Rapid Prototyping of Matlab/Java Distributed Applications using the JavaPorts Components Framework

Elias S. Manolakos

Electrical and Computer Engineering Department
Northeastern University, Boston MA 02115

Elias@ece.neu.edu

This work was supported in part by CenSSIS, the Center for Subsurface Sensing and Imaging Systems, under the ERC Program of the National Science Foundation (Award Number EEC-9986821)

Goals

JavaPorts is a *Distributed Processing Environment* (DPE) designed to facilitate the rapid prototyping and evaluation of end-to-end computation strategies that involve many concurrent tasks interacting with each other and assigned to a pool of networked heterogeneous resources.

Specifically, the goals of this project are to provide:

- Platform-independent Distributed Processing Environment
- Reusable Component Support
- Matlab support for legacy code and to take advantage of Matlab's extensive library of image processing functions
- Automated tools for configuration and execution that speed development and increase productivity

Features

With the latest version, *JavaPorts* 2.5, developers can:

- Create reusable software components in both Matlab and Java
- Seamlessly integrate Matlab and Java components in a distributed application

Furthermore, *JavaPorts* provides to developers:

- High-level Task Graph Abstraction and graphical tool (JPGUI) for describing Distributed Applications
- *JavaPorts* API that supports Anonymous Message Passing
- Automated Tools for Launching Distributed Application on the Network

Significance

- Many signal and image processing algorithms are inherently parallel and can take advantage of multiprocessor configurations to reduce computation time.
- Although computing PC clusters are an increasingly popular parallel processing platform, there is still a need for software tools to facilitate the rapid prototyping of distributed applications on these clusters.
- While Java provides the software developer with *"write once, run anywhere"* platform-independence, *JavaPorts* extends this capability to multithreaded code for networked computers with heterogeneous nodes.
- A parallel application developed using JavaPorts can easily be reconfigured to take advantage of new computing resources, or even moved entirely onto a different network, without requiring *any* code changes.

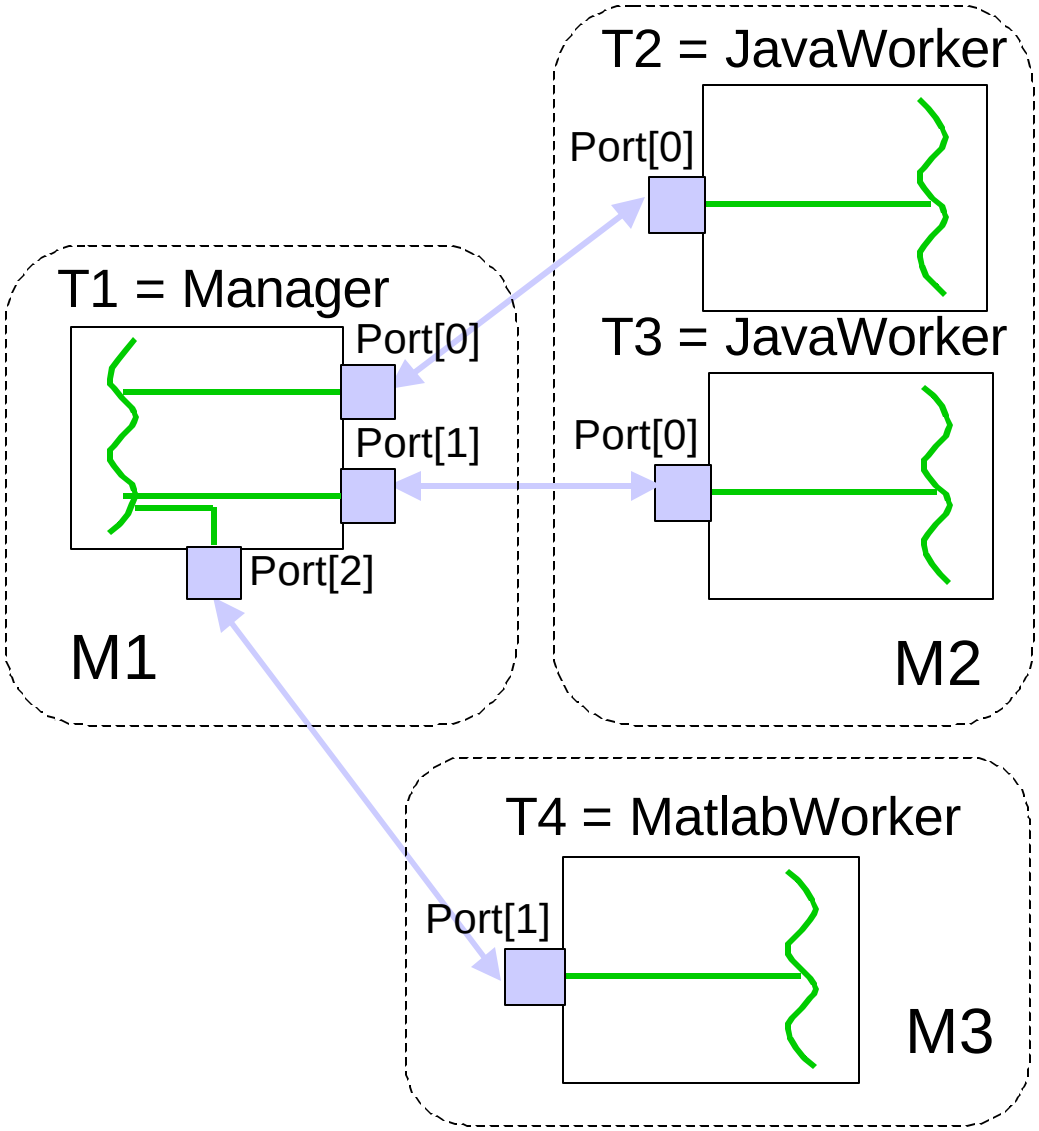
Contributions

- The *JavaPorts* project contributes to component-based parallel computing by providing an intuitive means of defining a distributed application via a high-level Task Graph abstraction.
- The *JavaPorts* environment provides the developer with all the tools necessary to develop a distributed application and launch it on a network of heterogeneous computing nodes.
- Collectively the *JavaPorts* GUI, Ports API, and runtime environment fill a void that exists in the area of component-based distributed application development.
- *JavaPorts* can be a powerful tool for anyone involved in image processing by enabling them to rapidly develop a distributed processing application using a combination of existing “off-the-shelf” software modules.
- *JavaPorts*-compliant software components can be written in Java or Matlab and integrated into the same application.

Related Projects

- Java//
  - A Java library that uses reification and reflection to support transparent remote objects and seamless distributed computing in both shared and distributed memory multiprocessors
- JavaParty
  - Supports transparent object communication by employing a run-time manager responsible for contacting local manager and distributing objects declared as remote
- JMPF
  - Provides portability of user applications over heterogeneous platforms while eliminating the need for complicated socket programming in the application code
- JavaPorts
  - Application is defined in a top-down fashion as a collection of interconnected concurrent tasks with clearly marked boundaries
  - Top-level application architect determines “what goes where”
  - Each software component can be developed independently

