Final Exam Review 2

Kathleen Durant
CS 3200 Northeastern University
Lecture 23

QUERY EVALUATION PLAN

Representation of a SQL Command

```
SELECT          {DISTINCT} <list of columns>
FROM            <list of relations>
{WHERE          <list of "Boolean Factors">}
{GROUP BY <list of columns>
{HAVING         <list of Boolean Factors>}}
{ORDER BY <list of columns>};
```

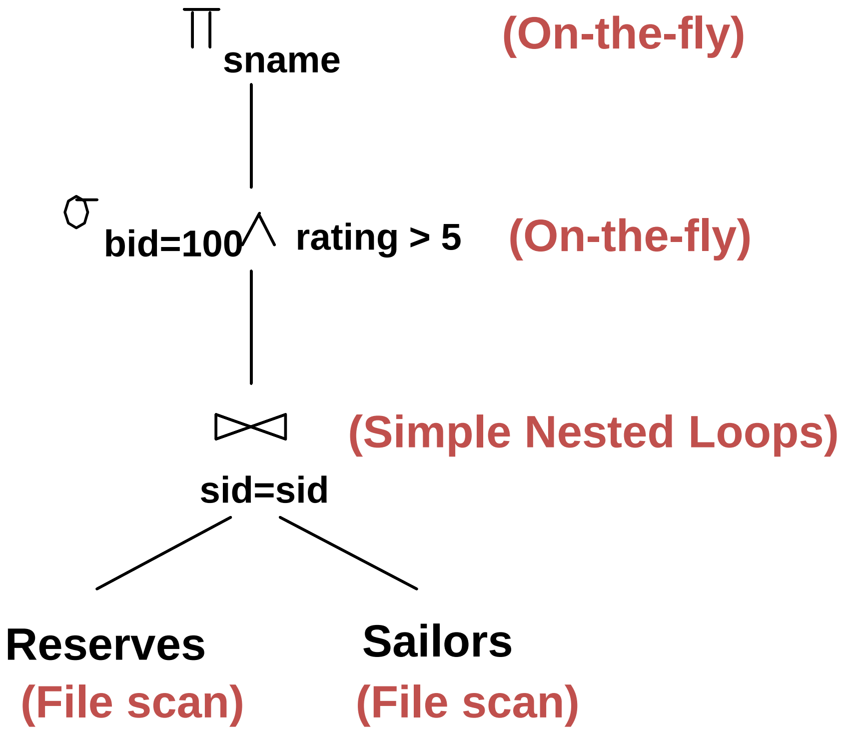
- Query Semantics:
  1. Take Cartesian product (a.k.a. cross-product) of relations in FROM clause, projecting only those columns that appear in other clauses
  2. If a WHERE clause exists, apply all filters in it
  3. If a GROUP BY clause exists, form groups on the result
  4. If a HAVING clause exists, filter groups with it
  5. If an ORDER BY clause exists, make sure output is in the right order
  6. If there is a DISTINCT modifier, remove duplicates

System Catalog

- System information: buffer pool size and page size.
- For each relation:
  - relation name, file name, file structure (e.g., heap file)
  - attribute name and type of each attribute
  - index name of each index on the relation
  - integrity constraints…
- For each index:
  - index name and structure (B+ tree)
  - search key attribute(s)
- For each view:
  - view name and definition
- Statistics about each relation (R) and index (I):

Query Evaluation Plan

- *Query evaluation plan* is an extended RA tree, with additional annotations:
  - *access method* for each relation;
  - *implementation method* for each relational operator.
- Cost Approximation
- Manipulating plans:
  - *Relational Alebra Equivalence*
  - *Push selections below the join*.
  - *Materialization*: store a temporary relation T,
  - if the subsequent join needs to *scan T multiple times*.
    - The opposite is *pipelining*



Equivalence Rules

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

3. Only the last in a sequence of projection operations is needed, the others can be omitted.

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$$

4. Selections can be combined with Cartesian products and theta joins.
   a. $\sigma_{\theta}(E_1 \text{ X } E_2) = E_1 \bowtie_{\theta} E_2$
   b. $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$

Equivalence Rules (Slide 2)

5. Theta-join operations (and natural joins) are commutative.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. (a) Natural join operations are associative:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(b) Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

where θ_2 involves attributes from only E_2 and E_3.

Equivalence Rules (Slide 3)

8. The projections operation distributes over the theta join operation as follows:

   (a) if Π involves only attributes from $L_1 \cup L_2$:

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   (b) Consider a join $E_1 \bowtie_\theta E_2$.

