Kernel Exploitation

And more

Kernel Exploitation - Content

- Kernel Basics
- Kernel Exploitation Basics
- Kernel Address Space
- Hardware Layout
- Virtual vs. Logical vs. Physical Addresses
- Kernel Exploitation Theory
- Remote Kernel Exploit
- Linux Kernel Exploit Mitigations
- Linus and Security related Kernel Development

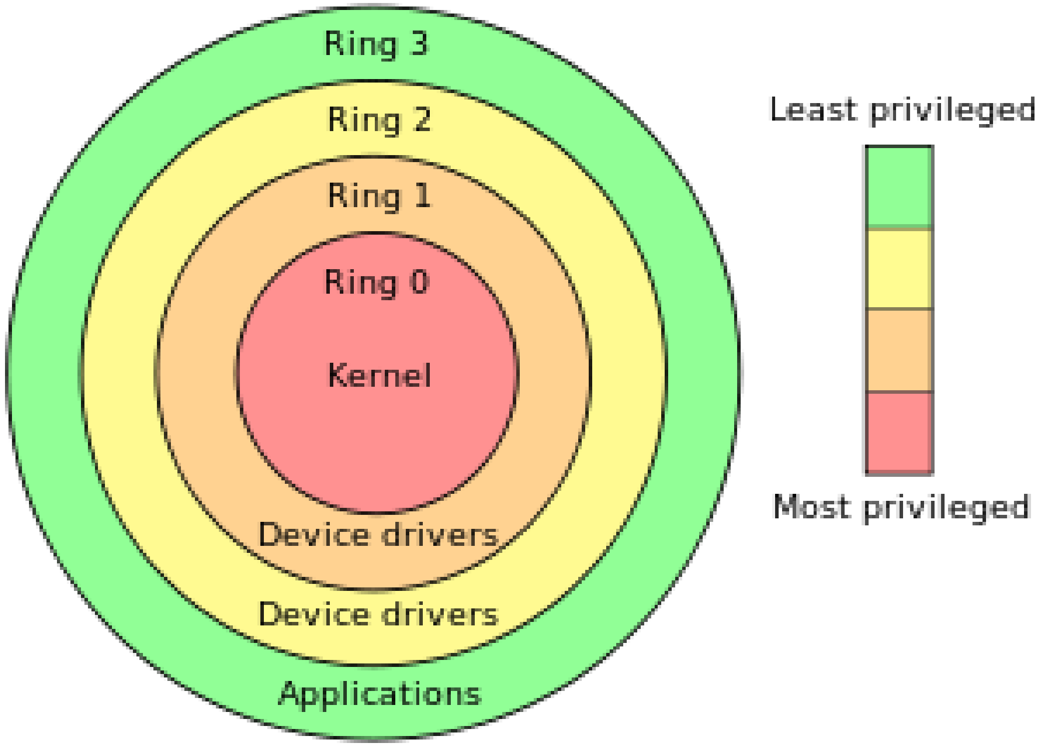
Kernel Exploitation

Kernel Basics

Kernel Basics

Why exploiting the kernel?

- Userspace is restricted
- Kernel has access to everything
  - All processes (root processes)
  - All secrets (harddisk password)
  - All security mechanisms (SELinux, Seccomp-bpf)
- Kernel Attack Surface is wide open
  - Containerization (Docker, LXC)



Kernel Basics – Kernel Mode

Kernel Mode / Supervisor Mode / Unrestricted Mode / System Mode

- Access to all memory
- Access to special CPU registers

User Mode / Non-Privileged Mode / Restricted Mode

Kernel Basics – Process Context

Execution Context:

- Kernel can be executed («supervisor mode»)
  - **Interrupt context**: no backing process (e.g. Received a network packet)
  - **Process context**: backing process (e.g. Syscall)
    - Have a descriptor ready available with info about process – registers, uid etc.
    - Can access data of process, as its TLB is loaded – for Shellcode

How to get Kernel Execution – Dev Way

Linux:

- Write LKM (Linux Kernel Module)
- Load as Root
- Redhat 7: *"When Secure Boot is enabled, the EFI operating system boot loaders, the Red Hat Enterprise Linux kernel, and **all kernel modules must be signed with a private key** and authenticated with the corresponding public key"*

Windows:

- Reboot in unsafe / development mode
- Or: Sign code with Driver Certificate ($$$ to Microsoft)
- -> **No Untrusted (unsigned) Code in Windows Kernel**!

Kernel Exploiting – Things to consider

Difficulties in Exploiting:

- If exploit crash -> Crash the system ☹
- No simple system() shellcode
  - Spawning new processes is hard
  - Traverse memory to find process handle, set uid = 0
- No brute force
  - E.g. ASLR

Easier Exploiting:

- Information disclosure is easier (local)
- Kernel ASLR (kASLR) is hard to implement
- Attack surface is gigantic (local)

Kernel Exploiting – Use-Cases

Use-cases

- Mobile (Android, iOS) exploiting / jailbreaking (App -> Root)
- Local privilege escalation (www-data Apache, non-localadmin)
- Pwning the cloud (containerization)
- Rootkits (post breach persistence / hiding)
- Backdoors (gain access again on compromised host)
- Cheats (PC, Console)

Kernel Exploitation

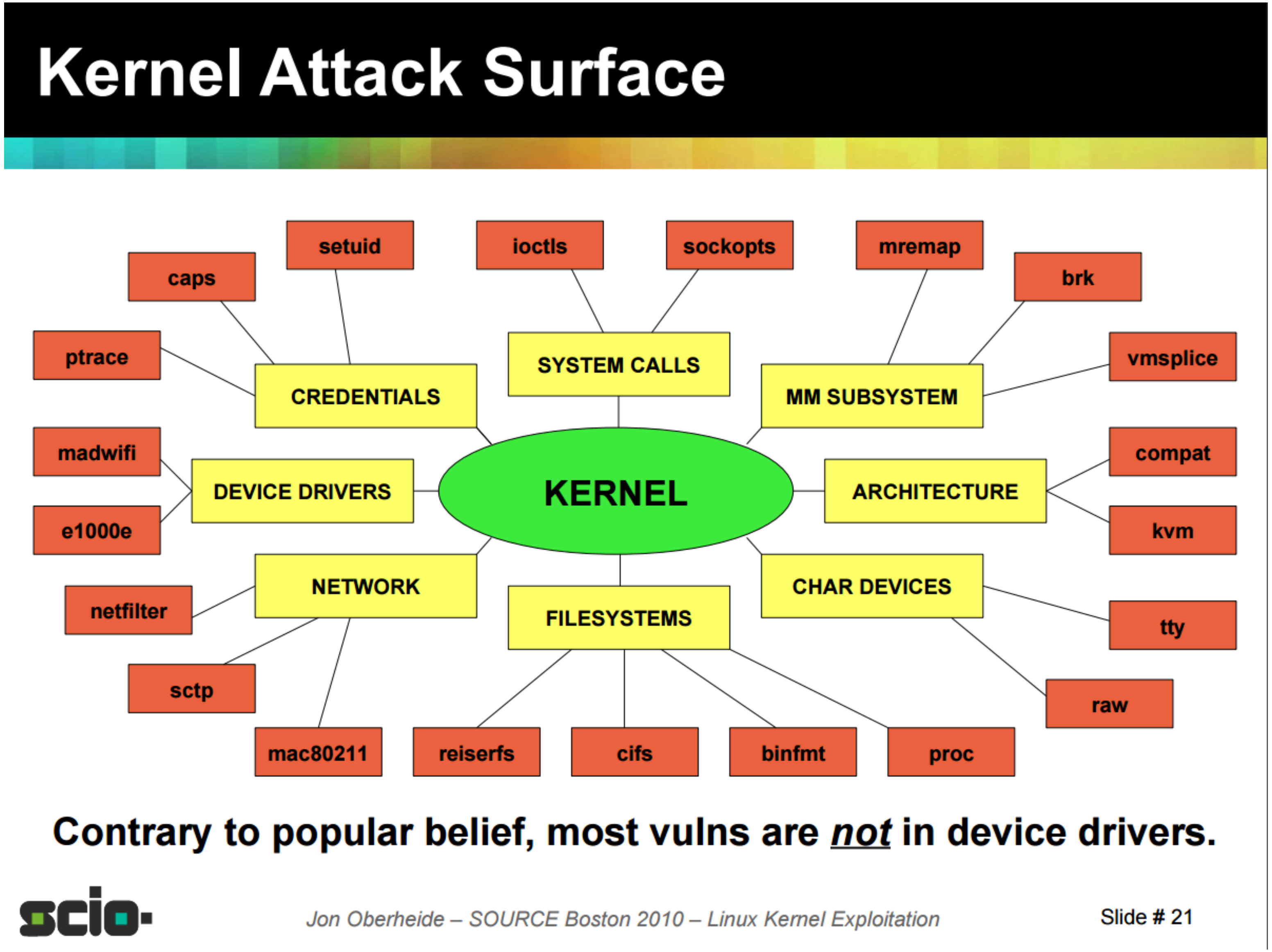
Local Kernel Exploits

Local Kernel Exploits

Attack Surface examples:

- Drivers
- File Systems
- Sockets
- Syscalls
- /proc, /sys

Kernel Exploitation – Attack Surface



Kernel Exploitation - Syscalls

38 sys_rename
39 sys_mkdir
40 sys_rmdir
41 sys_dup
42 sys_pipe
43 sys_times
44 sys_prof [sys_ni_syscall]
45 sys_brk
46 sys_setgid
47 sys_getgid
48 sys_signal
49 sys_geteuid
50 sys_getegid
51 sys_acct
52 sys_umount2 [sys_umount] (2.2+)
53 sys_lock [sys_ni_syscall]
54 sys_ioctl
55 sys_fcntl
56 sys_mpx [sys_ni_syscall]
57 sys_setpgid
58 sys_ulimit [sys_ni_syscall]
59 sys_oldolduname
60 sys_umask
61 sys_chroot
62 sys_ustat
63 sys_dup2
64 sys_getppid
65 sys_getpgrp
66 sys_setsid
67 sys_sigaction
68 sys_sgetmask
69 sys_ssetmask

108 sys_fstat [sys_newfstat]
109 sys_olduname [sys_uname]
110 sys_iopl
111 sys_vhangup
112 sys_idle
113 sys_vm86old
114 sys_wait4
115 sys_swapoff
116 sys_sysinfo
117 sys_ipc
118 sys_fsync
119 sys_sigreturn
120 sys_clone
121 sys_setdomainname
122 sys_uname [sys_newuname]
123 sys_modify_ldt
124 sys_adjtimex
125 sys_mprotect
126 sys_sigprocmask
127 sys_create_module
128 sys_init_module
129 sys_delete_module
130 sys_get_kernel_sy
131 sys_quotactl
132 sys_getpgid
133 sys_fchdir
134 sys_bdflush
135 sys_sysfs
136 sys_personality
137 sys_afs_syscall [sy
138 sys_setfsuid
139 sys_setfsgid

178 sys_rt_sigqueueinfo (2.2+)
179 sys_rt_sigsuspend (2.2+)
180 sys_pread (2.2+)
181 sys_pwrite (2.2+)
182 sys_chown (2.2+)
183 sys_getcwd (2.2+)
184 sys_capget (2.2+)
185 sys_capset (2.2+)
186 sys_sigaltstack (2.2+)
187 sys_sendfile (2.2+)
188 sys_getpmsg [sys_ni_syscall]
189 sys_putpmsg [sys_ni_syscall]
190 sys_vfork (2.2+)

syzkaller - kernel fuzzer

build passing

`syzkaller` is an unsupervised coverage-guided kernel fuzzer. `Linux` kernel fuzzing has the most support, `akaros`, `freebsd`, `fuchsia`, `netbsd` and `windows` are supported to varying degrees.

