Selected VM Algorithms From FreeBSD

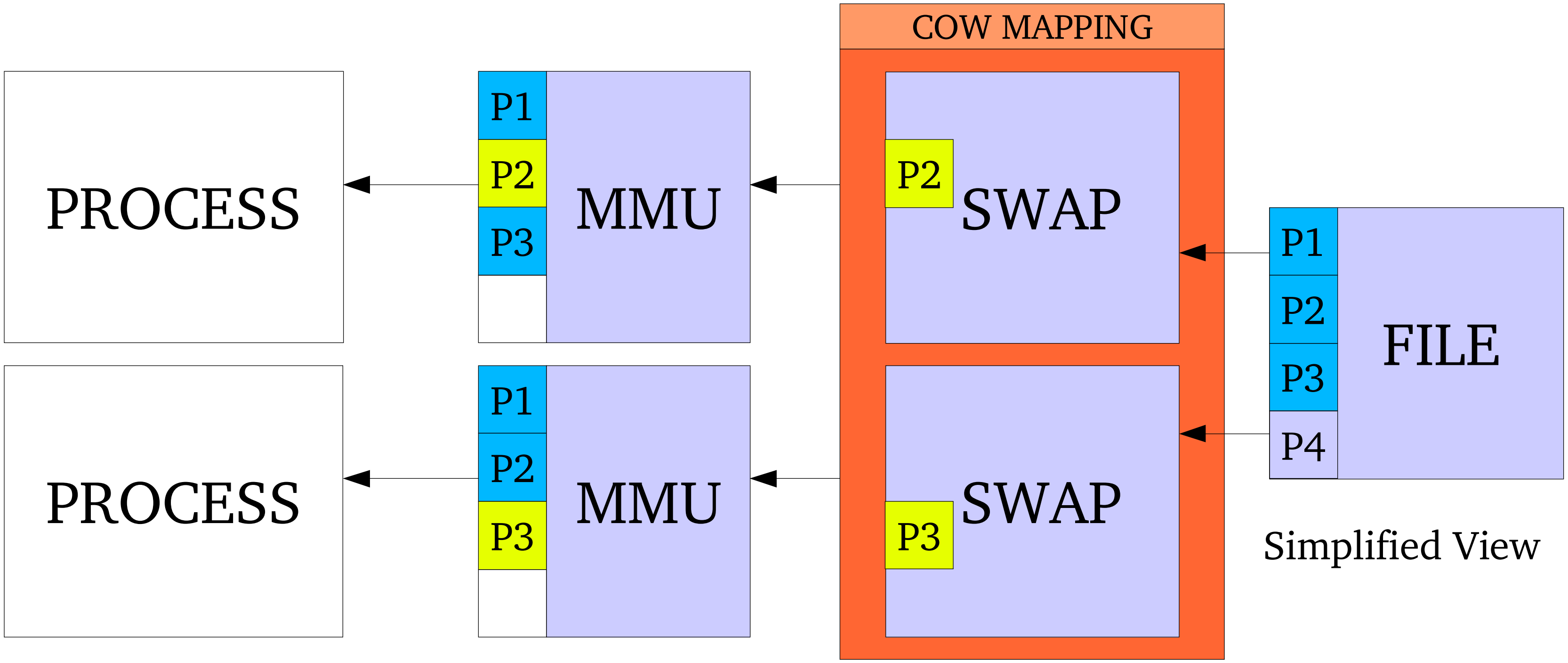
Matthew Dillon
FreeBSD Project

The FreeBSD VM System – What is Memory Anyway?

- Anonymous Memory (Allocations, backed by SWAP)
- File-Backed Memory (Program Binary, mmap())
- SysV Shared Memory
- ZAPHOD's Memory (well, ok, really ZFOD memory = zero-fill-on-demand)
- Device Memory (e.g. Like a Video Frame Buffer)

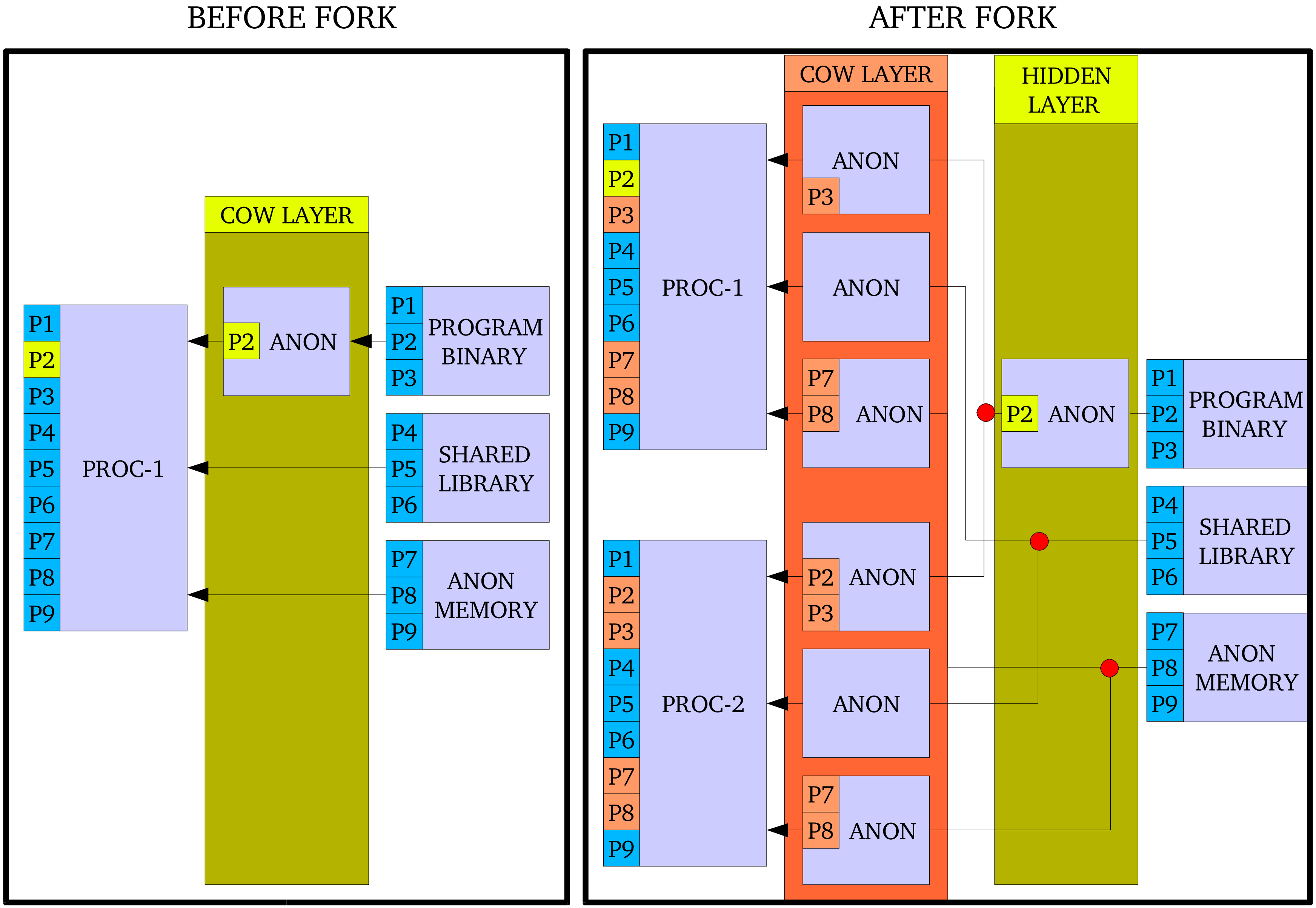
- SHARED mapping verses PRIVATE (copy-on-write) mapping

