Fast and Flexible Mach-based Systems

Jay Lepreau
John Carter

Computer Systems Laboratory
Department of Computer Science
University of Utah

http://www.cs.utah.edu/projects/flexmach/

January 19, 1995

Key Ideas

- Lightweight & modular kernel functionality
- Adaptive specialization
- Attributed interfaces: “presentation”
- Integrated module management
- Shared address space(s) ease sharing
- Transparent distribution through efficient distributed shared memory

Impact

- Decomposed yet high performance OS
- Statically and dynamically adaptive OS
- Evolutionary and distributed shared-address-space system
- New generation of Mach microkernel
- Component-based systems?

Mach: an Excellent Research Base

- Powerful, portable, free, expertise available
- And it *needs* research contributions!
- Working in an existing system means:
  - *Less* constraint: working system provides framework for experimentation.
  - Incremental results
  - No startup costs, lower infrastructure costs
  - Validation is easy: lots of existing pgms
  - Commercial users give tech transfer path

Mach 4.0: Goals and Approach

- Rethink “fundamental” Mach abstractions
- Yet retain, via *evolutionary* approach:
  - Backwards compatibility (mostly)
  - Code base
  - Community expertise
  - Relevance to commercial efforts
  - Tech transfer path
- Achieve:
  - Higher performance
  - More flexible & powerful base for R&D

