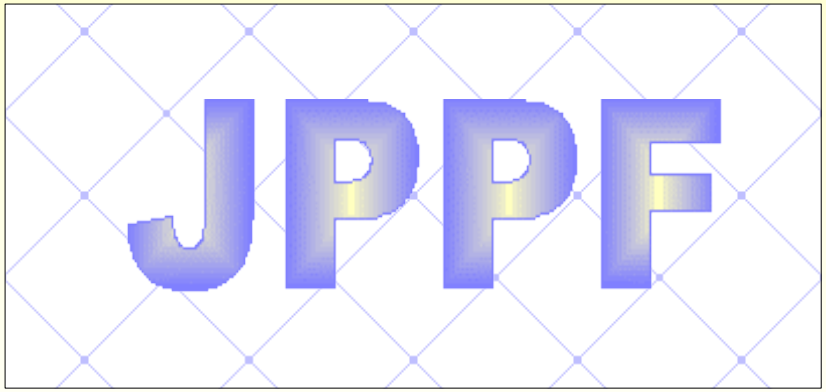
Java Parallel Proccessing Framework



An Open Source alternative to grid computing

by Laurent Cohen

Agenda

- Why JPPF
- Main features
- General architecture
- How to use it with my own Java program
- How to get rid of deployment hassles
- Optimized for performance
- Robustness and scalability
- A word about security
- Administration and monitoring
- And tomorrow...

Why JPPF? 1/2

The origins of the framework

- Tackling AI problems
  - Neural networks
  - Genetic algorithms
- Very resource hungry
- Why not better utilize existing hardware?
- Need for parallelization
- Issues with existing frameworks
  - Heavy
  - Complex to setup and use
  - Not platform independent
  - Not Open-Source

Why JPPF? 2/2

How it came to life

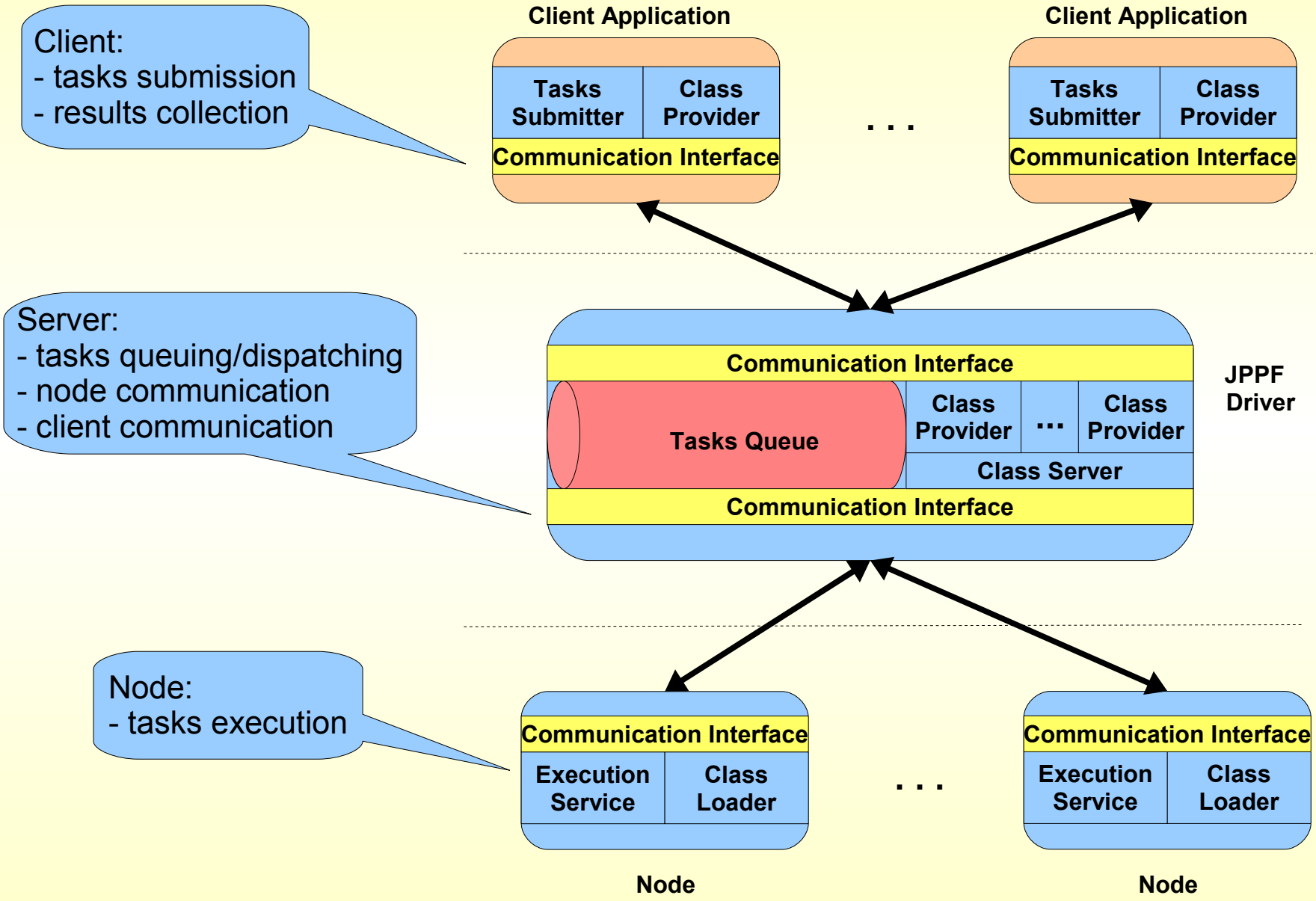
- Excitement for the project
  - parallel computing is a great challenge
  - Open-Source adventure (thanks SF.net)
- Need to parallelize a variety of applications
- Need to work on heterogenous environments
- Need for performance and scalability
- Confidence in software engineering skills
- Need for ubiquitous, platform-independant language
  - Java is a great choice
  - plenty of professional and personal experience

