22. Object Relational Systems

- incorporate the benefits of OO languages
  - Complex data types (sets, arrays, text, blobs, unnested relations)

title	*author-set*	*publisher*	*keyword-set*
		(*name, branch*)	
Compilers	{Smith, Jones}	(McGraw-Hill, New York)	{parsing, analysis}
Networks	{Jones, Frick}	(Oxford, London)	{Internet, Web}

  - Classes of objects, methods & property inheritance

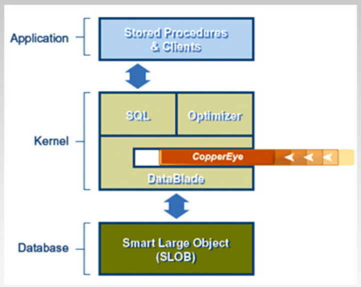
```
create table customer (      name           Name,
                             address        Address,
                             dateOfBirth date)
        create type CustomerType as (name Name,
                             address Address,
                             dateOfBirth date)
```

  - Inheritance was implemented using joins on foreign keys

© Nick Roussopoulos

OR User Defined Types & Access Methods

- Initially termed Extensible Relational
  - until it was more profitable to change the name (helped in the death of OODB and claimed all merits)
- Datablades (Oracle calls them data catridges, DB2 Extenders)



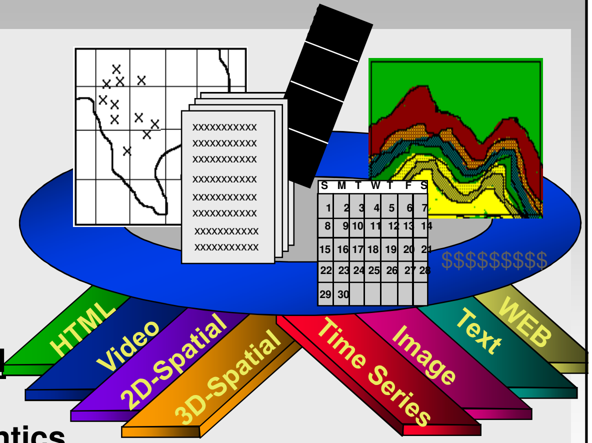
- Performance issue: where does the datablade code run?
  - in the kernel (Informix, DB2 extenders)
  - In the user space (Oracle's cartridge, DB2 extenders)

© Nick Roussopoulos

DataBlade Variety

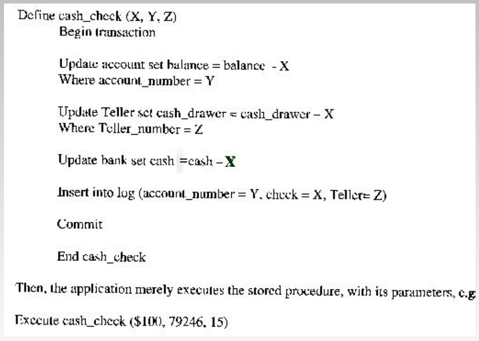
Total Extensibility
- DataTypes
- Access Methods
- Data Behavior

The OR-DBMS retained
- SQL compatibility
- Transactional Semantics



Stored Procedures

- Similar idea for executing multiple SQL commands (Sybase late 90's)
  - Instead of in and out from DBMS kernel, get in once and execute a bunch of commands as one piece of code (like user-defined function):



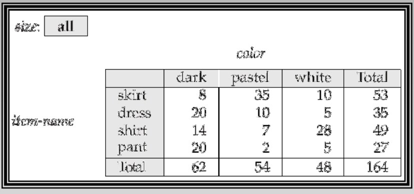
```
Define cash_check (X, Y, Z)
      Begin transaction

      Update account set balance = balance  - X
      Where account_number = Y

      Update Teller set cash_drawer = cash_drawer – X
      Where Teller_number = Z

      Update bank set cash  =cash – X

      Insert into log (account_number = Y, check = X, Teller= Z)

      Commit

      End cash_check
Then, the application merely executes the stored procedure, with its parameters, e.g

Execute cash_check ($100, 79246, 15)
```

  - Benefit: performance
  - Requires error handling inside such a procedure

Chapter 20: Data Analysis and Mining

- **Data Analysis and OLAP**
  - **For each product category and each region, what were the total sales in the last quarter and how do they compare with the same quarter last year**
- **A** data warehouse **archives information gathered from multiple sources, and stores it under a unified schema, at a single site.**
  - **Important for large businesses that generate data from multiple divisions, possibly at multiple sites**
- **Data Mining**
  - **Discover automatically patterns and behavior of customers by category (income, location, education, etc.)**

Cross Tabulation of *sales* by *item-name* and *color*



size: all

		color			
		dark	pastel	white	Total
item-name	skirt	8	35	10	53
	dress	20	10	5	35
	shirt	14	7	28	49
	pant	20	2	5	27
	Total	62	54	48	164

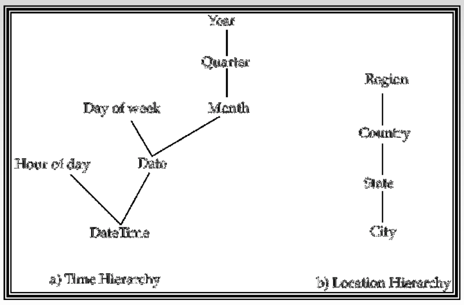
- **The table above is an example of a** cross-tabulation (cross-tab)**, also called** pivot-table**.**
  - **Values for one of the dimension attributes form the row headers**
  - **Values for another dimension attribute form the column headers**
  - **Other dimension attributes are listed on top**
  - **Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.**

- Cross-tabs can be represented as relations
  - **all** is used to represent aggregates

item-name	*color*	*number*
skirt	dark	8
skirt	pastel	35
skirt	white	10
skirt	**all**	53
dress	dark	20
dress	pastel	10
dress	white	5
dress	**all**	35
shirt	dark	14
shirt	pastel	7
shirt	white	28
shirt	**all**	49
pant	dark	20
pant	pastel	2
pant	white	5
pant	**all**	27
all	dark	62
all	pastel	54
all	white	48
all	**all**	164

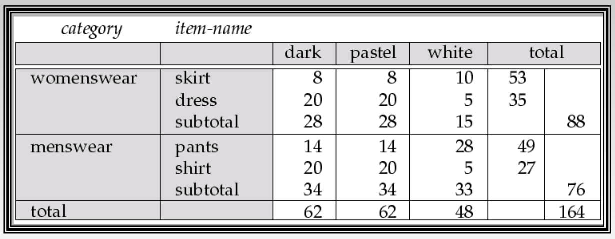
Multi-Dimensional & Hierarchies on Dimensions

- **Large number of dimensions (perhaps hundreds)**
- **Hierarchies** on dimension attributes: lets dimensions to be viewed at different levels of detail
  - E.g. the dimension DateTime can be used to aggregate by hour of day, date, day of week, month, quarter or year



Cross Tabulation With Hierarchy

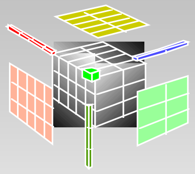
- Cross-tabs can be easily extended to deal with hierarchies
  - Can drill down or roll up on a hierarchy

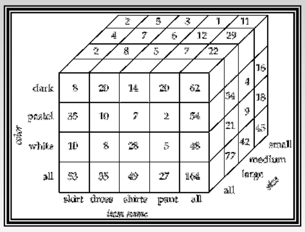


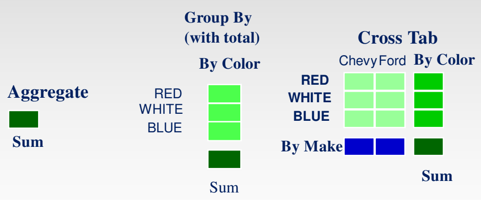
category	*item-name*					
		dark	pastel	white	total	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

Better Represented by Data Cubes

- A data cube is a multidimensional generalization of a cross-tab
- Typical *10-200* dimensions; visualize only 3 below
- Cross-tabs can be used as views with aggregates on a data cube
  - Total= 2^n views (projections)- Extremely Large Storage



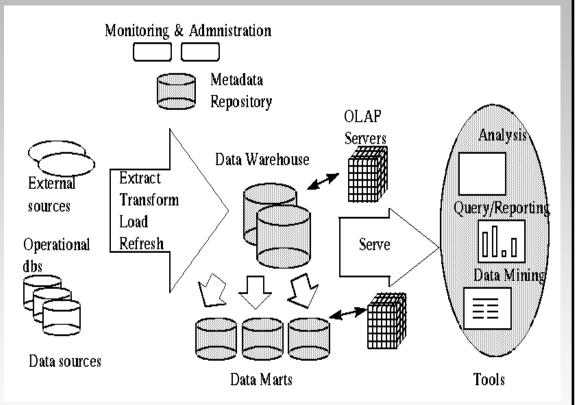




- Slice & Dice the data
  - revenue by customer
  - profit/revenue by supplier

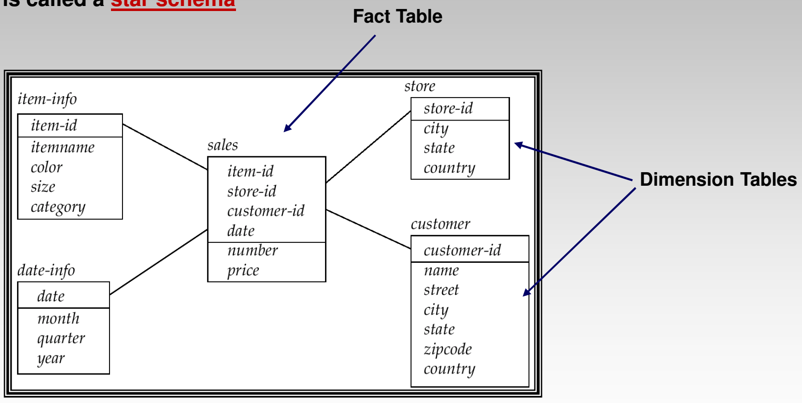
Data Warehousing Architecture

- Extract data from OLTP/external sources
  - Cleaning, transformation
- Bulk load/refresh
  - Warehouse is offline
  - perhaps not
- Got bigger than the sources
  - history, not a snapshot
- Performance is the issue
  - queries are typically long and slow
  - refresh of the warehouse is very slow



Data Warehouse Schemas

- Dimension values are usually encoded using small integers and mapped to full values via dimension tables
- This is called a star schema



407

© Nick Roussopoulos

Online Analytical Processing (OLAP)

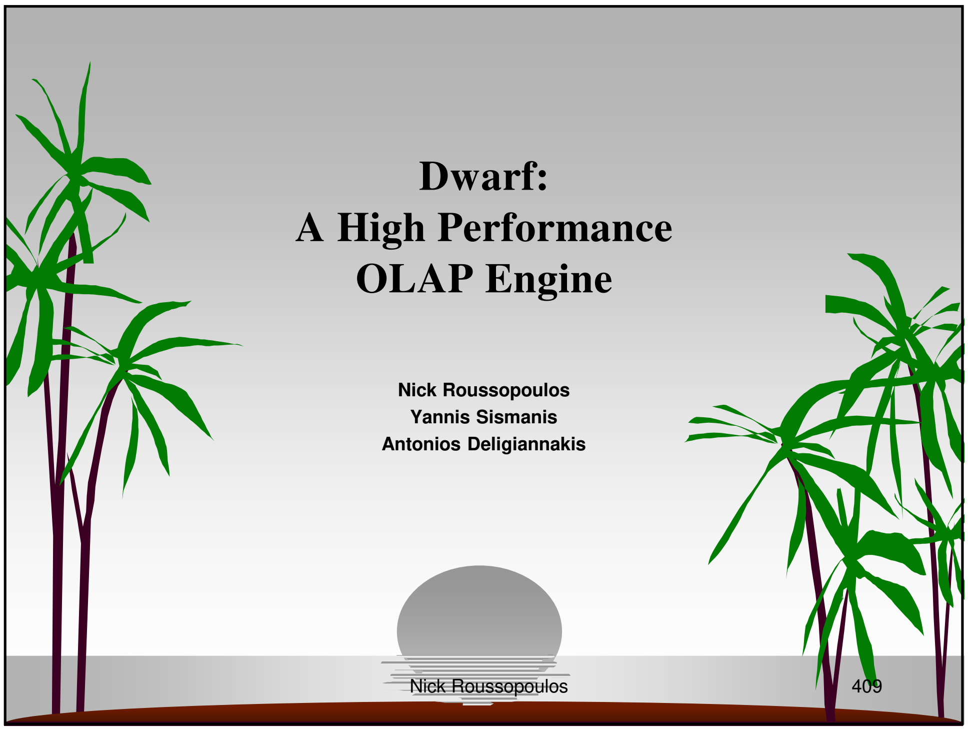
- Slicing: creating a cross-tab for fixed values only
  - Sometimes called dicing, when values for multiple dimensions are fixed
  - Correspond to group by with aggregation (sum, count, etc) queries

```
select  item-id,store-id, sum(number)
from    sales
where 19980100 < date< 19991232
group by item-id, store-id
```

- Rollup: moving from finer-granularity data to a coarser granularity
- Drill down: The opposite operation - that of moving from coarser-granularity data to finer-granularity data

