Large-Scale Systems (2): Legion

- Legion vision:
  Metasystem consisting of **millions** of hosts, **billions** of objects, co-
existing in a **loose confederation** tied together with high-speed links.

- Legion objectives

- Legion object model

- Legion home page:
  http://www.cs.virginia.edu/~legion

- Reading:
  - A.S. Grimshaw, Wm. A. Wulf. “Legion. The Next Logical Step Toward
    the World-Wide Virtual Computer”
    (http://legion.virginia.edu/copy-cacm.html)
    (http://www.cs.virginia.edu/~legion/copy-core.html)

Legion Objectives (1)

- Site autonomy
  Organizations want to keep juristictional boundaries in place.

- Extensible core
  Allow users to construct their own mechanisms and policies.

- Scalable architecture
  No centralized structures

- Easy-to-use, seamless computational environment
  Legion must mask the complexity of the hardware environment and of
  communication and synchronization of parallel processing.

- High performance via parallelism
  e.g. task and data parallelism

- Single, persistent name space
  Single name space for file and data access.
Legion Objectives (2)

- Security for users and resource owners
  Cannot strengthen existing OS protection and security mechanisms. Existing mechanisms should not be weakened by Legion. Need mechanisms for users to manage security needs.

- Management and exploitation of resource heterogeneity
  Inter-operability between heterogeneous hardware and software components.

- Multiple language support and inter-operability
  Integrate heterogeneous source language applications, support for legacy codes.

- Fault-tolerance
  Dealing with failure and dynamic re-configuration

Constraints

... *cannot replace host operating systems*
  Operate at middleware level.

... *cannot legislate changes to the interconnection network*
  Can layer better protocols over existing ones.

... *cannot require that Legion run as “root”*
  Most Legion users want it to run with the least possible privileges.
The Core Legion Object Model

- Each object belongs to class; each class is itself an object.

- **Object-mandatory** member functions:
  - `may_I()`, `save_state()`, `restore_state()`
- **Class-mandatory** member functions:
  - `create()`, `derive()`, `inherit_from()`

- User-level class objects responsible for managing instances and subclasses:
  - creation, location, security policies, object placement policies

Security

- Various users can have wide variety of security concerns.

- Provide users with mechanisms to build robust security measures.

- **Message layer:**
  - Inter-object communication and authentication
- **Discretionary layer:**
  - Access control predicate `may_I();` must be called before invoking any method.
- **Mandatory layer:**
  - Legion objects can act as Security Agents; monitor other objects.
Naming and Binding

- **LOID**: Every Legion object is named by a Legion Object Identifier.

<table>
<thead>
<tr>
<th>Format</th>
<th>Class Identifier</th>
<th>Instance Number</th>
<th>Public Key</th>
</tr>
</thead>
</table>

- Format: defines size, format of other fields
- Class identifier: handled by LegionClass
- Instance number: handled by class object
- Public key: allows entire LOID to be used as public key for object.

- Binding: LOIDs have meaning only at Legion level. Need to be bound to names that have meaning at underlying protocol levels.

Binding

- Need physical **Object Address** to communicate with another object.

- Binding: (LOID, Object Address, invalidation time); first class entities.

- Binding Agents: derived from **LegionBindingAgent**.
  - `binding get_binding(LOID); binding get_binding(binding)`
  - `invalidate_binding(LOID); invalidate_binding(binding)`
  - `add_binding(binding)`
Object States

- **Active** objects are running as process on Legion Host and can be accessed via an Object Address.
- **Inert** objects exist in persistent storage
  - Controlled by **Legion Vault**
  - Described by Object Persistent Representation (OPR)
  - Located using Object Persistent Address (OPA)
- **Object Persistent Representation:**
  - Every object exports `save_state()` and `restore_state()` methods.
  - Objects can carry state when they migrate between hosts.
- **Object Persistent Address:**
  - Analogous to Object Address of active object.
  - Typically file name, meaningful to Legion Vault.

Class Objects

- Class objects export class-mandatory member functions to
  - create new instances: `create()`
  - create new subclasses: `derive()`
  - delete instances: `delete()`
  - find instances and subclasses: `get_binding()`
- Assigns LOID to instances and subclasses
  - For new instance:
    • assign Class Identifier to match own
    • assign Instance Number as it sees fit
  - For new subclass:
    • Contact **LegionClass** to obtain new Class Identifier
- Logically maintain table for objects:
  - LOID, Object Address, Placement Mapper, Current Vault Set, Candidate Vault Set.
Mechanisms: Binding

- Model:
  - Ultimatively, responsibility for providing bindings lays with class.
  - To support scalability, other objects may be involved as well.

  
  \[ \text{C.get\_binding}(B) \]

  
  \[ \text{BA.get\_binding}(B) \]

- How to find responsible class object?

Scalability

- **Scalable Architecture:**
  “As the number of processors increases, the granularity of computation does not need to increase to keep the machine balanced.”
  - e.g. hypercubes, meshes, tori, rings.
  - Problem: Scalability of architecture must be claimed with respect to particular application

- **Distributed Systems Principle:**
  “The number of requests to any particular system component must not be an increasing function of the number of hosts or objects in the system.”