Mitglied der Helmholtz-Gemeinschaft

UNICORE

JÜLICH FORSCHUNGSZENTRUM

Perspectives for RESTful services in UNICORE

Bernd Schuller[1], Jedrzej Rybicki[1], Krzydsztof Benedyczak[2]

[1] Federated Systems and Data division, JSC Forschungszentrum Jülich GmbH
[2] ICM, University of Warsaw
24 June 2014, UNICORE Summit 2014, Leipzig

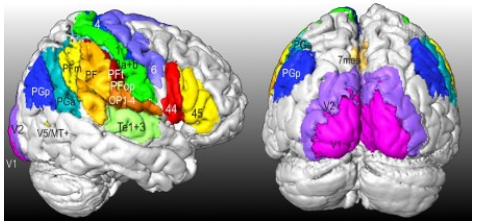
JÜLICH
FORSCHUNGSZENTRUM

Outline

- Motivation and Human Brain use case
- WS(RF) vs RESTful
- UNICORE Services Environment (USE)
- Architecture, Security and all that
- Current state of development and some first results
- Outlook

Human Brain Project

- FET Flagship
- ~10 years, ~1 Billion € (50% EC funding)
- Coordinated by EPFL (Lausanne)
- Huge Consortium
  - Consortium starts with > 80 partners from 23 countries
  - New partners through *Competitive Calls* (~16% of budget)
  - ~200 partners by Y5
- www.humanbrainproject.eu

