

From Legion to Avaki: The Persistence of Vision Part 1

Andrew S. Grimshaw, Anand Natrajan
Marty A. Humphrey, Michael J. Lewis
Anh Nguyen-Tuong, John F. Karpovich
Mark M. Morgan, Adam J. Ferrari

Based on slides by Lee, Hwang Jik
Supercomputing Lab, Yonsei University and from
University of Virginia

2004-10-19

3

Introduction

- ◆ Grids Are Here
- ◆ Grid Architecture Requirements
- ◆ Legion Principles and Philosophy
- ◆ Using Legion in Day-to-Day Operations
- ◆ The Legion Grid Architecture: Under the Covers
- ◆ Core Legion Objects
- ◆ The Transformation From Legion to Avaki

2004-10-19

2

Grids Are Here

- ◆ Avaki (Commercial ventures)
 - ◆ Has its roots in Legion, a Grid project at the University of Virginia begun in 1993
- ◆ The near future
 - ◆ No longer be executed on supercomputers and single workstations using local data sources
 - ◆ => Users will be presented the illusion of a single, very powerful computer
 - ◆ The system will schedule application components on processors, manage data transfer, and provide communication and synchronization

2004-10-19

3

Grid Architecture Requirements

- ◆ Definitions
 - ◆ Grid system
 - ◆ A collection of distributed resources connected by a network
 - ◆ Grid application
 - ◆ Operates in a Grid environment or is "on" a Grid system
 - ◆ Grid software
 - ◆ Facilitates writing Grid Applications and manages the underlying Grid infrastructure

2004-10-19

4

Grid Architecture Requirements

◆ Requirements (1/3)

- ◆ *Security*: A Grid system must have mechanisms that allow users and resource owners to select policies that fit particular security and performance needs
- ◆ *Global name space*: All Grid objects must be able to access any other Grid object *transparently* without regard to location or replication
- ◆ *Fault tolerance*: Hosts, networks, disks and applications frequently fail, restart, disappear and behave otherwise unexpectedly

2004-10-19

5

Grid Architecture Requirements

◆ Requirements (2/3)

- ◆ *Accommodating heterogeneity*: A Grid system must support interoperability between heterogeneous hardware and software platforms
- ◆ *Binary management*: The underlying system should keep track of executables and libraries, knowing which ones are current
- ◆ *Multi-language support*: Fortran or C
- ◆ *Scalability*: The service demanded of any given component must be independent of the number of components in the system => distributed systems principle
- ◆ *Persistence*: I/O and the ability to read and write persistent data are critical in order to communicate between applications and to save data

2004-10-19

6

Grid Architecture Requirements

◆ Requirements (3/3)

- ◆ *Extensibility*: Grid systems must be flexible enough to satisfy current user demands and unanticipated future needs => value-added services
- ◆ *Site autonomy*: For each resource the owner must be able to limit or deny use by particular users, specify when it can be used
- ◆ *Complexity management*: Providing the programmer and system administrator with clean abstractions is critical to reducing the cognitive burden

2004-10-19

7

Legion Principles and Philosophy

◆ The Design principles and philosophy

- ◆ Provide a single-system view
 - ◆ To reduce the complexity of the overall system and provides a single namespace
- ◆ Provide transparency as a means of hiding detail
 - ◆ Users and programmers should not have to know where an object is located in order to use it
- ◆ Provide flexible semantics
- ◆ By default the user should not have to think about plumbing or infrastructure
- ◆ Reduce "activation energy"
- ◆ Do not change host operating systems
- ◆ Do not change network interfaces
- ◆ Do not require Grids to run in privileged mode
 - ◆ Require Grid software to run with the lowest possible privileges

2004-10-19

8

Using Legion in Day-to-Day Operations

- ◆ A compute Grid and a data Grid of Legion
 - Allowing processing power to be shared
 - A virtual single set of files that can be accessed without regard to location or platform
- ◆ A typical scenario
 - A user sits down at a terminal, authenticates to Legion (logs in) and runs the command
 - `legion_run my_application my_data`

