Programming Environments for Grids and Distributed Computing Systems

Vaidy Sunderam
Emory University, Atlanta, USA
vss@emory.edu

Edgar Gabriel
University of Tennessee
egabriel@cs.utk.edu

EMORY UNIVERSITY

Harness II Research Project

- Emory Participants
  - Dawid Kurzyniec
  - Tomasz Wrosek
  - Tomasz Ampula
  - Peter Hwang
  - Vaidy Sunderam
- Collaborators
  - Oak Ridge Labs (A. Geist, C. Engelmann, J. Kohl)
  - Univ. Tennessee (J. Dongarra, G. Fagg, E. Gabriel)
- Sponsors
  - U. S. Department of Energy, MICS
  - National Science Foundation, ACR

EMORY UNIVERSITY

Outline

- Introduction and Background
- Grids and metacomputing systems
- Service oriented distributed computing
  - Overview of GT3
- The H2O and Harness projects
  - RMIX and H2O Programming
- FT-MPI
  - Concepts, Architecture, API
  - Example programs and Results
  - Demonstration
- Summary and Conclusions

Parallel Programming

- Research and development
  - Over 60 parallel programming models/languages
    - http://www.cs.rit.edu/~ncs/parallel.html
- Parallel programming for HPC
  - Evolved over approximately 2 decades
  - Closely tied to machine architectures
    - Intel iPSC and nCUBE message passing
    - Sequent shared memory
  - Closely tied to applications and user requirements
    - Support for Fortran, porting/updating of legacy codes
    - Procedural programming; manual partitioning/mapping
    - Some compiler technology (decomposition, directives)

Issues and Challenges

- Parallel processing: divide work
- Task management
  - UoP (process, thread) creation, resource provision
  - Location, relocation, substitution
    - Goal: provide UoP with max resources as long as needed
- Interaction (all overhead for HPC)
  - Communication
    - Goal: "max bw, min latency" – but must also match the nature of the interaction (frequency, pattern, robustness)
  - Synchronization
    - Avoid if possible, minimize cost

Computing Platforms

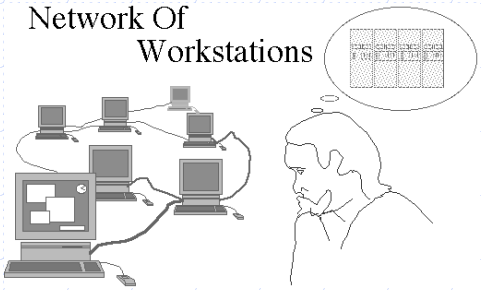
Higher
Efficacy
& Cost

Ideal Grid

Loosely
Coupled/Managed

Tightly
Coupled

Network Of
Workstations



Lower
Efficacy
& Cost

EMORY
UNIVERSITY

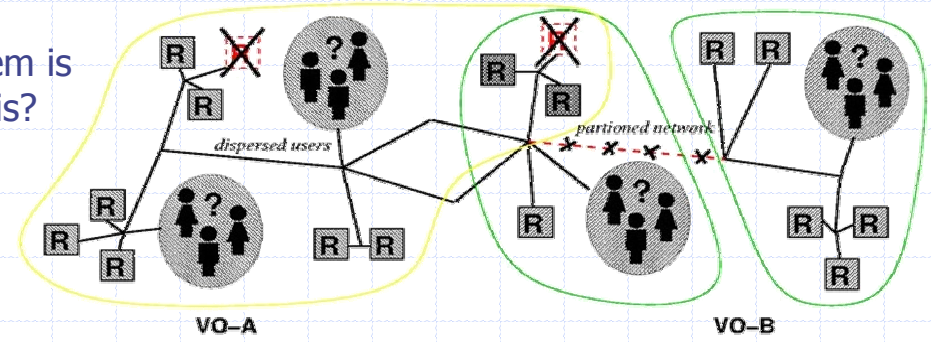
What is a/the Grid ?

- A Grid is NOT
  - The Next generation Internet
  - A new Operating System
  - Just (a) a way to exploit unused cycles (b) a new mode of paralell computing (c) a new mode of P2P networking
- Definitions
  - A paradigm/infrastructure that enables the sharing, selection, & aggregation of geographically distributed resources (computers, software, data(bases), people) [share (virtualized) resources]
  - . . . depending on availability, capability, cost, QoS requirements
  - . . . for solving large-scale problems/applications
  - . . . within virtual organizations [multiple administrative domains]

Grid Functionality

- MAD Resource Sharing
  - Security: authentication, authorization, delegation, etc, etc
  - Management: staging, allocation, co-allocation
  - Scheduling, mapping, steering
- Virtualizing Resources
  - Description, interoperability, access
  - Publishing, discovery and lookup
  - Instantiation, lifetime and state management
- But what about the programming model?

What concurrent programming system is well matched to this?



EMORY UNIVERSITY

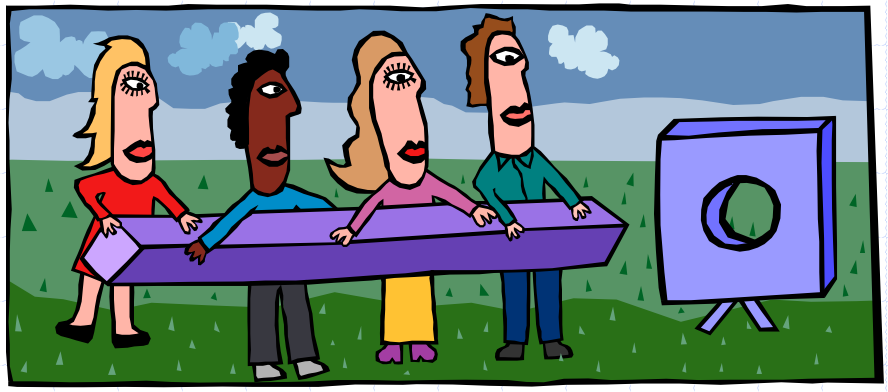
Parallel Computing Today

- Multiscale parallelism
  - Hardware/instruction level
  - Parallelizing compilers & directive-based compilers
  - Explicitly parallel programming (message passing or MTP)
    - MPI standard very comprehensive and popular – primarily MPPs, clusters
    - Works best on homogeneous, regular, symmetric platforms; manual partitioning/parallelization
  - Frameworks/components
    - Composability, reuse, evolvability, interoperability
    - Performance/scalability may not meet expectations
  - One-of-a-kind, research ideas
    - One-of-a-kind, research ideas

Most widespread

Grids and HPC?

- Grid computing
  - Resource sharing across X, Y, Z
    - Geographic distance (latency, variability)
    - Administrative domains (access, security, policies)
    - Heterogeneity (work decomposition vs. diff resources)
- Parallel programming
  - Uniform, regular, homogeneous



Massively parallel programs *Metasystems*

Prognosis?

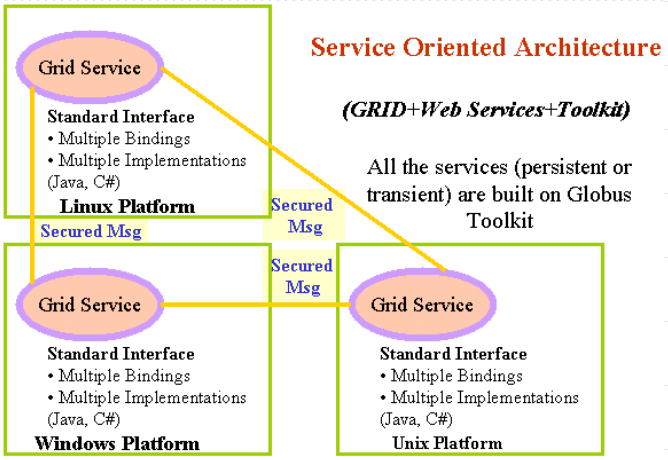
- Grids 2040

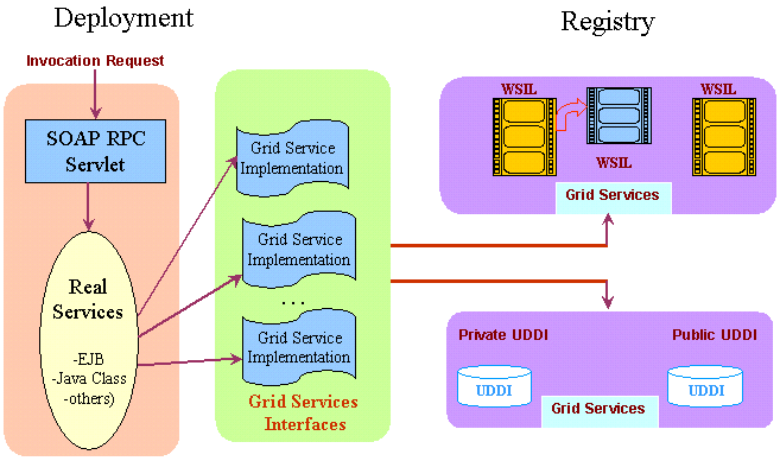
  - Wall-socket computing/information processing ("Grid")

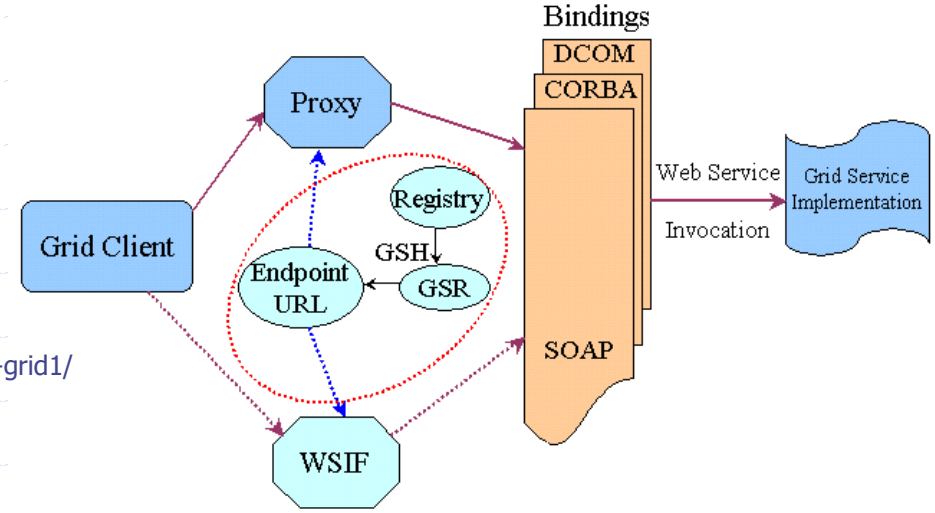
- Grids 2004
  - Service-oriented metacomputing
  - Pragmatic solutions through engineering: facilitate MPI across X, Y, Z boundaries and leave the rest to the user/application
    - Harness and FT-MPI
    - Others: PACX, StaMPI, MPICH-G2, LAM with IMPI

