THE APACHE® SOFTWARE FOUNDATION
http://www.apache.org/

Apache Mavibot

A new Backend for Apache Directory Server

Emmanuel Lécharny
elecharny@apache.org
http://symas.com

LDAP CON BRUSSELS 2017

MAVIBOT

"BLUE BOAT" IN TURKISH

(THANKS TO ERSIN ER FOR THE NAME)

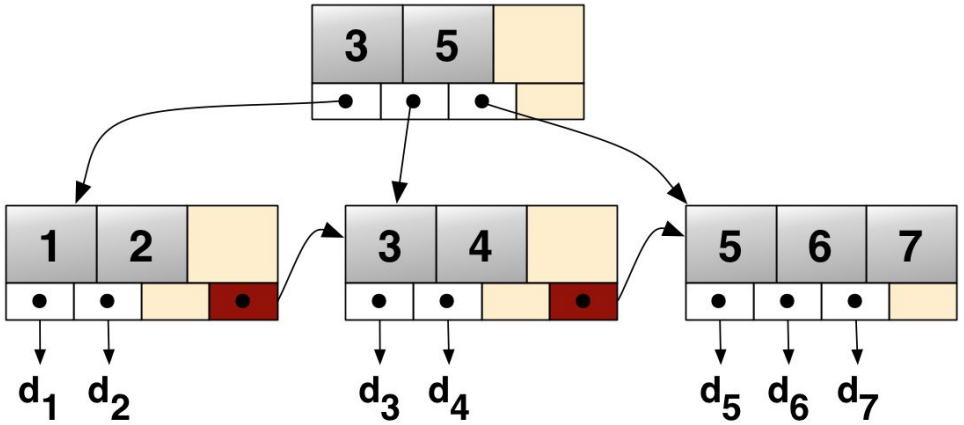
ALSO BUILT FROM MVBT

(MULTI-VERSIONS B-TREE)

Principles

B+TREES

- A NODE HAS AT LEAST N/2 KEYS AND UP TO N KEYS (EXCEPT THE ROOT NODE)
- LEAVES HAVE KEYS AND VALUES
- THE B+TREE IS BALANCED (IE ALL LEAVES ARE AT THE SAME LEVEL)
- OPERATIONS :
  - ADD
  - DELETE
  - BROWSE



B+TREES, CONTINUED

- LOG N OPERATIONS
- LOCKS ARE USED TO GUARANTEE CONSISTENCY
- ON AVERAGE, THE B-TREE NODES ARE 3/4 FULL
- TRANSACTIONS ARE COMPLEX TO IMPLEMENT...

B-TREES, WHAT FOR ?

- USED EVERYWHERE
  - DATABASE (INDEXES)
  - FILE SYSTEM
    - HSF+
    - NTFS
    - BTRFS
    - EXT4
    - REISER 4 (B*TREE)

MVCC

- Stands for "Multi-Version Concurrency Control"
- Invented back in 1978 (David P. Reed)
- Allows consistent concurrent reads and writes
- Locks can be avoided while reading
- Transactions can easily be implemented

Think about Version Control Systems...

Apache Directory Server

- Entries stored in a B-tree
- Index stored in B-trees
- In-memory B-trees needed
- Transaction support cross B-trees
- Multi-value support

Currently...

- Using JDBM
  - No cross B-tree transactions
  - No locks
  - No crash recovery system
  - No bulk load
- AVL trees in memory
- Locks all over the server to guarantee consistency

Other options ?

- BDB JE
  - Incompatible license...
  - Complex configuration
  - Heavy...
- LMDB
  - C code base (ie, JNI ...)
  - Fixed size initialization
  - Need a Java Wrapper

Mavibot

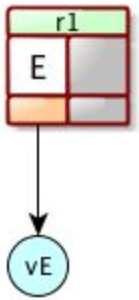
History

- CouchDB
  - 2006 conference
  - Sadly, Erlang code base
  - However, a perfect fit
- JDBM limits reached
  - 2011 : adding locks all over the server
  - 2012 : implementing a crash recovery system resulting in awful performances
  - No transaction (well, no cross B-tree transactions)
  - no bulk load tool
  - Database corruption

Most wanted features

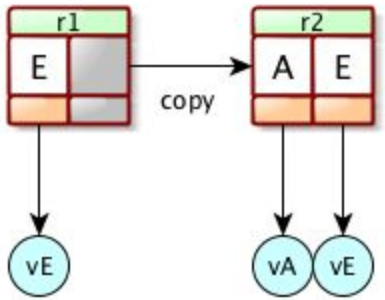
- MVCC
- Cross B-trees transactions
- Crash resistance
- Bulk-load
- In-memory and persisted
- Multi-value support
- fast

How does it work ?



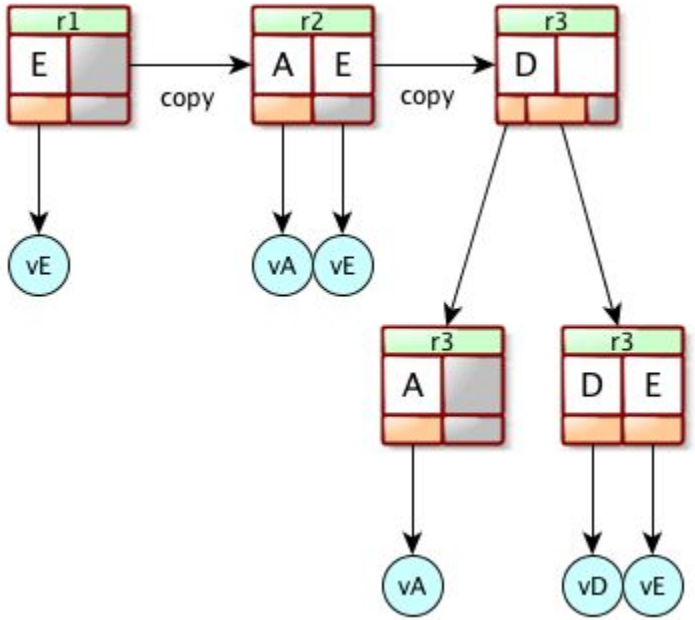
LDAP CON BRUSSELS 2017

How does it work ?



