1 - Introduction to databases

Guillaume Claret (class responsible)
Albert Bifet (course author)

Course

- Introduction to Databases and Relational Model
- Relational Algebra
- SQL, Views and Updates
- Functional Dependencies and Normalization
- E/R Design

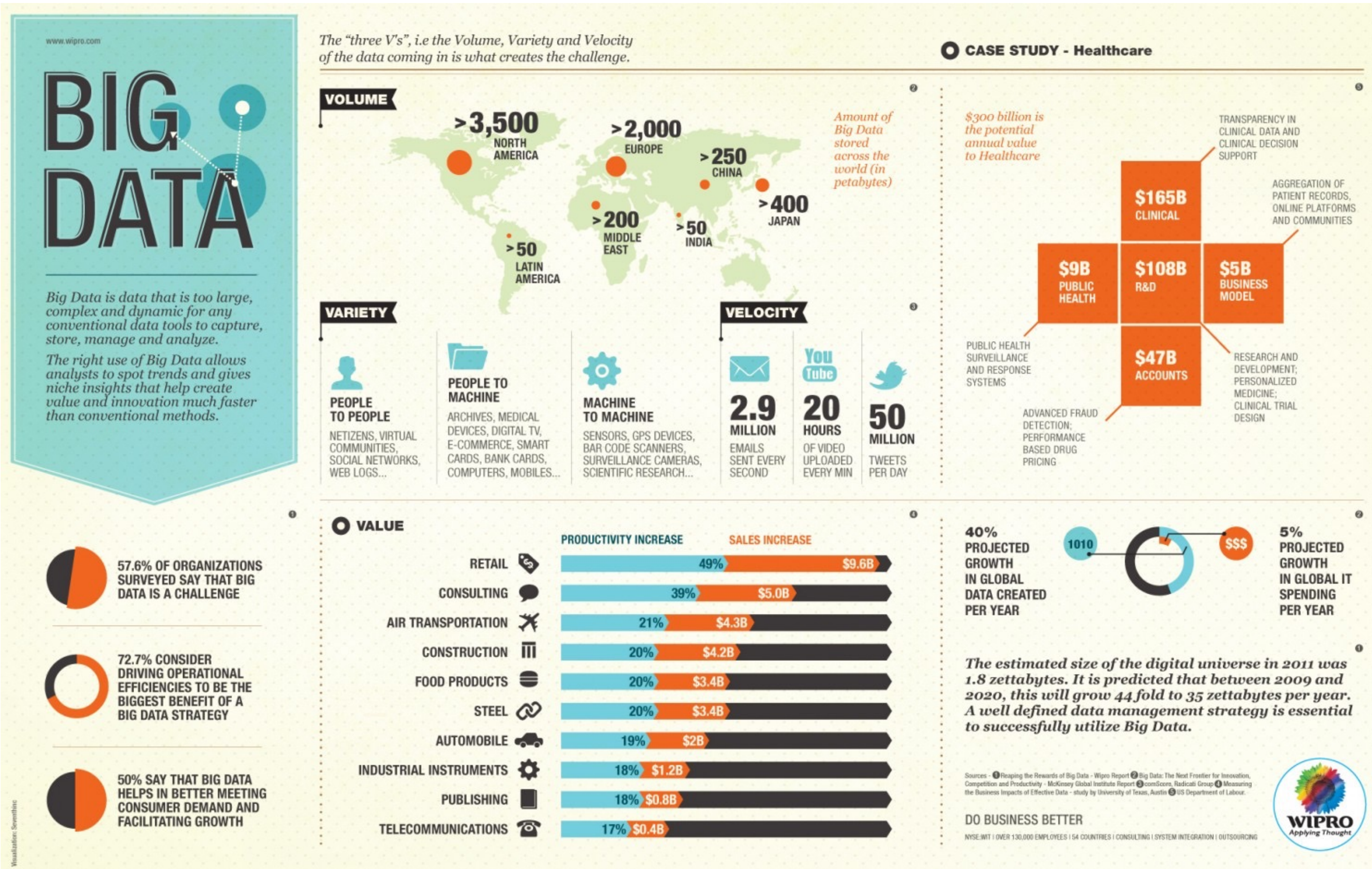
Labs: Jupyter

- Jupyter notebooks are interactive shells which **save output in a nice notebook format**
- Notebooks will be in **python**
- **Lab 1: Functional Dependencies**
- **Lab 2: SQL**
- **Lab 3: SQL**

Databases

Data-driven Society

Big Data



www.wipro.com
BIG DATA
Big Data is data that is too large, complex and dynamic for any conventional data tools to capture, store, manage and analyze.
The right use of Big Data allows analysts to spot trends and gives niche insights that help create value and innovation much faster than conventional methods.
57.6% OF ORGANIZATIONS SURVEYED SAY THAT BIG DATA IS A CHALLENGE
72.7% CONSIDER DRIVING OPERATIONAL EFFICIENCIES TO BE THE BIGGEST BENEFIT OF A BIG DATA STRATEGY
50% SAY THAT BIG DATA HELPS IN BETTER MEETING CONSUMER DEMAND AND FACILITATING GROWTH
The "three V's", i.e the Volume, Variety and Velocity of the data coming in is what creates the challenge.
VOLUME
>3,500 NORTH AMERICA
>2,000 EUROPE
>250 CHINA
>400 JAPAN
>200 MIDDLE EAST
>50 INDIA
>50 LATIN AMERICA
Amount of Big Data stored across the world (in petabytes)
VARIETY
PEOPLE TO PEOPLE
NETIZENS, VIRTUAL COMMUNITIES, SOCIAL NETWORKS, WEB LOGS...
PEOPLE TO MACHINE
ARCHIVES, MEDICAL DEVICES, DIGITAL TV, E-COMMERCE, SMART CARDS, BANK CARDS, COMPUTERS, MOBILES...
MACHINE TO MACHINE
SENSORS, GPS DEVICES, BAR CODE SCANNERS, SURVEILLANCE CAMERAS, SCIENTIFIC RESEARCH...
VELOCITY
2.9 MILLION EMAILS SENT EVERY SECOND
You Tube
20 HOURS OF VIDEO UPLOADED EVERY MIN
50 MILLION TWEETS PER DAY
CASE STUDY - Healthcare
$300 billion is the potential annual value to Healthcare
$165B CLINICAL
TRANSPARENCY IN CLINICAL DATA AND CLINICAL DECISION SUPPORT
$9B PUBLIC HEALTH
PUBLIC HEALTH SURVEILLANCE AND RESPONSE SYSTEMS
$108B R&D
RESEARCH AND DEVELOPMENT; PERSONALIZED MEDICINE; CLINICAL TRIAL DESIGN
$5B BUSINESS MODEL
AGGREGATION OF PATIENT RECORDS, ONLINE PLATFORMS AND COMMUNITIES
$47B ACCOUNTS
ADVANCED FRAUD DETECTION; PERFORMANCE BASED DRUG PRICING
VALUE
PRODUCTIVITY INCREASE
SALES INCREASE
RETAIL 49% $9.6B
CONSULTING 39% $5.0B
AIR TRANSPORTATION 21% $4.3B
CONSTRUCTION 20% $4.2B
FOOD PRODUCTS 20% $3.4B
STEEL 20% $3.4B
AUTOMOBILE 19% $2B
INDUSTRIAL INSTRUMENTS 18% $1.2B
PUBLISHING 18% $0.8B
TELECOMMUNICATIONS 17% $0.4B
40% PROJECTED GROWTH IN GLOBAL DATA CREATED PER YEAR
1010
$$$
5% PROJECTED GROWTH IN GLOBAL IT SPENDING PER YEAR
The estimated size of the digital universe in 2011 was 1.8 zettabytes. It is predicted that between 2009 and 2020, this will grow 44 fold to 35 zettabytes per year. A well defined data management strategy is essential to successfully utilize Big Data.
DO BUSINESS BETTER
WIPRO Applying Thought

All business manage data

Walmart
Save money. Live better.

(210) 377 - 1899
MANAGER SCOTT REDMAN
8500 JONES MALTSBERGER RD
SAN ANTONIO TX 78216
ST# 2404 OP# 00008097 TE# 01 TR# 01680

GARD GARLRYE	001600019333	F	2.18	N
CHEETOS	002840023985	F	1.98	N
CHEETOS	002840023986	F	1.98	N
CHEETOS	002840023988	F	1.98	N
	SUBTOTAL		8.12	
	TOTAL		8.12	
	CASH TEND		8.12	
	CHANGE DUE		0.00	

ITEMS SOLD 4

TC# 1422 8092 2433 0422 083

Our Guaranteed Low Prices
Are Unbeatable with Ad Match!
05/03/14 21:21:56

All business manage data

Data-intensive Applications

- Store data (databases)
- Speed up reads, remembering results (caches)
- Search data by keywords (search index)
- Send messages to another process asynchronously (stream application)
- Periodically crunch a large amount of accumulated data (batch processing)

Popular SQL Databases

- Open Source Databases
  - MySQL
  - PostgreSQL
  - MariaDB
- Commercial Databases
  - Oracle 12c
  - Microsoft SQL Server
  - IBM DB2
  - SAP Hana

Small Data

- SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine.
- SQLite is the most used database engine in the world
- SQLite competes with fopen().

Let’s build a database!

