CASTOR Tutorial

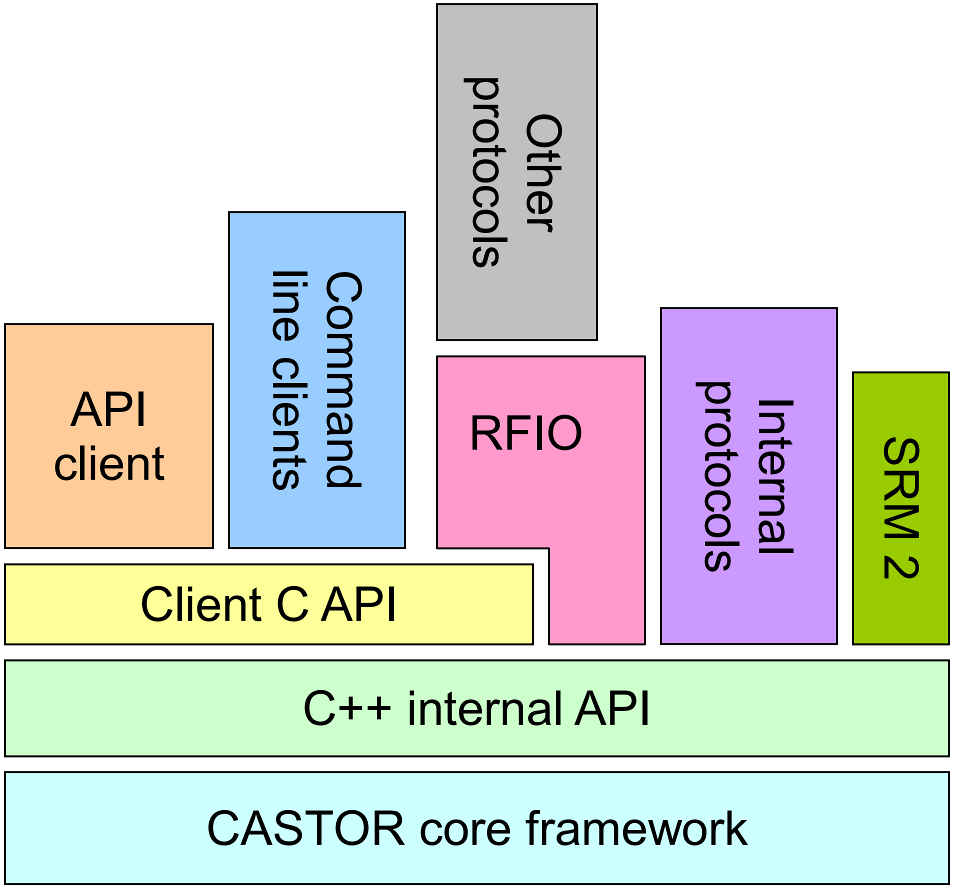
Part 3
Protocols, clients and APIs

Outline

- Internal C++ API vs client, C API
- Command line clients
- Protocol supported
  - Internal vs external protocol support
  - Protocols supported
    - RFIO, ROOT, XROOT, gridFTP
  - Specificities of Xroot