Kernel Exploiting – Kernel and User Memory

Linux: copy FROM Kernelspace TO Userspace

```
int copy_to_user(void *dst, const void *src, unsigned int size);
```

Linux Kernel Exploits

https://github.com/mzet-/linux-exploit-suggester

linux-exploit-suggester.sh aims to contain list of all publicly known Linux kernel exploits applicable for kernels 2.6 and up

```
[mz@katana linux-exploit-suggester]$ ./linux-exploit-suggester.sh -k 3.1

Kernel version: 3.1
Architecture:
Distribution:
Package list:

Possible Exploits:

[+] [CVE-2012-0056] memodipper

   Details: https://git.zx2c4.com/CVE-2012-0056/about/
   Tags: ubuntu=10.04|11.10
   Download URL: https://git.zx2c4.com/CVE-2012-0056/plain/mempodipper.c

[+] [CVE-2013-2094] perf_swevent

   Details: http://timetobleed.com/a-closer-look-at-a-recent-privilege-escalation-bug-in-linux-cve-2013-2094/
   Tags: RHEL=6,ubuntu=12.04
   Download URL: https://www.exploit-db.com/download/26131

[+] [CVE-2013-2094] perf_swevent 2

   Details: http://timetobleed.com/a-closer-look-at-a-recent-privilege-escalation-bug-in-linux-cve-2013-2094/
   Tags: ubuntu=12.04
   Download URL: https://cyseclabs.com/exploits/vnik_v1.c

[+] [CVE-2013-0268] msr

   Details: https://www.exploit-db.com/exploits/27297/
   Download URL: https://www.exploit-db.com/download/27297
```

Linux Local Kernel Exploit – Dirty Cow

Dirty-Cow: Logic Bug / Race Condition

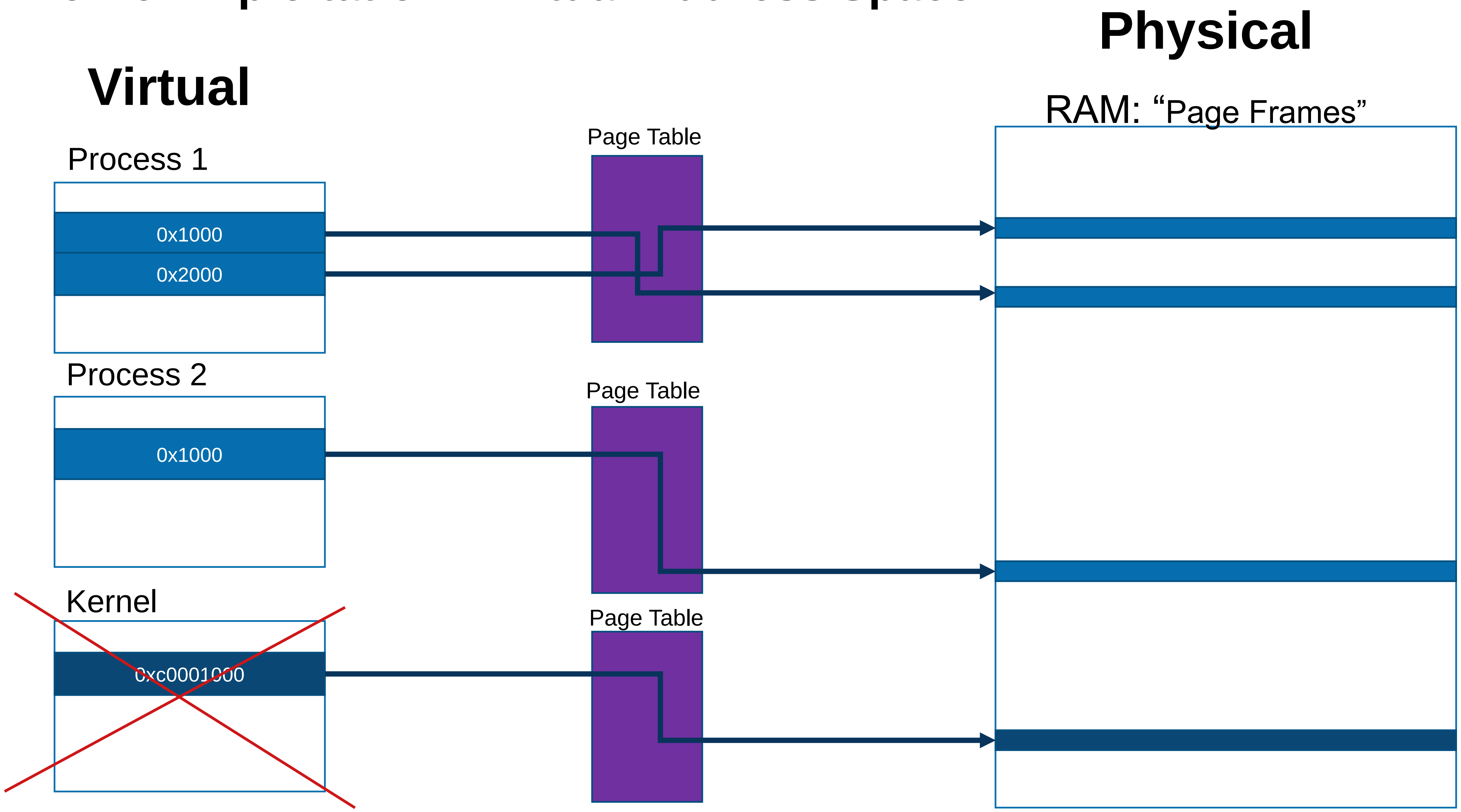
"A race condition was found in the way the Linux kernel's memory subsystem handled the copy-on-write (COW) breakage of private read-only memory mappings.

An unprivileged local user could use this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system."

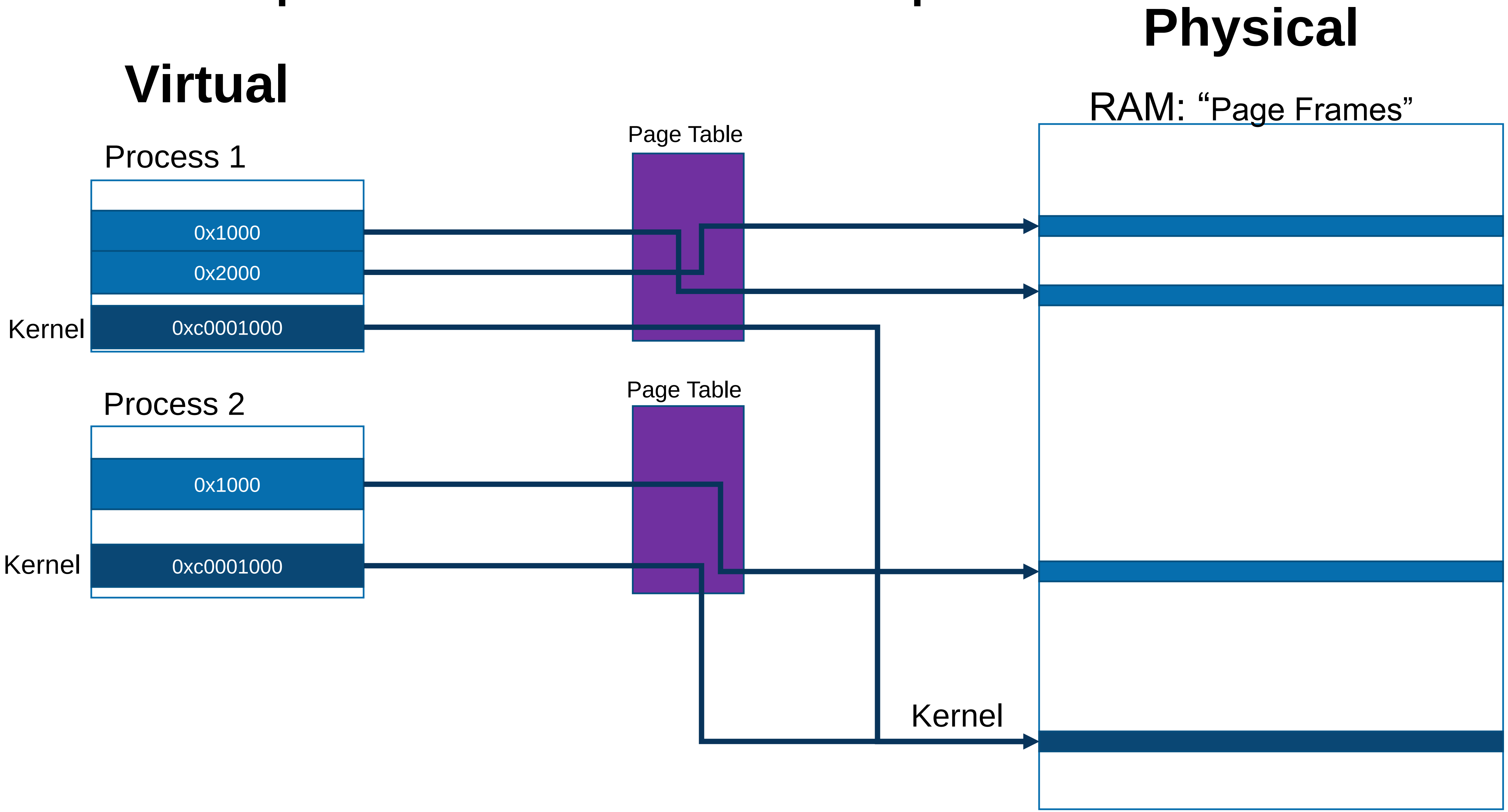
Kernel Address Space

Via TLB

Kernel Exploitation - Virtual Address Space



Kernel Exploitation - Virtual Address Space



Kernel Exploiting – Virtual Addresses

Virtual / Logical Address -> Real / Physical Address translation

- Via Page-Table
- Stored in register CR3
- Per-process
  - But kernel always included

