zdbg

Hypervisor Debugging with r2

Presenter

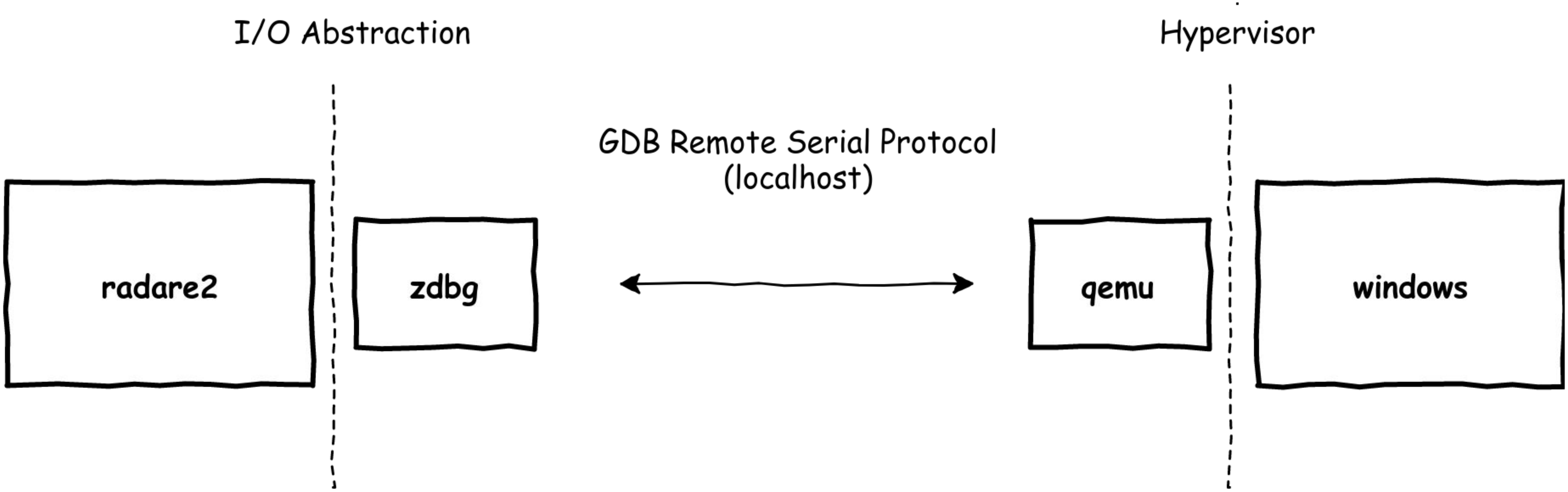
Lars Haukli

@zutle

Date

200 000 000 BC

Symantec.



I/O Abstraction

Hypervisor

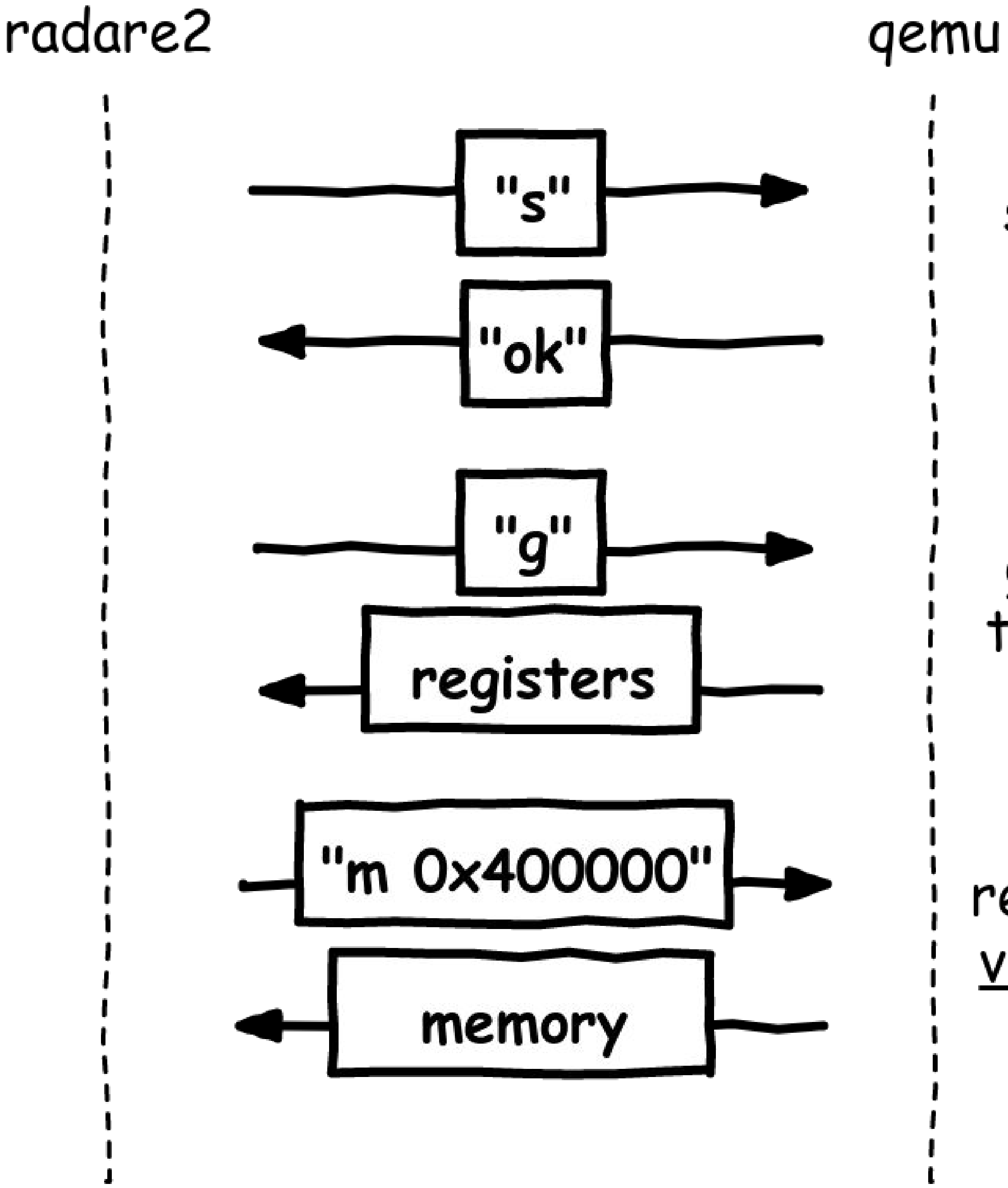
GDB Remote Serial Protocol
(localhost)

radare2

zdbg

qemu

windows



radare2

qemu

"s"

"ok"

