

# Rapid Upgrades With Pg\_Migrator

---

BRUCE MOMJIAN,  
ENTERPRISEDB

August, 2009

***Enterprise*DB™**

## Abstract

Pg\_Migrator allows migration between major releases of Postgres without a data dump/reload. This presentation explains how pg\_migrator works.

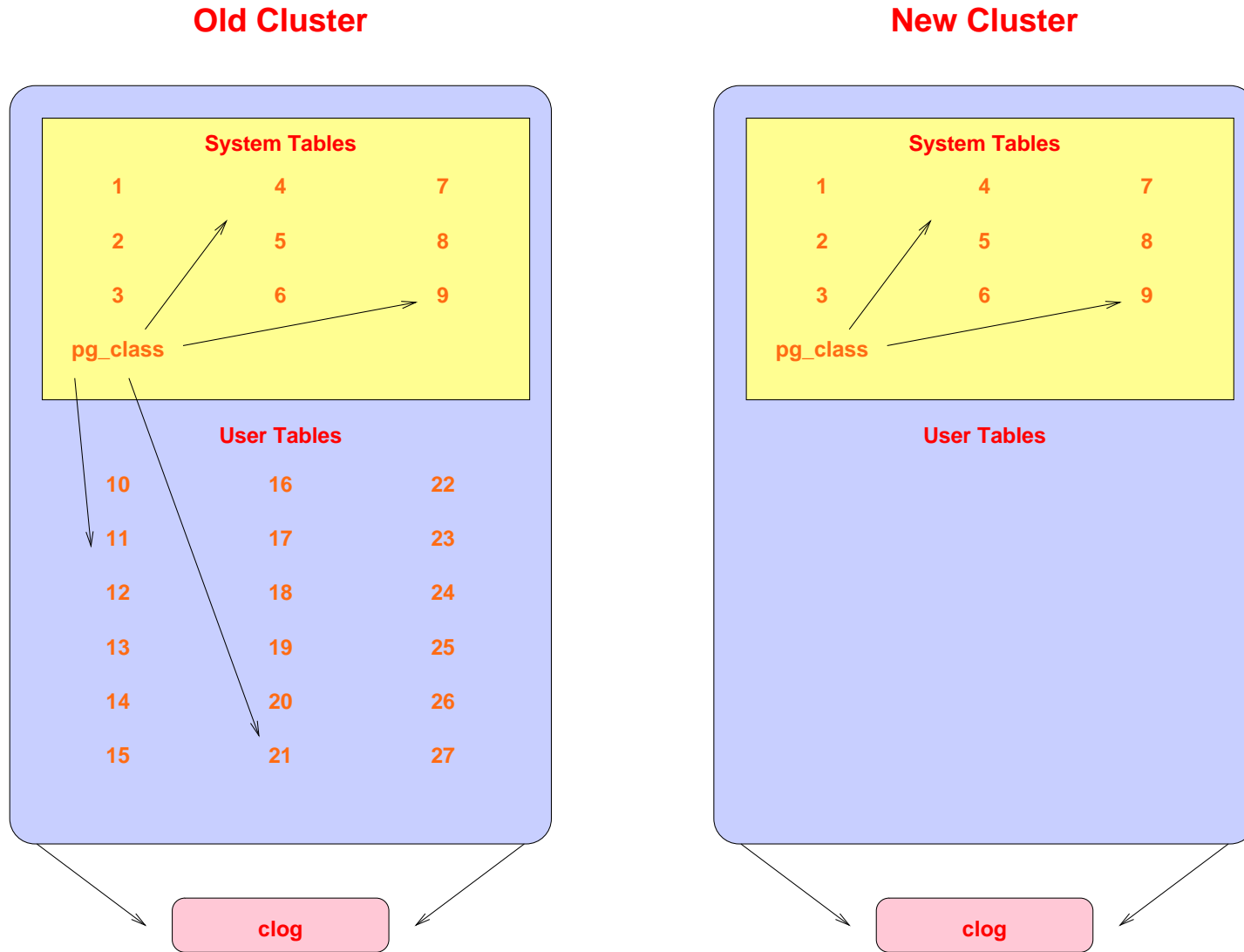
# Why Pg\_Migrator

- Very fast upgrades
- Optionally no additional disk space

