Oracle GoldenGate Parallel Replication Internals

Adam Leszczyński

POUG 2018, Sopot
07.09.2018

Questions for today

- How does OGG makes sure that:
  - There are no duplicate transactions?
  - None of the transactions are missed?
  - … event in case of hardware failure?
  - … how is this data managed when the replication uses parallel processing?
  - Is this technical data available to the user?
- What is the order of the transactions that are replicated using Parallel Replication?
  - Can this order be changed? What drives it?
  - What about transaction dependencies?
  - Is the order of COMMIT operations always preserved in the target database?
  - How different Replicat options affect the order of replicated transactions?
- What does mean: eager apply, serialized, dependent, dependent eager, commit serialization and how do all apply options for Replicat influence the behavior or parallel replication?

Disclaimer

- Don’t believe a single word you see here – Check everything by yourself!
- Always first read the documentation and MOS docs
- Results presented in this presentation are based on own research done using the following software versions:
  - Database: 12.1 + PSU 12.1.0.2.180717, 12.2 + RU 12.2.0.1.180717
  - OGG: 12.2.0.2.2, 12.3.0.1.4
- Important note: This presentation contains some simplifications for educational purposes
- **Do not make any business decisions based on this presentation!**

 © 2018 by Adam Leszczyński, all rights reserved

A few words about me

- Independent consultant
- Oracle GoldenGate 10, 11g, 12c Certified Implementation Specialist
- Oracle Database 12c OCA/OCP, SQL Expert
- First programming environment: GW-BASIC (MS DOS 3.3)
- Started with Oracle 8.0.3 and PL/SQL
- Worked for many years with Sybase/SAP ASE, Replication Server
- Internet:
  - bersler.com
  - github.com/bersler
  - twitter.com/bersler
  - stackoverflow.com/users/8217702

Quick introduction to OGG

Source Database (redo log)	Extract Process (memory)	Trail file	Target Database (applied using a single apply session)
T1 insert	T1 insert	T1 insert	T1 insert
T2 update	T1 commit	T1 commit	T1 commit
T3 delete	T2 update	T4 delete	T4 delete
T1 commit	T2 update	T4 insert	T4 insert
T2 update	T2 update	T4 commit	T4 commit
T2 update	T2 insert	T2 update	T2 update
T4 delete	T2 commit	T2 update	T2 update
T3 rollback	T3 delete	T2 update	T2 update
T4 insert	T3 rollback	T2 insert	T2 insert
T4 commit	T4 delete	T2 commit	T2 commit
T5 insert	T4 insert		
T5 rollback	T4 commit		
T2 insert	T5 insert		
T2 commit	T5 rollback		

Crucial points

- Transactions that are committed are written to the trail file
  - **The order of commits in the redo log determines the order of transactions**
  - Transactions start time, time span, interleaving is irrelevant for purpose of replication
  - Transactions that are rolled back are ignored
  - Changes to tables that are not to be replicated are ignored
- The transaction it is not written to the trail until it is committed
  - The trail does not contain transactions that might be rolled back
  - After successful commit (redo is written to disk) the transaction may be processed further
- The result of the Extract process is actually fully deterministic
  - You can delete the trail files and restart Extract process and you would receive the same result – the same DMLs, same CSNs, same commit order (not counting the metadata)

Not talking today about

- Supplemental logging
  - It has to be configured prior to the start of replication
  - Let’s assume that the redo logs contain all necessary information (PK, FK, UI, etc.)
- Trail file format
  - Proper parameters are being used to make sure that the trail files contains all important information (before/after row images)
- Table metadata
  - The schema does not change during the replication
  - All schema metadata is added to the trail (OGG 12.2+)
- Performance parameters
  - Like number of threads, monitoring details, tuning, etc. – there are a lot of MOS notes describing those
- We have enough archived redo logs and old trail files for our purposes
  - Not talking today about how to secure them, when are they allowed to delete
- Transaction splitting used by Parallel Replicat

Checkpoint table

- How to confirm if the transaction is applied to the target database?
- There is no way to transactionally write confirmation to an OGG file and to the database
  - There is a risk that one write will succeed and the other not
  - There is a risk of missed or duplicated transaction
  - Cheating (HANDLECOLLISIONS parameter) does not work in the long run
- The only real solid solution is a checkpoint table
  - An additional technical table in the schema of the target database
  - Checkpoint row UPDATE DML is added to <u>every</u> replicated transaction
  - If the replicated transaction fails the checkpoint table won't be updated
  - For serial replication (parallellism = 1) the table has one row with the SCN of last transaction
  - Works also with transaction grouping
- Not using CHECKPOINT TABLE is only allowed with Classic Replicat (OGG 12.3)

Checkpoint table example

Source Database (redo log)	
SCN 100	T1 insert
SCN 101	T2 update
SCN 102	T3 delete
SCN 103	T1 commit
SCN 104	T2 update
SCN 105	T2 update
SCN 106	T4 delete
SCN 107	T3 rollback
SCN 108	T4 insert
SCN 109	T4 commit
SCN 110	T5 insert
SCN 111	T5 rollback
SCN 112	T2 insert
SCN 113	T2 commit

UPDATE CHECKPOINT_TABLE
SET LAST_APPLIED_SCN = 103

Target Database (applied using a single apply session)
T1 insert
CHK 103
T1 commit
T4 delete
T4 insert
CHK 109
T4 commit
T2 update
T2 update
T2 update
T2 insert
CHK 113
T2 commit

 © 2018 by Adam Leszczyński, all rights reserved

Asynchronous commit

- When you have a checkpoint table in the target database:
  - You do not need to make synchronous commit operations
  - If the target database is restarted we do know what is committed and what not thanks to the checkpoint table
  - The transactions can be reapplied after the database restart/recovery
  - No single transaction would be lost
- Till OGG 11.1.1 the solution was to use
  - *SQLEXEC "alter session set commit_wait = 'NOWAIT'";*
  - Note: requires Oracle 10g R2
- Since OGG 11.1.1.1 (Oct 2011) it is no longer required
  - The Asynchronous Commit is on by default
  - New default parameter: DISABLECOMMITNOWAIT (default off)
  - Still many people did not notice that and still use the OGG 11.1.1 old approach

