ER - DIAGRAM

COURSE 1: Databases

DBMS (Database Management System)

- System designed to define and manipulate data.
  - Storage.
  - Retrieval.
  - Updates.

DBMS (Database Management System)

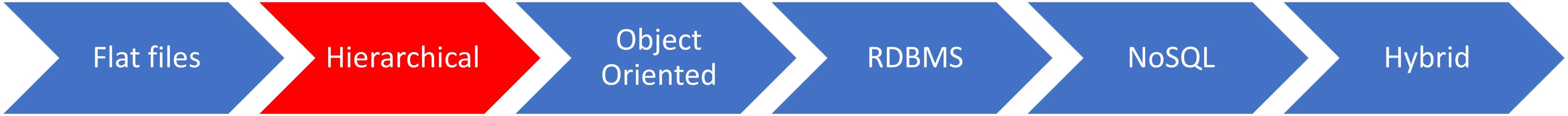
- Avoid redundancy, inconsistency.
- Concurrent data access.
- Provides security and recovery.
- Declarative language to manipulate, query, define and control data.
- DDL, DML, DCL.
- Data dictionary: database providing info about database structure.

DBMS (Database Management System)

Flat files | Hierarchical | Object Oriented | RDBMS | NoSQL | Hybrid

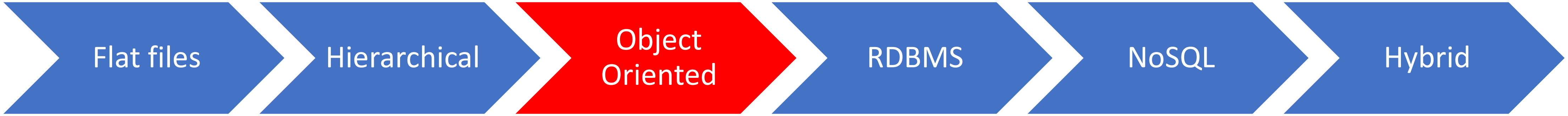
- Text database, example CSV format.
- Implemented in 1970 (IBM).
- File = table with a single record on each line.
- Read, store and send.
- Simple structure.
- Inefficient: slow, duplicated values, hard to update etc.

DBMS (Database Management System)



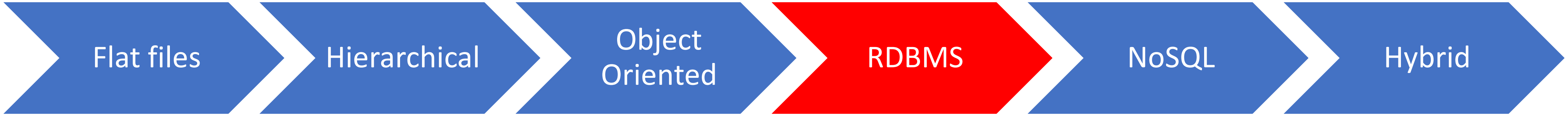
- Tree structure, examples: file system, Windows Registry
- IBM Information Management System (IMS)
- XML, XAML
- Used in mainframe era.
- Rigid structure.
- Only *One-to-many* relationship.
- Traversing very easy, moving a node difficult

DBMS (Database Management System)



- Hybrid relation + objects =>> tables of objects.
- Realm database for Android/IoS: classes used as schema definition, alternative for SQLite.
- Next: MongoDB Realm.

DBMS (Database Management System)



- Transaction oriented systems (financial transactions).
- ACID: Atomicity, consistency, isolation, durability.
- Suitable for structured data.

DBMS (Database Management System)

Flat files

Hierarchical

Object Oriented

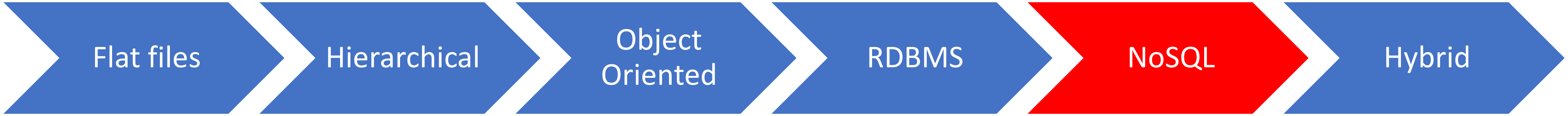
RDBMS

NoSQL

Hybrid

- RDBMS hard to scale (only scale vertically, not horizontally).
- RDB Restrictive schemas =>> flexible structure.
- The state of the database can change.
- !!! availability, scalability, performance
- Sharding: distribute data on different servers

DBMS (Database Management System)



- Cloud and bigdata.
- BASE (Basically Available, Soft state, Eventually consistent)
- Types:
  - key-value: Redis
  - Document: Mongo
  - Column: Apache Cassandra
  - Graph: Neo4j

Sql or NoSQL

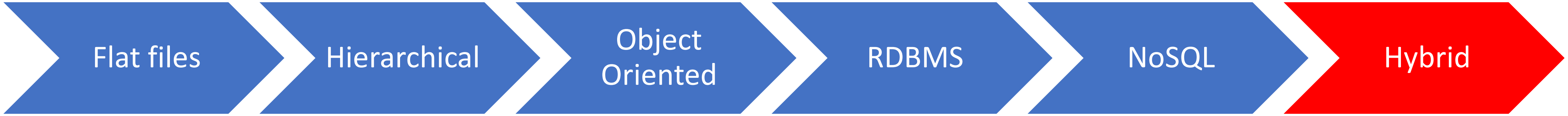
Relational

- Vertical scalability
- ACID
- pre-defined schema
- SQL language
- Normalized data

NoSql

- Horizontal scalabitily
- BASE
- Flexible schema
- No standard
- Collections, redundancy

DBMS (Database Management System)

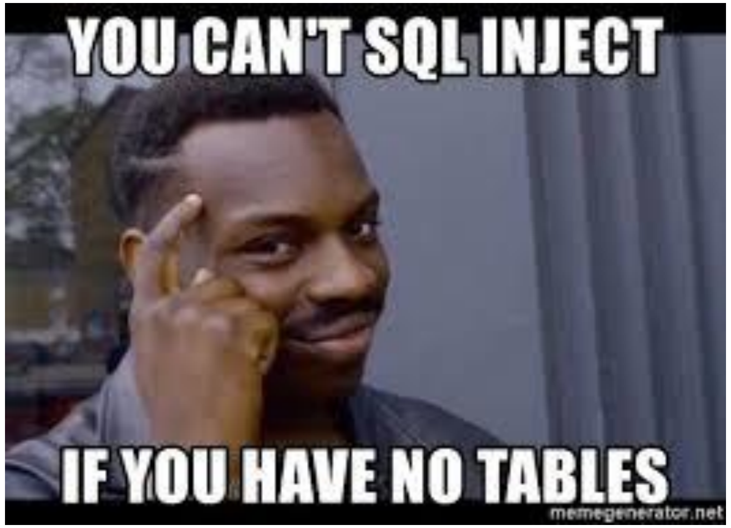


- Integration of Relational and NoSQL databases.
- Integration of in-memory DB and on-disk DB
- Altibase, Orient DB

Course roadmap

- Database design
- SQL
- NoSql
- … & other topics …

Course roadmap



ER diagram

- Visual representation of the ER model.
- Describes the logical structure of the (relational) database.
- Proposed by Chen in 1971.
- Easy to translate into relational tables.
- High-level design.
- Suitable for structured systems.

ER - Diagram

Entity Relationship model

ER - Diagram

- Visual representation of the ER conceptual data model.
- Describes the structure of the (relational) database.
- Not linked to the implementation or hardware.

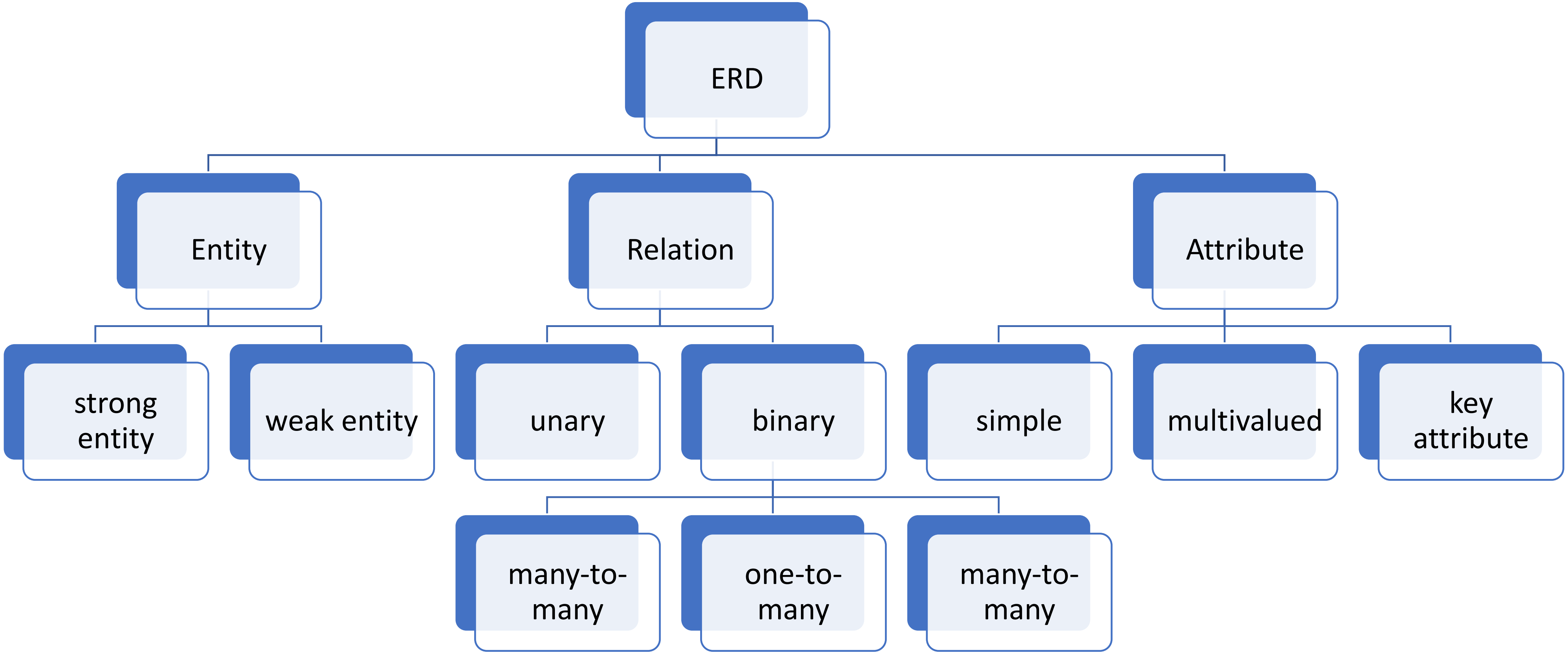
- Peter Chen developed ERDs in 1976.

ER - Diagram

- User story/requirement analysis ➔ **ER** ➔relational database schema.
- Easy to translate into relational tables.

- High-level design.
- Suitable for structured systems.

ERD - components



ER - Diagram

ENTITY

person, place, activity, event, concept, real world object etc.

usually a noun

RELATION

ATTRIBUTE

ER - Diagram

ENTITY

person, place, activity, event, concept, real world object etc.

usually a noun

RELATION

links entities (unary, binary, ternary).

usually a verb

ATTRIBUTE

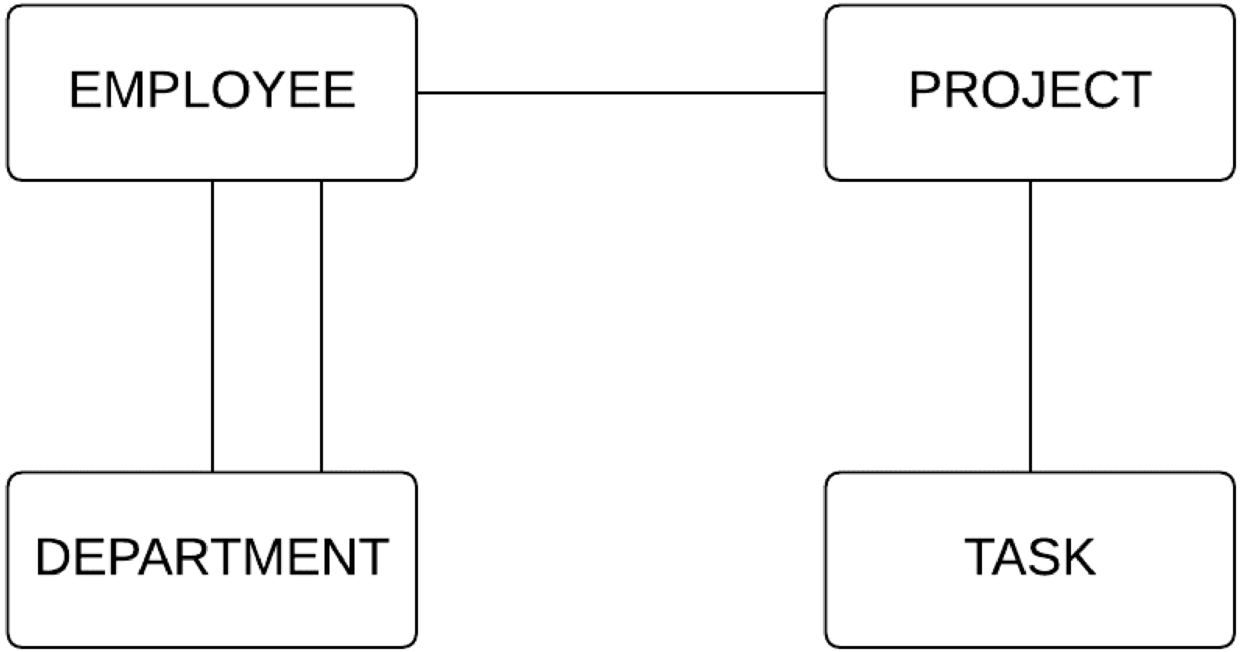
ER - Diagram

ENTITY — person, place, activity, event, concept, real world object etc.
usually a noun

RELATION — links entities (unary, binary, ternary).
usually a verb

ATTRIBUTE — describe entities or relations

Entities



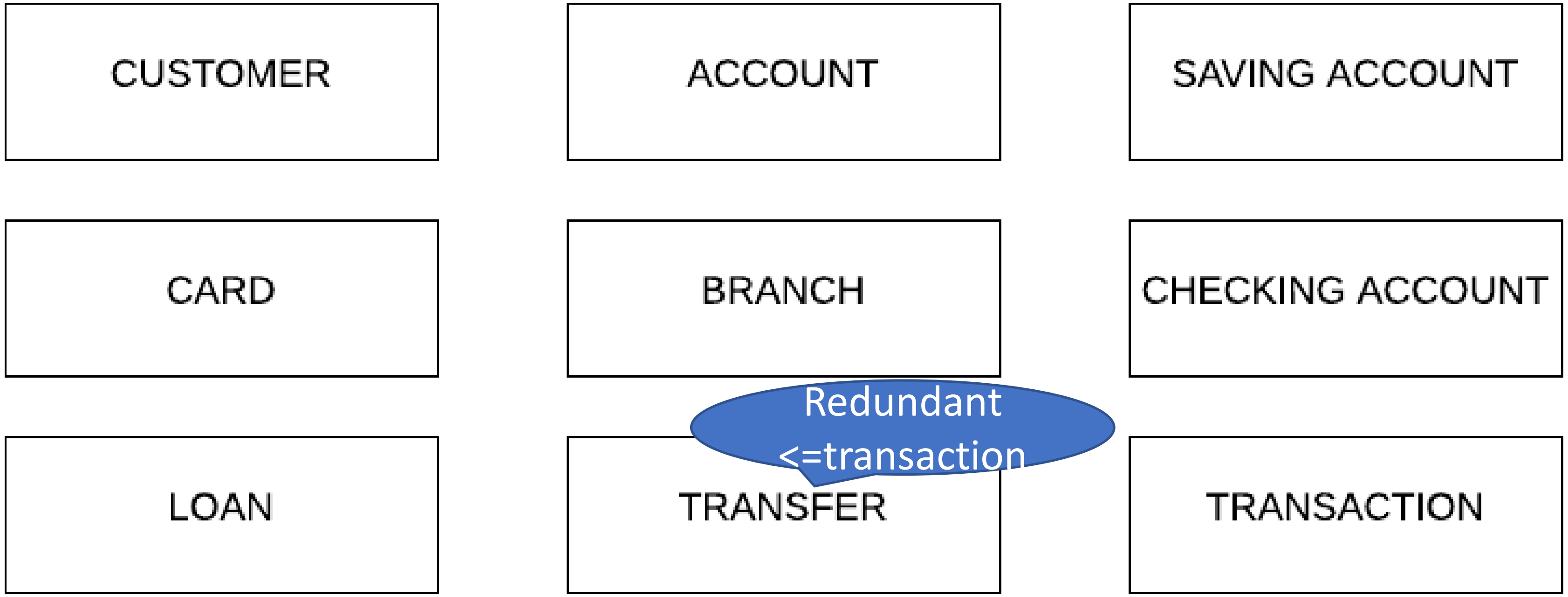
Banking (1)Entities

- A customer opens a saving account or a checking account, at a bank branch. He may also access loans. For each checking account he has a card. Periodically he may withdraw money from his account or partially pay his loans. He may also transfer money from one account to another.

Entities

- Unique names, uppercase characters
- Graphical representation: rectangles

- Relational database: entity ➔ table (line & columns)
- Primary key: attribute or group of attributes that uniquely identifies an entity instance


CUSTOMER
ACCOUNT
SAVING ACCOUNT
CARD
BRANCH
CHECKING ACCOUNT
LOAN
TRANSFER
TRANSACTION
Redundant
<=transaction

Primary key

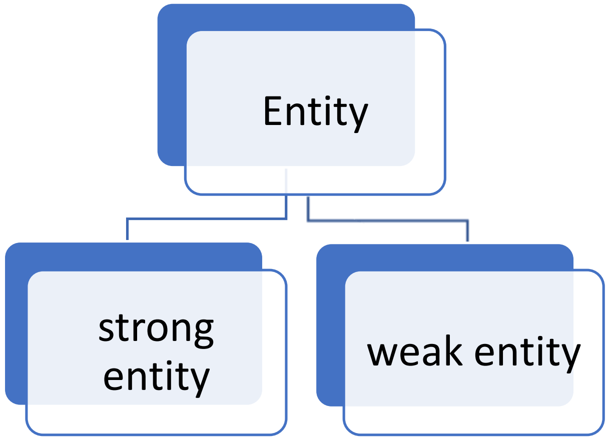
- Unique identifier
- Must be known at any moment
- Simple
- No ambiguities
- Immutable

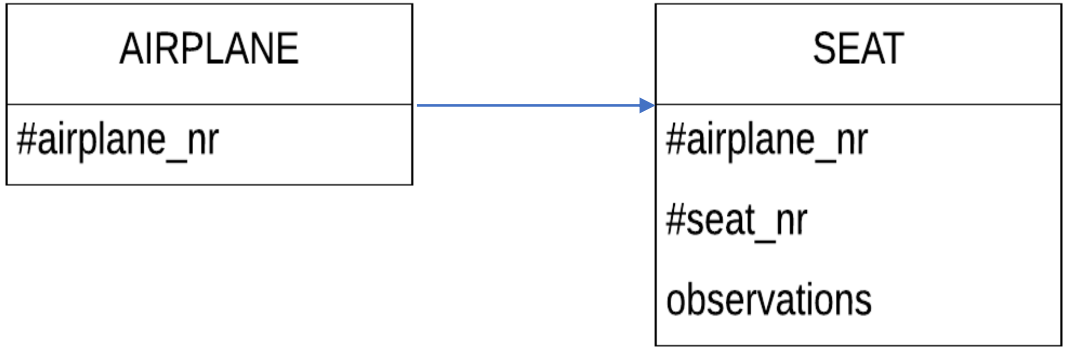
- Composed keys may be replaced with an artificial key.

Airline (1)Entities

- The airline has one or more airplanes. An airplane has a model number, and capacity. Each flight is carried out by airplanes. An airplane is uniquely identified by its Registration_no and a flight is identified by its Flight_no. A passenger can book a ticket for a flight.

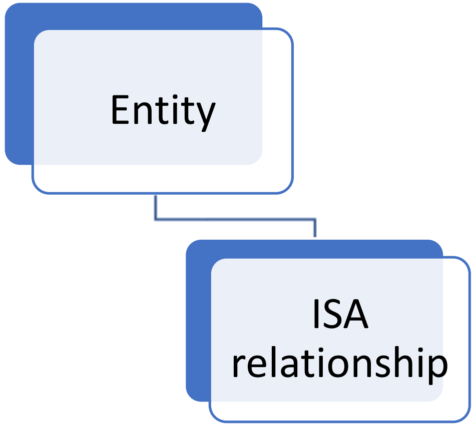
Entities

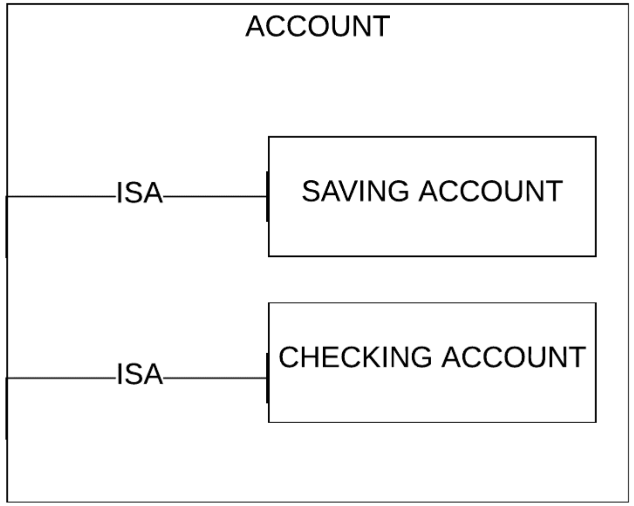




- Weak entity is an entity that depends on another entity.
- The primary key of a weak entity contains the primary key of the strong entity that it depends on + description/partial key.

Entities





- A sub-entity has the same key as the *super*-entity and all its attributes and relationships.

Relationship

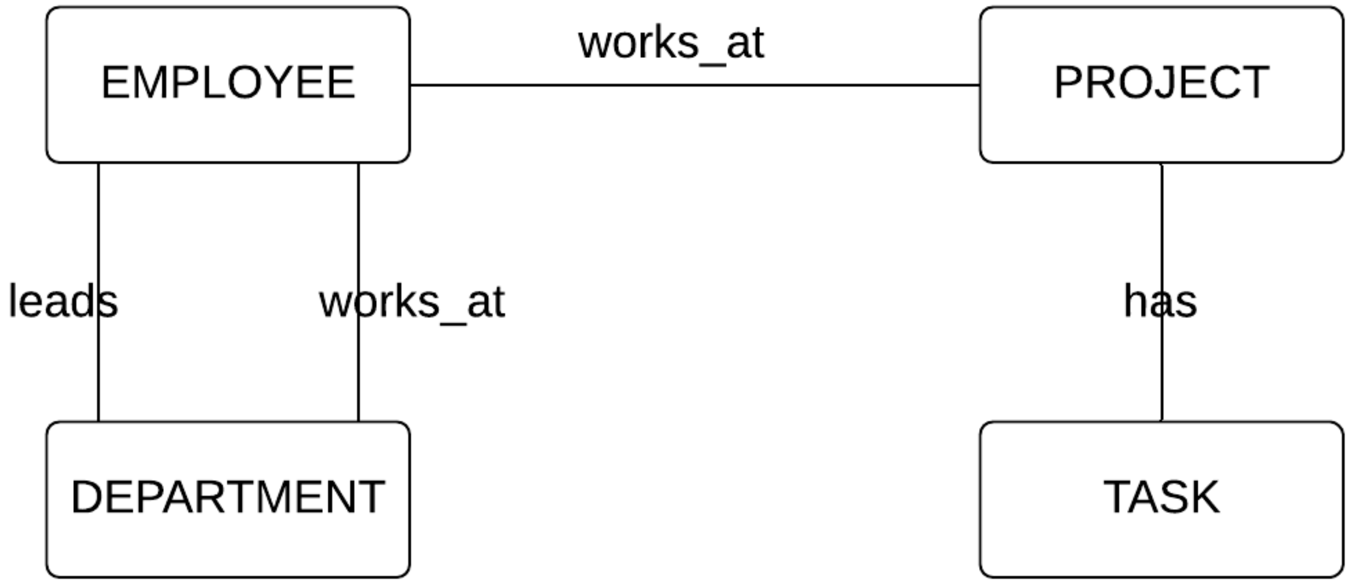
- Association between two or more entities (binary, ternary etc.)
- Relationship ➔ column (foreign key) or table.
- Graphical representation: oriented arc.

- Two relationships with the same name link different entities.

- Cardinality defines the numerical attributes of the relationship between two entities: MANDATORY (min) OPTIONAL (max)

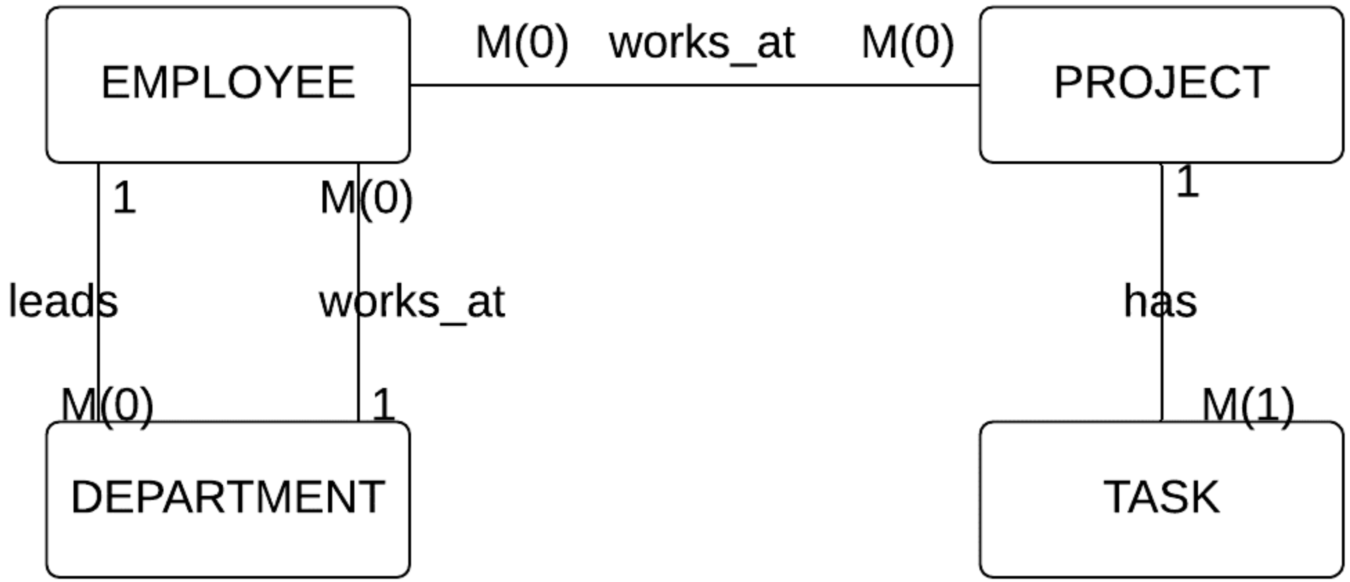
Relationship cardinality

- MANDATORY (must)
- OPTIONAL (may)



Relationship cardinality

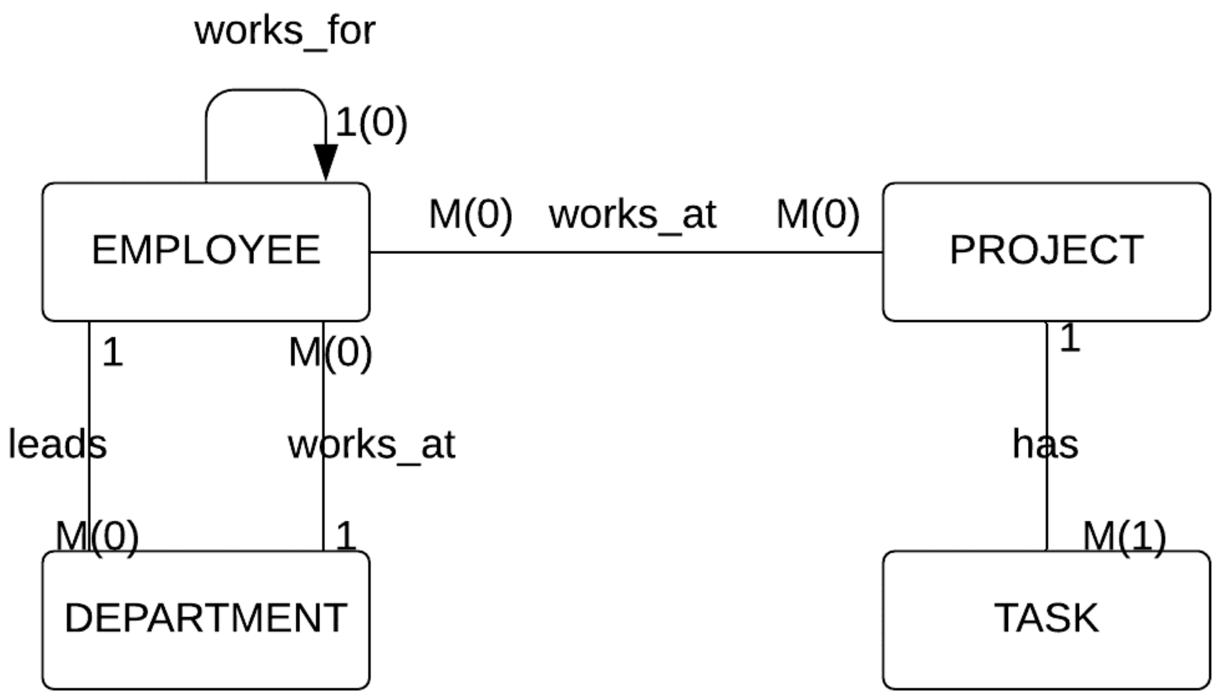
- MANDATORY (must)
- OPTIONAL (may)

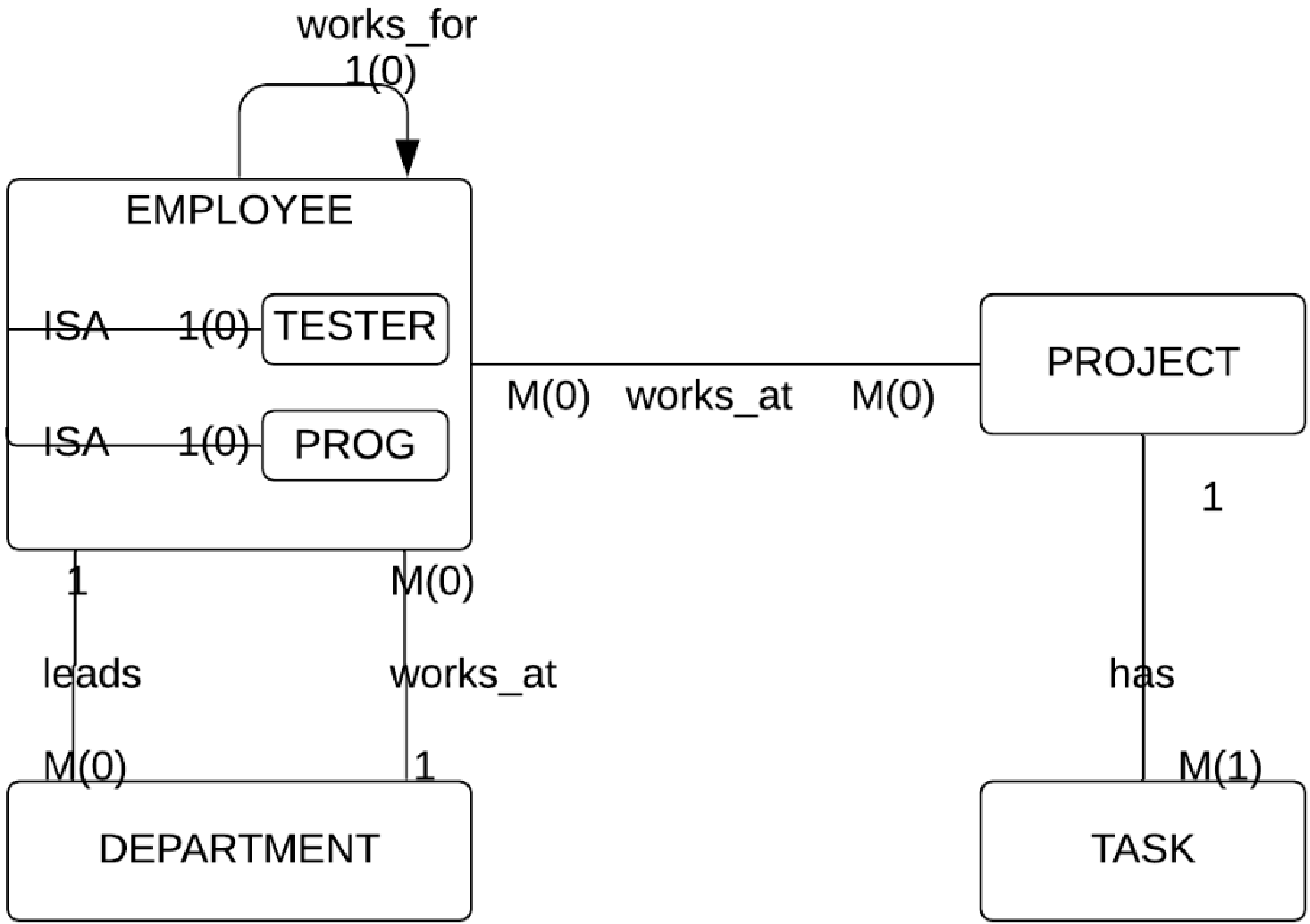


Relationship cardinality

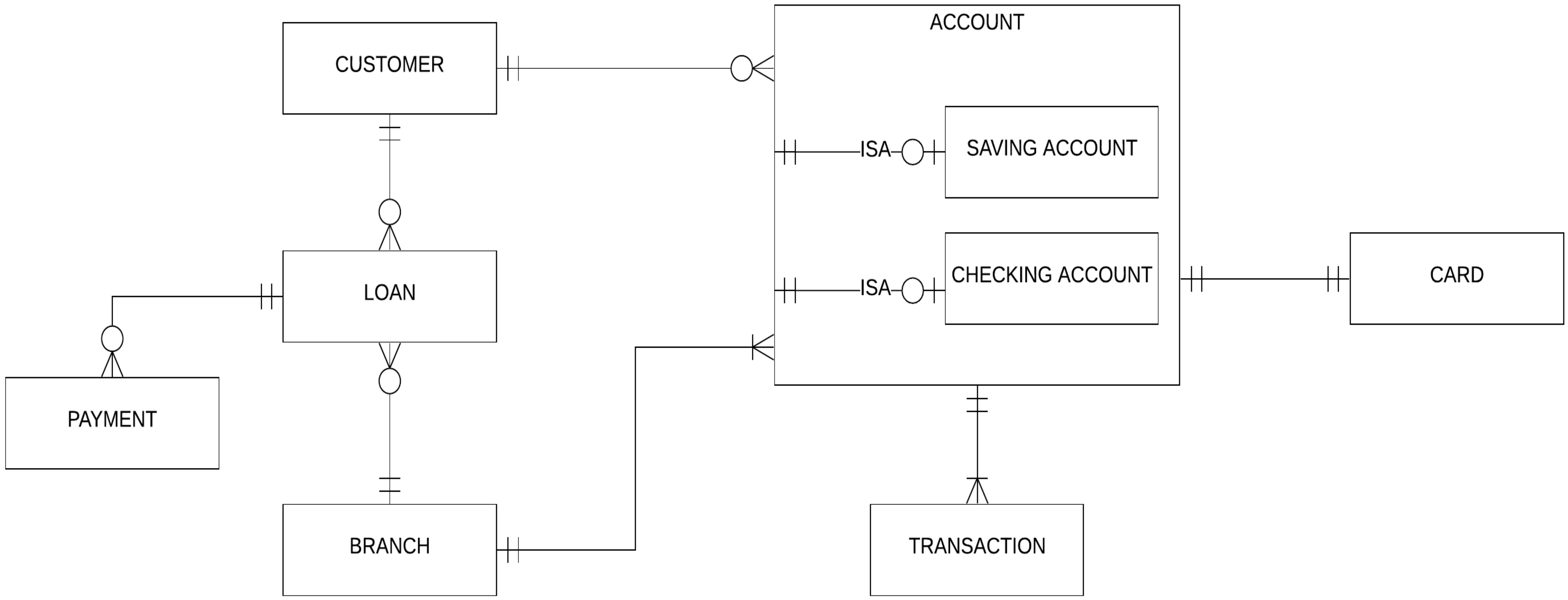
- Reflexive relationship

  unary relationship.



