Data, Data Storage, Data Collection

Lecture 7: Data Storage

Romain Pascual

MICS, CentraleSupélec, Université Paris-Saclay

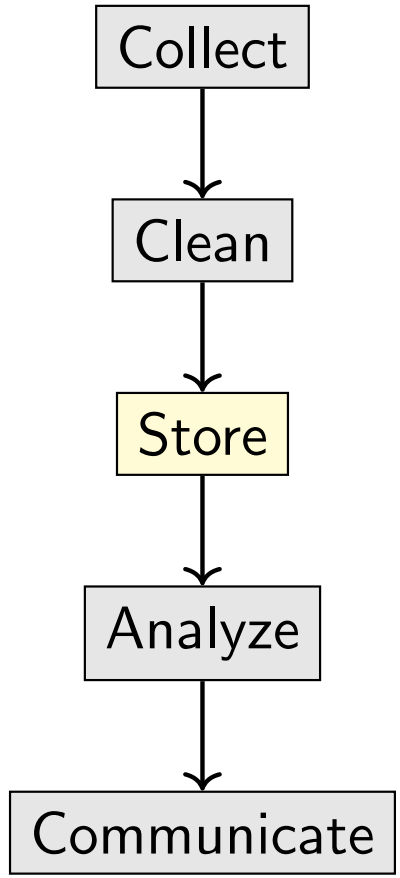
Recap

Recap: Data Wrangling

1. Start with initial assessment of the data quality
2. Handle missing data and outliers
3. Transform the data to prepare it for analysis
4. Reshape the data for better readability or comparability with specific analysis techniques.
5. Keep the original data and document the cleaning steps

Introduction

Within the lifecycle



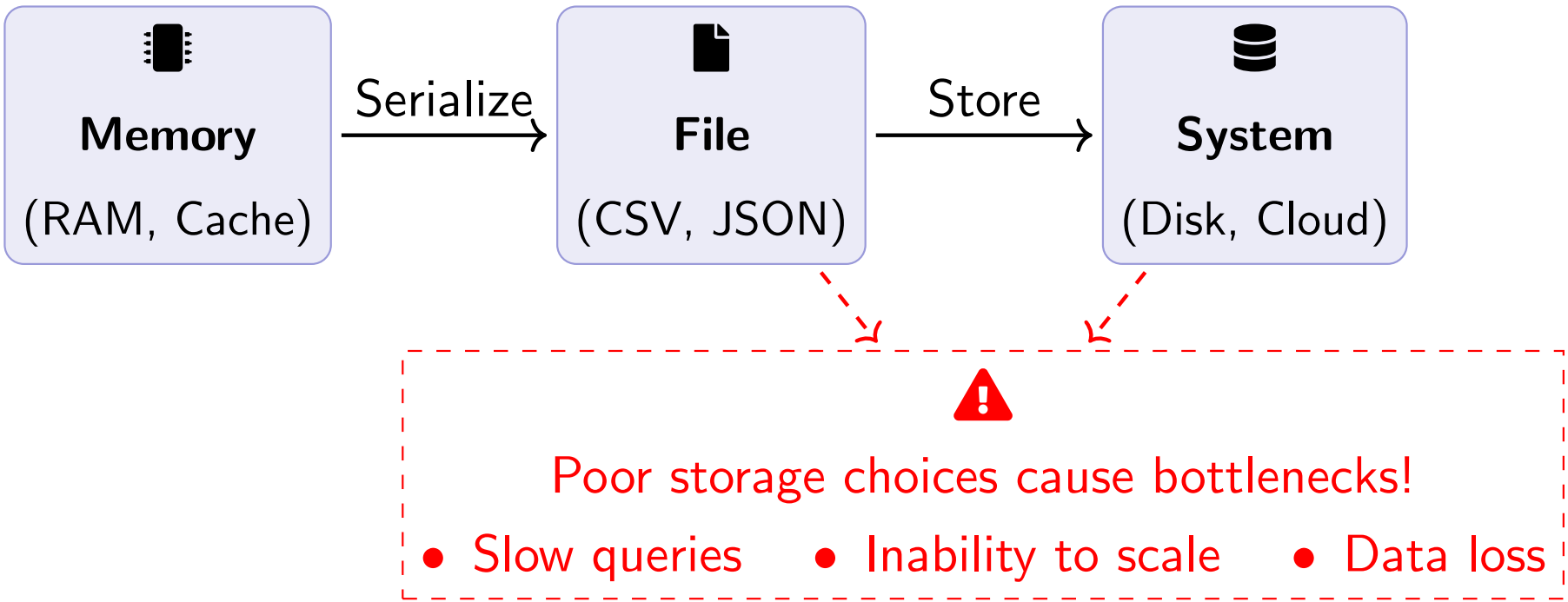
Session Objectives

At the end of this session, you should be able to:

- Distinguish between storage structures (flat files, tabular, hierarchical, columnar).
- Compare formats (CSV, JSON, Parquet) based on trade-offs (readability, size, speed, interoperability).
- Explain the differences between SQL, NoSQL, and cloud storage use cases.
- Apply a decision framework to select appropriate storage for a given scenario.
- Discuss the broader implications of storage choices (e.g., ethics, sustainability)

Why Storage Matters?

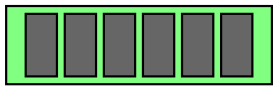
After wrangling, we need to store the data for analysis

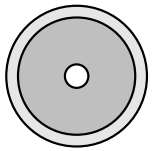


We discuss the logic behind data storage, not just the technologies.

Storage Persistence, Strategies

In-Memory vs On-Disk





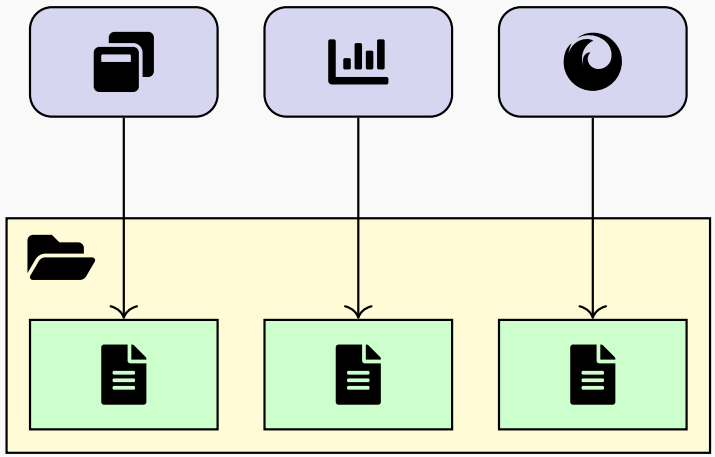
	In-Memory (RAM)	On-Disk (HDD/SSD)
Speed	10-100 ns	0.1-10 ms
Persistence	Volatile	Persistent
Cost (2025)	$5-$10 per GB	$0.02-$0.10 per GB
Capacity	GBs	TBs

```python
# In-Memory
data = {"key": "value"}  # Stored in RAM

# On-Disk
import pickle
pickle.dump(data, open('data', 'wb'))  # Save to disk
data=pickle.load(open('data', 'rb'))  # Load from disk
```

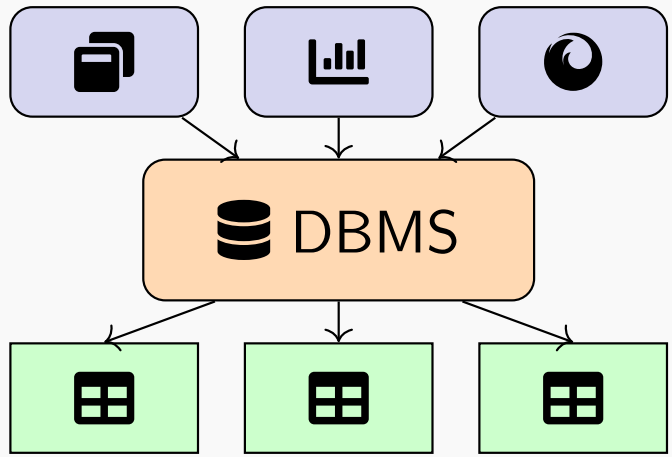
Filesystems vs. Database Management Systems

FMS



- OS method to organize data files
- Simple folder/file hierarchy
- Keep redundant data
- Low complexity
- No built-in querying capabilities

DBMS



- Software for accessing, and manipulating data
- Structured tables/collections
- Eliminates redundant data
- High complexity
- Supports complex queries and indexing

Storage Structures and Formats

Data Storage Models

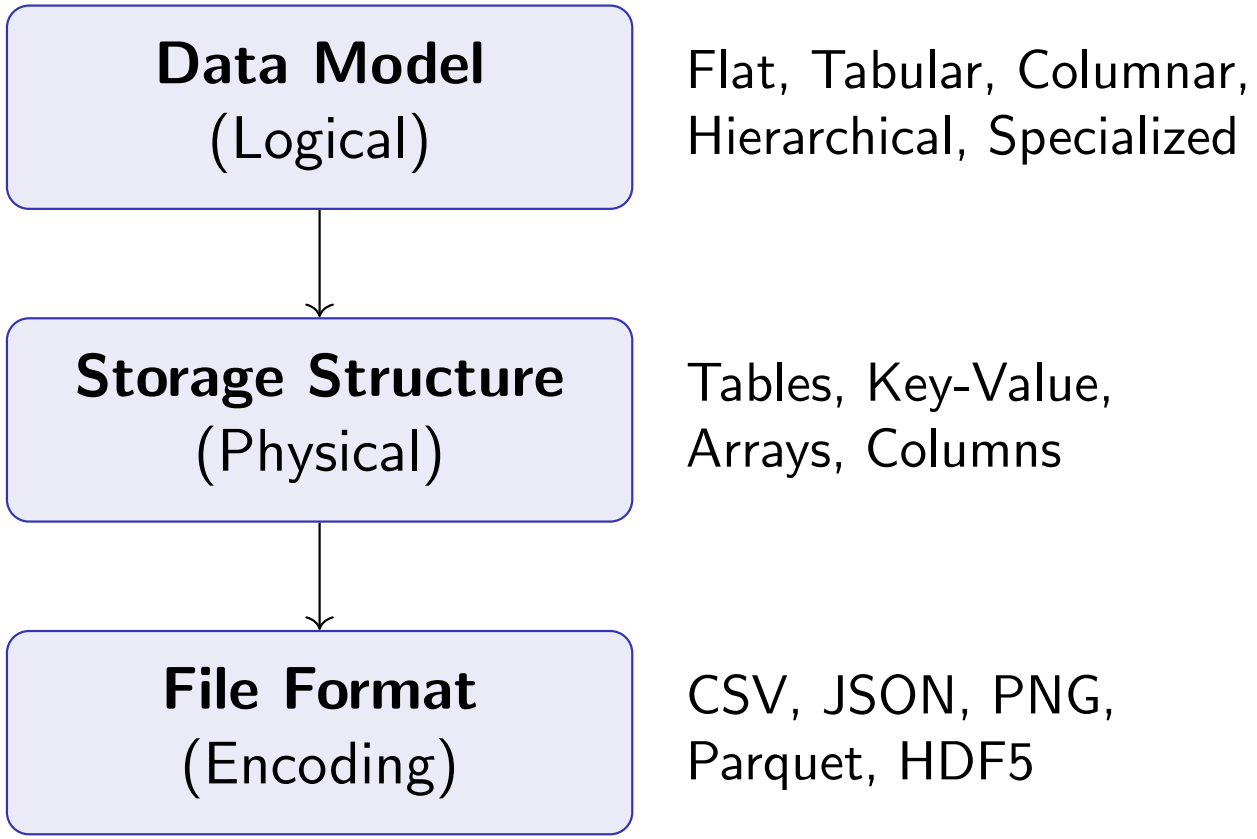
Definition (Storage Model)

Conceptual structure or paradigm for organizing data, defining how it is logically stored and accessed.

- **Flat files**: Simple textual files
- **Tabular**: Organised in rows and columns
- **Hierarchical**: Nested key-value pairs
- **Columnar**: Data stored by columns
- **Specialized**: Domain-specific formats

This is about the structure of the data.

From Models to Formats



- **Data model** defines the logical structure.
- **Storage Structure** defines the physical organization.
- **File format** defines the encoding.

Excel (Tabular)

Characteristics:

- Widely used in business environments
- Proprietary format (.xlsx)
- 1 048 576 rows $\times$ 16 384 columns per sheet
- Supports formulas, charts, and multiple sheets
- User-friendly interface

Limitations:

- Not ideal for big data or automation
- Compatibility issues between versions

Tabular File Format Comparison

Excel

Characteristics

- Business standard (.xlsx)
- 1 048 576 rows $\times$ 16 384 columns
- Formulas, charts, multiple sheets
- User-friendly interface

Limitations

- Not for big data
- Version compatibility issues

CSV

Characteristics

- Universal, simple format
- Human-readable
- Works with almost any tool or programming language

Limitations

- No data types (everything is a string)
- Easy to break
- No complex structures

When Excel is not the right tool...