- Virtual / Logic Address
  - What the processes see
  - What the kernel sees

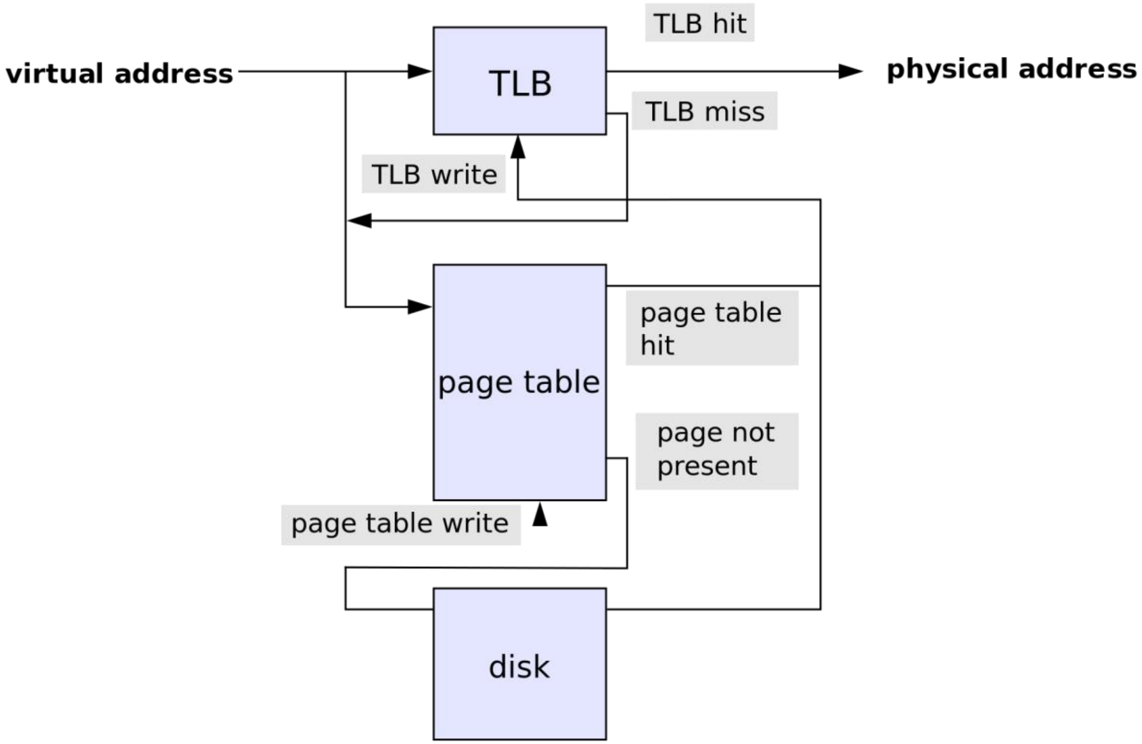
- Physical Address:
  - CPU untranslated
  - What the CPU see’s on the bus

Page Table / TLB

Page Table: Map virtual addresses to physical

TLB: Translation Lookaside Buffer

- Cache in MMU (Memory Management Unit)



Graphics from https://en.wikipedia.org/wiki/Page_table

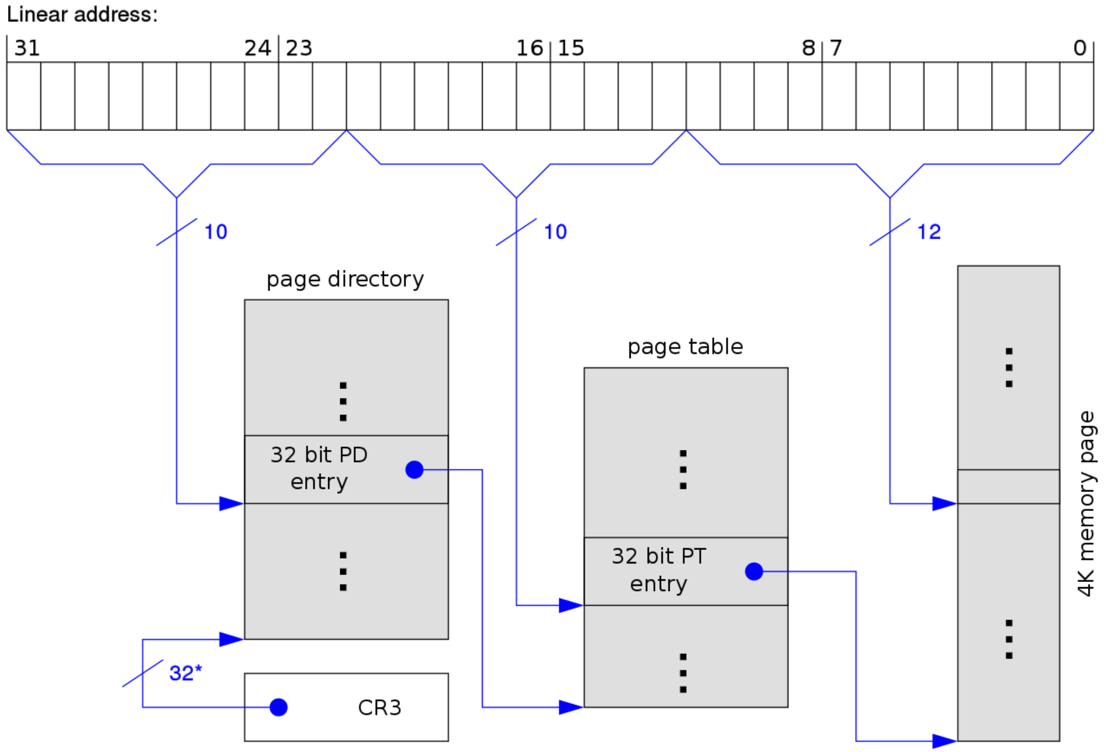
Page Table / TLB

Page Table: Map virtual addresses to physical

TLB: Translation Lookaside Buffer

- Cache in MMU (Memory Management Unit)

Multilevel Page Table



*) 32 bits aligned to a 4-KByte boundary

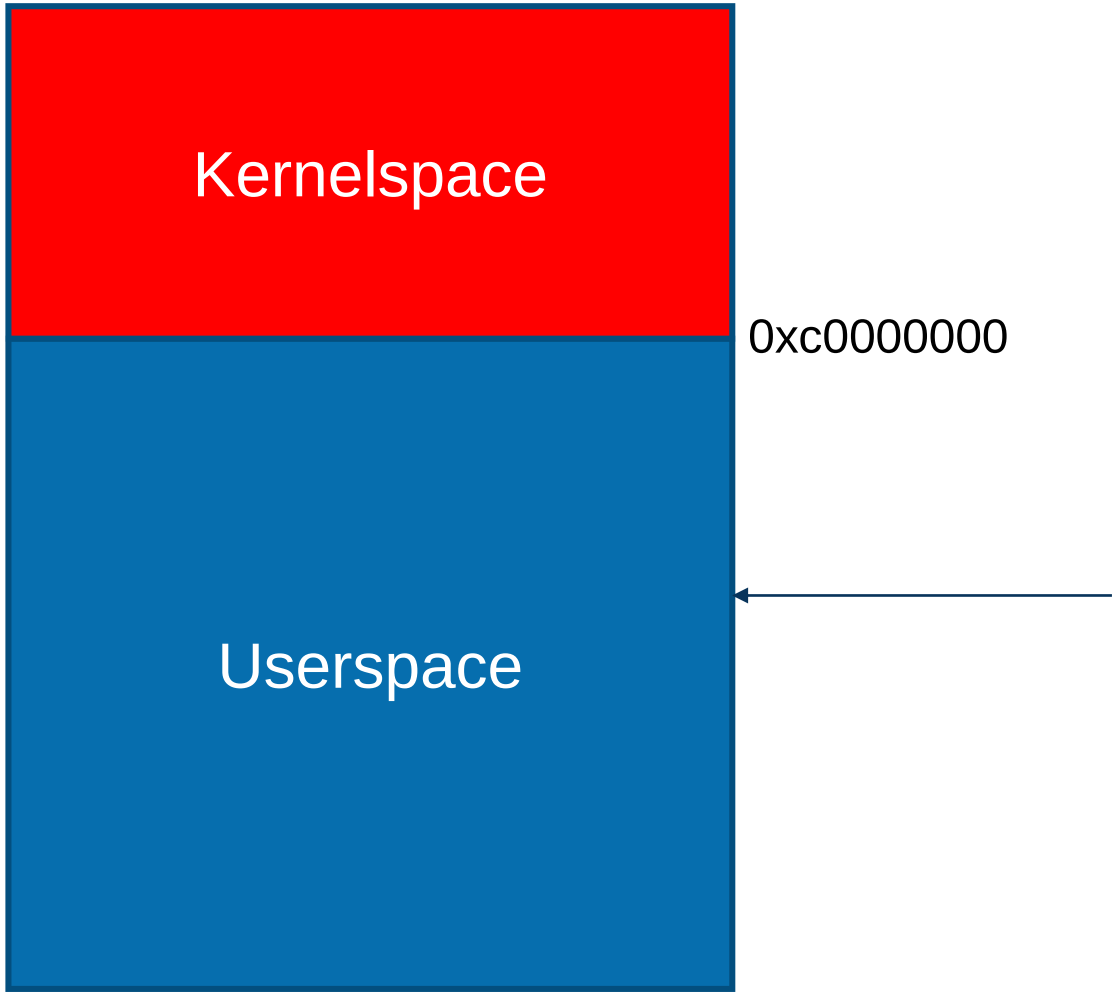
Graphics from https://en.wikipedia.org/wiki/Page_table

In-Userspace

Virtual:

Process

4GB



3GB

0xc0000000

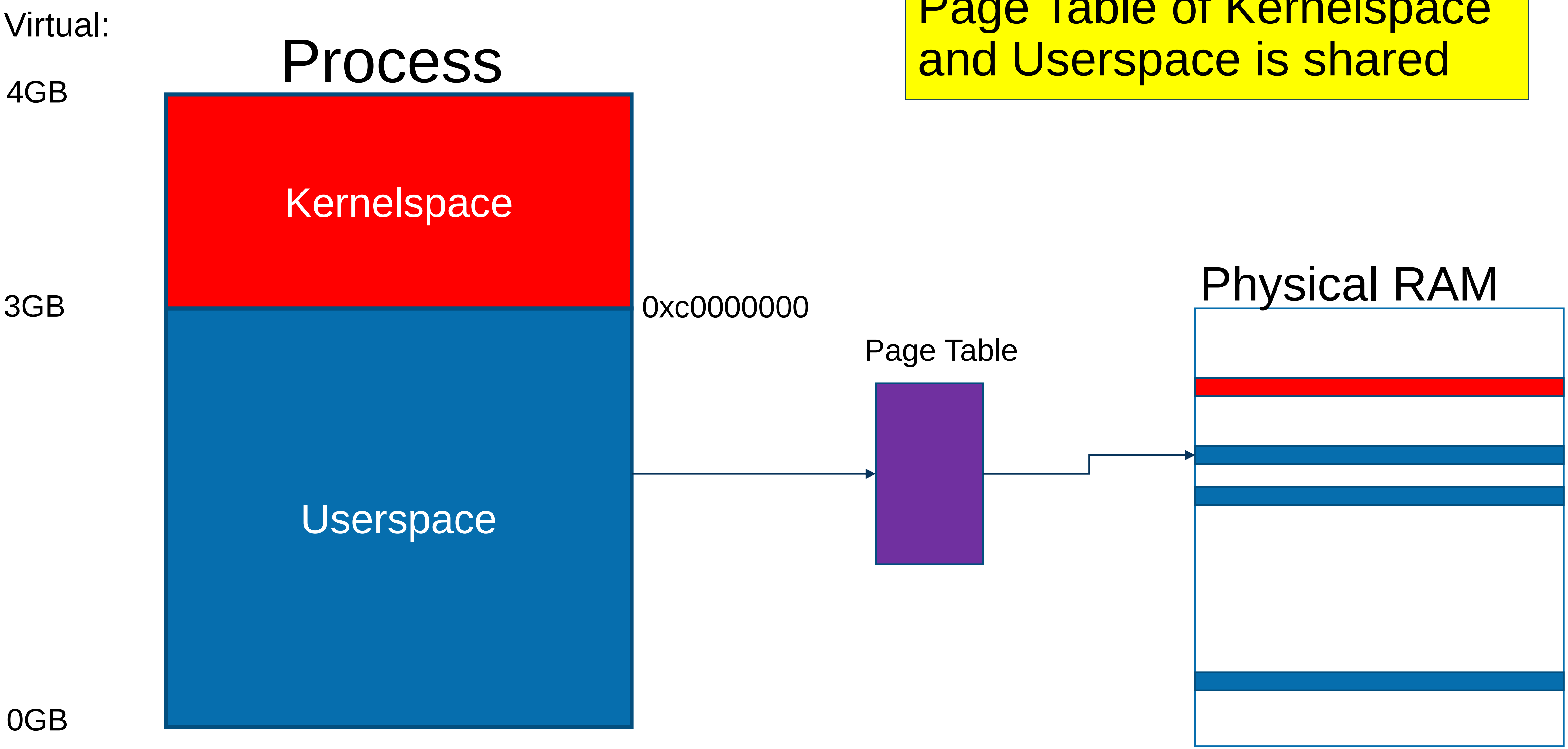
Current Process

0GB

Page Table of Kernelspace and Userspace is shared

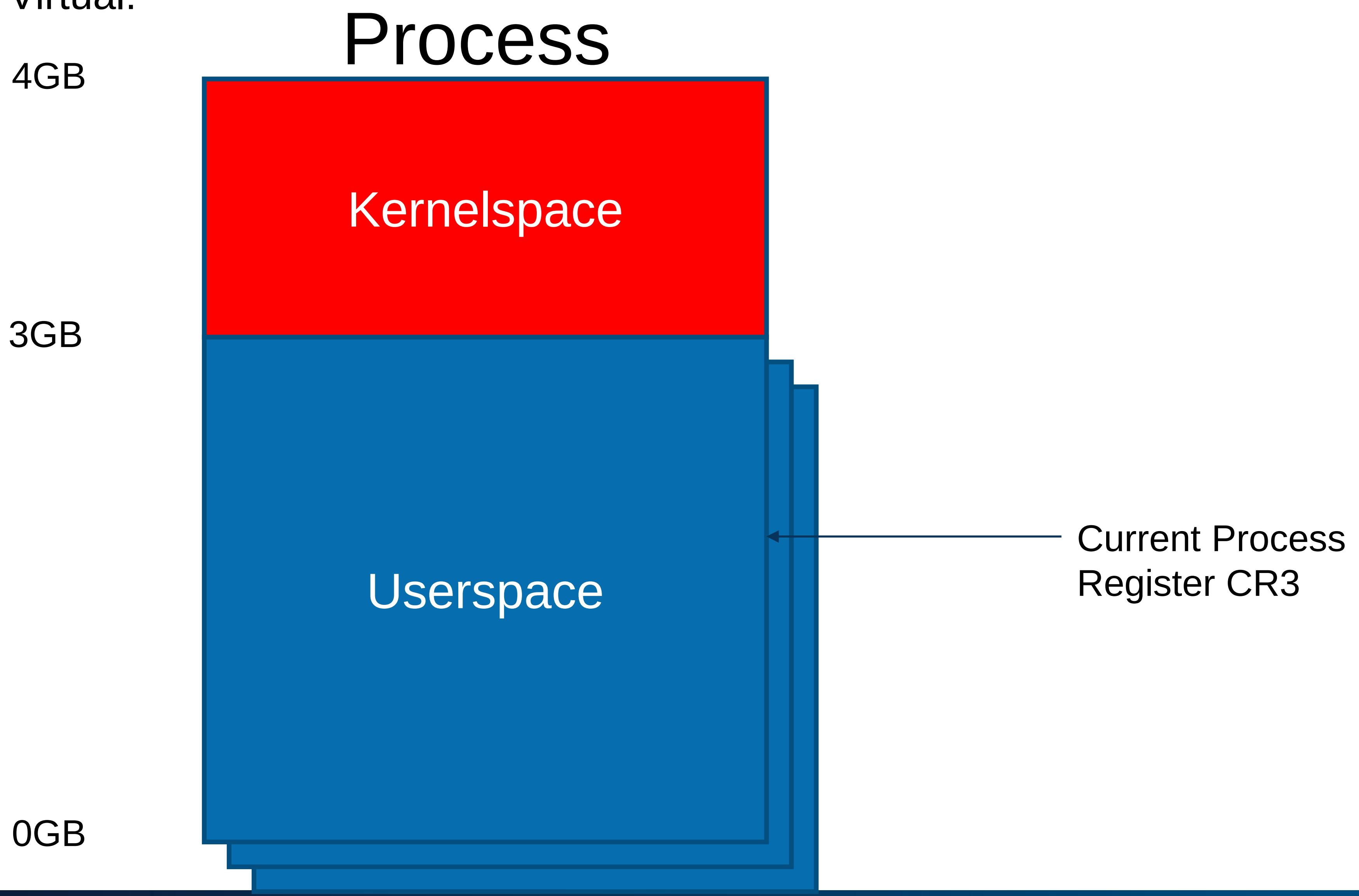
In-Userspace

Page Table of Kernelspace and Userspace is shared

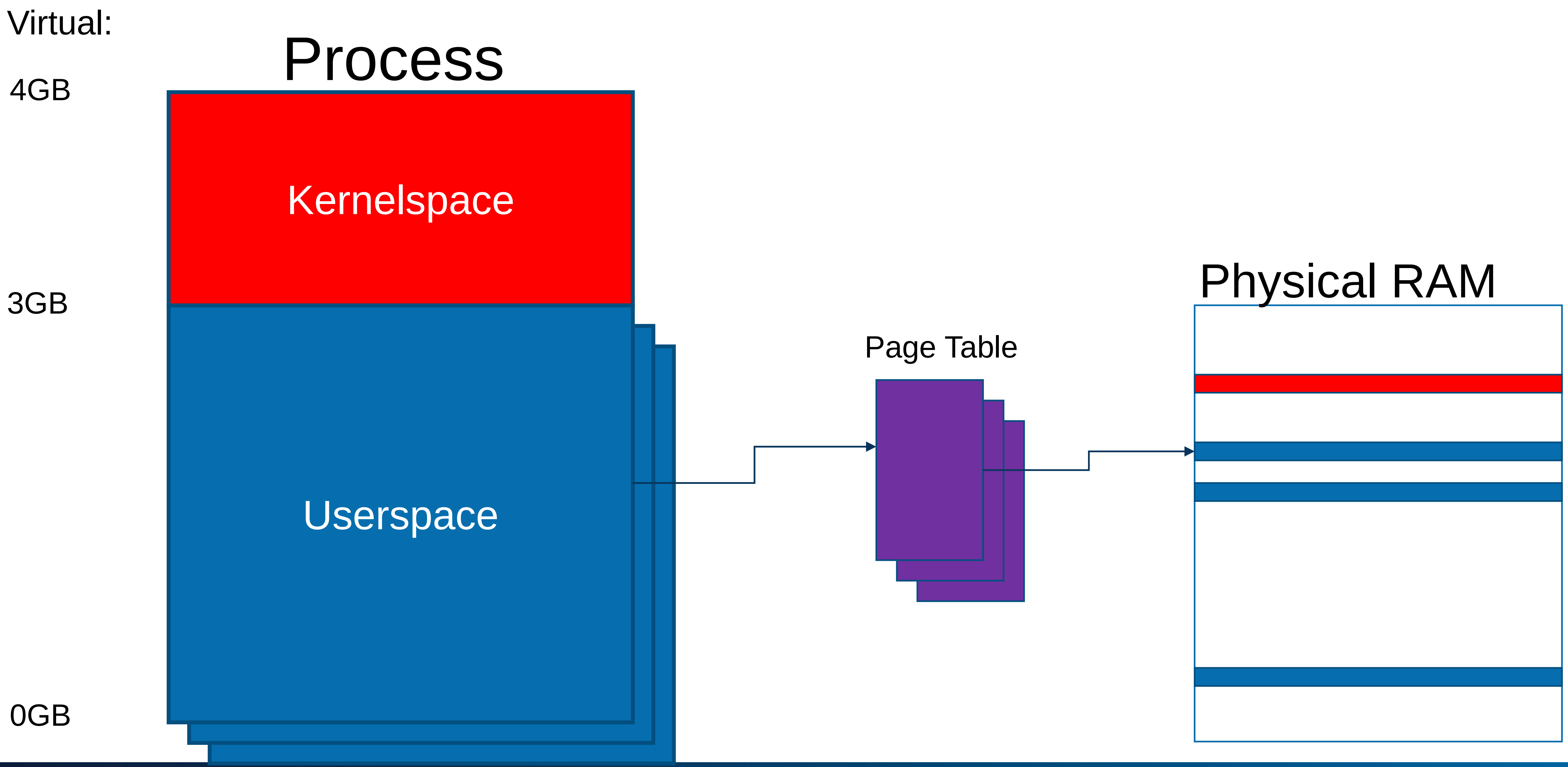


In-Userspace

Virtual:



In-Userspace



From Userspace to Device

From Userspace to Device

Send a network packet, slow:

- Userspace process create buffer with data (e.g. A HTTP request)
- Syscall to kernel with address of buffer
- Kernel **copies** userspace buffer to kernel space
- Kernel splits and manages buffer (split into MTU size, add TCP/IP/Ethernet header etc.) to create network packets
- Kernel **copies** network packets to special address mapped to NIC RAM
- Kernel «syscalls» NIC (network interface card) with buffer address (in their RAM)
- NIC sends it over the wire (via local RAM)

From Userspace to Device

Send a network packet, fast:

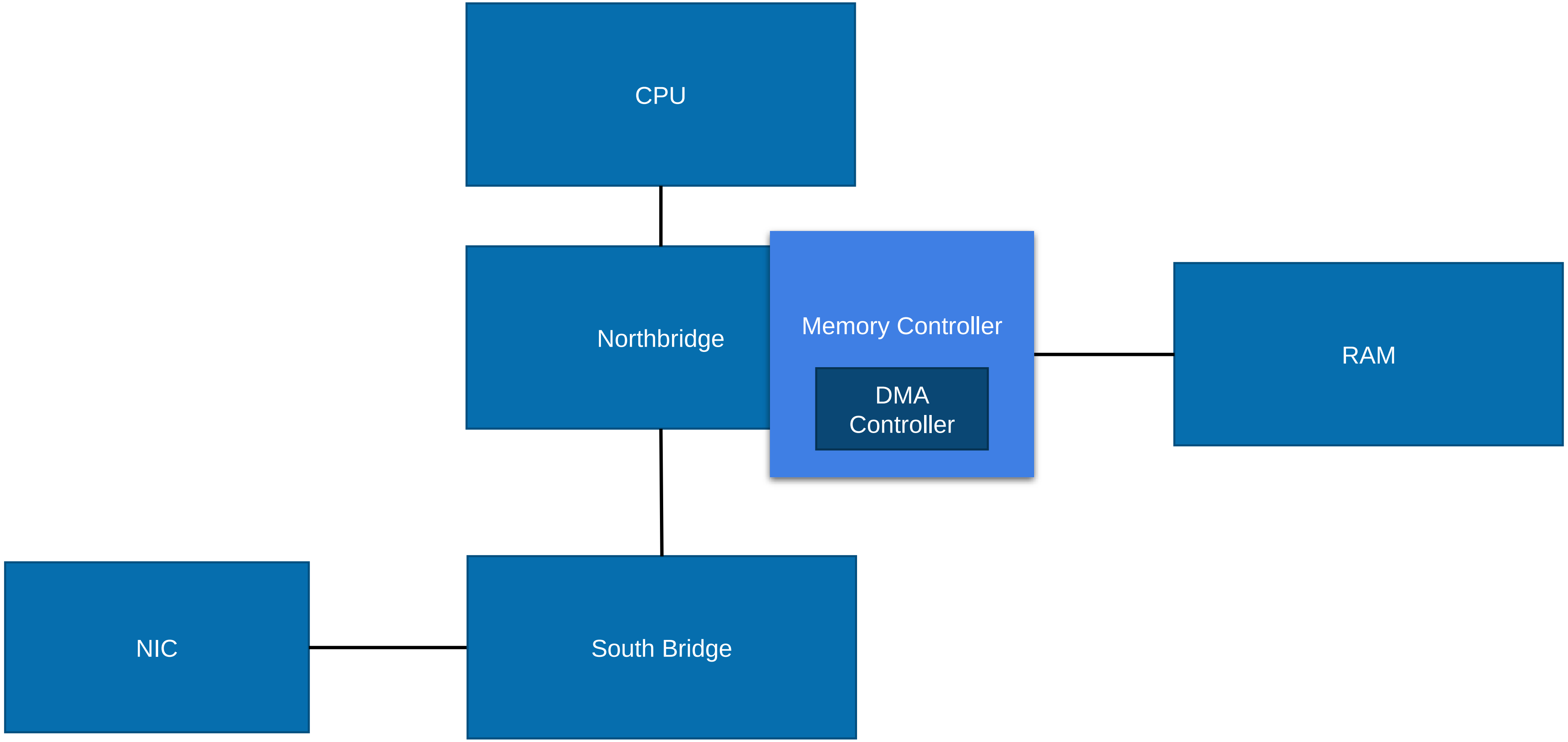
- Userspace process create buffer with data (e.g. A HTTP request)
- Syscall to kernel
- Kernel ~~copies~~ maps userspace buffer to kernel space
- Kernel splits and manages buffer (split into MTU, add TCP/IP/Ethernet header etc.) to create network packets
- Kernel «syscalls» NIC (network interface card) with network packet physical address
- ~~Buffer gets copied to NIC (NIC’s RAM)~~
- NIC sends packet at physical address it over the wire (via DMA)

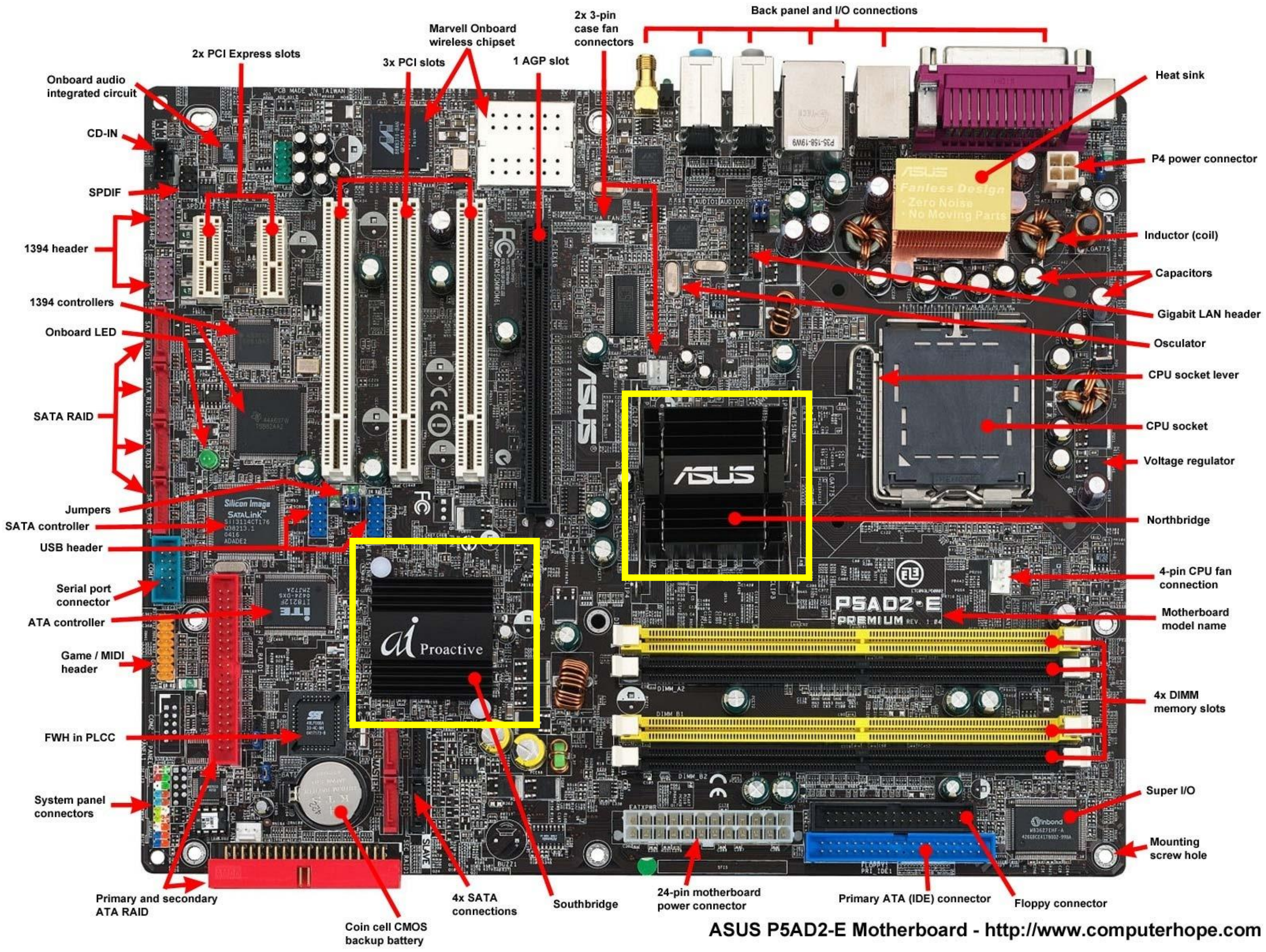
From Userspace to Device

Send a network packet, sendfile() / **zerocopy**:

- Userspace process has a file with data (e.g. FTP Server, JustinBieberGreatestHits.rar) mapped
- Syscall to kernel with file memory mapping address
- Kernel ~~copies~~ maps userspace buffer to kernel space
- ~~Kernel splits and manages buffer (split into MTU, add TCP/IP/Ethernet header etc.)~~
- Kernel «syscalls» NIC (network interface card) with file physical/virtual? address
- ~~Buffer gets copied to NIC (NIC's RAM)~~
- NIC sends packet at physical/virtual? address it over the wire (via DMA)

