



Backup and Recovery

For
Postgres Women India Upskill Program

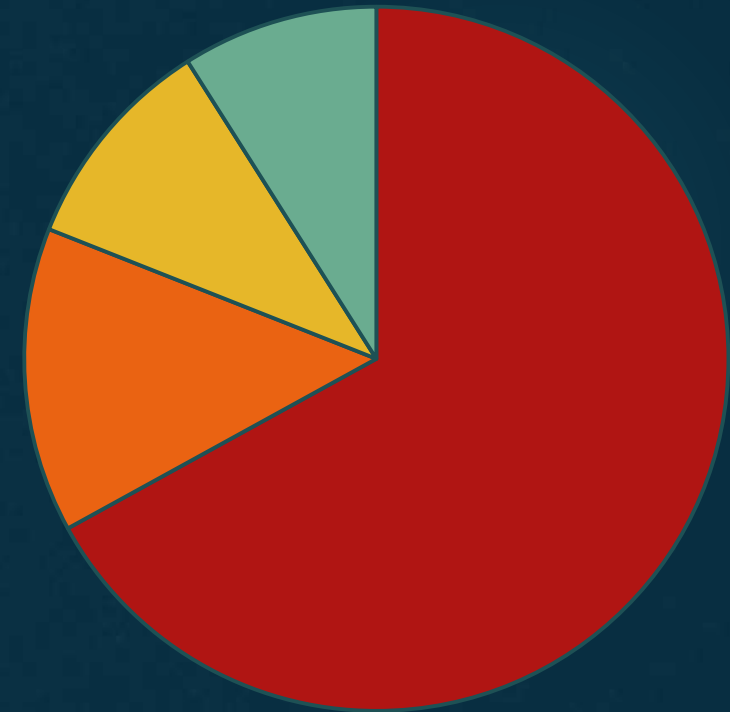
Agenda

- Introduction – ~10 min
- Logical Backup & Demo – ~40 min
- Physical Backup Introduction – 45 min
- Physical Backup Demo – 40 min
- Introduction to streaming replication – 5 min
- pgBackRest – 10 min
- QnA - ??

Why Backup and Recovery Matters?

- A backup is a process of creating a copy of data from a system or application so that it can be restored in case of data loss, corruption, or disaster. Backups are essential for ensuring data durability, business continuity, and recovery from accidental or malicious events.
- In PostgreSQL, a backup refers to capturing a consistent copy of the database's data—either in a **logical format** (SQL commands) or a **physical format** (binary files)—so that it can be restored to recover the database to a previous, stable state. Backups can be full, incremental, or continuous using WAL archiving.