# Other Upgrade Options

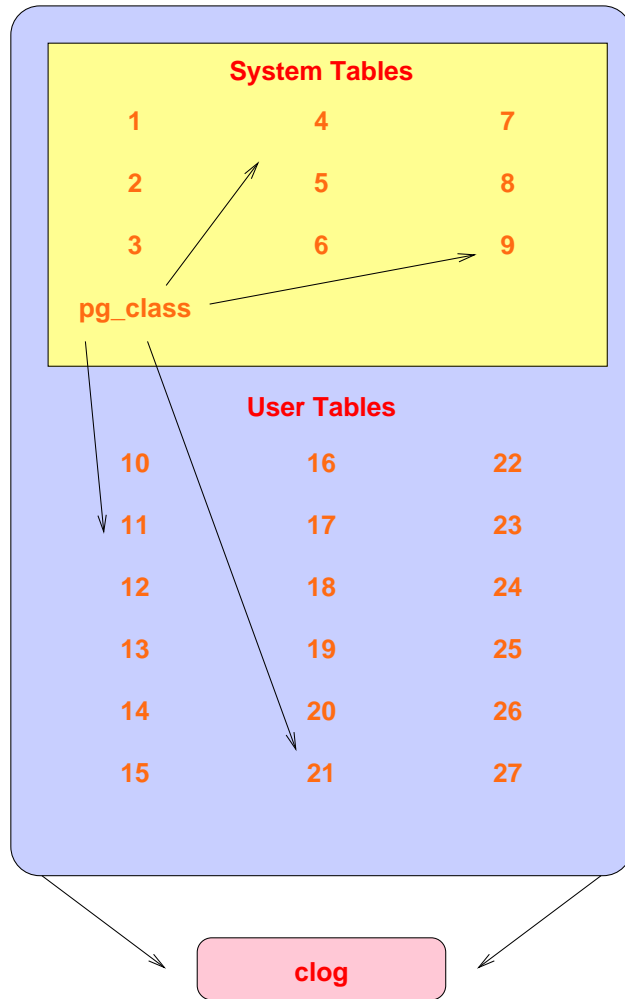
- dump/restore
- Slony

# How It Works: Initial Setup

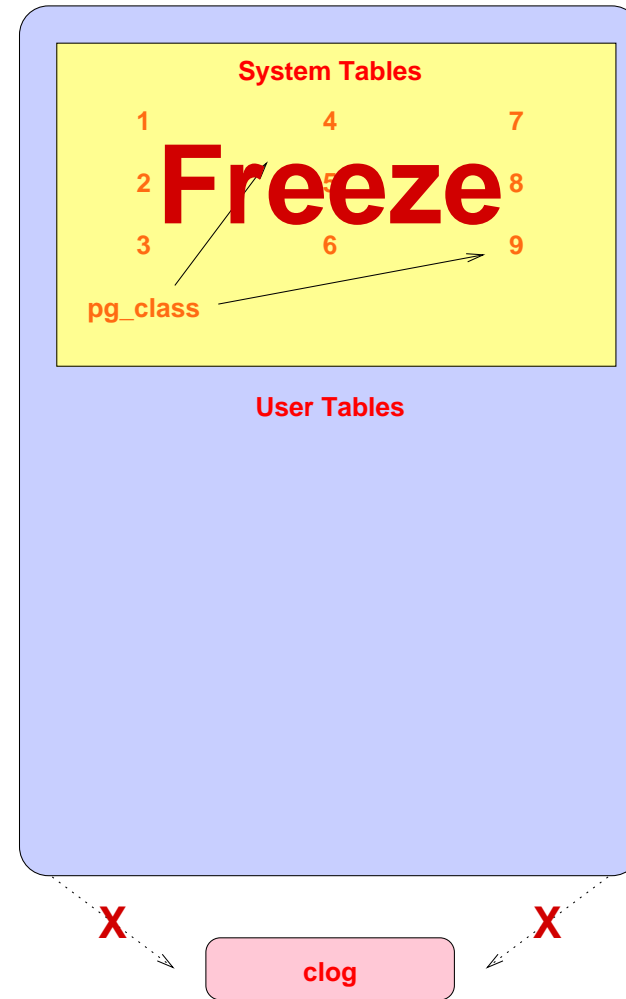


# Decouple New Clog Via Freezing

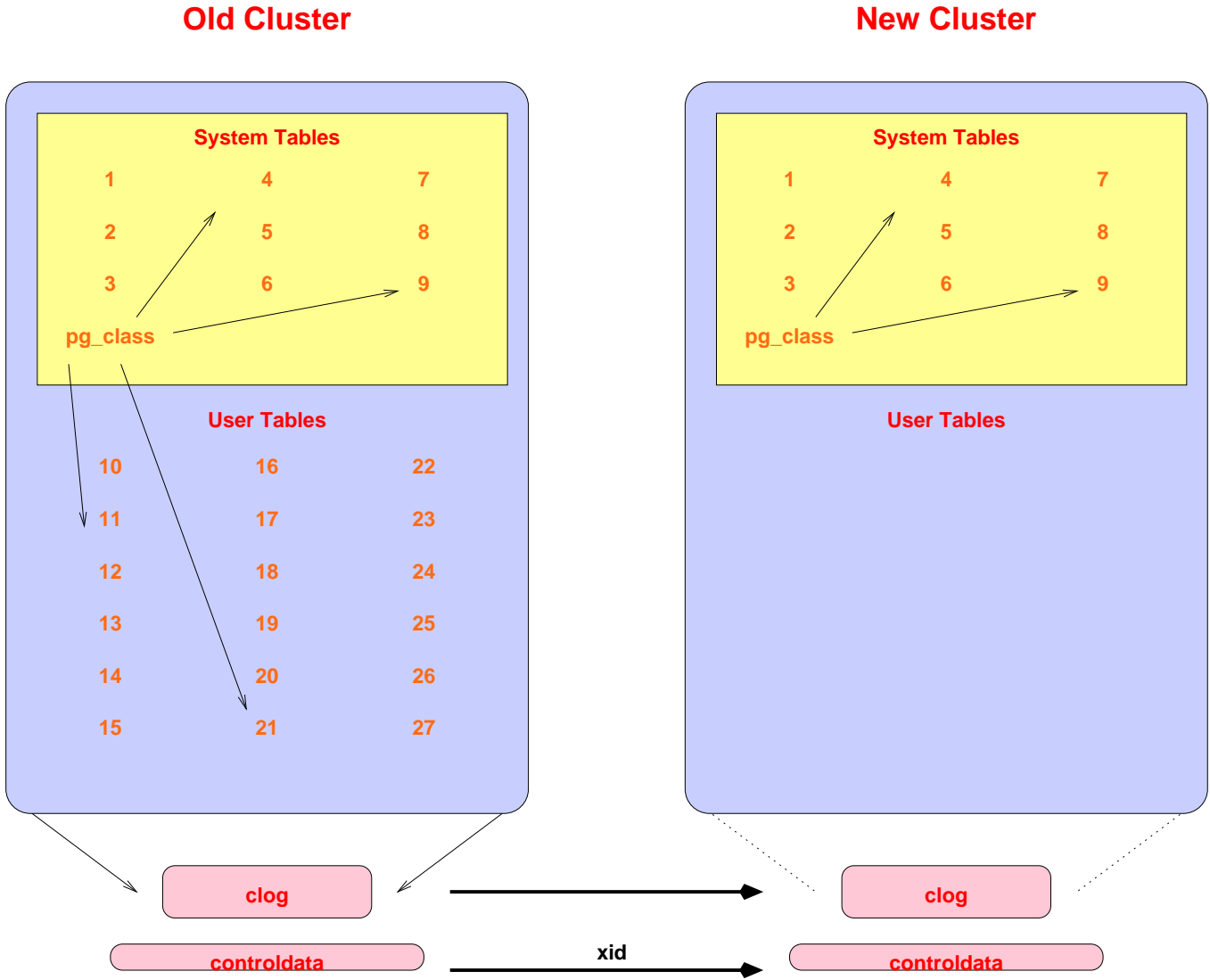
