ProCKSI - TechTalk

Dr. Daniel Barthel
School of Computer Science
University of Nottingham

Nottingham, 25-09-2007

Overview

- What is Parallelisation?
- Workflow
- System Architecture
- Scheduling
- Data Interchange
- Data Resources
- Agenda
- Conclusion

Introduction

Parallelisation (Wikipedia)
Simultaneous execution of some combination of multiple instances of programmed instructions and data on multiple processors in order to obtain results faster.

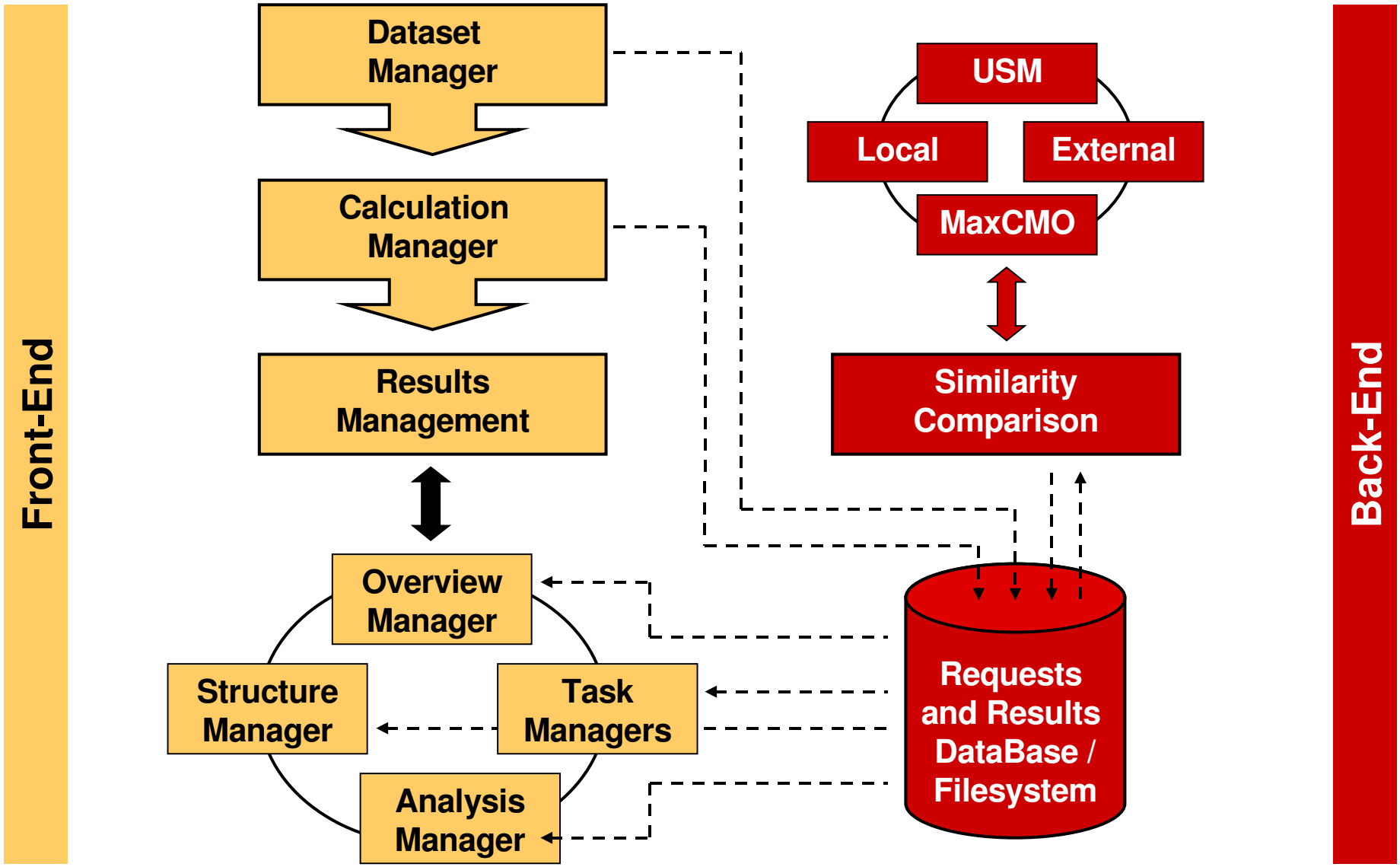
Parallel Computing (Wikipedia):
Different parts of a program run simultaneously on two or more processors *[or cores]* that are part of the same computer.

Distributed Computing (Wikipedia)
Different parts of a program run simultaneously on two or more computers that are communicating with each other over a network.

Cluster computing (Wikipedia)
Jobs or processes happening on the separate cluster computer nodes need to communicate actively during the computation.

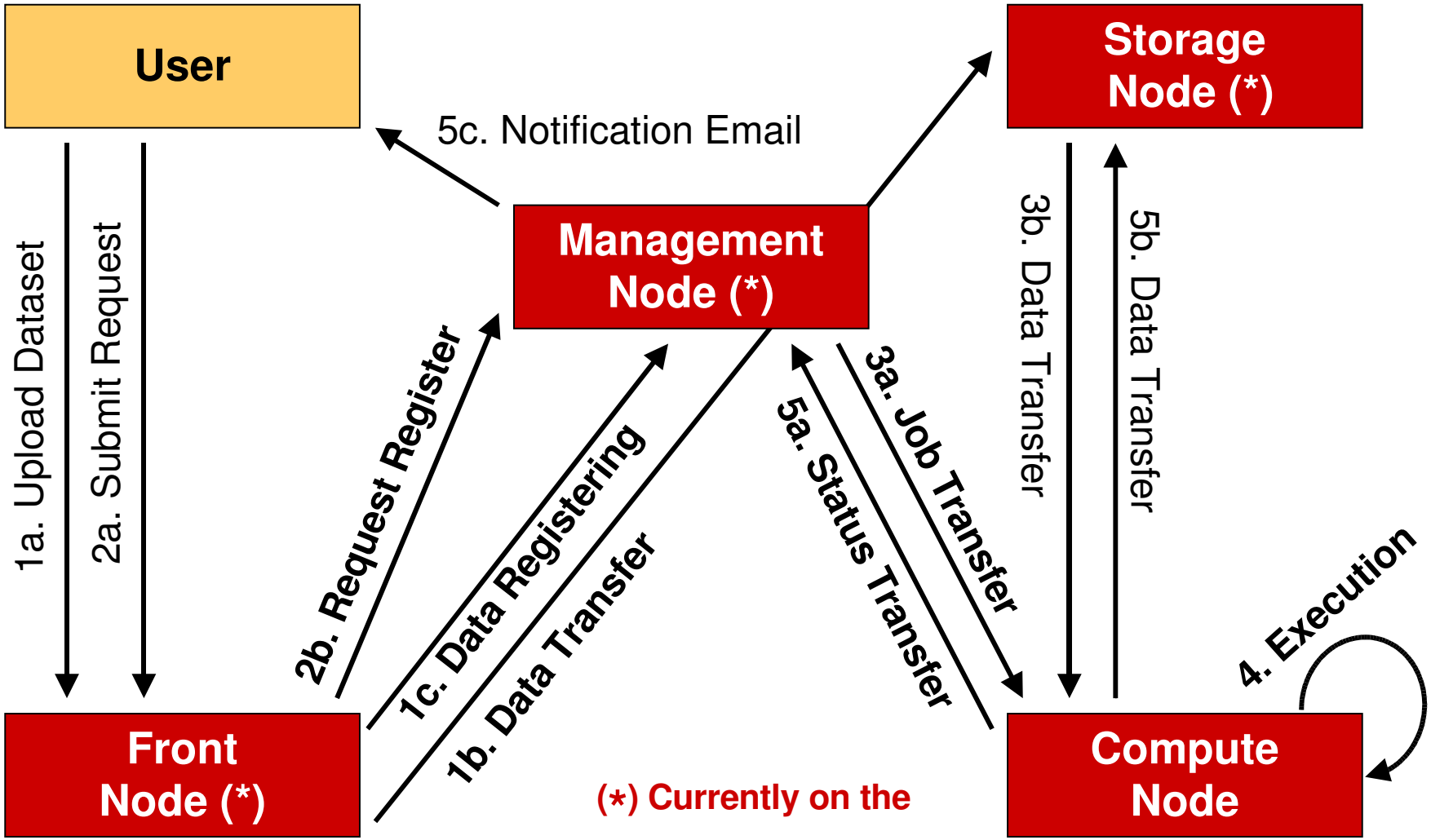
Grid computing (Wikipedia)
Many independent jobs or packets of work, which do not have to share data between the jobs during the computation process.

