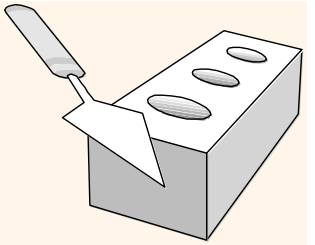
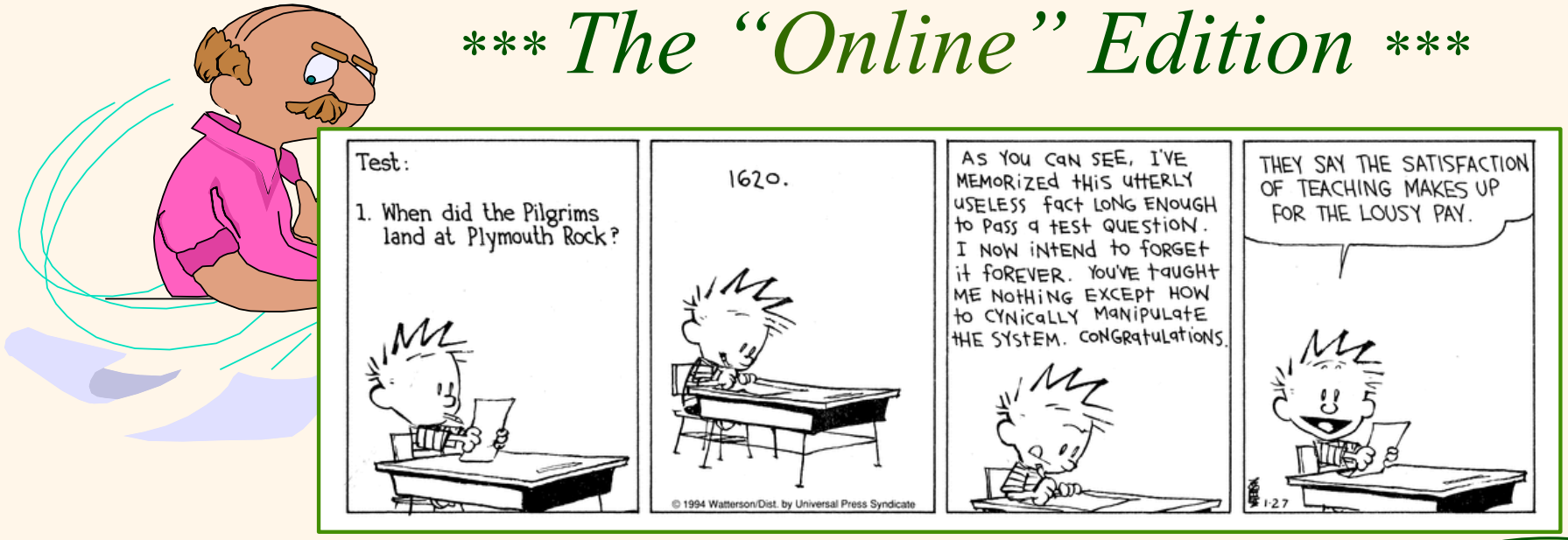
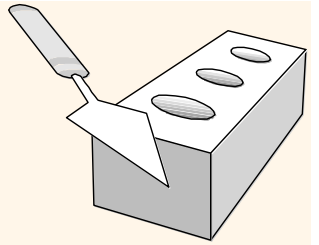


Introduction to Data Management

**** The "Online" Edition ****

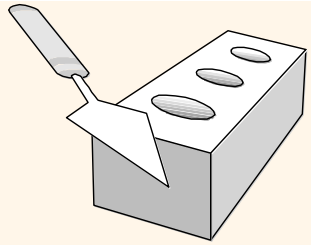


Instructor: Mike Carey
mjcarey@ics.uci.edu

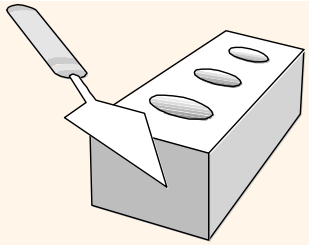


Announcements

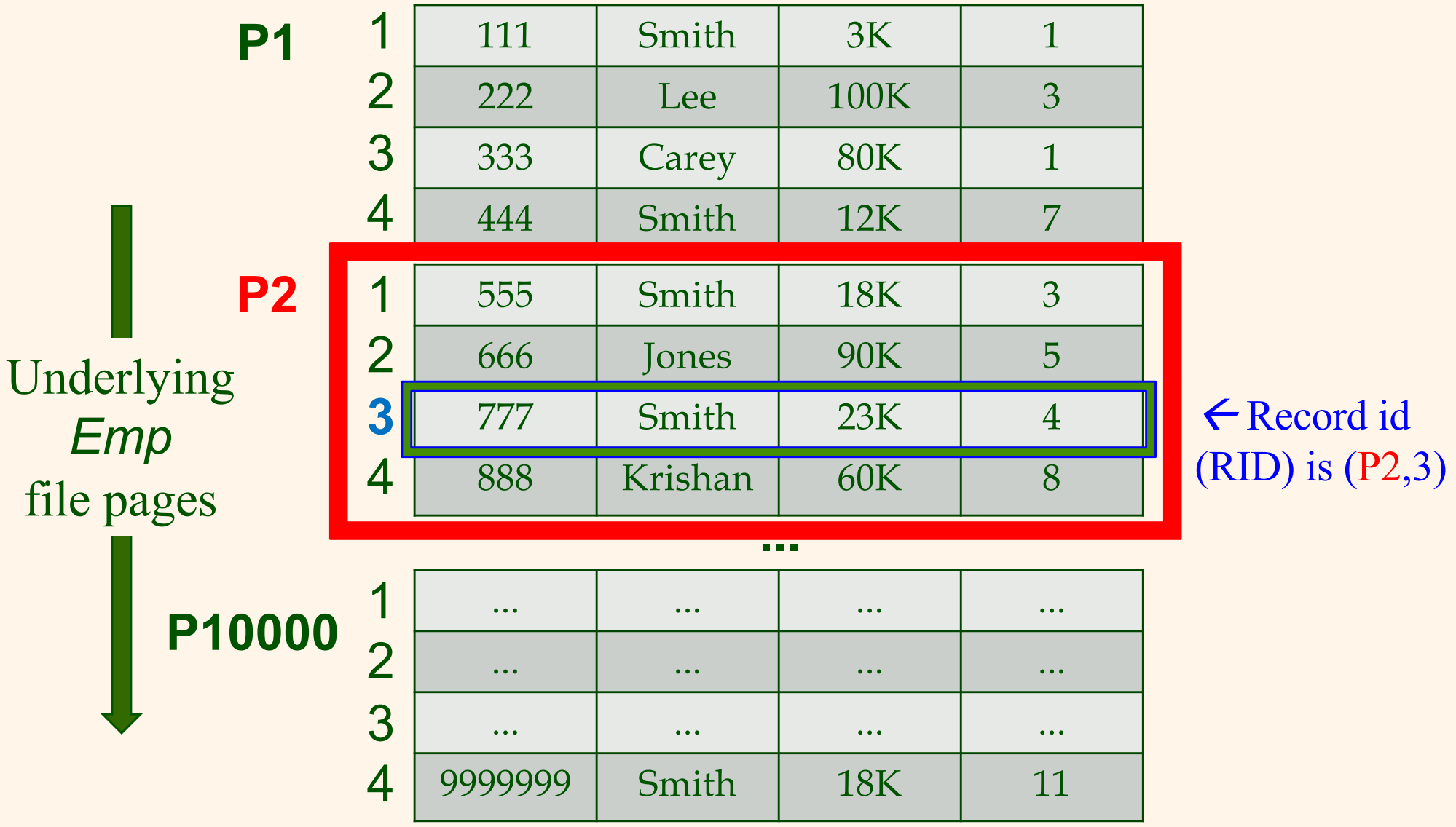
- Homework wrap-up:
  - HW8 is still in flight!
  - Due on **Sunday** at **6 PM!** (Or Monday at 6 PM if late)
- A note on Piazza points:
  - Be sure not to miss the 1-point "final"! ☺
- Endterm exam: Wed, Dec 16, from 4-5:00 PM !!!
    - Cheat sheet recommended, but open "everything" (solo!)
    - **Non**-cumulative (see Wiki syllabus for the official scope)
  - Sample exam and solution from last time now available
  - *Included: Indexes, physical design, NoSQL/JSON, transactions, ...*
- Final quiz and discussion:
  - Quiz 10 is now available and will be discussed in the optional Endterm review sessions in Mondays "discussion time slots"

