Systems Design and Implementation

I.7 –Memory Management

System Architecture Group, SS 2009

University of Karlsruhe

June 9, 2009

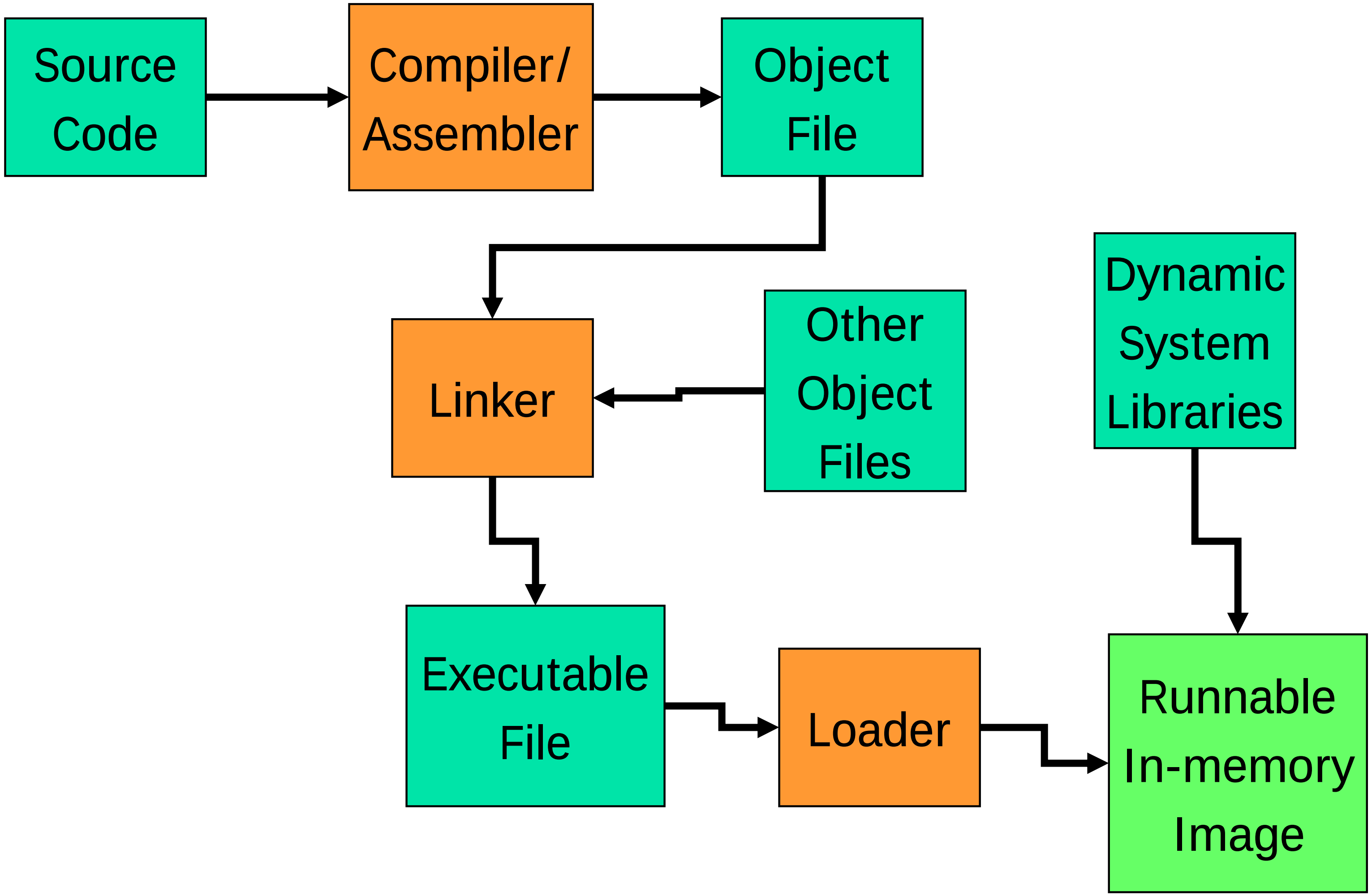
Jan Stoess

University of Karlsruhe

Overview

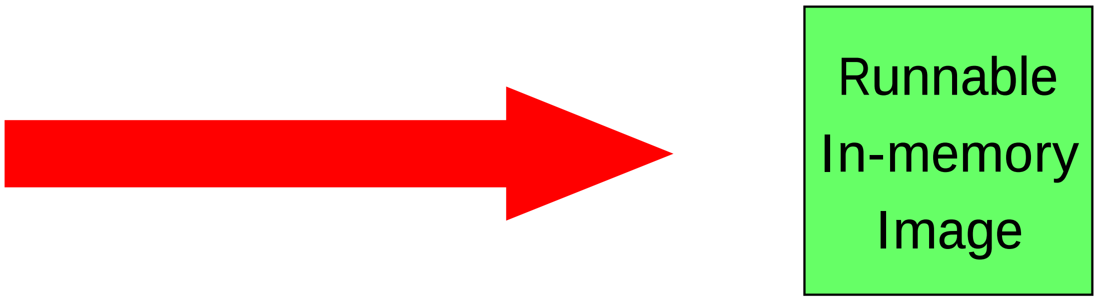
- Loading a program into memory
- Case Studies
  - 4.3 BSD Virtual Memory on VAX-11
  - SVR4 VM Architecture
  - Sawmill Dataspaces

The Road to a Running Program



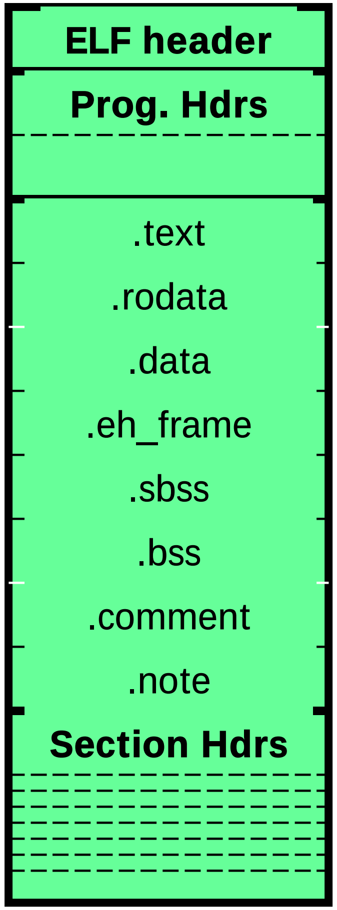
How do we get there?

- What do we load?
- Where do we load it?

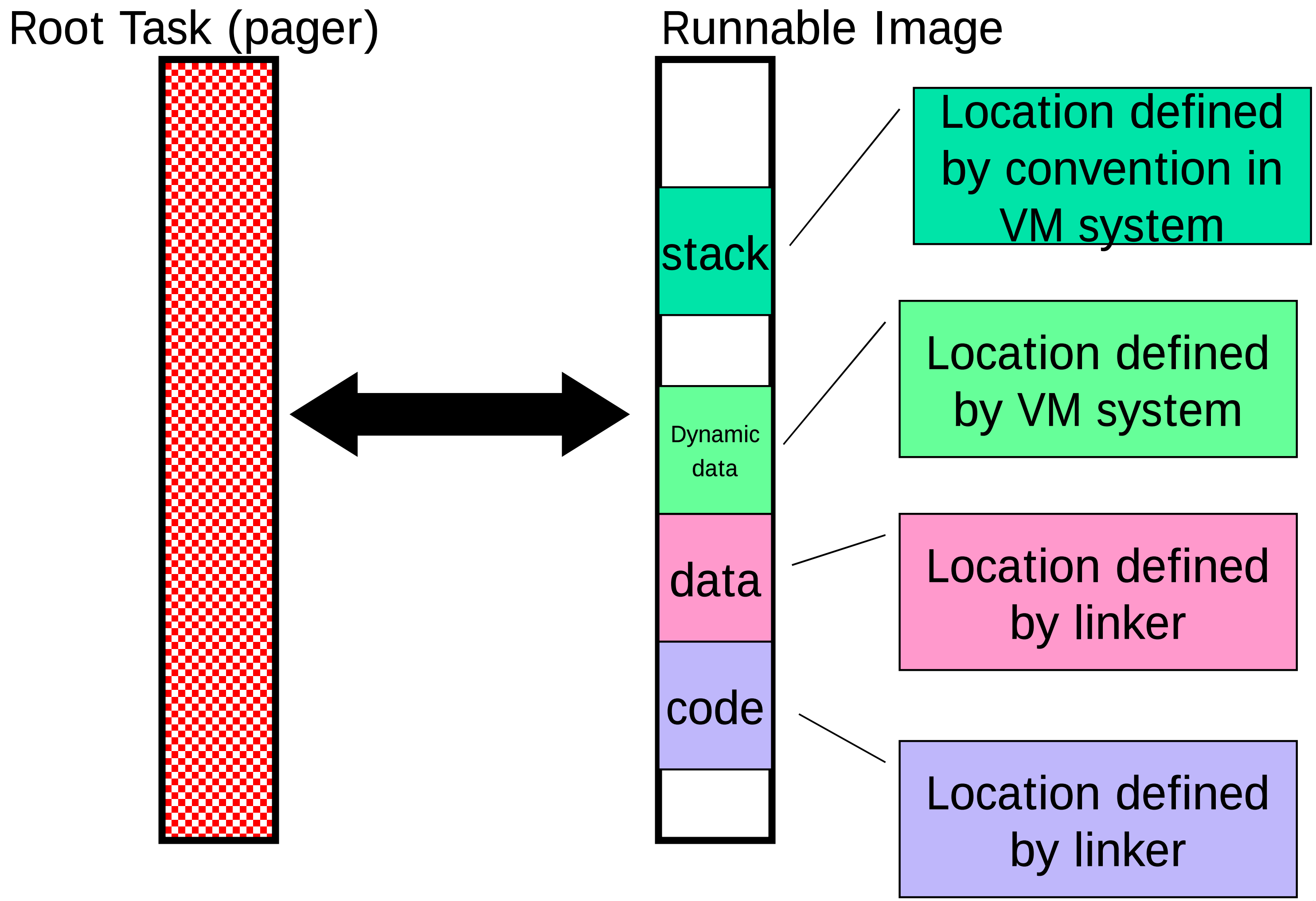


What do we load ?

