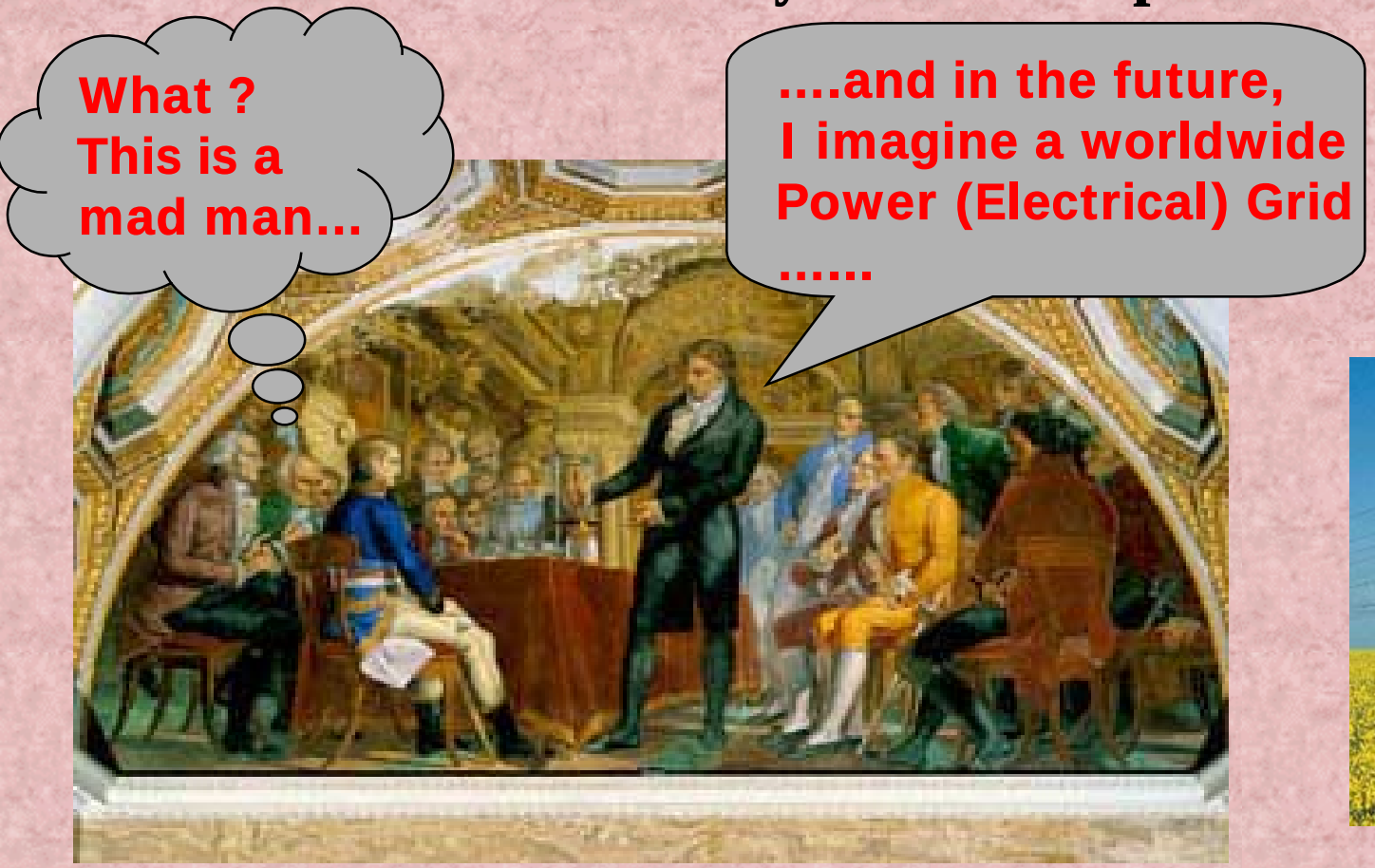
ALiCE – A Java-based Grid Computing System

Assoc Professor Teo Yong Meng
Department of Computer Science
National University of Singapore
teoym@comp.nus.edu.sg
http://www.comp.nus.edu.sg/~teoym

The Electric Revolution

Triennale exhibition palace, Milan

Alessandro Volta in Paris in 1801 inside French National Institute shows the battery while in the presence of Napoleon I



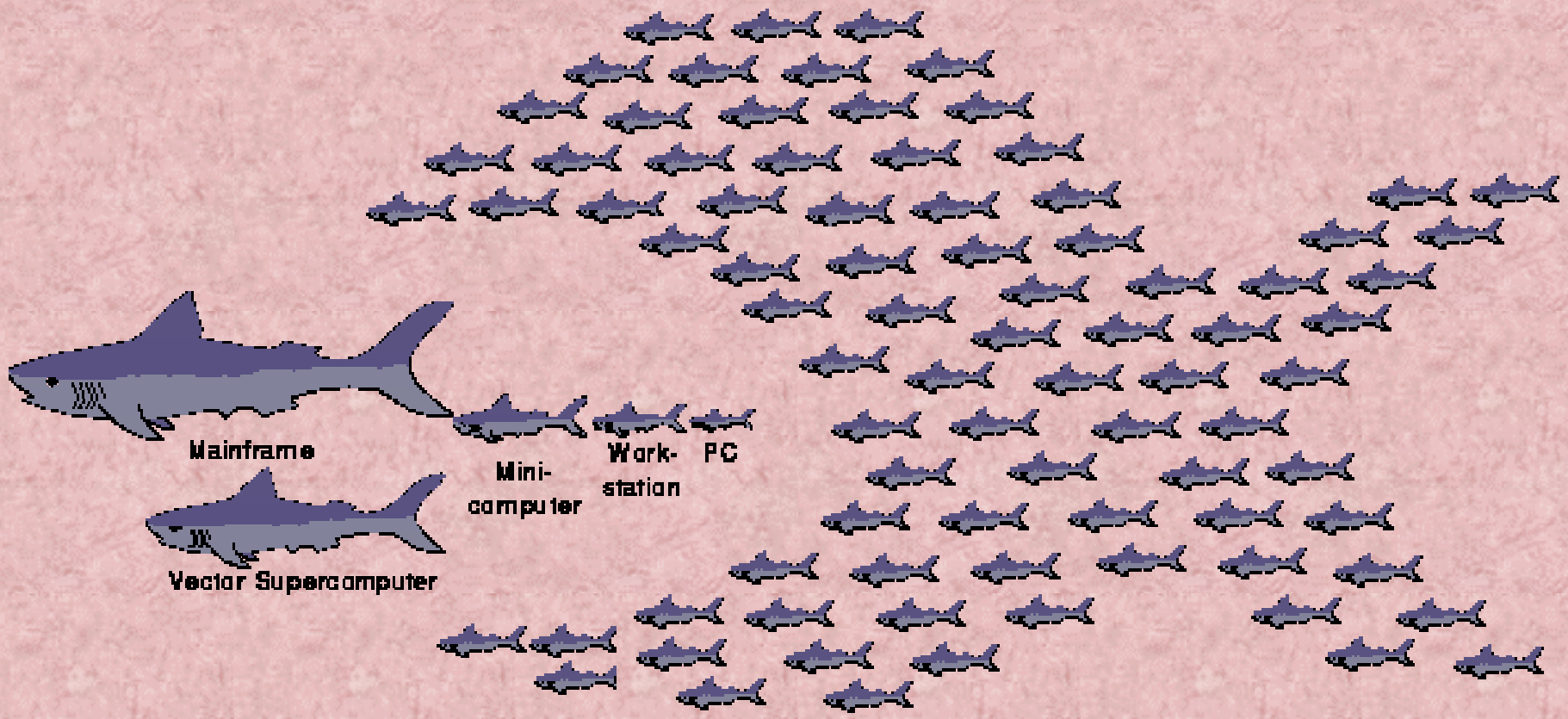
Fresco by N. Cianfanelli (1841)

(Zoological Section "La Specula" of National History Museum of Florence University)

Overview

- **grid computing**
- **potential benefits**
- **ALiCE**
  - main features
  - execution model
  - Implementation
  - programming model
  - some ALiCE examples
- **related works**
  - current status
  - limitations
  - examples

Grid Computing



Current Internet Technology

- Internet was conceived in the late 1960s as a peer-to-peer system to share computing resources.
- Current Internet technologies address communication and information exchange among computers but not the coordinated use of resources at multiple sites for computation.
- The aim of grid computing is to exploit unused latent computing capabilities of idle resources (hardware, software, etc.) for solving large-scale compute intensive problems.

