Memory Management Part 4

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

Friday's Quiz

We'd like to virtualize EPT. Assume that setting EPTP causes a VMexit if done on a VMM that's not running in real ring -1. What does the VMM running at level 0 (in ring -1) do when it receives such a VMexit from a VMM running at level 1?

a) it sets EPTP to point to the composition of the page tables mapping VMM_0's address space to real memory and the page tables pointed to by the value being attempted to be put in EPT

b) nothing: the EPT mechanism is virtualized by the hardware

c) something else

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

VMX

- New processor mode: root
  - ring -1: root mode
  - rings 0-3: non-root mode
- Certain actions cause processor in non-root mode to switch to root mode
  - VMexit
- When in root mode, processor can switch back to non-root mode
  - VMenter

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

VMCS

- Virtual machine control structures
  – guest state
    - virtualized CPU registers (non-root mode)
  – host state
    - registers to be restored when switching to root mode (VMexit)
  – control data
    - which events in non-root mode cause VMexits

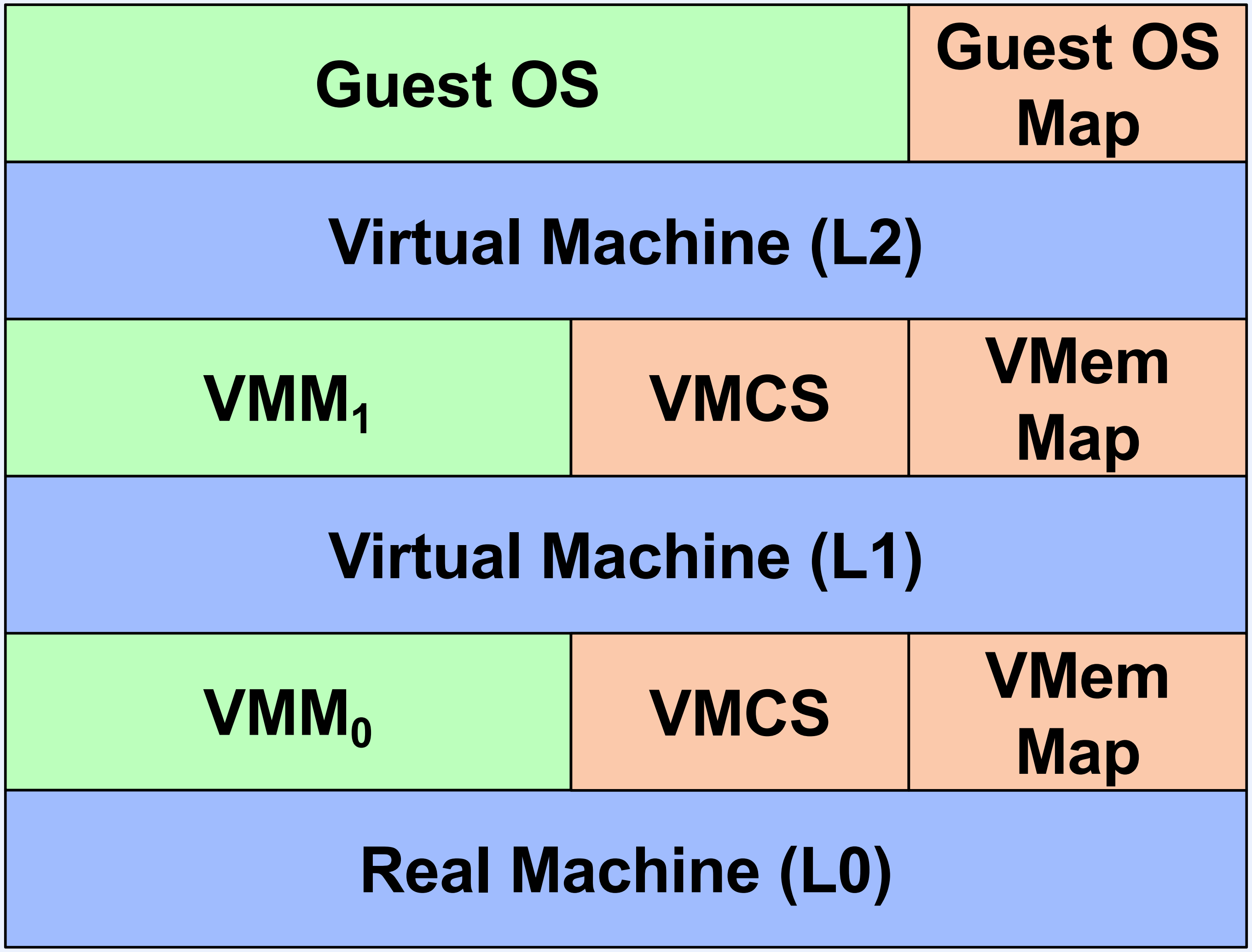
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Nested Virtualization on VMX

- A VMM is designed to use VMX extensions (including EPT)
- It supports VMs that appear to be real x86's (but without VMX extensions)
- Can the VMM run in a VM of the level-0 VMM?

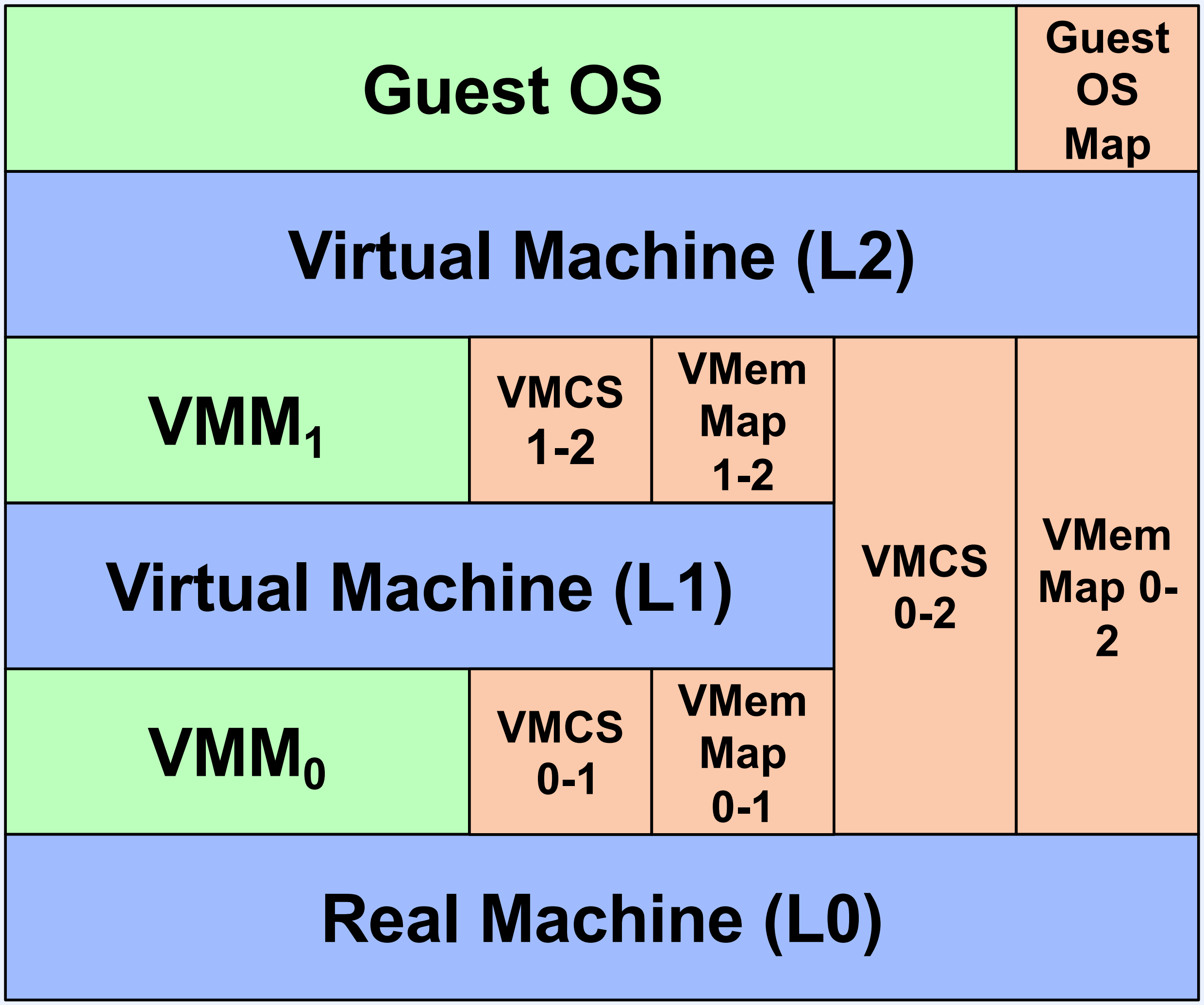
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Nested Virtualization with VMX



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

Composed Virtualization



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

Traditional OS Paging Issues

- Fetch policy
- Placement policy
- Replacement policy

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

A Simple Paging Scheme

- Fetch policy
  - start process off with no pages in primary storage
  - bring in pages on demand (and only on demand — this is known as demand paging)
- Placement policy
  - it usually doesn't matter — put the incoming page in the first available page frame
- Replacement policy
  - replace the page that has been in primary storage the longest (FIFO policy)

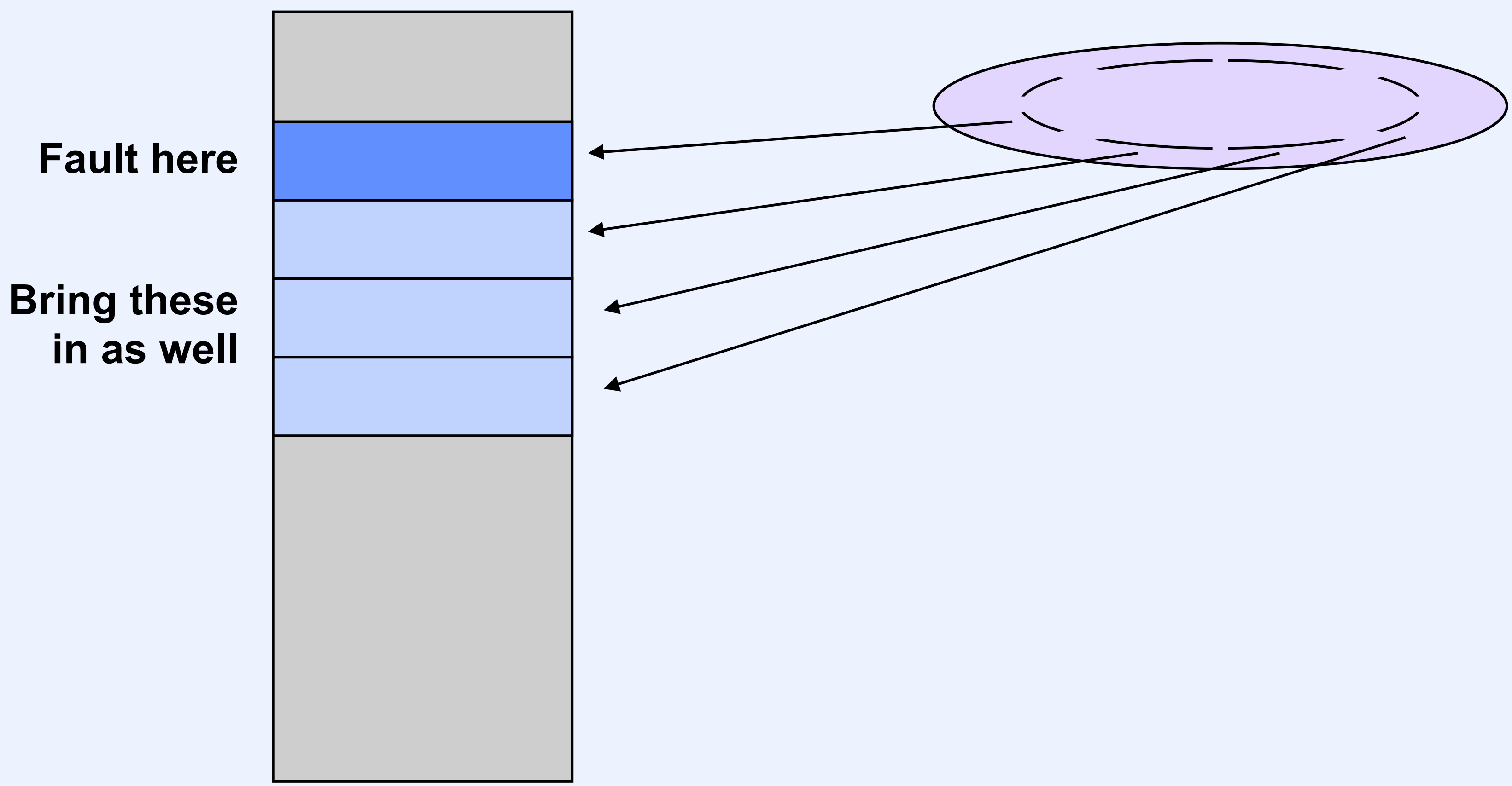
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Performance