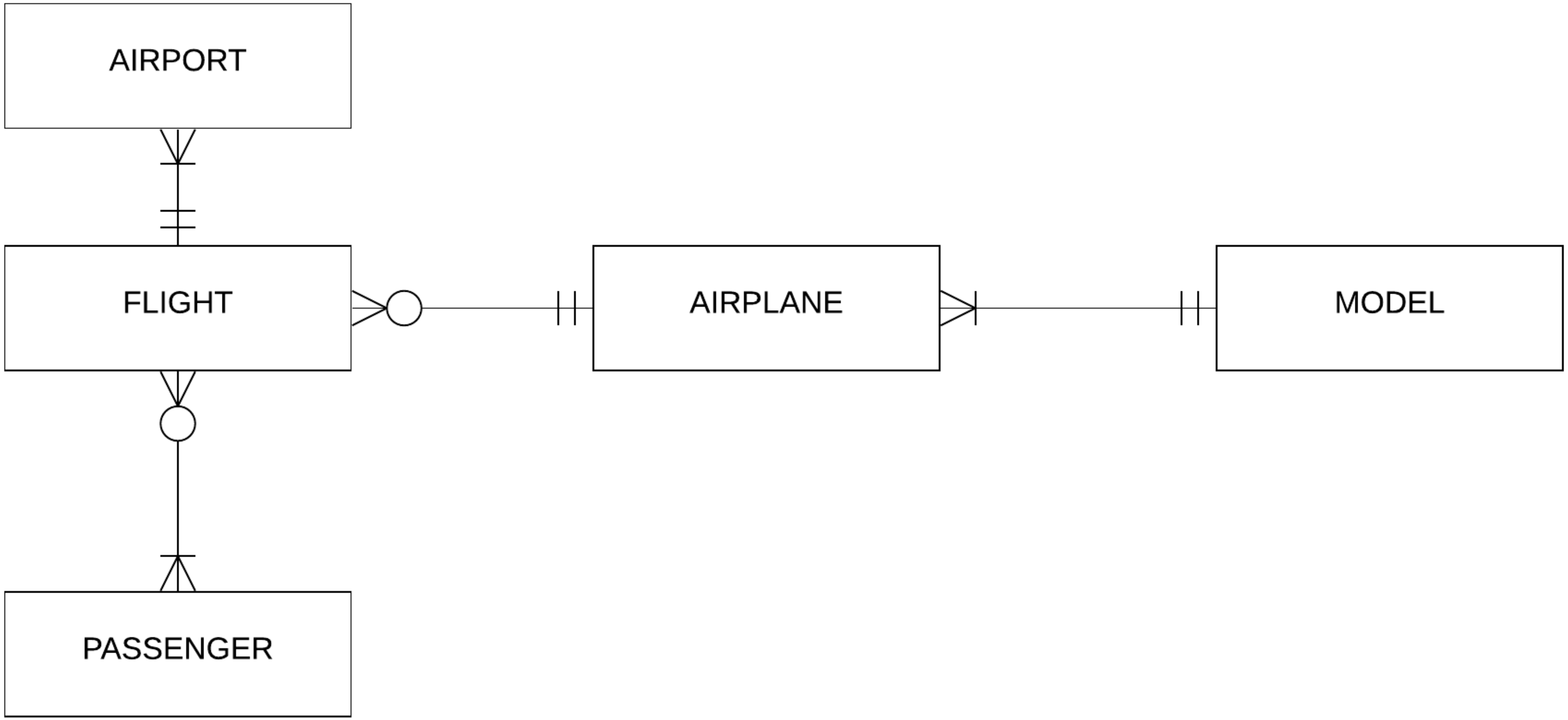

works_for
1(0)
EMPLOYEE
ISA
1(0)
TESTER
ISA
1(0)
PROG
M(0)
works_at
M(0)
PROJECT
1
1
M(0)
leads
works_at
has
M(0)
1
M(1)
DEPARTMENT
TASK

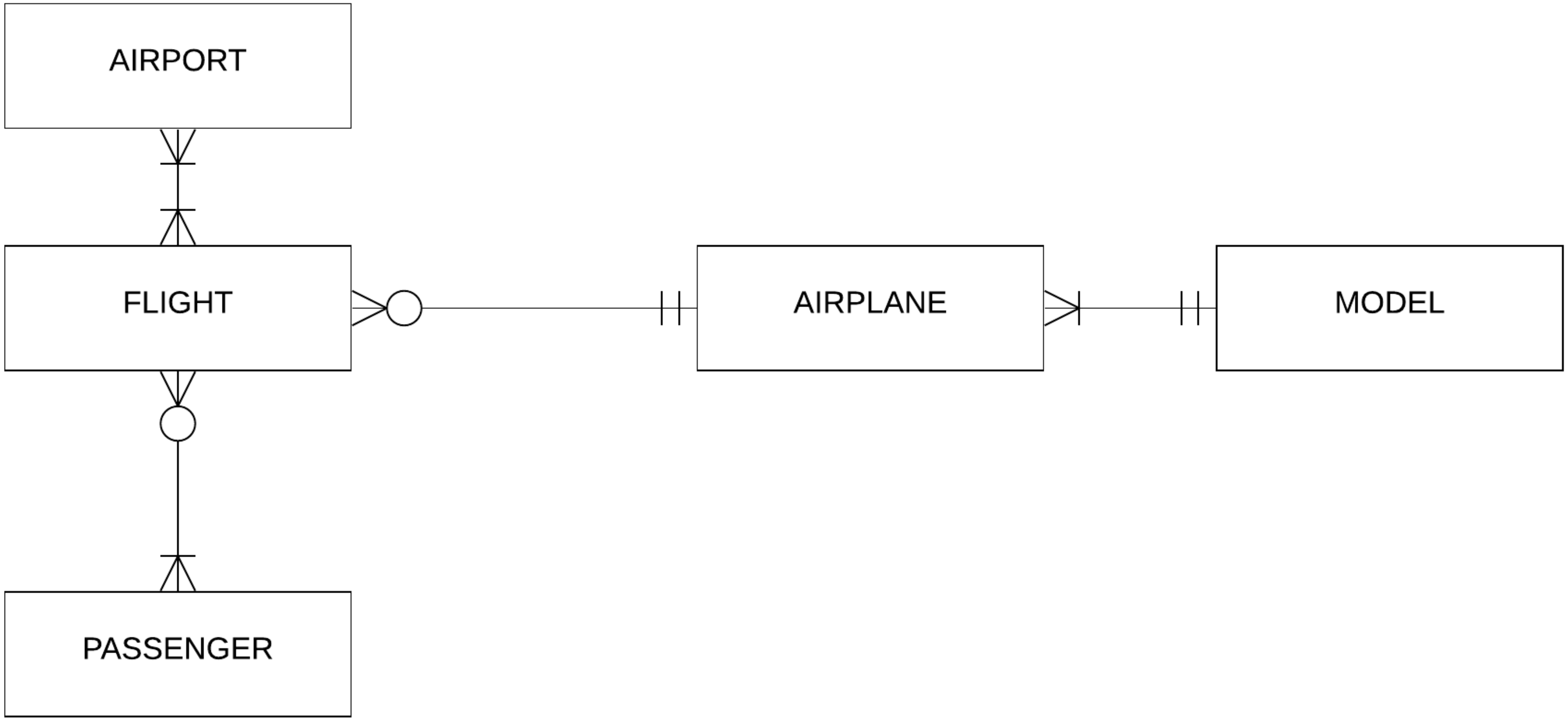
Banking (2)Relationships



Airline (1)Relationships

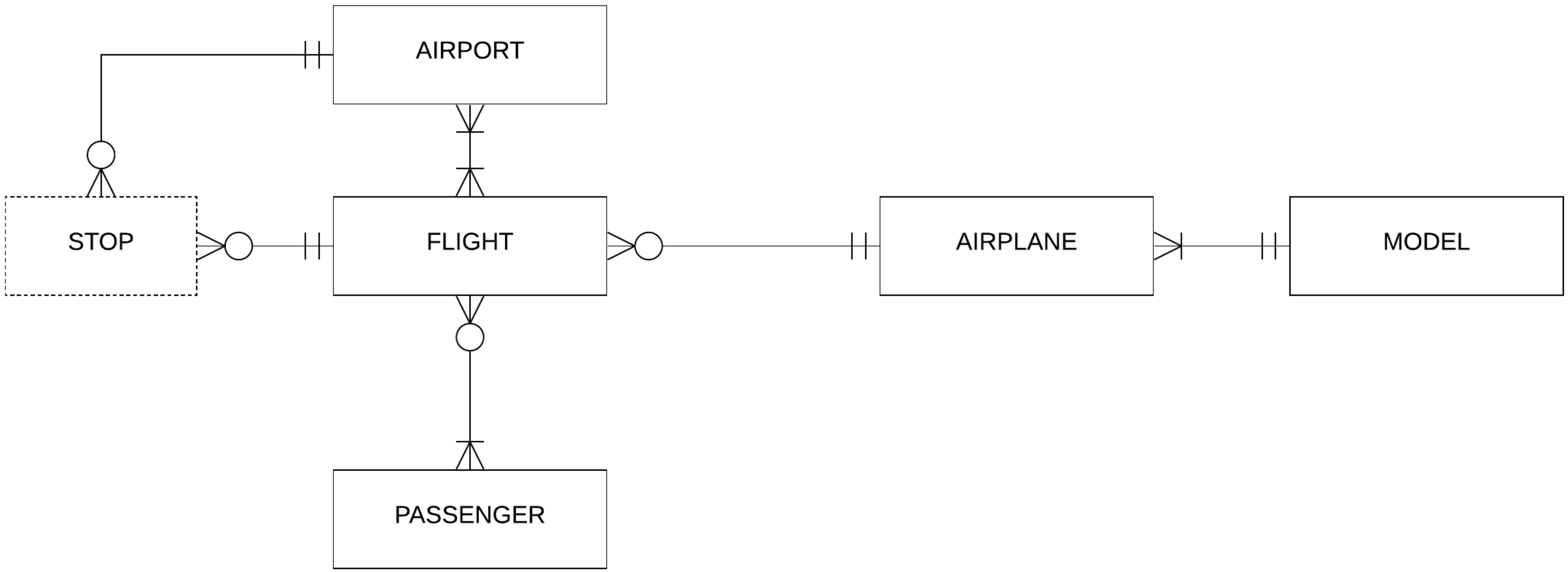
- The airline has one or more airplanes. An airplane has a model number, and capacity. Each flight is carried out by airplanes. An airplane is uniquely identified by its Registration_No and a flight is identified by its Flight_No. A passenger can book a ticket for a flight.


AIRPORT
FLIGHT
AIRPLANE
MODEL
PASSENGER


AIRPORT
FLIGHT
AIRPLANE
MODEL
PASSENGER

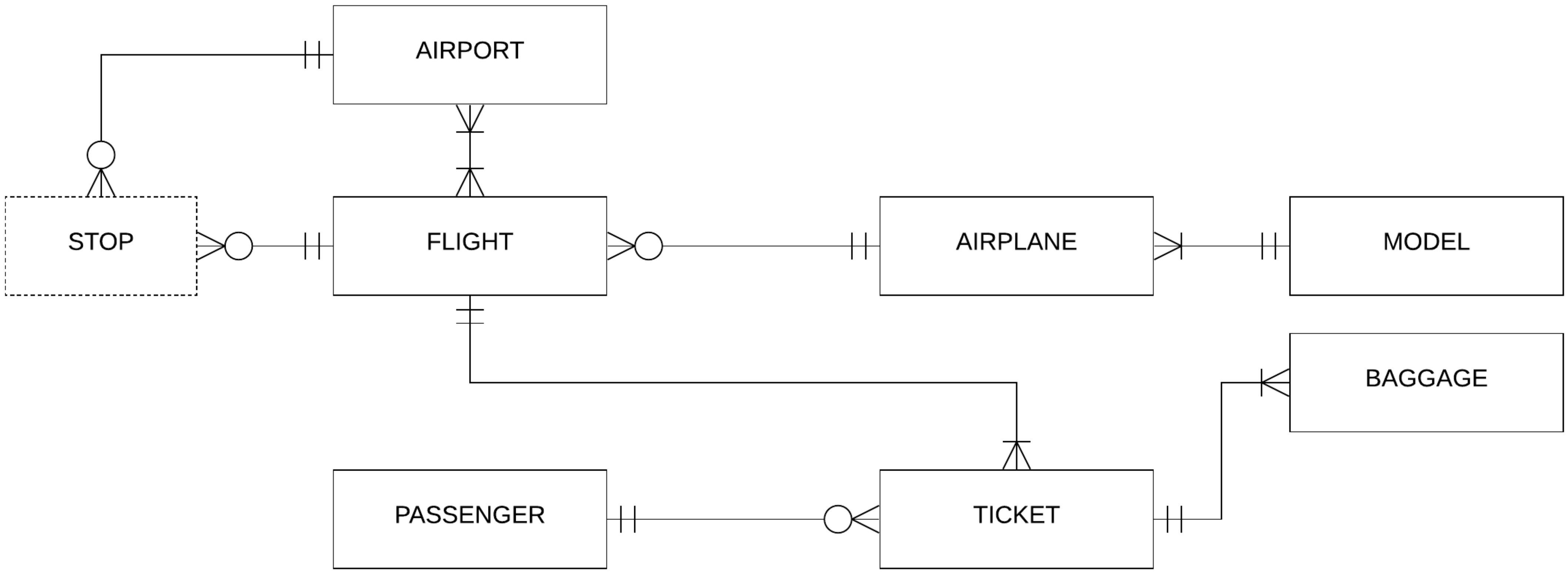
Airline (1)Relationships

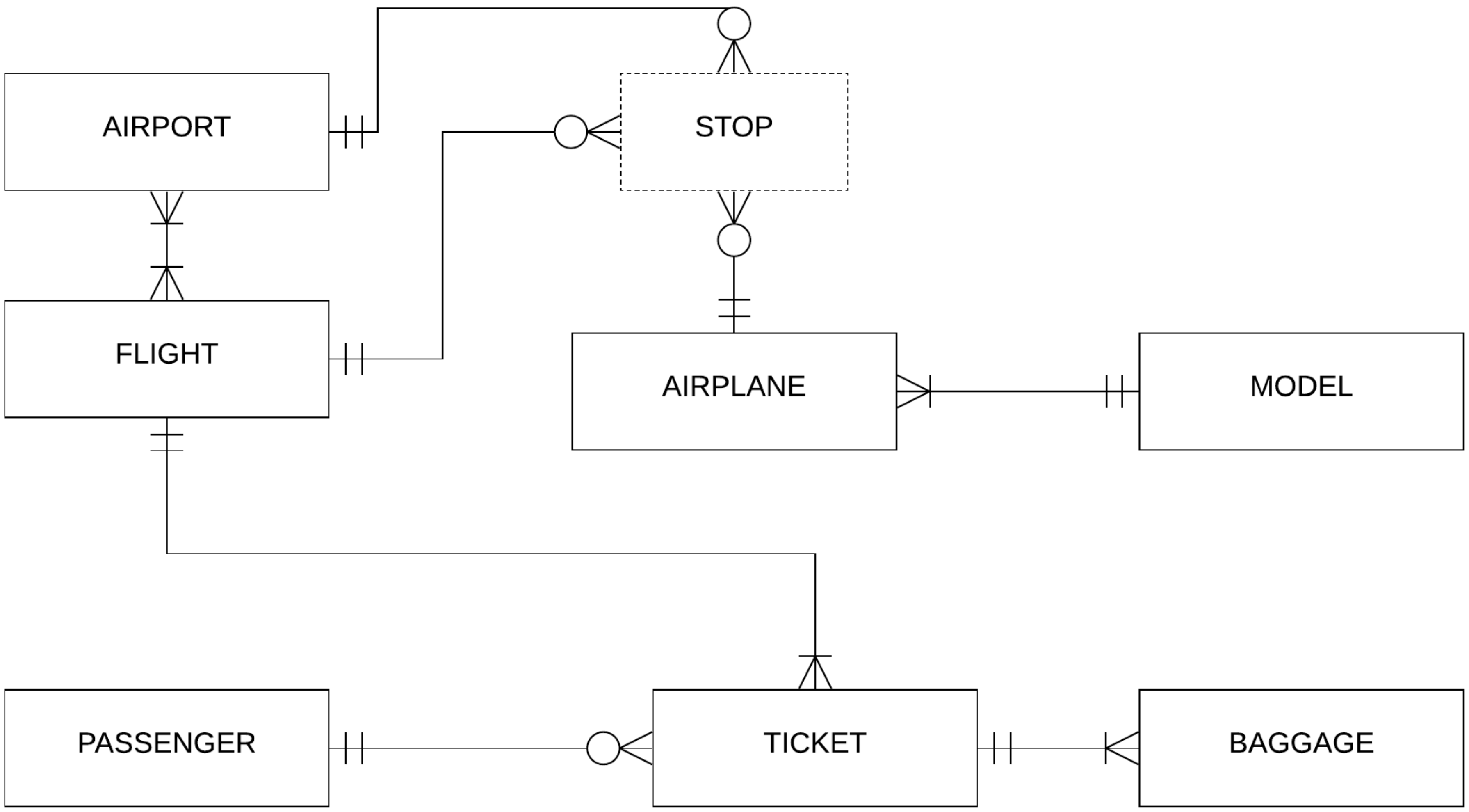
- The airline has one or more airplanes. An airplane has a model number, and capacity. Each flight is carried out by airplanes. An airplane is uniquely identified by its Registration_No and a flight is identified by its Flight_No. A passenger can book a ticket for a flight. A flight may have one or more stops.


AIRPORT
STOP
FLIGHT
AIRPLANE
MODEL
PASSENGER

Airline (1)Relationships

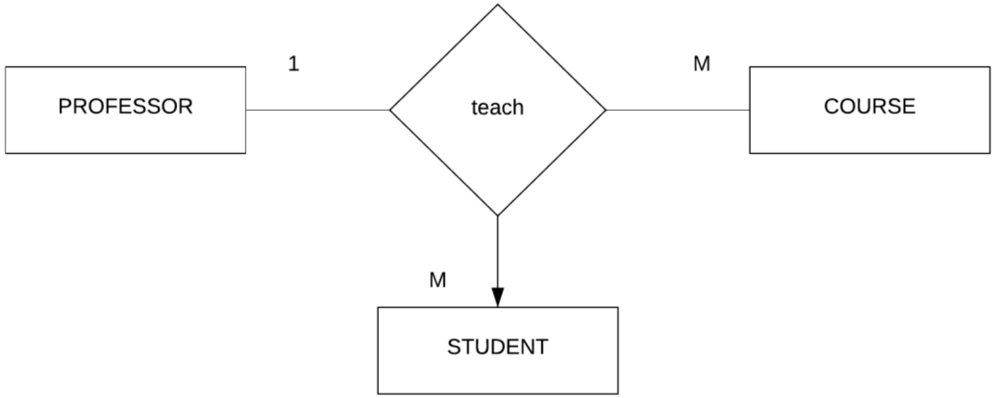
- The airline has one or more airplanes. An airplane has a model number, and capacity. Each flight is carried out by airplanes. An airplane is uniquely identified by its Registration_No and a flight is identified by its Flight_No. A passenger can book a ticket for a flight. A flight may have one or more stops. The passenger will pay for extra baggage.


AIRPORT
STOP
FLIGHT
AIRPLANE
MODEL
BAGGAGE
PASSENGER
TICKET


AIRPORT
STOP
FLIGHT
AIRPLANE
MODEL
PASSENGER
TICKET
BAGGAGE

Ternary relationships

- Relationship binding simultaneously 3 entities.



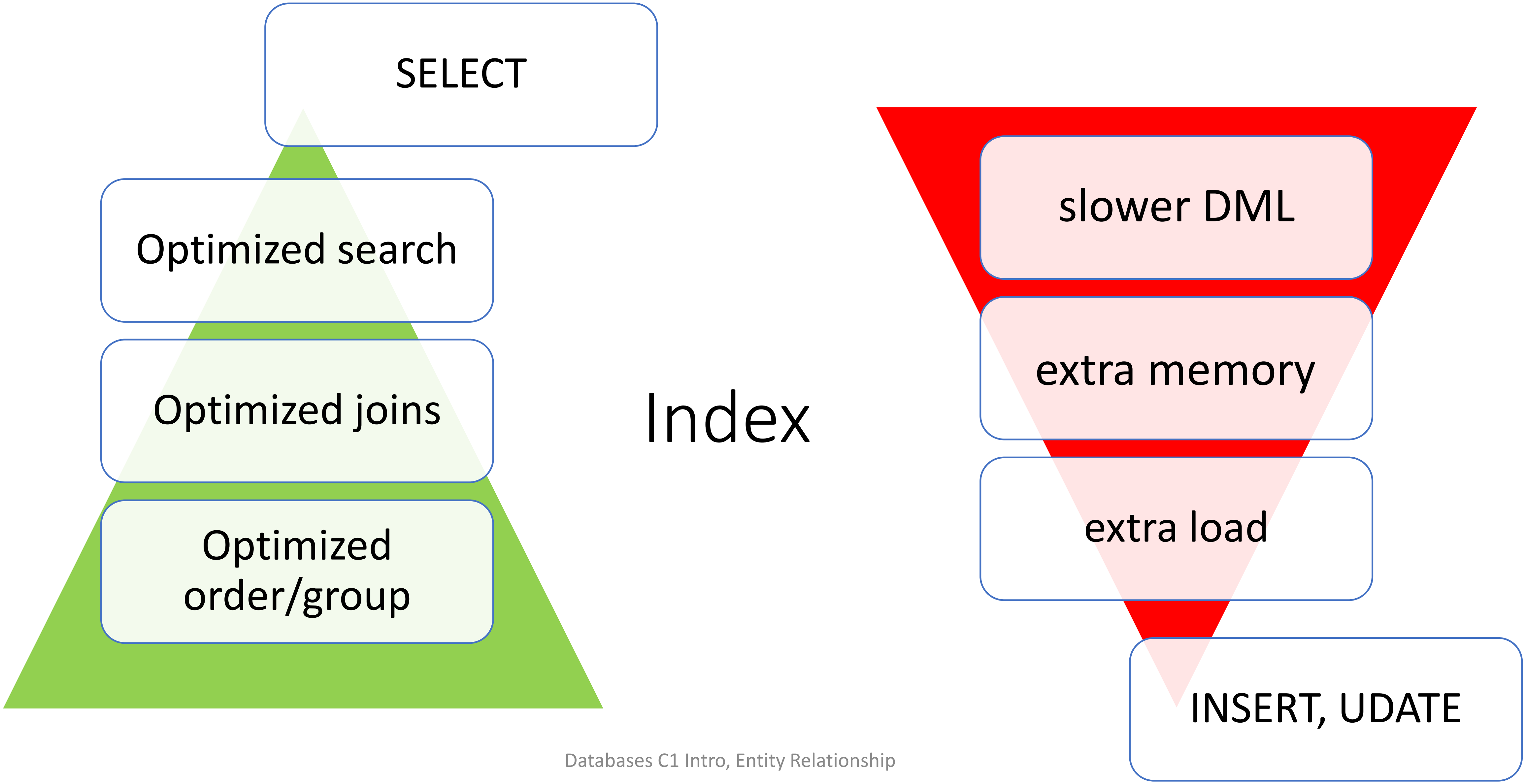
Indexes

Indexes

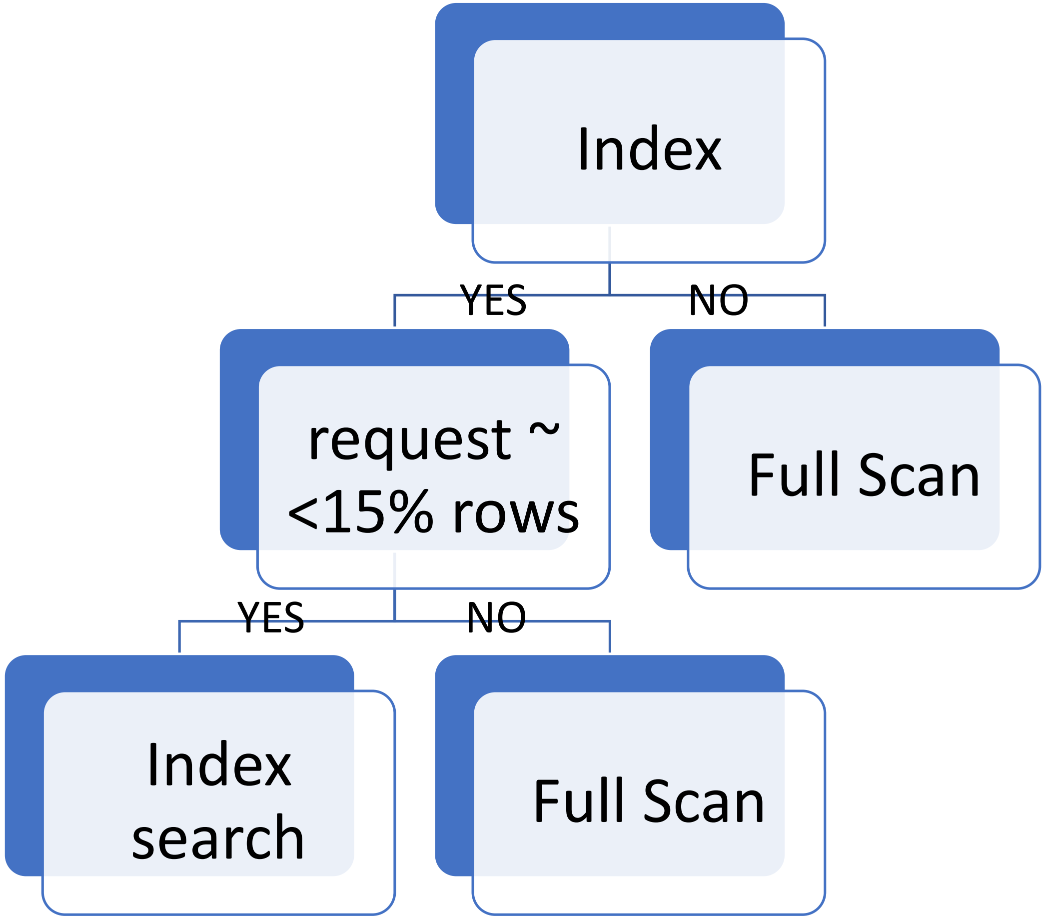
- Maps search key to data using specific data structures.

- Optimized search.
- Optimized joins (lookup in more than one table)
- Optimized order/group

- slower DML (insert and update operations).
- extra memory


SELECT
Optimized search
Optimized joins
Optimized order/group
Index
slower DML
extra memory
extra load
INSERT, UDATE

Sql Optimizer



Autogenerated columns

- MySQL auto-generated index (key):
  - DB_ROW_ID increases monotonically as new rows are inserted.
  - DB_ROLL_PTR roll pointer, points to log record.
  - DB_TRX_ID last transaction that updated or inserted the row.

- Oracle rowid:
  - Pseudo column 18 characters = 10 + 4 + 4 (block, row, file).
  - Store and return row address in hexadecimal format (string).
  - Unique identifier for each row.
  - Immutable.

Autogenerated columns

- Oracle rowid:
  - Used in where clause to select/update/delete a row.

- Oracle rownum:
  - Sequential number in which oracle has fetched the row, before ordering the result
  - Temporary generated along with a select statement.

- Mongo
  - ObjectID (timestamp 4Bytes + random 5Bytes + Count 3Bytes.

Index

- Data structure that optimize search.
- Automatically created when a primary key is defined.

MySQL

```
SHOW EXTENDED INDEX FROM index_test;
```

Oracle

```
select * from user_indexes
where table_name = 'INDEX_TEST';
```

Primay key

- Constraint imposed on insert/update behavior.
- NotNull & Unique.

MySQL

```
select * from  information_schema.statistics
where table_name = 'index_test1’
        and index_name = 'primary';
```

Oracle

```
select * from user_constraints
where table_name = 'INDEX_TEST';
```

Index types

Clustered index (SqlServer, MySql)

- Defines the order in which data is physically stored in a table. (index on column semester)
- Only one clustered index on a table (data can be stored in only one order)
- A cluster index is created automatically when a primary key is defined.
- No second data structure for the table

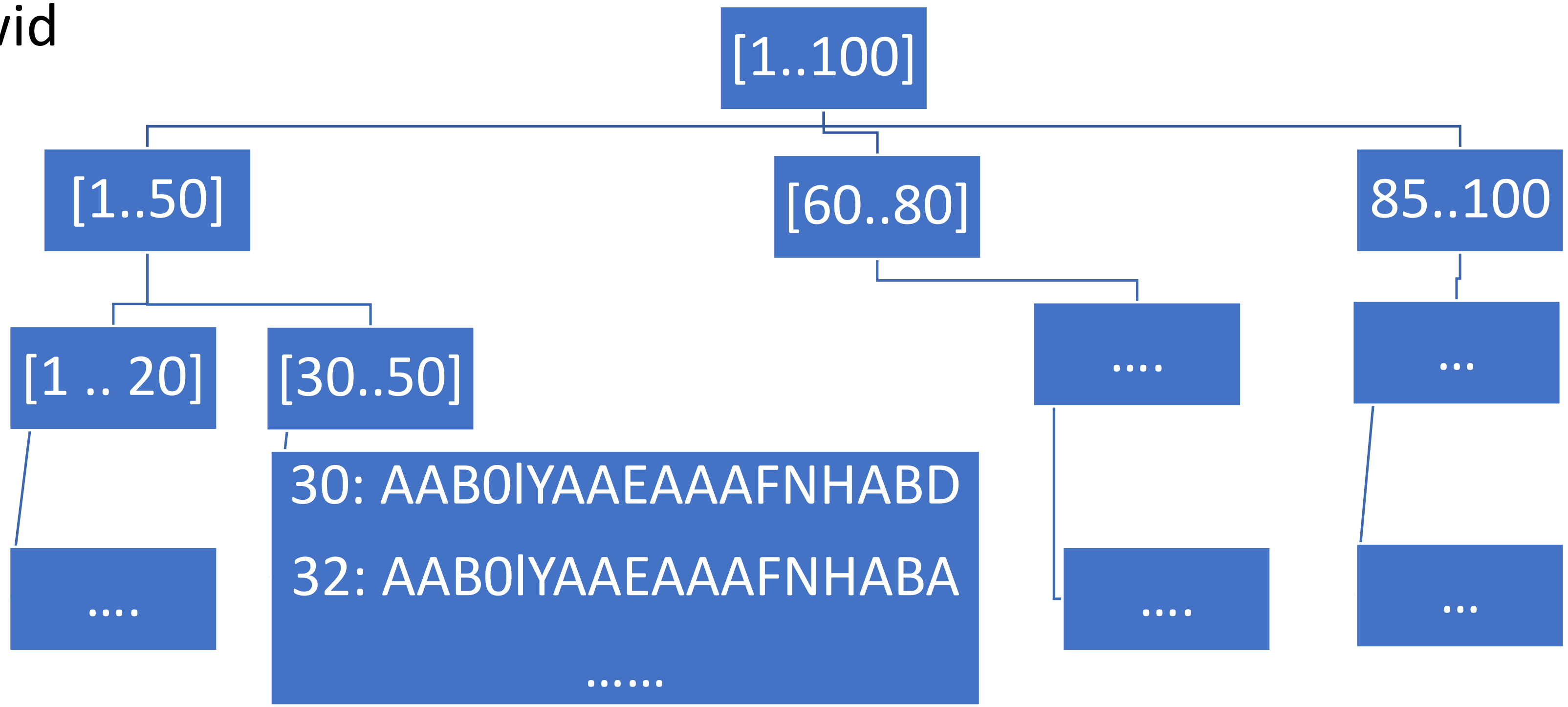
- Oracle: IOT index organized tables. Table is stored in a B-tree structure. (key and non-keys column are stored in leafs)

B – Tree

- B -- Balanced tree.
- Default index type in Oracle.
- Two types of nodes: branch blocks and leaf blocks.
- Branch blocks pointers to lower levels.
- Leaf blocks contain rowids/physical address.
- The number of blocks traversed in order to reach a leaf block is the same for each leaf block.

B – Tree

- create index idx_emp_id on employees(employee_id).
  - Devide employee_id values in sorted ranges.
  - Leafs nodes store rowid



Reverse index

- B – tree where keys are in reverse order. Key 4573 is stored 3754.
- Optimized insert operations.
- Key 4573 will be stored in the same block with key 9573

  while 4574 will be stored in a different block.

Bitmap index

- Used for columns with limited number of distinct values.
- Example: language proficiency levels (en)

emp_id	en	fr
1	A1	B1
2	A2	B2
3	C1	A1
4	A1	B1
5	A1	

row_id	A1	A2	B1	B2	C1	C2
AAB0lYAAEAAAFNHABD	1	0	0	0	0	0
AAB0lYAAEAAAFNHABV	0	1	0	0	0	0
AAB0lYAAEAAAFNHABX	0	0	0	0	1	0
AAB0lYAAEAAAFNHAAv	1	0	0	0	0	0
AAB0lYAAEAAAFNHAAV	1	0	0	0	0	0

Relational Model

COURSE 2: Databases

Relational model

Relational model

- Database = collection of RELATIONS
  - relation in relational model ≠ relationship in ERD.

- Relation Schema: A relation schema represents the name of the relation with its attributes.

- Attribute domain – Each attribute has some pre-defined values.

Relational model

- Codd rules 1985 → Is DBMS relational? If yes, to what degree?

https://computing.derby.ac.uk/c/codds-twelve-rules/

Relational Integrity constraints

RELATIONS

OPERATORS

Relational Integrity constraints

RELATIONS

OPERATORS

- Domain constraints
  - the value of each attribute must be unique, specified data types integers, real numbers, characters, Booleans, variable length strings etc.

- Key constraint
  - Unique + not null   PK

- Referential integrity constraints
  - the value of a FK is null or it coresponds to the value of a PK.

Relational Integrity constraints

RELATIONS

OPERATORS

- Relational shema $R(A_1, A_2, \ldots, A_n)$

- $R \subset D_1 \times D_2 \times \cdots \times D_n, D_i \; domain$

Relational Integrity constraints

RELATIONS

OPERATORS

- UNION, INTERSECT, PRODUCT, DIFFERENCE

- PROJECT
- SELECT
- JOIN
- DIVISION

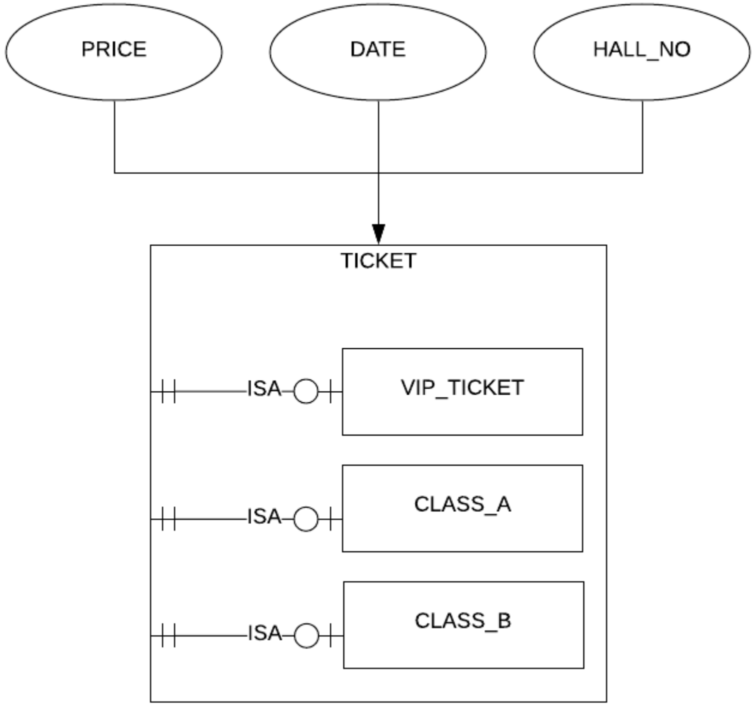
Converting ER into RM

Rules for entities

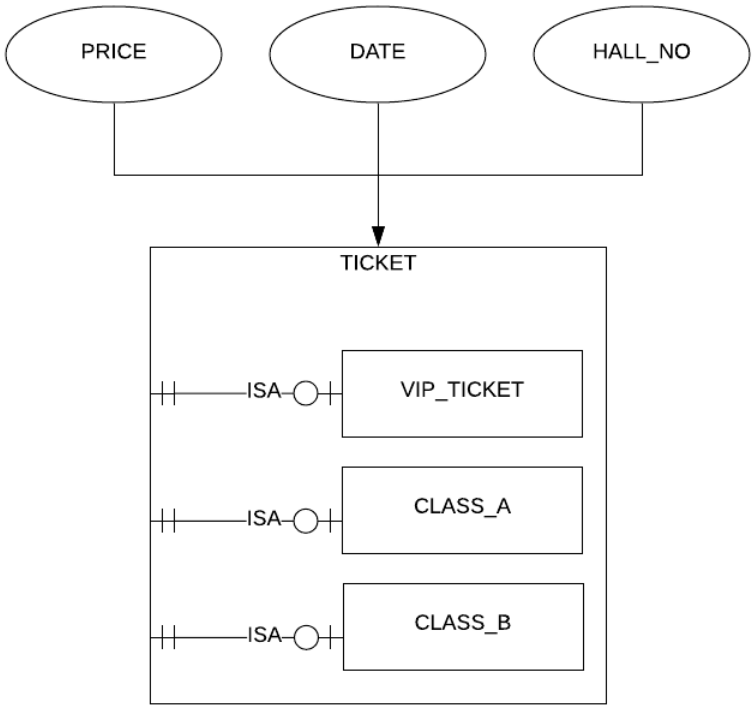
- Strong entities → independent tables
  - PK doesn't contain foreign keys.
- Weak entities → table
  - PK contains the key of the related strong entity and or more key attributes.
- Sub-entities → one ore more tables, Boolean attribute, type_attribute
  - PK may also represent a FK.

