Evaluating Linux Kernel Crash Dumping Mechanisms

2006/7/22

NTT Data Intellilink Corporation
Fernando Luis Vázquez Cao

Who am I ?

- LKDTT (Linux Kernel Dump Test Tool) maintainer
- MKDump (Mini Kernel Dump) co-maintainer

Agenda

1. Crash dumping
2. Crash dump testing: LKDTT
3. LKDTT example
4. Testing results
5. kdump shortcomings
6. Conclusion
7. Future lines of work

1. crash dumping

1.1. Kernel crash dump

- Snapshot of the system state at the time that the kernel crashed
  - Includes (at least): memory image, register contents
  - Used for post-crash analysis of the system

1.2. Kernel crash dumping process

- Multi-stage process
  1. Crash detection
  2. Minimal machine shutdown
  3. Crash dump capture

1.3. Crash dumping peculiarities

- Crash dumping mechanisms are not supposed to be used
  - The kernel is bug-free, right?
- Certain kernel bugs are difficult to reproduce
  - We have only one shot at it
- Crash dumping mechanisms have to be reliable
  - Bugs may reappear in a production system
  - Requirement in enterprise systems

2. crash dump testing

2.1. The need for a testing suite

- Evaluation of crash dump capturing mechanisms
  - Reliability: under which crash scenarios can a crash dump captured?
  - Crash dump integrity: do crash dump images reflect the state of the system at the time of the crash?
- Fair comparison of existing crash dumping solutions
  - Define and establish standard testing procedures

2.2. Traditional crash dump testing

- A testing module artificially causes the kernel to crash
  - Direct calls to panic and BUG, null pointer dereference, etc.
- Control of the tests through the /proc file system
  - Crash type to be induced is configurable
- Test results seem to contradict theory and reality
  - All crash dumping solutions seem to be very close in terms of reliability

2.2. Traditional crash dump testing (cont)

- Shortcomings
  - The coverage of the tests is limited in scope. Essential factors are left out of the picture:
    - ✗ HW conditions: DMA, interrupt and I/O rates, etc.
    - ✗ Execution context: interrupt, NMI, bottom half

2.3. LKDTT (Linux Kernel Dump Test Tool)

- LKDTT is a tool that forces the system to crash by artificially creating crash scenarios
  - Execution context can be precisely defined (through crash points)
  - Necessary HW and load conditions can be recreated (using auxiliary tools)
- Offers control over the testing process
  - Tests are configurable by means of the user-space tool (ttutils)

2.4. Crash Point (CP)

- In LKDTT the failures are injected at predefined *crash points*
  - Determines execution context at the time of the crash
- Implemented as kernel hooks
  - Disadvantage: need to patch and recompile the kernel
  - Does not alter the execution context
    - ✗ Dynamic probing (i.e. kprobes) may affect the test results since it uses breakpoints

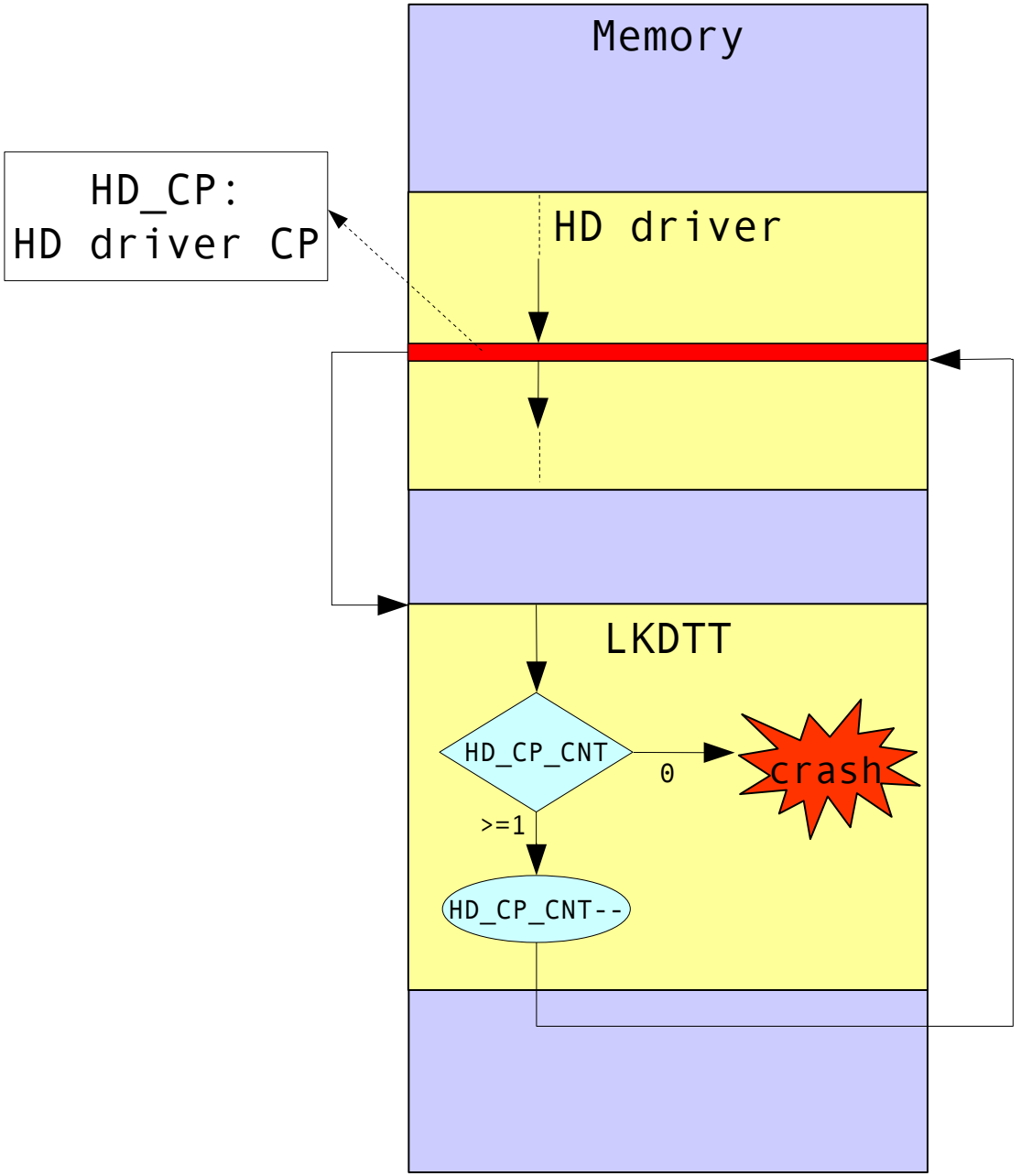
2.5. Crash point attributes

- CPs have two attributes that are configurable
  - Crash type: panic, oops, exception, stack overflow, lockup
  - Counter: number of times the crash point has to be hit before the kernel is forced to crash

