Functional description

Detailed view of the system

Status and features

Giuseppe Lo Presti, Olof Bärring
CERN / IT

Castor Readiness Review – June 2006

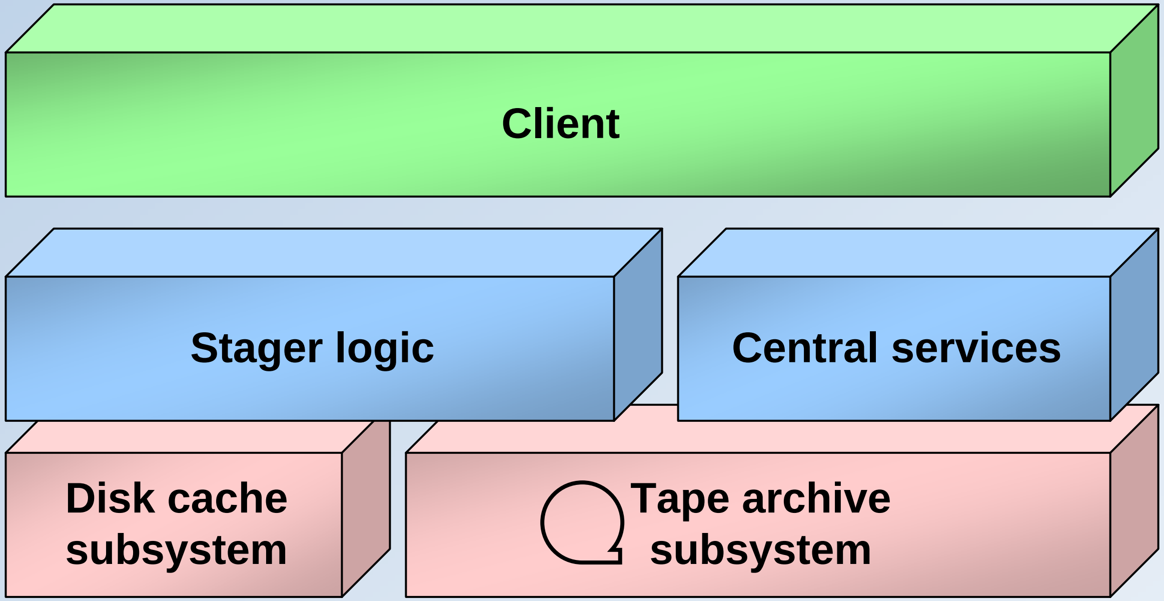
Outline

- Detailed view of the architecture
  - Lifecycle of a GET and a PUT request
- Description and status of the components
  - Main daemons
  - Diskserver related
  - Central services
  - Tape related
- Tape migration and recall
  - Workflow details

Outline

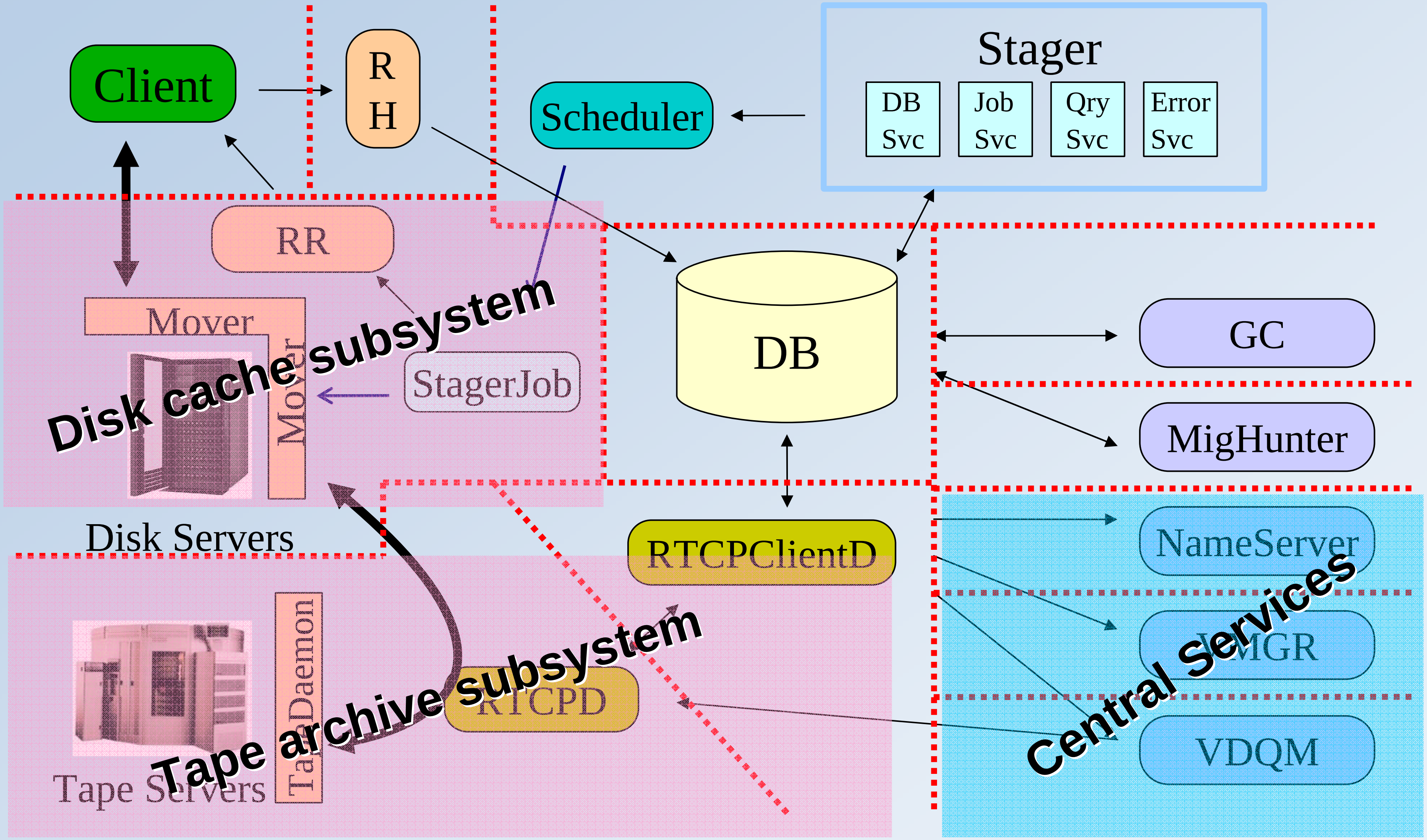
- Detailed view of the architecture
  - Lifecycle of a GET and a PUT request
- Description and status of the components
  - Main daemons
  - Diskserver related
  - Central services
  - Tape related
- Tape migration and recall
  - Workflow details

Castor 2 Architecture



From the "simple" view …

Castor 2 Architecture



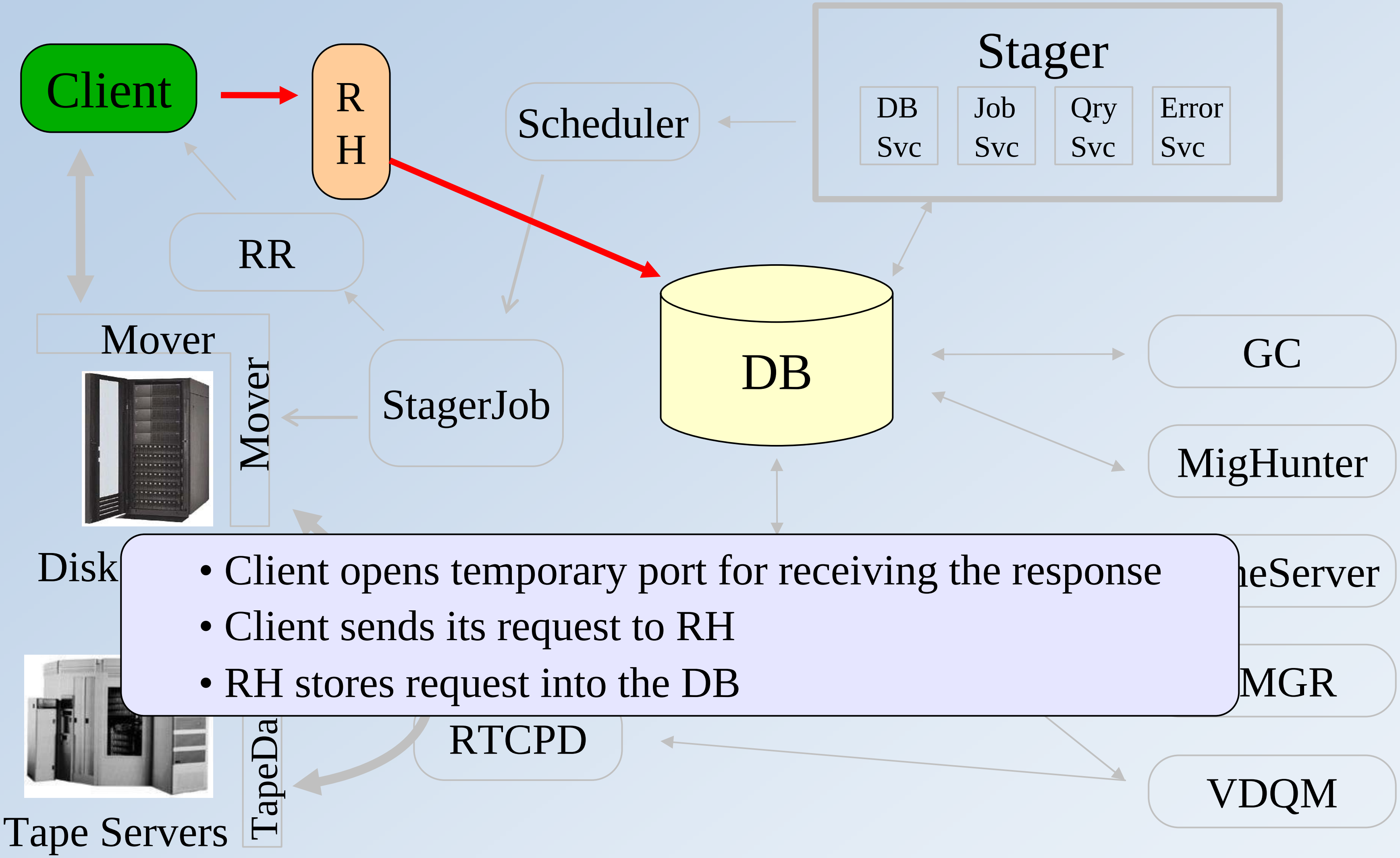
… to a more detailed one

Lifecycle of a GET + recall

1. Client connects to the RH
2. RH stores the request into the db
3. Stager polls the db and checks for file availability
4. If the file is not available, the recall process is activated
5. Once the file is available, stager asks the scheduler to schedule the access to the file
6. Client gets a callback and can initiate the transfer

