Understanding Linux Memory Management
SHARE 102 Session 9241

Dr. Ulrich Weigand
Linux on zSeries Development, IBM Lab Böblingen
Ulrich.Weigand@de.ibm.com

Linux Memory Management - A Mystery?

- What does this mean:

```
$ free
                total       used       free     shared    buffers     cached
Mem:           512832     473256      39576          0      52300     236776
-/+ buffers/cache:        184180     328652
Swap:          524280        912     523368
```

- Or this:

```
$ cat /proc/meminfo
MemTotal:        512832 kB                    Dirty:                28 kB
MemFree:          39512 kB                    Writeback:             0 kB
Buffers:          52308 kB                    Mapped:             5492 kB
Cached:          236768 kB                    Slab:             158608 kB
SwapCached:         532 kB                    Committed_AS:       7656 kB
Active:          246328 kB                    PageTables:          208 kB
Inactive:         61920 kB                    VmallocTotal:    1564671 kB
HighTotal:            0 kB                    VmallocUsed:         724 kB
HighFree:             0 kB                    VmallocChunk:    1563947 kB
LowTotal:        512832 kB
LowFree:          39512 kB
SwapTotal:       524280 kB
SwapFree:        523368 kB
```

Agenda

- Physical Memory
- Dynamic Address Translation
- Process Address Spaces and Page Cache
- Kernel Memory Allocators
- Page Replacement and Swapping
- Virtualization Considerations

Physical Memory

- Basic allocation unit: Page (4096 bytes)
  - Use of each page described by 'struct page'
    - Allocation status, backing store, LRU lists, dirty flag, ...
  - Pages aggregated into memory zones
  - Zones may be aggregated into nodes (NUMA systems)
- Boot process
  - Kernel loaded at bottom of memory
  - Determine size of memory, create 'struct page' array
  - Kernel pages and boot memory marked as 'reserved'
  - All other pages form the master pool for page allocation

Memory Zones

GPF_HIGHMEM
("High memory zone")

Memory not directly addressable from kernel space

- On Intel machines all > 4/2/1 GB
- On zSeries empty

GPF_NORMAL
("Normal zone")

Memory generally usable by the kernel, except possibly for I/O

- On 31-bit zSeries empty
- On 64-bit zSeries all > 2 GB

GPF_DMA
("DMA zone")

Memory usable without restrictions

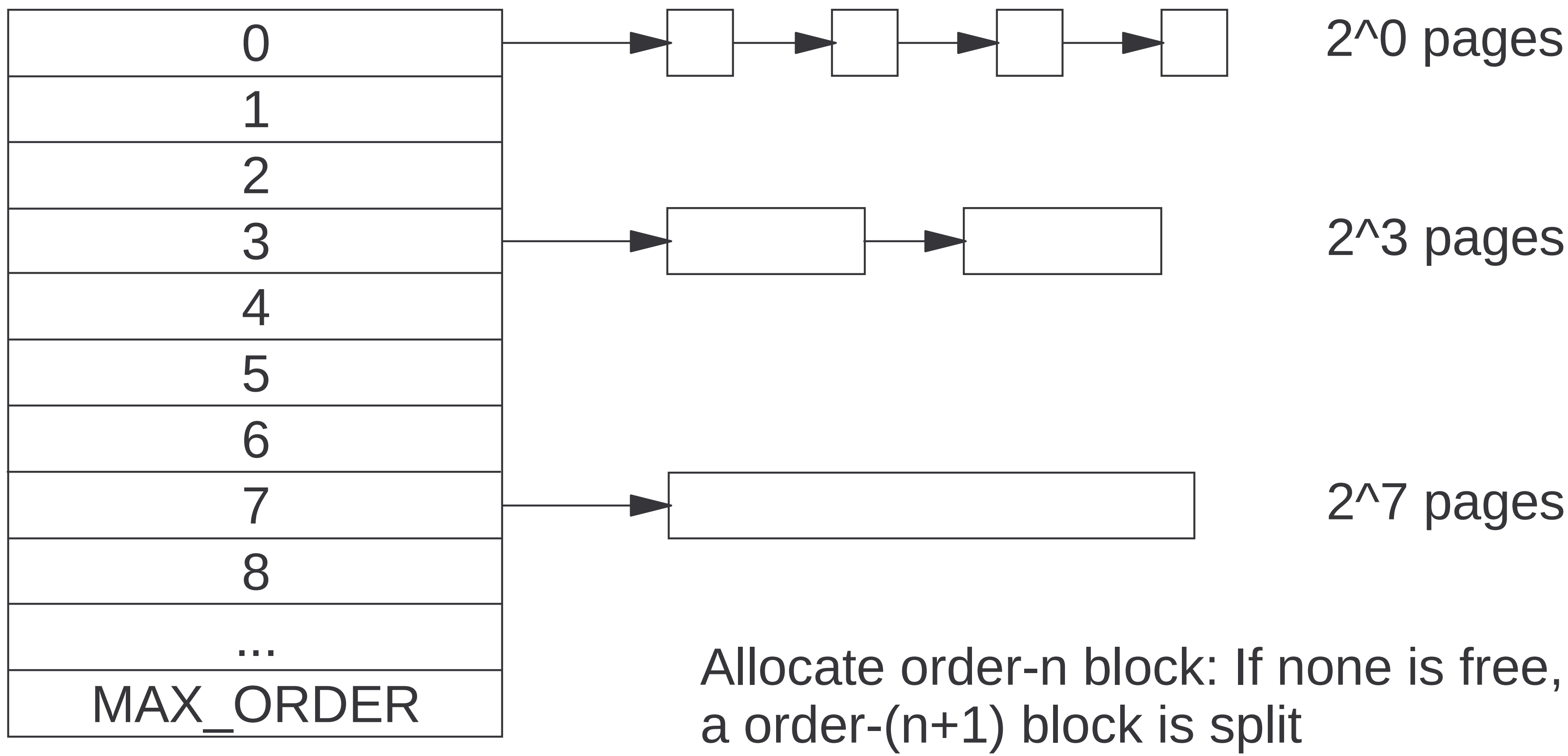
- On 31-bit zSeries all memory
- On 64-bit zSeries all < 2 GB
- (On Intel all < 16 MB)