2004-10-19

9

Using Legion in Day-to-Day Operations

- ◆ A typical scenario (cont.)
 - Determine the binaries available
 - Find and select a host on which to execute *my_application*
 - Manage the secure transport of credentials
 - Interact with the local operating environment on the selected host (SGE queue)
 - Create accounting records
 - Check to see if the current version of the application has been installed
 - Move all of the data around as necessary
 - Return the results to the user

2004-10-19

10

Using Legion in Day-to-Day Operations

- ◆ Key features
 - Global name space
 - Names everything: processors, applications, queues, data files and directories
 - Wide-area access to data
 - All of the named entities are mapped into the local file system directory structure of her workstation, making access to the Grid transparent
 - Access to distributed and heterogeneous computing resources - binary availability/versions
 - Single sign-on
 - Policy-based administration of the resource base
 - Accounting both for resource usage information and auditing purposes
 - Find-grained security that protects both her resources and those of others
 - Failure detection and recovery

2004-10-19

11

Creating and Administering a Legion Grid

- ◆ Once a Grid is created, users can think of it as one computer with one directory structure and one batch processing protocol
- ◆ Two administrative ways
 - As a single administrative domain: When all resources on the Grid are owned or controlled by single department or division
 - As a federation of multiple administrative domains: When resources are part of multiple administrative domains
 - Administrators define which of their resources are made available to the Grid and who has access

2004-10-19

12

Creating and Administering a Legion Grid

- ◆ Legion provides features for the convenience of administrators
 - Monitoring load, idle time, etc
 - Add/remove resources
 - Dynamic reconfiguration based on policy
 - Logging
 - Collection of resource usage information

2004-10-19

13

Legion Data Grid

- ◆ Concepts of Legion data Grid
 - Users access files by name - typically a pathname in the Legion virtual directory
 - There is no need to know the physical location of the files
 - How the data is accessed, and how the data is included into the Grid

2004-10-19

14

Legion Data Grid

- ◆ Data Access
 - DAP Access (a Legion-aware NFS server)
 - Provides a standards-based mechanism to access a Legion Data Grid
 - Differences
 - It has no actual disk or file system behind it
 - It supports the Legion security mechanisms
 - It caches data aggressively
 - Can have DAP per site or per host
 - Command Line Access
 - A set of command line tools that mimic the Unix file system commands such as *ls*, *cat*, etc → *legion_ls*, etc
 - I/O Libraries
 - A set of I/O libraries that mimic that *stdio* libraries

2004-10-19

15

Legion Data Grid

- ◆ Data Inclusion
 - "copy"
 - Copy of the file is made in the grid
 - "legion_cp" command
 - "container"
 - Copy of the file is made in a container on the grid
 - Reduces the overhead associated with having one service per file
 - "share"
 - The data continues to reside on the original machine
 - "legion_export_dir" command starts a daemon that maps a file or rooted directory in Unix or Windows NT

2004-10-19

16

Distributed Processing

- ◆ In a typical network
 - The user must know where the file is, where the application is, and whether the resources are sufficient to complete the work
- ◆ With Legion
 - Users have a single point of access to an entire Grid
 - Users log in, define application parameters and submit a program to run on available resources
 - Input data is read securely from distributed sources without necessarily being copied to a local disk

2004-10-19

17

Distributed Processing

- ◆ Automated Resource Matching and File Staging
 - Administrative controls and predefined policies
 - Matches applications with queues in different ways
 - Through access controls: a user and application may or may not have access to a specific queue
 - Through matching of application requirements and host characteristics: a specific operating system/library
 - Through prioritization: based on policies and load conditions
- ◆ Support for Legacy Applications - No Modification Necessary
 - Applications can run anywhere at all on the Grid without regard to location or platform as long as resources are available that match the application's needs

