File Systems Part 7

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Scenarios

- Power failure at inopportune moment
  - "live data" is not modified
  - single lost write can be recovered
- Obscure bug in controller firmware or OS
  - detected by checksum in pointer
- Sysadmin accidentally scribbled on one drive
  - detected and repaired
- Out of disk space
  - add to the pool; SPA will cope
- Out of address space
  - 2^{128} is big
    - 1 address per cubic yard of a sphere bounded by the orbit of Neptune

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

More from ZFS

- Adaptive replacement cache
- Advanced prefetching

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

LRU Caching

- LRU cache holds n least-recently-used disk blocks
  - working sets of current processes
- New process reads n-block file sequentially
  - cache fills with this file's blocks
  - old contents flushed
  - new cache contents never accessed again

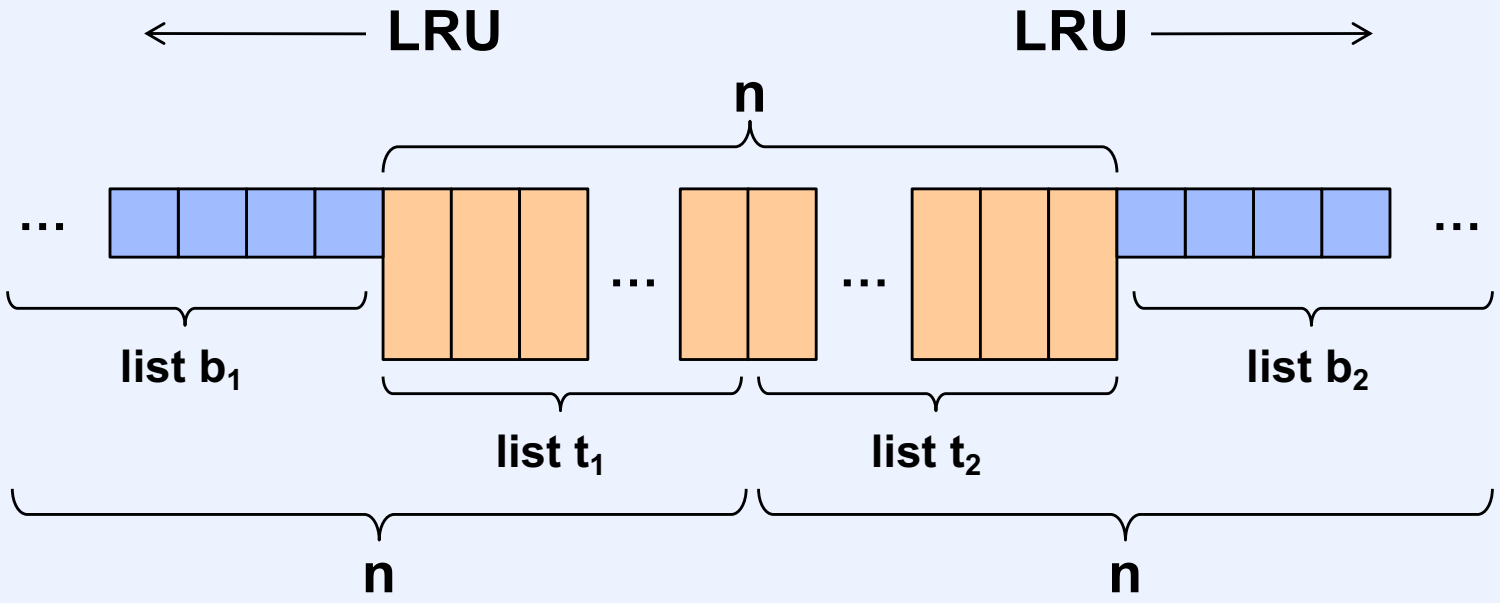
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

(Non-Adaptive) Solution

- Split cache in two
  - half of it is for blocks that have been referenced exactly once
  - half of it is for blocks that have been referenced more than once
- Is 50/50 split the right thing to do?

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Adaptive Replacement Cache

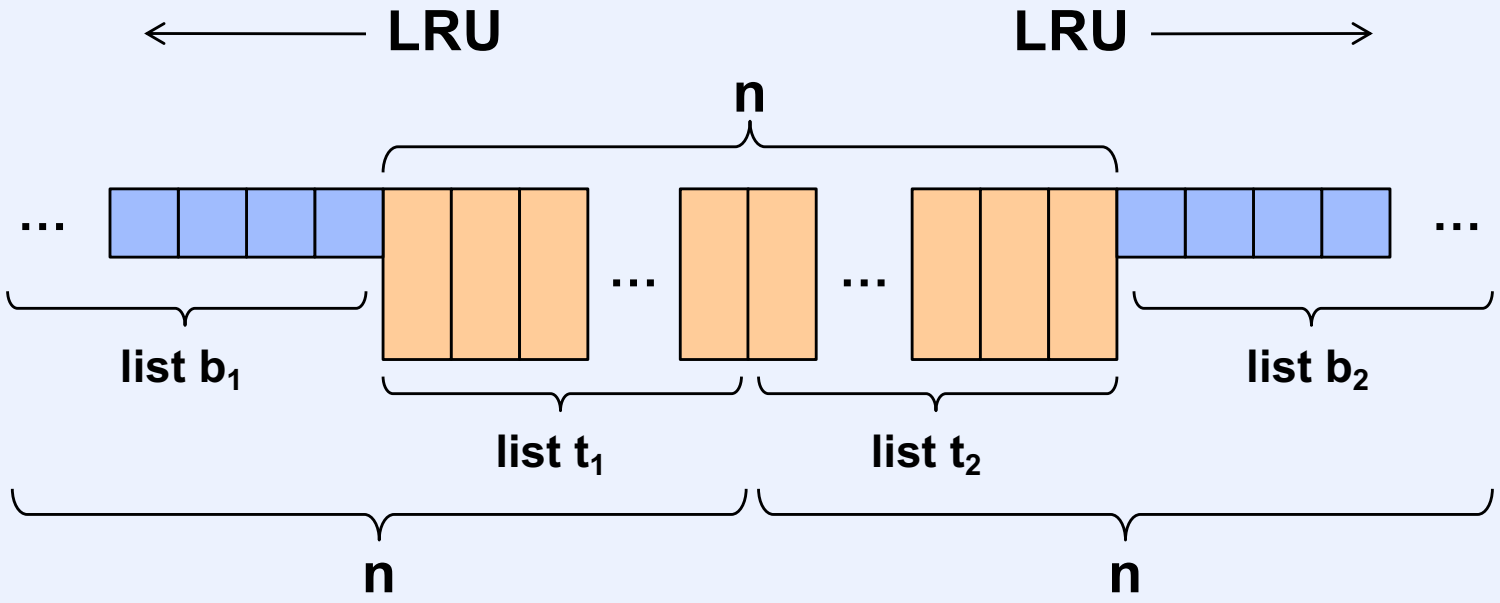


t_1 ; b_1:
LRU list of blocks referenced once
t_1 list (most recently used) contain contents
b_1 list (least recently used) contain just references

t_2 ; b_2:
LRU list of blocks referenced more than once
t_2 list (most recently used) contain contents
b_2 list (least recently used) contain just references

