Recovery Principles in MySQL Cluster 5.1

Mikael Ronström
Senior Software Architect

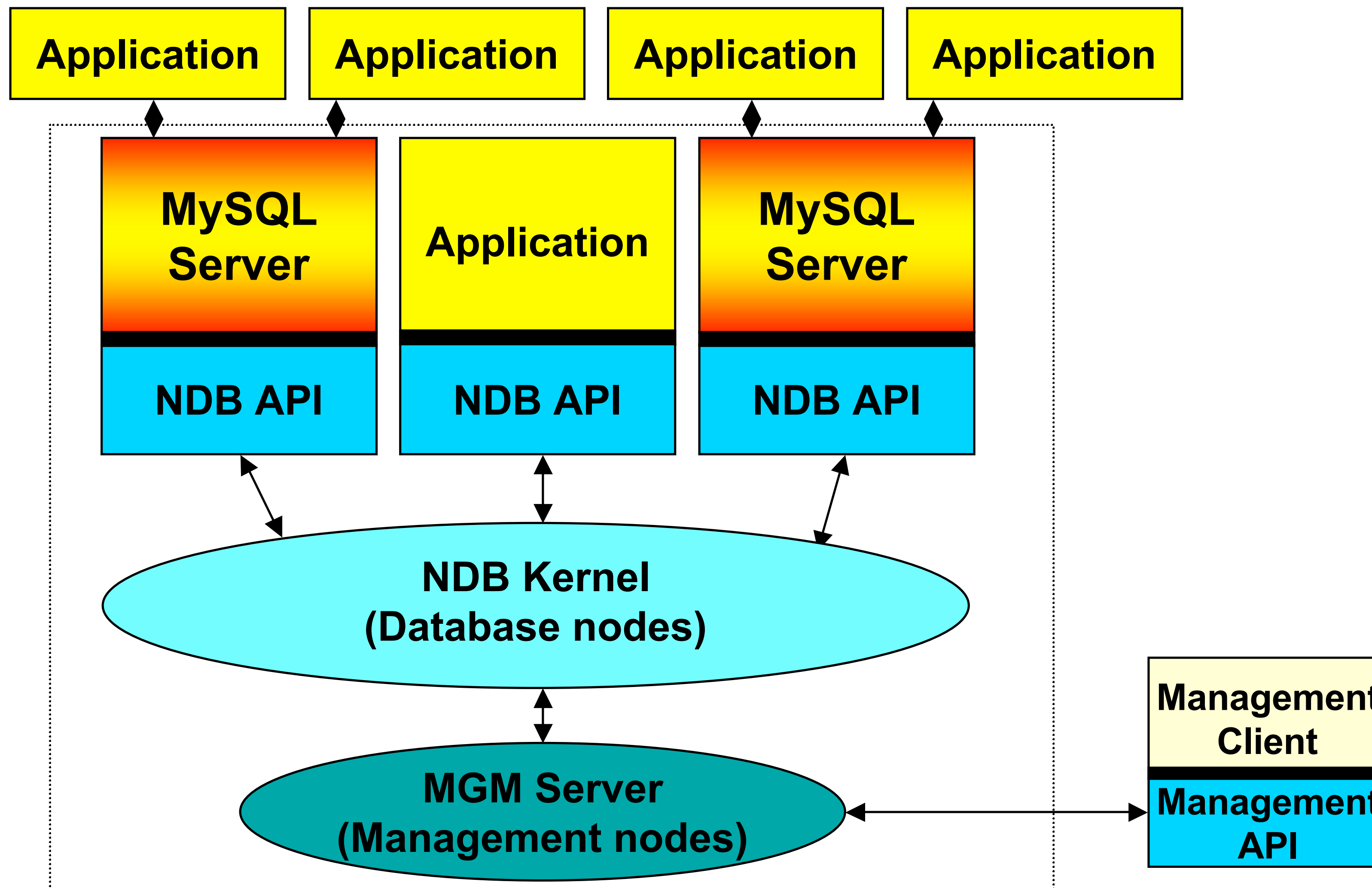
MySQL AB

Outline of Talk

- Introduction of MySQL Cluster in version 4.1 and 5.0
- Discussion of requirements for MySQL Cluster version 5.1
- Adaption of System Recovery algorithms
- Adaption of Node Recovery algorithms

