1. Introduction
Spring is a microkernel OS with two components executing in kernel mode (the virtual memory manager and the microkernel itself known at nucleus). The paper focuses on the Spring fast IPC.

2. The Spring IPC model
Spring is an object-oriented system that does not attempt to dictate how objects should be implemented. Spring however provides certain commonly desirable properties to its users, among which:

- secure access to services: to avoid the need for user-level authentication the Spring nucleus provides kernel-managed capabilities;
- easy recovery in the presence of client or server crashes, and
- a highly efficient IPC mechanism particularly for the most common case where the number of arguments and results is small.

Doors are the basic IPC mechanism in Spring. A door represents an entry point for a cross-domain call. Associated with the door are (a) the address of an entry point in the target domain and (b) an integer datum that can be used to specify an individual object in the target domain.

Whenever the nucleus executes a door call, it stores the caller domain, the PC address to which to return the call and the current stack value of the caller.

Each domain has a table of doors to which the domain has access. To prevent tampering, Spring stores these door tickets or capabilities inside the nucleus.

Doors are destroyed when no domain can access them. Since each domain always maintains a link to each door, it implements doors, a door becomes “unreferenced” when its reference count is equal to one. To prevent race conditions between incoming calls and the door being unreferenced, the door reference count is incremented each time a call is send through the door and decremented each time a call returns.

3. The Spring thread model
For performance reasons, the thread issuing a cross-domain call and all downstream threads should be treated as a single scheduling entity. However, merging these threads into a single thread managed by the calling domain would cause problems in two cases:

a) when the domain being called resides on another machine, and
b) when the calling domain wants to suspend a thread and the thread is inside a critical section of the target domain.

Hence Spring groups all threads directly or indirectly involved in a cross-domain call into a shuttle that form a single scheduling entity.

Spring applications can explicitly create pools of server threads. Whenever the nucleus executes a cross-domain call, it tries to locate an idle thread within the target domain. If it succeeds, the nucleus delivers the data into the data buffer of the thread and gives the CPU to the thread. To allow target domains to create new threads whenever needed, Spring includes a system call that blocks until no available threads can be found in the domain.

Spring treats failures of target domains as involuntary returns and returns an error code to the calling thread. Failures of calling threads result in the propagation downstream of an alert bit that can be caught by the threads executing the call.

Whenever a thread in the middle of a call chain crashes, Spring breaks the call chain and creates two distinct shuttles to provide for faster notification of the threads that are upstream of the thread that crashed.

4. Details of door invocation
Spring provides three implementations of the door invocation:

a) a fast path that supports the case when all door arguments are simple values and total less than 16 bytes (over 83% of all calls and over 70% of all returns); the nucleus executes roughly 60 SPARC instructions for a fast call and 40 SPARC instructions for a fast return; one of the tricks being used is masking registers instead of saving them;

b) a vanilla path that copies the argument data into the target domain and moves any argument doors across;

c) a bulk path that is used to transfer large quantities of page-aligned data using Spring’s virtual memory services.

5. Performance
Thanks to fast paths, Spring cross-domain call times are roughly one sixth of those of Mach and NT, 60% of those of Taos LRPC and less than 2% of those achieved by Berkeley sockets.

Even faster times are possible but it would require ignoring security and debugging issues.