Workflow


Front-End
Back-End
Dataset Manager
Calculation Manager
Results Management
USM
Local
External
MaxCMO
Similarity Comparison
Overview Manager
Structure Manager
Task Managers
Analysis Manager
Requests and Results DataBase / Filesystem

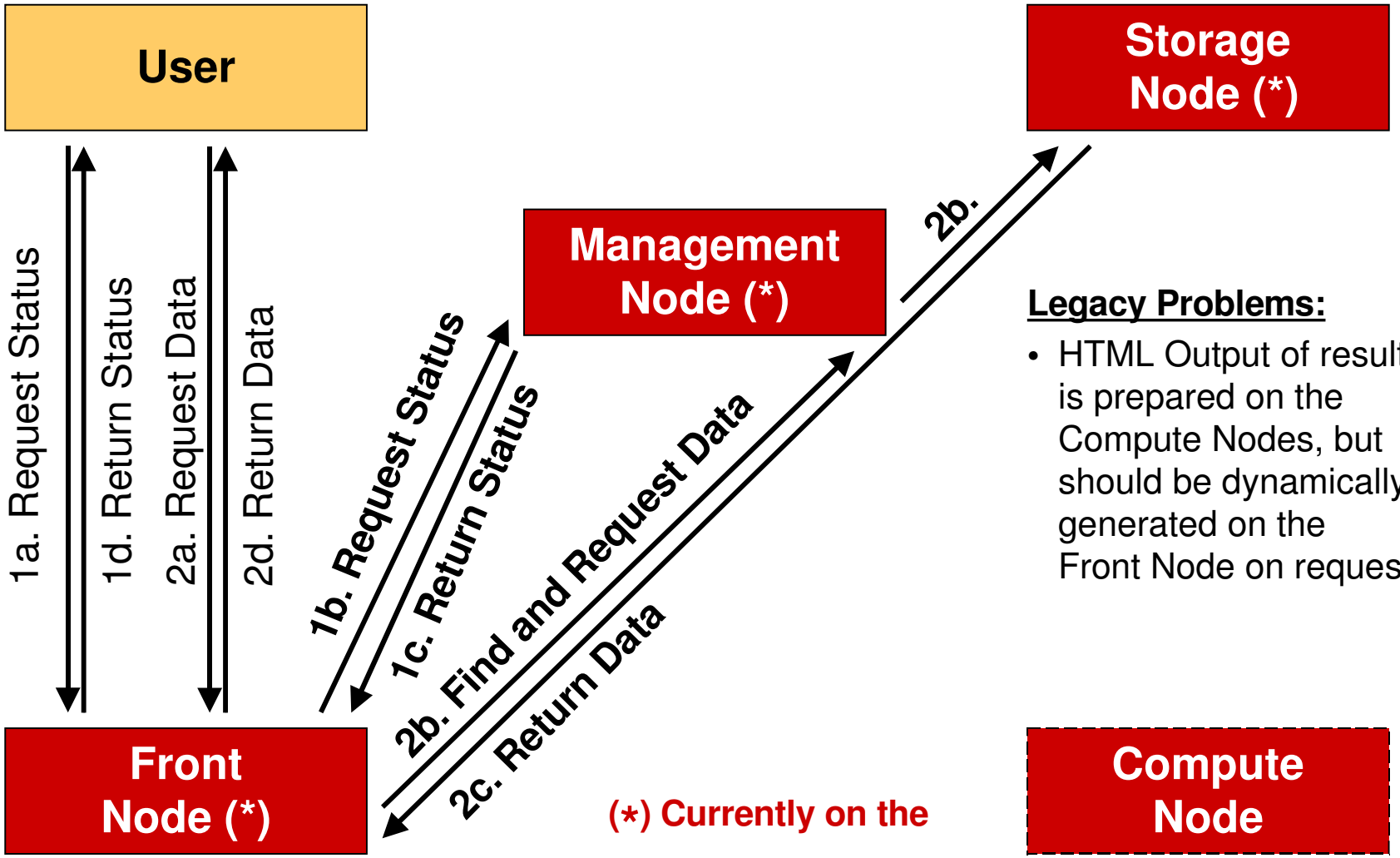
Workflow

Request Submission



Workflow

Result Retrieval

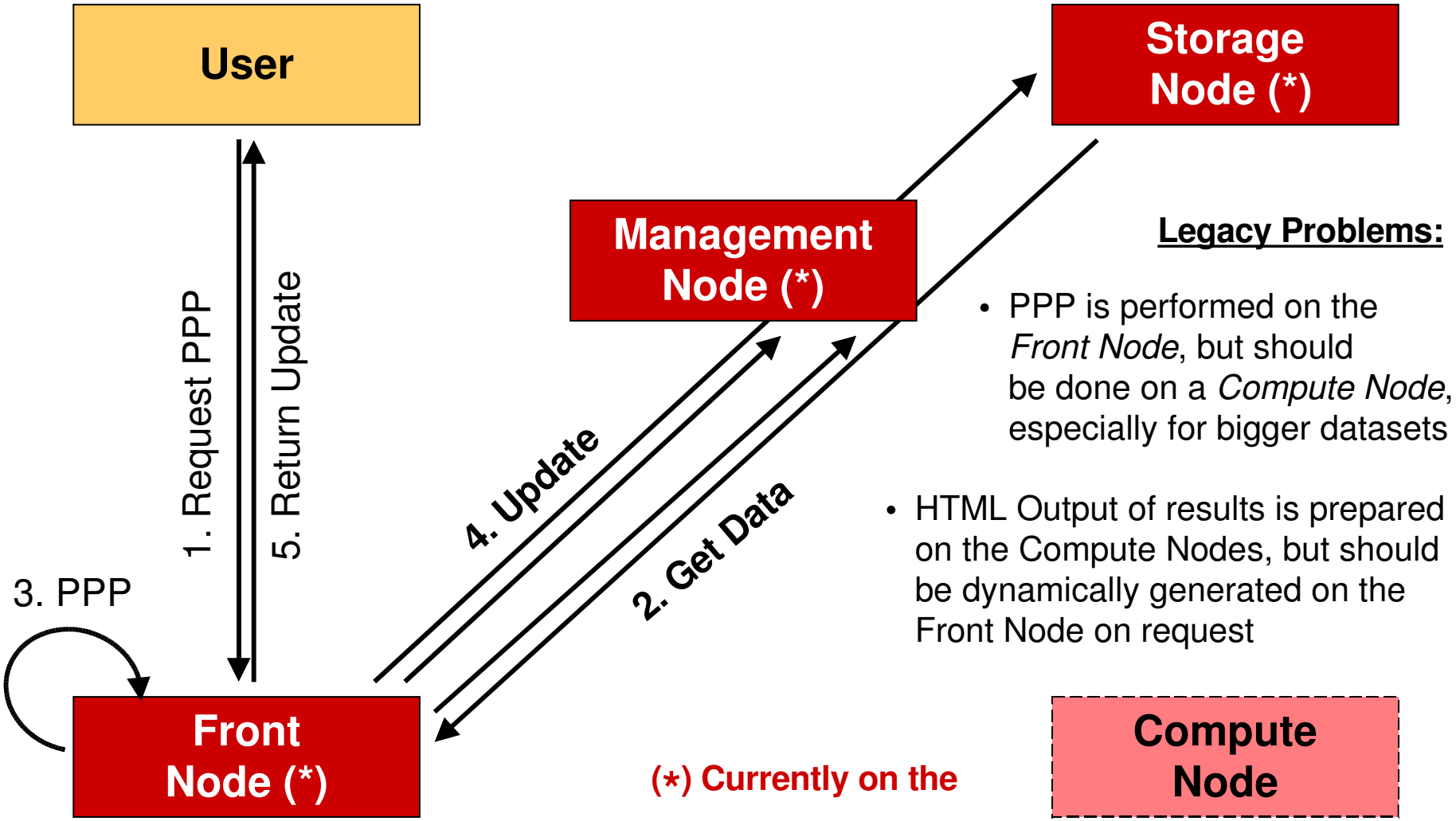


Legacy Problems:

- HTML Output of results is prepared on the Compute Nodes, but should be dynamically generated on the Front Node on request

Workflow

Pre-/Post-Processing (PPP)



Legacy Problems:

- PPP is performed on the *Front Node*, but should be done on a *Compute Node*, especially for bigger datasets
- HTML Output of results is prepared on the Compute Nodes, but should be dynamically generated on the Front Node on request

System Architecture

Hardware

Remote User

Browser

Local Cluster

Master Node
dual-processor
Xeon 3.2GHz / 4GB
1000GB SATA Raid5

2x Slave Node
dual-processor
Xeon 3.2GHz / 4GB
80GB SATA HD

2x Slave Node
dual-processor, dual-core
Xeon 3.2GHz / 4GB
80GB SATA HD

4 Compute Nodes
8 Processors
12 Cores

NOW

FUTURE

Remote Systems

University Cluster

GRID Clusters

System Architecture

Hardware (*Currently*)

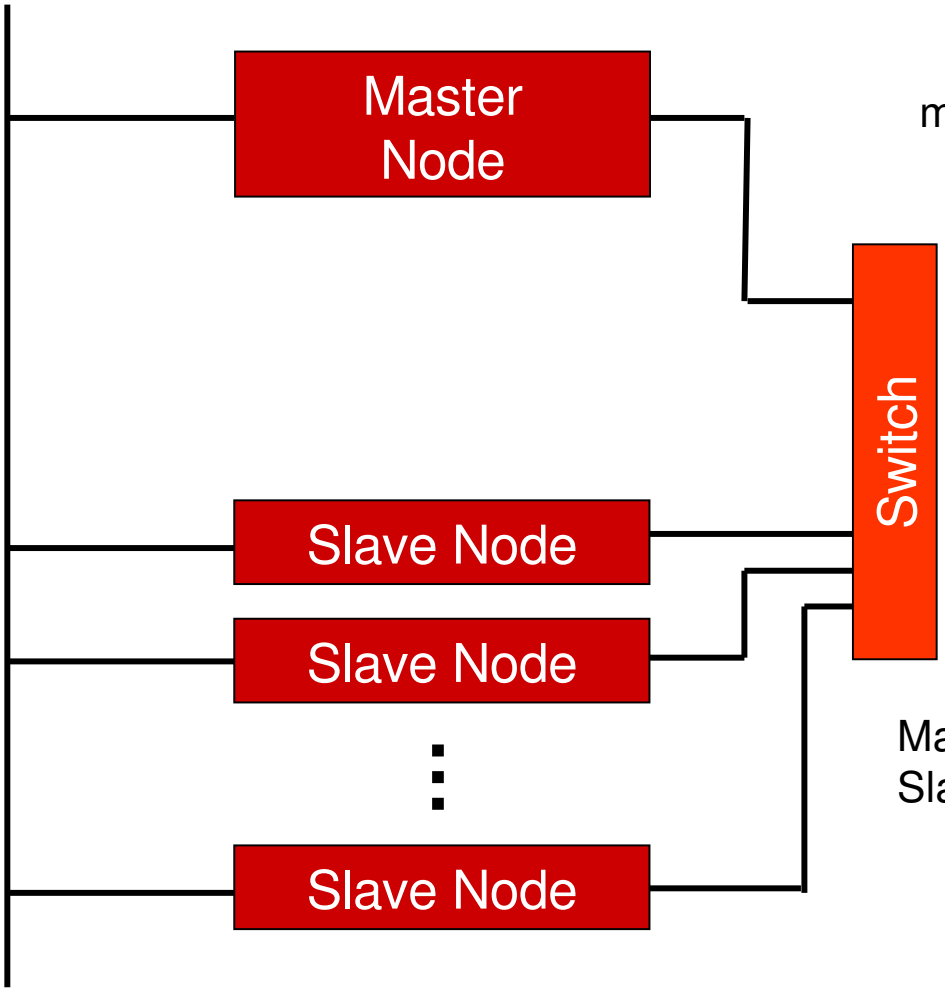
EXTERNAL

Name Resolution
procksi.cs.nott.ac.uk
procksi*X*.cs.nott.ac.uk

Network Card
eth1

Trusted Services
Master: www, ssh
Slave: ssh

IP Range
Master: 128.243.21.180
Slave: 128.243.21.18*X*



INTERNAL

Name Resolution
master*0Y*.procksi.local
slave*0Z*.procksi.local

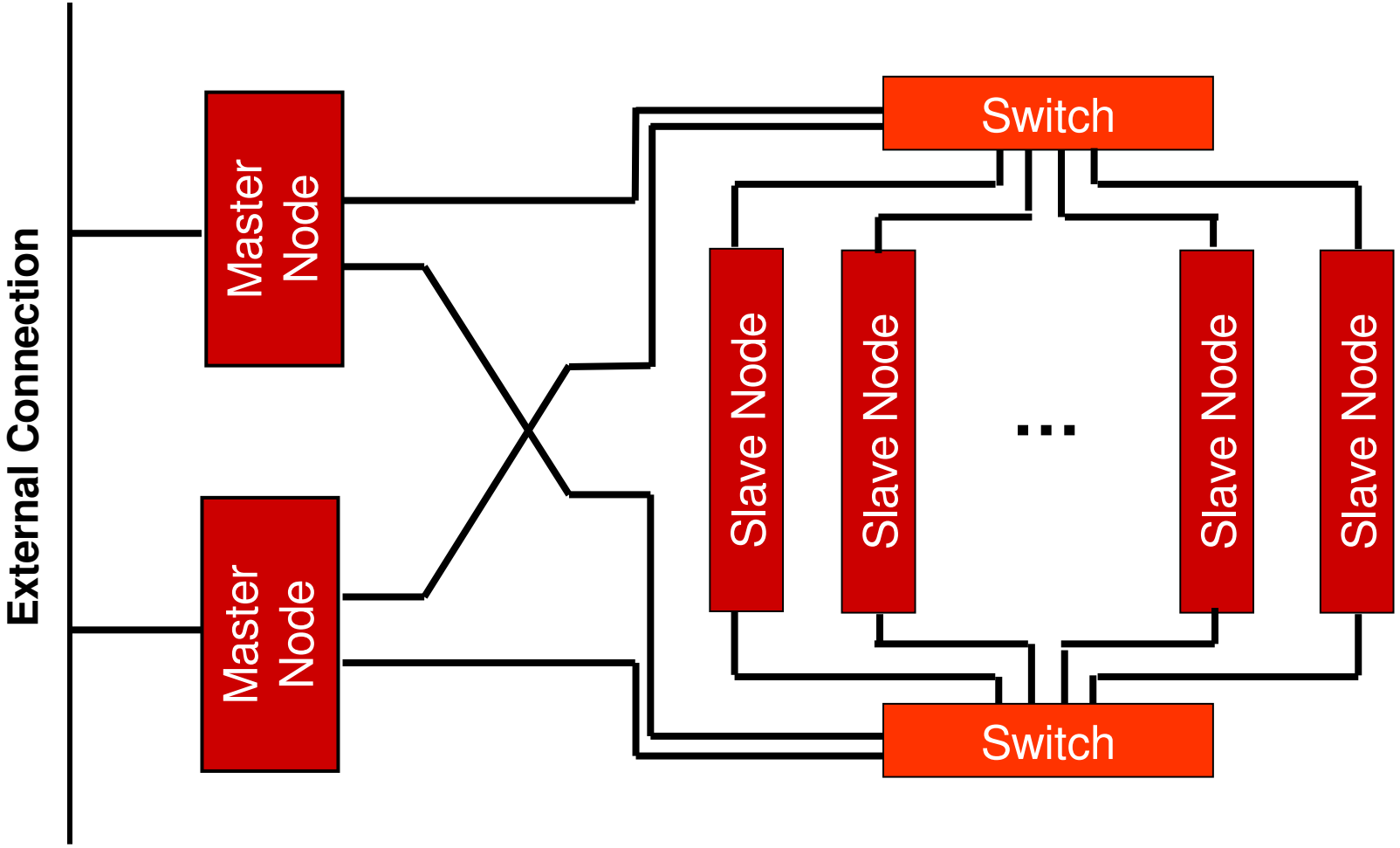
Network Card
eth0

Trusted Services
Master: all
Slave: all

IP Range
Master: 192.168.199.*0Y*
Slave: 192.268.199.*1Z*

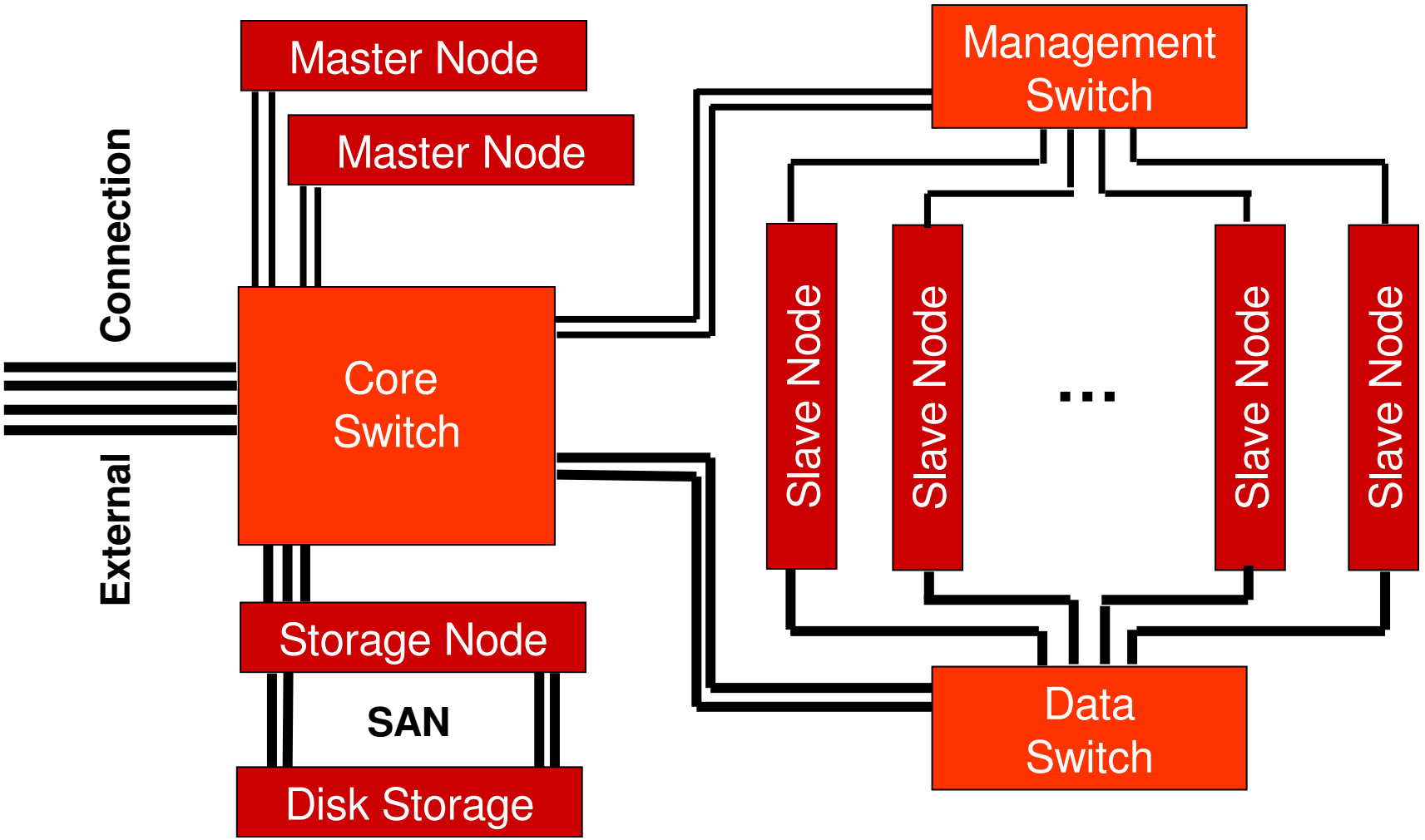
System Architecture

Hardware (*Future Plans 1*)



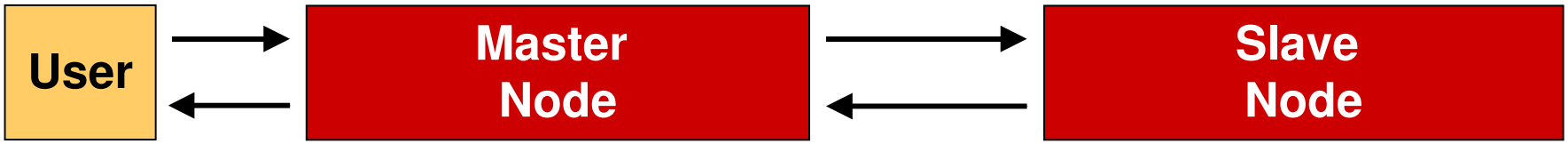
System Architecture

Hardware (*Future Plans 2*)



