Unit 7: The Input/Output System

7.2. The Windows 2000 I/O System

The Windows 2000 I/O System

Component of Windows 2000 executive;
resides in NTOSKRNL.EXE

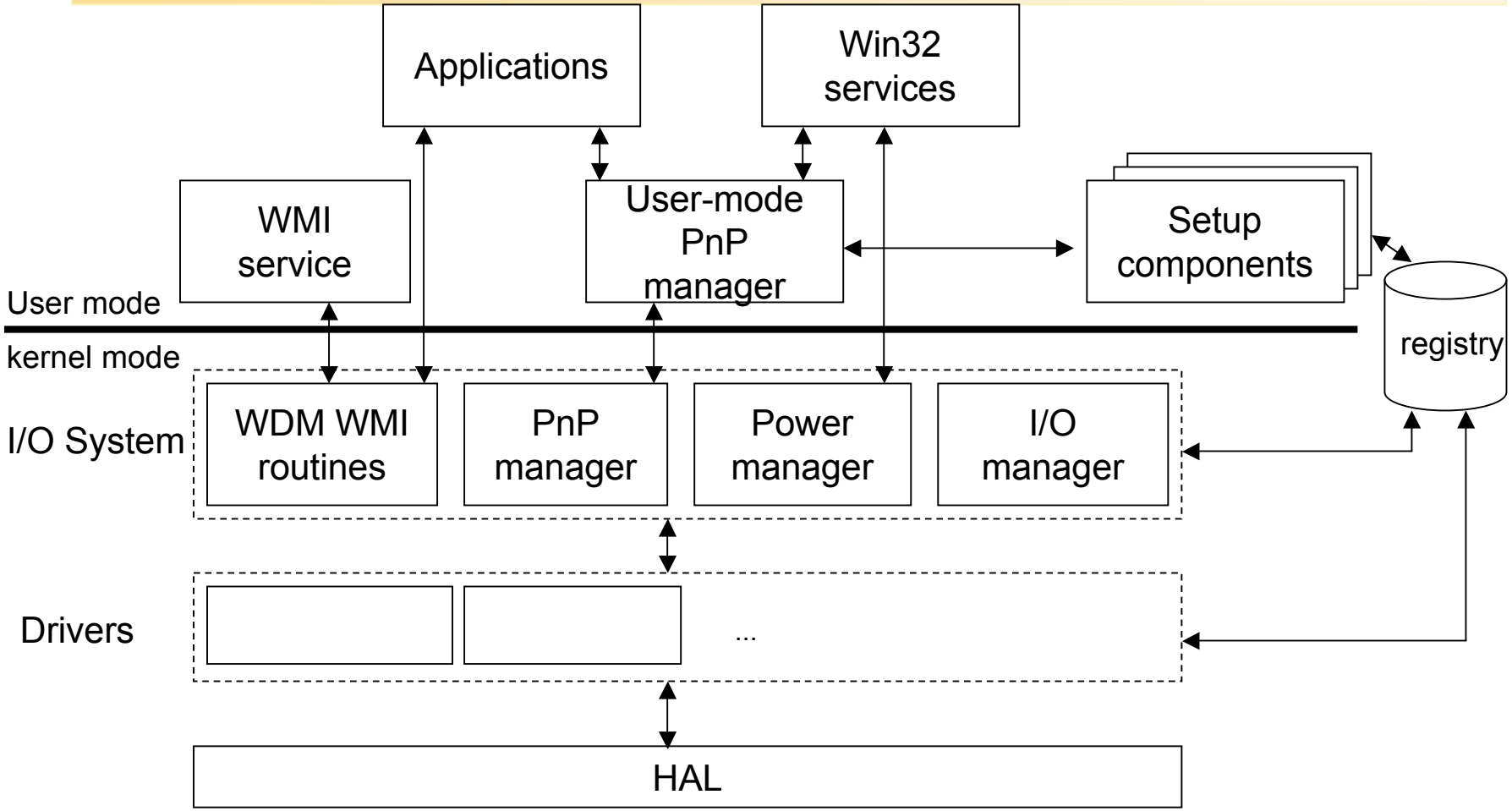
- Accepts I/O requests from user-mode/kernel-mode
- Delivers them to I/O devices