   - Let L_1 and L_2 be sets of attributes from E_1 and E_2, respectively.
   - Let L_3 be attributes of E_1 that are involved in join condition θ, but are not in $L_1 \cup L_2$, and
   - let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$.

$$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$

Equivalence Rules (Slide 4)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

   - (set difference is not commutative).

10. Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. The selection operation distributes over $\cup$, $\cap$ and $-$.

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta(E_2)$$

and similarly for $\cup$ and $\cap$ in place of $-$

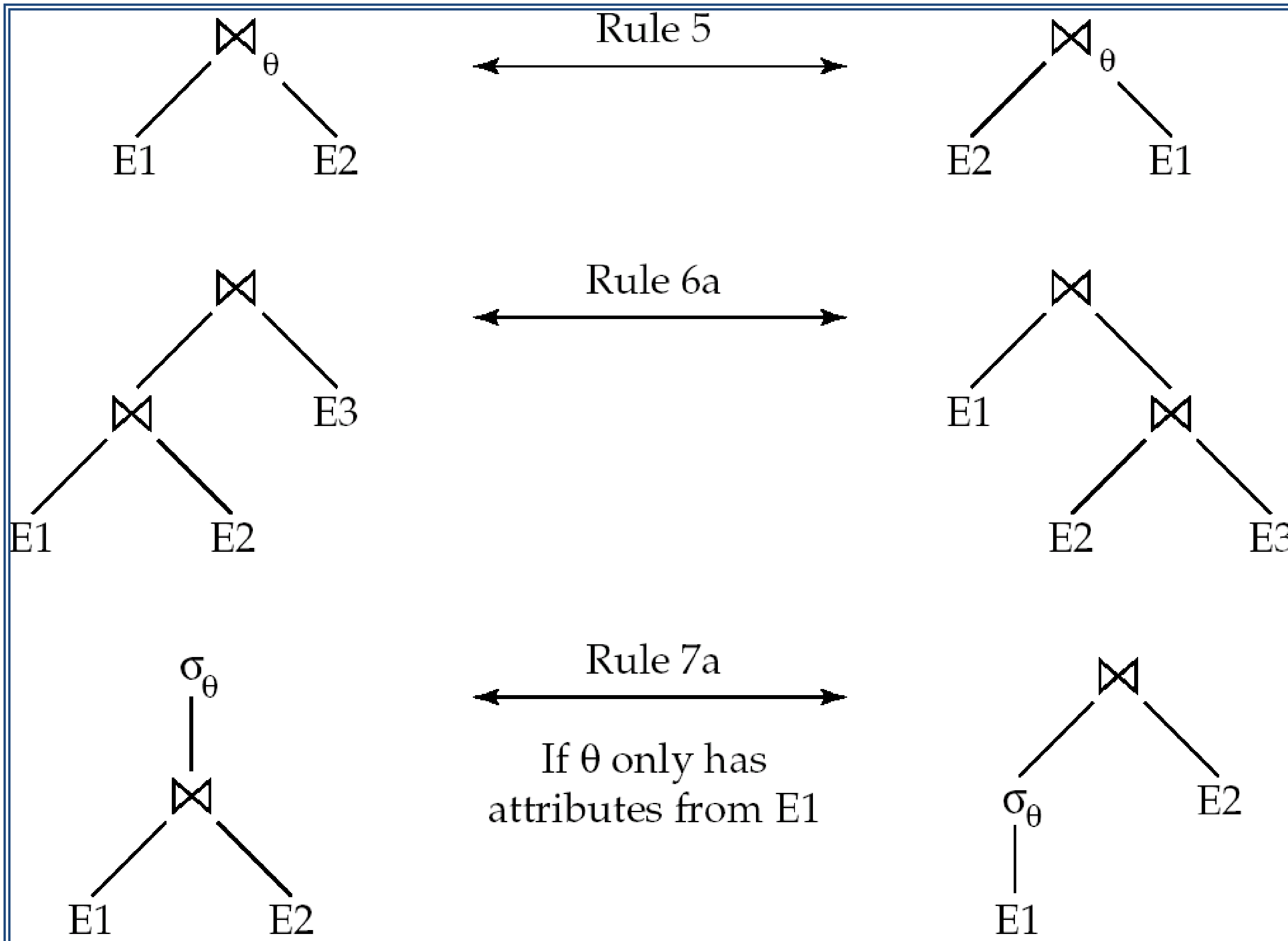
Also: $$\sigma_\theta (E_1 - E_2) = \sigma_\theta(E_1) - E_2$$

and similarly for $\cap$ in place of $-$, but not for $\cup$

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

Pictorial Depiction of Equivalence Rules



Query Blocks: Units of Optimization

- An SQL query is parsed into a collection of *query blocks*, and these are optimized one block at a time.

```
SELECT  S.sname
FROM    Sailors S
WHERE  S.age IN
    (SELECT  MAX (S2.age)
      FROM  Sailors S2
      GROUP BY  S2.rating)
```

Outer block   *Nested block*

- Nested blocks are usually treated as calls to a subroutine, made once per outer tuple.

