How TRACE32® handles...

Memory Management Units

Lars Zimmermann

2017 / 09 / 26

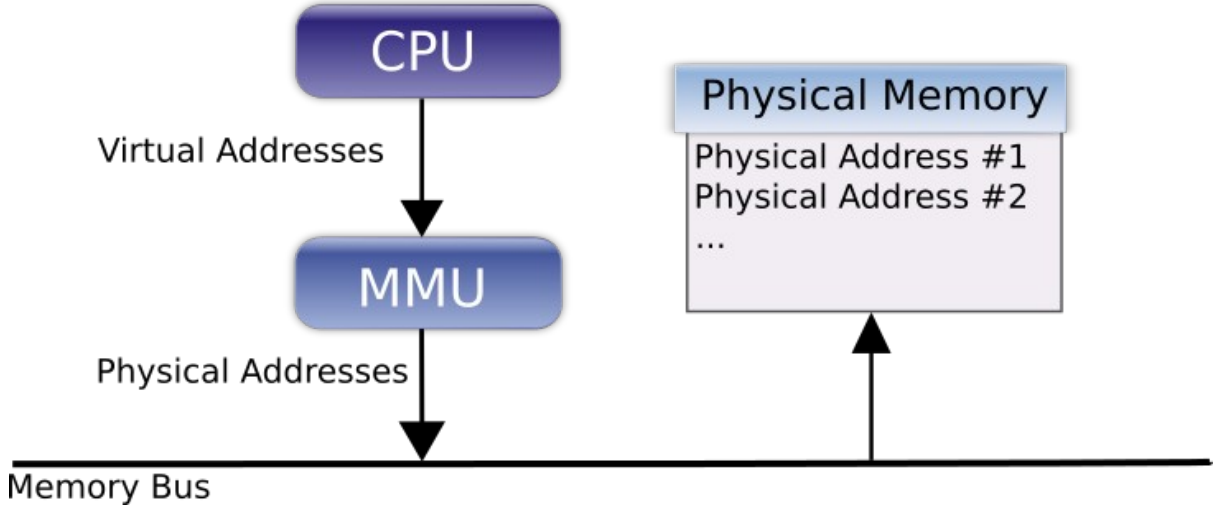
www.lauterbach.com

LAUTERBACH
DEVELOPMENT TOOLS

Table of Contents

Table of Contents

What Is A Memory Management Unit?



- The memory management unit (MMU) is a hardware component which is part of the CPU.
- The task of the MMU is to abstract the physical memory addresses to one or more virtual address spaces.
- When the MMU is enabled, the CPU accesses memory via virtual addresses which will be translated by the MMU to physical addresses before sending them on the memory bus.
- Every CPU core has its own MMU.

Why Memory Abstraction?

Memory abstraction has three big advantages:

In a multi-tasking environment an OS can assign a dedicated virtual address space to every process.

1.) As a result, a process cannot access any physical memory which belongs to another process.

→ **Memory protection**

2.) A linear address range for each process makes a dynamical allocation of memory during run-time much easier and less time-consuming.

→ **Dynamical memory allocation**

3.) The OS can extend the physical memory by swapping currently unused memory to a mass storage (hard disk, for example).

