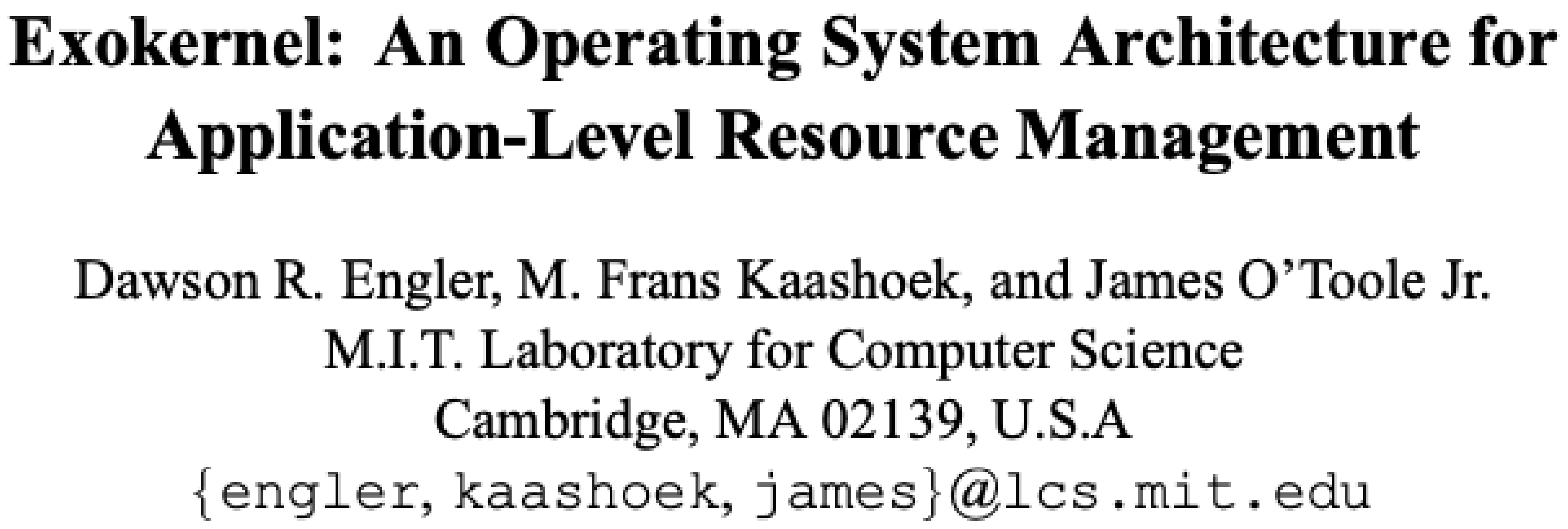
CS 60038: Advances in Operating Systems Design
Lecture 2: Exokernel - Application Level Resource Management

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

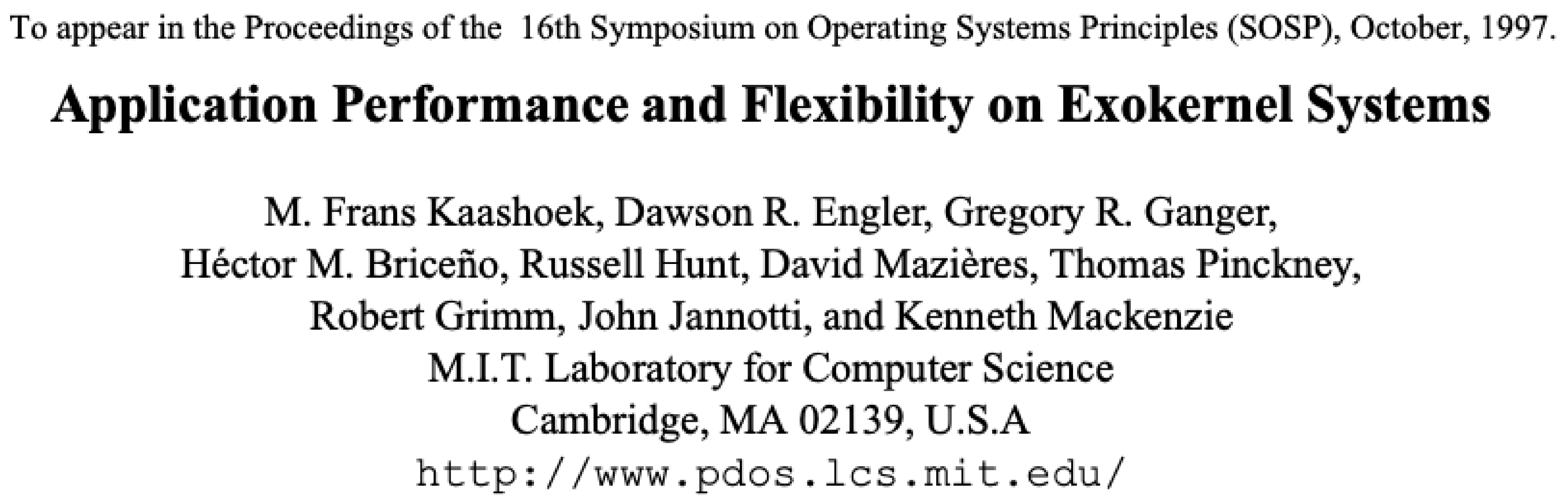
Exokernel (ACM SOSP 1995)



Exokernel: An Operating System Architecture for Application-Level Resource Management

Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.
M.I.T. Laboratory for Computer Science
Cambridge, MA 02139, U.S.A
`{engler, kaashoek, james}@lcs.mit.edu`

Exokernel (ACM SOSP 1997)



To appear in the Proceedings of the 16th Symposium on Operating Systems Principles (SOSP), October, 1997.

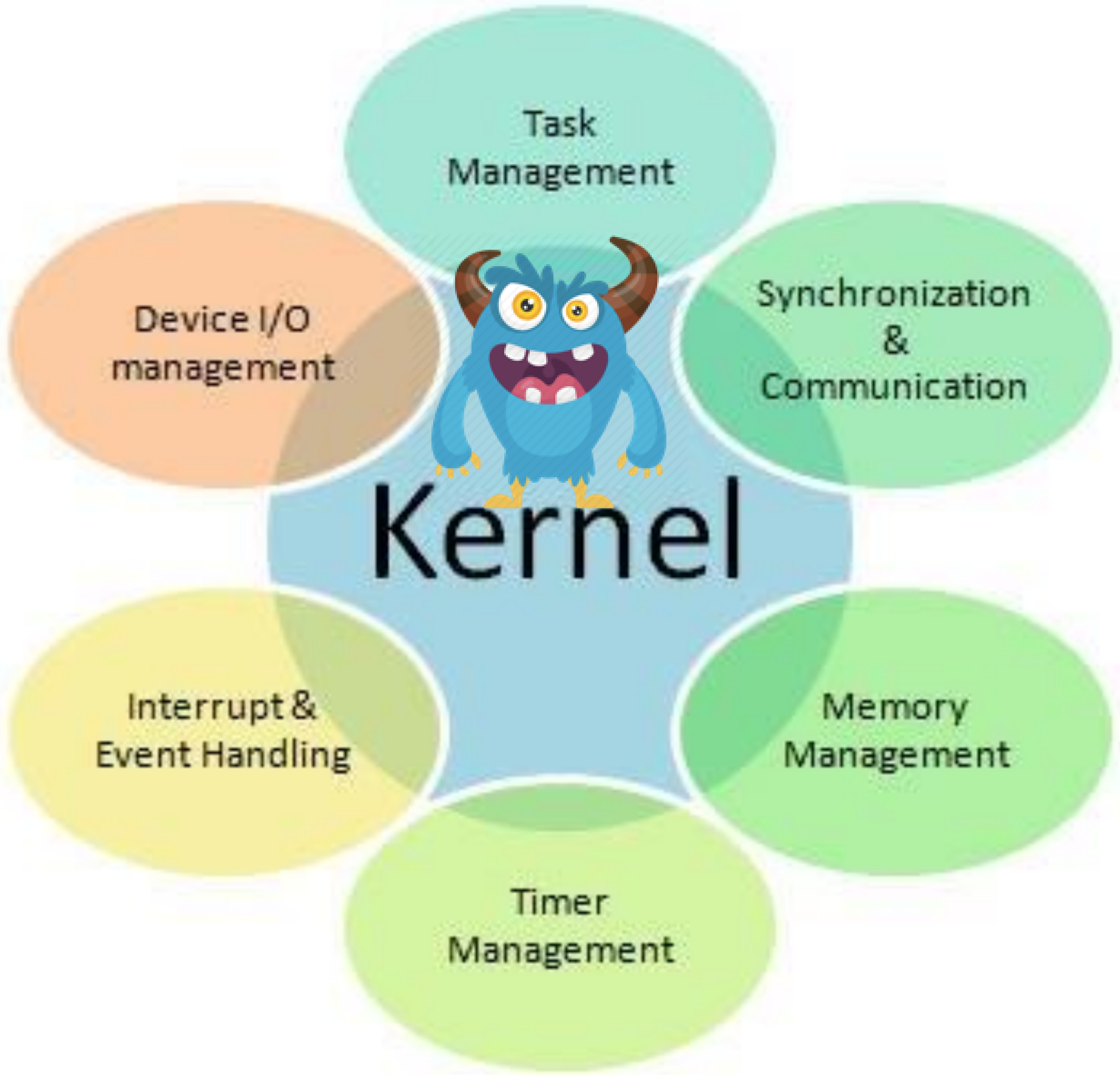
Application Performance and Flexibility on Exokernel Systems

M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger,
Héctor M. Briceño, Russell Hunt, David Mazières, Thomas Pinckney,
Robert Grimm, John Jannotti, and Kenneth Mackenzie
M.I.T. Laboratory for Computer Science
Cambridge, MA 02139, U.S.A
`http://www.pdos.lcs.mit.edu/`

Traditional OS

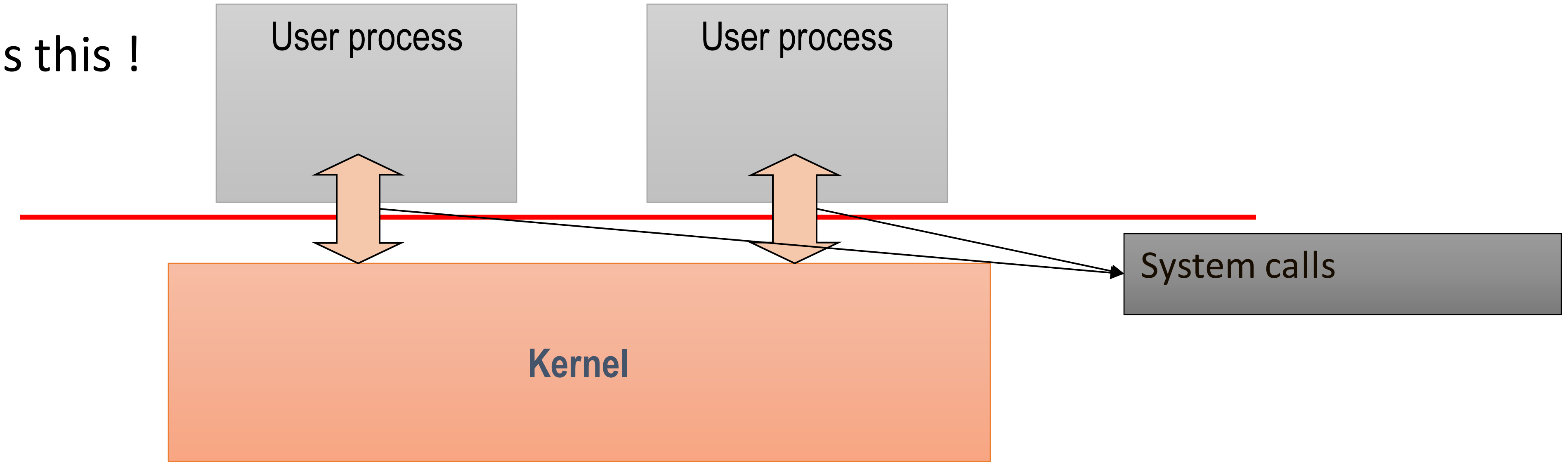
- Only privileged servers and the kernel can manage system resources
- Both resource management and protection are done by kernel
  - Centralized control
- Untrusted applications are limited to the interface
  - it denies applications the advantages of domain-specific optimizations,
  - it discourages changes to the implementations of existing abstractions, and
  - it restricts the flexibility of application builders, since new abstractions can only be added by awkward emulation on top of existing one

