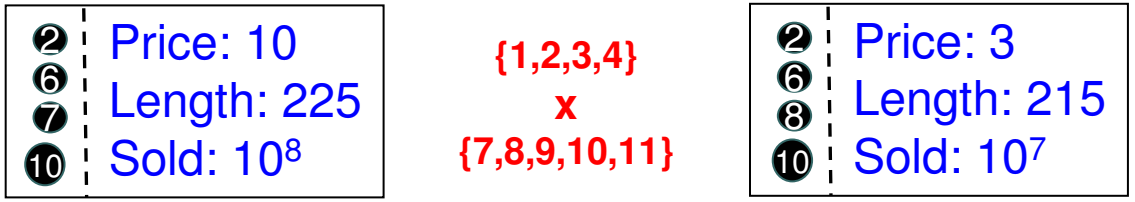


4. Aggregate counts and expressions per CDNA sequence

Cubing

CDNA Sequences - Notes

1. Each CDNA sequence affects a certain set of cube cells
   - A cell may be affected by multiple CDNA sequences



CDNA {2,7,10} affects cells {(2,7), (2,10)}

CDNA {2,8,10} affects cells {(2,8), (2,10)}

2. Documents that share the same CDNA affect exactly the same set of cube cells
3. Assumption is that for large result sets, the plurality of CDNA sequences is much smaller than the size of the result set itself
   - For small result sets, it doesn't matter - number of CDNA sequences is always bounded by number of results

Cubing

CDNA Sequences – What Now?

- CDNA Sequences per-se aren't quite what we need, as they don't map 1:1 to cube cells
- To fill a cube cell, we need to aggregate the contributions of all CDNA sequences that affect the cell

Cell (2,7):
- Count = 20+61 = 81
- Avg. Price = (20*15+61*14)/81
- Max. Sold = Max{$5*10^7$, $4*10^7$)= $5*10^7$

- So what should we do?

Sequence	Count	Avg. Price	Max. Sold
{2,4,7}	20	15	$5*10^7$
{4,9}	100	18	$9*10^6$
{3,8,9}	84	21	$6*10^6$
{1,8,10}	123	19	$2*10^8$
{2,7}	61	14	$4*10^7$
{1,7,8}	34	25	$5*10^7$
{4,7,9,11}	78	12	$3*10^7$

Cubing

Building the Cube Structure: Indexing CDNA Sequences (1)

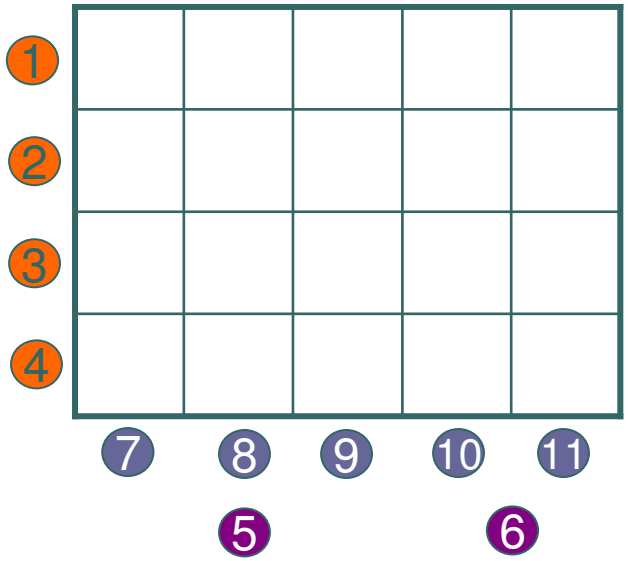
Idea:

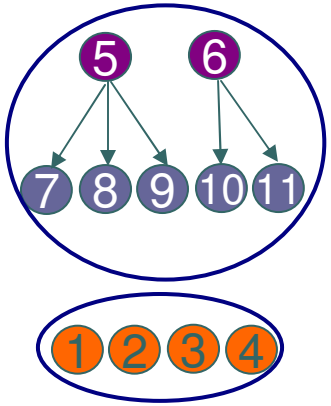
1. Create a virtual document from each CDNA sequence:
   - No textual content
   - Categories are the DNA elements
   - Numeric attributes are the count, and the calculated expressions (measures)
2. Index the virtual documents by the same faceted search mechanism!
3. The resulting faceted index will serve as the "cube"

