Architecture and components of ARC

Oxana Smirnova

Outline of the lecture

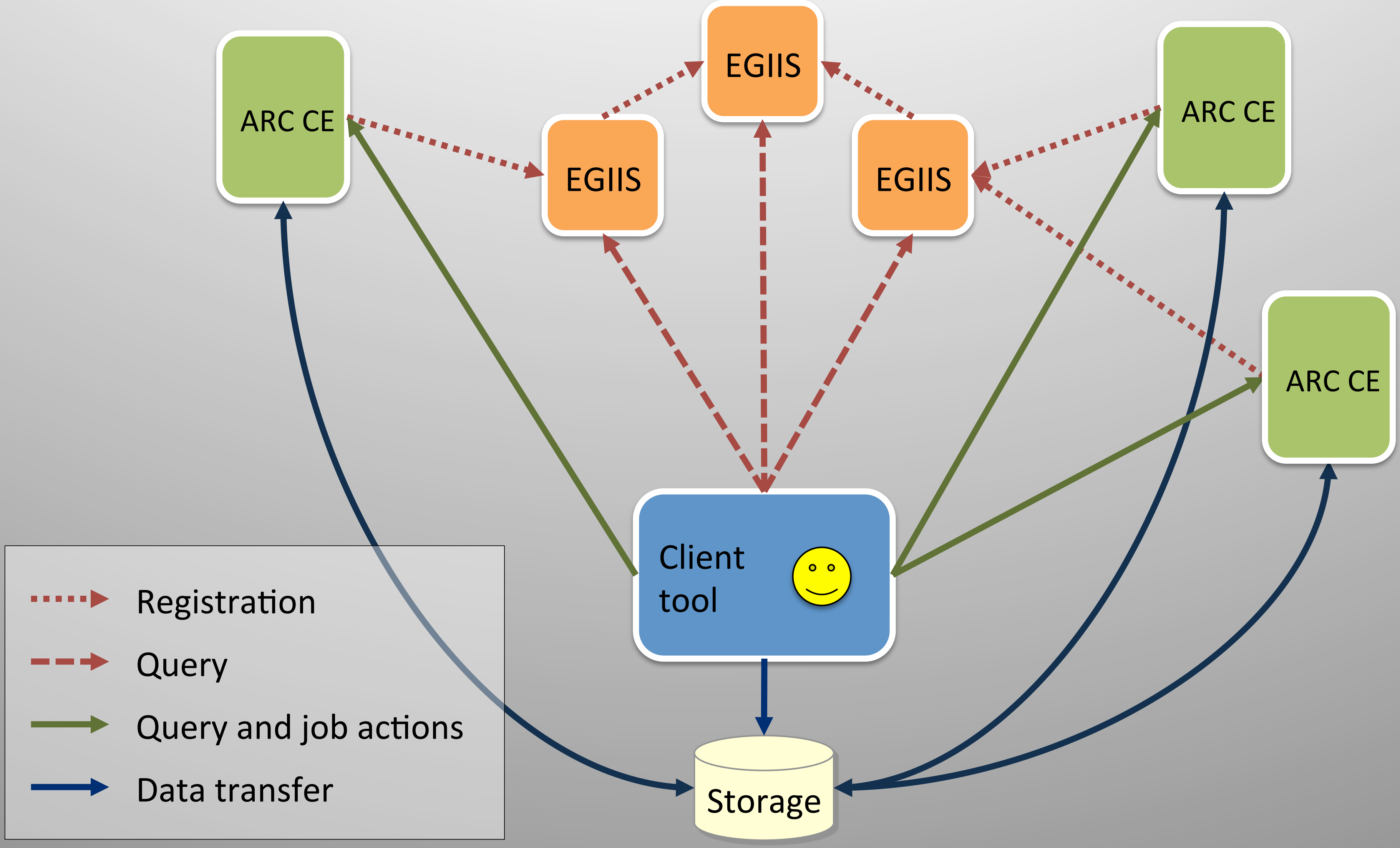
- Overview and key concepts of ARC
- ARC components and functionalities
- Computational jobs and environments