2.6. Default crash points

- Interrupt handling with interrupts disabled
- Interrupt handling with interrupts enabled
- Tasklet execution with interrupts disabled
- Tasklet execution with interrupts enabled
- Swap-out
- Timer processing
- Block I/O
- SCSI command dispatching
- IDE command dispatching

2.7. Implementation



2.8. Auxiliary tools

- Artfully inserting a crash point does not always suffice to recreate certain crash scenarios
  - Some execution paths are rarely trodden
    - Kernel has to be lured to take the *right wrong way*
  - We might want HW to be in a particular state
    - Ongoing DMA and/or I/O, certain memory and CPU usage levels, etc.
- The *auxiliary tools* induce the necessary additional conditions

3. LKDTT example

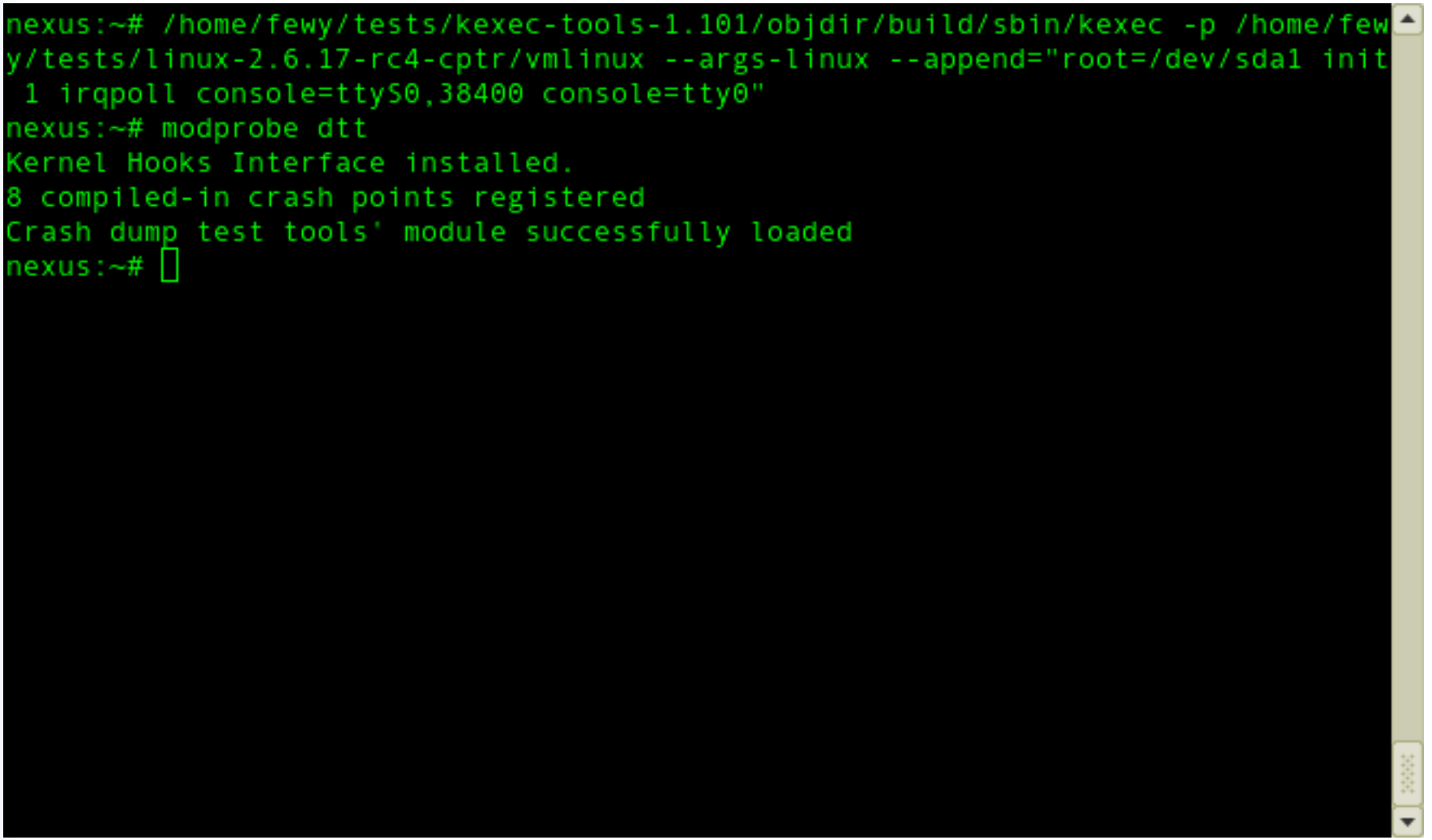
3.7. Testing procedure

1. Configure kdump
2. Load LKDTT module
3. Register a crash point (compiled-in crash points are registered automatically) ← ttutils
4. Configure the crash point ← ttutils
5. Recreate the desired testing conditions ← auxiliary tools
6. Make the kernel traverse the crash point ← auxiliary tools

3.1. Test example – kdump configuration

```
nexus:~# /home/fewy/tests/kexec-tools-1.101/objdir/build/sbin/kexec -p /home/few
y/tests/linux-2.6.17-rc4-cptr/vmlinux --args-linux --append="root=/dev/sda1 init
 1 irqpoll console=ttyS0,38400 console=tty0"
nexus:~#
```