Copyright 2005 MySQL AB

MySQL Cluster Architecture



Overview of MySQL Cluster in version 4.1 and 5.0

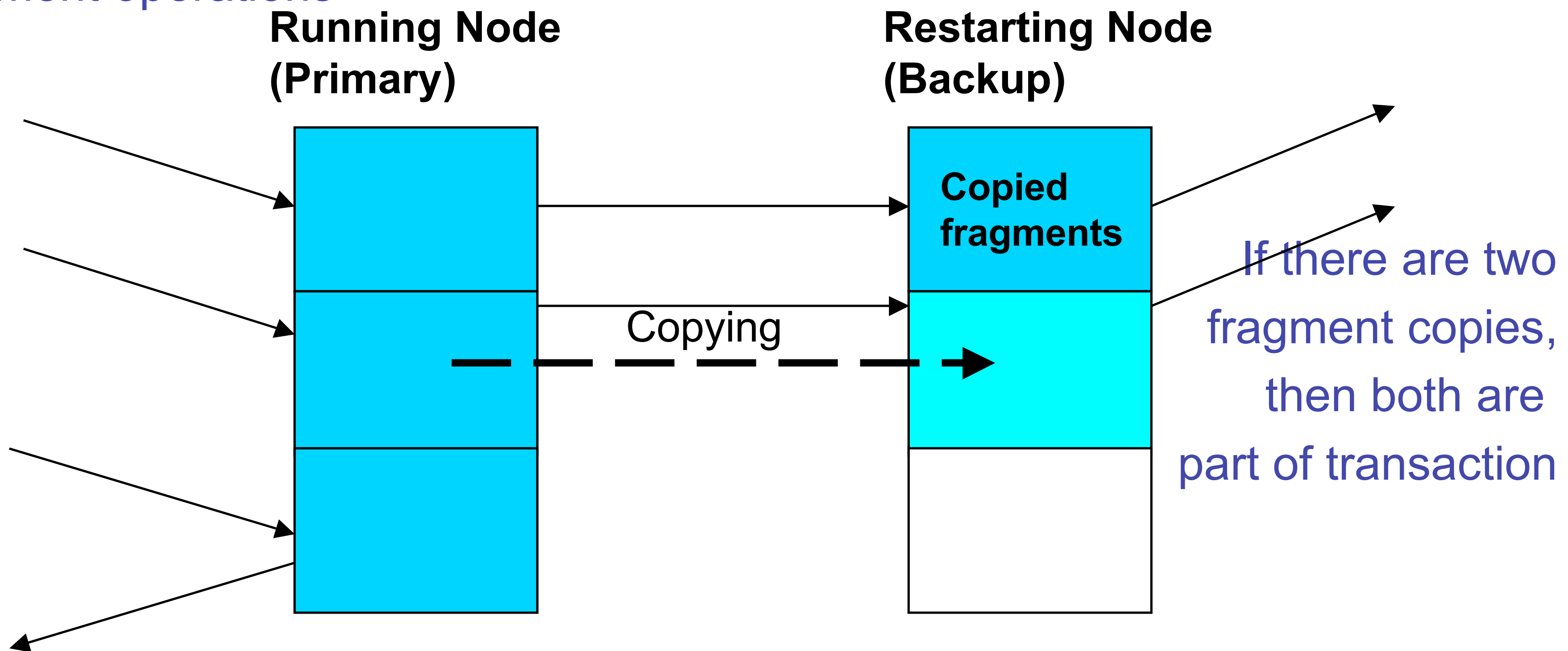
- **Main memory**
  - Complemented with logging to disk
- **Clustered**
  - Database distributed over many nodes
  - Synchronous replication between nodes
  - Database automatically partitioned over the nodes
  - Fail-over capability in case of node failure
- **Update in one MySQL server, immediately see result in another**
- **Support for high update loads (as well as very high read loads)**