OVERVIEW AND KEY CONCEPTS

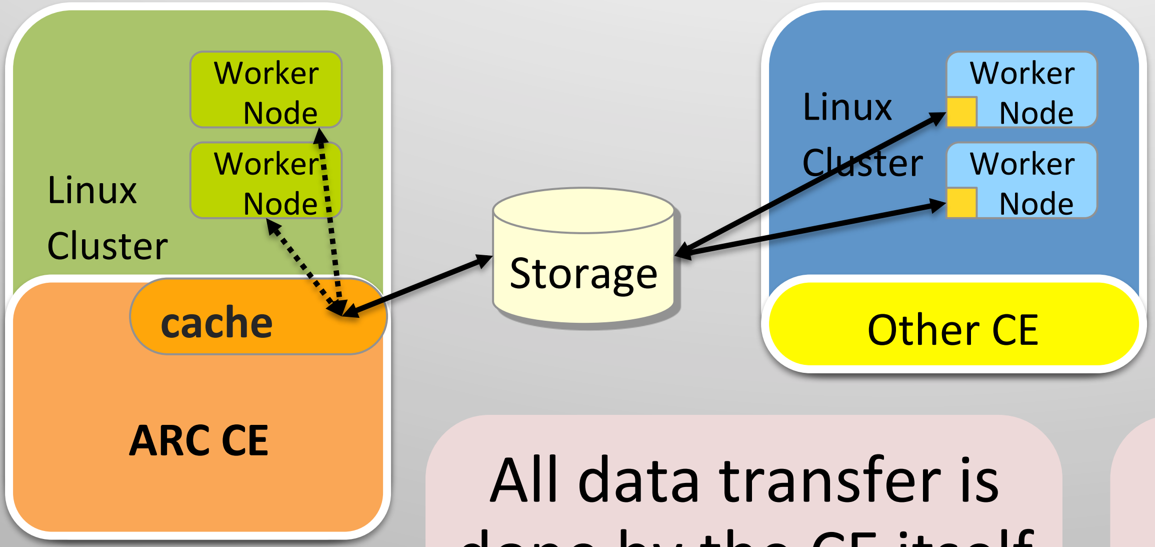
Components

- Front-end to computing resources: the **ARC Compute Element (CE)**
  - CE is a set of services and modules
    - Authorisation and access control
    - Job handling
    - Job files handling (input/output data)
    - Information handling
    - Accounting
- **Resource indexing server (EGIIS)**
  - A lightweight LDAP-based server, probably to be replaced by a 3rd party one (EMIR)
- **Client tools**
  - Built upon libraries, some shared with the CE

High-level ARC Grid architecture



ARC CE key concept: optimized for data-intensive jobs



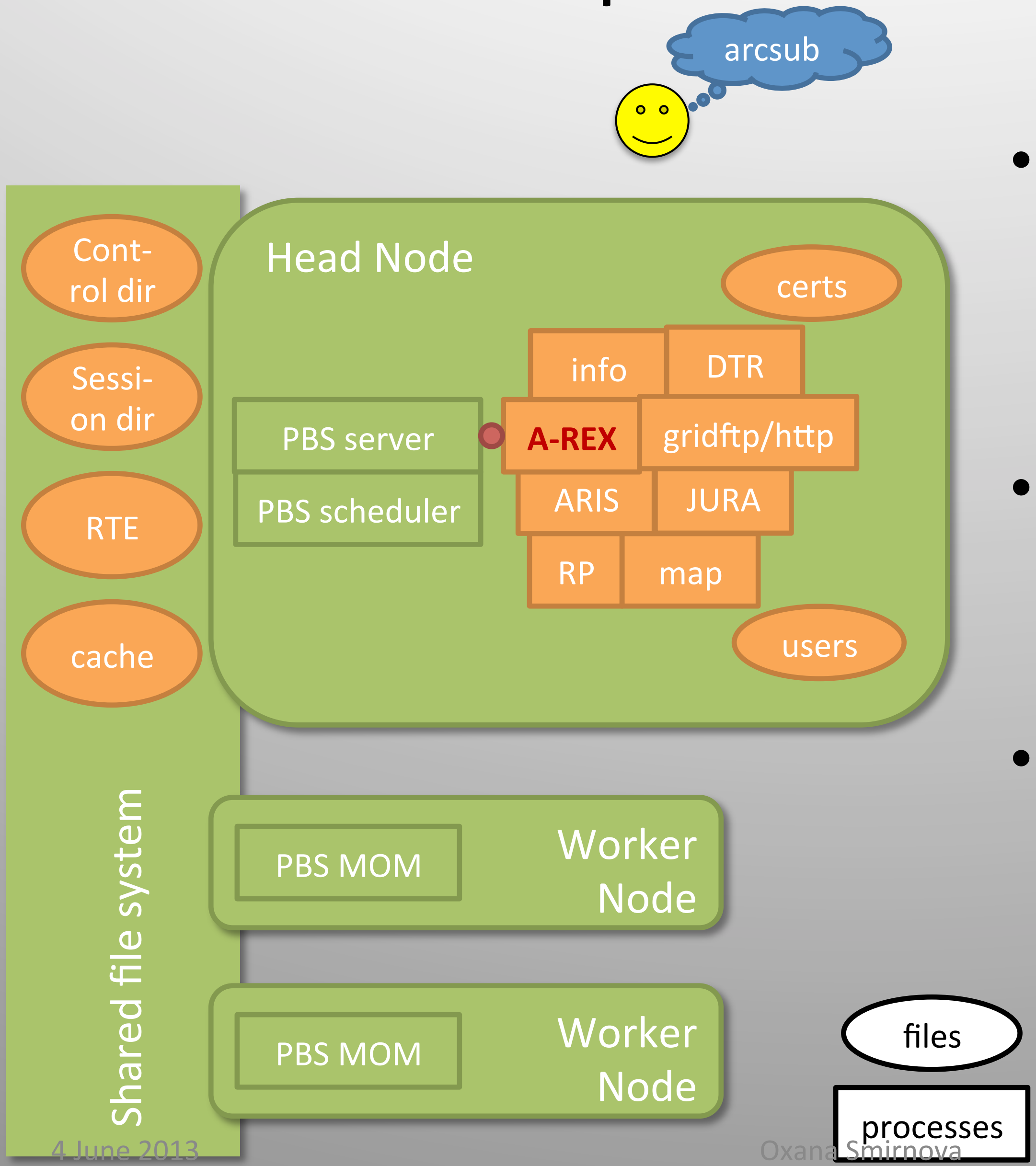
All data transfer is done by the CE itself

