Meet FRED: Future Face of x86 Event Processing

Shivansh Dhiman, AMD India

AMD
together we advance_

Motivation

The way x86 handles interrupts, exceptions, and system calls is **~40 years old**, and it's showing its age.

- 🔒 **A security risk**: fragile steps like IRET led to real exploits (e.g., CVE-2014-9322).
- 🧩 **Hard to get right**: among the most bug-prone code in the kernel.
- ⚙️ **Complex & inconsistent**: extra memory lookups and a different stack layout per exception.

FRED is a ground-up redesign, by x86 Ecosystem Advisory Group: one unified, hardware-managed path for all events that is safer and simpler.

What are Events?

⌨️ You press a key → CPU stops to read it

⏱️ A timer fires → OS switches between tasks

➗ A program divides by zero → CPU flags an error

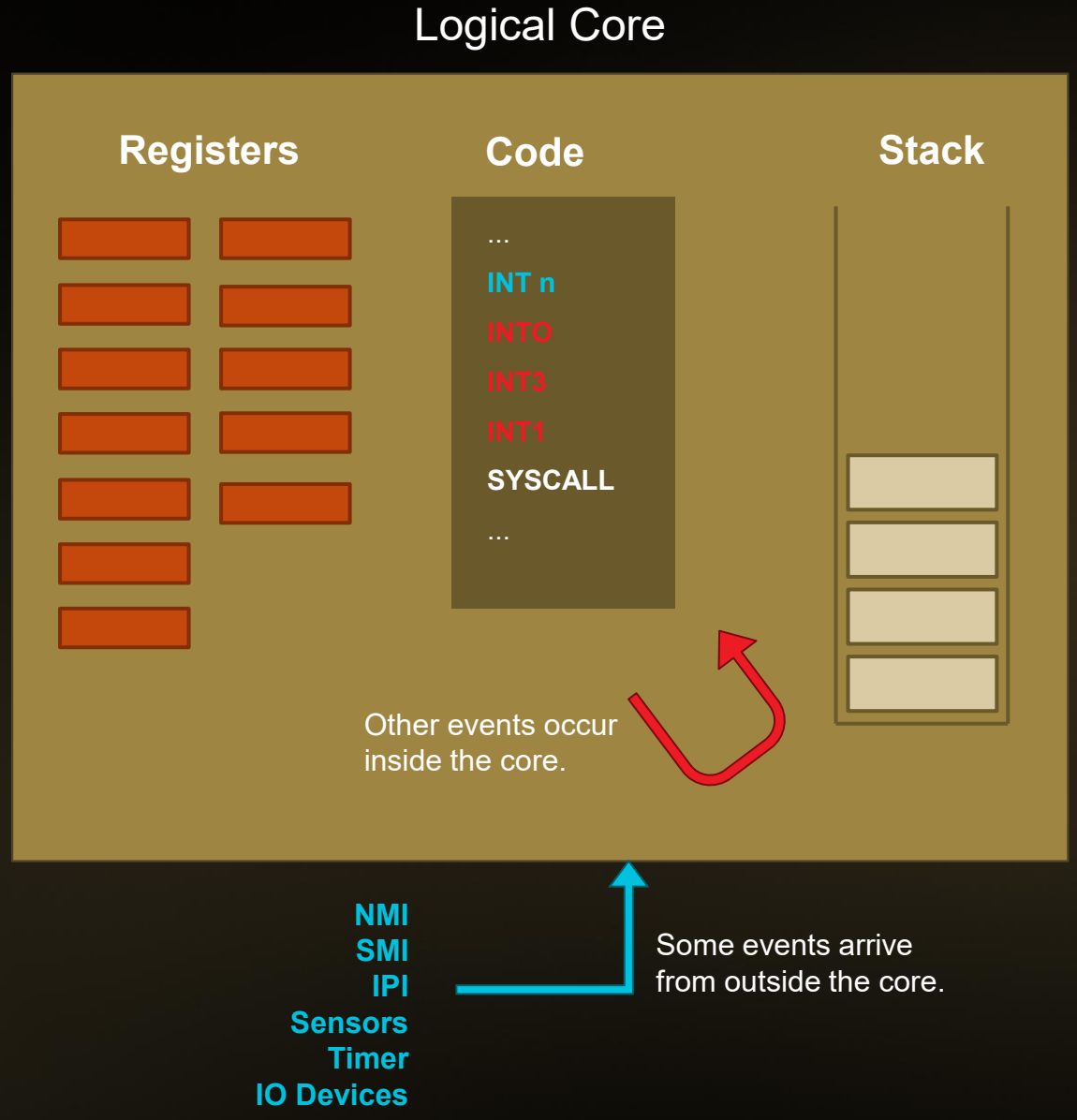
🌐 A network packet arrives → CPU stops to process it