Memory Structure is Managed by VM Objects

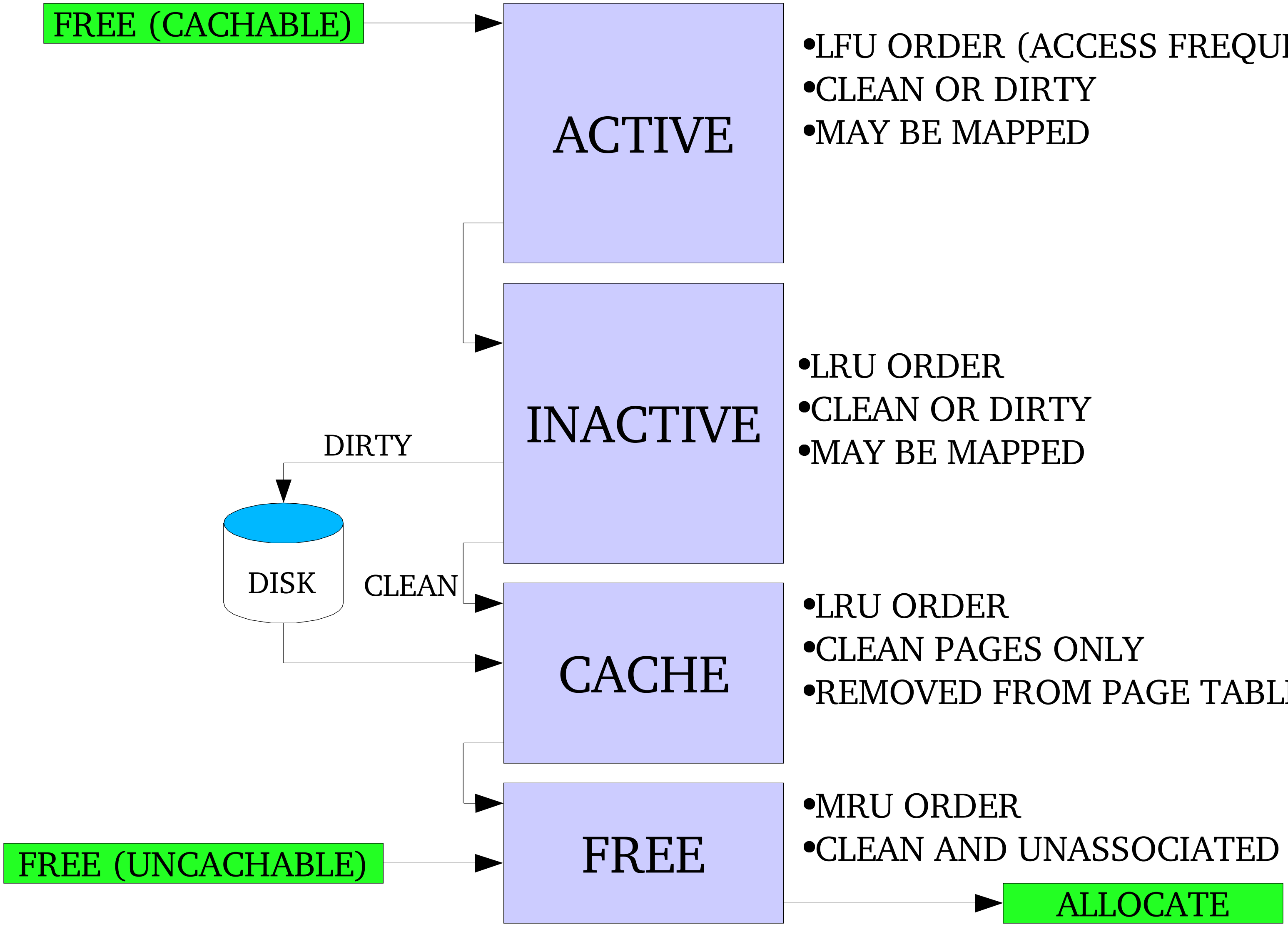


Simplified View

VM Object Stacking can get Complex



Managing Pages of Memory – VM Page Queues



FREE (CACHABLE)

