Protocol, clients and APIs

Sebastien Ponce, Giuseppe Lo Presti, German Cancio
CERN / IT

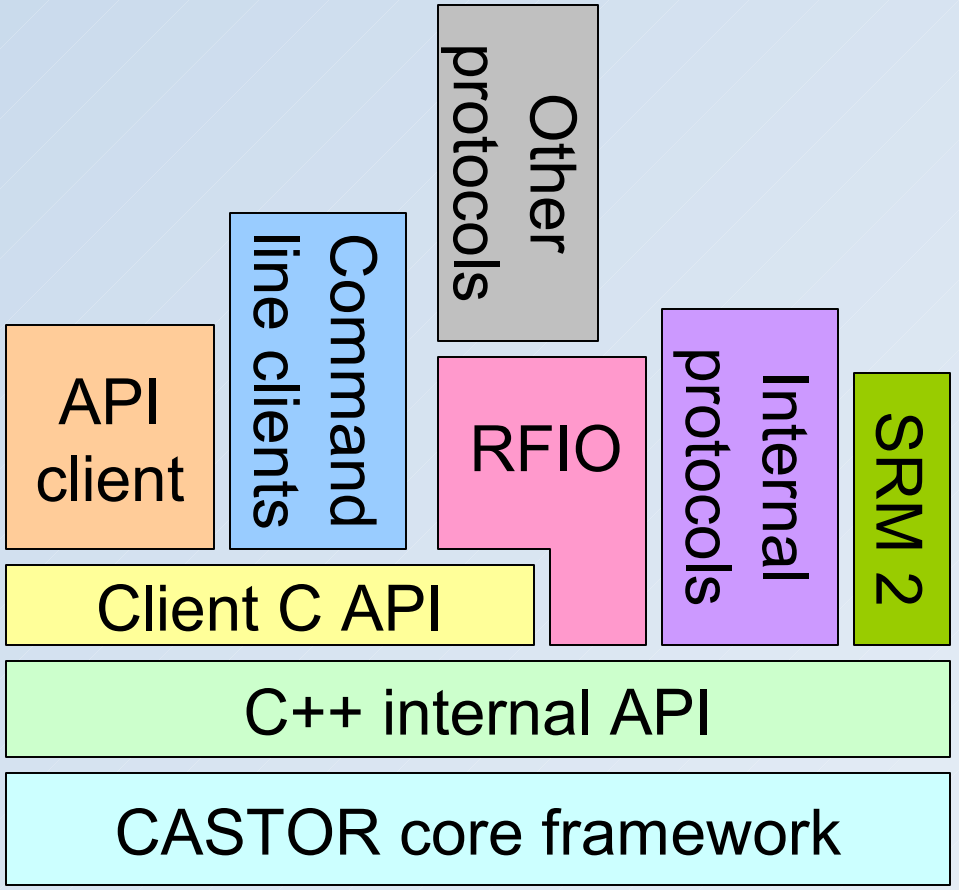
Castor Readiness Review – June 2006

Outline

- Internal C++ API vs client, C API
- Command line clients
- Protocol supported
  - Rfio, root as internal
  - Xroot being integrated
  - gridFTP as external
- Few words about the SRM status

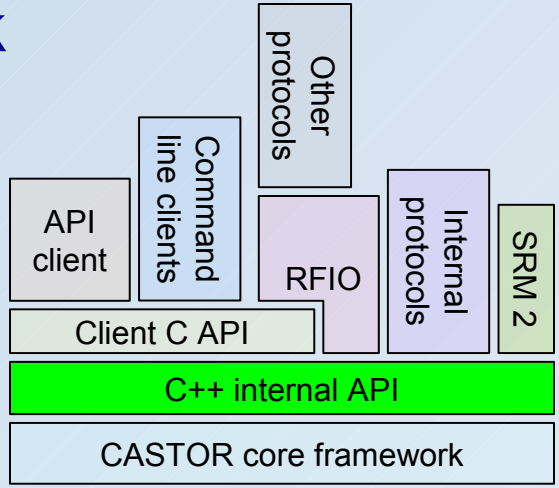
The APIs and client pile

- 2 levels of API
  - Internal, C++
  - Client, C
- 2 levels of support for protocols
  - Internal
  - Other