"g"

registers

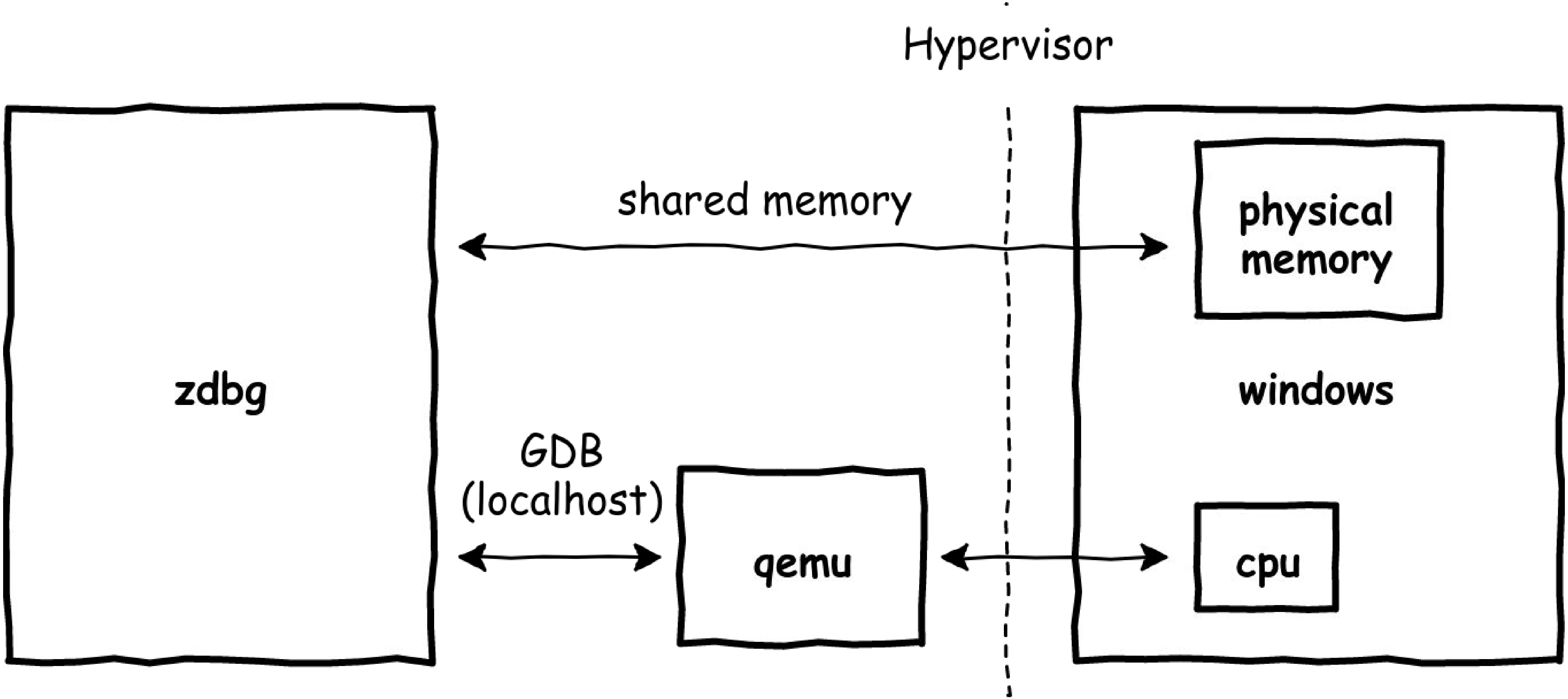
"m 0x400000"

