Reinitialization of devices after a kexec reboot

2006/7/21

NTT Data Intellilink Corporation
Fernando Luis Vázquez Cao

Purpose of this BOF

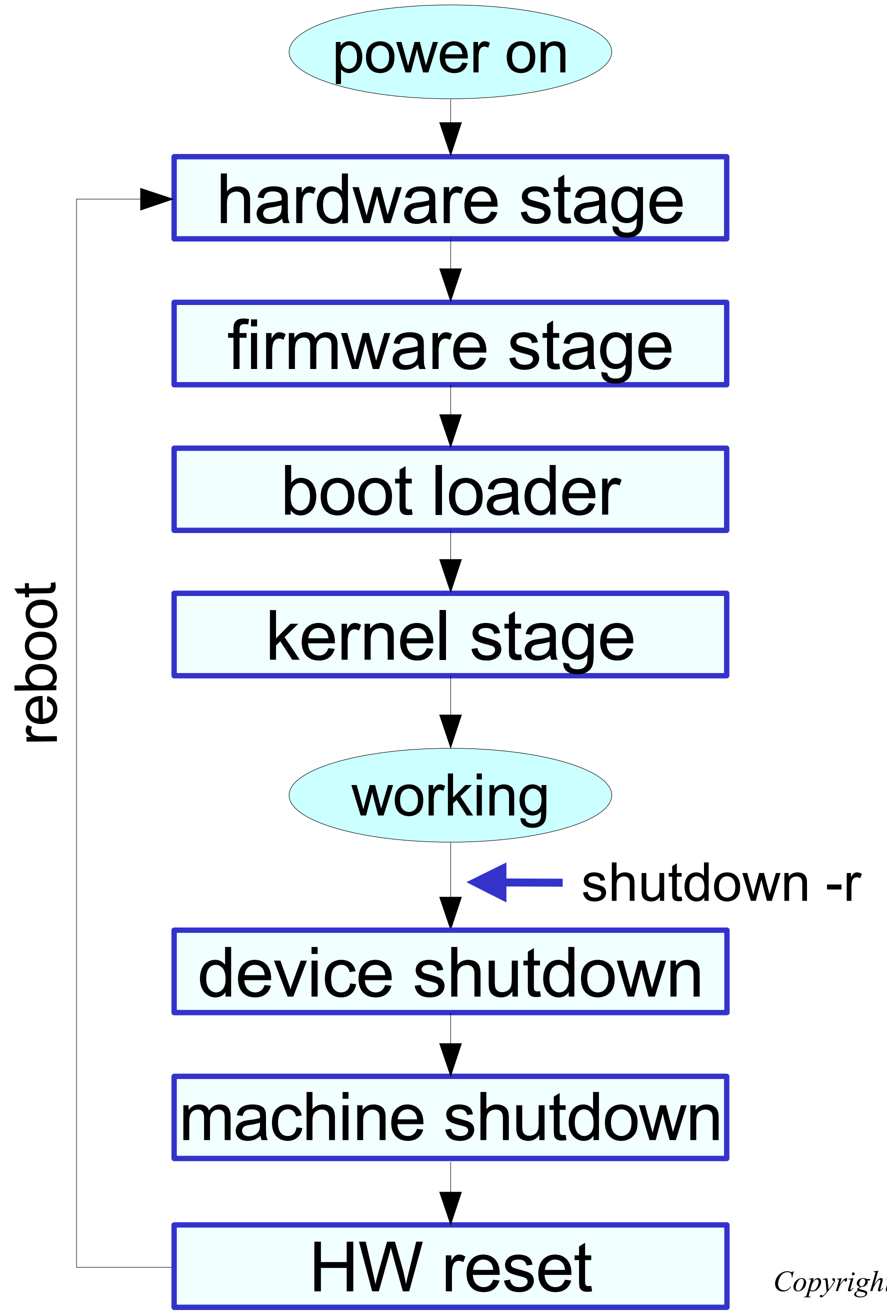
- Discuss main inhibitor to kexec/kdump adoption: reinitialization of devices in the second kernel
- Present possible approaches to solve the device reinitialization problem
- Propose a solution
- Reach a consensus