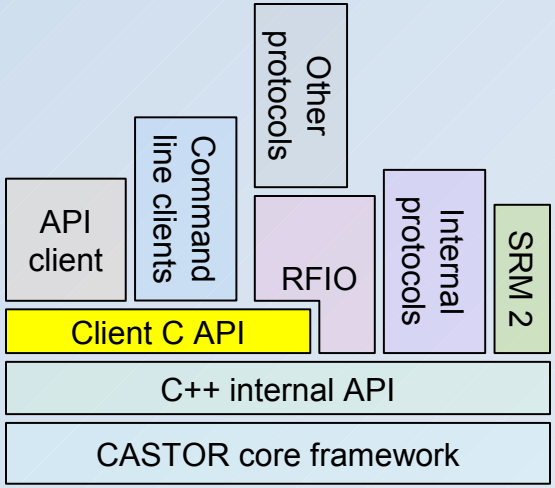
Internal API

- API to the core CASTOR 2 framework
- Used by the castor components and some tighly integrated external parts
  - Protocols like rfio, root, xroot
  - SRM 2
- Implemented in C++
  - With some parts interfaced in C thanks to code generation
- Not distributed in RPMs yet, only in CVS
- Not stable on the long term



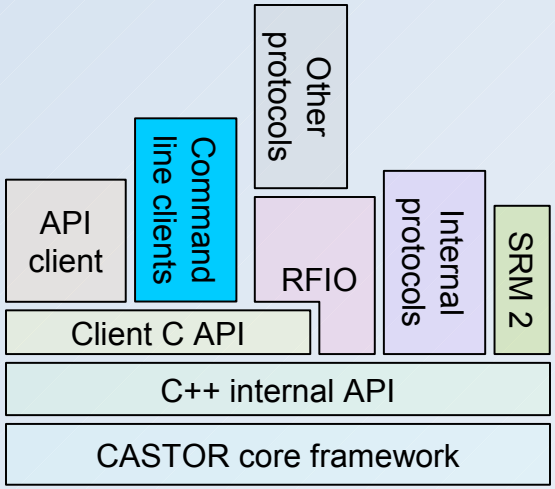
Client API

- API to be used by the clients
  - External software
  - Command line clients
- Covers all castor components (old and new ones)
- C API at general request
  - Still implemented in C++
- Distributed in the castor-devel RPM
- Very stable
  - Guaranteed backward compatibility within major release
  - Ensured by soname of the libshift.so.x.y library



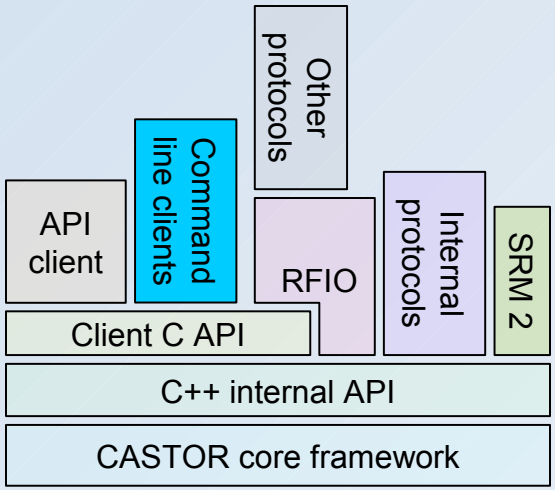
Command line clients

- Split into several sets
  - Nameserver commands (prefix ns)
    - For metadata, related to the namespace
  - RFIO commands (prefix rf)
    - For transfers using rfio
  - Stager commands (prefix stager, stage for old ones)
    - For creating, recalling, migrating, querying the files
  - Tape related commands
    - For the drive queue (prefix vdqm)
    - For the volume management (prefix vmgr)
    - For the tape handling (prefix tape)
    - For the transfer to tape (prefix rtcp)
  - Privileges commands
    - For internal priviledges (prefix cupv)
  - Many others (admin, logging, …)



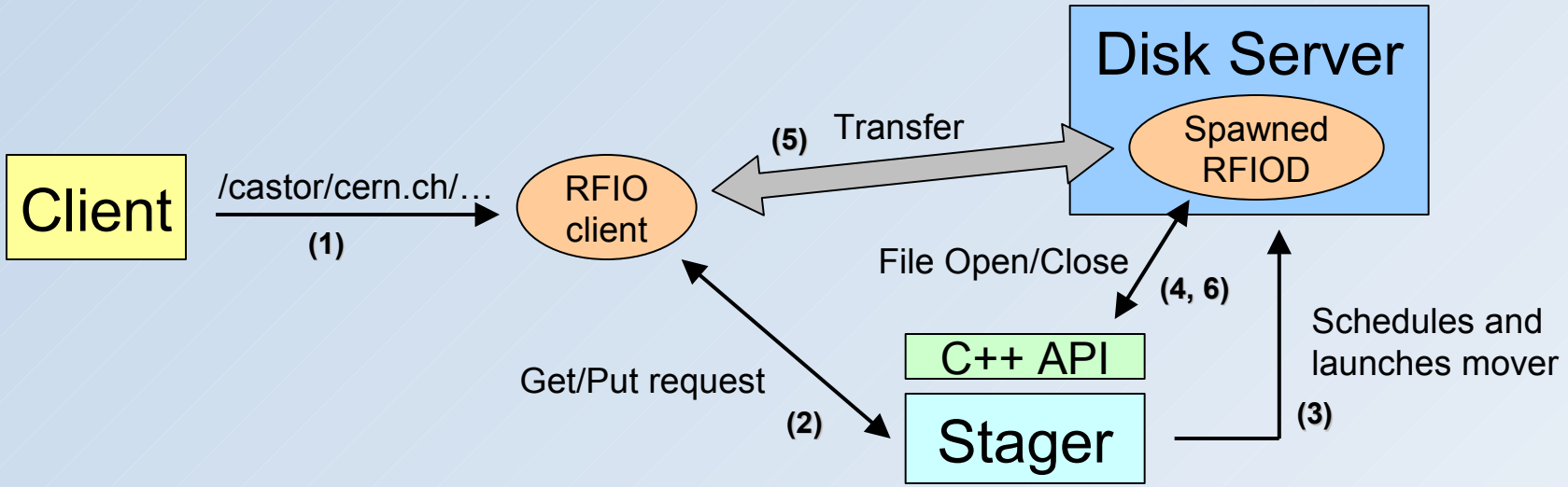
Command line clients (2)