Kernel Memory Allocators

- Low-level page allocator
  - Buddy system for contiguous multi-page allocations
  - Provides pages for
    - in-kernel allocations (slab cache)
    - vmalloc areas (kernel modules, multi-page data areas)
    - page cache, anonymous user pages
    - misc. other users
- Slab cache
  - Manages allocations of objects of the same type
  - Large-scale users: inodes, dentries, block I/O, network ...
  - kmalloc (generic allocator) implemented on top

Buddy Allocator

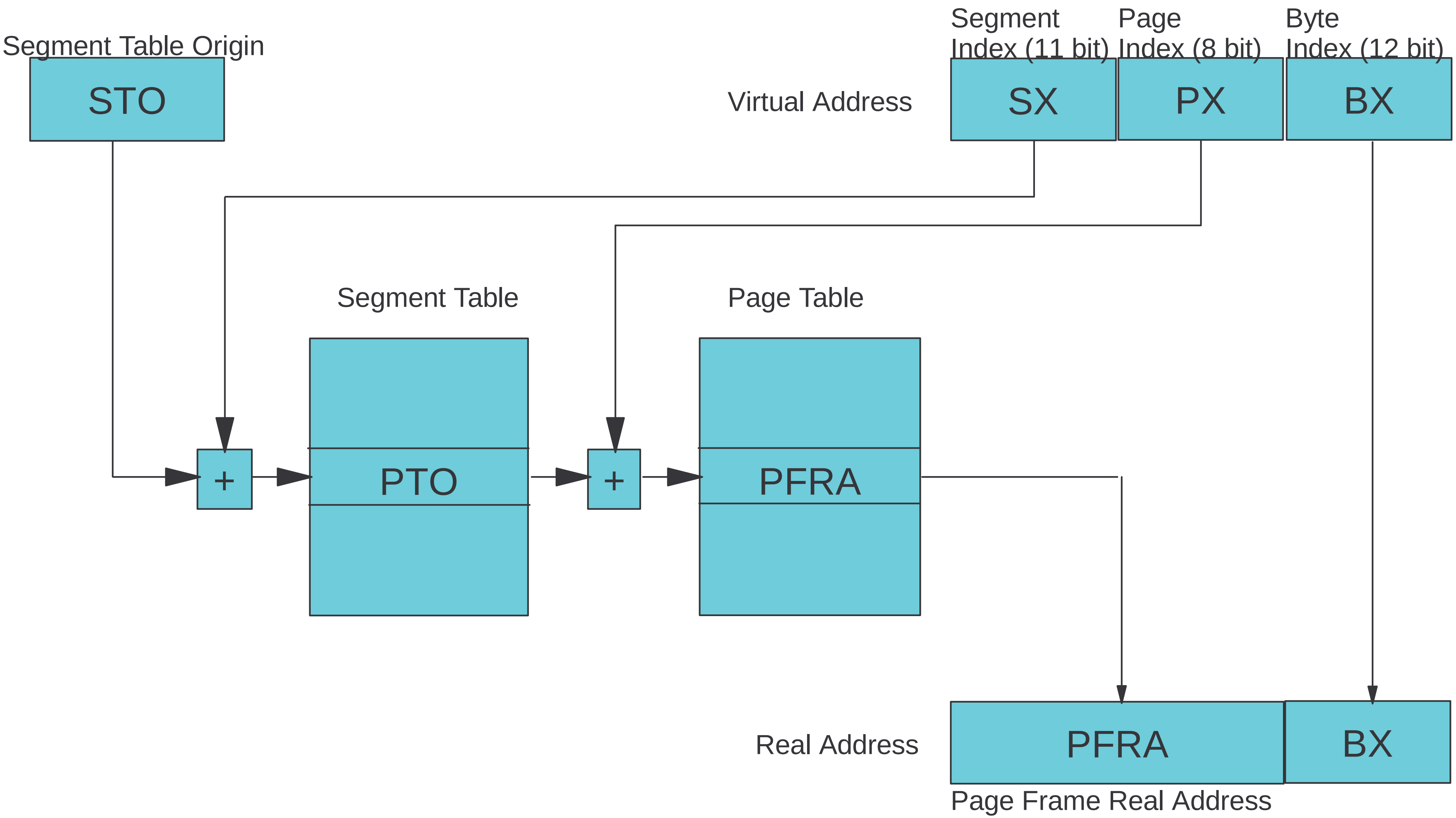
Order-n free lists (per zone)