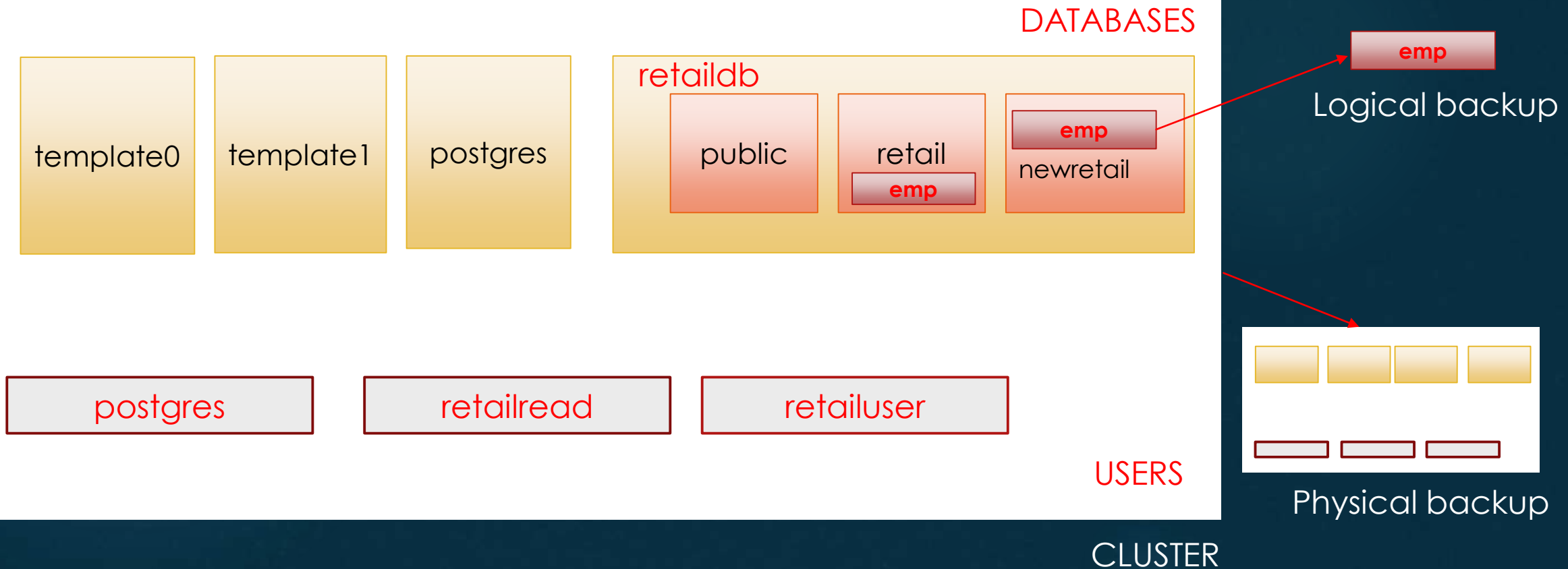
Reasons for Data loss



■ Hardware failure ■ Human Error ■ Software issues ■ Others

Ref: <https://www.neweratech.com/nz/blog/10-common-causes-of-data-loss/>

Databases, schemas, Users



Types: Logical vs Physical Backups

Feature	Logical Backup	Physical Backup
Use Case	Migrations, Deployment refresh	Disaster recovery, full system restore
What is backed up	Data and schema (SQL commands)	Raw data files and directories
Tooling	pg_dump, pg_dumpall, psql, pg_restore	pg_basebackup, OS file tools, pgbackrest, barman
Speed (Backup)	Slower	Faster
Size	Potentially smaller	Relatively equal
Granularity	Per database/table level	Primarily entire database cluster
Portability	Higher (across versions/systems)	Lower (version and architecture-specific)
PITR	No	Yes with WALs
Online Backup	Should be online	Cold backup can be offline
Complexity	More complex restore for full DB	Simpler restore for full DB

Logical Backups

Generate a text file with SQL commands

- ▶ PostgreSQL provides the utility program **pg_dump** for this purpose
- ▶ Dumps created by **pg_dump** are internally consistent, that is, the dump represents a **snapshot of the database** as of the time **pg_dump** begins running.
- ▶ Dump **doesn't include system tables** and configuration files.
- ▶ **Logical backup requires the server to be running**
- ▶ Logical backup is basically a **select operation** on the tables.

Entire Cluster - SQL Dump

- ✓ pg_dumpall is used to dump an entire database cluster in **plain-text SQL format** (only)
- ✓ Dumps global objects - users, groups, and associated permissions
- ✓ It does not allow compression or parallelization.
- ✓ Use psql to restore

Syntax:

```
$ pg_dumpall [options...] > filename.sql
```

How to restore?