Grid Computing

- high performance computing techniques that seeks to efficiently coordinate the use of geographically dispersed compute resources linked by networks to solve large, complex problems

- a computational grid must integrate heterogeneous compute resources with dynamic availability, connected by an unreliable network.

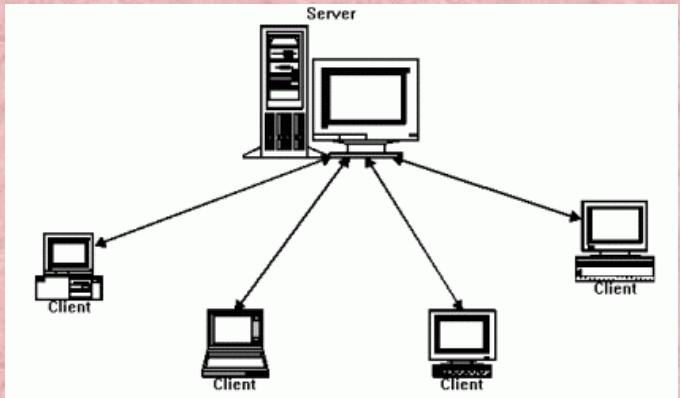
Potential Benefits of Grid Computing

1. sharing but more importantly **aggregation of resources** and **leveraging on resources** you don't own;
2. **reduce business cost** by maximizing the utilization (and return of investment) of existing corporate computing resources through better utilization (and selling of "idle" resources)
3. meet the need for short-term supercomputing power without the need for costly investment - **computational capabilities**;
4. provide **remote access** to expensive computational resources and equipments, and hard to distribute, large and proprietary data sets;
5. computing on demand **reduces business costs**

reduces business costs, capabilities and scalability, fault tolerance, ...

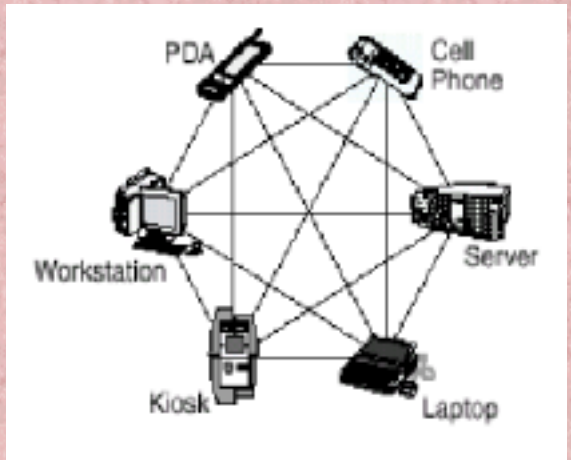
ALiCE Grid Computing

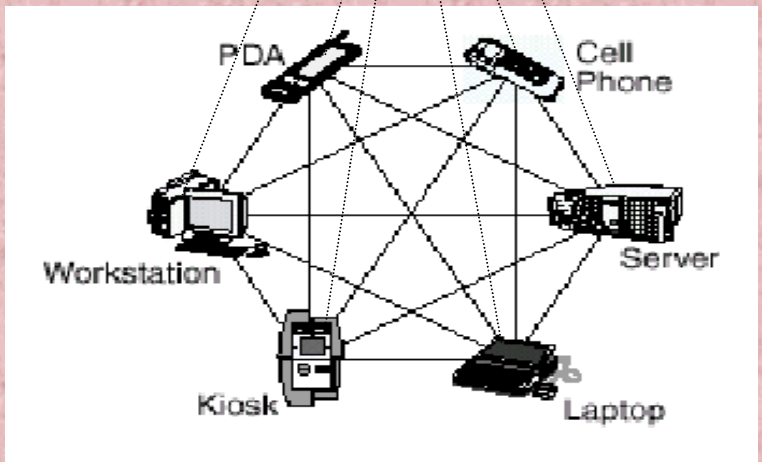
ALiCE (Adaptive and scaLable Internet-based Computing Engine)



Client/Server Model

ALiCE
Brokered Grid Model





Grid Model

Main Objectives

- to support computing on demand and resource aggregation (resource sharing)

- to support the development and execution of generic grid applications, i.e., run applications over a geographically distributed, heterogeneous collection of resources (computers, networks, etc.)

- a middleware that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware and operating systems

Main Features

- job-parallelism to maximize throughput
  and (Java) object-parallelism to maximize performance

- **distributed load-balancing algorithm**
  - **application-driven** distributed load balancing;
  - task replications for fault-tolerant and to meet performance deadline

- differentiated levels of security (code, data and result) at varying costs

- ALiCE is implemented using ubiquitous and easy-to-use Java, Java Jini/JavaSpaces technologies

- adaptive parallelism, cross-platform portability, extensibility and scalability

ALiCE Producer-Consumer Model

- **consumers (clients)**
  - execute jobs for solving problem of varying size and complexity
  - benefit by selecting and aggregating resources wisely
  - tradeoff timeframe vs cost – minimize IT expenses
- **producers (servers)**
  - contribute "idle" resource for executing consumer jobs
  - benefit by maximizing resource utilization
  - tradeoff local requirements vs market opportunity – maximize return on IT investment
- **resource broker (agency)**
  - facilitates and automate the process of resource sharing
  - coordinates the allocation of producer resources to match consumer required performance requirement and cost
  - regulates demand-and-supply

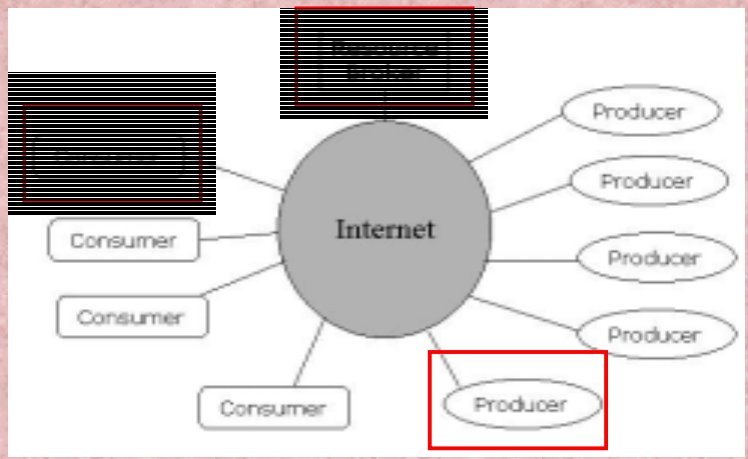
ALiCE Architecture (1)

Consumers (C)

- interface to users
- launch point for applications
- collection point for results (visualization)

Resource Broker (RB)

- authentication
- application execution control
- resource management
  - scheduling
  - load balancing
- ...



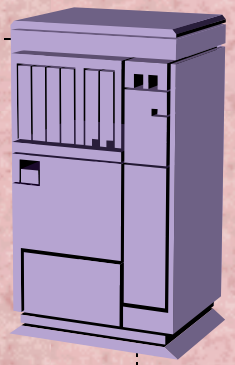
Producers (P)

- provide computing power
- executes tasks

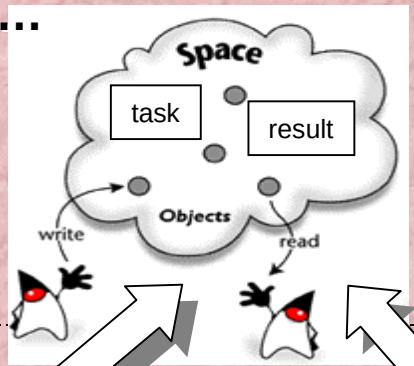
ALiCE Architecture (2)

Resource Broker

Scheduler
Accounting
Performance Monitor
Security Manager
Pricing Manager
……



Task Pool



Producer

Registration

WorkEngine

Local Scheduler

Performance Measurement

……

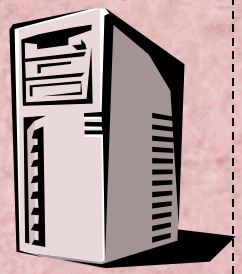
Registration

Task Submission

Result Retrieval

…….