1) Trap occurs (page fault)
2) Find free page frame
3) Write page out if no free page frame
4) Fetch page
5) Return from trap

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

Improving the Fetch Policy



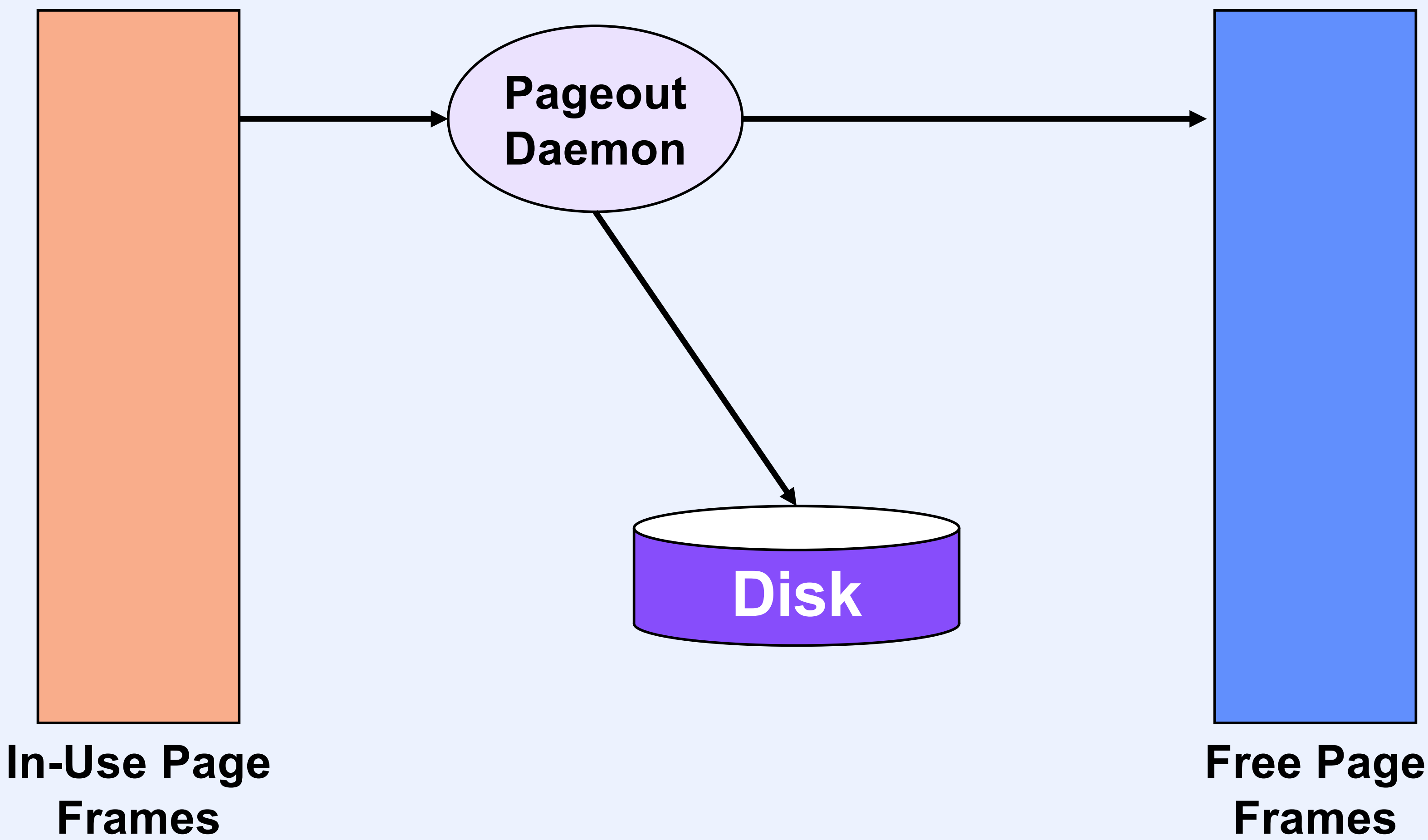
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Improving the Replacement Policy

- **When is replacement done?**
  - doing it “on demand” causes excessive delays
  - should be performed as a separate, concurrent activity
- **Which pages are replaced?**
  - FIFO policy is not good
  - want to replace those pages least likely to be referenced soon

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

The "Pageout Daemon"



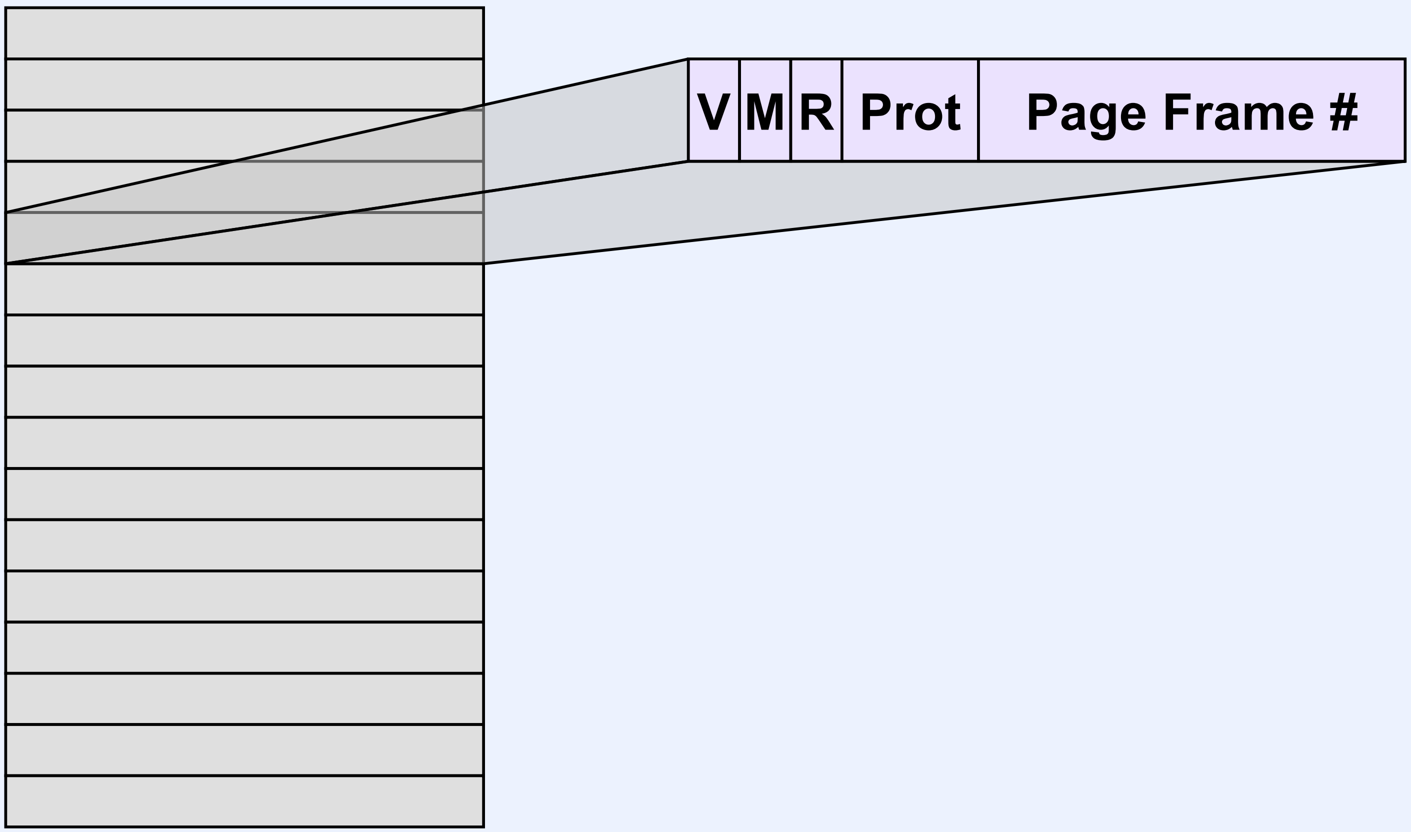
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Choosing the Page to Remove

- Idealized policies:
  - FIFO (First-In-First-Out)
  - LRU (Least-Recently-Used)
  - LFU (Least-Frequently-Used)
- Optimal
  - replace page so as to minimize number of page faults
    - replace page whose next reference is furthest in the future

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

Implementing LRU



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

Quiz 1

Your computer is running one process. Pretty much all available real memory is being actively used and processor utilization is around 90%. You now add another process that's similar to the first in terms of both memory and processor utilization (though it's running a different program). Assume the LRU page replacement policy is used.

a) Processor utilization will rise to nearly 100%

b) Processor utilization will stay at around 90%

c) Processor utilization will drop precipitously

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

Global vs. Local Allocation

- **Global allocation**
  - all processes compete for page frames from a single pool
- **Local allocation**
  - each process has its own private pool of page frames

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

Thrashing

- Consider a system that has exactly two page frames:
  - process A has a page in frame 1
  - process B has a page in frame 2
- Process A causes a page fault
- The page in frame 2 is removed
- Process B faults; the page in frame 1 is removed
- Process A resumes execution and faults again; the page in frame 2 is removed
- ...