Grille d'auto-évaluation par compétence

étudiant :

date

activité pédagogique : Stage de fin d'études

COMPETENCE Ci	QUESTIONS CLES D'EVALUATION PAR Ci : pour chaque Ci : Est-ce que l'élève (ou l'équipe) a su…	Points forts	Points d'amélioration	PASS /FAIL
C1-complexité	Analyser, concevoir et réaliser des systèmes complexes			
Analyser un système dans sa globalité, identifier ses dimensions scientifiques, économiques, humaines, etc., modéliser avec l'échelle et les hypothèses pertinentes, résoudre le problème, et concevoir tout ou partie d'un système complexe.	* analyser le système dans sa globalité, identifier les disciplines mises en jeu, caractériser les interactions internes du système, et ses dimensions extérieures ? * utiliser un modèle adapté, avec une échelle de modélisation adéquate, des hypothèses pertinentes, et argumenter le choix de son modèle ? * résoudre un problème avec une pratique de l'approximation, simulation, observation et expérimentation, en sachant identifier les incertitudes et prendre du recul sur les résultats obtenus ? * concevoir, spécifier un cahier des charges, réaliser, tester et valider tout ou partie d'un système complexe, en sachant adapter sa démarche de manière itérative selon les résultats intermédiaires ? * globalement prendre du recul et faire un retour d'expérience sur sa démarche ?			
C2-métier ingénieur	Développer ses compétences dans un domaine d'ingénieur et dans un métiers			
L'approfondissement disciplinaire ou sectoriel doit permettre au diplômé d'aller au fond d'un domaine, de réfléchir de manière profonde, de comprendre les difficultés et les subtilités d'un sujet. Cette approche de type « expertise », permet de s'approprier des modes de pensée complexe qui peuvent être ensuite transposés à d'autres secteurs, elle donne du recul pour aborder efficacement de futurs apprentissages. 2 axes sont visés : le secteur d'activité, et la ligne métier transverse.	* développer son expertise dans un domaine des sciences de l'ingénieur ? * identifier les domaines connexes au sujet de son travail, en exploiter les connaissances, intégrer les contraintes spécifiques, et ainsi enrichir son approche ? * détecter et acquérir de manière autonome les nouvelles connaissances et savoir-faire nécessaires, dans son domaine d'ingénieur, pour le problème traité ? * bâtir et suivre une démarche scientifique et technique d'investigation d'un problème (état de l'art, collecte et analyse de données, modélisation, expérimentation, prise de recul, etc.) dans son domaine d'ingénierie et la rendre réutilisable ? * accroître savoir-faire et savoir-être en interagissant avec le milieu professionnel ?			
C3-innover entreprendre	Agir, entreprendre, innover en environnement scientifique et technologique			
La compétence entrepreneur se comprend dans un sens large (entrepreneur de soi-même, capacité d'action, intrapreneur, créateur d'entreprise, etc.) : observer et s'autoriser à critiquer le monde tel qu'il est, remettre en cause ses hypothèses de départ, proposer des alternatives intégrant risques et incertitude, mettre en œuvre concrètement des idées novatrices, industrialisation pour délivrer des résultats tangibles.	* questionner la formulation du problème posé pour comprendre le besoin sous-jacent dans un contexte plus large ? * proposer des idées nouvelles pour répondre au problème et/ou quant à la démarche à suivre pour en améliorer l'efficacité ? * commencer, sur ces idées nouvelles, à essayer de valider 1/ la faisabilité de la solution (technique, économique, humaine, éthique) 2/ l'intérêt réel pour le bénéficiaire ? * regarder si ce problème a été résolu ailleurs, et de quelle manière ? (ne pas réinventer la roue et se positionner par rapport à d'autres solutions)			
C4-création valeur	Avoir le sens de la création de valeur pour son entreprise et ses clients			
Partant d'une définition pour l'entreprise d'augmentation de la productivité et de la rentabilité, la Création de Valeur est élargie ici à l'apport de progrès et d'amélioration dans des domaines comme l'efficacité d'un processus, la performance ou la fiabilité d'une solution technique, ou l'impact environnemental ou sociétal ; de telle sorte que le (ou les bénéficiaires), individuels ou collectifs (entreprises, organisations, société) puissent en constater et en mesurer les effets.	* comprendre, par un dialogue pertinent avec le demandeur (ou client), les enjeux de la problématique, identifier les éventuelles autres parties prenantes et le cadre dans lequel il doit créer la valeur ? * reformuler simplement la création de valeur attendue par le demandeur, obtenir son approbation sur cette nouvelle formulation et convenir avec lui de la façon de la mesurer ? * proposer et présenter de manière convaincante une solution conforme à la reformulation du besoin du demandeur (voire plusieurs solutions) ? * évaluer cette solution en fonction des indicateurs de création de valeur convenus ?			

JSON: JavaScript Object Notation (Hierarchical)

Characteristics

- Standard for API and config.
- Nested key-value structure
- Human-readable format

Limitations

- Verbose syntax
- No support for comments
- Inefficient for large datasets

```
{
  "title": "The Hitchhiker's Guide to the Galaxy",
  "author": "Douglas Adams",
  "year": 1979,
  "characters": [
    {"name": "Arthur Dent", "species": "Human", "role": "Protagonist"},
    {"name": "Ford Prefect", "species": "Betelgeusian", "role": "Researcher"},
    {"name": "Marvin", "species": "Robot", "role": "Android"}
  ],
  "notable_quote": "Don't Panic.",
  "answer_to_life": 42,
  "adaptations": ["radio", "TV", "film"]
}
```

Parquet: (Columns)

Characteristics:

- Columnar format optimized for big data
- High performance compression and encoding schemes
- Efficient for analytical queries
- Works well with Spark, Hadoop, and other big data tools

Limitations:

- Binary format (not human-readable)

Row vs. Column Storage

Name	Age	City
Alice	28	Paris
Bob	32	London
Charlie	25	Berlin

Row Storage	
Row 1	Alice
	28
	Paris
Row 2	Bob
	32
	London
Row 3	Charlie
	25
	Berlin

Column Storage	
Name	Alice
	Bob
	Charlie
Age	28
	32
	25
City	Paris
	London
	Berlin

Row-based access
reads entire row

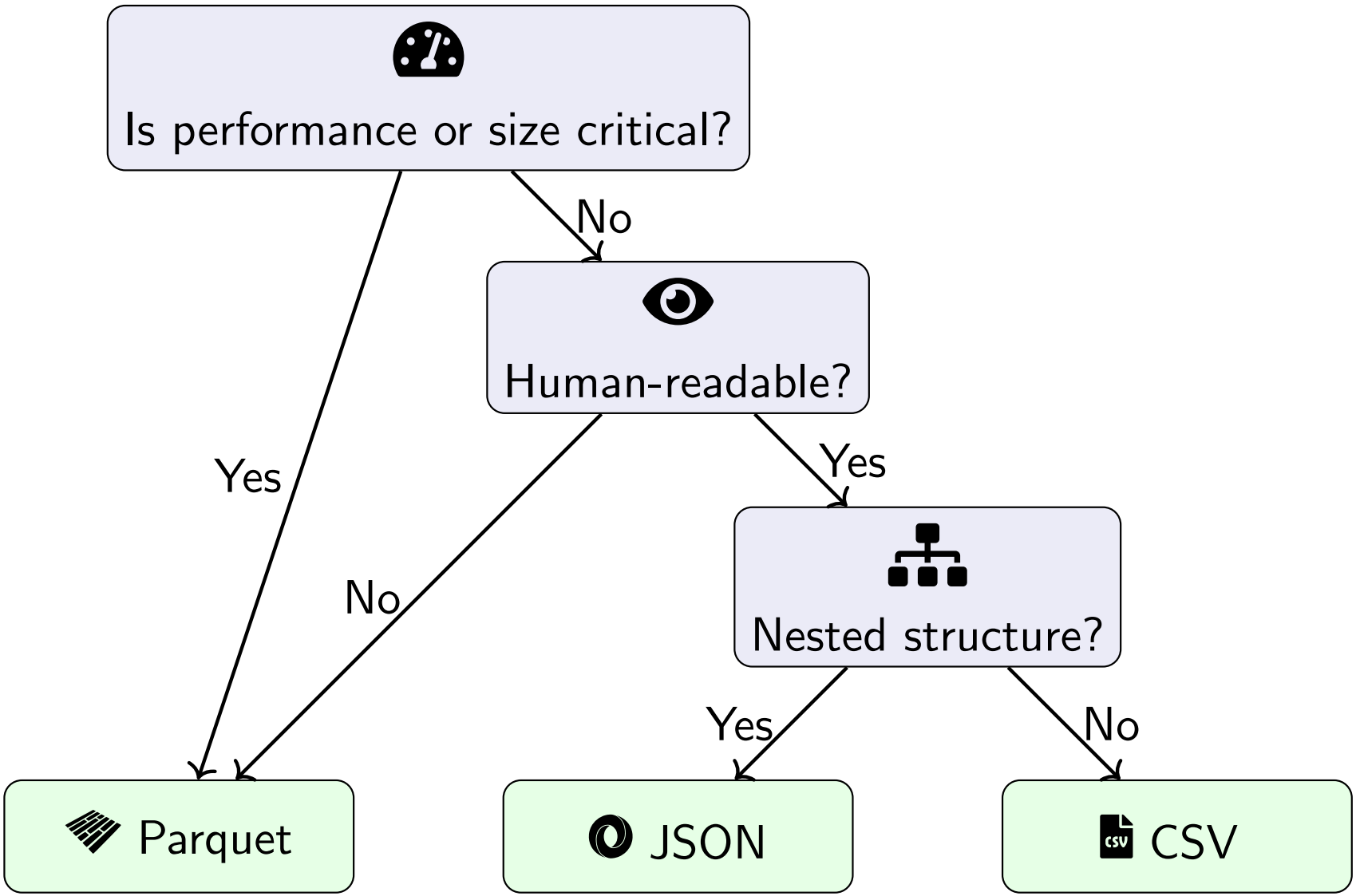
Column-based access
reads only needed columns

Summary: Comparing Common File Formats

Criterion	Excel	CSV	JSON	Parquet
Fast to read/write	✗	●	●	✓
Space-efficient	●	✗	✗	✓
Compatibility	✗	✓	✓	✗
Human-readable	✓	✓	✓	✗
Supports nested data	●	✗	✓	✗
Good for sharing	●	✓	✓	●
Good for analysis	✓	●	✗	✓

Excel is user-friendly but proprietary,
CSV and **JSON** are widely compatible and flexible,
Parquet is optimized for large-scale analytics.

Choosing the Right File Format



Databases

SQL: Relational Databases

Key Concepts

- Tables with rows and columns
- Keys (primary, foreign)
- **A**tomicity, **C**onsistency, **I**solation, **D**urability
- Underlying formalization: Relational algebra

users

id	name	age	email
1	Alice	37	alice@example.com
2	Frank	22	frank@example.com
3	Zoe	21	zoe@example.com
4	David	20	david@example.com
5	Eve	30	eve@example.com
6	Bob	19	bob@example.com
7	Grace	27	grace@example.com
8	Charlie	32	charlie@example.com

Input

```sql
SELECT name, email
FROM users
WHERE age > 29
ORDER BY name;
```

Result

name	email
Alice	alice@example.com
Charlie	charlie@example.com
Eve	eve@example.com

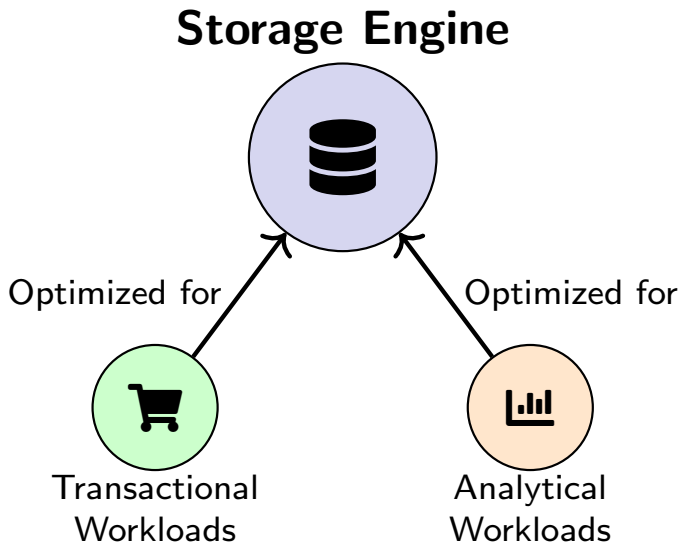
Understanding Storage Engines

You will not build a storage engine from scratch . . .

. . . But you will need to

- Select the right engine.
- Tune it for your workload.

So you need to understand what is under the hood!



