Upgrades Don’t Have To Hurt

Ants Aasma

pgday.at 2025

Hello

About me

What This Talk Is About

- Why PostgreSQL requires an upgrade process.
- Methods for upgrading the database.
- What problems you can run into.
- Which tools are available.

What This Talk Is Not About

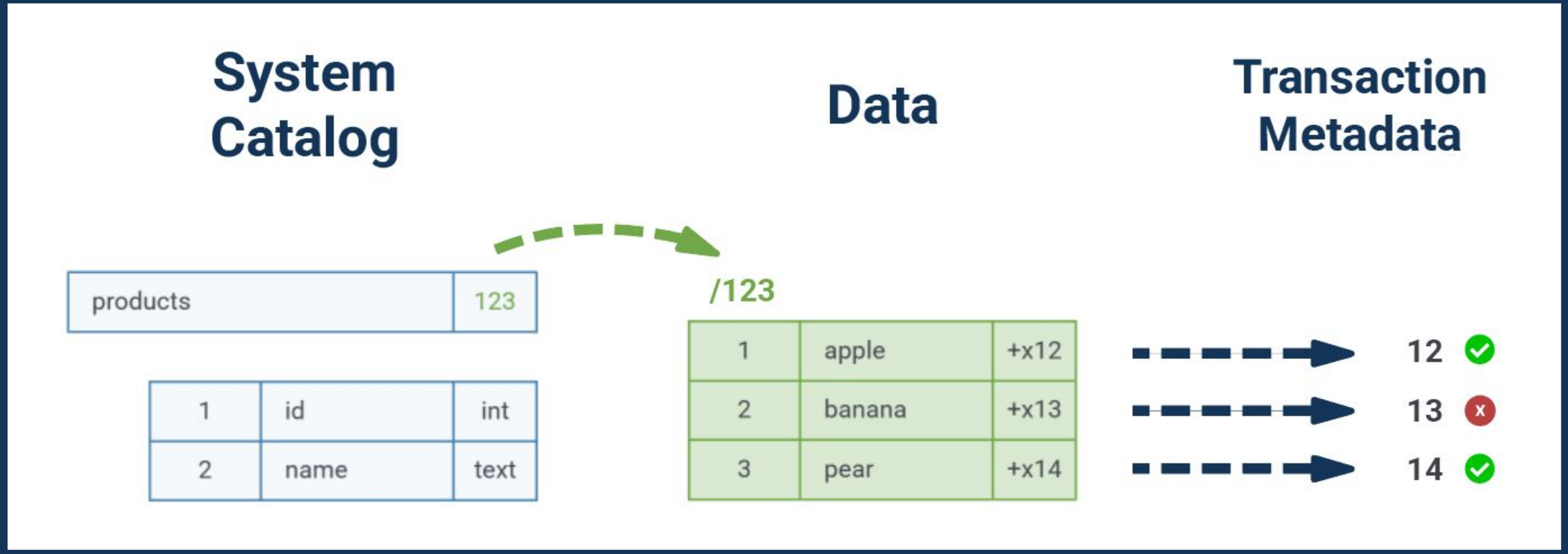
- We are talking about major version upgrades - this means feature releases.
- Bugfix releases just need a restart.
- You should be applying these as part of standard operations.

This Is About Self Managed Databases

- Typically don’t have access to do it yourself.
- DBaaS providers handle upgrade automation for you.
- Sometimes they do it well, sometimes less so.
- Very similar things happen behind the curtain.

Why Do We Need To Upgrade?

How PostgreSQL Stores Data



More Details

See this blogpost for more in-depth discussion:

Why PostgreSQL major version upgrades are hard by Peter Eisentraut

The Bad Old Days

Export And Import

- pg_dump interprets old catalog into SQL statements.
- New version creates system catalogs from SQL statements.
- Data is transferred using COPY.
- Indexes and constraints are added.

How Much Downtime Do You Need?!