3.2. Test example – LKDTT module loading



3.3. Test example – crash point listing

```
nexus:~# /home/fewy/tests/kexec-tools-1.101/objdir/build/sbin/kexec -p /home/few
y/tests/linux-2.6.17-rc4-cptr/vmlinux --args-linux --append="root=/dev/sda1 init
 1 irqpoll console=ttyS0,38400 console=tty0"
nexus:~# modprobe dtt
Kernel Hooks Interface installed.
8 compiled-in crash points registered
Crash dump test tools' module successfully loaded
nexus:~# ttutils ls
id      crash type      crash point name        count   location
1       none            INT_HARDWARE_ENTRY      0       kern
2       none            INT_TASKLET_ENTRY       0       kern
3       none            FS_DEVRW                0       kern
4       none            MEM_SWAPOUT             0       kern
5       none            TASKLET                 0       kern
6       none            TIMERADD                0       kern
8       none            INT_HW_IRQ_EN           0       kern
101     none            SCSI_DISPATCH_CMD       0       kern
nexus:~#
```

3.4. Test example – crash point configuration

```
nexus:~# /home/fewy/tests/kexec-tools-1.101/objdir/build/sbin/kexec -p /home/few
y/tests/linux-2.6.17-rc4-cptr/vmlinux --args-linux --append="root=/dev/sda1 init
 1 irqpoll console=ttyS0,38400 console=tty0"
nexus:~# modprobe dtt
Kernel Hooks Interface installed.
8 compiled-in crash points registered
Crash dump test tools' module successfully loaded
nexus:~# ttutils ls
id      crash type      crash point name        count   location
1       none            INT_HARDWARE_ENTRY      0       kern
2       none            INT_TASKLET_ENTRY       0       kern
3       none            FS_DEVRW                0       kern
4       none            MEM_SWAPOUT             0       kern
5       none            TASKLET                 0       kern
6       none            TIMERADD                0       kern
8       none            INT_HW_IRQ_EN           0       kern
101     none            SCSI_DISPATCH_CMD       0       kern
nexus:~# ttutils set -p MEM_SWAPOUT -t panic -c 15
[crash point set] name:MEM_SWAPOUT type:panic count:15
nexus:~#
```

3.5. Test example – auxiliary tool execution

```
nexus:~# /home/fewy/tests/kexec-tools-1.101/objdir/build/sbin/kexec -p /home/few
y/tests/linux-2.6.17-rc4-cptr/vmlinux --args-linux --append="root=/dev/sda1 init
 1 irqpoll console=ttyS0,38400 console=tty0"
nexus:~# modprobe dtt
Kernel Hooks Interface installed.
8 compiled-in crash points registered
Crash dump test tools' module successfully loaded
nexus:~# ttutils ls
id      crash type      crash point name        count   location
1       none            INT_HARDWARE_ENTRY      0       kern
2       none            INT_TASKLET_ENTRY       0       kern
3       none            FS_DEVRW                0       kern
4       none            MEM_SWAPOUT             0       kern
5       none            TASKLET                 0       kern
6       none            TIMERADD                0       kern
8       none            INT_HW_IRQ_EN           0       kern
101     none            SCSI_DISPATCH_CMD       0       kern
nexus:~# ttutils set -p MEM_SWAPOUT -t panic -c 15
[crash point set] name:MEM_SWAPOUT type:panic count:15
nexus:~# ./bin/brk
```

3.6. Test example – here we go!

```
nexus:~# /home/fewy/tests/kexec-tools-1.101/objdir/build/sbin/kexec -p /home/few
y/tests/linux-2.6.17-rc4-cptr/vmlinux --args-linux --append="root=/dev/sda1 init
 1 irqpoll console=ttyS0,38400 console=tty0"
nexus:~# modprobe dtt
Kernel Hooks Interface installed.
8 compiled-in crash points registered
Crash dump test tools' module successfully loaded
nexus:~# ttutils ls
id      crash type      crash point name        count   location
1       none            INT_HARDWARE_ENTRY      0       kern
2       none            INT_TASKLET_ENTRY       0       kern
3       none            FS_DEVRW                0       kern
4       none            MEM_SWAPOUT             0       kern
5       none            TASKLET                 0       kern
6       none            TIMERADD                0       kern
8       none            INT_HW_IRQ_EN           0       kern
101     none            SCSI_DISPATCH_CMD       0       kern
nexus:~# ttutils set -p MEM_SWAPOUT -t panic -c 15
[crash point set] name:MEM_SWAPOUT type:panic count:15
nexus:~# ./bin/brk
Kernel panic - not syncing: dumptest
 Linux version 2.6.17-rc4-cptr (fewy@nexus) (gcc version 4.0.4 20060507 (prerele
ase) (Debian 4.0.3-3)) #1 PREEMPT Tue May 16 15:06:50 JST 2006
BIOS-provided physical RAM map:
```

4. testing results

4.1. Results: LKCD vs kdump

Crash point	Trigger	LKCD 6.1.0_2.6.9 preemptive	kdump 2.6.13-rc7 preemptive
IRQ handling with IRQs disabled	panic	×	○
	oops	×	○
	exception	×	○
	lockup	×	×
	stack overflow (>STACK_WARN)	×	×
SCSI command dispatching	panic	×	○
	oops	×	○
	exception	×	○
	lockup	×	×
	stack overflow (>STACK_WARN)	×	×
Paging	panic	○	○
	oops	○	○
	exception	○	○
	lockup	×	×
	stack overflow (>STACK_WARN)	×	×