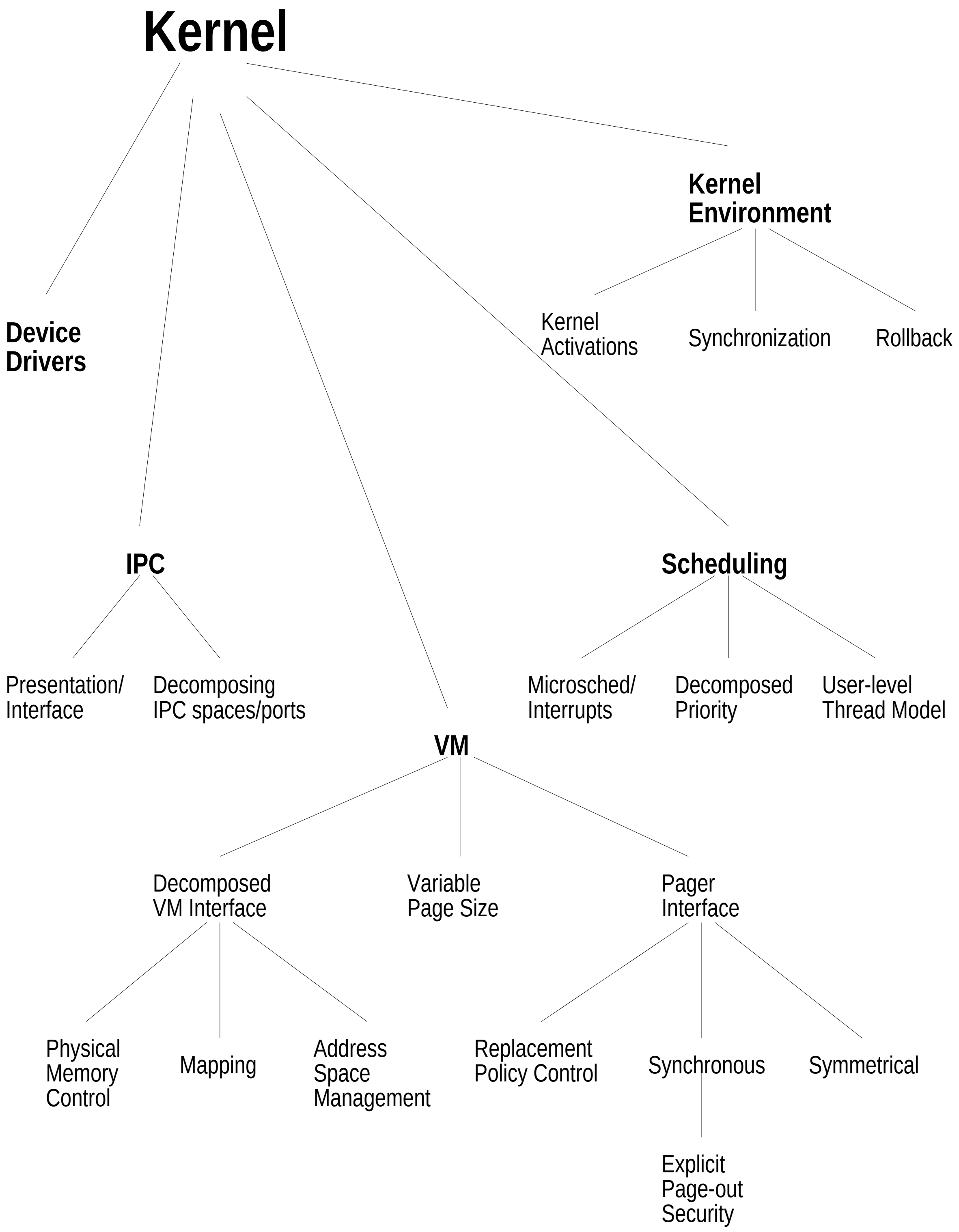
Mach 4.0: Design Themes

- Simple & lightweight kernel mechanisms
- Export lower level (modular) abstractions
- Compose for convenience & compatibility

Modularity — Kernel

- Modular *Implementation*
  - "Kernel activations"
  - Transition paths
- Modular *Functionality*
  - Thread decomposition
  - Task decomposition
  - VM decomposition
  - Port decomposition
  - ...
- Modular *Interfaces*
  - Well defined
  - Annotated
- Modular *Source Organization*
