Orphans, Corruption, Careful Write, and Logging, or Gfix says my database is ***CORRUPT*** or Database Integrity - then, now, future

Ann W. Harrison
James A. Starkey

A Word of Thanks to our Sponsors

MOSCOW EXCHANGE *Platinum*

SITA SOFTWARE *Platinum*

IBPhoenix *Platinum*

IBSurgeon *Platinum*

REDSOFT

IBObjects

CopyCat

Upscene

And to Vlad Khorsun

Core 4562

Some errors reported by database validation (such as orphan pages and a few others) are not critical for database, i.e. don’t affect query results and\or logical consistency of user data.
Such defects should not be counted as errors to not scare users.

Fixed 28 Sept 2014

Questions?

MVCC – Quick Review

- Read consistency, undo, and update concurrency provided in one durable mechanism.
- Data is never overwritten.
- Update or Delete creates new record version linked to old.
- Transaction reads the version committed when it started (or at the instant for Read Committed)
- Each record chain has at most one uncommitted version.
- Rollback removes uncommitted version.
- .

What does Gfix do?

Reads entire database verifying internal consistency:

Of interest now:

- Allocated pages are in use
- Unused pages are not allocated
- Primary record links to
  - Fragments
  - Back versions

Before 28 September 2014, any problem was an error

What are Orphans?

And what do they have to do with this? (not what you think)

Database Integrity

- Disasters occur (more often circa 1985)
  - Database System, O/S, Network, Power, Disk
- Classic Solutions
  - Write Ahead Log
  - Shadow Pages
  - After image Log
- Firebird Solution
  - Careful write, multi-version records
  - Write once

Disk Failure

- InterBase V1
  - Journal
    - After image
    - Abandoned by Borland
  - Shadow
    - Complete copy on separate disk
    - Better done in RAID

Careful Write

Order writes to disk (fsync)

Database is **always** consistent on disk

- Rule: write the object pointed to then the pointer
- Record examples: record before index, back version before main, fragment before main record
- Page examples: mark as allocated before using, release before marking free

Requires disciplined development

Record Before Index

- Indexes are always considered "noisy"
  - Start at the first value below desired value
  - Stop at next value above
- Index will be written before commit completes
- After crash:
  - New uncommitted records not in index
  - Uncommitted deleted records stay in index
  - Gfix reports index corruption

Back Version Before Record

When the back version is on a different page

- Write the back version first
- Write the record pointing to the back version next

After crash:

- Old record still exists
- New back version wastes space
- Gfix reports orphan back versions

Fragment Before Record

- Record bigger than page size
  - Write the last page of the record
  - Write the next to last, point to the last
  - Write other pages in reverse order, pointing to prior
  - Write the first bit, pointing to next page
- After crash:
  - Record fragments are unusable space
  - Gfix reports orphan record fragments

Page Allocation

Allocation:

- Mark page as allocated on PIP
- Format page
- Enter page in table, index, or internal structure

After crash:

- Page is unusable
- Gfix reports orphan page

Page Release

Release

- Remove page from table or index
- Mark page as unallocated

After crash:

- Page is unusable
- Gfix reports orphan page

Precedence

If index page A points to a record on page B, page B must be written before page A.

If the record on page B has a back version on page C, page C must be written before page B.

Firebird maintains a complete graph of precedence.

If a cache conflict requires writing page A, C and B must be written first.

If the graph develops a cycle, all pages must be written.