Using psql client

- ▶ Backups taken using pg_dump with plain text format(Fp)
- ▶ Backups taken using pg_dumpall

Using pg_restore utility

- ▶ Backup taken using pg_dump with custom(Fc),tar(Ft) or director(Fd) formats
- ▶ Supports parallel jobs for during restore
- ▶ Selected objects can be restored

Tools for logical backup

`pg_dump`, to backup only instance databases, completely or partially, with many options and formats;

```
[postgres@lab02 ~]$ pg_dump --help
```

```
pg_dump dumps a database as a text file or to other formats.
```

Usage:

```
pg_dump [OPTION]... [DBNAME]
```

`pg_dumpall` to save all database definitions and data in a single SQL script, as well as global objects (roles, tablespaces).

```
[postgres@lab02 ~]$ pg_dumpall --help
```

```
pg_dumpall extracts a PostgreSQL database cluster into an SQL script file.
```

Usage:

```
pg_dumpall [OPTION]...
```

Important pg_dump options

-a	- Data only. Do not dump the data definitions (schema)
-s	- Data definitions (schema) only. Do not dump the data
-n <schema>	- Dump from the specified schema only
-t <table>	- Dump specified table only
-f <file name>	- Send dump to specified file. Filename can be specified using absolute or relative location
-Fp	- Dump in plain-text SQL script (default)
-Ft	- Dump in tar format
-Fc	- Dump in compressed, custom format
-Fd	- Dump in directory format
-j njobs	- dump in parallel by dumping n jobs tables simultaneously.
-Z --compress=0-9	- Compression level
-v	- Verbose option

Tools for logical backup restore..

psql, if the backup is taken in plain text format

```
[postgres@lab02 ~]$ psql -f <filename.sql>
```

pg_dumpall to save all database definitions and data in a single SQL script, as well as global objects (roles, tablespaces).

```
[postgres@lab02 ~]$ pg_restore --help
```

pg_restore restores a PostgreSQL database from an archive created by pg_dump.

Usage:

```
pg_restore [OPTION]... [FILE]
```

Physical Backup

- ▶ Takes the copy of your cluster to backup location
- ▶ Command to take physical backup
 - ▶ `Pg_basebackup -D /u01/backup -Ft -checkpoint=fast`
- ▶ Creates two folders, base.tar and wal.tar, base.tar contains your entire cluster and wal.tar contains wals generated during backup
- ▶ Restoring backup is nothing but extracting the backup piece and using that directory.
- ▶ Recovery is nothing but applying the transactions logs on top of restored backup piece.

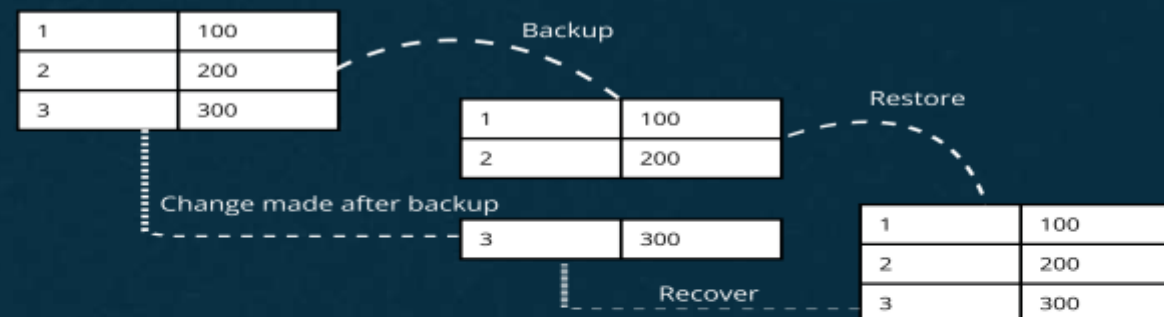
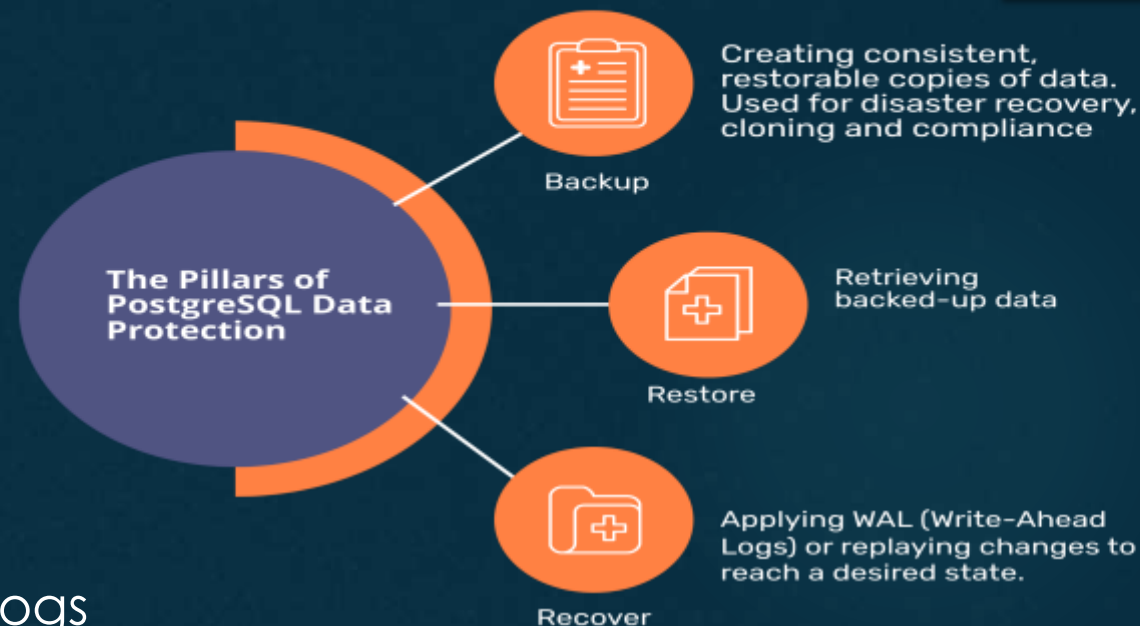
Key components of Physical backup