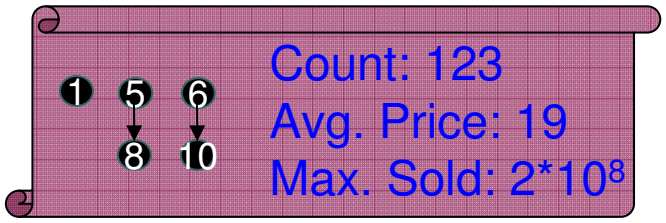
Sequence	Count	Avg. Price	Max. Sold
{2,4,7}	20	15	$5*10^7$
{4,9}	100	18	$9*10^6$
{3,8,9}	84	21	$6*10^6$
{1,8,10}	123	19	$2*10^8$
{2,7}	61	14	$4*10^7$
{1,7,8}	34	25	$5*10^7$
{4,7,9,11}	78	12	$3*10^7$

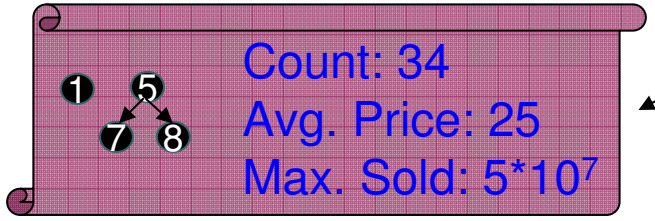
Cubing

Building the Cube Structure: Indexing CDNA Sequences (2)







