



Let your PostgreSQL data  
travel back in time

French acronym for  
*"Enregistrement des Mises A Jour"*  
*i.e. "updates recording"*

## *E-Maj, what is it for?*

- E-Maj allows the data content to **travel back in time**, with a table level granularity
- By recording updates on sets of application tables, it is possible to
  - **Count** them (statistic function),
  - Easily **view** them (audit function),
  - **Revert** them ("rollback" function),
  - **Replay** them (script generation, or revert a revert...)
- Usable with
  - applications in test or in production
  - databases of all sizes

## *The gains*

- In **test** environment
  - Helps the application tests management by providing a quick way to
    - Examine updates generated by the application, for debugging purpose
    - Cancel updates generated by the application in order to easily repeat tests
- In **production** environment
  - Allows to cancel processings
    - Without being obliged to save and restore the instance by `pg_dump/pg_restore` or by physical copy
    - With a finer granularity
  - Avoids to loose entire batch processing nights by helping the recovery after failure
  - Very interesting with large tables and few updates

## The components

- **E-Maj**, the heart
  - A PostgreSQL extension
  - Open Source, under GPL licence
  - Download from [pgxn.org](https://pgxn.org) - <https://pgxn.org/dist/e-maj/>
  - Sources available on [github.com](https://github.com) - <https://github.com/dalibo/emaj>
- **Emaj\_web**
  - A web client - [https://github.com/dalibo/emaj\\_web](https://github.com/dalibo/emaj_web)
- The online **documentation**
  - In English (or French) - <https://emaj.readthedocs.io/en/latest/>



## *The characteristics which drove the design*

- **Reliability**
  - Absolute data integrity after updates cancellation
  - Management of all usual objects (tables, sequences, constraints,...)
- **Ease of use** for DBAs, production people, application developers and testers,...
  - Easy to understand and use
  - Easy to integrate into an automatized production (thus scriptable)
- **Performance**
  - Limited log overhead
  - Acceptable “rollback” duration
- **Security**
- **Maintainability**

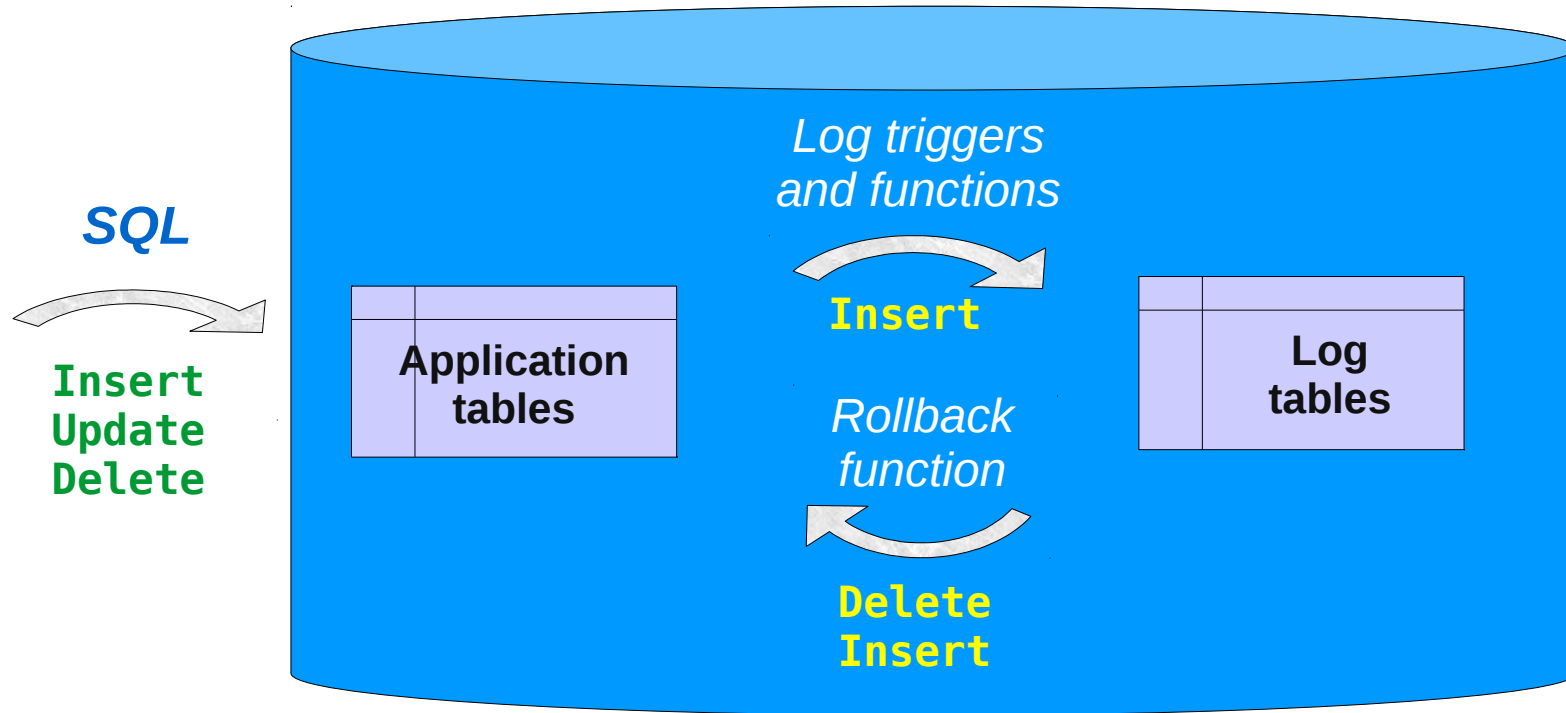
## Concepts

- **Tables Group** = a set of tables and/or sequences belonging to one or several schemas and having the same life cycle ; it's the only object manipulated by users
- **Mark** = stable point in the life of a tables group, whose state can be set back ; identified by a name
- **E-Maj Rollback** = positioning of a tables group at a previously set mark state
  - NB: this concept is different from the transaction rollbacks performed by the RDBMS
    - a “RDBMS-rollback” cancels the current transaction
    - a “E-Maj rollback” cancels updates from several committed transactions

