HTTP-TPC transfers with nearline storage

Transfers that target tape: progress in three parts

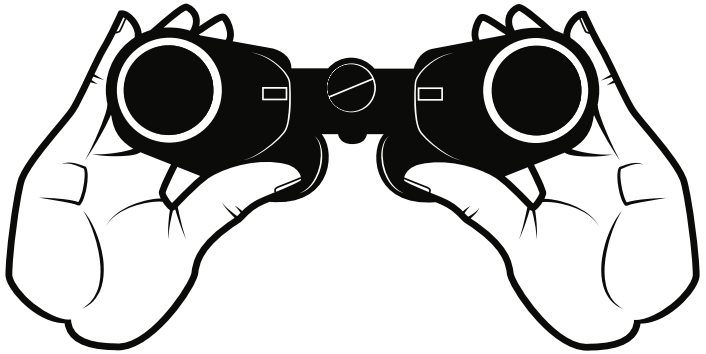
Paul Millar
Hamburg, 2020-10-07

My thanks go to Andrea Ceccanti and Mihai Patrascoiu
for their help in putting together these slides.

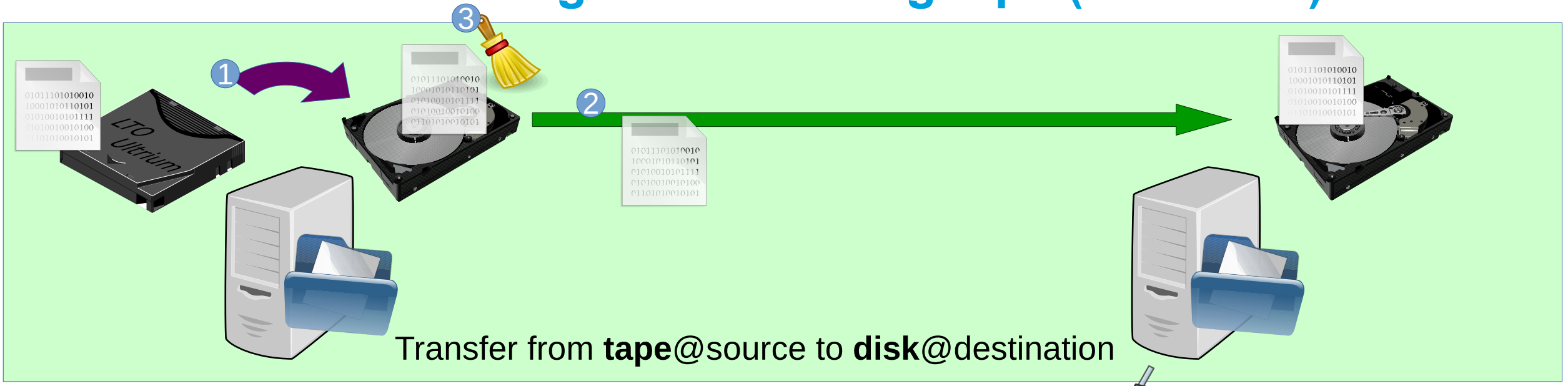
HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

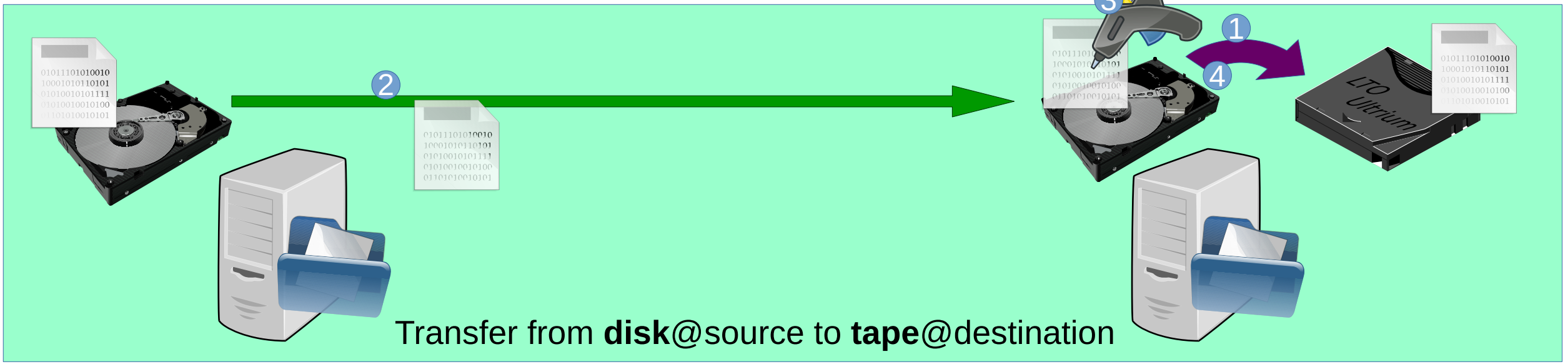
DESY.

