IA-64 Linux Kernel Internals

David Mosberger
Hewlett-Packard

Don Dugger
VA Linux Systems

Agenda

- Trillian Project Overview
- IA-64 Linux Kernel Technical Details
- IA-32 Support
- IA-64 Linux Demos
- Summary
- Question and Answer Session

The Trillian Project

- Goals
  - **Single IA-64 Linux port**
  - **Optimized for IA-64**
  - **Open source availability at or before Itanium™ processor launch**
    - **Source code released on 2/2/00 at www.kernel.org**
- Co-operative effort to deliver the best code
  - **Similar to classic Linux model**
  - **Many players contributing technology and resources**
    - **Caldera, CERN, HP, IBM, Intel, Red Hat, SGI, SuSE, TurboLinux, and VA Linux Systems**

Visit http://www.ia64linux.org for more details

The Team – Founding Members

Company	*Tasks*
HP	kernel, initial gcc, gas, ld, emacs
IBM	performance tools, measurement, and analysis
Intel	kernel, IA-32, platform, apache,EFI, FPSWA, SCSI, SMP, libm
Red Hat (Cygnus)	GNUPro Toolkit (gcc, g++, gdb)
SGI	compiler, kdb, OpenGL
VA Linux Systems	kernel, platform, E, E-Term, XFree86, cmds & libs, bootloader, SMP, IA-32

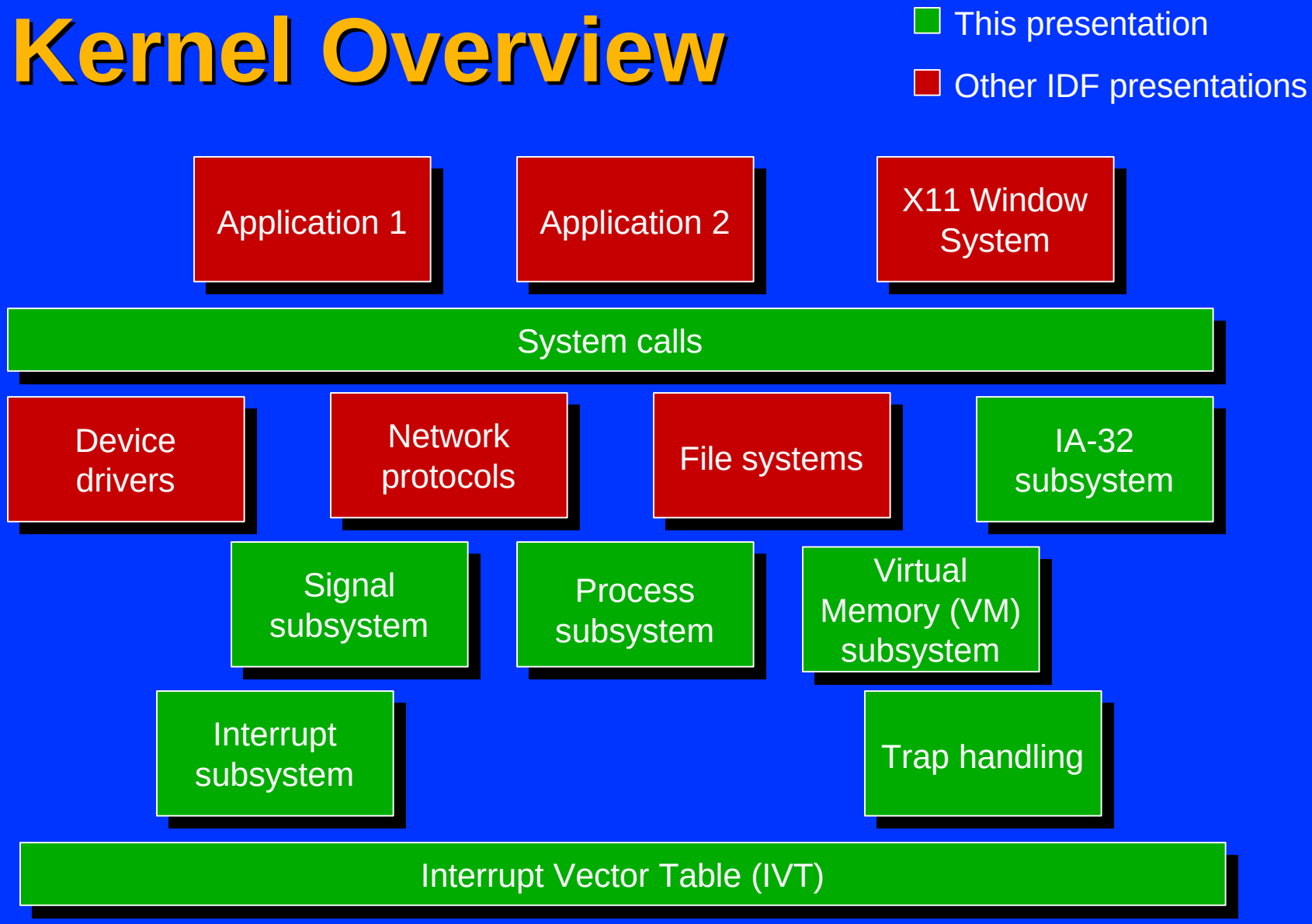
The Team – Contributing Members

Company	*Tasks*
Caldera	distribution
CERN	glibc
Red Hat	Commands, GNOME, distribution
SuSE	KDE, distribution
TurboLinux	performance counters, distribution

Design Goals & Approach

- Pure 64-bit kernel for IA-64 (no legacy)
- APIs compatible with Linux/x86 wherever possible (e.g., error-, signal-, ioctl-codes)
- Minimize changes to platform-independent code (started with 2.1.126, now at 2.3.35)
- Optimize for 64-bit performance
- Follow standards whenever possible: IA-64 SW conventions, EFI, DIG, UNIX ABI, etc.

