

UNICORE Architecture and Server Components

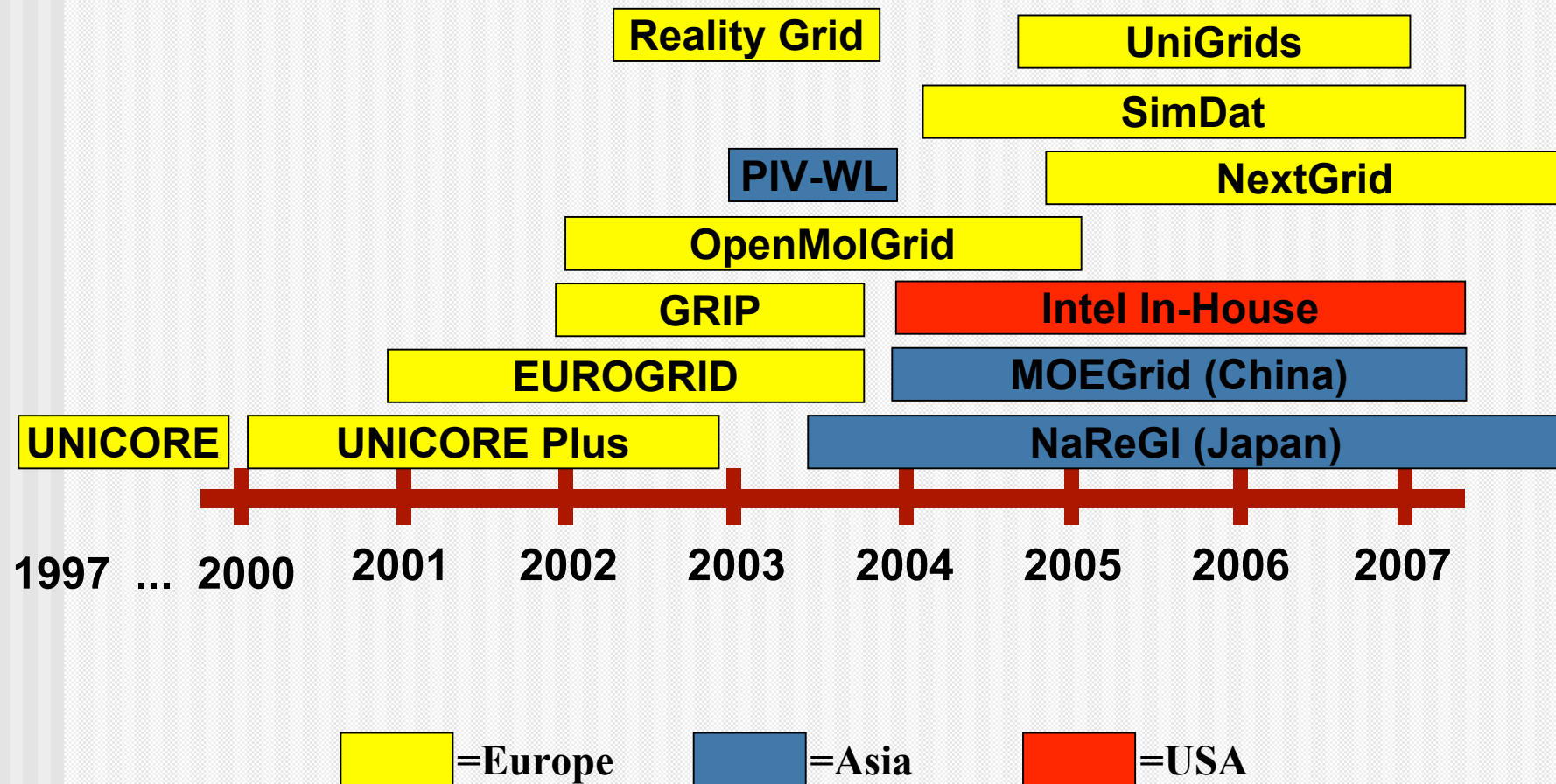
Sven van den Berghe
Fujitsu Laboratories of Europe
sven.vandenberghe@uk.fujitsu.com

UNICORE
Grid Summer School, July 28th, 2004

Contents

- Part 1: UNICORE Architecture
 - What UNICORE is and does
 - UNICORE Architecture
 - Important UNICORE Concepts
- Part 2: UNICORE Server Components
 - Functions and capabilities
 - Extensibility
 - Deployment:
 - Requirements
 - Possibilities and choices