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

The Working-Set Principle

- The set of pages being used by a program (the working set) is relatively small and changes slowly with time
  - WS(P,T) is the set of pages used by process P over time period T
- Over time period T, P should be given |WS(P,T)| page frames
  - if space isn't available, then P should not run and should be swapped out

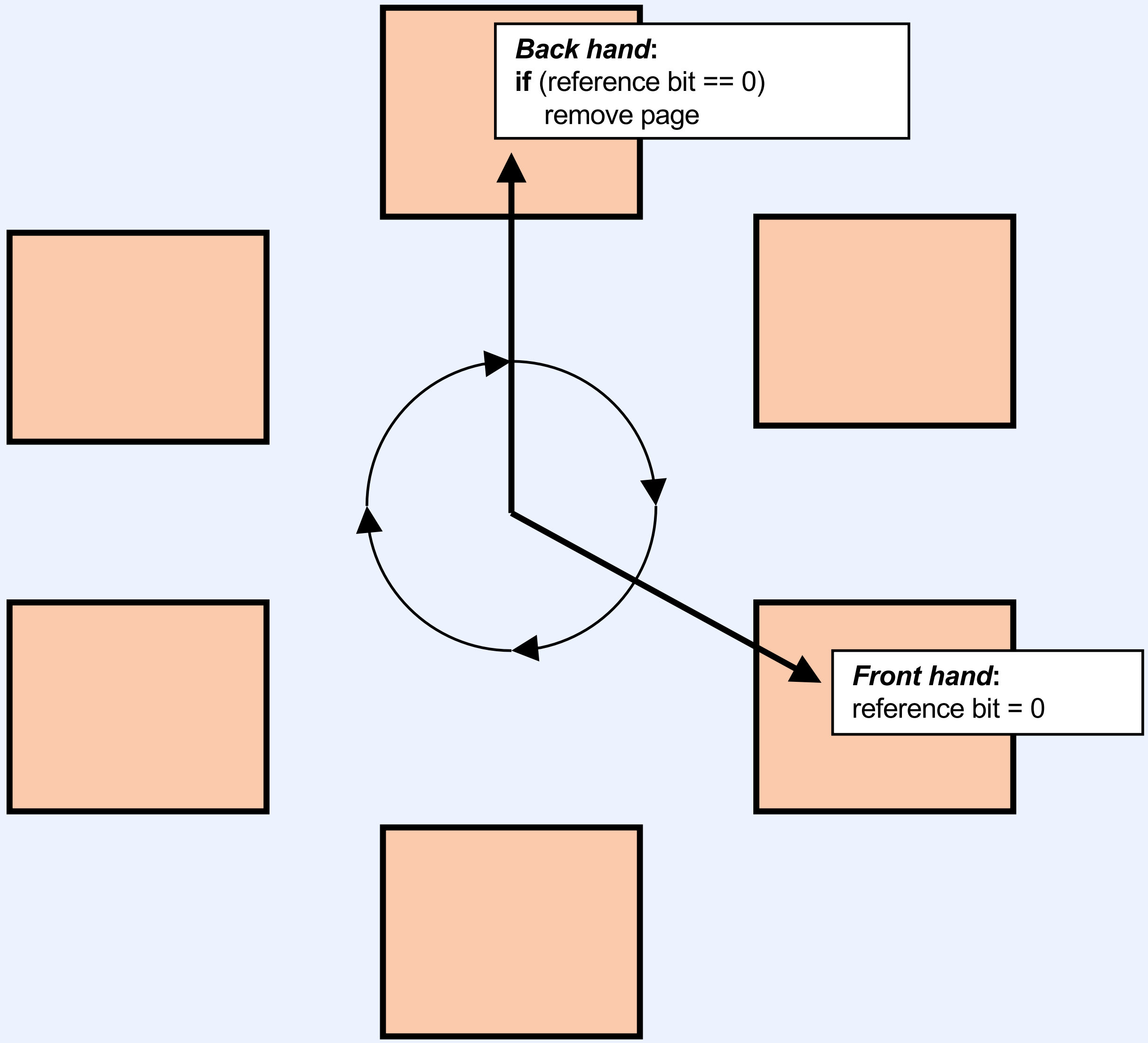
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Two Issues

- If a process is active, which of its pages should be in real memory?
- If there is too much of a demand for memory, which processes should run (and which should not run)?

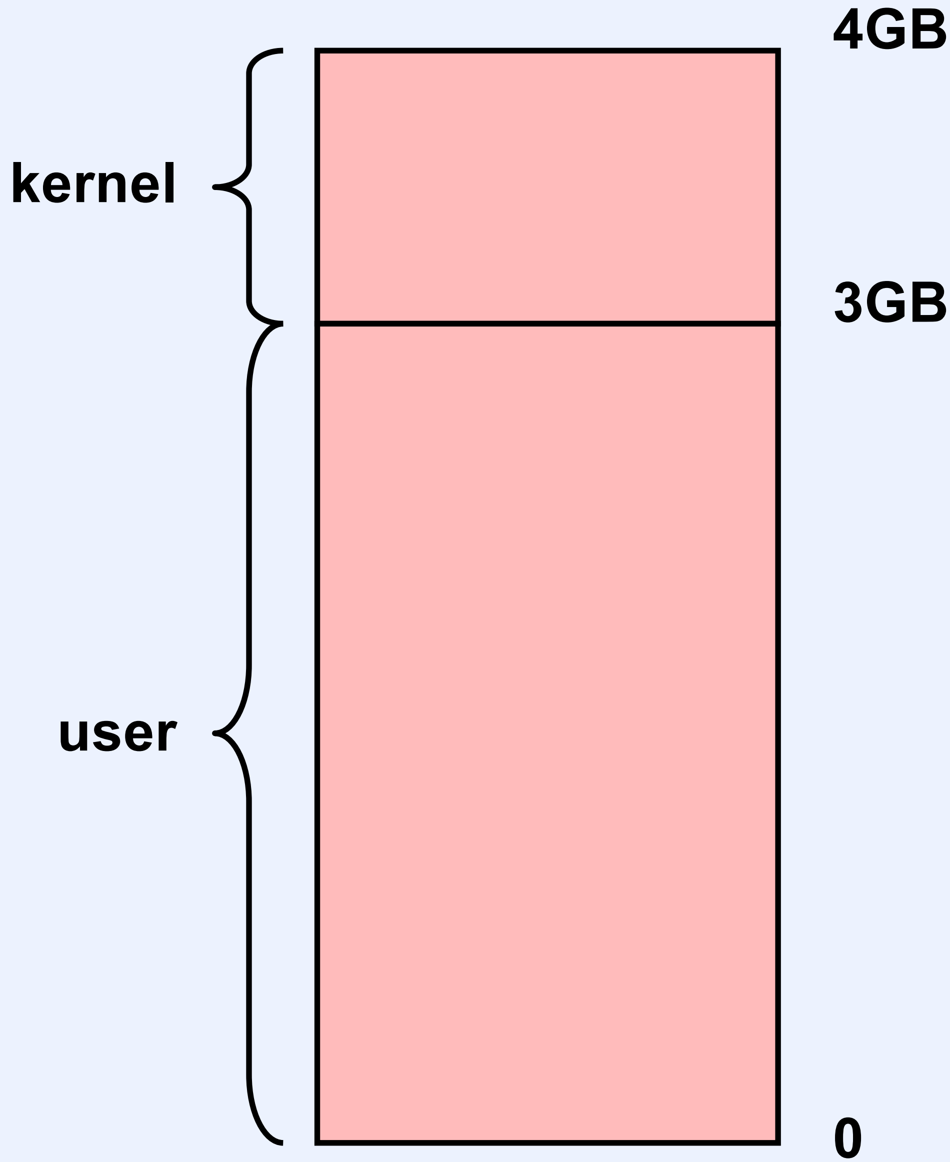
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Clock Algorithm



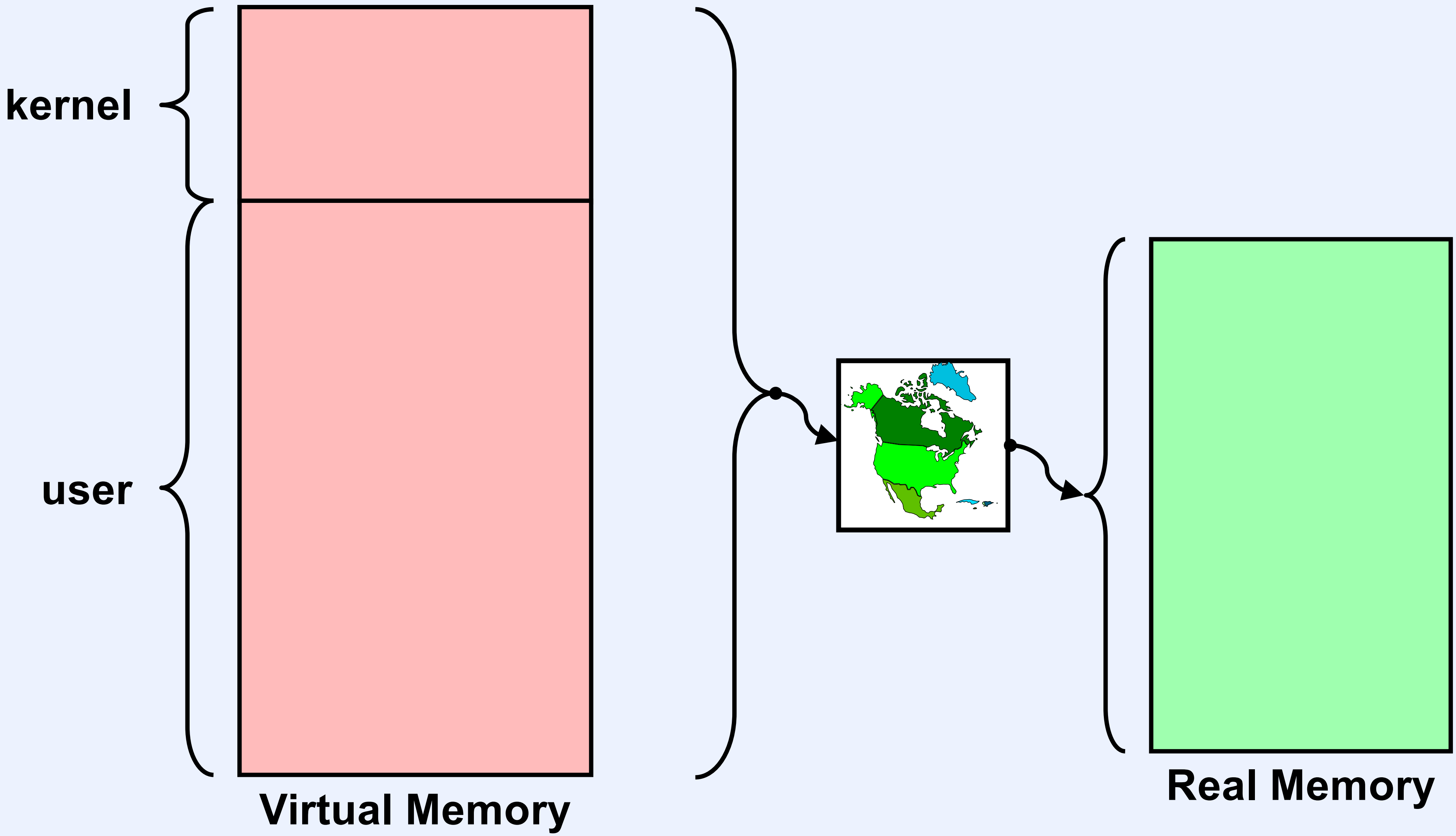
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Linux Intel x86 VM Layout