OGSA/OGSI/GT3

- Open Grid Services Architecture/Infrastructure







Developing Grid computing applications
http://www-106.ibm.com/developerworks/webservices/library/ws-grid1/

EMORY UNIVERSITY

GT3 Programming

- Acknowledgements
  - Globus website/related links
  - Sotomayor tutorial (http://www.casa-sotomayor.net/gt3-tutorial)
- Prerequisite background
  - Web service technologies
  - Java, XML, SOAP
  - OGSA and OGSI
  - Grid Services and GT3
- Relationship of various technologies and concepts
  - http://www.casa-sotomayor.net/gt3-tutorial/start/key/ogsa_ogsi.html
- Web services overview
  - http://www.casa-sotomayor.net/gt3-tutorial/start/key/web_services.html
- Grid services, factories, GSH, GSR
  - http://www.casa-sotomayor.net/gt3-tutorial/start/key/grid_service.html

GT3 Architecture

- Layers
  - Grid services (core)
  - Security services
  - Base services
    - Managed job service
    - Index service
    - Remote File Transfer (RFT) service
  - Data services

Writing a GT3 Grid Service

- Define service interface

```
package gt3tutorial.core.first.impl;
public interface Math {
public int add(int a, int b); public int subtract(int a, int b);
public int multiply(int a, int b); public float divide(int a, int b); }
```

- Generate WSDL
  - (Compile Java interface)
  - Apache Axis tool
  - GT3 tool "DecorateWSDL" (add GT3 specific items to WSDL)
- Generate client and server-side stubs
  - GT3 tool "GSDL2Java"
- Implement service

```
public class MathImpl extends GridServiceImpl implements MathPortType
```

  - Details: http://www.casa-sotomayor.net/gt3-utorial/core/first/interface.html

Deploying a GT3 Grid Service

- Deployment Descriptor
  - URL, Java class, other params of Grid Service
  - Name of GS + Base address of GS container == GSH
    - GT3 standalone container http://localhost:8080/ogsa/services
- Compilation
  - Stubs, create JAR
  - Implementation, create JAR
  - Create GAR (stubs.jar, impl.jar, gs.wsdl, deployment descriptor)
- Deployment
  - "ant deploy gridservice.gar"

Invoking a GT3 Grid Service

- Get reference to remote grid service

```
URL GSH = new java.net.URL(args[0]); // Get a reference to the remote web service
MathServiceLocator mathService = new MathServiceLocator();
MathPortType math = mathService.getMathService(GSH);
```

- Invoke service

```
int sum = math.add(a,b);
```

- Operation
  - "globus-start-container"
  - Compile and run client
- Summary
  - http://www.casa-sotomayor.net/gt3-tutorial/core/ant/bigpicture.html

GT3 Factories

- Creation of Grid Service
  - One instance per client (stateful-transient)
  - One instance shared by clients (stateful-nontransient)
  - Service implementation has static/private variables
- Deployment descriptor
  - Substantial changes
  - Example:

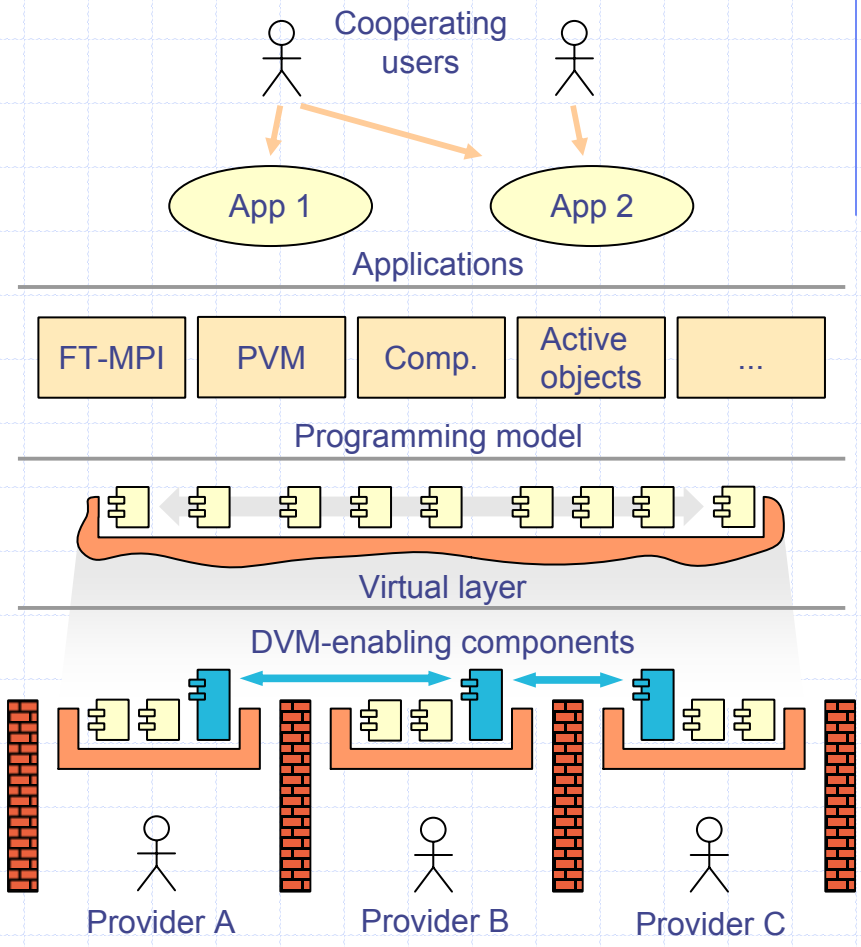
  http://www.casa-sotomayor.net/gt3-utorial/core/grid_services/math_factory.html
- Deploy
  - Start container
  - "ogsi-create-service MathFactoryService(GSH) math"
  - Clients connect to Factory, create service, invoke service

Advanced Core Topics

- GWSDL
  - One instance per client (stateful-transient)
- Operation Providers
  - Substantial changes
- Lifecycle Management
  - Start container
- Service Data
- Notifications

The Harness II Project

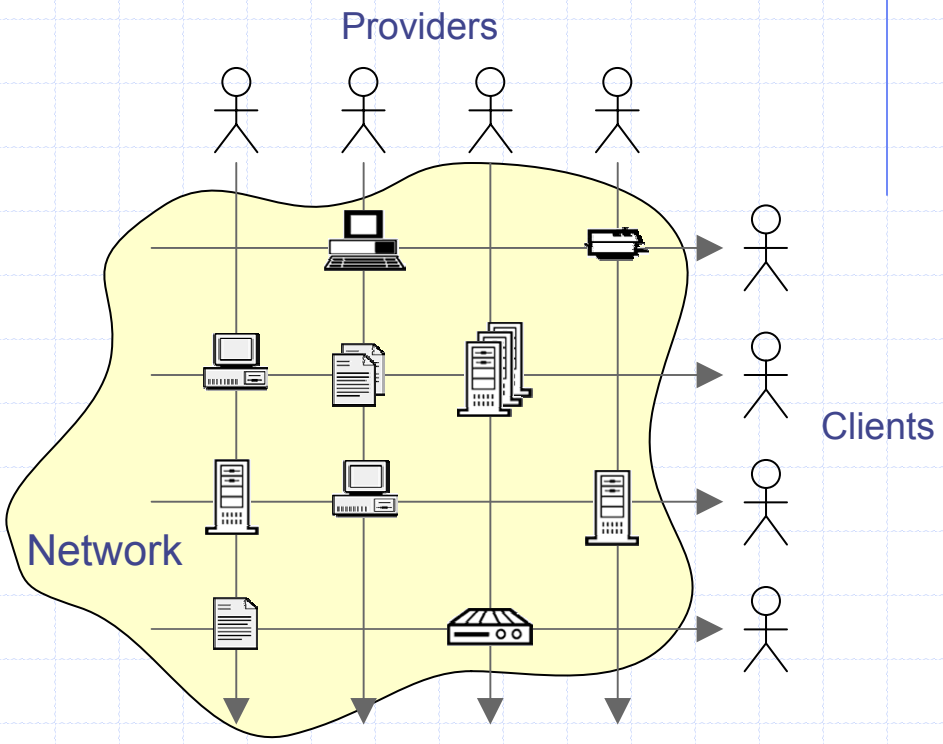
- Joint between Emory, UTK, and ORNL
- Cooperative Fault Tolerant Distributed Computing
- Programming framework: Fault tolerant MPI, lightweight components, service oriented
- Flexible, lightweight, middleware
- Hosting layer: H2O substrate
  - Stateless, lightweight



EMORY UNIVERSITY

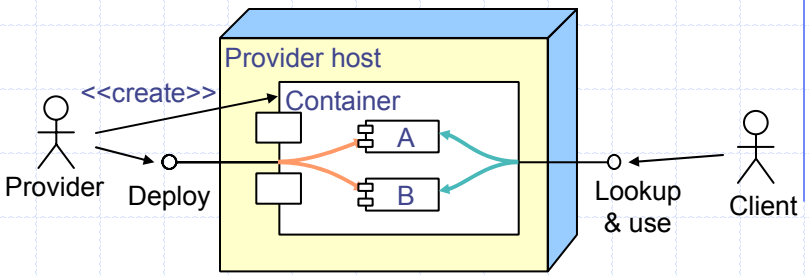
H2O Abstraction

- Providers owning resources
- They independently make them available over the network
- Clients discover, locate, and utilize resources
- Resource sharing occurs between single provider and single client
  - Relationships may be tailored as appropriate
  - Including identity formats, resource allocation, compensation agreements
- Clients can themselves be providers
  - Cascading pairwise relationships may be formed

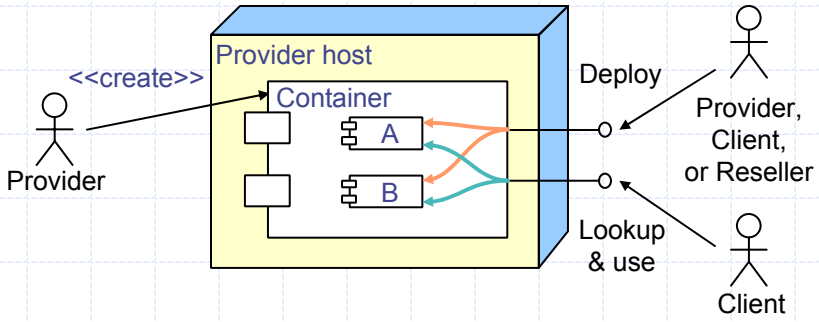


H2O Framework

- Resources provided as services
  - Service = active software component exposing functionality of the resource
  - May represent „added value”
  - Run within a provider's container (execution context)
  - May be deployed by any authorized party: provider, client, or third-party reseller
- Decoupling
  - Providers/providers/clients