System Architecture

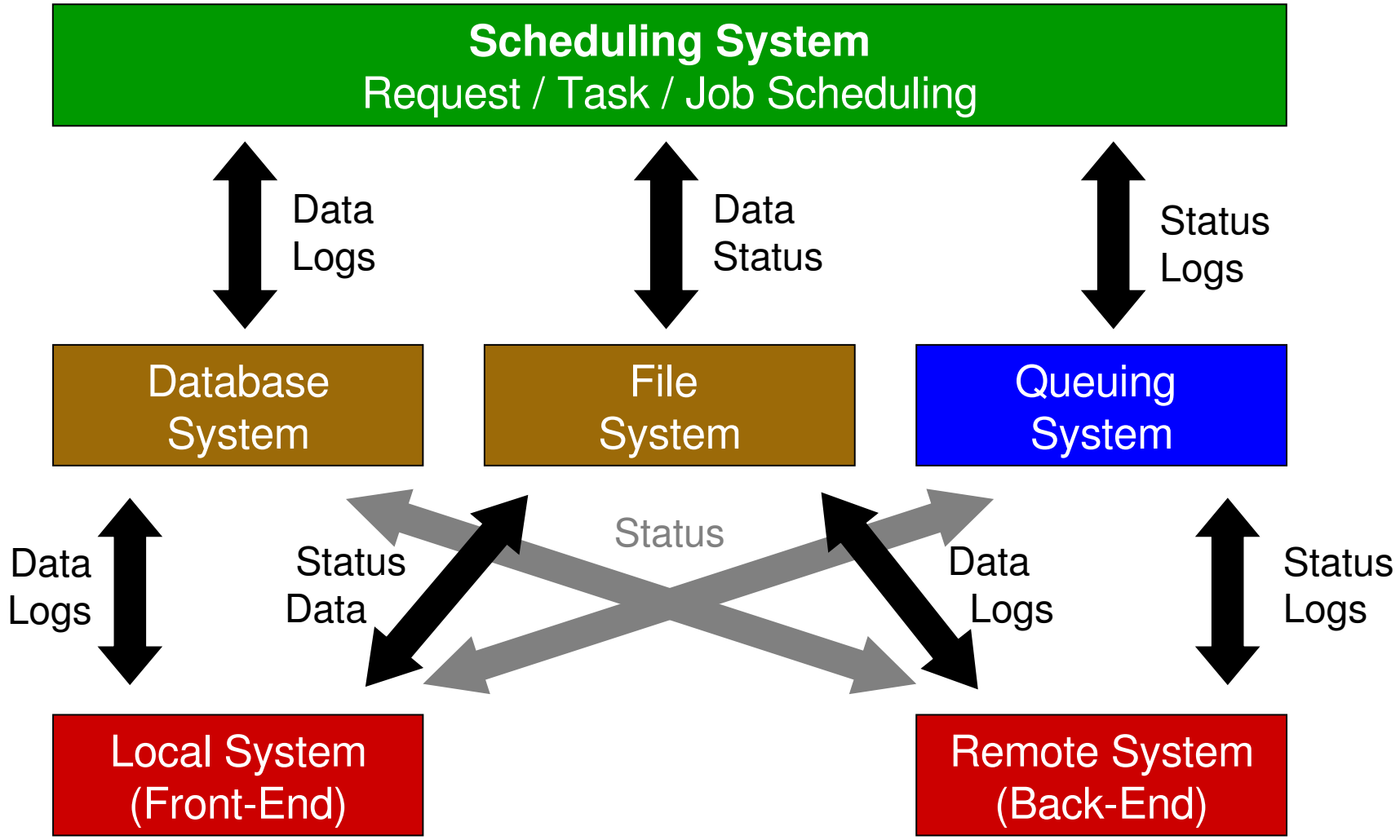
Services



- Browser (incl. Java, JavaScript)
- Email-Program
- (Shell)

Master Node	Slave Node
• Web Server (apache)	
• Database Server (mysql)	
• File System Server (nfs)	• File System Client (nfs)
• Queuing System (pbs: server, sched)	• Queing System (pbs: mom)
• Email Server (postfix)	
• Scheduling Server (cron)	
• Monitoring System (ganglia: gmetad)	• Monitoring System (gmond)
• Information System (yp: serv, passwd)	• Information System (yp: bind)
• System Access (ssh)	• System Access (ssh)
• Time Synchronisation (ntp)	• Time Synchronisation (ntp)
• ProCKSI Framework	• ProCKSI Executables

System Architecture



Definitions

Data Management

Dataset:

- Currently: Collection of PDB structures (chains) and uploaded similarity matrices
- Future plans: Previously calculated, uploaded similarity matrices should be checked against the list of structures and be reformatted to internal format

Results (1) from Similarity Comparisons:

- Currently: Entire similarity matrices of different sources
- In development:
  Additionally, all generated files: pictures, intermediate results (e.g. contact maps), …
- Future plans:
  Generate similarity matrices dynamically from single pair-wise comparison results stored in the database

Results (2) from Post-Processing (Analysis)

- Currently:
  Derivatives of similarity matrices, e.g. clustering, depending from one result file (1)
- Future plans:
  Multiple derivatives of one parent possible, e.g. from different clustering steps

Definitions

Meta-Data Management (*Currently*)

Request:
- Unique handle for the combination of a dataset, tasks, and request parameters
- Request parameters: e.g. request description, settings for notification by email
- Independent requests for the same “user”

Task:
- Something to be performed with the given dataset,
  - calculation of PDB structure pictures (1D)
  - comparison of pairs of proteins with a given similarity method (2D)
- Task parameters: e.g. parameters for each comparison method, output parameters for picture generation, ...
- Could be renamed into “*action*” as “*task*” is also used in other contexts

Job:
- Everything that lives in a queue, e.g. local queue (ProCKSI cluster), remote queue (University cluster), external queue (web service, grid service)

Definitions

User Management (*Future Plans*)

User:
- Unique entity, represented by a unique email-address
- Can manage multiple requests using the same dataset
- Authentication to gain access to user data (requests, personalised settings)

Alternative:
- Introduce weak layer of security:
  MD5-Hash representing the request instead of an integer ID (from database)
- Allow the user to access/delete data without authentication,
  but just by knowing the MD5-hash

Queuing System

Job Management

1. MN: Necessary data for each job is chosen from database, prepared (packed + compressed) and saved on SN: */home/input.tgz*

2. MN: Next job in the queue is sent to CN: *job.pbs*
3. CN: New unique job directory is created: */scratch/id_job/*
4. CN: Necessary data is fetched from SN and stored (uncompressed + unpacked) in */scratch/id_job/input/*

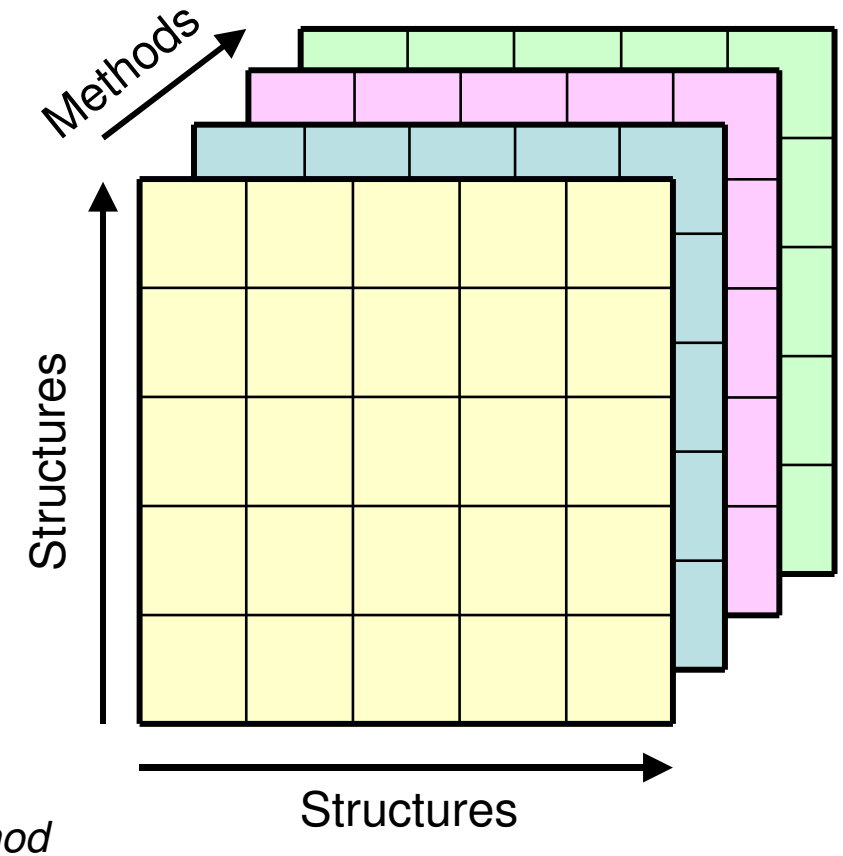
6. CN: Main execution starts and writes its results into */scratch/id_job/output/*

8. CN: All results in the output directory are prepared (packed + compressed) and saved on SN: */home/output.tgz*
9. CN: Unique job directory is deleted
10. MN: Job status is checked periodically and sent back from CN when finished

11. MN: Results are handled (uncompressed + unpacked) and registered in database

Scheduling

Problem Space:
The problem space for an all-against-all comparison of a dataset of S protein structures using M different similarity comparison methods can be represented as 3D cube.



Partitionig the Problem Space:
For a most efficient calculation of all cells in the 3D problem space, it must be subdivided into sub-cubes, which are called jobs when placed into the queue of a queuing system.

