Databases for Flash-based Systems

Dr Nigel Day, Technical Director
nigel.day@polyhedra.com

Enea – embedded for leaders

- The world's leading supplier of real-time operating systems, middleware, development tools, database technology and professional services for high-availability distributed multiprocessing applications such as telecommunications infrastructure, mobile devices, medical instrumentation, and automobile control/infotainment.
- Enea's flagship operating system, OSE, is deployed in approximately half of the world's 3G mobile phones and base stations.
- Global software company with a strong professional services offering.
- We provide customer value through complete embedded solutions, reduced development time and cost.
- ~500 employees; revenue SEK ~700 M (approximately $100M)

Enea Market Offering

Software	- NASP (Network Architecture Services Platform) - Element™ Middleware Platform - OSE™, Real Time Operating System - Optima™ Eclipse-based Tools - Polyhedra™, In Memory Database
Consulting services	- Application development, systems integration, testing - Hardware and software design - Training - 300 professional consultants
Third party products	- Best of breed tools, network protocols and applications

Enea Polyhedra™

- Family of relational database products for embedded use
  - Polyhedra and Polyhedra64 – in memory for speed
    - Hot standby configurations for high availability
  - Polyhedra FlashLite – data in flash for low RAM footprint
- Polyhedra project started in 1991
- First released in 1993, now on release 6.3 (Q2 2006)
  - Polyhedra FlashLite first released March 2006
- Polyhedra company bought by Enea AB in 2001
  - Product still developed and supported by original team

Patterns of data use in embedded systems

- Structures are relatively static, but should be dynamically changeable without data loss
- Changes can be frequent, but rarely involve many records
- Queries can be very frequent (and/or need fast response)
  - Notification technologies can reduce the need for 'polling'
- Database structure tuned for known queries and operations
  - Retrieve (part of) a record via an indexed field
  - Scan a table to retrieve (parts of) records matching specified criteria
  - Update a set of records (perhaps in different tables) where the indexes/primary keys are known

What is a database

- A set of structured data and access mechanisms
- Logically separated from the applications that use it
  - Different apps can share data 'safely'
  - Structure on disk/file not known/accessible to user code
- Best known database model: Relational
  - Data is held in a set of tables; queries can span tables
  - Fully transactional
  - Industry standard access language (SQL), APIs (ODBC, JDBC)
- Note for purists:
  - 'Database' refers to the information; **D**ata**b**ase **M**anagement **S**ystem or DBMS is the software used to store and access it

Databases versus data stores

- Databases are logically separate
  - Flexible: can add tables, columns, etc without invalidating running applications
- Usually, client-server architecture
  - Data better protected
  - Overall app can be split into smaller, simpler applets
  - Cross machine connections possible
- Standard APIs
  - Skills and code more reusable

- Data store keeps information in native data structures
  - Faster
  - Less flexible
  - Less portable
- Usually, data kept in same address space
  - Vulnerable to application errors!
- Non-standard APIs
  - Makes learning, application portability more difficult

Why use a database?

- To store information…
  - … that needs to be **preserved**
    - Static configuration information
      - Includes user data, such as phonebooks, ring tones, high scores, play lists, account information, …
    - Event logs
  - … that needs to be **shared**
    - Including transient data that does not need to be preserved
      - Status information
      - Dynamic configuration data
  - … that must be **controlled**
    - Not all parts of the system needs access to all the data
    - Changes must obey rules, be transactional
  - … whose **structure may change** over time