Serial apply is slow

- Serial apply = applied using 1 apply session
- Source database: there are multiple connections and multiple simultaneous transactions
- Every transactions takes some time:
  - CPU time: changes in the database (tables, indexes, undo, redo logs, etc)
  - Wait time: disk read, commit sync, network time, etc.
- Serialization (executing all of them serially one by one) of all transactions serializes also all operations and takes time:
  - CPU time: for changes in the database,
  - Wait time: not all data is in cache,
- Serial apply does not scale

 © 2018 by Adam Leszczyński, all rights reserved

Scaling replication

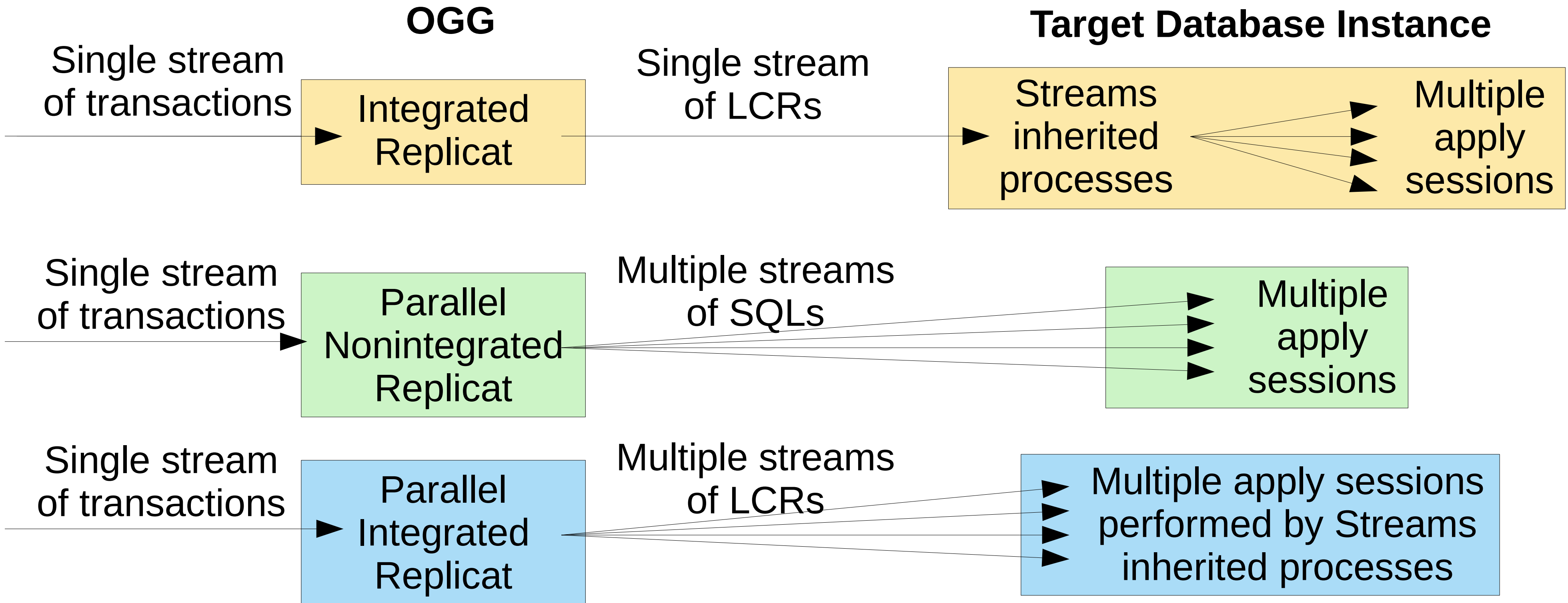
- Multi-threaded Extract — Boring subject, nothing interesting here
- Multiple Extract processes
- Multiple Replicat processes
  - Coordinated Replicat
  - FILTER, @RANGE clauses

  } Boring subject, this is not a transactional approach
- Non-serial Replicat
  - Integrated Replicat
  - Parallel Nonintegrated Replicat
  - Parallel Integrated Replicat

  } Cool stuff

© 2018 by Adam Leszczyński, all rights reserved

Introducing parallelism to Replicat



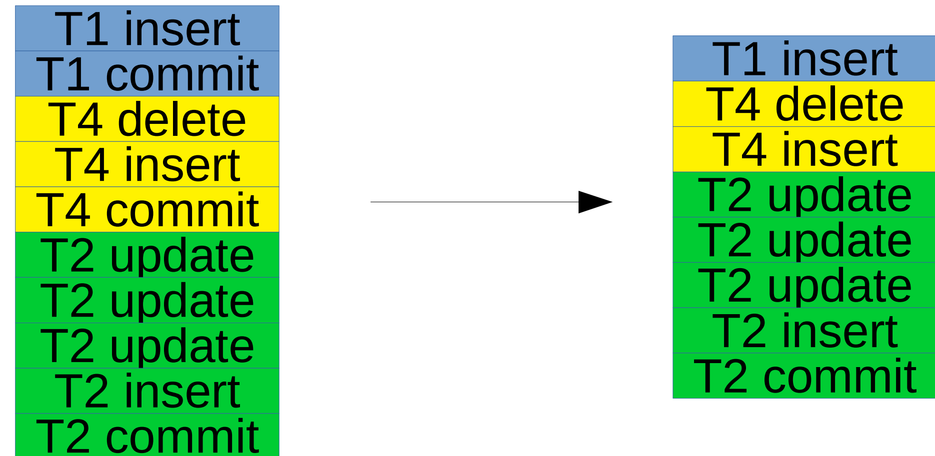
Non-serial Replicat

Function	Integrated Replicat	Parallel Nonintegrated Replicat	Parallel Integrated Replicat
Dividing stream of transactions into multiple apply sessions	Database	OGG	OGG
Transaction format	LCR	SQL	LCR
Supported databases	Oracle	Oracle + non Oracle (future versions)	Oracle
Checkpoint table schema	SYS	User defined	User defined
OGG min. version	12.1	12.3	12.3
Target database min. version	11.2.0.4+	11.2.0.4+	12.2.0.1+