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

Real Memory



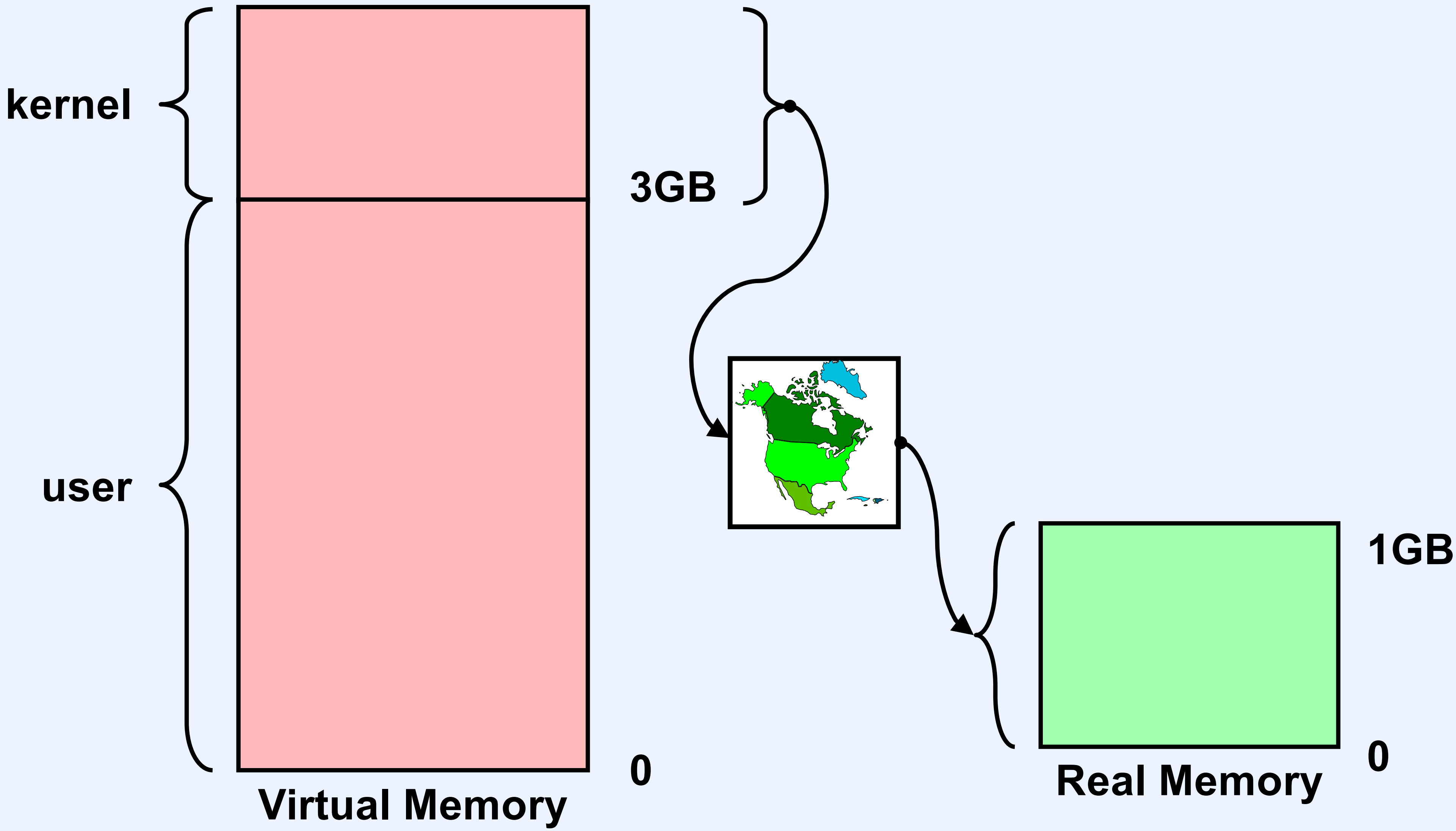
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Memory Allocation

- User
  - virtual allocation
    - fork
    - pthread_create
    - exec
    - brk
    - mmap
  - real allocation
    - (not done)
- OS kernel
  - virtual allocation
    - fork, etc.
    - kernel data structures
  - real allocation
    - page faults
    - kernel data structures

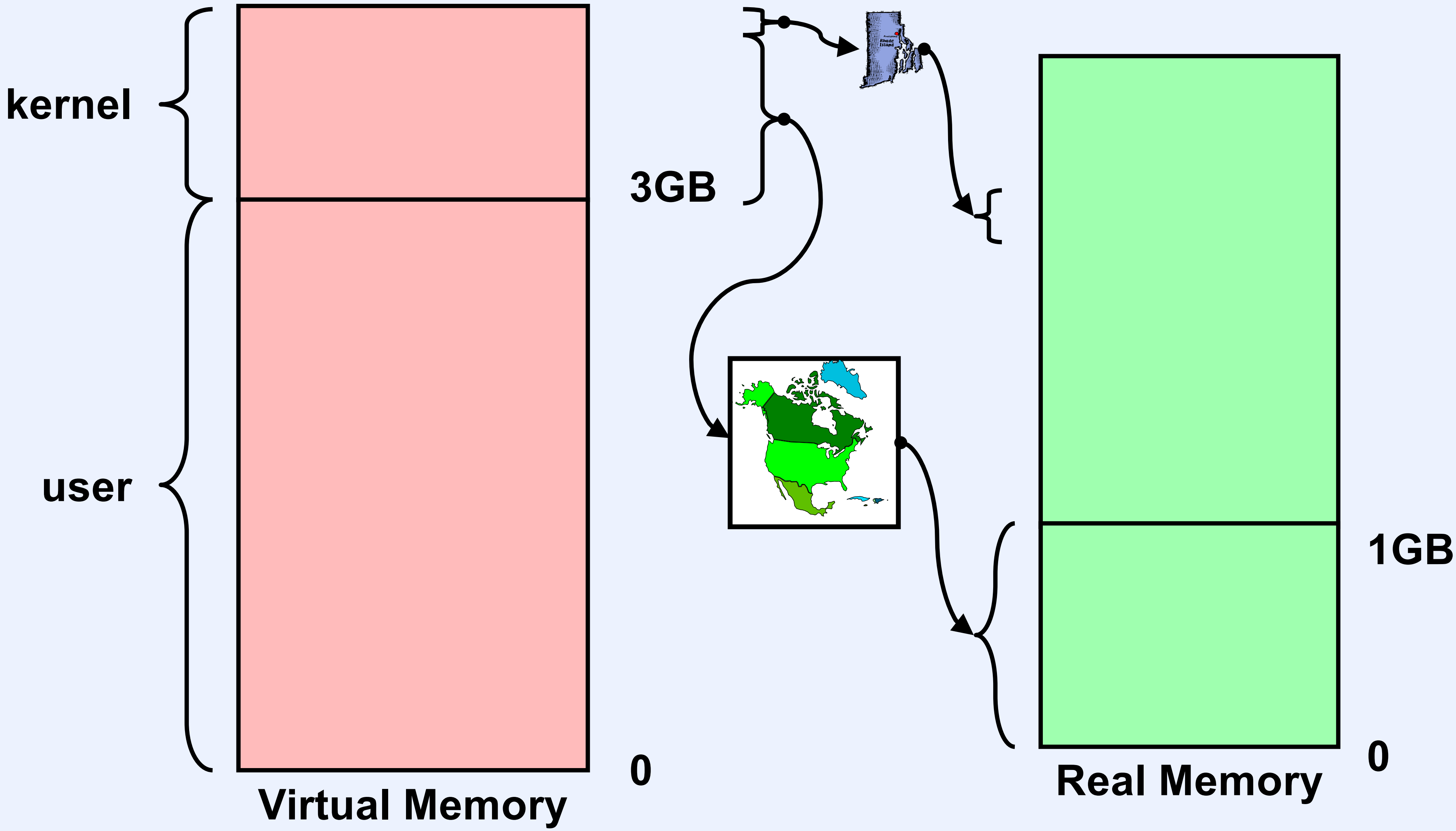
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Linux and Real Memory



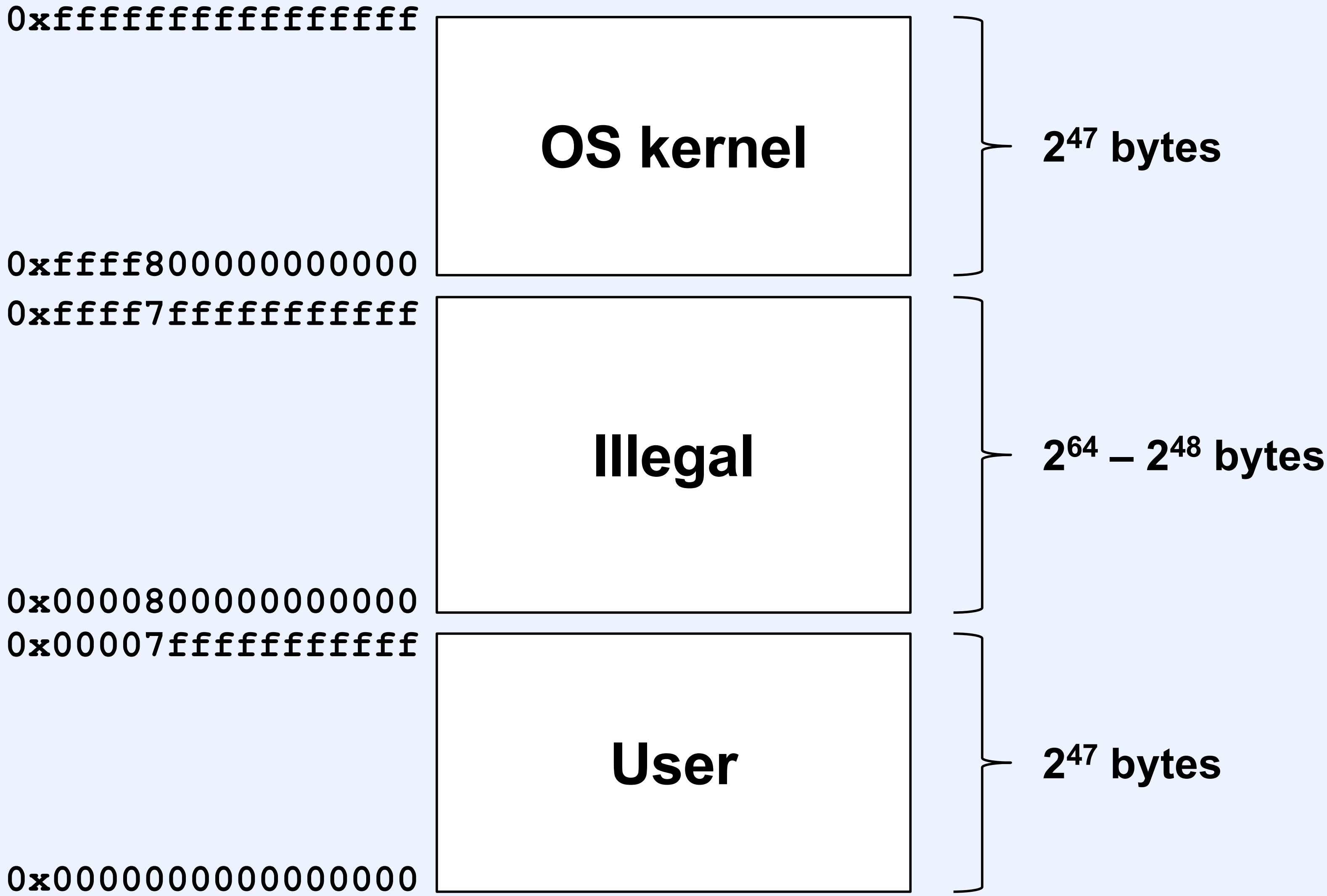
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Lots of Real Memory



