Authorization, Confinement, and Virtualization

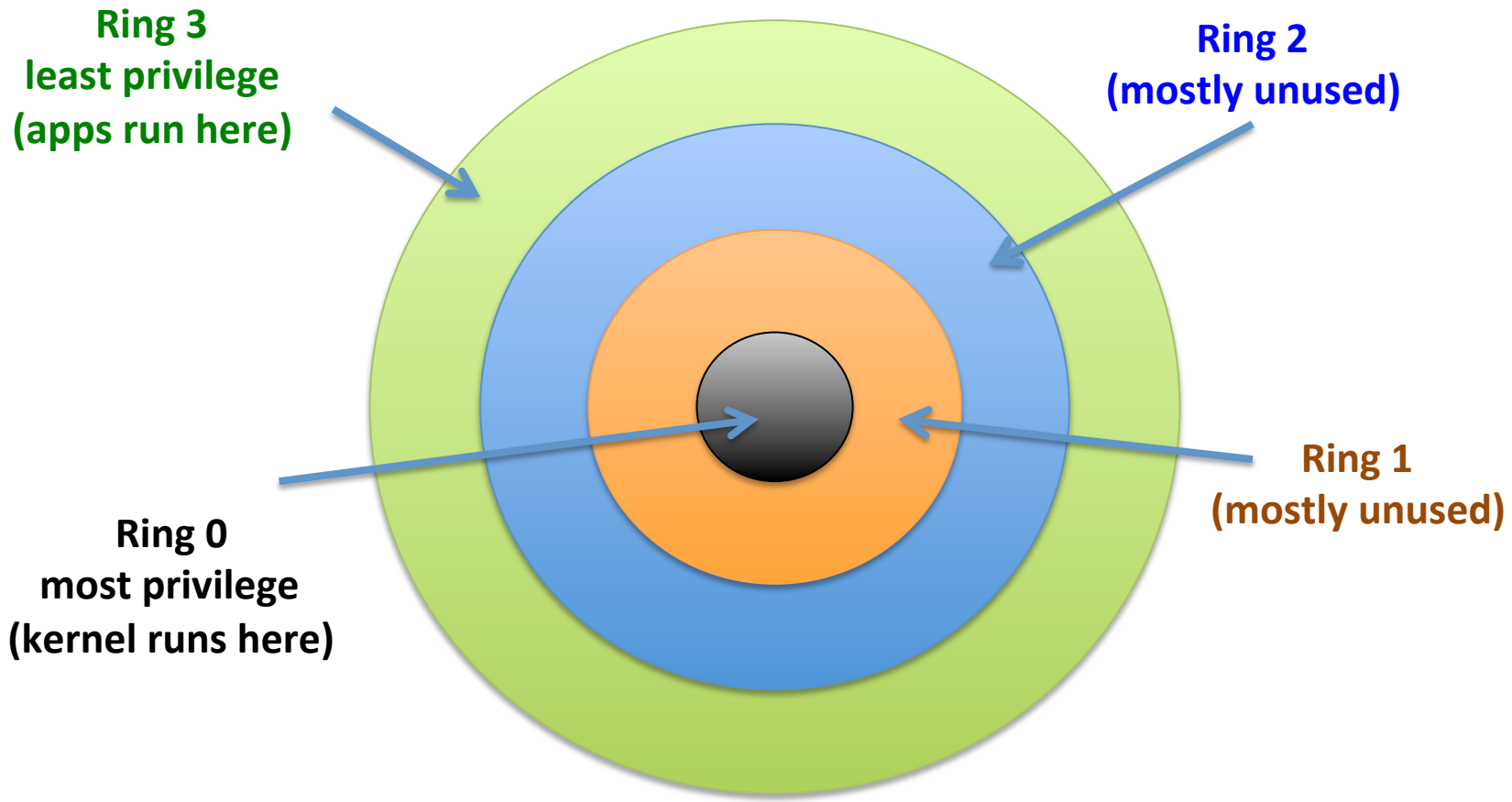
CS 491/591

Fall 2015

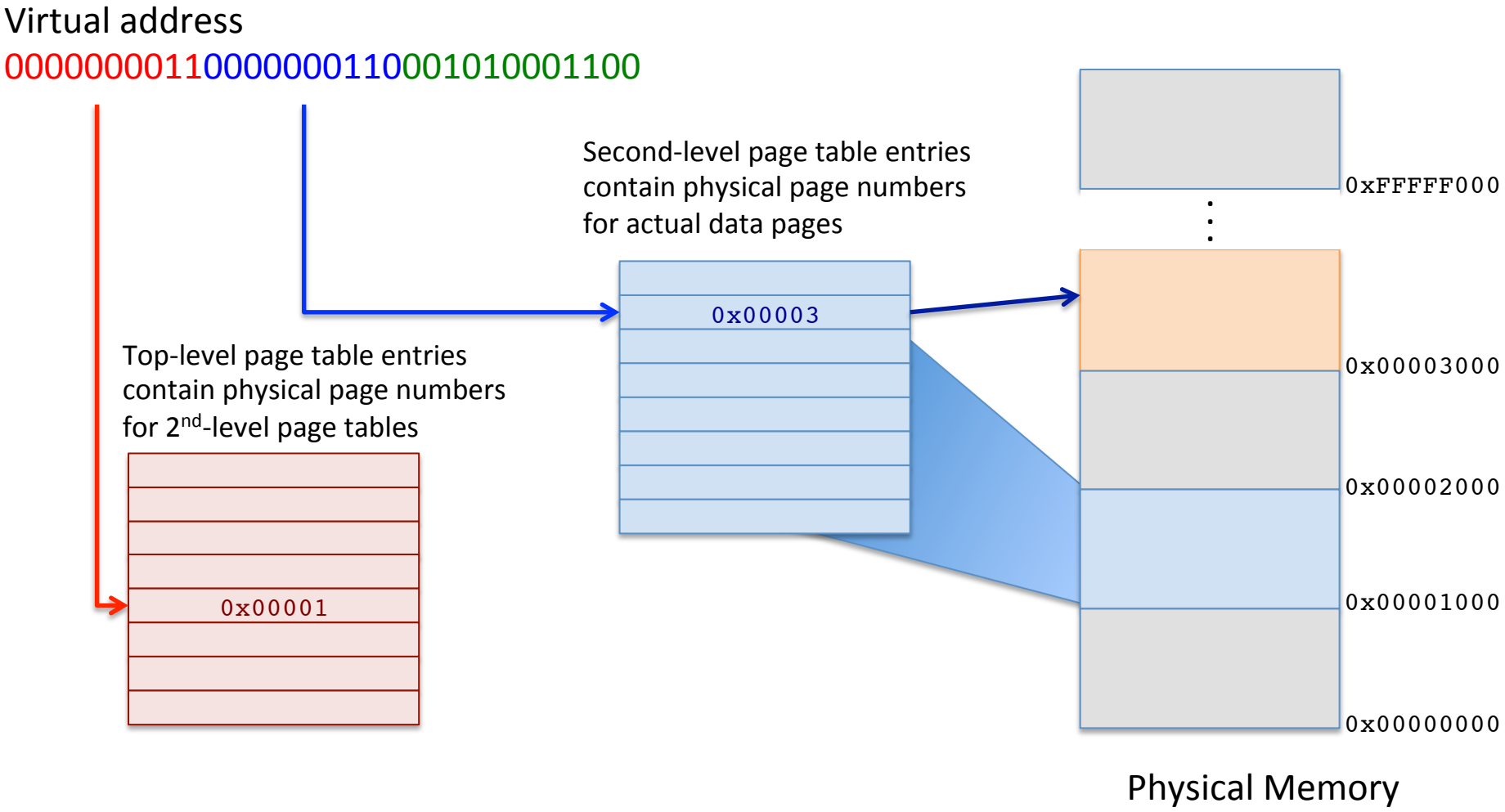
Outline for Today

- Review: Hardware support for security
  - Protecting access to code
  - Protecting access to memory
- Authorization
  - Theory
  - Access control lists (ACL's)
  - Capability systems
- Virtualization as access control
- Access Control Policies