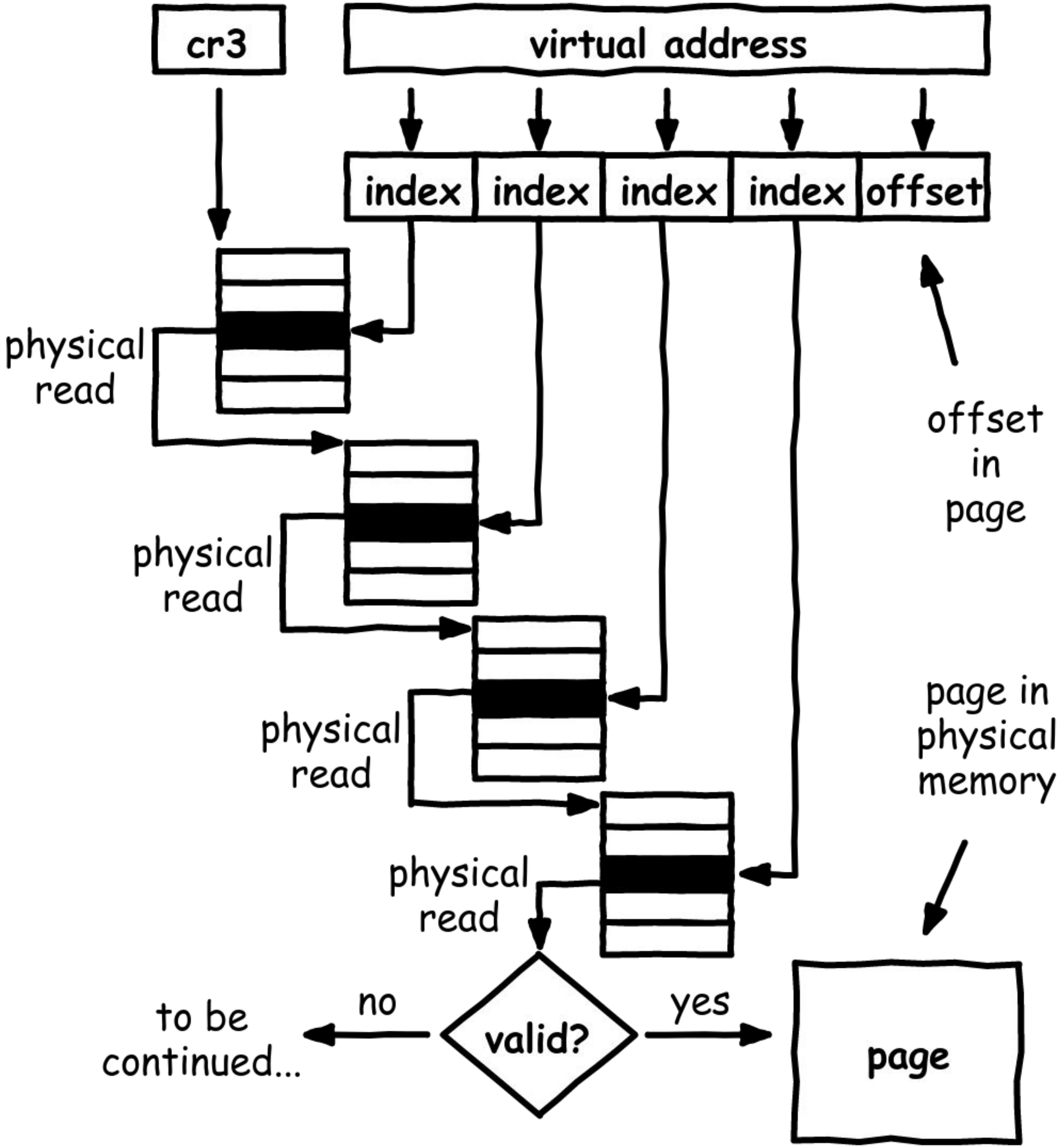
memory

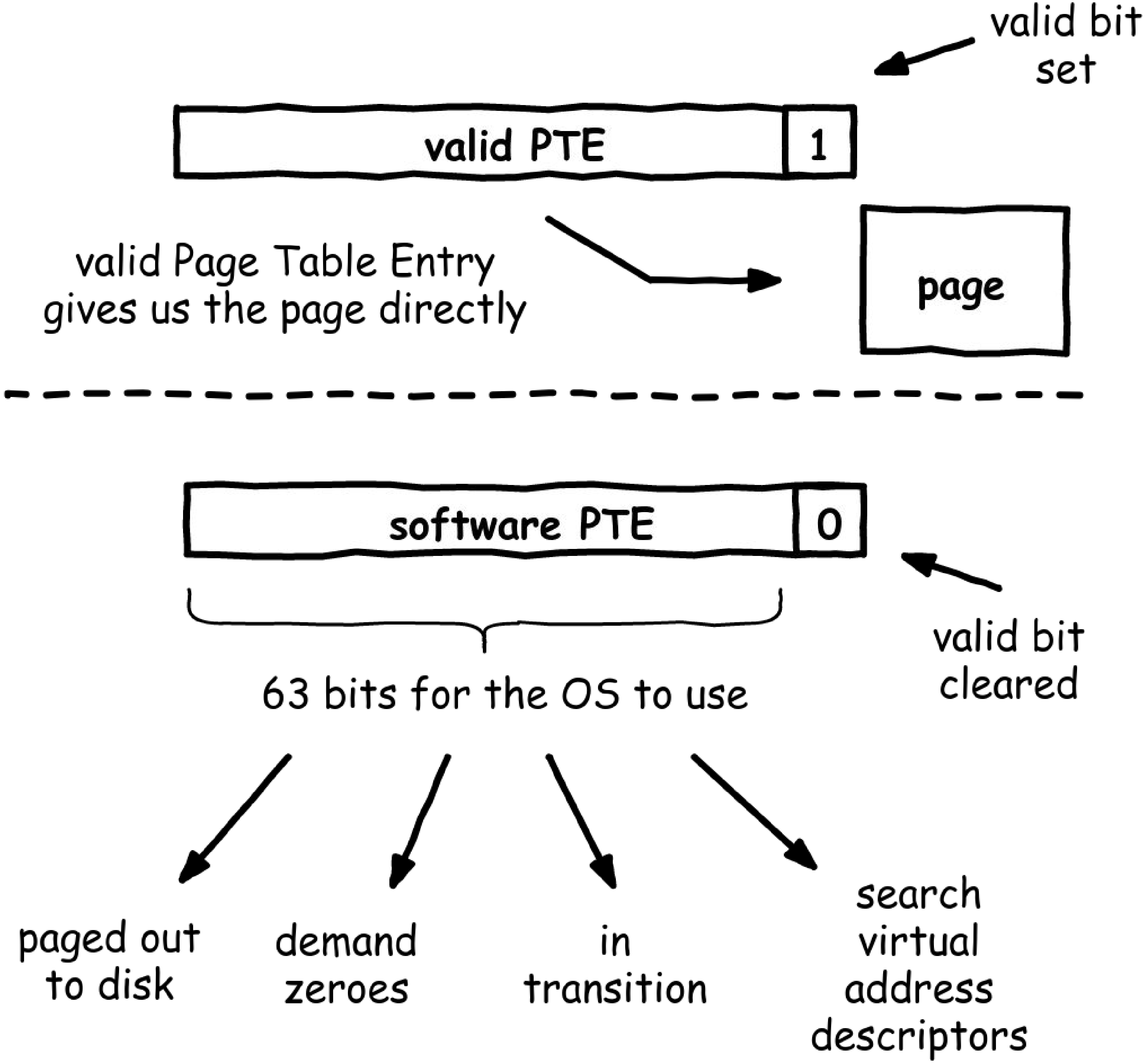
single step CPU

get some of the registers

read memory at virtual address


Hypervisor
shared memory
physical memory
zdbg
windows
GDB (localhost)
qemu
cpu


cr3
virtual address
index
index
index
index
offset
offset in page
physical read
physical read
physical read
physical read
page in physical memory
to be continued...
no
valid?
yes
page



