Contents:

- Present capabilities of AliEn and design limitations based on experience from

  - Implementation work
  - Analysis prototyping
  - PCD04 running

- Detailed implementation proposals for improvements towards a 2nd generation alice grid middleware as needed by an end -to-end analysis platform

Some thesis of me ...

- AliEn has been a very successful project with small man power thanks to Predrag, Pablo and others ...

- AliEn has been more successful compared to other official GRID projects because of its 'smallness'

  f.e. Perl AliEn Core code 45.000 lines

  (f.c. C++ aiod code 21.000 lines)

- AliEn has proven to have made the choices in the right directions towards a GRID system

... on the other hand ...

- AliEn is still a prototype system under heavy development not fullfilling 'professional' standards – mainly scalability and security barriers

AliEn security problems

- every user can kill via a SOAP call all jobs of other users
- every user can read all files in a storage element knowing the physical file name
- every user can delete all files existing in a storage element
- every user can call every implemented SOAP function in the central web services
- every user can kill via an AliEn job all jobs in a AliEn site
- every user can spy on the files used in the working directory by other jobs in the same site
- every user can submit via an AliEn job new jobs
- every user can fill the existing storage space
- every user can use all available CE ressources
- file access by aiod does not use a real authentication

=> this won't stand even a minimal security standard

=> AliEn misses a minimal security model for services and ressources! Only catalogue information is protected!

- there are solutions for a part of the problems, BUT they are not in place ! -

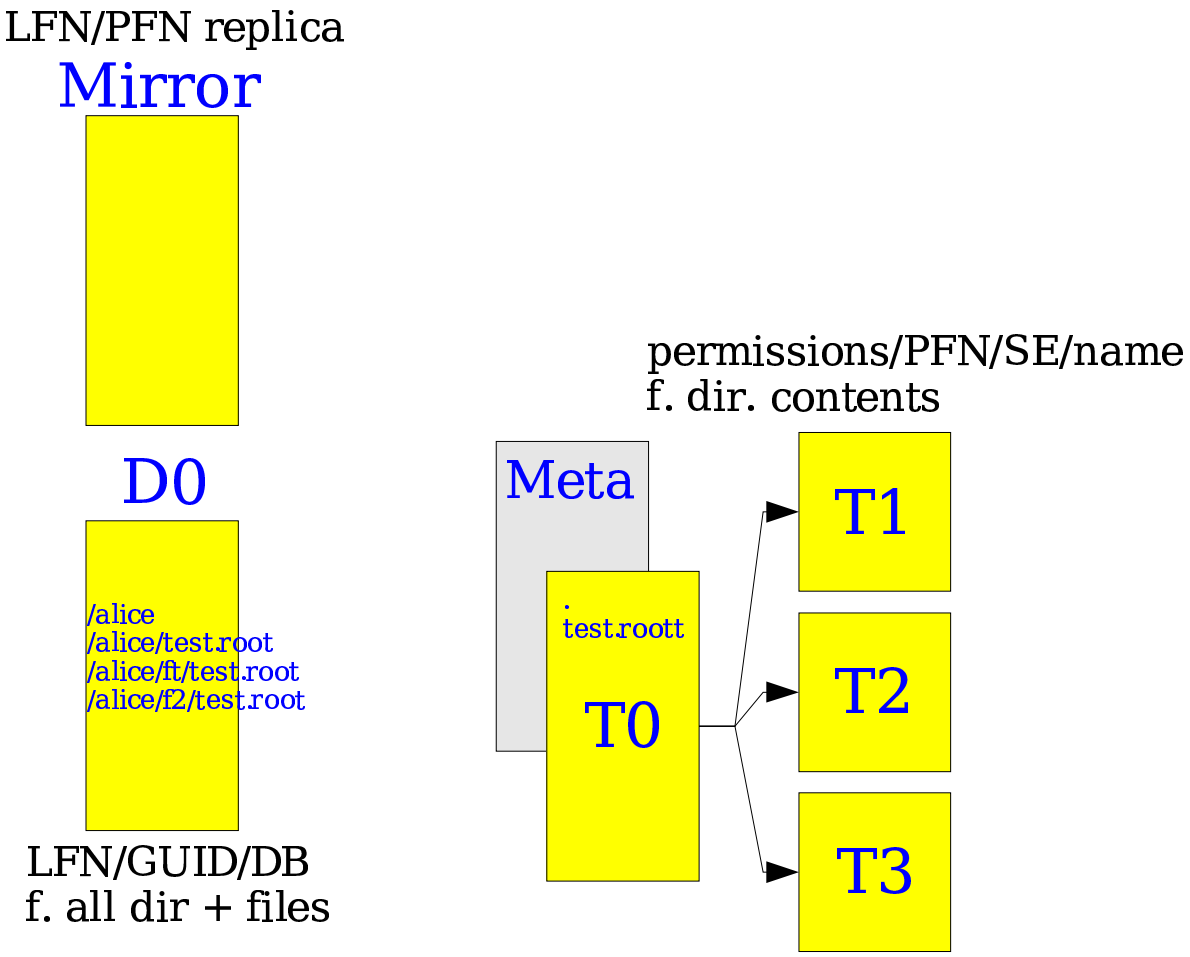
AliEn for Physics Analysis .....

