

Indexing and Processing Big Data

Patrick Valduriez
INRIA, Montpellier



Why Big Data Today?

- **Overwhelming amounts of data**
 - Generated by all kinds of devices, networks and programs
 - E.g. sensors, mobile devices, internet, social networks, computer simulations, satellites, radiotelescopes, etc.
- **And we can now store these data!**
 - Storage capacity has doubled every 3 years since 1980 with prices steadily going down
 - 1 Gigabyte for: 1M\$ in 1982, 1K\$ in 1995, 0.12\$ in 2011
- **But what do we do with these data?**
 - Produce high-value information and knowledge
 - Critical for data analysis, decision support, forecasting, business intelligence, research, (data-intensive) science, etc.

Some estimates*

- 1,8 zetabytes (10²¹ bytes, or 1,000 exabytes)
 - An estimation for the data stored by humankind in 2011
- 40 zetabytes in 2020
- But
 - Less than 1% of big data is analyzed
 - Less than 20% of big data is protected

* Source: Digital Universe study of International Data Corporation (IDC), december 2012

3

Big Data Dimensions: the three V's

- Volume
 - Refers to massive amounts of data
 - Makes it hard to store, manage, and analyze (big analytics)
- Velocity
 - Continuous data streams are being produced
 - Makes it hard to perform online processing
- Variety
 - Different data formats, different semantics, uncertain data, multiscale data, etc.
 - Makes it hard to integrate and analyze
- Other V's
 - Validity: is the data correct and accurate?
 - Veracity: are the results meaningful?
 - Volatility: how long do you need to store this data?

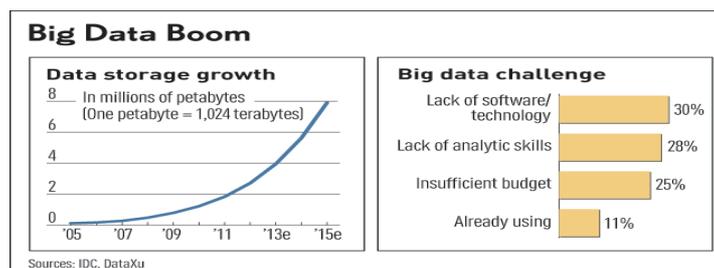
4

Big Data Dimensions: the three V's

- **Volume** Parallel database systems
 - Refers to the amount of data
 - Makes it hard to store (distributed systems)
- **Velocity** Data stream management systems
 - Continuous data flow
 - Makes it hard to process
- **Variety** Data integration systems
 - Different data formats, different semantics, uncertain data, multiscale data, etc.
 - Makes it hard to integrate and analyze
- **Other V's**
 - **Validity**: is the data correct and accurate?
 - **Veracity**: are the results meaningful?
 - **Volatility**: how long do you need to store this data?

5

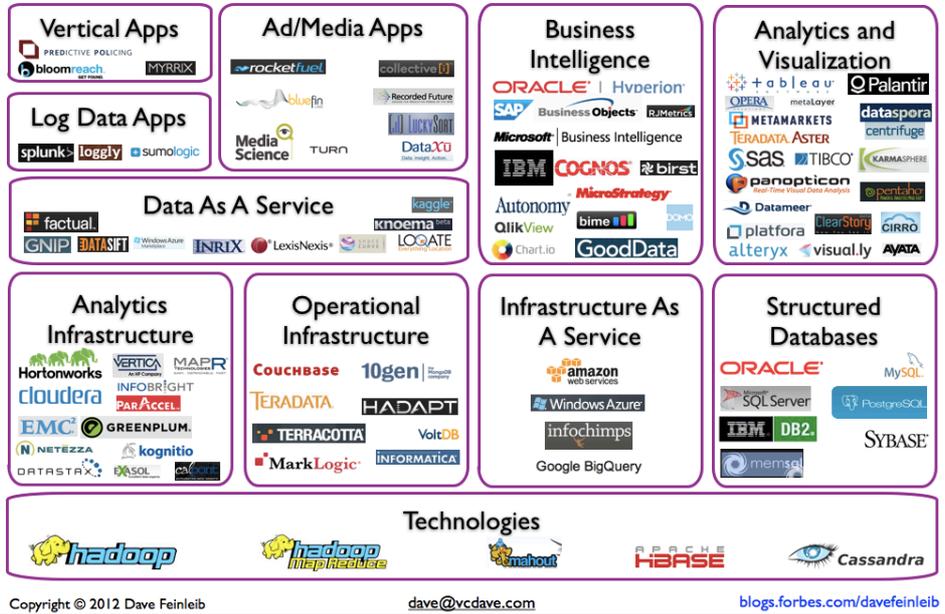
Big Data Challenge



- **The needs are in technology**
 - New architectures, algorithms, techniques
- **AND technical skills**
 - Experts in using the new technology and dealing with big data, e.g. big data scientists