- What we load is partially defined by ELF.
- **E**xecutable and **L**inkable **F**ormat
- Four major parts in ELF file
  - ELF header – roadmap
  - Program headers describe segments directly related to program loading
  - Section headers describe contents of the file
  - The data itself

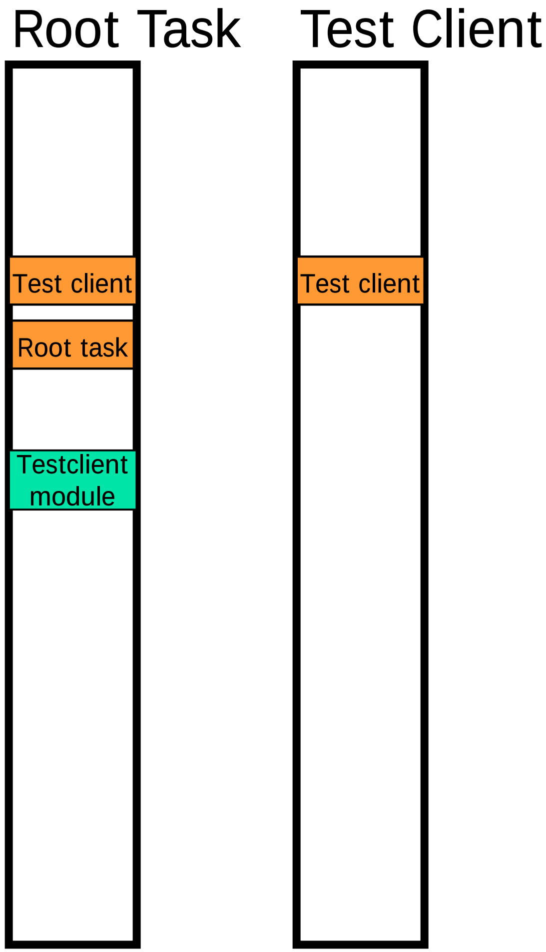


An Example


Root Task (pager)
Runnable Image
stack
Dynamic data
data
code
Location defined by convention in VM system
Location defined by VM system
Location defined by linker
Location defined by linker

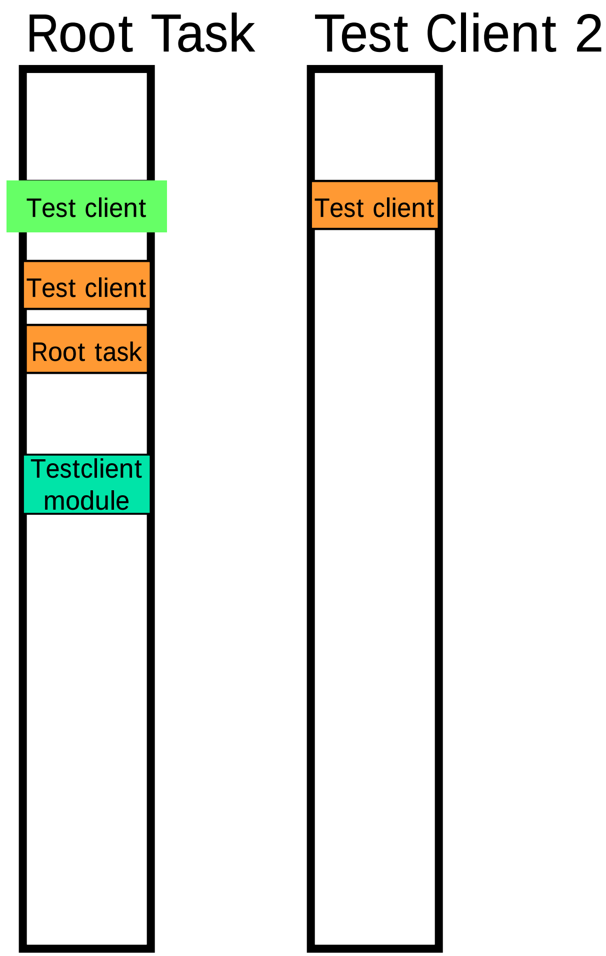
Load directly in physical memory?

- Pros
  - Simple one-to-one virtual-to-physical mapping
  - Easy
- Cons
  - Only one image per executable
  - Must link at the correct address
    - error prone and cumbersome
  - Fragmentation is a problem
  - Limited to physical memory size



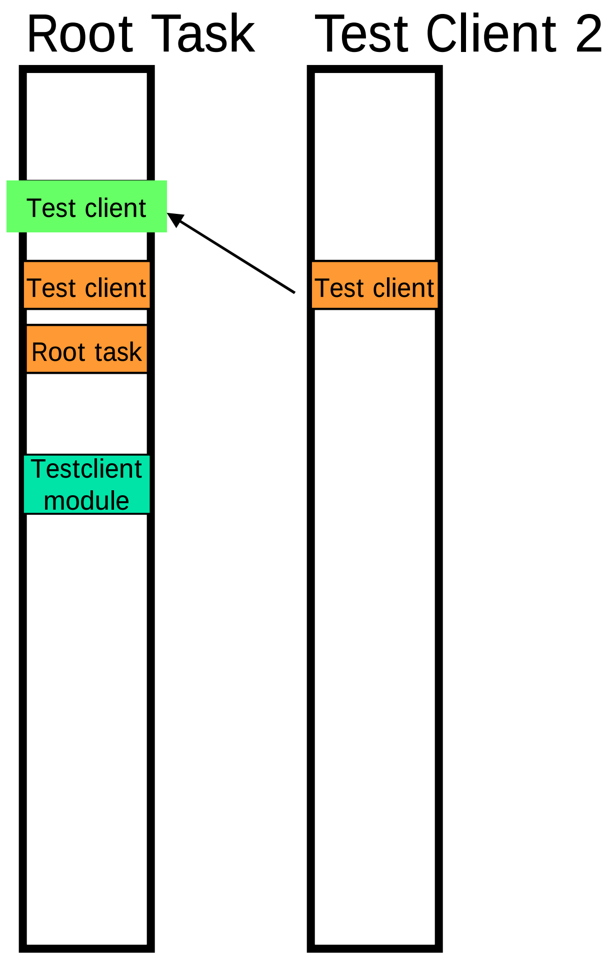
Dynamic relocation

- Pros
  - Simple one-to-one virtual-to-physical mapping
- Cons
  - PIC code has performance penalty, or relocation has a startup penalty
  - Fragmentation is still a problem
  - Still limited to physical memory size



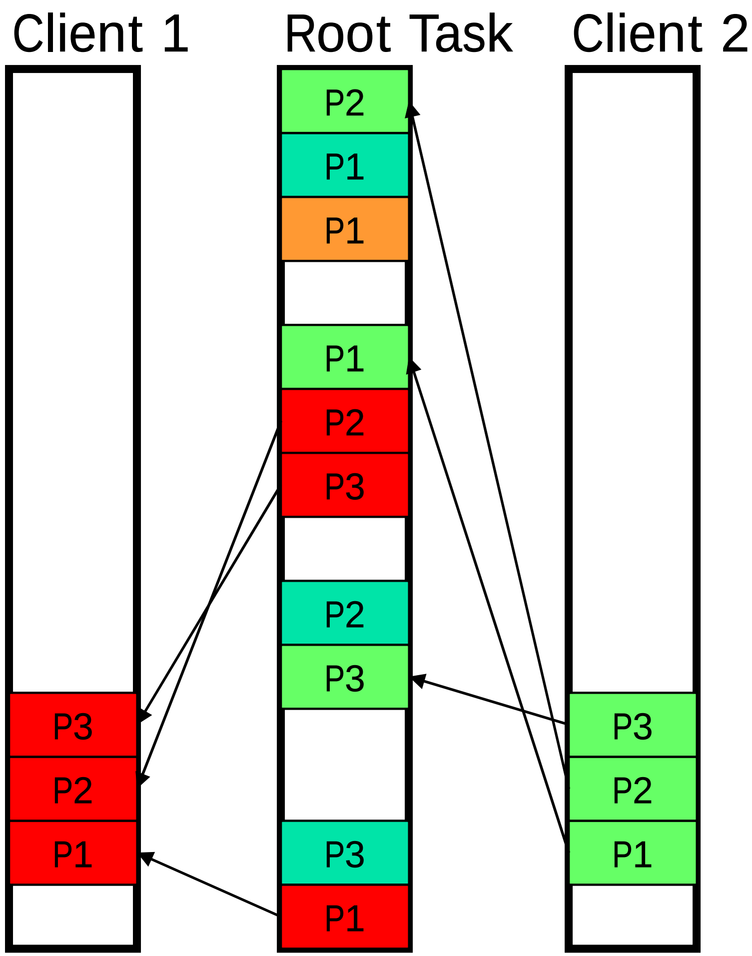
Address Translation / Segmentation

- Pros
  - Multiple instances of same executable
  - No a priori constructed load map
  - No relocation or PIC
  - Can use more than available physical memory
- Cons
  - Need a translation table (base,limit)
  - Fragmentation is still a problem (why?)
  - Swapping is coarse grained



