pallas

Member of the ExperTeam Group

The UNICOREpro Client

Programming Client Plug-Ins

Ralf Ratering
Pallas GmbH
Hermülheimer Straße 10
50321 Brühl, Germany

ralf.ratering@pallas.com
http://www.unicorepro.com

Getting started (1)

- Download http://www.unicorepro.com/gridSchool/GridSchoolProject.tar.gz
- Unpack it to your working directory
- Start Eclipse
- Add new java project starting in „GridSchool“ directory
  - Source: GridSchool/src
  - Classes: GridSchool/bin
- Run the Client from the lib directory
  - Run -> new Java Application
  - Main class: com.pallas.unicore.client.Client
- Get a test certificate from the Pallas Test Grid
  - For convenience choose a simple password. It‘s just a test certificate....
- Run a test job (New Job->Add Script „date“)

Getting started (2)

- Edit IP address in GridSchool/files/gridschoolSites.xml
- Add gridschoolSites.xml to User Defaults->UNICORE Sites
- Import certificate GridSchool/certificates/EurogridCA.der in Keystore Editor
- „Refresh“ on „Job Monitor“ to reload UNICORE Sites
- GridSchool Sites should appear now
- Run test job on a GridSchool site

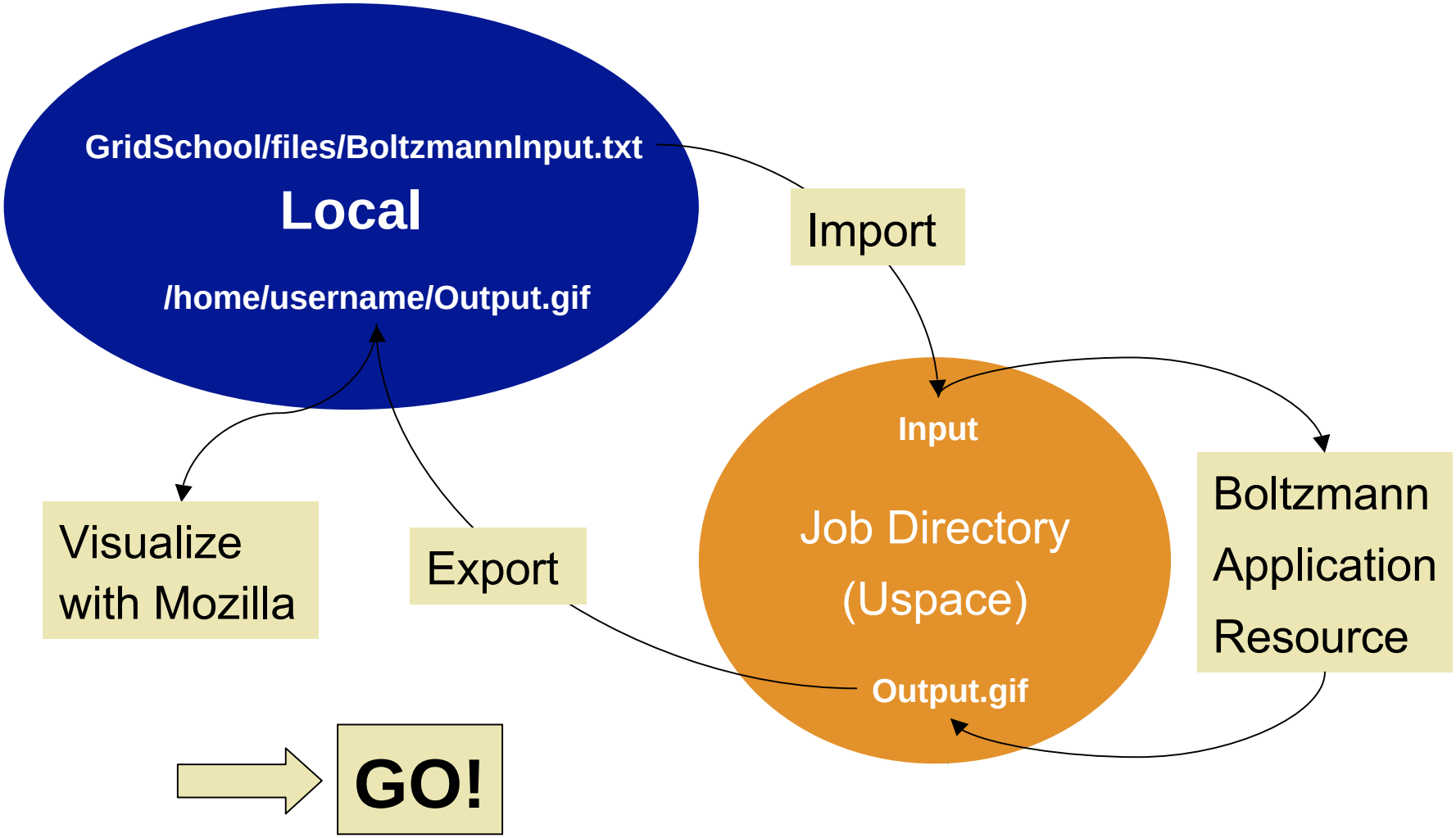
Our Example: The Lattice Boltzmann Application

- Scenario: A site wants to make new application available on the Grid
- Example: Lattice Boltzmann
  - Simulation of fluent mixing
  - Output: a gif animation
  - Intermediate sample files are generated
  - A control file can change parameters while application is executing
- Integrate Boltzmann application into Client GUI with a Plugin!