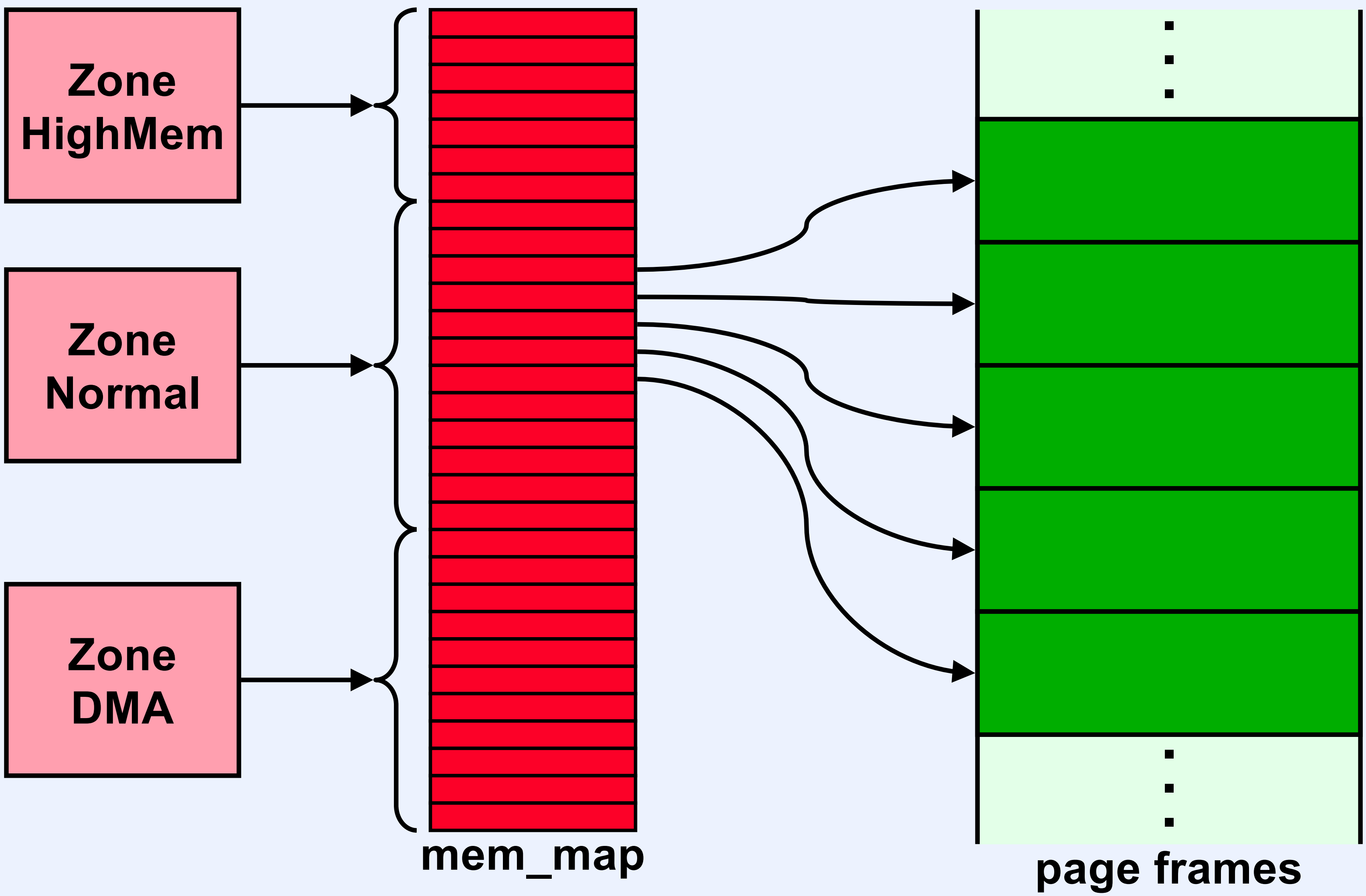
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Address Space



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

Mem_map and Zones



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

Quiz 2

We have a disk whose controller uses 64-bit memory addresses. We'd like to set it up for a read that will transfer 32K bytes. Thus the block will be read into a buffer that occupies eight 4KB pages.

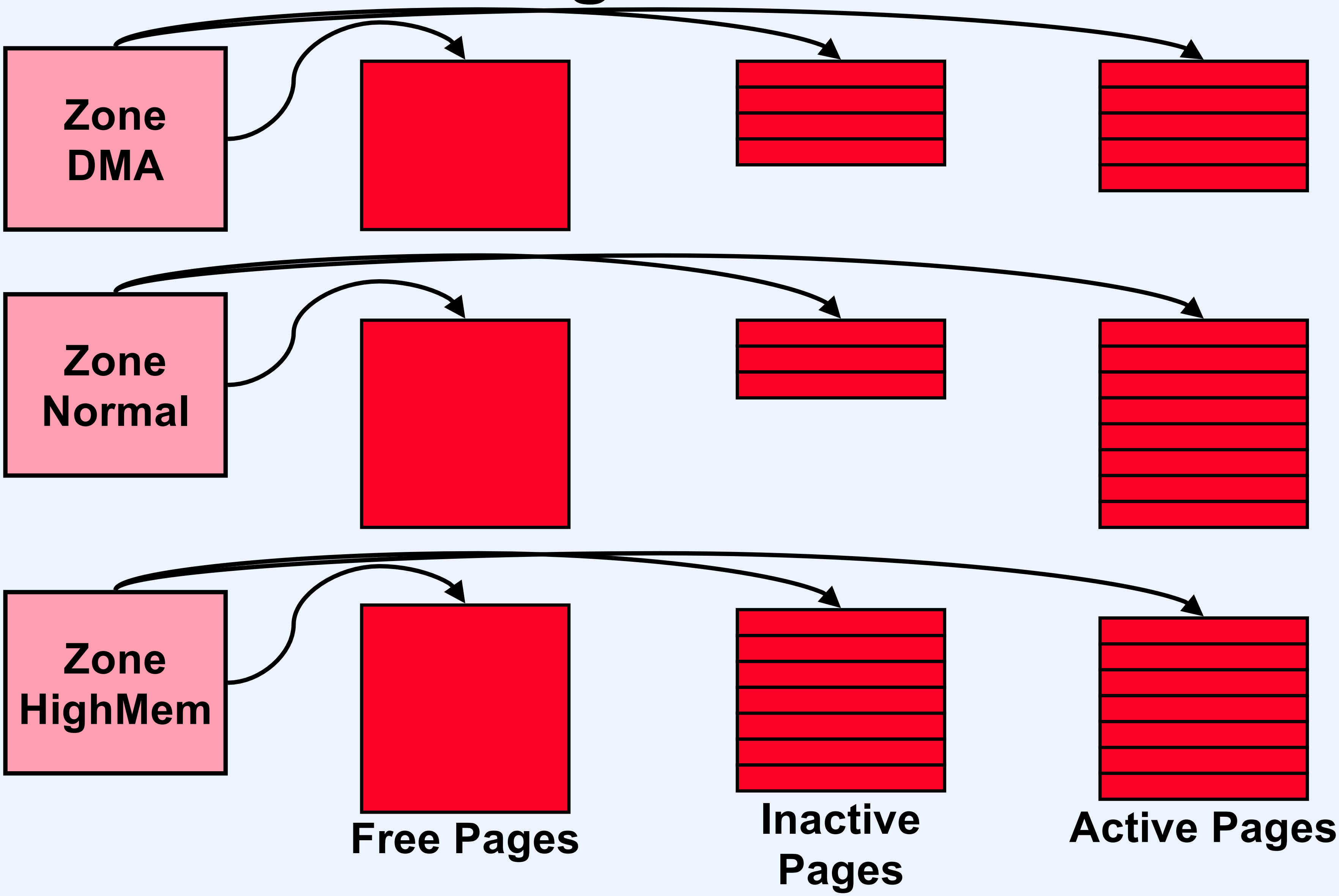
a) Since our system uses virtual memory, the buffer occupies contiguous pages of virtual memory, which may be mapped into non-contiguous page frames of real memory

b) As in a, except that the contiguous pages of virtual memory need not have valid mappings (and thus page faults are generated and handled)

c) The buffer must occupy eight page frames of contiguous real memory

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

Page Lists



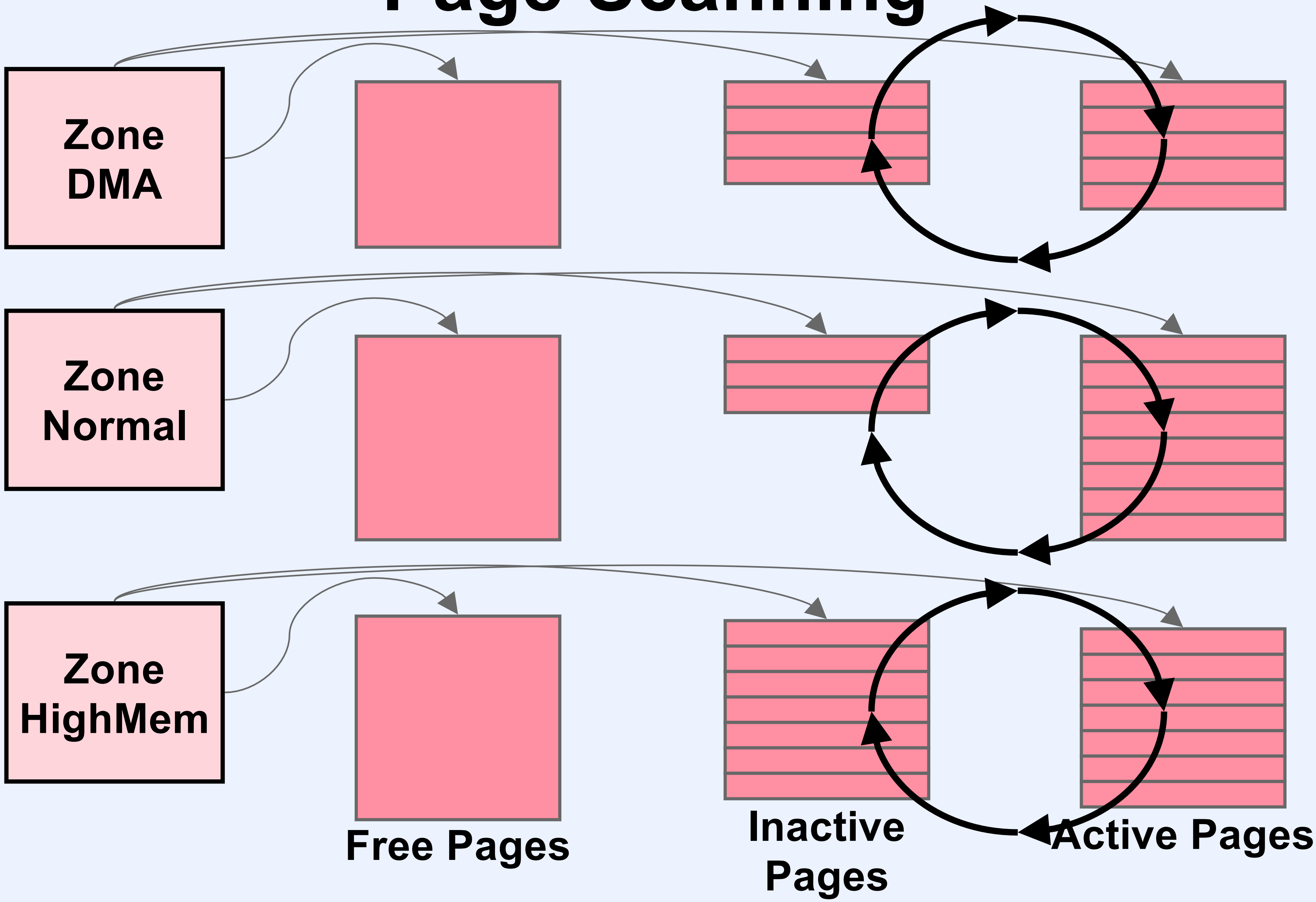
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Page Management