ACTIVE

- LFU ORDER (ACCESS FREQUENCY)
- CLEAN OR DIRTY
- MAY BE MAPPED

INACTIVE

- LRU ORDER
- CLEAN OR DIRTY
- MAY BE MAPPED

DIRTY

DISK

CLEAN

CACHE

- LRU ORDER
- CLEAN PAGES ONLY
- REMOVED FROM PAGE TABLES

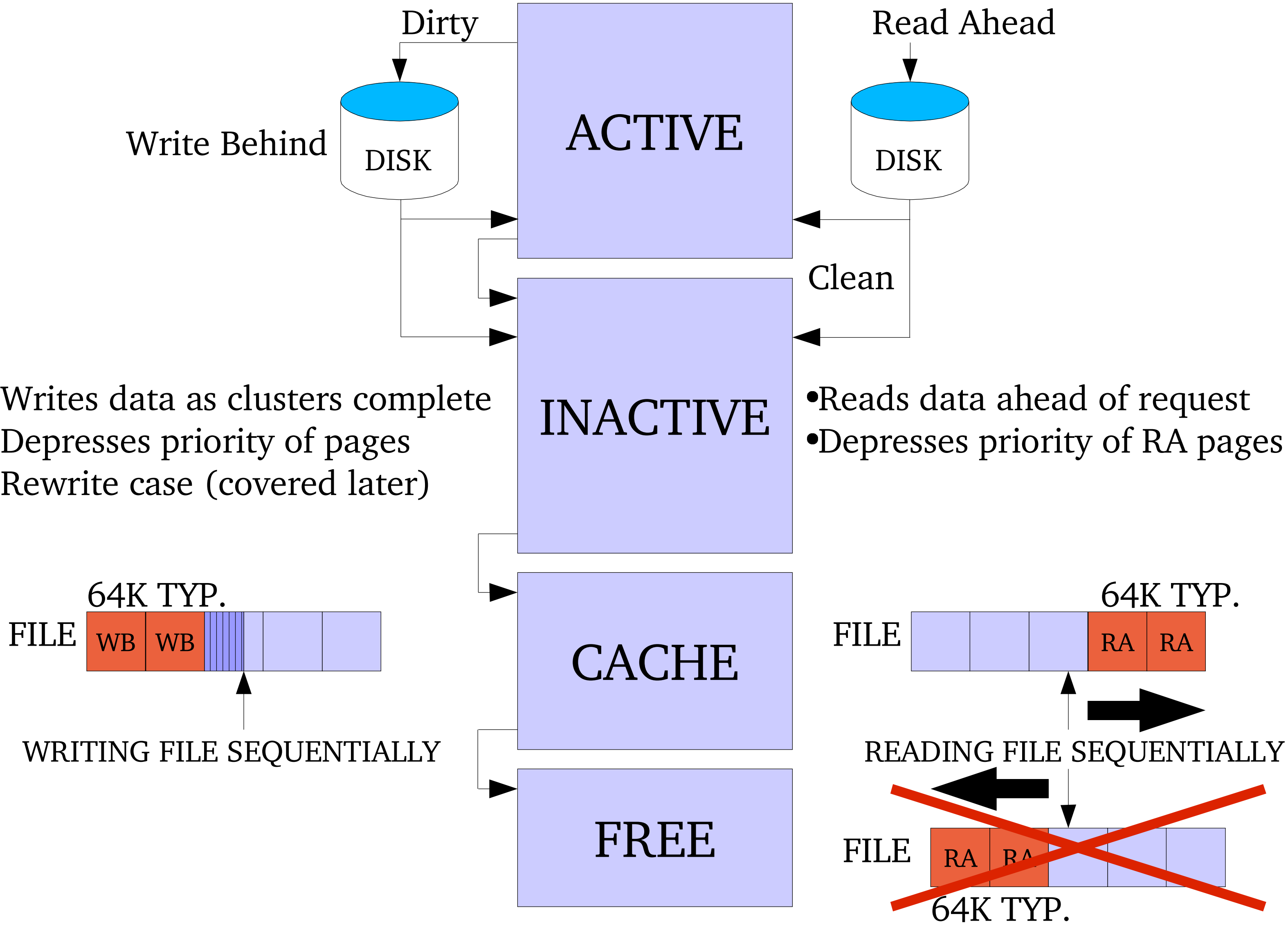
FREE (UNCACHABLE)

FREE

- MRU ORDER
- CLEAN AND UNASSOCIATED

ALLOCATE

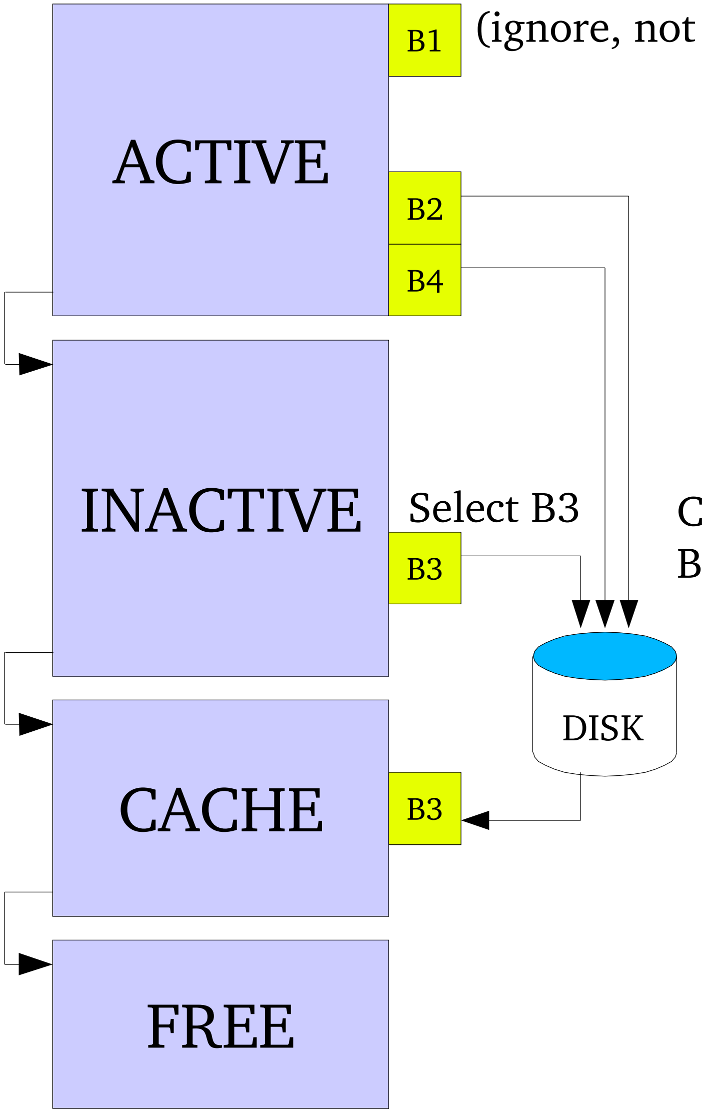
Sequential Heuristics



- •Writes data as clusters complete
- •Depresses priority of pages
- •Rewrite case (covered later)