Rules for entities strong – weak entity

Rules for entities ISA

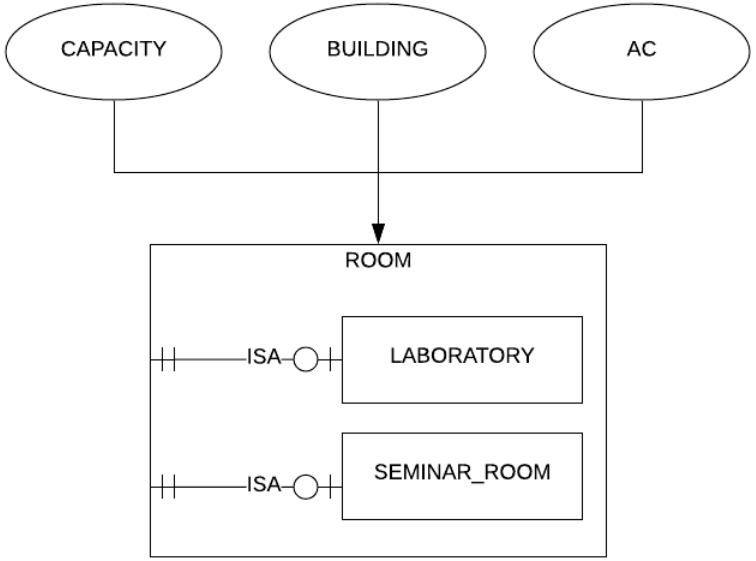


Rules for entities ISA

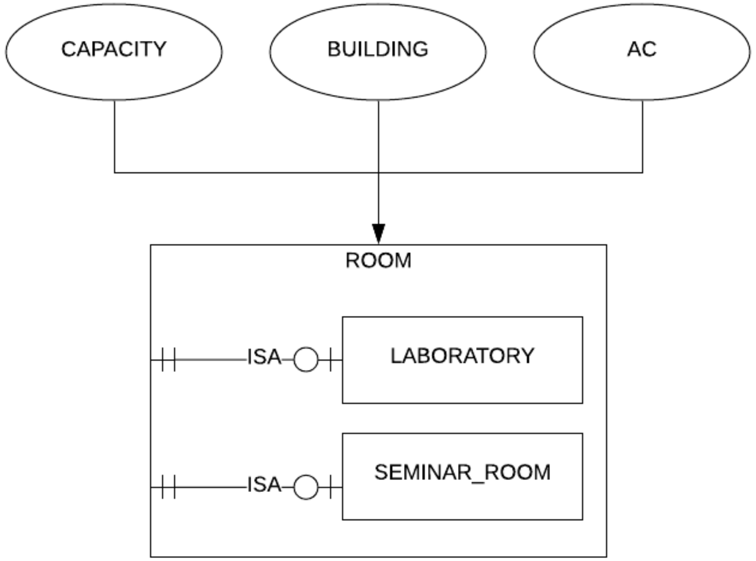


TICKET_ID	PRICE	HALL_NO	DATE	TYPE
1	200	Coliseum	08/03/20	VIP
2	150	Lyttelton	14/04/20	A
3	140	Olivier	01/05/20	A
4	90	Coliseum	04/06/20	B
5	220	Lyttelton	08/03/20	VIP
6	95	Olivier	14/04/20	B
7	210	Coliseum	20/03/20	VIP

Rules for entities ISA

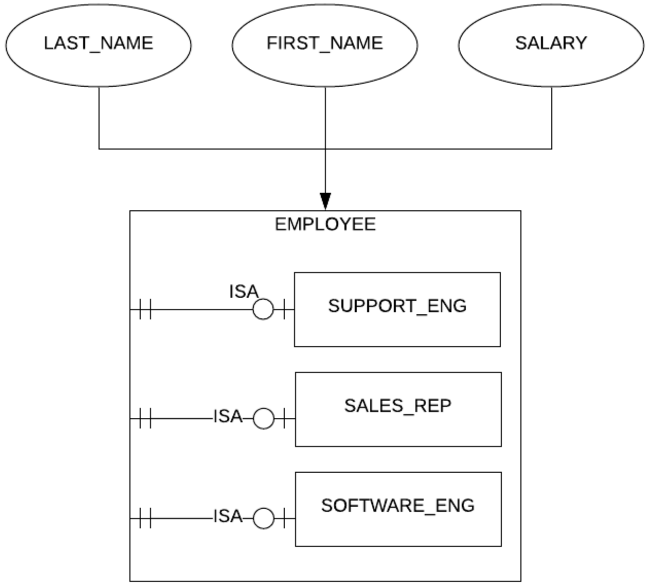


Rules for entities ISA

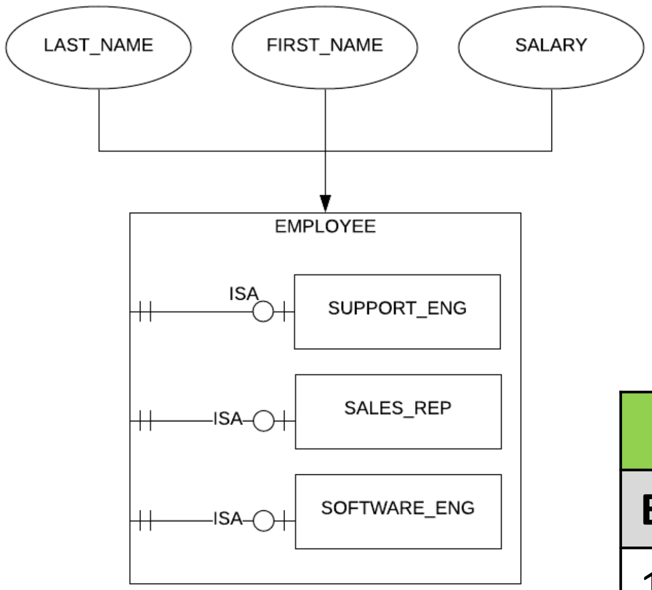


ROOM_ID	CAPACITY	BUILDING	LAB	SEM
1	40	FMI	1	1
2	45	Magurele	1	0
3	30	Geografie	0	0
4	90	FMI	1	0
5	80	FMI	1	0
6	95	Drept	0	1
7	20	FMI	1	1

Rules for entities ISA



Rules for entities ISA



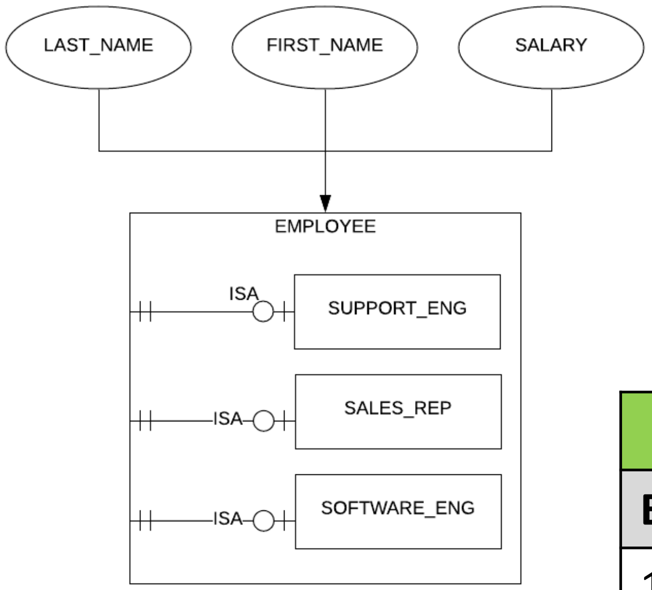
EMPLOYEES			
EMP_ID	**LAST_NAME**	**FIRST_NAME**	**SALARY**
1	Smith	John	2500
2	Grant	Anne	2700
3	Brown	Gregory	2300
…			

SUPPORT_ENG	
EMP_ID	**LEVEL**
1	3
…	…

SALES_REP	
EMP_ID	**TARGET**
2	25
…	…

SOFTWARE_ENG	
EMP_ID	**TEEM**
3	
…	…

Rules for entities ISA



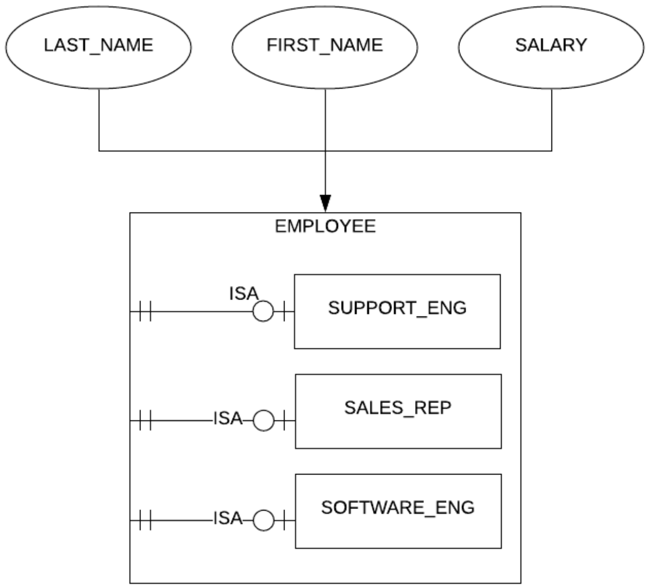
EMPLOYEES			
EMP_ID	**LAST_NAME**	**FIRST_NAME**	**SALARY**
1	Smith	John	2500
2	Grant	Anne	2700
3	Brown	Gregory	2300
…			

SUPPORT_ENG	
EMP_ID	**LEVEL**
1	3
…	…

SALES_REP	
EMP_ID	**TARGET**
2	25
…	…

SOFTWARE_ENG	
EMP_ID	**TEEM**
3	
…	…

Rules for entities ISA



SUPPORT_ENG				
EMP_ID	**LEVEL**	**LAST_NAME**	**FIRST_NAME**	**SALARY**
1	3	Smith	John	2500
…	…			

SALES_REP				
EMP_ID	**TARGET**	**LAST_NAME**	**FIRST_NAME**	**SALARY**
2	25	Grant	Anee	2700
…	…			

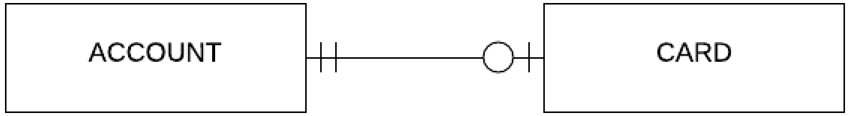
SOFTWARE_ENG				
EMP_ID	**TEEM**	**LAST_NAME**	**FIRST_NAME**	**SALARY**
3	3	Brown	Gregory	2300
…	…			

Rules for relationships

- 1 to 1 & 1 to M → foreign keys.
  - 1 (PK) to M (FK)
  - Usually in 1 to 1 relationships the FK is placed in the tables with fewer rows.
- M to M → associative table.
  - PK contains FKs and additional column.
- Ternary relationships → associative table.
  - PK contains FKs and additional column.

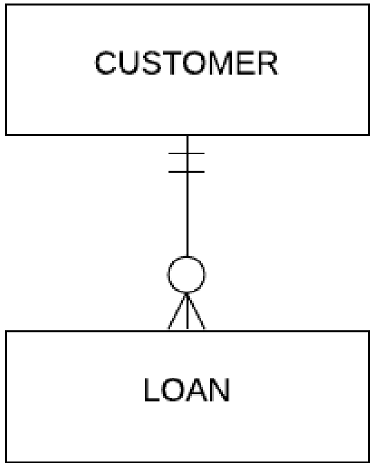
One to One

ACCOUNT			
ACCOUNT_ID	**LAST_NAME**	**FIRST_NAME**	**DATE**
10	Snow	John	08/03/20
22	Grant	Anee	14/04/20
300	Brown	Gregory	01/05/20
…	…	…	…



CARD			
CARD_ID	**ACCOUNT_ID**	**CVN**	**DATE**
16897	10	125	18/04/21
24789	22	987	14/04/22
34597	300	875	03/05/21
…	…	…	…

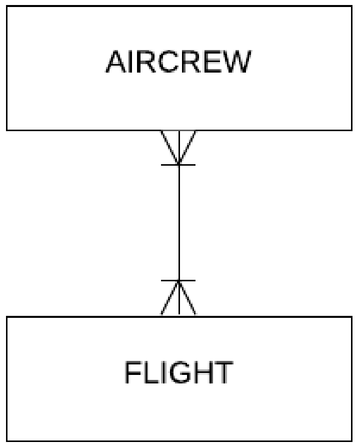
One to Many



CUSTOMER			
CUSTOMER_ID	**LAST_NAME**	**FIRST_NAME**	**....**
10	Snow	John	....
22	Grant	Anee	....
300	Brown	Gregory	....
…	…	…	…

LOAN			
LOAN_ID	**CUSTOMER_ID**	**VALUES**	**DATE**
16897	10	125000	18/04/21
24789	22	987000	14/04/22
34597	300	87500	03/05/21
…	…	…	…

Many to Many

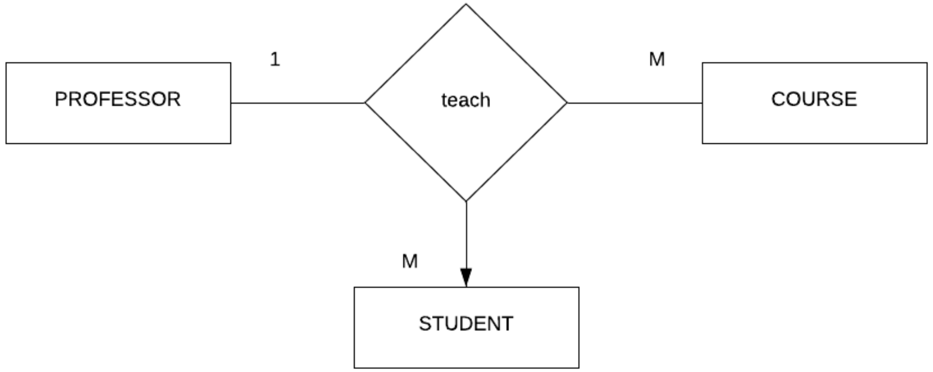


FLIGHT			
FLIGHT_ID	DEP_AIRPORT	DATE	….
1	Gatwick Airport	20/04/21	….
2	Grant	14/05/20	….
…	…	…	…

FLIGHT_CREW		
CREW_ID	FLIGHT_ID	OBSERVATIONS
10	1	…
22	1	…
10	2	…

AIRCREW			
CREW_ID	LAST_NAME	FIRST_NAME	JOB_ID
10	Snow	John	captain
22	Grant	Anee	first_officer
…	…	…	…

Ternary Relationships

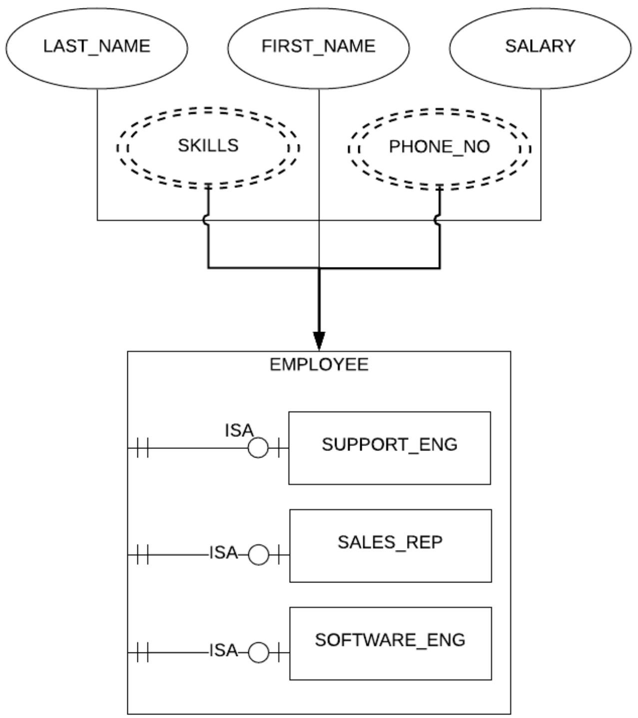


TEACH			
PROFESSOR_ID	**COURSE_ID**	**STUDENT_ID**	**GRADE**
1	BD	1001	9
1	SGBD	1002	10
1	BD	1002	8
2	TAP	1001	8
2	TAP	1002	10
2	AG	1001	5
….	….	….	….

Rules for attributes

- Simple attribute → column
- Multivalued attributes → weak entity → table
  → set of columns

Rules for entities ISA



EMPLOYEES					
EMP_ID	**LAST_NAME**	**FIRST_NAME**	**SALARY**	**PHONE1**	**PHONE2**
1	Smith	John	2500	0745…	0720…
2	Grant	Anne	2700	07497…	NULL
3	Brown	Gregory	2300	NULL	07458..
…	…	…	…	…	…

EMP_SKILL		
EMP_ID	**SKILL**	**LEVEL**
1	Python	3
1	C++	2
1	NoSql	3
2	SQL	1

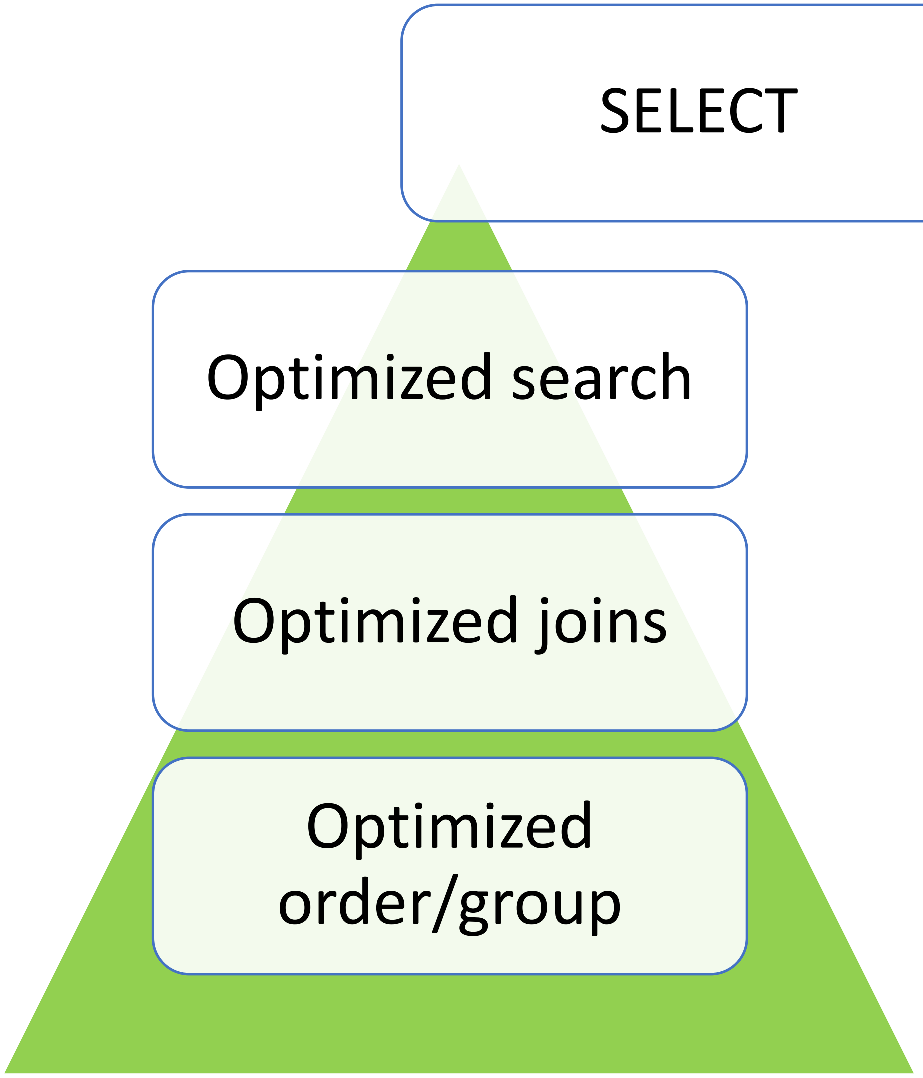
Indexes

Indexes

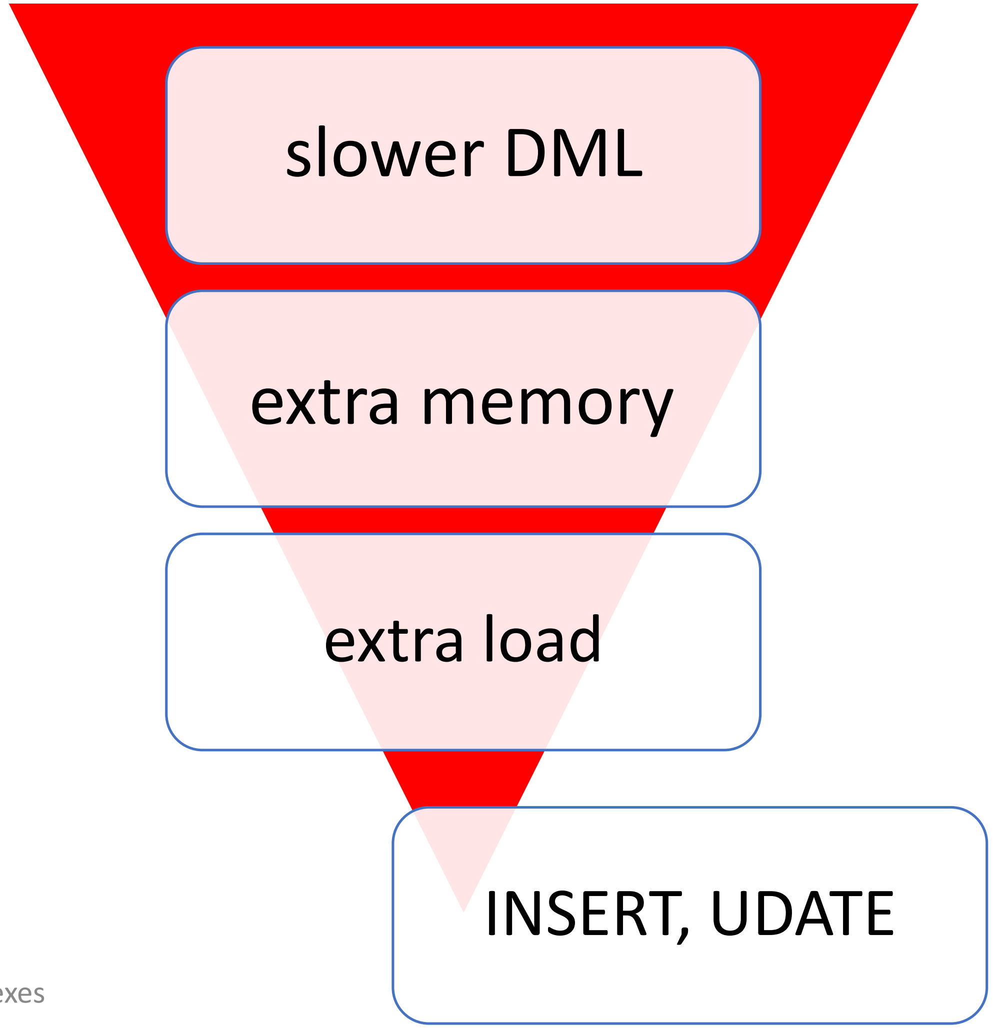
- Maps search key to data using specific data structures.

- Optimized search.
- Optimized joins (lookup in more than one table)
- Optimized order/group

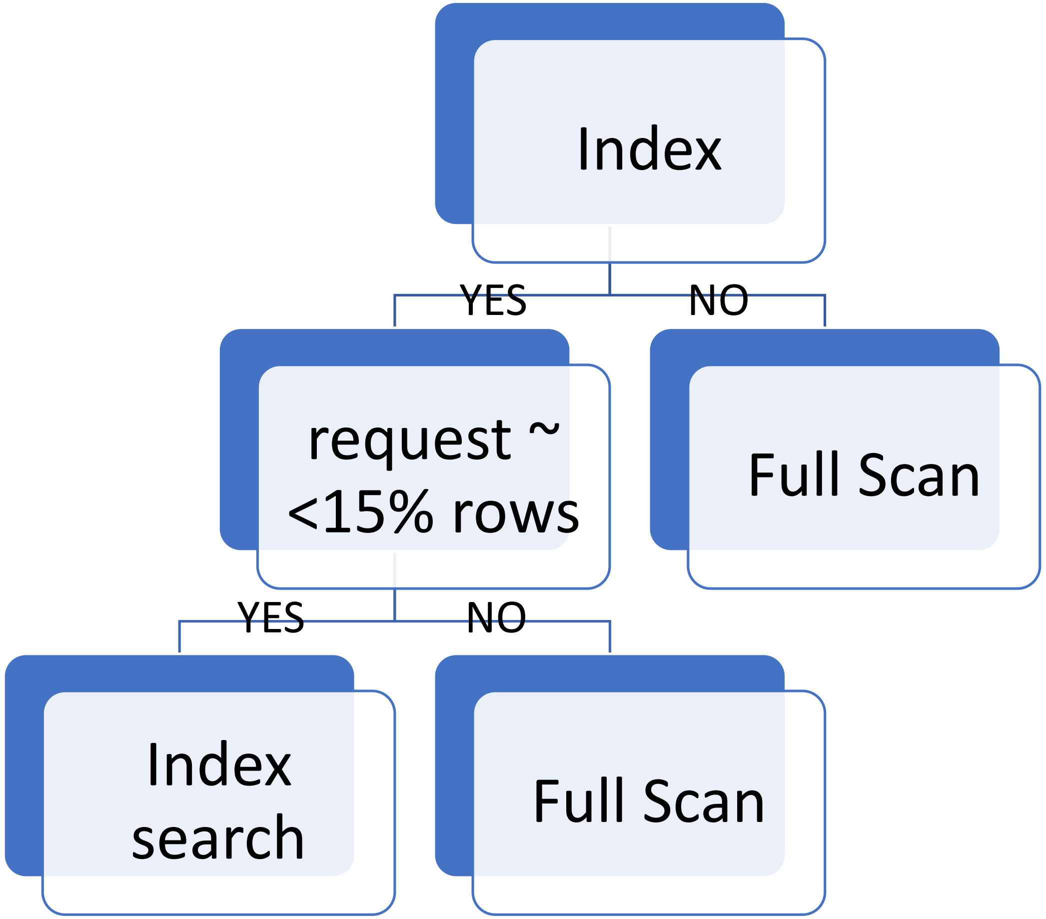
- slower DML (insert and update operations).
- extra memory


SELECT
Optimized search
Optimized joins
Optimized order/group

Index


slower DML
extra memory
extra load
INSERT, UDATE

Sql Optimizer



Autogenerated columns

- MySQL auto-generated index (key):
  - DB_ROW_ID increases monotonically as new rows are inserted.
  - DB_ROLL_PTR roll pointer, points to log record.
  - DB_TRX_ID last transaction that updated or inserted the row.

- Oracle rowid:
  - Pseudo column 18 characters = 10 + 4 + 4 (block, row, file).
  - Store and return row address in hexadecimal format (string).
  - Unique identifier for each row.
  - Immutable.

Autogenerated columns

- Oracle rowid:
  - Used in where clause to select/update/delete a row.
- Oracle rownum:
  - Sequential number in which oracle has fetched the row, before ordering the result
  - Temporary generated along with a select statement.
- Mongo
  - ObjectID (timestamp 4Bytes + random 5Bytes + Count 3Bytes.

Index

- Data structure that optimize search.
- Automatically created when a PK/unique constraint is defined.

MySQL

```
SHOW EXTENDED INDEX FROM index_test;
```

Oracle

```
select * from user_indexes
where table_name = 'INDEX_TEST';
```

Primay key

- Constraint imposed on insert/update behavior.
- NotNull & Unique.

MySQL

```
select * from  information_schema.statistics
where table_name = 'index_test1’
      and index_name = 'primary';
```

Oracle

```
select * from user_constraints
where table_name = 'INDEX_TEST';
```

Index types

Clustered index (SqlServer, MySql)

- Defines the order in which data is physically stored in a table. (index on column semester)
- Only one clustered index on a table (data can be stored in only one order)
- A cluster index is created automatically when a primary key is defined.
- No second data structure for the table

- Oracle: IOT index organized tables. Table is stored in a B-tree structure. (key and non-keys column are stored in leafs)

B – Tree

- B -- Balanced tree.
- Default index type in Oracle.
- Two types of nodes: branch blocks and leaf blocks.
- Branch blocks pointers to lower levels.
- Leaf blocks contain rowids/physical address.
- The number of blocks traversed in order to reach a leaf block is the same for each leaf block.

B – Tree

- create index idx_emp_id on employees(employee_id).
  - Divide employee_id values in sorted ranges.
  - Leaves nodes store rowid


[1. . 100]
[1. . 50]
[60. . 80]
[85. . 100]
[1. . 20]
[30. . 50]
....
.....
....
30: AAB0lYAAEAAAFNHABD
32: AAB0lYAAEAAAFNHABA .....
....
....

Reverse index

- B – tree where keys are in reverse order. Key 4573 is stored 3754.
- Optimized insert operations.
- Key 4573 will be stored in the same block with key 9573

  while 4574 will be stored in a different block.

Bitmap index

- Used for columns with limited number of distinct values.
- Example: language proficiency levels (en)

emp_id	en	fr
1	A1	B1
2	A2	B2
3	C1	A1
4	A1	B1
5	A1	

row_id	A1	A2	B1	B2	C1	C2
AAB0lYAAEAAAFNHABD	1	0	0	0	0	0
AAB0lYAAEAAAFNHABV	0	1	0	0	0	0
AAB0lYAAEAAAFNHABX	0	0	0	0	1	0
AAB0lYAAEAAAFNHAAv	1	0	0	0	0	0
AAB0lYAAEAAAFNHAAV	1	0	0	0	0	0

Transactional systems

COURSE 4: Databases

Transactional systems

Transaction

- Set of operations on the database, set of statements:
  - insert, update, delete
- Delimited by statements or function calls of type:
  - begin transaction
  - end transaction
- All operations are finalized with success or none is saved in the db.
- A transactional system must
  - manage concurrent transactions.
  - ensure consistent data in case of failure.

Transaction

Statement 1

Statement 2

    commit -- end transaction 1

Statement 3

Statement 4

Statement 5

    commit -- end transaction 2

Transaction properties

ACID

ATOMICITY | CONSISTENCY | ISOLATION | DURABILITY

- all changes or none
  - collection of steps → single indivisible unit.
- If one operation fails all changes to the database must be undone
  - Failures in transaction, example: statement error, violating unique constraint.
  - System failures, OS crashed.

ATOMICITY

CONSISTENCY

ISOLATION

DURABILITY

- If a transaction is run starting from a database in a consistent state, the database must be consistent at the end of the transaction.
  - Database constraints
    - PRIMARY KEY key constraint, UNIQUE, NOT NULL, FOREIGN KEY referential integrity, CHECK
  - Business constrains

ATOMICITY | **CONSISTENCY** | ISOLATION | DURABILITY

- The database may at some point be in an *inconsistent state*.
- Inconsistencies are not visible in a database system (ensured by *atomicity*).
- The old values of any data on which a transaction performs is written to a log file used by a

  → *recovery system*

ATOMICITY

CONSISTENCY

ISOLATION

DURABILITY

- The database system must ensure that transactions run without interference.
  - For any pair of transactions T_i, T_j,

  first statement of transaction T_i is executed after T_j finished or

  first statement of transaction T_j is executed after T_i finished.

ATOMICITY

CONSISTENCY

ISOLATION

DURABILITY

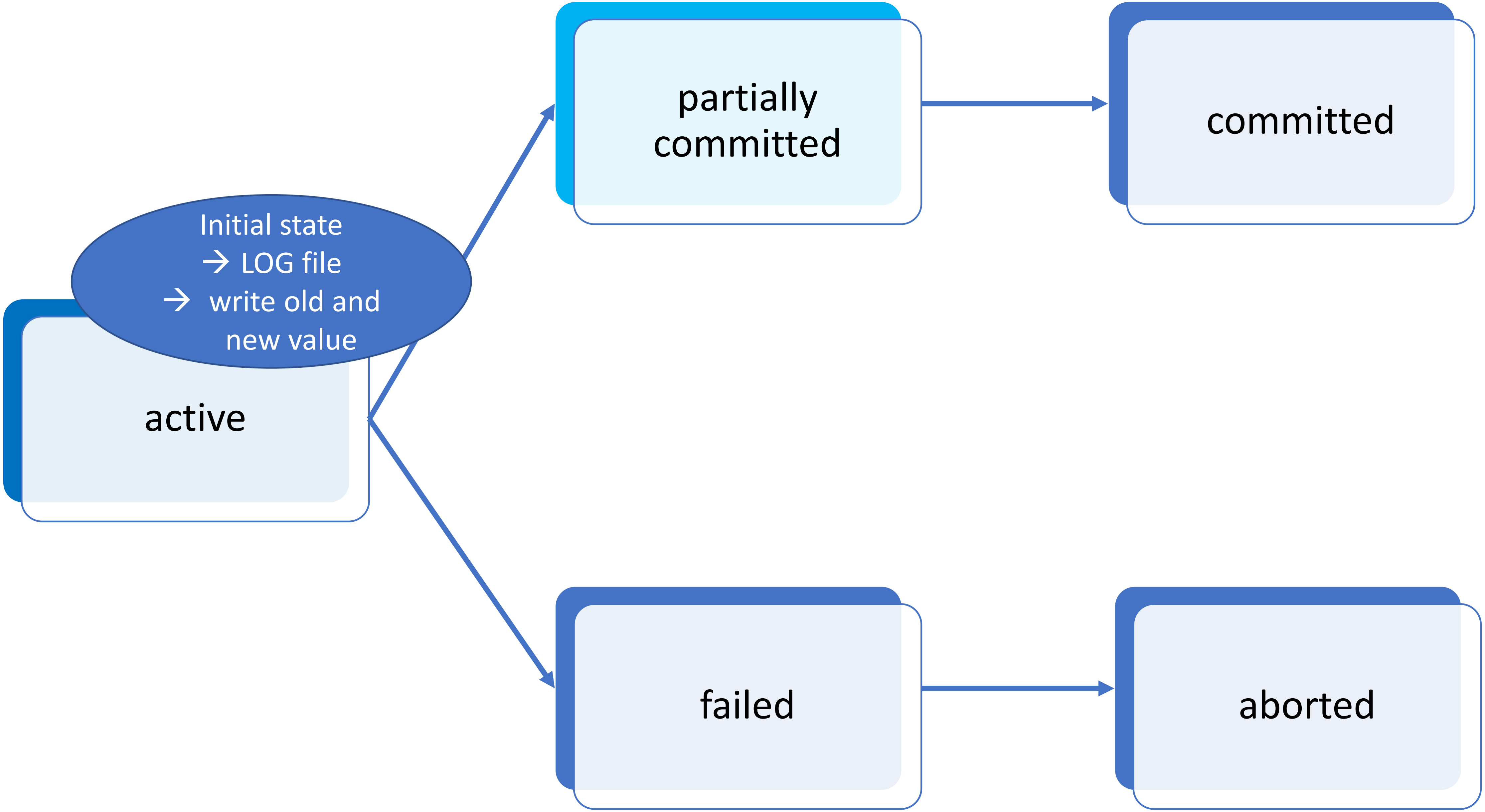
- After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

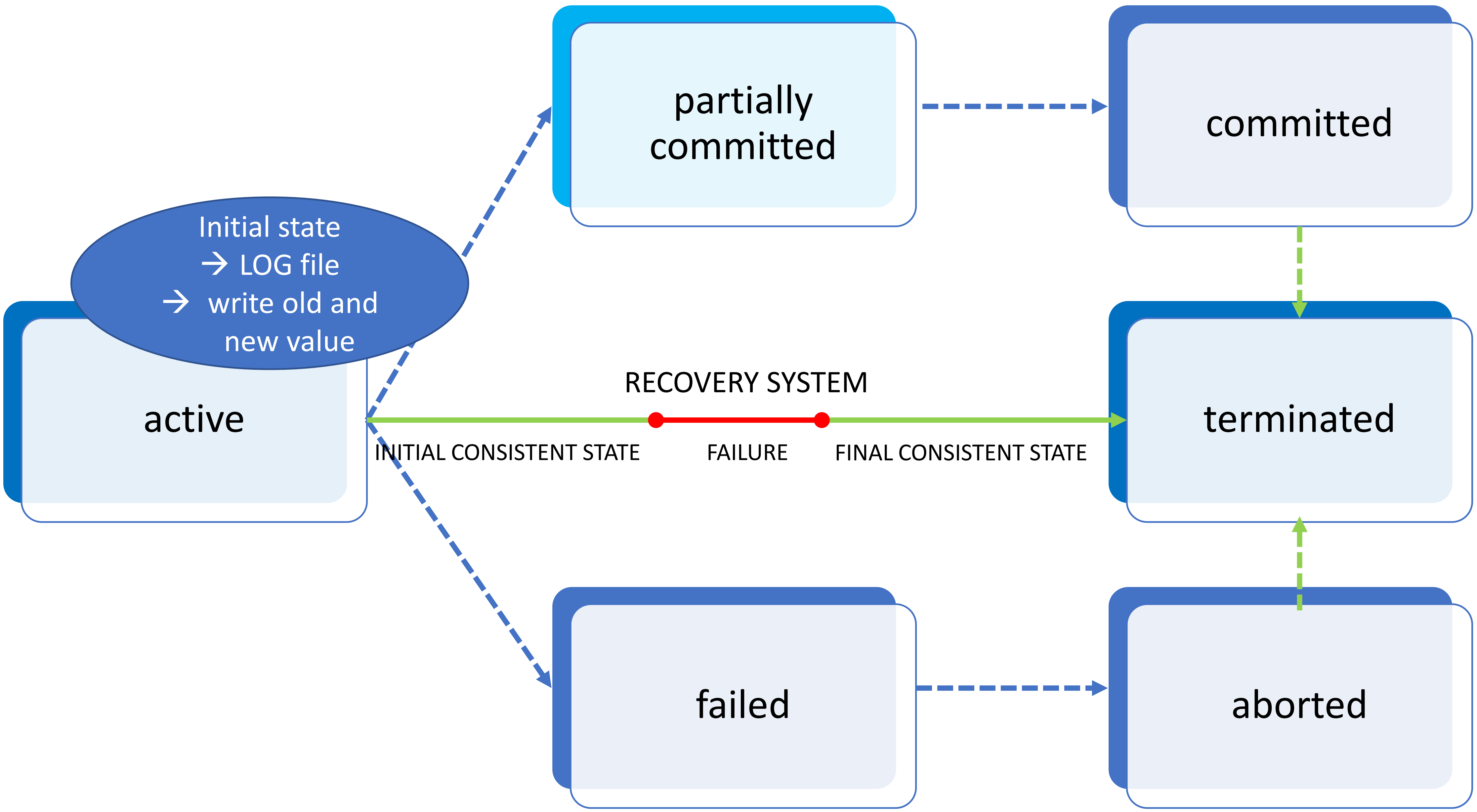
- Information about the updates performed by the transaction is written to disk and used to reconstruct the database after failure.

