

# s390 KVM memory management... and its pitfalls

—  
Janosch Frank <frankja@de.ibm.com>

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)

The following are trademarks or registered trademarks of other companies.

- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

# Contents

## **Physical memory**

Address types & Lowcore

Storage Keys

## **Virtual Memory (DAT)**

Tables

ASCEs

## **KVM**

SIE & pageable storage mode

GMAP

PGSTEs and notifiers

Huge Pages

## **Lessons Learned**

# Physical Memory

# Storage (Memory)

- 24, 31 and 64 bit addressing
- Physical memory is addressed via three address types:

Type	Description
Absolute	Continuous 1:1 mapping of available storage.
Real	A prefix for lowcore translates to absolute addresses.
Virtual	DAT to real or absolute translation. AR to real or absolute translation.

# Absolute / Real Storage

- Cores have dedicated 8k housekeeping area (lowcore).
- Contains old / new exception addresses, interrupt codes, etc.
- The prefix register specifies the **absolute** location of its lowcore.
  - Accesses to 0 – 8k will be redirected to prefix location.
  - Access to the prefix will be redirected to 0.

# Storage Keys

# Storage Keys

- Each 4k physical block has a key attached:



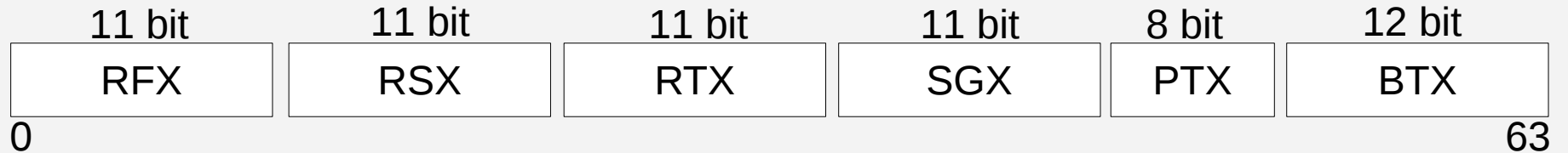
- Kept in not addressable memory.
- Manipulated through **privileged** instructions.
- Key ACC has to match with current ACC field in CPU status register for store.
- If **F** is set fetches also need a ACC match.
- **R & C**: Indicates that the block has been referenced / changed



# Virtual Memory


# Dynamic Address Translation (DAT)

- Virtual address contains indexes into tables:



- Each entry has the location of the next lower table.
- A PTE designates a 4k block instead of a table.
- The last 12 bit are a byte index into the block.
- Entries are 8 bytes long.

# DAT - Table types

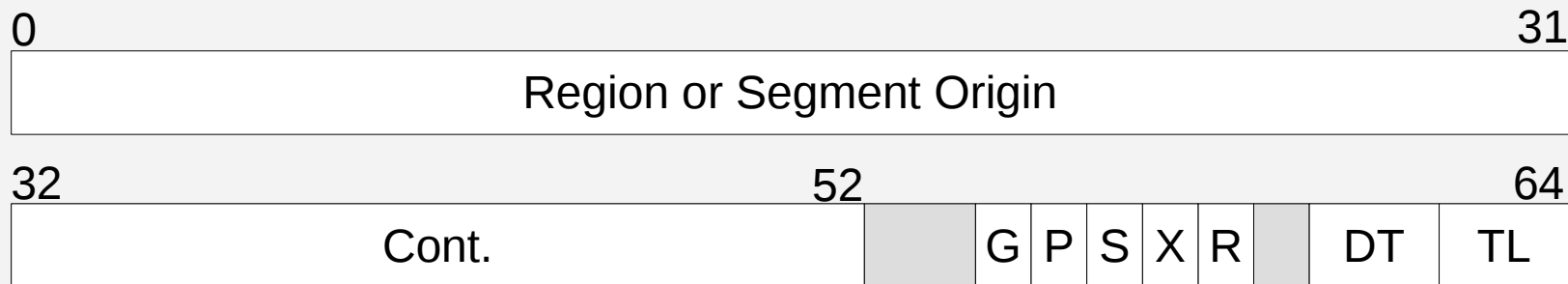
Name	Designation Size	Linux	# of Entries
Page	4k	pte	256
Segment	1M	pmd	2048
Region 3	2G	pud	2048
Region 2	4T	p4d	2048
Region 1	8P	pgd	2048
Sum:	16E		Magic Number