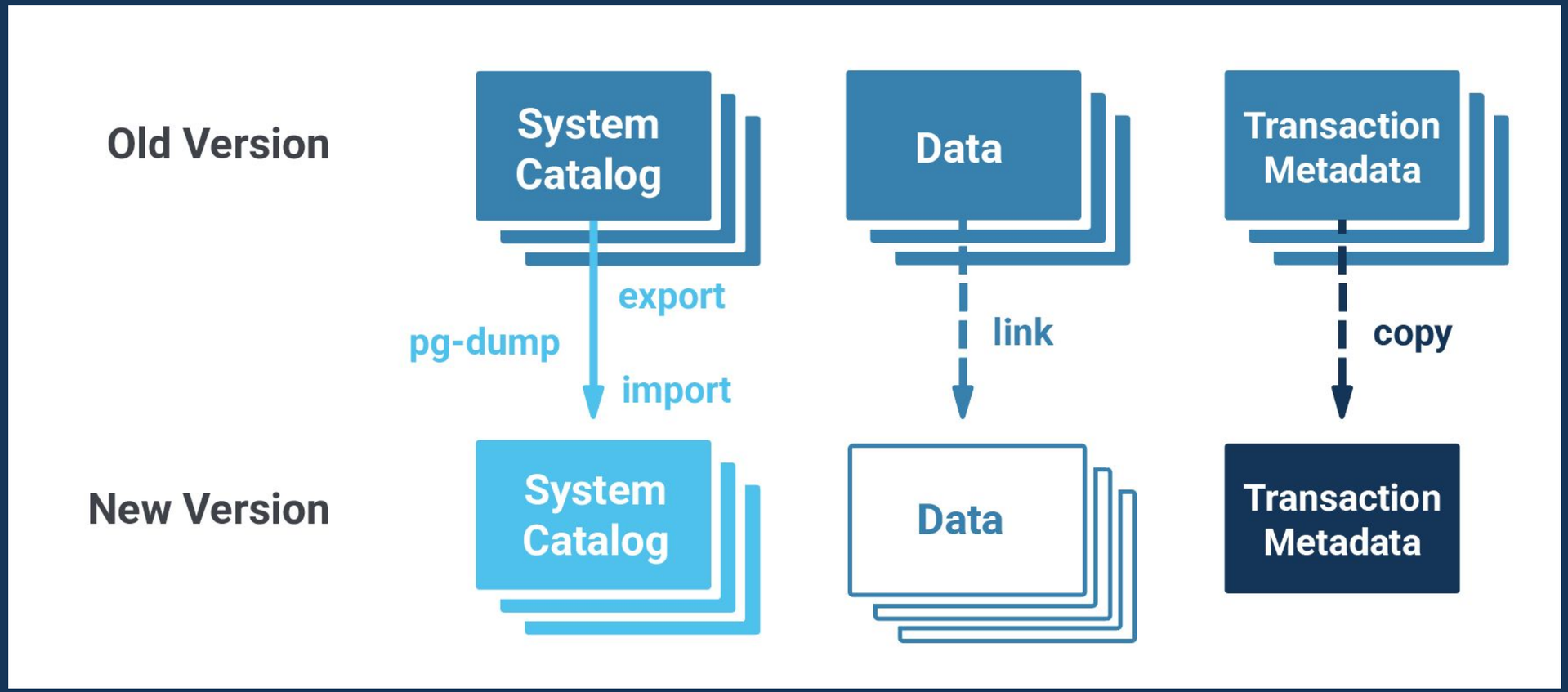
- Hours or days for large databases.
- Application writes must be disabled.
- Maybe acceptable for small databases that are not needed 24/7.

There Must Be A Better Way

- The actual data format doesn’t really change.
  - Rows are rows.
  - Integers are integers.
  - Strings are strings.
- Can we make use of this?

The pg_upgrade Way

Transferring



Database Memory Transplantation

ISO 27001
CERTIFIED

Copy Based Transfer

- Reads and writes all of the data files.
- Double disk space needed temporarily.

Link Mode

- Metadata only operation.
- Enabled by pg_upgrade --link option.
- Source and target must be on the same filesystem.
- Must not touch old version after new version has been started!
- On some filesystems (XFS/Btrfs) --clone and --copy-file-range can do copy on write.