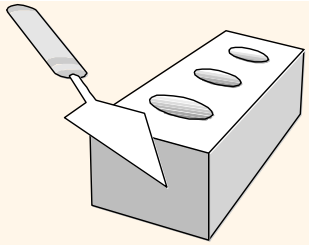


Review: Indexing Concepts



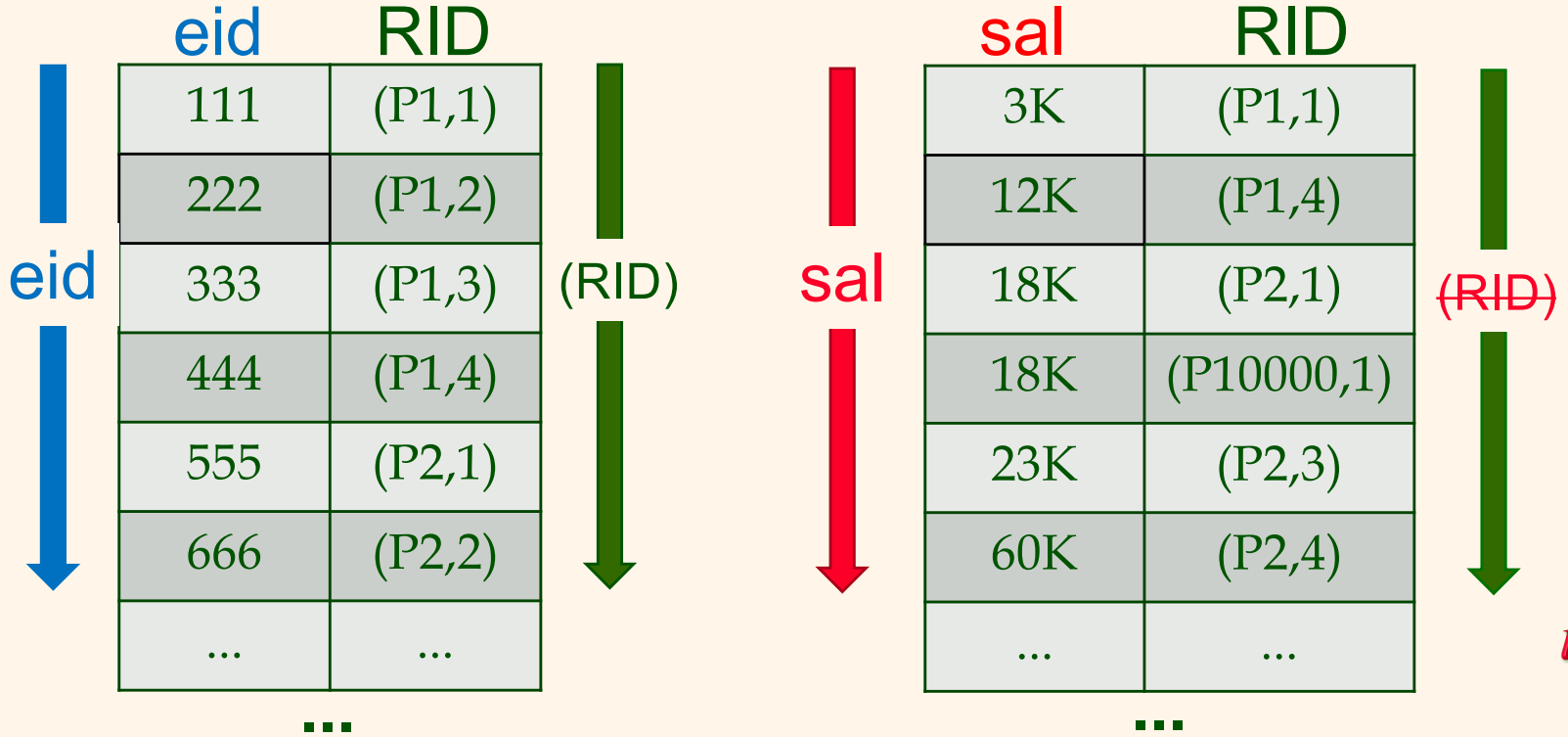
Ex: Emp(eid, ename, sal, deptid)



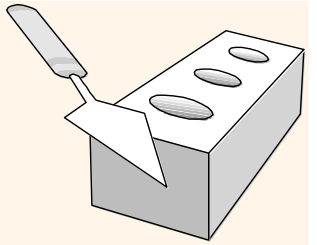


Ex: Emp(eid, ename, sal, deptid)

- Why are indexes efficient?
  - *Hundreds* of (key, RID) entries=fit on a *single page*
  - Index is therefore *much* smaller than the data file
  - Data structures guide searches – *O(1)* or $O(\log_F N)$
  - Less I/O (*fewer reads!*) to search for RIDs of interest