How DBMS's use Flash

Concentrating on Polyhedra and
Polyhedra FlashLite

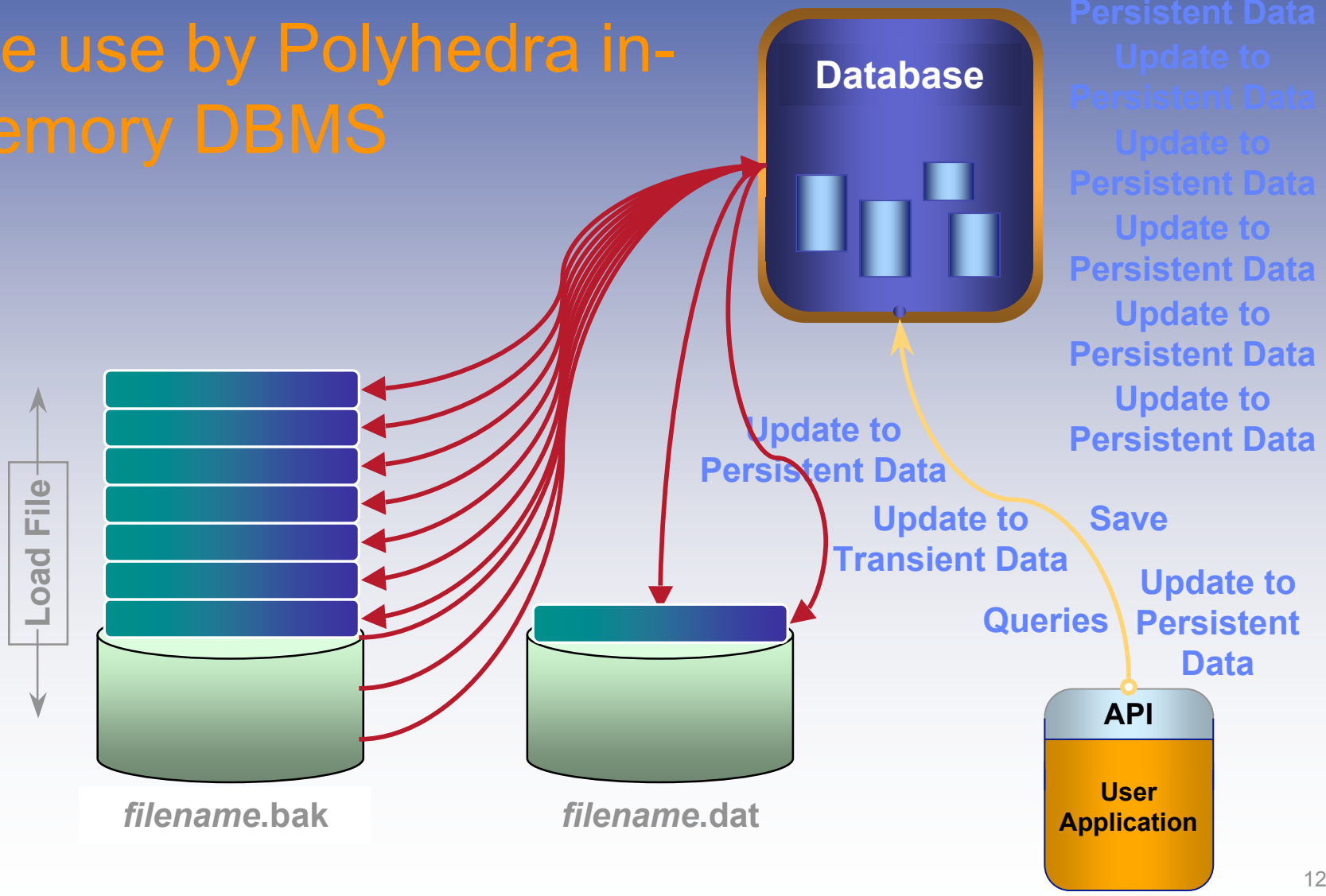
Typical disk use by standard DBMS

- Altering the disk to record what is changing is easy – recovering after a failure is less so
  - If a transaction fails, must be able to patch back to where we were before: need to store rollback info
  - If the system fails mid-transaction, must be able to patch back to a consistent state
  - If rollback info might be lost, we need separate mechanism to patch system back after system failure
- Belt-and-braces approach:
  - Write rollback info in separate file, flushing the file before doing the alteration to the 'real' data
  - Write journalling info to a separate file, flushing at the end of each transaction; can be used with a backup copy of database to rebuild a corrupted database.

Polyhedra in-memory DBMS

- Main copy of the data is kept in RAM
- Snapshots on demand, to write a copy of the database to file
  - Open a temporary file for writing
  - Write the schema information and table contents
  - Close file, rename old file to one side, rename temporary file
- Journal records are appended to the snapshot
  - Done post-transactionally, asynchronously for speed
  - Sequential write, then synch/flush
- On startup, snapshot is read, then all journal records
  - File marker positioned at end of snapshot or last valid journal record, ready for next transaction.
- Very simple pattern of use!