→ **Page swapping**

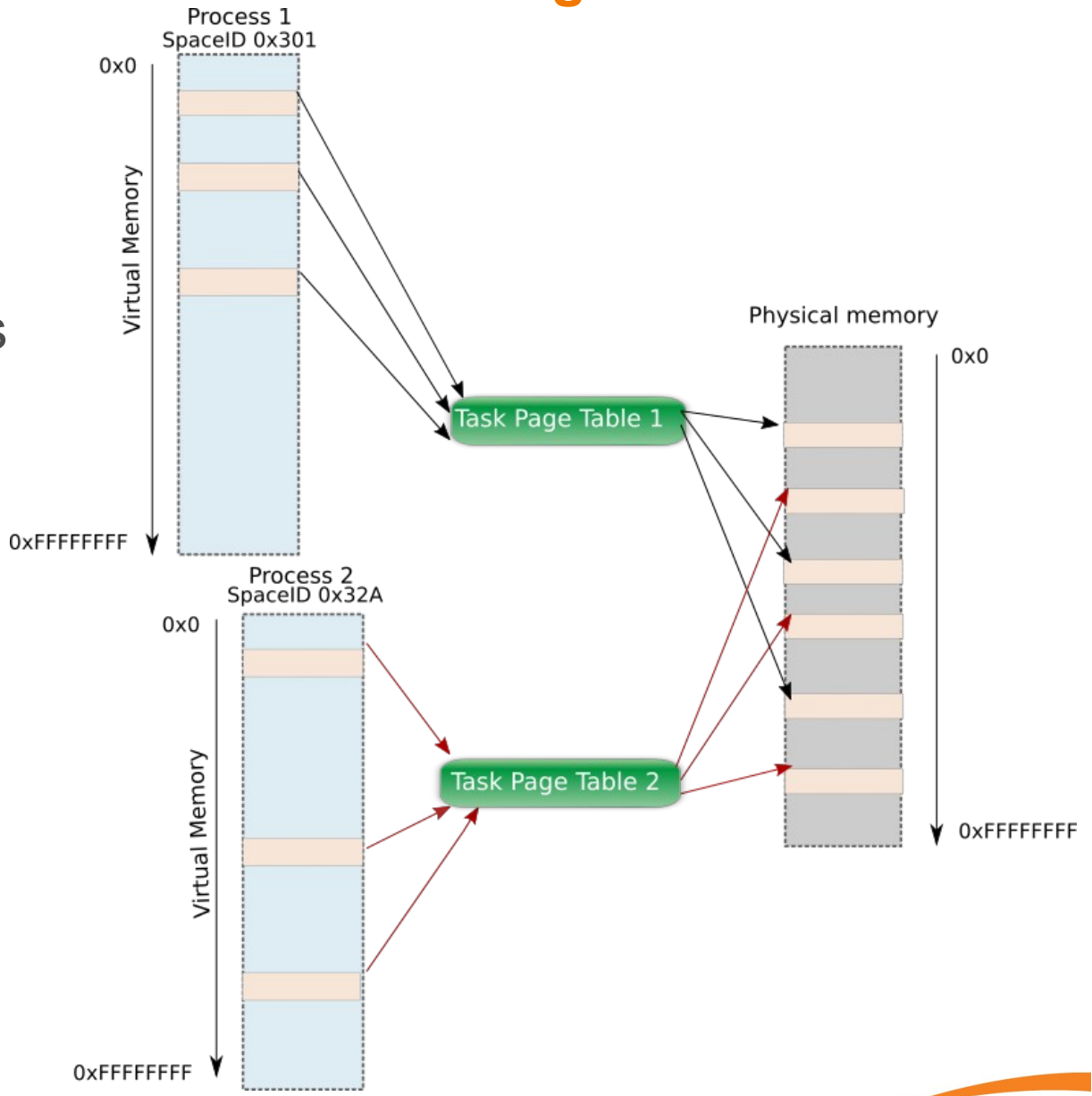
Table of Contents

MMU Structure

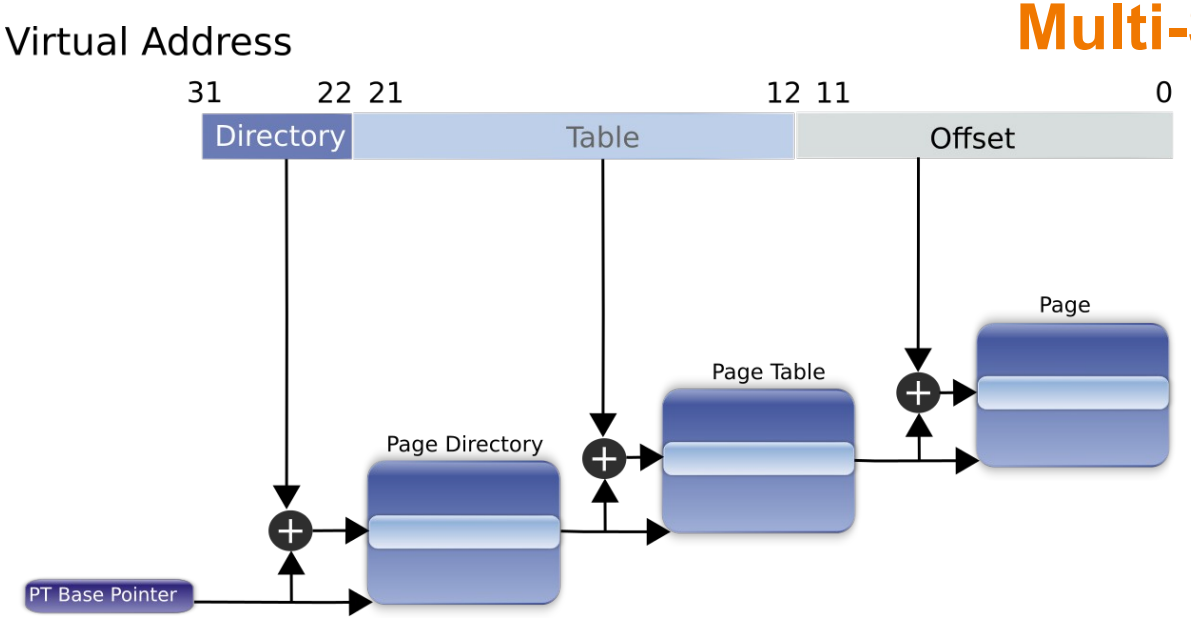
- The mapping of virtual addresses to physical addresses is done in blocks of 4 … 64 kB (CPU-specific). These blocks are called *pages*.
- The mapping between virtual addresses and physical addresses is stored by the OS in dedicated data structures which are called *page tables*.
- If the CPU needs a translation from a virtual address to a physical address, the MMU searches in the page tables for the corresponding entry. This procedure is called *table walk*.

How To Distinguish Between Different Logical Addresses?

- The example on the right shows how an MMU translates the virtual address space of two processes via page tables to the physical memory.
- Every process has its own Task Page Table
- TRACE32 introduces so called *SpaceIDs* to distinguish between logical addresses of different processes in TRACE32 PowerView.



Multi-Stage Systems



- Page tables are built in a tree-like structure of sub-tables to store logical-to-physical translations. This is an efficient way to keep them as small as possible in memory.
- The *branches* in the tree are called *page directories*. Entries in the page directories point to the next deeper level of the tree.
  The last level, the *leaves,* are the tables holding the final physical address translation
- The *indices* selecting the table entry for each level are calculated from bit slices of the logical input address which is to be translated.

Translation Lookaside Buffer

- Doing a table walk is time-consuming and decreases the performance of the CPU.
- Therefore MMUs are equipped with a cache which is called *Translation Lookaside Buffer* (TLB).
- The MMU is caching the latest found mappings in the TLB to reduce the number of table walks.