A review of the current status



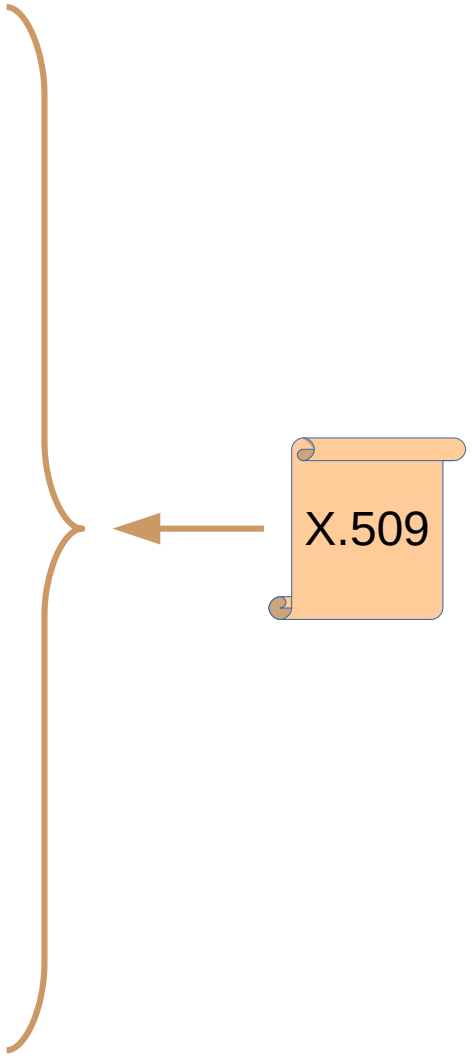
Scenarios: transferring data involving tape (with SRM)





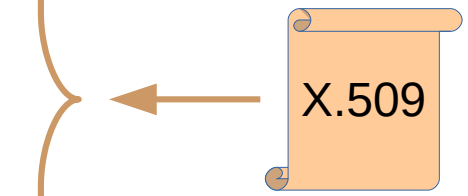
Today's solution: transferring tape to disk

1. Stage file on **source** server to disk: **`srmBringOnline`**
2. Monitor stage request's progress: **`srmStatusOfBringOnlineRequest`**.
3. Once stage request completes:
   - Request a GridFTP TURL on **source** server: **`srmPrepareToGet`**
   - Request a GridFTP TURL on **destination** server: **`srmPrepareToPut`**
4. Connect to **destination** GridFTP server and request to upload file.
   Response is an IP address and port number that will receive the data.
5. Connect to **source** GridFTP server and request file is sent
   - Give **source** GridFTP server the IP address and port number above.
   - **Source** server connects to **destination** server directly → third-party transfer.
6. Monitor transfer's progress.
7. Once transfer completes:
   - Release pins on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**



Today's solution: transferring disk to tape

1. Request a GridFTP TURL on **source** server: **`srmPrepareToGet`**
2. Request a GridFTP TURL on **destination** server: **`srmPrepareToPut`**
   Specify the desired Access Latency (`NEARLINE`) and Retention Policy (`CUSTODIAL`) → "write this file to tape".
3. Connect to **destination** GridFTP server and request to upload file.
   Response is an IP address and port number that will receive the data.
4. Connect to **source** GridFTP server and request file is sent
   - Give **source** GridFTP server the IP address and port number above.
   - **Source** server connects to **destination** server directly → third-party transfer.
5. Monitor transfer's progress.
6. Once transfer completes:
   - Release the pin on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**
7. (Optionally) monitor file being written to tape: **`srmLs`**



Part 1: tape transfers without GridFTP

… but still using X.509 to get Storage-issued tokens (“macaroons”).