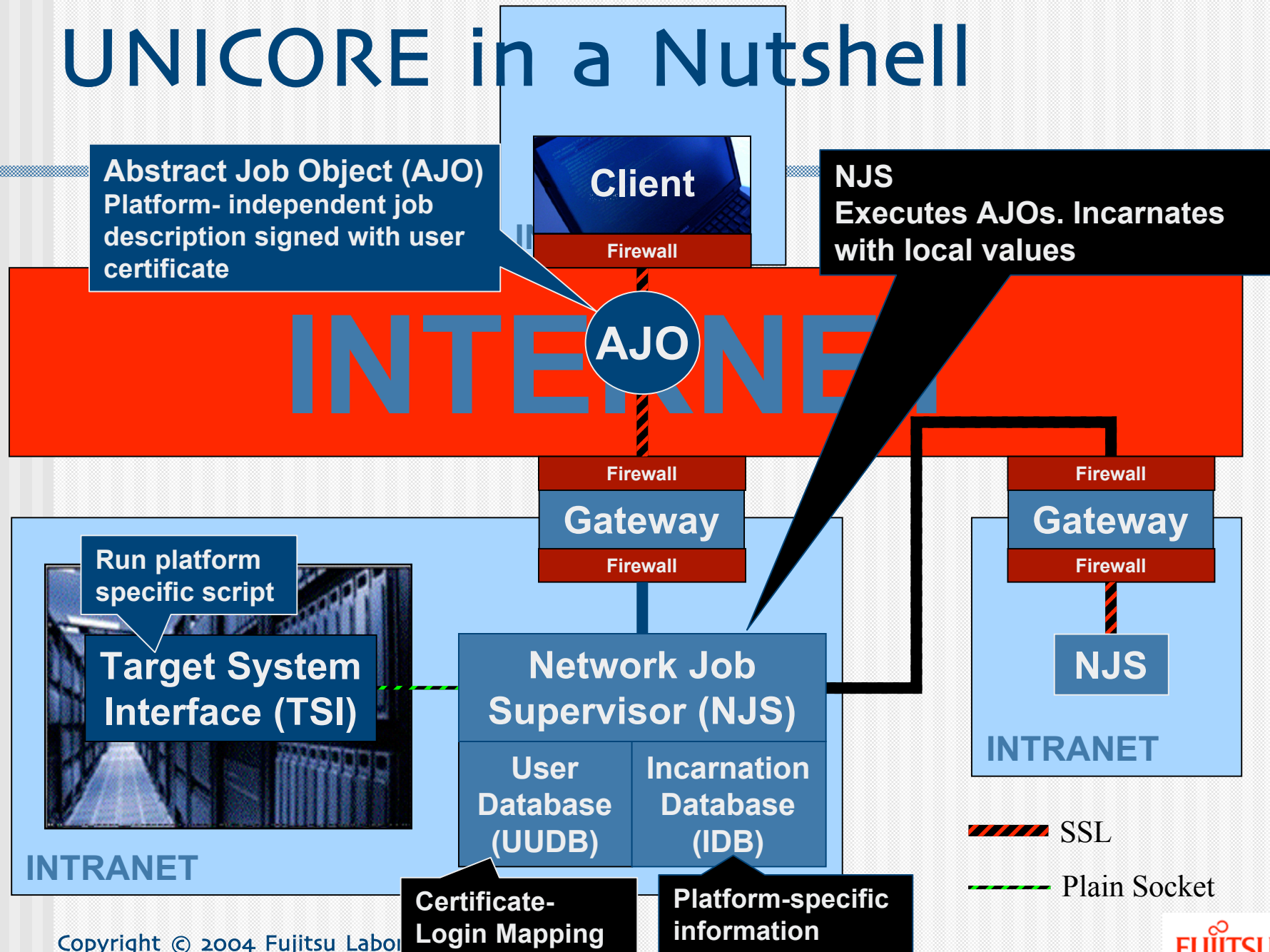
UNICORE History



Concepts

- Abstract Job Object
 - Seamlessness
 - Incarnation
 - Distributed
- Components
 - Servers
 - Gateway
 - NJS
 - Clients
- Security

