Operating System Kernels

Presenter: Saikat Guha

Cornell University

CS 614, Fall 2005

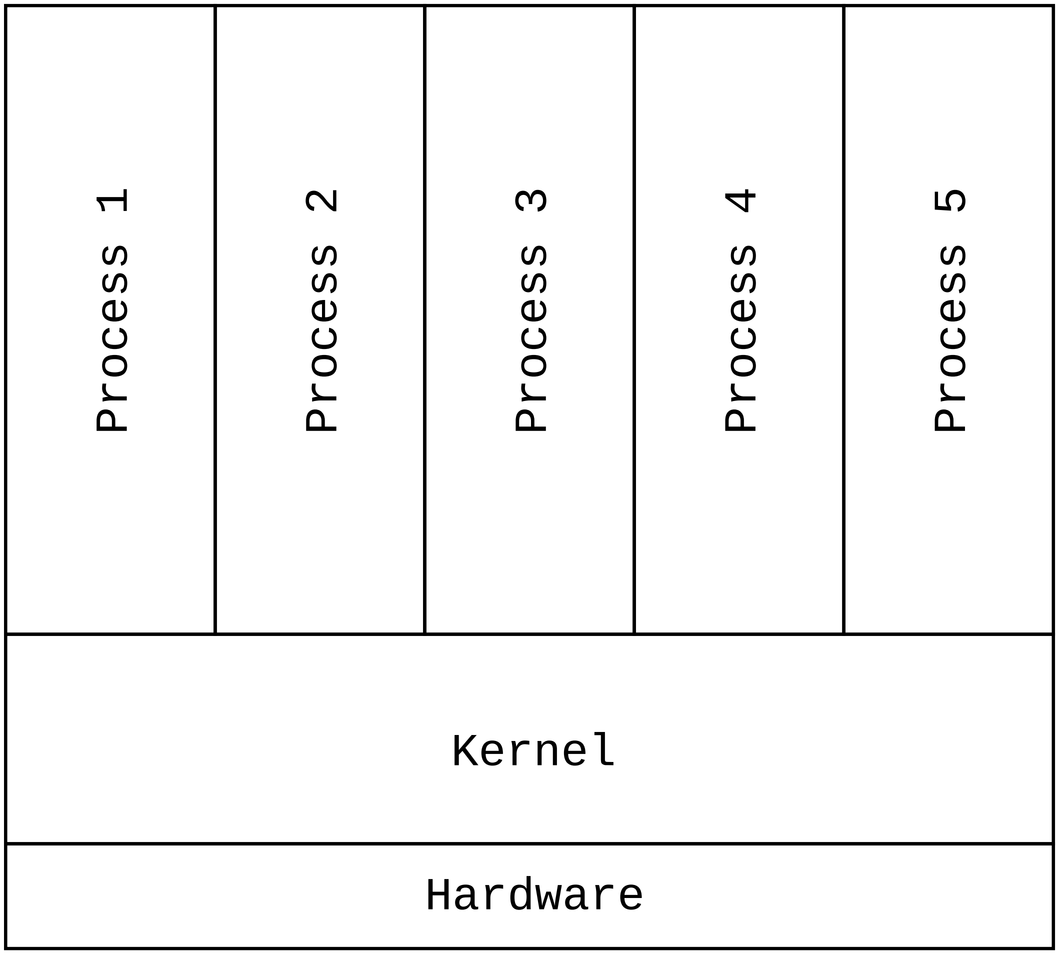
Operating Systems

- Initially, the OS was a run-time library
- Batch ('55–'65): Resident, spooled jobs
- Multiprogrammed (late '60): Multiple jobs
- Time-sharing ('70s): Interactive jobs
  - Multics, UNIX
- Networked OS, Distributed OS, Parallel OS, Real-time OS