Old Cluster



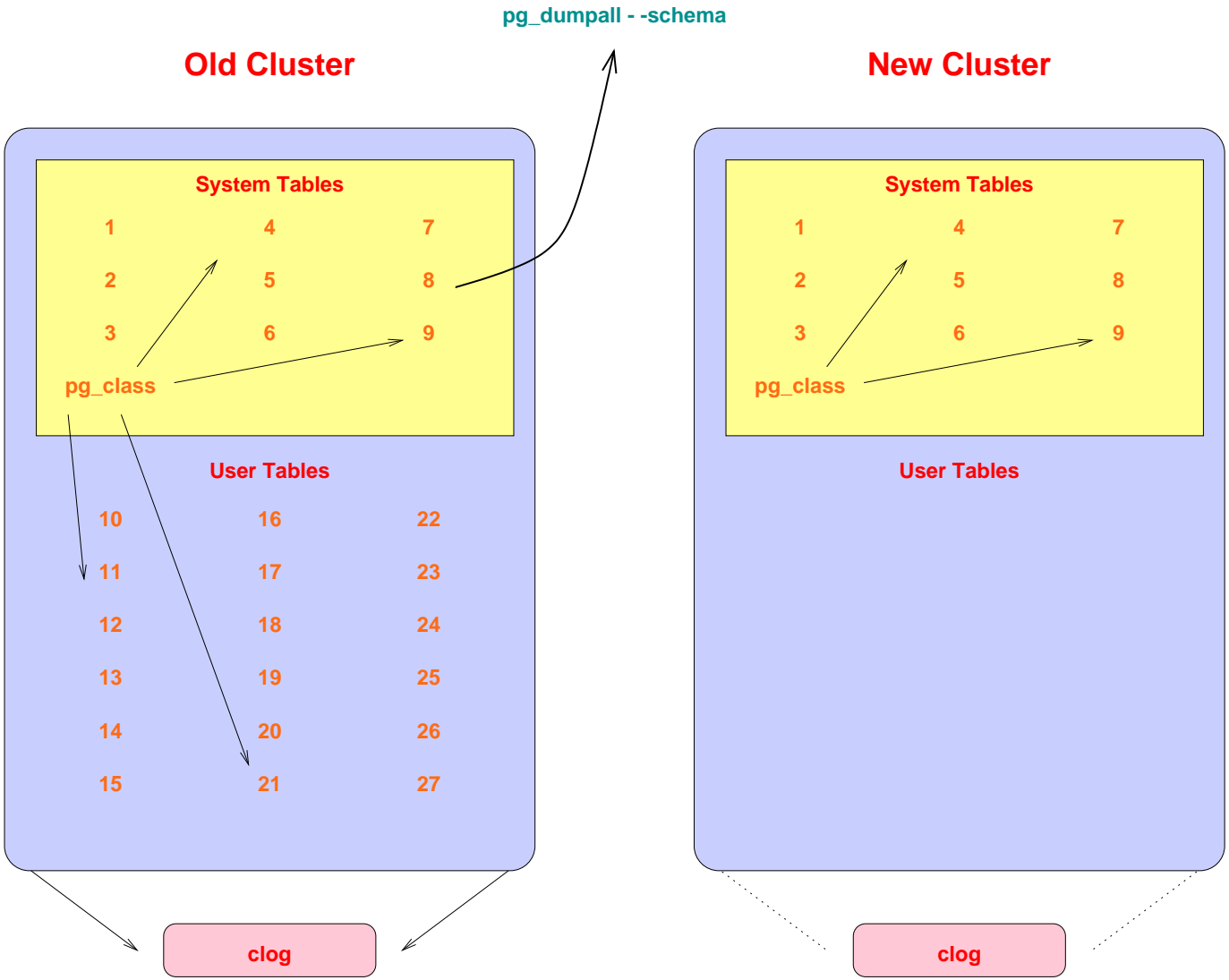
New Cluster



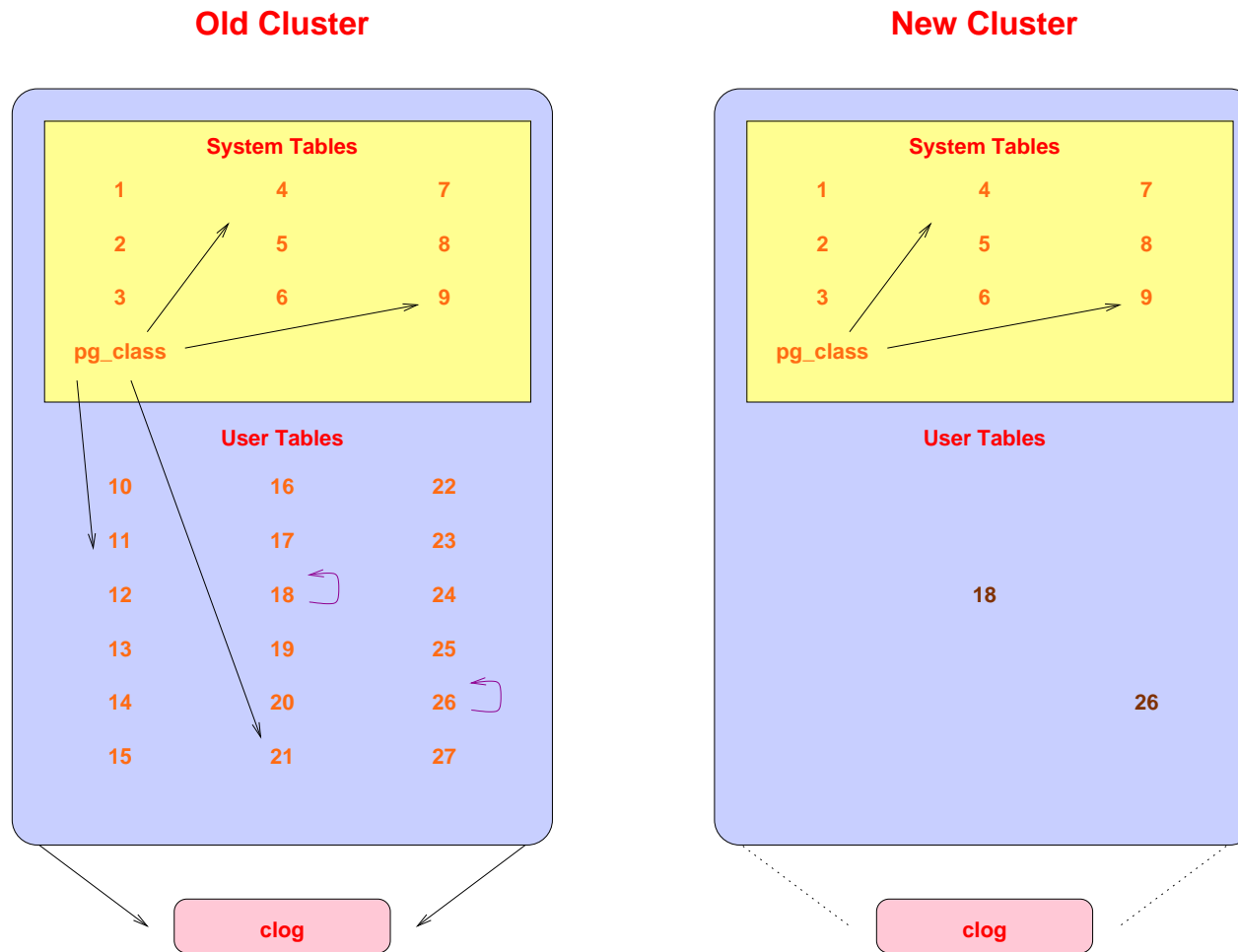
# Transfer Clog and XID



# Get Schema Dump

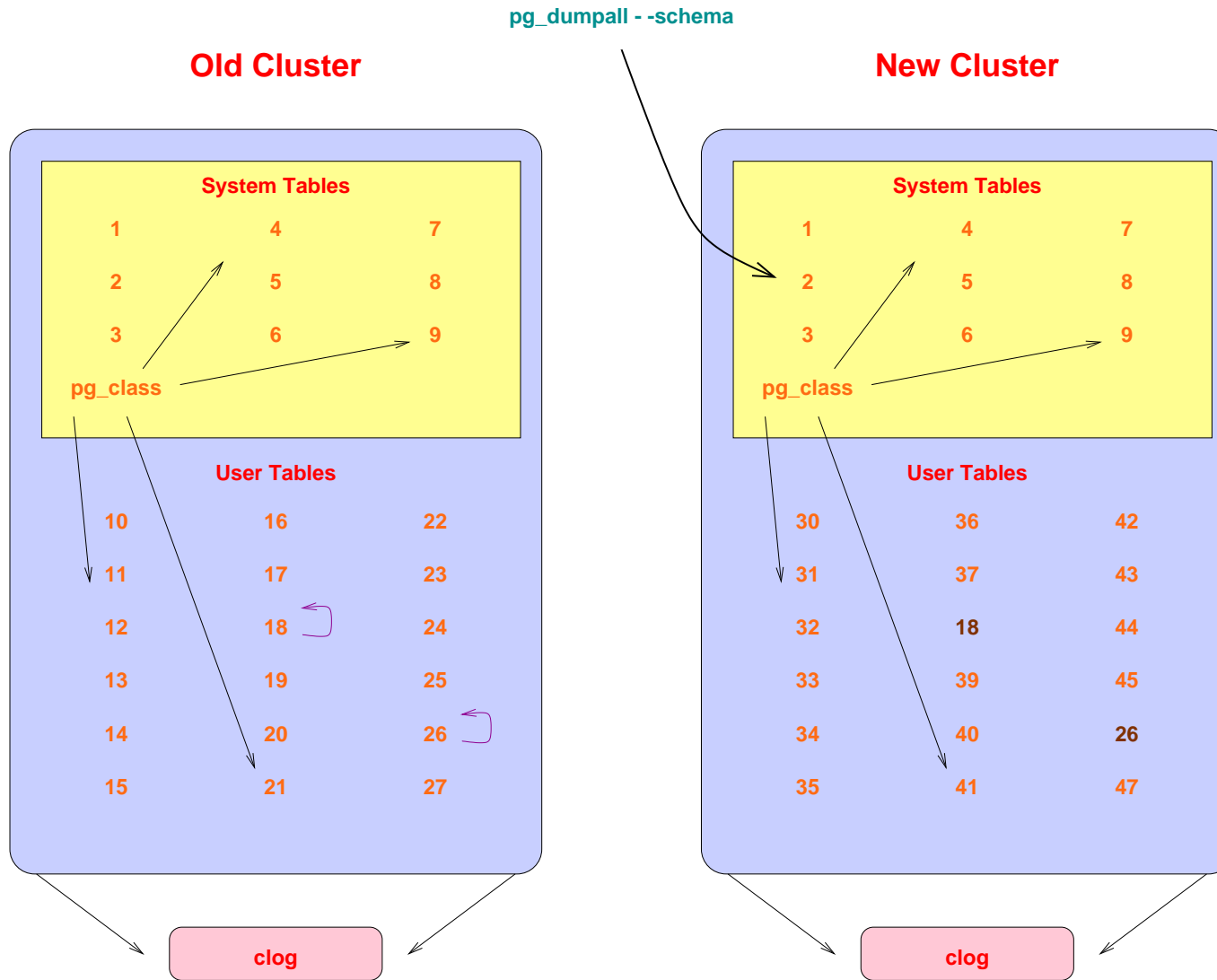