 © 2018 by Adam Leszczyński, all rights reserved

Transaction grouping

- Does not change the order of DMLs, just reduces the number of commit operations
- Integrated Replicat:
  - Controlled by parameter GROUPTRANSOPS
  - Default set to 1
  - Parallelism off (DBOPTIONS INTEGRATEDPARAMS (PARALLELISM 1) sets the value to 50 (when BATCHSQL is not used)
- Parallel Replicat
  - Controlled automatically
  - Probably somehow grouped automatic up to the value of LOOK_AHEAD_TRANSACTIONS (values: 1000 – 100 000, default 10 000)
  - Works with BATCHSQL
  - Can’t turn it off



Transaction batching

- Can be turned on by parameter: BATCHSQL
  - By default off
- Rearranges the DML order within transactions to achieve higher performance
- Primary developed for Classic Repicat, works with other Replicat types but Oracle discourages to use it
- Integrated Replicat:
  - Sets GROUPTRANSOPS to 1 (grouping disabled)
- Parallel Nonintegrated Replicat:
  - Works with on transaction grouping (automatic)
- Parallel Integrated Replicat
  - Nondeterministic behavior (?) – not sure if it is on or off

Delete A
Update B
Insert A
Insert B
Insert A
Delete B
Update B
Insert A
Update A
Commit

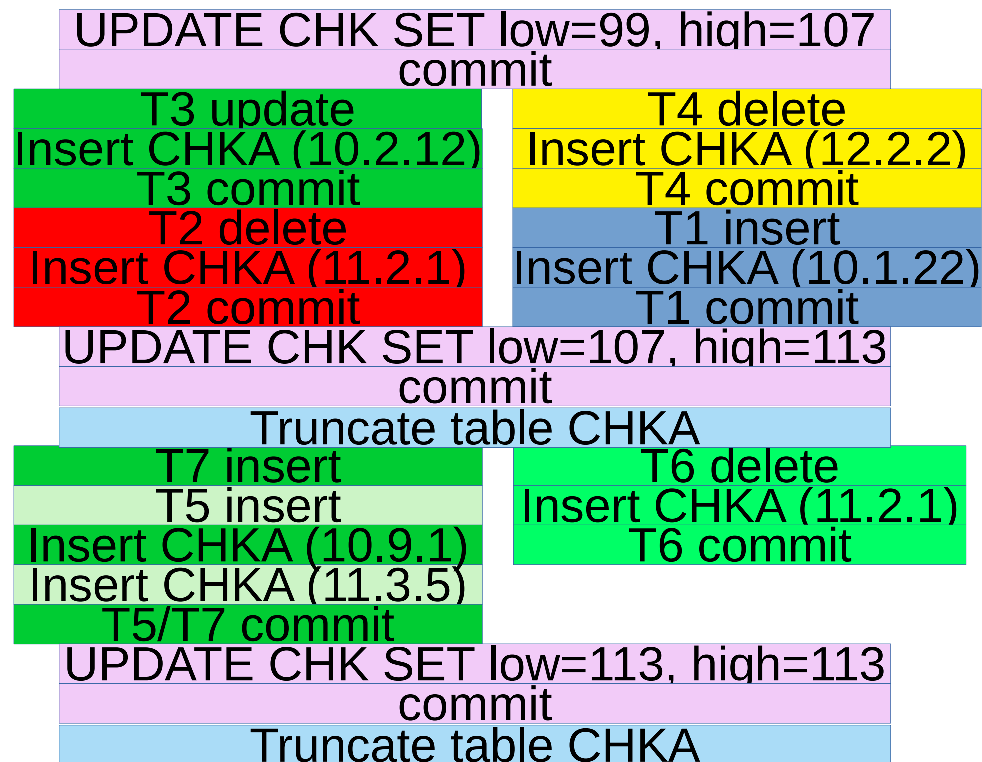
Insert A
Insert A
Insert A
Update A
Delete A
Insert B
Update B
Update B
Delete B
Commit

Checkpoint table – “parallel style”

Source Database
(redo log)

SCN 100	Tx 10.1.22	T1 insert
SCN 101	Tx 10.1.22	T1 commit
SCN 102	Tx 11.2.1	T2 delete
SCN 103	Tx 11.2.1	T2 commit
SCN 104	Tx 10.2.12	T3 update
SCN 105	Tx 10.2.12	T3 commit
SCN 106	Tx 12.2.2	T4 delete
SCN 107	Tx 12.2.2	T4 commit
SCN 108	Tx 11.3.5	T5 insert
SCN 109	Tx 11.3.5	T5 commit
SCN 110	Tx 11.2.1	T6 delete
SCN 111	Tx 11.2.1	T6 commit
SCN 112	Tx 10.9.1	T7 insert
SCN 113	Tx 10.9.1	T7 commit

Target Database



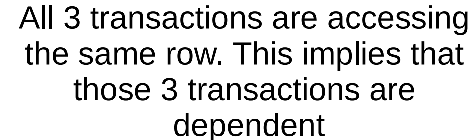
Checkpoint table – “parallel style”

- Multiple checkpoint tables are required:
  - One general with **high/low watermark** of applied SCN
  - One/Many with **csn/transaction id’s** of applied transactions
    - Integrated Replicat: One table
    - Parallel Replicat: Many tables
- This generates additional load for every transaction:
  - Integrated Replicat: +1 additional INSERT per transaction, +1 additional DELETE per transaction
  - Parallel Replicat: +1 additional INSERT per transaction, from time to time: TRUNCATE
  - For non-parallel Replicat (like Classic Replicat) there could be just on update per transaction group
- The high/low watermark is updated from time to time
- For Integrated Replicat (not Parallel Integrated Replicat!) the checkpoint tables reside in **SYS schema**

Transaction dependencies

- Example:

```
CREATE TABLE (
  A NUMERIC PRIMARY KEY,
  B NUMERIC);
INSERT INTO A VALUES (1,1);
COMMIT;
UPDATE A SET B = 2 WHERE A = 1;
COMMIT;
UPDATE A SET B = 3 WHERE A = 1;
COMMIT;
```