6

Big Data Landscape



Big Data Landscape



Many Different Types of Big Data

- **Key-value**
 - The simplest structure
- **Relational table**
 - Nested table, Bigtable
- **Array**
 - Nested array, multidimensional
- **Document**
 - Unstructured text data (Web)
 - Semi-structured data (XML, JSON, etc.)
- **Graph**
 - Social network, Semantic Web (RDF), ...
- **Data stream**
 - Data (from sensors, etc.) that continuously flow in
- **Time series**
 - Sequence of data points, at uniform time intervals

9

Outline of the Talk

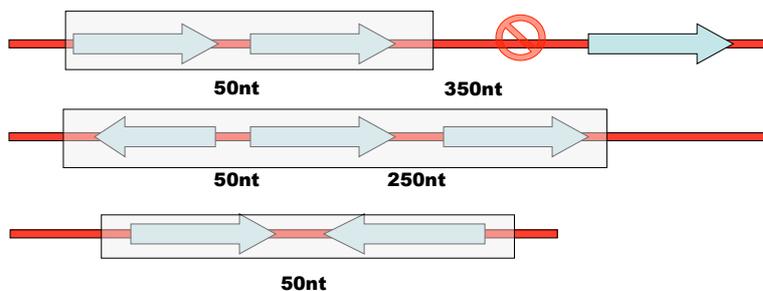
- Big data example
- Data indexing
- Parallel data processing
- Big data indexing using bitmaps
- Conclusion

Big Data: example*

- * A. Romosan, A. Shoshani, K. Wu, V. Markowitz, K. Mavrommatis. Accelerating Gene Context Analysis Using Bitmaps. SSDBM 2013.

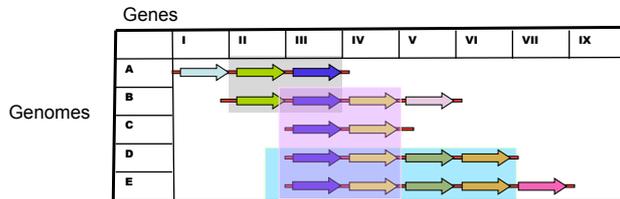
Gene Functional Annotation

- **Sequence similarity**
 - Gene context provides information for the function of genes
 - Functionally related genes are frequently found in the same chromosomal neighborhood
- **Gene cassette**
 - A modular DNA sequence encoding one or more genes for a single biochemical function
 - Parallel or Divergent orientation
 - Distance < 300nt



Conserved Gene Regions

- Groups of at least two common genes between two or more gene cassettes
- Genes are replaced by protein families (COGs, pfams, etc.)
 - One gene => multiple families
 - Referred to as "properties", such as "cog0087 cog0088 pfam00181 pfam00189 pfam00203"
- Example of two-or-more conserved regions across multiple genomes
 - In gray: 2 genes across 2 genomes
 - In pink, 2 genes across 4 genomes
 - In blue, 4 genes across 2 genomes



13

Gene Cassette Search Tool at IMG*

Cassette Search Results (ii)

Cassette ID	Cassette Gene Count	Function ID	Function Name	Gene ID	Genome Name
Number of cassette members: 3					
646042713	128	COG0126	3-phosphoglycerate kinase	646042713	Chlorobacterium phages phiC100
646042714	62	COG0126	3-phosphoglycerate kinase	646042714	Chlorobacterium phages phiC100
646042715	62	COG0126	3-phosphoglycerate kinase	646042715	Chlorobacterium phages phiC100

Phylogenetic Profiler for Gene Cassettes Results By COG Conserved Cassettes

Thermoplasma acidophilum DSM 1728 (v)

243 (58.88%) gene cassettes in query genome from a total of 412
 849 (55.31%) genes in query genome from a total of 1535 genes.