- •Reads data ahead of request
- •Depresses priority of RA pages

Write Clustering by the Pageout Daemon



(ignore, not close to deactivation)

- Activated when we decide to clean a dirty page
- Locates other nearby dirty pages (logical)
- Must be in a state similar to the original page
- Must also be physically contiguous
- Works when writing to SWAP or to a FILE

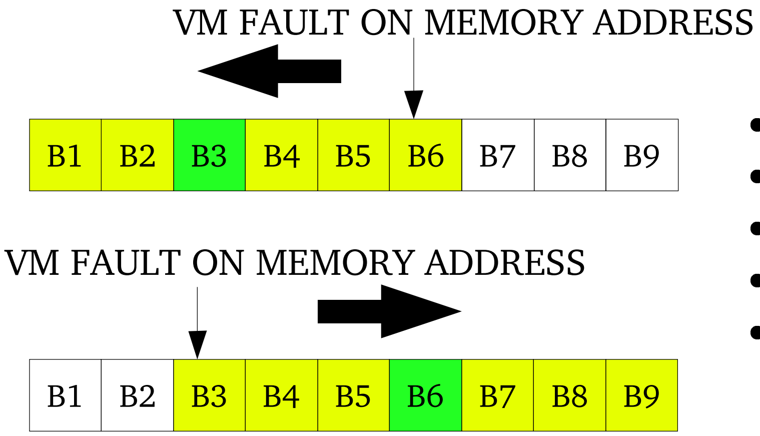
Select B3

Clustering adds
B2 and B4

B1	B2	B3	B4	B5	B6

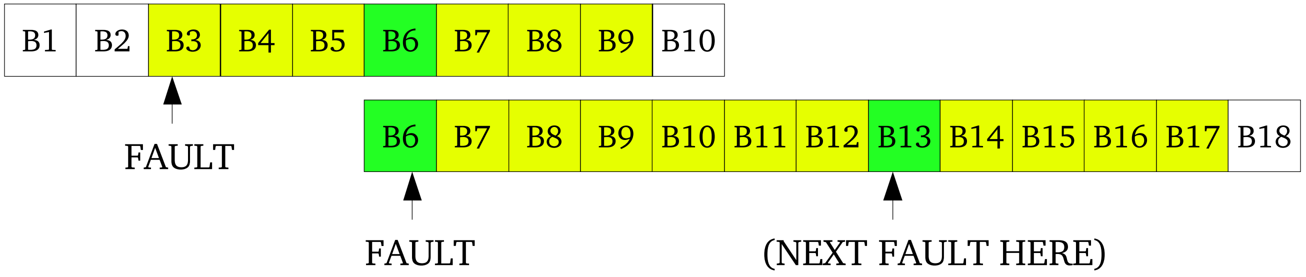
DIRTY PAGES IN VM OBJECT

Read Clustering by the VM Fault handler

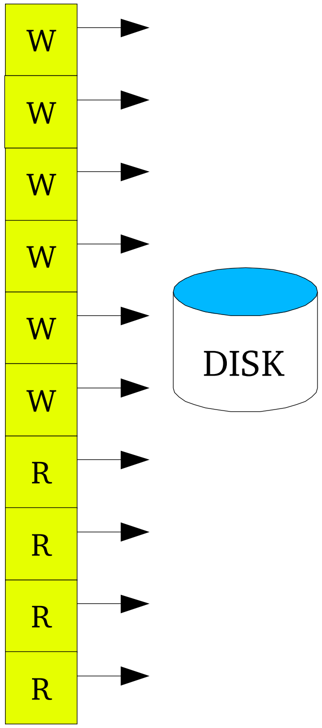

VM FAULT ON MEMORY ADDRESS
B1 B2 B3 B4 B5 B6 B7 B8 B9
VM FAULT ON MEMORY ADDRESS
B1 B2 B3 B4 B5 B6 B7 B8 B9

- •Uses an unmapped page to trigger read-ahead
- •Reads data ahead of request
- •Detects if memory is being accessed backwards
- •Depresses priority of RA pages
- •Works when reading from a FILE or SWAP

- •B3-B9 are read
- •B6 is unmapped from the process (but remains in the VM Page cache)
- •If a fault is taken on B6, an asynchronous read is issued for B10-B17, B13 is unmapped
- •If a fault is taken on B6, the VM fault handler is able to return immediately
- •Greatly improved performance, processing overhead of program overlaps I/O
- •Intentional faults (e.g. B6) are used to detect direction


B1 B2 B3 B4 B5 B6 B7 B8 B9 B10
FAULT
B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 B16 B17 B18
FAULT
(NEXT FAULT HERE)

Running Write I/O Limit

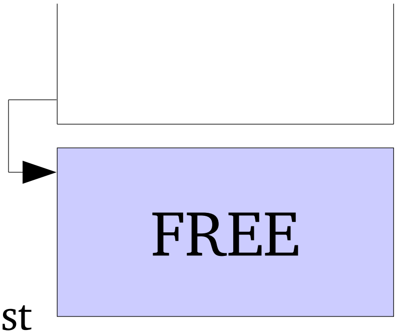


- Writes are almost always asynchronous (pageout, update)
- Reads are typically synchronous or semi-synchronous
- Tags create an issue with traditional I/O queueing and sorting
- IDE W/write-caching turned on has a similar problem
- Pace disk writes to reduce read latency
- Reserve most tags for read operations (not yet in FreeBSD)
- Priority Scheduler for I/O (not yet in FreeBSD)

```
vfs.lorunningspace=524288
vfs.hirunningspace=1048576
```

