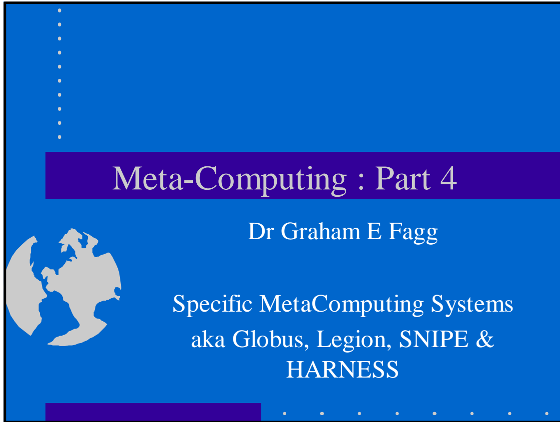
Meta-Computing : Part 4



Dr Graham E Fagg

Specific MetaComputing Systems aka Globus, Legion, SNIPE & HARNESS

Overview

- Cover the most common Meta-Computer systems currently on offer and a few that are/were in the pipe.

Meta-Computing Projects

- Legion - University of Virginia
  - Object based
- Wide Area Metacomputer Manager (WAMM) – Pisa, Italy
  - Grow from experiments with PVM
- Wide Area Network Environment (WANE) -Florida State University
- WebFlow and MetaWeb - Syracuse University
- DISCWorld - University of Adelaide / University of Bangor
- Globus - Argonne National Lab., ISI, USC
  - Tool kit
- SNIPE - UTK
- HARNESS - UTK/ORNL/Emory
- Globe - Virje University in Amsterdam.
  - See http://www.cs.vu.nl/~steen/globe/

Legion

- Legion is an object-based, meta-systems software project at the University of Virginia. From the project's beginning in late 1993, the Legion Research Group`s goal has been a highly useable, efficient, and scalable system founded on solid principles. We have been guided by our own work in object-oriented parallel processing, distributed computing, and security, as well as by decades of research in distributed computing systems. Our system addresses key issues such as scalability, programming ease, fault tolerance, security, site autonomy, etc. Legion is designed to support large degrees of parallelism in application code and manage the complexities of the physical system for the user. The first public release was made at Supercomputing '97, San Jose, California, on November 17, 1997.

Legion

- Legion, an object-based metasystems software project designed to build a system of millions of hosts and trillions of objects, tied together with high-speed links.
- Users working on their home machines have the illusion of working on a single computer, with access to all kinds of data and physical resources, such as digital libraries, physical simulations, cameras, linear accelerators, and video streams.
- Groups of users can construct shared virtual work spaces, to collaborate research and exchange information.
- Legion supports this abstraction with transparent scheduling, data management, fault tolerance, site autonomy, and a wide range of security options.

Legion

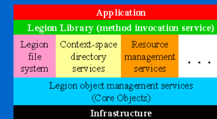
- Legion is an open system, designed to encourage third party development of new or updated applications, run-time library implementations, and core components.

Legions Design

- Legion sits on top of the user's operating system, negotiating between the computer's resources and whichever resources or applications are required.
- Legion handles resources scheduling and security issues so that the user isn't bogged down with time-consuming negotiations with outside systems and system administrators.
- Legion offers a user-controlled naming system called *context space*, so that users can easily track and use objects in farflung systems.
- Users can also run applications written in multiple languages, since Legion supports interoperability between objects written in multiple languages.

Legions Design

- Legion is built from a number of basic components, in the common layer fashion.
- Note that there is a core object known as the Legion Class. This is the root of all .. Legion!



Legion Design

- Philosophy
  - Legion users will require a wide range of services in many different dimensions, including security, performance, and functionality. No single policy or static set of policies will satisfy every user, so, whenever possible, users must be able to decide what trade-offs are necessary and desirable. Several characteristics of the Legion architecture reflect and support this philosophy.

    ? ***Everything is an object:*** The Legion system will consist of a variety of hardware and software resources, each of which will be represented by a *Legion object*, which is an active process that responds to member function invocations from other objects in the system. Legion defines the message format and high-level protocol for object interaction, but not the programming language or the communications protocol.

Legion Design

? ***Classes manage their instances:*** Every Legion object is defined and managed by its *class object*, which is itself an active Legion object. Class objects are given system-level responsibility; classes create new instances, schedule them for execution, activate and deactivate them, and provide information about their current location to client objects that wish to communicate with them. In this sense, classes are *managers* and *policy makers*, not just definers of instances. Classes whose instances are themselves classes are called *metaclasses*.

? ***Users can provide their own classes:*** Legion allows users to define and build their own class objects; therefore, Legion programmers can determine and even change the system-level mechanisms that support their objects. Legion 1.4 (and future Legion systems) contains default implementations of several useful types of classes and metaclasses. Users will not be forced to use these implementations, however, particularly if they do not meet the users' performance, security, or functionality requirements.

Legion Design

? ***Core objects implement common services:*** Legion defines the interface and basic functionality of a set of core object types that support basic system services, such as naming and binding, and object creation, activation, deactivation, and deletion. Core Legion objects provide the mechanisms that classes use to implement policies appropriate for their instances.

  ? Examples of core objects include ***hosts***, ***vaults***, ***contexts***, ***binding agents***, and ***implementations.***

Legion Model

- Legion objects are independent, logically address-space-disjoint active objects that communicate with one another via non-blocking method calls that may be accepted in any order by the called object.
- Each method has a signature that describes the parameters and return value, if any, of the method. The complete set of method signatures for an object fully describes that object's interface, which is determined by its class.
- Legion class interfaces can be described in an *interface description language* (IDL), several of which will be supported by Legion.

Legion Design

- Legion implements a three-level naming system.
  - At the highest level, users refer to objects using human-readable strings, called *context names*. Context objects map context names to *LOIDs* (Legion object identifiers), which are location-independent identifiers that include an RSA public key. LOIDs they are location independent, LOIDs by themselves are insufficient for communication.
  - For communication, a LOID is mapped to an *LOA* (Legion object address) for communication. An LOA is a physical address (or set of addresses in the case of a replicated object) that contains sufficient information to allow other objects to communicate with the object (e.g., an <IP address, port number> pair).

Legion Design

- Legion will contain too many objects to simultaneously represent all of them as active processes. Legion has a strategy for maintaining and managing the representations of these objects on persistent storage.
- A Legion object can be in one of two different states, *active* or *inert*.
  - An inert object is represented by an *OPR* (object persistent representation), which is a set of associated bytes that exists in stable storage somewhere in the Legion system.
  - The OPR contains state information that enables the object to move to an active state.
  - An active object runs as a process that is ready to accept member function invocations; an active object's state is typically maintained in the address space of the process (although this is not strictly necessary).