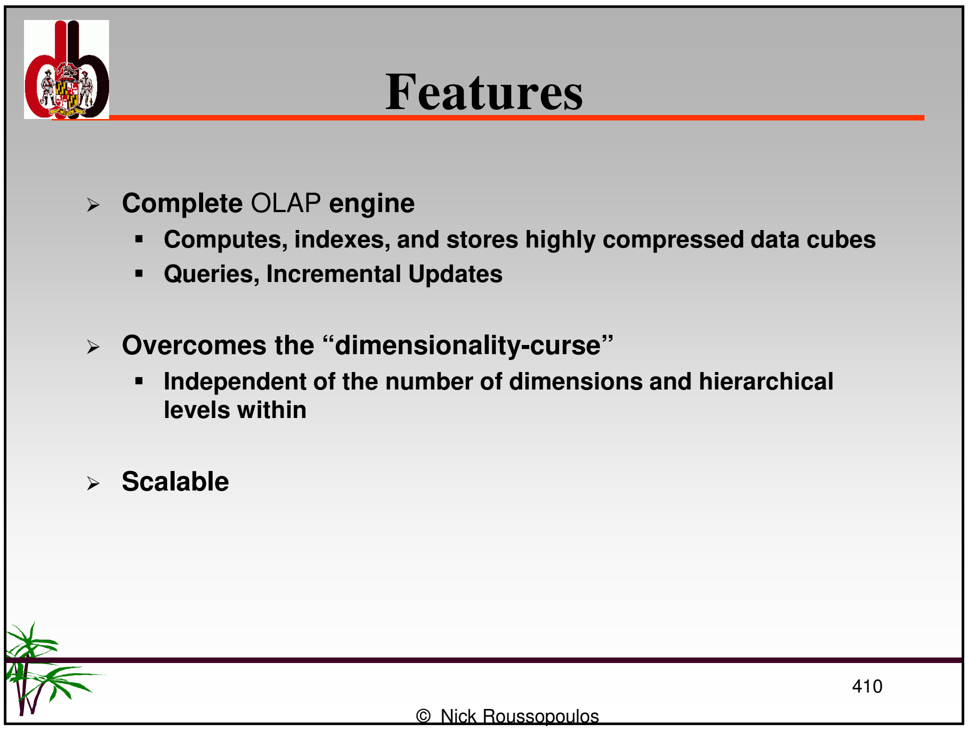
© Nick Roussopoulos

408



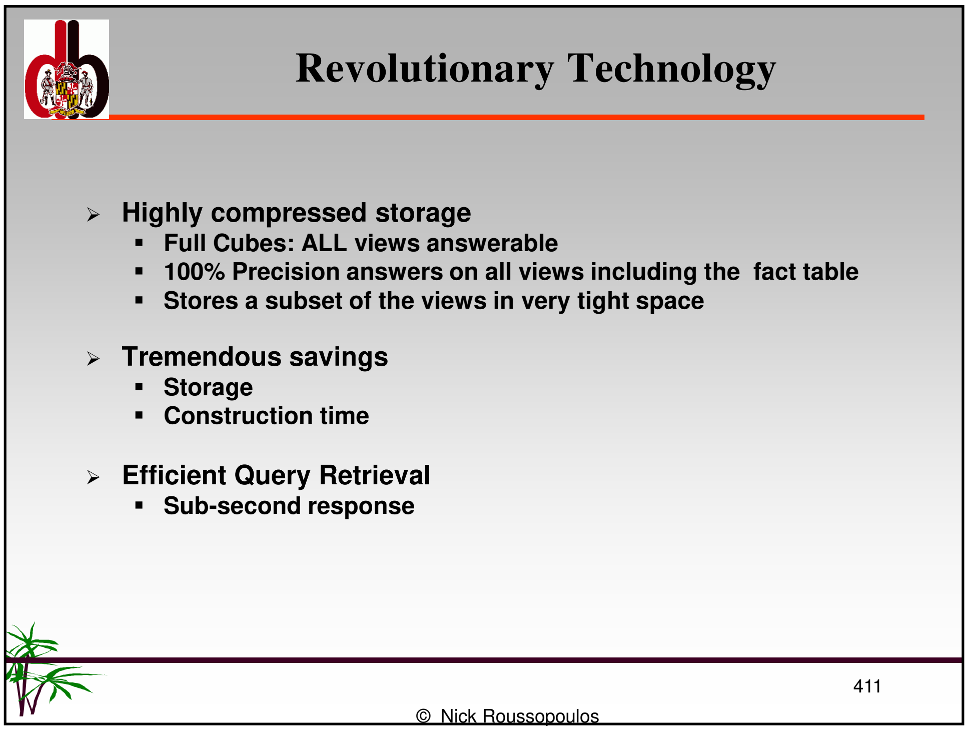
Dwarf: A High Performance OLAP Engine

Nick Roussopoulos
Yannis Sismanis
Antonios Deligiannakis



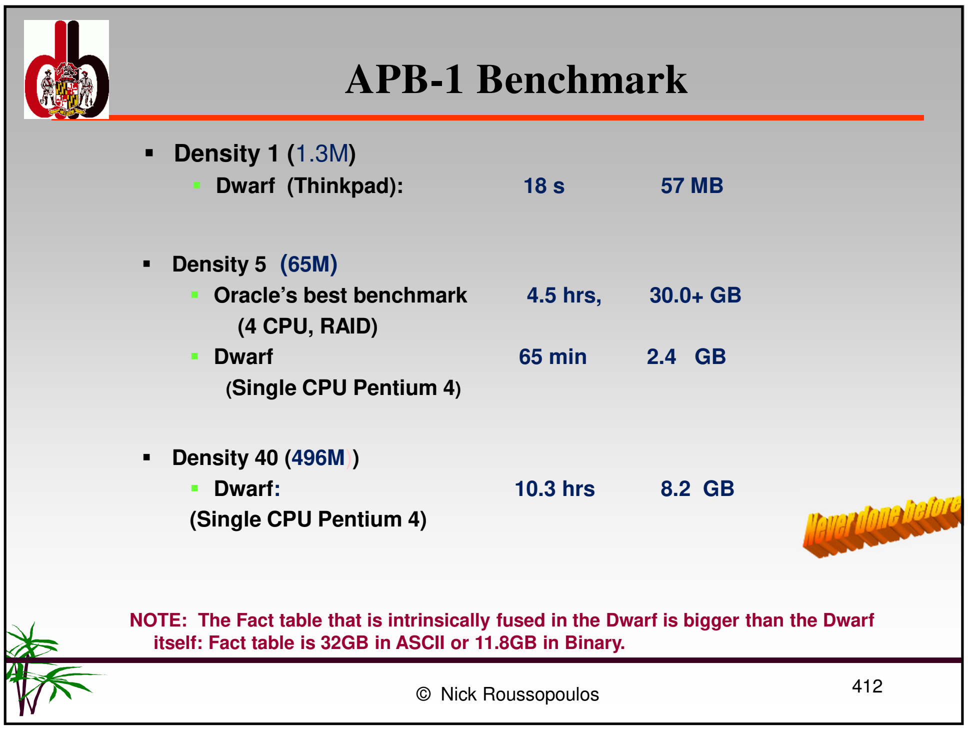
Features

- Complete OLAP engine
  - Computes, indexes, and stores highly compressed data cubes
  - Queries, Incremental Updates
- Overcomes the "dimensionality-curse"
  - Independent of the number of dimensions and hierarchical levels within
- Scalable



Revolutionary Technology

- Highly compressed storage
  - Full Cubes: ALL views answerable
  - 100% Precision answers on all views including the fact table
  - Stores a subset of the views in very tight space
- Tremendous savings
  - Storage
  - Construction time
- Efficient Query Retrieval
  - Sub-second response



APB-1 Benchmark

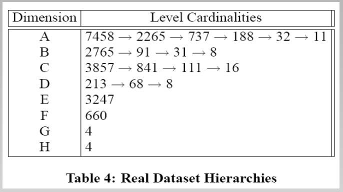
Density 1 (1.3M)		
Dwarf (Thinkpad):	18 s	57 MB
Density 5 (65M)		
Oracle's best benchmark (4 CPU, RAID)	4.5 hrs,	30.0+ GB
Dwarf (Single CPU Pentium 4)	65 min	2.4 GB
Density 40 (496M)		
Dwarf: (Single CPU Pentium 4)	10.3 hrs	8.2 GB

Never done before

NOTE: The Fact table that is intrinsically fused in the Dwarf is bigger than the Dwarf itself: Fact table is 32GB in ASCII or 11.8GB in Binary.

Real Data Set

- **Fact Table: 13,449,327**
  - **Dimensions:** ***8***
  - **Views: 11,200**
  - **Creation time: 100 min**



Dimension	Level Cardinalities
A	7458 → 2265 → 737 → 188 → 32 → 11
B	2765 → 91 → 31 → 8
C	3857 → 841 → 111 → 16
D	213 → 68 → 8
E	3247
F	660
G	4
H	4

Table 4: Real Dataset Hierarchies

- **Challenged by a leading OLAP vendor**
  - **Took 48 hrs for a "wizard" to decide what to materialize**
  - **Several more hrs to create and index summary tables**
  - **Huge storage**

- **Dwarf Datacube**
  - **Creation time: 100 min**
  - **Size: 6.7 GB**
  - **1000 Queries*: 15.8 sec**

* Each query asks for 10 different values for 3 randomly selected dimensions (e.g. ***v1* | *v2* |… | *v10***) and "all" for a 4th dimension- 10*10*10 point query

© Nick Roussopoulos

Dream DataCube[1]

- Fact table (**5,000,000**):
  - **Dimensions:** ***10 (3x9L, 4x4L, 3x2L)***
  - **Views: 16,875,000**

- Dwarf's response
  - **Creation time: 123 min**
  - **Size: 6.3 GB**
  - **1000 Queries: 325 sec**

Never done before

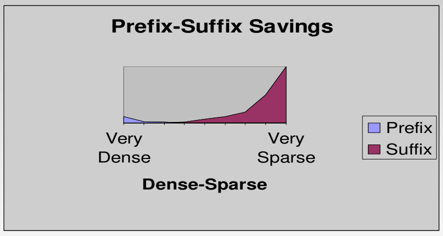
[1]Challenged by a leading OLAP Vendor:

"A full cube for this data set can never be built!"

© Nick Roussopoulos

Dwarf Data Fusion

- Two breakthrough discoveries
  - Suffix redundancy
  - Fusion of prefix and suffix redundancy



US Patent 7,133,876

- Identifies and factors out these redundancies ***before*** computing any aggregates for them

Dwarf Technology

- Complete solution
  - Extends to high dimensionality
  - Deep hierarchies
  - Queries the full cube- any dimension & level
  - Incremental updates
  - Indexing is inherent – all in one structure
  - Dwarf holds in the fact table too!

- No gotchas
  - No expensive preprocessing (just a single sort)
  - No TEMP space required for construction
  - No hidden post-construction costs
  - No information loss (100% precision)

Data Driven Tuning

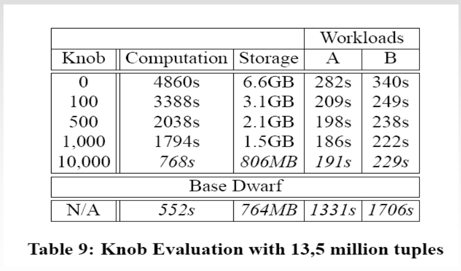
- GMin:
  - Min # of records per aggregate group to be explicitly stored
  - Reduces storage in lower levels of the hierarchies
- The Knob:
  - Max # of aggregations to be computed on the fly (not stored)
  - Reduces storage in the higher levels of the hierarchies

© Nick Roussopoulos

Data Driven Tuning Effects

- Gmin
- "The Knob"

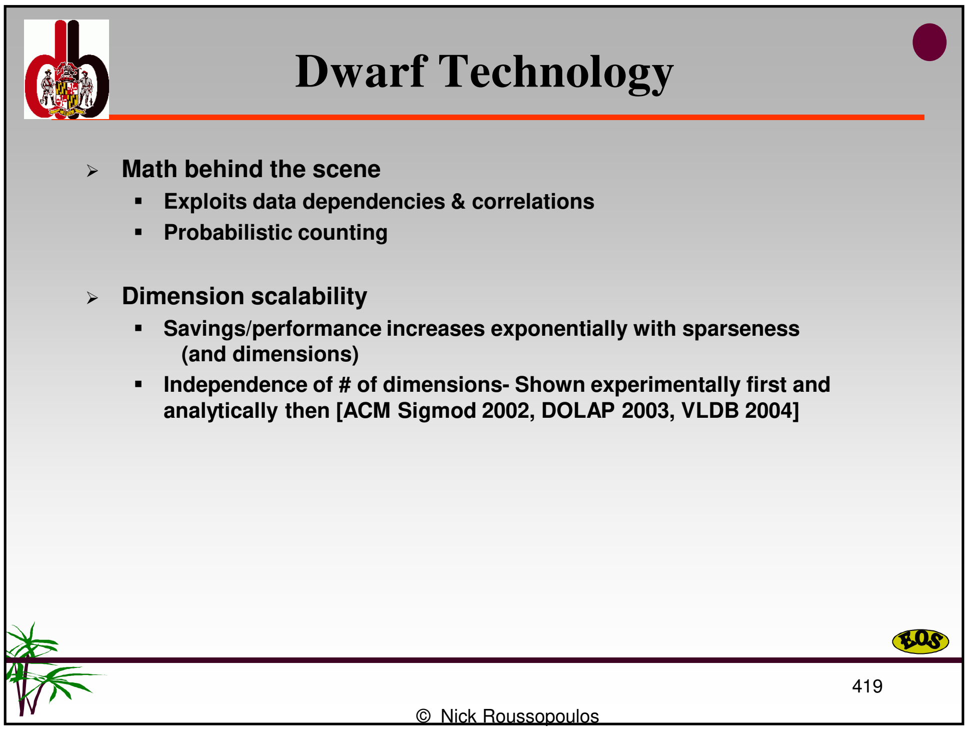
G_{min}	Space(MB)	Construction(sec)	Queries(sec)
0	490	202	154
100	400	74	110
1000	312	59	317
5000	166	29	408
20,000	151	25	476



			Workloads	
Knob	Computation	Storage	A	B
0	4860s	6.6GB	282s	340s
100	3388s	3.1GB	209s	249s
500	2038s	2.1GB	198s	238s
1,000	1794s	1.5GB	186s	222s
10,000	*768s*	*806MB*	*191s*	*229s*
Base Dwarf				
N/A	*552s*	*764MB*	*1331s*	*1706s*

Table 9: Knob Evaluation with 13,5 million tuples

© Nick Roussopoulos

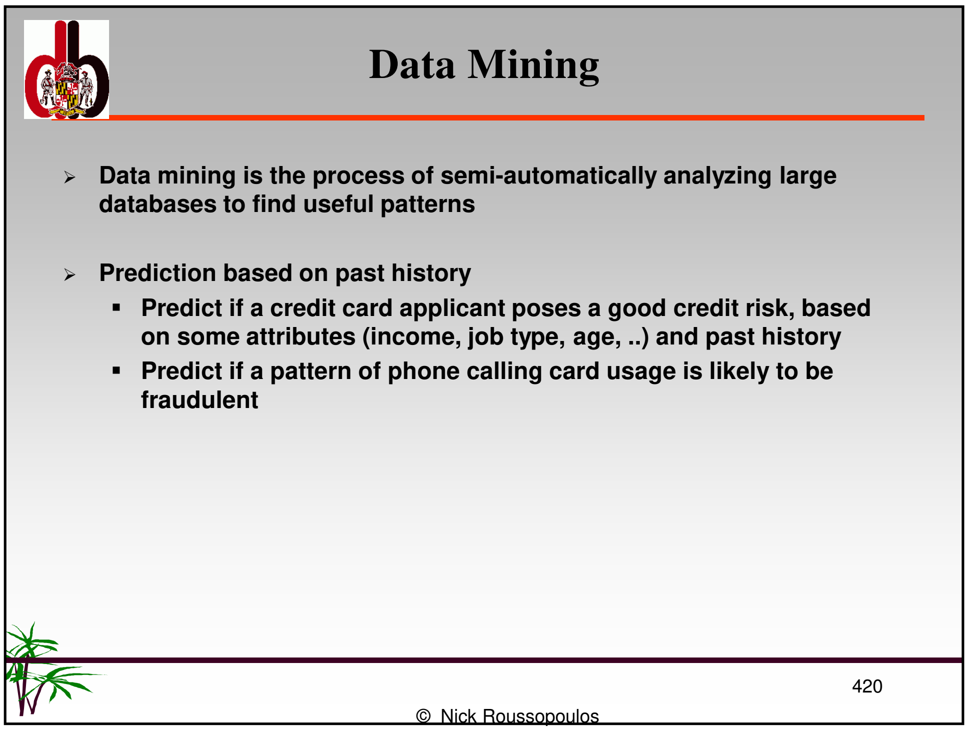


Dwarf Technology

- Math behind the scene
  - Exploits data dependencies & correlations
  - Probabilistic counting
- Dimension scalability
  - Savings/performance increases exponentially with sparseness (and dimensions)
  - Independence of # of dimensions- Shown experimentally first and analytically then [ACM Sigmod 2002, DOLAP 2003, VLDB 2004]

419

© Nick Roussopoulos



Data Mining

- Data mining is the process of semi-automatically analyzing large databases to find useful patterns
- Prediction based on past history
  - Predict if a credit card applicant poses a good credit risk, based on some attributes (income, job type, age, ..) and past history
  - Predict if a pattern of phone calling card usage is likely to be fraudulent

420

© Nick Roussopoulos

Data Mining (Cont.)

- Some examples of prediction mechanisms:
- A. Classification
  - Given a new item whose class is unknown, predict to which class it belongs
  - Find books that are often bought by "similar" customers. If a new such customer buys one such book, suggest the others too.
- B. Associations may be used as a first step in detecting causation
  - E.g., association between exposure to chemical X and cancer,
- C. Clustering
  - E.g., typhoid cases were clustered in an area surrounding a contaminated well
  - Detection of clusters remains important in detecting epidemics

421

© Nick Roussopoulos

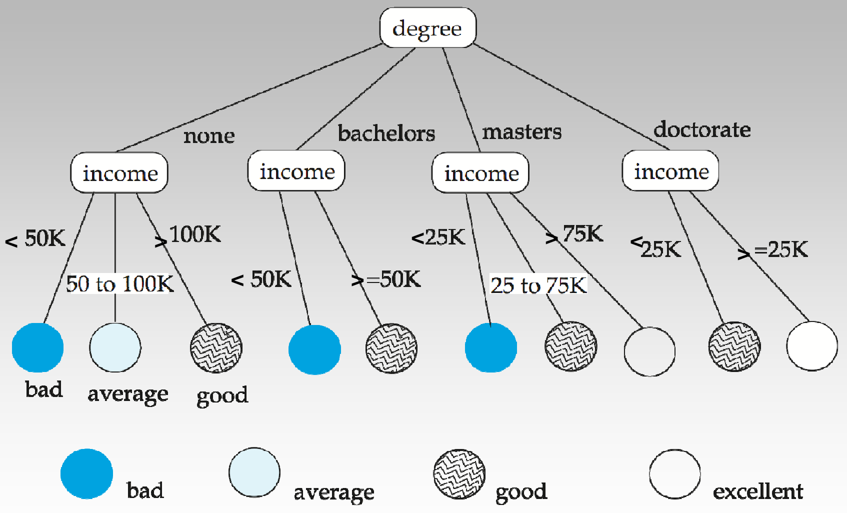
A. Classification Rules

- Classification rules help assign new objects to classes.
  - E.g., given a new automobile insurance applicant, should he or she be classified as low risk, medium risk or high risk?
- Classification rules for above example could use a variety of data, such as educational level, salary, age, etc.
  - $\forall$ person P, P.degree = masters and P.income > 75,000
    $\Rightarrow$ P.credit = excellent
  - $\forall$ person P, P.degree = bachelors and
    (P.income $\geq$ 25,000 and P.income $\leq$ 75,000)
    $\Rightarrow$ P.credit = good
- Rules are not necessarily exact: there may be some misclassifications
- Classification rules can be shown compactly as a decision tree.