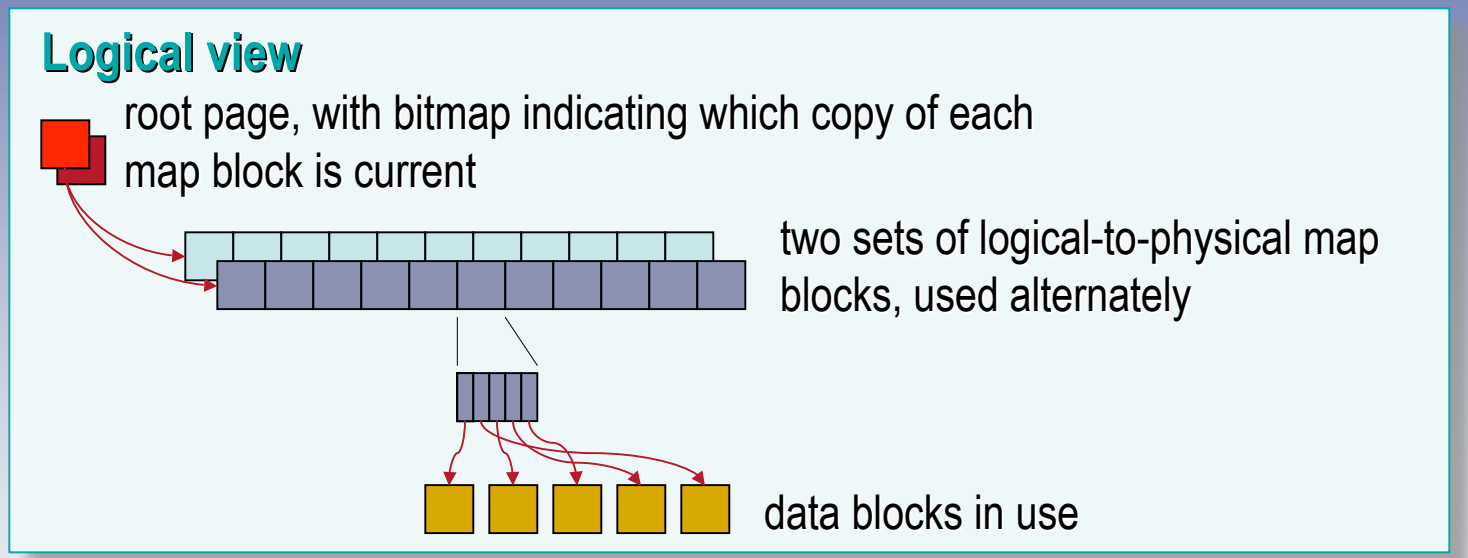
File use by Polyhedra in-memory DBMS



Polyhedra FlashLite

- Aims
  - Polyhedra functionality, but reduced code and RAM footprint
    - Relational DBMS, SQL, ODBC, transactional
  - **Optimised for Flash rather than Hard Disk**
- Assumptions on Flash
  - Reading is fast, but data not necessarily directly addressable
    - (treat as) page-based, with all pages approximately the same cost of access
  - Writing can be slow, and is to be minimised
  - A flash manager is available, to handle remapping of bad blocks, wear levelling, erasure, etc.
    - Alternatively, interface routines can be provided by the customer
    - Isolates us from the low-level access issues!

Polyhedra FlashLite 'file' usage



Polyhedra FlashLite's use of Flash

- Treats a Flash file as a series of pages, with two copies of logical to physical map pages and 2 copies of a root block
  - Root blocks indicates which copies of map pages are current
- A transaction does not overwrite data pages in use
  - uses unused pages and then adjusts relevant logical to physical map pages and root block
    - Updated map pages are written to their spare copies, and new root block written to its spare copy
    - PolyLite can revert to older state if system crashes mid-transaction or if a transaction fails
    - Rollback is cheap!
- Flash file can be used via the file system
  - … or by customer-supplied interface functions

Use of Flash when querying

- Pages are read into cache as needed
  - Indexing reduces the number of pages that need to be read
  - Records for a table are localised, so table scans are efficient
- No page writes!
  - Polyhedra does not use temporary tables on backing store to keep the results of a query

Use of Flash when writing

- When records are being updated in a transaction:
  - The relevant data block(s) will be got into the cache, and modified
  - The cached copy of the relevant map block(s) are updated
  - The bitmap in the cached copy of the current root block is updated
  - The data block(s) are written to spare locations
  - The map block(s) are written to their spare locations
  - The 'file' is synced, to ensure above writes are done before next ones
  - The root block is written to the spare location, and flushed
    - Once completed OK, the transaction is 'durable'

Summary

- Database management systems are emerging that are designed with the capabilities of Flash storage in mind
- Their design seeks to minimise the number of writes while maintaining transactional safeness
- Simple pattern of use imposes little strain on wear-levelling software.

- *For more information on the Polyhedra DBMS family, including Polyhedra FlashLite, visit www.polyhedra.com or www.enea.com/polyhedra*