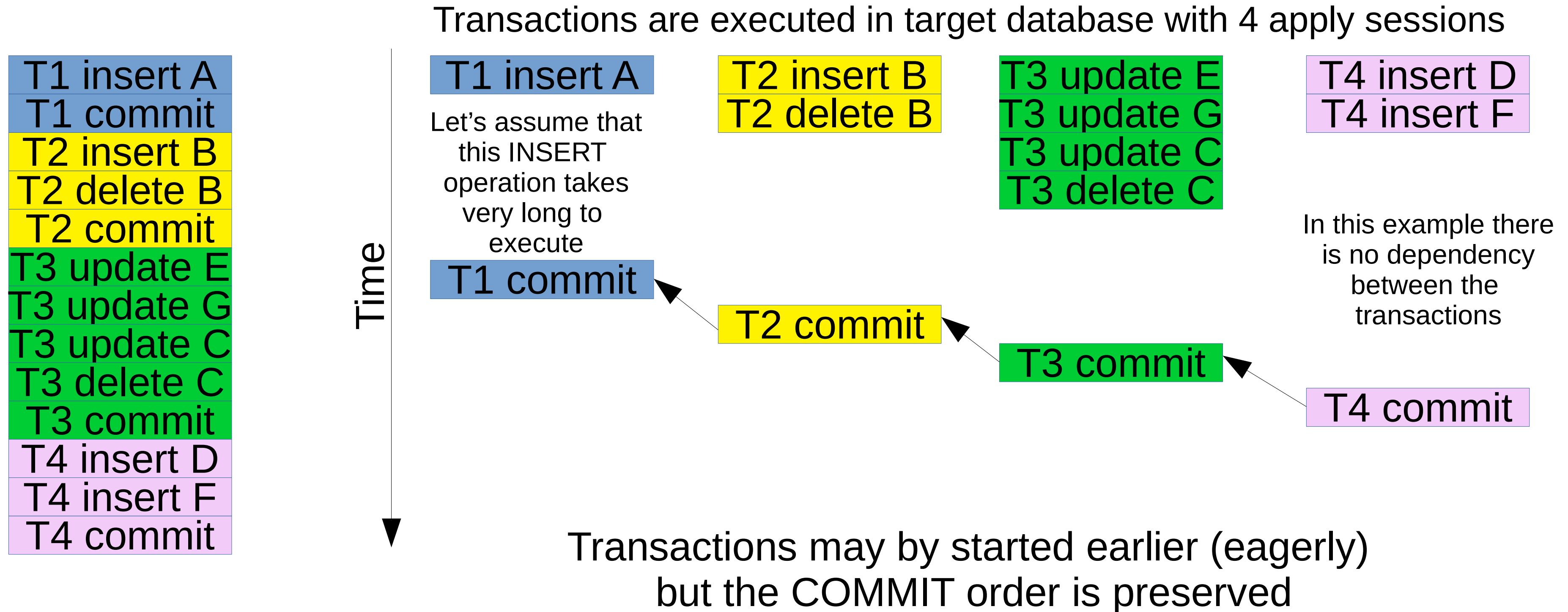


- Integrated/Parallel Replicat:
  - OGG identifies transaction dependencies based on analysis of rows that are modified (knowing what are the constraints)
  - The dependency information is taken into account when the transactions are scheduled – for any identified transaction dependency the order of COMMITs and dependent DMLs may not change

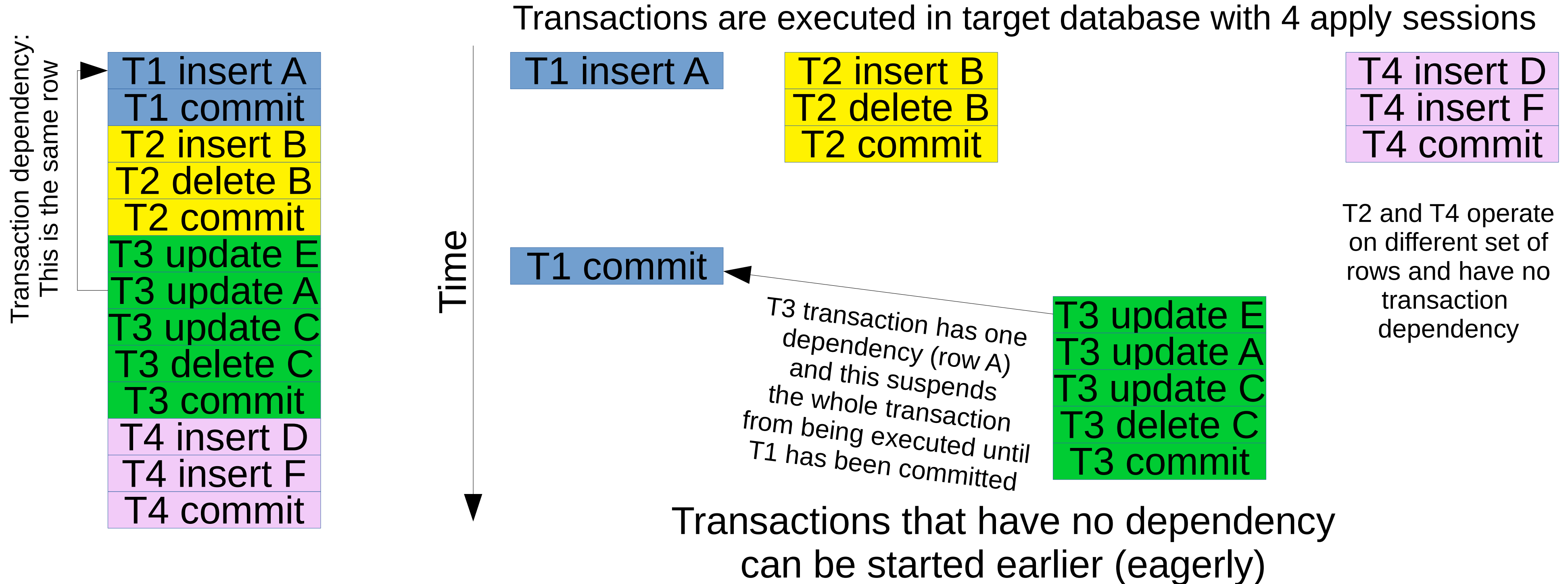
Transaction dependencies

- Primary Key dependency
  - All DMLs which are changing a certain row have to be executed in the same order
- Unique Index dependency
  - Like for PK
- Foreign Key dependency
  - All DMLs which are changing a row that might be referenced have to be executed in the same order
- Dependency calculation requires that certain constraints exists in the source or target database
  - There is no other way to tell OGG that there are constraints to be respected
  - The target database can not have constraints that do no exists in the source database
  - “Conditional” supplemental logging are required for UI and FK constraints
- **Always** honored by: Integrated Replicat & Parallel Replicat
- Note: the constraints do not have to be DEFFERABLE on the target – OGG can defer them anyway