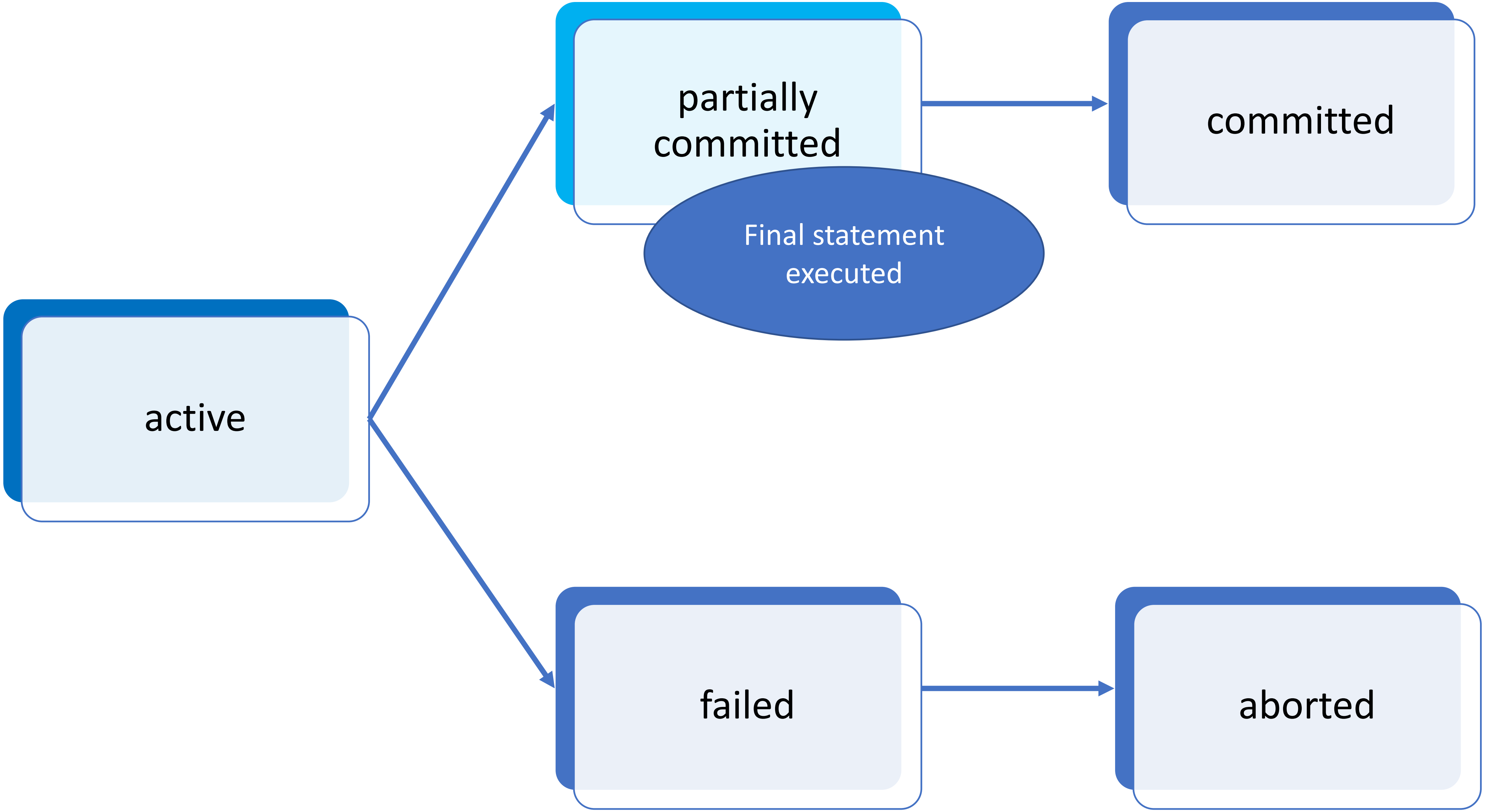
  → *recovery system*

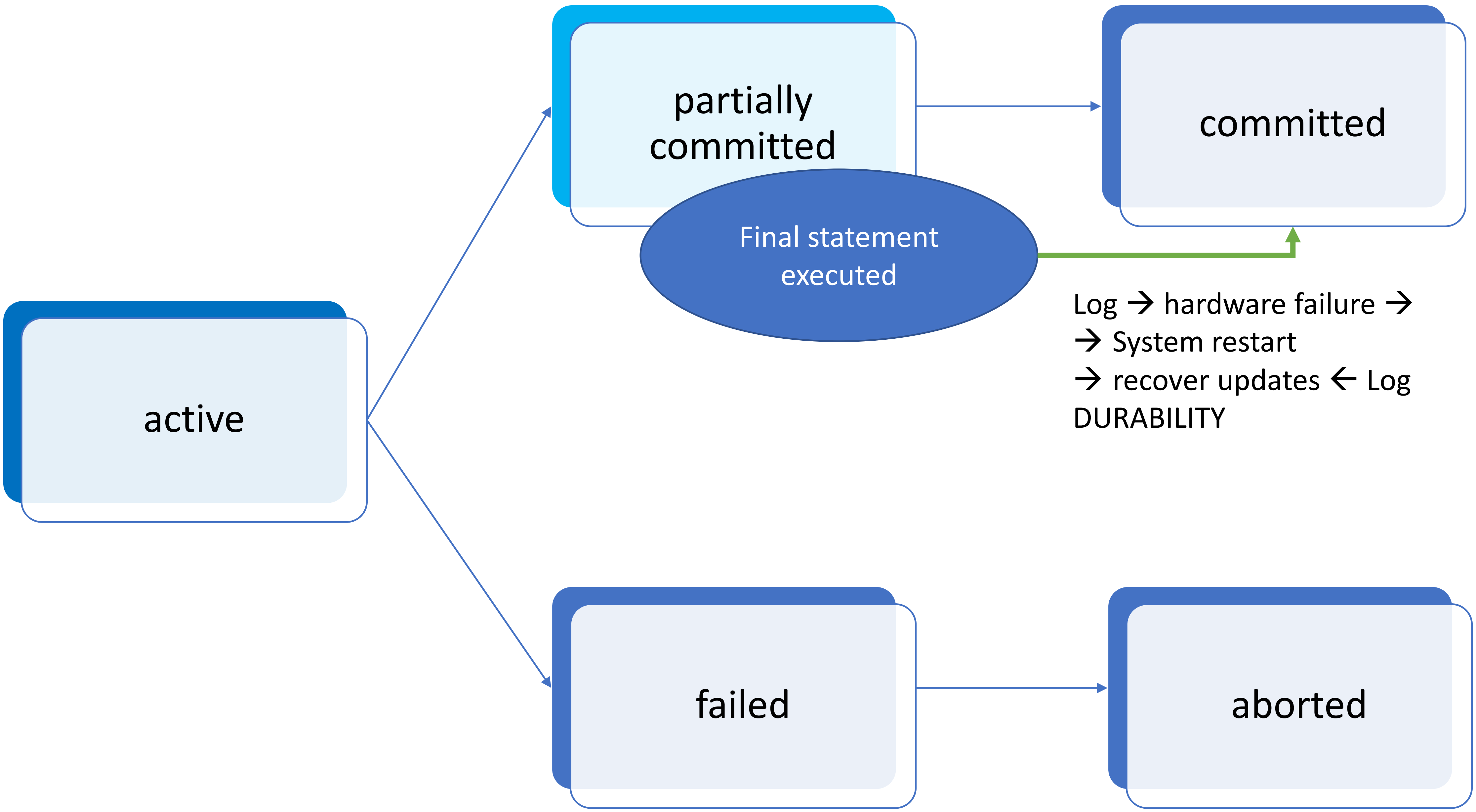
- Please answer www.menti.com 13 52 85 Q1, Q2, Q3, Q4

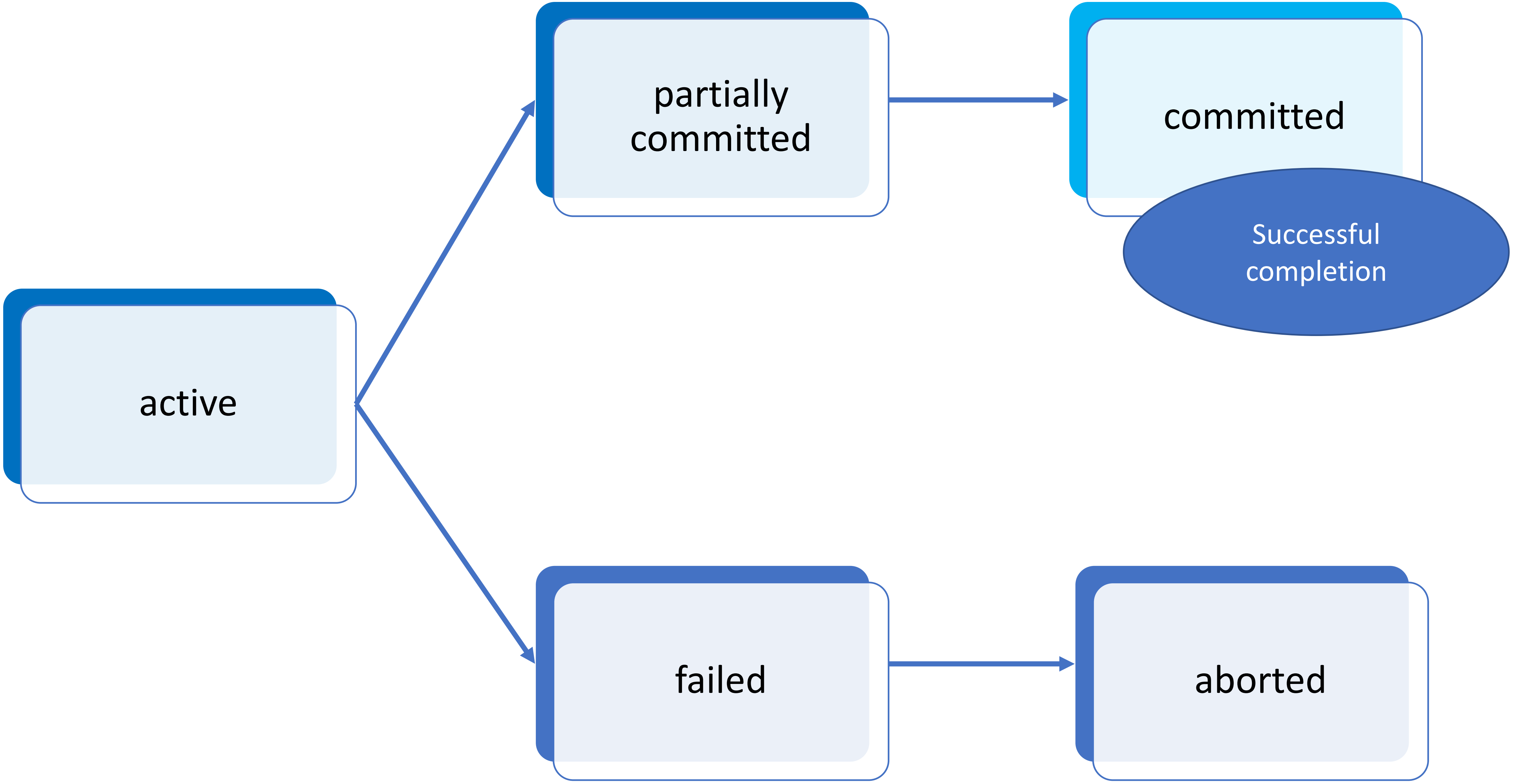
Transaction states

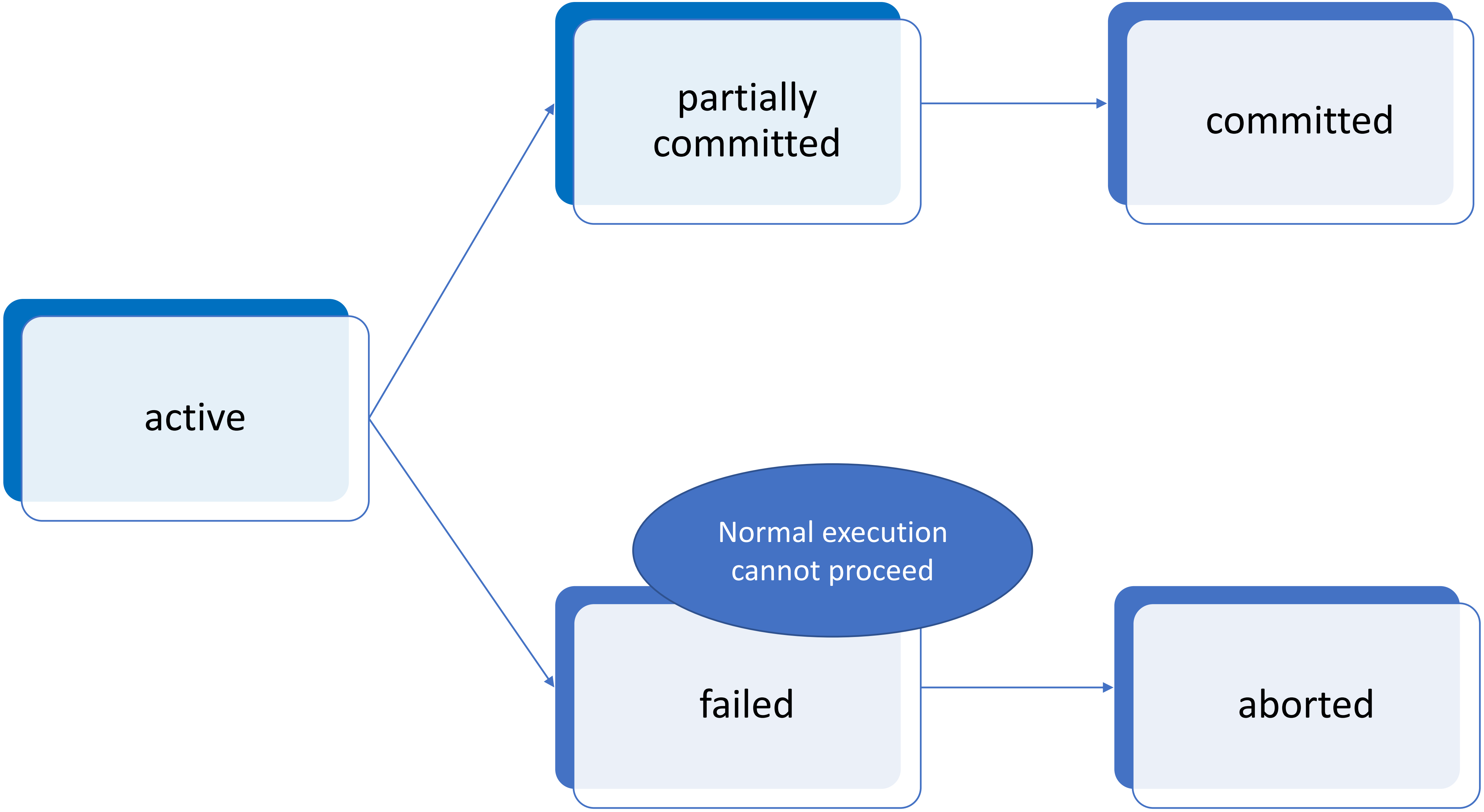

partially committed
committed
Initial state
→ LOG file
→ write old and new value
active
failed
aborted

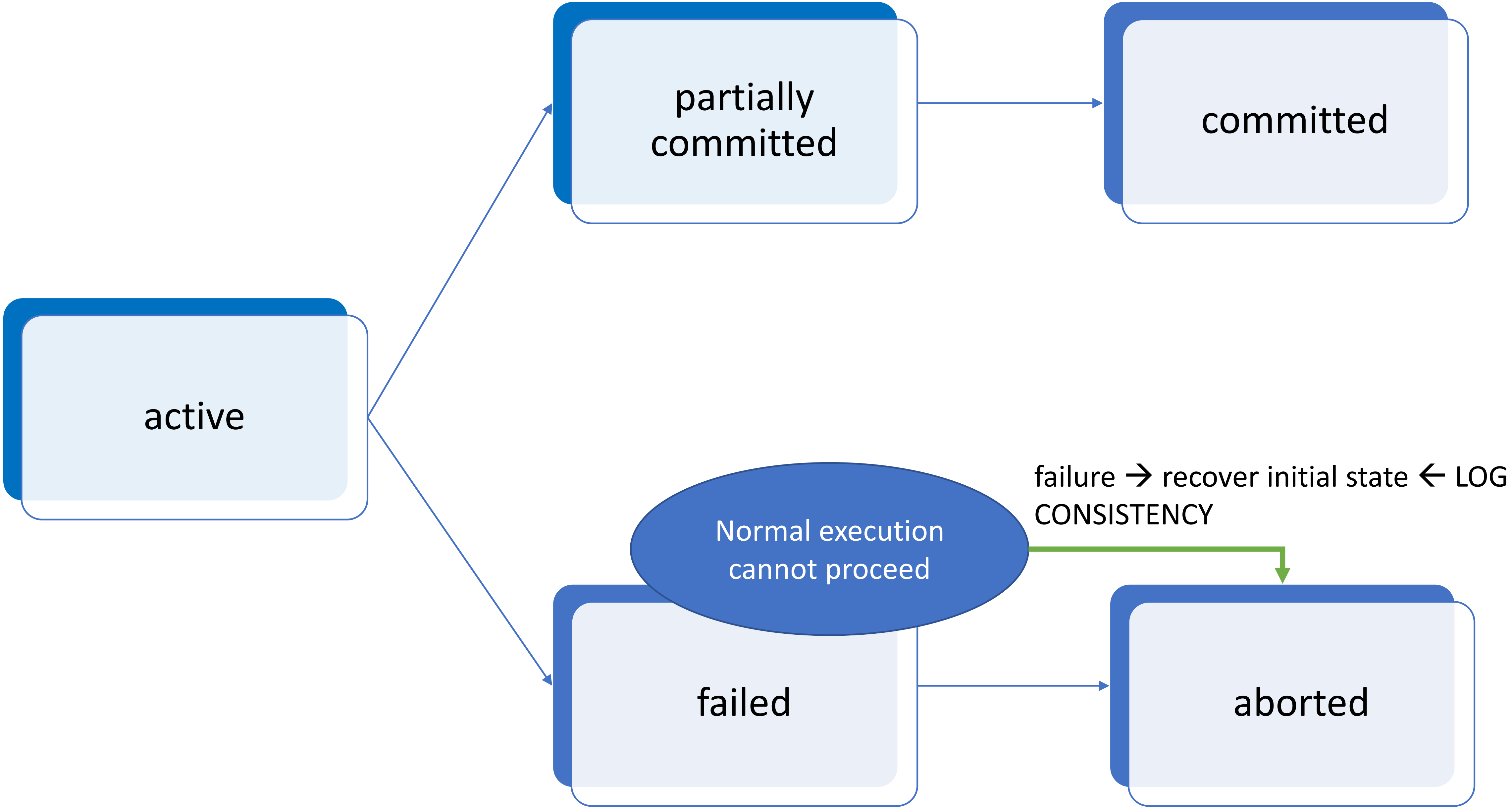

partially committed
committed
Initial state
→ LOG file
→ write old and new value
active
RECOVERY SYSTEM
INITIAL CONSISTENT STATE
FAILURE
FINAL CONSISTENT STATE
terminated
failed
aborted

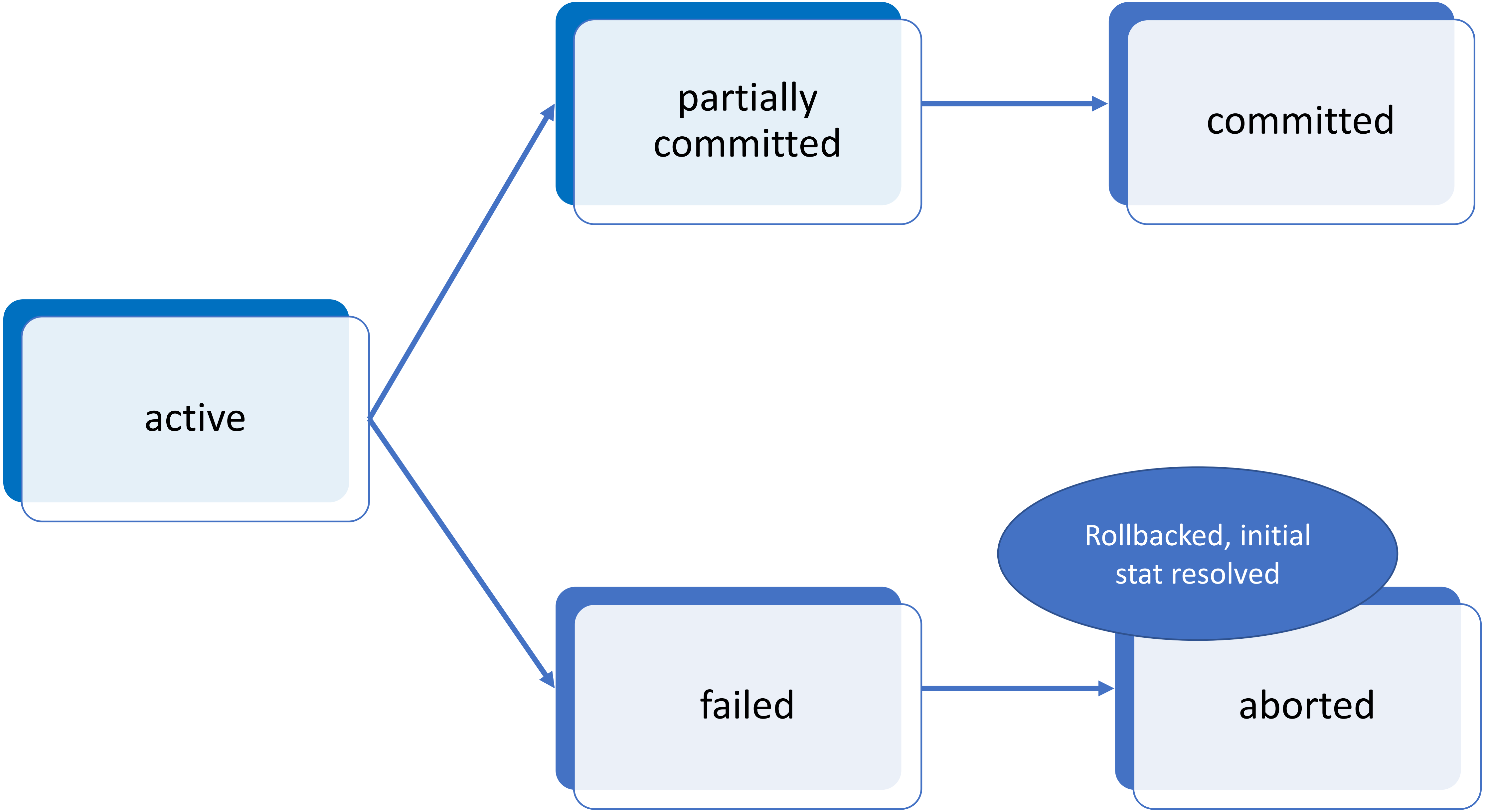

partially committed
committed
Final statement executed
active
failed
aborted


partially committed
committed
Final statement executed
Log → hardware failure →
→ System restart
→ recover updates ← Log
DURABILITY
active
failed
aborted


partially committed
committed
Successful completion
active
failed
aborted


partially committed
committed
active
Normal execution cannot proceed
failed
aborted


partially committed
committed
active
Normal execution cannot proceed
failure → recover initial state ← LOG
CONSISTENCY
failed
aborted


partially committed
committed
active
Rollbacked, initial stat resolved
failed
aborted

Concurrent transactions

Concurrent transactions

- Reduce response time: time for a transaction to be completed.

- Improved workload/resource utilization.

- ISOLATION may be violated → as a result database may be found in an inconsistent state
  - → *Concurrency control*

Concurrent transactions - *conflicts*

- Serial execution preserves consistency, assuming that transactions preserve consistency.
  - first statement of transaction T_i is executed after T_j finished or
  - first statement of transaction T_j is executed after T_i finished
  - single threaded transactions

- Instructions I of T_i and J of T_j conflict ⬄ there exists a *data* accessed by both I and J, and at least one of I an J write *data*.

1. I = read(data)	J = read(data)	I and J don't conflict.
2. I = read(data)	J = write(data)	conflict
3. I = write(data)	J = read(data)	conflict
4. I = write(data)	J = write(data)	conflict

Concurrent transactions -- Schedules

- Schedules: sequences of instructions that specify the chronological order in which instructions of concurrent transactions are executed
  - A schedule for a set of transactions must consist of all instructions of those transactions.
  - A schedule must preserve the order in which the instructions appear in each individual transaction.

  - A transaction that successfully completes its execution will have a commit instructions as the last statement
    - By default transaction assumed to execute commit instruction as its last step.
  - A transaction that fails to successfully complete its execution will have an abort instruction as the last statement.

Schedules example S1

- Serial execution.
- No conflicts.
- DB in consistent state
  - A.new + B.new = A.old + B.old

T1	T2
read (A) A := A - 50 write (A) read (B) B := B + 50 write (B) commit	
	read (A) temp := A * 0.1 A := A - temp write (A) read (B) B := B + temp write (B) commit

Schedules example S2

- Not a serial execution.
- Equivalent to Schedule S1.
- DB in consistent state
  - A.new + B.new = A.old + B.old

T1	T2

`read (A)`
`A := A - 50`
`write (A)`	

	`read (A)`
	`temp := A * 0.1`
	`A := A - temp`
	`write (A)`

`read (B)`
`B := B + 50`
`write (B)`
`commit`	

	`read (B)`
	`B := B + temp`
	`write (B)`
	`commit`

Concurrent transactions

- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence:

    1. Conflict serializability

    2. View serializability

Concurrent transactions

- A (possibly concurrent) schedule is serializable if it is equivalent to a serial schedule. Different forms of schedule equivalence:

  1. Conflict serializability

     If a schedule S can be transformed into a schedule S’ by a series of swaps of non-conflicting instructions, we say that S and S’ are conflict equivalent.

  2. View serializability

Schedules example S2

- Not a serial execution.
- Equivalent to Schedule S1.
- DB in consistent state
  - A.new + B.new = A.old + B.old

no conflict, no data item is updated by both blocks, by swapping the two blocks we obtain S1

T1	T2
read (A) A := A - 50 write (A)	
	read (A) temp := A * 0.1 A := A - temp write (A)
read (B) B := B + 50 write (B) commit	
	read (B) B := B + temp write (B) commit

Schedules example S3

- Not a serial execution.
- Not equivalent to Schedule S1.
- DB in inconsistent state
  - A.new + B.new != A.old + B.old

conflict, A is updated by both blocks

T1	T2
read (A) A := A - 50	
	read (A) temp := A * 0.1 A := A - temp write (A)
write (A) read (B) B := B + 50 write (B) commit	
	read (B) B := B + temp write (B) commit

Concurrent transactions

1. Conflict serializability
2. View serializability

   Let S and S’ be 2 schedules with the same set of transactions. S and S’ are view equivalent if the following 3 conditions are met, for each data item Q:

   - If in schedule S, transaction Ti reads the initial value of Q, then in schedule S’ also transaction Ti must read the initial value of Q.
   - If in schedule S transaction Ti executes read(Q), and that value was produced by transaction Tj (if any), then in schedule S’ also transaction Ti must read the value of Q that was produced by the same write(Q) operation of transaction Tj .
   - The transaction (if any) that performs the final write(Q) operation in schedule S must also perform the final write(Q) operation in schedule S’.

   View equivalence is also based purely on reads and writes alone.

Concurrent transactions

- Test serializability:
  1. Conflict serializability

     - Consider some schedule of a set of transactions T1, T2, ..., Tn
     - Precedence graph — a direct graph where the vertices are the transactions (names).
     - We draw an arc from Ti to Tj if the 2 transaction conflict, and Ti accessed the data item on which the conflict arose earlier.
     - We may label the arc by the item that was accessed.
     - A schedule is CS if and only if its precedence graph is acyclic.
     - If precedence graph is acyclic, the serializability order can be obtained by a **topological sorting** of the graph.

Concurrent transactions

- Test serializability :

    2. View serializability

        - The problem of checking if a schedule is view serializable falls in the class of NP-complete problems. Thus, existence of an efficient algorithm is extremely unlikely.

        - Practical algorithms that just check some sufficient conditions for view serializability can still be used.

Please answer www.menti.com 13 52 85 Q5

Isolation levels

Isolation levels

- **Isolation:** execute a transaction ***as if*** there are no other concurrent transactions running simultaneously.
  - Prevent read or write of incorrect, temporary, aborted data processed by concurrent transactions
- **Isolation levels:** trade off between *perfect* isolation and performance
  - response time: time before a transaction completes
  - throughput: number of transactions per second

Level **Serializability**, perfect isolation

- The final state of the database is equivalent to a state of the database if the transactions were run sequentially.
  - serializable schedule

- Way of obtaining serializability:
  - locking
  - timestamp validation
  - multi-versioning

Transactions errors

T1	T2
select qte into **:nS** from stock where n_prod = 100 --nS = 13	
	select qte into **:nS** from stock where n_prod = 100
update stock set qte = **:nS - 1** where n_prod = 100	
	update stock set qte = **:nS - 1** where n_prod = 100
insert into orders(n_prod, qte) values(100, 1) commit	
	insert into orders(n_prod, qte) values(100, 1)

lost-update anomaly

final stock 12!

Transactions errors

T1	T2

`select qte into :nS`
`from stock`
`where n_prod = 100`
`--nS = 13`	

`update stock`
`set qte = :nS -  1`
`where n_prod = 100`	

	`select sum(qte)`
	`into :nO`
	`from orders`
	`where n_prod = 100`
	
	`select qte into :nS`
	`from stock`
	`where n_prod = 100`
	`--nO+nS!=init_stock`

`insert into orders(n_prod,`
`qte) values(100, 1)`
`commit`	

dirty-read anomaly
number of products ordered + qte_stock != initial stock
1 product missing!
Read uncommitted data

Transactions errors

T1	T2

…	`select qte into :nS`
	`from stock`
	`where n_prod = 100`
	`--nS = 10`

	`if nS < 15 and nS >= 10`
	`  insert into restock(n_prod, qte)`
	`  values(100, 5)`

`update stock`
`set qte = :nS - 1`
`where n_prod = 100`
 	
`insert into orders`
`(n_prod, qte)`
`values(100, 1)`
 	
`commit`	

	`select qte into :nS`
	`from stock`
	`where n_prod = 100`
	
	`if nS < 10`
	`  insert into restock(n_prod, qte)`
	`  values(100, 15)`

non-repeatable read anomaly
only one insert into restock is needed!
read twice, different values

Transactions errors

phantom-read anomaly
AVG > MAX!

new lignes inserted

T1	T2
	`select MAX(qte) into :`**`max`**` from orders where n_prod = 100`
`insert into orders(n_prod,qte) values(100, `**`789455`**`) commit`	
	`select AVG(qte) into :`**`avarage`**` from orders where n_prod = 100`

Transactions errors

T1	T2
select qte into **:nS** from stock where n_prod = 100 --nS = **13**	
update stock set qte = **:nS - 1** where n_prod = 100	
	select qte into **:nS** from stock where n_prod = 100 --nS = **12**
	update stock set qte = **:nS - 1** where n_prod = 100 --nS = **11**
abort	
	insert … commit

dirty-write anomaly
final stock 11! In the first transaction, the stock returns to 13. Only one update should decrease the number of products.

Isolation levels

- weaker the isolation level → more anomalies may occur

LEVEL \ ERROR	lost-update	dirty-reads	non-repeatable reads	phantom
READ UNCOMMITTED	✗	✓	✓	✓
READ COMMITTED	✗	✗	✓	✓
REPEATABLE READ	✗	✗	✗	✓
SERIALIZABLE	✗	✗	✗	✗

Isolation levels

	ERROR	lost-update	dirty-reads	non-repeatable reads	phantom
LEVEL					
REPEATABLE READ		✗	✗	✗	✓

- read only committed
- between two reads of an item by a transaction, no other transaction is allowed to update it.
- a transaction may find other data inserted by a committed transaction

Isolation levels

LEVEL \ ERROR	lost-update	dirty-reads	non-repeatable reads	phantom
READ COMMITTED	✗	✗	✓	✓

- read only committed
- does not require repeatable reads. Between two reads of a data item by the transaction, another transaction may have updated the data item and committed.

Isolation levels

ERROR / LEVEL	lost-update	dirty-reads	non-repeatable reads	phantom
READ UNCOMMITTED	✗	✓	✓	✓

- allows uncommitted data to be read
- all the isolation levels prevent writes to a data item that has already been written by another transaction not yet committed or aborted (rollbacked).

- Please answer www.menti.com 13 52 85 Q6, Q7

Achieving isolation

- Versioning
    - Transactions read from a “snapshot” of the database.
- Locking
- Timestamp

Locking

- Locks prevent destructive interactions between transactions accessing the same resource.
  - Shared access to read
  - Exclusive access to read and write

  - Locks (Shared, Shared) compatible.
  - Locks (Shared, Exclusive) not compatible.

- A transaction waits until all incompatible locks held by other transactions are released.

- https://oracle-base.com/articles/misc/deadlocks
- https://docs.oracle.com/cd/B19306_01/server.102/b14220/consist.htm

Snapshot isolation

- Snapshot of the database at the beginning of each transaction.
- The transaction operates only on that snapshot.
- The snapshot consists only of committed values.
- Updates are kept in transaction workspace until commit.
- Implemented with timestamp-versioning

Consistency levels

To be added, more info in the following video.

BASE

NoSql consistency model

To be added, more info in the following video.

Normalization

COURSE 4: Databases

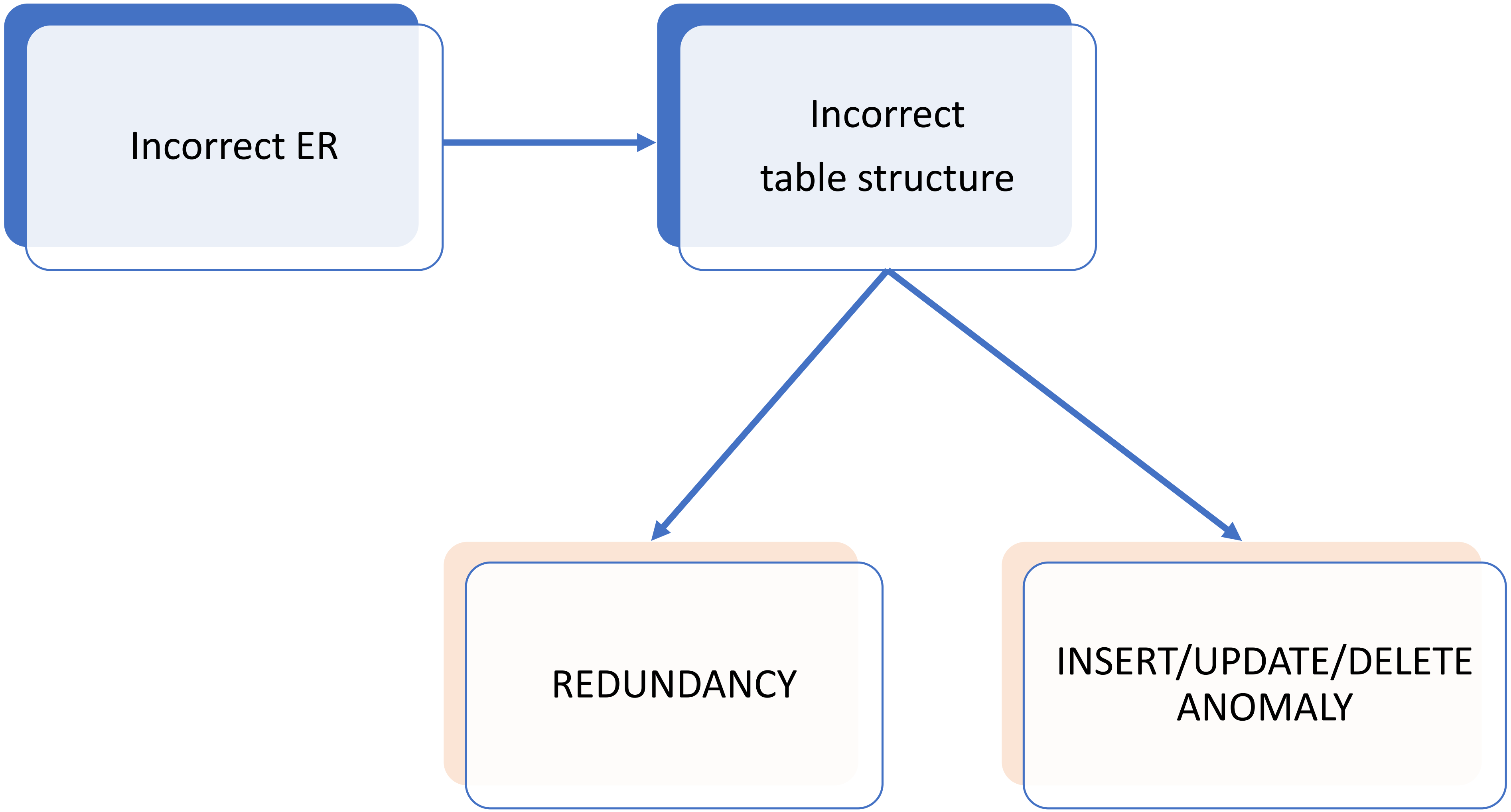
Normalization

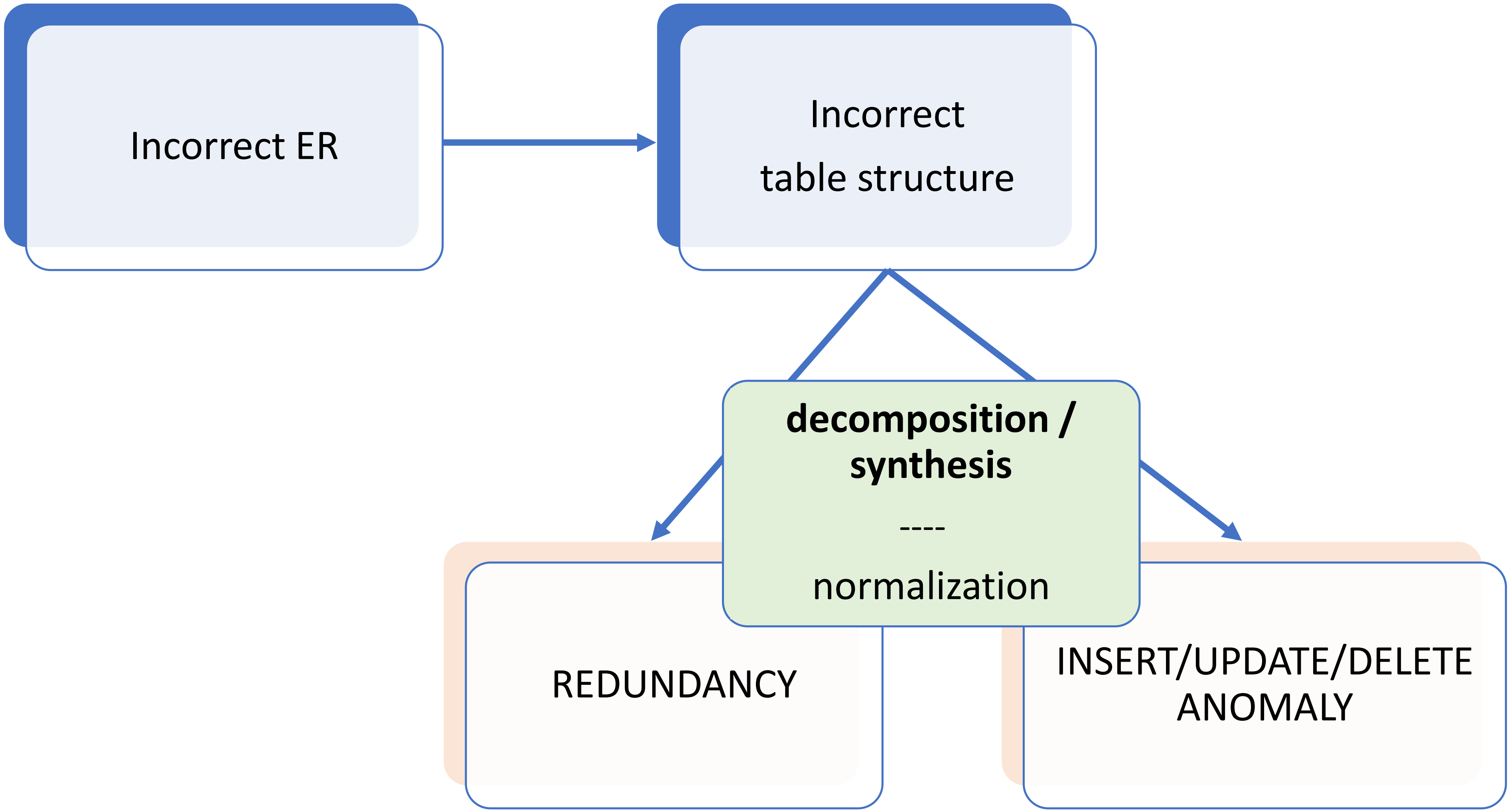
when and why

Normalization

- Informal:
  - Organize data in a relational database in order to avoid redundancy and data manipulation anomalies.