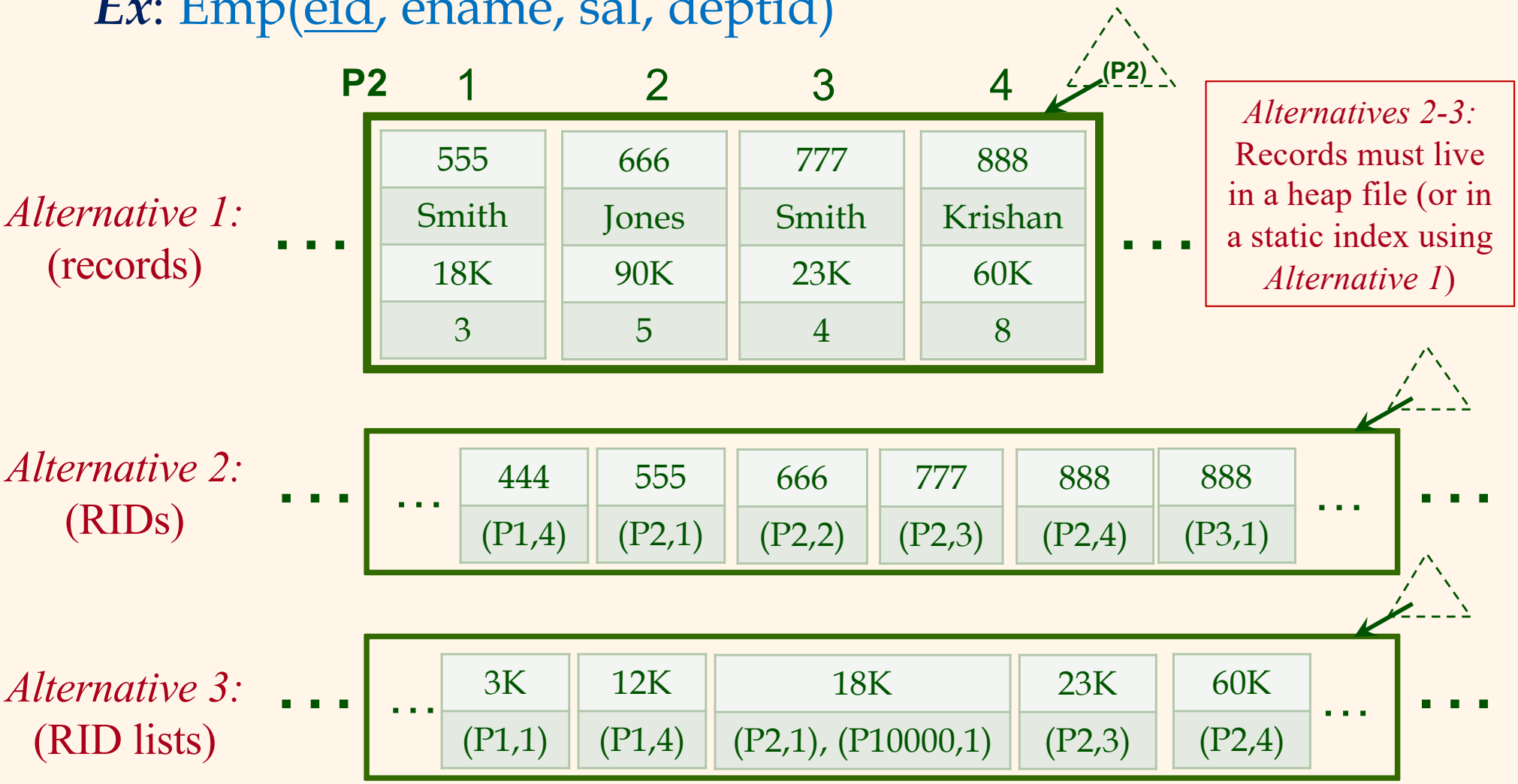


Note:
Can have multiple unclustered indexes!

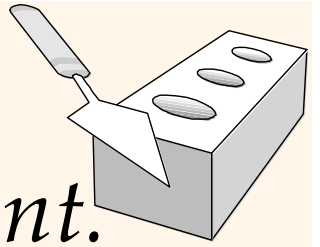


Index Leaf Page I(k) Alternatives

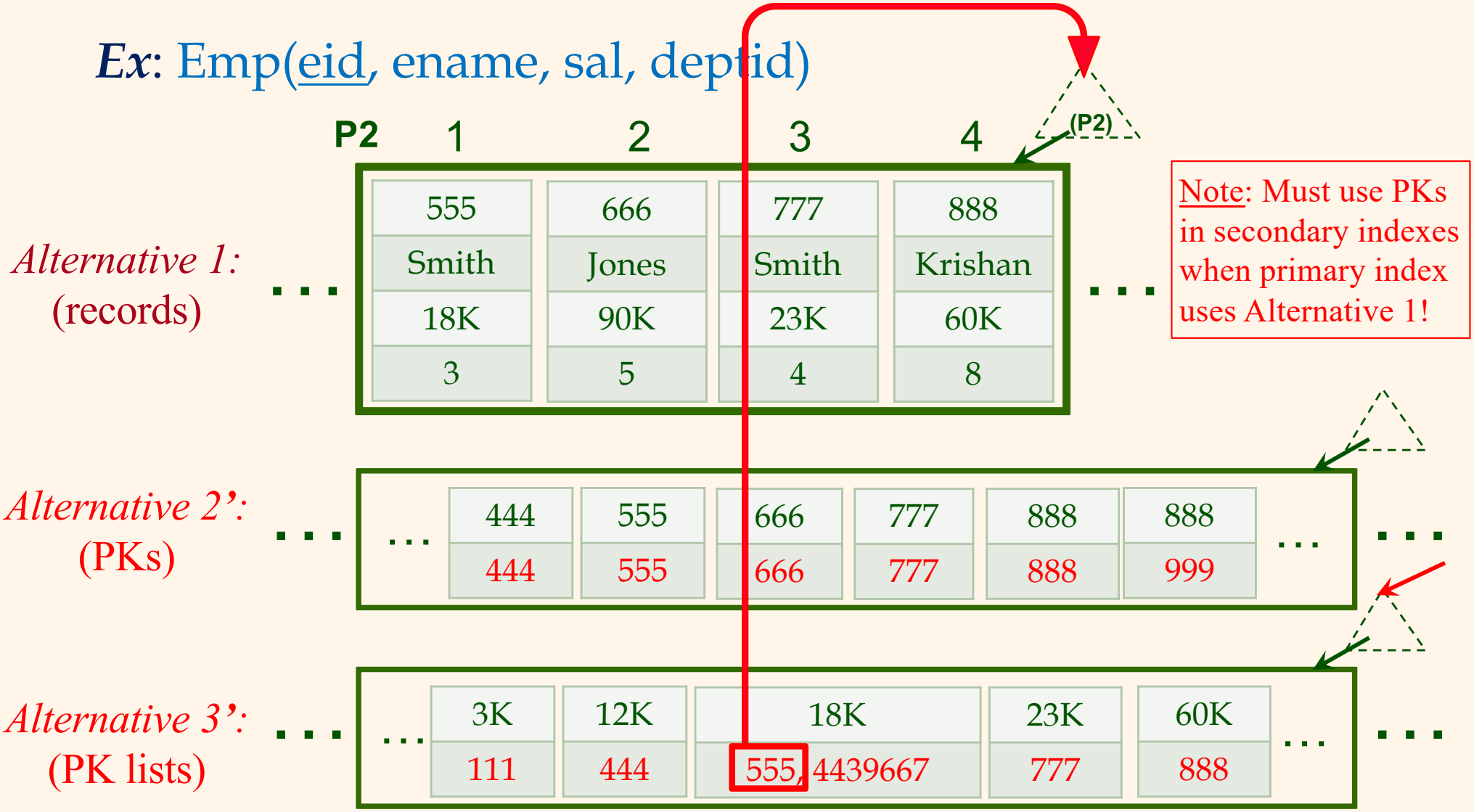
Ex: Emp(<u>eid</u>, ename, sal, deptid)