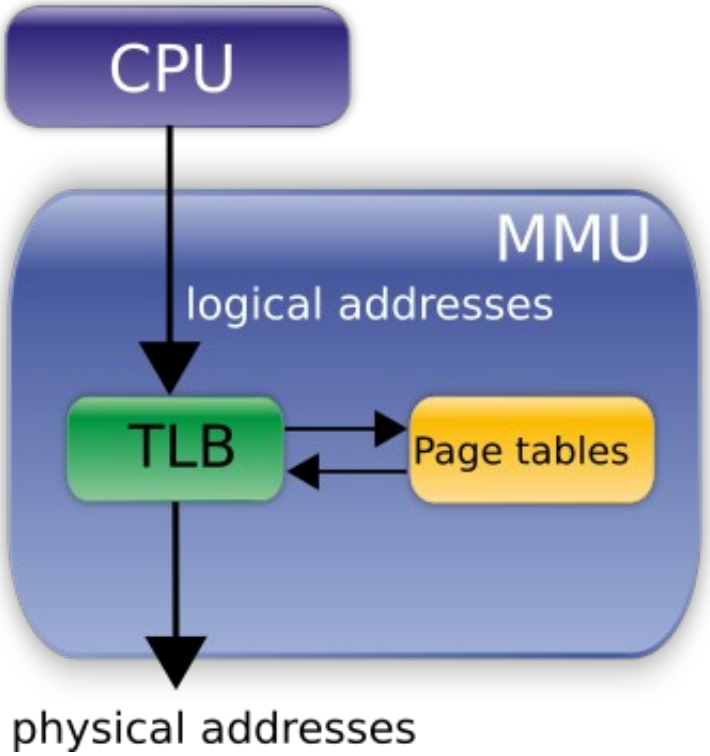


Table of Contents

TRACE32® MMU Support: Hardware Access

TRACE32 supports the MMU in two different ways:

- First, TRACE32 provides access to the hardware MMU.

  The command group **MMU**
  allows access to the MMU registers, the current page tables and allows to dump the TLB for MMU analysis
  (Not all TLBs can be accessed by a debugger).

TRACE32® MMU Support: Internal Address Translation

- Second, TRACE32 is able to perform address translations by itself without the usage of the hardware MMU:
  In addition to user defined (static) translations, TRACE32 is able to do MMU page table walks for defined page tables dynamically created by an OS.

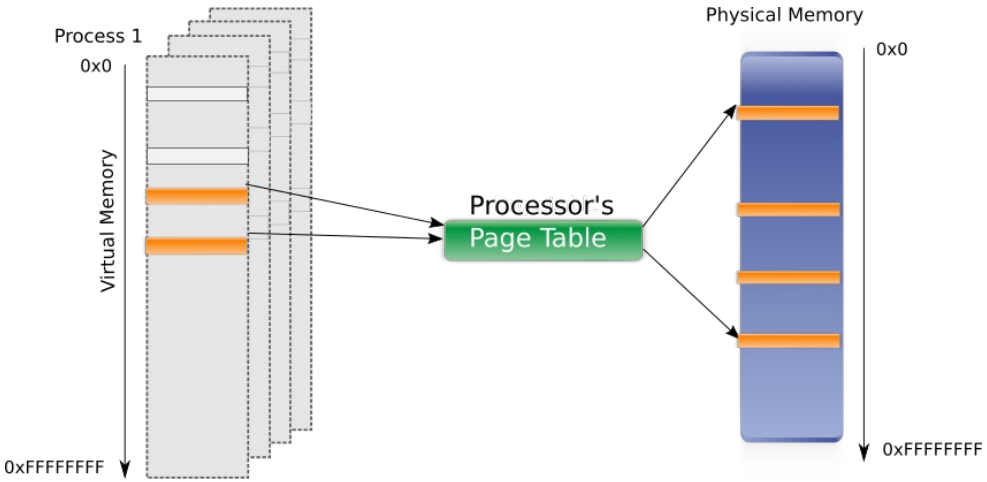
  The command group **TRANSlation**
  contains all commands related to the TRACE32 internal address translation.

Why Is An Internal Address Translation Needed?

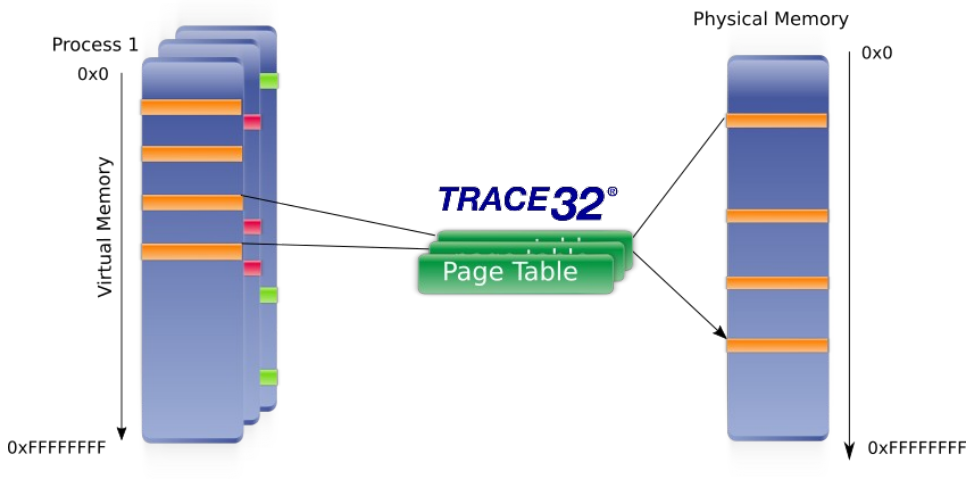
- In a multi-process environment each process has it's own page table. As the CPU's MMU is configured for the page table of the running process, translations for inactive processes are not known.
- In conjunction with a kernel awareness, TRACE32 can do the page table walk for all processes in the system, using its own internal address translation.
- This allows TRACE32 to get access to data and code of every process at any time. The MMU hardware registers and TLB content remain unchanged.