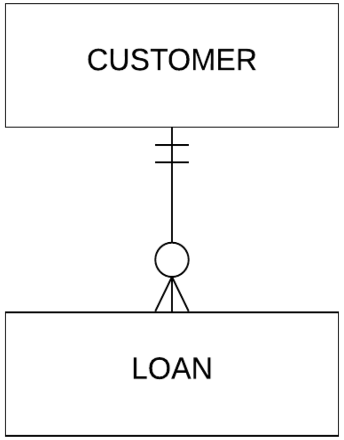
  - Decompose a relation (table) without loosing information.


Incorrect ER
Incorrect
table structure
REDUNDANCY
INSERT/UPDATE/DELETE
ANOMALY


Incorrect ER
Incorrect
table structure
decomposition /
synthesis
----
normalization
REDUNDANCY
INSERT/UPDATE/DELETE
ANOMALY

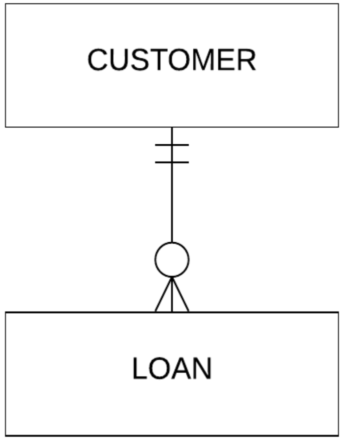
Normalization

- Avoid redundancy



Normalization

- Avoid redundancy

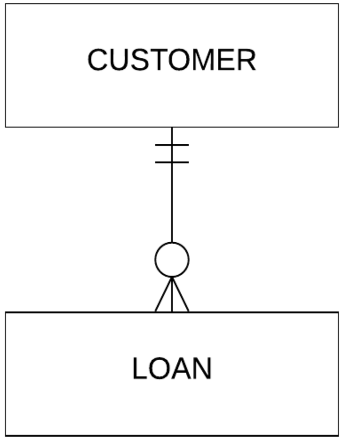


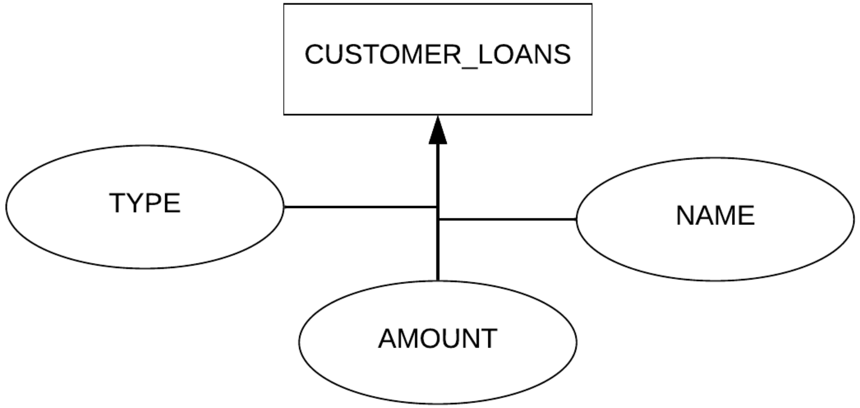
CUSTOMER			
CUSTOMER_ID	**LAST_NAME**	**…**	**….**
1	Smith	…	….
2	Green	…	….
3	Avery	…	….

LOAN			
LOAN_ID	**CUSTOMER_ID**	**AMOUNT**	**DATE**
101	1	125000	18/04/21
102	1	25000	14/04/22
103	2	12500	03/05/21
127	2	20000	…
389	3	75000	…

Normalization

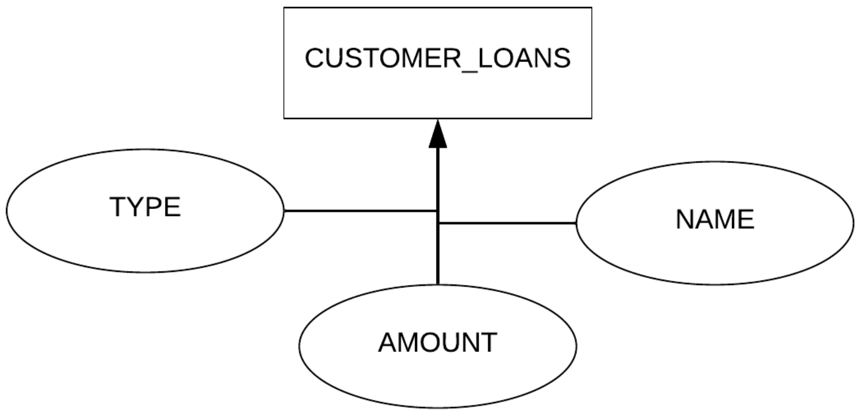
- Avoid redundancy





Normalization

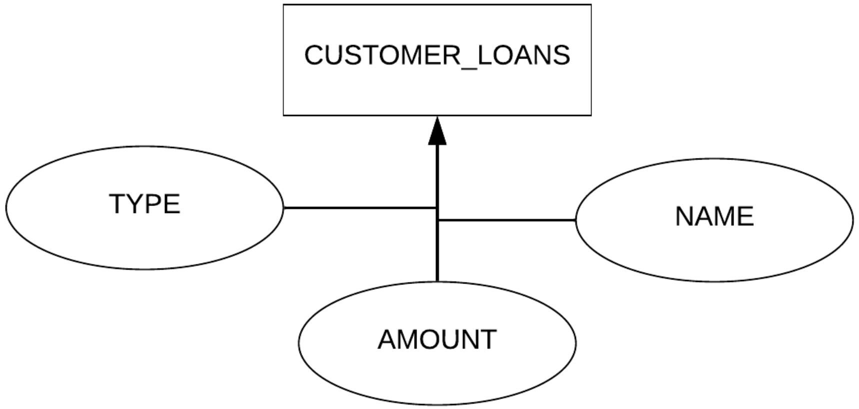
- Avoid redundancy



CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
2	Green	103	credit card	12500
2	Green	127	mortgage	20000
3	Avery	389	mortgage	75000
3	Avery	486	credit card	5000
3	Avery	769	mortgage	45000

Normalization

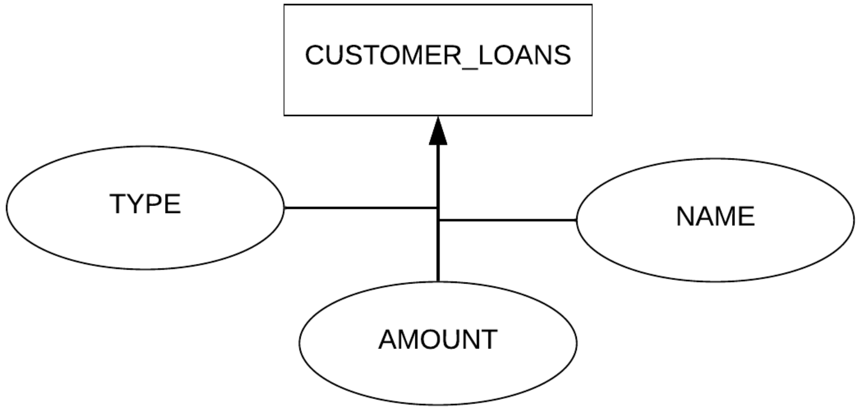
- INSERT anomaly



CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
2	Green	103	credit card	12500
2	Green	127	mortgage	20000
3	Avery	389	mortgage	75000
3	Avery	486	credit card	5000
4	Stark	???	null	null

Normalization

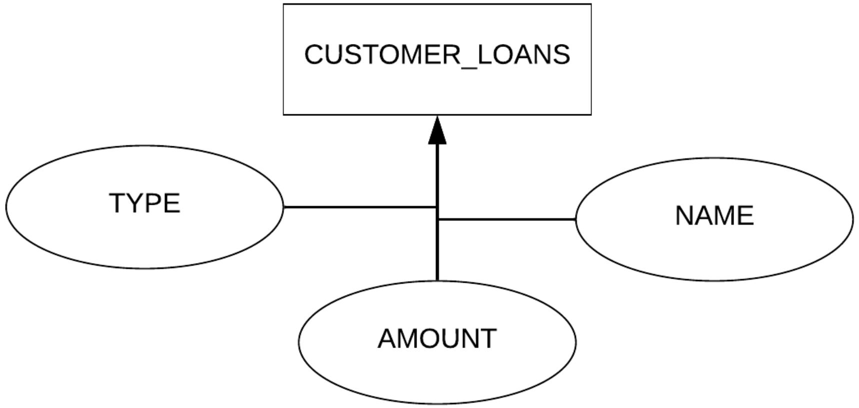
- UPDATE anomaly



CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
2	Green	103	credit card	12500
2	Green	127	mortgage	20000
3	Avery	389	mortgage	75000
3	Avery	486	credit card	5000
3	Avery	769	mortgage	45000

Normalization

- DELETE anomaly



CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
2	Green	103	credit card	12500
2	Green	127	mortgage	20000
3	Avery	389	mortgage	75000
3	Avery	486	credit card	5000
4	Stark	700	mortgage	45000

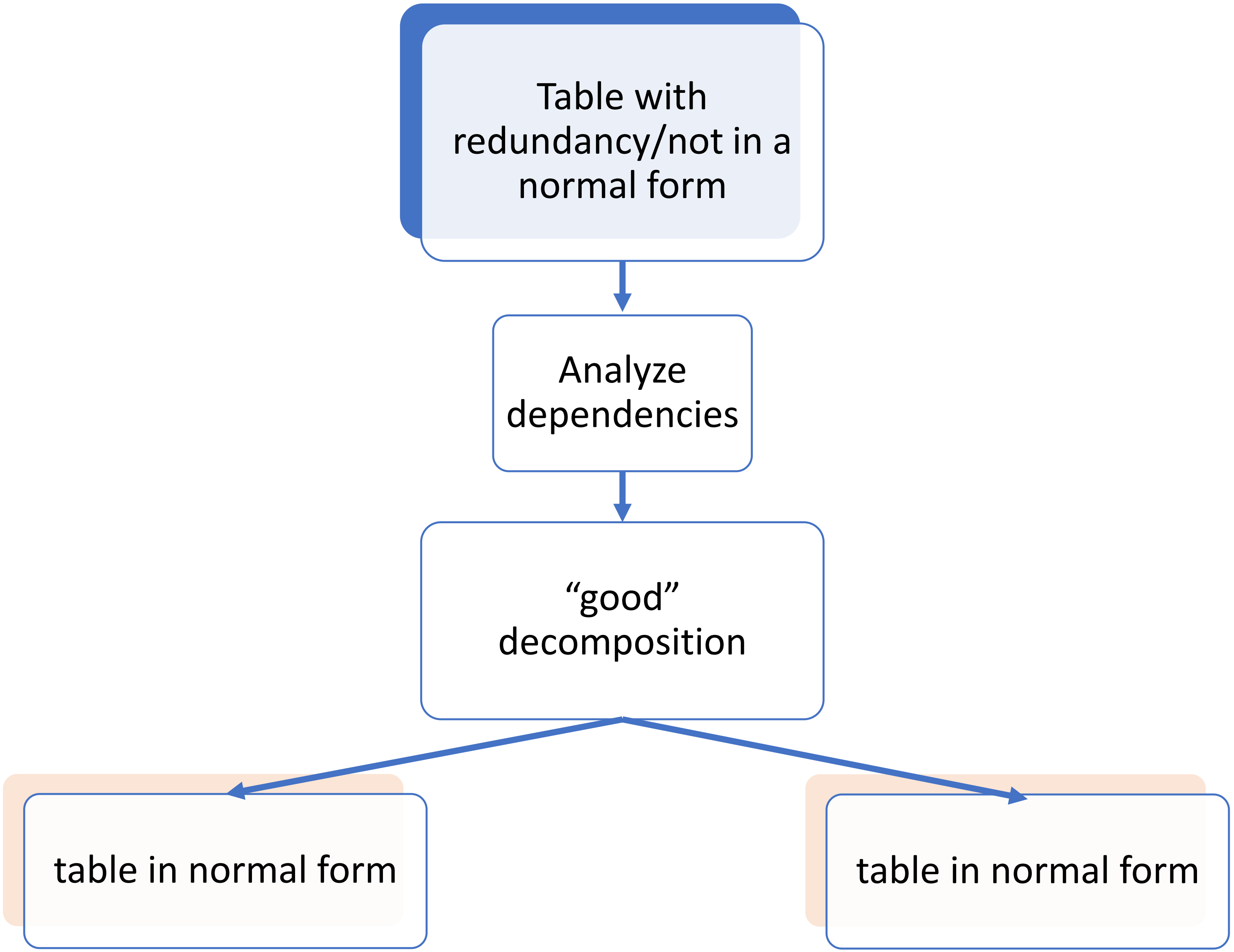


Table with redundancy/not in a normal form
Analyze dependencies
“good” decomposition
table in normal form
table in normal form

Decomposition

Decomposition Step 1: Projection

- $S_1 = \Pi_{(NAME, LOANID, TYPE, AMOUNT)} R$

CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
2	Green	103	credit card	12500
3	Smith	389	mortgage	75000

- $S_2 = \Pi_{(CUSTOMERID, NAME)} R$

NAME	LOAN_ID	TYPE	AMOUNT
Smith	101	mortgage	125000
Smith	102	credit card	25000
Green	103	credit card	12500
Smith	389	mortgage	75000

CUSTOMER_ID	NAME
1	Smith
1	Smith
2	Green
3	Smith

Decomposition Step 2: Join

CUSTOMER_ID	NAME
1	Smith
1	Smith
2	Green
3	Smith

NAME	LOAN_ID	TYPE	AMOUNT
Smith	101	mortgage	125000
Smith	102	credit card	25000
Green	103	credit card	12500
Smith	389	mortgage	75000

- Lossy decomposition
  $S_1 \bowtie S_2 \supseteq R$

CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
1	Smith	389	mortgage	75000
3	Smith	101	mortgage	125000
3	Smith	102	credit card	25000
3	Smith	389	mortgage	75000
2	Green	103	credit card	12500

Decomposition Step 1: Projection

- $S_1 = \Pi_{(CUSTOMERID, LOANID, TYPE, AMOUNT)}$

CUSTOMER_ID	LOAN_ID	TYPE	AMOUNT
1	101	mortgage	125000
1	102	credit card	25000
2	103	credit card	12500
3	389	mortgage	75000

CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
2	Green	103	credit card	12500
3	Smith	389	mortgage	75000

CUSTOMER_ID	NAME
1	Smith
1	Smith
2	Green
3	Smith

- $S_2 = \Pi_{(CUSTOMERID, NAME)} R$

Decomposition Step 2: Join

CUSTOMER_ID	NAME
1	Smith
1	Smith
2	Green
3	Smith

CUSTOMER_ID	LOAN_ID	TYPE	AMOUNT
1	101	mortgage	125000
1	102	credit card	25000
2	103	credit card	12500
3	389	mortgage	75000

- Lossless decomposition
  $S_1 \bowtie S_2 = R$

CUSTOMER_ID	NAME	LOAN_ID	TYPE	AMOUNT
1	Smith	101	mortgage	125000
1	Smith	102	credit card	25000
3	Smith	389	mortgage	75000
2	Green	103	credit card	12500

Decomposition

- lossy decompositions and lossless decompositions.

- Lossy: R → decompose(R): S1, S2 → recompose(S1,S2) $\supseteq$ R

  lossy =/= less data, (less is more!)

  lossy = lost information

- Lossless R → decompose(R): S1, S2 → recompose(S1,S2) = R

Decomposition

- Lossy

$$\prod_{R1} R \bowtie \prod_{R2} R \supseteq \mathrm{R}$$

- Lossless

$$\prod_{R1} R \bowtie \prod_{R2} R = \mathrm{R}$$

Functional dependencies

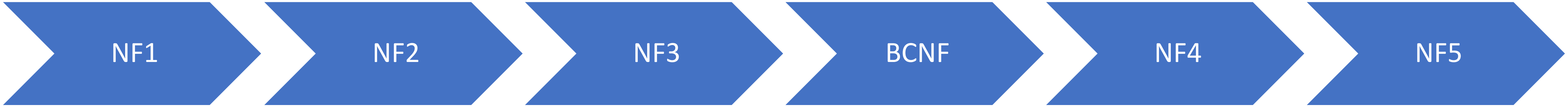
Functional dependencies

CUSTOMER_ID	NAME
1	Smith
1	Smith
2	Green
3	Smith

X	Y	Z	T
X1	Y1	Z1	T1
X1	Y2	Z1	T2
X2	Y2	Z2	T2
X2	Y3	Z2	T3
X3	Y3	Z2	T4

- CUSTOMER_ID -> NAME
- X -> Z
- Z --/--> X
- X -> X

Normal Forms


NF1
NF2
NF3
BCNF
NF4
NF5

NF1

ATOMIC ATTRIBUTES

NF1

- Atomic attributes
- No multi-valued attributes

- The domain of each attribute contains only atomic values and each attribute contains only a value of its domain.

- A relational database is **at least in NF1**

NF1

EMP_ID	NAME	EMAIL
1	Williams	williams@gmail.com williams@yahoo.com
2	Davis	davis@gmail.com davis@academy.com
3	Miller	miller@gmail.com
4	Stewart	stewart@gmail.com office@academy.com

Optional:
Artificial key

Composed key

EMAIL_ID	EMP_ID	EMAIL
1	1	williams@gmail.com
2	1	williams@yahoo.com
3	2	davis@gmail.com
4	2	davis@academy.com
5	3	miller@gmail.com
6	4	stewart@gmail.com
7	4	office@academy.com

NF2

NO PARTIAL DEPENDENCIES

NF2

- Tables in NF1
- No non-key attributes (not part of the key) that depend on a subset of the attributes forming the key.

- There are no partial dependencies.

Functional dependencies

X	Y	Z	T
X1	Y1	Z1	T1
X2	Y1	Z1	T2
X2	Y2	Z2	T3
X2	Y3	Z2	T3
X2	Y3	Z2	T3

X	Y	Z	T
X1	**Y1**	Z1	…
X2	**Y1**	Z1	…
X2	**Y2**	Z2	…
X2	**Y3**	Z2	…
X2	**Y3**	Z2	…

- partial (X,Y) $\rightarrow$ Z
  - Y $\rightarrow$Z

Functional dependencies

X	Y	Z	T
X1	Y1	…	T1
X2	Y1	…	T2
X2	Y2	…	T3
X2	Y3	…	T3
X2	Y3	…	T3

X	Y	Z	T
…	Y1	…	T1
…	Y1	…	T2
…	…	…	…
…	…	…	…
…	…	…	…

X	Y	Z	T
…	…	…	…
X2	…	…	T2
X2	…	…	T3
…	…	…	…
…	…	…	…

- total (X,Y) → T
  - X -/-> T
  - Y -/-> T

Functional dependencies

AIRPORT_ID	AIRPLANE_ID	DEPARTURE	AIRPLANE_MODEL	BOARDING_GATE
1	101	30/03/20 17:00	Boeing 777	42
1	102	02/05/20 09:30	Airbus A320	50
2	201	06/08/20 10:45	Boeing 757	35
2	202	10/10/20 06:20	Airbus A320	10
1	101	06/04/20 16:35	Boeing 777	23

NF2

dependencies

K1 -> X

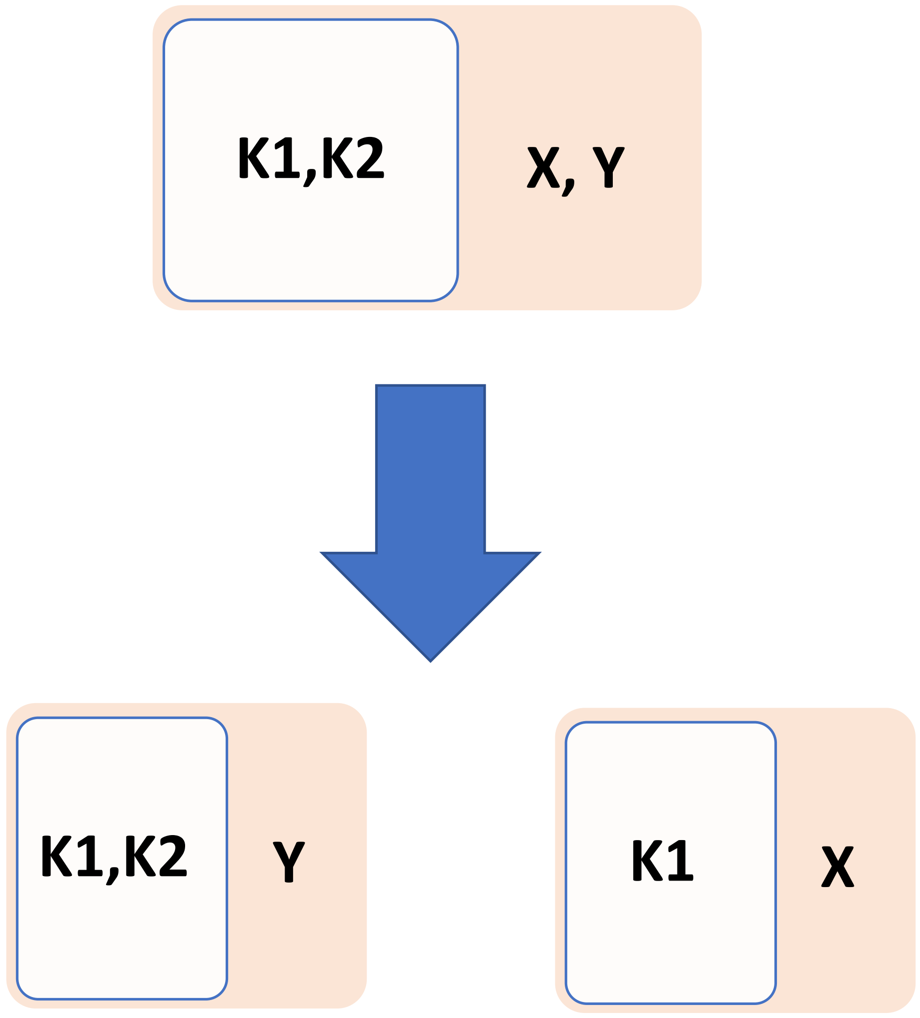
(K1, K2) -> Y

K1,K2 **X, Y**

NF2

dependencies

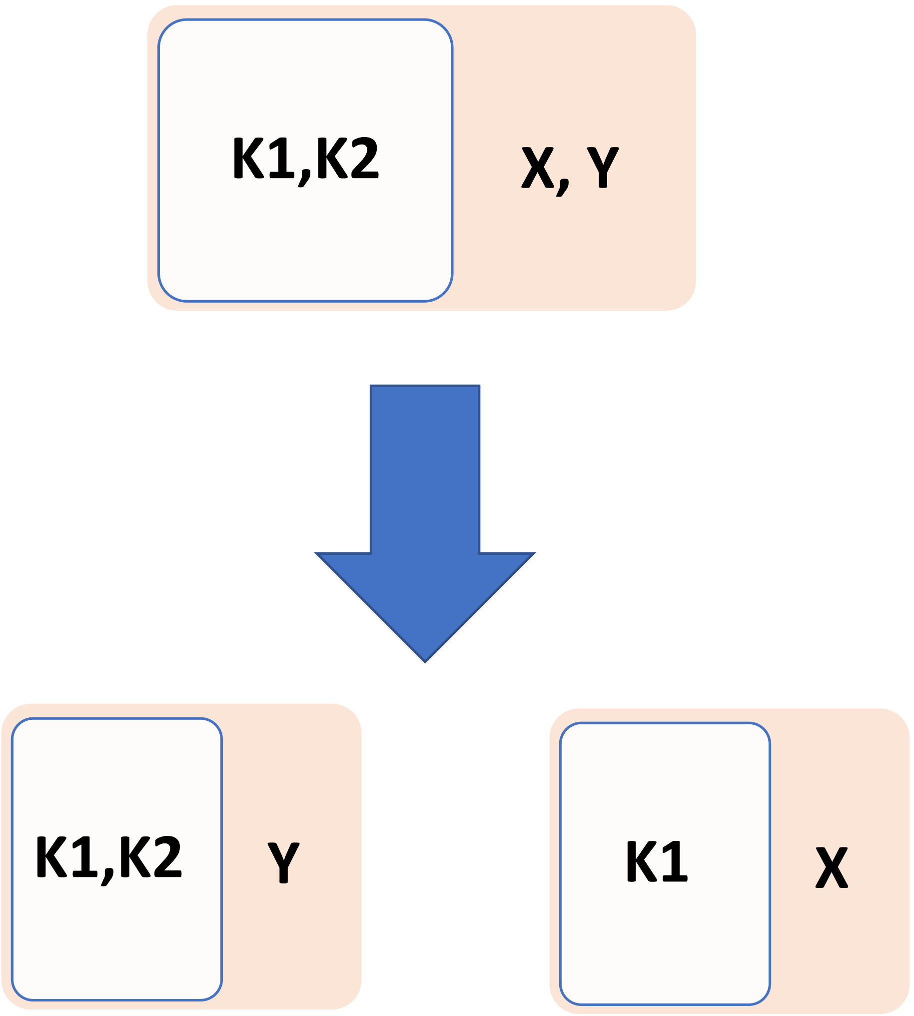
- K1 -> X
- (K1, K2) -> Y



NF2

dependencies

- **K1 -> X**
- **(K1, K2) -> Y**



K1 = AIRPLANE_ID
K2 = AIRPORT_ID, DEPARTURE
Y = BOARDING_GATE
X = AIRPLANE_MODEL

AIRPORT_ID	AIRPLANE_ID	DEPARTURE	AIRPLANE_MODEL	BOARDING_GATE
1	101	30/03/20 17:00	Boeing 777	42
1	102	02/05/20 09:30	Airbus A320	50
2	201	06/08/20 10:45	Boeing 757	35
2	202	10/10/20 06:20	Airbus A320	10
1	101	06/04/20 16:35	Boeing 777	23

AIRPORT_ID	AIRPLANE_ID	DEPARTURE	BOARDING_GATE
1	101	30/03/20 17:00	42
1	102	02/05/20 09:30	50
2	201	06/08/20 10:45	35
2	202	10/10/20 06:20	10
1	101	06/04/20 16:35	23

AIRPLANE_ID	AIRPLANE_MODEL
101	Boeing 777
102	Airbus A320
201	Boeing 757
202	Airbus A320

NF3

NO TRANSITIVE DEPENDENCIES

NF3

- Tables in NF2
- Non-key attributes (not part of the key) depend on the entire key and only on the key.

- There are no transitive dependencies.

AIRPORT_ID	AIRPLANE_ID	DEPARTURE	MODEL	CAPACITY	REVISION_DATE	BOARDING_GATE
1	101	30/03/20 17:00	Boeing 777	451	01/01/2021	42
1	102	02/05/20 09:30	Airbus A320	150	01/03/2020	50
2	201	06/08/20 10:45	Boeing 757	295	03/05/2020	35
2	202	10/10/20 06:20	Airbus A320	150	04/06/2021	10
1	101	06/04/20 16:35	Boeing 777	451	08/09/2020	23

AIRPORT_ID	AIRPLANE_ID	DEPARTURE	BOARDING_GATE
1	101	30/03/20 17:00	42
1	102	02/05/20 09:30	50
2	201	06/08/20 10:45	35
2	202	10/10/20 06:20	10
1	101	06/04/20 16:35	23

AIRPLANE_ID	MODEL	CAPACITY	REVISION_DATE
101	Boeing 777	451	01/01/2021
102	Airbus A320	150	01/03/2020
201	Boeing 757	259	03/05/2020
202	Airbus A320	150	04/06/2021

AIRPLANE_ID	MODEL	CAPACITY	REVISION_DATE
101	Boeing 777	451	01/01/2021
102	Airbus A320	150	01/03/2020
201	Boeing 757	259	03/05/2020
202	Airbus A320	150	04/06/2021

NF3

dependencies

K -> X

X -> Y

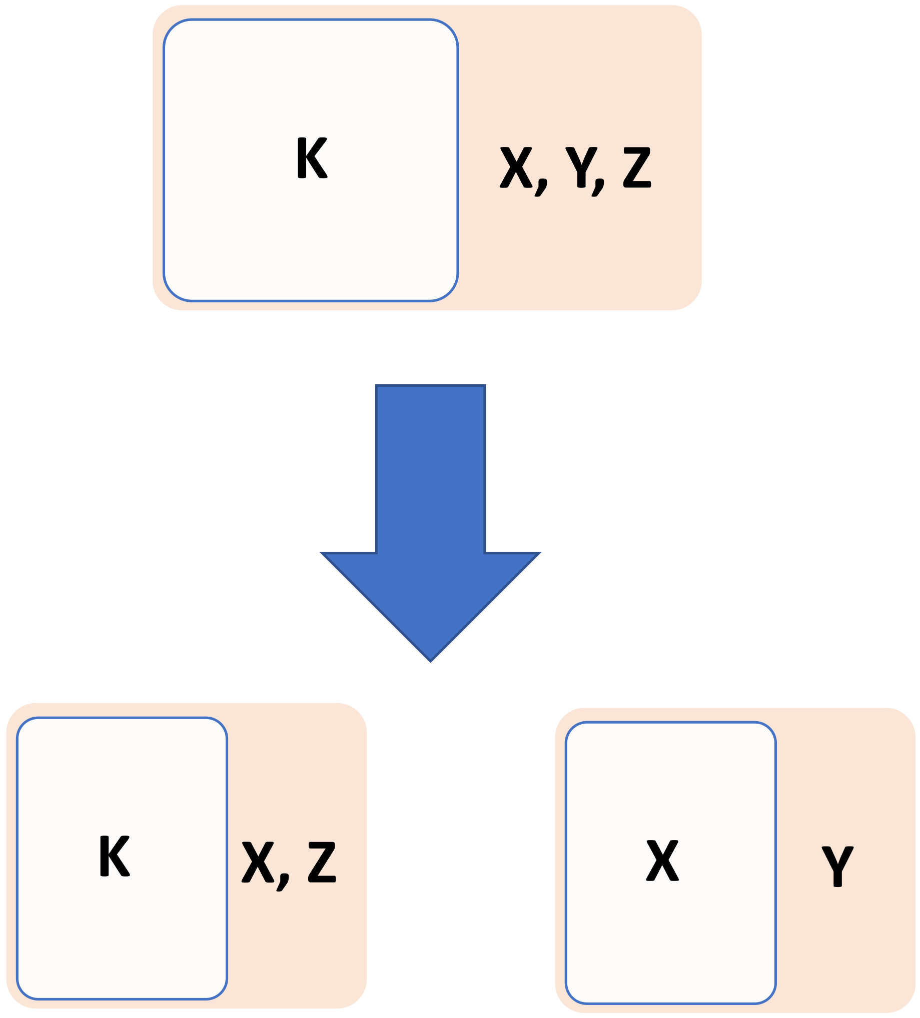
K | **X, Y, Z**

NF3

dependencies

K -> X

X -> Y

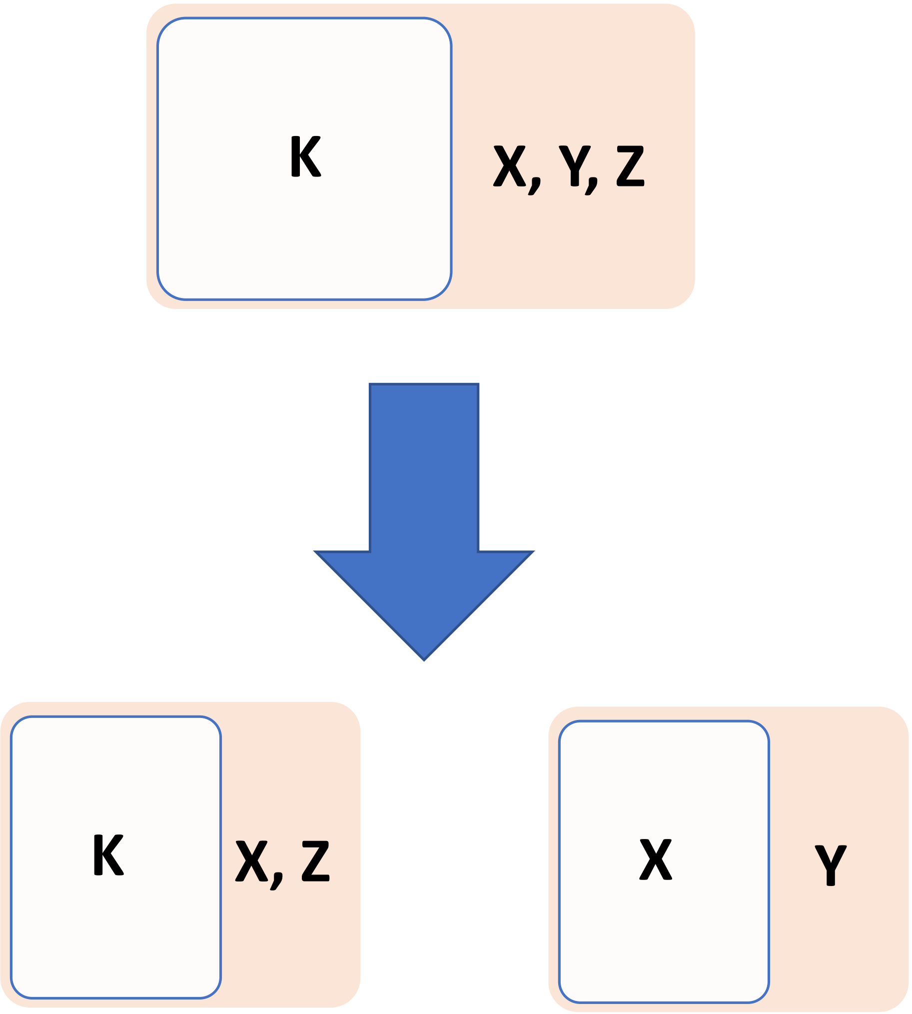


NF3

dependencies

K -> X

X -> Y



K = AIRPLANE_ID
X = AIRPLANE_MODEL
Y = CAPACITY
Z= REVISION_DATE

AIRPLANE_ID	MODEL	CAPACITY	REVISION_DATE
101	Boeing 777	451	01/01/2021
102	Airbus A320	150	01/03/2020
201	Boeing 757	259	03/05/2020
202	Airbus A320	150	04/06/2021

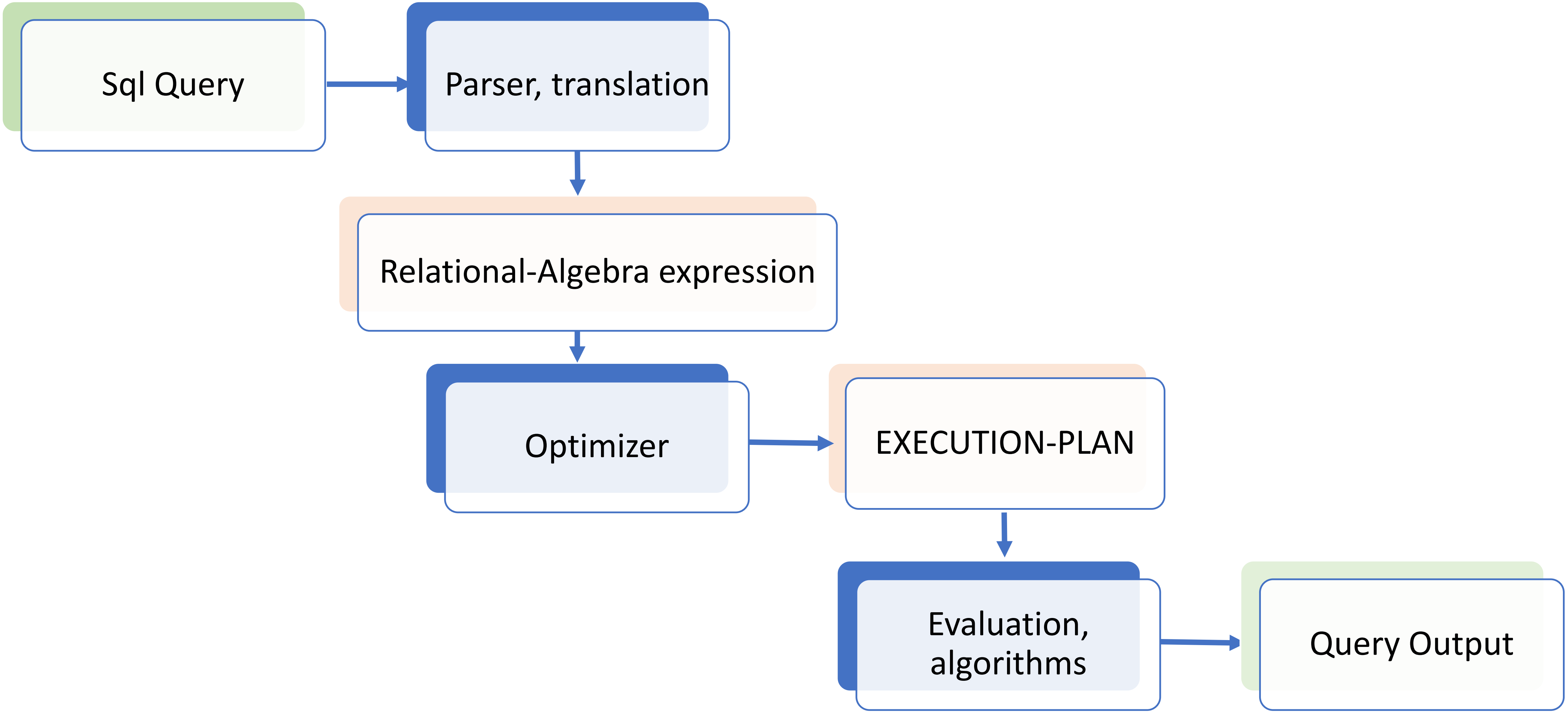
AIRPLANE_ID	MODEL	REVISION_DATE
101	Boeing 777	01/01/2021
102	Airbus A320	01/03/2020
201	Boeing 757	03/05/2020
202	Airbus A320	04/06/2021