  - Oriented to special configurations
  - Currently shared by DOS extender

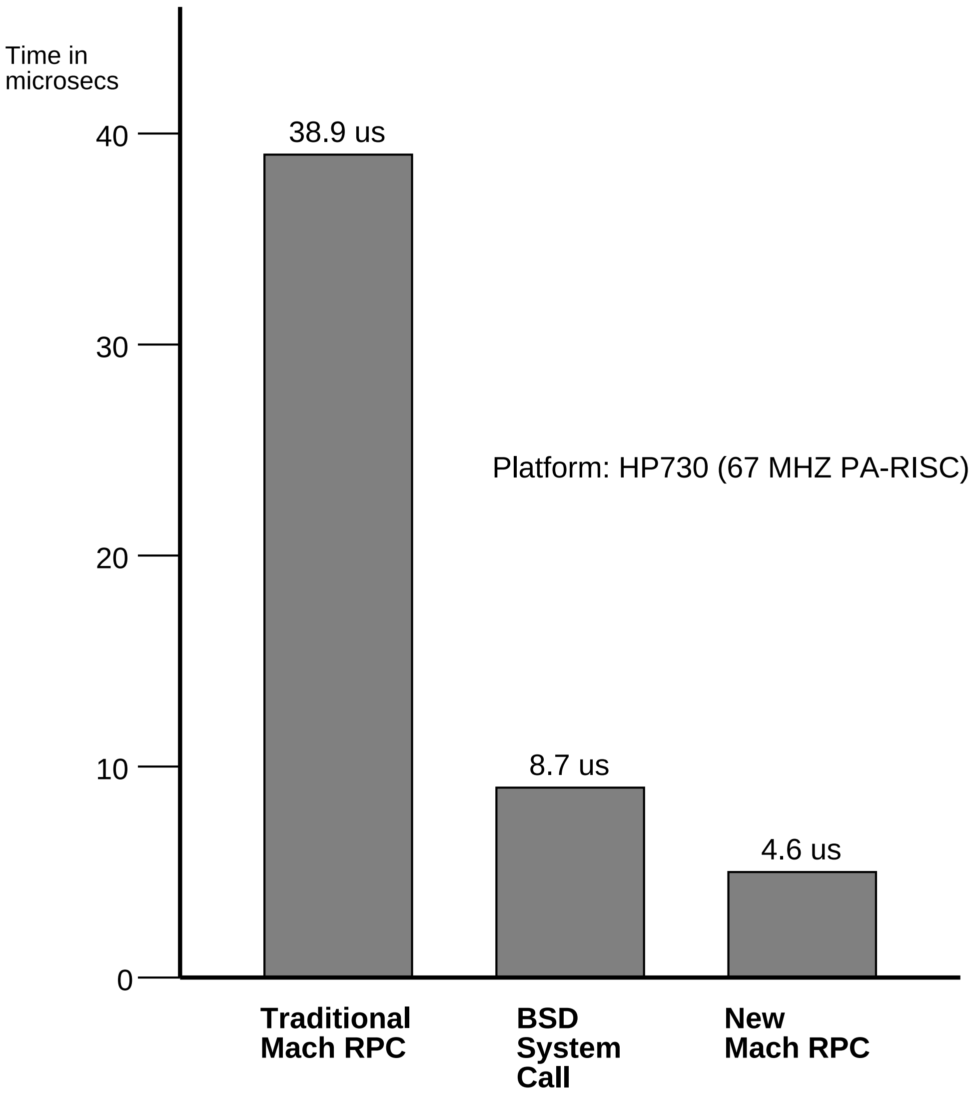
MACH 4 KERNEL PLANS



Lightweight Kernel Functionality

- IPC
  - Simple control transfer (migrating threads)
  - Simple kernel functionality
    - Restricted types of parameters
    - Restricted number of parameters
  - Complex IPC layered above
  - Idiosyncratic *presentations* separated from common *interfaces*
  - Specialized/synthesized paths
- VM
- Paging interface
- Scheduling
- Port model

