The JX Operating System

Michael Golm
Meik Felser, Christian Wawersich, Jürgen Kleinöder

University of Erlangen-Nürnberg
Department of Computer Science
(Distributed Systems and Operating Systems)
Martensstraße 1
91058 Erlangen, Germany
golm@cs.fau.de

A Dancing Bear

This is a "dancing bear" paper -- it's not how well the bear dances, but that it dances at all.
And this bear dances.

Anonymous Reviewer

Outline

- Motivation
- JX Architecture
  - Protection domains
  - Communication mechanism
  - The Microkernel
- System-level programming in Java
- Performance

Why another OS?

Abstraction levels in software engineering

Assembler

↓ portability, productivity, robustness

C

↓ portability, productivity, robustness

Smalltalk, Java, C#

Why another OS?

Abstraction levels in software engineering

Assembler

↓ portability
productivity
robustness

C

portability
productivity
robustness ↓

Smalltalk, Java, C#

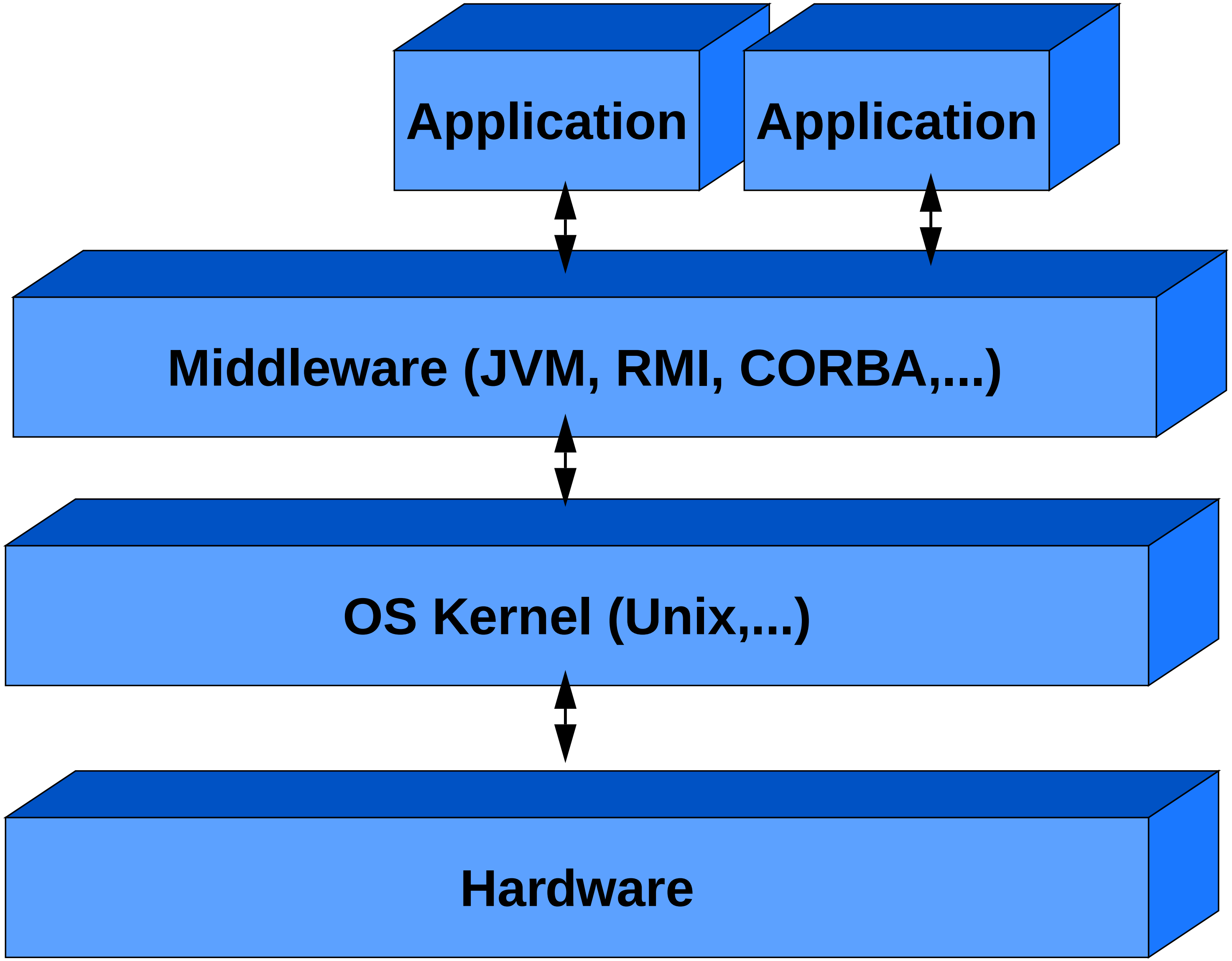
↓ ?

?