UNIX

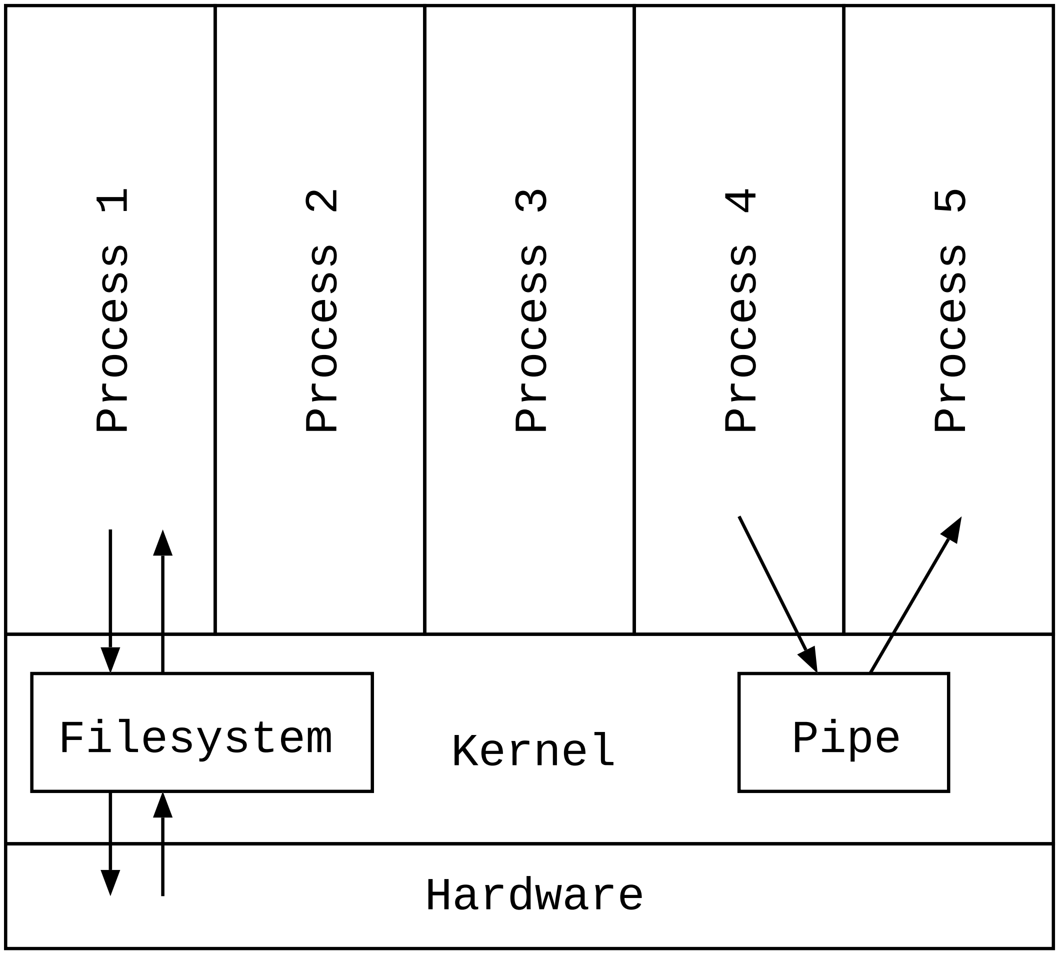
- **THE** operating system
  - Dennis Ritchie, Ken Thompson at AT&T
- "File" Abstraction
- Kernel
  - Processes, IPC
  - Filesystem
  - Networking (eventually)
  - Graphics (Windows)
- Userspace
  - Shell
  - Commands

