MySQL - InnoDB

Pre-FOSDEM 2026

About Mayank

- Mayank Prasad
- From Bangalore, India
- M.Tech in Computer science
- Part of MySQL-InnoDB Storage Engine Development team
- Part of MySQL Since 2011

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda

- ➡Configurable REDO logs
- ➡INSTANT ADD/DROP COLUMN
- ➡Q&A

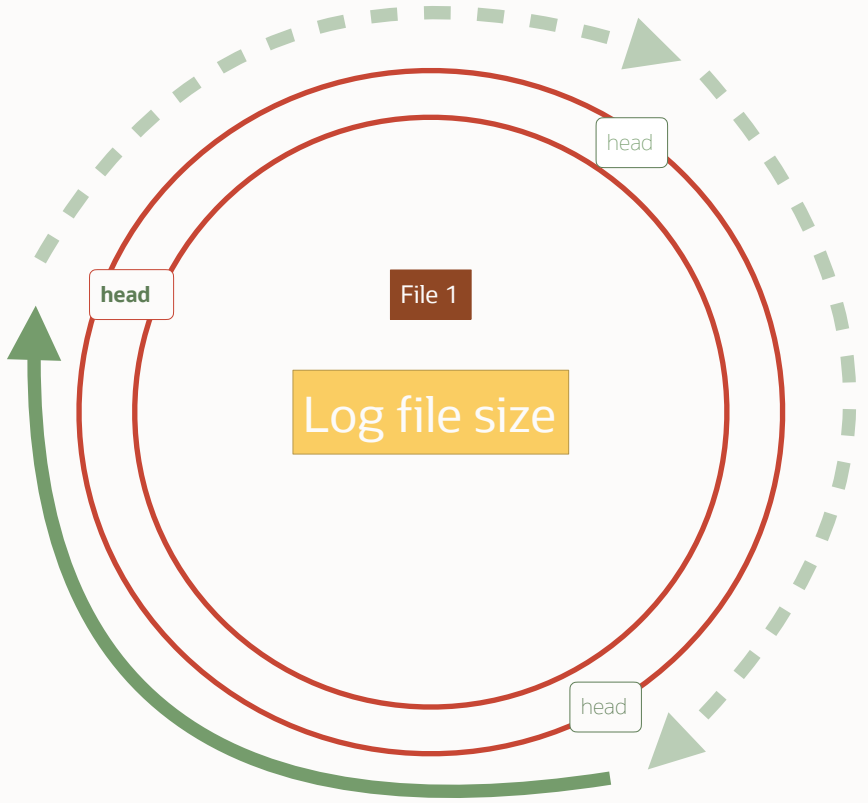
Agenda

- ➡Configurable REDO logs
- ➡INSTANT ADD/DROP COLUMN
- ➡Q&A

Dynamic Configuration of REDOs

Introduction

- REDO Logs are WAL for InnoDB
- To avoid flushing dirty pages (16K) to disk every time they change.
- An on disk circular buffer
- Snake Metaphor : A snake eating it's own tail.



Dynamic Configuration of REDOs

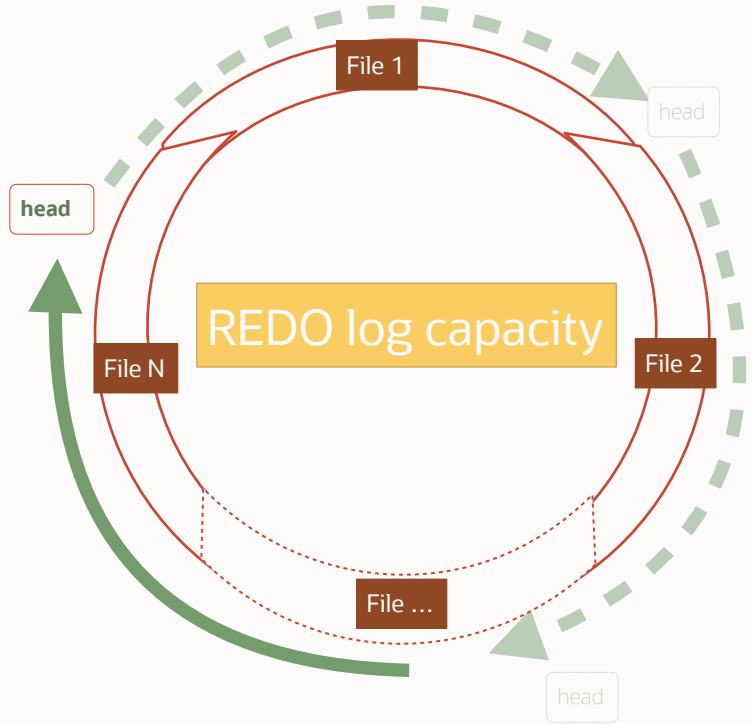
Introduction

- Longer REDOs => Less page flushes but Longer recovery. Slower slow shutdown.
- Shorter REDOs => More page flushes but shorter recovery. Performance impact.
- Trial and error to see what suits according to Workload
- Can specify size of REDO Log file (`innodb_log_file_size`)
- But REDO file size Can't be changed at run time. A server restart is needed
- There were requests to make it dynamic.