Paged Virtual Memory

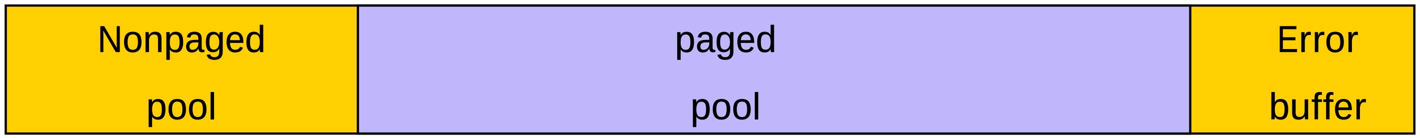
- Pros
  - Multiple instances of same executable
  - No a priori constructed load map
  - No relocation or PIC
  - Simple physical memory management
  - Fragmentation dramatically reduced (internal fragmentation only)
  - Can use more than available physical memory
- Cons
  - Need a page-based translation table and hardware
    - This is not free



Case Study
4.3 BSD Virtual Memory on VAX-11

- Three major components
  1. Core map
     - Global frame table
  2. Page tables
     - Per Process translation table
  3. Swap maps
     - Per-process mapping table from virtual pages space to disk blocks in swap space

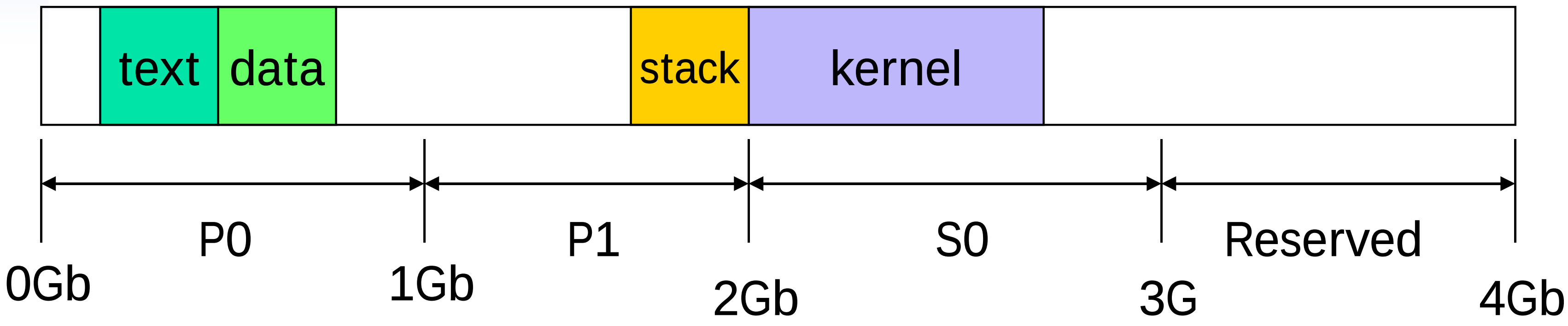
Core Map



Physical Memory

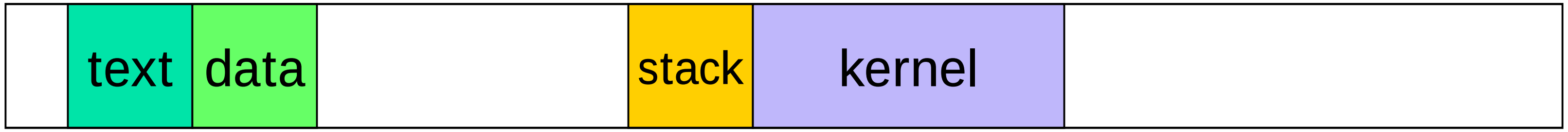
- Page pool described by an array of cmap entries
- Cmap entries contain
  - Name <type, owner, virtual page number>
  - Free list pointers <next, prev>
  - For text pages <device, block number>
  - Synchronization flags

Virtual Address Space



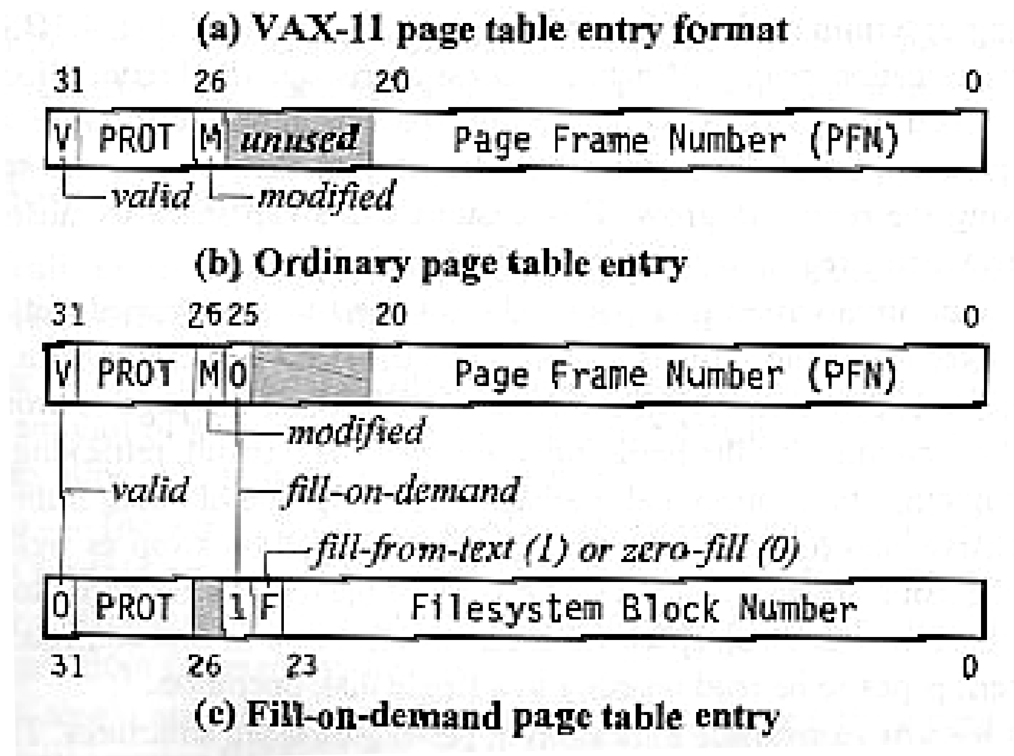
- Note: Data and stack areas are free to grow.
- The page tables describe the layout of the address space.

Page States

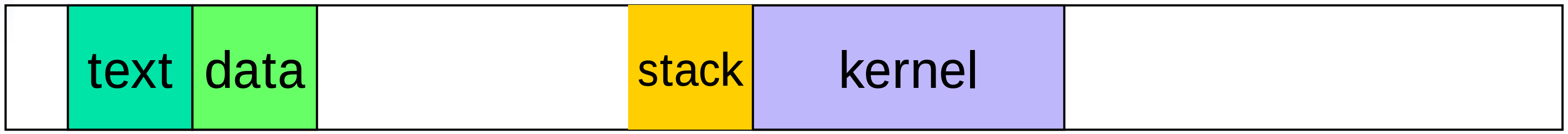


- Each individual page may be in one of the following states.
  - Resident
  - Fill-on-demand
    - Fill-from-text
    - Zero-fill
  - Swapped out

- State encoded in page table entries

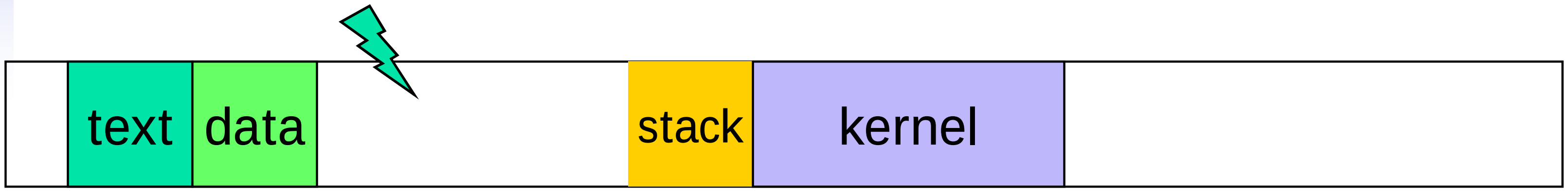


Process creation



- Allocate page tables representing the address space.
- Initialize all page table entries as either
  - Fill-from-text
  - Zero-fill

Page-fault handling



- Bounds error – access to inaccessible area.
  - Easy to detect, bounds error if
    - access to invalid page table entry, or
    - access to non-existent page table entry.
  - Except, automatic stack/heap growth.