valid bit set

valid PTE

1

valid Page Table Entry gives us the page directly

page

software PTE

0

valid bit cleared

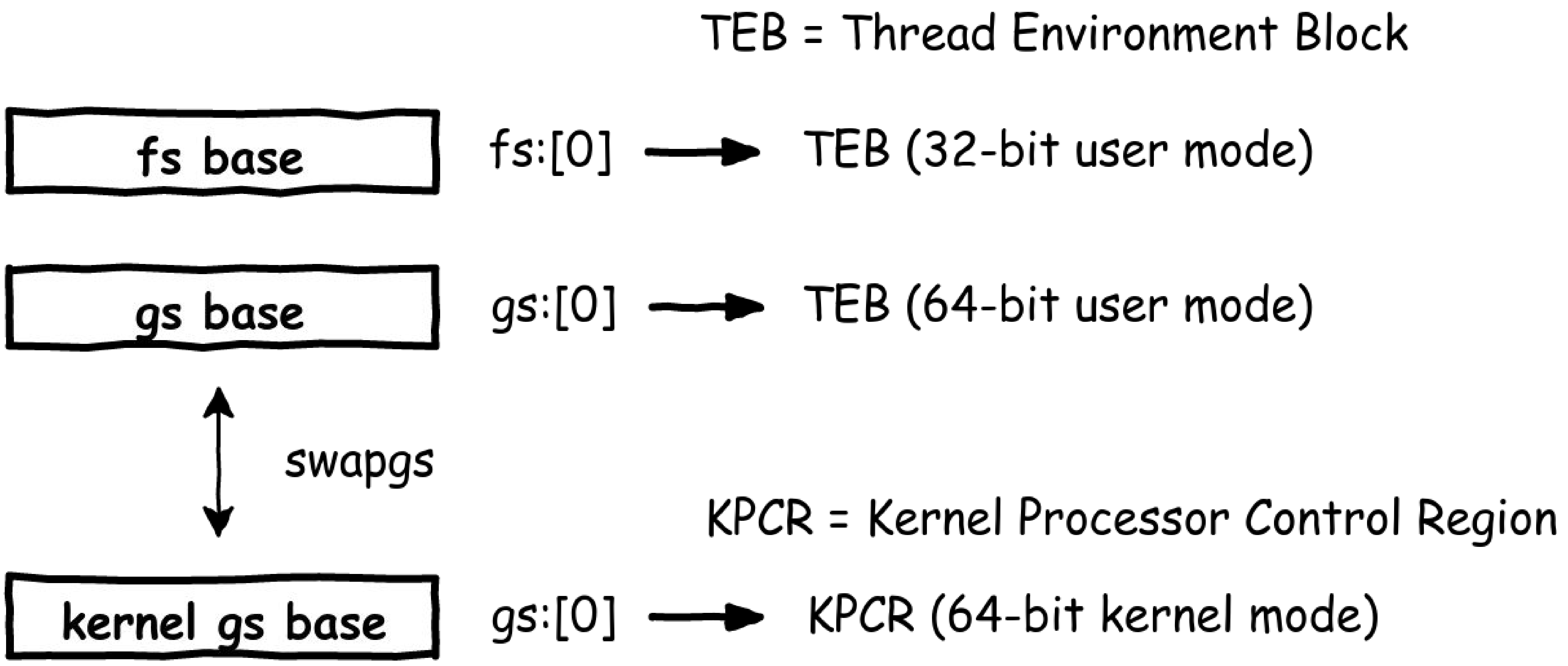
63 bits for the OS to use

paged out to disk

demand zeroes

in transition

search virtual address descriptors



TEB = Thread Environment Block

fs base

fs:[0] → TEB (32-bit user mode)

gs base

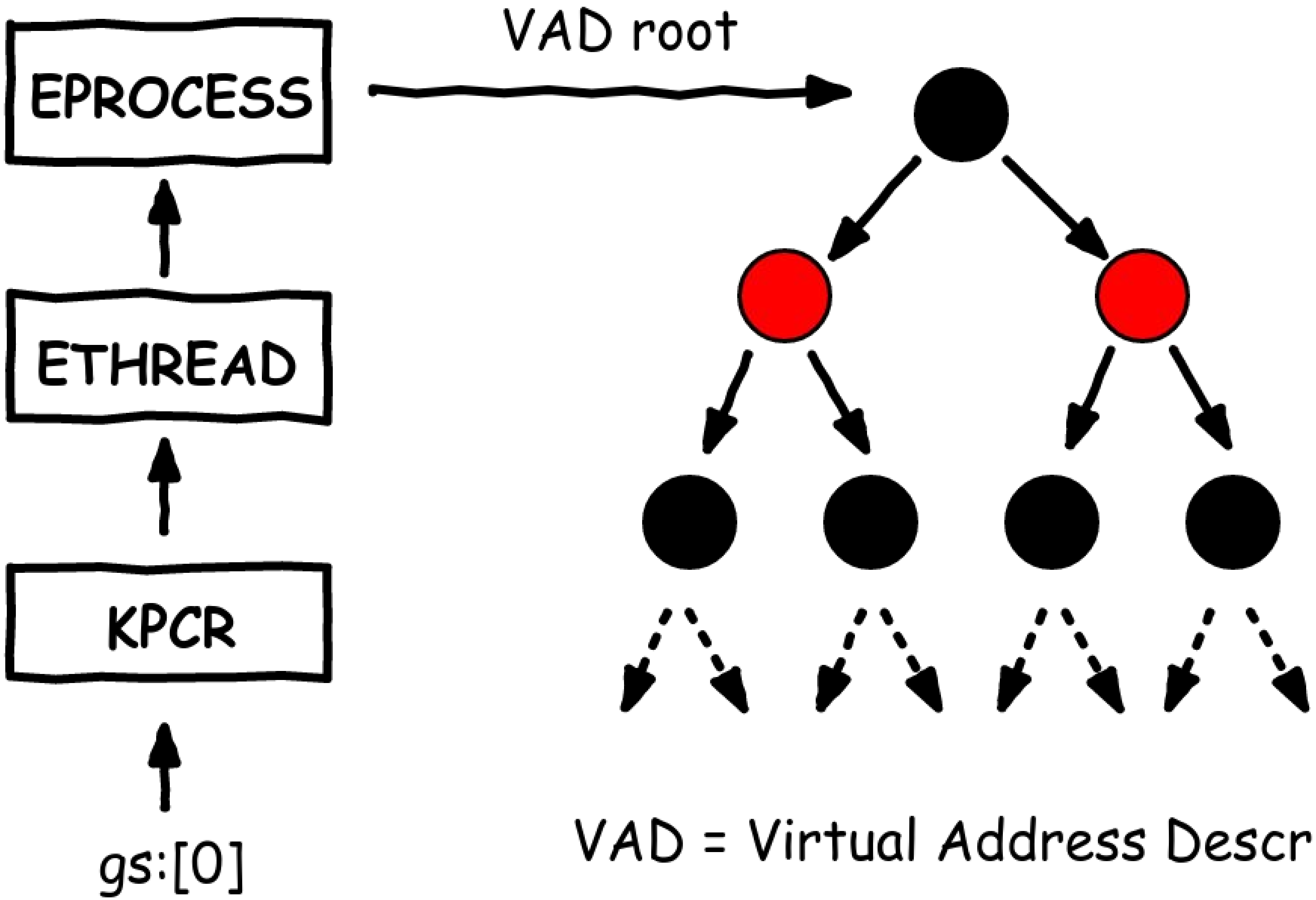
gs:[0] → TEB (64-bit user mode)

swapgs

KPCR = Kernel Processor Control Region

kernel gs base

gs:[0] → KPCR (64-bit kernel mode)