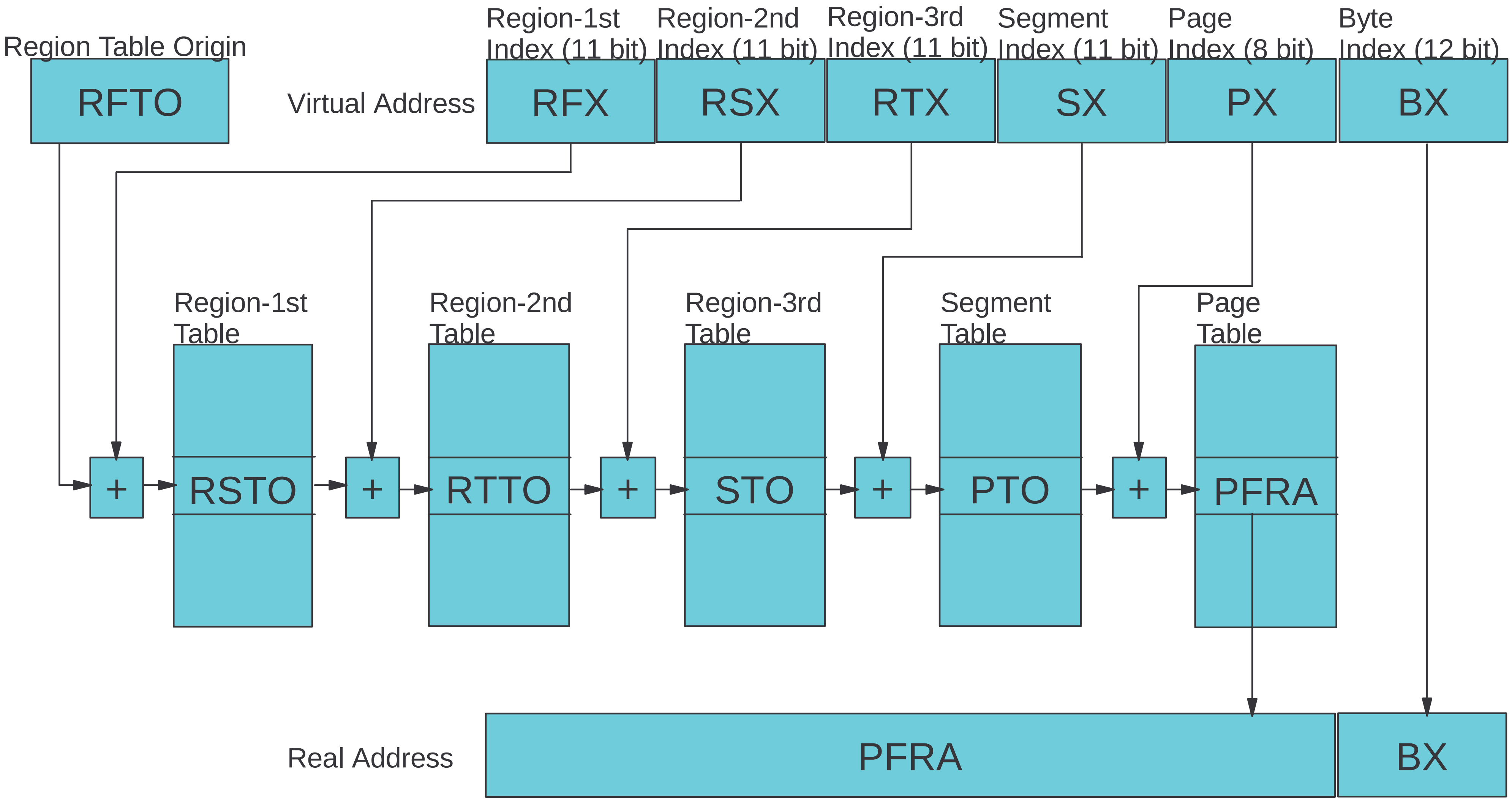
Allocate order-n block: If none is free, a order-(n+1) block is split

Free order-n block: If 'buddy' is also free, merge them to order-(n+1) block

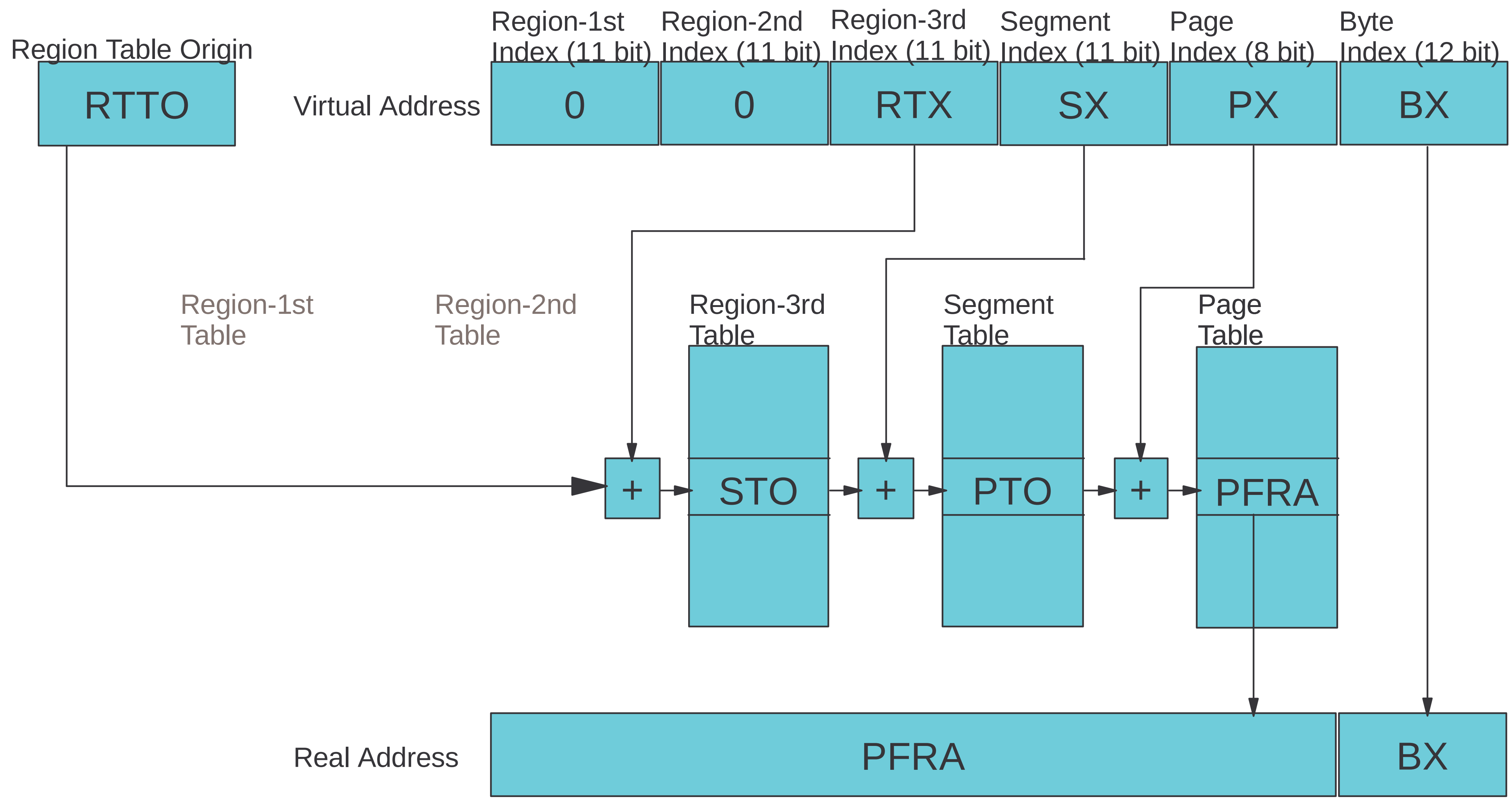
Dynamic Address Translation: 31-bit



Dynamic Address Translation: 64-bit



DAT: 64-bit Three Level Translation



zSeries Address Translation Modes

- Directly accessible address spaces
  - Primary space: STO/RTO in Control Register 1
  - Secondary space: STO/RTO in Control Register 7
  - Home space: STO/RTO in Control Register 13
  - Access-register specified spaces
- Access registers
  - Base register used in memory access identifies AR
  - AR specifies STO/RTO via Access List Entry Token
  - Operating System manages ALETs and grants privilege
  - ALET 0 is primary space, ALET 1 is secondary space