- Allows to cache frequently used files
- Minimizes bandwidth
- Maximizes WN usage efficiency

ARC CE is a very complex service

- Is built of many individual services and tools
- Requires high-end storage for cache

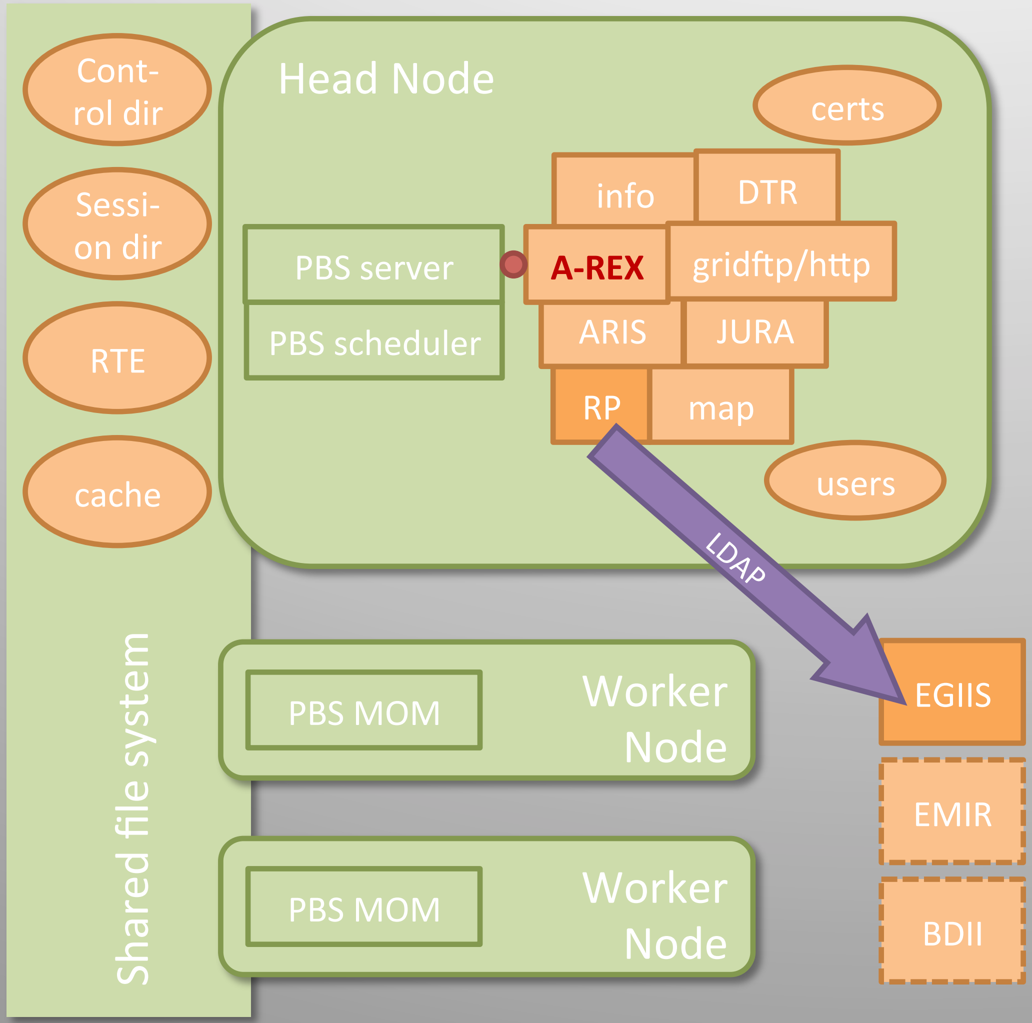
ARC CE components on a (PBS) cluster



- Interfaced to a number of batch systems
  - SLURM, SGE, PBS, LSF, LL, Condor
- ARC CE is a uniform interface: batch system specifics are not exposed
- No ARC component is installed on worker nodes
  - No need, because the CE handles transfers