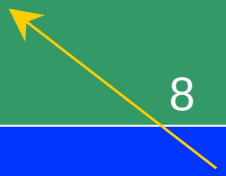
Kernel Overview



Global Kernel Properties

- **Data model: LP64**

Type	*Size*	*Alignment*	*Type*	*Size*	*Alignment*
char	1	1	float	4	4
short	2	2	double	8	8
int	4	4	long double	10	16
long int	8	8			
long long int	8	8	void *	8	8



with current gcc: size=8, align=8

- **Byte order:**
  - **little-endian is native byte order**
  - **big-endian processes are possible**

Kernel Register Usage

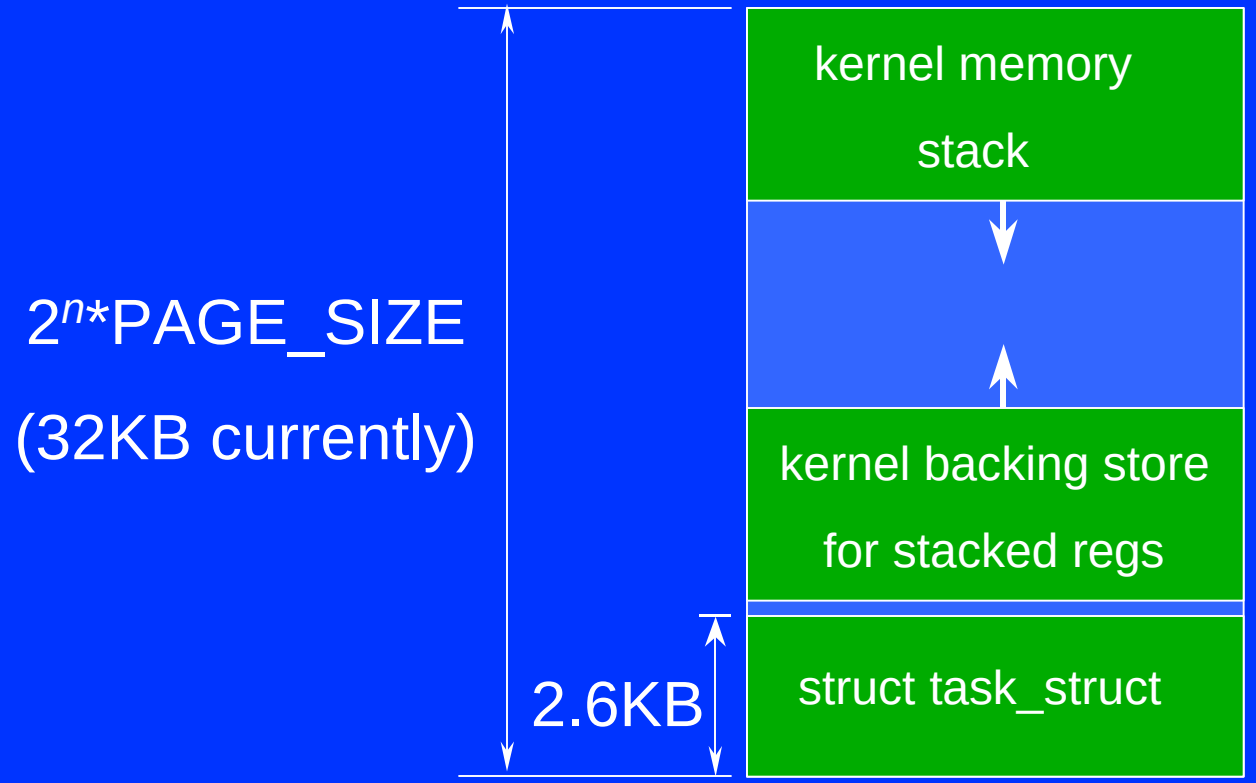
- Follows SW Conventions standard except:
  - f10-f15 and f32-f127 are not available in kernel
  - Note: other fp regs are available in kernel-mode
    - needed for integer multiply (uses fp regs)
    - good for certain ops, e.g., “find highest bit set”
    - considering a change to only provide f6-f11 to the compiler for integer multiply and divide
- Current kernel register usage:

```
  r13: current task pointer ("thread pointer")
ar.k0: legacy I/O base addr (as per PRM)
ar.k5: fph owner
ar.k6: phys addr of current task
ar.k7: phys addr of page table
```