Operating Systems



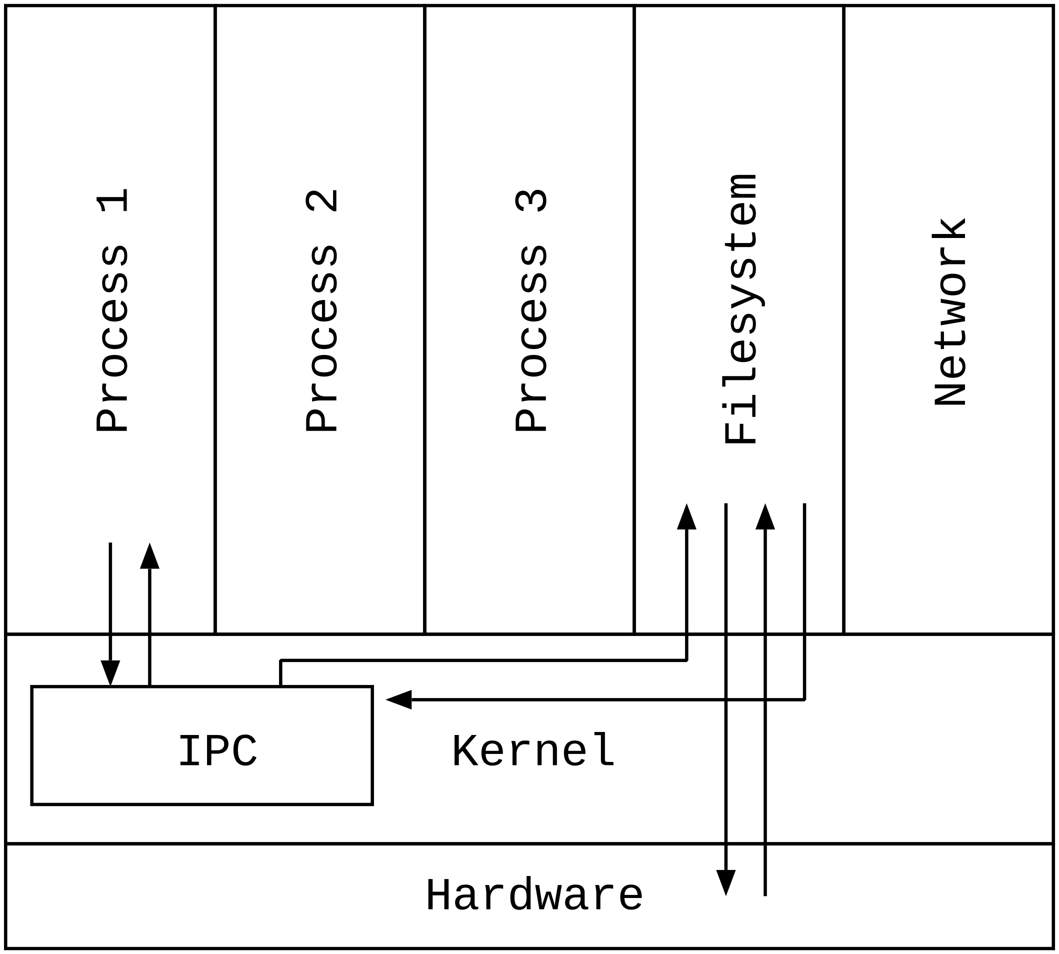
User-Kernel Split

Operating Systems



Monolothic Kernel

Operating Systems



Microkernel

μ-Kernels

- Minimal services
- Usually threads or processes, address space and inter-process communication (IPC)
- User-space Filesystem, Network, Graphics, even device drivers sometimes.

Monolithic Kernels: Advantages

- Kernel has access to everything
  - All optimizations possible
  - All techniques/mechanisms/concepts can be implemented
- Extended by simply adding more code
  - Linux at 3.3M lines of code
- Tackle complexity
  - Layered kernels
  - Modular kernels
  - Object oriented kernels. Do C++, Java, C# help?

μ-Kernels: Advantages

- Minimal
  - Smaller trusted base
  - Less error prone
  - Server malfunction easily isolated
- Elegant
  - Enforces modularity
  - Restartable user-level services
- Extensible
  - Different servers/API can coexist

μ-Kernels

- 1st generation μ-kernels
  - Mach (CMU)[1]
  - Chorus (Inria, Chorus systems)
  - Amoeba (Vrije University)
  - L3 (GMD)[2]

[1] External pager

[2] User-Level Driver

μ-Kernels: Problems

- Overheads
  - Chen and Bershad, '93
  - Impact of caches, locality, TLB collisions
  - Up 66% degradation in Mach
- Co-located servers for performance
- Can be optimized to be fast on an architecture
  - But, performance not preserved on other architectures

