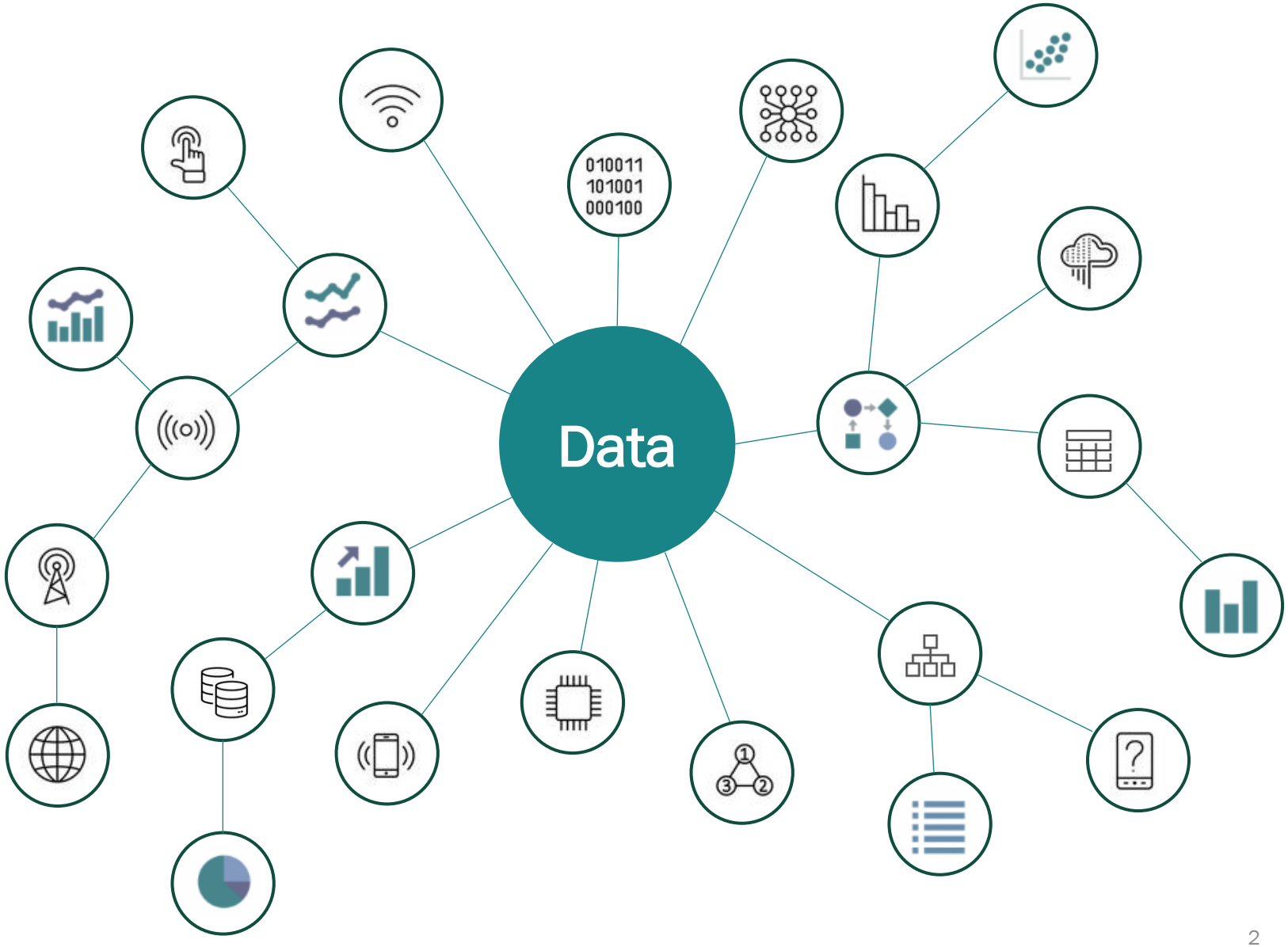
Data

What is Data?

- Facts
- Observations
- Perceptions
- Numbers
- Characters
- Symbols
- Images
- …

Can be interpreted to derive meaning



Data Structures

Structured Data

- Can be organized into rows and columns
- SQL databases and online transaction processing (or OLTP) systems
- Spreadsheets
- 20% of Enterprise Data

Semi-structured Data

- Has some organizational properties but cannot be stored into rows and columns
- Organized using tags and elements
- Emails
  XML Files

Unstructured Data

- Complex, and mostly qualitative information
  Does not have an easily identifiable structure
- Images, videos and audio
- Web pages
- Social media

Data Structure

Structured

- Follows a rigid format and can be organized neatly into rows and columns (well defined schemas – can be represented in a tabular manner)
- Typically seen in databases and spreadsheets
- Sources:
  - SQL databases and online transaction processing (or OLTP) systems
  - Spreadsheets such as excel and google spreadsheets
  - Online forms
  - Sensors such as global positioning systems (or GPS) and radio frequency identification (or RFID) tags; and
  - Network and web server logs

Data Structure

Semi-structured

- A mix of data that has consistent characteristics and data that doesn't conform to a rigid structure
- Has some organizational properties but lacks a fixed or rigid schema (cannot be stored in the form of rows and columns)
- Contains tags and elements, or metadata, which is used to group data and organize it in a hierarchy
- Example/sources:
  - E-mails
  - XML(Extensible Markup Language) and other markup languages
  - Binary executables
  - TCP/IP packets
  - Zipped files
  - Integration of data from different sources
- Data formats:
  - XML and JSON allow users to define tags and attributes to store data in a hierarchical form and are used widely to store and exchange semi-structured data

Data Structure

Unstructured

- Complex, and mostly qualitative information that is impossible to reduce to rows and columns
- Does not have an easily identifiable structure
- No particular formats, sequence, semantics or rules
- Example/sources:
  - Web pages
  - Social media feeds
  - Images in varied file formats (such as JPEG, GIF, and PNG)
  - Video and audio files
  - Documents and PDF files
  - Powerpoint presentations
  - Media logs; and
  - Surveys
- Stored in files and docs for manual analysis or in nosql databases

Data Sources

Web Services

APIs

Data Streams

Social Platforms

Sensor Devices

Text

Images & Videos

Audio Files

Click Streams

IoT Devices

Data File Formats

Delimited text file formats (flat files)

- Store data in rows
- Entities are separated using a delimiter such as comma, tab, colon, vertical bar, space
- CSV
- TSV

Portable document format (PDF)

Spreadsheets

- Special type of flat files
- Microsoft Excel open XML spreadsheet, or XLSX
- Store data in rows and columns

Language Files

- Contain data marked up by tags
- Readable by both humans and machines
- Platform independent and programming language independent
- Great for data transfer over the internet
- XML, JSON

Data File Formats

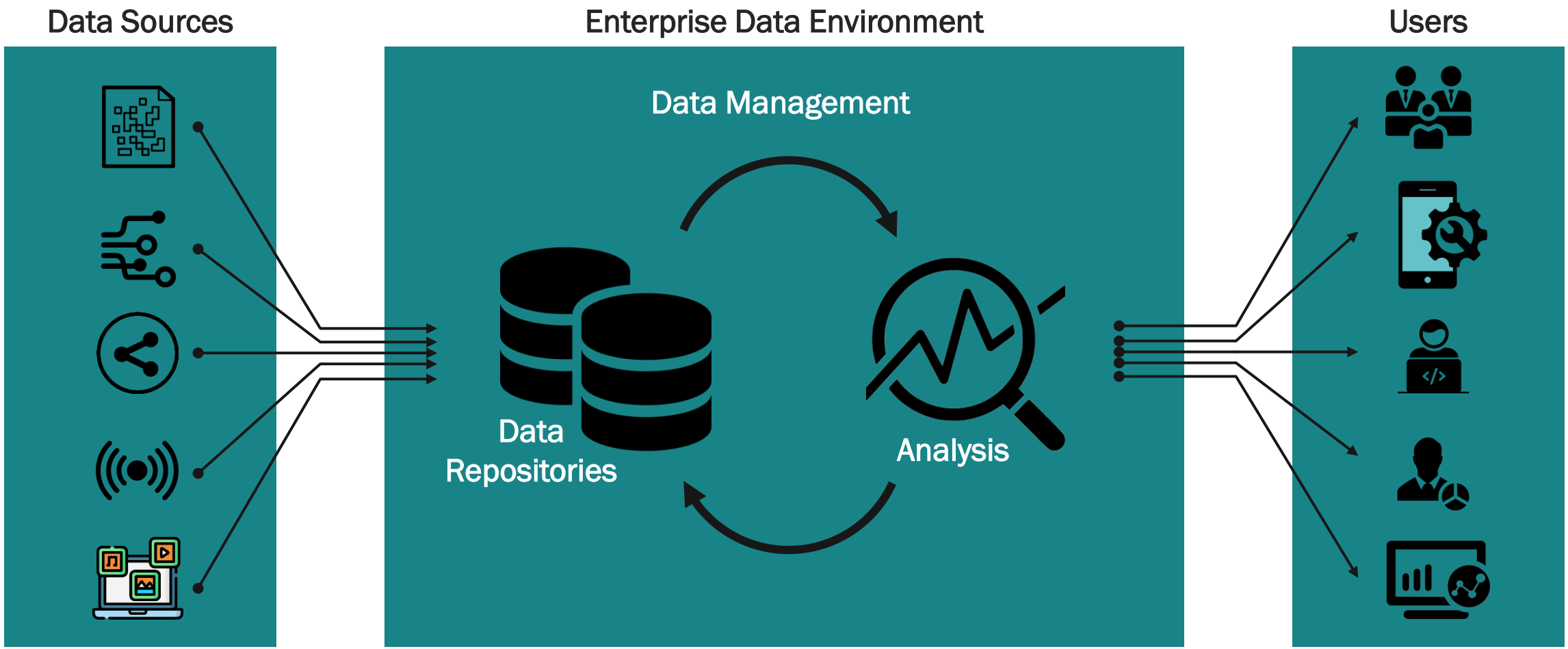
- Delimited text file formats (flat files)
  - Store data in rows
  - Entities are separated using a delimiter such as comma, tab, colon, vertical bar, space
  - CSV
  - TSV
- Spreadsheets(special type of flat files) - Microsoft excel open XML spreadsheet, or XLSX
  - Store data in rows and columns
- Language files - extensible markup language, or XML
  - Contain data marked up by tags
  - Readable by both humans and machines
  - Platform independent and programming language independent
  - Great for data transfer over the internet
- Portable document format, or PDF
- Language files - JavaScript object notation, or JSON
  - Text-based open standard
  - Programming language independent
  - Great for data transfer over the internet

Data Types

	Graph Data		Spatial Data
Relational Data		Data Streams	

Data Ecosystem

Data Ecosystem



Data Ecosystem

- A modern data ecosystem includes a whole network of interconnected, independent, and continually evolving entities. It includes:
  - Data that has to be integrated from disparate sources;
  - Different types of analysis and skills to generate insights;
  - Active stakeholders to collaborate and act on insights generated;
  - And tools, applications, and infrastructure to store, process, and disseminate data as required.

Data Ecosystem

Enterprise Data Environment

- Raw data needs to get organized, cleaned up, and optimized for access by end-users. The data will also need to conform to compliances and standards enforced in the organization
- Requires:
  - Enterprise data repository
  - Data management

Data Ecosystem

Users

- Business stakeholders,
- Applications,
- Programmers,
- Analysts, and
- Data science use cases
- All pulling this data from the enterprise data repository
- Requires:
  - Interfaces
  - APIs
  - Applications

Data Ecosystem

New Technologies

- Emerging technologies affecting the modern data ecosystem:
  - Cloud technologies
    - Limitless storage, high-performance computing, open source technologies, machine learning technologies, and the latest tools and libraries
  - Machine learning
    - Data scientists are creating predictive models by training machine learning algorithms on past data
  - Big data
    - New tools and techniques and also new knowledge and insights

Data Professionals

Data Professionals

Data Engineers

Converts raw data into usable data
Maintains data

Data Analysts

Uses usable data to generate insights

Data Scientists

Uses data analytics and data engineering to predict future using data from the past

Business Analysts

Uses the insights and predictions to drive decisions that benefit and grow the business

Data Professionals

Data Engineers

- Develop and maintain data architectures and make data available for business operations and analysis
- Extract, integrate, and organize data from disparate sources;
- Clean, transform, and prepare data;
- Design, store, and manage data in data repositories
- Enable data to be accessible in formats and systems that the various business applications, as well as stakeholders like data analysts and data scientists, can utilize
- Have good knowledge of programming, sound knowledge of systems and technology architectures, and in-depth understanding of relational databases and non-relational datastores
- Summary:
  - Converts raw data into usable data

Data Professionals

Data Analysts

- Translates data and numbers into plain language, so organizations can make decisions
- Inspect, and clean data for deriving insights;
- Identify correlations, find patterns, and apply statistical methods to analyze and mine data; and
- Visualize data to interpret and present the findings of data analysis
- Answer questions
- Such as “is there a correlation between sales of one product and another."
- Require good knowledge of spreadsheets, writing queries, and using statistical tools to create charts and dashboards, have some programming skills, strong analytical and story-telling skills
- Summary:
  - Uses usable data to generate insights

Data Professionals

Data Scientists

- Analyze data for actionable insights and build machine learning or deep learning models that train on past data to create predictive models
- Answer questions such as “which customers will go bad in the next 6 months”
- Require knowledge of mathematics, statistics, and a fair understanding of programming languages, databases, and building data models. They also need to have domain knowledge
- Summary:
  - Uses data analytics and data engineering to predict future using data from the past

Data Professionals

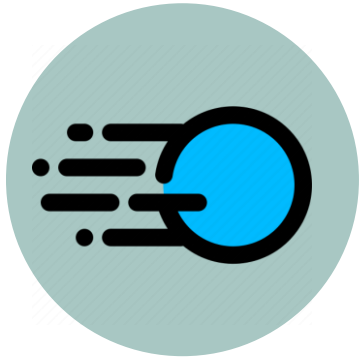
Business Analysts & Business Intelligence (or BI) Analyst

- Business analysts
  - Leverage the work of data analysts and data scientists to look at possible implications for their business and the actions they need to take or recommend
- Business intelligence (or BI) analysts
  - Same as business analysts with a focus on the market forces and external influences that shape their business
  - Provide business intelligence solutions by organizing and monitoring data on different business functions and exploring that data to extract insights and actionables that improve business performance
  - Summary:
    - Use the insights and predictions to drive decisions that benefit and grow the business

Big Data

Big Data

V's of Big Data



Velocity

Volume

Variety

Veracity

Value

Challenges
Not feasible to use conventional tools due to the large scale of data

Solution
Tools with distributed computing resources such as Apache Hadoop, Hive, and Spark

Big Data

There is no one definition of big data, but there are certain elements (v's of big data) that are common across the different definitions, such as

- Velocity: the speed at which data accumulates - Big data is often available in real-time.
- Volume: The quantity of generated and stored data.
- Variety: the diversity of the data (both in type and source) -
- Veracity: the quality and origin of data, and its conformity to facts and accuracy
- Value: The worth in information that can be achieved by the processing and analysis of large datasets
- Challenges:
  - The scale of the data being collected means that it's not feasible to use conventional data analysis tools. However, alternative tools that leverage distributed computing power can overcome this problem.
- Solution:
  - Tools such as Apache spark, Hadoop and its ecosystem provide ways to extract, load, analyze, and process the data across distributed compute resources, providing new insights and knowledge.

Big Data

Big Data Tools

Apache Hadoop

A collection of tools that provides distributed storage and processing of big data

Apache Hive

A data warehouse for data query and analysis built on top of Hadoop

Apache Spark

A distributed data analytics framework designed to perform complex data analytics in real-time

Big Data

Apache Hadoop

A collection of tools that provides distributed storage and processing of big data

- Allows distributed storage and processing of large datasets across clusters of computers
- A node is a single computer, and a collection of nodes forms a cluster
- Can scale up from a single node to any number of nodes, each offering local storage and computation
- Provides a reliable, scalable, and cost-effective solution for storing data with no format requirements
- Hadoop distributed file system, or HDFS, is a storage system for big data that runs on multiple commodity hardware connected through a network
- Provides scalable and reliable big data storage by partitioning files over multiple nodes

Big Data

Apache Hive

A data warehouse for data query and analysis built on top of Hadoop

- Limitations:
- Queries have high latency (not good for apps that require fast response times)
- Read-based, not suitable for transaction processing
- Great for data warehousing task such ETL and data analysis

Big Data

Apache Spark

A distributed data analytics framework designed to perform complex data analytics in real-time

- Has in-memory processing
- Provides interface for major programming languages (python, java, scala, R, ...)
- Can run independently or on other infrastructures such as hadoop

Good To Know...

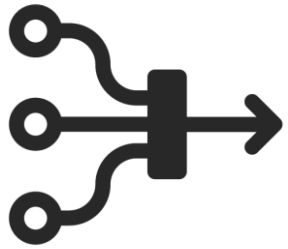
Infrastructure Components

- **Virtual machines**
- **Networking**
- **Application services**
- **Cloud-based services**
  - Amazon (AWS)
  - Google (GCP)
  - IBM (IBM Cloud)
  - Microsoft (Azure)

Database and Data Warehouses

- **RDBMS** (such as MySQL, PostgreSQL, IBM DB2, SQL Server, Oracle, ...)
- **NoSQL** (MongoDB, Cassandra, Redis, Neo4j, ...)
- **Data warehouses** (Oracle Exadata, Amazon Redshift, IBM Netezza, ...)

Data Pipelines



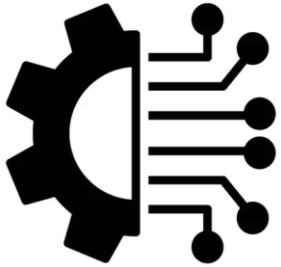
- **Apache Beam**
  - **Airflow**
  - **Dataflow**

Good To Know...

Languages

- Query languages
  - SQL for relational database
  - SQL-like query languages for NoSQL databases
- Programming languages
  - Python, r, java
- Shell and scripting languages
  - Unix/Linux shell and PowerShell

ETL Tools



- IBM Infoshpere
  - AWS Glue
  - Improvado

Big Data Processing Tools

- Apache Hadoop
  - Apache Hive
  - Apache Spark

Data Repositories

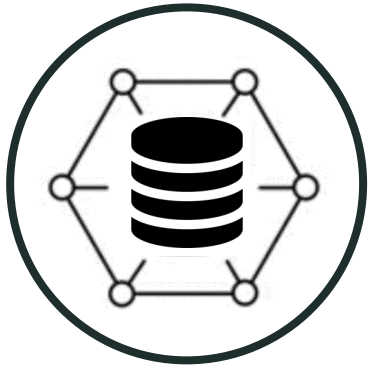
Data Repositories

Data that has been collected, organized and isolated for use in business operations or mined for reporting and data analysis.

Databases

Data Warehouses

Big Data Stores



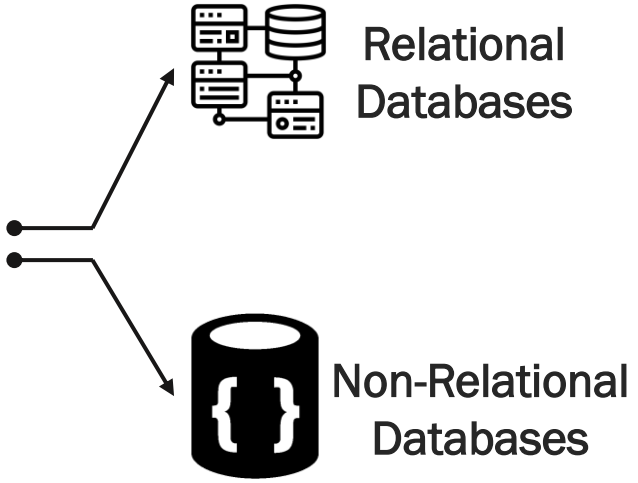
Data Marts

Data Lakes

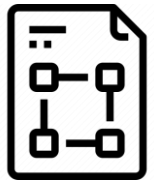
Data Repositories

Databases

A collection of data, or information, designed for the input, storage, search and retrieval, and modification of data.

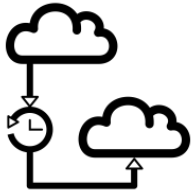


Factors influencing the choice of database:

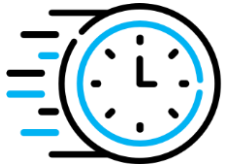


Data type and structure

Querying mechanisms



Latency requirements



Transaction speeds

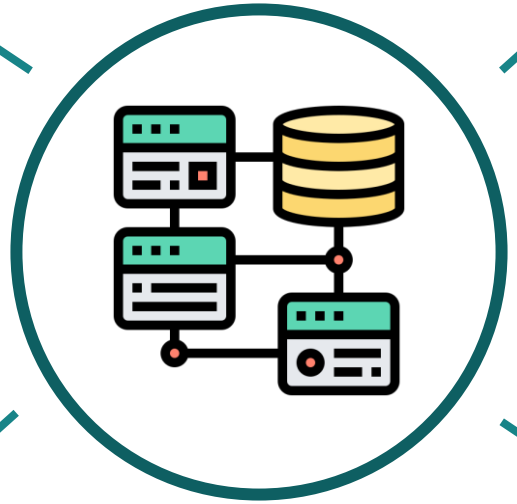
Intended use of the data

Data Repositories

Databases: Relational

Build on the organizational principles of **flat files**

Data organized into a tabular format with **rows** and **columns** (**records** and **attributes**)



A well-defined structure and schema

Structured query language, or **SQL**, is the standard querying language for relational databases

Data Repositories

Databases: Relational

Relational Databases

Optimized storage, retrieval, and processing of data for large volumes of data

Relationships between tables

Fields can be restricted to specific data types and values

VS

SpreadSheets

Limited number of rows and columns
Not optimized for large data

No Relationships between tables

No possibility for restriction

Data Repositories

Databases: Relational

Commercial
Closed Source

ORACLE DATABASE

IBM DB2

Microsoft SQL Server

Open Source
+ Internal/Commercial Support

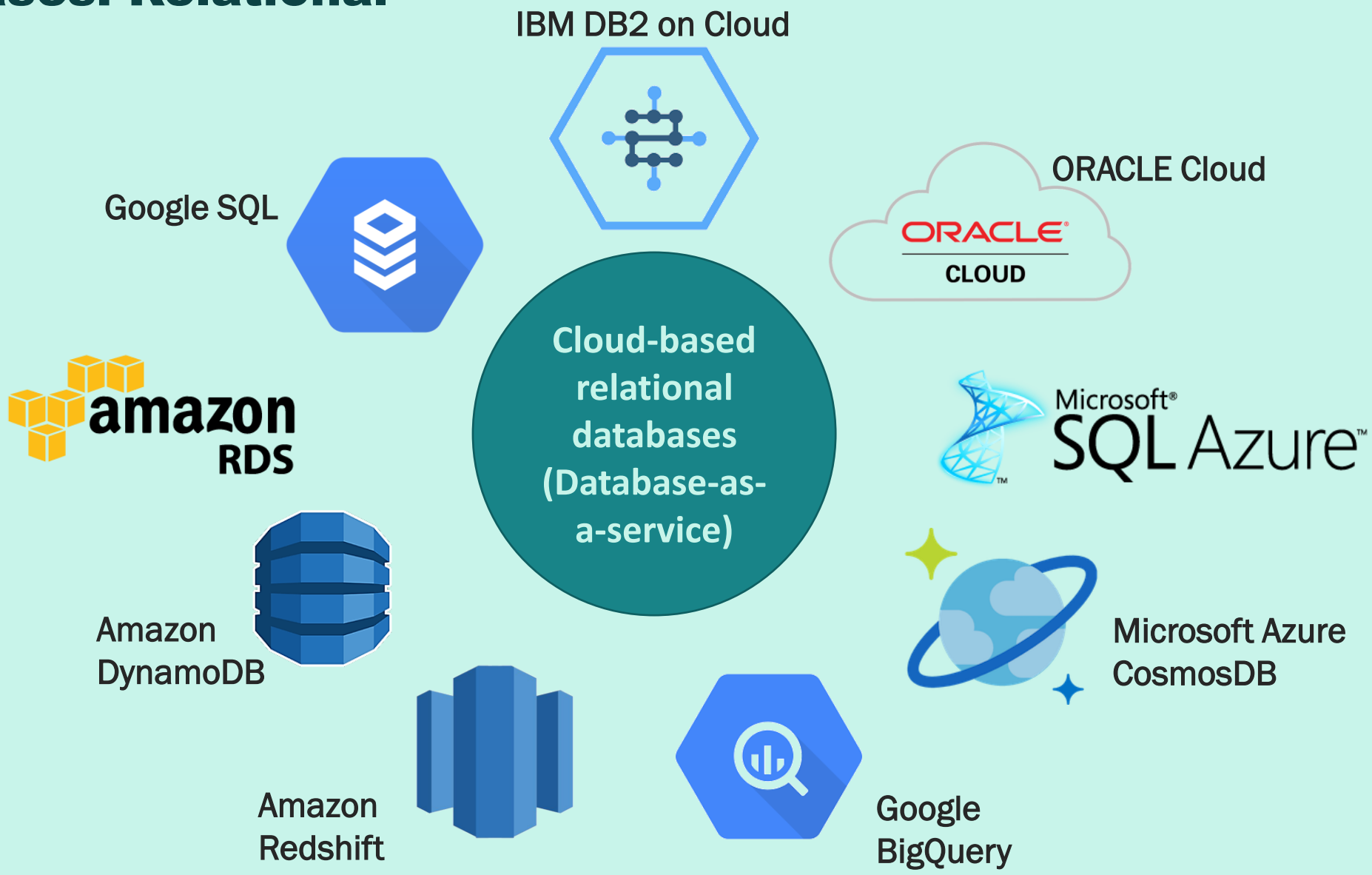
MySQL

PostgreSQL

SQLite

Data Repositories

Databases: Relational

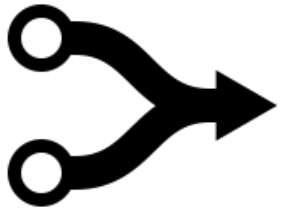


Data Repositories

Databases: Relational

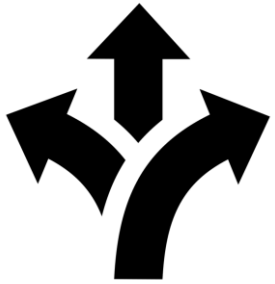
Advantages of the relational databases approach

Reduced redundancy: relational databases minimize data redundancy

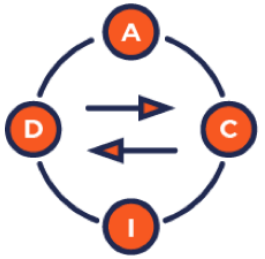


Ability to create meaningful information by joining tables

Ease of backup and disaster recovery



Flexibility: can add new columns, new tables, rename relations, and make other changes while the database is running, and queries are happening



Acid-compliance: data in the database remains accurate and consistent despite failures, and database transactions are processed reliably

Data Repositories

Databases: Relational

Use cases for RDBMS

Online Transaction Processing (OLTP)

Transaction-oriented tasks that run at high rates
Designed to store high volume day-to-day operational data

Online Analytical Processing (OLAP)

Historical data is analyzed for business intelligence
Also include non-relational DBS, data warehouses, data lakes, big data stores

IoT solutions: Internet of Things (IoT)

Data from a system of interconnected objects that communicate through the internet. Also include non-relational DBS.

Data Repositories

Databases: Relational

Limitations of RDBMS

Only for Structured data

Does not work well with semi-structured and unstructured data. Not suitable for extensive analytics on such data

Have a limit on the length of data fields

Relational databases impose limits on field lengths. While designing the database, it is necessary that you specify the data volume you intend to introduce within any field.

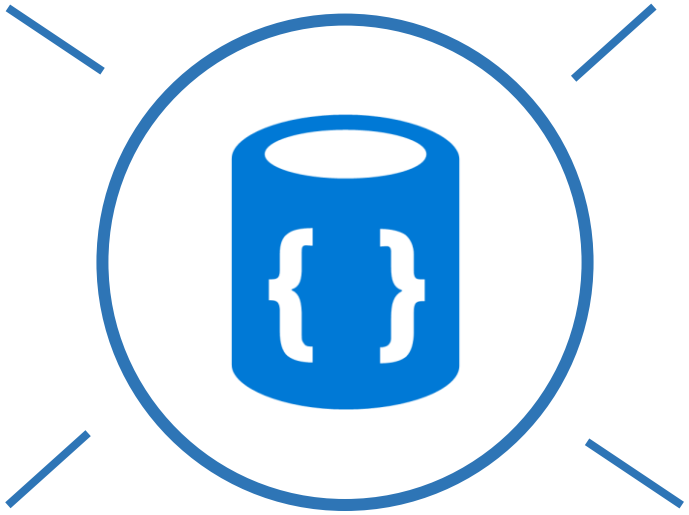
Lack of Flexibility

Inadequacy to operate with languages outside of SQL

Data Repositories

Databases: Non-Relational

Provides flexible schemas for the storage and retrieval of data
Makes it possible to store data in a schema-less or free-form fashion



Influenced by advances in **cloud computing**, the **internet of things**, and **social media** proliferation

Emerged in response to the **volume**, **diversity**, and **speed** at which data is being generated today

Built for **speed**, **flexibility**, and **scale**

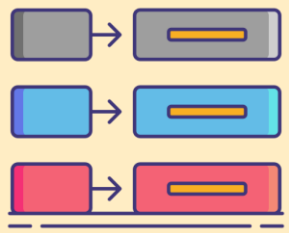
NoSQL (not only SQL) is widely used for processing big data

Some non-relational databases come with their own querying tools
CQL (Cassandra)
GraphQL (Neo4J)

Data Repositories

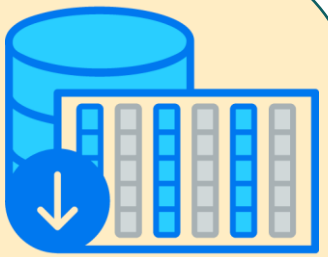
Databases: Non-Relational

Types based on storing model

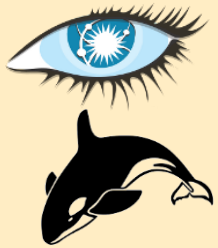


Key-value store

Document based



Column based



Graph based

Data Repositories

Databases: Relational

Types based on storing model

- Key-value store
  - Data stored as a collection of key-value pairs
  - key represents an attribute of the data
  - great for storing user session data and user preferences, making real-time recommendations and targeted advertising
  - examples:
    - Redis,
    - Memcached
    - DynamoDB
- Document based
  - store each record and its associated data within a single document
  - preferable for eCommerce platforms, medical records storage, CRM platforms
  - Examples:
    - MongoDB
    - DocumentDB
    - CouchDB
    - Cloudant

Data Repositories

Databases: Relational

Types based on storing model

- Column based
  - store data in cells grouped as columns of data instead of rows
  - great for systems that require heavy write requests, storing time-series data, weather data, and IoT data
  - examples:
    - Cassandra
    - Apache HBASE
- Graph based
  - use a graphical model to represent and store data
  - excellent choice for working with connected data, which is data that contains lots of interconnected relationships
  - great for social networks, real-time product recommendations, network diagrams, fraud detection, and access management
  - Examples:
    - Neo4J
    - CosmosDB

Data Repositories

Databases: Non-Relational

Advantages of NoSQL

Flexibility in structure

Ability to handle large volumes of structured, semi-structured, and unstructured data

Scalability and Availability

Simpler design, better control over availability, and improved scalability that enables you to be more agile, more flexible, and to iterate more quickly

Supports distributed computing

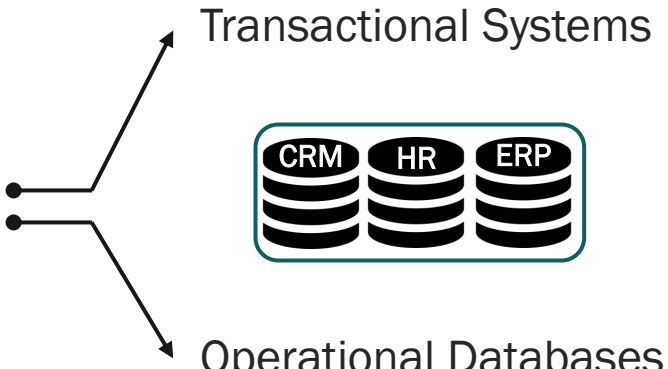
Ability to run as distributed systems scaled across multiple data centers, which enables them to take advantage of cloud computing infrastructure; an efficient and cost-effective scale-out architecture that provides additional capacity and performance with the addition of new nodes

Data Repositories

Data Warehouses

Works as a central repository that

Merges information coming from multiple sources



Consolidates it through ETL process or through ELT process, into one comprehensive database for analytics and business intelligence.