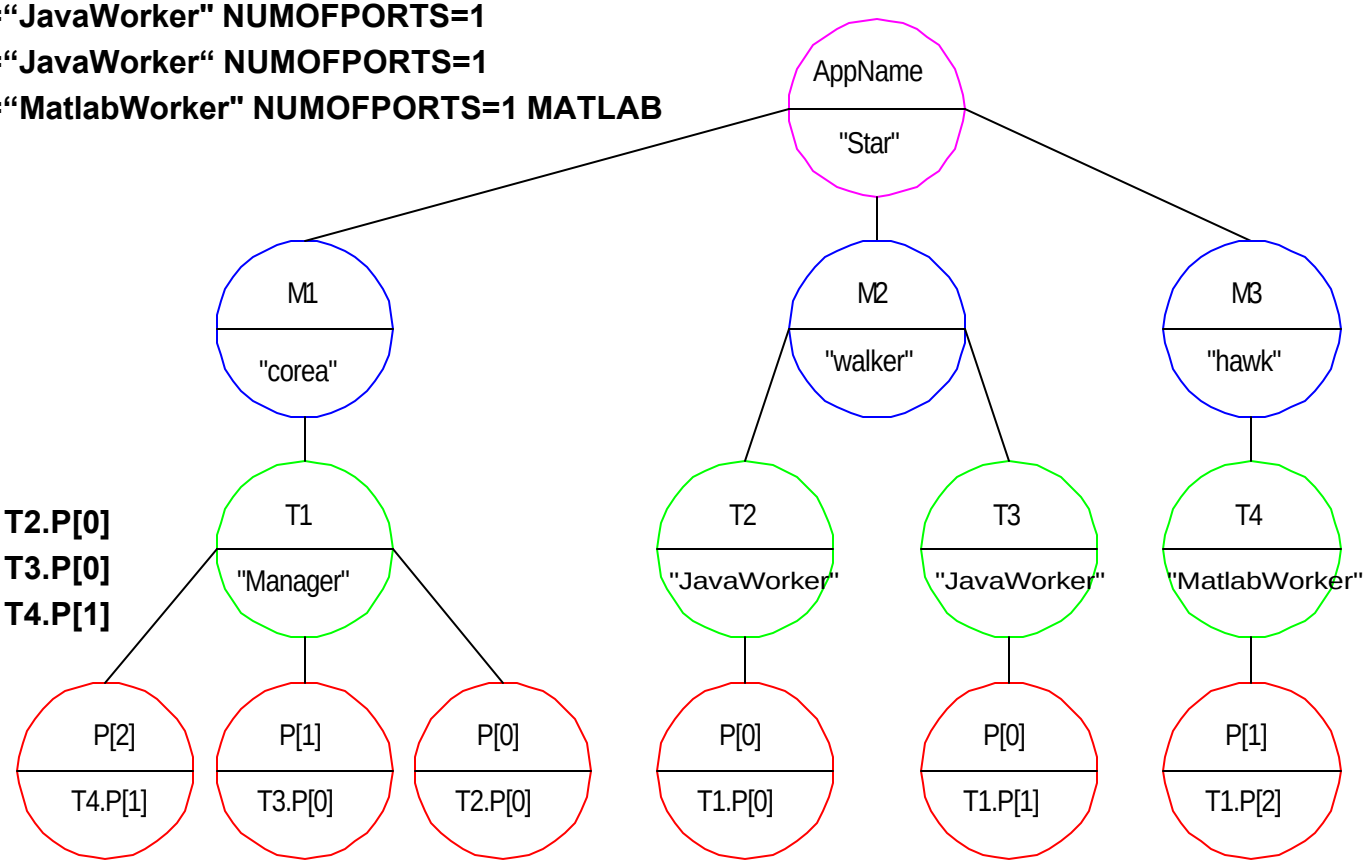
The JavaPorts Task Graph



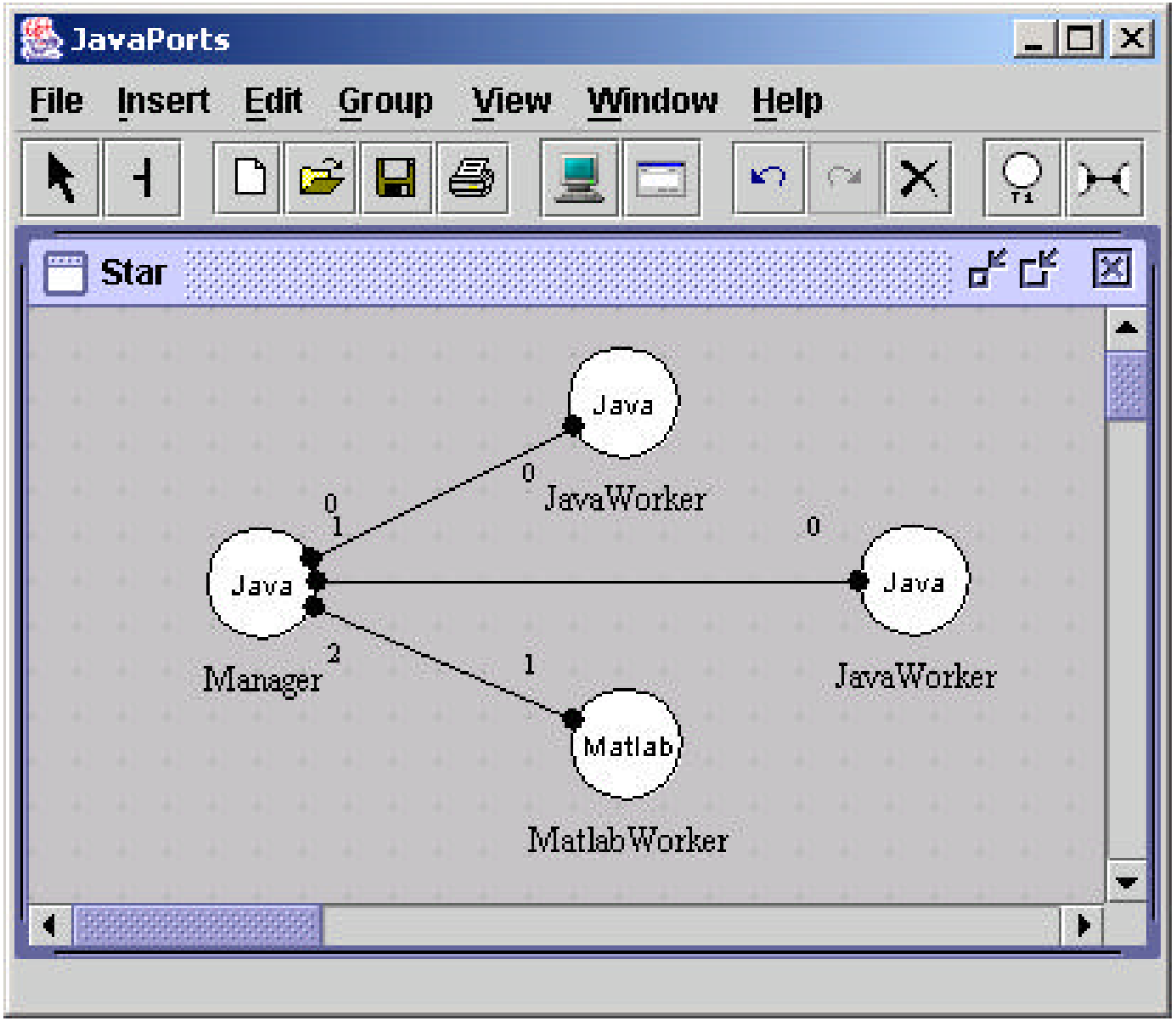
- Each Task is a Java thread or Matlab function
- Tasks communicate via Port objects
- A task writes to/reads from its own Ports and does not need to know where its Ports are connected to ***(anonymous message passing)***
- Tasks may correspond to new or existing software components
- Tasks are allocated to Machines (possibly many-to-one)
- ***Task source code remains the same even if the task allocation changes (portability)***

Task Graph Representations

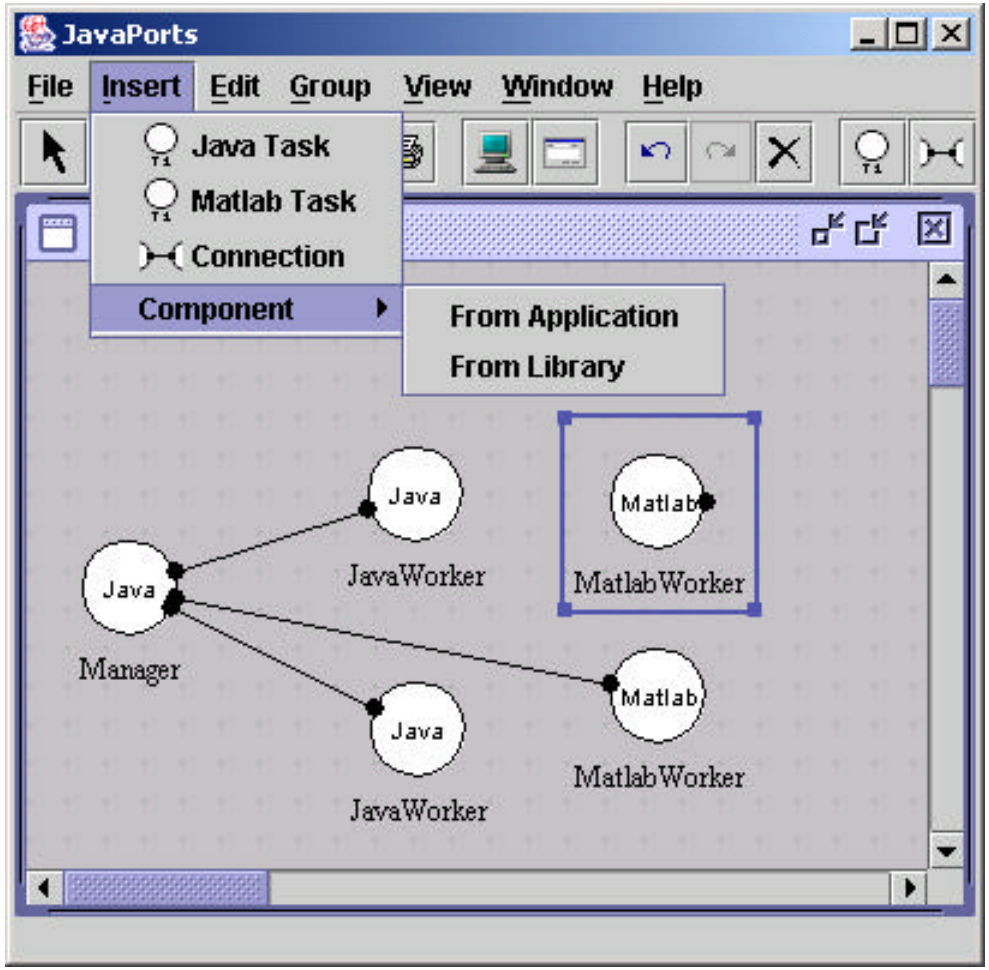
```
BEGIN CONFIGURATION
    BEGIN DEFINITIONS
        DEFINE APPLICATION “Star”
        DEFINE MACHINE M1=“corea" MASTER
        DEFINE MACHINE M2=“walker“
        DEFINE MACHINE M3=“hawk”
        DEFINE TASK T1="Manager" NUMOFPORTS=3
        DEFINE TASK T2=“JavaWorker" NUMOFPORTS=1
        DEFINE TASK T3=“JavaWorker“ NUMOFPORTS=1
        DEFINE TASK T4=“MatlabWorker" NUMOFPORTS=1 MATLAB
    END DEFINITIONS
    BEGIN ALLOCATIONS
        ALLOCATE T1 M1
        ALLOCATE T2 M2
        ALLOCATE T3 M2
        ALLOCATE T4 M3
    END ALLOCATIONS
    BEGIN CONNECTIONS
        CONNECT T1.P[0] T2.P[0]
        CONNECT T1.P[1] T3.P[0]
        CONNECT T1.P[2] T4.P[1]
    END CONNECTIONS
END CONFIGURATION
```