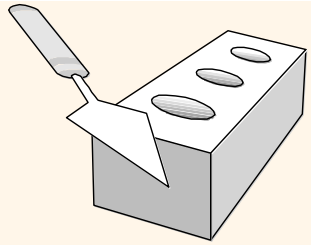
Index Leaf Page I(k) Alternatives, cont.

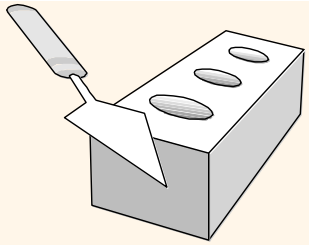


Ex: Emp(eid, ename, sal, deptid)

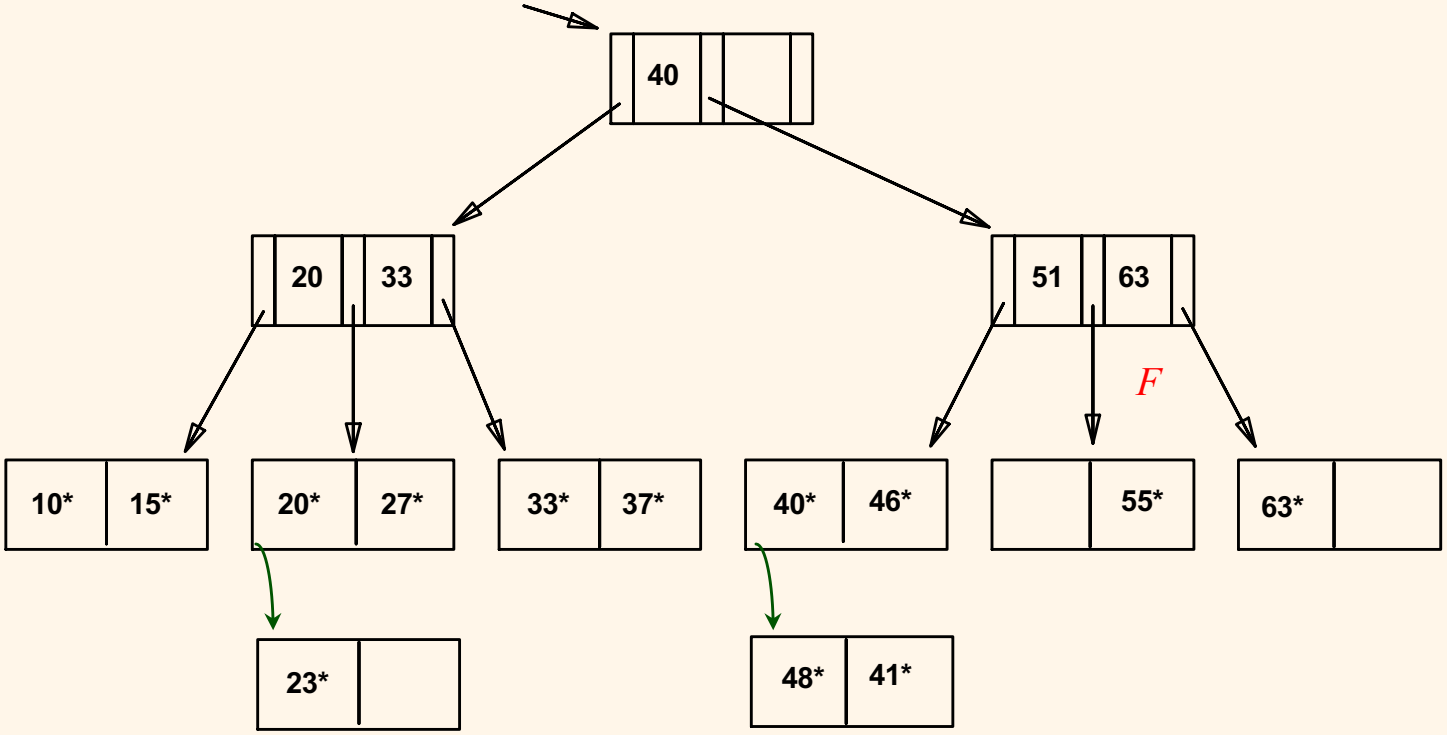


Review: Index Structures

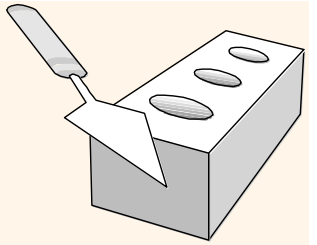




ISAM (Indexed Sequential Access Method)