AliEn hits very soon big limitations, if it has to be used for large scale/multi user analysis:

=> the catalogue layout is not appropriate for analysis
=> it does not support collections and partial file indexing as needed by other experiments
=> it does not bundle the DB information together
=> the catalogue layout does not scale (D0 table)

Example: to find 1000 events(26 files) for analysis in a given directory, AliEn needs 1 DB call to find the LFN's, 26.000 DB calls to find the location of the files, which are needed for a 'smart' distributed analysis.
Example: to store 10.000.000 files, the main catalogue table grows to 1 GB => every insert needs to modify this table!

AliEn Catalogue Layout .....



AliEn Catalogue Layout .....

- AliEn Catalogue is 'LFN' oriented
=> difficult to fit with other concepts like in POOL ...

- directory branches can be splitted 'by hand' into distributed databases,
but every table can grow to infinite size

- catalogue is not scalable because of big D0 table with low information contents
(gzip reduces DB tables to 5% of original size => redundant information!)

- every catalogue access needs a scan of the (potentially huge) D0 table

- catalogue contains PFN in host/port/file path format, which can change

- catalogue assigns PFN to SE, which are not really part of the SE

- D0 table makes catalogue replication (parallel catalogues) very difficult!

- replica locations are kept in a seperate table => bad performance for
file location resolving, which is essential for analysis tasks

- catalogue does not allow sharing of directories by different users

- catalogue access is done by direct database connections

Proposal for a new Catalogue Layout .....

Principle:
- no PFN in the file catalogue
- everything is an inode identified by a GUID
- keep tables as small as possible!

How?

New Catalogue Layout .....

Directory Inode Table

/alice/	1
/alice/cern.ch/	2
/alice/software/	3
/alice/user/	4

1
test1.root
test2.root
test3.root
test4.root
test5.root

2
test1.root
test2.root
test3.root
test4.root
test5.root

GUID + Offset + Type + Permissions
+ SE Locations + ...

GUID <=> Dir Inode Table

Organized with GUID
16-bit hash subtables

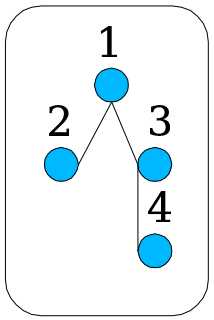
SE Location Table

Alice::CERN::LCG	1
Alice::Torino::LCG	2
Alice::CERN::Lxshare	4
Alice::Prague::PBS	8
.....	

0xf = in all SE's!

Dir Inode Link table

Src	Dst DIN
1	2
1	3
2	2
3	4



DIN Creation

- not every logical directory name produces a new DB table:

Put atleast 1000 files into one table, before creating a new one:

```
mkdir -nInode=1000 /alice/production/0001
```

/alice/production/0001/0001/galice.root
/alice/production/0001/0002/galice.root
/alice/production/0001/0003/galice.root
....
/alice/production/0001/<N>/galice.root

}

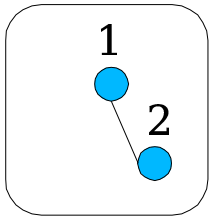
1
0001/galice.root
0002/galice.root
0003/galice.root
<N>/galice.root

How do we avoid, that this table grows forever?