Agenda

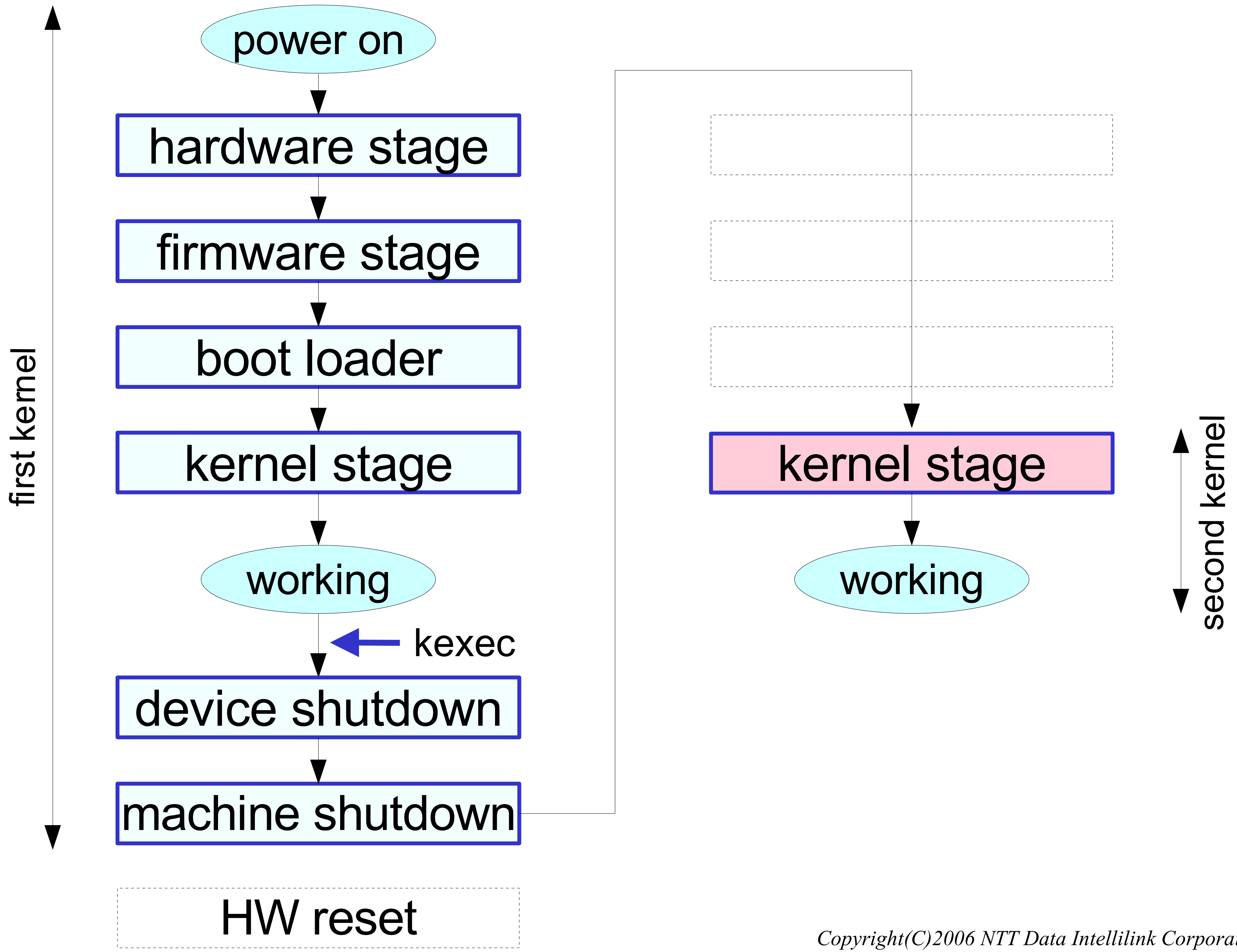
1. Kexec/kdump reboot
2. Device reinitialization
3. Tackling device reinitialization
    - Device black list
    - Device / bus reset
    - Device hardening
4. Solution proposal