- Replacement
  - two-handed clock algorithm
  - applied to zones in sequence
  - essentially global in scope

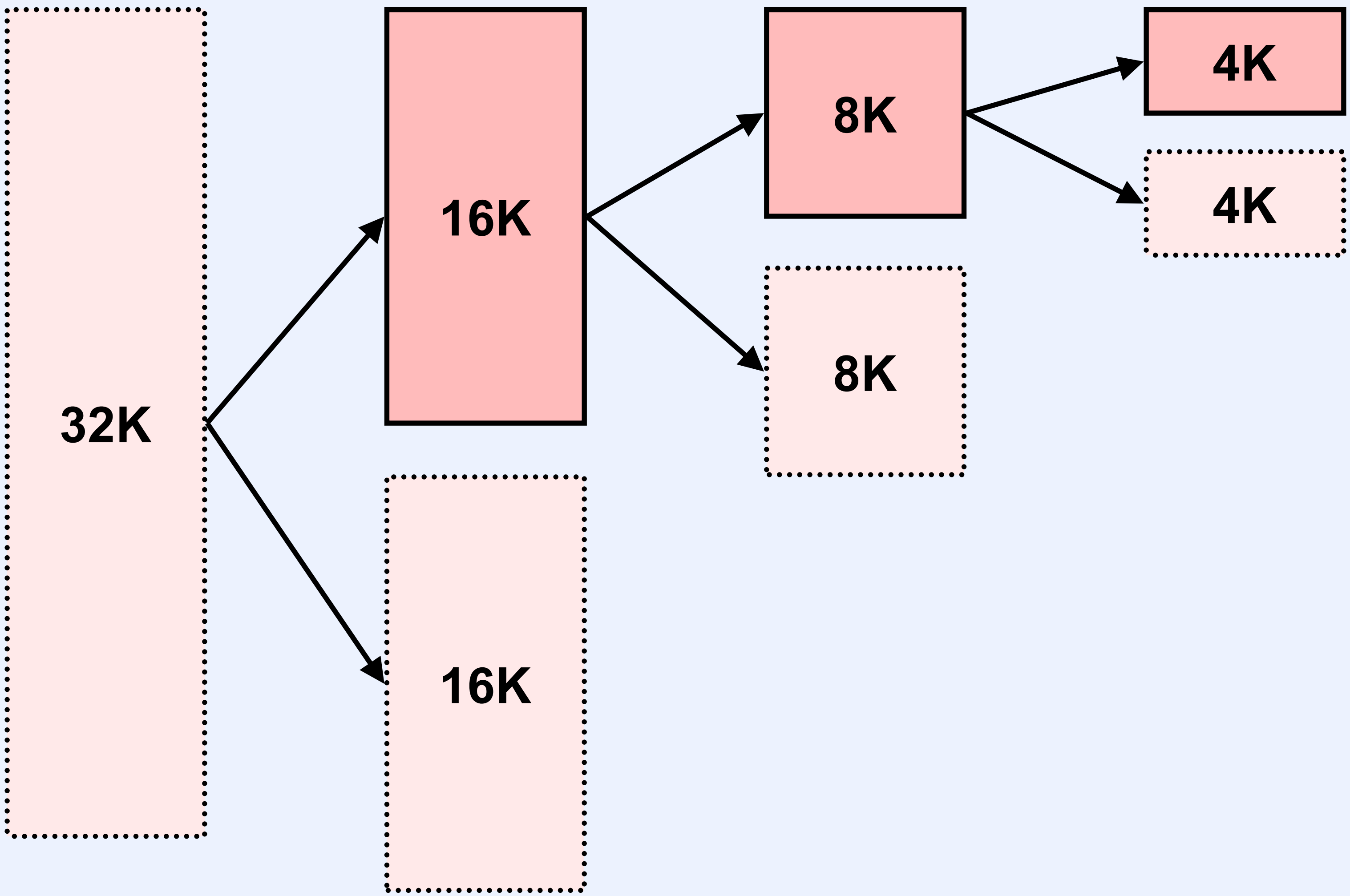
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Page Scanning



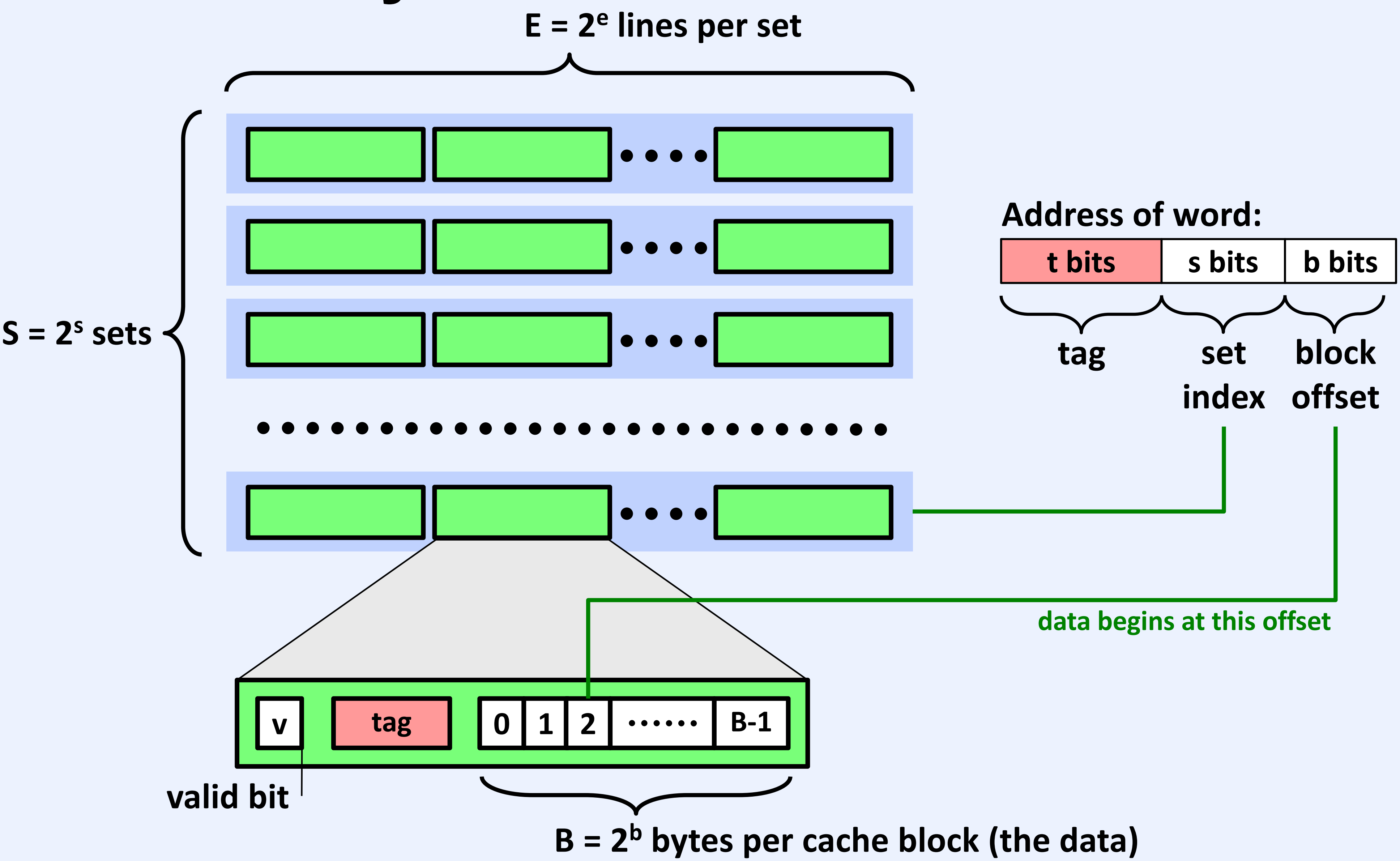
 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Buddy Lists

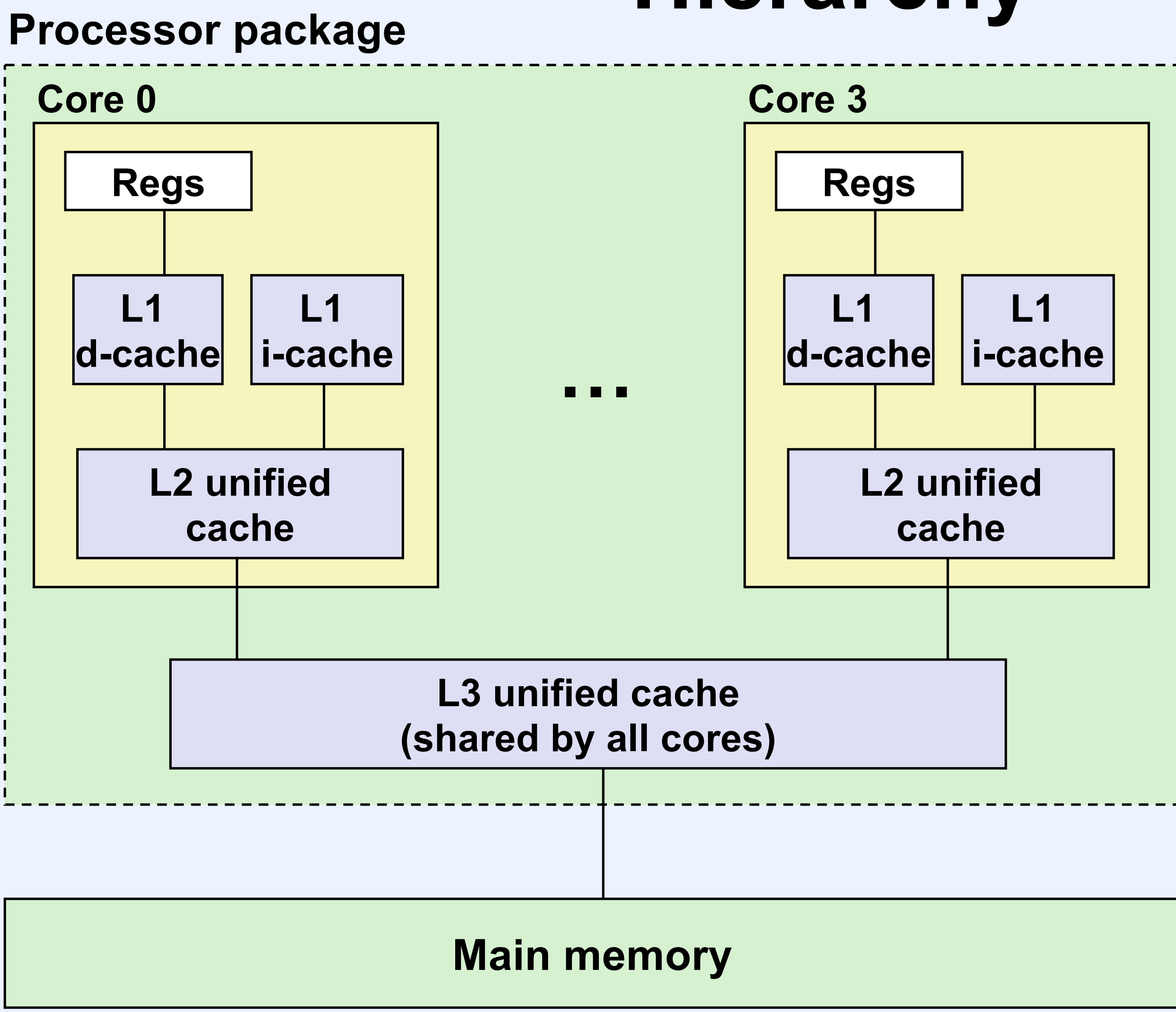


 Copyright © 2025 Thomas W. Doeppner. All rights reserved.