EPROCESS

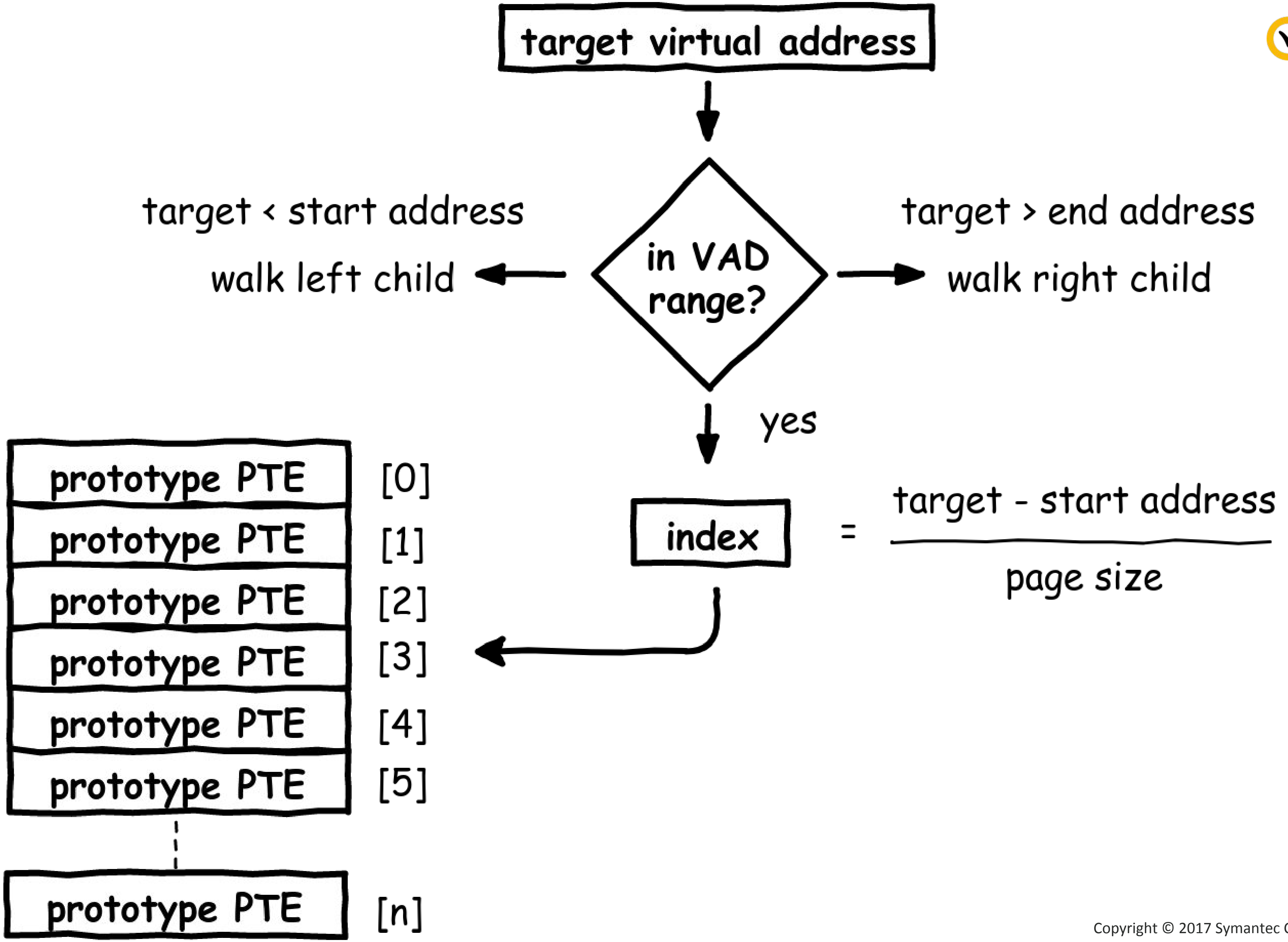
ETHREAD

KPCR

gs:[0]

VAD root

VAD = Virtual Address Descriptor



target virtual address

in VAD range?