JPGUI tool for Task Graph capture

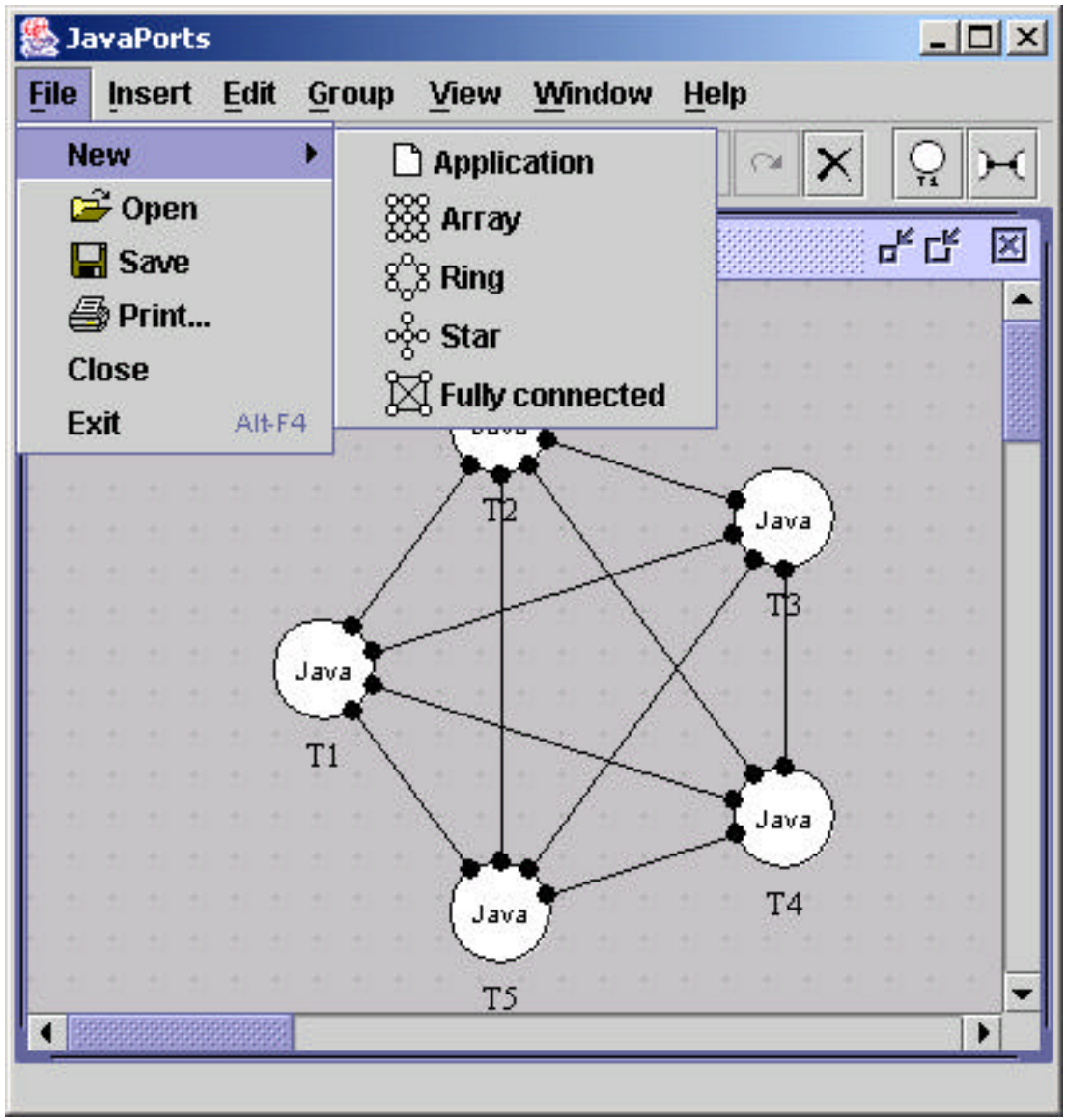


Graphical Component-based Development

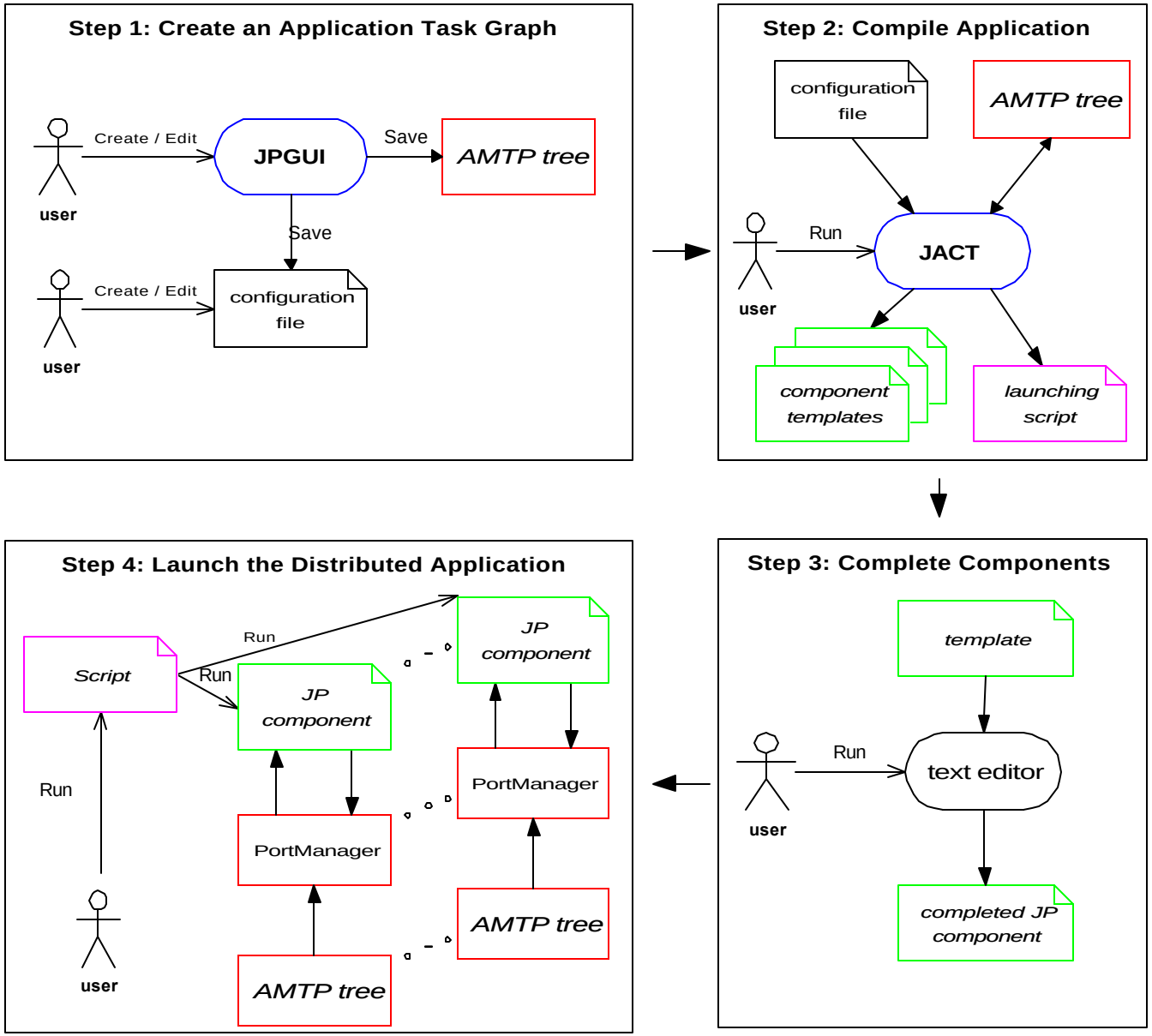


A second MatlabWorker component is imported into the application

Automatic Task Graph Generation



The Application Development Cycle



The JavaPorts API

- **public Object AsyncRead(int MsgKey) throws RemoteException**
  - Allows calling task to retrieve a message from one of its ports. The message is identified by the integer-valued *MsgKey*. If the message has not arrived AsyncRead does not block - it returns null and terminates.

- **public void AsyncWrite(Object msg, int MsgKey) throws RemoteException**
  - Spawns a new thread in order to transfer (asynchronously) the specified message Object. If the receiving port is not waiting, the message will be stored in its port list. The calling task is not blocked.

- **public Object SyncRead(int MsgKey) throws RemoteException**
  - Similar to AsyncRead but it will block the calling task until the specified message arrives at the port.

- **public void SyncWrite(Object msg, int MsgKey) throws RemoteException**
  - Similar to AsyncWrite but the calling task will block until the message is retrieved by the receiving task. No new thread is spawned.

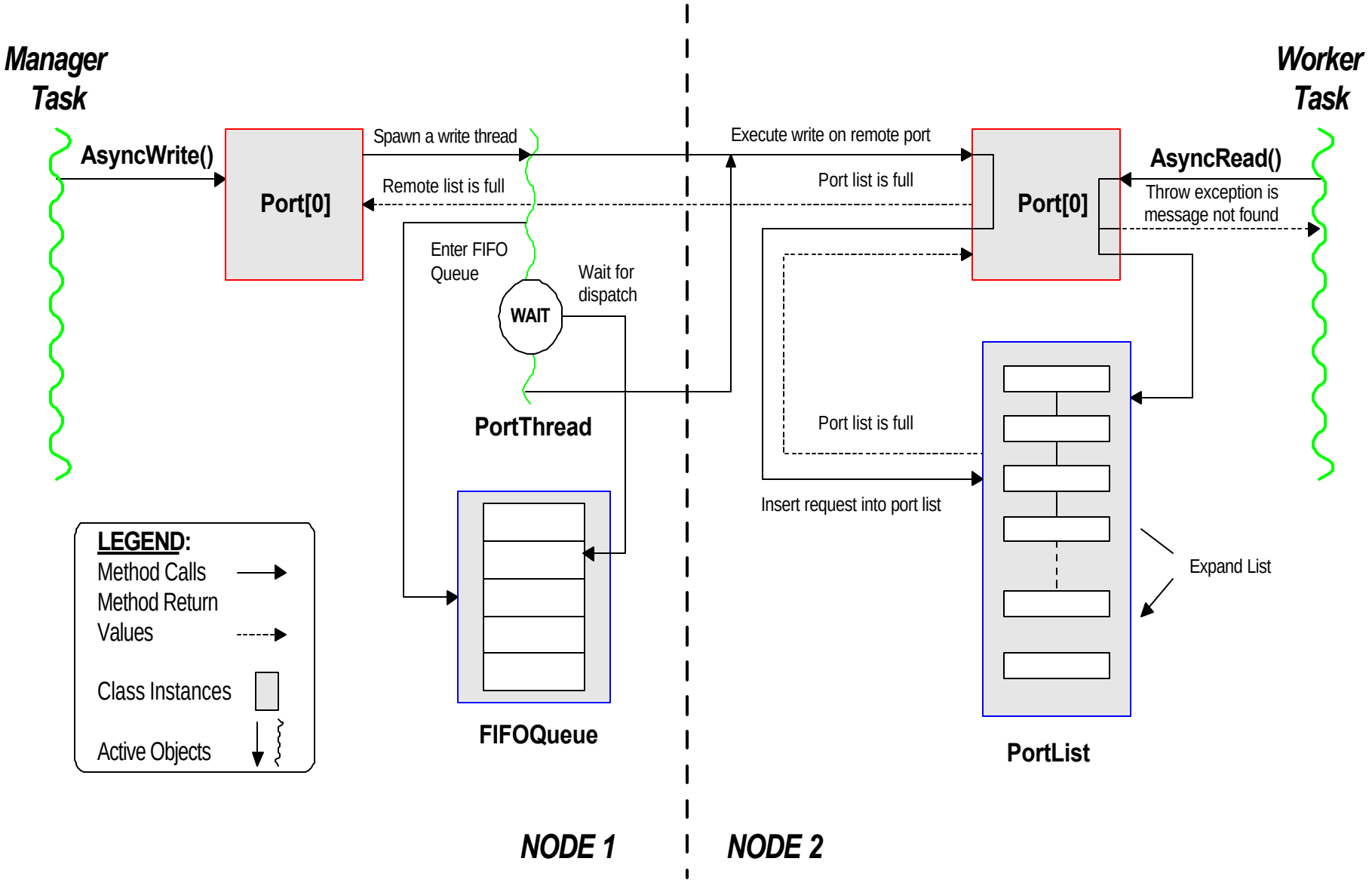
The Java Code Template

```
public class Manager extends Thread {
    private static Port[] port_;
    private String TaskVarName_;
    public Manager(String TaskVarName) { // constructor
        super();
        TaskVarName_ = TaskVarName;
    }
    public synchronized void run () { try {
        PortManager portmanager = new PortManager();
        port_ = portmanager.configure("Star", TaskVarName_);
        // User application code goes here
        portmanager.release(); // distributed termination
    } catch(Throwable e) { e.printStackTrace(); System.exit(-1); } }
    public static void main (String args[]) {
        Manager ManagerThread = new Manager(args[0]);
        ManagerThread.start();
        try { ManagerThread.join(); } catch (Throwable e) { e.printStackTrace();
                System.exit(-1); } System.exit(0); }
}
```

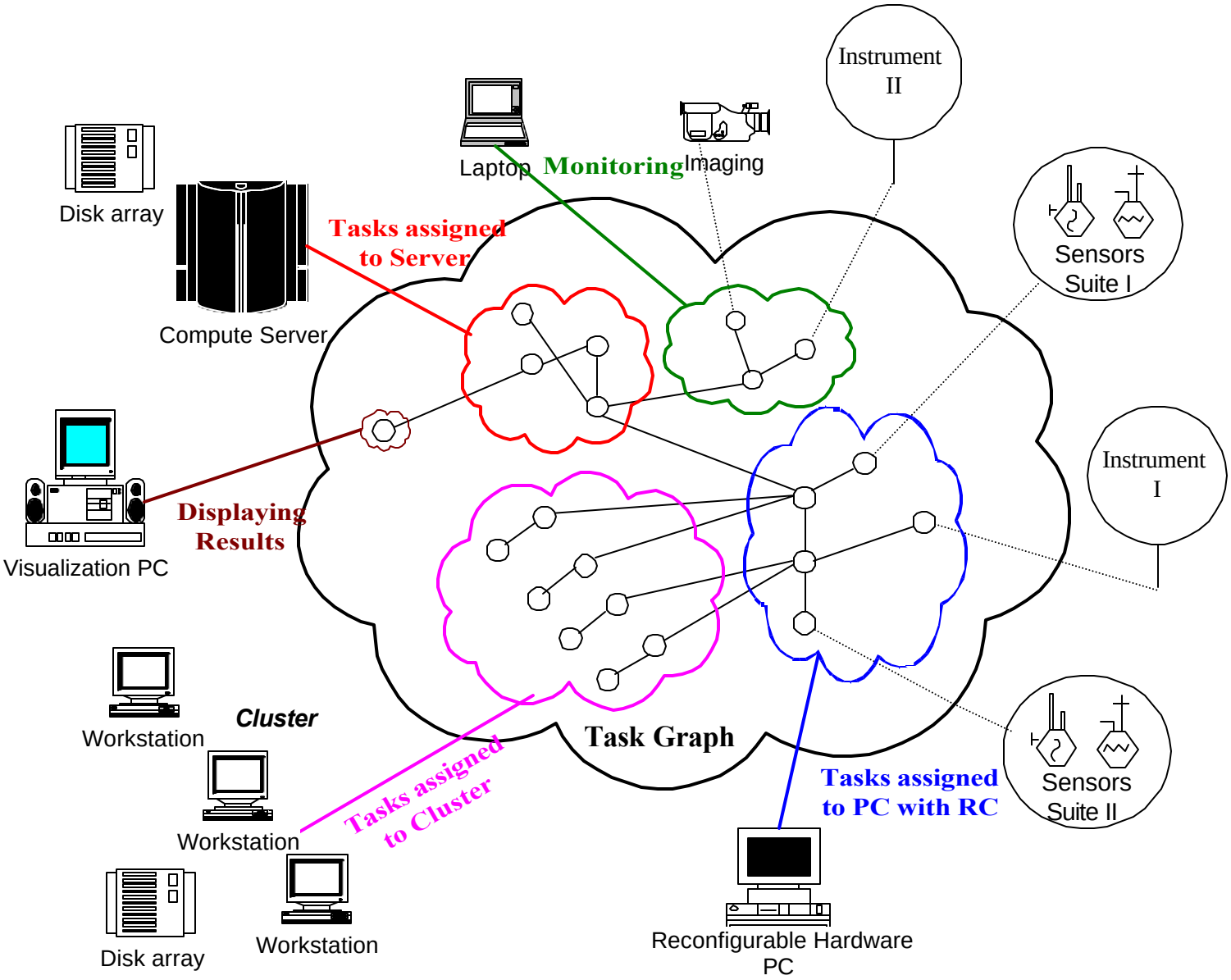
The Matlab Code Template

```
function Manager(TaskVarName)

try
    portmanager = PortManager;
    port_ = portmanager.configure(“Star”, TaskVarName);

    % User Application code goes here

catch
    exceptionstr = lasterr;
    exceptionstr
end

portmanager.release; // distributed termination
quit;
```