Mach IPC Performance



Presentation vs. Interface in IPC: Example

Interface of "File" Object

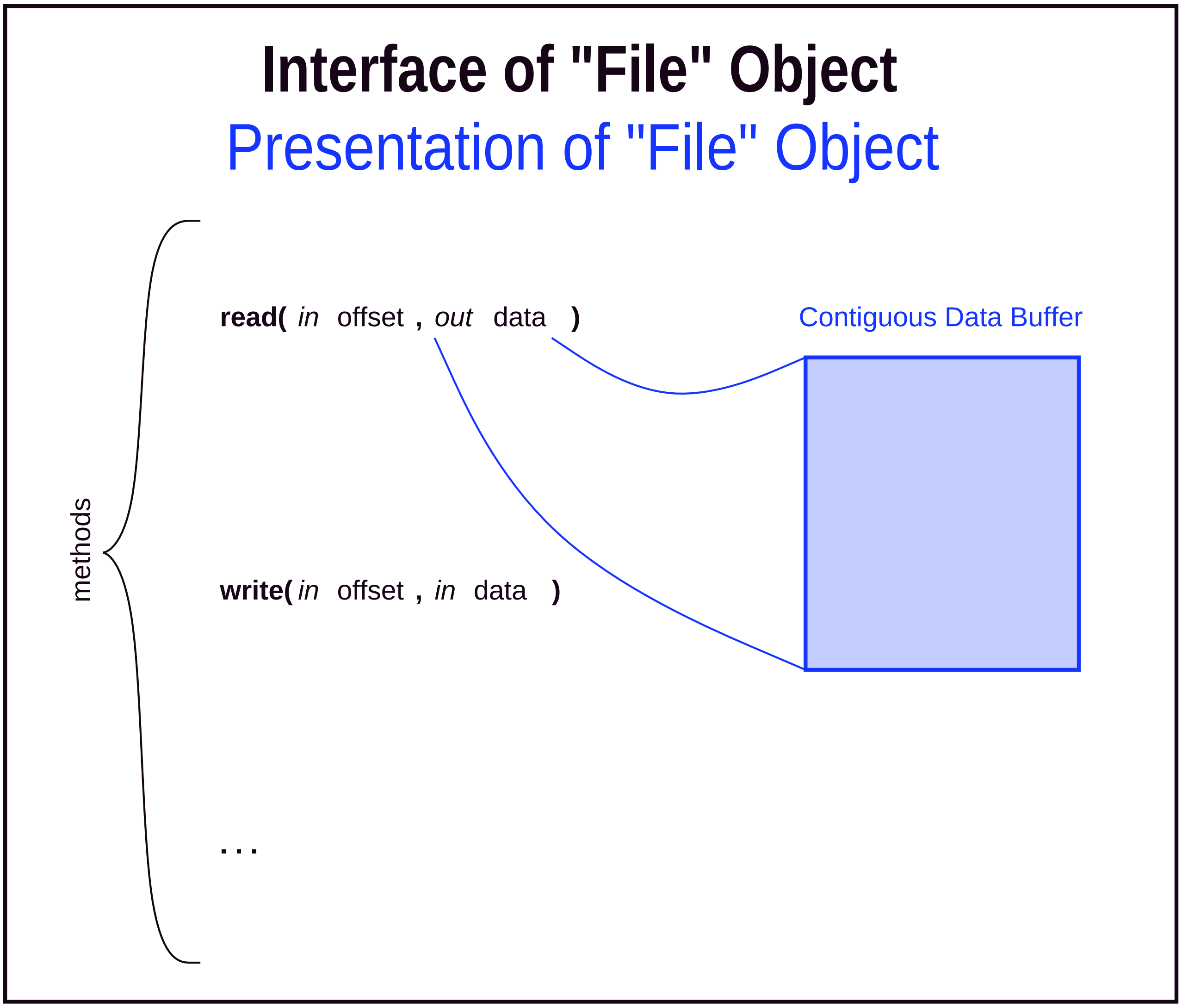
methods

read(*in* offset **,** *out* data **)**

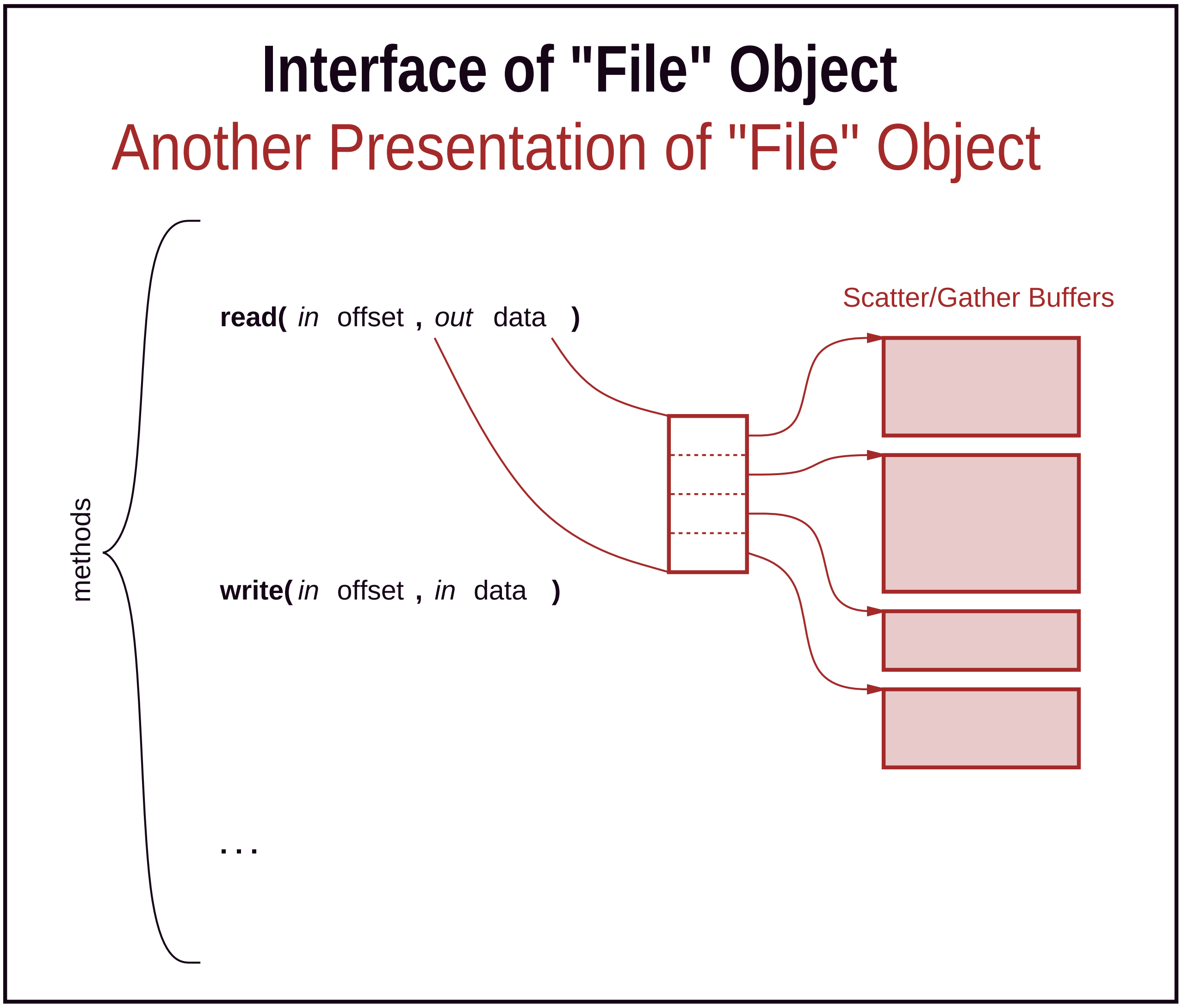
write(*in* offset **,** *in* data **)**

...

Presentation vs. Interface: Example (cont.)



Presentation vs. Interface: Example (cont.)