Address Translation Modes (cont.)

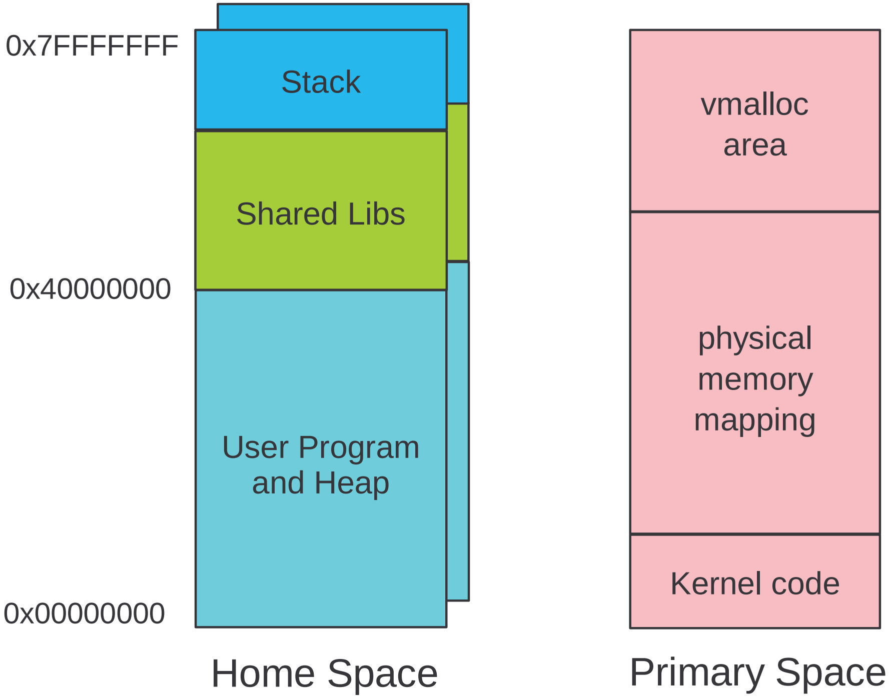
- Translation mode specified in PSW
  - Primary space mode
    - Instructions fetched from primary space, data in primary space
  - Secondary space mode
    - Instructions in primary, data in secondary
  - Home space mode
    - Instructions and data in home space
  - Access register mode
    - Instructions in primary, data in AR-specified address space

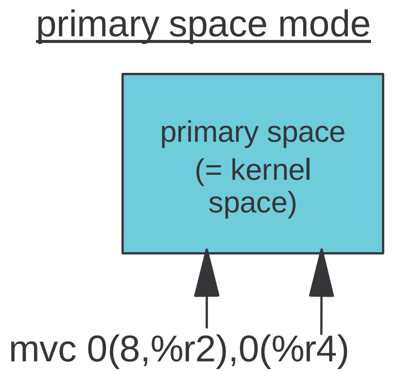
zSeries Program Status Word

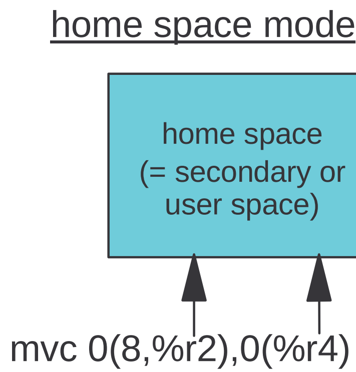

```
 _ _ _ _ _ _ _ _ _ _______ _ _ _ _ _ ___ ___ _______ _____________ _
| | | | | | |I|E|       | | | | | |   |   | Prog  |             |E|
|0|R|0|0|0|T|O|X|  Key  |0|M|W|P|A S|C C| Mask  |0 0 0 0 0 0 0|A|
|_|_|_|_|_|_|_|_|_______|_|_|_|_|___|___|_______|_____________|_|
0         5     8      12      16  18  20      24              31

 _ _____________________________________________________________
|B|                                                             |
|A|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0|
|_|_____________________________________________________________|
32                                                             63

 _______________________________________________________________
|                                                               |
|                      Instruction Address                      |
|_______________________________________________________________|
64                                                             95

 _______________________________________________________________
|                                                               |
|                Instruction Address (Continued)                |
|_______________________________________________________________|
96                                                            127
```

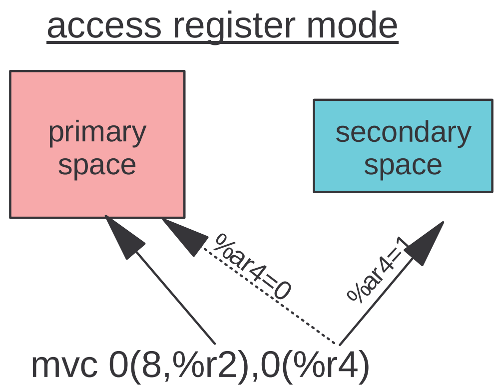
```
R:   Program Event Recording Mask
T:   Dynamic Address Translation Mode
IO:  I/O Interruption Mask
EX:  External Interruption Mask
Key: PSW Key (storage proctection)
M:   Machine Check Mask
W:   Wait State
P:   Problem State
AS:  Address Space Control
CC:  Condition Code
PM:  Program Mask
EA:  Extended Addressing Mode
BA:  Basic Addressing Mode
```