Swap Mode

- New in PostgreSQL 18
- pg_upgrade --swap --sync-method=fsync
- Moves the whole data directory over and injects new system catalog.
- Helps when there are thousands of table and index segments.

Statistics

- Table statistics are not transferred.
- An analysis process had to be run to recalculate statistics.
- Danger of terrible query plans before this.
- Phased to get some rough estimates quickly.
  - Danger: per column statistics can make even the rough estimates take very long.
- Fixed in version 18!

Large Objects Are Bad

- Large objects get transferred by export-import.
- Transferring can take a very long time.
- Prior to version 17, a very large number could break the upgrade.

No pg_downgrade

- New version will use incompatible data formats.
- Backup only helps so much - will lose everything since the backup.
- Test first!

Sanity Check

- pg_upgrade --check is able to catch some errors.
- Runs quickly and with old version still running.
- Run it before the main upgrade process starts.

Upgrading Replicas

- Streaming replication only works with the same major version.
- Replicas must be byte-by-byte identical copies.
- Upgrade process is not deterministic.
- Trying to run pg_upgrade on a replica will result in failure or corruption.

The Brute Force Way

- Recreate replica from a fresh backup.
- Takes quite a bit of time.
- Reduced redundancy while replica reinitialization is underway.

The Hacky Way

- The old version data directory is identical before the upgrade if the replica was caught up. (check LSN!)
- rsync can transfer hard links.
- A special rsync command can copy over system catalogs and redo the hard links on replica.
  - See pg_upgrade documentation.
- Must happen before primary is started, otherwise silent data corruption.
- Unlogged tables will get transferred too.
  - Workaround: truncate before the upgrade.

pg_ugprade Summary

- Not many extra resources needed.
- Usually fast enough.
  - Typically takes from seconds to a few minutes.
- Bad cases can take hours.
- Replica handling is messy.
- One way street.

Logical Replication Upgrades

Online Export And Import

- Logical replication is effectively an export-import, but with changes replicated.
- Works between different PostgreSQL major versions.
- Initial synchronization still takes a long time.

The Big Picture

1. Set up logical replication to target version.
2. Wait for a suitable time.
3. Stop application traffic to source version.
4. Transfer non-replicated objects and wait for catch-up.
5. Point application at target version.

Not Burning Any Bridges

- To retain capability to go backwards, set up reverse replication during switchover.
- With origin=none option this could be set up ahead of time.