MODEL	CAPACITY
Boeing 777	451
Airbus A320	150
Boeing 757	259

Query Optimization

COURSE 6: Databases

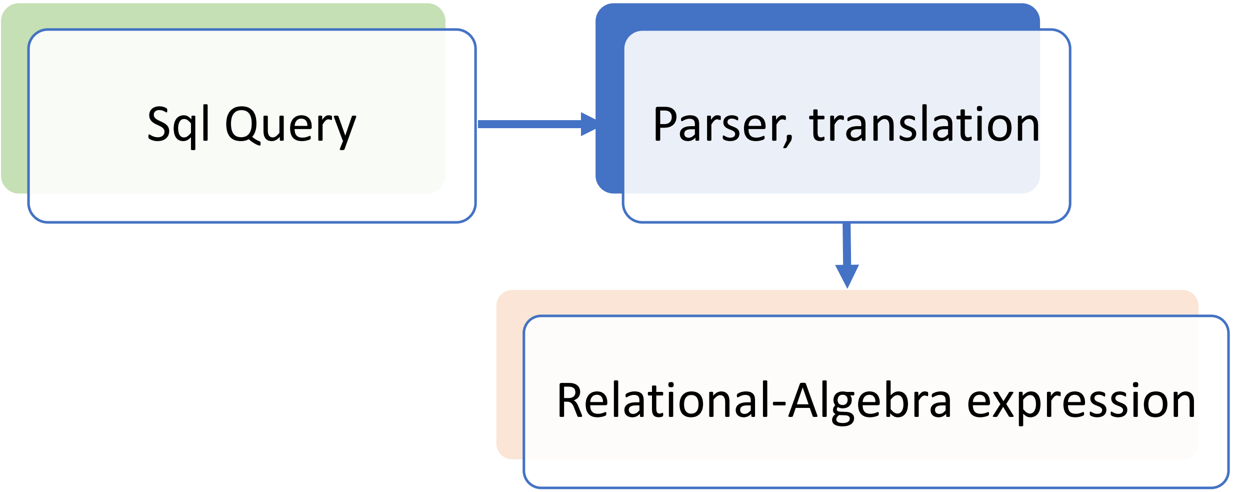
Query execution


Sql Query
Parser, translation
Relational-Algebra expression
Optimizer
EXECUTION-PLAN
Evaluation, algorithms
Query Output

Sql Query → Parser, translation

```
select p1.prod_name, p2.prod_name, p1.prod_min_price
from products p1 join products p2
        on p1.prod_min_price = p2.prod_min_price
```

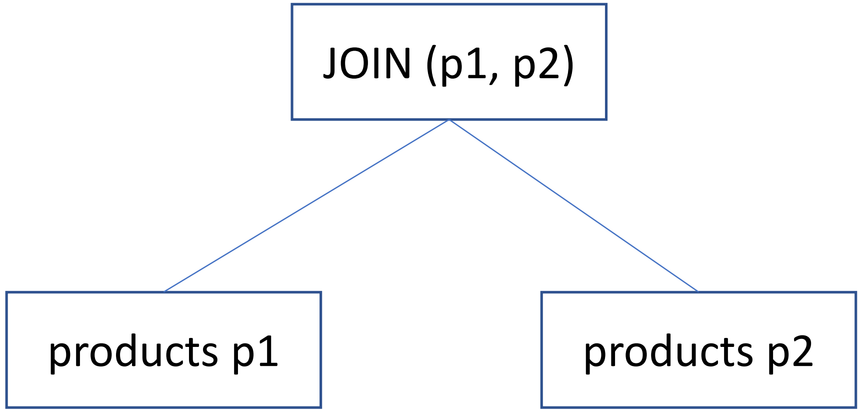
check syntax, table names, column names

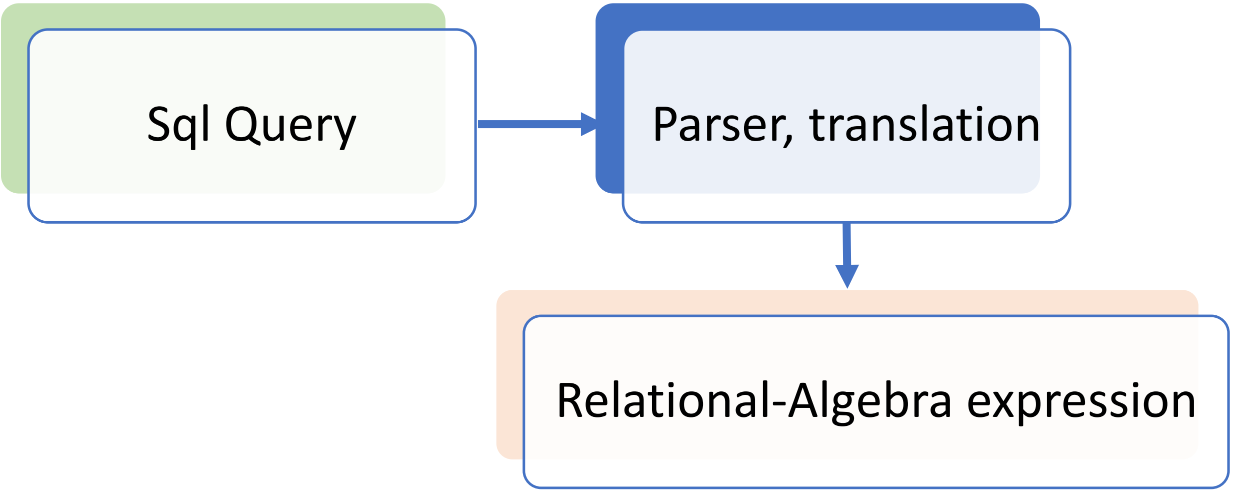


select p1.prod_name, p2.prod_name, p1.prod_min_price
from products p1 join products p2
on p1.prod_min_price = p2.prod_min_price

relations + operators

$$JOIN(\,p1, p2)$$

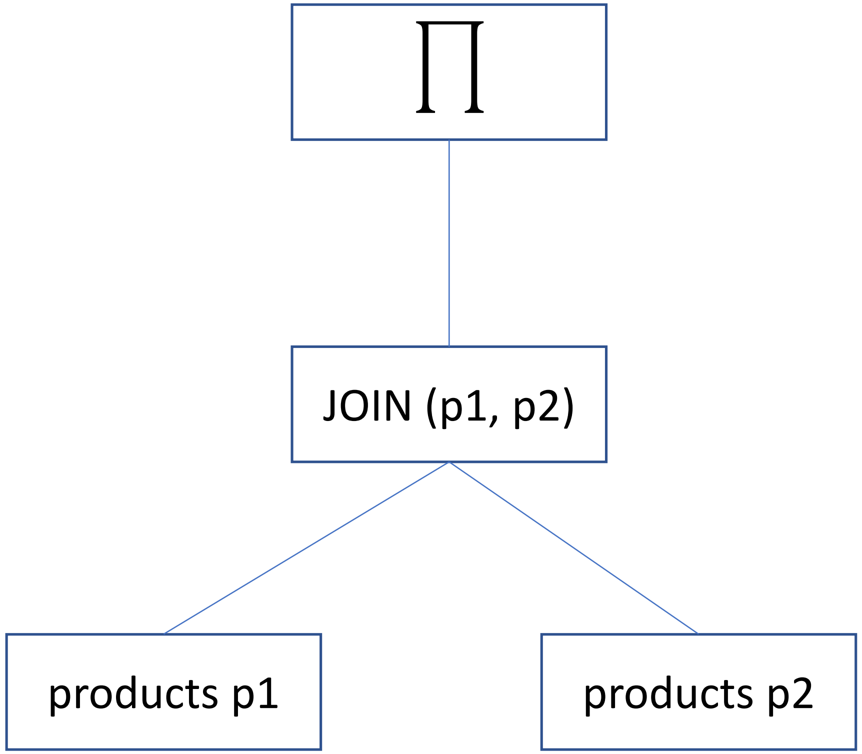


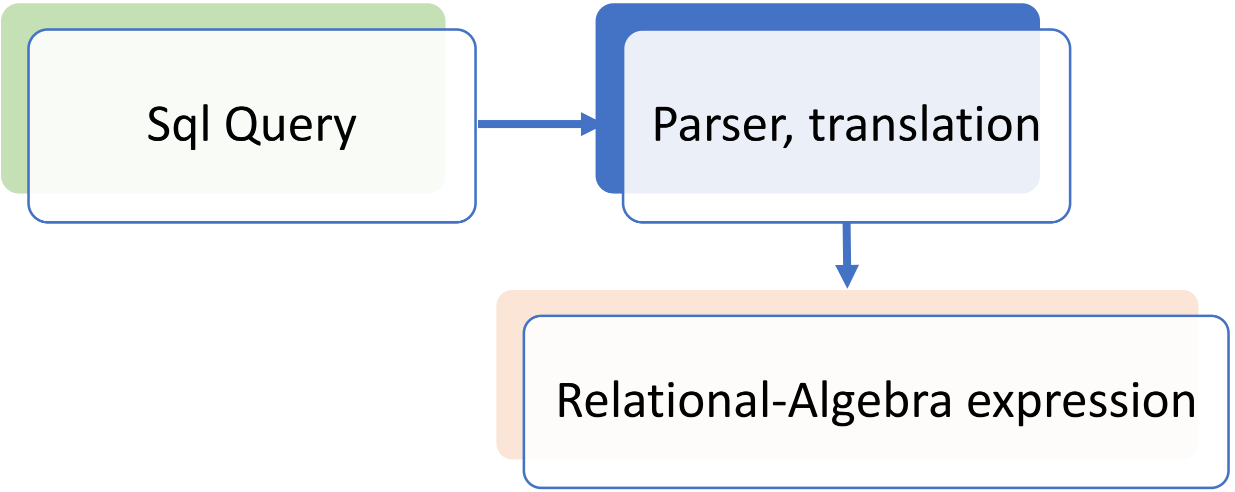


select p1.prod_name, p2.prod_name, p1.prod_min_price
from products p1 join products p2
on p1.prod_min_price = p2.prod_min_price

relations + operators

$$\prod_{p1.name,p1.minprice,p2.name,p2.price} JOIN(p1,p2)$$

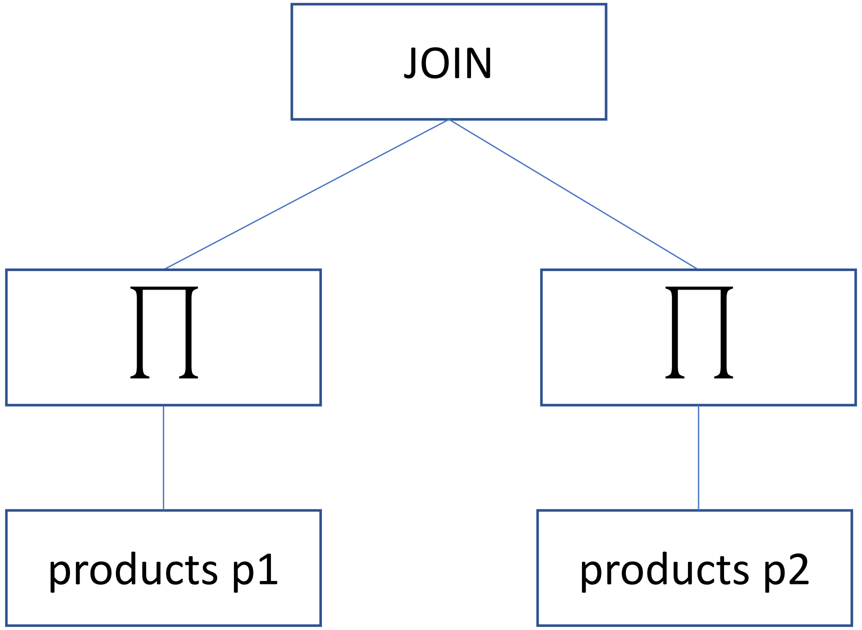


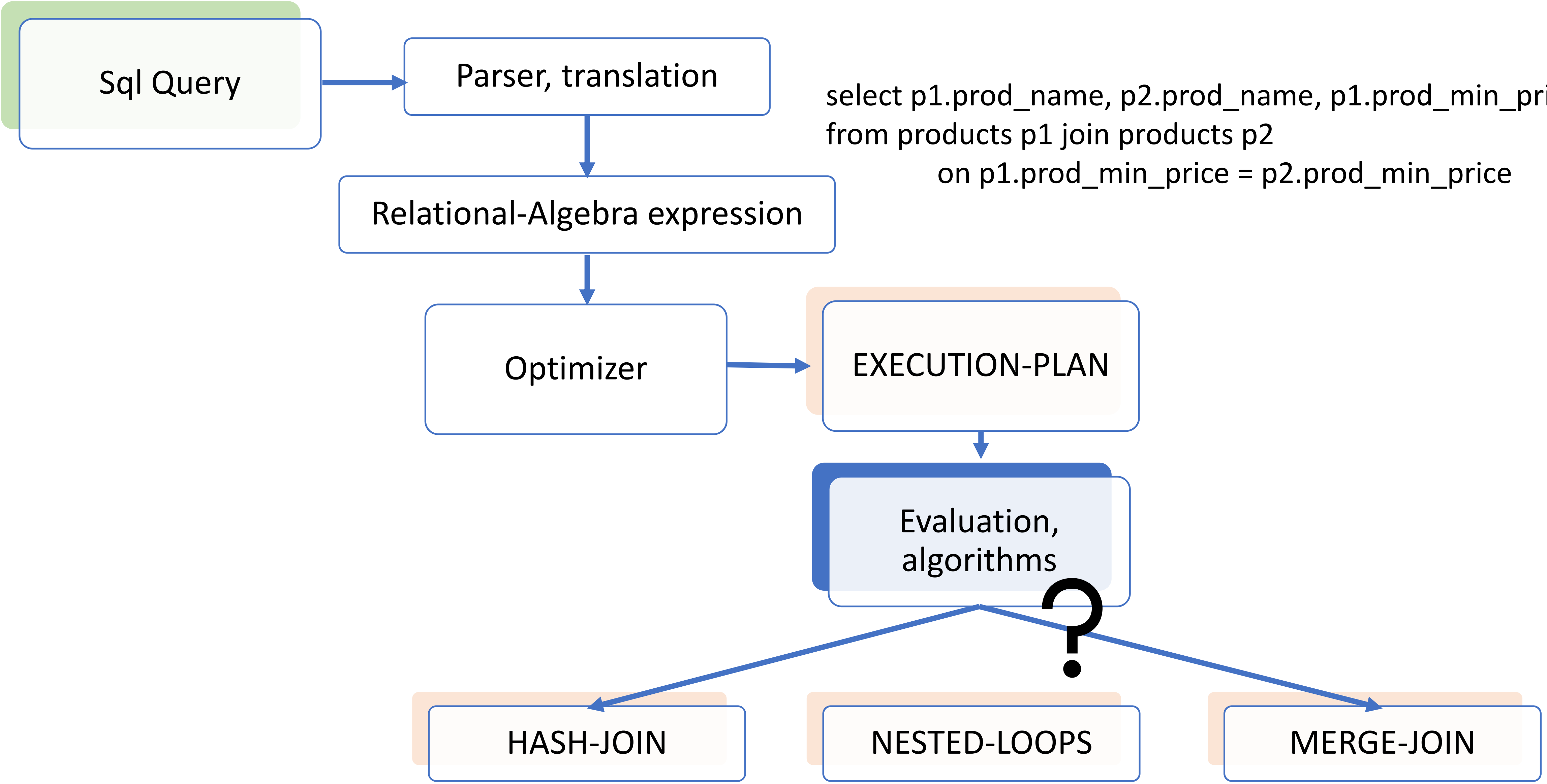


select p1.prod_name, p2.prod_name, p1.prod_min_price
from products p1 join products p2
on p1.prod_min_price = p2.prod_min_price

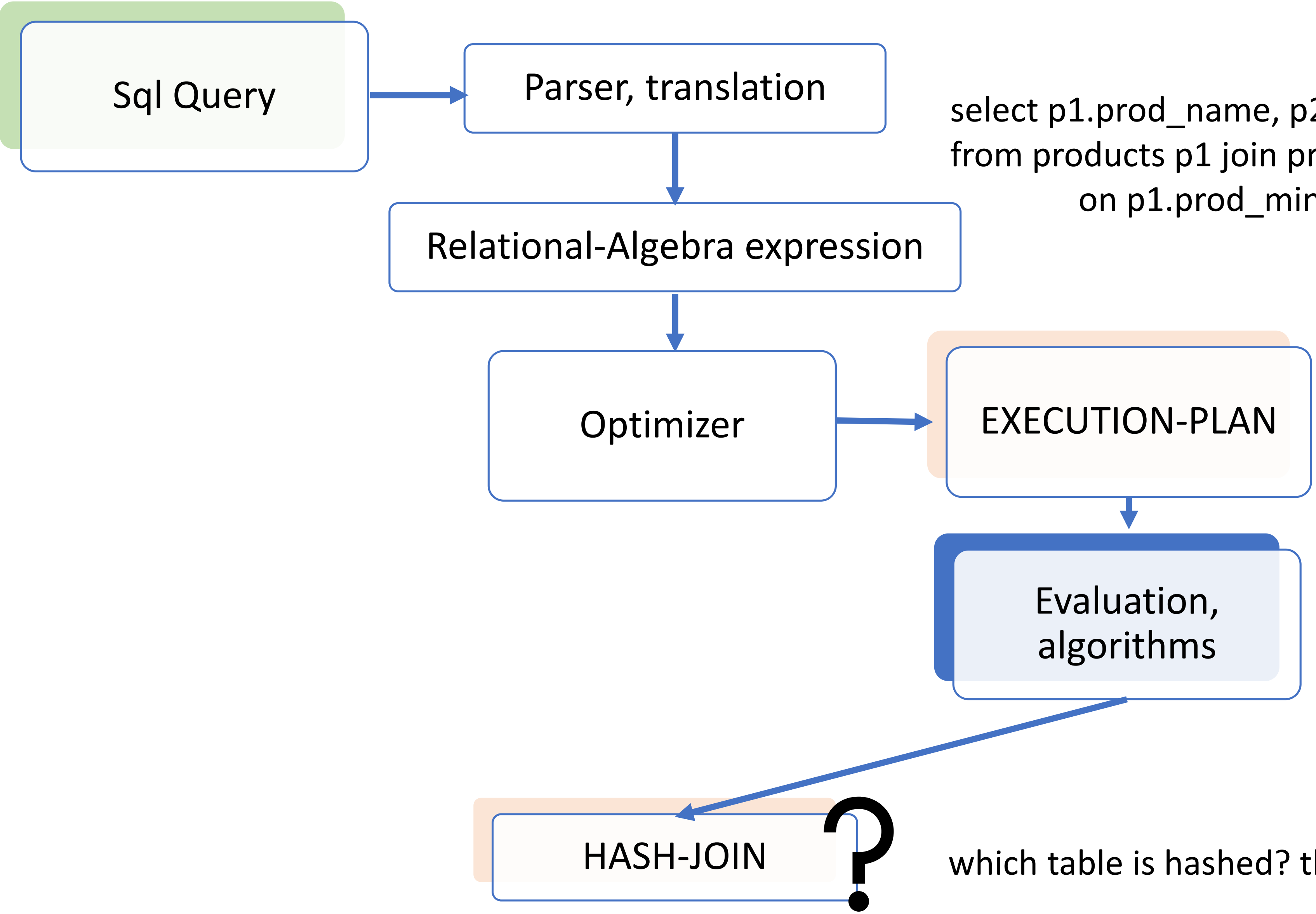
relations + operators

$$JOIN(\prod_{name,minprice} p1, \prod_{name,minprice} p2)$$






```
select p1.prod_name, p2.prod_name, p1.prod_min_price
from products p1 join products p2
        on p1.prod_min_price = p2.prod_min_price
```



select p1.prod_name, p2.prod_name, p1.prod_min_price
from products p1 join products p2
on p1.prod_min_price = p2.prod_min_price

which table is hashed? the “smaller”

Relational algebra properties

Relational algebra properties

- PROP1: join and cross product commute

$$\text{JOIN}(R1, R2) = \text{JOIN}(R2, R1)$$

$$R1 \times R2 = R2 \times R1$$

- PROP2: associativity

$$\text{JOIN}(\text{JOIN}(R1, R2), R3) = \text{JOIN}(R1, \text{JOIN}(R2, R3))$$

$$(R1 \times R2) \times R3 = R1 \times (R2 \times R3)$$

Relational algebra properties

- PROP3: projection composition

$$\Pi_{A1,\dots,Am}(\Pi_{B1,\dots,Bn}(R)) = \Pi_{A1,\dots,Am}(R),$$

$$\{A_1, A_2,\dots,A_m\} \subseteq \{B_1, B_2,\dots,B_n\}.$$

- PROP4: selection composition

$$\sigma_{cond1}(\sigma_{cond2}(R)) = \sigma_{cond1 \wedge cond2}(R) = \sigma_{cond2}(\sigma_{cond1}(R)),$$

Relational algebra properties

- PROP5: selection and projection commute

$$\Pi_{A1,\ldots,Am} (\sigma_{cond} (R)) = \sigma_{cond} (\Pi_{A1,\ldots,Am} (R))$$

$$\Pi_{A1,\ldots,Am} (\sigma_{cond} (R)) = \Pi_{A1,\ldots,Am} (\sigma_{cond} (\Pi_{A1,\ldots,Am,B1,\ldots,Bn} (R)))$$

- PROP6: selection and cross join commute

$$\sigma_{cond} (R1 \times R2) = \sigma_{cond} (R1) \times R2$$

$$\sigma_{cond} (R1 \times R2) = \sigma_{cond1} (R1) \times \sigma_{cond2} (R2)$$

$$\sigma_{cond} (R1 \times R2) = \sigma_{cond2} (\sigma_{cond1} (R1) \times R2)$$

Relational algebra properties

- PROP7: selection and union commute

$$\sigma_{cond}(R1 \cup R2) = \sigma_{cond}(R1) \cup \sigma_{cond}(R2)$$

- PROP8: selection and difference commute

$$\sigma_{cond}(R1 - R2) = \sigma_{cond}(R1) - \sigma_{cond}(R2)$$

Relational algebra properties

- PROP9: projection and cross product commute

$$\Pi_{A1,\ldots,Am}(R1 \times R2) = \Pi_{B1,\ldots,Bn}(R1) \times \Pi_{C1,\ldots,Ck}(R2)$$

- PROP10: projection and union commute

$$\Pi_{A1,\ldots,Am}(R1 \cup R2) = \Pi_{A1,\ldots,Am}(R1) \cup \Pi_{A1,\ldots,Am}(R2)$$

Relational algebra properties

- PROP11: join and projection commute

$$\Pi_{A1,\ldots,Am}\ (\text{JOIN}(R1,R2,D)) = \Pi_{A1,\ldots,Am}\ (\text{JOIN}(\Pi_{D,B1,\ldots,Bn}(R1),\ \Pi_{D,C1,\ldots,Ck}(R2),D)$$

- PROP12: selection and join composition

$$\sigma_{\text{cond}}\ (\text{JOIN}\ (R1,\ R2,\ D)) = \sigma_{\text{cond}}\ (\text{JOIN}\ (\Pi_{D,A}\ (R1),\ \Pi_{D,A}\ (R2),\ D))$$

General optimization rules

General optimization rules

- Execute selections first
  - Reduce relation size (number of rows)
- Avoid cross-joins, use joins
- First join to be executed is the one obtaining the smaller relation
- Execute projections first

Mesure Query Cost

rule-based execution plans

obsolite

cost-based execution plans

IO-cost

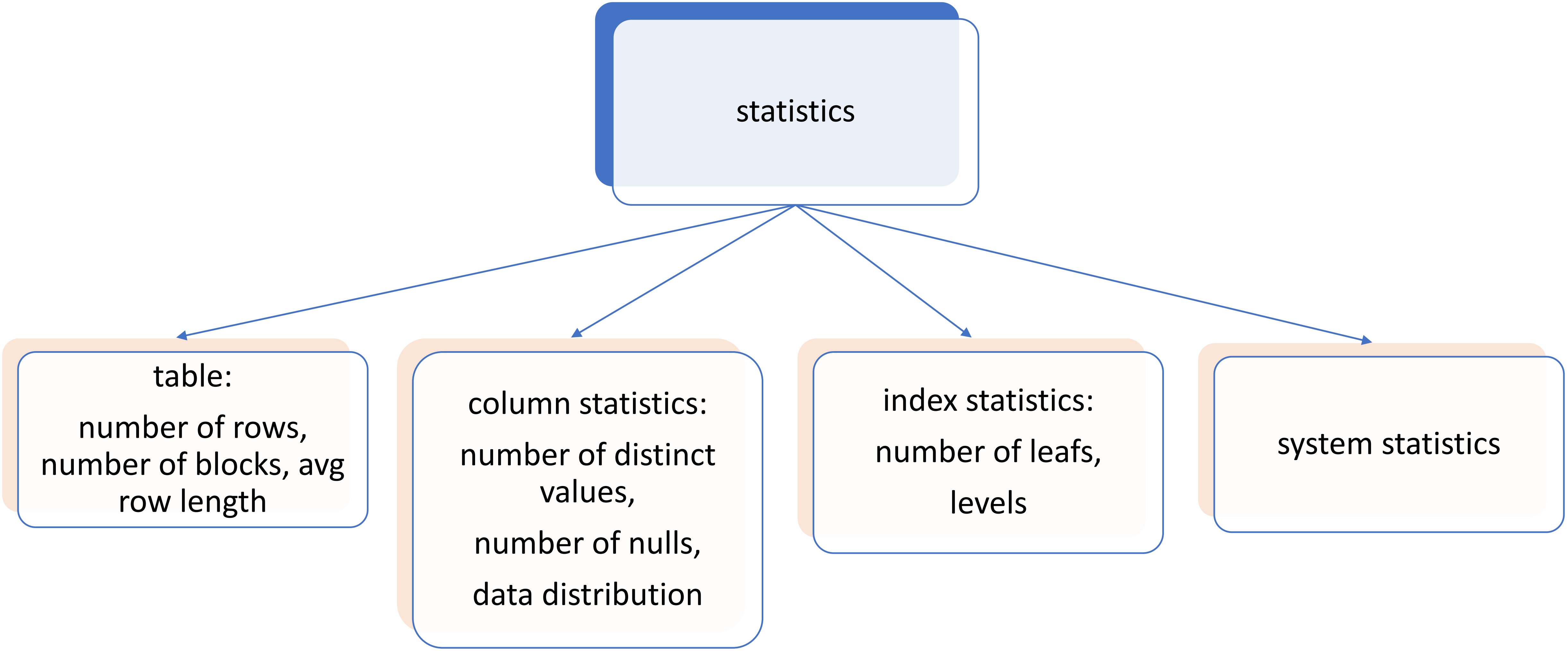
CPU-cost

disk accesses

- number of blocks transferred
- number of tuples

CPU time

- cost for processing a tuple
- cost for processing an index entry
- cost for processing a tuple
- cost for processing a function .....


statistics
table:
number of rows, number of blocks, avg row length
column statistics:
number of distinct values,
number of nulls,
data distribution
index statistics:
number of leafs,
levels
system statistics

BigData

Relational database vs BigData

- Structured data vs semi-structured data, graph data
- Data from a single enterprise
- BigData requires high degree of parallelism (storage and processing)
- Sharding, key-value storage systems and documents stores

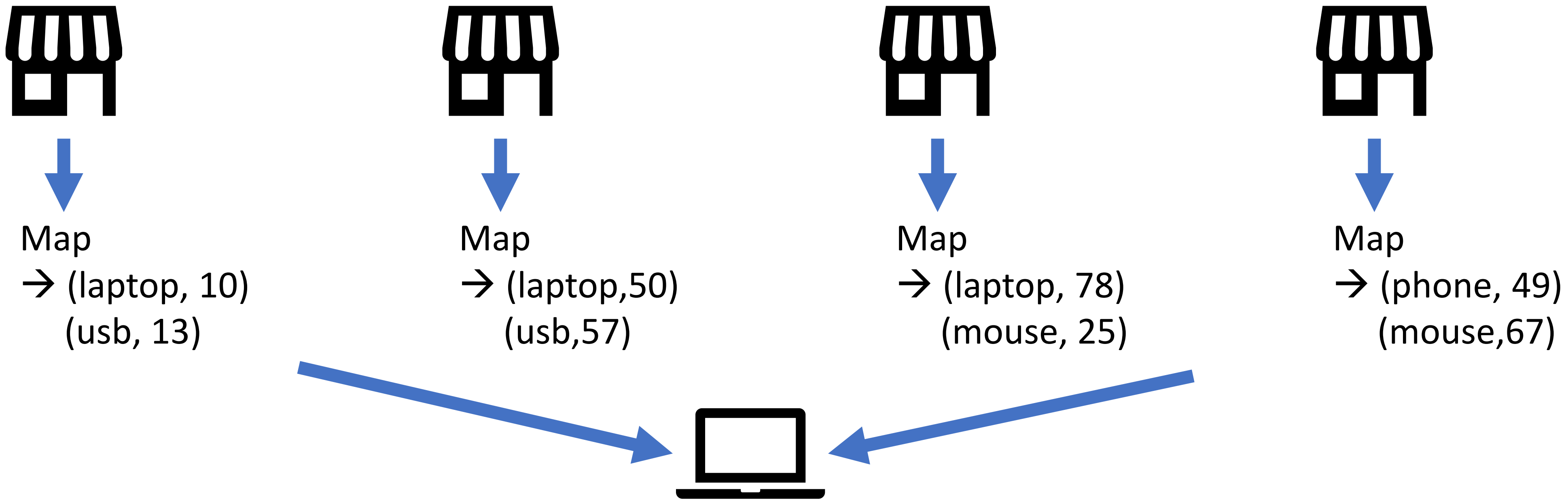
Map-reduce

Map Reduce algorithms

- Used in parallel processing.
- Fault tolerant.
- Programming paradigm (model) → framework,
  - examples Hadoop, Google
- Allows to process large volumes of data.
- Input in different formats.

Map Reduce example

- Counting product that clients entering local buy.
  - Input collected by multiple machines in parallel.
  - Data processed by multiple machines.



Map Reduce example

- MAP phase
  - map function provided by the developer will run on multiple nodes in parallel, process input data.

map
→ (key, value)
reduce key

....

....

....

Map Reduce example

- REDUCE phase
  - reduce function provided by the developer, reduce the output produced by map functions, aggregate.
  - a call for a reduce function is for a single reduced key.

(laptop, 10)
(usb, 13)

(laptop,50)
(usb,57)

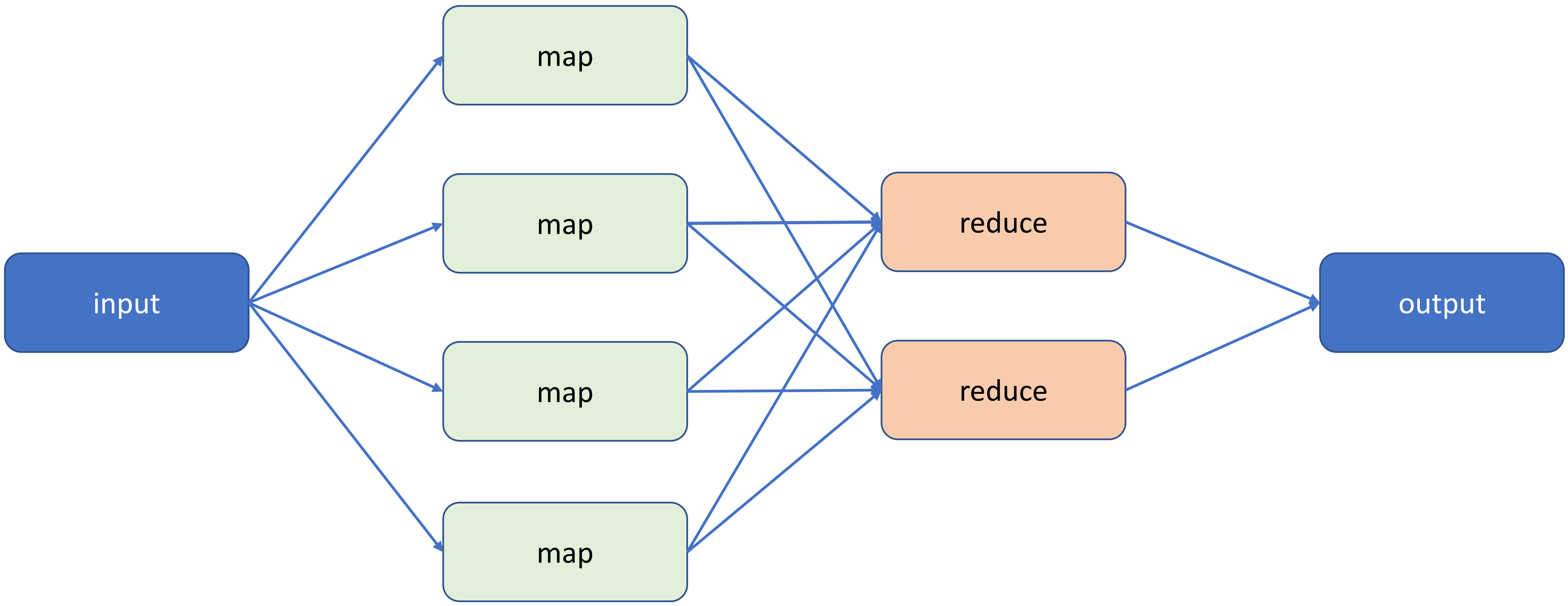
(laptop, 78)
(mouse, 25)

(phone, 49)
(mouse,67)

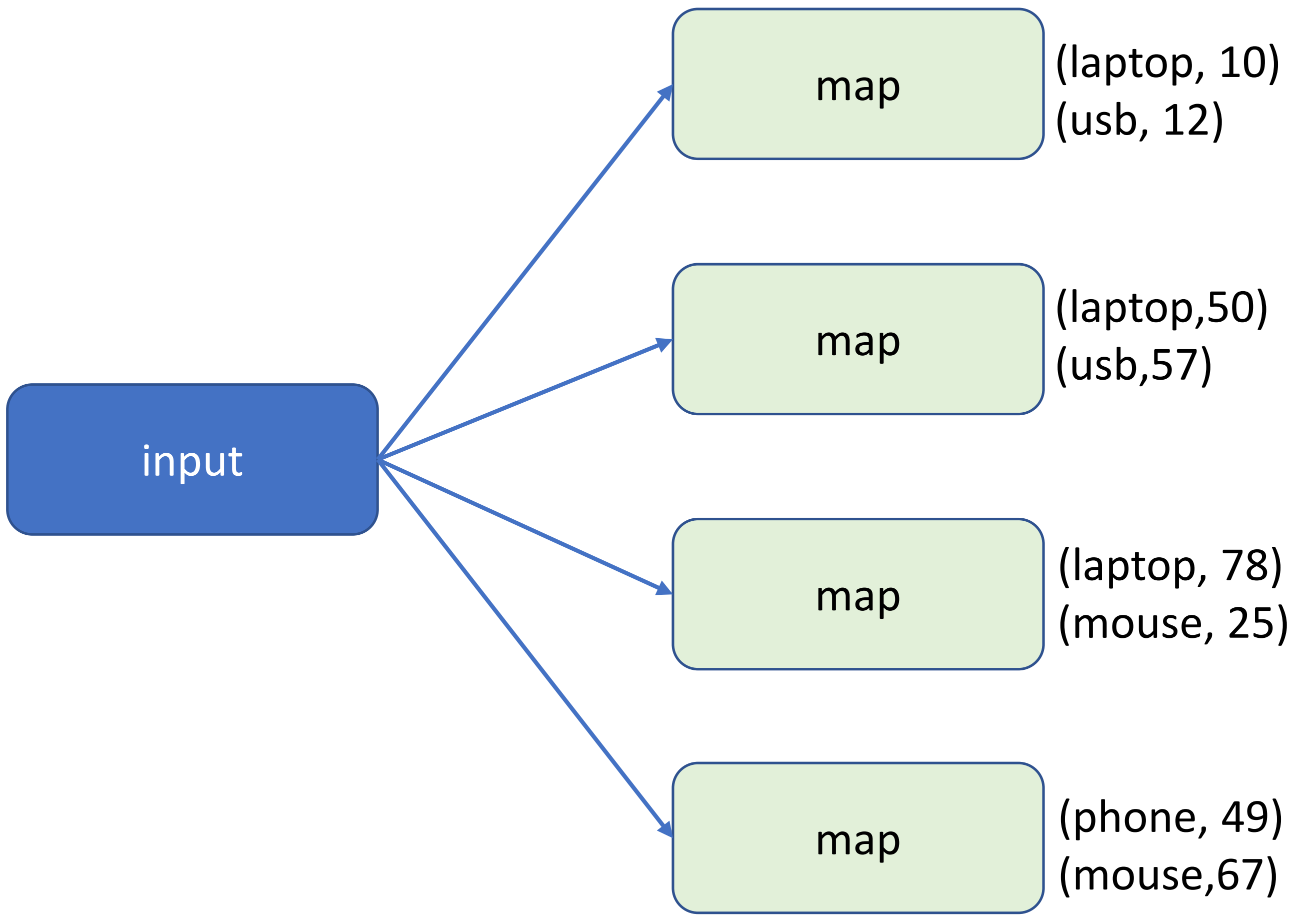
Shuffle,
Sort,
Reduce

output

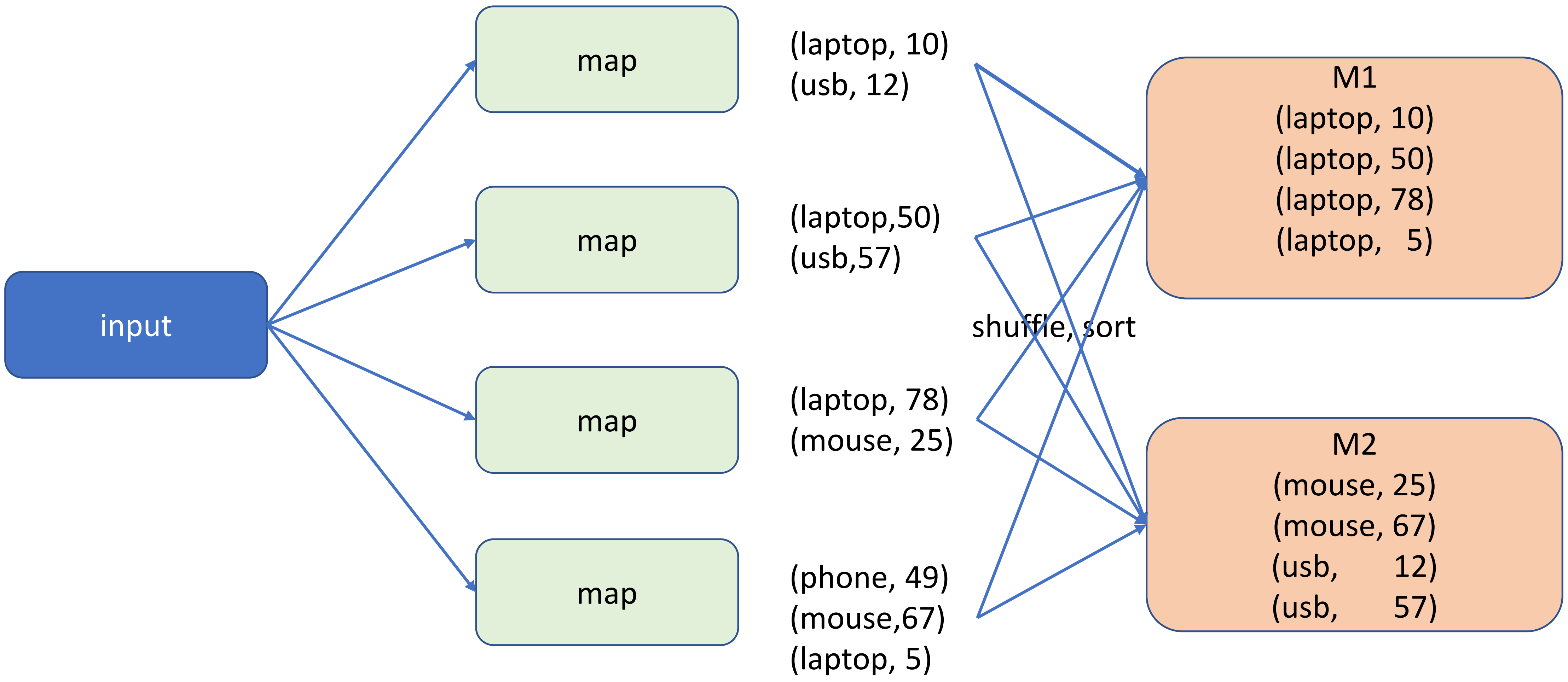
Map reduce



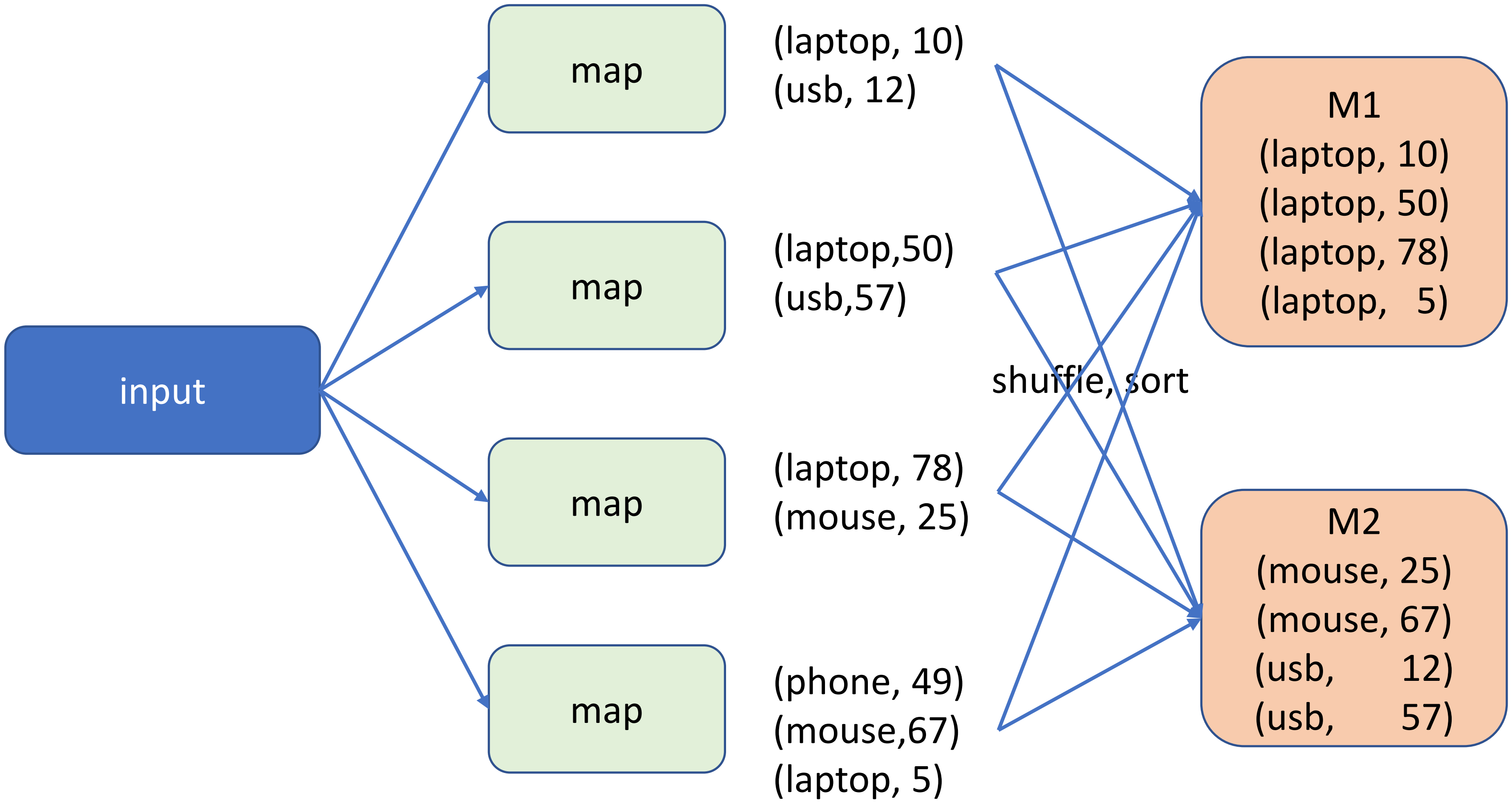
Map reduce


input
map
(laptop, 10)
(usb, 12)
map
(laptop,50)
(usb,57)
map
(laptop, 78)
(mouse, 25)
map
(phone, 49)
(mouse,67)

