

DB Week 10

Workshop

INFO20003 | Sandy Luo

Workshop Overview

01

**Capacity
Planning**

02

**Backup &
Recovery**

03

Transactions

04

**Lab:
Transactions**

01

**Disk space
requirements**

02

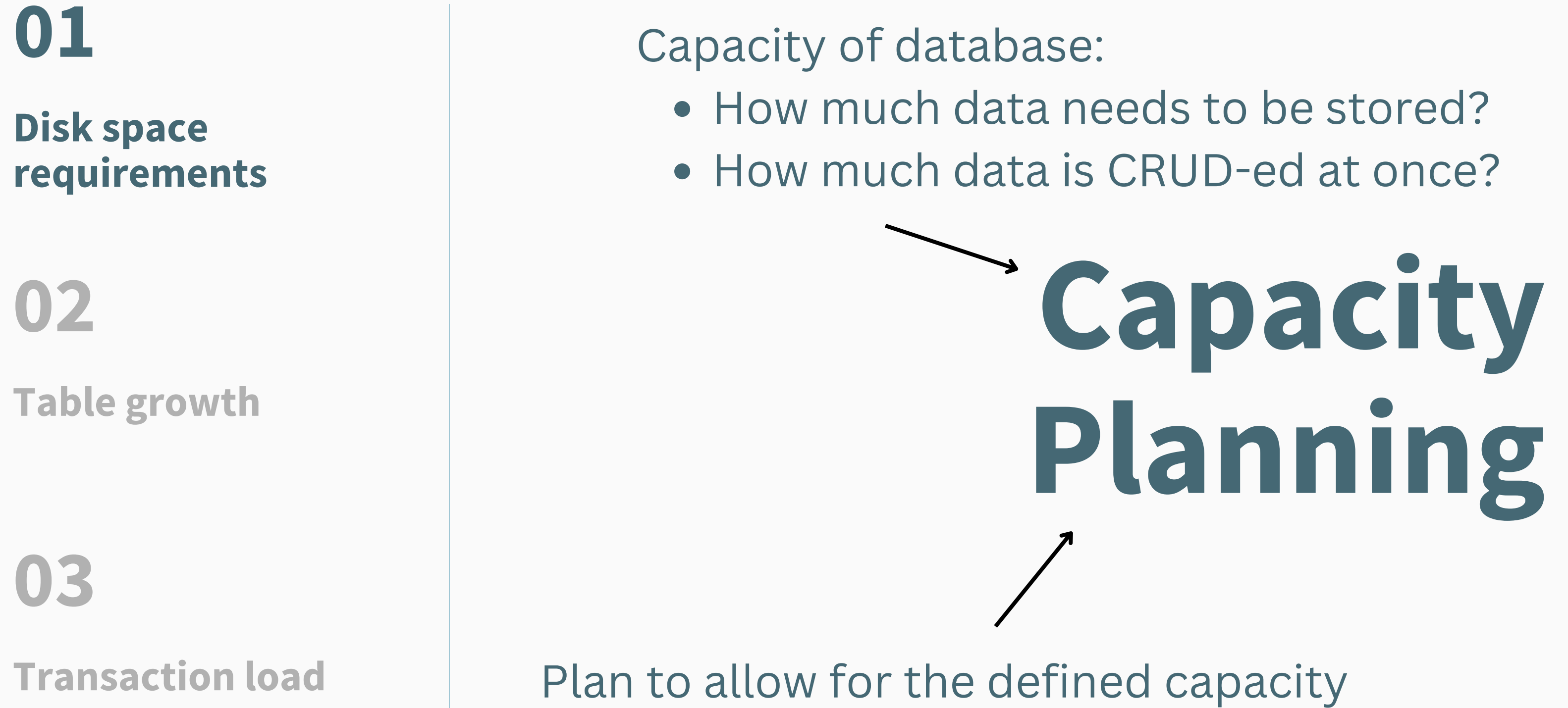
Table growth

03

Transaction load

Capacity of database:

- How much data needs to be stored?
- How much data is CRUD-ed at once?



Capacity Planning

Plan to allow for the defined capacity

Capacity Planning

- Estimating disk space & transaction load

1. Estimate *disk space requirements*

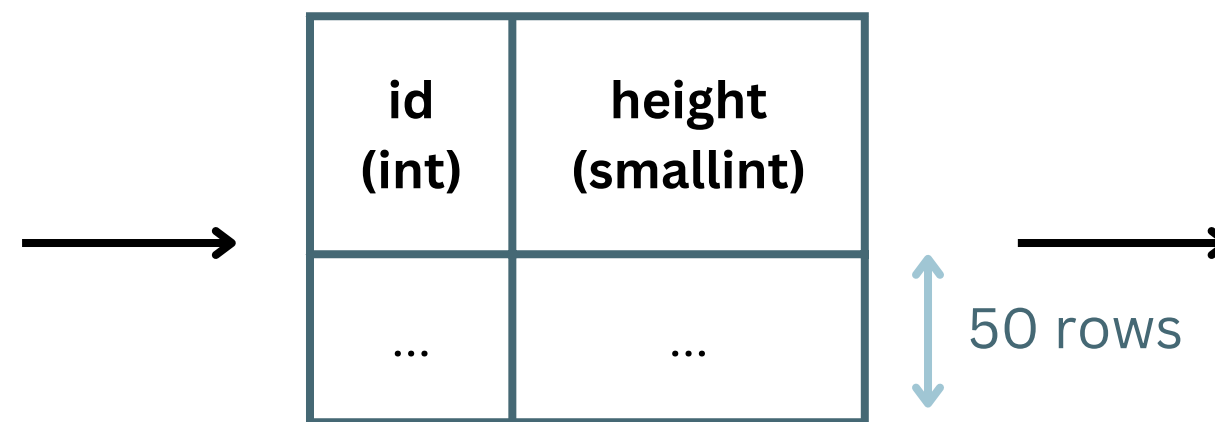
2. Estimate *table growth*

3. Estimate *transaction load*

(a) Disk Space Requirements

- Average row width = sum of attribute data type sizes
- Estimated table size = **#rows * average row width**
- Estimated DB size = sum of **all** table sizes

Data Type	Storage Required
<u>TINYINT</u>	1 byte
<u>SMALLINT</u>	2 bytes
<u>MEDIUMINT</u>	3 bytes
<u>INT, INTEGER</u>	4 bytes
<u>BIGINT</u>	8 bytes