  - planned changes: use bank 0 registers instead

Process Subsystem

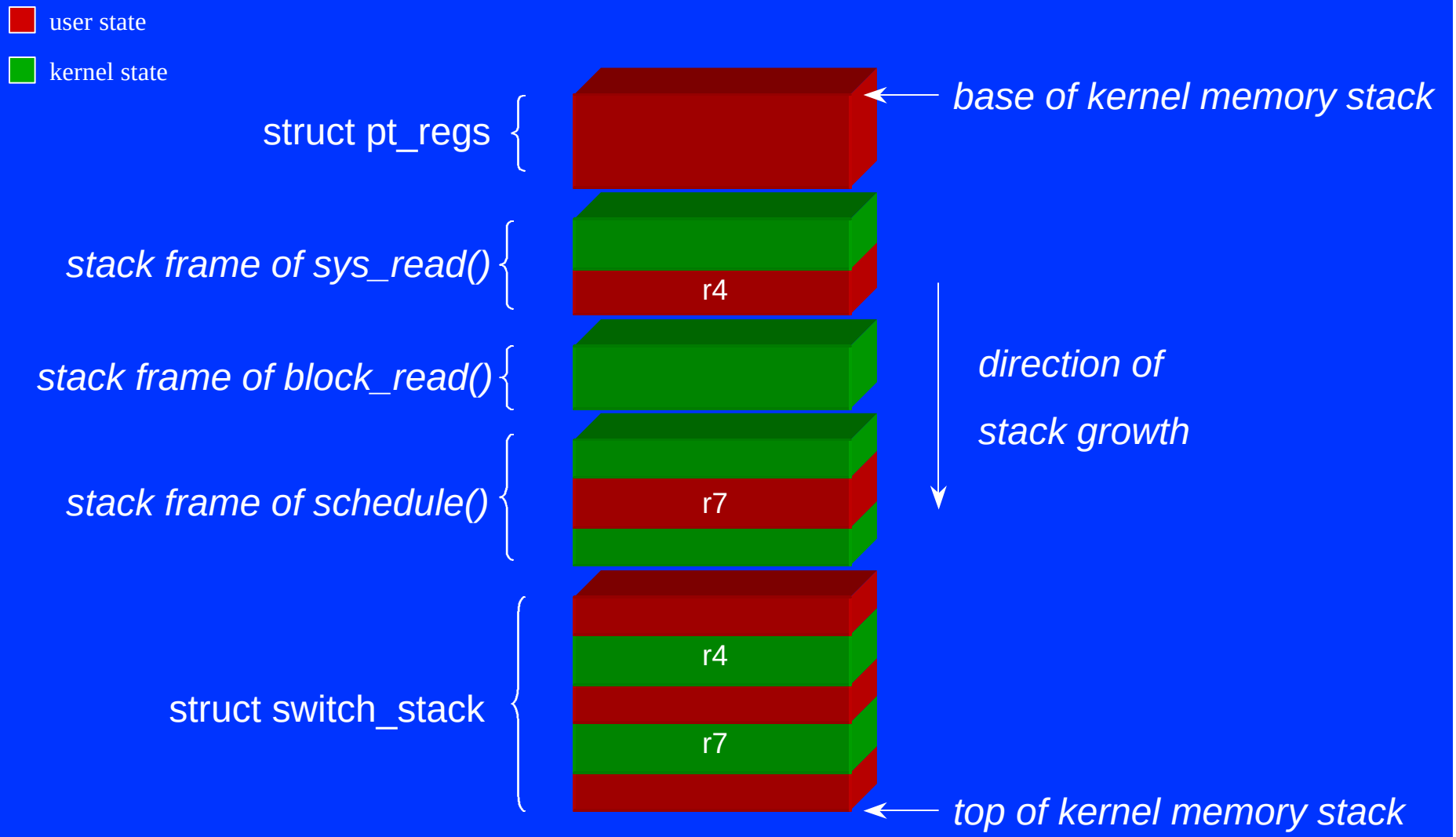
- **Kernel task structure:**



Process State

- struct pt_regs:
  - allocated on kernel mem stack on kernel entry
  - contains “scratch” registers (~400 bytes)
- struct task_struct:
  - allocated on kernel mem stack when blocking execution (context switch)
  - contains “preserved” registers (~560 bytes)
- struct thread_struct:
  - arch. specific part of struct task_struct
  - contains ksp, lazy state: fph, ibrs, dbrs, ...

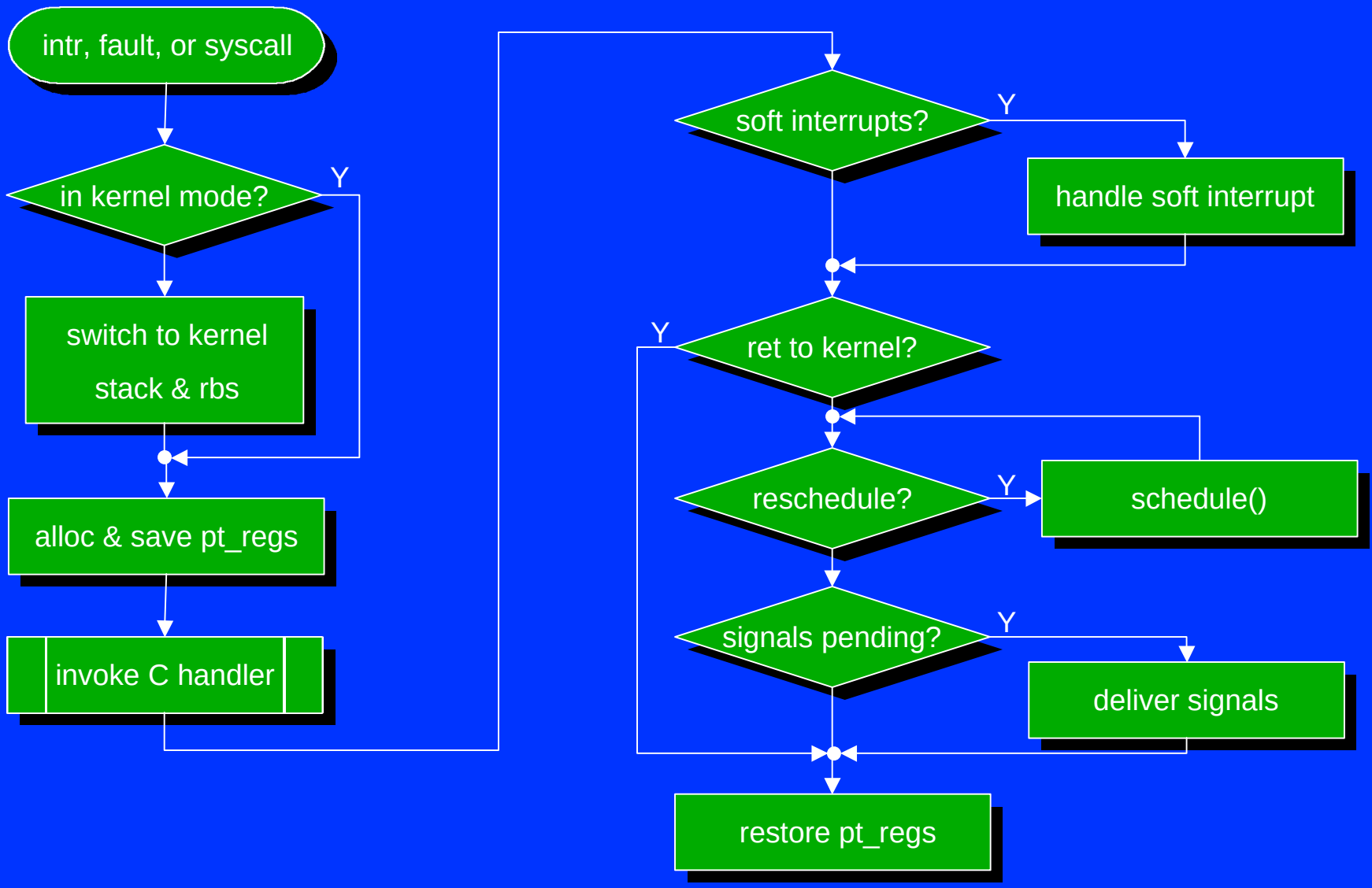
Example of Blocked Process



Lazily Managed State

- floating-point high partition (f32-f127):
  - UP:
    - disable access to when process blocks
    - re-enable access when process resumes
    - take fault & switch context if used another process
  - MP:
    - always save when process blocks
    - alternative: use IPI to fetch state from another CPU
- debug & performance monitor registers:
  - context-switch only if in use