Linux on zSeries: Use of Address Spaces







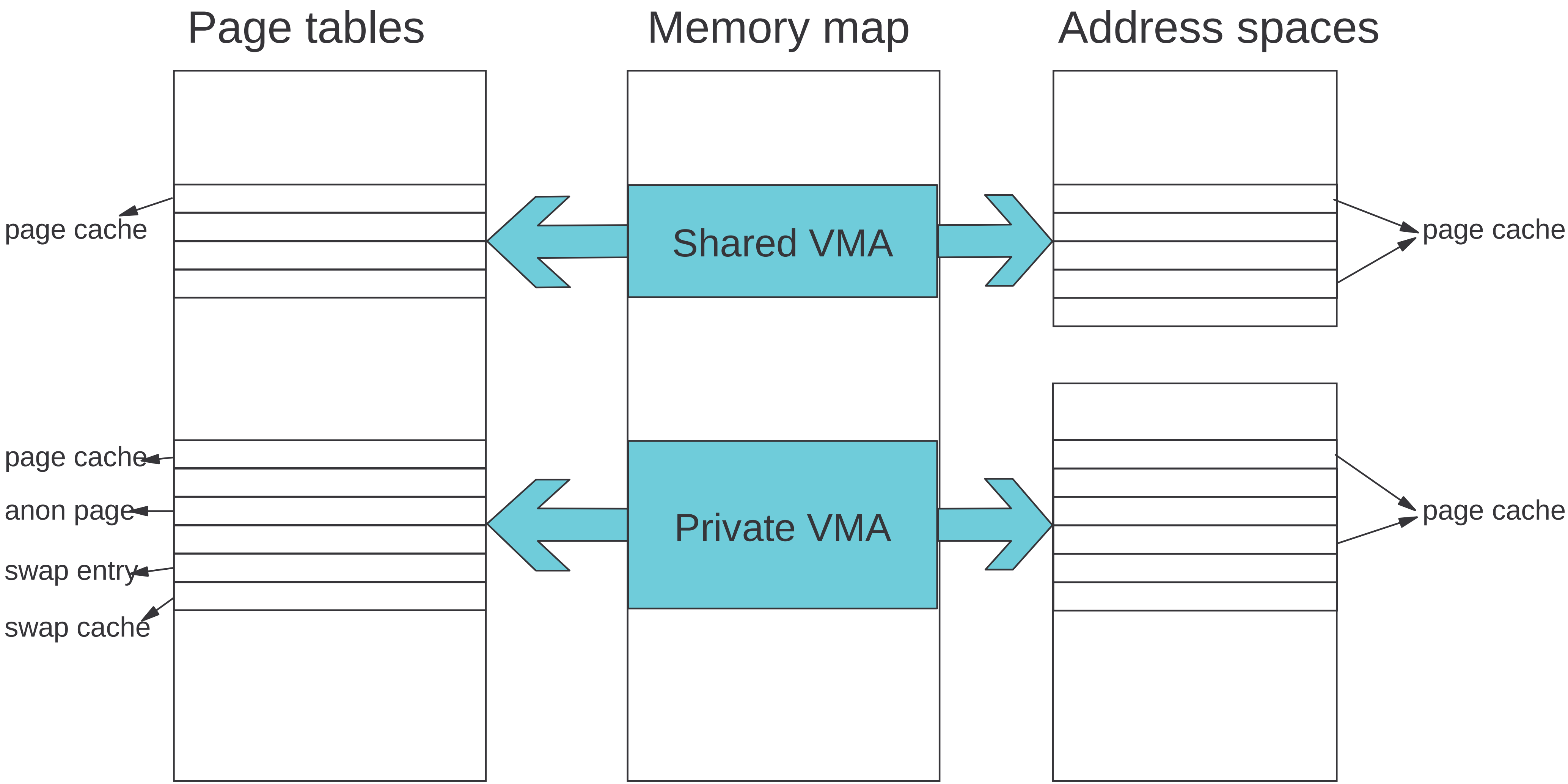



```
la 4,<source>
la 2,<destination>
sacf 512
mvc 0(8,%r2),0(%r4)
sacf 0
```

Memory Management Data Structures

- 'Address spaces'
  - Represent some page-addressed data
  - Examples: inodes (files), shared memory segments, swap
  - Contents cached in 'page cache'
- 'Memory map'
  - Describes a process' user address space
  - List of 'virtual memory arenas'
  - VMA maps part of an address space into a process
- Page tables
  - Hardware-defined structure: region, segment, page tables
  - Linux uses platform-independent abstraction
  - Contents filled on-demand as defined by MM

Virtual Memory Arenas



Page Table Entries

```
 _   _____________________   _ _ _ _  _________
|0|          PFRA          |0|I|P|0|  - - -   |
|_|________________________|_|_|_|_|_________|
0   1                      20       24       31
```

PFRA: Page Frame Real Address
I: Invalid
P: Page Protection

```
 _   _____________________   _ _ _ _  _________
|0|            0           |0|1|0|0|    0   |0|
|_|________________________|_|_|_|_|________|_|
0   1                      20       24       31
```

Empty PTE slot

```
 _   _____________________   _ _ _ _  _________
|0|          PFRA          |0|0|0|0|    0   |0|
|_|________________________|_|_|_|_|________|_|
0   1                      20       24       31
```

Read-write page

```
 _   _____________________   _ _ _ _  _________
|0|          PFRA          |0|0|1|0|    0   |0|
|_|________________________|_|_|_|_|________|_|
0   1                      20       24       31
```

Read-only page

```
 _   _____________________   _ _ _ _  _________
|0|          PFRA          |0|1|0|0|    0   |1|
|_|________________________|_|_|_|_|________|_|
0   1                      20       24       31
```