http://lkdtt.sourceforge.net/

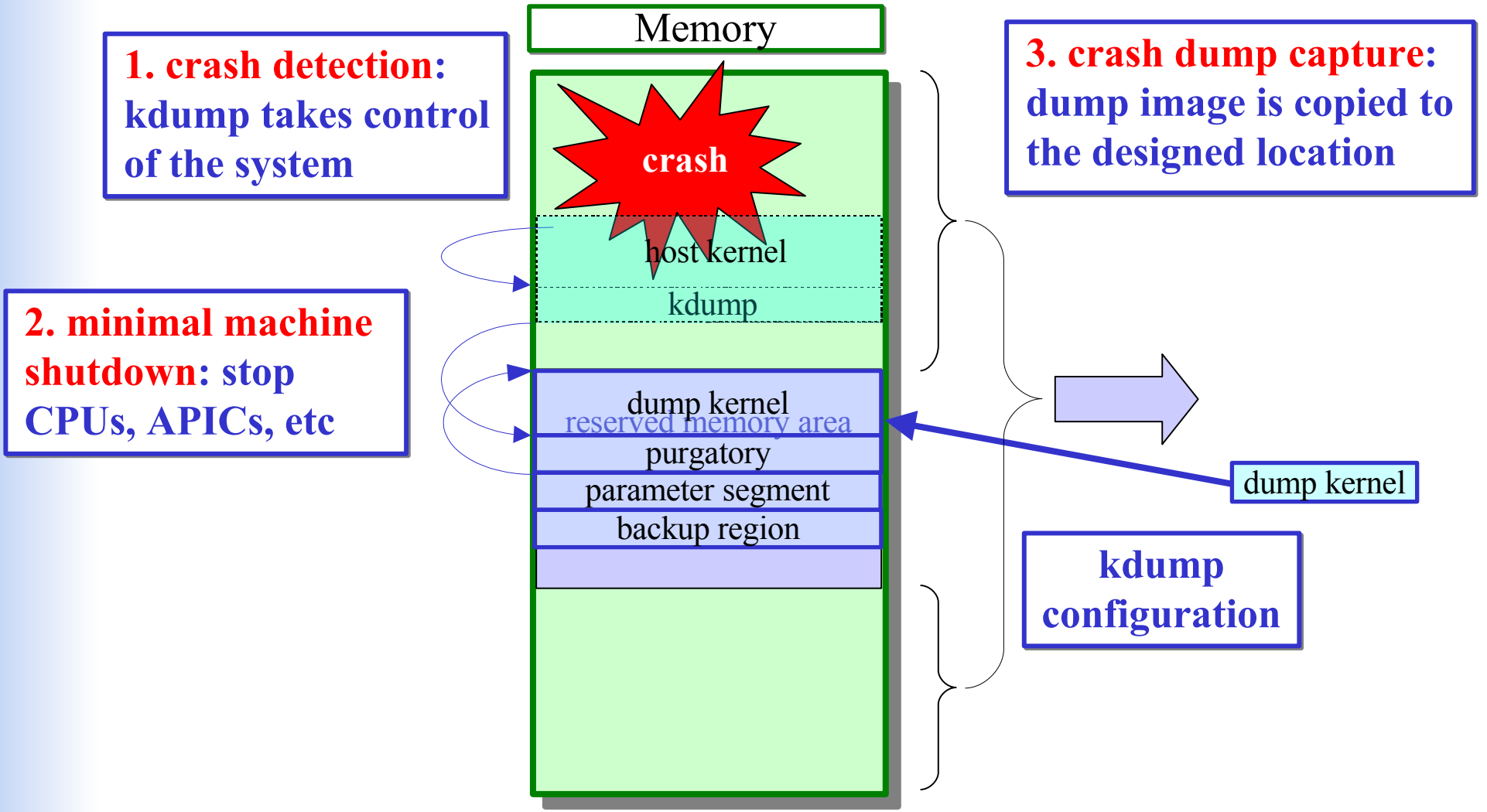
4.2. Results: kdump

Crash point	Type	kdump 2.6.13-rc7 preemptive	kdump 2.6.16 preemptive	kdump 2.6.17-rc4 preemptive	kdump 2.6.17-rc4 non-preemptive
Tasklet with IRQs enabled	panic	○	○	○	○
	oops	○	○	○	○
	exception	○	○	×3	○
	lockup	×1	×1	×1	×1
	stack overflow (>STACK_WARN)	×	×	×	×
Block I/O	panic	○	×3	○	○
	oops	○	×1	×1	×1
	exception	○	×1	×1	×1
	lockup	×1	×1	×1	×1
	stack overflow (>STACK_WARN)	×	×	×	×
Swap-out	panic	○	○	○	○
	oops	○1	○1	○1	○1
	exception	○1	○1	○1	○1
	lockup	×1	×1	×1	×1
	stack overflow (>STACK_WARN)	×	×	×	×

×# / ○# : stage of crash dump process in which there was an issue

5. kdump shortcomings

5. kexec-based crash dumping



5. Kdump issues

- Problems in all three stages of the dump process

1. Crash detection: lockups (not detectable by NMI watchdog or soft lockup detector), spontaneous reboots, recursive page faults
2. Minimal machine shutdown: spontaneous reboots, recursive page faults
3. Crash dump capture: failures initializing devices in the dump capture kernel

5.1. Crash detection – lockups

- Lockups with interrupts enabled are not always detectable (not detectable by current soft-lockup detector)
  - Tasks in TASK_UNINTERRUPTIBLE state may leave the system unresponsive
- Solution proposal: detector of uninterruptible tasks
  - Trigger crash dump process when certain tasks have been stuck in TASK_UNINTERRUPTIBLE state for a long time

5.1. Crash detection – reboots

- *Triple faults* can cause spontaneous reboots
  - A frequent cause of triple faults is a kernel stack overflow
- Problem alleviation: stack overflow detection
  - Current mechanism checks the free stack space whenever an interrupt occurs
    - ✗ Stack might be bloated before the next check
  - Use instant detection instead
    - ✗ Stack guard page: use small pages and add unmapped page at the top of the stack

5.2. Minimal machine shutdown

- Reboot path to the dump kernel is not safe
  - Structures likely to be stomped in the event of a stack overflow (such as thread_info) are used
  - Critical parts of the kernel (page fault handler, double fault handler, NMI handler) make assumptions about the validity of the stack
    - ✗ Possible consequences: recursive page faults, triple faults, incorrect information in crash dumps, etc

5.2. Minimal machine shutdown (cont)

- Hardening reboot path to dump kernel
  - Use stack overflow-safe functions (merged)
  - Switch to a new dedicated stack as soon as the crash dump path is entered
  - Use special NMI handler to stop CPUs in *minimal machine shutdown* stage

5.3. Crash dump capture

- Device reinitialization in the dump kernel
  - No device shutdown in the crashing kernel
  - Firmware stages of the boot process are skipped
  - Device drivers assume that the BIOS did all the dirty work
    - ✗ Drivers find devices in an unexpected state or receive a message from the previous kernel's context and fail
- Soft-boot case not handled by drivers
  - Need to leverage device driver subsystem?

