# Unit 14: The Mach Operating System

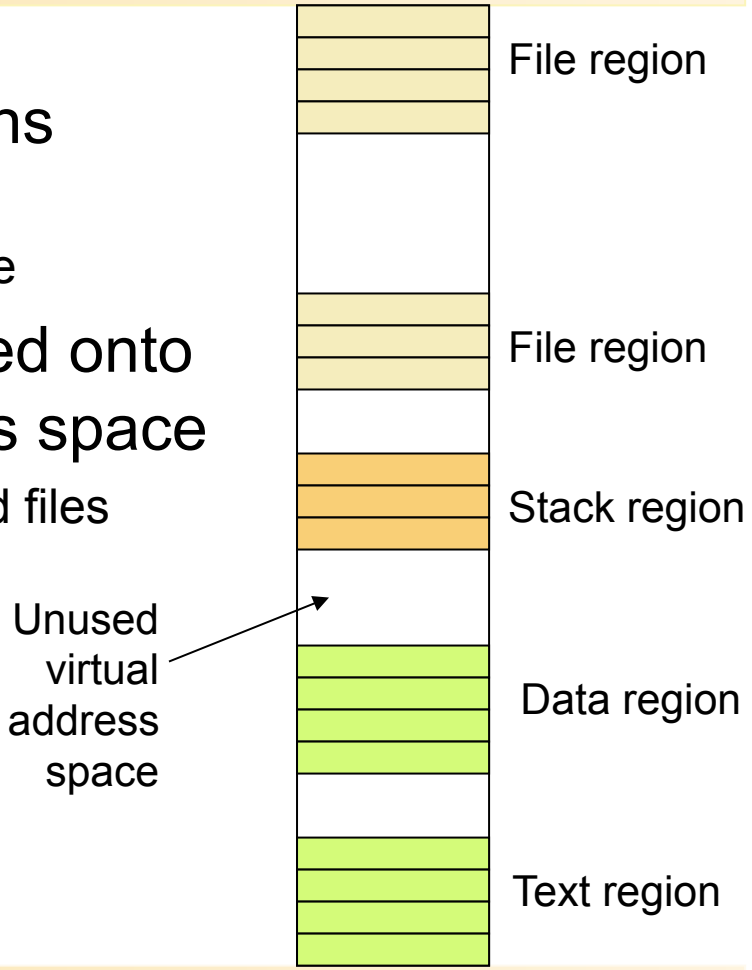
## 14.3. Mach Memory Management

# Mach Virtual Memory Management

- Each Mach task receives a 4-gigabyte virtual address space for its threads to execute in.
- A task can modify its address space in several ways. It can:
  - Allocate a region of virtual memory (on a page boundary).
  - Deallocate a region of virtual memory.
  - Set the protection status of a region of virtual memory.
  - Specify the inheritance of a region of virtual memory.
  - Create and manage a memory object that can then be mapped into the space of another task.
- Regions for virtual memory operations must be aligned on system page boundaries.
  - The size in bytes of a virtual memory page is contained in the vm_page_size variable.

# Virtual Memory

- Mach manages memory regions
  - sections of virtual address space
  - Identified by base address and a size
- Memory objects can be mapped onto unused portions of the address space
  - Page, set of pages, memory mapped files



# Inheritance and Protection of Memory

- With UNIX, creating a new process entails creating a copy of the parent's address space.
  - inefficient operation;
  - often child task touches only a portion of its copy of the parent's address space.
- Under Mach, the child task initially shares the parent's address space.
- Copying occurs only when needed, on a page-by-page basis.

# Inheritance of Memory (contd.)

- A task may specify that pages of its address space be inherited by child tasks in three ways:

**Copy:**

- Pages marked as copy are logically copied by value;
- For efficiency copy-on-write techniques are used.
- This is the default mode of inheritance if no mode is specified.

**Shared**:

- Pages specified as shared can be read from and written to by both the parent and child.

**None**

- Pages marked as none aren't passed to a child.
- The child's corresponding address is left unallocated.