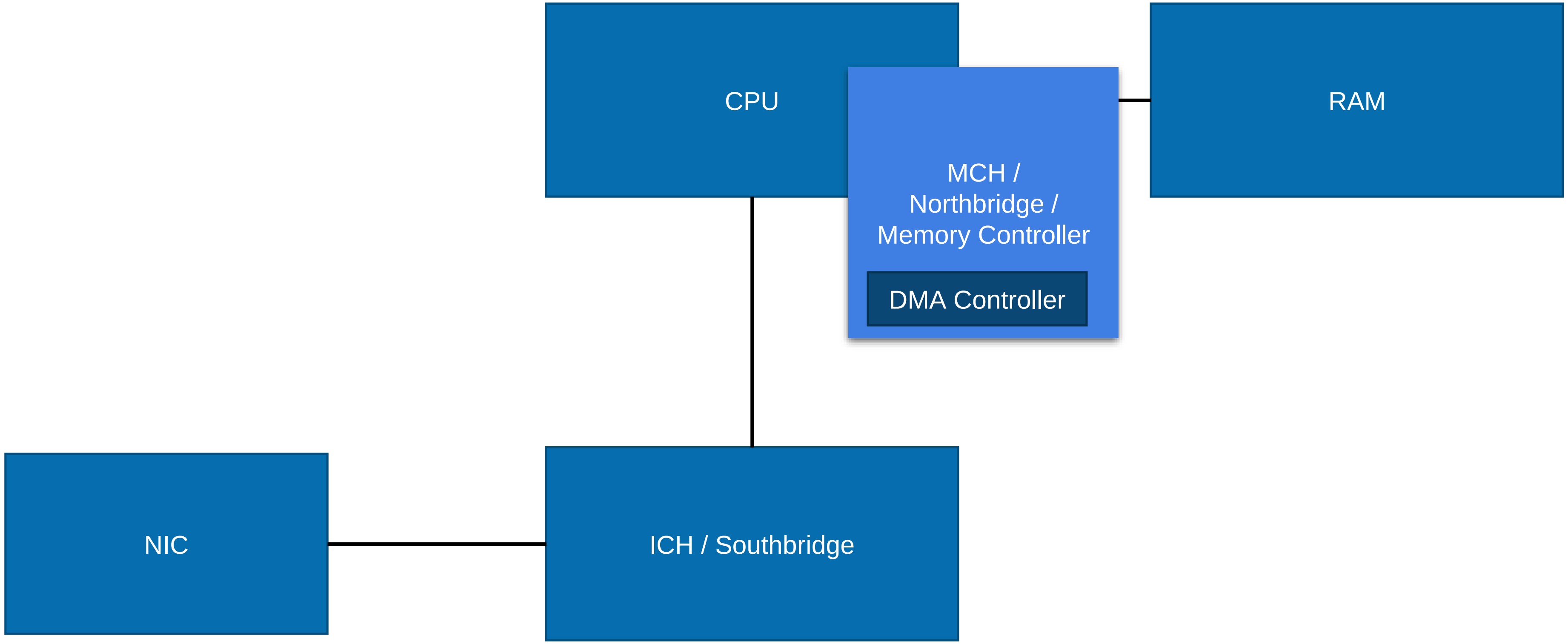
Hardware Layout - Old



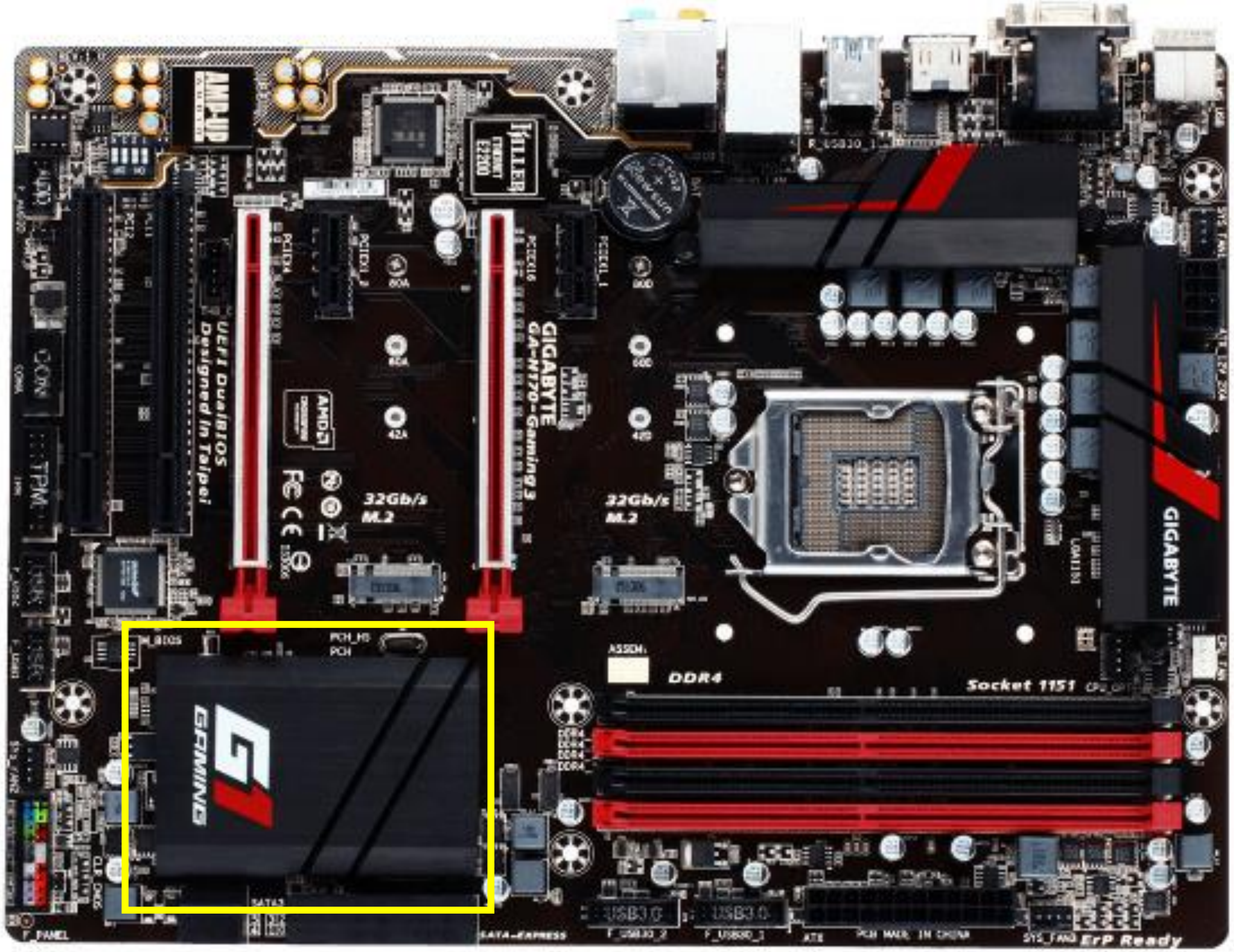


https://www.quora.com/Where-is-the-North-Bridge-in-Kaby-Lake-Intel-motherboard

Hardware Layout - New

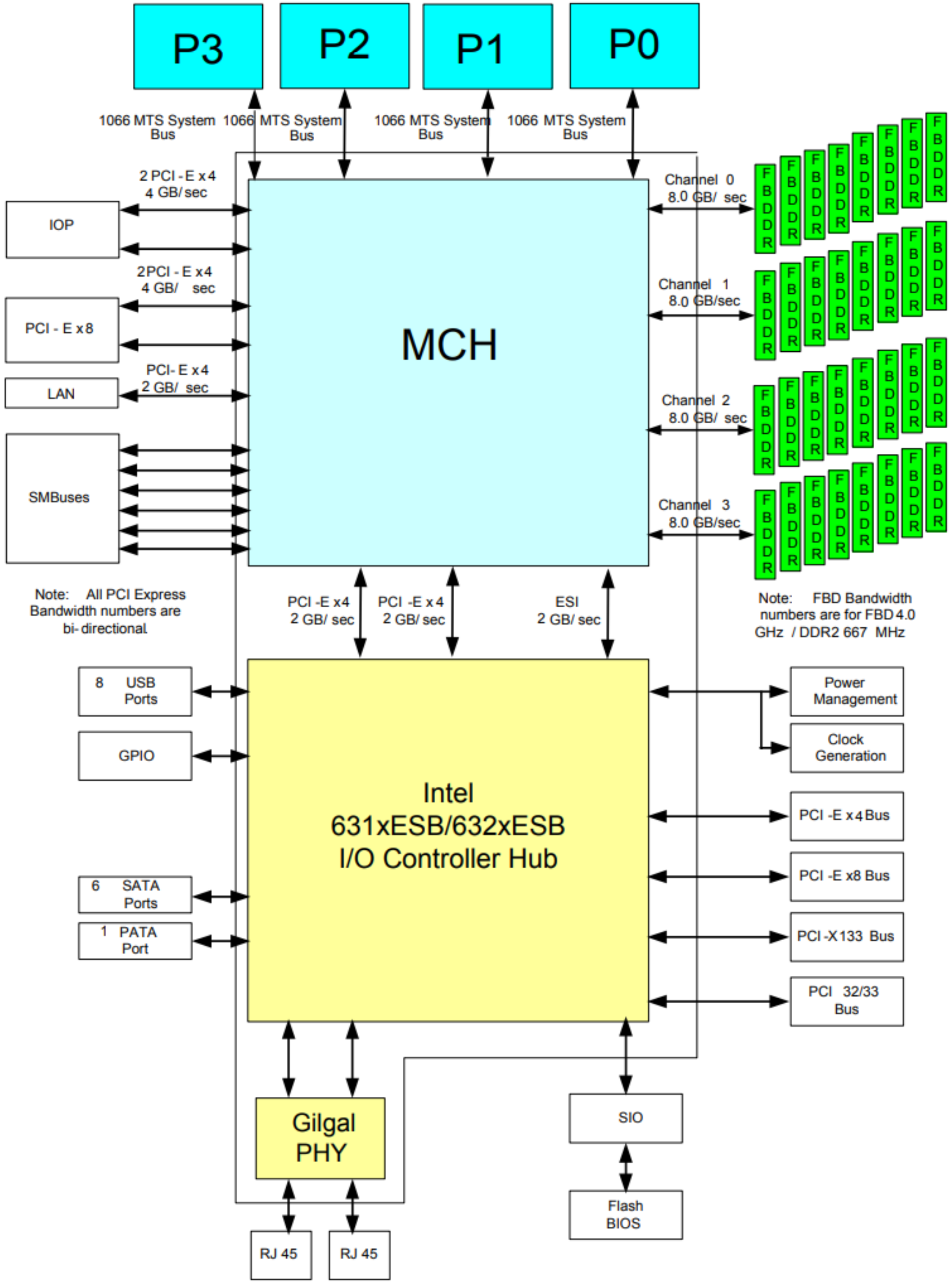


Hardware Layout - New

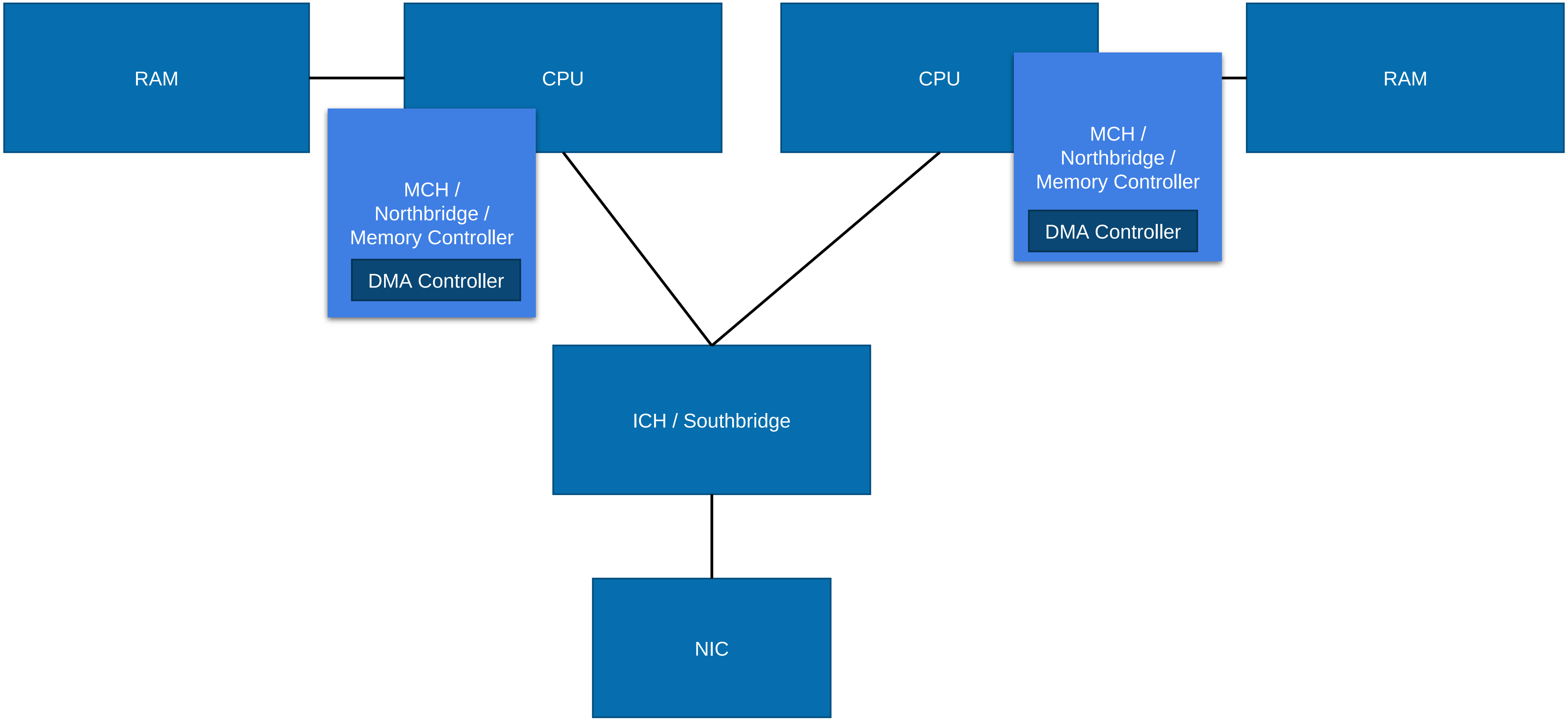


Hardware Layout- New

https://www.intel.com/content/dam/doc/datasheet/7300-chipset-memory-controller-hub-datasheet.pdf