E-Way Set-Associative Cache



Intel Core i5 and i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Block size: 64 bytes for all caches

Quiz 3

You’re designing the algorithm for allocating an often-used and -allocated kernel data structure that fits within a cache line. We’d like to make sure that a number of these data structures can coexist in the hardware caches. Which one of the following would help make this happen (and is doable)?

a) Rounding the size of the data structure up to a power of 2

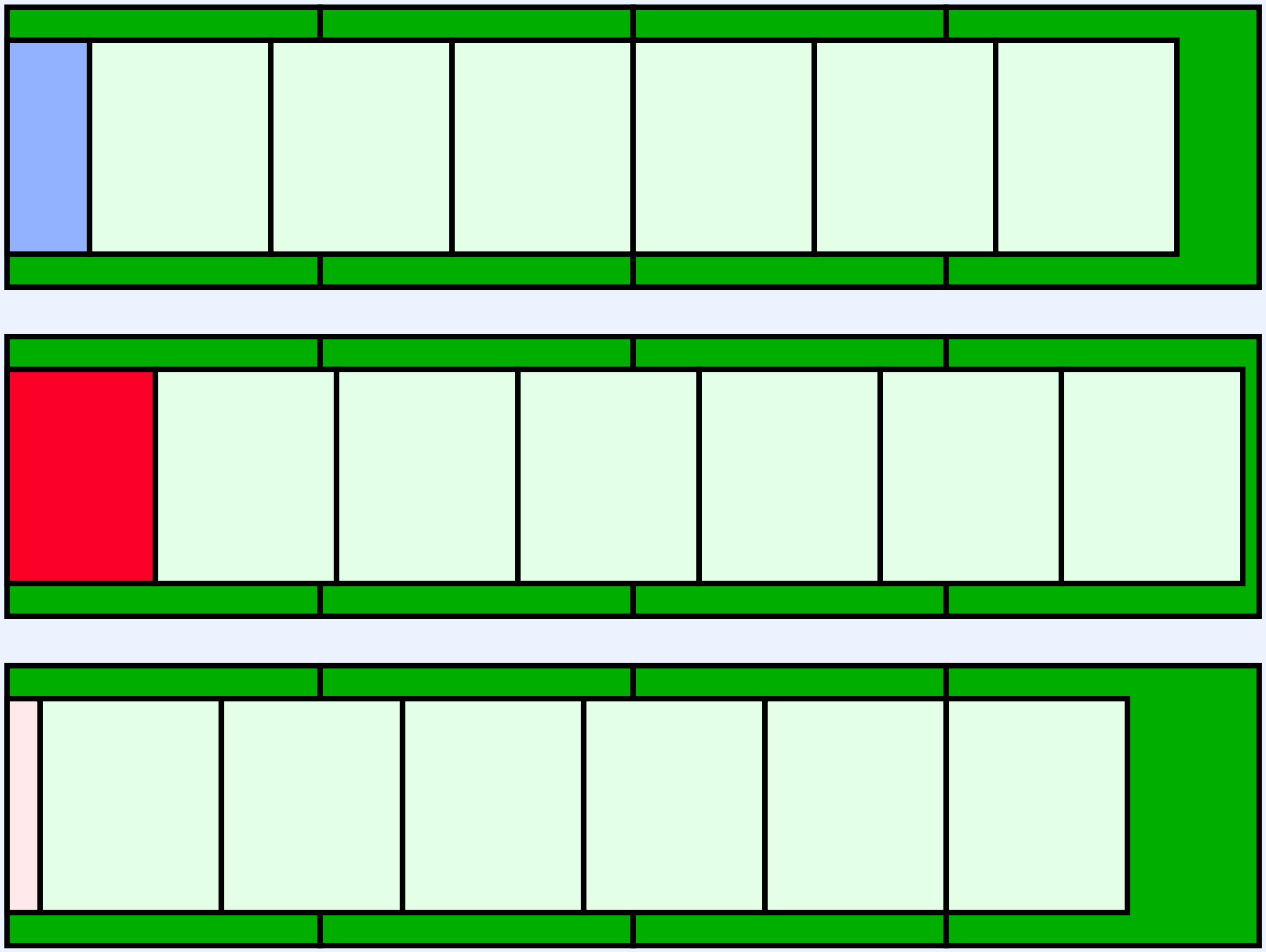
b) Making sure all reside in the same cache set

c) Making sure they are distributed across cache sets

d) Nothing would help

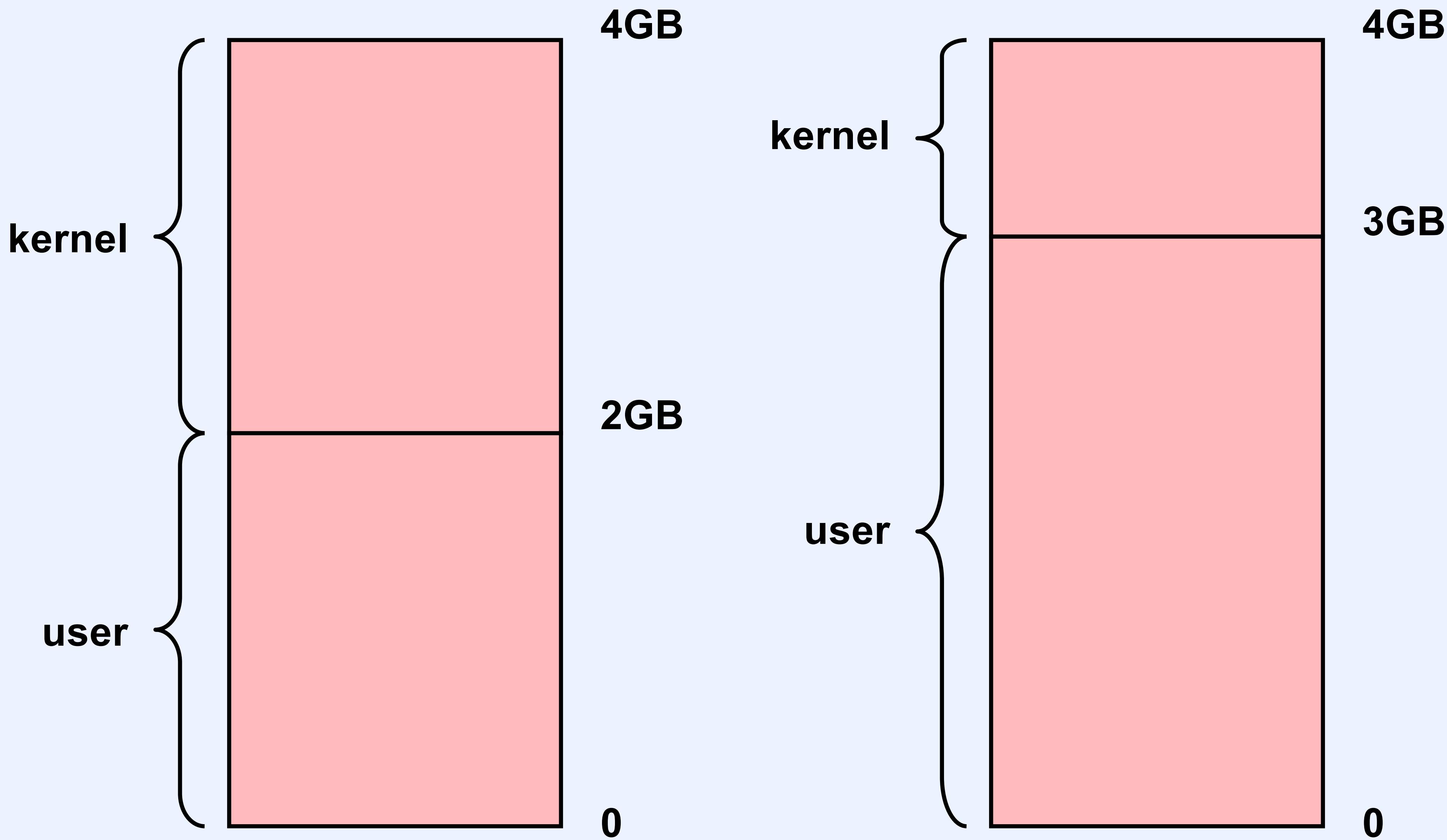
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Slab Allocation