1．kexec/kdump reboot

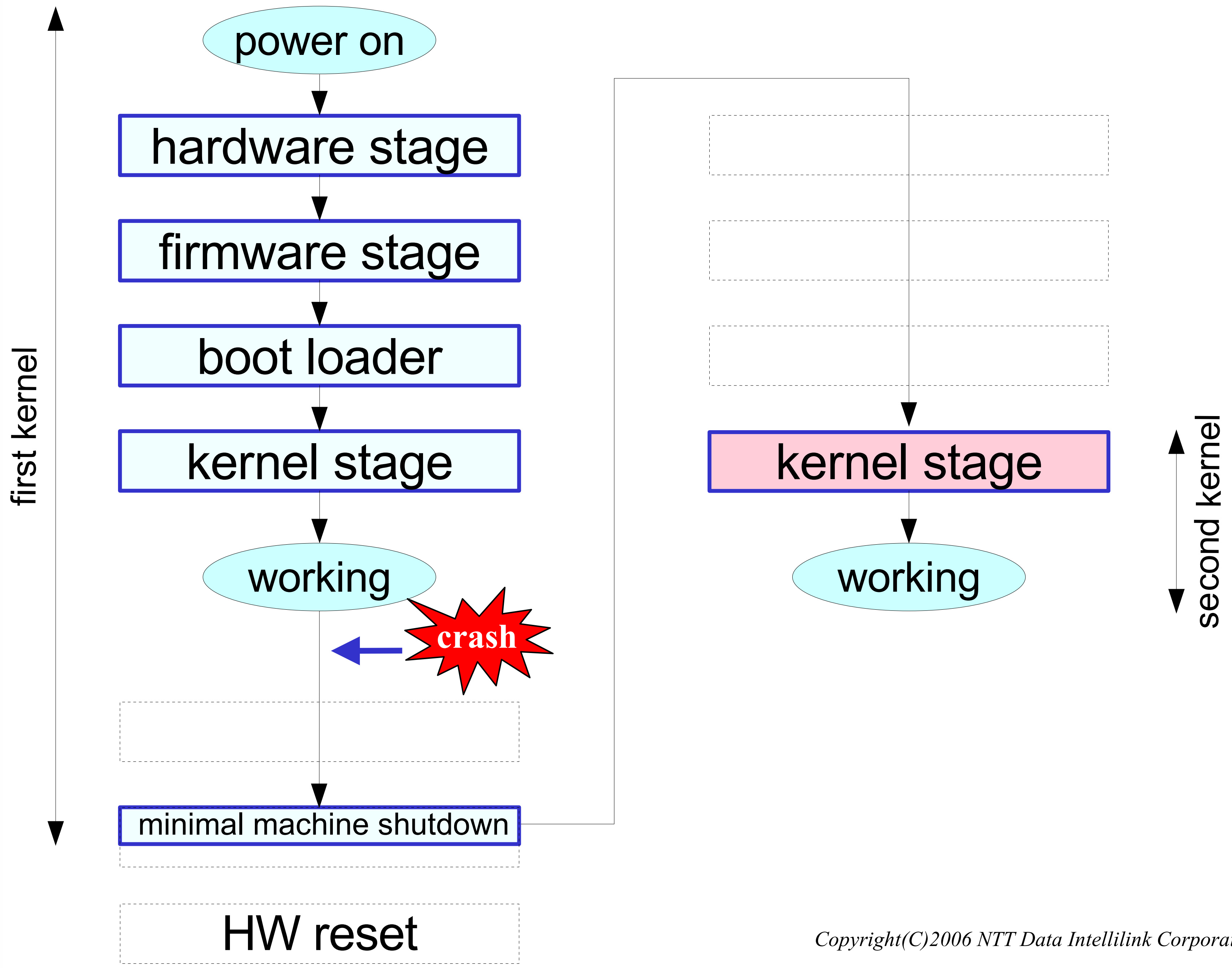
1.1. Standard boot process



1.2. Kexec boot process



1.3. Kdump boot process



2. device reinitialization

2.1. Device reinitialization issue

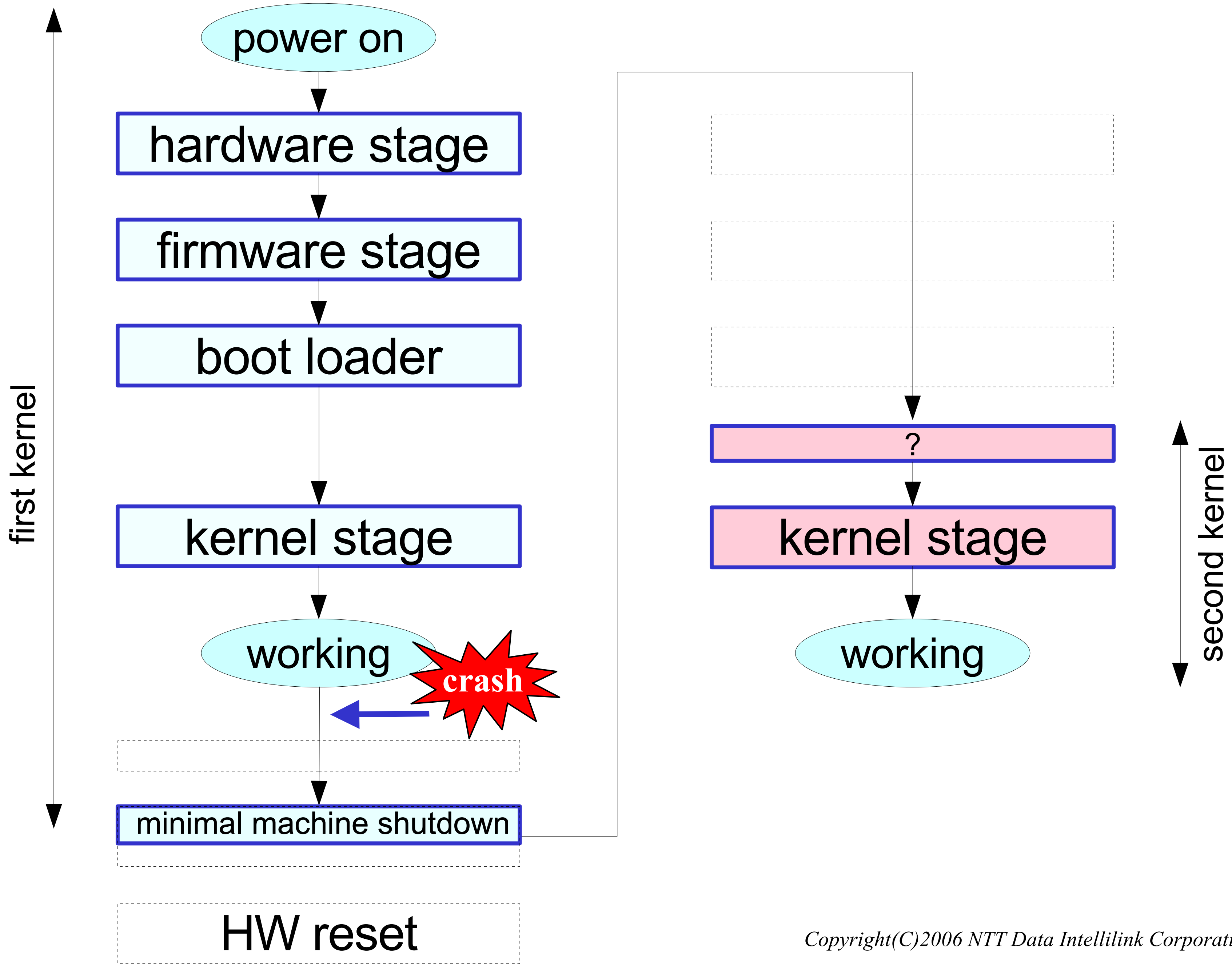
- State of devices after a kdump boot
  - No device shutdown in the crashing kernel
  - Firmware stage of the boot process is skipped
    - Devices are not reset
  - Devices might be operational or in an unknown state