Little known secret #1: SRM is transfer protocol agnostic

The **SRM protocol** does *not* require that transfer use gsiftp/GridFTP, but allows the SRM client and SRM server to negotiate which protocol to use.

- A powerful feature built into the specification.

GridFTP (`gsiftp://`) is the **common choice** for a protocol, but HTTPS (`https://`) will work if the server supports it.

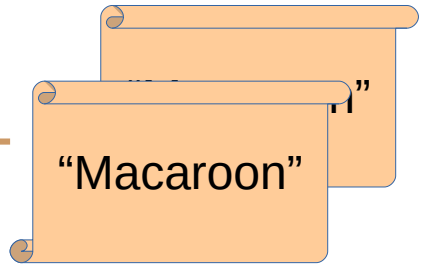
- Tested with dCache and StoRM: both work fine.

Requesting an HTTPS TURL allows the client to **use HTTP-TPC**

- This is analogous to how requesting a GridFTP TURL allows the client to initiate an FTP third-party transfer.

Transferring tape to disk with HTTP-TPC and “macaroons”

1. Stage file on **source** server to disk: **`srmBringOnline`**
2. Monitor stage request’s progress:
   `srmStatusOfBringOnlineRequest`.
3. Once stage request completes:
   - Request a **HTTP** TURL on **source** server: **`srmPrepareToGet`**
   - Request a **HTTP** TURL on **destination** server: **`srmPrepareToPut`**
4. **Obtain bearer tokens (“macaroons”):**
   - **Connect to source HTTP server and request a token.**
   - **Connect to destination HTTP server and request a token.**
5. Connect to **destination HTTP** server and request **HTTP-TPC**.
6. Monitor transfer’s progress.
7. Once transfer completes:
   - Release pins on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**



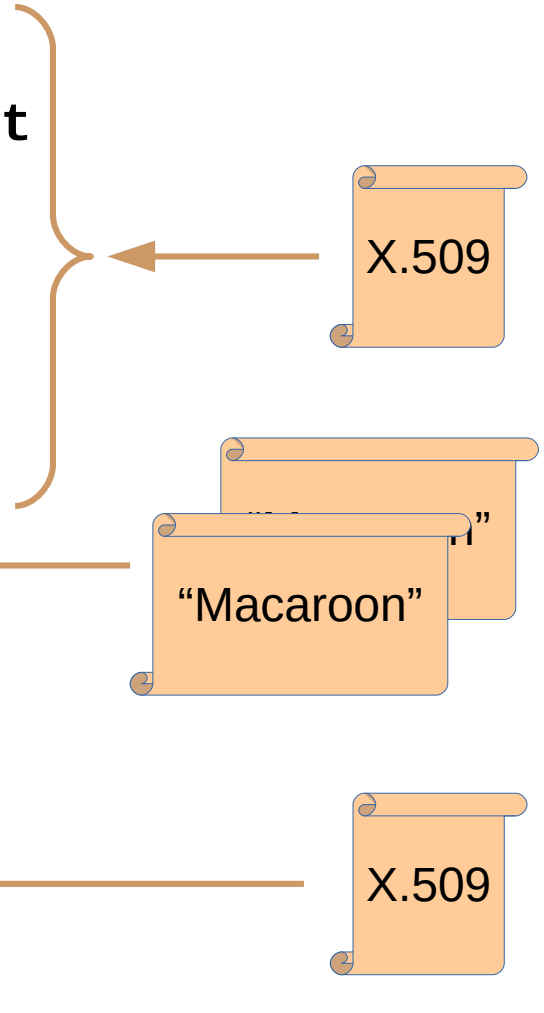
Purple marks parts that are different from today.

Transferring disk to tape with HTTP-TPC and “macaroons”

1. Request an **HTTP** TURL on **source** server: **`srmPrepareToGet`**
2. Request an **HTTP** TURL on **destination** server: **`srmPrepareToPut`**

   Specify the desired Access Latency (`NEARLINE`) and Retention Policy (`CUSTODIAL`) → “write this file to tape”.