Simplest Database

```
#!/bin/bash

db_set (){
echo "$1,$2" >> database
}

db_get (){
grep "^$1," database | sed -e "s/^$1,//" | tail -n 1
}
```

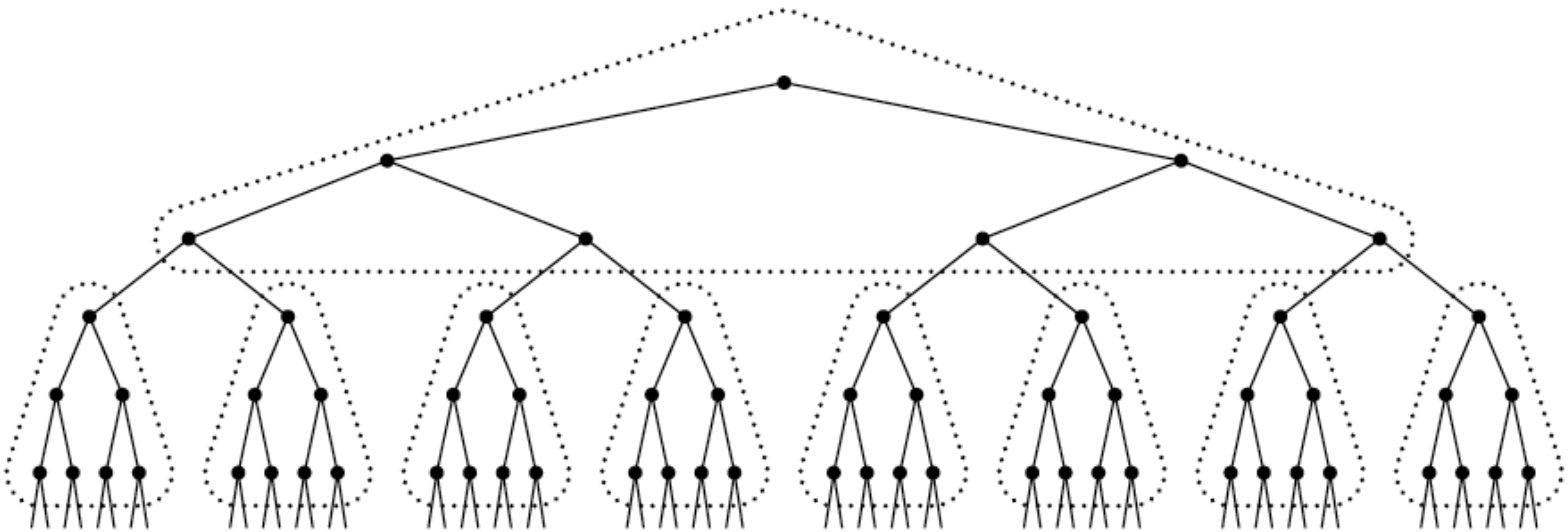
Simplest Database

```
db_set 1324 ‘John Doe, Rue Barrault, Paris’

db_set 4324 ‘Paul Ryan, Avenue Italie, Paris’


db_get 4324

Paul Ryan, Avenue Italie, Paris
```

What is missing?

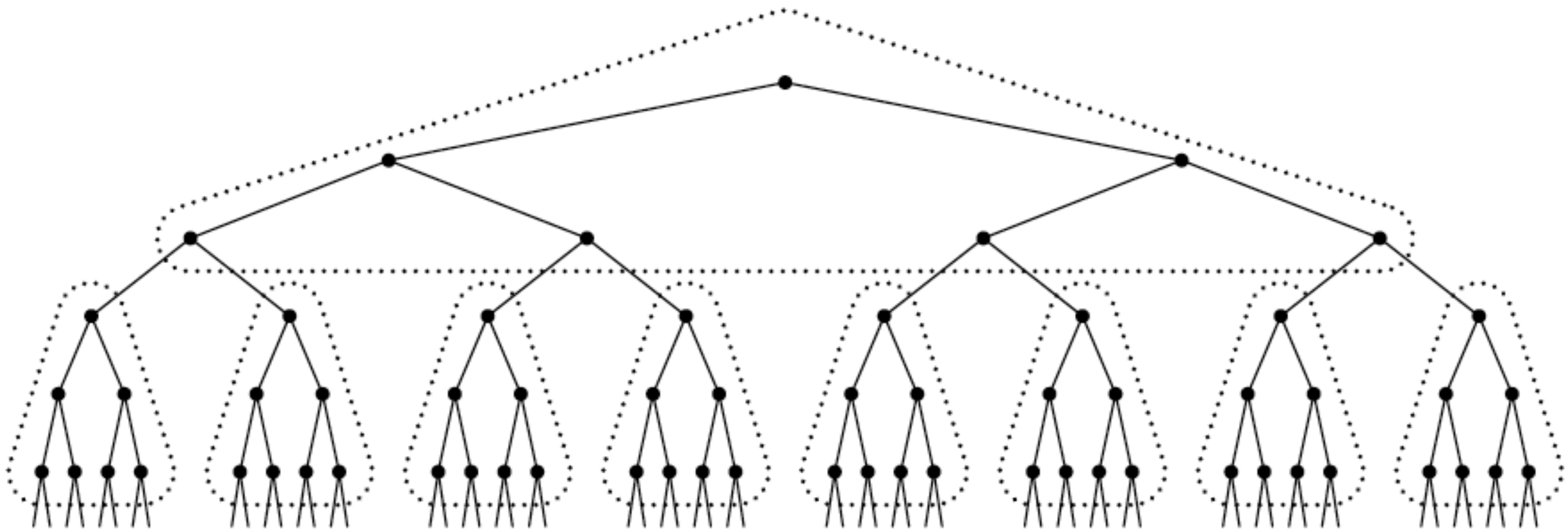
Database Indexing

Database Index

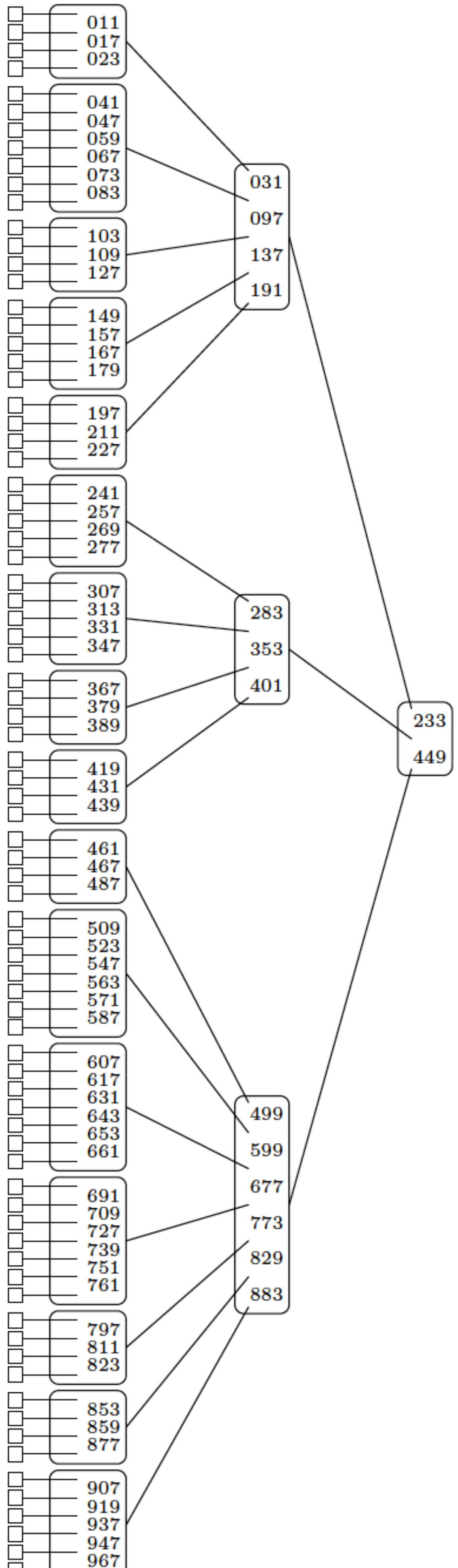


- Large binary search trees can be divided into “pages”

Database Index



- **B-Trees** are **balanced search trees** designed to work on disks and other storage devices



Motivation

B-Tree is a data structure that makes it possible to

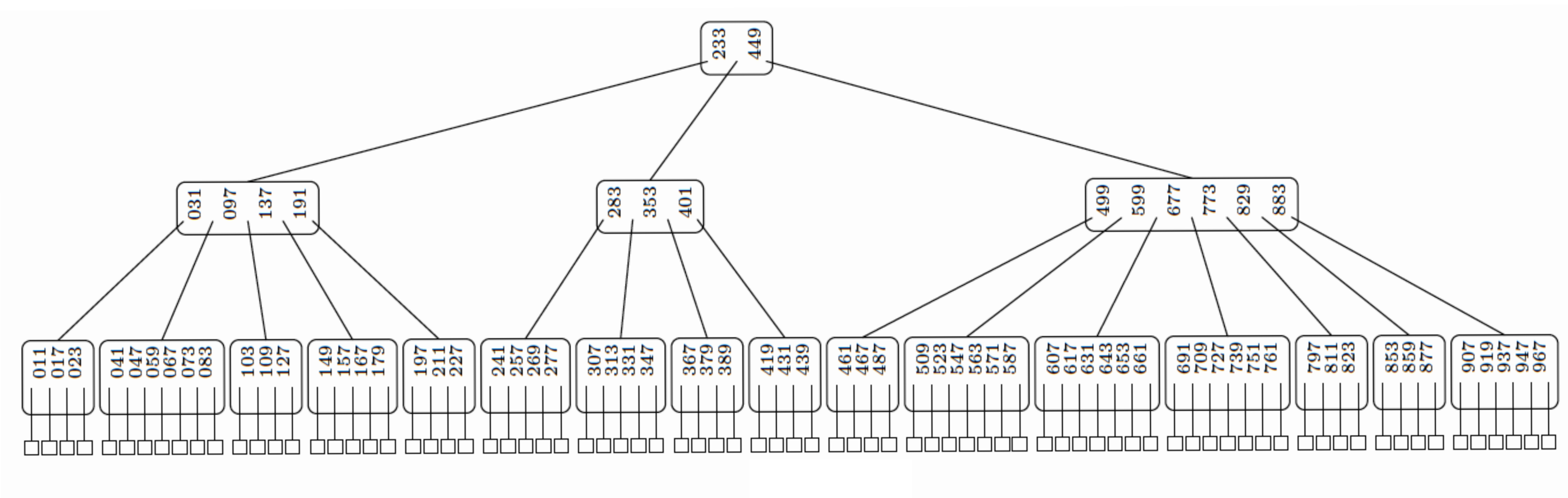
- search
- update

a large file with guaranteed efficiency, in time O(lg(*n*))

Motivation

The origin of the name “**B-Tree**” is unknown:

- Balanced, Broad, Bushy, Boeing, Bayer



Definition (Knuth)

- A **B-tree** of *minimum degree t* is a tree that satisfies:
  - Every node has at most *2t* children
  - Every node, except for the root and the leaves, has at least *t* children
  - The root has at least *2* children (unless it is a leaf)
  - All leaves appear on the same level, and carry no information
  - A non leaf node with *k* children contains *k-1* keys

2-3-4 Tree

- A **B-tree** of *minimum degree* ***t=2*** is a tree that satisfies:
    - Every node has at most ***4*** children
    - Every node, except for the root and the leaves, has at least ***2*** children
    - The root has at least 2 children (unless it is a leaf)
    - All leaves appear on the same level, and carry no information
    - A non leaf node with k children contains k-1 keys

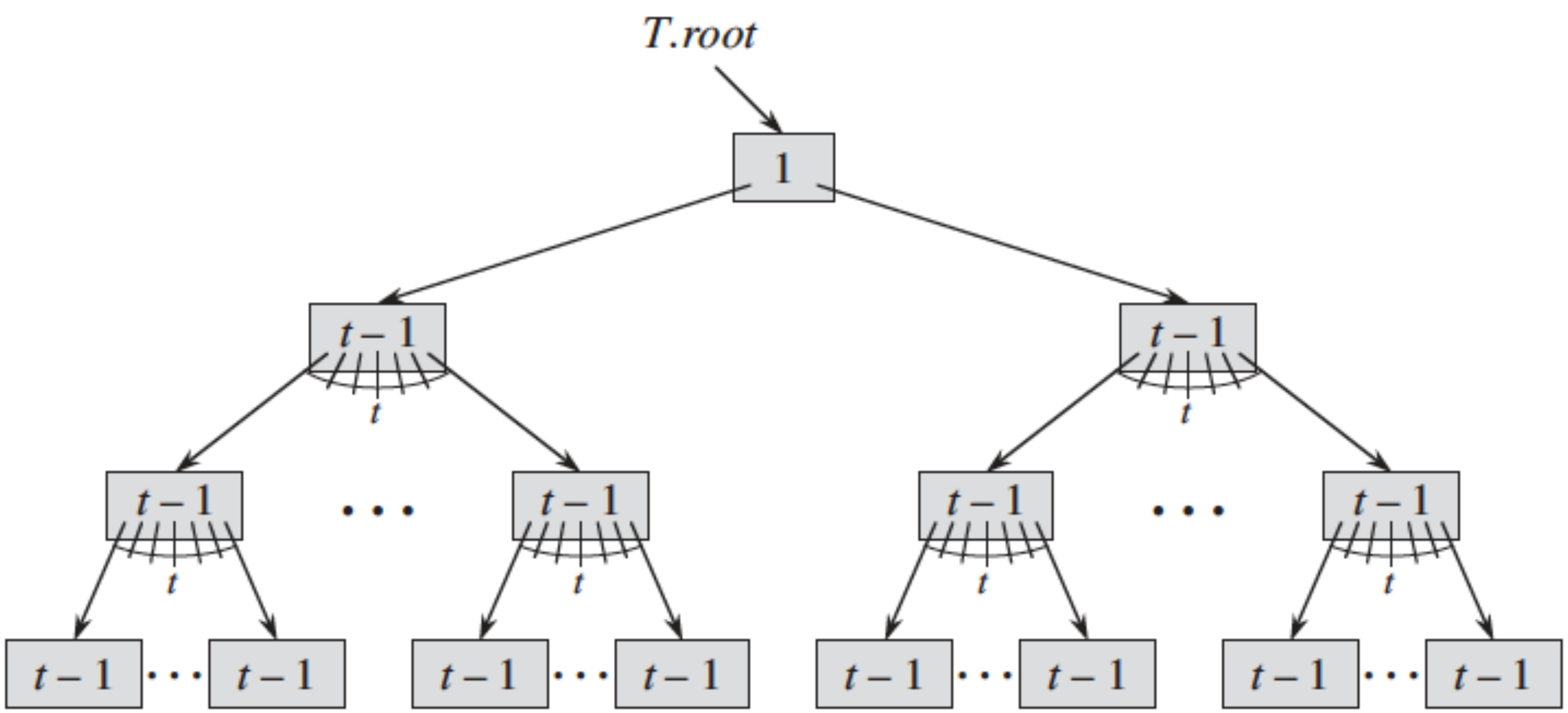
B-Tree Operations

- Search: $O(t \log_t n)$
- Insert: if a node gets too big, we split it into two nodes
- Delete: if a node gets too small, we combine two nodes

Balance is achieved from the **top** of the tree

- since the height is only modified when the root splits or merges

Operation Costs



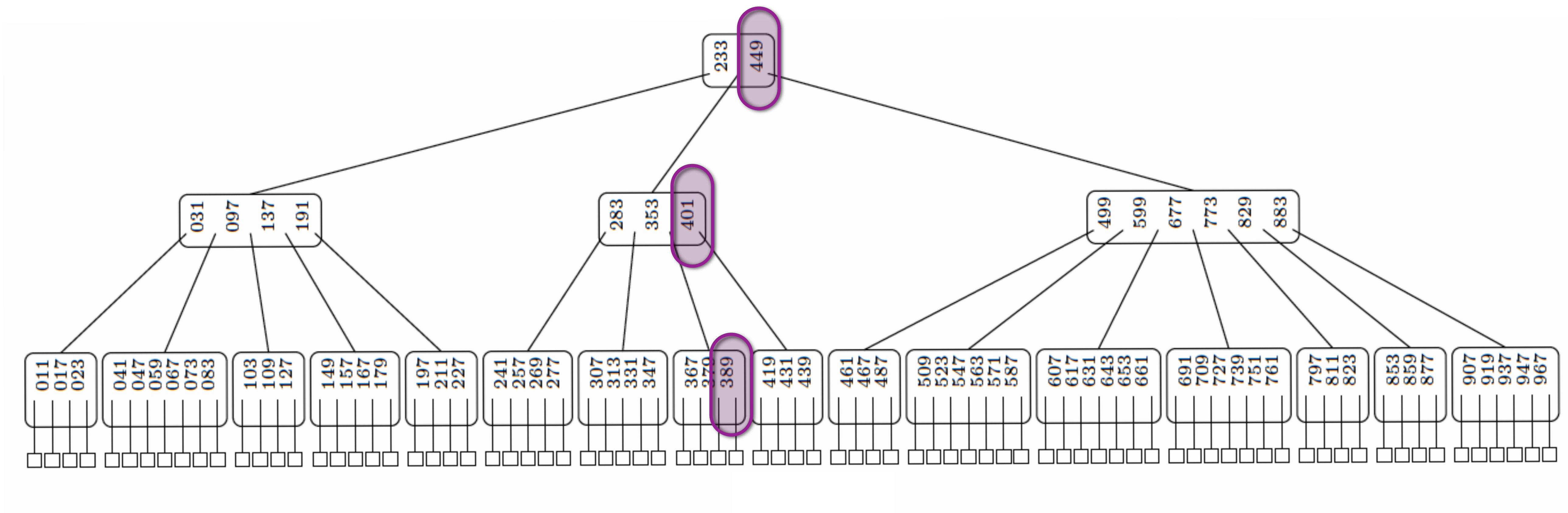
depth	number of nodes
0	1
1	2
2	$2t$
3	$2t^2$

$$
\begin{aligned}
n &\geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} \\
&= 1 + 2(t-1)\left(\frac{t^h - 1}{t-1}\right) \\
&= 2t^h - 1 .
\end{aligned}
$$

$$h \leq \log_t \frac{n+1}{2}$$

Search

- Form a simple path downward from the root of the tree

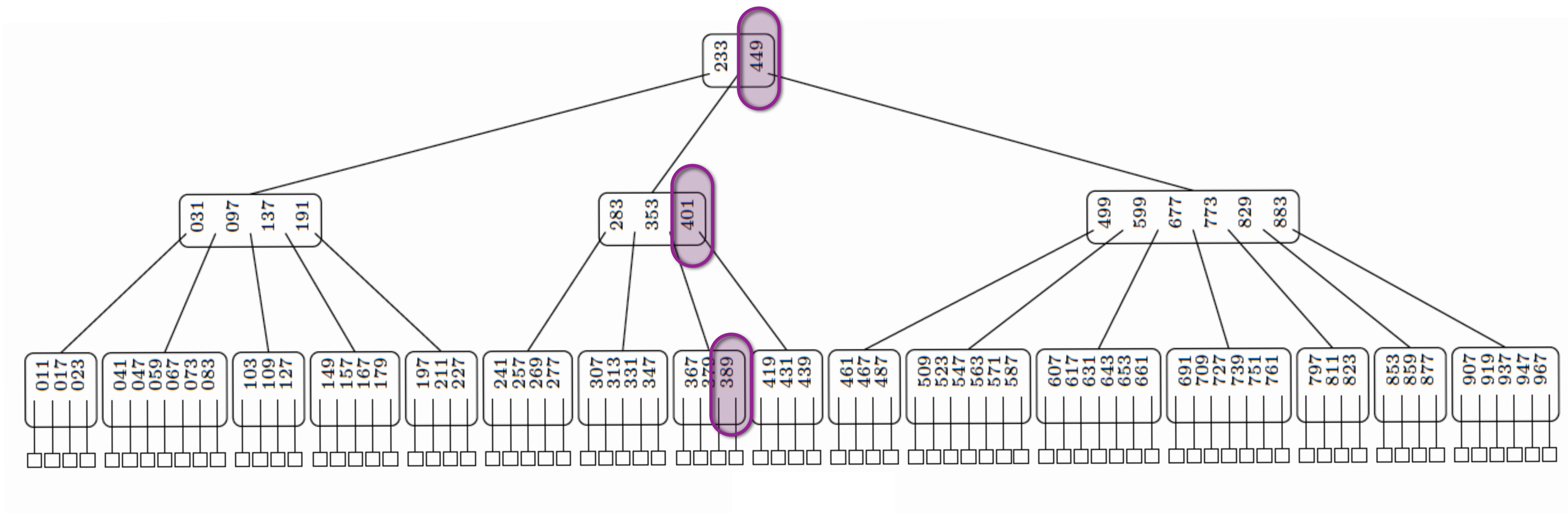


Search

- **Form a simple path downward from the root of the tree**
  - Recursively, starting at the root
    - Look for the appropriate position in the node
      - **if** the key is found, **return** the key
      - **else**
        - **if** the node is a leaf, **return** NIL
        - **else** continue recursively checking the appropriate child