Cost Estimation for Multi-relation Plans

- Consider a query block:

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

- *Reduction factor (RF)* is associated with each *term*.
- *Max number tuples in result* = the product of the cardinalities of relations in the FROM clause.
- *Result cardinality* = max # tuples * product of all RF's.
- Multi-relation plans are built up by joining one new relation at a time.
  - Cost of join method, plus estimate of join cardinality gives us both cost estimate and result size estimate.

Query Optimization: Summary

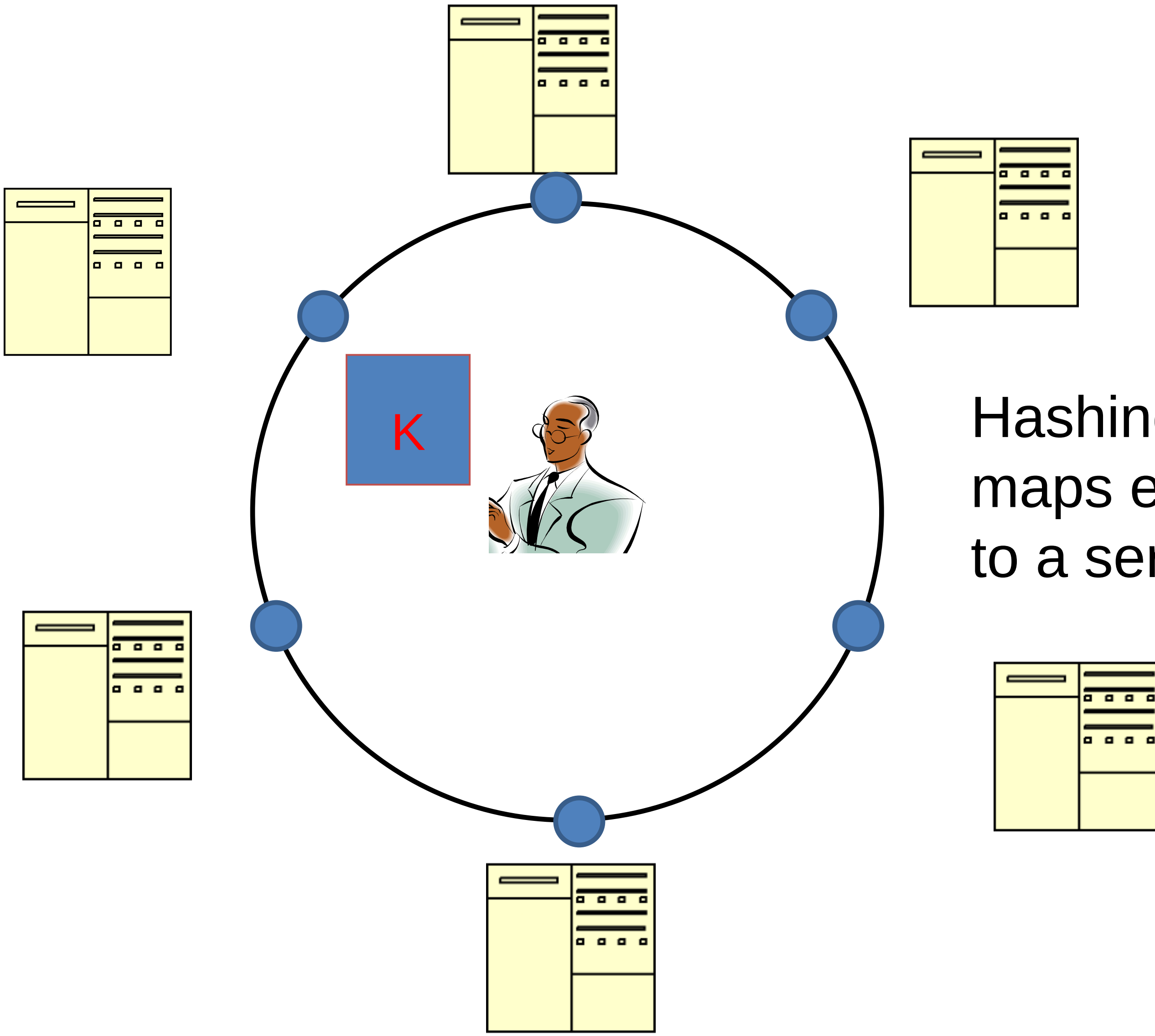
- Two parts to optimizing a query:
  - Consider a set of alternative plans.
    - Must prune search space; typically, left-deep plans only.
  - Must estimate cost of each plan that is considered.
    - Must estimate size of result and cost for each plan node.
    - *Key issues*: Statistics, indexes, operator implementations.