# DAT - Address Space Control Element (ASCE)

## Available ASCEs:

- Primary
  - Secondary
  - Home
  - Access Registers (16)
- } CRs 1,7,13

DT bits	Type
11	R1
10	R2
01	R3
00	Segment



# The KVM / SIE side of things

# Start Interpretive Execution (SIE)

- Runs code in guest context
- Input: SIE control block (SCB) and memory designation
- KVM uses pageable-storage mode
  
- Multiple SCB formats and two levels of hardware virtualization support.

# Pageable storage mode

- Primary ASCE used for guest absolute → host real translation
- DAT management like other instructions:
  - SIE DAT faults result in host exception / SIE exit
  - DAT write protection can be used for COW and migration
  - Invalidation instructions will remove SIE related TLB entries
- Guest can have own prefixing and DAT tables:
  - Host virtual / guest absolute + prefixing → guest real
  - Guest real + guest DAT → guest virtual

# GMAP

- The **g**uest **m**apping code handles the guest's address space.
- Mapping data stored in gmap struct.

- **gmap\_create(mm, memory limit)**

Creating a gmap struct for the guest.

- **gmap\_map\_segment(gmap, vmaddr, gaddr, size)**

Creating mapping data to make faulting possible.

Maps QEMU virtual addresses to guest physical.



# GMAP – Struct

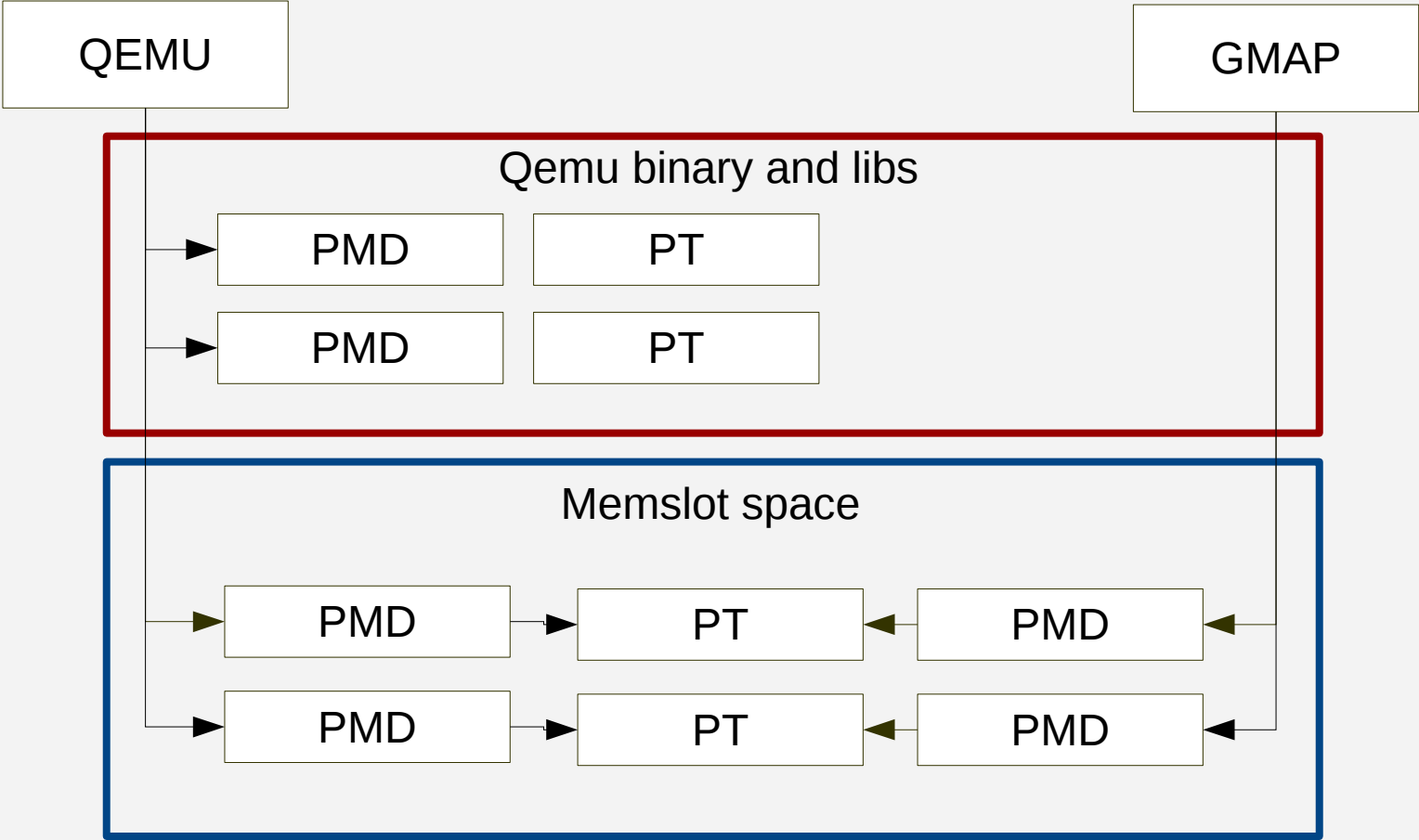
The GMAP struct contains all necessary information.