Data Repositories

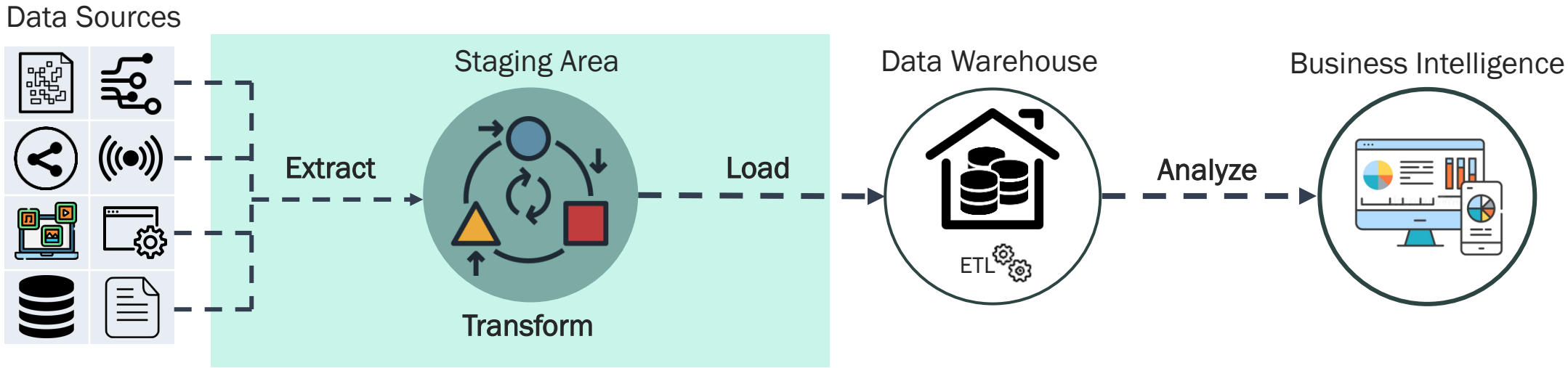
Data Warehouses

Works as a central repository that

- merges information coming from multiple sources
  - store relational data from transactional systems and operational databases such as CRM, ERP, HR, and Finance applications
  - also store non-relational data
- consolidates it through ETL process (extract, transform, and load) or through ELT process (extract, load, transform), into one comprehensive database for analytics and business intelligence.
  - ETL Process
    - extract data from different data sources,
    - transform the data into a clean and usable state, and
    - load the data into the enterprise's data repository
  - ELT Process

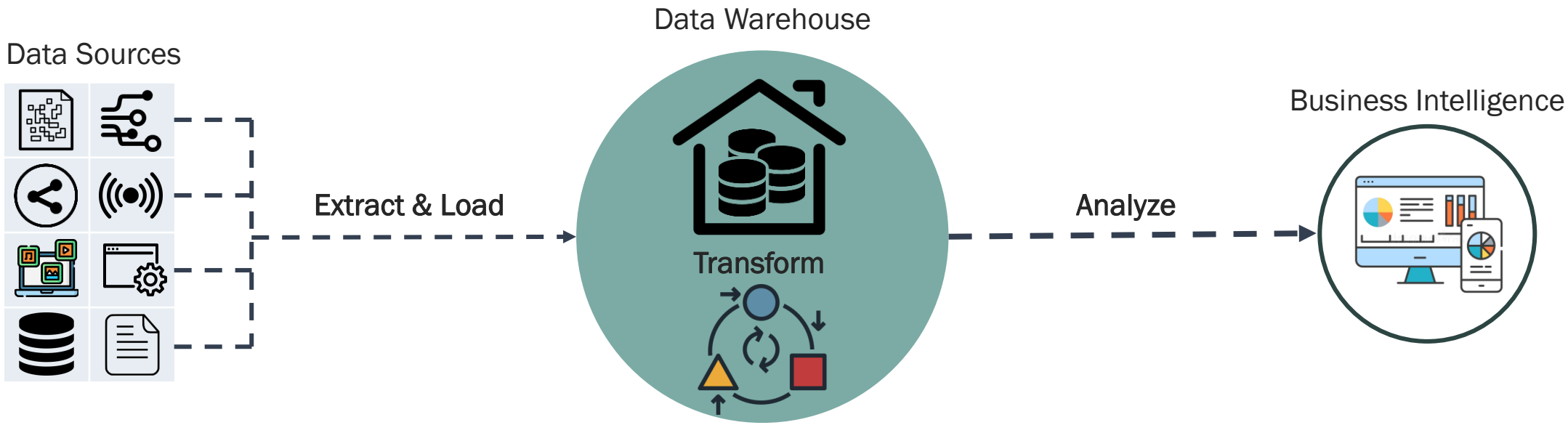
Data Repositories

Extract, Transform, Load



Data Repositories

Extract, Load, Transform



Data Repositories

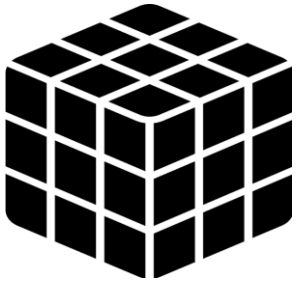
Data Warehouses

Top Tier

Reporting Tools, Data Mining Tools, Analysis Tools

Business Intelligence & Analysis

Middle Tier

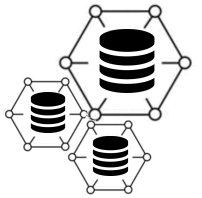


OLAP Server and Tools

Bottom Tier

ETL

Data Warehouse



Data Marts

Data Warehouse

Data Sources

Data Repositories

Data Warehouses

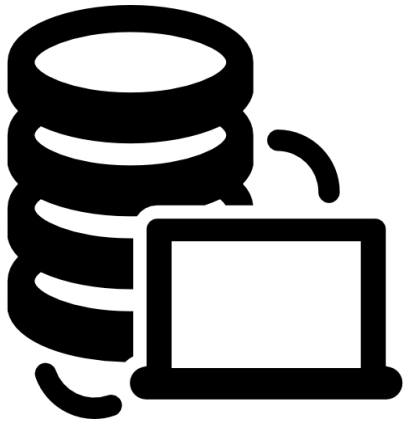
On-Premise Data Warehouse

Cloud Data Warehouse

- Lower Costs
- More Scalability
- Limitless Storage and Computation
- Faster Disaster Recovery

Database Deployment

Database Deployment



Local Deployment

Client/Server Deployment

Cloud Deployment

2-Tier Architecture

3-Tier Architecture

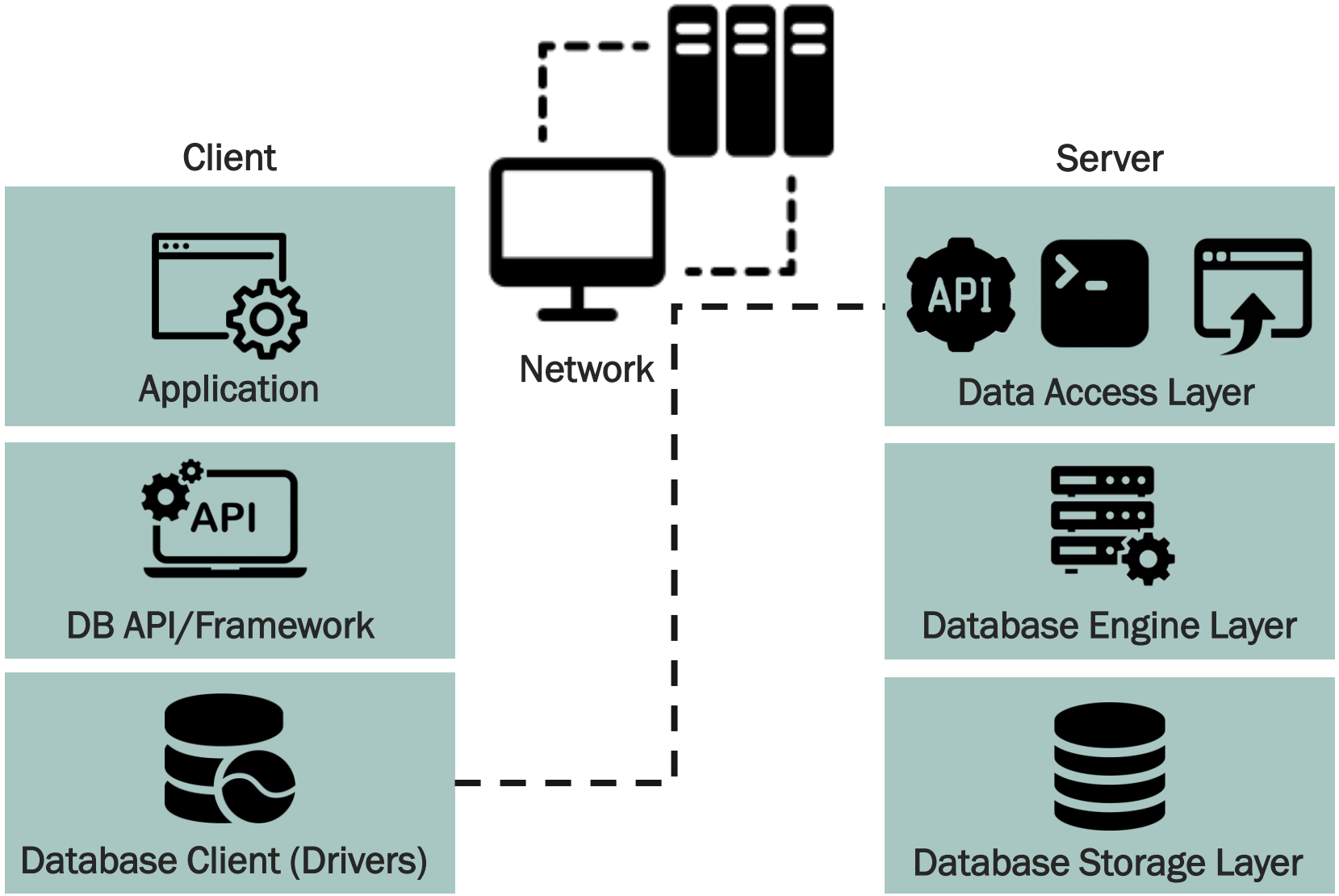
Database Deployment

Local Deployment

- Local/Desktop
- Also known as single-tier architecture
- Database resides on the user's system
- Access is often limited to a single user (single user environment)
- Good for development/testing and database embedded in local applications

Database Deployment

Client/Server Deployment



2-Tier Architecture

3-Tier Architecture

Database Deployment

Client/Server Deployment

- Client/server
- Database resides on a remote database server
- Users access database from a client system (webpage or local application)
- Sometimes there is a middle-tier (application server layer)
- Good for multi-user scenarios and production environments
- Also known as 2-Tier architecture
  - Client application and database server run in separate tiers
  - Application (C, Java, Python, …) connects to the database through an API or framework
  - Database interface communicates through a database client
  - Database management system on the server includes multiple layers
    - Data access layer
      - Includes interface for different types of clients
      - API (such ODBC or JDBC)
      - CLP (command line processor)
      - Prop (vendor specific or propriety interfaces)
    - Database engine layer
    - Database storage layer
      - Local storage on the same device
      - Reside physically on network storage

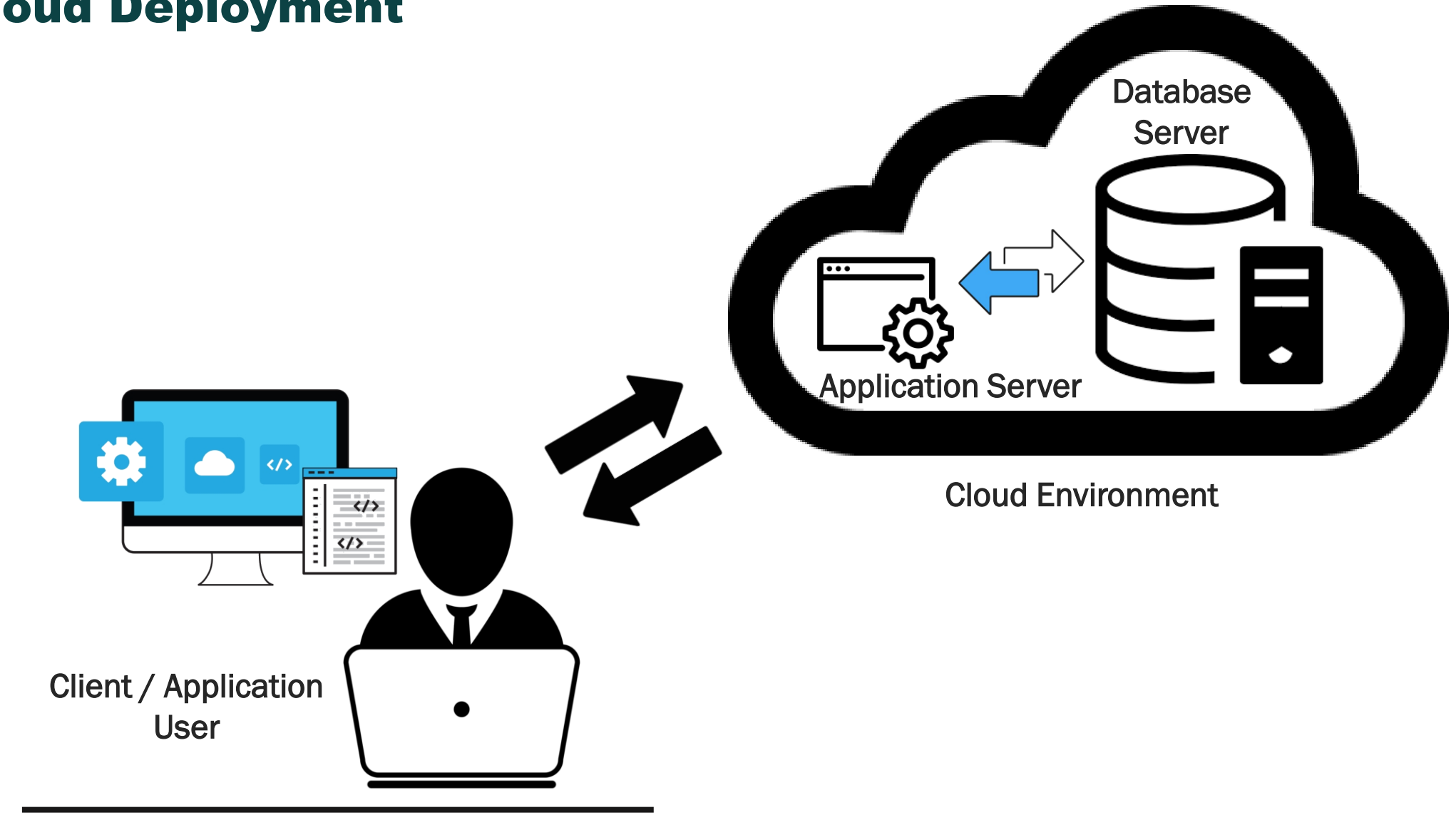
Database Deployment

3-Tier Architecture

- Application presentation layer and business logic layer reside in different tiers
- Users interact with a client presentation layer (interface) such as a mobile application or a desktop application or a web browser
- Client application communicates with application server
- Application server communicates with a database server

Database Deployment

Cloud Deployment



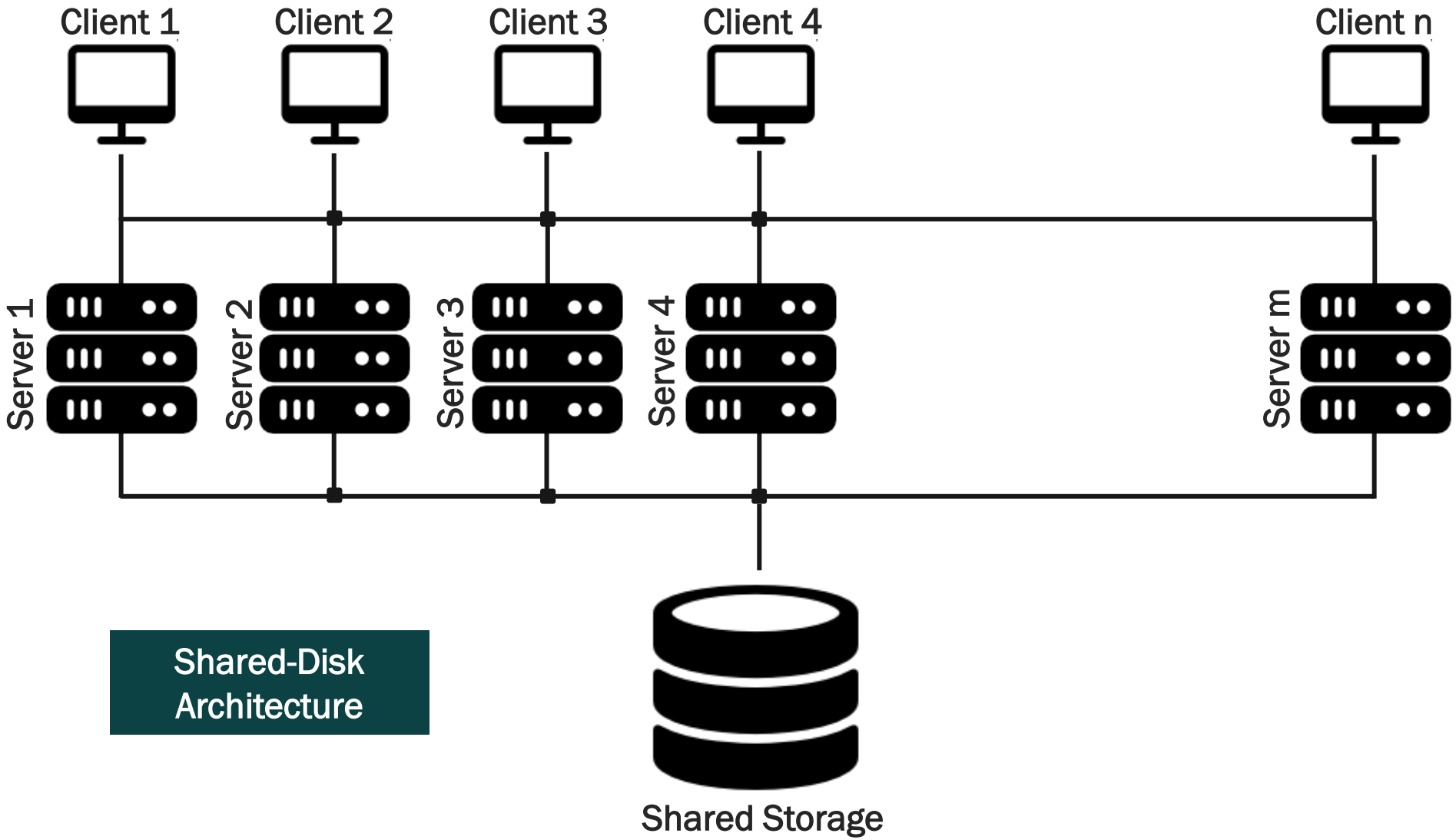
Database Deployment

Cloud Deployment

- Database resides in a cloud environment
- No need to download or install softwares
- Easy access to the cloud through an application server layer or interface in the cloud

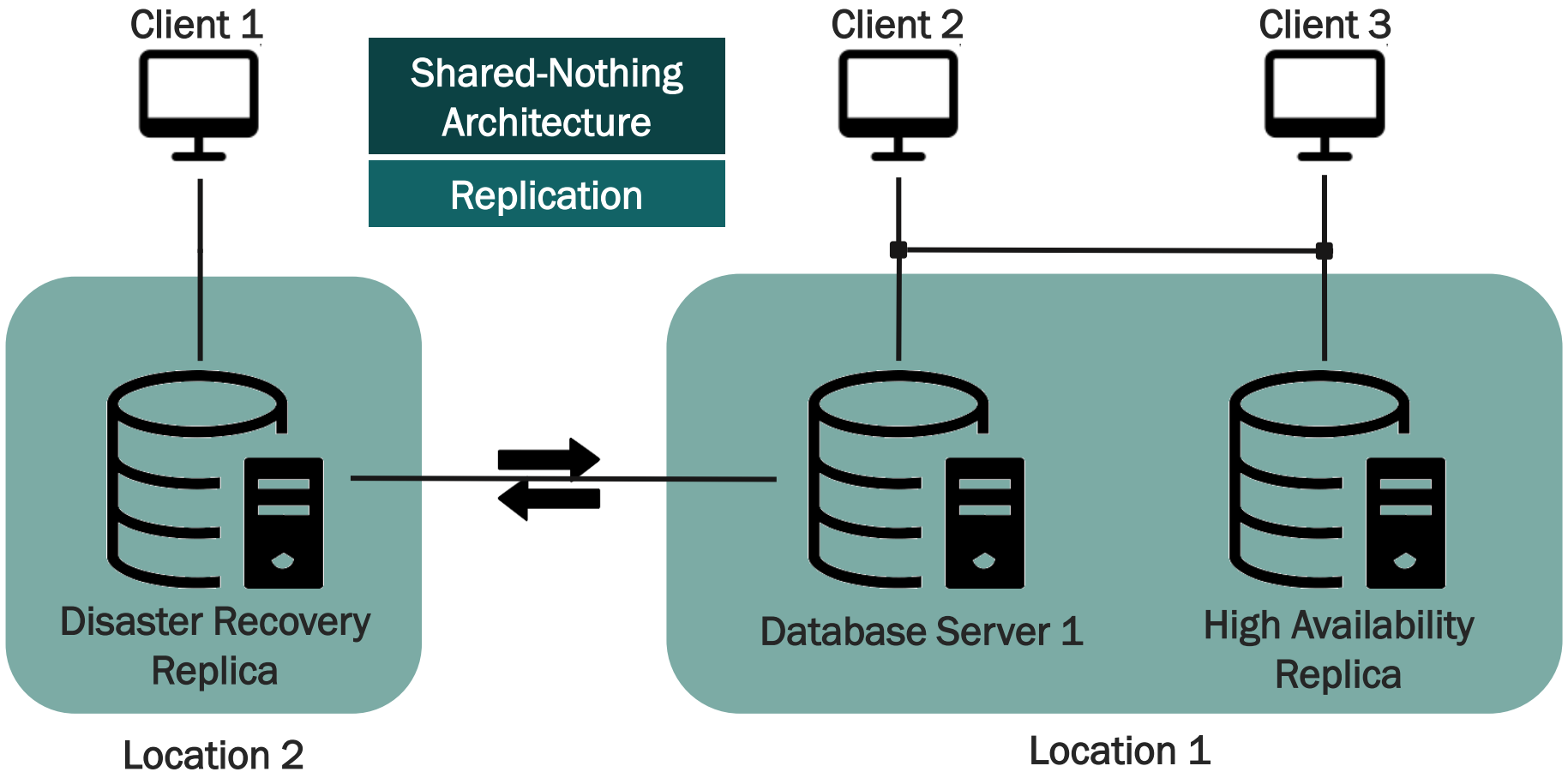
Database Deployment

Distributed Architecture



Database Deployment

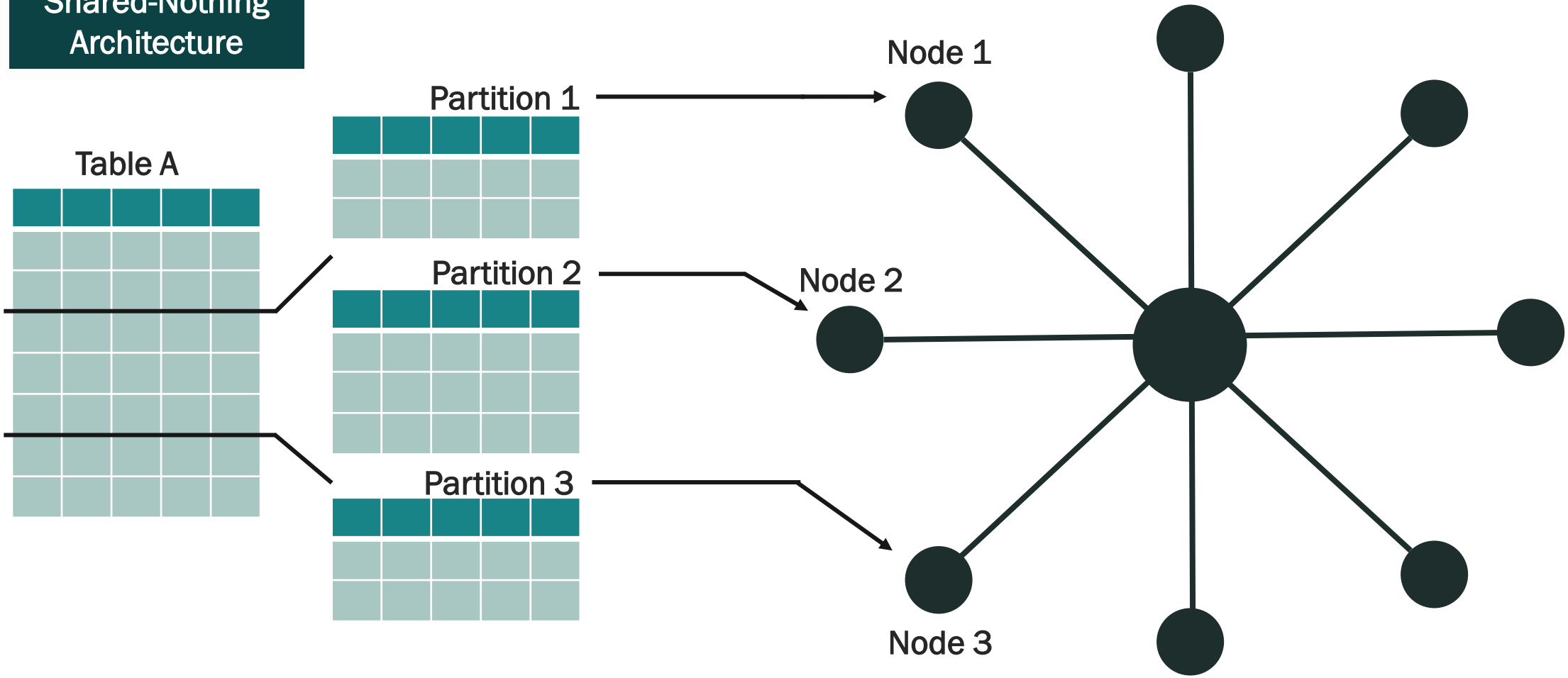
Distributed Architecture



Database Deployment

Distributed Architecture

Shared-Nothing Architecture



Partitioning

Sharding

Database Deployment

Distributed Architecture

Where high availability and high scalability is required, RDBMs provide distributed architecture.

- The main types are:
  - Shared-Disk architecture
  - Multiple database servers process the workload in parallel
  - Share common storage
- Shared-Nothing architecture
  - Replication
    - Changes taking place on database server are replicated in one or more replicas
    - High availability replicas
    - Disaster recovery replicas
  - Partitioning and Sharding
    - Tables with large amount of data are partitioned into multiple logical partitions
    - If each partition is placed in a different node it is called sharding
    - Each shard contains its computing resources (CPU, Memory, Storage…)
    - Queries from the client are processed in parallel on multiple nodes or shards
    - Common for data warehousing

Database Users

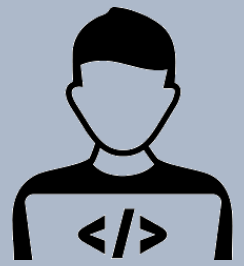
Database Users

Data Engineers

Access type:
Create / Manage

Access through:

Application Developers



Access type:
Read / Write

Access through:

Data Scientists

Access type:
Read

Access through:

+ SQL Query Tools

Business Analysts

Access type:
Read

Access through:

+ SQL Query Tools

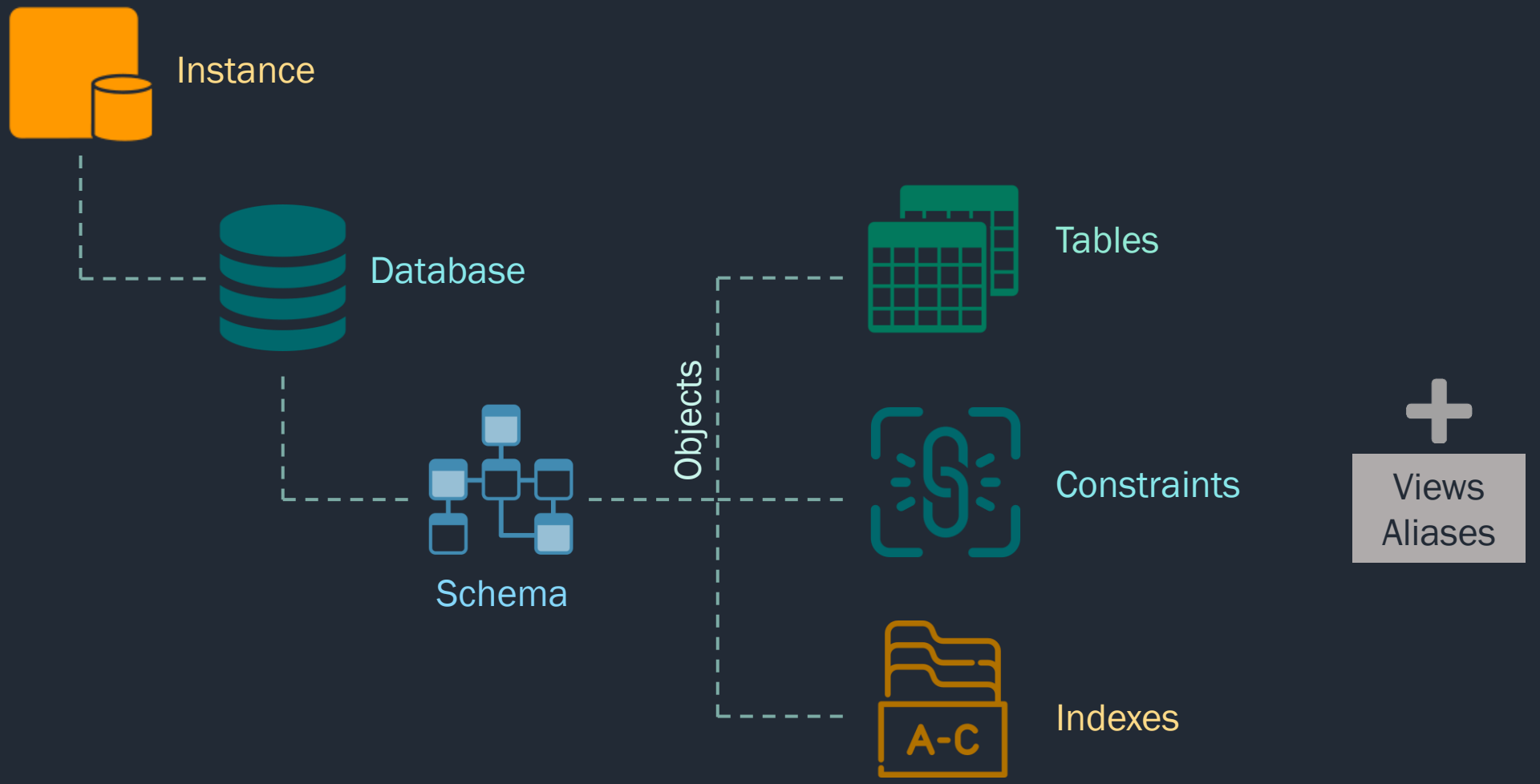
Database Users

There are basically 3 types of users:

- Data Engineers
  - Create / Manage database
  - Access through Command Line, API, or GUI
- Data Scientist and Business Analysts
  - Typically read-only but also write
  - Use Jupyter, R studio, Zapplin, SAS, SPSS
  - Or BI Tools such as Excel, IBM Cognos, Microsoft PowerBI, Tableau, MicroStrategy
- Application Developers
  - Read / Write
  - Use SQL interfaces and APIs or ORM frameworks

Database Hierarchy

Database Hierarchy



Database Hierarchy

Instance

- Is a logical boundary for a database or set of databases where you organize database objects and set of configuration parameters.
- Every database within an instance is assigned a unique name, has its set of system catalog tables which keep track of the objects within the database and has its own configuration files.
- Can have more than one instance on a physical server.
- Allows isolation between databases(we can have different configuration and access for each).

Database Hierarchy

Database

- We already discussed this: a database stores, manages and provides access to data
- Contains objects like tables, views, indexes, functions, triggers and packages

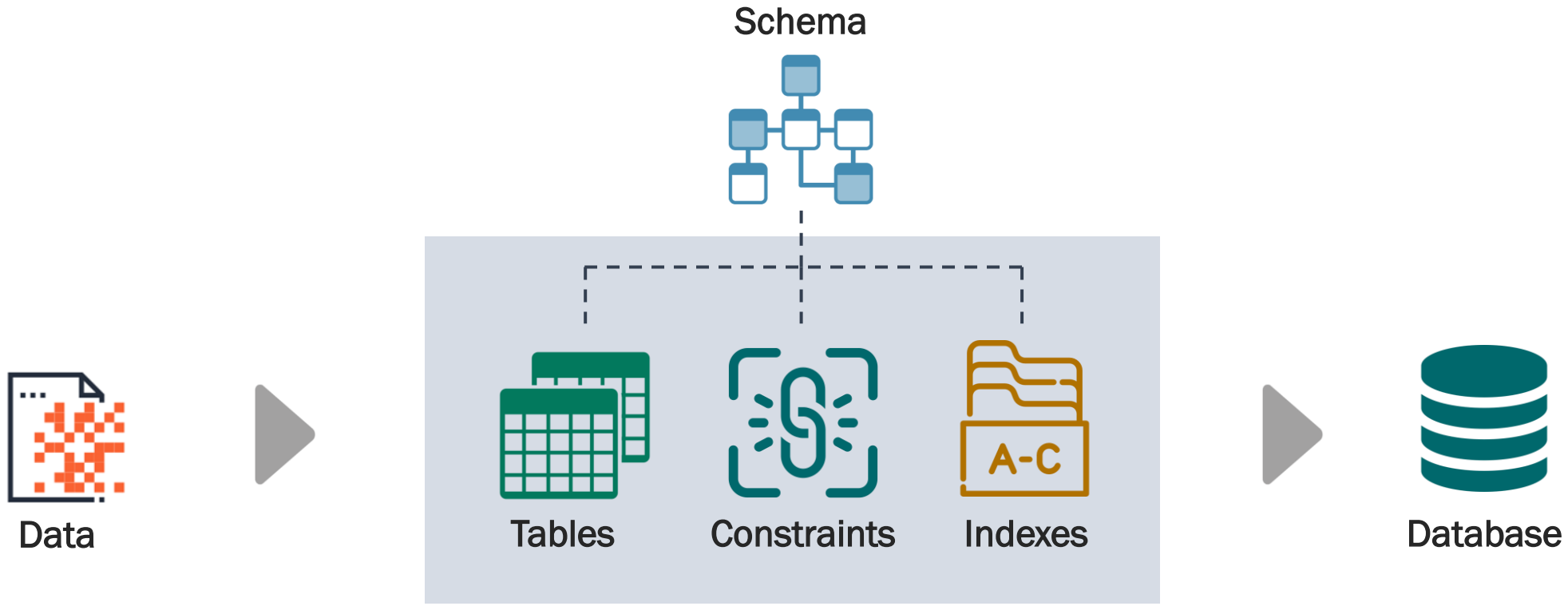
Database Hierarchy

Schema

- Organizes the database objects
- Contains tables, views, nicknames, functions, triggers, packages and …
- Can have multiple schemas
- Provides a naming context
  - For example: Internal.Table_1 and External.Table_1 (we can have two table with the same name but in different schemas)

Database Design

Database Design

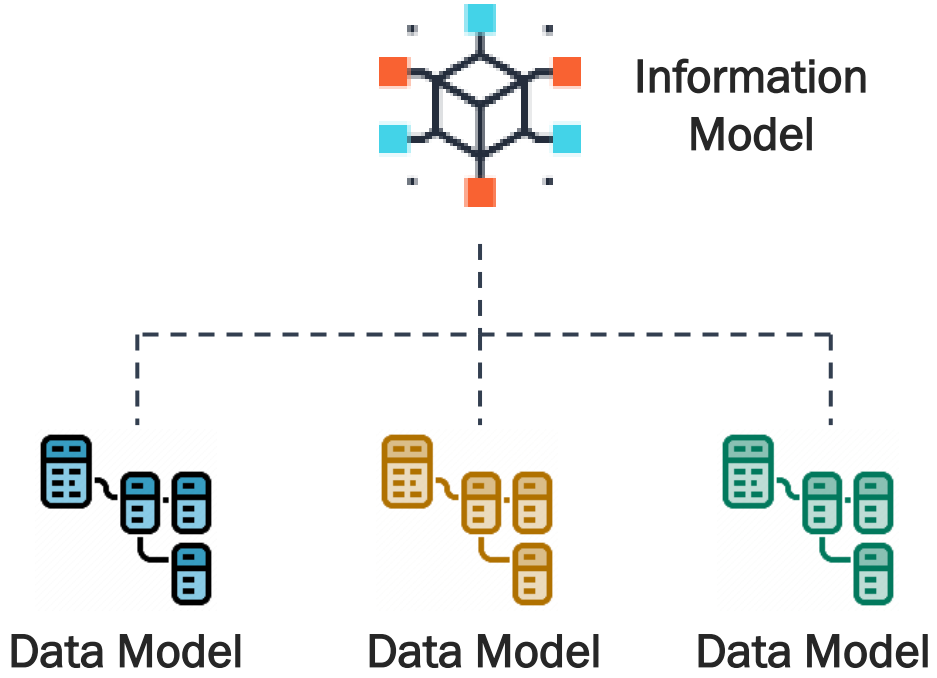


Database Design

- The process of defining database objects and their relationship with each other
- Is the organization of data according to a database model
- The designer determines what data must be stored and how the data elements interrelate
- With this information, they can begin to fit the data to the database model
- Database design involves classifying data and identifying interrelationships

Database Design

Data Models vs Information Models



Abstract formal representation
More at Conceptual level
For designers and operators

More Specific
More at Concrete level
Blueprint of any database system
For implementation

Database Design

Data Models vs Information Models

- Information Models
  - Abstract formal representation of entities that include their properties, their relationships and the operations that could be performed on them.
  - More at a conceptual level and define relationships between objects.
  - An information model provides formalism to the description of a problem domain without constraining how that description is mapped to an actual implementation in software. There may be many mappings of the information model. Such mappings are called data models
- Data Models
  - More at concrete level
  - Are more specific and include more details
  - A blueprint of any database systems

Database Design

Relational Model

Based on a mathematical model

A collection of predicates

Fundamental Assumption:
All data is represented as mathematical relations

Relational Model » Relational Database

Database Design

Relational Model

- Are data models that are based on a mathematical model
- The relational model's central idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values.
- The relational model was the first database model to be described in formal mathematical terms.
- The relational model of data permits the database designer to create a consistent, logical representation of information
- A database organized in terms of the relational model is a relational database.
- The fundamental assumption of the relational model is that all data is represented as mathematical relations
- Is the most used database model for databases
  - Logical data independence
  - Physical data independence
  - Physical storage independence

Database Design

Sets and Relations

If **A** and **B** are sets, a relation **R** is a subset of $\boldsymbol{A \times B}$

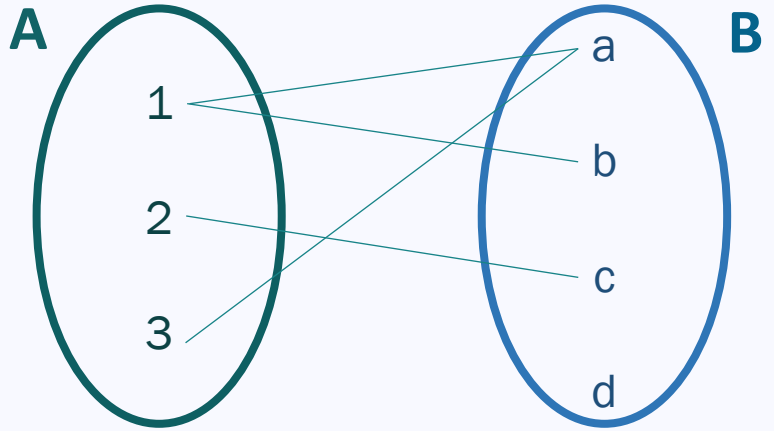
$$R \subset A \times B$$

Example:

$$A = \{1,2,3\}$$

$$B = \{a, b, x, d\}$$

$$R = \{(1, a), (1, b), (2, c), (3, a)\}$$



Database Design

Relation and Relation Schema

Relation (Attribute 1, Attribute 2, Attribute 3, ..., Attribute n)

$R(A_1, A_2, A_3, \dots, A_n)$ — Attributes

$D_i = dom(A_i)$ — Domain

relation / extension / r(R)

$r = \{t_1, t_2, t_3, \dots, t_m\}$ — Tuples

$t_i =< v_1, v_2, v_3, \dots, v_n >$

$v_i \in D_i \cup \{NULL\}$

Primary Key

Students (<u>student_ID</u>,
first_name ,
last_name,
age,
city,
state)

student (100,
Michael,
Johnson,
23,
Dallas,
TX)