Query Optimization: Summary

- Single-relation queries:
  - All access paths considered, cheapest is chosen.
  - *Issues*: Selections that *match* index, whether index key has all needed fields and/or provides tuples in a desired order.
- Multiple-relation queries:
  - All single-relation plans are first enumerated.
    - Selections/projections considered as early as possible.
  - Next, for each 1-relation plan, all ways of joining another relation (as inner) are considered.
  - Next, for each 2-relation plan that is `retained', all ways of joining another relation (as inner) are considered, etc.
  - At each level, for each subset of relations, only best plan for each interesting order of tuples is `retained'.

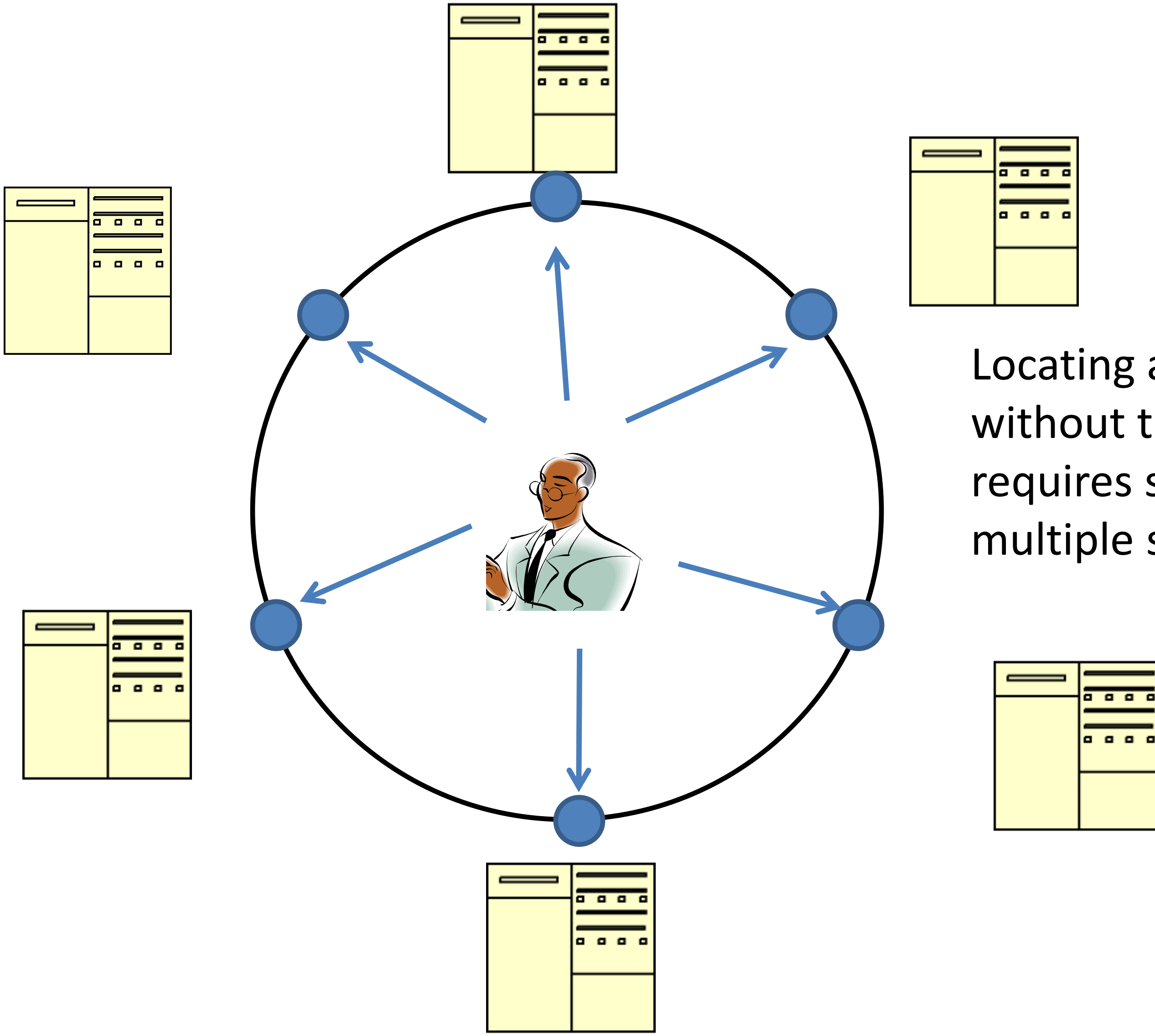
NO SQL

Typical NoSQL architecture



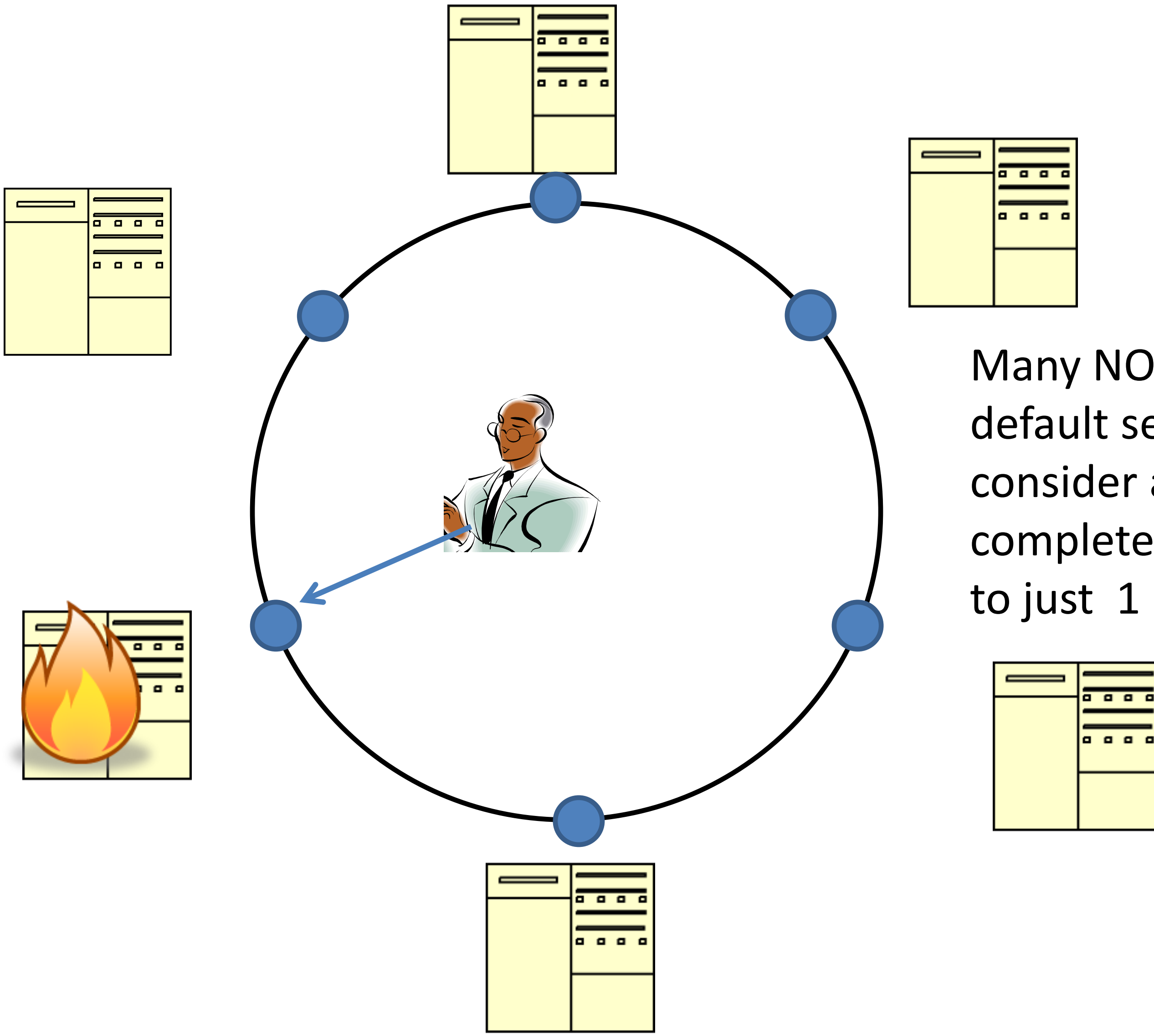
Hashing function maps each key to a server