In particular, storage engines are optimized for different workloads

Databases: OLTP vs. OLAP

OLTP (Online **Transaction** Processing)

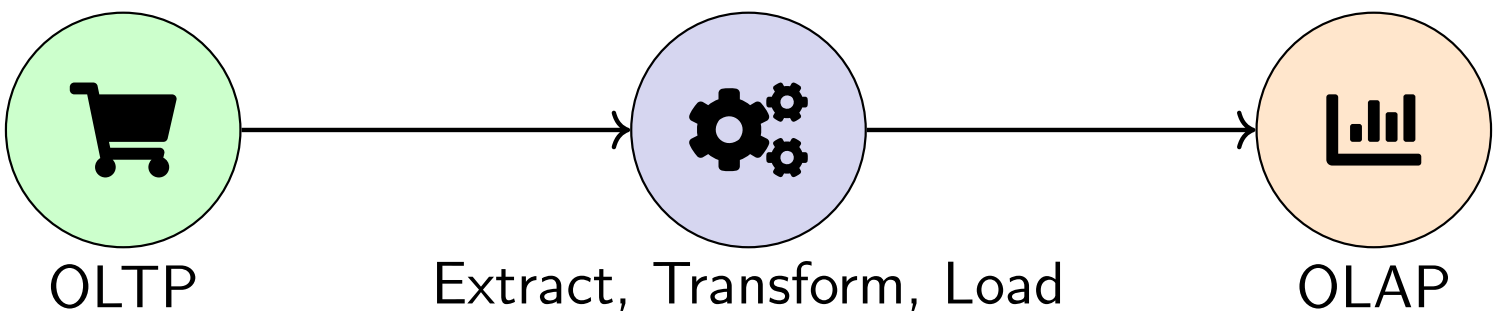
- Fast, frequent reads/writes
- Small, simple transactions
- Indexes for quick lookups

Example: Banking transactions

OLAP (Online **Analytical** Processing)

- Complex queries and aggregations
- Large datasets, few columns
- Calculates statistics

Example: Business intelligence



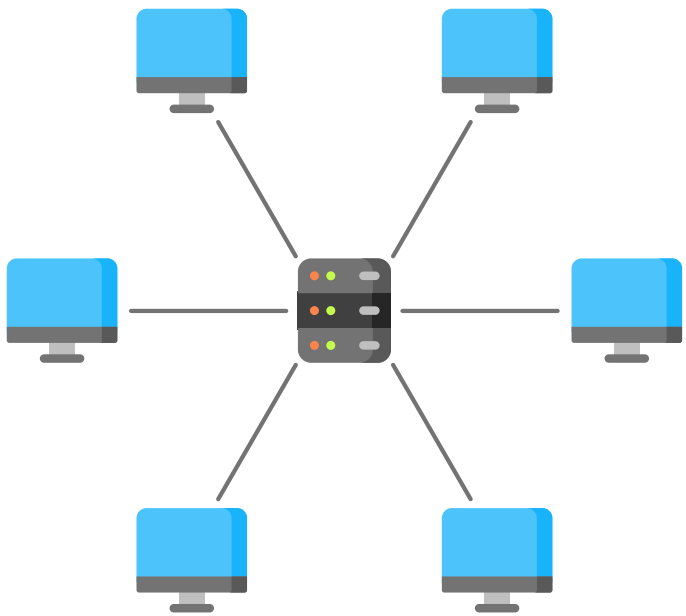
NoSQL: Beyond Relational

The Rise of NoSQL: Volume, Velocity, Variety

- Volume: Massive data growth
- Velocity: High-speed data generation
- Variety: Diverse data types

The Rise of NoSQL: Volume, Velocity, Variety

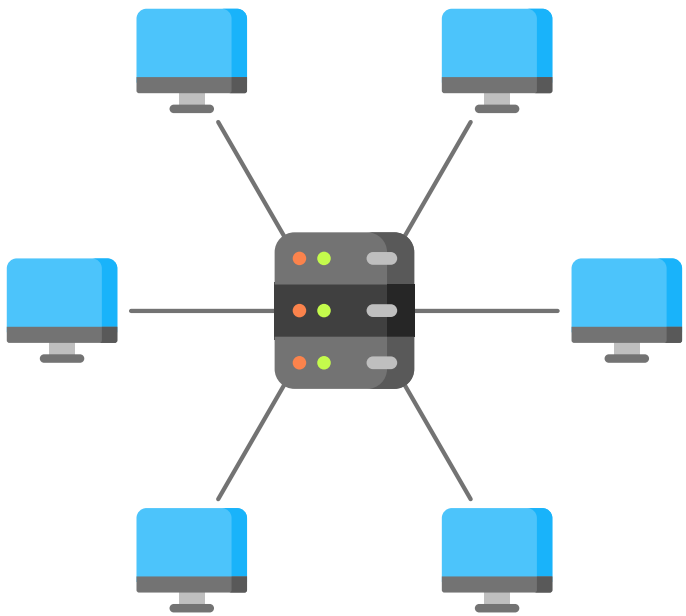
- Volume: Massive data growth
- Velocity: High-speed data generation
- Variety: Diverse data types



Centralized

The Rise of NoSQL: Volume, Velocity, Variety

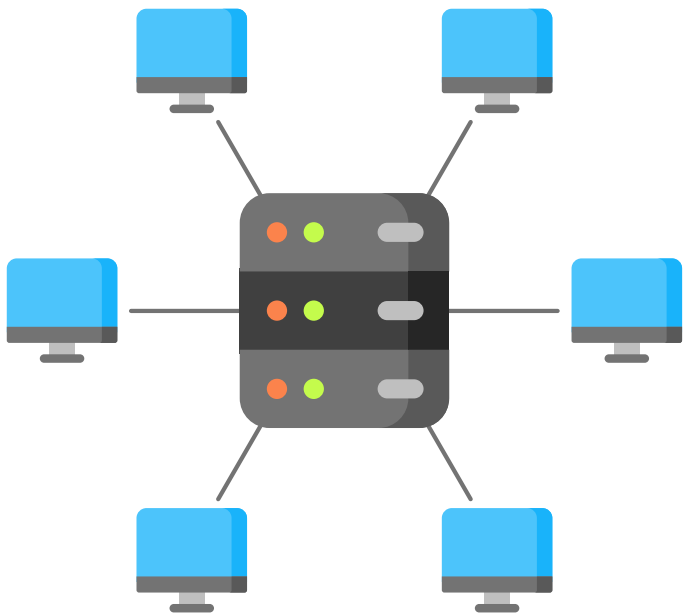
- Volume: Massive data growth
- Velocity: High-speed data generation
- Variety: Diverse data types



Centralized

The Rise of NoSQL: Volume, Velocity, Variety

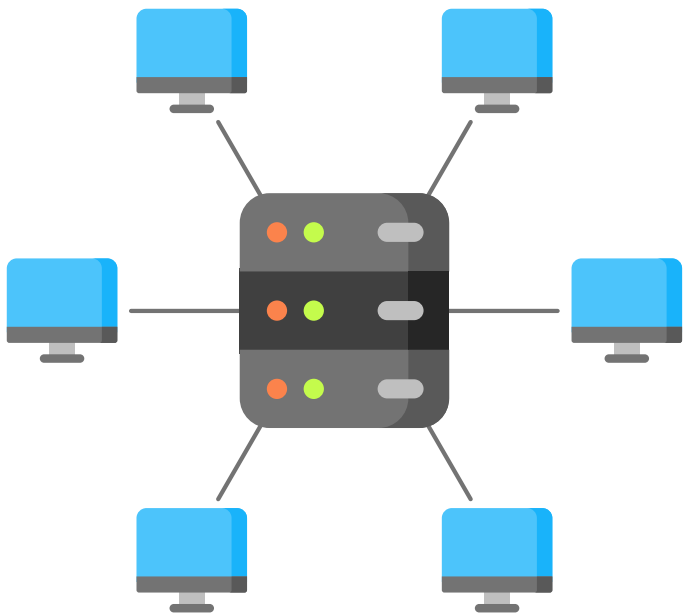
- Volume: Massive data growth
- Velocity: High-speed data generation
- Variety: Diverse data types



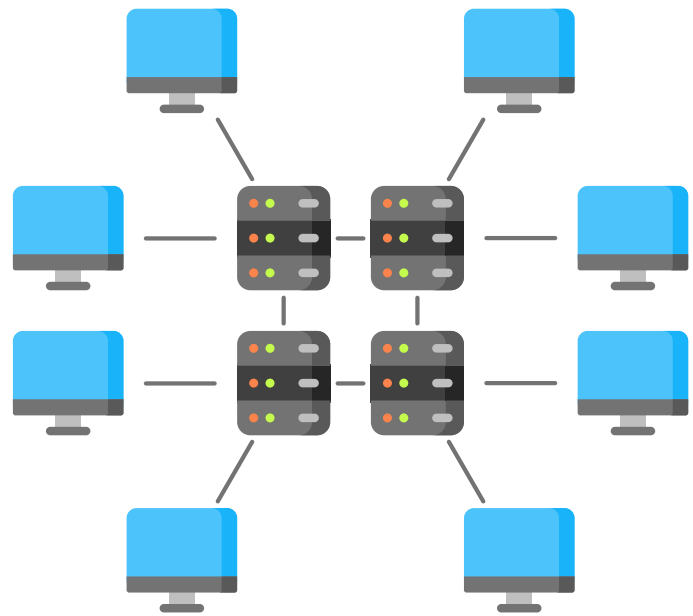
Centralized

The Rise of NoSQL: Volume, Velocity, Variety

- Volume: Massive data growth
- Velocity: High-speed data generation
- Variety: Diverse data types



Centralized



Distributed

The Evolution of Database Systems: From SQL to NoSQL

Historical Context

- **Pre-SQL Era**: Early databases (1960s-70s) were often hierarchical or network models without standardized schema
- **1970**: Edgar F. Codd proposed the relational model
- **early 1970's**: SEQUEL (Structured English Query Language), designed by IBM
- **1979**: First SQL database (Oracle) commercialized
- **1980s-90s**: SQL became the dominant standard for structured data

The Rise of "Not Only SQL"

As a response to the three 3 V's

- **Architecture Shift**: From centralized to distributed systems
- **Data Shift**: Unstructured or rapidly changing data

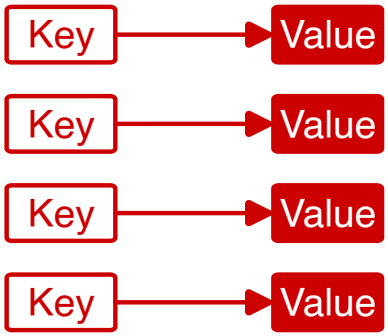
BASE Principles (vs. ACID)

- **B**asically **A**vailable: System appears to work most of the time
- **S**oft state: Data may change over time without explicit updates
- **E**ventual consistency: Data will be consistent... eventually

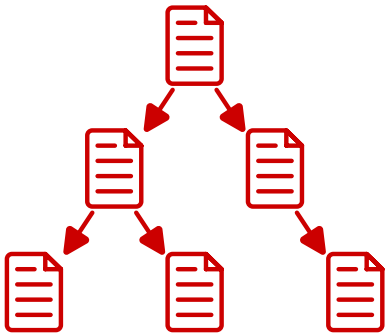
Why NoSQL?

- Better suited for large-scale, distributed data
- More flexible schema design
- Horizontal scalability
- Optimized for specific data models and access patterns