μ-Kernels

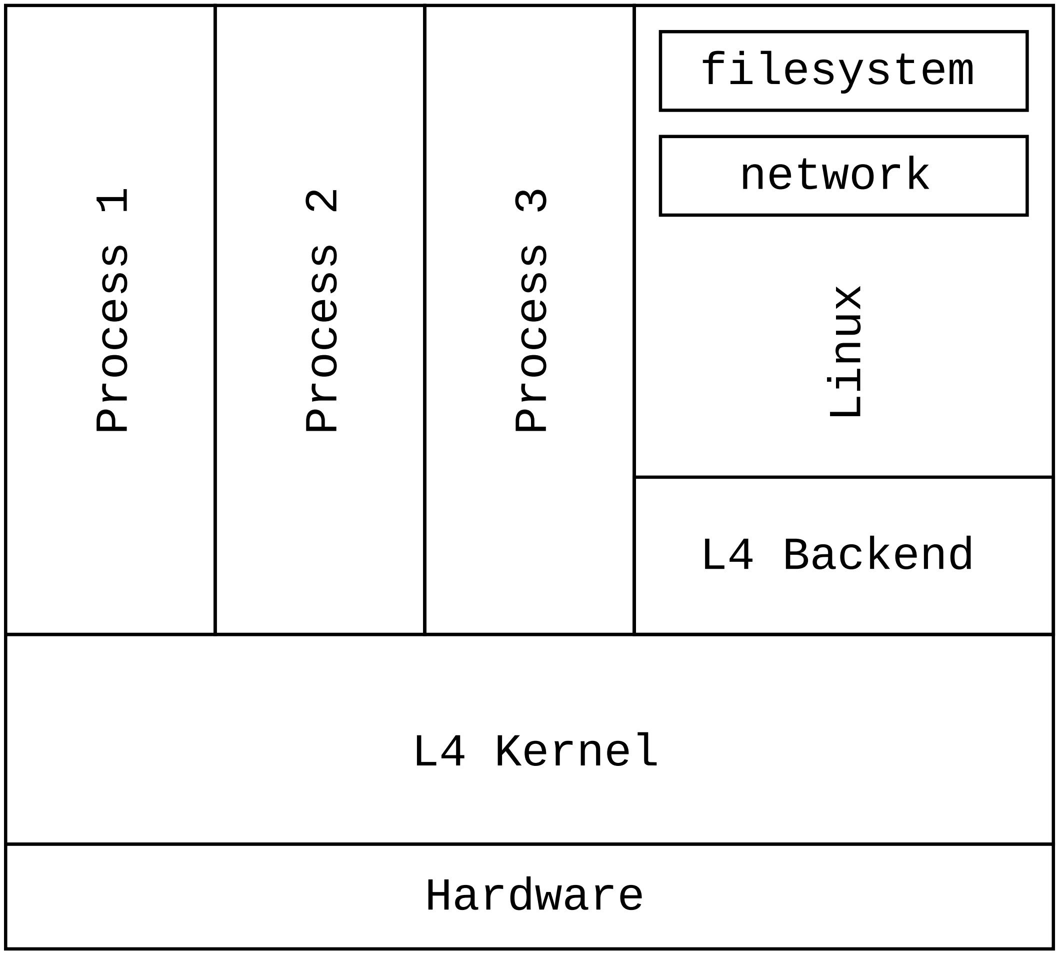
- 2nd generation μ-kernels
  - Spin (UWash)
  - Exokernel (MIT)
  - L4 (GMD/IBM/UKa)[3]

[3] User-Level Address Space

Summary of First Paper

- The Performance of μ-Kernel-Based Systems (Härtig et al., SOSP '97)
- Evaluates a L^4 μ-kernel based system
- Ports Linux to run on top of L^4
- Suggests improvements

L^4-Linux



L^4-Linux

L^4-Linux

- 2 basic concepts
  - Threads
  - Address Spaces (AS)
- Recursive construction of AS
  - Grant - Give a page to another AS
  - Map - Share a page with another AS
  - Demap - Revoke a mapped or granted page
- I/O ports treated as AS
- Hardware interrupts treated as IPC