Legion Design

- Several core object types implement the basic system-level mechanisms required by all Legion objects. Like classes and metaclasses, core objects are replaceable system components
  - ***Host objects:*** Host objects represent processors in Legion. One or more host objects run on each computing resource that is included in Legion. Host objects create and manage processes for active Legion objects. Classes invoke the member functions on host objects in order to activate instances on the computing resources that the hosts represent. Representing computing resources with Legion objects abstracts the heterogeneity that results from different operating systems having different mechanisms for creating processes. Further, it provides resource owners with the ability to manage and control their resources as they see fit.

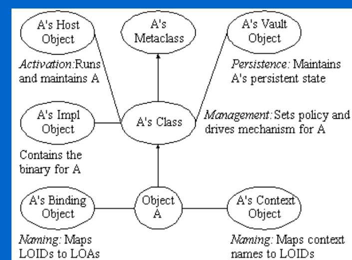
Legion Design

- ***Vault objects:*** Just as a host object represents computing resources and maintains active Legion objects, a vault object represents persistent storage, but only for the purpose of maintaining the state, in OPRs, of the inert Legion objects that the vault object supports. Context objects: Context objects map context names to LOIDs, allowing users to name objects with arbitrary high-level string names, and enabling multiple disjoint name spaces to exist within Legion. All objects have a current context and a root context, which define parts of the name space in which context names are evaluated.
- ***Binding agents:*** Binding agents are Legion objects that map LOIDs to LOAs. A <LOID, LOA> pair is called a binding. Binding agents can cache bindings and organize themselves in hierarchies and software combining trees, in order to implement the binding mechanism in a scalable and efficient manner.

Legion Design

? ***Implementation objects:*** Implementation objects allow other Legion objects to run as processes in the system. An implementation object typically contains machine code that is executed when a request to create or activate an object is made; more specifically, an implementation object is generally maintained as an executable file that a host object can execute when it receives a request to activate or create an object. An implementation object (or the name of an implementation object) is transferred from a class object to a host object to enable the host to create processes with the appropriate characteristics.

Legion Design

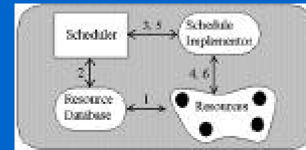


Legion

- Legion supports parallel processing in four ways:
  - Supporting popular parallel libraries, such as MPI
  - Supporting parallel languages, such as MPL
  - Offering wrap parallel components
  - exporting the run-time library interface to library, toolkit, and compiler writers

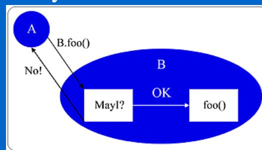
Legion

- Scheduling under Legion:
  - Users can choose their own scheduler(class) or use the default method available



Legion Security

- All classes should provide a means of access control for their own resources.
- All accesses to an object can be controlled by the “MayI” class.



Legion Security

- Public-key cryptography based on RSAREF 2.0.
- Three message-layer security modes: private (encrypted communication), protected (fast digested communication with unforgeable secrets to ensure authentic replies to message calls), and no security.
- Caching secret-keys for faster encryption of multiple messages between communicating parties.
- Auto-encrypted bearer credentials with free-form rights. Propagation of security modes and certificates through calling trees (e.g., if a caller demands encryption, all downstream calls will use it automatically).
- Drop-in addition of MayI functionality to existing objects.

Legion Security

- Secure legion shell to allow users to login to their authentication objects and obtain associated credentials and environment information.
- Isolation and protection of objects using local OS accounts.
- Easily checked Process Control Daemon for granting limited OS privileges to Legion Host Objects.
- Context space configured with access control for multiple users.

Legion : Hands on

- Run through the tutorials at legion.virginia.edu

- Make sure you don’t get confused between what is in the Unix world and the Legion world.

Legion : Hands on

- Before running anything… login using
  - Legion_login /users/<USER_NAME>

Legion : Hands on

- Running an MPI job
  - Build it as a Legion MPI executable
    - Native compile
    - Run legion_tty /home/<USER>/tty so that we can get output from Legion Space

Legion : Hands on

- Compiling Legion MPI jobs
  - They are the same as normal (mpich) MPI jobs

```
gcc -O -I/usr/local/Legion/include/MPI  bwtest.c -c

legion_link -mpi bwtest.o -o bwtest

torc0.cs.utk.edu> which legion_link
/usr/local/Legion/bin/ linux/legion_link
```

Legion : Hands on

- Before it can be used, it must be in Legion space as an object of the correct type.
- Needs to be registered
  - `torc0.cs.utk.edu> legion_mpi_register`
  - `usage: legion_mpi_register <class name> <task binary> <legion architecture>`

Legion : Hands on

```
torc0.cs.utk.edu> ls -la bwtest
-rwxr-xr-x   1 fagg     dongarra     14757 Apr 12 21:50
   bwtest*

torc0.cs.utk.edu> legion_mpi_register bwtest bwtest
   linux
"/home/fagg/mpi" does not exist - creating it
"/home/fagg/mpi/programs" does not exist - creating it
"/home/fagg/mpi/instances" does not exist - creating it
Task class "/home/ fagg/mpi/programs/bwtest" does not
   exist - creating it
Added 1 attributes(s) to object
"/home/fagg/mpi/instances/bwtest" does not exist -
   creating it
Registering implementation of  bwtest
```

Legion : Hands on

```
torc0.cs.utk.edu> legion_ls /home/fagg
mpi
tty
torc0.cs.utk.edu> legion_ls -laL
   /home/fagg/mpi/programs
.         (context)
   1.3aefa481.05.18000000.000001fc0bf7954c7cecb69d76eb29
   72bb8c5b426ac91dbb7c3582901cc4c0e236e769da92cf9777a1c
   ce317d14108d1b128a12c1f558e9d155d60f0fc6e50294f149efd
..        (context)
   1.3aefa481.05.17000000.000001fc0bf7954c7cecb69d76eb29
   72bb8c5b426ac91dbb7c3582901cc4c0e236e769da92cf9777a1c
   ce317d14108d1b128a12c1f558e9d155d60f0fc6e50294f149efd
bwtest    (MPI class)
   1.3aefa481.77000000..000001fc0e7338e446c8b4d8e3902829
   a96960699ca637544d4ea00e03297c396033a101784db8f764e62
   9b4df1f467d3a6022062ed0546f84baf09b8dd380e6181d5de3
```