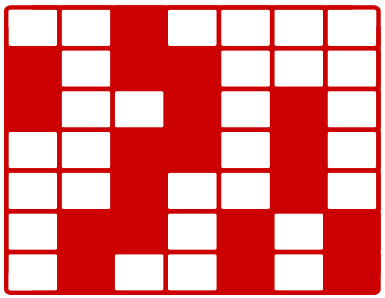
Four families of NoSQL



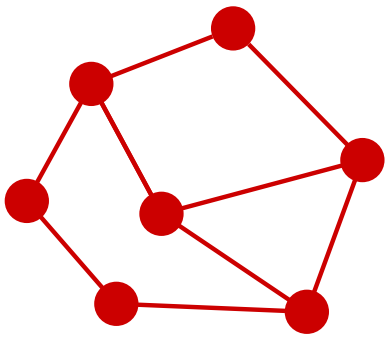
Key-Value
Simple, fast lookups



Document
Nested KV pairs

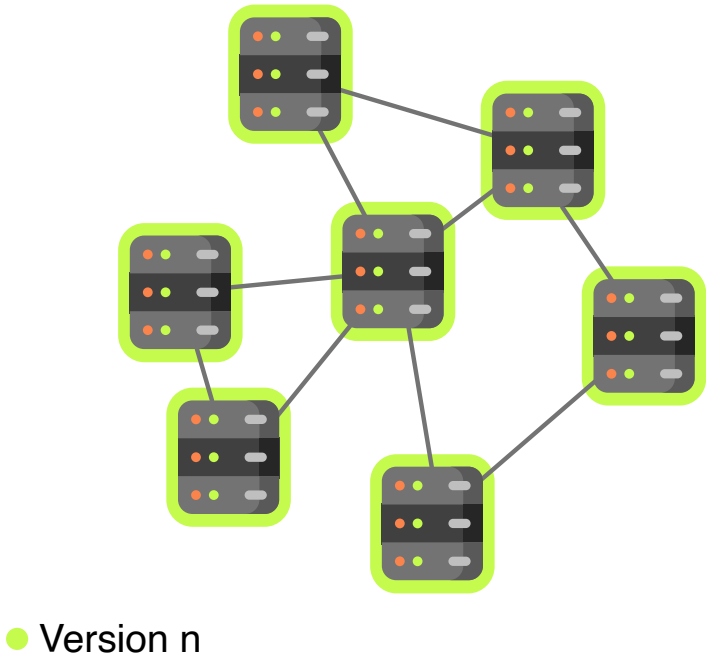


Column
Optimized for analytics

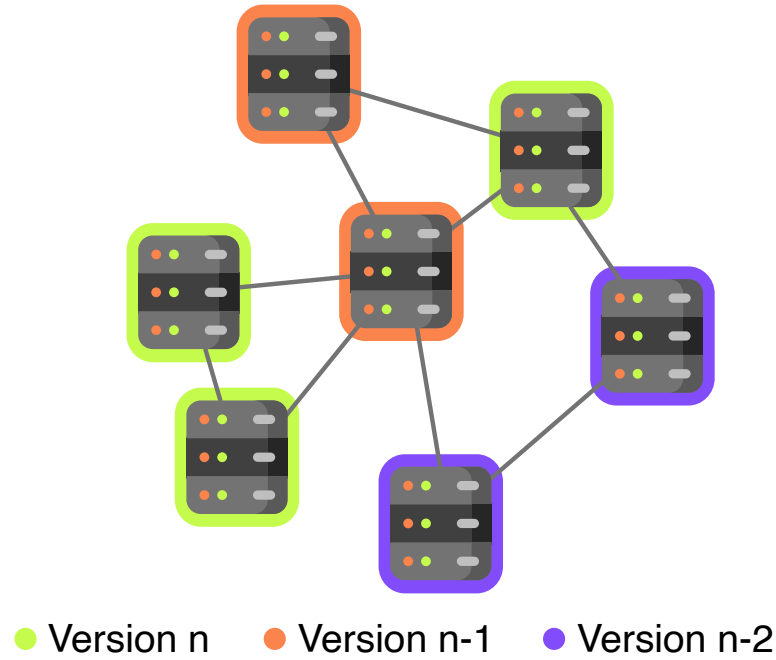


Graph Relationships

The CAP Theorem and its Implications



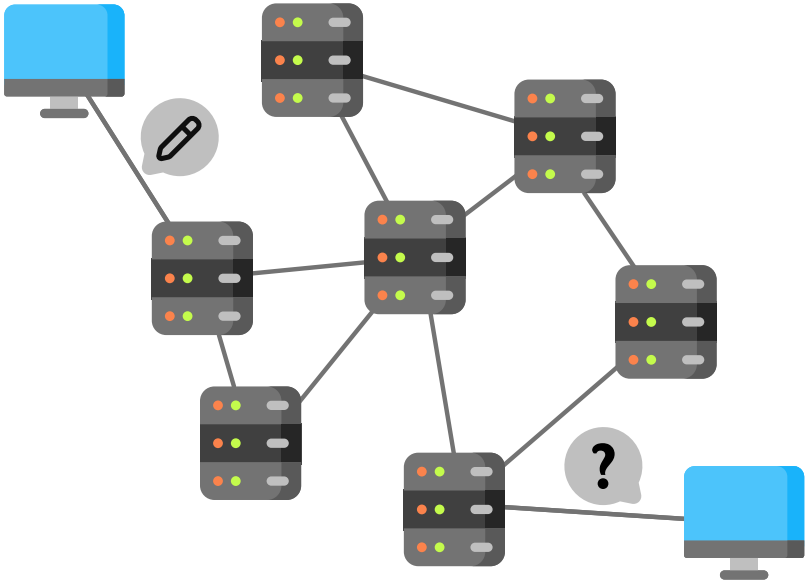
Consistent



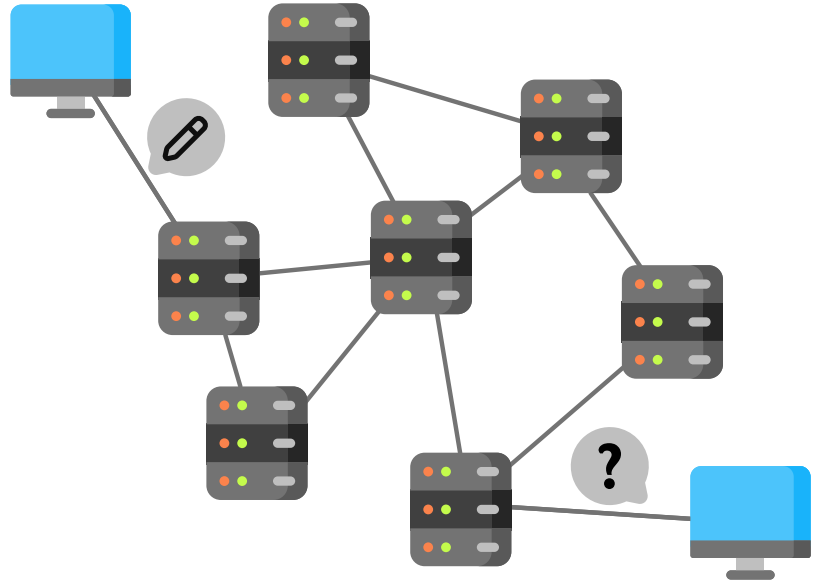
Inconsistent

Consistency (C):

- Every read receives the most recent write (or an error).
- All nodes see the same data at the same time.



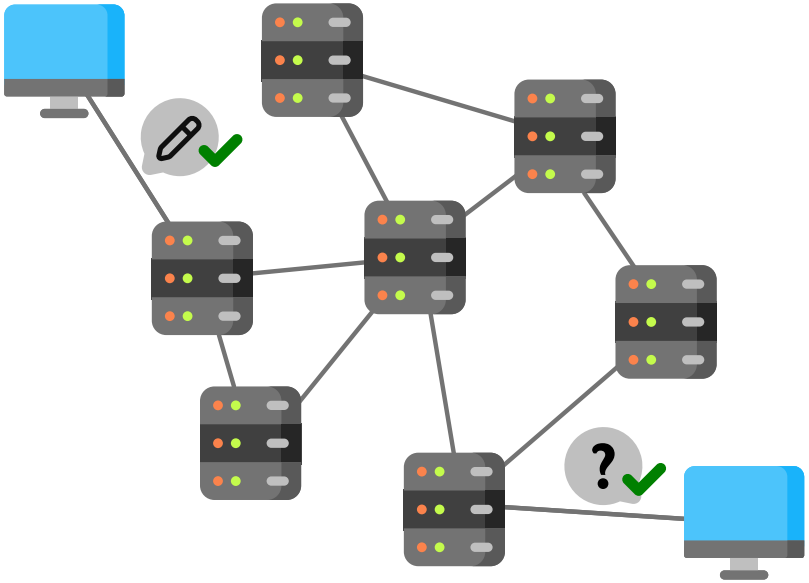
Available



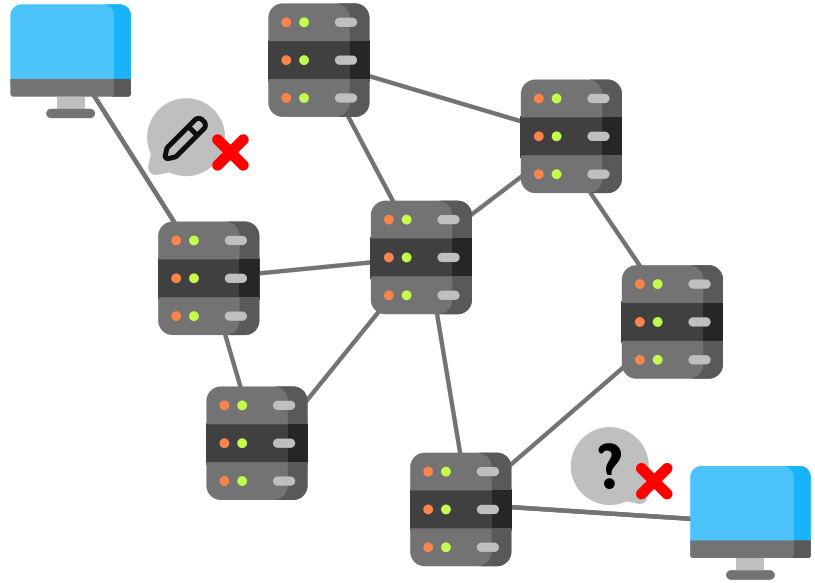
Unavailable

Availability (A):

- Every request received by a non-failing node in the system must result in a response



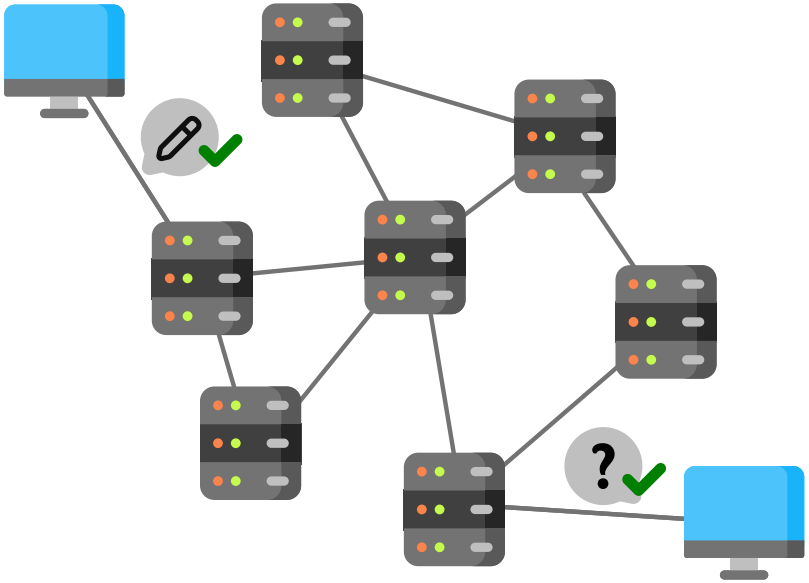
Available



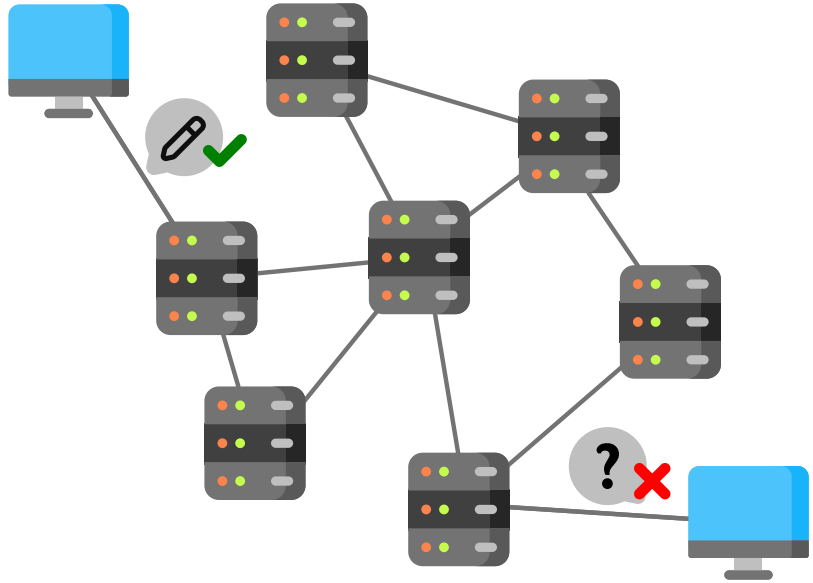
Unavailable

Availability (A):

- Every request received by a non-failing node in the system must result in a response



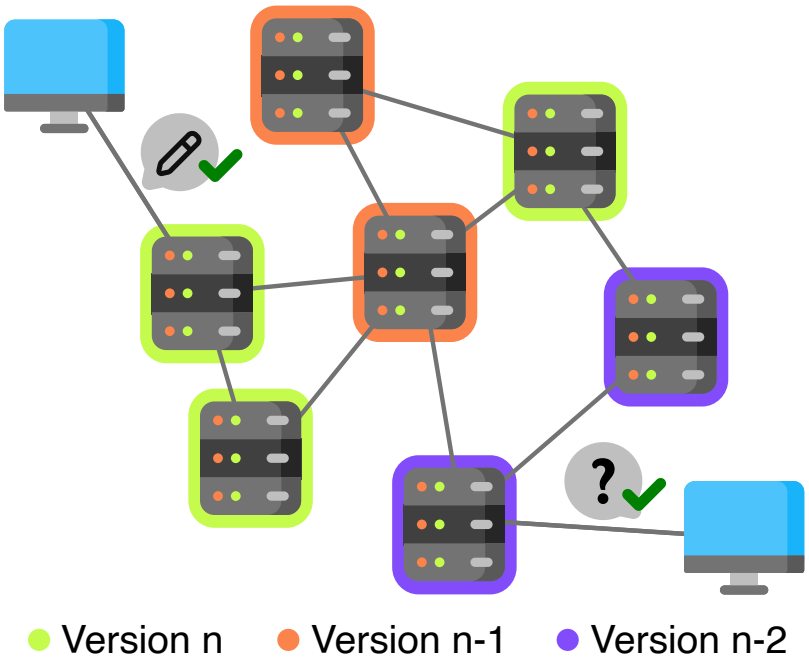
Available



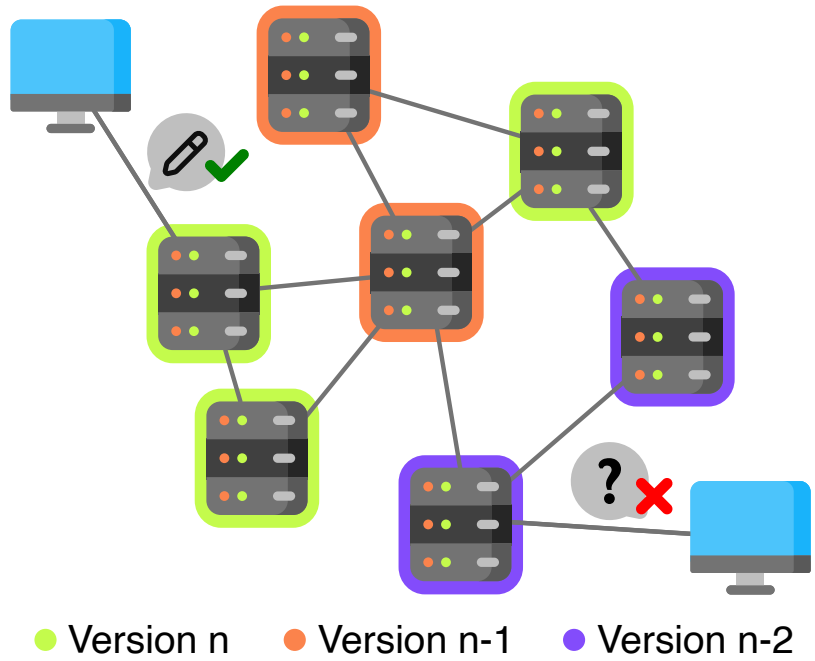
Unavailable

Availability (A):