No. of Conserved Cassettes	Occurrences
29	1
13	1
12	1
11	3
10	3
9	2

Chromosomal Cassette By COG (vi)

Select	COG ID	COG Name	Gene ID
<input type="checkbox"/>	COG0001	Transferrinase	646042133
<input type="checkbox"/>	COG0002	Glyoxaldehyde-3-phosphate	646042134
<input type="checkbox"/>	COG0003	Adipoyl-CoA synthetase-3	646042135
<input type="checkbox"/>	COG0004	Polyphosphatase (cytochrome oxidase	646042136
<input type="checkbox"/>	COG0005	3-phosphoglycerate kinase	646042137
<input type="checkbox"/>	COG0006	Transphosphatase	646042138

*Integrated Microbial Genomes (IMG), Lawrence Berkeley Labs

14

Why is this problem hard?

- **Very big data**
 - Predictions: 100 million cassettes, with properties from about 25,000 possible genomes
 - Total number of elements: 2.5×10^{12}
- **Query types**
 - Given a cassette, find all cassettes that have the same properties in common
 - That is a massive multi-value search
 - Given a cassette find all cassettes that have 2-or-more properties in common
 - Explosive search of all possible combinations of 2-or-more
- **Experiments @ IMG**
 - With an RDBMS
 - 3.3 million gene cassettes across 8,000 genomes
 - Correlation table of more than 2 billion rows along with a dozen auxiliary tables
 - 16.5 hours to build and a typical query requires 5 to 10 minutes to answer

15

Data Indexing

Data Storage: row versus column



+ Add/delete row efficient

- Reads of useless data

+ Efficient access to useful data

- Add/delete row inefficient

- Column-store better for
 - Data-intensive apps (OLAP, not OLTP)
 - Big data
 - With lots of columns (or dimensions)

17

Purpose of Data Indexing

- Accelerate the access to data (records, objects, documents, etc.) in
 - Data containers: files, databases (row-store or column-store)
 - Each with its API
 - By reducing the number of disk and/or memory accesses
- Basic functions (direct access using primary key)
 - Put (key, data), Update (key, data)
 - Get (key): data
 - Delete (key)
- Search or query types
 - Exact match (point query), partial match
 - Range
 - Multikey
 - Top-k, skyline
 - Similarity (content-based)
 - Keyword
 - Etc.

18

Index Data Structure

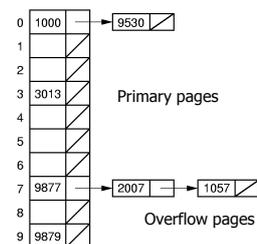
- Given that data are stored in fixed units of access
 - Disk pages, memory areas
- Associates keys with corresponding data
 - Requires each data to be uniquely identified
 - On disk: RID, TID, ... (e.g. disk page address)
 - In main memory: by a pointer to the data
- Organizes keys in a way that speeds up key lookup
- Basic structures
 - Hash table
 - Tree-based index
 - Multidimensional index
 - Bitmap index

19

Hash Table

- $h(\text{key})$, e.g. $\text{key} \bmod n$ yields a page number
- Good for exact match search
 - Access in $O(1)$
- Static hashing
 - Lists of overflow buckets degrade performance => periodic reorg.
- Dynamic hashing
 - Extendible hashing [Fagin 79], linear hashing [Litwin 80]
 - Allows the file to grow and shrink without overflowing
 - Needs a (compact) index, e.g. bit table

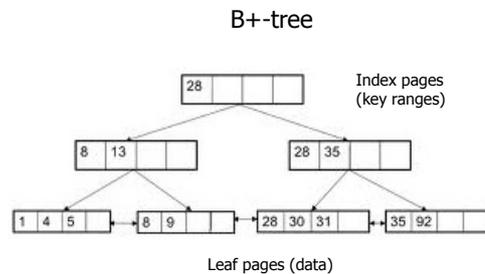
Static hashing



20

Tree-based Index

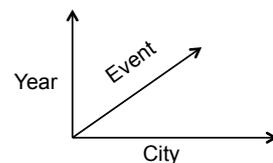
- Preserves key order
- B-tree
 - Exact match query
 - Range query
 - Access in $O(\log n)$
 - Dynamic updates
- Many variations
 - Disk-based: B+-tree [Bayer 72]
 - Memory-based: T-tree [Lehman 86]
- Issues
 - Index size
 - Building cost