© Nick Roussopoulos

422

Decision Tree



B. Association Rules

- Retail shops are often interested in associations between different items that people buy.
  - Someone who buys bread is quite likely also to buy milk
  - A person who bought the book *Database System Concepts* is quite likely also to buy the book *Operating System Concepts*.
- Associations information can be used in several ways.
  - E.g., when a customer buys a particular book, an online shop may suggest associated books.
- Association rules:

  bread ⇒ *milk*

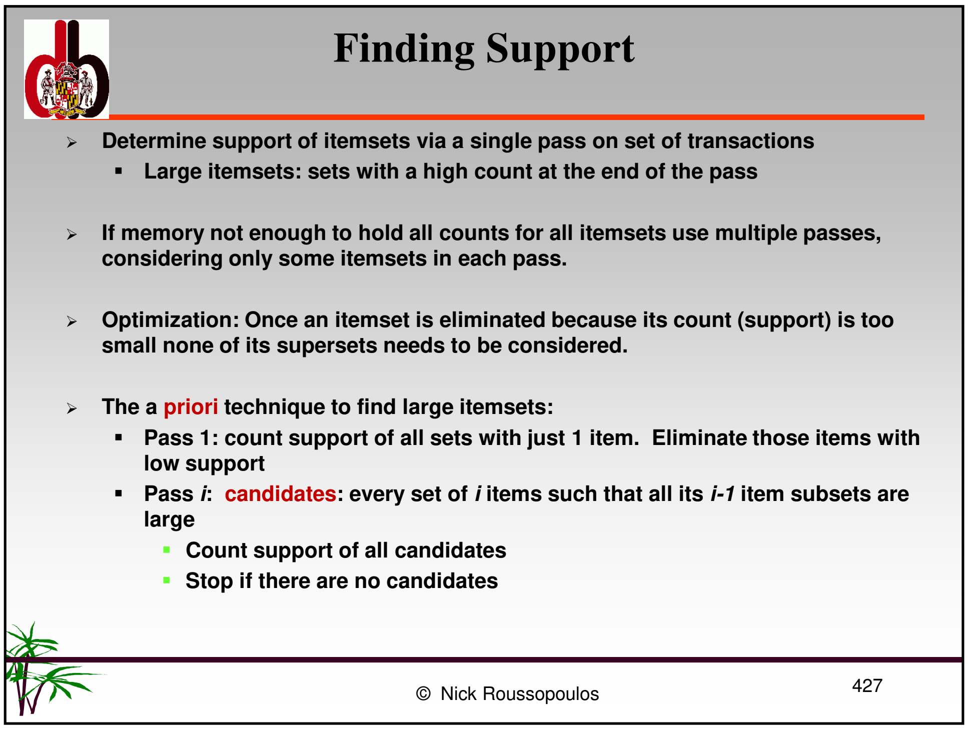
  DB-Concepts, OS-Concepts ⇒ Networks
  - Left hand side: antecedent, right hand side: consequent
  - An association rule must have an associated population: Number of instances supporting it
    - E.g., each transaction (sale) at a shop is an instance, and the set of all transactions is the population

Association Rules (Cont.)

- Rules have an associated support and confidence.
- Support is a measure of what fraction of the population satisfies both the antecedent and the consequent of the rule.
  - E.g., suppose only 0.001 percent of all purchases include milk and screwdrivers. The support for the rule is *milk* ⇒ *screwdrivers* is low.
- Confidence is a measure of how often the consequent is true when the antecedent is true.
  - E.g., the rule *bread* ⇒ *milk* has a confidence of 80 percent if 80 percent of the purchases that include bread also include milk.
- We are generally only interested in association rules with reasonably high support (e.g., support of 2% or greater)

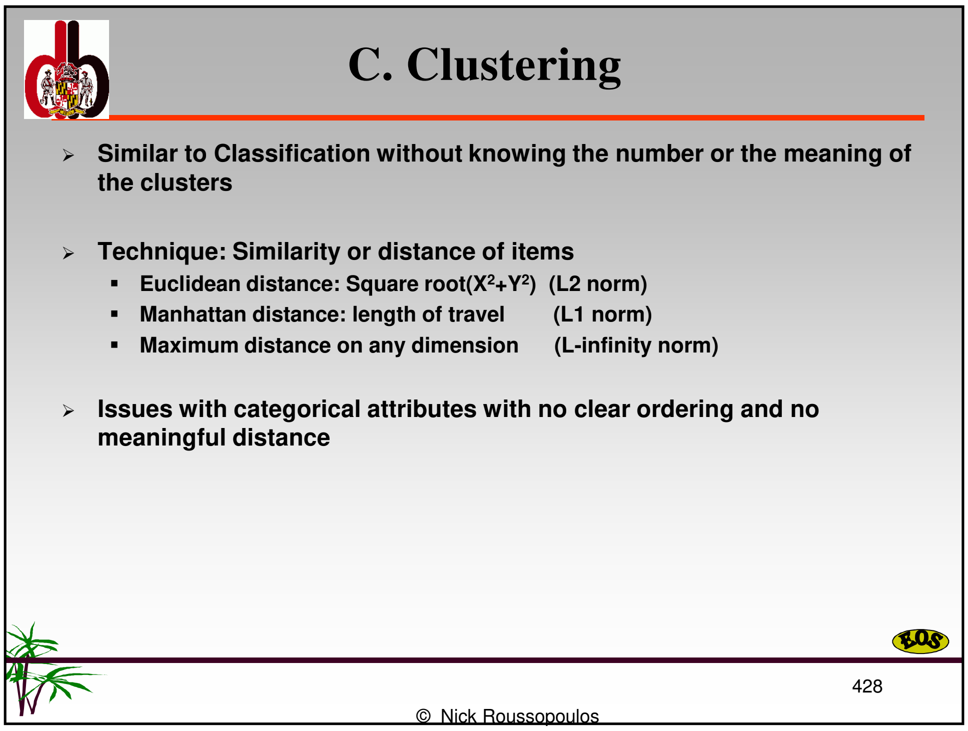
Finding Association Rules

- We are generally only interested in association rules with reasonably high support (e.g., support of 2% or greater)
- Naïve algorithm
  1. Consider all possible sets of relevant items.
  2. For each set find its support (i.e., count how many transactions purchase all items in the set).
     - Large itemsets: sets with sufficiently high support
  3. Use large itemsets to generate association rules.
     1. From itemset *A* generate the rule *A* - {*b* } ⇒*b* for each *b* ∈ *A.*
        - Support of rule = support (*A).*
        - Confidence of rule = support (*A*) / support (*A* - {*b* })



Finding Support

- Determine support of itemsets via a single pass on set of transactions
  - Large itemsets: sets with a high count at the end of the pass
- If memory not enough to hold all counts for all itemsets use multiple passes, considering only some itemsets in each pass.
- Optimization: Once an itemset is eliminated because its count (support) is too small none of its supersets needs to be considered.
- The a priori technique to find large itemsets:
  - Pass 1: count support of all sets with just 1 item. Eliminate those items with low support
  - Pass *i*: candidates: every set of *i* items such that all its *i-1* item subsets are large
    - Count support of all candidates
    - Stop if there are no candidates



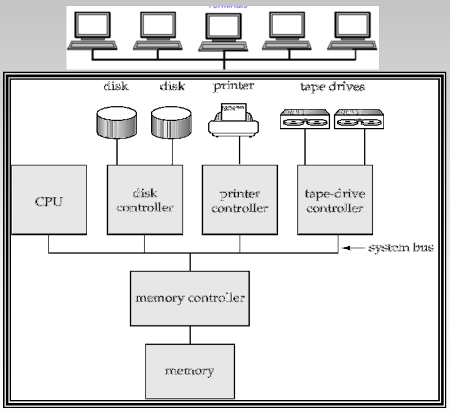
C. Clustering

- Similar to Classification without knowing the number or the meaning of the clusters
- Technique: Similarity or distance of items
  - Euclidean distance: Square root(X^2+Y^2) (L2 norm)
  - Manhattan distance: length of travel (L1 norm)
  - Maximum distance on any dimension (L-infinity norm)
- Issues with categorical attributes with no clear ordering and no meaningful distance

Chapters 17 Database System Architectures

Centralized Architecture

- Run on a single computer system
- General-purpose computer system: one to a few CPUs and a number of device controllers
- Multi-user system: several disks, lots of memory, multiple CPUs, and a multi-user OS. Serve a large number of users who are connected to the system vie terminals. Often called *server* systems.
- Single-user system (e.g., personal computer or workstation)



429

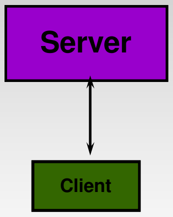
© Nick Roussopoulos

Client-Server Database Architectures

- server gets dedicated to one or a limited number of specialized tasks (file, name, page, SQL, object)
- client runs the application and the presentation layer (GUI)
  - provides device independence
  - keyboard, mouse
  - vt100, windows, bitmaps

CS Flavors

- Standard Client-Server
- Replication Servers
- Page-Server
- Enhanced Client-Server

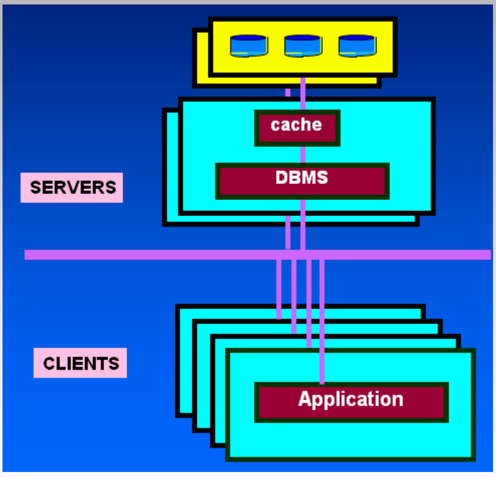


430

© Nick Roussopoulos

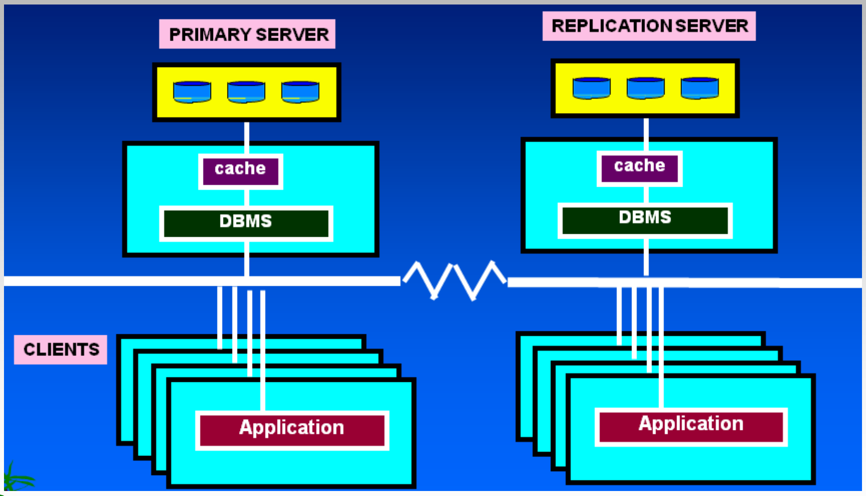
Standard Client-Server DB Architecture

- dynamic SQL interface
  - pass queries
  - return results
- overloaded server
- high network traffic



© Nick Roussopoulos

Replication Servers



© Nick Roussopoulos

Replication Servers

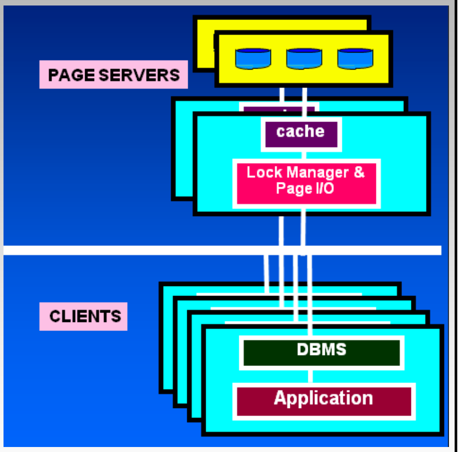
- Advantages
  - improves performance (offload the server)
  - reduces network load
  - collocates users and data
  - potential scaleable architecture
- Consistency of replicas
  - tight consistency (2-phase commit, expensive)
  - loose consistency (update with some replication latency)
- Methods for update propagation
  - complete DB transfers
  - incremental transaction log transfers
- Offered by COTS DBMSs (Ingres, Sybase, Oracle)

433

© Nick Roussopoulos

Page Servers

- query processing is mostly done on the client site
- locking and all disk I/O on the server
- advantages
  - offloads servers
- disadvantage
  - high volume transfers (unfiltered data pages)



Dead On Arrival

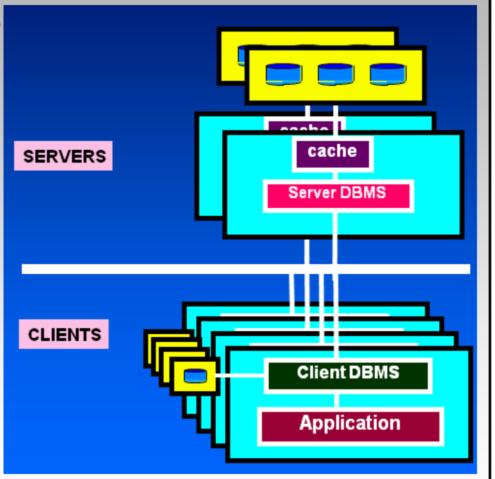
434

© Nick Roussopoulos

ADMS+- Enhanced Client-Server Architecture

- each client has a disk and a DBMS to manage the cached fragments
- client & server DBMSs co-operate and share the load
- dynamic replication of query results

Still not offered



© Nick Roussopoulos

Chapter 18: Parallel DBMSs

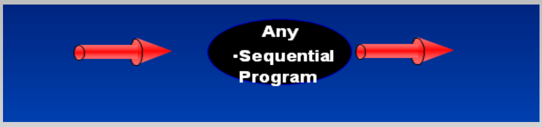
- specialized hardware - presumed dead!
- disk throughput increases linearly in the last 15 years
- processor speed increases almost exponentially
  - Joy's law MIPS= 2**(current year-1984)=2 ^21=2 TMIPS
- data volumes increase polynomially

Patterson: "stop improving CPUs- work on I/O systems"