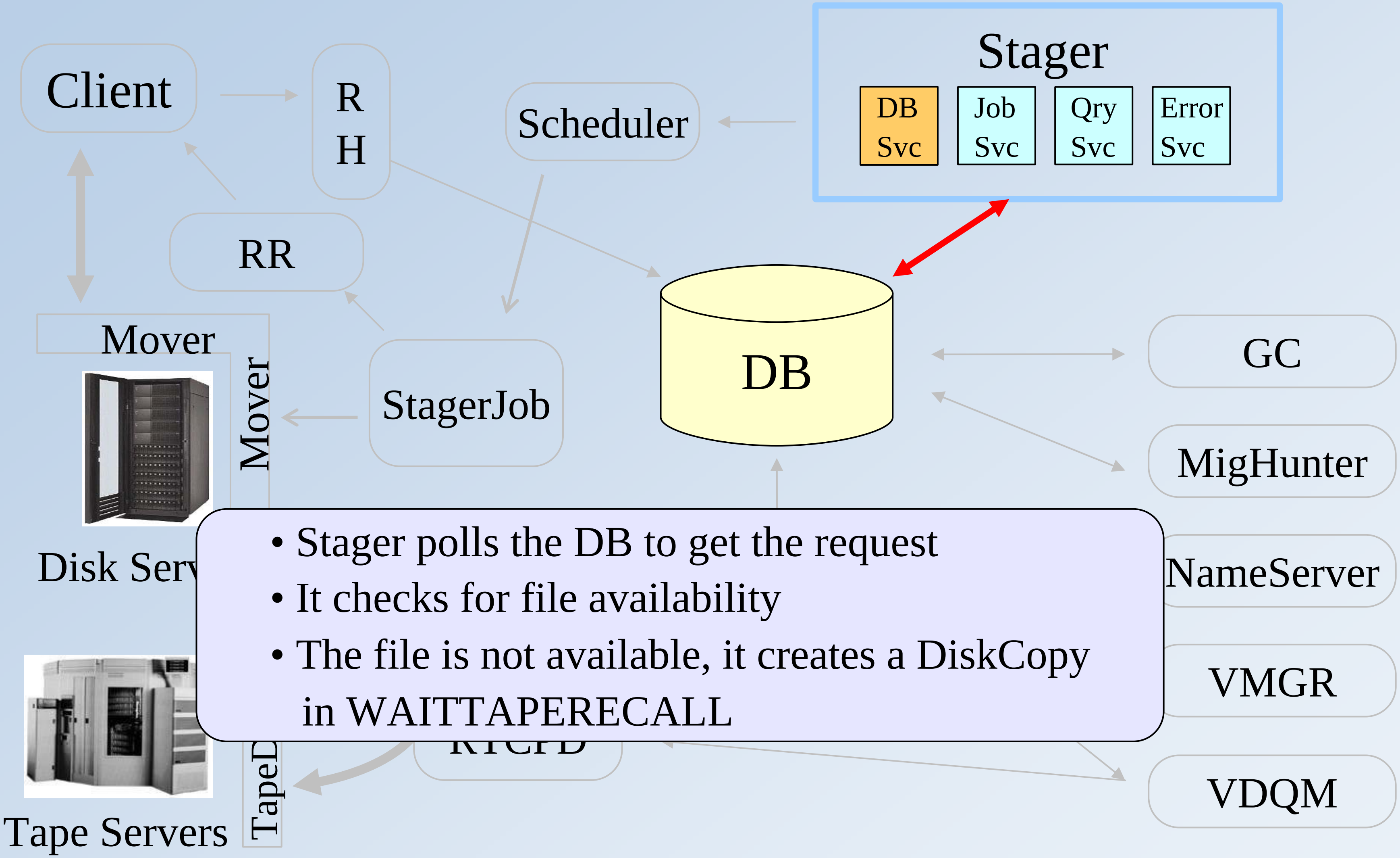
- The commandline is stager_get

stager_get (1)



- Client opens temporary port for receiving the response
- Client sends its request to RH
- RH stores request into the DB

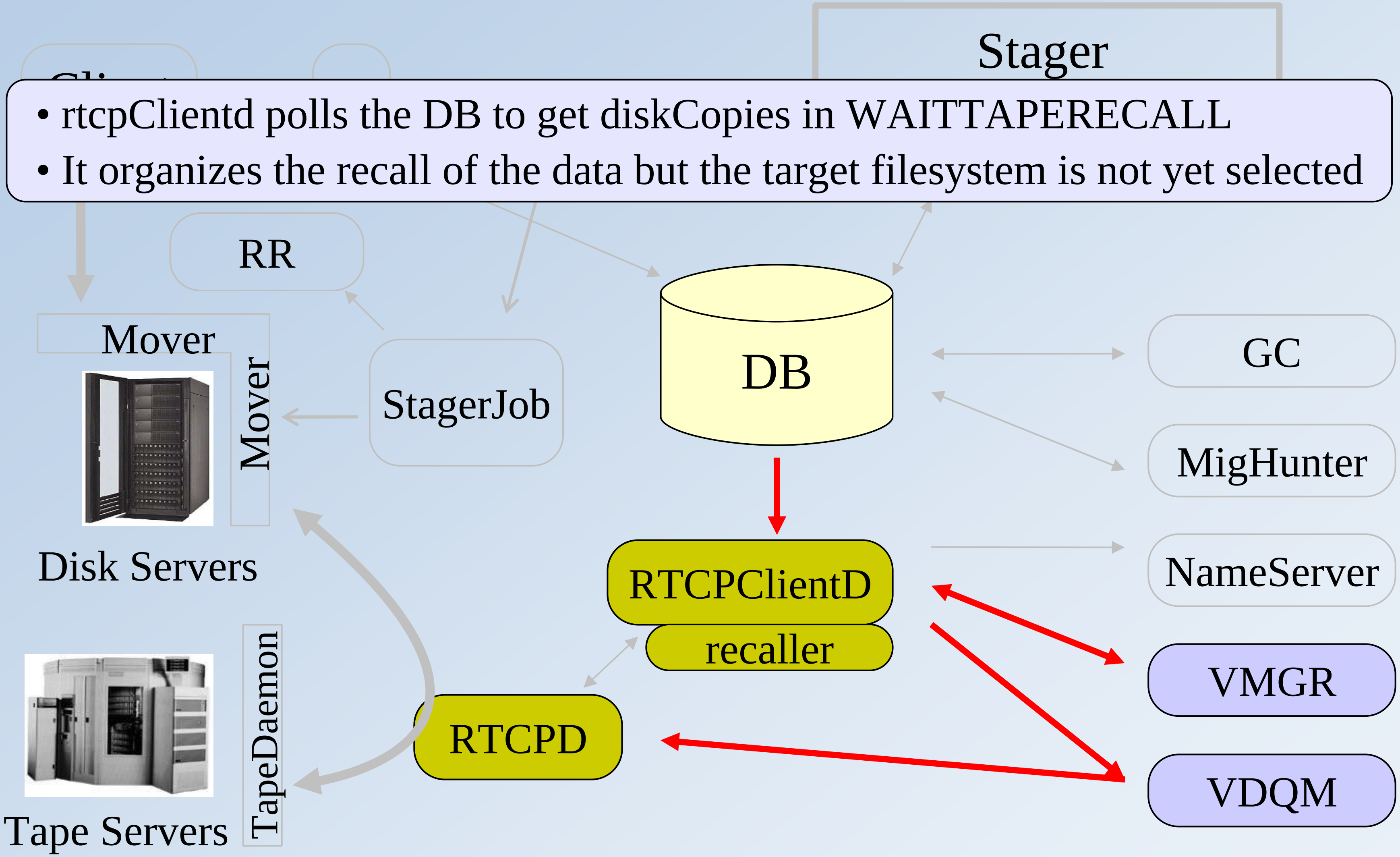
stager_get (2)



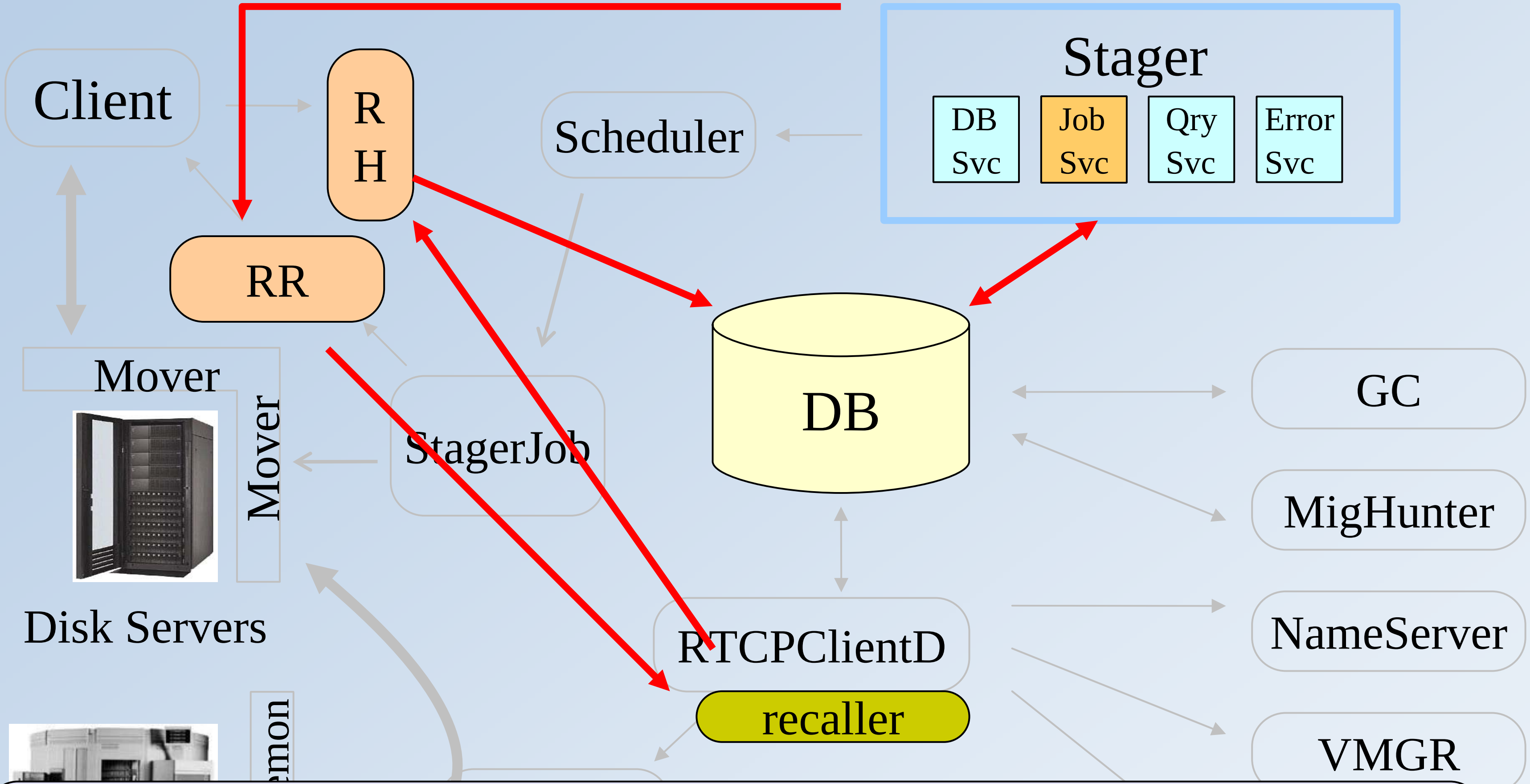
- Stager polls the DB to get the request
- It checks for file availability
- The file is not available, it creates a DiskCopy in WAITTAPERECALL

stager_get (3)

- rtcpClientd polls the DB to get diskCopies in WAITTAPERECALL
- It organizes the recall of the data but the target filesystem is not yet selected