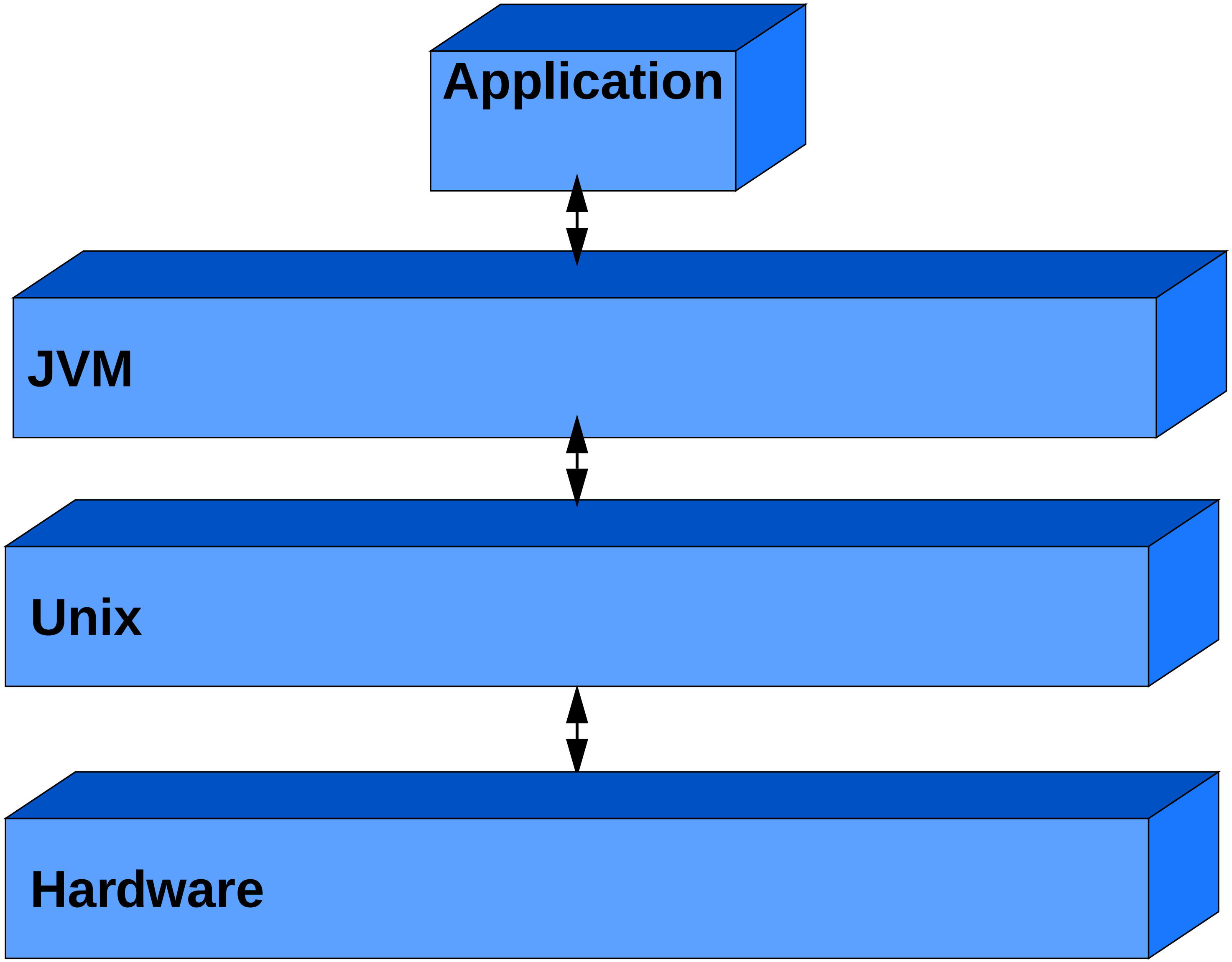
Applications

Operating Systems

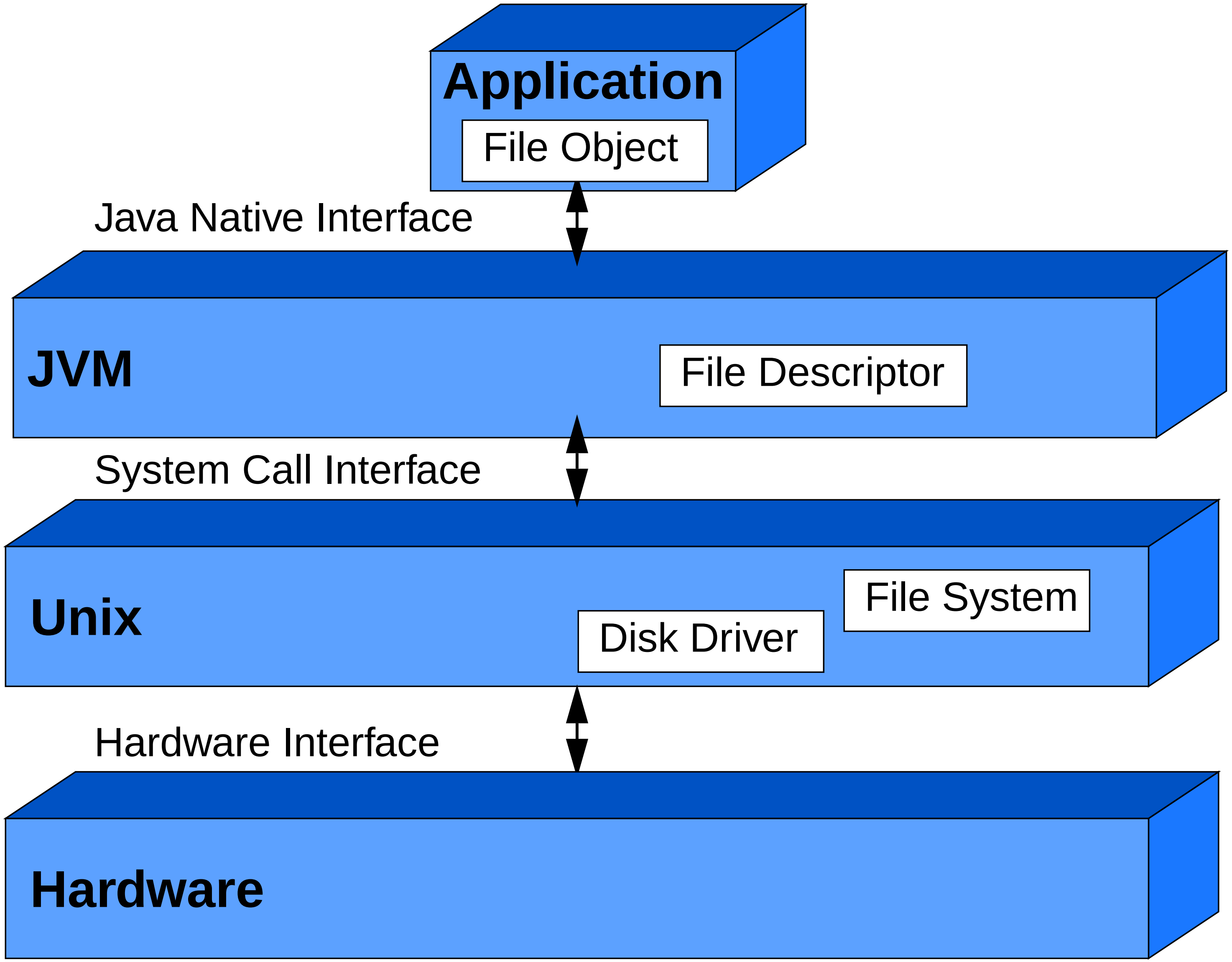
Typical System Structure



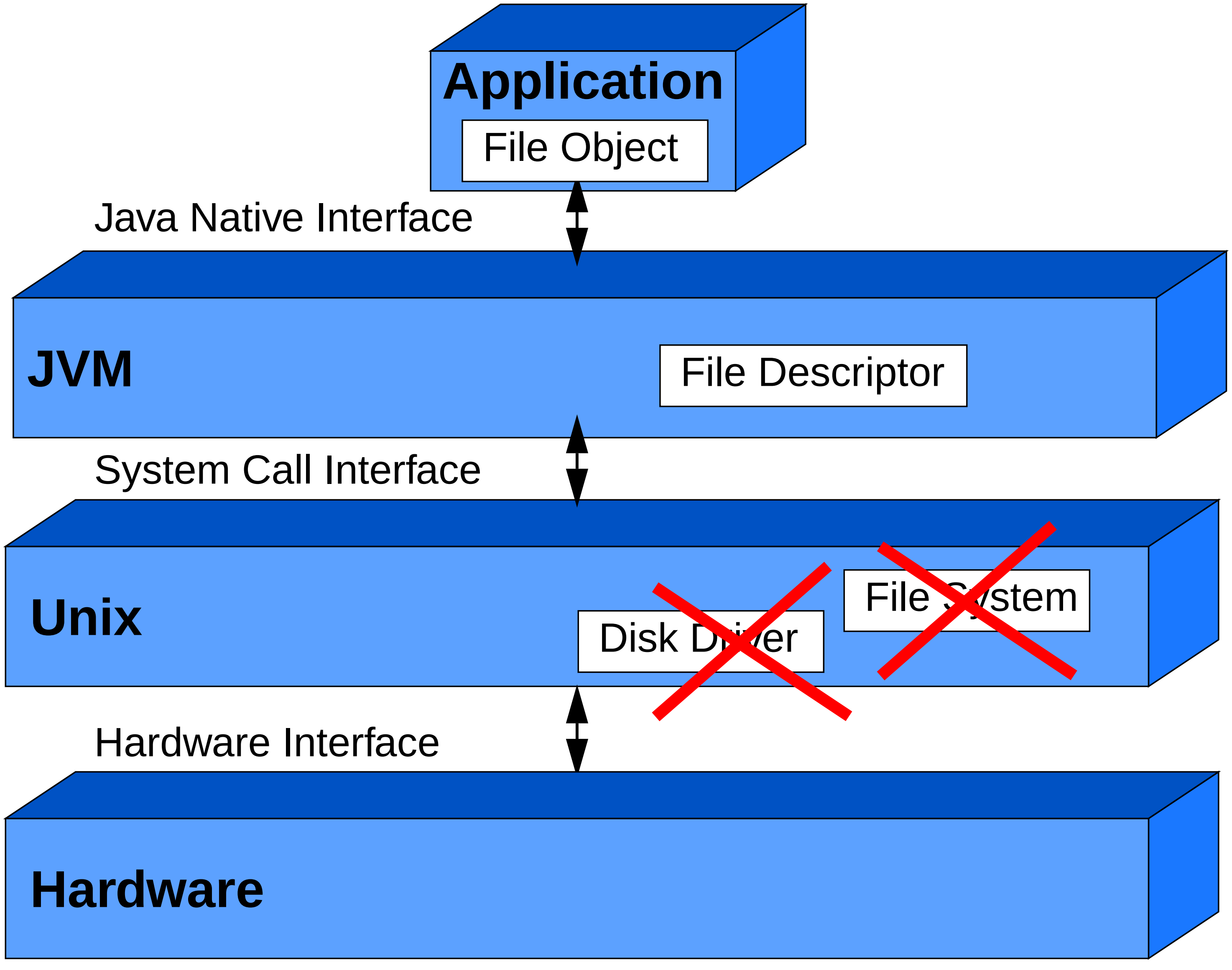
Typical System Structure



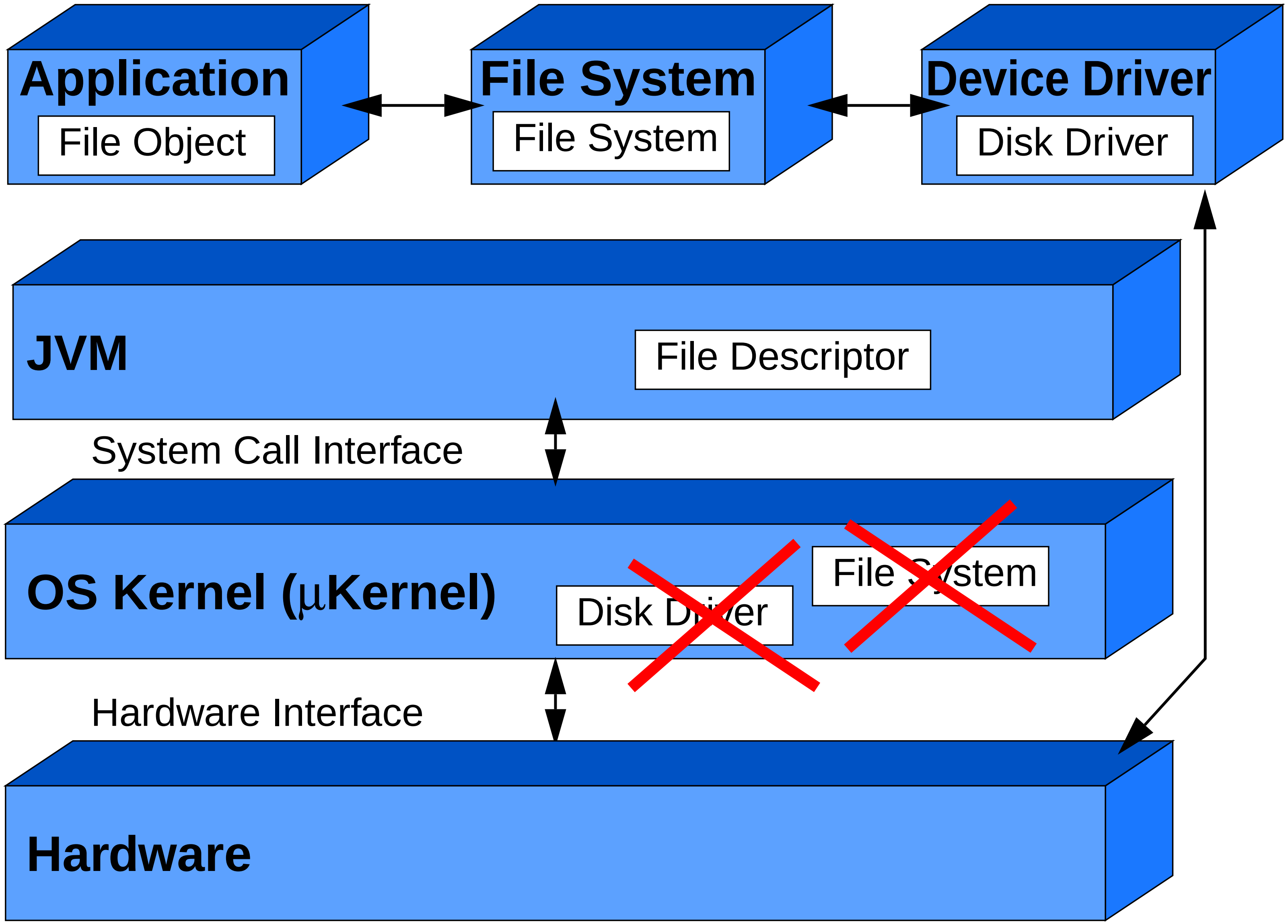
Typical System Structure



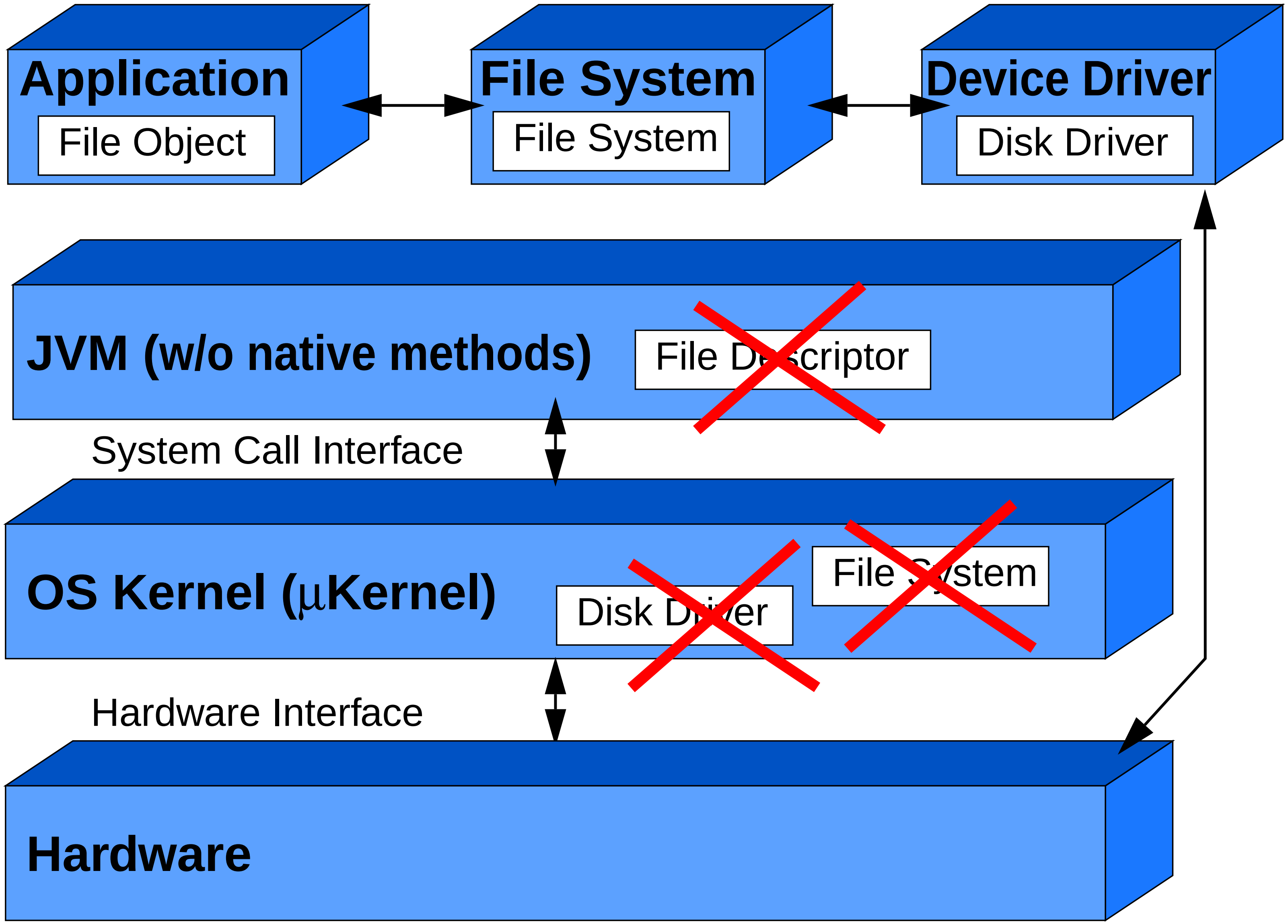
The Way to a new Architecture



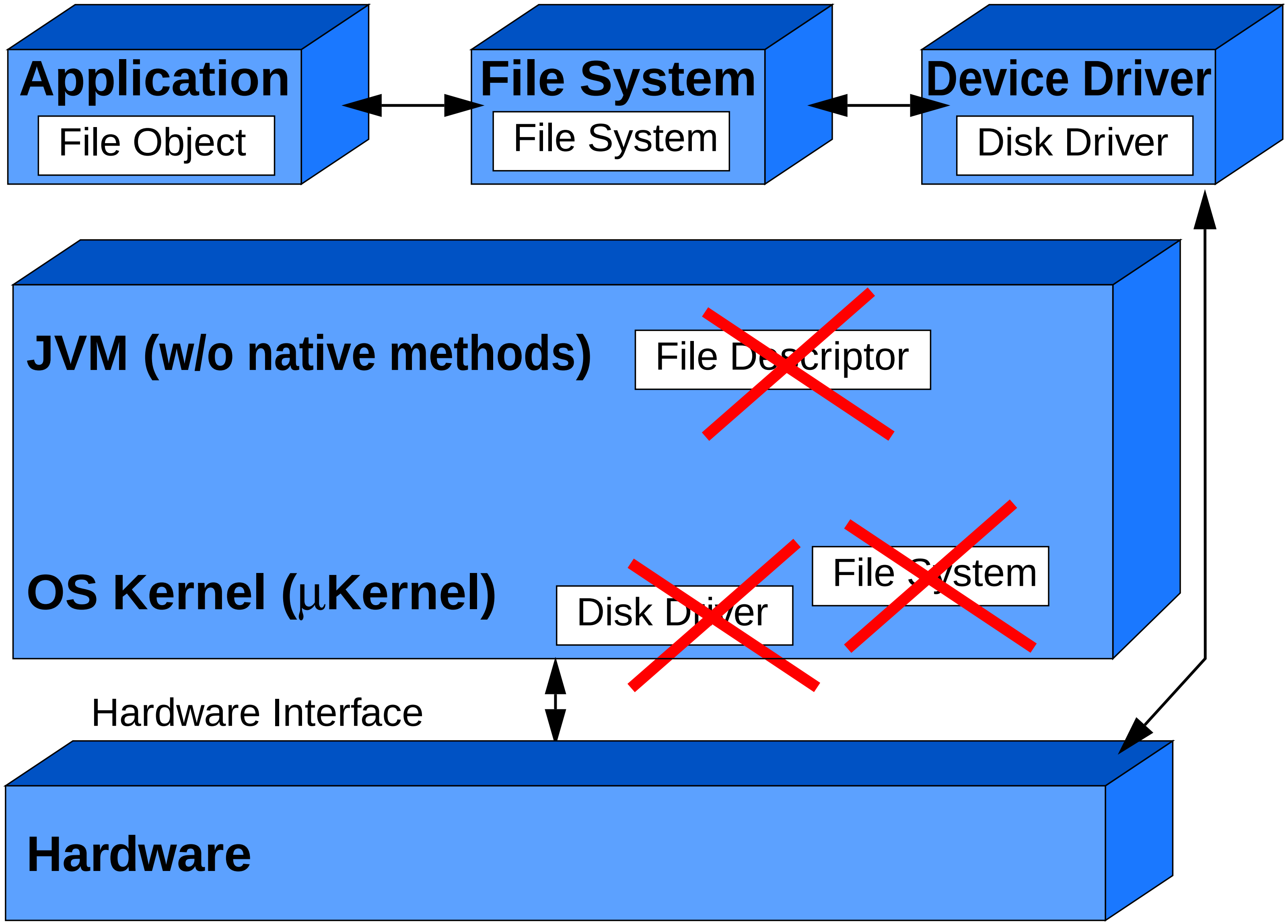
The Way to a new Architecture



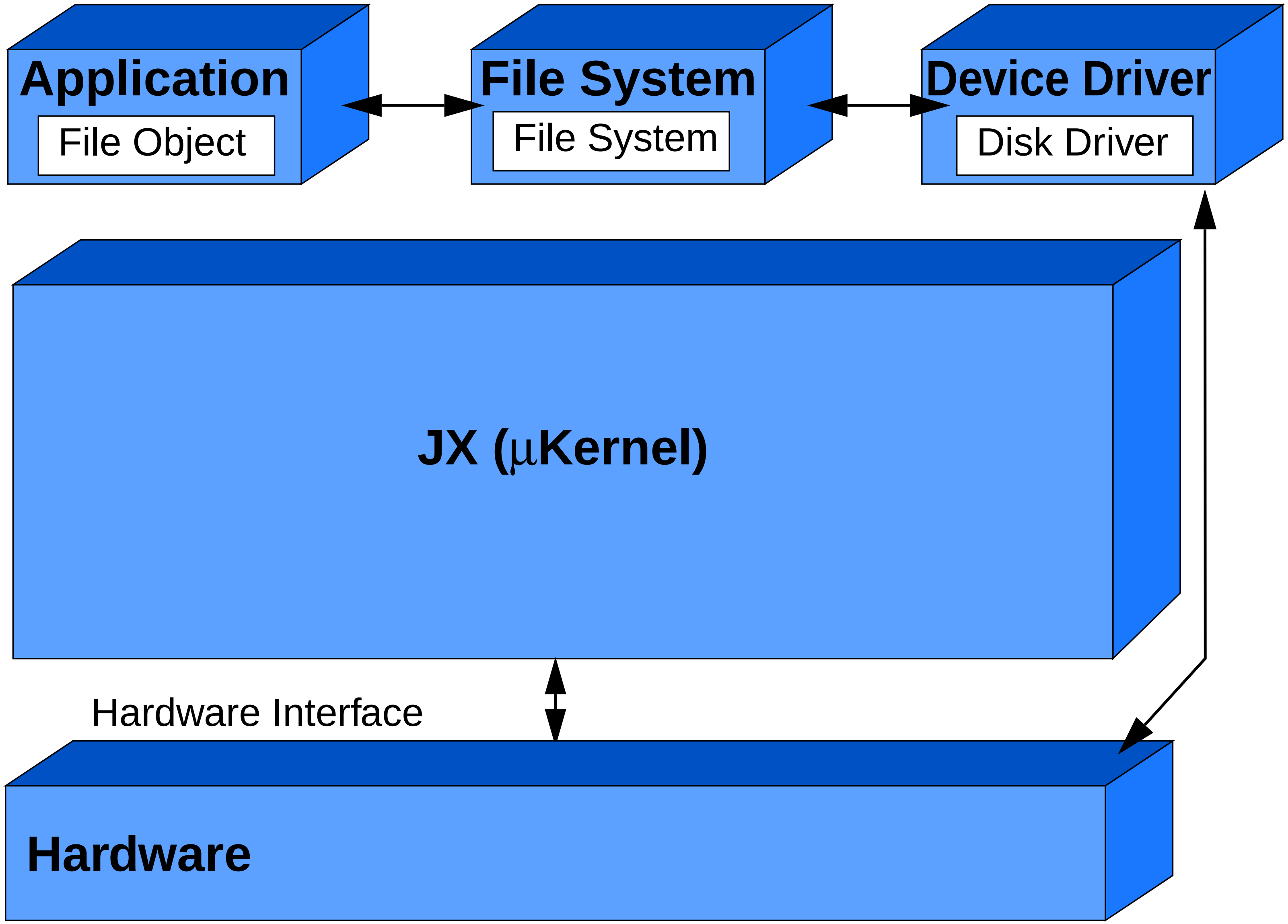
The Way to a new Architecture



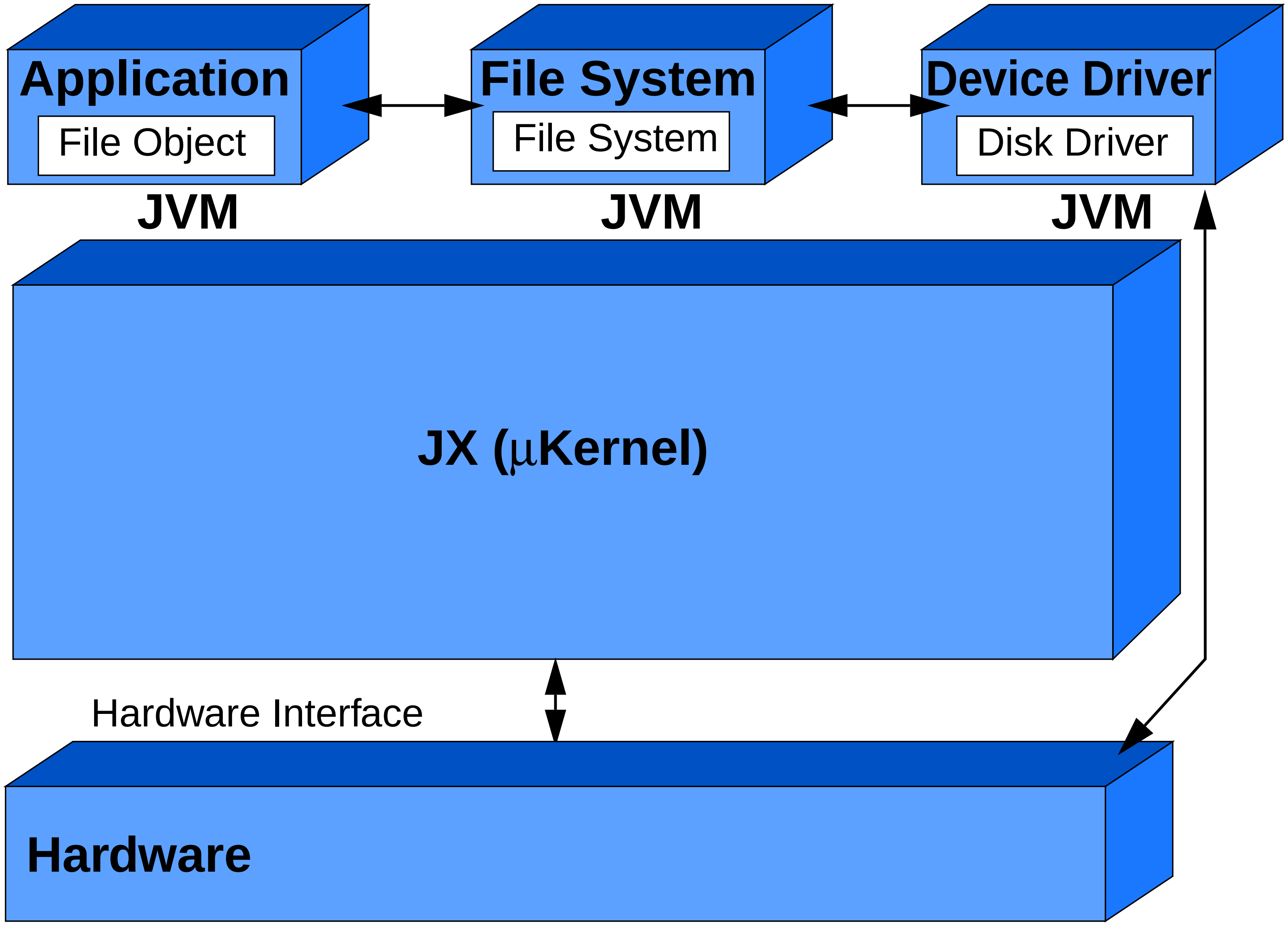
The Way to a new Architecture



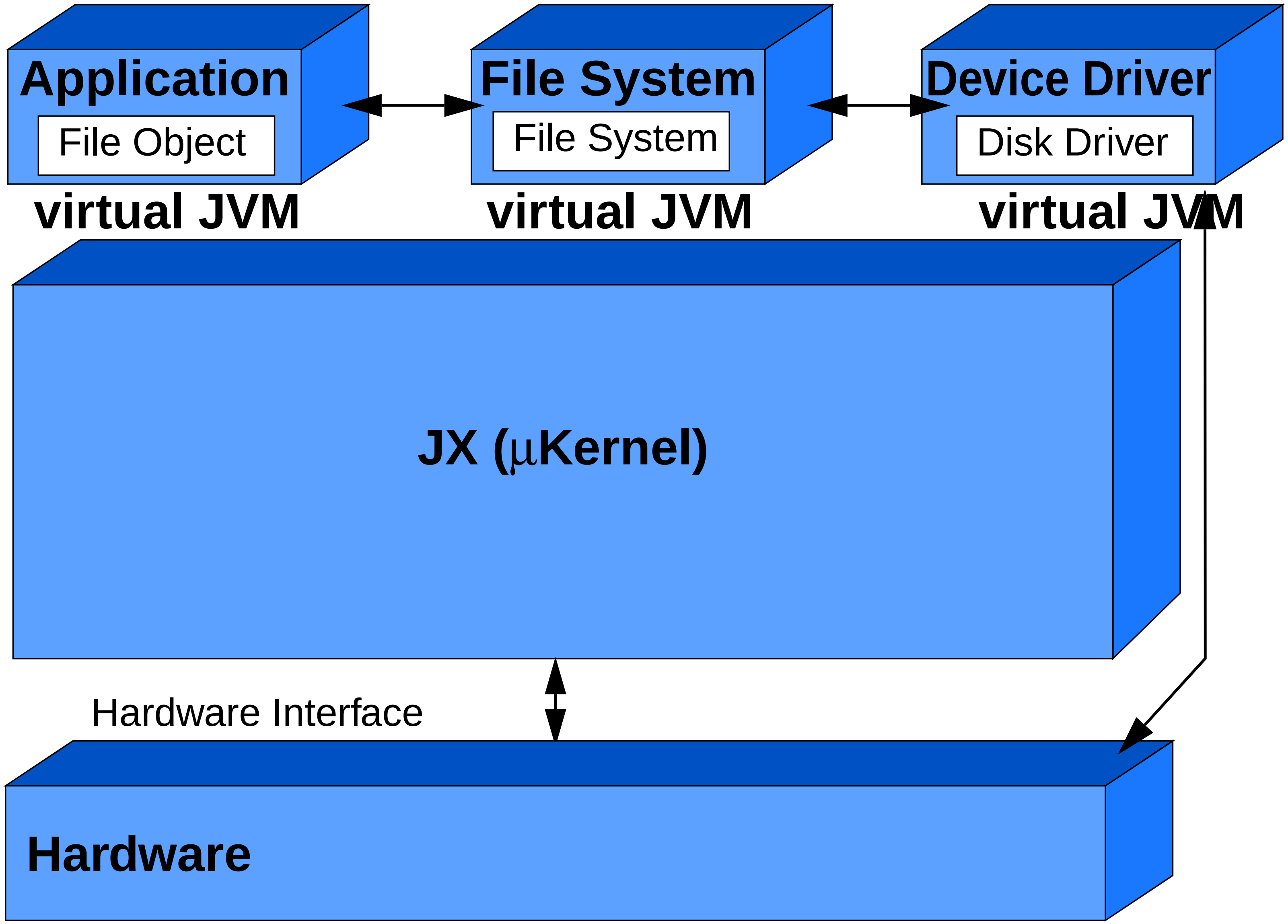
The JX Operating System



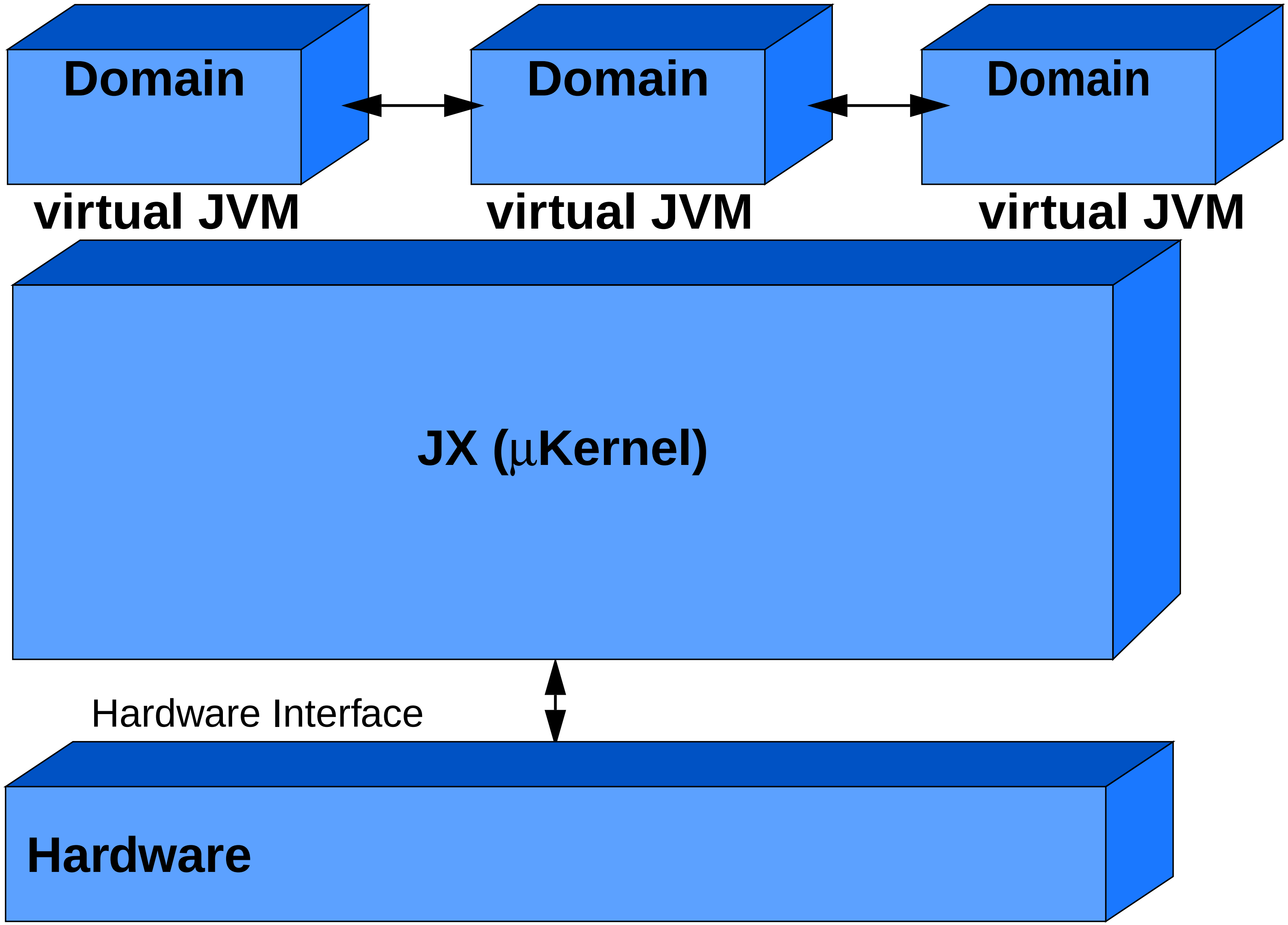
The JX Operating System



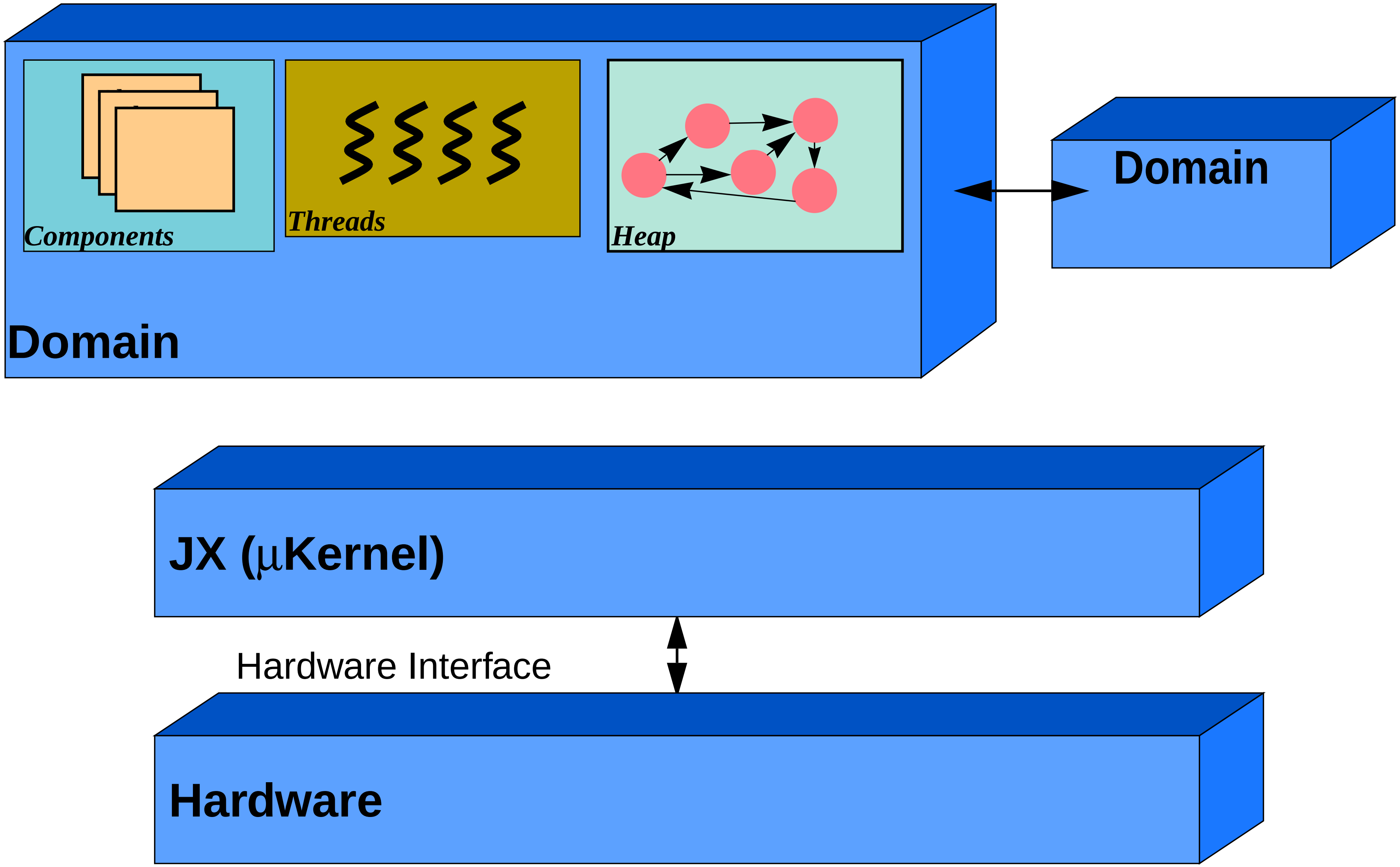