Overview

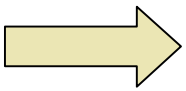
- **Step 1**
  - Use Command Task to run application
  - Specify input and output files in import/export panels
- **Step 2**
  - Write a specialized Boltzmann plugin task
- **Step 3**
  - Edit and automatically send the input file
  - Automatically set output file export
- **Step 4**
  - Get sample files while application is executing
  - Visualize sample files in outcome area
- **Step 5**
  - Start your own little project

Step1: Executing a Command Task



Writing Task Plugins

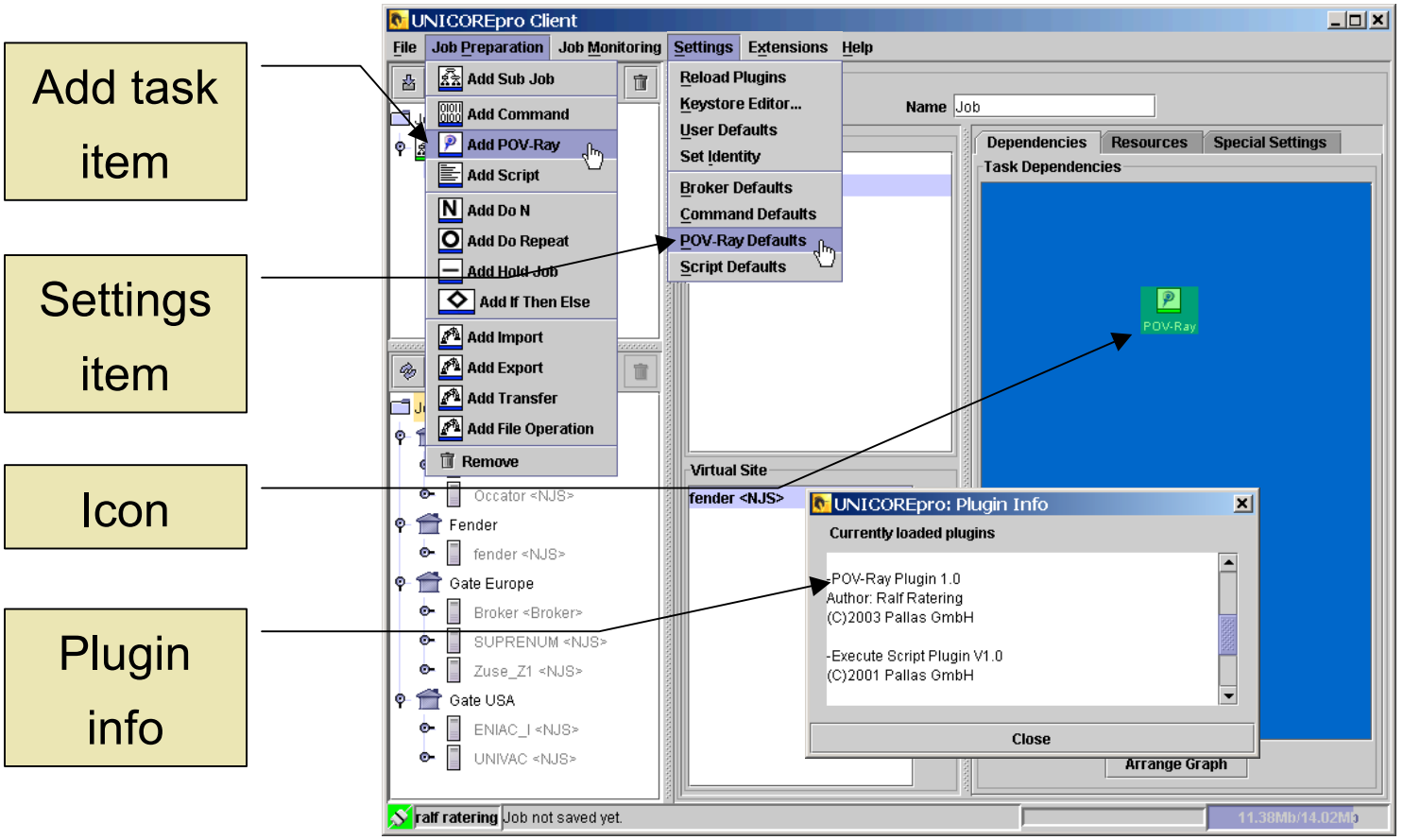
- Disadvantages of Command task
  - Input file has to be edited outside Client
  - Imports and Exports have to be specified manually
  - No integrated GUI for parameters
  - Results have to be visualized outside client
  - No additional functionality possible (sample and control files)



Write a specialized Boltzmann Plugin Task!

Task Plugins

- Add a new type of task to the Client GUI
- New task can be integrated into complex jobs
- Application support: CPMD, Fluent, Gaussian, etc.