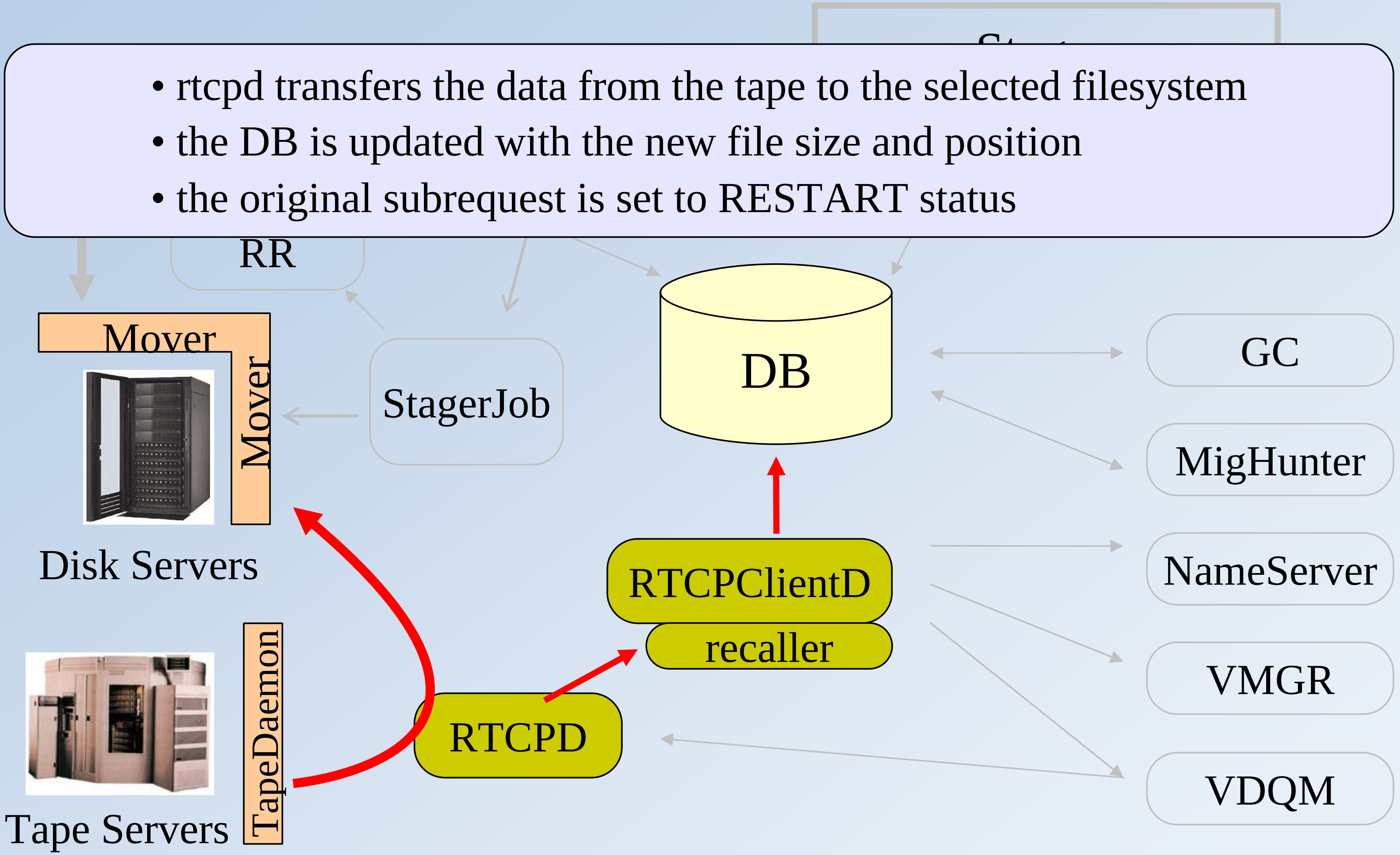
stager_get (4)



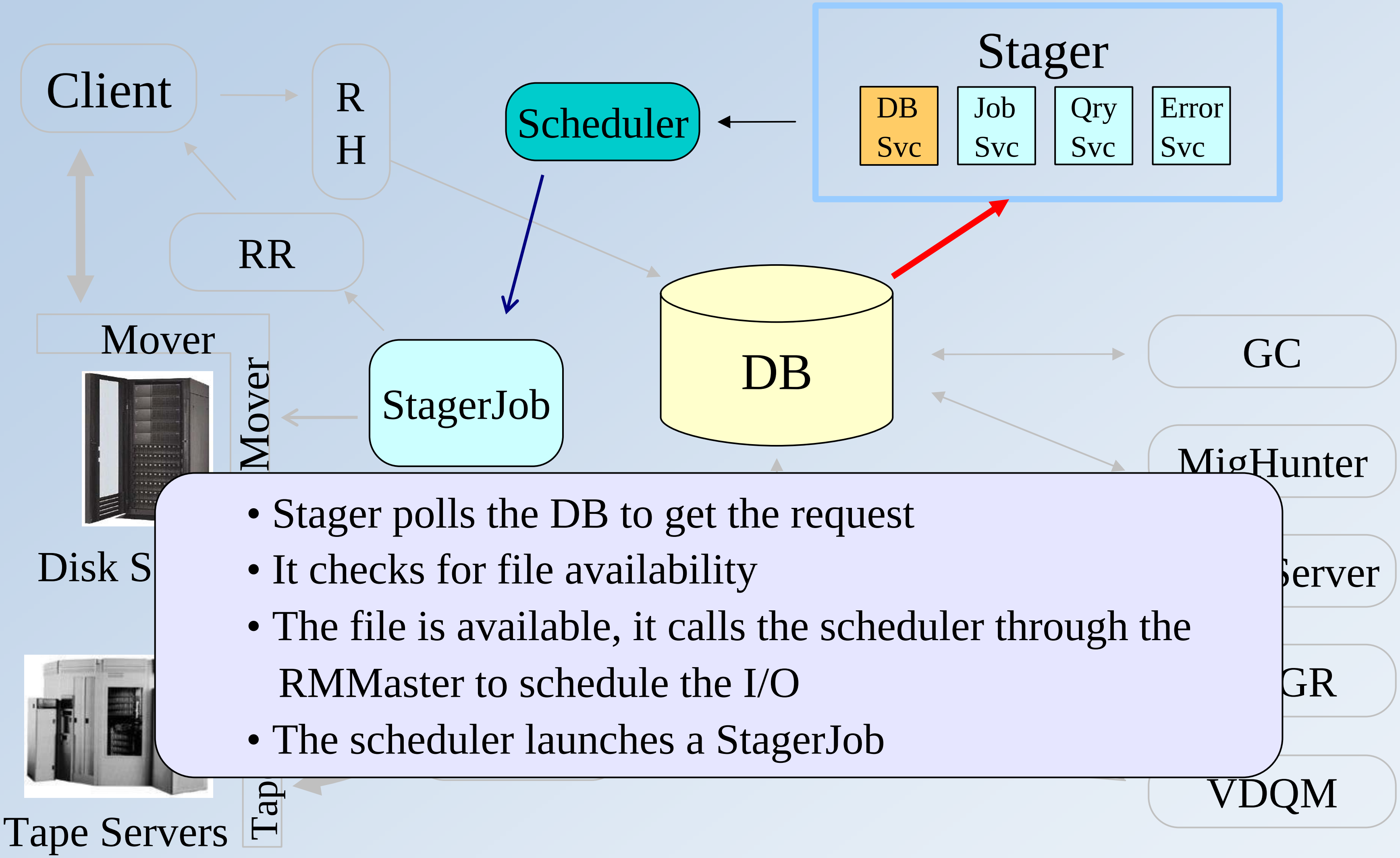
- recaller sends a request to the stager in order to know where to put the file
- the request goes through the usual way: Request Handler, DB, stager (job service), Request Replier

stager_get (5)

- rtcpd transfers the data from the tape to the selected filesystem
- the DB is updated with the new file size and position
- the original subrequest is set to RESTART status

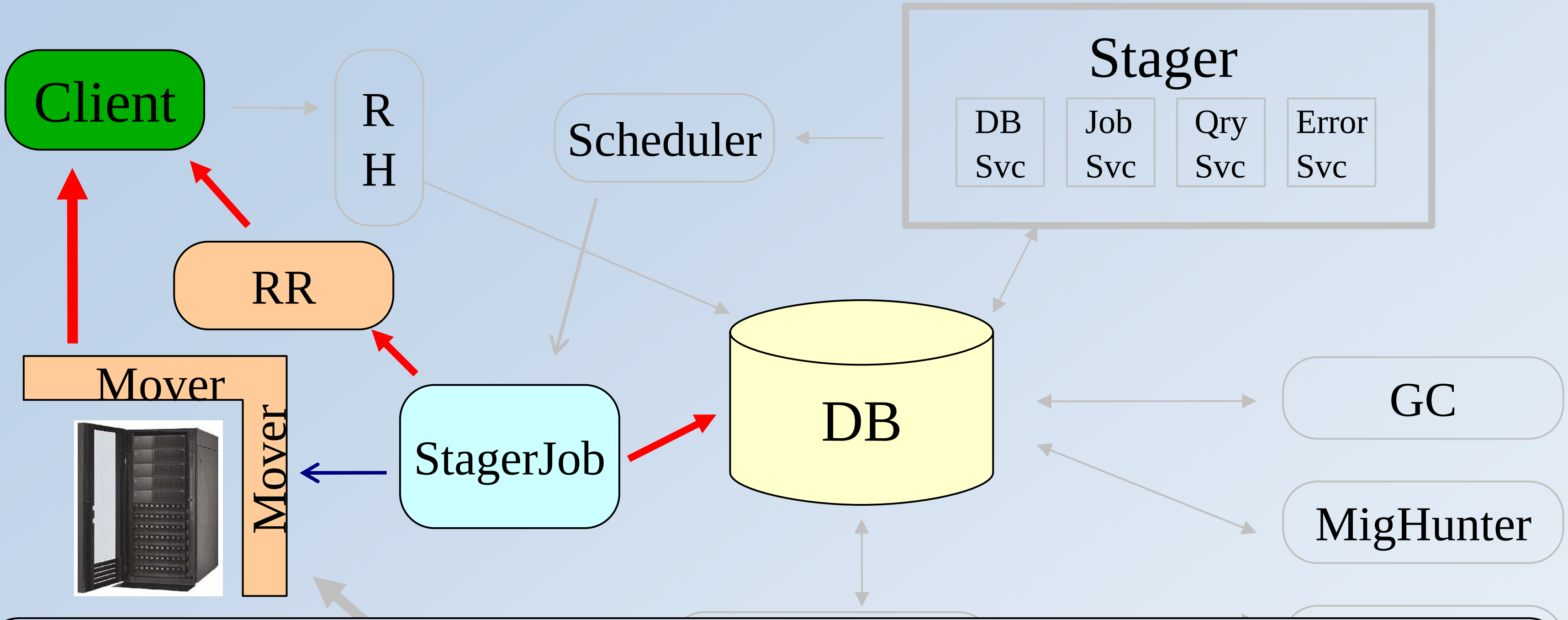


stager_get (6)



- Stager polls the DB to get the request
- It checks for file availability
- The file is available, it calls the scheduler through the RMMaster to schedule the I/O
- The scheduler launches a StagerJob

stager_get (7)