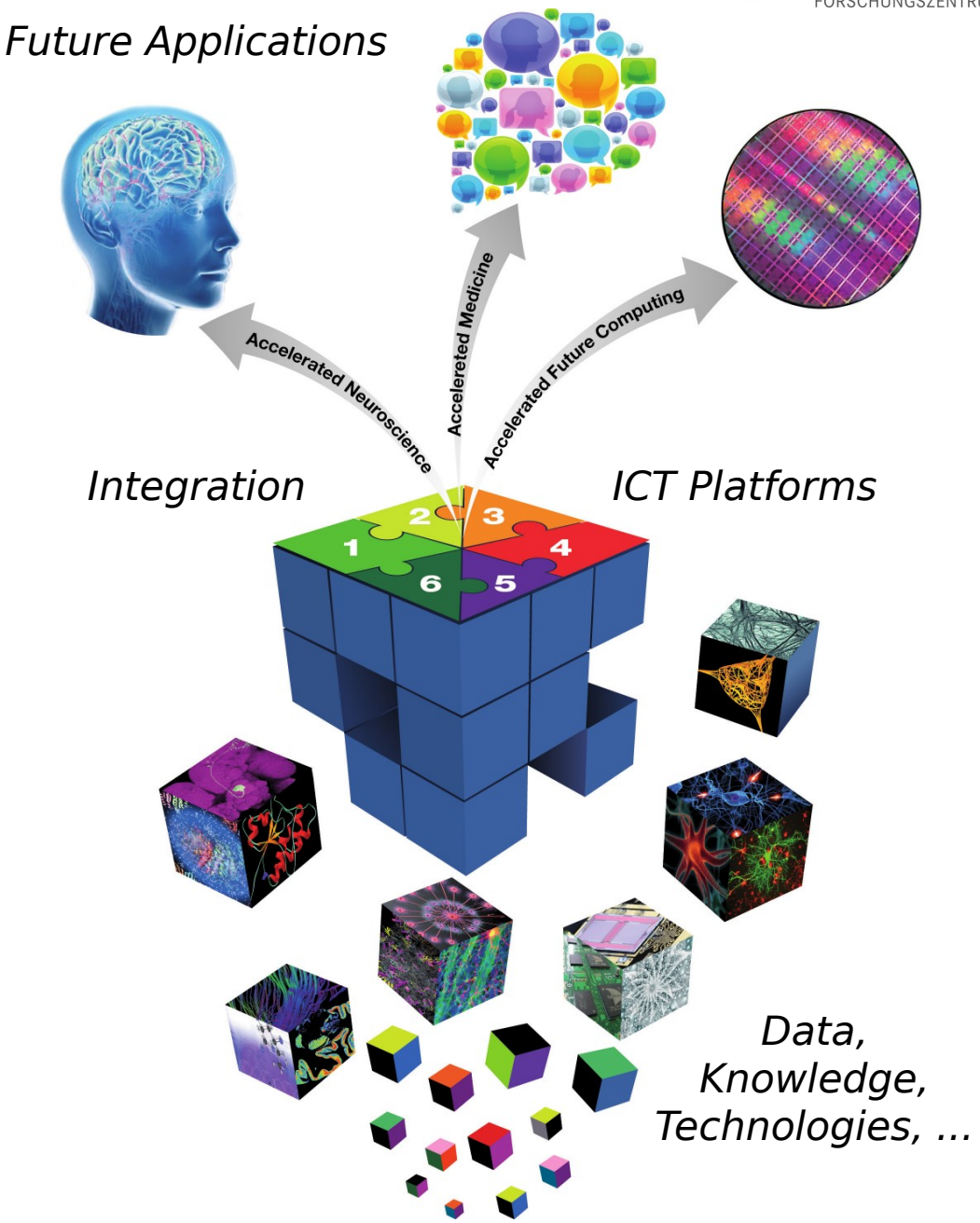


HBP Goal

To build an integrated ICT infrastructure enabling a

Global collaborative effort towards understanding the human brain, and ultimately

To emulate its computational capabilities



HBP High performance computing platform

JÜLICH FORSCHUNGSZENTRUM

Technology evaluation and deployment of HPC systems

Main production system at Jülich (Exascale capability around 2021/22) plus facilities at CSCS, BSC, CINECA

Applications requirements analysis, subcontracting for R&D and prototypes

Mathematical methods, programming models and tools

Parallel and distributed programming models, work flows, middleware for resource management, performance analysis & prediction, numerical algorithms for neuroscience

Interactive visualization, analysis and control

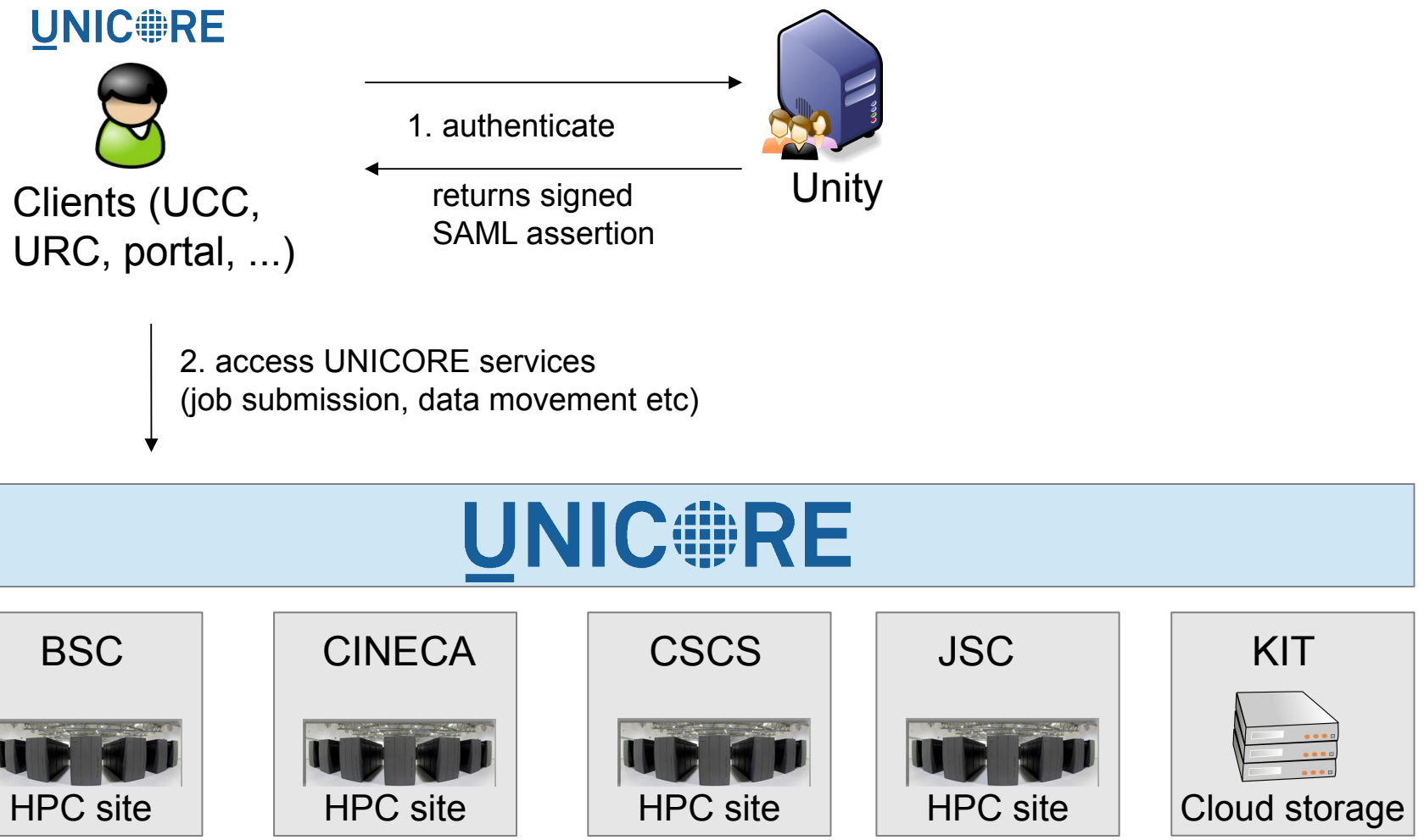
In-situ visualization and interactive steering and analysis of simulations