## Concepts (addon)

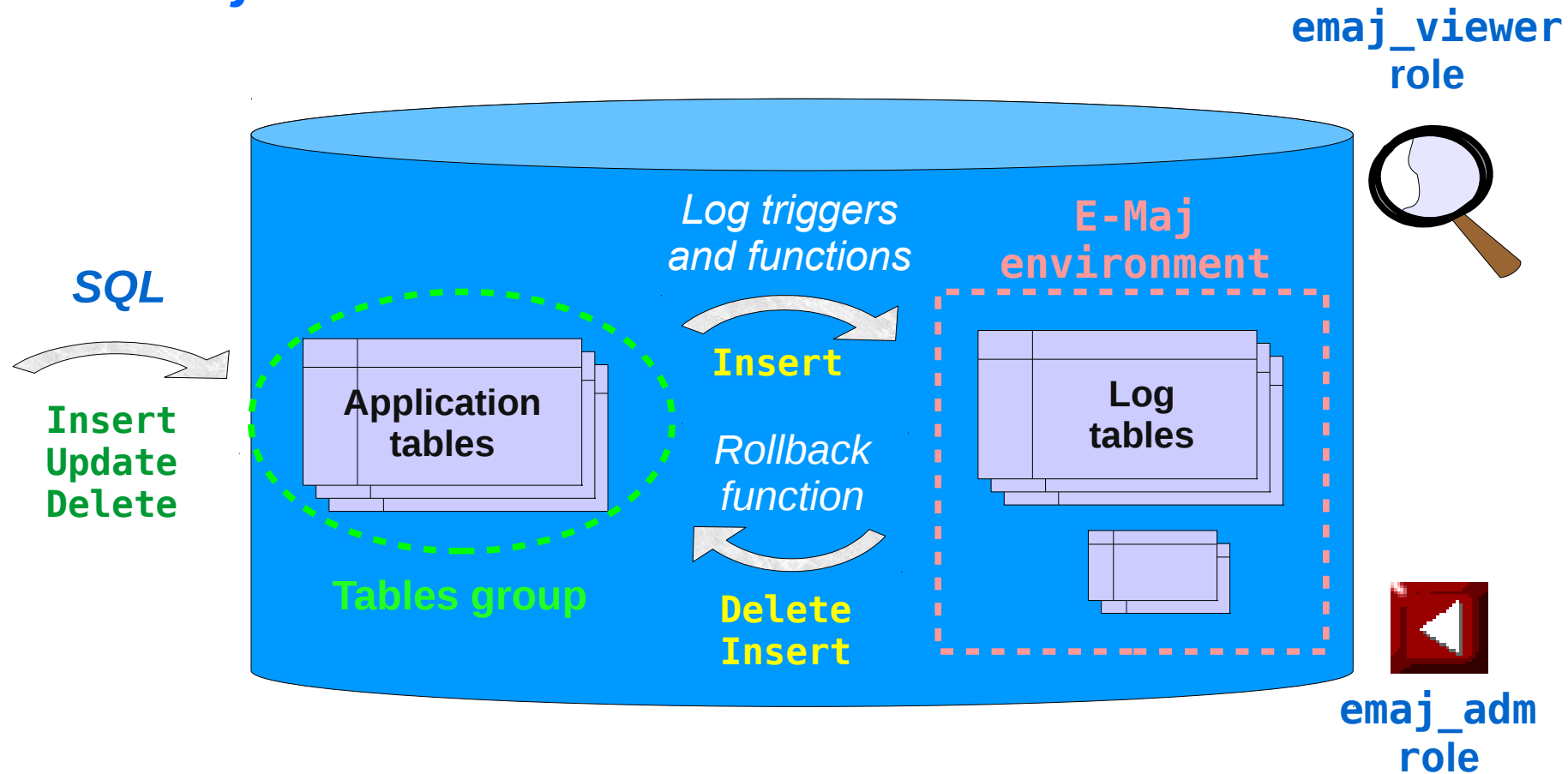
- By default, a tables group is created as “**rollbackable**”
- A tables group may be created as “**audit-only**”
  - E-Maj rollbacks are not possible
  - But
    - TRUNCATE are recorded and not blocked
    - A table may have no declared PRIMARY KEY
    - A table may have been created as UNLOGGED or WITH OIDS

# An updates recording based on triggers





# Main objects



## *Management of application sequences*

- Sequence increments are not individually recorded
- At set mark time
  - The state of each sequence of the group is stored into an internal table
- At E-Maj rollback time
  - Each sequence is reset to its state recorded at the targeted mark

## Install E-Maj

- Download and unzip the extension
- Standart install
  - Copy emaj.control and sql/\*.sql files into \$SHAREDIR/extension
  - Log on the target database as super-user and execute
    - `CREATE EXTENSION emaj CASCADE;`
- Install on DBaaS cloud environment
  - `psql ... -f sql/emaj-<version>.sql`
- This adds to the database
  - the extensions `dblink` et `btree_gist` if needed
  - 1 schema, named 'emaj', with about 110 functions, 15 technical tables, 8 types, 1 view, 1 sequence, 2 event triggers
  - 2 roles

## Initialization

- Populate the `emaj_group_def` table to define the tables groups content
  - 1 row per application table/sequence
  - At least `grpdef_group`, `grpdef_schema` and `grpdef_tblseq` columns
- For each group:
  - `SELECT emaj_create_group (group, is_rollbackable);`
  - Creates for each application table:
    - 1 log table + 1 sequence
    - 1 trigger + 1 log function
  - NB: `SELECT emaj_drop_group (group)`
    - ... drops an existing group

## The 3 main functions to manage groups

- “Starting” a group
  - `emaj_start_group (group, mark)`  
activates the log triggers and sets a first mark
- Setting a mark
  - `emaj_set_mark_group (group, mark)`  
sets an intermediate mark
- “Stopping” a group
  - `emaj_stop_group (group [,mark])`  
deactivates the log triggers => a rollback is not possible anymore
- The % character in a mark name represents the current date and time

## Examine logs

- Examining log tables may largely help the application debugging
- Each application table has its own log table
  - `emaj_<schema>.<table>_log`
- A log table contains
  - The same columns as its related application table
  - And some technical columns
- A single row change in an application table generates
  - 1 log row for an INSERT (image of the new row)
  - 1 log row for a DELETE (image of the old row)
  - 2 log rows for an UPDATE (image of the old and the new rows)
- A TRUNCATE generates a single log row