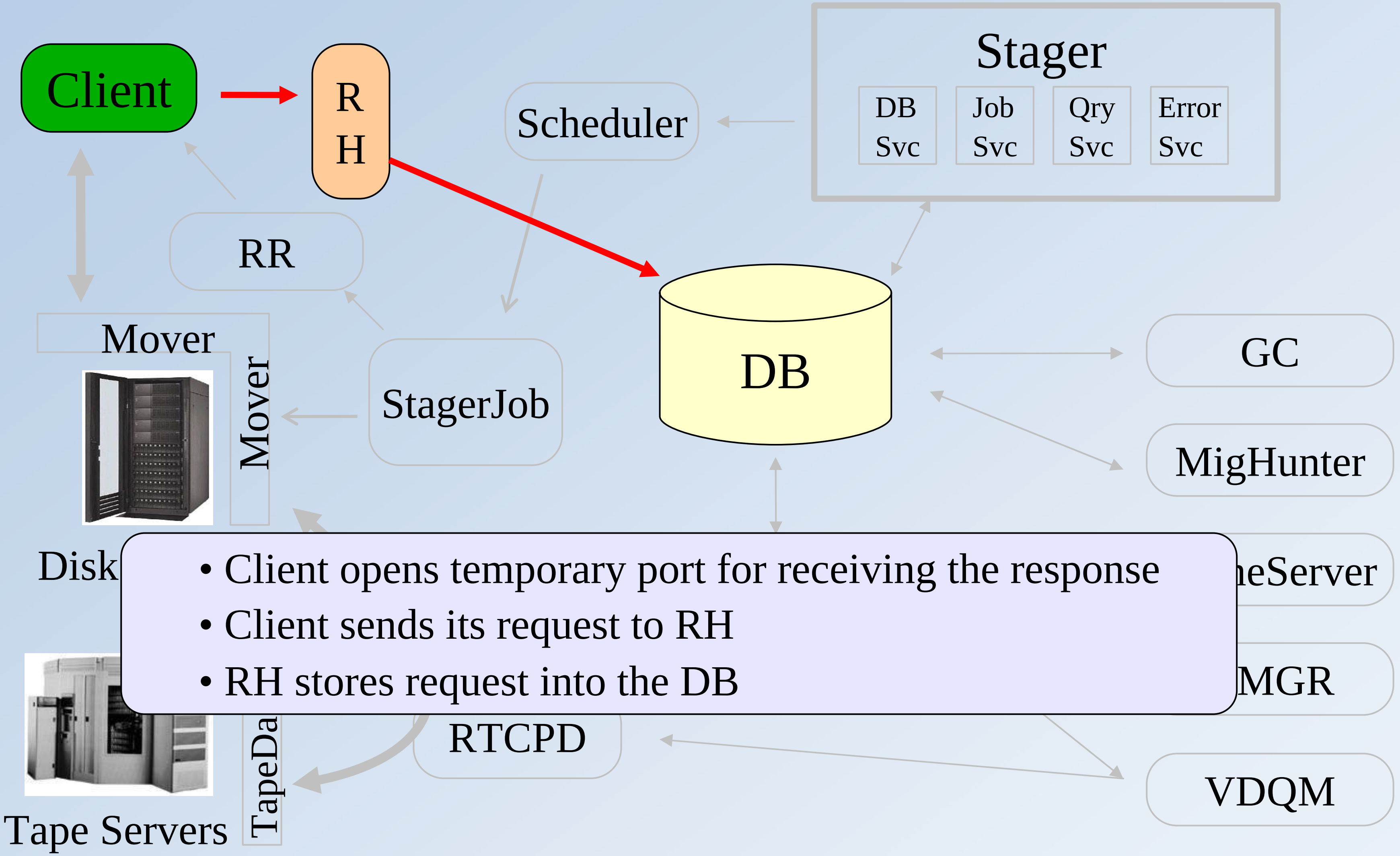
- the StagerJob launches the right mover corresponding to the client request (note that the scheduler takes available movers into account)
- it answers to the client, giving to it the machine and port where to contact the mover
- data is transfered
- DB is updated

Lifecycle of a PUT + migration

1. Client connects to the RH
2. RH stores the request into the db
3. Stager polls the db and looks for a candidate filesystem for the transfer
4. Client gets a callback and can initiate the transfer
5. After the transfer is completed, migration to tape is performed

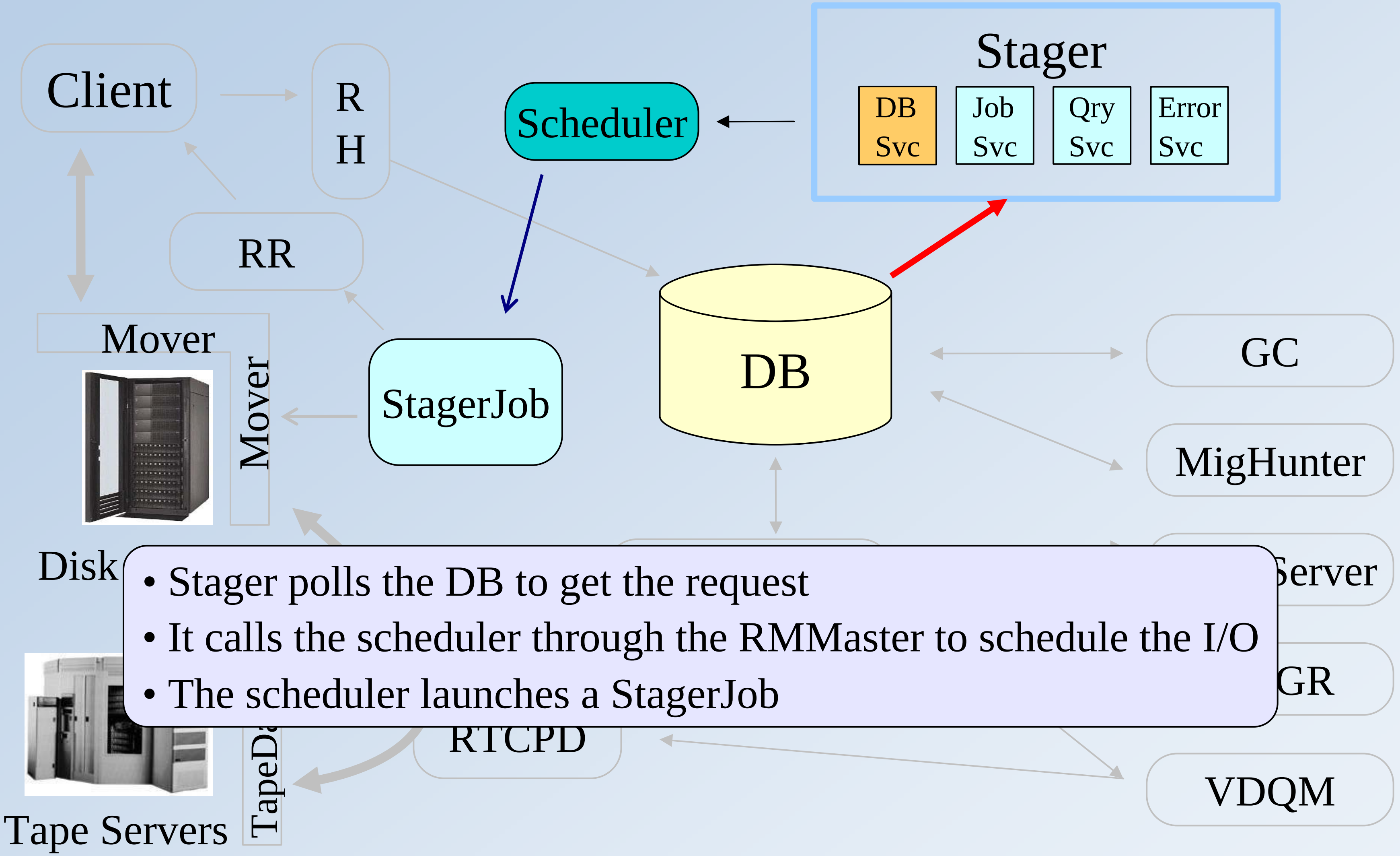
- The commandline is stager_put

stager_put (1)



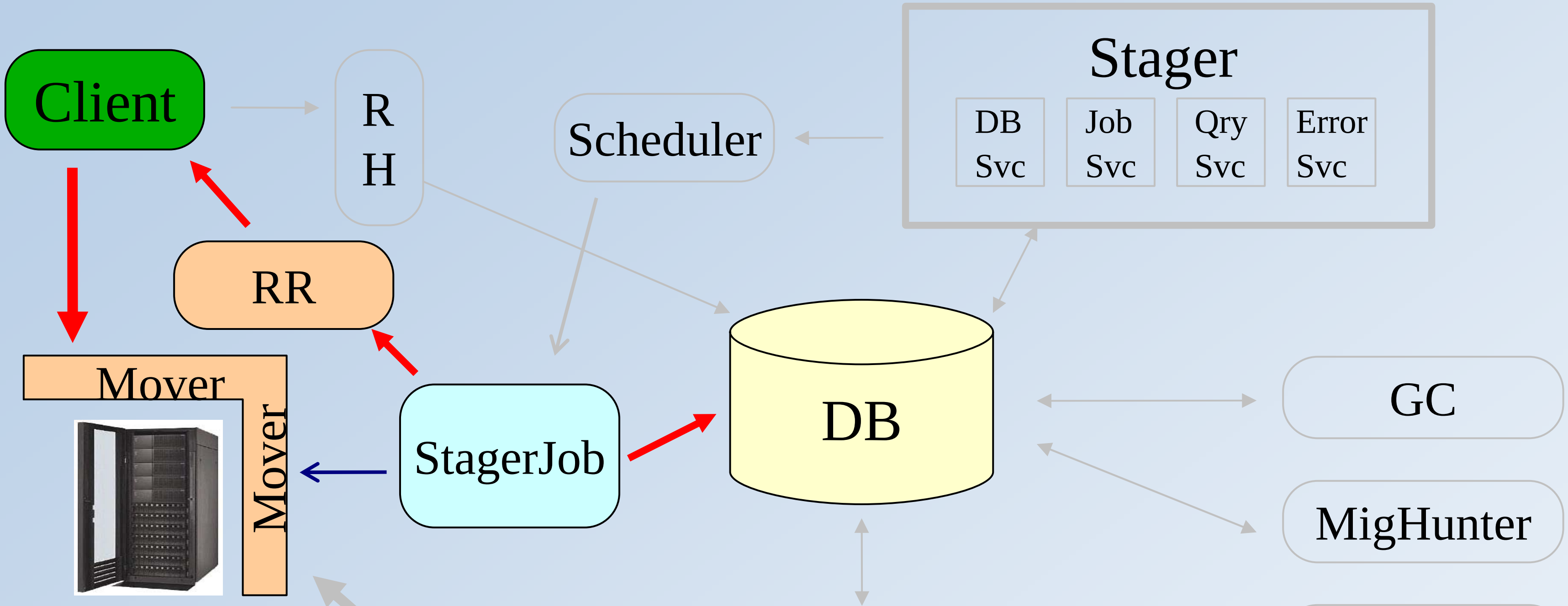
- Client opens temporary port for receiving the response
- Client sends its request to RH
- RH stores request into the DB

stager_put (2)