The Port-to-Port Communication Protocol



JavaPorts Cluster Applications



Ping Pong Experiments

- Simple ping-pong message passing between two *JavaPorts* tasks
- Used to benchmark performance when passing large images between nodes
- Considered four different message passing combinations (Sync-Sync, Async-Sync, Sync-Async, Async-Async)
- Compared JavaPorts to MPI, and Java tasks with Matlab tasks

Java Ping Pong Code

Manager

```
for (int r = 0; r < R; r++) {
long time = System.currentTimeMillis();

port_[0].SyncWrite(message, 0);

messRecv=(Message)port_[0].SyncRea
    d(0);

time = System.currentTimeMillis() -
    time;
}
```

Worker

```
for (int r = 0; r < Manager.R; r++) {

message=
(Message)port_[0].SyncRead(0);

port_[0].SyncWrite(message, 0);

}
```

Matlab Ping Pong Code

Manager	Worker
for r = 1:R, tic;	for r = 1:R,
port_(1).SyncWrite(sendMessage, 0); ⟶	message = port_(1).SyncRead(0);
recvMessage = port_(1).SyncRead(0); ⟵	port_(1).SyncWrite(message, 0);
rtt(r) = toc; end;	end;

LAM/MPI Ping Pong Code

Manager

```
start = MPI\_Wtime();
for (i = 1; i <=  NTIMES; i++) {

MPI_Ssend(buffer, length, MPI_CHAR,
proc_B, ping, MPI_COMM_WORLD);

MPI_Recv(buffer, length, MPI_CHAR,
proc_B, pong, MPI_COMM_WORLD,
    &status);

}
finish = MPI_Wtime();
```

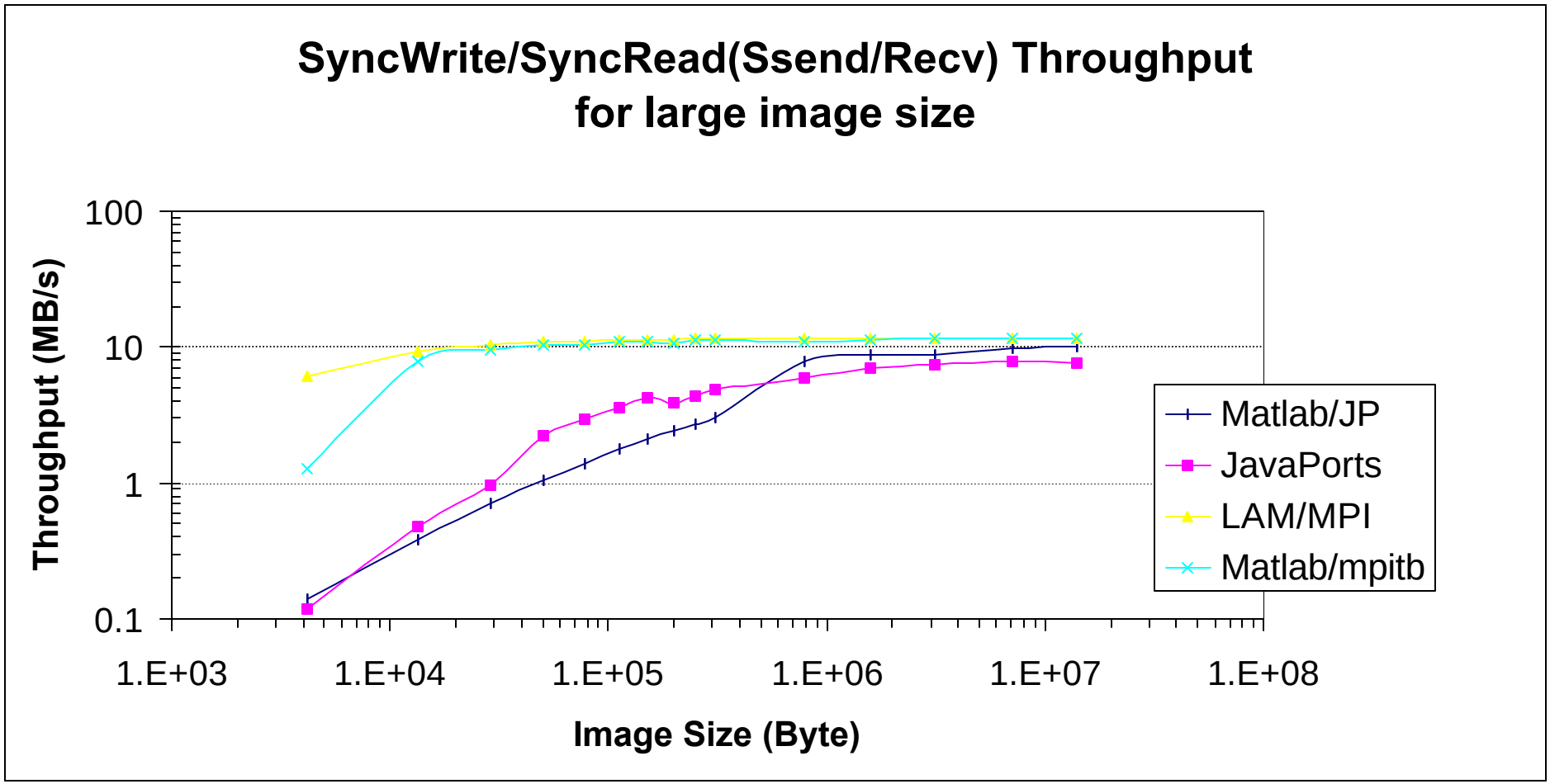
Worker

```
for (i = 1; i <= NTIMES; i++) {

MPI_Recv(buffer,  length,MPI_CHAR,
proc_A,  ping,  MPI_COMM_WORLD,
&status);

MPI_Ssend(buffer, length, MPI_CHAR,
proc_A, pong, MPI_COMM_WORLD);
}
```

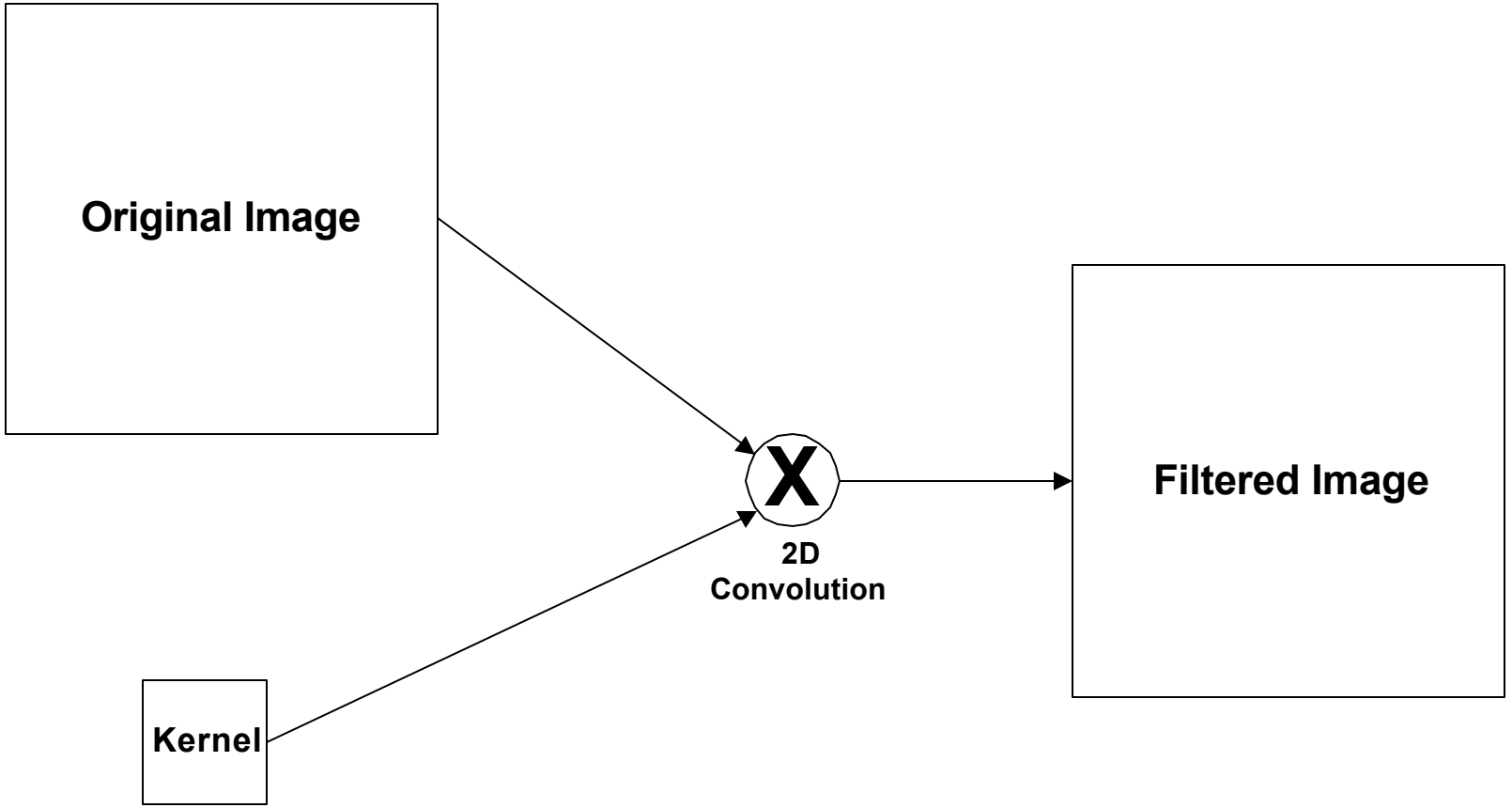
Matlab/MPITB Ping Pong Code

Manager		Worker
T=clock;		
for i=1:NTIMES		for i=1:NTIMES
MPI_Ssend(array,1,TAG,NEWORLD);	→	MPI_Recv(array,0,TAG,NEWORLD);
MPI_Recv(array,1,TAG,NEWORLD);	←	MPI_Ssend(array,0,TAG,NEWORLD);
end		end
T=etime(clock,T);		