# Paging Objects

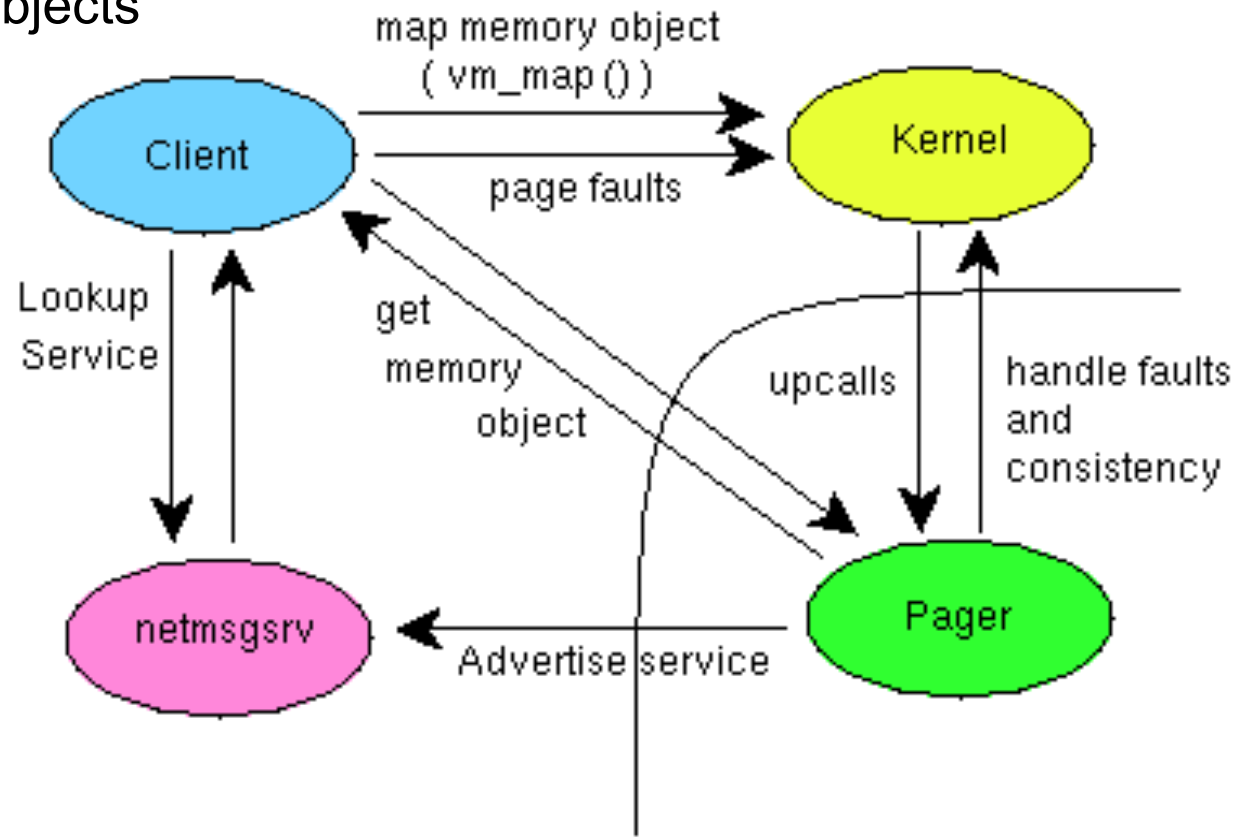
- Paging Objects
  - secondary storage object that's mapped into a task's virtual memory.
  - Paging objects are commonly files managed by a file server;
  - may be implemented by any port that can handle requests to read and write data.
- Physical pages in an address space have paging objects associated with them.
  - identify the backing storage to be used when a page is to be read in or written.

