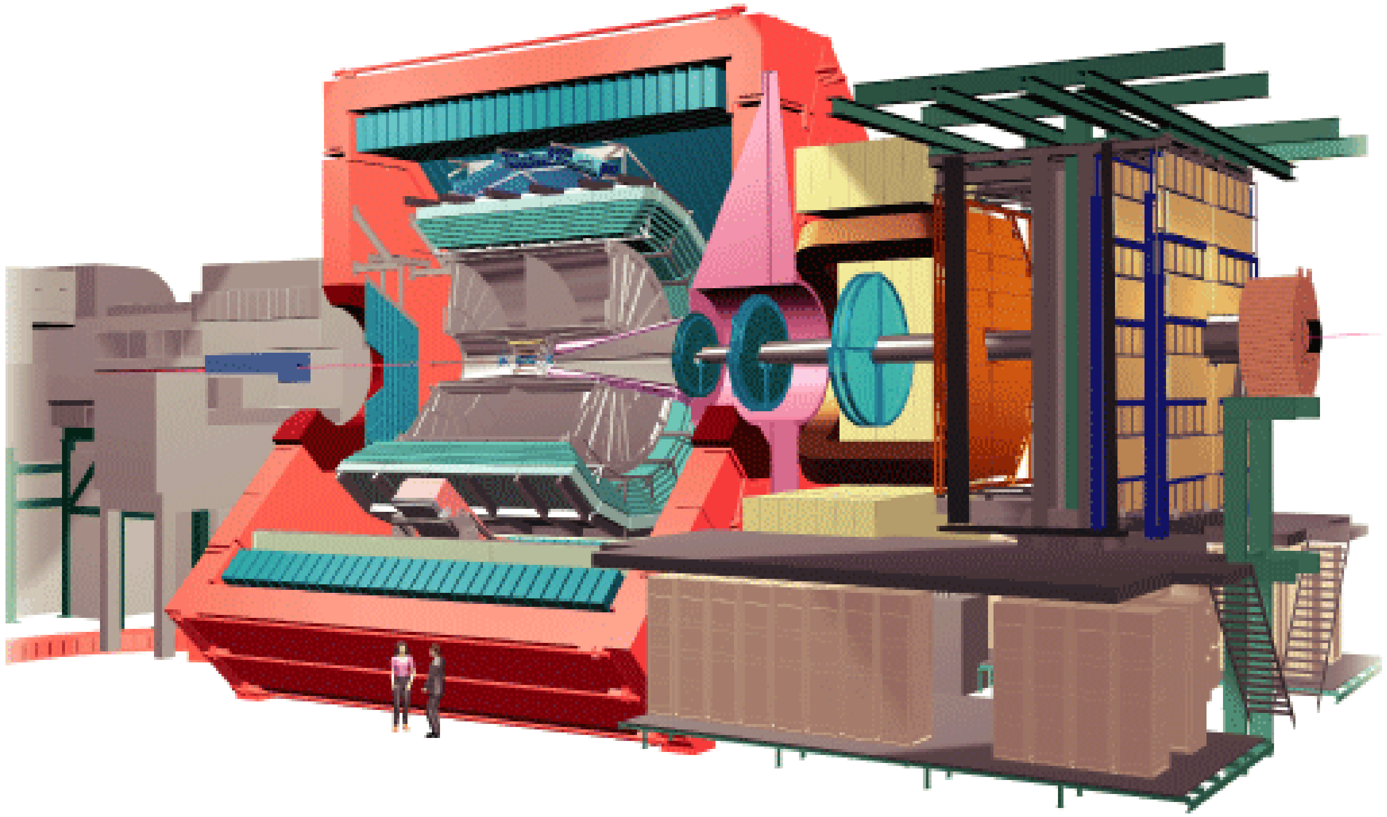
The new AliRoot DB access classes

Alberto Colla

(Alice off-line Calibration and Alignment grup)



Alice off-line meeting

Cern, October 3, 2005

Summary

- History
- Underlying principles
- New features (wrt first publication, June 2005)
- Description of the CDB access classes
- Examples of use cases

“History” of DB access classes

- Original idea and first implementation by **T. Kuhr** (late 2004)
- Since February 2005: work on the framework implementation is carried out in the **core offline group**
- First presentation of the prototype to the off-line community: **June 2005 Alice off-line meeting**
- Development performed taking into account the many and useful discussions with **software** and **detector experts** which followed the publication of the prototype

Underlying philosophy

- The Alice offline calibration and alignment framework provides the **software infrastructure** for **storage and access to the experiment condition data**
- Calibration and alignment objects are **Root TObjects** stored into **Root files**
- Calibration and alignment objects must be **run dependent objects**
- Database is **read-only** (automatic versioning tools)
- The framework provides storage and access into **Grid** and **local** environment
- Storage and retrieval technique is **transparent to the user**