The APIs and client pile

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



Other protocols

Command line clients

API client

RFIO

Internal protocols

SRM 2

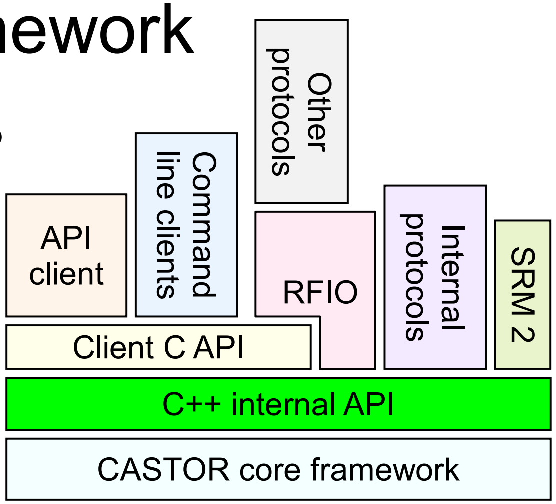
Client C API

C++ internal API

CASTOR core framework

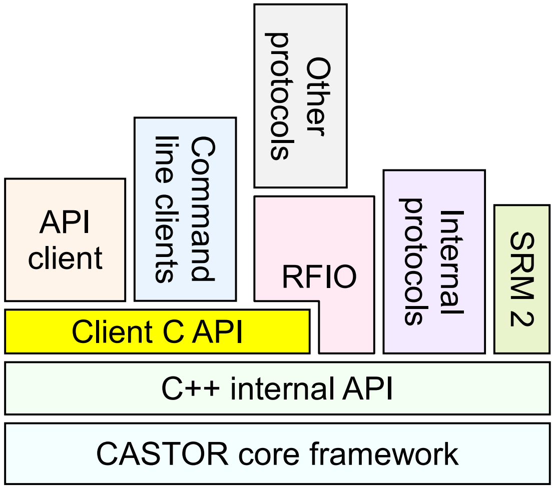
Internal API

- 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, only in CVS
  - Makes compilation of e.g. SRM tricky
- Not stable on the middle/long term

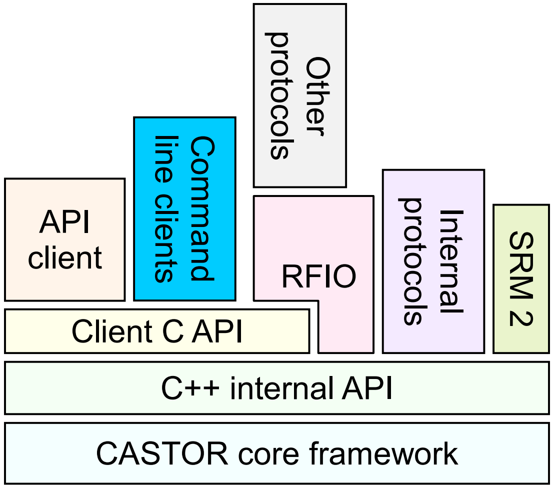


Client API

- API to be used by the clients
  - External software
  - Command line clients
- Covers all castor components
- C API at general request
  - Still implemented in C++
  - This is transparent
- Distributed in the castor-devel RPM
- Very stable
  - Guaranteed backward compatibility within major release
  - Ensured by soname of the CASTOR libraries
    - e.g. CMS is still using CASTOR 2.1.1 clients



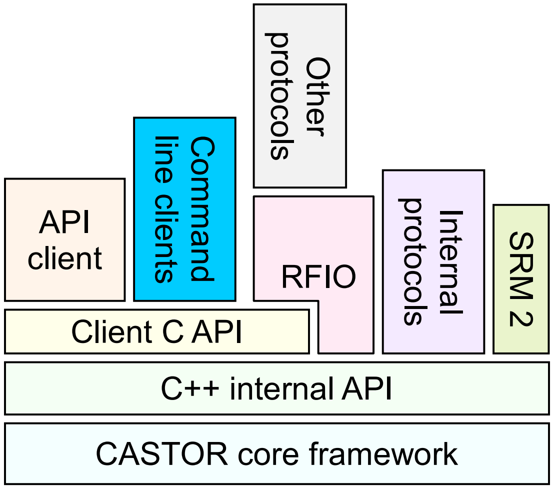
Command line clients



- Split into several sets
  - Nameserver commands (prefix ns)
    - For metadata, namespace related
  - RFIO commands (prefix rf)
    - For transfers using rfio
  - Stager commands (prefix stager)
    - 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 or tp)
    - For the transfer to tape (prefix rtcp)
  - Privileges commands
    - For internal privileges (prefix cupv)
  - Many others (admin, logging, …)

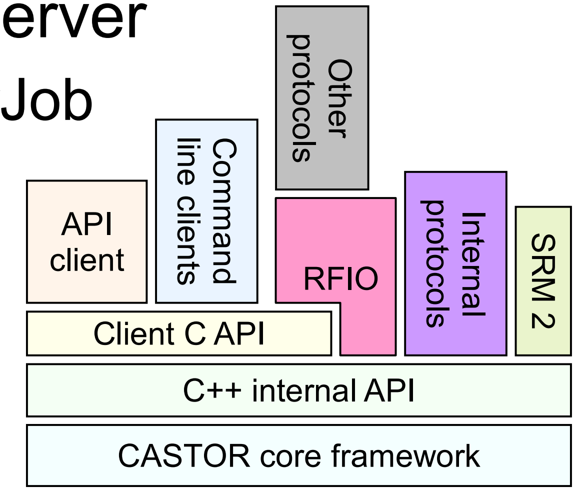
Command line clients (2)

- Command lines are distributed in separate RPMs
  - 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