Search

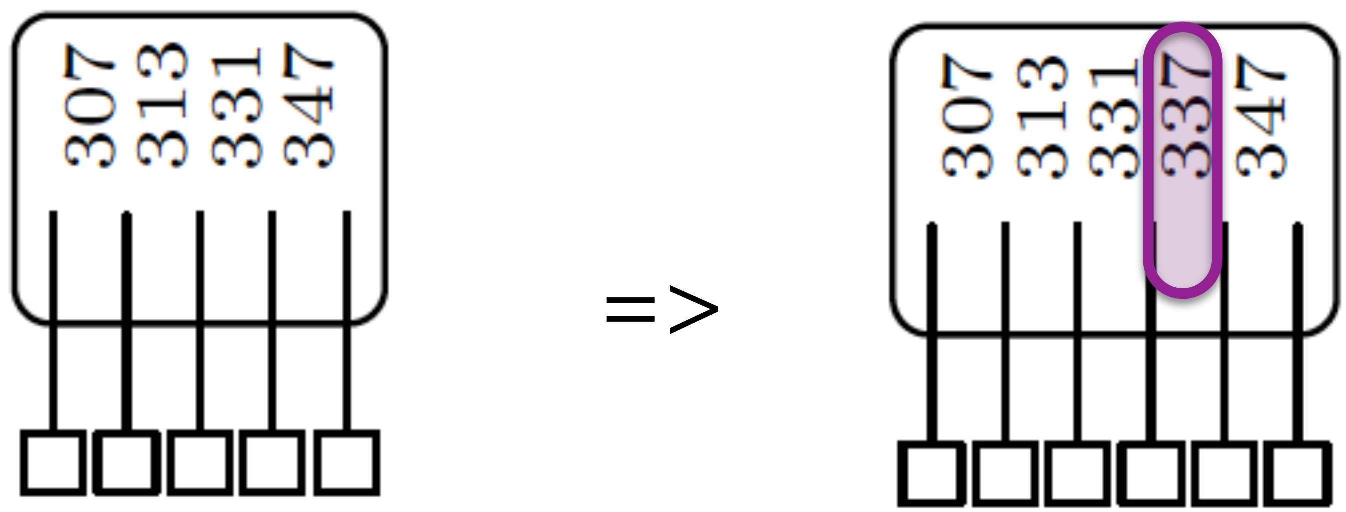
- Form a simple path downward from the root of the tree

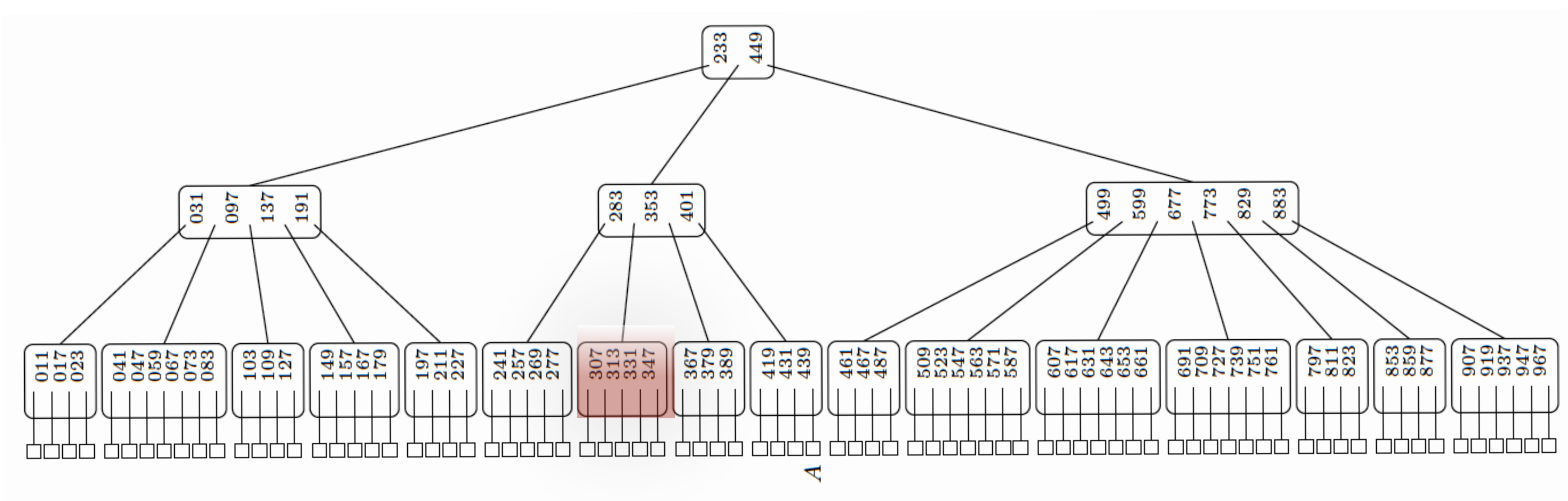


Insertion

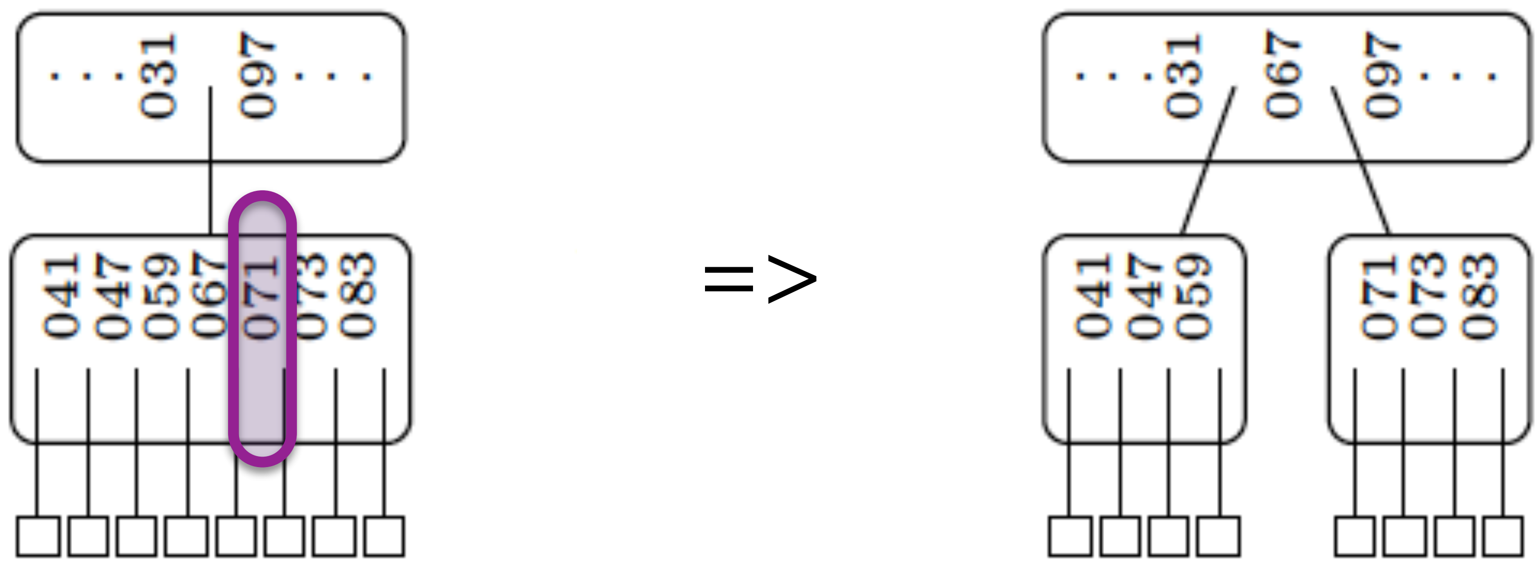
- Search from the root the proper leaf for insertion
- Do insertion
- If the leaf is too large:
  overflow: redistribution of keys to restore balance
  - Split the leaf in two and put the middle key in the parent node
  - Recursively split parents, putting an additional key in the parent node, until there is no need to split or we reach the root. If the node has no parent (root), create a new root above the node

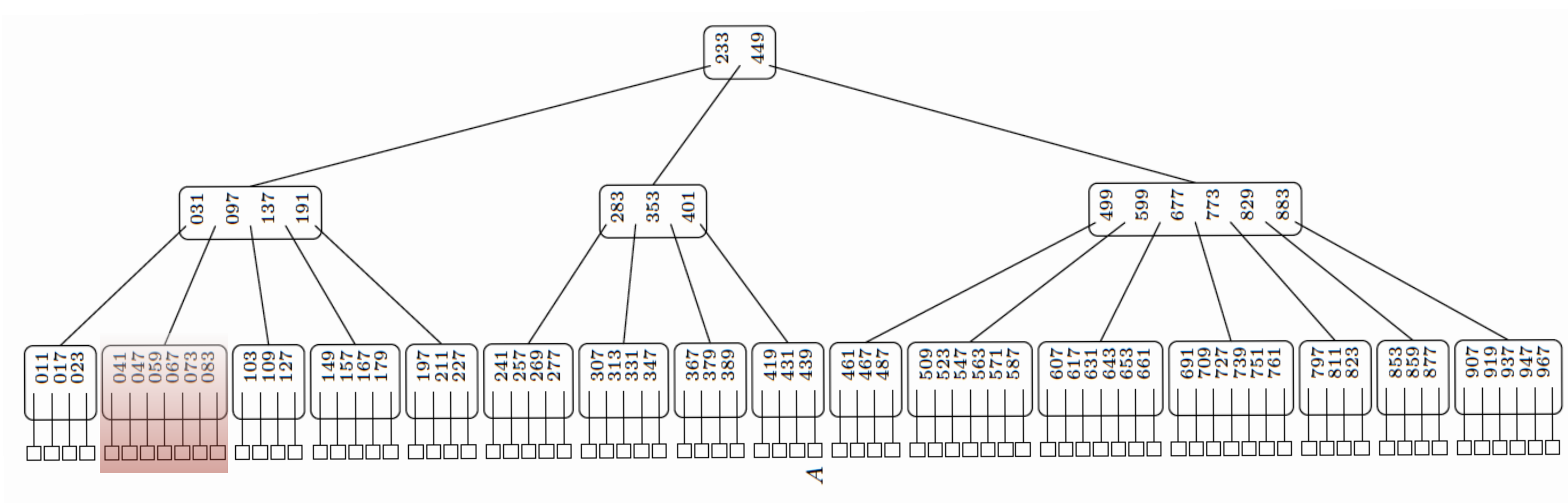
Insertion (337)