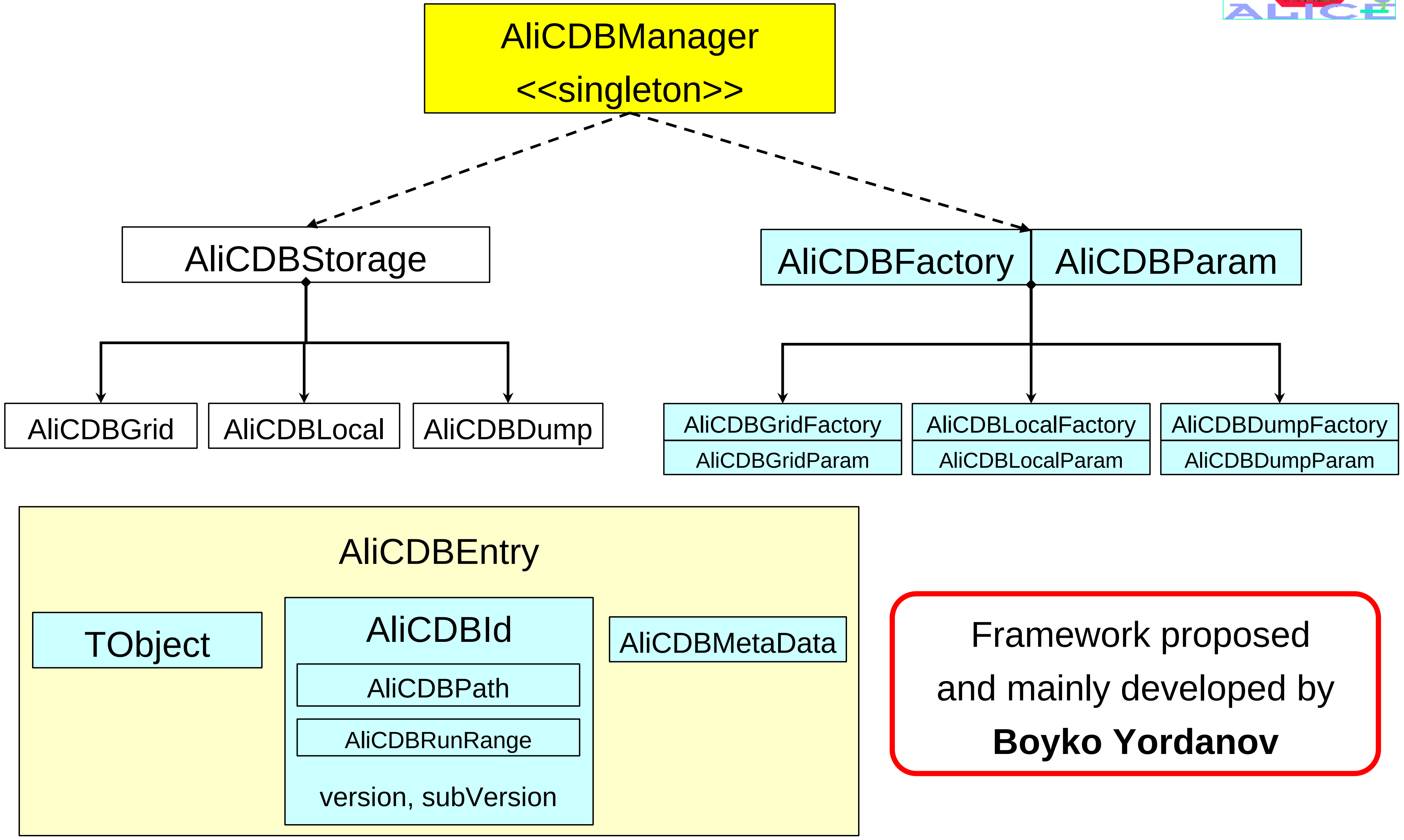
New features (Introduction)

- Implementation of the Grid storage access class **AliCDBGrid**
- New manager class **AliCDBManager**
  - Handles **activation** and **deactivation** of one or more storage system
  - **Owns** the instances of the active storages
- “**Factory**” and “**Parameter**” classes associated to each specific storage
  - Used by the manager to **activate storage locations**
  - Storage systems are identified by a **string** (“**uri**”) or **set of parameters**
- New **versioning schema** introduced
  - **Two** version numbers: **“Grid” version** and **“local” (sub)version**
- Object's container class (**AliCDBEntry**) and object's **metadata classes** have been **redesigned**

Software requirements

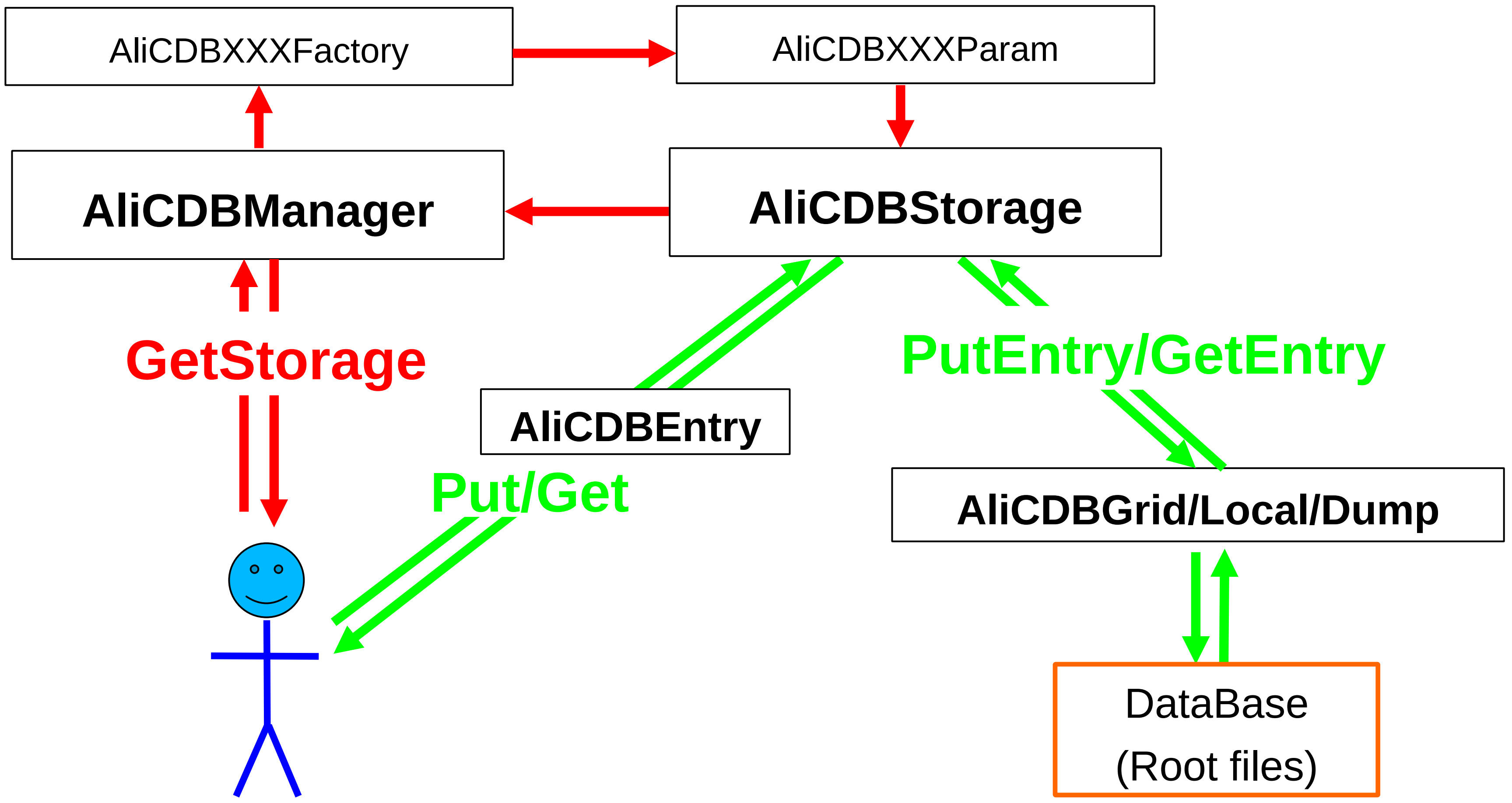
- AliRoot **HEAD**
- Root **v5-04-00**
- For Grid access: Alice VO registration, AliEn client (**gShell**)
- AliCDB* classes are in STEER:
  - **AliCDBManager**
  - **AliCDBStorage**
  - **AliCDBGrid**, **AliCDBLocal**, **AliCDBDump**
  - **AliCDBEntry**
  - **AliCDBId**
  - **AliCDBMetaData**