Exascale data management

Scalable querying of datasets, data analytics, data provenance and preservation

Brain-inspired supercomputing

HPC platform – UNICORE infrastructure



Human Brain Project – Unified Portal

HBP Unified Portal
(HTML+JavaScript),
Python applications, etc

authenticate

returns OIDC token

OIDC server

Access to HPC platform services

REST APIs

UNICORE

BSC

HPC site

CINECA

HPC site

CSCS

HPC site

JSC

HPC site

KIT

Cloud storage

Human Brain Project – initial requirements

- Authentication with OpenID Connect (OIDC) → OAuth2
- Service/Site registry
- Job submission and management
- Later
  - Full support for data staging and data movement
  - Quota information (CPU/Storage)
  - Deeper integration with Unified Portal (especially applications management)

WS(RF)

- Style
  - XML messages
  - Service-specific interfaces, custom methods
  - RPC style method invocation with params
  - WSRF: resources
- Security
  - HTTPS
  - WS-Security
  - SAML assertions for delegation and authentication (UNICORE specific)

WS(RF) – pros and cons

- Pros
  - Strongly typed
  - Messages can be validated
  - SOAP: headers/envelope mechanism

- Cons
  - CPU intensive (XML processing, XML signatures)
  - Complex interface (WSDL!)
  - Only Java and C# can be realistically used on the client side

RESTful

- Style
  - „HTTP-like“ interface (GET, PUT, POST, DELETE)
  - Resource oriented
  - JSON messages
- Security
  - HTTPS
  - Authentication?
  - Delegation?