Dynamic Configuration of REDOs

Solution

- Still an on disk circular buffer but spanned across multiple files
- Instead of *file size*, we talk in terms of capacity (`innodb_redo_log_capacity`) in bytes.
- Can be changed at runtime i.e. no server restart.
- Number of files and their size is calculated automatically



Dynamic Configuration of REDOs

Solution contd

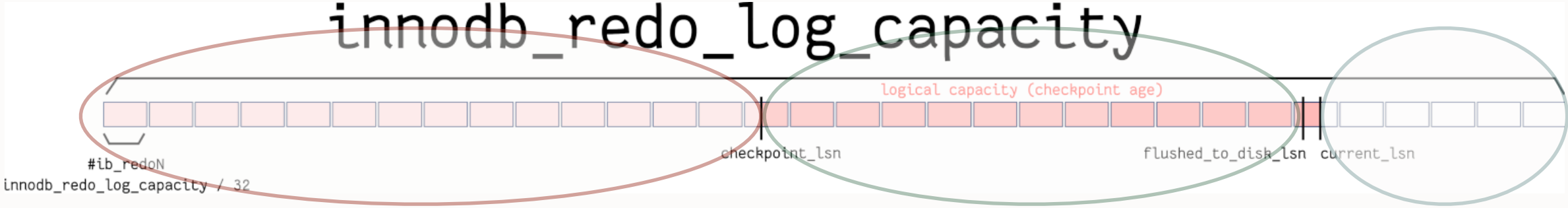
- A new dedicated folder: **#innodb_redo** in MySQL's datadir
- InnoDB creates **32** redo log files

```
[root@dell mysql]# ls \#innodb_redo
'#ib_redo6893'  '#ib_redo6900'      '#ib_redo6907_tmp'  '#ib_redo6914_tmp'  '#ib_redo6921_tmp'
'#ib_redo6894'  '#ib_redo6901_tmp'  '#ib_redo6908_tmp'  '#ib_redo6915_tmp'  '#ib_redo6922_tmp'
'#ib_redo6895'  '#ib_redo6902_tmp'  '#ib_redo6909_tmp'  '#ib_redo6916_tmp'  '#ib_redo6923_tmp'
'#ib_redo6896'  '#ib_redo6903_tmp'  '#ib_redo6910_tmp'  '#ib_redo6917_tmp'  '#ib_redo6924_tmp'
'#ib_redo6897'  '#ib_redo6904_tmp'  '#ib_redo6911_tmp'  '#ib_redo6918_tmp'
'#ib_redo6898'  '#ib_redo6905_tmp'  '#ib_redo6912_tmp'  '#ib_redo6919_tmp'
'#ib_redo6899'  '#ib_redo6906_tmp'  '#ib_redo6913_tmp'  '#ib_redo6920_tmp'
```

- If more space is needed, space can be reclaimed by deleting old "not needed" files and create new to be reused for new REDOs.
- Not needed : Files containing REDOs which are behind checkpoint LSN.

Dynamic Configuration of REDOs

Solution contd



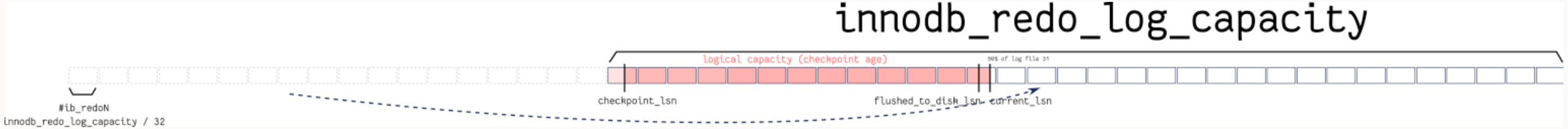
- **checkpoint_lsn** (`Innodb_redo_log_checkpoint_lsn`):
  - an LSN point up to which all changes to the pages are guaranteed to have already been written and fsynced back to tablespace files – basically, the still needed portion of redo log starts here.

- **current_lsn** (`Innodb_redo_log_current_lsn`):
  - the last written position in the redo log. That write could still be buffered inside MySQL processes buffer.

- **flushed_to_disk_lsn** (`Innodb_redo_log_flushed_to_disk_lsn`):
  - the last position in the redo log that InnoDB has been flushed to disk.

Dynamic Configuration of REDOs

Solution contd



- When needed, the log files governor will perform some cleanup and some active files that are not needed anymore will become the new spare ones
- When the background thread is not able to remove a log file from the left to put it to the right, the user transaction will get stuck waiting for REDO buffers to be written to disk. DBAs get warning in the error log notifying them to increase the InnoDB Redo Log Capacity

```
[Warning] [MY-013865] [InnoDB] Redo log writer is waiting for a new redo log file.
 Consider increasing innodb_redo_log_capacity.
```

Dynamic Configuration of REDOs

Observability

- The new Redo Log is instrumented in `performance_schema` in the table **`innodb_redo_log_files`**

```
MySQL  localhost  information_schema  2022-08-25 16:58:21
SQL  select * from performance_schema.innodb_redo_log_files;
+---------+-----------------------------+-----------+-----------+---------------+---------+----------------+
| FILE_ID | FILE_NAME                   | START_LSN | END_LSN   | SIZE_IN_BYTES | IS_FULL | CONSUMER_LEVEL |
+---------+-----------------------------+-----------+-----------+---------------+---------+----------------+
|    7498 | ./#innodb_redo/#ib_redo7498 | 4401811456 | 4402071552 |        262144 |       1 |              0 |
|    7499 | ./#innodb_redo/#ib_redo7499 | 4402071552 | 4402331648 |        262144 |       1 |              0 |
|    7500 | ./#innodb_redo/#ib_redo7500 | 4402331648 | 4402591744 |        262144 |       1 |              0 |
|    7501 | ./#innodb_redo/#ib_redo7501 | 4402591744 | 4402851840 |        262144 |       1 |              0 |
|    7502 | ./#innodb_redo/#ib_redo7502 | 4402851840 | 4403111936 |        262144 |       0 |              0 |
+---------+-----------------------------+-----------+-----------+---------------+---------+----------------+
5 rows in set (0.0003 sec)
```