# Reserve TOAST OIDs Using Relfilenodes

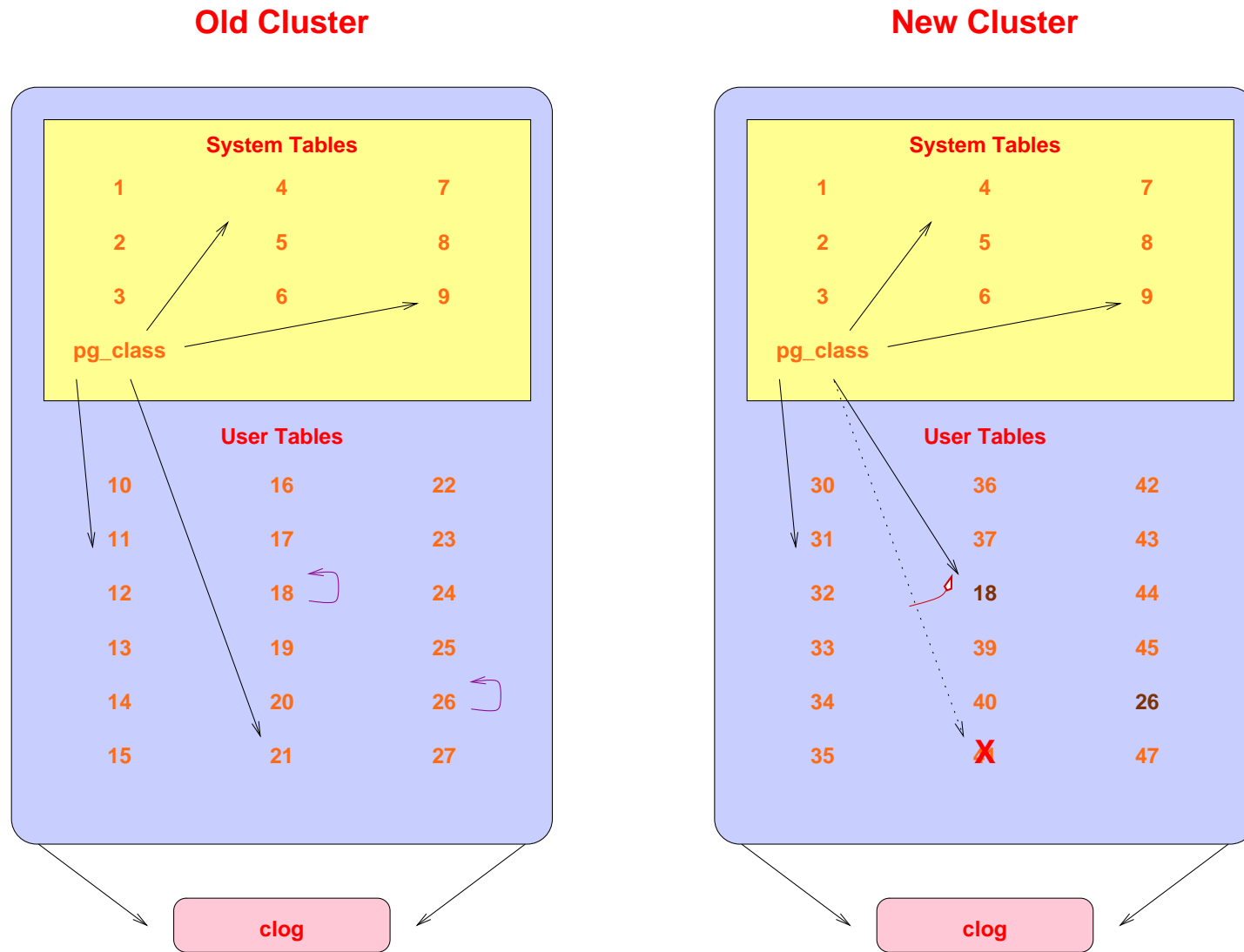


This is necessary because heap references to TOAST tables contain the TOAST oids for easy lookup.