Legion : Hands on

- Running the MPI job

```
torc0.cs.utk.edu> legion_mpi_run -n 2 -v
   /home/fagg/mpi/programs/bwtest
MPI run ID 1288753583
emptying context /home/fagg/mpi/instances/bwtest
getting context /home/fagg/mpi/instances/bwtest
setting context /home/fagg/mpi/instances/bwtest
Looking up program name /home/fagg/mpi/programs/bwtest
Creating 2 objects:
Creating object 0...
Creating object 1...
Application is done -- all children are done.
torc0.cs.utk.edu>
```

Legion : Hands on

- And some time later (via the legion_tty object)

```
torc0.cs.utk.edu> Hello from node 0 of 2 hostname
   torc0.cs.utk.edu
0 2.045000 ms 0.000000 MB/Sec
8 2.262700 ms 0.003372 MB/Sec
80 2.186600 ms 0.034892 MB/Sec
800 2.384650 ms 0.319938 MB/Sec
8000 28.465600 ms 0.268022 MB/Sec
80000 18.631050 ms 4.094989 MB/Sec
800000 185.564750 ms 4.111446 MB/Sec
Hello from node 1 of 2 hostname torc2
```

Legion : Hands on

- Controlling where you run the code:
  - Default scheduling policy will round robin execution
  - Or use a hostfile
    - Hostfile can be in legion space or on a unix file system
      - [-hf LegionHostsFile] [-HF LocalOSHostsFile]

Legion : Hands on

- I.e.

```
torc0.cs.utk.edu> legion_mpi_run -v -n 2 -HF h12
   /home/fagg/mpi/programs/infompi

MPI run ID 1293543413
emptying context /home/fagg/mpi/instances/infompi
getting context /home/fagg/mpi/instances/infompi
setting context /home/fagg/mpi/instances/infompi
Looking up program name
   /home/fagg/mpi/programs/infompi
Creating 2 objects:
Vector spawning 1 objects
Vector spawning 1 objects
Application is done -- all children are done.
torc0.cs.utk.edu>
```

Legion : Hands on

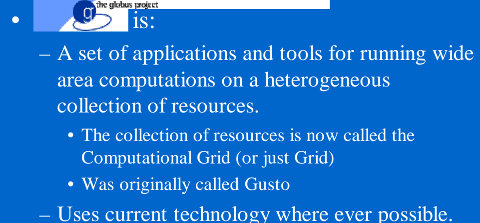
- The host file must list either the real legion contexts (that long string) or
- The short hand human readable form

  i.e. /hosts/<hname> <default number of resources>

```
torc0.cs.utk.edu> cat h12
/hosts/torc1.cs.utk.edu 1
/hosts/torc2.cs.utk.edu 1
```

This could be set to 2 as these machines are Dual processors!

Globus MetaComputing Toolkit



- is:
  - A set of applications and tools for running wide area computations on a heterogeneous collection of resources.
    - The collection of resources is now called the Computational Grid (or just Grid)
    - Was originally called Gusto
  - Uses current technology where ever possible.

Globus

- Mission statement:
  - The Globus project is developing the fundamental technology that is needed to build computational grids, execution environments that enable an application to integrate geographically-distributed instruments, displays, and computational and information resources. Such computations may link tens or hundreds of these resources.
    - Note: 10s to 100s… people have run PVM on a few hundred machines in '92! Also not aimed at workstation as computational resources but just as entry devices.
    - Real aim, MPPs… unlike Legion and Globe etc

Globus Components

- Globus is a modular system, so you can use some components without the need for others.
  - Communication
  - Resource Management
  - Remote Access
  - Security
  - Information Services

Globus Components Communications

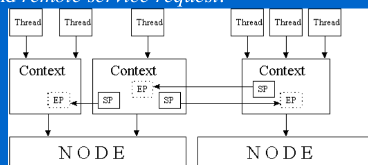
- Two systems are supported;
  - Native MPI (from Vendors or via MPICH)
  - NEXUS
    - Two version of NEXUS
      - Native, which is multithreaded
      - Lite, which is just socket based and single threaded
  - A special version of MPI using NEXUS as its ADI exists known as MPICH-g.
    - Newest version MPICH-G2 does not use NEXUS but only TCP!

Globus Components Communications

- NEXUS
  - Nexus is a portable library providing the multithreaded communication facilities required to implement advanced languages, libraries, and applications in heterogeneous parallel and distributed computing environments. Its interface provides a global memory model via interprocessor references; asynchronous events; and is thread safe when used in conjunction with the Globus thread library. Its implementation supports multiple communication protocols and resource characterization mechanisms that allow automatic selection of optimal protocols.
  - Nexus is intended for use by compiler writers and library developers, rather than application programmers.

Globus Components Communications

- NEXUS
  - Nexus supports five basic abstractions: the *node*, *context*, *thread*, *communication endpoint*, and *remote service request*.



Globus Components Communications

- Remote Service Request (RSR). A thread can request that an action be performed in a remote context by issuing a *remote service request*. This takes a handler identifier, a communication startpoint reference, and a message buffer as arguments and causes the specified handler to be executed on the node and within the context of the communication endpoint to which the startpoint is bound. The handler is passed the communication endpoint and the message buffer as arguments.
- Note this is an async function.
  - Message passing would use this to say put the sent message into the receivers memory!
- The endpoints cannot move, but the startpoints can.
- Communication between points is uni-directional!

Globus Components Resource Management

- The Globus Resource Allocation Manager (GRAM) processes the requests for resources for remote application execution, allocates the required resources, and manages the active jobs. It also returns updated information regarding the capabilities and availability of the computing resources to the Metacomputing Directory Service (MDS).
  - MDS is now known as Grid Information Service (GIS)
- GRAM provides an API for submitting and canceling a job request, as well as checking the status of a submitted job. The specifications are written by the user in the Resource Specification Language (RSL), and is processed by GRAM as part of the job request.

Globus Components Resource Management

- A job is submitted, the request is sent to the gatekeeper of the remote computer. The gatekeeper handles the request and creates a job manager for the job. The job manager starts and monitors the remote program, communicating state changes back to the user on the local machine. When the remote application terminates, normally or by failing, the job manager terminates as well.
  - This is much like the host objects under Legion.

Globus Components Resource Management

Globus Components Resource Management

- The structure of the individual GRAMs are much like the GRM and Tasker combinations in PVM.

Globus Components Resource Management

- **Scheduling Model**
  - The GRAM supports the following scheduling model. A user or resource broker submits a job request, which initially registers as a pending job. The job then undergoes state changes according to job outcome.
    - Pending - resources have not yet been allocated for the job.
    - Active - the job has received resources, and the application is executing.
    - Failed - the job terminated before completion, as a result of an error, or a user or system cancel.
    - Done - the job completed successfully.