- This means there are 5 active redo log files and 27 (32-5) spare ones (_tmp):

```
[root@dell mysql]# ls \#innodb_redo
'#ib_redo7498'      '#ib_redo7505_tmp'  '#ib_redo7512_tmp'  '#ib_redo7519_tmp'  '#ib_redo7526_tmp'
'#ib_redo7499'      '#ib_redo7506_tmp'  '#ib_redo7513_tmp'  '#ib_redo7520_tmp'  '#ib_redo7527_tmp'
'#ib_redo7500'      '#ib_redo7507_tmp'  '#ib_redo7514_tmp'  '#ib_redo7521_tmp'  '#ib_redo7528_tmp'
'#ib_redo7501'      '#ib_redo7508_tmp'  '#ib_redo7515_tmp'  '#ib_redo7522_tmp'  '#ib_redo7529_tmp'
'#ib_redo7502'      '#ib_redo7509_tmp'  '#ib_redo7516_tmp'  '#ib_redo7523_tmp'
'#ib_redo7503_tmp'  '#ib_redo7510_tmp'  '#ib_redo7517_tmp'  '#ib_redo7524_tmp'
'#ib_redo7504_tmp'  '#ib_redo7511_tmp'  '#ib_redo7518_tmp'  '#ib_redo7525_tmp'
```

Dynamic Configuration of REDOs

Observability contd

- Resizing
  - Eg : make capacity to 200MB

```
SET GLOBAL innodb_redo_log_capacity =200*1024*1024;
```

- Resizing Up
  - Instant
- Resizing Down
  - Need to wait for existing REDOs to get truncated.
  - Happens in background

- Resizing status

```
mysql> SHOW STATUS LIKE 'Innodb_redo_log_resize_status';
+-------------------------------+------------------+
| Variable_name                 | Value            |
+-------------------------------+------------------+
| Innodb_redo_log_resize_status | Resizing down    |
+-------------------------------+------------------+

mysql> SHOW STATUS LIKE 'Innodb_redo_log_resize_status';
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| Innodb_redo_log_resize_status | OK    |
+-------------------------------+-------+
```

- Current REDO log capacity

```
mysql> SHOW STATUS LIKE 'Innodb_redo_log_capacity_resized';
+---------------------------------+-----------+
| Variable_name                   | Value     |
+---------------------------------+-----------+
| Innodb_redo_log_capacity_resized | 104857600 |
+---------------------------------+-----------+
```