Map reduce

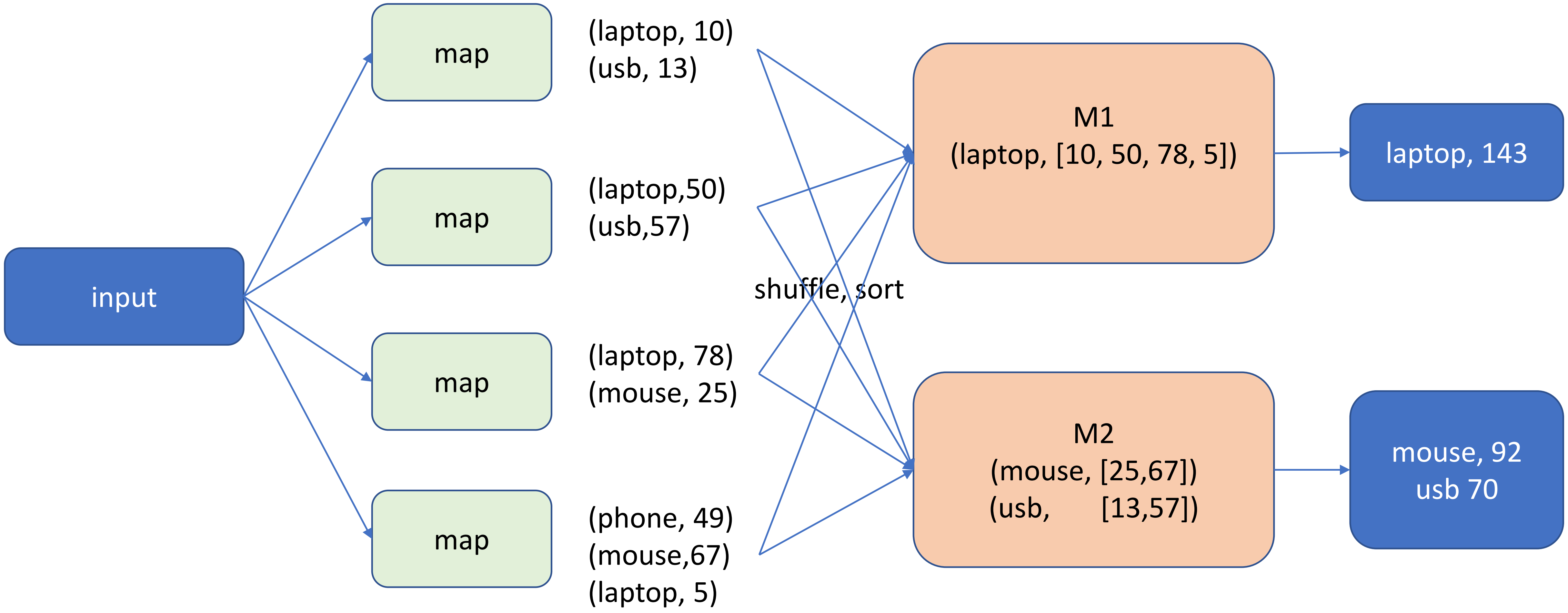

input
map
map
map
map
(laptop, 10)
(usb, 12)
(laptop,50)
(usb,57)
(laptop, 78)
(mouse, 25)
(phone, 49)
(mouse,67)
(laptop, 5)
shuffle, sort
M1
(laptop, 10)
(laptop, 50)
(laptop, 78)
(laptop, 5)
M2
(mouse, 25)
(mouse, 67)
(usb, 12)
(usb, 57)

Map reduce



input
map
map
map
map
(laptop, 10)
(usb, 12)
(laptop,50)
(usb,57)
(laptop, 78)
(mouse, 25)
(phone, 49)
(mouse,67)
(laptop, 5)
shuffle, sort
M1
(laptop, 10)
(laptop, 50)
(laptop, 78)
(laptop, 5)
M2
(mouse, 25)
(mouse, 67)
(usb, 12)
(usb, 57)

Map reduce


input
map
map
map
map
(laptop, 10)
(usb, 13)
(laptop,50)
(usb,57)
(laptop, 78)
(mouse, 25)
(phone, 49)
(mouse,67)
(laptop, 5)
shuffle, sort
M1
(laptop, [10, 50, 78, 5])
M2
(mouse, [25,67])
(usb, [13,57])
laptop, 143
mouse, 92
usb 70

MapReduce Hadoop

- Open source from Apache. https://hadoop.apache.org/

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

- Written in Java, also provide implementations in C++/Python.

- Components
  - MapReduce

  - Hadoop Distributed file system HDFS
    - Each file is stored as a sequence of blocks
    - Fault tolerant: Each block is replicated

- Master-slave architecture: NameNode (master), DataNodes (slaves).

MapReduce Hadoop

- map, reduce and combine function.

- combine perform partial aggregation before maps sends the result to reduce.

- combine -- reduce the amount of data sent over the network.

- combine -- Decrease the shuffling cost

- A MapReduce job can be configured to process map function phase only

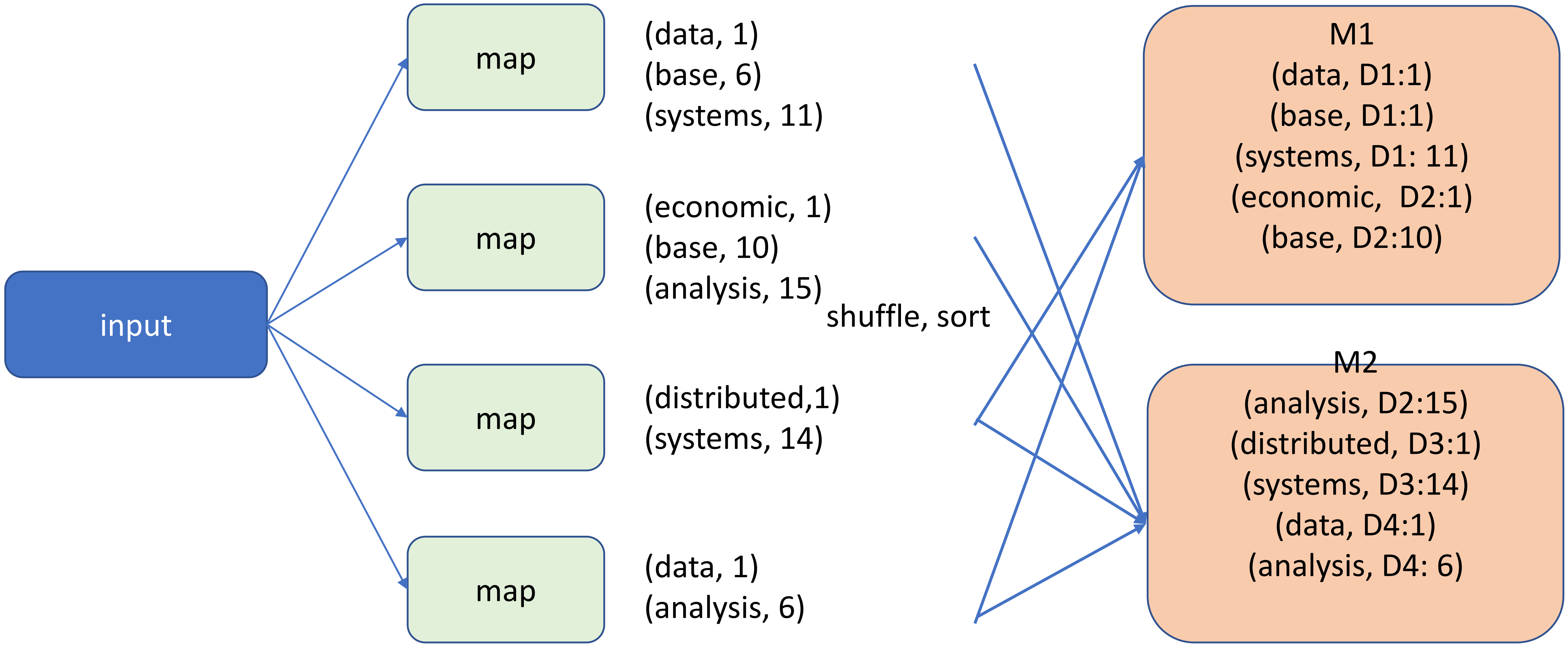
Inverted index

MapReduce Inverted index

- Web search engines (including Google).
- Maps content to location.
- Fast text search.
- PageRank-ing

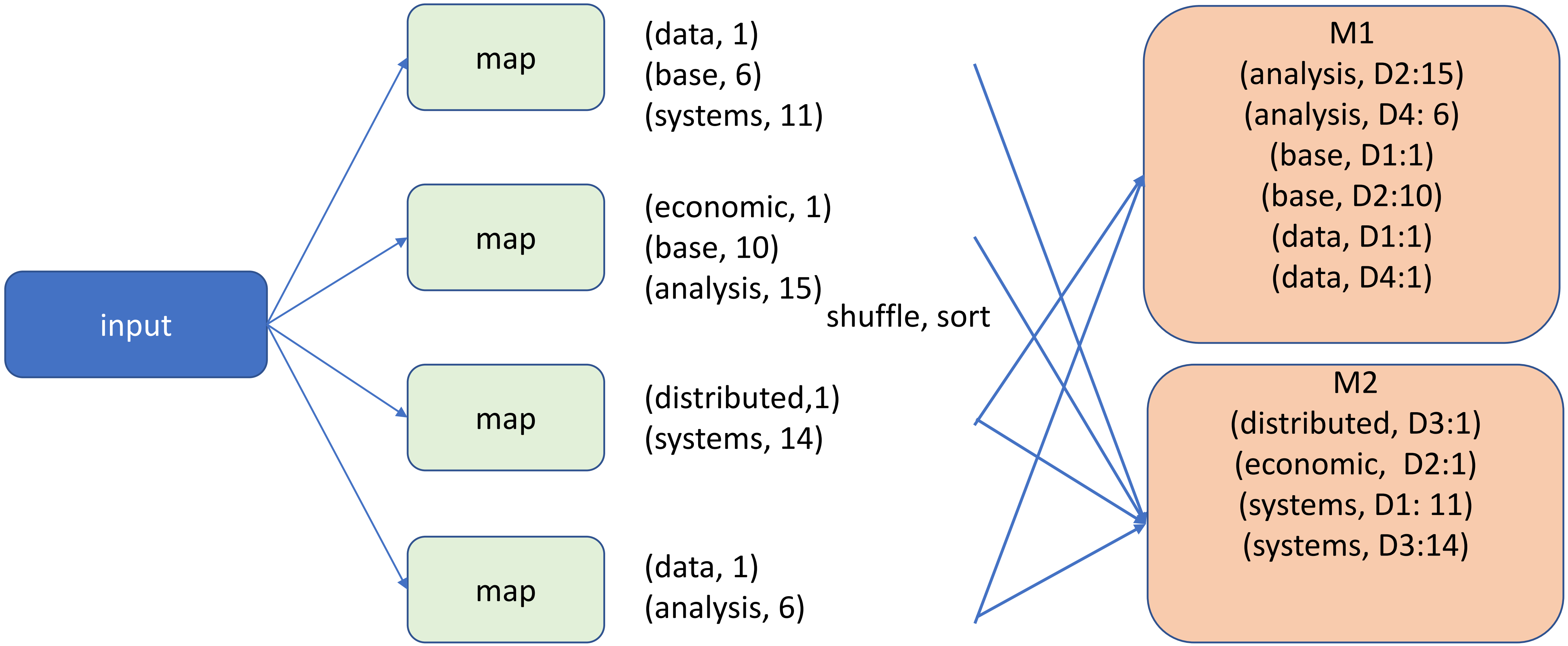
Inverted index

D1: data base systems,
D2: economic base analysis
D3: distributed systems
D4: data analysis



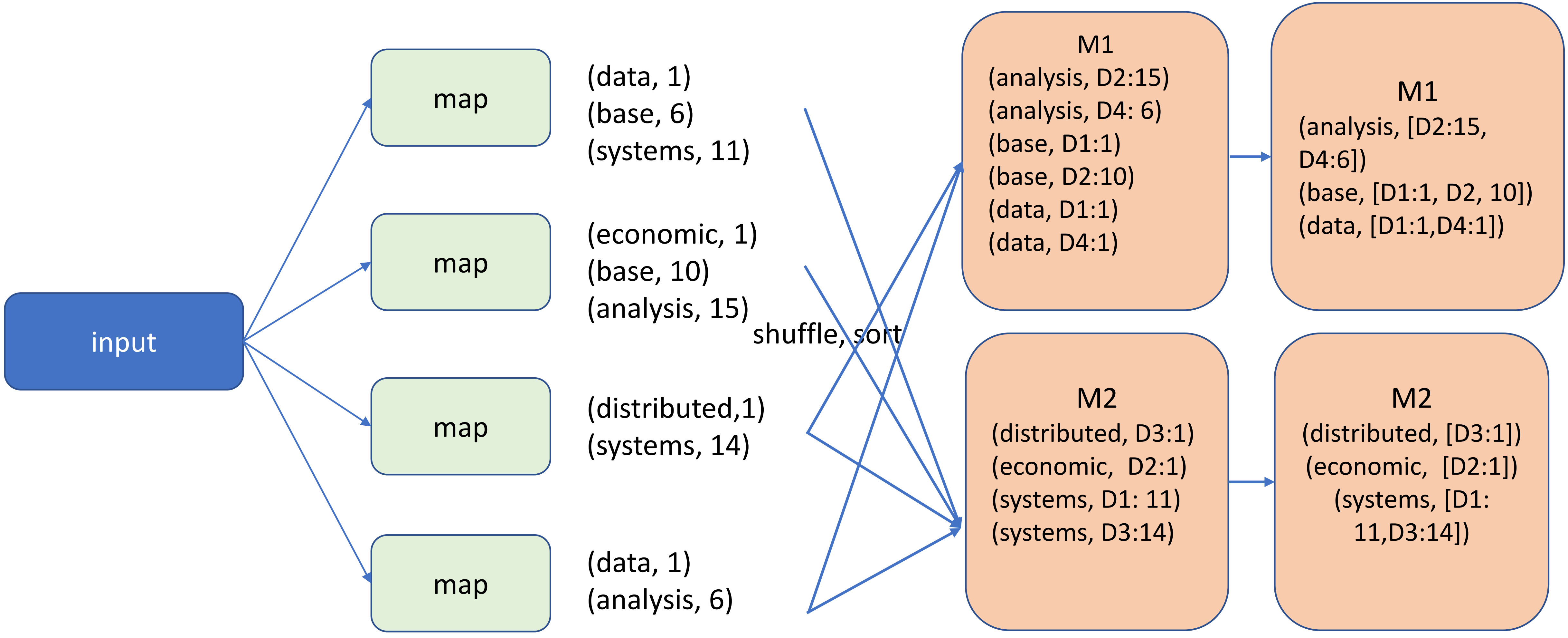
Inverted index

D1: data base systems,
D2: economic base analysis
D3: distributed systems
D4: data analysis



Inverted index

D1: data base systems,
D2: economic base analysis
D3: distributed systems
D4: data analysis



Sql operators

MapReduce: Sql operators

- Selection
- Group by
- Join

EMPLOYEES		
emp_id	name	dep_id
100	Steven King	90
102	Lex De Hann	90
108	Nancy Greenberg	100
116	Shelli Baida	30
117	Sigal Tobias	30

map

key	Value
90	(Emp, Steven King, 90)
90	(Emp, 102, Lex De Hann, 90)
100	(Emp, 108, Nancy Greenberg, 90)
30	(Emp, 116, Shelli Baida, 30)
30	(Emp, 117, Sigal Tobias, 30

DEPARTMENTS	
dep_id	dep_name
30	Purchasing
90	Executive
100	Finance
20	Marketing

map

key	Value
30	(Dep, 30, Purchasing)
90	(Dep, 90, Executive)
100	(Dep, 100, Finance)
20	(Dep, 20, Marketing)

EMPLOYEES		
emp_id	name	dep_id
100	Steven King	90
102	Lex De Hann	90
108	Nancy Greenberg	100
116	Shelli Baida	30
117	Sigal Tobias	30

map

key	Value
90	(Emp, Steven King, 90)
90	(Emp, 102, Lex De Hann, 90)
100	(Emp, 108, Nancy Greenberg, 90)
30	(Emp, 116, Shelli Baida, 30)
30	(Emp, 117, Sigal Tobias, 30

DEPARTMENTS	
dep_id	dep_name
30	Purchasing
90	Executive
100	Finance
20	Marketing

map

key	Value
30	(Dep, 30, Purchasing)
90	(Dep, 90, Executive)
100	(Dep, 100, Finance)
20	(Dep, 20, Marketing)

shuffle

key	Value
20	(Dep, 20, Marketing)
30	(Dep, 30, Purchasing)
30	(Emp, 116, Shelli Baida, 30)
30	(Emp, 117, Sigal Tobias, 30
90	(Dep, 90, Executive)
90	(Emp, Steven King, 90)
90	(Emp, 102, Lex De Hann, 90)
100	(Dep, 100, Finance)
100	(Emp, 108, Nancy Greenberg, 90)

EMPLOYEES		
emp_id	name	dep_id
100	Steven King	90
102	Lex De Hann	90
108	Nancy Greenberg	100
116	Shelli Baida	30
117	Sigal Tobias	30

DEPARTMENTS	
dep_id	dep_name
30	Purchasing
90	Executive
100	Finance
20	Marketing

map

key	Value
90	(Emp, Steven King, 90)
90	(Emp, 102, Lex De Hann, 90)
100	(Emp, 108, Nancy Greenberg, 90)
30	(Emp, 116, Shelli Baida, 30)
30	(Emp, 117, Sigal Tobias, 30

map

key	Value
30	(Dep, 30, Purchasing)
90	(Dep, 90, Executive)
100	(Dep, 100, Finance)
20	(Dep, 20, Marketing)

shuffle

key	Value
~~20~~	~~(Dep, 20, Marketing)~~
30	(Dep, 30, Purchasing)
30	(Emp, 116, Shelli Baida, 30)
30	(Emp, 117, Sigal Tobias, 30
90	(Dep, 90, Executive)
90	(Emp, Steven King, 90)
90	(Emp, 102, Lex De Hann, 90)
100	(Dep, 100, Finance)
100	(Emp, 108, Nancy Greenberg, 90)

reduce

key	Value
30	[(Dep, 30, Purchasing), (Emp, 116, Shelli Baida, 30), (Emp, 117, Sigal Tobias, 30]
90	[(Dep, 90, Executive), (Emp, Steven King, 90), (Emp, 102, Lex De Hann, 90)]
100	[(Dep, 100, Finance), (Emp, 108, Nancy Greenberg, 90)]

NoSql

NoSql

- Flexible schema
  - Does not use a structured query language.
    - In RDBMs normalized models.
  - Easy to migrate.
  - Suitable for semi-structured, complex, nested data.
- Typically do not support transactions.
  - Relax some ACID properties to ensure scalability.
- High performance.
- Open Source/specific API.

NoSql Key-value databases

- Key-value databases
  - Store/update/retrieve record with an associate key.

    →Put(key, value)
    →Get(Key)

- Examples
  - Bigtable, Apache HBase, Dynamo, Cassandra, MongoDB, Azure etc.

- Document stores (MongoDB)
  - data follow a specific data representation, example JSON format.
  - Execute simple queries based on stored values.

Partitioning/sharding

- Key-value databases
  - Records are partitioned among a cluster,
    each nodes performs lookups and updates on a subset of records.

- Challenges: manage request that must access data from multiple shards

  → replicas in order to ensure availability in case of failure,

  → keep replicas consistent,

  → expensive joins if tables are stores on different nodes,
  depends on the speed of the communication network.

Sharding

- Types of partitioning: horizontal partitioning (example sharding),
  vertical partitioning

- Partition is done on attributes refereed as
  partitioning key or shard keys
  → range partitioning divide data into ranges based on the key value
  → hash partitioning even data distribution but range-queries target more shards

Sharding in MongoDb

- Chunk: lower and upper range based on the shared key.

- Architecture:
  - Mongos: query routers
  - Config Servers
  - Shards (replicas)

- If queries do not include the shard, mongos performs a broadcast operation.

CAP theorem

- No distributed database can guarantee more than two of the following:

- Consistency: read the most recent write or an error,
  (linearizable consistency) once an operation is complete,
  it is visible to all nodes.
  eventual consistency
- Availability: every request receives a non-error response
- Partition tolerance: system operates despite arbitrary number of messages being lost

- No distributed database can guarantee more than two of the following:

- Consistency: read the most recent write or an error,
- Availability: every request receives a non-error response

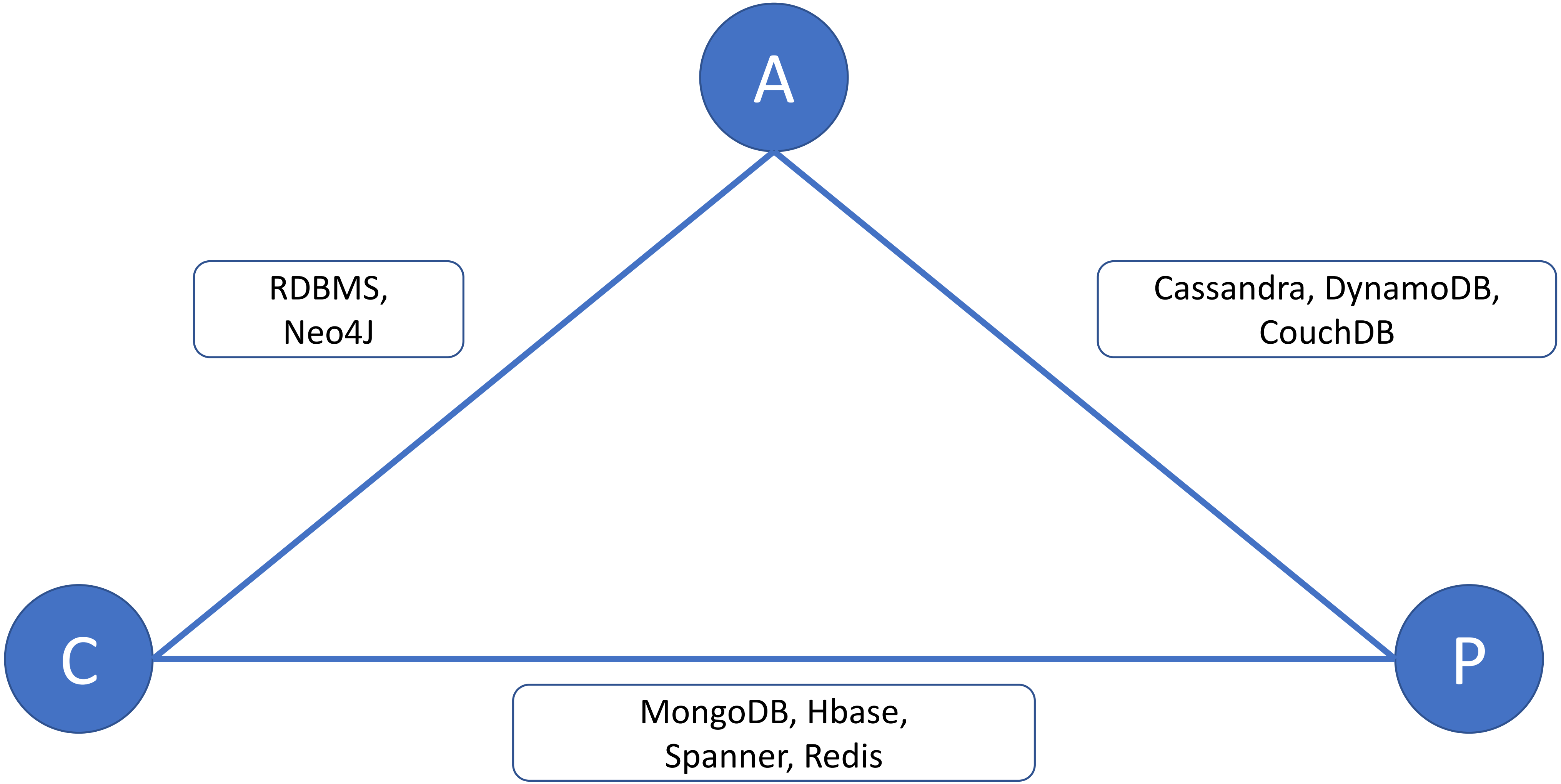
  non-failing nodes receiving requests returns a response

  high availability
- Partition tolerance: system operates despite arbitrary number of messages being lost

- No distributed database can guarantee more than two of the following:

- CP: sacrifice availability, consistency and partition tolerance
- AP: sacrifice consistency, availability and partition tolerance
- CA: sacrifice partition tolerance, consistency and availability

- Alternative: PACELC


A
C
P
RDBMS,
Neo4J
Cassandra, DynamoDB,
CouchDB
MongoDB, Hbase,
Spanner, Redis

CAP Theorem

- MongoDB **CP** datastore.

- Each replica set one primary nodes receives write operations.
- Secondary nodes replicate primary node’s operations.

- If case of failure of the primary node, a secondary node replace it (node with the most recent log).
- The cluster becomes available only when all the secondary nodes replicate the primary node.

CAP Theorem

- Cassandra **CP** datastore.

- **Eventually consistent**: it's not guarantee that all replicas have the same data.
- Consistency level: number of replicas that needs to respond to a read/write operation.
  - ONE: closest replica
  - QUORUM: synchronize → majority,

Consistency levels

- **Strict consistency:** global clock, all reads seen instantaneously by all processors.
- **Sequential consistency:** global order on write operations.
- **Atomic consistency or linearizability:** global order on operations that do not overlap in time.
- **Casual consistency:** global order on related write operations.
- **Eventually consistent**: if there are no writes for a period of time that is system dependent, every node will “see” the value of the last write.

BASE

BASE

- Basically Available: low latency, high availability
- Soft state: nodes are updated without any input.
- Eventually consistent

Mongo DB

Mongo DB and SQL

Mongo	RDBMS
Document: set of key-value pairs, similar to JSON objects	row in a table
Collection: set of documents, documents in a collection may have different sets of fields	table
Field in JSON document	column
$lookup and embedded documents	joins
…	
https://docs.mongodb.com/manual/reference/sql-comparison/	

Mongo API

Use/create/delete database	
show dbs	show available databases
use database_identifier	create database/switch to database
db.dropDatabase()	drop selected database
Use/create/delete collection	
db.createCollection(id_collection)	
show collections	
db.createCollection("cappedCollection", {capped:true, size: 10000, max:3})	fixed size collection, replace oldest record
db.cappedCollection.drop()	drop collection
https://docs.mongodb.com/manual/core/databases-and-collections/	

Mongo Keys and indexes

Mongo keys and indexes

- Mongo automatically creates a key for the inserted objects.
  - _id attribute
  - Index on _id is created by default, structure:
    - a 4-byte *timestamp value*
    - a 5-byte *random value*
    - a 3-byte *incrementing counter*, initialized to a random value

- Single field index
- Compound index
- Multi key index
- Geospatial index
- Text index
- Hashed index

Optimization techniques

Bloom filters

Bloom filters

- Probabilistic data structure, check membership for a value in a set.

- How it works: S, set of n values → *const * n* bits
  calculate hash(v) $\in [1, const * n]$
  set bit hash(v) to 1

  Test w ∈ S → h(w) = 1 ?

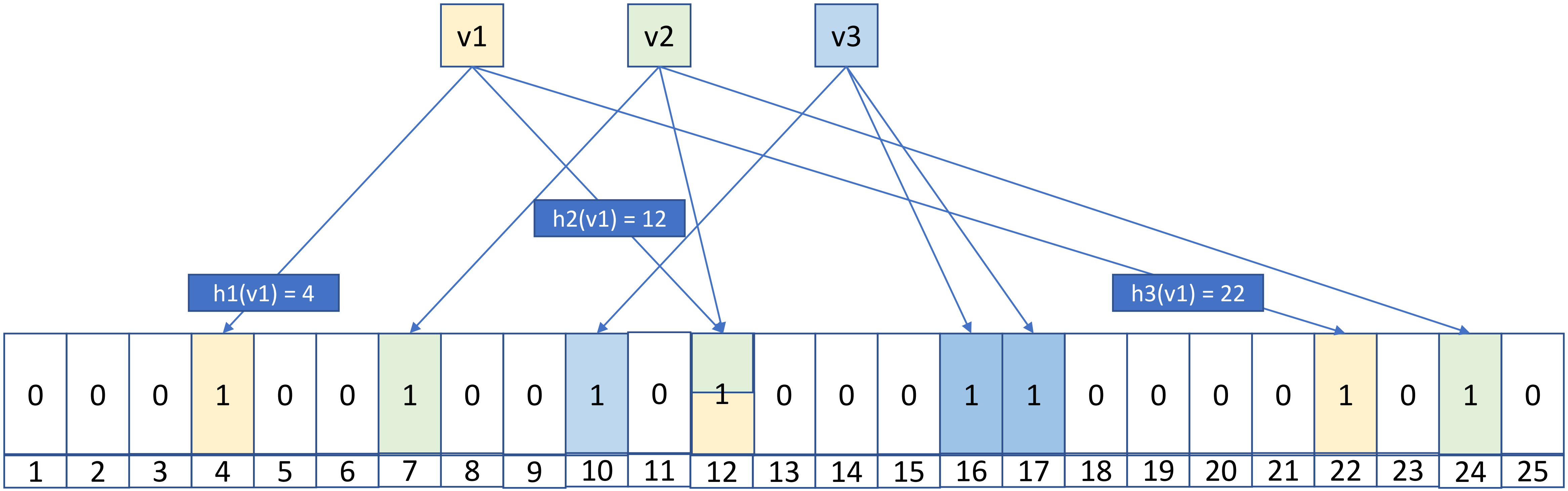
- Small probability of **false positive**. w1 ∈ S, w2 ∉ S h(w1) = h(w2)

Bloom filters

- To reduce the probability of false positives use k > 1 independent hash functions.

- How it works: S, set of n values → *const * n* bits
  calculate $h_1(v)$, $h_2(v)$ … $h_k(v) \in [1, const * n]$
  set bits $h_1(v)$, $h_2(v)$ … $h_k(v)$ to 1

  Test w ∈ S → $h_1(v) = 1$ and $h_2(v) = 1$ … and $h_k(v) = 1$?



Small probability of **false positive**.

Probability of **false negative** = 0.

Bloom filters

- Used only to add elements or the test membership.

- Once an element is added to the filter it cannot be removed.

- If all bits are set to 1, the probability of false positives increases.

  More space → more accuracy.
- More hash functions

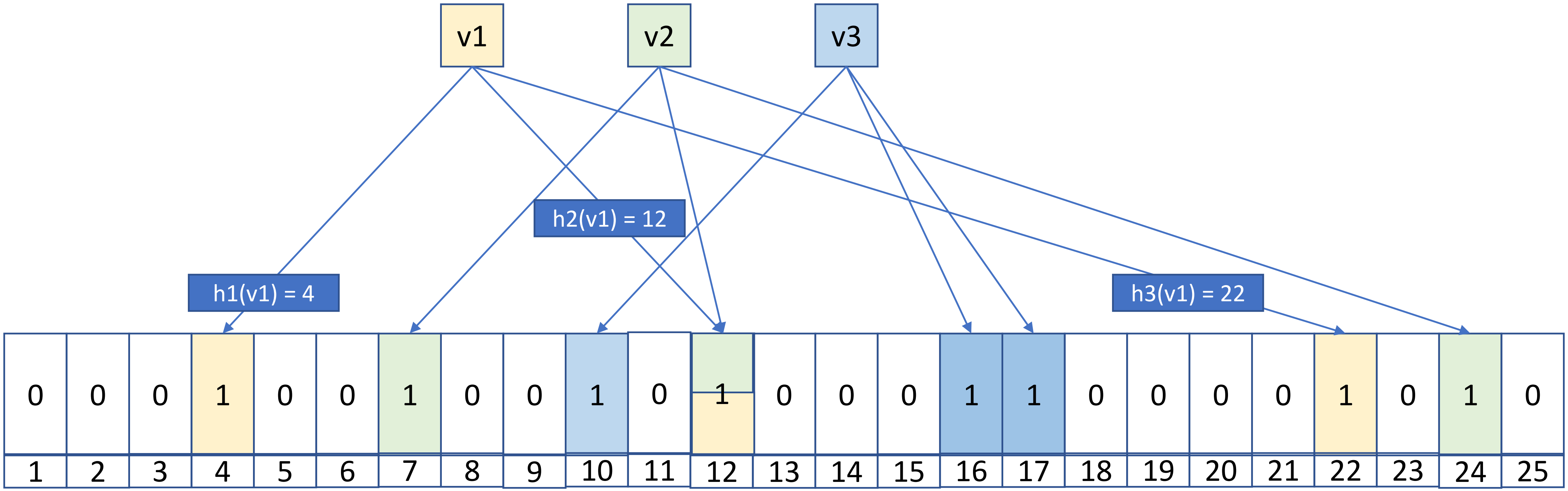
  Latency → more accuracy.

Bloom filters – independent hashing

- A family of hash functions $H = \{h: U \to [1..m]\}$ is k-independent if $\forall\, (x_1, x_2 \ldots x_k) \in U^k$ and $\forall\, (y_1, y_2 \ldots y_k) \in [1..m]^k$:

  - $Pr_{h \in H}\, [h(x_1) = y_1 \wedge h(x_2) = y_2 \ldots \wedge h(x_k) = y_k] = \frac{1}{m^k}$

- $h(x_1)$ uniformly distributed.
- $h(x_1), h(x_2)\, , \ldots h(x_k)$ independent random variables.



Small probability of **false positive**.

false positive. Value *w: B[h1(w)] = 1 B[h2(w)] =1 … B[hk[w]] = 1*

Probability of **false negative** = 0.

