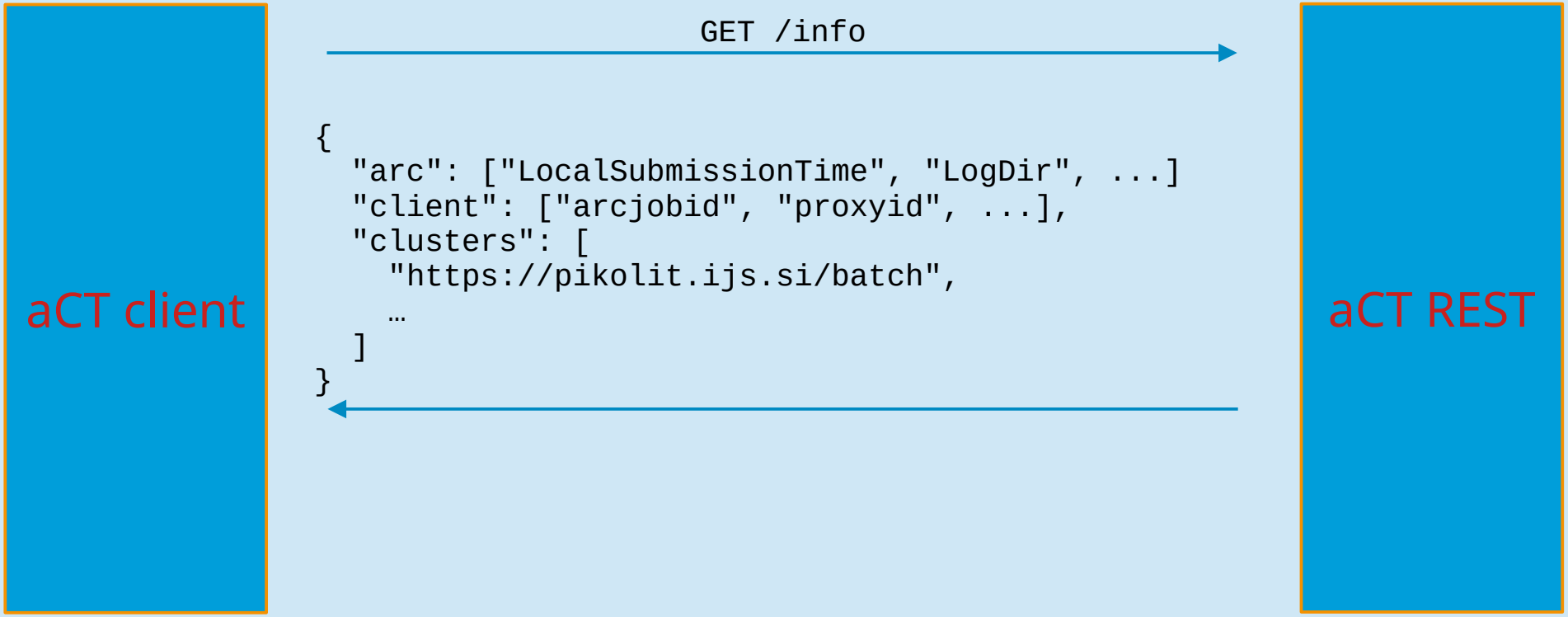
Report on development of aCT REST, client and core

Jakob Merljak

Info

```
$ act info
aCT server URL: https://act.vega.izum.si
Clusters:
https://pikolit.ijs.si/batch
https://rebula.ijs.si/batch
https://arc01.vega.izum.si/cpu
https://arc02.vega.izum.si/cpu
https://arc01.vega.izum.si/largemem
https://arc02.vega.izum.si/largemem
https://hpc.arnes.si/all
https://nsc.ijs.si/gridlong
```

Info


aCT client
GET /info
{
"arc": ["LocalSubmissionTime", "LogDir", ...]
"client": ["arcjobid", "proxyid", ...],
"clusters": [
"https://pikolit.ijs.si/batch",
…
]
}
aCT REST

Authentication

```
$ act proxy
Successfully inserted proxy. Access
token stored in
/afs/f9.ijs.si/home/jakobm/.local/share/
act-client/token
```

Authentication

Job status

```
$ act stat -a
id    jobname  JobID                                                State    arcstate
-------------------------------------------------------------------------------------
33940 arctest1 https://pikolit.ijs.si:443/arex/jAXMDmMa85... Finished done
33941 arctest2 https://pikolit.ijs.si:443/arex/2INODmMa85... Finished done
33942 arctest3 https://pikolit.ijs.si:443/arex/O6mLDmNa85... Finished done
33943 arctest4 https://pikolit.ijs.si:443/arex/s1pNDmNa85... Finished done
```