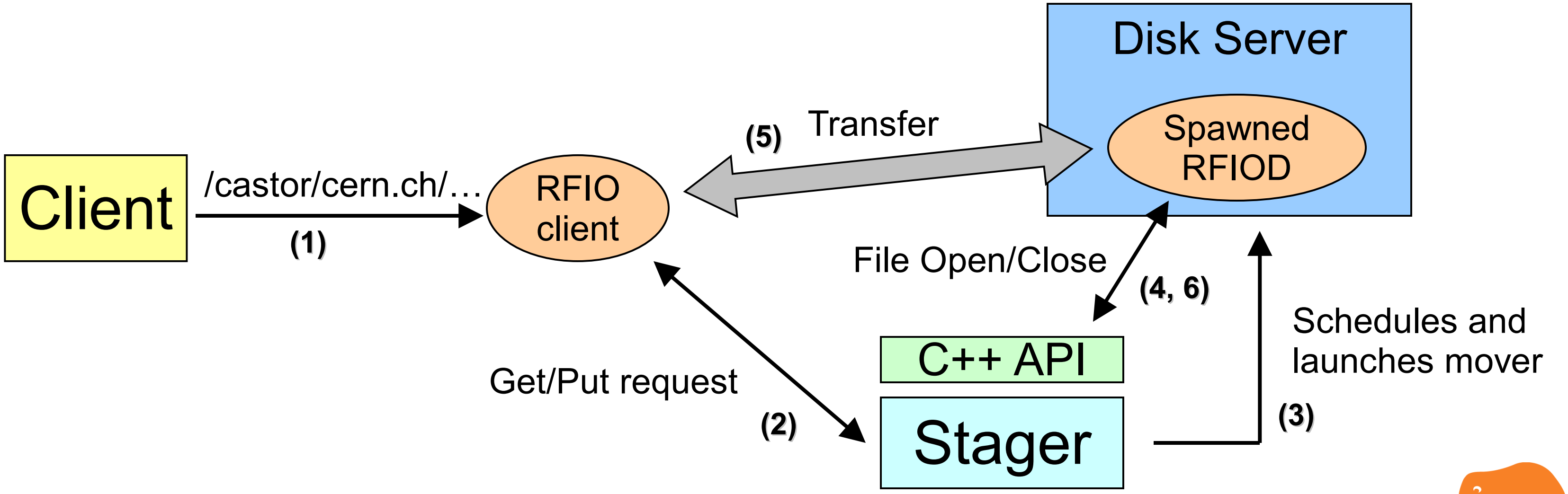
Protocols

- Run inside stagerJob on the diskserver
  - Can be even forked by the stagerJob
- 2 levels of CASTOR integration
  - Internal protocols can access CASTOR disk cache
  - External protocols go through an internal one
- For internal ones, 2 levels of code integration
  - Raw protocols are only wrapped into stagerJob
  - Integrated protocols are CASTOR aware
    - And call the internal C++ API



