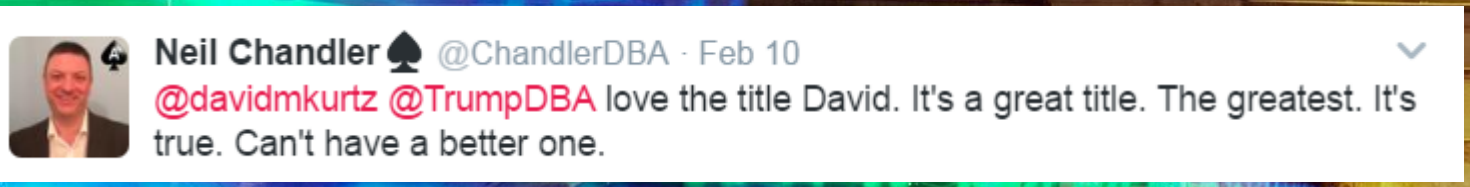
accenture

Accenture Enkitec Group

Making Materialized Views Great Again

High performance. Delivered.

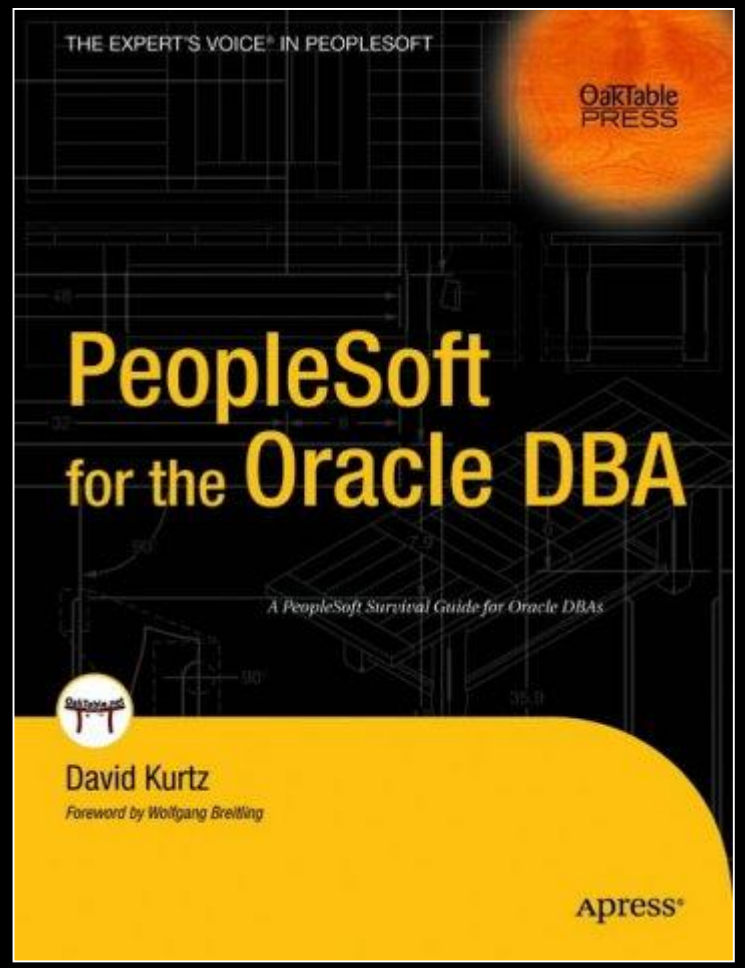
David Kurtz
david.kurtz@accenture.com
DOAG2018



Strategy | Consulting | Digital | Technology | Operations

Who Am I

- **Accenture Enkitec Group**
- Performance tuning
  - PeopleSoft ERP
  - Oracle RDBMS
- Book
  - www.go-faster.co.uk
  - blog.psftdba.com
- Oak Table
- ORACLE ACE Director



Caveat

- *Details have been withheld and changed to preserve client confidentiality and anonymity*