Main Features

What JPPF does

- API to execute independent tasks in parallel
- Scalability to many thousands of nodes
- Platform-independance for nodes and clients
- Execute your code without need for deployment
- Code executed in a secure environment
- Fail-over and recovery
- Graphical monitoring of the framework health
- Remote administration

General Architecture



How to use it in my Java program 1/2

A glimpse of the API: defining an atomic task

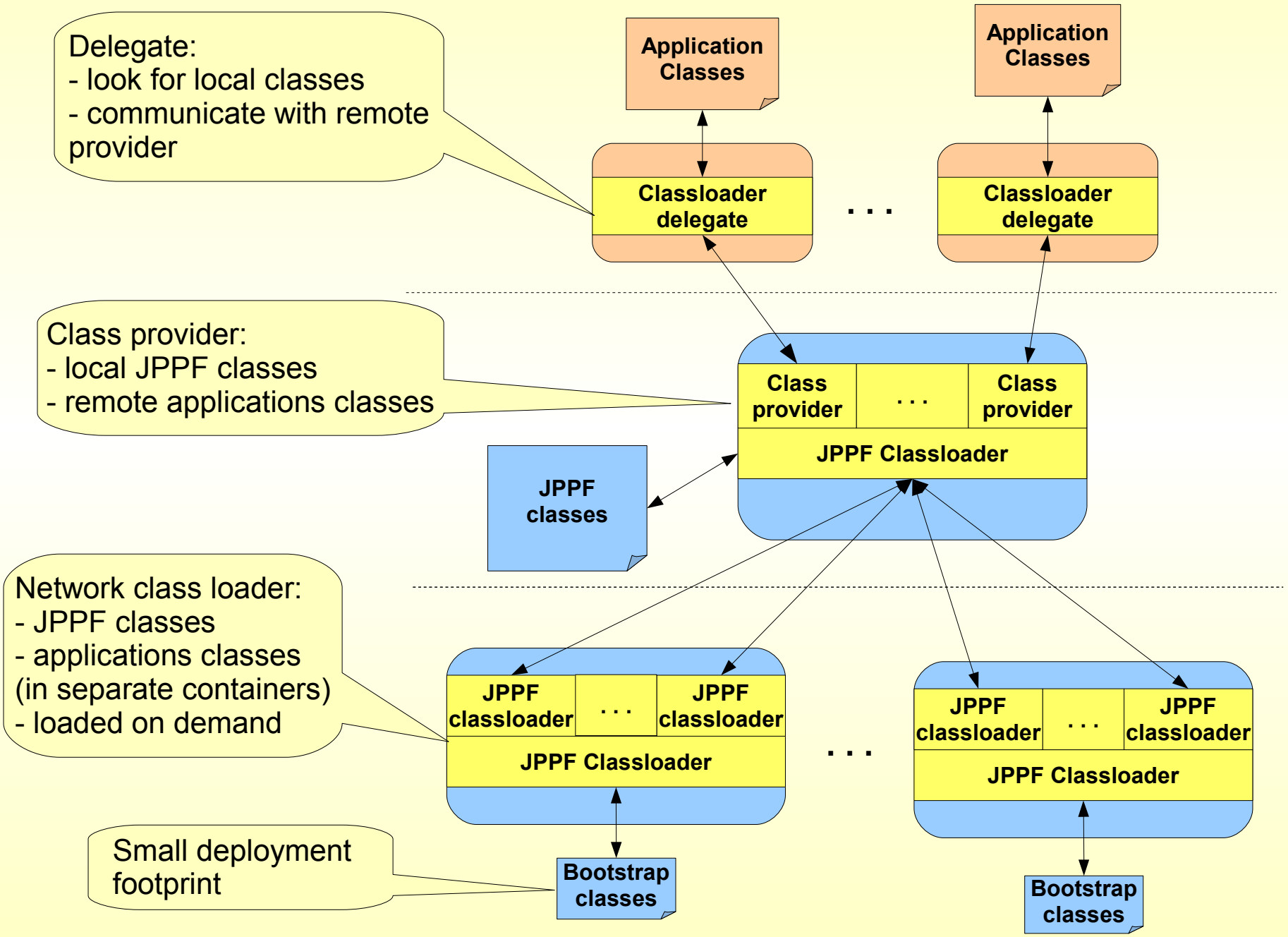
```
/**
 * This class performs my parallelized code.
 * @author Laurent Cohen
 */
public class MyTask extends JPPFTask
{
  /**
   * This is where it happens.
   * @see java.lang.Runnable#run()
   */
  public void run()
  {
    // here it begins
    ... my code ...
    // here it ends
    setResult(theComputationResult);
  }
}
```