3. **Obtain bearer tokens (“macaroon”):**
   - **Connect to source HTTP server and request a token.**
   - **Connect to destination HTTP server and request a token.**
4. Connect to **destination HTTP** server and request **HTTP-TPC**.
5. Monitor transfer’s progress.
6. Once transfer completes:
   - Release the pin on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**
7. (Optionally) monitor file being written to tape: **`srmLs`**



The good news

- **No changes** needed to StoRM and dCache:
  The deployed, production servers already support this.

- The SRM requests made by “FTS” are **largely unchanged**:
  - The SRM tape interactions (those that recall a file from tape or those that mean an upload will be written to tape) remain unchanged.
  - “FTS” must request an `https://` TURL instead of a `gsiftp://` TURL.

- **No changes** needed for the HTTP-TPC based transfer: this is what “FTS” does already.

The bad news

- Here, “FTS” is really composed of **two things**: the FTS transfer-agent and the gfal2 library.
  FTS transfer-agent handles ancillary activity, while gfal2 initiates the transfer.
- For HTTP-TPC, the **FTS transfer-agent** requests the “macaroons”.
  - To do this, it needs to know the `https://` TURLs.
  - The “macaroons” are then passed to gfal2, which initiates the transfer.
- For tape, the `https://` TURL is in the response to an srmPrepareToX request: **`srmPrepareToGet`** or **`srmPrepareToPut`**.
- The gfal2 library makes these srmPrepareToX requests.
- So, “macaroon”-based transfers are **currently not possible**.
  HTTP-TPC with X.509 delegation *should* work, but we (DOMA-TPC) decided not to use that.

The good news about the bad news

- Changes needed in FTS transfer-agent and gfal2 library are **simple**:
  - Largely consisting of moving code from transfer-agent into gfal2.
  - There is already a plan to do this.
  - This should be relatively **quick to implement**.

- There are fewer FTS servers (three) than production storage servers.
  Changes should be **easy to roll out**.

- Can deploy new FTS version on a test FTS instance, allowing us to test SRM-based tape transfers **quickly**.
  No need to wait for an upgrade to production FTS instances.

Open issue: what about CTA?

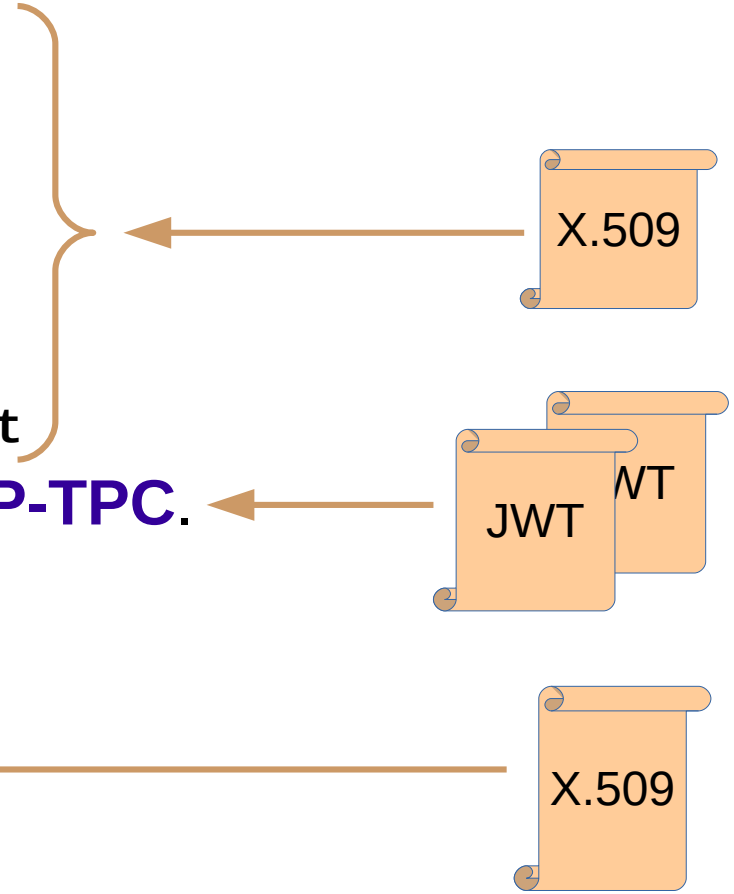
- Discussion has focused on SRM, which is **not supported by CTA**.
- For ATLAS and CMS, the plan is that tape files are first transferred from **CTA to an EOS instance**.
  - Transfers from EOS to Tier-1 are either DISK-to-DISK (direct HTTP-TPC) or DISK-to-TAPE (using SRM to manage tape side).
- For LHCb, the plan is to transfer data from **CTA to Tier-1 storage**.
- We would need **CTA to support HTTP-TPC**:
  - SLAC xrootd software (used in CTA) supports HTTP-TPC (hat tip to Brian)
  - Brian has offered to help update CTA's configuration to enable this.
- We would also need **FTS to stage files in CTA**.
  - This is only possible **through xrootd,** supported by FTS.
  - Would need FTS to support **stage-then-transfer**, similar to SRM