📂 A program opens a file → CPU asks the OS for help

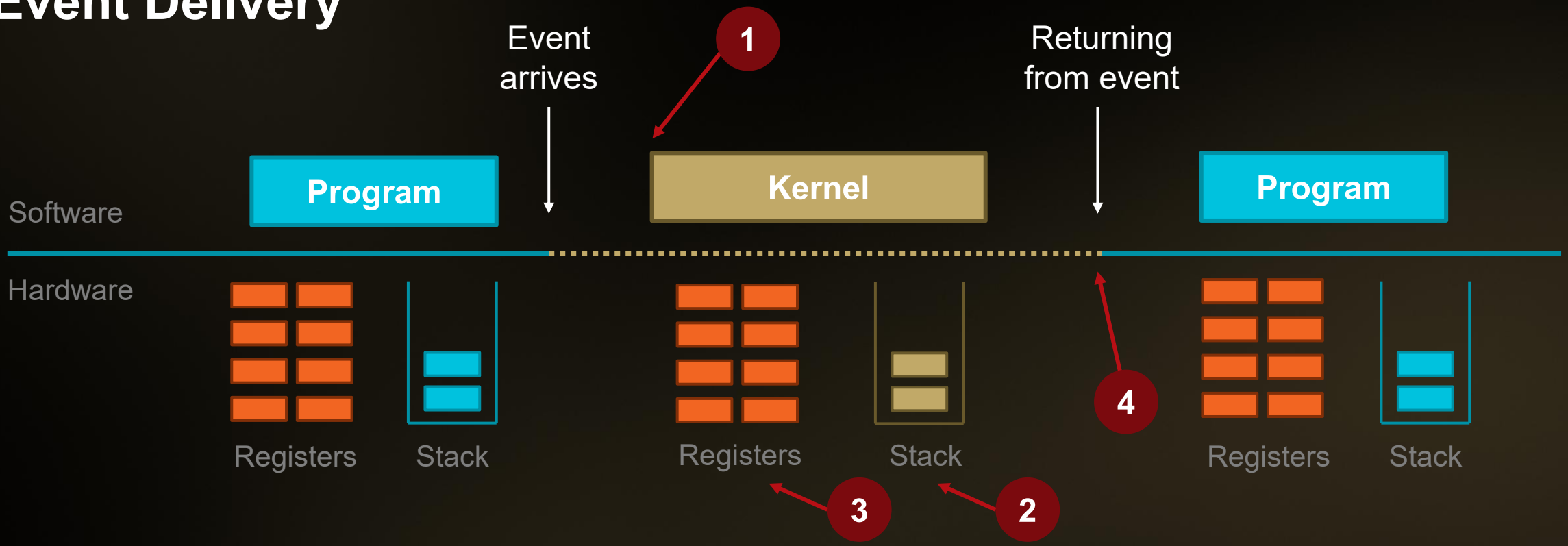
An event is any signal, from hardware, an error, or a program, that needs the CPU's immediate attention before it goes back to what it was doing.

Types of Events

- **Interrupts**
  - External event signalled to the CPU.
  - Generally initiated by the hardware, sometimes by software using INT N.
- **Exceptions**
  - Unexpected internal event when CPU executes an instruction.
  - Fault, trap or an abort.
- **System Calls**
  - A user program's deliberate request to the OS to perform a privileged operation.