The Kernel - The Almighty



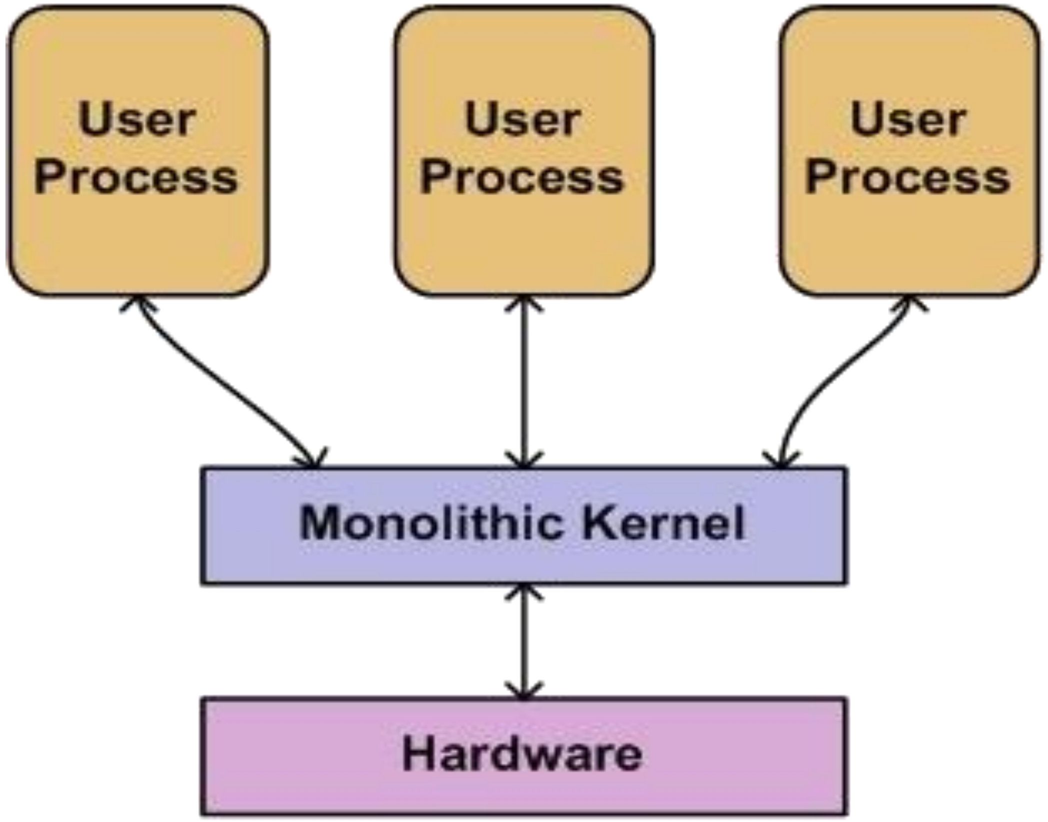
Traditional OS

- An interface designed to accommodate every application must anticipate all possible needs
- Flawed - Protection versus Resource Management
- Solution:
  - Allow applications enough control over resources by separating protection from management
  - “Exokernel” does this !



Monolithic Kernel

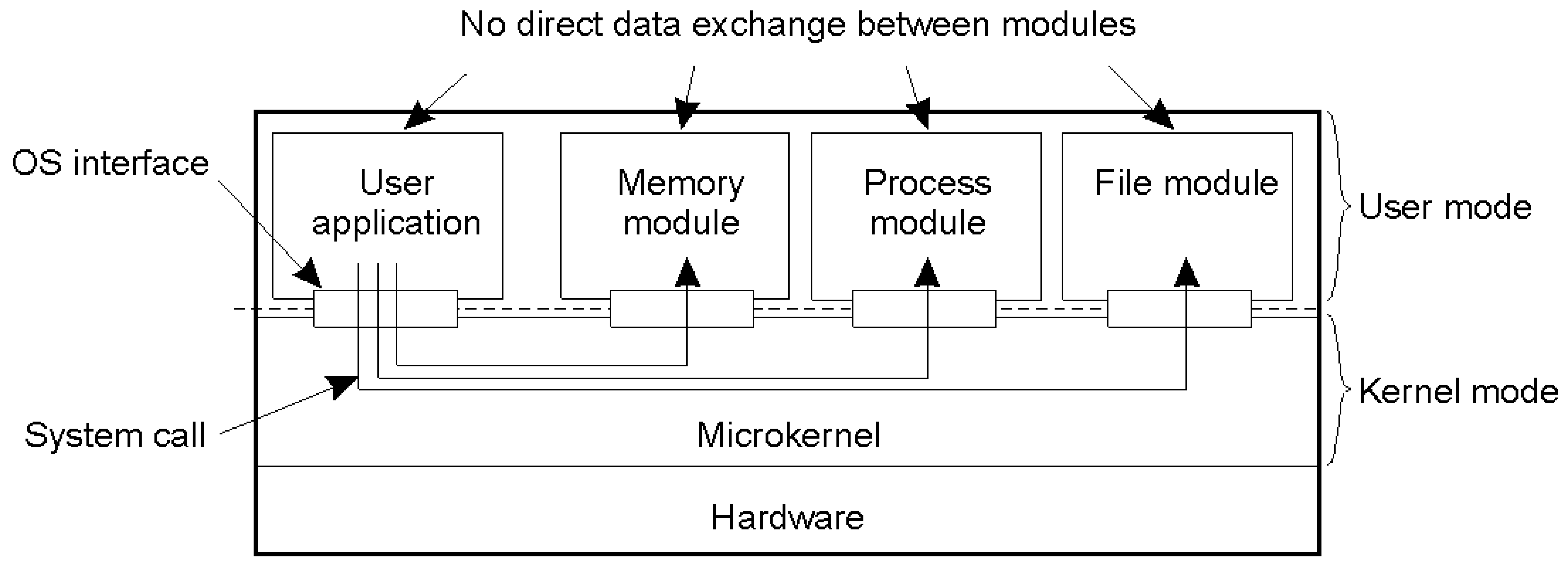
- Kernel takes care of almost all the system tasks
- Applications do not have control over resources
- Example:
  - Windows 9x series: Windows 95, 98
  - BSD: FreeBSD, OpenBSD
  - Linux



Ref: Kaashoek et al.

Microkernel

- Runs most of the operating system services at the user space.
- Parts that require privilege (IPC) are in kernel mode and other critical parts (FS, Network Stack) in user mode.
- Example: L4 microkernel, Mach (Predecessor of MacOS)
- Performance issue !



Ref: Tanenbaum's distributed systems.

Problems of Fixed High Level Abstractions

- *Hurts application performance*
  - No single way to optimize the performance of all applications
  - Example: Page replacement algorithm -- which one is the most popular and commonly used in a general purpose OS?
  - Can you think of a better page replacement algorithm for relational databases?
  - Application specific caching can improve performance as high as 45% [Cao et al. OSDI'94]

Problems of Fixed High Level Abstractions

- *Hurts application performance*
  - No single way to optimize the performance of all applications
  - Example: Page replacement algorithm -- which one is the most popular and commonly used in a general purpose OS?
  - Can you think of a better page replacement algorithm for relational databases?
  - Application specific caching can improve performance as high as 45% [Cao et al. OSDI'94]

- *Hide Information from applications*
  - Example: System hides page faults and timer interrupts from application program
  - Implications: Implement a lightweight thread on top of a heavyweight process -- the thread suffers -- why?
  - Read *Anderson et. al. Scheduler activations: Effective kernel support for the userlevel management of parallelism, SOSP'91*

Problems of Fixed High Level Abstractions

- *Limit the functionality of applications*
  - Multiple applications use the same abstraction for decades
  - It is difficult to change the abstractions once they got standardized
  - A good example: Network protocol stack -- people knows TCP sucks but they are bound to use it !
  - Thanks to Google to dominate the industry -- we are having QUIC now !
  - Read the QUIC paper from Google - *The QUIC Transport Protocol: Design and Internet-Scale Deployment*

An End-to-end Argument

"Applications know better than operating systems what the goal of their resource management decisions should be and therefore, they should be given as much control as possible over those decisions"

Exokernel -- Separate Protection and Management

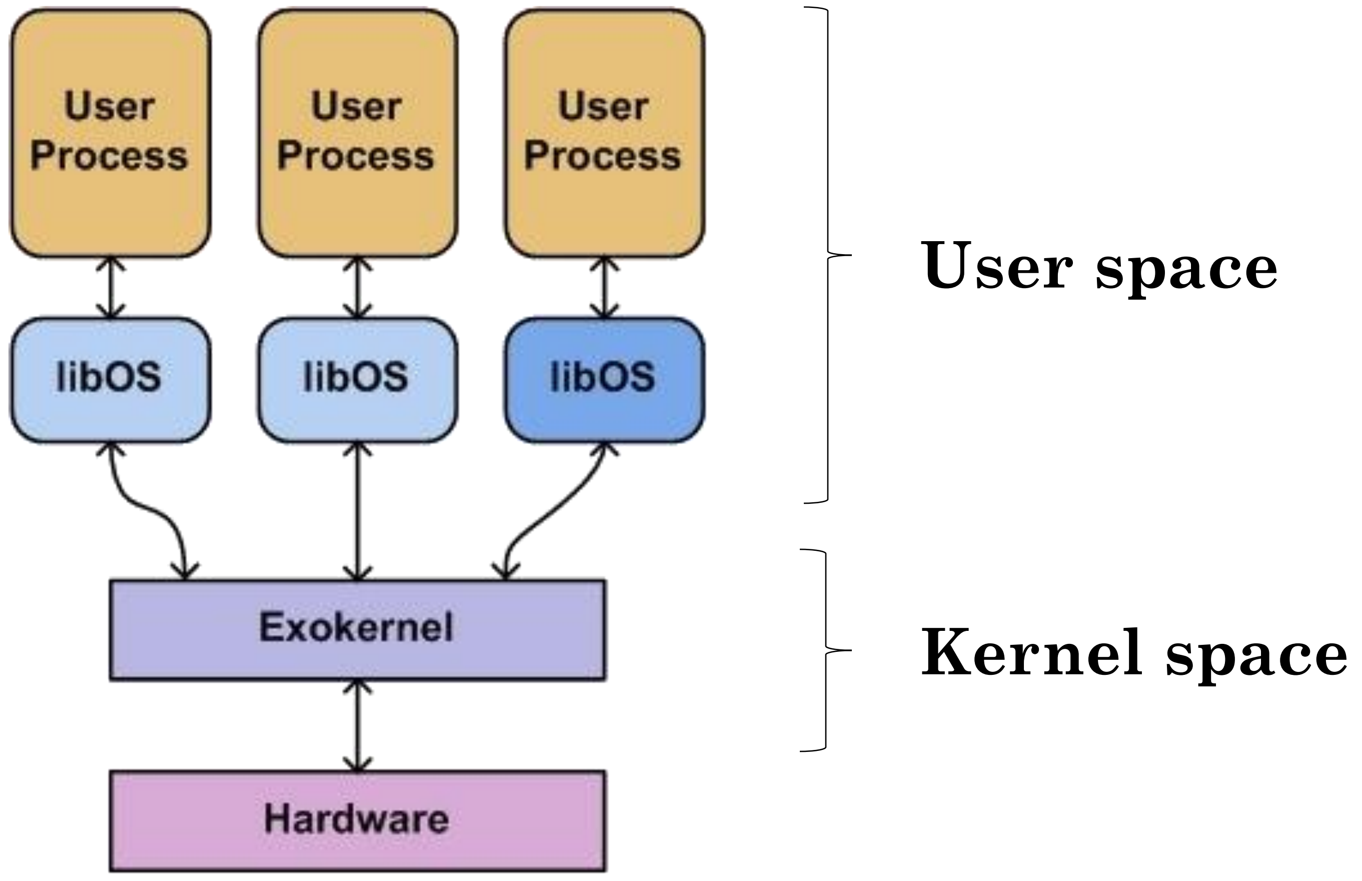
- **Management**: Allocate resources based on the requirement
  - Example: An application asks for a memory block
  - However, read-write strategy over that memory block will be decided by the application
- **Protection:** Ensures no conflict in resource allocation
  - Maintain ownership of resources
  - Example: Do not allocate a memory block if it is already allocated to another application