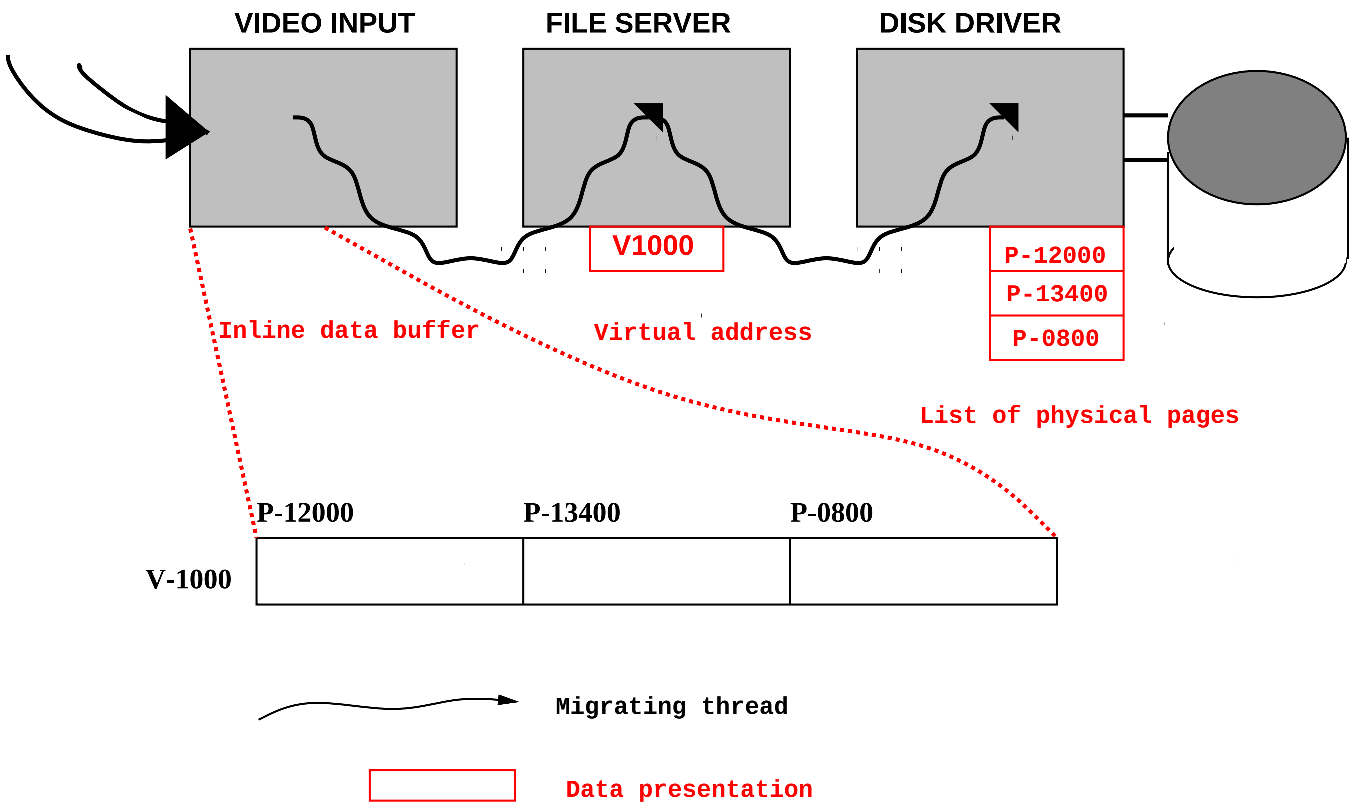
Presentation/Interface: Motivation

- Enhance interoperability.
- Clean many things up,
- Thereby making complexity manageable,
- Thereby providing opportunities for optimization:
  - Now easy to specify *additional* semantic attributes and presentation constraints
  - System has more info, which it can use to optimize end-to-end