Avg row width = 4 + 2 = 6 bytes

Table size = 6 * 50 = 300 bytes

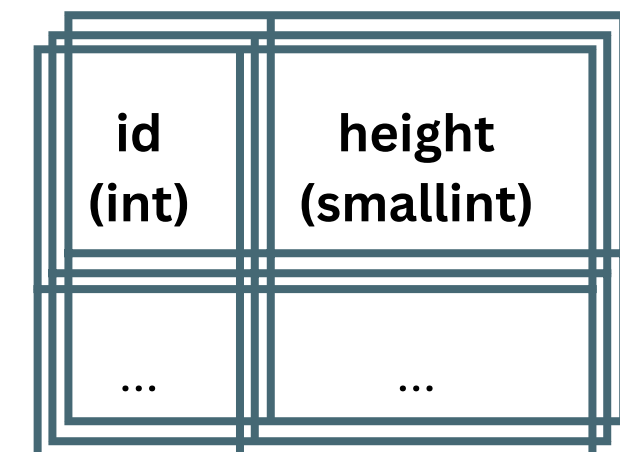


Table sizes: 300 x 3

Estimated DB size = 900 bytes

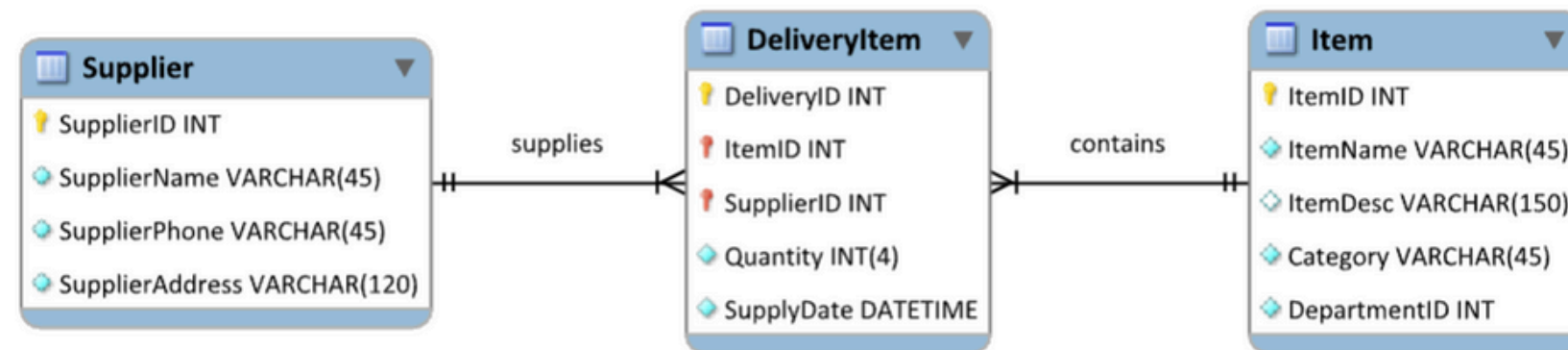
(b) Table growth

- Gather estimates during system analysis
- This information will generally be provided:
 - e.g. “*Table x grows by 1M rows / day*”

(c) Transaction Load

- Consider **each** business transaction:
 - a. How often will each transaction be run?
 - b. For each transaction, what SQL statements are being run?

1. Capacity Planning



Consider the case of a department store. An analyst has determined that there are 50 distinct suppliers that provide 2000 distinct items to the store. They have determined that the average delivery is of 40 distinct items and that each supplier delivers approximately once a week (the analyst has estimated this to be 50 deliveries a year). For each delivery by a supplier, there are on average 40 rows added to the DeliveryItem table. While the Item and Supplier tables stay constant in size, the DeliveryItem table grows by 100,000 rows every year.

This assumes that suppliers and items stay constant; however, if the business is successful, the suppliers and frequency of deliveries and number of distinct items delivered can be expected to grow. If we know the length of each row, we can estimate how big each table will be year by year.

Using information about data type storage from the MySQL documentation and information from the data dictionary, the analyst has determined the following average row lengths of each table:

Table	Number of rows	Average row length
Supplier	50 rows	144 bytes
Item	2000 rows	170 bytes
DeliveryItem	0 rows	19 bytes

Table 1: Row volume and row length for the Supplier delivers Item entities.

Assume that the number of suppliers and number of items do not change from year to year, and that the delivery schedule remains the same. Calculate the size of the three tables:

- When database use begins (year 0)
- After one year of database use
- After five years of database use

1.

Consider the case of a department store. An analyst has determined that there are 50 distinct suppliers that provide 2000 distinct items to the store. They have determined that the average delivery is of 40 distinct items and that each supplier delivers approximately once a week (the analyst has estimated this to be 50 deliveries a year). For each delivery by a supplier, there are on average 40 rows added to the DeliveryItem table. While the Item and Supplier tables stay constant in size, the DeliveryItem table grows by 100,000 rows every year.

- 50 suppliers * 50 deliveries/year/supplier = 2500 deliveries/year
- 2500 deliveries/year * 40 items/delivery = 100000 deliveryItems/year

1.

Table	Number of rows	Average row length
Supplier	50 rows	144 bytes
Item	2000 rows	170 bytes
DeliveryItem	0 rows	19 bytes

Table 1: Row volume and row length for the Supplier delivers Item entities.

Assume that the number of suppliers and number of items do not change from year to year, and that the delivery schedule remains the same. Calculate the size of the three tables:

(a) When the database use begins (year 0).