Page fault handling

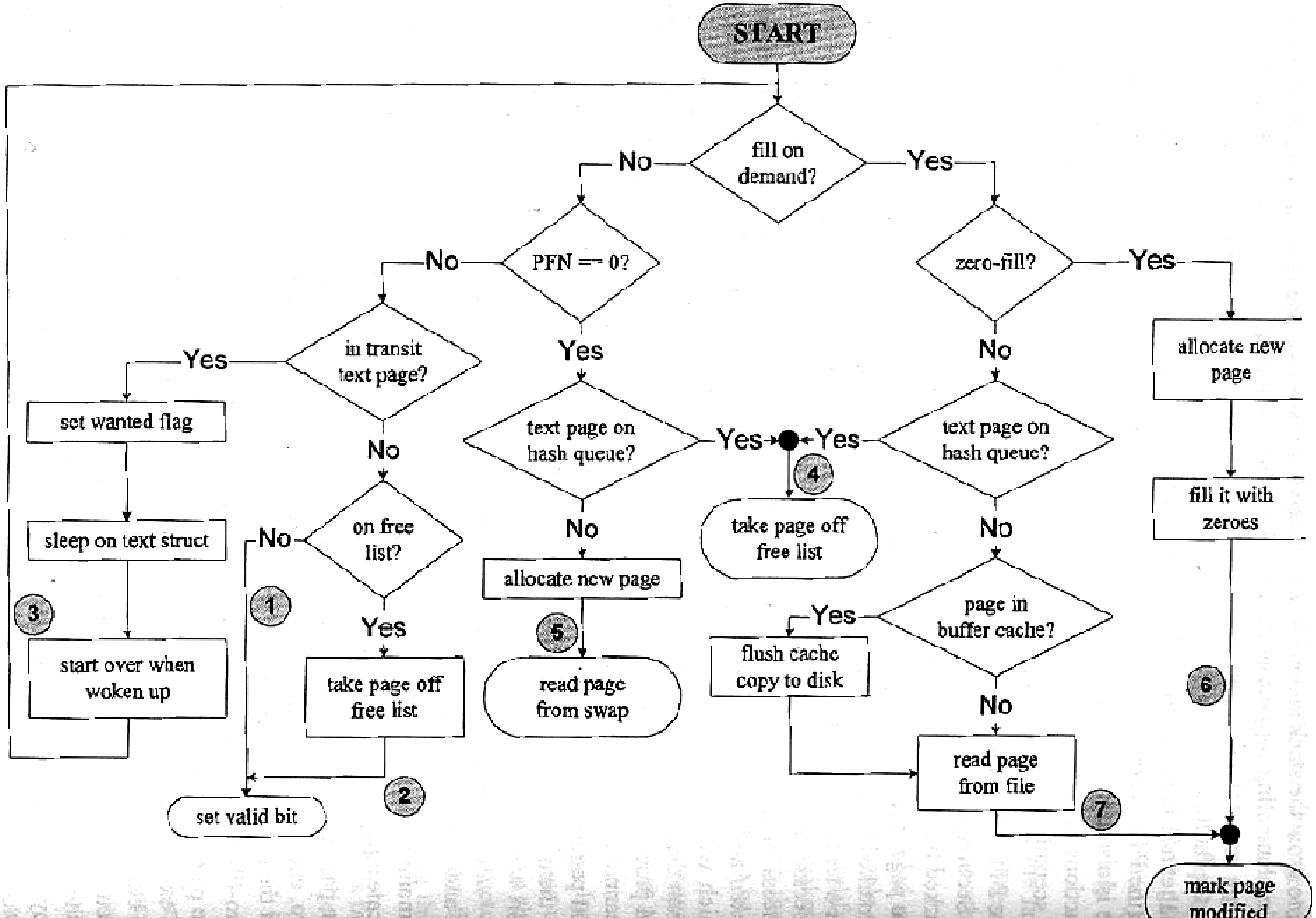


Figure 13-18. The pagein() algorithm.

Disadvantages

- No support for shared memory.
- No support for memory mapped files.
- No support for shared libraries.
- No copy-on-write support.
- VAX-specific architecture
  - VAX specific optimisations and structure
  - Not portable
- Code not modular – difficult to add features.

Memory mapped files

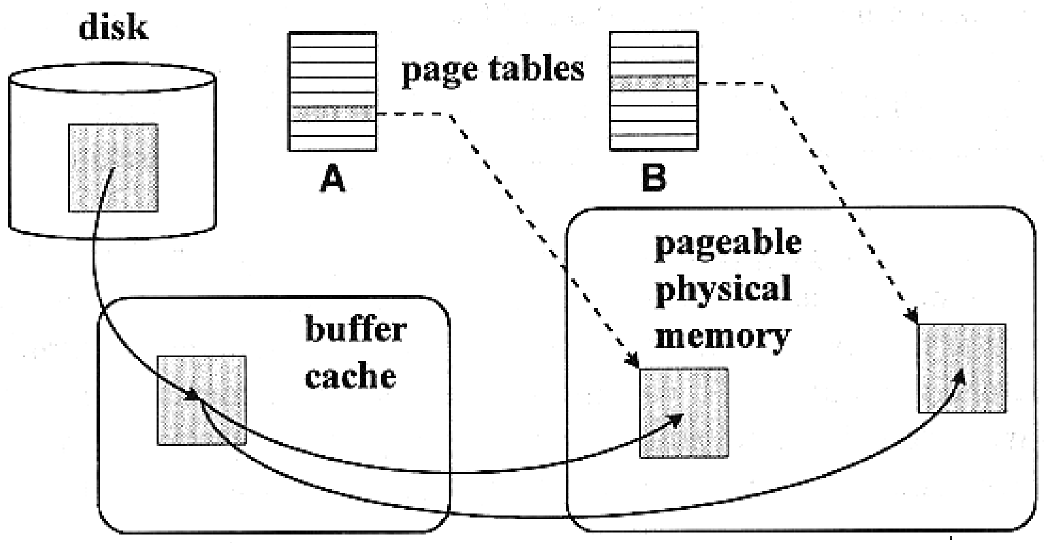
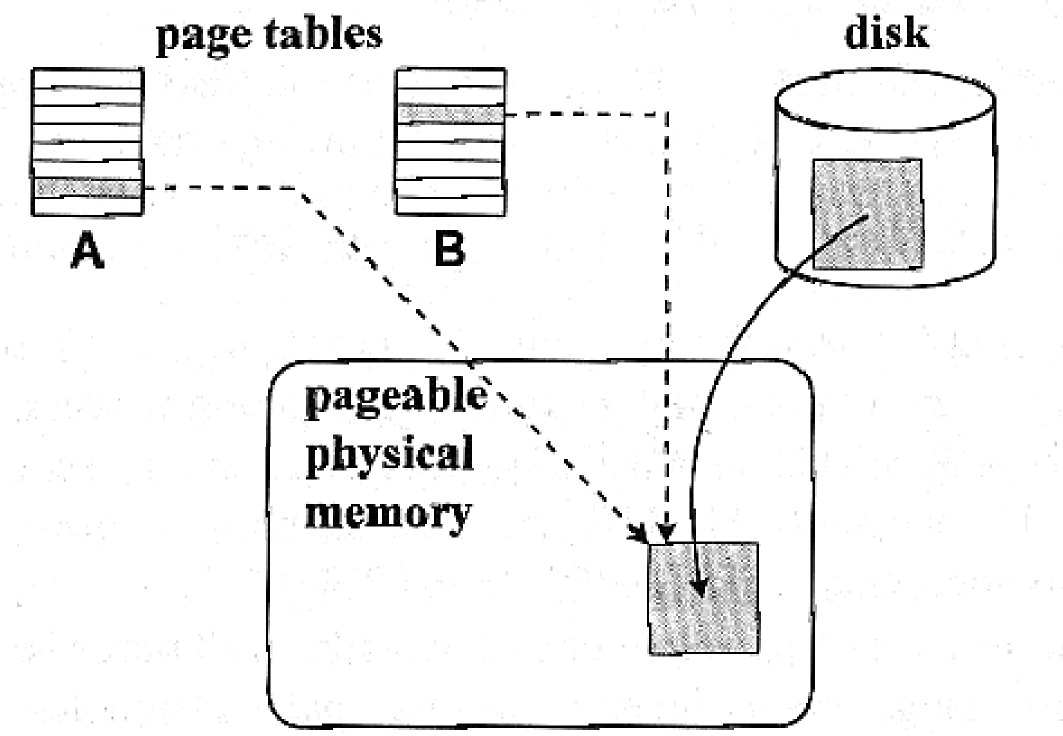


Figure 14-1. Two processes read the same page in traditional UNIX.

Memory mapped files



- Can be used as a mechanism to implement:
  - Shared memory
  - Shared libraries

Case Study
SVR4 VM Architecture

- The basic concepts are
  - Page – a frame of physical memory
  - Address space
  - Segment – a region in an address space
  - Hardware address translation – page tables
  - Anonymous page – page with no permanent storage

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Case Study
SVR4 VM Architecture