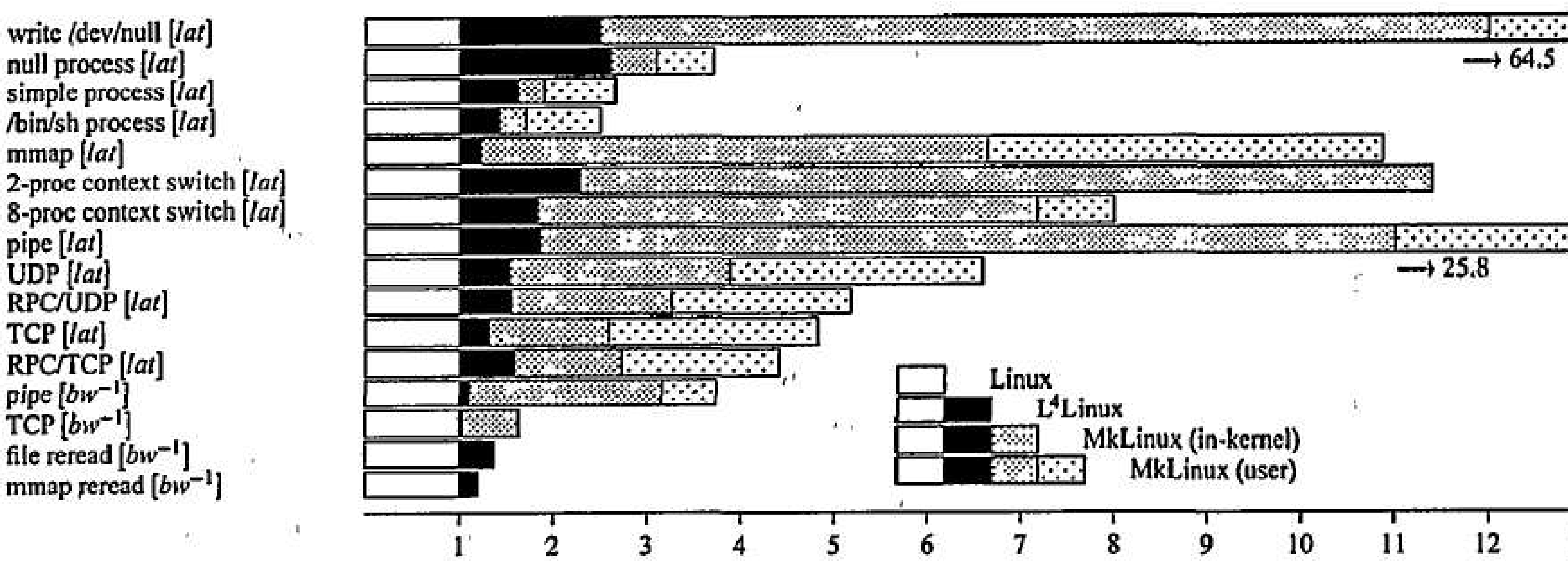
L^4-Linux

- TLB caches page-table lookups
  - Flused during context switch
  - Flushing not necessary for tagged TLBs
- L^4-Linux avoids frequent flushes
  - Pentium CPU's emulate tagged TLBs for small address spaces
- syscall time
  - Unix – 20μs
  - Mach – 114μs
  - L^4 – 5μs

Performance

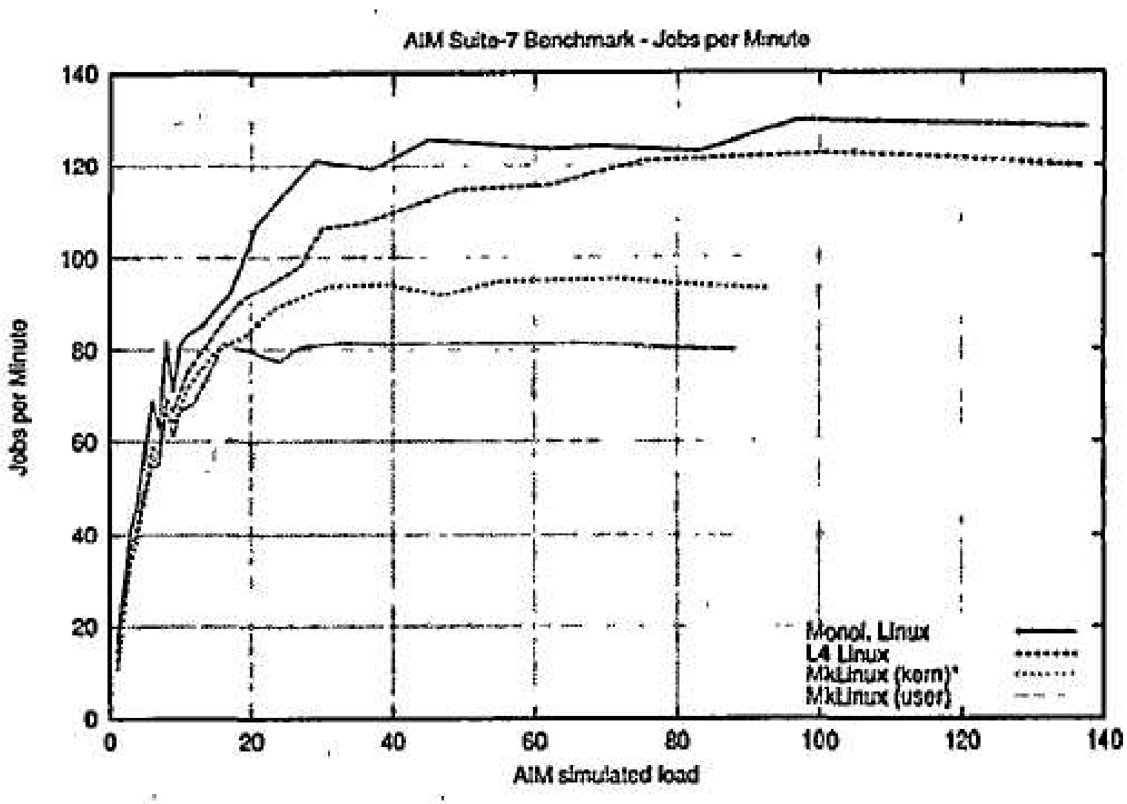
Is L^4-Linux a practical system?

