Unit 6: Device Management

6.3. Windows I/O Processing

Roadmap for Section 6.3

- Driver and Device Objects
- I/O Request Packets (IRP) Processing
- Driver Layering and Filtering
- Plug-and-Play (PnP) and Power Manager Operation
- Monitoring I/O Activity with Filemon

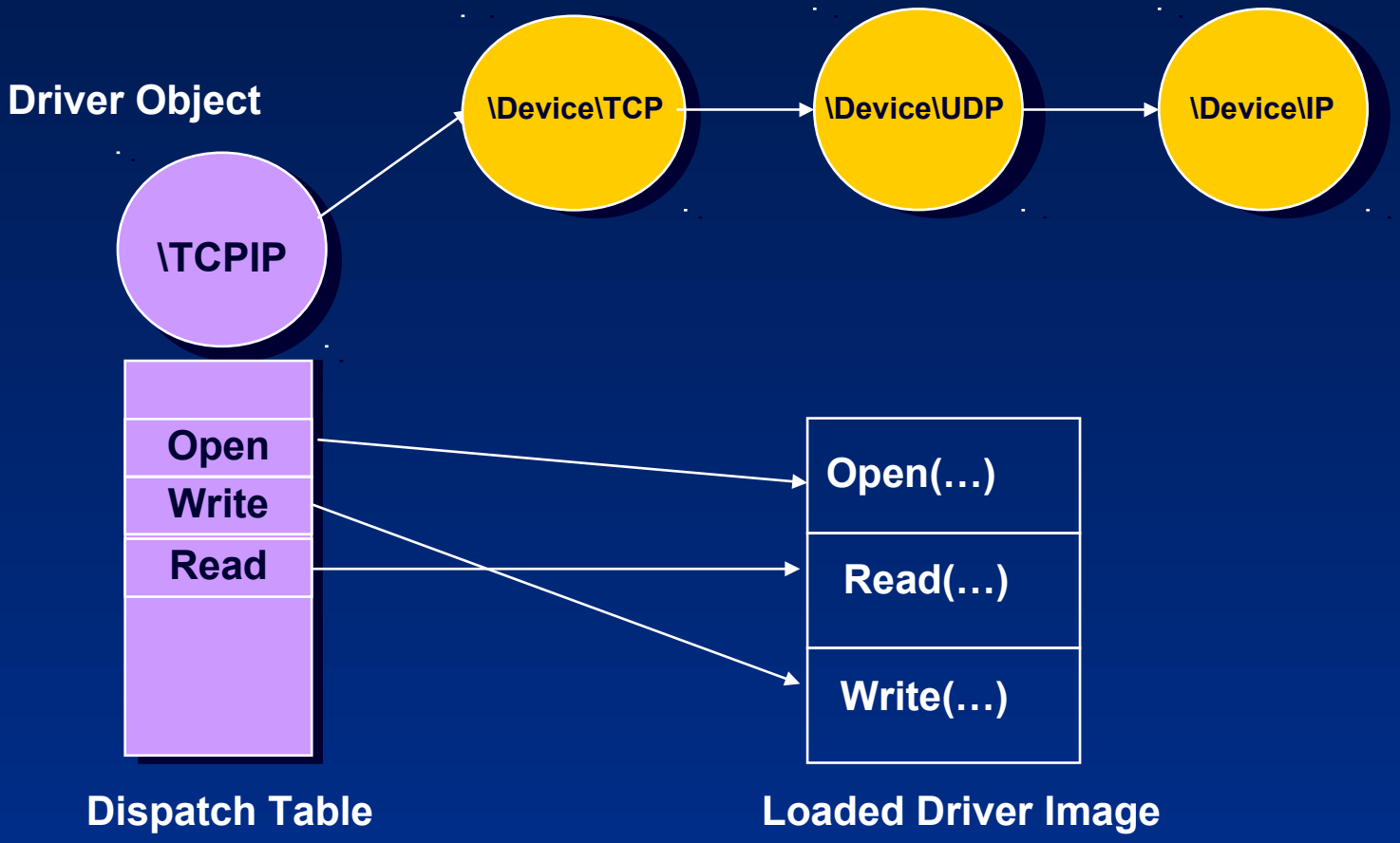
Driver Object

- A driver object represents a loaded driver
  - Names are visible in the Object Manager namespace under \Drivers
  - A driver fills in its driver object with pointers to its I/O functions e.g. open, read, write
  - When you get the “One or More Drivers Failed to Start” message it is because the Service Control Manager didn’t find one or more driver objects in the \Drivers directory for drivers that *should* have started

Device Objects

- A device object represents an instance of a device
  - Device objects are linked in a list off the driver object
  - A driver creates device objects to represent the interface to the logical device, so each generally has a unique name visible under \Devices
  - Device objects point back at the Driver object