Globus Components Remote Access

- Allows access to remote data:
  - GASS provides basic access to remote files
  - The Remote IO (RIO) library implements a distributed implementation of the MPI-IO, parallel I/O API.
  - Globus Executable Management (GEM) enables loading and executing a remote file through the GRAM resource manager.

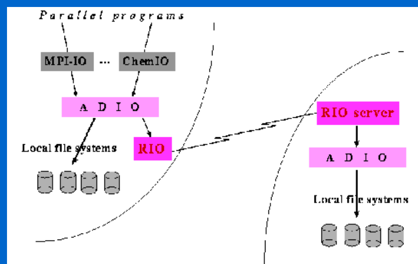
Globus Components
Remote Access

- GSS
  - A program that is to run on one or more remote MPPs can be modified to replace all open/close/fopen/fclose calls with gass_* versions. The program will then operate correctly on the remote computers. Furthermore, if multiple processes created on the remote computer read the same file, the file is copied only once.
  - GSS caches the data files in much the same way as AFS

Globus Components
Remote Access

- RIO: Remote I/O for Metasystems
  - The RIO library provides basic mechanisms for tools and applications that require high-performance access to data located in remote, potentially parallel file systems. RIO implements the ADIO abstract I/O device interface specification, which defines basic I/O functionality that can be used to implement a variety of higher-level I/O libraries.
  - The RIO implementation comprises two components, as shown in the following figure. The RIO *client* defines a remote I/O device for ADIO. This component translates ADIO requests that name a remote file (by using a URL-like syntax) into appropriate communictions to a remote I/O *server*. The server provides an interface to a particular file system, and serves requests from remote RIO clients that need to access that file system. The RIO server component itself calls ADIO routines to access different file systems.

Globus Components
Remote Access



Globus
Other Components

- Security
  - Is based on SSLEAY and uses SSL and X509 to identify both users and resource (providers)
  - Globus Security Infrastructure (GSI)
- Fault tolerance
  - Uses the Heart Beat Monitor (HBM)

Globus : Hands on

- Set the env variables and then
- Login!

```
torc0.cs.utk.edu> grid-proxy-init
Enter PEM pass phrase:
...+++++
...+++++
```

Globus : Hands on

- Am I logged in… names and auth info…

```
torc0.cs.utk.edu> grid-proxy-info -all
subject  : /C=US/O=Globus/O=University of Tennessee at
   Knoxville/OU=Innovative Computing
   Laboratory/CN=Graham E Fagg/CN=proxy
issuer   : /C=US/O=Globus/O=University of Tennessee at
   Knoxville/OU=Innovative Computing
   Laboratory/CN=Graham E Fagg
type     : full
strength : 512 bits
timeleft : 11:59:46
torc0.cs.utk.edu>
```

Globus : Hands on

- GLOBUS still needs a machines as did MPICH
- Difference is it needs it in the current directory

Globus : Hands on

```
cat ./machines
Torc2:2119/jobmanager 1
Torc3:2119/jobmanager 1
Torc5:2119/jobmanager 1
Torc6:2119/jobmanager 1
Torc7:2119/jobmanager 1
Torc8:2119/jobmanager 1

Used by MPICH to make a RSL file which is then passed
   to globus-run
```

Globus : Hands on

- BUT /usr/local/gt-mpich/bin/mpirun does not work! (MDS problem… grr)
  - Well it hangs… it does get their in the end (sometimes)

```
torc0.cs.utk.edu> /usr/local/gt-
  mpich/bin/mpirun -t -v -np 3 infompi
running /ptfe/homes/fagg/globus/infompi on 3
  LINUX globus processors
```

Globus : Hands on

- This means we have to do what it does
  - It builds an RSL file and then globusruns it
- So we have to build a RSL file by hand !
  - Yes I am serious
    - But we can get MPIRUN to do most of it for us

Globus : Hands on

```
torc0.cs.utk.edu> /usr/local/gt-mpich/bin/mpirun -np
  1 -globusargs dumprsl infompi  > c.rsl

torc0.cs.utk.edu> cat c.rsl
+
(
  &(resourceManagerContact="torc3.cs.utk.edu:2119/jo
  bmanager")
   (count=1)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
   (directory=/ptfe/homes/fagg/globus)
   (executable=/ptfe/homes/fagg/globus/infompi)
)
torc0.cs.utk.edu>
```

Globus : Hands on

```
torc0.cs.utk.edu> /usr/local/gt-mpich/bin/mpirun -
  globusrsl c.rsl

[torc3:24098]   MPI version dependant startup
  information

[torc3:24098]   Argc = 1 and first argv is
  /ptfe/homes/fagg/globus/infompi
[torc3:24098]   0 /ptfe/homes/fagg/globus/infompi

[torc3:24098]   MPI independant information

[torc3:24098]   Host [torc3] Size of 1, my rank in
  that size 0
```

Globus : Hands on

```
torc0.cs.utk.edu> cat single.rsl
+
(& (resourceManagerContact ="torc2.cs.utk.edu:2119/jobmanager")
   (count=2)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
   (arguments=" ")
   (directory="/ptfe/homes/fagg/globus")
   (executable="/ptfe/homes/fagg/globus/infompi")
)

torc0.cs.utk.edu>
```

Globus : Hands on

```
torc0.cs.utk.edu> cat twin.rsl
+
(& (resourceManagerContact ="torc2.cs.utk.edu:2119/jobmanager")
   (count=1)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0))
   (arguments=" ")
   (directory="/ptfe/homes/fagg/globus")
   (executable="/ptfe/homes/fagg/globus/infompi")
)
(& (resourceManagerContact ="torc3.cs.utk.edu:2119/jobmanager")
   (count=1)
   (label="subjob 1")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1))
   (arguments=" ")
   (directory="/ptfe/homes/fagg/globus")
   (executable="/ptfe/homes/fagg/globus/infompi")
)

torc0.cs.utk.edu>
```

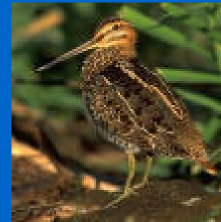
Globus : Hands on

- Now to run from an RSL file
  - Two methods…
    - Pass it to mpirun

```
torc0.cs.utk.edu> /usr/local/gt-mpich/bin/mpirun
  -globusrsl b.rsl
```

    - Pass it to globusrun

SNIPE
Scalable Networked Information Processing Environment



What is SNIPE?