Backup: Creating copies of data for safekeeping.

Restore: Retrieving backed-up data.

Recover: Returning the database to a functional state by recovering archive logs

You need at least one full backup to recover archive logs.



Insert the record

```
postgres=# create table test (id int,sal int);
```

```
CREATE TABLE
```

```
postgres=#
```

```
postgres=# insert into test values(1,100);
```

```
INSERT 0 1
```

```
postgres=# select xmin, * from test;
```

```
xmin | id | sal
```

```
-----+-----+-----
```

```
768 | 1 | 100
```

```
(1 row)
```

```
768 | 1 | 100
```

768 1 100	

update the record

```
postgres=# update test set sal=200 where id=1;
```

```
UPDATE 1
```

```
postgres=# select xmin, * from test;
```

```
xmin | id | sal
```

```
-----+-----+-----
```

```
769 | 1 | 200
```

```
(1 row)
```

```
769 | 1 | 200
```

```
769 | 1 | 200
```

```
768 | 1 | 100
```

update the record

```
postgres=# update test set sal=300 where id=1;
```

```
UPDATE 1
```

```
postgres=# select xmin, * from test;
```

```
xmin | id | sal
```

```
-----+-----+-----
```

```
770 | 1 | 300
```

```
(1 row)
```

```
770 | 1 | 300
```

```
770 | 1 | 300
```

```
769 | 1 | 200
```

```
768 | 1 | 100
```

update the record

```
postgres=# update test set sal=400 where id=1;
```

```
UPDATE 1
```

```
postgres=# select xmin, * from test;
```

```
xmin | id | sal
```

```
-----+-----+-----
```

```
771 | 1 | 400
```

```
(1 row)
```

```
771 | 1 | 400
```

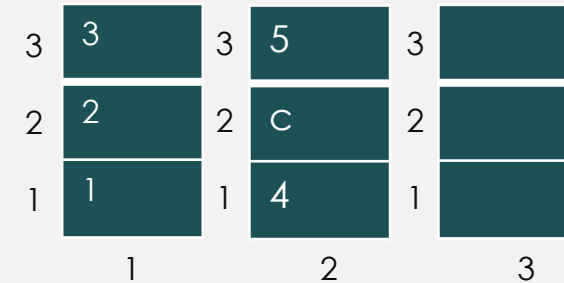
770 1 300	...
769 1 200	...
768 1 100	771 1 400

Checkpoint and LSN

Checkpoint ensures the change reached datafile.