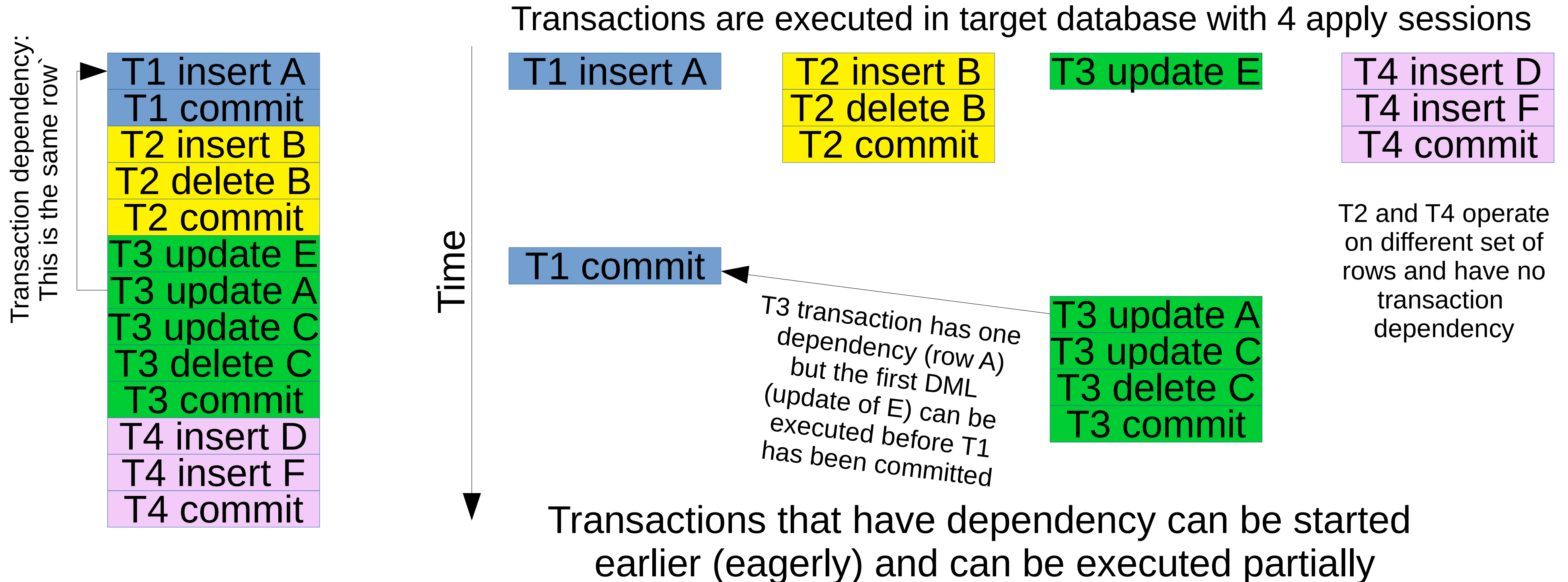
Preserving COMMIT order



Eager start of non-dependent transactions



Eager start of all transactions

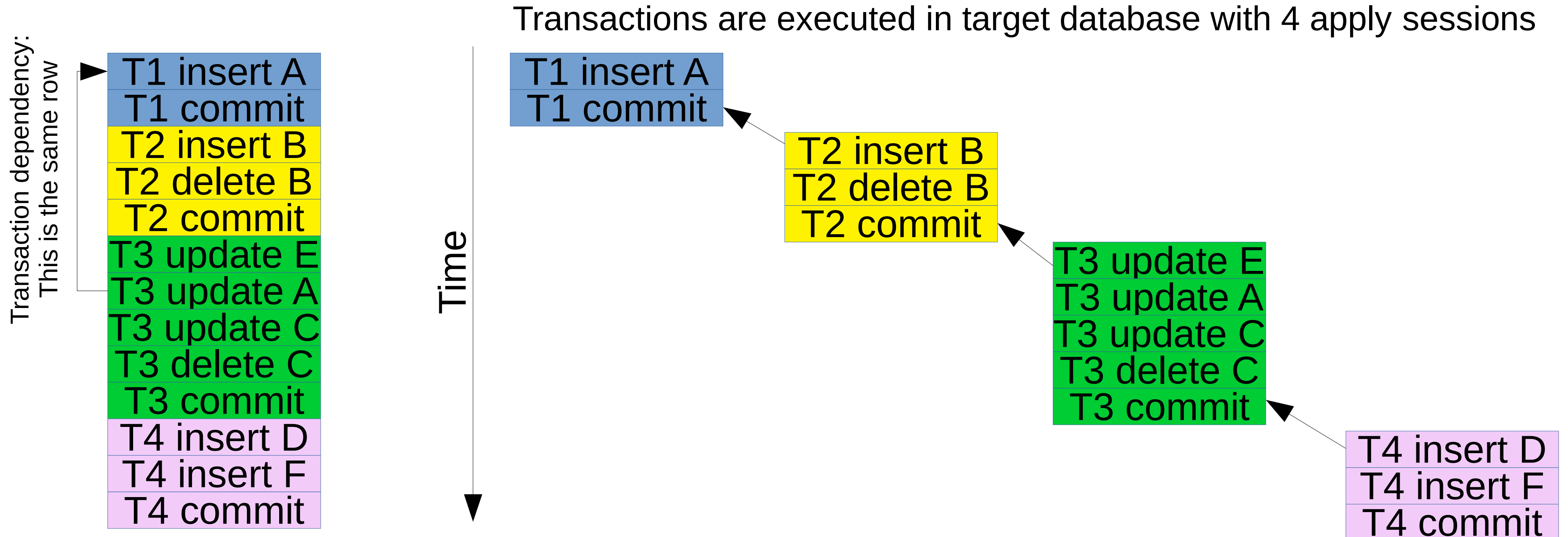


Transactions that have dependency can be started earlier (eagerly) and can be executed partially

Modes of operation

Mode	Preserved commit order	Eager start of non-dependent transactions	Eager start of dependent transactions	Integrated Replicat	Parallel Nonintegrated Replicat	Parallel Integrated Replicat
Parallelism Off	YES			Available	Available	Available
Serialized Transactions	YES	YES	YES	Available		
Dependent (Default)		**YES**		**Available**	**Available**	**Available**
Dependent Eager		YES	YES	Available		