Driver and Device Objects



TCP/IP Drivers Driver and Device Objects

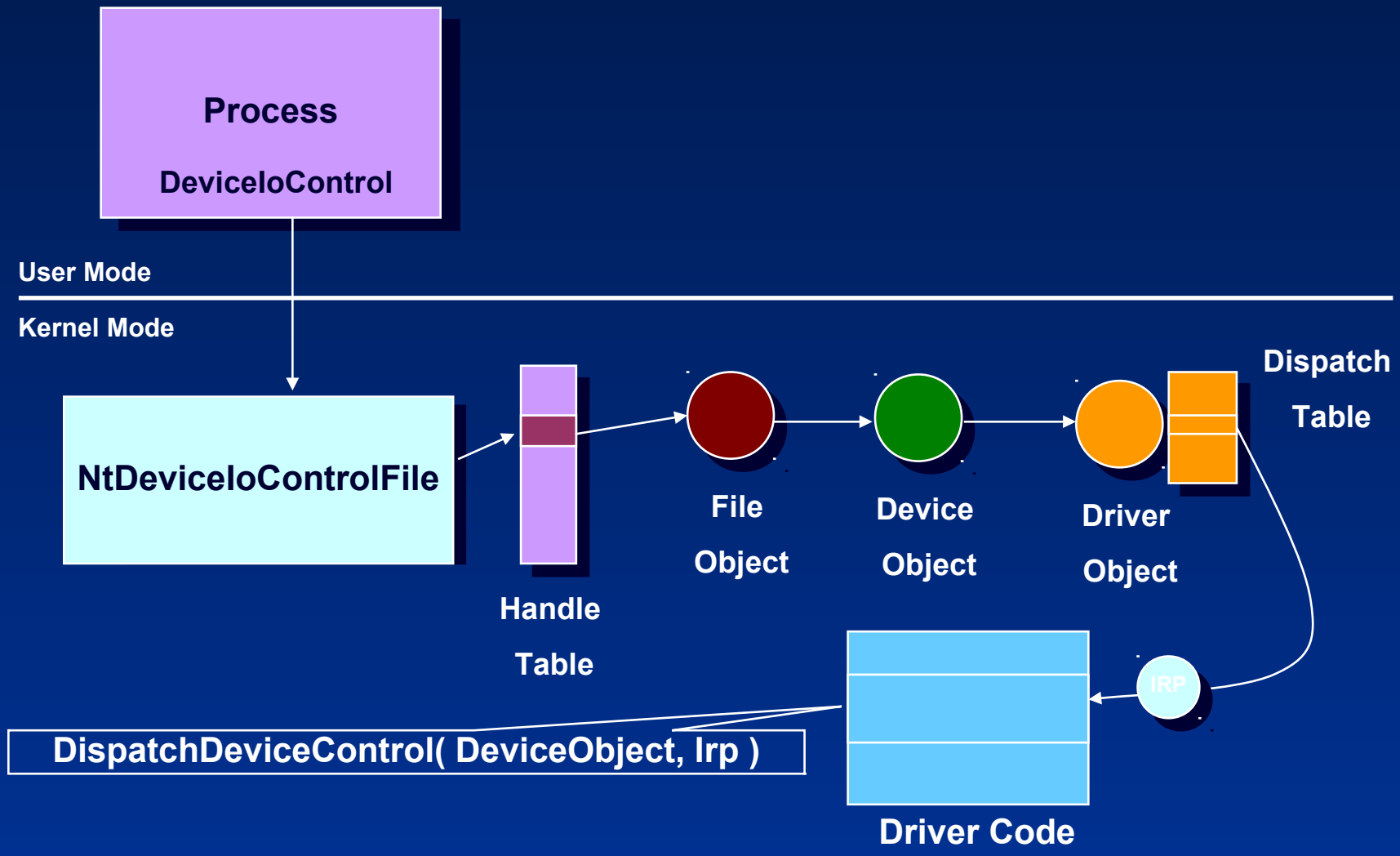
File Objects

- Represents open instance of a device (files on a volume are virtual devices)
  - Applications and drivers “open” devices by name
  - The name is parsed by the Object Manager
  - When an open succeeds the Object Manager creates a file object to represent the open instance of the device and a file handle in the process handle table
- A file object links to the device object of the “device” which is opened
- File objects store additional information
  - File offset for sequential access
  - File open characteristics (e.g. delete-on-close)
  - File name
  - Accesses granted for convenience

I/O Request Packets

- System services and drivers allocate I/O request packets to describe I/O
- A request packet contains:
  - File object at which I/O is directed
  - I/O characteristics (e.g. synchronous, non-buffered)
  - Byte offset
  - Length
  - Buffer location
- The I/O Manager locates the driver to which to hand the IRP by following the links:

Putting it Together: Request Flow

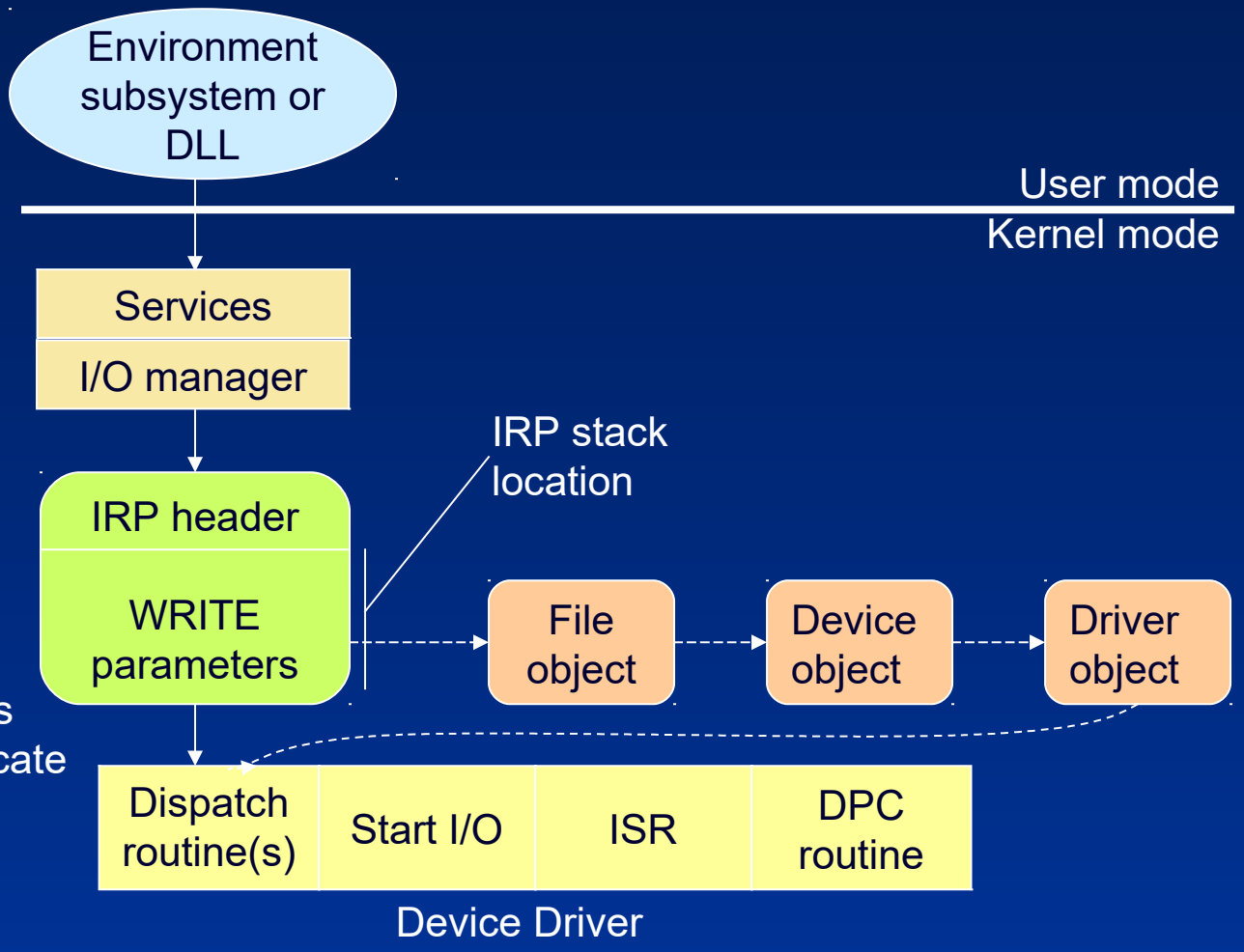


I/O Request Packet

1)An application writes a file to the printer, passing a handle to the file object