- Supplier: $50 * 144 = 7,200$ bytes
- Item: $2000 * 170 = 340,000$ bytes
- DeliveryItem: $0 * 19 = 0$ bytes

(b) After one year of database use

- Supplier & Item size unchanged
- DeliveryItem: $100,000 * 19 = 1,900,000$ bytes

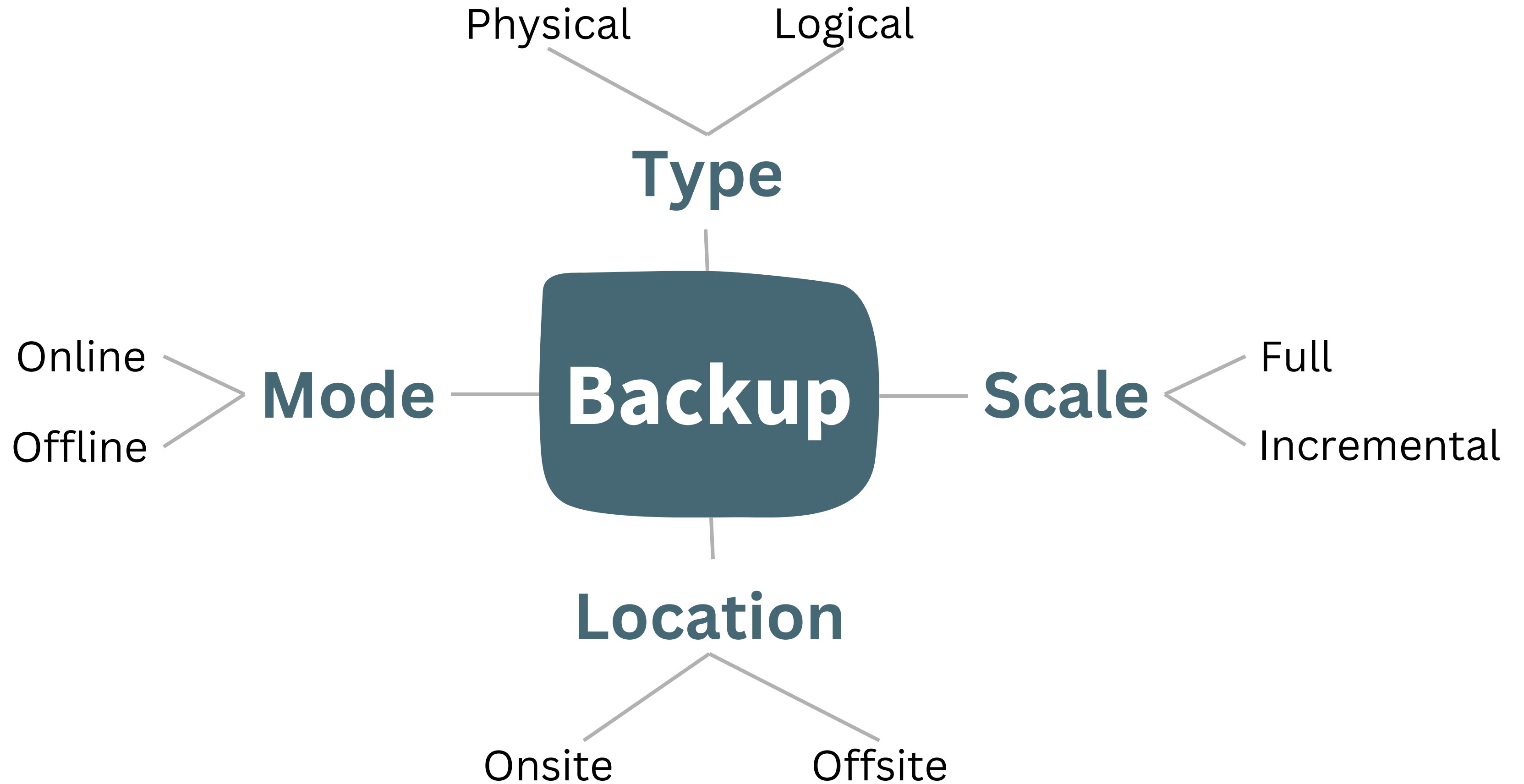
(c) After five years of database use

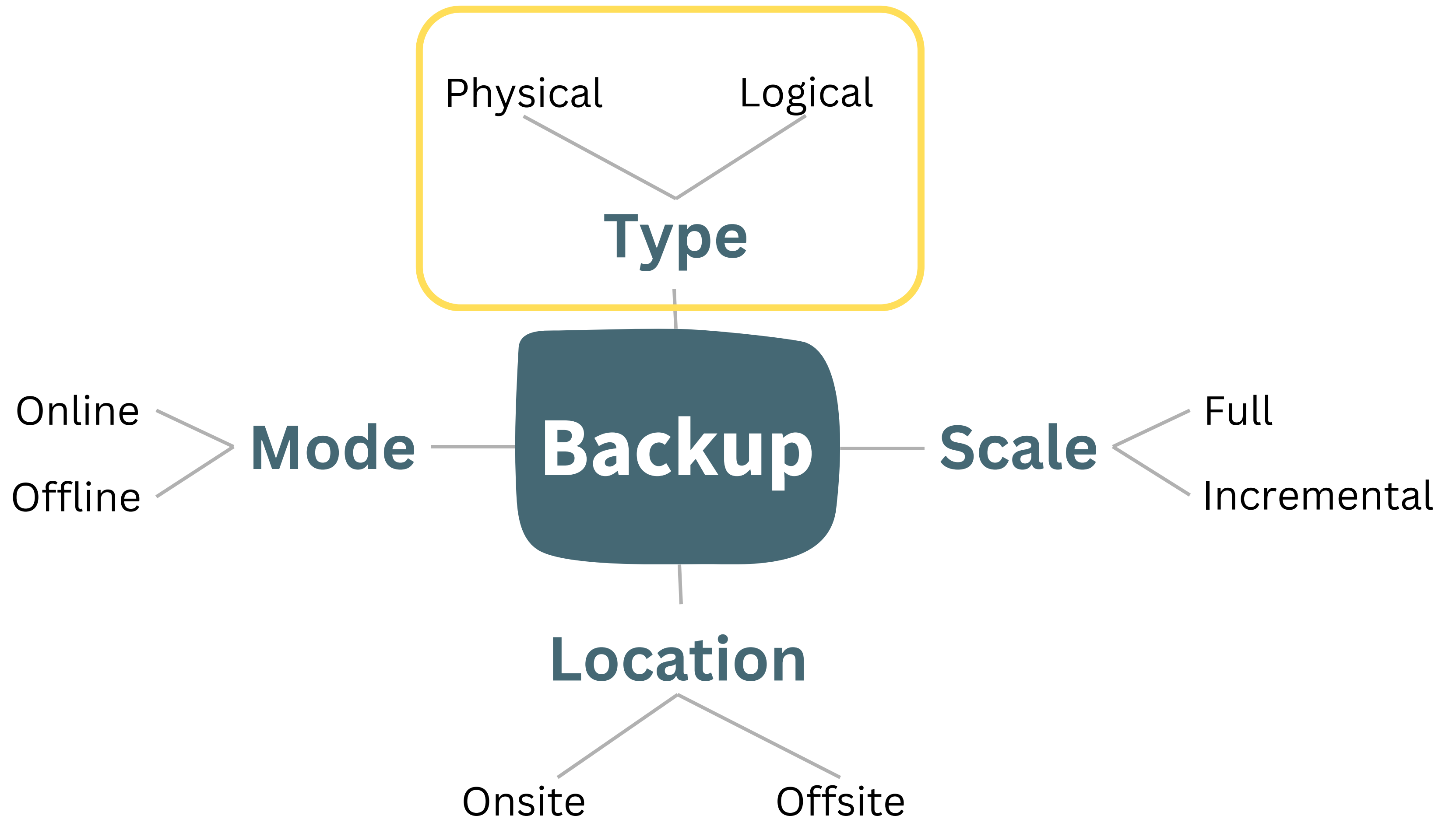
- Supplier & Item size unchanged
- DeliveryItem: $500,000 * 19 = 9,500,000$ bytes