How to use it in my Java program 2/2

A glimpse of the API: submitting tasks for execution

```
/**
 * This class submits atomic tasks for execution.
 */
public class TaskSubmitter
{
  /**
   * Submit the tasks and wait for their execution.
   * @throws Exception if any exception is raised.
   */
  public void submit() throws Exception
  {
    JPPFClient jppfClient =  new JPPFClient();
    List<JPPFTask> taskList = new ArrayList<JPPFTask>();
    taskList.add(new MyTask());
    taskList.add(new MyTask());
    jppfClient.submit(taskList, null);
    System.out.println("The result of the first task is: "
      + taskList.get(0).getResult());
  }
}
```

How to get rid of deployment hassles 1/2

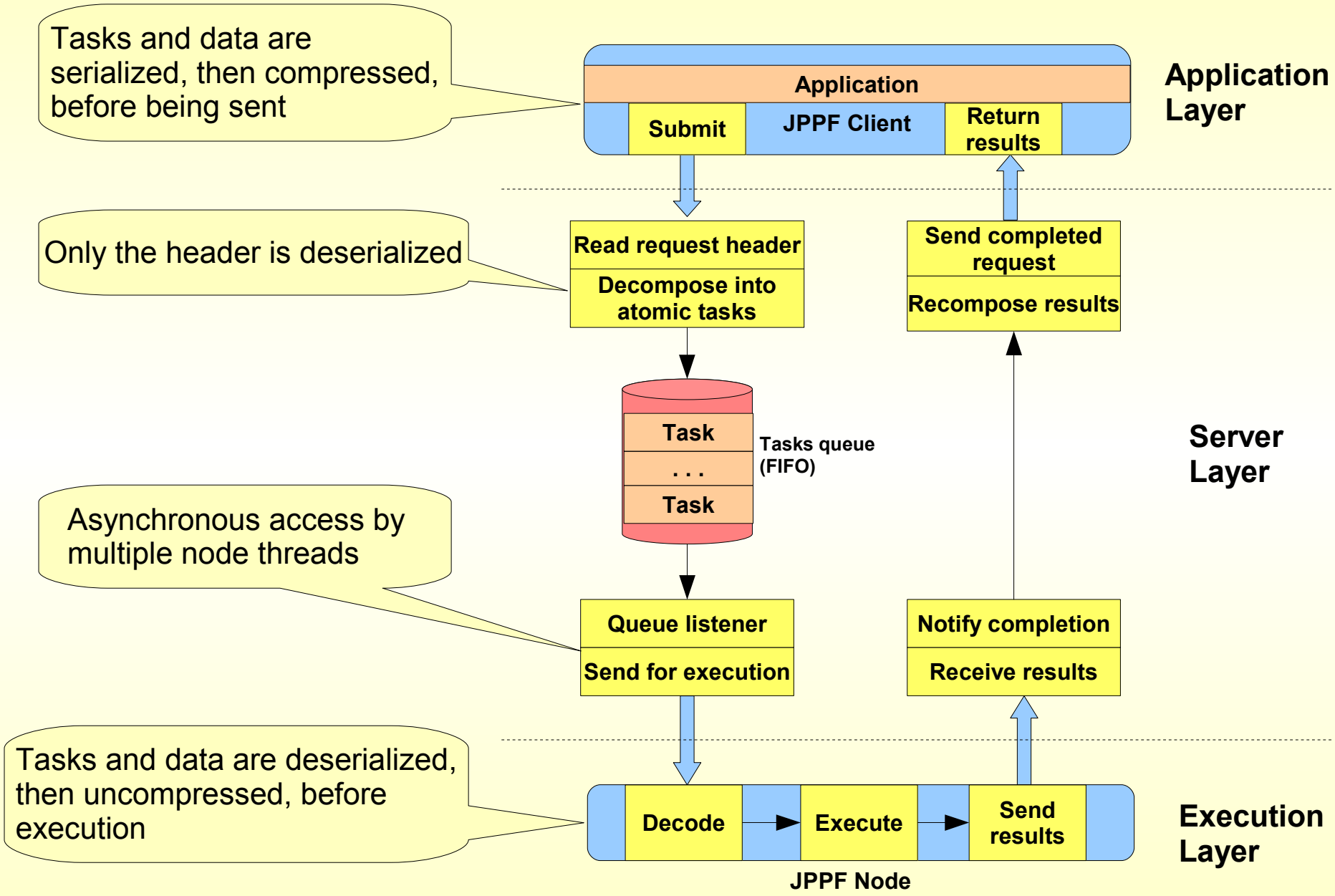


How to get rid of deployment hassles 2/2

How do I benefit from dynamic code loading?

- No need to deploy classes and libraries
  - The API takes care of it
- Code updates automatically accounted for
- Security of separate containers
  - Other applications don't see my code
  - I can't see other applications code
- Framework updates automatically propagated
  - The nodes download their own code

Optimized for performance



Optimized for performance 2/2

How JPPF deals with major performance sinks

- The server is an obvious bottleneck
  - Doesn't need to see the code as such
  - Asynchonous queue removes managemenent overhead
  - Server code profiled to minimize the CPU utilization
- Network transport optimization
  - Compression/decompression of tasks code, data and results on the client/node side
  - Asynchronous communication limits network traffic
  - TCP sockets rather than high-level object broker

Robustness and scalability