Part 2: a world without macaroons

… using VO-issued tokens (JWT from OpenID Provider; e.g., INDIGO-IAM) instead

Proposed HTTP-TPC solution: transferring tape to disk

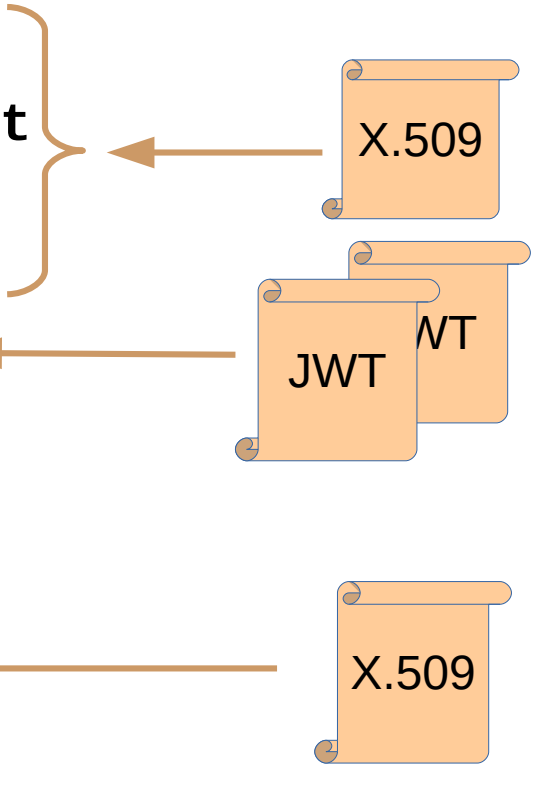
1. Stage file on **source** server to disk: **`srmBringOnline`**
2. Monitor stage request's progress:
   `srmStatusOfBringOnlineRequest`.
3. Once stage request completes:
   - Request a **HTTP** TURL on **source** server: **`srmPrepareToGet`**
   - Request a **HTTP** TURL on **destination** server: **`srmPrepareToPut`**
4. Connect to **destination HTTP** server and request **HTTP-TPC**.
5. Monitor transfer's progress.
6. Once transfer completes:
   - Release pins on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**



Purple marks parts that are different from today.

Proposed HTTP-TPC solution: transferring disk to tape

1. Request an **HTTP** TURL on **source** server: **`srmPrepareToGet`**
2. Request an **HTTP** TURL on **destination** server: **`srmPrepareToPut`**
   Specify the desired Access Latency (`NEARLINE`) and Retention Policy (`CUSTODIAL`) → "write this file to tape".
3. Connect to **destination HTTP** server and request **HTTP-TPC**.
4. Monitor transfer's progress.
5. Once transfer completes:
   - Release pins on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**
6. (Optionally) monitor file being written to tape: **`srmLs`**



Purple marks parts that are different from today.

The good news

- This is a **simpler process**, with fewer steps
  - No “macaroon” request step:
  - More standards compliant.

- **No development** needed in dCache and StoRM
  Production deployed software support JWT.

- **No development**(*) needed in FTS.
  - Unlike for the “macaroon” solution.
  - (*) we need to double-check this, to be sure. However, any work would be minimal.

- dCache and StoRM have been shown to **work well** with JWT.

The bad news

- Using JWT for HTTP-TPC is currently **an R&D activity**.
  - Potential for introducing problems.
  - Testing is an on-going activity.
  - No production experience (yet).
- Deployed production storage services would need to **update their configuration**.