2.1. Device reinitialization issue (cont)

- Drivers assume that the devices have been reset and/or that some pre-initialization has been performed
  - Drivers find devices in an unexpected state or receive an interrupt from the previous kernel's context
    - ✗ Drivers fail or raise an oops because this is an anomalous situation

3. tackling device reinitialization

3.1. Tackling device reinitialization



3.2. Possible solutions

- Make black list of drivers that are known to have problems
- Device reset (device soft-reset, PCI bus reset)
- Driver hardening to be able to initialize in potentially unreliable environments

3.3. Device reset

- Two possibilities
  - Individual device soft-reset
  - Bus resets (PCI, etc)
- Problems
  - Individual device soft-reset
    - May need to configure undocumented device registers
    - Not all devices have this capability
    - It is a time-consuming operation in some devices
  - PCI bus reset
    - Reset functionality not supported by all PCI buses

3.4. Driver hardening

- Things that can be done to initialize a device in an unreliable environment
  - Add hacks to the initialization code
  - Relax driver's consistency checks
  - Put devices into a good known state before proceeding with standard initialization (device pre-configuration)

4. a new approach

4.1. Device pre-configuration

- How do we restore devices to a good state after a soft-boot?
  1. Documentation available: follow the manual
  2. No documentation available: need to find out a good configuration
- During a normal boot the firmware performs part of the configuration and the driver does the rest
  - Need an infrastructure in the second kernel doing the job the firmware does during a regular boot?