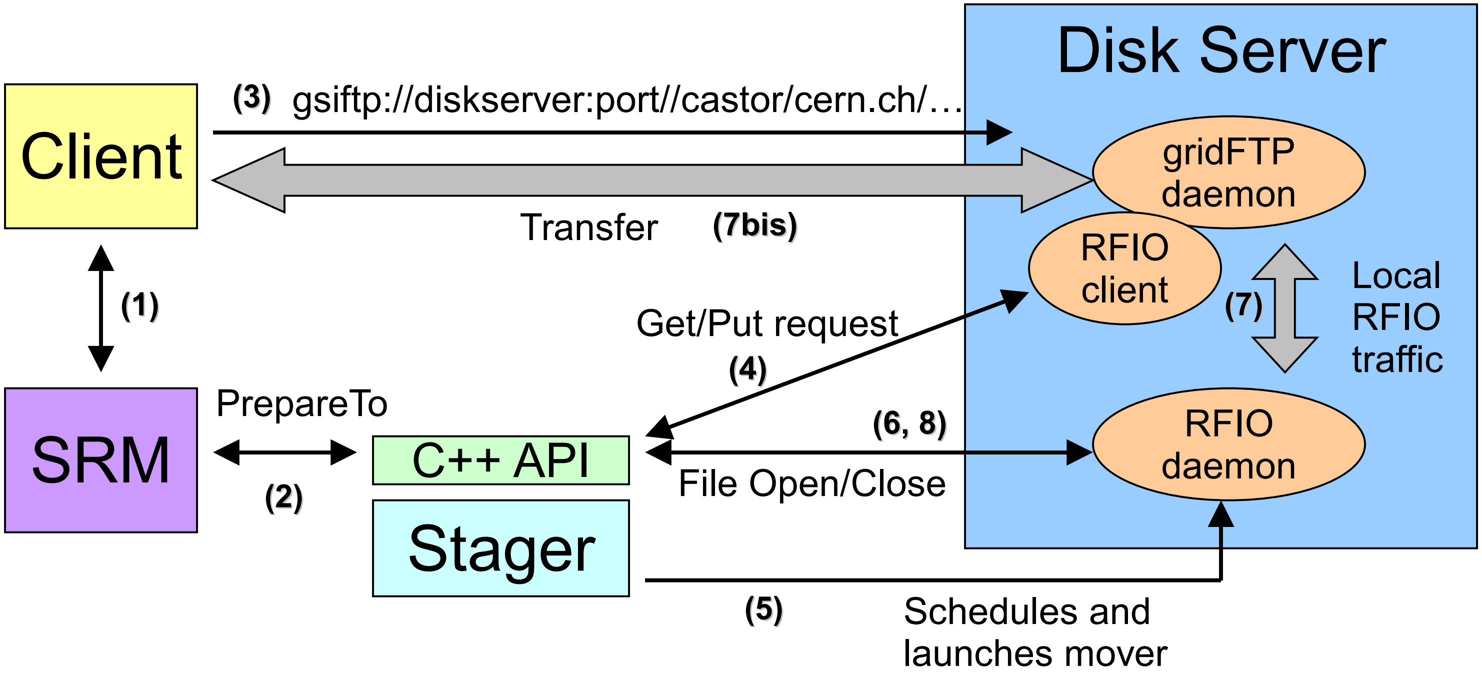
Support of internal protocols

- 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

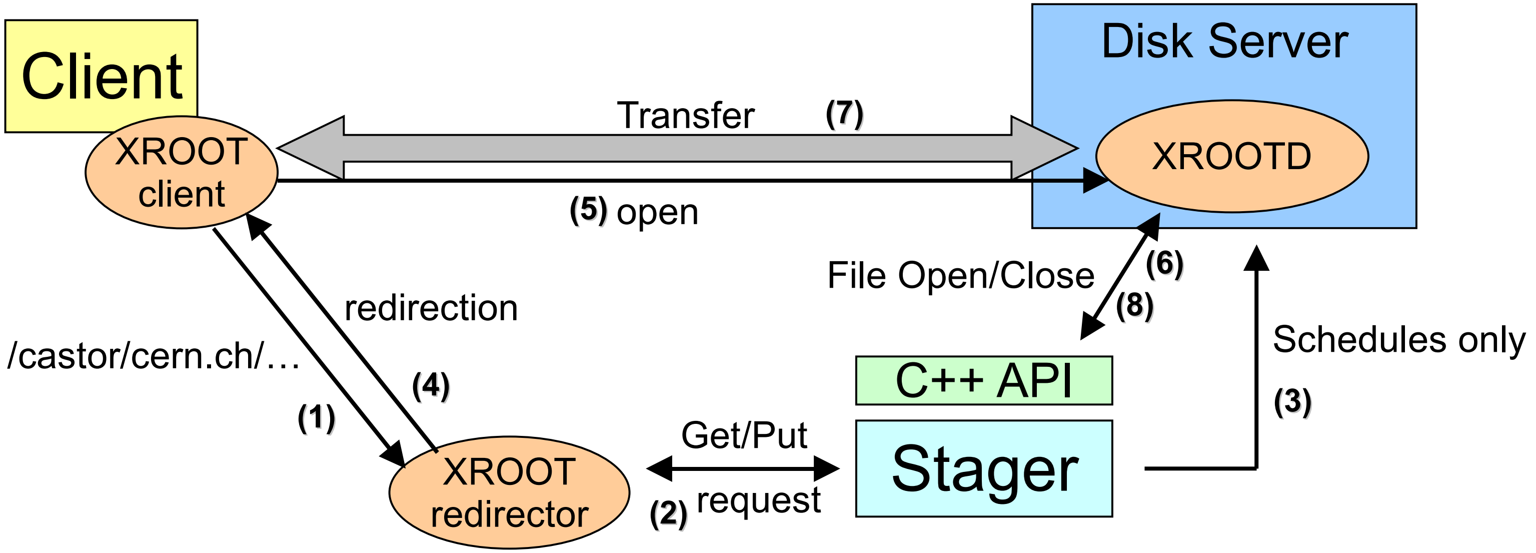
- Internal protocols, running on all diskservers
- RFIO is part of the castor distribution
- ROOT is part of the ROOT distribution
  - Initial implementation by the CASTOR team
  - Supported by the ROOT team
  - Patched by CASTOR developers when needed
- The stager is called for each single file access, 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