Insertion (071)

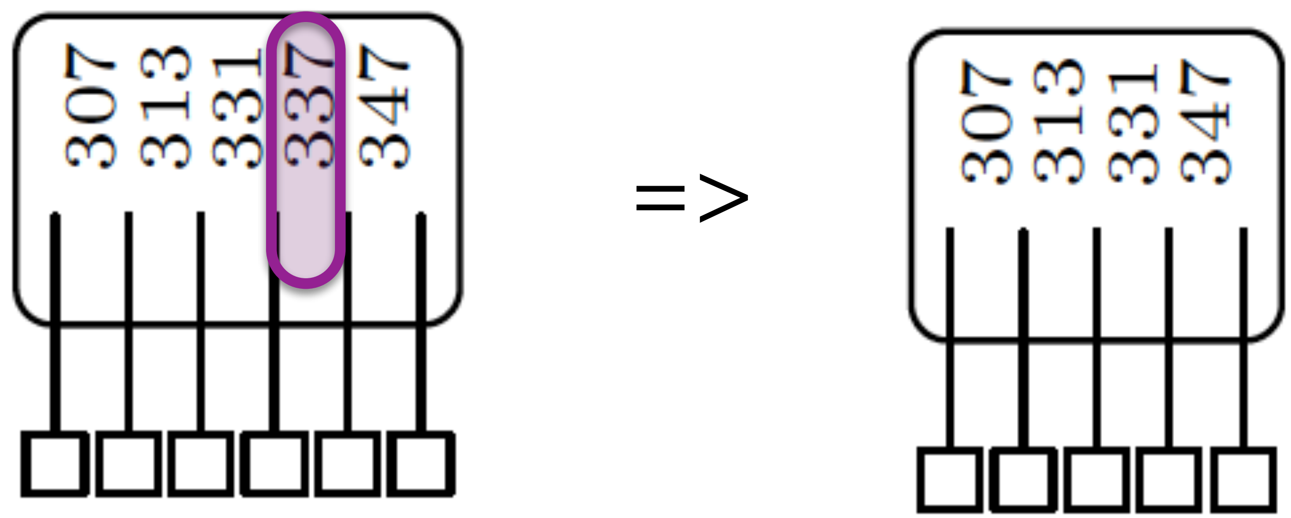


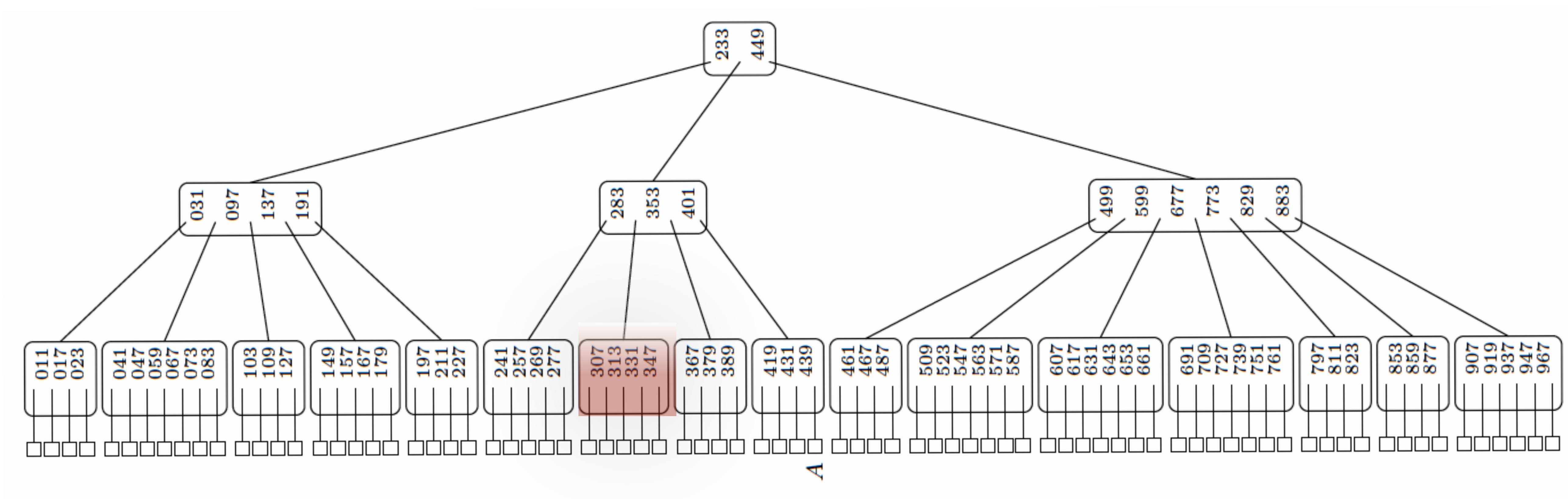


Deletion

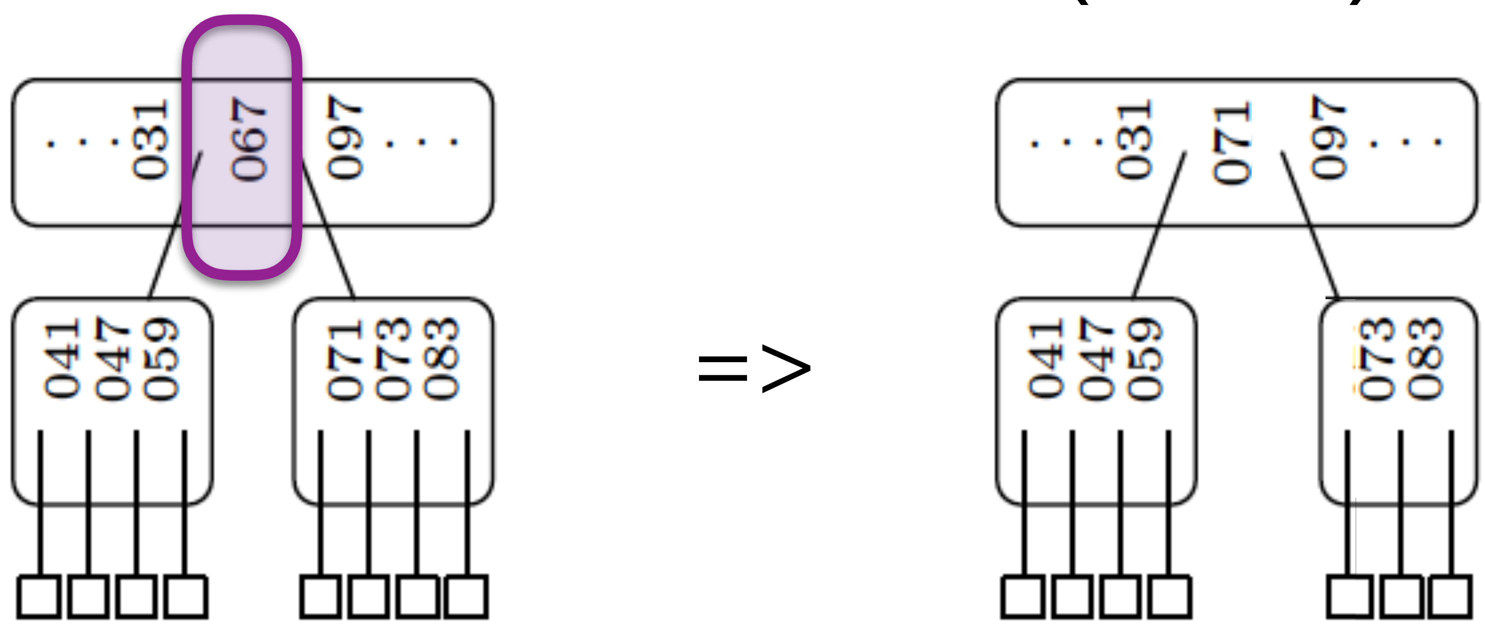
- A search proceeds from the root to locate the proper node
  - If the key resides in a leaf, remove it
  - If the key resides in a non-leaf node
    - an adjacent key (previous key or next key) is found and swapped into the vacated position
    - Remove the swapped key stored at a leaf