Consumer

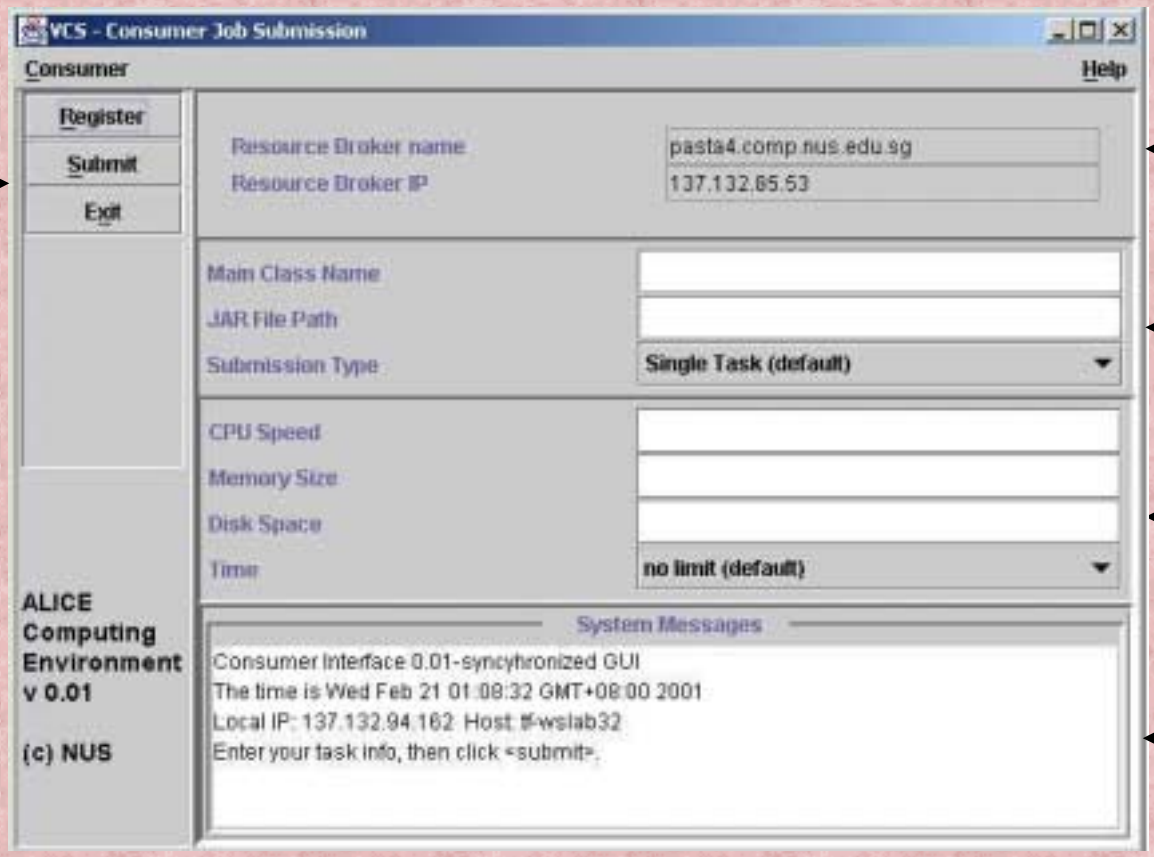


Types of Applications Supported

1. **Sequential Jobs (parametric computation)**
   - supports single-tasking programs with well-defined methods like main() or run()

2. **Parallel Jobs - Object-level Parallelism**
   - supports various parallel programming models via programming templates
   - allows task and result objects to be exchanged between consumers and producers through resource broker

ALiCE Consumer GUI



Control Panel

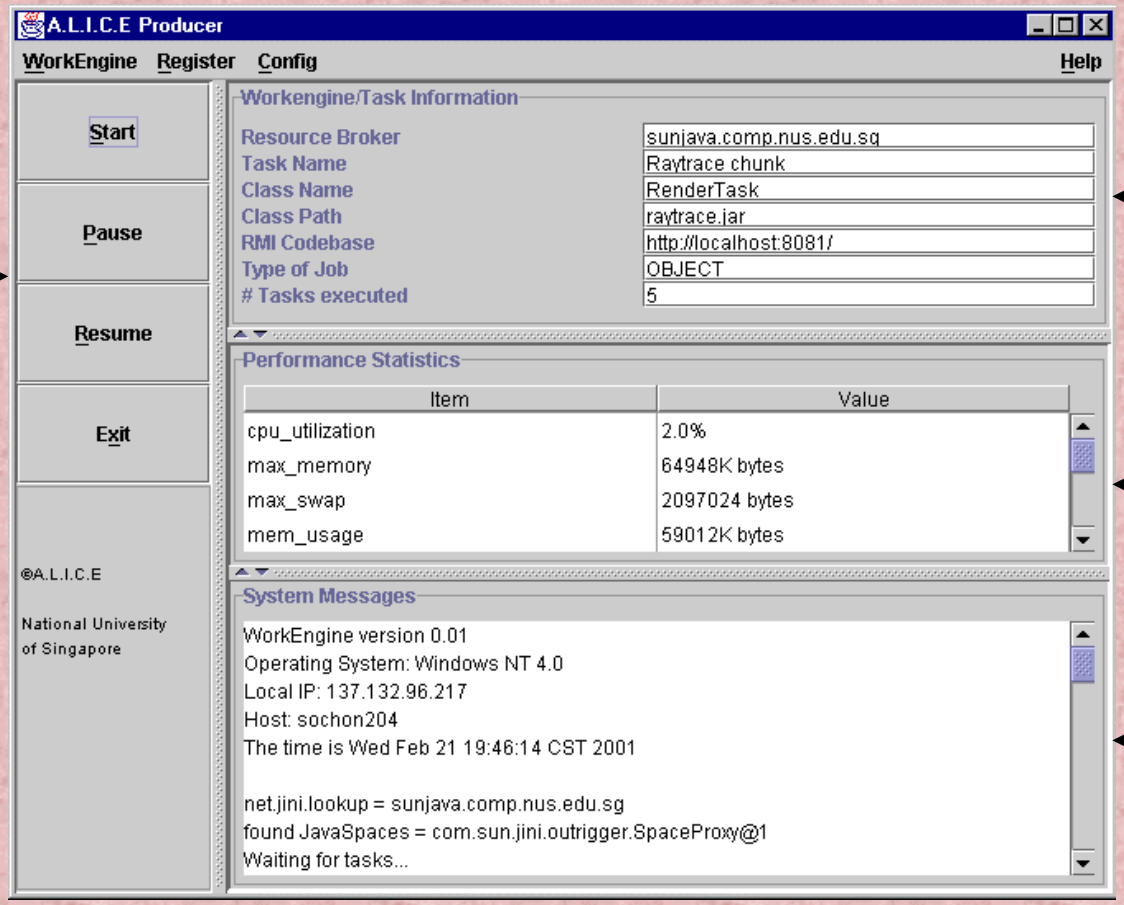
Resource Broker Info

Task Input

User Requirements

Messages

ALiCE Producer GUI



Control Panel

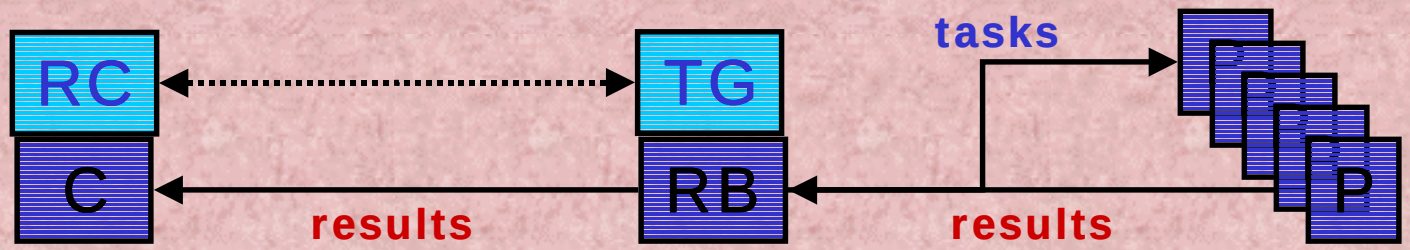
Task Information

Performance

Messages

ALiCE Programming Models

- Master-slave, shared-memory, data parallel, etc. through a library of programming templates
- transparency & programmability – hides JavaSpaces
- ***TaskGenerator-ResultCollector*** model:
  - Task Generator (TG)
  - Result Collector (RC)