Kernel Entry & Exit



Syscall Invocation

- Currently:
  - via break instruction; e.g., stub for open():

```
        mov r15=1028
        break.i 0x100000
        cmp.eq p6=-1,r10
(p6)    br.cond.spnt __syscall_error
        br.ret.sptk.many b0
```

- Future:
  - use “epc” instruction to optimize syscall path
  - syscall will look like function call into the gate page (kernel mapped execute & promote page)

Syscall Argument Passing

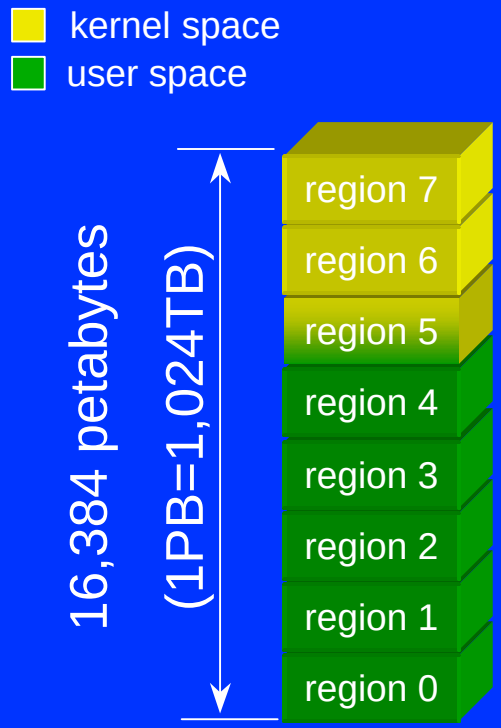
- Naively: pass args on memory stack
  - slow:
    - different from normal SW Conventions
    - need to copy-in args (may fault)
- Better: pass args in stacked registers
  - syscall path must be careful to preserve args across rbs switches on kernel entry & exit
    - avoid “flushrs” like the pestilence
  - to enable efficient syscall restart, syscall handlers may not modify input args
    - indicated by “syscall_linkage” function attribute

VM Subsystem

- **page size:**
  - **kernel configurable to 4, 8, 16, or 64KB**
  - **use getpagesize() to get page size in app (DON'T hardcode any particular value)**
  - **why a choice of page size?**
    - **4KB allows perfect Linux/x86 emulation**
    - **>4KB:**
      - allows for good Linux/x86 emulation (netscape etc.)
      - better for native IA-64 binaries (8 or 16KB best)
      - bigger implemented virtual address space:
        - 2x page size increases implemented VA by 16x
  - **remaining discussion: assume 8KB page size**

Virtual Address Space

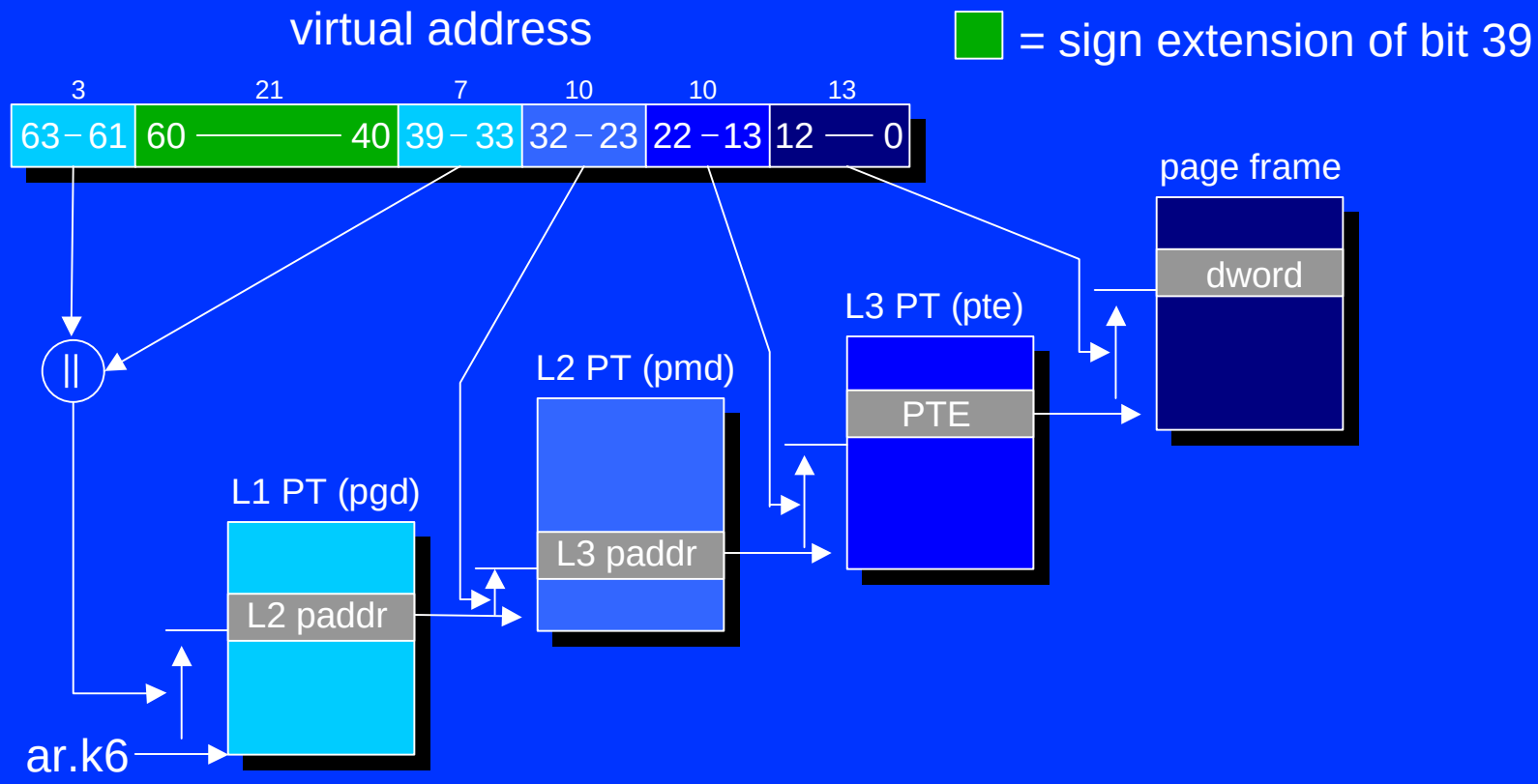
- 8 regions of 61 bits each (2,048 PB)
  - provides headroom for future growth & different mapping properties