This is what the architect designed

The story so far…

Primary Database

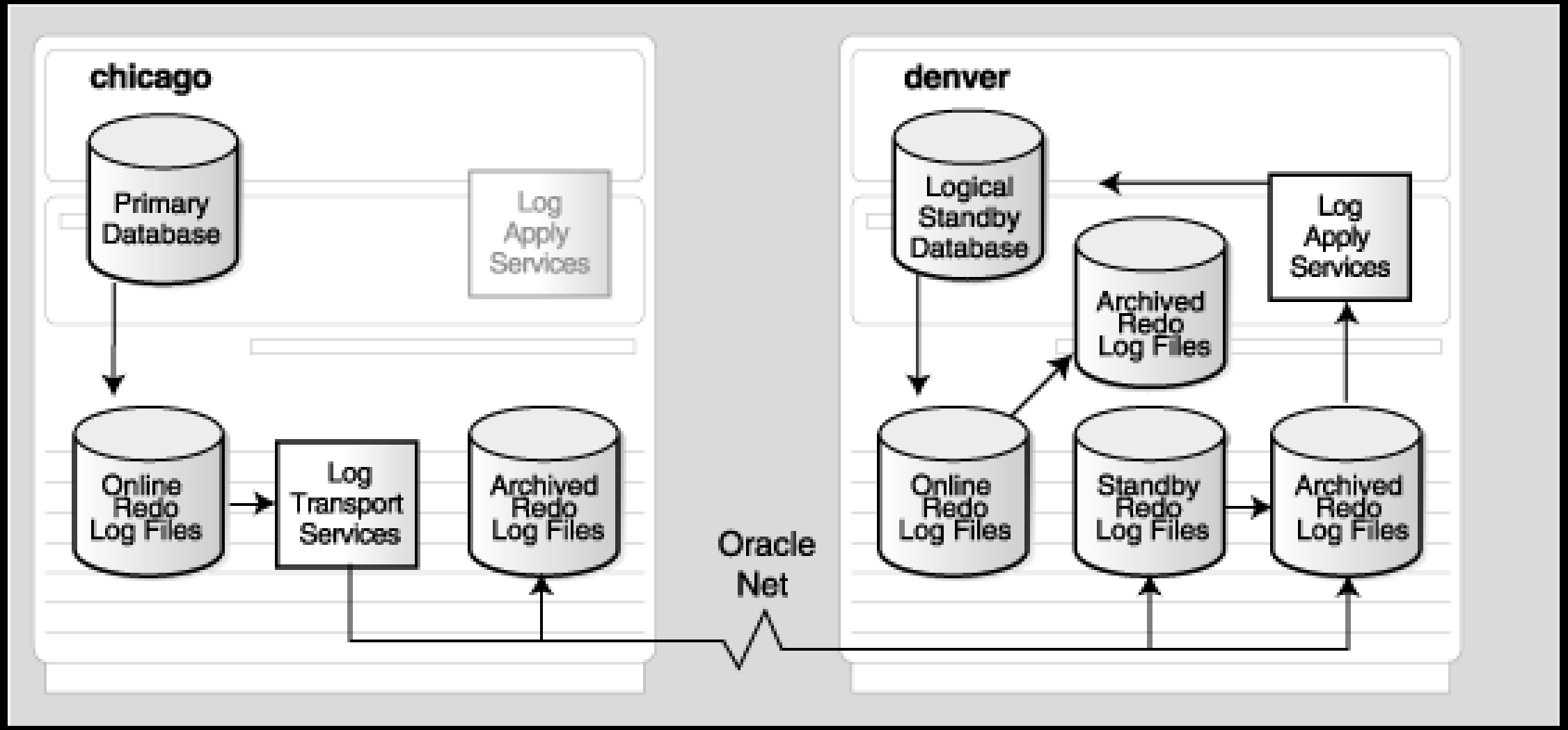
- OLTP application
- Real-time transactional feed
- Some reporting

Secondary Database

- Reporting database
  - Essentially a full copy of the primary
- Logical Standby
  - Replicate application data
- Complex Materialized Views for reporting