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

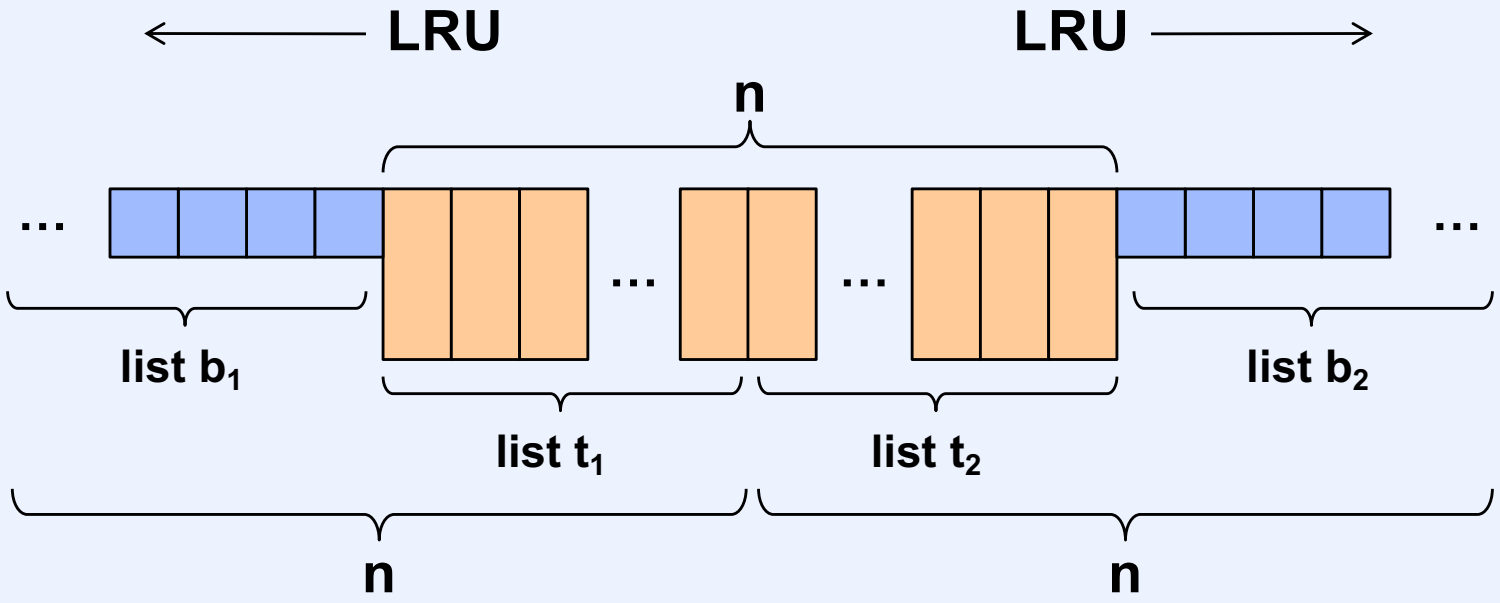
Adaptive Replacement Cache



cache miss:
  if t_1 is full
    evict LRU(t_1) and make it MRU(b_1)
  referenced block becomes MRU(t_1)

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Adaptive Replacement Cache



cache hit:
    if in t_1 or t_2, block becomes MRU(t_2)
    otherwise
        if block is referred to by b_1, increase t_1 space at expense of t_2
        otherwise (referred to by b_1)
            increase t_2 space at expense of t_1
        if t_1 is full, evict LRU(t_1) and make it MRU(b_1)
        if t_2 is full, evict LRU(t_2) and make it MRU(b_2)
        insert block as MRU(t_2)

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Quiz 1

Lists b_1 and b_2 do not contain cached blocks, but just their addresses. Why are they needed?

a) So that one can determine how much better things would be if the cache were twice as large

b) As placeholders so that when these blocks are read in, it's known where in the cache they would go

c) So that we would know, if the addressed block is referenced, whether it would have been in the cache if the corresponding t list were larger

d) They are used by the file system to help determine block reference patterns

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Prefetch

- FFS prefetch
  - keeps track of last block read by each process
  - fetches block i+1 if current block is i and previous was i-1
  - chokes on
    - diff file1 file2

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

zfetch

- Tracks multiple prefetch streams
- Handles four patterns
  - forward sequential access
  - backward sequential access
  - forward strided access
    - iterating across columns of matrix stored by columns
  - backward strided access

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Apple File System (APFS)

- Optimized for SSDs
  - can be used with HDDs
- Utilizes shadow paging
  - called “crash protection”
- Cloning
  - utilizes “copy on write” to make inexpensive clones of files
- snapshots
  - accessed via “Disk Utility”

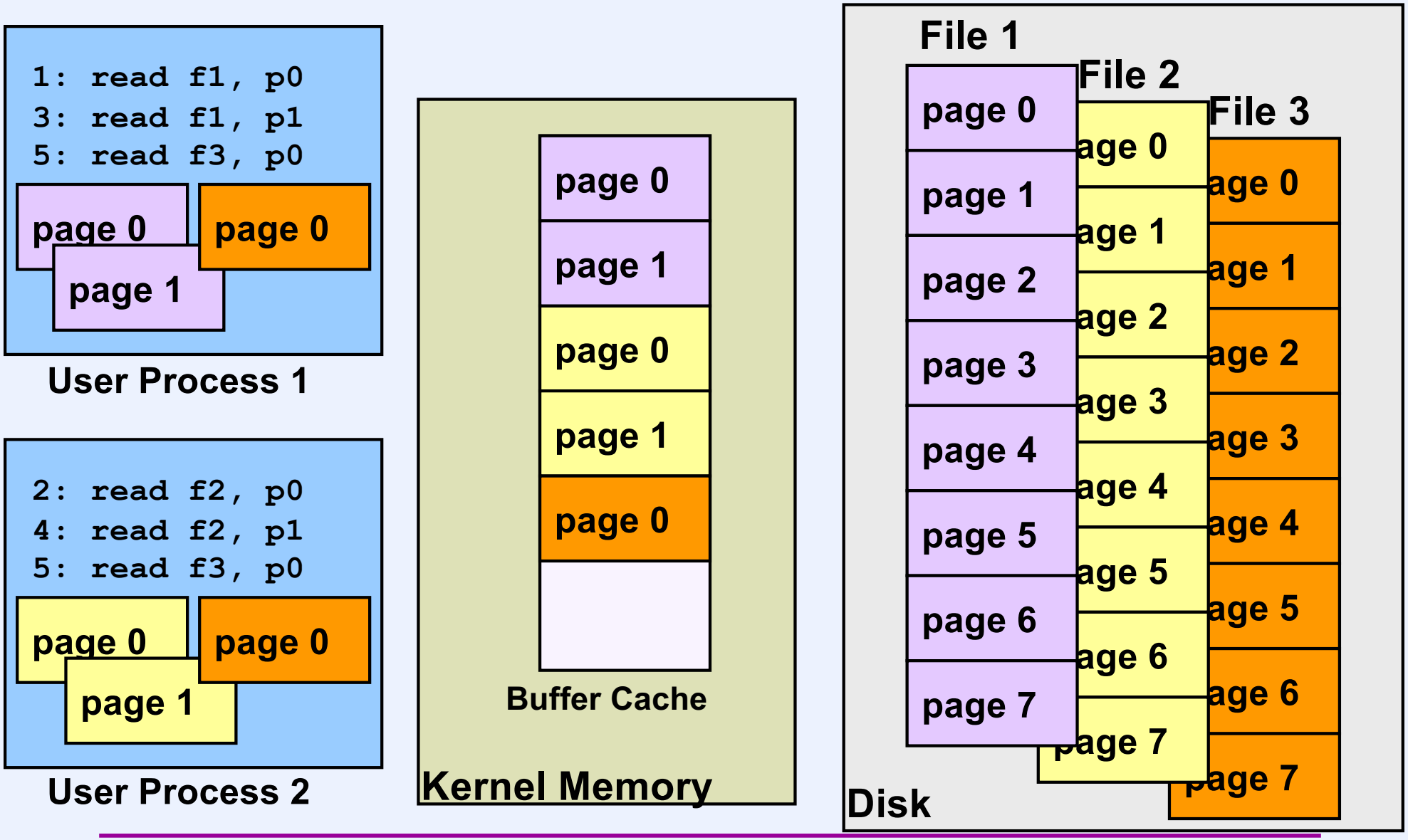
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

File System Implementation Concerns

- System-call access to files vs. mmap access
  - VFS integration with virtual memory
- File-based block indexing vs. file-system-based block indexing
- File-system block size vs. page size
  - conveniently identical on Weenix (4096 bytes)

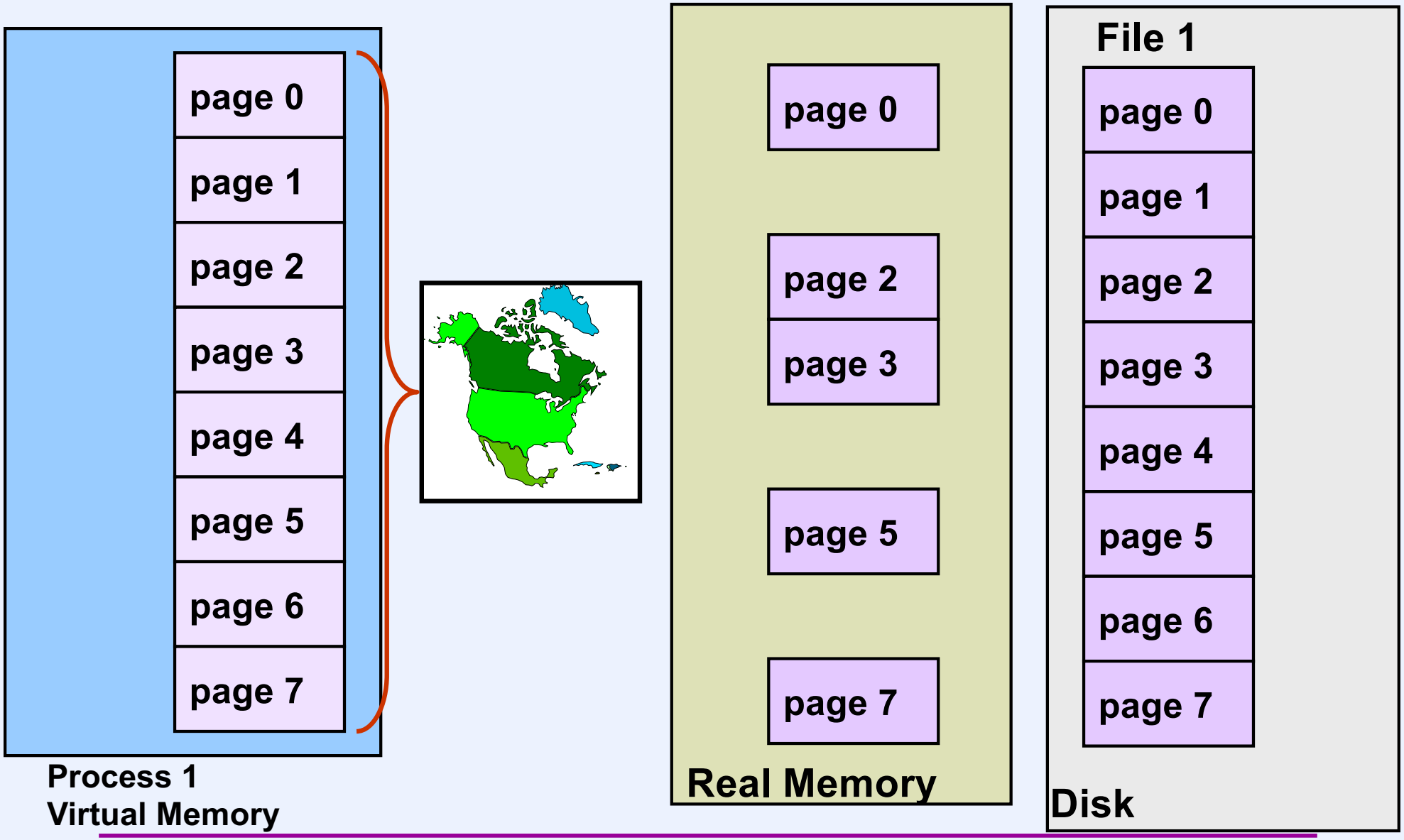
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Traditional I/O