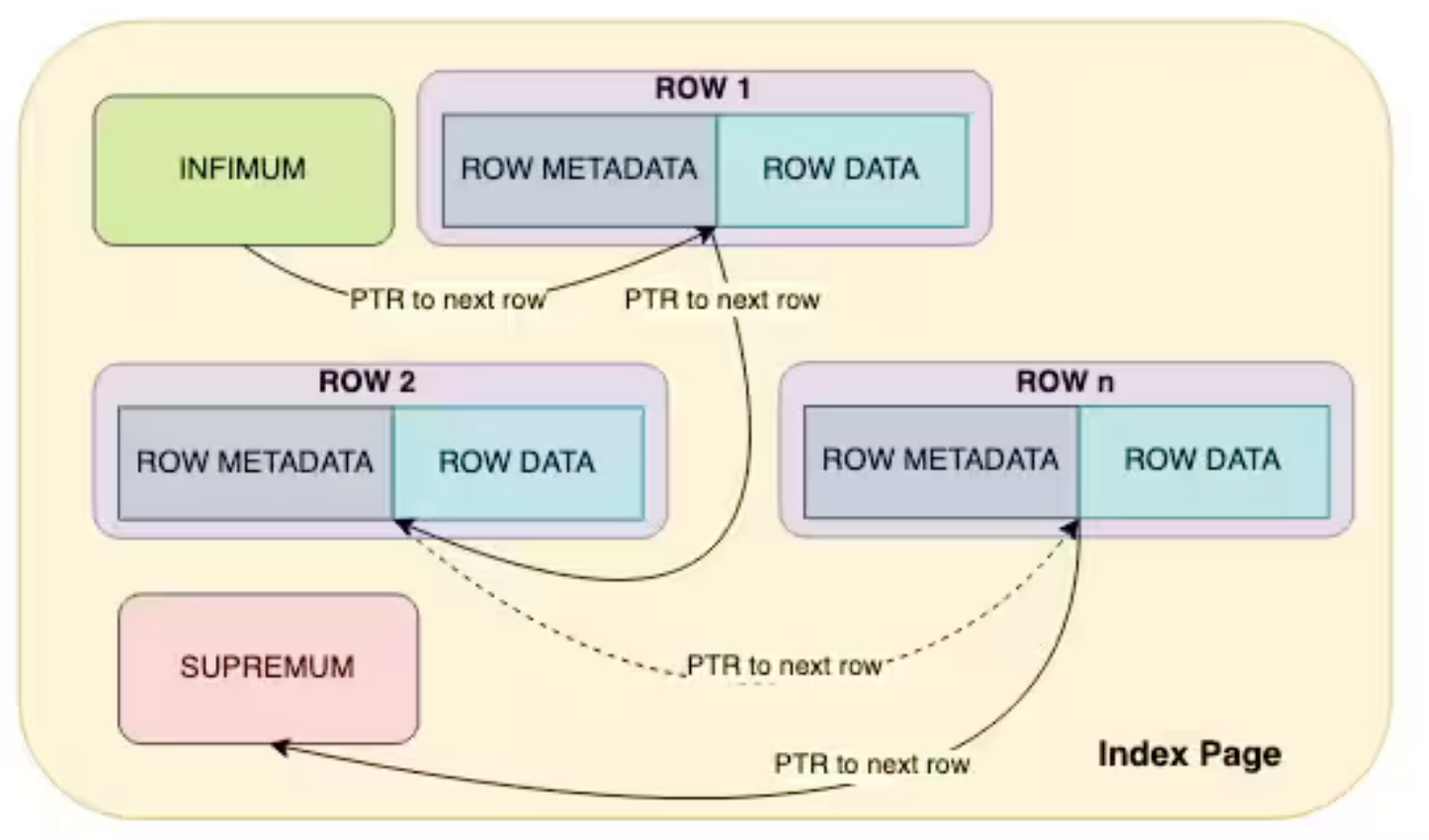
Agenda

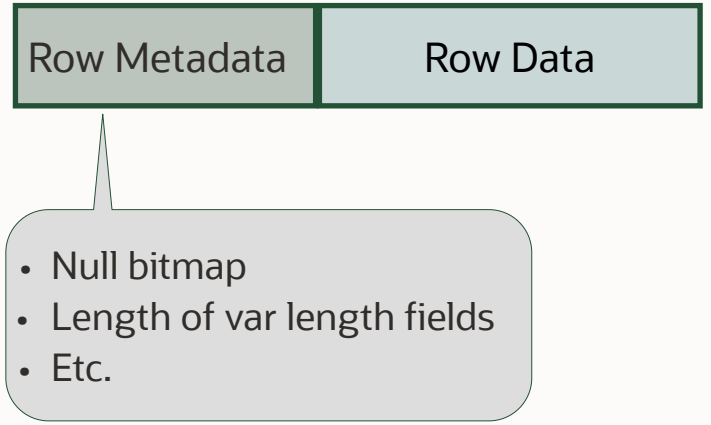
- Configurable REDO logs
- **INSTANT ADD/DROP COLUMNS**
- Q&A