Automatic Table splitting

Example splitting limit 1000 files

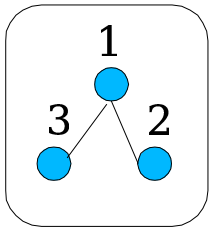
Src	Dst DIN
1	2



-DIN 1+2 contain 999 files
-now the user adds 1 more file

Dir Inode Link table

Src	Dst DIN
1	2:3



-to list all files under DIN 1, the files in DIN 2+3 are listed

Last DIN used for new inserts

Parallel File Catalogues

Some statements:

- A centralized file catalogue gives only good performance to people who are close (RTT US-Europe)
- File catalogues are dominated by READ operations
- After 1st generation entries are quite 'stable'
- A huge D0 table disables parallel file catlogues

Master/Slave File Catalogues

Directory Inode Table Master

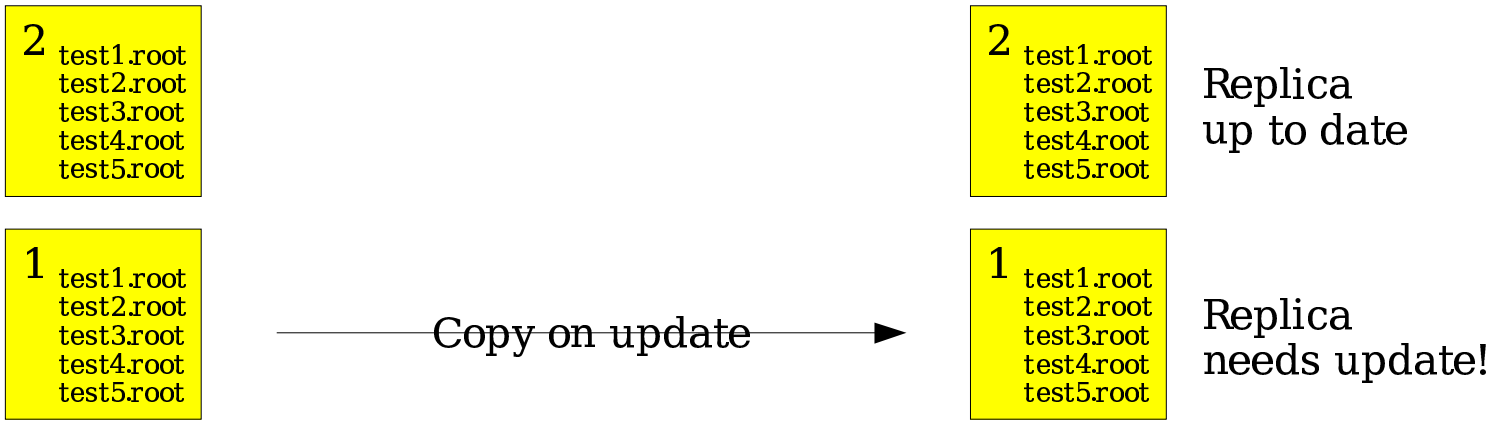
DIN-M

LDN	Dnode	Mod.Time
/alice/	1	12342321
/alice/cern.ch/	2	12342322
/alice/software/	3	12342334
/alice/user/	4	12342335

Directory Inode Table Slave

DIN-S

LDN	Dnode	Mod.Time
/alice/	1	12342200
/alice/cern.ch/	2	12342322
/alice/software/	3	12342334
/alice/user/	4	12342335



Two operation modes:

- lazy: DIN-S entry is synchronized every <x> sec. with DIN-M
- realtime: DIN-S is synchronized with every request to DIN-M

Example of usage: run a slave catalogue in America with fast response!

Meta Data Catalogue

The Meta Data Catalogue can be seperated from the file Catalogue! How can this be fast?