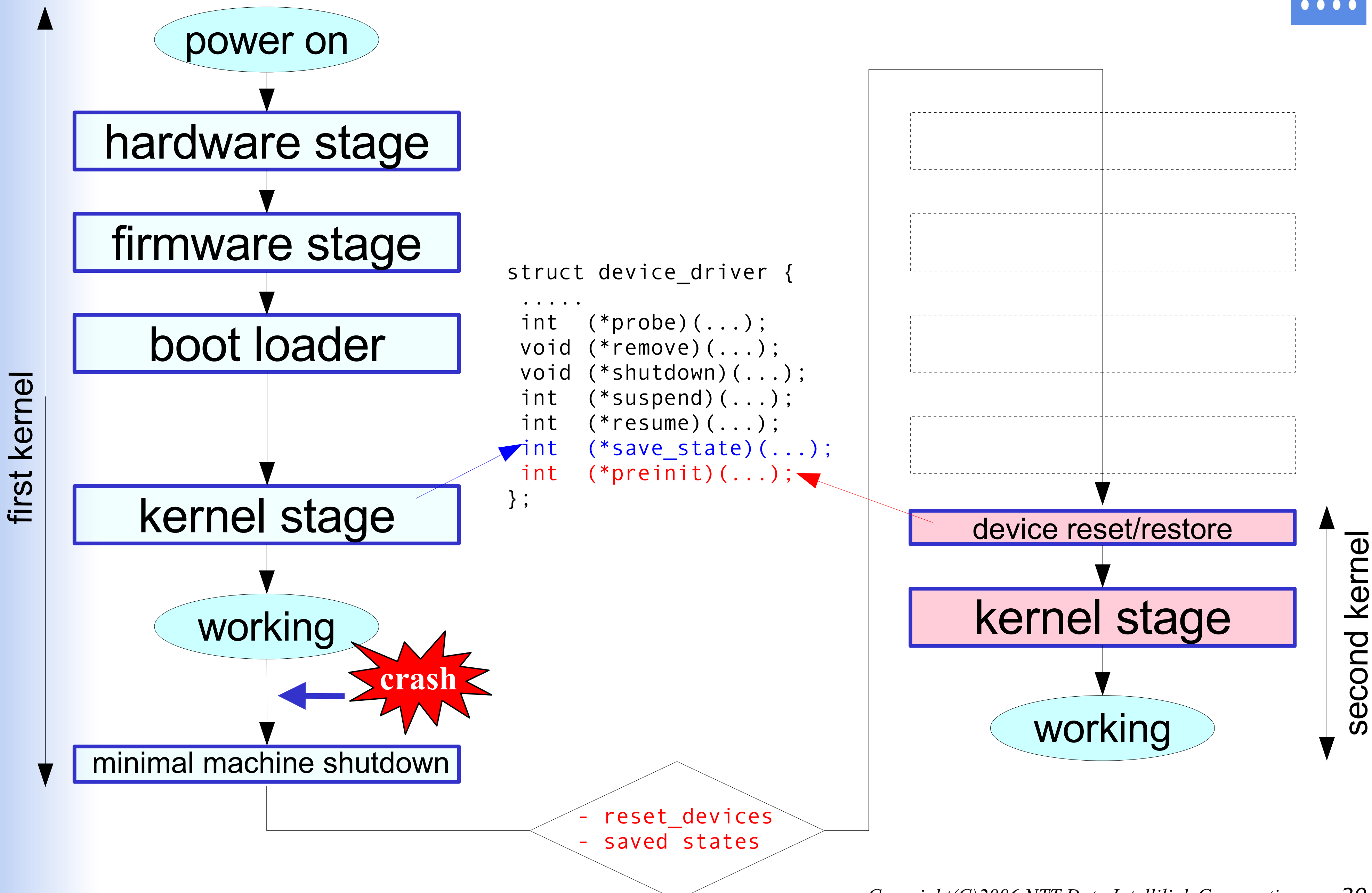
4.2. Device configuration restoration

- Save/restore device configuration
  - After a normal boot through the firmware save the configuration of all devices *before* trying to initialize them in the kernel stage of the boot process
  - In the event of a crash pass this information to the second kernel (infrastructure needed)
  - Use this information to pre-configure devices
    - This simulates the work done by the firmware
    - Look for inspiration from suspend/resume code
  - Proceed with the standard initialization

4.2. Device configuration restoration

- Save/restore device configuration
  - After a normal boot through the firmware save the configuration of all devices *before* trying to initialize them in the kernel stage of the boot process
  - In the event of a crash pass this information to the second kernel (infrastructure needed)
  - Use this information to pre-configure devices
    - This simulates the work done by the firmware
    - Look for inspiration from suspend/resume code
  - Proceed with the standard initialization