$D_1 = dom(student_{ID}) = \{x : x \in INT\}$

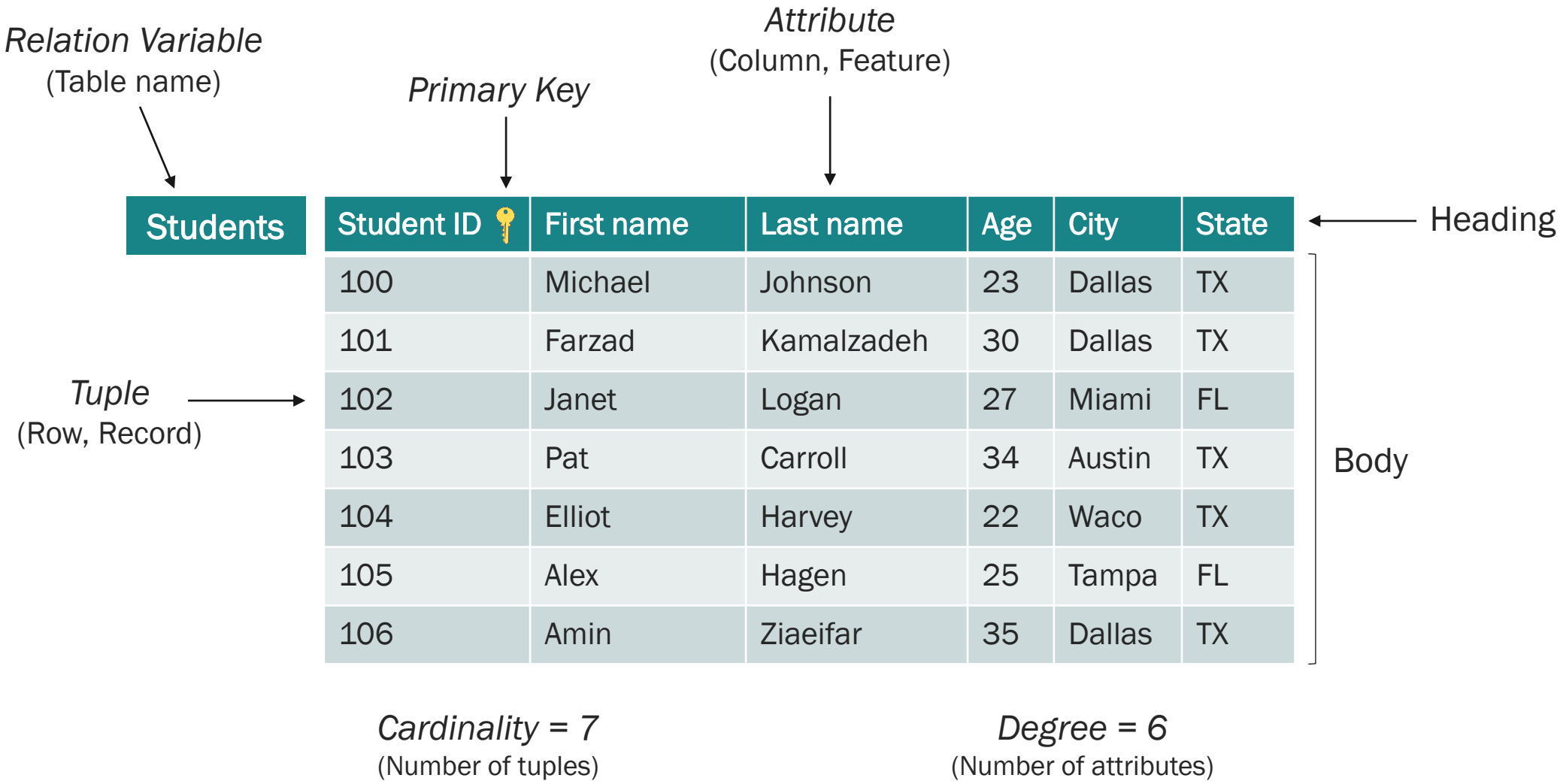
Database Design

Relational Model

- Sets
  - Unordered collection of distinct elements
  - Items of same type
  - No order and no duplicates
- Building blocks:
  - Domain/data type
  - Tuple : an ordered set of attribute values
  - Attribute : an ordered pair of attribute name and type name
- all data is represented in terms of tuples, grouped into relations.
- The relational database is a set of relations
- A relation is defined as a set of n-tuples.
- Relation is mathematical term for table

Database Design

Relation Instance (Table)



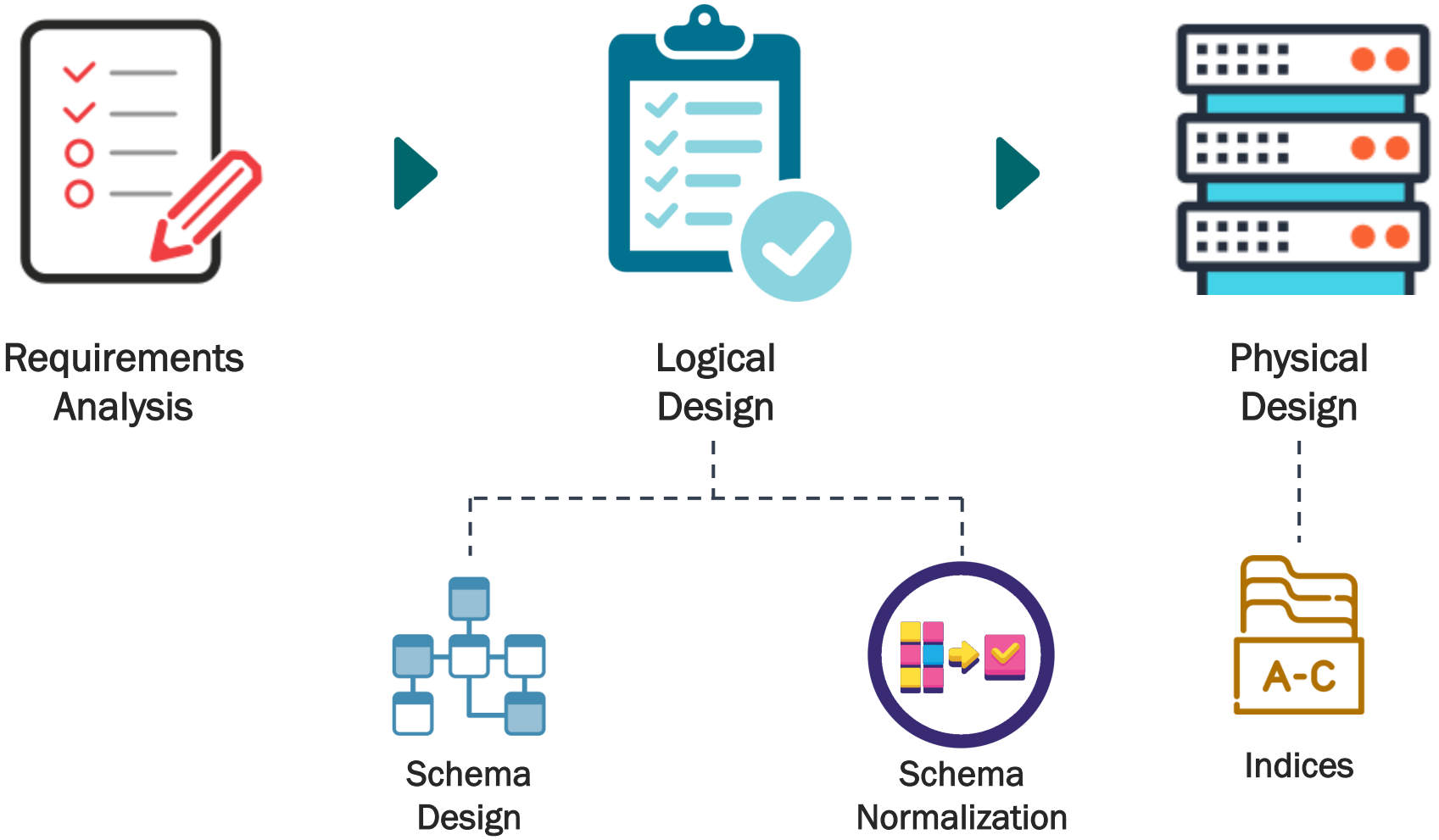
Database Design

Relational Model

- Relation schema
  - Specifies the name of the relation and name and type of the columns
- Relation instance
  - A table made up of rows and columns that follow the schema
- Cardinality
  - The number of tuples
- Degree
  - The number of attributes

Database Design

Design Process



Database Design

Design Process

- Requirement analysis
  - Based on use cases, business process descriptions
- Schema design (conceptual design)
  - Model what the DB is about, e.g. via ER diagrams
- Schema normalization
  - E.g., reduce data redundancy via transformation
- Physical tuning
  - E.g., decide which indices to create or sort order
  - Impact of database management system:
    - Data types
    - Naming rules
    - Indexes
    - constraints

Database Design

Design Process

1. Determine the purpose of the database - This helps prepare for the remaining steps.
2. Find and organize the information required - Gather all of the types of information to record in the database, such as product name and order number.
3. Divide the information into tables - Divide information items into major entities or subjects, such as Products or Orders. Each subject then becomes a table.
4. Turn information items into columns - Decide what information needs to be stored in each table. Each item becomes a field, and is displayed as a column in the table. For example, an Employees table might include fields such as Last Name and Hire Date.
5. Specify primary keys - Choose each table's primary key. The primary key is a column, or a set of columns, that is used to uniquely identify each row. An example might be Product ID or Order ID.
6. Set up the table relationships - Look at each table and decide how the data in one table is related to the data in other tables. Add fields to tables or create new tables to clarify the relationships, as necessary.
7. Refine the design - Analyze the design for errors. Create tables and add a few records of sample data. Check if results come from the tables as expected. Make adjustments to the design, as needed.
8. Apply the normalization rules - Apply the data normalization rules to see if tables are structured correctly. Make adjustments to the tables, as needed.

Schema Design

Schema Design

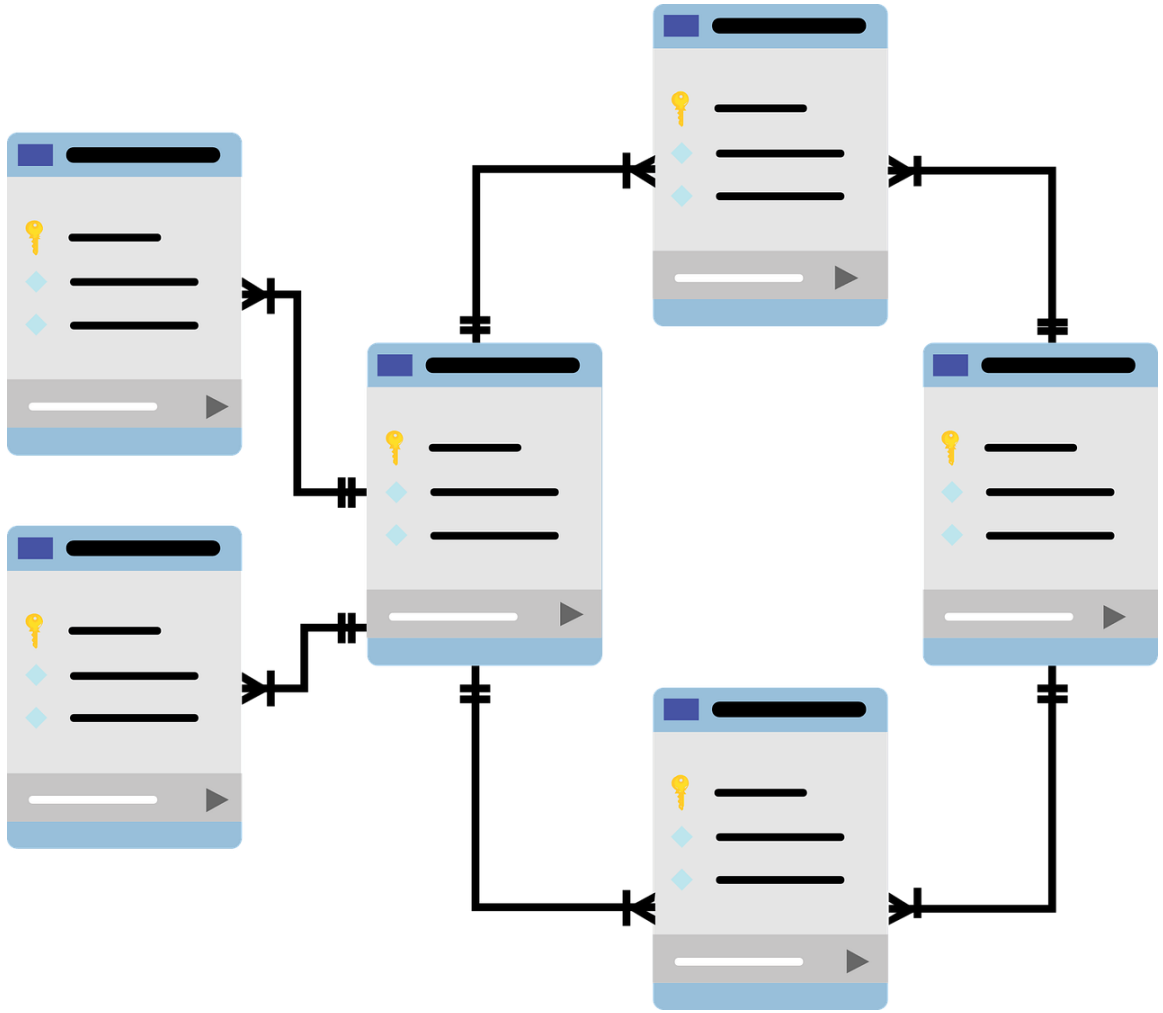
Entity-Relationship Model

A tool used to design Relational Databases

Entities

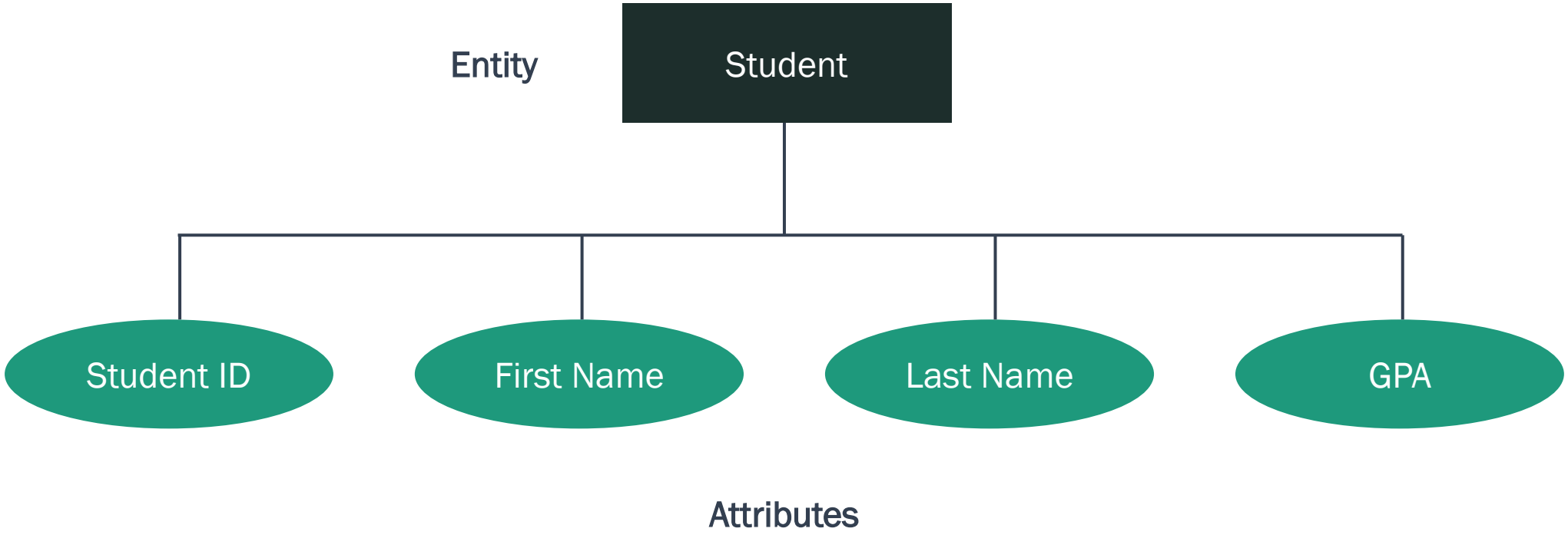
Attributes

Relation ships



Schema Design

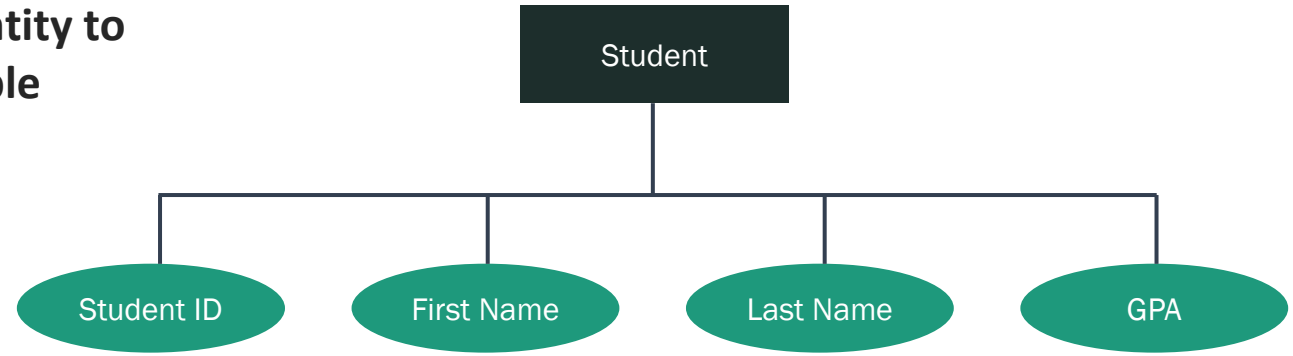
Entity



Schema Design

Entity

Mapping an Entity to a Database table



Student ID	First name	Last name	GPA
100	Michael	Johnson	4
101	Farzad	Kamalzadeh	3.5
102	Janet	Logan	2.7
103	Pat	Carroll	3.4
104	Elliot	Harvey	2.2
105	Alex	Hagen	2.5
106	Amin	Ziaeifar	3.5

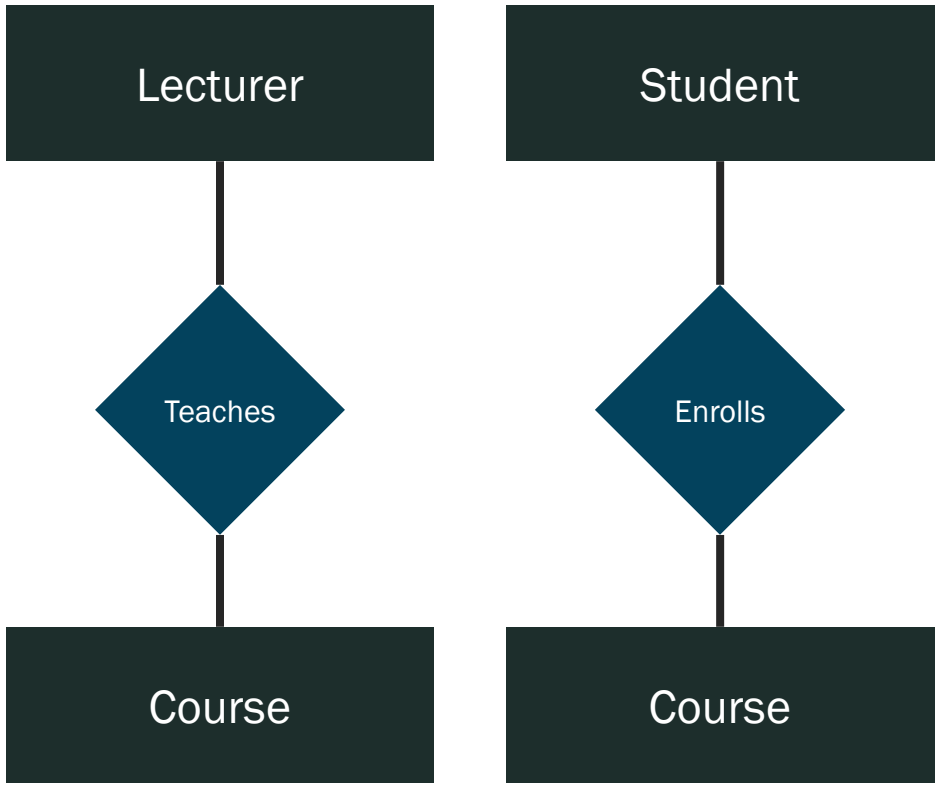
Schema Design

Entity

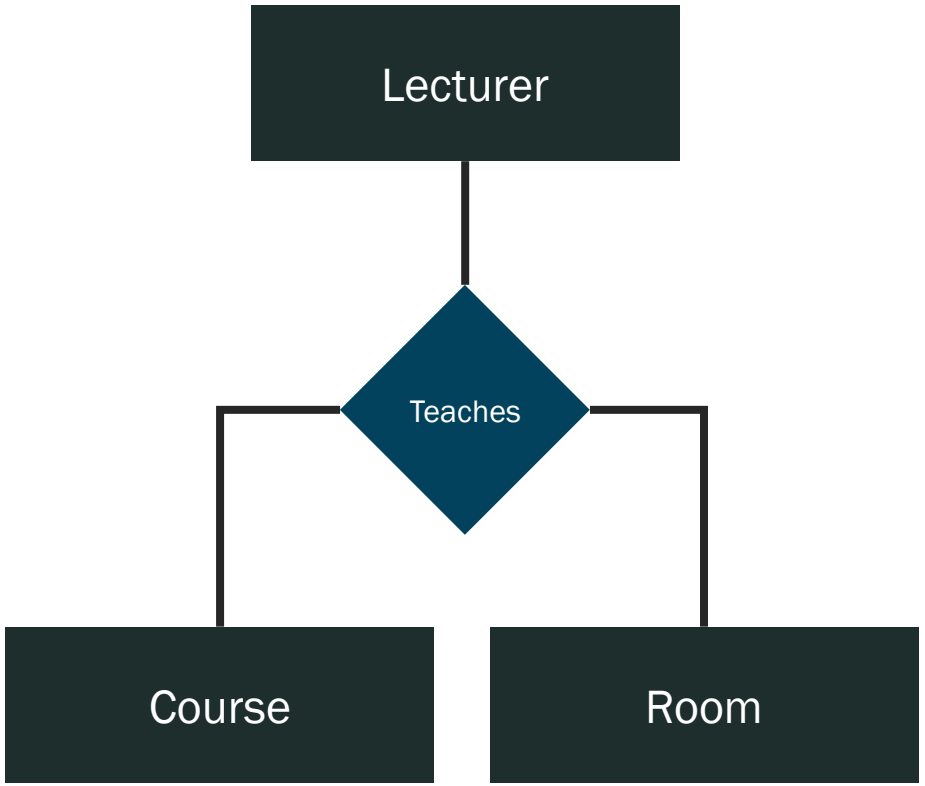
- Objects like a noun (person, place, thing) that exists independently of other entities in database
- Entity set: multiple entities of same type
  - Represented as rectangle in ER diagram
- Attribute: a property connected to an entity set that characterizes the entity
  - Represented as oval in ER diagram
  - Connected via lines to associated entity
  - Underlined if (part of) a key attribute
  - Attributes have simple values (e.g., integer)

Schema Design

Relationships



Binary Relationship



Ternary Relationship

Schema Design

Relationships

- A relationship connects entities
- Relationships are represented as diamonds
- Connecting lines indicate targeted entities
- May connect two or more entities

Schema Design

Classification of Relationships: Multiplicity

For each X in E, how many Y's are there in F?

Multiplicity = Participation + Cardinality

(at least)
(minimum)

(at most)
(maximum)

Notation:

— or ▬
Zero One

— or →
Many One

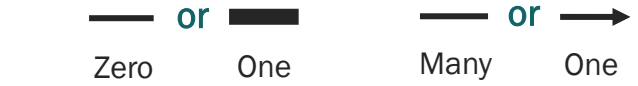
Schema Design

Classification of Relationships: Multiplicity

- Can constrain number of relationships per entity
- Cardinality: Denotes the maximum number of possible relationship occurrences in which a certain entity can participate in (in simple terms: at most).
- Participation: Denotes if all or only some entity occurrences participate in a relationship (in simple terms: at least).
- Multiplicity = Cardinality + Participation
  - A specification of the number of possible occurrences of a property or the number of allowable elements that may participate in a given relationship
- Types:
  - One-to-one
  - One-to-many
  - Many-to-one
  - Many-to-many
- In simple words: for each X, how many Y's are there?
- Common Notation types:
  - UML
  - Chen
  - Crowsfoot
- Participation constraint: entity must relate at least once
  - Represented by a thick line (entity to relationship)
- At-most-one constraint: entity relates at most once
  - Represented by arrow (from entity to relationship)

Schema Design

Classification of Relationships: Multiplicity



From E to F: Each member of **E** can be connected by **R** to any number of members of **F** — Many-Many

From F to E: Each member of **F** can be connected by **R** to any number of members of **E** — Many-Many

From E to F: Each member of **E** can be connected by **R** to at most one member of **F** — Many-One

From F to E: Each member of **F** can be connected by **R** to any number of members of **E** — One-Many

From E to F: Each member of **E** can be connected by **R** to any number of members of **F** — One-Many

From F to E: Each member of **F** can be connected by **R** to at most one member of **E** — Many-One

From E to F: Each member of **E** can be connected by **R** to at most one member of **F** — Many-One

From F to E: Each member of **F** can be connected by **R** to at most one member of **E** — One-One

Schema Design

Classification of Relationships: Multiplicity

UML Notation

Minimum	Maximum
Mandatory : 1 Optional : 0	Forbidden : 0 Only One : 1 Many : *

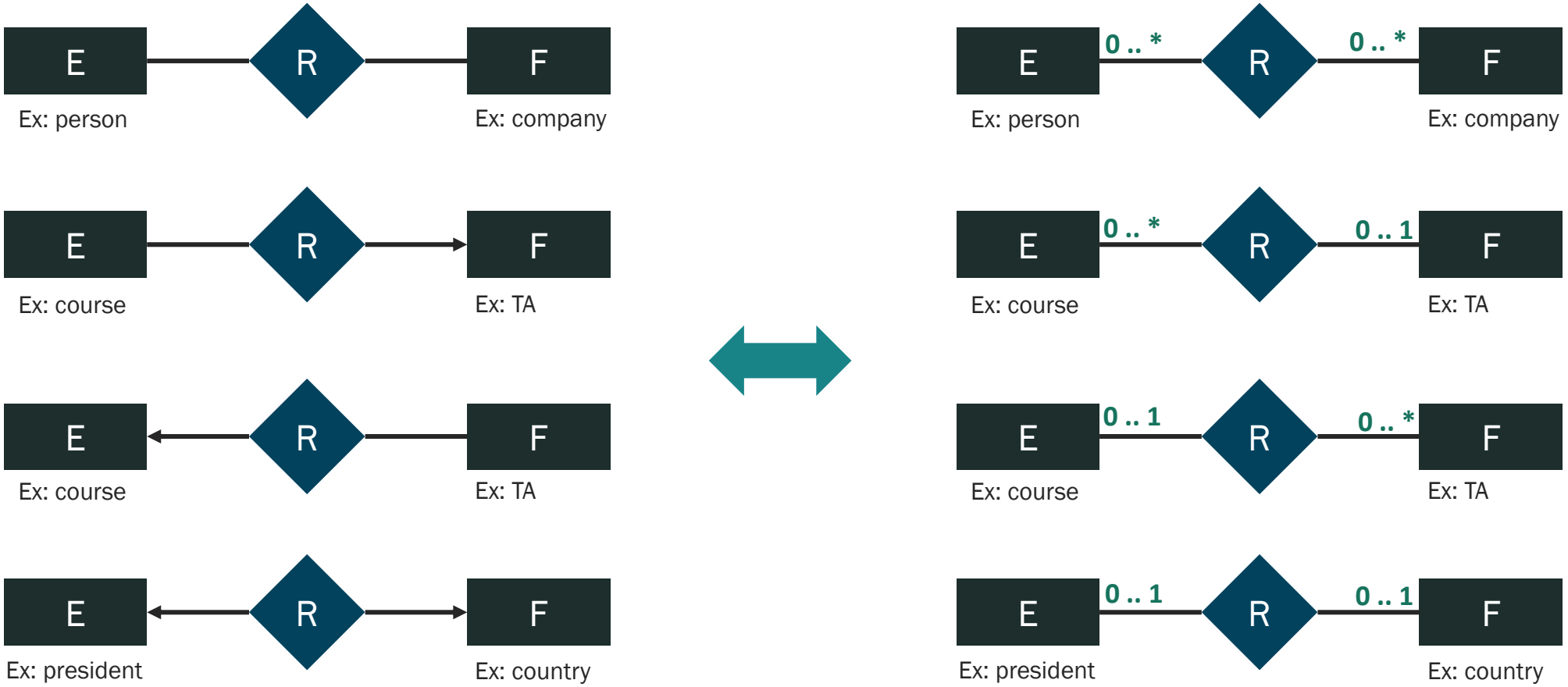
Schema Design

Classification of Relationships: Multiplicity

- UML Notation
- Minimum:
  - Mandatory: 1
  - Optional: 0
- Maximum:
  - Forbidden: 0
  - Only one: 1
  - Many: *
- Represented as min..max pairs: 0..1, 0..*, 1..1, 1..*

Schema Design

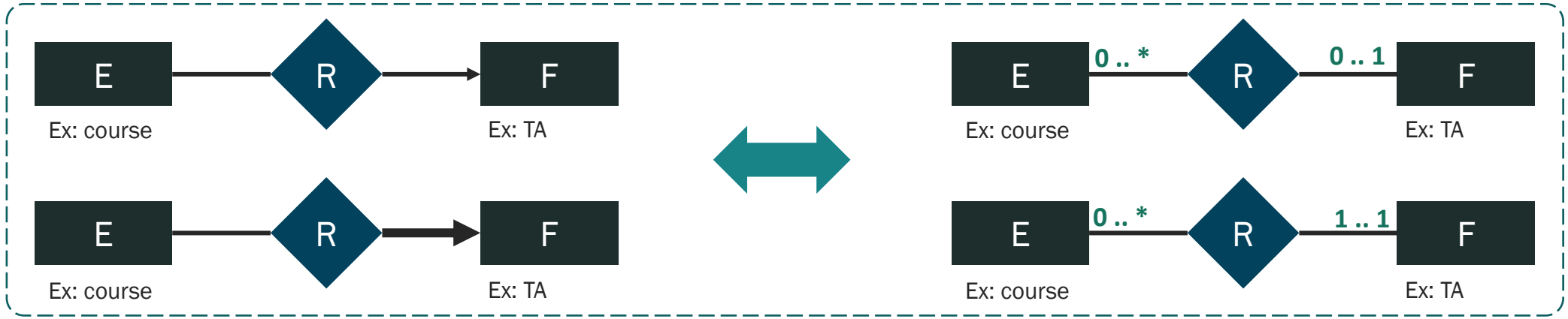
Classification of Relationships: Multiplicity

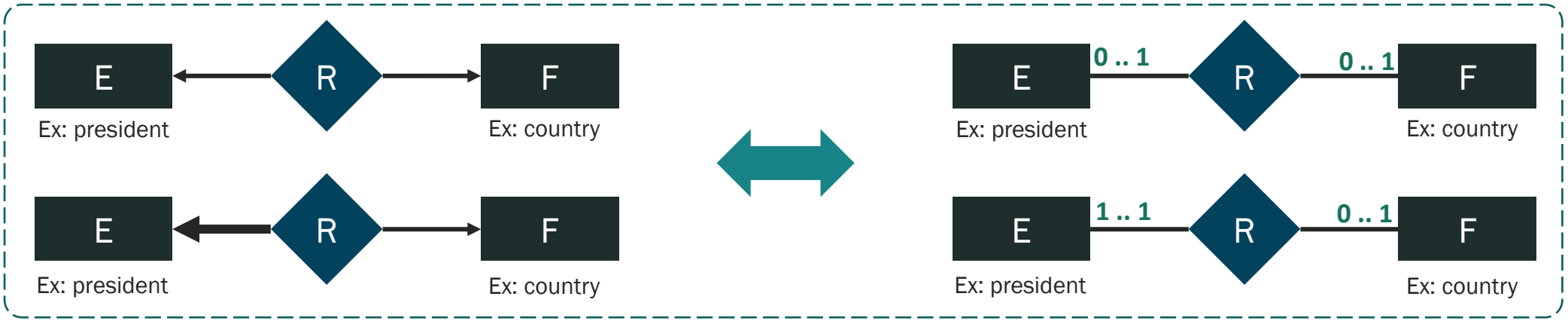


Schema Design

Classification of Relationships: Multiplicity

Optional vs Mandatory

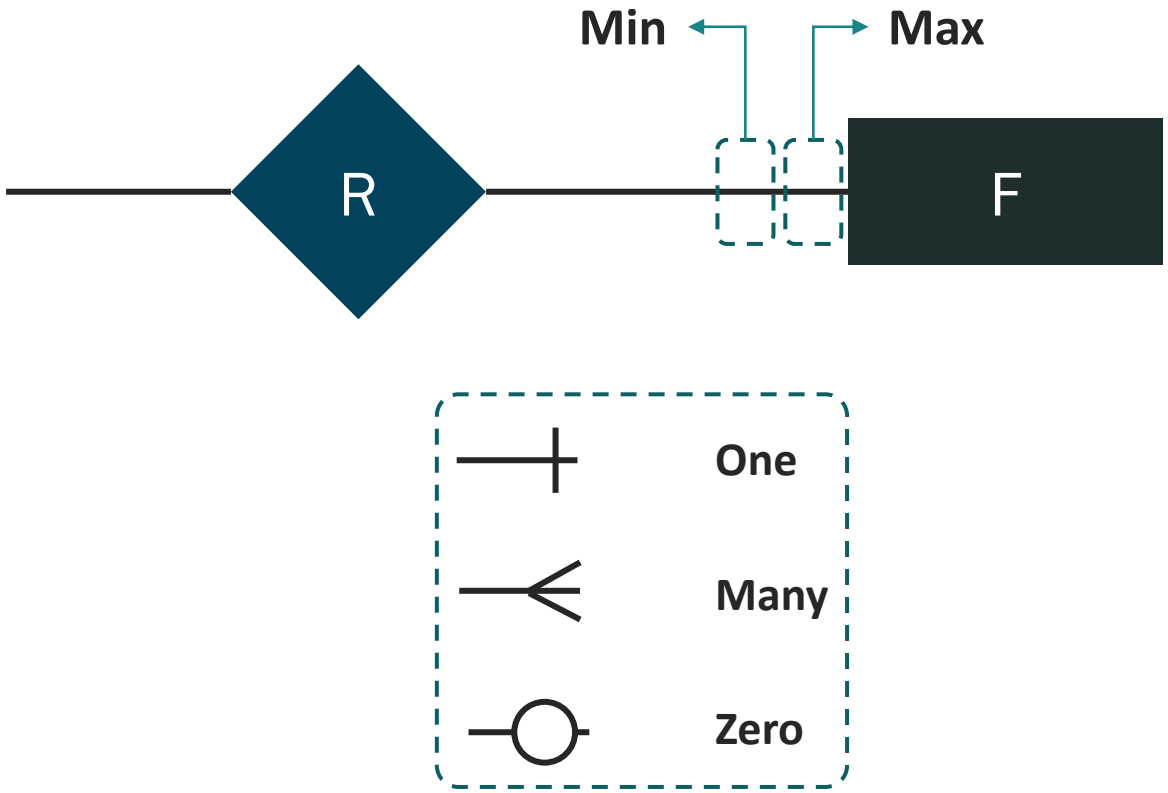




Schema Design

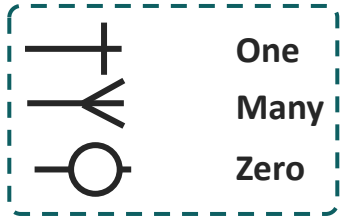
Classification of Relationships: Multiplicity

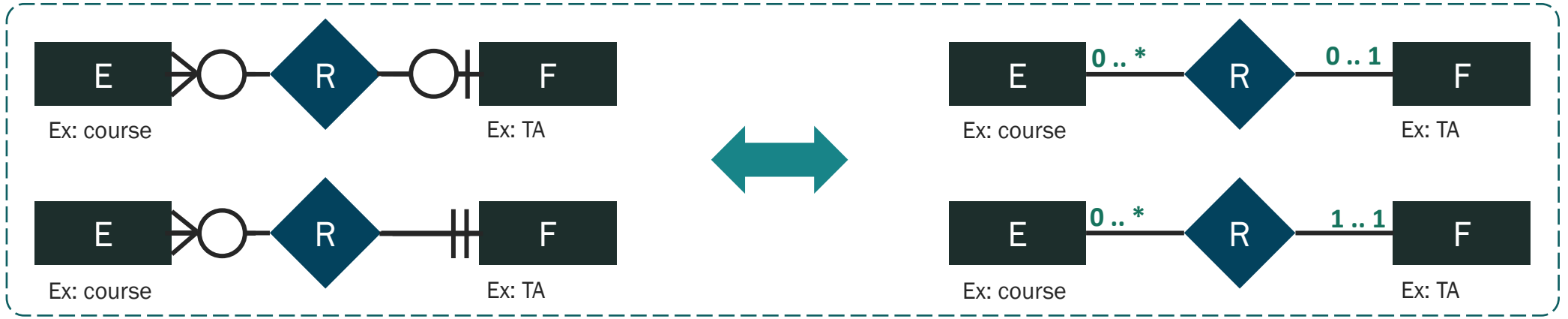
- Crowsfoot Notation

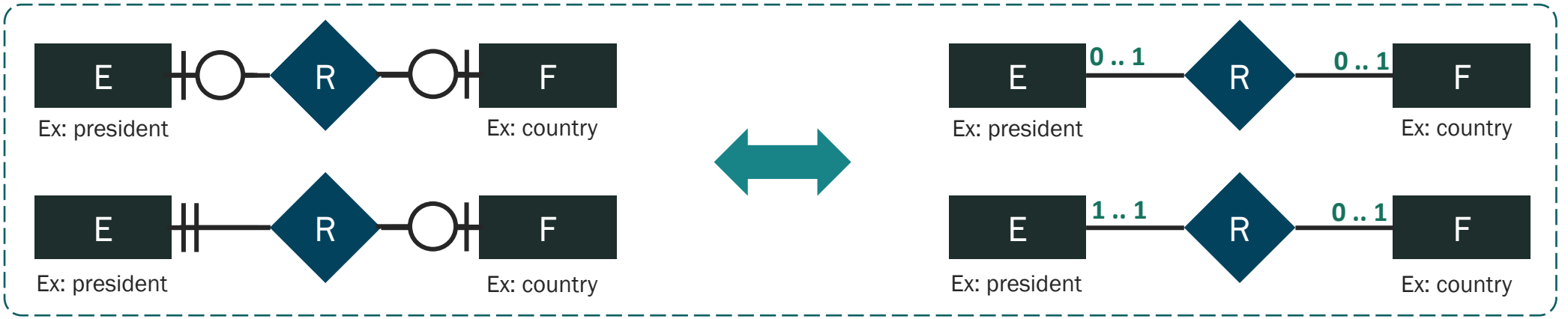


Schema Design

Classification of Relationships: Multiplicity







Schema Design

Exercise

Bank Example

Customers
Bank Accounts

- Each customer must have at least one bank account to be on file

Bank Example

Customers
Bank Accounts

- Each customer can have exactly one bank account on file
- Bank accounts are owned by only one customer