- Every request received by a non-failing node in the system must result in a response



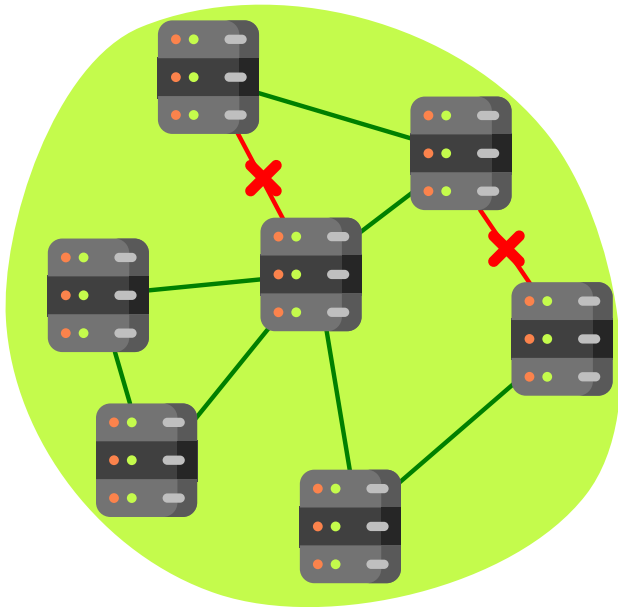
Available



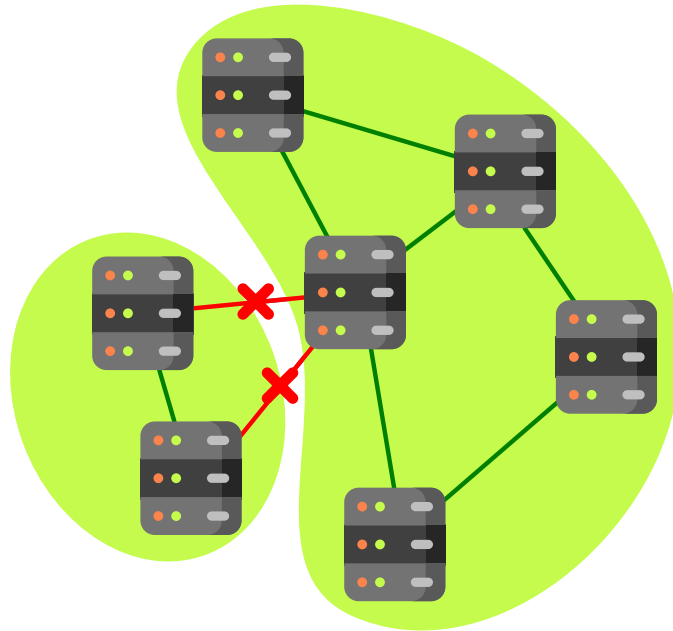
Unavailable

Availability (A):

- Every request received by a non-failing node in the system must result in a response, even if it is not with the latest data.



No partition



Partition

Partition Tolerance (P):

- The system continues to function despite network partitions.
- Tolerates arbitrarily many message losses.

Theorem (CAP[1])

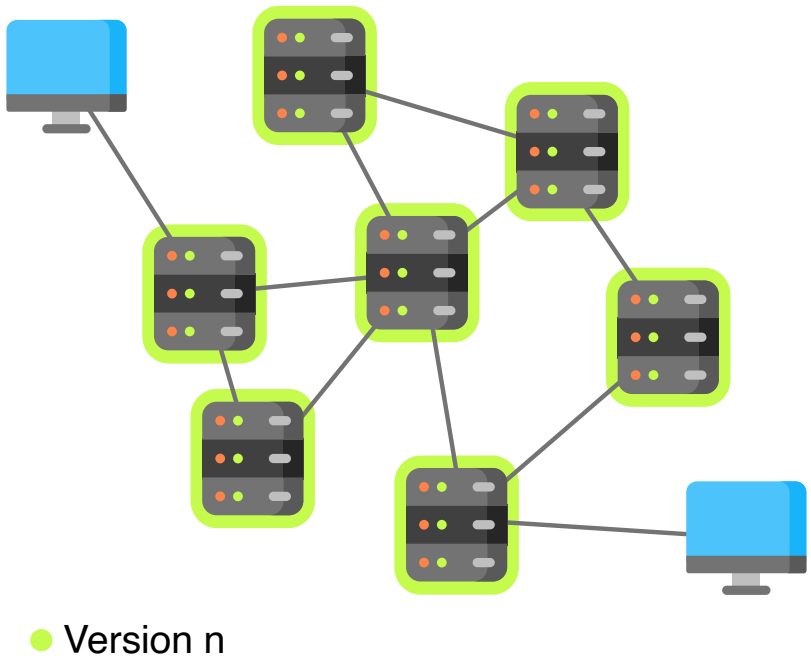
Any distributed system can have at most two of the following three properties:

- **C***onsistency*
- **A***vailability*
- **P***artition Tolerance*

Eric Brewer

[1] Armando Fox and Eric Brewer. "Harvest, Yield, and Scalable Tolerant Systems". In: **Proceedings of the Seventh Workshop on Hot Topics in Operating Systems**. Mar. 1999, pp. 174–178

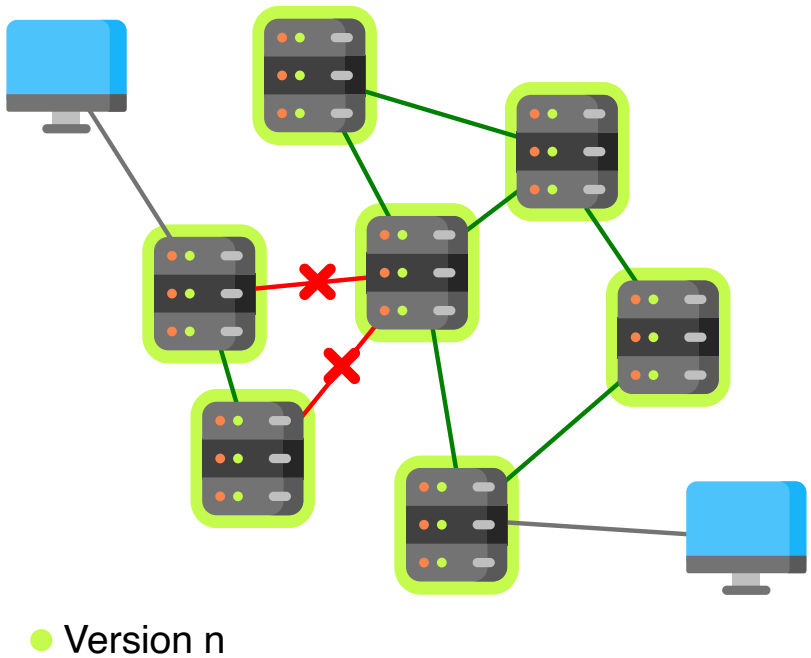
Sketch of the proof from Gilbert and Lynch[2]



1. By contradiction, consider a system S that fulfills the three properties.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

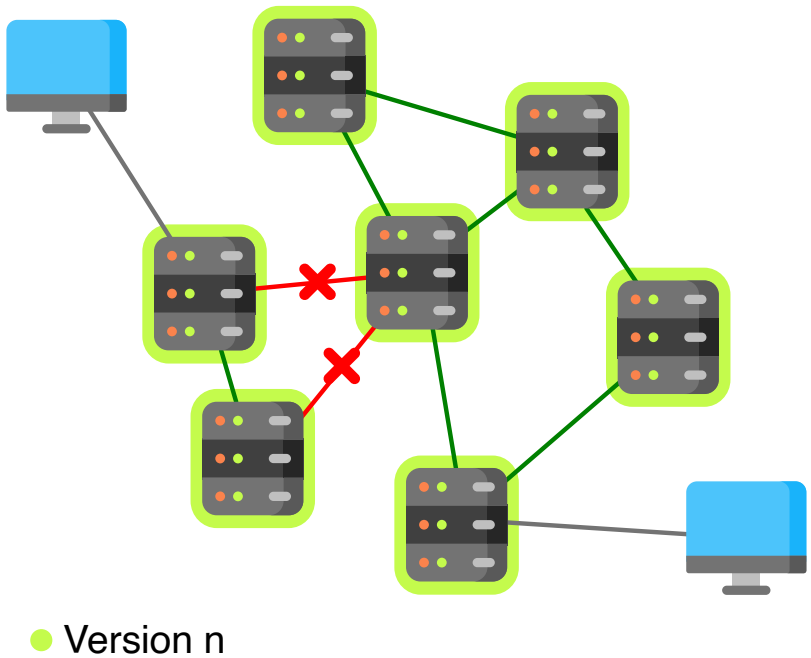
Sketch of the proof from Gilbert and Lynch[2]



2. Partition the system into two subsystems S_1 and S_2.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

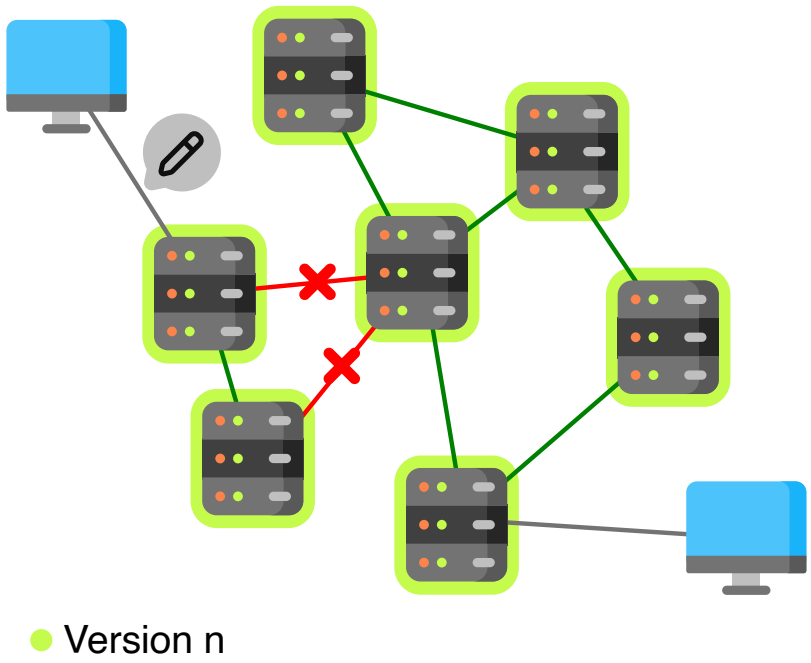
Sketch of the proof from Gilbert and Lynch[2]



3. By **partition tolerance**, the system continues to function.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

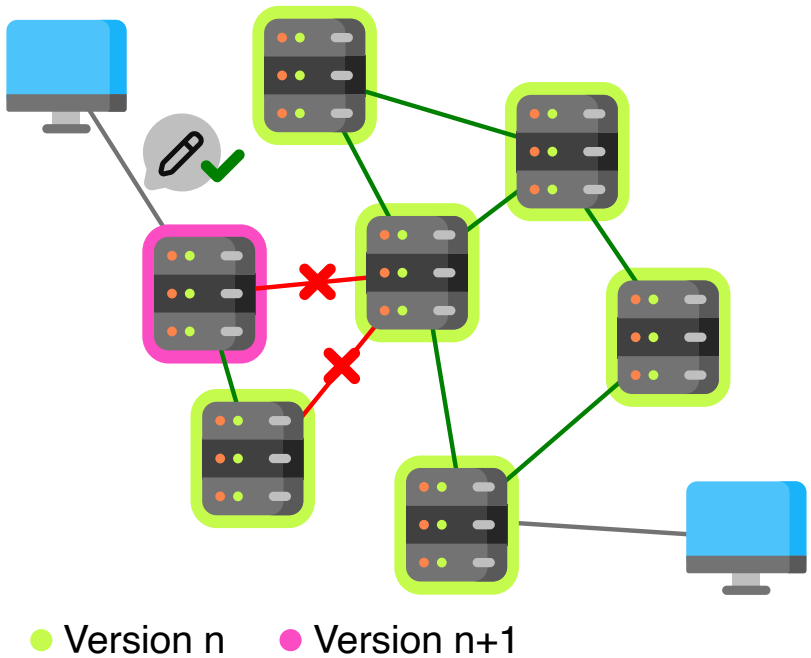
Sketch of the proof from Gilbert and Lynch[2]



4. Client c_1 requests to write new data to a node in S_1.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

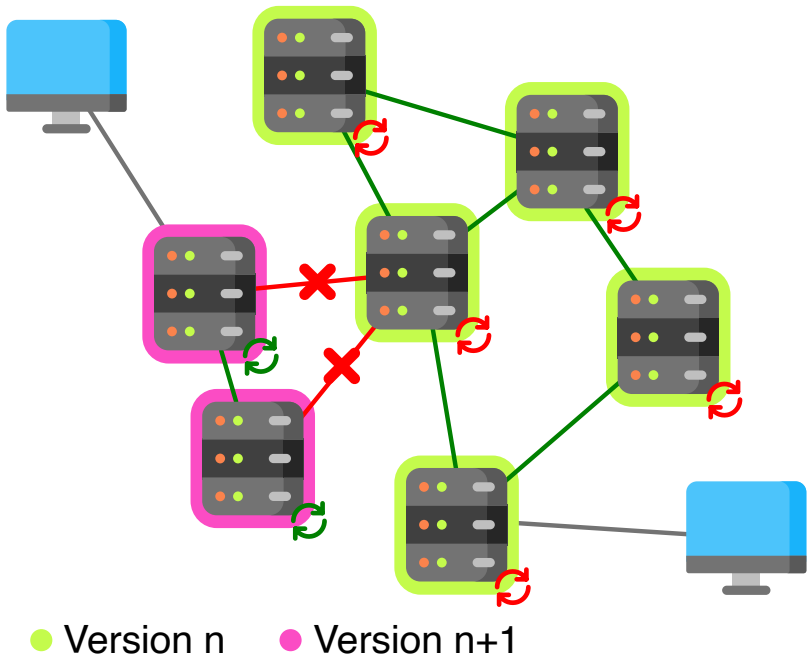
Sketch of the proof from Gilbert and Lynch[2]