2004-10-19

18

Distributed Processing

- ◆ Batch Processing - Queues and Scheduling
 - Users can execute applications interactively or submit them to a queue
- ◆ Queues
 - Shared processing power
 - Sequence jobs based on business priorities
 - Distribute jobs to available resources
 - Permit allocation of resources to groups of users
- ◆ Administrator tasks
 - Monitor usage from anywhere on the network
 - Preempt jobs, re-prioritize and re-queue jobs
 - Establish policies based on time windows, load conditions or job limits

2004-10-19

19

Security

- ◆ Security of Legion
 - Designed in the Legion architecture and implementation from the beginning
 - Authentication, authorization and data integrity
 - See references 7 and 9 in Chapter

2004-10-19

20

Automatic Failure Detection and Recovery

- **Fault-tolerant of Legion**
 - If a computer goes down, Legion can migrate applications to other computers based on predefined deployment policies as long as resources are available that match application requirements
- **Legion provides fat, transparent recovery from outages**
 - Hosts, jobs and queues automatically back up their current state, enabling them to restart with minimal loss of information
- **Systems can be reconfigured dynamically**
 - Processing continues using other resources without interrupting operations
- **Legion migrates jobs and files as needed**
 - The job is automatically migrated to another host and restarted

2004-10-19

21

The Legion Grid Architecture: Under the Covers

- **Legion**
 - An object-based system comprised of independent objects
- **Legion class interfaces**
 - Interface Description Language (IDL)
 - CORBA IDL, MPL, BFS
 - Communication
 - Supported for parallel applications (MPI libraries)
 - Supports cross-platform, cross-site MPI applications
- **All legion objects**
 - **Name, state** (which may or may not persist), **Meta-data** (<name, valueset> tuples) associated with their stat and an interface
 - Similar to OGSA

2004-10-19

22

Legion

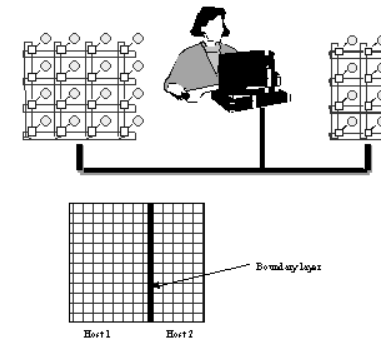
Parallel Processing

- Bag-of-tasks
 - ◆ Parameter space studies
 - ◆ Monte Carlos
 - ◆ Simple data-parallel
- Stencils (regular)
 - ◆ Ocean codes
 - ◆ PPM

Bag of Tasks



Wide-area parallel processing

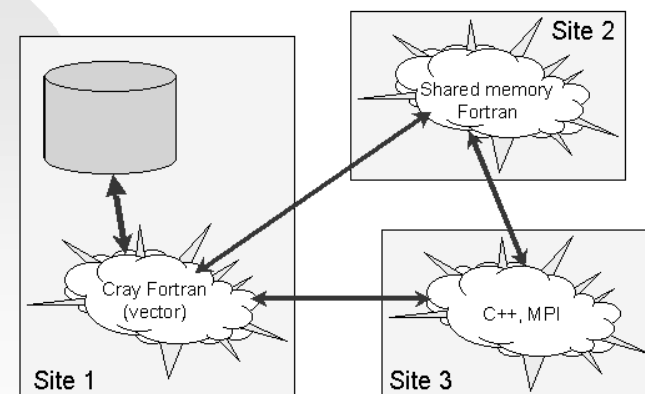


May be heterogeneous hosts

Meta-applications

A meta-application may be constructed from multiple components - each of which was previously a single application.