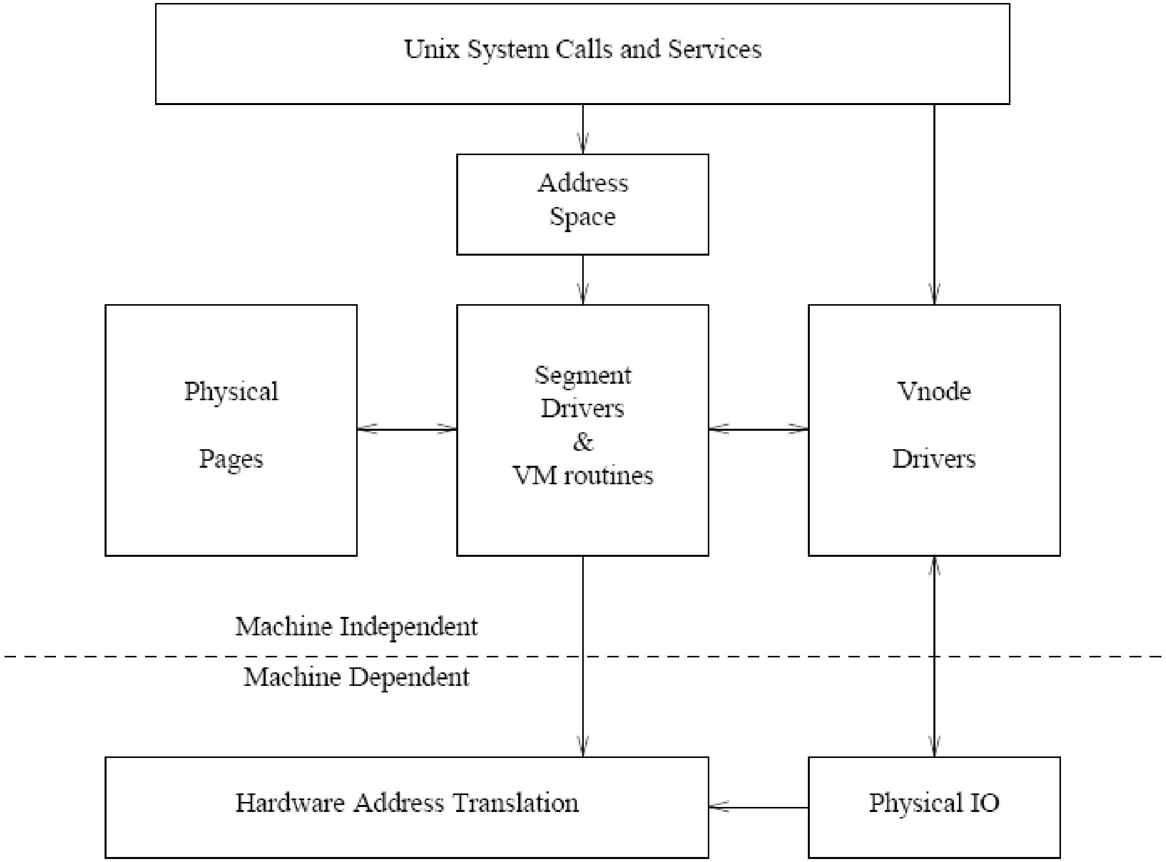


Figure 1

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Physical memory

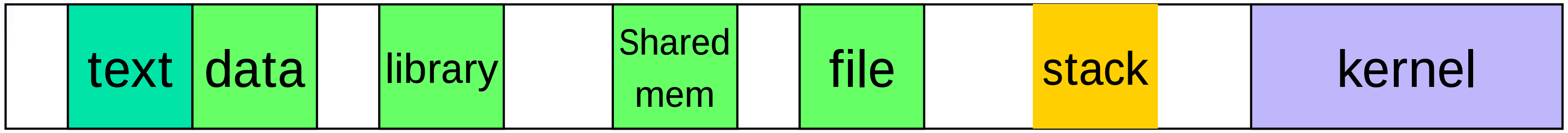
Nonpaged pool	paged pool	Error buffer

Physical Memory

- Page pool described by an array of page entries
- Page entries contain
  - Name <vnode, offset>
  - List pointers <next, prev> etc.
    - Free, I/O, hash chains, vnode
  - HAT info to locate all mappings
  - Synchronization flags

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Virtual Address Space



- Address spaces are composed of memory objects called segments.
  - Segments are a mapping between address space regions and backing-store objects (files, swap space, etc.)
- Operations on an address space
  - Alloc, free, dup, map, unmap, setprot, checkprot.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Anonymous memory

- Memory that is private to a process.
  - Not externally referable to, thus is "anonymous"
- Memory that has no permanent storage.
  - Contents are lost when process exits
- Paged by the kernel to swap space.
- Zero-filled.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Hardware Address Translation Layer

- Machine dependent – page tables.
- Operations
  - alloc, free.
  - memload, devload, unload.
  - pageunload, pagesync.
- Data in the HAT layer is redundant – it can be rebuilt from the machine independent layer.
- Interface is machine-independent.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Segments

- Each region has a segment driver associated with it
  - seg_vn – Mappings to regular files and anonymous object (vnodes).
  - seg_dev – Mappings to devices (frame buffers)

- Segment driver data
  - Current and max protection
  - Mapping type (shared or private)
  - Pointer to vnode
  - Offset to beginning of file

- Segment drivers support the following operations
  - create, dup, fault, setprot, checkprot, unmap, swap out, sync.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Creating a Mapping

- User calls mmap("file")
- Mmap checks permissions, existence, etc.
- Mmap calls as_map() to associate the file with a region in the address space.
- as_map() allocates segment data for the segment, and calls create() in the appropriate segment driver.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Creating a process

- Simplistically, involves creating mappings (segments):
  - Text -> appropriate region of executable file.
  - Data -> anonymous memory
  - Stack -> anonymous memory
- Shared libraries are mmaped by the client itself as needed.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Page fault handling outline

- Kernel calls as_fault().
  - Is the fault within a segment?
    - No -> bounds error!
    - Yes -> call the fault() routine of the associated segment driver.
      - The segment driver locates/allocates the data associated with the fault, and returns.

- All the complexity is in the segment specific drivers.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Advantages

- Design is modular
  - Easier to extend, modify
- Highly portable
  - Machine-dependent translation functionality in HAT layer
- Various types of memory sharing
  - Reduces physical memory usage
- Powerful Mmap
  - Supports file access, shared memory, and shared libraries
- Flexibility through segment drivers
  - Allows use of any object representable by vnode
    e.g. NFS files.

*J. Moran, "**SunOS Virtual Memory Implementation**", European UNIX Users Group (EUUG) Conference, Spring 1988*

Disadvantages

- Kernel consumes more memory
  - State associated with all the abstractions
- More complex and slower algorithms
- Modularity restricts flexibility
  - Framework may prevent machine-specific optimizations
- Copy-on-write not always faster than anticipatory copying

- However, in general the benefits of new functionality outweighs the performance penalties.

Implementing a Multi-Server OS with Dataspaces

Concept & Implementation

Motivation

Traditional OSes:

- Kernel-based VM
- Application-control impossible
- Extensibility limited

Multi-Server OSes:

- Kernel-based VM *management primitives* only
- VM management defined and implemented by user-level pagers

Observation: Address Space

Consists of regions

- Different semantics
- Different types
- Different resources

Involved Parties

- Tasks that need „data“
- Pager(s),
- Providers of „data“

We want…

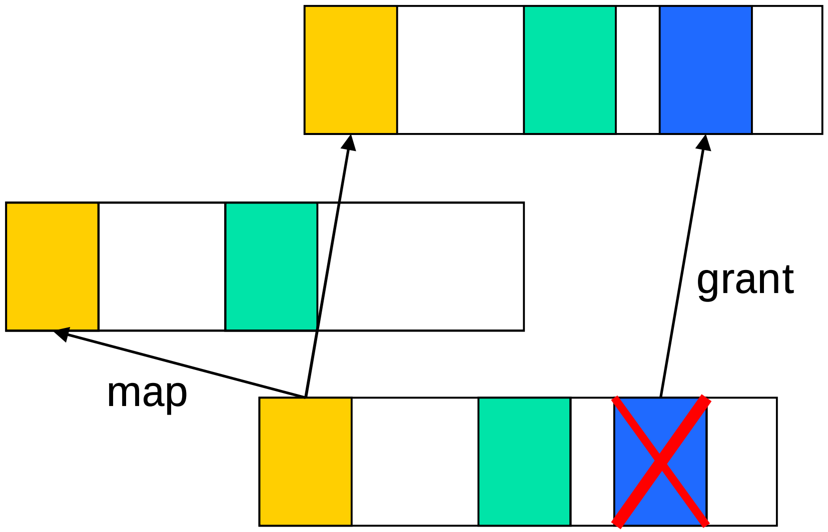
- Diversity
  - Customizable
  - Control over policies
- Dynamic extensibility
  - Code reuse
  - Easy implementation of policies
- Performance
  - Abstractions should not limit optimizations

The SawMill framework for VM diversity

- Framework to build and customize VMM
- Policy-free abstractions for VM management
- Decomposing of VMM into components
- Dynamic configurability

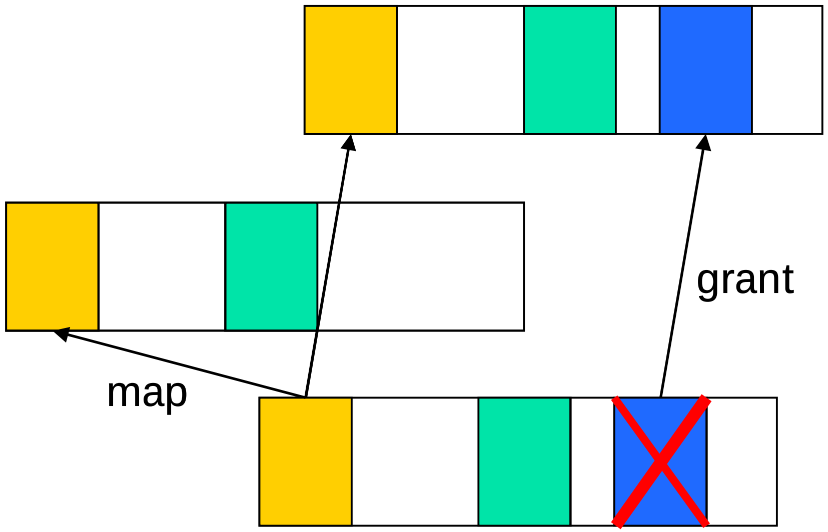
Microkernel Provides:

- Threads
- Address spaces
- IPC
- Hierarchical VM system
  - Mapping IPC
  - Grant IPC
- User-level-pager (per thread)



Microkernel Provides:

- Pagefault IPC
  - Source tid
  - Address
  - R/W
  - IP
  - Mapping



The Dataspace Concept

- A *dataspace* is
  - is memory concept
  - denotes an abstract data container
  - represents unstructured data
  - can be associated with files, memory, frame buffers, ...