- Platform for large-scale distributed systems
- Reliable and fault-tolerant
- Supports continuous operation
- Very heterogeneous (PDAs to supercomputers)
- Adaptable to changing conditions (link or host failures)
- Secure (provides authentication and privacy)

SNIPE Design Principles

- Reliability
  "Better living through redundancy"
- Scalability
  "Minimize the amount of shared state"
- Simplicity and Flexibility
  "Keep the core small"
- Efficiency
  "Optimize the common cases"

SNIPE Components

- RCDS metadata servers
- Host daemons
- Client library
- Resource managers
- File servers
- Playgrounds
- Consoles

(RCDS metadata servers, Host daemons, Client library: core)

RCDS

(Resource Cataloging and Distribution System)

- Provides stable storage for metadata
  (attributes of hosts, tasks, services, etc.)
- Allows dynamic update on a per-attribute basis
- Replicates data for redundancy
  (peer servers immediately/automatically updated)
- Scalable
  (queries or updates can go to any peer)
- Supports end-to-end security
  (authenticated updates; signed metadata)

Host Daemons

- Mediates the use of resources on a host
- Authenticate requests for local processing
- Enforce access restrictions on local resources
- Manage local tasks (startup, monitoring, cleanup)
- Deliver asynchronous messages to local tasks
  Inform interested parties of local state changes
  (e.g. local task dies or is moved elsewhere)
- Monitor local resource usage (cpu, disk, mem)

Client Library

- Provides programming interface to clients
- Communication
  (routing, message passing, fragmentation, data conversion, authentication, encryption, use of different communications media)
- Task management (initiate, monitor tasks)
- Resource location
  (obtain information about named resources)
- Access to data stores

Resource Managers

- Manage a group of named resources
  (hosts, processes, file servers)
- Implement/Enforce locally-specified policy
- Behaves as a “virtual host”
  (you make requests of it as if it were a host, which it delegates to other hosts)
- May manage resources on several hosts
- May be a replicated service
- May migrate tasks from one host to another

File Servers

- Provide access to data sets by SNIPE processes
- Replicate data sets as needed to provide redundancy
- Maintain file metadata (including locations) in RCDS
- Implemented as a simple SNIPE process which functions as a source or sink.

Playgrounds

- Facilitates secure execution of mobile code
- Can be native machine code (modulo OS support), or interpreted (Java, Limbo, Python)
- Downloads code from a file server
- Verifies authenticity and access rights
- Enforces usage and access restrictions
- Logs violations
- Provides run-time environment

SNIPE use of RCDS

- Host metadata:
  architecture, OS, network interfaces, resource manager, public keys
- Process metadata:
  current state and location, public keys
- Service metadata:
  locations of replicated services and data sets
- Multicast Group metadata:
  locations of multicast group servers

Host Metadata

- Indexed by host URL
  `x-snipe://cetus1a.cs.utk.edu/`
- Contains:
  - host architecture, OS name, and OS version
  - characteristics of host network interfaces (network, address, netmask, port, speed)
  - URLs of resource managers for this host
  - public keys for this host

Process Metadata

- Indexed by Process URL
- Contains:
  - current host, pid, uid, gid, etc.
  - current SAPs for process communications (network, address, netmask, port, bandwidth)
  - SAP of current host daemon (for asychronous messages)
  - "worry list" of processes that want to know if this process dies, migrates, cannot be reached, etc.

File and Service Metadata

- Description
  (file content-type, creation date, etc.)
- Signatures (for static files - optional)
- Current Locations

Multicast Group Metadata

- Multicast groups are named by URLs
- Host daemons elect themselves as servers for one or more multicast groups, depending on the presence or absence of other nearby servers for that group
- RCDS keeps track of locations of servers for any particular multicast group
- Multicast servers relay messages to other multicast servers, and to local clients
- Processes join a group by using RCDS to find servers for that group, then contacting a server that's nearby

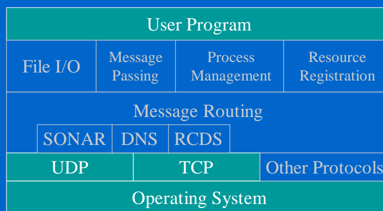
API Characteristics

- All identifers are globally scoped:
  - URLs (hosts, processes, services, data stores)
  - email addresses (security principals)
- Most identifiers are location-independent
- Integer resource-ids used within a process (similar to UNIX file descriptors)

Client Library Layers

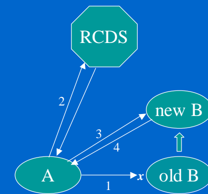
- Simple communications core:
  `snipe_connect (),snipe_listen (), snipe_read (), snipe_write (), snipe_sndmsg (),snipe_getmsg (), snipe_putattr (), snipe_getattr ()`
- Core uses TCP, UDP, DNS, RCDS, SONAR
- Resource management, multicast, process migration, external file I/O layered above
- Planned: MPI, RPC, PVM layers above that
  - What we have is FT_MPI and Hcore

Client Library



Connection Maintenance During Process Migration

1. Process B informs RCDS when it migrates to a new location. B also notifies its peers.
2. A gets connection migration notification from B, OR B fails to acknowlege message from A.
3. A asks RCDS for B's current SAPs
4. A contacts B at its new address, establishes new connection
5. Communication resumes

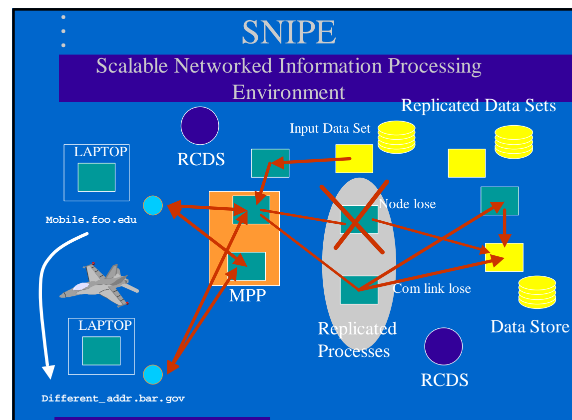


Other Features of SNIPE

- Can handle more than one connection media and protocol at a time, and then switch between them depending on message type, priority etc
  - NEXUS claims to be the only system that can do this.
    - NEXUS does not do it automatically but via a series of call backs.