The search problem: No Hash key



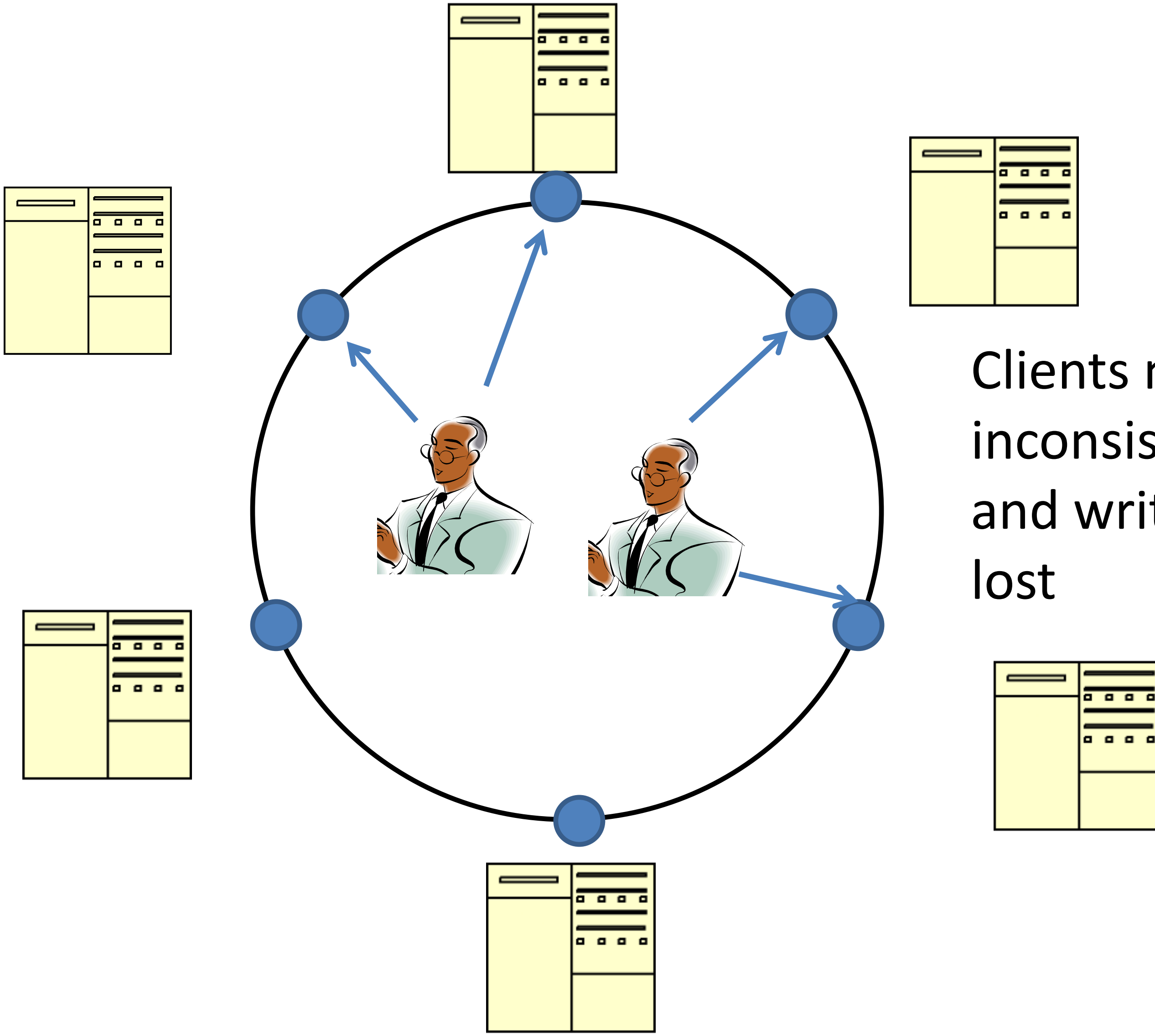
Locating a record without the hash key requires searching multiple servers

The Fault Tolerance problem



Many NOSQL system's default settings consider a write complete after writing to just 1 node

The consistency problem



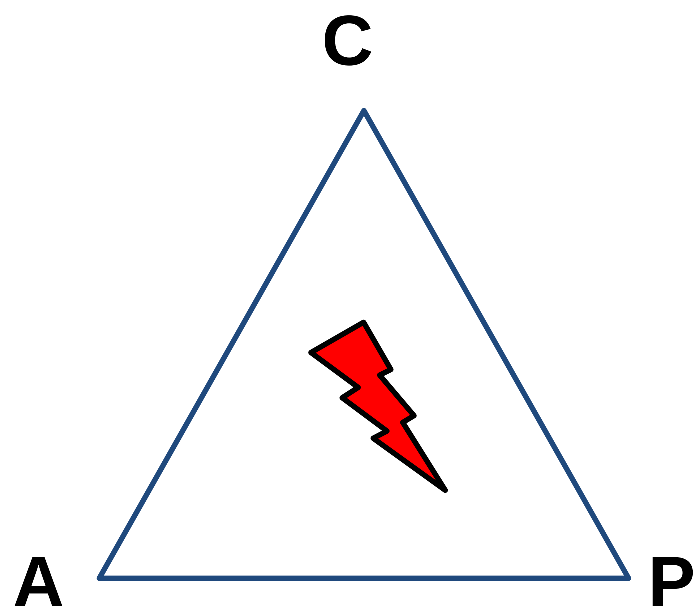
Clients may read inconsistent data and writes may be lost