Current usage:	*Page size:*	*Scope:*	*Mapping:*
cached	large (256MB)	global	identity
uncached	large (256MB)	global	identity
vmalloc	kconfig (8KB)	global	page-table
stack segment	kconfig (8KB)	process	page-table
data segment	kconfig (8KB)	process	page-table
text segment	kconfig (8KB)	process	page-table
shared memory	kconfig (8KB)	process	page-table
IA-32 emulation	kconfig (8KB)	process	page-table

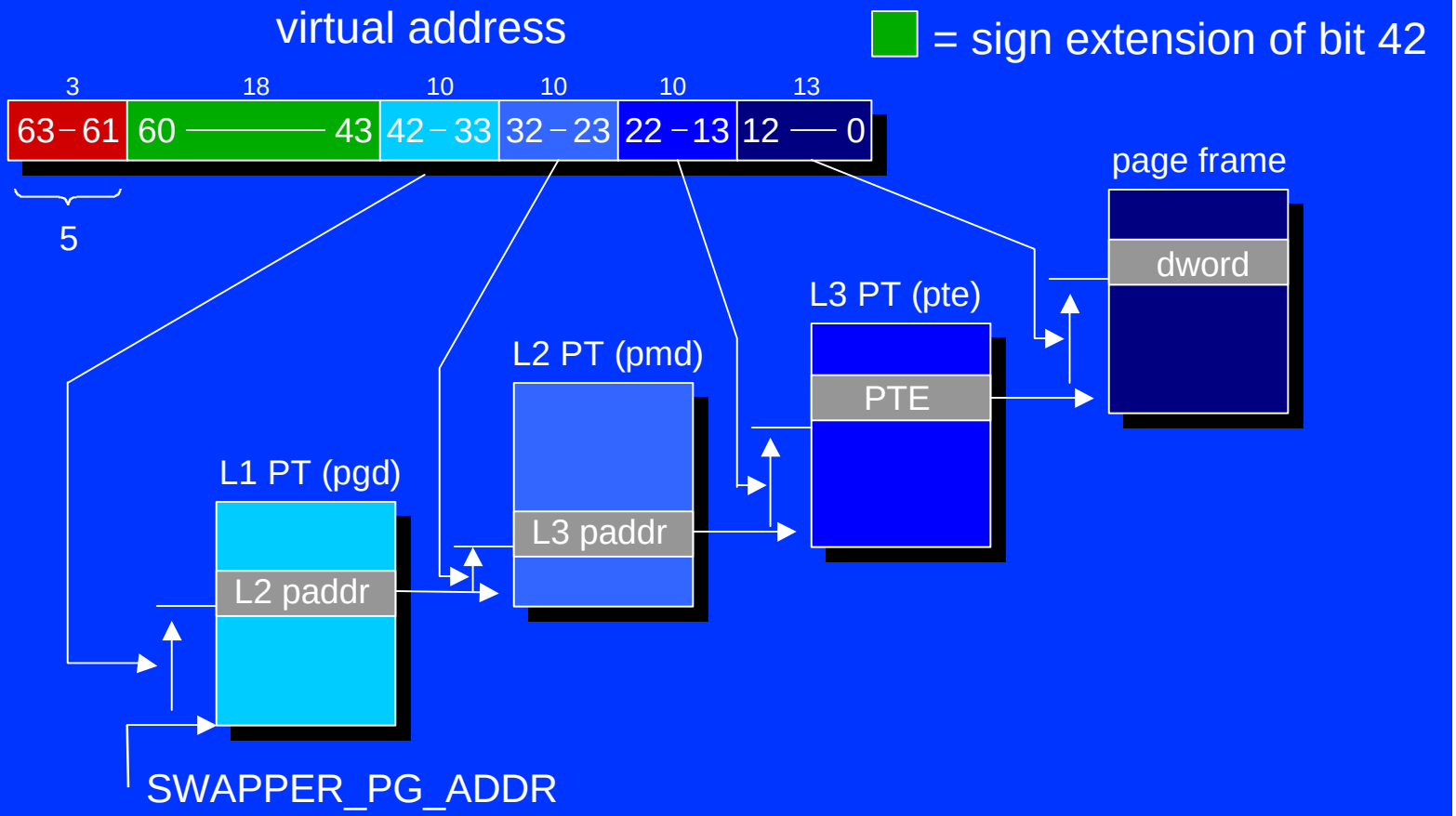
User Regions

- mapped by single 3-level page table
- each region gets 1/8th of level 1 page table



Mapped Kernel Region

- has its own 3-level page table
- full 43-bit address space (w/8KB page size)