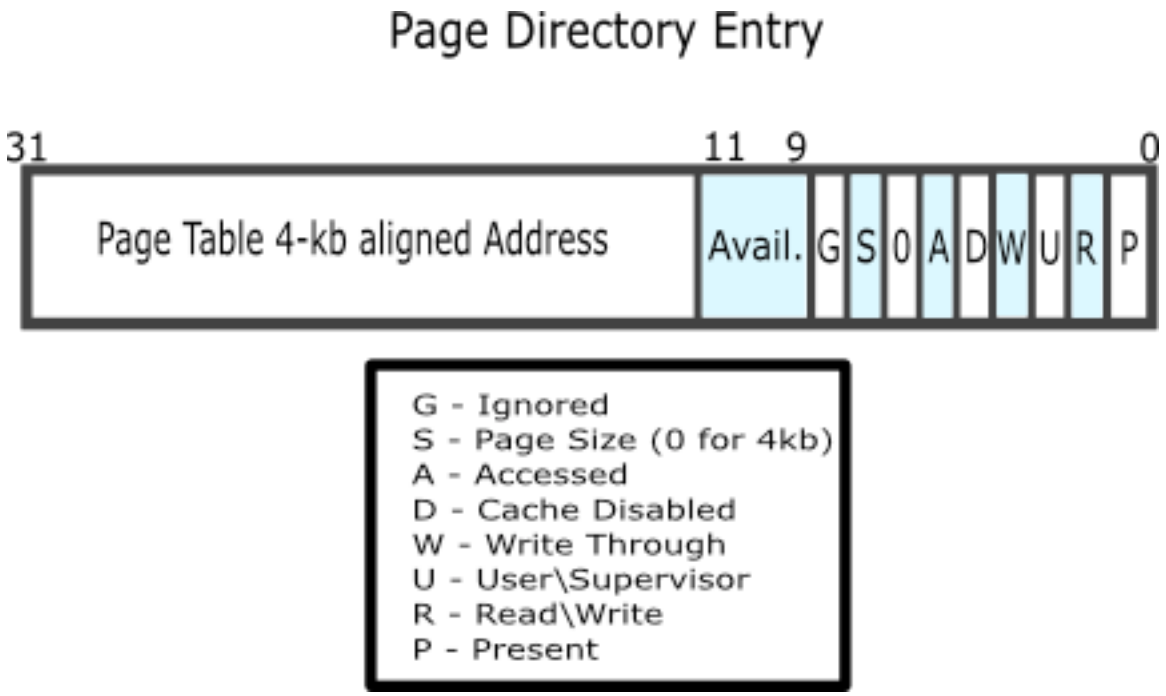
Hardware Privilege Levels



Multi-level Page Tables



Page Table Entries on x86



Credit: http://wiki.osdev.org/Paging

Outline for Today

- Review: Hardware support for security
  - Protecting access to code
  - Protecting access to memory
- Authorization
  - Theory
  - Access control lists (ACL's)
  - Capability systems
- Virtualization as access control

Pithy Quotes from Anderson

- Going all the way back to early time-sharing systems we systems people regarded the users, and any code they wrote, as the mortal enemies of us and each other. We were like the police force in a violent slum.
  – *Roger Needham*

- Microsoft could have incorporated effective security measures as standard, but good sense prevailed. Security systems have a nasty habit of backfiring and there is no doubt they would cause enormous problems.
  – *Rick Maybury*

Trustworthy Computing Memo

- Six months ago, I sent a call-to-action to Microsoft's 50,000 employees, outlining what I believe is the highest priority for the company and for our industry over the next decade: building a Trustworthy Computing environment for customers that is as reliable as the electricity that powers our homes and businesses today.
  – Bill Gates (July 18, 2002)

AAA

- Authentication
  - How do we know users are who they claim to be?
- Authorization – (today)
  - How do we decide what resources users and programs may access?
- Audit
  - How do we keep track of what users and programs are doing?

Authorization

```
Login: bob
Password: hunter2
```

How does the system decide whether to allow or deny access to resources?

```
Last Login 1/12/13 3:05pm from console
[bob@desktop ~]$ ls /home/bob
Desktop  Documents  Downloads  Music  Pictures
[bob@desktop ~]$ ls /home/joe
ls: cannot open directory /home/joe: Permission denied
```

Authorization: Some theory

- Access Control Matrix
- Access Control Lists
- Capabilities

Lampson's Access Control Matrix

Resources are columns

Principals are rows

	/home/bob	/home/bob/ Documents	/usr/share/ stuff	/home/joe	…
Bob	Read, write	Read, write	Read, write		
Joe			Read		
Sarah			Read		
Fred			Read		
Eliza			Read,write		
Jorge			Read		
Admin	Read	Read	Read, write	Read	
…					

Cells in the matrix says who's allowed to access which resources, and in which ways (read, write, etc.)

Lampson's Access Control Matrix

Resources are columns

Principals are rows

	/home/bob	/home/bob/...		/home/joe	...
Bob	Read				
Joe					
S					
Fr					
Eliz					
Jorg					
Admi			Read, write	Read	
...					

Problem: For real systems, this access control matrix would be HUGE!

How can we make this practical?

Cells in the matrix says who's allowed to access which resources, and in which ways (read, write, etc.)

Access Control Lists

- Store each column of the access control matrix along with the resource it describes

	/home/bob	/home/bob/ Documents	/usr/share/ stuff	/home/joe	…
Bob	Read, write	Read, write	Read, write		
Joe			Read		
Sarah			Read		
Fred			Read		
Eliza			Read,write		
Jorge			Read		
Admin	Read	Read	Read, write	Read	
…					

Unix file permissions

- Each file is owned by 1 user and 1 group
- File permissions stored as an access control list
  - R Read
  - W Write
  - X Execute (meaning traverse, for directories)
- Permissions are listed for
  - U The user who owns the file
  - G The group who owns the file
  - O Other users

Bit-vector representation

- Can represent each list of permissions as a vector of 3 bits (R,W,X)

Text	Binary	Octal
rwx	111	7
rw-	110	6
r-x	101	5
r--	100	4
-wx	011	3
-w-	010	2
--x	001	1
---	000	0

Unix processes

- Each running process is owned by a user (uid) and group (gid)
- Child process inherits ownership from parent
- Processes running as root (uid 0) have special privileges
- User and group id can be changed via system call
  - setuid(), setgid()
  - Only available to privileged (root) processes

Unix "Discretionary" Access Control

- Process A is allowed access to resource B if:
  - A is running as the user who owns B, AND the "user" permission bits allow it
  - OR process A is running as a user in the group that owns B, AND the "group" permission bits allow it
  - OR the "other" permission bits allow it

Unix file permissions: Examples

```
[bob@host ~]$ ls –l /home/
drwxrwxrwx   4 root   admin  4096 Jan 01 10:14 .
drwxrwxrwx  16 root   admin  4096 Jan 01 10:14 ..
drwxr-x--x  39 bob    users  4096 Nov 03 12:34 bob
drwxr-x--x  32 joe    users  4096 Nov 03 12:35 joe
[bob@host ~]$ chmod 0755 /home/bob
[bob@host ~]$ ls –l /home/
drwxrwxrwx   4 root   admin  4096 Jan 01 10:14 .
drwxrwxrwx  16 root   admin  4096 Jan 01 10:14 ..
drwxr-xr-x  39 bob    users  4096 Nov 03 12:34 bob
drwxr-x--x  32 joe    users  4096 Nov 03 12:35 joe
```

Problems with Unix-style permissions

- Root is all-powerful
- Very coarse-grained settings
  - Sometimes too broad
    - Solitaire can read Firefox's saved password file
  - Sometimes too confining (need root for network servers)
- Solutions: Various ways of making finer-grained protections

Outline for Today

- Review: Hardware support for security
  - Protecting access to code
  - Protecting access to memory
- Authorization
  - Theory
  - Access control lists (ACL's)
  - Capability systems
- Virtualization as access control

Lampson's Access Control Matrix

Resources are columns

Principals are rows

	/home/bob	/home/bob/ Documents	/usr/share/ stuff	/home/joe	…
Bob	Read, write	Read, write	Read, write		
Joe			Read		
Sarah			Read		
Fred			Read		
Eliza			Read,write		
Jorge			Read		
Admin	Read	Read	Read, write	Read	
…					

Cells in the matrix says who's allowed to access which resources, and in which ways (read, write, etc.)

Lampson's Access Control Matrix

Resources are columns

Principals are rows

	/home/bob	/home/bob/...		/home/joe	...
Bob	Read				
Joe					
S					
Fr					
Eliz					
Jorg					
Admi			Read, write	Read	
...					

Problem: For real systems, this access control matrix would be HUGE!

How can we make this practical?

Cells in the matrix says who's allowed to access which resources, and in which ways (read, write, etc.)