Exokernel

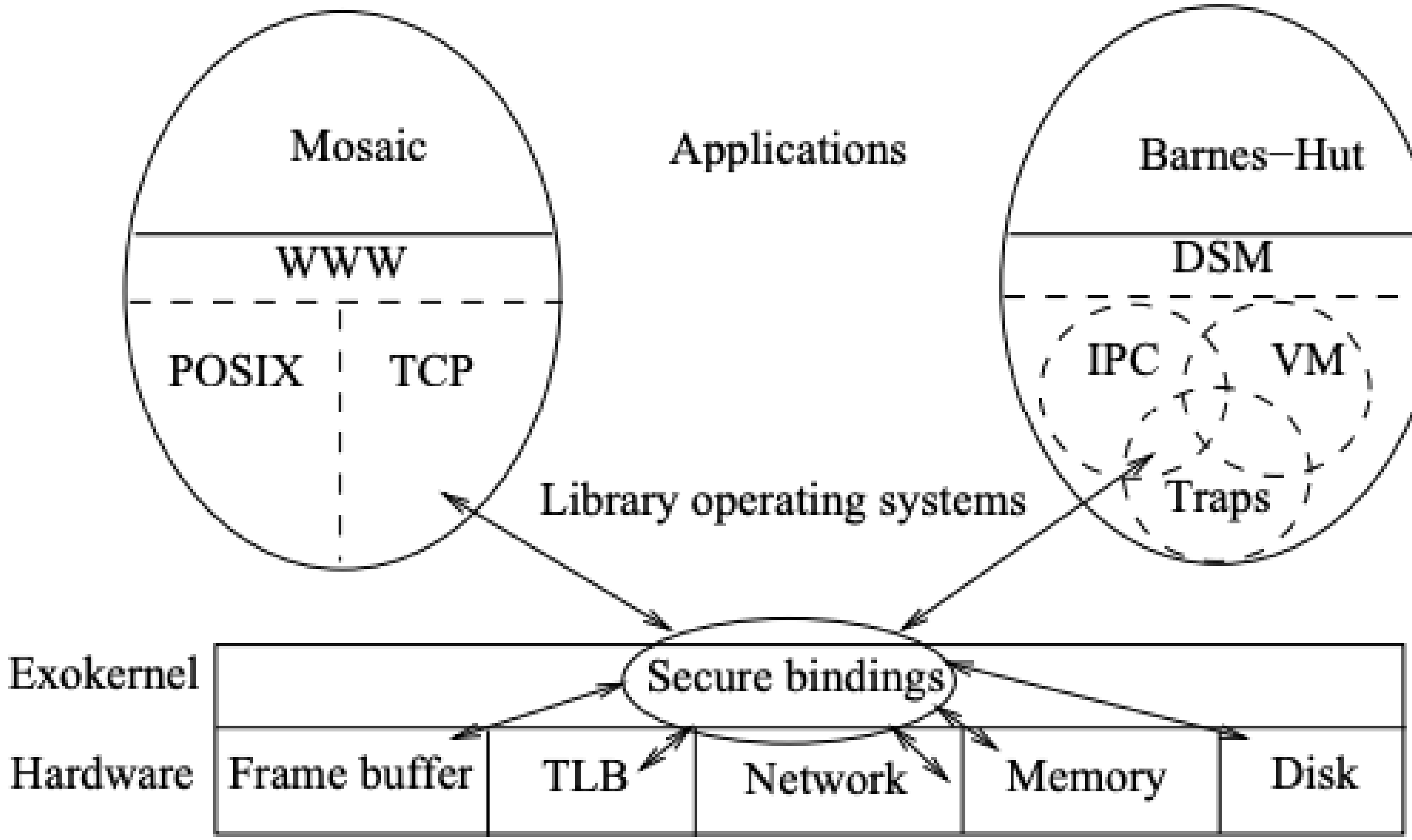
- Separates resource management from protection
  - Normal kernel does both
- Kernel
  - protect the resources
- Application
  - Manage the resources
  - Virtual memory, file system etc. are in application libraries
  - Gives untrusted software as much control over hardware and software resources as possible
  - Specialized applications can gain high performance without sacrificing the unmodified UNIX program

Exokernel: Example

- Application manages its disk-block cache and kernel allows cached pages to be shared securely between applications

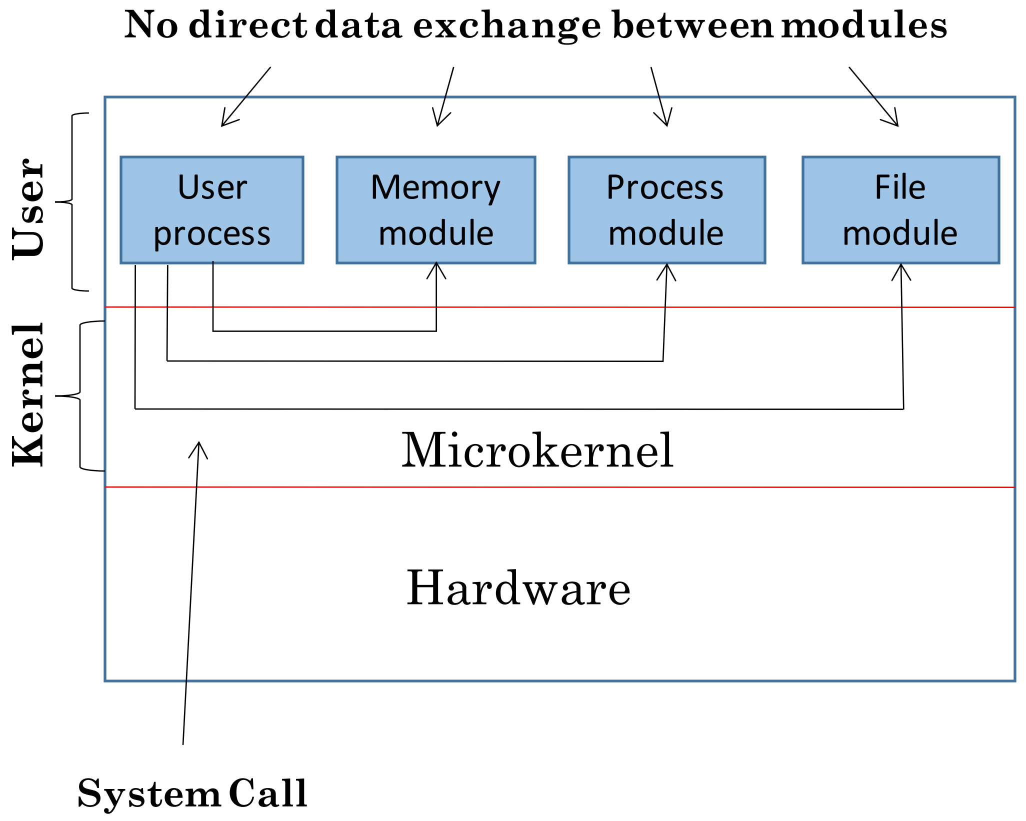


Exokernel - Application to Hardware

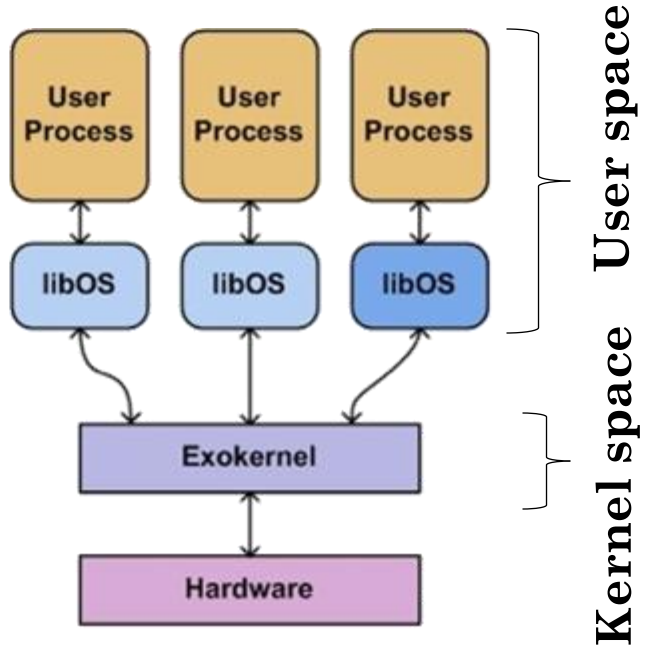


Microkernel vs. Exokernel

Microkernel



Exokernel



End-to-end Argument in System Design

Why are we learning Exokernel?