- **mm:** parent mm space (QEMU)
- **guest\_to\_host:** radix tree with guest to host address translation
- **host\_to\_guest:** radix tree with pointer to segment table entries
- **asce:** Guest ASCE used for SIE
- **table:** Pointer to first DAT table

# GMAP – Faulting

1. SIE DAT faults end up in host.
  2. Host searches in guest to host translation for vmaddr and locates associated PMD entry.
  3. Host creates intermediate guest tables between first and segment (PMD table).
  4. Host copies QEMU PMD entry.
  5. Host makes a host to guest entry pointing to the pmd entry.
- ➔ Shared page tables between userspace and guest.

# GMAP – Memory view



# GMAP – Sharing implications

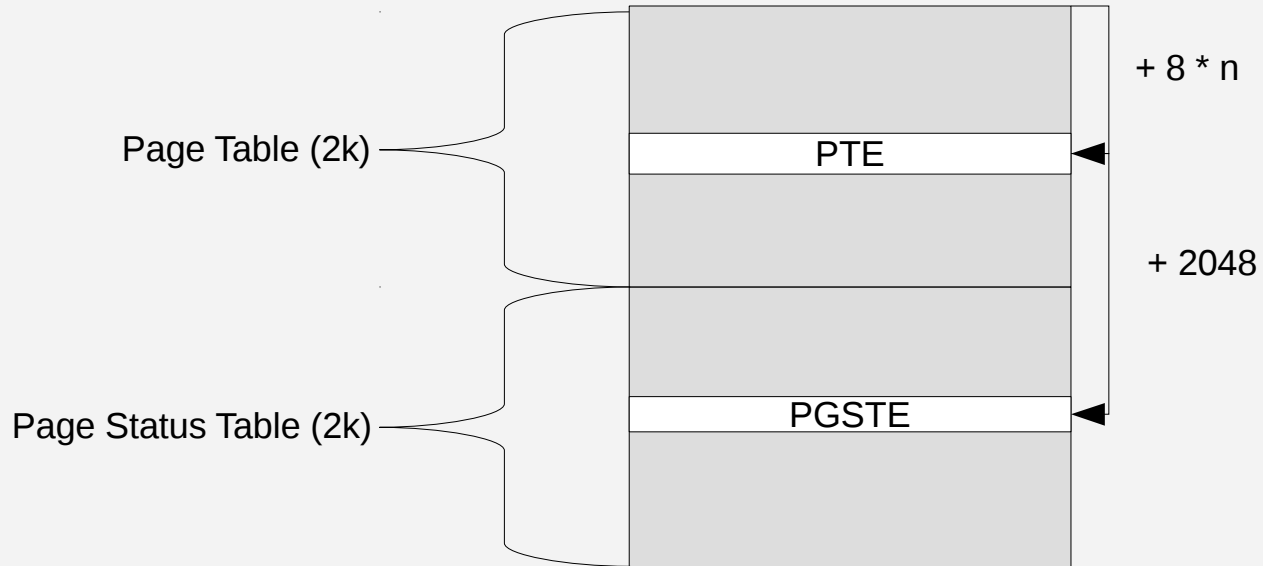
- Write protection tracks QEMU and the VM.
- One flush for QEMU and VM TLB entries.
- Linux PTE manipulation functions include VM management.
- Less page tables.
  