Meta-application example



Characteristics of Meta-applications

- Multiple components
 - ◆ modeling different physical components
 - ◆ written by different groups
 - ◆ historically separate applications
 - ◆ possibly written in different languages
 - ◆ with internal parallelism (task and data)
 - ◆ some have sub-components or sub-models
 - ◆ possibly executing on different scales

Architecture and Object Model

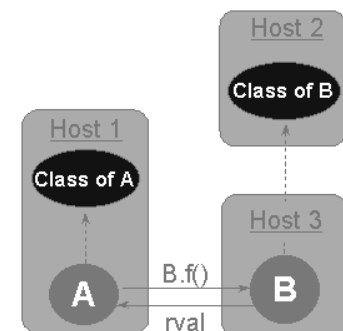
System architecture is key to the success of metasystems.

Legion is object-based

- Everything is an object
- Legion's core "system" objects
 - ◆ Hosts
 - ◆ Vaults
 - ◆ Classes - LegionClass
- All system objects can be replaced
- Classes are objects

The object model

- Legion objects
 - ◆ belong to classes
 - ◆ are logically independent, address-space-disjoint
 - ◆ communicate via non-blocking method calls
 - ◆ are *Active* or *Inert*
 - ◆ export *object-mandatory* member functions
 - ◆ are named using LOIDs



Legion Object Identifiers (LOIDs)

- LOIDs name objects and are location independent strings of bits
- LOIDs have many fields (arbitrary)
- Basic LOIDs have
 - ◆ type
 - ◆ “domain”
 - ◆ class id (integer)
 - ◆ instance id (integer)
 - ◆ public key (RSA) Secure communication without trusted 3rd part

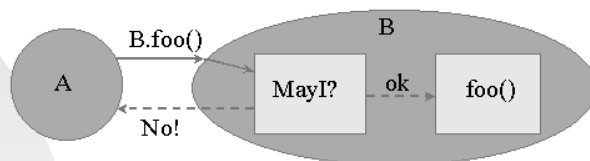
Dotted-hex notation

The Legion LOID

Format	Class Identifier	Instance Number	Public Key
1. 01.	07.	01.	000001fc0d24ab3512f 63533f3f96b2bba0572 aed8d47efec7c24618f ca3a3d49326fa84eaa3 20724c10264f225c978 20ace d7458619497425 6ee86c2ae0565d55c8e 623

1.01.07.01000000.000001fc0ce16fe5110ac2a6051c70191b05c42
4d3bb6c18a8de593a32ac18fle3d410a4636cb39edd281cf6eb264
758710c7609497003db974aeccc0a23ffeabdb87d8b

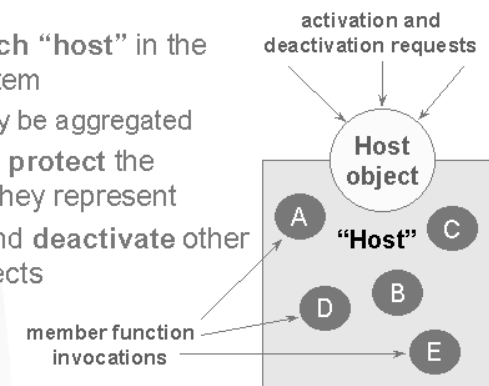
Access Control



- No single access control policy
 - ◆ Use a standard policy or define your own
- Security restricted to MayI function
 - ◆ Policies can be replaced and verified
 - ◆ use a simple access control language

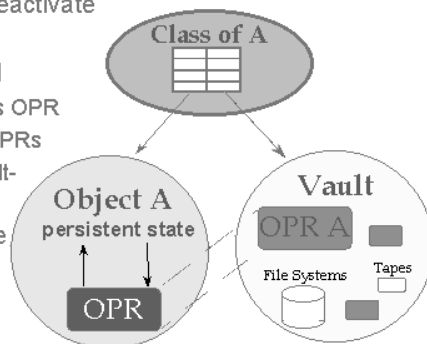
Host objects

- Run on each “host” in the Legion system
 - ◆ hosts may be aggregated
- Guard and protect the resources they represent
- Activate and deactivate other Legion objects



Vaults

- Objects will migrate, deactivate
 - ◆ State must persist
- Three objects involved
 - ◆ the object writes to its OPR
 - ◆ the Vault manages OPRs
 - ◆ Classes manage vault-object matching
- Vaults abstract storage