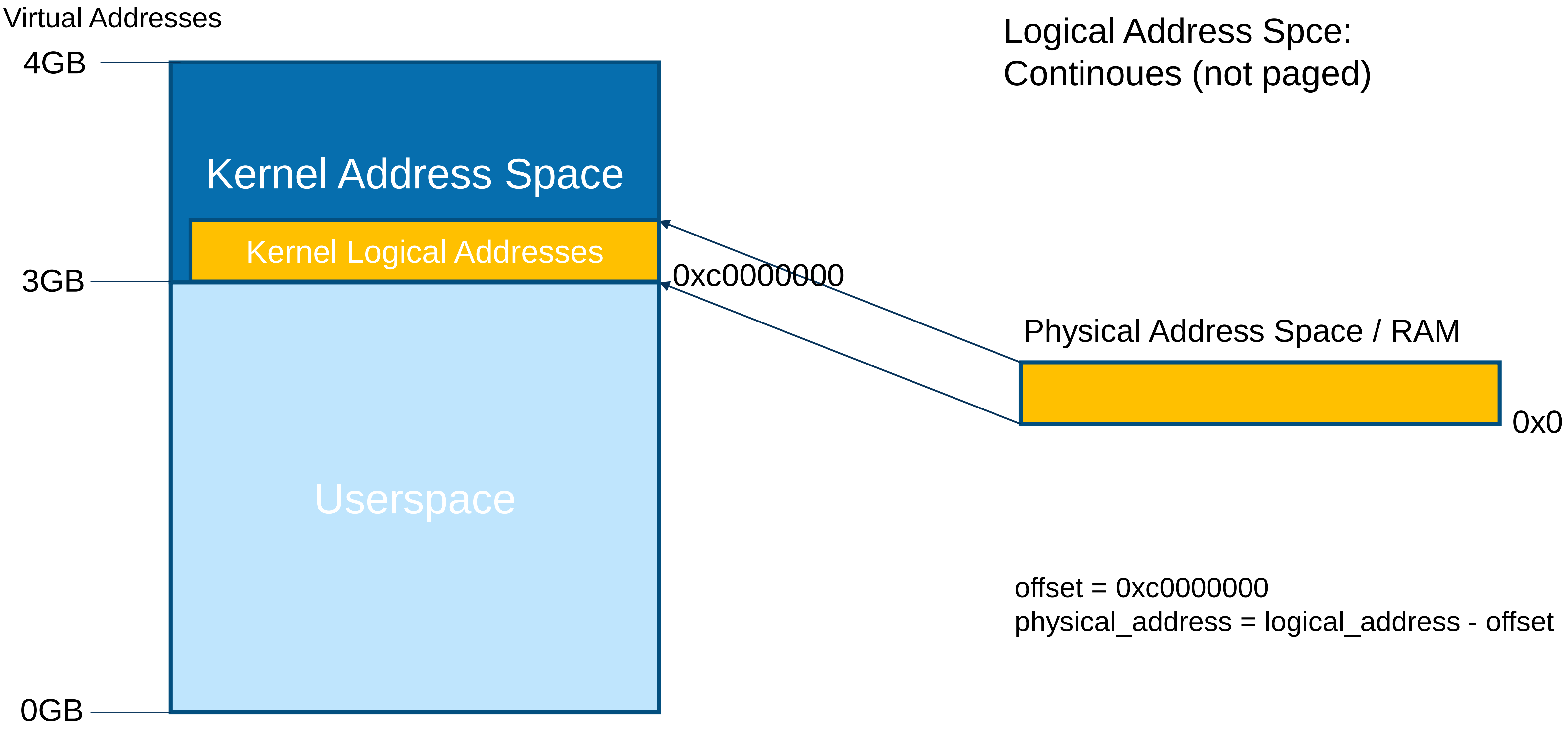
NUMA – Non Uniform Memory Architecture



NUMA – Non Uniform Memory Architecture

Virtual vs. Logical vs. Physical Addresses

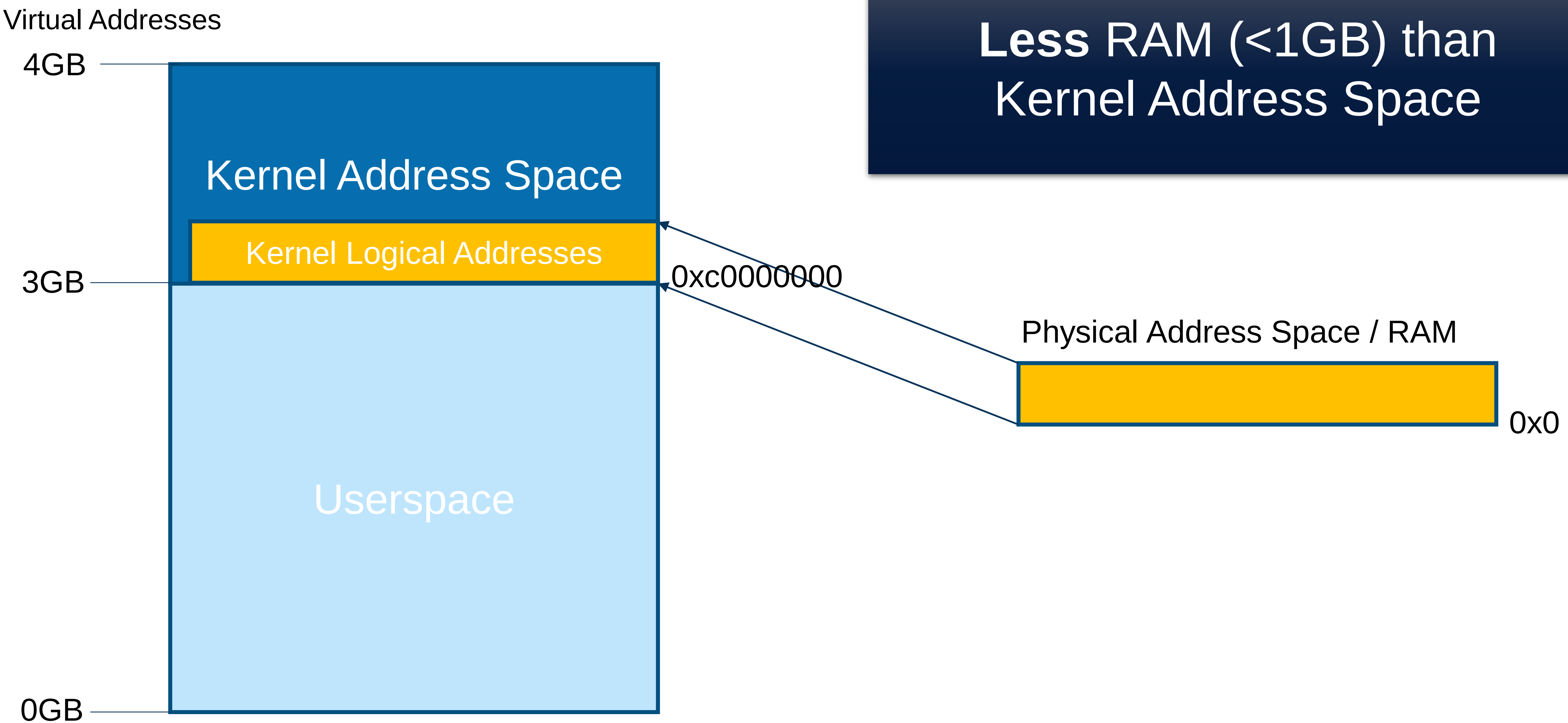
Kernel Memory Addresses



Logical Address Spce:
Continoues (not paged)

offset = 0xc0000000
physical_address = logical_address - offset

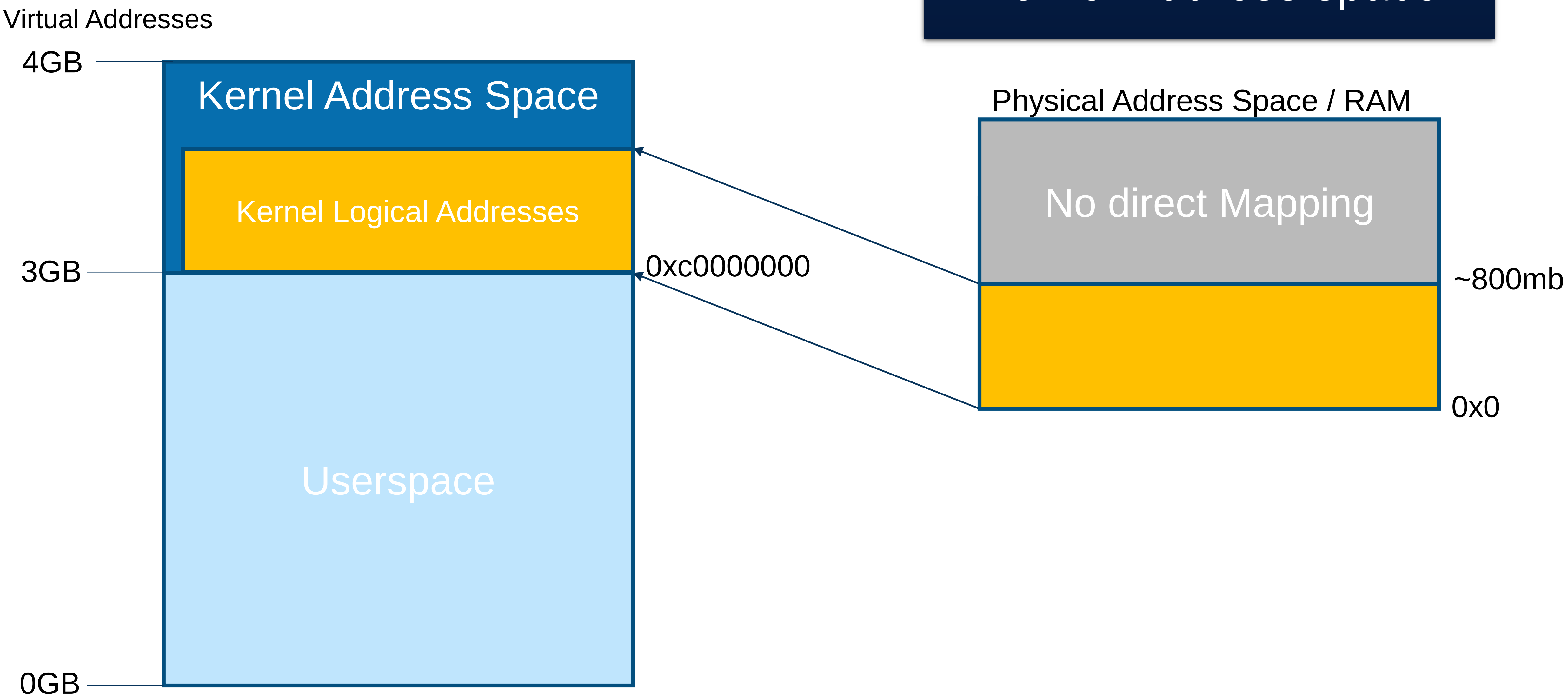
Kernel Memory Addresses



Less RAM (<1GB) than Kernel Address Space

Kernel Memory Addresses

More RAM (>1GB) than Kernel Address space



Kernel Memory Addresses

Logical Addressing:

- Base + Offset
- Linear Address Space
- Address mapps directly to some hardware address, via offset

Virtual / Linear Addressing:

- Addresses are contigous, but not linear (page table mapping)

Physical Addressing:

- The actual address of the main memory

Kernel Exploitation

Kernel Exploit Development

Kernel Exploit Development

- Techniques are same as in userspace
  - Stack based buffer overflow
  - Heap based buffer overflow (slab allocator)
  - Racing conditions
  - NULL/userspace dereference bugs
  - Logical bugs

- After getting RCE:
  - Patch syscall table
  - Or find our shell process and set uid to 0
  - Disable SELinux

Kernel Vulnerability Examples

Kernel Vulnerability Examples

All examples taken from «A Guide to Kernel Exploitation», Chapter 2, «A Taxonomy of Kernel Vulnerabilities»

Kernel Vulnerability Examples

Integer issue:

```
static int bluez_sock_create(struct socket *sock, int proto) {
        if (proto >= BLUEZ_MAX_PROTO)
                return -EINVAL;
        […]

        return bluez_proto[proto]->create(sock,proto);
}
```

Kernel Vulnerability Examples

UNINITIALIZED/NONVALIDATED/CORRUPTED POINTER DEREFERENCE

```
struct ucred ucred, *ucp; [1]
[…]

refcount_init(&ucred.cr_ref, 1);
ucred.cr_uid = ip->i_uid;
ucred.cr_ngroups = 1;
ucred.cr_groups[0] = dp->i_gid; [2]
ucp = &ucred;

struct ucred {
        u_int cr_ref;
        […]
        gid_t *cr_groups;
        int cr_agroups;
};
```

Kernel Vulnerability Examples

Unchecked pointer:

```
static long vmsplice_to_user(struct file *file, const struct iovec __user *uiov,
                             unsigned long nr_segs, unsigned int flags) {
        error = get_user(base, &iov->iov_base); [1]
        […]
        if (unlikely(!base)) {
                error = -EFAULT;
                break;
        }
        sd.u.userptr = base; [2]
        size = pipe_to_user(pipe, &sd, pipe_to_user);

static int pipe_to_user(struct pipe_inode_info *pipe, struct pipe_buffer *buf, struct splice_desc *sd) {
        if (!fault_in_pages_writeable(sd->u.userptr, sd->len)) {
                src = buf->ops->map(pipe, buf, 1);
                ret = __copy_to_user_inatomic(sd->u.userptr, src + buf->offset, sd->len); [3]
                buf->ops->unmap(pipe, buf, src); […] }
```

Kernel Exploitation

Remote Kernel Exploit

Kernel Exploiting: Remote Kernel Exploit

Attack Surface

- TCP/IP Stack
- IPSec
- Drivers: Bluetooth / Wifi or similar
- Whatever else is in Kernelspace, and reachable via Network / Signals

Example Windows Kernel Exploit: Winnuke

The term **WinNuke** refers to a remote denial-of-service attack (DoS) that affected the Microsoft Windows 95, Microsoft Windows NT and Microsoft Windows 3.1x computer operating systems. The exploit sent a string of OOB (out of band) data to the target computer on port 139 (NetBIOS), causing it to lock up and display a Blue Screen of Death

Example Windows Kernel Exploit: Eternal Blue

Eternal Blue

“This module is a port of **the Equation Group ETERNALBLUE** exploit, part of the FuzzBunch toolkit released by **Shadow Brokers**. There is a **buffer overflow** memmove operation in Srv!SrvOs2FeaToNt. The size is calculated in Srv!SrvOs2FeaListSizeToNt, with mathematical error where a DWORD is subtracted into a WORD. The kernel pool is groomed so that overflow is well laid-out to overwrite an SMBv1 buffer. Actual RIP hijack is later completed in srvnet!SrvNetWskReceiveComplete.

This exploit, like the original may not trigger 100% of the time, and should be run continuously until triggered. It seems like the pool will get hot streaks and need a cool down period before the shells rain in again. The module will attempt to use Anonymous login, by default, to authenticate to perform the exploit. If the user supplies credentials in the SMBUser, SMBPass, and SMBDomain options it will use those instead. **On some systems, this module may cause system instability and crashes, such as a BSOD or a reboot.** This may be more likely with some payloads.”

https://www.rapid7.com/db/modules/exploit/windows/smb/ms17_010_eternalblue
https://github.com/rapid7/metasploit-framework/pull/9473

Kernel Exploitation

Linux Kernel Exploit Mitigations

Kernel Exploitation – Mitigation Features

- kASLR
  - Available, but disabled by default (hibernation problems)
- DEP
  - Default
  - CONFIG_DEBUG_RODATA
  - But some pages are W & X…
    - Because of X86 (BIOS etc.)
    - Therefore, not so useful
- The usual compile time stuff (but in hard)
  - Stack canaries (default)
  - Fortify source (default)
  - Randstruct
    - Randmizes order of struct entries (per build)

Linux Kernel Exploit Mitigations

2012: Ivy Bridge	(e.g. i7 48xx, 49xx)
2013: Haswell	(e.g. i7 47xx)
2014: Broadwell	(e.g. i7 56xx, 55xx, 58xx, 59xx)
2015: Skylake	(e.g. i7 65xx, 66xx, 67xx, 68xx, 69xx)

- SM**E**P
  - Prevents **E**xecution of userspace code in kernelspace (ret2usr)
  - Since Kernel 3.0
  - Needs CPU support: Ivy Bridge ++
  - **Enabled by default** in modern distributions
  - Workaround: In-kernel ROP
  - cat /proc/cpuinfo | grep smep

- SM**A**P
  - Deny Kernel direct **A**ccess to userspace memory
  - Since Kernel 3.7
  - Needs CPU support: Broadwell ++
  - **Enabled by default** in modern distributions

Linux Kernel Exploitation

Linus

Linus about bugs

```
From: Linus Torvalds <torvalds@linux-foundation.org>
Newsgroups: fa.linux.kernel
Subject: Re: [stable] Linux 2.6.25.10
Date: Tue, 15 Jul 2008 02:28:23 UTC
Message-ID: <fa.FocnvcnLqG7kPaYdjYdPJfSdhjc@ifi.uio.no>

On Tue, 15 Jul 2008, pageexec@freemail.hu wrote:
>
> so guys (meaning not only Greg but Andrew, Linus, et al.), when will you
> publicly explain why you're covering up security impact of bugs? and even
> more importantly, when will you change your policy or bring your process
> in line with what you declared?

We went through this discussion a couple of weeks ago, and I had
absolutely zero interest in explaining it again.

I personally don't like embargoes. I don't think they work. That means
that I want to fix things asap. But that also means that there is never a
time when you can "let people know", except when it's not an issue any
more, at which point there is no _point_ in letting people know any more.

So I personally consider security bugs to be just "normal bugs". I don't
cover them up, but I also don't have any reason what-so-ever to think it's
a good idea to track them and announce them as something special.

So there is no "policy". Nor is it likely to change.

                        Linus
```

Linus about bugs

From: Linus Torvalds <torvalds@linux-foundation.org>
Newsgroups: fa.linux.kernel
Subject: Re: [stable] Linux 2.6.25.10
Date: Tue, 15 Jul 2008 16:14:00 UTC
Message-ID: <fa.a8PYBfKwFq0Fl6ls/kFjBfKbV44@ifi.uio.no>

On Tue, 15 Jul 2008, Linus Torvalds wrote:
>
> So as far as I'm concerned, "disclosing" is the fixing of the bug. It's
> the "look at the source" approach.

Btw, and you may not like this, since you are so focused on security, one reason I refuse to bother with the whole security circus is that I think it glorifies - and thus encourages - the wrong behavior.

It makes "heroes" out of security people, as if the people who don't just fix normal bugs aren't as important.

In fact, all the boring normal bugs are _way_ more important, just because there's a lot more of them. I don't think some spectacular security hole should be glorified or cared about as being any more "special" than a random spectacular crash due to bad locking.

Security people are often the black-and-white kind of people that I can't stand. I think the OpenBSD crowd is a bunch of masturbating monkeys, in that they make such a big deal about concentrating on security to the point where they pretty much admit that nothing else matters to them.

To me, security is important. But it's no less important than everything *else* that is also important!

Linus

Linus about bugs

We had a ton of issues with the original hardened usercopy just doing bad things.

We _still_ have outstanding issues with the structure randomization corrupting the kernel.

These "hardening" things really seem to be a source of random bugs, and they haven't been extensively tested, and the people involved quite often don't seem to care about basic cleanliness (because "security is so important that nothing else matters").

http://lkml.iu.edu/hypermail/linux/kernel/1711.2/01701.html

So honestly, this is the kind of completely unacceptable "security person" behavior that we had with the original user access hardening too, and made that much more painful than it ever should have been.

IT IS NOT ACCEPTABLE when security people set magical new rules, and then make the kernel panic when those new rules are violated.

That is pure and utter bullshit. We've had more than a quarter century _without_ those rules, you don't then suddenly walz in and say "oh, everbody must do this, and if you haven't, we will kill the kernel".

The fact that you "introduced the fallback mode" late in that series just shows HOW INCREDIBLY BROKEN the series started out.

Seriously.

As a security person, you need to repeat this mantra:

"security problems are just bugs"

and you need to _internalize_ it, instead of scoff at it.

The important part about "just bugs" is that you need to understand that the patches you then introduce for things like hardening are primarly for DEBUGGING.

I'm not at all interested in killing processes. The only process I'm interested in is the _development_ process, where we find bugs and fix them.

Linux Kernel Politics

- Infrastructure people
  - Don’t know they are there, except when something breaks
  - Availability #1 prio
- 8 stable kernel trees!
  - And Distros have their own stable kernels…
- Actively hide security fixes in commit messages
  - And they are honest about this
- Distros in charge of security
  - Is this good or not?
- Update: Kernel Self Protection Project (catch up to grsec)
- Conclusion:
  - Important security fixes are maybe not backported to stable kernels

Kernel Exploitation

Recap

Kernel Exploitation

Recap

- Kernel Exploitation is similar to Userspace
- Kernel Exploit Mitigations similar to Userspace
- Kernel has big attack surface (locally)

References

- List:
  - https://github.com/xairy/linux-kernel-exploitation
- Attacking the Core : Kernel Exploiting Notes
  - http://phrack.org/archives/issues/64/6.txt