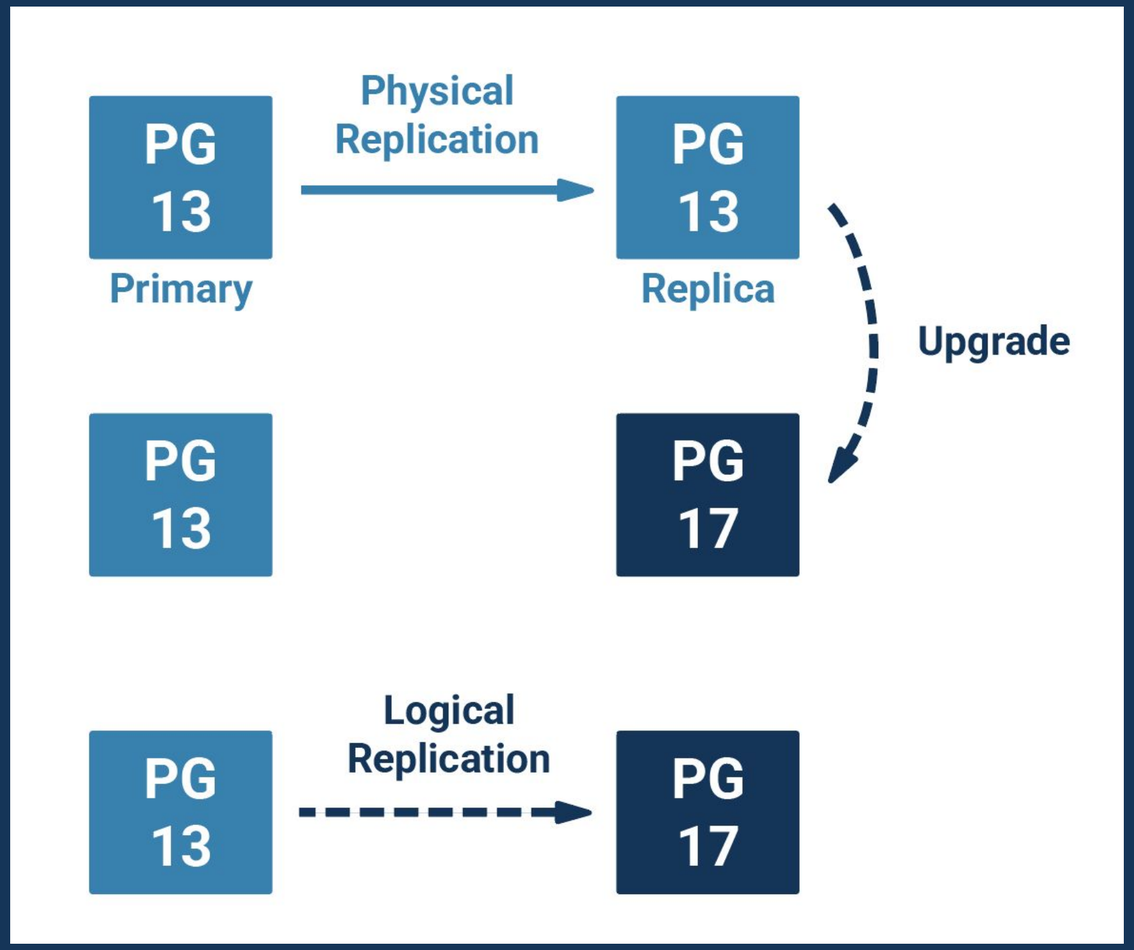
Quiescing The Application

- Many ways to do this.
- pgbouncer and the PAUSE feature can make this easy.

Speeding Up The Replication Setup

- Export and import of huge databases can take way too long.
- Use a streaming replica as a basis.
- Promote and upgrade the replica.
- Set up logical replication to start where physical replication left off.
- Implemented as pg_createsubscriber in version 17

Converting Physical Replica To Logical



Schema Changes Are Not Replicated

- Need to manually apply schema changes in target.
- Typically easier to not change the schema during upgrade process.
- This schema freeze can be enforced using an event trigger.

Sequences

- Sequence values are not incremented with logical replication.
- A quick solution is to sync them during switchover:

```
1| SELECT format('SELECT setval(''%I.%I'', %s);',
2| schemaname, sequencename, last_value)
3| FROM pg_sequences WHERE last_value IS NOT NULL;
```

Other Non-Replicated Objects

- Materialized view contents need to be refreshed on target.
- Large objects are not replicated.
- Unlogged tables will be empty.
- Replication slots for CDC need manual syncing.

Throughput Issues

- One apply worker per subscription doing the work of many backends.
- At best 20k updates/s per apply worker.
- Can split up tables between multiple subscriptions.

Logical Upgrade In A Nutshell

- Unavailability for writes measured in seconds.
  - Can be made to look externally as a small latency spike.
- Need a new server, or sacrifice a streaming replica during the upgrade.
- More moving pieces to manage.
- Very large databases require extreme care.

Things To Worry About

Application Compatibility

- PostgreSQL tries to not break applications unnecessarily.
- Sometimes things do change or deprecated features get dropped.
- Application developers should review release notes of the major version(s)
- Test!

Application Compatibility Examples

- 18: Unlogged partitioned tables are now an error, partitions still fine.
- 17: Functions used in indexes and materialized views must specify search path explicitly.
- 16: plpgsql cursor names are dynamically generated to avoid conflicts.
- 15: Only database owner can create objects public schema by default.
- 14: Custom array functions may need to be dropped and recreated after upgrade.

Plan Changes

- New version has new and better ways to execute queries.
- Flip side: planner has new and more interesting ways to generate bad plans.
- Always a risk, but a higher risk during upgrades.
- Test!
- Have pg_stat_statements, auto_explain and pg_hint_plan ready to fix issues that weren’t caught.