4.3. Tackling device reinitialization



4.4. Discussions topics

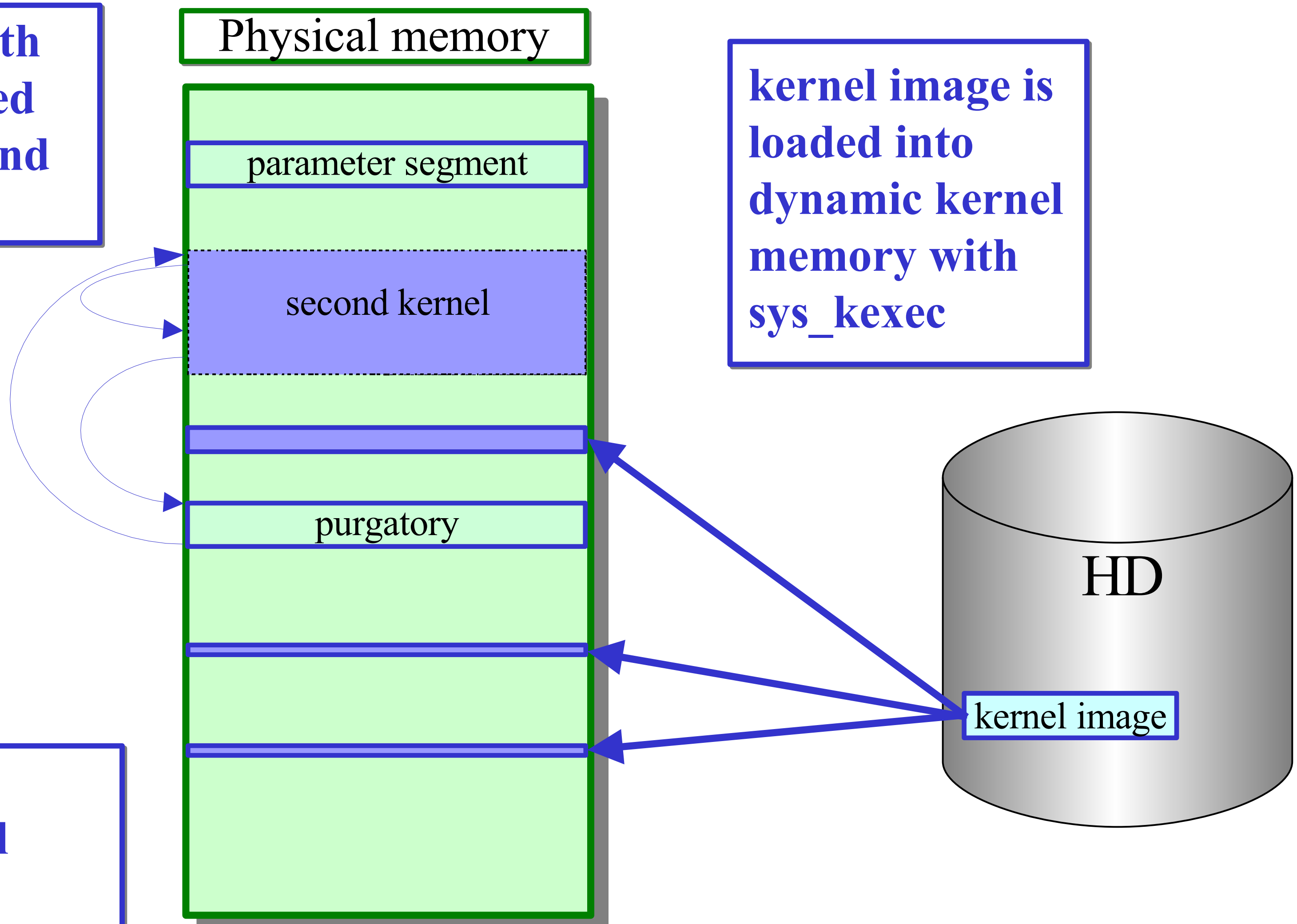
- Need to notify the kernel that it is booting into a special environment?
- Need to pass configuration information between the first and the second kernel?
  - Infrastructure to pass information to second kernel
  - New function callback in device drivers to save the configuration as performed by the firmware (does not have to be provided)
  - `preinit` function callback to be invoked when `reset_devices` has been set
    - Soft-reset or pre-configure devices when possible