INSTANT ADD/DROP Column

Introduction

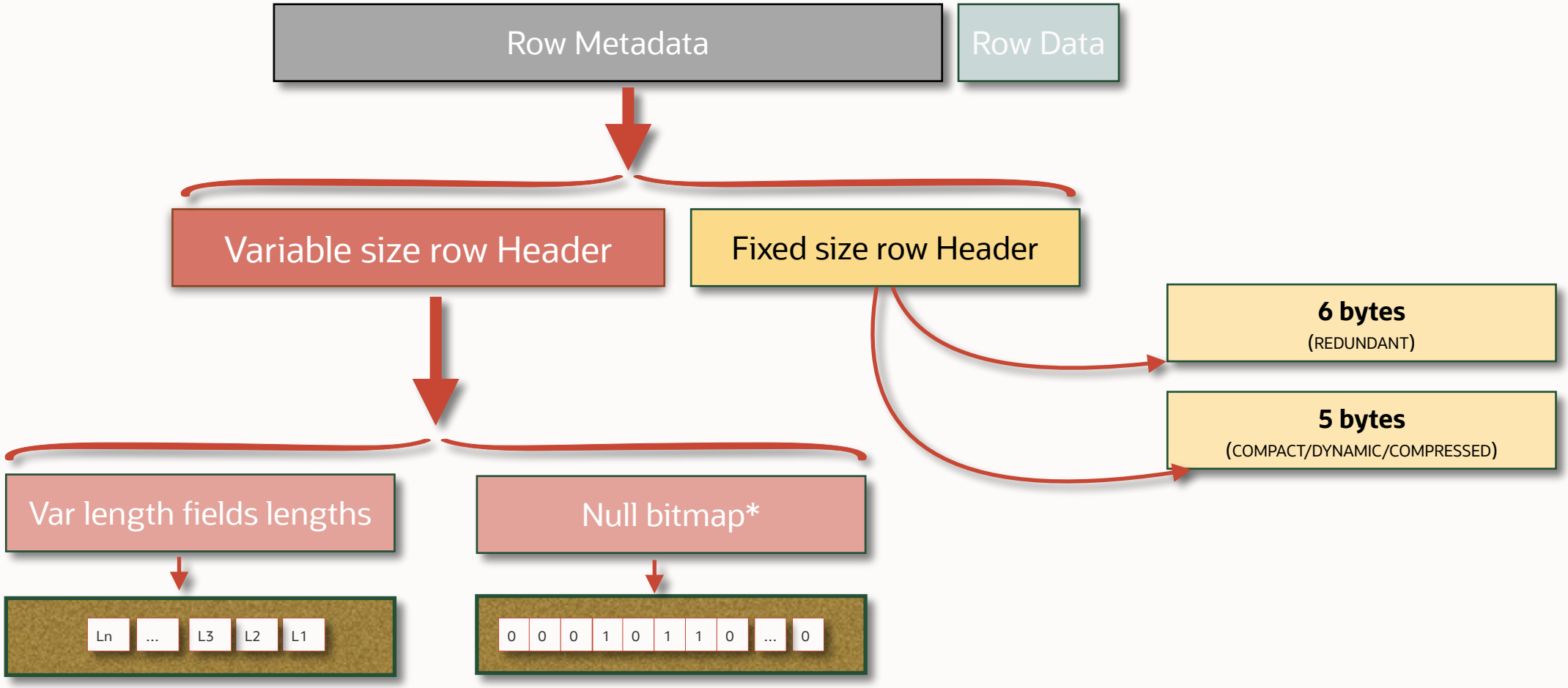
- InnoDB on disk Row Format





INSTANT ADD/DROP Column

InnoDB Row Header



**null bitmap is not present in REDUNDANT row format*

INSTANT ADD/DROP Column

InnoDB Row Format Impact

- Each record header keeps columns specific info.
- Add/Drop column would need each row header to be modified.
- Causes table rebuild
- Bigger the table, more the time it take.
- Costly - time, disk, resource, schedule
- Table lock
- Replication - Master does it. Then Replica does the same.
- Big requirement from users/customers

INSTANT ADD/DROP Column

Idea behind solution

- Only metadata change.
- No table rebuild.
- No table lock required for long duration
- Saves time/space/resources
- Replication would benefit.

INSTANT ADD/DROP Column

Earlier attempt and new design

- `MySQL 8.0.12`
  - `ALTER TABLE t1 ADD COLUMN … ALGORITHM=INSTANT;`
  - Limitations
    - Only ADD COLUMN support
    - Column can be added only at the end. `AFTER/FIRST` clause throws error.

- New design
  - Can add column at any place.
  - Design work for DROP column as well. So column can be dropped from any place.
  - `ALTER TABLE t1 ADD COLUMN [AFTER/FIRST] … ALGORITHM=INSTANT;`
  - `ALTER TABLE t1 DROP COLUMN … ALGORITHM=INSTANT;`

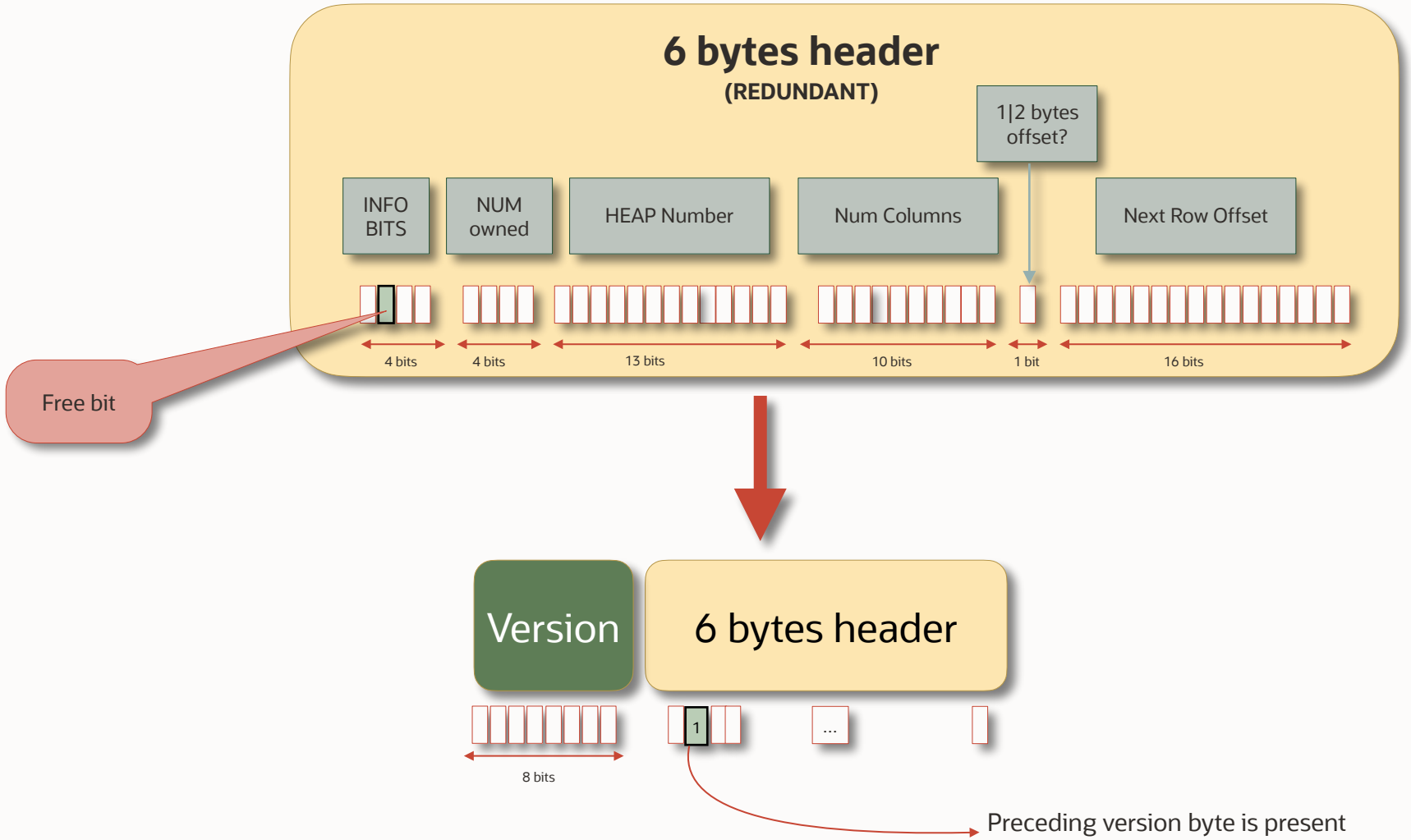
INSTANT ADD/DROP Column

Solution : New design, Row versions

- Introduced “Row versioning”
- Every time a new INSTANT ADD/DROP Done, bump the versions
- Keep information of columns in the version they are ADDed or DROPped.
- Stamp this version on the row when it is inserted/updated.
- When fetched, rows are transformed (if needed) to current row version.