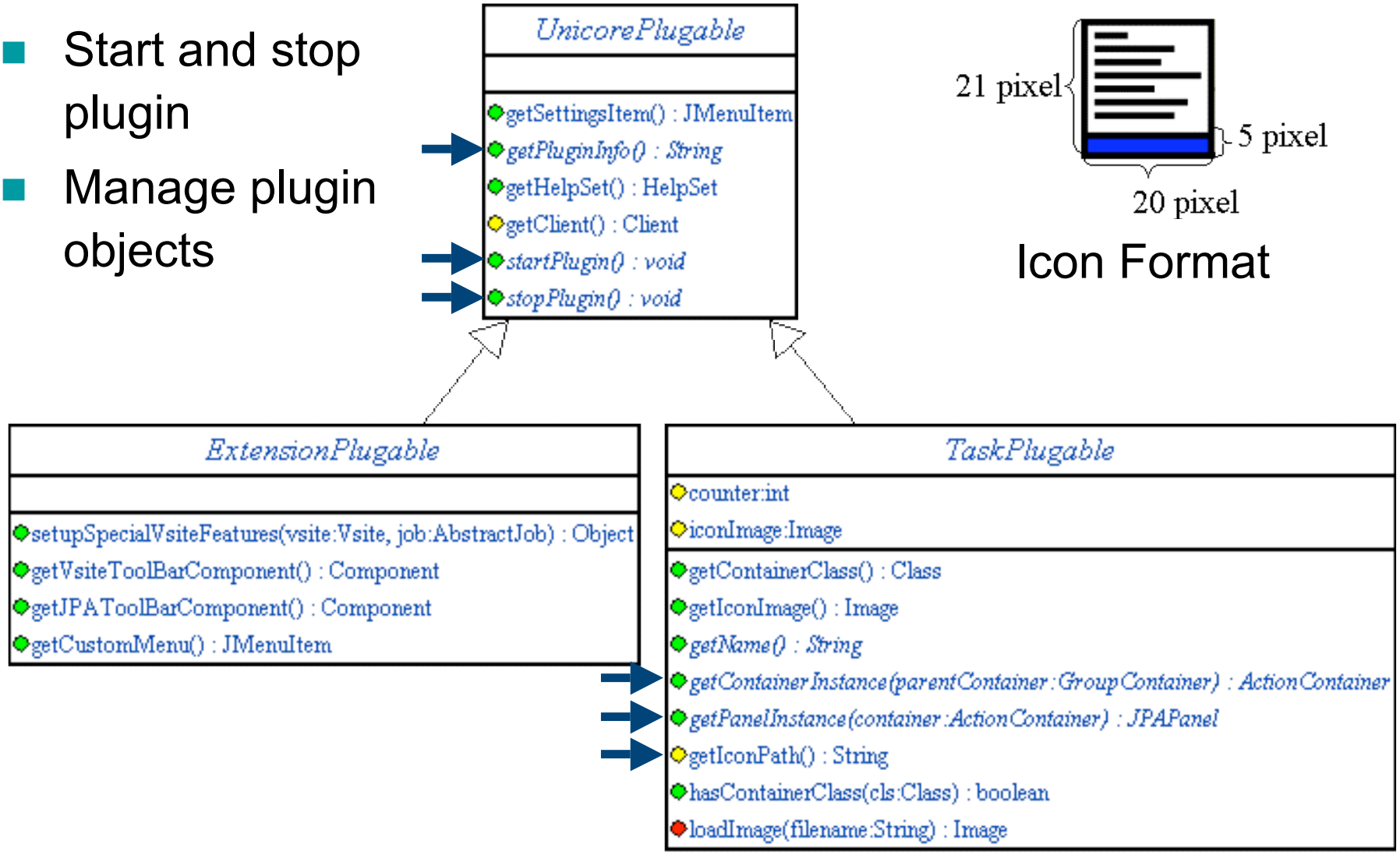


Writing a Task Plugin

- Implement 3 Classes
  - Main plugin class
  - Plugin Container
  - JPAPanel
- Build a Jar Archive named „*Plugin.jar“
- Sign the Jar with your Certificate