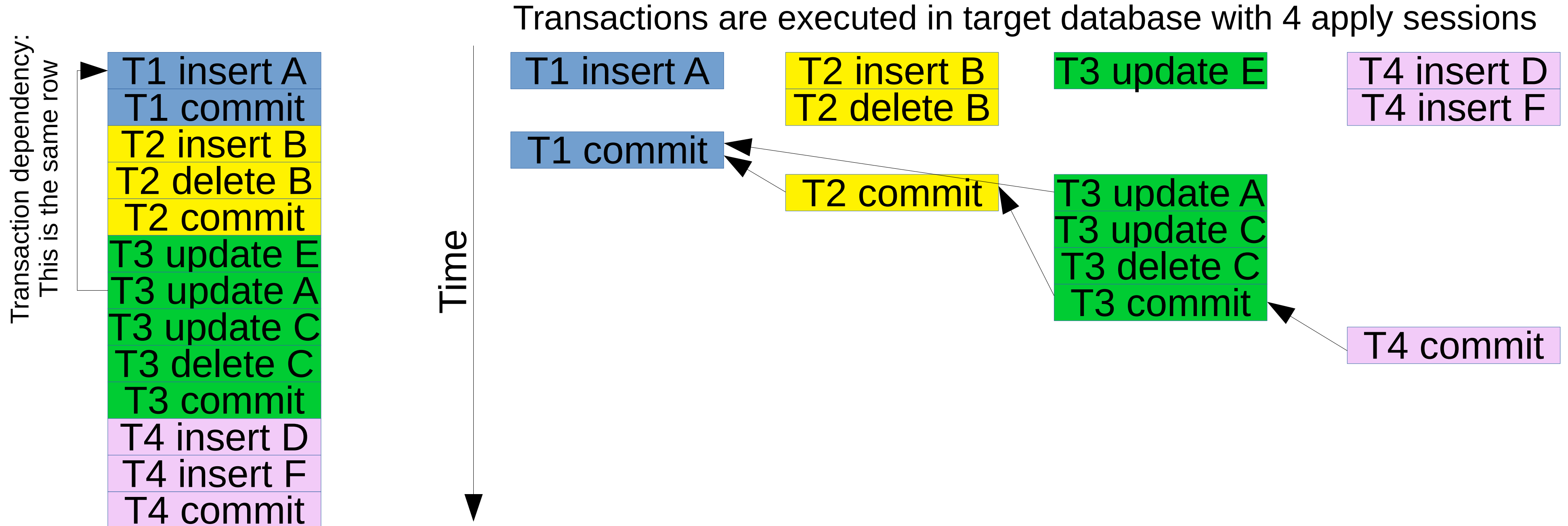
Parallelism Off



Parallelism Off

- Offers maximum safety – works like Classic Replicat
  - **Does not start a following transaction before committing the current transaction**
  - Always using the “parallel style” checkpoint table – Even when working with just 1 apply session
  - Used for “large transactions”
- Integrated Replicat:
  - DBOPTIONS INTEGRATEDPARAMS (PARALLELISM 1)
  - Automatically sets: GROUPTRANSOPS 25
- Parallel Nonintegrated Replicat:
  - APPLY_PARALLELISM 1
  - or: COMMIT_SERIALIZATION
- Parallel Integrated Replicat:
  - APPLY_PARALLELISM 1

Serialized Transactions

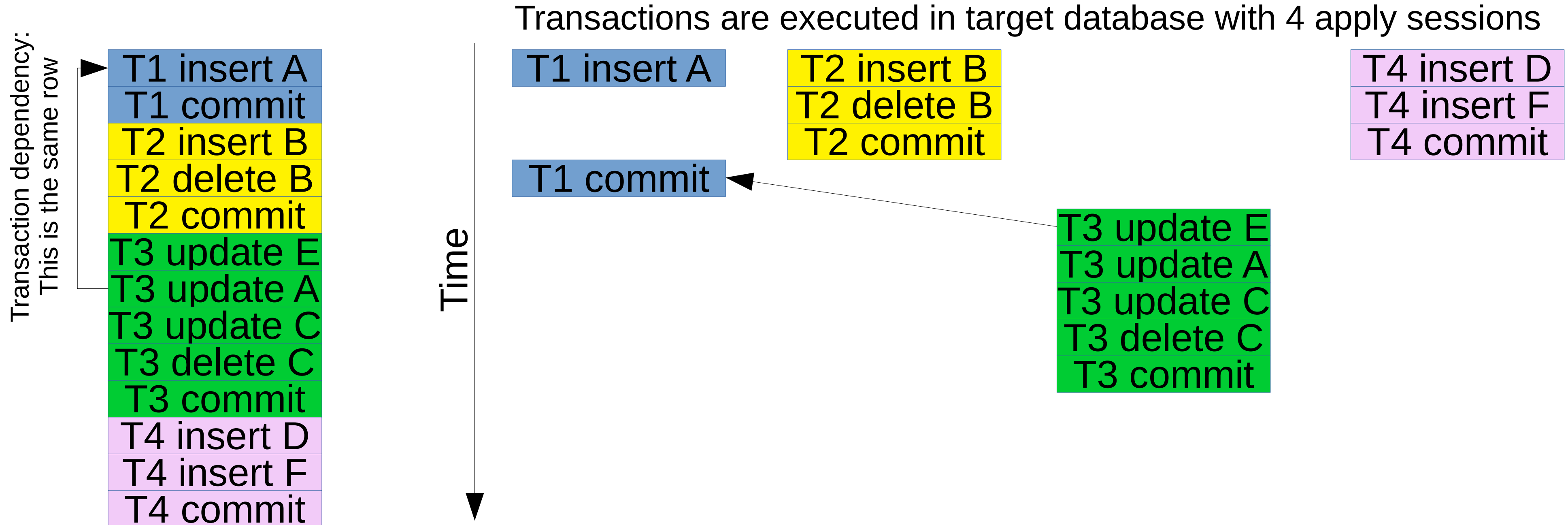


Serialized Transactions

- Start a following transaction before the previous is committed
  - **COMMIT order is preserved**
  - **Transaction dependency is preserved**
  - No risk of changing the transaction order – in case of fault the transactions can be retried using 1 apply session
  - Note: “Parallelism Off” parameters can not be used
- Integrated Replicat:
  - DBOPTIONS INTEGRATEDPARAMS (COMMIT_SERIALIZATION FULL)
  - Note: DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE SEQUENTIAL) it will trigger Dependent Eager mode
  - Note: Unpredictable behavior when BATCHSQL is used (COMMIT order might not be preserved)
- Parallel Nonintegrated Replicat:
  - Not available
  - Note: There is option COMMIT_SERIALIZATION but works like parallelism is off
- Parallel Integrated Replicat
  - Not available