Copyright 2005 MySQL AB

Node Recovery

Transaction Coordinator coordinates fragment operations

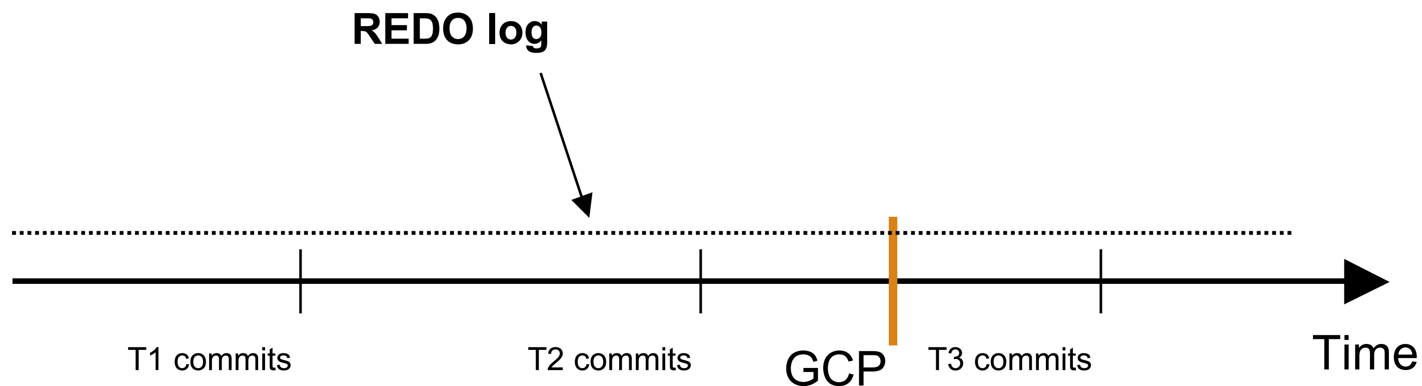
Example with two replicas



If there are two fragment copies, then both are part of transaction

Copyright 2005 MySQL AB

System Recovery: Flushing the log to disk



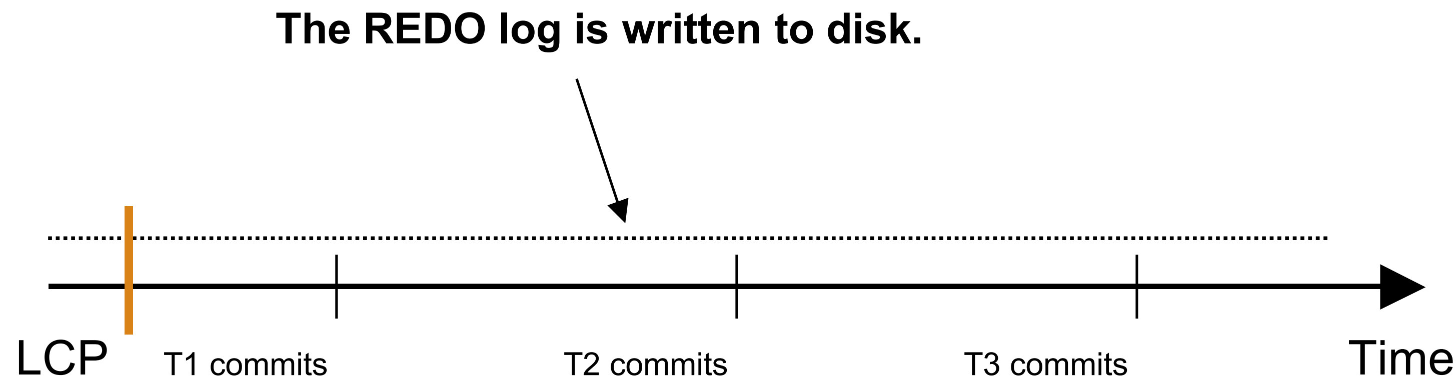
A *global checkpoint* GCP flushes REDO log to disk.

Transactions T1 and T2 become *disk persistent*.

(Also called *group commit*.)

Copyright 2005 MySQL AB

System Recovery: Log and Local checkpoints



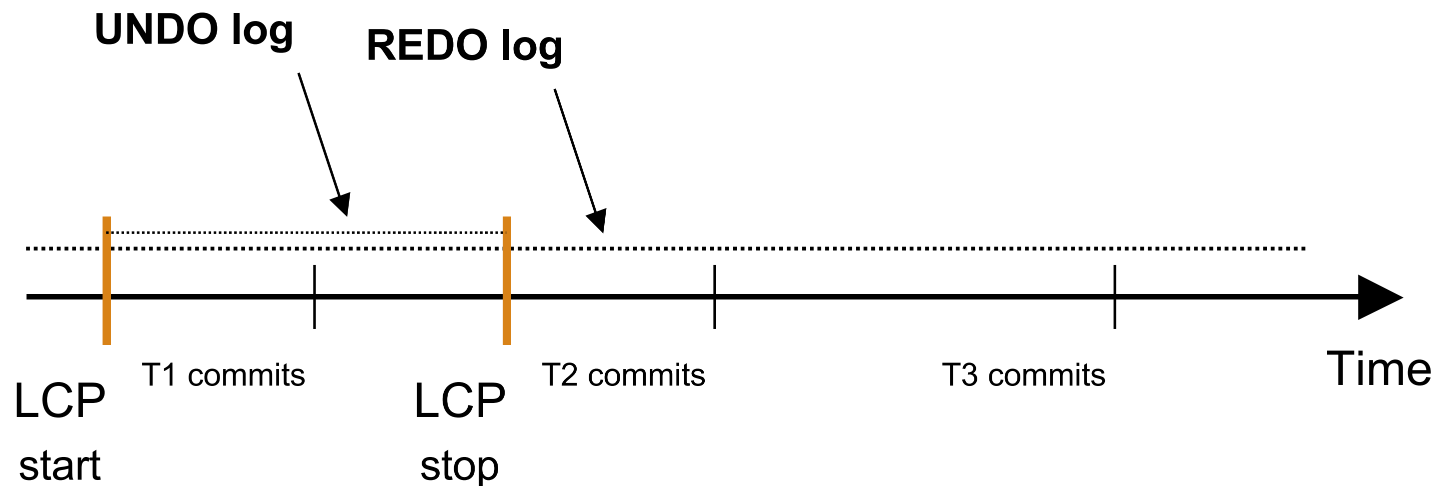
A *local checkpoint* LCP saves an image of the database on disk.
This enables cutting the tail of the REDO log.