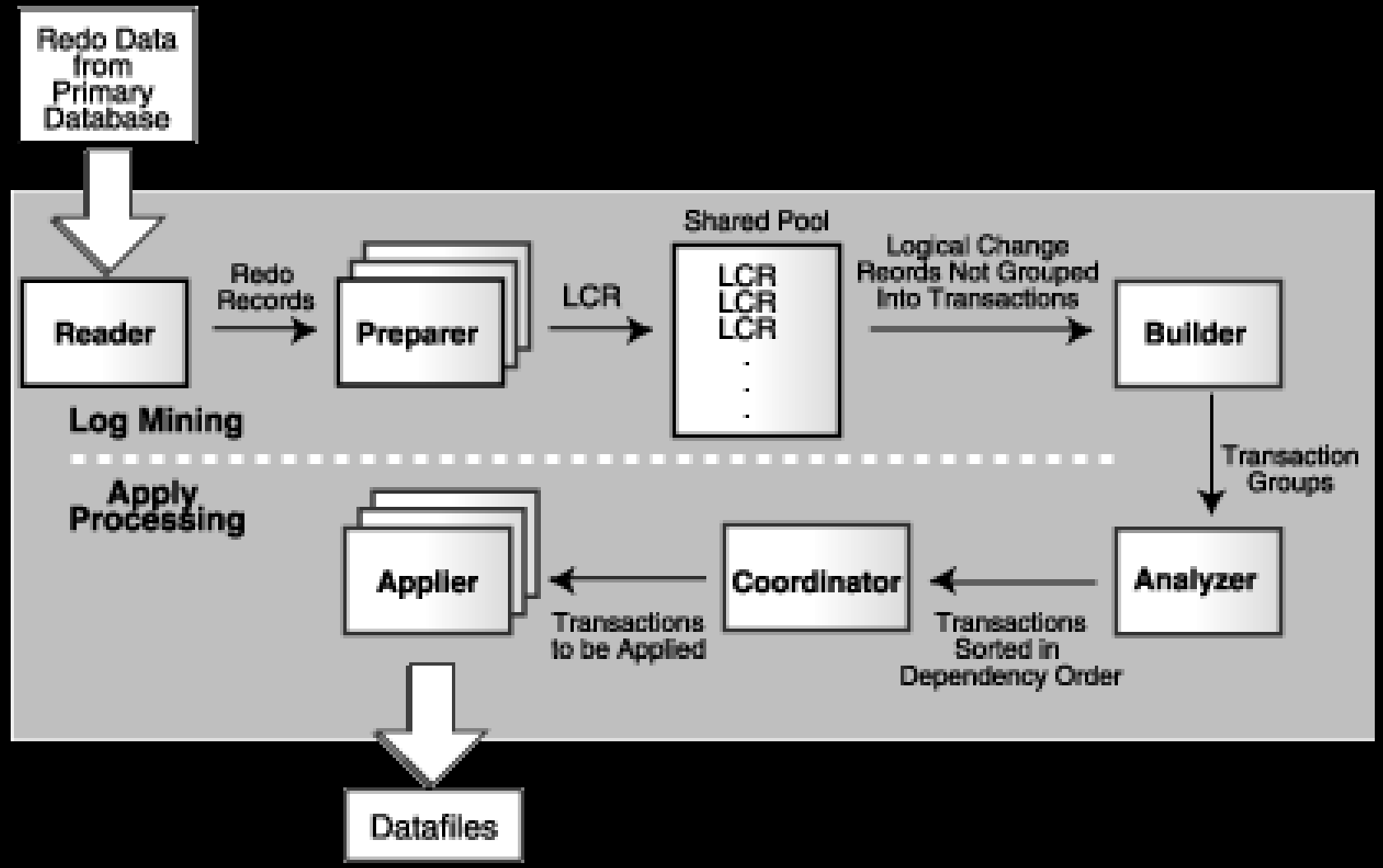
Logical Standby

- From Oracle 10g documentation

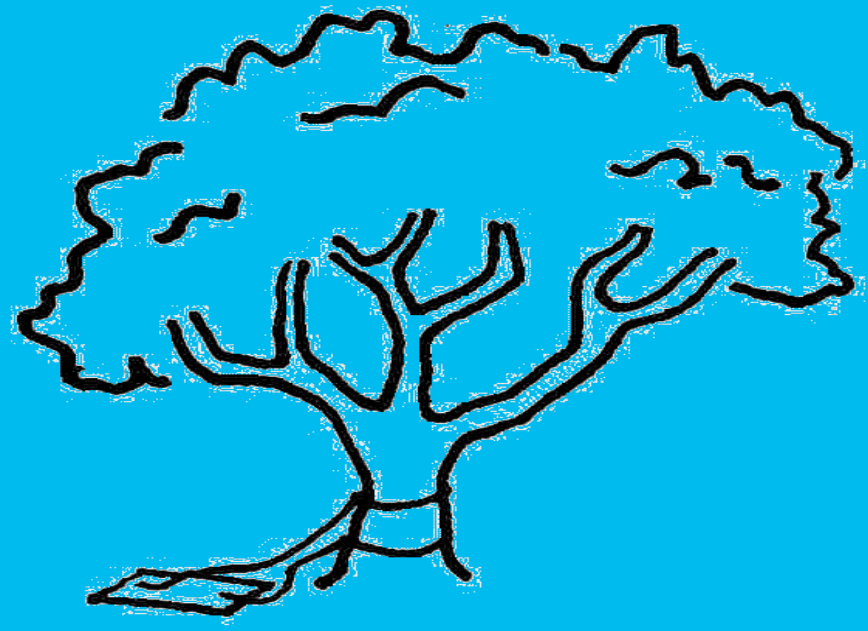


There is a lot going on!

- From Oracle documentation



This is what got built



When I walked in…

Primary Database

- OLTP application
  - row-by-row insert
- Real-time transactional feed
  - Large data volumes
    - Several tables >25M rows
    - 1 table > 1B rows
    - No partitioning
    - No archiving
    - Not licenced for advanced compression
- Some reporting
  - More when the reporting materialized views don’t refresh

Secondary Database

- Reporting database
  - All the same Indexes as primary
- Complex Materialized Views for reporting
  - Can only be fully refreshed
  - Some of which take a long time to refresh
- Logical Standby
  - Also row-by-row
  - It can’t keep up