Database Backup

Backup

- To recover in the event of failure of the database system
 - Network failure, memory failure, user error...
- Avoid:
 - Loss of data
 - Data integrity errors
 - Data mismatch



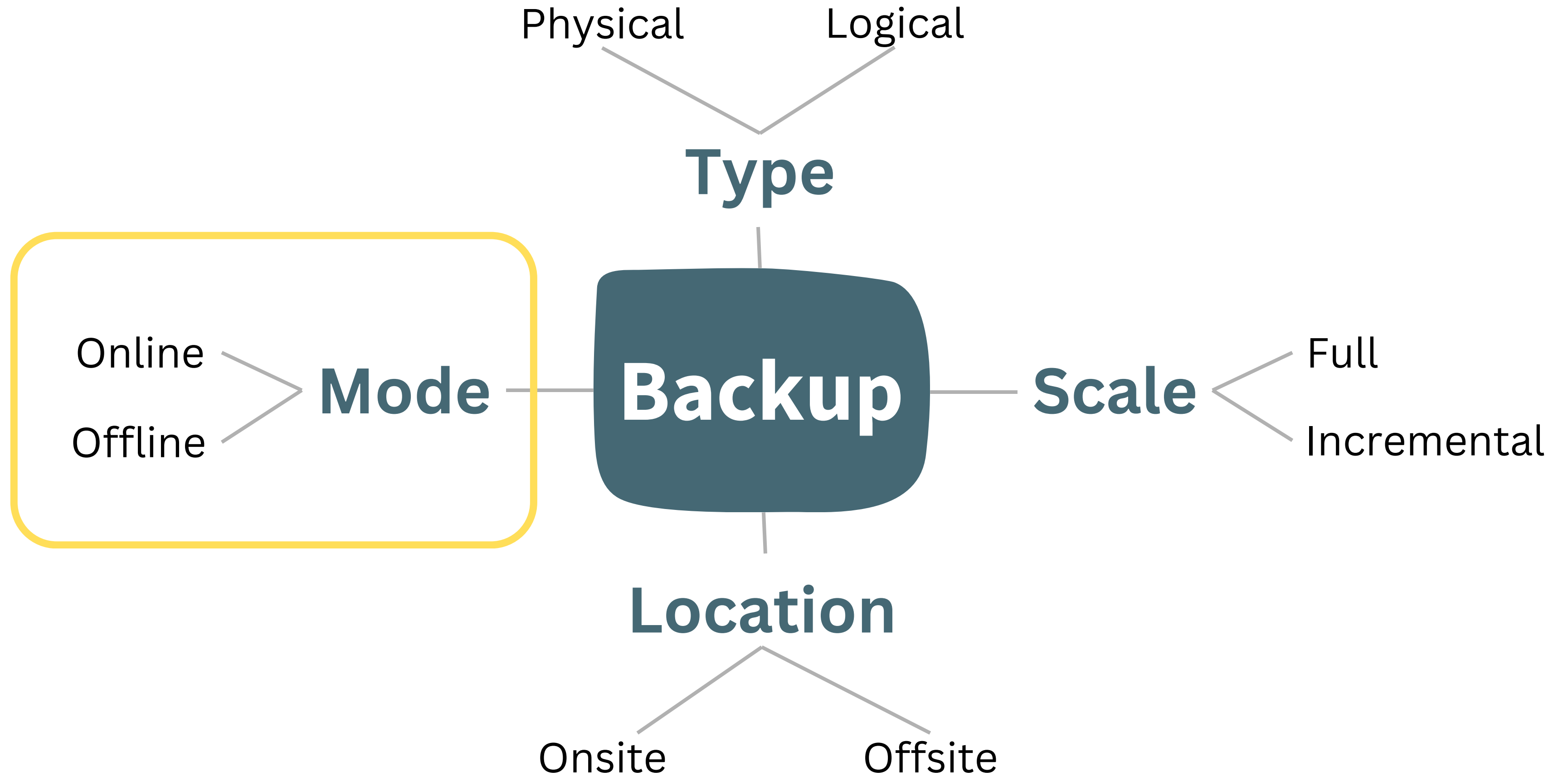


Physical Backup

- A direct image ***copy of the physical database files on the disk***
- Fastest way to make a copy of the database
- In the event of a database failure:
 - The physical copies can be restored to their original location
 - DBA replays all transactions using the **Crash Recovery log**
- Either **offline / online** (preferably offline)
- Good for large DB that needs fast recovery, but backup is only portable to machines with similar config. (**machine dependent**)

Logical Backup

- Completed through ***SQL queries***
- Keep a record of the data & all metadata
 - Takes more information than physical backup (table structure etc.)
- Larger and slower than physical backup
- Good for:
 - Moving data from one operating system to another
 - Migrating data from one database to a completely different database and environment
- **Must be *online*** → server is available during backup