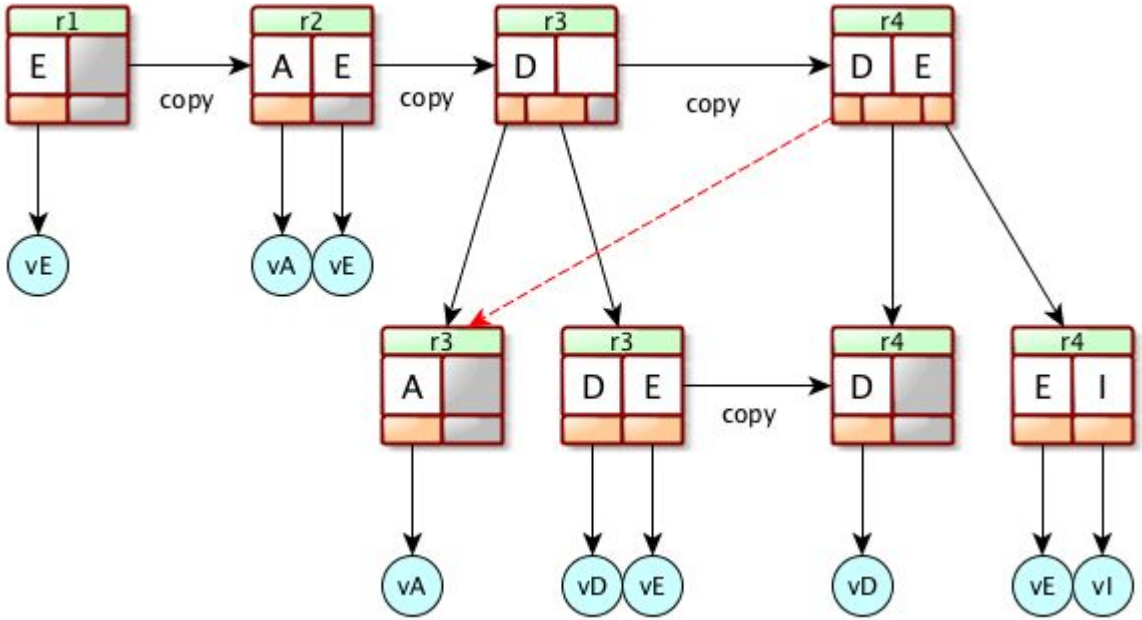
LDAP CON BRUSSELS 2017

How does it work ?



LDAP CON BRUSSELS 2017

How does it work ?



Operations

- Read
  - Cursor Browse()
  - Cursor BrowseFrom(K)
  - Boolean Contains(K, V)
  - V Get(K)
  - Boolean hasKey(K)
- Write
  - Delete(K)
  - insert(K, V)

Cursor

- Navigation
  - afterLast()
  - beforeFirst()
  - next()
  - prev()
- Action
  - close()
  - get()
- State
  - available()
  - hasNext()
  - hasPrev()
  - isClosed()

B-tree management

- Adding a B-tree :
  - Create its definition
  - Insert it in the RecordManager
    - Creates the B-tree
    - Stores its first revision in the BoB
    - Adds it to the list of managed B-trees
- No deletion (atm)

Example : init

```
RecordManager recordManager = new RecordManager( "/tmp/mavibot.db" );

// Create a new BTree
try ( WriteTransaction transaction = recordManager.beginWriteTransaction() )
{
    btree = recordManager.addBTree(
        transaction,
        "test",
        LongSerializer.INSTANCE,
        StringSerializer.INSTANCE );
 }
```

Example : insert

```
long[] values = new long[]
{
    14, 7, 43, 37, 49, 3, 20, 26, 17, 29,
    40, 33, 21, 18, 9, 30, 45, 36, 12, 8
};

try ( WriteTransaction writeTxn = recordManager.beginWriteTransaction() )
{
    for ( long value : values )
    {
        BTree<Long, String> btree = writeTxn.getBTree( "test" );

        btree.insert( writeTxn, value, Long.toString( value ) );
    }
}
```

Example : Browse

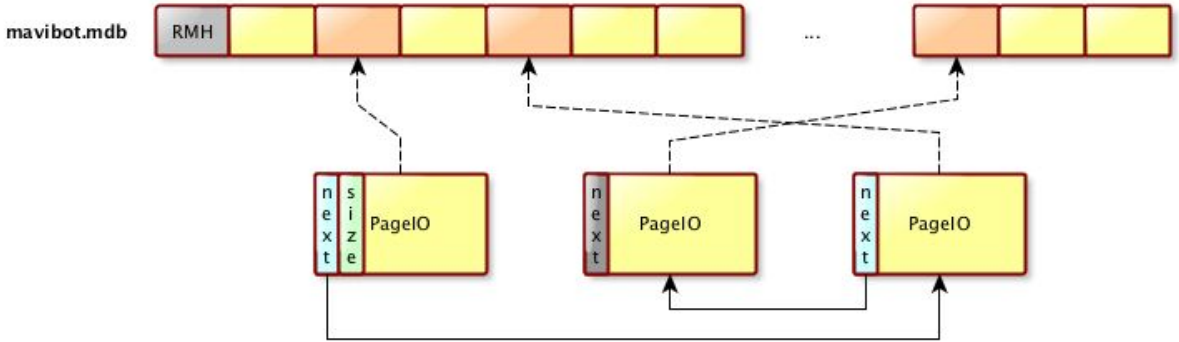
```
try ( Transaction readTxn = recordManager.beginReadTransaction() )
{
    BTree<Long, String> btree = readTxn.getBTree( "test" );
    TupleCursor<Long, String> cursor = btree.browse( readTxn );

    while ( cursor.hasNext() )
    {
        cursor.next();
        Tuple<Long, String> tuple = cursor.get();

        System.out.println( "<" + tuple.getKey() + "," + tuple.getValue() + ">" );
    }
}
```

Let's get our hands dirty...