The JX Operating System



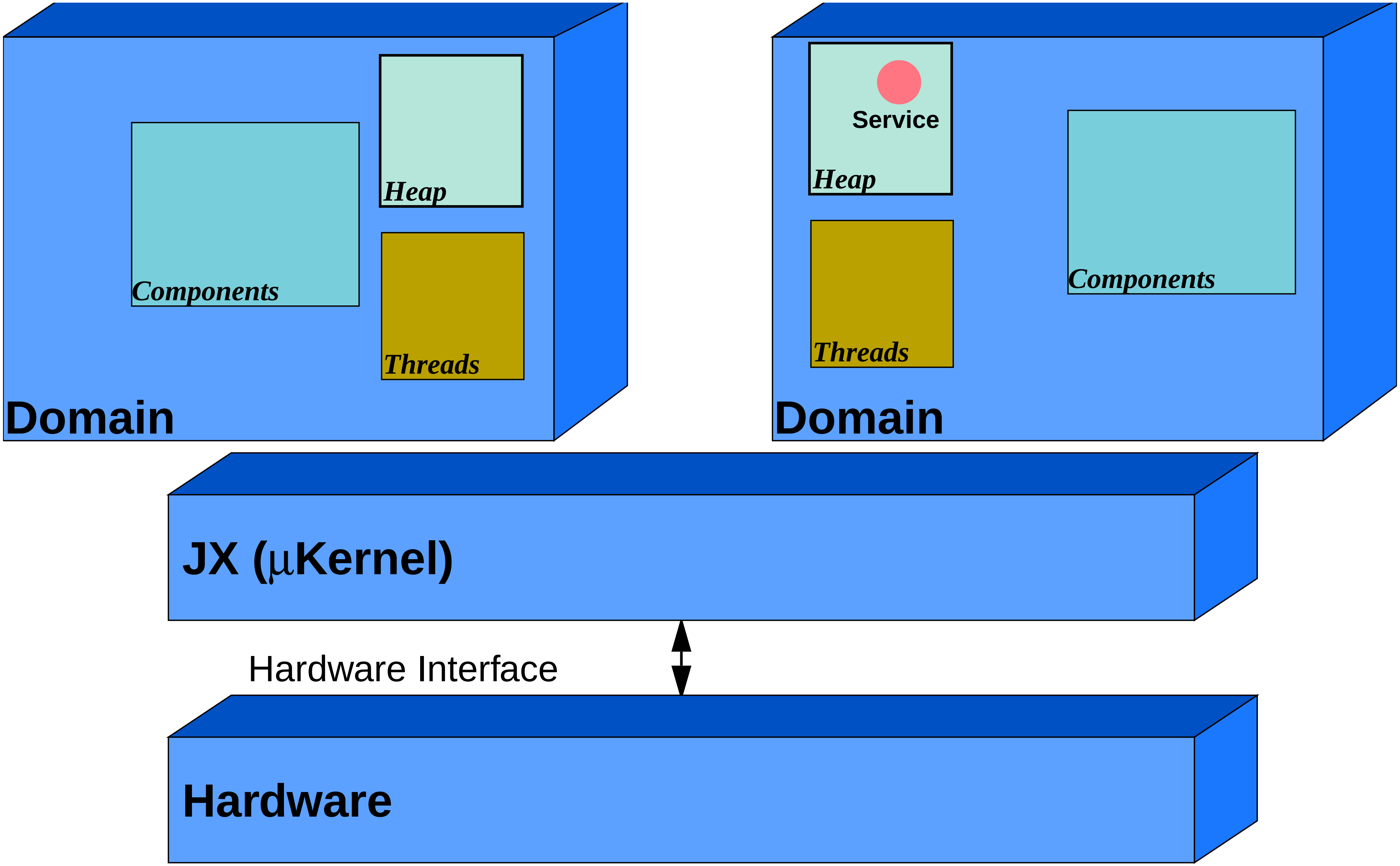
Protection Domains



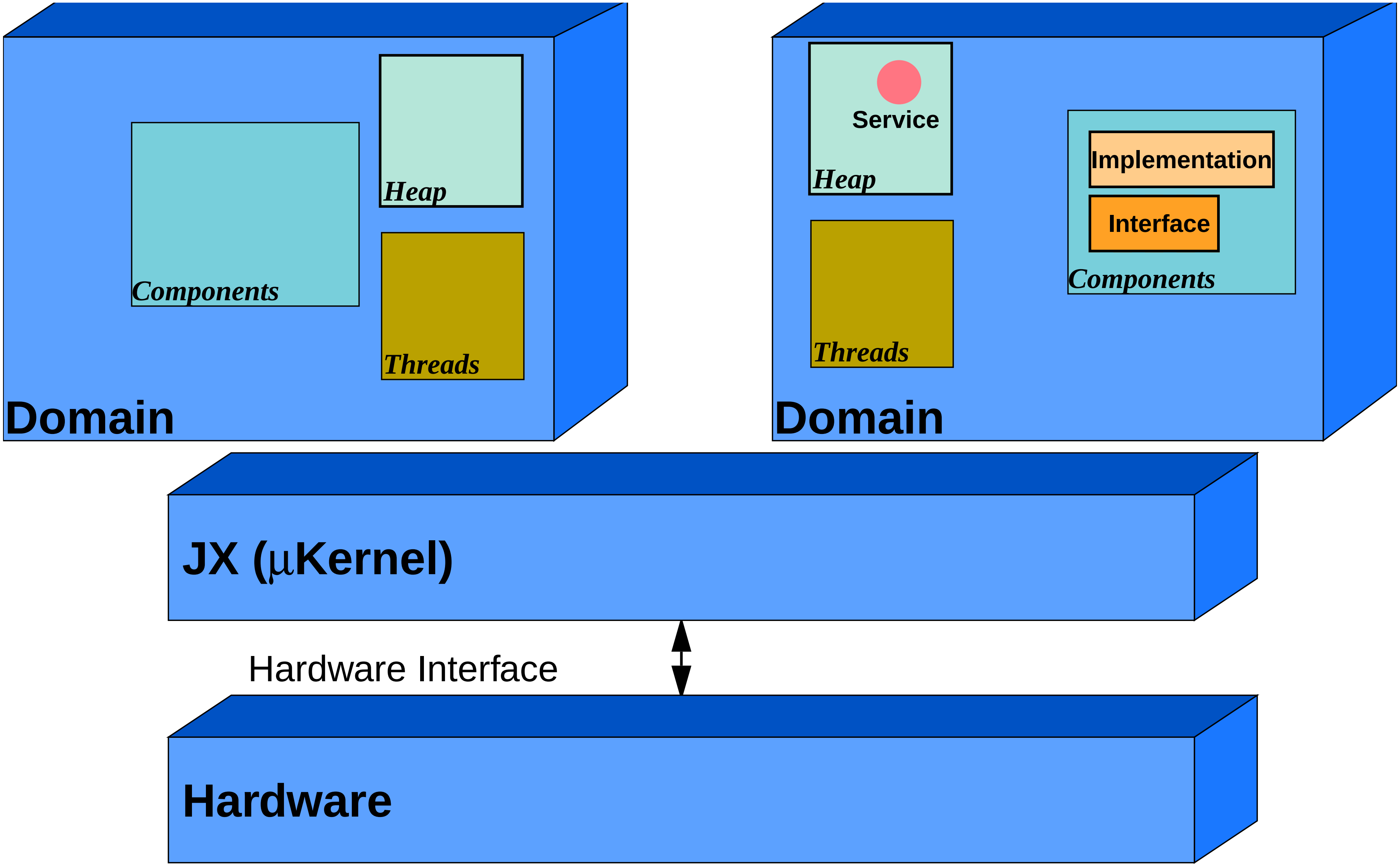
Protection Domains



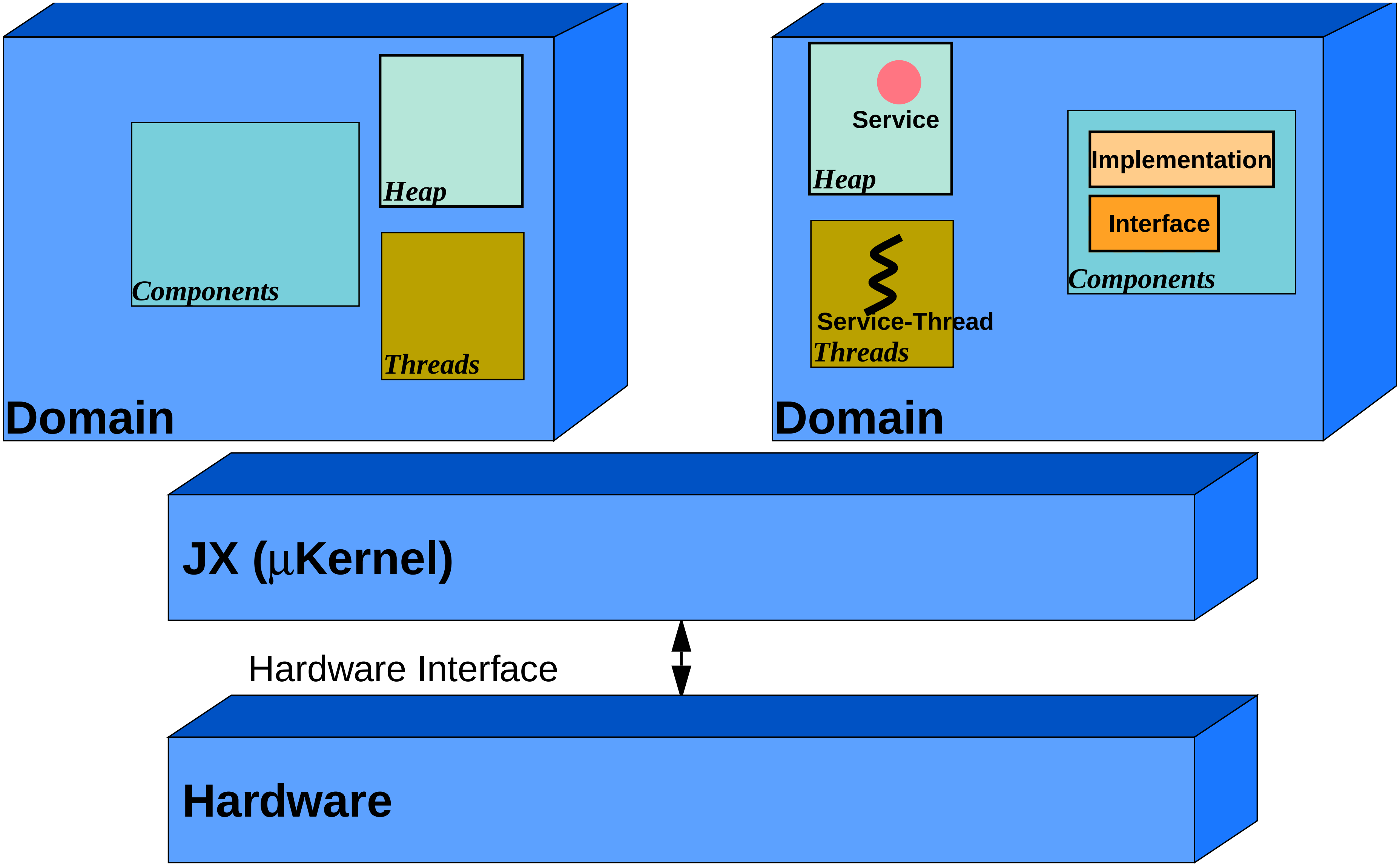
Communication: Portals



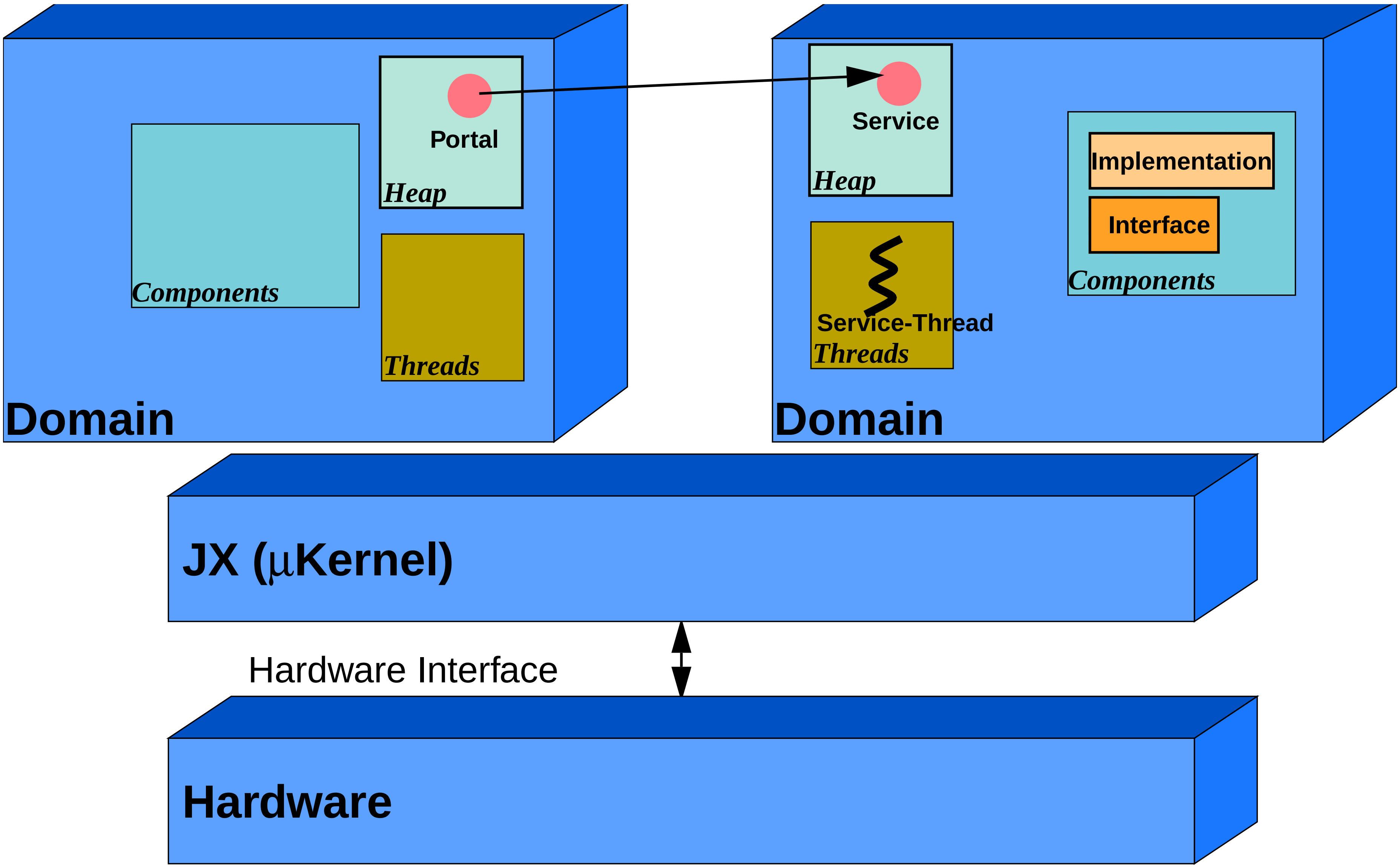
Communication: Portals



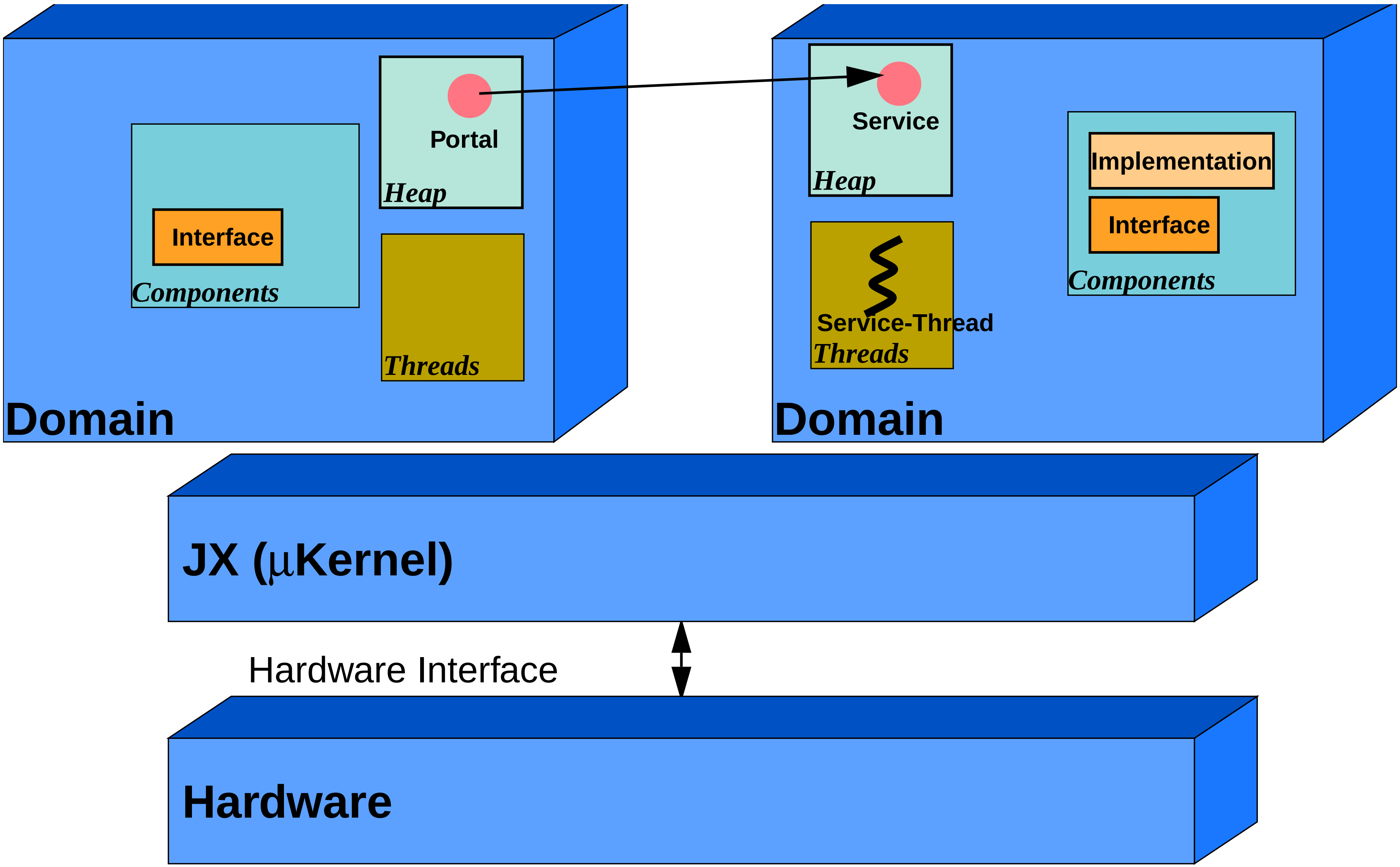
Communication: Portals



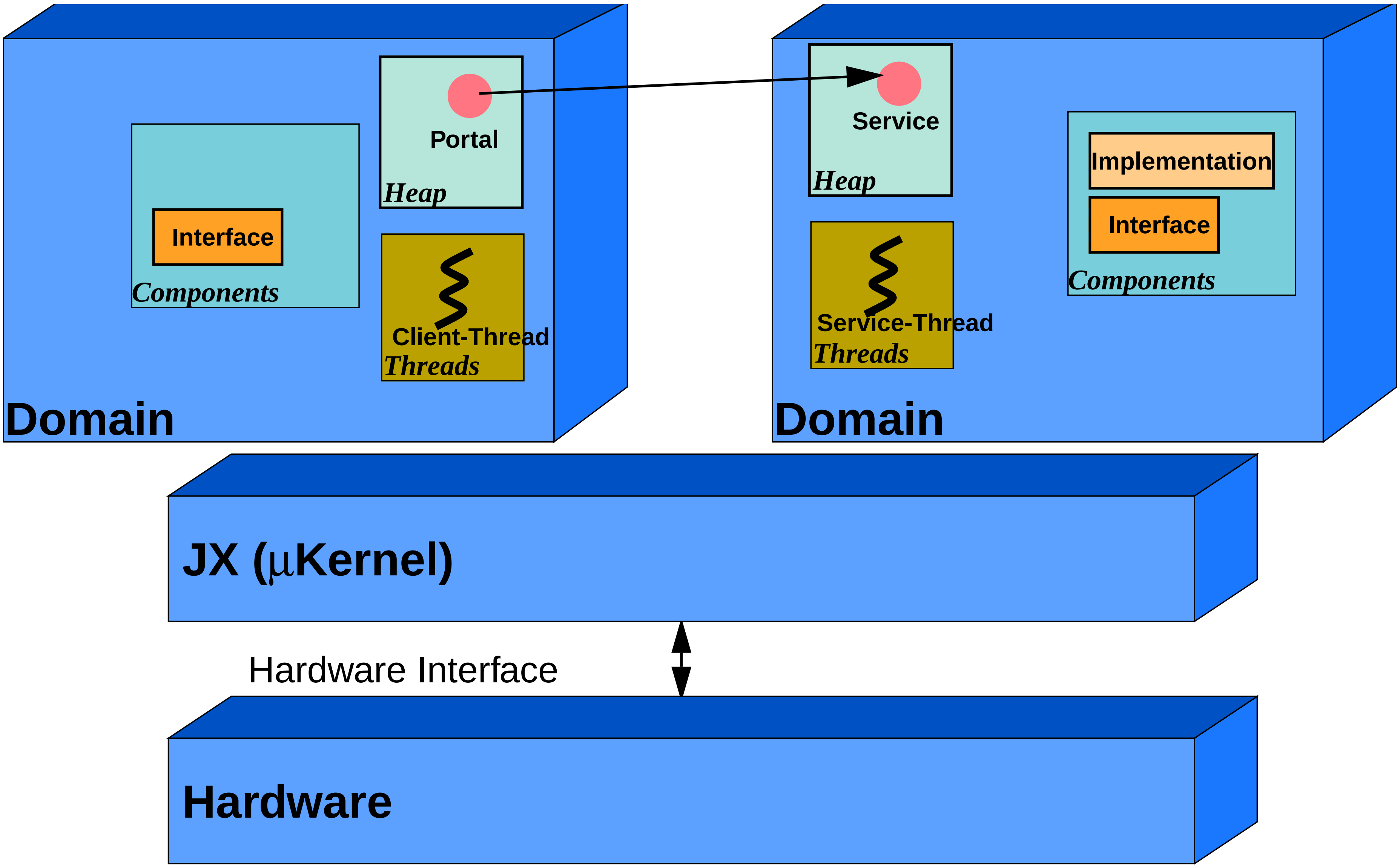
Communication: Portals



Communication: Portals



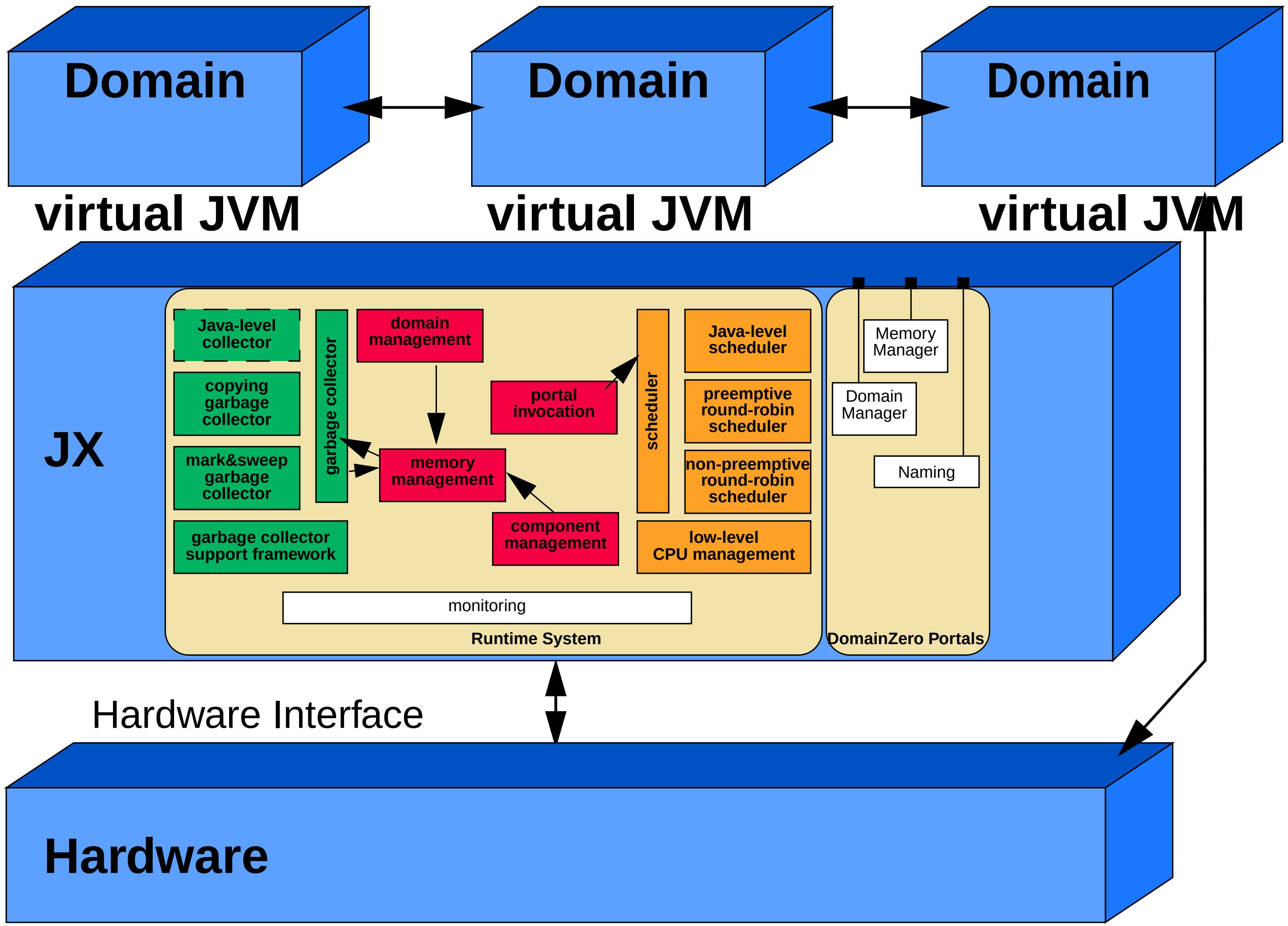
Communication: Portals



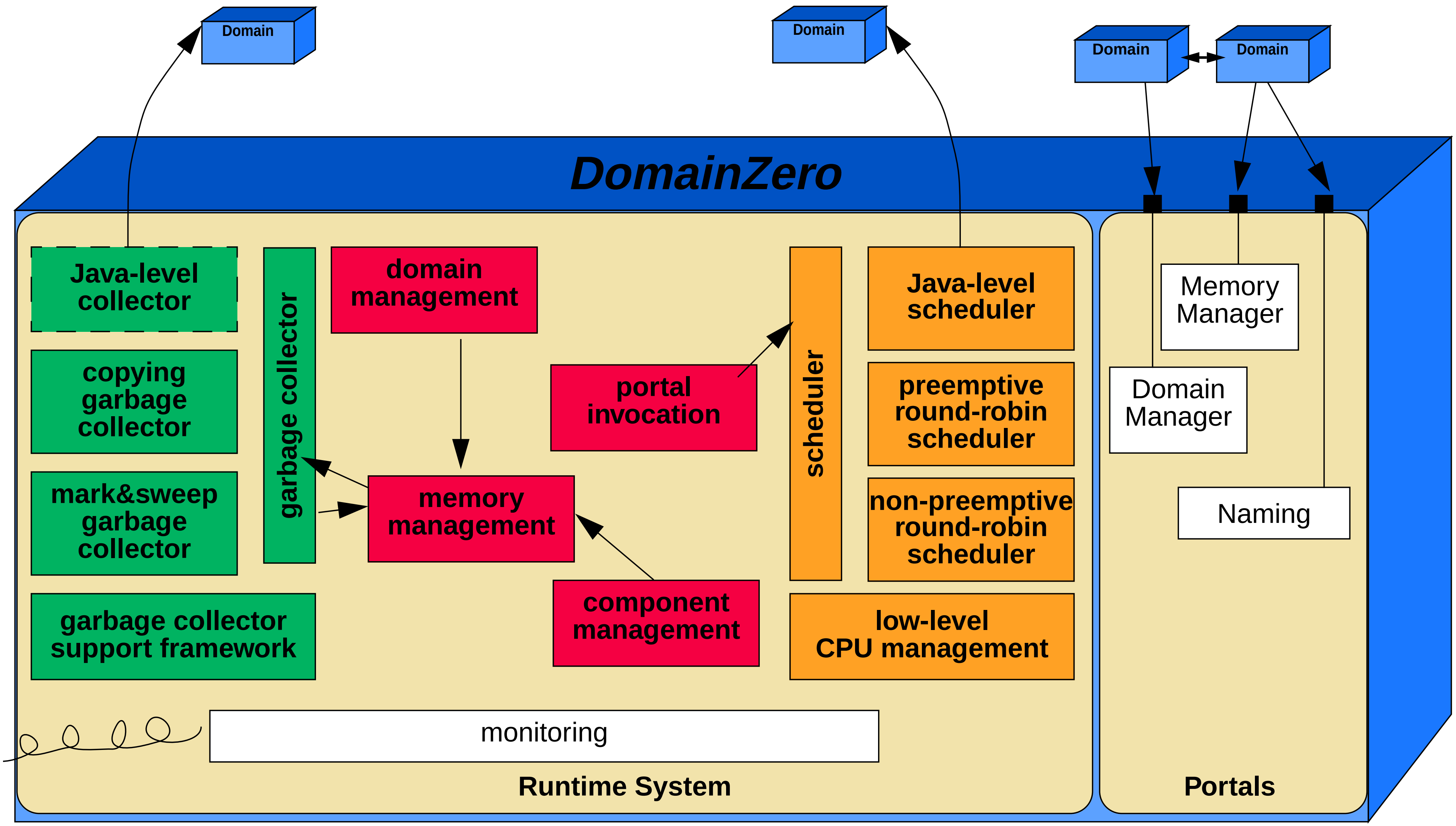
Protection Domains

- Each domain has its own heap
  - no shared objects → no accounting problems (memory, GC time)
  - no GC dependencies → no scalability problems
  - explicit application boundaries → confinement