2)The I/O manager creates an IRP and initializes first stack location

3)The I/O manager uses the driver object to locate the WRITE dispatch routine and calls it, passing the IRP



IRP data

IRP consists of two parts:

- Fixed portion (header):
  - Type and size of the request
  - Whether request is synchronous or asynchronous
  - Pointer to buffer for buffered I/O
  - State information (changes with progress of the request)
- One or more stack locations:
  - Function code
  - Function-specific parameters
  - Pointer to caller‘s file object
- While active, IRPs are stored in a thread-specific queue
  - I/O system may free any outstanding IRPs if thread terminates

I/O Processing – synchonous I/O to a single-layered driver

1. The I/O request passes through a subsystem DLL
2. The subsystem DLL calls the I/O manager‘s NtWriteFile() service
3. I/O manager sends the request in form of an IRP to the driver (a device driver)
4. The driver starts the I/O operation
5. When the device completes the operation and interrupts the CPU, the device driver services the interrupt
6. The I/O manager completes the I/O request

Completing an I/O request

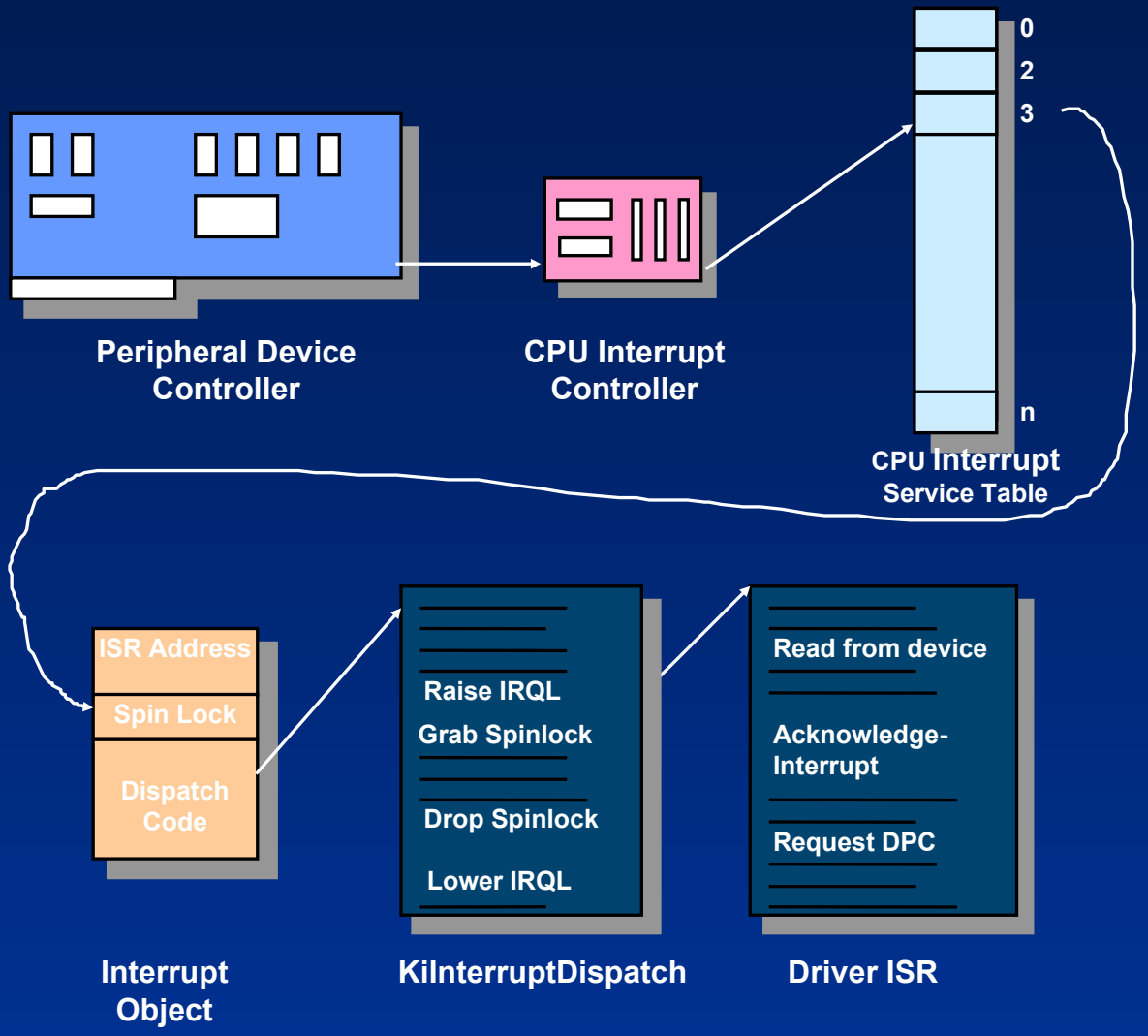
Servicing an interrupt:

- ISR schedules Deferred Procedure Call (**DPC**); dismisses interrupt
- **DPC** routine starts next I/O request and completes interrupt servicing
- May call completion routine of higher-level driver

I/O completion:

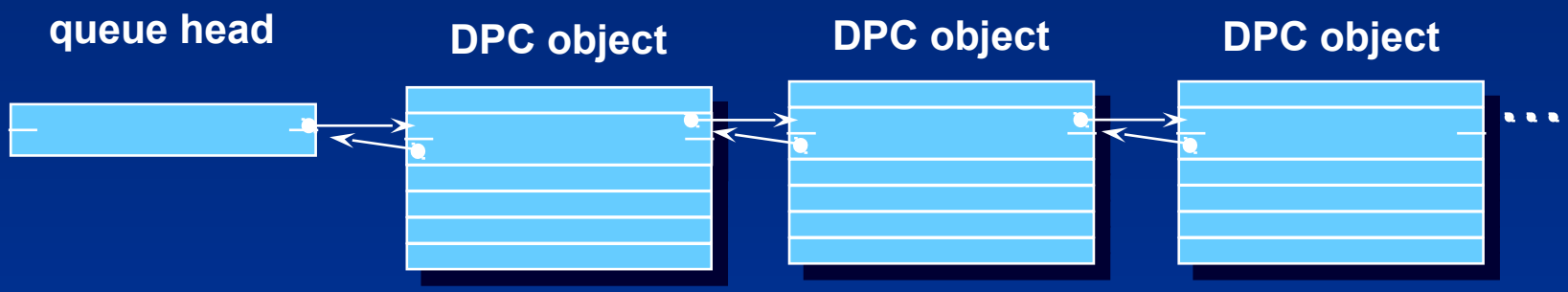
- Record the outcome of the operation in an I/O status block
- Return data to the calling thread – by queuing a kernel-mode Asynchronous Procedure Call (**APC**)
- **APC** executes in context of calling thread; copies data; frees IRP; sets calling thread to signaled state
- I/O is now considered complete; waiting threads are released