Dataspace vs. Region

Dataspace

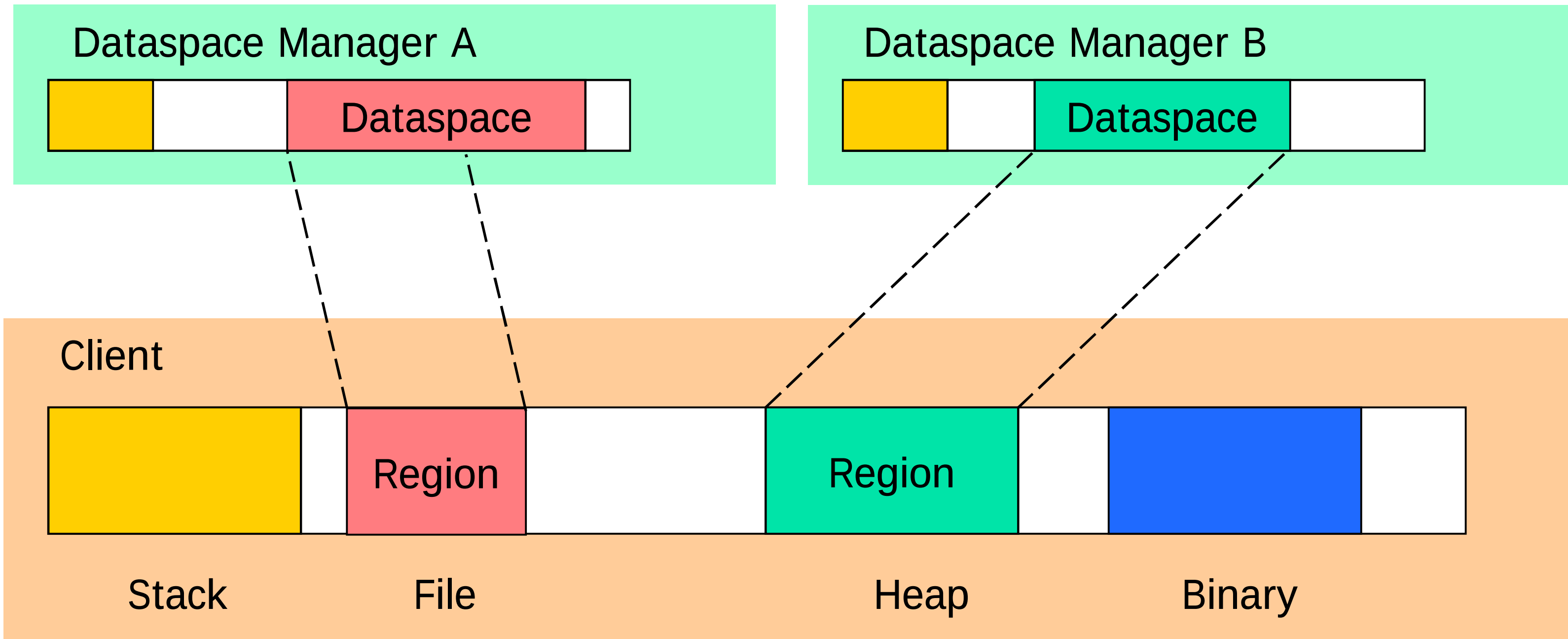
- Entity of a dataspace manager
- No size associated
- Logical object

Region

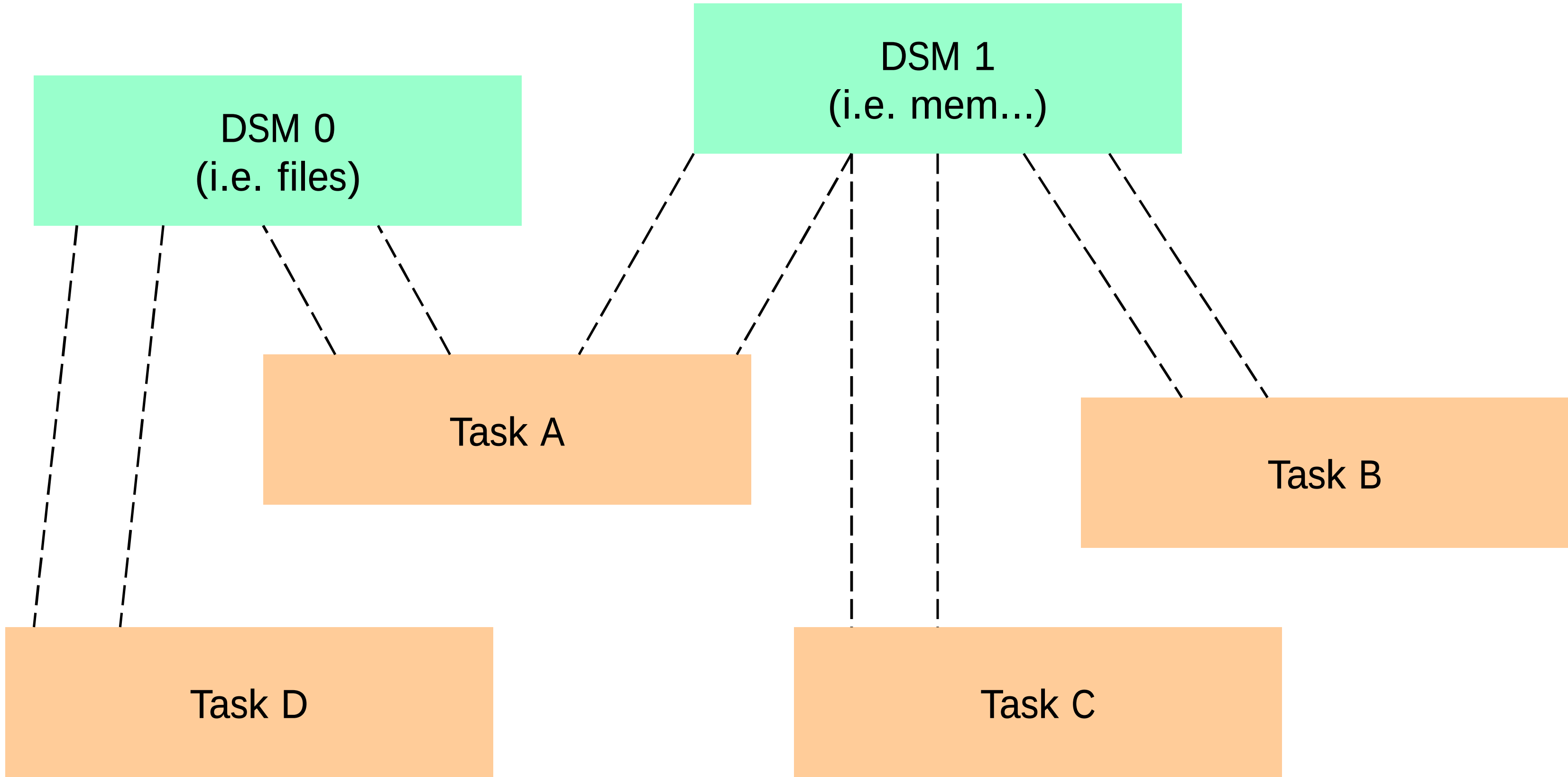
- In client‘s address space
  - fpage
- Has a size
- Makes part of a dataspace accessible