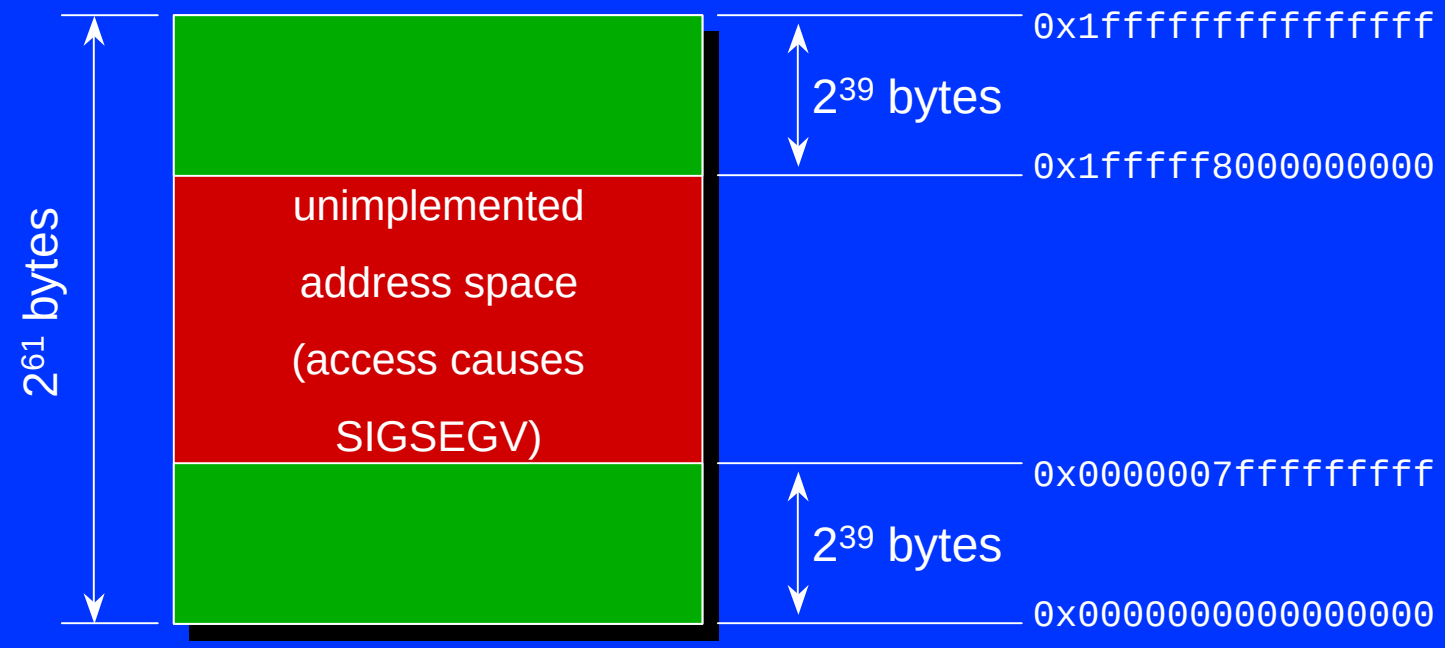


Planned Changes

- Change 3-level PT to 4-level PT
  - 43 bits per region (with 8KB pages)
  - top-level is indexed by region number
  - allow different PT sharing on per-region basis:
    - global (like current region 5)
    - global w/copy-on-modify (for shared libraries)
    - shared (for multi-threading)
    - private (normal UNIX semantics)
- On other platforms, top-level is a no-op

Anatomy of a User Region

- **Within each region, bits 40-59 must be sign-extension of bit 39:**



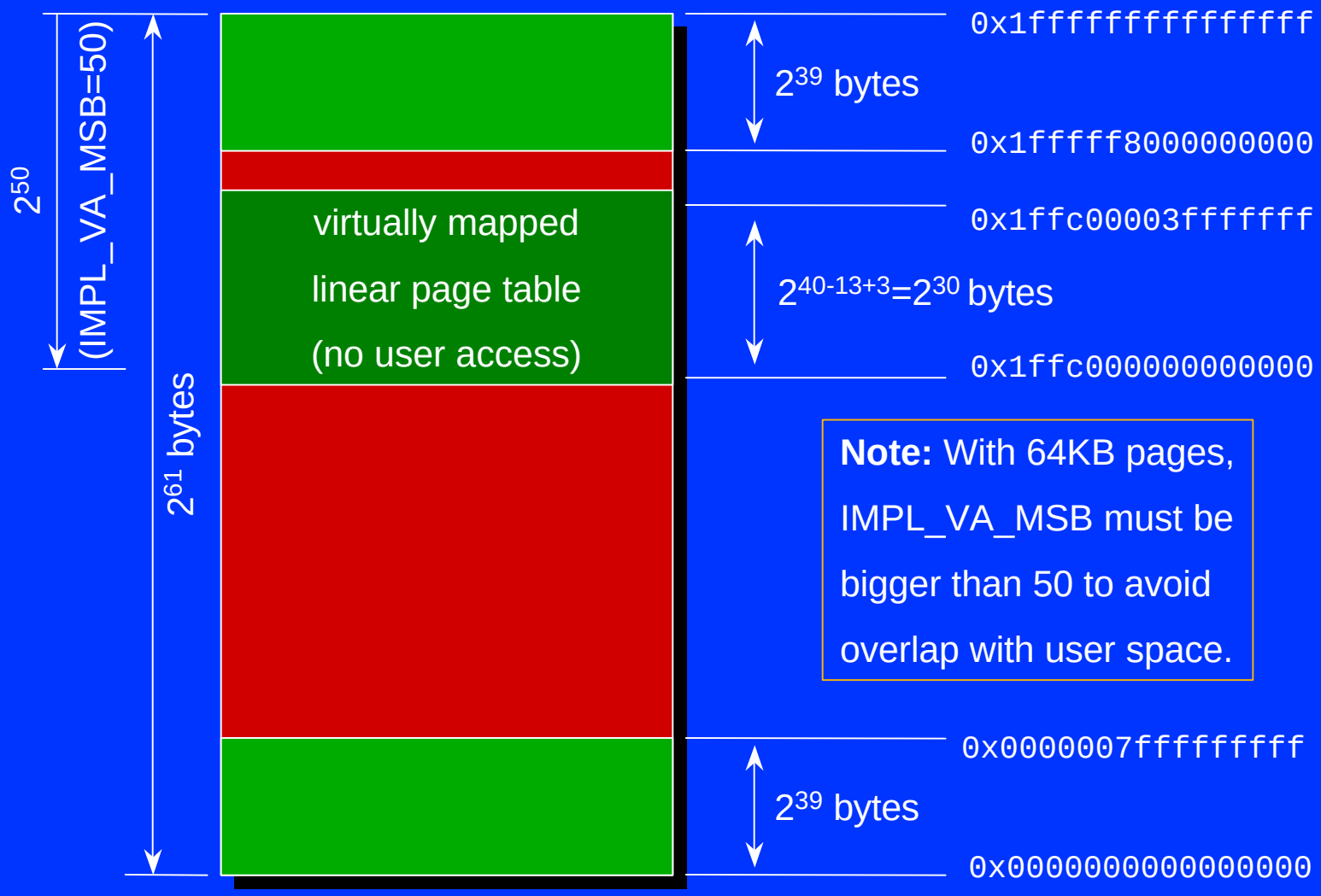
Virtual Hash Page Table (VHPT)