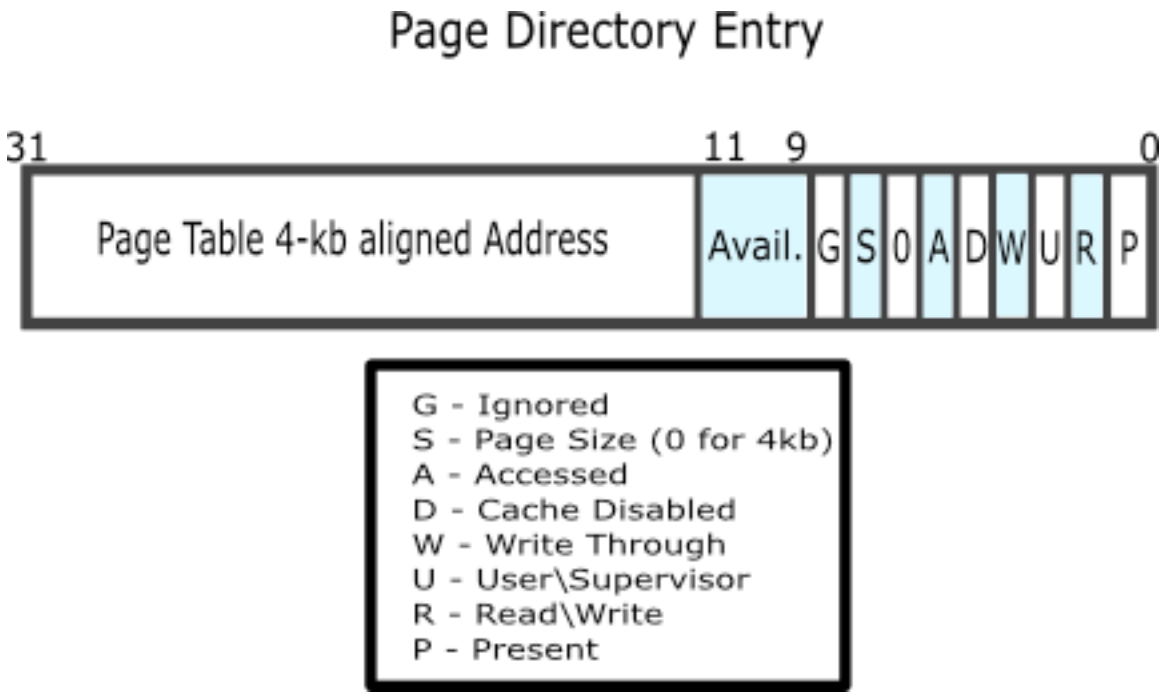
Capabilities

Store the access rights of each principal along with that principal

	/home/bob	/home/bob/ Documents	/usr/share/ stuff	/home/joe	…
Bob	Read, write	Read, write	Read, write		
Joe			Read		
Sarah			Read		
Fred			Read		
Eliza			Read,write		
Jorge			Read		
Admin	Read	Read	Read, write	Read	
…					

Fortunately, the “principals” in computer systems are really programs, not people

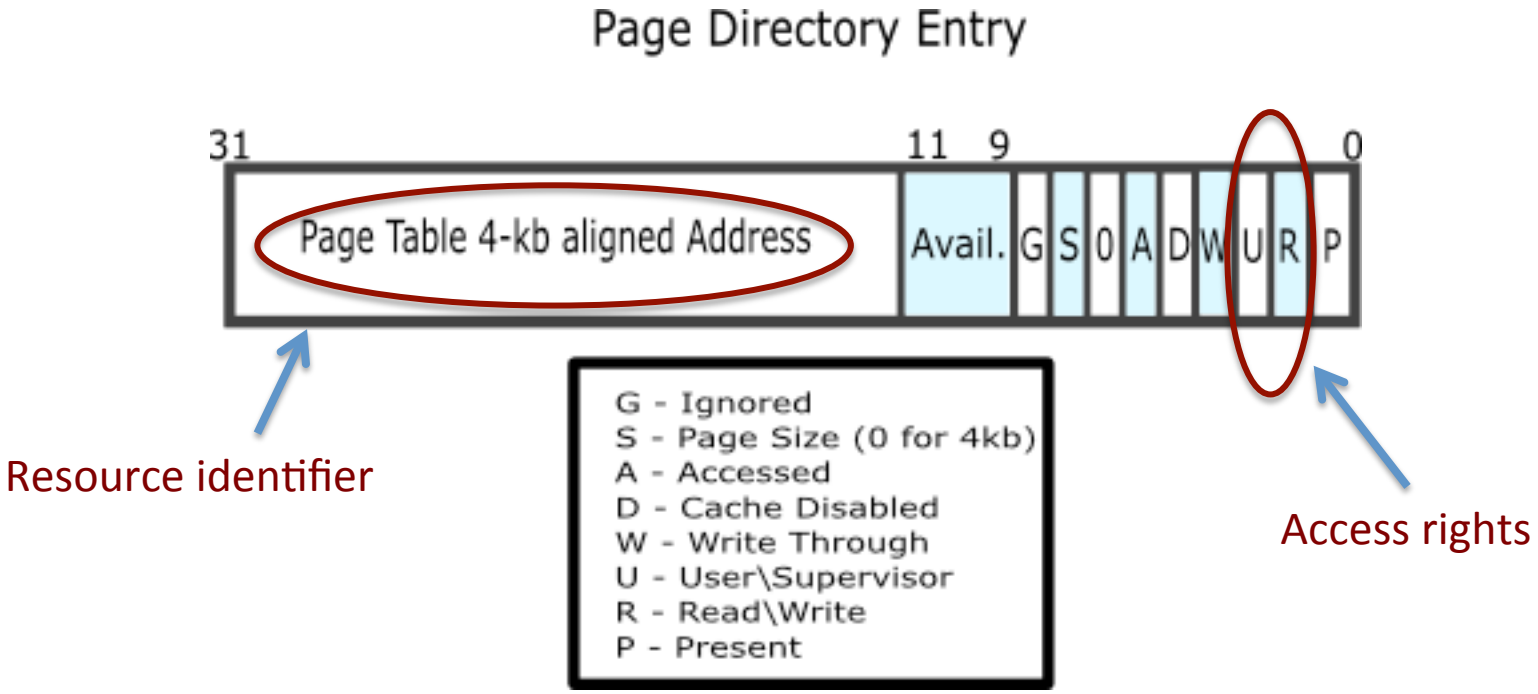
Page Table Entries on x86



Credit: http://wiki.osdev.org/Paging

Does this structure look familiar?

Hey! That looks like a capability!



Credit: http://wiki.osdev.org/Paging

File Descriptors as Capabilities

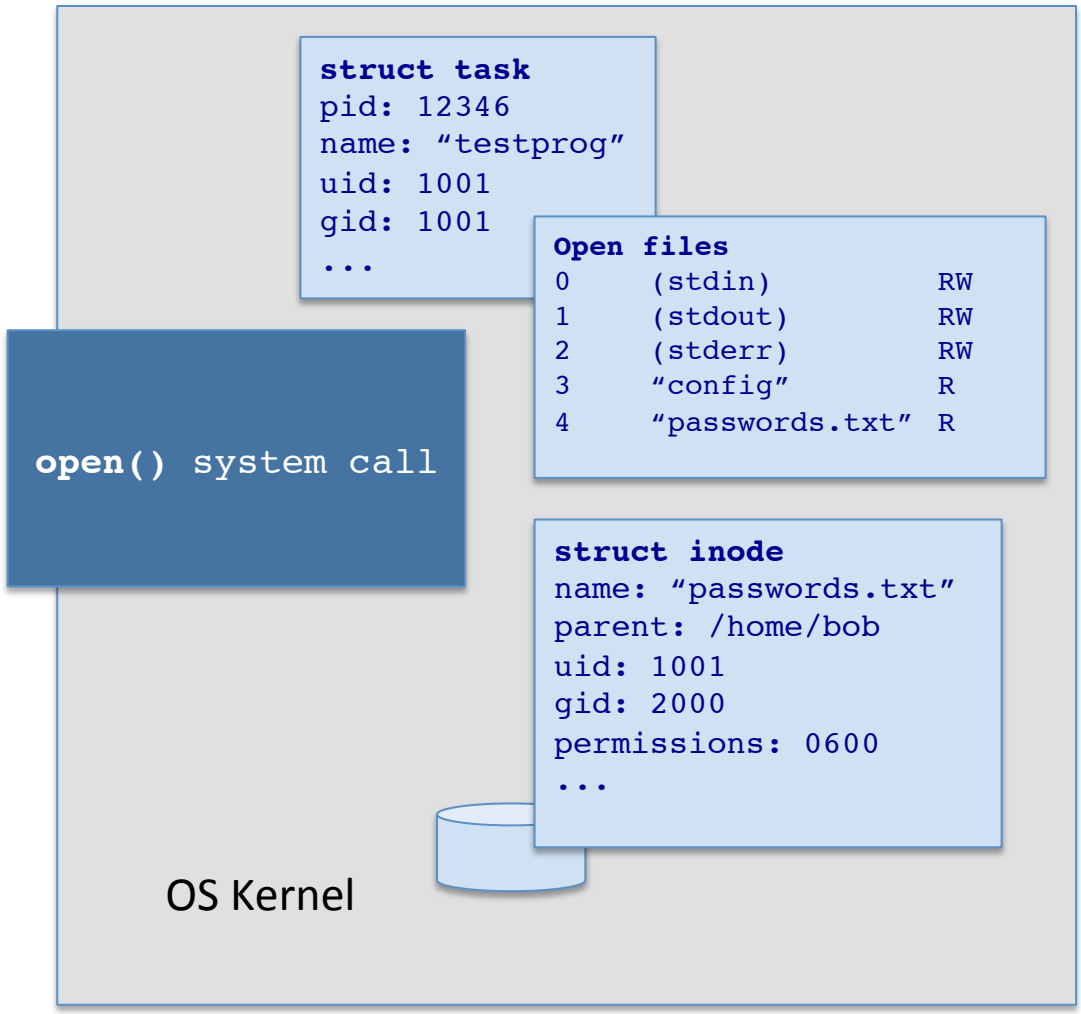
What happens when we run this code?

```
...
int fd = open("/home/bob/passwords.txt", O_RDONLY);
read(fd, buf, 8);
...
```

Acquiring a file descriptor

```
testprog
...
int fd = open("/home/bob/
  passwords.txt", O_RDONLY);
read(fd, buf, 8);
...
```