Online Shop Example

Customers
Orders

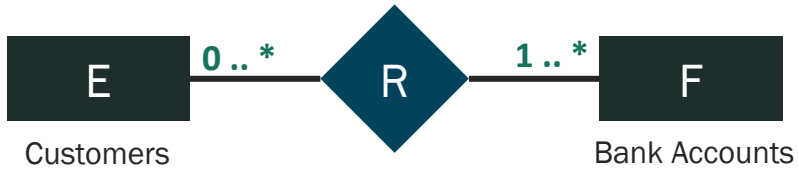
- Each customer is on file if they have at least one order
- Orders cannot be placed by more than one customer
- There are no orders without customers

Schema Design

Answer

Bank Example

Customers
Bank Accounts



Bank Example

Customers
Bank Accounts

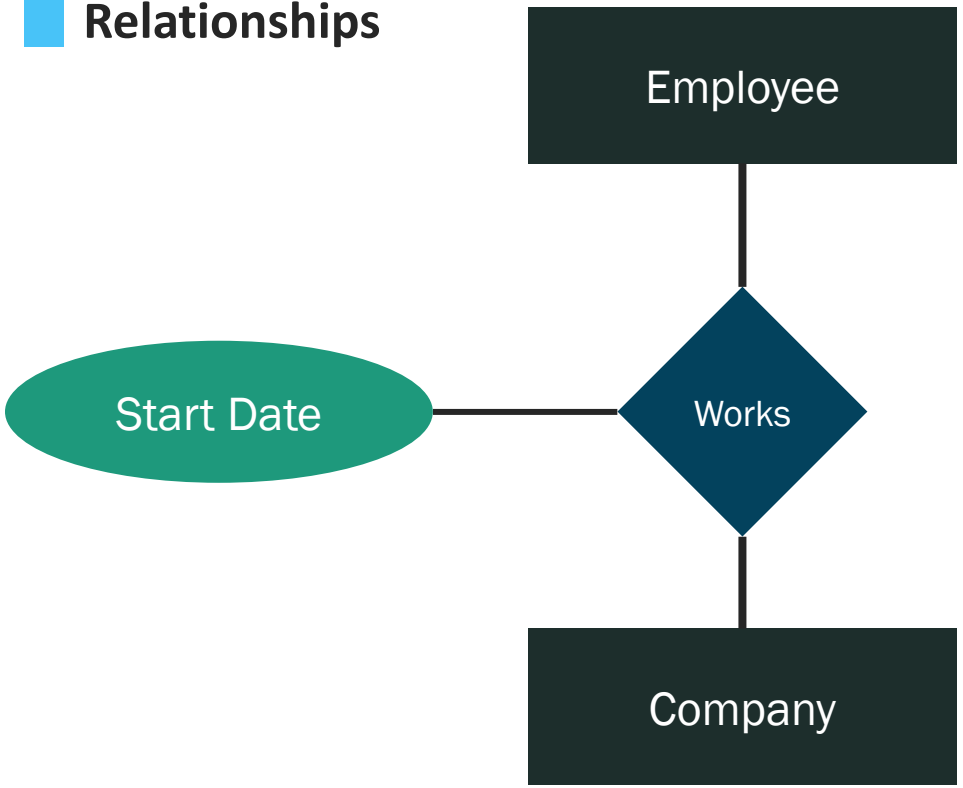
Online Shop Example

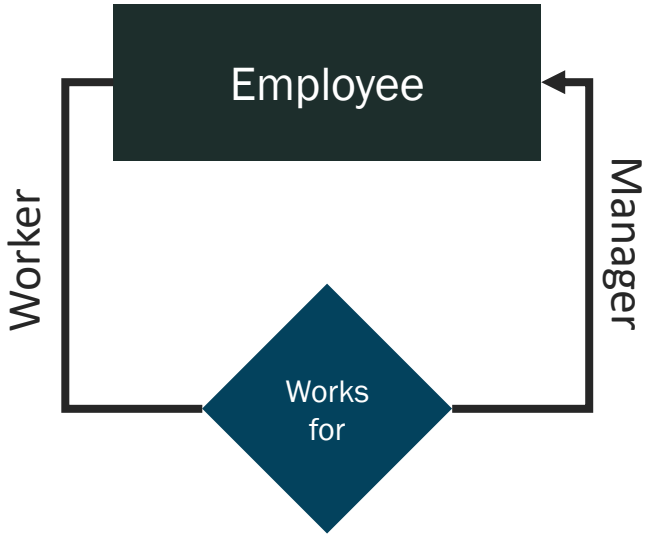
Customers
Orders

Schema Design

More E/R Diagram Features

Attributes of Relationships





Roles of Entities

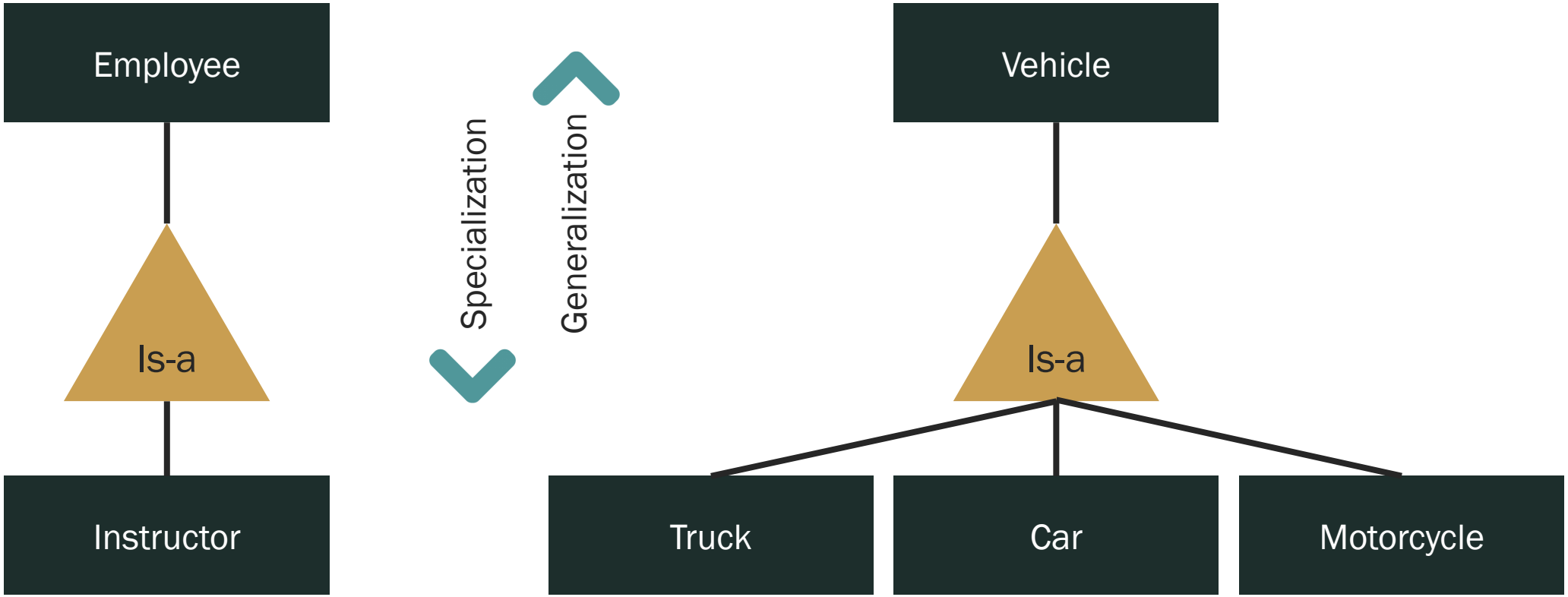
Schema Design

More E/R Diagram Features

- Can associate relationships with attributes
  - Same representation as for entity attributes
  - Refers to related entity combinations
- Can assign entities to roles
  - The function that an entity plays in a relationship is called its role. Roles are normally explicit and not specified.
  - Represent role as label for connecting edge
  - They are useful when the meaning of a relationship set needs clarification.
    - Required when connecting entities of same type

Schema Design

Sub-Classes



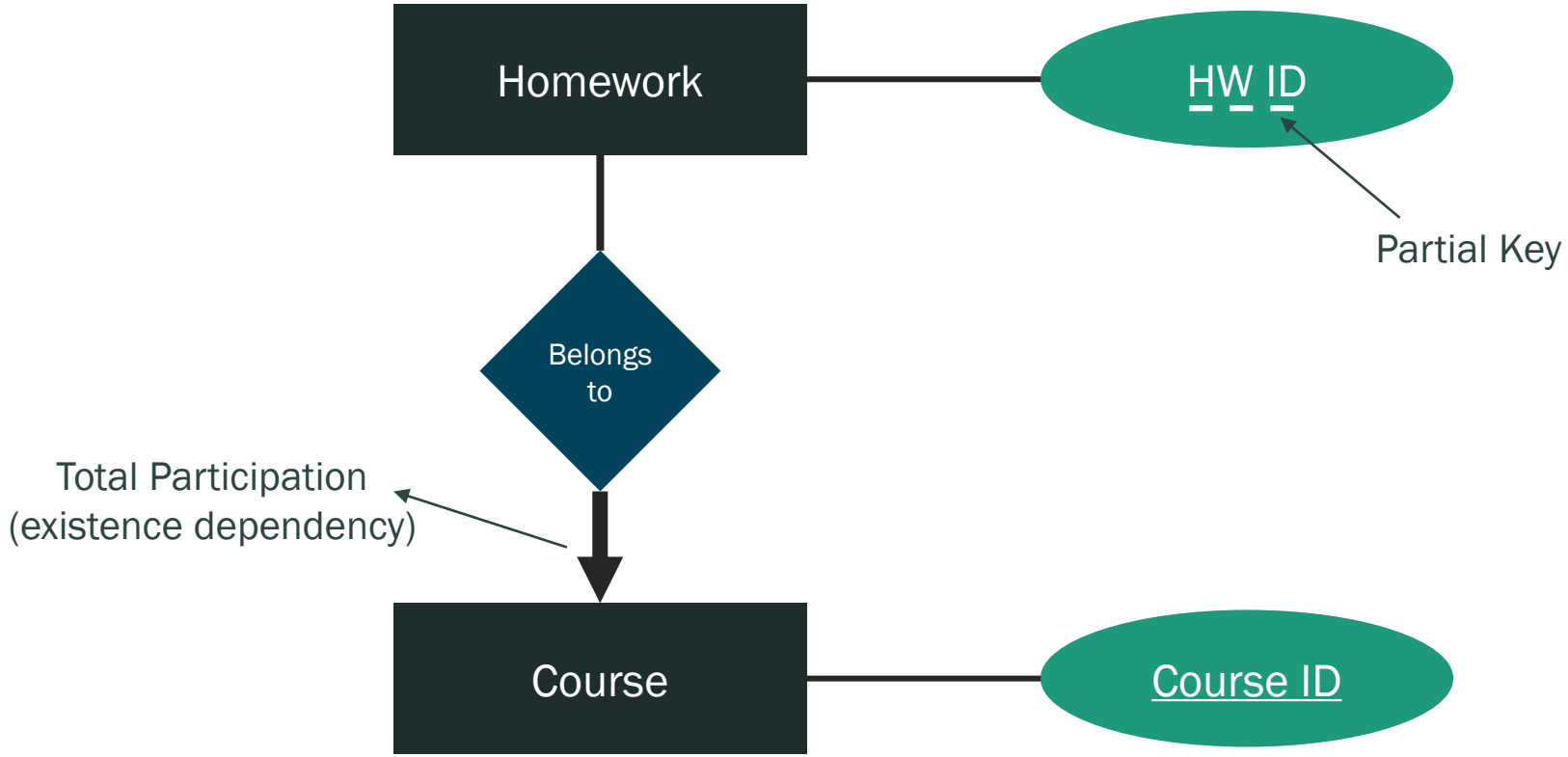
Schema Design

Sub-Classes

- You most likely know concept from OO languages
- Sub-classing allows to reduce redundancy in diagram
- Sub-classes inherit the attributes from parent
- Sub-classes inherit relationships from parent
- Represent sub-classes via triangles ("Is-A")

Schema Design

Weak Entities



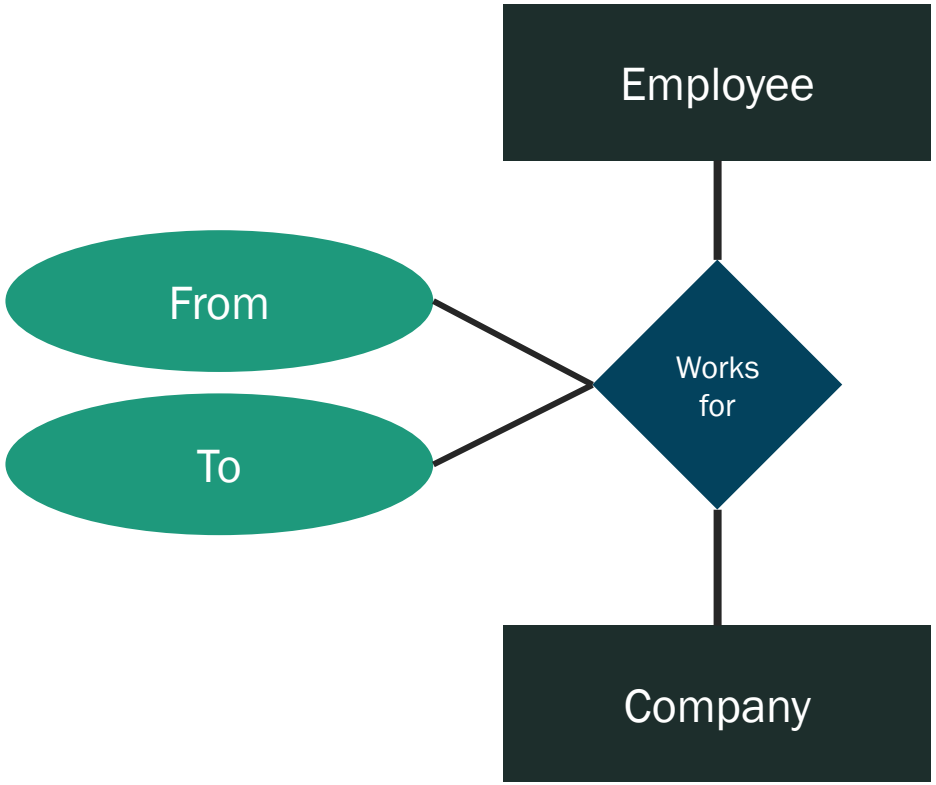
Schema Design

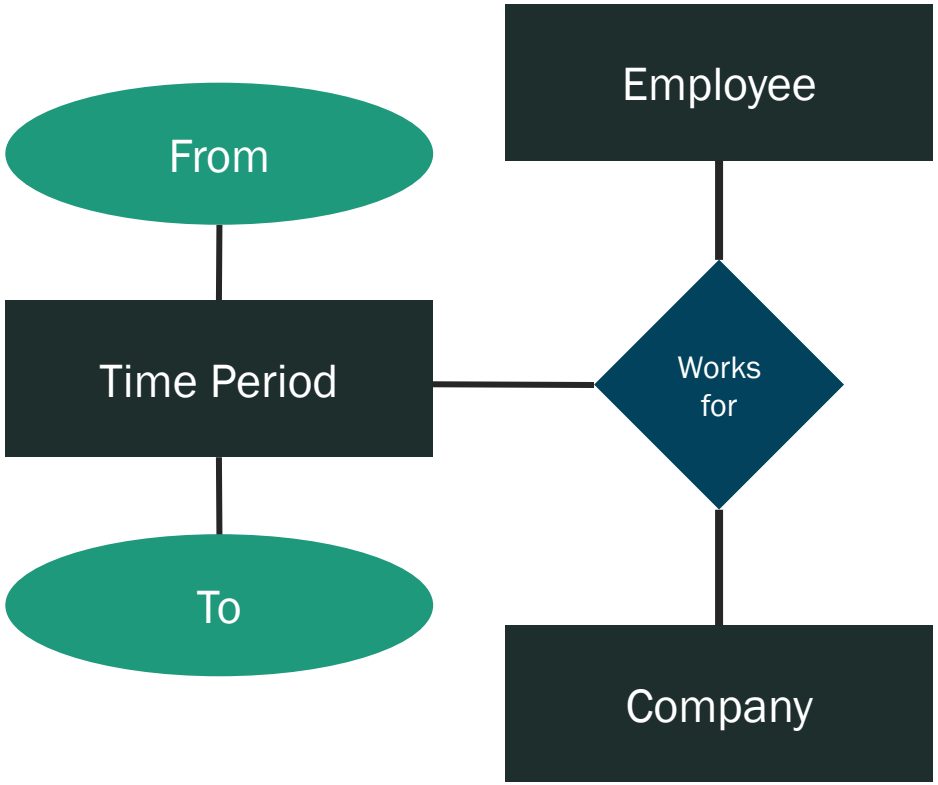
Weak Entities

- Weak entity can only be uniquely identified if its "owner" entity exists
- Weak entity connects to owner via identifying relationship
- Weak entity must participate in identifying relationship
- Also, each weak entity can appear at most once in it
- Weak entity types have partial keys.
  - Partial Keys are set of attributes with the help of which the tuples of the weak entities can be distinguished and identified

Schema Design

Design Choices: Entities vs Attributes





Schema Design

Design Choices: Entities vs Attributes

- Often can choose between entities and attributes
  - E.g., model address as attribute or connected entity?
- Use entity if employees can have multiple addresses
  - Attribute values cannot be set valued
- Model as entity if we want to structure address further
- Can model components as attributes

Schema Design

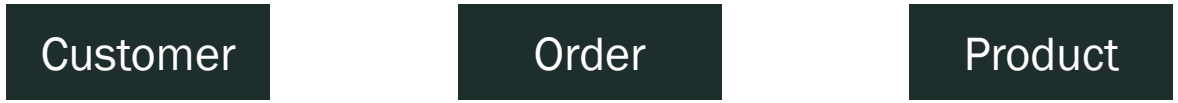
Exercise

- Draw an ER Diagram for the following situation:
- Customers buying from an online shop:
  - Each customer has a unique ID, name, email address
- Customers can place orders
  - Each order is associated with exactly one customer
  - Orders have unique IDs and consist of at least one product/item
- Items/products have ID, name, price

Schema Design

Answer

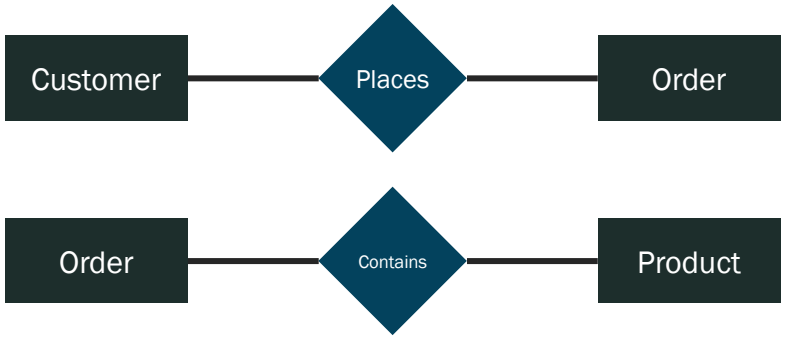
- **Entities:**



- **Attributes:**

What is Primary Key for each Entity?

- **Relationships:**



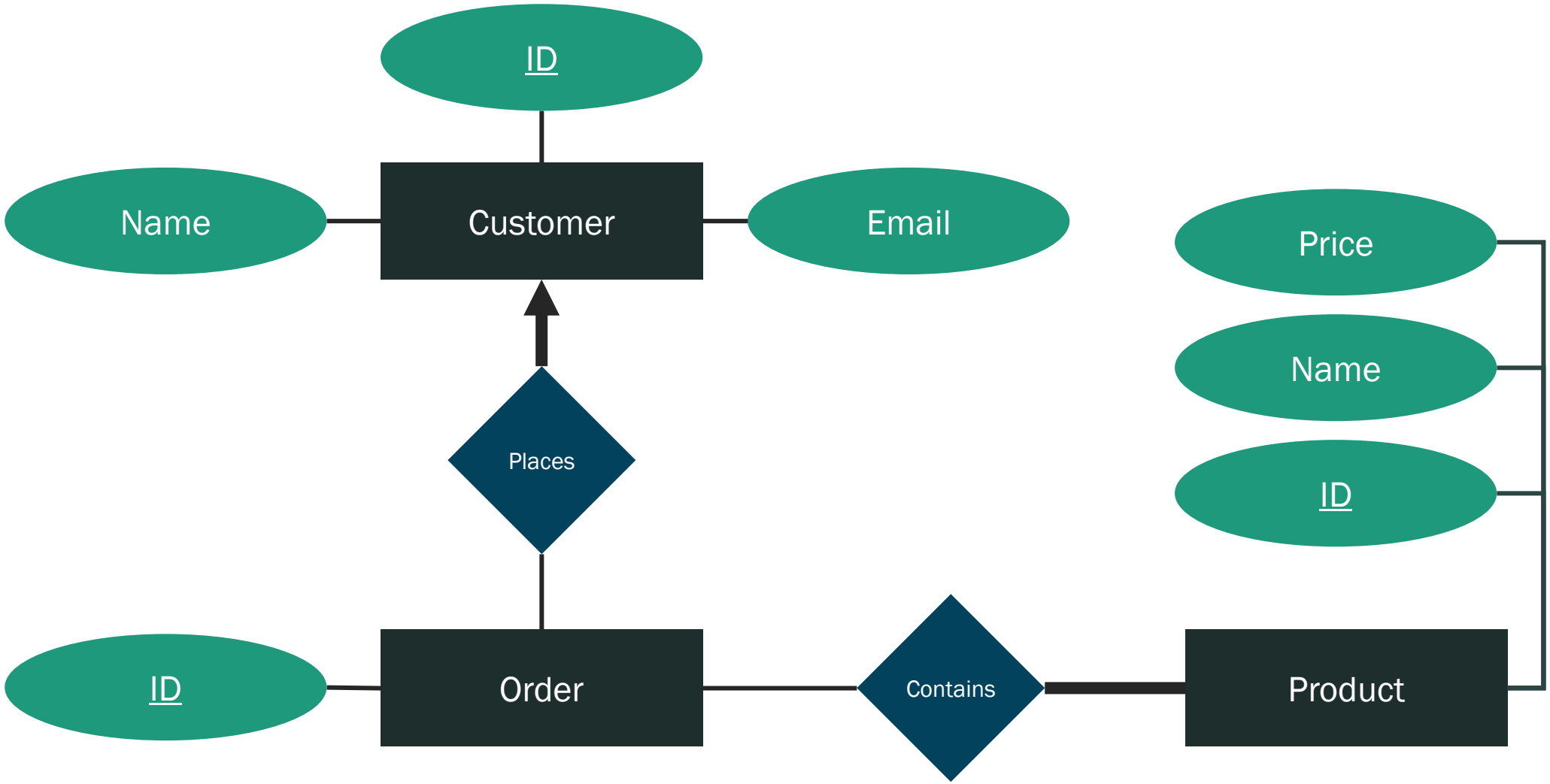
- **Constraints:**

Each order is associated with exactly one customer

Orders consist of at least one product/item

Schema Design

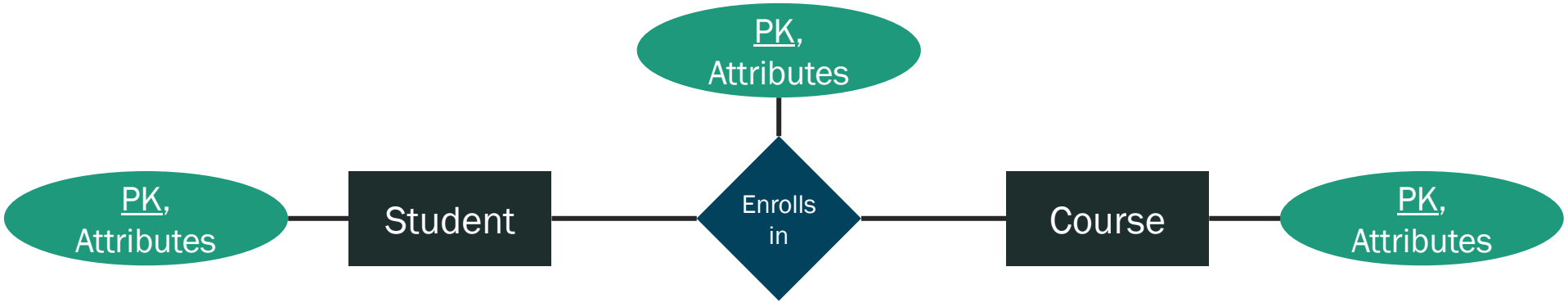
Answer



Schema Design

ERDs as Relations

Translating ERDs to Relations



Students (<u>student_ID</u>, first_name , last_name, age)

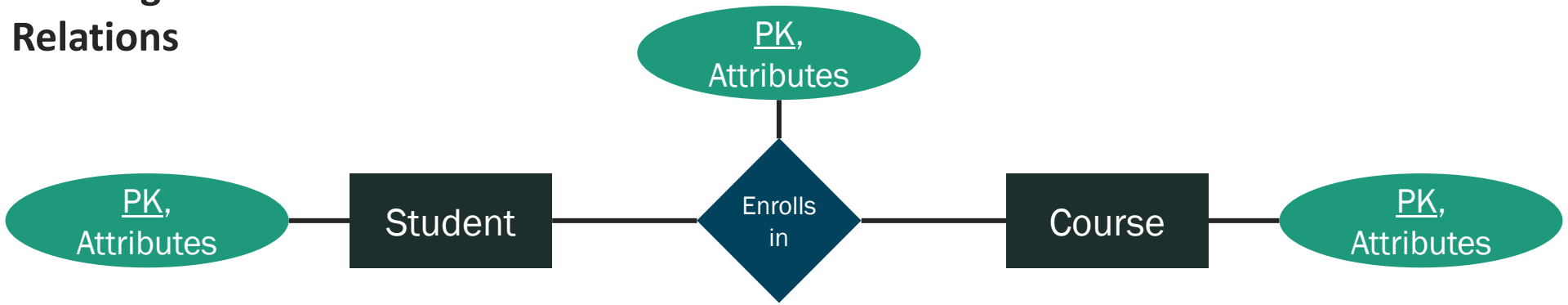
Enrollments (<u>student_ID</u>, <u>course_ID</u>, semester, room)

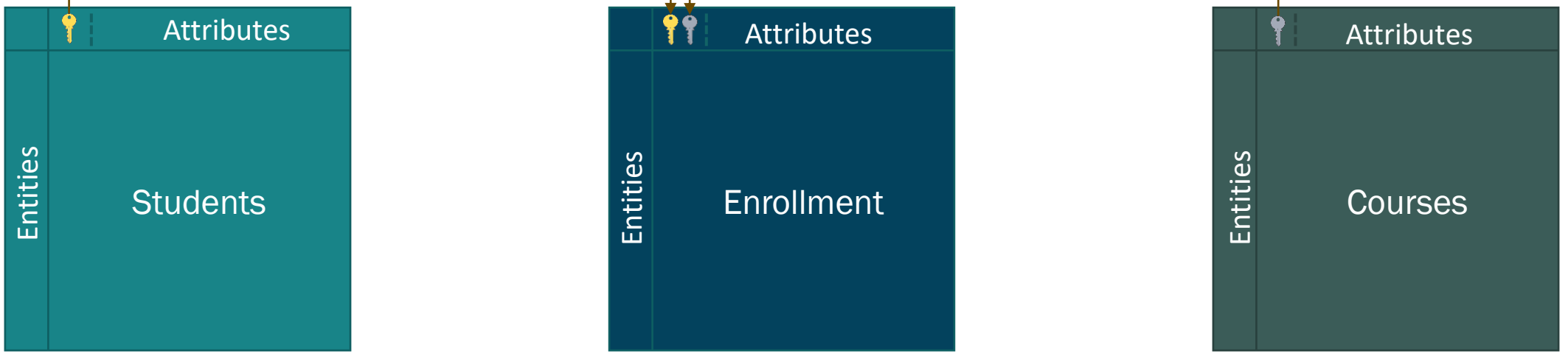
Courses (<u>course_ID</u>, course_name, credit_hours, level)

Schema Design

ERDs as Relations

Translating ERDs to Relations





Schema Design

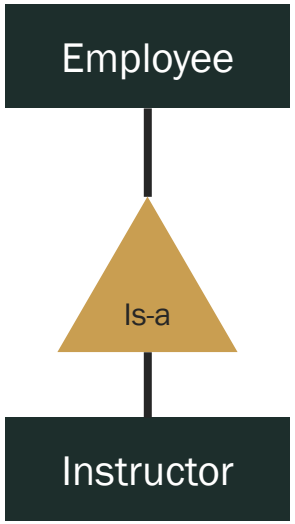
ERDs as Relations

- Need to translate ER diagrams to relations
- Introduce relations for entity types (translating entities)
  - Each entity becomes row in relation
  - Properties are represented as columns
  - Underlined attributes are part of primary key
- Generic method: introduce relation capturing relationships (translating relationships)
  - Columns store primary keys of all connected entities
  - Row represents relationship between specific entities
  - Primary key combines primary keys of entities
  - Additional attributes become columns as well

Schema Design

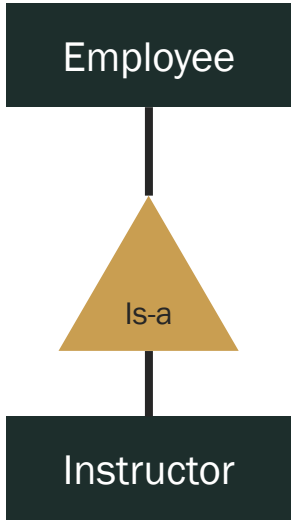
ERDs as Relations

Translating Sub-Classes



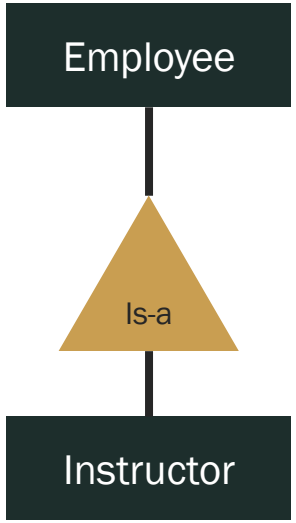
Employee (<u>employee_ID</u>,
name,
job_title)

Instructor (<u>employee_ID</u>,
name,
job_title,
department)



Employee (<u>employee_ID</u>,
name,
job_title)

Instructor (<u>employee_ID</u>,
department)



EmployeeOrInstructor
(<u>employee_ID</u>,
name,
job_title,
department)

Schema Design

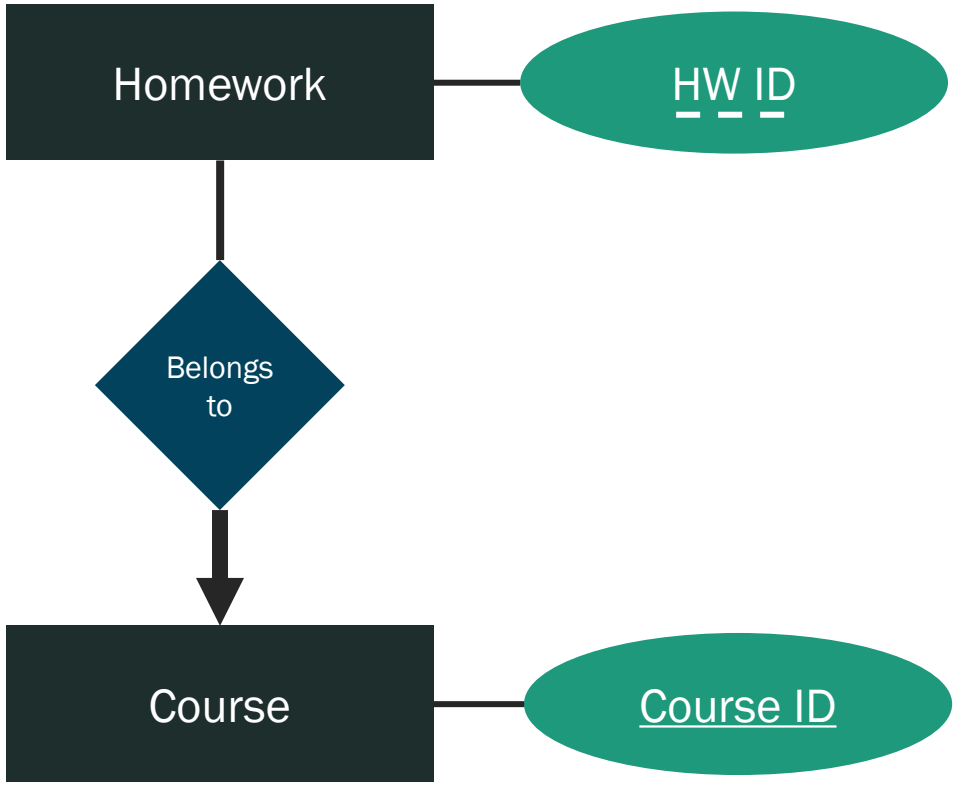
ERDs as Relations

- Translating sub-classes
- Entities of sub-class may have additional attributes
- Can be represented in multiple different ways
  - Separate relations for superclass and sub-class
  - Introduce multiple relations linking key to attributes
  - Use relation for sub-class, set unused attributes to null

Schema Design

ERDs as Relations

Translating Weak Entities



Homework (homework_No,
course_ID,
grade)
Foreign key (course_ID)
references course
on delete cascade

Schema Design

ERDs as Relations

- Translating Weak Entities
- Introduce new relation for storing weak entities
- Add foreign key columns referencing owner entity
- In SQL: cascading delete depending on owner

Schema Design

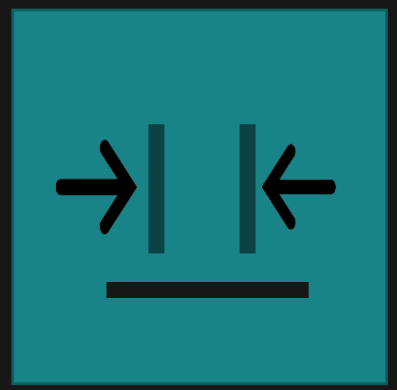
Modeling of Constraints

Classification of Constraints

Key Constraints

Single-Value Constraints

Referential Constraints



Domain Constraints

General Constraints

Schema Design

Modeling of Constraints

- Keys are attributes or sets of attributes that uniquely identify an entity within its entity set. No two entities may agree in their values for all of the attributes that constitute a key. It is permissible, however, for two entities to agree on some, but not all, of the key attributes.
  - Every entity set must have a key.
  - A key can consist of more than one attribute.
  - There can also be more than one possible key for an entity set. However, it is customary to pick one key as the "primary key," and to act as if that were the only key.
  - In our E/R diagram notation, we underline the attributes belonging to a key for an entity set.
- Example:
  - SSN uniquely identifies a person

Schema Design

Modeling of Constraints

- Single-value constraints are requirements that the value in a certain context be unique. Keys are a major source of single-value constraints, since they require that each entity in an entity set has unique value(s) for the key attribute(s). However, there are other sources of single-value constraints, such as many-one relationships.
  - A relationship R that is many-one from entity set E to entity set F implies a single-value constraint. That is, for each entity e in E, there is at most one associated entity f in F.
- Example:
  - A country can have only one president