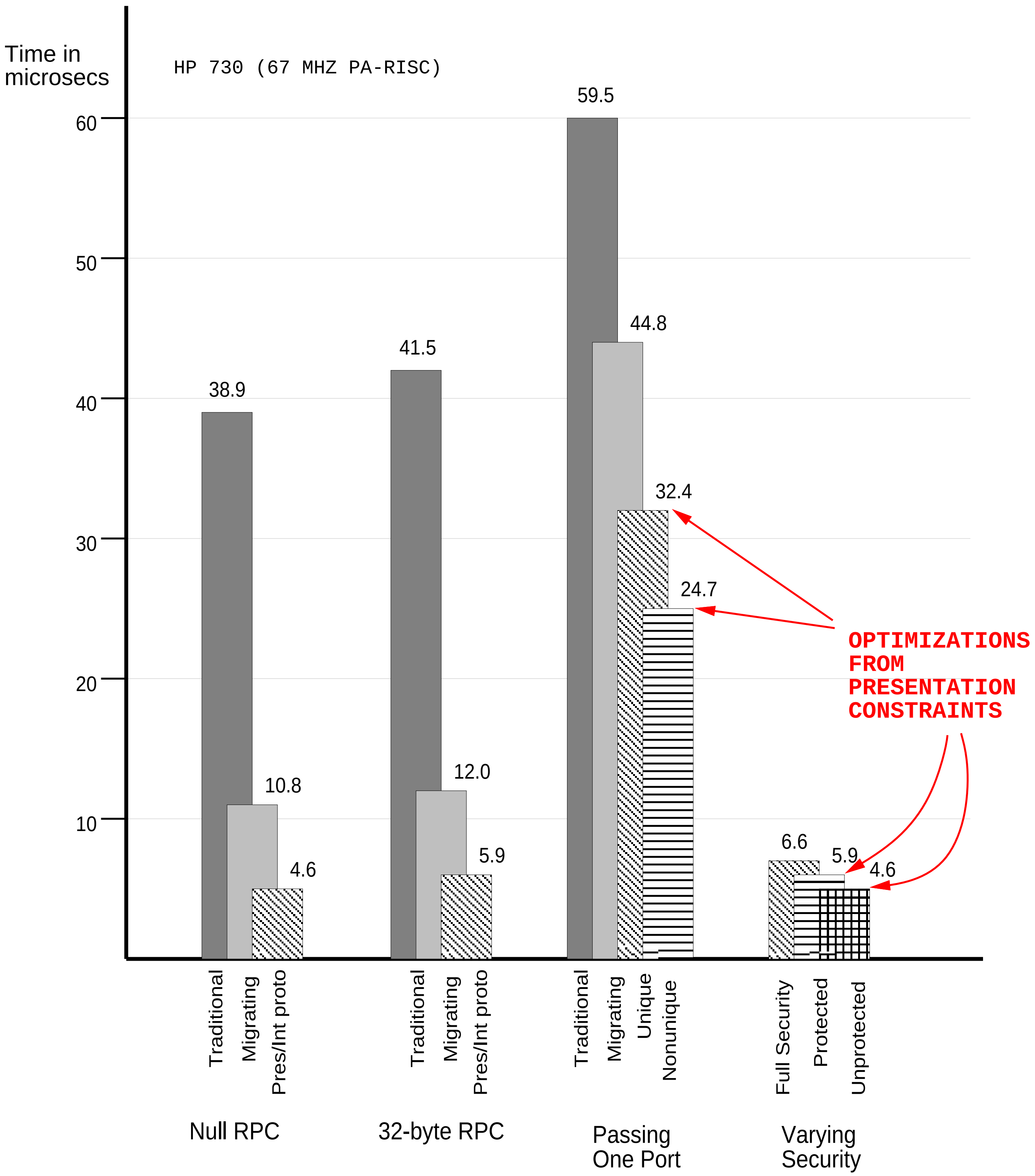
Presentation Optimization

Example: I/O

ONE INTERFACE (send/rcv); THREE DIFFERENT PRESENTATIONS



PRESENTATION OPTIMIZATIONS



Why a new VM system?

- Current VM is not as *flexible* as we would like
  - 1 task == 1 address space == 1 protection domain
  - kernel controls address space allocation
  - kernel controls physical memory allocation
  - kernel controls page replacement

Why a new VM system? (cont.)

- Current VM is not as *easy to use* as we would like
  - makes sharing inconvenient
  - external paging interface is complex

Why a new VM system? (cont.)

- Current VM is not as *efficient* as we would like
  - duplicates effort
  - implements general mechanisms not always needed or desired