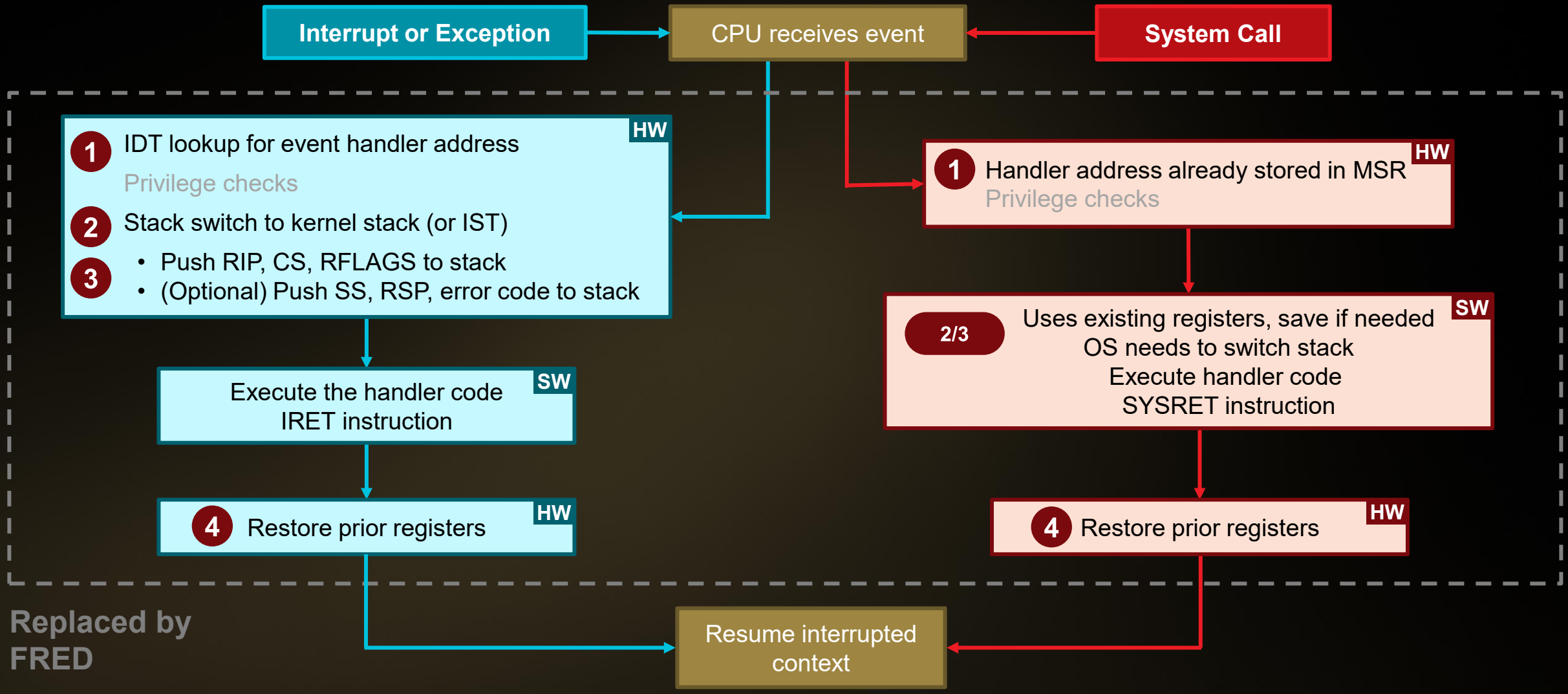


Event Delivery

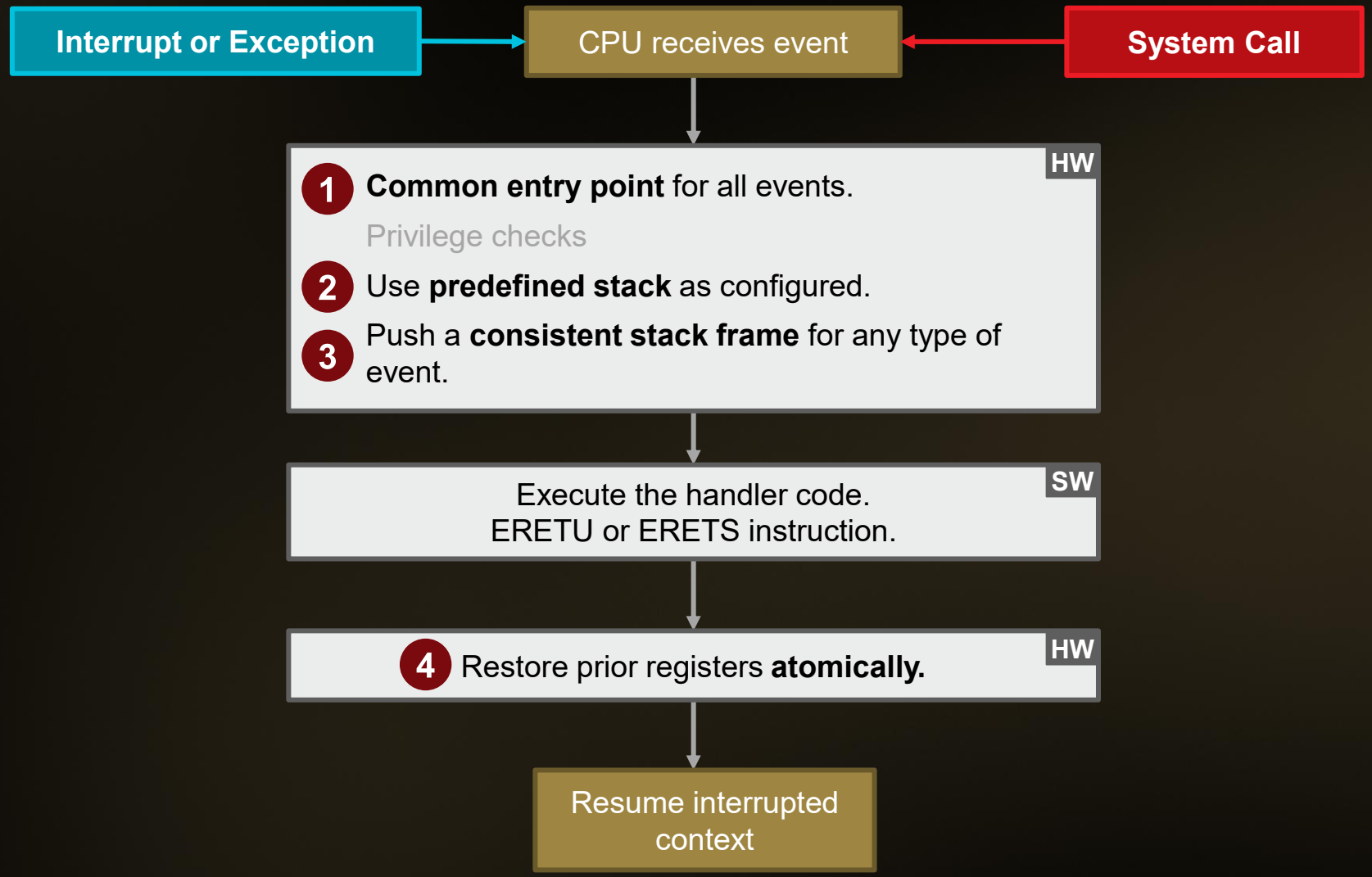


1. From which address to start kernel execution?
2. Which stack to use?
3. Kernel may need same CPU registers to run, so, where to save existing userspace registers?
4. How to return back to userspace program?