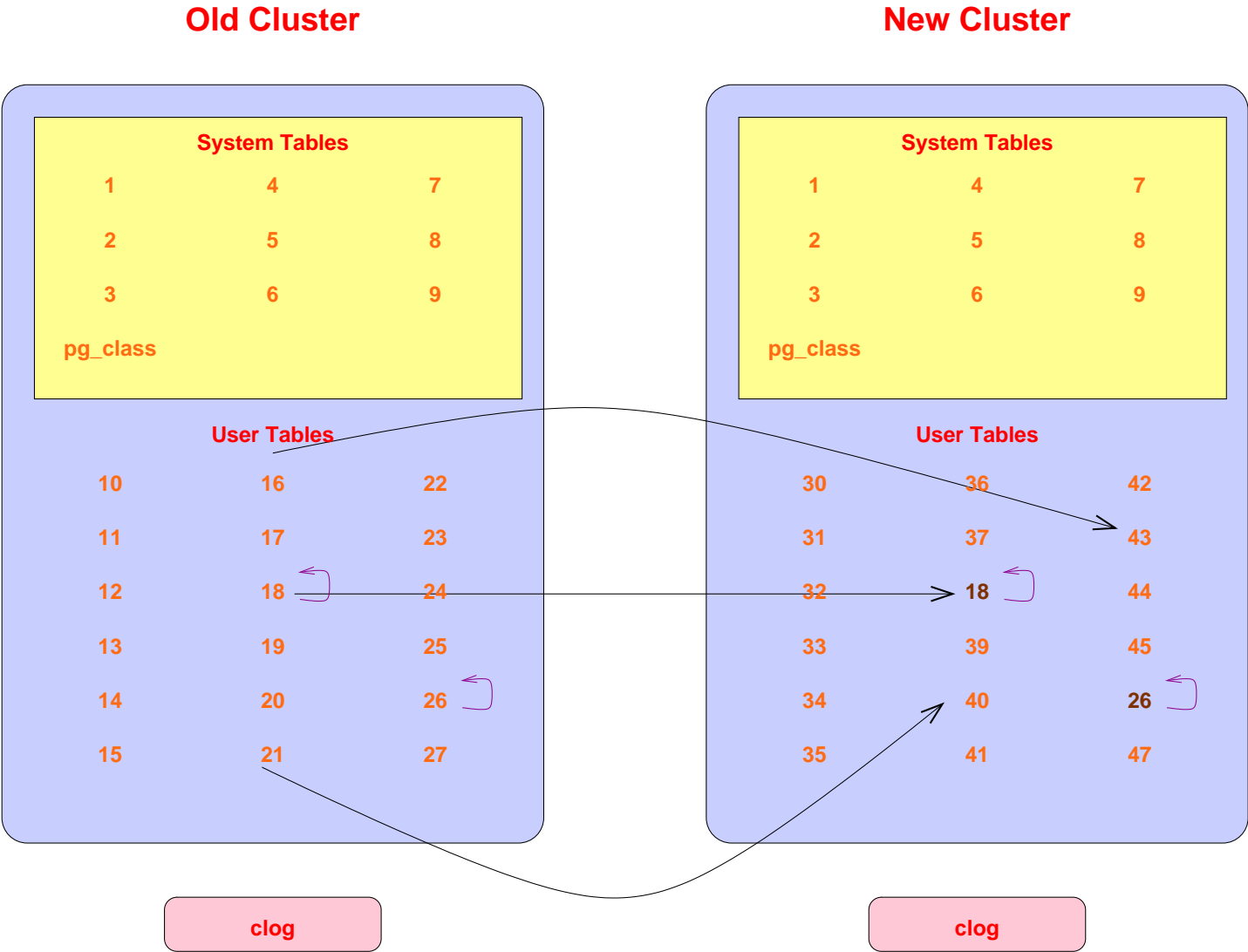
# Restore Schema In New Cluster



# Connect TOAST Placeholders To the Proper Relations

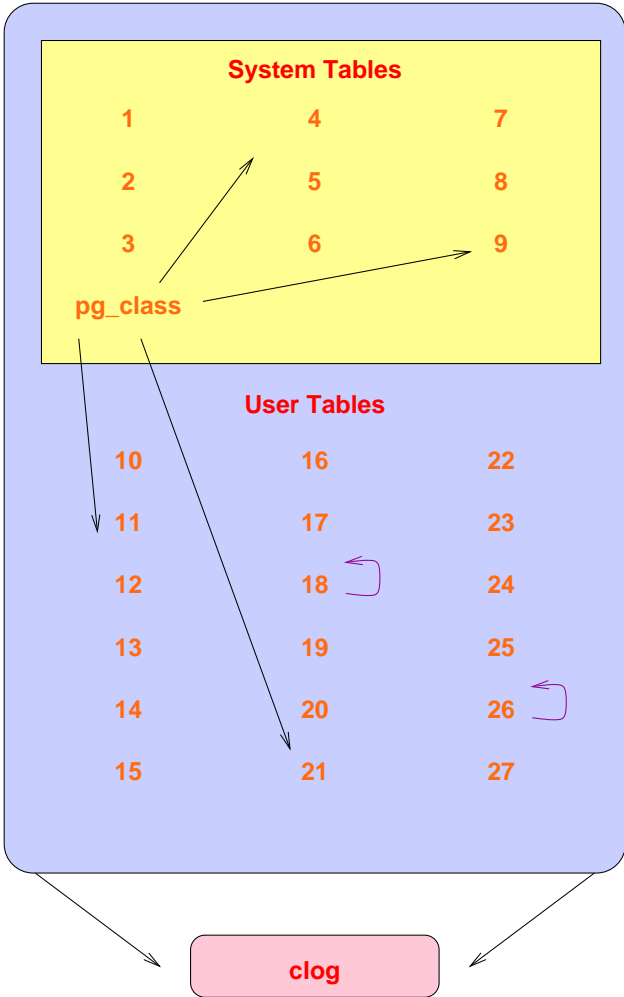


# Copy User Heap/Index Files

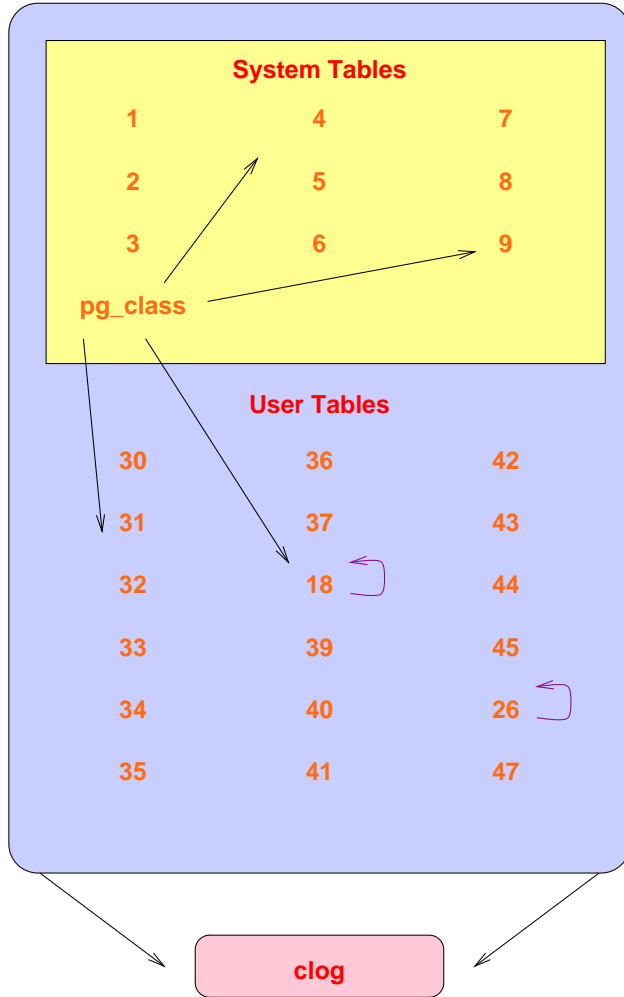


# Complete

Old Cluster



New Cluster



# How It Works: In Detail

- Check for cluster compatability
  - locale
  - encoding
  - integer datetimes (default changed from 8.3 -> 8.4)
- Use `pg_dumpall` to dump old cluster schema (no data)
- Freeze all new cluster rows (remove reference to clog entries)
- Rename tablespaces to `*_old`
- New cluster uses old xid counter value (see freeze above)
  - Set system table frozen xids to match the current xid
- Create new users/databases