SNIPE

Scalable Networked Information Processing Environment



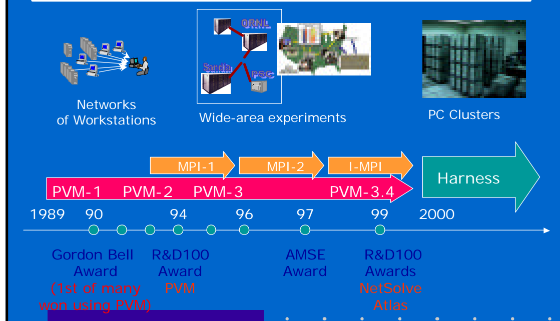
SNIPE Applications

- MPI_Connect ( )
- FT_MPI
- Fixing IBMs implementation of MPI on the SMP SP2 nodes.
  - As used by LLNL for 2nd place in the 1999 Fall Top500 list with 2.144 TerraFlops
    - Was #1 for a month…

HARNESS

- HARNESS
  - Heterogeneous Adaptable Reconfigurable NEtworked SystemS
- Also described as a
  - “*distributed, reconfigurable and heterogeneous computing environment that supports dynamically adaptable parallel applications*”

Why HARNESS



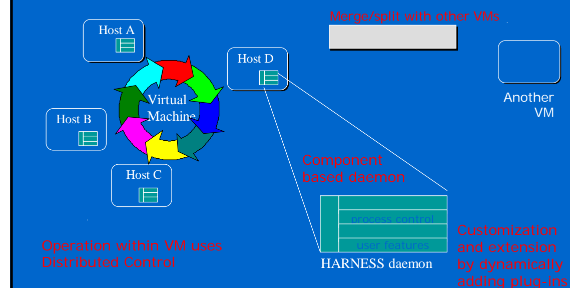
HARNESS

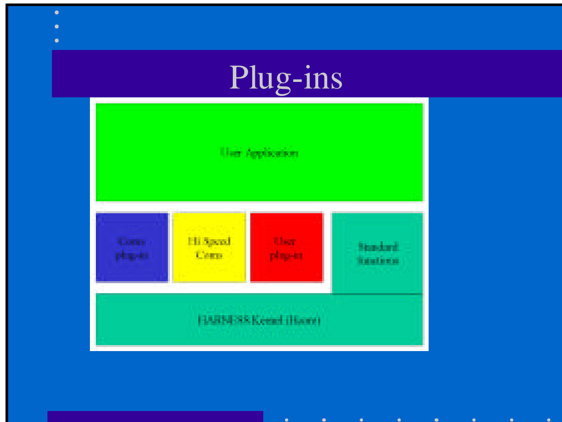
- Basics:
  - Built on the Distributed Virtual Machine (DVM) concept
  - Allows different applications (**new** and *legacy*) to interact and alter their environment
    - change their view of the world / multiple different views
- Why heterogeneous?
  - In terms of architecture, OS and high level programming paradigm.
    - I.e. a message passing code on a dedicated MPP should be able to communicate with a VSM application on a workstation cluster .

HARNESS

- Adaptable and Reconfigurable - in terms of both the DVM itself and the application
- Application level:
  - An application can be reconfigured and steered by external processes (via tool modules)
  - An application can ask to change its view of the world by requesting that the DVM view changes its actions/functionality or capabilities.
    - I.e. I need a FFT solver, (the DVM finds one, loads it, makes its interface available and this discards it)

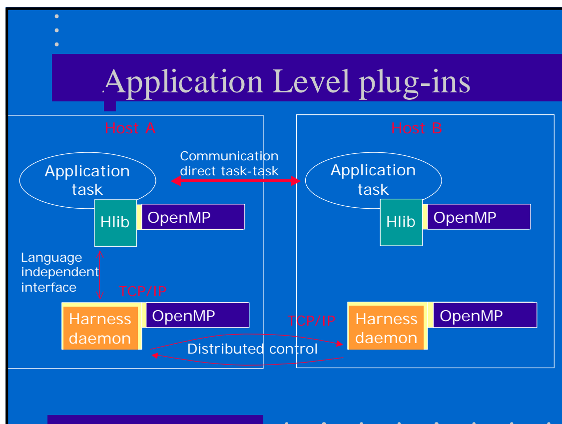
HARNESS





Programming Models

- Different parts of an application can use different programming models/API
- 4 to be included
  - plug-in interface (HARNESS itself)
  - MPI
  - PVM
  - IceT (from Emory)
    - Java based message passing system that allows code segments to be uploaded and executed on the fly.



HARNESS

- EMORY University
  - Has made a complete HARNESS system
  - JAVA based so no problems with Plug-ins not working
    - Its all BYTECODE
  - They also have a PVM plug-in
    - Works at the TD and SD protocol level
      - OK the level the messages work at in between the PVM Daemons and the PVM tasks.. I.e. no need to recompile for PVM to work on there system.

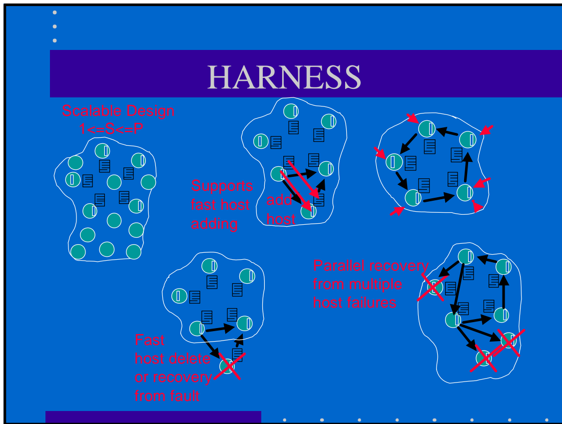
HARNESS

- EMORY DVM system
  - Consists of many parts
  - DVMs
    - Each has a master kernel that keeps track of the others in the DVM (HDVM server)
  - Single point of contact name service so you can find your own DVM
    - Had fault tolerance problems
    - We fixed this by making a ring of name servers

HARNESS

- ORNL was building the control algorithms and C-based plug-ins
  - No single point (or set of points) of failure for Harness. It survives as long as one member still lives.
  - All members know the state of the virtual machine, and their knowledge is kept consistent w.r.t. the order of changes of state. (Important parallel programming requirement!)
  - No member is more important than any other (at any instant) i.e. here isn't a pass-around "control token"

HARNESS



HARNESS

- ORNL
  - Symmetric Control Algorithm and C based HCORE
- Not built yet.. Although they do have a control algorithm prototype in Java.

HARNESS

- UTK had to build a FT-MPI implementation
- What ICL-UTK needed for FT-MPI
  - A kernel to plug into
  - Naming service
  - MetaData store to hold our plug-in state information
  - ALL IN ‘C’

HARNESS

- Three tracks

  Build Plug-ins for the EMORY java system

  Build our own HCORE / naming service etc

  HELP ORNL with their Hcore

HARNESS

- Three tracks

  Build Plug-ins for the EMORY java system
  Done it (see next slide)

  Build our own HCORE / naming service etc
  Almost finished this !