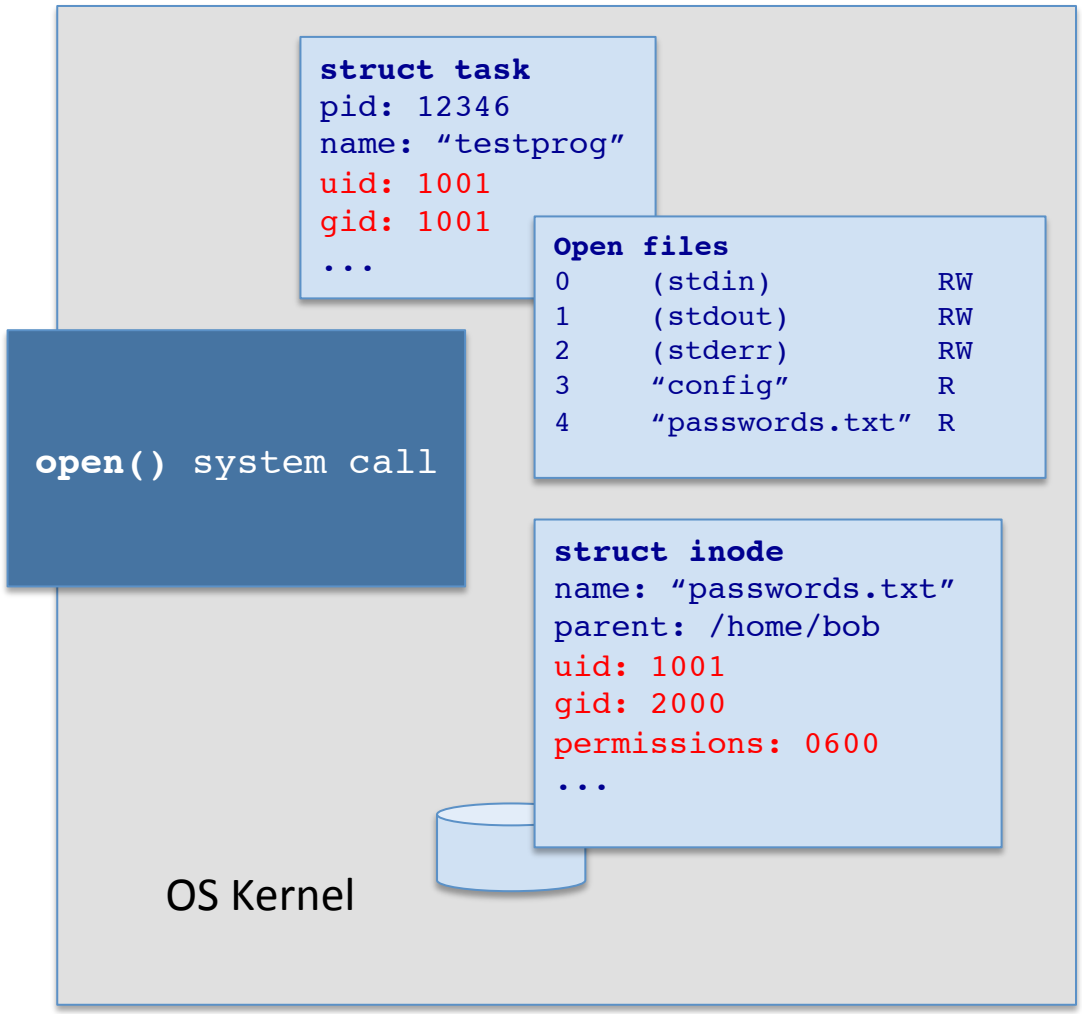
1. testprog calls the open() system call with the given file name



Acquiring a file descriptor

```
testprog
...
int fd = open("/home/bob/
  passwords.txt", O_RDONLY);
read(fd, buf, 8);
...
```

2. open() checks testprog's user and group id's against the file's access permissions