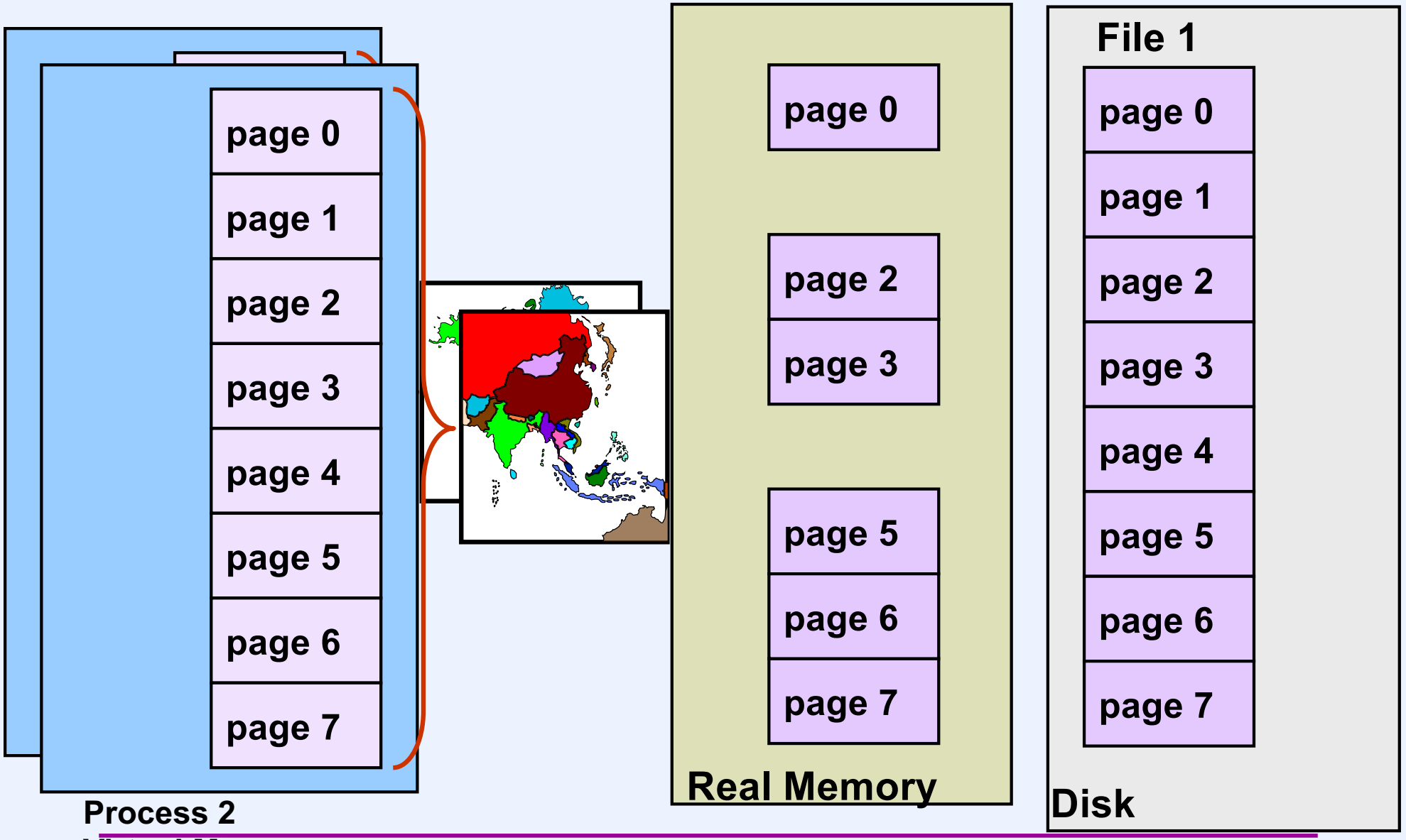
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Mapped File I/O



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Multi-Process Mapped File I/O



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Mapped Files

- **Traditional File I/O**

```
char buf[BigEnough];
fd = open(file, O_RDWR);
for (i=0; i<n_recs; i++) {
    read(fd, buf, sizeof(buf));
    use(buf);
}
```

- **Mapped File I/O**

```
void *MappedFile;
fd = open(file, O_RDWR);
MappedFile = mmap(... , fd, ...);
for (i=0; i<n_recs; i++)
    use(MappedFile[i]);
```

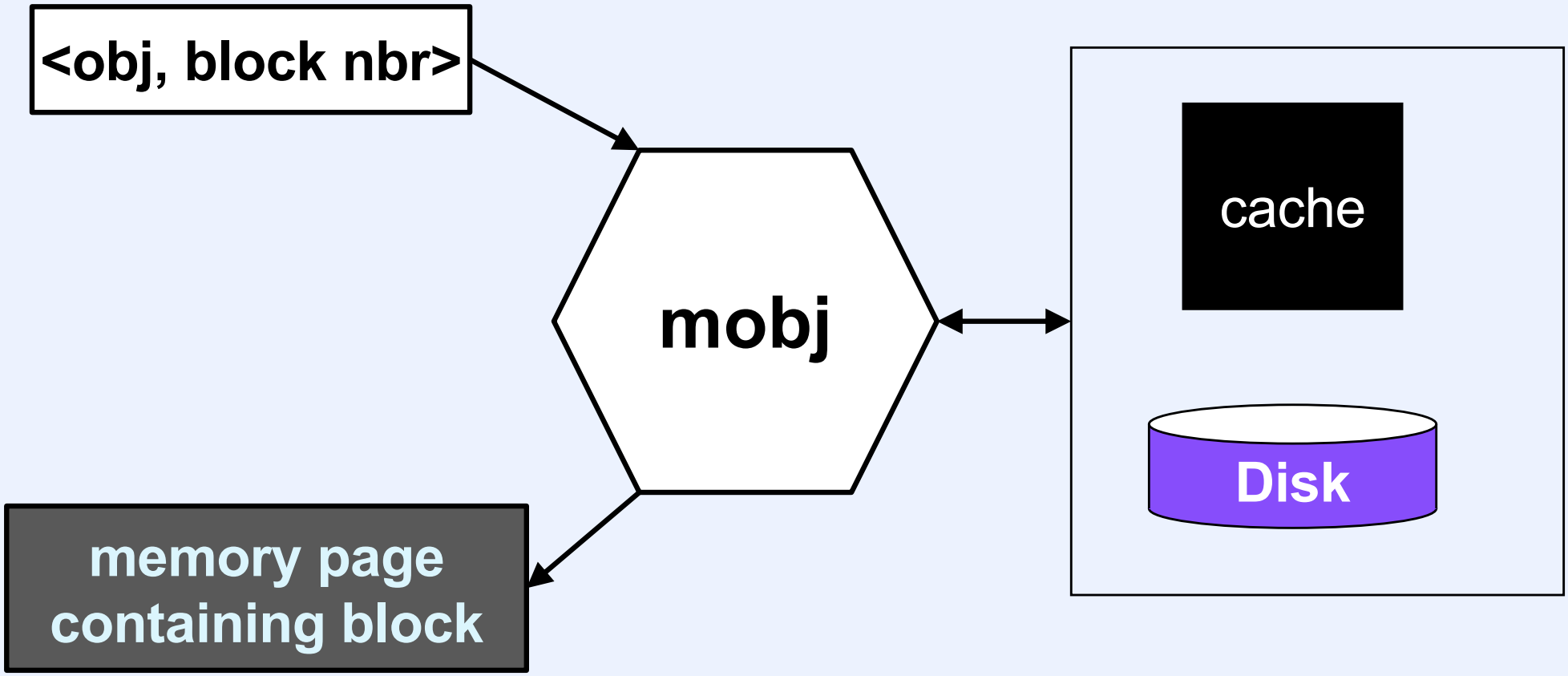

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Consistency

```
typedef struct
    {int flags; char morestuff[OSIZE];} object_t;
object_t object, *mregion;
int fd;
int buf;
fd = open("file", O_RDWR);
mregion = (object_t *)mmap(0, sizeof(object),
    PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
buf = 6;
write(fd, &buf, sizeof(buf));
if (mregion->flags != 6)
  fprintf(stderr, "something is wrong!\n");
```