Each hash of w equals a hash of an element in the set

Bloom filters – accuracy

- m size of array, n number of elements in S, k number of hash functions.
- Probability of false positive:

$$P = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^{k} \quad \text{or}$$

$$P = \left(1 - e^{-\frac{kn}{m}}\right)^{k}$$

- m = 10 * n and k = 7 $\simeq$ 0,01

Bloom filters – accuracy

- m size of array, n number of elements in S, k number of hash functions.
- Probability of false positive:

h(w) != h1(v1)

$$P = \left(1 - \boxed{\left(1 - \frac{1}{m}\right)}^{kn}\right)^{k} \quad \text{or}$$

- m = 10 * n and k = 7 ≃ 0,01

Bloom filters – accuracy

- m size of array, n number of elements in S, k number of hash functions.
- Probability of false positive:

$$P = \left(1 - \boxed{\left(1 - \frac{1}{m}\right)^{kn}}\right)^{k} \quad \text{or}$$

h1(w) != h1(v1)
h1(w) != h1(v1)
.....
h1(w) != hn(v1)
h1(w) != h1(v2)
...
h1(w) != hn(v2)
...

- m = 10 * n and k = 7 ≃ 0,01

Bloom filters – accuracy

- m size of array, n number of elements in S, k number of hash functions.
- Probability of false positive:

$$P = \left(\boxed{1 - \left(1 - \frac{1}{m}\right)^{kn}}\right)^{k} \quad \text{or}$$

h1(w) = h1(v1)
or
h1(w) = h1(v1)
.....
h1(w) = hn(v1)
or
h1(w) = h1(v2)
...
h1(w) = hn(v2)
...

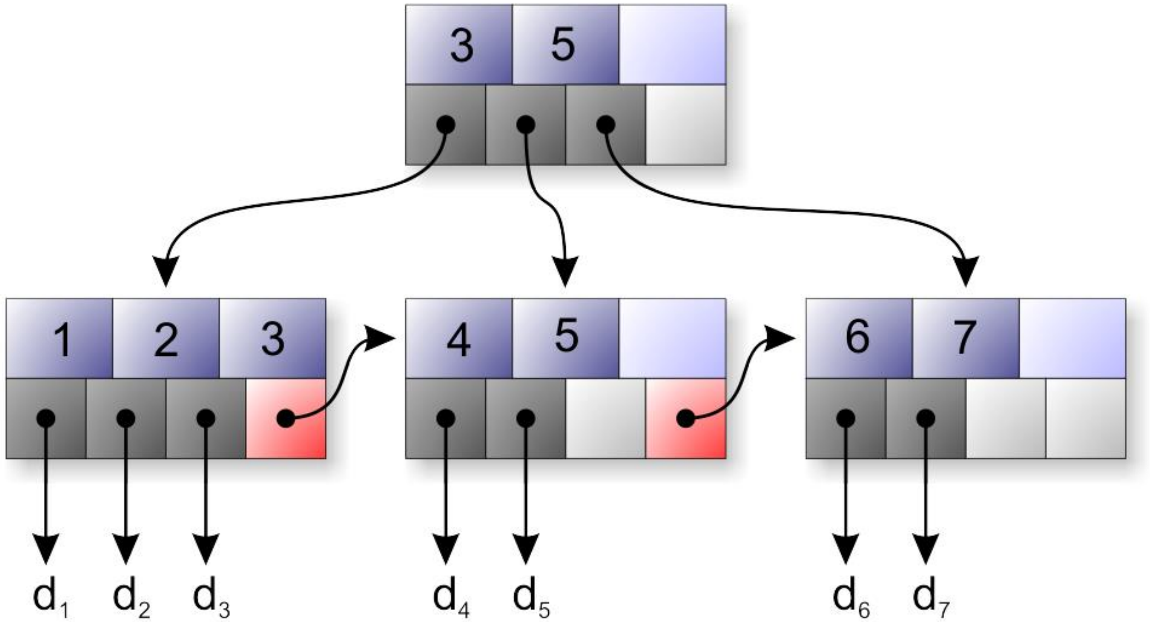
- m = 10 * n and k = 7 ≃ 0,01

Log Structured Merge-tree

Log Structured Merge Trees

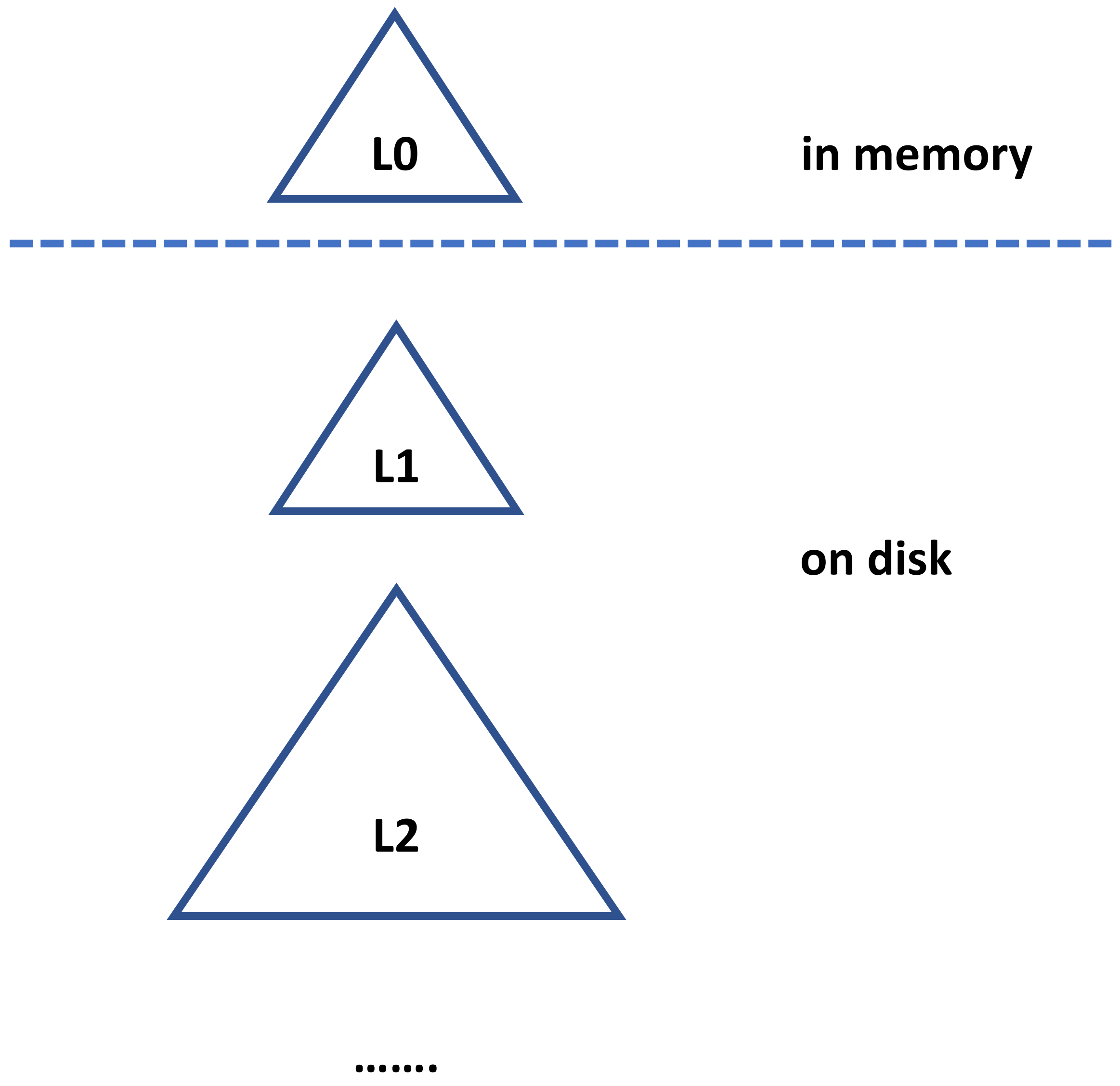
- Optimize I/O operations.
- Used by: Bigtable, LevelDB, Apache Cassandra etc.
- Data organized in B+ trees.
- Advantages: leaves sequentially located,
  leaves are full.

B+ tree

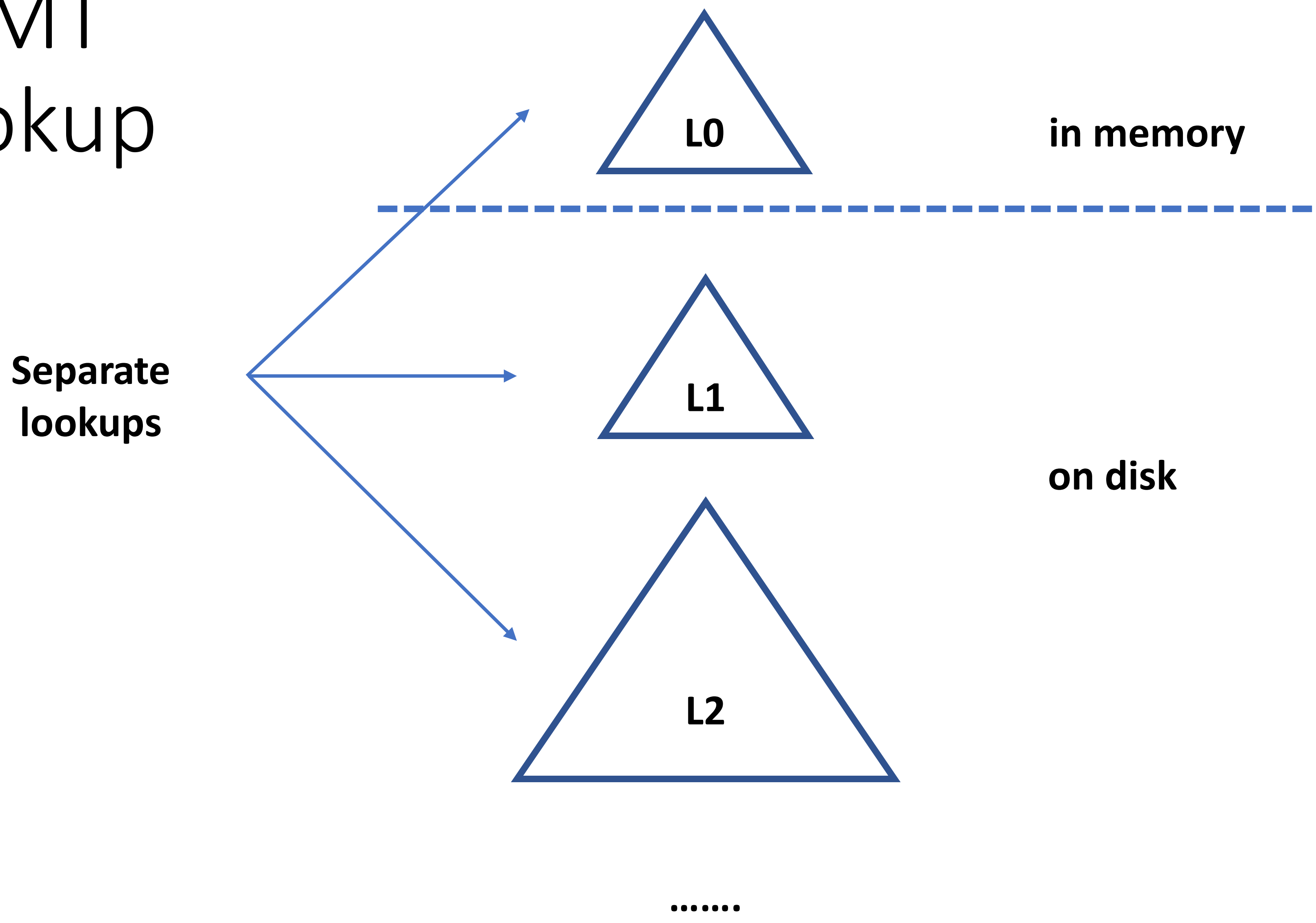


https://commons.wikimedia.org/wiki/File:Btree.png

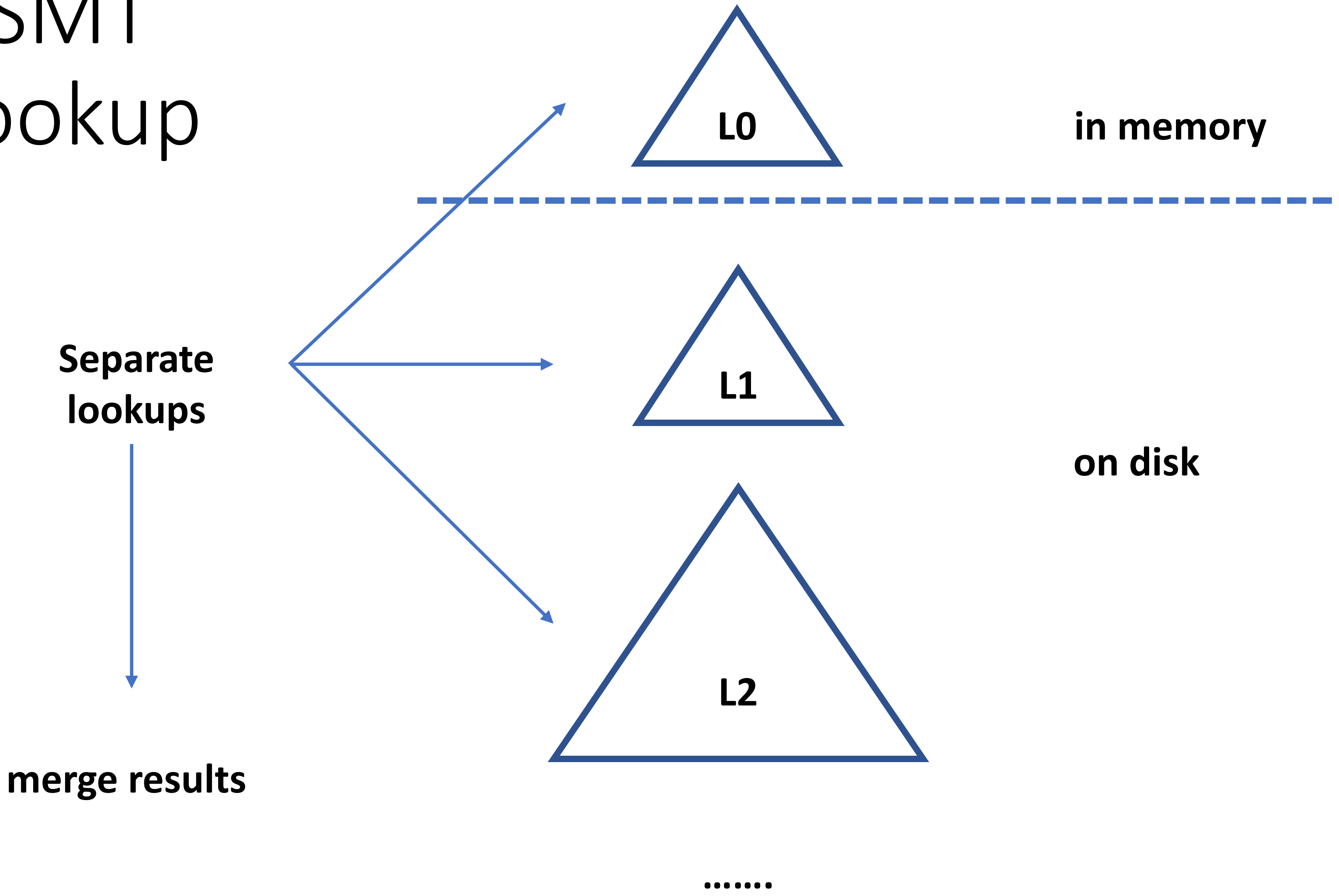
LSMT



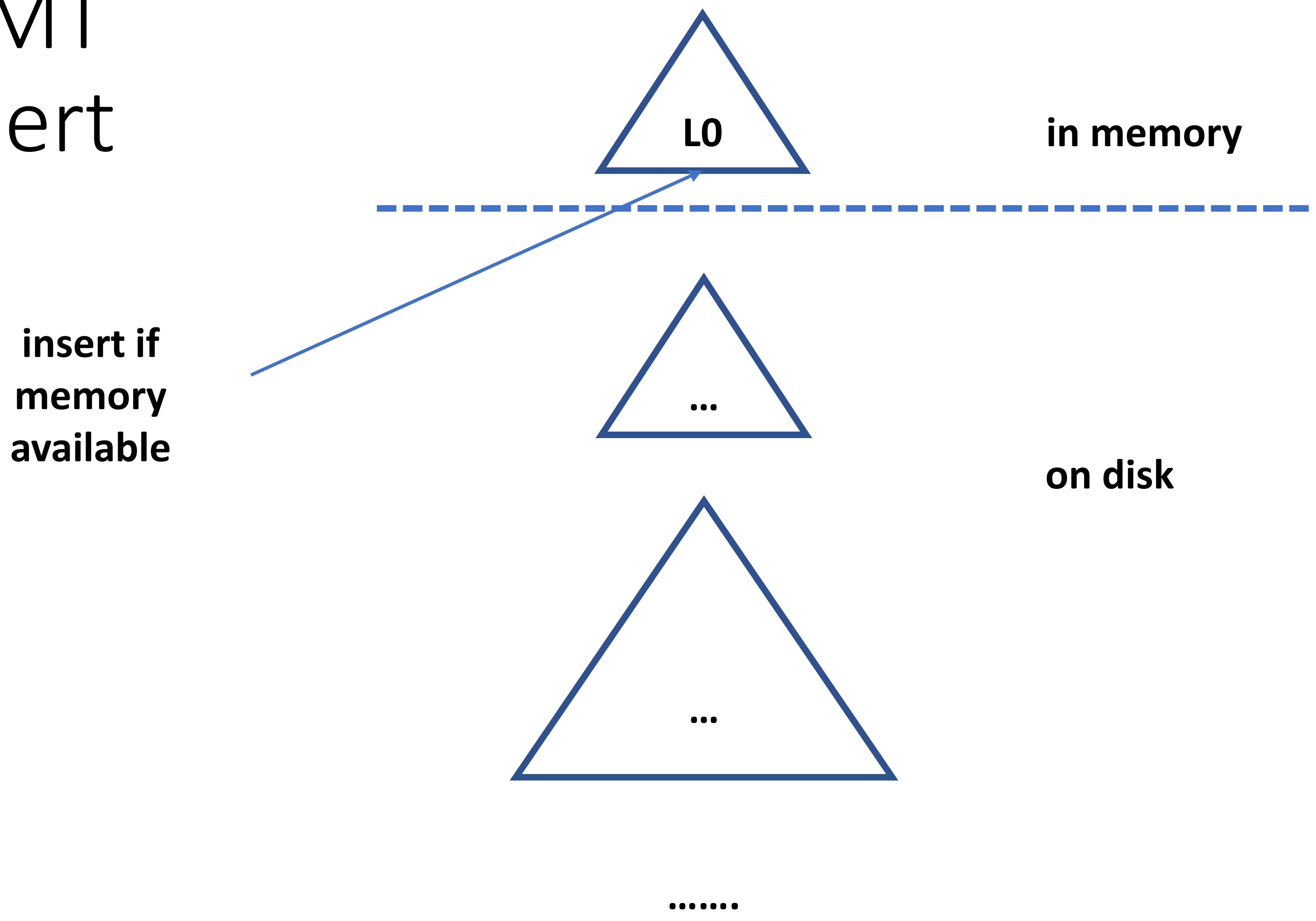
LSMT lookup



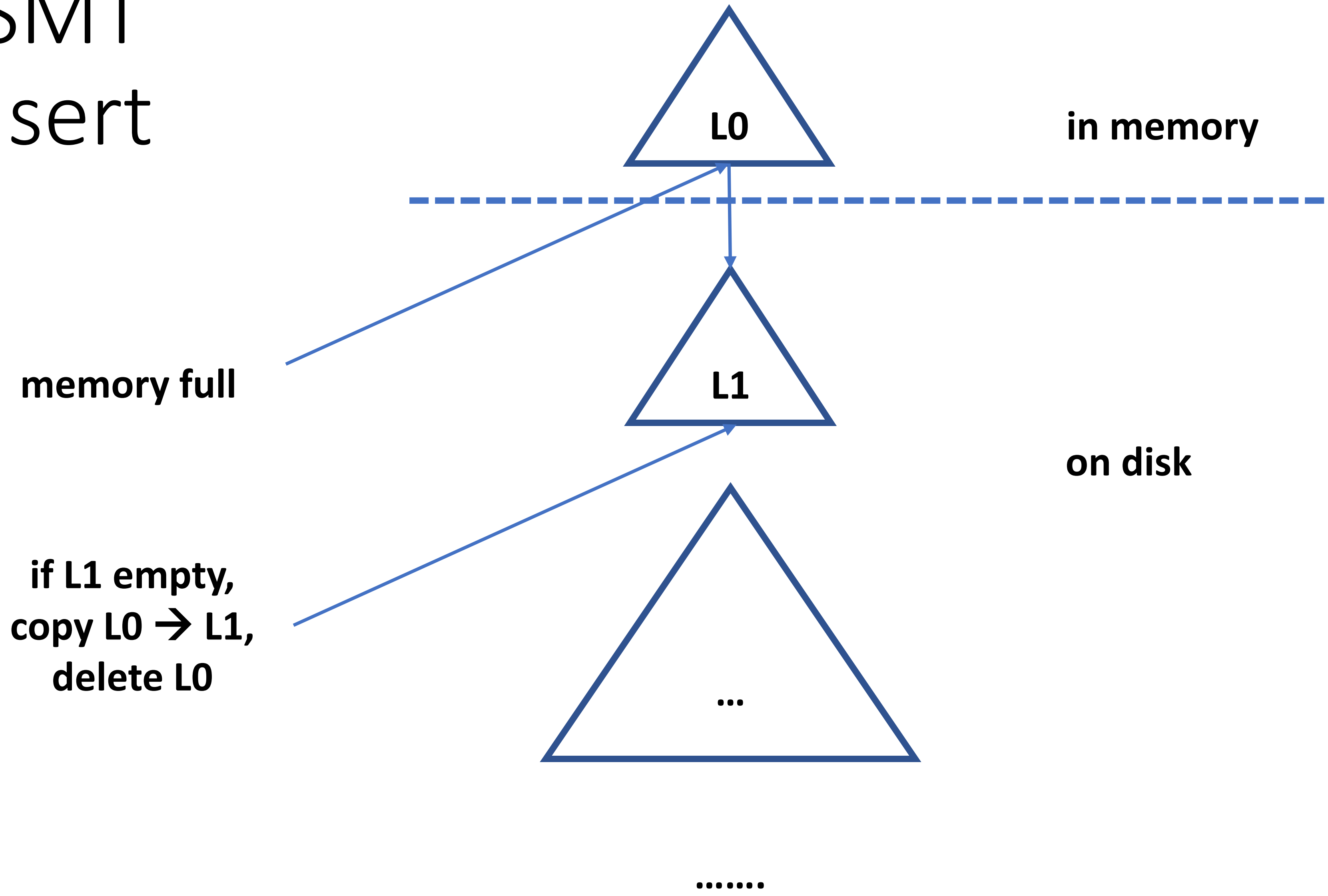
LSMT lookup



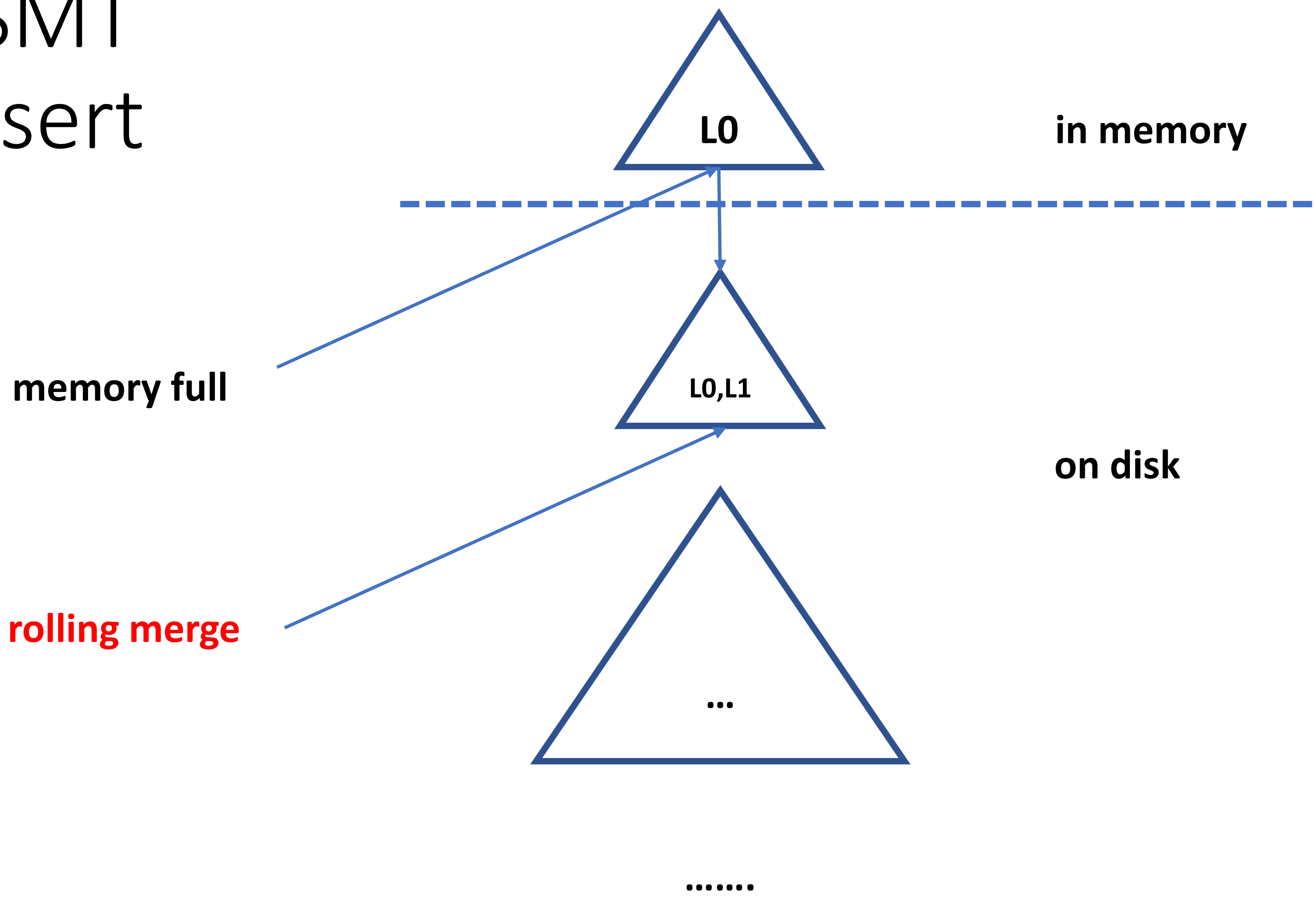
LSMT insert



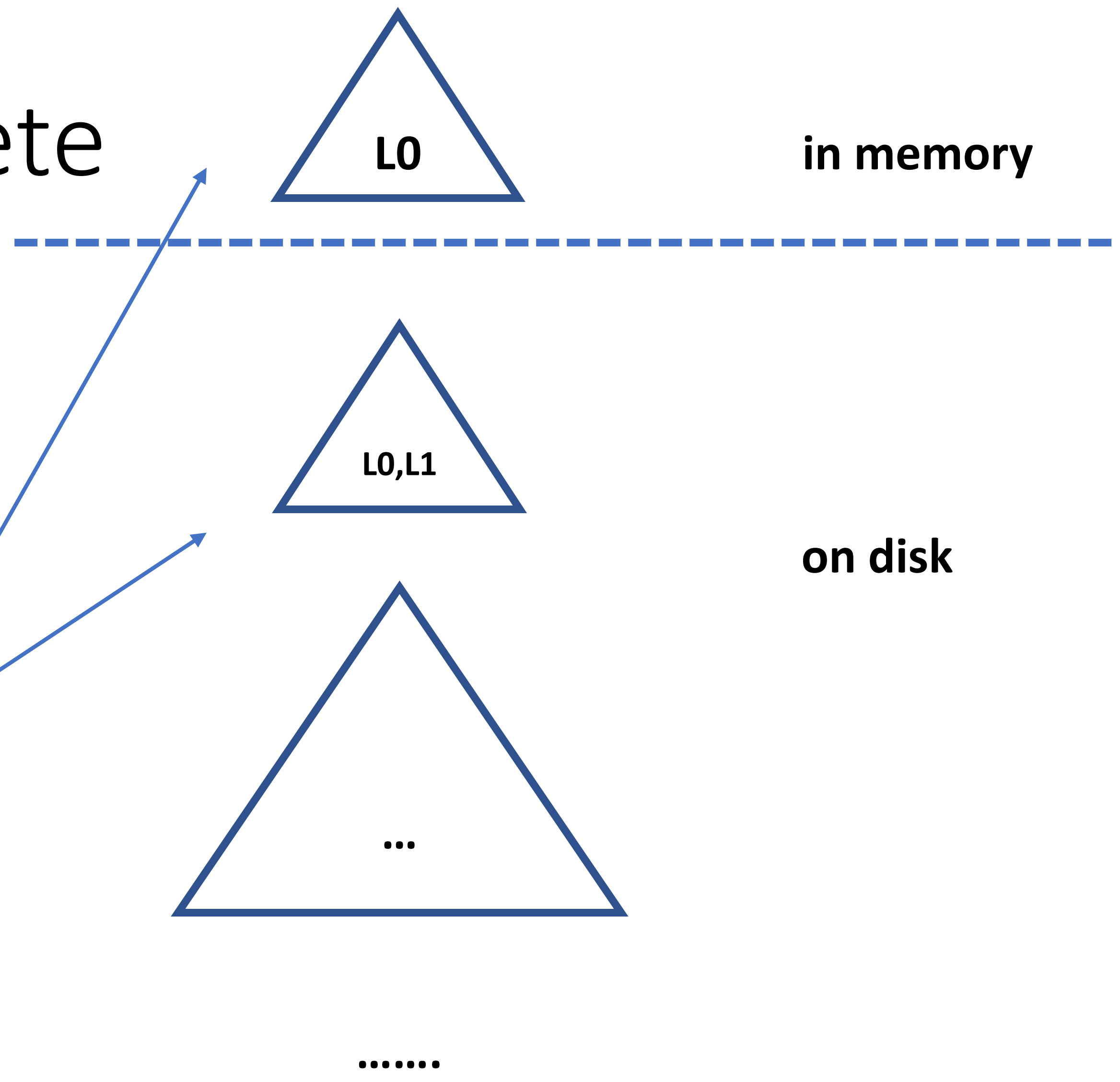
LSMT insert


L0
in memory
memory full
L1
on disk
if L1 empty,
copy L0 → L1,
delete L0
...
....

LSMT insert



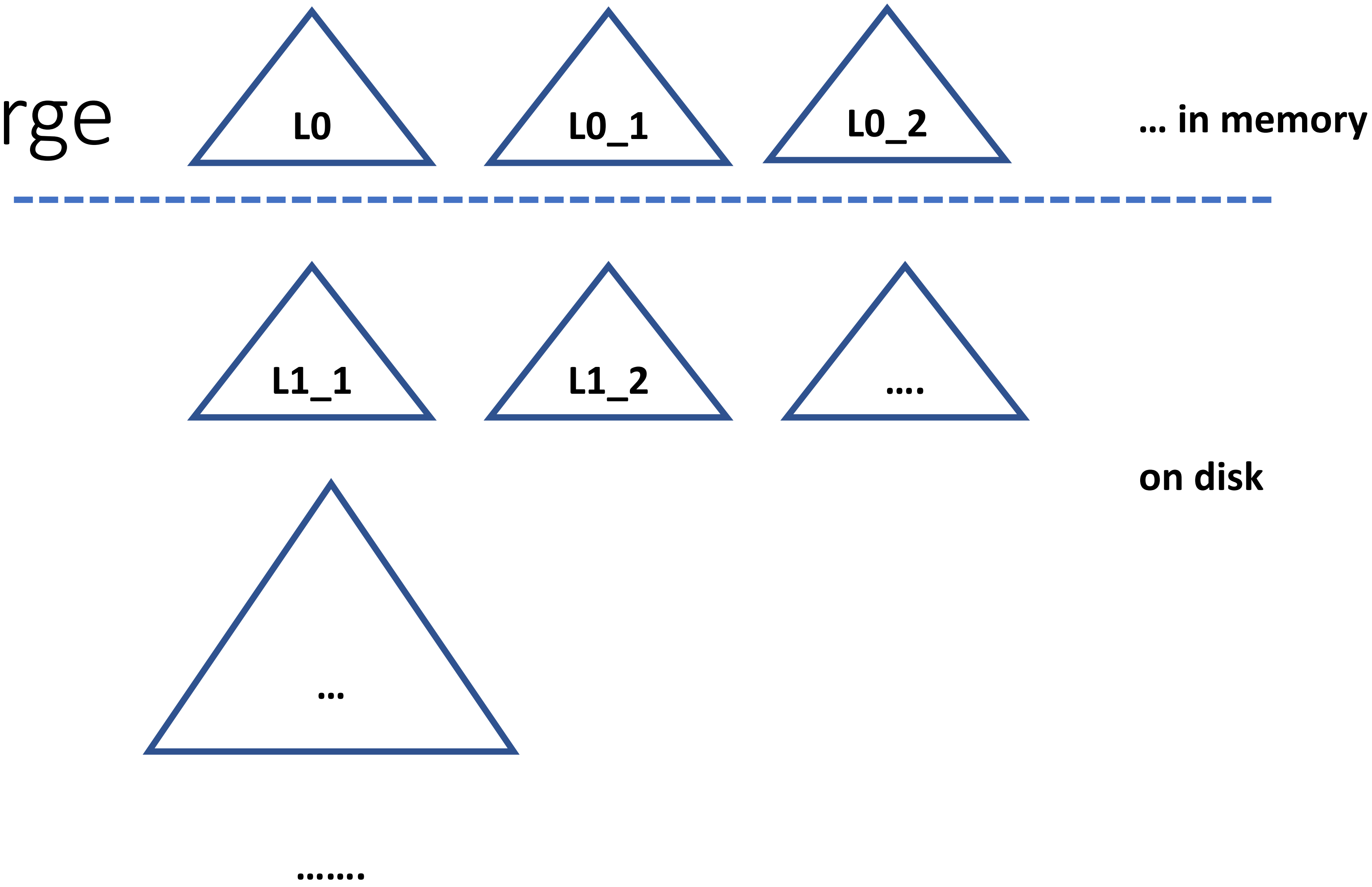
LSMT
update, delete



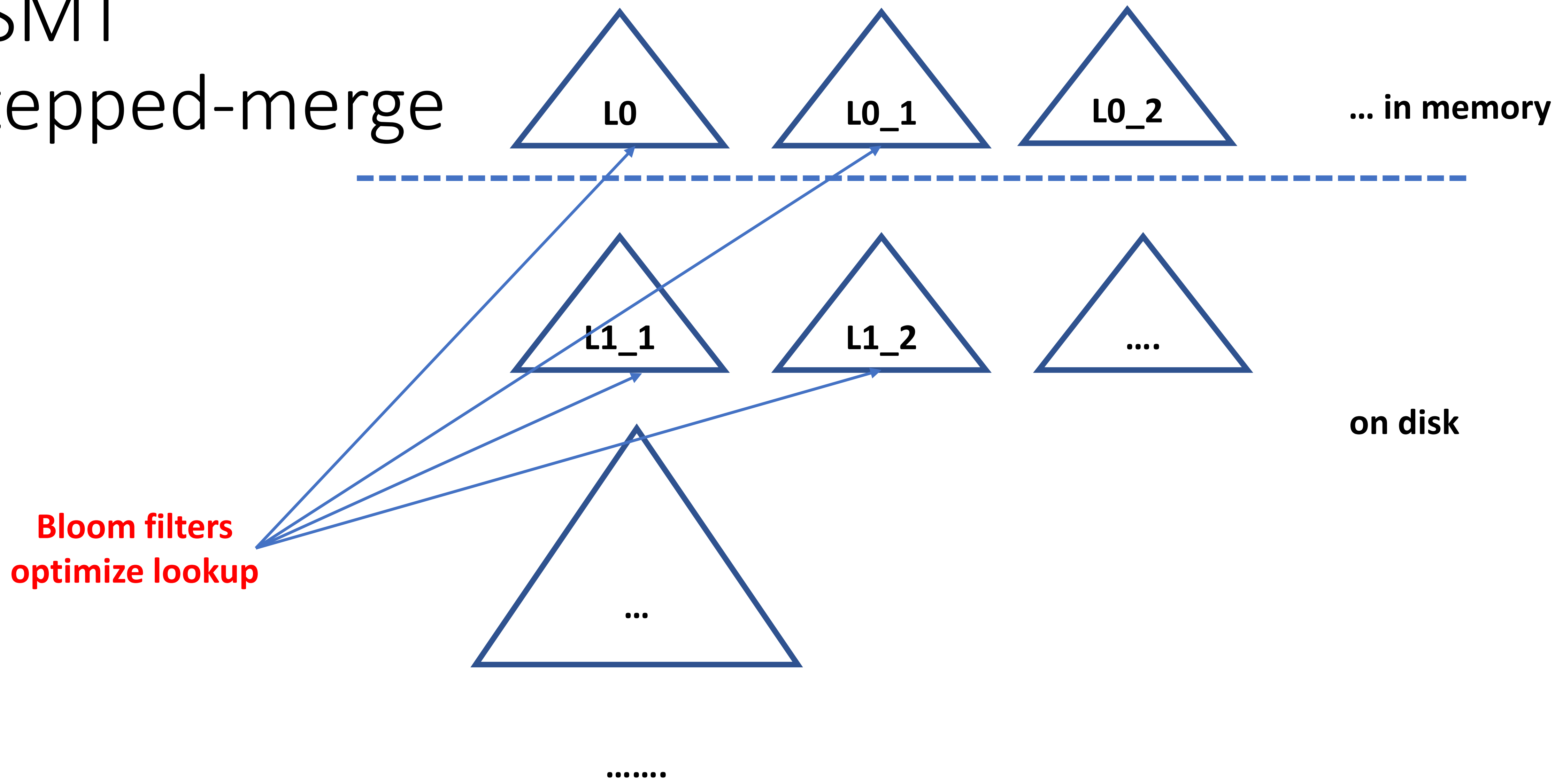
“deleted” or “updated” recoreds,

udapted merge: check if record is deleted or updated

LSMT stepped-merge



LSMT stepped-merge



Materialized views

Materialized views

- redundant data, contents can be inferred from the definition
- immediate view refresh
- deferred view refresh
- incremental update: modify only the affected parts of the materialized view

Materialized views

- Join operation
- Selection
- Projection
- Aggregation