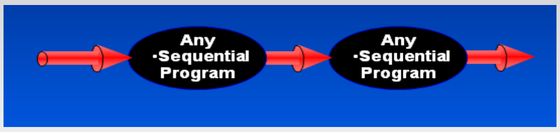
© Nick Roussopoulos

Parallelism

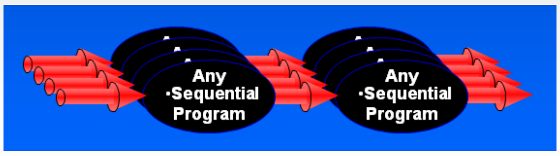
None



Pipeline



Partition
outputs split N ways
inputs merge M ways



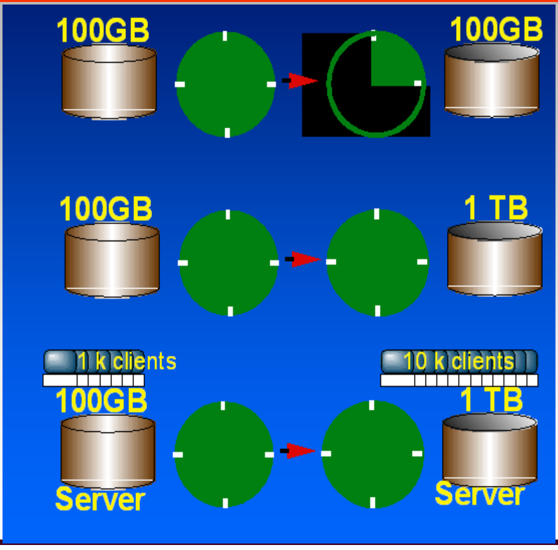
© Nick Roussopoulos

Speedup & Scaleup

Speedup:
Same Job,
More Hardware
Less time

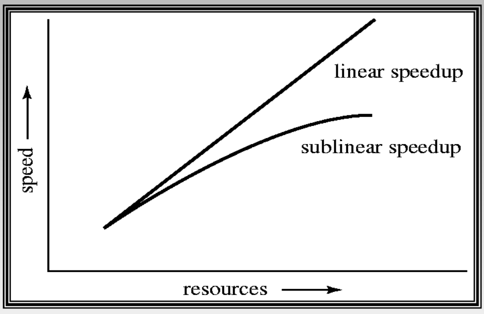
Scaleup:
Bigger Job,
More Hardware
Same time

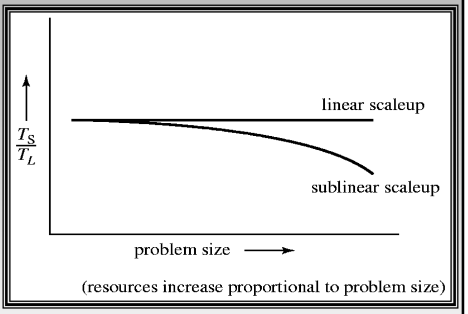
Transaction Scaleup:
more clients/servers
Same response time



© Nick Roussopoulos

Speedup & Scaleup





Speedup

Scaleup

Batch and Transaction Scaleup

- Batch scaleup:
  - **A single large job; typical of most database queries and scientific simulation.**
  - **Use an *N*-times larger computer on *N*-times larger problem.**
- Transaction scaleup**:**
  - **Numerous small queries submitted by independent users to a shared database; typical transaction processing and timesharing systems.**
  - ***N*-times as many users submitting requests (hence, *N*-times as many requests) to an *N*-times larger database, on an *N*-times larger computer.**
  - **Well-suited to parallel execution.**

Cost of parallelism

- startup - cost to spawn a process
- interference - synchronization, dependencies, locking, blocking
- network cost- protocol overhead & drivers, network load

- And MSI
  - will work in the year 20,005
    - Release 8,863,634,101b from Hell
    - Service Pack 3,005, Version 6, Oct 27, 20,005 AC, Redmond@Hell.com

Database System Hardware Architectures

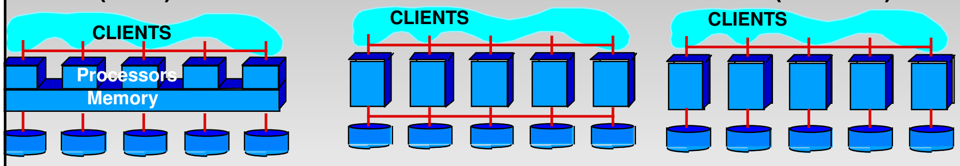
A. Parallel query processing
  - interquery (many queries in parallel)
  - intraquery (many query fragments in parallel)
  - intraoperation (many operation fragments in parallel)

B. Parallel Architectures

C. Parallel I/O

Parallel Architectures

Shared Memory (SMP) | Shared Disk | Shared Nothing (network)



Sequent, SGI, Sun | VMScluster, Sysplex | Tandem, Teradata, SP2

Winner will be a synthesis of these ideas

Distributed shared memory (DASH, Encore) blurs

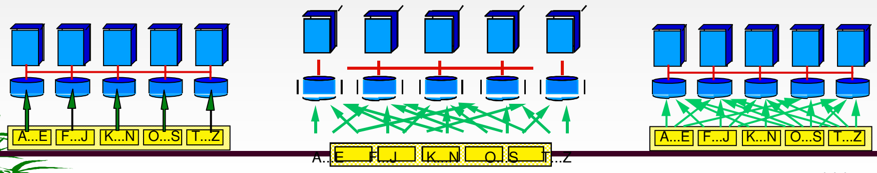
Parallel I/O

A. **Data placement**
- enhance I/O bandwidth
- load balancing
- declustering vs clustering
  - interoperation parallelism - cluster
  - intraoperation parallelism - decluster
- clustering/declustering techniques
  - fixed using a hash function
  - variable based on sizes and number of nodes (difficult, requires reorganization)
- reorganization affects compiled queries (supposed to be fast)

Split data to subset of nodes & disks

Partition within set:

Range | **Hash** | **Round Robin**



Parallel I/O

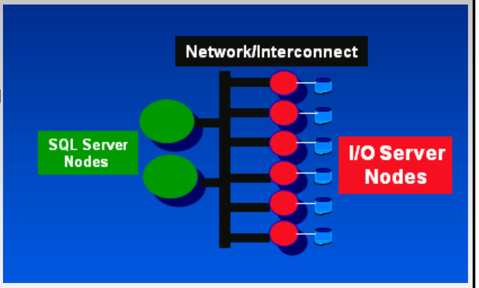
B. Data Replication (fault tolerance)
- mirrowing
- interleaved declustering
- chained declustering
- RAID
- RAID 0- page level distribution

445

© Nick Roussopoulos

Disk Farms

- nodes run asynchronously
- nodes can be configured as SQL server or I/O servers
- Benefits
  - pipeline & partition
  - parallel request streams
  - parallel data streams
- Configurable striping and declustering
  - partition, hash, round robin
- Asynchronicity
  - network & disk I/O
- Caching
  - server & client adaptive buffering
- Prefetching
  - bulk data requests
  - application specific prefetching
  - adaptive pipeline feed



446

© Nick Roussopoulos

Chapter 19: Distributed Databases

- Distributed database system
  - Multiple sites that share no components
  - Each site runs independently
- Homogeneous and Heterogeneous Databases
  - Identical software run on multiple sites as a single system
  - Different software, schema, data
- Distributed Data Storage
  - Data fragmentation
  - Data replication
  - Data transparency
- Distributed Transactions
  - May run in multiple sites
  - Each site has a transaction coordinator
- Commit Protocols
  - 2PC
  - 3PC
- Concurrency Control in Distributed Databases
  - Single lock manager
  - Distributed lock manager
  - Distributed deadlock avoidance
- Distributed Query Processing, Etc.

Data Replication

- A relation or fragment of a relation is replicated if it is stored redundantly in two or more sites.
- Full replication of a relation is the case where the relation is stored at all sites.
- Fully redundant databases are those in which every site contains a copy of the entire database.

- Advantages of Replication
  - Availability: failure of site containing relation *r* does not result in unavailability of *r* is replicas exist.
  - Parallelism: queries on *r* may be processed by several nodes in parallel.
  - Reduced data transfer: relation *r* is available locally at each site containing a replica of *r*.
- Disadvantages of Replication
  - Increased cost of updates: each replica of relation *r* must be updated.
  - Increased complexity of concurrency control: concurrent updates to distinct replicas may lead to inconsistent data unless special concurrency control mechanisms are implemented.
    - One solution: choose one copy as primary copy and apply concurrency control operations on primary copy

Data Fragmentation

- Division of relation r into fragments $r_1, r_2, \ldots, r_n$ which contain sufficient information to reconstruct relation r.
- Horizontal fragmentation: each tuple of *r* is assigned to one or more fragments
- Vertical fragmentation: the schema for relation *r* is split into several smaller schemas
  - All schemas must contain a common candidate key (or superkey) to ensure lossless join property.
  - A special attribute, the tuple-id attribute may be added to each schema to serve as a candidate key.

Horizontal Fragmentation of *account* Relation

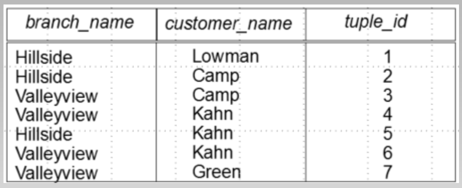
branch_name	*account_number*	*balance*
Hillside	A-305	500
Hillside	A-226	336
Hillside	A-155	62

$$account_1 = \sigma_{branch\_name = \text{“Hillside”}}(account)$$

branch_name	*account_number*	*balance*
Valleyview	A-177	205
Valleyview	A-402	10000
Valleyview	A-408	1123
Valleyview	A-639	750

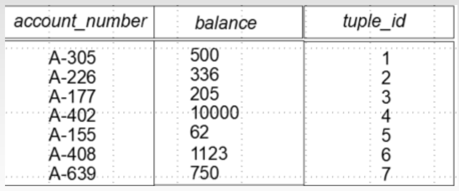
$$account_2 = \sigma_{branch\_name = \text{“Valleyview”}}(account)$$

Vertical Fragmentation of *employee_info* Relation



branch_name	*customer_name*	*tuple_id*
Hillside	Lowman	1
Hillside	Camp	2
Valleyview	Camp	3
Valleyview	Kahn	4
Hillside	Kahn	5
Valleyview	Kahn	6
Valleyview	Green	7

$$deposit_1 = \Pi_{branch\_name,\ customer\_name,\ tuple\_id}(employee\_info)$$

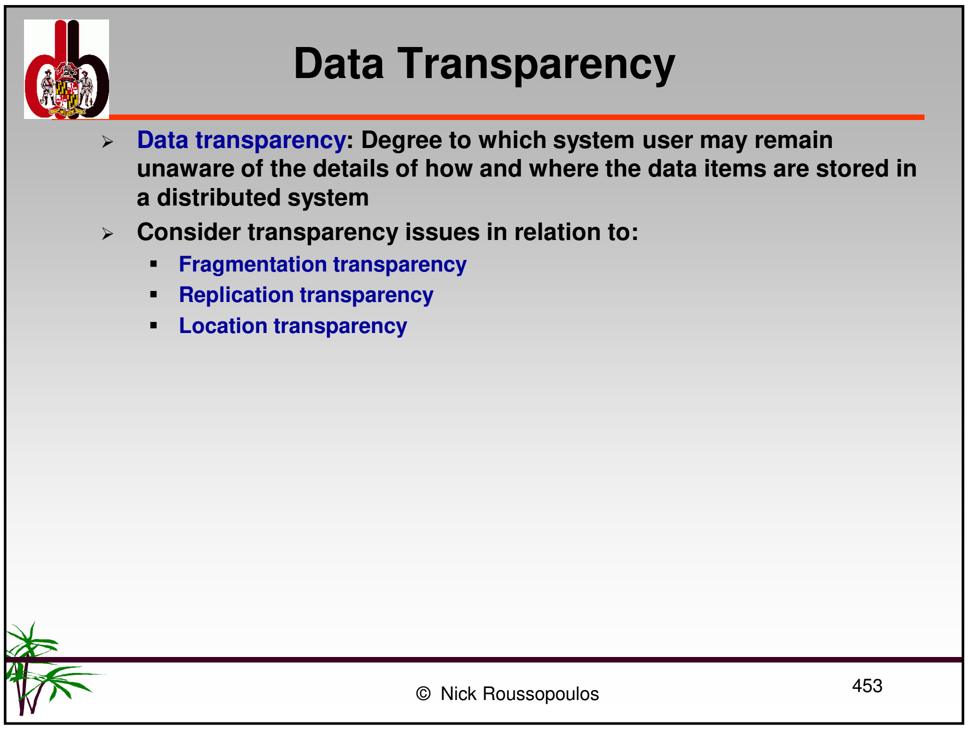


account_number	*balance*	*tuple_id*
A-305	500	1
A-226	336	2
A-177	205	3
A-402	10000	4
A-155	62	5
A-408	1123	6
A-639	750	7

$$deposit_2 = \Pi_{account\_number,\ balance,\ tuple\_id}(employee\_info)$$

Advantages of Fragmentation

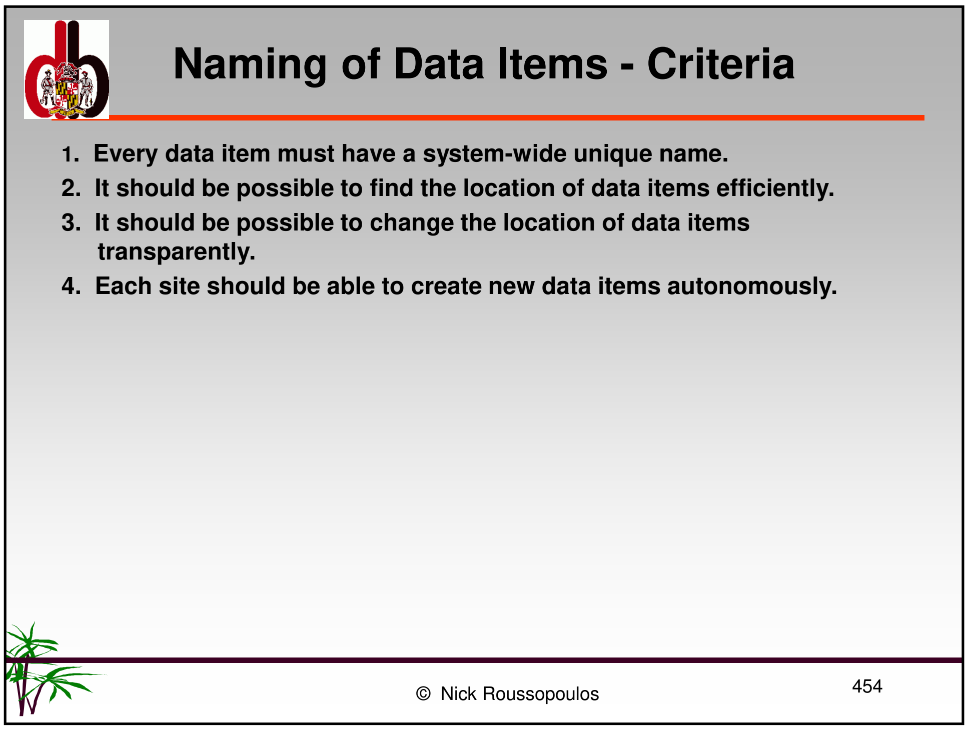
- **Horizontal:**
  - **allows parallel processing on fragments of a relation**
  - **allows a relation to be split so that tuples are located where they are most frequently accessed**
- **Vertical:**
  - **allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed**
  - **tuple-id attribute allows efficient joining of vertical fragments**
  - **allows parallel processing on a relation**
- **Vertical and horizontal fragmentation can be mixed.**
  - **Fragments may be successively fragmented to an arbitrary depth.**



Data Transparency

- **Data transparency: Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system**
- **Consider transparency issues in relation to:**
  - **Fragmentation transparency**
  - **Replication transparency**
  - **Location transparency**

© Nick Roussopoulos 453



Naming of Data Items - Criteria

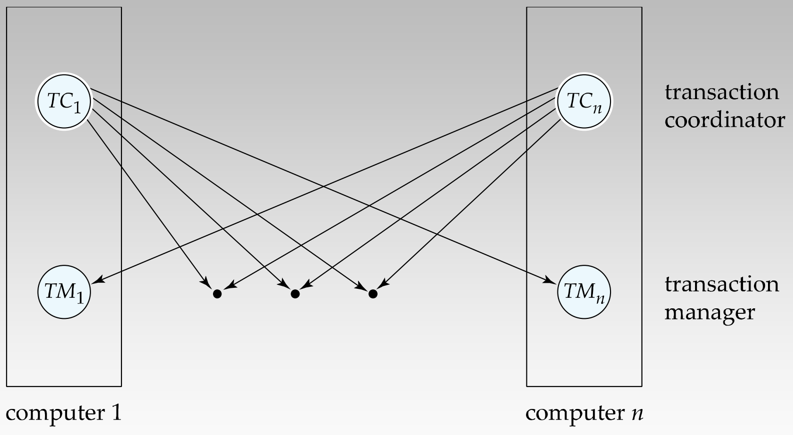
1. **Every data item must have a system-wide unique name.**
2. **It should be possible to find the location of data items efficiently.**
3. **It should be possible to change the location of data items transparently.**
4. **Each site should be able to create new data items autonomously.**

© Nick Roussopoulos 454

Distributed Transactions

- Transaction may access data at several sites.
- Each site has a local transaction manager responsible for:
  - Maintaining a log for recovery purposes
  - Participating in coordinating the concurrent execution of the transactions executing at that site.
- Each site has a transaction coordinator, which is responsible for:
  - Starting the execution of transactions that originate at the site.
  - Distributing subtransactions at appropriate sites for execution.
  - Coordinating the termination of each transaction that originates at the site, which may result in the transaction being committed at all sites or aborted at all sites.