6. conclusion

6.1. Conclusion – testing

- Crash dump mechanisms are seldom used
  - When invoked we cannot afford a failure
    - ✗ We might not be able to reproduce the problem
    - ✗ Problem might reappear in a production system

6.1. Conclusion – testing (cont)

- Test cases should be realistic and precise: LKDTT
  - Definition of the execution context
  - Recreation of HW and load conditions
  - Configurability and controllability
- Without regular testing regressions pass unnoticed
- Need to be paranoid and anticipate possible problems!
- DTT revealed shortcomings in kdump not detectable by traditional testing methods

7. future lines of work

7.1. Future lines of work

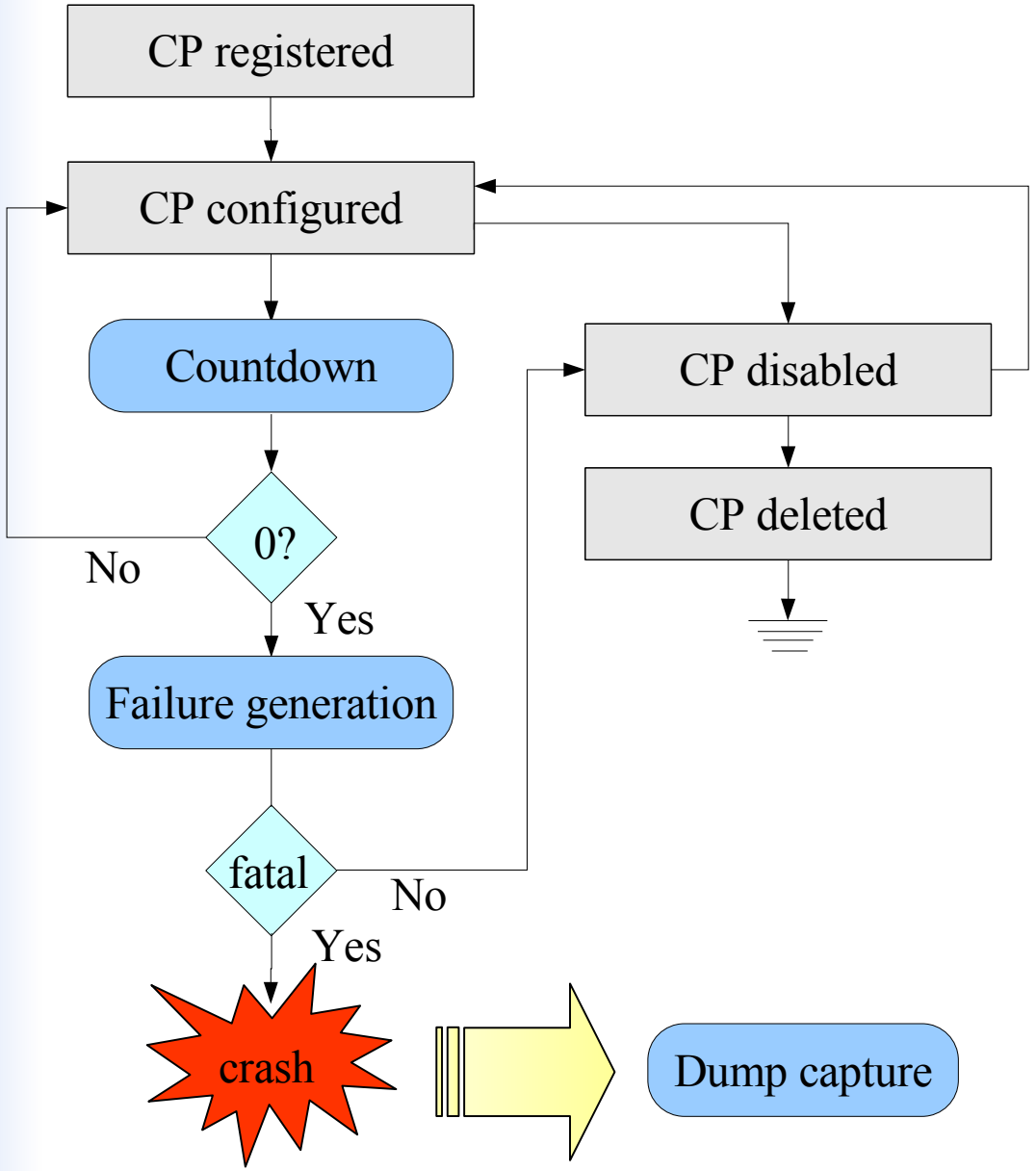
- LKDTT
  - Test automation
  - Creation of test cases for the device reinitialization problems after a soft-boot, and kdump in particular

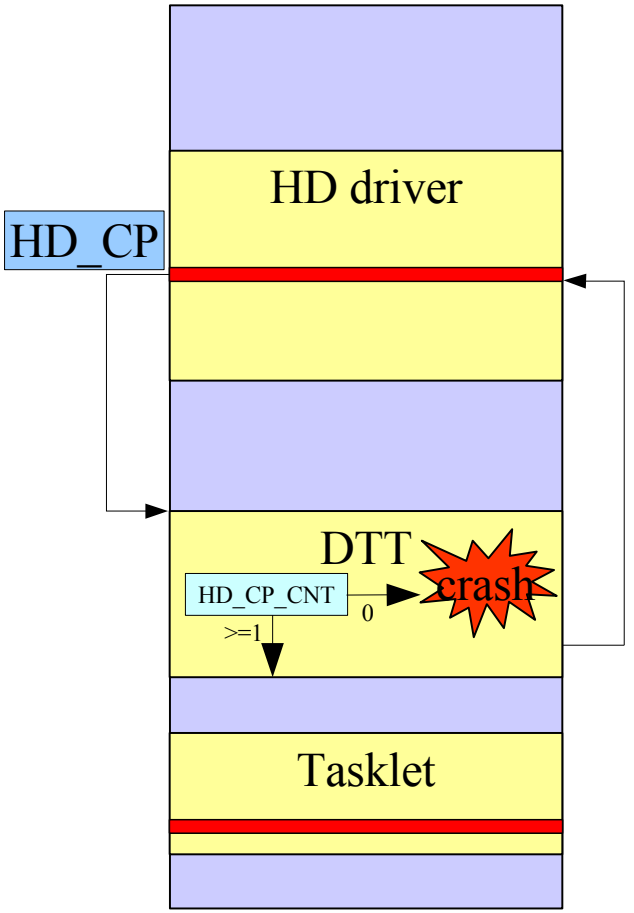
Thanks for your attention

Contact: fernando@intellilink.co.jp

LKDTT: http://lkdtt.sourceforge.net/

2.7. Implementation





4.1. Results (I)

Crash point	Type	LKCD	kdump 2.6.13-rc7 preemptive	kdump 2.6.16 preemptive	kdump 2.6.17-rc4 preemptive	kdump 2.6.17-rc4 non-preemptive
IRQ handling with IRQs disabled	panic	×	○	○	○	○
	oops	×	○	○	○	○
	exception	×	○	○	×	○
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×
IRQ handling with IRQs enabled	panic	×	○	×	○	○
	oops	×	○	×	×	×
	exception	×	○	×	×	×
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×
Tasklet with IRQs disabled	panic	○	○	○	○	○
	oops	○	○	○	×	×
	exception	○	○	○	×	×
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×