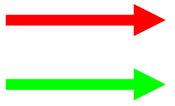
CDB access classes schema



Framework proposed
and mainly developed by
Boyko Yordanov

CDB access classes relationships





Summary

- History
- Underlying principles
- New features (wrt first publication, June 2005)
- Description of the CDB access classes
- Examples of use cases

AliCDBEntry

- Container class. It has:
  - The **calibration or alignment object** (anything inheriting from **TObject**)
  - The **object's identifier** (**AliCDBId**)
  - The **object's metadata** (**AliCDBMetaData**)

- Remember: Each AliCDBEntry contains a **single object** (which can be a container of more objects). It is identified by a **name** (**path**) and its validity is specified by a **run range** and a **version**.

- Some public AliCDBEntry methods:
  - `SetObject(TObject*), TObject* GetObject()`
  - `SetId(const AliCDBId&), AliCDBId& GetId()`
  - `SetMetaData(AliCDBMetaData*), AliCDBMetaData* GetMetaData()`
  - `SetOwner(Bool_t), Bool_t IsOwner()`

  SetOwner sets AliCDBEntry object as the **owner** of the TObject and AliCDBMetaData objects (so that they are deleted with AliCDBEntry)

AliCDBId

- Contains the set of the object's metadata which **uniquely identifies it** (**path**, **run validity range**, **versions**)
- It has two purposes:
  - During storage it is used to **build the location** (e.g. directory path, file name) where the object will be stored
  - During retrieval it is used to **identify the object** and, if needed, to **specify the required version**

- Data members:
  - `AliCDBPath fPath:` the object's path
  - `AliCDBRunRange fRunRange:` the object's validity range
  - `Int_t fVersion, Int_t fSubVersion:` the object's Grid and local versions
  - `TString fLastStorage:` “previous” storage location of the object (new, grid, local, dump). It is set at first storage and during object's retrieval and helps to “backtrace” the object's history.

AliCDBPath, AliCDBRunRange

- AliCDBPath contains the **object's path name** (`TString fPath`)
- The path must have a **three-level directory structure**:

`“level0/level1/level2”`

- Example: **`“ZDC/Calib/Pedestals”`**
- Wildcard character **`*`** allowed if path is used to specify **selection criteria** or for **multiple object retrieval** (e.g. **`“ZDC/*”`** or **`“TPC/Calib/*”`** . . .)

- AliCDBRunRange contains the **run validity range** of the object

  (**`Int_t fFirstRun, Int_t fLastRun`**)

- **AliCDBId** contains public **getter/setters** for path, run numbers, versions ...

AliCDBMetaData

- Contains the set of the object's metadata **not used** for storage/retrieval

- Data members and getter/setters for:
  - **Object's class name** (`TString`)
  - **Responsible's name** (`TString`)
  - **AliRoot version** used for the object (`TString`)
  - **Beam period** number (`UInt_t`)
  - **Comment** string (`TString`)
  - `TMap` of any additional **set of** "**properties**":
    - TMap format: (`const char* property, TObject* object`)
    - Getter function to get the metadata object associated to "property":

      `TObject* GetProperty(const char* property)`
    - see also: `RemoveProperty(…), PrintMetaData()`

AliCDBManager

- **Singleton** (`AliCDBManager::Instance()`)
- **Owner** of the activated storage object instances

- holds:
  - ➔ List of the **registered factories** (3 available storage factories: **Dump**, **Local**, **Grid**).
  - ➔ List (TMap) of **active storages** (storage object instances created with AliCDBManager::GetStorage())