Example

Without TRACE32 internal address translation



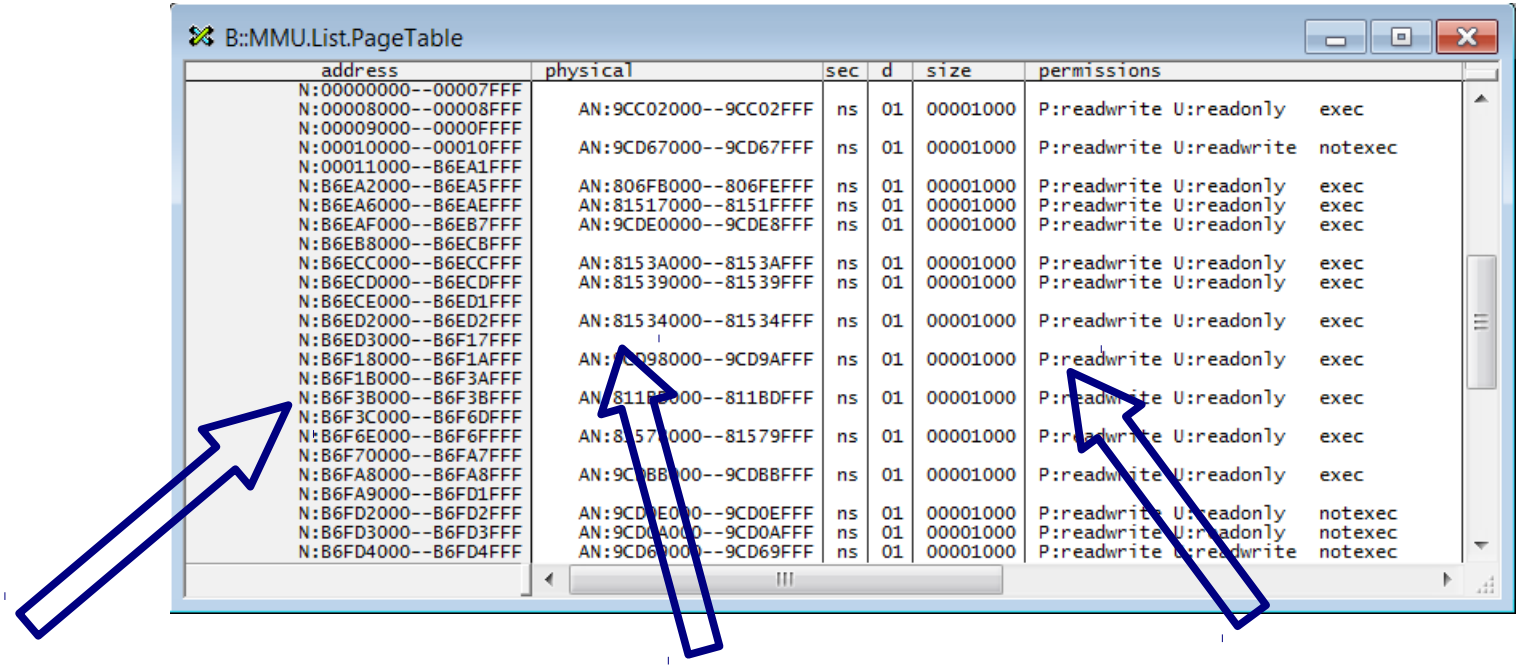
WithTRACE32 internal address translation



- Without the TRACE32 internal address translation, only the current process and kernel data would be visible.
- With the TRACE32 internal address translation, it is also possible to access the kernel data and data of every other running process.

Access To MMU Hardware

- The command MMU.List.PageTable shows the relationsship between the logical and physical addresses, which are currently translated by the hardware MMU.



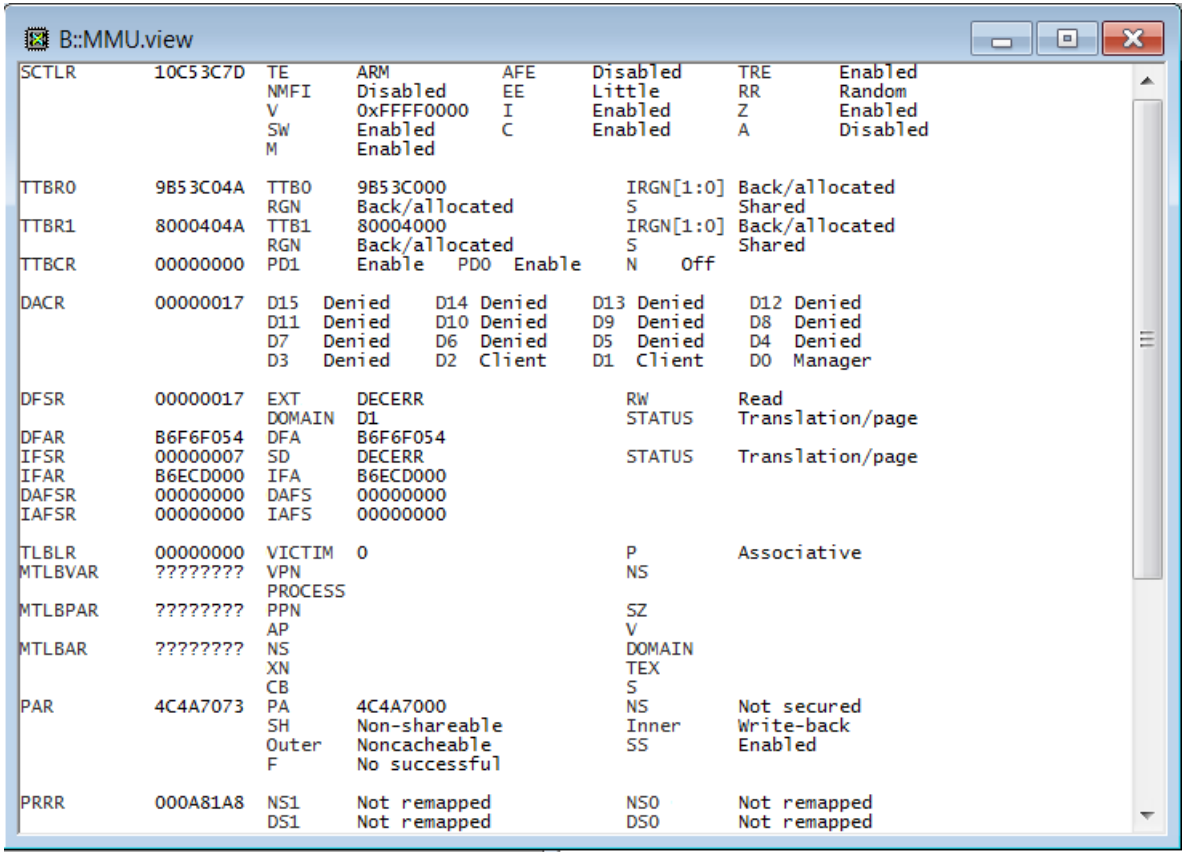
logical addresses

The access class AN indicates a physical address (‘A’ = absolute)

Page flags such as the permission or cache-ability are shown

MMU Registers

- MMU.view allows access to the MMU registers.