Programming Templates

ALiCE programming template includes the following specifications:

1. Task Generator, Result Collector, and Task Classes

2. Method used to send Task objects to ALiCE

3. Result object used to store result information

4. Method used to retrieve the Result objects from ALiCE.

Task Generator Template

```
// Task Generator Template
import alice.application.*; // import the templates

public class CLASSNAME extends TaskGenerator {

// place your variables here


// Constructor
 public CLASSNAME(){};

 // The no parameter constructor is a MUST.

public void init() {
 // place your initialization code here
}// init()

/**
   * generateTasks() - generates tasks
   *
   **/
 public void generateTasks() {
```

```
// This is where the tasks are generated
// Usually tasks are generated in a loop,
// and in this loop each task is sent for
// processing by calling the
//
// "public void process(Task t)" method

 } // generateTasks()


  /**
    * main method
    **/
  public static void main(String args[]) {
     CLASSNAME m = new CLASSNAME();
     m.init();
     m.generateTasks();
  }

} // end class
```

Result Collector Template

```
// Template for ResultCollector
//
import alice.application.*; // import the
templates

public class CLASSNAME extends
ResultCollector {

     // place your variables here
     //

    public static void main(String args[])
{

      CLASSNAME MV = new CLASSNAME();
      MV.init();
      MV.collectAllResults();
    }

 // the no argument constructor MUST exist
    public CLASSNAME() {
    }

    public void init() {
      // place your init codes here
      //
    }
```

```
public void collectAllResults() {


 // Here is the result handling code.
 // Usually result handling involves a loop
 // that repeatedly calls the collectResult()
 // method of the ResultCollector superclass.
 //
 // This method returns a Result Object.
 //
 // The contents of this Result Object can be
 // inspected for result handling/processing.



    }
}
```

Task Template

```
// Template for Task

import alice.application.*; // import the templates

public class CLASSNAME implements Task {

    // place your variables here
    //
    public CLASSNAME() {}
     public Result execute() {
          // This is where you do your calculation
       // The results are stored in the Result class
       // which functions as a datastructure
       // with which you can store results of any Object type
     }

public String toString() {
    // returns a String that can be used to ID your task
}
}
```

ALiCE Implementation (1)

Issues:

- How to send user application from Consumer to RB?
- How to initiate the user application at RB with tasks executing at Producers?
- How to distribute code and data?
- How to collect results?
- How to coordinate communication between consumers, producers and RB?

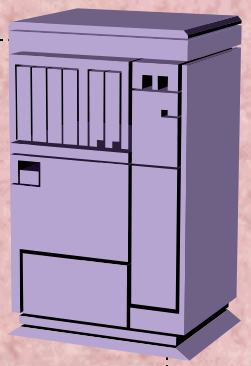
Core Mechanisms

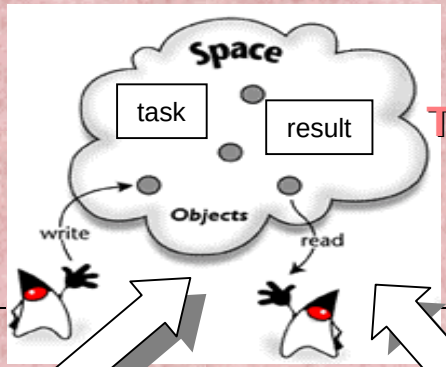
- JavaSpaces
- Reflection API & RMI ClassLoader
- Job Submission
- Job Execution
- Result Collection & Visualization

ALiCE IMPLEMENTATION (2)

Resource Broker

- Java SDK 1.3
- Java Jini 1.1/JavaSpaces
- RMI





Task Pool

- Java SDK 1.3
- Java Jini 1.1/
- JavaSpaces
- RMI
- Swing

Consumer

Producer

- Java SDK 1.3
- Java Jini 1.1/ JavaSpaces
- Java Reflection API
- RMI
- Swing

JavaSpaces

- **JavaSpaces**
  - a tuple-space for Java Objects
  - a communication medium for distributed Java applications
- **JavaSpaces Server** (resides at RB) is the communication medium between Consumers, Producers and RB
- However, JavaSpaces is <u>not designed</u> for dynamic code distribution and execution.

Reflection API & RMI ClassLoader

- **Java reflection API**
  - Represents Classes, Objects in the Java Virtual Machine
  - Allows invocation of methods whose name and Class are known only at runtime.
- **RMI Classloader**
  - A mechanism that provides the dynamic class loading of the RMI runtime system
  - Allows the transport and loading of classes from remote hosts.

We augment JavaSpaceS with these two mechanisms to provide a solution for dynamic code distribution and execution.