Examples:

- Comparison of *one pair of proteins* using *one method* in the task list => SxSxM jobs, each performing 1 comparison
- All-against-all comparison of the *entire dataset* with *one method* => M jobs, each performing SxS comparisons
- Comparison of *one pair of proteins* using *all methods* in the task list => SxS jobs, each performing M comparisons
- Intelligent partitioning of the 3D problem space, comparing a subset of proteins with a subset of methods

Scheduling

Current Implementation:

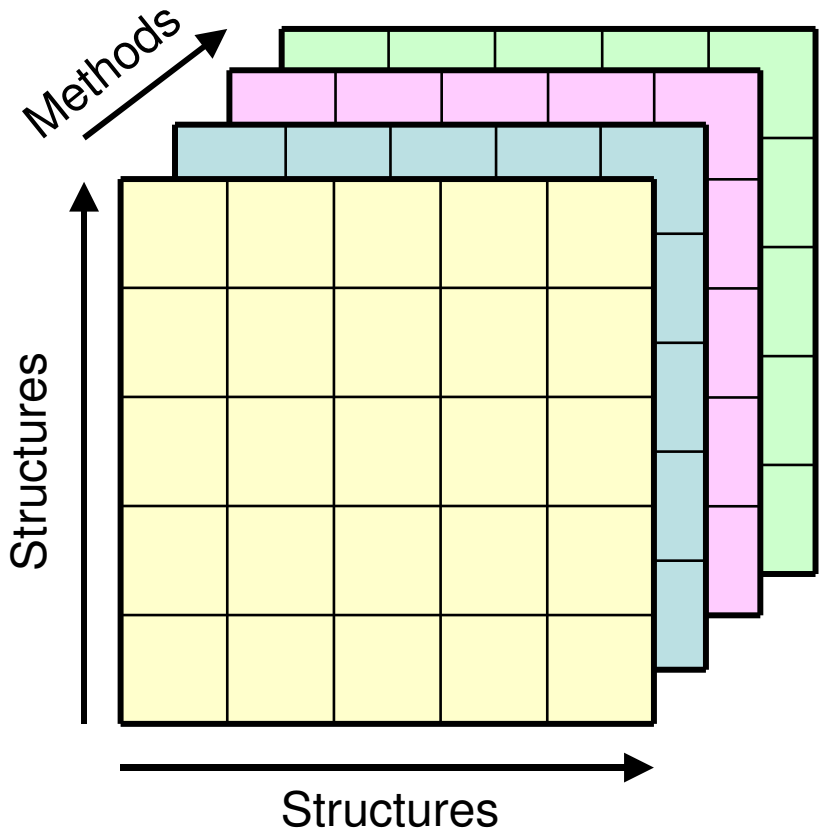
- Problem space is sliced per task, forming M jobs
- Each job in the queue performs an SxS comparisons for each method

Legacy Problems:

- Tasks and jobs are not differentiated in the database

  Task = Job

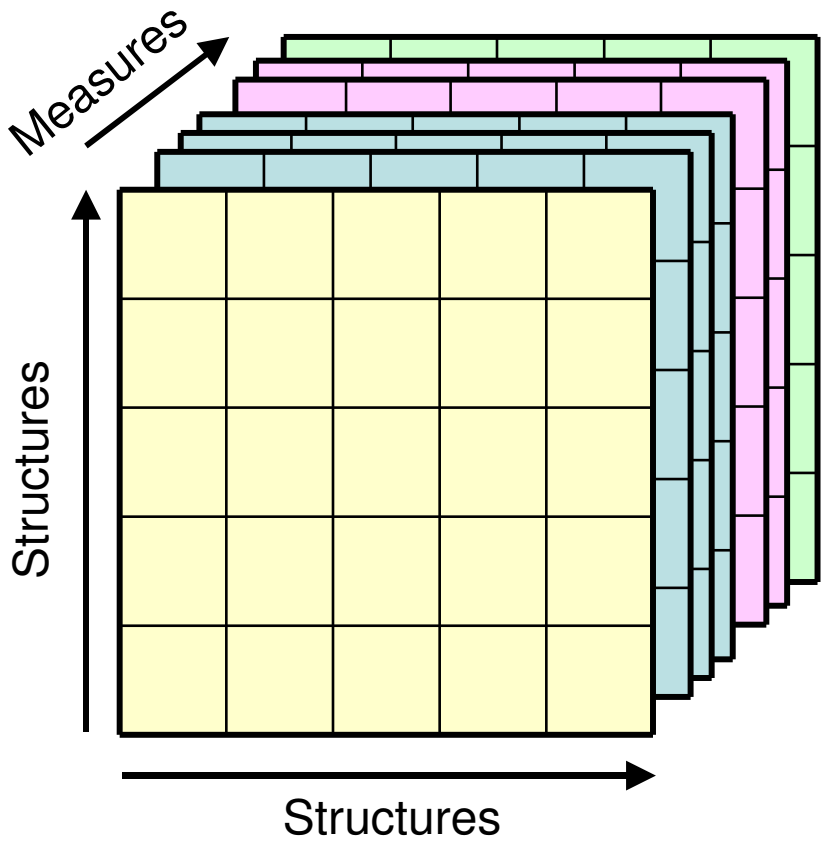
- Quadratic growth of the number of comparisons per job with linear growth of the number of structures



Scheduling

Solution Space:
Each similarity comparison *methods* can provide several similarity *measures*

For one slice in the 3D problems space using one particular method, we might get several slices in the 3D solution space providing several measures



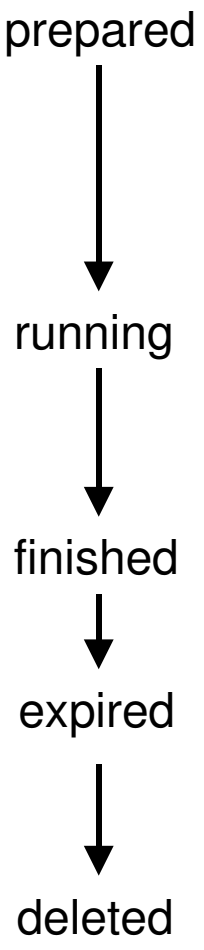
Special Cases:
The 3D problem space is reduced to a 2D problem space (1xPxM) when using methods that do not compare pairs of proteins but work on one single protein, e.g. calculating the PDB picture, or getting additional data from the iHOP web service.