Schema Design

Modeling of Constraints

- Referential integrity constraints are requirements that a value referred to by some object actually exists in the database. Referential integrity is analogous to a prohibition against dangling pointers, or other kinds of dangling references, in conventional programs.
- There are several ways this constraint could be enforced.
  - We could forbid the deletion of a referenced entity.
  - We could require that if a referenced entity is deleted, then all entities that reference it are deleted as well.
- Represented by a rounded arrow in ER Diagram
- Example:
  - If you work for A company, it must exist in the database

Schema Design

Modeling of Constraints

- Domain constraints require that the value of an attribute must be drawn from a specific set of values or lie within a specific range.
- Example:
  - Age is between 0 and 150
- General constraints are arbitrary assertions that are required to hold in the database. For example, we might wish to require that no more than ten stars be listed for any one movie.
- Example:
  - At most 80 students enroll in a class

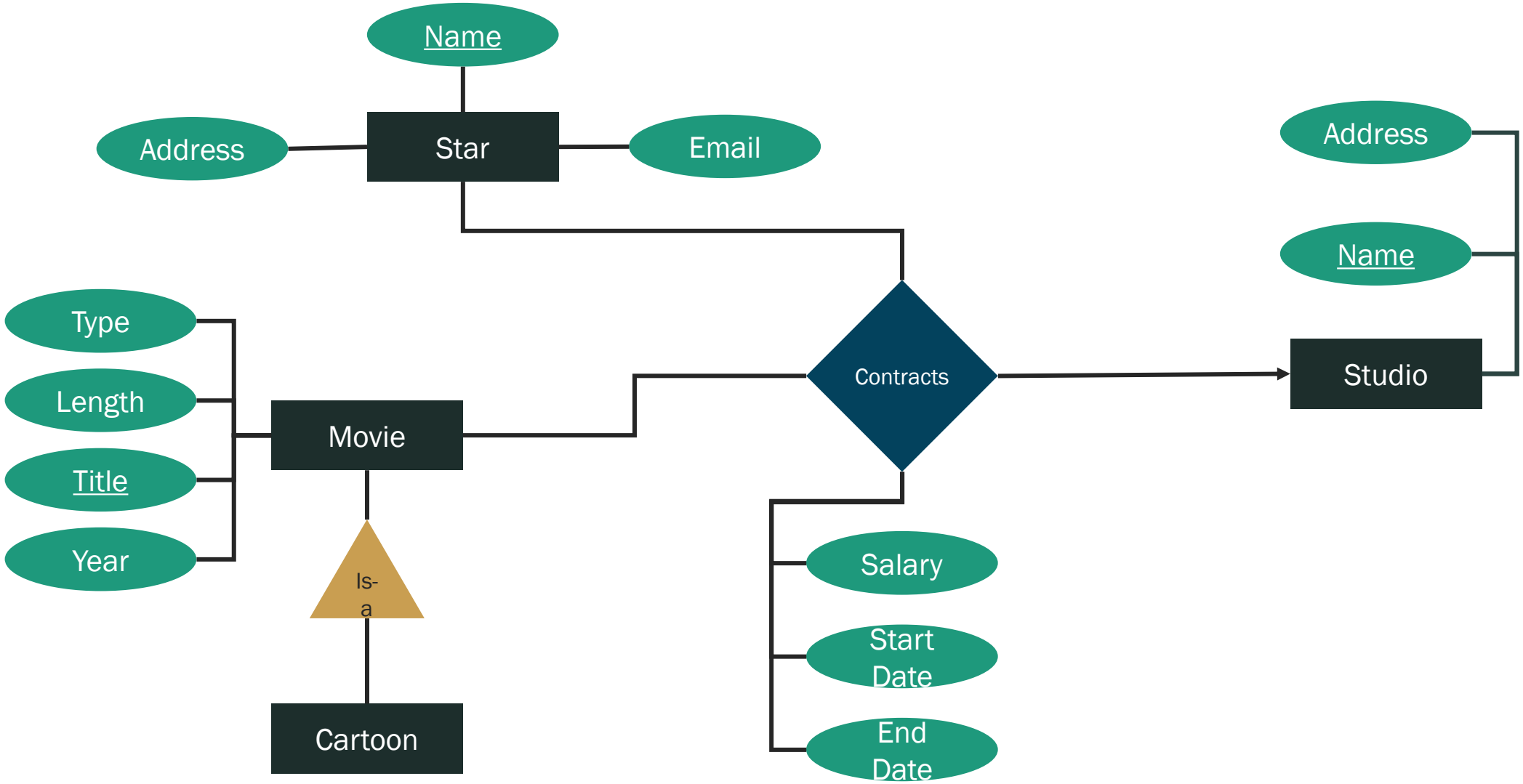
Schema Design

Exercise

- Draw an ER Diagram for the following situation:
- A star acts in movies
  - A star has a name, email and address
- Movies are owned by only one studio
  - Movies have title, length, type, and the year created
  - Movies can have sequels
- Studios have name and address
- Studios hire stars to act in their movies by contract
  - Contracts specify the salary of the star and start and end date of the contract
- Cartoons are a specific kind of movies where stars act only as a voice-actor

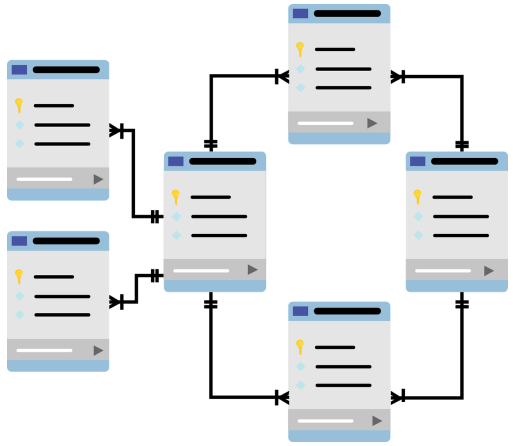
Schema Design

Answer



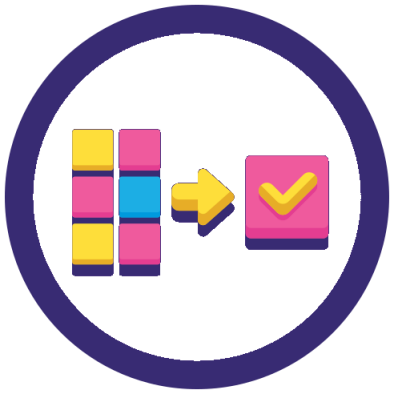
Schema Normalization

Schema Normalization



First Sketch

Most probably sub-optimal
With data redundancy



Normalization

By Normalization, we reduce data redundancy and anomalies and optimize the schema

Schema Normalization

- We can prepare first sketch of database schema via ER
- Resulting schema will most likely be sub-optimal
  - I.e., the schema implies lots of data redundancy
- Data redundancy leads to various problems
- We optimize initial schema via schema normalization

Schema Normalization

Anomalies

- Problems that arise with a flawed schema

Redundancy	Insertion Anomalies	Update Anomalies	Deletion Anomalies
Information may be repeated unnecessarily	Cannot add information without adding other information	changing information in one tuple but leaving the same information unchanged in another	If a set of values becomes empty, we may lose other information as a side effect

The accepted way to eliminate these anomalies is to normalize relations by decomposing them

Schema Normalization

Anomalies & Functional Dependency

Name	Position Type	Salary
Max	Full Time	10000
Farzad	Part Time	5000
Alex	Full Time	10000
Mary	Full Time	10000
David	Part Time	5000
Robert	Full Time	10000
Sam	Part Time	5000

With Dependency

Name	Position Type
Max	Full Time
Farzad	Part Time
Alex	Full Time
Mary	Full Time
David	Part Time
Robert	Full Time
Sam	Part Time

Position Type	Salary
Full Time	10000
Part Time	5000

Dependency Removed

Schema Normalization

Functional Dependencies

- Used to detect data redundancies
- Values in some columns uniquely decide values in others
- Notation: X ⟶ Y means values in X decide values in Y
- Problems
  - Update anomaly: could make TA salaries inconsistent
  - Insertion anomaly: could lack salary info for new hours
  - Deletion anomaly: could lose salary info after deletions
- Solution Analysis
  - We removed redundancy by decomposing table
    - FD does not connect columns in same table
    - Prior anomalies cannot happen anymore
  - Must avoid data loss via decomposition
    - Can reconstruct based on FD (Hours ⟶ Salary)
    - Recompose by looking up Salary for Hours value
  - Decomposing table may require additional joins!

Schema Normalization

Functional Dependency

Functional Dependency on a Relation R

$$FD\ on\ R \quad : \quad A_1A_2 \dots A_n \rightarrow B$$

Determinant (LHS)
A set of attributes

Dependent (RHS)
A set of attributes

$X \rightarrow Y$

If two tuples of R agree on attributes $A_1A_2 \dots A_n$,they must also agree on attribute B.

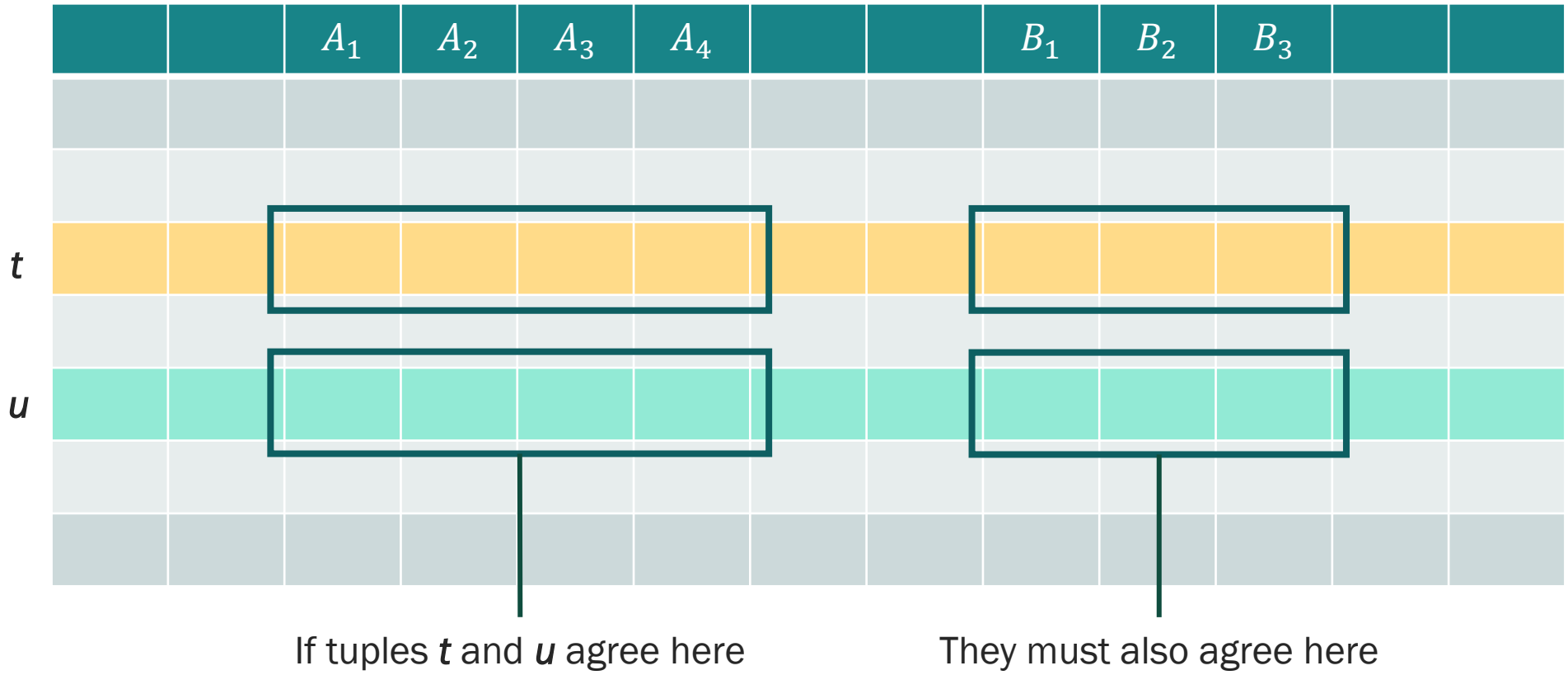
$$FD_1: \quad A_1A_2 \dots A_n \rightarrow B_1$$
$$FD_2: \quad A_1A_2 \dots A_n \rightarrow B_2$$
$$\vdots$$
$$FD_m: \quad A_1A_2 \dots A_n \rightarrow B_m$$

$$A_1A_2 \dots A_n \rightarrow B_1B_2 \dots B_m$$

If a set of attributes, functionally determines more than one attribute (a set of FDs)

Schema Normalization

Functional Dependency



Schema Normalization

Functional Dependency and Redundancy

- FDs state that values in X determine values in Y
- Redundant storage of Y if X stored multiple times
  - Sufficient to store Y once for each X value
- Want to design schema to avoid this in each case
  - Considering all possible future database states
- Want to avoid storing X and Y in same table, except …
- What is the only FD that we allow in database?

Schema Normalization

Keys of Relations

$$A_1 A_2 \dots A_n \rightarrow B_1 B_2 \dots B_m$$

Condition 1

A_1	A_2	...	A_n	B_1	B_2	B_3	B_4	...	...	B_m

$\{A_1, A_2, \dots, A_n\}$ is the Key of this Relation

A Key should be minimal

Condition 2

Schema Normalization

Keys of Relations

Student ID	First Name	Last Name	Major	GPA
125	Janet	Logan	EE	3.7
423	Janet	Carroll	CS	3.4
854	Farzad	Kamalzadeh	OR	3.6
239	Alex	Hagen	CE	3.9
371	Janet	Logan	EE	3.8

First Name	Last Name	Father's Name	Major	GPA
Janet	Logan	Max	EE	3.7
Janet	Carroll	Sam	CS	3.4
Farzad	Kamalzadeh	Reza	OR	3.6
Alex	Hagen	David	CE	3.9
Janet	Logan	Sam	EE	3.8

Schema Normalization

Keys of Relations

- We say a set of one or more attributes $\{A_1, A_2, \dots, A_n\}$ is a key for a relation R, if:
  - Those attributes functionally determine all other attributes of the relation. That is, because relations are sets it is impossible for two distinct tuples of R to agree on all of $A_1, A_2, \dots, A_n$.
  - No proper subset of $\{A_1, A_2, \dots, A_n\}$ functionally determines all other attributes of R. That means, a key must be minimal.
  - When a key consists of a single attribute we often say that A (rather than $\{A\}$) is a key.

Schema Normalization

Decomposition

Student ID	First Name	Last Name	Major	GPA	Course ID	Course Name	Department	Grade	Semester
125	Janet	Logan	EE	3.7	6532	Python	IS	A	Fall 2021
423	Jane	Carroll	CS	3.4	6458	Databases	MS	B	Spring 2022
854	Farzad	Kamalzadeh	OR	3.6	3467	Data Mining	CS	A-	Fall 2021
239	Alex	Hagen	CE	3.9	7524	Big Data	IS	B+	Fall 2021
371	Janet	Sanchez	EE	3.8	4582	ML	CS	A	Spring 2022
239	Alex	Hagen	CE	3.9	6532	Python	IS	B	Fall 2021
854	Farzad	Kamalzadeh	OR	3.6	6532	Python	IS	A-	Fall 2021
423	Jane	Carroll	CS	3.4	7524	Big Data	IS	B+	Fall 2021

Schema Normalization

Decomposition

Student ID	First Name	Last Name	Major	GPA
125	Janet	Logan	EE	3.7
423	Jane	Carroll	CS	3.4
854	Farzad	Kamalzadeh	OR	3.6
239	Alex	Hagen	CE	3.9
371	Janet	Sanchez	EE	3.8

Course ID	Course Name	Department
6532	Python	IS
6458	Databases	MS
3467	Data Mining	CS
7524	Big Data	IS
4582	ML	CS

Student ID	Course ID	Semester	Grade
125	6532	Fall 2021	A
423	6458	Spring 2022	B
854	3467	Fall 2021	A-
239	7524	Fall 2021	B+
371	4582	Spring 2022	A
239	6532	Fall 2021	B
854	6532	Fall 2021	A-
423	7524	Fall 2021	B+

Schema Normalization

Normalization Steps

1. Identifying FDs of the Schema
2. Determining whether FDs and the Schema meet the Normal Forms
3. Decomposing the Schema to meet the Normal Forms

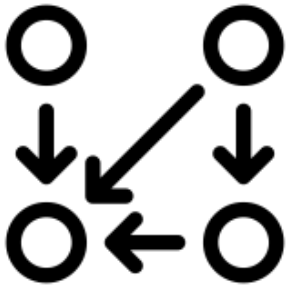
Schema Normalization

Finding FDs

Finding FDs by looking at data

Valid sources for mining FDs:

Domain knowledge



Inferring new FDs from given FDs

Schema Normalization

Finding FDs

- Inferring new FDs from given FDs

$F_1 \models F_2$

$\boldsymbol{F_1}$ and $\boldsymbol{F_2}$ are sets of FDs

FD $\boldsymbol{F_1}$ implies FD $\boldsymbol{F_2}$

No Relation can satisfy $\boldsymbol{F_1}$ without satisfying $\boldsymbol{F_2}$

Schema Normalization

Finding FDs

- Armstrong's Axioms

Reflexivity	If $Y \subseteq X$ then $X \rightarrow Y$ is implied
Augmentation	If $X \rightarrow Y$ then $XZ \rightarrow YZ$ for any Z
Transitivity	If $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Schema Normalization

Trivial Functional Dependencies

Trivial FD

$A_1A_2 \ldots A_n \rightarrow B$ is trivial if:

$$B \subset \{A_1, A_2, \ldots, A_n\}$$

Example: {Course, Department} → {Department}

Schema Normalization

Finding FDs

- Common mistake: try finding FDs by looking at data
- Data only captures current state of the database
- Not all functional dependencies may appear
- Data may suggest misleading "pseudo FDs"
- Two valid sources for mining FDs:
- Domain knowledge
- Inferring new FDs from given FDs
  - Notation F1 |= F2 means FDs F1 imply FDs F2
    - No relation can satisfy F1 without satisfying F2
  - Can infer all FDs by applying Armstrong's Axioms:
    - Reflexivity: if $Y \subseteq X$ then $X \rightarrow Y$ is implied
    - Augmentation: if $X \rightarrow Y$ then $XZ \rightarrow YZ$ for any Z
    - Transitivity: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

Schema Normalization

FD Closure and Cover

Closure of F

Set of all implied FDs

$$F^+ = \{f: F \models f\}$$

- Can be calculated using Armstrong's Axioms
- Can be extremely large

Cover for G

F is a cover for G if:

$$F^+ = G^+$$

Schema Normalization

Finding FDs

- Closure of a set of FDs are all implied FDs
- F+ = {f | F |= f}
- Can be calculated using Armstrong's axioms
- F is a cover for G if F+ = G+
- The closure can be extremely large

Schema Normalization

Inferring FDs: Example

Given

F={
{Course} → {Lecturer},
{Course} → {Department},
{Lecturer, Department} → {Office}
}

Inferred

{Course, Department} → {Department}

{Course, Lecturer} → {Department, Lecturer}

{Course} → {Office}

How?

{Course} → {Course, Lecturer}

{Course, Lecturer} → {Department, Lecturer}

{Course} → {Department, Lecturer}

{Course} → {Office}

Schema Normalization

Attribute Closure

Closure of X — set of all attributes that can be determined given a set of attributes X and a set of FDs F

A	Set of all attributes of a relation
X	Set of attributes for which Closure is being calculated
F	Set of given FDs

$\downarrow$

X^+	Set of all the attributes that can be determined by X

Schema Normalization

Attribute Closure: Calculation

Calculation Steps

Step 1 X^+ is initialized as X

Step 2 Search for some FD: $B_1 B_2 \dots B_m \rightarrow C$ such that all of $B_1, B_2, \dots, B_m$ exist in X^+
Add C to X^+

Step 3 Repeat step 2 until no more attributes can be added to X^+

X^+ Is the attribute closure of X

Schema Normalization

Attribute Closure: Calculation

- Example

A $\quad A = \{A_1, A_2, A_3, A_4, A_5, A_6\}$

F $\quad F = \{$
$\{A_1, A_2\} \rightarrow \{A_3\},$
$\{A_2, A_3\} \rightarrow \{A_1, A_4\},$
$\{A_4\} \rightarrow \{A_5\},$
$\{A_3, A_6\} \rightarrow \{A_2\}$
$\}$

X $\quad X = \{A_1, A_2\}$

X^+

Schema Normalization

Attribute Closure: Calculation

- Answer

Step 1 X^+ is initialized as X → $X^+ = \{A_1, A_2\}$

Step 2 $\{A_1, A_2\} \rightarrow \{A_3\}$

$X^+ = \{A_1, A_2\}$ → A_3 is added to X^+ → $X^+ = \{A_1, A_2, A_3\}$

Step 2 $\{A_2, A_3\} \rightarrow \{A_1, A_4\}$

$X^+ = \{A_1, A_2, A_3\}$ → A_4 is added to X^+ → $X^+ = \{A_1, A_2, A_3, A_4\}$

Step 2 $\{A_4\} \rightarrow \{A_5\}$

$X^+ = \{A_1, A_2, A_3, A_4\}$ → A_5 is added to X^+ → $X^+ = \{A_1, A_2, A_3, A_4, A_5\}$

Step 3 No more attributes can be added to X^+ → $X^+ = \{A_1, A_2, A_3, A_4, A_5\}$

Schema Normalization

Keys of a Relation

A	Set of all attributes of a relation
K	Any given set of attributes
F	Set of given FDs

K is a key of the relation if $K^+ = A$

Schema Normalization

Exercise

A $A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$

F $F = \{$
$\{A_1, A_2\} \rightarrow \{A_5\}$,
$\{A_2, A_6\} \rightarrow \{A_3, A_4\}$,
$\{A_5\} \rightarrow \{A_6\}$,
$\{A_3, A_6\} \rightarrow \{A_7\}$
$\}$

X $X = \{A_1, A_2\}$

X^+

Schema Normalization

Answer

- Answer

Step 1 X^+ is initialized as X → $X^+ = \{A_1, A_2\}$

Step 2 $\{A_1, A_2\} \rightarrow \{A_5\}$, $X^+ = \{A_1, A_2\}$ → A_5 is added to X^+ → $X^+ = \{A_1, A_2, A_5\}$

Step 2 $\{A_5\} \rightarrow \{A_6\}$, $X^+ = \{A_1, A_2, A_5\}$ → A_6 is added to X^+ → $X^+ = \{A_1, A_2, A_5, A_6\}$

Step 2 $\{A_2, A_6\} \rightarrow \{A_3, A_4\}$, $X^+ = \{A_1, A_2, A_5, A_6\}$ → A_3, A_4 is added to X^+ → $X^+ = \{A_1, A_2, A_3, A_4, A_5, A_6\}$

Step 2 $\{A_3, A_6\} \rightarrow \{A_7\}$, $X^+ = \{A_1, A_2, A_3, A_4, A_5, A_6\}$ → A_7 is added to X^+ → $X^+ = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$

Step 3 All the attributes are added to X^+ → $X^+ = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$

Schema Normalization

Normal Forms

Normal Forms: Desirable Formal Schema Properties

1NF

First Normal Form

- Each row must be unique
- Each cell must contain only a single value

2NF

Second Normal Form

- Must be in in 1NF
- Create separate tables for sets of values that apply to multiple rows

3NF

Third Normal Form

- For all FDs A ⟶ b whose attributes are in same table
  - Either b is element in A ("trivial" FD)
  - Or A contains a key of its associated table
  - Or b is part of some minimal key (allows some redundancy)
- This must apply to given and inferred FDs!

BCNF

Boyce-Codd Normal Form

- For all FDs A ⟶ b whose attributes are in same table
  - Either b is element in A ("trivial" FD)
  - Or A contains a key of its associated table
- This must apply to given and inferred FDs!

Schema Normalization

Normal Forms

- First normal form (1NF)
  - Each row must be unique
  - Each cell must contain only a single value
- Second normal form (2NF)
  - Must be in in 1NF
  - Create separate tables for sets of values that apply to multiple rows
- Third Normal Form (3NF)
  - A schema is in 3NF if the following conditions holds
    - For all FDs A ⟶ b whose attributes are in same table
      - Either b is element in A ("trivial" FD)
      - Or A contains a key of its associated table
    - Or b is part of some minimal key (allows some redundancy)
  - This must apply to given and inferred FDs!

- Boyce-Codd Normal Form (BCNF)
  - A schema is in BCNF if the following conditions holds
    - For all FDs A ⟶ b whose attributes are in same table
      - Either b is element in A ("trivial" FD)
      - Or A contains a key of its associated table
    - This must apply to given and inferred FDs!
  - Does not permit any redundancy!

Schema Normalization

Normal Forms

- BCNF Example

Table 1 (a,b)

Table 2 (a,d,e)

Table 3 (c,d)

Initial FDs

$$
\begin{aligned}
\boldsymbol{F}=\{ & \\
& \{a\} \rightarrow \{b\}, \\
& \{b, c\} \rightarrow \{d\}, \\
& \{a\} \rightarrow \{c\}, \\
& \{d\} \rightarrow \{a, e\} \\
\} &
\end{aligned}
$$

Is this schema in BCNF?

Schema Normalization

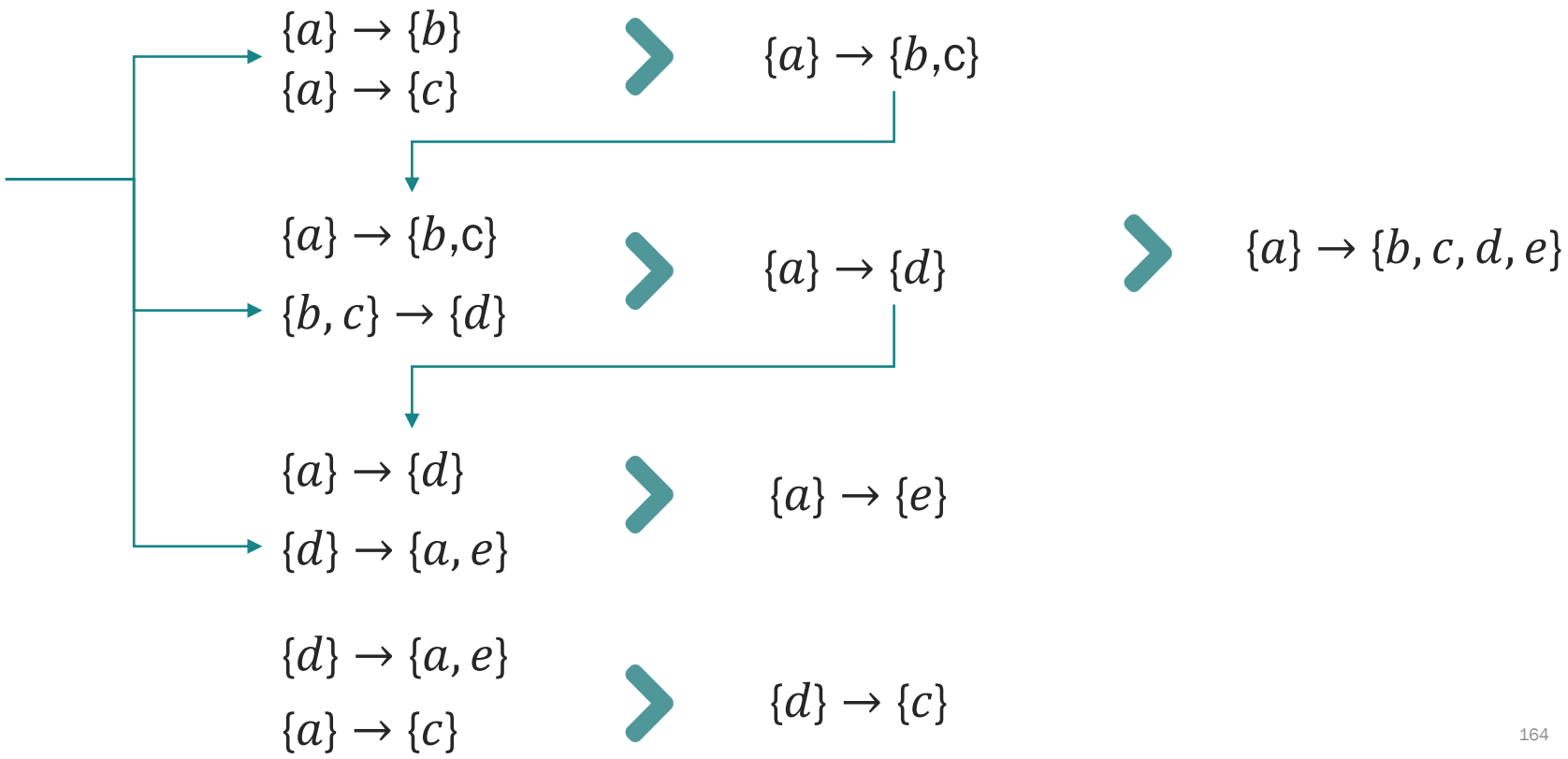
Normal Forms

Answer

We need to first infer all the FDs from the given set of FDs

Initial FDs

$$F=\{\ \{a\}\rightarrow\{b\},\ \{b,c\}\rightarrow\{d\},\ \{a\}\rightarrow\{c\},\ \{d\}\rightarrow\{a,e\}\ \}$$



Schema Normalization

Normal Forms

Answer

Now we need to check for any given FD, the BCNF condition:

$F = \{$

- $\{a\} \rightarrow \{b\}$, > ***a*** and ***b*** are both in Table 1, ***a*** is a key ✓
- $\{b, c\} \rightarrow \{d\}$, > ***b, c,*** and ***d*** are not all in the same table ✓
- $\{a\} \rightarrow \{c\}$, > ***a*** and ***c*** are not all in the same table ✓
- $\{d\} \rightarrow \{a, e\}$, > ***a, d*** and ***e*** are all in Table 2, ***d*** is a key ✓
- $\{a\} \rightarrow \{b, c\}$, > ***a, b*** and ***c*** are not all in the same table ✓
- $\{a\} \rightarrow \{d\}$, > ***a*** and ***d*** are both in Table 2, ***a*** is a key ✓
- $\{a\} \rightarrow \{e\}$, > ***a*** and ***e*** are both in Table 2, ***a*** is a key ✓
- $\{d\} \rightarrow \{c\}$, > ***c*** and ***d*** are both in Table 3, ***d*** is the key ✓
- $\{a\} \rightarrow \{b, c, d, e\}$ > ***They*** are not in the same table ✓

$\}$

Table 1 (a,b)

Table 2 (a,d,e)

Table 3 (c,d)

✓ The given schema is in BCNF

Schema Normalization

Normal Forms

- 3NF Example

Table 1 (a,b)

Table 2 (a,d,e)

Table 3 (c,d)

Initial FDs

$$
\begin{aligned}
\boldsymbol{F}=\{ & \\
& \{a\} \rightarrow \{b\}, \\
& \{b, c\} \rightarrow \{d\}, \\
& \{a\} \rightarrow \{c\}, \\
& \{d\} \rightarrow \{a, e\} \\
\} &
\end{aligned}
$$

Is this schema in 3NF?

Schema Normalization

Comparison of Normal Forms

3NF

Third Normal Form

Allows redundancy in some cases

- **Pro:** can always preserve dependencies
- **Con:** may still have some negative effects

BCNF

Boyce-Codd Normal Form

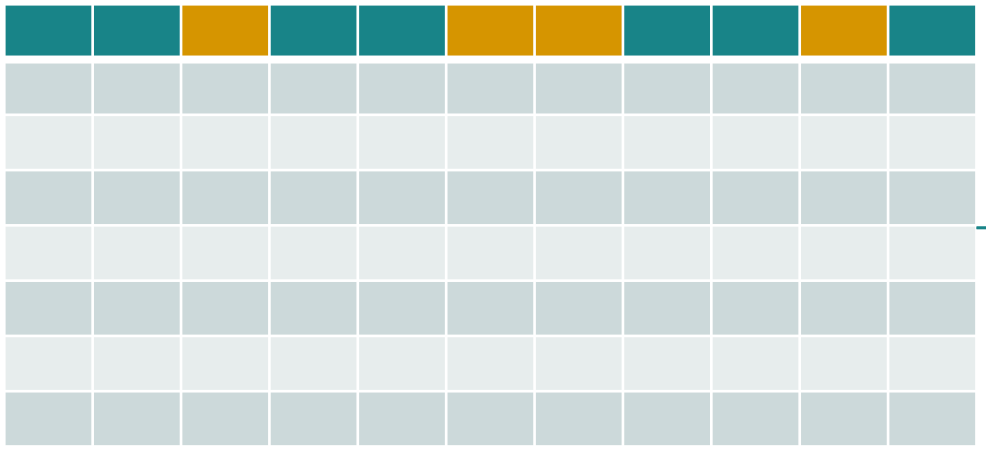
Disallows any redundancy

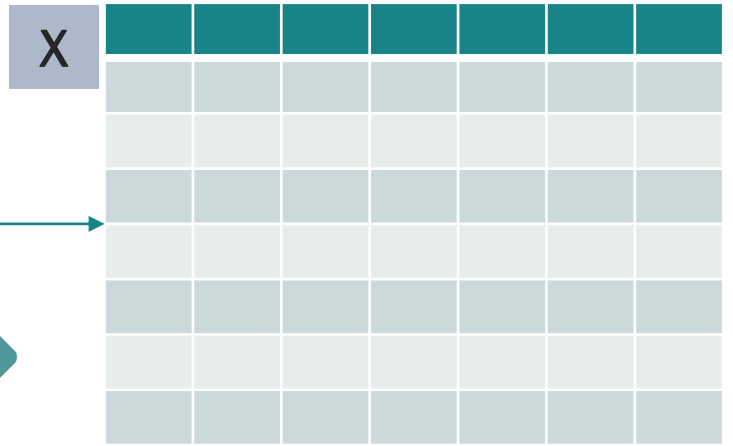
- **Pro:** avoids all negative effects of redundancy
- **Con:** may require breaking up dependencies

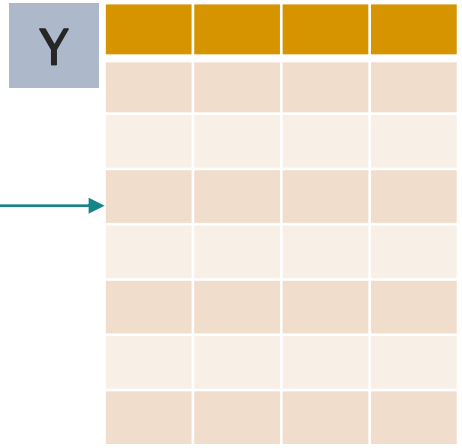
Schema Normalization

Decomposition

R Not in a Normal Form







We can decompose **R** if
$X \cap Y \rightarrow X$ or $X \cap Y \rightarrow Y$ is an FD

Schema Normalization

Lossless Decomposition and Recomposition

Name	Position Type	Salary
Max	Full Time	10000
Farzad	Part Time	5000
Alex	Full Time	10000
Mary	Full Time	10000
David	Part Time	5000
Robert	Full Time	10000
Sam	Part Time	5000

Name	Position Type
Max	Full Time
Farzad	Part Time
Alex	Full Time
Mary	Full Time
David	Part Time
Robert	Full Time
Sam	Part Time

Position Type	Salary
Full Time	10000
Part Time	5000

Schema Normalization

Decomposition

- Normal forms impose conditions on FDs in single table
- Decompose tables into smaller tables to satisfy them
- Decomposition must allow to reconstruct original data
  - Assume we decomposed R into X and Y
  - We can do so if $X \bigcap Y \longrightarrow X$ or $X \bigcap Y \longrightarrow Y$ is an FD
    - Can then match each row from Y to one row from X/ Can then match each row from X to one row from Y

Schema Normalization

Decomposition Algorithm

Toward BCNF

Step 1 Repeat while some FD $A \rightarrow b$ on R violates BCNF rules

Step 2 Decompose R into $R\text{-}b$ and Ab

Step 3 Repeat step 1 until no FD violates BCNF rules

b : a single attribute

A : a set of attributes

End result may depend on decomposition order

Schema Normalization

Decomposition Algorithm

- Towards BCNF
  - Repeat while some FD A⟶b on R violates BCNF rules
    - Decompose R into R-b and Ab
  - All decompositions are lossless as (R-b)⋂Ab=A⟶b
  - Will terminate as tables get smaller and smaller
  - End result may depend on decomposition order

Schema Normalization

Decomposition Algorithm

- Example

A

$$A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$$

A_1 is the key

F

$$F = \{ \{A_3, A_5\} \rightarrow \{A_1\}, \{A_2, A_4\} \rightarrow \{A_5\}, \{A_3\} \rightarrow \{A_2\} \}$$

Bring into BCNF

Schema Normalization

Decomposition Algorithm

- Example

A $A = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$ A_1 is the key

F $F=\{$
$\{A_3, A_5\} \rightarrow \{A_1\},$
$\{A_2, A_4\} \rightarrow \{A_5\},$
$\{A_3\} \rightarrow \{A_2\}$
$\}$