## *Log tables technical columns*

- 6 technical columns at the end of each log row
  - `emaj_verb` : SQL statement type - INS/UPD/DEL/TRU
  - `emaj_tuple` : row type - OLD/NEW
  - `emaj_gid` : internal sequence number
  - `emaj_changed` : time of the update - `clock_timestamp()`
  - `emaj_txid` : transaction identifier - `txid_current()`
  - `emaj_user` : connection role of the client - `session_user`
- ... and some others can be added
- It is possible to identify clients and transactions, and analyze the timing of the program execution

## Counting updates

- 2 statistical functions
  - `emaj_log_stat_group (group, start_mark, end_mark)`  
quickly returns an estimate of recorded updates
    - by table
    - between 2 marks (or between 1 mark and the current state)
  - `emaj_detailed_log_stat_group (group, start_mark, end_mark)`  
scans log tables and returns precise statistics on their content
    - by table
    - by statement type (INSERT / UPDATE / DELETE)
    - by ROLE
    - between 2 marks (or between 1 mark and the current state)

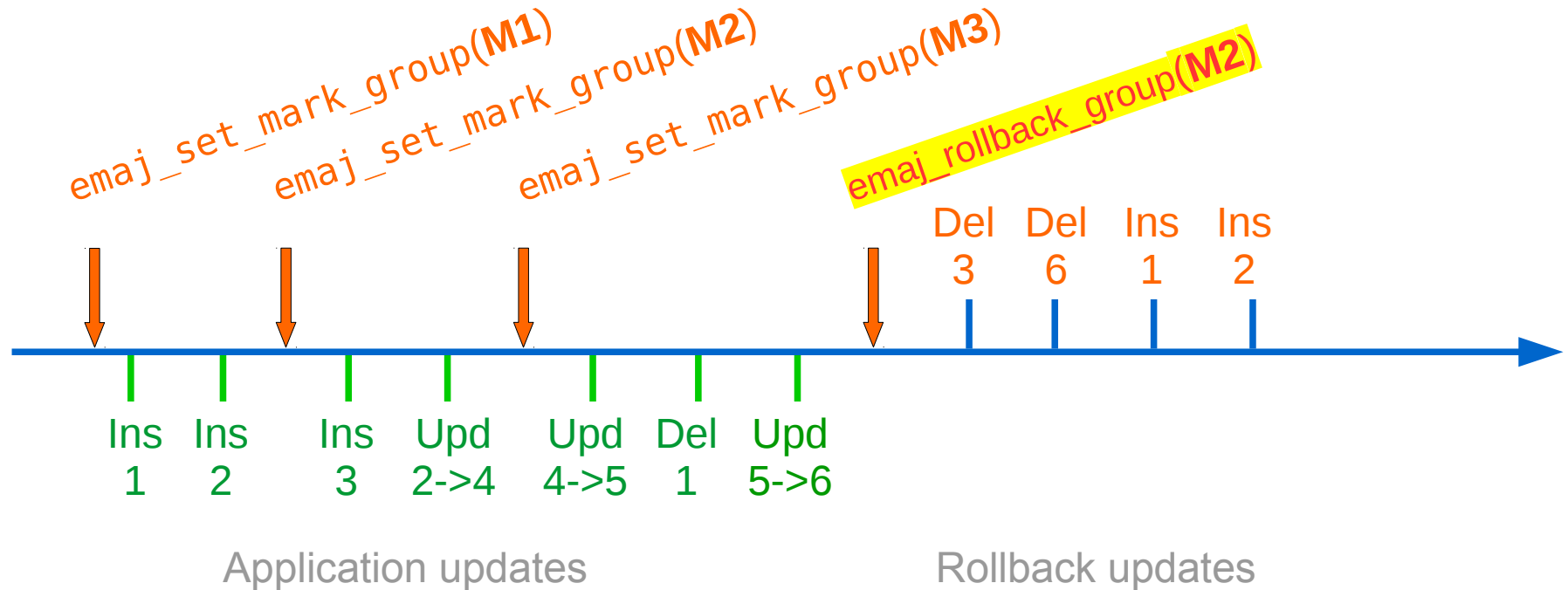


## Cancel updates : the “simple” rollback

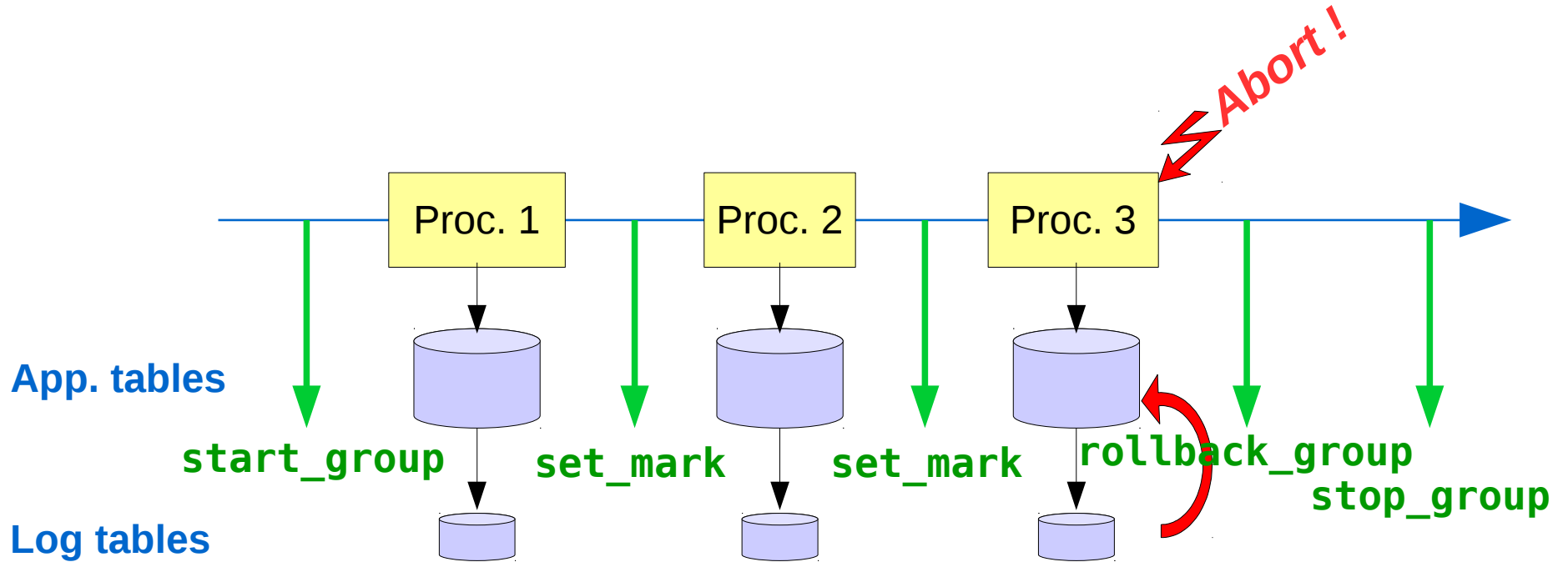
- A “rollback” function allows to reset a tables group in the state it had at a given mark
  - `emaj_rollback_group (group, mark, false)`
- How this works
  - Log triggers are deactivated during the operation
  - Each table is reset to its mark state using an optimised algorithm
  - Application sequences are reset to their mark state
  - Takes into account the foreign keys, if any
  - The canceled logs and marks are deleted
    - => all what is after the rollback mark is forgotten