target < start address

walk left child

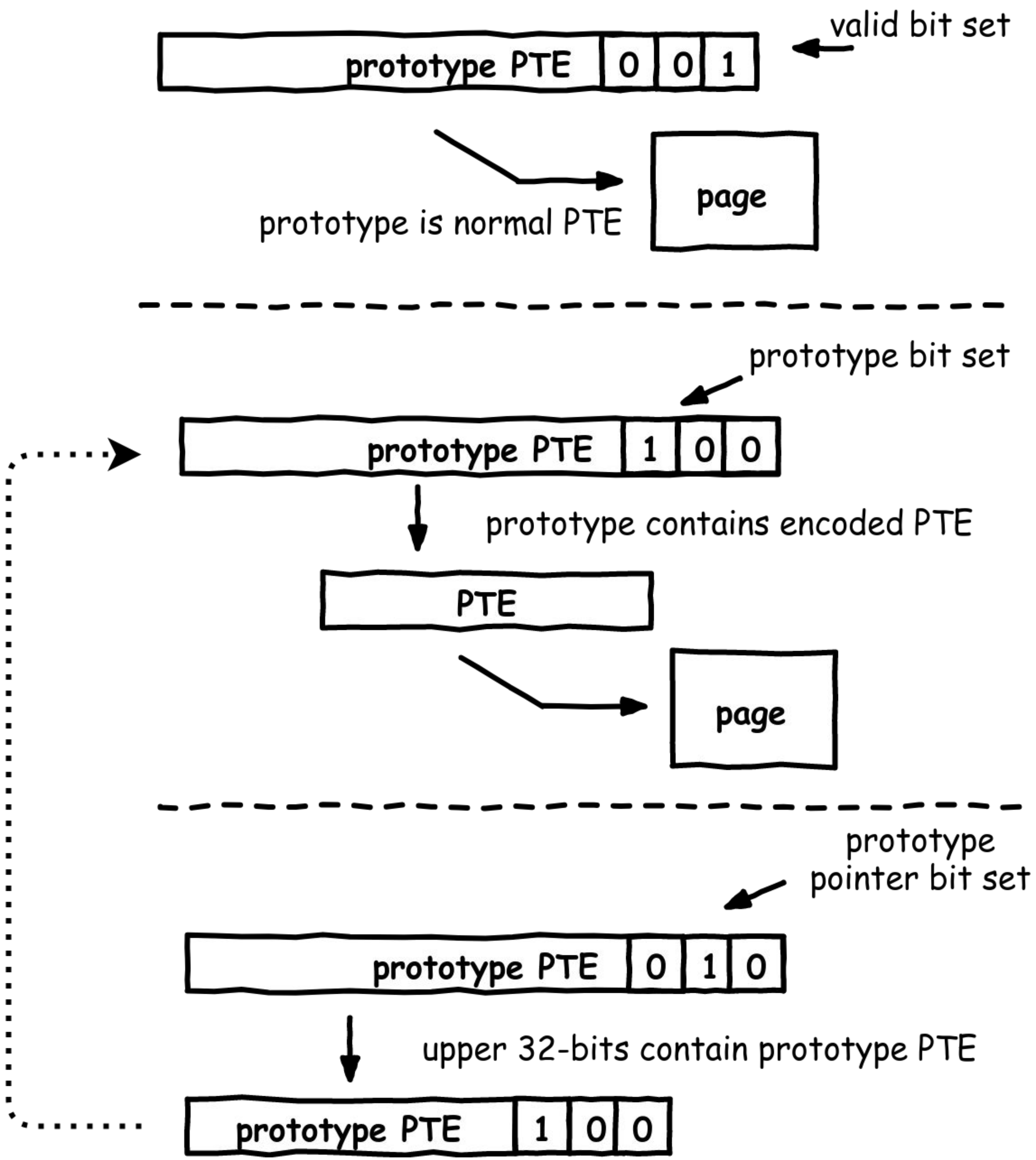
target > end address

walk right child

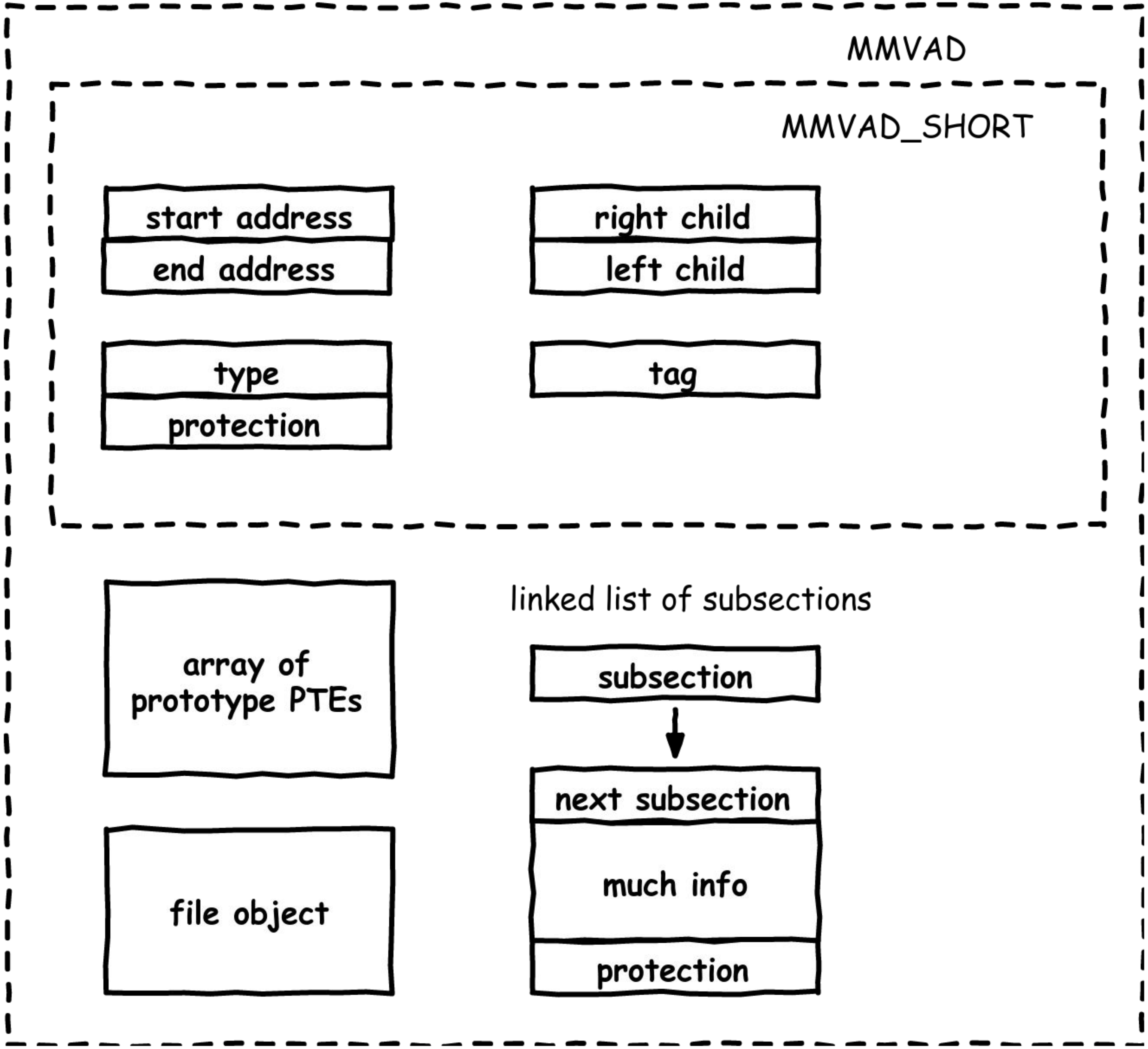
yes

$$\text{index} = \frac{\text{target} - \text{start address}}{\text{page size}}$$

prototype PTE	[0]
prototype PTE	[1]
prototype PTE	[2]
prototype PTE	[3]
prototype PTE	[4]
prototype PTE	[5]
prototype PTE	[n]

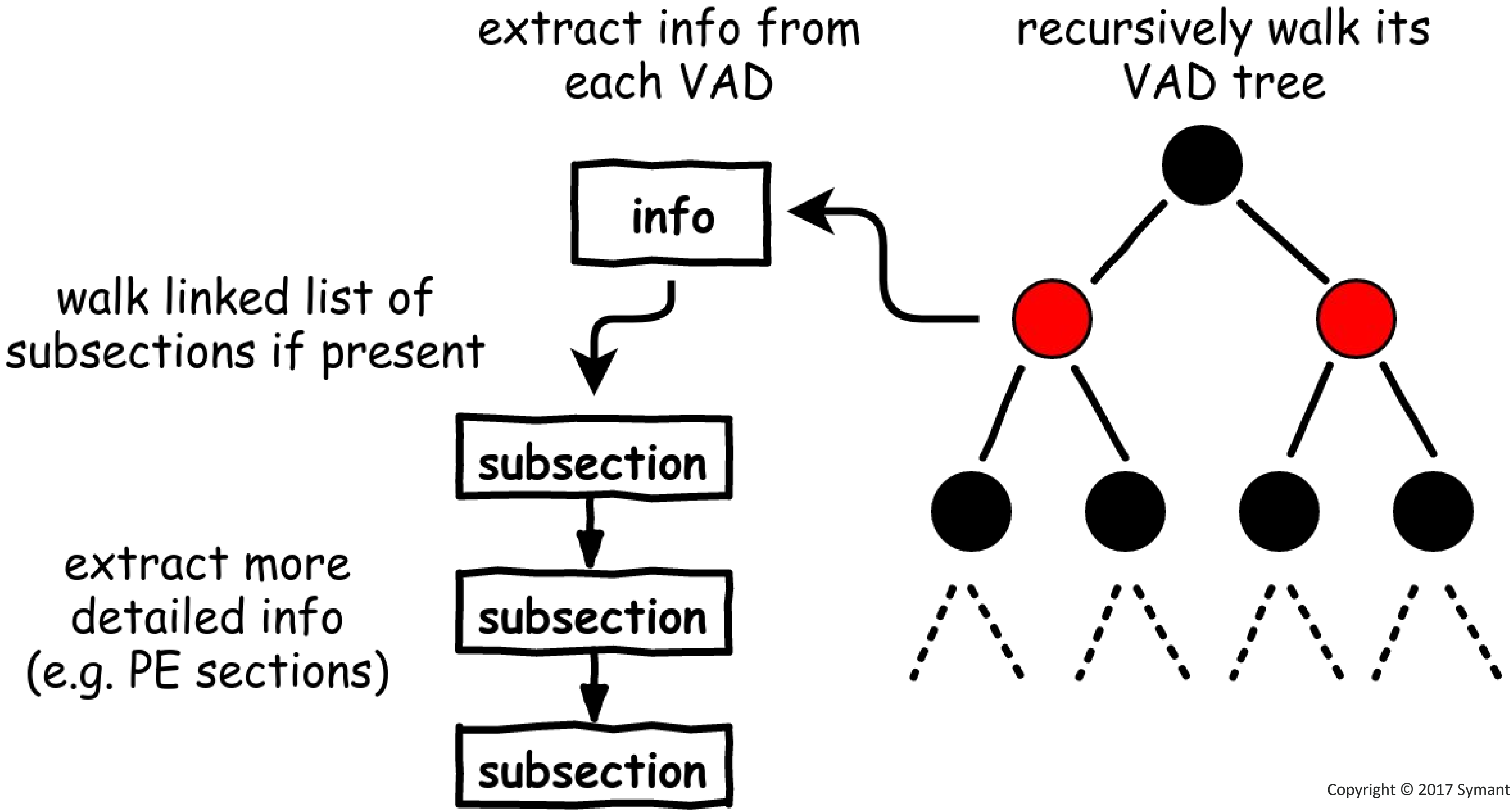


prototype PTE
0
0
1
valid bit set
prototype is normal PTE
page
prototype bit set
prototype PTE
1
0
0
prototype contains encoded PTE
PTE
page
prototype
pointer bit set
prototype PTE
0
1
0
upper 32-bits contain prototype PTE
prototype PTE
1
0
0