Dataspaces Server and Client

- accessing dataspaces

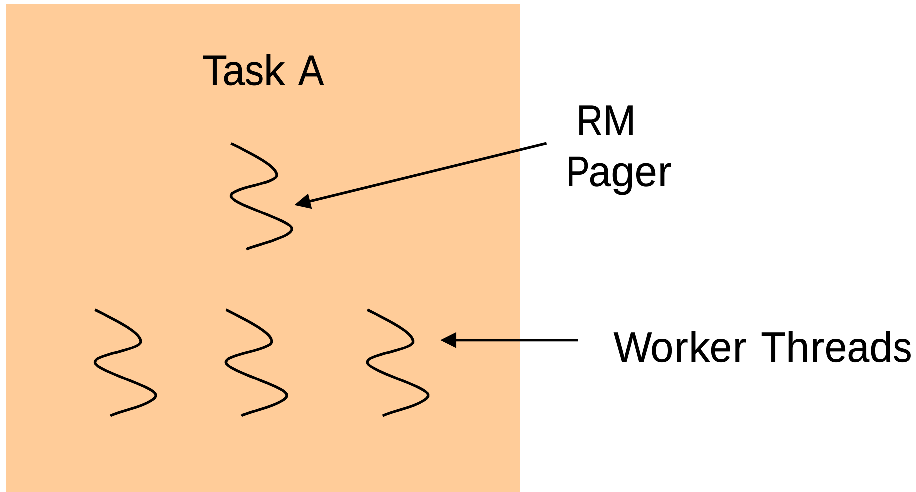


Multiple Dataspace Managers



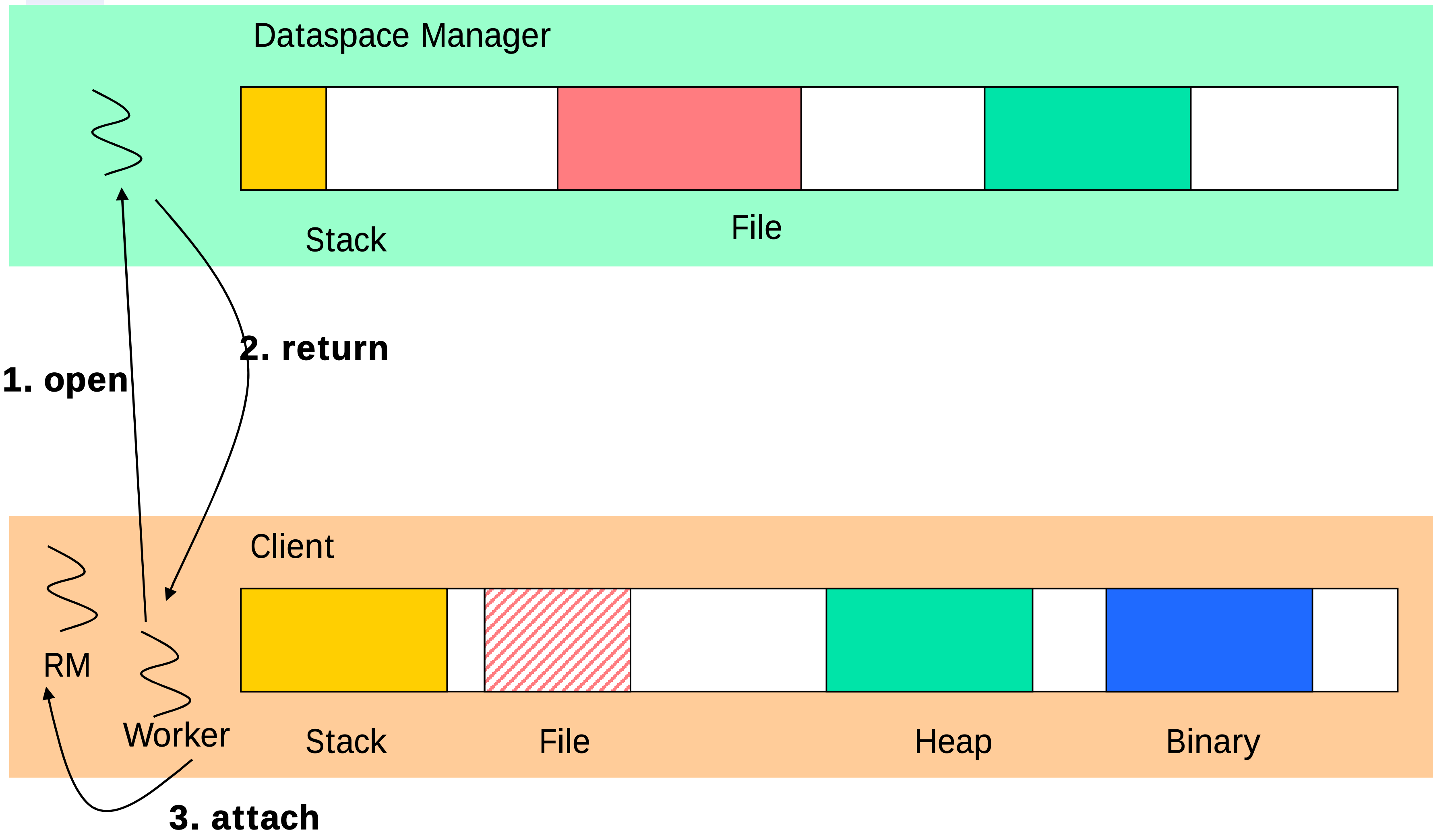
Region Mapper (RM)

- ☑ One user level pager per task
- ☑ Customizable
- ☑ Efficient



Attach Scenario

1. open
2. return dataspace
3. attach dataspace

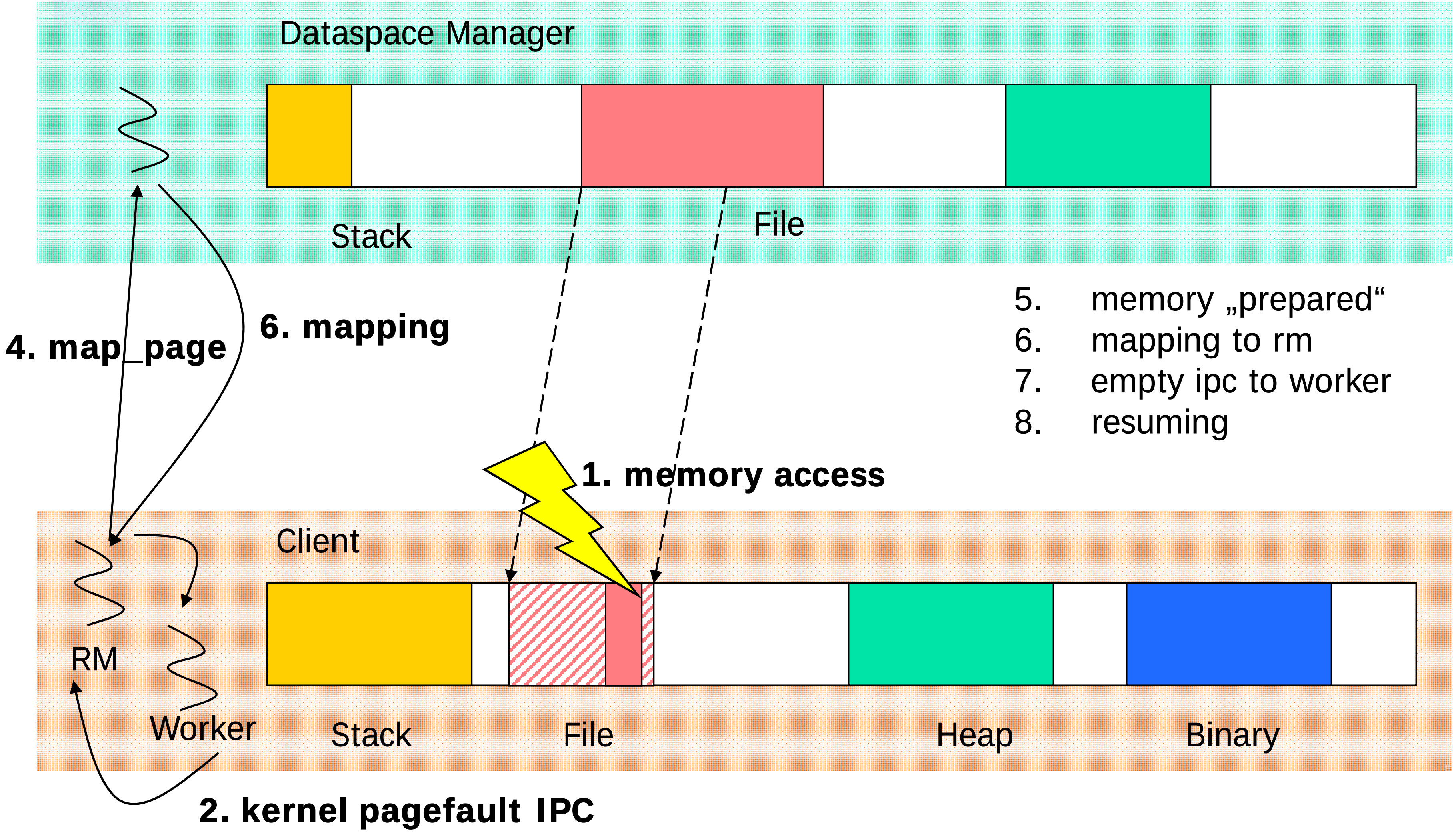


Involved Parties

- Client address space
- Pagefault handler (region mapper)
- Dataspace manager

Pagefault

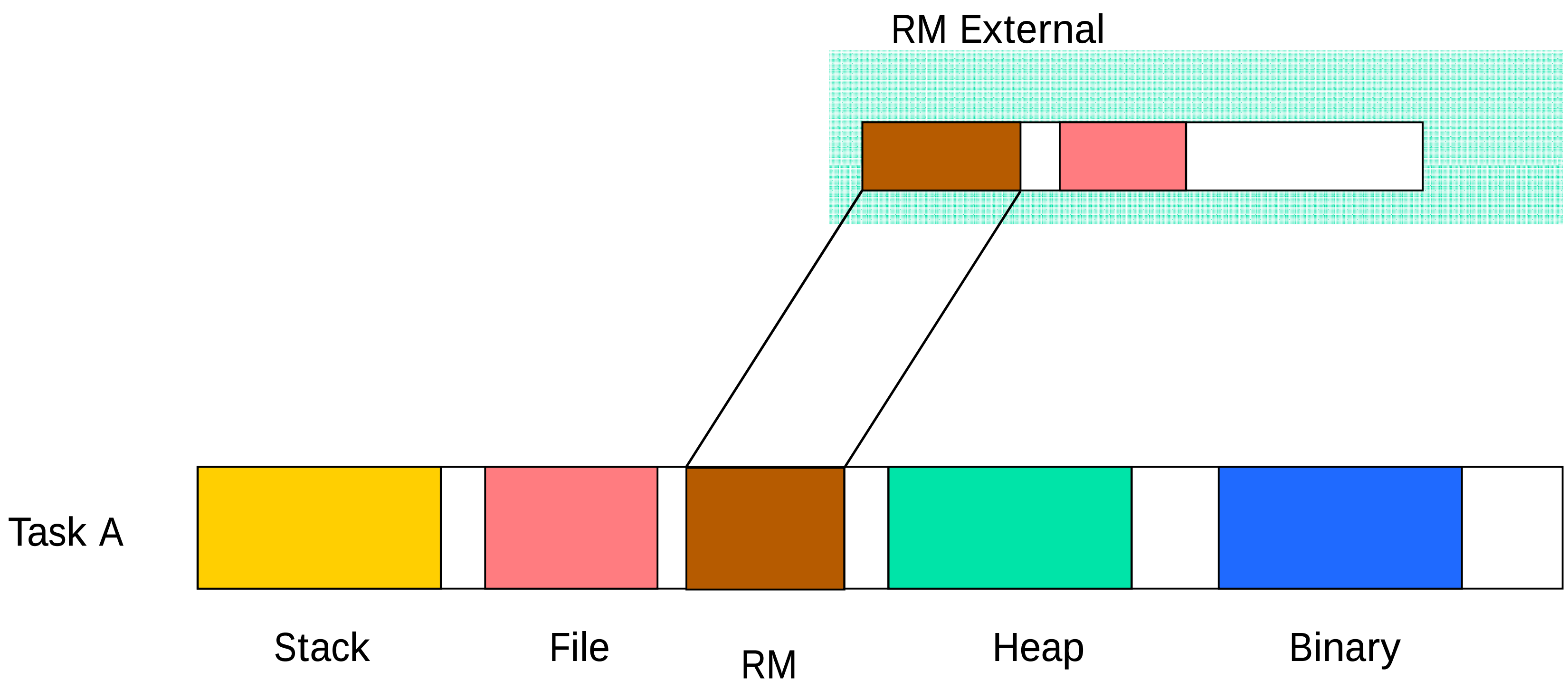
1. access memory
2. kernel pagefault IPC
3. identify region
4. request page mapping