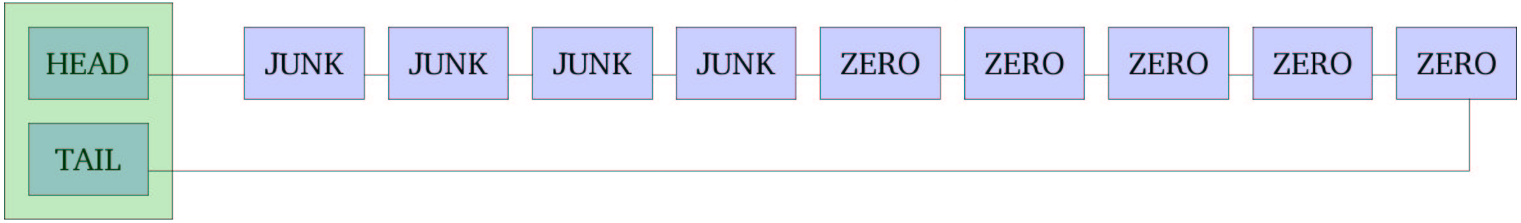
Optimizing Zero-Fill Faults

- Uses idle CPU cycles and hysteresis to pre-zero free pages
- Uses Most-Recently-Used ordering for VM page free queue
- Pulls pages off the head or tail based on need for a cleared page
- Zeros the page manually if no cleared pages available
- Some kernel facilities are able to return cleared pages to the free list



FREE

FREE PAGE LIST (SIMPLIFIED)



HEAD

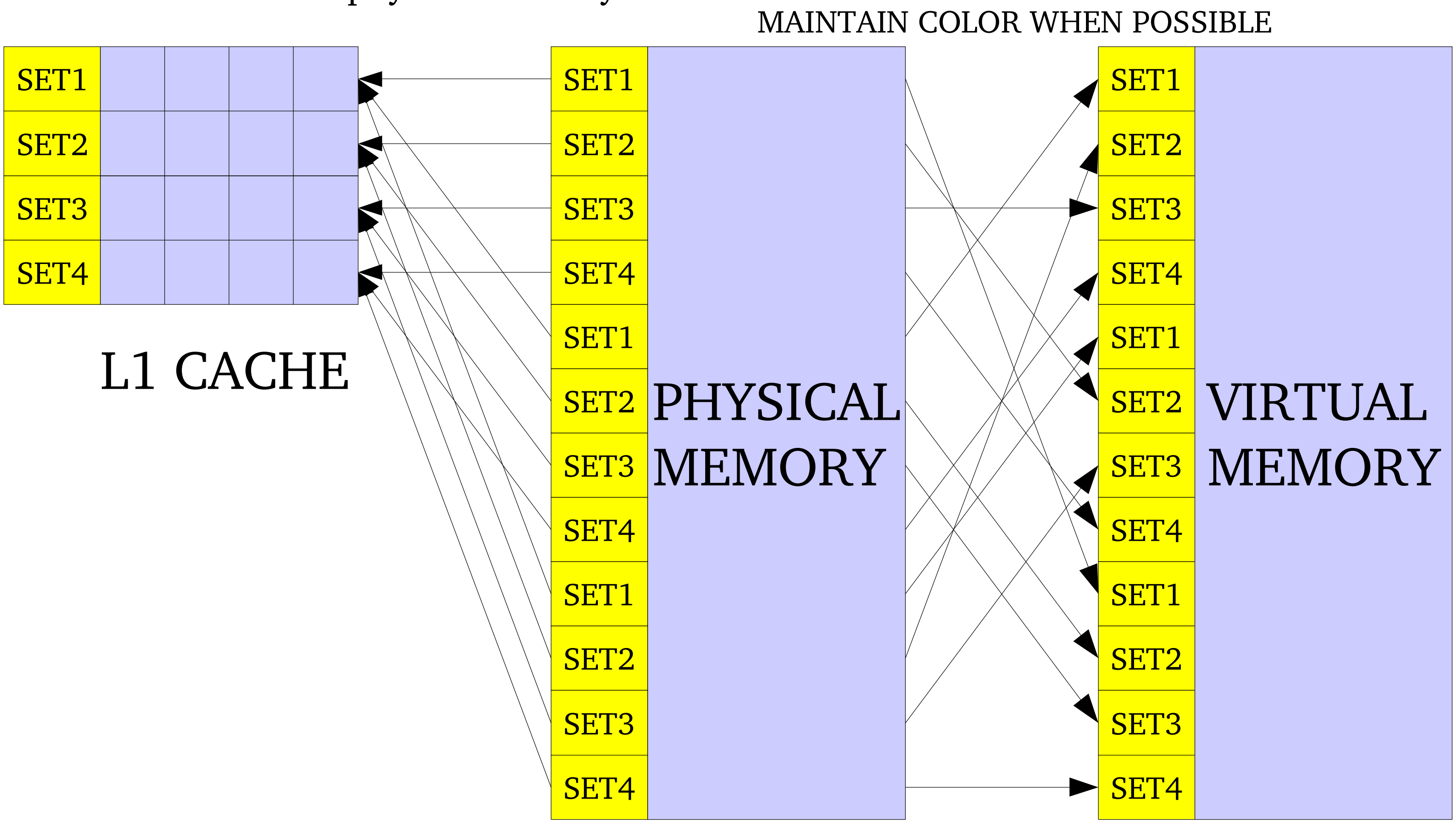
TAIL

JUNK JUNK JUNK JUNK ZERO ZERO ZERO ZERO ZERO

- Actually multiple lists, indexed by cache color (next slide)