- Factory registration is **hard-coded**; it is done at the first call of AliCDBManager::Instance()
  - ➔ AliCDBGridFactory is registered only if **Root** is **enabled for AliEn access**
  - ➔ If Grid factory is not registered the corresponding storage cannot be activated (null AliCDBStorage pointer returned)

AliCDBManager (2)

- To activate a new storage instance use AliCDBManager method `GetStorage`:

Storage type "URI"

```
➔ AliCDBStorage* GetStorage(const char* dbString);
➔ AliCDBStorage* GetStorage(const AliCDBParam* param);
```

Returns pointer to the active instance of AliCDBStorage

Set of parameters identifying the storage

- `GetActiveStorages()` returns **list of active storages**
- Public methods added to select single "**default storage**" and "**drain storage**" (see later)
- `Destroy()` method deletes AliCDBManager instance and all the active storages

AliCDBStorage

- **Interface** for the concrete storage types (**Dump**, **Local**, **Grid**)

- Public virtual functions to store/retrieve objects:

  - `Bool_t Put(AliCDBEntry* entry);`
  - `Bool_t Put(object, Id, MetaData)`

  - `AliCDBEntry* Get(const AliCDBId& query)` → **Single request**
  - `AliCDBEntry* Get("path", runNumber, version, subVersion)`

  - `TList* GetAll(const AliCDBId& query)` → **Multiple request**
  - `TList* GetAll("path", runNumber, …)`

AliCDBStorage (2)

- During retrieval, **AliCDBId query** is used to specify:
  - The **path** of the requested object (wildcards allowed for multiple requests)
  - The **run number**
  - Optionally, the **version** and **subversion** (highest version search if not specified)

- Possibility to specify a list of "**selection criteria**" has been mantained:
  - `Void AddSelection(const AliCDBId& selection)`
  - `Void AddSelection("path", firstRun, lastRun, version, subVersion)`
  - See also: `RemoveSelection(...)`, `RemoveAllSelections()`, `PrintSelectionList()`

AliCDBGrid

- **Access class** to an object stored into a **Grid database**
- Based on the Root **TGrid/Talien plugin**, uses **gliteUI libraries**

- U.r.i. pattern: `"alien://host:port;user;DBPath;SE"`

  Example:

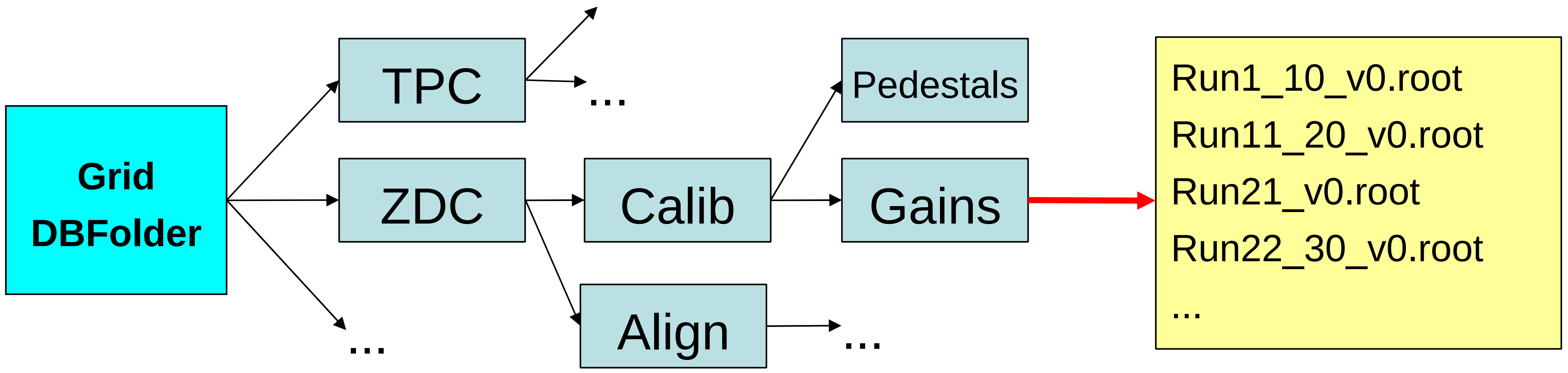
  `"alien://aliendb4.cern.ch:9000;colla;DBFolder;ALICE::CERN::se01"`

  ➔ If `DBFolder` is not a full path it is created from the home directory

- **AliCDBGridParam** members: `fHost`, `fPort` (UInt_t), `fUser`, `fDBPath`, `fSE` (TString)

- One single AliCDBEntry stored in each TAlienFile:

`DBFolder / level1 / level2 / level3 / Run#fr_#lr_v#gv.root`

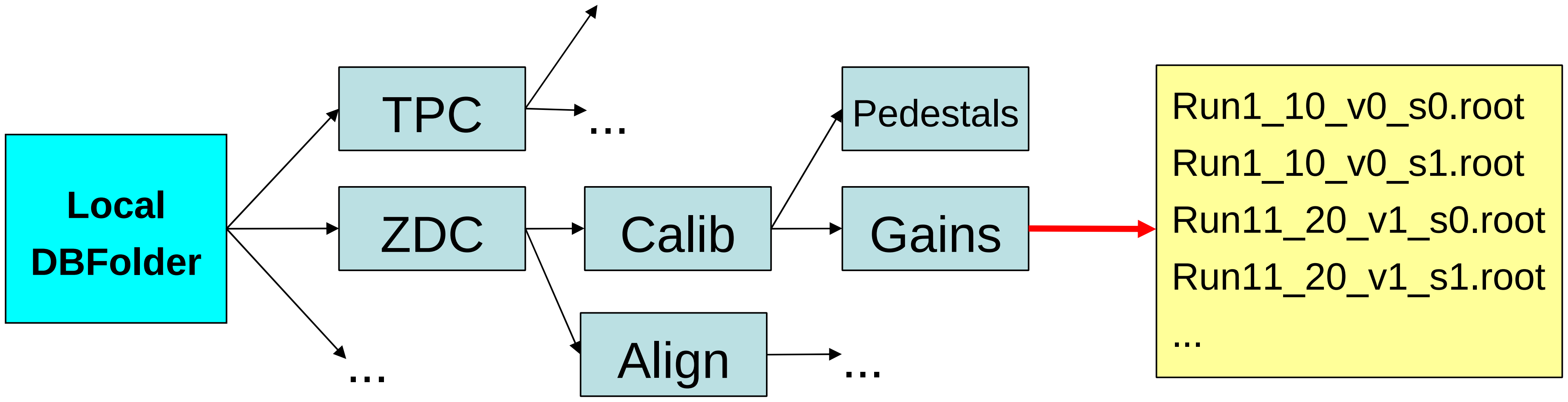


Run1_10_v0.root
Run11_20_v0.root
Run21_v0.root
Run22_30_v0.root
...

AliCDBLocal

- **Access class** to an object stored into a **local database**
- U.r.i. pattern: "`local://DBPath`"
  - ➔ If `DBPath` is not a full path it is created from the working directory
- **AliCDBLocalParam** member: `fDBPath`
- One single AliCDBEntry stored in each local root file:

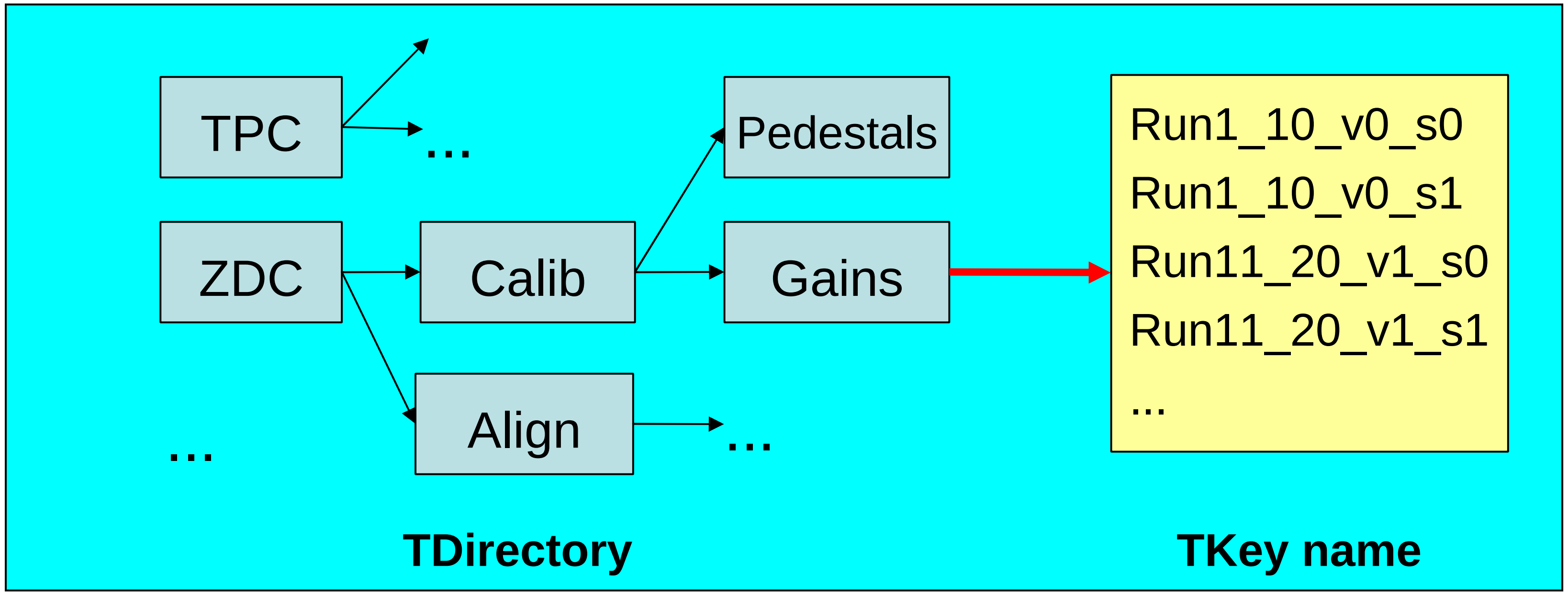
`DBFolder/level1/level2/level3/Run#fr_#lr_v#gv_s#lv.root`