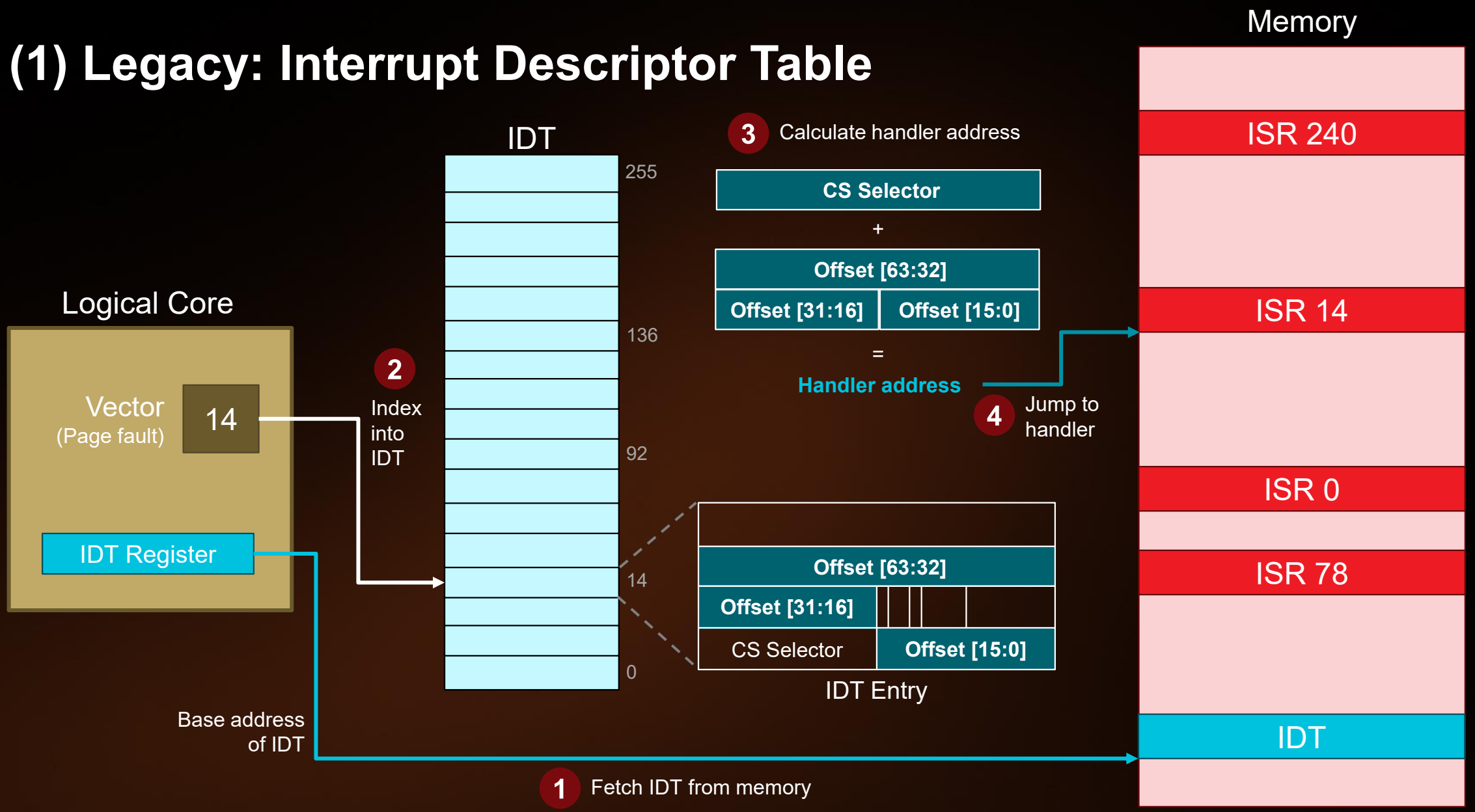
Legacy methods of Event Delivery



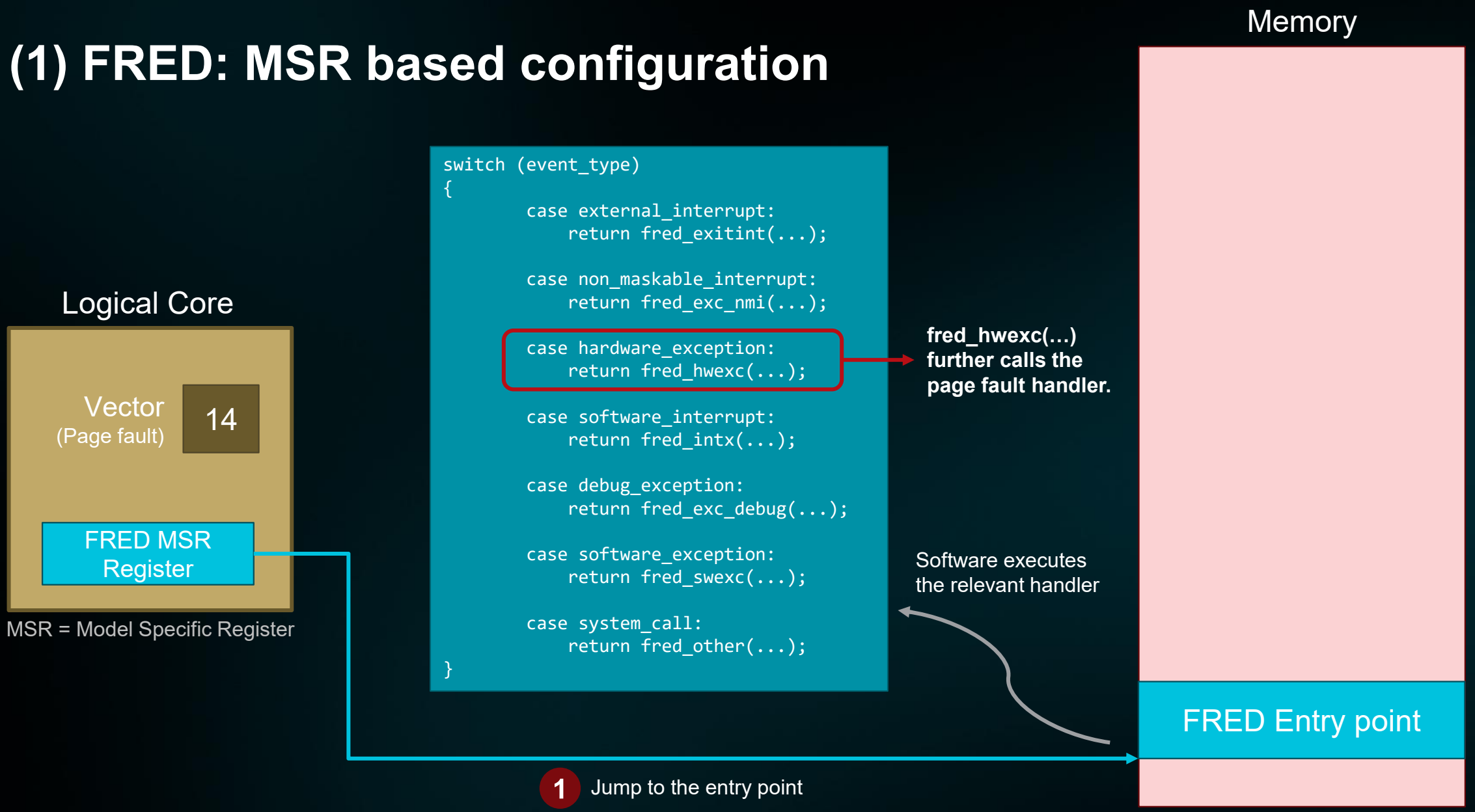
Overview of FRED



(1) Legacy: Interrupt Descriptor Table



(1) FRED: MSR based configuration



(2) Legacy: Several different exception stack layouts

For most exceptions:

Interrupt Stack
(different CPL)

0000	Old SS