4.1. Results (II)

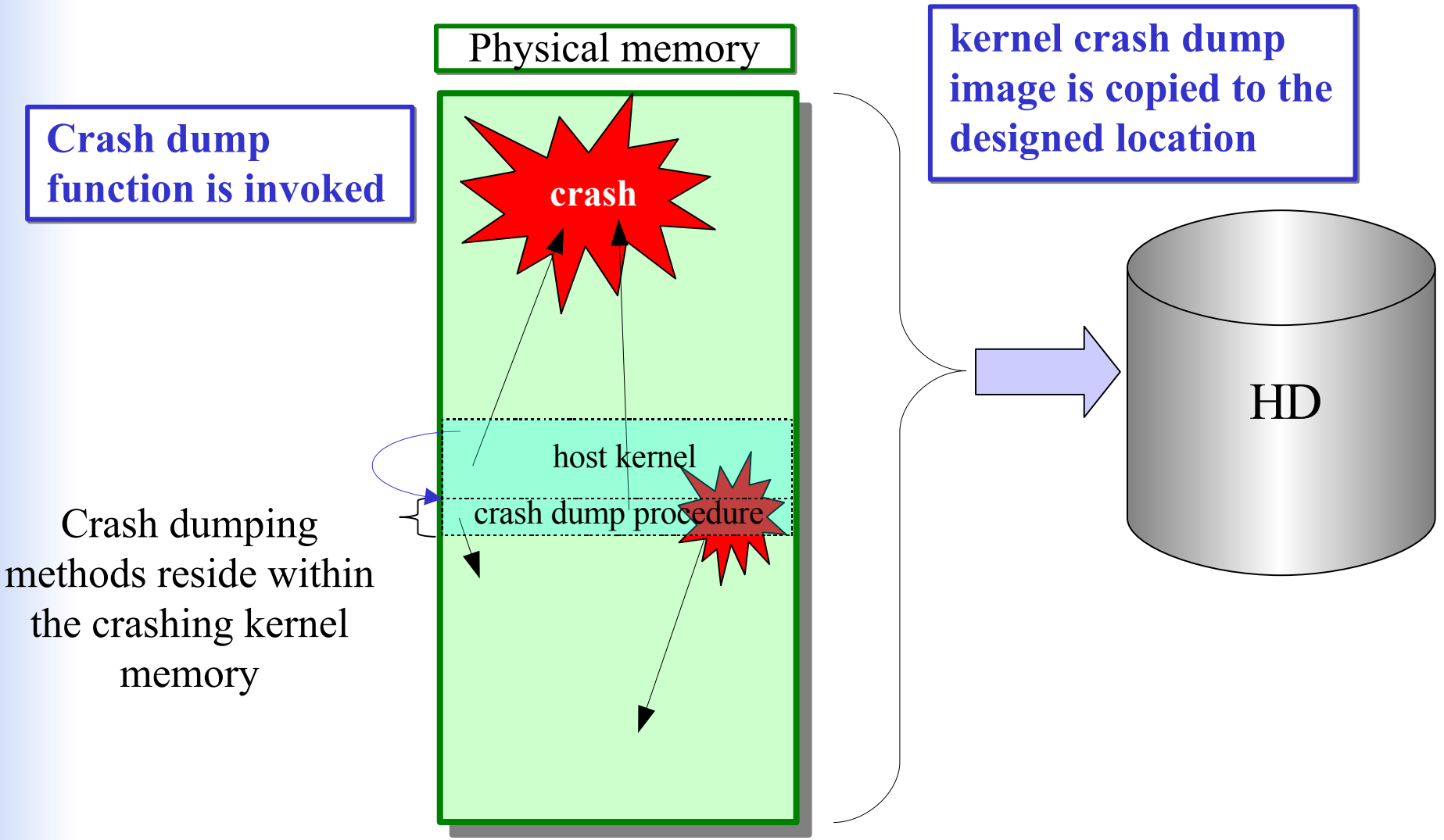
Crash point	Type	LKCD	kdump 2.6.13-rc7 preemptive	kdump 2.6.16 preemptive	kdump 2.6.17-rc4 preemptive	kdump 2.6.17-rc4 non-preemptive
Tasklet with IRQs enabled	panic	○	○	○	○	○
	oops	○	○	○	○	○
	exception	○	○	○	×	○
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×
Block I/O	panic	○	○	×	○	○
	oops	○	○	×	×	×
	exception	○	○	×	×	×
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×
Swap-out	panic	○	○	○	○	○
	oops	○	○	○	×	×
	exception	○	○	○	×	×
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×

4.1. Results (III)

Crash point	Type	LKCD	kdump 2.6.13-rc7 preemptive	kdump 2.6.16 preemptive	kdump 2.6.17-rc4 preemptive	kdump 2.6.17-rc4 non-preemptive
Timer processing	panic	○	○	○	○	○
	oops	○	○	○	○	○
	exception	○	○	○	×	○
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×
SCSI command	panic	×	○	×	○	○
	oops	×	○	×	×	×
	exception	×	○	×	×	×
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×
IDE command	panic	×	○	○	○	○
	oops	○	○	○	×	×
	exception	×	○	○	×	×
	lockup	×	×	×	×	×
	stack overflow	×	×	×	×	×