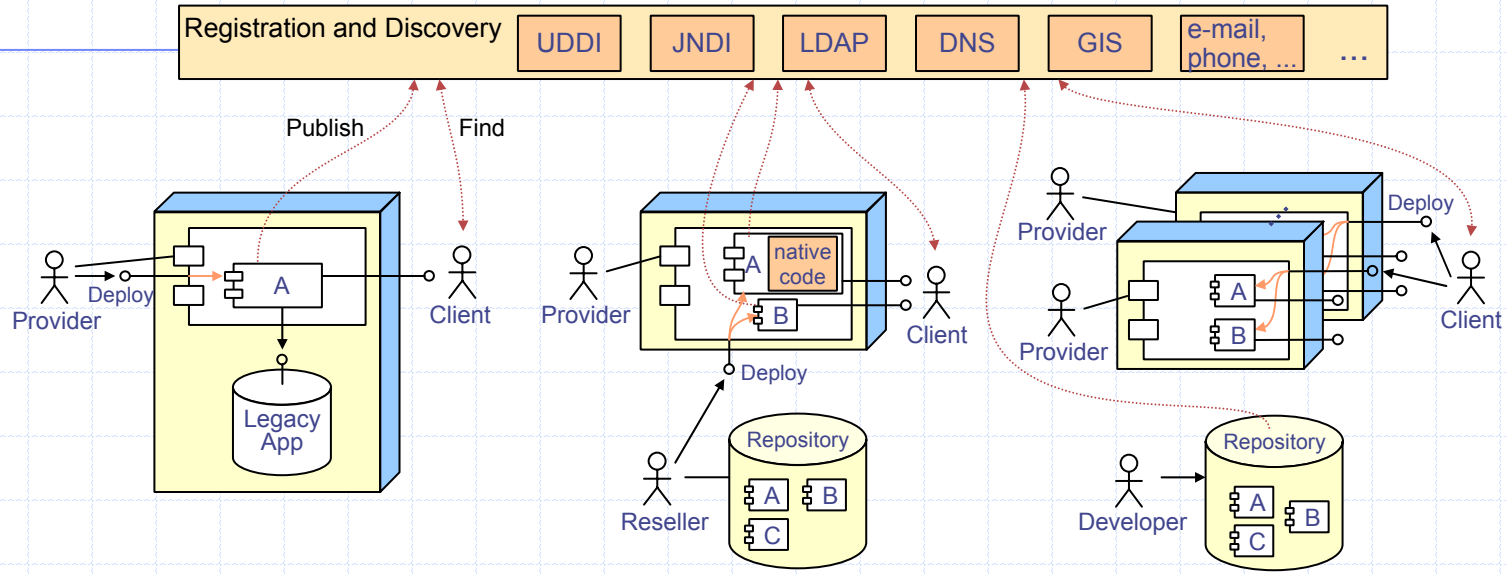
Traditional model



Proposed model

EMORY UNIVERSITY

Example usage scenarios



- Resource = legacy application
- Provider deploys the service
- Provider stores the information about the service in a registry
- Client discovers the service
- Client accesses legacy application through the service

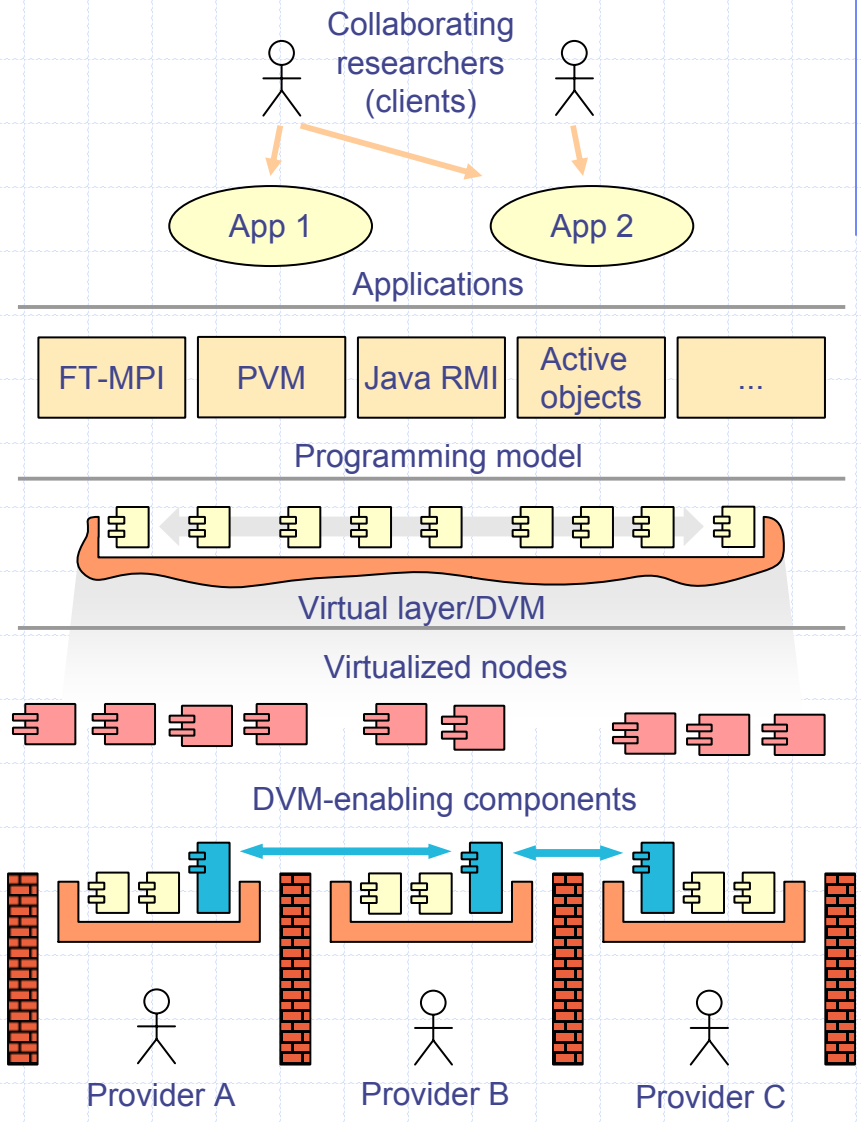
- Resource = computational service
- Reseller deploys software component into provider's container
- Reseller notifies the client about the offered computational service
- Client utilizes the service

- Resource = raw CPU power
- Client gathers application components
- Client deploys components into providers' containers
- Client executes distributed application utilizing providers' CPU power

EMORY UNIVERSITY

Metacomputing

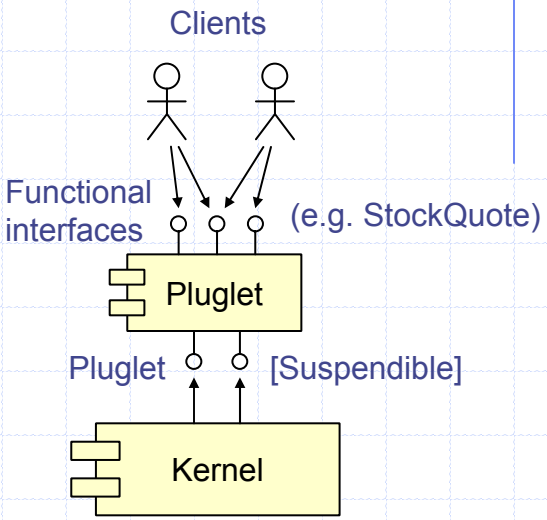
- Clients aggregate resources
  - that may belong to them
  - or may come from independent providers
- Clients upload components
  - DVM-enabling components communicate with each other and provide an illusion of Distributed Virtual Machine
- Specific pluglets provide concrete programming environments
  - PVM and Fault Tolerant MPI components currently being adapted to H2O
- Distributed state managed by DVM-enabling components alone
  - statelessness at the provider level
  - resources decoupled and independent



Model and Implementation

- H2O nomenclature
  - container = *kernel*
  - component = *pluglet*
- Object-oriented model, Java-based prototype implementation
- Pluglet = remotely accessible object
  - Must implement *Pluglet* interface, may implement *Suspendible* interface
  - Used by kernel to signal/trigger pluglet state changes

```
Interface StockQuote {
    double getStockQuote();
}
```

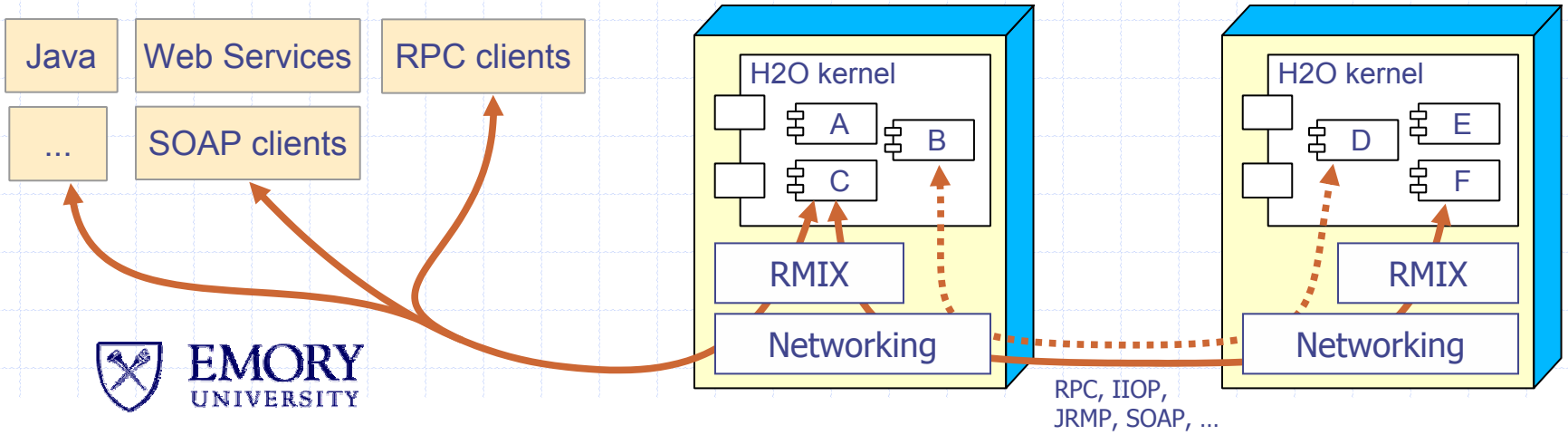



```
Interface Pluglet {
    void init(ExecutionContext cxt);
    void start();
    void stop();
    void destroy();
}
```

```
Interface Suspendible {
    void suspend();
    void resume();
}
```