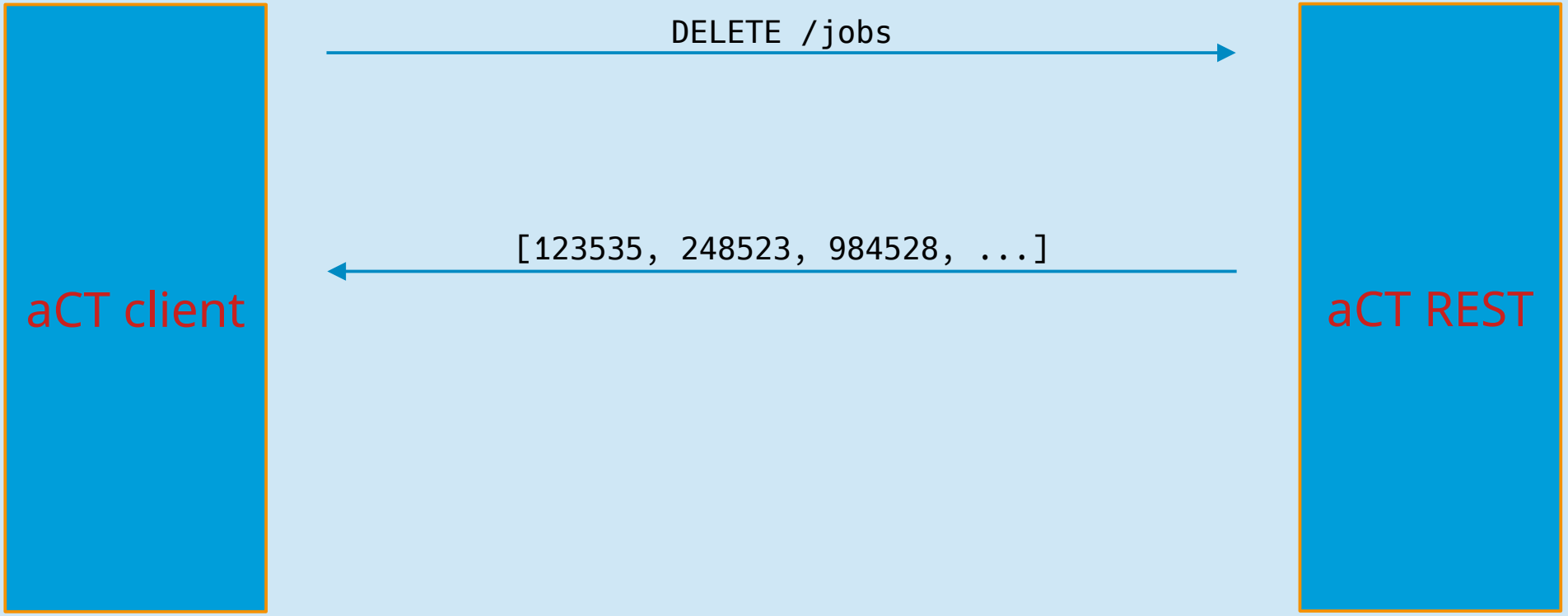
Job status


aCT client
GET /jobs
[
{
"a_JobID":
"https://pikolit.ijs.si:443/arex/jAXMDmMa850n...",
"a_State": "Finished",
"a_arcstate": "done",
"c_id": 33940,
"c_jobname": "arctest1"
},
...
]
aCT REST