TRACE32® Internal Address Translation

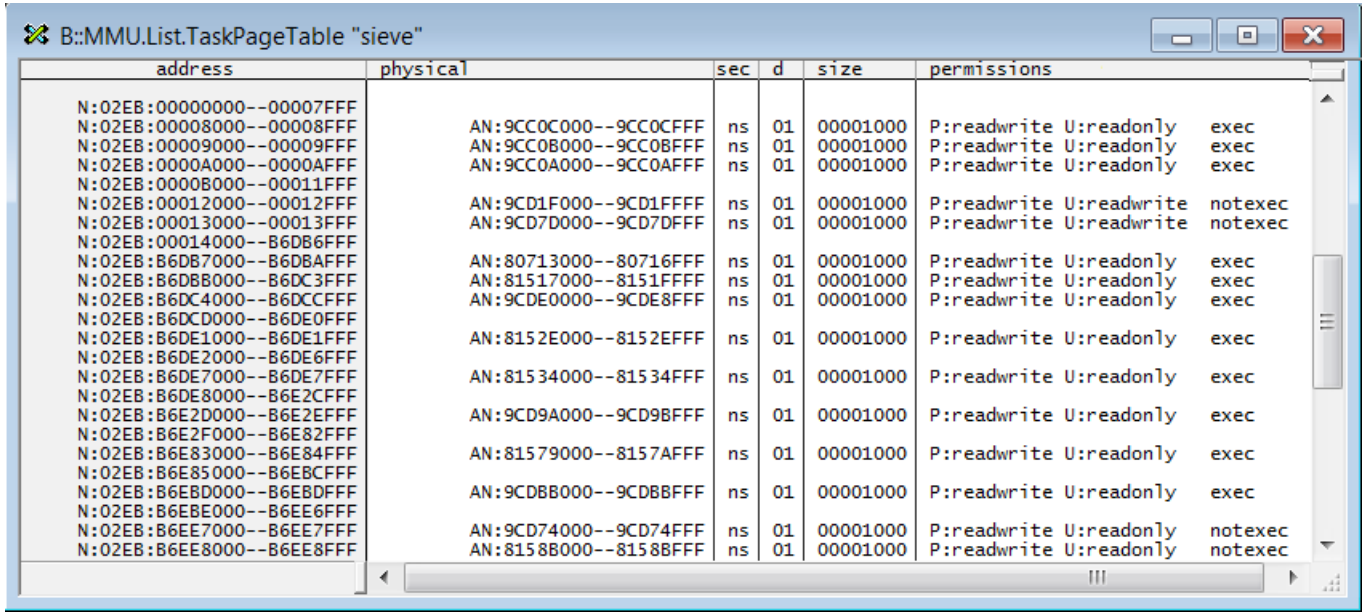
- TRANSlation.ON: Enables the internal debugger address translation mechanism. TRACE32 translates logical address to physical addresses before accessing the target.

- TRANSlation.TableWalk ON: TRACE32 performs a table walk, if no user defined static address translation is available.

- With TRANSlation.ON and TRANSlation.TableWalk ON, the TRACE32 address translation is enable.
  Then TRACE32 bypasses the hardware MMU by doing the address translation from virtual to physical memory internally. TRACE32 accesses the target memory only via physical addresses then.

Task Page Table Of A Specified Process

- With MMU.List.TaskPageTable <process>, TRACE32 shows the page table for the specified process on the system.