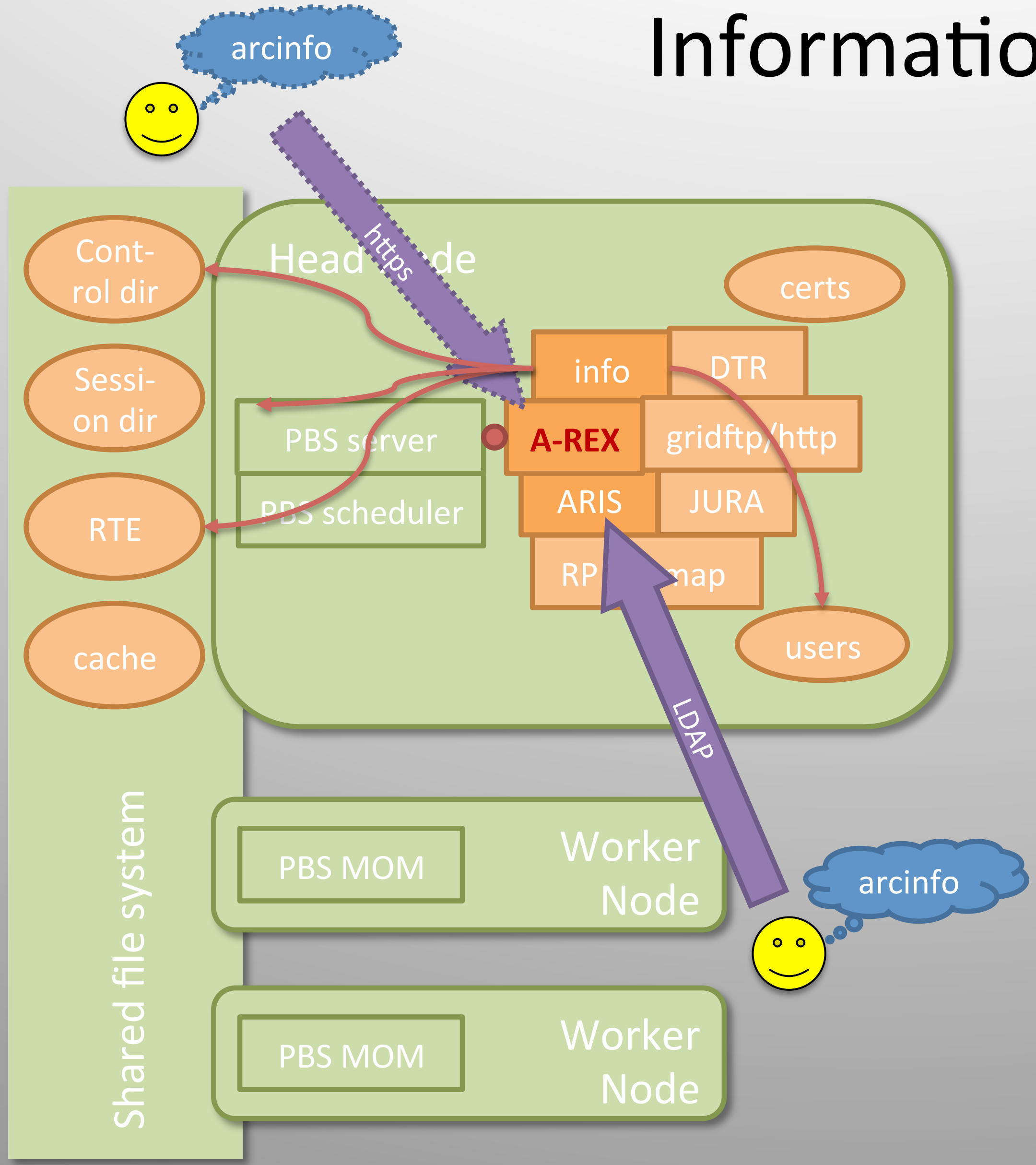
COMPONENTS AND FUNCTIONALITIES

Registration



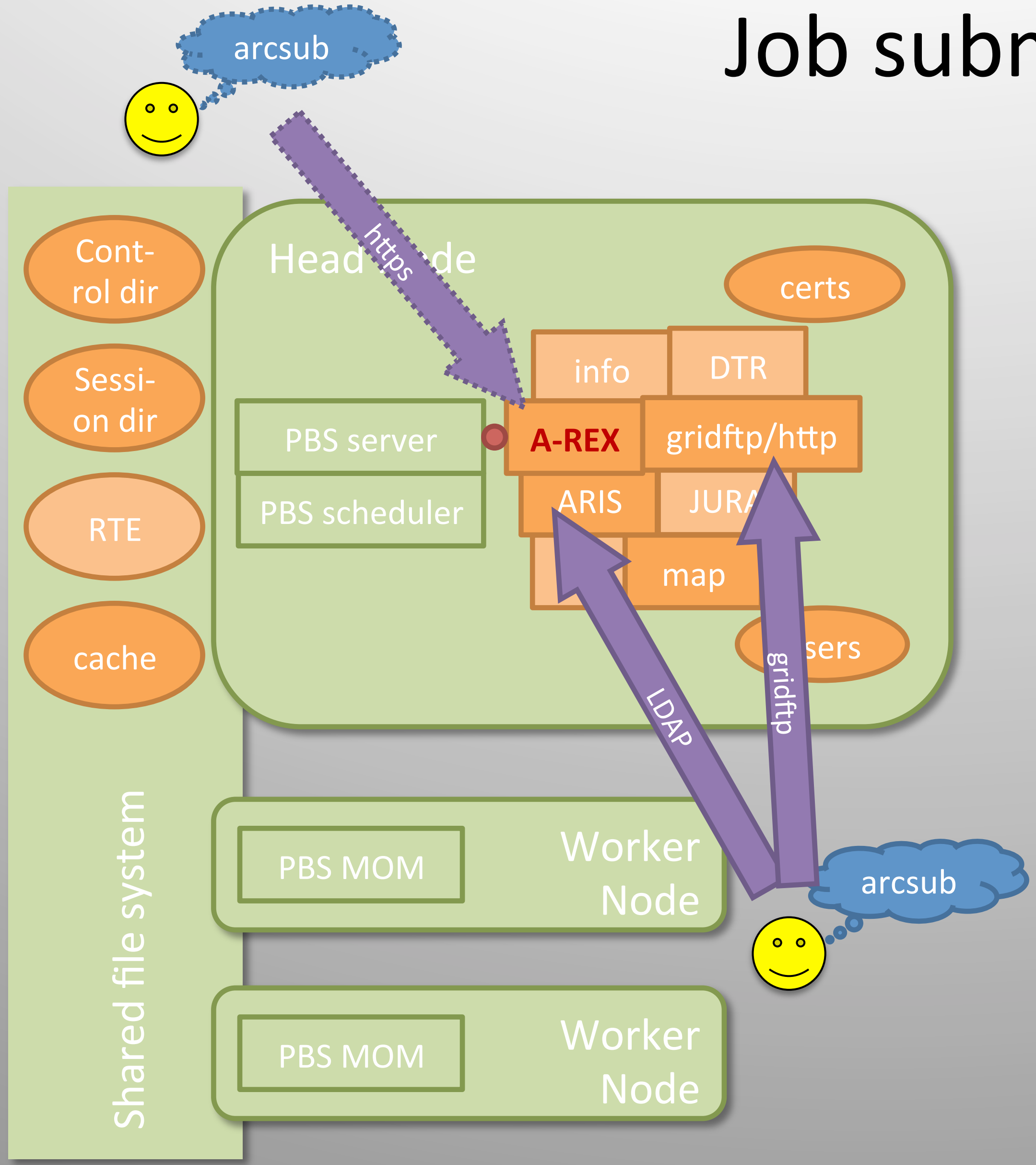
- Information Registration Process periodically sends pre-configured information to one or more pre-configured information registries
  - Service type (cluster in this case)
  - Service contact details (contact string, port etc)
- Currently all such data are communicated via LDAP
  - Other technologies are possible, as primary data are stored internally as regular files