Since MySQL Cluster replicate all data, the REDO log and LCPs are only used to recover from whole system failures.

Transactions T1, T2 and T3 are *main memory persistent in all replicas*.

Copyright 2005 MySQL AB

System Recovery: Checkpointing takes time

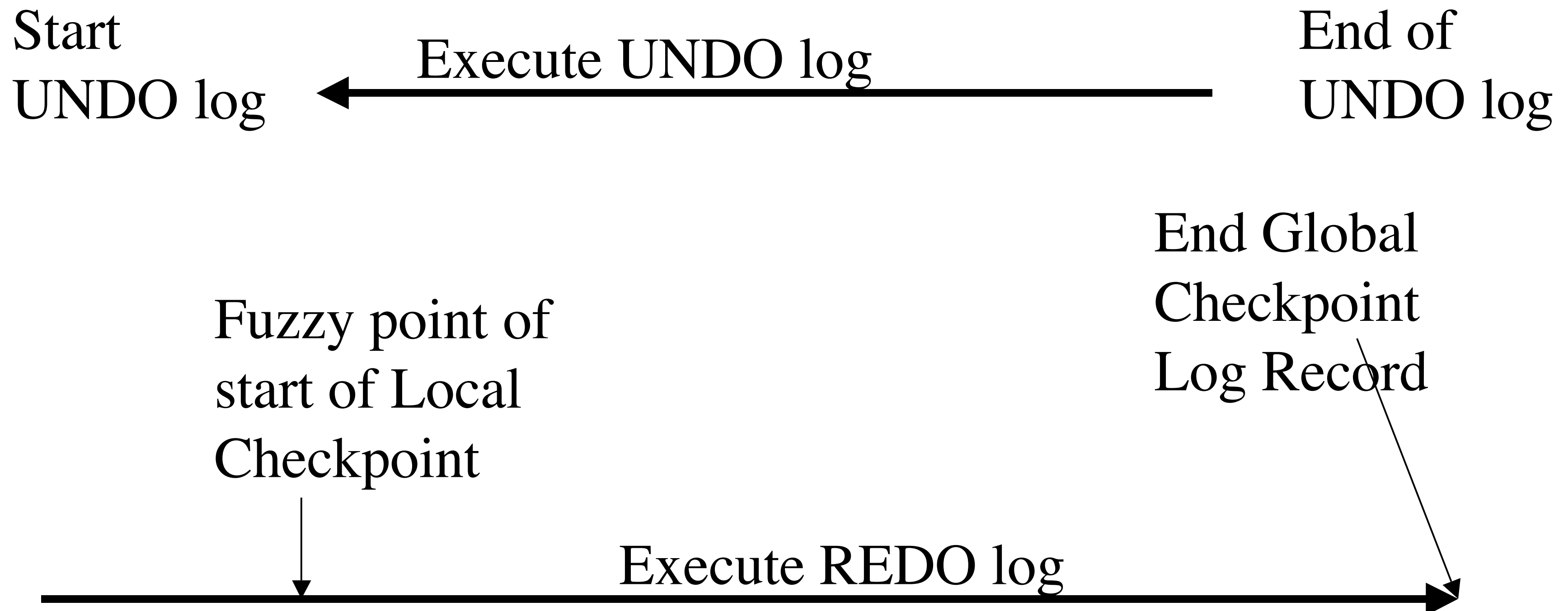


UNDO log makes LCP image consistent with database at LCP start time. This is needed since REDO log records are exactly the operations performed and thus the LCP must be action-consistent.