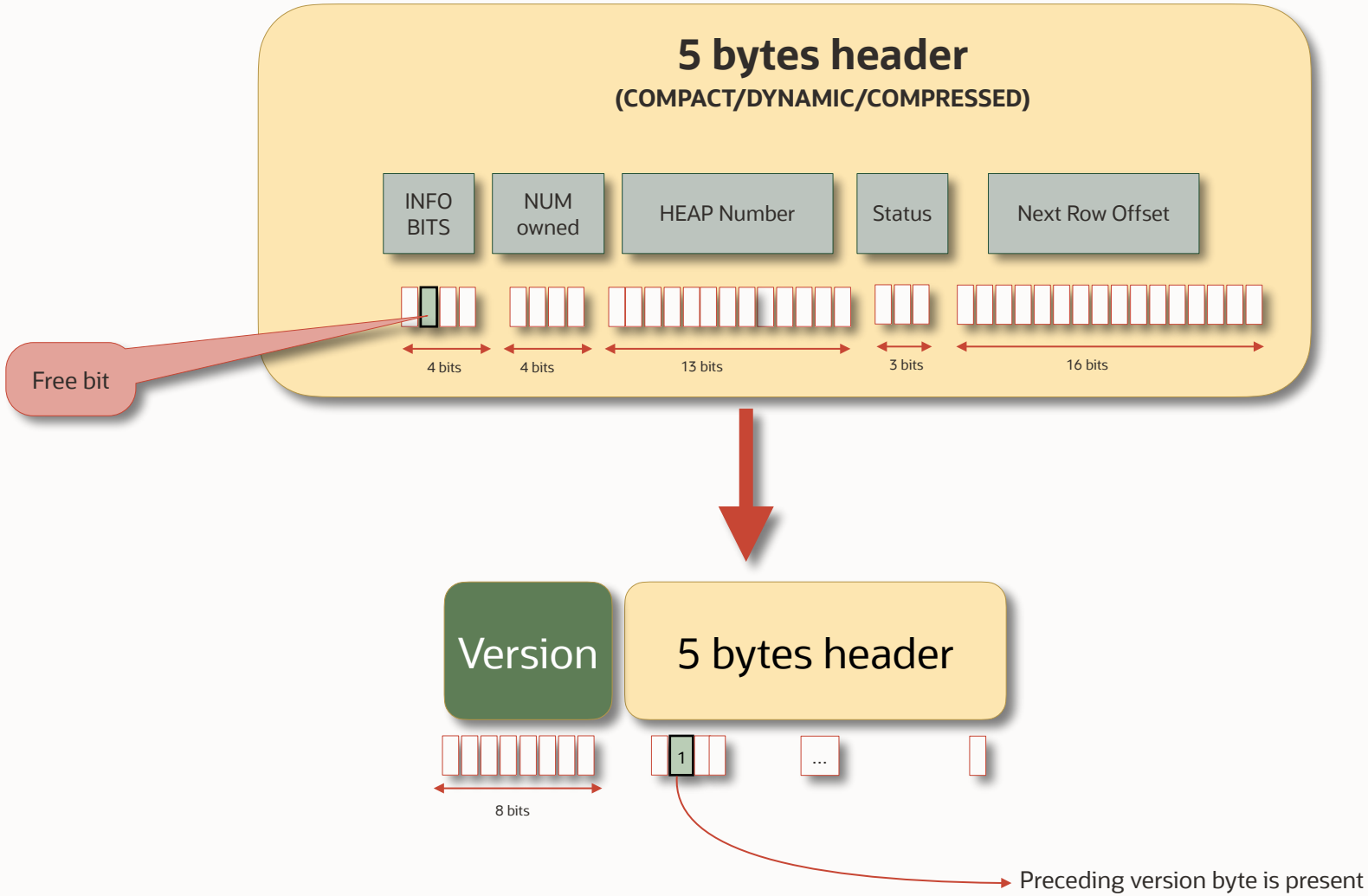
INSTANT ADD/DROP Column

Solution : On disk row format change



INSTANT ADD/DROP Column

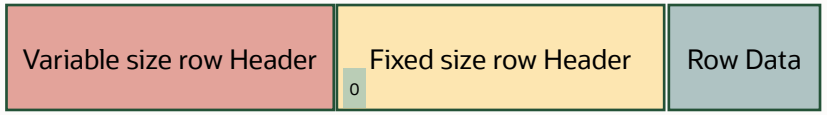
Solution : On disk row format change contd.



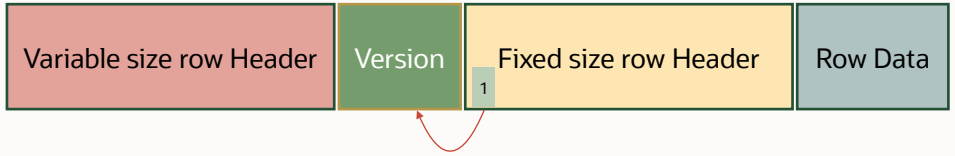
INSTANT ADD/DROP Column

Solution : On disk row format change contd.