Transaction System Architecture



System Failure Modes

- Failures unique to distributed systems:
  - Failure of a site.
  - Loss of massages
    - Handled by network transmission control protocols such as TCP-IP
  - Failure of a communication link
    - Handled by network protocols, by routing messages via alternative links
  - Network partition
    - A network is said to be partitioned when it has been split into two or more subsystems that lack any connection between them
      - Note: a subsystem may consist of a single node
- Network partitioning and site failures are generally indistinguishable.

Commit Protocols

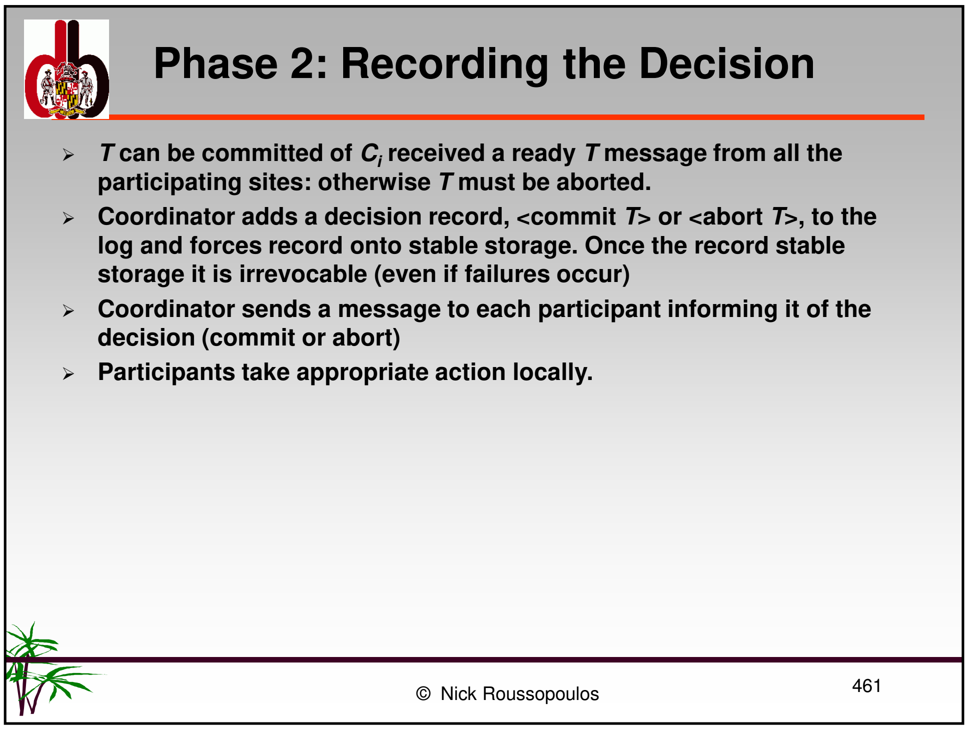
- Commit protocols are used to ensure atomicity across sites
  - a transaction which executes at multiple sites must either be committed at all the sites, or aborted at all the sites.
  - not acceptable to have a transaction committed at one site and aborted at another
- The *two-phase commit* (2PC) protocol is widely used
- The *three-phase commit* (3PC) protocol is more complicated and more expensive, but avoids some drawbacks of two-phase commit protocol. This protocol is not used in practice.

Two Phase Commit Protocol (2PC)

- Assumes fail-stop model – failed sites simply stop working, and do not cause any other harm, such as sending incorrect messages to other sites.
- Execution of the protocol is initiated by the coordinator after the last step of the transaction has been reached.
- The protocol involves all the local sites at which the transaction executed
- Let *T* be a transaction initiated at site S_i, and let the transaction coordinator at S_i be C_i

Phase 1: Obtaining a Decision

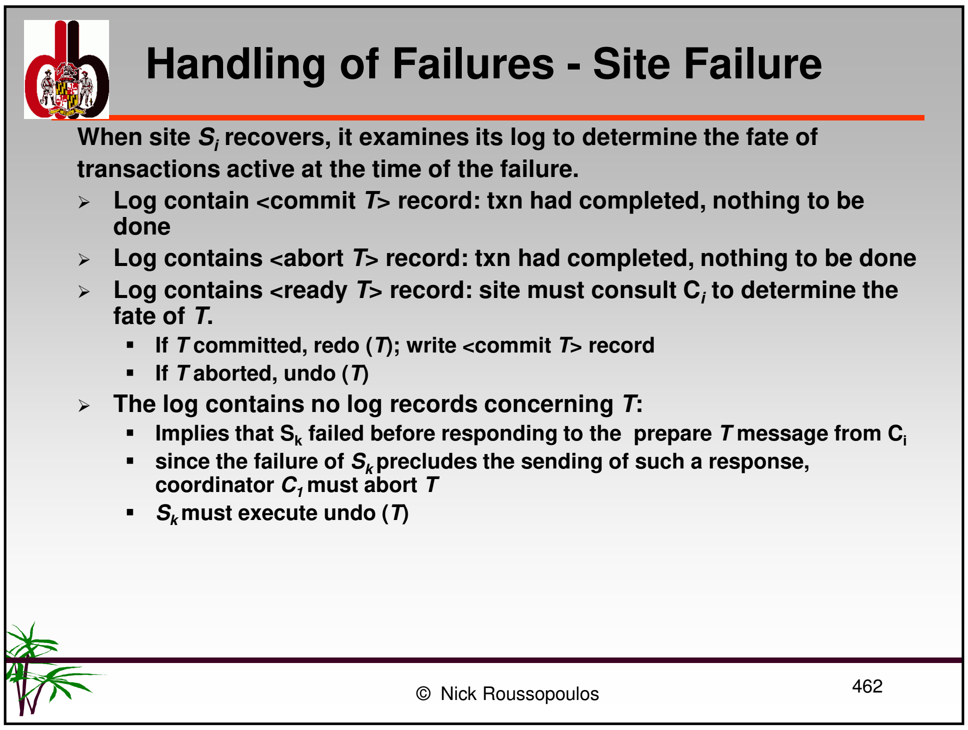
- Coordinator asks all participants to ***prepare*** to commit transaction T_i.
  - C_i adds the records <prepare *T*> to the log and forces log to stable storage
  - sends prepare *T* messages to all sites at which *T* executed
- Upon receiving message, transaction manager at site determines if it can commit the transaction
  - if not, add a record <no *T*> to the log and send abort *T* message to C_i
  - if the transaction can be committed, then:
  - add the record <ready *T*> to the log
  - force *all records* for *T* to stable storage
  - send ready *T* message to C_i



Phase 2: Recording the Decision

- *T* can be committed of C_i received a ready *T* message from all the participating sites: otherwise *T* must be aborted.
- Coordinator adds a decision record, <commit *T*> or <abort *T*>, to the log and forces record onto stable storage. Once the record stable storage it is irrevocable (even if failures occur)
- Coordinator sends a message to each participant informing it of the decision (commit or abort)
- Participants take appropriate action locally.

© Nick Roussopoulos 461

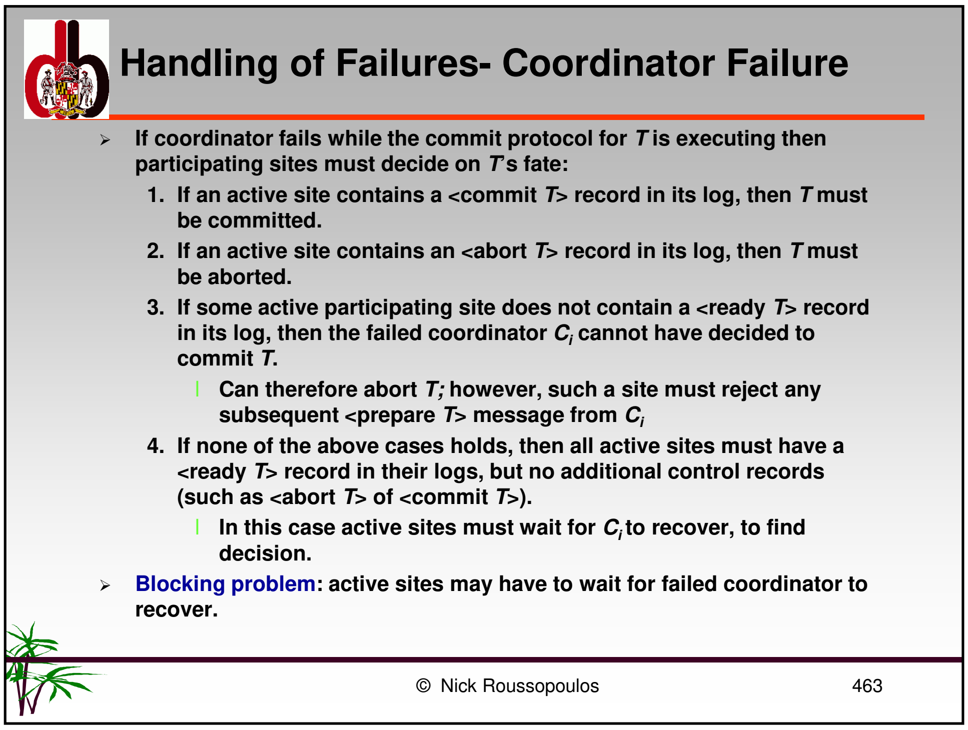


Handling of Failures - Site Failure

When site S_i recovers, it examines its log to determine the fate of transactions active at the time of the failure.

- Log contain <commit *T*> record: txn had completed, nothing to be done
- Log contains <abort *T*> record: txn had completed, nothing to be done
- Log contains <ready *T*> record: site must consult C_i to determine the fate of *T*.
  - If *T* committed, redo (*T*); write <commit *T*> record
  - If *T* aborted, undo (*T*)
- The log contains no log records concerning *T*:
  - Implies that S_k failed before responding to the prepare *T* message from C_i
  - since the failure of S_k precludes the sending of such a response, coordinator C_1 must abort *T*
  - S_k must execute undo (*T*)

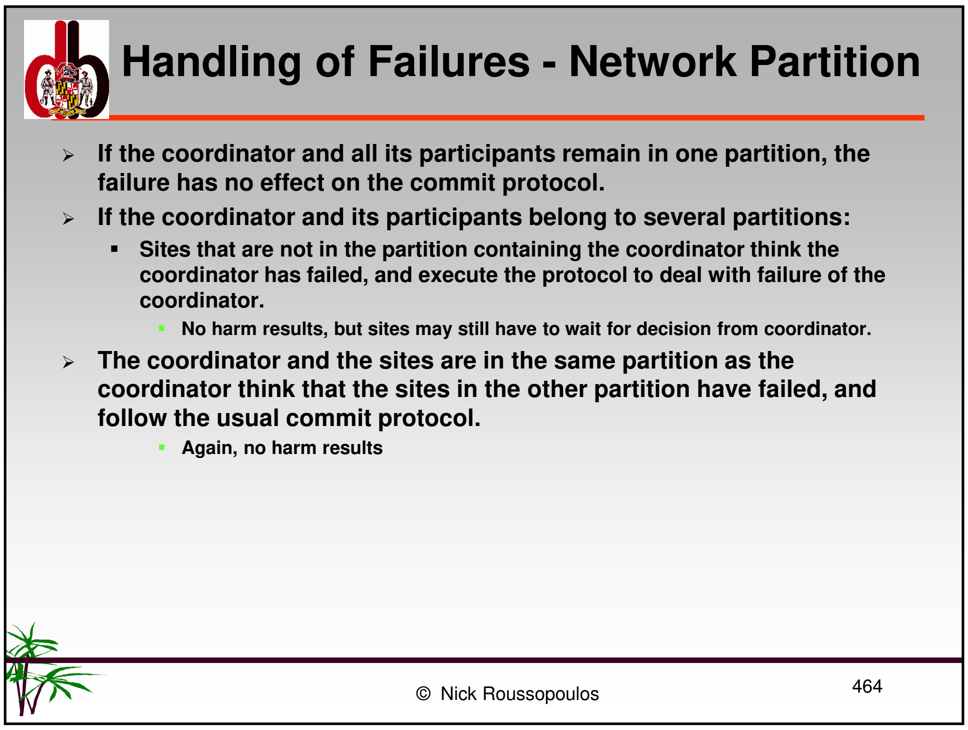
© Nick Roussopoulos 462



Handling of Failures- Coordinator Failure

- If coordinator fails while the commit protocol for *T* is executing then participating sites must decide on *T*'s fate:
  1. If an active site contains a <commit *T*> record in its log, then *T* must be committed.
  2. If an active site contains an <abort *T*> record in its log, then *T* must be aborted.
  3. If some active participating site does not contain a <ready *T*> record in its log, then the failed coordinator C_i cannot have decided to commit *T*.
     - Can therefore abort *T;* however, such a site must reject any subsequent <prepare *T*> message from C_i
  4. If none of the above cases holds, then all active sites must have a <ready *T*> record in their logs, but no additional control records (such as <abort *T*> of <commit *T*>).
     - In this case active sites must wait for C_i to recover, to find decision.
- **Blocking problem**: active sites may have to wait for failed coordinator to recover.