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

Windows x86 Layout



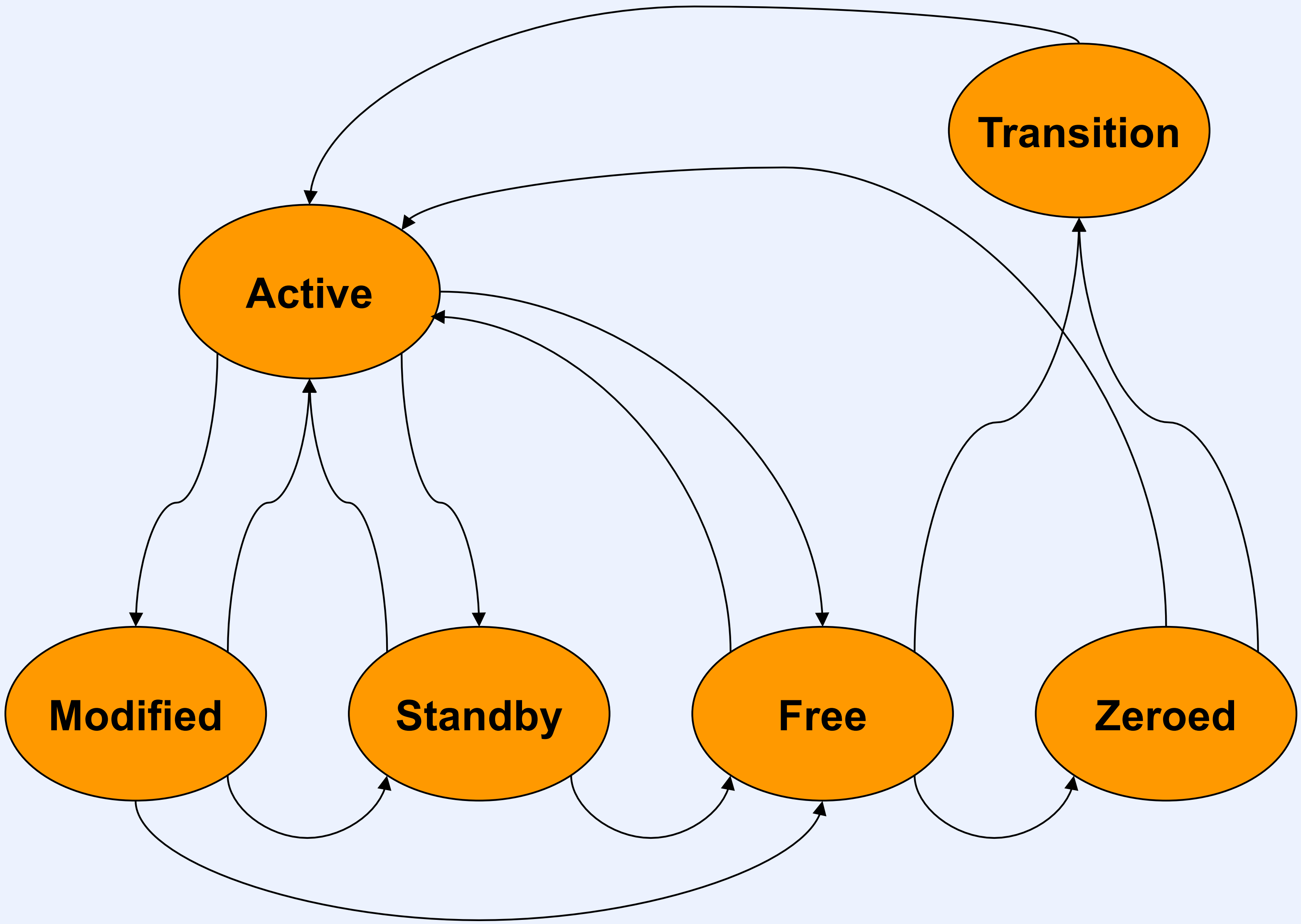
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Windows Paging Strategy

- All processes guaranteed a “working set”
  - lower bound on page frames
- Competition for additional page frames
- “Balance-set” manager thread maintains working sets
  - one-handed clock algorithm
- Swapper thread swaps out idle processes
  - first kernel stacks
  - then working set
- Some of kernel memory is paged
  - page faults are possible

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

Windows Page-Frame States



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

Unix and Virtual Memory: The *fork/exec* Problem

- **Naive implementation:**
  - fork actually makes a copy of the parent's address space for the child
  - child executes a few instructions (setting up file descriptors, etc.)
  - child calls exec
  - result: a lot of time wasted copying the address space, though very little of the copy is actually used

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

Quiz 4

How many pages of virtual memory must be copied from the parent to the child in the following code?

```
if (fork() == 0) {
   close(0);
   dup(open("input_file", O_RDONLY));
   execv("newprog", 0);
}
```

a) 0

b) 1-2

c) 4-8

d) lots

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

vfork

- Don't make a copy of the address space for the child; instead, give the address space to the child
  - the parent is suspended until the child returns it
- The child executes a few instructions, then does an *exec*
  - as part of the *exec*, the address space is handed back to the parent
- Advantages
  - very efficient
- Disadvantages
  - works only if child does an *exec*
  - child shouldn't do anything to the address space

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

Quiz 5

Will the assertion evaluate to true?

```
volatile int A = 6;
…
if (vfork() == 0) {
  A = 7;
  exit(0);
}
assert(A == 7);
…
```

a) it is never executed
b) it definitely won't evaluate to true
c) it will evaluate to true

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

Lazy Evaluation

- Always put things off as long as possible
- If you wait long enough, you might not have to do them

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

A Better *fork*

- Parent and child share the pages comprising their address spaces
  - if either party attempts to modify a page, the modifying process gets a copy of just that page
- Advantages
  - semantically equivalent to the original *fork*
  - usually faster than the original *fork*
- Disadvantages
  - slower than *vfork*

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

Quiz 6

How many pages of virtual memory must be copied from the parent to the child in the following code?

```
if (fork() == 0) {
  close(0);
  dup(open("input_file", O_RDONLY));
  execv("newprog", 0);
}
```

a) 0

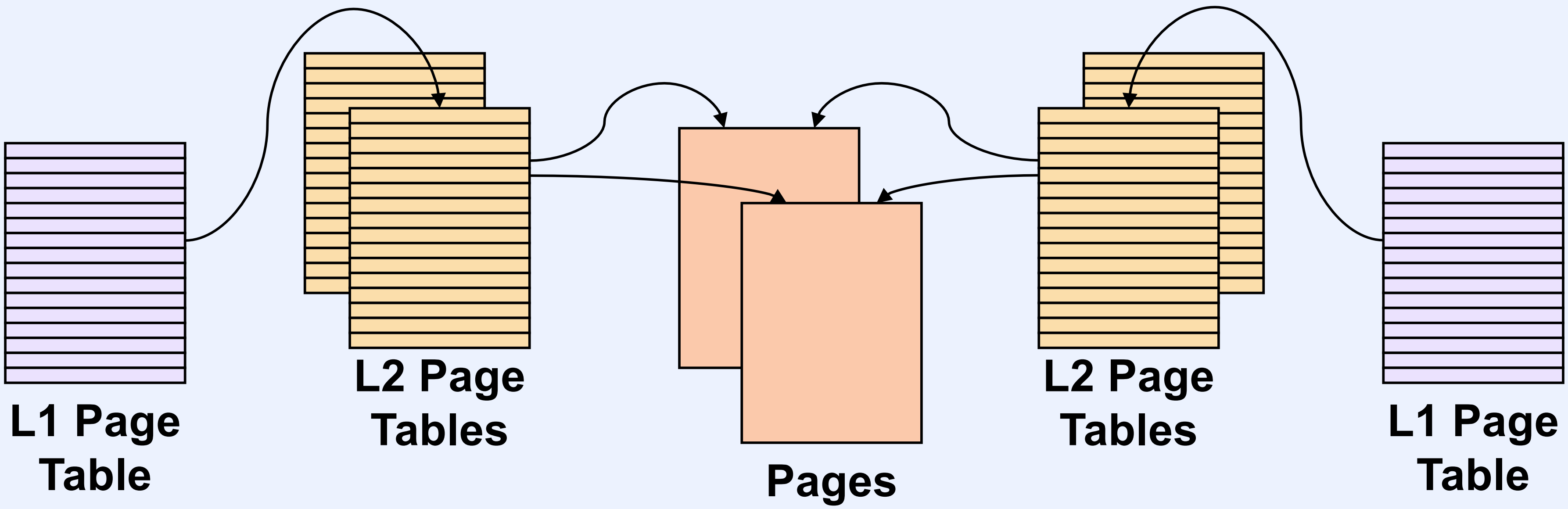
b) 1-2

c) 4-8

d) lots

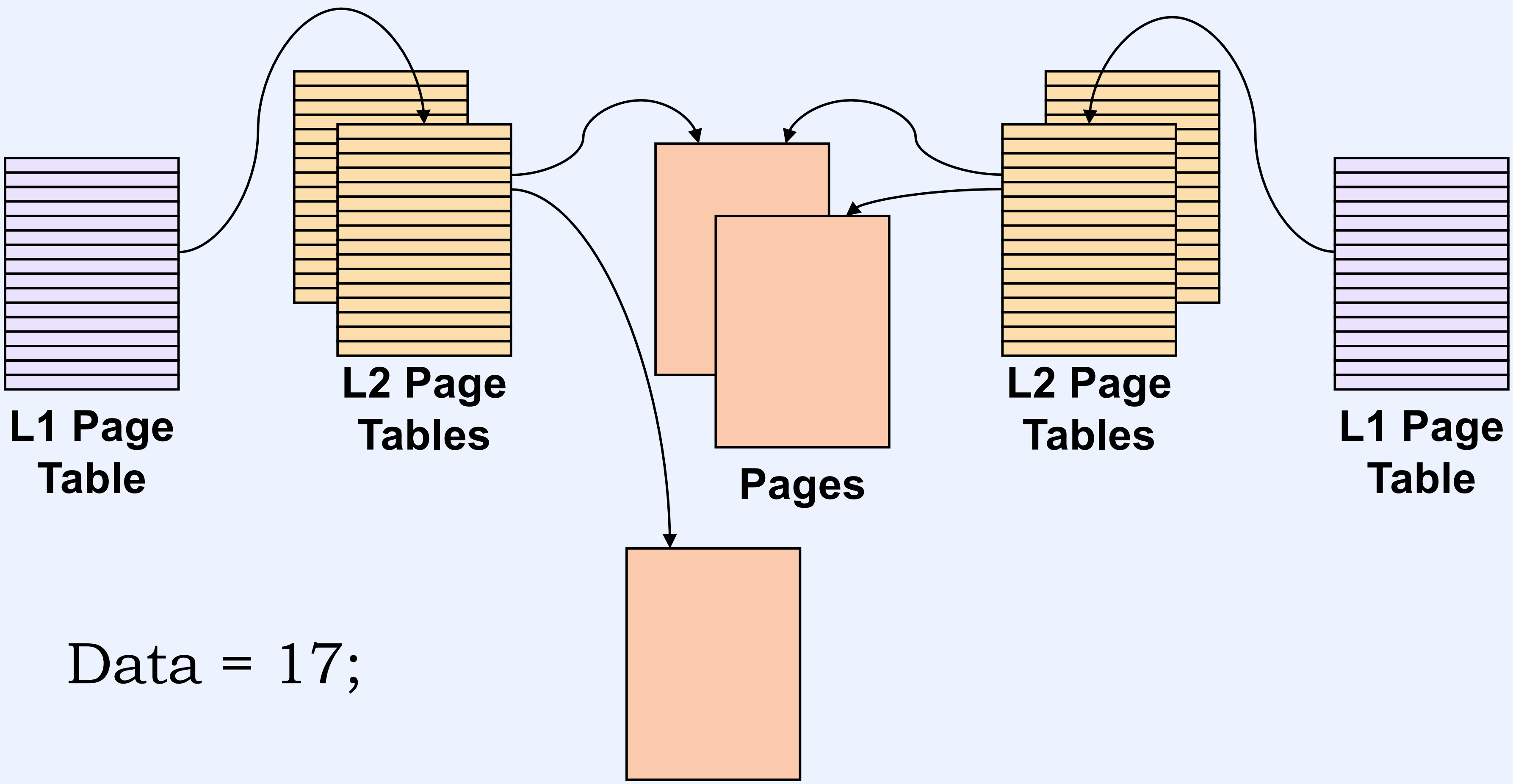
Copyright © 2025 Thomas W. Doeppner. All rights reserved.

Copy on Write (1)



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

Copy on Write (2)



Data = 17;

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

Quiz 7

We have a file that contains one billion 64-bit integers. We are writing a program to read in the file and add up all the integers. Which approach will be fastest:

a) read the file 8 bytes at a time, adding to a running total what is read in

b) read the file 8k bytes at a time, then add each of the integers contained in that block to the running total

c) *mmap* the file into the process's address space, then sum up all the integers in this mapped region of memory

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