Main Plugin Class

- Start and stop plugin
- Manage plugin objects



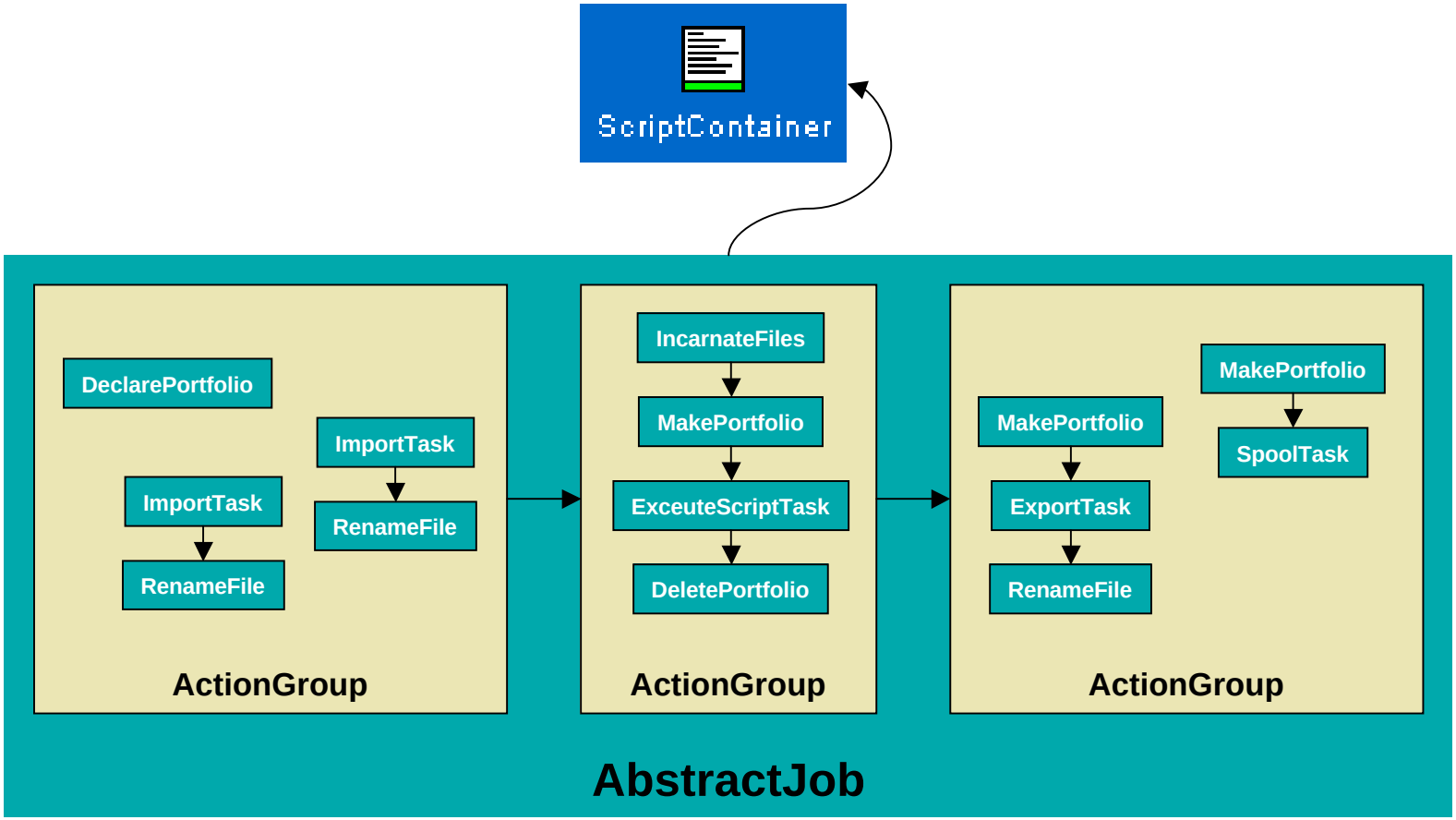
Icon Format

Plugin Container

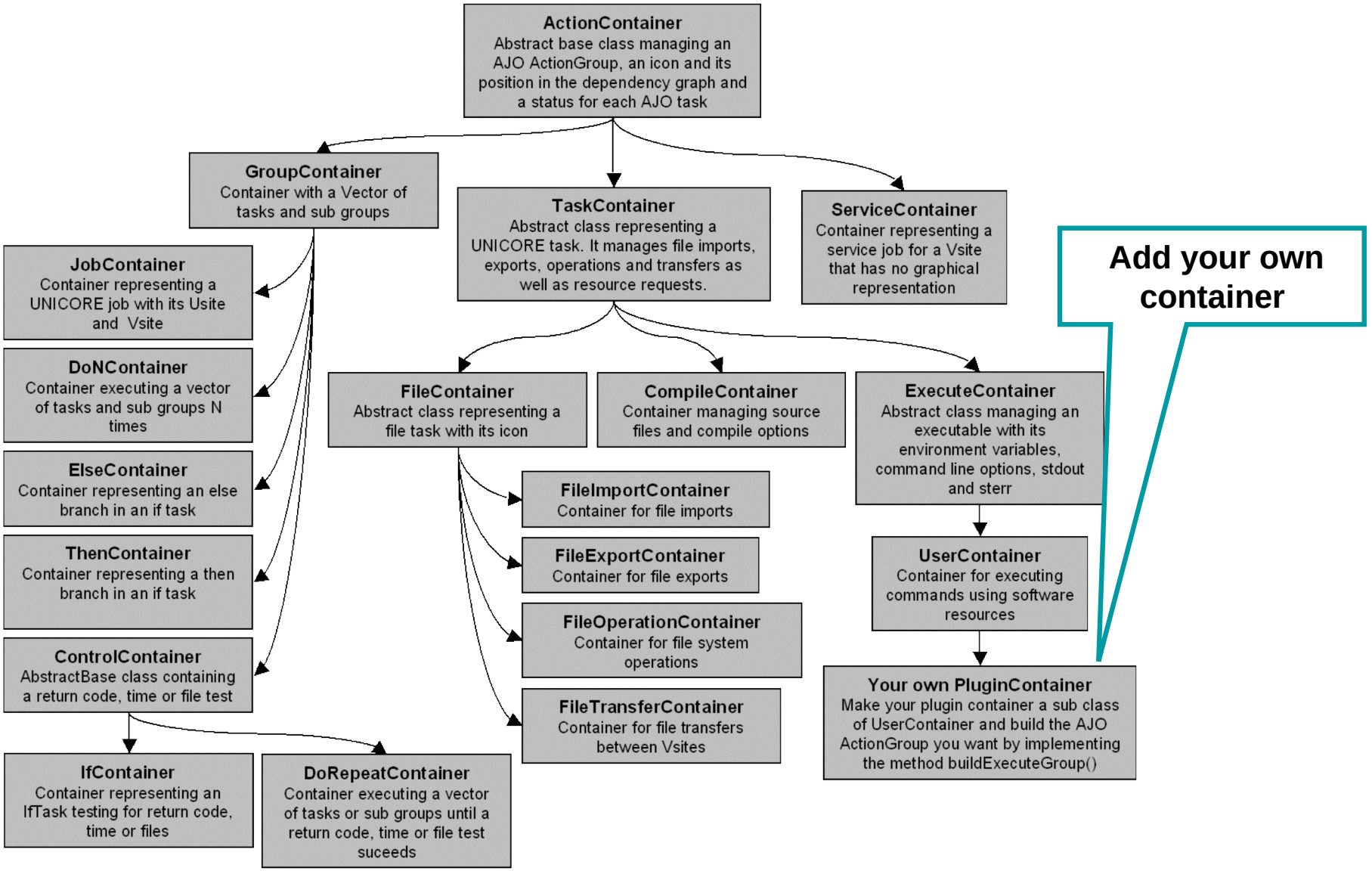
- Build **Abstract Job Object** (AJO)
- Manage imports, exports and execution
- Hold parameters
- Keep status
- Check errors