No-access page

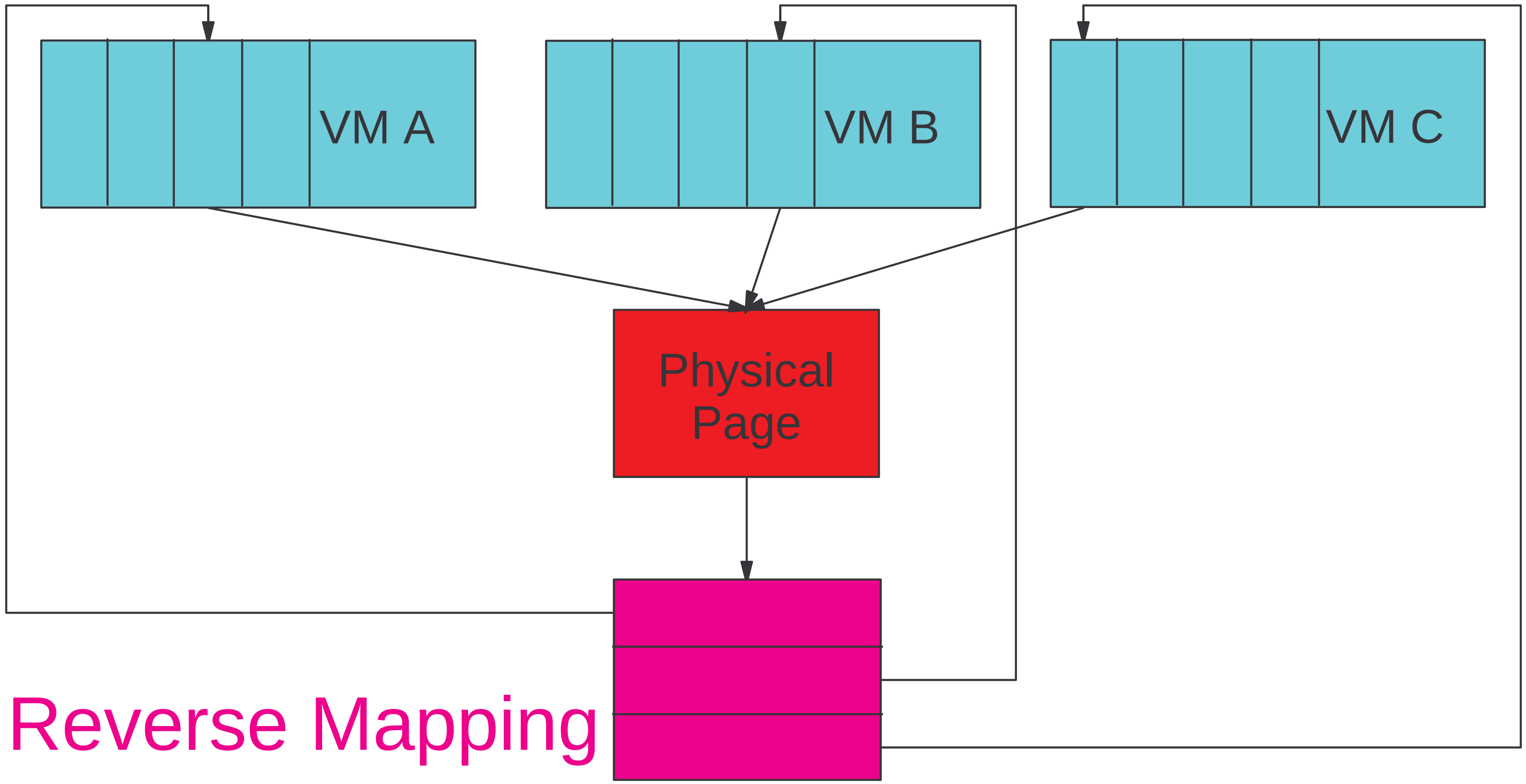
```
 _   _____________________   _ _ _ _  _________
|0|           SO           |0|1|1|0|    SA  |0|
|_|________________________|_|_|_|_|________|_|
0   1                      20       24       31
```

Swapped-out page
SA: Swap area (file/device)
SO: Offset within area

```
 _   _____________________   _ _ _ _  _________
|0|           FOH          |0|1|1|0|   FOL  |1|
|_|________________________|_|_|_|_|________|_|
0   1                      20       24       31
```

Paged-out remapped page
FOL: Offset (low bits)
FOH: Offset (high bits)

Reverse Mappings



Reverse Mappings (cont.)

SHARE
Technology • Connections • Results

- Advantages of reverse mappings
  - Easy to unmap page from all address spaces
  - Page replacement scans based on physical pages
  - Less CPU spent inside memory manager
  - Less fragile behaviour under extreme load
- Challenges with reverse mappings
  - Overhead to set up rmap structures
  - Out of memory while allocating rmap?