Cache Color

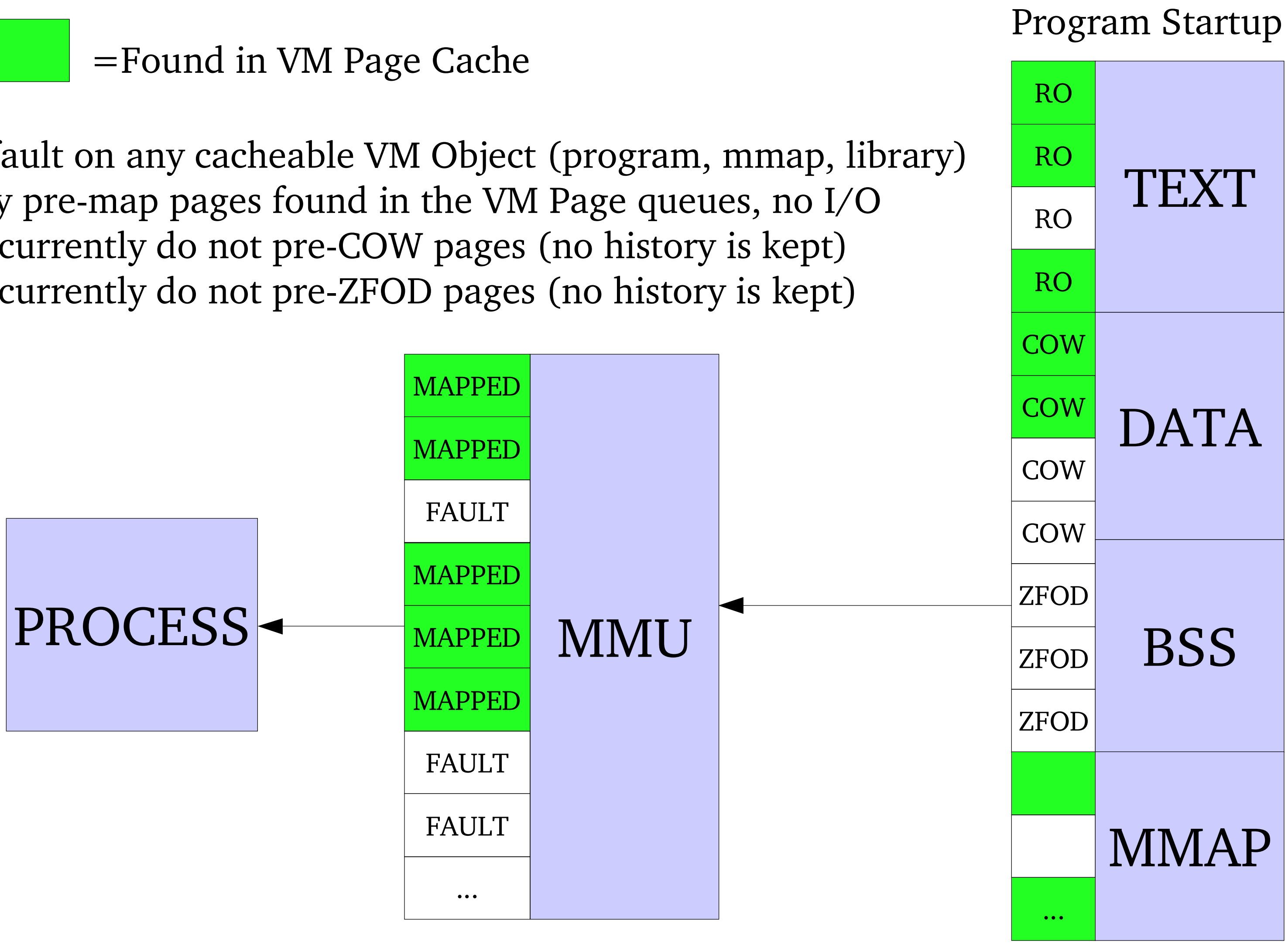
- Make Linear addresses of VM have the same cache behavior as linear addresses of physical memory.



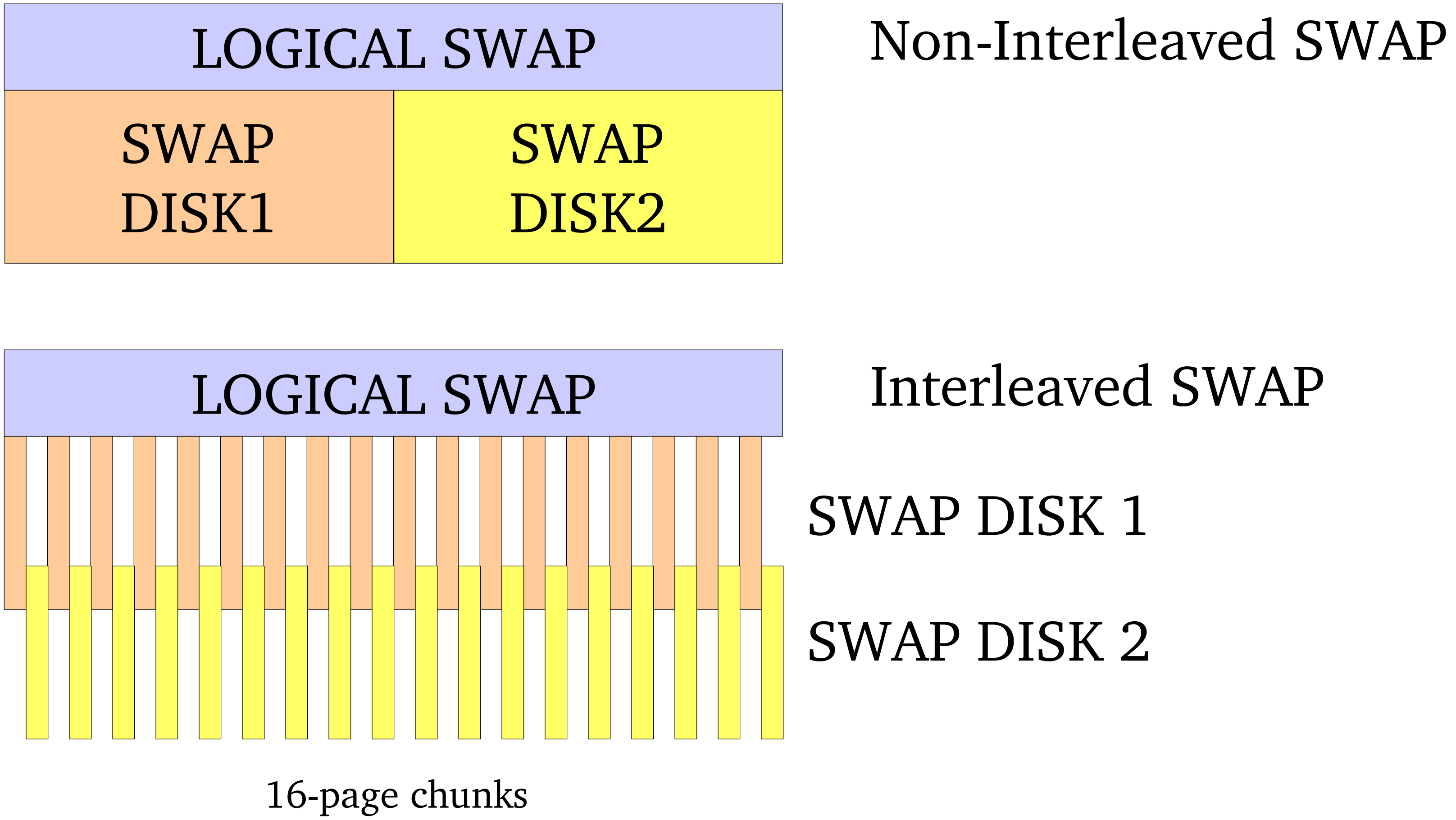
Prefaulting Pages to Improve Performance