The Problem With Redo

Primary Database

- 1 time

```
UPDATE table
SET column = value
WHERE column = value
```

  - 1000 rows updated.
  - 1000 redo records

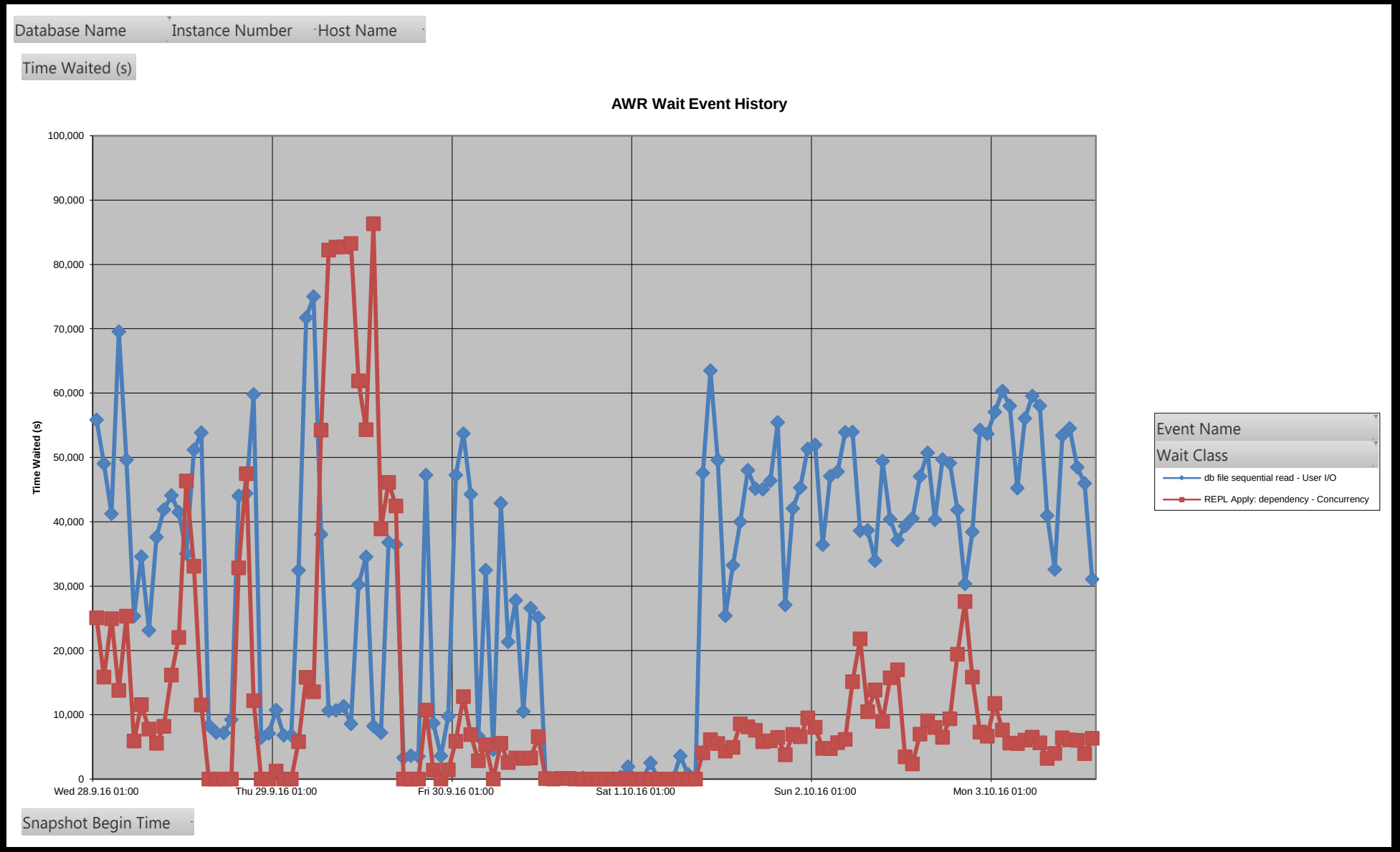
Secondary Database

- 1000 times

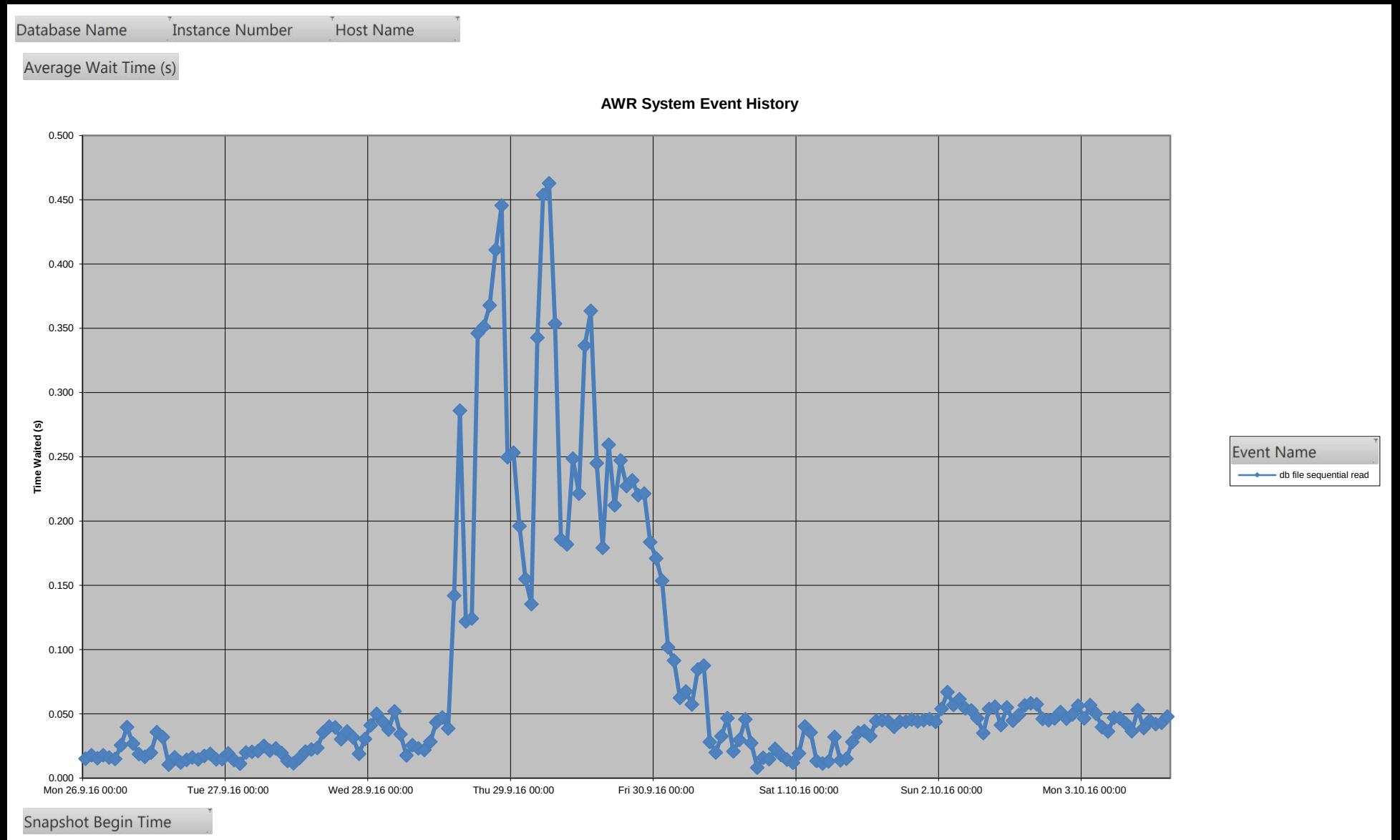
```
UPDATE table
SET column =value
WHERE pk_column = pk_value
```

  - 1 row updated

db file sequential read
REPL Apply: dependency - Concurrency



Single block read from the disk could be very slow – ~~it has improved since~~



ASH Analysis

Time spent in Logical Standby Apply Processing

MODULE	ACTION	EVENT	ASH_SECS
Streams	- Apply Server	db file sequential read	5313250
Streams	- Apply Server	REPL Apply: dependency	2283370
		db file parallel write	1142700
Logical Standby	Logminer Reader	LogMiner reader: buffer	583410
Oracle		Data Guard server operation completion	496090
JDBC Thin Client		direct path read	459310
Logical Standby	Logminer Preparer	LogMiner preparer: memory	399060
Streams	- Apply Reader	CPU+CPU Wait	326510
Logical Standby	Logminer Builder	LogMiner builder: DDL	324560
		log file parallel write	292630
Streams	- Apply Server	CPU+CPU Wait	248310
Streams	- Apply Server	enq: TX - index contention	198890
Sqlplus		db file sequential read	194040
JDBC Thin Client		db file sequential read	187240
Logical Standby	Logminer Preparer	latch free	168160
Logical Standby	Logminer Builder	LogMiner builder: memory	132110
		log file sequential read	123250
Logical Standby	Logminer Preparer	CPU+CPU Wait	117590
Sqlplus		enq: JI - contention	112690