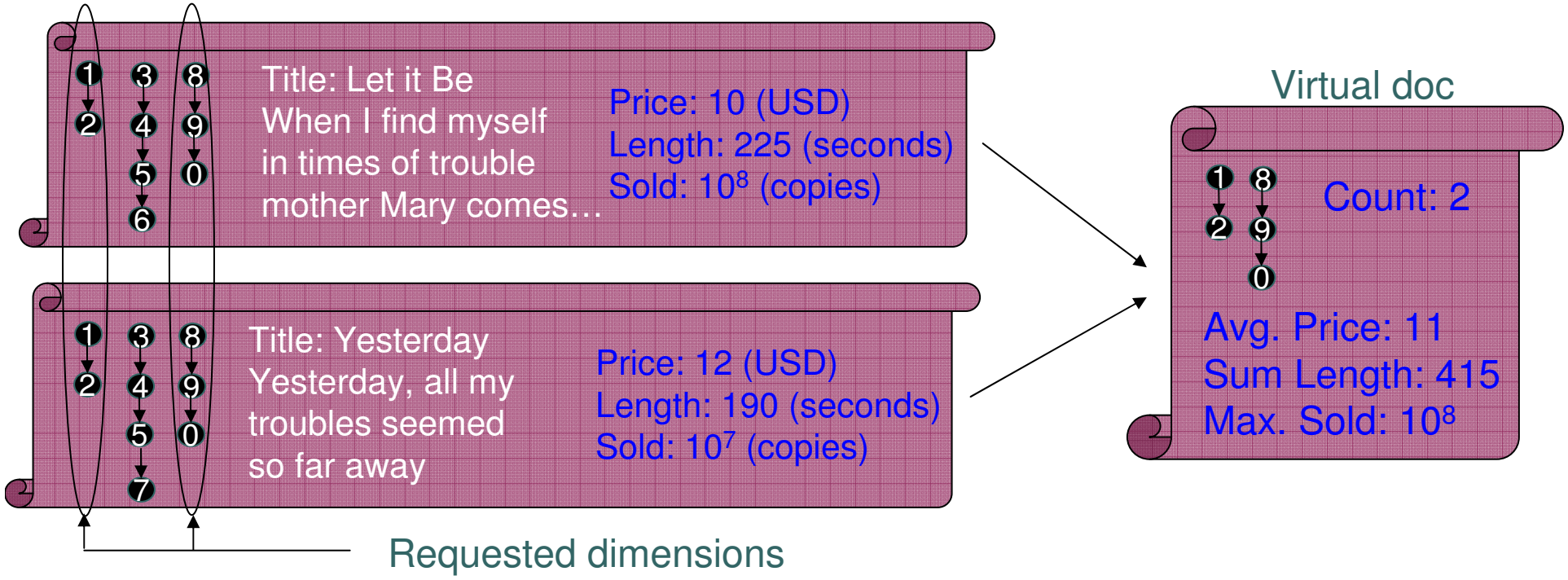


Sequence	Count	Avg. Price	Max. Sold
{2,4,7}	20	15	$5*10^7$
{4,9}	100	18	$9*10^6$
{3,8,9}	84	21	$6*10^6$
{1,8,10}	123	19	$2*10^8$
{2,7}	61	14	$4*10^7$
{1,7,8}	34	25	$5*10^7$
{4,7,9,11}	78	12	$3*10^7$

Cubing

Building the Cube Structure: The Virtual CDNA Documents

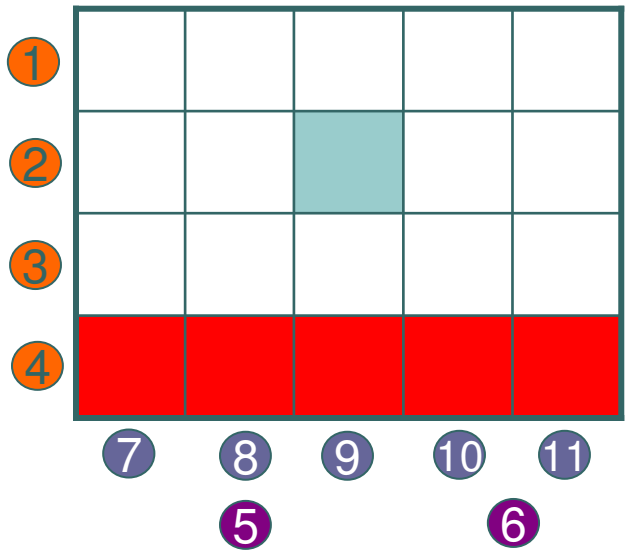
- The virtual documents represent aggregations of groups of original documents whose CDNA is identical, i.e. that are indistinguishable in the context of this cube
  - Many-to-one relationship between original documents (in OLAP terminology - facts) and CDNA virtual documents:



Cubing

The CDNA Index – Operations

- To fill the cell (2,9) – just query for documents containing category:2 category:9
- To fill the line (4,*), query for documents containing category:4 and treat categories 7-11 as a one-dimensional facet
- Queries on the CDNA index run in O(| CDNA result set |), i.e. the number of CDNA entries that affect the requested cell/hyper-plain



Summary

Recap of "Cubing" Scheme

- With one query over the original data (in the original faceted index), we can build a CDNA index in O(| result set |)
- Assumption: typically, the number of CDNA sequences per result set is not much greater than the number of populated cube cells in the requested hyper-plain
  - Number of virtual documents in CDNA index is O(|populated cells|)
  - Most cubes are sparse
  - Number of populated cube cells is typically much less than the number of (real) documents over which the cube is being built
- CDNA index CDNA index is yet another faceted index that supports cubing queries in O(| CDNA result set |)
  - It may reside on disk, in which case it is persistent and requires practically no RAM overhead – in particular, no need to materialize a cube in memory

Summary

Summary and Roadmap

- Presented a “recursive” invocation of faceted indexing that can support cubing operations
  - First invocation returns, as its result, a second index of the same type on which subsequent operations run
- Whether the performance tradeoffs implied by this approach make sense is subject to experiments
- But we like the algebraic idea nonetheless
- Use-case is real, and is yet another example of the merging between the structured, DB-oriented world and the semi-structured world of search engines

Thank You