- Collect cluster information
- Install support functions that call internal backend functions
- Create placeholder files to reserve relfilenode file names
- Create schema in new cluster
- Adjust new cluster to use reserved relfilenode names
  - Delete placeholder toast relfilenode files
  - Remove new cluster toast tables
  - Create new cluster toast table using reserved relfilenode
  - Assign new toast tables with proper relfilenodes to relations
- Copy or link files from old cluster to new cluster
  - Toast tables have the same relfilenodes as in the old cluster
- Warn about any remaining issues, like REINDEX requirements

# Sample Run: Validation 1

Performing consistency checks

```
-----  
Checking old data directory /u/pgsql.old/data  
  checking base ok  
  checking global ok  
  checking pg_clog ok  
  checking pg_multixact ok  
  checking pg_subtrans ok  
  checking pg_tblspc ok  
  checking pg_twophase ok  
  checking pg_xlog ok  
Checking new data directory /u/pgsql/data  
  checking base ok  
  checking global ok  
  checking pg_clog ok  
  checking pg_multixact ok  
  checking pg_subtrans ok  
  checking pg_tblspc ok  
  checking pg_twophase ok  
  checking pg_xlog ok  
Checking binaries in old cluster (/u/pgsql.old/bin)  
  checking postgres ok  
  checking pg_ctl ok  
  checking pg_dumpall ok  
  checking psql ok
```

## Sample Run: Validation 2

```
Checking binaries in new cluster (/u/pgsql/bin)
  checking postgres                               ok
  checking pg_ctl                                 ok
  checking pg_dumpall                             ok
  checking psql                                   ok
Starting postmaster to service old cluster
  waiting for postmaster to start                 ok
Getting pg_database and pg_largeobject relfilenodes ok
Checking for columns with user-defined composite types ok
Checking for columns with user-defined array types ok
Checking for columns with user-defined enum types ok
Checking for /contrib/isn with bigint-passing mismatch ok
Checking for invalid 'name' user columns          ok
Checking for tsquery user columns                 ok
Creating script to adjust sequences               ok
Creating catalog dump                             ok
Splitting old dump file                           ok
Stopping postmaster servicing old cluster         ok
Starting postmaster to service new cluster
  waiting for postmaster to start                 ok
Checking for presence of required libraries       ok
Stopping postmaster servicing new cluster         ok
*Checks complete*
```



# Sample Run: Migration 1

```
Performing migration
-----
Adding ".old" suffix to old global/pg_control      ok
Renaming tablespaces to *.old                     ok
Deleting old commit clogs                          ok
Copying commit clogs                               ok
Setting next transaction id for new cluster        ok
Resetting WAL archives                             ok
Starting postmaster to service new cluster
  waiting for postmaster to start                  ok
Setting frozenxid counters in new cluster          ok
Creating databases in new cluster                  ok
Stopping postmaster servicing new cluster          ok
Creating placeholder relfiles for toast relations
                                                    ok

Starting postmaster to service new cluster
  waiting for postmaster to start                  ok
Restoring database schema                          ok
Adding support functions to new cluster            ok
Restoring relations to use fixed toast file names
                                                    ok
```

# Sample Run: Migration 2

```
Removing support functions from new cluster      ok
Stopping postmaster servicing new cluster       ok
Restoring user relation files                   ok

Setting next oid for new cluster                 ok
Adjusting sequences                             ok
Checking for tsvector user columns              ok
Checking for hash and gin indexes               ok

Checking for bpchar_pattern_ops indexes         ok
```

```
*Upgrade complete*
| Optimizer statistics and free space information
| are not transferred by pg_migrator, so consider
| running:
|     vacuumdb --all --analyze
| on the newly-upgraded cluster.
```

# Possible Post-8.4 Data Format Changes

Change	Conversion Method
clog	none
heap page header, including bitmask	convert to new page format on read
tuple header, including bitmask	convert to new page format on read
data value format	create old data type in new cluster
index page format	reindex, or recreate index methods
TOAST page format	convert to new page format on read

# Migration Timings

Migration Method	Minutes
dump/restore	300.0
dump with parallel restore	180.0
pg_migrator in copy mode	44.0
pg_migrator in link mode	0.7

Database size: 150GB, 850 tables

The last duration is *44 seconds*.

*Timings courtesy of  
Stefan Kaltenbrunner  
(mastermind on IRC)*

# Conclusion

PG 8.3

PG 8.4