…

Top Wait Events

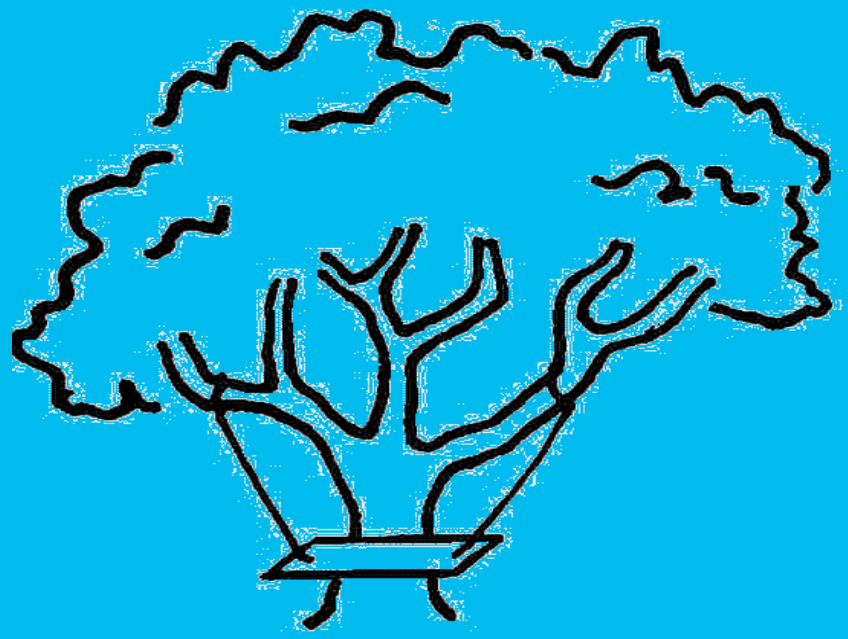
db file sequential read

- Single block read
  - Moslty associated with index maintenance
    - *v$active_session_history.current_obj#*
- Logical standby is inserting rows
  - Indexes maintenance is synchronous
  - Rows have to be inserted in the correct place
  - So you look up the new value in the index to find the correct leaf block.

REPL Apply: dependency

- Many different sessions created by apply processing
- Transactions must be applied and committed in order
- Lots of contention in the coordination of transactions.
  - Blocking sessions often waiting on sessions waiting on sequential read

Some initial tuning



Logical Standby Could not keep up

- Poor response of disk subsystem 15-25ms.
  - Common SAN
  - Either need to do fewer IOs
  - Or need to faster IOs
  - Or Ideally both!
- Logical standby would occasionally hang
  - A transaction could not be applied

- Disk connectivity?
- Virtualisation problems
- More buffer cache
- Fewer indexes

Reporting Materialized Views took a long time to refresh

Reporting Materialized Views on Standby

- Long running query
  - Execution plan
    - Full scan on underlying table
  - ASH Data
    - SQL_PLAN_LINE_ID
    - For that line in the plan
      - db file sequential read
      - on the undo segment
    - Full scan is normally a scattered read
- Read consistency
  - Oracle is reading undo information so it can generate a read consistent copy of the data block in buffer cache.

Options

1. Stop Logical Standby during refresh of report materialized views
   - Certainly improved refresh performance.
   - Not popular. It already couldn't keep up, stopping refresh not acceptable.
2. Replace Logical Standby with simple materialized views

```
CREATE MATERIALIZED VIEW apptable
WITH PRIMARY KEY ON PREBUILT TABLE
AS SELECT * FROM apptable@primary;
```

   - Materialized view logs on the primary key
   - Regular incremental refresh on the secondary.
   - Multiple concurrent refresh jobs for different tables.
   - Suspend refresh during refresh of reporting MVs with locking mechanism to avoid consistent read.

The fastest way to do anything is not to do it at all

- Index maintenance on insert/update statements.
  - Disable enforcement of referential integrity constraints from standby
    - RI enforced on primary database
    - Use MV Refresh Groups so MVs maintained in same database transaction
    - Didn't need indexes on foreign keys
  - Remove Redundant Indexes from Primary & Secondary*
  - Remove unnecessary/unused non-unique indexes
    - Use ASH to identify index usage, and therefore candidate unused indexes
    - Make indexes invisible for a while before dropping
  - Extended Statistics
    - Replace multi-column indexes with extended statistics (before making indexes invisible)*

Redundant Indexes

Superset Index

```
CREATE INDEX i3 ON t (a, b, c)
```

- Queries on A,B that currently used index i2 can use index i3

Subset Index

```
CREATE INDEX i2 ON t (a, b)
```

- Optimizer can get number of distinct values of combination of A and B from statistics on index i2.
- Index not referenced in the execution plan.
- Drop index i2, but replace with extended statistics

```
SELECT
dbms_stats.create_extended_stats(NULL, 'T',
'(a,b)') FROM dual;

dbms_stats.set_table_prefs(NULL, 'T',
method_opt=>'FOR ALL COLUMNS SIZE AUTO FOR
COLUMNS SIZE 1 (a,b)');

dbms_stats.gather_table_stats(null, 'T',
method_opt=>'FOR ALL COLUMNS SIZE AUTO FOR
COLUMNS SIZE 1 (a,b)');
```

Redundant Unique Indexes

Unique Subset Index

```
CREATE TABLE t
(a NUMBER
,b NUMBER
,CONSTRAINT t_pk PRIMARY KEY (a)
);
```