AliCDBDump

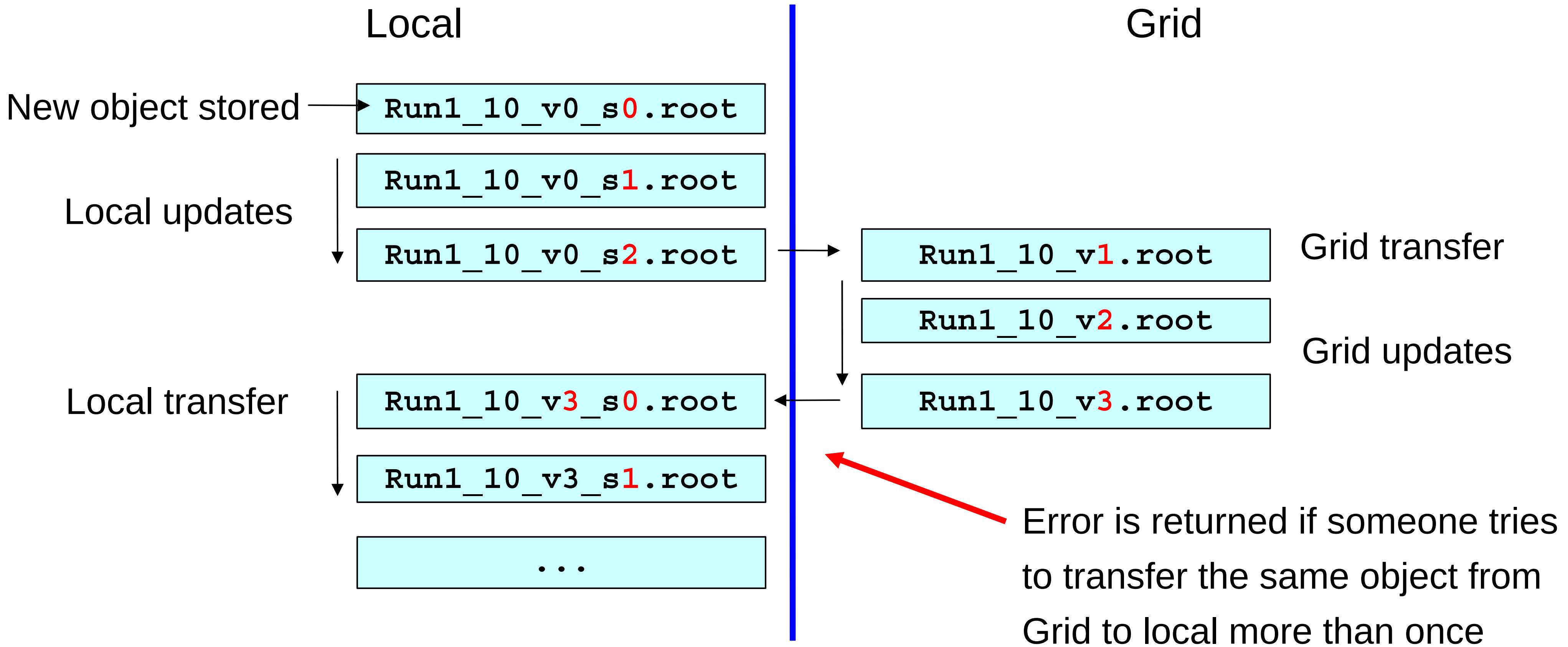
- **Access class** to an object stored into a **“dump” local file**
- U.r.i. pattern: `“dump://fileName(;ReadOnly)”`
  - ➔ If **fileName** is not a full path the file is created/opened in the working directory
  - ➔ If **ReadOnly** is specified the file is opened in read-only mode
- **AliCDBDumpParam** member: `fDBPath`, `Bool_t fReadOnly`
- All the AliCDBEntry objects stored in the dump root file:

Local DumpFile.root:



New versioning schema

- Object version is automatically set during storage
- Two version numbers: the first one stands for "**Grid version**", the second (**subVersion**) stands for "**Local version**"
- Example:



Error is returned if someone tries to transfer the same object from Grid to local more than once (protection against mess)

Summary

- History
- Underlying principles
- New features (wrt first publication, June 2005)
- Description of the CDB access classes
- Examples of use cases

Activation of new storage locations

- Using the storage's URI

```
AliCDBManager *man = AliCDBManager::Instance();

AliCDBStorage *storGrid = man->GetStorage
   ("alien://aliendb4.cern.ch:9000;colla;DBFolder;ALICE::CERN::se01");

AliCDBStorage *storLoc = man->GetStorage("local:///work/DBFolder");
```

- Using the AliCDBParam class

```
AliCDBGridParam param
   ("aliendb4.cern.ch",9000,"colla","DBFolder","ALICE::CERN::se01");

AliCDBStorage *storGrid =

     AliCDBManager::Instance()->GetStorage(&param);
```

Object storage

```
// some code to create the TObject *object
...
// set object's path and run validity range in AliCDBId
AliCDBId id("ZDC/Calib/Pedestals",1,10);
                   path              runRange
// Set additional object's metadata
AliCDBMetadata md;
md.Set... //fill metadata using AliCDBMetaData setters


// Put object into the database
storLoc->Put(object, id, &md);
```

➔ Object stored into local file:

`DBFolder/ZDC/Calib/Pedestals/Run1_10_v0_s0.root`

Object retrieval

- Single object retrieval

run

```
AliCDBEntry *entry;
entry = storLoc->Get("ZDC/Calib/Pedestals",5);
entry = storLoc->Get("ZDC/Calib/Pedestals",5,2);
entry = storLoc->Get("ZDC/Calib/Pedestals",5,2,4);
// Get Id, metaData, object from entry
AliCDBId id = entry->GetId();
AliCDBMetadata *md = entry->GetMetaData();
ObjClass *obj = entry->GetObject();
```

Look for highest version & subVersion

Look for version 2 & highest subVersion

Look for version 2 & subVersion 4

- Multiple object retrieval

```
TList *list; // list will contain AliCDBEntry obj's
list = storLoc->GetAll("ZDC/Calib/*",5);
entry = (AliCDBEntry*) list->At(0);
```

AliCDBEntry must be cast!

Object retrieval (2)

- Object retrieval using AliCDBStorage “selection criteria” methods:

```
// I want version 2 for all “ZDC/Calib/*” obj's for runs 1 to 100
storLoc->AddSelection(“ZDC/Calib/*”,1,100,2)
// and version 1_0 for “ZDC/Calib/Pedestals” obj's for runs 5-10
storLoc->AddSelection(“ZDC/Calib/Pedestals”,5,10,1,0)


TList *list = storLoc->GetAll(“ZDC/*”,5)
```

- “General” selection criteria (“ZDC/*”) should be added before more specific ones!