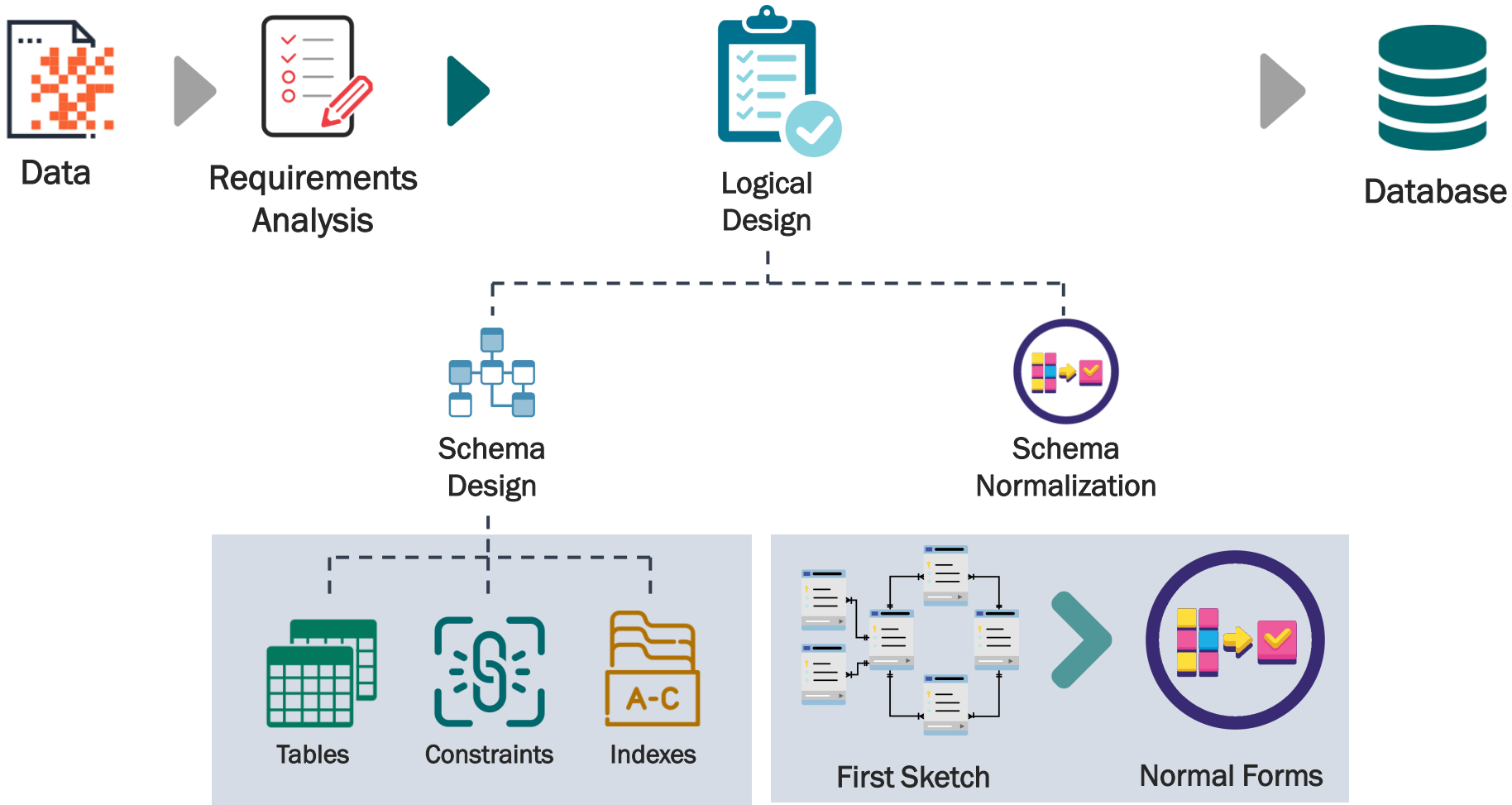
$\{A_3, A_5\} \rightarrow \{A_1\}$ > Violates BCNF ✗ > Decompose into $\{A_1, A_3, A_5\}$, $\{A_2, A_3, A_4, A_5, A_6, A_7\}$

$\{A_3\} \rightarrow \{A_2\}$ > Violates BCNF ✗ > Decompose $\{A_2, A_3, A_4, A_5, A_6, A_7\}$ into $\{A_2, A_3\}$, $\{A_3, A_4, A_5, A_6, A_7\}$

$\{A_2, A_4\} \rightarrow \{A_5\}$ > No violation ✓

Final Database Schema $\{A_1, A_3, A_5\}, \{A_2, A_3\}, \{A_3, A_4, A_5, A_6, A_7\}$

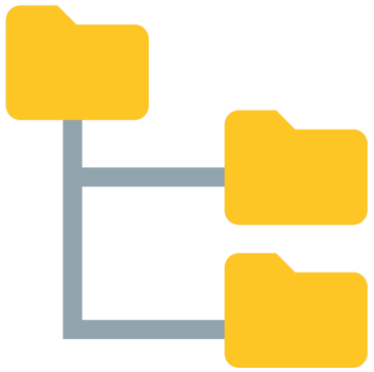
Schema Design



Database Management Systems

Data Management

History



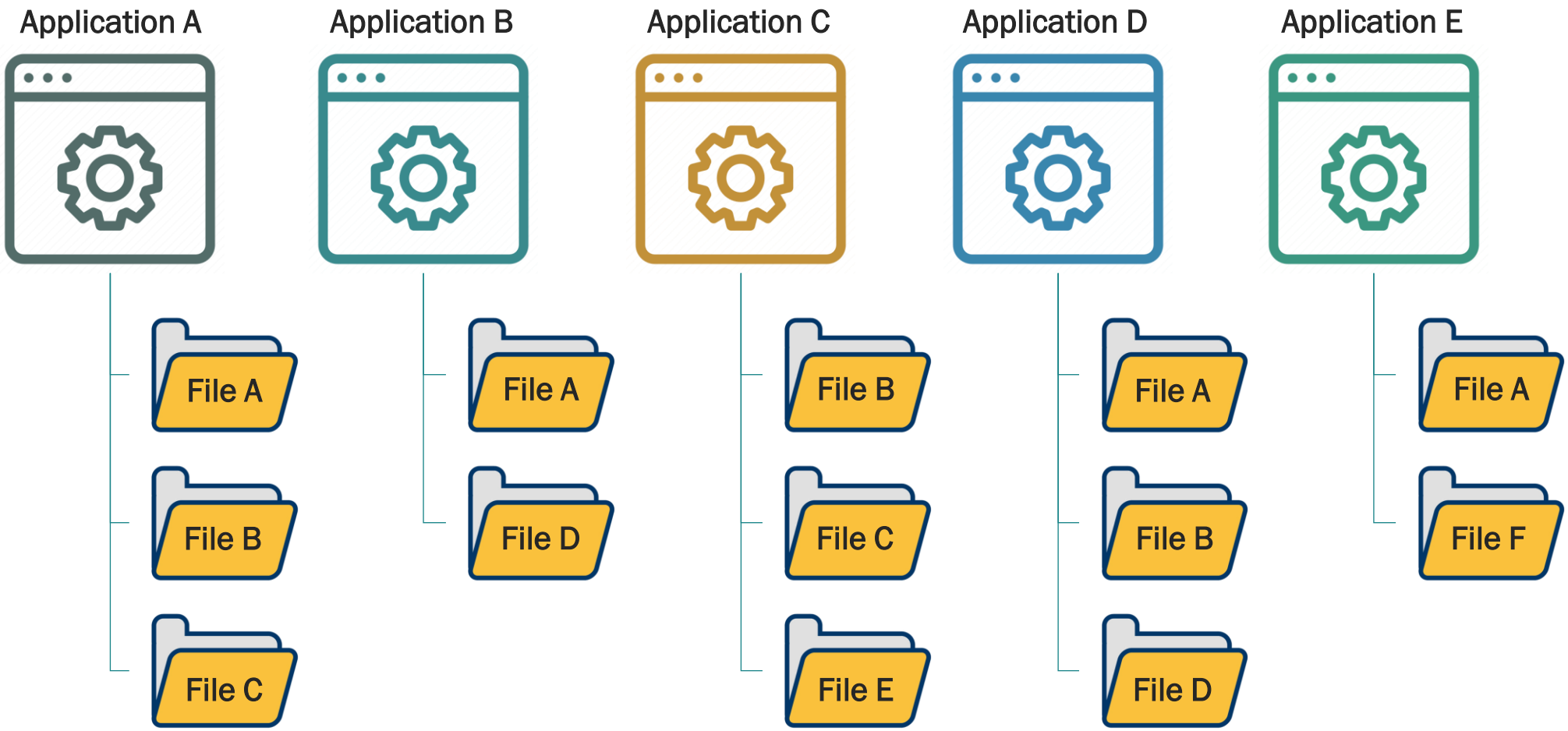
File-Based Approach

Shared-File Approach

Database Management Systems

Data Management

Traditional File-Based Approach



Data Management

Traditional File-Based Approach

- The term 'file-based approach' refers to the situation where data is stored in one or more separate computer files defined and managed by different application programs.
  - Typically, for example, the details of customers may be stored in one file, orders in another, etc.
- Computer programs access the stored files to perform the various tasks required by the business.
  - Each program, or sometimes a related set of programs, is called a computer application.
  - For example, all of the programs associated with processing customers' orders are referred to as the order processing application.
  - The file-based approach might have application programs that deal with purchase orders, invoices, sales and marketing, suppliers, customers, employees, and so on.

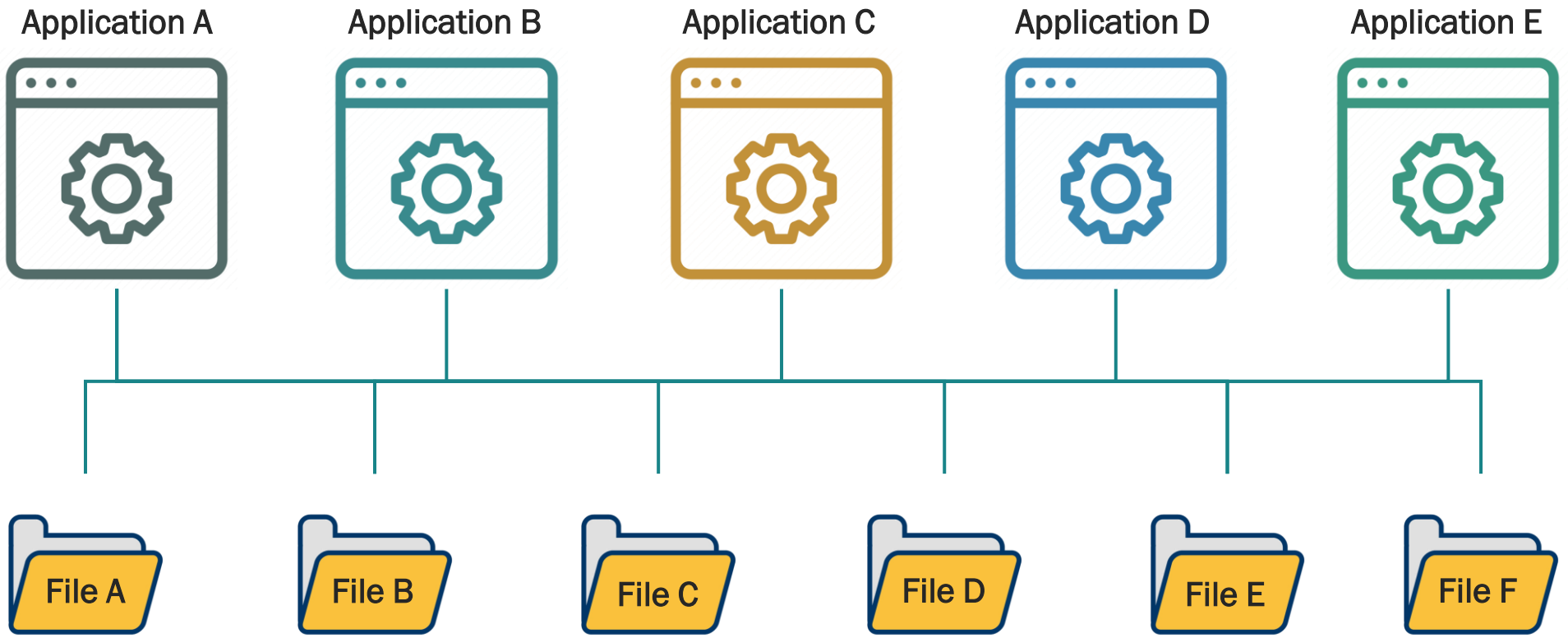
Data Management

Limitations and Disadvantages

- Data overlap, redundancy and duplication
  - Each program stores its own separate files. If the same data is to be accessed by different programs, then each program must store its own copy of the same data.
- Data inconsistency
  - If the data is kept in different files, there could be problems when an item of data needs updating, as it will need to be updated in all the relevant files; if this is not done, the data will be inconsistent, and this could lead to errors.
- Code overlap
- Difficult to implement data security
  - Data is stored in different files by different application programs. This makes it difficult and expensive to implement organization-wide security procedures on the data.

Data Management

Shared-File Approach



Data Management

Shared-File Approach

- One approach to solving the problem of each application having its own set of files is to share files between different applications.
- This will alleviate the problem of duplication and inconsistent data between different applications.
- The introduction of shared files solves the problem of duplication and inconsistent data across different versions of the same file held by different departments, but other problems may emerge

Data Management

Limitations and Disadvantages

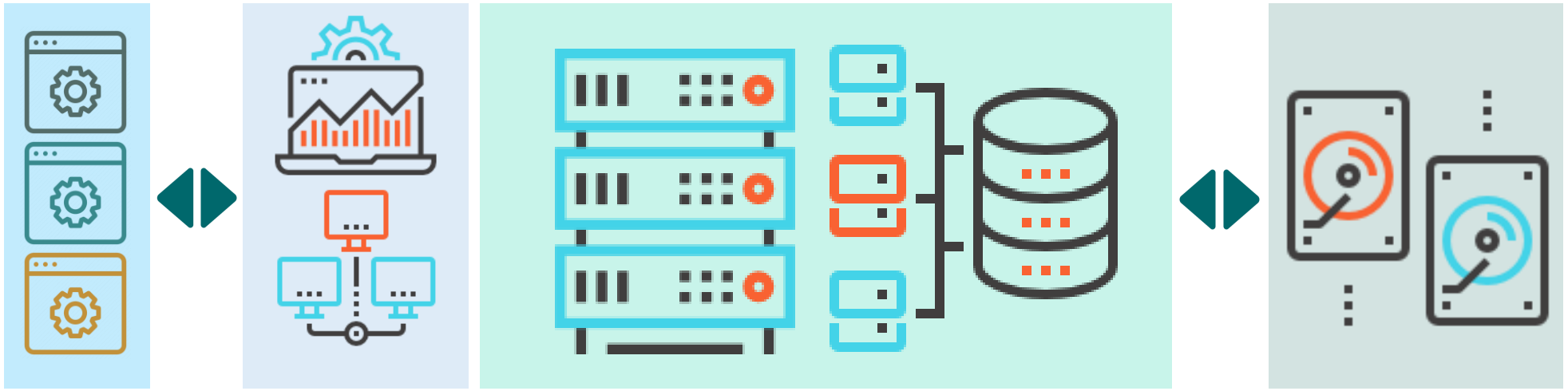
- File incompatibility:
  - When each department had its own version of a file for processing, each department could ensure that the structure of the file suited their specific application. If departments have to share files, the file structure that suits one department might not suit another. For example, data might need to be sorted in a different sequence for different applications (for instance, customer details could be stored in alphabetical order, or numerical order, or ascending or descending order of customer number).
- Difficult to control access:
  - Some applications may require access to more data than others; for instance, a credit control application will need access to customer credit limit information, whereas a delivery note printing application will only need access to customer name and address details. The file will still need to contain the additional information to support the application that requires it.

Data Management

Limitations and Disadvantages

- Physical data dependence:
  - If the structure of the data file needs to be changed in some way (for example, to reflect a change in currency), this alteration will need to be reflected in all application programs that use that data file. This problem is known as physical data dependence.
- Difficult to implement concurrency:
  - While a data file is being processed by one application, the file will not be available for other applications or for ad hoc queries. This is because, if more than one application is allowed to alter data in a file at one time, serious problems can arise in ensuring that the updates made by each application do not clash with one another. This issue of ensuring consistent, concurrent updating of information is an extremely important one, and is dealt with in detail for database systems in the chapter on concurrency control. File-based systems avoid these problems by not allowing more than one application to access a file at one time.

Database Management Systems

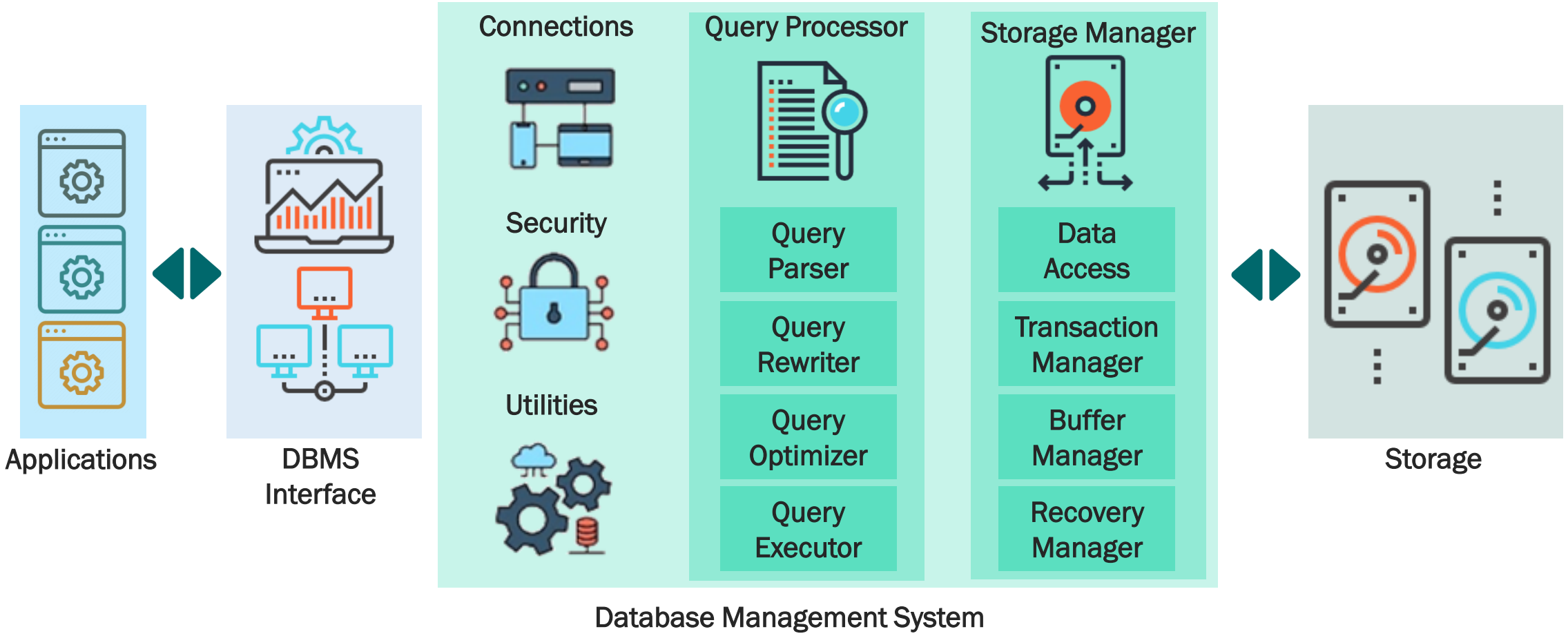


Database Management System

- A set of programs that creates and maintains the database. It allows you to store, modify, and extract information from the database using a function called querying.
- A database management system (DBMS) is a software package designed to define, manipulate, retrieve and manage data in a database.
- A DBMS generally manipulates the data itself, the data format, field names, record structure and file structure.
- It also defines rules to validate and manipulate this data.
- Advantages:
  - Single interface,
  - one universal management system,
  - no data redundancy or inconsistency

Database Management Systems

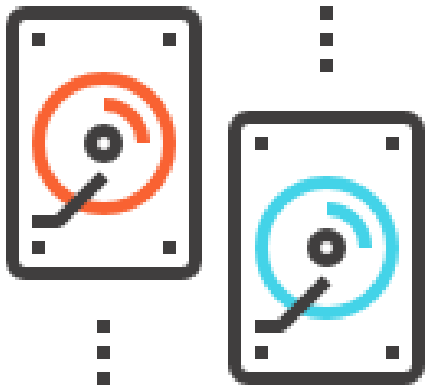
Components



Storage Manager

Database Management Systems

Storage Manager



Storage Media
(Hardware)

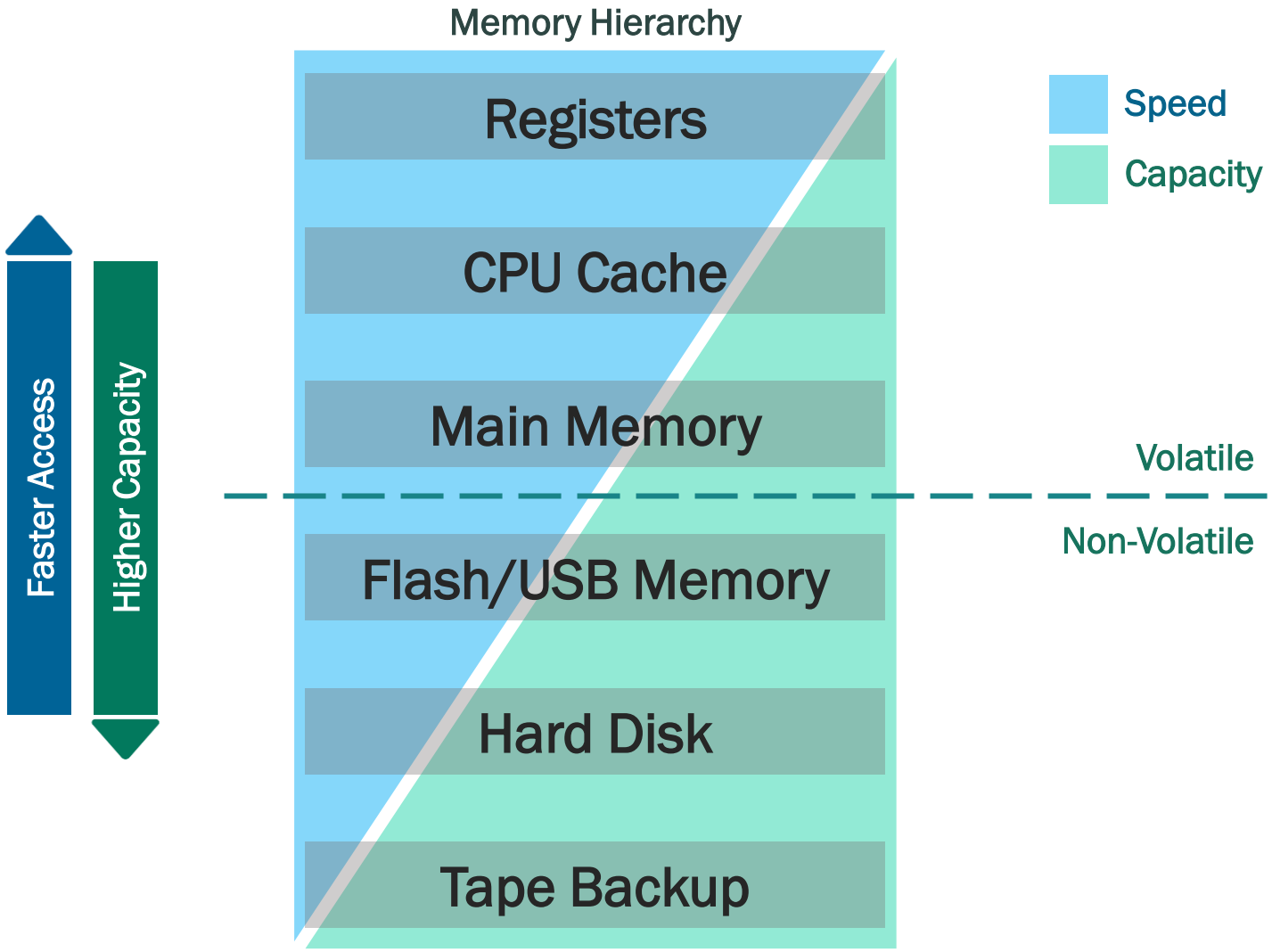
What devices to store data on?

Storage Format
(Layout)

How to represent data (relations)?

Storage Manager

Storage Media



Storage Manager

Storage Media

- Why knowing memories?
  - Capacity limits force data to lower parts of hierarchy
  - Data access speed may become bottleneck
  - Design algorithms to minimize data movements
  - Random data access is expensive
  - Read data in larger chunks ("pages")
  - Keep related data close together
  - Consider volatility for recovery considerations

Storage Manager

Tape Storage

Bits as magnetic information on tape

Very slow access
(10s of seconds)

Moderate read speed
(up to 300 MB/second)

Very cheap
(around $0.02 per Gigabyte)

Used for long-term archival
(e.g., by Google)

Storage Manager

Hard Disk Drive (HDD)

Bits as magnetic information on platter
(Patters spin under read/write heads)

Slow access
(10s of milliseconds access time)

Moderate read speed
(around 200 MB/second)

Cheap
(around $0.035 per Gigabyte)

Used for less frequently accessed data

Storage Manager

Solid State Drive (SSD)

Bits as small electric charges

Fast access
(around 1 millisecond)

Elevated speed
(around 500 MB/second)

Elevated price
(around $0.25 per Gigabyte)

Limited number of write cycles (memory wear)

Storage Manager

Main Memory

Bits as small electric charges

Very fast access
(order of nanoseconds)

High bandwidth
(Gigabytes per second)

Expensive
(several dollars per Gigabyte)

Used to access hot data
(all if economically feasible!)

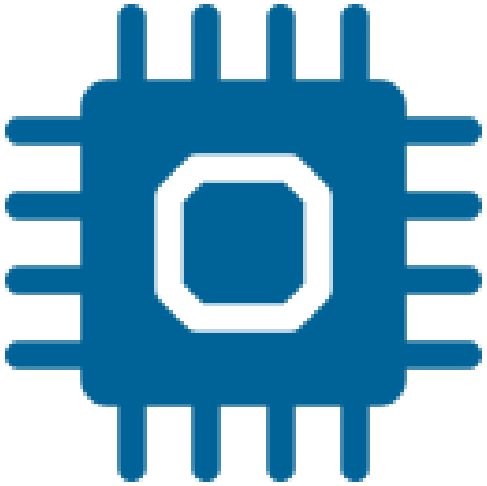
Storage Manager

Cache Memory

Bits as small electric charges
(Typically organized as cache hierarchy)

Near-instantaneous access
(few nanoseconds)

Very high bandwidth
(tens of Gigabytes per second)



Very expensive
(hundreds of dollars per Gigabyte)

Used to store immediately relevant data

Storage Manager

Storage Format/Layout

Tables

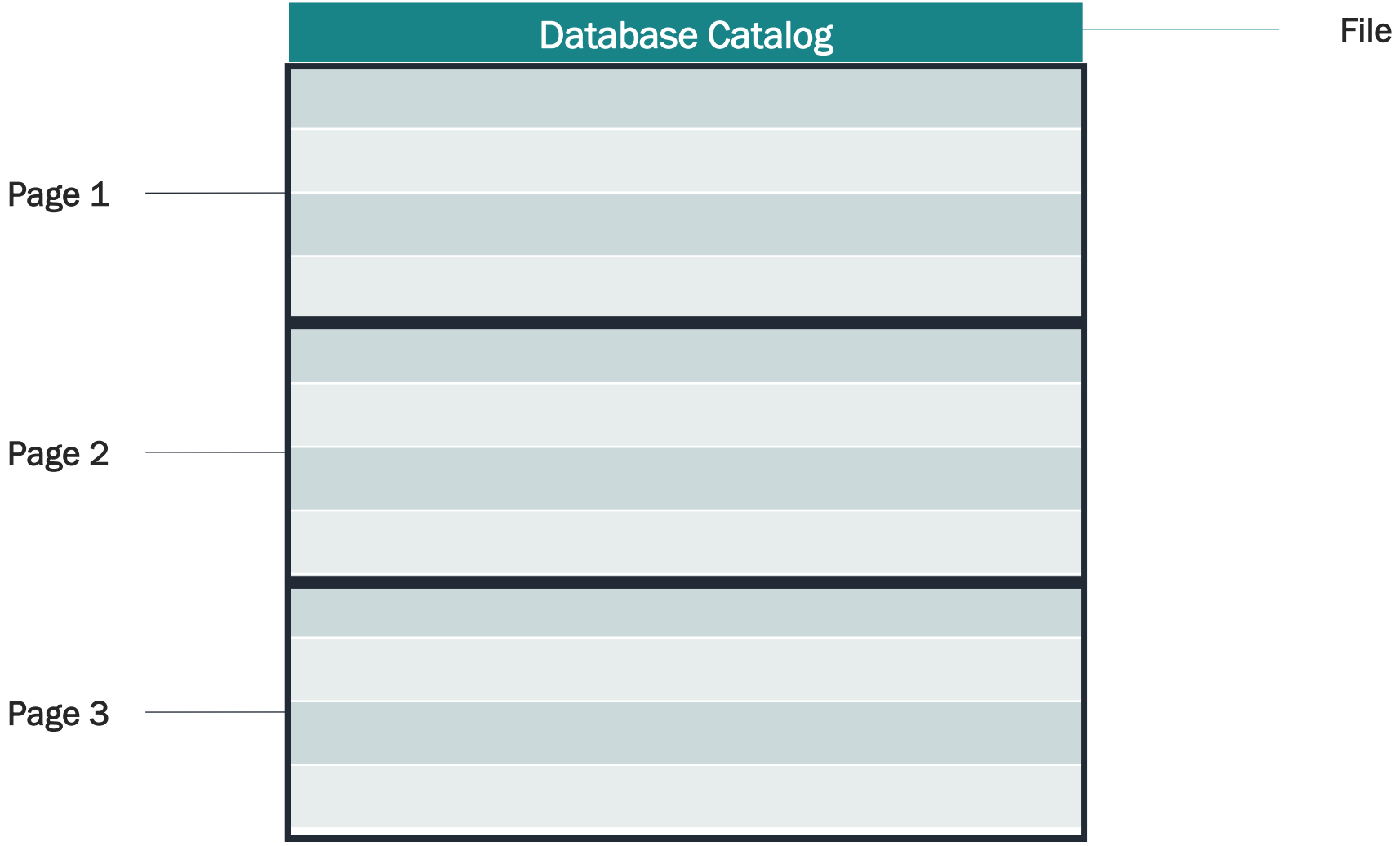
User's Perspective

Files

Storage Perspective

Storage Manager

Tables as Files



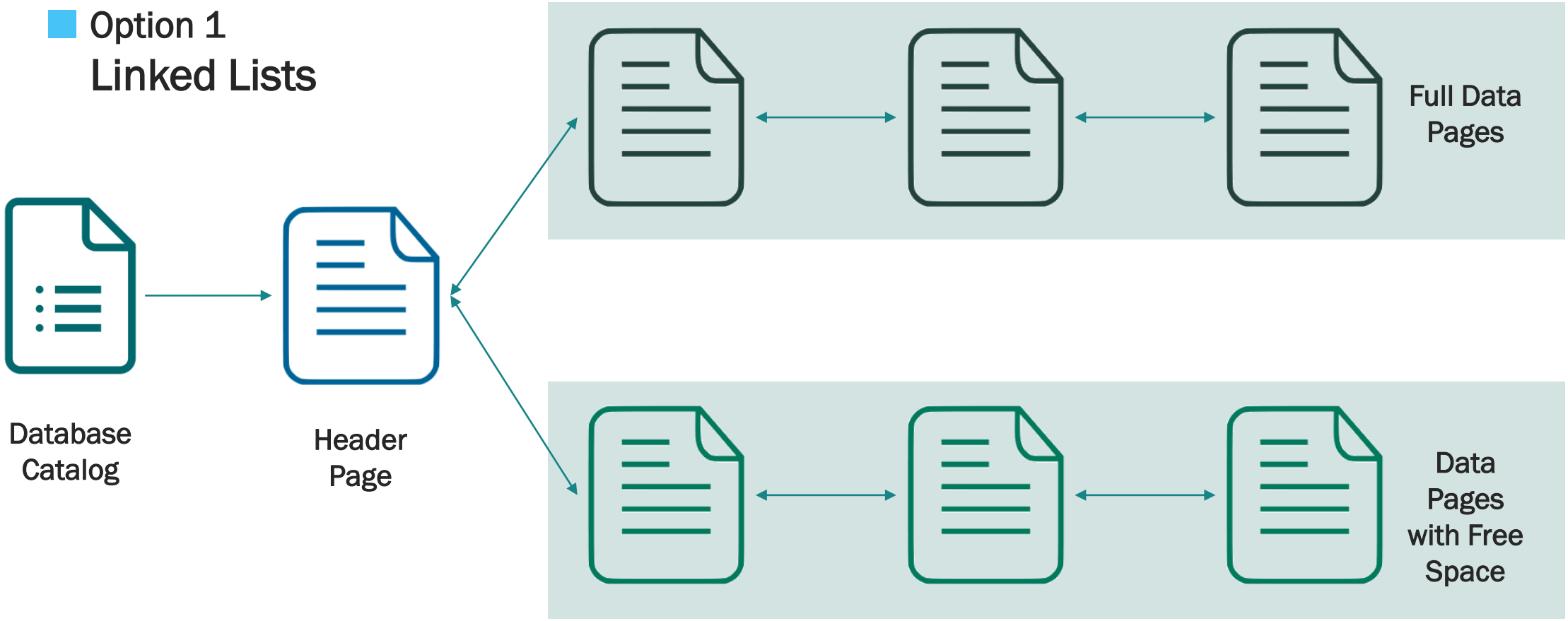
Storage Manager

Tables as Files

- Table schema information is stored in database catalog
- Table content is stored as collection of pages (“file”)
  - Each page typically stores a few KB of data
  - Enough to store multiple rows but not the entire table

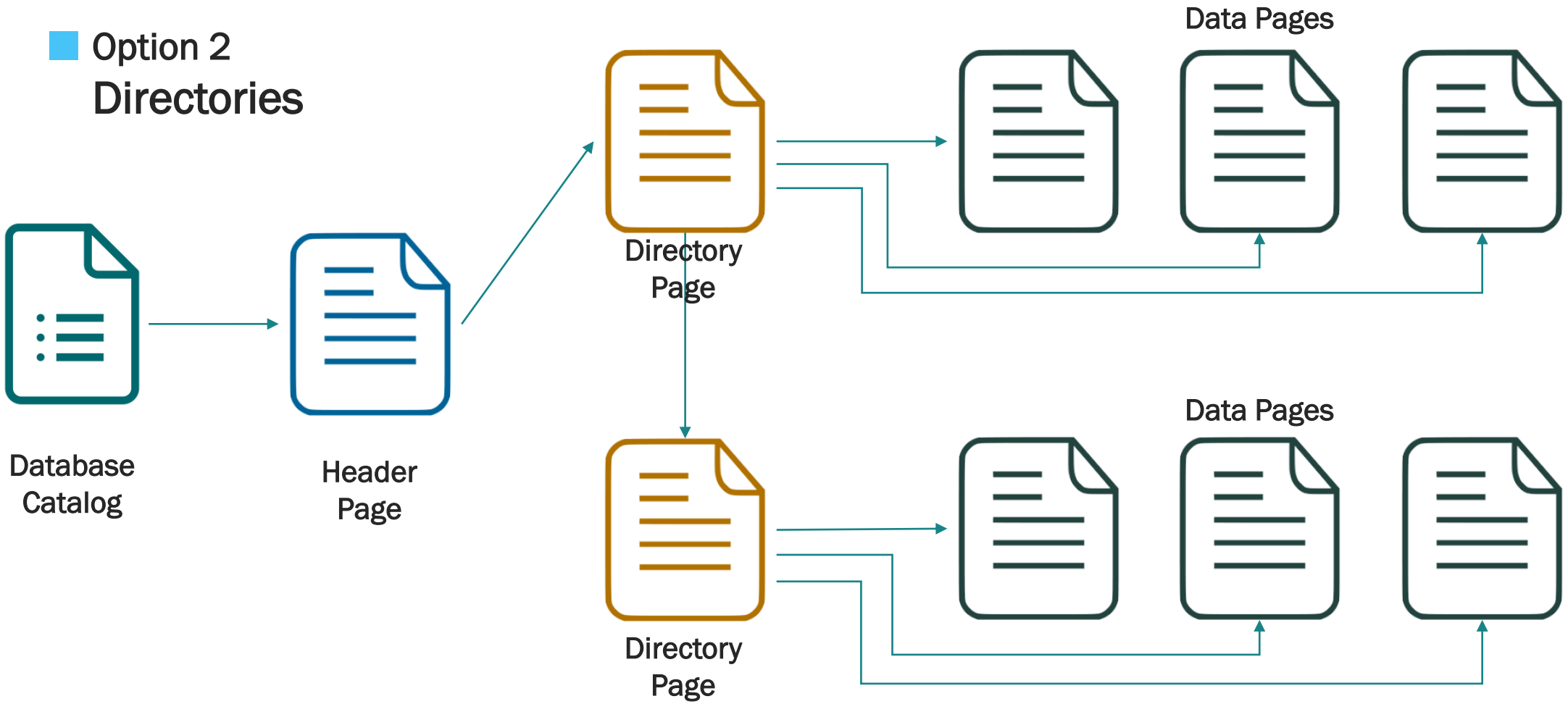
Storage Manager

From Files to Pages



Storage Manager

From Files to Pages



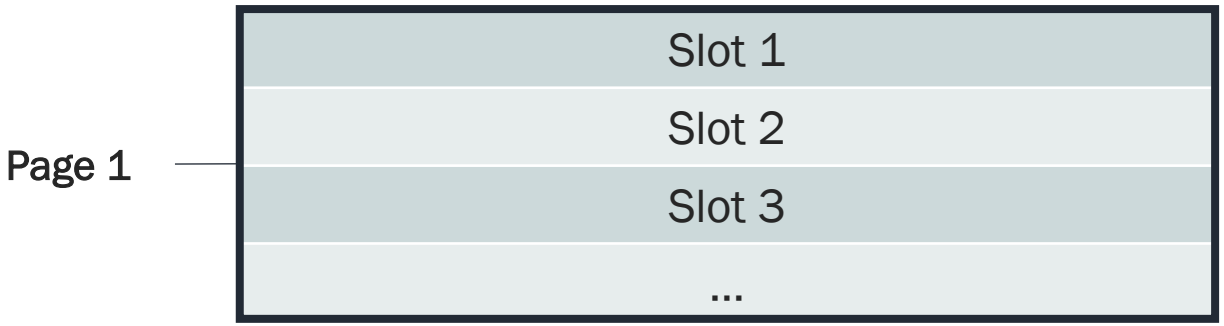
Storage Manager

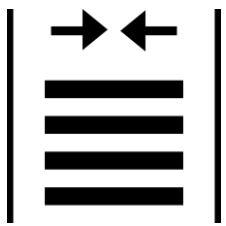
From Files to Pages

- Possibility 1: store pages as (doubly) linked list
  - Each page contains pointers to next/prior page
  - Can use separate lists for full/partially empty pages
  - Reference to header page stored in DB catalog
- Possibility 2: directory with pointers to pages
  - Directory pages reference data pages with meta-data

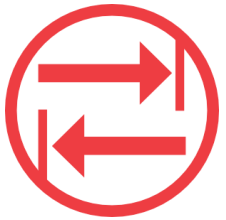
Storage Manager

From Pages to Slots





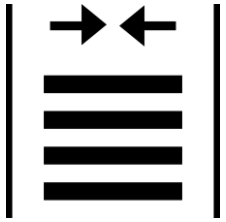
Fixed Length Records



Variable Length Records

Storage Manager

From Pages to Slots



Fixed Length Records

Packed Representation
USED
USED
USED
USED
USED
FREE
FREE
FREE
FREE
FREE
5

Unpacked Representation
USED
USED
FREE
FREE
USED
FREE
USED
USED
FREE
USED
1100101101

Storage Manager

From Pages to Slots

- Pages are divided into slots:
- Each slot stores one record (one table row)
- Can refer to records via(pageID, slotID)
- Multiple ways to divide pages into slots
  - Fixed-length records
    - Number of bytes per slot is determined a-priori
    - Need to keep track of which slots are used (insertions …)
    - Packed representation uses consecutive slots
      - Only keep track of number of slots used
      - What is the problem with deletion in this representation?
    - Unpacked representation allows unused slots in-between
      - Need bitmap to keep track of used slots

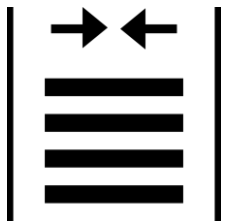
Storage Manager

From Pages to Slots

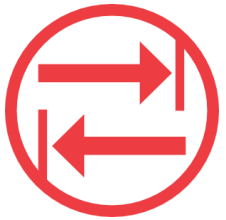
- Variable length records
  - E.g., records with variable-length text fields
  - Number of bytes per slot is not fixed a-priori
  - Each page maintains directory about used slots
  - Store first byte and length of slots
  - Flexibility to move around records on page
  - Can use that for regular compaction

Storage Manager

From Slots to Fields



Fixed Length Fields



Variable Length Fields

Storage Manager

From Slots to Fields

- Slides are divided into fields
- Fixed length fields
  - store field sizes in DB catalog
- Variable length fields
  - store field sizes on page
  - Option 1: use special delimiter symbol between fields
  - Option 2: store "field directory" at beginning of record

Data Access

Data Access

Indexes

What are indexes?