Ping Pong Timing Results

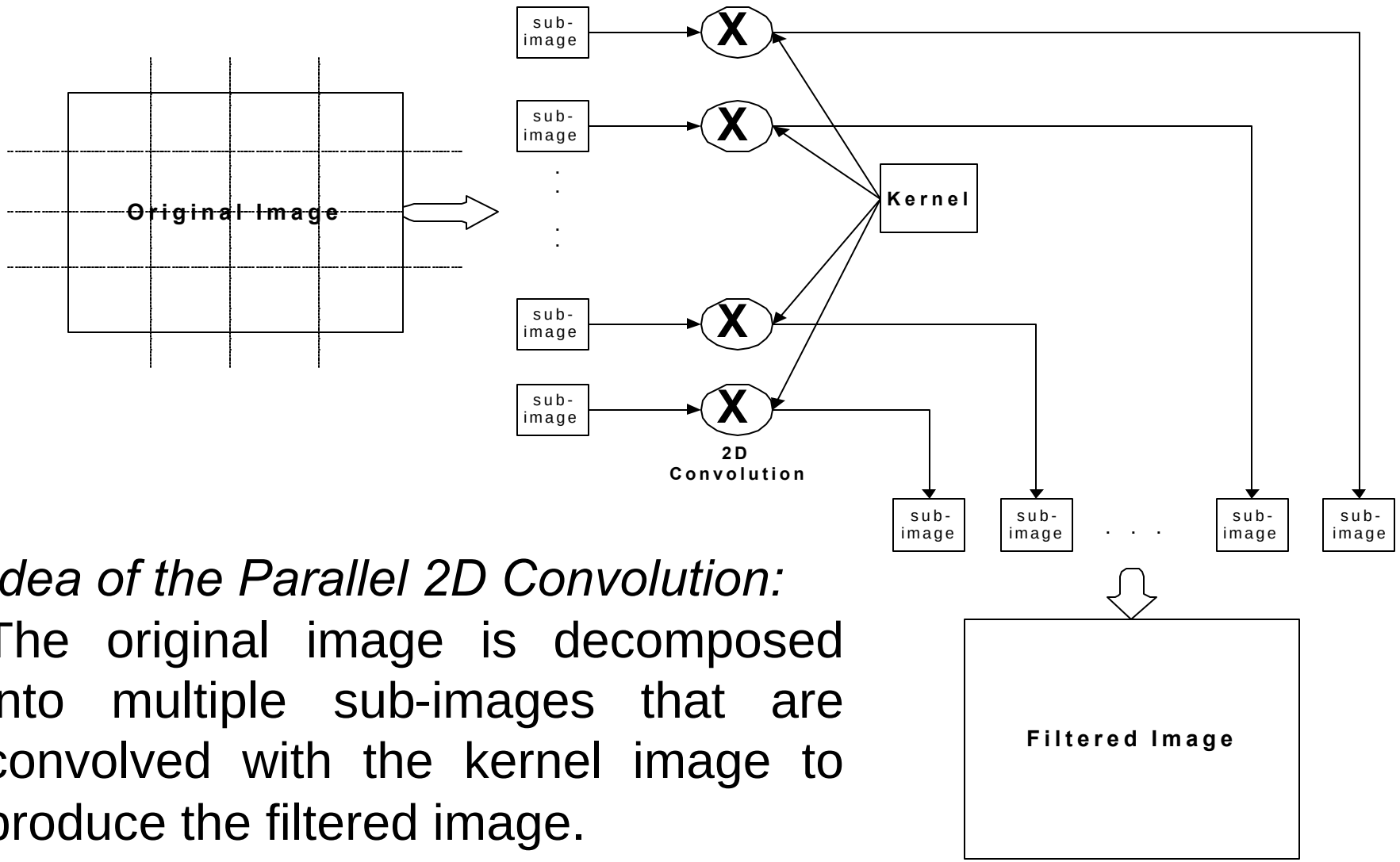


Serial Image Deblurring



Serial 2D Convolution

Parallel Image Deblurring



Idea of the Parallel 2D Convolution:

The original image is decomposed into multiple sub-images that are convolved with the kernel image to produce the filtered image.

Image Deblurring Application

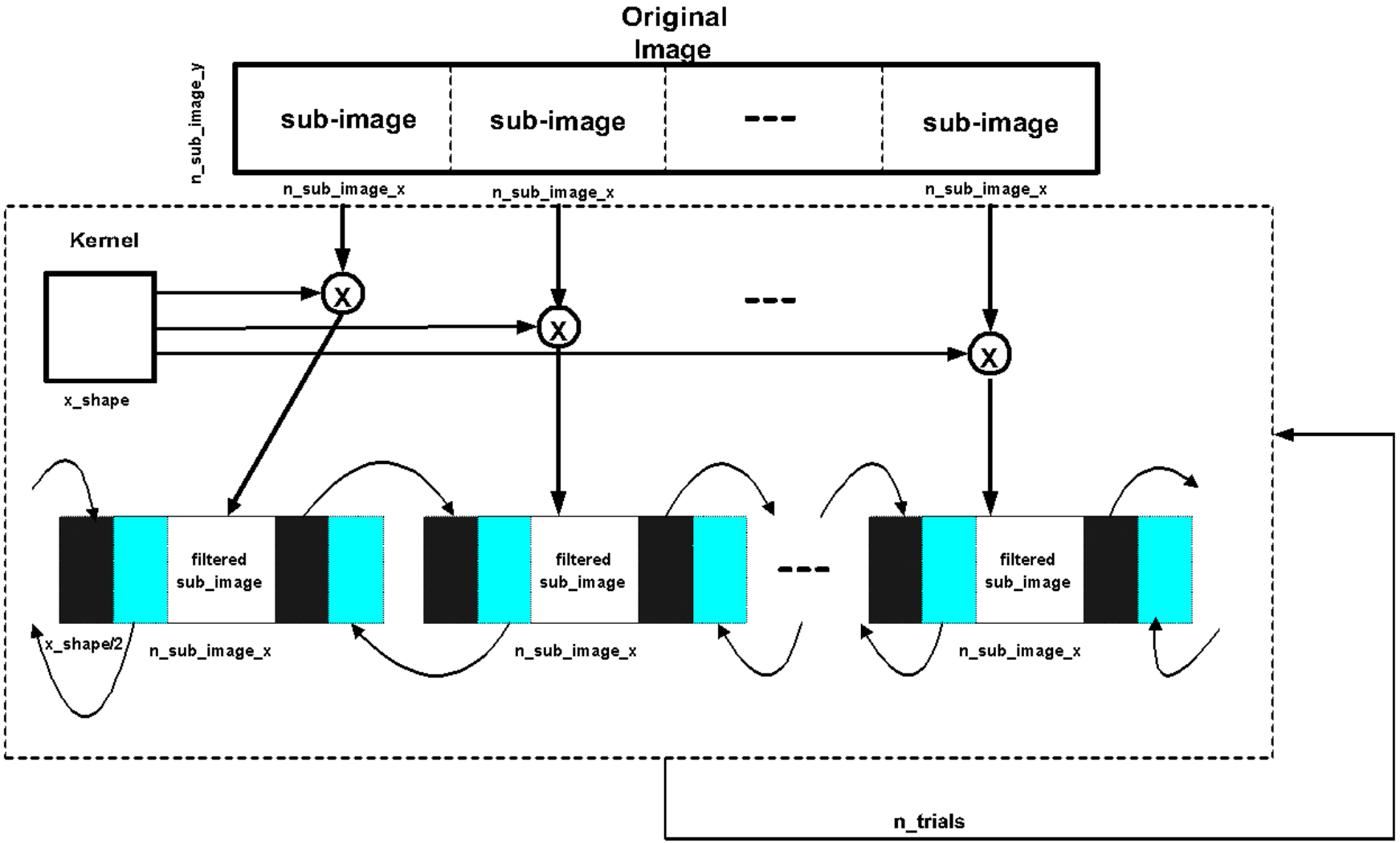
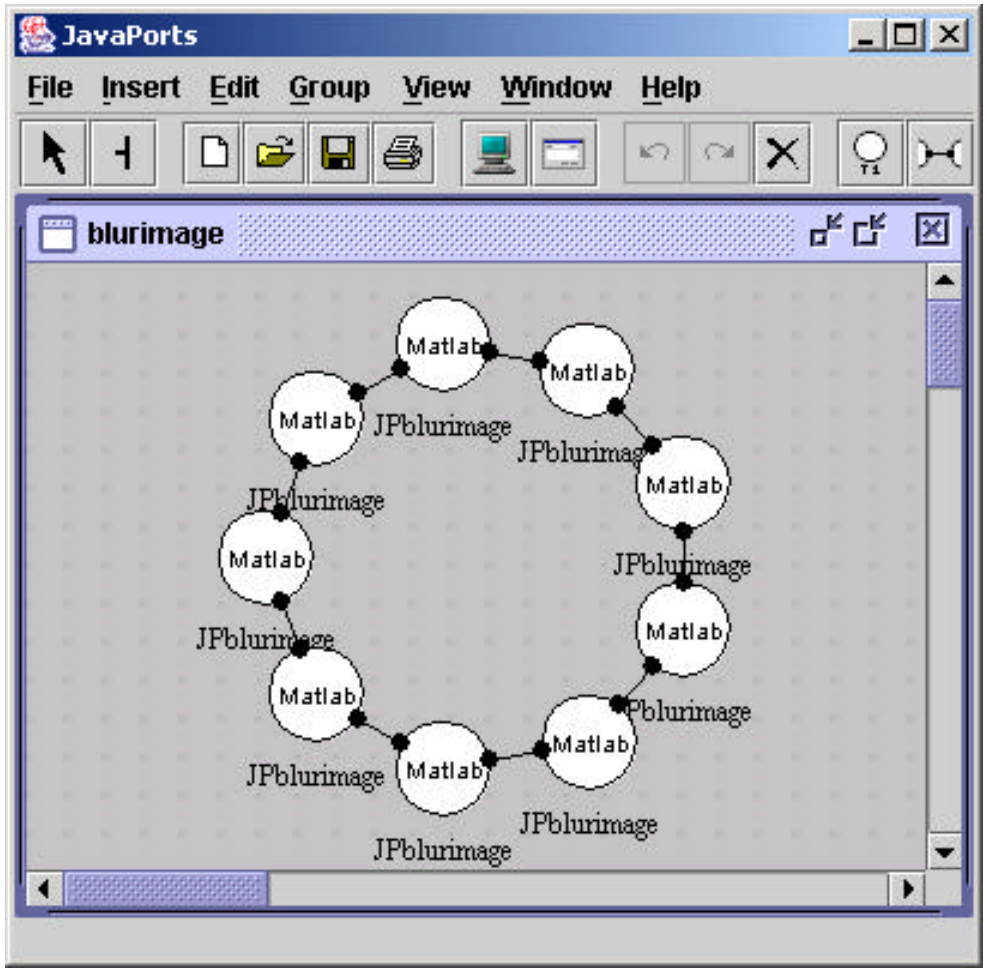
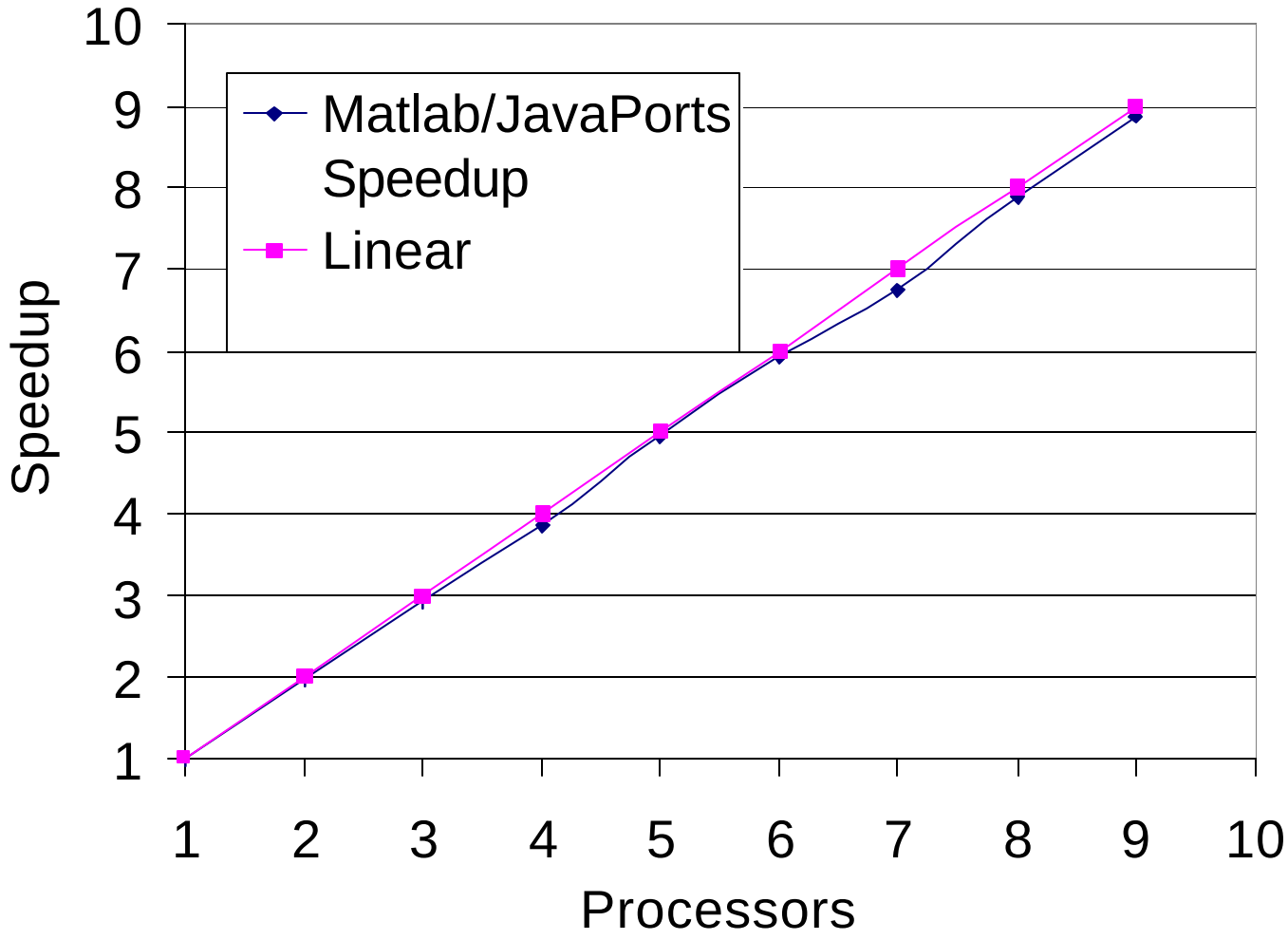