## An optimised rollback algorithm

- It processes each primary key value only once



# A typical E-Maj usage (production batch processing)

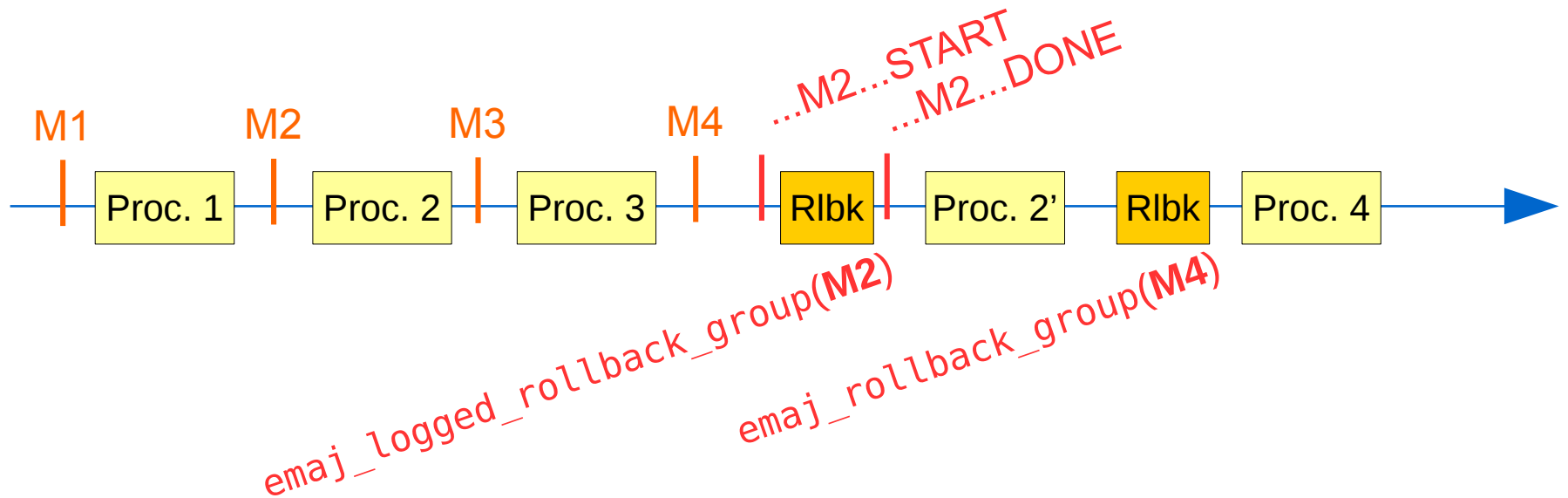


## *Cancelling updates : the “logged” rollback*

- Different from the “simple” rollback
  - Log triggers are NOT deactivated during the operation  
=> the updates generated by the rollback are recorded
  - Cancelled logs et marks are NOT deleted
- So we can revert an E-Maj rollback ! And more generally let a tables group travel back and forth in time !
- 2 marks are automatically set before and after the rollback
  - `RLBK_<marque cible>_<HH.MI.SS.MS>_START`
  - `RLBK_<marque cible>_<HH.MI.SS.MS>_DONE`
- During the rollback, tables remain accessible in read mode

## A typical E-Maj usage in test environment

- 4 processings to test in sequence
- After test 3, a new version of processing 2 must be re-tested
- Then perform the remaining tests



## *Estimating an E-Maj rollback duration*

- In order to know if we have enough time to perform the operation or if another way to recover would be more efficient
- A function estimates the time needed to rollback a group to a given mark
  - `emaj_estimate_rollback_group (group, mark)`

## *Executing a parallel E-Maj rollback*

- A php or perl client performs rollbacks with parallelism
  - `emajParallelRollback.php -d <database> -h <host> -p <port> -U <user> -W <password> -g <group_name or groups_list> -m <mark> -s <nb_sessions> [-l]`
- Automatically spreads the tables to process into a given number of parallel sessions
- All sessions belong to a single transaction (2PC)
  - => `max_prepared_transaction` >= nb sessions
- Needs php or perl with its PostgreSQL extension

## Monitoring E-Maj rollbacks in execution

- A function
  - `SELECT * FROM emaj.emaj_rollback_activity ();`
  - returns
    - The characteristics of rollbacks (group, mark...)
    - Their state
    - Their current duration
    - An estimate of the remaining duration and the already executed %
- Needs to setup the value of the “`dblink_user_password`” parameter in the `emaj_param` table



## Monitoring E-Maj rollbacks

- A php or perl client to monitor the executing or completed rollbacks
  - `emajRollbackMonitor.php -d <database> -h <host> -p <port> -U <user> -W <password> -n <nb_iterations> -i <refresh_rate_in_seconds> -l <nb_completed_rollbacks> -a <completed_rollbacks_history_depth_in_hours>`

```
E-Maj (version 2.2.0) - Monitoring rollbacks activity
```

```
-----  
04/09/2017 - 12:07:17
```

```
** rollback 35 started at 2017-09-04 12:06:21.474217+02 for groups {myGroup1}  
status: COMMITTED ; ended at 2017-09-04 12:06:21.787615+02
```