🟩 =Found in VM Page Cache

- •Prefault on any cacheable VM Object (program, mmap, library)
- •Only pre-map pages found in the VM Page queues, no I/O
- •We currently do not pre-COW pages (no history is kept)
- •We currently do not pre-ZFOD pages (no history is kept)



Interleaved SWAP



LOGICAL SWAP

SWAP DISK1

SWAP DISK2

Non-Interleaved SWAP

LOGICAL SWAP

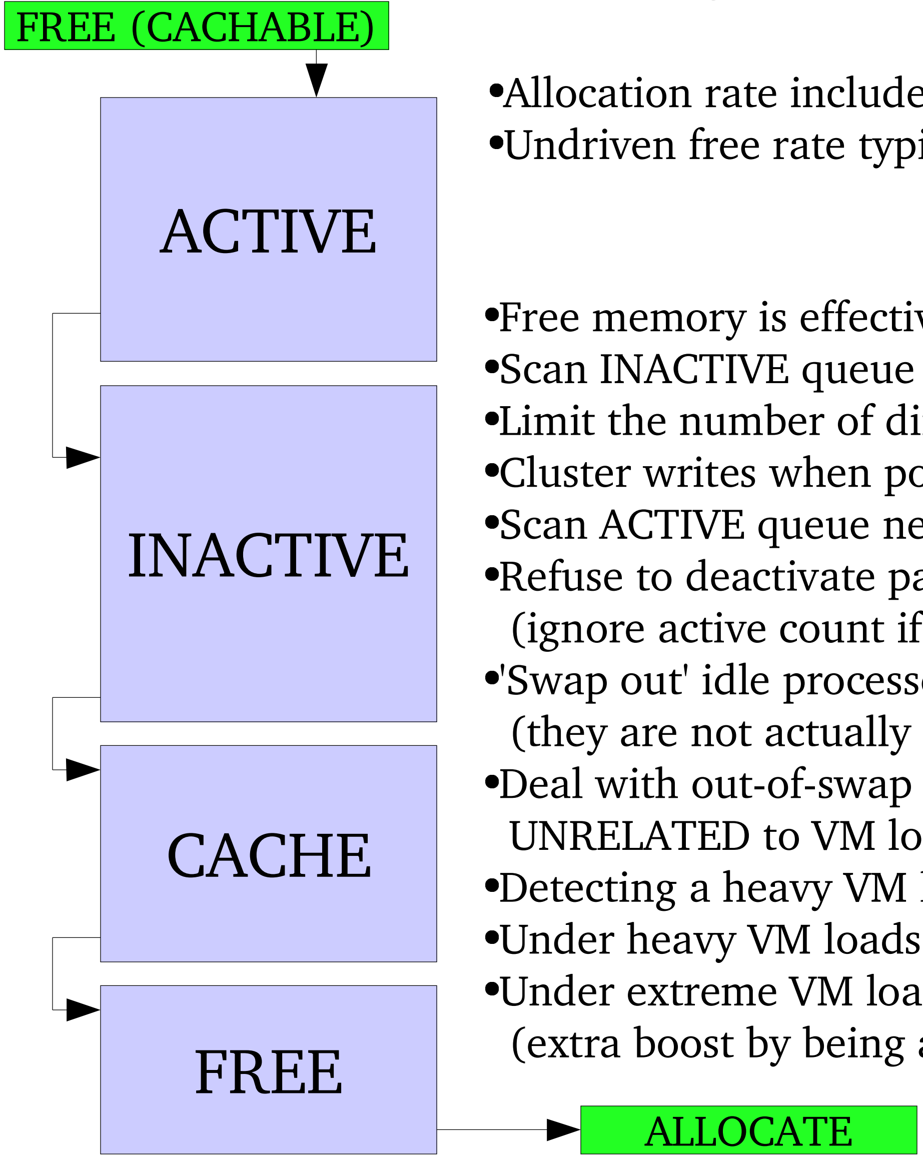
Interleaved SWAP

SWAP DISK 1

SWAP DISK 2

16-page chunks

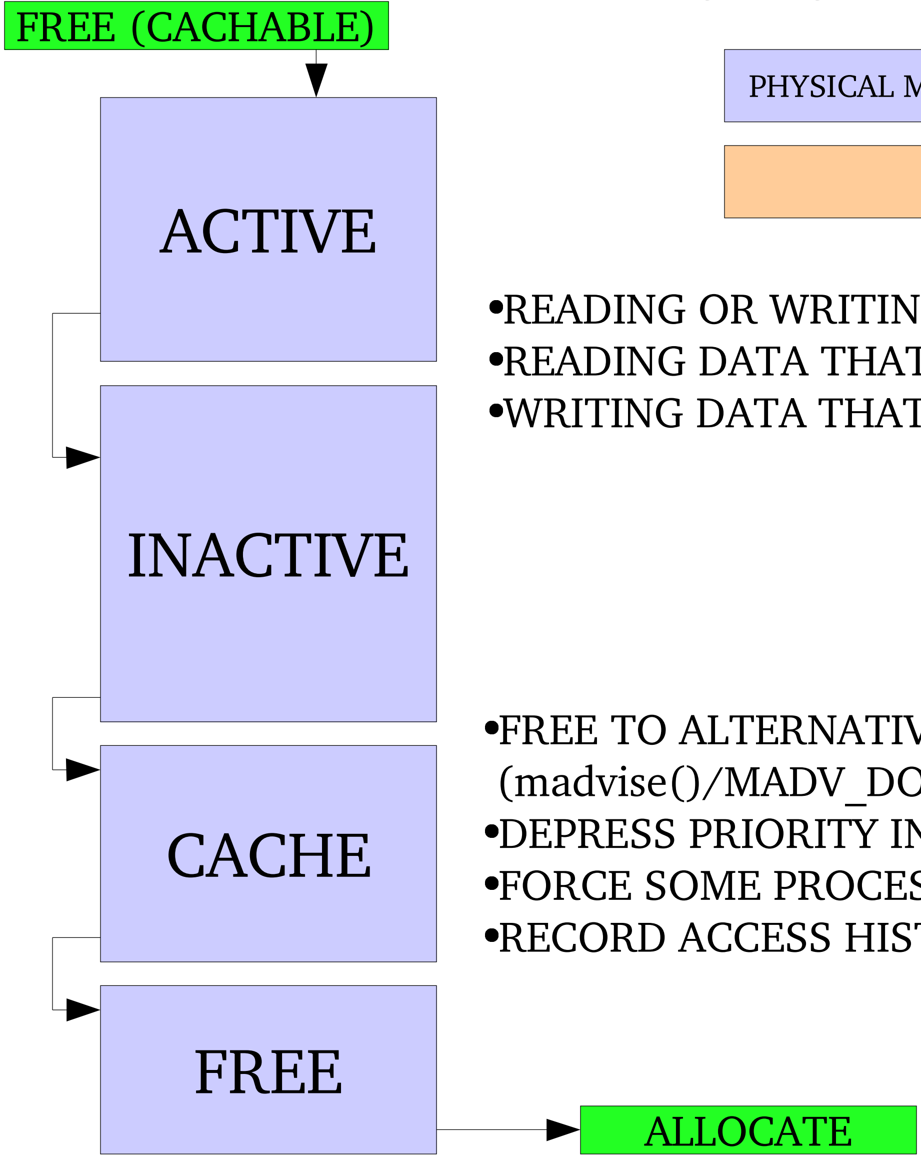
Scaling to the VM Load

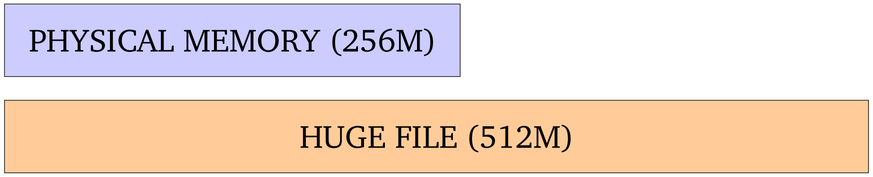


- Allocation rate includes pageins, but not a good indicator of load
- Undriven free rate typically due to program exits

- Free memory is effectively CACHE+FREE
- Scan INACTIVE queue first, INACTIVE->CACHE
- Limit the number of dirty pages Laundered
- Cluster writes when possible
- Scan ACTIVE queue next, deactivating pages, ACTIVE->INACTIVE
- Refuse to deactivate pages with a non-zero active count (ignore active count if object not referenced, revert to LRU)
- 'Swap out' idle processes only by depressing their paging priority (they are not actually swapped out synchronously)