Copyright 2005 MySQL AB

Restoring a Fragment at System Restart

1) Load Data Pages from local checkpoint into memory



Copyright 2005 MySQL AB

New Requirements for MySQL Cluster 5.1

- Disk data to handle larger data volumes
- Variable sized records
- User defined partitioning
- Replication between clusters

Copyright 2005 MySQL AB

Discussion of Disk data requirement (1)

- Two major options
  - Put in existing disk-based engine into NDB kernel
  - Careful reengineering of NDB kernel to add support for disk-based

Copyright 2005 MySQL AB

Discussion of Disk data requirement (2)

- We opted for a careful reengineering of the NDB kernel to handle disk-based data
- The aim to keep its very good performance although data resides on disk
- Implementation divided in two phases, first phase puts non-indexed data on disk
- Second phase implement disk-based indexes

Discussion of Disk data requirement (3)

- MySQL Cluster version 5.1 implements the first phase, non-indexed fields on disk
- The approach of careful reengineering also enabled and in some cases forced us to improve on parts of the recovery architecture
- This paper presents some of those improvements

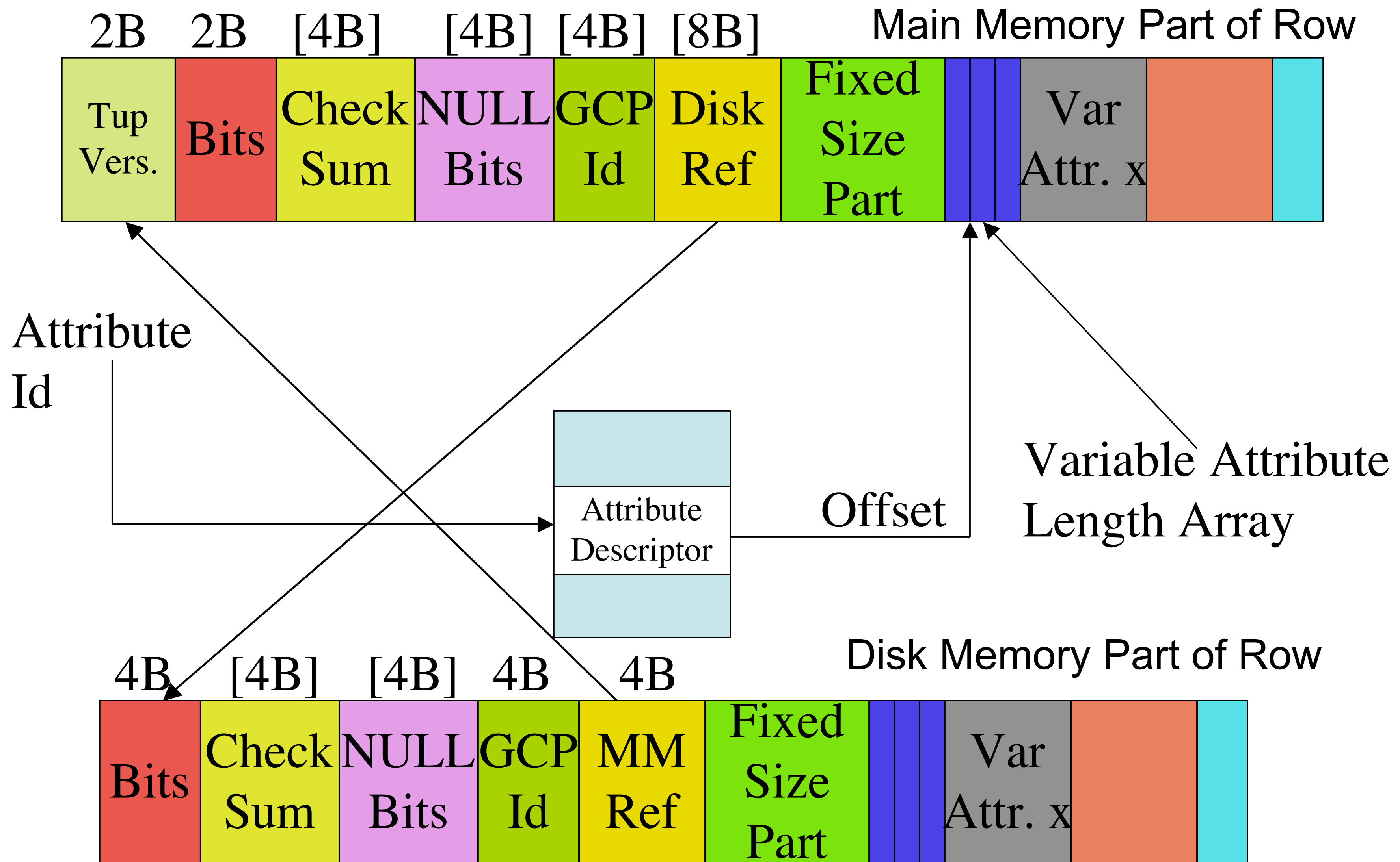
Copyright 2005 MySQL AB

Availability of MySQL Cluster 5.1

- The 5.1 version of MySQL has just been made available for source download
- As of today the version includes support for user defined partitioning
- As of soon the version will include support for disk-based data
- The cluster replication will soon be available in this downloadable tree

Copyright 2005 MySQL AB

Structure of Disk-based Row



Copyright 2005 MySQL AB

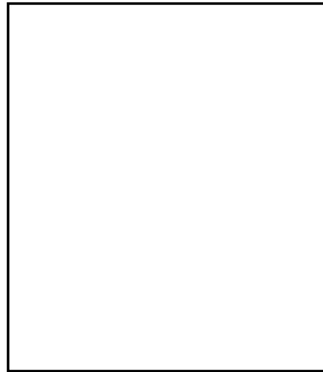
Node Recovery

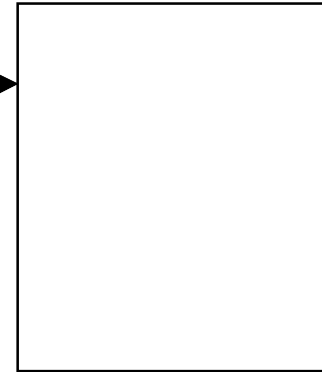