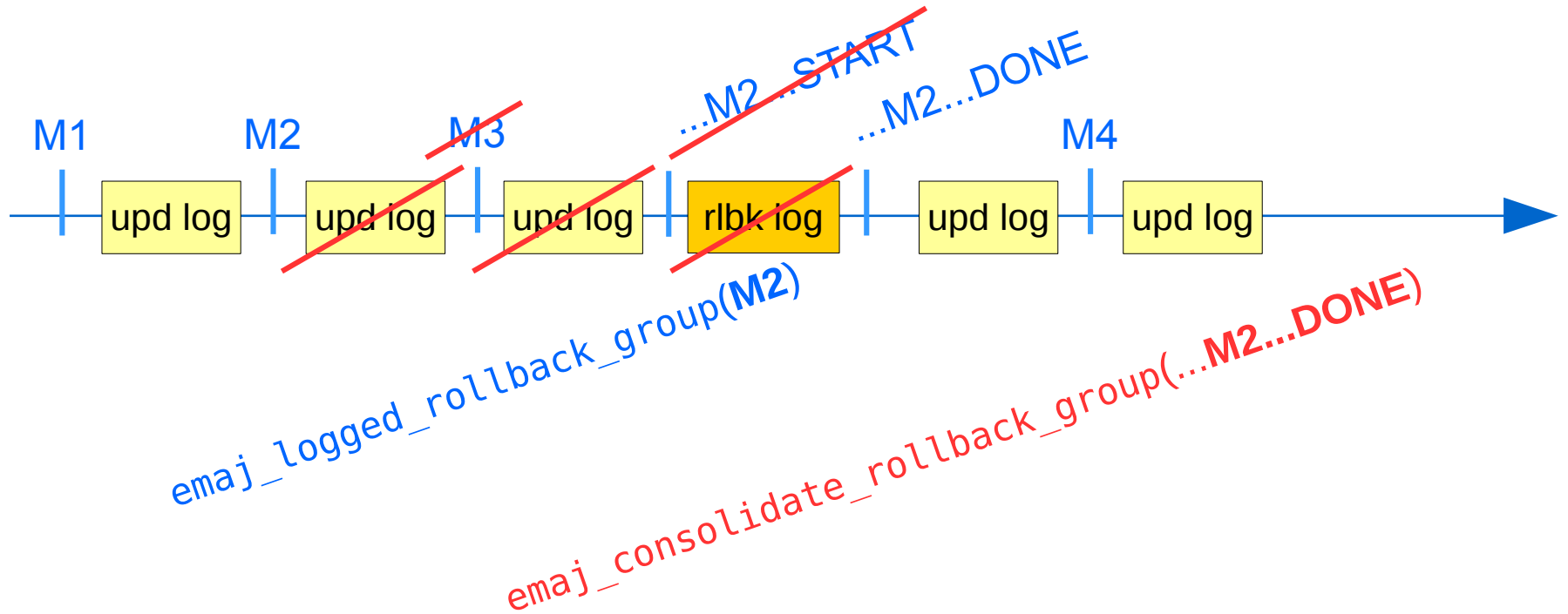
```
-> rollback 36 started at 2017-09-04 12:04:31.769992+02 for groups {group1232}  
status: EXECUTING ; completion 89 % ; 00:00:20 remaining
```

```
-> rollback 37 started at 2017-09-04 12:04:21.894546+02 for groups {group1233}  
status: LOCKING ; completion 0 % ; 00:22:20 remaining
```

## Consolidate a “logged” rollback

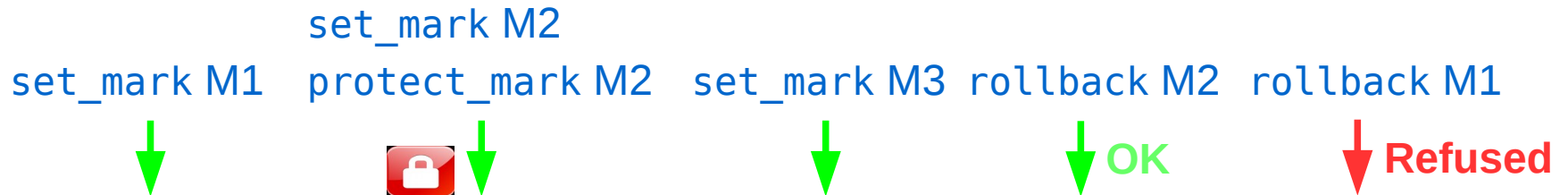
- “Consolidate” a rollback means transform a “logged rollback” into a “simple rollback”
- Intermediate logs and marks are deleted, recovering some place in the logs
  - `emaj_consolidate_rollback_group (groups, end_rollback_mark)`
- Tables can be updated during the consolidation
- A function returns the list of consolidable rollbacks
  - `emaj_get_consolidable_rollbacks ()`