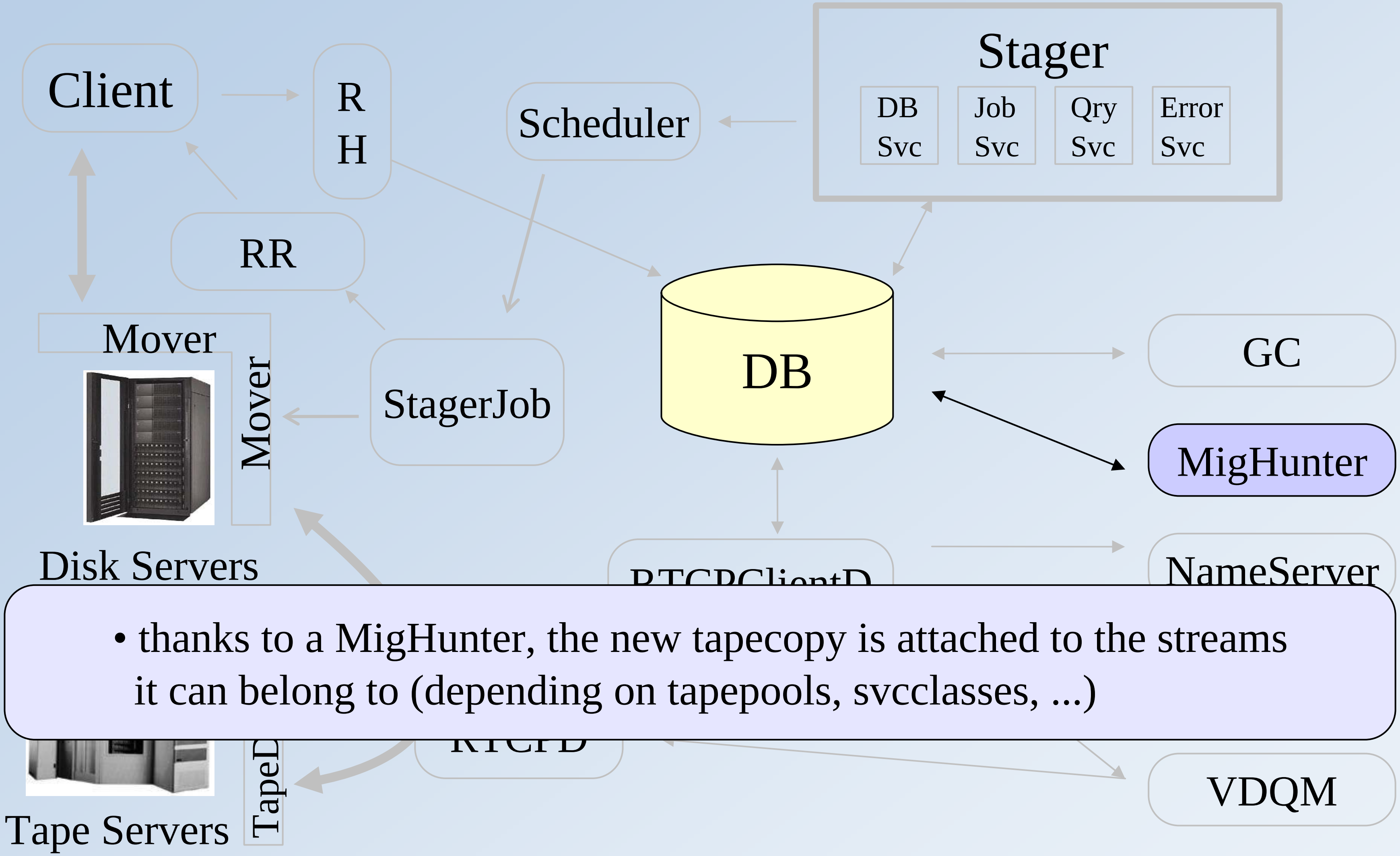
- Stager polls the DB to get the request
- It calls the scheduler through the RMMaster to schedule the I/O
- The scheduler launches a StagerJob

stager_put (3)



- the StagerJob launches the right mover corresponding to the client request (note that the scheduler takes available movers into account)
- it answers to the client, giving to it the machine and port where to contact the mover
- data is transferred
- DB is updated with the file size and the diskcopy is set in CANBEMIGR and one or many TapeCopies are created

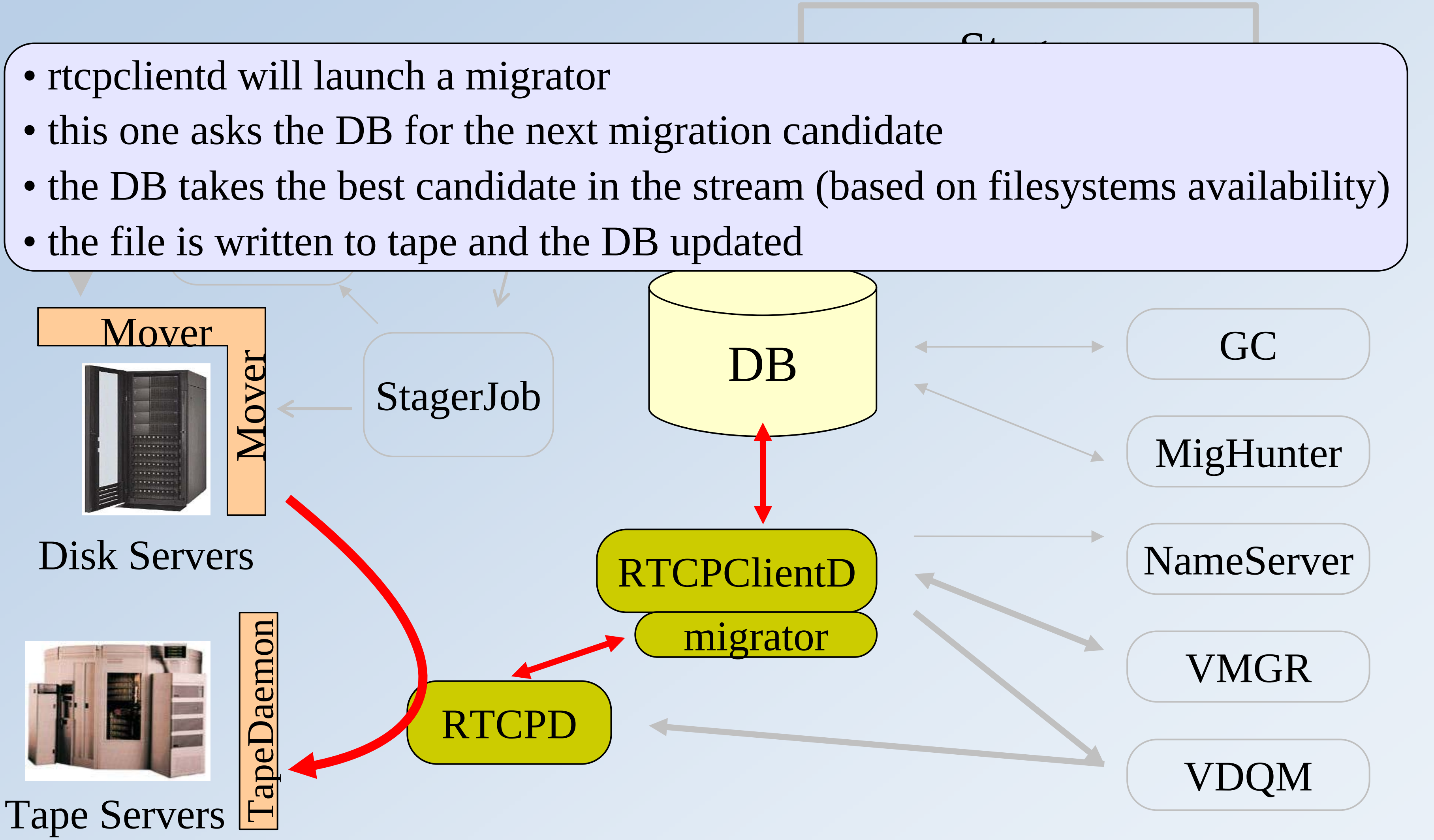
stager_put (4)



- thanks to a MigHunter, the new tapecopy is attached to the streams it can belong to (depending on tapepools, svcclasses, ...)

stager_put (5)

- rtcpclientd will launch a migrator
- this one asks the DB for the next migration candidate
- the DB takes the best candidate in the stream (based on filesystems availability)
- the file is written to tape and the DB updated



Mover

Mover

Disk Servers

StagerJob

DB

GC

MigHunter

NameServer

RTCPClientD

migrator

VMGR

RTCPD

VDQM

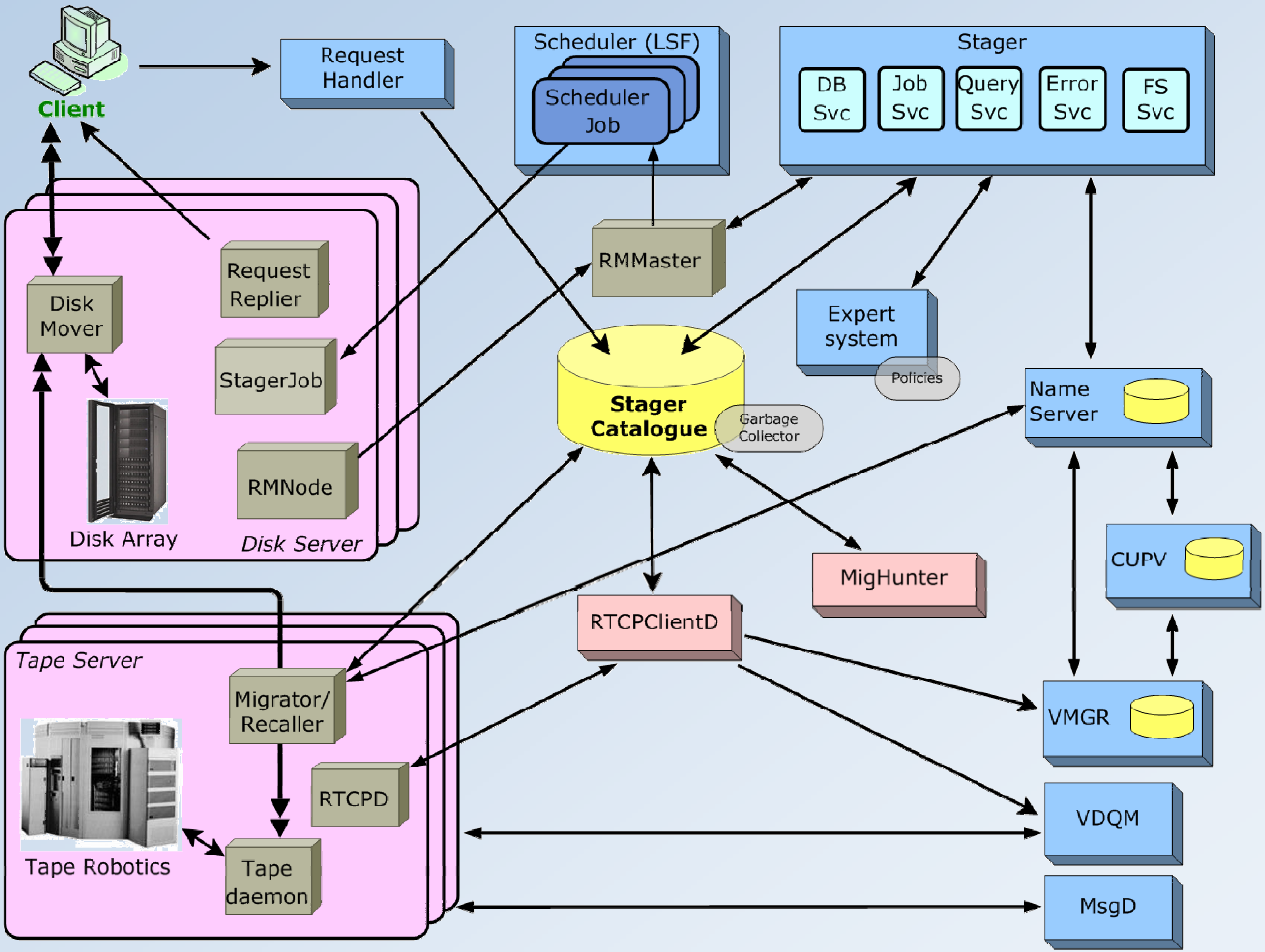
TapeDaemon

Tape Servers

Outline

- Detailed view of the architecture
  - Lifecycle of a GET and a PUT request
- Description and status of the components
  - Main daemons
  - Diskserver related
  - Central services
  - Tape related
- Tape migration and recall
  - Workflow details