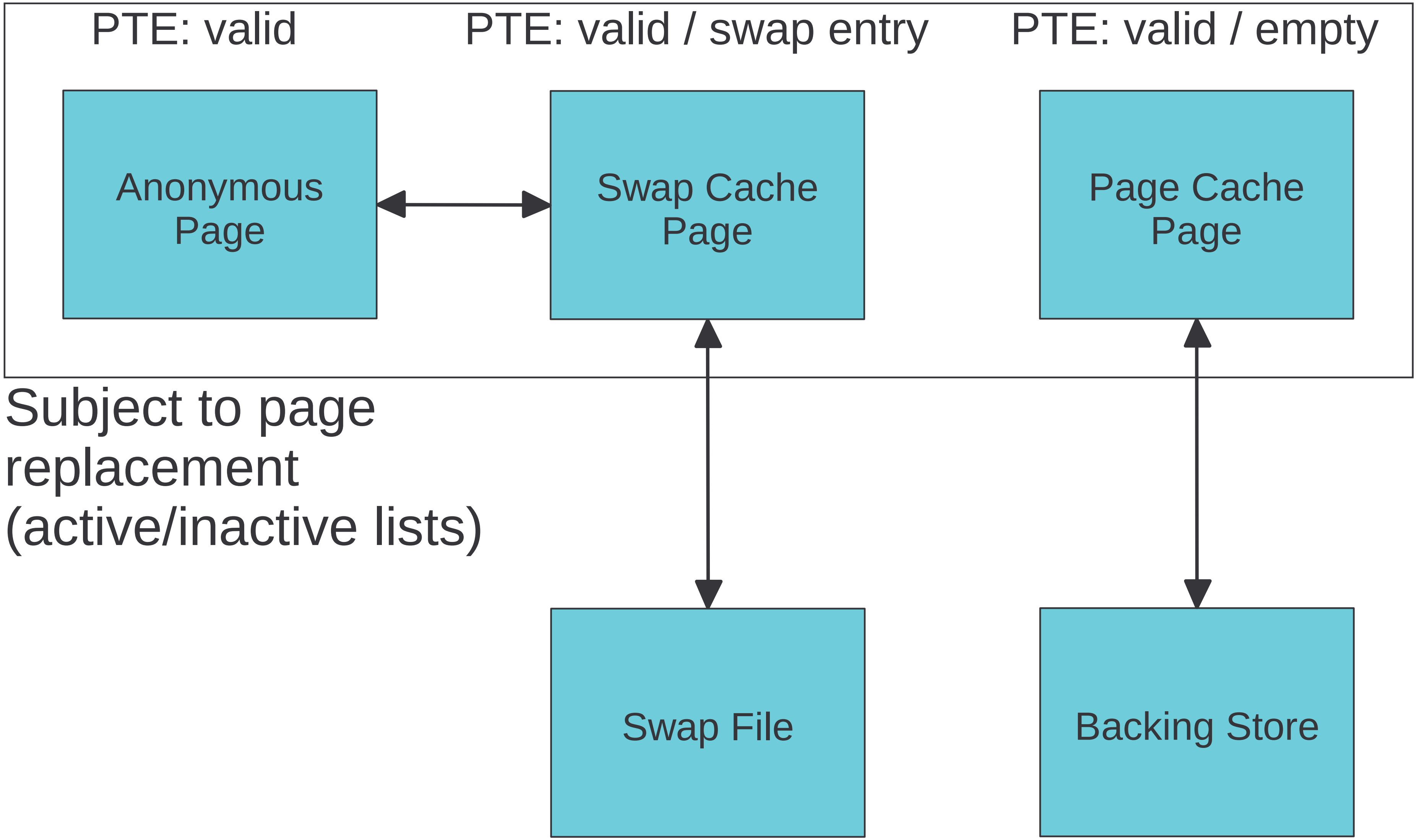
Page Fault Handling

- Hardware support
  - Accessing invalid pages causes 'page translation' check
  - Writing to protected pages causes 'protection exception'
  - Translation-exception identification provides address
  - *'Suppression on protection' facility essential!*
- Linux kernel page fault handler
  - Determine address/access validity according to VMA
  - Invalid accesses cause SIGSEGV delivery
  - Valid accesses trigger: page-in, swap-in, copy-on-write
  - Extra support for stack VMA: grows automatically
  - Out-of-memory if overcommitted causes SIGBUS