File Catalogue

Directory Inode Table

/alice/	1
/alice/cern.ch/	2
/alice/software/	3
/alice/user/	4

1
test1.root
test2.root
test3.root
test4.root
test5.root

2
test1.root
test2.root
test3.root
test4.root
test5.root

Meta Data Catalogue

TN-1

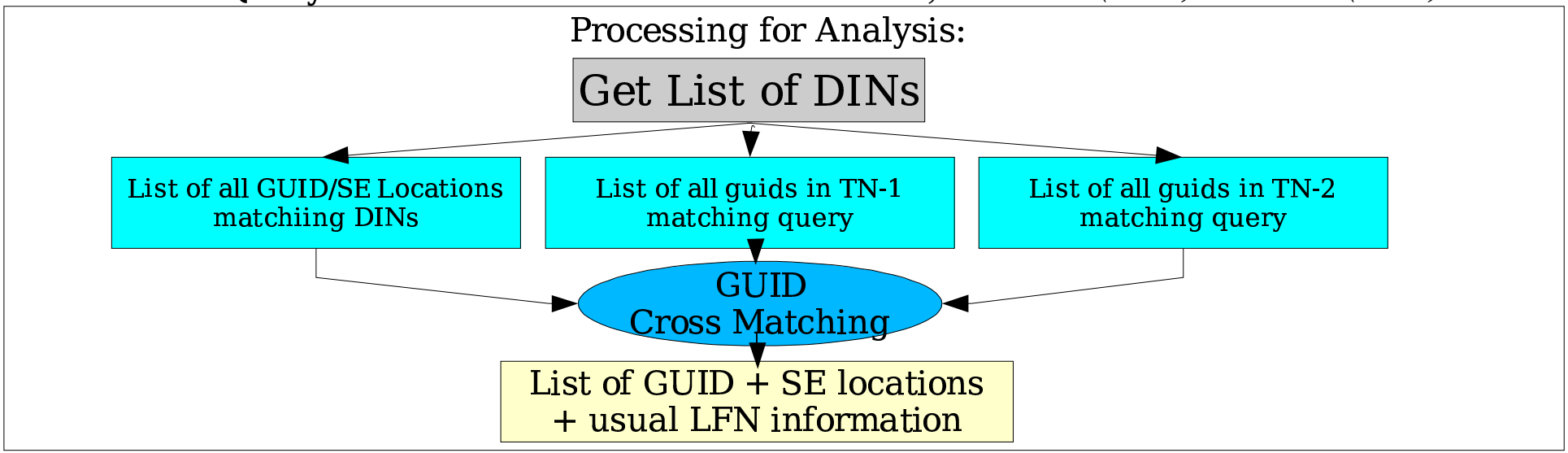
1 test1.root	x=10	y=20	<guid1>
test2.root	x=11	y=21	<guid2>
test3.root	x=12	y=21	<guid3>
test4.root	x=13	y=24	<guid4>
test5.root	x=14	y=15	<guid5>

TN-2

1 test1.root	z=10	a=10	<guid1>
test2.root	z=11	a=11	<guid2>
test3.root	z=12	a=11	<guid3>
test4.root	z=13	a=14	<guid4>
test5.root	z=14	a=15	<guid5>

Query Scheme: *Find all under Dirnode <x>, with TN-1(x>10)and TN-2 (z>10)*

Processing for Analysis:



VO Catalogue Sharing

Directory Inode Table

/alice/	1
/alice/cern.ch/	2
/alice/software/	3
/alice/user/	4

/LHCb	1
/LHCb/cern.ch/	2
/LHCb/software/	3
/LHCb/user/	4

DB-Mount Table

/alice/	DB type:host:port
/alice/test	DB type:host:port
/LHCb	DB type:host:port
/atlas	DB type:host:port
/cms	DB type host:port

Remark: don't query this table
for every catalogue
access, cache it on a daily base f.e.

New Catalogue Layout .....