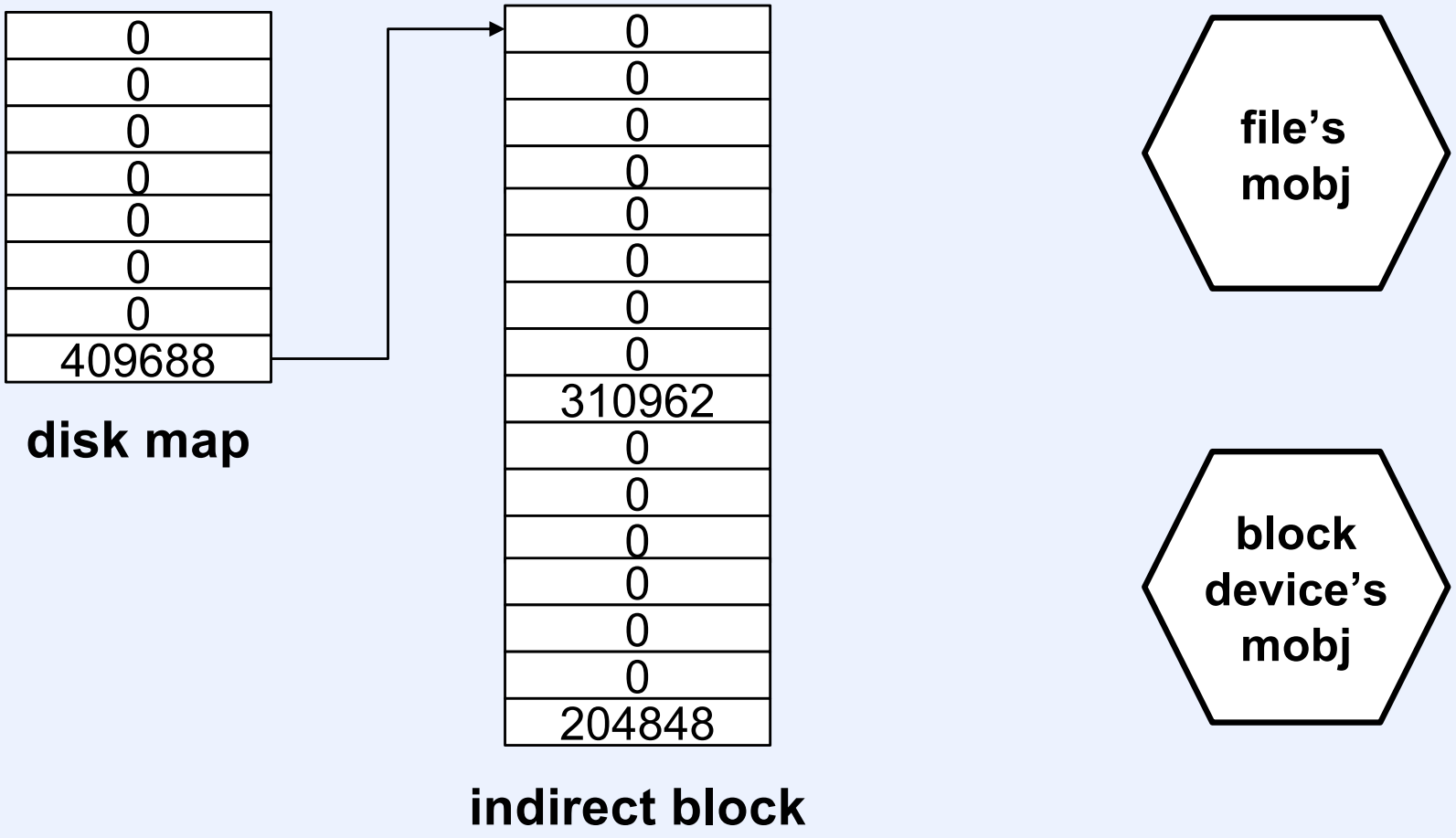

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Memory Objects



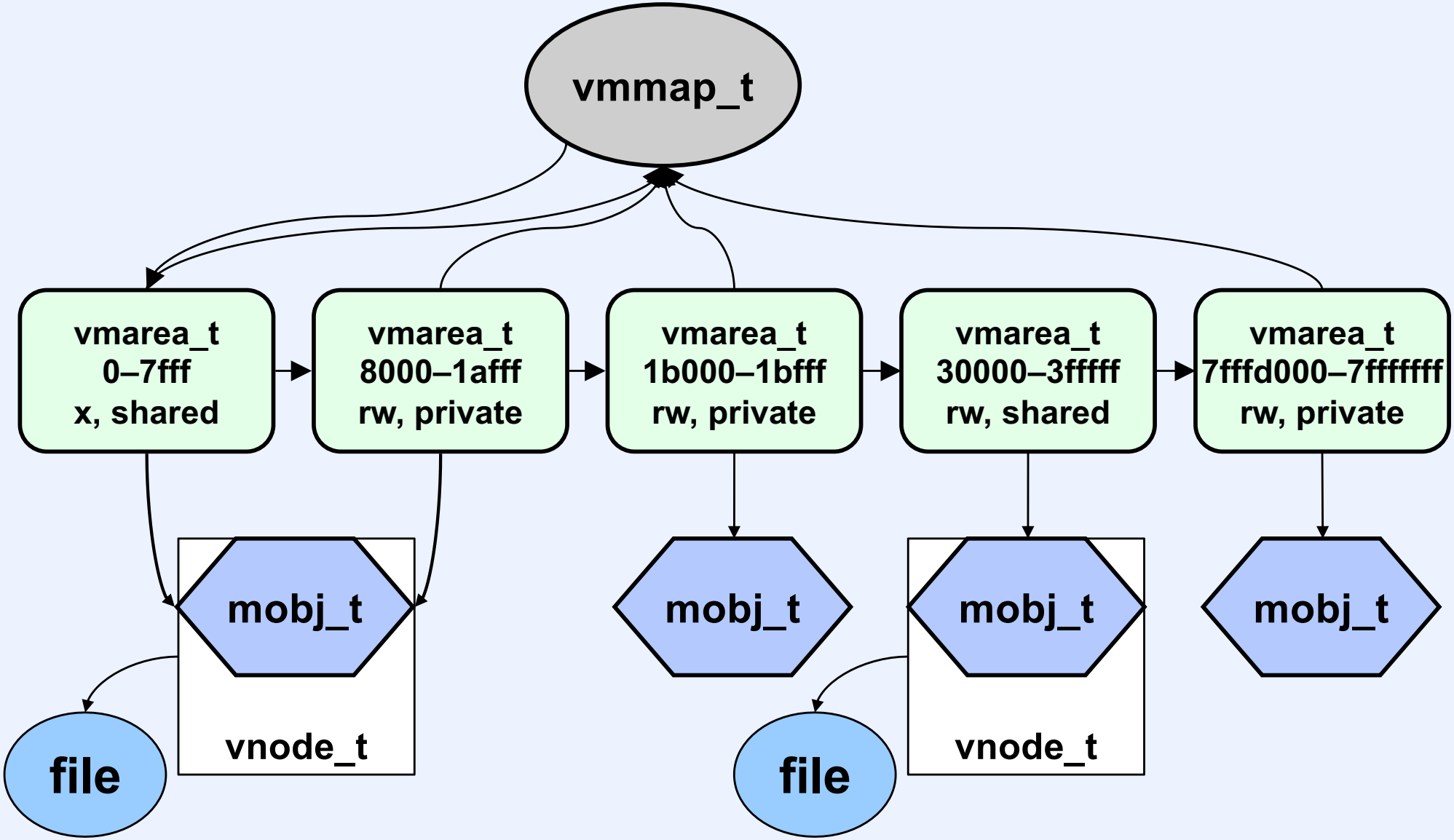
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Memory Objects and Files