## Example of E-Maj rollback consolidation



## Being protected against unattended E-Maj rollbacks

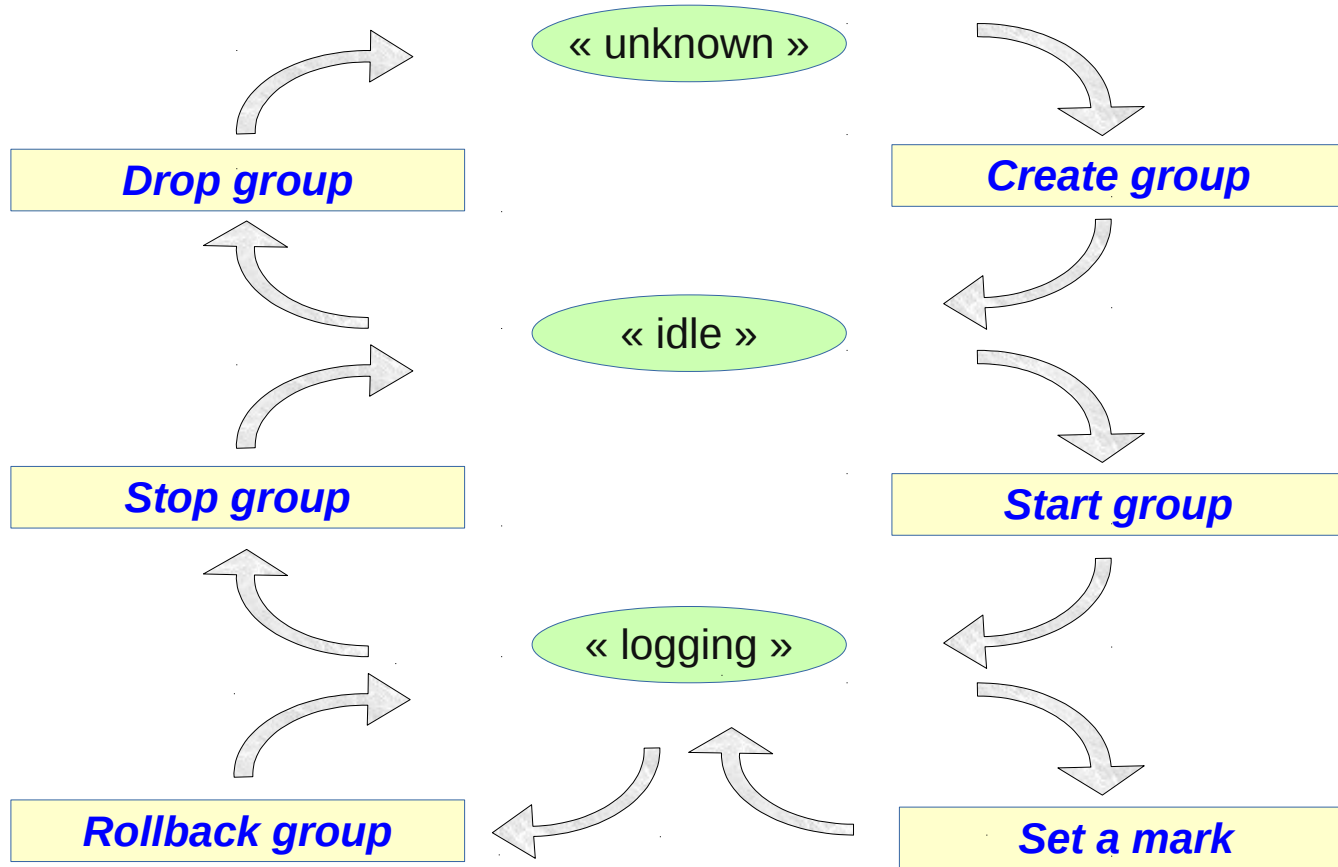
- 2 functions to manage the protection of a tables group
  - `emaj_protect_group (group)`
  - `emaj_unprotect_group (group)`
- 2 functions to manage the protection of a mark
  - `emaj_protect_mark_group (group, mark)` blocks any attempt to rollback to a mark prior the protected mark
  - `emaj_unprotect_mark_group (group, mark)`



## Exporting from an E-Maj environment

- Generate a sql script replaying the recorded updates between 2 marks, for some or all tables and sequences of a group
  - *emaj\_gen\_sql\_group (group, start\_mark, end\_mark, dest\_file [,tables/seq\_list])*
- Snap on files in a given directory, by COPY, all tables and sequences of a group
  - *emaj\_snap\_group (group, directory, copy\_options)*
- Snap on files in a given directory, by COPY, a part of log tables and sequences of a group
  - *emaj\_snap\_log\_group (group, start\_mark, end\_mark, directory, copy\_options)*
- Useful in test to compare several executions of a processing or to “replicate” the updates produced by a processing

# The tables group life cycle



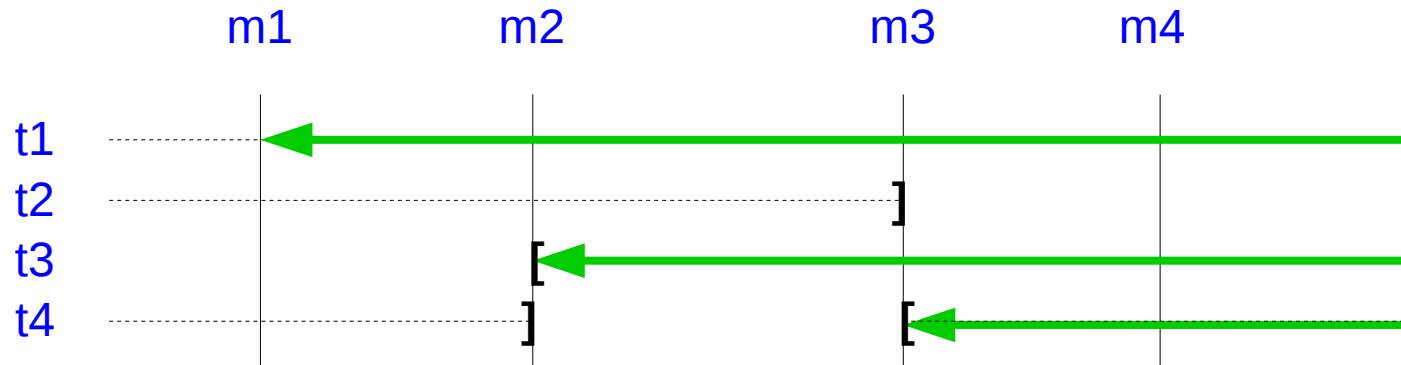
## Modifying the groups structure

- 2 steps
  - Modify the content of the `emaj_group_def` table (insert/delete rows, change attributes)
  - Call the function `emaj_alter_group (group)`
- No restriction for tables group in IDLE state
- The tables group may remain in LOGGING state, to
  - Modify attributes in `emaj_group_def`
  - Remove a table or a sequence from its tables group
  - Add a table or a sequence to a tables group
  - Change the group ownership for a table or a sequence