RESTful – pros and cons

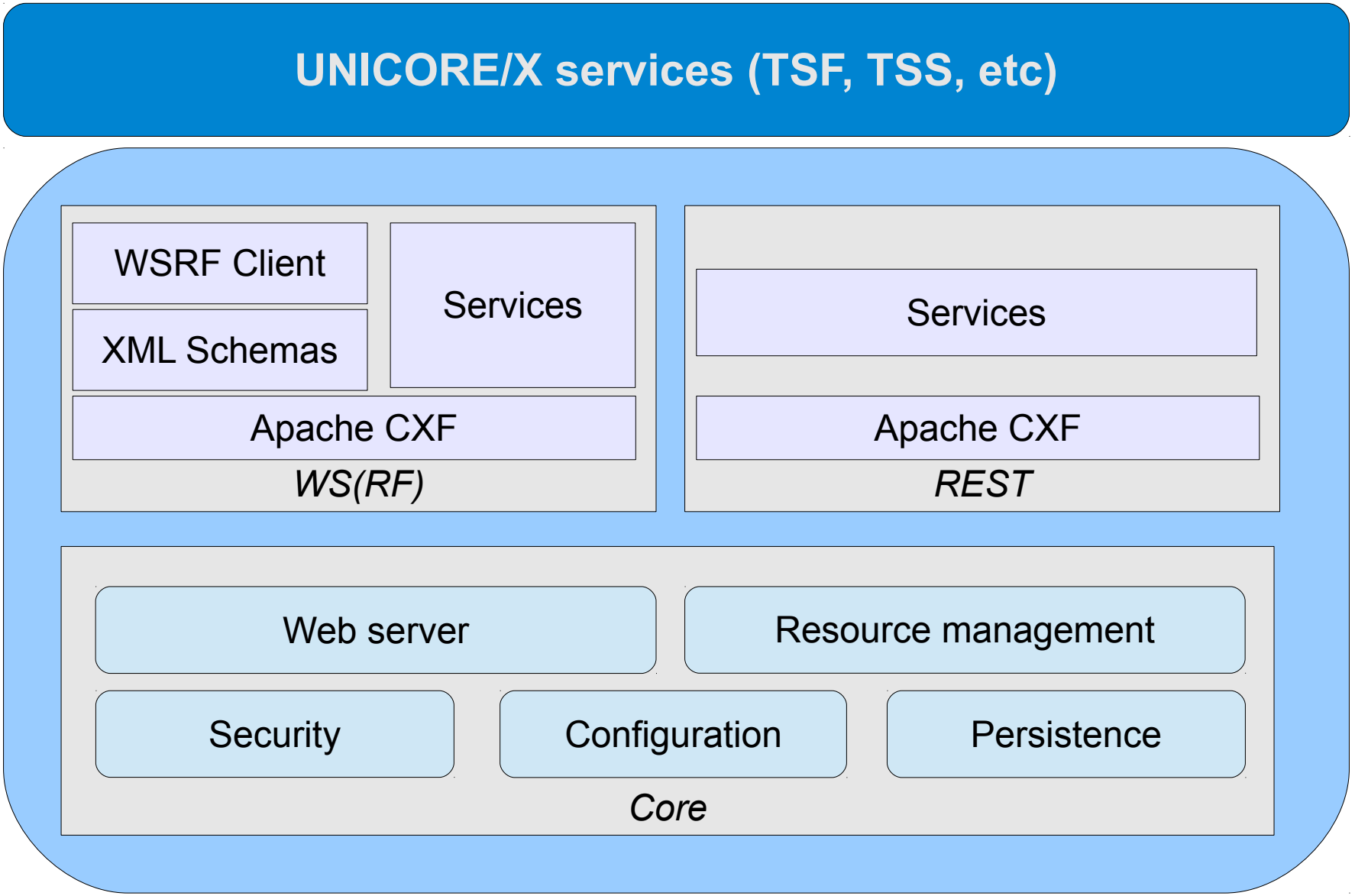
- Pros
  - Weakly coupled
  - HTTP benefits (error codes, caching, …)
  - Multiple representations of a resource can be served
    - *JSON for services*
    - *HTML for browsers*
  - Simple clients (even *curl* or *wget*)

- Cons
  - Weakly typed → no easy validation
  - Trust delegation is a problem!

Design goals

- Keep WSRF services!
  - Backwards compatibility
  - Delegation problem is solved nicely (SAML trust delegation chain)
  - Not all services may be suitable for RESTful style
- Want consistency between WSRF and RESTful versions of the same services
  - Access to same jobs, data, etc
- Consistent security layer (user DNs, attributes mapping, access control)
- Gateway support