Mavibot layout : File

Mavibot database is stored in a single file

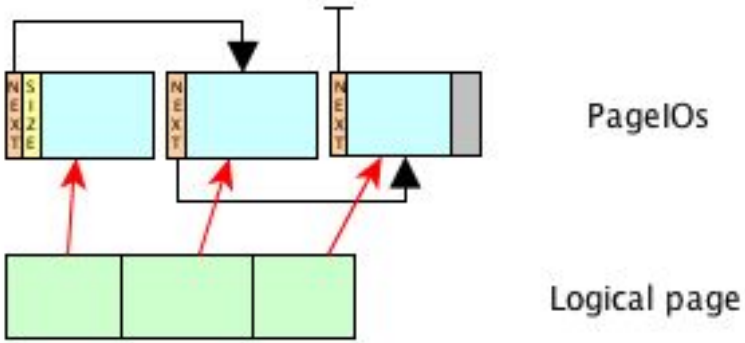
This file is split in fixed size physical pages that may be linked together : pageIO



Mavibot layout : PageIO

Physical pages (PageIO) are holding the content of logical Pages :

- Leaf
- Node
- BTreeHeader
- BTreeInfo

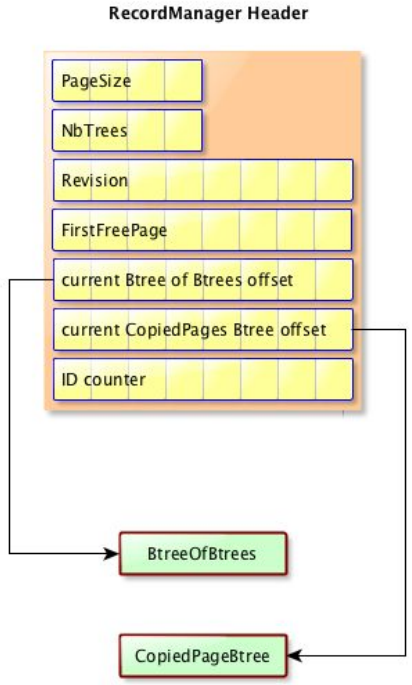


Mavibot layout : RMH

The RecordManager Header is the very first page

It stores references to Mavibot content :

- B-tree of B-trees
- Copied Pages B-tree
- Free pages list
- ID Counter



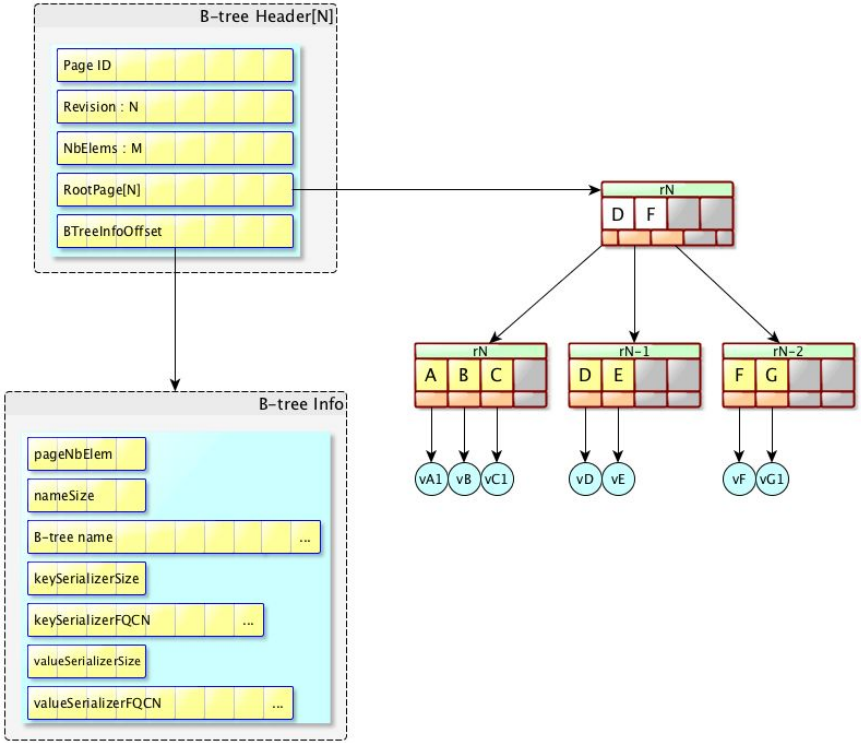
RecordManager Header

- One single page, can be written in one I/O
- Always updated after every revision creation or version cleanup
- Stores the current status :
  - Current revision
  - Current page ID
  - Number of managed B-trees
  - Page-IO size