- Exists in 2 versions : internal and external
- Only version 2 is supported (Version 1 dropped)
- In external mode :
  - Runs independently of CASTOR, under root
  - Uses internally RFIO to retrieve the files
    - This RFIO should always be local (use of SRM)
    - SRM is not mandatory
  - gridmap file needed for user mapping
- In internal mode :
  - Spawned on demand, under stage/st
  - SRM required
  - gridmap file ignored

2.1.7 XROOT integration

- XROOT is an internal protocol
- Xroot daemon runs independently of CASTOR
- The redirector is CASTOR aware
  - And talks to the stager

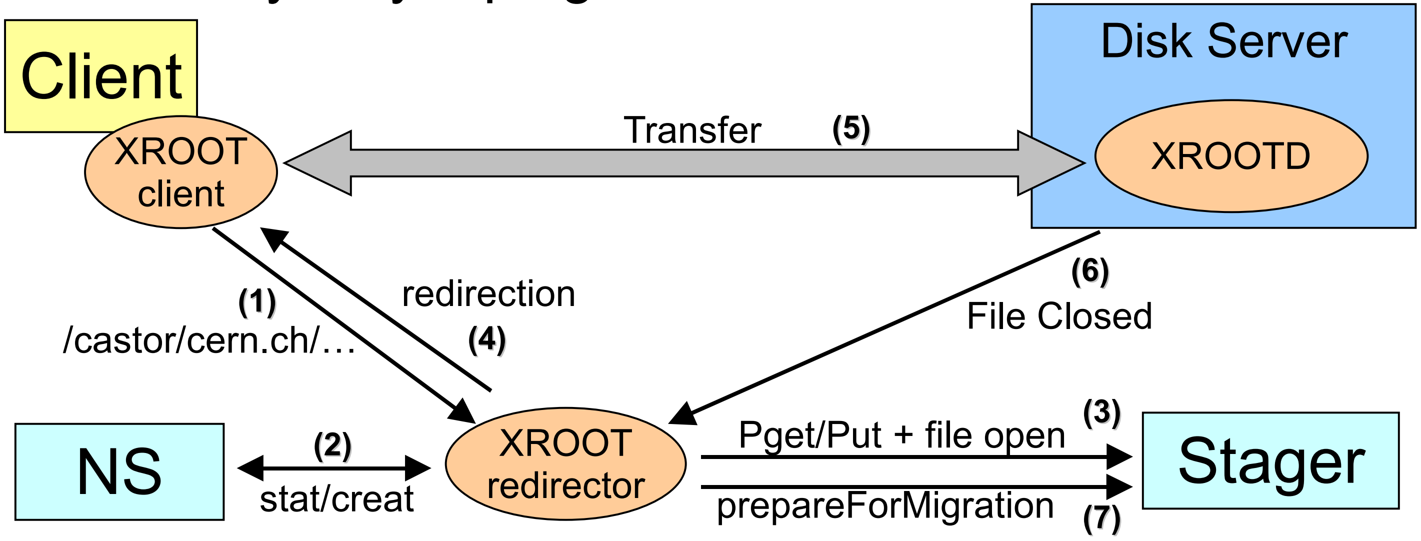


❖ If concurrent accesses to one file

- Steps 2, 3 are skipped
- Steps 6, 8 are only issued once

2.1.8 XROOT integration

- XROOT is part of CASTOR, much tighter integration
  - Actually only a plugin to XROOT



- Step 3 is configurable for Get requests
  - can be ignored with step 2 using XROOT caching
  - can schedule with LSF
  - default is none of those
- Step 6 and 7 only exist for puts
- Native Xroot in case of reads
  - but for one access to the nameserver on first read

2.1.8 XROOT integration

- Benefits from low latency of XROOT
  - 80ms per file opening (1-2s for CASTOR)
    - Will be lowered via nameserver optimizations
  - few ms if XROOT cache is activated
- Many connections per second (small files)
  - >700 connections per second
- Native XROOT for bandwidth optimizations
  - Can serve concurrently 100s of streams per node
- Extensions of XROOT
  - Security (Globus, kerberos)
  - Stream scheduling on a disk server
    - Ability to dynamically lower throughput dedicated to users when a tape stream starts