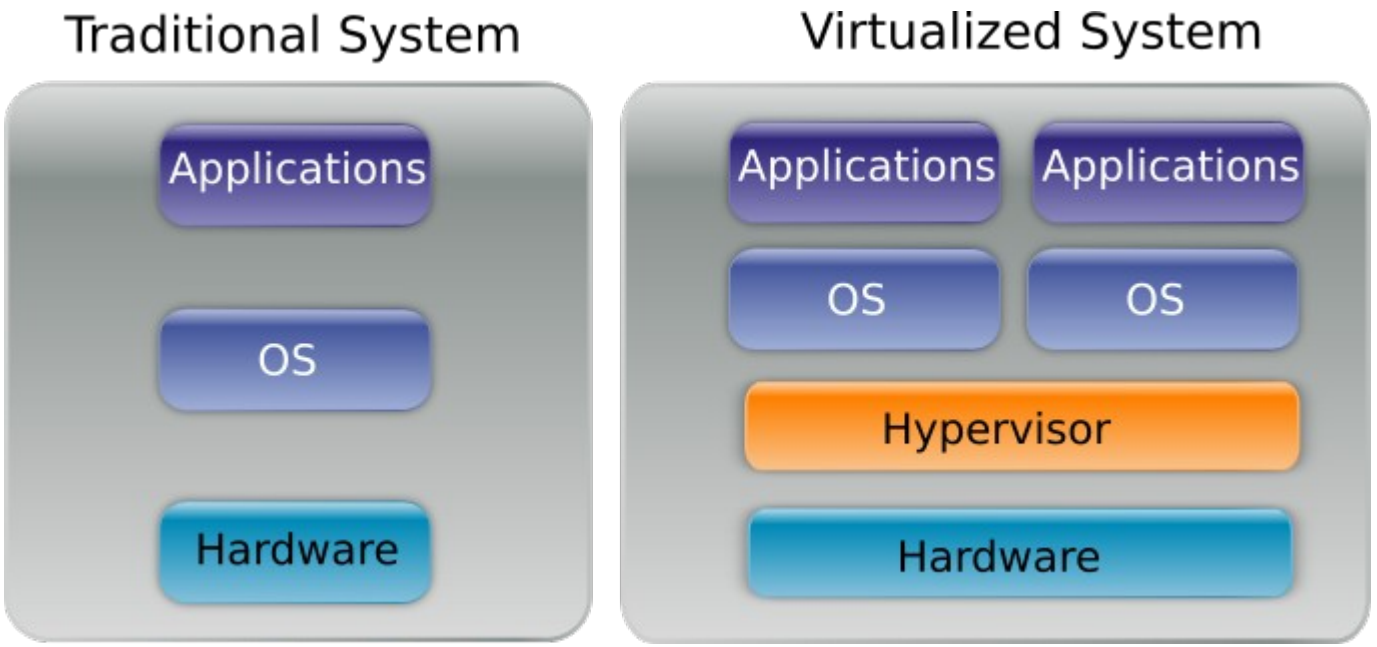


address	physical	sec	d	size	permissions	
N:02EB:00000000--00007FFF						
N:02EB:00008000--00008FFF	AN:9CC0C000--9CC0CFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:00009000--00009FFF	AN:9CC0B000--9CC0BFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:0000A000--0000AFFF	AN:9CC0A000--9CC0AFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:0000B000--00011FFF						
N:02EB:00012000--00012FFF	AN:9CD1F000--9CD1FFFF	ns	01	00001000	P:readwrite U:readwrite	notexec
N:02EB:00013000--00013FFF	AN:9CD7D000--9CD7DFFF	ns	01	00001000	P:readwrite U:readwrite	notexec
N:02EB:00014000--B6DB6FFF						
N:02EB:B6DB7000--B6DBAFFF	AN:80713000--80716FFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6DBB000--B6DC3FFF	AN:81517000--8151FFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6DC4000--B6DCCFFF	AN:9CDE0000--9CDE8FFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6DCD000--B6DE0FFF						
N:02EB:B6DE1000--B6DE1FFF	AN:8152E000--8152EFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6DE2000--B6DE6FFF						
N:02EB:B6DE7000--B6DE7FFF	AN:81534000--81534FFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6DE8000--B6E2CFFF						
N:02EB:B6E2D000--B6E2EFFF	AN:9CD9A000--9CD9BFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6E2F000--B6E82FFF						
N:02EB:B6E83000--B6E84FFF	AN:81579000--8157AFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6E85000--B6EBCFFF						
N:02EB:B6EBD000--B6EBDFFF	AN:9CDBB000--9CDBBFFF	ns	01	00001000	P:readwrite U:readonly	exec
N:02EB:B6EBE000--B6EE6FFF						
N:02EB:B6EE7000--B6EE7FFF	AN:9CD74000--9CD74FFF	ns	01	00001000	P:readwrite U:readonly	notexec
N:02EB:B6EE8000--B6EE8FFF	AN:8158B000--8158BFFF	ns	01	00001000	P:readwrite U:readonly	notexec

Table of Contents

Memory Management In Virtualized Systems

- In virtualized systems multiple guest OSes are running in virtual machines under the control of a hypervisor.



Two-Stage MMU

- Due to the fact that multiple guest OSes are sharing one physical memory, a second stage of memory abstraction is introduced in hardware-assisted virtualized systems .