Detailed picture of CASTOR



Status of all system components

- Request Handler
- Stager
- RMMaster & RMNode
- Distributed Logging Facility
- External plugins (LSF, expertd)
- gcDaemon
- Central services (NameServer, VDQM, VMGR, CUPV)
- Tape part (MigHunter, migrator/recaller, rtcopy, rtcpclientd)

Request Handler

- Scope
  - Stores incoming requests into the DB
- Features
  - Very lightweight
  - Allows for request throttling
- Maturity
  - Production, no changes since mid 2005
- Implementation
  - Fully C++
  - Usage of the internal DB API

Stager

- Scope
  - Main daemon for requests processing
- Features
  - Stateless
  - Multi-services implementation by *thread pools*
    - Allows for **independent** services execution, even on different nodes
    - Enhanced scalability
- Maturity
  - Fairly stable, some bug fixes in the last months
  - Development still going on to implement missing features
  - Bugs and RFEs open on it (e.g. memory leak)
- Implementation
  - Main core in C, internal services in C or C++
  - Usage of the internal DB API

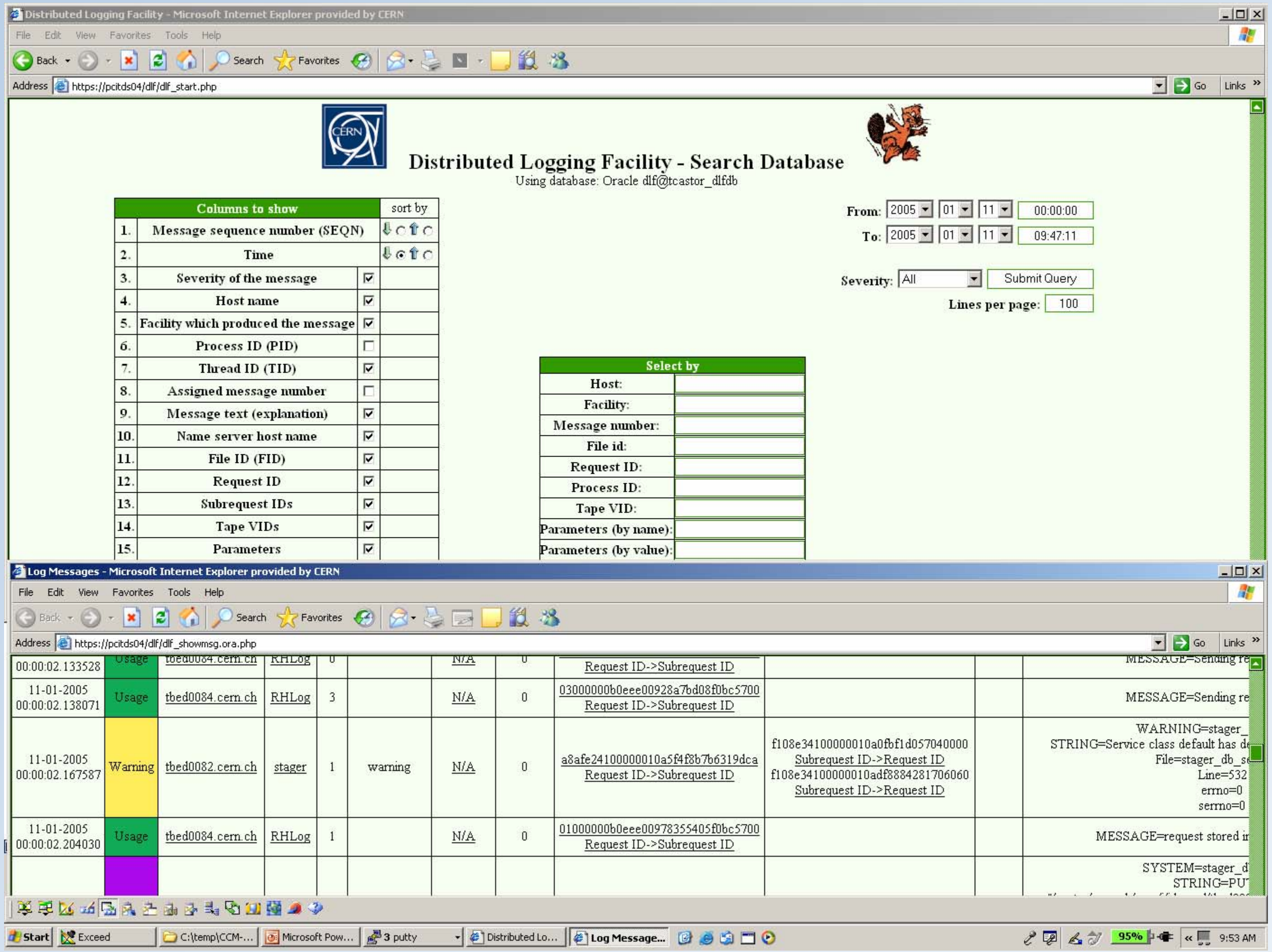
RMMaster & RMNode

- Scope
  - Gather monitoring information from nodes
  - Submit jobs to the scheduler
- Features
  - Not fully stateless
  - RMMaster gathers data from RMNode
  - RMNode runs on the diskservers and polls `/proc` data
- Maturity
  - Fairly stable, not many bugfixes in the last months
  - But quite a number of major issues are open on it
- Implementation
  - Fully C
  - No usage of DB, internal protocol to communicate with stager

Distributed Logging Facility

- Scope
  - Central DB-based logging system
- Features
  - A daemon accepts and stores any log entry from any Castor subsystem
  - A PHP-based GUI allows for querying the log
- Maturity
  - Fairly stable, development still going on to improve performances
- Implementation
  - Fully C, “legacy” DB API

DLF GUI


Distributed Logging Facility - Microsoft Internet Explorer provided by CERN
File Edit View Favorites Tools Help
Back Search Favorites
Address https://pcitds04/dlf/dlf_start.php
Go Links

Distributed Logging Facility - Search Database

Using database: Oracle dlf@tcastor_dlfdb

	Columns to show		sort by
1.	Message sequence number (SEQN)		⇩ ○ ⇧ ○
2.	Time		⇩ ● ⇧ ○
3.	Severity of the message	☑	
4.	Host name	☑	
5.	Facility which produced the message	☑	
6.	Process ID (PID)	☐	
7.	Thread ID (TID)	☑	
8.	Assigned message number	☐	
9.	Message text (explanation)	☑	
10.	Name server host name	☑	
11.	File ID (FID)	☑	
12.	Request ID	☑	
13.	Subrequest IDs	☑	
14.	Tape VIDs	☑	
15.	Parameters	☑	

From: 2005 01 11 00:00:00

To: 2005 01 11 09:47:11

Severity: All

Submit Query

Lines per page: 100

Select by	
Host:	
Facility:	
Message number:	
File id:	
Request ID:	
Process ID:	
Tape VID:	
Parameters (by name):	
Parameters (by value):	

Log Messages - Microsoft Internet Explorer provided by CERN
File Edit View Favorites Tools Help
Back Search Favorites
Address https://pcitds04/dlf/dlf_showmsg.ora.php
Go Links
Start Exceed C:\temp\CCM-... Microsoft Pow... 3 putty Distributed Lo... Log Message...
95%
9:53 AM

00:00:02.133528	Usage	tbed0084.cern.ch	RHLog	0		N/A	0	Request ID->Subrequest ID			MESSAGE=Sending re
11-01-2005 00:00:02.138071	Usage	tbed0084.cern.ch	RHLog	3		N/A	0	03000000b0eee00928a7bd08f0bc5700 Request ID->Subrequest ID			MESSAGE=Sending re
11-01-2005 00:00:02.167587	Warning	tbed0082.cern.ch	stager	1	warning	N/A	0	a8afe24100000010a5f4f8b7b6319dca Request ID->Subrequest ID	f108e34100000010a0fbf1d057040000 Subrequest ID->Request ID f108e34100000010adf8884281706060 Subrequest ID->Request ID		WARNING=stager_ STRING=Service class default has de File=stager_db_se Line=532 errno=0 serrno=0
11-01-2005 00:00:02.204030	Usage	tbed0084.cern.ch	RHLog	1		N/A	0	01000000b0eee00978355405f0bc5700 Request ID->Subrequest ID			MESSAGE=request stored ir
											SYSTEM=stager_d STRING=PU

LSF plugin

- Scope
  - Select best candidate resource (file system) among the set proposed by LSF
- Features
  - Not entirely stateless due to lack of information flow in LSF API
- Maturity
  - Stable, few changes in the last months
  - Lack of optimization due to lack of functionality in the LSF API
    - Need for further development in conjunction with LSF people
- Implementation
  - C
  - Usage of the internal DB API

gcDaemon

- Scope
  - Deletes files marked for garbage collection
- Features
  - Stateless daemon implemented as a stager client
- Maturity
  - Production, no changes since Dec 2004
- Implementation
  - C++
  - Usage of the client API and the internal API
    - proxy “remotized” implementation of the stager

NameServer

- Scope
  - Archive the filesystem-like information for the HSM files
  - Associate tape related information
- Features
  - Stateless daemon, DB backend
- Maturity
  - Production, last change has been a merge with DPM’s NameServer in Jan 2006, otherwise no changes since 2004
- Implementation
  - Fully C

Expert daemon (expertd)

- Scope
  - Externalize decisions based on policies
- Features
  - Framework for executing policy scripts
  - Receives policy requests from other components (stager, MigHunter, TapeErrorHandler)
  - Supported policy requests types are:
    - Filesystem weight
    - Replication
    - Migrator
    - Recaller

Filesystem policy

- Scope
  - Provide an evaluation of each resource (filesystem) from gathered monitoring information
- Features
  - Single formula implementation
  - Currently only global, to be converted soon to policy per service class
- Maturity
  - Under development, the current implementation works in production but has demonstrated not to be stable enough under very heavy load (Tier0 Data Challenge)
- Implementation
  - Rule in CLIPS logic engine, going to be converted to Perl

Volume and Drive Queue Mgr (VDQM)

- Scope
  - Manage the tape queue and device status
- Features
  - Supports drive dedication (regexp)
  - Supports request prioritization
  - Allows for re-use of mounted tapes (useful for CASTOR1)
- Maturity
  - In production since 2000
  - Scheduling algorithm melts down beyond ~4000 queued requests
  - New implementation (VDQM 2) ready to be rolled out
- Implementation
  - C
  - C++ and DB API for the new VDQM 2

Volume Manager (VMGR)

- Scope
  - Logical Volume Repository. Inventory of all tapes and their status
- Features
  - Tape pools
    - Grouping of tapes for given activities
    - Counters for total and free space (calculated using compression rates)
- Maturity
  - In production since 2000
- Implementation
  - C
  - Oracle Pro-C

Castor User Privileges (Cupv)

- Scope
  - Manages administrative authorization rights on other CASTOR modules (nameserver, VMGR)
- Features
  - Flat repository of privileges
  - Supports regular expressions
- Maturity
  - In production since 2000
- Implementation
  - C
  - Oracle Pro-C

Tape mover (rtcpd)

- Scope
  - Copy files between tape and disk
- Features
  - Highly multithreaded
    - Overlaid network and tape I/O
    - Large memory buffers allows for copying multiple files in parallel
  - Supports a large number of legacy tape formats…
- Maturity
  - In production since 2000
- Implementation
  - C