- HW assist to speed up TLB lookup
- Can operate in two modes:
  - long mode (hash table mode):
    - on TLB miss, lookup hash table; if hit, install PTE
  - short mode (virtually mapped linear page table)
    - L3 page table pages linearly mapped into virtual space
    - on TLB miss, access PTE through virtually mapped page table; if no fault, install PTE

VHPT Tradeoffs

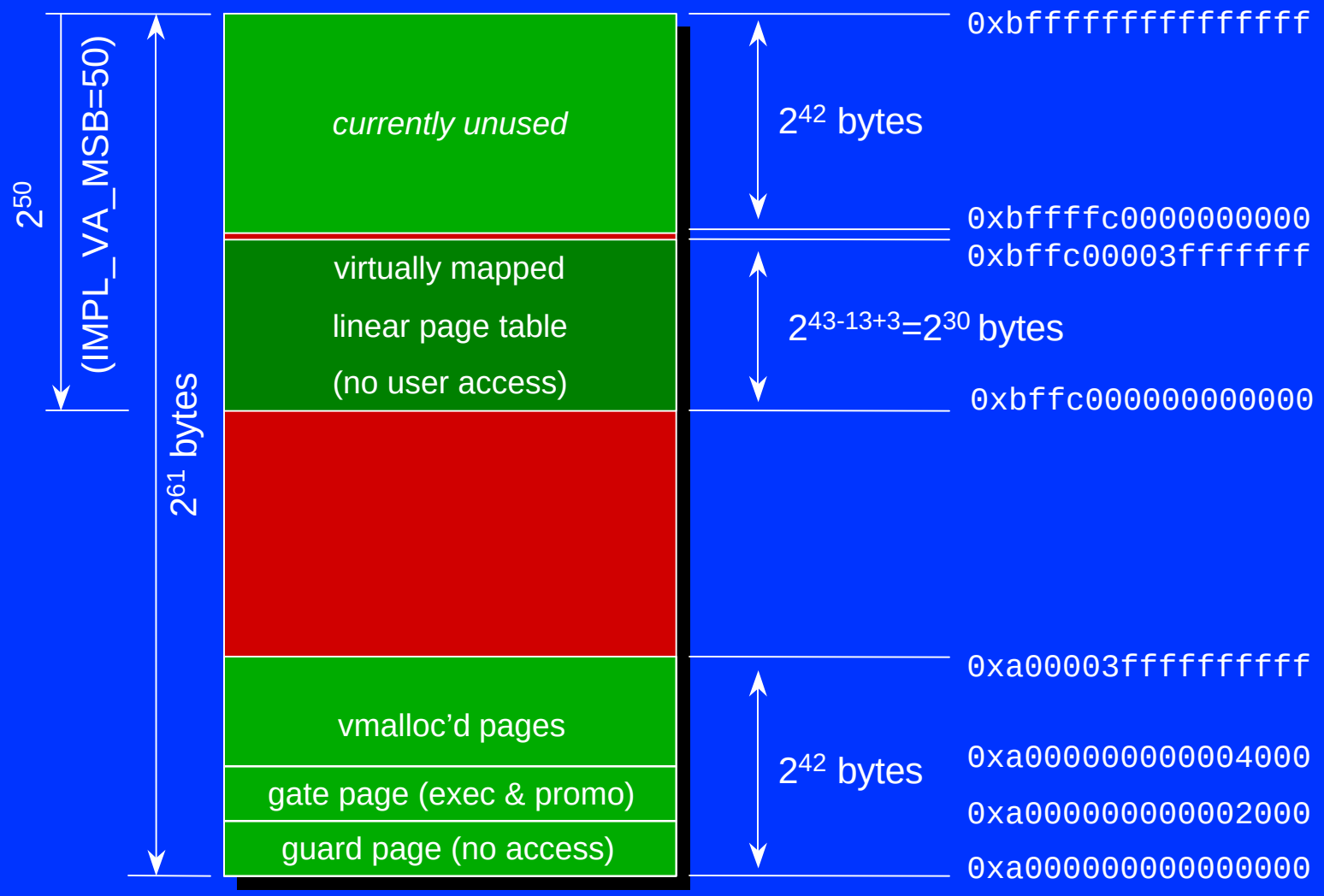
- Long mode (hash table mode)
  - 32 bytes/entry
  - more flexible (e.g., page size can vary per PTE)
  - good for extremely sparse access patterns
  - duplicates information in the page tables
- Short mode (virtually mapped linear PT)
  - 8 bytes/entry (same memory as PTs)
  - less flexible, but almost perfect fit for Linux
  - great for reasonably dense access patterns (e.g, LPT PTE maps 8MB of physical memory)
  - needs up to 2x the TLB entries as long mode

Anatomy with VMLPT



Note: With 64KB pages, IMPL_VA_MSB must be bigger than 50 to avoid overlap with user space.