- Node Restart clearly affected by needing to copy entire data if data volume goes up by a factor of 10-100.
- 10-20 minutes of node restart with a few Gbytes of data is acceptable since no downtime happens but 10 hours of node restart with a few hundred Gbytes of data is not an acceptable period.
- Thus ”forced” reengineering

Node Restart, Replay Log Variant

Running Node

Starting Node





2) Replay Log from
First GCP not restored
(e.g. 17 in this case) to
Current GCP

1) Restart node from
Local logs (e.g. GCP 16)

3) Complex Hand-over (most likely involving
some exclusive locks when log replay completed

Copyright 2005 MySQL AB

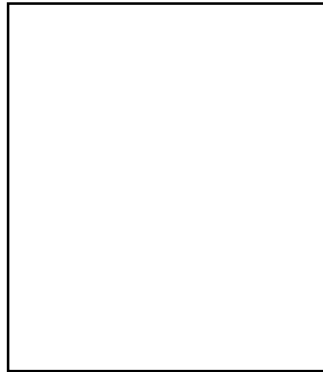
Pros and Cons with Replay Log Variant

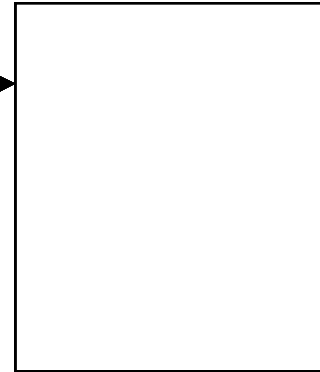
- Cons 1: Need to copy all rows if not enough log records in the running node
- Cons 2: Need to read the REDO log while writing it
- Cons 3: Difficult hand-over problem when end-of-log is coming closer to complete the Copy Phase
- Pros 1: Minimum effort when REDO logs exist

Node Restart, Delete Log Variant

Running Node

Starting Node





3) Replay Delete Log from First GCP not restored (e.g. 17 in this case) to GCP of 2)

4) Scan Running Node for records with higher GCP then GCP restored (e.g. 16 here), synchronise with starting node those rows

1) Restart node from Local logs (e.g. GCP 16)

2) Start participating in transactions

Copyright 2005 MySQL AB

Pros and Cons with Delete Log Variant

- Cons 1: Need to copy all data if not enough delete log records in the running node
- Cons 2: Need to read the Delete log while writing it
- Cons 3: Need to scan all rows in running node
- Cons 4: Need TIMESTAMP on rows
- Pros 1: No hand-over problem
- Pros 2: Delete Log much smaller than REDO log