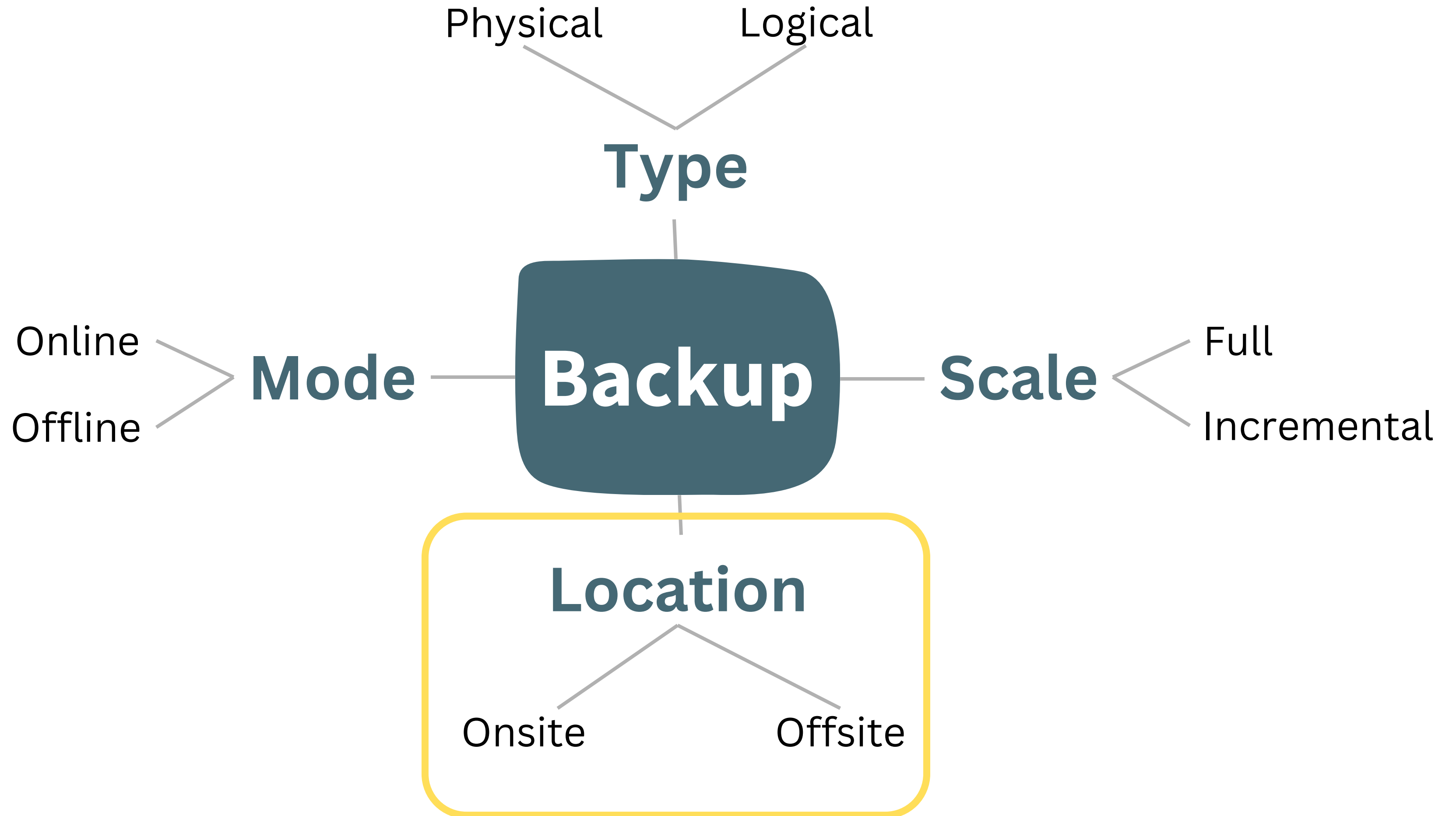


Online (HOT) Backup

- Backups occur when DB is “**live**”
 - Clients don’t realise a backup is in progress
- Need to have appropriate locking to ensure integrity of data