- Filters between user-mode I/O and hardware:
  - File system drivers
  - Filter drivers
  - Low-level device drivers

Design Goals

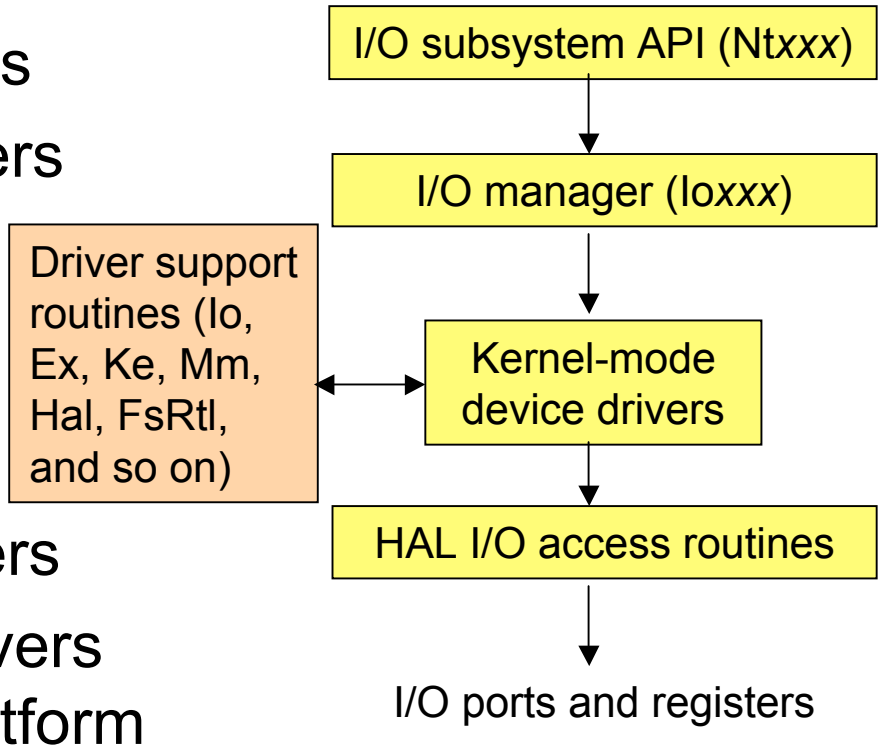
- Fast I/O processing on single / multiprocessor systems
- Protection for shareable resources
  - Using Windows 2000 security mechanisms
- Meet requirements dictated by different subsystems
- Provide common services for device drivers
  - Ease device driver development
  - Allow drivers to be written in high-level language
- Dynamic addition/removal of device drivers
- Support multiple file systems (FAT, CDFS, UDF, NTFS)
- Provide mapped file I/O capabilities

I/O System Components



The Flow of a typical I/O Request

- I/O manager controls processing of I/O requests
- Kernel-mode device drivers translate I/O requests into control requests to hardware devices
- Driver support routines are called by device drivers
- HAL routines insulate drivers from variations in HW platform



I/O Manager

- Framework for delivery of I/O request packets (IRPs)
- IRPs control processing of all I/O operations
  (exception: fast I/O does not use IRPs)
- I/O manager:
  - creates an IRP for each I/O operation;
  - passes IRP to correct drivers;
  - deletes IRP when I/O operation is complete
- Driver:
  - Receives IRP
  - Performs operations specified by IRP
  - Passes IRP back to I/O manager or to another driver (via I/O manager) for further processing