The good news about the bad news

- **Current direction** of DOMA-TPC (independent of SRM/Tape "subtask")
  JWT is the future – we will get there.

- The SRM approach (for handling tape) is **compatible** with using JWT.
  When we get there, using SRM should work.

- If we want to, (at the risk of increased complexity) we can run both "macaroon" and JWT in parallel
  - FTS uses whichever method makes the most sense.
  - Somewhat mitigate the risks
  - Could allow a smooth transition.

Part 3: a world without X.509

… using VO-issued tokens (JWTs from OpenID Provider; e.g., INDIGO-IAM) instead.

SRM uses X.509 authentication

- We're trying to **move away** from X.509 authentication.
  - In DOMA-TPC, we are building up and and exercising a JWT testbed.
  - This is based on work from the WLCG AuthZ WG.

- Currently **all SRM requests** from FTS use an **X.509 credential** when authenticating with the SRM server.

- Although it would be possible to convert a bearer token into an X.509 credential (for SRM authentication) using an online CA, this **would be "awkward"**.
  - Online CA would need to be trusted by all SRM instances (c.f. IOTA vs CLASSIC),
  - This would be a lossy translation as X.509 (currently) has no AuthZ language (c.f. SciToken and AuthZ-JWT)

Little known secret #2: SRM isn't based on X.509

- SRM v2.2 specification is **authentication agnostic**.
  The term "X.509" (or "X509") simply doesn't appear in the specification!

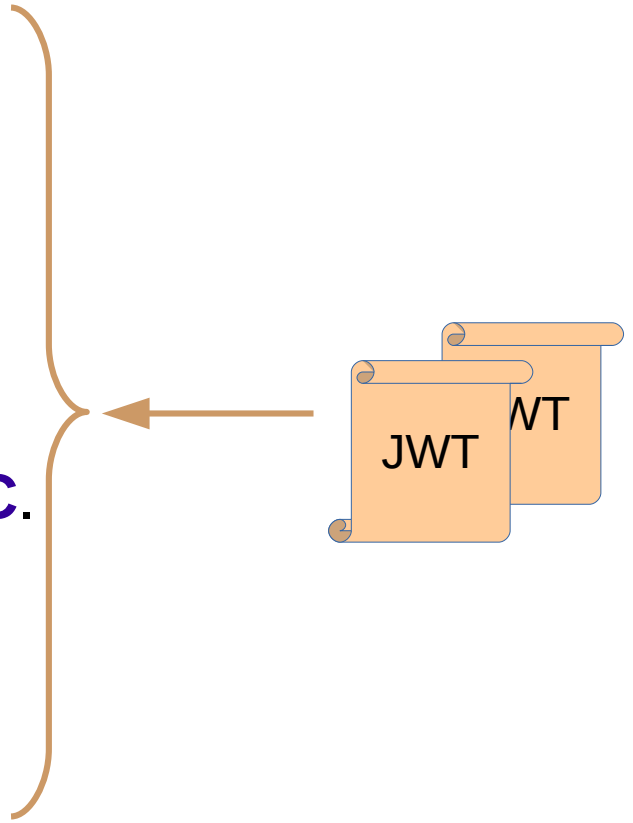
- There's a further binding in which **SRM operations** are mapped into **SOAP requests**.
  - In particular, SRM uses the very common **SOAP-over-HTTPS**.
  - In SOAP-over-HTTPS, the client makes (normal) HTTP POST requests.

- Currently SRM-using-SOAP-over-HTTPS requires client to authenticate using an **X.509 credential**.

- As SOAP-over-HTTPS is based on HTTPS, SRM can (in principle) use any authentication that HTTPS supports
  This includes our standard **JWT** approach.

Proposed HTTP-TPC solution: transferring tape to disk

1. Stage file on **source** server to disk: **`srmBringOnline`**
2. Monitor stage request's progress:
   `srmStatusOfBringOnlineRequest`.
3. Once stage request completes:
   - Request a **HTTP** TURL on **source** server: **`srmPrepareToGet`**
   - Request a **HTTP** TURL on **destination** server: **`srmPrepareToPut`**
4. Connect to **destination HTTP** server and request **HTTP-TPC**.
5. Monitor transfer's progress.
6. Once transfer completes:
   - Release pins on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**

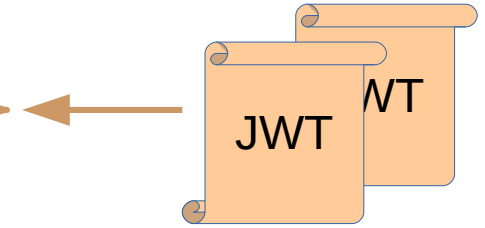


Purple marks parts that are different from today.