- Exokernel was not very successful commercially or in OS revolution
- However, very recently the system community has started taking a move towards the “End-to-End Arguments” concepts
- Example 1: Network Function Virtualization -- Pull out network stack out of OS kernel and make it an application program
- Example 2: Storage Virtualization -- Use application program (Like GlusterFS) to manage your storage

Extensible Operating systems

- Extensibility lets new functionalities to be included in the operating systems
- Goal is to let applications safely modify system behavior for the applications' own need
- Different approaches to extensible OS:
  - Exokernel (MIT)
  - SPIN (UW)
  - VINO (Harvard)
  - L4 (IBM)
  - Fluke/OSKit (Utah)

3 Real Exokernel Systems

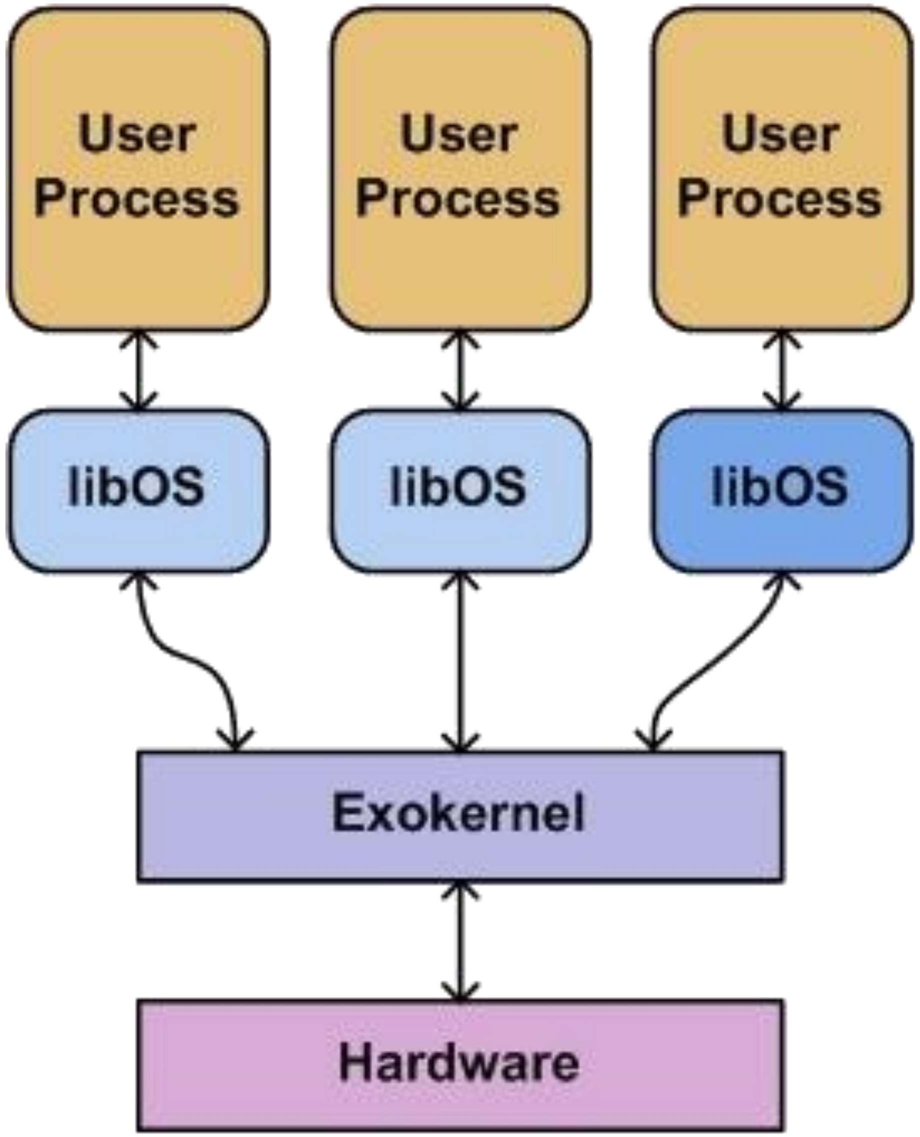
- XoK (EXoKernel)
  - For Intel x86 based computers
  - Multiplexes physical resources (disk etc)
- Aegis
  - Runs on MIPS based DEC Stations
- Glaze
  - For the Fugu microprocessor

libOSes

- Library operating systems.
- “Unix as a library”
- Can implement traditional OS abstraction.
- Most application programs will be linked to libOSes of their choices instead of communicating with the exokernel.
- Unprivileged libraries – can be modified or replaced at will.
- Different libOSes can coexist on the top of same exokernel.
    - This allows system to emulate behaviors of several conventional OSs.

ExOS

- ExOS is Xok's default library.
- Much code borrowed from OpenBSD.



Separating Protection from Management

- Track ownership of resources
- Ensure protection by guarding all resource usage or binding points
- Revoke access resources

- Exokernel employs three techniques to achieve the above tasks --
  - ***Secure Bindings***: Securely bind LibOSes to machine resources
  - ***Visible Revocation***: Allows LibOSes to participate in a resource revocation protocol
  - ***Abort Protocol***: Break secure bindings of uncooperative LibOSes by force

Exokernel Principles

- Separate resource protection and management
  - Exokernel and libOSes
  - Minimum resource management as required by protection (allocation, revocation and ownership)
- Expose allocation
  - Applications allocate resources
  - Kernel allows the allocation requests
- Expose names
  - Exokernels use physical names wherever possible
- Expose revocation
  - Let (good) application choose which instance of resource is to give up
- Expose information
  - Expose all system information and collect data that application can not easily derive locally

Policy Decisions

- Exokernel hands over resource policy decisions to library operating systems
- An application (or collection of cooperative applications) can make decisions about how best to use these resources
- Exokernel must include policy to arbitrate between competing library operating systems
  - Forced revocation of resources if it deems absolutely necessary (prevent starvation)
- **Points to note: Policy decisions are with libOSes until there is no conflict, exokernel overrides the decisions in case of a conflict**

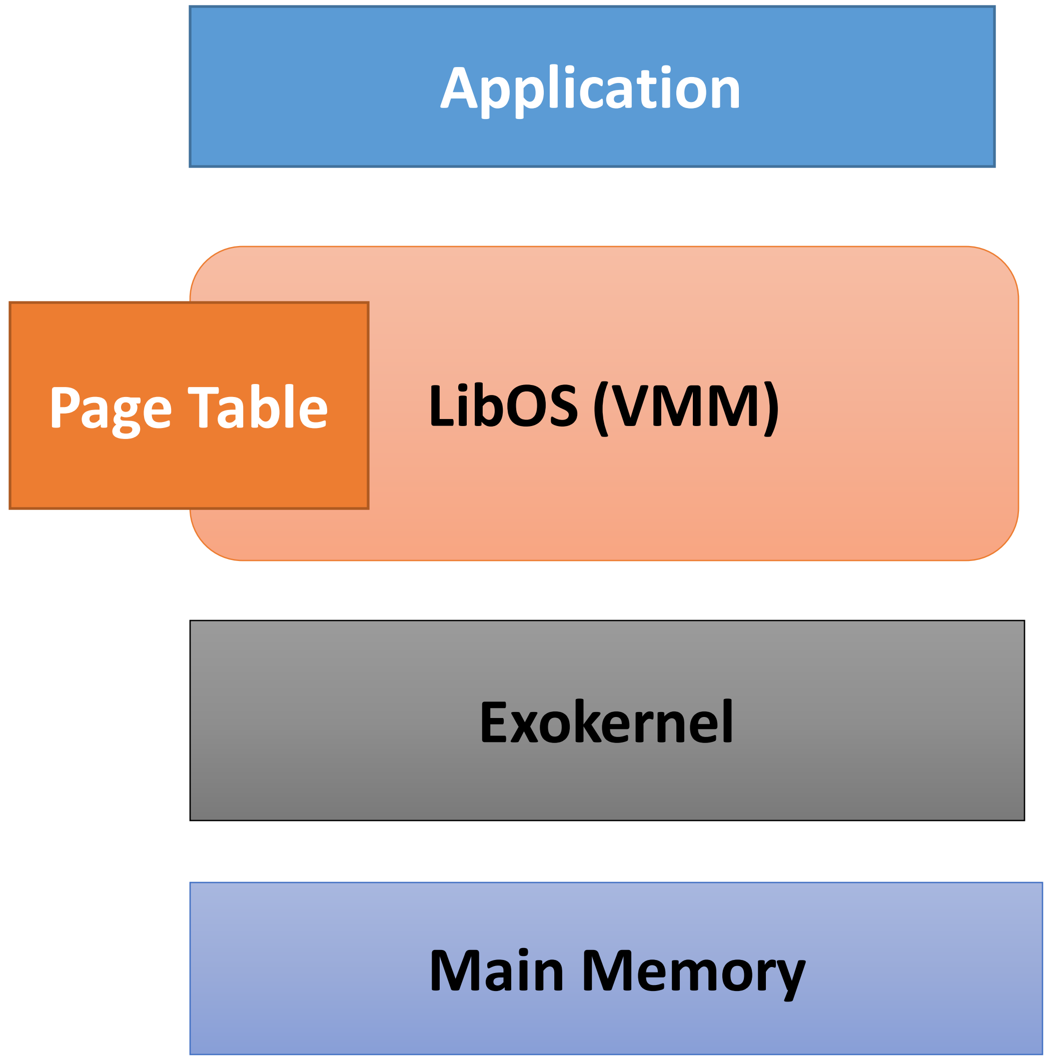
Secure Bindings - Multiplex Resources Securely

- Provides protection to mutually distrustful applications

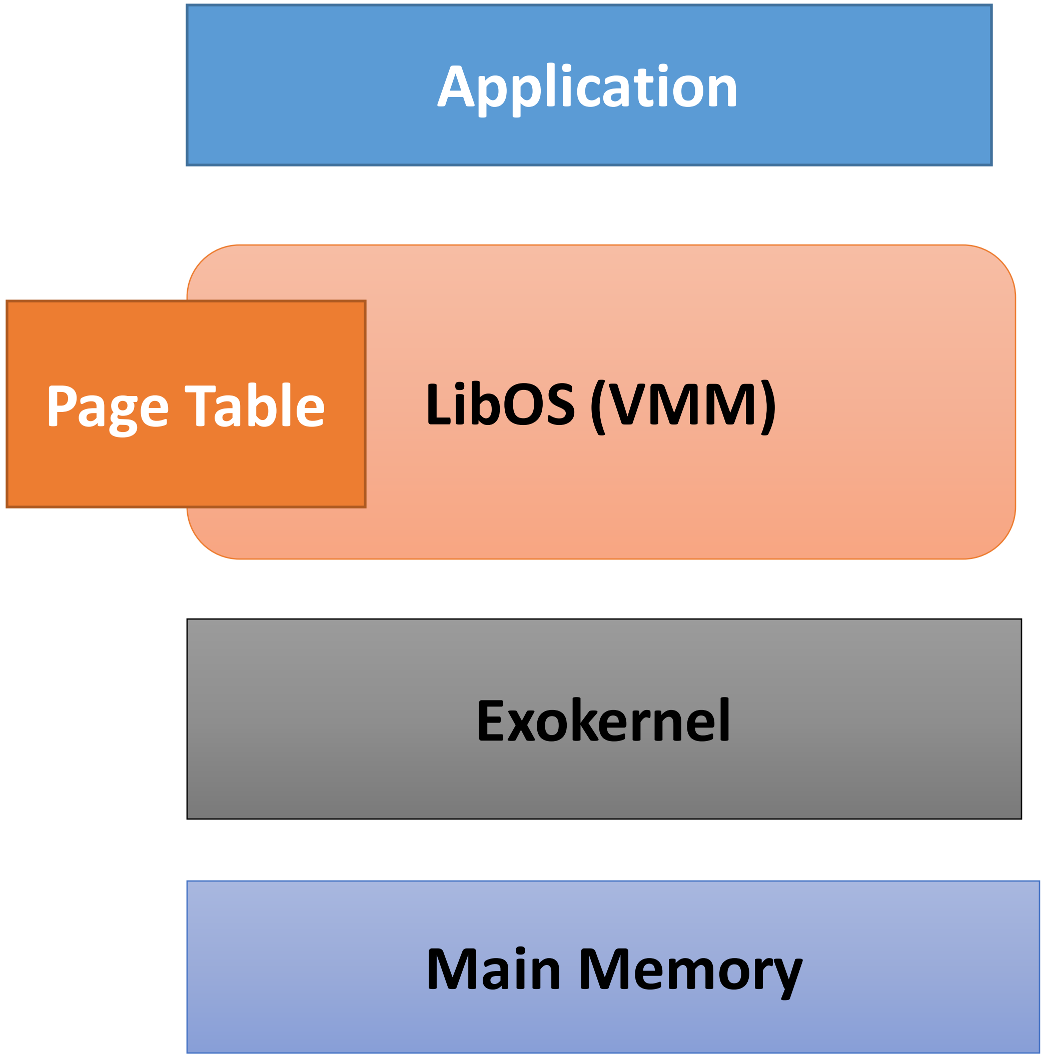
- Exokernel guards each resources to provide protection -- allows libOSes to bind to resources using *secure bindings*