- Each domain has its own threads
  - no migrating threads → no domain termination problems
- Each domain has its own code
  - no trusted code → improved security

The Microkernel



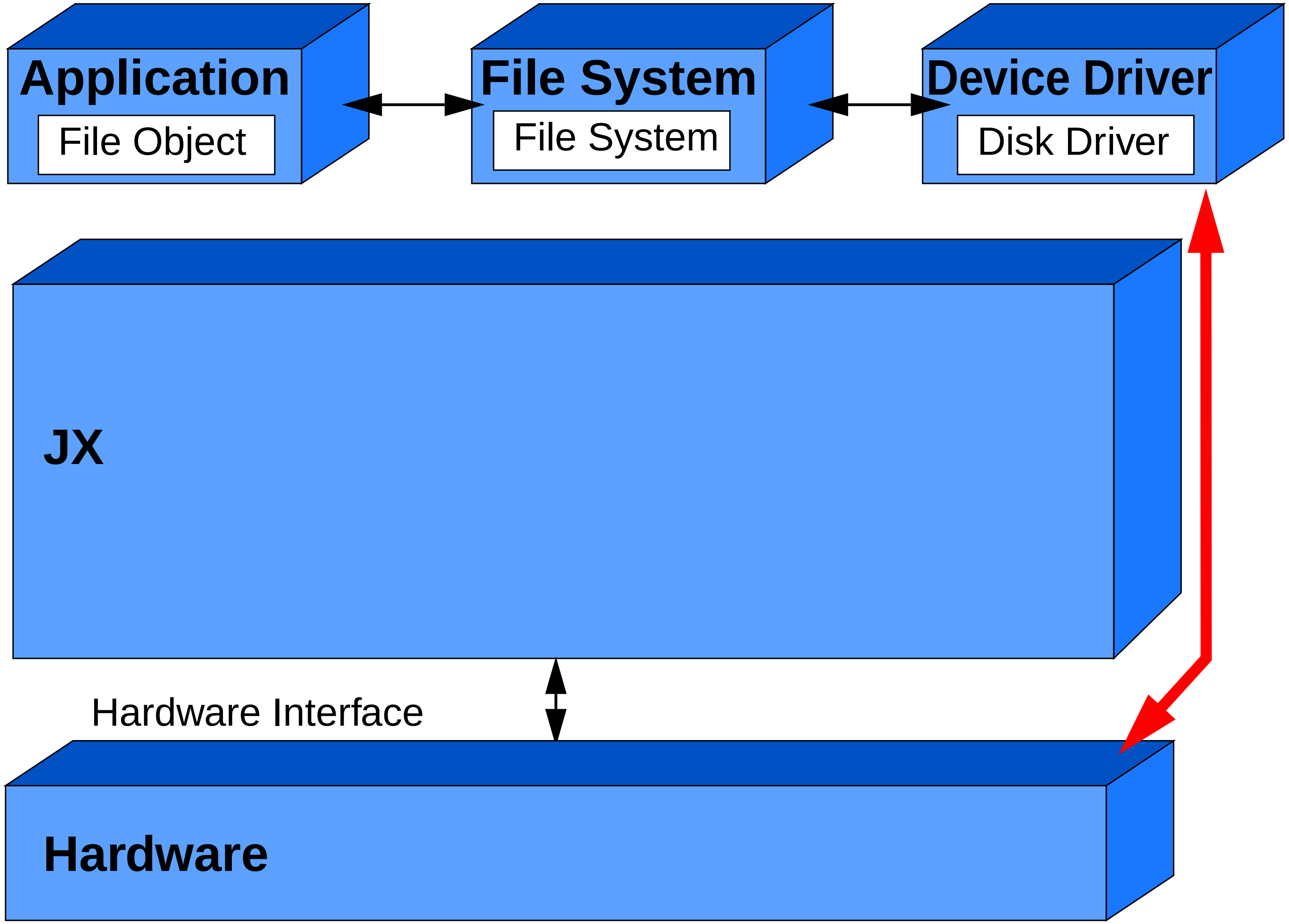
The Microkernel



Outline

- Motivation
- JX Architecture
  - Protection domains
  - Communication mechanism
  - The Microkernel
- **System-level programming in Java**
- Performance

System-level Programming in Java



System-level Programming in Java

- Problems
  - Management of large memory, registers, and on-device memory
  - Interrupt handlers

System-level Programming in Java

- Problems
  - Management of large memory, registers, and on-device memory
  - Interrupt handlers
- Requirement
  - No changes to the language or bytecode instruction set