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Weenix Address Space Rep



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Some Data Structures

- pframe_t
  - represents a page frame
    - points to actual frame
    - refers to frame in lists
- mobj_t
  - refers to list of in-memory pages (page frames) of an object such as a file
  - page frames represented by pframe_t's
- vmarea_t
  - represents a region within an address space
  - into which an object is mapped
    - represented by an mobj_t

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

More

- vnode_t
  - represents an open file
  - isolates most of OS from details of file system
  - contains
    - function pointers for file ops
    - mobj_t for in-memory file pages
    - adjacent to inode for S5FS files

```
typedef struct s5_node {
    vnode_t vnode;
    s5_inode_t inode;
    long dirtied_inode;
} s5_node_t;
```


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

vnode

```
typedef struct vnode {
   unsigned short vn_refcount;
   struct fs *vn_vfsmounted;
   struct fs *vn_vfs;
   unsigned long vn_vno;
   int vn_mode;
   int vn_len;
   link_list_t vn_link;
   kmutex_t vn_mutex;
   struct vnode_ops *vn_op;
      /* function pointers */
   mobj_t mobj;
   void *vn_i;
      /* extra stuff in subclasses */
} vnode_t;
```


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Caching

- A file's list of cached pages is in its mobj_t
- System-call file access
  - get to file's mobj_t via vnode_t, and then to block device's mobj_t
- Mmap file access
  - get to mobj_t via vmarea_t

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

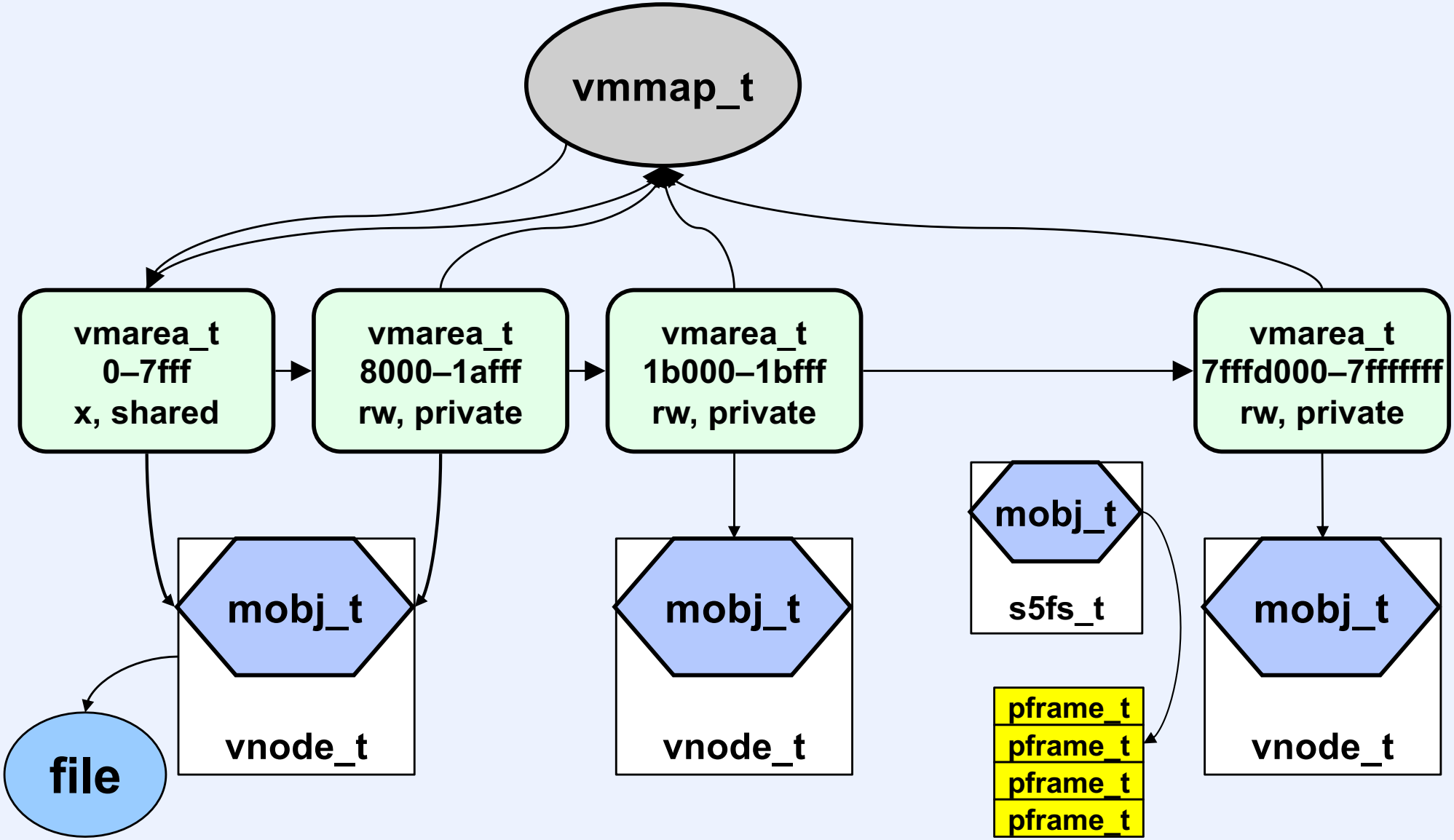
s5fs

```
typedef struct fs {
  char fs_dev[STR_MAX];
  char fs_mountpt[STR_MAX];
  struct vnode *fs_vnodecovered;
  struct vnode *fs_root;
  fs_ops_t *fs_op;
     /* function pointers */
  void *fs_i;
     /* extra stuff in subclasses */
} fs_t;
```

```
typedef struct s5fs {
  blockdev_t *s5f_bdev;
     // refers to fs's mobj
  s5_super_t s5f_super;
  kmutex_t s5f_mutex;
  fs_t *s5f_fs;
} s5fs_t;
```


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Weenix Address Space Rep



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Quiz 2

A file is created. A byte, x, is written at location 2^{20}. A byte, y, is read from location 2^{10}. Assume nothing happens that would cause file blocks to be removed from kernel memory, and no other blocks have been accessed.

a) No blocks of the file are cached in system memory

b) The block containing x is cached, nothing else

c) The blocks containing x and the indirect block are cached, nothing else

d) The blocks containing x and y are cached, nothing else

e) The blocks containing x, y, and the indirect block are cached, nothing else

f) The blocks containing x, y, the indirect block, and some other blocks are cached

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

```
1  static long s5fs_get_pframe(..., long forwrite, pframe_t **pfp) {
2      if (vnode->vn_len <= pagenum * PAGE_SIZE)
3           return -EINVAL;
4      int new;
5      long loc =
           s5_file_block_to_disk_block(VNODE_TO_S5NODE(vnode),
               pagenum, forwrite, &new);
5      if (loc < 0) return loc;
6      if (loc) {
7          if (new)
8              *pfp = s5_file_cache_and_clear_block(&vnode->vn_mobj
9                  pagenum, loc);
10         else {
11             s5_get_file_disk_block(vnode, pagenum, loc, forwrite,
                   pfp);
12         }
13         return 0;
14     } else {
     …
```


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

```
1  static long s5fs_get_pframe(..., long forwrite, pframe_t **pfp) {
2      if (vnode->vn_len <= pagenum * PAGE_SIZE)
3           return -EINVAL;
4      int new;
5      long loc =
           s5_file_block_to_disk_block(VNODE_TO_S5NODE(vnode),
               pagenum, forwrite, &new);
5      if (loc < 0) return loc;
6      if (loc) {
7          if (new)
8              *pfp = s5_file_cache_and_clear_block(&vnode->vn_mobj
9                  pagenum, loc);
10         else {
11             s5_get_file_disk_block(vnode, pagenum, loc, forwrite,
                  pfp);
12         }
13         return 0;
14     } else {
   …
```


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

```
6   if (loc) {
7           if (new)
8               *pfp =
                    s5_file_cache_and_clear_block(&vnode->vn_mobj,
                    pagenum, loc);
6           else {
7               s5_get_file_disk_block(vnode, pagenum, loc,
                    forwrite, pfp);
12          }
13          return 0;
14      } else {
14          KASSERT(!forwrite);
15          return mobj_default_get_pframe(&vnode->vn_mobj,
                pagenum, forwrite, pfp);
16      }
17  }
```


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Quiz 3

Suppose a thread does a *read* system call (which calls *s5fs_get_pframe*) to read a portion of a block that is sparse. It then writes data to the block, using the *write* system call. Will, as part of handling this write, *mobj_default_get_pframe* be called?

a) yes, since the block is sparse, *mobj_default_get_pframe* must be called to zero the block, then modify a portion of it

b) yes, for some other reason

c) no, the block was zeroed by the *read* call

d) no, the block doesn't need to be zeroed and the caller of *s5fs_get_pframe* will fill it in