4. in-kernel crash dump issues

4.1. In-kernel crash dumping



4.2. In-kernel crash dumping issues

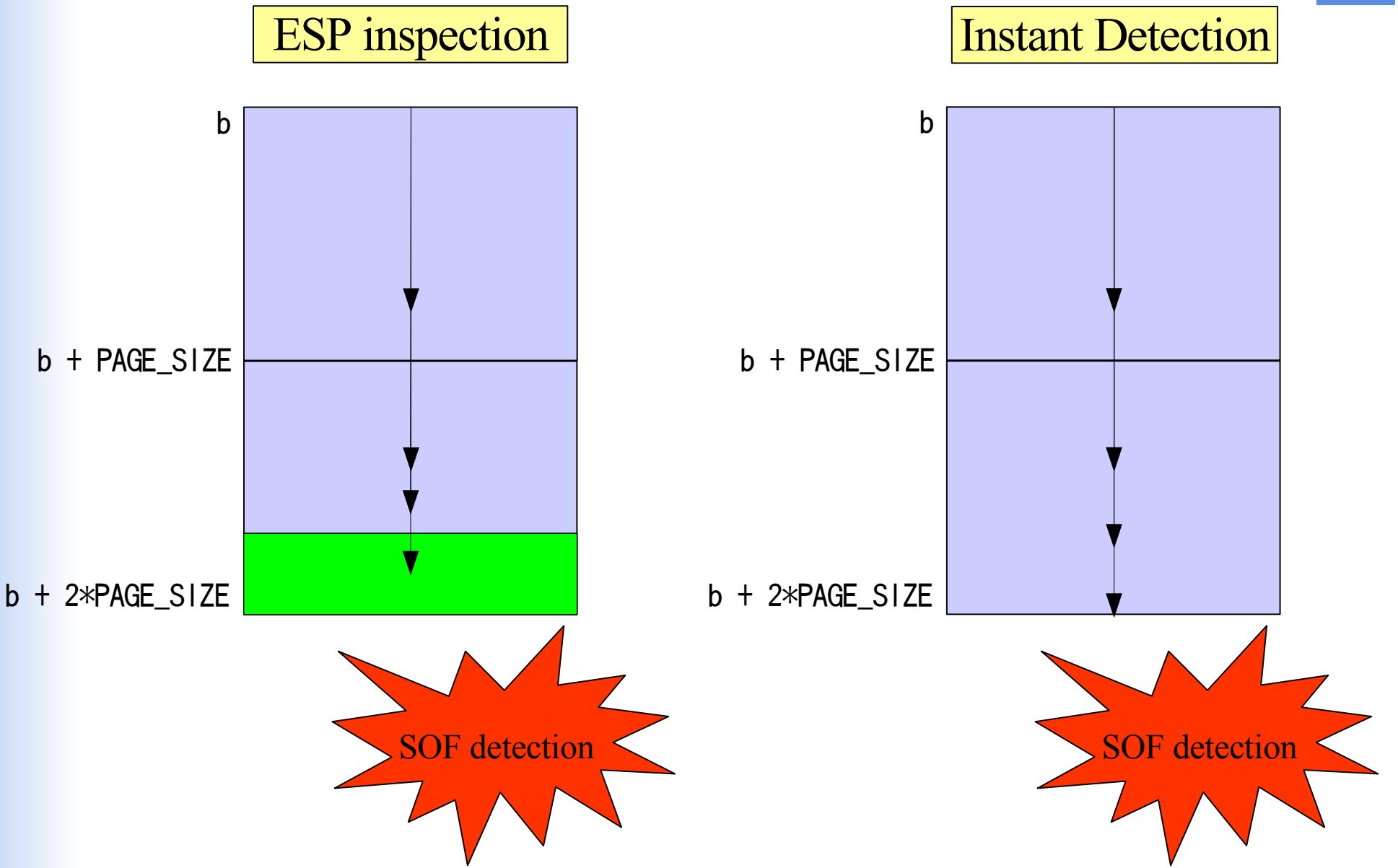
- Flawed assumption that the kernel can be trusted and will be operating in a normal fashion
  - Resource lock-up
    - ✗ Locking status at the time of the crash is unknown
    - ✗ Drivers and services of the crashing kernel are used to capture the dump
    - ✗ Attempts to obtain a lock that was held before the crash will cause a deadlock
    - ✗ Can be alleviated by using polling mode to communicate with the dump device

4.2. In-kernel crash dumping issues (cont)

- Damaged control structures
  - ✗ Unpredictable behavior
- Corrupted page tables
  - ✗ Invalid page faults (double faults, triple faults, etc)
- Corrupted registers (stack pointer, etc)
  - ✗ Invalid faults (page faults, double faults, invalid TSS, etc)

- Setting up a controllable dump route within the kernel is very difficult
  - Sparked the apparition of soft boot inspired solutions

5.2.1. Stack overflow detection



5.5. Dump kernel – APICs

- Current APICs partial reinitialization code assumes they work properly
  - Problems with broken BIOSes and old systems: the system stops receiving timer interrupts
- Restore APICs to its original status (i.e. as configured by the BIOS)
  - Properly reinitializes the APICs even in machines with a broken BIOS a
  - Requires relocation to BSP
    - Can do SMP (on i386, x86_64) b
    - Inter-CPU NMIs used for relocation ignored in some machines (c)
  - Trade-off between (a,b) and (c)