Interoperability – the RMIX layer

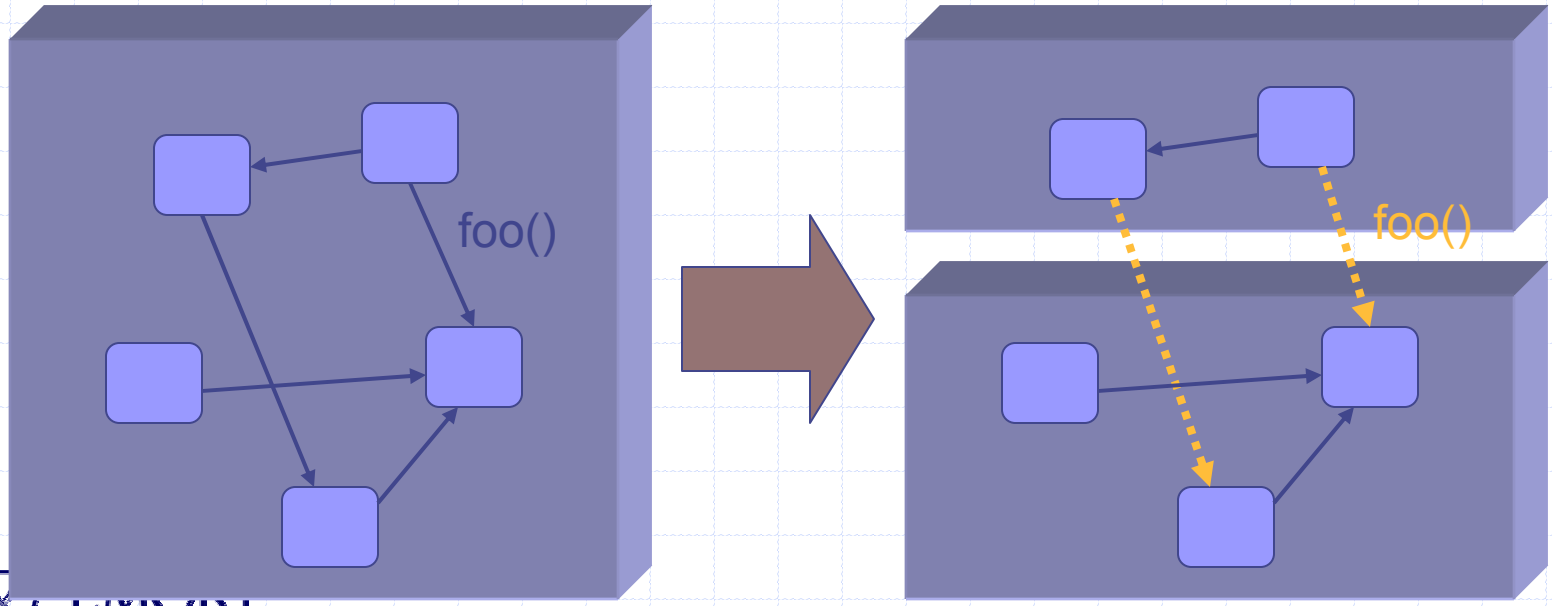
- H2O built on top of RMIX communication substrate
  - Provides flexible p2p communication layer for H2O applications
- Enable various message layer protocols within a single, provider-based framework library
  - Adopting common RMI semantics
- Enable high performance and interoperability
  - Easy porting between protocols, dynamic protocol negotiation
- Offer flexible communication model, but retain RMI simplicity
  - Asynchronous and one-way calls



EMORY UNIVERSITY

RMI: Java-based Distributed Computing

- Java object model extended for multiple, distributed VMs
- Simplicity, easy to distribute existing code
- RPC paradigm with the benefits of the object model
  - serializable remote references, callbacks, dynamic export, ...



Issue #1: Interoperability

- Java Remote Method Protocol (JRMP)
  - Sophisticated and highly Java-specific
  - Disallows connectivity with non-Java services
- Alternatives: RMI-IIOP, JAX-RPC, ...
  - Independent solutions using different wire protocols (IIOP, SOAP, ...)
  - Differences in the programming model
  - Effect: difficult to port code between libraries
  - And what about multiprotocol applications?...

Issue #2: Performance

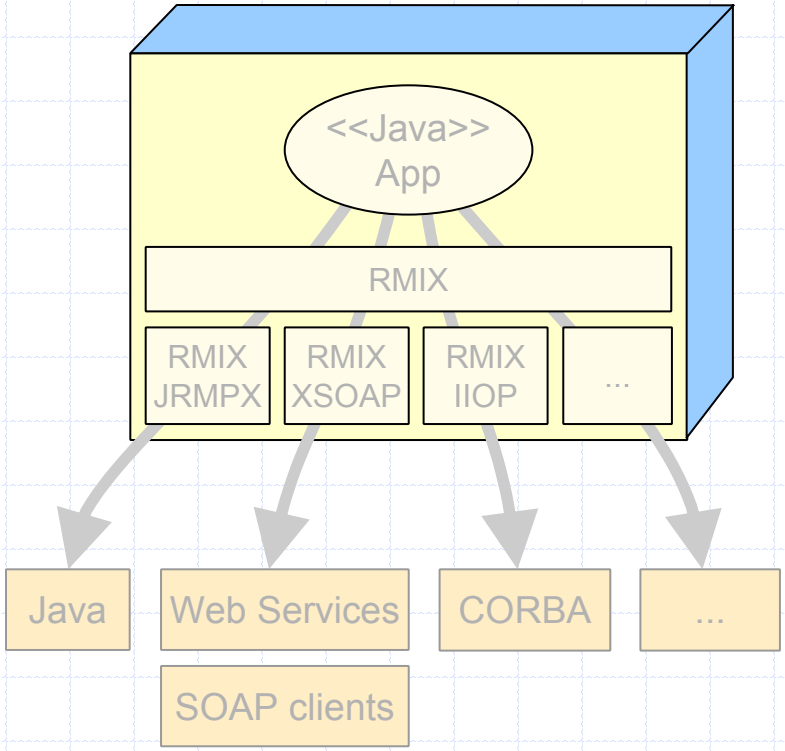
- Early implementations of RMI (Java 1.1)
  - Poor communication performance
  - Not targeted for high-bandwidth networks
- Current status (2003, Java 1.4)
  - Significant improvements, especially for primitive types and arrays thereof
  - However, impossible to optimize object serialization without breaking compatibility
- Solutions
  - Alternative RMI implementations (KaRMI, Manta, ...)
  - Not interoperable, useful only in tightly-coupled applications

Issue #3: Stiff programming model

- Security in loosely-coupled systems
  - Lack of primitives enabling authentication (e.g. ability to create independent server references to the same object for different client sessions)
  - No message-level security features (encryption, MAC, replay detection)
- Communication paradigm
  - No support for asynchronous or one-way calls
  - Some attempts: ARMI, ...

RMIX – design goals

- Enable various RMI solutions within a single, provider-based framework library
- Enable high-performance communication
- Enable interoperability with heterogeneous services
- Offer more flexible communication model, but retain simplicity
- Easy deployment



RMIX – the programming model

- Object model inherited from Java RMI
- Remote proxies
- Remote interfaces
- Remote methods
- Remote exceptions
- Parameters passed by value

RMIX – „Hello, World!”

```
public interface Hello
       extends java.rmi.Remote
{
  String hello(String msg)
    throws java.rmi.RemoteException;
}
```

Hello.java

```
import edu.emory.mathcs.rmix.Naming;

public class HelloSrv implements Hello
{
  public String hello(String msg) {
    return "Hello, " + msg;
  }

  public static void main(String[] args)
  {
    HelloSrv hello = new HelloSrv();
    Naming.rebind("hello", hello);
  }
}
```

HelloSrv.java

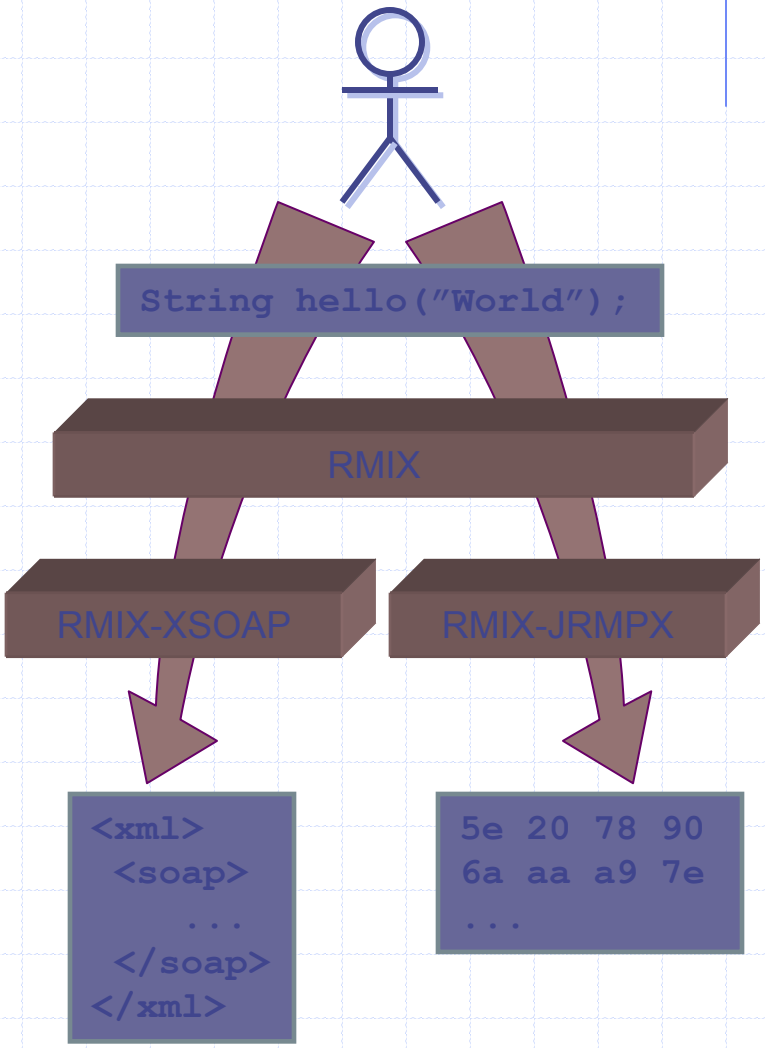
```
import edu.emory.mathcs.rmix.Naming;

public class HelloClient
{
  public static void main(String[] args)
  {
    String srv = args[0];
    String name = "//" + srv + "/hello";
    Hello hello =
      (Hello)Naming.lookup(name);
    String response = hello.hello("World");
    System.out.println(response);
  }
}
```

HelloClient.java

RMIX – data marshalling

- Remote transmission
  - Calls, their parameters, return values, and exceptions are sent over the network
  - Serial form depends on the protocol
  - Marshalling by pluggable provider modules
- Framework defines only the *semantics*
  - Serializable types of parameters, return values and exceptions
  - What the application may depend on?
  - What the provider modules must support?