# Operation of Mach VM

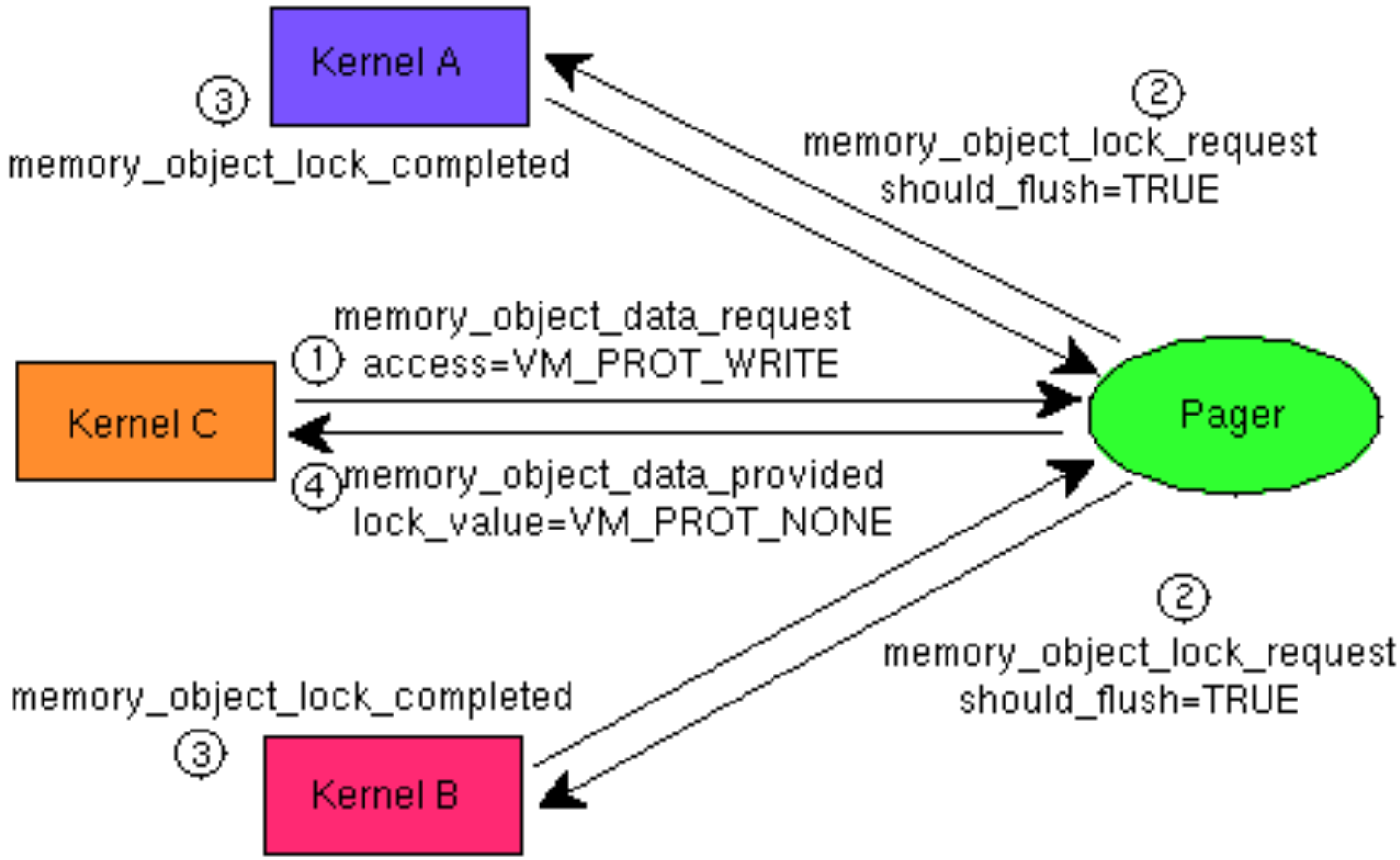
- Code is split into three parts
  - Pmap module runs in kernel and deals with MMU (Memory Management Unit)
  - Machine-independent kernel code
  - External pager (user-space memory manager)
- Pager manages backing store (disk)
  - Kernel and memory manager communicate via well-defined protocol
  - Users may write their own memory managers
  - Pagers are not required to use secondary storage at all: instead of paging onto a disk they may send memory pages to remote machines across a network
  - This allows for transparent implementation of distributed shared memory