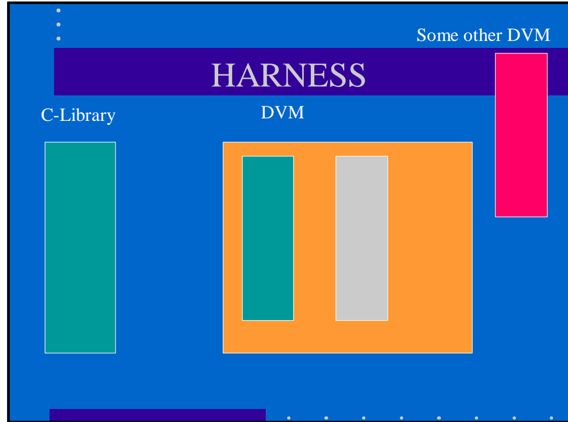
  HELP ORNL with their Hcore
  Code sharing

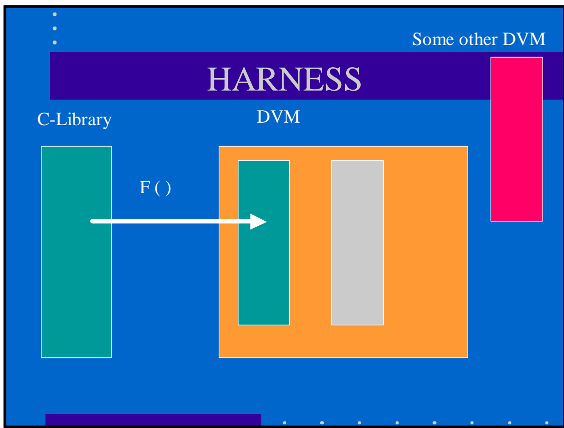
HARNESS

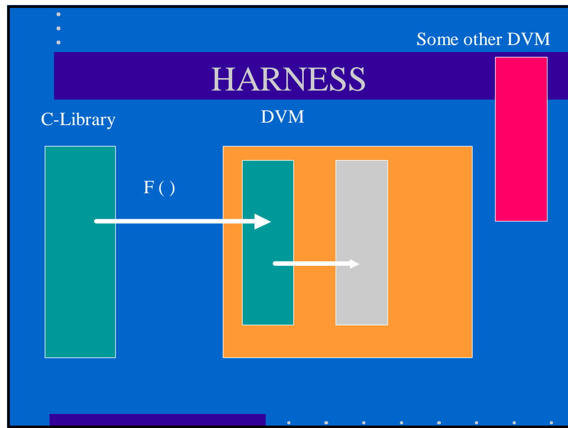
- JAVA and C mixed or “the tail of two protocols”
  - TCP on –the-wire protocol and JNI-RMI mixed
- Support two features
  - (1) A C interface to the EMORY DVM
    - Emory had a very very nice SWING GUI
  - (2) Allow C plug-ins to be used with the DVM
    - Originally could only do Java Class libraries

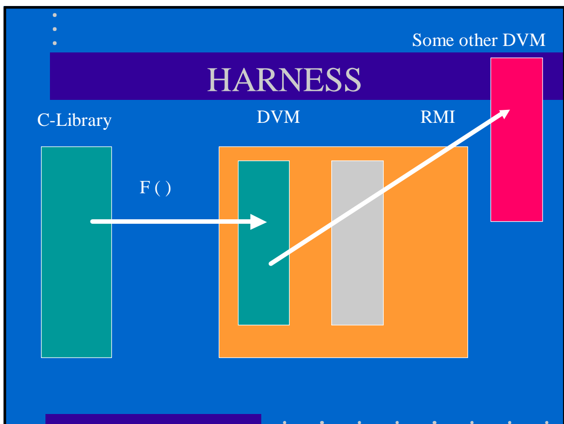
HARNESS

- C interface to the DVM is done by a small library that makes requests to a special handler thread within the DVM
  - Also supports the insertion of “Events” that are broadcast in an ordered why by the DVM
    - Allows global synchronous state updates