RMIX – data marshalling (cont.)

- Goal: accomodate even very simple protocols
  - Examples: Sun-RPC / XDR
  - Limited number of serializable types
  - No distributed garbage collection, polymorphism, object graphs
- Minimal required semantics must then be very narrow
  - Primitive types, strings, arrays thereof
  - Bean-like classes
  - Support for more sophisticated types is the provider „Quality of Implementation”
- Runtime „capability query” mechanisms considered
  - Accomodate applications that require more than bare minimum, but without binding to a specific provider
  - Let protocol providers advertise their marshalling capabilities
  - The application may then query RMIX for providers that fulfill given requirements

RMIX – protocol switching

- Goal: program can dynamically switch communication protocol
  - Run-time protocol negotiation
- Prerequisite: any provider must be able to marshall stubs of any other provider
  - But we do not want inter-provider dependencies!
- Solution: easily serializable *unified reference format*
  - Every provider is able to unify (pre-serialize) its own stubs
  - Then, providers need to deal only with unified stubs

```
public class HelloImpl
       implements Hello
{
  Hello switch(String newProt) {
    Hello newStub = (Hello)
     Rmix.export(newProt, this)
     .bind();
    return newStub;
  }
}
```

H2O Operational Overview

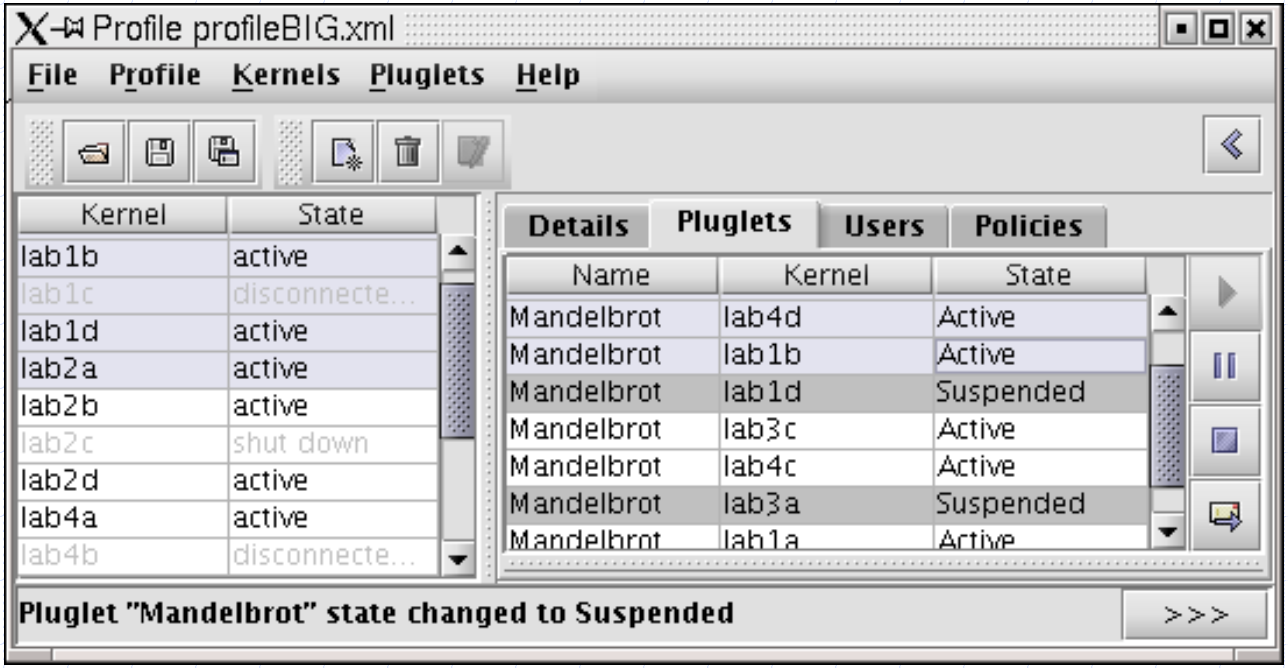
- Providers start H2O kernel on individual machines
  - Kernel profile created/updated

```
<kernelEntry>
 <name>my_red_kernel</name>
 <RemoteRef protocol='SOAP 1.1/RMIX-XSOAP' binding='1.0R'
   interfaces='edu.emory.mathcs.h2o.server.GateKeeperSrv'
   guid='11d1def534ea1be0:1b26af32aa43251b:0'
   location='http://170.140.150.185:34787/11d1def534ea1be0:1b26af32aa
   43251b:2'/>
 <startup method='ssh' autostart='true'>
   <parameter name="user" value="neo"/>
   <parameter name="command" value="/home/neo/h2o/bin/h2o-kernel"/>
   <parameter name="host" value="matrix.mathcs.emory.edu"/>
 </startup>
</kernelEntry>
```

EMORY UNIVERSITY

H2O -- GUI

- Application to help H2O users manage kernels they use
  - load or kill a pluglet, suspend or resume
  - check state of a kernel/pluglet
  - start or shutdown a kernel



EMORY UNIVERSITY

H2O Security – Authorization

- Global H2O kernel policy
  - XML-based policy file
  - Permissions granted to authenticated end-users (JAAS principals) and/or to signed and authenticated code
  - Temporal restrictions

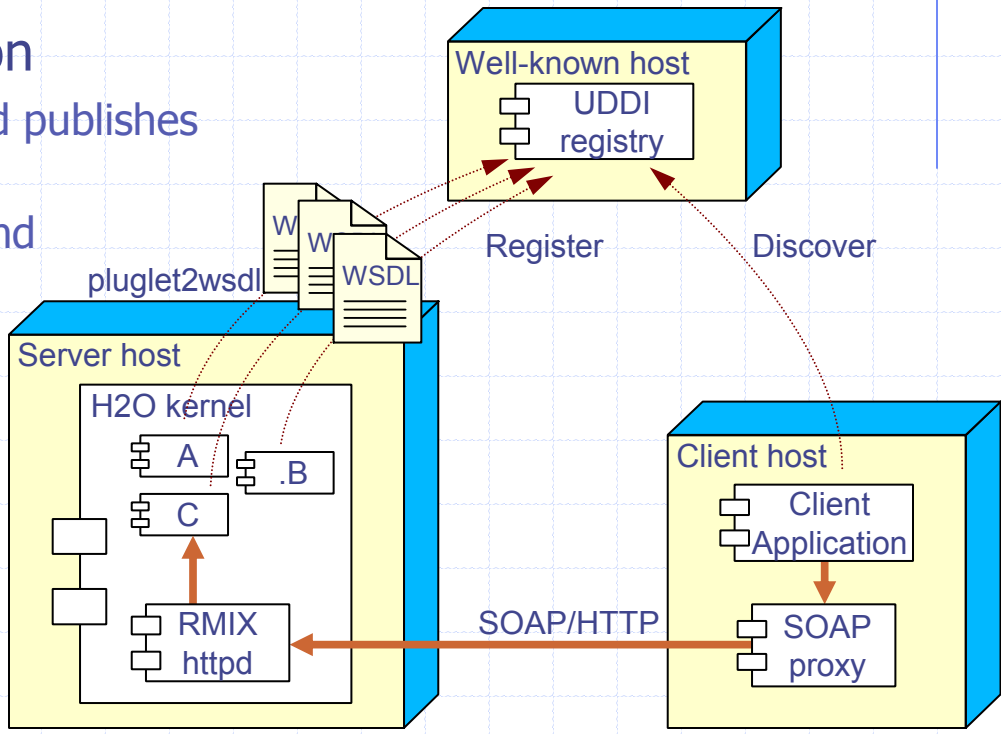
```
<?xml version="1.0"?><!DOCTYPE policy SYSTEM "XMLPolicy.dtd">
<policy>
  <grant codebase="http://trusted.host.net/classes/*" signedBy="trustedPlugletSource">
    <valid from="10/25/2002" to="11/25/2002" pattern="*:8.00-9.00;*:10.00-12.00"/>
    <permission classname=„java.lang.RuntimePermission" target="getClassLoader"/>
  </grant>
  <grant>
    <valid from="10/9/2002" to="11/8/2003" pattern="MTW:*"/>
    <principal classname="edu.emory.mathcs.h2o.SimplePrincipal" name="Alice"/>
    <permission classname="java.net.SocketPermissions" target="*" actions="connect"/>
    <permission classname="java.lang.PropertyPermission" target="*" actions="read"/>
  </grant>
</policy>
```

H2O Security (contd)

- Other Authorizations
  - H2O-specific security permissions and security checks e.g. to load pluglets, change their state, etc.
  - Pluglet deployer policy: Who can do what on pluglets I load?
- Authentication
  - Multiple actors need to authenticate each other
    - Providers, deployers, end-users, software suppliers
  - End-user authentication by providers (rest in progress)
    - Allows multiple credentials and pluggable authentication technologies (user/password, X.509 based remote auth)
- H2O Platform: Result = Virtual Machine
  - Configurable as required by authorized entities

H2O Programming and API

- Connection and authentication
  - (Provider instantiates kernel and publishes reference)
  - User obtains kernel reference and connects to it
  - Kernel authenticates the client (opt. client auths. kernel)
  - If successful, client obtains *kernel context*
- Deploying services
  - Client (or TPR) may use *kernel context* to upload pluglets
  - Need to specify: location of binaries (URL path), class name, optionally: additional meta-information
- Invoking services
  - Client invokes functional interface methods



Basic H2O Programming

- Like RMI
  - Write interface

```
public interface Hello extends Remote {
String sayHello() throws RemoteException;}
```

  - Write implementation

```
public class HelloImpl implements Hello, Pluglet {
  public HelloImpl();
  public void init(plugletContext pc, Object[] params);
  public void start();
  public void stop();
  public void destroy();
  public String sayHello() { return "Hello World!";}
}
```

Basic H2O Programming (contd)

- Write Client (invocation)

```
public static class HelloClient {
  String kernel_ref = // Get kernel reference
  URL plugletCodeBase = // Get pluglet code base
  KernelContext kc = H2O.login(kernel_ref);
  PlugletContext pc = kc.load([Hello Pluglet]);
  Hello obj = (Hello)pc.getPluglet(TRANSPORT_PROVIDER);
  // Can be RMIX-XSOAP, RMIX-JRMPX, RMIX-JRPCX etc
  String message = obj.sayHello();
}
```

- Just like RMI except multiple transports, language indep.