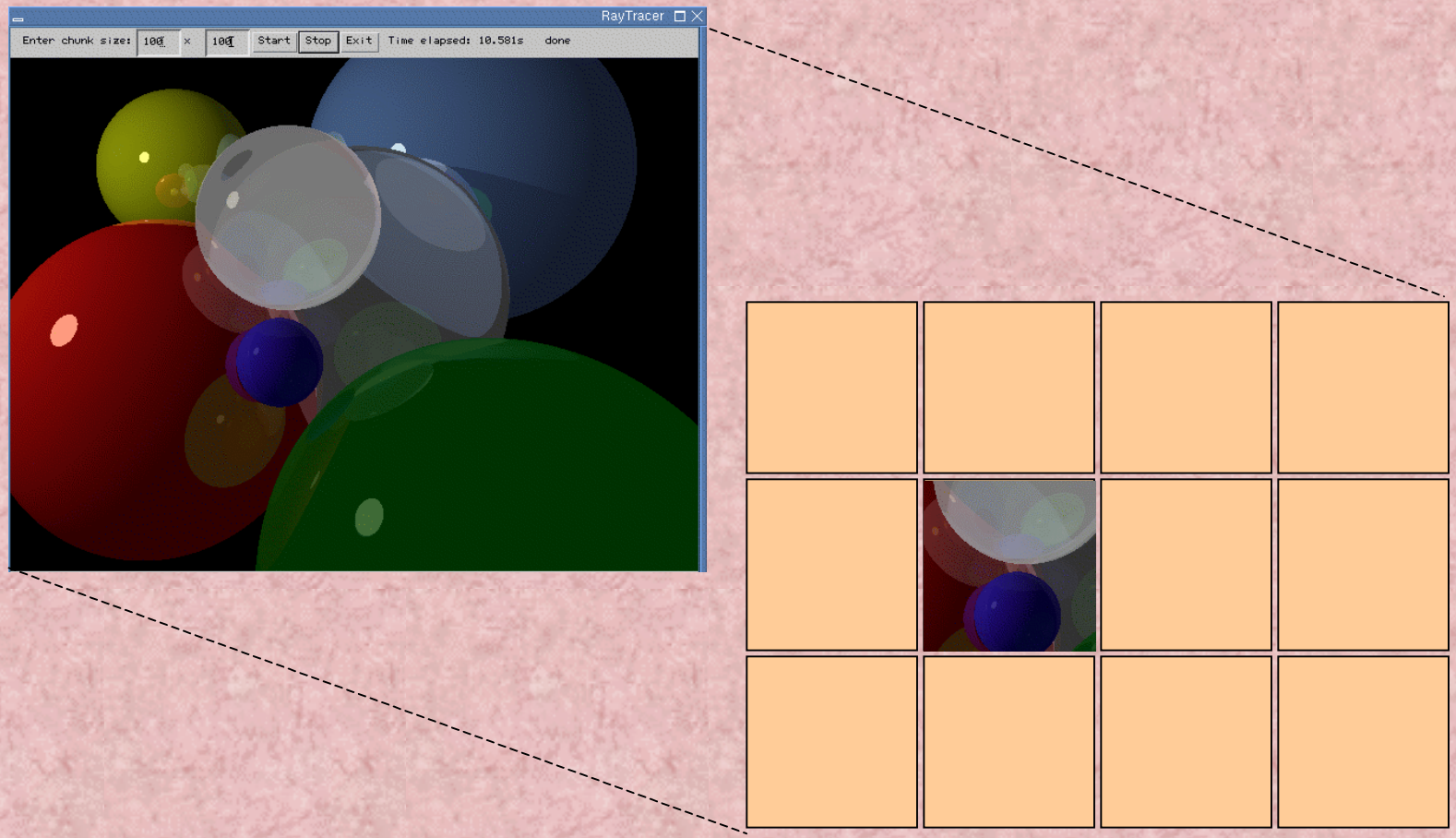
Some Applications-Drivers

1. **Distributed HPC (Supercomputing)**
   - computational science – a massively, scalable, distributed, virtual supercomputer
2. **High-throughput Computing**
   - large-scale simulation/chip design/complex financial analysis & parameter studies
3. **Data-intensive Computing**
   - data mining, drug design, particle physics (CERN),….

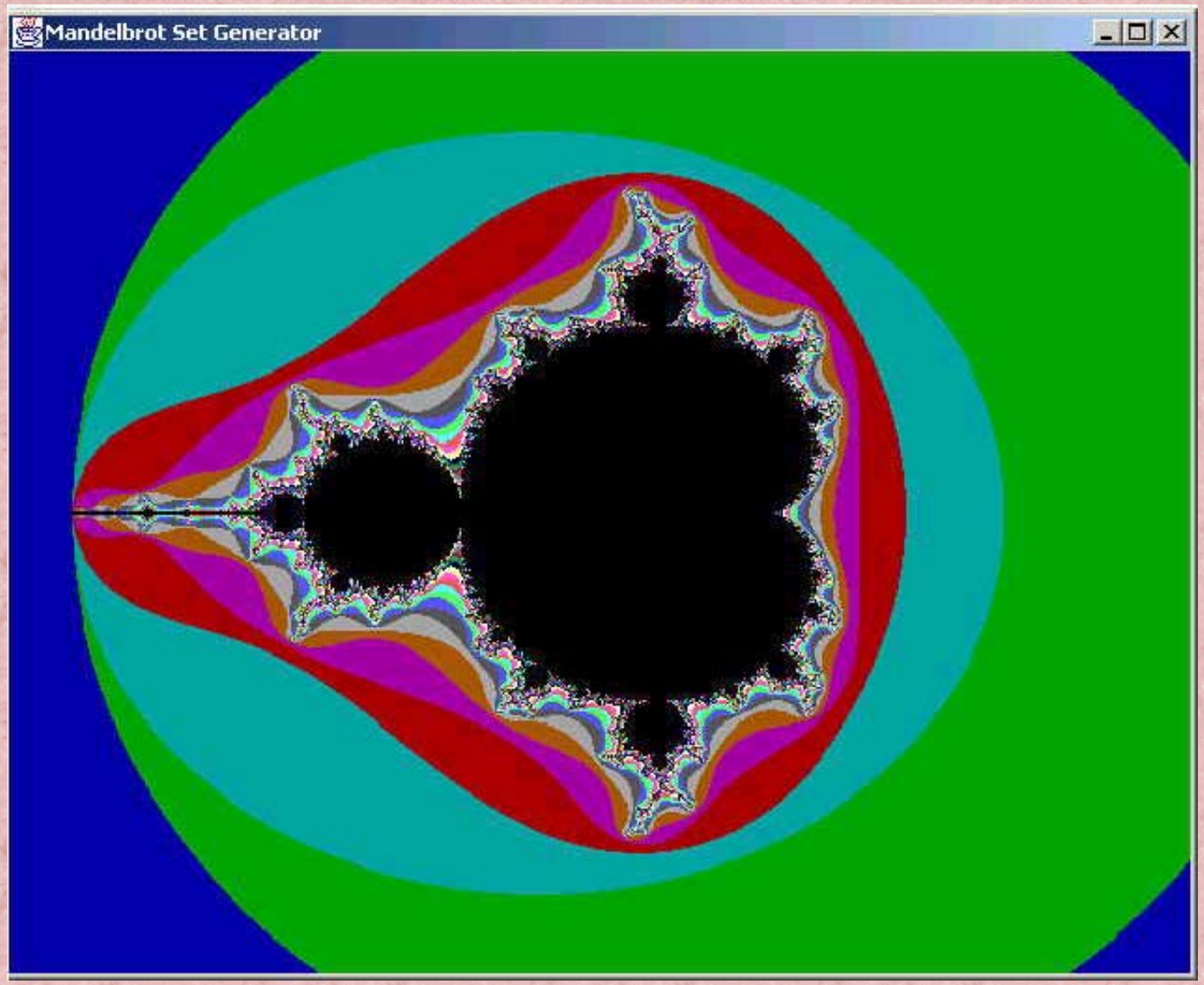
Some ALiCE Applications

- ray tracing
- Mandelbrot set
- georectification of satellite images

ALiCE -Distributed Ray Tracing



ALiCE - Distributed Mandelbrot Set



Tropical Red Tide Monitoring

The Seastar satellite which carries the SeaWiFS sensor

L0 Data
Raw data from SeaWiFS sensor.

Add radiometric and geometric calibration coefficients and computed georeferencing parameters.

L1A Data
Reconstructed, unprocessed instrument data at full-resolution.

Georectification

L1A Georectified Data
Earth geometry corrected data.

Apply sensor calibration data.

L1B Data
Calibration corrected radiance data.

Georectification

L1B Georectified Data
Earth geometry corrected data.

Calculate geophysical values for radiance data.

L2 Data
Geophysical data.

Georectification

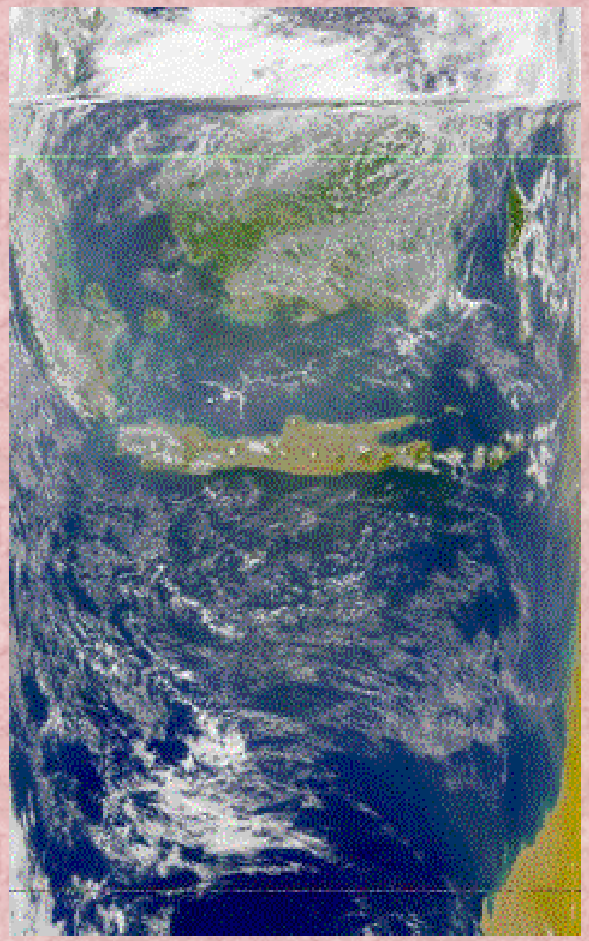
L2 Georectified Data
Earth geometry corrected data.