21

Multidimensional data

- Data represented by points in a multidimensional space
 - A dimension is like a column in a relational table
- Straightforward indexing
 - One primary (also called *placement*) index on one key
 - For instance, with a B+-tree index, we get *clustered* access to data in the same interval
 - Secondary indexes on other keys
 - With *random* access to the data
 - *But only for point and range queries*
 - Sequential scan better for other queries



A 3D point and its data
(Paris, 2014, Mastodons)
= [2 k, 80 p]

22

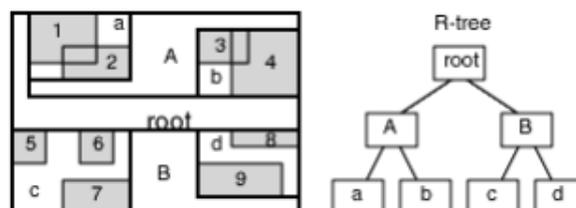
Multidimensional Index

- Needed to store and access spatial data
 - E.g. geographical coordinates or geometric objects (lines, polygons, circles, etc.)
- With more powerful search capabilities
 - Point and range queries
 - Proximity (window query), spatial join
 - Similarity: Nearest Neighbour Search
 - Similarity measure: application dependent
- Index structures for a few dimensions (<10)
 - R-tree [Guttman 84], KD-tree [Bentley 75], Predicate trees [Valduriez 84]
 - Variants of R-trees: R*, R+, X-tree

23

R-tree Example

- Each multidimensional data (e.g. a geometry) is approximated by a single rectangle (called the *minimum bounding rectangle*) that minimally encloses it
- Works well with up to four dimensions
 - E.g. spatio-temporal: altitude, longitude, latitude, time
- Access in $O(\log n)$, but worst case insert in $O(n)$
- The issue is to minimize coverage and overlap of rectangles



24

High Dimensional Data

- Applications
 - Multimedia: image, video
 - Dimensions: object features
 - Document retrieval, e.g. keyword search
 - Dimensions: document terms
 - Biology, e.g. gene functional annotation
 - Dimensions: genes
- Problem: curse of dimensionality
 - Exponential growth of the hypervolume as a function of dimension
- Typical solutions
 - Partition and reduce the high dimensional space
 - While preserving partition balancing and locality
 - Using multiple hash tables or tree structures (randomized KD-tree, hierarchical k-means, etc.)
 - Compress the data
 - Using hashing techniques (e.g. hamming embedding)

25

Bitmap Index

- Easy to build: faster than B-trees
- Efficient for querying: only bitwise logical operations
 - $X < 2 \Rightarrow b_0 \text{ OR } b_1$
 - $X > 2 \Rightarrow b_3 \text{ OR } b_4$
- Efficient for multi-dimensional queries
 - Use bitwise operations to combine the partial results
- Size: one bit per distinct value per object
 - Cardinality == number of distinct values
 - Need to control size for high cardinality attributes
- Solution
 - Efficient compression method
 - Operations directly on compressed data

RID	X	bitmaps				
		b_0 =0	b_1 =1	b_2 =2	b_3 =3	b_4 =4
1	1	0	1	0	0	0
2	0	1	0	0	0	0
3	4	0	0	0	0	1
4	2	0	0	1	0	0
5	3	0	0	0	1	0
6	3	0	0	0	1	0
7	1	0	1	0	0	0
8	4	0	0	0	0	1

26

Other Major Structures

- **Inverted files**
 - Special tables with rows of the form: $\langle \text{value}, \text{list of keys} \rangle$ pairs
 - Ex. $\langle \text{value}, (\text{doc-id:}id_1, \text{doc-id:}id_2, \text{doc-id:}id_{10}) \rangle$
 - Given an attribute value, returns all corresponding keys
 - Like secondary indexes, these keys can in turn be used to access the corresponding data using a primary index
- **Join indices [Valduriez 87]**
 - Capture the join of two tables in a compact way
 - Also useful for complex structures, e.g. graphs
 - Speeds up complex query processing
 - Many variations, e.g. Oracle bitmap join indices

27

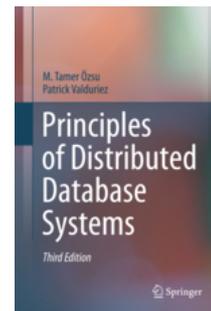
Big Data Indexing Requirements*

- **Speed of search**
 - Search over billions – trillions data values in seconds
- **Multi-variable queries**
 - Be efficient for combining results from individual variable search results
- **Size of index**
 - Index size should be a fraction of original data
- **Parallelism**
 - Should be easily partitioned into pieces for parallel processing
- **Speed of index generation**
 - For in situ processing, index should be built at the rate of data generation

* A. Shoshani. On the Role of Indexing in Scientific Domains. Bigdata and Extreme Computing, 2013.