Flow of Interrupts



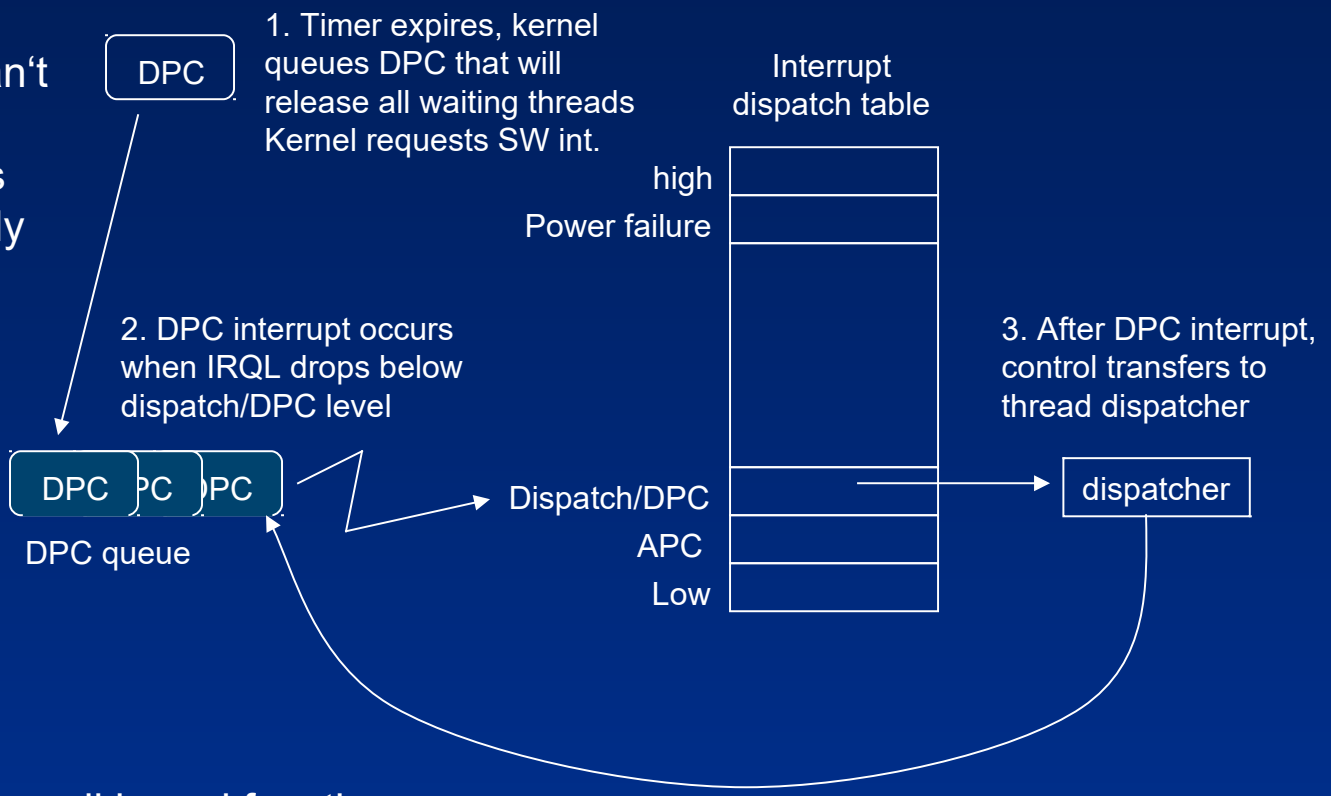
Servicing an Interrupt: Deferred Procedure Calls (DPCs)

- Used to defer processing from higher (device) interrupt level to a lower (dispatch) level
  - Also used for quantum end and timer expiration
- Driver (usually ISR) queues request
  - One queue per CPU. DPCs are normally queued to the current processor, but can be targeted to other CPUs
  - Executes specified procedure at dispatch IRQL (or “dispatch level”, also “DPC level”) when all higher-IRQL work (interrupts) completed
  - Maximum times recommended: ISR: 10 usec, DPC: 25 usec
    - See http://www.microsoft.com/whdc/driver/perform/mmdrv.mspx



Delivering a DPC

DPC routines can‘t assume what process address space is currently mapped



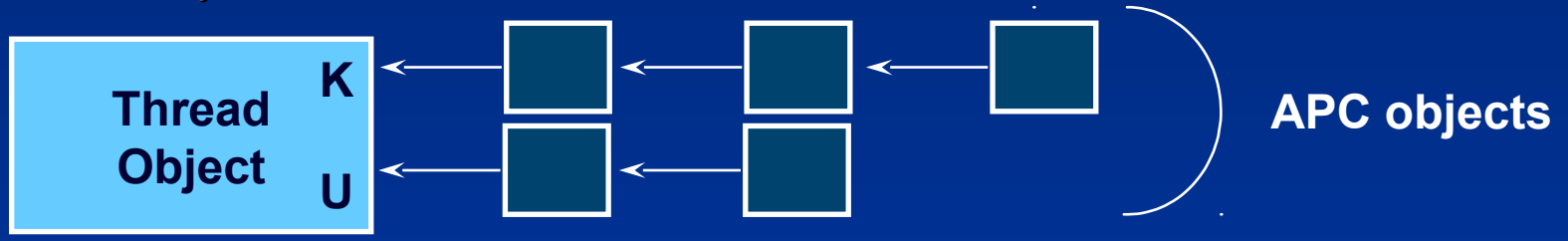
DPC routines can call kernel functions but can‘t call system services, generate page faults, or create or wait on objects

I/O Completion: Asynchronous Procedure Calls (APCs)