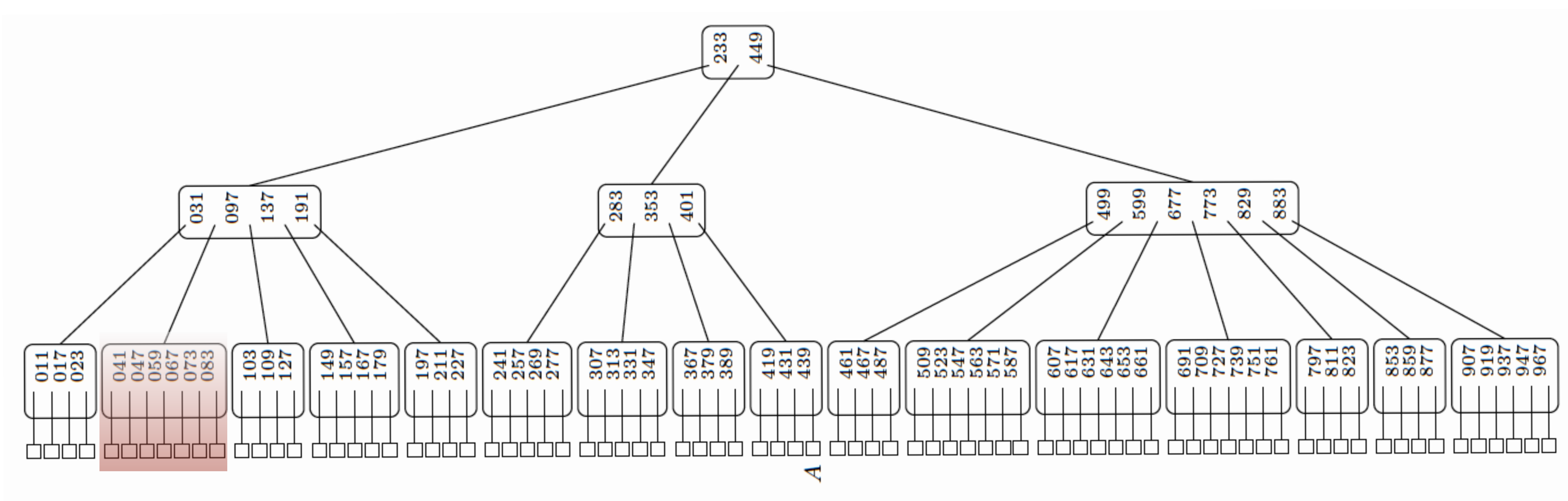
Deletion (337)





Deletion (067)

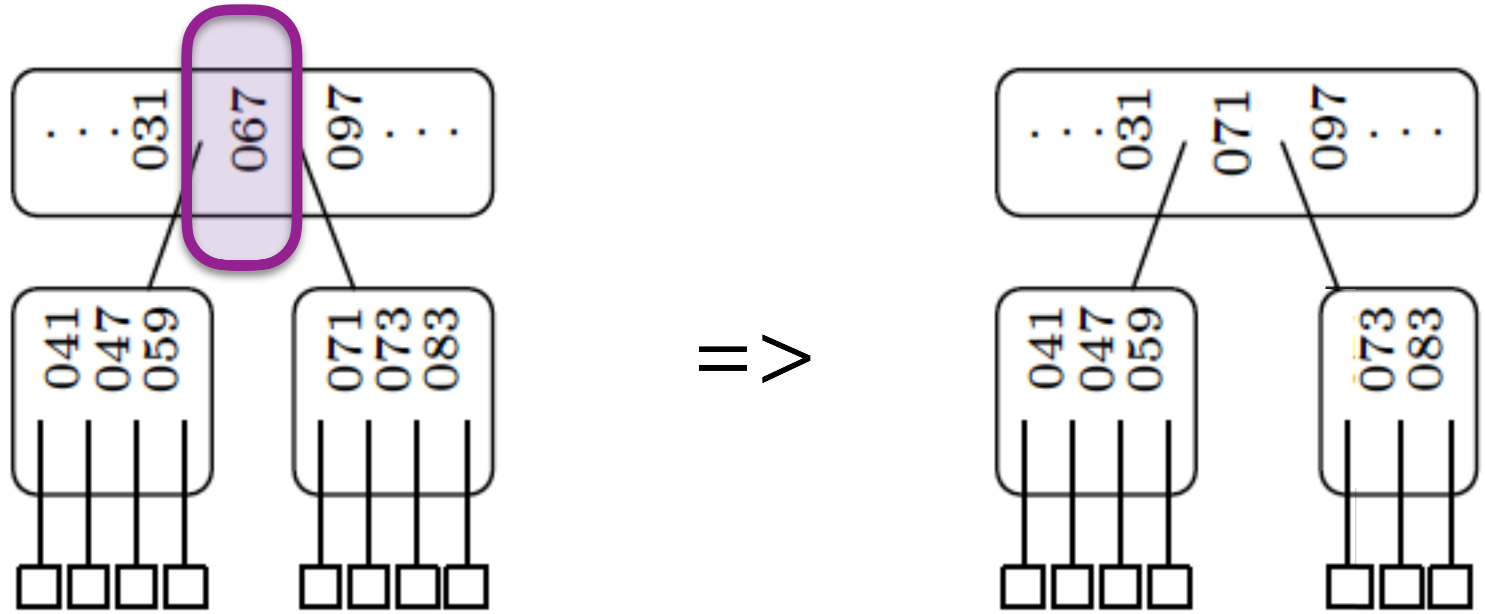


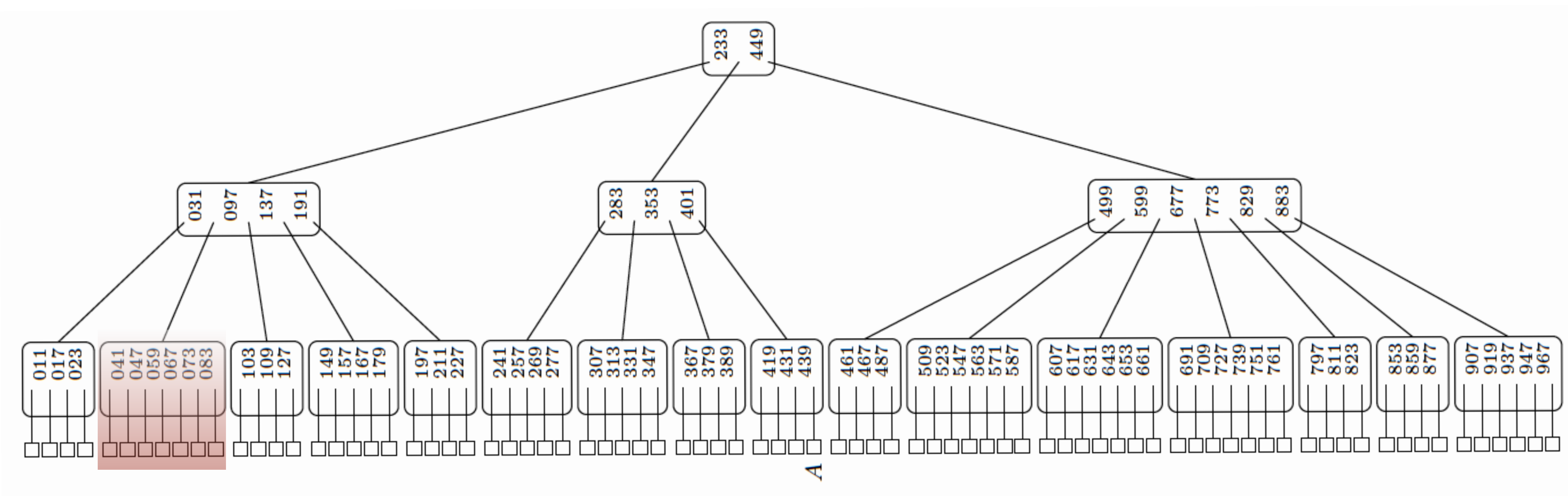


Deletion

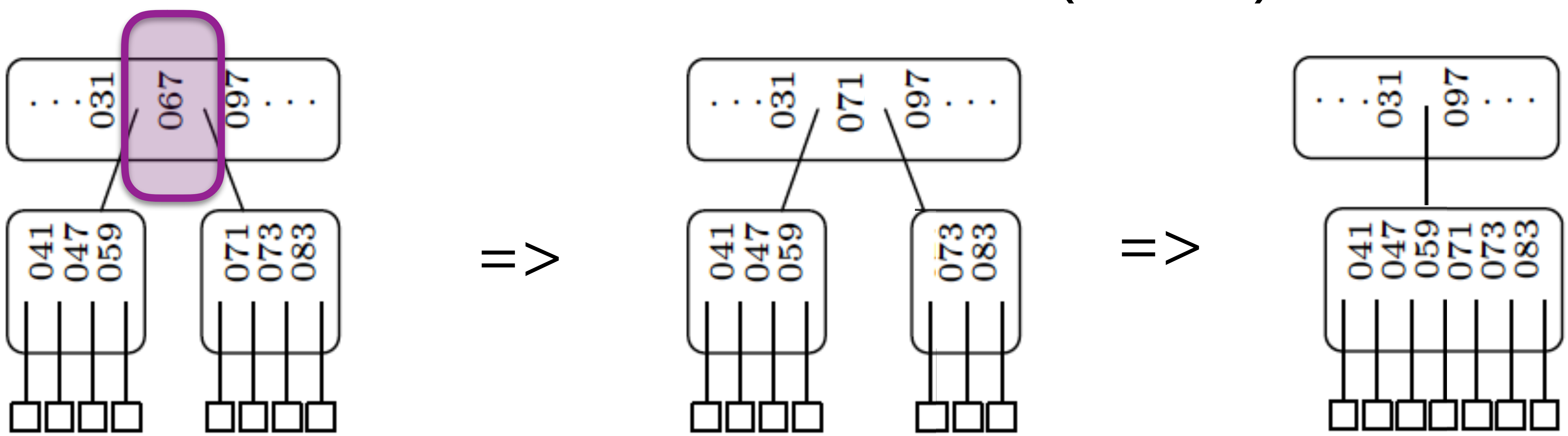
- If the node has not enough keys **underflow: redistribution of keys to restore balance**,
  - Keys are obtained from a neighbouring subtrees if it exists and if this does not cause underflow
    - If this is not possible, concatenation (inverse of splitting)