Scheduling

Lifecycle of Requests

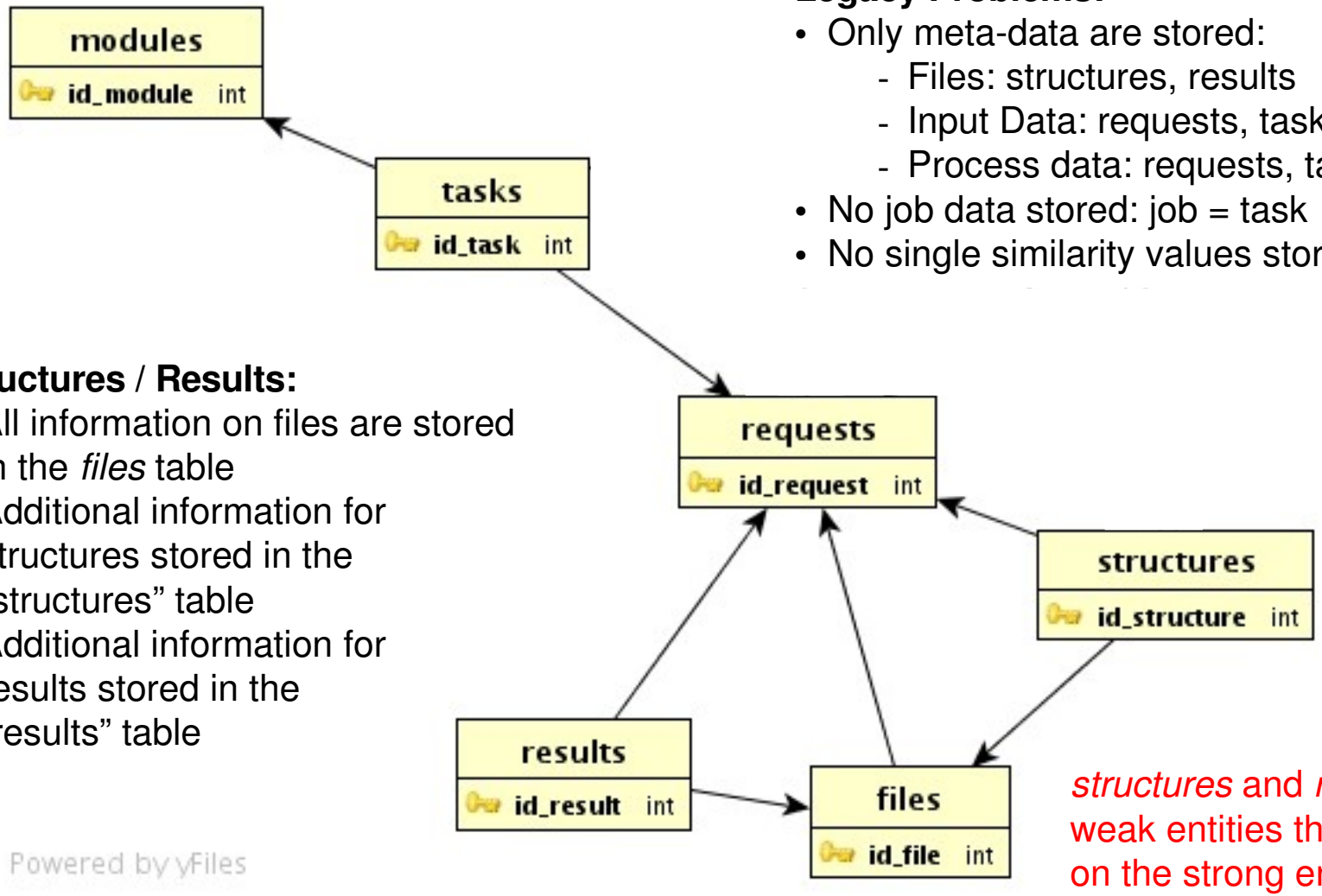
2. A new request is submitted in the browser
   - The request is registered in the database
   - Request parameters and the dataset (structures and matrices) are registered into the database

4. The scheduler (*cron*) checks the status of all *tasks* periodically and sets the status of the *request* accordingly:
   - As soon as the first task has been queued (Q) or been processed even further (R, C, F), the request is said to be *running* and the status of the request is changed in the database.
   - As soon as the last task has finished successfully (F) or with errors (E), the request is said to have *finished* and its status is changed in the database.
   - If a request has finished, and the expiration date (soft limit) has been exceeded, the request is said to have *expired* and its status is changed in the database.
   - If a request has been expired and the deletion date (hard limit) has been exceeded, the complete requeste including all tasks and data is deleted from the database and hard disk.



Database

Current Design



Legacy Problems:

- Only meta-data are stored:
  - Files: structures, results
  - Input Data: requests, tasks
  - Process data: requests, tasks
- No job data stored: job = task
- No single similarity values stored

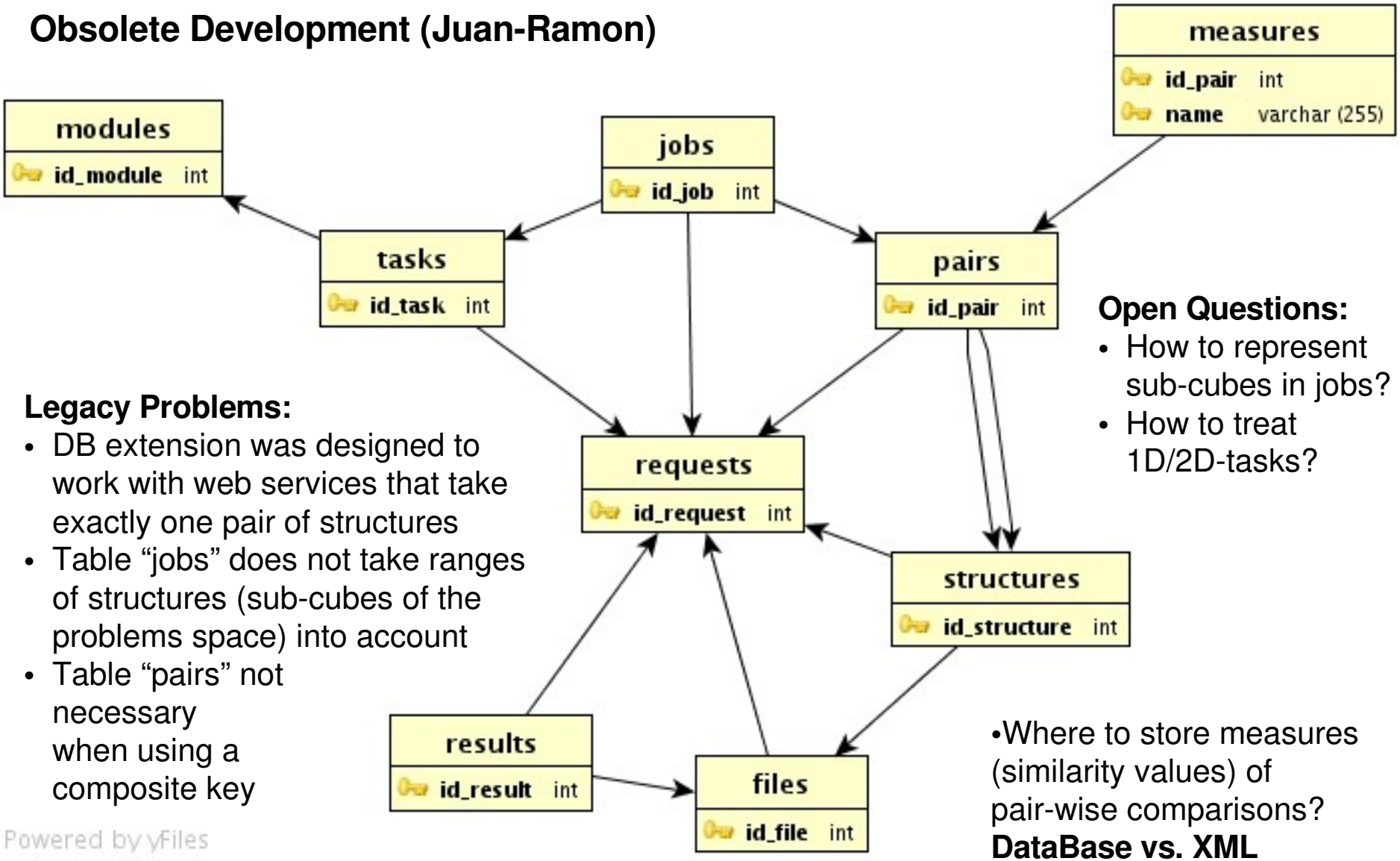
Structures / Results:

- All information on files are stored in the *files* table
- Additional information for structures stored in the "structures" table
- Additional information for results stored in the "results" table

structures and *results* are weak entities that depend on the strong entity *files.*

Database

Obsolete Development (Juan-Ramon)



Legacy Problems:

- DB extension was designed to work with web services that take exactly one pair of structures
- Table “jobs” does not take ranges of structures (sub-cubes of the problems space) into account
- Table “pairs” not necessary when using a composite key

Open Questions:

- How to represent sub-cubes in jobs?
- How to treat 1D/2D-tasks?

•Where to store measures (similarity values) of pair-wise comparisons?
DataBase vs. XML

Scheduling

Lifecycle of Tasks

2. A new request is submitted in the browser
   - All *tasks* that the user has selected to be performed are prepared and registered into the database.