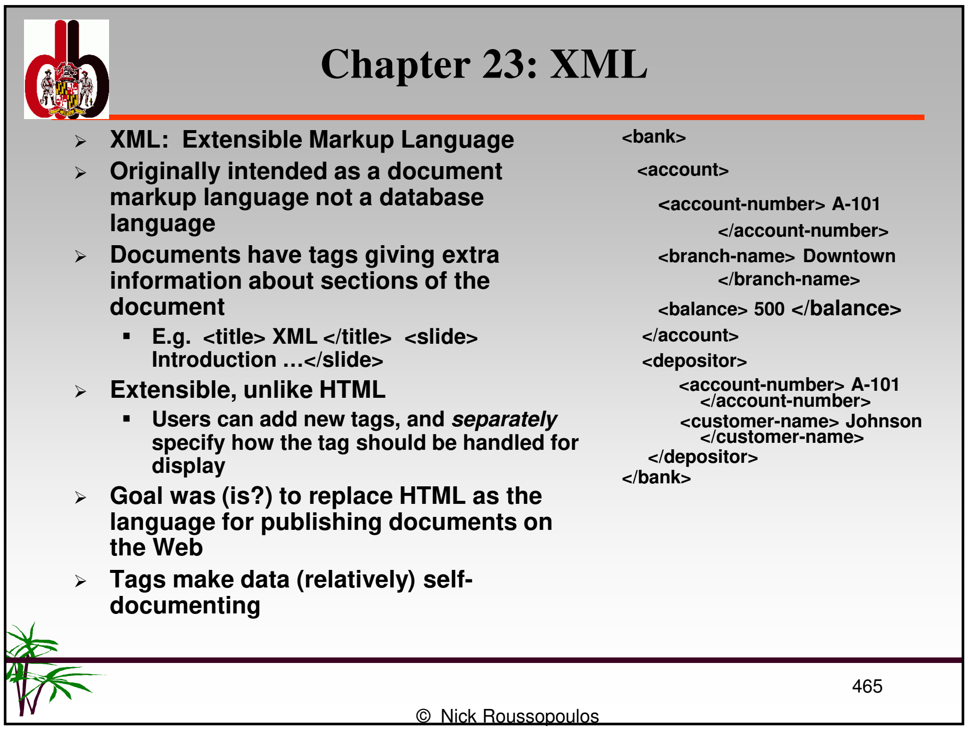
© Nick Roussopoulos 463



Handling of Failures - Network Partition

- If the coordinator and all its participants remain in one partition, the failure has no effect on the commit protocol.
- If the coordinator and its participants belong to several partitions:
  - Sites that are not in the partition containing the coordinator think the coordinator has failed, and execute the protocol to deal with failure of the coordinator.
    - No harm results, but sites may still have to wait for decision from coordinator.
- The coordinator and the sites are in the same partition as the coordinator think that the sites in the other partition have failed, and follow the usual commit protocol.
    - Again, no harm results

© Nick Roussopoulos 464



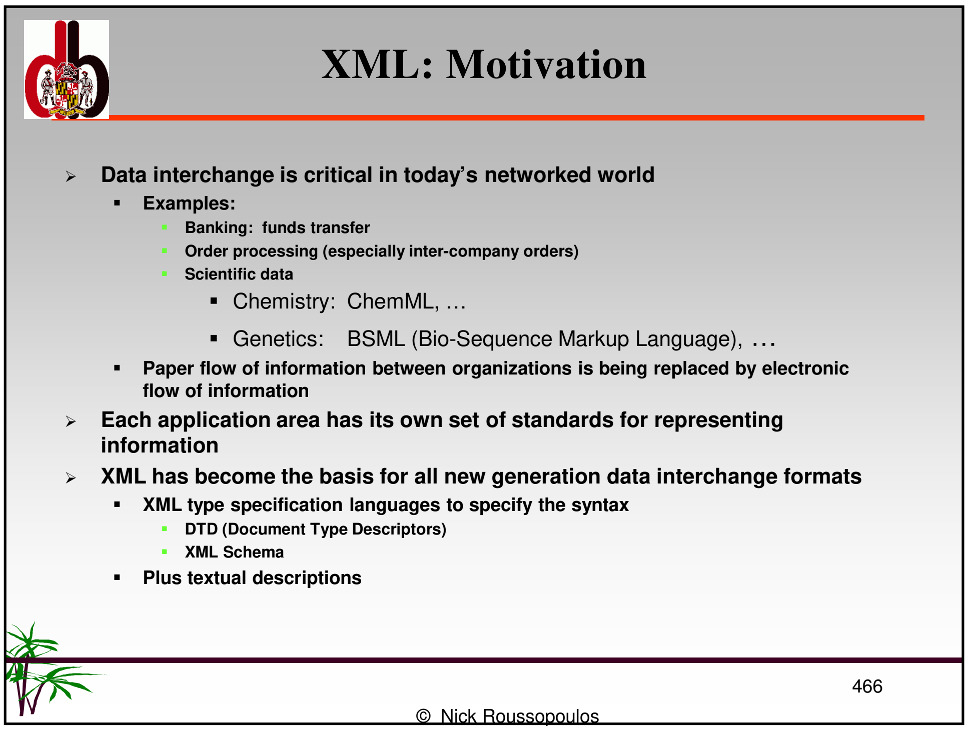
Chapter 23: XML

- XML: Extensible Markup Language
- Originally intended as a document markup language not a database language
- Documents have tags giving extra information about sections of the document
  - E.g. <title> XML </title> <slide> Introduction …</slide>
- Extensible, unlike HTML
  - Users can add new tags, and *separately* specify how the tag should be handled for display
- Goal was (is?) to replace HTML as the language for publishing documents on the Web
- Tags make data (relatively) self-documenting

```
<bank>
  <account>
    <account-number> A-101
         </account-number>
    <branch-name> Downtown
         </branch-name>
    <balance> 500 </balance>
  </account>
  <depositor>
     <account-number> A-101
       </account-number>
     <customer-name> Johnson
       </customer-name>
   </depositor>
</bank>
```

465

© Nick Roussopoulos

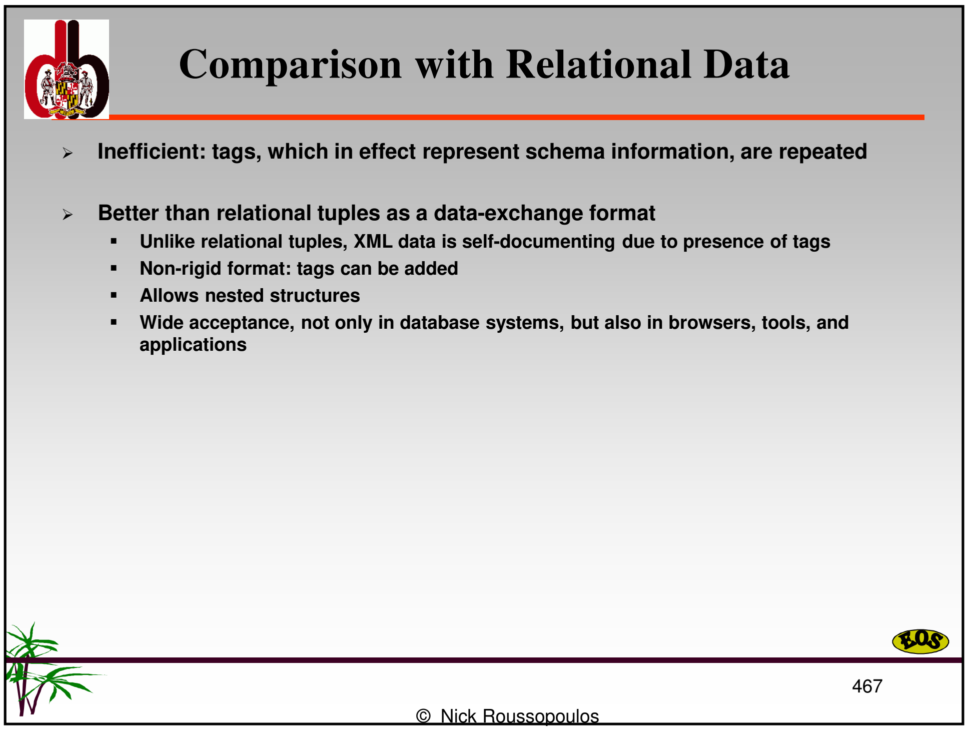


XML: Motivation

- Data interchange is critical in today's networked world
  - Examples:
    - Banking: funds transfer
    - Order processing (especially inter-company orders)
    - Scientific data
      - Chemistry: ChemML, …
      - Genetics: BSML (Bio-Sequence Markup Language), …
  - Paper flow of information between organizations is being replaced by electronic flow of information
- Each application area has its own set of standards for representing information
- XML has become the basis for all new generation data interchange formats
  - XML type specification languages to specify the syntax
    - DTD (Document Type Descriptors)
    - XML Schema
  - Plus textual descriptions

466

© Nick Roussopoulos

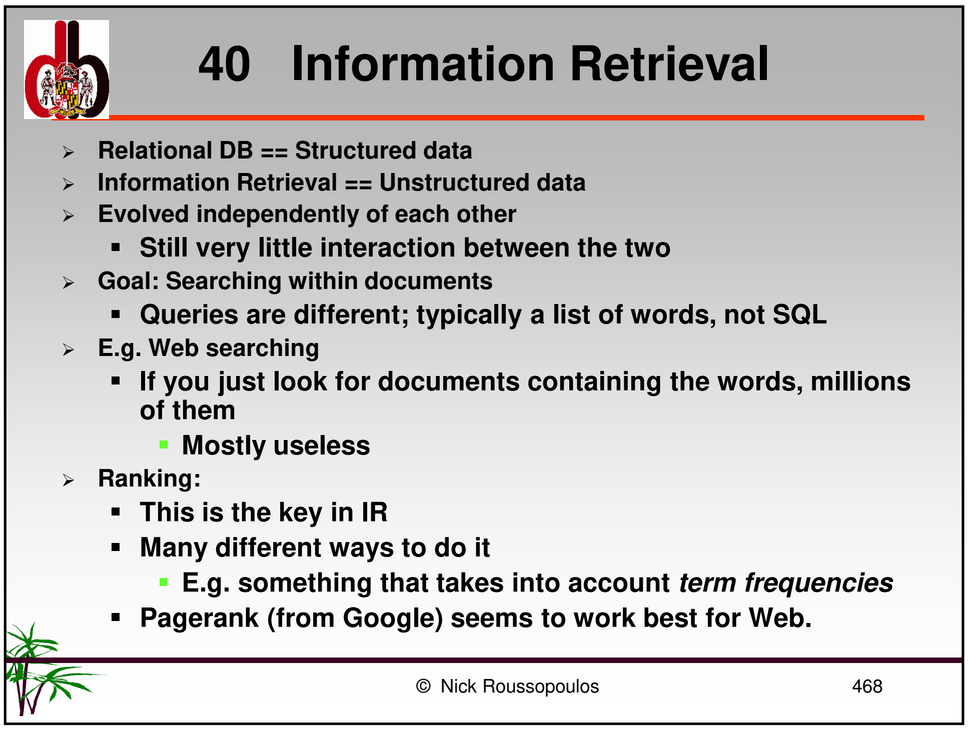


Comparison with Relational Data

- Inefficient: tags, which in effect represent schema information, are repeated
- Better than relational tuples as a data-exchange format
  - Unlike relational tuples, XML data is self-documenting due to presence of tags
  - Non-rigid format: tags can be added
  - Allows nested structures
  - Wide acceptance, not only in database systems, but also in browsers, tools, and applications

467

© Nick Roussopoulos



40 Information Retrieval

- Relational DB == Structured data
- Information Retrieval == Unstructured data
- Evolved independently of each other
  - Still very little interaction between the two
- Goal: Searching within documents
  - Queries are different; typically a list of words, not SQL
- E.g. Web searching
  - If you just look for documents containing the words, millions of them
    - Mostly useless
- Ranking:
  - This is the key in IR
  - Many different ways to do it
    - E.g. something that takes into account *term frequencies*
  - Pagerank (from Google) seems to work best for Web.

© Nick Roussopoulos 468

Relevance Ranking Using Terms

- **TF-IDF (Term frequency/Inverse Document frequency) ranking:**
  - **Let *n*(*d*) = number of terms in the document *d***
  - ***n*(*d, t*) = number of occurrences of term *t* in the document *d*.**
  - **Relevance of a document *d* to a *term t***

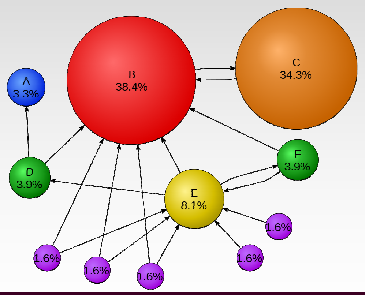
$$TF(d, t) = \log\left(1 + \frac{n(d, t)}{n(d)}\right)$$

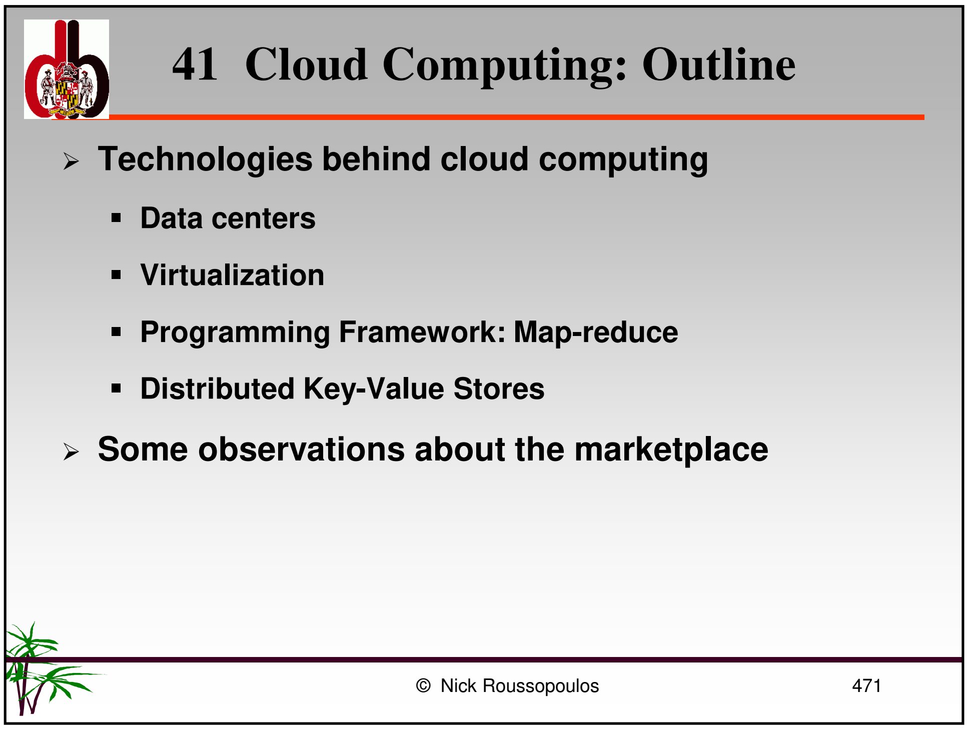
  - **The log factor is to avoid excessive weight to frequent terms**
  - **Relevance of document to *query Q***

$$r(d, Q) = \sum_{t \in Q} \frac{TF(d, t)}{n(t)}$$

PageRank

- **The probability that a random surfer (who follows links randomly) will end up at a particular page**
  - **Intuitively: Higher the probability, the more important the page**
- **Surfer model:**
  - **Choose a random page to visit with probability "alpha"**
  - **If the number of outgoing edges = n, then visit one of those pages with probability (1 – alpha)/n**

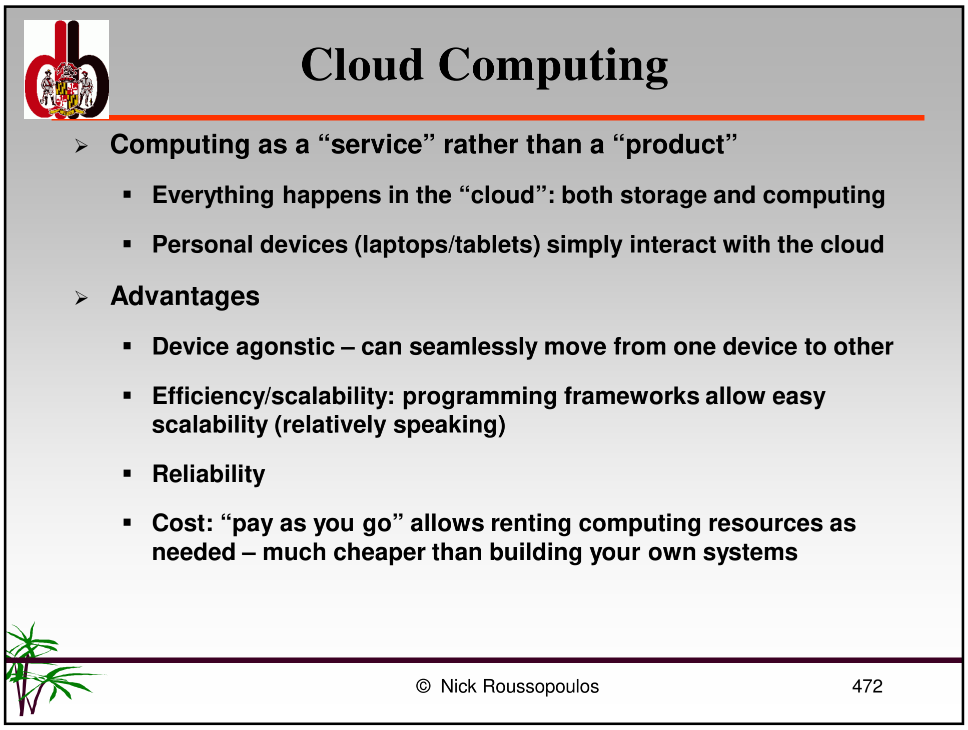




41 Cloud Computing: Outline

- Technologies behind cloud computing
  - Data centers
  - Virtualization
  - Programming Framework: Map-reduce
  - Distributed Key-Value Stores
- Some observations about the marketplace