RMIX-RPCX: Java obj. -> RPC -> C stub

```
Hello server = new HelloImpl();
RpcxServerRef ref = Rmix.export(server);
RPCGenFormatter.generate("hello.x", ref);
```

Java server

RPCGenFormatter

```
(..)
const PROGRAM_PORT = 51097
program PROG40000000 {
version VERSe6397446 {
    java_lang_String_ReplyRPCUnion
    hello (
    java_lang_String_ParamRPCUnion
    p0 ) = 405467934;
} = 432442298;
} = 1073741824;
```

Rpcgen .x file

rpcgen tool

```
void prog40000000_405467934(char *host)
    {
(…)
sockaddr_in *address = new sockaddr_in()
address->host = host;
address->port = PROGRAM_PORT;
CLIENT *clnt = clnttcp_create (address,
    PROG40000000, VERSe6397446,
    RPC_ANYSOCKET, 0, 0);
(…)
result_1 =
    hello_405467934(&hello_405467934_arg,
    clnt);
}
```

RPC C-client code

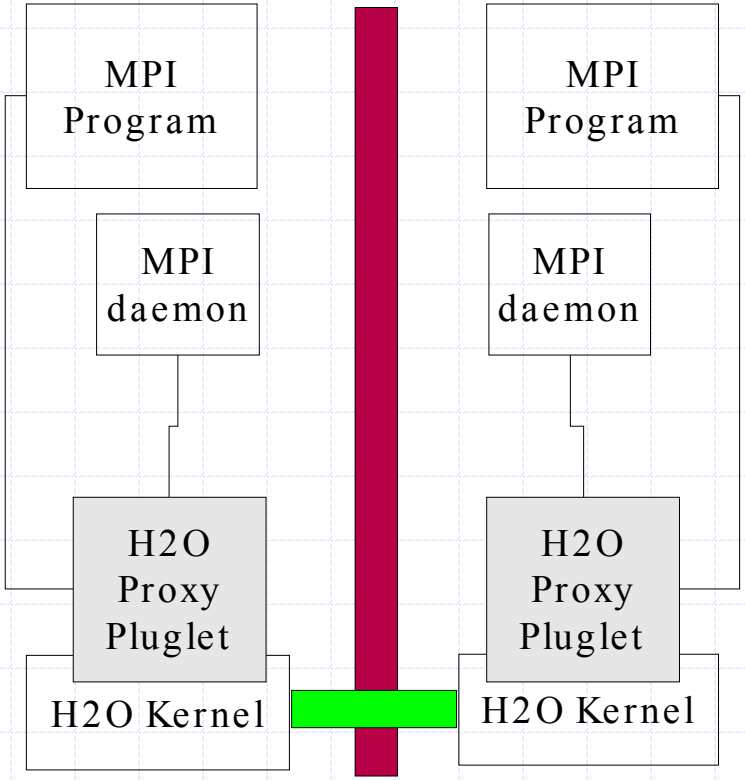
EMORY UNIVERSITY

H2O Programming and Operation

- Start kernel
- Kernel reference printed or output
- Login to each kernel
  - May be done by application
- Load appropriate pluglets on each kernel
- Start pluglets
- Monitor with GUI

MPI on Metasystems

- Many scenarios:
  - Firewalls
  - Non-routable NW's
  - Heterogeneous CoCs
  - Grid-enabled
- MPI across firewalls
  - Replace all runtime connections by tunneling all MPI communication through H2O channels



EMORY UNIVERSITY

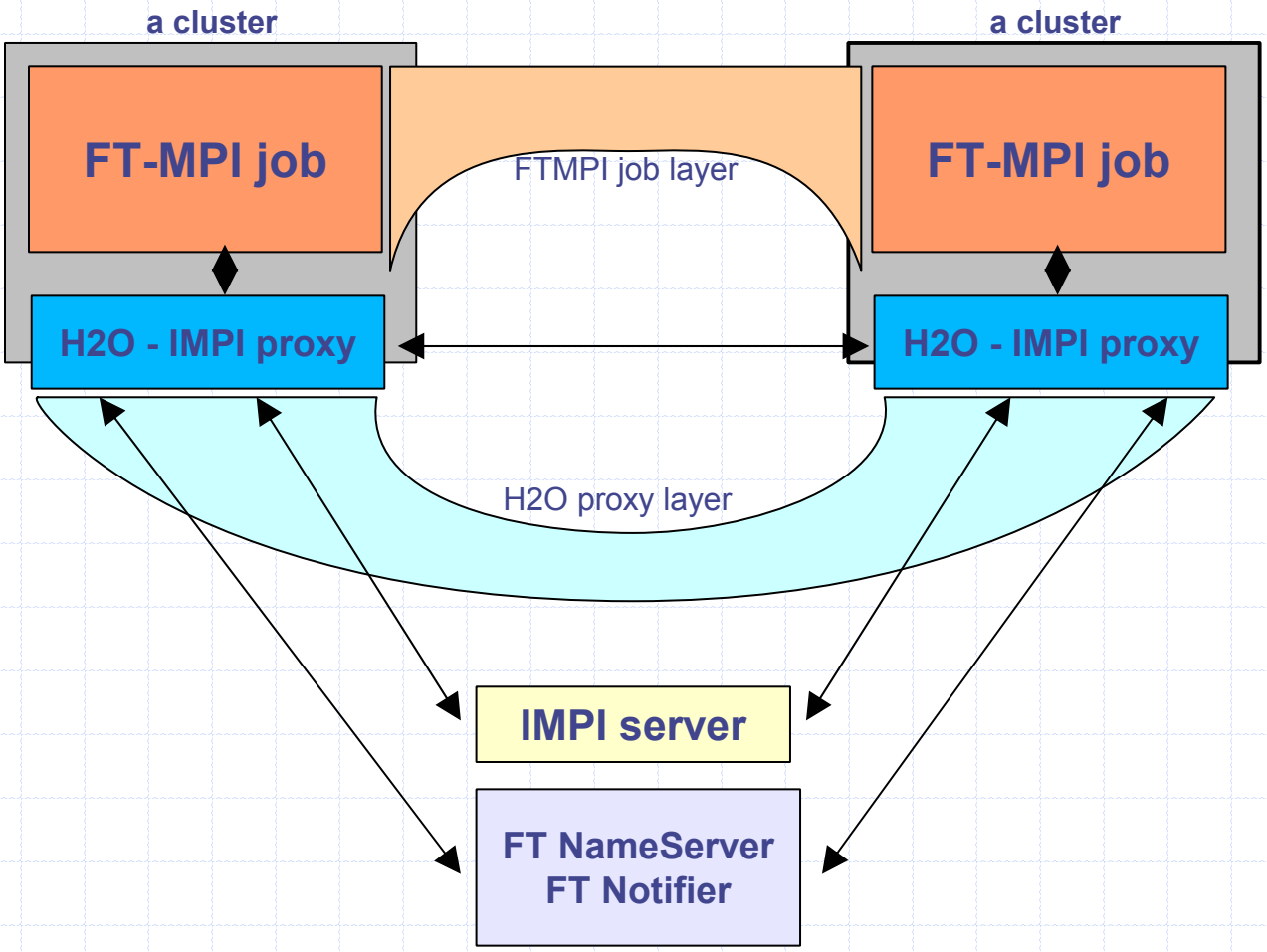
MPI on Metasystems: FTMPI/H2O/IMPI

	MPICH	LAM/MPI	PACX-MPI	Madeleine III	Stampi	LA-MPI	FT-MPI	UTK - Emory
heterogeneity	✓	✓?⅛	✓	✓?	✓	✓	✓	✓
non-routable & non-IP networks	✗	✗	✓	✗	✓	✓	✗	✓
fault tolerance	✗	✗	✗	✗	✗	✓½	✓	✓

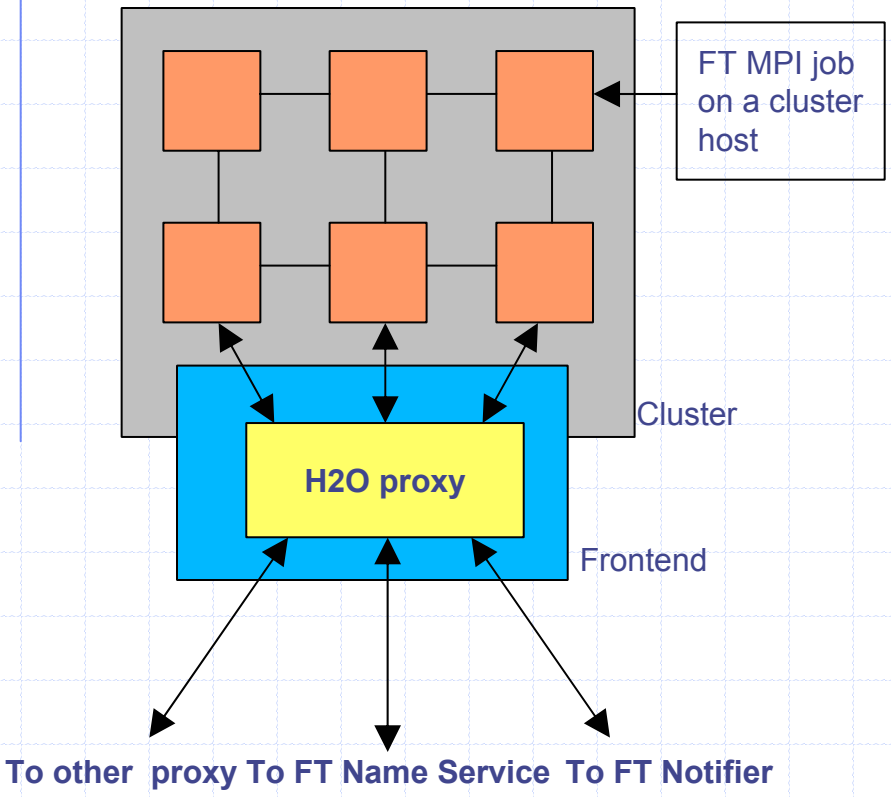
1. **Heterogeneity** – machine architectures, operating systems
2. **Non-routable networks** – 192.168.*.*, Myrinet netwoks …
3. **Fault tolerance** – network failures, process failures