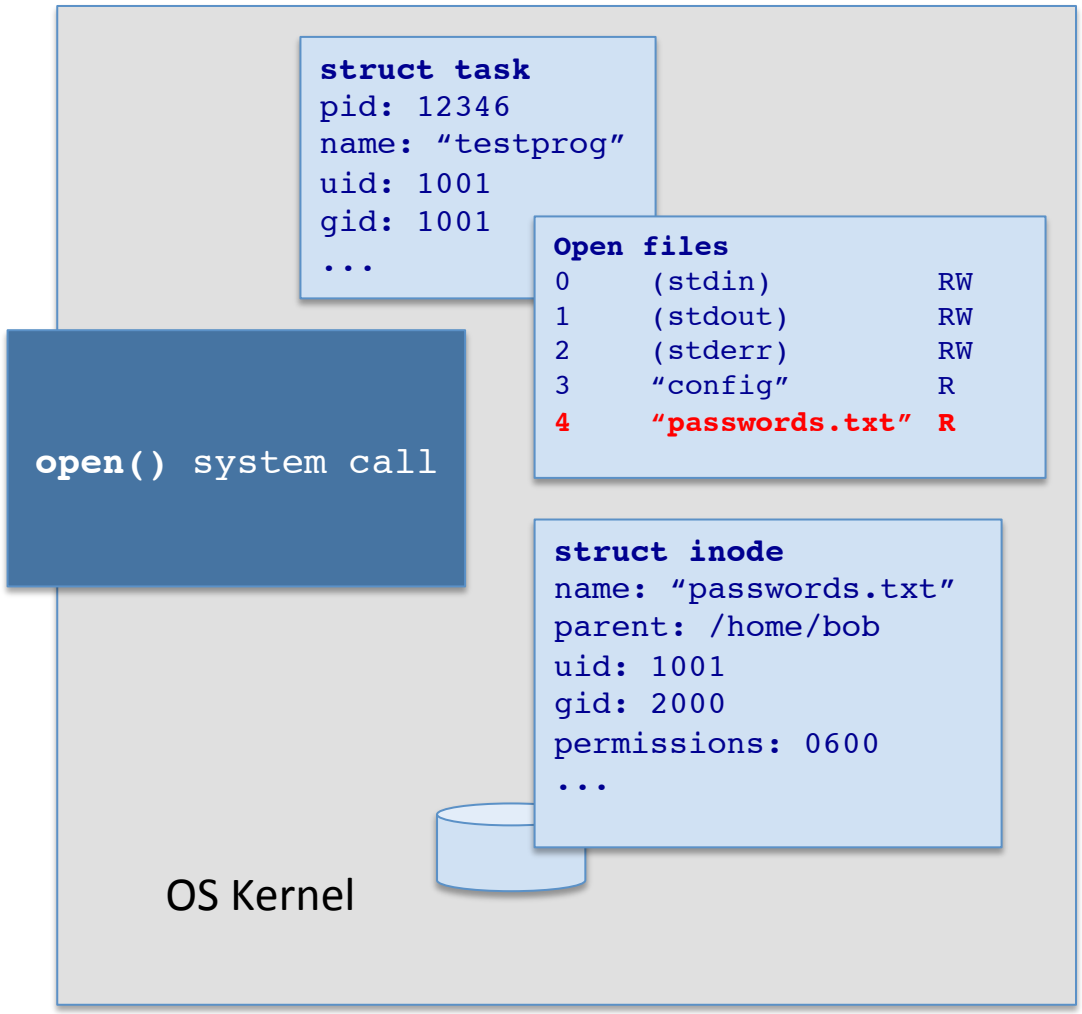


Acquiring a file descriptor

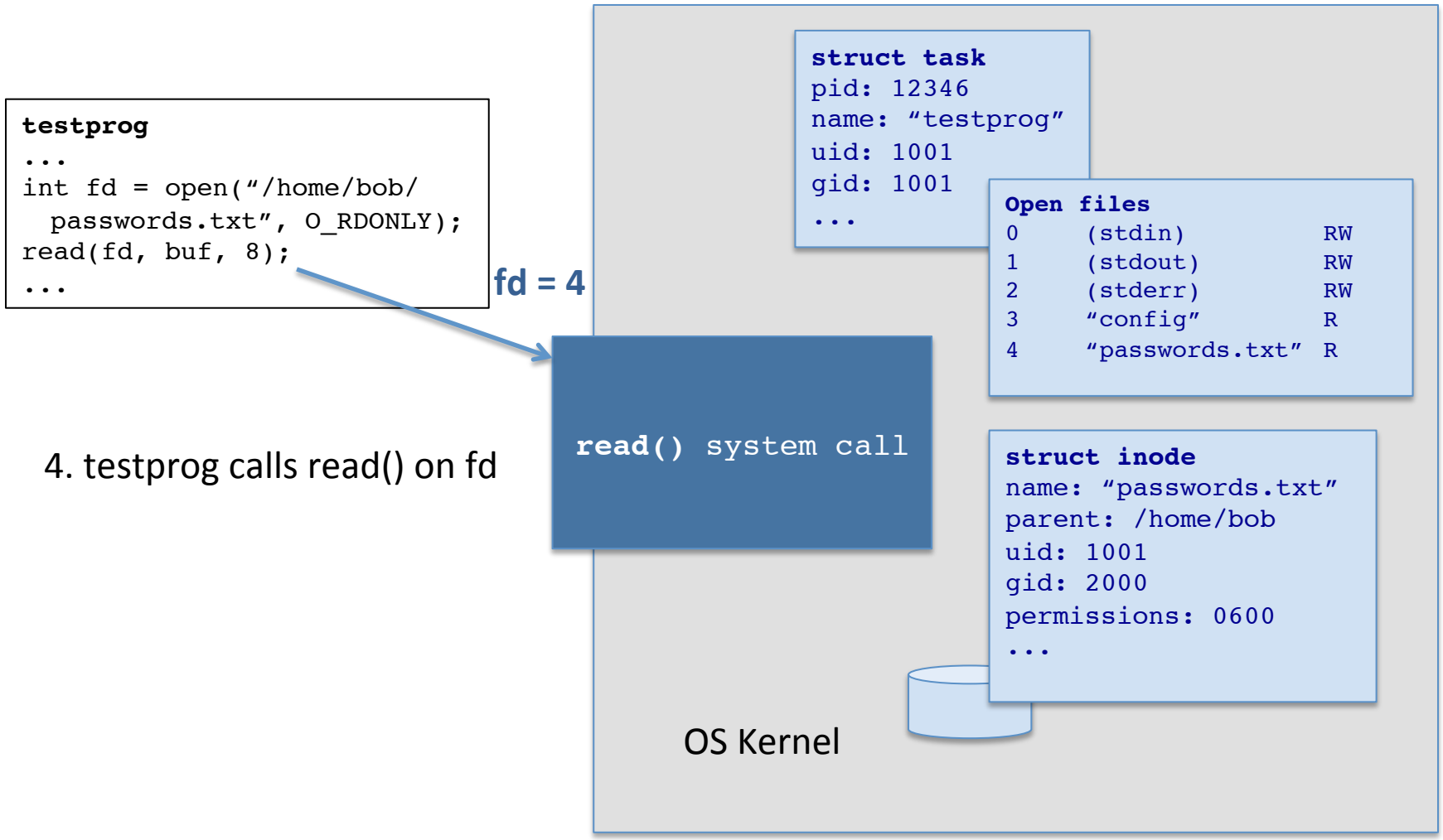
```
testprog
...
int fd = open("/home/bob/
  passwords.txt", O_RDONLY);
read(fd, buf, 8);
...
```

fd = 4

3. Kernel creates a new open file structure for testprog, stores it in testprog's list, and returns the index



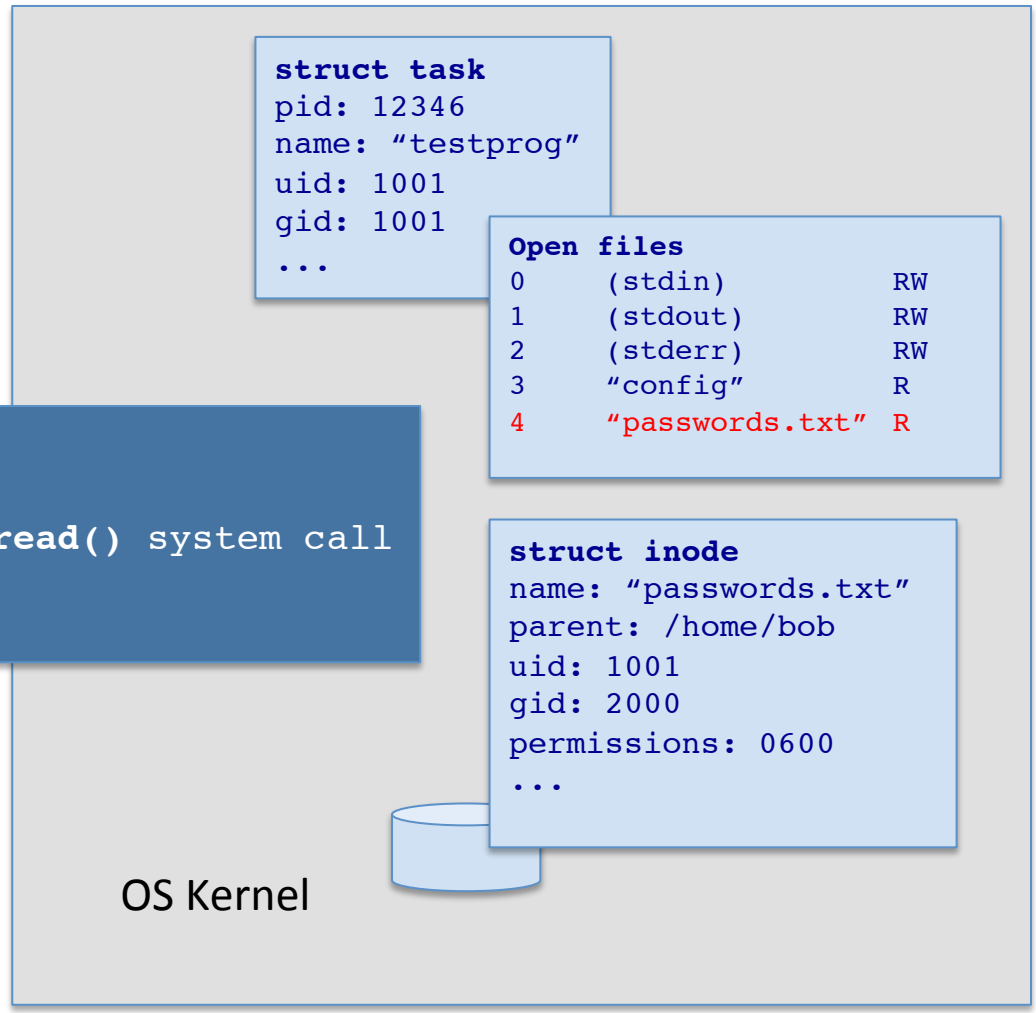
Using a file descriptor



4. testprog calls read() on fd

Using a file descriptor

```
testprog
...
int fd = open("/home/bob/
  passwords.txt", O_RDONLY);
read(fd, buf, 8);
...
```

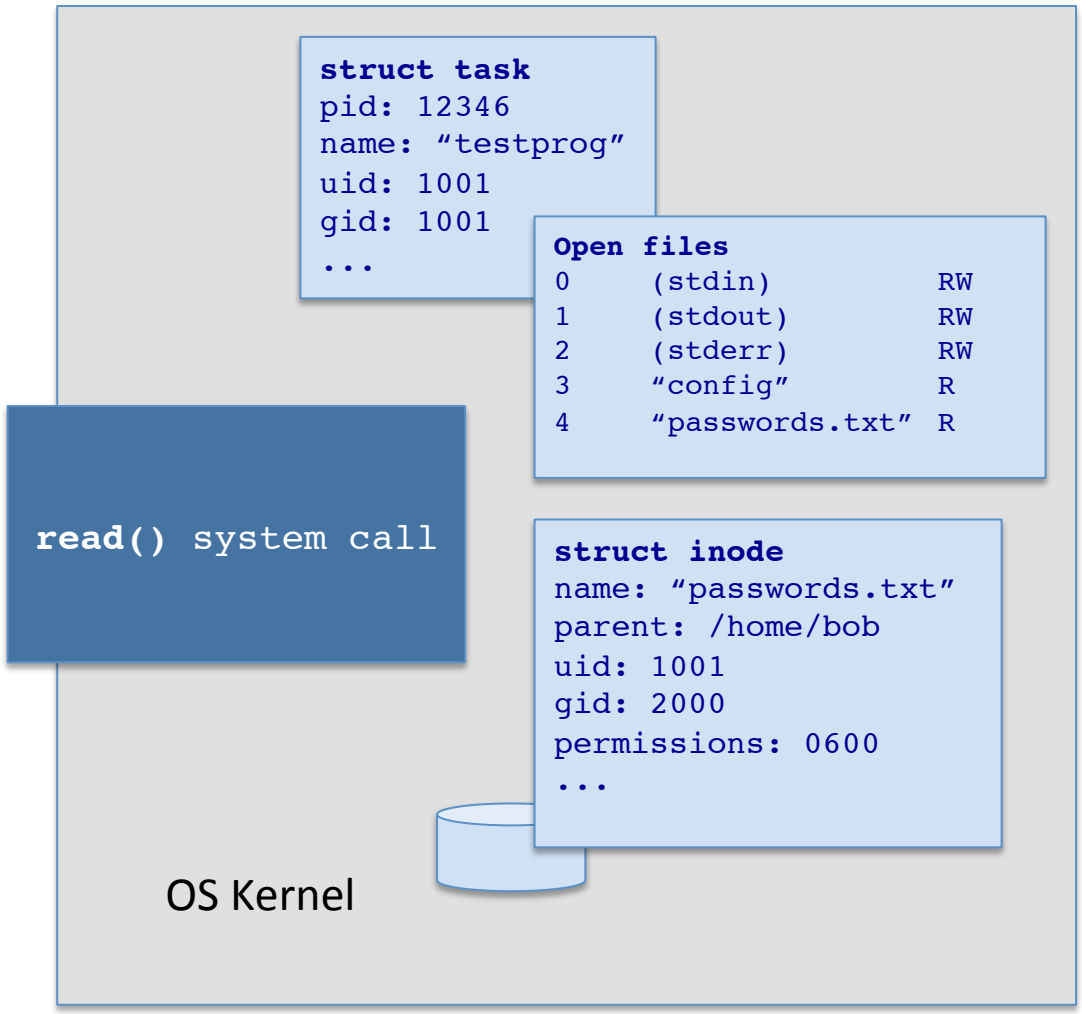


5. Kernel checks access rights for entry #4 in testprog's list for "R". Check succeeds!

Using a file descriptor

```
testprog
...
int fd = open("/home/bob/
  passwords.txt", O_RDONLY);
read(fd, buf, 8);
...
```

6. Kernel reads data from the file, returns it to user space.



File Descriptors as Capabilities

Entries in this table are capabilities!