Performance



Is L^4-Linux a practical system? **Yes**

Performance



Is L^4-Linux a practical system? **Yes**

Performance

- L^4 incurs 5%–10% overhead
- Collocation alone does not solve performance problems
  - What about L^4 without collocation?
- L^4-Linux is proof-of-concept
  - Pipes can be made faster
  - Better VM in non-legacy mode
  - Can benefit from cache partitioning

Comparison

L^4-Linux

- Highly optimized (for x86)
- Functionality limited by Linux
- Untrusted components isolated

Flux OSKit

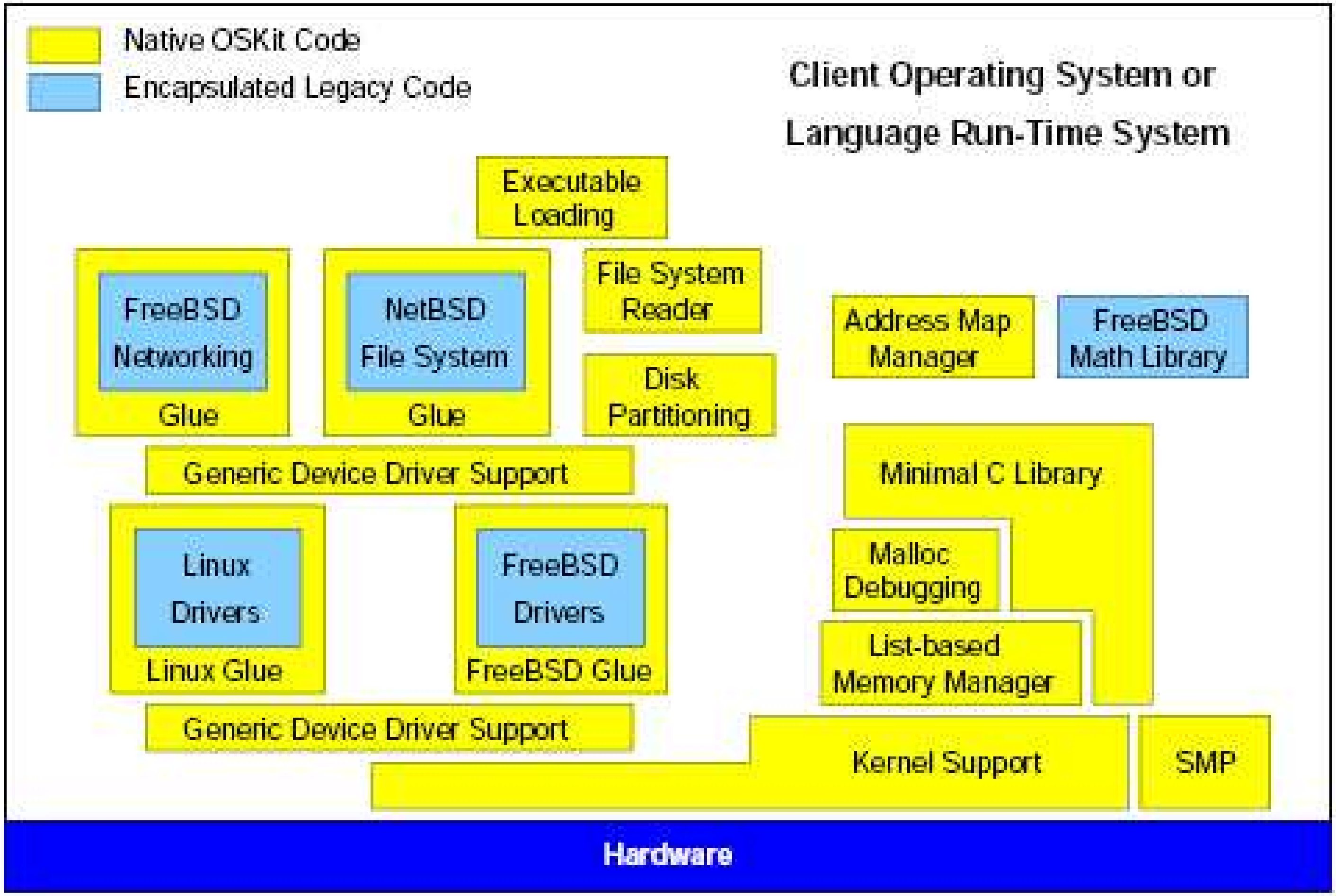
- Tons of functionality

  (Linux, BSD, Java, SML, ...)
- Not tuned for high performance
- Implementation details exposed

Flux OSKit

- Framework and reusable OS components
- Focus on component of research-interest
- Reuse other existing components for functionality

Flux OSKit



Flux OSKit Components

Flux OSKit

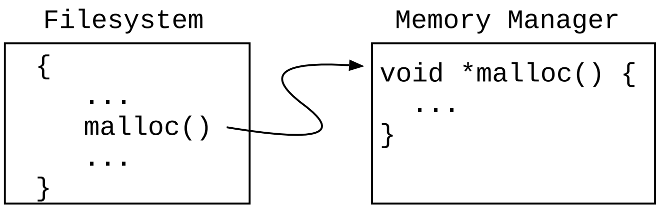
- Bootloader
  - Multiboot compliant
- Kernel Support Library
  - Architecture specific
- Memory Management Library