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Memory Management Part 1

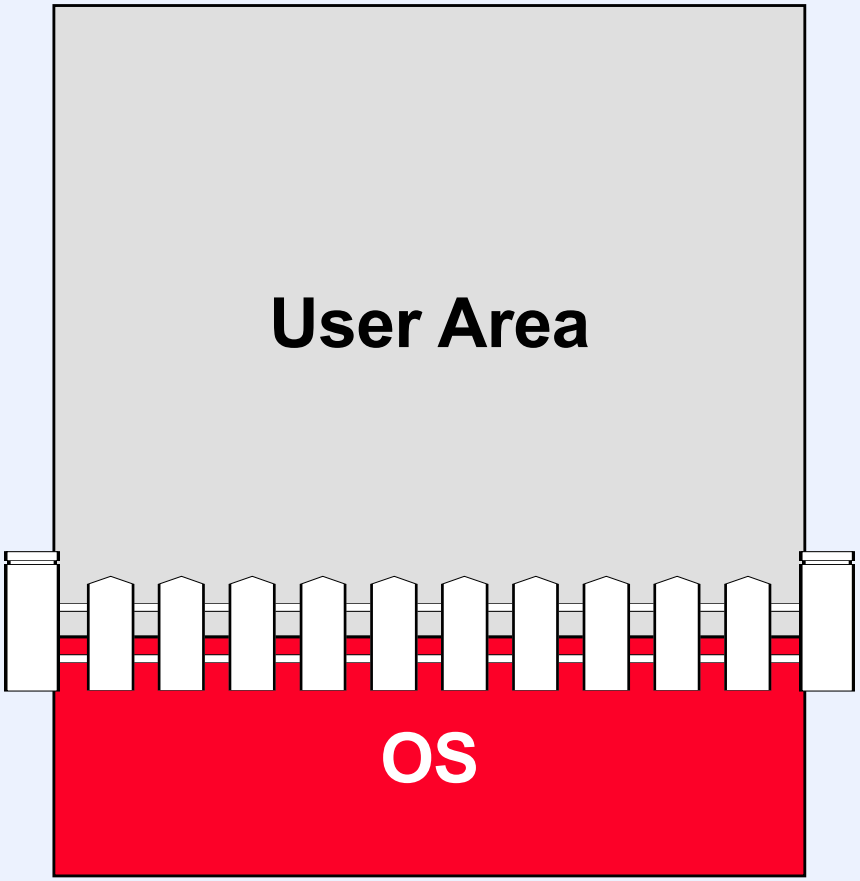
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

The Address-Space Concept

- Protect processes from one another
- Protect the OS from user processes
- Provide efficient management of available storage

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Memory Fence

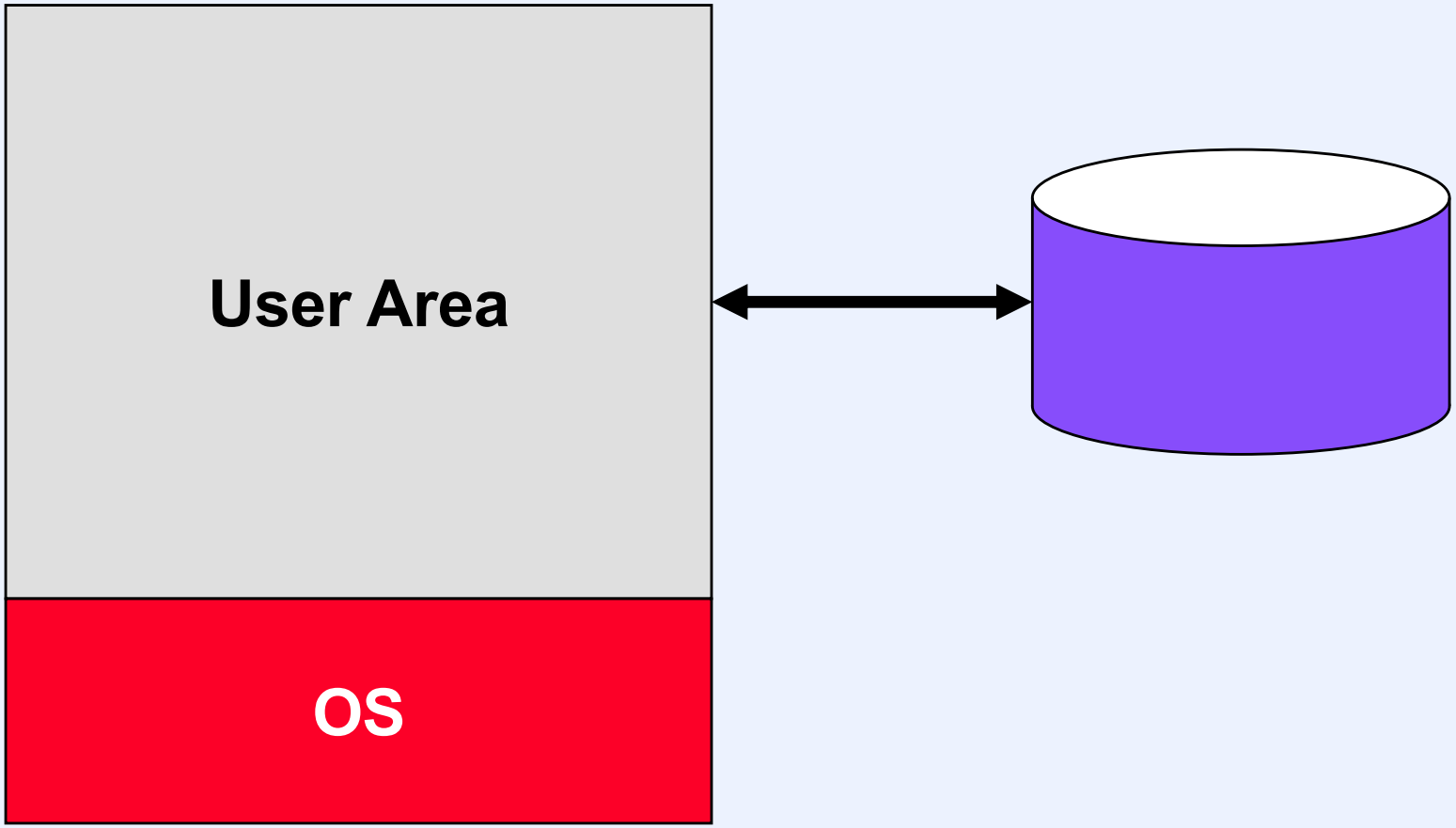


Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Base and Bounds Registers

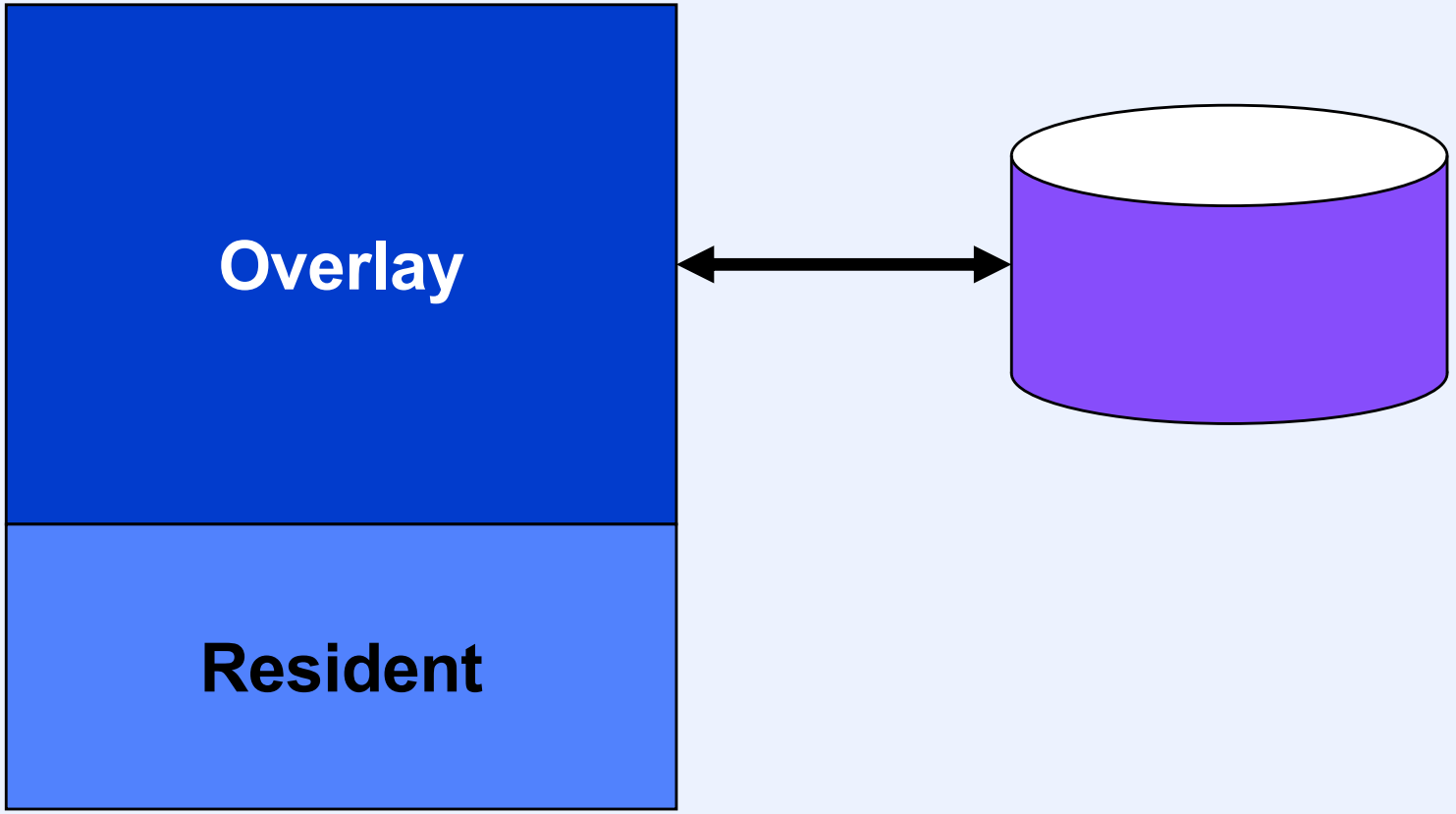
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Swapping