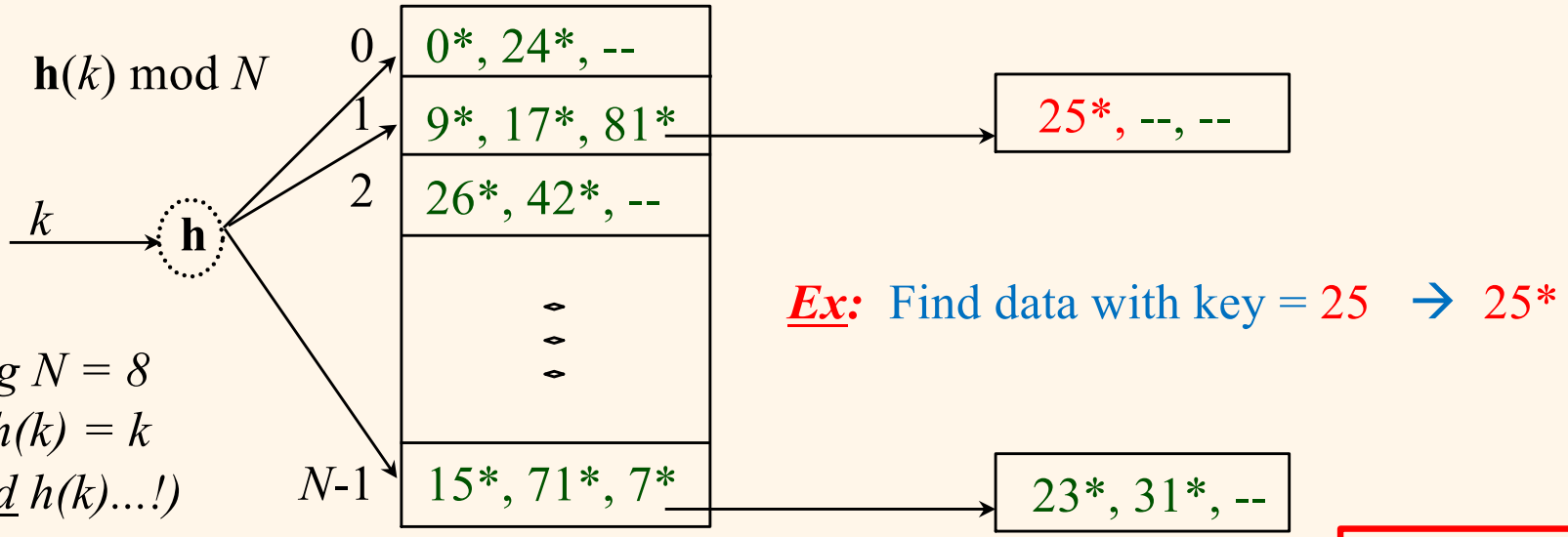


$O(\log_F N)$ – initially...



Static Hashed Indexes

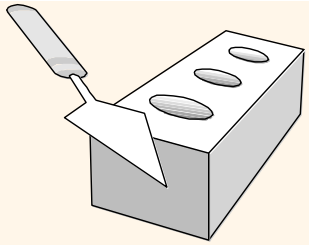
- # primary pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- **h**(*k*) mod *N* = bucket (page) to which data entry with key *k* belongs. (*N* = # of buckets)



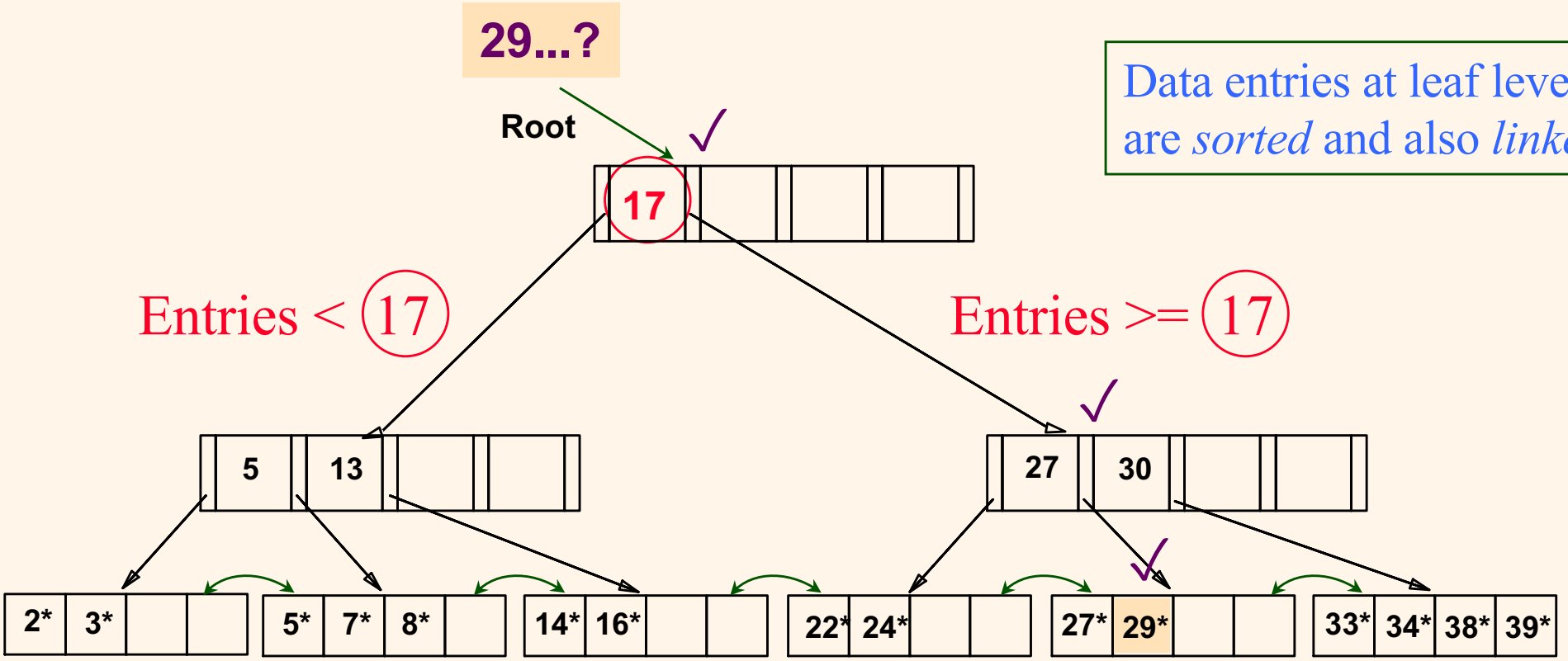
Ex: *Using N = 8 and h(k) = k (a bad h(k)...!)*

Ex: Find data with key = 25 → 25*

O(1) – initially...



B+ Trees



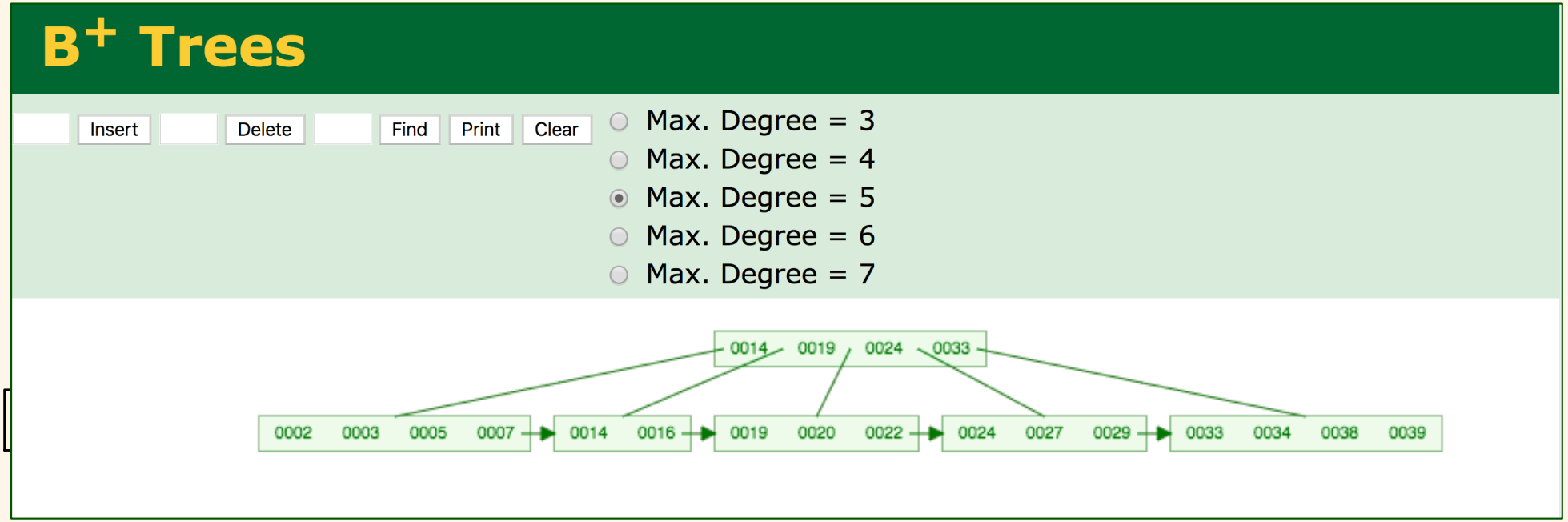
Data entries at leaf level are *sorted* and also *linked*