Mavibot layout : B-tree

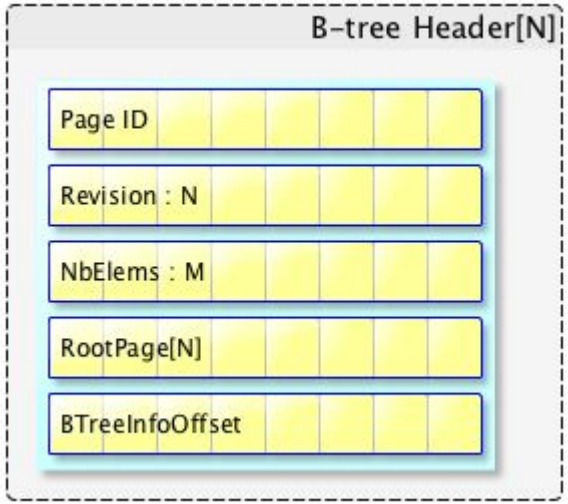
A Mavibot B-tree is :

- A B-tree header page
- A B-tree info page
- Some pages
  - Nodes
  - Leaves



B-tree header

- The B-tree entry point
- Points to the Root-Page
- Points to the B-tree info
- Contains the number of elements
- Contains the current revision

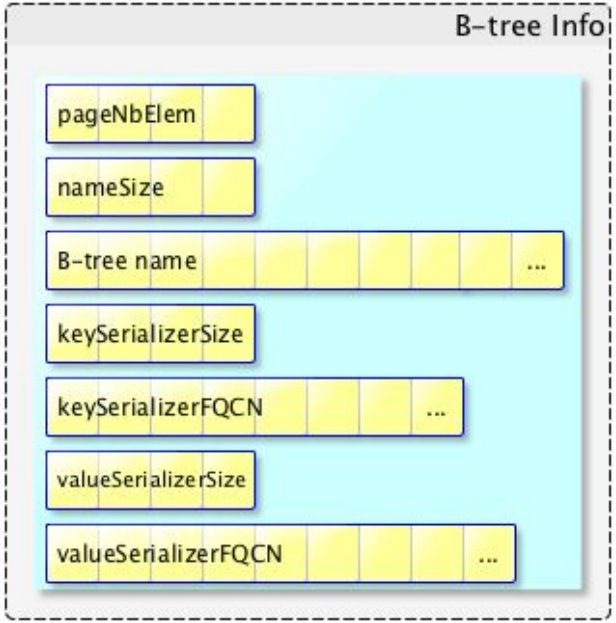


B-tree Info

The B-tree informations

- B-tree name
- Nb elements per page
- Key serializer
- Value serializer

Immutable



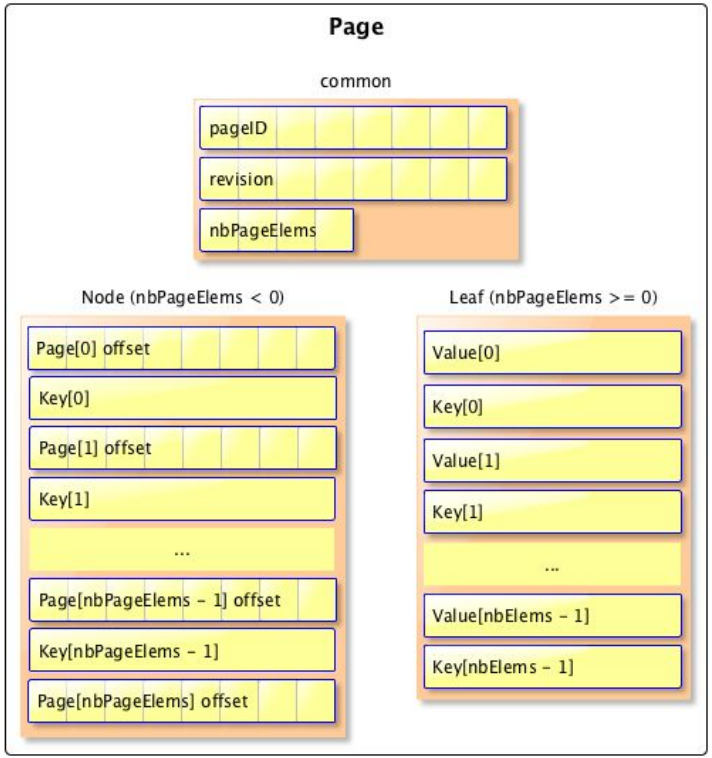
B-tree pages

2 flavors of pages :

- Leaf : up to N keys and Values
- Node : up to N keys and N+1 offset

Every page has :

- A unique ID (a 64 bits long)
- A revision
- The number of elements stored



B-tree of B-trees

- This B-tree stores a reference to any **managed** B-tree (except itself and the Copied Pages B-tree)
- We can retrieve a given **B-tree** from it
- New **revisions** are stored in this B-tree

Read at startup to load each B-tree in its latest version

Copied Pages B-tree

- When we create a new version, the copied pages are stored in this B-tree
- When the revision is freed, the associated copied pages can be released and moved to the free list

Free pages list

- We use a list of free pageIOs
- They are re-used on demand
- When empty, we create a new pageIO at the end of the file
- The free pageIOs are linked
- We can grab more than one pageIO
- Writes use one single thread, no lock needed for this list