## Impact of logging group structure changes on rollbacks

Table t2 removed at mark m3, t3 added at m2, t4 removed at m2 and added at m3

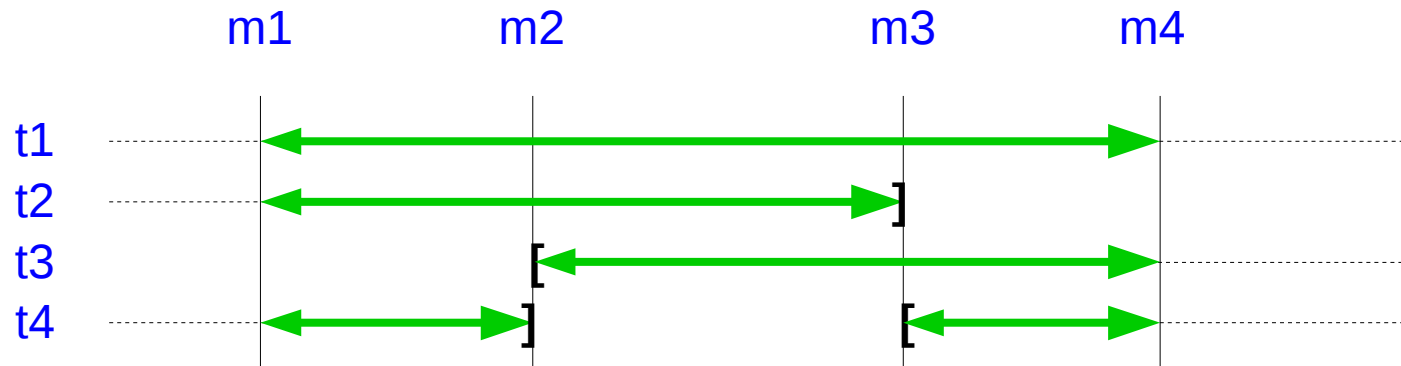
`emaj_rollback_group(<groupe>,'m1')` would process:





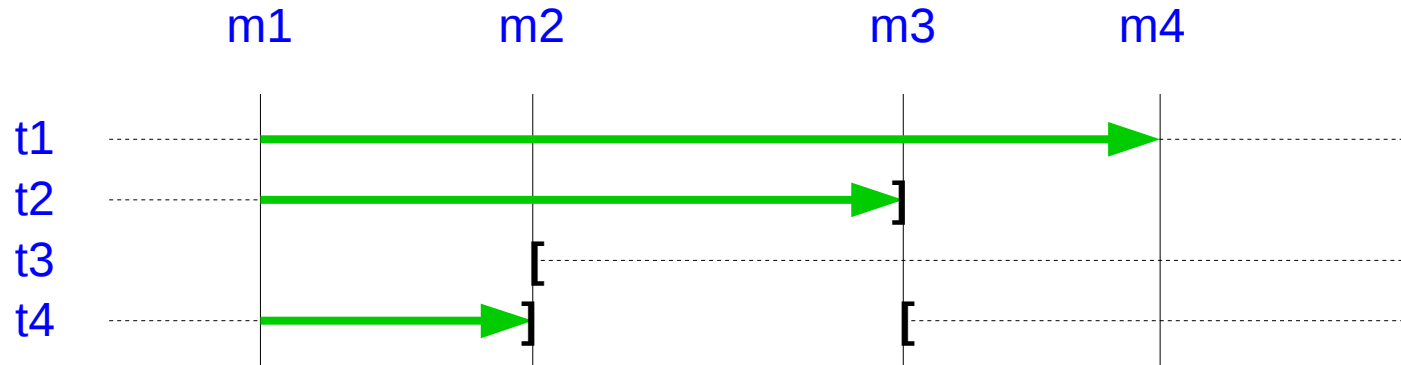
## Impact of logging group structure changes on statistics

`emaj_log_stat_group(<groupe>,'m1','m4')` would report:



## Impact of logging group structure changes on the SQL scripts generation

`emaj_gen_sql_group(<group>,'m1','m4')` would process:



## *Modify the structure of a table in a LOGGING group*

- For actions like: rename the table, change its schema, add/drop/rename a column, change a column type
- The log table structure is impacted
- 3 steps
  - Remove the table from its tables group
  - ALTER TABLE
  - Add the table into its tables group
- Constraint: an E-Maj rollback to a prior mark will not be able to go beyond the structure change
- Idem to rename a sequence of change its schema

## Processing several groups in a single operation

- Some “multi-groups” variants of functions
  - `emaj_start_groups (groups_array, ... )`
  - `emaj_stop_groups (groups_array, ... )`
  - `emaj_set_mark_groups (groups_array, ... )`
  - `emaj_rollback_groups (groups_array, ... )`
  - `emaj_logged_rollback_groups (groups_array, ... )`
  - `emaj_log_stat_groups (groups_array, ... )`
  - `emaj_gen_sql_groups (groups_array, ... )`
  - `emaj_alter_groups (groups_array, ... )`
- Allows to get marks shared by several groups
- Both PostgreSQL syntaxes for groups arrays
  - `ARRAY['group 1', 'group 2', ... ]`
  - `'{"group 1", "group 2", ... }'`

## Managing marks

- Comment a mark for a group (add/modify/suppress)
  - `emaj_comment_mark_group (group, mark, comment)`
- Rename a mark
  - `emaj_rename_mark_group (group, old_name, new_name)`
- Delete a mark
  - `emaj_delete_mark_group (group, mark)`
  - If the deleted mark is the first one, logs prior the second one are deleted
- Delete all marks prior a given mark
  - `emaj_delete_before_mark_group (group, mark)`
  - Deletes logs prior the mark (it may take a long time...)

## Managing mark (2)

- Search for marks
  - `emaj_find_previous_mark_group (group, date-time)` returns the mark immediately preceding a given date and time
  - `emaj_find_previous_mark_group (group, mark)` returns the mark immediately preceding a given mark
- “`EMAJ_LAST_MARK`” represents the last set mark for a group
  - Usable for all parameters defining an existing mark

## Other actions on groups

- Comment a group (add/modify/suppress)
  - `emaj_comment_group (group, comment)`
- Purge log tables of a stopped group (anticipating its next restart)
  - `emaj_reset_group (group)`
- Force a group stop (in case of problem with the normal stop function)
  - `emaj_force_stop_group (group)`

## *Other actions*

- Verify the good health of the E-Maj installation
  - `emaj_verify_all ()`
- Get the current log table of a given application table
  - `emaj_get_current_log_table ()`