Open files

Index	Name	Rights	Position	...
0	(stdin)	RW		
1	(stdout)	RW		
2	(stderr)	RW		
3	config	R	123	...
4	passwords.txt	R	0	...

Benefits of Capability Systems

- Must give the capability along with any access request
  - Reduces the potential for "confused deputy" problems
- Can remove shared name spaces
  - File system
  - Process ID's
- May lead to simpler design

Confused Deputy Problem

- Program has legitimate access to two files
  - Attacker tricks it into using them incorrectly

- Example: Compiler
  - Imagine that the system restricts access to files by program (not just by user id), and it charges you to compile programs
  - So gcc saves billing information into some file each time it compiles a program for you

Confused Deputy Problem

- Say for example that billing info goes in
  - /var/gcc/billing
- What happens when you do this?
  - gcc -o /var/gcc/billing game.c server.c
- This really happened!
  - http://www.cis.upenn.edu/~KeyKOS/ConfusedDeputy.html

Challenges with capability models

- How does each program *get* its capabilities to begin with?
  - Chicken-and-egg problem
  - In the example with file descriptors, the system uses an ACL to decide whether or not to grant the capability!
  - Solution? Persistence (EROS and KeyKOS)
- Biggest cause of this problem: File systems

Practical Capability Systems (?)

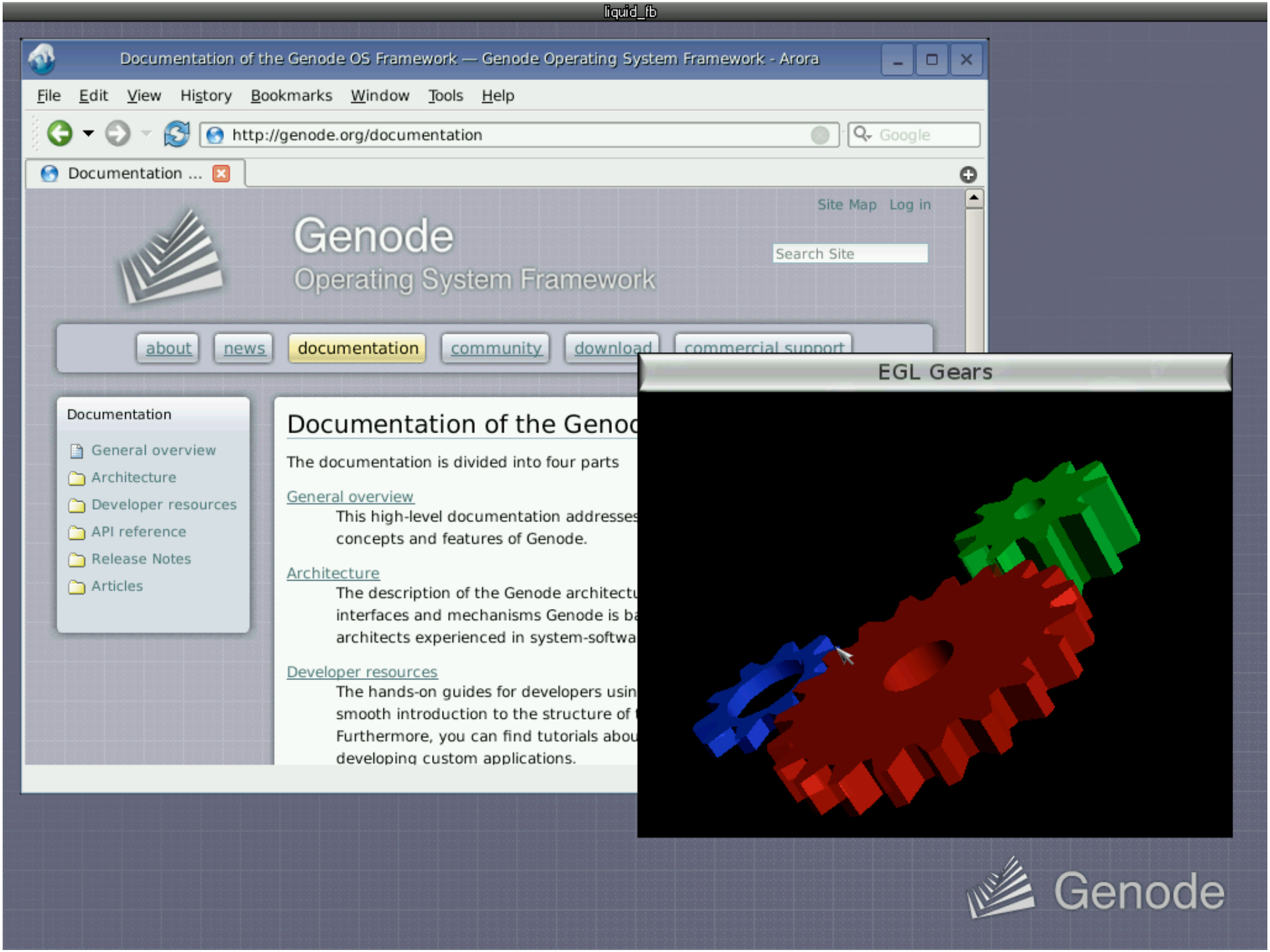
- seL4: Formal Verification of an Operating System Kernel
  - Formally verified correctness and security of a tiny OS microkernel based on L4
  - http://ssrg.nicta.com.au/projects/seL4/

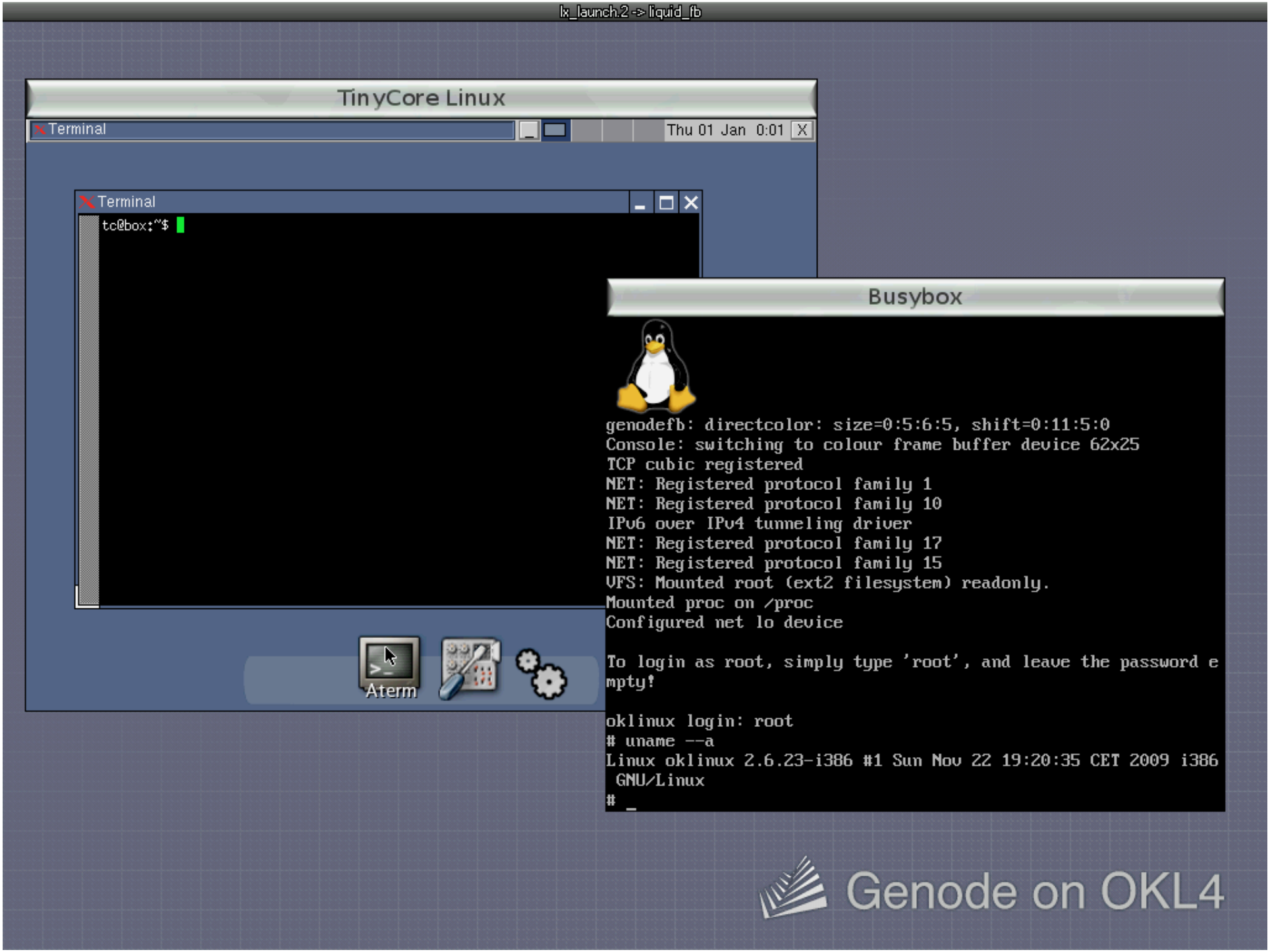
Other recent work in the open source community

- Many active projects
  - Fiasco.OC
    - http://os.inf.tu-dresden.de/fiasco/
  - HelenOS/SPARTAN
    - http://www.helenos.org/
  - Genode
    - http://www.genode.org/
  - NOVA microhypervisor
    - http://www.hypervisor.org/
- New systems are gaining functionality quickly
  - Will this be like Linux in the 1990's?
  - Will they be viable for real use soon? Are they already?