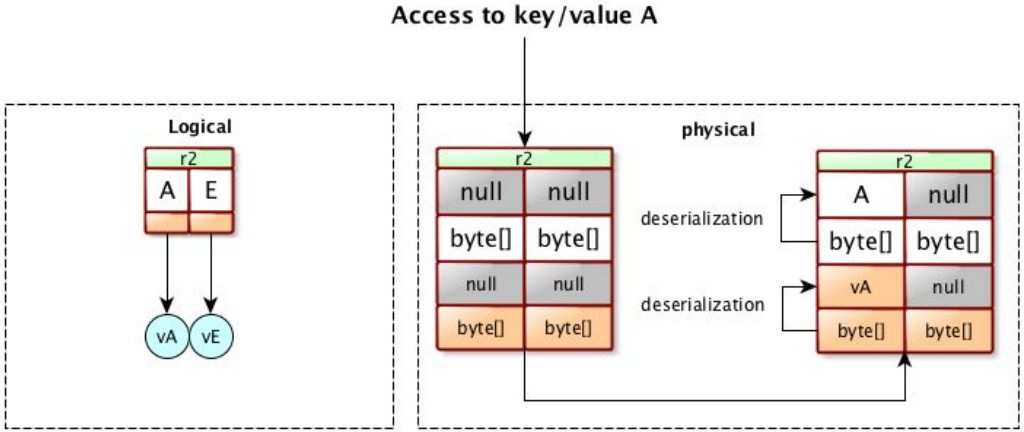
Features

(De)Serialization

Values/Keys has to be (de)serialized (byte[] <-> object)

(De)Serialization is co:

Differing saves time



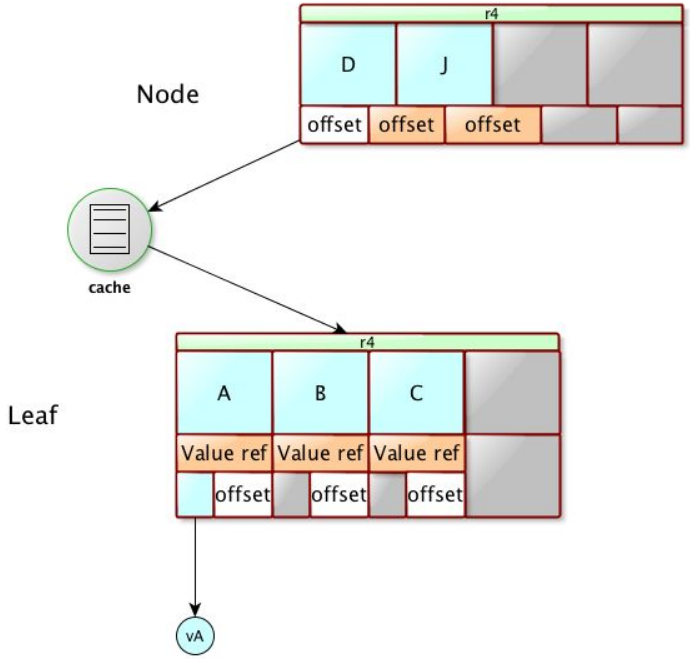
Cache

Node's children are referenced using their page offset

Pages are cached (LRU)

Better have a big cache !

WeakReference is too slow...



Free pages management

Node's children are referenced using their page offset

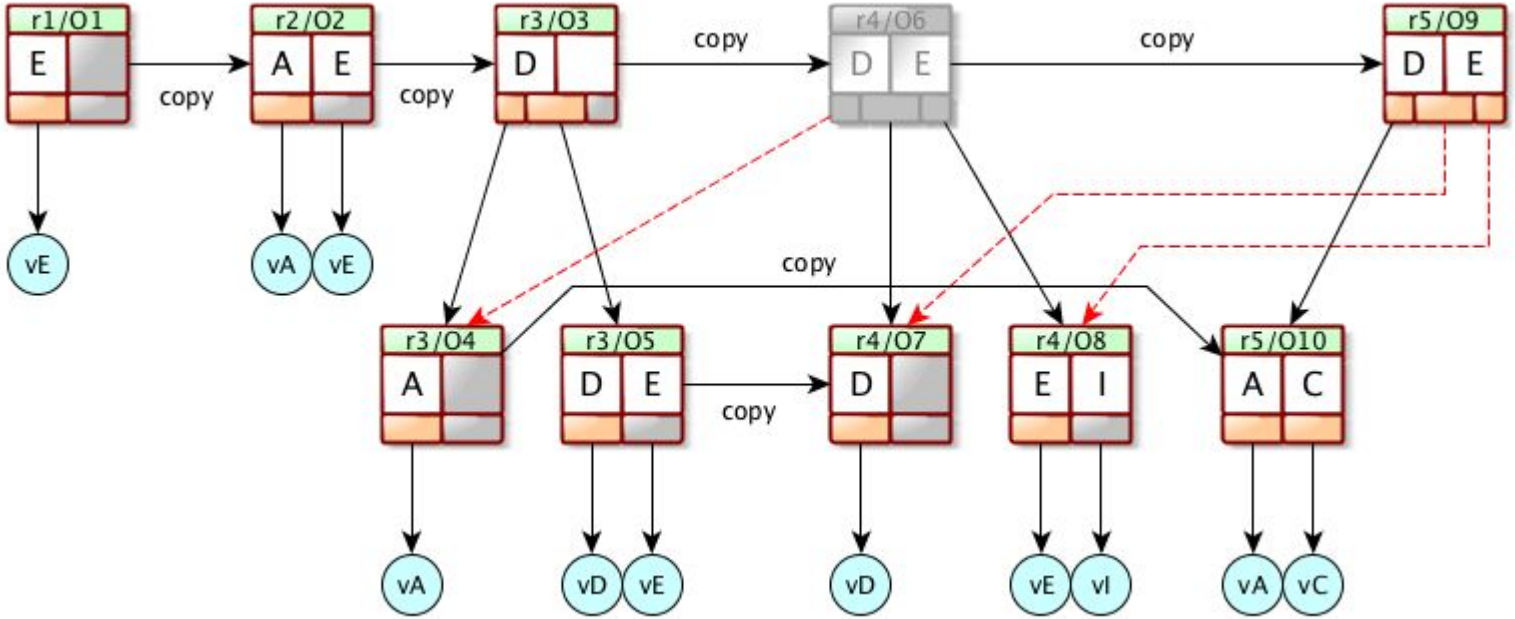
Pages are cached (LRU)

Better have a big cache !