- Execute code in context of a particular user thread
  - APC routines can acquire resources (objects), incur page faults, call system services
- APC queue is thread-specific
- User mode & kernel mode APCs
  - Permission required for user mode APCs
- Executive uses APCs to complete work in thread space
  - Wait for asynchronous I/O operation
  - Emulate delivery of POSIX signals
  - Make threads suspend/terminate itself (env. subsystems)
- APCs are delivered when thread is in alertable wait state
  - WaitForMultipleObjectsEx(), SleepEx()

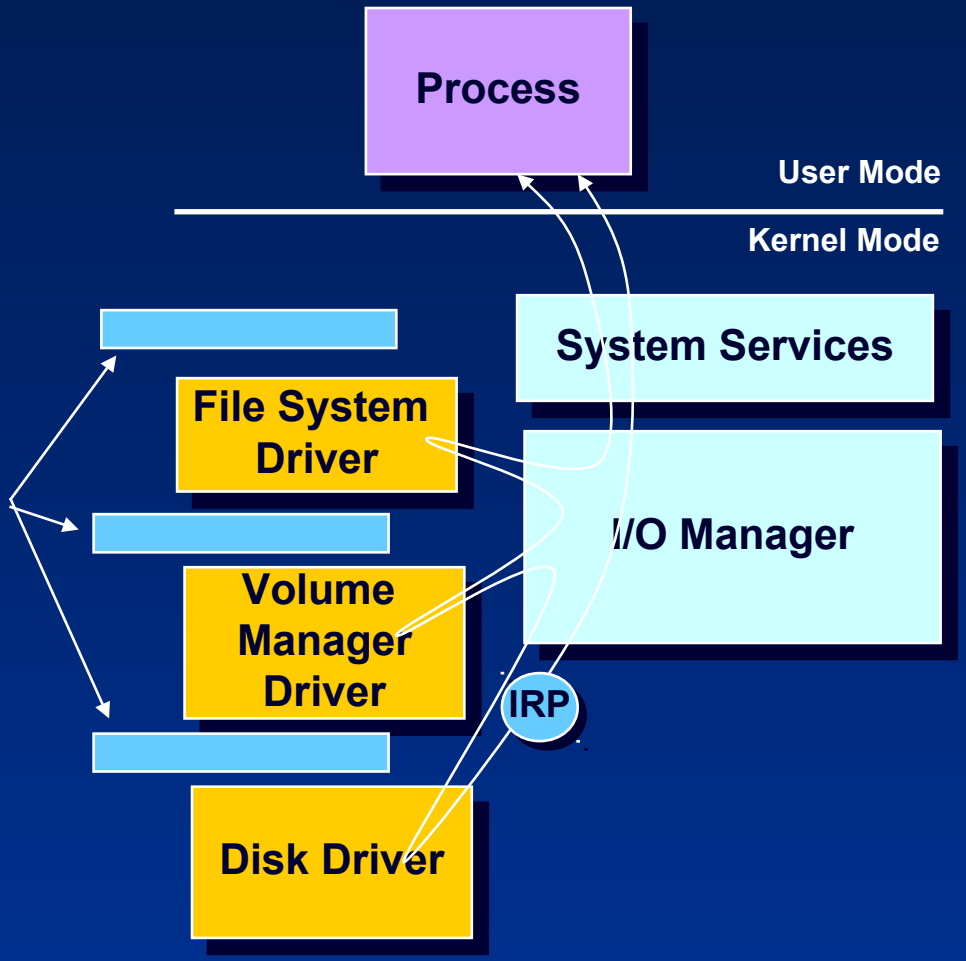
Asynchronous Procedure Calls (APCs)

- Special kernel APCs
  - Run in kernel mode, at IRQL 1
  - Always deliverable unless thread is already at IRQL 1 or above
  - Used for I/O completion reporting from “arbitrary thread context”
  - Kernel-mode interface is linkable, but not documented
- “Ordinary” kernel APCs
  - Always deliverable if at IRQL 0, unless explicitly disabled (disable with KeEnterCriticalRegion)
- User mode APCs
  - Used for I/O completion callback routines (see ReadFileEx, WriteFileEx); also, QueueUserApc
  - Only deliverable when thread is in “alertable wait”



Driver Layering and Filtering

- To divide functionality across drivers, provide added value, etc.
  - Only the lowest layer talks to the I/O hardware
- "Filter drivers" attach their devices to other devices
  - They see all requests first and can manipulate them
  - Example filter drivers:
    - File system filter driver
    - Bus filter driver