Default and Drain storages

- Among the active AliCDBStorage objects collected by AliCDBManager, one can choose two as the “default” and “drain” storages:

```
AliCDBManager::Instance()->SetDefaultStorage(const char* “uri”);

AliCDBManager::Instance()->SetDefaultStorage(AliCDBParam* param);

AliCDBManager::Instance()->SetDefaultStorage(AliCDBStorage* sto);


AliCDBManager::Instance()->SetDrain(const char* “uri”);

AliCDBManager::Instance()->SetDrain(AliCDBParam* param);

AliCDBManager::Instance()->SetDrain(AliCDBStorage* sto);
```

  - ➔ If the storage instance is not present in the collection it is created and added to it
  - ➔ The first created storage instance is automatically set as the default storage

- Removal of default and drain storages (objects aren’t removed from list of active storages!)

```
AliCDBManager::Instance()->RemoveDefaultStorage();

AliCDBManager::Instance()->RemoveDrain();
```

Use of default and drain storages

- To check the activation of the default and drain storage pointers:

```
(Bool_t) AliCDBManager::Instance()->IsDefaultStorageSet();

(Bool_t) AliCDBManager::Instance()->IsDrainSet();
```

- The pointer to the default storage is returned by:

```
AliCDBManager::Instance()->GetDefaultStorage();
```

- If the drain storage is activated, each entry retrieved from any storage is put into it:

```
AliCDBManager *man = AliCDBManager::Instance();

man->GetStorage("alien://..."); // this is the default storage

man->SetDrain("dump://DBDrain.root"); // this is the drain storage

AliCDBEntry *entry;

entry = man->GetDefaultStorage()->Get("ZDC/Calib/Pedestals",5);
```

➔ Retrieved entry is drained into dump file!

For further examples...

- Run tutorial macro `macros/DBAccessTutorial.C`
  - ➔It requires AliEn access! If AliEn is not enabled in Root, replace the alien storage activation with a local “dummy” one ...
- Follow today's **“live” tutorial!**

Proposal of a new storage schema

- New storage schema proposed by **Boyko Yordanov** which optimizes **time efficiency** of storage and retrieval processes

- Current implementation:

  - ➔ Every set of objects with same name (e.g. “ZDC/Calib/Pedestals”) is stored in the **same location** (“linearly”), regardless of their versions:

ZDC/Calib/Pedestals:

Run0_10_v1.root
Run0_10_v2.root
Run11_20_v1.root
...

  - ➔ For a modified object (new version), it is necessary to **iterate** over the already existing ones to get the **version number**.
  - ➔ The same number of iterations is needed for **automatic data retrieval** (highest version)

New data storage idea

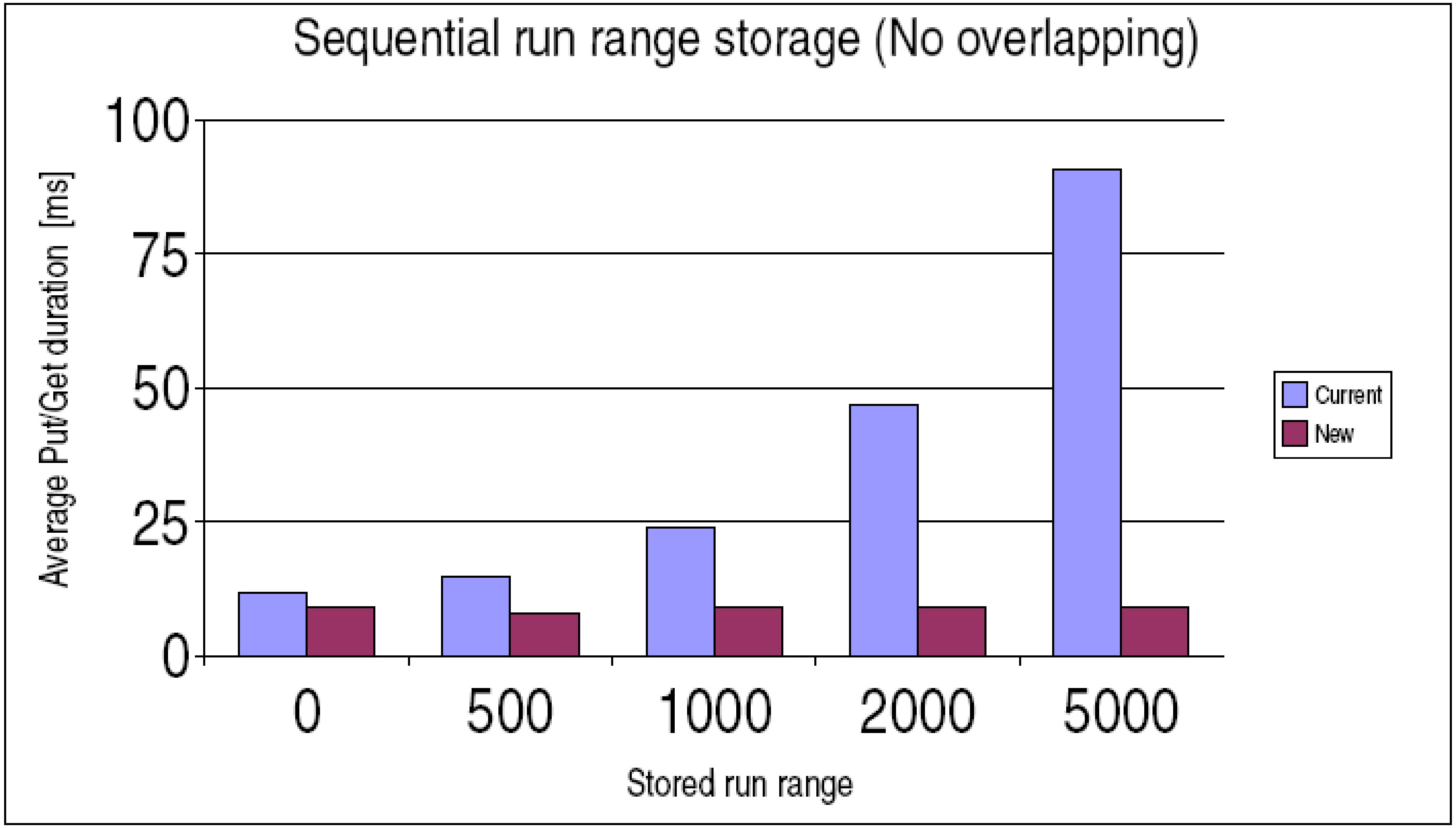
- For a new object (e.g. ZDC/Calib/Pedestals): new branch, and **for every version new sub-branch** with the **same name as the version number**
- The **“leaves”** are the objects (files, root keys etc.) with name determined by the run range (and possibly the version still appended for clarity)
- Taking into account that for a given version there is **no overlapping** run ranges, we can **order them**
- This structure allows for **less iterations** in most of the cases thanks to the additional version branch and “Binary Tree” optimization.