- *Secure Bindings* - a protection mechanism that decouples authorization from the actual use of resources
    - Simple operations that the kernel or the hardware can implement quickly
    - Performs authorization only at bind time

Secure Bindings - A Page Fault Occurs

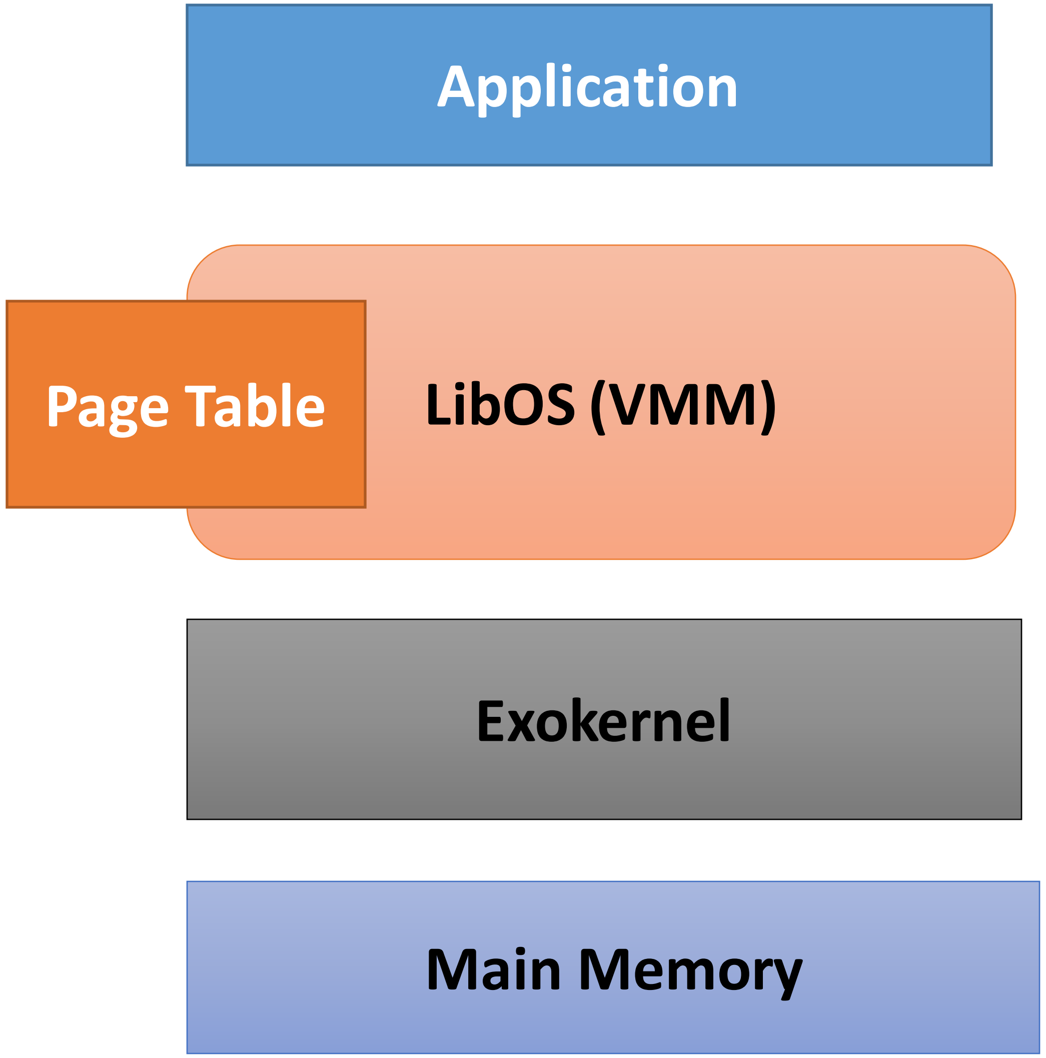


Secure Bindings - A Page Fault Occurs

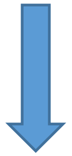


Request for a memory access

Secure Bindings - A Page Fault Occurs

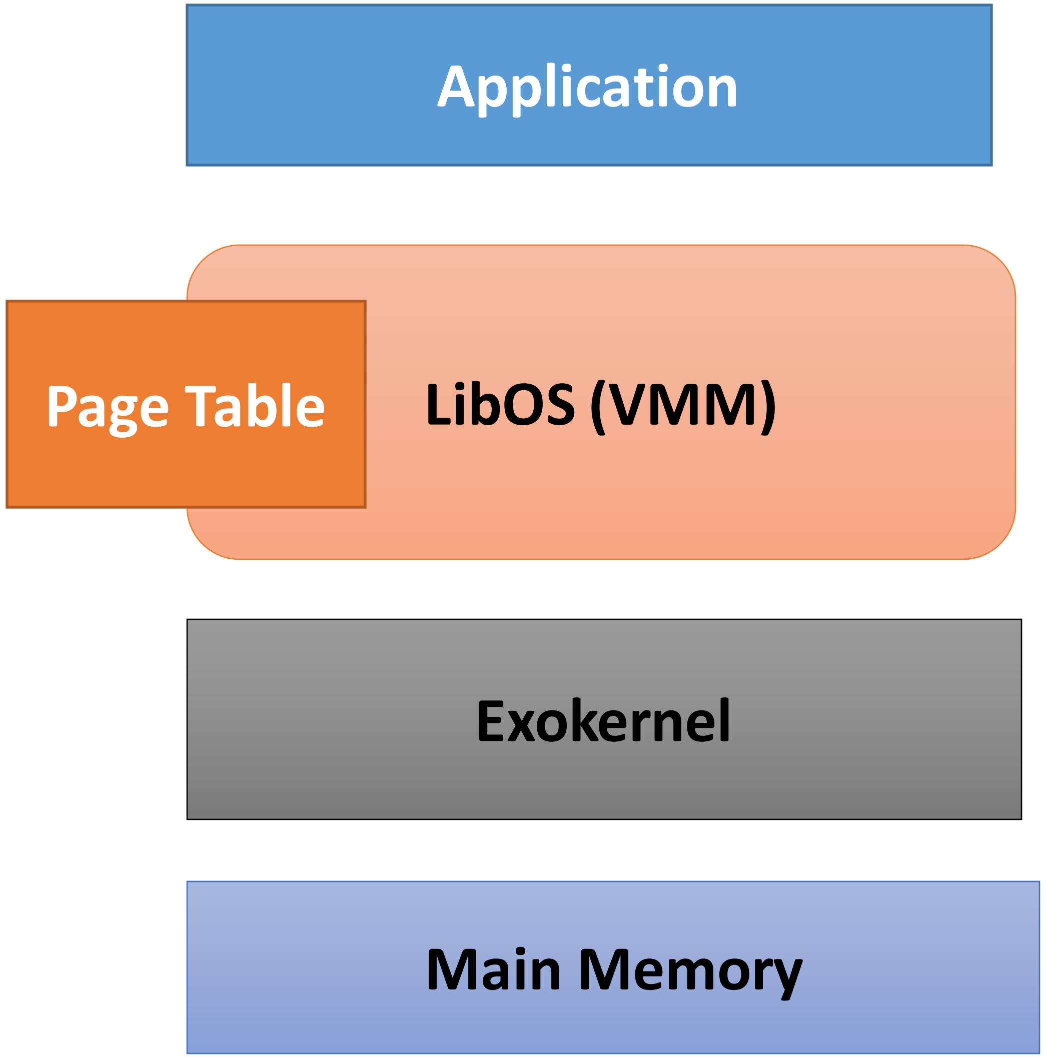


Request for a memory access



A page fault occurs

Secure Bindings - A Page Fault Occurs



Request for a memory access

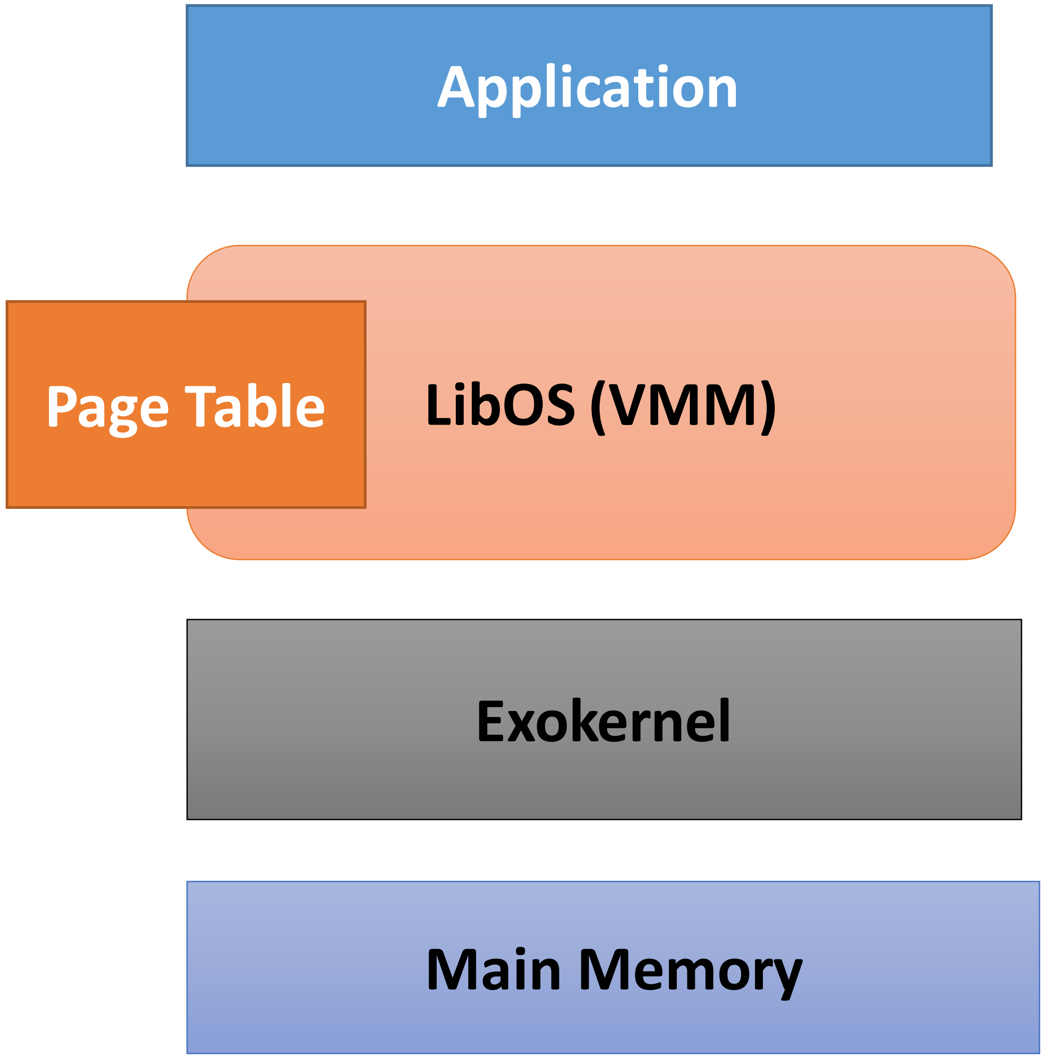
$\Downarrow$

A page fault occurs

$\Downarrow$

Page Table Lookup, Decide the main memory frame to load

Secure Bindings - A Page Fault Occurs



Request for a memory access

↓

A page fault occurs

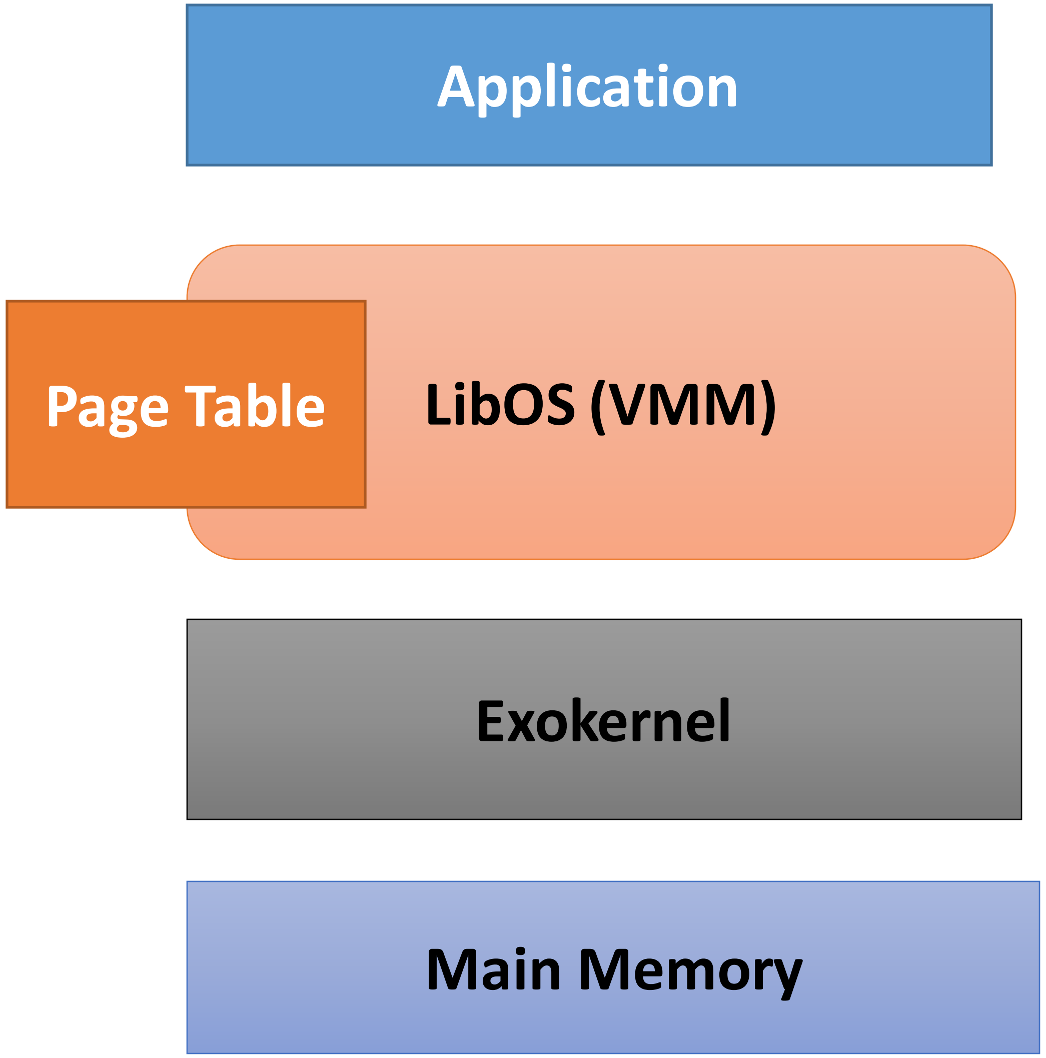
↓

Page Table Lookup, Decide the main memory frame to load

↓

Request for the main memory frame

Secure Bindings - A Page Fault Occurs



Request for a memory access

↓

A page fault occurs

↓

Page Table Lookup, Decide the main memory frame to load

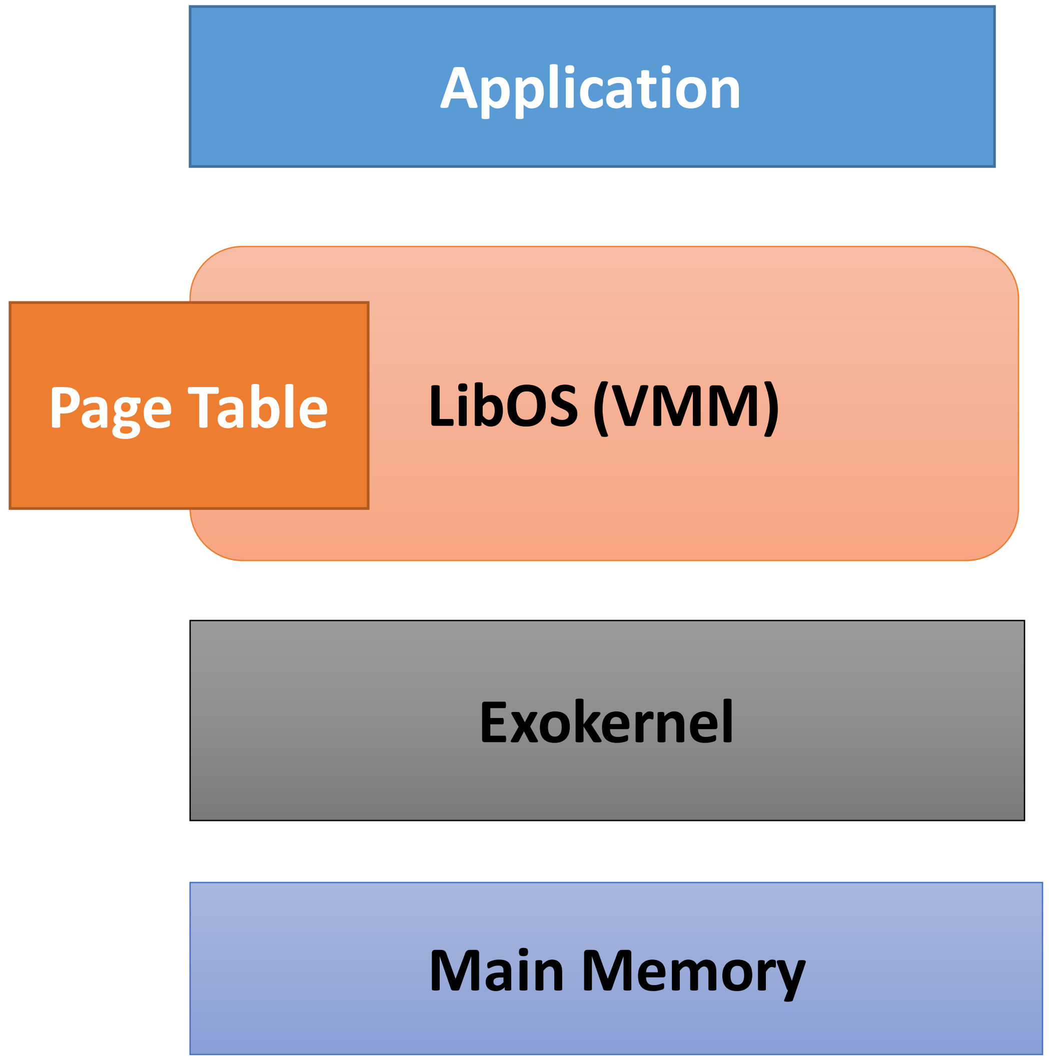
↓

Request for the main memory frame

↓

Allocate the memory, load the frame to main memory, **bind the memory block with the application (Secure binding)**

Secure Bindings - Access Valid Memory Address



Request for a memory access

↓

A valid page is available

↓

Page Table Lookup, to decide the memory address

↓

Request for access to the memory address

↓