Information publishing



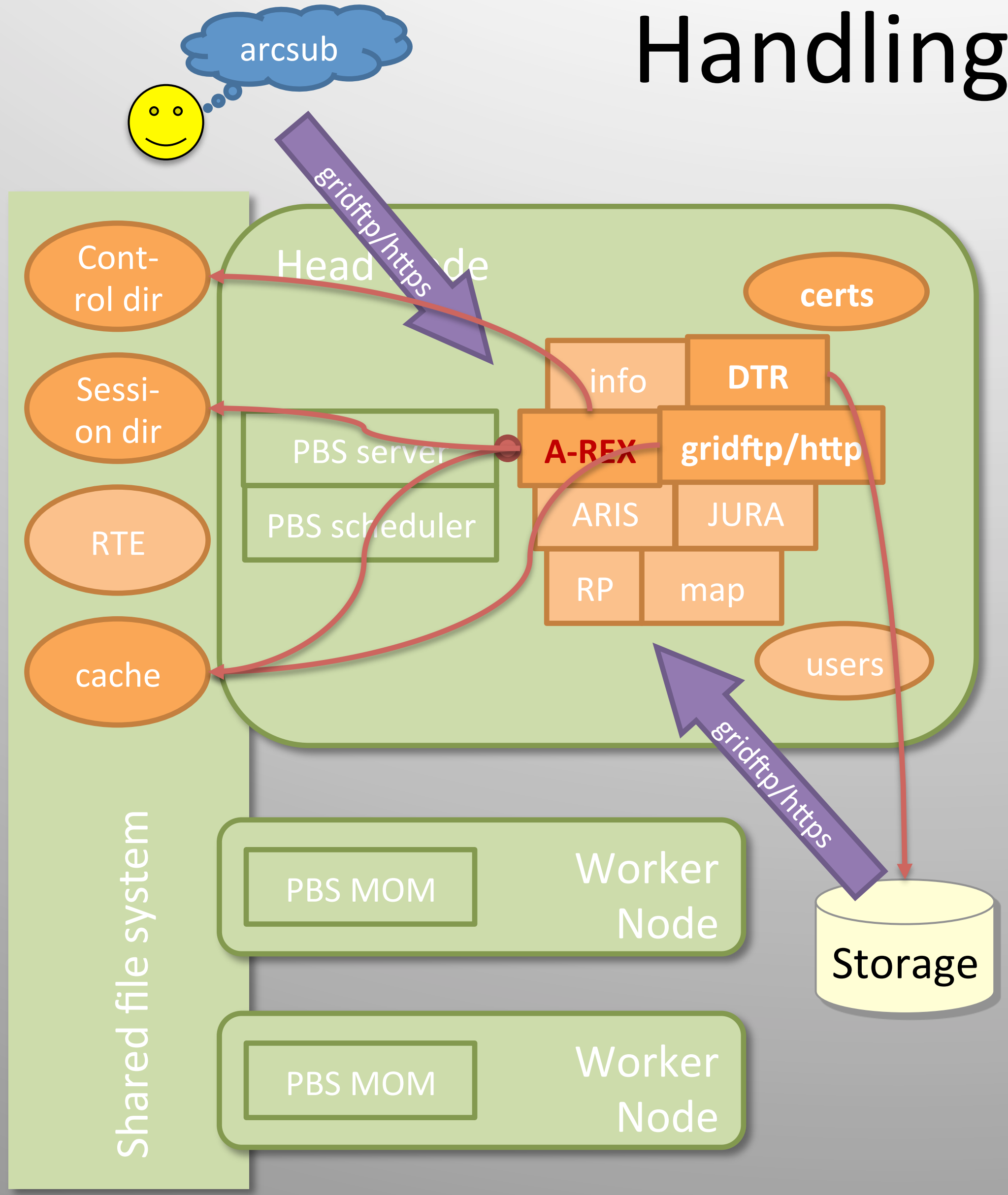
- A-REX periodically launches information providers which:
  - Collect all details defined by relevant information schemas, such as
    - Hardware details
    - LRMS details
    - Available application software (RTE)
    - Authorised users (DNs)
    - etc
  - Create formatted output ready to be served on request
  - Populate ARIS databases
- ARIS serves information when queried via LDAP
- Same information can be obtained by a WS query to A-REX
  - Not a default configuration yet

