

## DSM Case Study I:

### Java/DSM A Platform for Heterogeneous Computing

W. Yu, A. Cox  
Department of Computer Science  
Rice University  
July, 1997

### Java/DSM: Introduction

- DSM (as opposed to, e.g., message passing) allows to focus on algorithmic issues instead than on managing partitioned data sets.
- Java RMI = RPC + support for object references across machines
- However:
  - Programmer still needs to manage coherency of replicated data
  - Programmer still needs to fine-tune remote interfaces

## Java/DSM: Motivation

- Systems for heterogeneous computing environments:
  - **Shared memory abstraction** : (Agora, Mermaid)
    - Mermaid based on IVY DSM system; supports C language.
    - Hardware differences are exposed to programmer (padding and ordering of aggregate structures)
    - Mermaid automatically performs format conversions between machine, but requires non-standard compiler support.
  - **Message passing**
    - MPI, PVM, etc...
    - PVM: `send()`/`receive()` primitives, and programmer needs to marshal data using `pack()`/`unpack()` routines.
    - Little/no support for sharing of complex data structures.
  - **RPC/RMI**
    - Programmer must decide when/to whom/what to communicate

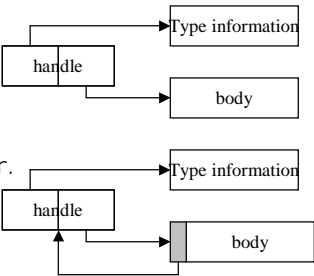
## Java/DSM: User Interface

- One Java VM per host.
- Java VM similar to JDK Java VM, except that all objects are allocated in *shared memory* region.
- Threads cannot migrate.
- RMI :
  - Remote object can only be accessed through remote interface methods.
- Java/DSM:
  - Remote objects are accessed just as local objects.
  - Static variables are shared by all objects in the system.

## Java/DSM: Memory Management

- Do not want to maintain entire object reference graph across all machines.
- Instead, have each machine perform garbage collection independently.
- However, need to be careful that objects referenced only by remote machines are not prematurely reclaimed.
- Java/DSM garbage collector:
  - Each collector maintains an export list, and an import list:
  - **Export list**: remote references to locally created objects. Maintained by parsing all outgoing messages.
  - **Import list**: local references to remote objects. Maintained by parsing incoming messages.
- During GC:
  - Exported references are treated as root set of references.
  - Imported references that are not marked are sent back to their owners.
  - Owners use reference counting to decide when a reference can be removed from the export list.
- Occasionally, synchronized collection is needed to reclaim cyclic structures.

## Java/DSM: Data Conversion

- When a data item is passed between machines, data conversion is required.
- Type lookup:
  - Objects in JDK-1.0.2: handle & body
- Add back pointer from body to handler.
 
- Require all objects allocated from the same page to be of same size.
- Given an address, can identify page number, look up size of objects within the page, find beginning of object, and follow back pointer to find type information.
- Special problem when a data item crosses a page boundary.

## Java/DSM: Data Conversion

```
page_number = (page_start - first_shared_page) / page_size;
object_size = page_info[page_number].size;
/* size of objects in this page. */
addr = page_start;
while (addr < next_page) {
    back_pointer_address = addr + back_pointer_offset;
    back_pointer = (JHandle*)(*(long*) back_pointer_address);
    class = back_pointer -> class;

    convert_pointer(back_pointer_address);
    addr += sizeof(long*); /* locate the start of data. */
    for (i = 0; i < class->fields_count; i++) {
        type_code = class->fields[i].signature;
        convert(addr, type_code);
        addr += field_length(type_code);
    }
}
```

## Java/DSM: Changes to JVM

- The heap is allocated using ThreadMarks' shared memory allocation routine
- Classes loaded by the JVM are put in shared memory. The loading is synchronized.
- During garbage collection, the GC must recognize objects created by remote machines, mark those references locally, and send the unused remote references to their owners after the collection.
- To support data conversion, we require a mapping from an arbitrary address to the object's handle. The straightforward solution is to require that all objects on a page are the same size, and put a pointer to the handle at the beginning of every object.
- Most of the changes are in the memory management module. Rest of JVM is virtually unchanged.

## Example Application: Distributed Spreadsheet

- Distributed spreadsheet based on public domain *sc*.
- Extensions for distributed operation:
  - Define regions of cells.
  - Assign read/write privileges for each region to users.
  - Lock a region to prevent others from writing it.
  - Check if other users have made changes.
  - Incorporate changes made by other users.
- Internal data structure:
  - Cells are organized in a two-dimensional array.
  - Each cell entry contains information such as its type, current value, printable form, and a lock bit.
  - If entry is expression, this is stored in form of an E-tree (expression tree) with references to other cells.

## Distributed SpreadSheet: Realization

- DSM (simple!)
  - Identify race conditions
  - Add synchronization
- RMI
  - Central server gives poor performance
    - Server does not scale well (hot spot)
    - Interface to remote objects must be fine tuned (e.g. must be able to aggregate requests to clusters of cells)
  - Replicated data needs support for coherence
    - Update scheme: changes to the spreadsheet are actively propagated to others. Simple, but has much overhead.
    - Invalidate scheme: short invalidation messages are sent instead of the changed data. Higher performance, but requires complicated timestamp mechanism to track ordering of events.

# Problem: Reference Marshaling in RMI Version

Example: cell A[0] contains expression

$$A[0] = A[2] + A[4] * A[6]$$