UNICORE in a Nutshell



Copyright © 2004 Fujitsu Labor

Picture based on an original by Intel

Certificate-Login Mapping

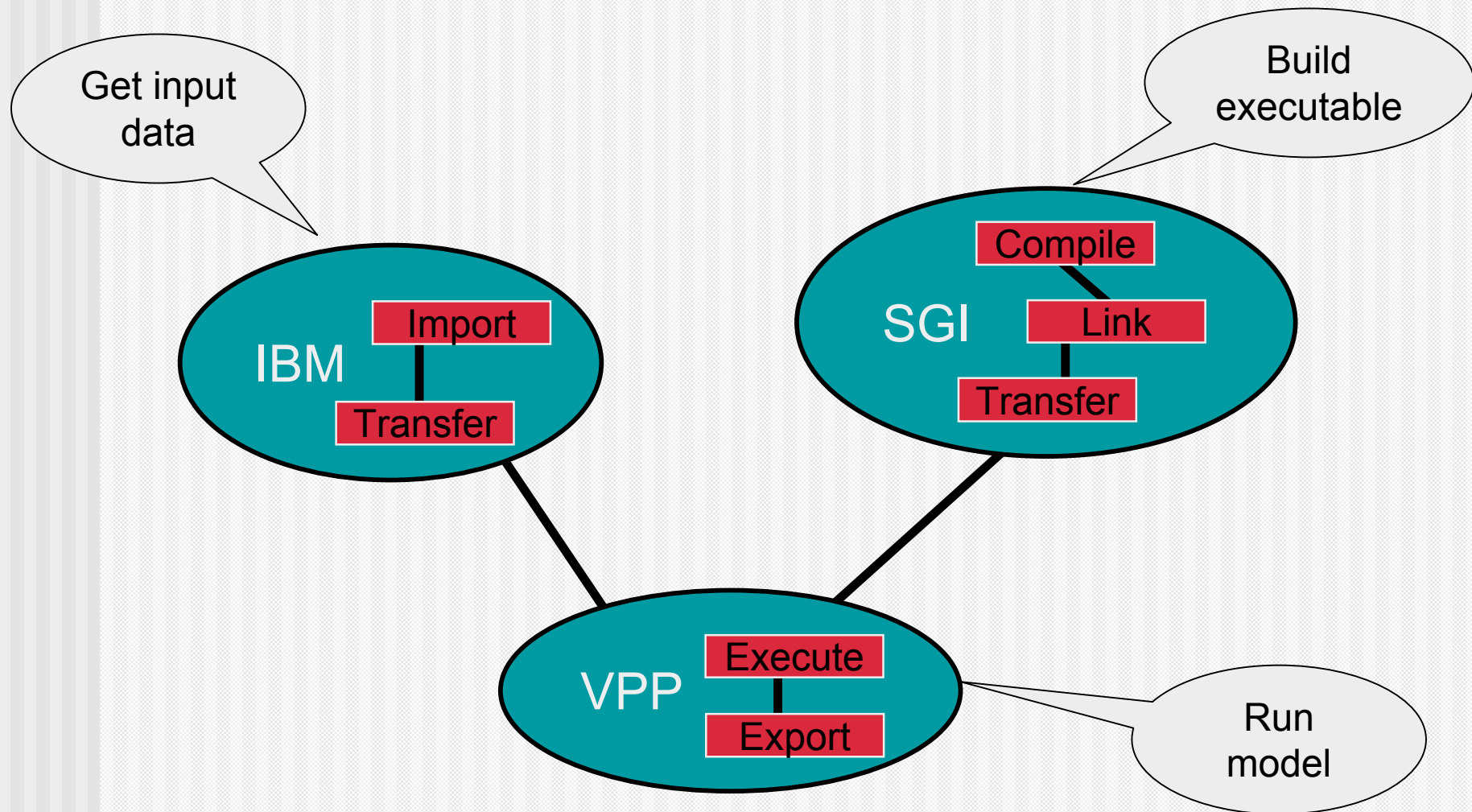
Platform-specific information



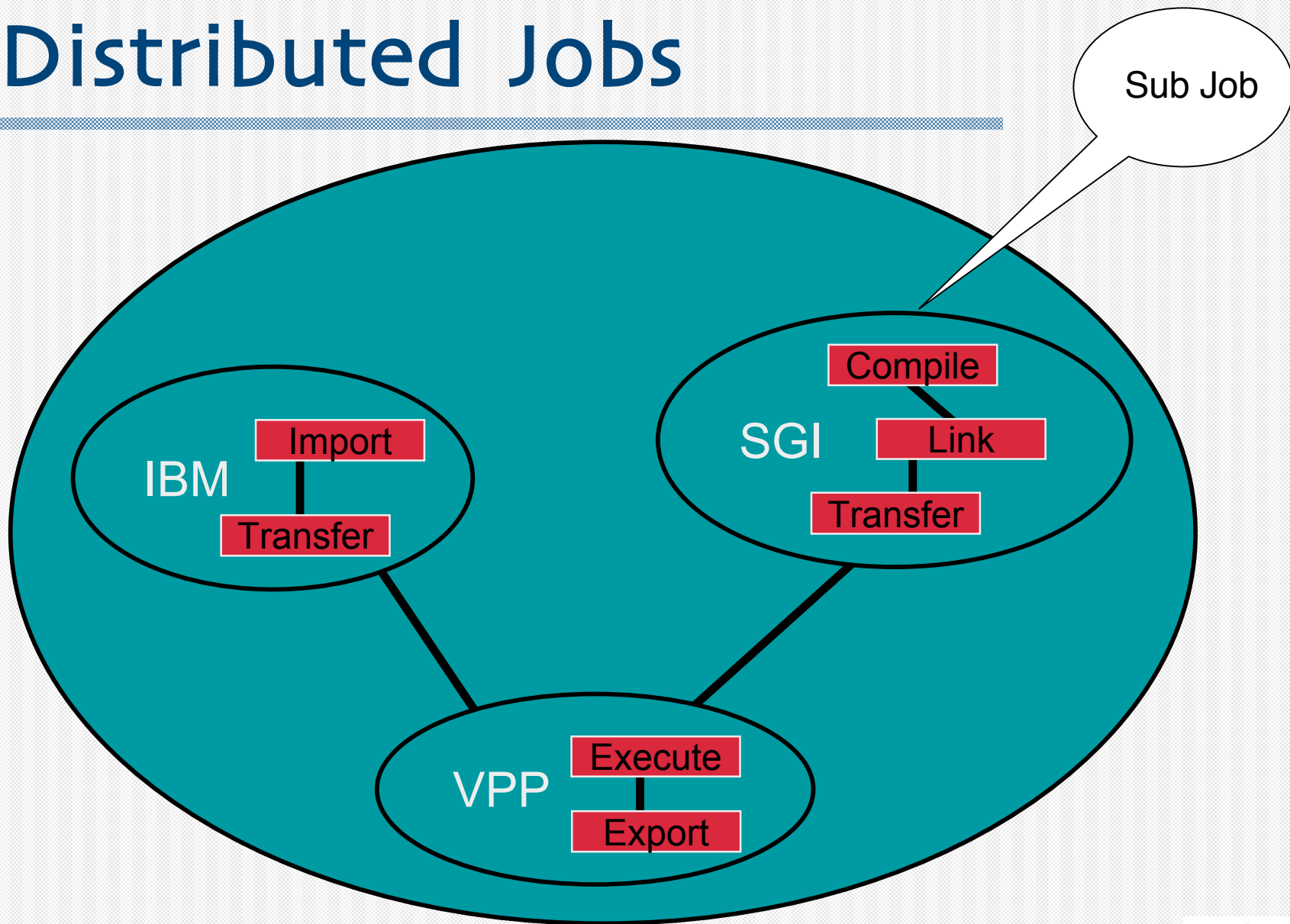
Distributed Jobs

- Use case for architecture of Unicore
 - Job uses resources that are sited at a number of different resources
- Example:
 - Fetch data - from storage server hosted on an IBM
 - Build executable - cross compilation on a SGI
 - Execute job - on a VPP

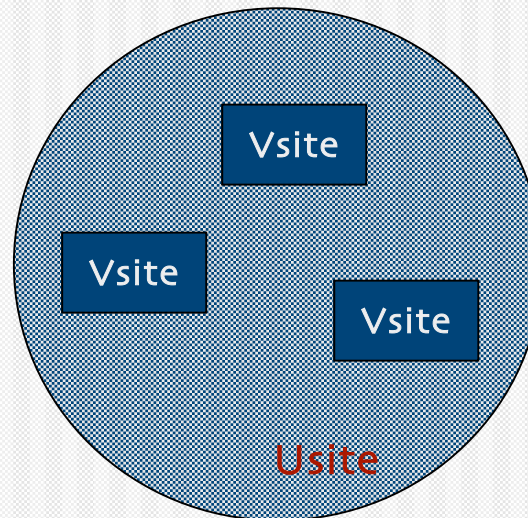
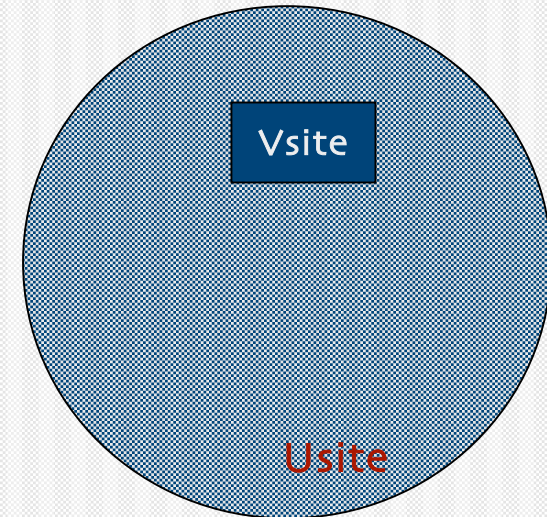
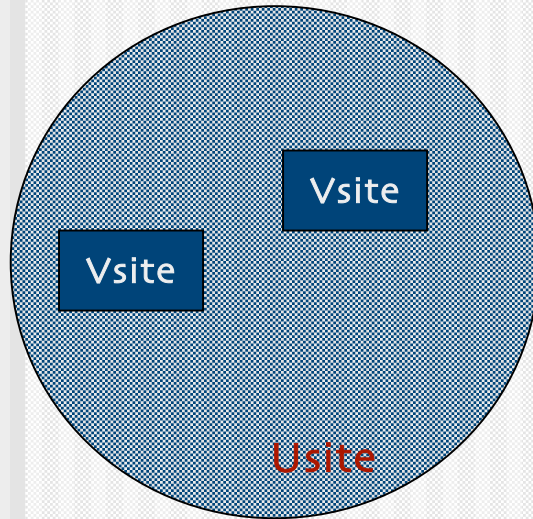
Distributed Jobs



Distributed Jobs



Usites and Vsites



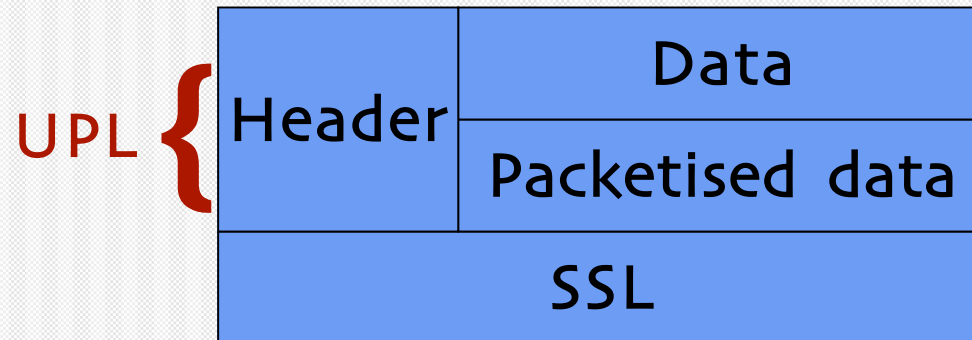
Vsites

- Where Jobs are executed
 - Some sort of computational resource
 - e.g. interface to a Batch sub-system
- Has Hardware resources
 - Processors, memory, file store etc
- Has Software resources
 - Compilers
 - Scripts
 - Applications
 - Packages: Gaussian, StarCD etc
 - Local specials
- Resources are seamless/abstracted

UNICORE Protocol Layer

- Protocol layer below work description and data transfer in Unicore
- Level interpreted by Gateways
- Allows security and switching without need for interpretation by Gateways

UNICORE Protocol Layer



Headers:

- ConsignJob
- RetrieveOutcome
 - retrieveOutcomeAck
- ListVsites

SSL:

- authentication
- integrity
- privacy

Data:

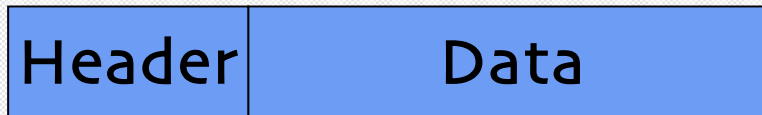
- ZIP stream
- compression

PDS:

- Gateway traversal

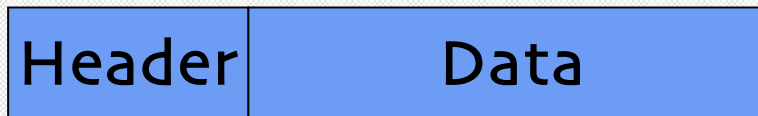
Synchronous - Client initiated

UPL: ConsignJob



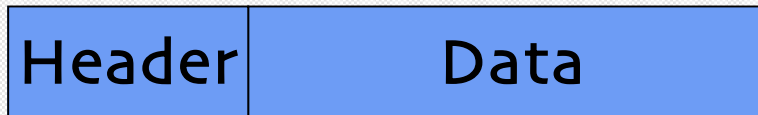
- Send some work to a server
- Header
 - AJO
 - Signature of AJO
 - Signer
- Data
 - Client local files available to job at start of execution
- Reply header is an acknowledge (no data)

UPL: RetrieveOutcome



- Request status of a Job's execution
- Header
 - Job identifier
 - (identity of requestor from SSL)
- No Data
- Reply
 - Header: status
 - Data: files that the Job requests are sent to client
 - Only if Job complete
- RetrieveOutcomeAck: Job deletion

UPL: ListVsites



- Request Vsites behind a Gateway
 - Header: empty
 - No data
- Reply
 - Header: list of Vsites
 - Characteristics
 - Types (possibly non-UNICORE)
 - No Data

Abstract Job Object

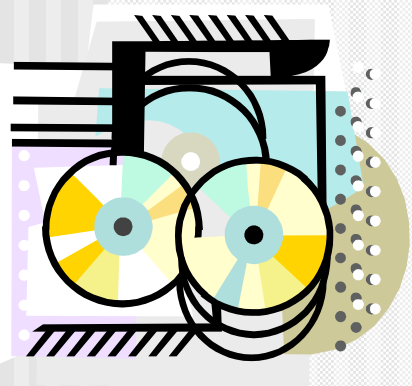
- Describes actions that a Job can take
 - Execution - scripts, executables
 - File and data transfer
 - Control
 - Flow
 - Lifetime management
- A resource model
- Defines the form of the results (Outcomes)
- Java classes (at the moment)

AJO: Seamlessness

- A Job defined in an AJO is seamless and can be retargeted to another Vsite with no (or minimal changes)
- A seamless job requires **incarnation** - conversion to scripts that can be executed on the local system with local values.
- NJS performs incarnation.

Storage model for seamlessness

A Job has a number of different types of storage available at a Vsite



Uspace: Job working directory. Initially empty. Transfer files in from other spaces and direct from client. Deleted at end of Job execution.

Spool: quasi-permanent storage to transfer files between Jobs (seamless abstraction)

Outcome: write-only area. Returned to client.

File Systems: (inc Home, Temp). Normal systems. Availability is seamful.

Seamless resources

- Same resource types are used to requests (in jobs) and advertisements (from Vsites)
- Ideal for a job is a single description that applies unchanged to all Vsites
 - Provided that they can do the same things
 - Simplifies brokering

Capacity resources

- Resources that have some quantity/measure
 - Nodes/processors
 - Memory
 - Network
 - Storage
 - Time
- Seamless
 - Run time is a time *and* a rate
 - Supports rescaling for different machines
 - Aim is for “application units”
- Currently a simple abstraction of architecture

Capability resources

- Can do something
- Most useful is software resource - Application
 - Abstraction
- Brokering support

An AJO is a DAG

- AbstractActions are executed when all predecessors are successfully complete
 - Need to specify dependencies
- A failed AbstractAction terminates execution of its enclosing DAG
 - Flag to ignore failure

AJO: Job flow

- Conditional tasks
 - Test on final status of a predecessor
 - Test on a code set by a predecessor
 - Decidable tasks e.g. file status, exit status of a script, time
- Loops
 - For
 - Repeat on value of a Decidable

AJO: Execute Tasks

- Core of a Unicore job - does something on the Vsite's systems
 - Thus - all ExecuteTasks require resources
- Can execute:
 - Scripts
 - Files transferred into the Uspace by any means
 - ... and files produced by other tasks (compile/link)
 - Executables referred to by a Resource
 - Seamless execution of packages
 - Control capabilities of users

AJO: File and data transfer

- Transfer files to and from all UNICORE file spaces
 - Spool on the local Vsite
 - Any defined file systems on the local Vsite
 - Any other Uspace on any other Vsite
 - ... and so file transfer to remote Vsites
- Transfers usually have the Job's Uspace as the source or destination
- Remember that there is also UPL file transfer
- Interact with local file systems
 - List, move, copy, delete etc

AJO: Control

- AJOs can control other AJOs:
 - Return current status
 - Hold/resume
 - Abort
 - Fetch current results
- Permissions controlled by UNICORE identity

AJO: sub-Jobs

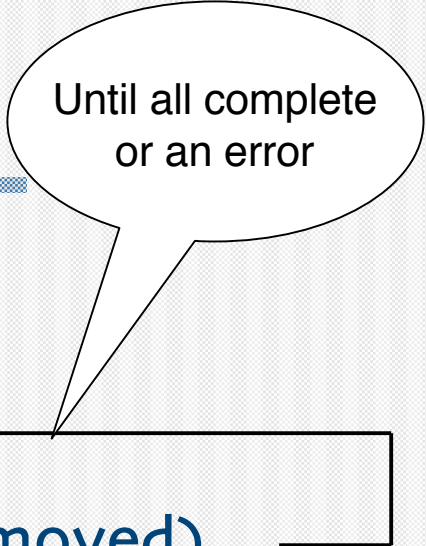
- Part of an AJO's DAG
- Consigned by NJS to another Vsite
- Parent NJS monitors execution
 - Uses UPL to poll
 - Fetches Outcome (to return to its parent)

AJO: Life cycle

- AbstractActions can be in one of three states:
 - PENDING
 - EXECUTING
 - DONE
- AbstractActions go to PENDING when they arrive at their **target** Vsite
- AbstractActions go EXECUTING if and when the predecessor dependencies allow
- Go DONE when execution complete (or not executed)
 - Remain "in" UNICORE
- AbstractActions leave UNICORE when their parent AJO is deleted (e.g. subject to a UPL RetrieveOutcomeAck)

AJO: Execution cycle

- AJO received by Vsite
- Uspace created, Outcome created
- Available AbstractActions executed
- AbstractAction finishes (dependency removed)
- AJO completes
- Uspace deleted
- Outcome still available to UPL
 - Spooled files still available other AJOs
- ROA received Outcome deleted
 - Spool OK



Until all complete
or an error

AJO: Lifetime

- Every AbstractAction has a Termination Time
 - Set when Job created
 - Can be changed when an AbstractAction is PENDING or EXECUTING
- When the Termination Time passes:
 - AJOs are deleted
 - Other AbstractActions are aborted

Authentication

- Client-server communication is over an authenticated SSL connection
- Jobs must be signed
- Users are issued with a long-term SSL certificates by a CA trusted by the Usite
 - Standard client certificate
- Servers are issued with a long-term SSL certificates by a CA trusted by the Usite
 - Standard client certificate
- Authentication is seamless to the User
 - Single sign-on to client

Trust: roles

- Endorser - the user who created the Job
 - Public certificate is passed in the header of a ConsignJob
 - AJO is signed by the endorser
- Consignor - the entity that submitted the Job
 - Public certificate is extracted from the SSL connection
 - Can be either a user or an NJS

Authorisation

- Site
- NJS
- User

Site Authorisation

- All all incarnations of AbstractActions are executed on behalf of a User using a local account (Xlogin)
- Users are identified by the Endorser certificate
- Authorisation is done by the NJS
 - Gets a mapping mapping of Endorser to Xlogin
 - Gets the type of the Consignor
 - NJS implementation issue, see later
- Site authorisation applies to the Xlogin
 - All incarnated actions are controlled by this