*k** = (*k*, I(*k*)), *e.g.*, (29, *RID(s)*)

Note: Just *3 page reads* to get from root to (any) leaf here!

$O(\log_F N)$

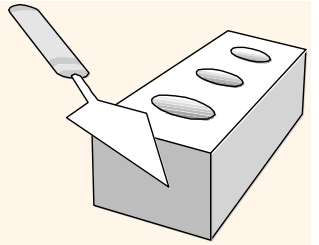
Online B+ Tree Simulator!

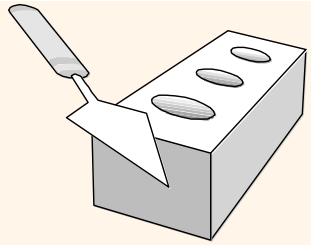


Note: Cool B+ tree algorithm visualizer (*insert, delete, search*)

- https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html
- Slight differences from our algorithms (e.g., delete key 19 above)
- Their "*Max. Degree*" is our 2d+1 (limit of 5 ptrs/node above)
- ***Try it***: 2, 3, 14, 16, 19, 20, 24, 27, 33, 34, 5, 7, 38, 39, 22, 29

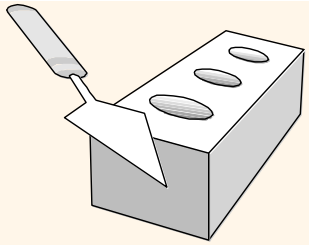
Review: Physical Design



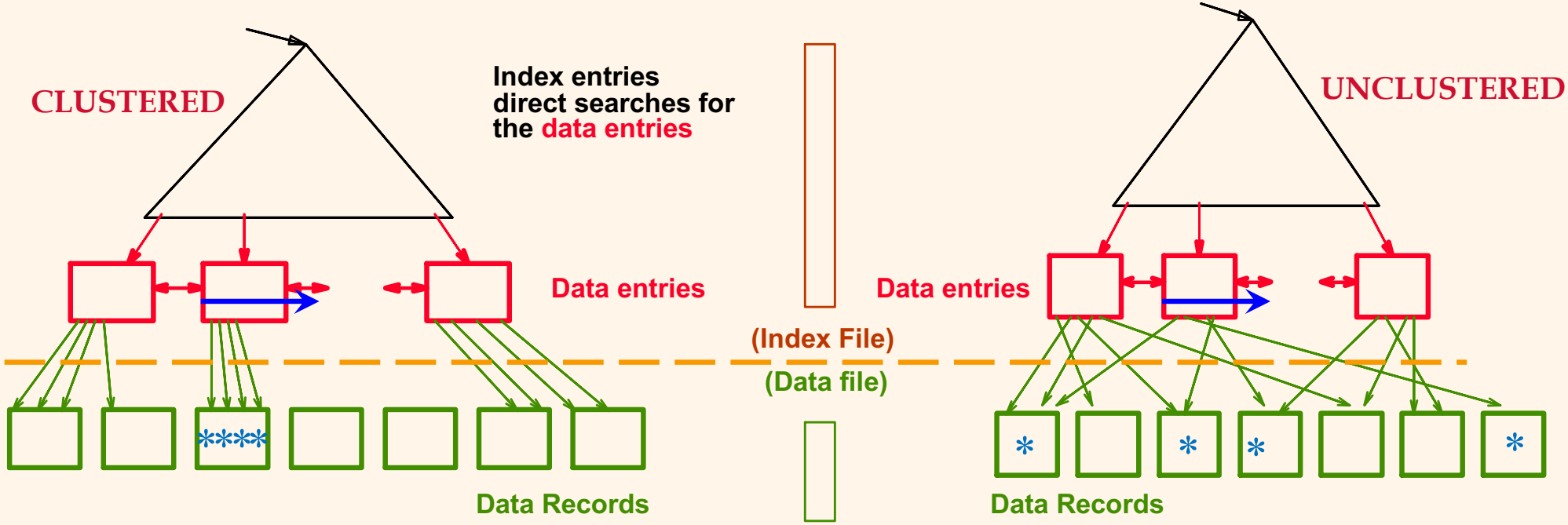


Index Classification

- *Primary* vs. *secondary*: If index search key contains primary key, this is the *primary* index.
  - *Unique* index: Search key contains a *candidate* key.
- *Clustered* vs. *unclustered*: If order of data entries matches (or nearly matches) order of data records, this is a clustered index.
  - A table can be clustered on *at most one* search key.
  - Cost of retrieving data records via an index varies ***greatly*** based on whether index is clustered or not!
  - Some systems *always* cluster on primary key.



Clustered vs. Unclustered Indexes



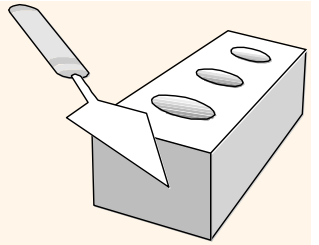
Read each data page once.

If 20 records can fit on a data page, fetching 100 records takes 5-6 I/Os

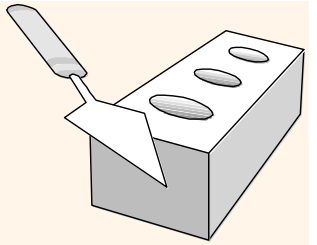
Read more pages – and repeatedly!

Fetching 100 records in this case can take 100 I/Os (one per record)

Joins & Index Selection



- When considering a query's join condition:
  - Index Nested Loop join (**INLJ**) method:
    - For each outer tuple, use its join column value to probe the inner table for matching tuples.
    - Indexing inner table join column helps!
    - Good for index to be ***clustered*** if join column is ***not*** the inner table's PK (e.g., it's an FK column).
  - Sort-Merge join (**SMJ**) method:
    - Sort outer and inner tables on join column value and scan them concurrently to match (join) tuples.
    - *Clustered* B+ trees on both join column(s) provide a free sort!
  - Hash join (**HJ**) or regular NL join (**NLJ**) methods:
    - Indexing not needed (not for the join part, that is).