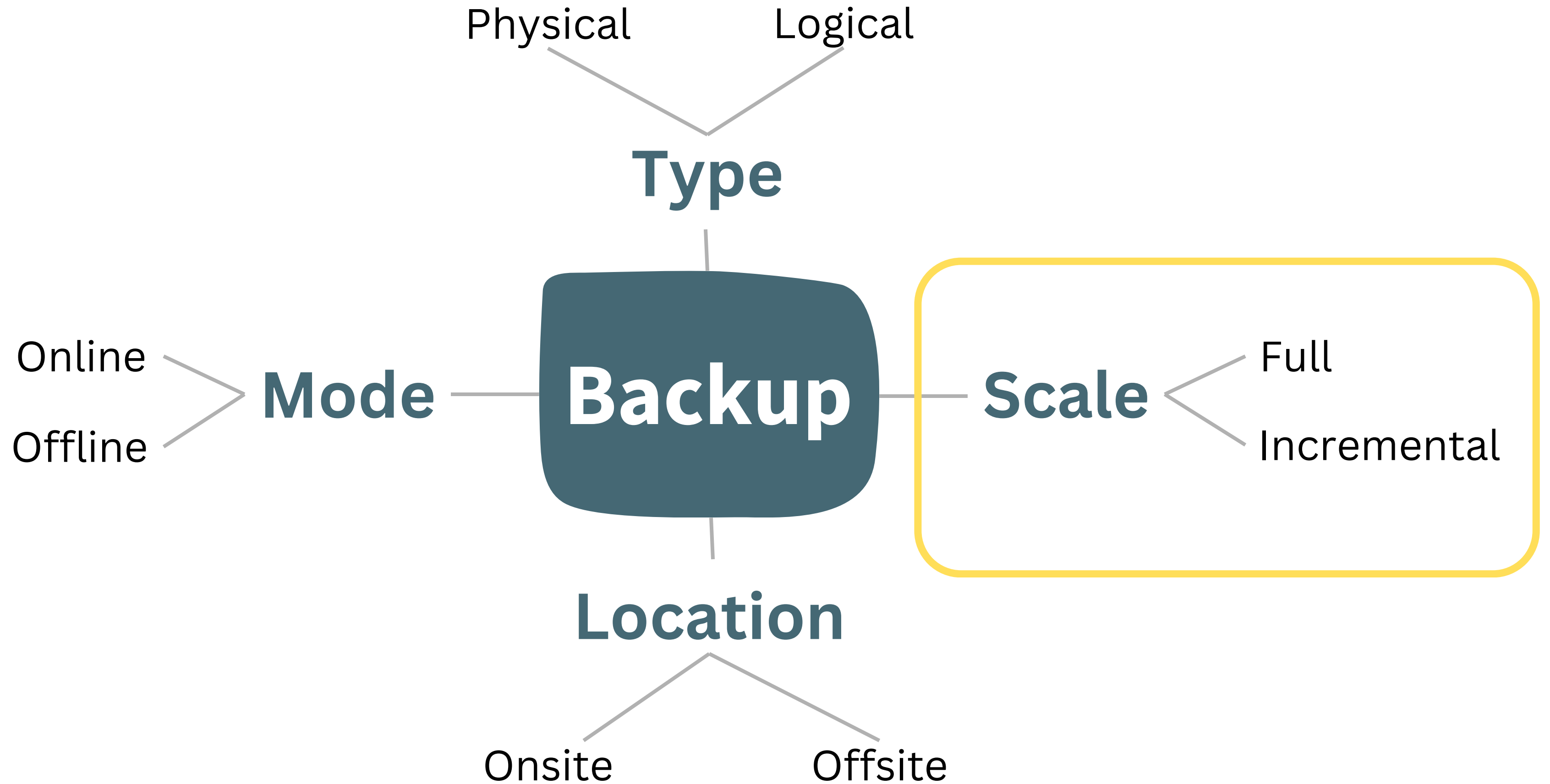
Offline (COLD) Backup

- Backups occur when DB is **shut down**
 - Maximise availability to users → backup from replication server, not live
- Simpler to perform
 - More preferable than HOT due to data integrity reasons, but not available in all situations (e.g. applications w/o downtime)



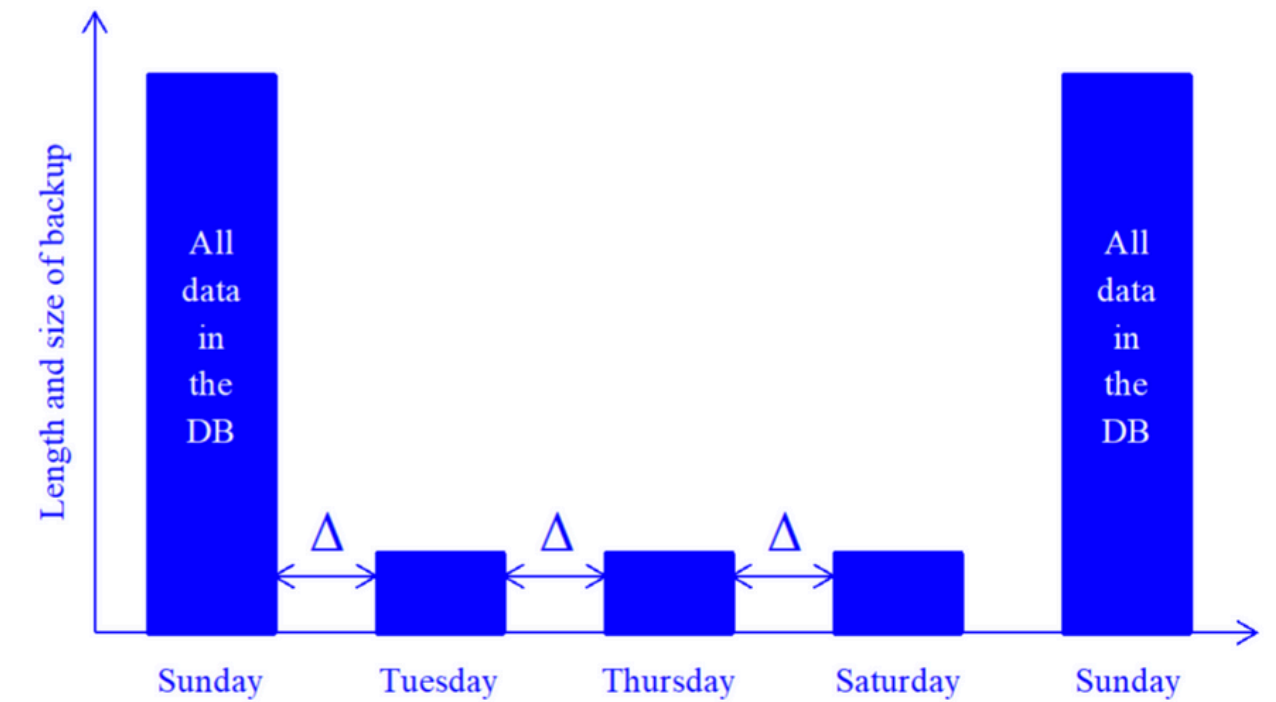
Backup Location

- Onsite:
 - Stored on the same premises, but not the same machine as the DB
- Offsite:
 - Backup NOT physically near disaster site (generally (>160 km away))
 - Enables disaster recovery



Backup Scale

- Full:
 - Back up entire DB
 - Very costly, takes a lot of time
- Incremental:
 - Only backs up the changes since the last backup
 - Smaller in size, shorter time
- Backup policies generally use a combination of full & incremental



Data Recovery

Generally in two phases:

1. Restore backup to the database server machine.
2. Recovery up until the point of the database failure
 - Known as a full recovery
 - No committed data in the database should be lost

2.

ACME Manufacturing makes widgets in its factory. The factory runs 24 hours a day, 7 days a week in three shifts. The quietest shift is the Sunday night shift, which runs from midnight Sunday to 8am Monday. While ACME manufactures widgets, the database must run. This is ACME's only widget factory.

The database administrator has implemented a backup policy that takes a full backup every Sunday at 3am during the night shift, and then an incremental backup on Tuesday, Thursday and Saturday mornings at 3am.

The backup strategy has determined that if there is a database failure, restoration of the database is time-critical. ACME must have the shortest outage time to restore and recover the database. This means the database must be restored quickly so that the manufacturing can continue. ACME must have the smallest elapsed time from the point of failure to the database being fully operational and useable.

2.(a)

ACME Manufacturing makes widgets in its factory. The factory runs 24 hours a day, 7 days a week in three shifts. The quietest shift is the Sunday night shift, which runs from midnight Sunday to 8am Monday. While ACME manufactures widgets, the database must run. This is ACME's only widget factory.

The database administrator has implemented a backup policy that takes a full backup every Sunday at 3am during the night shift, and then an incremental backup on Tuesday, Thursday and Saturday mornings at 3am.

The backup strategy has determined that if there is a database failure, restoration of the database is time-critical. ACME must have the shortest outage time to restore and recover the database. This means the database must be restored quickly so that the manufacturing can continue. ACME must have the smallest elapsed time from the point of failure to the database being fully operational and useable.

Given the business requirements and the database administrator's backup policy, what database backup **type**, **mode** and **site** would you recommend?