Microkernels

- Any functionality not absolutely required in the kernel is moved into “user space” (applications)
- Very small code size
  - Fewer bugs?
  - Fewer vulnerabilities?
- Better separation of privileges
  - Principle of least authority (POLA)


liquid_fb
Documentation of the Genode OS Framework — Genode Operating System Framework - Arora
File Edit View History Bookmarks Window Tools Help
http://genode.org/documentation
Google
Documentation ...
Site Map Log in
Genode
Operating System Framework
Search Site
about news documentation community download commercial support
Documentation
General overview
Architecture
Developer resources
API reference
Release Notes
Articles
Documentation of the Genoc
The documentation is divided into four parts
General overview
This high-level documentation addresses
concepts and features of Genode.
Architecture
The description of the Genode architectu
interfaces and mechanisms Genode is ba
architects experienced in system-softwa
Developer resources
The hands-on guides for developers usin
smooth introduction to the structure of
Furthermore, you can find tutorials abou
developing custom applications.
EGL Gears
Genode


lx_launch.2 -> liquid_fb
TinyCore Linux
Terminal
Thu 01 Jan 0:01
Terminal
tc@box:~$
Aterm
Busybox
genodefb: directcolor: size=0:5:6:5, shift=0:11:5:0
Console: switching to colour frame buffer device 62x25
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 10
IPv6 over IPv4 tunneling driver
NET: Registered protocol family 17
NET: Registered protocol family 15
VFS: Mounted root (ext2 filesystem) readonly.
Mounted proc on /proc
Configured net lo device
To login as root, simply type 'root', and leave the password e
mpty!
oklinux login: root
uname --a
Linux oklinux 2.6.23-i386 #1 Sun Nov 22 19:20:35 CET 2009 i386
GNU/Linux
_
Genode on OKL4

Virtualization for Confinement

- Full-system Virtualization
- OS-level Virtualization
- Application-level Sandboxing

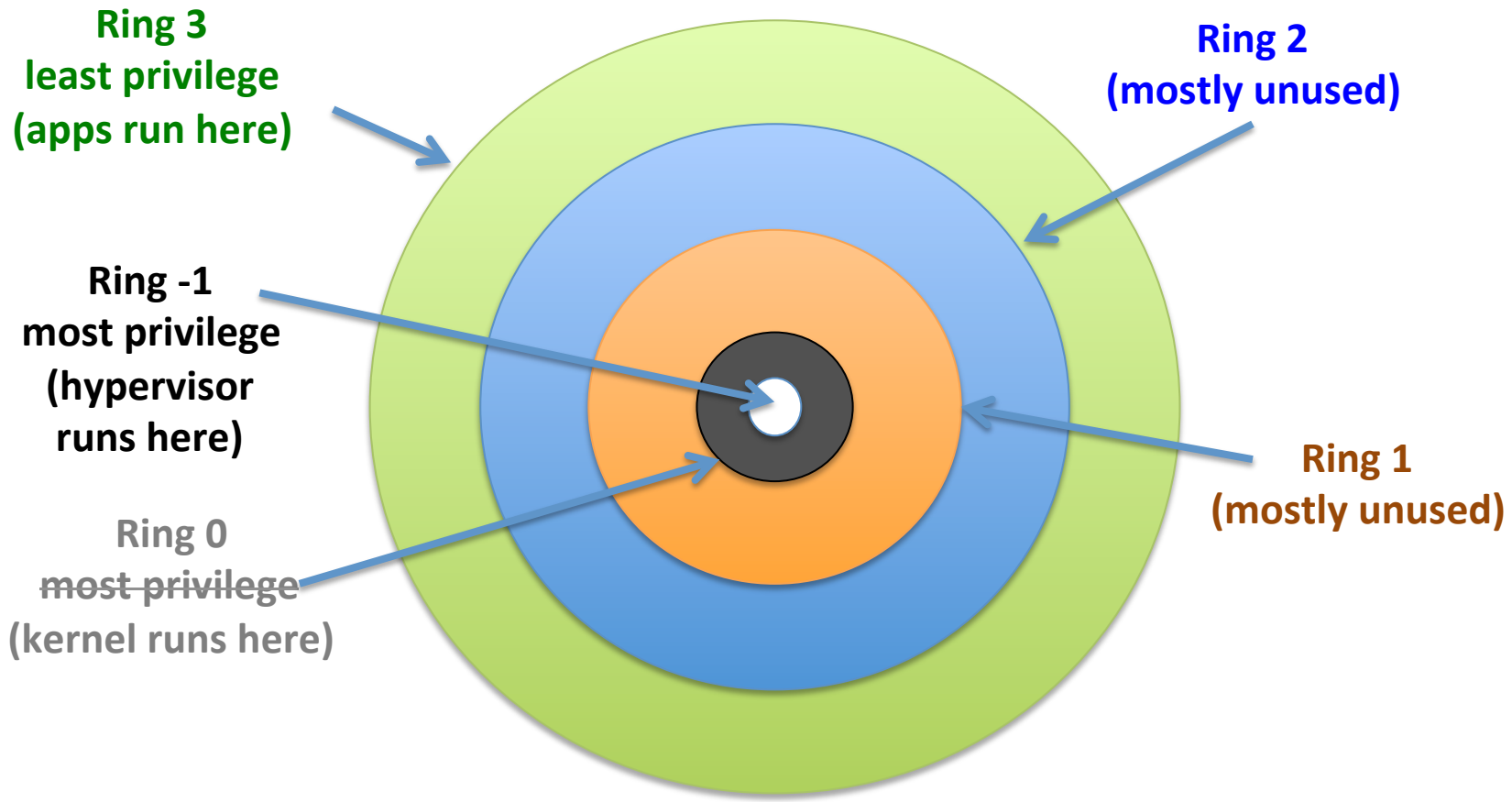
System-Level Virtualization

- Run multiple copies of unmodified OS's
  - OS expects to run in Ring 0
  - OS expects full access to hardware
- Protect guest OS's from each other
  - Control access to code (instructions)
  - Control access to data (memory)