Access the memory, validate the entry of secure binding to give access

Secure Binding - Multiplexing the Network

- How do you share the network among two applications?

Secure Binding - Multiplexing the Network

- How do you share the network among two applications?

- Hardware based solution
  - Use virtual circuit switching (Similar to ATM networks)
  - Use TDM or FDM if supported by the network

- Software based solution
  - Use packet filtering -- separate out the packets from different applications and then schedule them based on some policy

Downloading Code

- Used to improve performance
  - Elimination of Kernel crossing by downloading the code at Kernel
  - The execution time of the downloaded code can be bounded

- Example: Packet filters -- download the executable of packet filter as a part of the Kernel and execute whenever required.

Visible Revocations

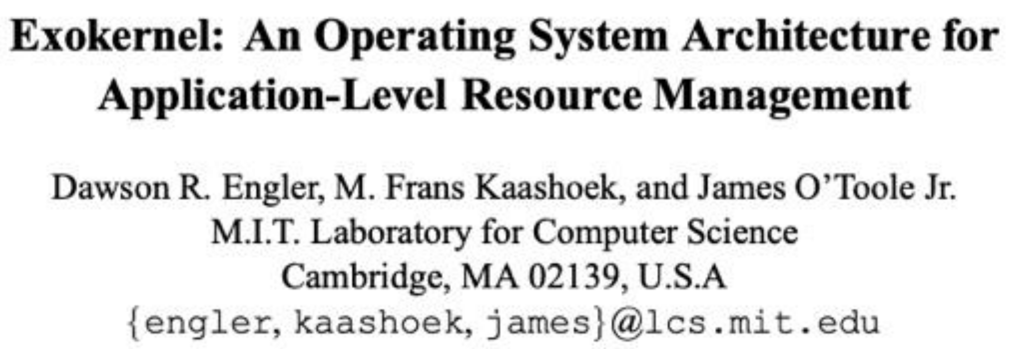
- Reclaim the resources and break their secure bindings
- Traditional OS: Revocation is not under the control of an application
  - Allocate or deallocate physical memory without informing the applications
- Exokernel uses visible revocation -- revocation decisions are informed to the application before they are revoked
- Advantages: Applications can decide how to react to such revocations
  - Revocation of the processor -- application decides which part of the PCB to save before the context switch

Visible Revocations

- *Invisible Revocations* are better sometime
  - *Can you give an example?*
- **The abort protocol**: Revoke the resources forcibly when the application does not responds for a time threshold