Old ESP	
Old EFLAGS	
0000	Old CS
Old EIP	

Interrupt Stack
(same CPL)

Old EFLAGS	
0000	Old CS
Old EIP	

For certain exceptions, like #DF, #GP, #PF, #VC:

Interrupt Stack
(different CPL)

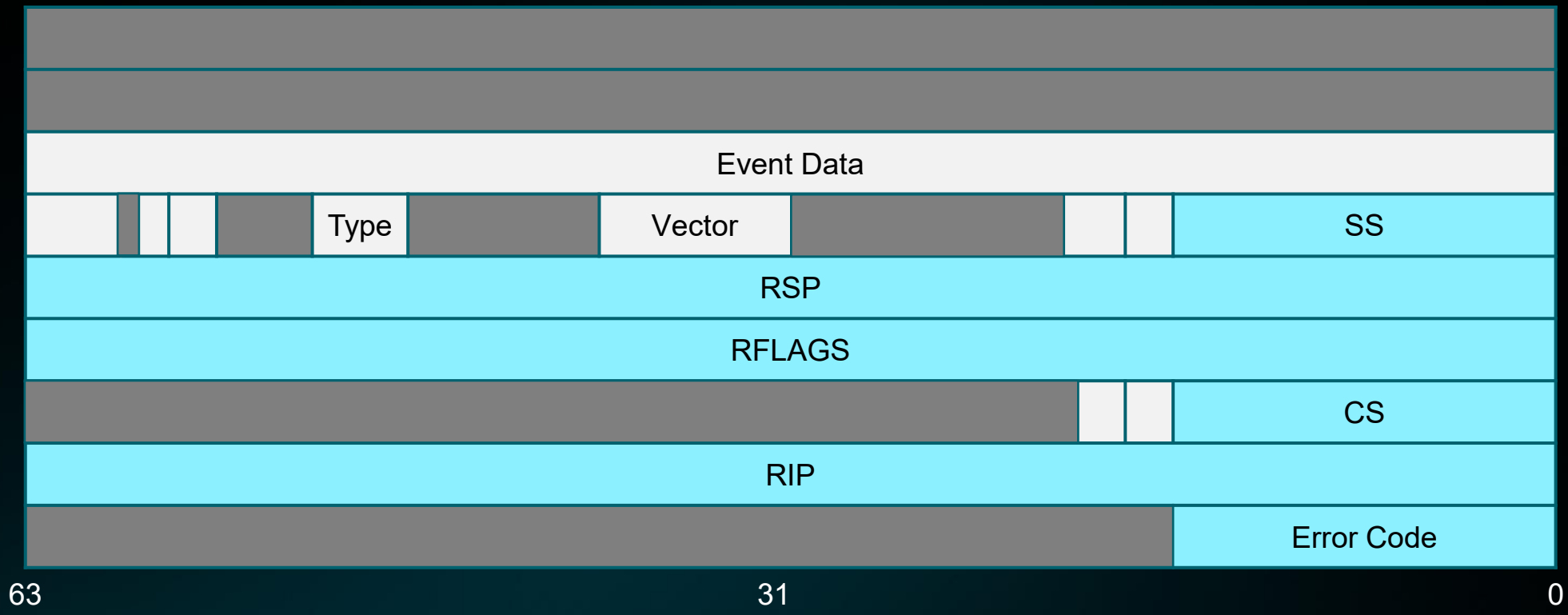
0000	Old SS
Old ESP	
Old EFLAGS	
0000	Old CS
Old EIP	
Error Code	

Interrupt Stack
(different CPL)