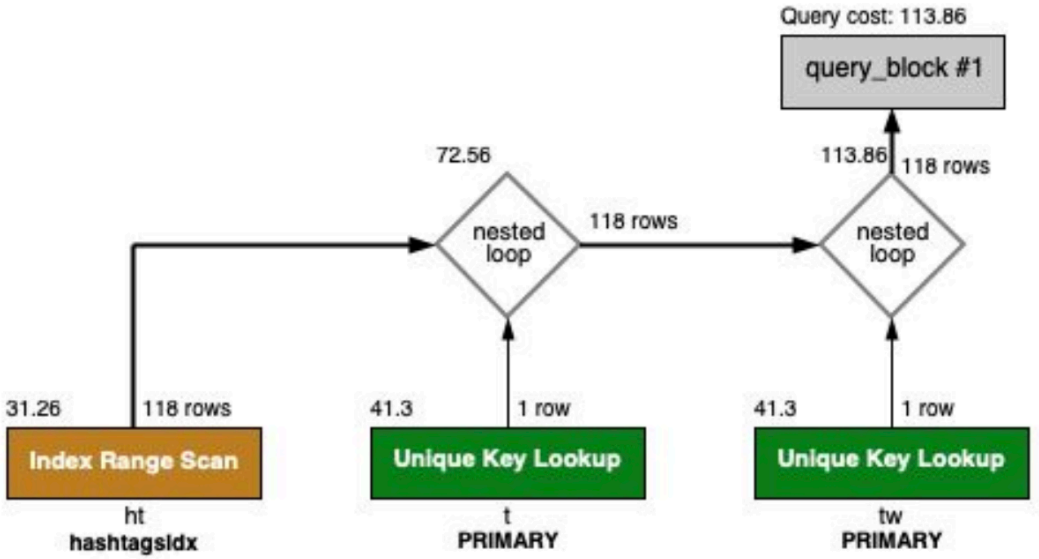


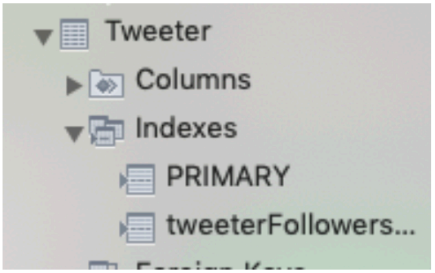
Query Plans (EXPLAIN)

the instructors' answer, *where instructors collectively construct a single answer*

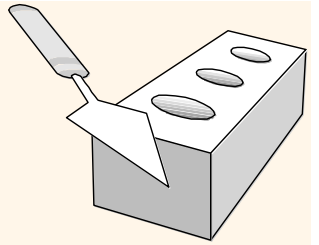
Actions

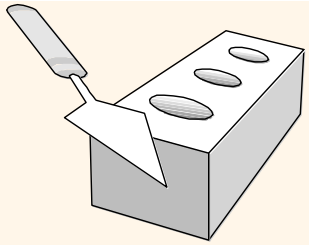
From explain (shown below), you can see that **t** (Tweet) and **tw** (Tweeter) both used the index PRIMARY.
If you see what indexes Tweeter has, you can see that it has two indexes (PRIMARY and TweeterFollowersCountIndex).
See the second screenshot below.





Review: NoSQL





Play with Apache AsterixDB!

AsterixDB

Last Published: 2020-12-06 Documentation Home Version: 0.9.6-SNAPSHOT

AsterixDB 101: An ADM and SQL++ Primer

Welcome to AsterixDB!

This document introduces the main features of AsterixDB's data model (ADM) and its new SQL-like query language (SQL++) by example. The example is a simple scenario involving (synthetic) sample data modeled after data from the social domain. This document describes a set of sample datasets, together with a set of illustrative queries, to introduce you to the "AsterixDB user experience". The complete set of steps required to create and load a handful of sample datasets, along with runnable queries and the expected results for each query, are included.

This document assumes that you are at least vaguely familiar with AsterixDB and why you might want to use it. Most importantly, it assumes you already have a running instance of AsterixDB and that you know how to query it using AsterixDB's basic web interface. For more information on these topics, you should go through the steps in Installing Asterix Using Managix before reading this document and make sure that you have a running AsterixDB instance ready to go. To get your feet wet, you should probably start with a simple local installation of AsterixDB on your favorite machine, accepting all of the default settings that Managix offers. Later you can graduate to trying AsterixDB on a cluster, its real intended home (since it targets Big Data). (Note: With the exception of specifying the correct locations where you put the source data for this example, there should no changes needed in the SQL++ statements to run the examples locally and/or to run them on a cluster when you are ready to take that step.)

As you read through this document, you should try each step for yourself on your own AsterixDB instance. You will use the AsterixDB web interface to do this, and for SQL++ you will need to select SQL++ instead of AQL as your language of choice in the Query Language box that sits underneath the UI's query entry area. Once you have reached the end of this tutorial, you will be fully armed and dangerous, with all the basic AsterixDB knowledge that you'll need to start down the path of modeling, storing, and querying your own semistructured data.

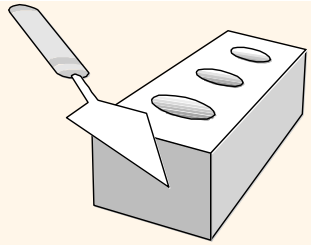
ADM: Modeling Semistructured Data in AsterixDB

In this section you will learn all about modeling Big Data using ADM, the data model of the AsterixDB BDMS.

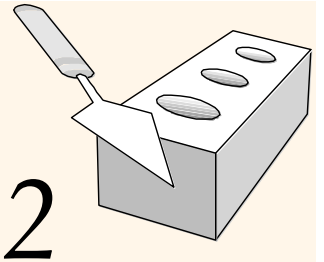
Dataverses, Datatypes, and Datasets

The top-level organizing concept in the AsterixDB world is the *dataverse*. A dataverse—short for "data universe"—is a place (similar to a database in a relational DBMS) in which to create and manage the types, datasets, functions, and other artifacts for a given AsterixDB application. When you start using an AsterixDB instance for the first time, it starts out "empty"; it contains no data other than the AsterixDB system catalogs (which live in a special dataverse called the Metadata dataverse). To store your data in AsterixDB, you will first create a dataverse and then you use it for the *datatypes* and *datasets* for managing your own data. A datatype tells AsterixDB what you know (or more accurately, what you want it to know) a priori about one of the kinds of data instances that you want AsterixDB to hold for you. A dataset is a collection of data instances of a datatype, and AsterixDB makes sure that the data instances that you put in it conform to its specified type. Since AsterixDB targets semistructured data, you can use *open* datatypes and tell it as little or as much as you wish about your data up front; the more you tell it up front, the less information it will have to store repeatedly in the individual data instances that you give it. Instances of open datatypes are permitted to have additional content, beyond what the datatype says, as long as they at least contain the information prescribed by the datatype definition. Open typing allows data to vary from one instance to another and it leaves wiggle room for application evolution in terms of what might need to be stored in the future. If you want to restrict data instances in a dataset to have only what the datatype says, and nothing extra, you can define a *closed* datatype for that dataset and AsterixDB will keep users from storing objects that have extra data in them. Datatypes are open by