Job clean

```
$ act clean -a
Cleaned 4 jobs
Cleaning WebDAV directories ...
```

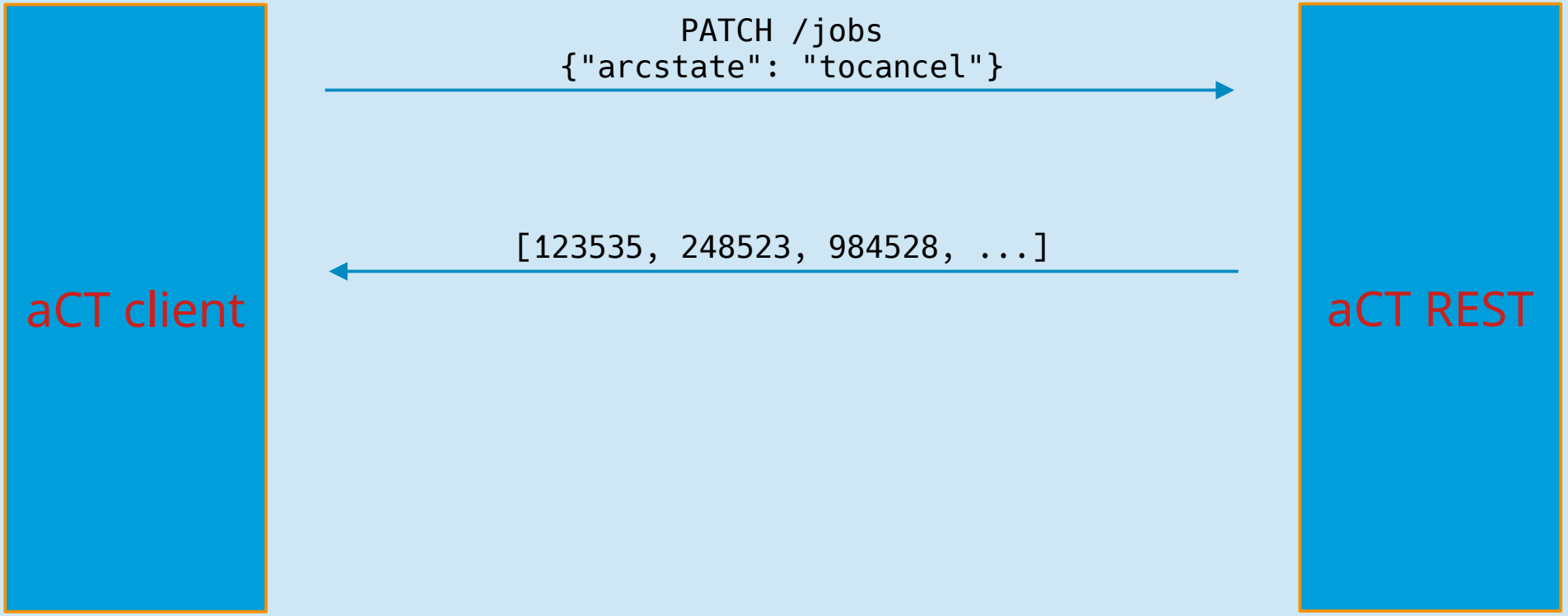
Job clean



Job kill

```
$ act kill -a
Will kill 4 jobs
```

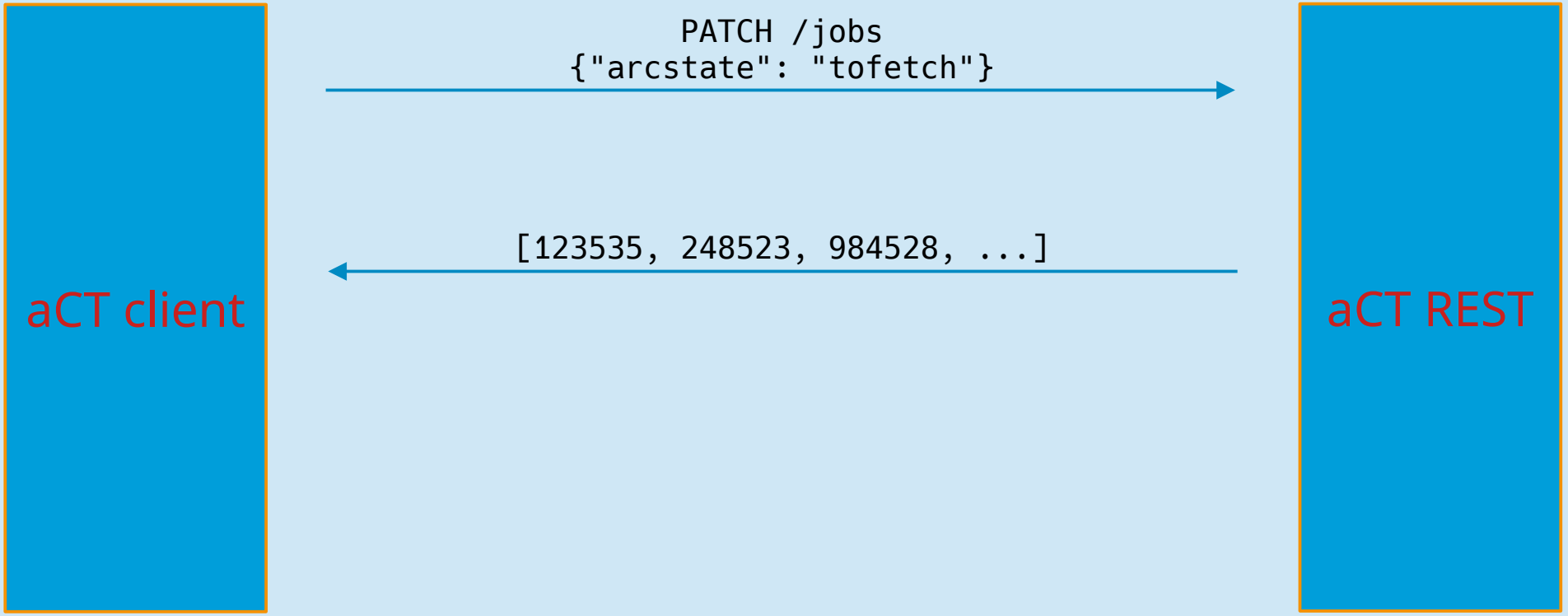
Job kill


aCT client
PATCH /jobs
{"arcstate": "tocancel"}
[123535, 248523, 984528, ...]
aCT REST