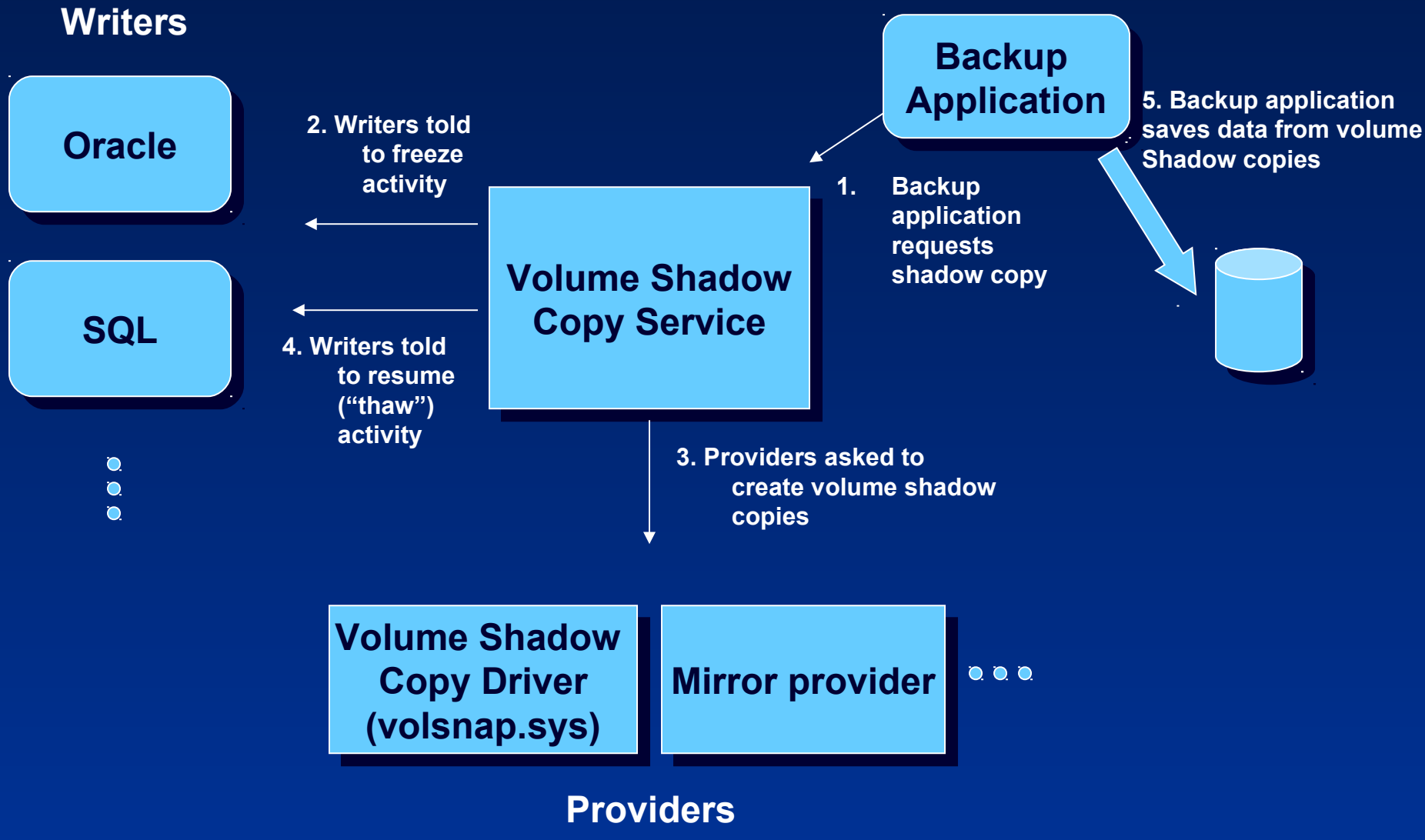
Driver Filtering: Volume Shadow Copy

- Volume Shadow Copy was introduced in Windows XP/Server 2003
  - Addresses the “backup open files” problem
- Volumes can be “snapshotted”
- Allows “hot backup” (including open files)
- Applications can tie in with mechanism to ensure consistent snapshots
  - Database servers flush transactions
  - Windows components such as the Registry flush data files
- Different snapshot providers can implement different snapshot mechanisms:
  - Copy-on-write
  - Mirroring

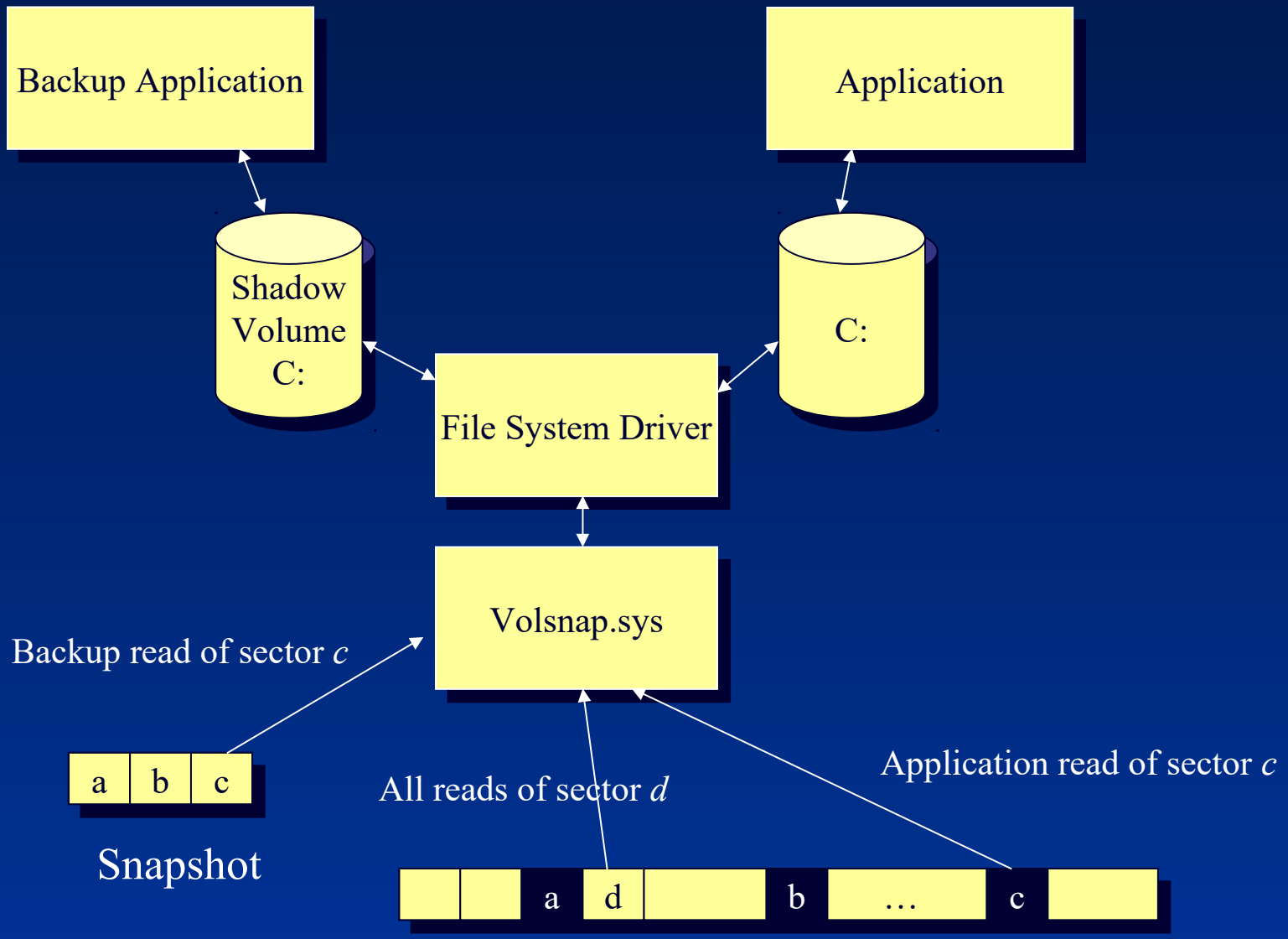
Volsnap is the built-in provider:

- Built into Windows XP/Server 2003
- Implements copy-on-write snapshots
- Saves volume changes in files on the volume
- Uses defrag API to determine where the file is and where paging file is to avoid tracking their changes