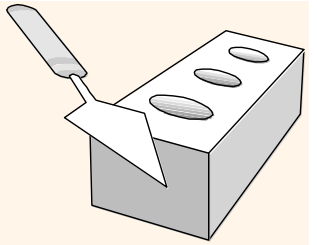
Review & Demo: Transactions



Support for Transactions in SQL-92



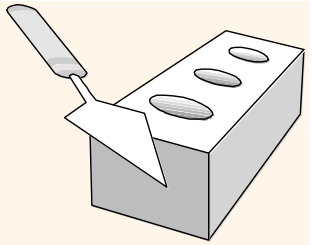
- A transaction is *automatically* started whenever a statement accesses or modifies the database
  - SELECT, UPDATE, CREATE TABLE, INSERT, ...
  - Multi-statement transactions are also supported
    - START TRANSACTION statement
- A transaction can be terminated by
  - A COMMIT statement
  - A ROLLBACK statement (SQL-speak for **abort**)
- Each transaction runs under a combination of an access mode and an *isolation level*



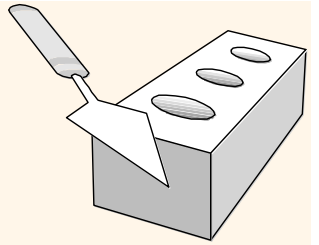
Transactions in SQL-92 (Cont'd.)

- Access mode – controls what the transaction can potentially do to the database:
  - READ ONLY: not permitted to modify the DB
  - READ WRITE (*default*): allowed to modify the DB
- ***Isolation level*** – controls the transaction's exposure to other (concurrent) transactions:
  - READ UNCOMMITTED: Can see "dirty" data!
  - READ COMMITTED: Won't ever see dirty data.
  - REPEATABLE READ: Re-reads get same result.
  - **SERIALIZABLE: No concurrency worries!**

Increasing isolation



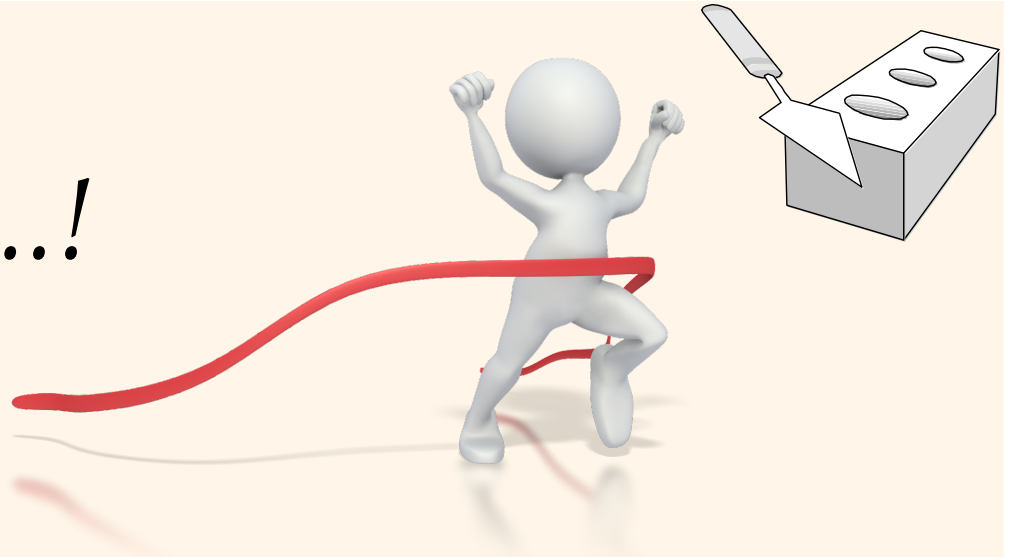
MySQL Demo



Remember the ACID Properties!

- Atomicity: Each transaction is *all or nothing*.
  - No worries about partial effects (if failures) and cleanup.
- Consistency: Each transaction moves the database from one *consistent state* to another one.
  - This is largely the application builder's responsibility.
- Isolation: Each transaction can be written as if it's the *only transaction* in existence (*if so desired*).
  - Minimize concurrency worries when building applications.
- Durability: Once a transaction has committed, its *effects will not be lost*.
  - Application code needn't worry about data loss.

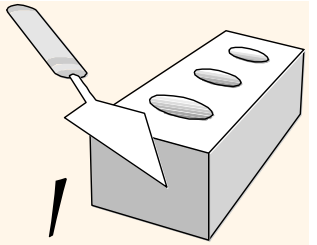
Yes...! You made it...!



Syllabus

Topic	Reading
Databases and DB Systems	Ch. 1
Entity-Relationship (E-R) Data Model	Ch. 2.1-2.5, 2.8
Relational Data Model	Ch. 3.1-3.2
E-R to Relational Translation	3.5
Relational Design Theory	Ch. 19.1-19.6, 20.8
Midterm Exam 1	*Fri, Oct 30* (during lecture time)
Relational Algebra	Ch. 4.1-4.2
Relational Calculus	Ch. 4.3-4.4
SQL Basics (SPJ and Nested Queries)	Ch. 3.4, 5.1-5.3
SQL Analytics (Aggregation, Nulls, and Outer Joins)	Ch. 5.4-5.6
Advanced SQL Goodies (Constraints, Triggers, Views, and Security)	Ch. 3.3, 3.6, 5.7-5.9, 21.1-21.3, 21.7
Midterm Exam 2	*Mon, Nov 23* (during lecture time)
Tree-Based Indexing	Ch. 9.1, 8.1-8.3, 10.1-10.2
Hash-Based Indexing	Ch. 10.3-10.8, 11.1
Physical DB Design	Ch. 8.5, 20.1-20.7
Semistructured Data Management (*a.k.a.* NoSQL)	AsterixDB SQL++ Primer, Couchbase SQL++ Book
Basics of Transactions	Ch. 16 and Lecture Notes
Endterm Exam	*Wed, Dec 16* (during final exam time)

*No no no...! That **can't** be all....!*



- **CS122A** has just given you an "outside" view of database management systems.
- **CS122B** is available to give you a "programmer's" view – with an emphasis on data-centric web applications.
- **CS122C** (*a.k.a.* CS222 lite) is available to give you an "insider's" (engine developer's) view of DB systems.
- **CS122D** goes "Beyond SQL Data Management"– NoSQL, Graph DBs, Hadoop/Spark, and more.
- **CS223** is available for learning all about transactions and distributed databases.
- **CS199** (independent project work) is also a possible avenue for gaining further experience.