MMVAD
MMVAD_SHORT
start address
end address
right child
left child
type
protection
tag
linked list of subsections
array of prototype PTEs
subsection
next subsection
much info
protection
file object

getting the memory map of a process



extract info from each VAD

recursively walk its VAD tree

info

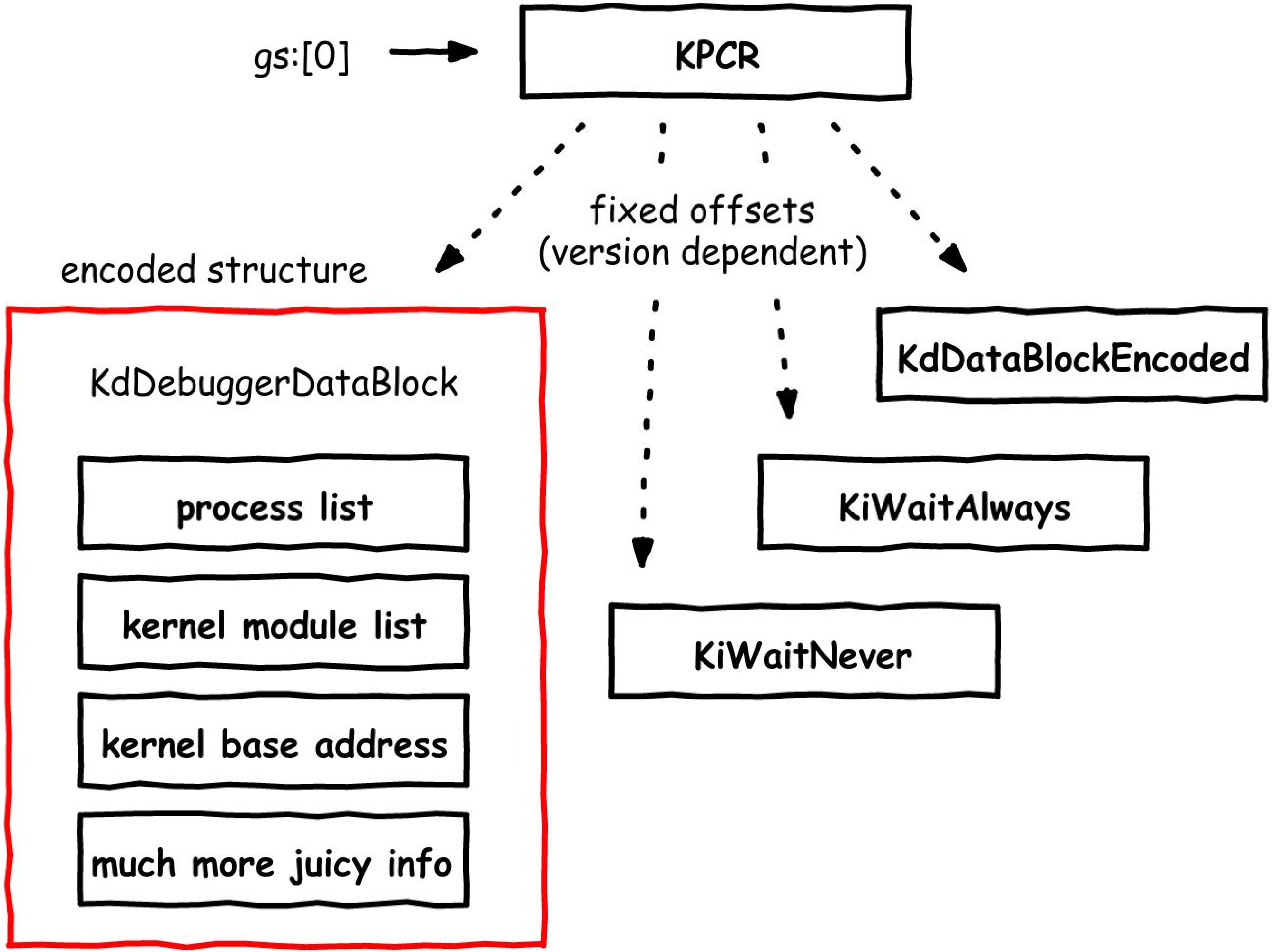
walk linked list of subsections if present

subsection

subsection

subsection

extract more detailed info (e.g. PE sections)



gs:[0]

KPCR

fixed offsets
(version dependent)