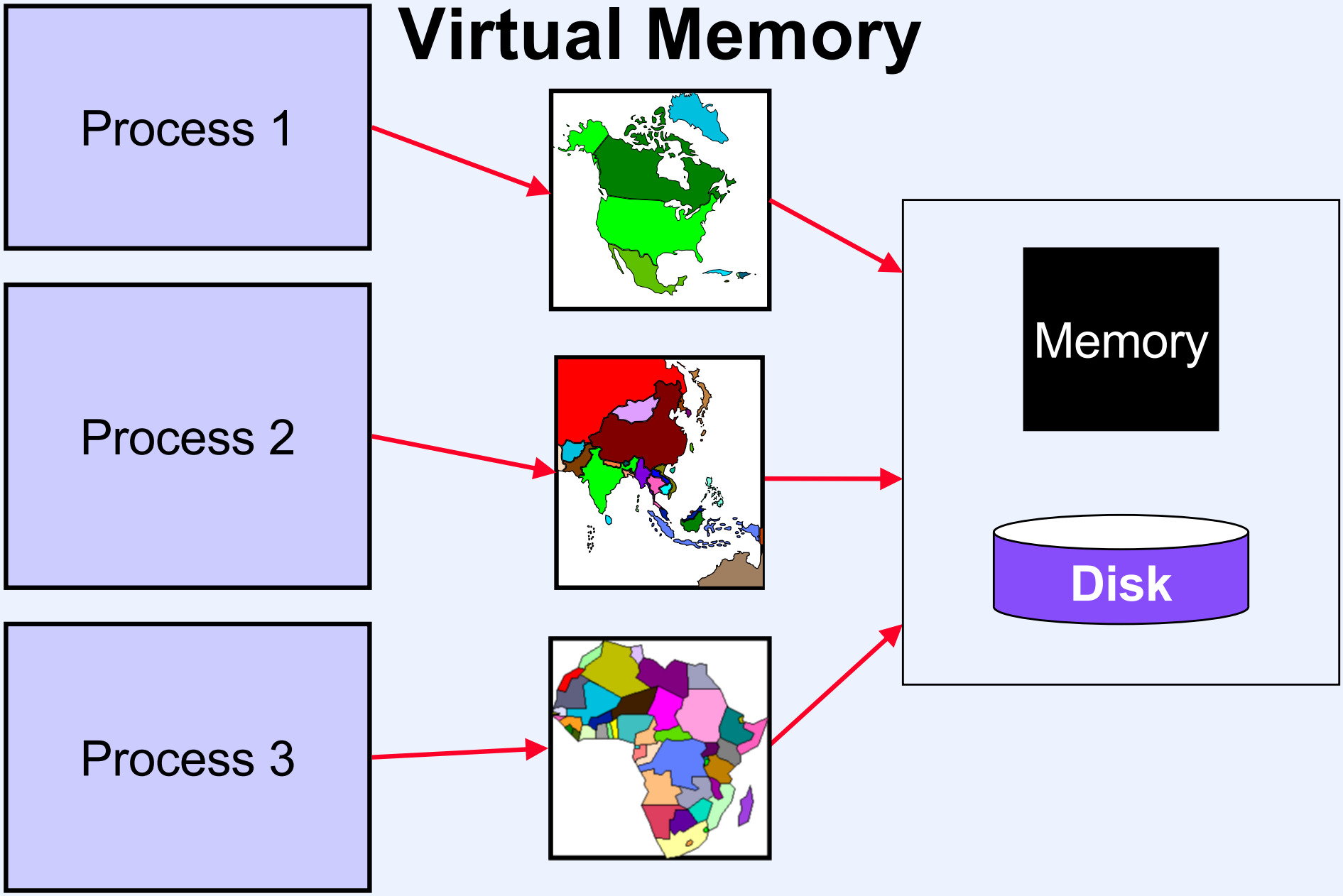
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Overlays



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Virtual Memory



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Structuring Virtual Memory

- Paging
  - divide the address space into fixed-size pages
- Segmentation
  - divide the address space into variable-size segments (typically each corresponding to some logical unit of the program, such as a module or subroutine)

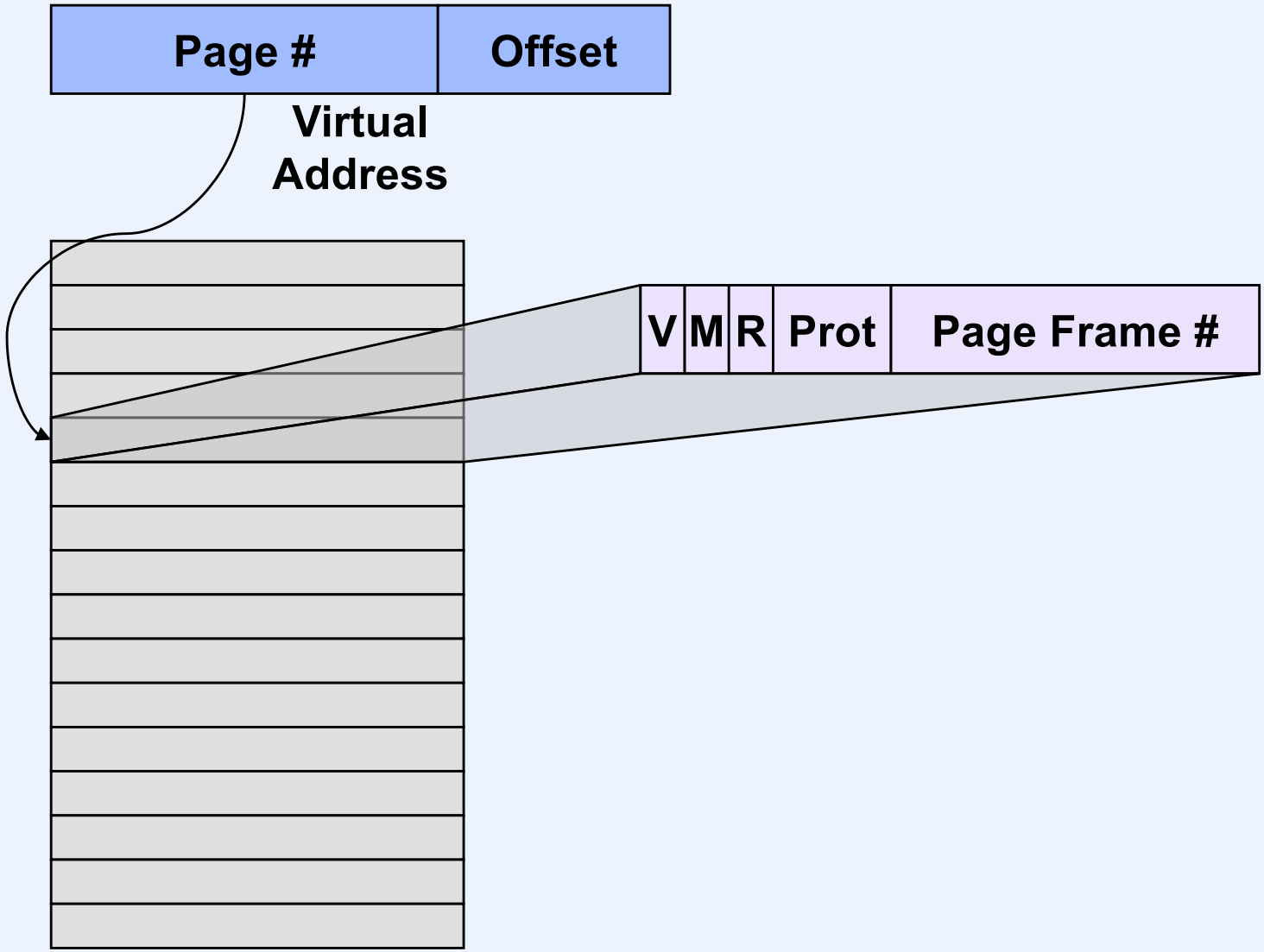
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Paging