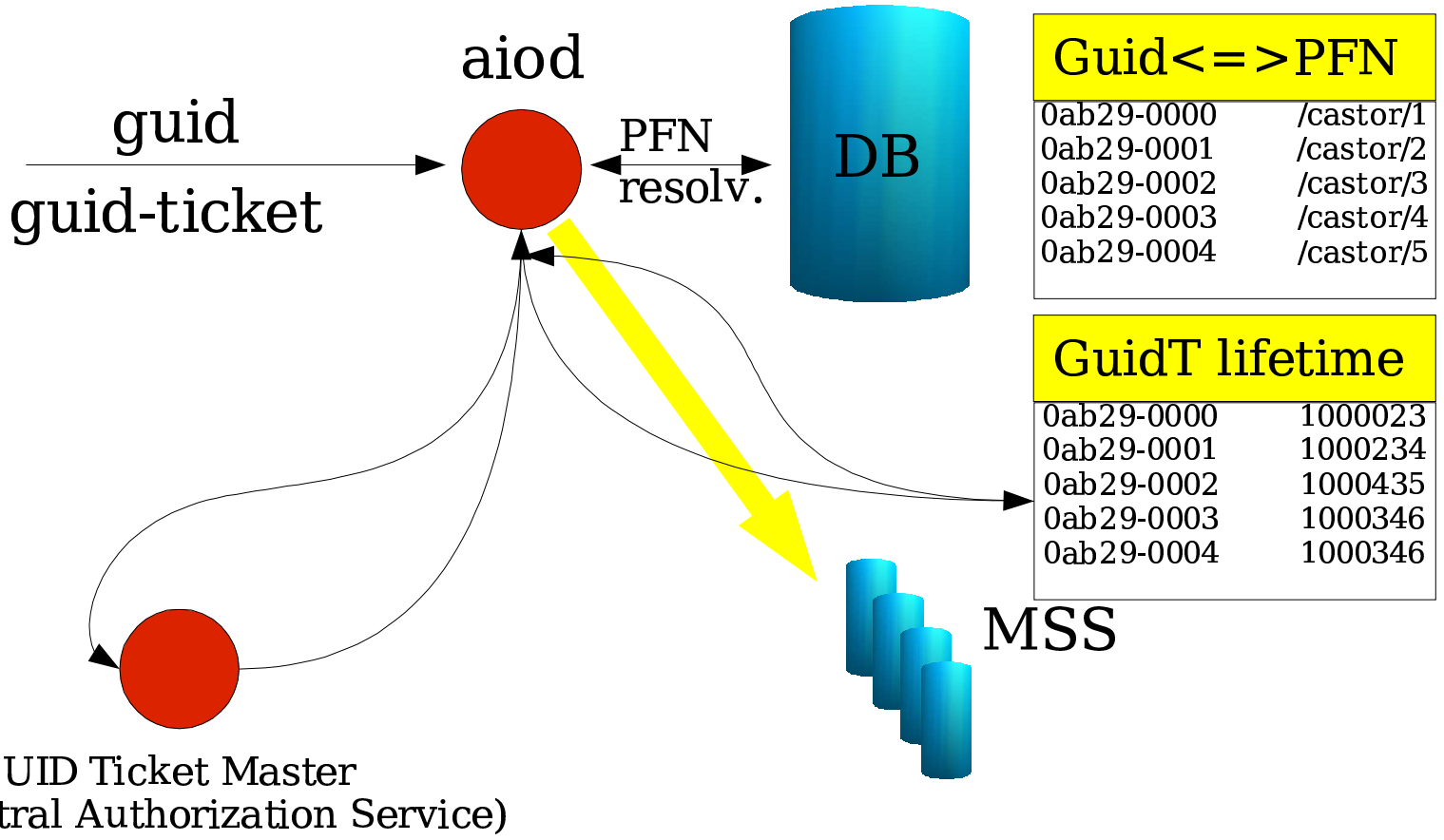
Summary

- DIN table replaces former D0 table
  - Much smaller (only directories)
- GUID<=>DIN table
  - Allows to find all GUID references in the file catalogue (POOL compliance)
- GID,UID, permissions, SE locations are written in a 'binary' format to reduce the table size
- A DIN<=>DIN tree table allows to find very fast all subdirectories for fast querying
- Logical directory names can be compounded by one DB table
- Logical directories can be split over several DB tables
- Collections can be identiefied by a normal DIN containing GUID entries

?? Where are the PFNs ??

New Sercure Storage Element Design

- all file access is identified by a GUID and a GUID-ticket
- all files are owned by a SE user
- jobs can read by default files only through the aiod, not directly with f.e. rfio



Low level TCP service *SOAP too slow, too much XML/Text overhead

File access is not connected with any PERL/SOAP lite => fast!