Job fetch

```
$ act fetch -a
Will fetch 4 jobs
```

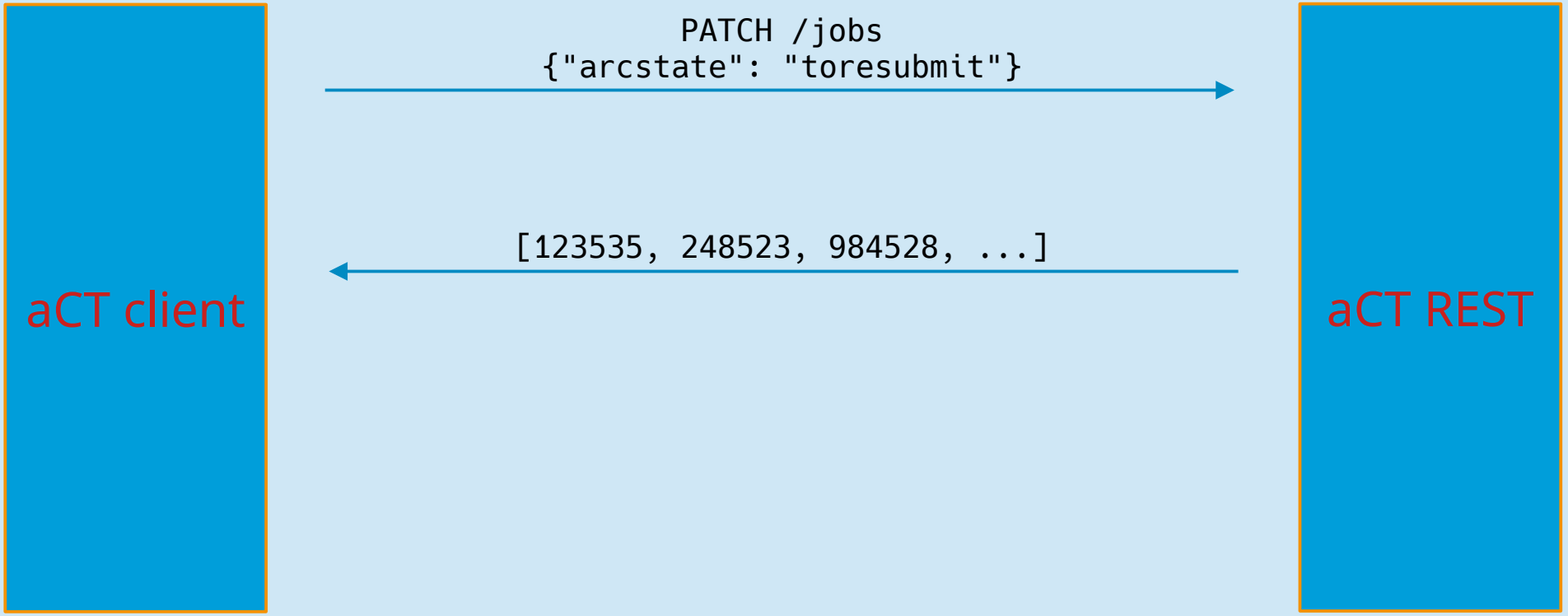
Job fetch


aCT client
PATCH /jobs
{"arcstate": "tofetch"}
[123535, 248523, 984528, ...]
aCT REST