WeakReference is too slow...

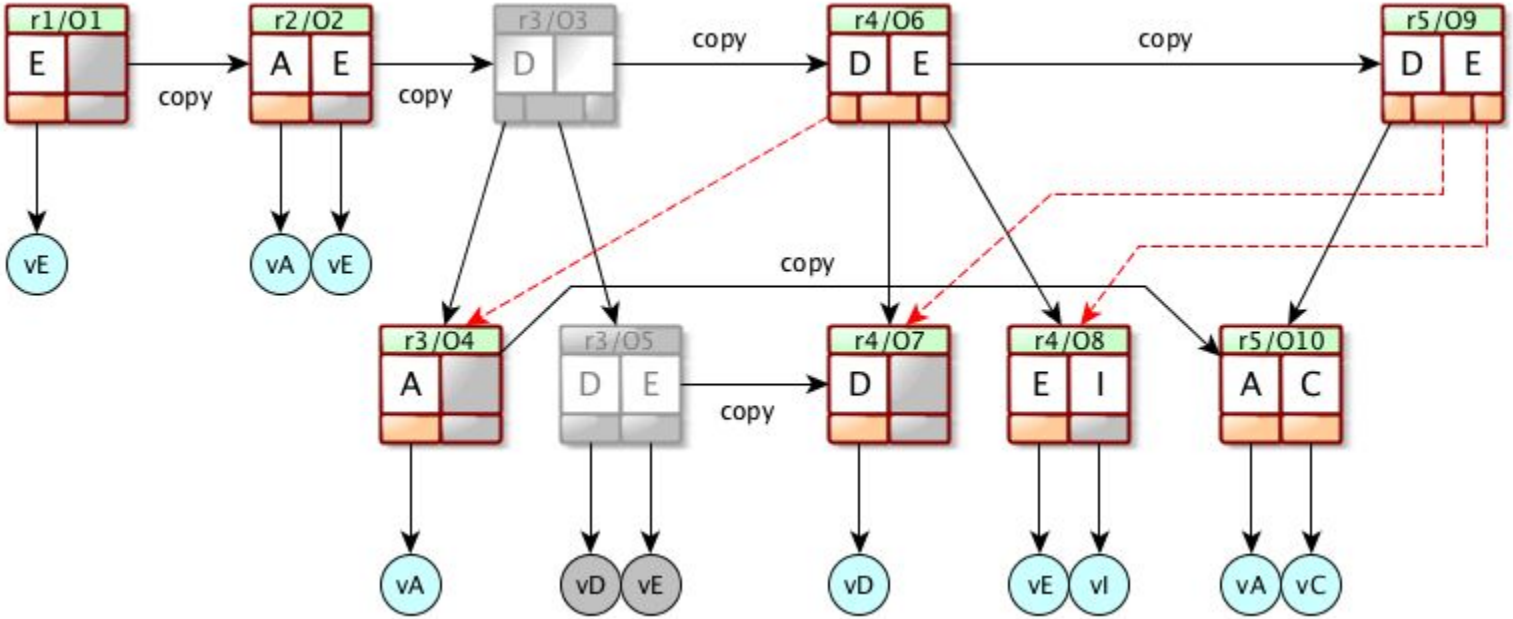
LDAP CON BRUSSELS 2017

Releasing version 4



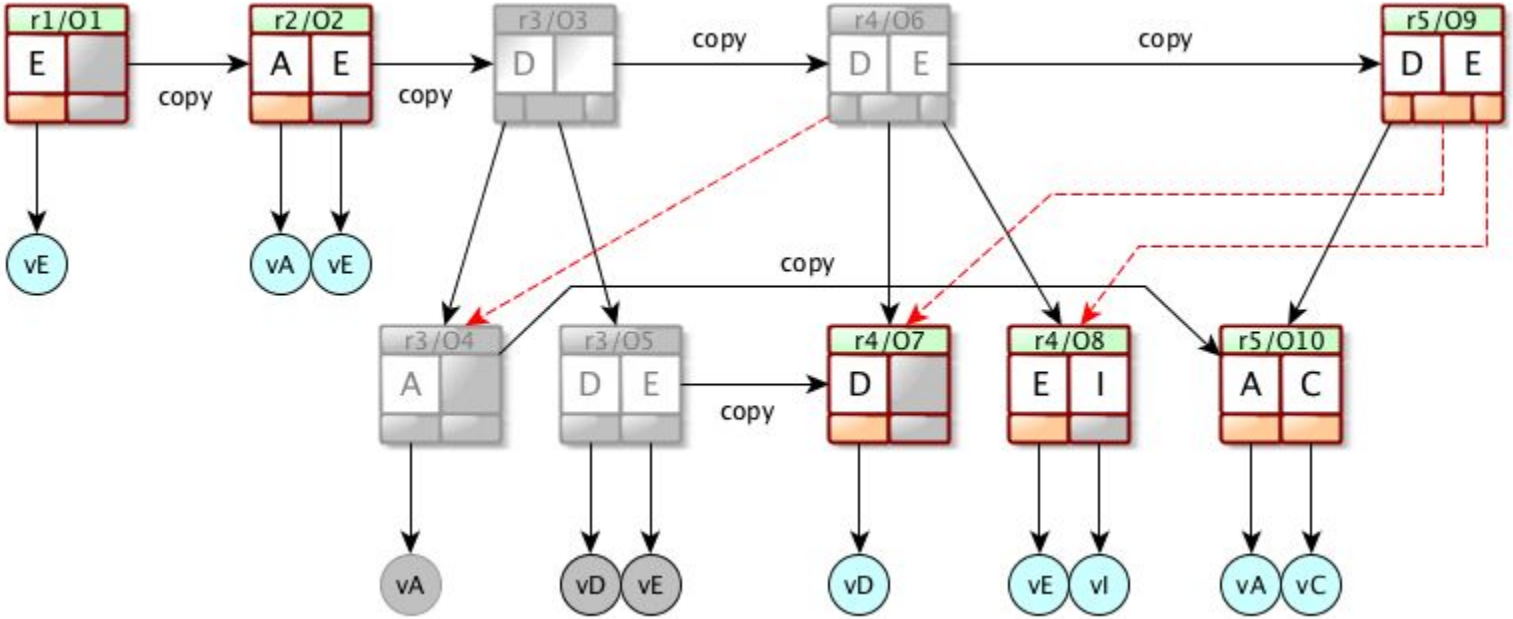
LDAP CON BRUSSELS 2017

Releasing version 3



LDAP CON BRUSSELS 2017

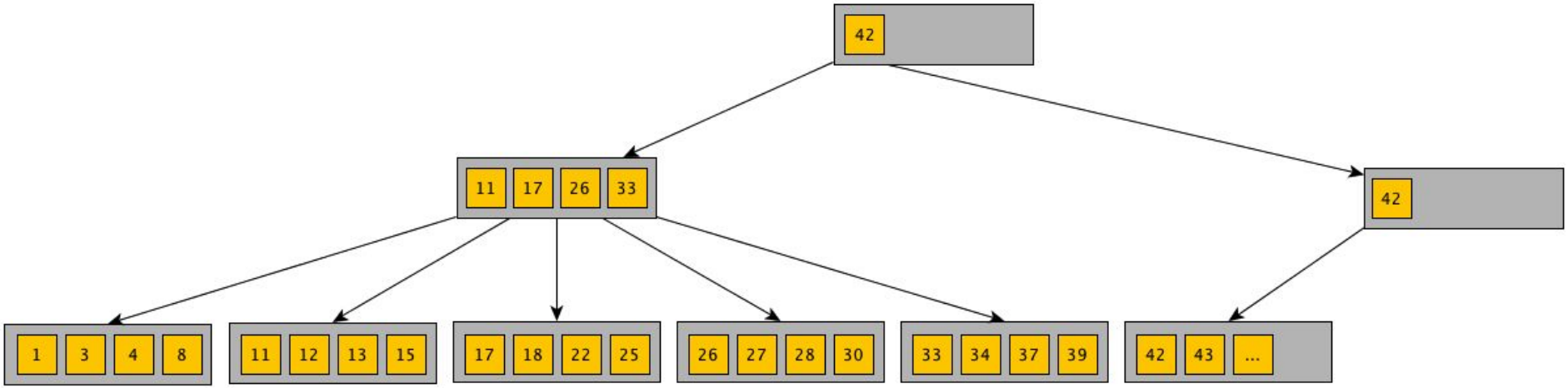
Releasing versions 3 and 4



Bulkload

- Load the data from the bottom up