LSN uniquely identifies the transaction.

- LSN for transaction 2 is 1/2
- LSN for transaction 5 is 2/3
- LSN for next trx is 3/1



WAL Files

Checkpoint and LSN

- Checkpoint is still at 2/2
- Transactions 1,2,3 are replaced by 9, 10 and 11 because of max_wal_size limit.
- I cannot take my database back to 1 or 2 or 3.



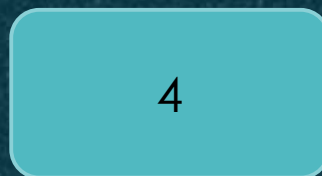
Datafile

3	3 11	3	5	3	8
2	2 10	2	C	2	7
1	1 9	1	4	1	6
	1		2		3

WAL Files

Enable archive logging

- Checkpoint is still at 2/2
- Transactions 1,2,3 are replaced by 9, 10 and 11 because of max_wal_size limit.
- I cannot take my database back to 1 or 2 or 3.



Datafile

WAL Files

3	3- 11	3	5	3	8
2	2- 10	2	C	2	7
1	1- 9	1	4	1	6
	1		2		3

3	3
2	2
1	1
	1

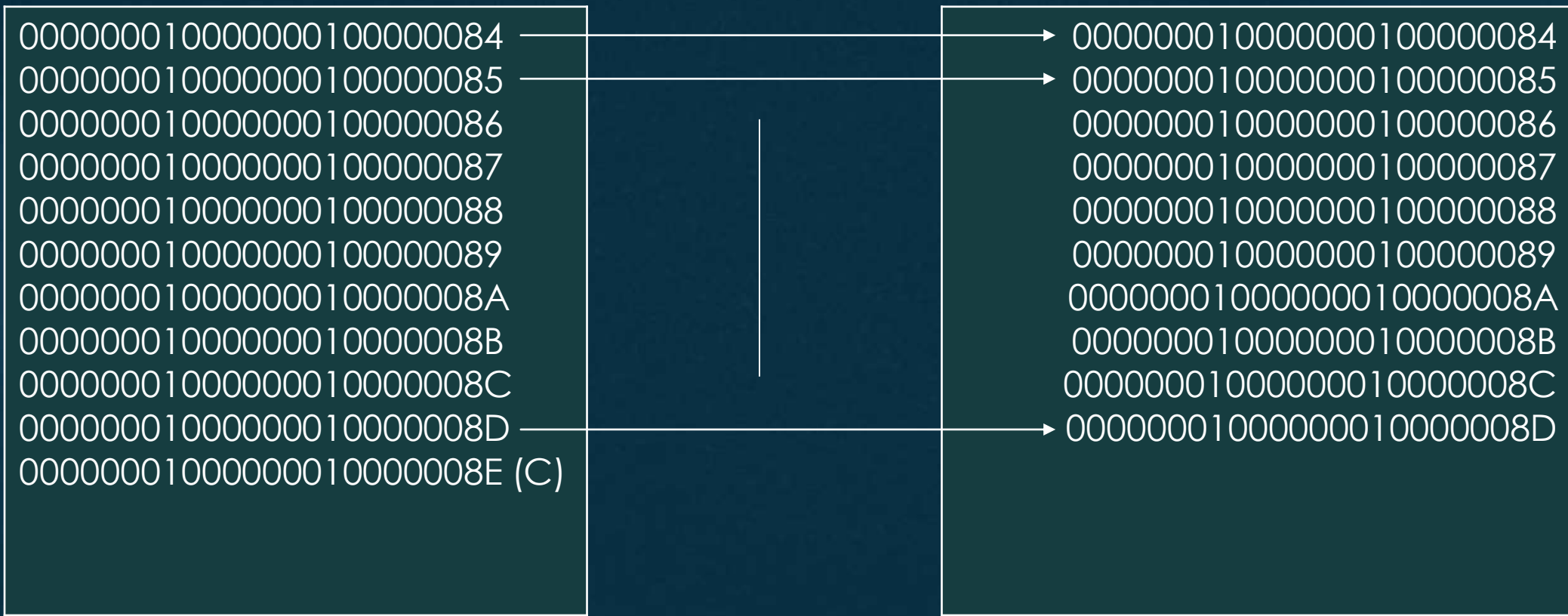
Archived WAL

Enable archive logging