New Sercure Storage Element Design

Concept of 'public' files:

- The autorization scheme is very secure, if the files are only readable by the aiod use, but produces a file access bottle neck with the aiods

- For HEP use cases, it is not possible to route all file accesses through aiod's

- Add a 'public' flag to GUIDs which don't need high security:
  - Files are stored as readable for group/others
  - Files access is rerouted from the aiod to direct MSS access

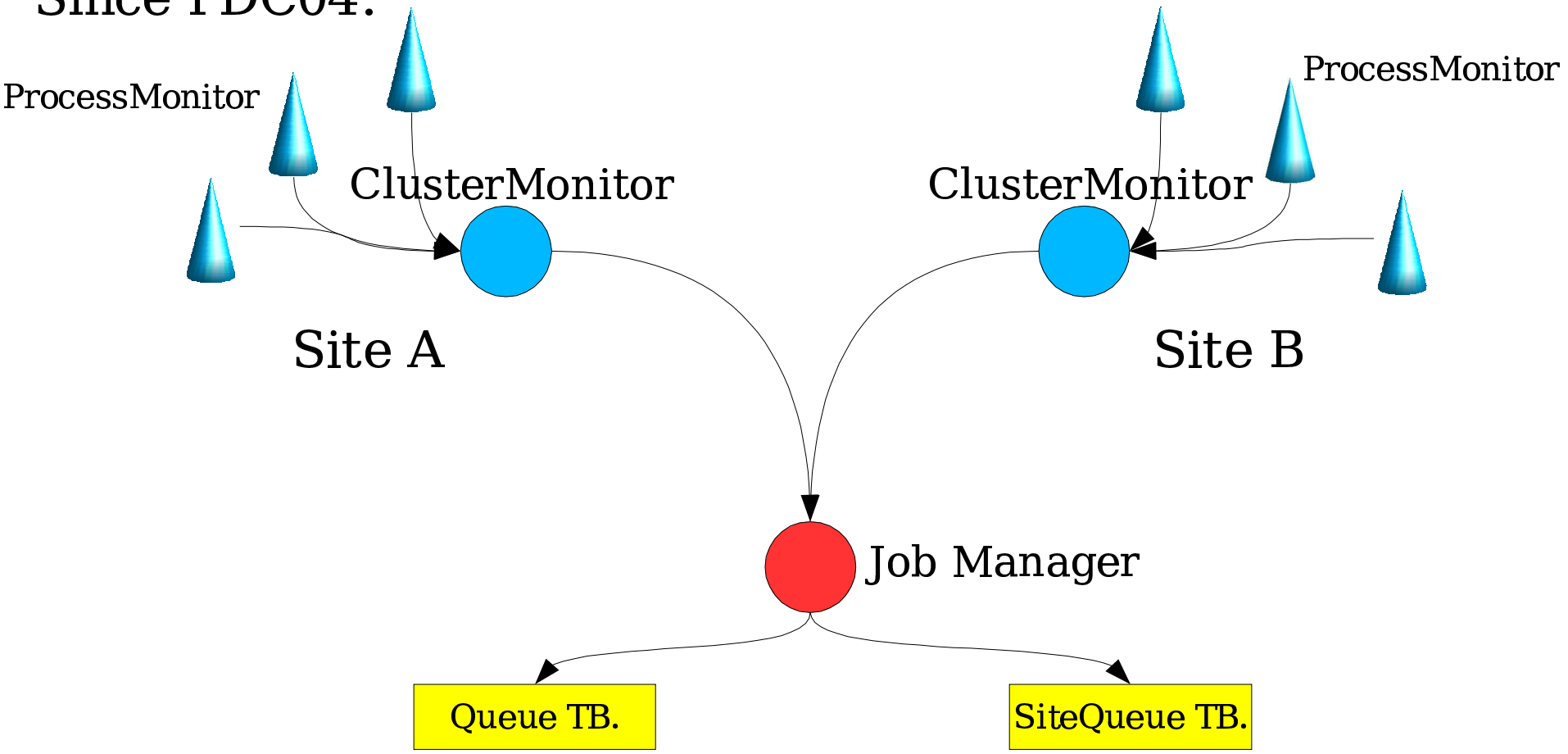
In general:

- support only one alien file access method, implement all others as plugins to aiod => dump all SOAP, HTTP, File, MSS etc.