- Deal with out-of-swap situations. Out-of-swap situations are UNRELATED to VM load.
- Detecting a heavy VM load: check CACHE+FREE at end of pass
- Under heavy VM loads enforce 'memoryuse' resource limit
- Under extreme VM loads, force runnable processes to go idle (extra boost by being able to free pagetable pages)

Where Caching Algorithms Break Down



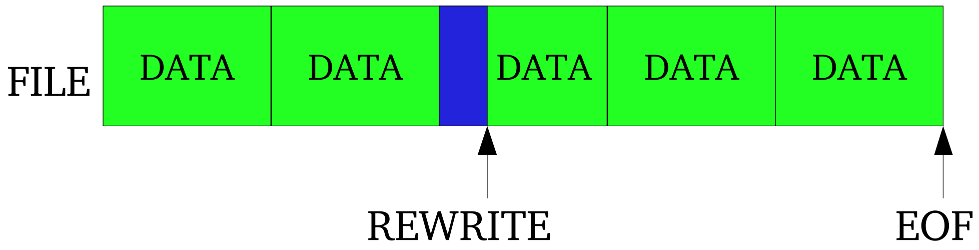


- READING OR WRITING A HUGE FILE (> PHYSICAL MEMORY)
- READING DATA THAT WILL NOT BE READ AGAIN SOON
- WRITING DATA THAT WILL NOT BE READ AGAIN SOON

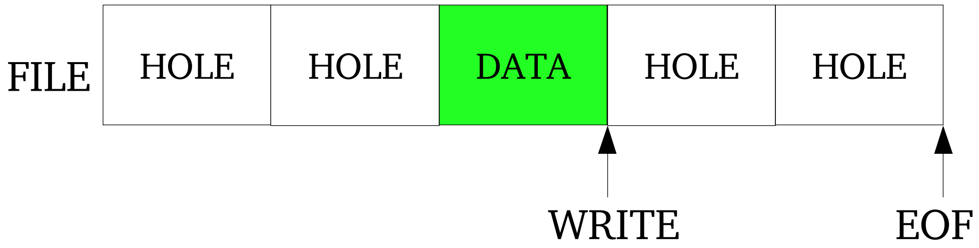
2000 PROCESSES (OOPS!)

- FREE TO ALTERNATIVE LOCATION IN VM PAGE QUEUES (madvise()/MADV_DONTNEED)
- DEPRESS PRIORITY IN ACTIVE QUEUE (DEPRESS-BEHIND)
- FORCE SOME PROCESSES TO GO IDLE FOR A PERIOD OF TIME
- RECORD ACCESS HISTORY OF OBJECT

Where Caching Algorithms Break Down



- Small-block file rewrite case (mismatched filesystem block size)
- Sequential Heuristic can help



- Filesystem fragmentation messes up read and write clustering
- Always write out the 'file' representing a new virtual disk