Volume Snapshots



Volsnap.sys

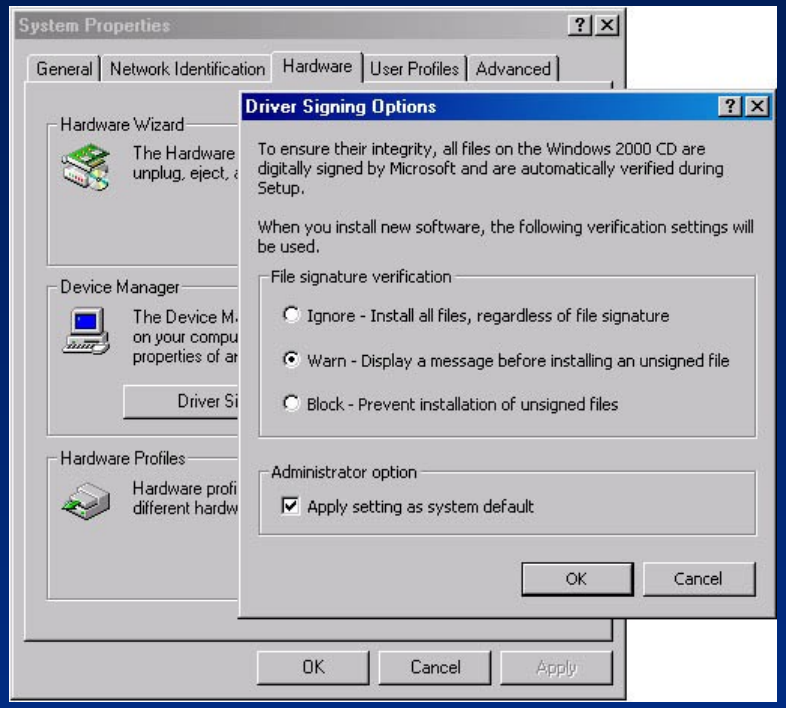

Backup Application
Application
Shadow Volume C:
C:
File System Driver
Volsnap.sys
Backup read of sector c
a
b
c
Snapshot
All reads of sector d
Application read of sector c
a
d
b
…
c

The PnP Manager

- In NT 4.0 each device driver was responsible for enumerating all supported busses in search of devices they support
- From Windows 2000, the PnP Manager has bus drivers enumerate their busses and inform it of present devices
- If the device driver for a device not already present on the system, the PnP Manager in the kernel informs the user-mode PnP Manager to start the Hardware Wizard

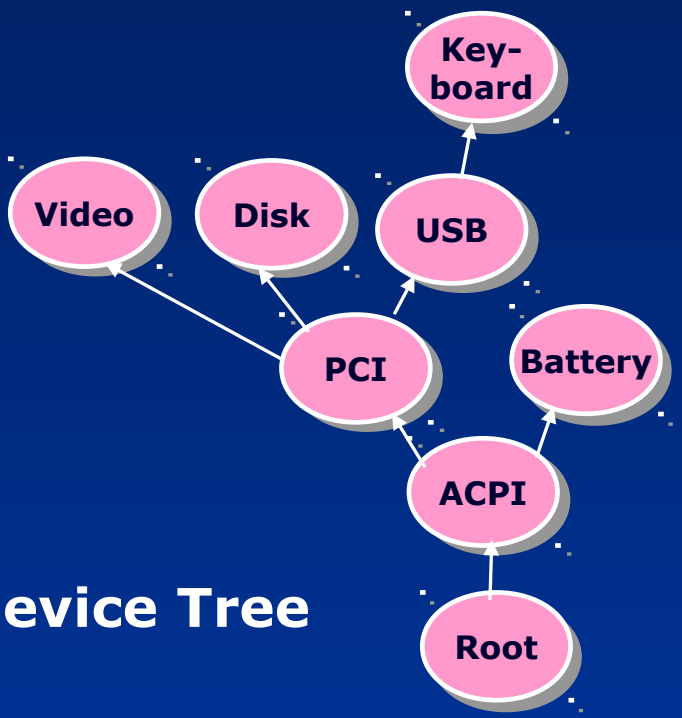
The PnP Manager

- Once a device driver is located, the PnP Manager determines if the driver is signed
  - If the driver is not signed, the system's driver signing policy determines whether or not the driver is installed
- After loading a driver, the PnP Manager calls the driver's AddDevice entry point
  - The driver informs the PnP Manager of the device's resource requirements
  - The PnP Manager reconfigures other devices to accommodate the new device

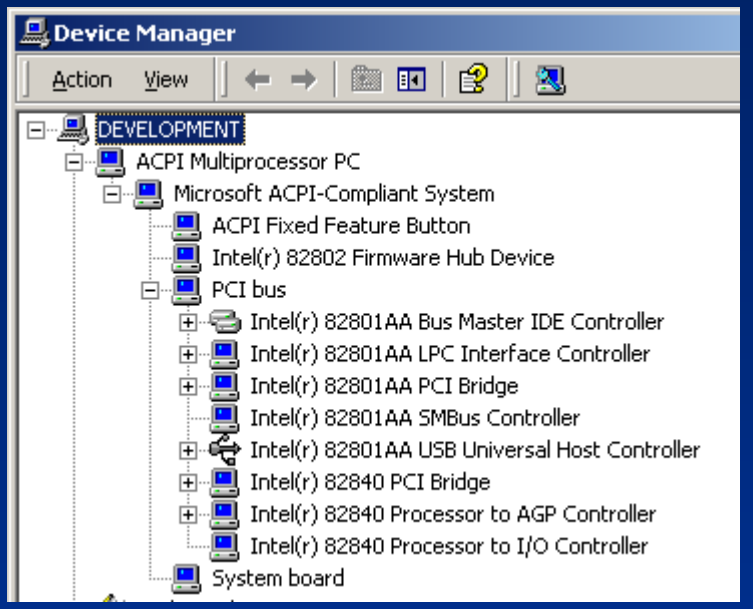


The PnP Manager

- Enumeration is recursive, and directed by bus drivers
  - Bus drivers identify device on a bus
- As busses and devices are registered, a device tree is constructed, and filled in with devices