Theory of NOSQL: CAP

GIVEN:

- Many nodes
- Nodes contain ***replicas of partitions*** of the data

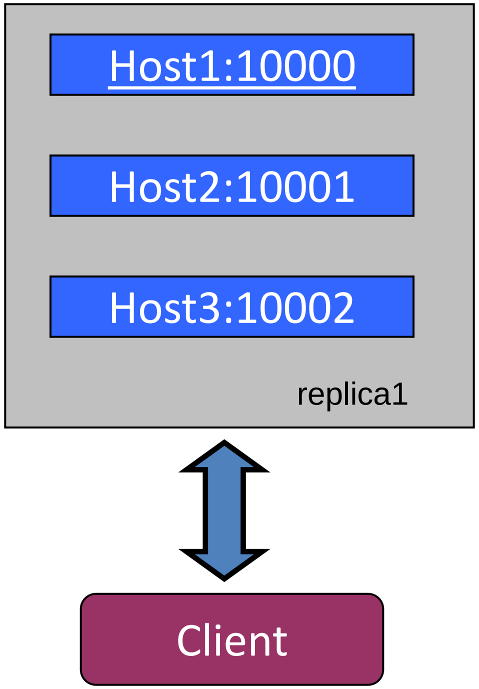
- **C**onsistency
  - all replicas contain the same version of data
- **A**vailability
  - system remains operational on failing nodes
- **P**artition tolarence
  - multiple entry points
  - system remains operational on system split



CAP Theorem: satisfying all three at the same time is impossible

Replica Sets

- Redundancy and Failover
- Zero downtime for upgrades and mainentance



- Master-slave replication
  - Strong Consistency
  - Delayed Consistency

- Geospatial features

How does it vary from SQL?

- Looser schema definition
- Various schema models
  - Key value pair
  - Document oriented
  - Graph
  - Column based
- Applications written to deal with specific documents
  - Applications aware of the schema definition as opposed to the data
- Designed to handle distributed, large databases
- Trade off: ad hoc queries for speed and growth of database

ACID - BASE

Atomicity

Consistency

Isolation

Durability

⟺

Basically

Available (CP)

Soft-state

Eventually consistent (Asynchronous propagation)

Pritchett, D.: BASE: An Acid Alternative (queue.acm.org/detail.cfm?id=1394128)

What is MapReduce?

- Programming model for expressing distributed computations on massive amounts of data

AND

- An execution framework for large-scale data processing on clusters of commodity servers

Programming Model

- Transforms set of input key-value pairs to set of output key-value pairs
  - Map function written by user
  - Map: (k1, v1) → list (k2, v2)
  - MapReduce library groups all intermediate pairs with same key together
- Reduce written by user
  - Reduce: (k2, list (v2)) → list (v2)
  - Usually zero or one output value per group
  - Intermediate values supplied via iterator (to handle lists that do not fit in memory)

Execution Framework

- Handles scheduling of the tasks
  - Assigns workers to maps and reduce tasks
  - Handles data distribution
    - Moves the process to the data
  - Handles synchronization
    - Gathers, sorts and shuffles intermediate data
  - Handles faults
    - Detects worker failures and restarts
  - Understands the distributed file system

MongoDB Basics

- A MongoDB instance may have zero or more databases
- A database may have zero or more ‘collections’.
- A collection may have zero or more ‘documents’.
- A document may have one or more ‘fields’.
- MongoDB ‘Indexes’ function much like their RDBMS counterparts.

RDB Concepts to NO SQL

RDBMS		MongoDB
Database	⇨	Database
Table, View	⇨ ⇨	Collection
Row	⇨ ⇨	Document (JSON, BSON)
Column		Field
Index	⇨	Index
Join	⇨	Embedded Document
Foreign Key	⇨	Reference
Partition	⇨	Shard

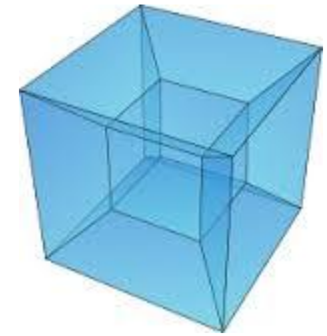
Collection is not strict about what it Stores