Old EFLAGS	
0000	Old CS
Old EIP	
Error Code	

CPL = Current Privilege Level
SS = Stack Segment
ESP = Stack Pointer
EFLAGS = Flags Register
CS = Code Segment
EIP = Instruction Pointer

(2) FRED: Consistent Stack Frame



- The same stack frame format is used for any event.
- Event Data: CR2 register value for #PF, debug data for #DB
- Type:
  - 0: External Interrupt
  - 2: Non-Maskable Interrupt
  - 3: Hardware exception
  - 4: INT n
  - 6: INT3 or INTO
  - 7: SYSCALL or SYSENTER

(3) Legacy: Several exception nesting problems

Case 1: NMI Nesting

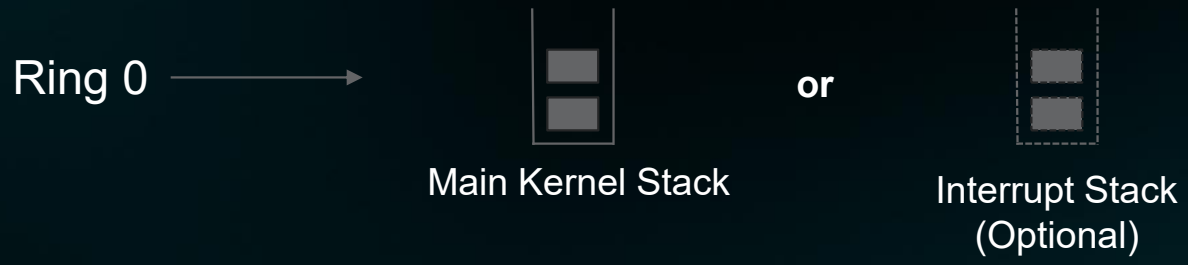
- NMI arrives
- CPU blocks further NMIs
- NMI handler starts executing
  - Page Fault occurs inside NMI handler
  - #PF handler runs and fixes the fault
  - #PF handler executes IRET, which unblocks NMI unconditionally.
- Back to NMI handler, but NMIs are now unblocked!
- Another NMI will make the NMI nest into itself, and corrupt the stack frame!

Case 2: MCE Nesting

- MCE arrives
- MCE handler starts executing using dedicated IST stack.
  - Another MCE occurs inside MCE handler
  - Which stack to use, since MCE is already using IST stack?
  - Uses the same IST stack, and overwrites the first MCE's stack frame!

(3) FRED: Stack Levels

- Legacy:
  - When an Interrupt arrives, the main kernel stack can be used or optionally a separate Interrupt stack is used.



- FRED:
  - 4 Stack Levels are defined.
  - When an Interrupt arrives, the stack to be used is pre-configured (using an MSR).

  - Stack Level 0: Routine exceptions and standard interrupts like page fault, divide by zero, etc.
  - Stack Level 1: More important interrupts like NMI.
  - Stack Level 2: High Priority interrupts like Machine Check.
  - Stack Level 3: Critical events like Double Fault.
  - **This is to ensure that important events get a clean stack to work with.**
  - **For any nested event, CPU sets a Nested Exception bit, and take measures to prevent stack corruption.**