Copyright 2005 MySQL AB

iSCSI Variant Intro

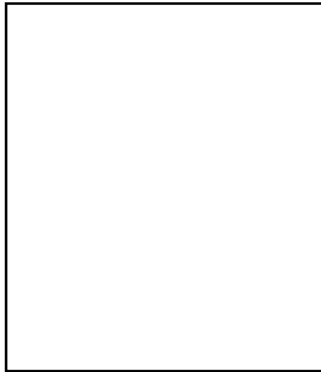
- View all data as ROWID entries
- ROWID’s only added or deleted in chunks of pages => uncommon operation
- Thus an INSERT/DELETE is transformed into an UPDATE of the ROWID entry
- Thus synchronisation can be done by a side-by-side check if the old and the new data is the same
- Efficient manner of performing check is by having TIMESTAMP on all ROWs (in our case TIMESTAMP = GCP id)

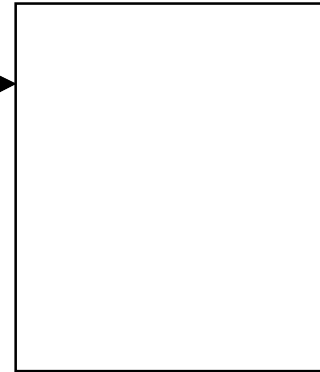
Copyright 2005 MySQL AB

Node Restart, iSCSI Variant

Running Node

Starting Node





2) Specify page ranges existing
in partition to copy

5) Synchronise all ROWID's, send new data when changed since GCP node restored (e.g. 16 in this case)

1) Restart node from Local logs (e.g. GCP 16)

3) Delete all pages no longer present

4) Start participating in transactions

Pros and Cons with iSCSI Variant

- Cons 1: Need to scan all rows in running node
- Cons 2: Need TIMESTAMP on rows
- Cons 3: Requires ROWID as record identifier
- Pros 1: No hand-over problem
- Pros 2: Not limited by Log Sizes

Copyright 2005 MySQL AB

System Recovery

- Algorithms that write entire data set each during local checkpoint cannot work for cached disk data
- Thus new algorithm needed for the disk-based parts
- Previous algorithm had a coupling between abort mechanisms and recovery mechanisms
- Here we found an ”enabled” reengineering

Copyright 2005 MySQL AB

Buffer Manager Problem

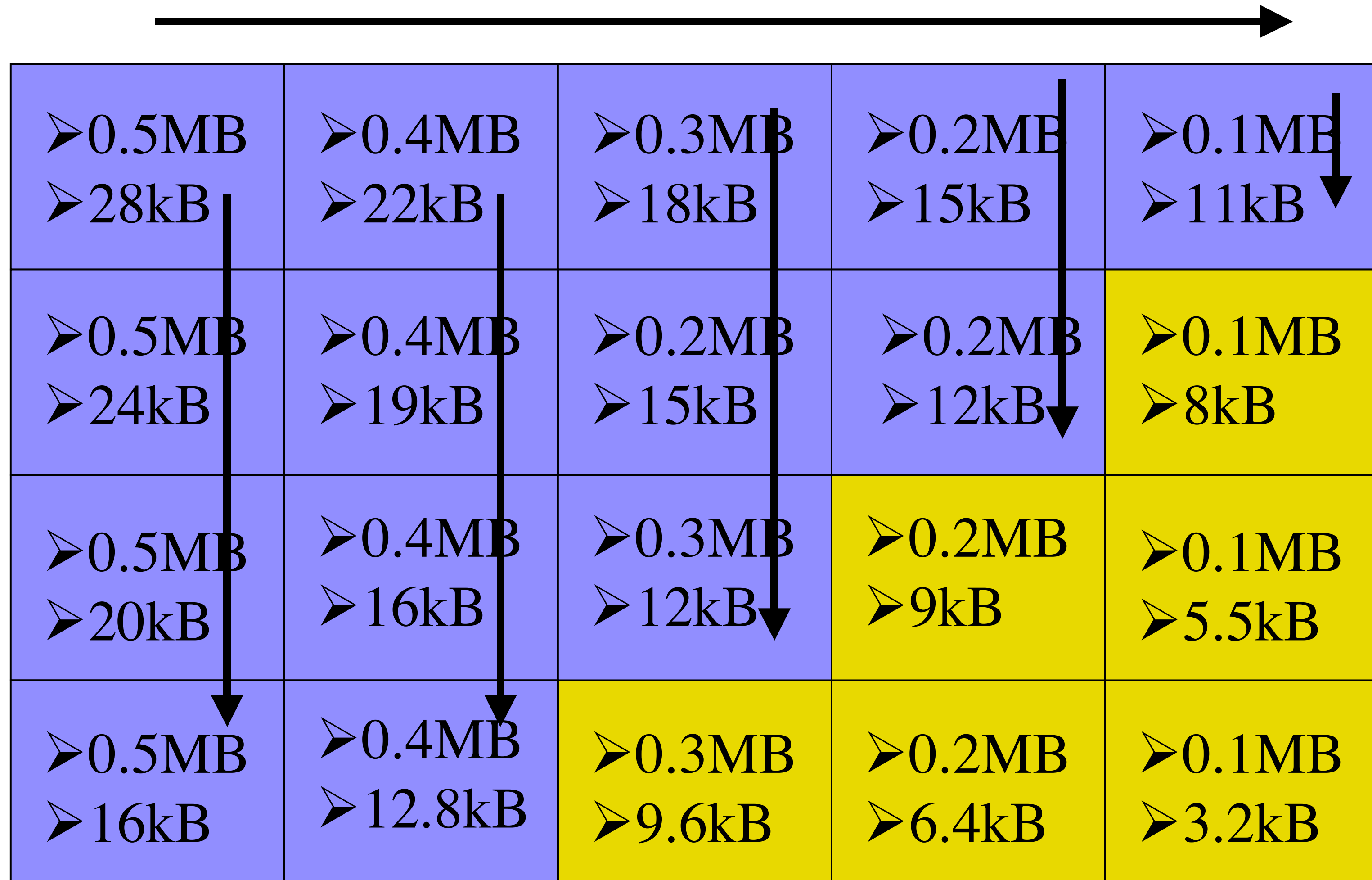
- Design Goal: Any update should at most generate two disk writes in the normal case to sustain very good performance for high update loads (side effect should be that we get almost comparable performance to file systems