5. By **availability**, the system handles the request and writes the new data.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

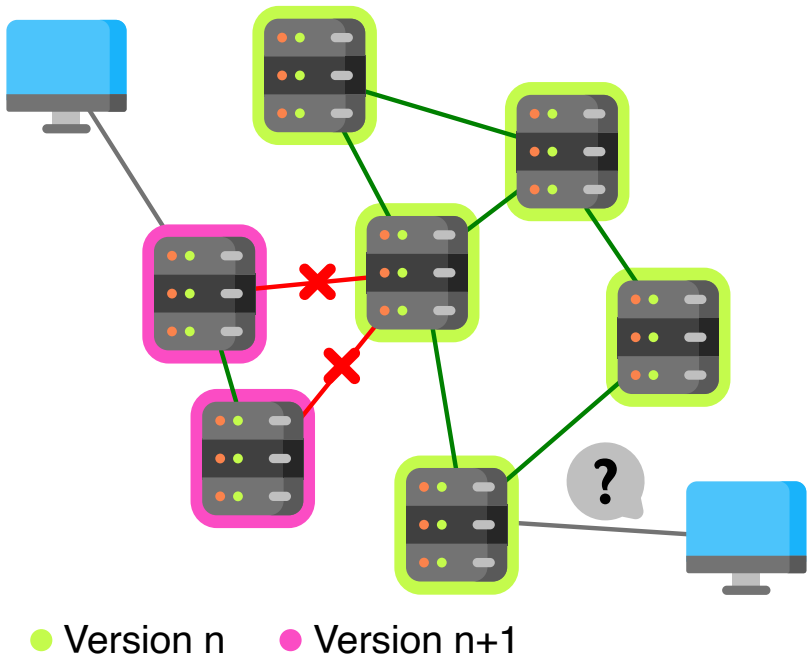
Sketch of the proof from Gilbert and Lynch[2]



6. Since the network is partitioned, the data cannot be replicated to the nodes in S_2.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

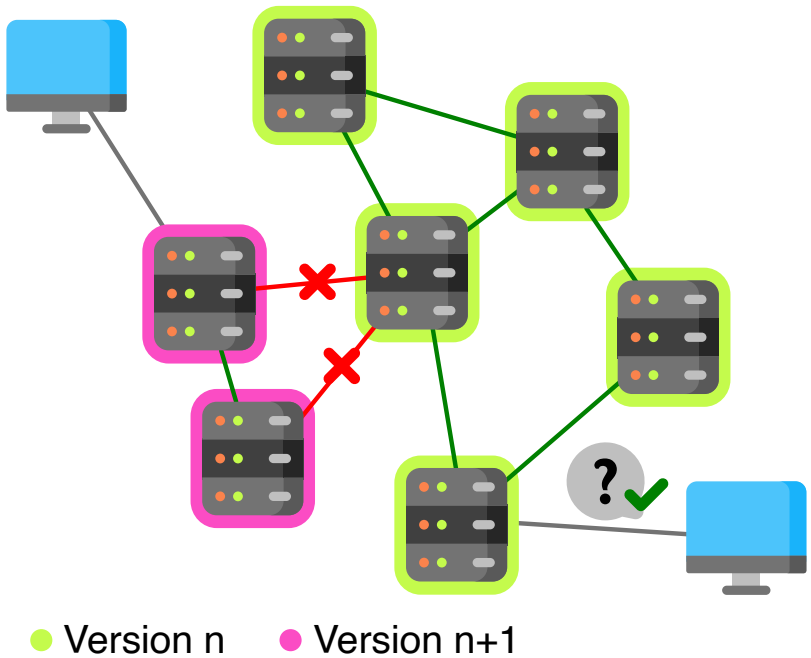
Sketch of the proof from Gilbert and Lynch[2]



7. Client c_2 requests to read the data to a node in S_2.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

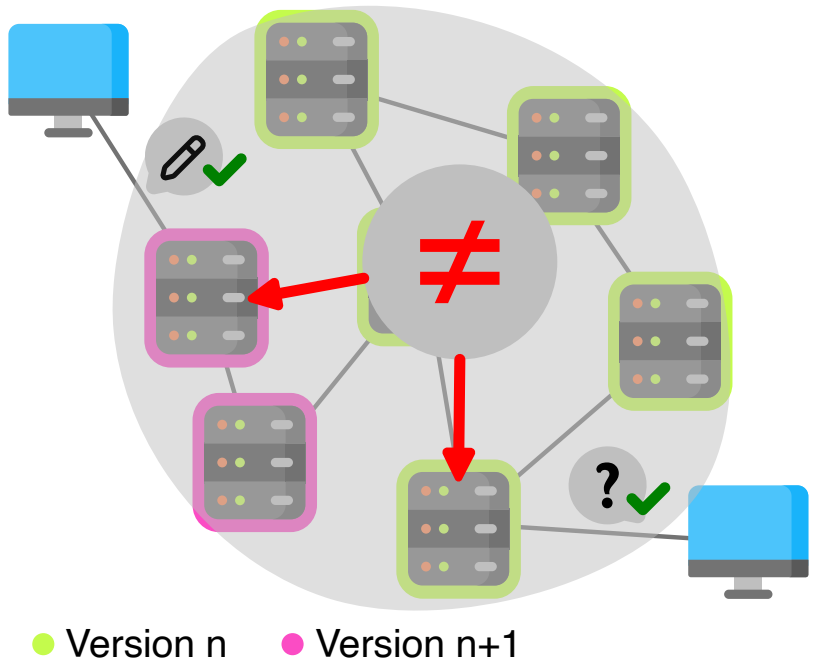
Sketch of the proof from Gilbert and Lynch[2]



8. By **availability**, the system handles the request and outputs the old data.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

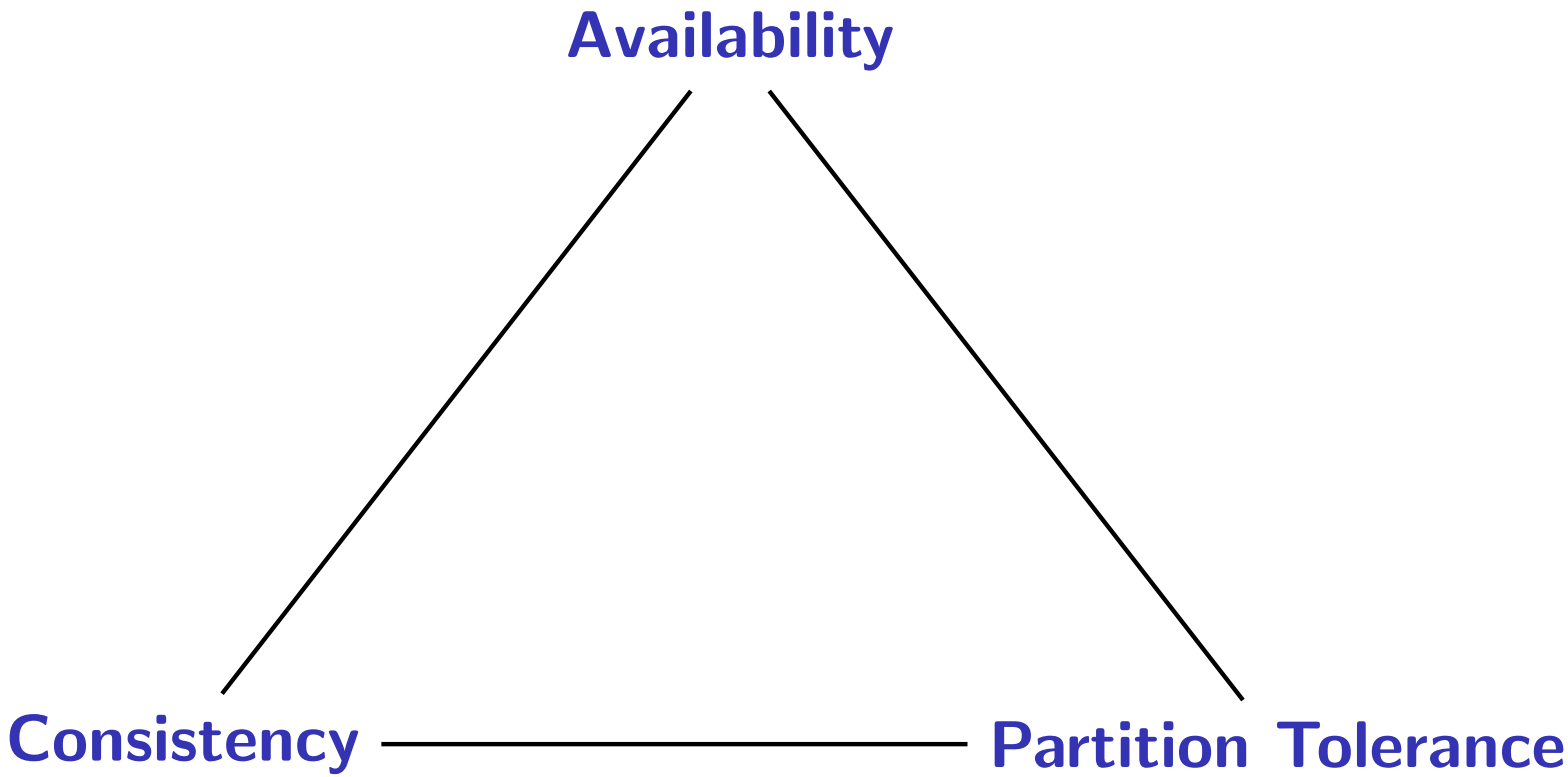
Sketch of the proof from Gilbert and Lynch[2]



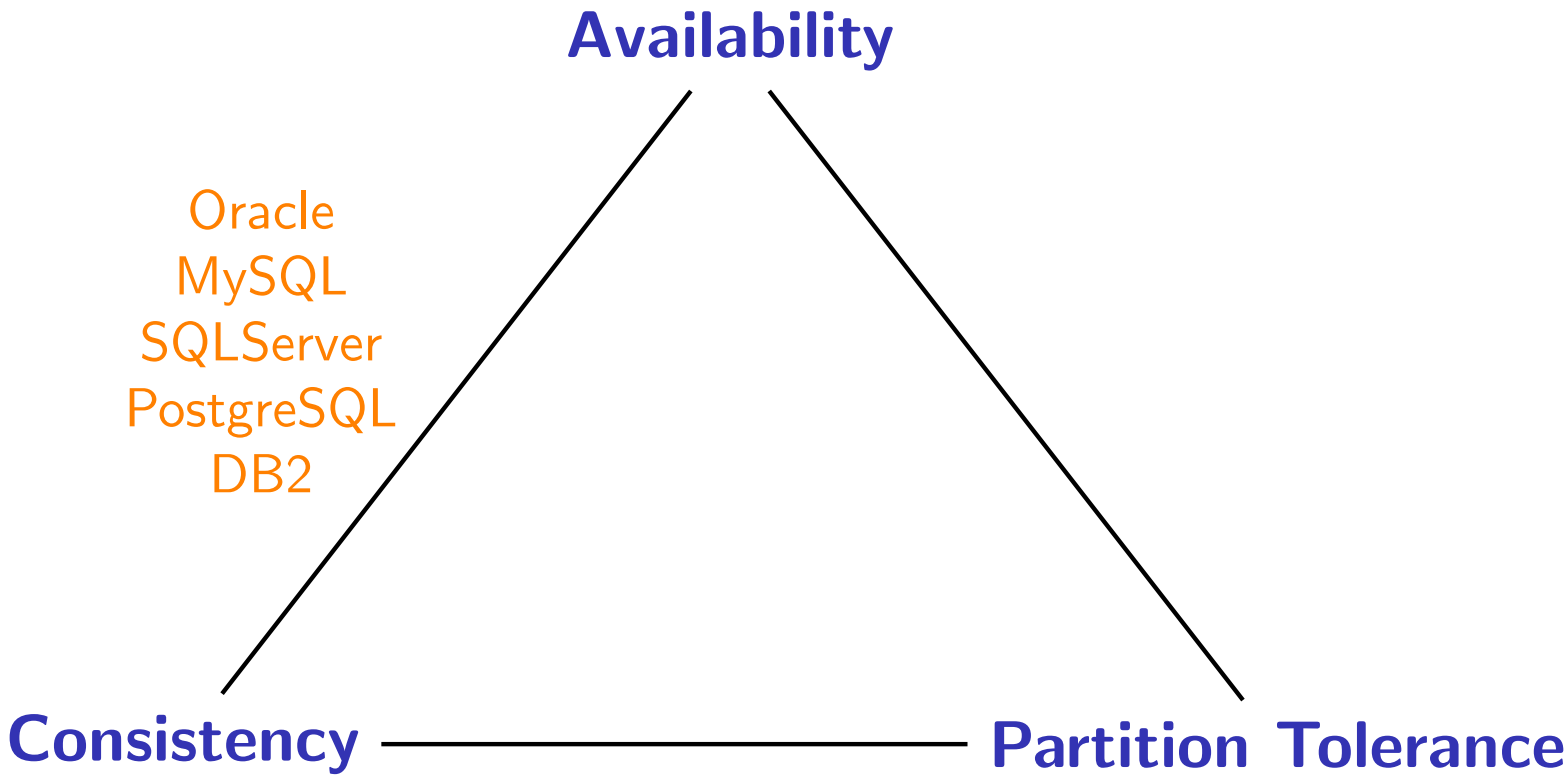
9. The second request yields an **inconsistent** answer.