- Unique index on A to police the primary key constraint

Superset Index

```
CREATE INDEX t_ab ON t (a,b);
```

- I could use index on A B for that instead

```
ALTER TABLE t DROP CONSTRAINT t_pk;

ALTER TABLE t
ADD CONSTRAINT t_pk PRIMARY KEY (a)
USING INDEX t_ab;
```

Read Consistency

- Long running reporting queries (especially in reporting MVs)
  - Full scan operations in execution plan
  - Single block read on undo segment
  - Consistent read. Undo information required to rollback read consistent buffer block back to SCN of query.
- Logical standby updating underlying tables during MV refresh
  - Can't pause Logical Standby so can’t control when an object is updated
- Production volumes demanded multiple concurrent MV refresh jobs
  - Similar issue with simple MV refreshing while reporting MV refreshed
  - But now I can take control…

DBMS_LOCK used by our SYNC_MV package

Simple Materialized View

```
CREATE MATERIALIZED VIEW t1
WITH PRIMARY KEY ON PREBUILT TABLE
AS
SELECT * FROM t1@primary;
```

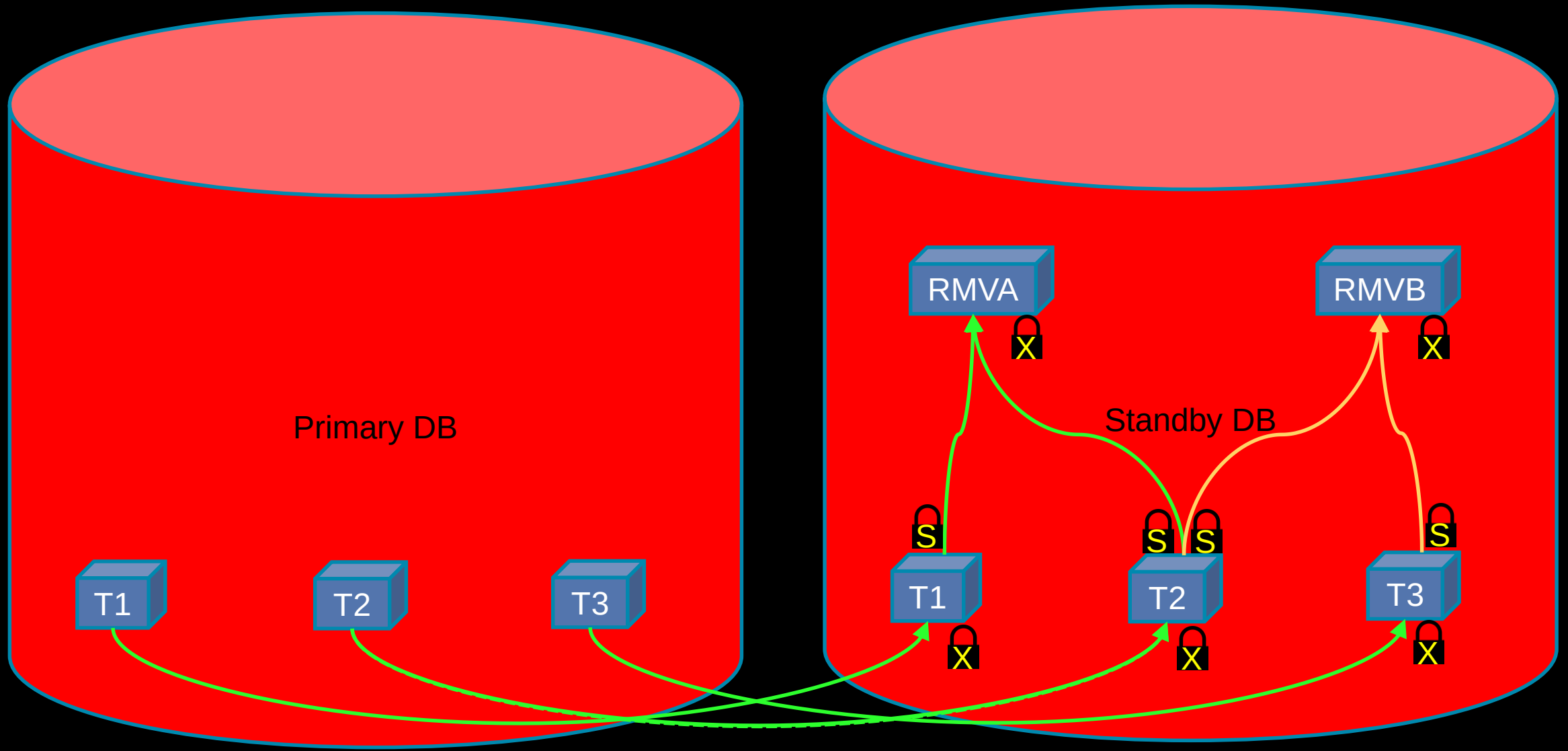
- Incremental Refresh
  - Materialized View Log on primary database
  - Join to table on primary database
- Exclusive Lock
  - Lock name: SYNC_MV_T1
  - Lock is not released by COMMIT
    - (but could chose to allow commit at end of atomic refresh to release lock)

Complex Materialized View

```
CREATE MATERIALIZED VIEW rmv1
AS SELECT …
FROM t1, t2
…;
```

- Non-Atomic Complete
  - Locks must be held through commit
- Exclusive Lock
  - Lock name: SYNC_MV_RMV1
- Shared Locks
  - Lock names:
    - SYNC_MV_T1
    - SYNC_MV_T2