- Implement a 'dummy' SE to do GUID<=>PFN translation of web adresses a.s.o

PDC04 Queue System Layout

Since PDC04:



PDC04 Queue System Layout

Pro:

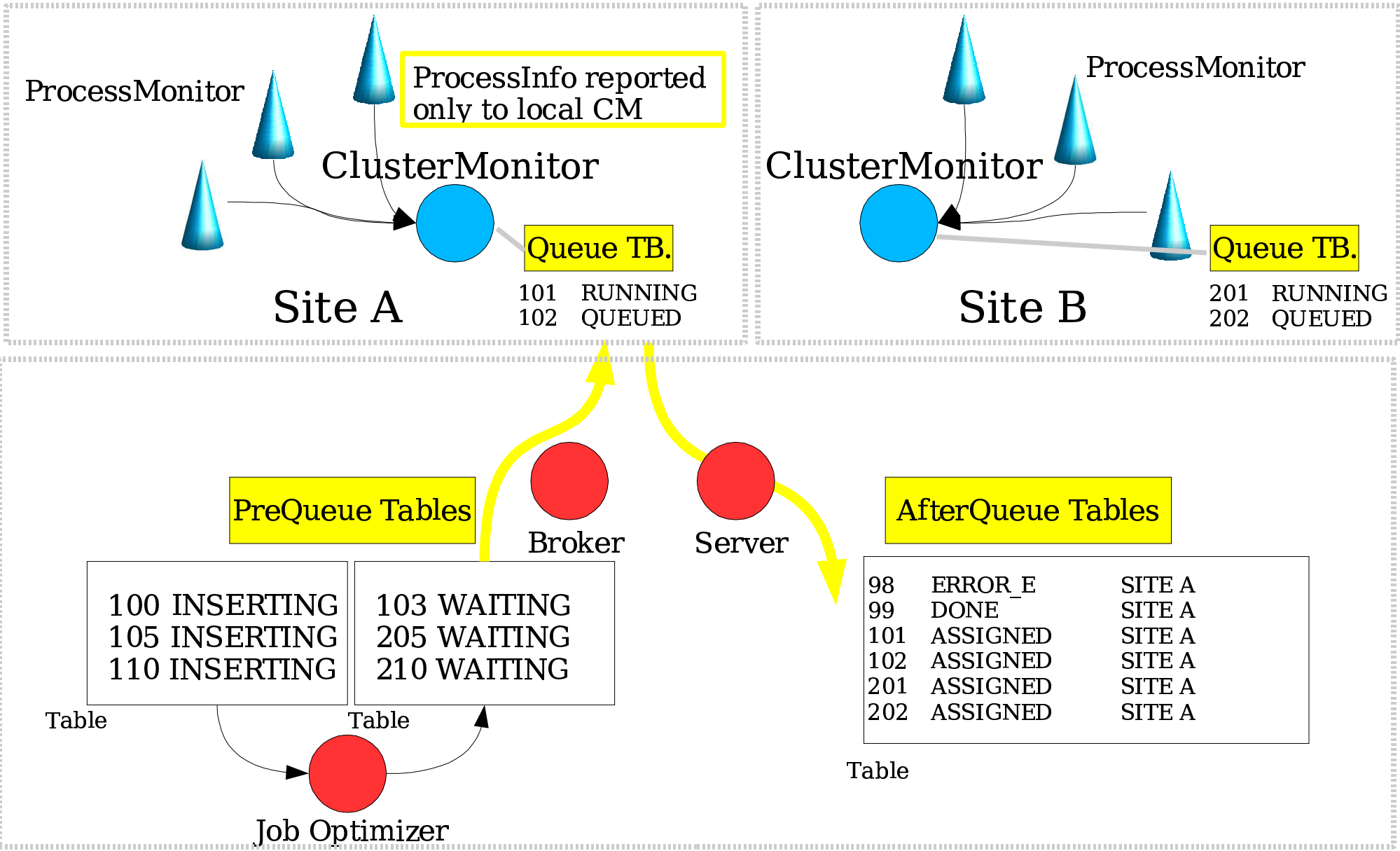
- Removed synchronization problem between ProcessMonitor DB and central DB
- Allows easy system view and opening/locking of certain sites

Contra:

- Does not scale because:
  - Number of ProcessInfo messages per second is limited in the central machine/DB
  - The QUEUE DB becomes huge with time and every status/ProcessInfo needs a manipulation of this table
  - Layout is highly centralized!

How to remove the disadvantages?

Proposal for new Queue System Layout



Proposal for new Queue System Layout

- Jobs with status before 'QUEUED' stay in the central DB tables:
  - Table INSERTING
  - Table QUEUED
    - JobOptimizer + Broker operate on small tables!