[2] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services". In: **SIGACT News** 33.2 (2002), pp. 51–59.

Applications to NoSQL

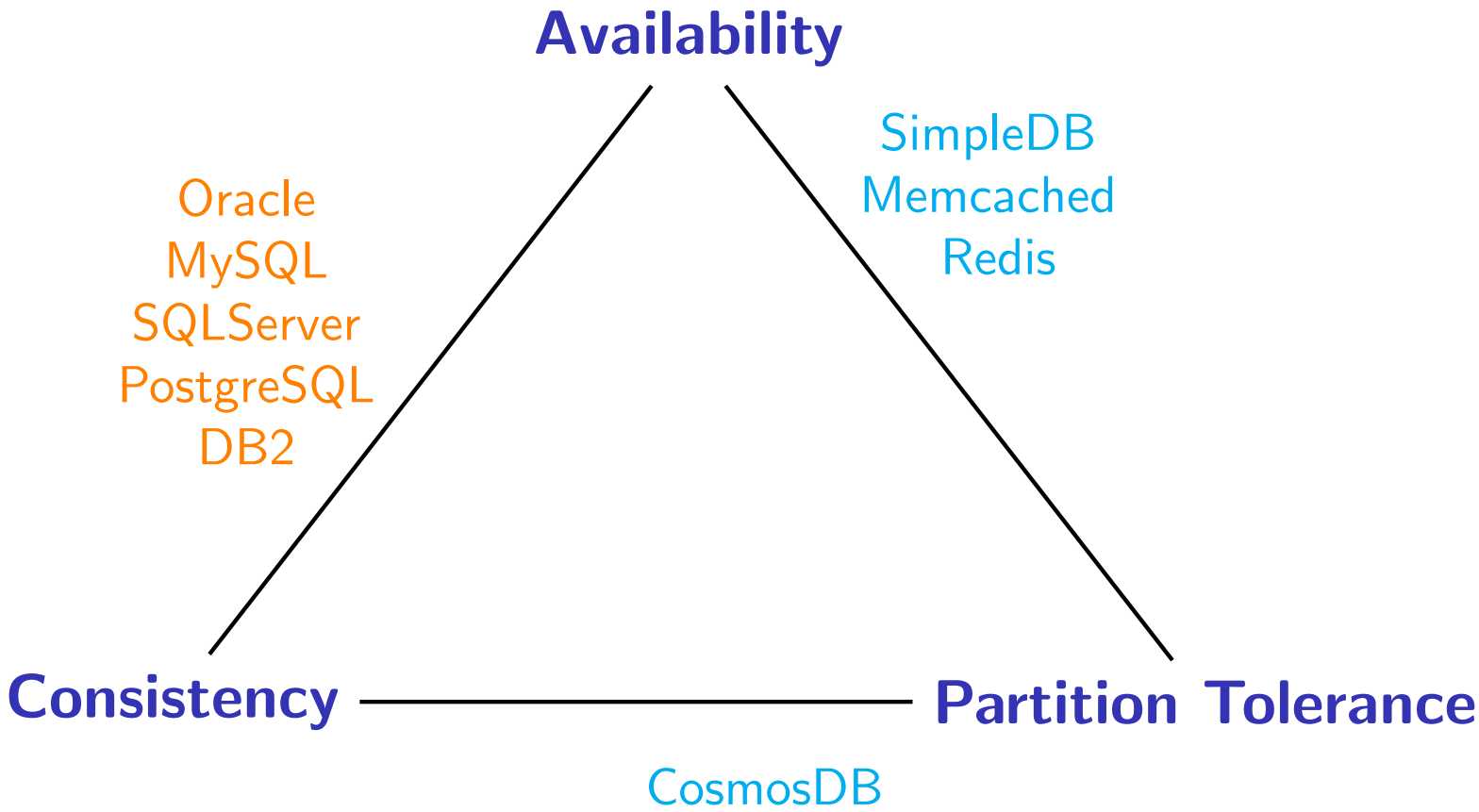


Applications to NoSQL



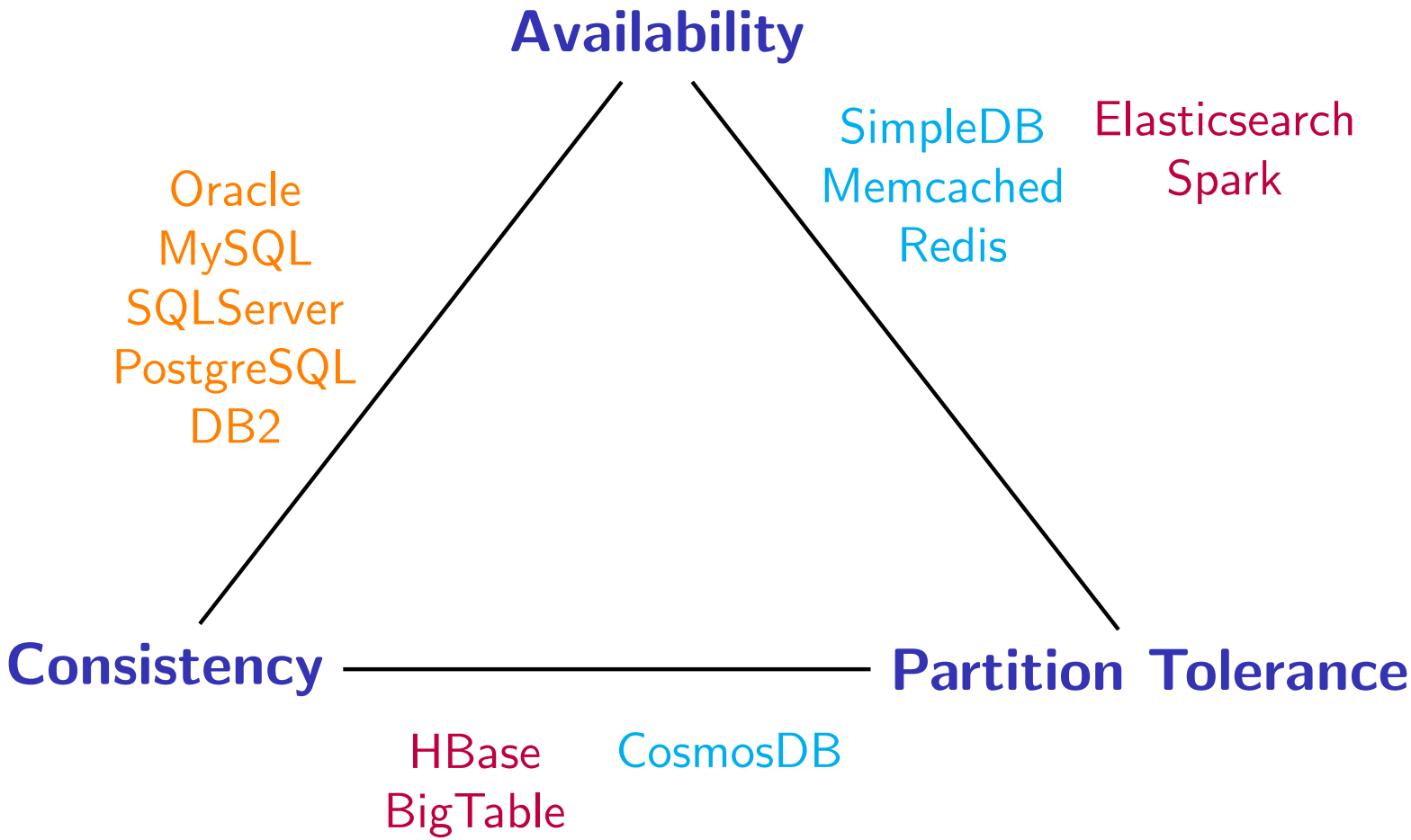
Relational

Applications to NoSQL

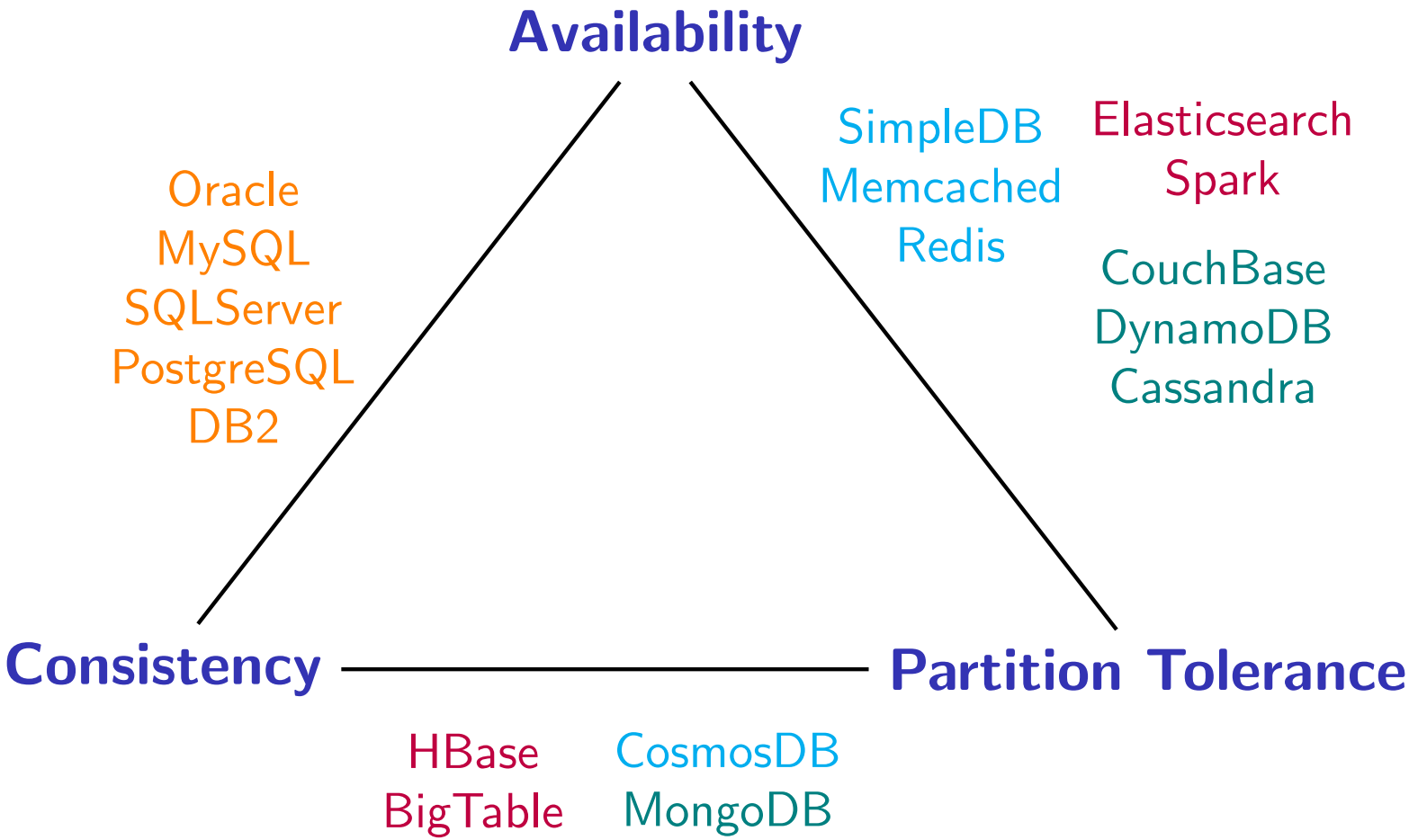


Relational Key-Value

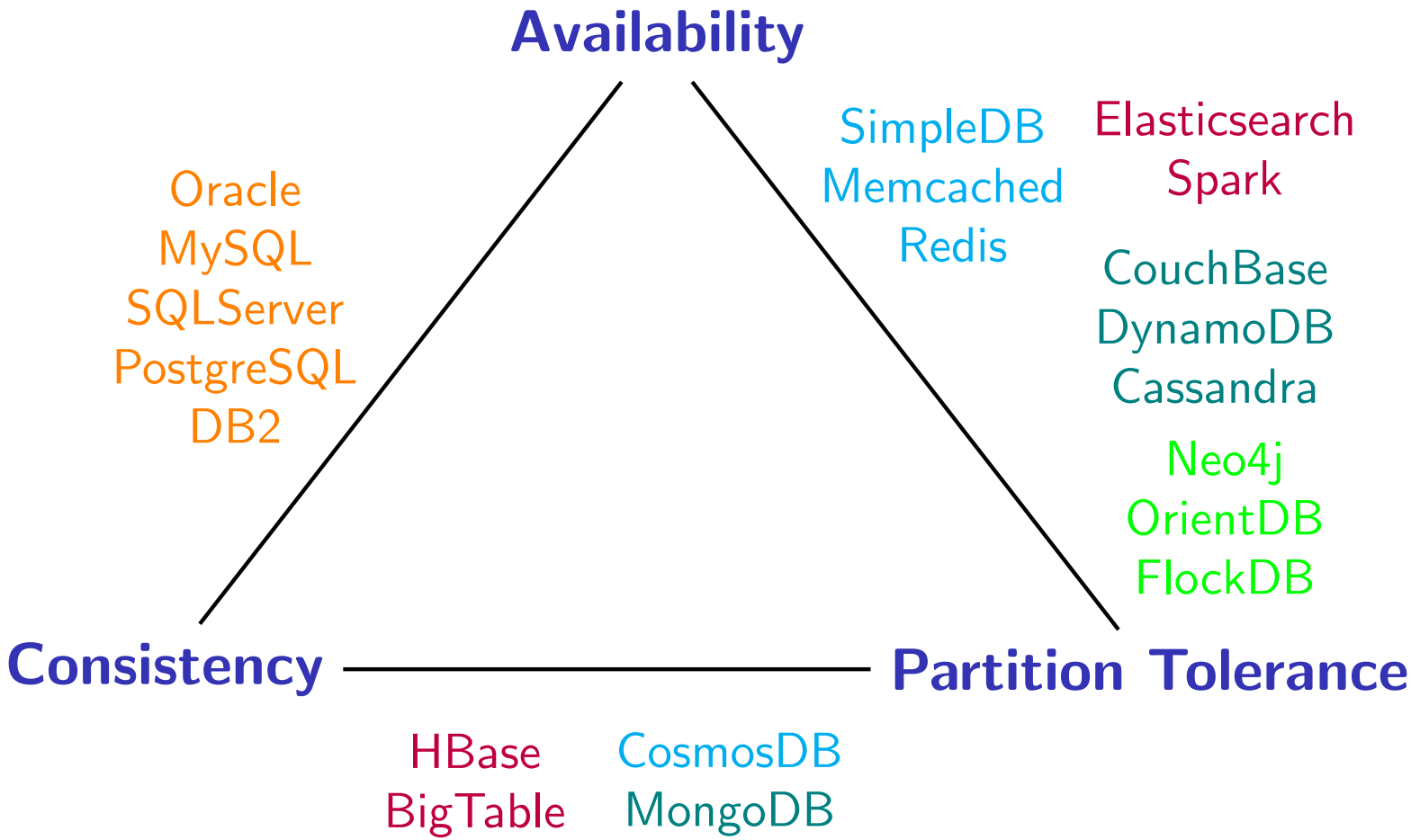
Applications to NoSQL



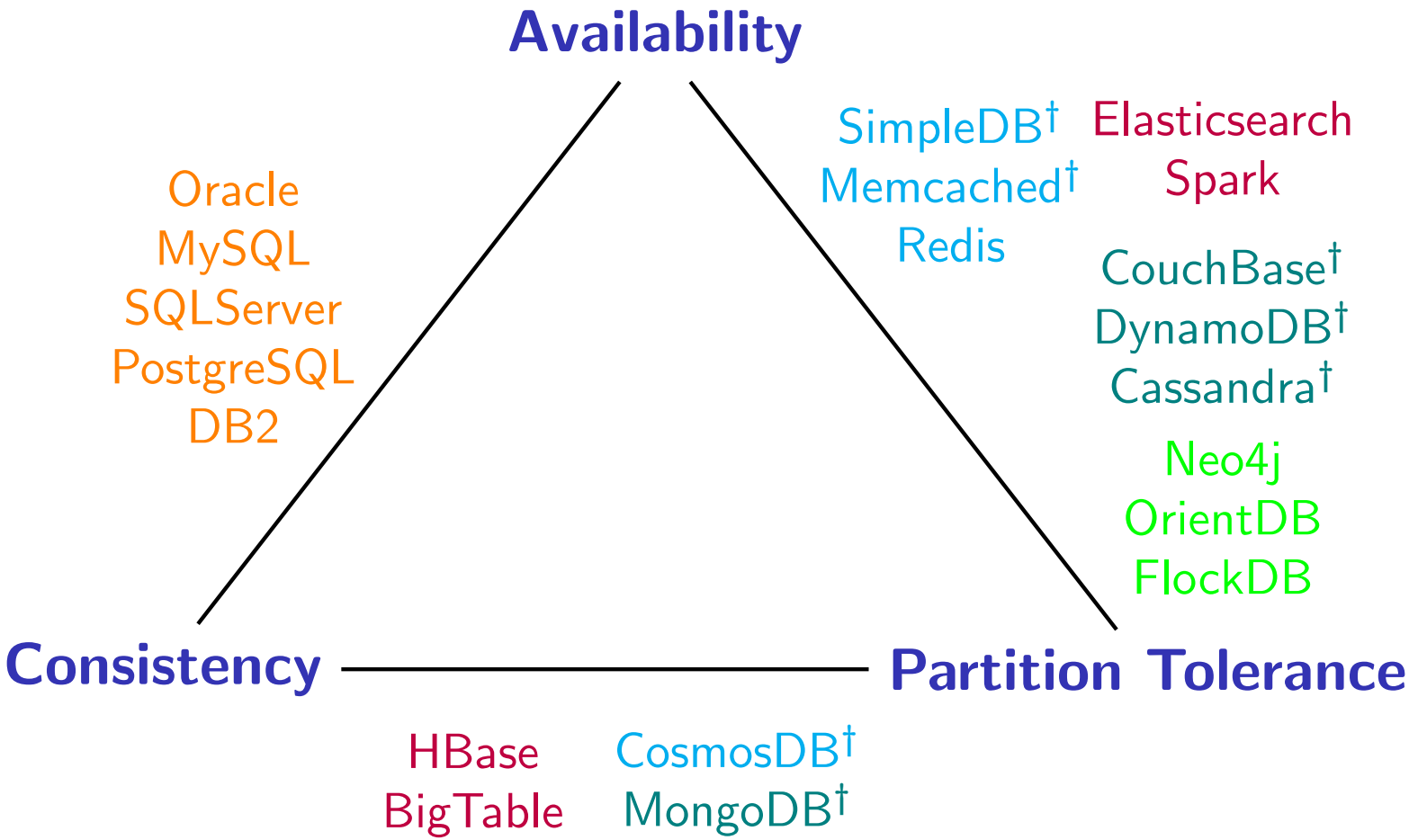
Applications to NoSQL



Applications to NoSQL



Applications to NoSQL



Relational Key-Value Column Document Graph †C↔A

Example: Bank Database

A financial database stores each client's bank balance.

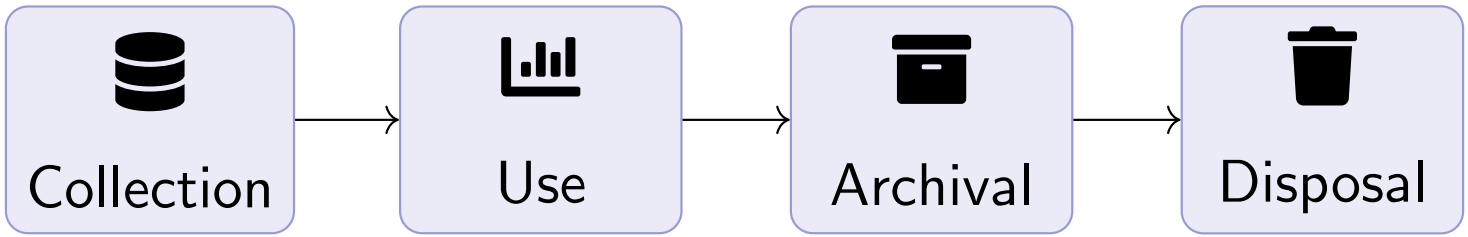
Clients interact via an ATM to:

- Check balance
- Withdraw money
- Deposit money

Given these operations, which aspect of the CAP theorem would you prioritize: consistency, availability, or partition tolerance?

Long Time Storage

Long-Term Storage in the Data Science Lifecycle



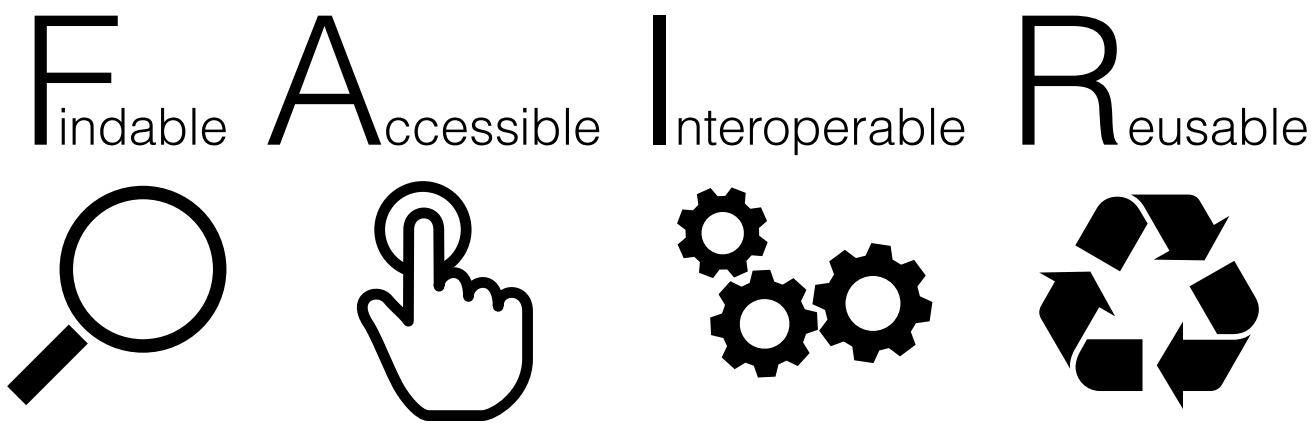
Why Is Data Archive Part of Data Science?

- Reproducibility of analyses and results
- Compliance with data retention policies
- Future reuse for new analyses or model training
- Preservation of data provenance and metadata

Challenges and Principles of Long-Term Storage

Technical Challenges:

- **Format Compatibility**: Ensuring readability by future tools
- **Metadata Preservation**: Maintaining reproducibility
- **Data Integrity**: Preventing corruption over time
- **Storage Costs**: Balancing accessibility and expense



Data Retention and Disposal

Data Retention Policies

Legal requirements

Organizational policies

Data value for future use

Regular policy review

Secure Data Disposal

Physical destruction

Cryptographic shredding

Ensure compliance with GDPR "right to be forgotten"

Formats for Archival

Criteria for Archival Formats:

- Open standards (e.g., CSV, JSON, Parquet)
- Self-describing (e.g., JSON with schema, Parquet with metadata)

```
{
  "metadata": {
    "format": "JSON",
    "created": "2025-10-12"
  },
  "data": "..."
}
```

Example: Archiving a Data Science Project