Memory

- Manage large memory areas
  - disk blocks, network buffers, ...
- Access special memory areas
  - video memory, device registers, ...
- Disadvantages of arrays:
  - no explicit interface
  - no specializations possible
  - located on heap, can be moved (DMA!)
  - can only be passed by copying

Memory Portals

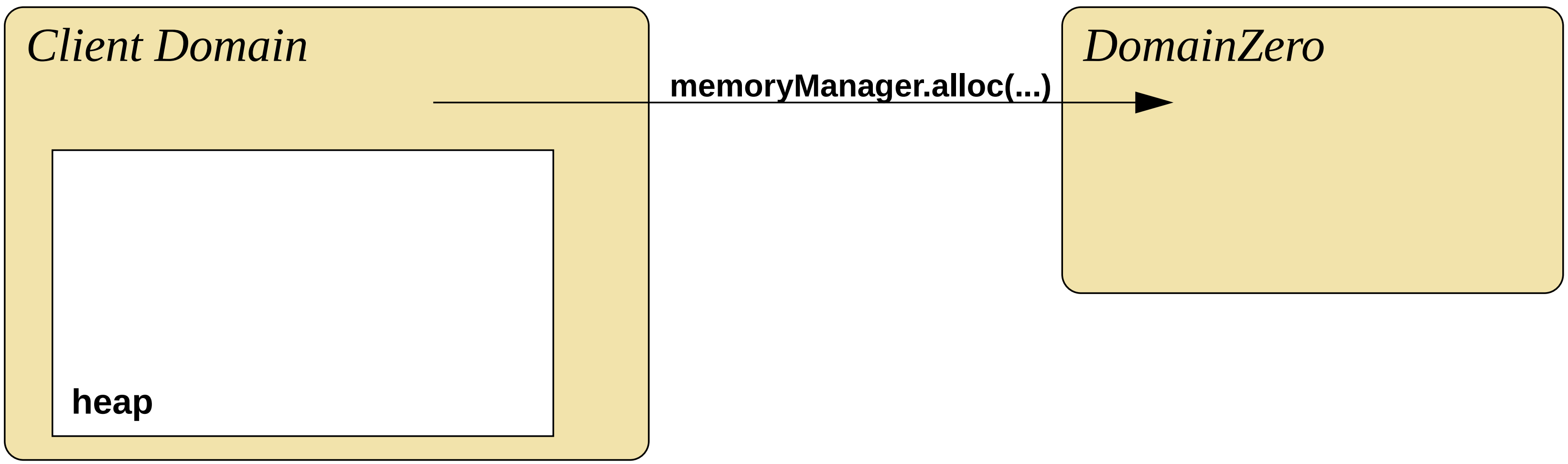
- Explicit interface
- Can be subtyped
- Can be used to access arbitrary memory areas
- Can be treated specially by the translator
- Share data between domains
- Pass subranges to other domains
- Revoke access

Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level

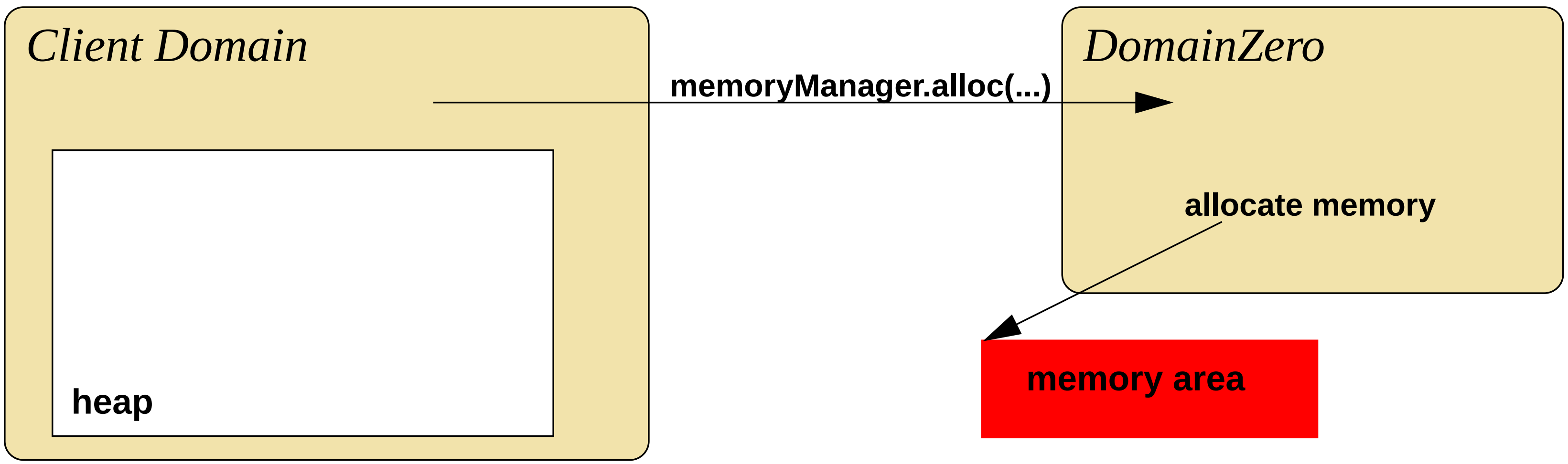
Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
- Example: Memory portal