Is a copy of your wals (transaction log backup)

```
archive_mode=on;
```

```
archive_command='cp %p /<archive_location>/%f'
```



RPO & RTO

8:00 AM — Point of Last Backup

This is the last time a successful backup was taken.

8:15 AM — Changes Made

New data or transactions are added to the system.

8:30 AM — Accepted RPO

This is the organization's defined threshold for how much data loss is acceptable.

8:45 AM — Disaster Occurs (Start of RTO)

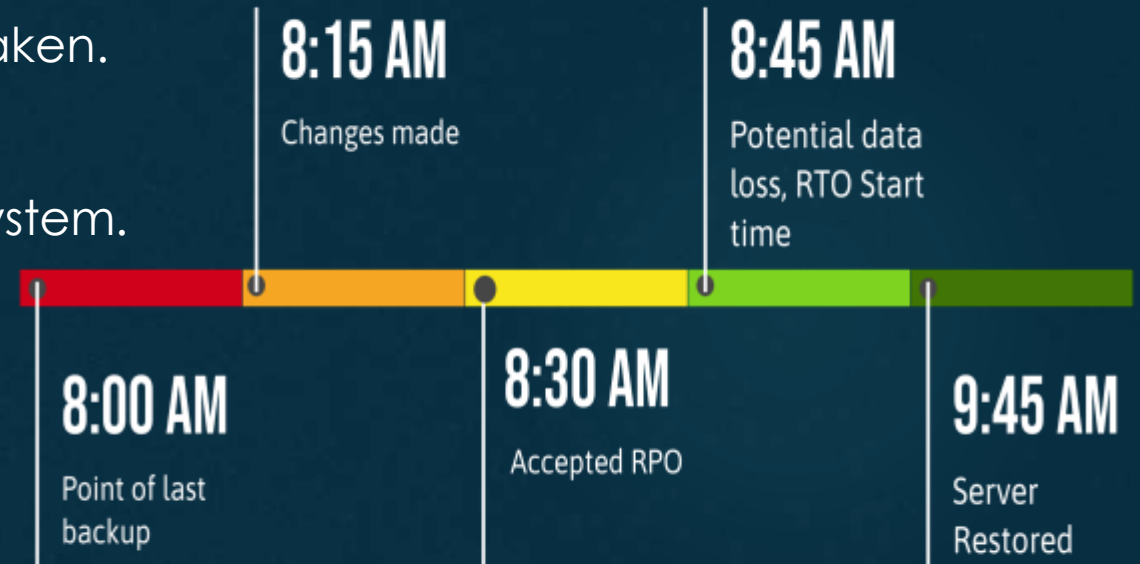
RTO starts here — the clock is ticking to bring the system back online.

9:45 AM — Server Restored

Recovery is complete.

Total downtime = 1 hour.

RTO = 1 hour.



- RPO is about how much data you can afford to lose. (15 min)
- RTO is about how quickly you must be back online. (1 hr)

How many backups I need to retain?