- Map fixed-size pages into memory (into page frames)
- Many hardware mapping techniques
  - page tables
  - translation lookaside buffers

Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Page Tables



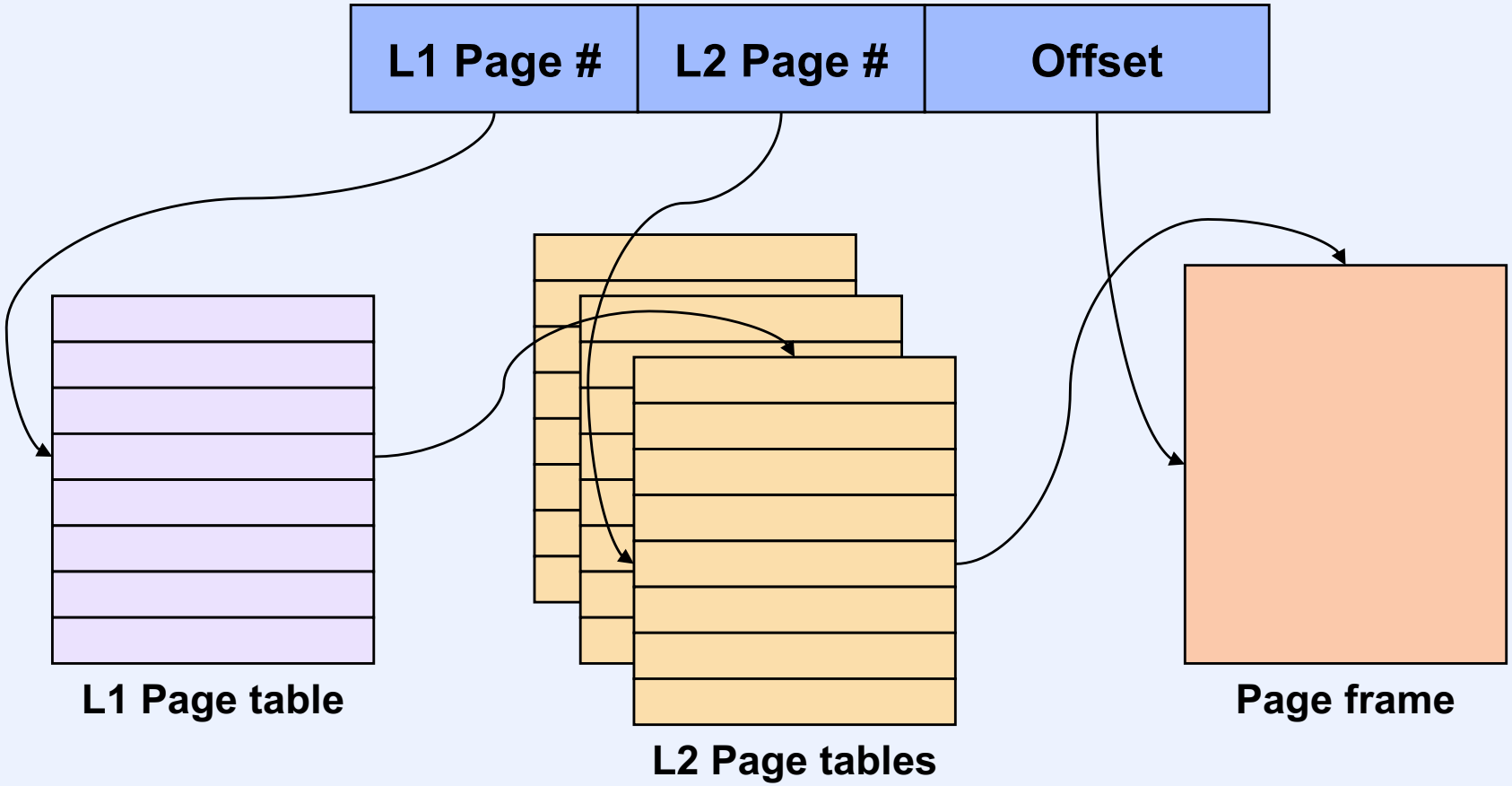
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Page-Table Size

- Consider a full 2^{32}-byte address space
  - assume 4096-byte (2^{12}-byte) pages
  - 4 bytes per page table entry
  - the page table would consist of $2^{32}/2^{12}$ (= 2^{20}) entries
  - its size would be 2^{22} bytes (or 4 megabytes)

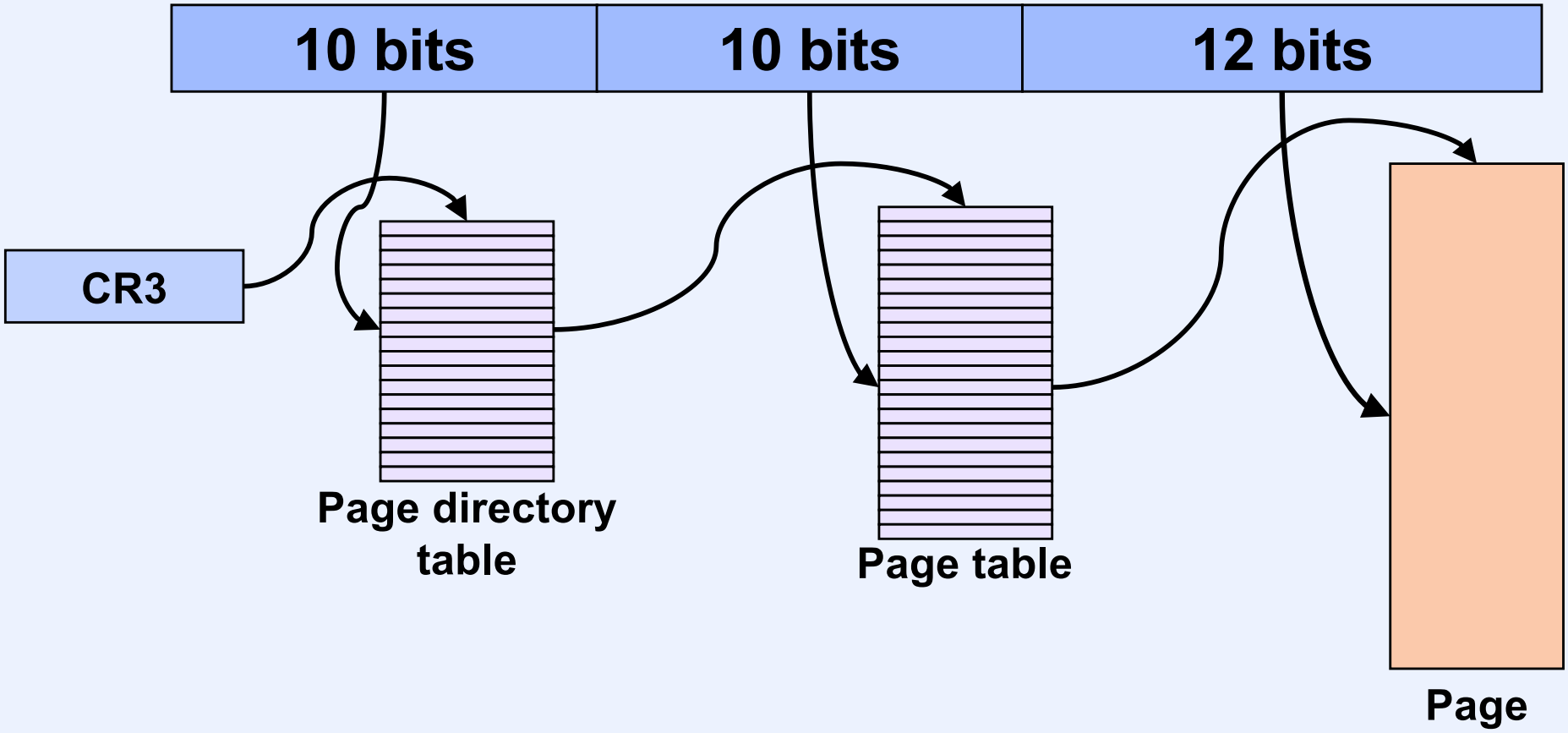
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Forward-Mapped Page Table



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

IA32 Paging



Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Quiz 4

Suppose a process on an IA32 has exactly one page residing in real memory. What is the total number of combined pages of page-directory table and page tables required to map this page?

a) 1

b) 2

c) 4

d) 8

Copyright © 2025 Thomas W. Doeppner. All rights reserved.