- Command lines are distributed in separate RPMs
  - castor-commands
  - ns-client, cupv-client, vmgr-client, vdqm-client, tape-client, rfio-client, …
- Command line should always use the client API
  - Thus light weighted, only parsing options and displaying result on the prompt
- All have
  - man pages
  - –h, --help flag



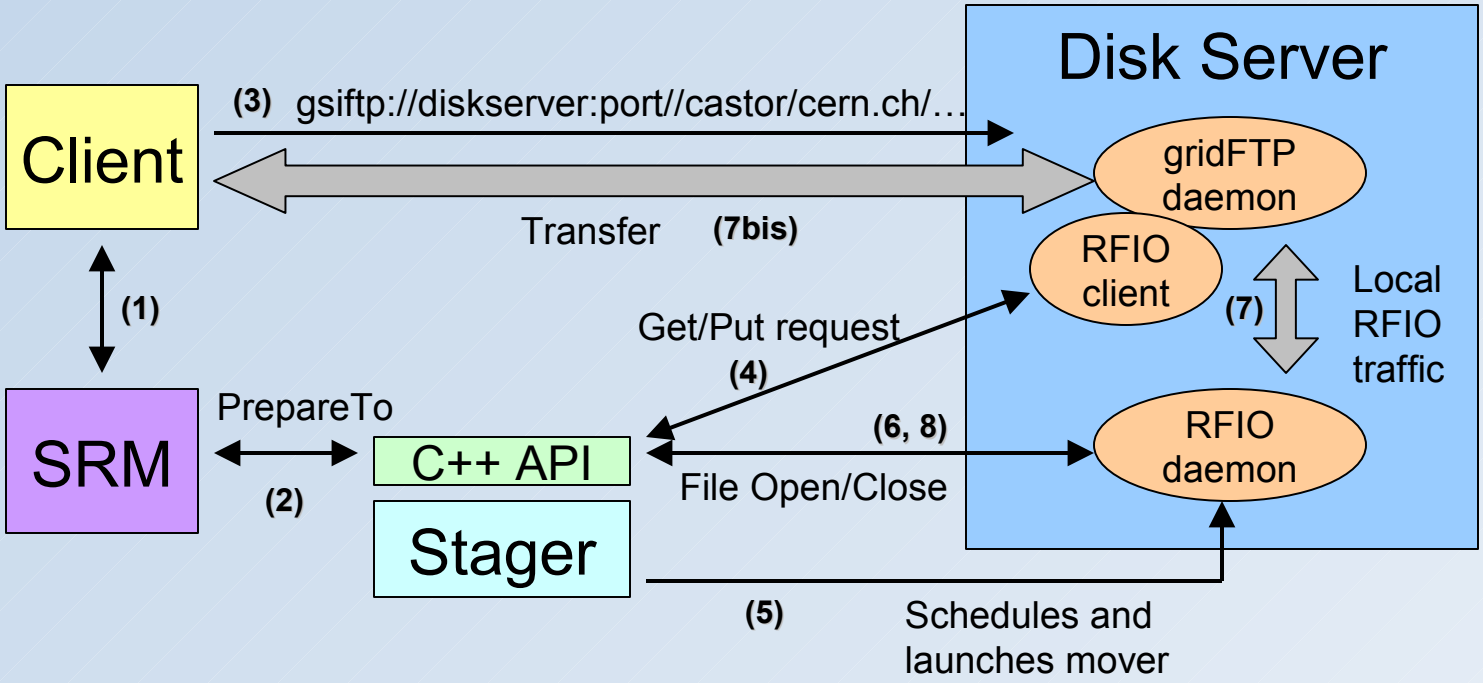
Support of internal protocols

- Internal Protocol
  - Has access to the castor internal C++ API
  - Deals with file names like /castor/… by calling the stager
  - Are used as native protocols to read the data from the disk on the diskserver and transfer it to the clients
  - Are developed/modified by the CASTOR team