- A row with no row version.
  - Bit in INFO BITS is 0.
  - No version on row on disk



- A row with no row version.
  - Bit in INFO BITS is 1.
  - Version is stamped on row on disk



INSTANT ADD/DROP Column

Solution : Transforming rows while fetch

New info on rows on disk

- **`rec_version`**

New column metadata

- **`version_added`** : The row version in which the column I added (0 if it was present during CREATE)
- **`version_dropped`** : The row version in which the column is dropped (0 if it is present currently)

When a record is read, with above informations, it is easy to determine which columns are present in the record.

And while fetching :

- Ignore all the columns value with **`version_dropped > 0`**
- Use default value for columns with **`version_added > rec_version`**

INSTANT ADD/DROP Column

Solution : Transforming rows while fetch

SQL	C1	C2	C3	C4	C5	Version
CREATE TABLE	Y	Y	Y	Y	-	0
ALTER TABLE … DROP C3	Y	Y	D	Y	-	1
ALTER TABLE … ADD C5 … DEFAULT C5Default	Y	Y	-	Y	A	2

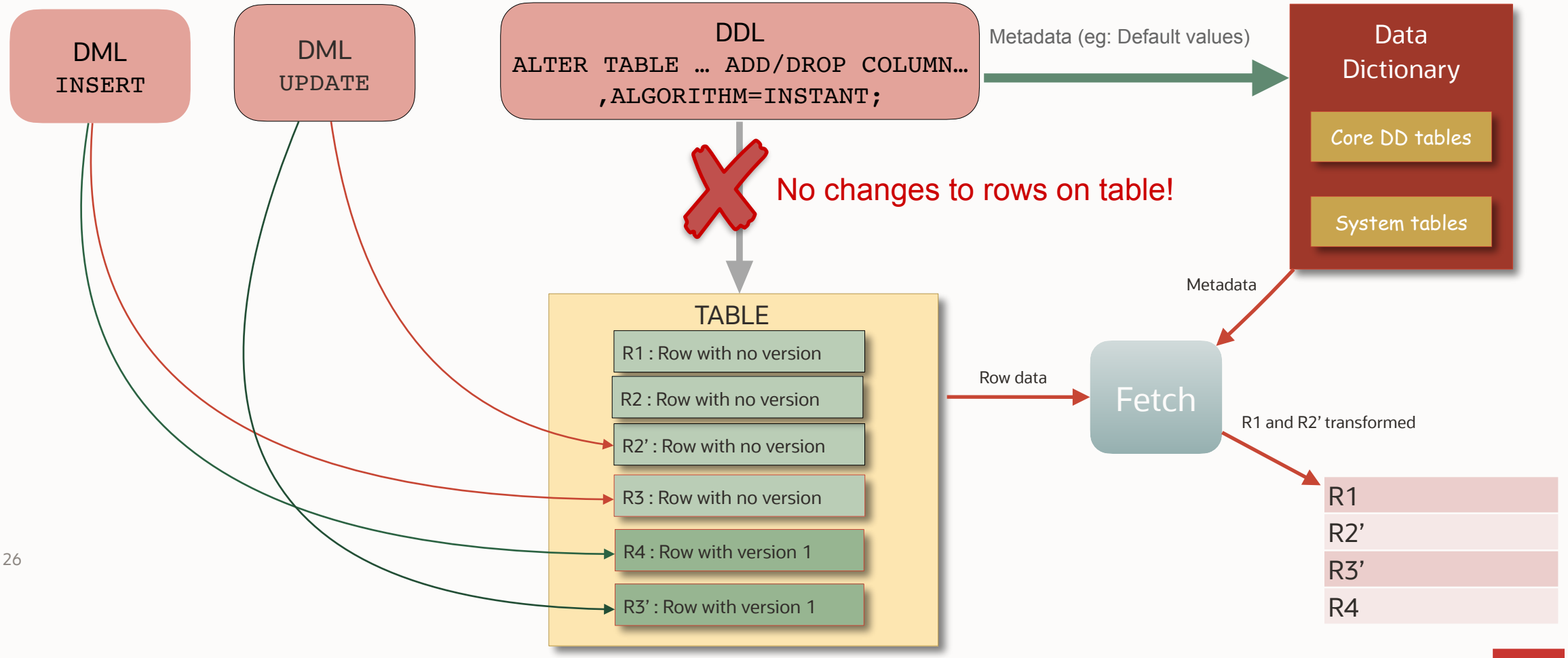
Insert

- (nothing stamped) rec_version 0
- Current version is 2.
- C3 was present on disk and to be ignored
- C5 wasn't present so need to populate default value

Column	version_added	version_dropped	Default value
C1	0	0	
C2	0	0	
C3	0	1	
C4	0	0	
C5	2	0	C5Default

INSTANT ADD/DROP Column

Solution : Flow explanation



INSTANT ADD/DROP Column

Default ALGORITHM

- For `ALTER TABLE … ADD/DROP COLUMN …`
  - `ALGORITHM=INSTANT` is **default**. I.e. if not specified explicitly we try to use INSTANT algorithm.
  - If `INSTANT` is not possible, it falls back to `ALGORITHM=INPLACE/COPY`.
- Once **`OPTIMIZE TABLE`** and other **`ALTER TABLE`** DDLs, which cause tables to rebuild, are executed, INSTANT metadata is reset.
- Column metadata is reset in DD
- row_version is reset to 0.
- **`TRUNCATE TABLE`** resets the INSTANT metadata (no row on disk to interpret).