MigHunter

- Scope
  - Attach migration candidates to streams
- Features
  - Stateless
  - Callout to expert system for executing migrator policies for fine-grained control
  - Can trigger on frequency or volume of data to be migrated
- Maturity
  - In production since 2006
  - Some known problems with files that have been deleted from the name server but not cleared in the catalogue
- Implementation
  - C
  - Usage of internal DB API

Migrator/recaller

- Scope
  - Controls the tape migration/recall
- Features
  - Stateless, multithreaded
- Maturity
  - In production since 2006
  - Some known problems with files that have been deleted from the name server but not cleared in the catalogue
  - Known aging problem resulting in inconsistency in one auxiliary oracle table that is updated through triggers
    - Workaround for oracle problem
    - Operational procedure exists for repairing the streams
- Implementation
  - C
  - Usage of internal DB API

rtcpclientd

- Scope
  - Master daemon controlling tape migration/recall
- Features
  - Not fully stateless due to VDQM
  - Single threaded
- Maturity
  - In production since 2006
- Implementation
  - C
  - Usage of internal DB API

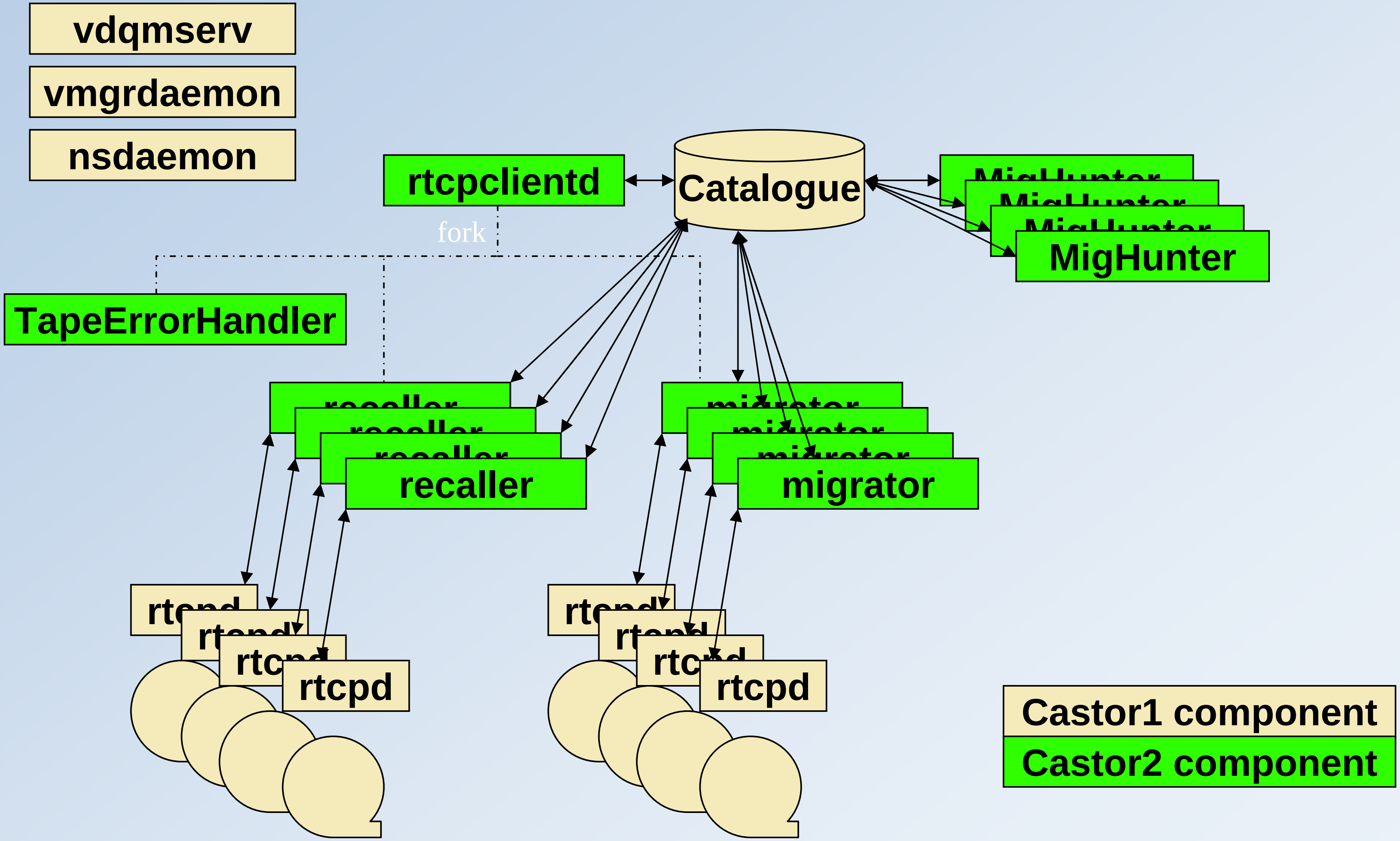
Outline

- Detailed view of the architecture
  - Lifecycle of a GET and a PUT request
- Description and status of the components
  - Main daemons
  - Diskserver related
  - Central services
  - Tape related
- Tape migration and recall

Tape Migration and Recall

- “rtcpclientd” is the main component dealing with all interaction to the CASTOR tape archive
  - For each running tape recall it forks a ‘recaller’ child process per tape
  - For each running tape migration it forks a ‘migrator’ child process per tape
- Migration streams are created and populated by the “MigHunter” component
- A TapeErrorHandler process is forked by the rtcpclientd daemon whenever a recaller or migrator child process exits with error status.
- Detailed description of the functioning and operation of tape migration and recall in CASTOR2 can be found at:
  - http://cern.ch/castor/docs/guides/admin/tapeMigrationAndRecall.pdf

Tape migration/recall components



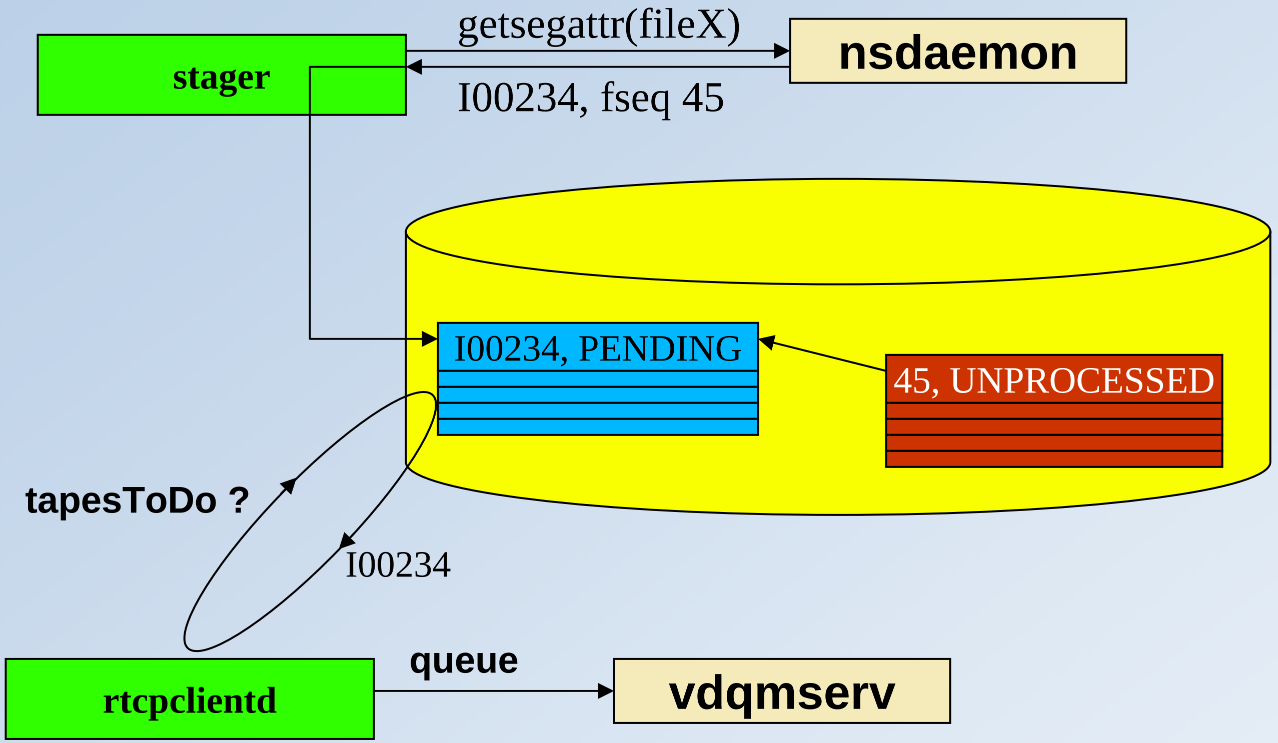
Tape recall (1)

- Tape recalls are triggered when the stager receives a request for a CASTOR file for which there is no available disk resident copy
  - Stager calls the castor name server to retrieve the tape segment information (VID, fseq, blockid)
  - Stager inserts the corresponding rows in the Tape and Segment tables in the catalogue
- rtcpclientd regularly (every 30s) checks the catalogue for tapes to be recalled
  - Submits the tape request to VDQM (tape queue)
  - When mover (rtcpd) starts it connects back to the rtcpclientd, which then forks a recaller process for servicing the tape recall

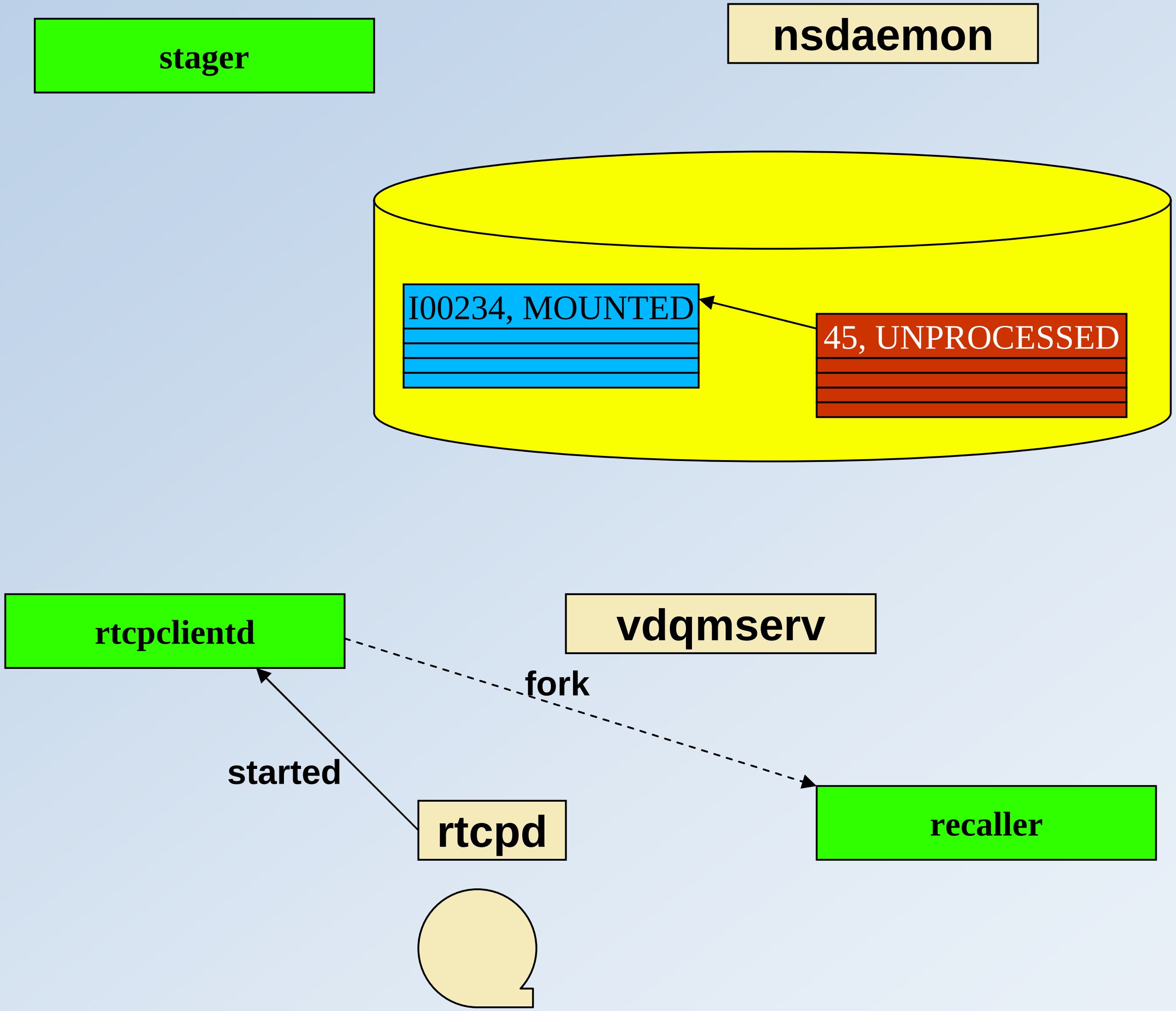
Tape recall (2)