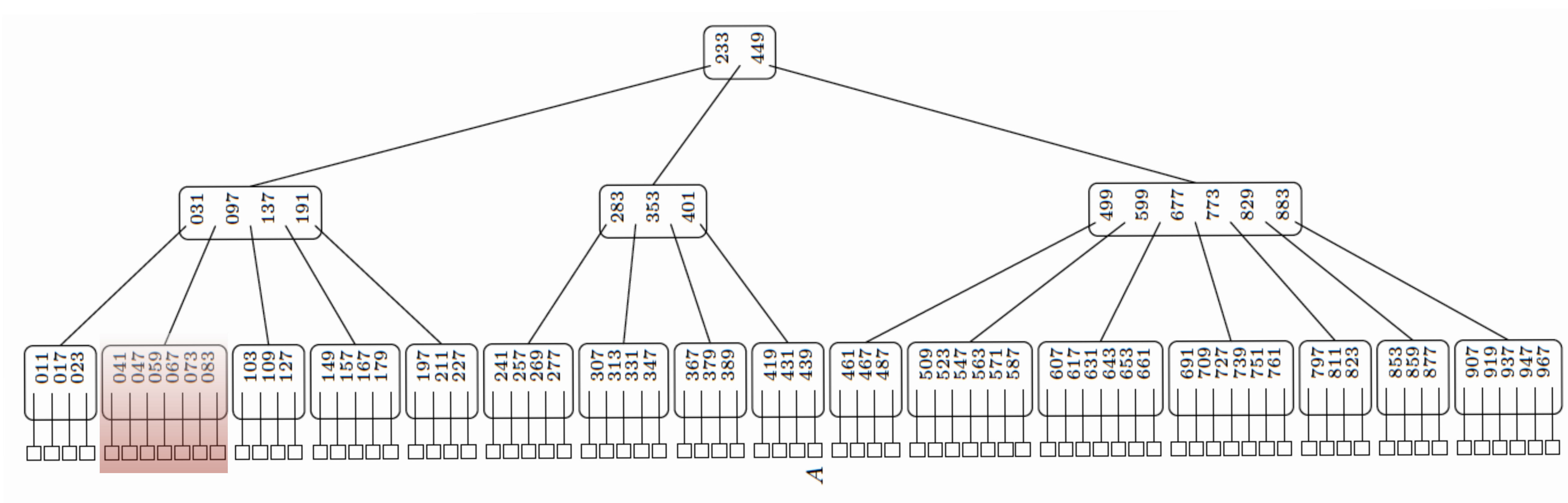
Deletion (067)





Deletion (067)





Applications

- Databases
- Filesystems
- File indexes

B-Tree Summary

- Balanced Tree designed to work with storage devices
  - Search, Update in time O(lg(*n*))
- Insert: if a node gets too big, we split it into two nodes
- Delete: if a node gets too small, we combine two nodes

Balance is achieved from the **top** of the tree

- since the height is only modified when the root splits or merges

Exercise

Insert the following elements in a 2-3-4 tree:

6   10   15   4   13   14   7   3   8   5   9   11   12

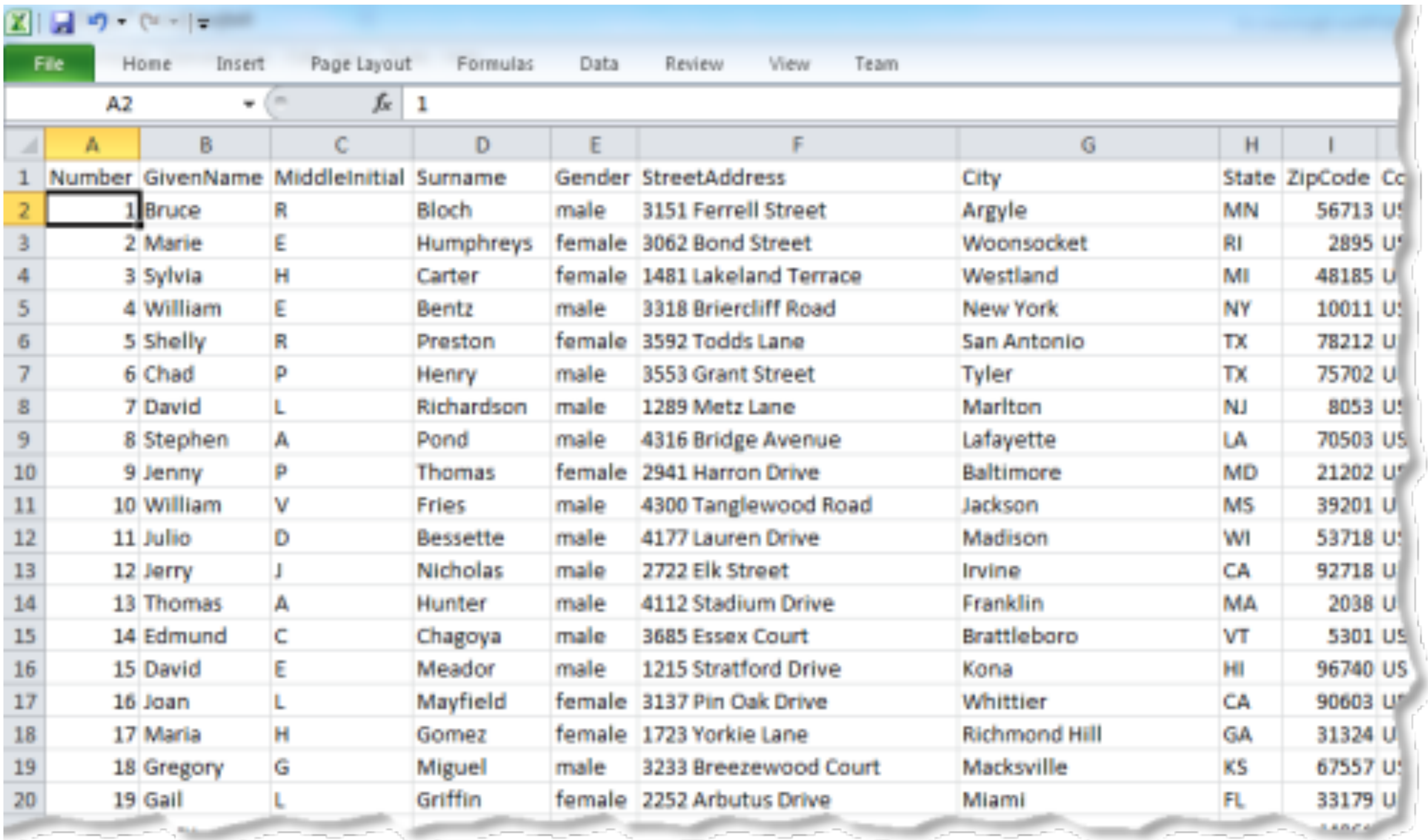
What is missing?

DBMS

DBMS

- A Database Management System (DBMS) is a software package designed to store and manage databases
  - Data independence and efficient access.
  - Reduced application development time.
  - Data integrity and security.
  - Uniform data administration.
  - Concurrent access, recovery from crashes.

Spreadsheet


File Home Insert Page Layout Formulas Data Review View Team

A2 fx 1

	A	B	C	D	E	F	G	H	I	
1	Number	GivenName	MiddleInitial	Surname	Gender	StreetAddress	City	State	ZipCode	Co
2	1	Bruce	R	Bloch	male	3151 Ferrell Street	Argyle	MN	56713	US
3	2	Marie	E	Humphreys	female	3062 Bond Street	Woonsocket	RI	2895	US
4	3	Sylvia	H	Carter	female	1481 Lakeland Terrace	Westland	MI	48185	U
5	4	William	E	Bentz	male	3318 Briercliff Road	New York	NY	10011	US
6	5	Shelly	R	Preston	female	3592 Todds Lane	San Antonio	TX	78212	U
7	6	Chad	P	Henry	male	3553 Grant Street	Tyler	TX	75702	U
8	7	David	L	Richardson	male	1289 Metz Lane	Marlton	NJ	8053	US
9	8	Stephen	A	Pond	male	4316 Bridge Avenue	Lafayette	LA	70503	US
10	9	Jenny	P	Thomas	female	2941 Harron Drive	Baltimore	MD	21202	US
11	10	William	V	Fries	male	4300 Tanglewood Road	Jackson	MS	39201	U
12	11	Julio	D	Bessette	male	4177 Lauren Drive	Madison	WI	53718	US
13	12	Jerry	J	Nicholas	male	2722 Elk Street	Irvine	CA	92718	U
14	13	Thomas	A	Hunter	male	4112 Stadium Drive	Franklin	MA	2038	U
15	14	Edmund	C	Chagoya	male	3685 Essex Court	Brattleboro	VT	5301	US
16	15	David	E	Meador	male	1215 Stratford Drive	Kona	HI	96740	US
17	16	Joan	L	Mayfield	female	3137 Pin Oak Drive	Whittier	CA	90603	US
18	17	Maria	H	Gomez	female	1723 Yorkie Lane	Richmond Hill	GA	31324	U
19	18	Gregory	G	Miguel	male	3233 Breezewood Court	Macksville	KS	67557	US
20	19	Gail	L	Griffin	female	2252 Arbutus Drive	Miami	FL	33179	U

Data Models

- A *data model* is a collection of concepts for describing data.

- A *schema* is a description of a particular collection of data, using the a given data model.