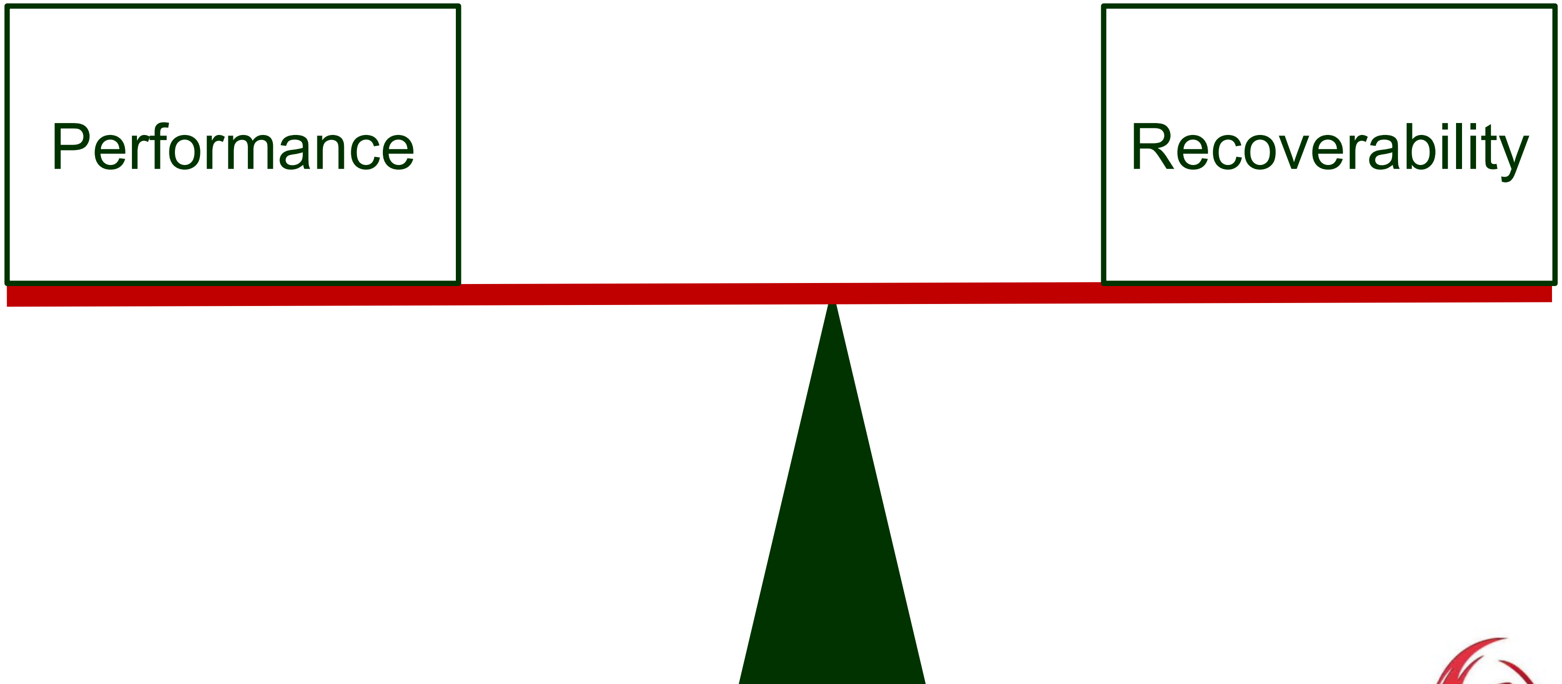
Downsides of Careful Write

Writes are random.

Precedence may cause multiple writes.

Cycles cause multiple writes.