Thanks for your attention

Contact: fernando@intellilink.co.jp

1.1. kexec reboot

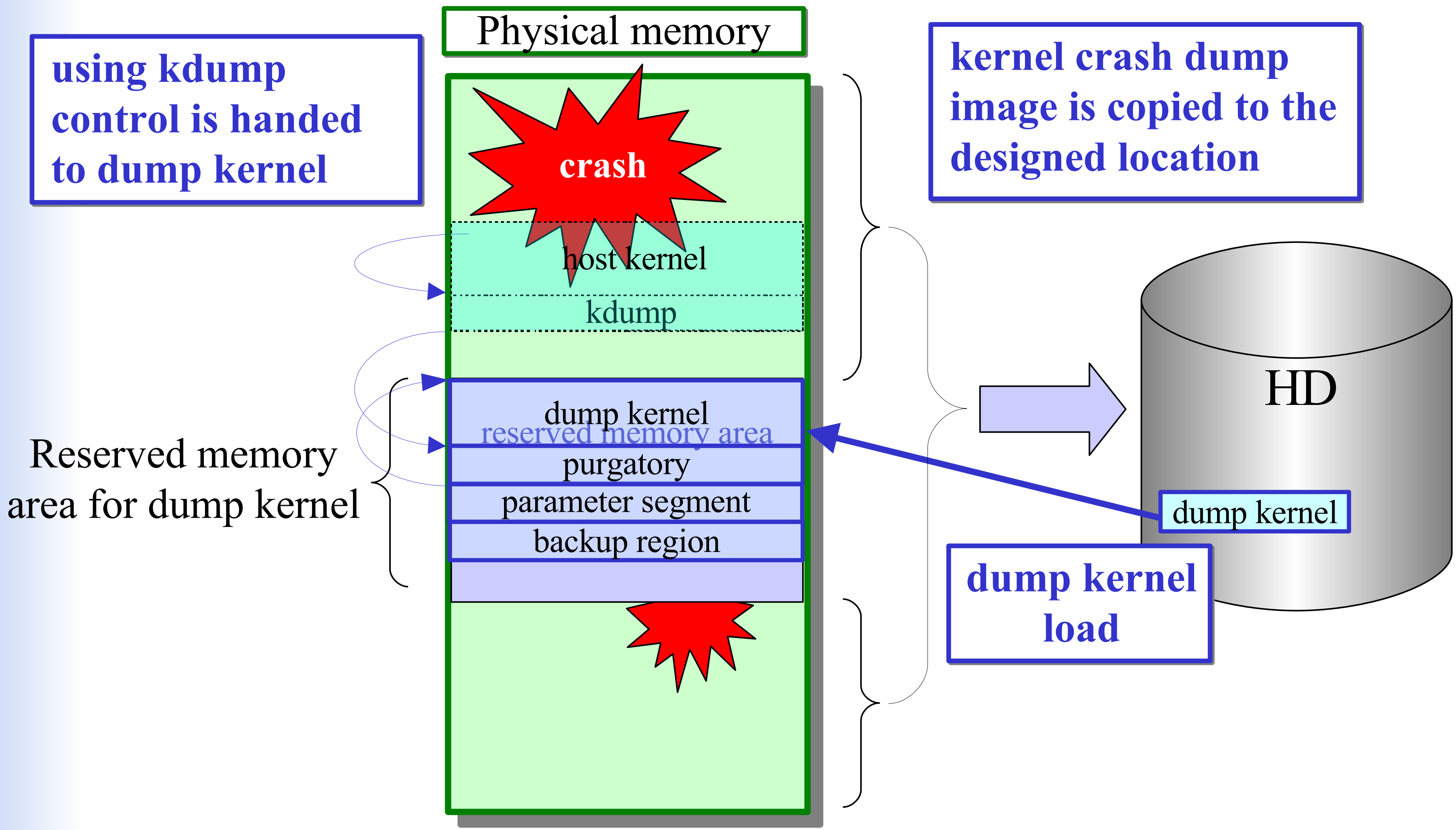
using reboot path control is handed over to the second kernel



kernel image is loaded into dynamic kernel memory with sys_kexec

kernel image is copied to its final destination

1.2. kexec-based crash dumping



2.2. Device reinitialization failure cases

- Example 1
  - After the first kernel crash the device is operational and sending interrupts
  - The driver loads
  - Underlying device sends an interrupt indicating completion of a command issued from the previous kernel's context
  - Driver does not know anything about it
  - Driver raises BUG() as this is anomalous

2.2. Device reinitialization failure cases

- Example 2
  - SCSI controller is left with interrupt line asserted and reply FIFO is not empty
  - Driver starts initializing in the second kernel
  - Driver receives the interrupt the moment request_irq() is called
  - Interrupt handler reads the message from reply FIFO
  - Interrupt handler tries to access the associated message frame
  - The message frame is not valid in the new kernel's context so the kernel panics

3.2. Changing initialization behavior

- A change in the normal initialization process can be initiated in two ways:
  - Make kernel kexec/kdump aware
    - Notify boot method to the second kernel using a kernel boot option
    - Should device reset be executed by default?
  - Look at the devices/controllers and see if they are in a bad/unexpected state

3.3. Device reset – device soft-reset

- Soft-reset the device before proceeding with rest of the initialization
  - The device flushes the messages issued from the previous kernel's context (if supported)
  - Resume initialization
- Problems
  - May need to configure undocumented device registers
  - Not all devices have this capability
  - It is a time-consuming operation in some devices
    - Firmware and self-test operations in SCSI controllers may be on the order of minutes

3.4. Device reset – PCI bus reset

- Set the PCI bus reset bit in the PCI bridge to so initiate the PCI bus reset
- Requires firmware/BIOS to export hook to SW
- Problems
  - Reset functionality is not supported by all PCI buses
  - Might be ignored by devices
  - Potentially unsafe in legacy systems
    - Might affect the memory bus too

3.6. 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)