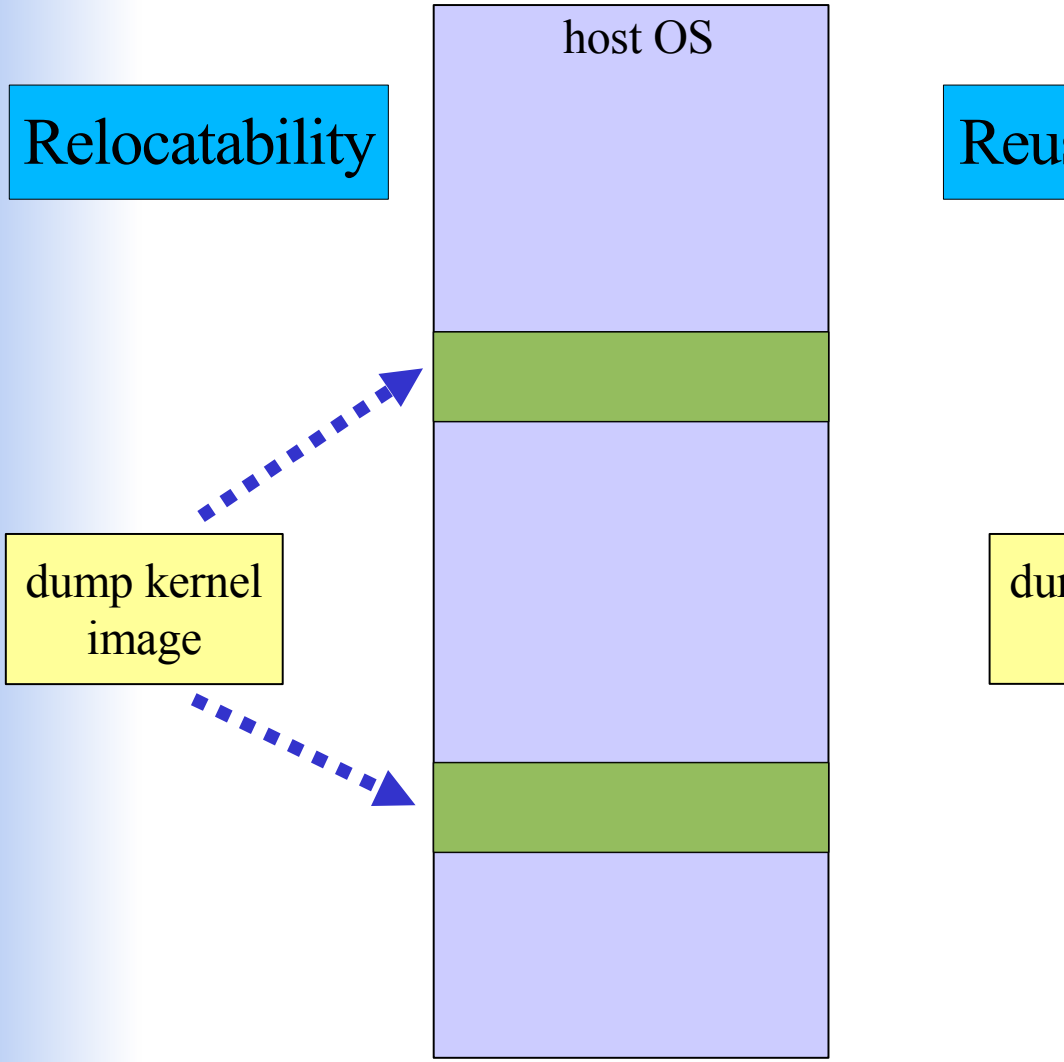
6.3. Conclusion: results

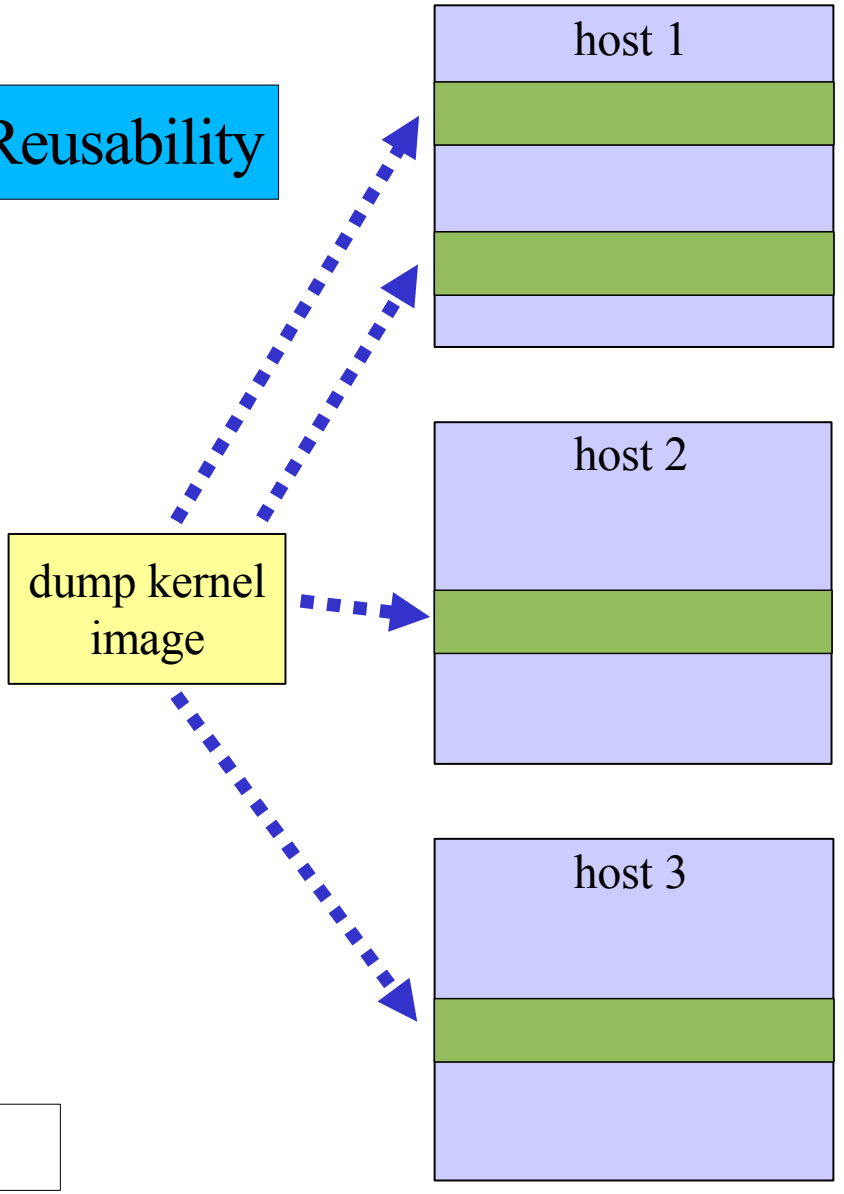
- Dependency ties with the crashed kernel must be avoided
  - Do not use crashing kernel's resources
  - The dump capture kernel must run from a dedicated and protected memory area an use its own resources (memory, drivers, etc)
- Proper crash detection mechanisms are indispensable
  - To limit the havoc caused by crashes
  - To trigger the crash dumping process

7.2. Run-time relocatable kernel

- Run-time relocatable kernel
  - No hard-coding of kernel's start address
    - Simplify installation automation: no need to build a set of dump kernels with different start addresses
    - Avoid potential overlap with ACPI tables
  - Mkdump implementation
    - Use *--emit-relocs* to leave relocation information in *arch/i386/boot/compressed/vmlinux.bin*
    - Pre-process relocation tables and fix relocations of absolute symbols (*reltab*)
    - Resolve addresses in *startup_32* (*kernel/head.S*)
  - Eric's suggestion
    - Use *-shared -Bsymbolic* and *vmlinux* instead

7.2. Run-time relocatable kernel (cont)





reserved memory area