Image Deblurring Application: Task Graph



Parallel Deblurring Speedup

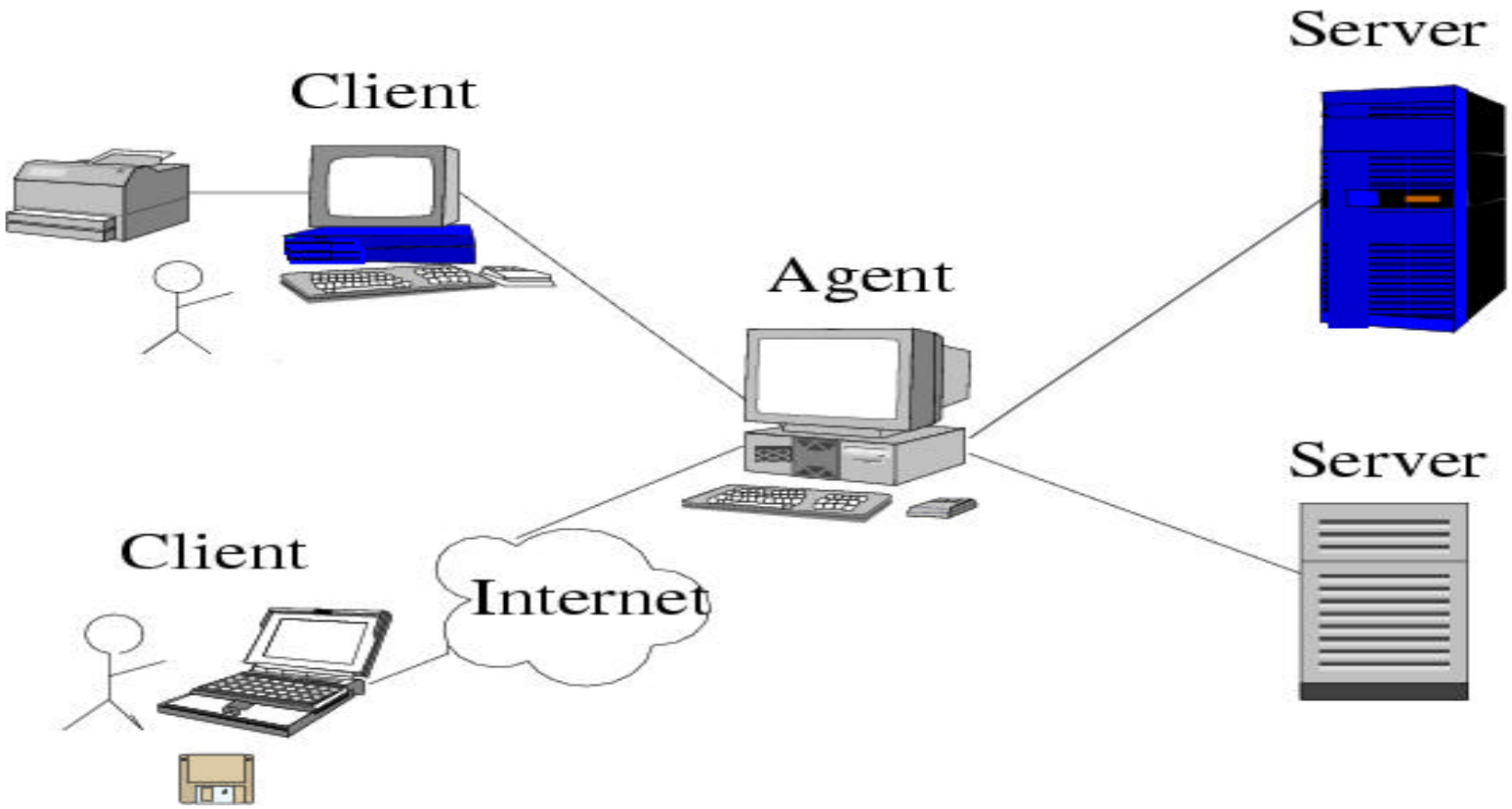


Client-Agent-Server Application

Client-Agent-Server architecture is a generalization of the Client-Server architecture. To have client and server focus on application-related work, the agent may facilitate:

- Load balancing
- Service discovery
- Fault tolerance
- Scalability

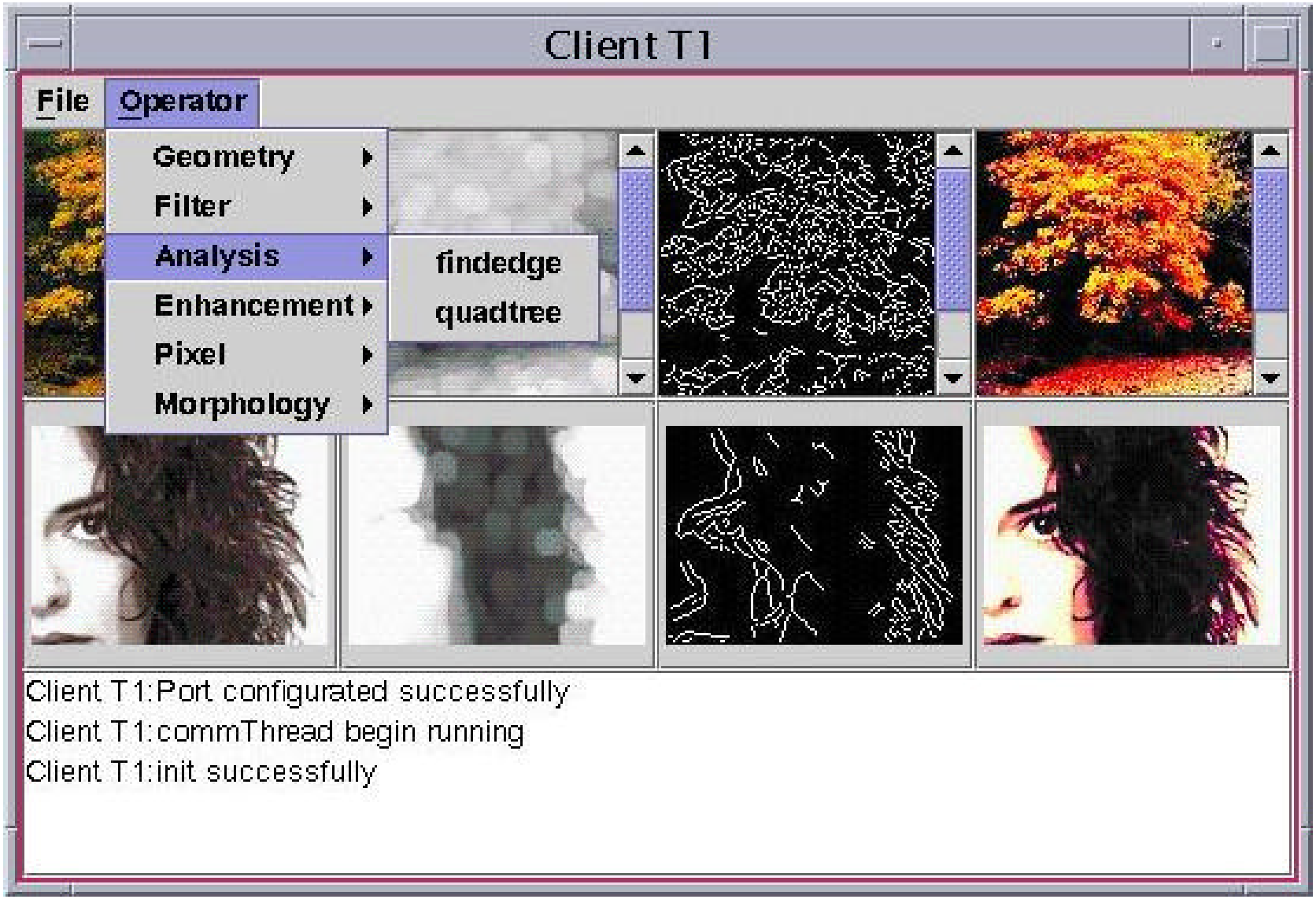
Client-Agent-Server Architecture



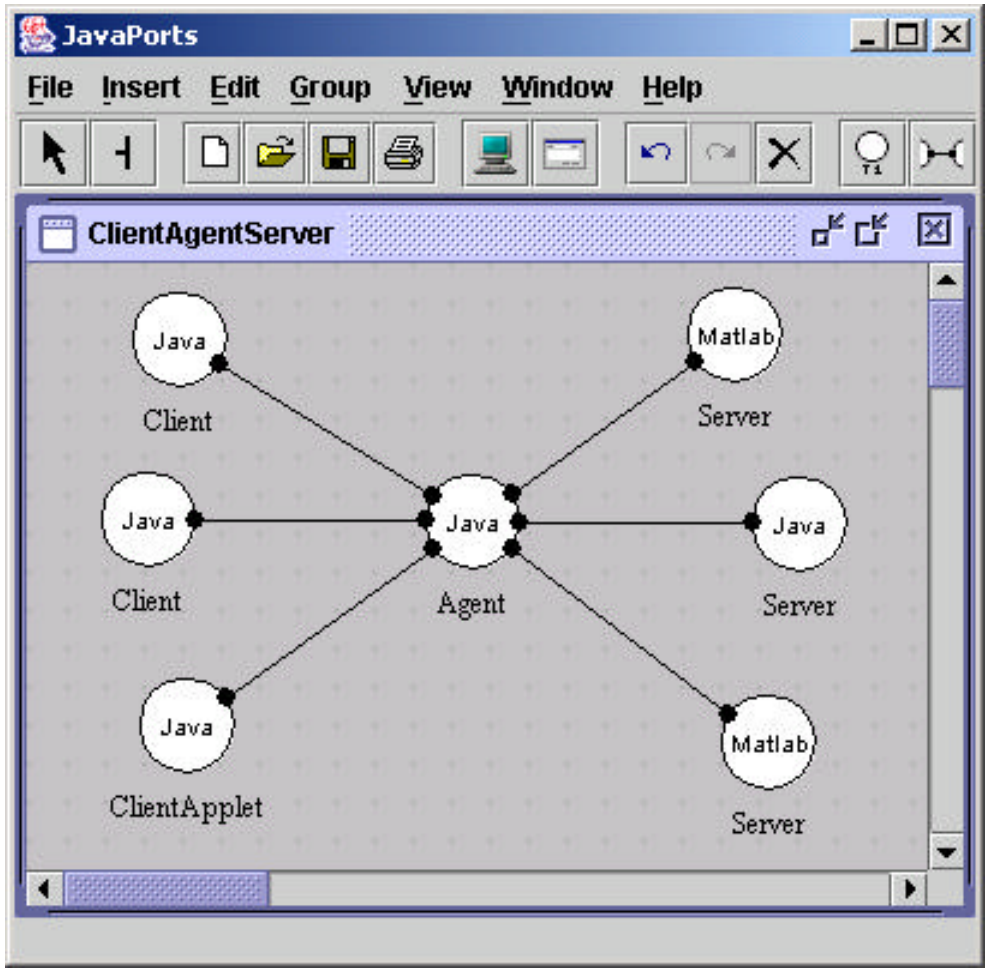
Distributed Image Processing

- Client is responsible for collecting image data and sending required service information to the Agent.
- Client can be a Java application or a Java Applet.
- Through an Internet browser the user can launch the Java Applet to request access to remote server resources.
- Agent has knowledge of the image processing services provided by each server, the network bandwidth, the system configuration, etc, and acts as a proxy between Client and Server.
- Server can implement an image processing components library exploiting the JavaPorts task-to-task communication API. E.g., Java Advanced Imaging(JAI), Matlab Image Processing Toolbox, etc.