- ▶ Full backup = 1- Current LSN
- ▶ 7th Full backup = 1-41
- ▶ 14th Full backup = 1-76
- ▶ 20th Full backup = 1-94

1. I want you to take my database to anytime in the past of 14 days.
2. $24 - 14 = 11$ th

St 1. Take my database to 11th

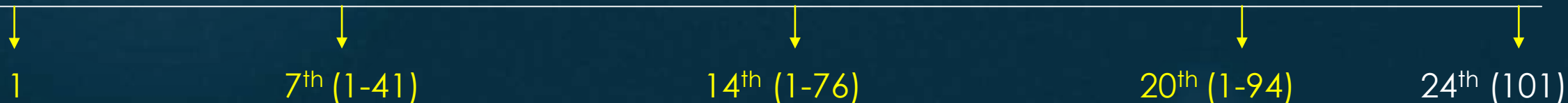
St 2. Take my database to 16th

St 3.

- Do I need 7th backup?
- If it is 16th, Do I need 7th?

St 4. I lost 23rd WAL, what to do?

1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 14 - 15 - 16 - 17 - 18 - 19 - 20 - 21 - 22 - 23 - 24



Exercise

What happens when you initiate backup

- Initiated backup at 84
- Issue log switch (85)
- Take the database datafiles backup
- Backup finished when my WAL at 8E, which means 85-8E are the WALs generated during backup. (take copy)
- 8E, 90 and 91 are WALs generated after backup.

```
/usr/pgsql-17/bin/pg_basebackup -D /u01/backups/ --checkpoint=fast
```

```
000000010000000100000083
```

```
000000010000000100000084 -> initiated backup
```

```
000000010000000100000085
```

```
000000010000000100000086
```

```
..
```

```
00000001000000010000008E
```

```
000000010000000100000085.backup
```

```
00000001000000010000008F
```

```
000000010000000100000090
```

```
000000010000000100000091
```

Task 1: Take backup & Restore

```
[postgres@lab01 17]$ /usr/pgsql-17/bin/pg_basebackup -D /u01/pgsql/17 --checkpoint=fast -P -h 192.168.140.135
```

Password:

1741483/1741483 kB (100%), 1/1 tablespace

Task 2: Recover archive logs

```
### Create recovery.signal file
```

```
touch recovery.signal
```

```
### Edit postgresql.conf
```

```
restore_command = 'scp 192.168.110.216:/u01/archivelogs/%f %p'
```

Task 3: Restore ... Until my fav. point

```
### Create recovery.signal file
```

```
touch recovery.signal
```

```
### Edit postgresql.conf
```

```
restore_command = 'scp 192.168.110.216:/u01/archivelogs/%f %p'
```

```
recovery_target_lsn = '0/29FFCF90'
```

Task 4: Wait until next archive to come

```
### Create recovery.signal file
```

```
touch standby.signal
```

```
### Edit postgresql.conf
```

```
restore_command = 'scp 192.168.110.216:/u01/archivelogs/%f %p'
```

pgBackRest

✦ Key Features:

- ✓ Full, Differential, and Incremental Backups
- ✓ Built-in Compression & Parallelism
- ✓ Remote Backup & Restore (SSH, dedicated repo servers)
- ✓ Seamless WAL Archiving & Point-In-Time Recovery (PITR)
- ✓ Backup Retention Policies & Expiration
- ✓ Support for Multiple PostgreSQL Clusters

```
sudo -u postgres pgbackrest --stanza=demo stanza-create
```

```
sudo -u postgres pgbackrest --stanza=demo backup
```

```
sudo -u postgres pgbackrest --stanza=demo restore
```

```
sudo -u postgres pgbackrest --stanza=demo --delta \
```

```
--target-timeline=current \
```

```
--type=time "--target=2025-04-23 22:26:16" --target-action=promote restore
```

Further reading

- ▶ <https://www.postgresql.org/docs/current/backup.html>
- ▶ <https://www.enterprisedb.com/postgresql-database-backup-recovery-what-works-wal-pitr>
- ▶ <https://pgdash.io/blog/postgres-incremental-backup-recovery.html>
- ▶ <https://www.linkedin.com/learning/postgresql-backup-and-restore-with-pgbackrest/installing-pgbackrest?u=76279468>