Job submission



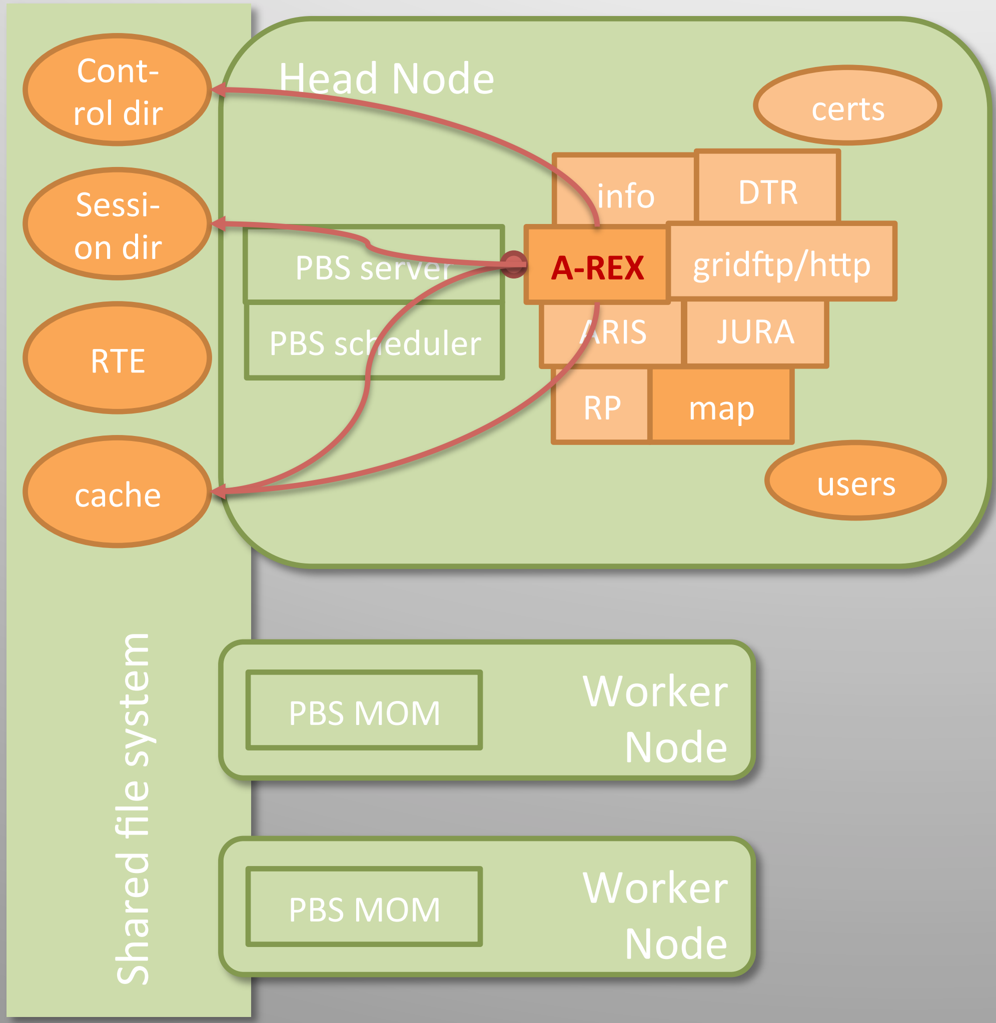
- **Client** tool must:
  - Query information
  - Match it to the job description document
  - Select the best site
  - Convert to a server document (deterministic)
  - Upload all the files
- **A-REX** discovers uploaded job files and launches job processing
- Currently, information and upload use different protocols
  - WS is needed for better consistency
- All steps require **authorisation**

Handling file transfers



- Jobs won't start before all input files are present
- Input files provided by the user are uploaded by the client tool
  - normally, cached
- External files are downloaded by **DTR** when triggered by A-REX
  - also cached by default
- All inputs are copied or linked to the session directory
- Output files are uploaded by DTR to external storage if requested