INSTANT ADD/DROP Column

ROW version limit

- Not frequent operations.
- Maximum row versions allowed are 255.
- After that `ALGORITHM=INSTANT` throws with following error

```
ERROR 4080 (HY000): Maximum row versions reached for table test/t1. No more columns can be added or dropped
instantly. Please use COPY/INPLACE.
```

- If `ALGORITHM=INSTANT` is not specified, it falls back to `ALGORITHM=COPY` automatically.
- As it rebuilds the table, INSTANT Metadata from each row is reset
- Column metadata is reset in DD
- row_version is reset to 0.

INSTANT ADD/DROP Column

Compatibility : Upgrade

- Earlier ADD COLUMN has different row format.
- No need to transform any rows on disk during upgrade.
- After upgrade we might have table which has
  - Rows INSERTED **before** earlier INSTANT ADD was done
  - Rows INSERTED **after** earlier INSTANT ADD was done
  - Rows INSERTED **before** new INSTANT ADD/DROP was done
  - Rows INSERTED **after** new INSTANT ADD/DROP was done
- These rows are transformed correctly when fetched and.
- Once table is rebuilt (`OPTIMIZE TABLE, ALTER` etc.) Instant metadata is cleared.

INSTANT ADD/DROP Column

Compatibility : IMPORT/EXPORT

- EXPORT
  - We write INSTANT metadata in the cfg file.
- IMPORT
  - cfg file is mandatory
  - allowed even if target table has only the columns which exists (eg: `CREATE TABLE LIKE`)
  - INSTANT metadata is populated from CFG file. Rows are interpreted correctly.

SOURCE (INSTANT ADD/DROP)	DESTINATION (INSTANT ADD/DROP)	ALLOWED	REMARK
NO	NO	YES	-
YES	NO	YES	-
NO	YES	NO	-
YES	YES	YES/NO	Allowed if INSTANT metadata matches

INSTANT ADD/DROP Column

Observability

```
mysql> CREATE TABLE t1 (c1 INT);
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT N_COLS, INSTANT_COLS, TOTAL_ROW_VERSIONS
FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME LIKE
"%t1%";
+--------+--------------+--------------------+
| N_COLS | INSTANT_COLS | TOTAL_ROW_VERSIONS |
+--------+--------------+--------------------+
|      4 |            0 |                  0 |
+--------+--------------+--------------------+
1 row in set (0.02 sec)
```

1, if column is added earlier INSTANT ADD implementation

```
mysql> SELECT HAS_DEFAULT, DEFAULT_VALUE FROM
INFORMATION_SCHEMA.INNODB_COLUMNS WHERE NAME LIKE "%c1%";
+-------------+-------------------------------+
| HAS_DEFAULT | DEFAULT_VALUE                 |
+-------------+-------------------------------+
|           0 | NULL                          |
+-------------+-------------------------------+
1 row in set (0.03 sec)
```

```
mysql> ALTER TABLE t1 ADD COLUMN c2 INT DEFAULT 100
FIRST, ALGORITHM=INSTANT;
Query OK, 0 rows affected (0.05 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> SELECT N_COLS, INSTANT_COLS, TOTAL_ROW_VERSIONS
FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME LIKE
"%t1%";
+--------+--------------+--------------------+
| N_COLS | INSTANT_COLS | TOTAL_ROW_VERSIONS |
+--------+--------------+--------------------+
|      5 |            0 |                  1 |
+--------+--------------+--------------------+
1 row in set (0.02 sec)
```

```
mysql> SELECT HAS_DEFAULT, DEFAULT_VALUE FROM
INFORMATION_SCHEMA.INNODB_COLUMNS WHERE NAME LIKE "%c2%";
+-------------+-------------------------------+
| HAS_DEFAULT | DEFAULT_VALUE                 |
+-------------+-------------------------------+
|           1 | 0x3830303030303634            |
+-------------+-------------------------------+
1 row in set (0.03 sec)
```

INSTANT ADD/DROP Column

Example and Limitation

Here is what I tried on my system :

```
mysql> select count(*) from t1;
+----------+
| count(*) |
+----------+
|  8388608 |
+----------+
1 row in set (0.22 sec)

mysql> alter table t1 add column c4 char(10), algorithm=copy;
Query OK, 8388608 rows affected (29.17 sec)
Records: 8388608  Duplicates: 0  Warnings: 0

mysql> alter table t1 add column c5 char(10), algorithm=instant;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```


INSTANT ADD/DROP Column

Limitations

- No support for COMPRESSED tables (seldom used)
- No support for tables with a full-text index
- No support for table residing in DD Tablespace (Not needed)
- No support for temporary tables
- More infos :
  - https://blogs.oracle.com/mysql/mysql-80-instant-add-drop-columns
  - https://blogs.oracle.com/mysql/mysql-80-instant-add-and-drop-columns-2

Thank you


ORACLE

Dynamic Configuration of REDOs

The Snake Metaphor

- A snake which can move from one cage to another connected cage.
- Needed redo log => snake body
- Each redo file => A cage snake moves into.
- When reached to the last cage, cages from left to be move to right side so that snake can move forward.
- Size of snake shirks when checkpoint_lsn moves forward
- Size of snake increases when more redos are written.