Real world Examples

Libraries

Books are stored and sorted based on the subject and alphabetically

Books

Indexes in books are used to keep references to words mentioned in the book

Data Access

Indexes

- Indexes are auxiliary data structures for finding data faster.
- Why do we use indexes?
  - Table Enrollment(sid, cid) links students to courses
  - E.g., search entries for specific student (e.g., sid=5)
  - Data stored as unordered file - must scan all pages!
  - Better: data sorted by student ID - apply binary search!
  - Problem: sometimes search for specific courses!
  - Only one sort order, cannot duplicate data …
- Real world example: books & Library
- We can have multiple indexes for the same table
- How it works
  - Index stores references to data records
  - I.e., stores page IDs and slot IDs
  - Index groups records by values in specific columns
  - Those columns are called the index search key
  - Index retrieves records for specific search key values

Data Access

Indexes

Enrollment Table

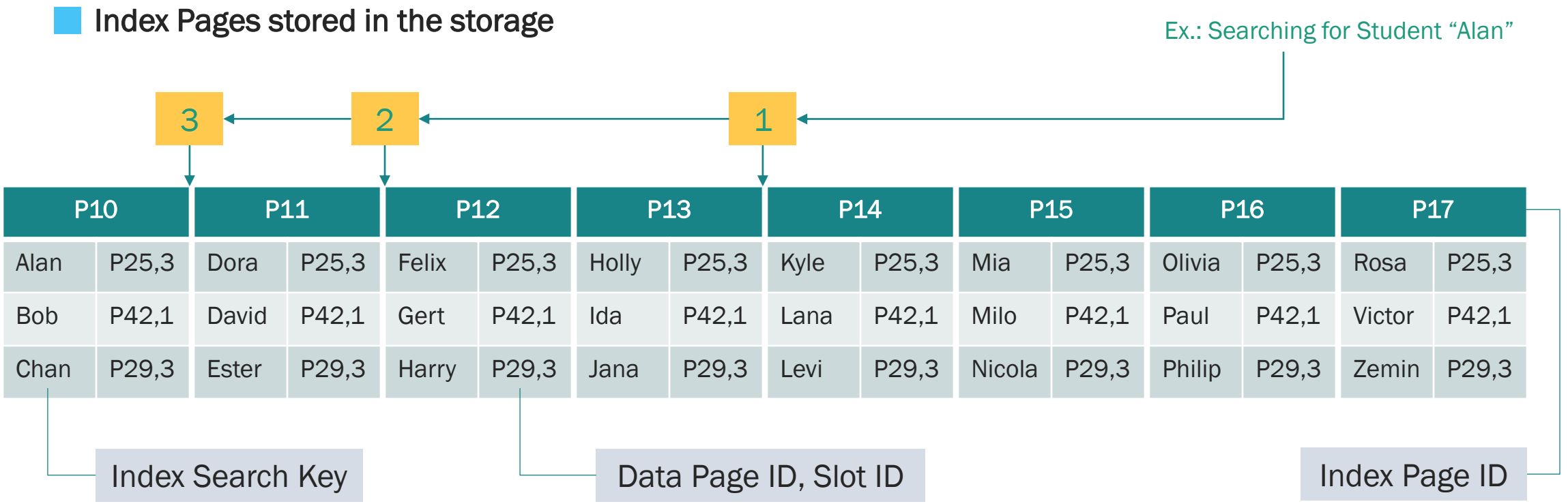
Student ID	Course ID	Student Name	Course Name	...

Index

On student name column

Data Access

Indexes



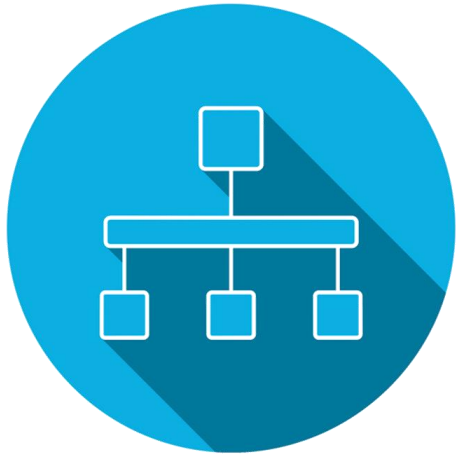
Data Access

Indexes

- Ideas for improvement?
- Binary search narrows down "search space" by factor 2
- Can we get a higher pruning factor per page read?
- Idea: (non-binary) search trees!

Data Access

Index Types



Tree Indexes

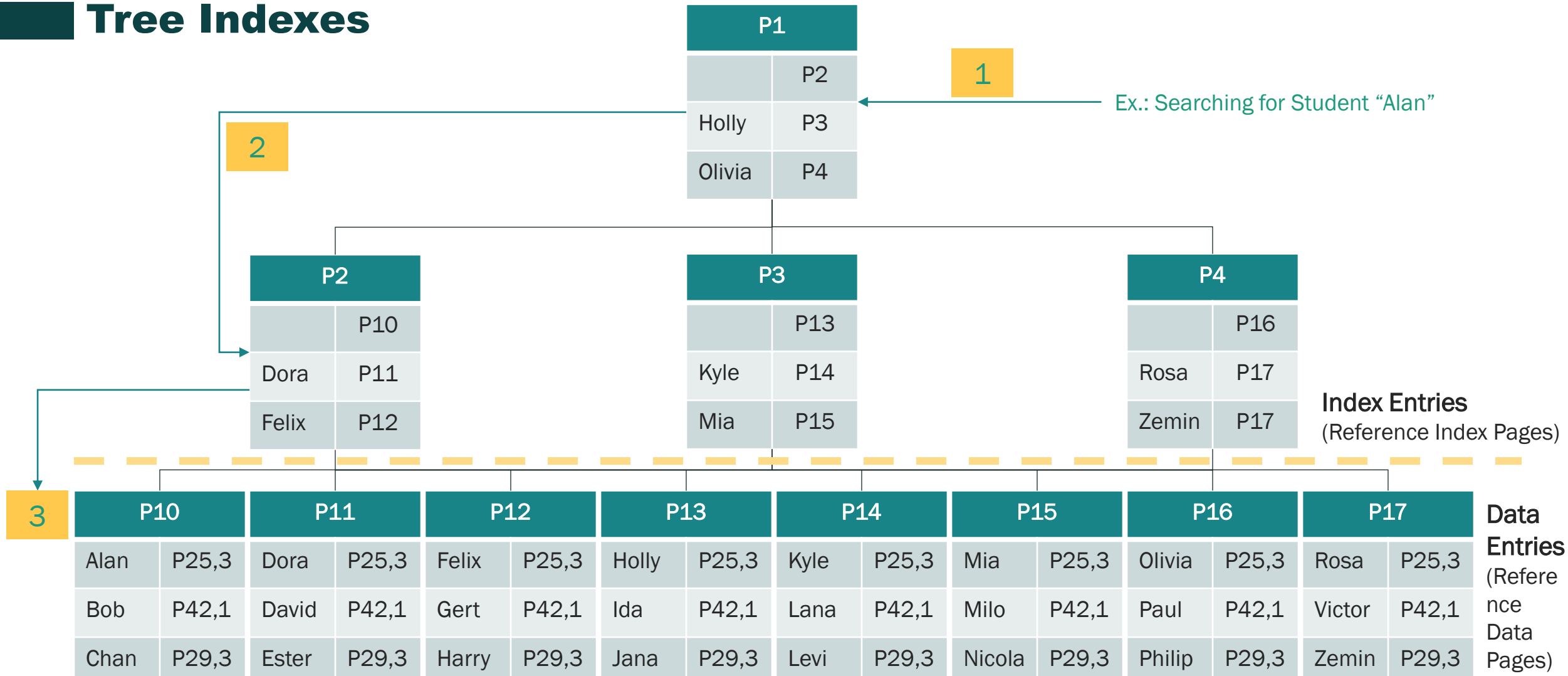
Traverse search tree to find leaves

Hash Indexes

Evaluate hash function to find buckets

Data Access

Tree Indexes



Data Access

Tree Indexes

- Where to use Tree indexes?

Equality Predicates

WHERE first_name = ‘Alan’

Inequality Predicates

WHERE age > 25

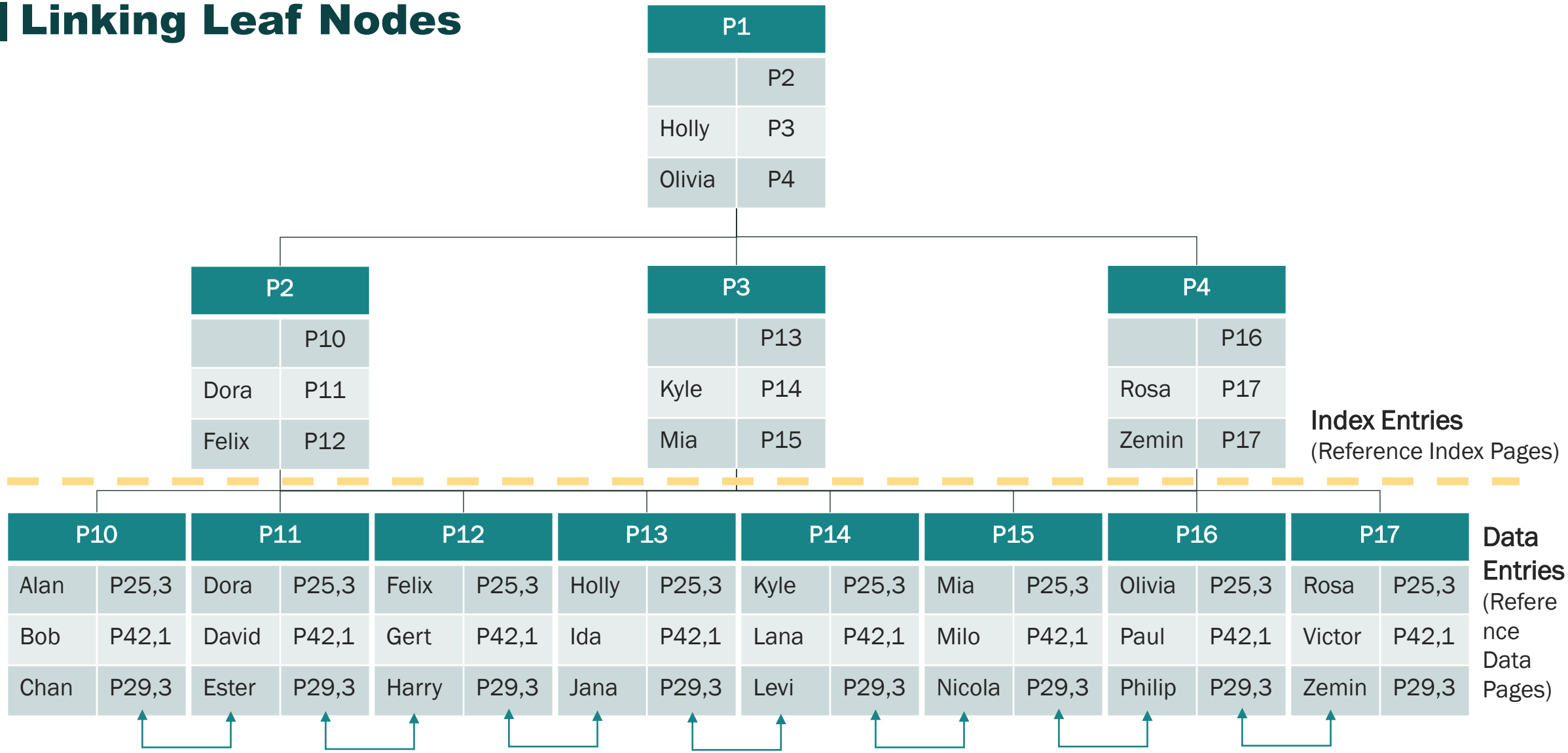
Data Access

Tree Indexes

- Tree indexes are based on a sort order between keys
- Can handle equality and inequality conditions
  - Consecutive keys stored close together
- Composite keys: useful for conditions on key prefix
  - Keys with same prefix value stored close together

Data Access

Linking Leaf Nodes



Data Access

Linking Leaf Nodes

- Often want entries from neighboring leaf nodes
- Could get leaf node references from parent nodes
- Better: store pointer to next/previous neighbor in leaf
- Leaf pages essentially become doubly linked list

Data Access

Indexes

- Which Indexes to create?

Typical and frequent queries

Predicates of the queries

Too many indexes are bad

Active area of research

Data Access

Updating Indexes

- Why indexes should be updated?

Index refers to database table

If data changes, so must the index

Need to change index in case of inserts/deletes

Not required but can improve efficiency

Data Access

Hash Indexes

Hash Function

Page ID = Hash % No. of Buckets

P0		P1		P2		P3	
Bob	P25,3	Alan	P25,3	Chan	P25,3	Olivia	P25,3
Jana	P42,1	David	P42,1	Gert	P42,1	Rosa	P42,1
Zemin	P29,3	Ida	P29,3	Lana	P29,3		

Data Entries
(Reference Data Pages)

Hash Function
(Key and Hash values)

Key	Hash
Alan	1
Bob	0
Chan	2
Dora	5
David	1
Ester	7
Felix	4
Gert	2
Holy	7
Ida	1
Jana	0
Kyle	6
Lana	6
Levi	5
Olivia	3
Philip	7
Rosa	3
Tia	6
Victor	5
Zemin	4

Data Access

Hash Indexes

- Where to use Hash indexes?

Equality Predicates

WHERE first_name = ‘Alan’

Data Access

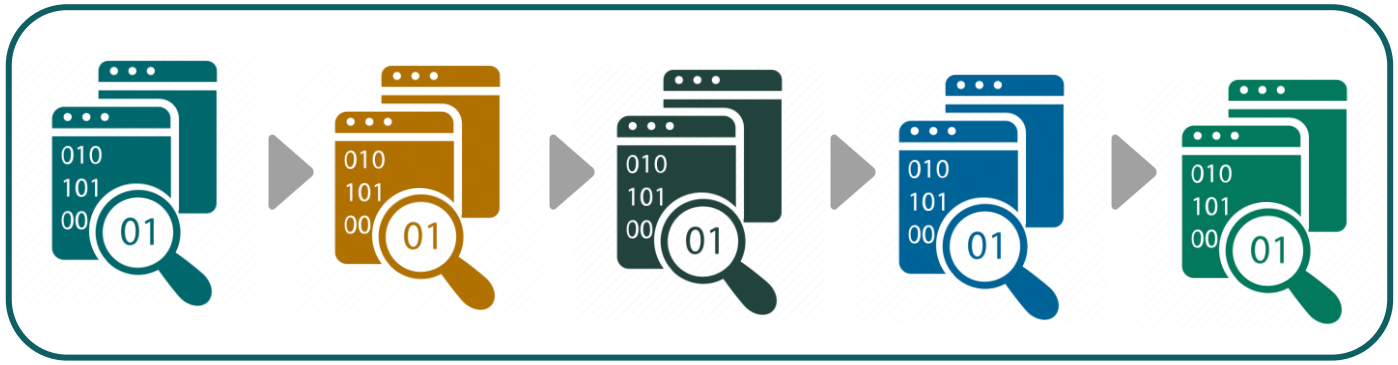
Hash Indexes

- Equality predicates
- Hash indexes are based on key hash values
- Only useful for equality conditions
- Consecutive keys may be stored far apart
- Similar hash value ⇏ similar key value
- Condition must constrain all components
- Keys with same prefix may be stored far apart

Transaction Manager

Transaction Manager

Transactions



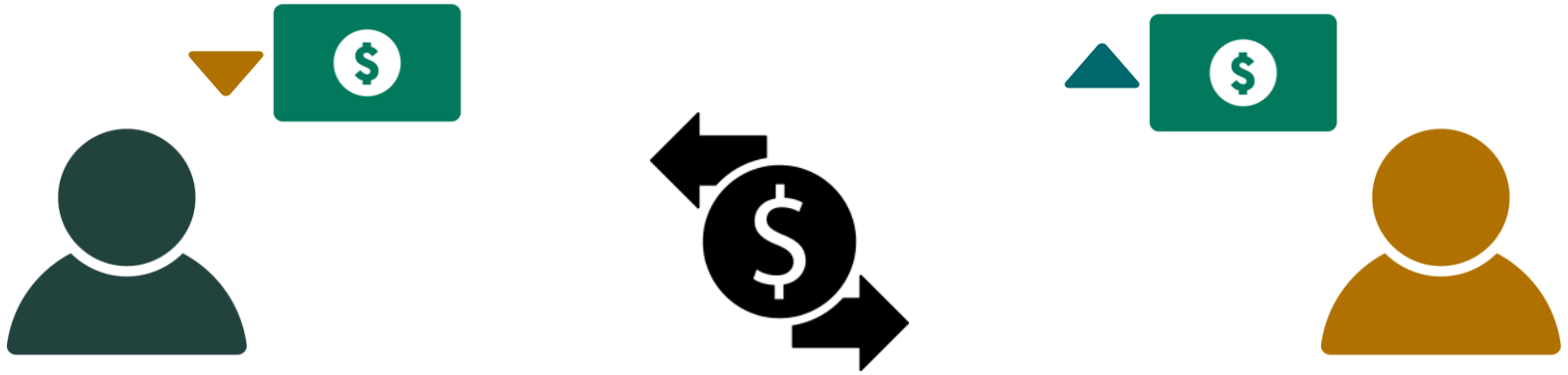
Transaction

A sequence of updates (or queries) that are "connected"

Transaction Manager

Transactions: Example

- Banking: Money Transfer



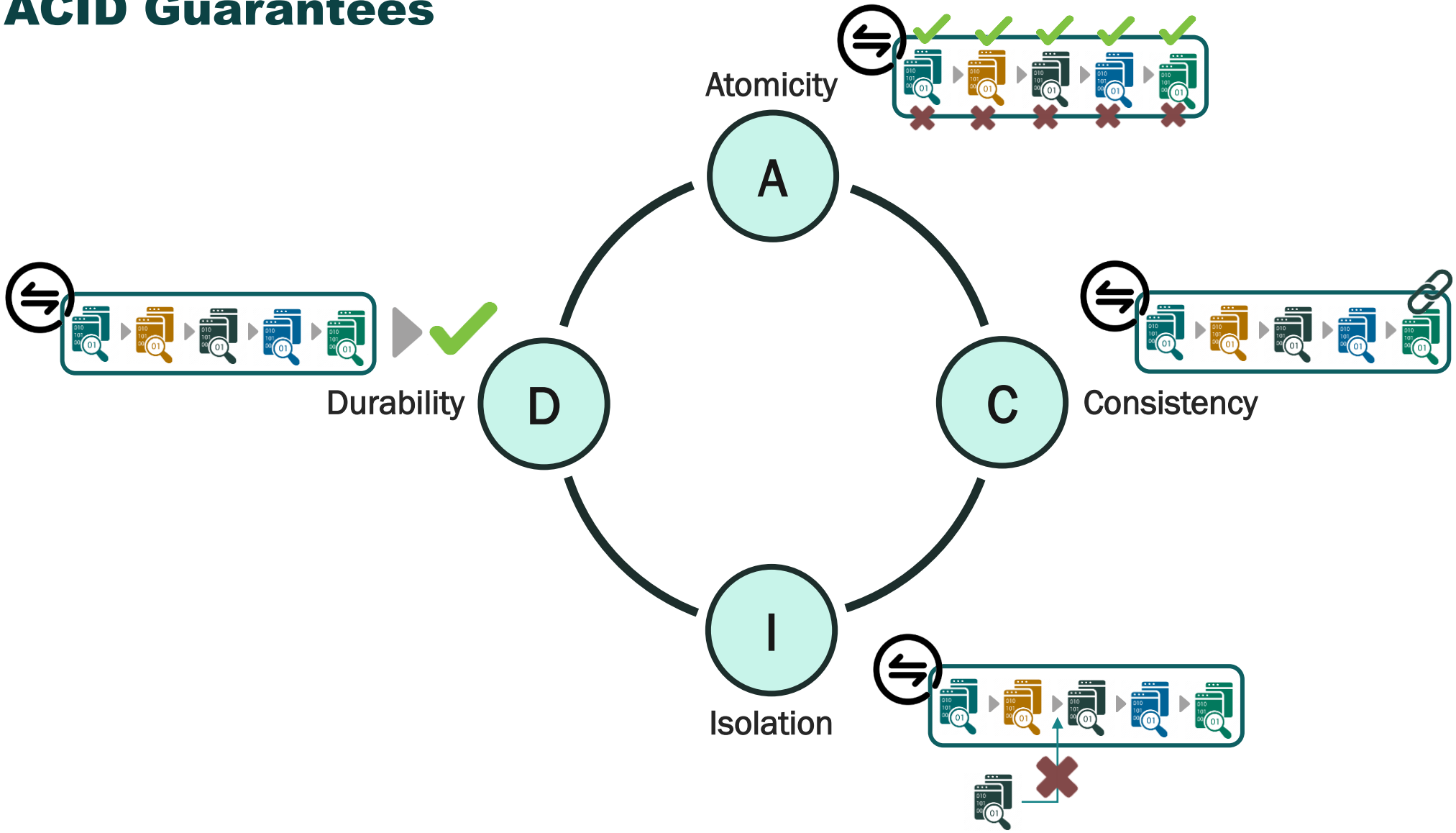
Transaction Manager

Transactions

- Transaction
  - A sequence of updates (or queries) that is "connected"
  - DBMS commands for assigning queries to transactions
  - DBMS makes certain guarantees about transactions
  - E.g., executes entire transaction or no part of it
- Example: banking
  - Want to wire $50 from Farzad to Max
  - Step 1: Subtract $50 from Farzad’s account
  - Step 2: Add $50 to Max’s account
  - Both steps are semantically related
  - I.e., want to execute both or none

Transaction Manager

ACID Guarantees



Transaction Manager

ACID

- Most RDBMS give ACID guarantees for transactions
- Atomicity (either execute all or nothing)
  - Atomicity means all steps execute or none of them
- Why would a transaction not fully execute?
  - System crash due to bugs or power failure
  - Transaction step violates integrity constraints
  - Explicit transaction abort (e.g., PG: ROLLBACK;)
- Internally, DBMS executes steps separately
  - May have to do cleanup in case of partial execution

Transaction Manager

ACID

- Consistency (enforce all integrity constraints)
- Consistency means data is consistent with all constraints
  - Primary key constraints
  - Referential integrity (foreign key constraints)
  - More complex constraints can be defined
- DBMS will abort transactions threatening consistency
- Careful, differs from distributed systems consistency

Transaction Manager

ACID

- Isolation (avoid interleaving transactions badly)
- Different users may execute transactions concurrently
  - E.g., bank has many clients transferring money
- Executing transactions sequentially is inefficient
  - E.g., can overlap idle times with other transaction
- DBMS may interleave steps from multiple transactions
- Isolation means simulating sequential execution to users

Transaction Manager

ACID

- Why interleaving?
- Motivation 1: long running transactions
  - Imagine user submitting very short transaction
  - User may have long wait if scheduled behind long transaction
  - Better: alternate between transaction steps
  - Long transaction barely slower, short transaction quick
- Motivation 2: idle time e.g. by disk access
  - Assume transaction 1 needs data from disk for next step
  - Could load data while executing step from other transaction

Transaction Manager

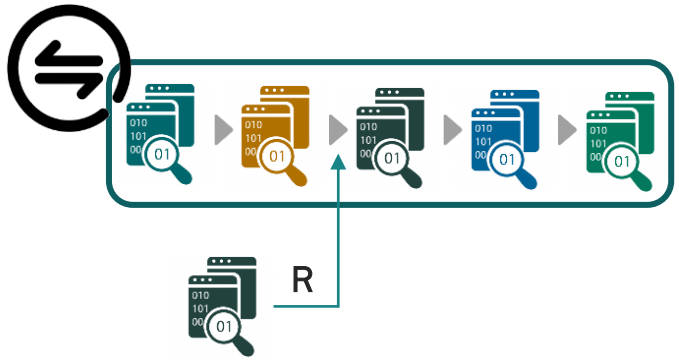
ACID

- Durability (ensure that updates are not lost)
- DBMS notifies users that transaction has committed
- Durability guarantees that committed updates persist
- This must hold even if DBMS or server crashes
- Must store enough info on disk to restore at startup
- Only notify user of commit after that has happened

Transaction Manager

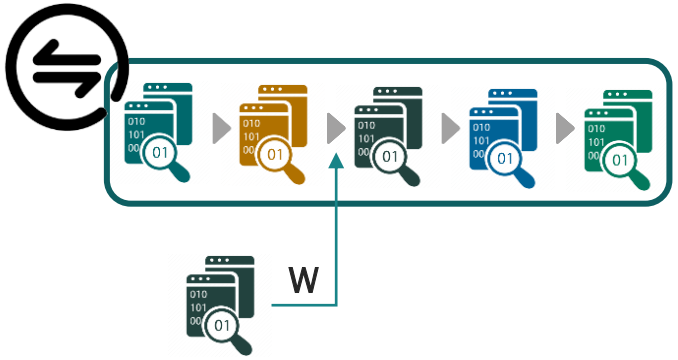
Isolation Anomalies

Dirty Reads



Unrepeatable Reads

Lost Updates



Transaction Manager

Isolation Anomalies

- Anomaly: may destroy illusion of sequential execution so DBMS wants to avoid them as much as possible
- Dirty reads: read data from unfinished transaction
- We read data written by uncommitted transaction
- E.g., what if writing transaction aborts?
- Need to undo all effects of aborted transaction
- Strange effects even if writing transaction commits
- Anomaly signature with short notation: Wx(A) Ry(A)

Transaction Manager

Isolation Anomalies

- Unrepeatable reads: data changes while working with it
- Reading committed data may be problematic, too
- We read data twice, changed from outside in between
- Means we read different values without changing value
- E.g., check if at least one item stored (read 1), proceed
- Other transaction reduces item count to zero
- Now try to reduce item count by one (read 2 & write)
- Anomaly signature in short notation: Rx(A) Wy(A) Cy Rx(A)

Transaction Manager

Isolation Anomalies

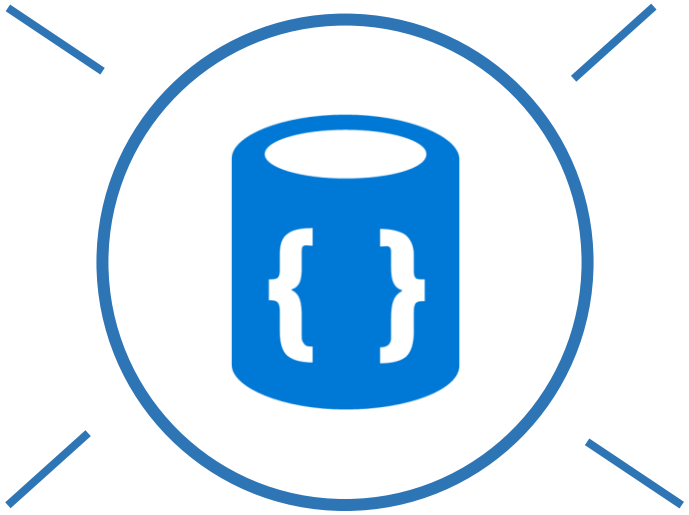
- Lost updates: unsaved changes are overridden
- We override value written by ongoing transaction
- E.g., want to pay same salary for all employees
- Have two transactions updating salary to different values
- Constraint holds if transactions execute sequentially
- But may not hold if interleaving transactions
- Anomaly signature in short notation: Wx(A) Wy(A)

Non-Relational Databases

Non-Relational Databases

NoSQL

Provides flexible schemas for the storage and retrieval of data
Makes it possible to store data in a schema-less or free-form fashion



Influenced by advances in **cloud computing**, the **internet of things**, and **social media** proliferation

Emerged in response to the **volume**, **diversity**, and **speed** at which data is being generated today

Built for **speed**, **flexibility**, and **scale**

NoSQL (not only SQL) is widely used for processing big data

Some non-relational databases come with their own querying tools
MQL (MongoDB), CQL (Cassandra), GraphQL (Neo4J)

Non-Relational Databases

NoSQL

- The name was introduced at an event to discuss all of the new open-source distributed databases that were coming onto the scene, and the name, NoSQL, has stuck ever since
- It does not mean ‘ No SQL’ – it means ‘Not only SQL’
- refers to a family of databases that vary widely in style and technology
- but share a common trait in that they are non-relational in nature,
- are not a standard row and column relational database management system or RDBMS
- a better name to describe them: non-relational database
- provide new ways of storing and querying data that address a number of issues for modern applications
- are geared to handle a different breed of scale problems that have arisen associated with the “big data” movement
- by scale we are referring to:
  - size of the data
  - concurrent users
- are typically also more specialized in their use cases and can be much simpler to develop application functionality for, than relational databases
- what ties NoSQL databases together: majority have their roots in open source community and many have been used and leveraged in an open source manner

Non-Relational Databases

NoSQL

Flexibility in structure

Ability to handle large volumes of structured, semi-structured, and unstructured data

Scalability and Availability

Simpler design, better control over availability, and improved scalability that enables you to be more agile, more flexible, and to iterate more quickly

Supports distributed computing

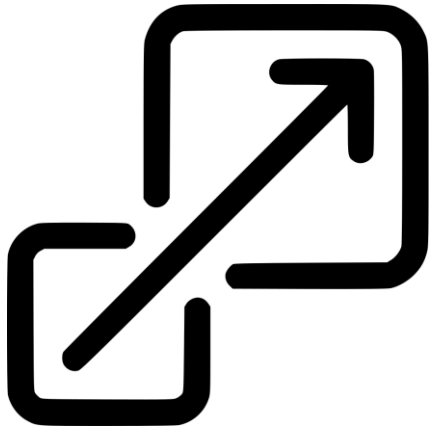
Ability to run as distributed systems scaled across multiple data centers, which enables them to take advantage of cloud computing infrastructure; an efficient and cost-effective scale-out architecture that provides additional capacity and performance with the addition of new nodes

Advantages of NoSQL

- Are built to scale horizontally
- Share data more easily than RDBMS
- Use a global unique key to simplify data partitioning/sharding
- More use case specific than RDBMS
- More developer friendly than RDBMS
- Allow more agile development via flexible schemas

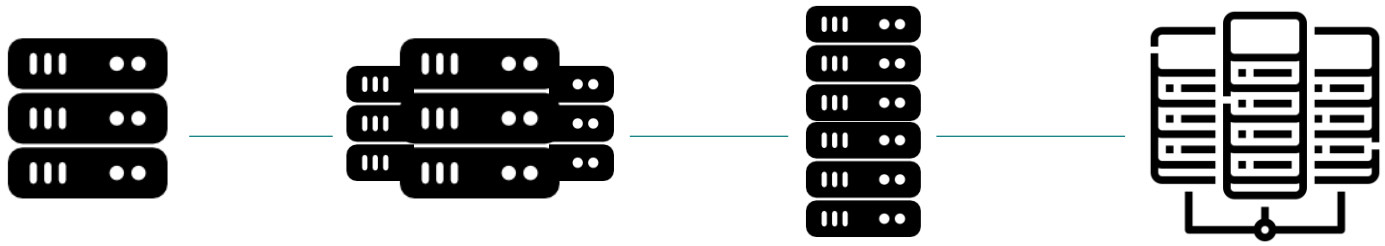
Non-Relational Databases

Benefits of NoSQL



Scalability

The ability to scale horizontally across clusters of servers, server racks and even data centers



Server

Server Cluster

Server Racks

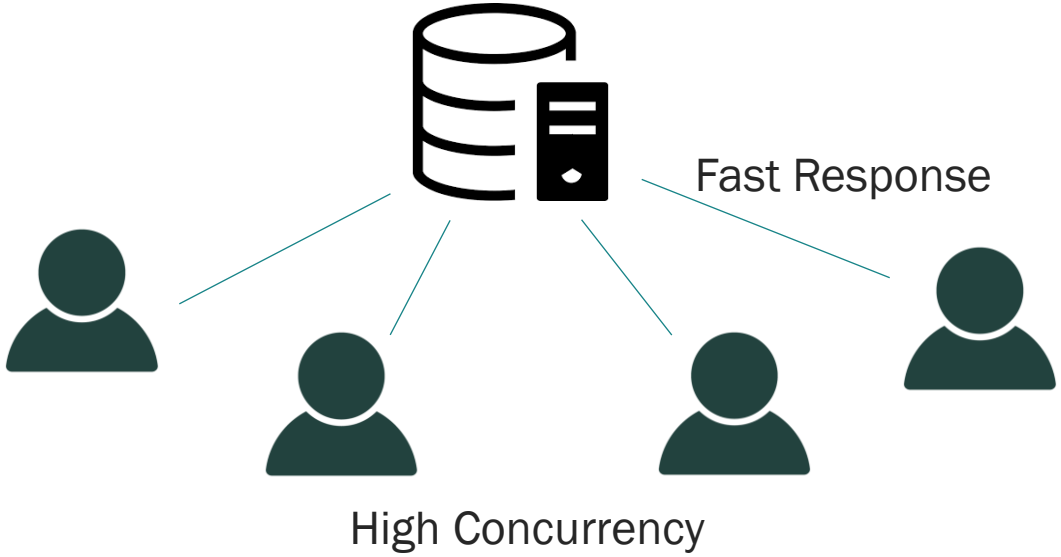
Data Center

Non-Relational Databases

Benefits of NoSQL

Performance

Fast response times and high concurrency of users and big data

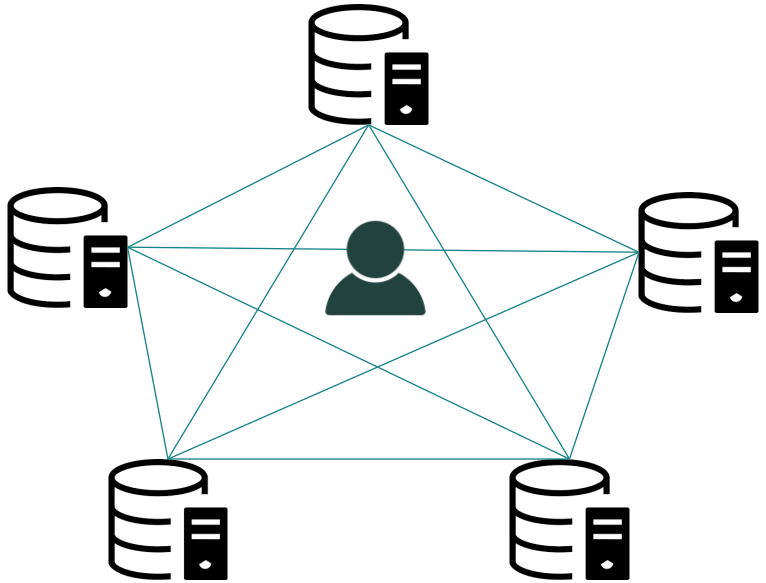


Non-Relational Databases

Benefits of NoSQL

Availability

By having a database run on a cluster of servers than just a single server



Non-Relational Databases

Benefits of NoSQL

Cloud Architecture

The distributed data nature of NoSQL databases means that they can be deployed and operated on clusters of servers in cloud architecture thereby reducing costs.