AJOs and Containers

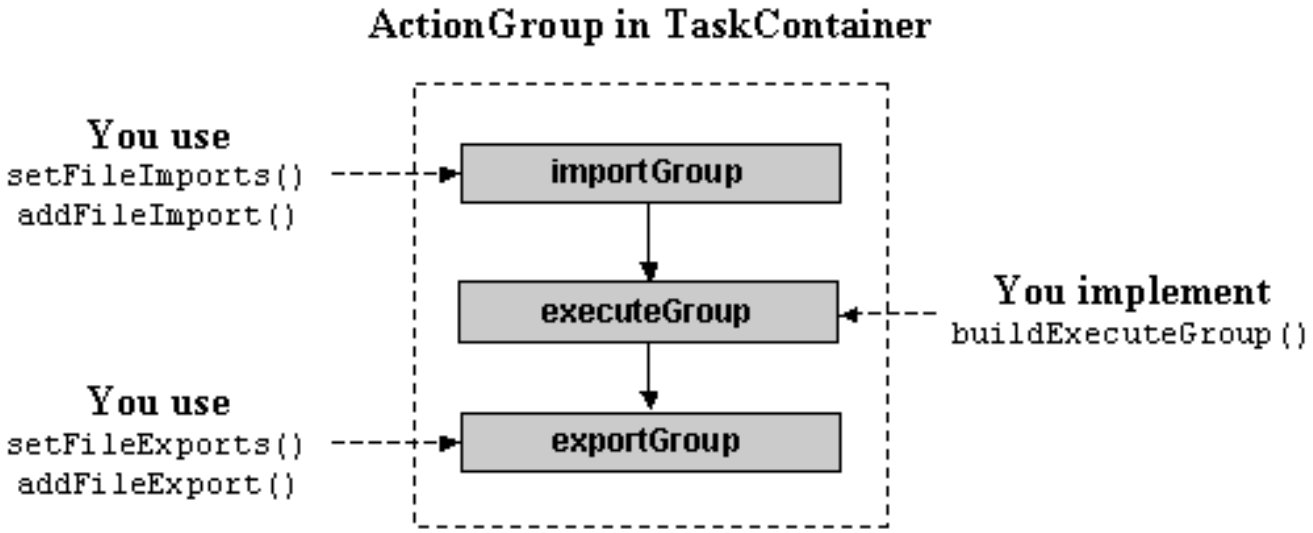
- AJO is the low-level „UNICORE language“
- Client containers encapsulate complex AJOs