(4) Legacy: In IRET, RSP not being restored properly

Ideal Case

Kernel Stack

User SS
User RSP
RFLAGS
User CS
User RIP

During IRET

- RIP popped
- CS popped
- RFLAGS popped
- RSP popped
- SS popped

The Bug

Kernel Stack

User SS
User RSP
RFLAGS
User CS
User RIP

During IRET

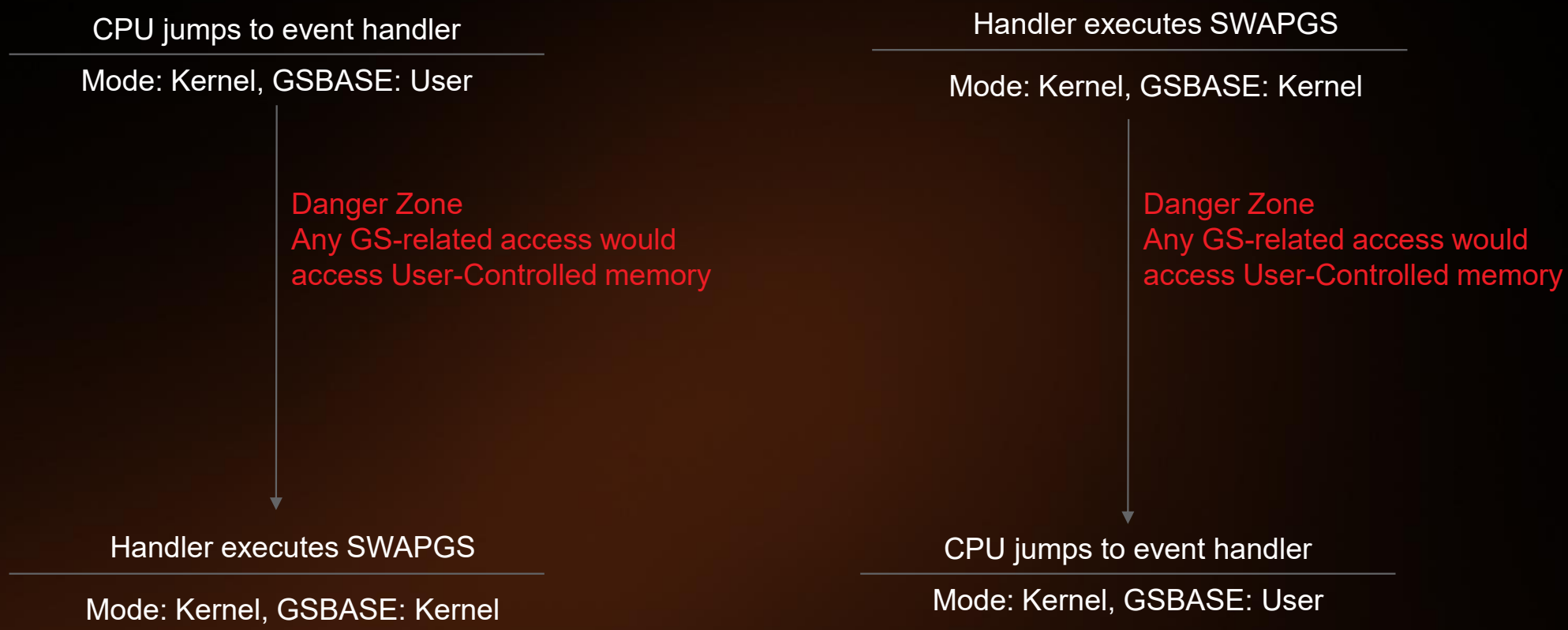
- RIP popped
- CS popped
- RFLAGS popped
- RSP popped
- Fault! #GP or #SS occurs

Cause:
IRET is Non-atomic

Vulnerability:
CVE-2014-9322

The CPU tries to deliver the event, but RSP is now User's RSP, not kernel stack.

(5) Legacy: Atomicity problems with SWAPGS



(4,5) FRED: New Return Instructions

	ERETU	ERETS
Full Name	Event Return to User	Event Return to Supervisor
Returns to	User mode (Ring 3)	Kernel mode (Ring 0)
GSBASE	Atomically swaps to user GSBASE	No swap (stays kernel GSBASE)
NMI unblocking	N/A (not returning from NMI to user)	Only if nested flag = 0 (true NMI return)
RSP restore	Atomic, all or nothing	Atomic, all or nothing

Virtualization

- Host machine must have FRED enabled for the guest to use it.
- Host kernel must support vFRED.
- Guest kernel must support FRED and is enabled.

Changes in VMCB Control Area

Fields' Summary
A bit that enables FRED Virtualization.
A nested exception bit in EXITINTINFO.
New field EXITINTDATA for Event Data.
New field EVENTINJDATA for injecting Events.

Changes in VMCB Save Area

Fields' Summary
Guest's FRED Stack Pointer registers RSP0-3
Guest's FRED Shadow Stack Pointer registers SSP0-3
Guest's FRED Entry point address
Guest's FRED Stack Levels configuration
Guest's Event Data fields

Current Status

- FRED support in Linux Kernel since v6.9.
- SVM Virtualization of FRED is on community mailing list.
  - Please study and review the patches.

References

- FRED Specification: https://www.intel.com/content/www/us/en/content-details/819481/flexible-return-and-event-delivery-fred-specification.html
- FRED Patches: https://lore.kernel.org/all/20231205105030.8698-1-xin3.li@intel.com/
- SVM vFRED Specification: https://docs.amd.com/v/u/en-US/69191-PUB
- SVM vFRED Patches: https://lore.kernel.org/kvm/20260402184240.1939480-1-shivansh.dhiman@amd.com/
- Video “FRED - ANVIN Peter, LI Xin”: (https://youtu.be/_PwBuDLBQw0?si=HNqY-o6CkPOm1_5y)

QnA

Copyright and disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION


AMD