Buffer Manager Solution (1)

- REDO log no changes
- Write Page using a Page Cache algorithm with Write Ahead Log (WAL) principle.
- We try to achieve multiple writes for inserts by choosing an extent with place for the record and as much free space as possible

Copyright 2005 MySQL AB

Search Extent to Write (Down First, Right Then) (Extent Size = 1 MB, 70% full Record Size = 10 kB



Copyright 2005 MySQL AB

Buffer Manager Solution (2)

- For disk-based data we need to UNDO log all structural changes (data movement, allocation, deallocation)
- Data UNDO logging is not needed if GCP was written since REDO log will replay it anyways
- Implementation through Filtering UNDO log records when preparing them for write to disk (avoids complex logic around page writes)

Copyright 2005 MySQL AB

Abort Handling Decoupled from Recovery

- Decided to use No-Steal algorithm (only committed pages are written to disk)
- => Simplifies Recovery
- => Requires Transaction State to be in main memory for optimum performance
- Large transactions could require that transactional information is spooled down to disk and back later
- As for MySQL Cluster 4.1 and 5.0 also 5.1 will impose a configurable limit to transaction sizes

Copyright 2005 MySQL AB

Buffer Manager Conclusions

- Good solution for minimising the number of disk writes for transactions that fit into main memory (thus can still be upto Gbytes in size)
- Growth of memory sizes makes design choices with easier architecture better (logical logging and No-Steal algorithm)

Copyright 2005 MySQL AB

New LCP Algorithm for Main Memory Parts that avoids UNDO logging entirely

- Scan Memory Parts and write each row part into checkpoint log
- Use one bit in row to synchronise write at start of write
- This bit doesn’t survive crashes and thus no problem maintaining it after a crash it is always initialised to 0
- Simple variant on Copy-On-Write

Copyright 2005 MySQL AB

Action by Insert/Update/Delete when scan ongoing and scan hasn't passed yet

- Commit Insert sets bit, no write to checkpoint
- Commit Update/Delete sets bit and writes to checkpoint log

Copyright 2005 MySQL AB

Action by scan

- If bit = 0 when scan reads it write it to the local checkpoint file
- If bit = 1 then set bit = 0 and don't write it to local checkpoint

Copyright 2005 MySQL AB

Index Recovery

- All indexes are main memory indexes and are rebuilt as part of system recovery and node recovery
- All index builds can be performed while updates are ongoing (feature of NDB kernel to be exported to SQL level in 5.1)

Copyright 2005 MySQL AB

Conclusions

- Careful reengineering of NDB kernel introducing non-indexed fields on disk for MySQL Cluster version 5.1
- Lots of improvements, both performance-wise and simplicity-wise of Recovery algorithms
- Welcome to play with it and check if there are areas needing improvement, code will soon be available

Copyright 2005 MySQL AB