The Client GUI



Client-Agent-Server Configuration

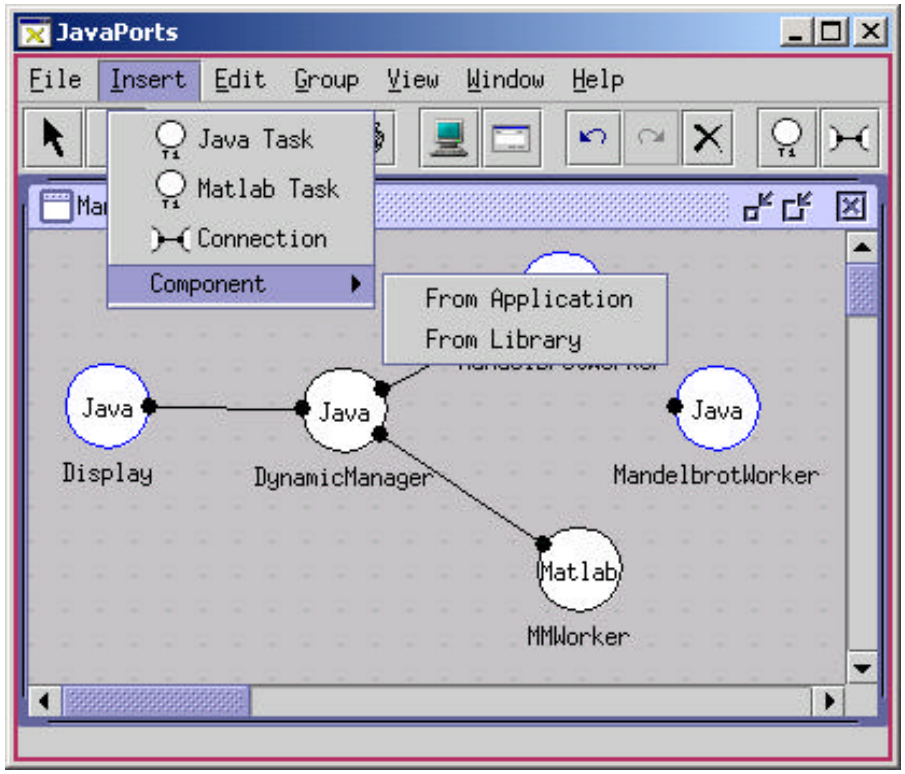


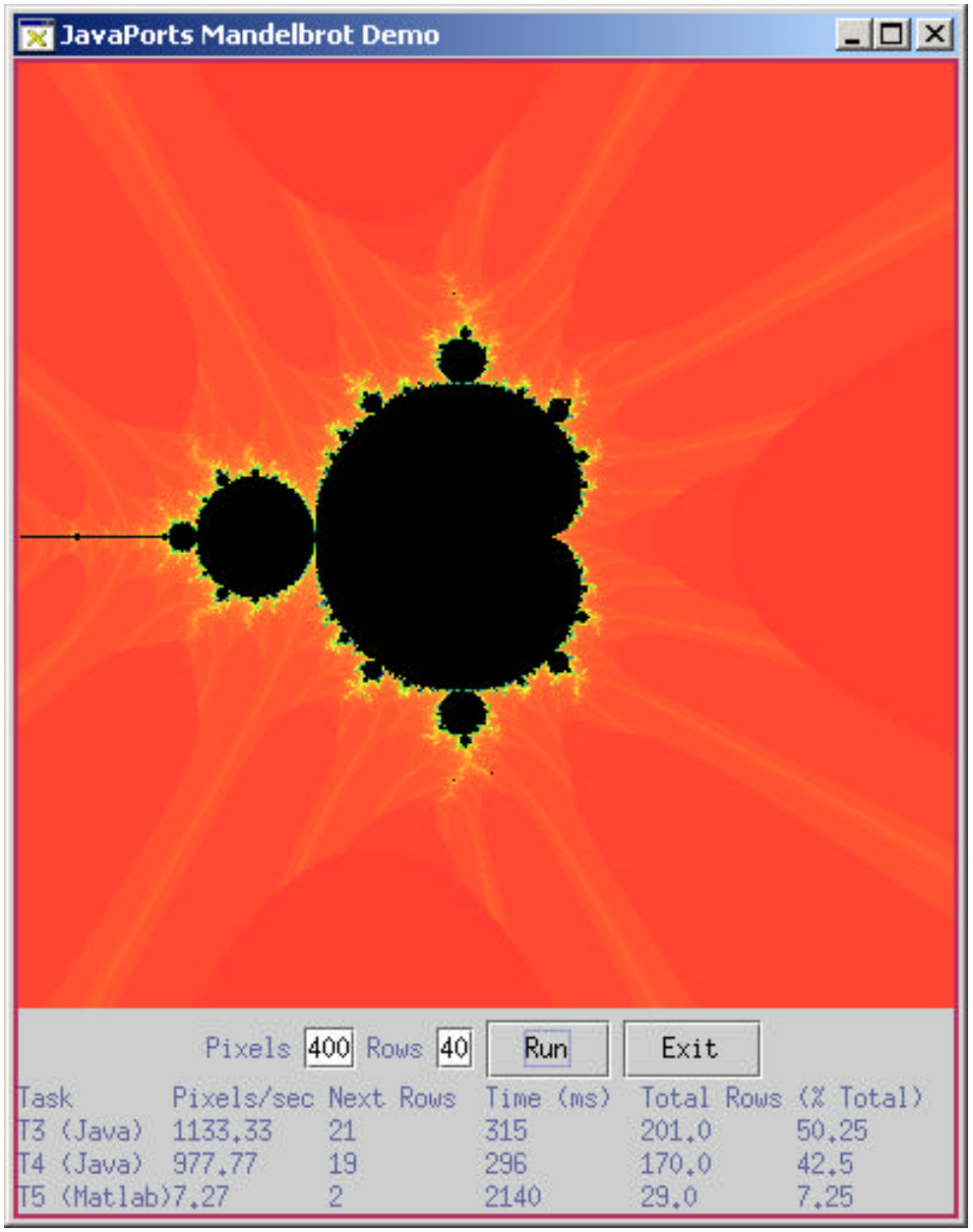
- The Application consists of Client components that interface with an Agent component to request image processing functions from the Server components.
- The Broker implements a load-balancing scheme to distribute the requests to the appropriate Servers in order to optimize the performance.
- ***Using the JPGUI, a variety of Client, Agent, and Server components can be swapped in and out of the application to meet the current requirements, without necessitating any code changes.***

The Mandelbrot Application

- Parallel Mandelbrot Set Computation
- Demonstrates ease with which a distributed application can be developed using a combination of reusable Matlab and Java software components
- Matlab and Java Worker components are completely interchangeable – no code changes are required to add or swap components
- Manager dynamically distributes computational load based on each Worker's performance