Controlling MV refresh with DBMS_LOCK



Refreshing Complex Materialized Views

Atomic_refresh=>TRUE

```
DELETE FROM … ;
INSERT … SELECT … ;
COMMIT;
```

- Benefits
  - Data never disappears from materialized view
- Drawbacks
  - Space freed by delete not available until after commit
  - Doesn't reset high water marks
  - Indexes maintained row-by-row
  - Queries executed on MV during refresh perform consistent read
    - Requiring information from undo segment
    - Slower

Atomic_refresh=>FALSE

```
TRUNCATE …;
INSERT /*+APPEND*/ SELECT;
COMMIT;
```

- Benefits
  - Truncate is DDL
    - Resets high water marks
    - Frees space
    - Faster than delete
  - Direct path insert enables simple compression
    - (or HCC on Engineered System)
  - Table stats automatically maintained in 12c
  - Indexes maintained after insert
- Drawbacks
  - Truncate is DDL
  - Materialized View empty during refresh

2 Materialized View Workaround for Atomic Refresh

Ingredients

```
CREATE MATERIALIZED VIEW rmva1
AS SELECT … FROM t1, t2 …;

CREATE MATERIALIZED VIEW rmva2
AS SELECT … FROM t1, t2 …;

CREATE OR REPLACE VIEW rmva
AS SELECT * FROM rmva1;
```

Method

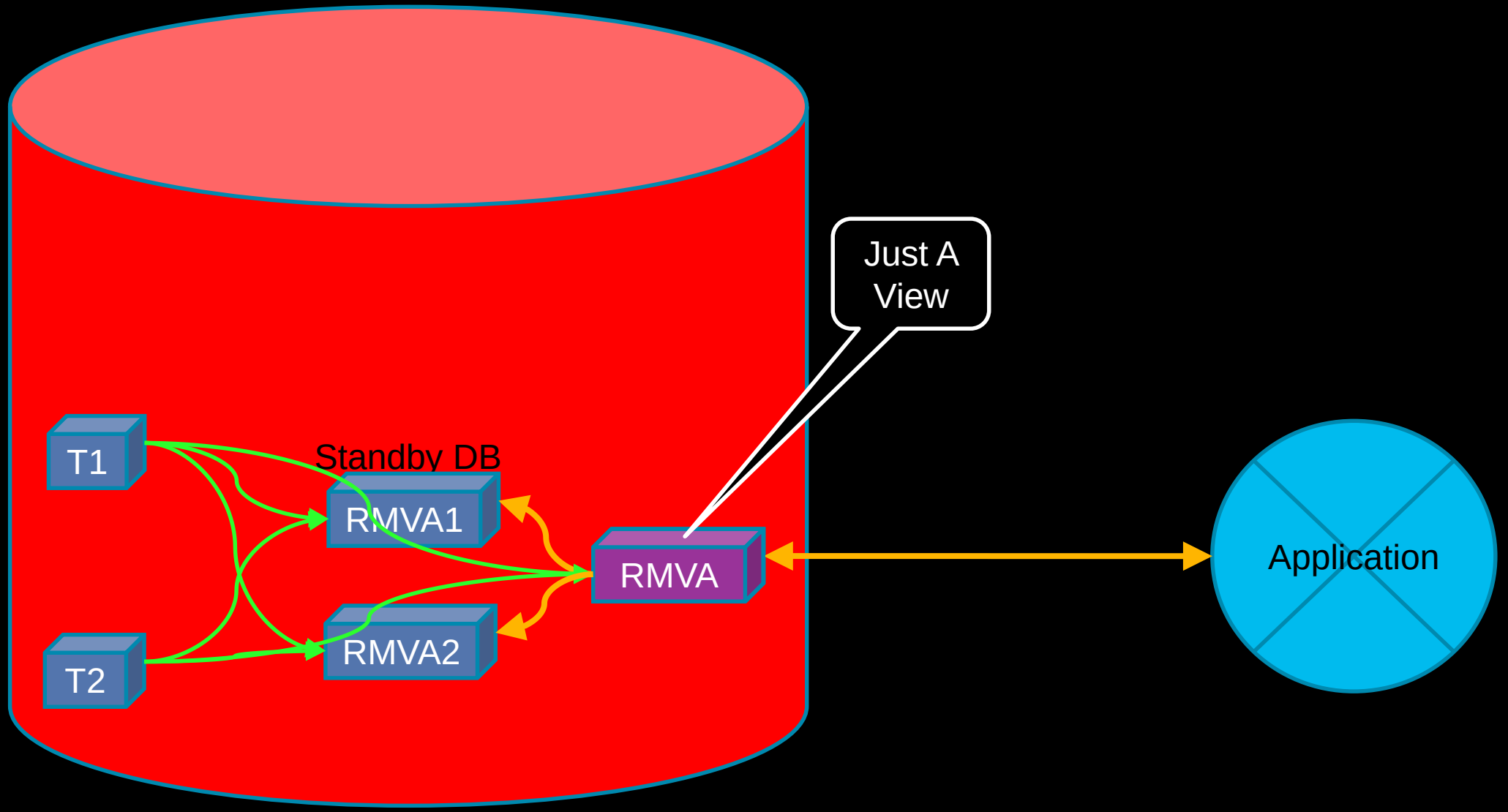
- 1. Refresh whichever materialized view has older refresh date

  ```
  user_mviews.last_refresh_date
  ```

- 2. Recreate view to reference materialized view just refreshed

  ```
  CREATE OR REPLACE VIEW rmva
  AS SELECT * FROM rmva2;
  ```

Non-Atomic Refresh with 2 MVs



2 Materialized View Workaround for Atomic Refresh

- 2 copies of the data
  - But takes up no more room than single atomically refreshed MV
    - Space freed by delete not available until commit
  - Can be much smaller due to compression
    - No licence needed for simple compression or HCC on Engineered Systems
- No read consistency problem
  - Application uses one MV while other is refreshed
  - Running queries not affected by recreating of view
- Need to refresh with custom package rather than just DBMS_MVIEW
- See http://blog.go-faster.co.uk/2016/07/complete-refresh-of-materialized-views.html