NJS Authorisation

	User	Server
Create AJOs (be endorser)	✓	✗
Submit AJOs (be consignor)	✓	✗
- self	✓	✗
- others	✗	✓
UPL on AJOs consigned by		
- self	✓	✓
- others (endorsed by self)	✓	✗
Act on AJOs endorsed by self	✓	✗

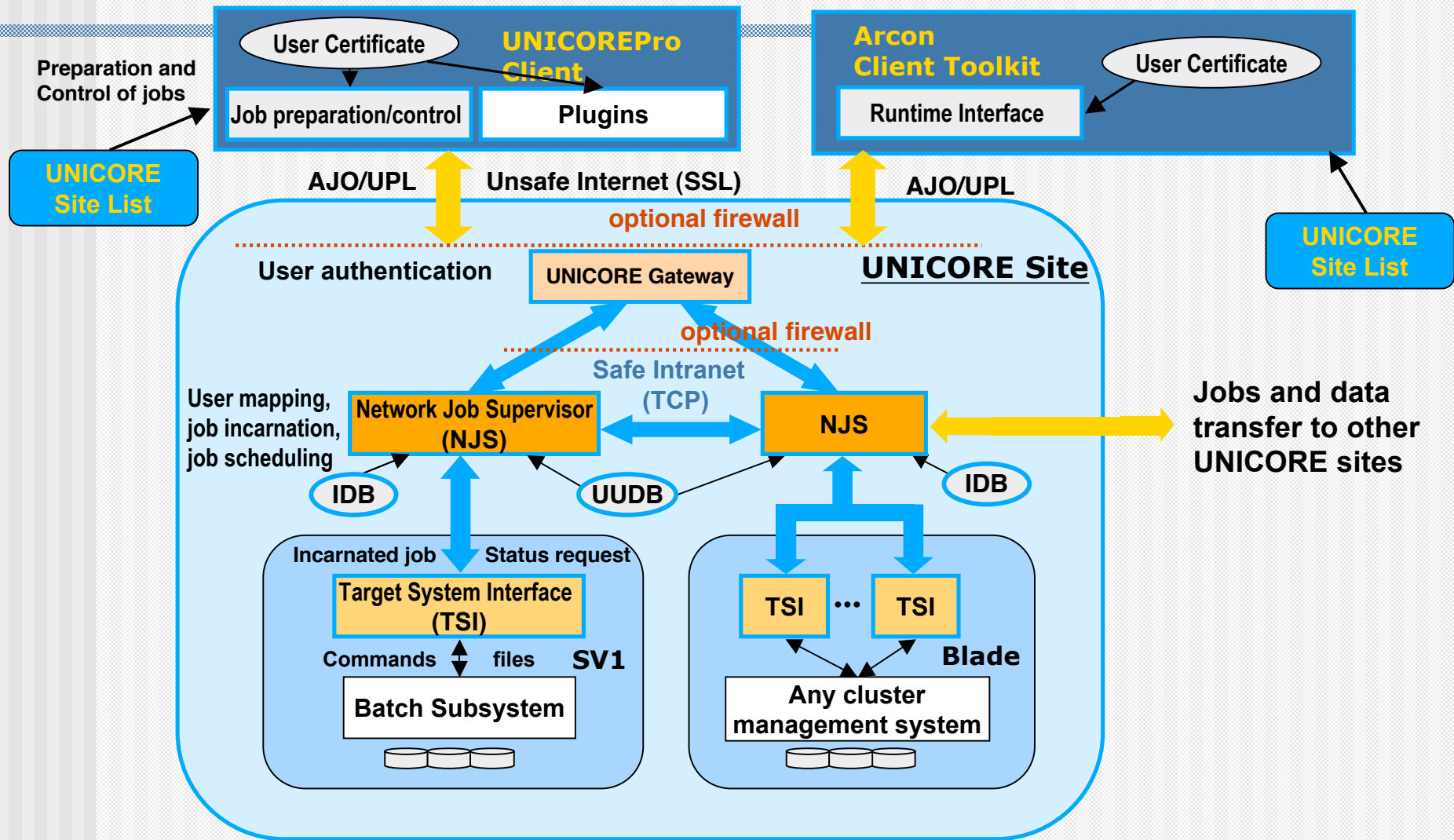
User Authorisation

- Uses delegates to servers by signing an AJO
- Only the job as described in executed
- Safe
- but too limiting?
 - Brokers
 - Portals
 - Dynamic jobs