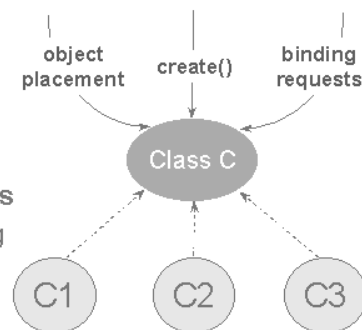


Vaults

- Are the **mechanism** for object persistence
 - ◆ “contain” inert Legion objects
 - ◆ keep the OPRs safe while an object is inert
 - ◆ are known by the class of the object
- Are **policy makers** (selected by the class)
 - ◆ how is the OPR maintained?
 - ◆ how is it kept “safe”?
 - ◆ many different implementations are possible
 - ◆ file system, DBMS, even shared memory
 - ◆ trust anyone, trust only the class, etc.

Class Objects

- Classes are **managers** that provide system-level services
 - ◆ creation
 - ◆ deletion
 - ◆ binding
 - ◆ keep track of binaries
- Classes are **policy makers**
 - ◆ placement and scheduling
 - ◆ fault-tolerance
 - ◆ instance implementation.



For example...

- Classes may desire
 - ◆ application-specific scheduling algorithms
 - ◆ to be part of a mandatory security protocol for all instances
 - ◆ to multiplex multiple instances within a single address space
 - ◆ to hand out different aliases for the same instance
 - ◆ to define and use special inheritance semantics
 - ◆ to manage changing instances

Binary Management

- User compiles code for each architecture of interest
- User registers binaries with a class object
- Class object, working with host object, ensures the right binaries are always used.

Implementation Objects

- ✦ Encapsulate Legion executables
- ✦ Write-once, read-many
- ✦ Typically contain code for single platform
- ✦ May be Java byte code, Perl scripts, or HLL that requires compilation
- ✦ Maintain a set of attributes
 - ✦ Type of executable, machine requirements, performance characteristics
- ✦ Class objects maintain list of acceptable implementation objects
- ✦ Allows multiple implementation with different time/space trade-offs

2004-10-19

42

Implementation Caches

- ✦ Avoid storage & comm costs by caching implementation
- ✦ Host object invoke cache objects to download implementation
- ✦ If Cache object does not have, it downloads and caches
- ✦ Invalidation of old versions easy
 - ✦ Class objects specify LOID of implementation
 - ✦ Need only change list of binaries
 - ✦ Future invocations will specify new binaries
 - ✦ Old versions will time out and be deleted from cache

2004-10-19

43

Legion to Avaki

- ✦ Legion began 1993 - determining requirements, designing architecture, coding
- ✦ First funding and code 1996
- ✦ 1997 first deployment
 - ✦ Initially difficult to maintain MTBF > 20 hrs due to peculiar failure modes
 - ✦ By Nov 1998 MTBF > 1 month
- ✦ Always intended to move the software into mainstream with commercial support
 - ✦ 2001 Avaki raised capital and released Avaki
 - ✦ Hardened, trimmed-down, commercially focussed version

2004-10-19

44

Avaki today

- Grid paradigm useful for dynamic business environment where companies regularly merge
 - May be geographically distributed, have heterogeneous architectures, require unified secure access
- What has changed from Legion to Avaki
 - Eliminates more esoteric features and functions
 - 2D files, heterogeneous MPI
 - Adds more stringent error-checking & recommends safer configurations
 - Increases usability through documentation, configuration guidelines, extra tools

2004-10-19

45

Differences between Avaki & OGSA

- RPC model
 - Avaki intends to address by becoming Web Services compliant i.e supporting XML/SOAP & WDSL
- Naming model
 - OGSA names have no security information
 - Binding resolvers in OGSA are location and protocol specific
- No security model in OGSA

2004-10-19

46

Users View

For more information including
access to on-line tutorials see:

<http://legion.virginia.edu>

First steps

- Setup shell environment variables

(ksh or sh users)

```
export LEGION=<Legion root dir path>
export LEGION_OPR=<Legion OPR root path>
(suggested: $LEGION/./OPR)
. $LEGION/bin/legion_env.sh
. $LEGION_OPR/legion_context_env.sh
```

- Specifies where binaries can be found and sets root context

Authentication (login)

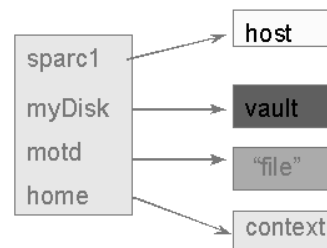
- `legion_login <user id>`
 - ◆ currently uses passwords - other mechanisms may be easily added
 - ◆ login object is the user's proxy to the world
 - ◆ login object generates a certificate
 - ◆ certificate identifies who you are

Legion (context) space

- Network-wide, transparent file system
- Programs can read/write files regardless of execution location
- Data files can easily be moved between Legion space and user's local file system
- I/O libraries for access
- All the usual utilities, `legion_cat`, `legion_ls`, `legion_ln`, `legion_cp`, ...

Contexts are more general than Unix directories

- Contexts map strings to LOIDs
- Directory-like - but can "point" to anything, not just other contexts and files
 - ◆ host objects
 - ◆ class objects
 - ◆ vaults
 - ◆ TTYs



Context Examples

```
$ legion_ls -l
.                (context)
class            (context)
hosts            (context)
vaults           (context)
home             (context)
$ legion_context_create /tmp
Creating context "tmp" in parent ".".
New context LOID = "1.01.05.608.003..."
$ legion_set_context /tmp
$ legion_ls -la
.                (context)
..               (context)
```

```
$ legion_cp -localsource myfile /tmp/myfile

$ legion_ls -la
.                (context)
..               (context)
myfile           1564 bytes
$
$ legion_ln myfile /ln_to_myfile
$ legion_ls -la /
.                (context)
class            (context)
hosts            (context)
vaults          (context)
home            (context)
tmp             (context)
ln_to_myfile     1564 bytes
```

```
$ legion_ls -la /hosts
.                (context)
..               (context)
BootstrapHost
bootstrap.host.DNS.name
stonesoup00.cs.virginia.edu
stonesoup03.cs.virginia.edu
```

Access Control

- Mayl implements access control lists on a function basis
- Users are named by their login object
- Sets of users defined by contexts with links to users

```
$ legion_change_permissions [+rwx] [-v] \
[-help] <group/user context path> \ <target
context path>
```

```
$legion_change_permissions +r /users/fred \
/home/grimshaw/myfile
```

Access Control

```
$legion_ls /users/grimshaw/mygroup
..
Tony
Sally
Tim

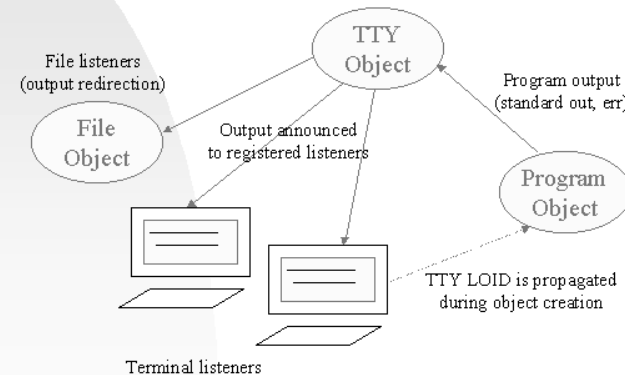
$legion_change_permissions +r \
/home/grimshaw/mygroup \
/home/grimshaw/myfile
```

Unified Console

- Run-time output can flow back to a central (or multiple) consoles
- Can dynamically connect and disconnect
- Faster, simpler debugging
- Easy monitoring of status, errors, etc.
- share console with others
- Commands:

```
legion_tty /mytty  
legion_set_tty /mytty  
legion_tty_watch /mytty &
```