Job resubmit

```
$ act resub -a
Will resubmit 4 jobs
```

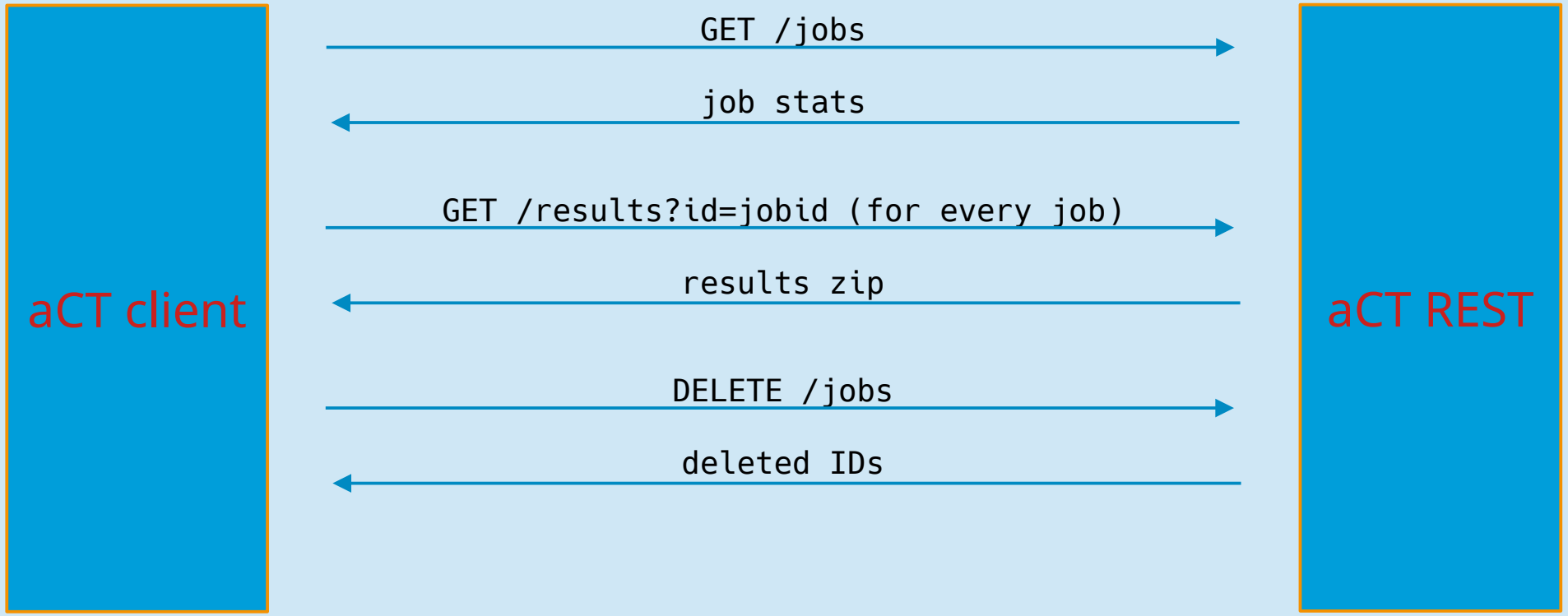
Job resubmit



Job get

```
$ act get -a
Results for job arctest1 stored in EXGODmTLR60nKS8tmqmuIMjmiDKYNnABFK...
Results for job arctest2 stored in 9XhMDmULR60nKS8tmqmuIMjmiDKYNnABFK...
Results for job arctest3 stored in in5MDmWLR60nKS8tmqmuIMjmiDKYNnABFK...
Results for job arctest4 stored in sjoKDmXLR60nKS8tmqmuIMjmiDKYNnABFK...
Cleaning WebDAV directories ...
```