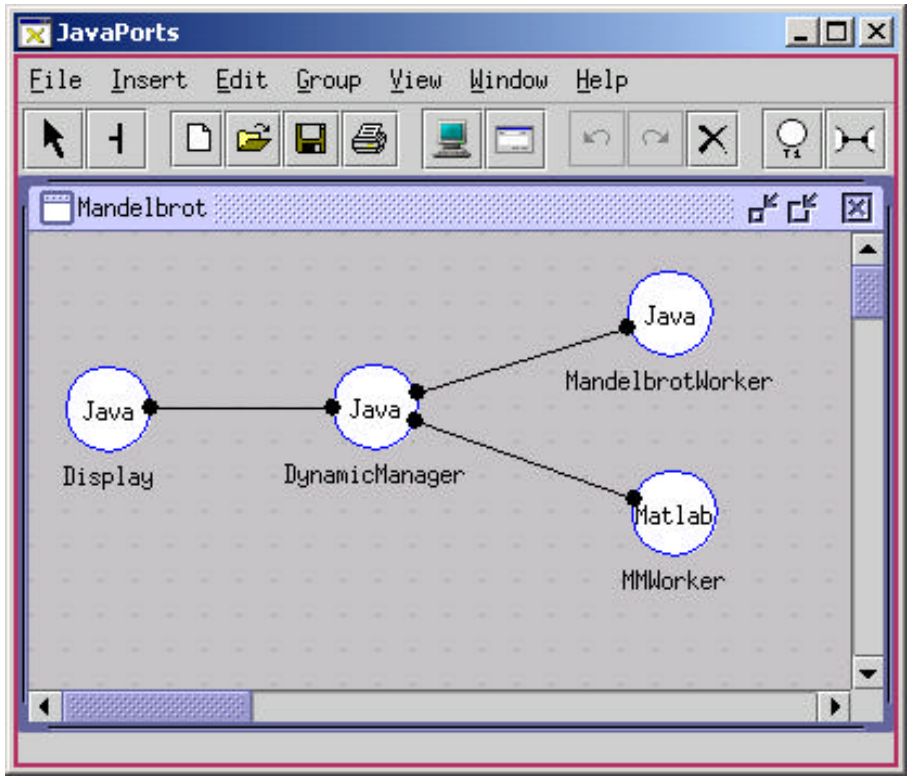
Mandelbrot Application Development

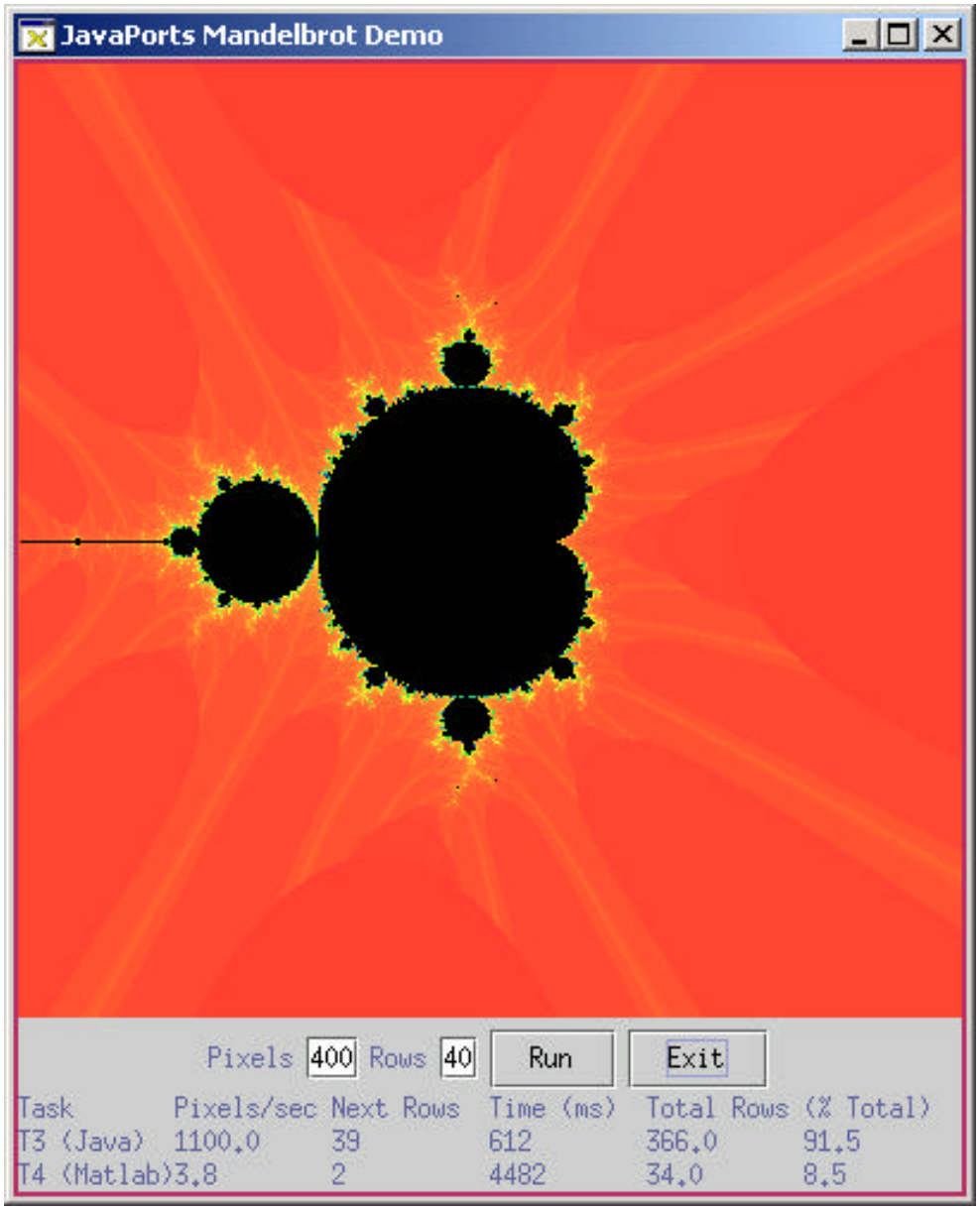




- A Java Worker component is added to increase performance
- Any combination of Java and Matlab Workers may be used

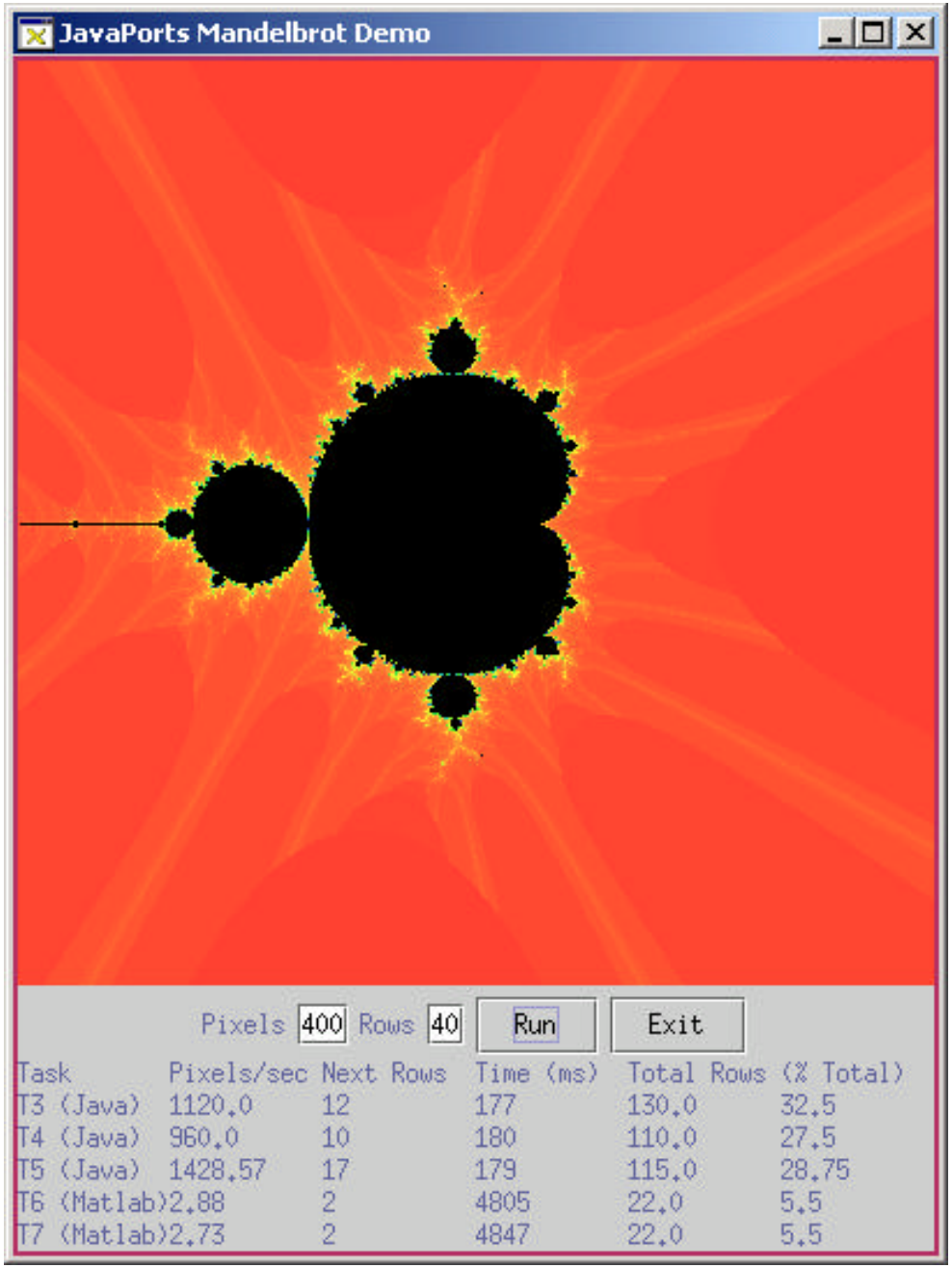
The Mandelbrot Application





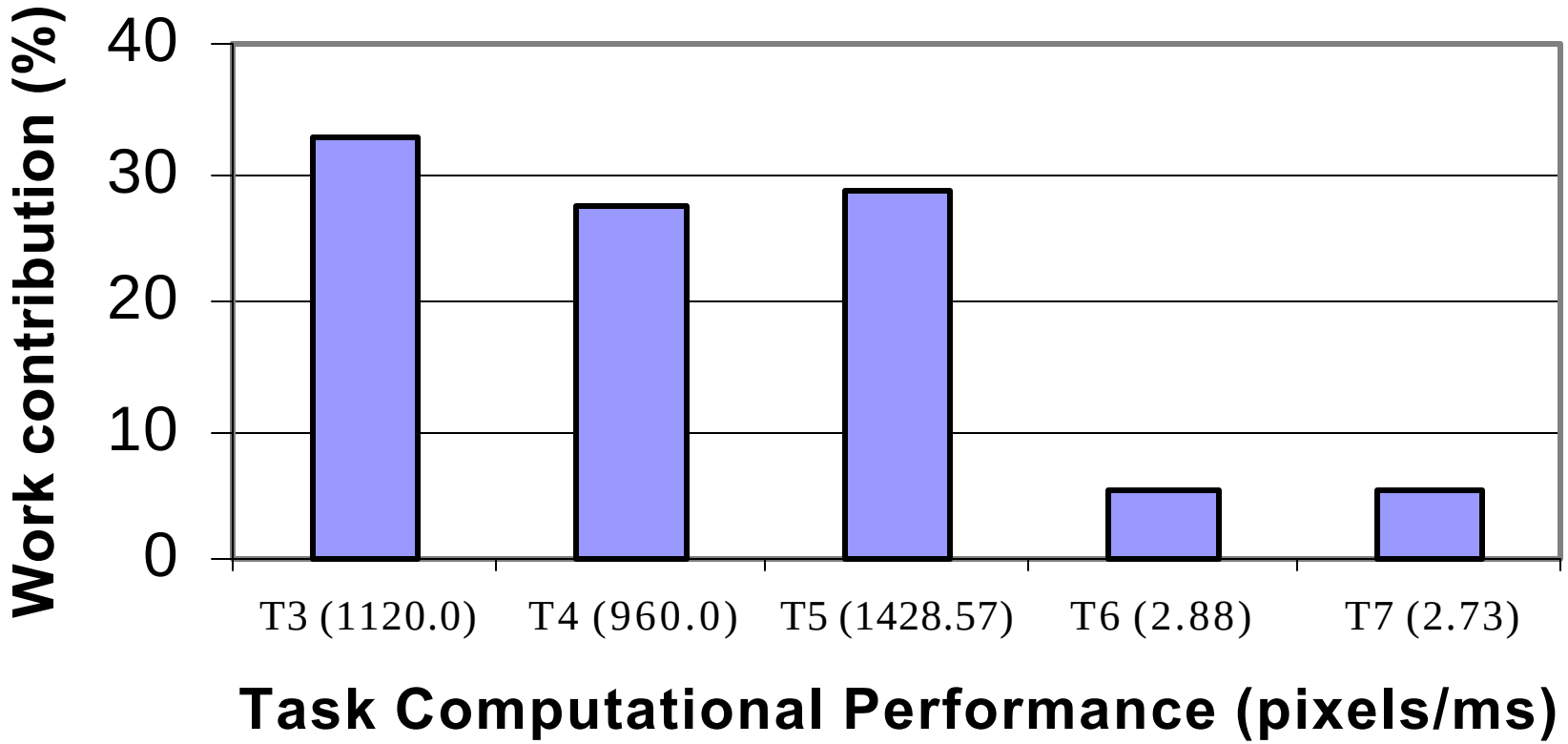
DynamicManager sends rows to each Worker based on its computational performance (pixels/ms) →

Mandelbrot Load Balancing



- DynamicManager starts each Task with an equal fraction of the Row increment
- In this case, the Row increment is 40, each of 5 Tasks starts with 8 rows of pixels to compute
- Some Tasks will be faster than others due to such factors as implementation or machine loading
- As the application progresses, the DynamicManager adjusts each Task's fraction up or down according to its computational performance

Mandelbrot Work Distribution



Project Evolution

We are currently investigating issues such as:

- Creation of a component library for Subsurface Sensing and Imaging applications and testBEDS
- Integration of *JavaPorts* components with functions in C, Matlab, for flexible task development
- Integration of *JavaPorts* with hardware components for hardware/software co-simulation and co-design
- *JavaPorts* tasks dispatching through applets and software agents
- Ports specialization
- Dynamic Port creation and deletion at run-time
- Application-level Quality Of Service (QoS)
- Passing active messages and code mobility

Recent Publications

- E. Manolakos, D. Galatopoullos and A. Funk. “Component-based peer-to-peer distributed processing in heterogeneous networks using *JavaPorts”*. *Proceedings of the 2001 IEEE International Symposium on Network Computing and Applications*, pp. 234-237, Cambridge MA, February 2002.

- E. S. Manolakos, D. Galatopoullos, A. Funk, "*JavaPorts*: An Environment for the Rapid Prototyping of Heterogeneous Network-centric Distributed Processing Applications" in the Proceedings of the *Fifth Workshop on High Performance Embedded Computing* (HPEC-01), MIT Lincoln Labs, Lexington, MA, November 2001

- L.A. King, H. Quinn, M. Leeser, D. Galatopoullos, E.S. Manolakos. “Runtime Execution of Reconfigurable Hardware in a Java Environment”. *Proceedings of the IEEE International Conference on Computer Design (ICCD-01)*, pp. 380-385, September 2001.

- D. Galatopoullos, E.S. Manolakos. “Developing Parallel Applications using the *JavaPorts* environment”. *Parallel and Distributed Processing*, Jose' Rolim Editor, Lecture Notes in Computer Science, Elsevier Publ., vol 1586, pp. 813-828, 1999.

- Contact Information: Prof. Elias S. Manolakos, ECE Dept. Northeastern University Phone: 617-373-3021, Fax: 617-373-4189, email: elias@ece.neu.edu

- *JavaPorts* Project web site: http://www.cdsp.neu.edu/info/faculty/manolakos/JavaPorts.html