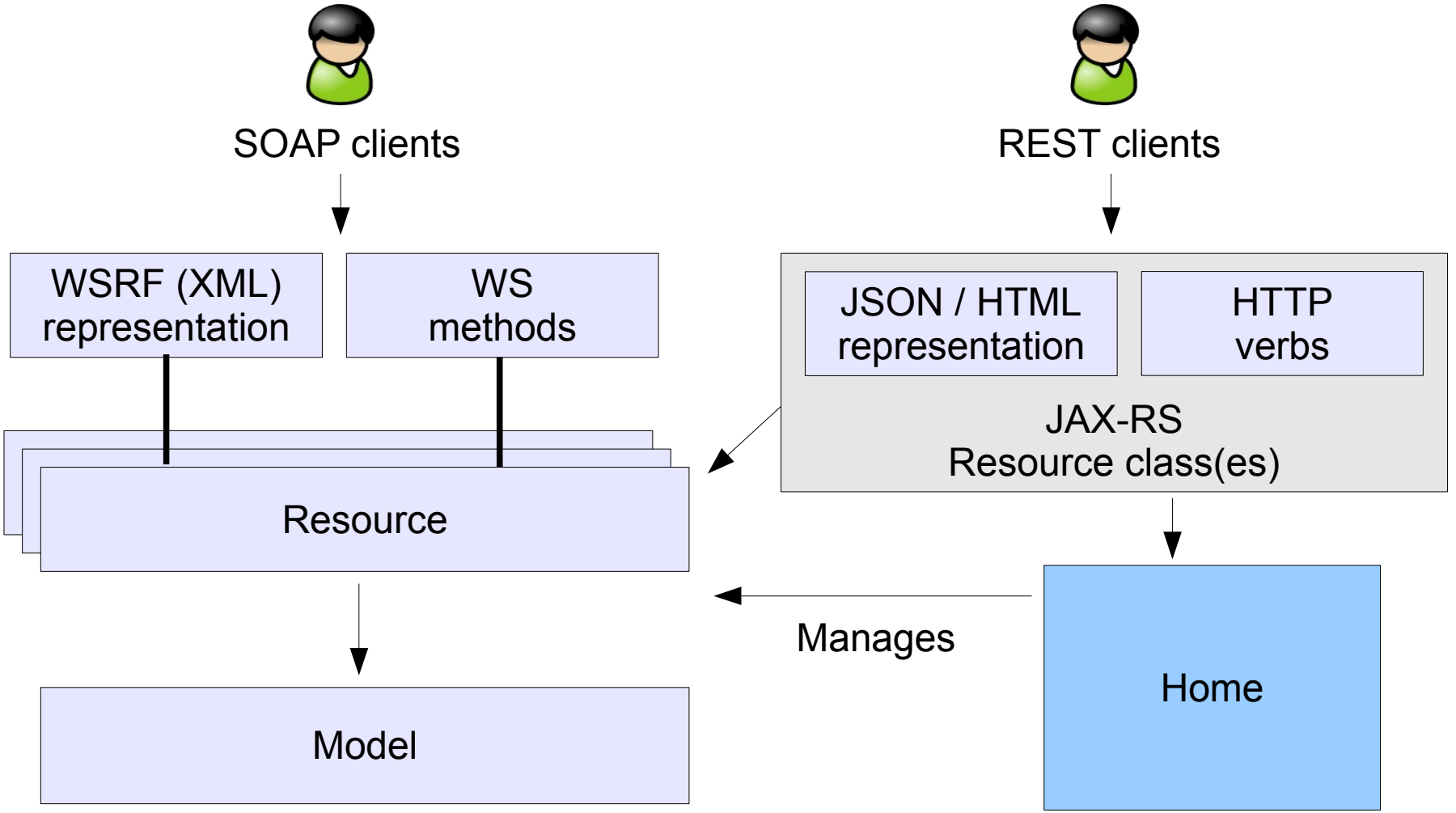
UNICORE Services Environment



USE resources

- Resource = thing accessed via a (stateful) service or via RESTful interfaces
- Example: a job
  - Real job lives on the cluster
  - UNICORE keeps additional information
  - Stored in a model and persisted to disk
  - Clients can access the information („representation“ of a resource)
  - Clients can modify the resource via the service (e.g. abort the job)
- Pattern: Model – View – Controller
  - Model = model + live information (e.g. job status taken from queue manager)
  - View = representations (XML, JSON, HTML, …)
  - Controller = methods for modifying the resource