Support of other protocols

- Other protocols
  - Use only the client API
  - Don't know about names like /castor/…
  - Use internally a native protocol to access the data



RFIO and ROOT

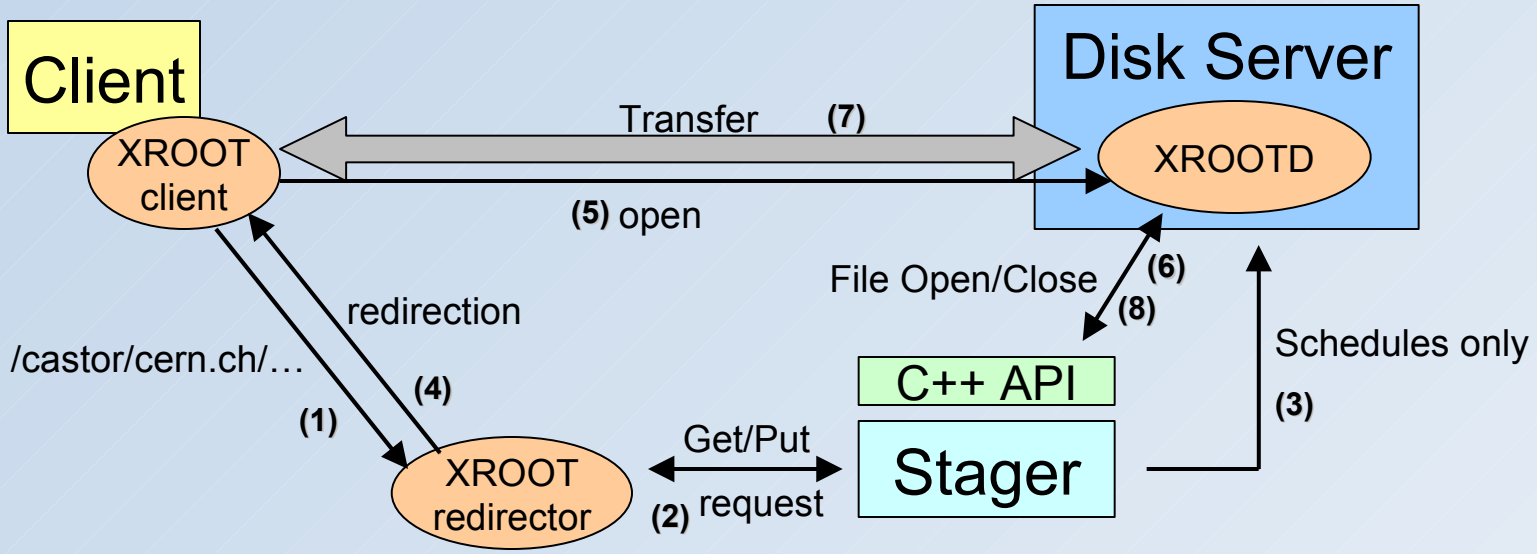
- Internal protocols, running on all diskservers
- RFIO is part of the castor distribution
- ROOT is part of the ROOT distribution
  - Initial implementation by the CASTOR team
  - Supported by Fons from the ROOT team
  - Patched by CASTOR developers when needed
- The stager is called for each single file access, allowing full I/O scheduling
- The daemons (rfiod and rootd) are started on demand
  - They are modified to only serve the scheduled file
  - They serve a single request in the specified mode (e.g. ro)

GridFTP

- Is NOT an internal protocol
- Only version 1 is supported
  - Version 2 should become an internal protocol at the end of the year (see planning)
- Runs only on WAN diskservers
- Runs independently of CASTOR
- Uses internally RFIO to retrieve the files
  - This RFIO is always local
  - The SRM takes care that the request is sent to the right diskserver at the gridFTP level

XROOT

- Recently integrated with CASTOR
  - Work done by Andrew Hanushevsky from XROOT team
  - Helped by Rosa for integration and tests
- XROOT is now an internal protocol



- If concurrent accesses to one file
  - Steps 2, 3 are skipped
  - Steps 6, 8 are only issued once

XROOT (2)

- XROOT will only run on dedicated pools (Alice)
- The daemons runs independently of CASTOR
- XROOT calls the stager for files opening/closing
  - Handles concurrent accesses with a single open/close from the CASTOR point of view
  - The plan is to let XROOT deal with the load balancing
- Future, optimal version
  - 2 levels of disk cache
  - one dedicated to tape access, running RFIO
    - Load balancing via strict scheduling from CASTOR
  - one dedicated to client, running XROOT
    - Load balancing by XROOT, no restriction from CASTOR