- The recaller attempts to optimize the use of tape and disk resources
  - Tape files are sorted
    - Current in fseq order. Ongoing work to find more optimal sorting taking into account the serpentine track layout on media
  - Requests for new files on same tape are dynamically added to running request
  - Target file system is decided given the current load picture when the tape file is positioned

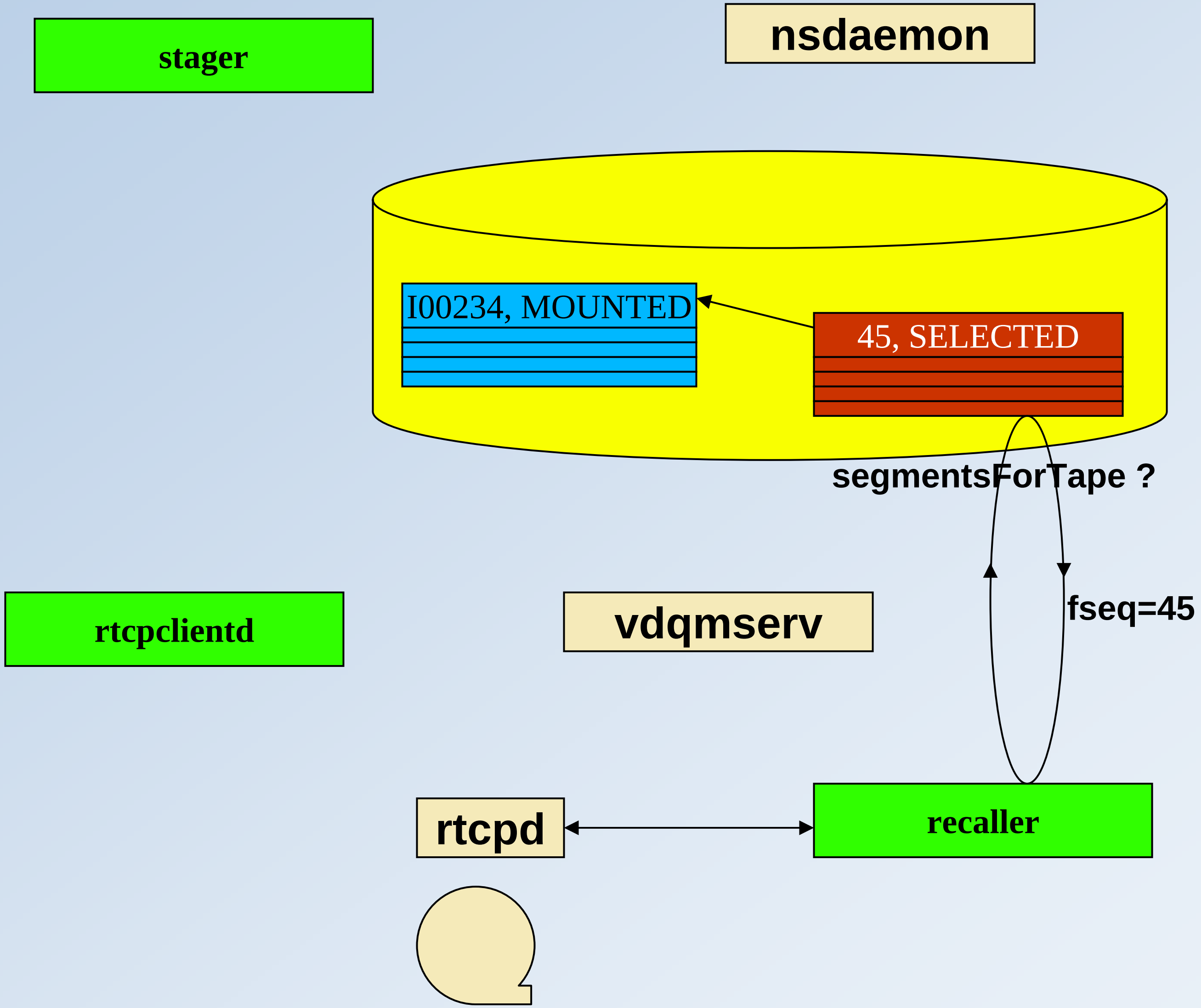
Tape recall flow



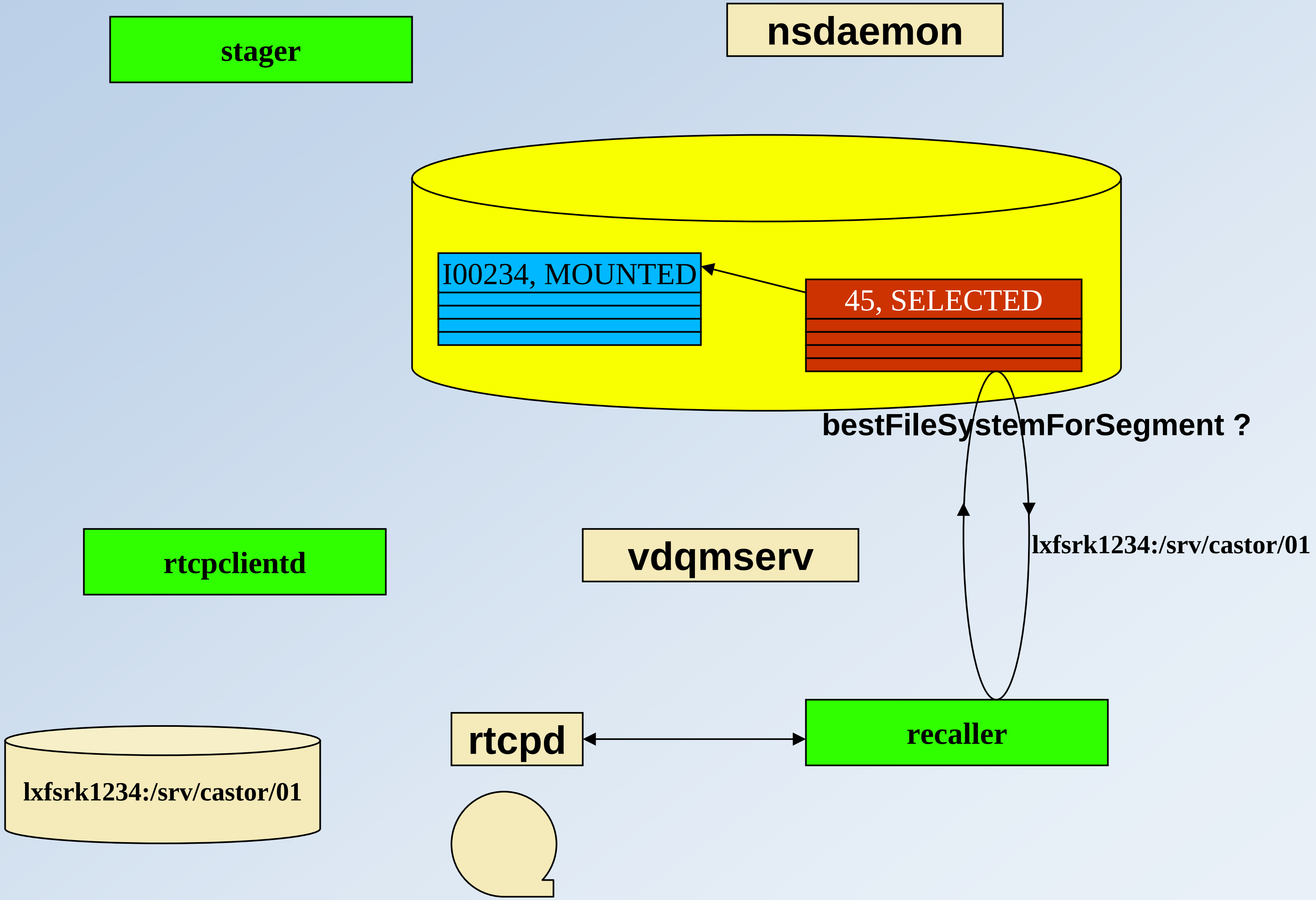
Tape recall flow



Tape recall flow



Tape recall flow


stager
nsdaemon
I00234, MOUNTED
45, SELECTED
bestFileSystemForSegment ?
lxfsrk1234:/srv/castor/01
rtcpclientd
vdqmserv
rtcpd
recaller
lxfsrk1234:/srv/castor/01

Tape migration

- Similar to tape recall but
  - Triggered by policy rather than waiting requests
- Migration candidates are attached to ‘streams’
  - A migration ‘Stream’ is a container of migration candidates
  - Each Stream is associated with 0 or 1 tapes:
    - 0 tape: stream not active (e.g. not yet picked up by rtcpclientd, or VMGR tape pool is full)
    - 1 tape: stream is running (tape write request is running) or waiting for tape mount
  - A Stream can survive many tapes (but only one at a time)
  - A TapeCopy can be linked to many Streams
    - When a TapeCopy is selected by one of the Streams, its status is atomically updated preventing it from being selected by another Stream
- The MigHunter process is responsible for attaching the migration candidates to the streams
  - Migrator policies can be used for fine-grained control over this process

Example policy

```
#!/usr/bin/perl -w
#
# Migration policy for distinguishing between small and large files
# - if fileSize < 100MB → smallfiles
# - if fileSize >= 100MB → largeFiles
#
use strict;
use diagnostics;
use POSIX;
my $doMigrate = 0;
END {print "$doMigrate\n";}

my ($tapePool,$castorFile,$copynb,$fileid,$fileSize,$mode,$uid,$gid,$atime,$mtime,$ctime,$fileClassId) = @ARGV;

if ( (($tapePool =~ “smallfiles") && ($fileSize < 100*1024*1024)) ||
     (($tapePool =~ “largeFiles") && ($fileSize >= 100*1024*1024)) ) {
        $doMigrate = 1;
}

exit(EXIT_SUCCESS);
```

Comments, questions?