Out-of-Place Refresh in 12c

New parameter in 12c

```
exec DBMS_MVIEW.REFRESH('MY_EXAMPLE1','C',
atomic_refresh=>TRUE, out_of_place=>TRUE);
```

- Oracle builds and populates a new table
  - temporarily called rv$xxxx (where xxxx is hex object ID)
  - New table is renamed and old table is dropped
    - Will go to recyclebin if enabled, so need to manage purging old versions
- Long running queries not directly affected
  - Different object, no consistent read
- Atomic Refresh only

Catches

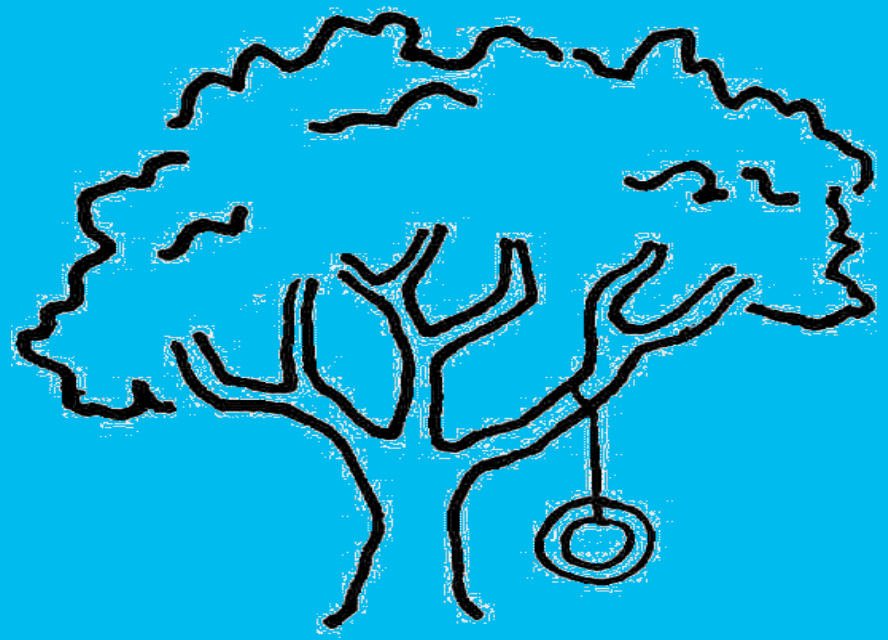
- Long running queries could fail when refresh completes with
  - *ORA-08103: object no longer exists*
    - *(if no recyclebin)*
- Insert not done in direct path mode
  - Indexes created after insert – not a problem
  - Cannot compress without Advanced Compression licence
  - Table statistics are not automatically created
    - But index statistics are
- http://blog.go-faster.co.uk/2016/07/complete-refresh-of-materialized-views.html
- https://jonathanlewis.wordpress.com/2015/03/26/12c-mview-refresh/

Current Status

Problems

- Still have residual disk performance problem
  - Average ~8ms, peak >20ms
- Still have logical standby
  - Major tables now omitted from Logical Standby and using MVs
- MV refresh invoked by Cron
  - Can only issue DDL on a logical standby AS SYSDBA
  - run from Unix account that is member of DBA group
  - Scripts connect / AS SYSDBA
  - Audit records / logs

Next Steps



Outstanding

- Disk subsystem
- Remove Logical Standby
  - Standby Database becomes independent reporting database
  - Replace Unix cron jobs with database scheduler jobs.
- Application Changes
  - Aggregation/Archive/Purge Policy
  - Design some sort of programmatic incremental refresh of reporting tables.

There is a time and a place for each

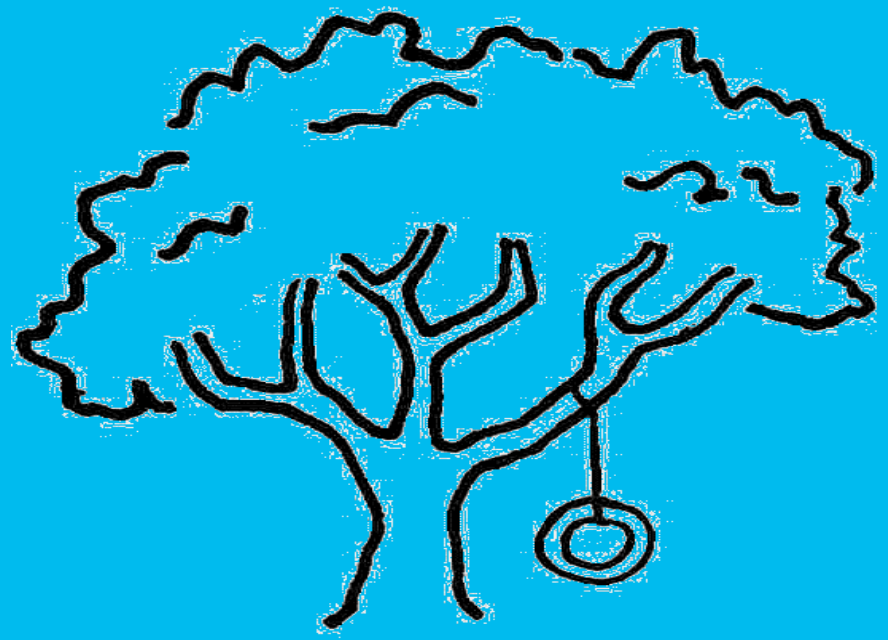
Logical Standby

- Row-by-row

DBMS_MVIEW

- Bulk SQL set processing

Questions?



ORA-0000 The morning is over

There have been no exceptions

It's time for ~~beer~~ lunch