28

Parallel Data Processing

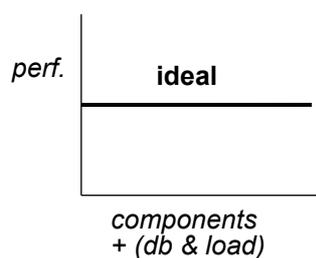
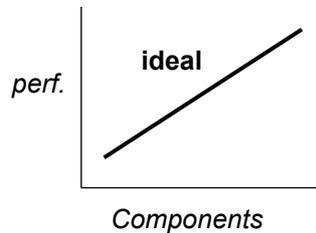


The solution to big data processing!

- **Exploit a massively parallel computer**
 - A computer that interconnects lots of CPUs, RAM and disk units
- **To obtain**
 - *High performance* through data-based parallelism
 - High throughput for OLTP loads
 - Low response time for OLAP queries
 - *High availability* and reliability through data replication
 - *Scalability* of the architecture

Scalability : ideal goals

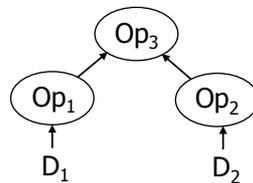
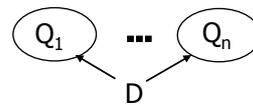
- *Linear increase* in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)
- *Sustained performance* for a linear increase of database size and load, and proportional increase of components



31

Data-based Parallelism

- **Inter-query**
 - Different queries on the same data
 - For concurrent queries
 - Lots of queries
- **Inter-operation**
 - Different operations of the same query on different data
 - For complex queries
 - Lots of ops/query
- **Intra-operation**
 - The same operation on different data
 - For large queries
 - Lots of data/op.



32

Shared-nothing (SN) Cluster

No sharing of either memory or disk across nodes

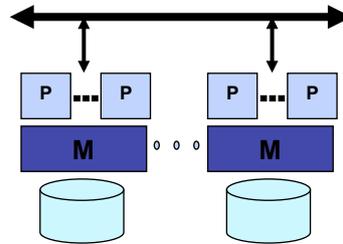
- Needs careful data partitioning

- Examples

- Parallel RDMS: DB2 DPF, SQL Server Parallel DW, Teradata, MySQLcluster
- Search engines: Google search
- NoSQL key-value stores (Bigtable, ...)

+ highest extensibility, cost

- updates, distributed trans.



Perfect match for big data (read intensive)

33

Parallel Techniques: design considerations

- **Big datasets**

- Data partitioning and indexing
 - Problem with skewed data distributions
- Disk is very slow (*100K* times slower than RAM)
 - Exploit RAM data structures and compression techniques
 - Exploit SSD (read 10 times faster than disk)

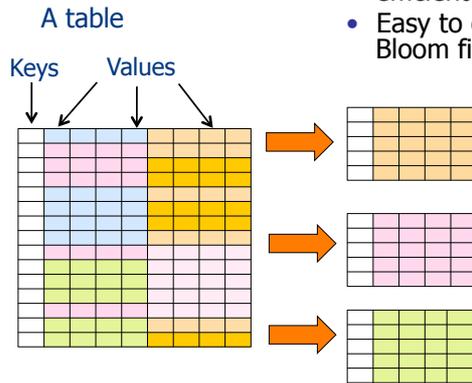
- **Query parallelization and optimization**

- Automatic if the query language is declarative (e.g. SQL)
 - Parallel algorithms for algebraic operators
 - Complex if multiple datasets (join, etc.)
- Programmer-assisted otherwise (e.g. MapReduce)

34

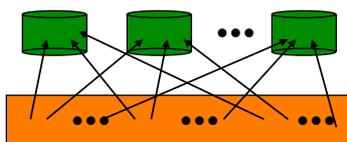
Data Partitioning

- **Vertical partitioning**
 - Base Basis for column stores (e.g. MonetDB, Vertica): efficient for OLAP queries
 - Easy to compress, e.g. using Bloom filters