Map geophysical data into separate “bins” on a space-time grid.

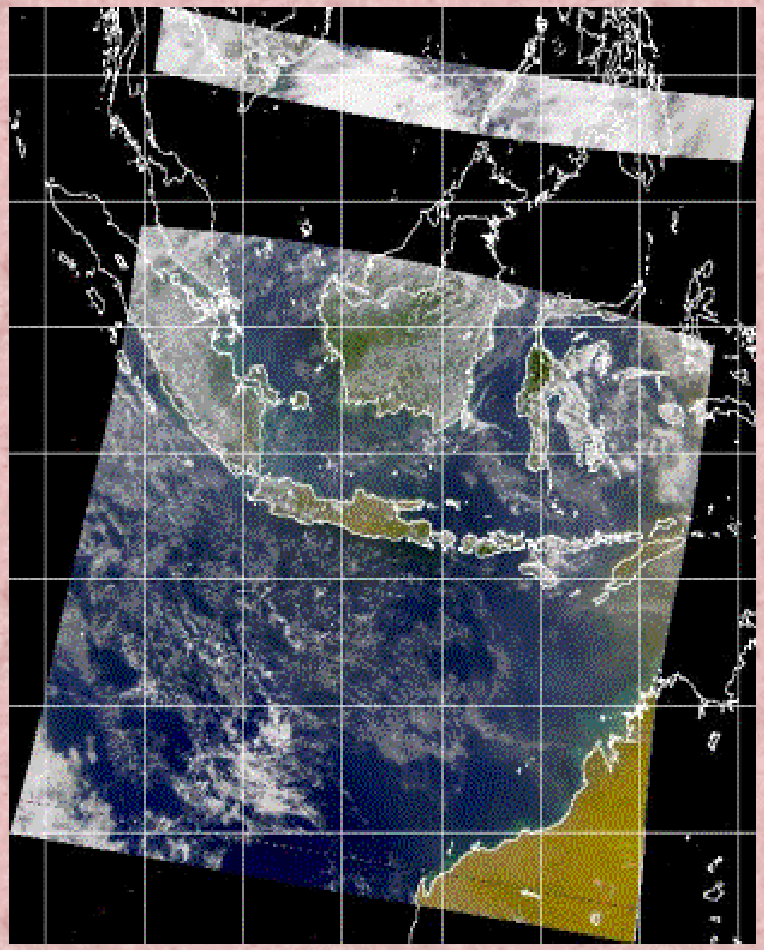
L3 Data
Space-time grid mapped geophysical data.

Scientific or geographical usage of data

Georectification of Satellite Images

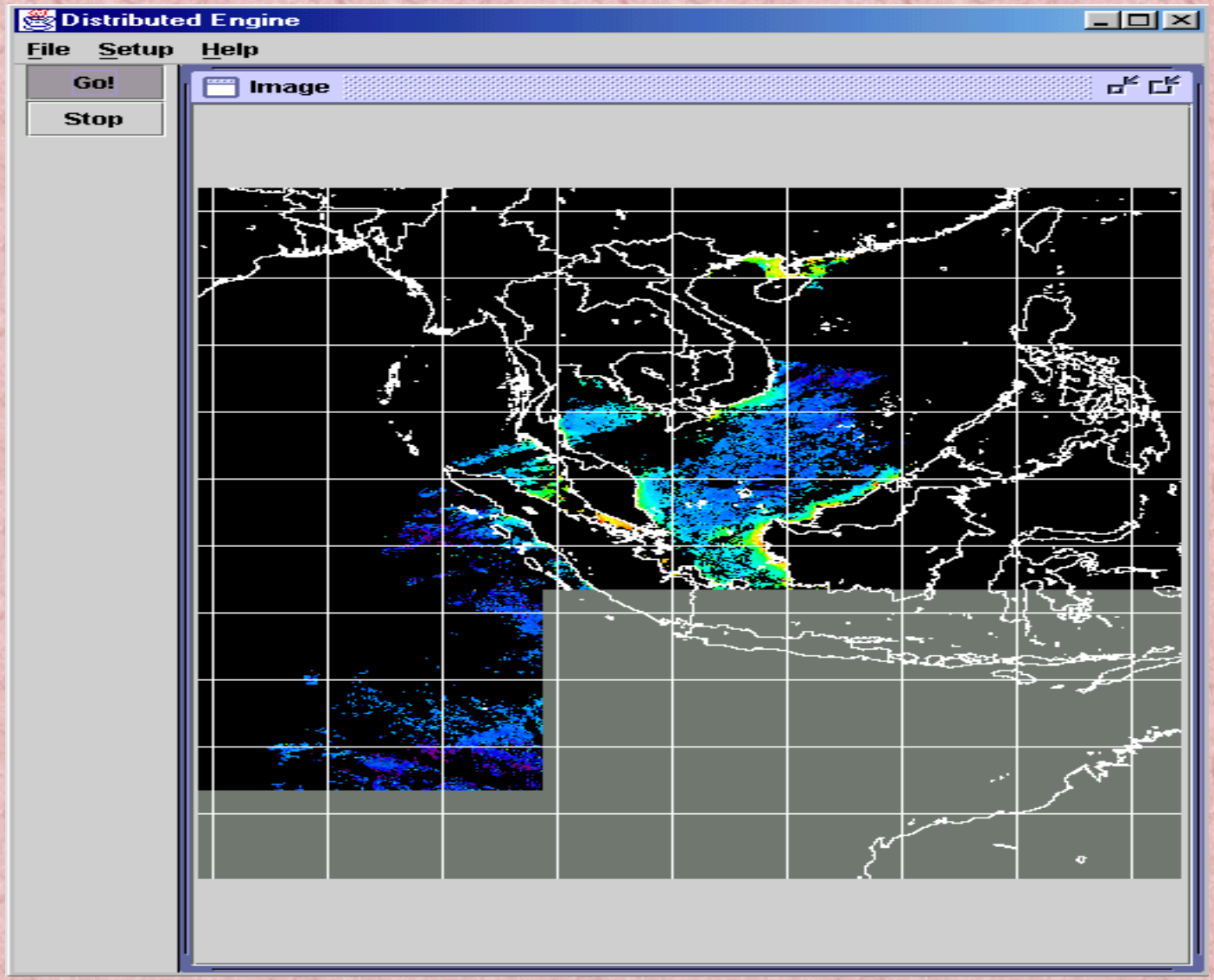


Before
Georectification



After
Georectification

ALiCE - Georectification of Images



Related Works

- current status
- limitations
- examples

Related Works

Grid System	Comm. Technology	Language Supported	Platforms	Architecture
Legion - Virginia	Sockets	C++, Mentat, Fortran, PVM, MPI	Solaris (SPARC), SGI, linux (x86, alpha, dec), HPUX, RS6000	*Tree Structure* (global root node)
GLOBE - Vrije	Java IDL	Java, C	Win NT, Unix	*Tree Structure* (global root node)
Javelin - UCSB	Java Applets, Java RMI	Java	Any Java enabled browser	*Three-Tier* Clients, Brokers, Hosts
Calpyso – NYU/Arizona	Sockets	C++	Solaris, Linux, Win NT	*Three-Tier* Clients, Brokers, Hosts
Charlotte - NYU	Java Applets, Java RMI	Java	Any Java enabled browser	*Three-Tier* Clients, Brokers, Hosts
Knitting Factory	Java Applets, Java RMI	Java	Any Java enabled browser	*Three-Tier* Clients, Brokers, Hosts
Condor - Wisconsin	RPC	C	Win NT, Unix	*Client-Server*
Nile - Cornell	Java, CORBA	Java	Any Java compatible platform	unknown
Bayanihan - MIT	HORB (Java RMI equivalent) using Java Applets	Java	Any Java enabled browser	*Client-Server* Clients are resources only, they cannot submit jobs
ALiCE	***JavaSpaces*** Java RMI	Java	Any Java compatible platform	*Three-Tier* Clients can be a resource provider, a resource consumer, and both