© Nick Roussopoulos 471



Cloud Computing

- Computing as a “service” rather than a “product”
  - Everything happens in the “cloud”: both storage and computing
  - Personal devices (laptops/tablets) simply interact with the cloud
- Advantages
  - Device agonstic – can seamlessly move from one device to other
  - Efficiency/scalability: programming frameworks allow easy scalability (relatively speaking)
  - Reliability
  - Cost: “pay as you go” allows renting computing resources as needed – much cheaper than building your own systems

© Nick Roussopoulos 472

Cloud Computing

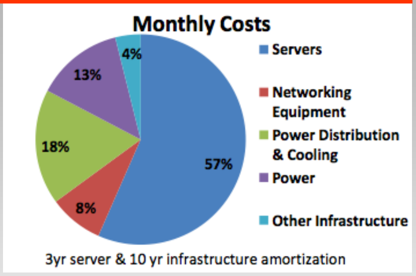
- Basic ideas have been around for a long time (going back to 1960's)
  - Mainframes + thin clients (more by necessity)
  - Grid computing a few year ago
  - Peer-to-peer
  - Client-server models
  - ...
- But it finally works as we wished for…
  - Why now?... A convergence of several key pieces over the last few years
  - Does it really? … Still many growing pains

Data Centers

- The key infrastructure piece that enables CC
- Everyone is building them
- Huge amount of work on deciding how to build/design them

Data Centers

- Amazon data centers: Some recent data
  - 8 MW data center can include about 46,000 servers
  - Costs about $88 million to build (just the facility)
  - Power a pretty large portion, but server costs still dominate



"Every day, Amazon Web Services adds enough new capacity to support all of Amazon.com's global infrastructure through the company's first 5 years, when it was a $2.76B annual revenue enterprise"

source: James Hamilton Presentation

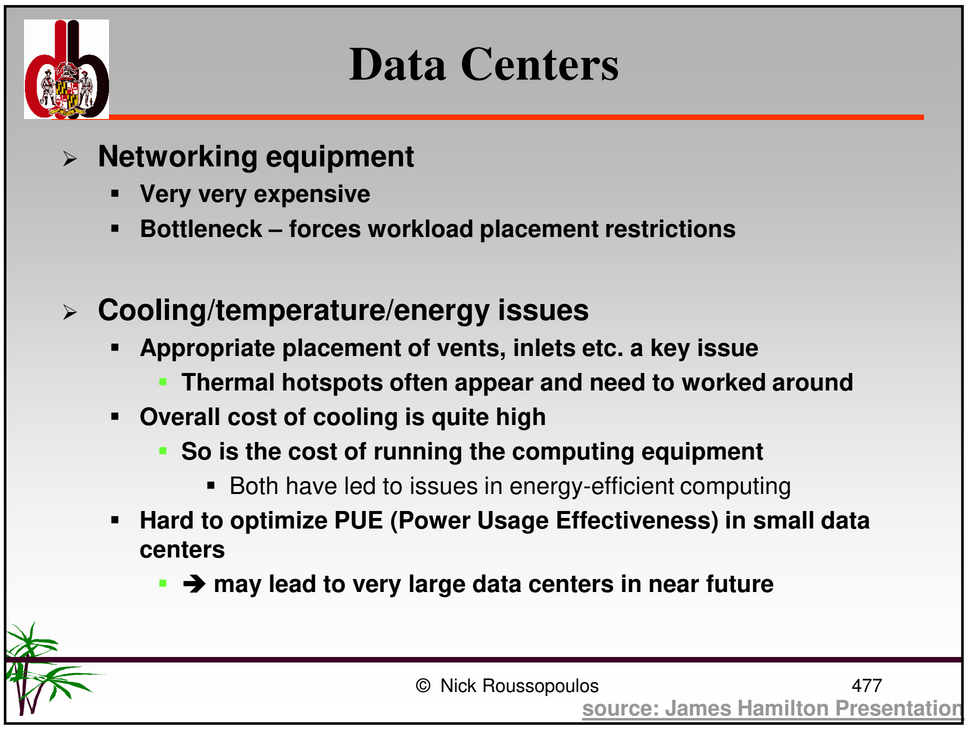
Data Centers

- Power distribution
  - Almost 11% lost in distribution – starts mattering when the total power consumption is in millions
- Modular and pre-fab designs
  - Fast and economic deployments, built in a factory

Microsoft ITPA

Amazon Perdix

source: James Hamilton Present

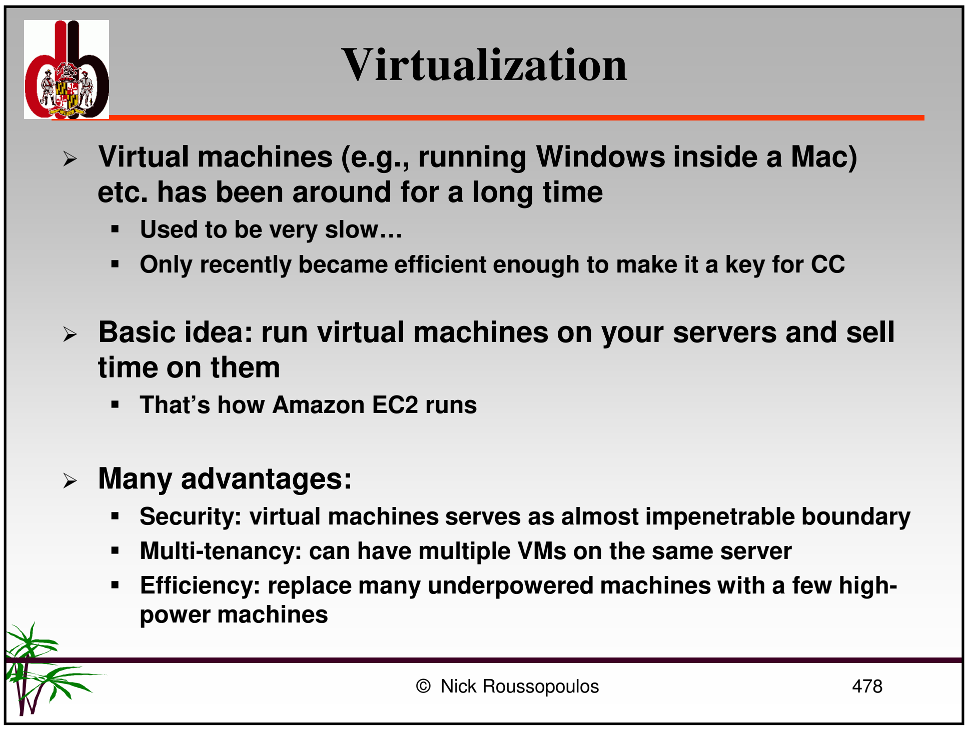


Data Centers

- Networking equipment
  - Very very expensive
  - Bottleneck – forces workload placement restrictions
- Cooling/temperature/energy issues
  - Appropriate placement of vents, inlets etc. a key issue
    - Thermal hotspots often appear and need to worked around
  - Overall cost of cooling is quite high
    - So is the cost of running the computing equipment
      - Both have led to issues in energy-efficient computing
  - Hard to optimize PUE (Power Usage Effectiveness) in small data centers
    - ➔ may lead to very large data centers in near future

© Nick Roussopoulos 477

source: James Hamilton Presentation



Virtualization

- Virtual machines (e.g., running Windows inside a Mac) etc. has been around for a long time
  - Used to be very slow…
  - Only recently became efficient enough to make it a key for CC
- Basic idea: run virtual machines on your servers and sell time on them
  - That's how Amazon EC2 runs
- Many advantages:
  - Security: virtual machines serves as almost impenetrable boundary
  - Multi-tenancy: can have multiple VMs on the same server
  - Efficiency: replace many underpowered machines with a few high-power machines

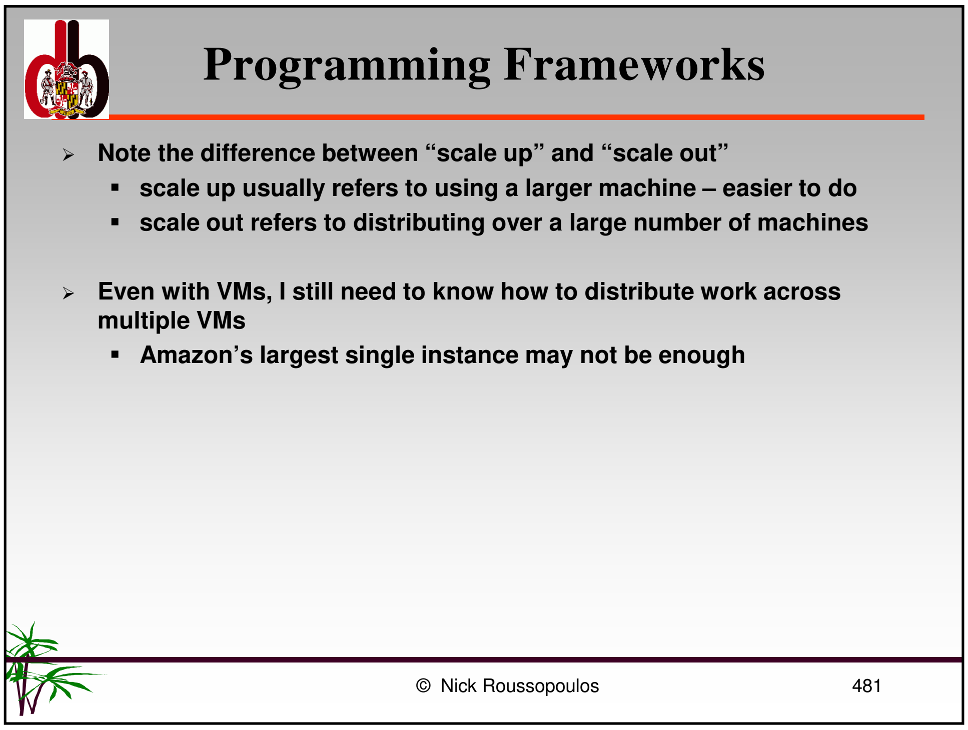
© Nick Roussopoulos 478

Virtualization

- Consumer VM products include VMWare, Parallels (for Mac) etc…
- Amazon uses "Xen" running on Redhat machines (may be old information)
  - They support both Windows and Linux Virtual Machines
- Some tricky things to keep in mind:
  - Harder to reason about performance (if you care)
  - Identical VMs may deliver somewhat different performance
- Much continuing work on the virtualization technology itself

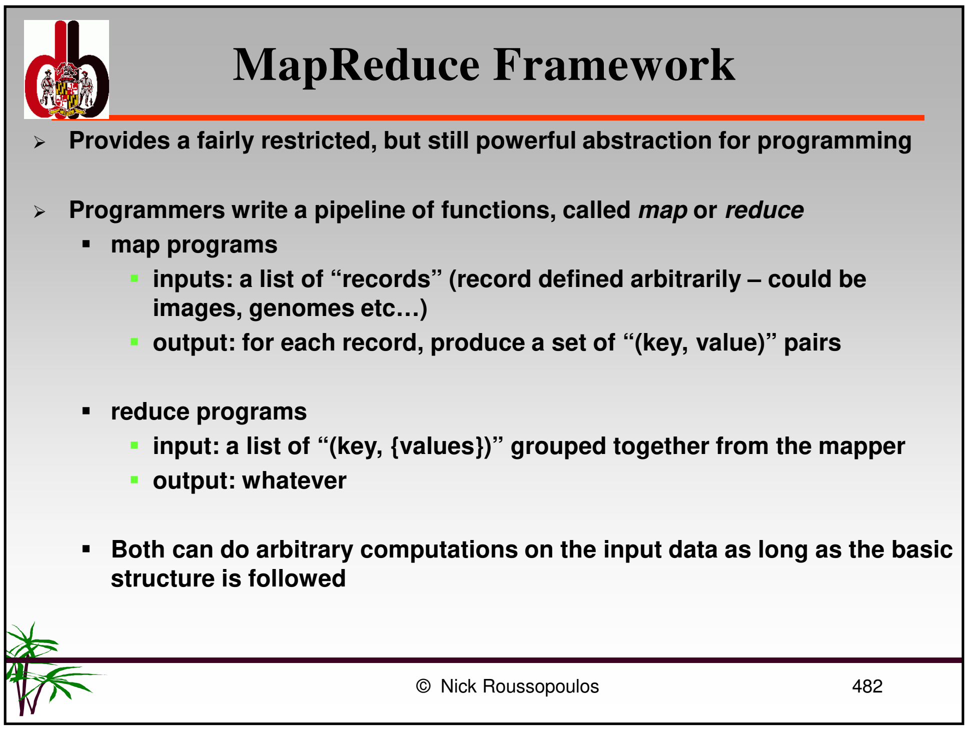
Programming Frameworks

- Third key piece emerged from efforts to "scale out"
  - i.e., distribute work over large numbers of machines (1000's of machines)
- Parallelism has been around for a long time
  - Both in a single machine, and as a cluster of computers
- But always been considered very hard to program, especially the distributed kind
  - Too many things to keep track of
    - How to parallelize, how to distribute the data, how to handle failures etc etc..
- Google developed MapReduce and BigTable frameworks, and ushered in a new era



Programming Frameworks

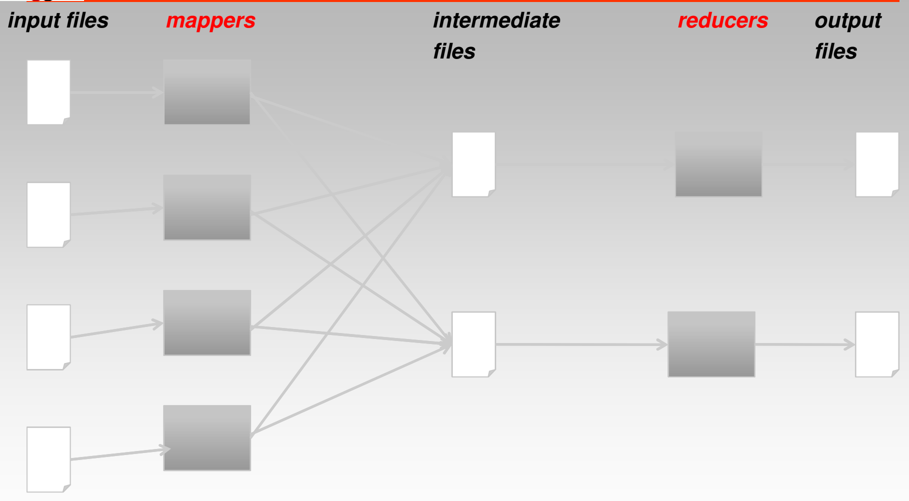
- Note the difference between "scale up" and "scale out"
  - scale up usually refers to using a larger machine – easier to do
  - scale out refers to distributing over a large number of machines
- Even with VMs, I still need to know how to distribute work across multiple VMs
  - Amazon's largest single instance may not be enough



MapReduce Framework

- Provides a fairly restricted, but still powerful abstraction for programming
- Programmers write a pipeline of functions, called *map* or *reduce*
  - map programs
    - inputs: a list of "records" (record defined arbitrarily – could be images, genomes etc…)
    - output: for each record, produce a set of "(key, value)" pairs
  - reduce programs
    - input: a list of "(key, {values})" grouped together from the mapper
    - output: whatever
  - Both can do arbitrary computations on the input data as long as the basic structure is followed