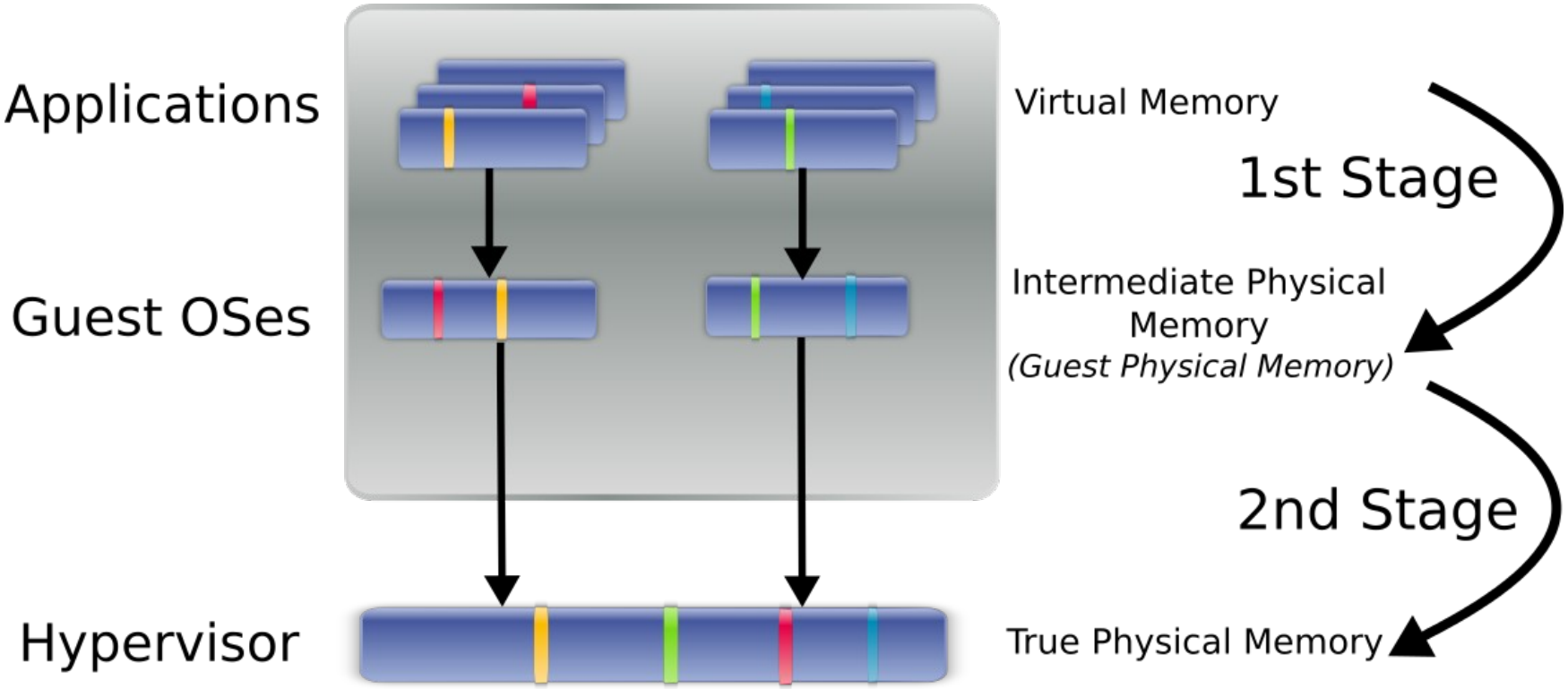
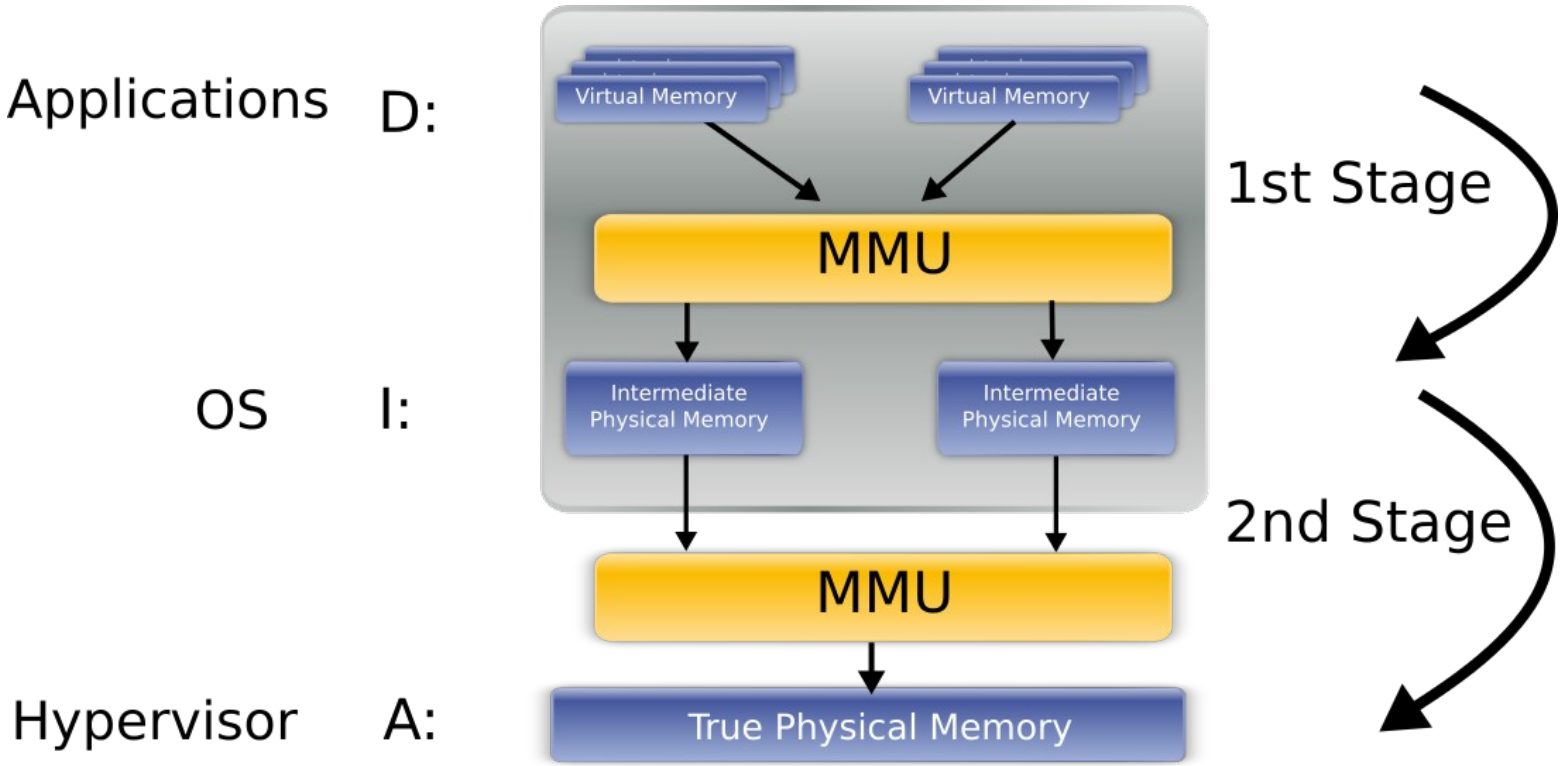