Current Status

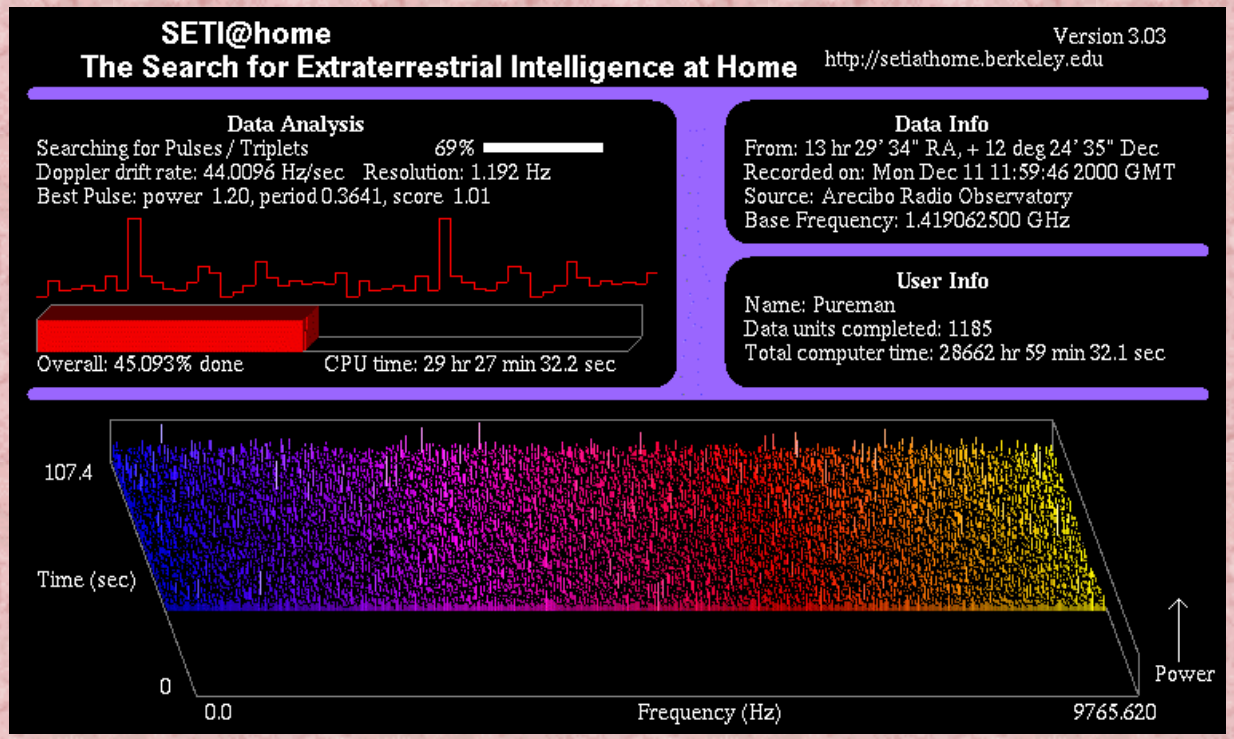
- who contribute resources?
  - Volunteers
- why?
  - Motivated by the public good, fun, challenge, fame, charismatic applications, etc.
- public good projects such as SETI@HOME,, genome@home, folding@home, etc..

Some Limitations

- specialized applications and clients
- cannot process generic data or handle generic computational tasks
- clients must be upgraded to handle new tasks

We need a ubiquitous, portable, infrastructure that can handle *generic* computational tasks and data.

SETI@Home

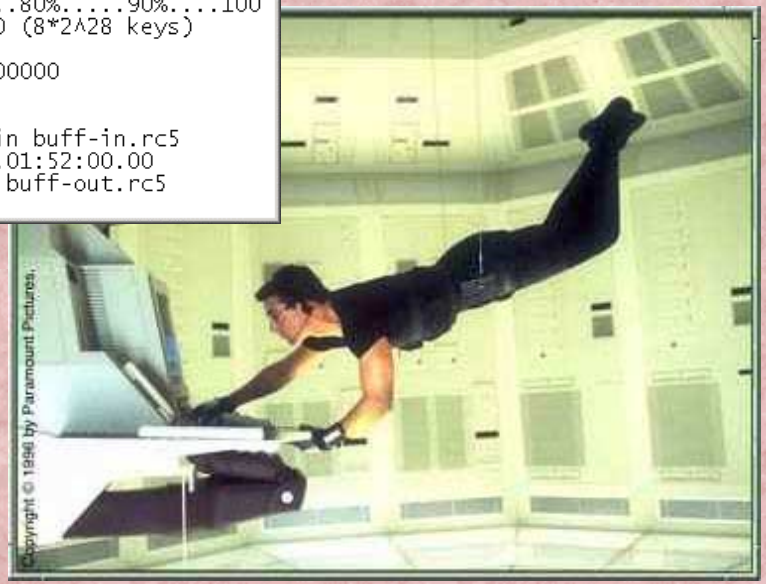

SETI@home
The Search for Extraterrestrial Intelligence at Home
Version 3.03
http://setiathome.berkeley.edu
Data Analysis
Searching for Pulses / Triplets 69%
Doppler drift rate: 44.0096 Hz/sec Resolution: 1.192 Hz
Best Pulse: power 1.20, period 0.3641, score 1.01
Overall: 45.093% done
CPU time: 29 hr 27 min 32.2 sec
Data Info
From: 13 hr 29' 34" RA, + 12 deg 24' 35" Dec
Recorded on: Mon Dec 11 11:59:46 2000 GMT
Source: Arecibo Radio Observatory
Base Frequency: 1.419062500 GHz
User Info
Name: Pureman
Data units completed: 1185
Total computer time: 28662 hr 59 min 32.1 sec
107.4
Time (sec)
0
0.0
Frequency (Hz)
9765.620
Power

Distributed.net (code-breaking)