EMORY UNIVERSITY

FT-MPI/H2O/IMPI



FT-MPI/H2O/IMPI Design



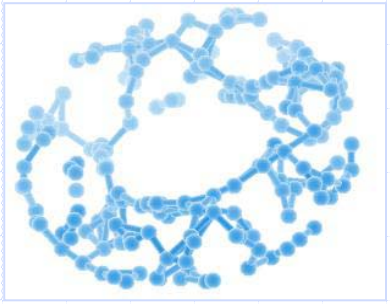
- Processes for intra-cluster communication use FTMPI
- Inter-cluster communication takes place through the proxy
- All messages to the name service and to the FT-notifier are also sent through the proxy using dedicated ports

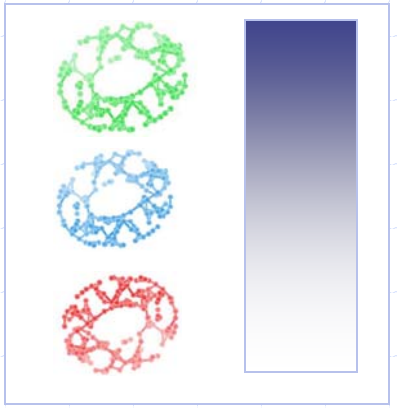
Paradigm Frameworks

- "Structured programming"
  - Scaffolding provided by system
  - User supplies domain-specific functionality
  - Examples: Master-Slave, Genetic Algorithms, Branch-and-Bound
- H2O Methodology
  - Paradigm pluglet supplied (may be enhanced by deployer)
  - End-user implements specified interface, initializes
  - System handles distribution, communication, synchronization
- Example:
  - Evolutionary algorithm – nanotechnology application

Paradigm Frameworks Example

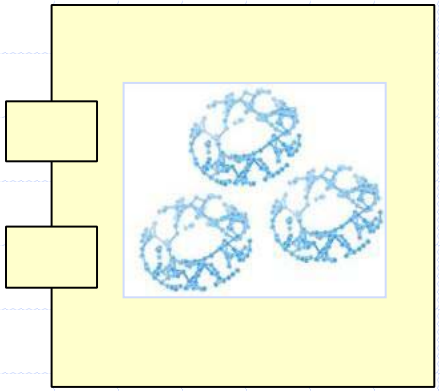
- Distributing Evolutionary Algorithms
- Structure
  - entity representing a candidate solution
- Population of structures
  - set of **n** structures
- Ranking of structures
  - structures in populations ordered by given fitness function

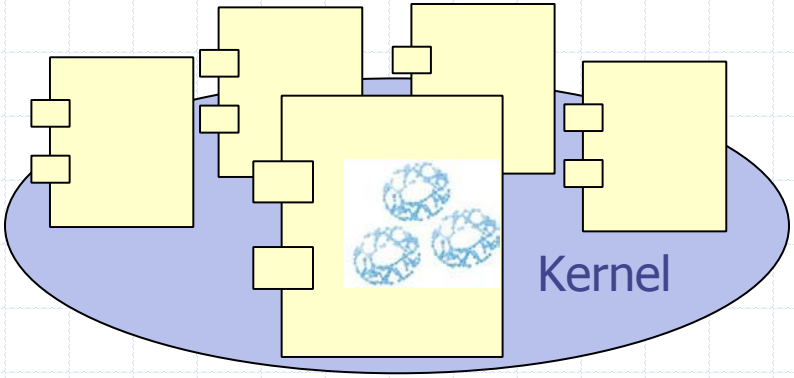




Methodology

- Population Pluglet
  - a container for growing populations within the H2O framework
  - Provided by H2O (or enhanced)
- Provider/Deployer
  - Instantiate kernel
  - Load Population pluglet





Methodology (contd)

- User's (or developer's) job
  - to write a class implementing Growable interface
  - to publish a jar file containing this class (optionally)

```
interface Growable {
  double solveFitnessValue();
  void disturb(double percent);
  void crossOver(Growable g1, Growable g2, List p);
  Object clone();
  void print();
}
```

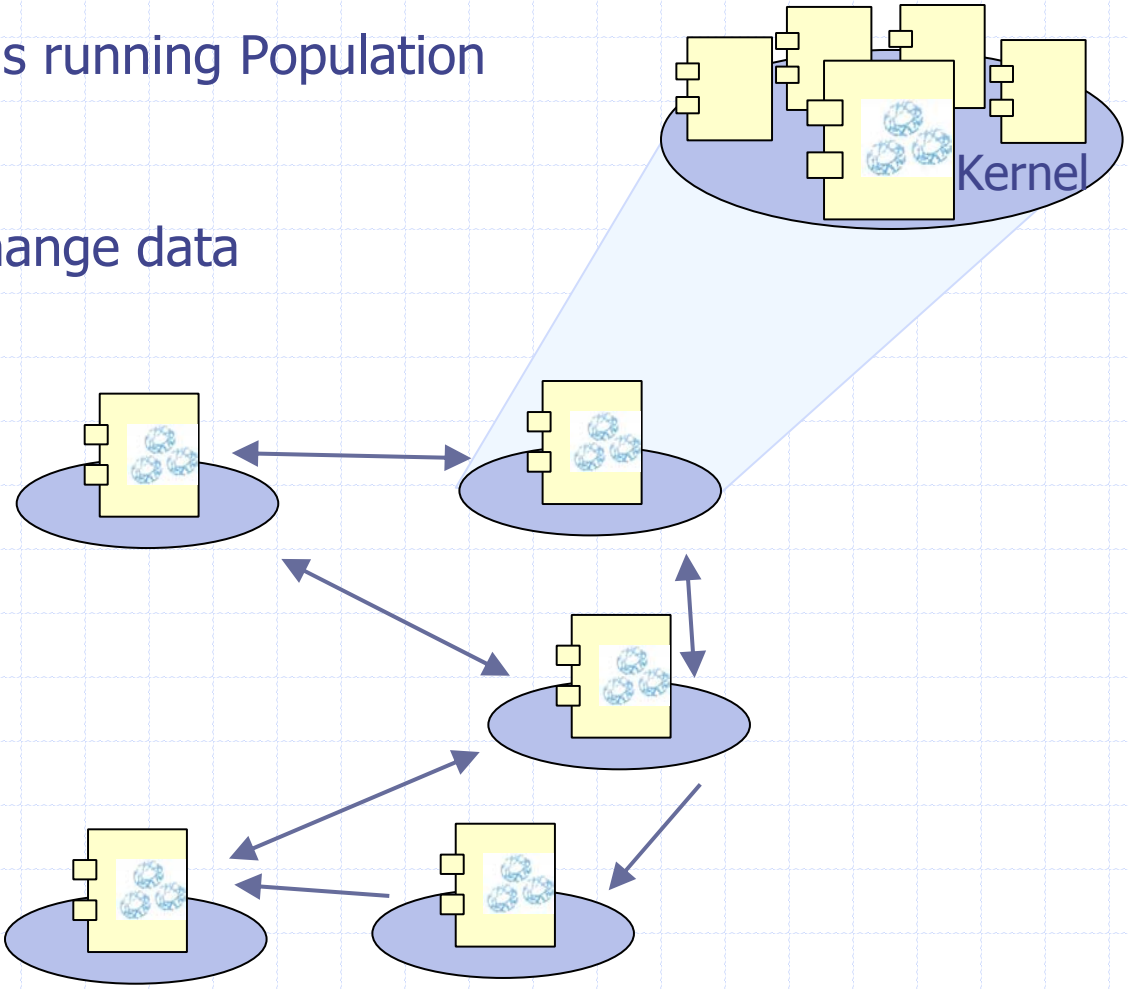
Execution of DEVs

- Controlling population
  - user logs in to H2O kernel containing Population Pluglet,
  - initializes population specifying its size and the name of the jar containing the class that implements Growable interface
  - connects multiple Population Pluglets in desired topology
  - starts and stops population
  - gains results

```
interface PopulationManager extends Remote {
    void initializePopulation(className, count);
    void startPopulation();
    void stopPopulation();
    Growable getBest();
    void setBridge(anotherPPluglet);
}
```

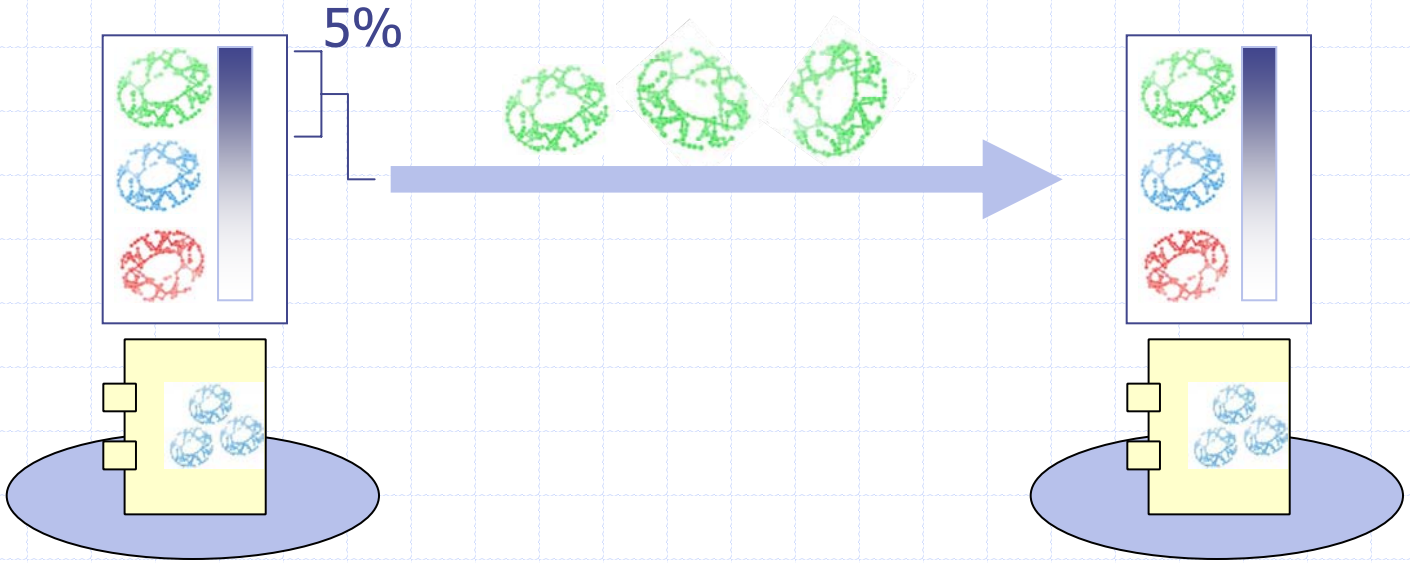
Execution of DEV's (contd)