encoded structure

KdDebuggerDataBlock

process list

kernel module list

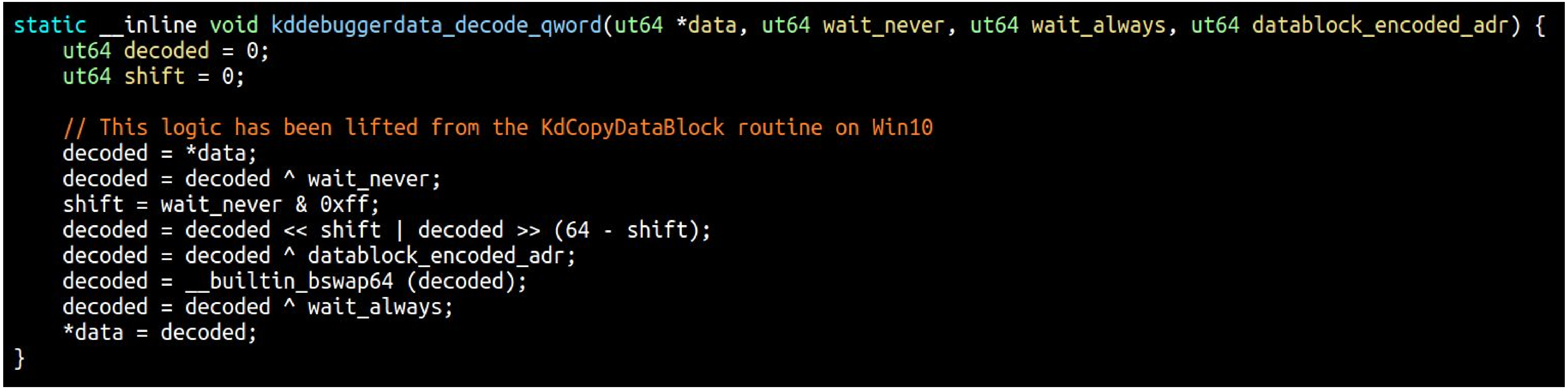
kernel base address

much more juicy info

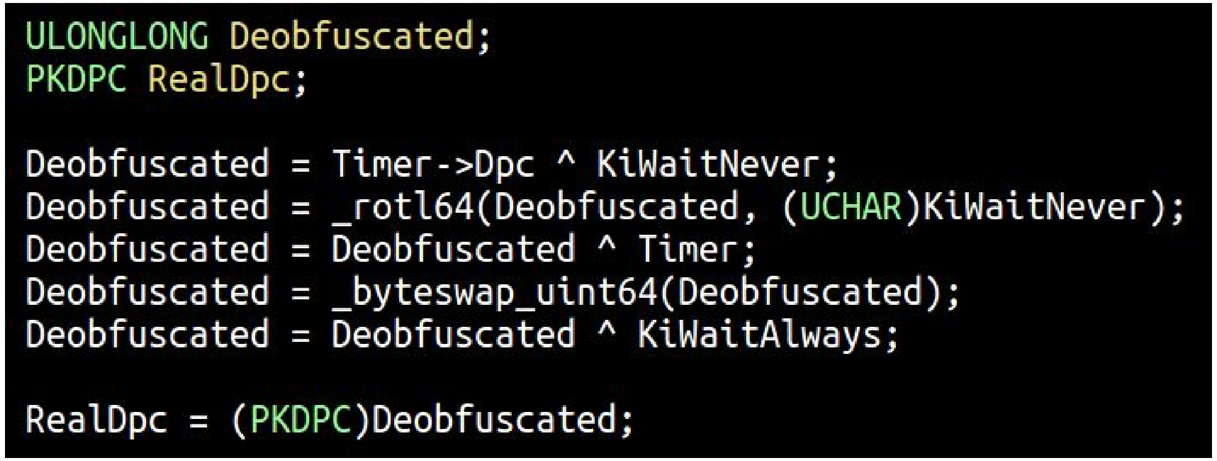
KdDataBlockEncoded

KiWaitAlways

KiWaitNever



```
static __inline void kddebuggerdata_decode_qword(ut64 *data, ut64 wait_never, ut64 wait_always, ut64 datablock_encoded_adr) {
    ut64 decoded = 0;
    ut64 shift = 0;

    // This logic has been lifted from the KdCopyDataBlock routine on Win10
    decoded = *data;
    decoded = decoded ^ wait_never;
    shift = wait_never & 0xff;
    decoded = decoded << shift | decoded >> (64 - shift);
    decoded = decoded ^ datablock_encoded_adr;
    decoded = __builtin_bswap64 (decoded);
    decoded = decoded ^ wait_always;
    *data = decoded;
}
```



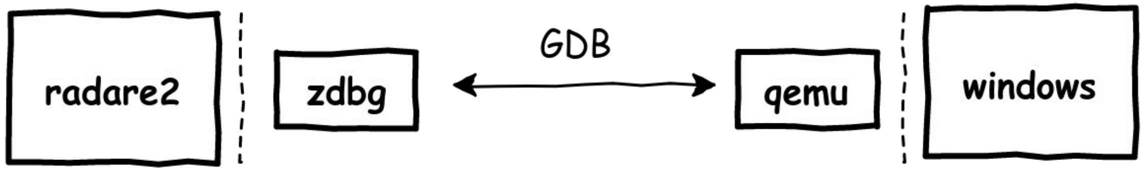
```
ULONGLONG Deobfuscated;
PKDPC RealDpc;

Deobfuscated = Timer->Dpc ^ KiWaitNever;
Deobfuscated = _rotl64(Deobfuscated, (UCHAR)KiWaitNever);
Deobfuscated = Deobfuscated ^ Timer;
Deobfuscated = _byteswap_uint64(Deobfuscated);
Deobfuscated = Deobfuscated ^ KiWaitAlways;

RealDpc = (PKDPC)Deobfuscated;
```

my code (2017)

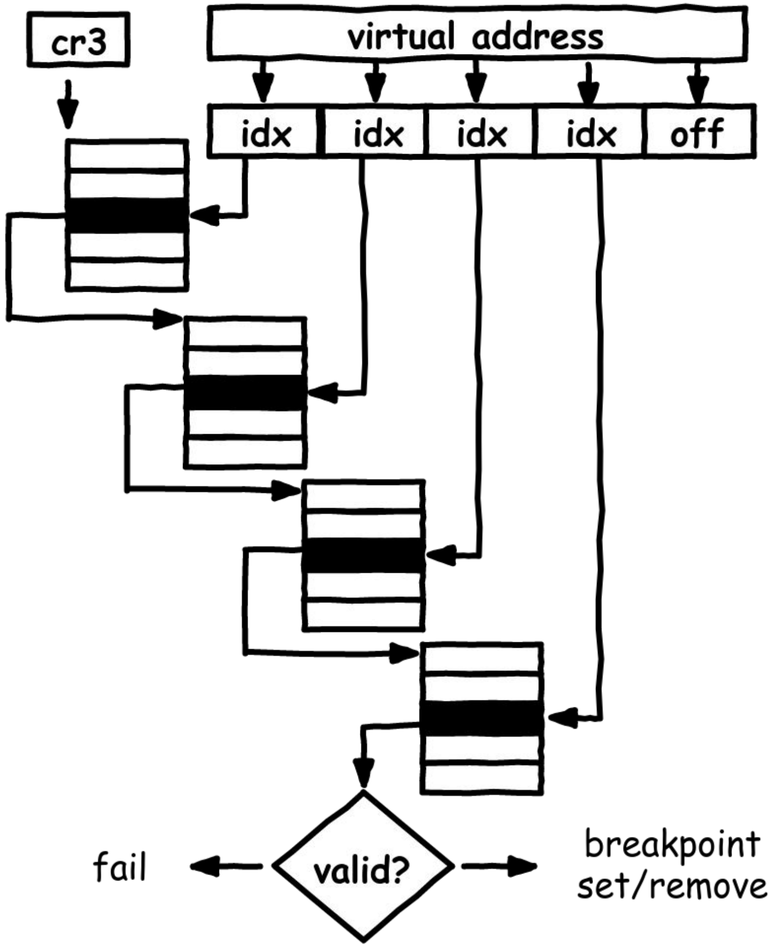
Skywing (2007)

GDB client
requests breakpoint

GDB server
implements breakpoint



qemu must translate virtual address
to read original contents and write int 3 instruction



Thanks!

- Symantec Norway
    - Stian Myhre
    - Bahaa Naamneh
    - and the rest of the team!

- radare2
    - pancake
    - defragger (gdb)
    - The Lemon Man (windbg)
    - inisider (pdb)
    - and all other contributors!

Questions?
More demos? :D

@zutle

Lars Haukli