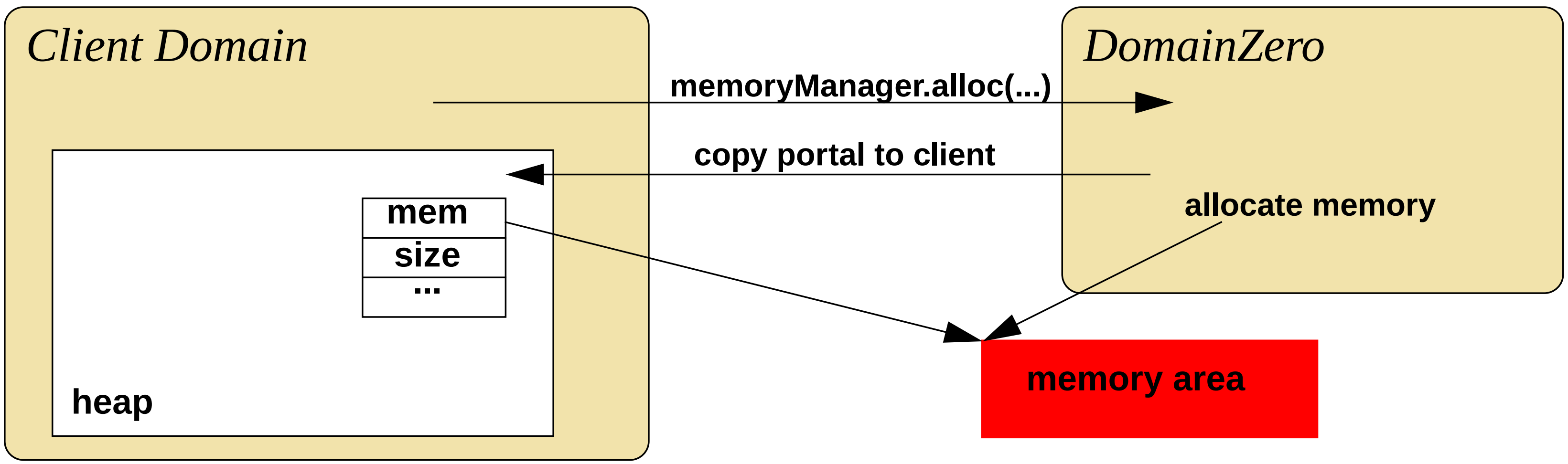
Fast Portals

- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
- Example: Memory portal



Fast Portals

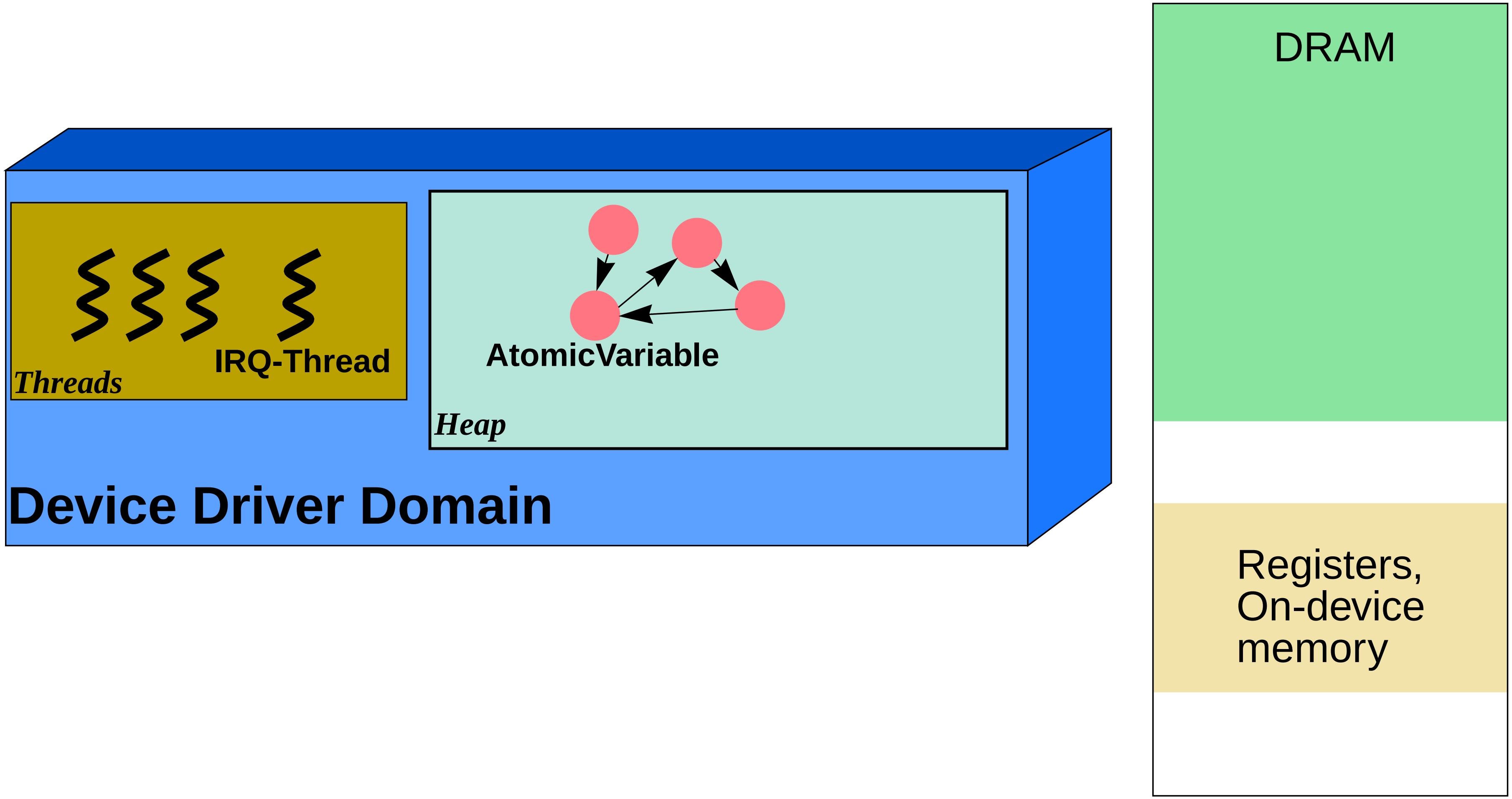
- Execute in caller context (caller thread/domain)
- Can only be created by DomainZero
- Portal object contains special data not directly accessible from the Java level
- Example: Memory portal



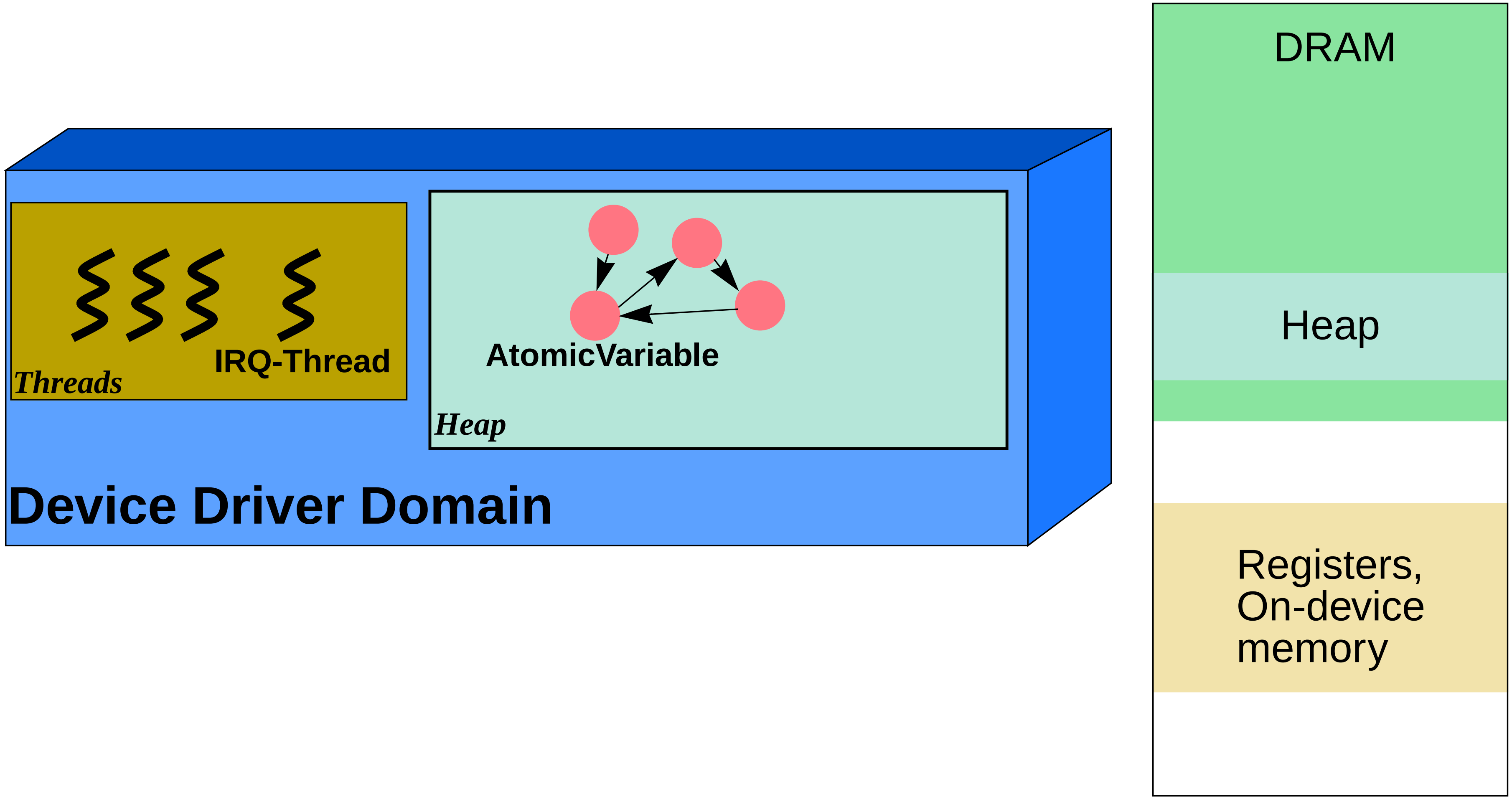
Interrupt Handler

- Interrupts handled in own thread with interrupts disabled
  - must not run in endless loop → verifier
  - must not invoke portals except fast portals
  - must synchronize with other threads without blocking
    - AtomicVariable
      - **`setValue(...)`**, **`unblock(...)`** used in interrupt thread
      - **`blockIfEqual(...)`** used outside interrupt thread
    - → Single Producer / Single Consumer Queue