4. The scheduler (*cron*) checks the status of all *tasks* periodically and acts accordingly:
   - Prepared tasks are submitted into the queuing system.
   - When a *Task* starts, it changes its own status in the database.
   - When a *Task* reaches its end, it changes its own status in the database.
   - When a *Task* has been completed, its results are post-processed (e.g. registered in the database), and the status of the task is changed in the database, and an expiration and deletion date is set.
   - In case that there have occured any serious problems, the status of the task is changed in the database accordingly.
   - Tasks are deleted automatically when the entire request is deleted.

Status

prepared

queued

running

completed

finished

(error)

(deleted)

Scheduling

Dependencies

- Some parent tasks must have finished successfully before a dependent task can be started:
  e.g. *Contacts* must have been calculated before *USM* and *MaxCMO* similarities can be calculated
- If a parent task fails, a dependent task must be considered as failed, too

Legacy Problems:

- Currently:
  Each task handler sets its own status, writing directly into the database.
  Future plans:
  The *scheduler* should check the status of a task/job directly in the PBS queuing system, detect if it has started/finished, and set the status in the database accordingly.
- Currently:
  There are separate schedulers for requests, tasks, and jobs.
  Future plans:
  There should be one scheduling *daemon* that can be started, stopped, modified.

Data Resources

Dataset

Protein structures can be obtained using the following ways:

User: Upload of arbitrary protein structures from his/her harddisk
+ no maintenance
- user needs to upload entire dataset

Online repository: PDB repository (http://www.pdb.org/)
+ repository is maintained externally and always up-to-date
- each protein in the dataset needs to be downloaded on demand over the internet
- the same protein structure might be downloaded multiple times for different datasets

Local repository: Local copy of the PDB repository
+ "caching" of structures: quick access
+ low data volume to be transferred
- synchronisation with official repository necessary (nightly/weekly updates)
- "caching" of structures: structures that are not available locally, must be download from online repository

Data Resources

Similarity Comparisons

Similarity comparisons can be performed using the following resources:

Local: ProCKSI Cluster
+ exclusive use of resources: highly available
+ direct access to database
+ usage: any software
- high maintenance
- high acquisition costs

Remote: University Cluster or the Grid
+ low maintenance
+ no acquisition costs
- usage: software that can be installed & runs in this environment
- shared use of resources: queuing times, restricted calculation times

External: Web services
+ low maintenance
+ no acquisition costs
+/- shared use of resources
- usage: only for provided services

Data Resources

Resources

Resources for similarity comparison methods, clustering, visualisation and further information

Local Methods:
+ implemented: USM, MaxCMO, DaliLite, TM-align, FAST, CE, Vorolign
+ implemented: qclust, HyperTree
- not available: LGA, FlexProt, SSM

Web Services:
+ available: DaliLite, iHOP, others
+/- in development: CATH-SSAP, Vorolign
- not available: SSM

Others: Barcelona Supercomputing Center, European Bioinformatics Institute

Web Pages
+/- linked: iHOP, RCSB, CATH, SCOP

Data Interchange

Job Submission

Job submission specifies

- Job name and description
- Resources needed (CPU, OS, …)
- File staging (before and after execution)
- Main command execution

can be described

- **for local execution**:
  job submission script
  (PBS)
- **for GRID execution**:
  job submission description language
  (JDSL)

Job Execution (*Proposal*)

```
<job id="ID" description="DESCRIPTION">
  <log filename="FILENAME" />

  <structure id="ID" label="LABEL" filename="FILENAME" />

  <method type="1D|2D" name="NAME">
    <param name="NAME" value="VALUE" />
    <measure name="NAME" />
    <return name="SIMILARITY|ALIGNMENT|MATRIX|FILE" />
  </method>
</job>
```

Data Interchange

Partial Results (*Proposal*)

```
<job id="ID" node="NODE" start="TIME" end="TIME">
  <status flag="FLAG" success="NUMBER" failed="NUMBER" />
  <log filename="FILENAME" />

  <message type="TYPE">MESSAGE</message>

  <structure id="ID" label="LABEL" filename="FILENAME" />

  <comparison id_structure1="ID1" id_structure2="ID2">
    <message type="TYPE">MESSAGE</message>

    <method name="NAME" version="VERSION">
      <similarity measure="MEASURE">VALUE</similarity>
      <alignment label="LABEL">ALIGNMENT</alignment>
      <matrix type="TYPE" label="LABEL">MATRIX</matrix>
      <file type="TYPE" name="NAME" label="LABEL" />
    </method>
  </comparison>
</job>
```

Agenda

Usability / Frontend

- Bigger datasets:
  - Pagination of results / output
  - Integrative view of results from different calculation
  - Download of results; size!
  - Send PPP tasks into queue
- Upload:
  - Prevent multiple uploads while files are being uploaded/split
  - Accept only ticked chains
  - Caching of input and results: PDB
  - Upload external similarity matrices in post- not pre-processing step
- Analysis:
  - Allow more trees to be displayed in one applet
  - Different standardisation/ clustering tasks must not overwrite existing results
- User management
  - One user with different requests
  - Restricted access to requests only by password
- Webdesign
  - Better navigation and layout
  - Corporate Design (CD) with standardised CSS
- Restrict number of emails to be sent (*in development*)

Agenda

Parallelisation (► Collaboration with Azhar)
•Standardised format for job submission, execution, results
•Scheduling of tasks/jobs
•Show status (requests, tasks, jobs) of entire cluster (*in development*)
•Integration of GRID middleware
•Resubmit failed jobs, tasks, requests on demand

Database
•Complete re-design / re-engineering
taking 3D problem/solution space, jobs, measures into account
•Referential integrity checks
•Performance!

Data Handling / Error Handling / Logging / Debugging
•Policy for all error, log, debug output: output chain
•Policy for errors, warnings, notices, … in different environments:
front-end (browser), back-end (files)
•Hierarchical representation of results in database

Agenda

Information / Representation

- Documantation
  - Description of input / output parameters
  - Tooltips (*in development*)
- Visualisation
  - Alignment from any method (standardisation!)
  - Contact maps with bows (*in development*)

Extension

- General approach to integrate web services, e.g. CATH-SSAP
- Other local methods, e.g. Vorolign, LGA, …
- Provide API to access ProCKSI as a web service

Research

- Reengineer USM according to latest papers + jbig compression
- Integrate Total Consensus approach

Misc

- Better installation / deployment tool: master, slave nodes
- License agreement

Conclusions

ProCKSI needs dedicated collaborators to move on!

Collaboration in terms of:

- Sharing code
  - SVN repository
- Sharing data
  - Database with pre-processed, post-processed data
- Sharing resources
  - Separated development systems
  - Shared test-, live-systems
- Sharing knowledge
  - Wiki
  - Ticketing System
  - Mailing list

Action Points

Priorities of the points given in the Agenda:

- Research Collaborations
  - USM (libraries, jbig): *Paolo Ferragina, Gabriel Valiente*
  - Scheduling (distribution, scalability): *Gianluigi Follino, Azar Shah*
  - Clustering (TE vs. TC): *Gianluigi Follino, Daniel Barthel*

- Resolve Legacy Problems
  - Generate HTML output from database on the front-node
  - Standardise/cluster results using the queuing system and compute nodes
  - Validate user similarity matrices against list of structures and convert it internal format e.g. by storing it in database
  - Account for bigger datasets and time needed for actions (e.g. pagination, meta-refresh)
  - Restrict access to user data (e.g. MD5 request ID, .htaccess)

Action Points

- Interoperability
  - Redesign database for scientific and meta-data
  - Design data exchange format for scientific and meta-data to go along with database
- Extension
  - Introduce caching on master / slave nodes: e.g. PDB
  - Improve cluster design: second switch, second master node
  - Use condor / *globus toolkit* for integration of external cluster / grid
  - Different entry points / front-ends for heavy vs. normal users:
    - FTP upload & download
    - XML “workflow” description to circumvent web-interface
- Documentation
  - Help pages and tooltips