  - kmalloc(), alignment etc.
- Minimal libc
  - non-buffered read(), write() etc.
  - minimizes dependencies

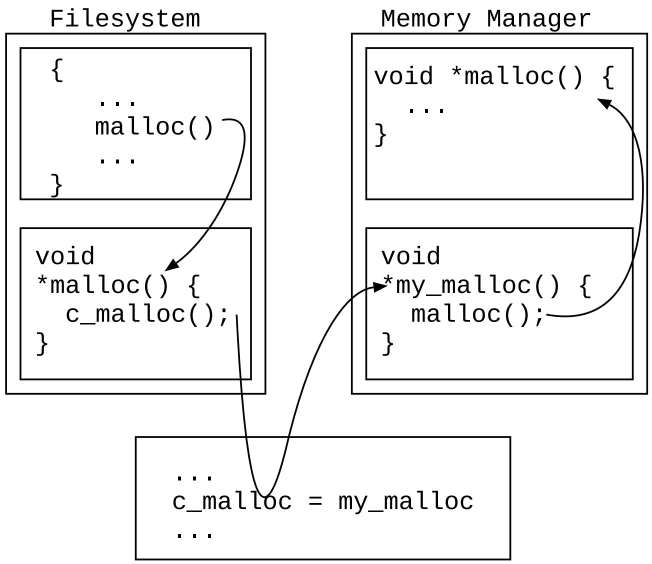
Flux OSKit

- Debugging Support
  - GDB over serial line
- Device Driver Support
  - Drivers from Linux, FreeBSD inside wrappers
- Protocol Stacks
  - "Wrapped" FreeBSD network stack
- File System
  - "Wrapped" NetBSD code

Flux OSKit

- OSKit components are separable, no dependence
  - Other OS: Modularity does not imply independence





Very little overhead

- Provides abstractions
- Doesn't hide implementation

Case Studies

- ML/OS
  - SML: Static Typing, Concurrency through continuations, No stack, Aggressive heap usage, Interpreted.
  - ML/OS: 2 people, one semester using OSKit
- Java
  - Existing JVM
  - Java/OS: 3 weeks using OSKit
- SR
  - Concurrent programming language

Summary

- L^4-Linux: μ-Kernels can be fast
  - Full system binary-compatible with Linux runs 5%–10% slower.
- FluxOSKit: Kernels from reusable components
  - Write fully-functional research OS in weeks