Proposed HTTP-TPC solution: transferring disk to tape

1. Request an **HTTP** TURL on **source** server: **`srmPrepareToGet`**
2. Request an **HTTP** TURL on **destination** server: **`srmPrepareToPut`**

   Specify the desired Access Latency (`NEARLINE`) and Retention Policy (`CUSTODIAL`) → “write this file to tape”.
3. Connect to **destination HTTP** server and request **HTTP-TPC**.
4. Monitor transfer’s progress.
5. Once transfer completes:
   - Release pins on **source** server: **`srmReleaseFile`**
   - Finalise upload on **destination** server: **`srmPutDone`**
6. (Optionally) monitor file being written to tape: **`srmLs`**



Purple marks parts that are different from today.

The good news

- **Operational procedures** continues largely as before, just with a different authentication scheme.

- This should be a **low risk strategy**, as the protocol stays the same.
  Storage should react the same way whether the client authenticates with X.509 or a bearer token.

The bad news

- Requires **development effort**: both on the SRM server (dCache, StoRM) and SRM client (FTS/gfal2/...).

- Supporting bearer-token SRM requires **updating both** production storage services and FTS instances.

- The roll-out strategy is an **open question**:
  - An upgrade campaign, switching over FTS to JWT-SRM once complete.
  - Run in a mixed-mode: some SRM endpoints are X509-only, others also support JWT-SRM.

- Possible **FTS strategies**, when given both JWT and X.509
  - Try JWT. If that fails then try X.509.
  - Try both X.509 and JWT in same request.
  - Use only the specified authentication (X.509 or JWT)

The good news about the bad news (part 1.)

- dCache code has **already been updated** to support non-X.509 SRM.
  - Part of the next major release (v7.0).
  - Already deployed on `prometheus.desy.de`.
  - Supported bearer tokens include: OpenID-Connect, Macaroons, SciTokens, **AuthZ-JWT**.
  - Support is enabled out-of-the-box: **no configuration changes** are needed.

- **StoRM** plans to update SRM to support bearer-token authentication as part of their work to remove Globus dependency.

- The dCache **SRM-clients** have already been updated to support bearer-token (as a proof-of-principle).
  - Changes have passed code-review.
  - Unreleased (“test”) snapshot available for people with a keen interest.

The good news about the bad news (part 2.)

- The changes in FTS should be **relatively easy**
  FTS transfer-agent ultimately uses **gsoap** for SRM interactions, which **already supports bearer tokens**, and provides good documentation how to use them.

- Some sites are **fast at deploying** new versions:
  dCache → NDGF, SurfSARA, …
  StoRM → INFN/CNAF, …
  We may not have to wait long before there are some production instances with which we can demonstrate non-X.509 SRM in production.

- Although this requires a (small) amount of development work, we **save time and expense** by not developing a new protocol to manage tape, but just continue using SRM.

Summary

- SRM based transfers involving tape **can work with HTTP-TPC**:
  - Based on what we do now with SRM: good operational knowledge.
  - No changes needed to deployed storage services (when using X.509-SRM + “macaroons”).
  - Only minor code-changes needed to FTS (when using “macaroons”).
  - Would still use X.509 to authentication with SRM servers.
- SRM base transfers **without X.509** is possible, requires more work:
  - dCache v7.0.0 will support this; similar support planned for StoRM.
  - Some additional work needed within FTS/gfal, still relatively minor.
- Adapting SRM should be **faster and lower cost** than creating a new protocol.
- Need to **verify** that these solutions will work “in the real world”:

  Both testing production instances in general, and with CTA endpoints in particular.

Thank you

Contact

DESY. Deutsches
Elektronen-Synchrotron

www.desy.de

Paul Millar
Research and Innovation in Computing (RIC),
IT department,
DESY
paul.millar@desy.de