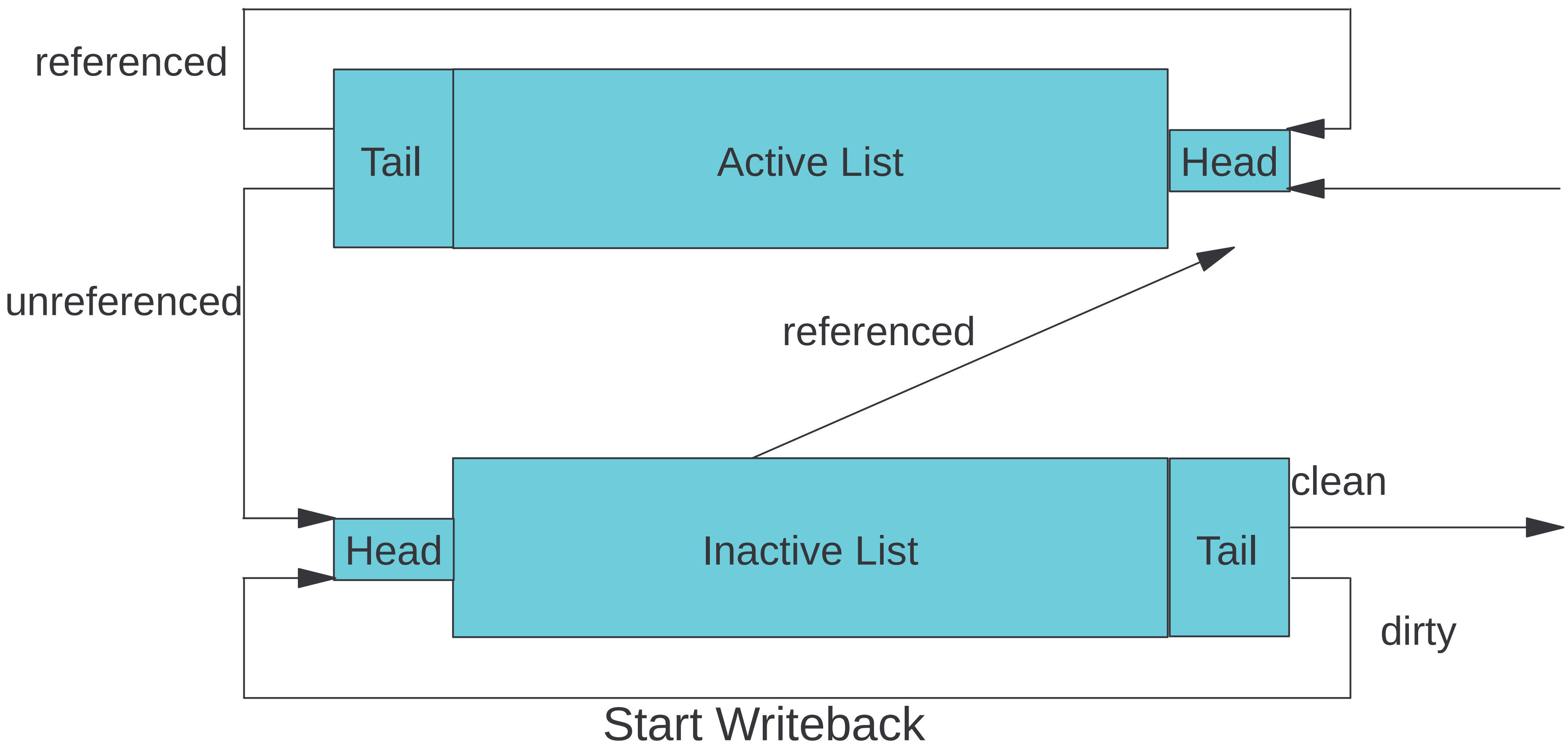
Page Replacement

- Page replacement strategy
  - Applies to all page cache and anonymous pages
  - Second-chance LRU using active/inactive page lists
  - Scan phys. memory zones, find PTEs via reverse map
- Page replacement actions
  - Allocate swap slot for anon. pages
  - Unmap from all process page tables
  - Async. write-back dirty pages to backing store
  - Remove clean pages from page cache
- Shrinking in-kernel slab caches
  - Call-back to release inode, dentry, ... cache entries
  - Balance against page cache replacement

Page Replacement Life Cycle



Page Replacement "LRU" Lists



The Mystery Solved

- What does this mean:

```
$ free
                     total       used       free     shared    buffers     cached
Mem:                512832     473256      39576          0      52300     236776
-/+ buffers/cache:             184180     328652
Swap:               524280        912     523368
```

- Or this:

```
$ cat /proc/meminfo
MemTotal:        512832 kB                     Dirty:               28 kB   Fixed
MemFree:          39512 kB                     Writeback:            0 kB
Buffers:          52308 kB  Page Cache         Mapped:            5492 kB   User
Cached:          236768 kB                     Slab:            158608 kB
SwapCached:         532 kB                     Committed_AS:      7656 kB
Active:          246328 kB  Total LRU          PageTables:         208 kB
Inactive:         61920 kB  (PC + Anon)        VmallocTotal:   1564671 kB
HighTotal:            0 kB                     VmallocUsed:        724 kB
HighFree:             0 kB                     VmallocChunk:   1563947 kB
LowTotal:        512832 kB
LowFree:          39512 kB
SwapTotal:       524280 kB
SwapFree:        523368 kB
```

Buffers - What's That?

- Original 'buffer cache' (up to Linux 2.0)
  - Cache block device contents
  - All file access used buffer cache; page cache separate
- Gradual elimination of buffer cache
  - Linux 2.2: Page cache reads bypass buffer cache
  - Linux 2.4: Buffer cache completely merged into page cache
  - Linux 2.6: 'Buffer heads' removed from block I/O layer
- Meaning of the 'buffers' field
  - Linux 2.4: Page cache pages with buffer heads attached
  - Linux 2.6: Page cache pages for block devices
  - Approximates size of cached file system metadata

Some Tunable Parameters

- sysctl or /proc interface
  - /proc/sys/vm/overcommit_memory / overcommit_ratio
    - Controls relation of committed AS to total memory + swap
  - /proc/sys/vm/swappiness
    - Influences page-out decision of mapped vs. unmapped pages
  - /proc/sys/vm/page-cluster
    - Controls swap-in read-ahead
  - /proc/sys/vm/dirty_background_ratio / dirty_ratio
    - Percentage of memory allowed to fill with dirty pages
  - /proc/sys/vm/dirty_writeback_centisecs / expire_centisecs
    - Average/maximum time a page is allowed to remain dirty

Virtualization Considerations

- Two-level dynamic address translation
  - Linux DAT: Linux virtual address -> Linux 'real' address
  - VM DAT: Guest 'real' address -> Host real address
- Two-level page replacement / swapping
  - Linux LRU prefers to touch pages VM swapped out
  - Linux / VM cooperative memory management
- Exploit VM shared memory features
  - Kernel in Named Saved Systems
  - Block device on Discontiguous Saved Segments

Two-level Page Fault Handling

- Two different scenarios possible
  - Guest page fault
    - Linux page fault handler invoked
    - Initiates page-in operation from backing store
    - Suspends user process until page-in completed
    - Other user processes continue to run
  - Host page fault
    - VM page fault handler invoked
    - Initiates page-in operation from backing store
    - Suspends **guest** until page-in completed
    - *No other user processes can run*

Two-level Page Fault Handling (cont.)

- Solution: Pseudo Page Faults
  - VM page fault handler invoked
    - Initiates page-in operation from backing store
    - Triggers guest 'pseudo page fault'
    - Linux pseudo page fault handler suspends user process
    - VM does not suspend the guest
  - On completion of page-in operation
    - VM calls guest pseudo page fault handler again
    - Linux handler wakes up blocked user process
- Caveats
  - Access to kernel pages
  - Access to user page from kernel code

Cooperative Memory Management

- Problem: Large guest size hurts performance
  - Linux will use all memory; LRU tends to reuse cold pages
  - Recommendation: Make guest size as small as possible
  - But how to determine that size?
- Cooperative Memory Management
  - New feature released on devWorks 01/2004
  - Allows to reserve a certain number of pages
    - Kernel module allocates pages, so Linux cannot use them anymore
    - Pages are (if possible) reported as free to VM
  - Changes effective available memory size without reboot!
  - IUCV special message interface allows central instance to manage server farm total memory consumption

Cooperative Memory Management (cont.)

- sysctl or /proc user interface
  - /proc/sys/vm/cmm_pages
    - Read to query number of pages permanently reserved
    - Write to set new target (will be achieved over time)
  - /proc/sys/vm/cmm_timed_pages
    - Read to query number of pages temporarily reserved
    - Write *increment* to add to target
  - /proc/sys/vm/cmm_timeout
    - Holds pair of N pages / X seconds (read/write)
    - Every time X seconds have passed, release N temporary pages
- IUCV special message interface
  - CMMSHRINK/CMMRELEASE/CMMREUSE
  - Same as cmm_pages/cmm_timed_pages/cmm_timeout write

Resources

- Mel Gorman's Linux VM Documentation
  http://www.csn.ul.ie/~mel/projects/vm/

- Linux on zSeries developerWorks page
  http://www.software.ibm.com/developerworks/opensource/linux390/index.html

- Linux on zSeries technical contact address
  linux390@de.ibm.com