Non-Relational Databases

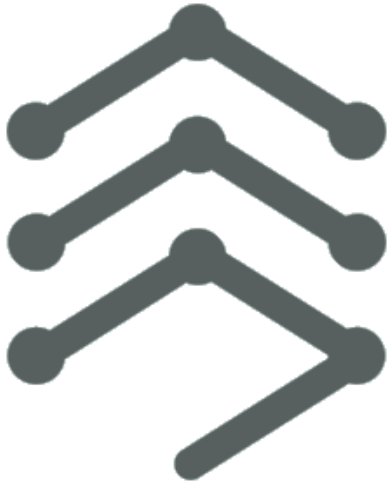
Benefits of NoSQL

Costs

NoSQL has much less costs compared to existing databases while having same or better performance or functionality

Non-Relational Databases

Benefits of NoSQL

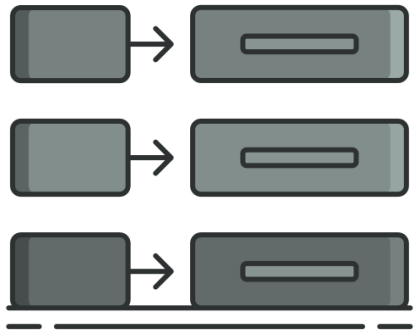


Schema Flexibility

Most NoSQL databases allow for having flexible and intuitive schemas which means new feature can easily be added to existing applications without any database locking or downtime.

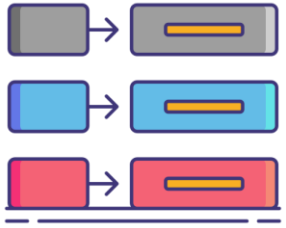
Non-Relational Databases

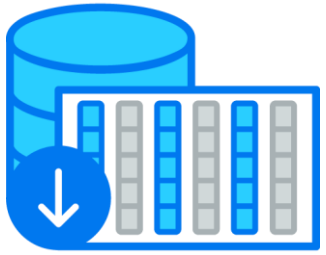
Benefits of NoSQL



Varied Data Structure

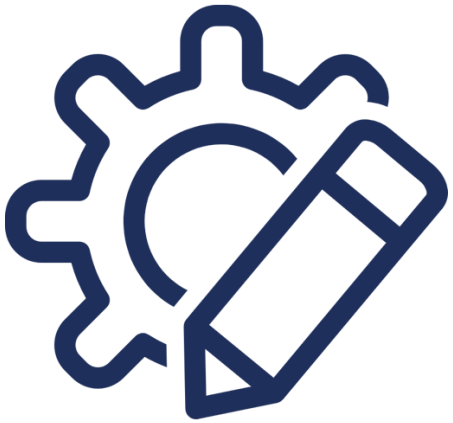
NoSQL databases have varied data structures which is more eloquent for solving development needs than relational data stores.





Non-Relational Databases

Benefits of NoSQL



Specialized Capabilities

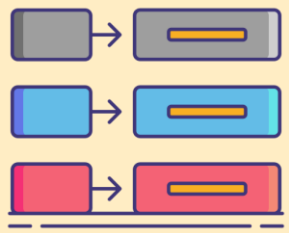
There are also very specialized capabilities that certain NoSQL databases offer such as:

- Specific Indexing and querying capabilities such as Geospatial search
- Data replication robustness
- Modern HTTP APIs

Non-Relational Databases

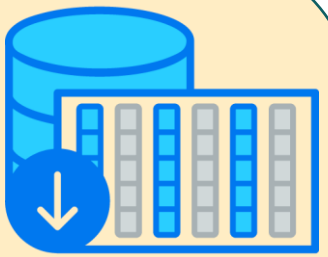
NoSQL

Types based on storing model

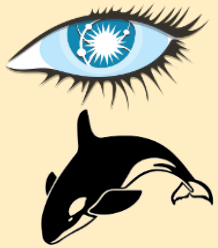


Key-value store

Document based



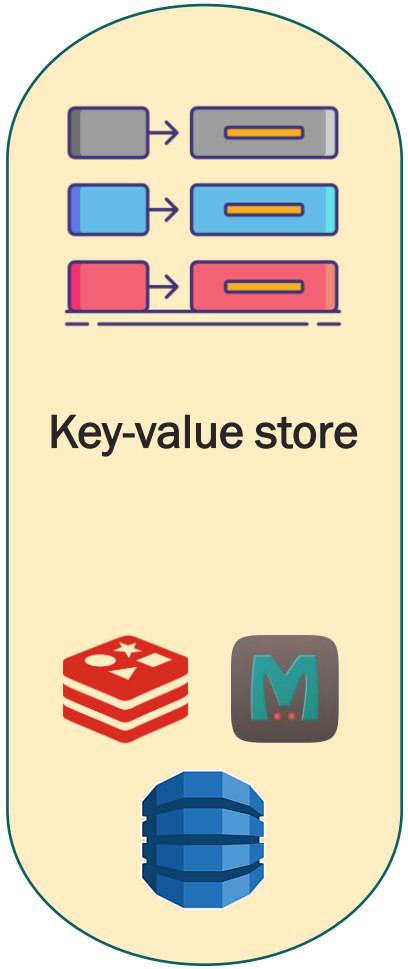
Column based



Graph based

Non-Relational Databases

Key-Value



All data is stored with a key and an associated value blob.

- They are the least complex of NoSQL
- Key-Values are represented as hashmaps
- Ideal for basic CRUD operations
- Easy to scale and shard
- Not intended for complex queries
- Value blobs are opaque to database
- Less flexible data indexing and querying

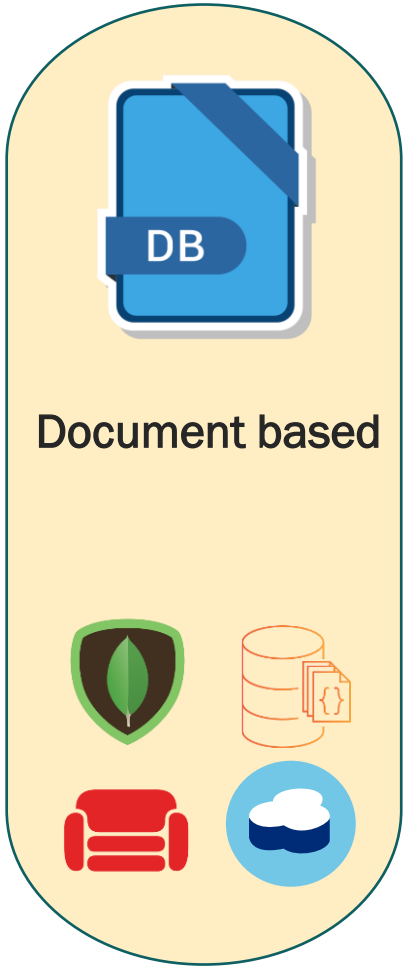
Non-Relational Databases

Key-Value

- Some suitable use cases are:
  - For quick basic CRUD operations on non-interconnected data
  - Storing and retrieving session information for a web application
  - Storing in-app user profiles and preferences
  - Shopping cart data for online stores
- Some unsuitable use cases are:
  - Data is interconnected with many-to-many relationships
  - Social networks
  - Recommendation engine scenarios
  - When high level consistency is required for multi-operation transactions with multiple keys
  - Need a database that provides ACID transactions
  - When apps run queries based on value rather than key
- Examples are:
  - Amazon DynamoDB
  - ORACLE NoSQL Database
  - Redis
  - AEROSPIKE
  - Memcached
  - Project Voldemort

Non-Relational Databases

Document Based



These databases which are based on the key-value databases, make the value visible and able to be queried.

- Each piece of data is considered a document and stored in JSON or XML format
- Each document offers a flexible schema
- No two documents need to be the same or contain the same information
- Contents of the documents can be indexed and queried using key and value range
- Are horizontally scalable
- Allow sharding across multiple nodes
- The most widespread of the NoSQL databases

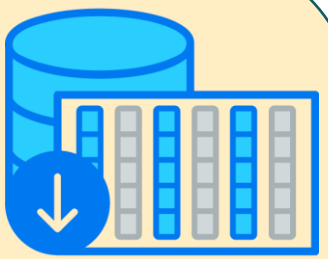
Non-Relational Databases

Document Based

- Suitable use cases are:
  - Event logging for apps and processes (each event instance is represented by a new document)
  - Online blogs (each user, post, comment, like or action is represented by a document)
  - Operational datasets and metadata for web and mobile apps
- Unsuitable use cases are:
  - When you require ACID transactions
  - Document transactions can’t handle transactions that operate over multiple documents
  - If your data is in an aggregate-oriented design
  - If data naturally falls into a normalized tabular model
- Examples are:
  - IBM Cloudant
  - mongoDB
  - CouchDB
  - Terrastore
  - OrientDB
  - Couchbase
  - RavenDB

Non-Relational Databases

Column Based



Column based

Spawned from the architecture that Google created called ‘Bigtable’, also known as Bitable clones or Columnar or wide-column databases.

- Store data in columns or groups of columns
- Column families are several rows each with unique keys or identifier that belong to one or more columns
- Grouped together as they are often accessed together
- Rows in a column family are not required to share any of the same columns
- Can share all, a subset or none
- Columns can be added to any number of rows or not

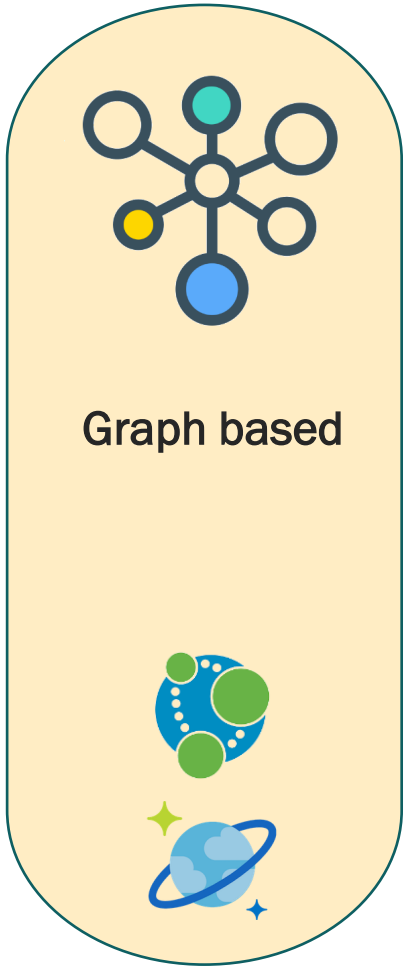
Non-Relational Databases

Column Based

- Suitable use cases are:
  - Great for large amounts of sparse data
  - Better compression therefore more storage save compared with row-based databases
  - Are horizontally scalable
  - Can be deployed across cluster of nodes
  - Can be used for event logging and blogs
  - Counters are a unique use case for column databases (for when application needs an easy way to count or increment as events occur)
  - Columns can have a TTL (Time to leave) parameter: useful for data with an expiration date or time (like trial periods)
- Unsuitable use cases are:
  - When traditional ACID transactions are required
  - In early development query patterns may change and require numerous changes to column-based databases
- Examples are:
  - Cassandra
  - Apache HBASE
  - Hypertable
  - Accumulo

Non-Relational Databases

Graph Based



Store information in entities (or nodes) and relationships (or edges)

- Great when dataset resembles a graph-like data structure
- Not easily horizontally scalable
- Traversing a graph with split nodes across multiple servers can become difficult and hurt performance
- Are ACID transaction compliant

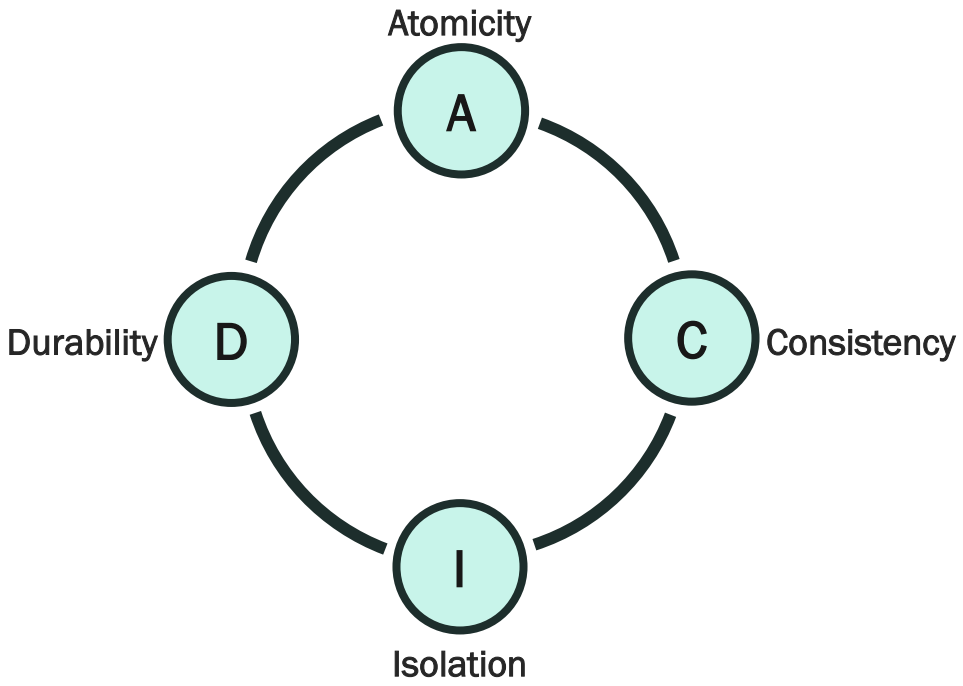
Non-Relational Databases

Graph Based

- Suitable use cases are:
  - For highly connected and related data
  - Social networking (friends, friends of friends, ...)
  - Routing, spatial and map apps (like finding close locations or shortest routes)
  - Recommendation engines (close relationships and links between products)
- Unsuitable use cases are:
  - When there is a need to scale horizontally
  - When trying to update all or subset of nodes with a given parameter
- Examples are:
  - Neo4j
  - OrientDB
  - ArangoDB
  - Amazon Neptune
  - Apache Giraph
  - Janus Graph

Non-Relational Databases

Consistency Models



Relational Databases

Basically Available

BASE

Soft State

Eventually Consistent

Non-Relational Databases

Non-Relational Databases

Consistency Models

- One of the differences between relational databases and non-relational (NoSQL) databases is their data consistency models. As we already know, relational databases use ACID consistency models, while NoSQL databases use BASE consistency models.
- In the NoSQL database world, ACID transactions are less fashionable because some databases have loosened the requirements for immediate consistency, data freshness, and accuracy. They do this to gain other benefits, such as availability, scale, and resilience.
- It mostly used in:
  - Marketing and customer service companies
  - Social media apps
  - Worldwide available online service
  - Such as NETFLIX, Apple, Spotify, Uber
  - They need o be available all time no matter which part of the world
- BASE favors availability over consistency of data.

Non-Relational Databases

Consistency Models

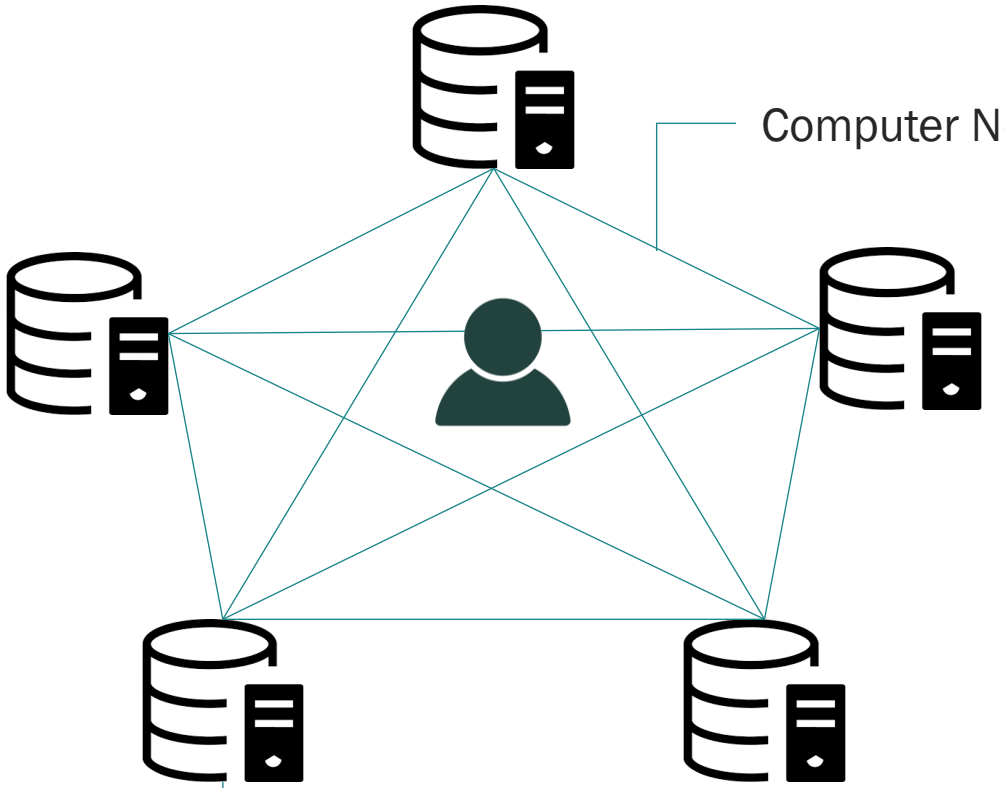
- Basically Available
  - Rather than enforcing immediate consistency, BASE model NoSQL databases ensure availability of data by spreading and replicating it across the nodes of database clusters.
- Soft State
  - Due to the lack of immediate consistency, data values may change overtime. In the BASE model data, stores don’t have to be write-consistent, nor do different replicas have to be mutually consistent all the time.
- Eventually consistent
  - The fact that the BASE model does not enforce immediate consistency does not mean that it never achieves it. However, until it does, data reads might be inconsistent.

Non-Relational Databases

Distributed Databases

Reliability

Availability



Concurrency Control?

Non-Relational Databases

Distributed Databases

- A distributed database is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.
- Is physically distributed across the data sites by fragmenting and replicating the data.
- Follows the BASE consistency model
- Advantages are:
  - Reliability and availability
  - Improved performance
  - Reduced query processing time
  - Ease of growth/scale
    - Can easily increase system capacity by just adding new servers
  - Continuous operations/availability (no reliance on the central site)
- Challenges are:
  - Concurrency control (consistency of data)
    - Solution: either write/read t only a single per fragment of data (data is synchronized in the background)
    - Write to all nodes holding that fragment, reads to a subset of nodes per consistency
  - No transaction support (or very limited)

Non-Relational Databases

Distributed Databases

- Fragmentation
- To store a large piece of data on all the servers of a distributed system, you need to break your data into smaller pieces
- It is also called partitioning or sharding of data by some NoSQL databases
- usually done by the key of the ‘key:value’ record in two ways:
  - By either grouping all keys lexically (e.g. all keys starting with A-C on the same server)
  - By grouping all records that have the same key, and placing them on the same server (e.g all records with key dept_id = 231 on the same server)
- Replication
- Now that data is distributed across the node, how do we make sure that if a node fails, the data in that note is not all lost?
- Replication is basically done for the protection of data against node failures
- All data fragments are stored redundantly in more than one site
- Increases the availability of data different sites
- Disadvantages:
  - Data needs to be consistently synchronized
  - Potential inconsistencies

Non-Relational Databases

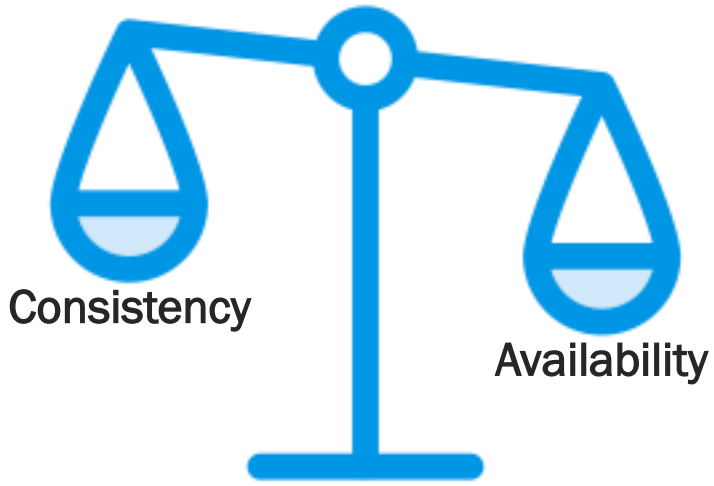
RDBMS vs NoSQL

Relational Databases	Non-Relational Databases
Consistency	High Performance
Structured Data (Fixed Schema)	Unstructured Data (Flexible Schema)
Transactions	Availability
Joins	Easy Scalability

There might be cases in which both relational and NoSQL databases will be needed. If you have too much data and need performance, and need to scale fast, But at the same time, you also need transactions support, and complex joins on your data, then you might think of a combined solution

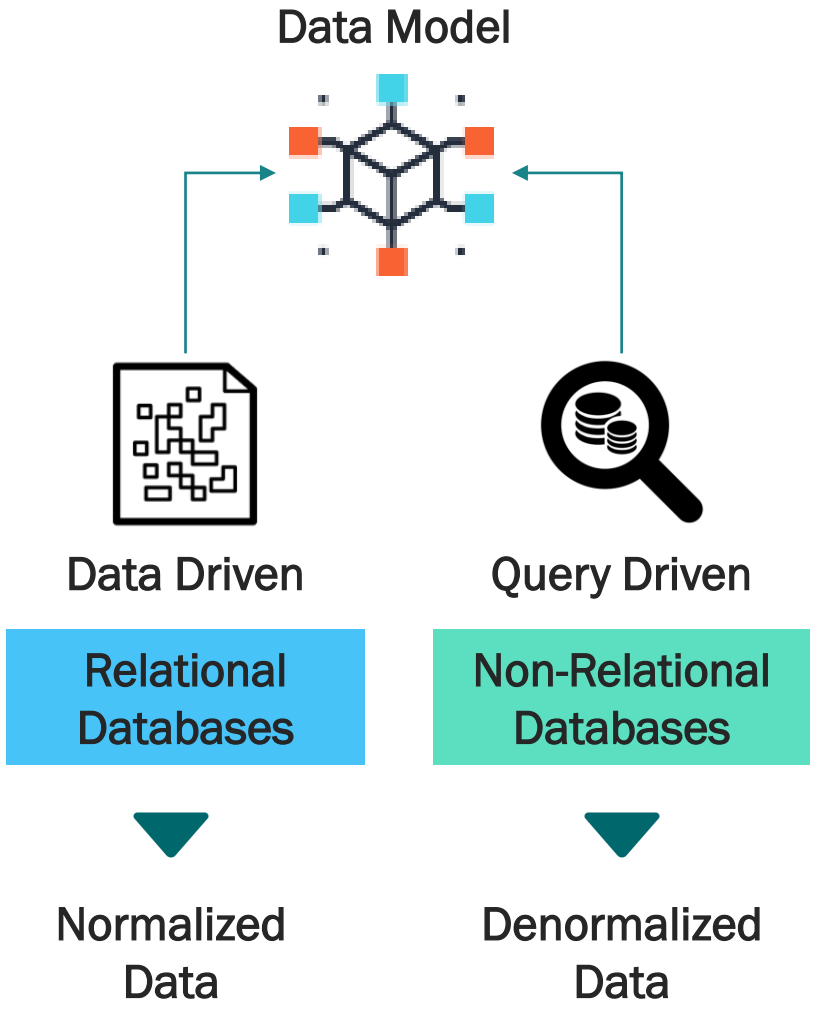
Non-Relational Databases

Migrating from RDBMS to NoSQL



Relational Databases

Non-Relational Databases



Non-Relational Databases

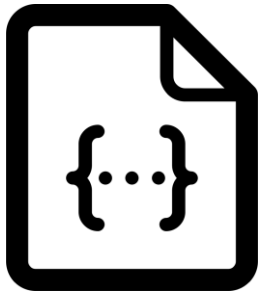
Migrating from RDBMS to NoSQL

- In NoSQL your data model is driven by your queries, not the data itself
  - In a relational world the solution design starts from the data, the entities, and their relationship.
  - In NoSQL it's not the data that drives your data model or schema. It's the way your application accesses the data and the queries you are going to make
  - In NoSQL, models should be based on how the app interacts with the data, rather than how the model can be stored as rows in one or more tables
- in RDBMS data is normalized, while in NoSQL it is denormalized
  - With NoSQL, starting from your query means that you will structure your data on disk accordingly. Thus, you may need to store the same data in different models just to answer the question
  - This will lead to data denormalization while data in RDBMS is normalized.
- Availability vs Consistency
  - sometimes services require availability more than consistency. When both availability and performance are needed (thus distributed systems), consistency cannot be ensured
  - Many of today's online services value availability more than consistency
- NoSQL databases are not designed to support transactions, or joins, or complex processing, except in limited cases

MongoDB

MongoDB

Structure



Document

Equivalent of a row in RDBMS

Collection

Equivalent of a table in RDBMS

Database

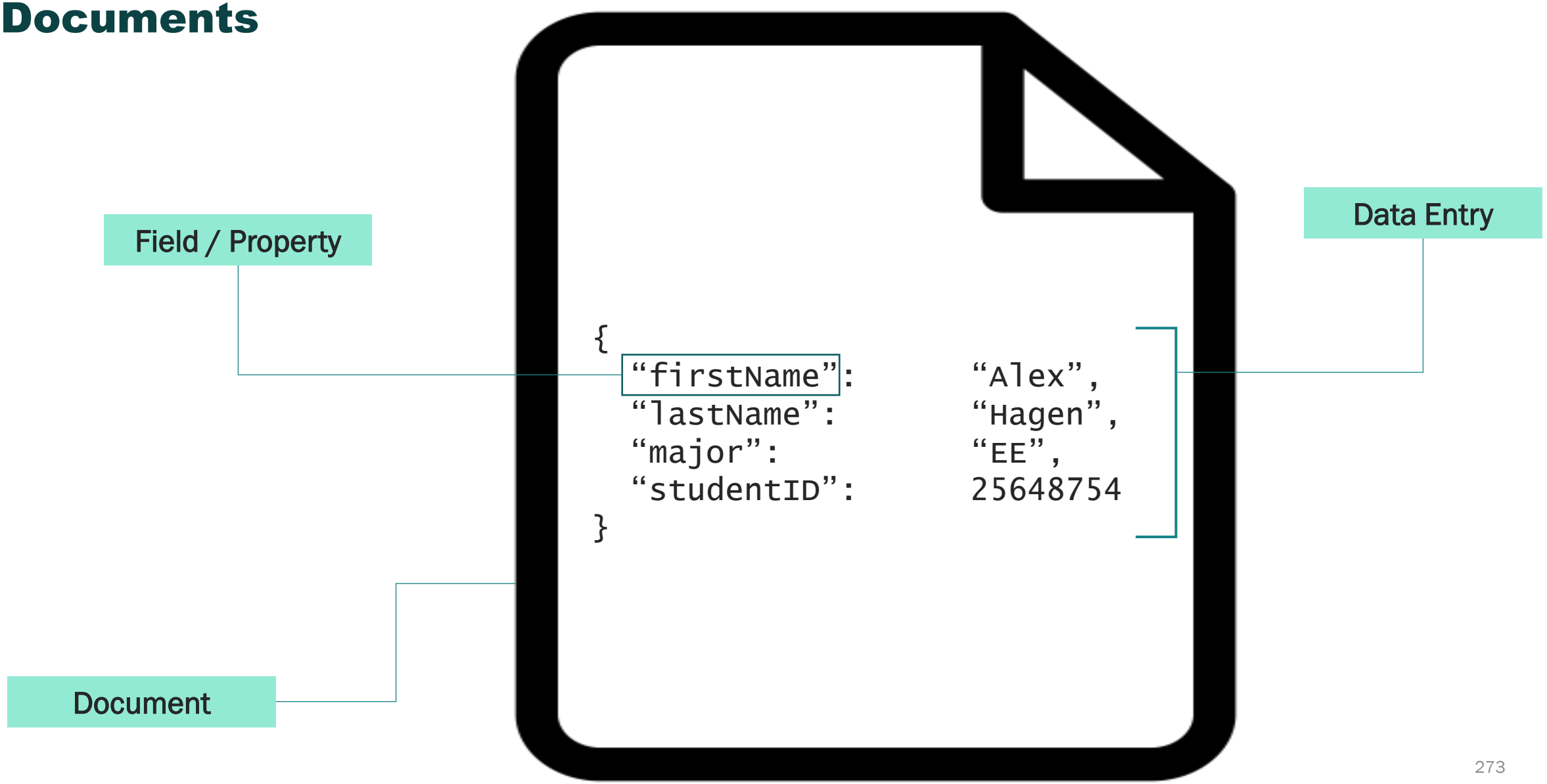
MongoDB

Structure

- It is a document and NoSQL database.
  - Each record is a document
  - Data is stored in a non-relational way
- Documents are associative arrays like JSON objects or Python dictionaries
- Collection: A group of stored documents.
  - For example all student records in Students collection.
  - Or for example all staff records are stored in employee collection
- Database in MongoDB is where all collections are stored.
  - MongoDB allows to model data as you read/write while traditional databases require you to cerate schema first and then tables
    - Adding or removing fields are easy
  - Allows for structured/unstructured data
  - High availability by keeping multiple copies of data

MongoDB

Documents



MongoDB

Documents

```
{
  “firstName”:          “Alex”,
  “lastName”:           “Hagen”,
  “dateOfBirth”:        ISODate(“2003-03-24T6:32:00”),
  “major”:              “EE”,
  “studentID”:          25648754,
  “enrolled”:           true,
  “tuition”:            24350.00,
  “address”:            {
                          “street”:  “8213 Ranchview Dr”,
                          “city”:    “Dallas”,
                          “zip”:     75254
                        },
  “GPA”:                3.57,
  “misc”:               [“ABC”, 234.5, “Max”, 65]
}
```

Embedded sub-document

List of values with mixed types

MongoDB

Documents

- Documents are associative arrays like JSON objects or Python dictionaries
  - Example: a student document
  - First name, last name, … are fields or properties.
  - MongoDB supports a variety of data types
- MongoDB allows for sub-documents to group secondary information together
- MongoDB allows lists of values with different types mixed together

MongoDB

Advantages

- Flexibility with Schema

```
{
  “street”:   “8213 Ranchview Dr”,
  “city”:     “Dallas”,
  “zip”:      75254
}
```

```
{
  “street”:     “5687 East St”,
  “city”:       “London”,
  “postcode”:   “W1 2LS”
}
```

- Code First Approach

```
{



}
```

```
{
  “street”:     “5687 East St”,
  “city”:       “London”,
  “postcode”:   “W1 2LS”
}
```

MongoDB

Advantages

- **Evolving Schema** (Dynamic Schema)

```
{
  “street”:   “8213 Ranchview Dr”,
  “city”:     “Dallas”,
  “zip”:      75254
}
```

```
{
  “street”:     “5687 East St”,
  “city”:       “London”,
  “postcode”:   “W1 2LS”,
  “contactlessDelivery”: true
}
```

- **Supports unstructured Data** (Data with different structures can be stored in the same collection)

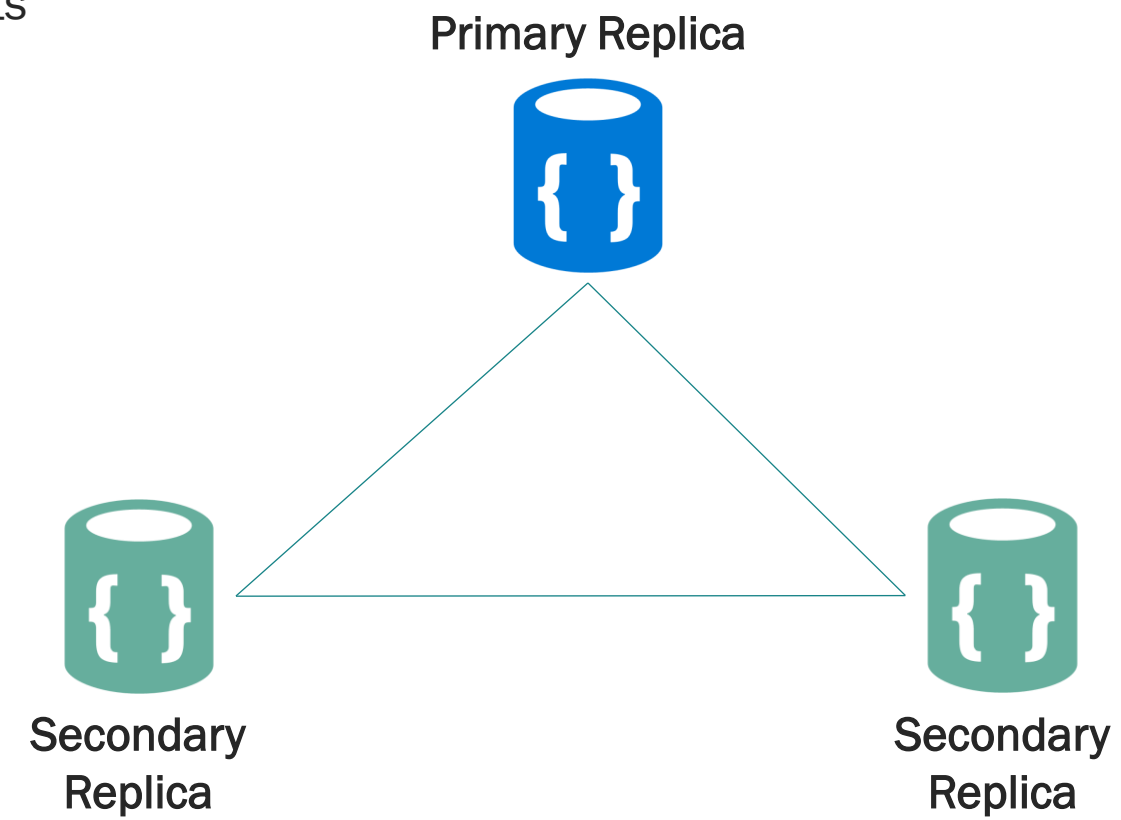
```
{
  “symbol”:          “IBM”,
  “open”:            459.65,
  “high”:            468.25,
  “low”:             454.75
}
```

```
{
  “stockname”:       “IBM”,
  “pricing”: {
                “o”:   459.65,
                “h”:   468.25,
                “l”:   454.75
}
```

MongoDB

Advantages

- **High Availability**
  Three-node replica sets



MongoDB

Advantages

- High availability
- MongoDB provides high availability by means of redundancy.
  - Typical MongoDB setups are three-node replica sets where one of them is a primary member and the others are secondary members.
  - Replications keeps a copy of data on other data nodes in the cluster.
  - There is no system maintenance downtime
  - No upgrade downtime

MongoDB

Use Cases

Internet of Things

E-Commerce

Real-Time Analytics

Gaming

Finance

MongoDB

Use Cases

- MongoDB allows you to bring data from different sources into it (Many sources – one view)
  - So instead of each part of your data living in silos, you can ingest multiple shapes and formats of data into MongoDB
  - No more data silos
  - Easy data ingestion
  - Consolidate different data
  - All because of flexible schema
- Internet of Things
- E-commerce
  - Products sold online have different attributes
- Real-time analytics
  - Quick response to change
  - Simplified ETL
  - Real-time along with operational data
- Gaming
  - Globally scalable
  - No downtime
  - Supporting rapid development
- Finance
  - Speed
  - Security (by encryption)
  - Reliability