Using pg_hint_plan To Fix Stuff At Runtime

- Example: terrible plan because join between orders and order_events was severely underestimated.

```
1| INSERT INTO hint_plan.hints(query_id, application_name, hints)
2| VALUES (1234567890, '', 'Rows(orders order_events *1000)');
```

Schema Compatibility

- Is the schema pg_dump restorable in the new version?
- May need to get rid of dropped features.
- System catalog references cause issues.
  - Drop and recreate usually enough.
  - pg_monitor role might be an alternative.
- Test!

Extensions

- Need to have in use extensions installed for the new version.
- Some extensions have specific compatibility limitations.
  - PostGIS
  - Timescale
  - May need to upgrade extension before PostgreSQL upgrade.
  - May need to upgrade in several steps.
- Some extensions may need to be dropped and recreated.
- If extension version is updated, need to upgrade the schema:
  - ALTER EXTENSION xyz UPDATE
- Test!

Configuration

- Configuration variables get added and removed between versions.
  - wal_keep_segments -> wal_keep_size in 13
  - Removal of stats_temp_directory in 15
- Some tweaks to configuration may be needed.
- why-upgrade.depesz.com can give you a list.
- Patroni handles per version config customization.

OS Upgrades

- For OS release upgrades you have to be careful about locales.
- If glibc version changes, text indexes may become corrupted and need to be rebuilt.
- glibc 2.28 is especially problematic.
  - EL7 -> EL8+
  - Debian 9 -> 10+
  - Ubuntu 18.04 -> 18.10+
- For more details, see Locale data changes in PostgreSQL wiki.

Automation

Always Automate

- Any manual step introduces a place where a mistake can be made.
- Makes sure testing matches what happens in production.
- Reduces the amount of downtime needed.
- Check as much as you can ahead of time.
- Fail early if anything looks wrong.
- Have a way to roll back or roll forwards from points of failure.

pg_upgradecluster

- Script for single command upgrade.
- Comes with postgresql-common.
- Run pg_upgrade like this:

```
1| pg_upgradecluster --method=upgrade --link 13 main
```

- Does initdb and adapts config files.
- No tablespaces handling.
- Can’t use the rsync trick for replicas.

Spilo

- PostgreSQL container intended to run in and outside of Kubernetes.
- Has a scripted approach to handle major version upgrade.
  - Runs various pre-flight checks.
  - Coordinates shutdown across replicas.
  - Does pg_upgrade --link and then the magic rsync to replicas.
  - Updates extensions and statistics.
  - Cleanup and new backup.
- Handles integration with Patroni.
- Single command to upgrade.

Kubernetes Operators

- Zalando postgres-operator uses Spilo.
- CloudNativePG can run pg_upgrade, but handles replicas using a full copy.
- Crunchy operator runs pg_upgrade as a separate container, requires multiple steps.
- Logical replication also works, but no significant help from any operator.

Future Work

Patroni

- Work underway to port Spilo upgrade process to Patroni.
- Hooks for handling special extensions.
- Goal: patronictl upgrade --version=18

pg_upgrade

- The rsync approach is not very nice.
- pg_upgrade could output a package file to convert up to date replicas to new version.
- On an idea level so far.

Summary

Upgrades That Work

- Test upgrade in development and staging environments.
- Test application on upgraded version.
- Script the whole process.
- Test the scripts.
- Have a backup available.
- Run the upgrade script.
- Initiate a new backup.

Which Option To Use

Can you afford a few minutes of maintenance window per year?

- If yes, just go with pg_uprade.
- If no, you need to invest the time.

“ ”

If it hurts,
do it more often.

Martin Fowler

Thank You

Questions?

Our Partners at PGDay Austria

readyset

redgate

Kachelbunt

Cloudway
IT Consulting & Services

PROVENTA

straDATA

OpenSourceDB
Your Trusted Data Partner

FerretDB

TechSurve

paxia
MULTIPLYING POTENTIAL

metalstack
cloud

Nibble
POSTGRES PROFESSIONALS

zebanza

Open Alliance

For PostgreSQL Education

Your Pathway to Verified PostgreSQL Skills

Scan for Updates

oapg-edu.org