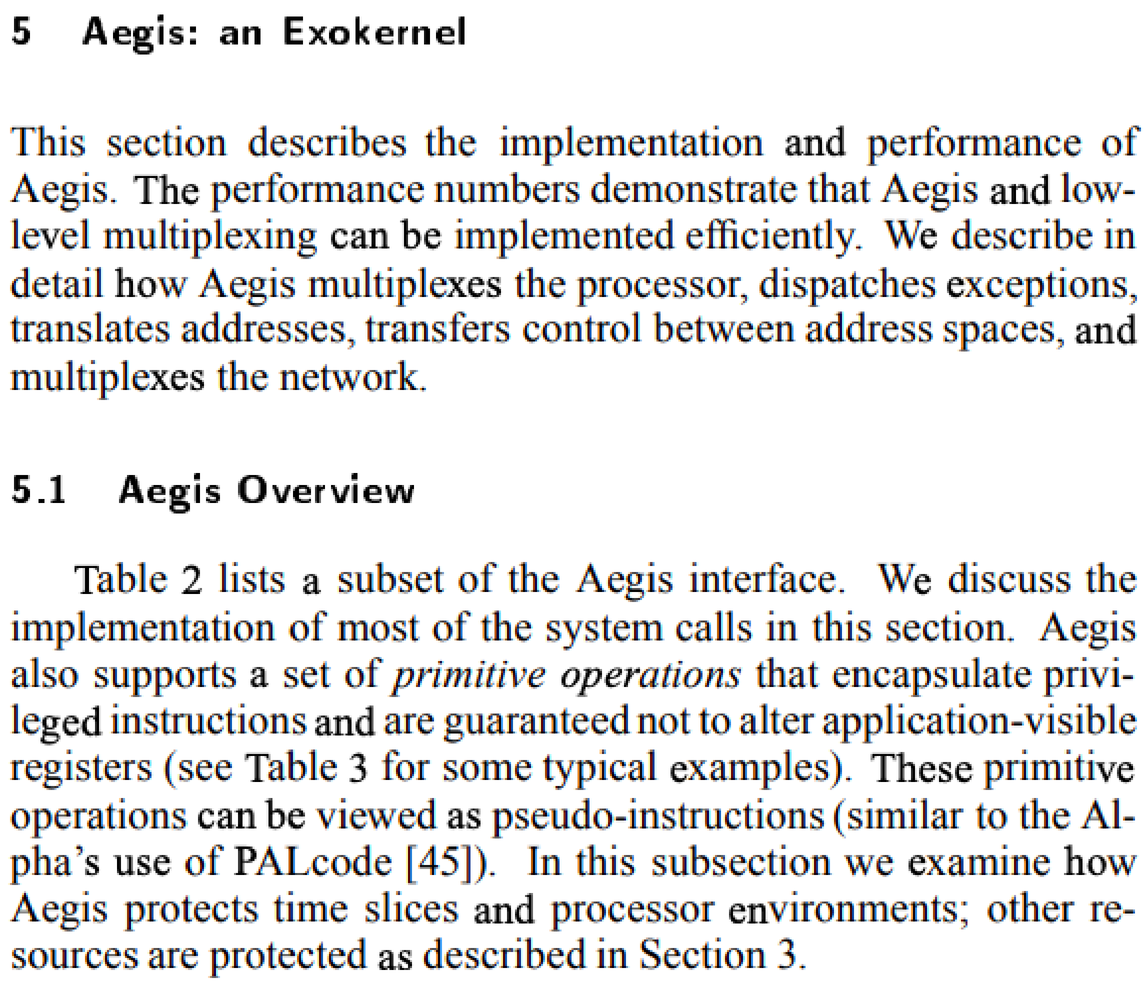
Self Study - 1

- Read the Aegis OS development details from the Exokernel paper



Exokernel: An Operating System Architecture for Application-Level Resource Management

Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr.
M.I.T. Laboratory for Computer Science
Cambridge, MA 02139, U.S.A
{engler, kaashoek, james}@lcs.mit.edu



5 Aegis: an Exokernel

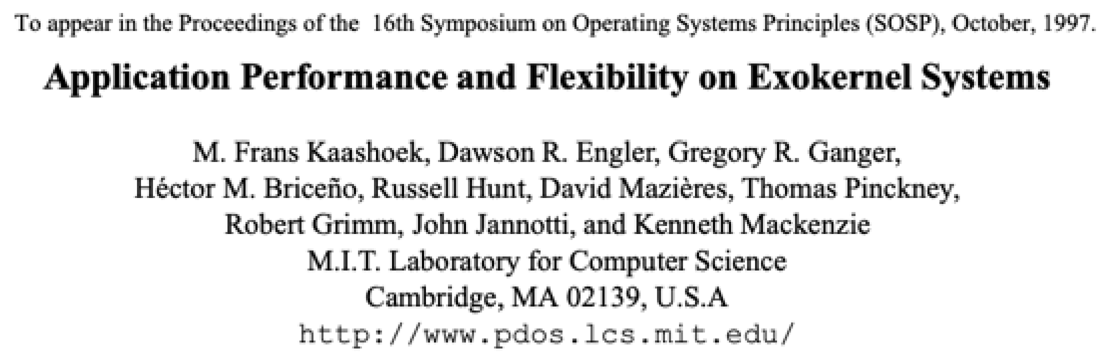
This section describes the implementation and performance of Aegis. The performance numbers demonstrate that Aegis and low-level multiplexing can be implemented efficiently. We describe in detail how Aegis multiplexes the processor, dispatches exceptions, translates addresses, transfers control between address spaces, and multiplexes the network.

5.1 Aegis Overview

Table 2 lists a subset of the Aegis interface. We discuss the implementation of most of the system calls in this section. Aegis also supports a set of *primitive operations* that encapsulate privileged instructions and are guaranteed not to alter application-visible registers (see Table 3 for some typical examples). These primitive operations can be viewed as pseudo-instructions (similar to the Alpha's use of PALcode [45]). In this subsection we examine how Aegis protects time slices and processor environments; other resources are protected as described in Section 3.

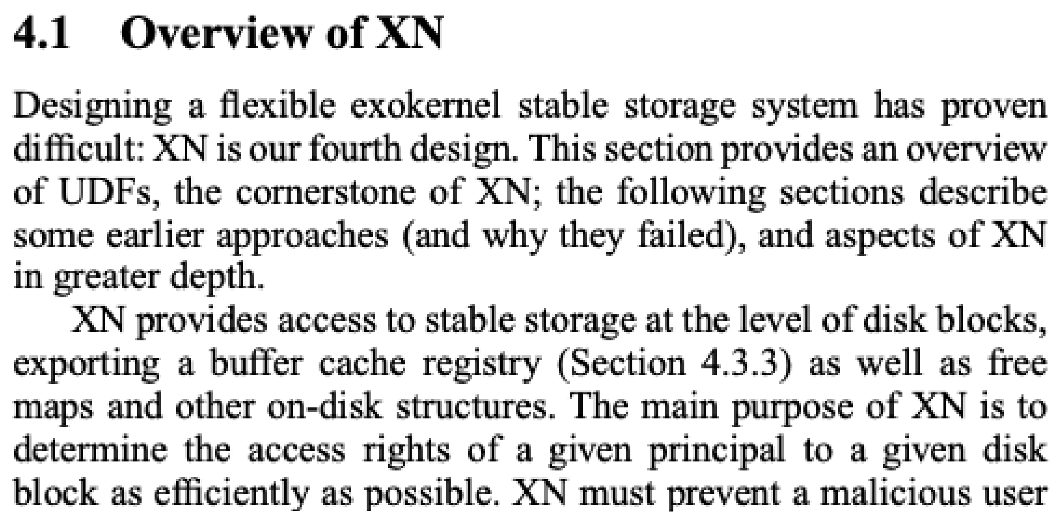
Self Study - 2

- Read XN (disk subsystem) and Xok (An exokernel for Intel x86) design details


To appear in the Proceedings of the 16th Symposium on Operating Systems Principles (SOSP), October, 1997.

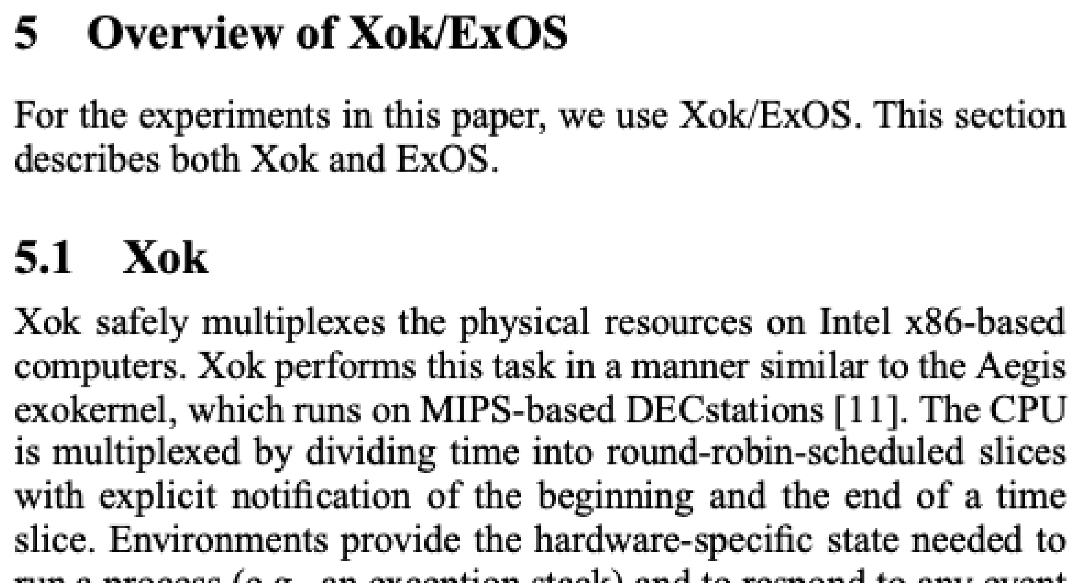
Application Performance and Flexibility on Exokernel Systems

M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger,
Héctor M. Briceño, Russell Hunt, David Mazières, Thomas Pinckney,
Robert Grimm, John Jannotti, and Kenneth Mackenzie
M.I.T. Laboratory for Computer Science
Cambridge, MA 02139, U.S.A
`http://www.pdos.lcs.mit.edu/`


4.1 Overview of XN

Designing a flexible exokernel stable storage system has proven difficult: XN is our fourth design. This section provides an overview of UDFs, the cornerstone of XN; the following sections describe some earlier approaches (and why they failed), and aspects of XN in greater depth.

XN provides access to stable storage at the level of disk blocks, exporting a buffer cache registry (Section 4.3.3) as well as free maps and other on-disk structures. The main purpose of XN is to determine the access rights of a given principal to a given disk block as efficiently as possible. XN must prevent a malicious user


5 Overview of Xok/ExOS

For the experiments in this paper, we use Xok/ExOS. This section describes both Xok and ExOS.