Table of Contents

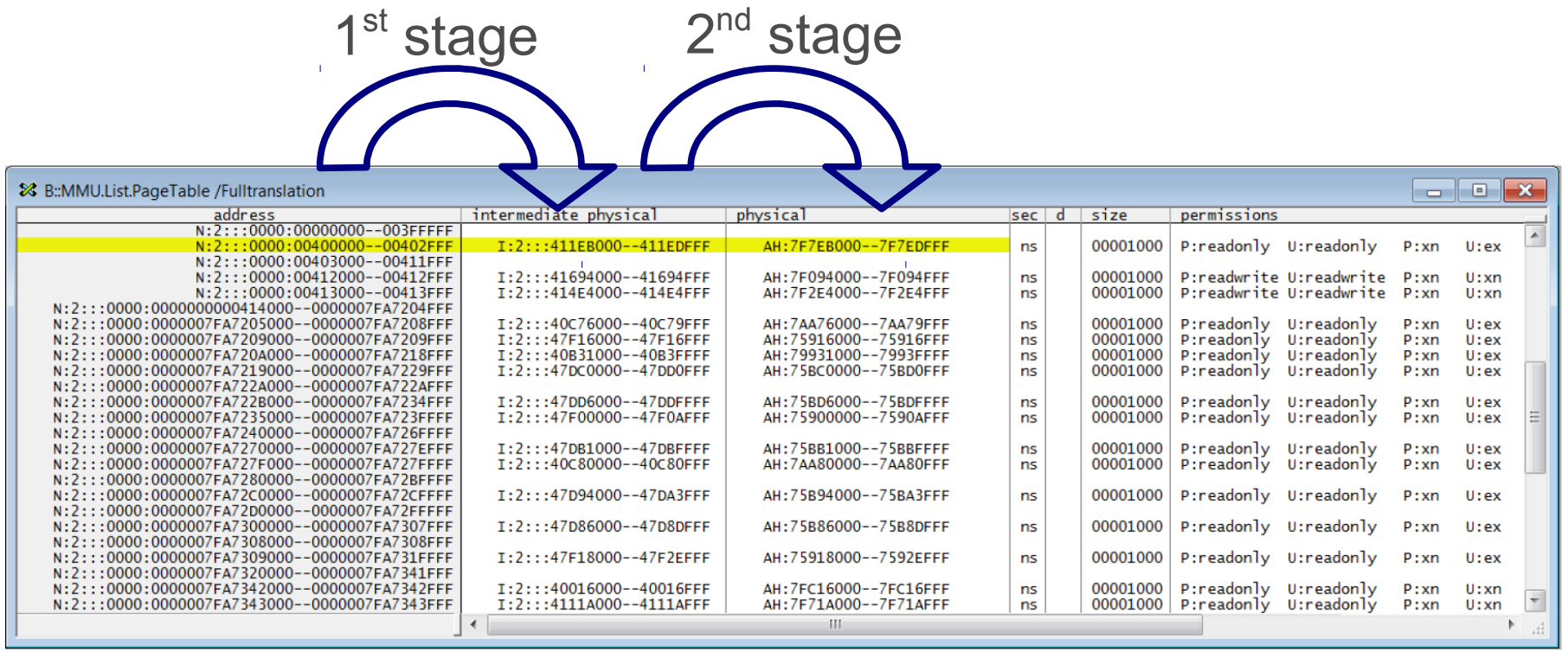
How TRACE32® Supports Two-Stage MMUs

- TRACE32 supports debugging and analysis of the two-stage memory translation.
- TRACE32 allows access to virtual addresses (e.g. with access class D or P), intermediate physical addresses (access class I) and physical addresses (access class A).



MMU.List With Two-Stage Translations

- In the example the virtual address range starting at 0x400000 is mapped to the intermediate physical address 0x411EB000 which is mapped to the absolute address starting at 0x7F7EB00.



address	intermediate physical	physical	sec	d	size	permissions
N:2:::0000:00000000--003FFFFF						
N:2:::0000:00400000--00402FFF	I:2:::411EB000--411EDFFF	AH:7F7EB000--7F7EDFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:00403000--00411FFF						
N:2:::0000:00412000--00412FFF	I:2:::41694000--41694FFF	AH:7F094000--7F094FFF	ns		00001000	P:readwrite U:readwrite P:xn U:xn
N:2:::0000:00413000--00413FFF	I:2:::414E4000--414E4FFF	AH:7F2E4000--7F2E4FFF	ns		00001000	P:readwrite U:readwrite P:xn U:xn
N:2:::0000:0000000000414000--0000007FA7204FFF						
N:2:::0000:0000007FA7205000--0000007FA7208FFF	I:2:::40C76000--40C79FFF	AH:7AA76000--7AA79FFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7209000--0000007FA7209FFF	I:2:::47F16000--47F16FFF	AH:75916000--75916FFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA720A000--0000007FA7218FFF	I:2:::40B31000--40B3FFFF	AH:79931000--7993FFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7219000--0000007FA7229FFF	I:2:::47DC0000--47DD0FFF	AH:75BC0000--75BD0FFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA722A000--0000007FA722AFFF						
N:2:::0000:0000007FA722B000--0000007FA7234FFF	I:2:::47DD6000--47DDFFFF	AH:75BD6000--75BDFFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7235000--0000007FA723FFFF	I:2:::47F00000--47F0AFFF	AH:75900000--7590AFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7240000--0000007FA726FFFF						
N:2:::0000:0000007FA7270000--0000007FA727EFFF	I:2:::47DB1000--47DBFFFF	AH:75BB1000--75BBFFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA727F000--0000007FA727FFFF	I:2:::40C80000--40C80FFF	AH:7AA80000--7AA80FFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7280000--0000007FA72BFFFF						
N:2:::0000:0000007FA72C0000--0000007FA72CFFFF	I:2:::47D94000--47DA3FFF	AH:75B94000--75BA3FFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA72D0000--0000007FA72FFFFF						
N:2:::0000:0000007FA7300000--0000007FA7307FFF	I:2:::47D86000--47D8DFFF	AH:75B86000--75B8DFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7308000--0000007FA7308FFF						
N:2:::0000:0000007FA7309000--0000007FA731FFFF	I:2:::47F18000--47F2EFFF	AH:75918000--7592EFFF	ns		00001000	P:readonly U:readonly P:xn U:ex
N:2:::0000:0000007FA7320000--0000007FA7341FFF						
N:2:::0000:0000007FA7342000--0000007FA7342FFF	I:2:::40016000--40016FFF	AH:7FC16000--7FC16FFF	ns		00001000	P:readonly U:readonly P:xn U:xn
N:2:::0000:0000007FA7343000--0000007FA7343FFF	I:2:::4111A000--4111AFFF	AH:7F71A000--7F71AFFF	ns		00001000	P:readonly U:readonly P:xn U:xn