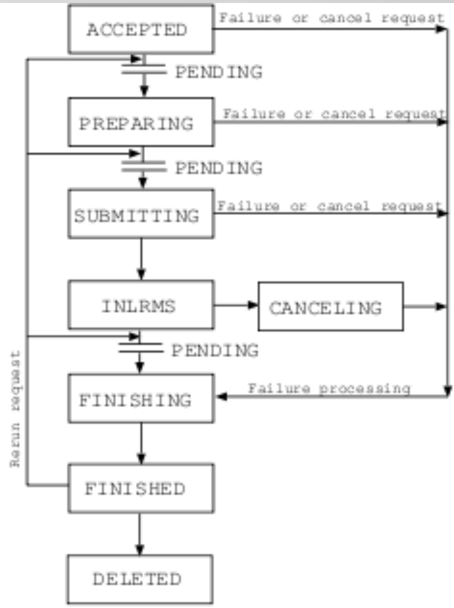
Job submission to the batch queue



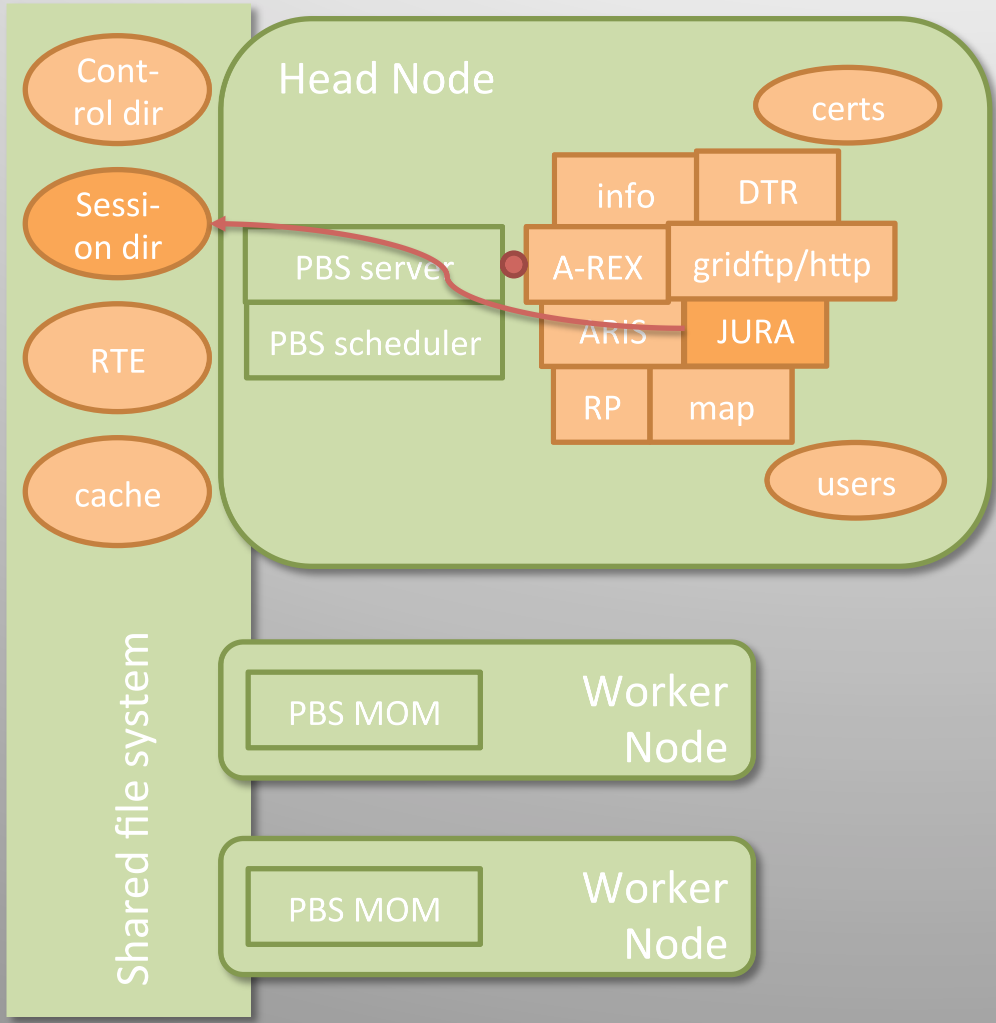
- Key component: batch "back-ends"
  - Encapsulate specific properties of different batch systems and map them to generic functionalities
- A-REX handles the job life cycle
  - Sends them to the batch queue via back-ends
  - Monitors status
  - Triggers data movement
  - Authorisation

Job handling by A-REX

Job state	Description
Accepted	the job has been submitted to the CE but hasn’t been processed yet
Preparing	input data are being gathered
Submitting	job is being submitted to the LRMS
Executing (InLRMS)	job is queued or being executed in the LRMS
Killing (Canceling)	job is being canceled
Finishing	output data are being processed (even if there was a failure)
Finished	job is in this state when either it finished successfully or there was an error during one of the earlier steps
Deleted	after specified amount of days the job gets deleted and only minimal information is kept about it



Accounting



- JURA harvests job information and submits it to an external accounting service
  - For completed jobs only

COMPUTATIONAL JOBS AND ENVIRONMENTS

Client interprets job description

- 37 attributes to find best matching resource
- Job description language: xRSL, JSDL or JDL, but internally is ADL-like