What it takes to have a high quality of service

- Failover and recovery
  - Nodes and clients reconnect automatically
  - Clients and server automatically resubmit tasks
  - Configurable to adapt to environment
- Current architecture limits
  - Single server architecture
  - Number of nodes limited by available TCP ports
  - 2 connections per node, so only 30,000 nodes max
- Server hardware a major factor
  - Multi-cpu highly recommended

A word about security

- Nodes can't do everything
  - File system access restrictions
  - Network connection to the JPPF server only
  - Limited access to system environment
  - Prevented from halting the JVM
  - Rights customizable through a policy file
- My code confidentiality is guaranteed
  - Runs in an isolated container
  - The server doesn't see it
- Administration requires authentication
  - No password sent in clear over the network

Administration and monitoring

Keeping control over the framework

- Capability to stop / pause / restart the server
  - Allow maintenance with minimal interruption
  - Handle overload situations
- Execution and server health data collection
  - Latest statistics
  - Variety of real-time charts
  - Fully customizable charting capabilities
- Graphical tool built on top of the client API
  - Acts as a client with special capabilities
  - Easy to plugin different kinds of tools

And tomorrow...

Project vision

- Multi-server architecture
  - Extend the scalability to millions of nodes
- Automate the server behavior
  - Configurable alert conditions
  - Automatic responses to alerts
- Pay per use grids: accounting for the time spent
- JPPF@Home
  - Node as screen saver, applet, Java Web Start app.
  - Keep the Open-Source spirits up!