- Data must be sorted beforehand
  - Quicksort in memory O(n log(n))
  - Merge sort when memory is limited O(n log(n))
  - Result is dense
- Big transactions
  - Simpler but slower
  - Less dense

Bulkload (2)

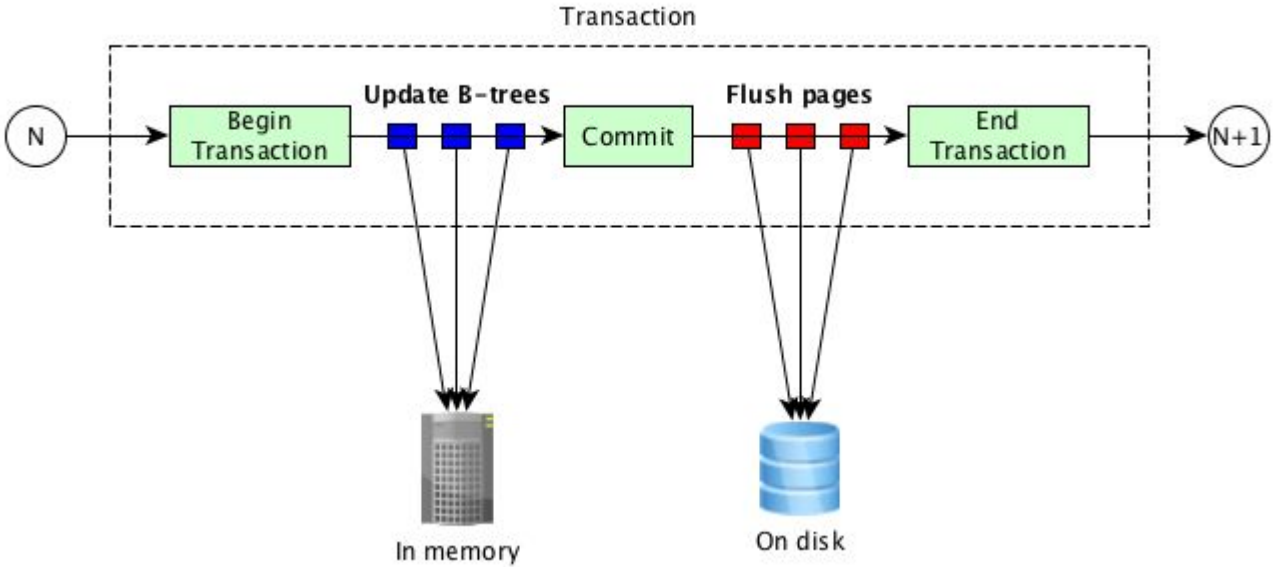

42
11 17 26 33
42
1 3 4 8
11 12 13 15
17 18 22 25
26 27 28 30
33 34 37 39
42 43 ...