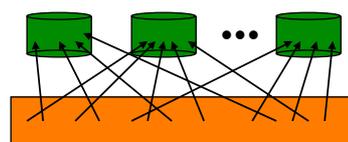
- **Horizontal partitioning (sharding)**
 - Shards can be stored (and replicated) at different nodes

Sharding Schemes



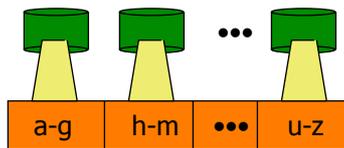
Round-Robin

- i th row to node $(i \bmod n)$
- perfect balancing
- but full scan only



Hashing

- (k,v) to node $h(k)$
- exact-match queries
- but problem with skew



Range

- (k,v) to node that holds k 's interval
- exact-match and range queries
- deals with skew

Parallel Indexing

- **2 level indexing**
 - Global level: key -> node
 - Like a table, should be partitioned and replicated
 - Problems
 - Index partitioning
 - Update propagation for replicas
 - Local level (within a node): like a normal index
- **Easier with**
 - Hash tables, bitmaps, inverted files
- **More complex with trees or graphs**
 - Index partitioning
 - Replicate top levels, partition low levels

37

Big Data Indexing using Bitmaps

Gene Cassette Search*

- **Recall: with an RDBMS**
 - 16.5 hours to build the correlation table and a typical query requires 5 to 10 minutes to answer
- **Fastbit (sdm.lbl.gov/fastbit)**
 - Efficient compressed bitmap technology
 - Logical ops directly on compressed data
- **Results with Fastbit**
 - Reading the input data takes about 1.5 hours and constructing the bitmaps takes only 8 minutes
 - "Killer queries" in seconds
 - An extremely complex query involving 160 genomes (needing a 160- way cross-product) takes less 10 seconds

Vertical bitmap

	Property 1	Property 2	Property 3					
Cassette 1	1	0	0	0	0			
Cassette 2	0	1	0	0	0			
Cassette 3	0	0	1	0	0			
	0	0	0	1	0			
	0	1	0	0	0			

* A. Romosan, A. Shoshani, K. Wu, V. Markowitz, K. Mavrommatis. Accelerating Gene Context Analysis Using Bitmaps. SSDBM 2013.

Scalability

- **Problem: the number of genomes sequenced is growing faster than Moore's law**
 - One of the reasons is the sequencing of combination of genomes, called meta-genomes
 - E.g., a soil sample has in it a large number of bacteria, each with its own genome
- **Fastbit solution**
 - Horizontal partitioning of bitmaps
 - For example, if we have 1 billion rows, it is possible to partition them into 100 chunks of bitmaps for every 10 million rows, with each chunk to be provided to one of 100 cores for parallel processing
 - Issue: chunk load balancing

Conclusion

Summary

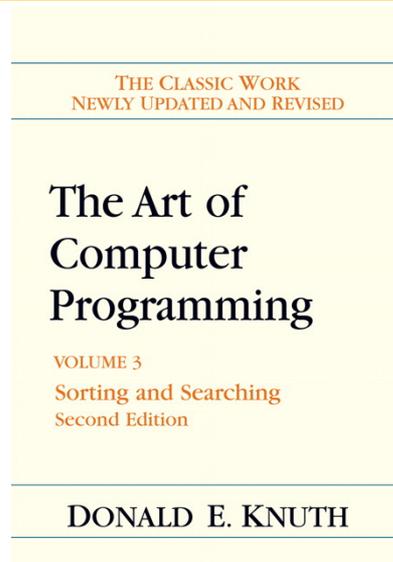
- Major problems in big data indexing
 - Scalability
 - Index building time
 - High dimensional indexing
- Solutions must combine indexing and parallel processing
 - Example Hadoop++ [Dittrich 12]
 - Adds B-tree indexes to Hadoop MapReduce
- Many different solutions
 - Different trade-offs
 - Access time vs. update time, row-store vs. column-store, etc.
 - Different objectives and apps

Research Directions

- **Much room for research and innovation**
 - Database cracking [Ideos 13]
 - Building the index dynamically during query processing
 - Easier with column-store
 - Algorithms that exploit new hardware capabilities
 - Memory hierarchies
 - Large RAM memory
 - Towards 1 Terabyte RAM chips
 - Flash memory and SSD
 - Parallel processing hierarchies
 - Multicore and GPUs
- **Bottleneck is in experimentation**
 - We need big data platforms to experiment with

43

My Favorite Old Book on Indexing



First Edition: 1973

44