TTY Objects in Action



Running sequential applications

Running sequential applications

- Can run Legion-aware or legacy programs
 - ◆ Primary distinction is the use of the Legion I/O and context Libraries.
- Register programs with a Legion class object
- Execute the program remotely using legion commands

Registering the binary

```
$ legion_register_program printLoc \
/bin/hostname linux
Creating class "printLoc"
Registering implementation for class "printLoc"
```

- Creates class, if necessary
- Register versions of program for multiple platforms with same class

Remote execution

```
legion_run [<options>] <prog_path> [<args>]
```

- non-Legion binaries and shell scripts
- selects a host for remote execution
- copies binary if necessary (system caches)
- copies files in and out as specified
- simplest way to do parameter space studies

Example 1

```
$ legion_tty myTTY
$ legion_run printLoc
stonesoup05.cs.virginia.edu
$ legion_run -h /hosts/stonesoup00 printLoc
stonesoup00.cs.virginia.edu
$ legion_run -a alpha_linux printLoc
centurion019.cs.virginia.edu
```

- Can select a host or platform type

Parameter space studies

- Perform multiple executions of a sequential program (e.g., simulation), each with different data
- E.g., examine changes in simulation results under different parameter values
- Simulation of each point in the parameter space can execute in parallel
- Natural application of Legion remote execution

legion_run_multi

```
legion_run_multi [<options>] -n <num nodes>  
-f <control_file> <prog_path> [<args>]
```

- Manages multiple legion_run sessions, providing easy-to-configure parameter space studies
- Control file uses simple language for specifying input/output file name patterns
- Provides support for directing scheduling of remote jobs

Example 1

- Run eight copies of my_prog on input files data.[1-8] producing output files results.[1-8]

```
$ legion_register_program my_prog my_prog sgi  
$ cat dataset.cfg  
  in data data.*  
  out results  
$ ls data.*  
data.1 data.2 data.3 data.4  
data.5 data.6 data.7 data.8  
$ legion_run_multi -n 8 -f dataset.cfg \  \  
  my_prog data -o results  
$ ls results.*  
results.1 results.2 results.3 results.4  
results.5 results.6 results.7 results.8
```

Example 2

- Wish to select the hosts to run on

```
$ cat hfile  
/hosts/centurion001    3  
/hosts/centurion002    3  
/hosts/stonesoup01     1  
/hosts/stonesoup02     1  
  
$ legion_run_multi -s hfile -n 8 -f dataset.cfg \  
  my_prog data -o results
```

- Place 3 tasks on each of the centurion hosts, and 1 on each of the stonesoup hosts.

Legion-aware sequential applications

- Can modify existing sequential applications to use Legion I/O and context libraries

Legion I/O

- A number of supported interfaces
 - ◆ C/C++ BasicFiles library interface
 - ◆ Fortran BasicFiles library interface
 - + Simple copy-in/copy-out routines
 - + Standard file access routines
 - ◆ C++ object-oriented interface
 - ◆ Java object-oriented interface

C interface example

```
long fd;
char buf[1024];

fd = BasicFiles_open(path, BASIC_FILE_O_CREAT |
                    BASIC_FILE_O_TRUNC);

BasicFiles_write(fd, buf, 1024);

BasicFiles_seek(fd, BASIC_FILES_SEEK_BEGINNING, 512);
```

Linkage requirements

- Link against Legion libraries
- Sample makefile:

```
CFLAGS = -I$(LEGION)/include \
        -L$(LEGION)/lib/$LEGION_ARCH/$(CC)
LIB      = -lBasicFiles -lLegion
example: example.c
        $(CC) $(CFLAGS) example.c -o example $(LIB)
```

- or use legion_link:

```
CFLAGS = -I$(LEGION)/include
example: example.c
        $(CC) $(CFLAGS) -c example.c
        legion_link example.o -o example
```