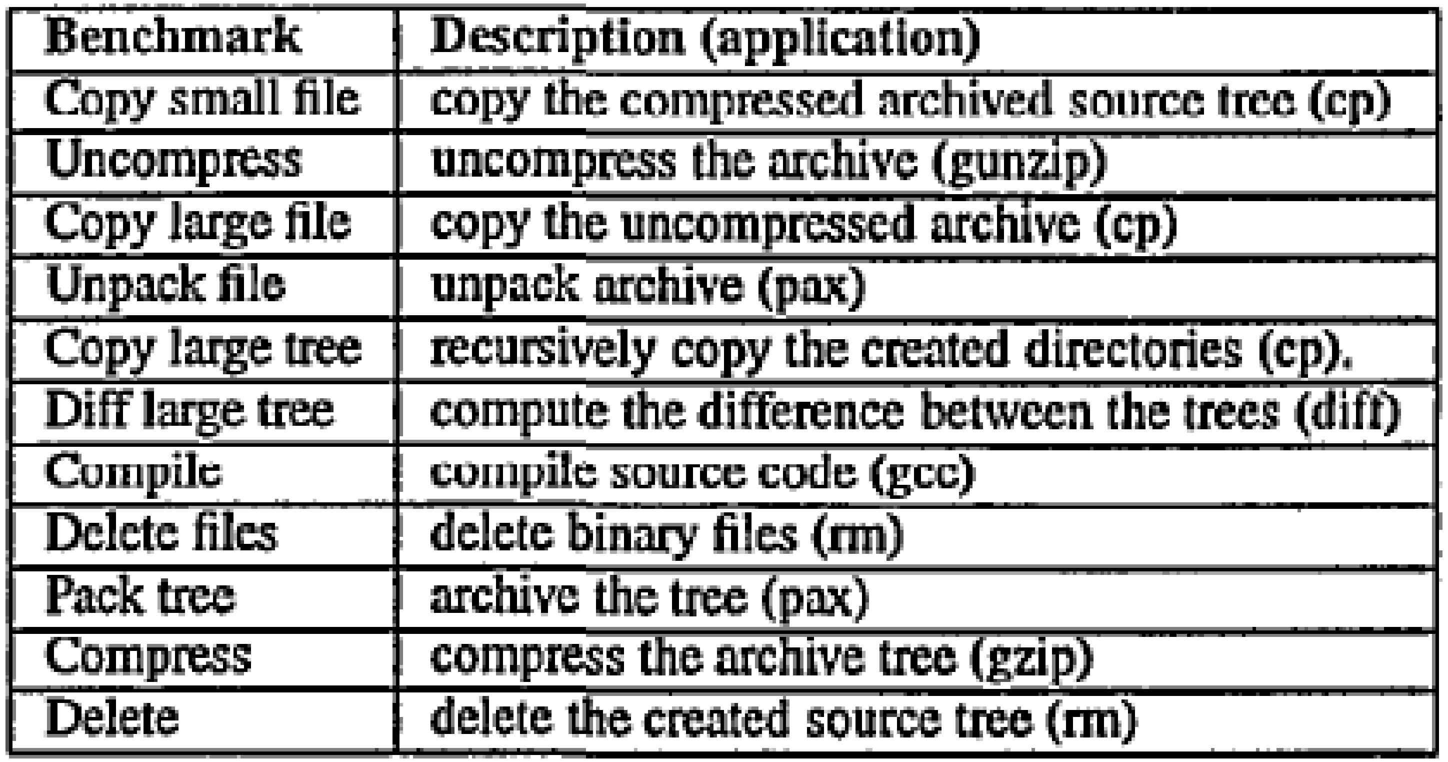
5.1 Xok

Xok safely multiplexes the physical resources on Intel x86-based computers. Xok performs this task in a manner similar to the Aegis exokernel, which runs on MIPS-based DECstations [11]. The CPU is multiplexed by dividing time into round-robin-scheduled slices with explicit notification of the beginning and the end of a time slice. Environments provide the hardware-specific state needed to

Comparison

- Evaluation of Exokernel is done by comparing end to end application performance on Xok and two widely used 4.4BSD UNIX Systems (FreeBSD and OpenBSD)
- Berkeley Software Distribution (BSD) is a UNIX operating system developed by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, from 1977 to 1995.
- FreeBSD and OpenBSD are operating systems descended from BSD UNIX.
- FreeBSD for desktop users
- OpenBSD is mostly for servers

Benchmarks



Benchmark	Description (application)
Copy small file	copy the compressed archived source tree (cp)
Uncompress	uncompress the archive (gunzip)
Copy large file	copy the uncompressed archive (cp)
Unpack file	unpack archive (pax)
Copy large tree	recursively copy the created directories (cp).
Diff large tree	compute the difference between the trees (diff)
Compile	compile source code (gcc)
Delete files	delete binary files (rm)
Pack tree	archive the tree (pax)
Compress	compress the archive tree (gzip)
Delete	delete the created source tree (rm)

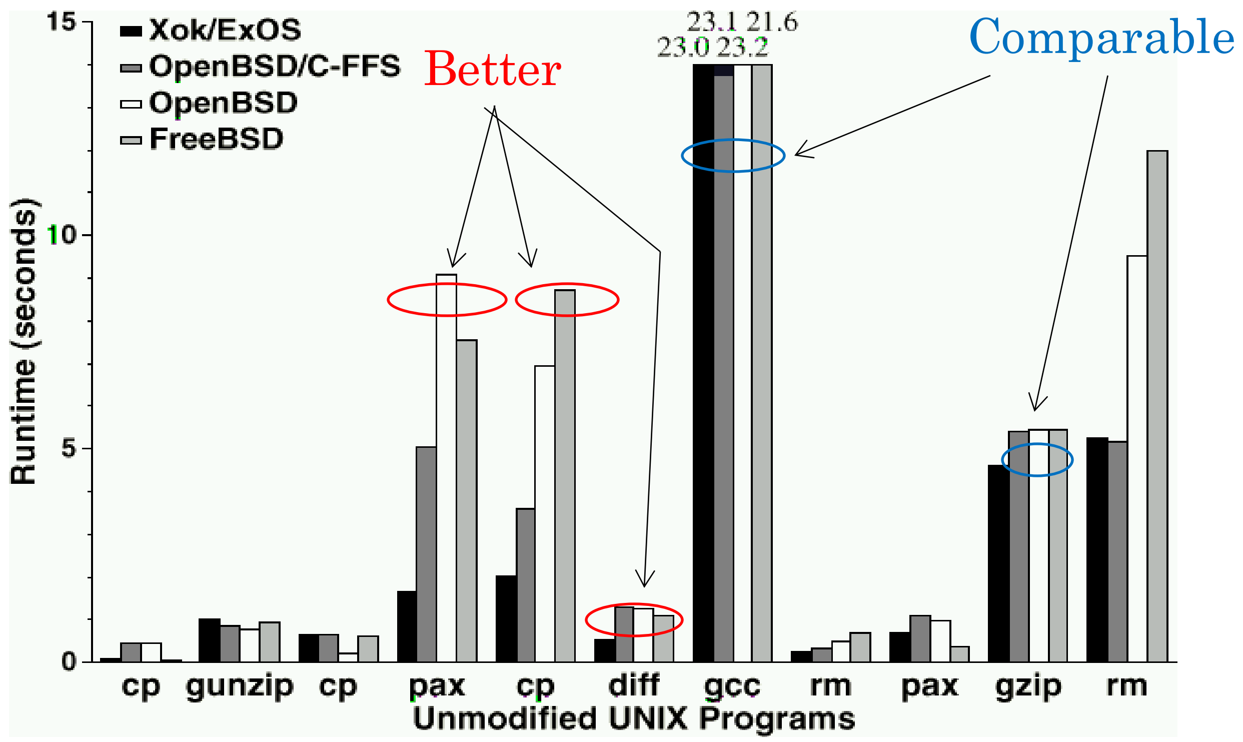
Table 1: The I/O-intensive workload installs a large application (the lcc compiler). The size of the compressed archive file for lcc is 1.1 MByte.

Performance Evaluation

- Unmodified UNIX applications:
  - 200-MHz Intel Pentium Pro, 64MB of memory
  - Applications either perform comparably on Xok/ExOS and the BSD UNIXes, or perform significantly better at a speed of 4x
  - Performace of 8 of 11 applications are comparable to BSD Unixes.
  - On 3 applications (pax, cp, diff) Xok/ExOS runs considerably faster.

Performance Evaluation

- Unmodified UNIX applications:
  - Xok/ExOS is the first bar

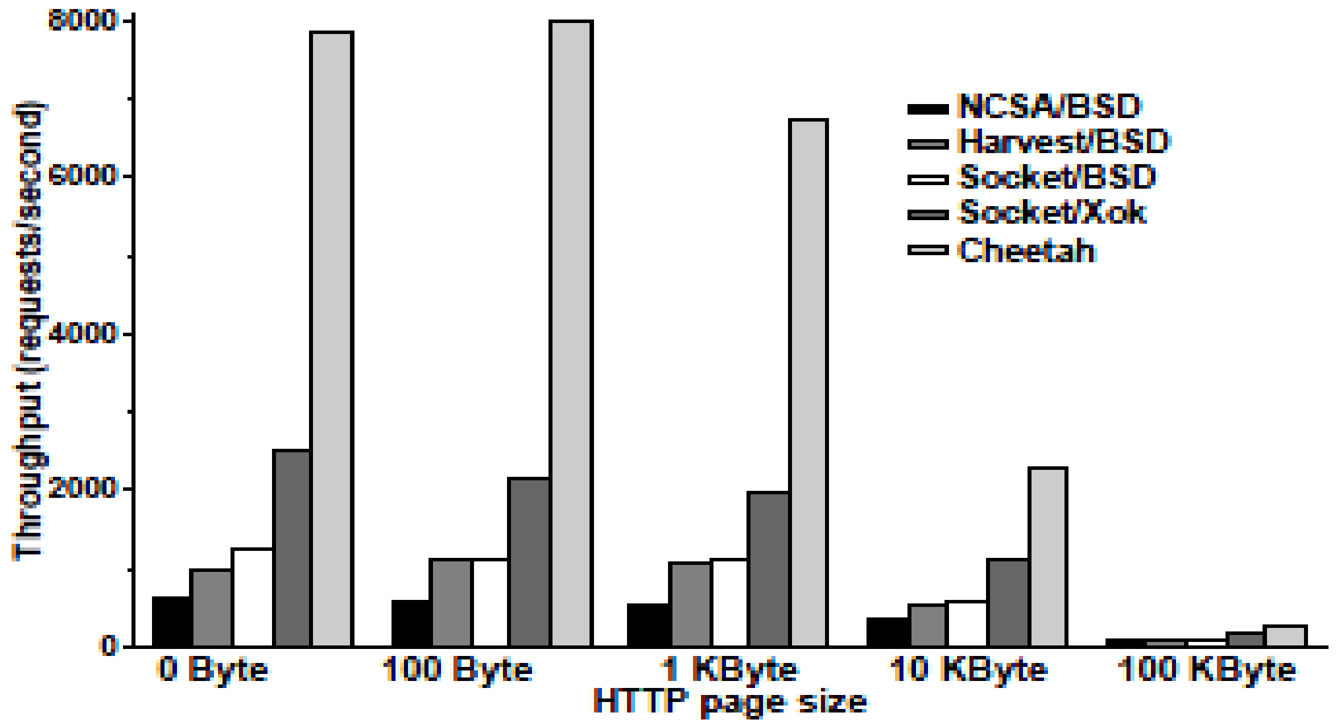


Cheetah HTTP Server: Modified Application

- Given a client request, HTTP server finds the appropriate document and sends it.
- Cheetah uses a file system and a TCP implementation customized for the properties of HTTP traffic.
- Cheetah performs up to eight (8) times faster than the best UNIX HTTP server we measured on the same hardware.
- Exokernel is well suited to building fast servers

Cheetah HTTP Server : Modified Application

- Cheetah is the last bar



Exokernel: Benefits

- Exposing kernel data structure
  - Can be accessed without system call overhead
- Flexibility
  - libOSes can be modified and debugged considerable more easily then kernels
  - "Edit, compile, debug" cycle of applications is considerably faster than the "edit, compile, reboot, debug" cycle of kernel.
- Performance
- Aggressive applications may gain speed up to 10x

Exokernel: Drawbacks

- Exokernel interface design is not simple
  - Most of the major exokernel interfaces have gone through multiple designs over several years
- The ease of creation and mixing of libOSes could lead to code messes
  - nightmare for maintenance coders and system administrators
- It is theoretically possible to provide libOSes that enable applications to run simultaneously on the same system, that would also mean different look & feels for each of them.
- Different libOSes may have varying levels of compatibility and interoperability with each other.
- Poorly chosen abstractions may cause lose of information
- Self-paging libOSes
  - Self-paging is difficult

Conclusion

- Exokernel Architecture:
  - Goal: safe application control of all resources.
  - How: by separating resource management from protection.
- Results found promising:
  - Unmodified applications run same or 4x better.
  - Customized applications can run up to 8x better.
  - Global performance is similarly good like UNIX.

Discussion

- Do normal applications benefit?
- Will the exokernel work for multiprocessor systems ?
- Is this similar in any way to virtual machines?
- Would you use it?
  - When and why?
  - When not and why?

Happy Learning