Device Tree

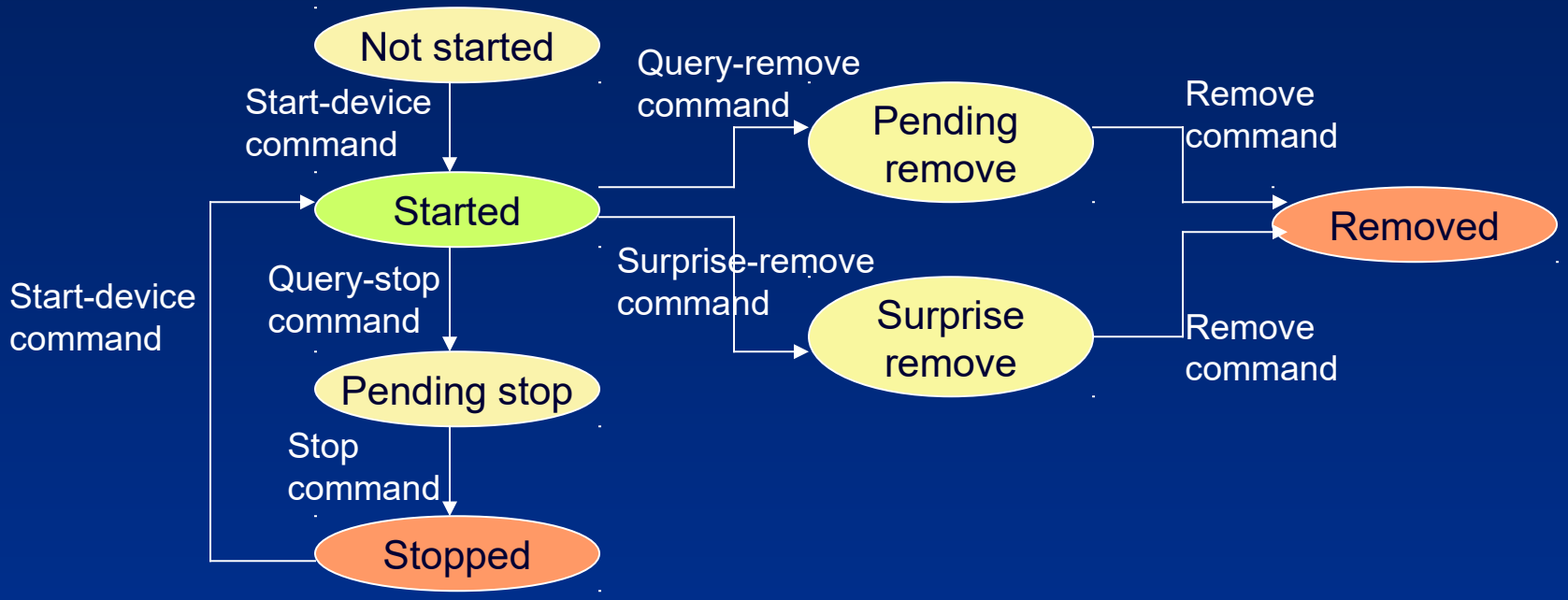


Resource Arbitration

- Devices require system hardware resources to function (e.g. IRQs, I/O ports)
- The PnP Manager keeps track of hardware resource assignments
- If a device requires a resource that's already been assigned, the PnP Manager tries to reassign resources in order to accommodate
- Example:
  1. Device 1 can use IRQ 5 or IRQ 6
  2. PnP Manager assigns IRQ 5 to it
  3. Device 2 can only use IRQ 5
  4. PnP Manager reassigns IRQ 6 to Device 1
  5. PnP Manager assigns IRQ 5 to Device 2

Plug and Play (PnP) State Transitions

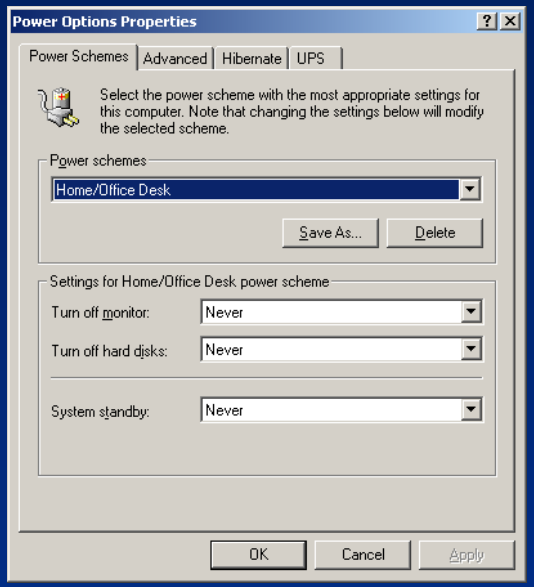
- PnP manager recognizes hardware, allocates resources, loads driver, notifies about configuration changes



Device Plug and Play state transitions

The Power Manager

- A system must have an ACPI-compliant BIOS for full compatibility (APM gives limited power support)
- A number of factors guide the Power Manager's decision to change power state:
  - System activity level
  - System battery level
  - Shutdown, hibernate, or sleep requests from applications
  - User actions, such as pressing the power button
  - Control Panel power settings



The system can go into low power modes, but it requires the cooperation of every device driver – applications can provide their input as well

The Power Manager

- There are different system power states:
  - On
    - Everything is fully on
  - Standby
    - Intermediate states
    - Lower standby states must consume less power than higher ones
  - Hibernating
    - Save memory to disk in a file called hiberfil.sys in the root directory of the system volume
  - Off
    - All devices are off
- Device drivers manage their own power level
  - Only a driver knows the capabilities of their device
  - Some devices only have “on” and “off”, others have intermediate states
- Drivers can control their own power independently of system power
  - Display can dim, disk spin down, etc.

Power Manager

- based on the Advanced Configuration and Power Interface (ACPI)

State	Power Consumption	Software Resumption	HW Latency
S0 (fully on)	Maximum	Not applicable	None
S1 (sleeping)	Less than S0, more than S2	System resumes where it left off (returns to S0)	Less than 2 sec.
S2 (sleeping)	Less than S1, more than S3	System resumes where it left off (returns to S0)	2 or more sec.
S3 (sleeping)	Less than S2, processor is off	System resumes where it left off (returns to S0)	Same as S2
S4 (hibernating)	Trickle current to power button and wake circuitry	System restarts from hibernate file and resumes where it left off (returns to S0)	Long and undefined
S5 (fully off)	Trickle current to power button	System boot	Long and undefined

System Power-State Definitions

Further Reading

- Pavel Yosifovich, Alex Ionescu, et al., “Windows Internals”, 7th Edition, Microsoft Press, 2017.
  - Chapter 6 – I/O system (from pp. 669)
    - I/O processing (from pp. 707)
    - The Plug and Play (PnP) manager (from pp. 776)
    - The power manager (from pp. 817)