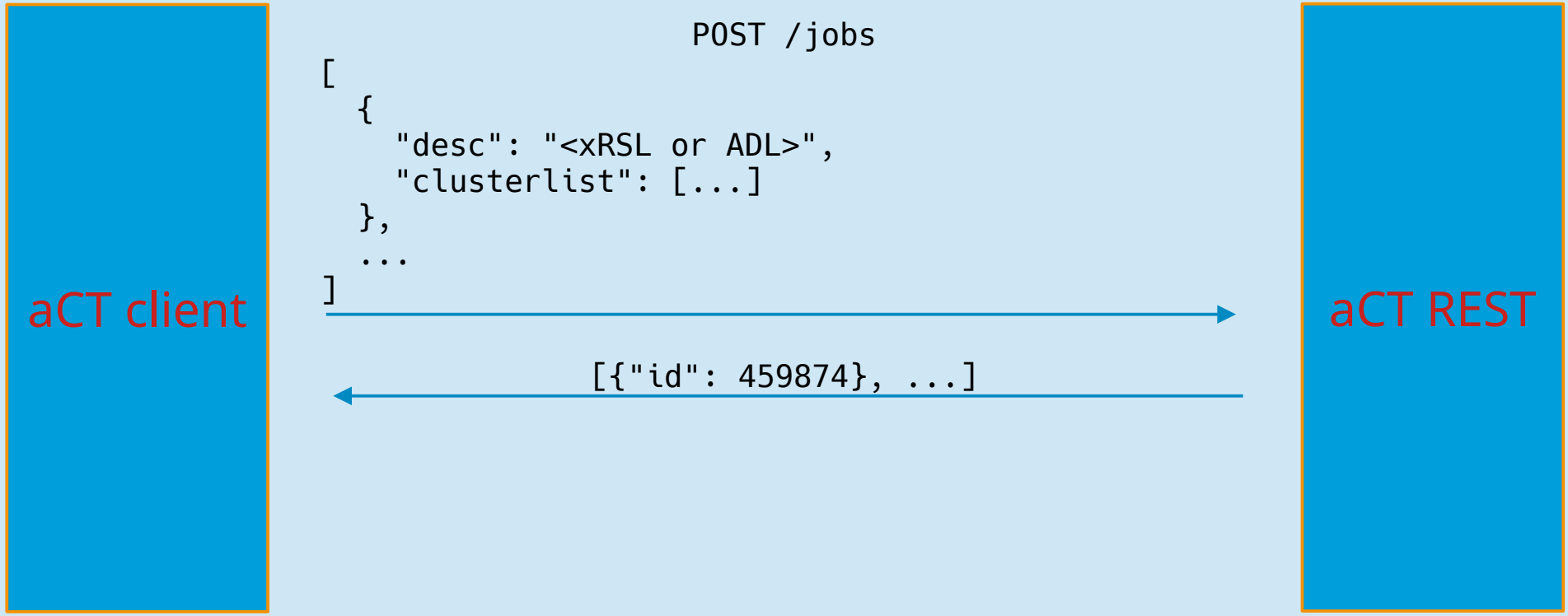
Job get


aCT client
GET /jobs
job stats
GET /results?id=jobid (for every job)
results zip
DELETE /jobs
deleted IDs
aCT REST

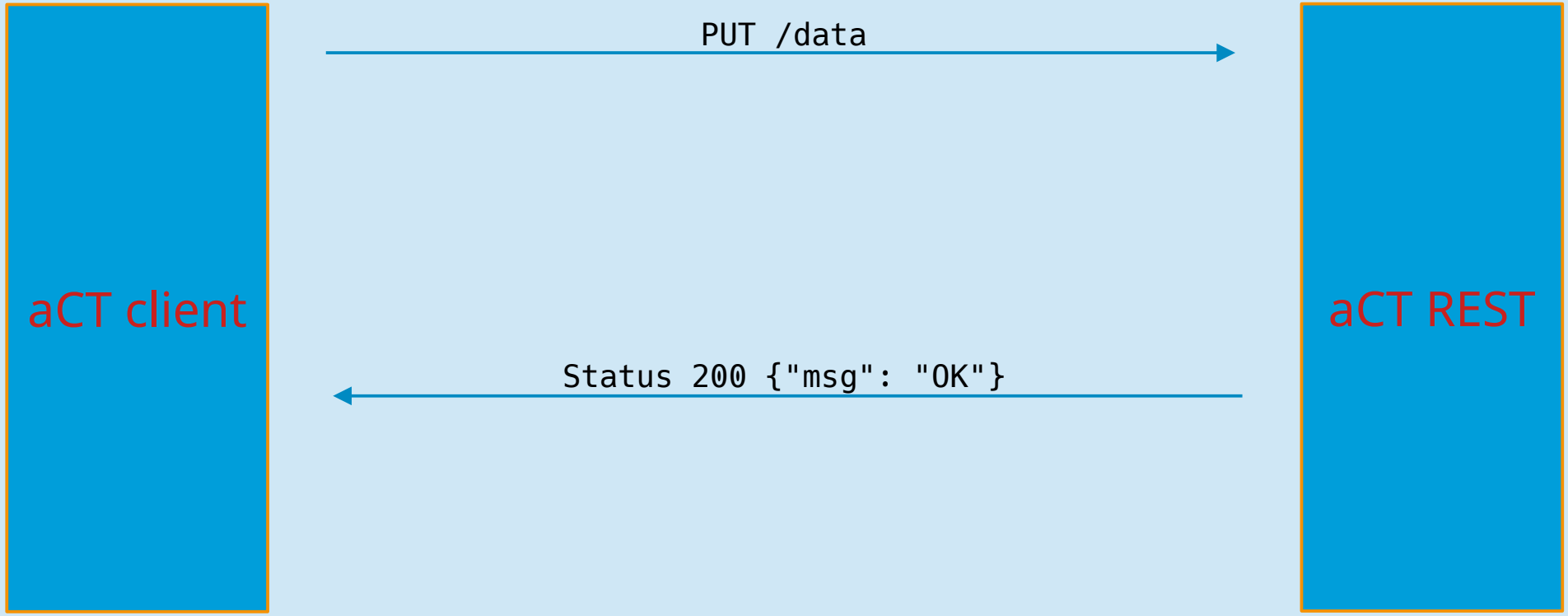
Job submit

```
$ act sub arctest1.xrsl arctest2.xrsl arctest3.xrsl arctest4.xrsl
Inserted job arctest1 with ID 33948
Inserted job arctest2 with ID 33949
Inserted job arctest3 with ID 33950
Inserted job arctest4 with ID 33951
```