5. memory „prepared“
6. mapping to rm
7. empty ipc to worker
8. resuming

Paging of the Region Mapper

External Region Mapper



Dataspace Operations

Dataspace manager

- open
  (signature depends on type of dataspace)
- close
- map page
- share / transfer

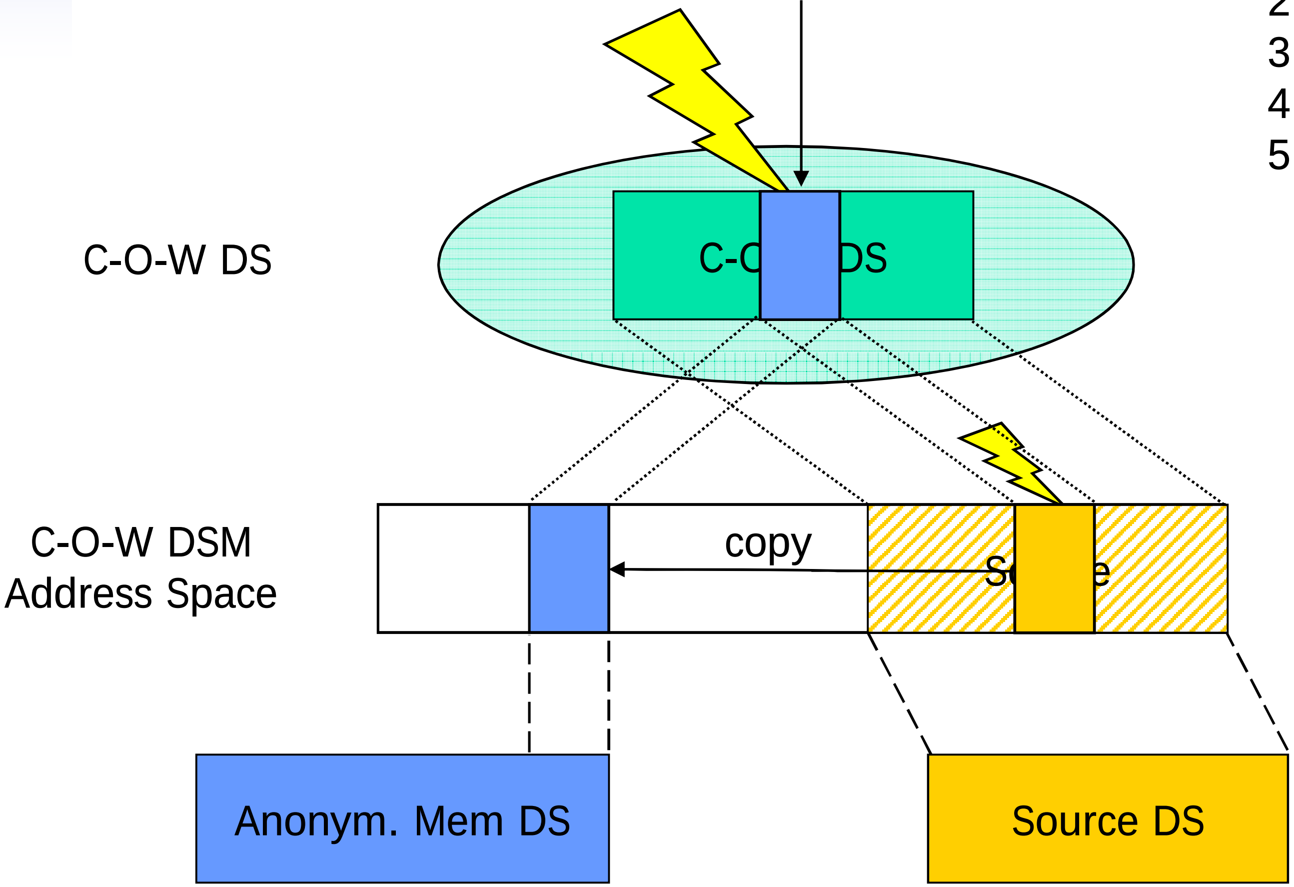
Region mapper

- attach
- detach
- pagefault

Hierachical Dataspaces

- Stacking - build dataspaces with different semantics out of simple dataspaces
- Specializing data container, allowing code reuse

C-O-W Dataspace



read:
1. read
2. mapping from source read olny

write:
1. write
2. pf
3. attach anonym backing
4. copy from source
5. return mapping

Summary Dataspaces

Pros:

+ decomposing
+ distribution
+ simple user pager possible
+ easy sharing
+ flexible
+ customizable

Cons:

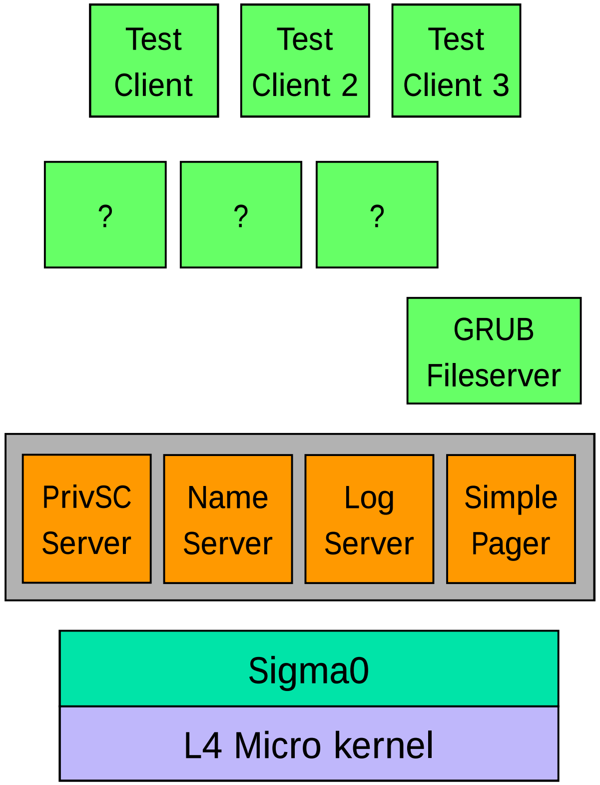
- overhead (on pf. crossing multiple protection domains)
- large virtual address space needed
- leads to confusion on first contact

Further reading

- The Design and Implementation of the 4.3BSD UNIX Operating System
  S.J. Leffler, M.K. McKusick, M.J. Karels, J.S. Quarterman
  Addison Wesley, May 1989
- SunOS Virtual Memory Implementation
  J.P. Moran
  Sun Microsystems, Inc.
- The Sawmill Framework for Virtual Memory Diversity
  M. Aron, L. Deller, K. Elphinstone, T. Jaeger, J. Liedtke, and Y. Park
  Sixth Australasian Computer Systems Architecture Conference (ACSAC2001), Bond University, Gold Coast, Queensland, January 29 - February 2, 2001

Tasks and Virtual Memory in SDI OS

- Wanted:
  - Paged virtual memory for tasks
    - Multiple instances of the same binary
  - Features
    - Create new task from executable file
    - Command line support?
    - Destroy tasks
    - More?
- What else do we need?



SDI OS

- We will implement page-based virtual memory

- What are the implementation issues???

Bits and Pieces

- Frame Table – maintains usable memory
  - What info needs to be in there?
  - Initialize based on available memory
    - Where does the memory come from?

    → Anonymous memory server
- Page Table – maintains virtual memory layout
  - Suggest 2-level x86-like
    - Advantage of having 4K node sizes, easy (de)allocation
  - Alternative: Section list
- Process Table
  - Bookkeeping about processes
    - Pointer to page table
    - Thread id, state, etc

Bits and Pieces

- Loader
  - Takes an ELF file
    - Builds an image using physical frames
    - Builds a page table to map virtual page in the new address space to frames in the root task
  - Don't forget: You need a convention for stack handling.
- Pager
  - Receives page faults.
  - Sends mappings based on information about task
    - Stored in a page table
    - Stored in section list
- Process Manager
  - Creating processes
  - Destroying processes
  - Listing processes?

Upcoming talks (thursday in a week)

- Memory and device server groups
  - Think up how to achieve paged virtual memory to support multiple testclients
    - What components?
    - How do they interact?
- Thursday: Holiday
- Tuesday Lecture:
  - Device drivers