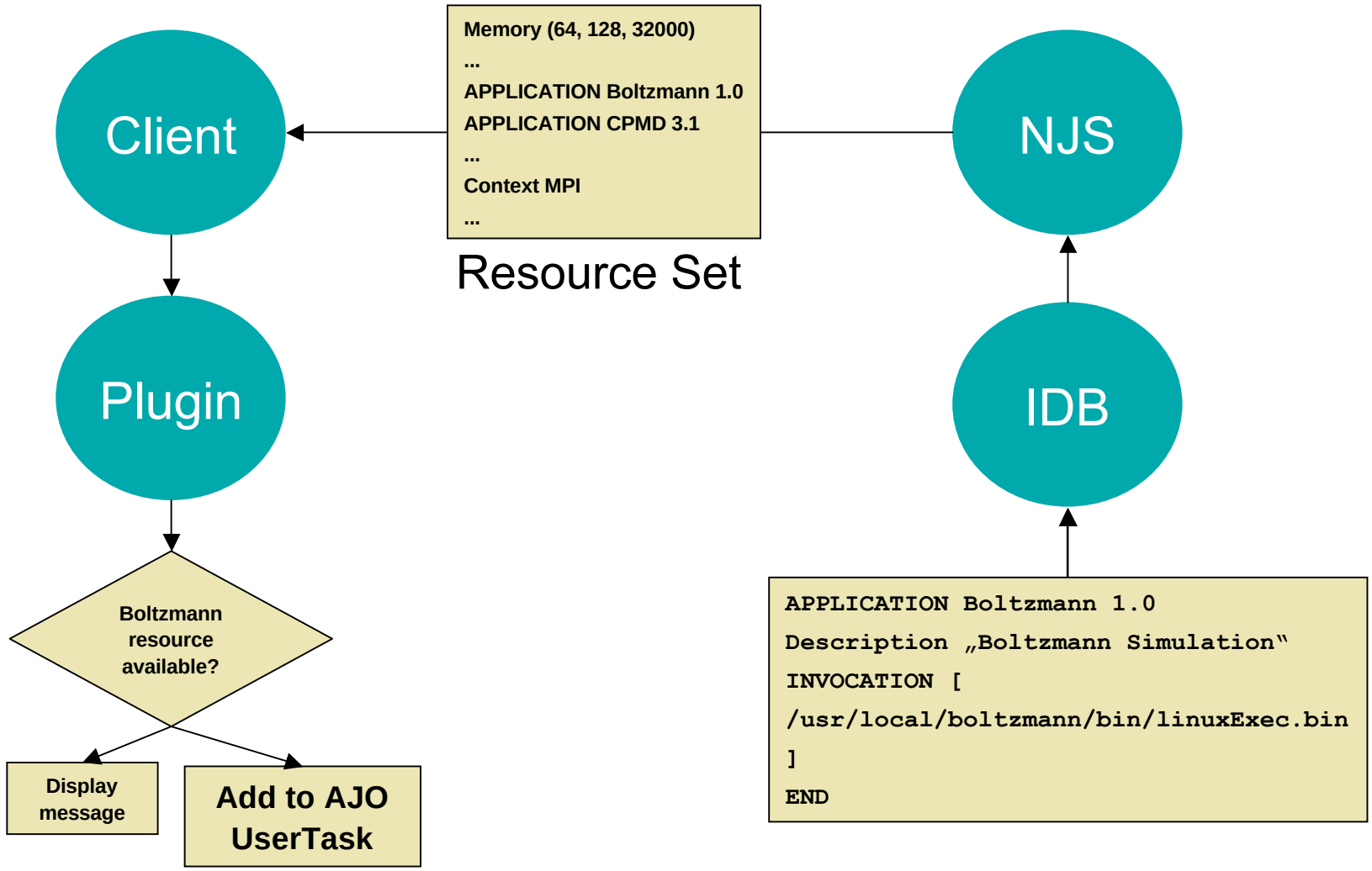
Container Hierarchy



Implementing the Container

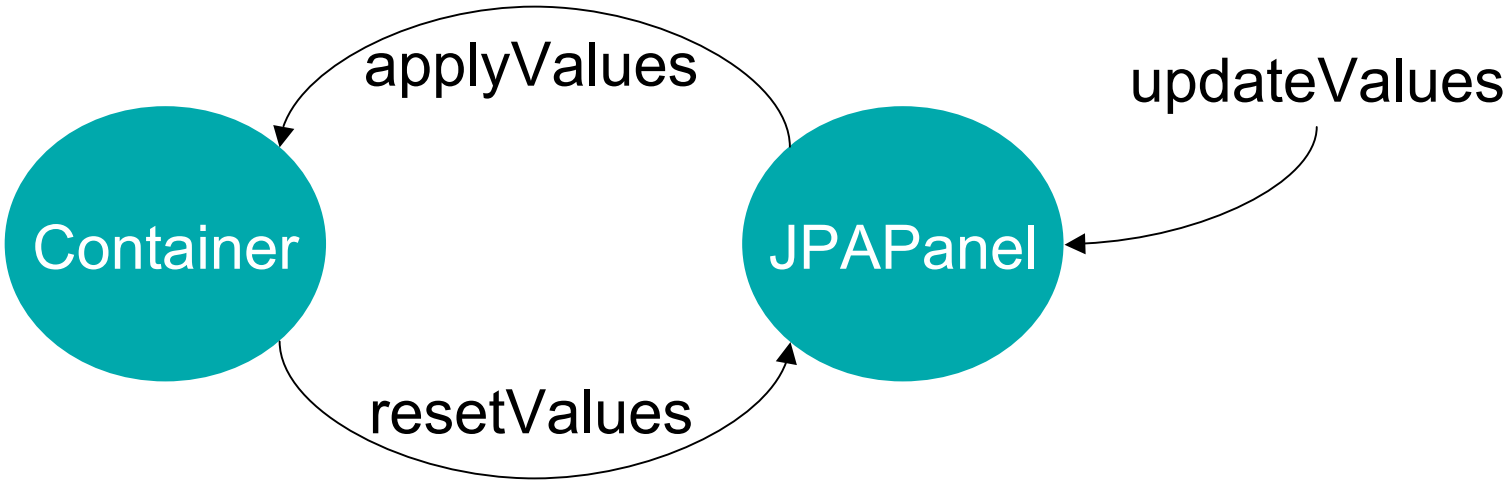


Using Application Resources



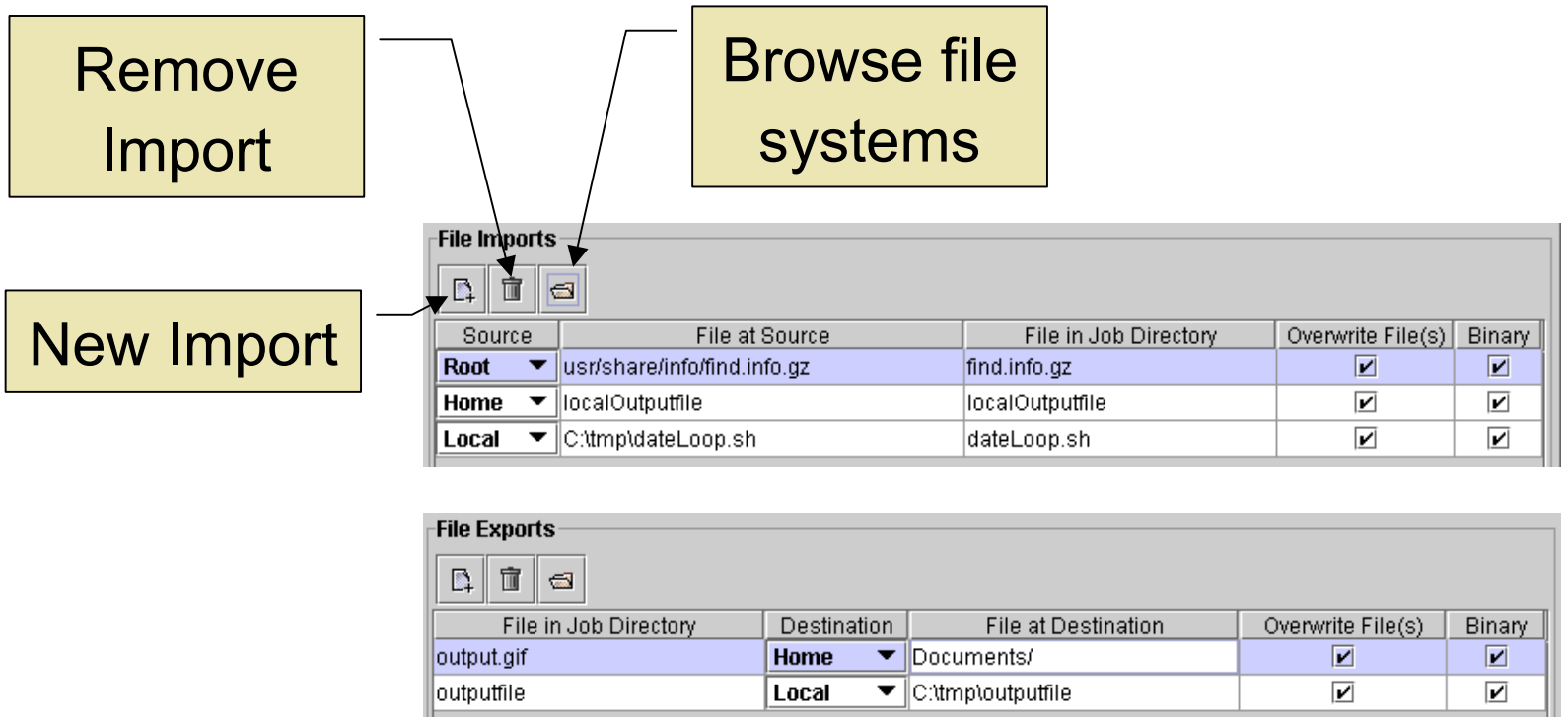
JPAPanel

- Set parameters in container
- Document/View paradigm
- Sub class of javax.swing.JPanel
- Implements interface Applyable
- Follow *Java Look and Feel Design Guidelines*