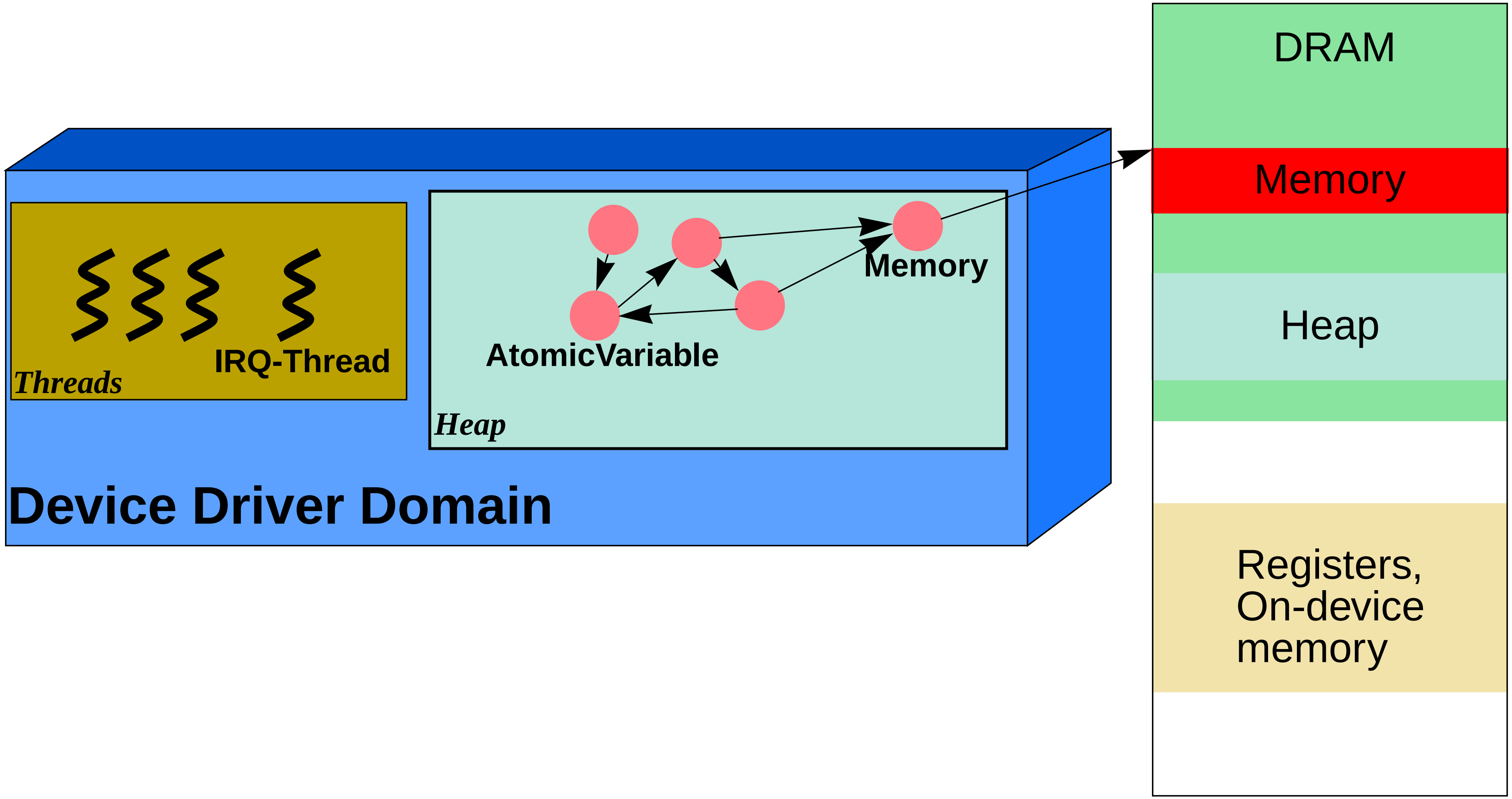
System-level Programming in Java



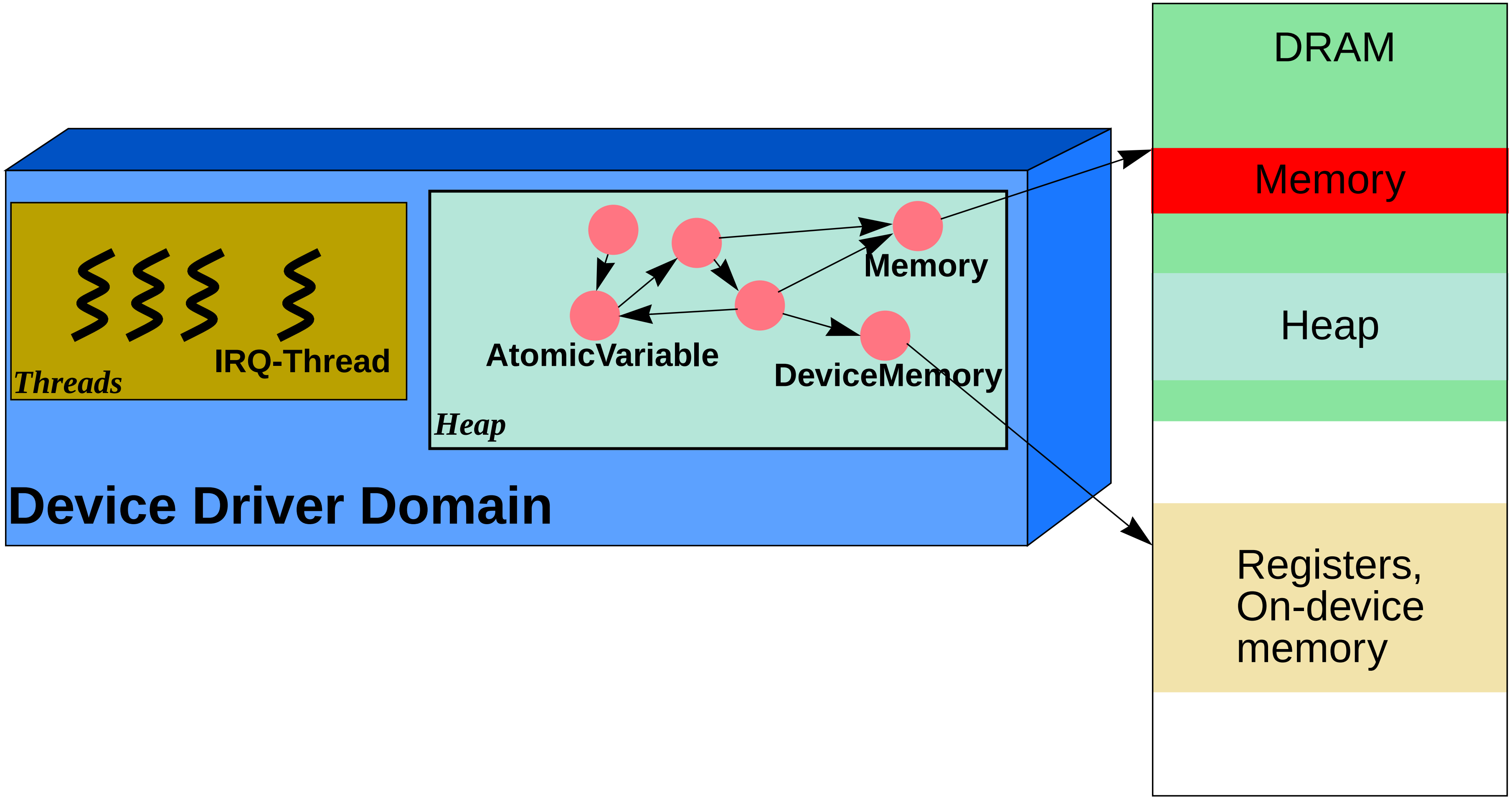
System-level Programming in Java



System-level Programming in Java



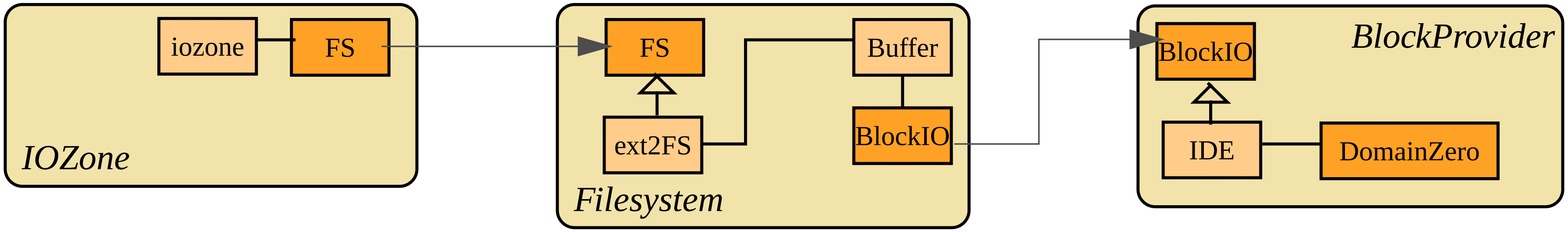
System-level Programming in Java

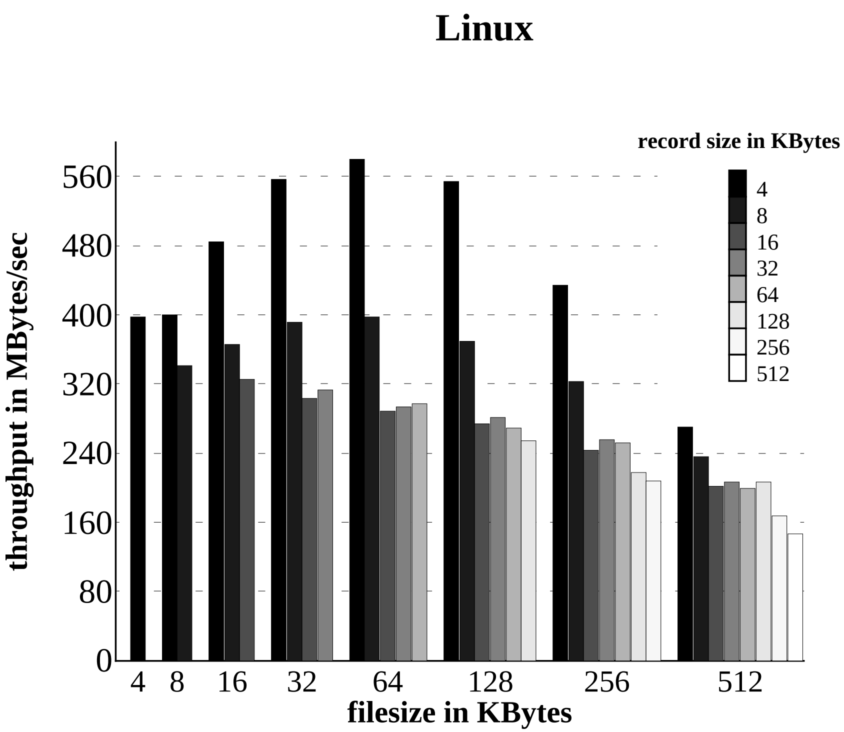


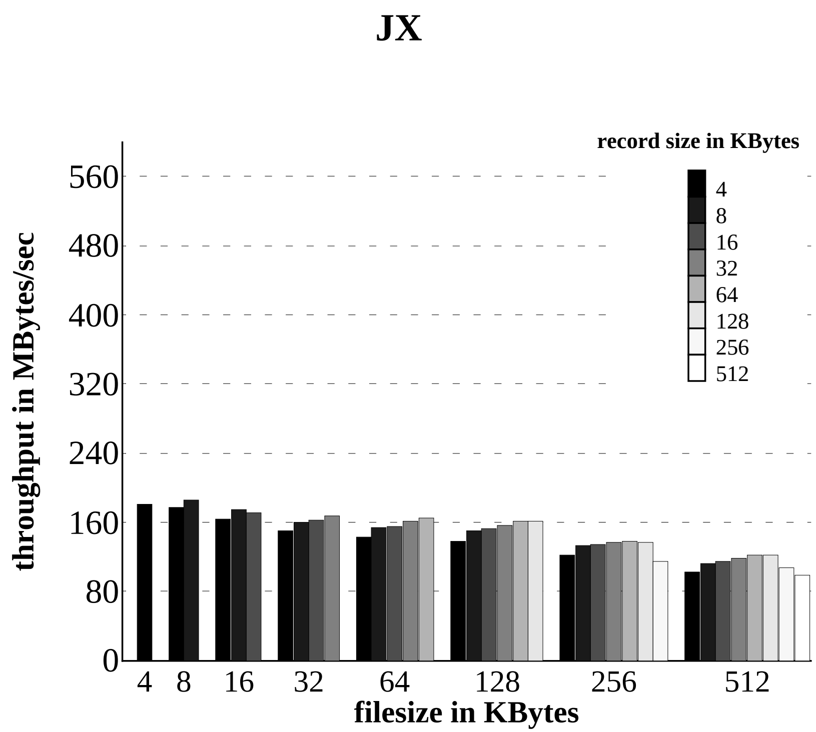
Outline

- Motivation
- JX Architecture
  - Protection domains
  - Communication mechanism
  - The Microkernel
- System-level programming in Java
- **Performance**
  - **IOZONE**
  - **NFS**