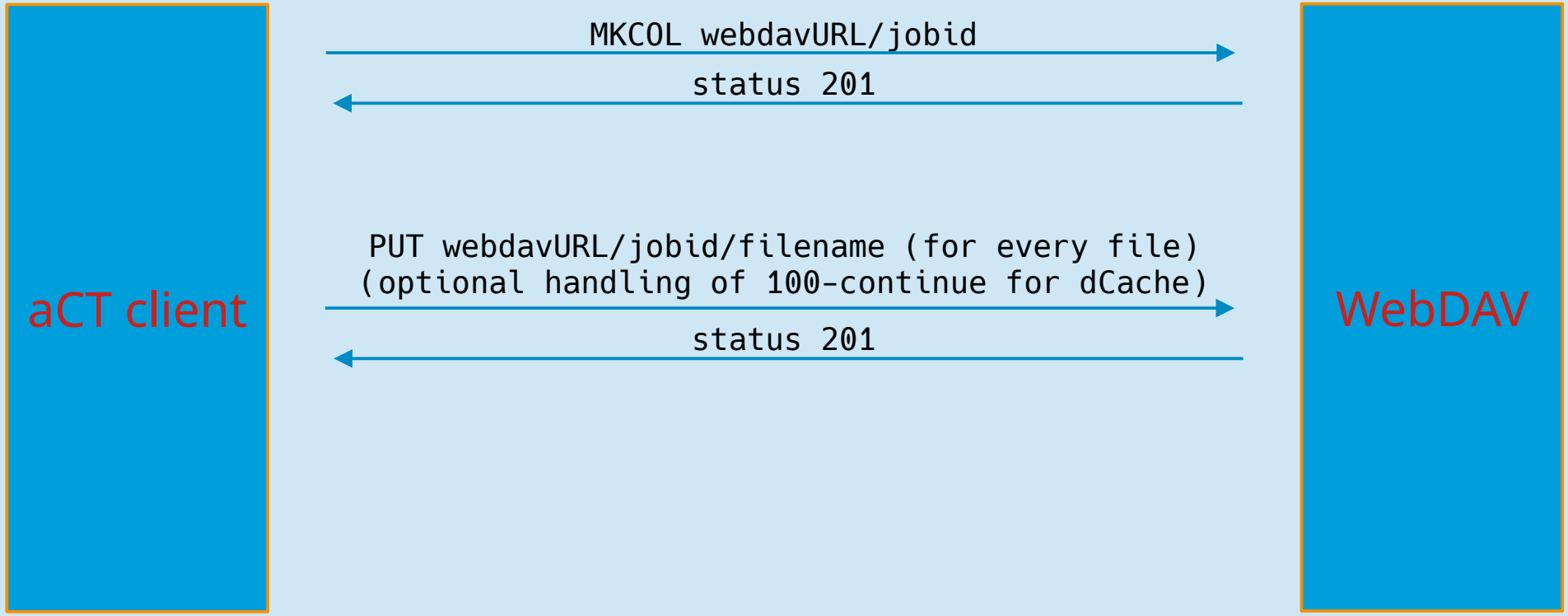
Job submit (step 1)