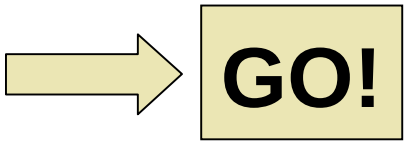
Import and Export Panels

- Specify file imports and exports from the GUI
- Use out of the box



Step 2: Writing the basic Boltzmann Plugin

1. Rebuild Project
2. Build Jar: File->Export->Jar File „lib/boltzmannPlugin.jar“
3. Sign Jar: jarsigner –keystore /home/yourname/.unicorepro/keystore -storetype JCEKS –storepass yourpassword boltzmannPlugin.jar „Your Alias“
4. Run Client, see if Boltzmann task is available
5. Build job, add Boltzmann task, specify imports and exports as in CommandTask



Remote Text Editor

- Load, edit and save files from remote and local file spaces
- Use out of the box

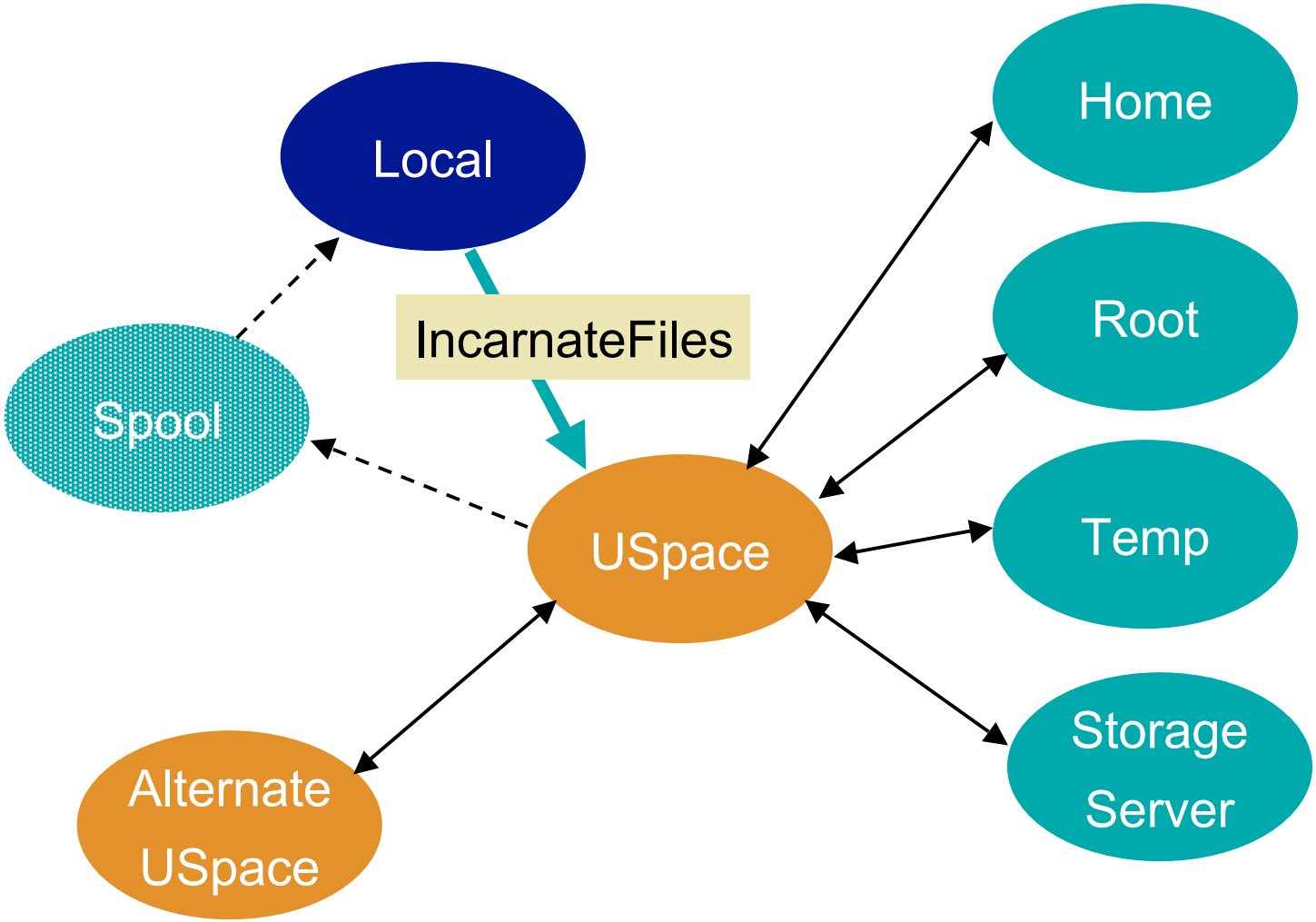
```
public class PluginJPAPanel extends JPAPanel {
  private PluginContainer container;
  private RemoteTextEditor textEditor;

  private buildComponents() {
    textEditor = new RemoteTextEditor();
    JScrollPane editorScrollPane =
      new JScrollPane(textEditor);
  }

  public void applyValues() {
    container.setText(textEditor.getText());
  }

  public void updateValues(boolean vsiteChanged) {
    if(vsiteChanged) {
      textEditor.setVsite(container.getVsite());
    }
  }
}
```