2.(a)

ACME Manufacturing makes widgets in its factory. The factory runs 24 hours a day, 7 days a week in three shifts. The quietest shift is the Sunday night shift, which runs from midnight Sunday to 8am Monday. While ACME manufactures widgets, the database must run. This is ACME's only widget factory.

onsite

must be online

The database administrator has implemented a backup policy that takes a full backup every Sunday at 3am during the night shift, and then an incremental backup on Tuesday, Thursday and Saturday mornings at 3am.

The backup strategy has determined that if there is a database failure, restoration of the database is time-critical. ACME must have the shortest outage time to restore and recover the database. This means the database must be restored quickly so that the manufacturing can continue. ACME must have the smallest elapsed time from the point of failure to the database being fully operational and useable.

physical

Given the business requirements and the database administrator's backup policy, what database backup **type**, **mode** and **site** would you recommend?

2.(b)

Consider the Full and Incremental backup timeline in Figure 1. If the database suffered a media failure on Friday at 9:23am, how many backups would need to be restored?

Need to restore:

- Latest full DB update
- All subsequent incremental updates
- The DBA would also replay the crash recovery logs from Thursday morning until Friday 9:23am

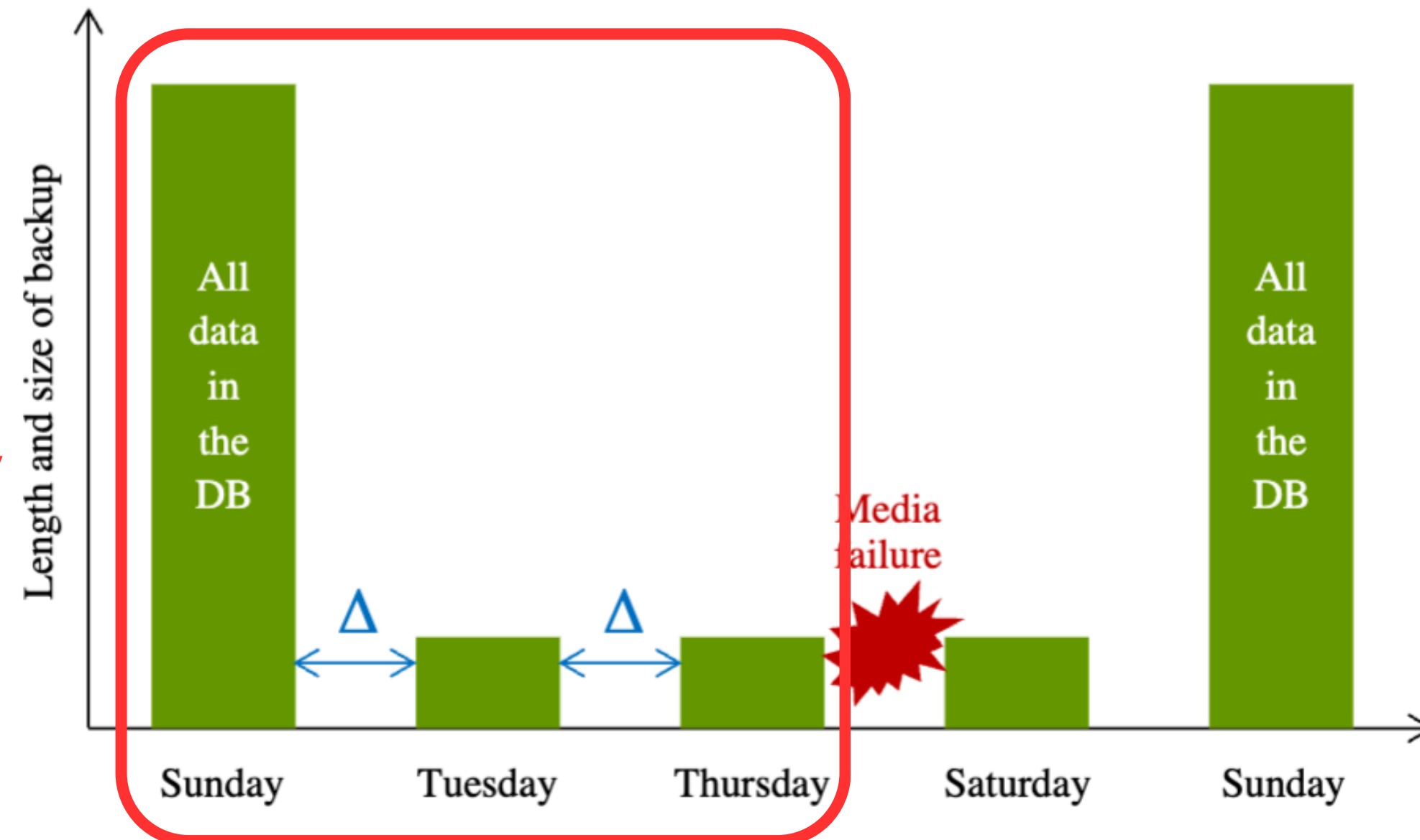


Figure 1. A timeline of full and incremental backups showing the media failure on Friday morning.

2.(c)

Given the same failure, what would be the benefits and costs of changing the backup strategy to do **full backups** on Sunday, Tuesday, Thursday and Saturday mornings at 3am?

- Benefits
 - Reduce the time to fully restore the database when failure occurs
- Costs
 - More space needed
 - More data to copy → backup takes longer
 - Risk that if we don't remove backups from the database server, we could fill up the file system

Transactions

What are Transactions?