# External Pagers

An external pager provides access to secondary storage through memory objects



# Shared Memory based on External Pagers



# Virtual Memory Functions

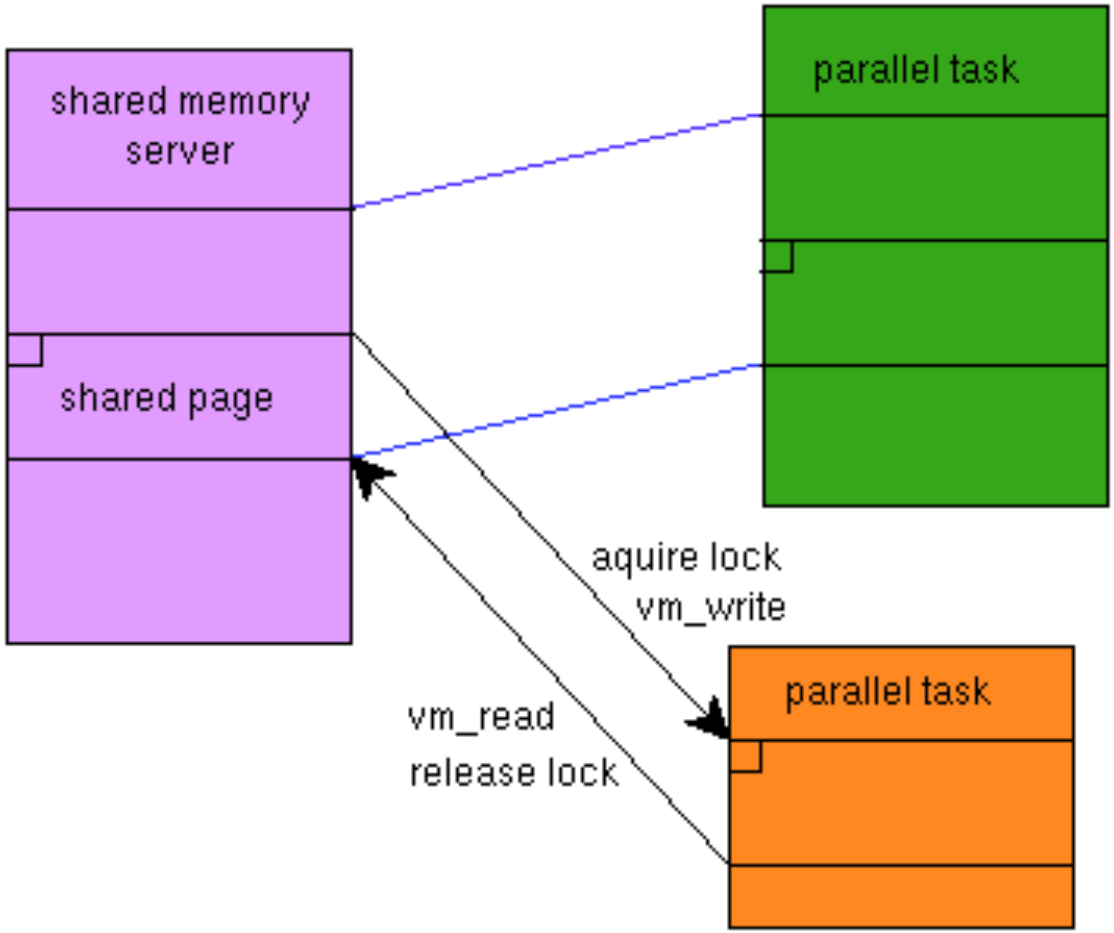
- vm_allocate() to get new virtual memory
- vm_deallocate() to free virtual memory

- The UNIX functions malloc(), calloc(), and free(), use vm_allocate() and vm_deallocate().
- Memory may appear in a task's address space as the result of a msg_receive() operation.

# VM Functions (contd.)

- malloc() and calloc() are library subroutine calls;
- vm_allocate() is a Mach kernel function, which is somewhat more expensive.
- If memory has been allocated with vm_allocate(),
  - it must be deallocated with vm_deallocate();
- if it was allocated with malloc()
  - it must be deallocated with free().
- Memory that's received out-of-line from a message has been allocated by the kernel with vm_allocate().
- vm_copy(), vm_read(), vm_write() copy memory pages between tasks.