Anatomy of Kernel Region 5



Signal Subsystem

- Normal Linux way of delivering a signal:
  - save machine state (pt_regs & switch_stack)
  - build signal frame on user stack
  - dynamically generate code to call signal handler in the signal frame
  - change pt_regs to make return address point to dynamically generated trampoline code
  - return from kernel to user mode

Signal Subsystem (cont.)

- Several issues with this approach:
  - lots of machine state to save
  - saving entire machine state requires flushrs
  - generating code on the fly requires icache flush
  - rbs cannot easily be switched in kernel because some user register may be on kernel rbs

Signal Subsystem (cont.)

- Solution:
  - save only scratch state (unless PF_PTRACED)
    - if signal handler wants to access preserved state, use unwind library to find correct location
    - avoids flushrs, unless sigaltstack()
  - use static trampoline in gate page
  - code to switch rbs (if necessary) is in static trampoline, which is executed in user-mode
- Result:
  - signal invocation only slightly slower than x86 (at same clock freq), despite larger state!

Miscellany: FPSWA Handling

- How to handle floating-point sw assist faults?
  - since architecture logically provides full IEEE fp arithmetic, FPSWA handler is provided by Intel in the form of an EFI driver:
    - provided as a binary-only module
    - normally in firmware, but can be loaded at boot-time
    - extensively tested for correctness
    - Intel will treat bugs in FPSWA like CPU “erratas”
    - boot-loader detects presence of FPSWA driver and passes callback entry point to kernel
    - on FPSWA fault, kernel invokes callback in virtual mode
    - anyone free to implement their own FPSWA handler

Miscellany: ACPI Parsing

- Problem:
  - unlike any other platform so far, IA-64 requires AML parsing to boot the system (e.g., to get interrupt routine info)
    - complex
    - would add a lot of kernel bloat
- Solution:
  - put AML parser in boot-loader and pass necessary info directly to kernel
  - all other AML parsing done at user-level

Lessons

- predicates really neat:
  - single store/load preserves 64 control-flow bits; saving this word also saves preserved predicates: great for optimizing code with complex control-flow, such as OS kernel
- stacked registers automatically adjust context switch cost:
  - programs with large register working set:
    - higher cswtch time, but benefit from more registers
  - programs with small register working set:
    - no penalty for unused registers

Lessons

- lazy fph management great for context switch performance
  - Corollary: DON'T touch f32-f127 frivolously!
- address space regions useful for:
  - implementing different sharing policies
    - globally shared vs. process-private
  - decoupling implemented virtual address space from address space layout

IA-32 Support Goals

- Provide a 64-bit OS that also supports 32-bit processes
- Not an OS for 32-bit processes that also supports 64-bit processes

Linux IA-64 is a true 64-bit OS!

IA-32 Support Capabilities

- User-level instructions
  - Application processes only (no drivers)
  - No Mixing of IA-64 and IA-32 instructions
- Kernel Services (handled by IA-64 Linux Kernel)
  - Page faults
  - Device interrupts
  - Device drivers

IA-32 Support Status

- IA-32 processes
- Dynamic libraries
  - No change to RTLD (Run-Time Loader)
- System calls
  - Some data structures are different
    - 32-bit longs vs. 64-bit longs

IA-32 Support Status

- System Calls (cont.)
  - Transparently translated by the IA-64 Linux Kernel
    - Shim code in the kernel does the translations
    - Only needed for certain system calls (exec, getdents, gettimeofday, ioctl, etc.)
    - Most calls require no changes since they only pass integers
  - Different page size
    - 16KB vs. 4KB
    - Mainly affects the ‘mmap’ system call

IA-32 Support Status

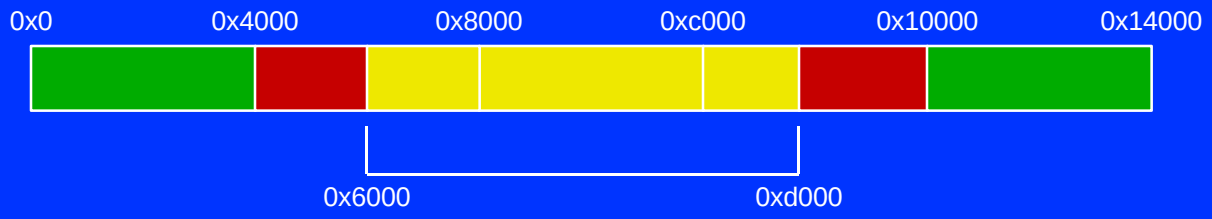
- mmap calls
  - *Good* calls
  - Just pass on to IA64 syscall handler
    - mmap(0, 0x6000, PROT_READ, MAP_PRIVATE, 4, 0)
    - mmap(0x4000, 0x6000, PROT_READ, MAP_PRIVATE, 4, 0)

IA-32 Support Status

- mmap calls (cont.)
  - *Bad* calls
  - Allocate and copy when needed
  - Less efficient than paging but it works
    - mmap(0x1000, 0x4000, PROT_READ, MAP_PRIVATE, 4, 0)
    - mmap(0x1000, 0x4000, PROT_READ, MAPP_SHARED, 4, 0)

IA-32 *bad* mmap call

mmap(0x6000, 0x8000, PROT_READ, MAP_SHARED | MAP_ANONYMOUS, -1, 0)



- Unmodified data
- mmaped data
- Saved data

IA-32 Support Status

- System calls (cont)
  - I/O Control (ioctl)
    - Not as bad as it seems
    - All calls have a unique identifier
      - ioctl(0, KDGETMODE, &l)
    - Shim code can translate each call
    - Only fails for private drivers
    - Solution is to add new shim code

IA-32 Support

- How can the open source community contribute?
  - Run your favorite IA-32 application
    - Report and/or fix any failures
  - Re-compile IA-32 applications for IA-64
    - Report and/or fix any failures

Summary

- The Trillian Project provided a solid start to the port of IA-64 Linux
- IA-64 Linux takes advantage of the new features of the IA-64 architecture
- IA-32 binaries run on IA-64 Linux
- Download the IA-64 Linux source code today!
  - Available at www.kernel.org