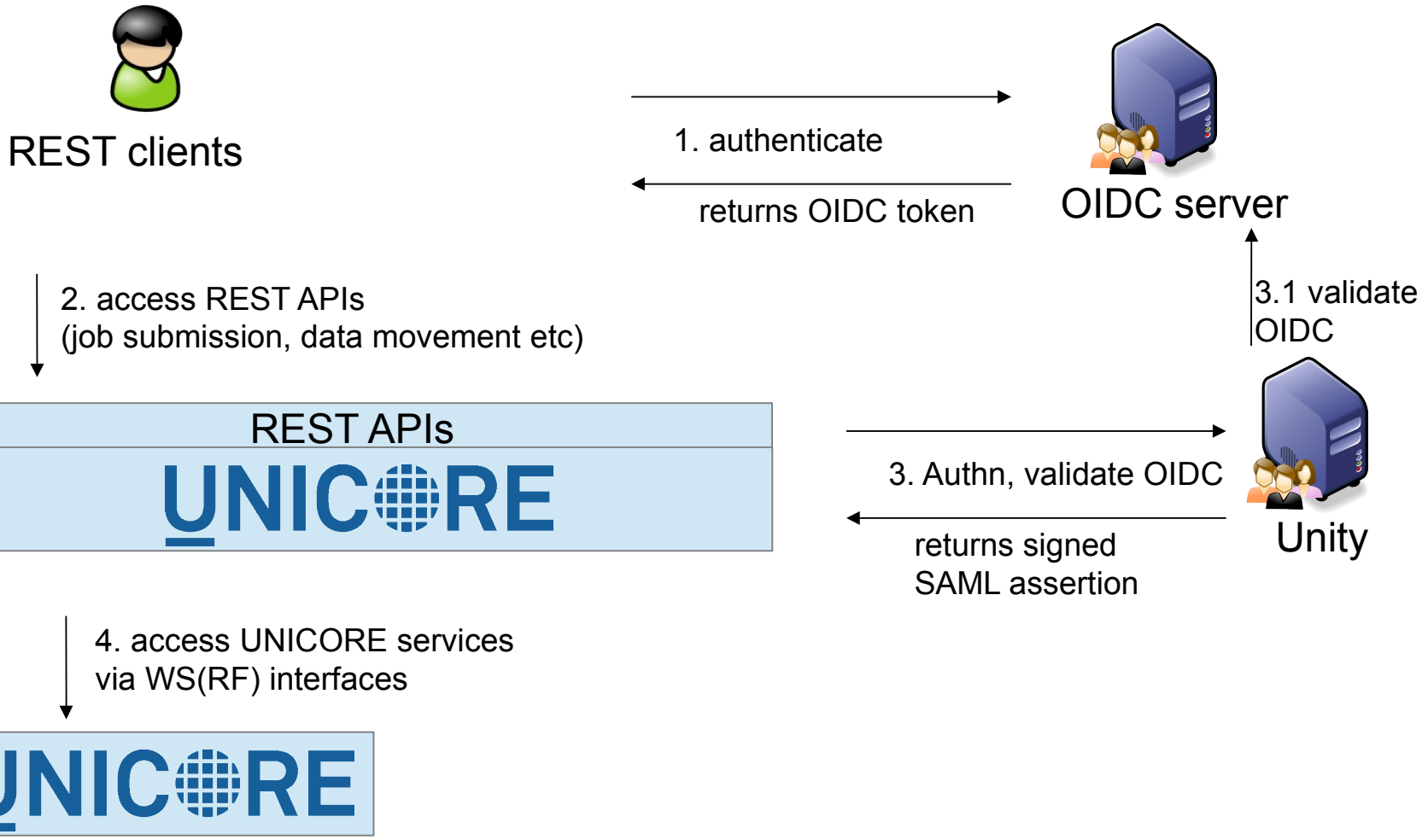
USE resources



REST invocation in USE

- Based on JAX-RS provided by Apache CXF
- UNICORE specific extensions
  - Authentication handler
    - Map user to a DN
    - Create full UNICORE security tokens
  - Resource handler
    - Map incoming call to existing resource home and unique ID
    - Perform access control
    - Inject home and resource, lock resource if required
  - Cleanup handler
    - Unlock if required
    - Clear security context

Multi-step calls (workflows, data staging, ...)



Status: basic problems are solved!

- Authentication
  - HTTP basic auth with mapping to DN
- USE integration
  - Get user attributes and create security tokens
  - Inject resource model and home
  - Locking and cleanup
- Access control

Next steps

- Authentication: what is the best way to support OIDC?
- Implement full integration with Unity
- Start work on UNICORE services
- Security session support
- Framework improvements, e.g.
  - Error handling and proper use of HTTP status codes
  - Implementation clean-up