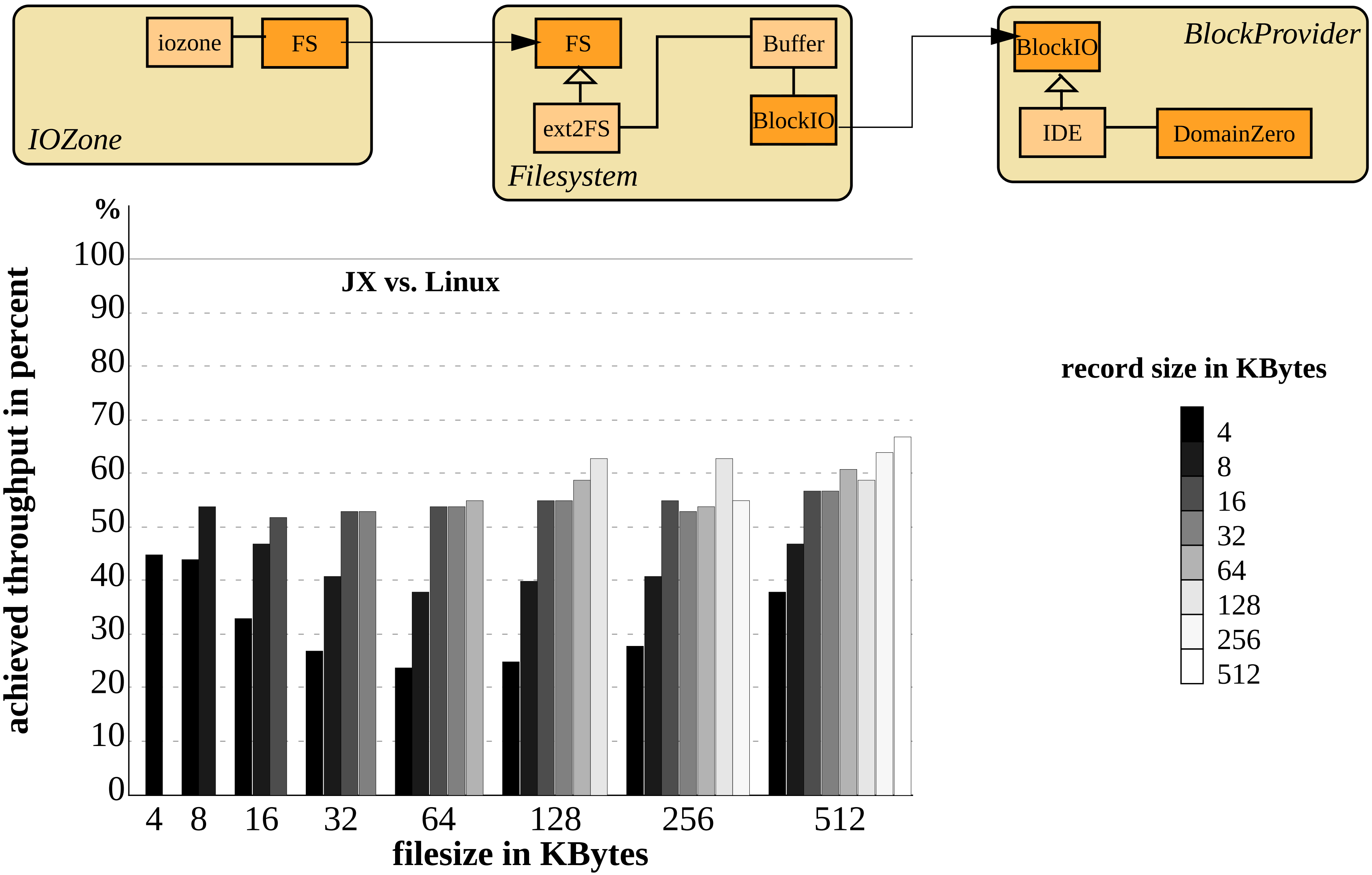
IOZone: Multi-domain Config



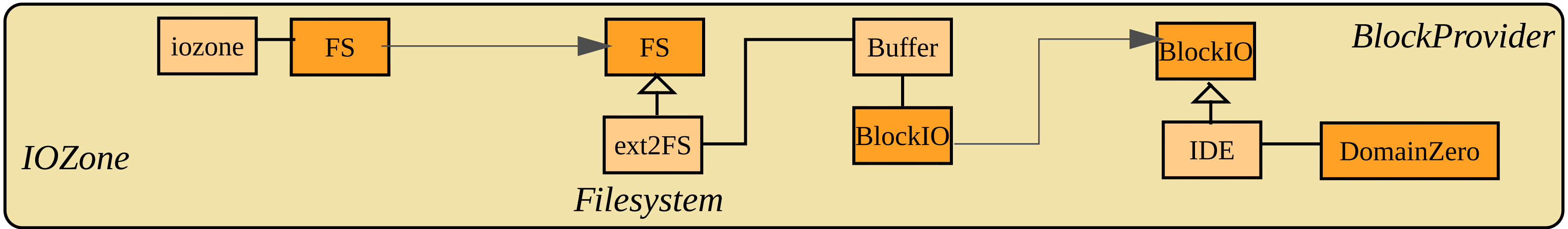


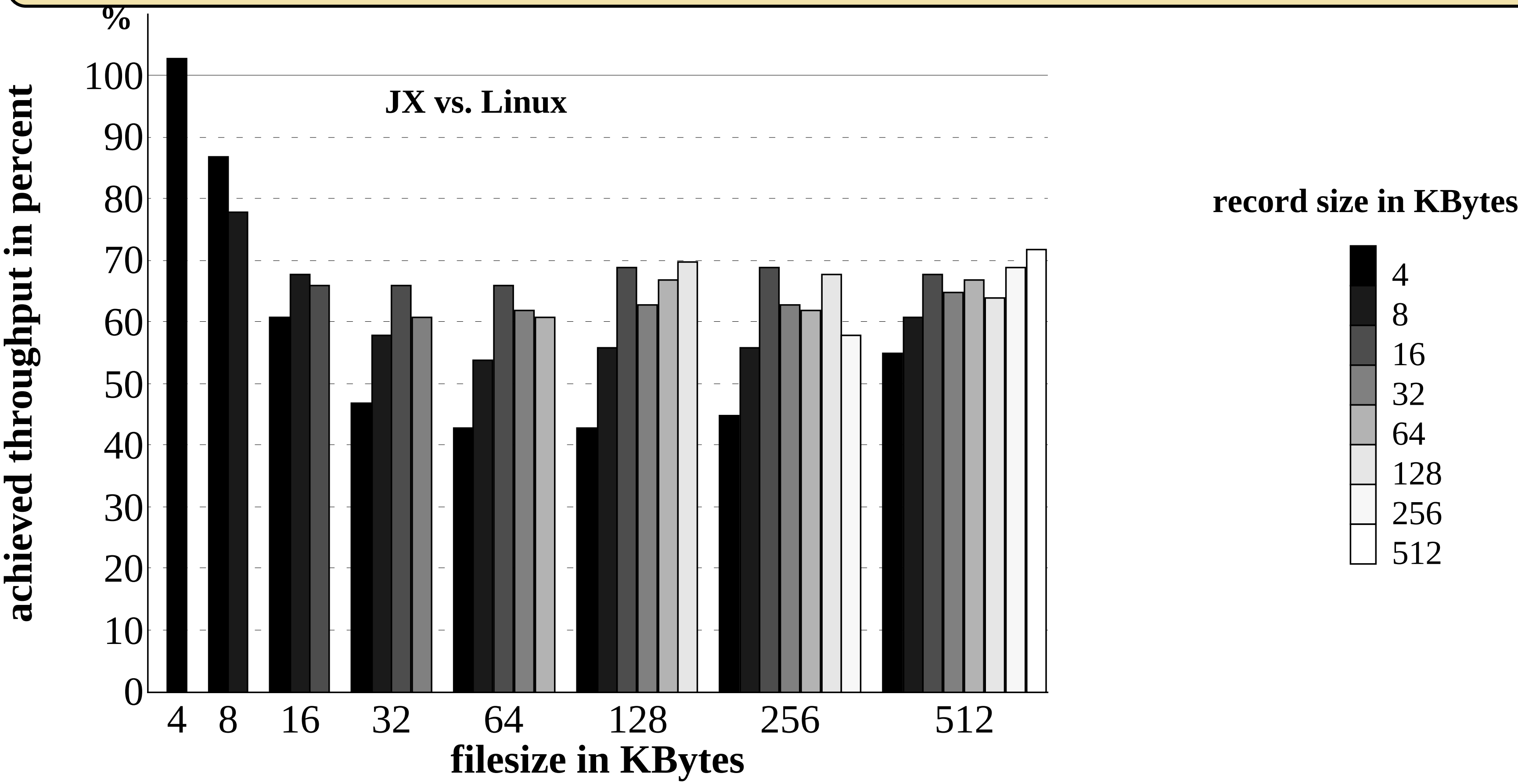


IOZone: Multi-domain Config

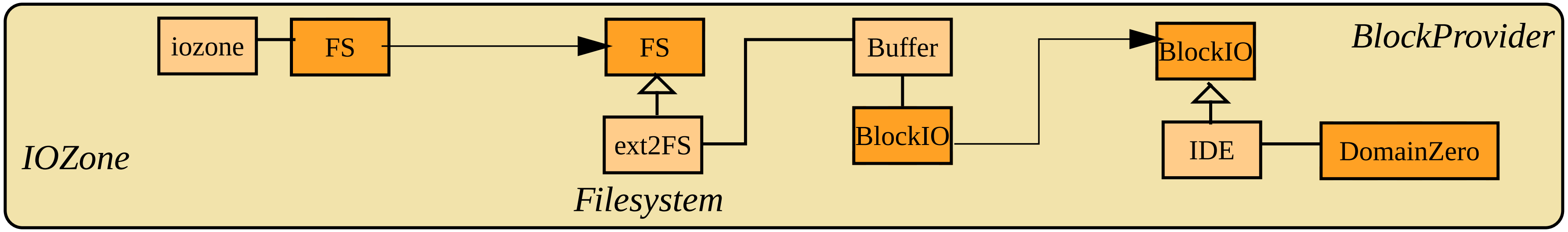


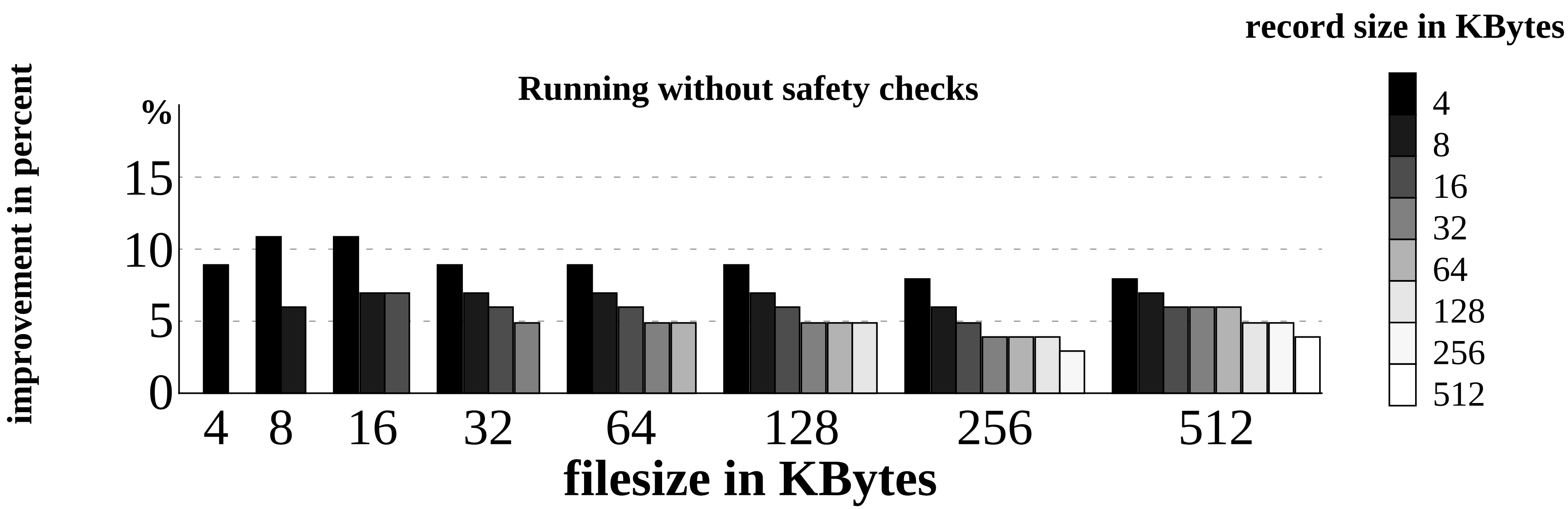
IOZone: Single-domain Config



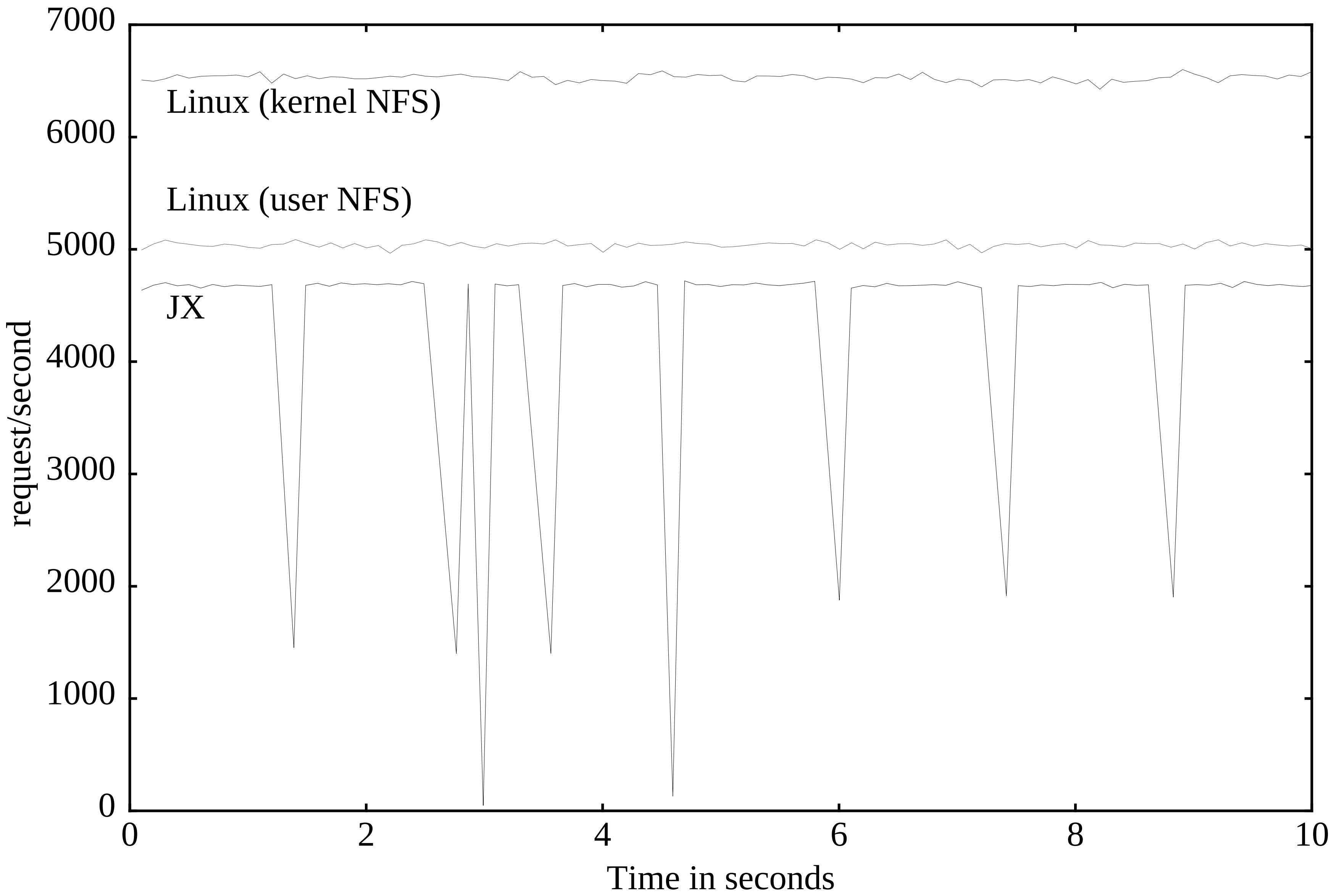


IOZone: Single-domain Config

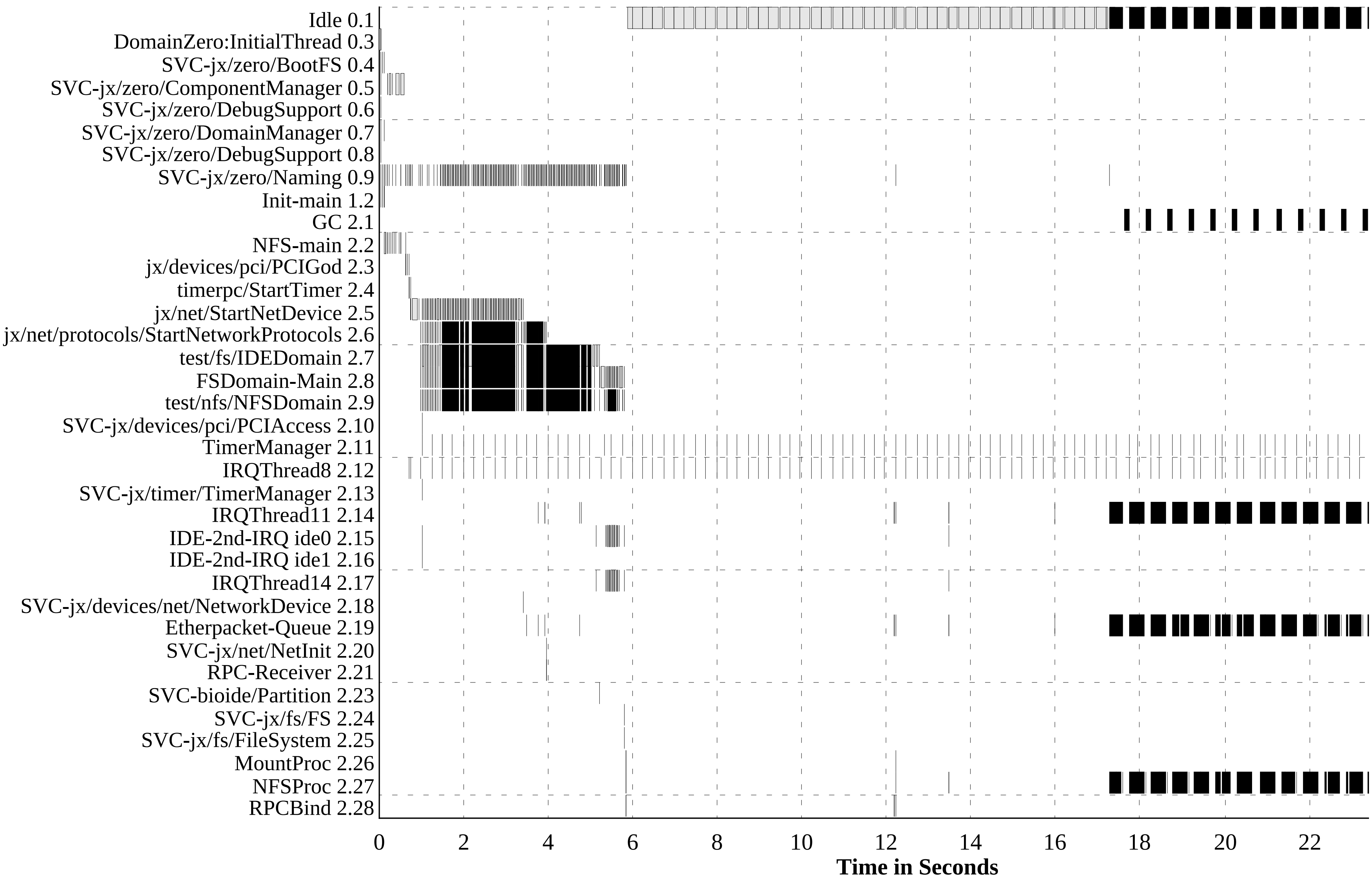




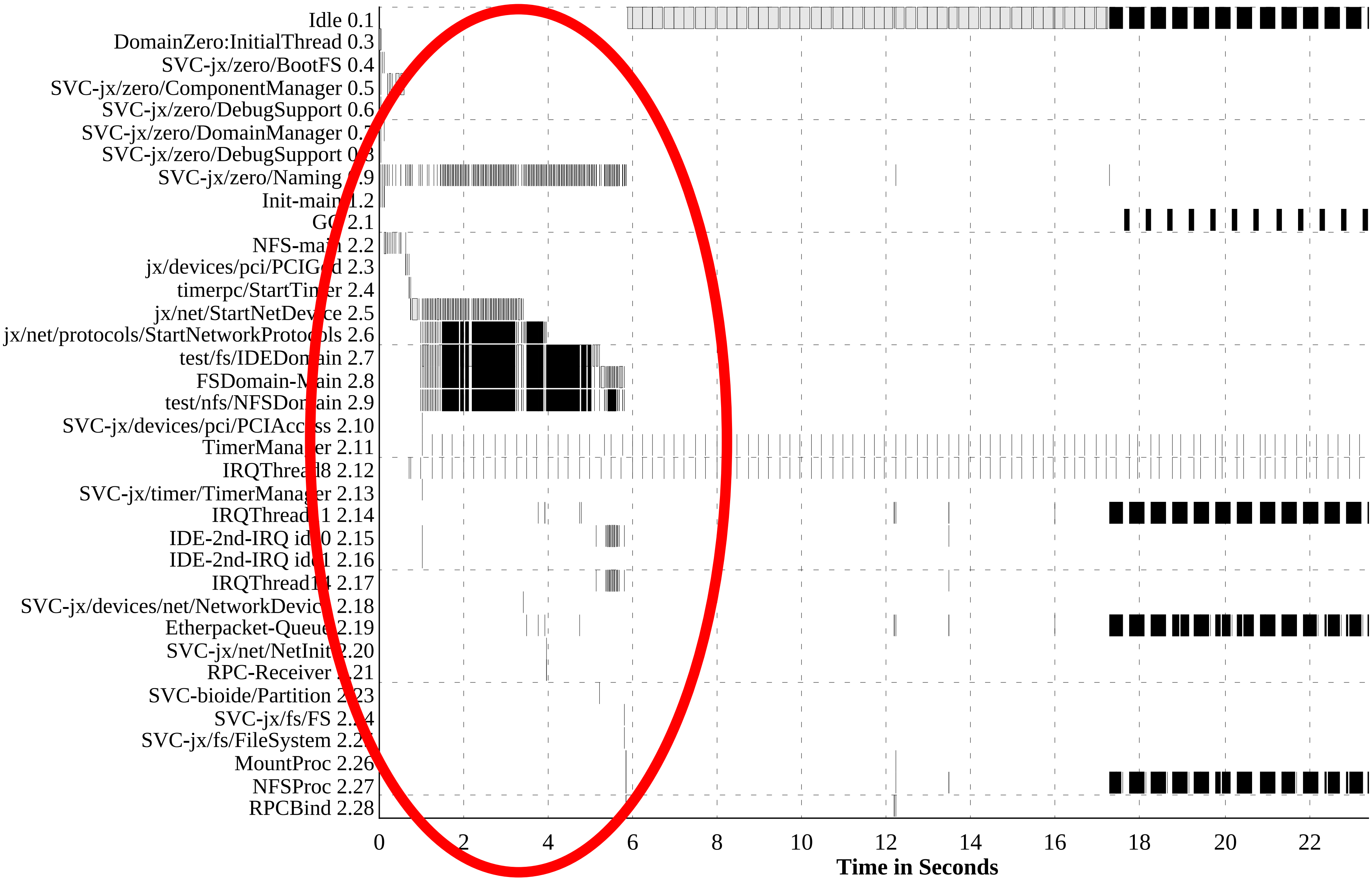
NFS: *getattr* request rate



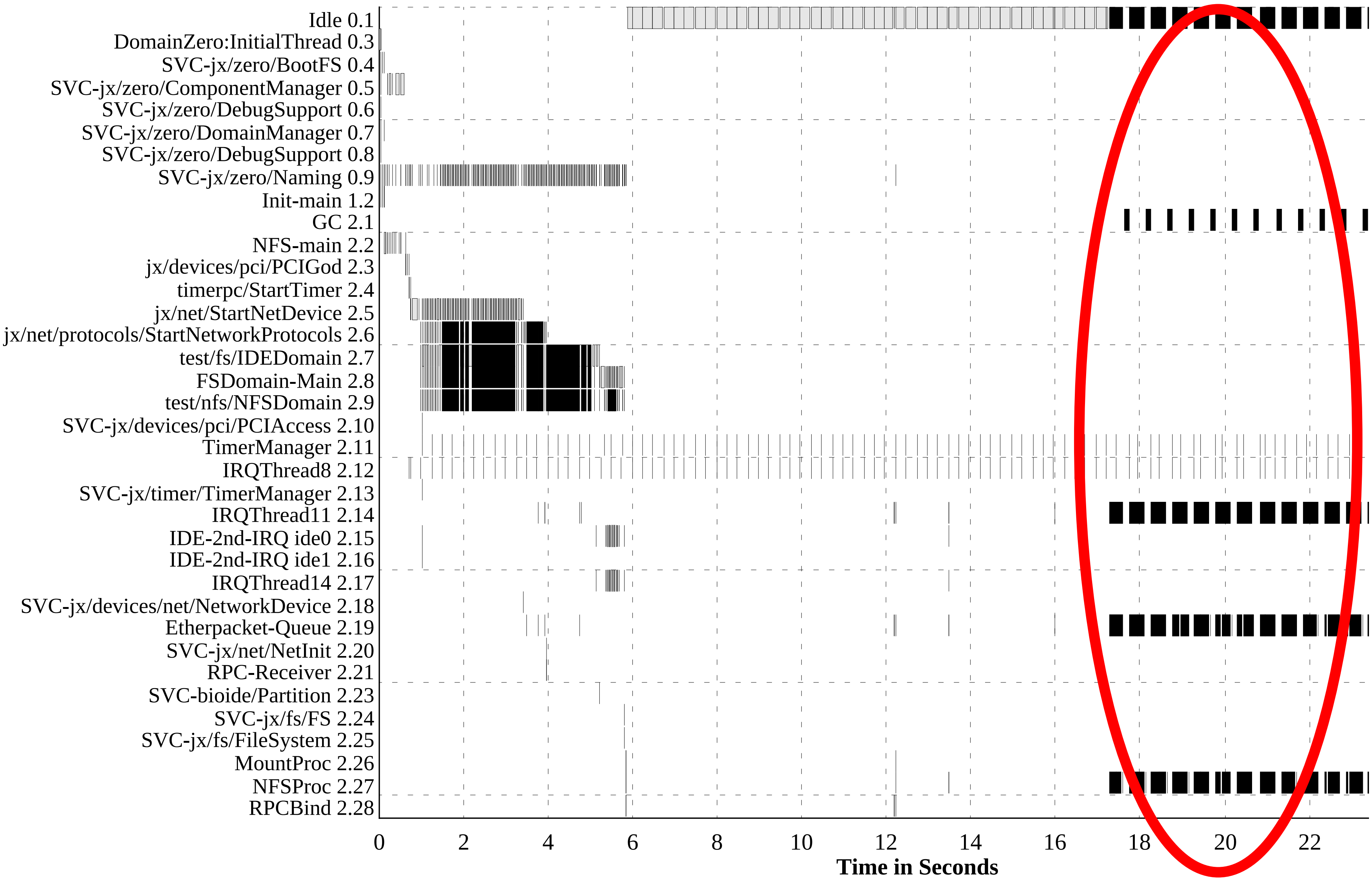
NFS: *getattr* request rate



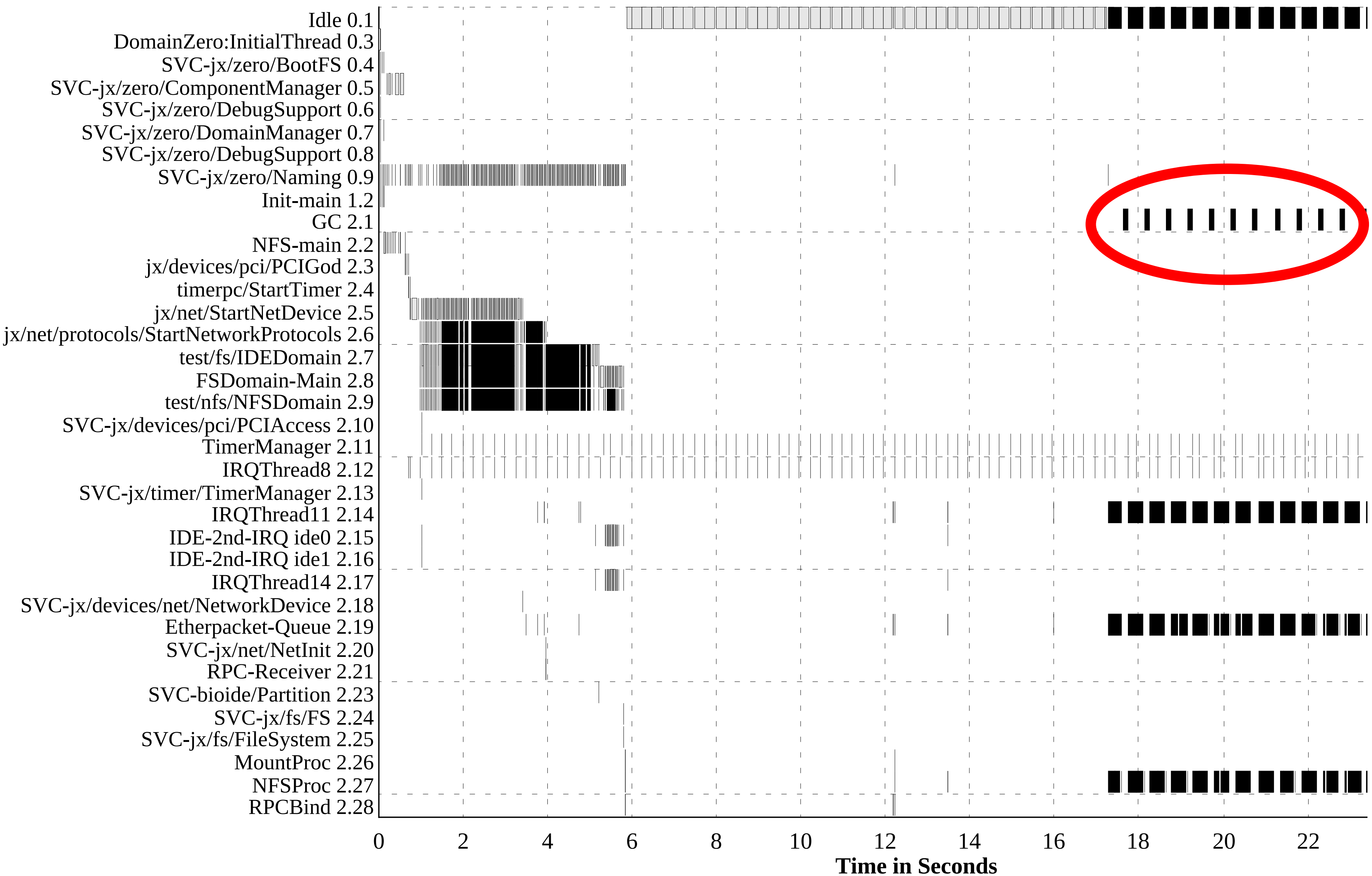
NFS: *getattr* request rate



NFS: *getattr* request rate



NFS: *getattr* request rate



Conclusion

- It is possible to build an OS based on type-safe protection
  - Isolation of protection domains
  - System-level programming without language extensions
- OS takes advantage of advanced language technology
  - Increased robustness of system components
  - Modular system
  - Flexibility in system configuration
  - Extensibility using domains
- Performance is sufficient for most applications
  - → http://www4.cs.fau.de/Projects/JX