```
project/
|-- data/
|   |-- raw/                    # Original immutable data
|   |-- processed/              # Cleaned/processed data
|   \-- README.md               # Data documentation
|-- models/
|   |-- model.onnx              # Serialized model
|   \-- metadata.json           # Model metadata
|-- notebooks/                  # Analysis notebooks
|-- requirements.txt            # Python dependencies
\-- README.md                   # Project documentation

```

Key Files to Archive

- Raw and processed data (in open formats)
- Trained models and their metadata
- Analysis code and notebooks
- Environment specifications
- Documentation and data dictionaries

Conclusion

Ethical and Practical Considerations

Ethical Implications

- ⚖ **Privacy**: Who has access to the data?
- 🌍 **Accessibility**: Is the data available to those who need it?
- 🌿 **Sustainability**: What's the environmental impact?

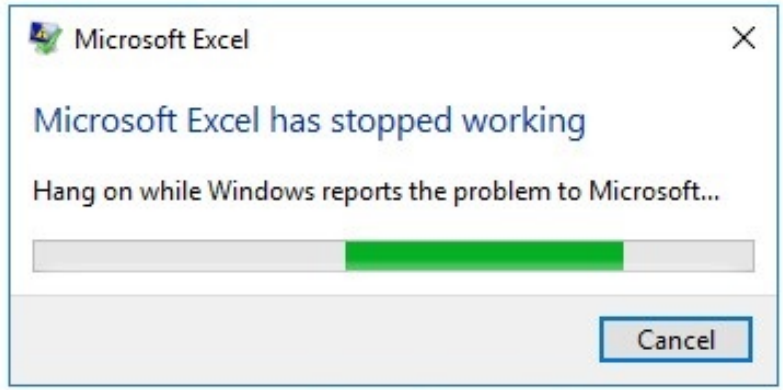
Practical Considerations

- 🕒 **Long-term storage**: Will you need this data in 5 years?
- $ **Cost**: What's the total cost of ownership?
- ⚙ **Maintenance**: Who will maintain this storage system?

Common Pitfalls and Best Practices

Pitfalls

- Using Excel for big data
- Over-engineering (e.g., graph DB for simple data)
- Ignoring costs (e.g., cloud storage fees)



Best Practices

1. Start simple (CSV/SQL), scale up as needed
2. Document storage schema and access patterns
3. Test with real data early

Takeaways: Data Storage

1. Understand the difference between storage **models** (flat, tabular, hierarchical, columnar) and **formats** (CSV, JSON, Parquet)
2. Choose the right format based on your use case: Excel for business, CSV/JSON for sharing, Parquet for analytics
3. Consider the trade-offs between readability, size, speed, and interoperability when selecting a format
4. For long-term storage, prioritize open standards and self-describing formats with proper metadata
5. Document your storage decisions and maintain data provenance for reproducibility
6. Consider ethical implications including privacy, accessibility, and sustainability