System-Level Virtualization

- Examples
  - VMWare ESX / Workstation / Fusion
  - VirtualBox
  - KVM
  - Xen

Hardware Privilege Levels



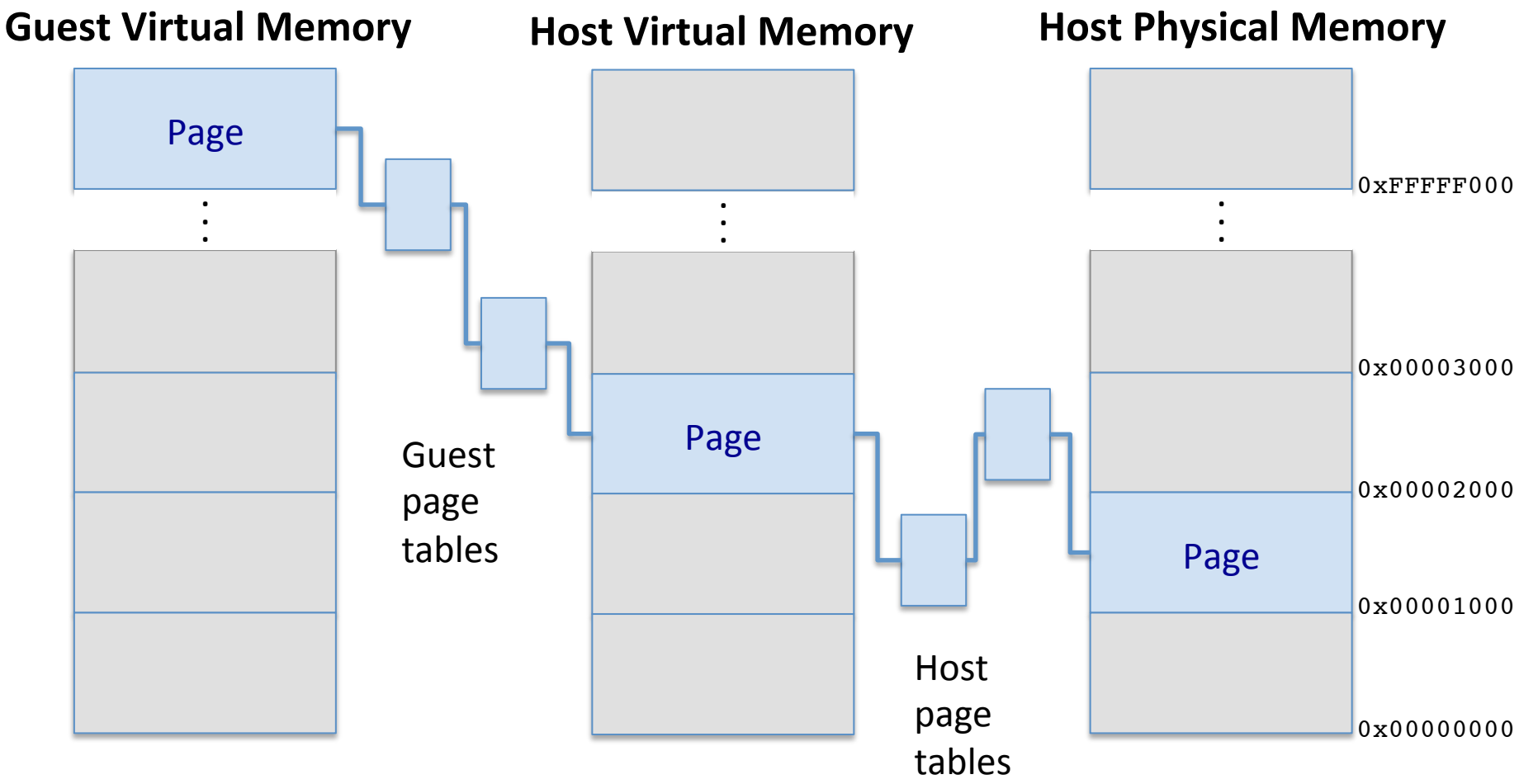
Protecting Instructions

- Intel VT extensions and AMD-V add new "even more privileged" instructions for managing VM's from ring -1
  - vmenter, vmexit, and friends
- Hypervisor sets up interrupt handlers, like the OS does
  - Virtualization HW calls these to service OS's requests
- New extensions enable creation of interrupts to the hypervisor for fine-grained monitoring
  - On page faults
  - On execution of given instructions
  - On memory reads or writes
  - …

Virtualization: Memory Challenges

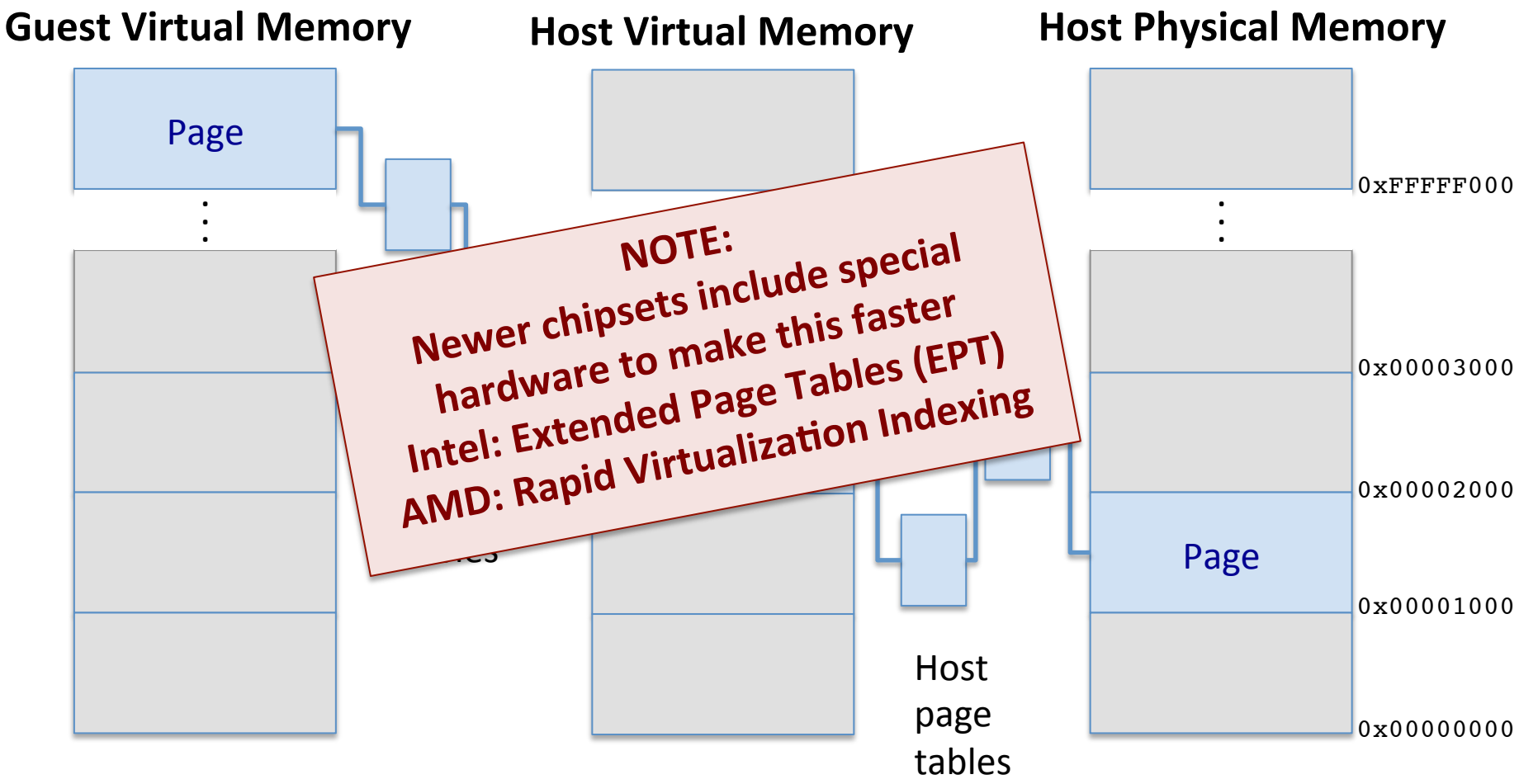
- Guest OS's expect full access to the physical (linear) address space
- Guest OS's want direct access to devices
- How to share the system's memory across multiple OS's at the same time?

Virtualized Virtual Memory



(Yo, we heard you like virtual memory… Xzibit would be proud.)

Virtualized Virtual Memory



(Yo, we heard you like virtual memory… Xzibit would be proud.)

OS-level Virtualization

- All processes share a single instance of the OS kernel
- Processes may get a different "view" of the rest of the system
  - Network
  - Filesystem
  - Process list
- Operating system enforces confinement of processes

OS-level Virtualization

- Examples:
  - Unix “chroot”
  - FreeBSD “jail”
  - Solaris “zones”
  - Linux LXC / OpenVZ / Docker

Application-Level Sandboxing

- Goal: Better confinement of individual applications within the OS

Application-level Sandboxing

- Capsicum: Practical Capabilities for Unix
  - Added support for “capability mode” in FreeBSD
  - http://www.cl.cam.ac.uk/research/security/capsicum/
- Process voluntarily enters “capability mode”
  - Old Unix syscalls no longer work – all denied
  - Process must use new capability-enabled versions

Capsicum Capability Mode

- Old code requests resources from the OS without providing a capability
  - `int fd = open("/home/bob/passwords.txt", O_RDONLY);`
- New code adds capability to the request
  - `int fd = open(dir_fd, "/home/bob/passwords.txt", O_RDONLY);`
- OS uses the capability to decide whether to allow or deny the new access

Linux Seccomp

- Process makes the “seccomp” system call to signal it wants to start executing in “secure computing” mode
- Kernel then disallows all syscalls except
  - Read
  - Write
  - Sigreturn
  - Exit

Linux Seccomp

- If file descriptors are capabilities, then…
- Is seccomp just another implementation of capability mode?

Linux Seccomp-BPF

- Recent Linux kernels provide a very basic mechanism for filtering system calls
  - Allow or deny the syscall based on its arguments
  - Re-uses filtering mechanism built for firewalling network packets
  - Available to user-level processes
  - https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt

Linux Seccomp Sandboxing

- Mbox – Generic application sandboxing framework
  - http://pdos.csail.mit.edu/mbox/
- Firefox and Chrome sandboxes
  - Used to confine plugins like Adobe Flash
  - https://wiki.mozilla.org/Security/Sandbox/Seccomp
  - https://code.google.com/p/chromium/wiki/LinuxSandboxing

OpenBSD pledge

- OpenBSD has a new application sandboxing mechanism, called “pledge”
  - See http://www.openbsd.org/papers/hackfest2015-pledge/mgp00004.html
    (Warning: Offensive content on ‘prev’ link)
- Similar idea to seccomp-bpf and capsicum
- Very simple
  - Application tells the OS which categories of system calls it should be allowed to make
  - OS rejects any other system call – terminates program

BACKUPS