## Temporary or permanent logging?

- **Temporary logging** = steps like
    - `emaj_start_group()`
    - repeat
      - processing
      - `emaj_set_mark()`
    - `emaj_stop_group()`
  - At next start, old logs are purged
  - But stops and starts set very heavy locks
- **Permanent logging** = no repeated group stop/restart
    - Obsolete data in log tables must be regularly deleted, using the `emaj_delete_before_mark()` function
  - The deletion can be costly if the volume of log to delete is big

## *For large databases...*

- Log tables and indexes can be stored into **tablespaces**
  - Can be configured for each table in `emaj_group_def`

## *To ensure the reliability*

- No change in the PostgreSQL engine
- Many systematic **checks**, in particular at group start, mark set or rollback times:
  - Do all required tables, sequences, functions and triggers exist?
  - Consistency of columns between the application tables and the related log tables (existence, type)?
- Heavy **locks** on tables at `start_group`, `set_mark_group` and `rollback_group`, to be sure that no transaction is currently updating application tables
  - The order of lock setting can be influence by a priority level defined for each table into the `emaj_group_def` table
- Rollback all tables and sequences by a single **transaction**

## *To ensure the reliability (2)*

- **TRUNCATE** statements are blocked for active “rollbackable” groups
- “**event triggers**” block unintentional drops or some component changes (tables, sequences, functions...)
  - 2 functions to deactivate/reactivate the lock-in
  - `emaj_disable_protection_by_event_triggers ()`
  - `emaj_enable_protection_by_event_triggers ()`

## *Impact of application triggers on E-Maj rollbacks*

- Triggers of type BEFORE on a table belonging to a tables group
  - Values really inserted into the database are recorded into the log
  - The trigger needs to be disabled at E-Maj rollback
- Triggers of type AFTER updating a table belonging to the same tables group
  - The trigger must be disabled at E-Maj rollback
  - The rollback will reset both tables with the right content
- Triggers of type AFTER updating a table not belonging to the same tables group, or more complex triggers
  - Study the impacts ! Just disabling the trigger may not be sufficient
- By default, application triggers are automatically disabled by E-Maj rollbacks
- A trigger may be left in its state at rollback time if it is registered with the `emaj_ignore_app_trigger ()` function

## *To contribute to the security*

- 2 NOLOGIN roles whose rights may be granted:
  - `emaj_adm` for the E-Maj administration
  - `emaj_viewer` to just look at E-Maj objects (logs, marks, statistics)
- E-Maj objects are only created and handled by a super-user or a member of the `emaj_adm` role
- No other right has to be granted on E-Maj schemas, tables and functions
- Log triggers are created with the “SECURITY DEFINER” attribute
- No need to give additional rights to application tables or sequences

## Performances

- Log overhead
  - Highly depends on hardware and on the application read/write SQL ratio
  - Typically a few % on elapse times
  - But can be much higher on pure data loading
- Rollback duration
  - Of course depends on the number of updates to cancel
  - Also highly depends on
    - The hardware configuration
    - Tables structure (row sizes, indexes, foreign keys, other constraints...)
  - But almost always shorter than a logical restore

## Emaj\_web

- For administrators and users
- Shows all E-Maj objects (groups, marks...) and their attributes
- Allows (almost) all possible actions on E-Maj objects

Connection: localhost:5496 - role "postgres" SQL | History | Logout English

Emaj\_web > Pg 9.6 > postgres

Groups    Groups configuration    Rollback operations    Schemas    E-Maj

### Tables groups in "LOGGING" state i

	Group	Creation date/time	# tables	# sequences	Type	# marks	Actions				Comment	
<input type="checkbox"/>	myGroup1	21/02/2019 18:45:01	5	1		3	Set a mark	Protect	Rollback	Stop	Set a comment	Useless comment!
<input type="checkbox"/>	myGroup2	21/02/2019 18:45:01	4	2		4	Set a mark	Protect	Rollback	Stop	Set a comment	

Select    Actions on selected objects (0)

All / Visible / None / Invert   

### Tables groups in "IDLE" state i

	Group	Creation date/time	# tables	# sequences	Type	# marks	Actions				Comment	
<input type="checkbox"/>	phil's group#3	21/02/2019 18:45:02	2	1		0	Start	Reset	Set a comment		Drop	

Select    Actions on selected objects (0)

All / Visible / None / Invert   

### Configured but not yet created tables groups i

No tables group is currently configured but not created.

To create a (another) tables group, go first to the groups configuration tab.  
Alternatively, you can create an empty group, then add tables and sequences into it, and apply the configuration change.

Tables groups list



# Emaj\_web : tables group details

Connection: localhost:5496 - role "postgres" SQL | History | Logout English

Emaj\_web > Pg 9.6 > postgres > myGroup1

Properties Log statistics Content

### Tables group "myGroup1" properties

State	Creation date/time	Type	# tables	# sequences	# marks	Log size
	2019-02-21 18:45:01.885922+01		5	1	3	144 kB

Comment: *Useless comment!*

[Set a mark](#) [Protect](#) [Stop](#) [Set a comment](#)

### Tables group "myGroup1" marks

	Mark	State	Date/Time	# row changes	Cumulative changes	Actions						Comment	
<input type="checkbox"/>	MARK3		2019-02-21 18:45:02.211977+01	0	0	Rollback	Rename	Delete	First mark	Protect		Set a comment	
<input type="checkbox"/>	MARK2		2019-02-21 18:45:02.181251+01	7	7	Rollback	Rename	Delete	First mark	Protect		Set a comment	End of 1st program
<input type="checkbox"/>	MARK1		2019-02-21 18:45:02.102865+01	19	26	Rollback	Rename	Delete		Protect		Set a comment	

Select Actions on selected objects (0)

[All](#) / [Visible](#) / [None](#) / [Invert](#) [Delete](#)

## *Current limitations*

- Since E-Maj 3.0, the minimum required PostgreSQL version is **9.5**
- Every application table belonging to a rollbackable group needs a **PRIMARY KEY**
- Table **TRUNCATE** statements cannot be canceled
- **DDL** statement cannot be managed by E-Maj

## *To conclude...*

- Many more **informations** in the documentation and in the README et CHANGES files
- Many thanks for their help to :
  - Andreas Scherbaum, Jean-Paul Argudo and the Dalibo team, CNAF DBA, Don Levine (for the english translation)
  - People who already contacted me for comments, requests...
- Feel free to give any **feedback** through github or email (phb.emaj@free.fr)