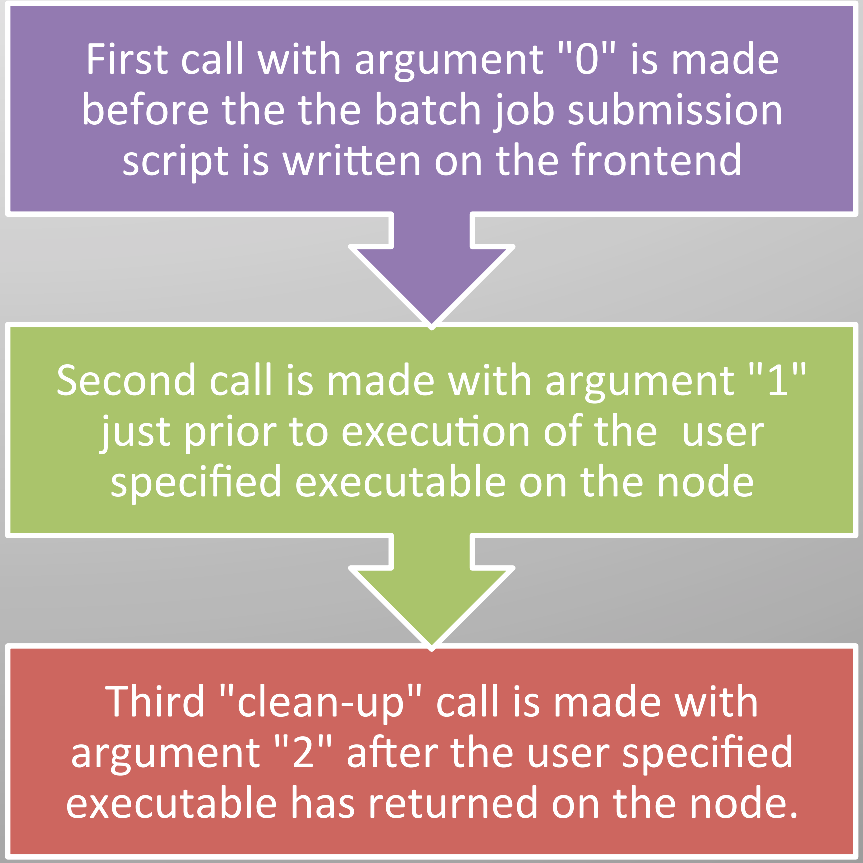
Job attribute	Example
Application environments	/APPS/HEP/ATLAS-20.1.0.1 /ENV/GLITE /ENV/FULLNODE
Main executable (binary or script)	Findhiggs.py
Arguments of executable	-i input.root -o output.root
Input files	srm://srm.infn.it/atlas/2012/file1.root
Output files	srm://srm.ndgf.org/atlas/ulf/higgs.root
Number of slots per job*	36
Queue name (**bad** practice)	mpi_jobs
Time (or benchmark), memory, disk	*numbers (or benchmark name, e.g. HS06)*
Standard input/output/error	stdout.txt
and many others	

** Interpretation of "slot" depends on batch system!*

Runtime Environment (RTE) concept

- Application environment is formalised as “Runtime Environment” (similar concepts exist in other CEs)
- Runtime Environment can encapsulate not just application software, but also:
  - Batch system peculiarities
  - Hardware aspects
  - Can even emulate gLite WN
- It is just a shell script
  - Created manually by sysadmins
  - Advertised via infosys
  - For many RTEs, namespaces are handy

Requested RTE script is called by A-REX 3 times:



First call with argument "0" is made before the the batch job submission script is written on the frontend

Second call is made with argument "1" just prior to execution of the user specified executable on the node

Third "clean-up" call is made with argument "2" after the user specified executable has returned on the node.

RTE example: how to make jobs to use 4 cores per node (for PBS)

Good way: RTE script

```
user@host# cat RESERVE_4_CORES
#!/bin/sh
case "$1" in
0)
      export joboption_nodeproperty_0="ppn=4"
      ;;
1)
      # Nothing to do
      ;;
2)
      # Nothing to do
      ;;
*)
      return 1;;
esac
```

- Add line to job description file: (runtimeenvironment= "RESERVE_4_CORES")
- As many RTE scripts as needed

Bad way: special queue

```
user@host# cat arc.conf
.....
[queue/mpi_jobs]
.....
queue_node_string="ppn=4"
```

- Add line to job description file: (queue=mpi_jobs)
- And so on, introduce a new queue for every imaginable configuration

RTE benefits

- In practice is the only way to make parallel/multi-core applications work in heterogeneous clusters
- No need to transfer executable, loader or libraries
- Possibility to build clusters which allow execution of specified applications only
- Better application performance with architecture specific optimizations
- Initialization of environment variables and paths, i.e. providing standard environment for executables submitted by user
- Version management

- Logistical problem:
  - Who/how keeps track of all RTEs?
    - Currently, just a Web page

Summary

- ARC CE is a complex modular service
  - The modules usually have no stand-alone meaning and can not be used with other services (except for perhaps DTR)
- ARC client tools perform simple actions, but have very complex internal logic
- Improvements of old components and introduction of new ones (especially on the client side) is a constant work in progress