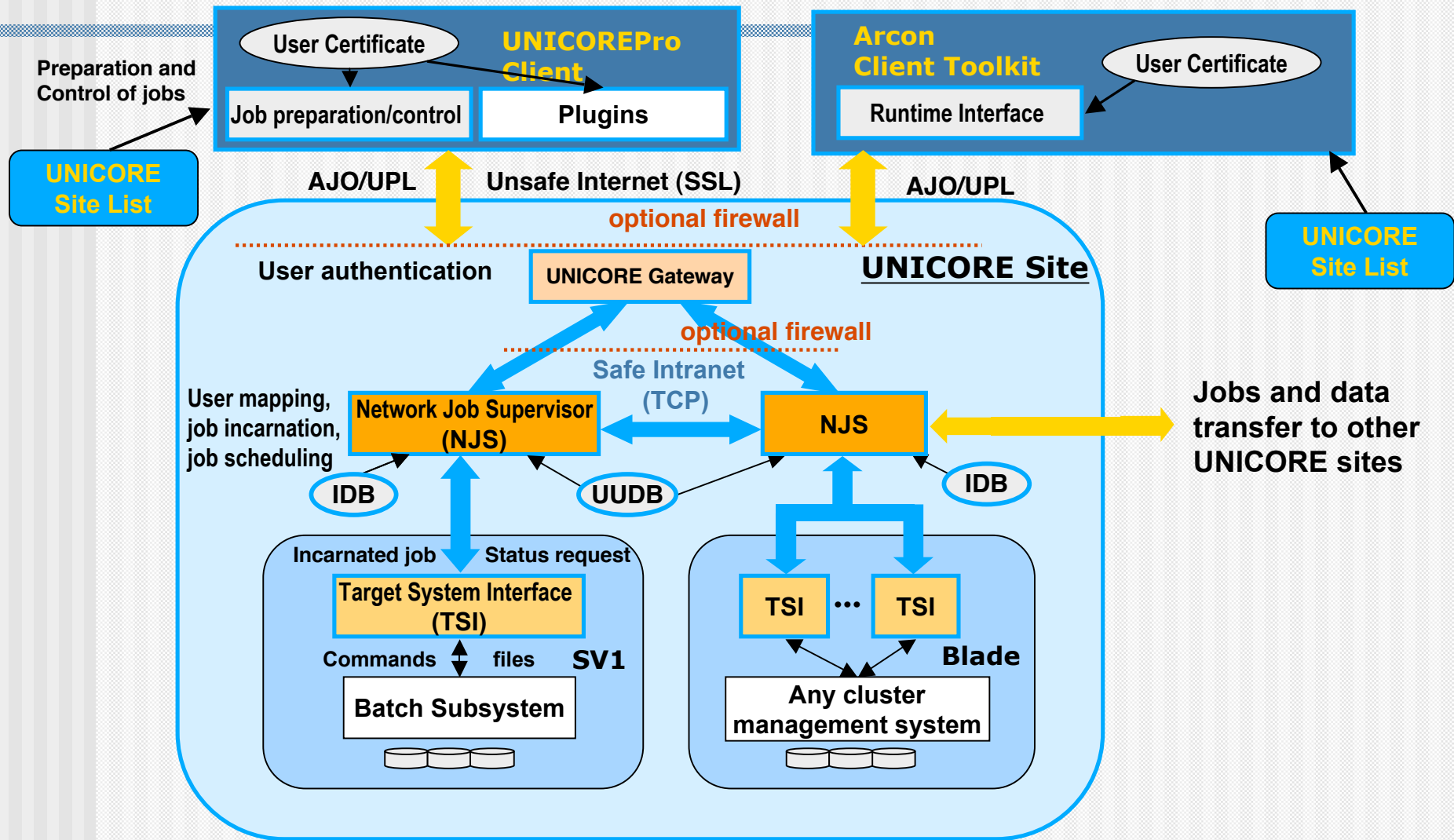
Security: notes

- Privacy
 - in transmission through SSL
 - locally through site mechanisms
 - UNICORE users protected from each other by NJS authorisation (even if mapped to same Xlogin, with caveats)
- Minimal changes to site policies
 - Single port used by Gateway for all in and out bound
 - Gateway to NJS in known IP to known IP address

Architecture



Architecture



Gateway: functions

- Authenticates identity of peer
 - First authorisation step, only allows connections presenting valid certificates from acceptable Certificate Authorities
 - Makes peer certificate available to NJS for further authorisation
- Presents a single point of entry for all UNICORE services at a Usite
 - Switches connections to correct Vsite
 - Only need to open a single port
- Supplies a list of Vsites

Gateway: processing

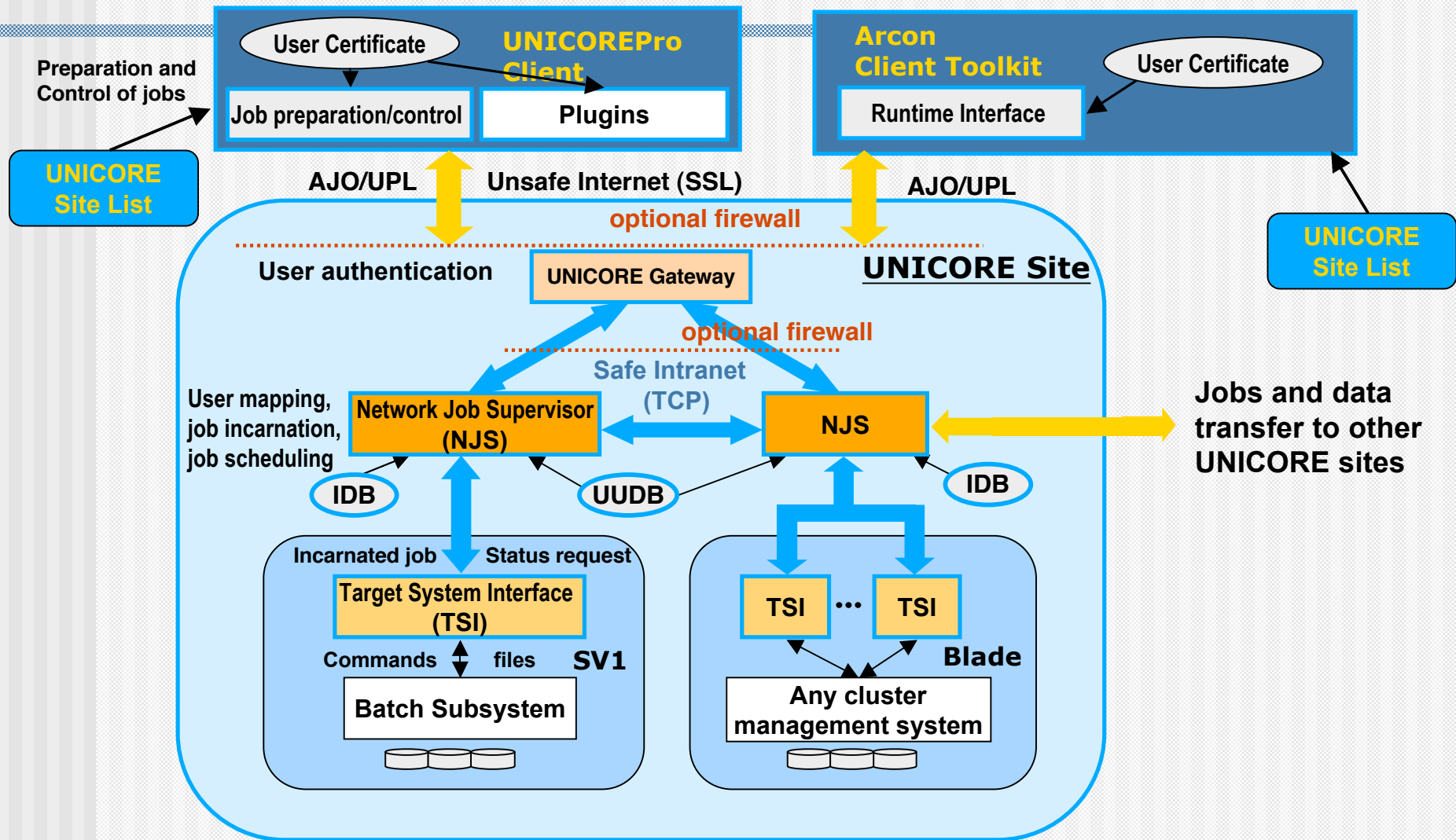
- Accepts connection
- Deserialises header to determine Vsite
 - Does not deserialise AJO
- Sends header to target Vsite
- Forwards following data to Vsite (as packetised stream)
- Waits for reply and forwards this to peer

- “Plug-in” interface makes this available to other protocols
 - Requires UPL header, rest is up to implementation

Gateway: deployment

- A Gateway is required for every Usite
- Requirements
 - Java 1.4
 - One or more certificates of Certificate Authorities that you trust to issue User and Server certificates
 - A private key (and certificate signed by a CA)
- Location
 - In your DMZ
 - Requires a port from Internet to Gateway
 - Requires “pinholes” from Gateway to NJS(s)
 - Otherwise on same machine as NJS

Architecture



NJS: Network Job Supervisor

- Core UNICORE server
- Executes AJOs

NJS: processing

- Authorises consignor and endorser
- Accepts AJO
 - Writes UPL data stream directly to Uspace on target system (through TSI)
- Processes DAG, executing AbstractActions
 - Uses TSI to execute scripts
 - Passes sub-AJOs to target Vsites
 - Monitors progress of AbstractAction execution
- Completed AJOs are retained until:
 - RetrieveOutcomeAck received
 - TerminationTime passes

NJS: administration

- Interface for administration
 - Command line client
 - GUI client
- Available operations include
 - List Jobs
 - Delete Jobs and AbstractActions
 - List and manage Uspace and Spool files
 - TSI control
 - Logging
 - System status monitoring
- Administration is for a local NJS only

NJS: recovery and logging

- Extensive logging
 - Levels can be changed at run time
- NJS can recover from crashes and shut-downs
 - Maintains state of executing AJOs for recovery
 - User's data is kept on target system, independently of NJS
- Restart and recovery of failed data transfers coming soon