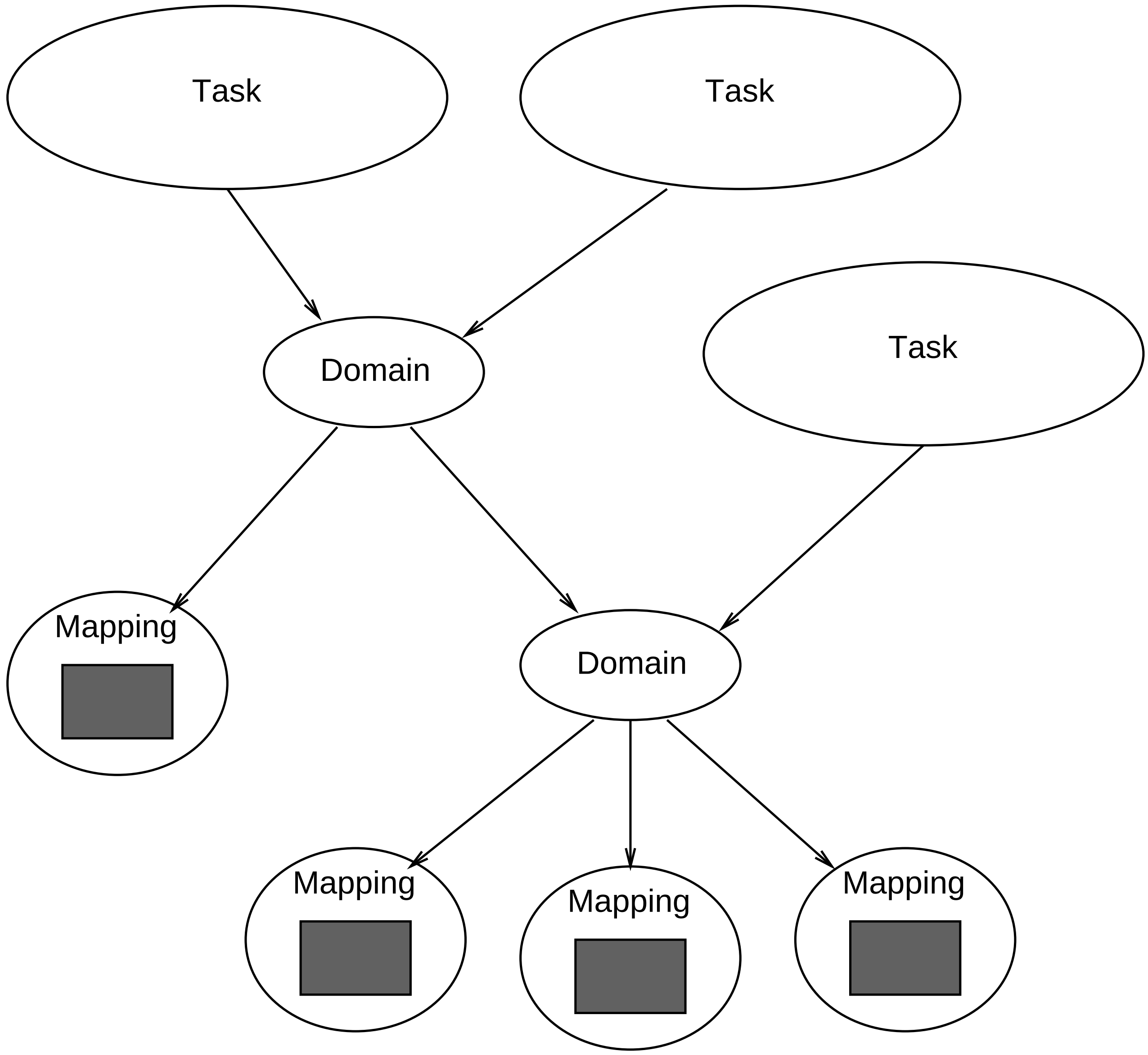
Our Approach

- Modularize
  - separate address space and protection domain management
  - split-off (and extend) physical memory management
  - reduced complexity of modules results in greater efficiency
  - exposing individual interfaces results in more flexibility