- But...

## Pitfall #1

For VMs to run, we need to switch the page table size from 2k to 4k.

This used to be system wide!

# PGSTEs – Page-Status-Table Entries



# PGSTE

- What happens to a storage key if the PTE is invalid?
- Special VM related status area for each PTE
- Each entry: 2048 bytes offset to corresponding PTE, 8 byte long
- Stores:
  - Storage key and host / guest R & C views
  - Special lock for PTE and PGSTE
  - Collaborative mm status and some software bits
- mm context *has\_pgste* attribute, if attribute is set, pte manipulation functions take PGSTE into account.

# Solution #1

- New **PT\_S390\_PGSTE** ELF segment type.
- If kernel encounters said segment, it will set **TIF\_PGSTE** and restart exec syscall.
- The mm context sets `alloc_pgste` when seeing the TIF bit and from thereon allocates 4k page tables for this process.
- QEMU has this special segment.



## Pitfall #2

A vCPU's lowcore has to be  
accessible as long as it's running.  
How can we set write protection?

## Solution #2 – Notifiers

- PTEs mapping lowcore are marked with special PGSTE notifier bit.
- Before invalidation of a guest's lowcore PTE a notifier triggers.
- The guest cpu is kicked out of SIE and waits on a lock until modification is done.
- After the modification, the cpu's thread PROT\_WRITEs the PTE and resets the notifier.
  
- The lowcore is heavily accessed.
- Fortunately invalidation is rare.

# Huge page support – Basics

- PMD tables are not shared, we copy the PMD.
- If QEMU PMD is flushed, we need a second GMAP flush.
- If we flush, we need to:
  - Look up the GMAP pmd.
  - Notify based on GMAP pmd software bits.
  - Remove the GMAP pmd and re-fault.
- Fortunately huge pages are normally not flushed after fault-in.

## Pitfall #3

Huge page locking

# Huge page support – Locking

- Locking is done via two main locks:
  - GMAP page\_table\_lock (one per GMAP)
  - pte lock (unified for gmap and QEMU, table sharing)
- Before huge pages, each had a distinct job.
- page\_table\_lock was locked for everything but pte access.
- pte lock was used for pte manipulation / access

# Huge page support – Locking

- Huge page support blurs lines:
  - Pmd lock for QEMU
  - `page_table_lock` for all GMAP pmd accesses
- This one `page_table_lock` is a huge performance bottleneck.
- Proposal: Let's take the QEMU pmd lock.
- Multiple locks → performance, also locking userspace, smaller deadlock risk

## Pitfall #4

Have you seen my storage keys?

I just put them in that unmapped area.

# Pitfall #4

- With huge pages, storage key instructions accessing an invalid last level table entry will fault.
- With hardware interpretation we only see the fault, fixup and re-drive execution.
- KVM emulation gets a lot more complicated.
- Mistakes were made, hosts did panic.



## Pitfall #5

What are these keys doing on my mint condition memory?

## Pitfall / Solution #5

- Before giving memory to a guest, old storage keys have to be wiped.
- New PTEs have a PGSTE storage key of 0
- Clearing is done when PTE becomes valid and key is automatically set from PGSTE.
- For huge pmds we have to track that within Linux.
  
- **PG\_arch\_1** page flag is used as an indicator if clearing has already happened.

# Lessons learned while doing s390 memory management

1. Optimizations might bite you in the end.
2. Do not work through all edge cases, build a general solution.
3. For locking this is especially important.
4. Broken mm code might run successfully if you don't put 100% load on it. Stress the host and guest simultaneously to find problems.

Thank you

## Pitfall #6

PGSTE handling while switching between  
huge and normal pages.

A nightmare

# Solution #6

- We would need to synchronize switchover to and from page tables.
  - PMD entry would need to be invalid for switch.
  - When switching to huge page, we'd need to set storage keys from PGSTEs.
- 
- Synchronization too hard for performance gain
  - Easier to force KVM emulation of instructions accessing PGSTE.