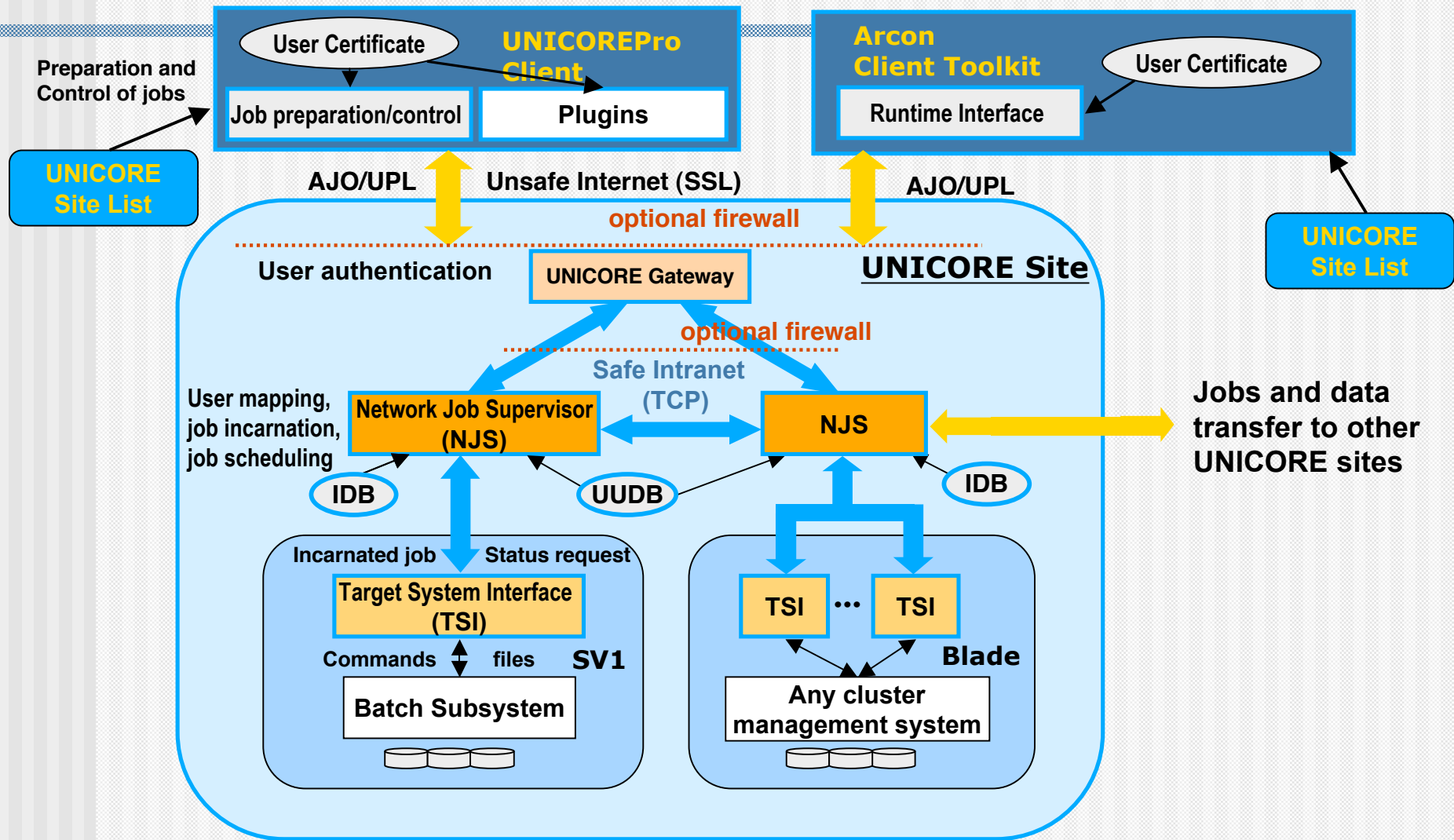
NJS: extensibility

- Published interfaces for:
 - Broker
 - UUDB
 - Alternative File Transfer
 - Interaction with accounting systems

NJS: deployment

- Usually on a machine separate from target system
 - DMZ is not suitable:
 - Gateway is simple, NJS is complex
 - NJS does authorisation (TSI setuid is exported to it)
- Requirements
 - Java 1.4
 - Certificates of CAs that issue Gateway certificates of external Usites
 - One or more private key/signed certificates to contact external Usites
 - Configured IDB
 - Configured UUDB

Architecture



UADB: UNICORE User Database

- Maps from Certificate to a local user and/or role
- Authorisation step
 - An Endorser mapping must exist for the UNICORE job to be executed
 - A Consignor mapping must exist for the UNICORE job (or UPL request) to be accepted by the NJS
- NJS publishes an interface so that this can be integrated into a site's mechanisms
 - Two sample implementations available