Design is Balance



Disaster Recovery

From DBMS crash

From OS crash

From CPU crash

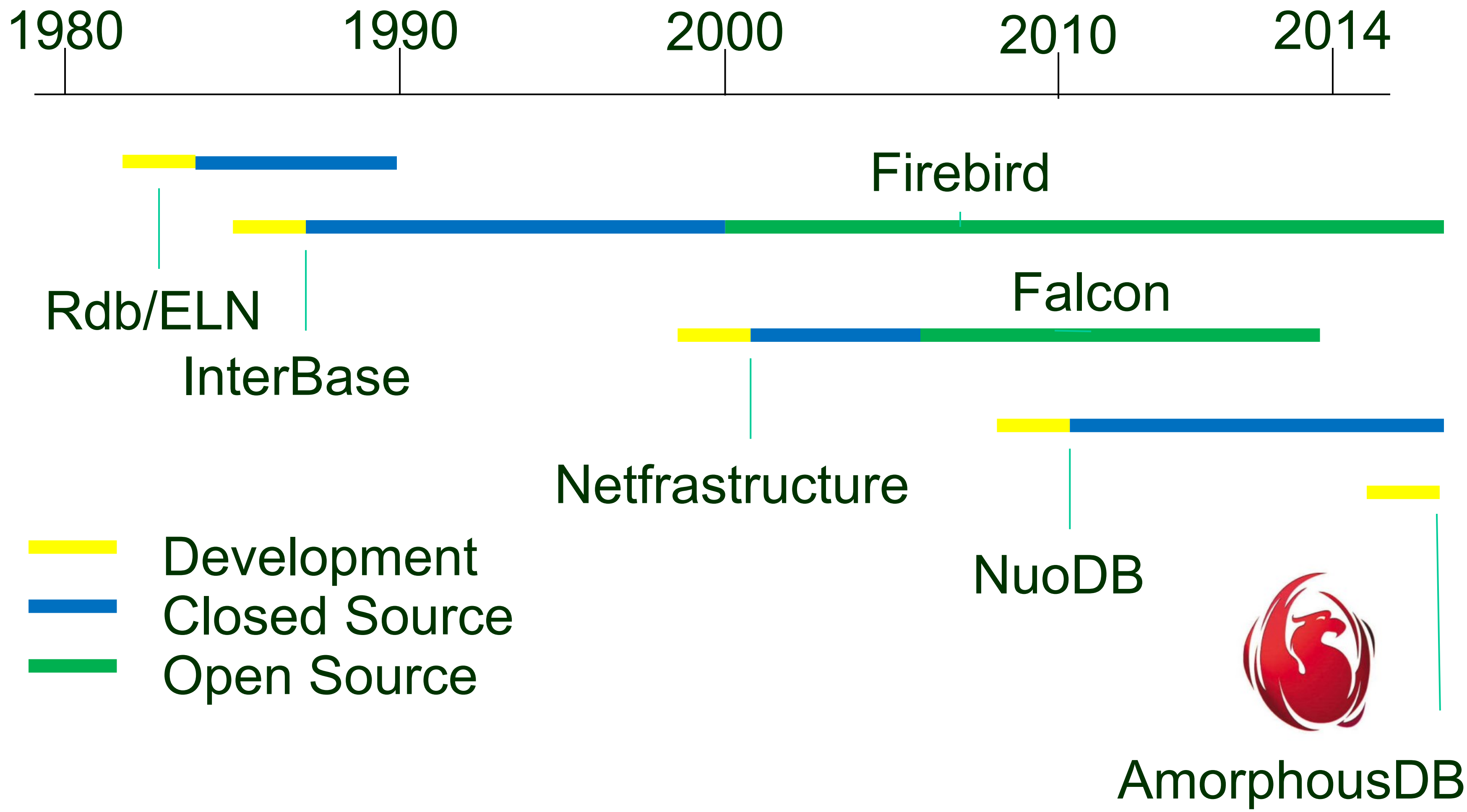
From Network failure

From Disk Crash

Antediluvian Technology
Long Term Journaling

- Before and after page images are journalled
- Required a *Tape Drive* (now extinct)
- Recovery
  - Roll forward from dump
  - Rollback from the current disk image
- Performance bounded by tape speed

JRD's Across History



Interbase 1.0, 1985
(Actually gds/Galaxy 1.0)

- MVCC + Careful Write
- Disk shadowing (raid not invented yet)
- GLTJ: Long term journal server
  - Dumped database to journal when enabled
  - Journalled page changes (or full page)
  - GLTJ could be shared among databases
  - Rarely, if ever, used
- Performance constrained by disk speed

Falcon

MVCC in memory

Disk used as back-fill for memory

Serial log for recovery

- Single log per database
- Page changes posted to log
- Log written with non-buffered writes
- Pages written when convenient

Performance constrained by CPU

NuoDB

DB layered on distributed objects called *Atoms*

Atoms replicate peer to peer

MVCC at Atom level

Transaction nodes pump SQL transactions

Storage managers persist serialized Atoms

Storage managers use serial log for replication messages

NuoDB Transactions

DBA has control over commit policy:

- Commit when transaction node sends commit messages
- Commit when <n> storage managers acknowledge commit messages
- Commit when <n> storage managers have written commit messages to serial log

Performance Implications
Disk Based MVCC

Many disk writes per transaction

Batch commit is possible

Performance is dozens of transactions per second with forced write

Higher transaction rate with buffered writes, but at risk of data loss

SSDs are a big win

Performance Implications
Serial Log

With fine granularity threading and 8 cores, benchmarked at 22,000 TPS

Serial log management is critical

Requires substantial non-interlocked data structures

Performance Implications
NuoDB

Bench marked at 3,000,000 TPS running on 40 commodity processors

Read only TPS is theoretically infinite

Questions?