- Logical unit of work that must be either **entirely COMPLETED or ABORTED** → *atomic*
- **Single task** that may involve several changes to the database
 - Single task, can't be half-done
 - e.g. removing an employee & all their data
 - e.g. Sending \$1M to my bank account
- Adhere to the ACID principles

ACID Principles

1. Atomic

- Each transaction must **entirely** either succeed / fail
- Any mid-transaction failures will be rolled back

2. Consistent

- Upon completion, data in DB must be consistent (data integrity)

3. Isolated

- Changes made in transaction A cannot be seen from within B

4. Durable

- Upon completion, all actions (CRUD) must persist permanently

01

Lost Update Problem

02

Uncommitted Data Problem

03

Inconsistent Retrieval Problem

Concurrency Problems

DB needs to support concurrency, allowing multiple users to use it simultaneously. However, these problems need to be considered:

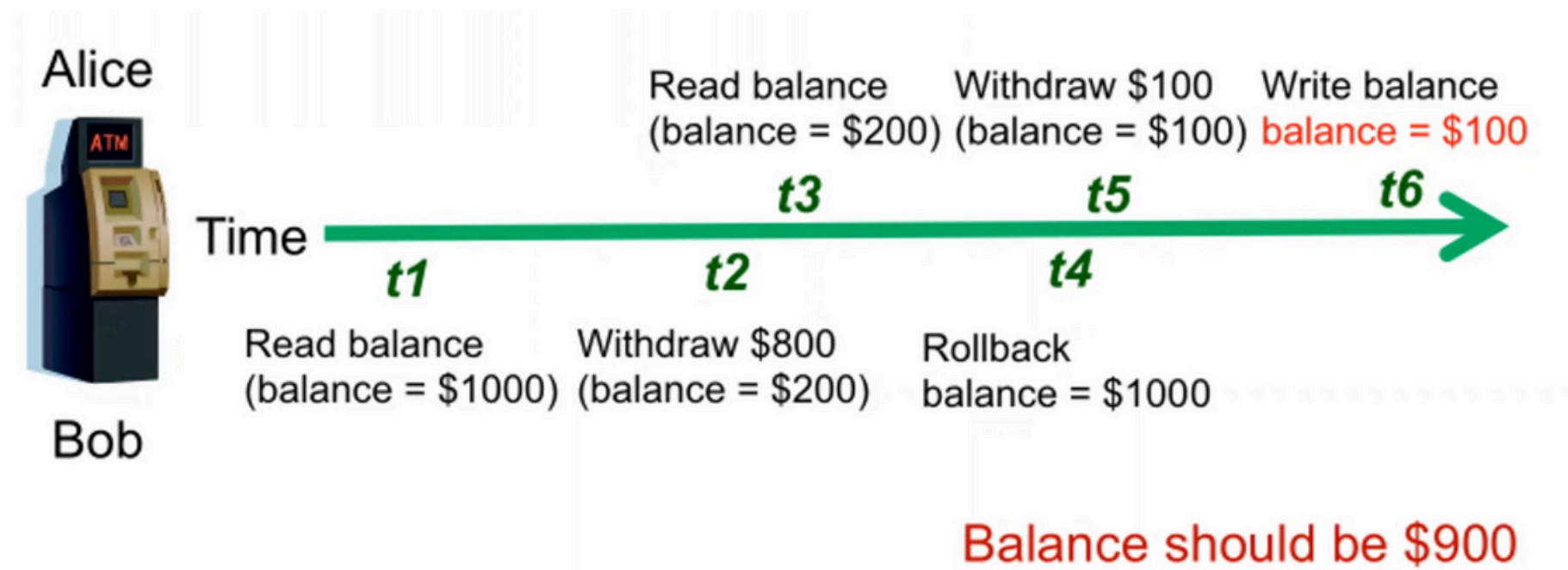
1. Lost Updates

- 2 users (A, B) wants to update the same piece of data at the same time
 - B's update ignores and overwrites user A's update
- e.g. A's bank account contains \$500. A buys a \$300 TV, while at exactly the same time, B pays A \$120. Instead of being \$180 poorer, A becomes \$120 richer (no TV costs deducted)!

My transaction at the electronics store	My employer's transaction
<i>Read</i> account balance (\$500)	
	<i>Read</i> account balance (\$500)
<i>Write</i> account balance less \$300 (\$200)	
	<i>Write</i> account balance plus \$120 (\$620)

2. Uncommitted Data

- A updates a value, which is used by B, but then A rolls back
 - Violates **isolated** property of transaction
 - **Inconsistent** state: A & B's base value read is different



2. Inconsistent Retrievals

- One transaction calculates some aggregate functions over a set of data, while other transactions are updating the data
- Some data may be read after they are changed and some before they are changed, yielding inconsistent results

Alice	Bob
SELECT SUM(Salary) FROM Employee;	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 33;
	UPDATE Employee SET Salary = Salary * 1.01 WHERE EmpID = 44;
(finishes calculating sum)	COMMIT;

Q3.

It's class registration day, when UniMelb students register in tutorial classes for the upcoming semester. In one particular subject, each tutorial class can fit a maximum of 24 students.

Eamonn and Jacqueline both wish to register in the Wednesday 10am tutorial class for this subject. This class already has 23 students enrolled – just one place remains.

Suppose the database contains tables like this:

FK
TutorialClass (SubjectCode, TutorialNumber, TotalEnrolments)

FK FK FK
TutorialEnrolment (SubjectCode, TutorialNumber, StudentNumber)

- a. Describe how a lost update could occur in this database when Eamonn and Jacqueline try to simultaneously register in the Wednesday 10am tutorial.
- b. How could the lost update problem be avoided in this situation?

Q3.(a)

It's class registration day, when UniMelb students register in tutorial classes for the upcoming semester. In one particular subject, each tutorial class can fit a maximum of 24 students.

Eamonn and Jacqueline both wish to register in the Wednesday 10am tutorial class for this subject. This class already has 23 students enrolled – just one place remains.

Suppose the database contains tables like this:

FK
TutorialClass (SubjectCode, TutorialNumber, TotalEnrolments)

FK FK FK
TutorialEnrolment (SubjectCode, TutorialNumber, StudentNumber)

- a. Describe how a lost update could occur in this database when Eamonn and Jacqueline try to simultaneously register in the Wednesday 10am tutorial.

Q3.(a)

- Suppose Eamonn's request is received a split second before Jacqueline's.
- The server might execute the operations in this order:
 - (Any order is possible so long as Jacqueline's Read is before Eamonn's Write)

Eamonn	Jacqueline
<i>Read</i> TutorialClass.TotalEnrolments (23)	
	<i>Read</i> TutorialClass.TotalEnrolments (23)
<i>Insert</i> row into TutorialEnrolment	
	<i>Insert</i> row into TutorialEnrolment
	<i>Write</i> TutorialClass.TotalEnrolments (24)
<i>Write</i> TutorialClass.TotalEnrolments (24)	

- Even though there are now 25 students enrolled in the class, TotalEnrolments=24
- **Lost update**

Q3.(b)

How could the lost update problem be avoided in this situation?

- **Serial execution?**
 - Inefficient (very slow)
 - However, guarantees that isolation is satisfied
- **Locking**
 - More efficient
- **Timestamp**
- **Optimistic concurrency control** (check at commit time)