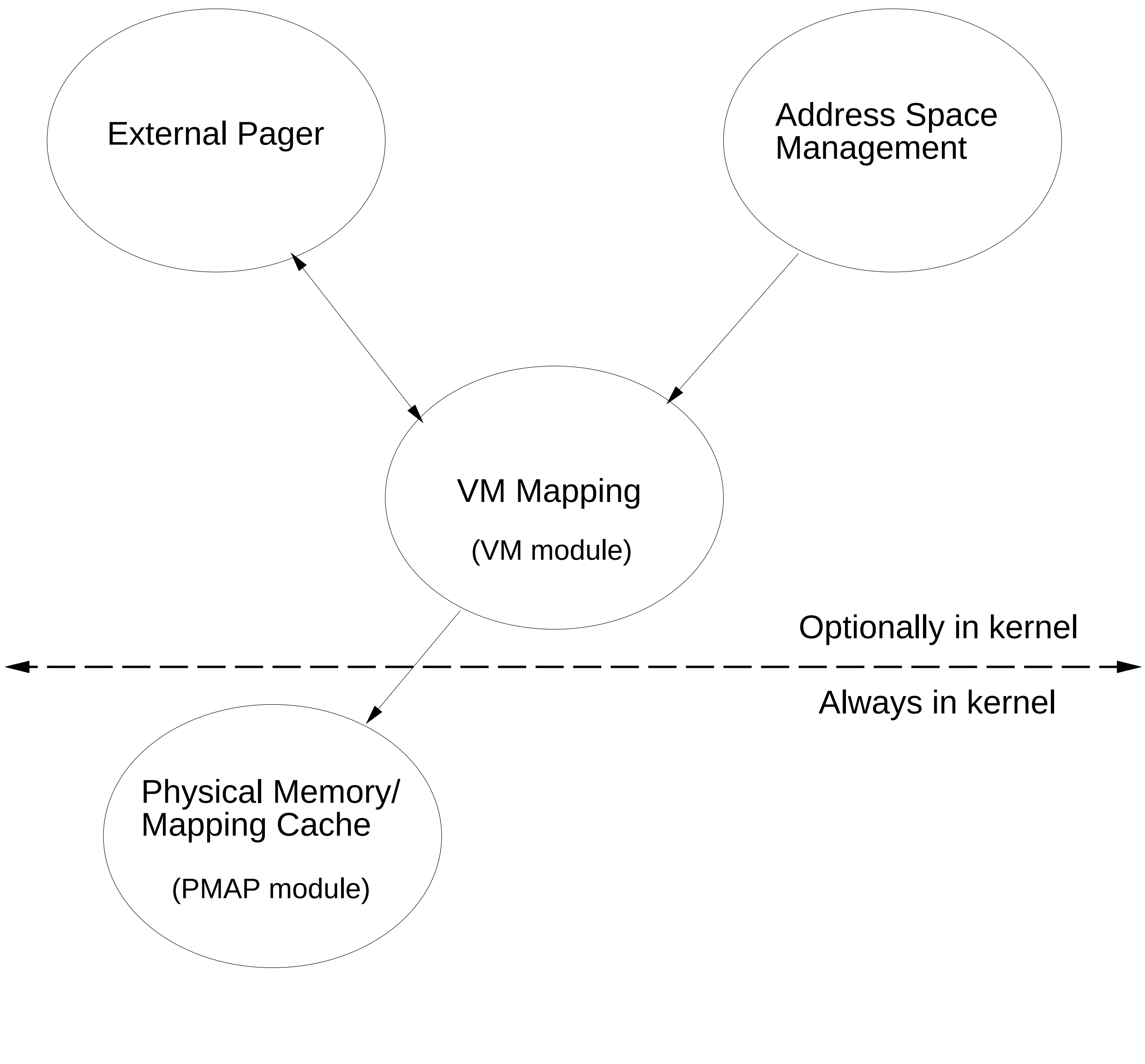
Our Approach (cont.)

- Make VM objects directly accessible
  - address mappings, domains, and physical memory pools referenced by capabilities (ports)
  - VM objects shared by passing capabilities
  - direct accessibility makes sharing convenient

Mapping Module Example: Sharing Tasks



VM System Modules



AS Management Example: Unix server in a SAS

- idea: clients and server exist in same AS making communication easier
- each client's top-level domain would contain mappings for private memory
- server's top-level domain would include private mappings and all client domains

Mapping Module Status

- Implemented a prototype SAS system and learned from that
- Early design work done
- Implementation starting soon

Modular Source and Functional Organization: Example

A “DOS Extender” is really an OS

- Runs “next to” DOS, supports real programs

- Mach4 provides VM, IPC, Trap handling

- In process of merging source trees

Distributed Shared Memory

- Incorporate high performance DSM mechanisms
  - protocols based on relaxed consistency models
  - Function shipping (RPC) versus data shipping (DSM)
  - compiler and module manager support
  - adaptive
- Investigate integration options
  - kernel (replace XMM), or server, or library?
  - Use Mach 4 optimized kernel mechanisms (IPC, exception handling, memory management, ...)
- Wild Idea: DSM as a distribution mechanism
- Status
  - Unix user-level version released 1/95 ("Quarks")
  - Fully integrated Mach version (kernel+user) 11/95

Flexibility: the Hammer

- OMOS – Object/Meta-Object Server
  - Understands *interfaces*
  - Intelligent and persistent *linker/loader*
  - Executable *objects*, specified and programmable with a scripting language
  - *Implementations* constructed at load time
  - Can use *legacy* code fragments
- Works on "user land" and kernel

Adaptive Specialization

- Executables, libraries, objects are little Scheme programs
- Scheme interpreter is OMOS, a
  - linker
  - loader
  - module manager
  - monitoring service
  - server
  - cacher (good thing!)
- "`ld`' produces the Scheme programs!
- Level of indirection, active entity
- Scheme: simple, elegant, structured, has some chance of being optimized

Release November '95

- New presentation/interface IPC system (CORBA IDL)

- New VM system

- Synchronous & symmetric external pager interface

- Prototype SAS using OMOS module mgr.

- Prototype efficient DSM in Mach

- Unencumbered kernel and Lites server on PA-RISC, maybe on x86. (1996: x86 for sure, other platforms likely)

Software Components of Potential Use to Research Community

- Mach 4.0 kernel
- Intelligent linker/loader (OMOS)
- High level scripting language for module composition (OMOS)
- Portable, efficient DSM system
- Experimental SAS system
- Unencumbered Unix server (Lites)
- CORBA IDL compiler