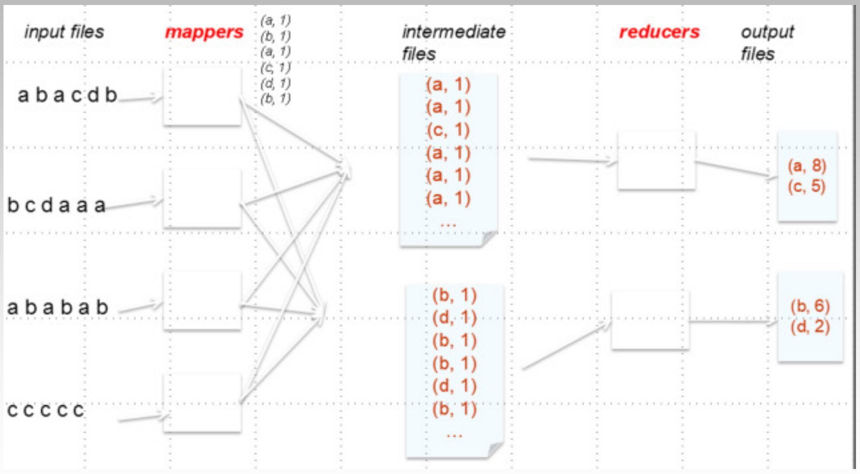
MapReduce Framework



Word Count Example

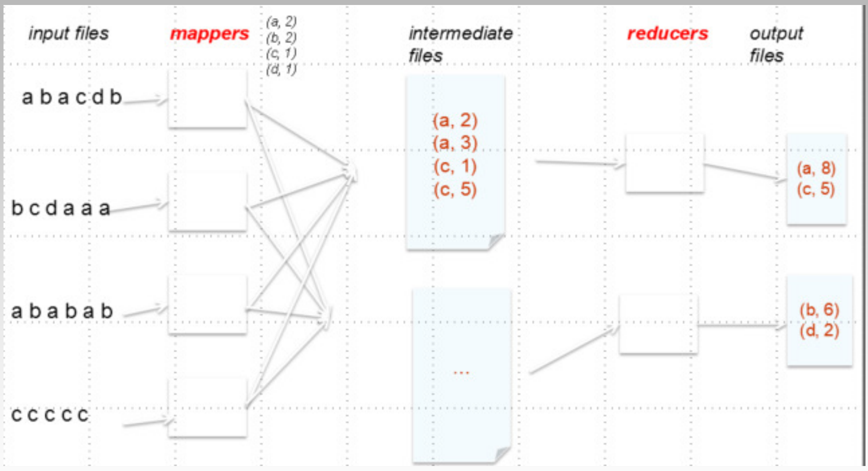
```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

MapReduce Framework: Word Count



© Nick Roussopoulos 485

More Efficient Word Count



Called "mapper-side" combiner

© Nick Roussopoulos 486

MapReduce Framework

- Has been used within Google for:
  - Large-scale machine learning problems
  - Clustering problems for Google News etc..
  - Generating summary reports
  - Large-scale graph computations
- Also replaced the original tools for large-scale indexing
  - i.e., generating the inverted indexes etc.
  - runs as a sequence of 5 to 10 Mapreduce operations
- Hadoop:
  - Open-source implementation of Mapreduce
  - Supports many other technologies as well
  - Very widely used
  - Many startups focusing on providing Hadoop services, different points in the Hadoop/DB space etc…

Bigtable/Key-Value Stores

- MapReduce/Hadoop great for batch processing of data
  - Much ongoing work on efficiency, other programming frameworks (e.g., for graph analytics, scientific applications)
- There is another usecase
  - Very very large-scale web applications that need real-time access with few ms latencies
- Bigtable (open source implementation: HBase)
  - Think of it as a very large distributed hash table
  - Support “put” and “get” operations
    - With some additional support to deal with versions
- Much work on these systems
  - Issues with “consistency” and “performance” quite challenging

Key-Value Stores

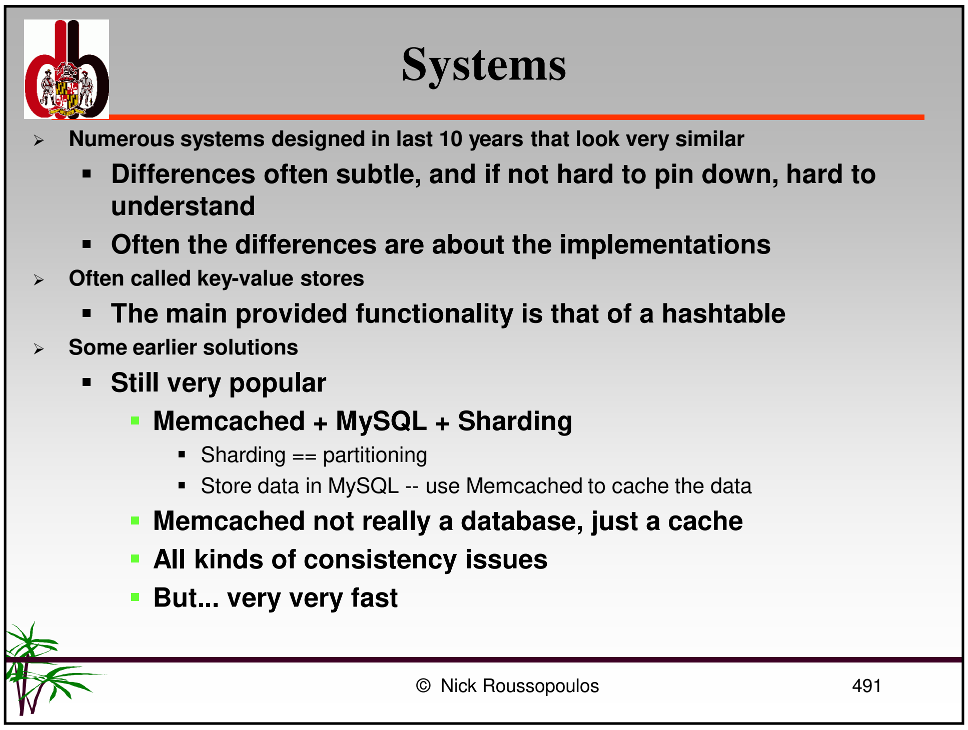
- Some Interesting (somewhat old) numbers (http://highscalability.com)
  - Twitter: 177M tweets sent on 3/1/2011 (nothing special about the date), 572,000 accounts added on 3/12/2011
  - Dropbox: 1M files saved every 15 mins
  - Stackoverflow: 3M page views a day (Redis for caching)
  - Wordnik: 10 million API Requests a Day on MongoDB and Scala
  - Mollom: Killing Over 373 Million Spams at 100 Requests Per Second (Cassandra)
  - Facebook's New Real-time Messaging System: HBase to Store 135+ Billion Messages a Month
  - Reddit: 270 Million Page Views a Month in May 2010 (Memcache)
- How to support such scale?
  - Databases typically not fast enough
  - Facebook aims for 3-5ms response times

© Nick Roussopoulos 489

Issues

- Data Consistency, High Availability, and Low Latency hard to guarantee simultaneously
  - Impossible in some cases especially if networks can fail
- Distributed transactions
  - If a transaction spans multiple machines, what to do ?
  - Correct solution: Two-phase Commit
    - Multi-round protocol
    - Too high latencies
- Dealing with replication
  - Replication of data is a must
  - How to keep them updated?
    - Eager vs lazy replication
    - Significant impact on consistency and availability
- Many systems in this space sacrifice consistency

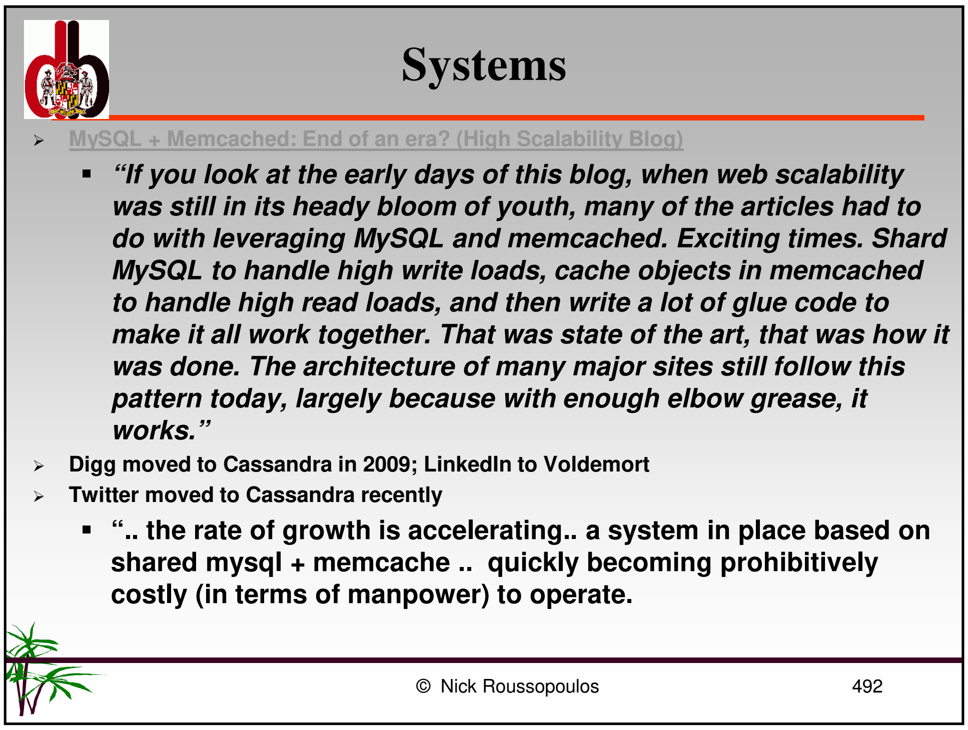
© Nick Roussopoulos 490



Systems

- Numerous systems designed in last 10 years that look very similar
  - Differences often subtle, and if not hard to pin down, hard to understand
  - Often the differences are about the implementations
- Often called key-value stores
  - The main provided functionality is that of a hashtable
- Some earlier solutions
  - Still very popular
    - Memcached + MySQL + Sharding
      - Sharding == partitioning
      - Store data in MySQL -- use Memcached to cache the data
    - Memcached not really a database, just a cache
    - All kinds of consistency issues
    - But... very very fast

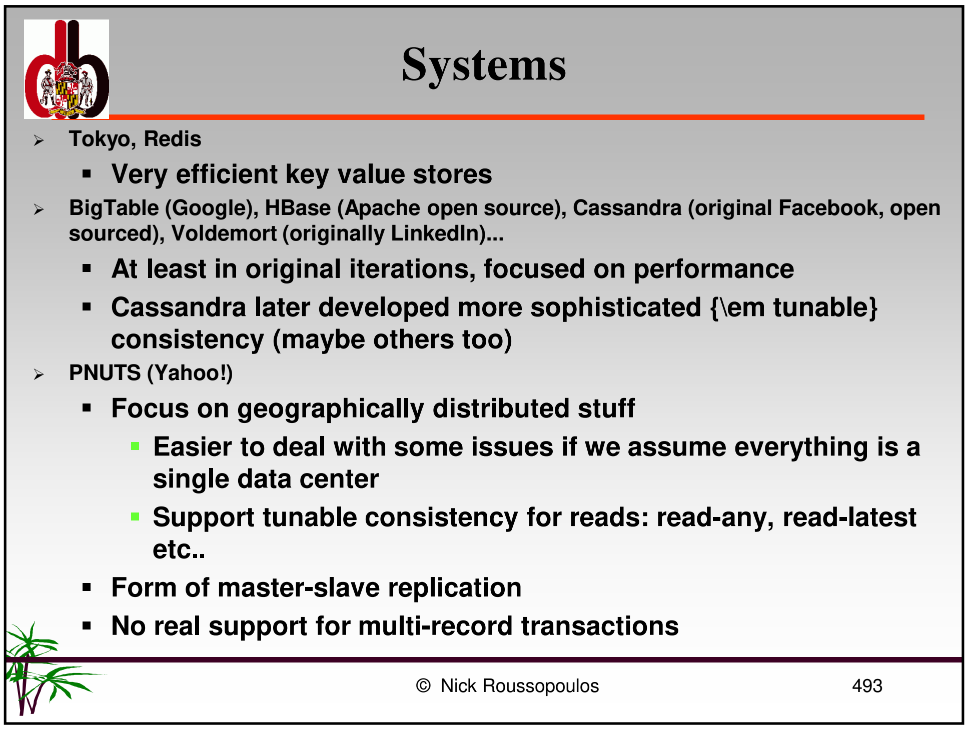
© Nick Roussopoulos 491



Systems

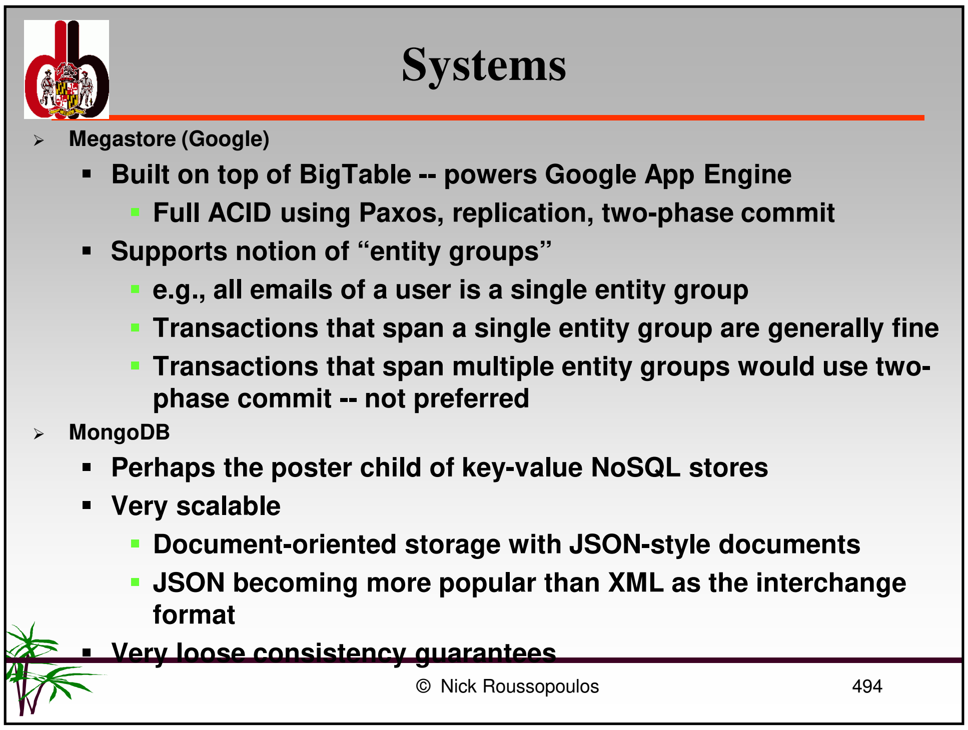
- MySQL + Memcached: End of an era? (High Scalability Blog)
  - *"If you look at the early days of this blog, when web scalability was still in its heady bloom of youth, many of the articles had to do with leveraging MySQL and memcached. Exciting times. Shard MySQL to handle high write loads, cache objects in memcached to handle high read loads, and then write a lot of glue code to make it all work together. That was state of the art, that was how it was done. The architecture of many major sites still follow this pattern today, largely because with enough elbow grease, it works."*
- Digg moved to Cassandra in 2009; LinkedIn to Voldemort
- Twitter moved to Cassandra recently
  - ".. the rate of growth is accelerating.. a system in place based on shared mysql + memcache .. quickly becoming prohibitively costly (in terms of manpower) to operate.

© Nick Roussopoulos 492



Systems

- Tokyo, Redis
  - Very efficient key value stores
- BigTable (Google), HBase (Apache open source), Cassandra (original Facebook, open sourced), Voldemort (originally LinkedIn)...
  - At least in original iterations, focused on performance
  - Cassandra later developed more sophisticated {\em tunable} consistency (maybe others too)
- PNUTS (Yahoo!)
  - Focus on geographically distributed stuff
    - Easier to deal with some issues if we assume everything is a single data center
    - Support tunable consistency for reads: read-any, read-latest etc..
  - Form of master-slave replication
  - No real support for multi-record transactions



Systems

- Megastore (Google)
  - Built on top of BigTable -- powers Google App Engine
    - Full ACID using Paxos, replication, two-phase commit
  - Supports notion of “entity groups”
    - e.g., all emails of a user is a single entity group
    - Transactions that span a single entity group are generally fine
    - Transactions that span multiple entity groups would use two-phase commit -- not preferred
- MongoDB
  - Perhaps the poster child of key-value NoSQL stores
  - Very scalable
    - Document-oriented storage with JSON-style documents
    - JSON becoming more popular than XML as the interchange format
  - Very loose consistency guarantees