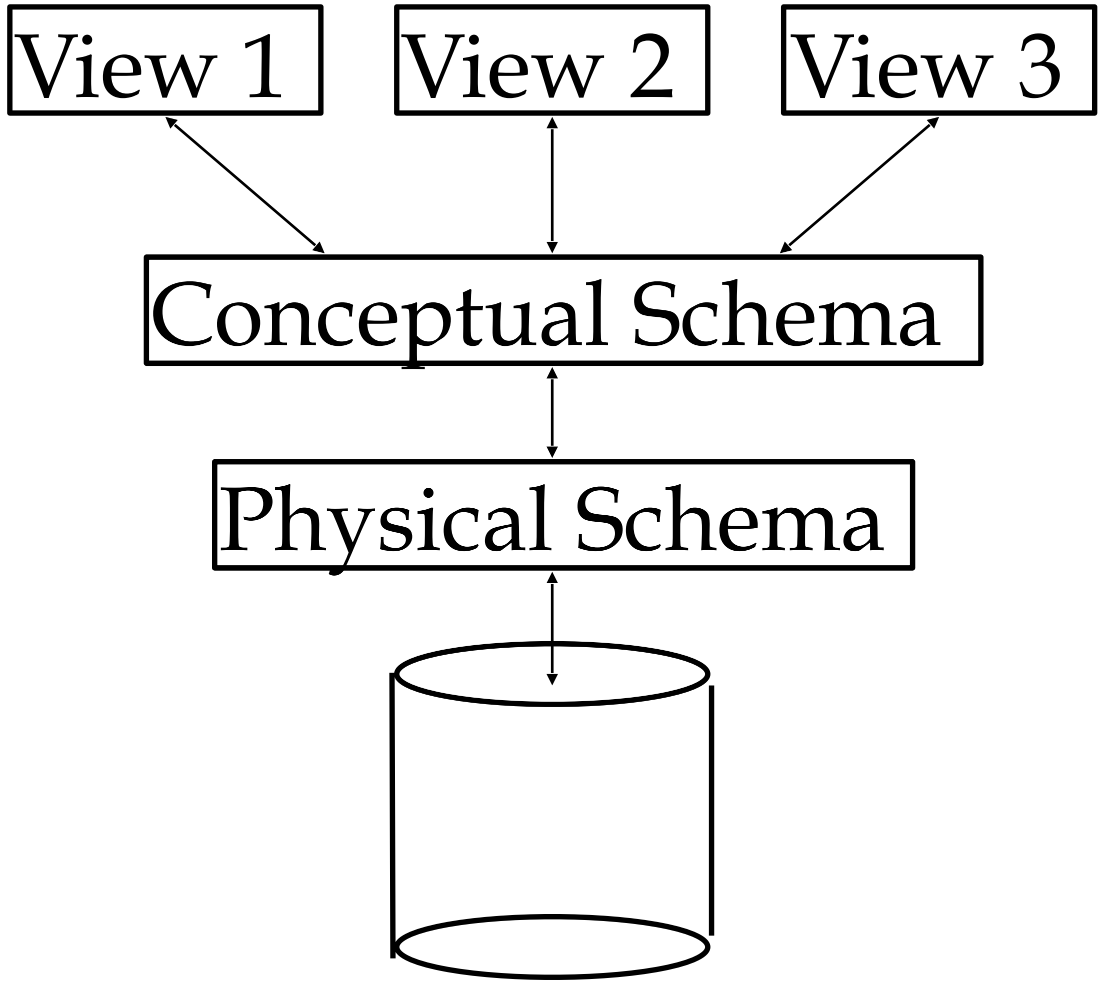
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a *schema*, which describes the columns, or fields.

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Levels of Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



☛ *Schemas are defined using DDL; data is modified/queried using DML.*

Example: University Database

- Conceptual schema:
  Students(sid: string, name: string, login: string,
  age: integer, gpa:real)
  Courses(cid: string, cname:string, credits:integer)
  Enrolled(sid:string, cid:string, grade:string)

- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.

- External Schema (View):
  Course_info(cid:string,enrollment:integer)

Data Independence

- Applications insulated from how data is structured and stored.
- *Logical data independence*: Protection from changes in *logical* structure of data.
- *Physical data independence*: Protection from changes in *physical* structure of data.

  ☛ *One of the most important benefits of using a DBMS!*

Concurrency Control

- Concurrent execution of user programs is essential for good DBMS performance.
  - Because disk accesses are frequent, and relatively slow, it is important to keep the cpu humming by working on several user programs concurrently.

- Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.

- DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

Transaction: An Execution of a DB Program

- Key concept is *transaction*, which is an *atomic* sequence of database actions (reads/writes).

- Each transaction, executed completely, must leave the DB in a *consistent state* if DB is consistent when the transaction begins.
  - Users can specify some simple *integrity constraints* on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user's responsibility!

Scheduling Concurrent Transactions

- DBMS ensures that execution of {T1, ... , Tn} is equivalent to some *serial* execution T1' ... Tn'.
  - Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. All locks are released at the end of the transaction. (Strict 2PL locking protocol.)
  - Idea: If an action of Ti (say, writing X) affects Tj (which perhaps reads X), one of them, say Ti, will obtain the lock on X first and Tj is forced to wait until Ti completes; this effectively orders the transactions.
  - What if Tj already has a lock on Y and Ti later requests a lock on Y? (Deadlock!) Ti or Tj is aborted and restarted!

Relational Model

Relational Database: Definitions

- Relational database: a set of relations
- Relation: made up of 2 parts:
  - Instance : a table, with rows and columns. #Rows = cardinality, #fields = degree / arity.
  - Schema : specifies name of relation, plus name and type of each column.
    - E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).
- Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, degree = 5, all rows distinct
- Do all columns in a relation instance have to be distinct?

Relational Query Languages

- A major strength of the relational model: supports simple, powerful *querying* of data.
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - The key: precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

The SQL Query Language

- Developed by IBM (system R) in the 1970s
- Need for a standard since it is used by many vendors
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, current standard)
  - SQL-99 (major extensions)

The SQL Query Language

- To find all 18 year old students, we can write:

```
SELECT *
FROM Students S
WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

Querying Multiple Relations

- What does the following query compute?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade="A"
```

Given the following instance of Enrolled (is this possible if the DBMS ensures referential integrity?):

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

we get:

S.name	E.cid
Smith	Topology112

Creating Relations in SQL

- Creates the Students relation. Observe that the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.
- As another example, the Enrolled table holds information about courses that students take.

```
CREATE TABLE Students
        (sid: CHAR(20),
         name: CHAR(20),
         login: CHAR(10),
         age: INTEGER,
         gpa: REAL)
```

```
CREATE TABLE Enrolled
        (sid: CHAR(20),
         cid: CHAR(20),
         grade: CHAR(2))
```

Destroying and Altering Relations

DROP TABLE Students

- Destroys the relation Students. The schema information *and* the tuples are deleted.

ALTER TABLE Students
ADD COLUMN firstYear: integer

- The schema of Students is altered by adding a new field; every tuple in the current instance is extended with a *null* value in the new field.

Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
FROM Students S
WHERE S.name = 'Smith'
```

☛ *Powerful variants of these commands are available; more later!*

Integrity Constraints (ICs)

- IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- A *legal* instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances.
- If the DBMS checks ICs, stored data is more faithful to real-world meaning.
  - Avoids data entry errors, too!

Primary Key Constraints

- A set of fields is a *key* for a relation if :
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.
- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

Foreign Keys, Referential Integrity

- *Foreign key* : Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Can you name a data model w/o referential integrity?
    - Links in HTML!

Enforcing Referential Integrity

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null*, denoting *\`unknown'* or *\`inapplicable'*.)
- Similar if primary key of Students tuple is updated.

Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

```
CREATE TABLE Enrolled
  (sid CHAR(20),
   cid CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students
        ON DELETE CASCADE
        ON UPDATE SET DEFAULT )
```

Relational Algebra

Relational Query Languages

- Query languages: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages != programming languages!
  - QLs not expected to be "Turing complete".
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

- Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

❶ Relational Algebra: More operational, very useful for representing execution plans.

❷ Relational Calculus: Lets users describe what they want, rather than how to compute it. (Non-operational, declarative.)

☛ *Understanding Algebra & Calculus is key to*
☛ *understanding SQL, query processing!*

Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in SQL

Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation, assume that names of fields in query results are `inherited' from names of fields in query input relations.

Reserves

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

Sailors

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
- Additional operations:
  - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

age
35.0
55.5

$$\pi_{age}(S2)$$

Selection

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

- Selects rows that satisfy *selection condition*.
- No duplicates in result!
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

$$\sigma_{rating>8}(S2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - `Corresponding' fields have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$$S1 \cup S2$$

sid	sname	rating	age
22	dustin	7	45.0

$$S1 - S2$$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$$S1 \cap S2$$

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one field per field of S1 and R1, with field names `inherited' if possible.
  - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

☛ *Renaming operator*: $\rho\ (C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

Joins

- *Condition Join*: $R \bowtie_c S = \sigma_c (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

Joins

- *Equi-Join*: A special case of condition join where the condition c contains only ***equalities***.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

- *Result schema* similar to cross-product, but only one copy of fields for which equality is specified.
- *Natural Join*: Equijoin on *all* common fields.

Division

- Not supported as a primitive operator, but useful for expressing queries like:

  *Find sailors who have reserved **all** boats*.

- Let A have 2 fields, x and y; B have only field y:
  - $A/B = \left\{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B \right\}$
  - i.e., ***A/B* contains all *x* tuples (sailors) such that for *every* *y* tuple (boat) in *B*, there is an *xy* tuple in *A*.**
  - *Or*: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, the x value is in A/B.

- In general, x and y can be any lists of fields; y is the list of fields in B, and $x \cup y$ is the list of fields of A.

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)
- *Idea*: For *A/B*, compute all *x* values that are not `disqualified' by some *y* value in *B*.
  - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*.

Disqualified *x* values: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A)$ – all disqualified tuples

Exercises

- Tables:

  Sailors: sid, sname, rating, age

  Reserves: sid, bid, day

  Boats: bid, color

- Find names of sailors who’ve reserved boat #103

- Find names of sailors who’ve reserved a red boat

- Find sailors who’ve reserved a red or a green boat

- Find the names of sailors who’ve reserved all boats

Find names of sailors who've reserved boat #103

- Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie Sailors)$

- Solution 2: $\rho\ (Temp1, \sigma_{bid=103} \text{Reserves})$

  $\rho\ (Temp2, Temp1 \bowtie Sailors)$

  $\pi_{sname}(Temp2)$

- Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie \text{Reserves} \bowtie Sailors)$$

- A more efficient solution:

$$\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'} Boats) \bowtie \text{Res}) \bowtie Sailors)$$

☛ *A query optimizer can find this given the first solution!*

Find sailors who've reserved a red or a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie \text{Re}serves \bowtie Sailors)$$

- Can also define Tempboats using union! (How?)
- What happens if $\vee$ is replaced by $\wedge$ in this query?

Find sailors who've reserved a red <u>and</u> a green boat

- Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho\ (Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie \text{Re}serves))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie \text{Re}serves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Find the names of sailors who've reserved all boats

- Uses division; schemas of the input relations to / must be carefully chosen:

$$\rho\ (Tempsids, (\pi_{sid,bid} \text{Re}serves) \ / \ (\pi_{bid} Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$\ldots\ldots \quad / \ \pi_{bid}(\sigma_{bname='Interlake'} Boats)$$

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.