 © 2018 by Adam Leszczyński, all rights reserved

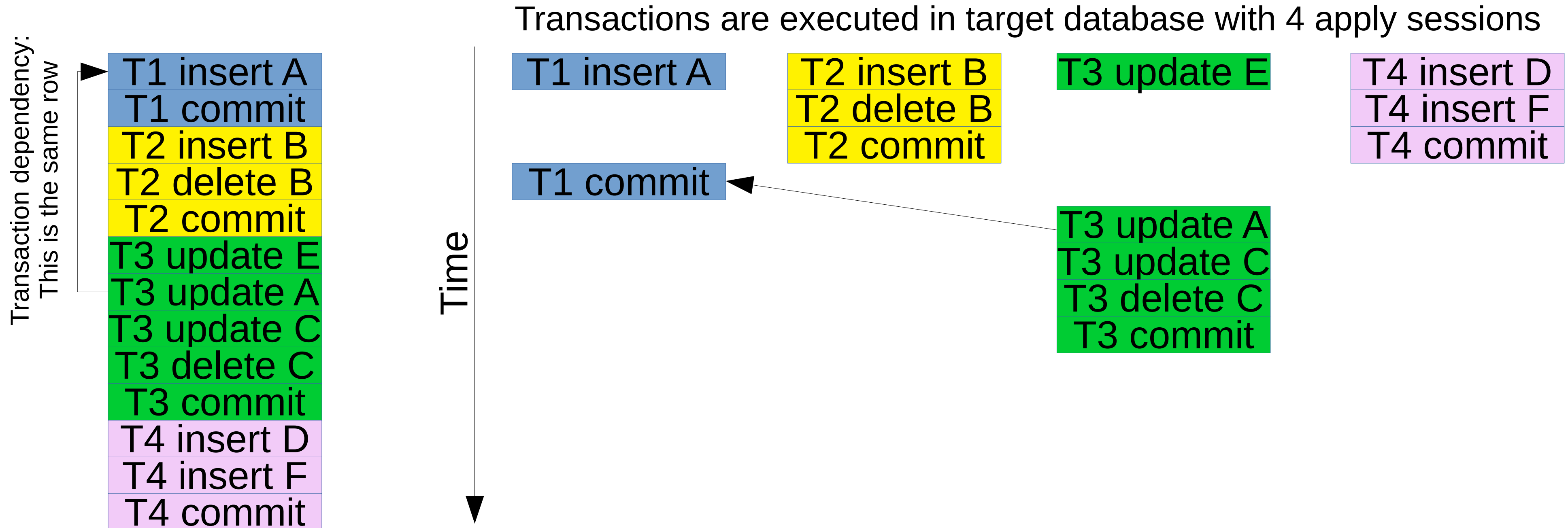
Dependent (Default)



Dependent (Default)

- The default mode – transaction dependencies are honored
  - **COMMIT order is NOT preserved**
  - **Transaction dependency is preserved**
  - **Does NOT “eagerly” starts dependent transactions**
  - Note: “Parallelism Off” parameters can not be used
- Integrated Replicat
  - Default parameter: DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE DEPENDENT)
  - Default parameter: DBOPTIONS INTEGRATEDPARAMS (COMMIT_SERIALIZATION DEPENDENT_TRANSACTIONS)
  - Note: Works with BATCHSQL
- Parallel Nonintegrated Replicat
  - Note: Works with BATCHSQL
- Parallel Integrated Replicat
  - The only mode available