- Islands
  - many kernels running Population Pluglets
  - Islands exchange data



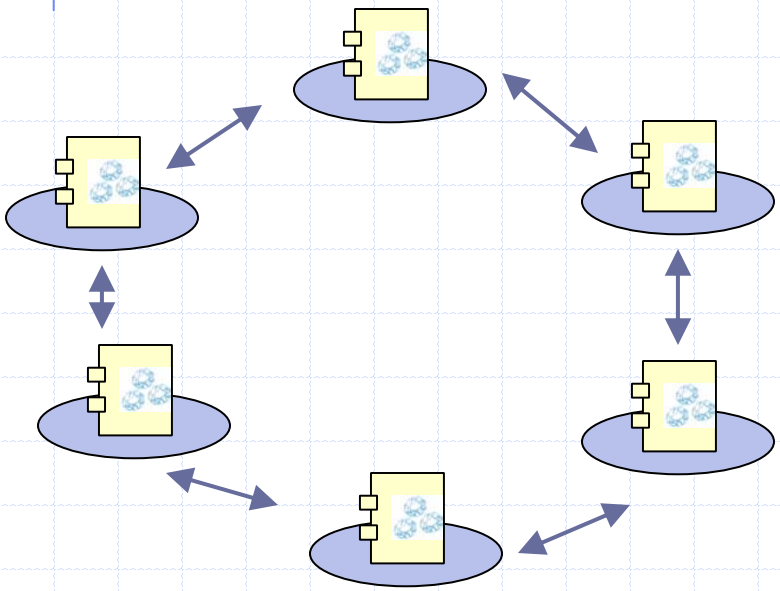
Interaction

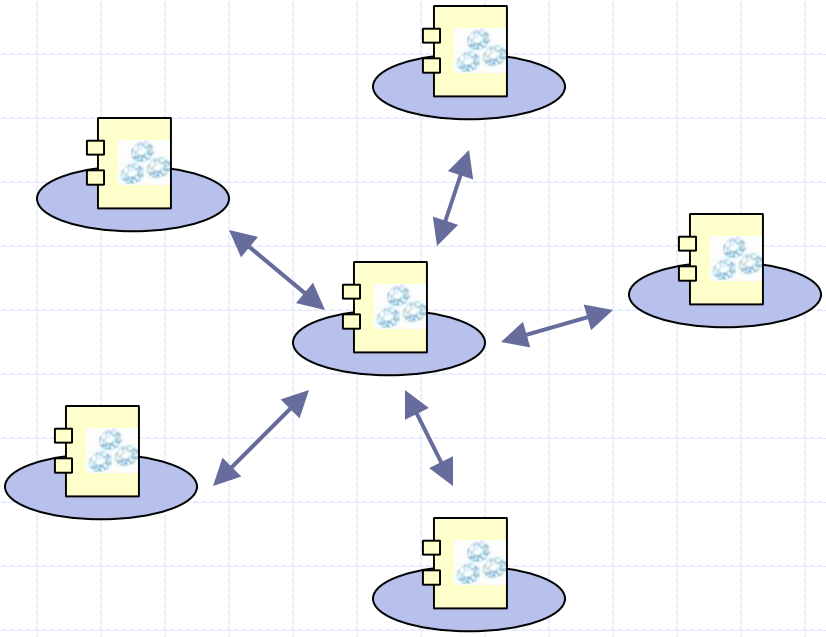
- Data flow between Islands
  - best structures are copied (migrate) to other Island.



Interaction (contd)

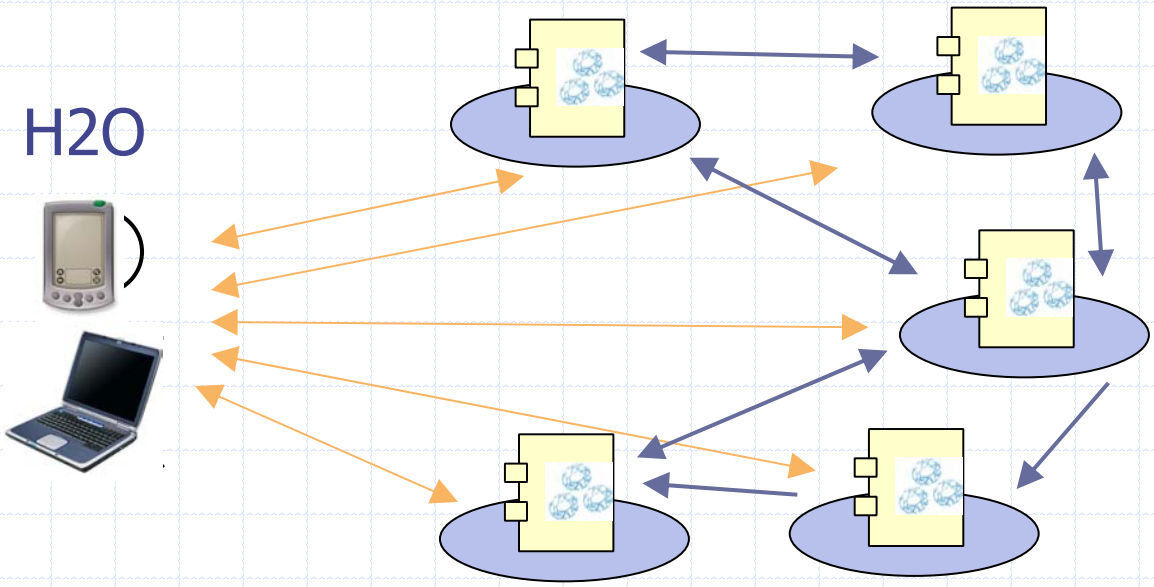
- User defined topologies
  - User specifies the topology of connected Islands.





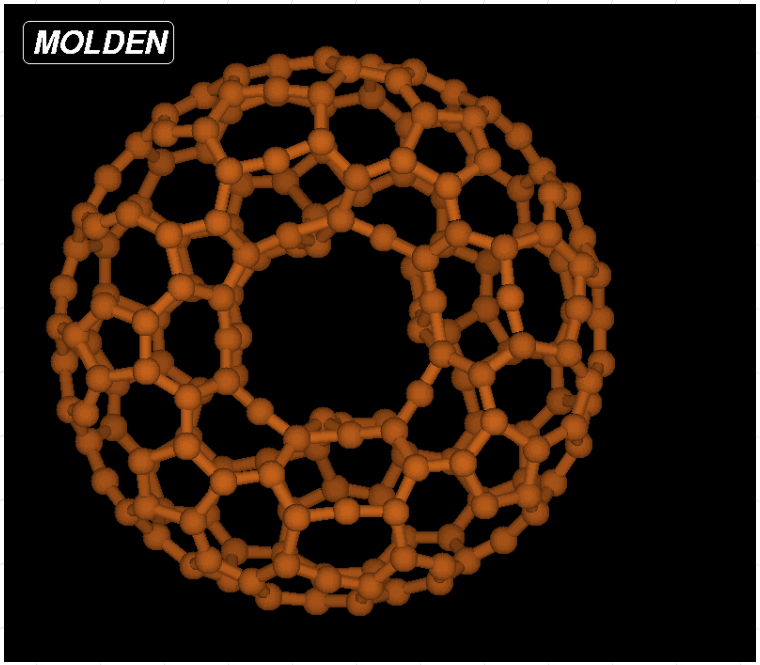
Control and Steering

- Manager
  - connects to pluglets to gain results and manage populations



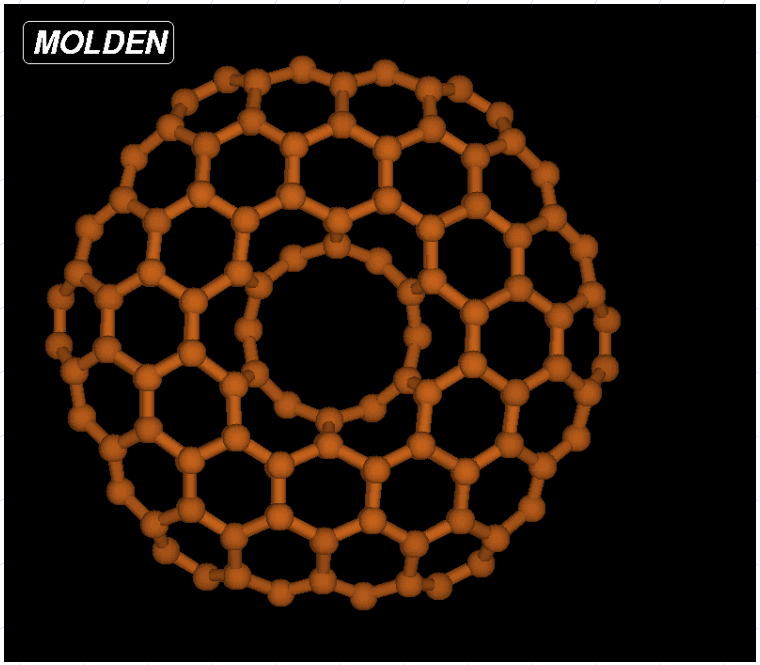
Nanotechnology Simulation

- Our best result



- 1276 eV

- The solution believed to be perfect



- 1337 eV

Summary and Continuation to FT-MPI . . .

- Parallel programming and HPC
  - Traditional model assumes regularity, homogeneity, single AD, exclusivity
  - Suitability for metacomputing ("Grid") environments ?
- Solutions
  - True "Metacomputing" model
    - Service interface to HPC components, frameworks
    - Workflow composition
  - Force fit MPI or similar
    - Transfer (partial) onus to applications
    - Provide engineering solutions (firewalls, heterogeneity, MC)
  - Combining several of these ideas . . .
    - FT-MPI (Edgar Gabriel) . . .