Transactions

Read vs Write transaction

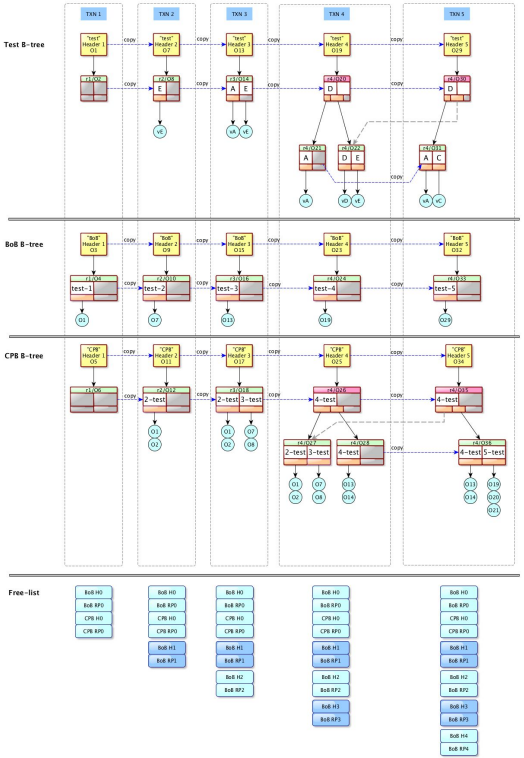
- Read transactions are concurrent
- Write transactions are serialized
- Every committed write transaction creates a new version
- Write transactions can gather more than one operation
- Read/Write transactions are cross-B-trees

Transaction processing



LDAPCon BRUSSELS 2017

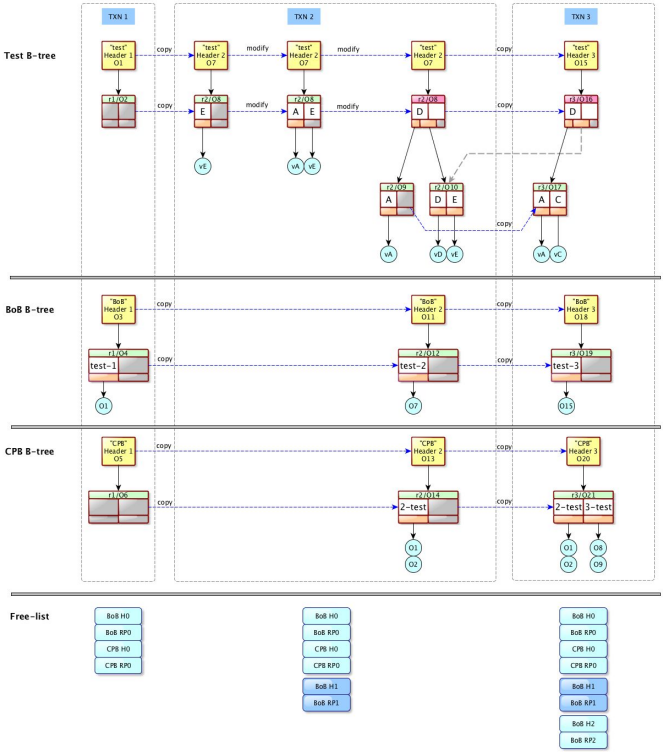
ONE VS MANY OPERATION/TXN



<- ONE

VS

MANY ->



Mavibot + ApacheDS

Current version

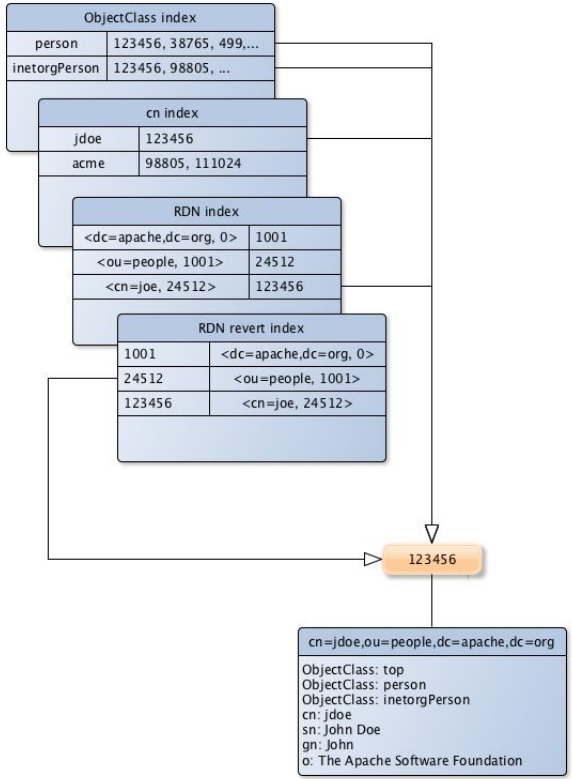
- ApacheDS embed version 1.0.0-M8
- Twice faster than JDBM
- No transaction support in 1.0.0-M8
- Free page management is not optimal
- Multi-value elements supported
- In-memory B-tree supported

New version

- No multi-value support : ApacheDS to deal with it
- Cross-B-tree Transaction support
- Free page management on the fly
- No in-memory B-tree
- Faster than 1.0.0-M8
- Cache

ApacheDS Index/B-trees

- Entry
- ApacheDS index
  - * Present
  - * ObjectClass
  - * entryUUID
  - * EntryCSN
  - ** RDN + reverse
  - Alias + reverse
  - OneAlias
  - subAlias
  - AdminRole



Conclusion

Not a big code base

- 18 253 slocs
- Long lasting effort (day job, family events...)
- Solve a lot of issues
- Should be completed soon !

Q+A

Thanks !