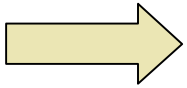
File Transfers in the AJO



Step 3: Transferring input and output files

- Edit the input file with the RemoteTextEditor
- Plugin sends the input file with IncarnateFiles
- User adds output file in export panel

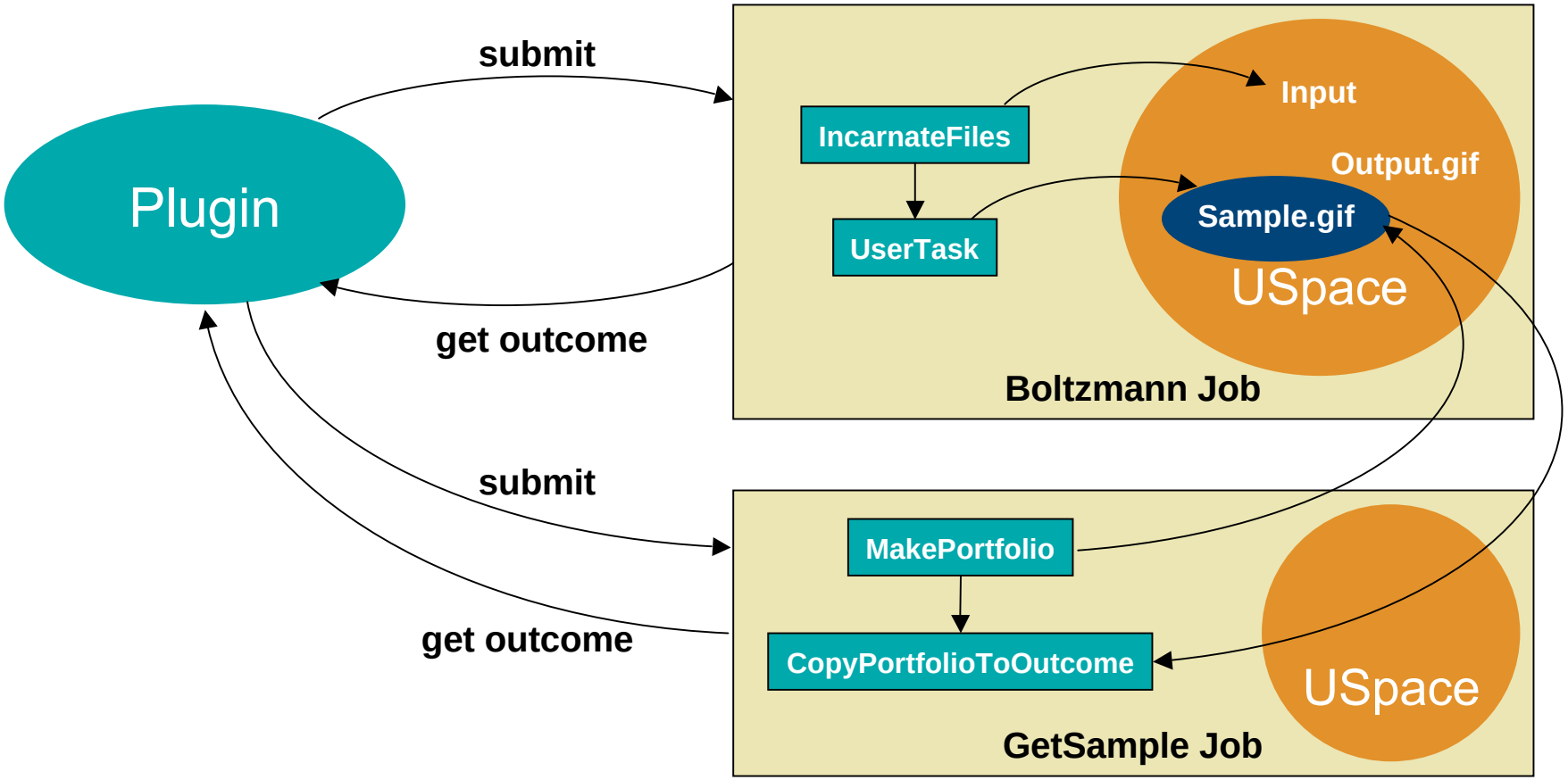
1. Unpack SourceStep3.tar.gz
2. Refresh Project
3. **Add IncarnateFiles task to execution ActionGroup in BoltzmannContainer.java**
4. Run Client



GO!

AJO Portfolios

- Wrap files in Uspace in portfolios
- Pass portfolios between tasks



Additional outcome panels

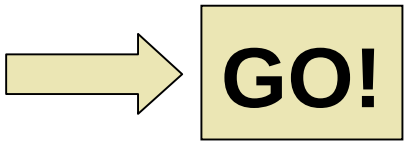
- Make your outcome panel a sub class of JPanel
- Implement interface IPanelProvider in Container
- Implement interface IApplyable in outcome panel

Use `com.pallas.unicore.client.panels.ImagePanel` to display sample.gif

Step 4: Getting intermediate results

- Get sample files with MakePortfolio and CopyPortfolioToOutcome (use GetFilesFromUSpace!)
- Visualize sample files in additional outcome panel

1. Unpack SourceStep4.tar.gz
2. Refresh Project
3. **Add GetFilesFormUspace request**
4. Run Client



Step 5: Free-Style

- Send Control files to running application
  - Add a control panel to outcome area
  - Write a SendFilesToUspace request
- Write a Load Monitor
  - extract CPU load from „top“
  - Use ResourceManager.getUsites()/getVsites()
- Write a Boltzmann Wizard
  - Add GUI elements to specify input parameters
  - Generate input file from GUI entries
- Automatically export output.gif
  - Add a FileExport object in Plugin Code