Dependent Eager



© 2018 by Adam Leszczyński, all rights reserved

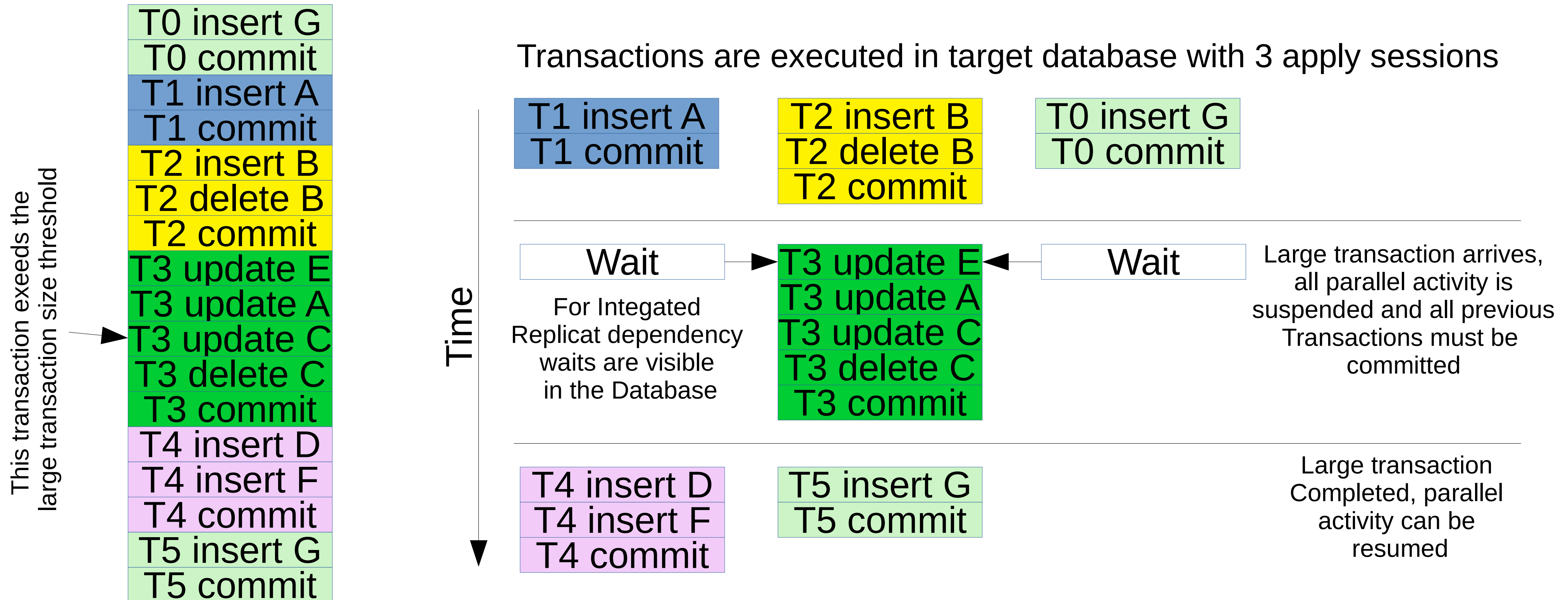
Dependent Eager

- The fastest mode possible – offers the highest possible level of parallelism
  - **COMMIT order is NOT preserved**
  - **Transaction dependency is preserved**
  - **Starts “eagerly” dependent transactions**
  - Note: “Parallelism Off” parameters can not be used
- Integrated Replicat:
  - DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE DEPENDENT_EAGER)
  - Or: DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE SEQUENTIAL)
  - Note: Works with BATCHSQL
- Parallel Nonintegrated Replicat
  - Not available
- Parallel Integrated Replicat
  - Not available

Modes of operation (again)

Mode	Preserved commit order	Eager start of non-dependent transactions	Eager start of dependent transactions	Integrated Replicat	Parallel Nonintegrated Replicat	Parallel Integrated Replicat
Parallelism Off	YES			Available	Available	Available
Serialized Transactions	YES	YES	YES	Available		
Dependent (Default)		**YES**		**Available**	**Available**	**Available**
Dependent Eager		YES	YES	Available		

Large transactions



Large transactions

- When a large transaction arrives
  - For Integrated Replicat: number of LCRs > EAGER_SIZE (default: 15 100 LCRs)
    - Warning: Setting the value high requires a lot of memory (STREAMS_POOL_SIZE)
  - For Parallel Replicat: size > CHUNK_SIZE (default: 1 000 000 000 bytes)
- Action:
  - All preceding transactions have to be committed
  - None following transaction can start until the large transaction has been committed
- Large transactions are executed serially (with parallelism turned off)
  - Even if there are no dependencies
  - Eager start of transactions (dependent and not-dependent) is not used

© 2018 by Adam Leszczyński, all rights reserved

Summary

- If you don’t need transactional consistence in replication use a non-transactional approach like Coordinated Replicat (lower checkpoint table overhead)
- If you don’t know your application – the only safe modes are: Parallelism Off mode and Serialized Transactions
- Start using Parallel Replicat instead of Integrated Replicat?
  - Oracle claims it to be up to 5x faster
  - Advantages:
    - Removed load from database host (cost of licenses $)
    - Handles much bigger transactions in parallel mode (1GB vs 15100 LCRs)
    - Better checkpoint table handling (user schema, uses truncate operations)
    - BATCHSQL works together with GROUPTRANSOPS (Parallel Noninegrated Replicat)
    - Big transactions splitting option (SPLIT_TRANS_RECS)
    - Patching OGG might not require database patching (Classic Parallel Replicat)
  - Disadvantages:
    - New functionality available for only for one year – might require more testing before using in production
    - Doesn’t have Serialized Transactions and Dependent Eager modes yet (OGG 12.3)

Appendix 1: unexpected features

- Integrated Replicat: DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE SEQUENTIAL) works the same as DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE DEPENDENT_EAGER) when BATCHSQL is not used
- Integrated Replicat: BATCHSQL + DBOPTIONS INTEGRATEDPARAMS (COMMIT_SERIALIZATION FULL) – the order of transaction batches is not preserved
- Parallel Integrated Replicat: rearranges commands in transactions (some hybrid mode of BATCHSQL which is always on – might swap INSERT and UPDATE's)
  - Happens in all possible configurations, can't turn off this feature (SR 3-18049467561)
- Parallel Nonintegrated Replicat: option COMMIT_SERIALIZATION is actually turning on serial mode
- Parallel Integrated Replicat accepts Integrated Replicat parameters like: DBOPTIONS INTEGRATEDPARAMS (COMMIT_SERIALIZATION xx) or DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE xx) but have no effect

Appendix 2: Investigated options

Option	Integrated Replicat	Parallel Nonintegrated Replicat	Parallel Integrated Replicat	Notes
DBOPTIONS INTEGRATEDPARAMS (COMMIT_SERIALIZATION FULL | DEPENDENT_TRANSACTIONS)	VALID	-	VALID (?)	Probably those options should not be possible to be used by Parallel Integrated Replicat
DBOPTIONS INTEGRATEDPARAMS (BATCHSQL_MODE DEPENDENT | DEPENDENT_EAGER | SEQUENTIAL)	VALID	-	VALID (?)	
DBOPTIONS INTEGRATEDPARAMS (PARALLELISM xxx)	VALID	-	-	
DBOPTIONS INTEGRATEDPARAMS (EAGER_SIZE xxx)	VALID	-	-	
BATCHSQL	VALID	VALID	VALID	Works very strange on Parallel Integrated Replicat
GROUPTRANSOPS xxx	VALID	-	-	
SPLIT_TRANS_RECS xxx	-	VALID	VALID	Conflicts with COMMIT_SERIALIZATION
COMMIT_SERIALIZATION	-	VALID	-	Conflicts with APPLY_PARALLELISM 1 and SPLIT_TRANS_RECS
LOOK_AHEAD_TRANSACTIONS xxx	-	VALID	VALID	
CHUNK_SIZE xxx	-	VALID	VALID	
APPLY_PARALLELISM xxx	-	VALID	VALID	APPLY_PARALLELISM 1 conflicts with COMMIT_SERIALIZATION
MAP_PARALLELISM xxx	-	VALID	VALID	

That’s all, folks

- I encourage you ALL to test the options I have presented on your environment and please do send me feedback: aleszczynski@bersler.com
- Thank you very much for your attention :)

 © 2018 by Adam Leszczyński, all rights reserved