  ➔ example:

ZDC/Calib/Pedestals/**1/**
Run0_10_v1.root **Run11_20_v1.root** **...**
ZDC/Calib/Pedestals/**2/**
Run0_10_v2.root **...**

Performance tests

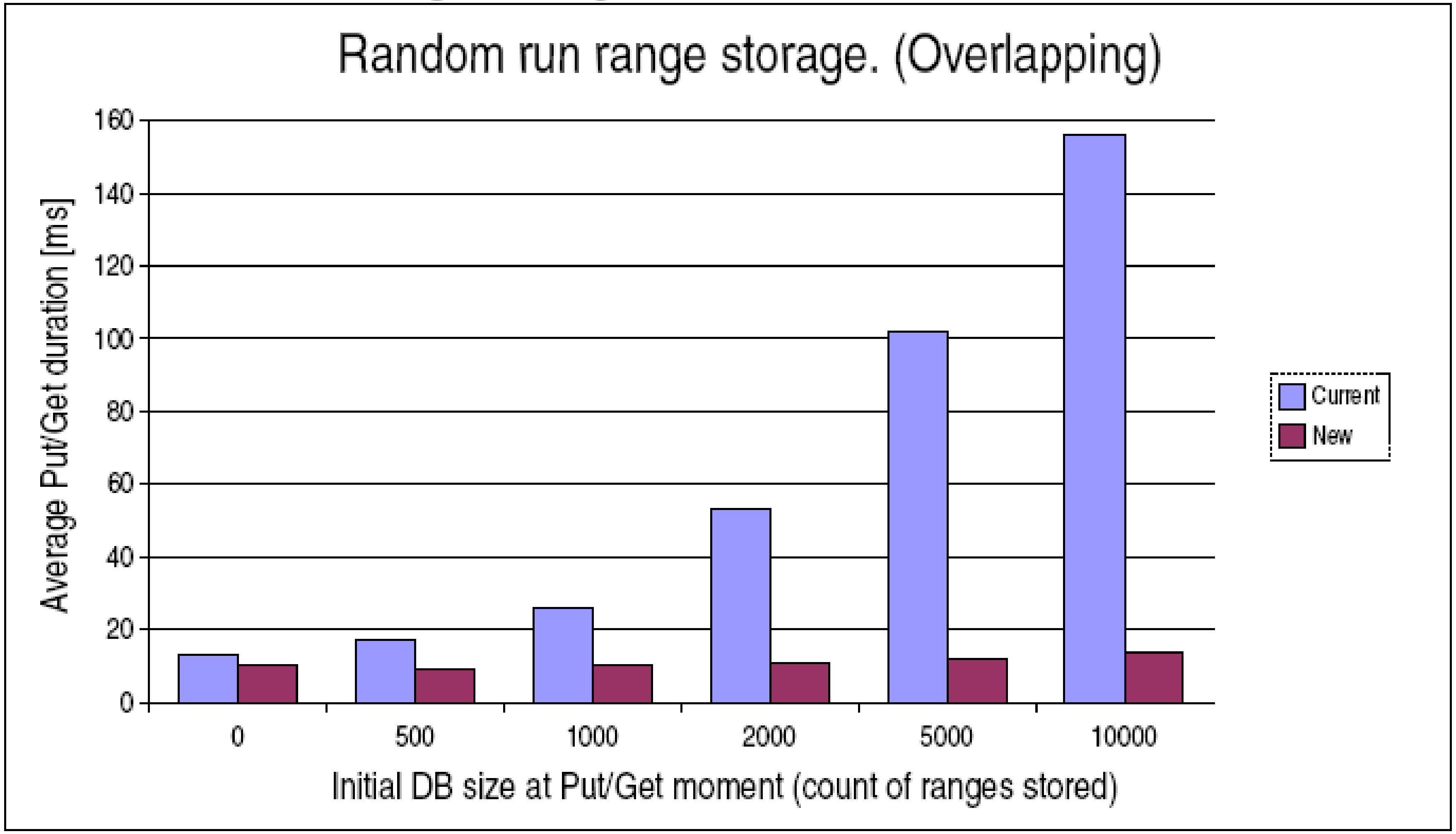
- Sequential run range storage (no overlapping):



This test stores and retrieves values for particular object increasing run range every time by one. There is only one version number. With the “current” method put/get time depends on the number of files. With the “New” method put/get time is constant.

Performance tests (2)

- Random run range storage (overlapping):



This test stores and retrieves values of particular object with random run range. Run range is overlapping and version number increases. With the “Current” method put/get time depends on the size of DB. With the “New” method put/get time is constant.