Schema-less

Hierarchy is evident in the design

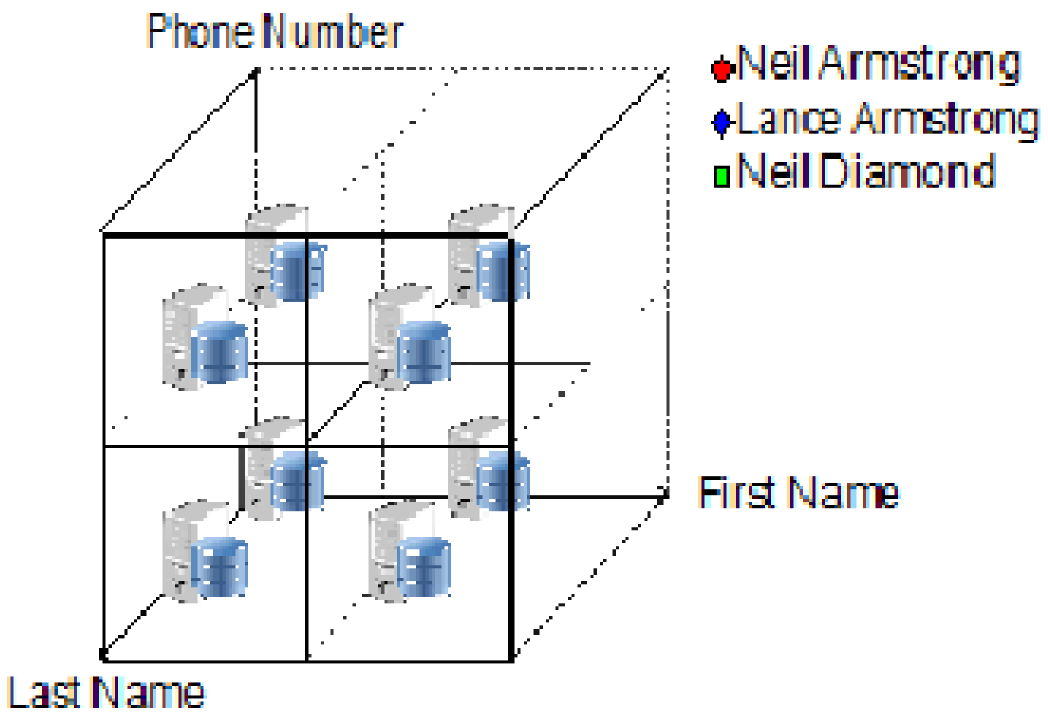
Embedded Document ?

HyperDex Key Points



- Maps records to a Hypercube Space
  - object’s key are stored in a dedicated one-dimensional subspace for efficient lookup
  - only need to contact the servers which match the regions of the hyperspace assigned for the search attributes
- Value-dependent chaining
  - Keeps replicas consistent without heavy overhead from coordination of servers
    - Uses the hypercube space
  - Appoints a point leader that contains the most recent update of a record
    - Other replicas are updated from the point leader

Each server is responsible for a region of the hyperspace



FINAL EXAM: LAST NOTES

Topics for the final exam

Topics

- File storage mechanisms
  - Abstraction:collection of records
  - Formats
  - Heap-based, Sorted, Indexed
  - RAID
- Buffer management
  - In relationship to the data manager
- Indexes
  - Primary vs. Secondary
  - Clustered vs. Unclustered
  - Tree-structured: ISAM, B+ trees
  - Hash-based indexes
- External Sort
- Query Evaluation
- Query Optimization
- NO SQL

Algorithms

- Cost model
  - Given a query, the approximate number of I/O's for different file storage mechanisms
- B+ tree bulk load
- Insertion/Deletion of records
  - B+ tree
  - ISAM
  - Extendible hashing
  - Linear hashing
- Query plan selection

Format of the final exam

- 1-2 Algorithmic/Calculation problems (40%)
  - I/O calculations
  - B+ tree insertion/deletion
  - Construct or Choose a query plan
- 1-2 open-ended responses (30%)
  - SQL vs. NO SQL
    - ACID vs. BASE
    - CAP theorem
  - Comparison of Join algorithms
  - Sort algorithms
- Some close-ended responses (30%)
  - Short collection of True and False
  - Multiple choice
  - Short definitions

Final Exam

- April 19, 2013 8:00 AM Shillman Hall 135
- Open books and open notes
  - But no portable devices (no laptops, no phones, etc.)
- 2 hour time period

That’s it

- Go over the lecture notes
- Read the book
- Go over homework 3
  - final exam questions will not be as difficult as homework problems
- Ask questions in piazza or via email
- Organize a study sheet
- Complete the example mid-term
- Practice problems