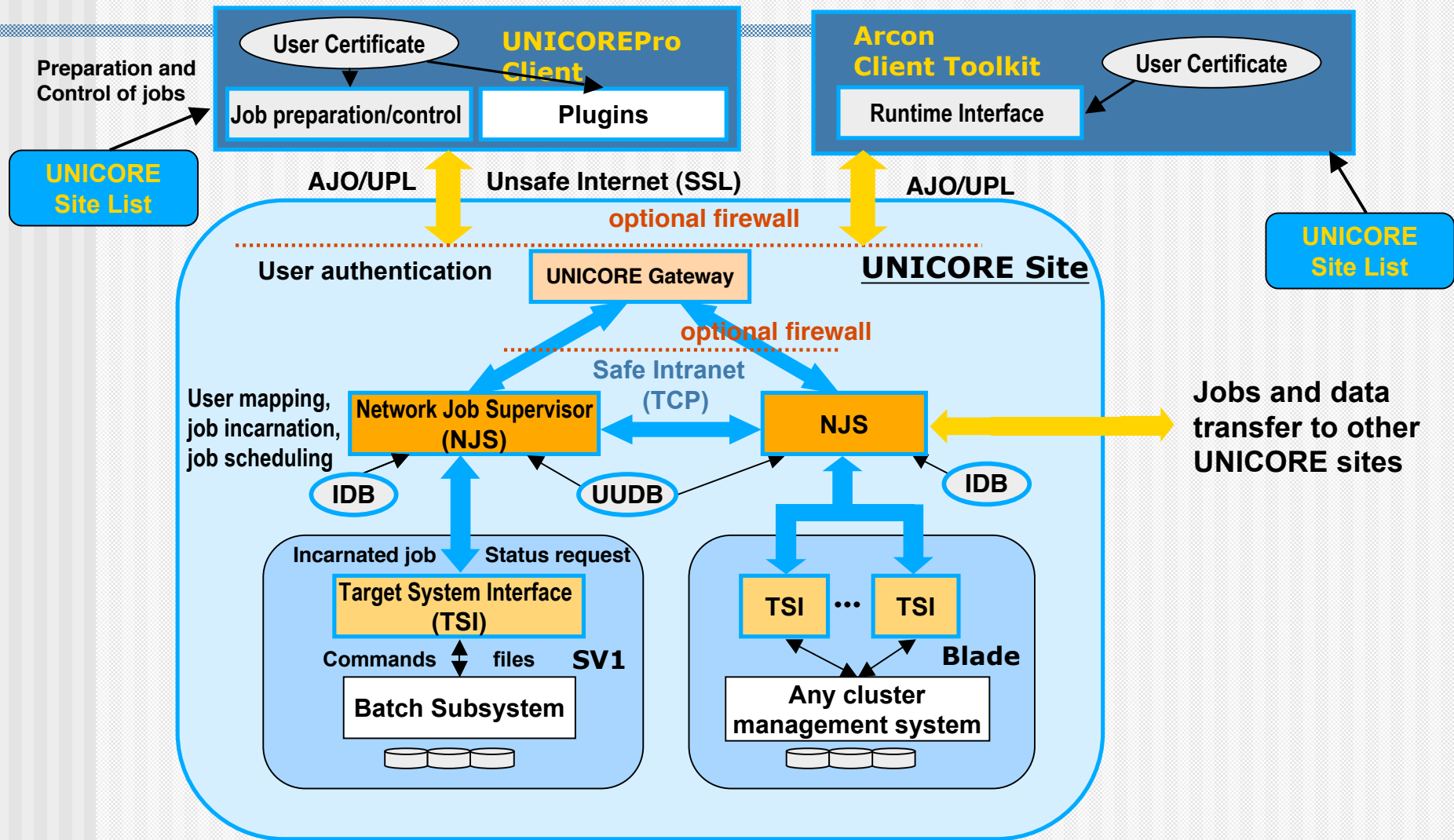
UADB: Mapping strategies

- One to one
- Many to one
 - All certificates from a CA map to one Xlogin
 - All proxy certificates from a User map to an Xlogin
- One to many
 - User can select Xlogin and project to use

UADB: roles

- User
 - Can be endorser and consignor
 - Can execute jobs on the local system
- Server
 - Consignor only
- Broker
 - Limited form of User so that a Resource Broker can create and manage query jobs (nothing executed outside NJS)
- NJS Administrator

Architecture



IDB: Incarnation Database

- **Incarnation** is the process of converting the seamless description of a job in an AJO to scripts that execute the commands
- IDB has two main functions:
 - describes the resources available at a Vsite
 - describes how AbstractActions are implemented as scripts
- IDB is a file with a simple language to structure the information
- An IDB must be created for each Vsite

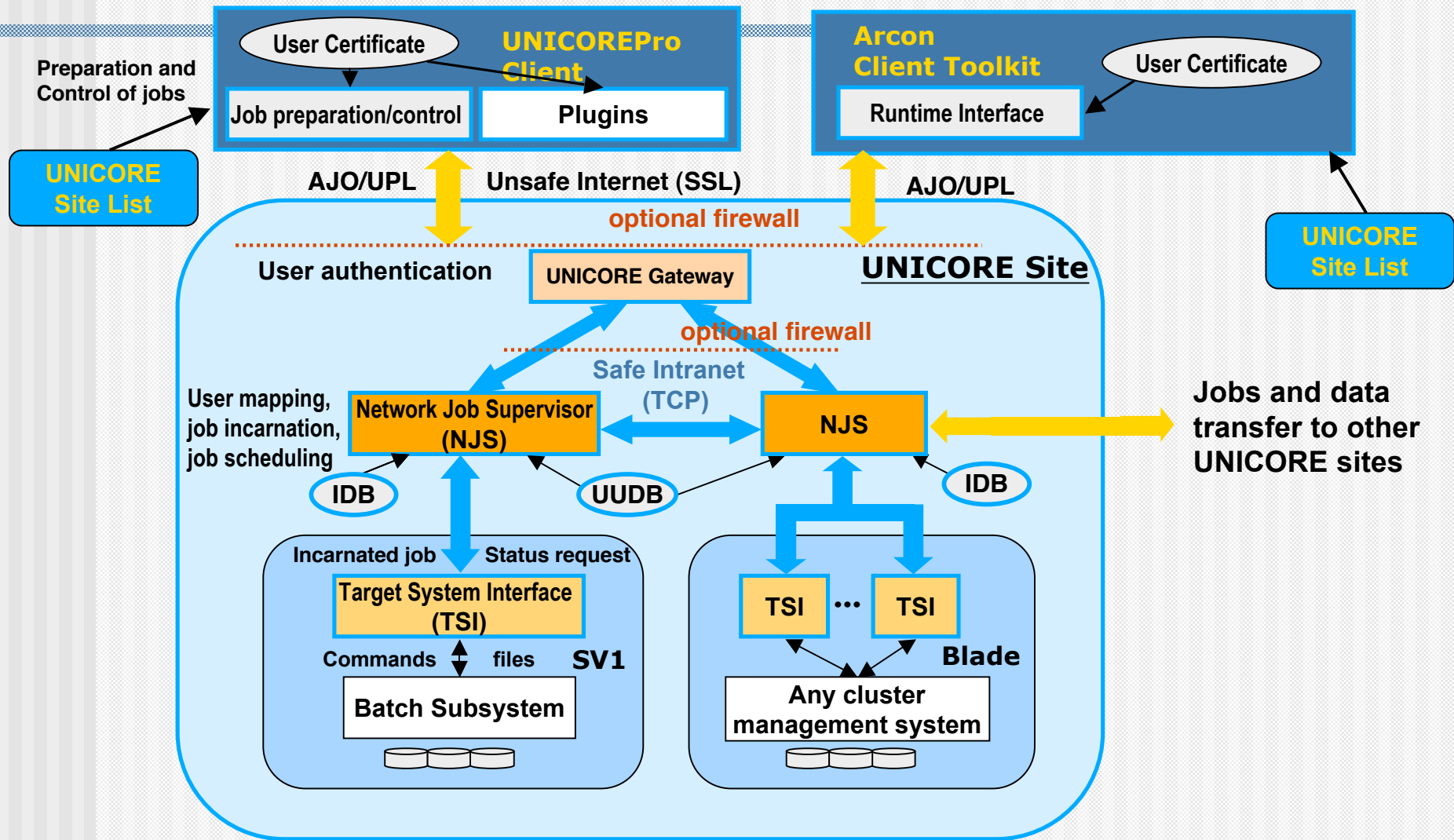
IDB: creating

- Experience has been that the bulk of the IDB is constant, changes are usually only required for the different resources
 - Hardware
 - Software
- Example IDB has four sections:
 - Description of resources
 - Paths to common Unix commands
 - Description of available software (local)
 - Description of incarnation of AbstractActions
 - Largely constant

IDB: Example

- APPLICATION povray 3.5
- DESCRIPTION "POV-Ray 3.5"
- INVOCATION [echo "Executing POV-Ray...";
cp /home/uspaces/stuff/povray.ini .
env DISPLAY=localhost:o.o
/usr/local/bin/povray -d \$POVRAY_INPUTFILE]
- END

Architecture



TSI: Target System Interface

- Interfaces between UNICORE and the the target resources to:
 - Execute incarnations of ExecuteTasks as batch jobs
 - Execute incarnations of other AbstractActions as scripts
 - Read and write files sent over UPL
- Requires OS and batch sub-system specific code
 - Only other place is the UUDB
- Requires non-user permissions
- Design principles
 - Small as possible
 - lightweight

TSI: deployment

- TSI is deployed on the target system
 - Access to BSS
 - Access to user file systems on which to place Uspaces
- Assumption is Unix
 - Cygwin has been known to work
- Requirements
 - Perl 5
 - Setuid permission if mapping to more than one Xlogin
- See if there is a port for your system available in the distribution
 - Base new port on close match (not too difficult)