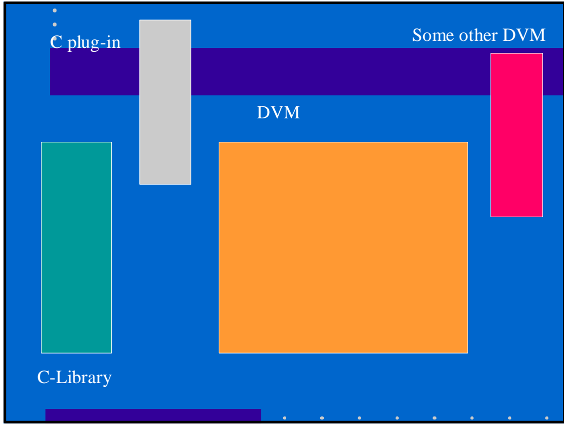


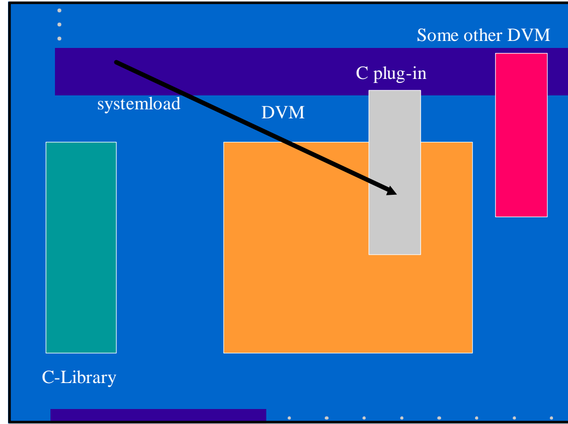


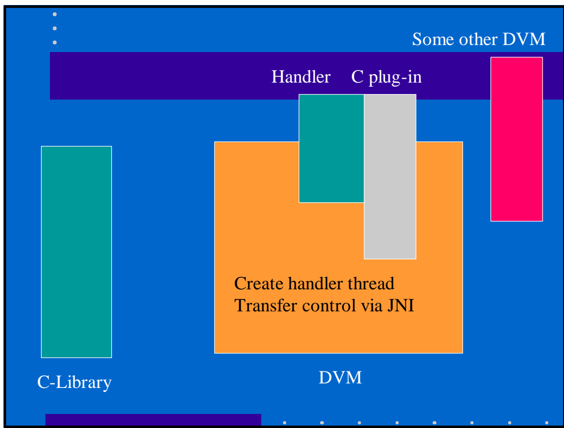


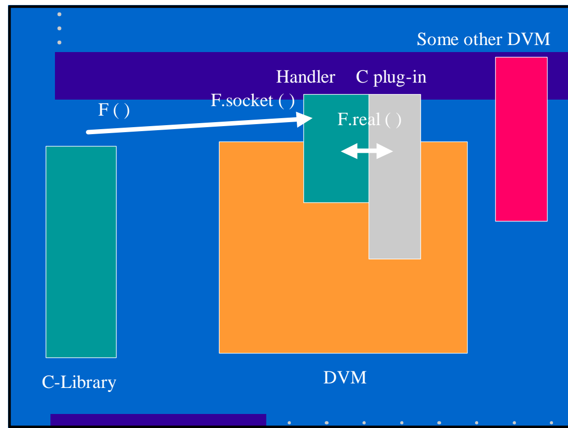
HARNESS

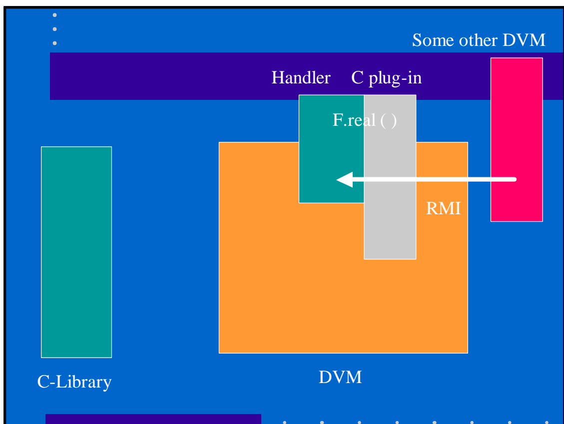
- C plug-ins into the JAVA HARNESS was harder
  - The DVM makes a JNI call to a handler thread in the C plug-in
    - This code is generated for us!
    - It handles a remote C based program talking to the C plug-in
  - Other JAVA systems can make RMI / JNI calls to the plug-ins via almost normal class invocation methods
- It works for us anyway











HARNESS

- UTK has build a small HARNESS Core
  - G_HCORE
  - C based
  - Supports a number of load/unload policies
  - Supports a number of protocols
- If asked to get a library it can not find locally, gets the entry via HTTP from a repository
  - (Can use RCDS for replicated repositories)
  - (As in the SC98 demo of SNIPE)

G_Hcore

- API
  - Load_run_unload (char *, …)
  - Load_only ( )
  - Unload ( )
  - Exec ( )
    - Local invocation (function call)
    - New thread
    - Fork
    - Call by socket
      - Protocol handler

G_Hcore

- Speedy
  - Emory DVM invocation times (mSec)
  - RMI Local : 0.172 Remote : 1.406
  - Socket Local : 10.4 Remote : 8.6
  - G_hcore
  - Direct (new thread) : 0.187
  - TCP Local : 0.58 Remote : 1.08
  - UDP Local : n/a Remote : 0.30
  - All tests done on the Torc cluster over 100MB ethernet

Harness

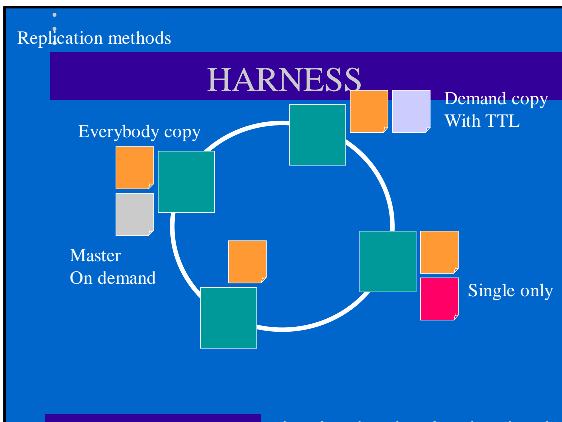
- Name Service and Meta-Data Store

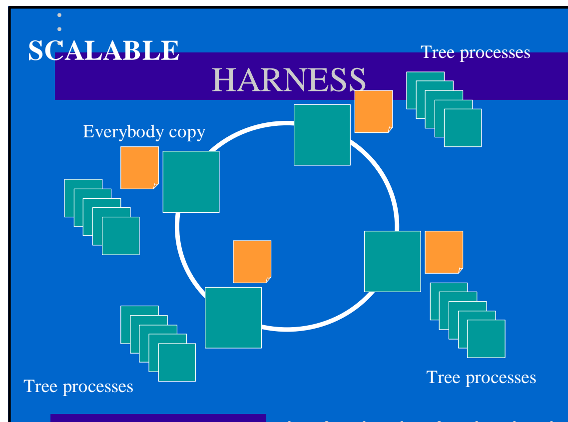
  Are the same thing?

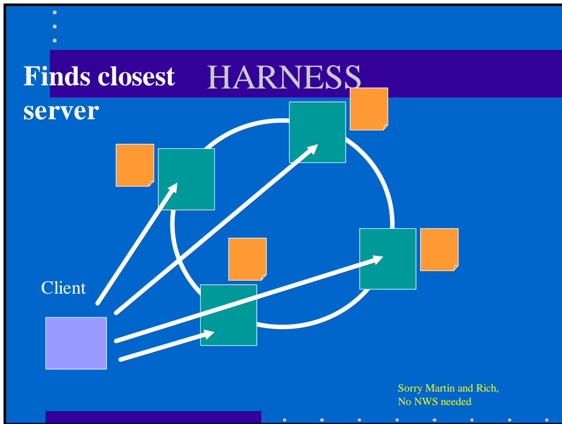
Jointly done by myself and Mickael Melki from ENS-Lyon

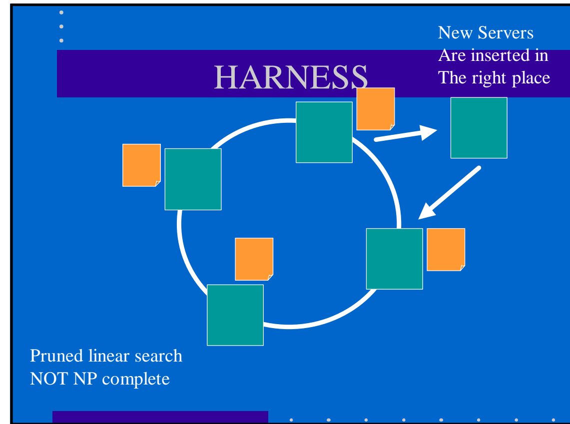
HARNESS

- DataRing that uses PVM 3.4 Mailbox searching semantics
  - Regular expressions searches
- DATA Records are:
  - Replicated
    - multiple methods (single, everybody, on demand)
  - Owned
    - Single owner (static, dynamic)
  - Have TTL or are linked to the owners life span
- Consistency is done by a multi-phase algorithm!
  - Aka no need to wait for the ORNL symmetric control algorithm









Next

- Break
- Harness demo
- Fix homework problems
- Set next homework