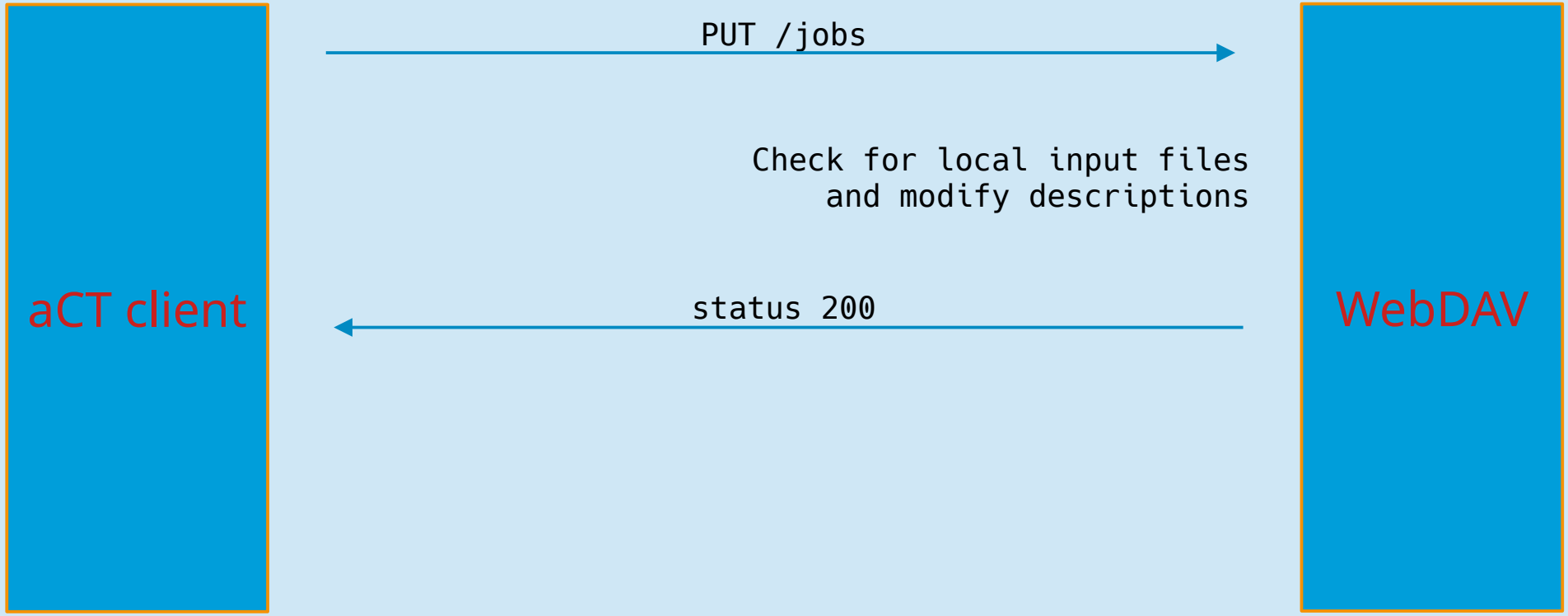
Job submit (step 2: upload to aCT)



Job submit (step 2: WebDAV)



Job submit (step 3)



- WebDAV integration is done entirely in aCT client
- kill, clean, get and submit operations must not be interrupted in certain stages to ensure proper cleanup of WebDAV files

aCT ARC REST

HTTPClient class

- Uses http.client library from python standard library
- Provides convenience method request() that handles sending JSON, encoding URL parameters, auth tokens, transparent use of HTTP or HTTPS protocol and request retries for specific error conditions
- Optionaly stores path to proxy certificate used to authenticate connection

RESTClient class

- Uses HTTPClient object for HTTP communication
- Provides higher level job management operations using REST interface

RESTClient class

- createDelegation() → delegationID
- renewDelegation(delegationID)
- deleteDelegation(delegationID)

RESTClient class

- Jobs are represented as dictionaries. A dedicated class should rather be created once all relevant job attributes for API is known
- getJobsInfo(jobs) – adds info document in parsed JSON format to every job (POST /jobs?action=info)
- getJobsStatus(jobs) – adds key with status value to every job (POST /jobs?action=status)

RESTClient class

- killJobs(jobs) – kills given jobs; error string is added to jobs that cannot be killed (POST /jobs?action=kill)
- cleanJobs(jobs) – cleans given jobs; error string for failed (POST /jobs?action=clean)
- restartJobs(jobs) – restarts jobs; error string for failed (POST /jobs?action=restart)
- getJobsDelegations(jobs) – adds delegation ID to jobs (or error string)

RESTClient class

- getJobsDelegations(jobs) – adds delegation ID to jobs (or error string)

RESTClient.submitJobs

1) Create delegation
2) Parse descriptions, add queue and delegation, unparse
3) Bulk submit
4) Upload local input files

Local input file upload

1) Put all local input files to queue (thread safe) along with their job ID to be able to identify failed uploads later
2) Run upload worker threads
3) If upload fails add error string to job dict

RESTClient.fetchJobs

- Spawns workers that recursively download files to a given directory using work queue
- Filtering mechanism for downloads is currently aCT specific

RESTClient class

- On errors, the given API lets through most exceptions so that the developer can handle them as needed (systematic documentation needed!!!)
- aCT ARC Engine now uses RESTClient that does all interaction with ARC CE
- ARC engine is now concerned with preparing the job dicts for RESTClient operations, calling those operations, handling their errors and updating the DB accordingly

Notes

- RESTClient, HTTPClient are not yet finished and consolidated with relevant parts of aCT client
- aCT switch to interaction with ARC CE over REST seems to be working but it is not finished yet