- Jobs with are picked up by a SITE are copied into the local site
  - QUEUE DB and to the afterQUEUE DB with status ASSIGEND
  - All job status before DONE or ERROR are reported only locally to the ClusterMonitor, also the process information/heart beat

  - Jobs changing to DONE or ERROR are removed from the local QUEUE DB and overwrite the ASSIGNED entry in the afterQUEUE DB

- The afterQUEUE DB is divided into tables f.e.

  T10000 f. queueId 1-9999
  T20000 f. queueId 10000-19999 a.s.o.

Implications by a new Queue System Layout

•The mechanism of the site queue table can stay the same, since it just counts the number of jobs in a certain state. The feature of locking a site can be kept.

•If one queries the status of a queued/running job, the master server has to contact all the ClusterMonitor for a report
=> easy, can be done in parallel, small time overhead.

•Each site needs to run a 'small' DB. The QUEUE table will contain as many entries as the maximum number of queued or running jobs

•The central DB tables can be kept small and the splitting of INSERTING and QUEUED jobs allows to do a faster job machting by the Broker.

•The ProcessMonitor's report only local, no central bottleneck anymore. The reports should be done with a mini TCP protocol, since SOAP is slow and many instances are expensive in memory (won't work sufficiently with huge sites (>1000 jobs).

Broker Upgrade for Multi-User Support

- To do multi-user job scheduling, the broker already sorts jobs for the matching by a priority value, which is equal for all jobs at the moment.

- One needs to add another DB table, which assigns priorities and job limitations to users. From this DB table one has to extract priority values for jobs owned by individual user.

- With every queued job, the priority values have to be recomputed

- The algorithm for this priority value can be very simple:
  - Allow a maximum number of jobs per user
  - Take into account the history of processed jobs per user
  - .....

- An upgrade of the Broker scheme is mandatory for multi-user support!

Service Communication

•SOAP Server have shown a good reliability and scalability in AliEn, but the PERL implementation is memory extensive and slow compared to specialized protocols implemented with plain TCP

•Central Services should access the underlying DB directly (as most of them do right now)

•Autohrization services (as proposed f. the SE) and Process Information should be handled by specialized NON-SOAP services

•To make the SOAP communication save, each service should use GUID tickets (SE) with a certain validity, which is included in the SOAP header. SOAP over SSL reduces already by a factor of three. Once authorized GUID-tickets can be cached for the validity of the ticket without reauthorization. (Remember a GUID contains also the production time and IP).
•The authentication service produces GUID tickets with standard auth. Methods (SSL/GRID certificates, AFS password etc.)

Catalogue Access

Since catalogue access follows the pattern,
<rare requests> => <a lot of output>

(besides file registration), it can be sufficient to implement the Catalogue Interface in a SOAP service (factory) without direct DB connections.

- The complete catalogue can be owned by one user, so the Catalogue Service can do all the desired actions without using another service (Authen creates now new tables=> slow).

- The system scales (# of Proxy problem), because 10 SOAP server can handle 500 users easily, if a ticket based authentication scheme is used for every SOAP call

- one should consider to use SQLR as a catalogue interface,
it has nice caching features and the code is faster and much smaller than PERL, which plays a role with 100 of users. Maybe also gSOAP.

PS: I made already a test implementation for catalogue access with SQLR

General Service Factory

- Implement a general Service Factory, which is able to start, stop and install all existing DB and services on demand.
- Every Factory request (tickets) has to be authorized with a central Authorization Service
- LDAP parameters are extracted by the service factory, but can be overwritten with local settings by site administrators

CE Jobs

- CE has to be run as root and individual users have to be mapped to different CE users, otherwise the system will never provide individual user security.

File Replication

- Use also 'active' file replication – triggered by a running job. This makes the system more decentralized.

Summary:

-If we just take AliEn as it is and continue, what we have, we will run against many walls in the near future without changing the basic architecture later.

-I think, it is worthwhile to reimplement some core features in AliEn to build a professional system which can be used in the future.

-I would focus on this basic features before starting external ends like test environments,
API, web portals etc - considering the size of the core code, this should not be a long procedure. Otherwise you will have to redo the same work again and again in the future, when you change the low-level functionality.

-The development should be decoupled from the PDC04 code.