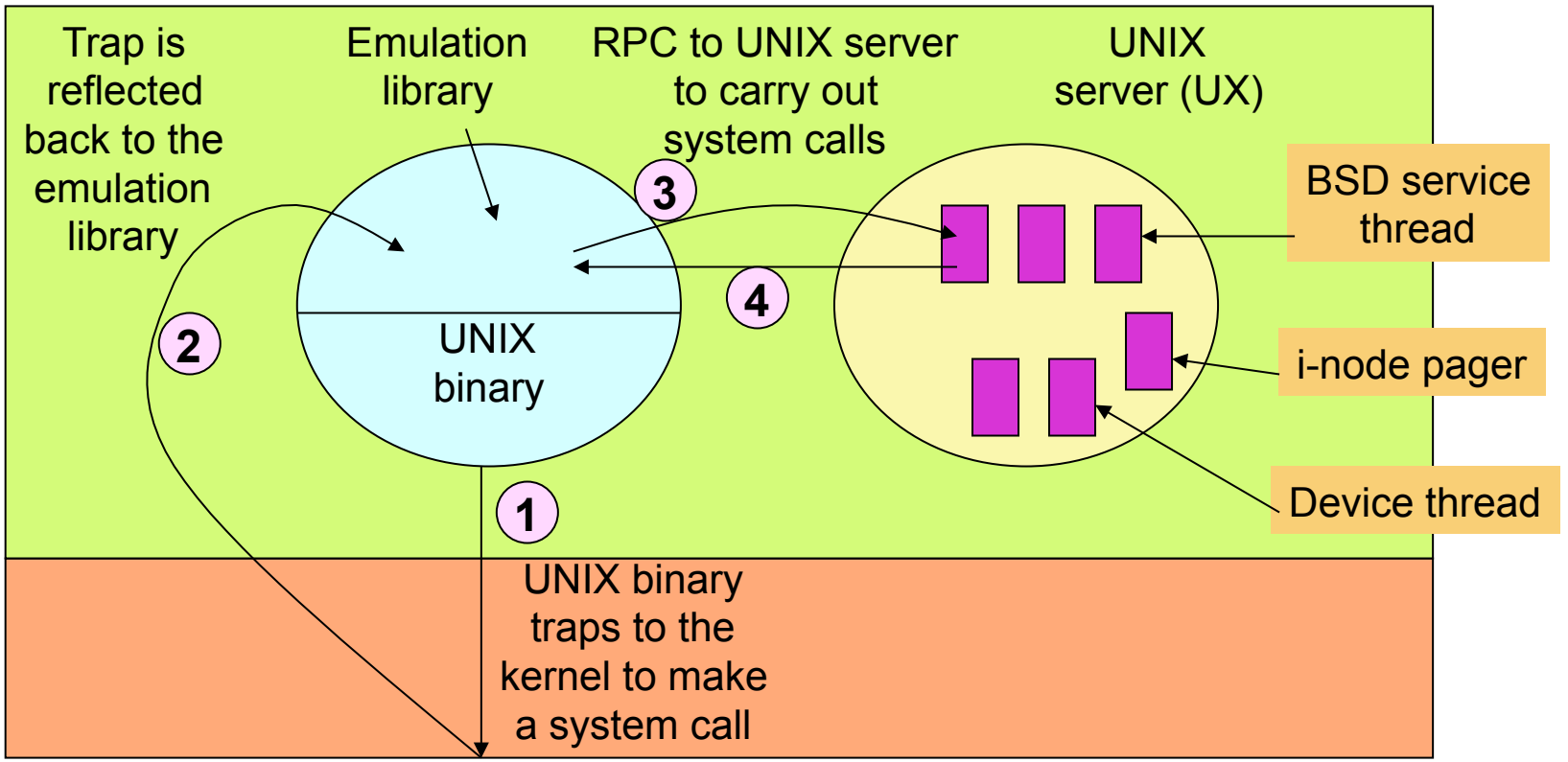
# Operation of vm_read/vm_write



# UNIX Emulation in Mach

- Mach has various servers that run on top of it
  - UNIX server contains large amount of Berkely UNIX code
  - Essentially the entire file system code is contained in UNIX server
- UNIX server (UX) operation:
  - Server and emulation library interact
  - At system start, UX instructs kernel to redirect system call traps to emulation library
  - Emulation library inspects processor registers to determine which system call was invoked
  - It calls UX server via Mach IPC (RPC)
  - On return, control is given directly to the caller (user program)
  - fork()/exec() have been modified so that the emulation library is attached to every newly created task

# UNIX Emulation (contd.)



# UNIX Server Implementation

- Implemented as a collection of C-threads
- Most threads handle BSD system calls
  - Emulation library communicates with server threads using Mach IPC
- When a message comes in...
  - An idle thread accepts it
  - Determines originator and extracts system call number and parameters
  - Executes the call and sends back the reply
- Most messages correspond exactly to one BSD system call

# Implementation of I/O Calls

- For performance reasons, I/O is implemented differently
- Files are mapped directly in caller‘s address space
  - Emulation library operates on mapped file
  - Page faults will occur when accessing the mapped file
- Each page fault requires interaction with external pager
  - i-node pager thread inside UX server operates as external pager
  - It accesses the disk and arranges for it to be mapped into the application program‘s address space
- i-node pager thread synchronizes operations on files opened by several UNIX tasks simultaneously

# Additional Reading

- J. Boykin, D. Kirschen, A. Langerman, S. LoVerso, „*Programming under Mach*“, Addison-Wesley, 1993.
- A.S. Tanenbaum, „*Distributed Operating Systems*“, Prentice Hall, 1995.
- David. L. Black. „*Scheduling Support for Concurrency and Parallelism in the Mach Operating System“* CMU Technical Report CMU-CS-90-125, also May 1990 IEEE Computer.
- David Golub, Randall Dean, Alessandro Forin, Richard Rashid. „*Unix as an Application Program“,* Proceedings of the USENIX Summer Conference, June 1990.
- www-2.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/mach.html