distributed.net client

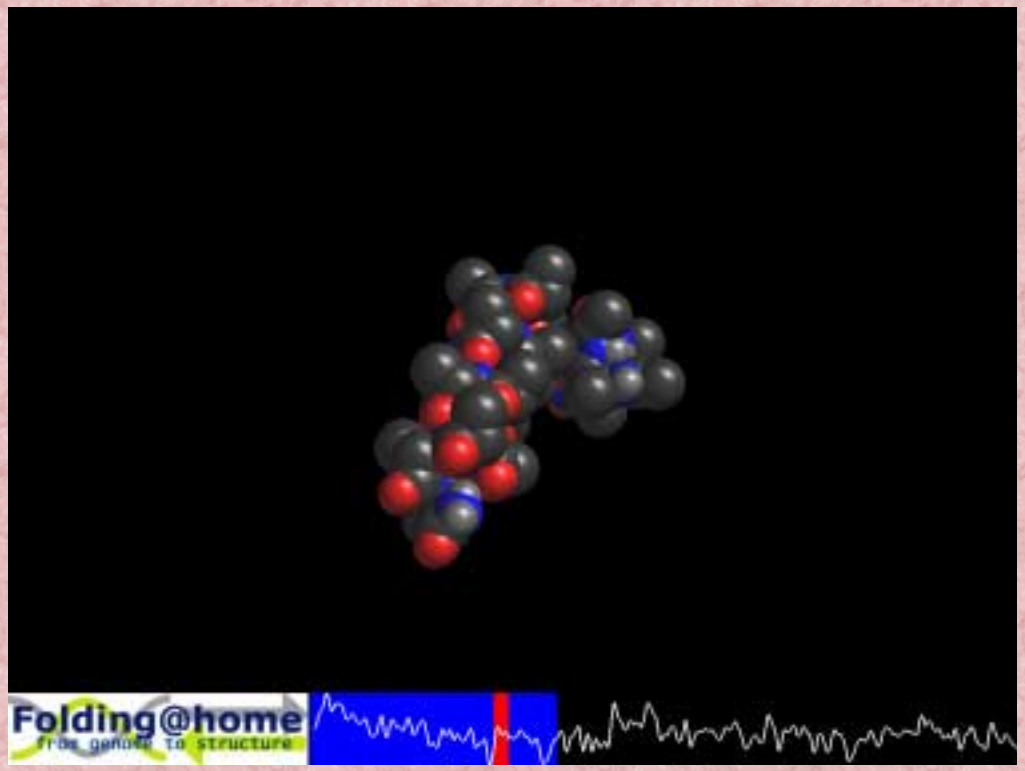
```
dnetc v2.8010-463-CTR-00071213 for Win32 (WindowsNT 5.0).
Please provide the *entire* version descriptor when submitting bug reports.
The distributed.net bug report pages are at http://www.distributed.net/bugs/
Using email address (distributed.net ID) 'jarretfalkner@att.net'

[Dec 11 03:22:22 UTC] Automatic processor detection found 1 processor.
[Dec 11 03:22:22 UTC] Automatic processor type detection found
                      an Intel Pentium II processor.
[Dec 11 03:22:22 UTC] RC5: using core #2 (RG class 6).
[Dec 11 03:22:22 UTC] Loaded RC5 8*2^28 packet F38CC719:70000000 (97.80% done)
[Dec 11 03:22:22 UTC] 4 RC5 packets (32 work units) remain in buff-in.rc5
[Dec 11 03:22:22 UTC] 10 RC5 packets (70 work units) are in buff-out.rc5
[Dec 11 03:22:22 UTC] 1 cruncher has been started.
.....10%.....20%.....30%.....40%.....50%.....60%.....70%.....80%.....90%....100
[Dec 11 03:23:10 UTC] Completed RC5 packet F38CC719:70000000 (8*2^28 keys)
                      0.00:00:48.17 - [957,920.36 keys/sec]
[Dec 11 03:23:10 UTC] Loaded RC5 8*2^28 packet F38CC7B4:B0000000
[Dec 11 03:23:10 UTC] Summary: 1 RC5 packet (8*2^28 keys)
                      0.00:00:48.17 - [957.92 kkeys/s]
[Dec 11 03:23:10 UTC] 3 RC5 packets (24 work units) remain in buff-in.rc5
[Dec 11 03:23:10 UTC] Projected ideal time to completion: 0.01:52:00.00
[Dec 11 03:23:10 UTC] 11 RC5 packets (78 work units) are in buff-out.rc5
.....10%.....20%._
```



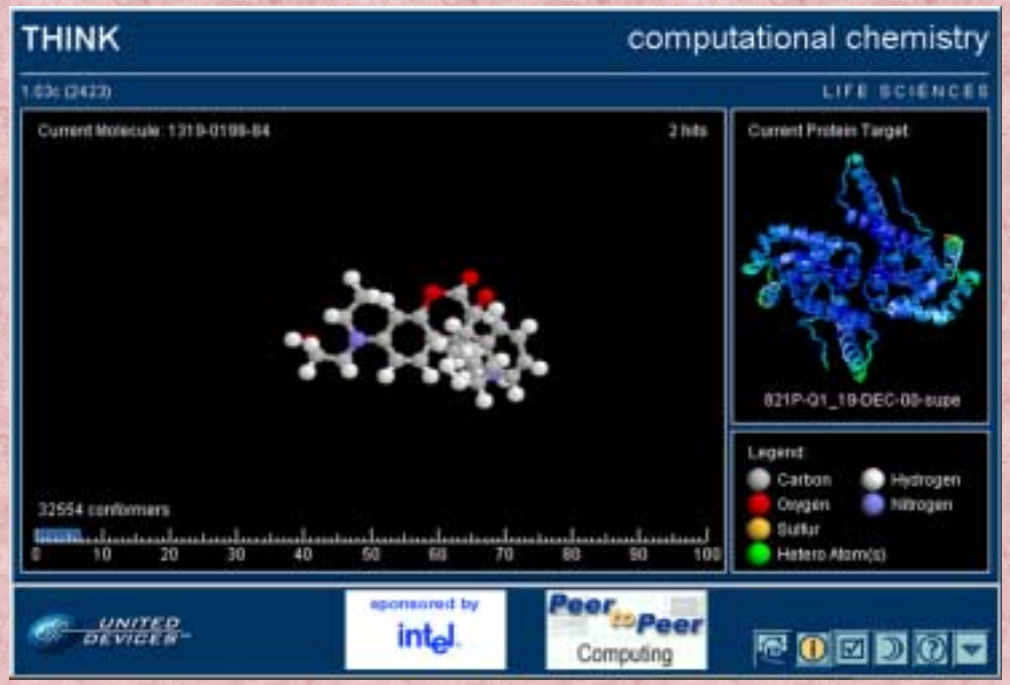
Pande Group – Stanford University

- genome@home
- folding@home



Intel Philanthropic Peer-to-peer Program

- Partners: American Cancer Society, National Foundation for Cancer Research, University of Oxford, United Devices



Leukemia Drug-Optimization Program

Hidden Markov Model – Pattern Matching in Genetic Sequences

Acknowledgements

Collaborator: Centre for Remote Imaging, Sensing and Processing (NUS)

Bioinformatics company (negotiating)

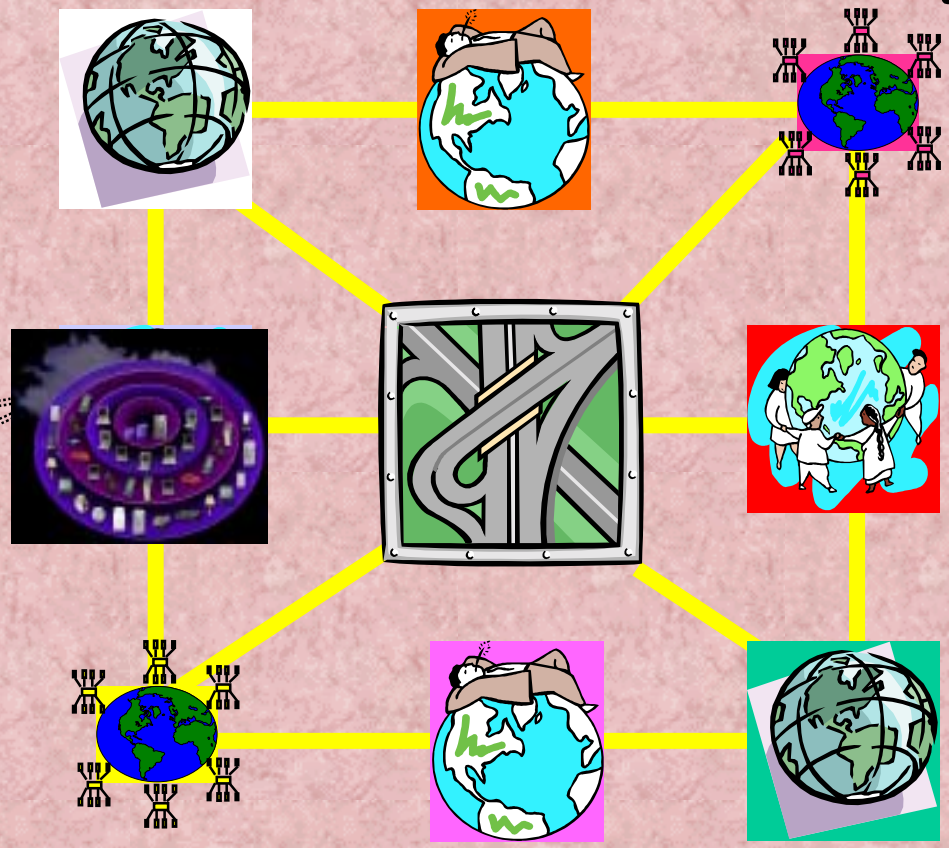
Current Students:

Johan Prawira Gozali, Ng Yean Seen, Low Seo Chin, Zheng Yudong, Ng Yew Kwong

Past Students: Lee Chwan Ren, Dr Wang Chen

The Future of P2P computing

Inter-Planetary Network (IPN)



Thank you.

Questions & Answers