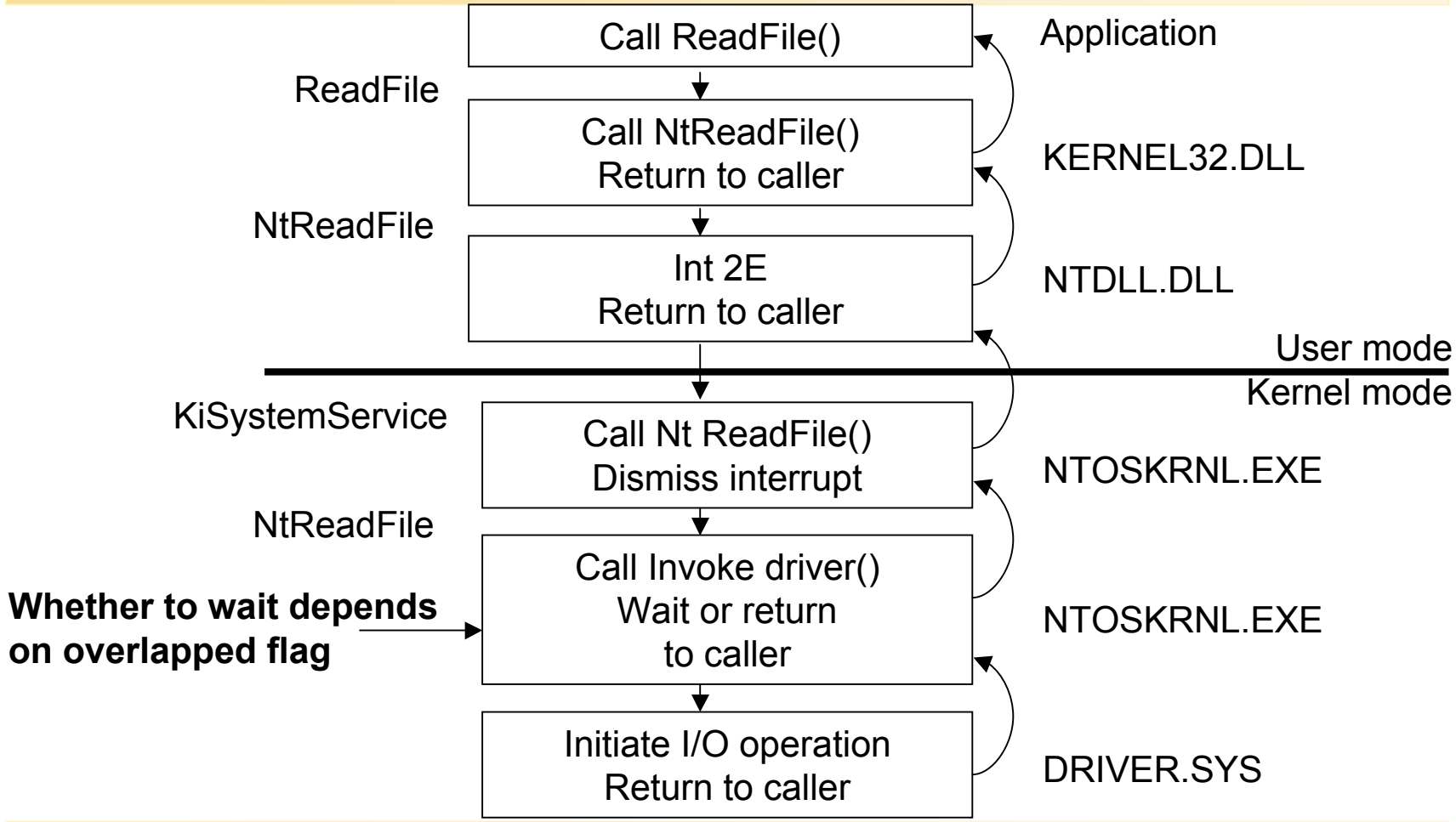
I/O Manager (contd.)

- Supplies common code for different drivers:
  - Drivers become simpler, more compact
- I/O manager:
  - Allows driver to call other drivers
  - Manages buffers for I/O requests
  - Provides time-out support for drivers
  - Records which installable file systems are loaded
  - Provides flexible I/O services to environment subsystems (Win32/POSIX asynchronous I/O)
- Layered processing of I/O requests possible:
  - Drivers can call each other (via I/O manager)

I/O Functions

- Advanced features beyond *open, close, read, write*:
- **Asynchronous I/O**:
  - May improve throughput/performance:
    continue program execution while I/O is in progress
  - Must specify FILE_FLAG_OVERLAPPED on Win32 CreateFile()
  - Programmer is responsible for synchronization of I/O requests
- Internally, all I/O is performed asynchronously
  - I/O system returns to caller only if file was opened for asynch. I/O
  - For synchronous I/O, wait is done in kernel mode depending on overlapped flag in file object
- Status of pending I/O can be tested:
  - via Win32 function: *HasOverlappedIoCompleted()*
  - when using I/O completion ports: *GetQueuedCompletionStatus()*

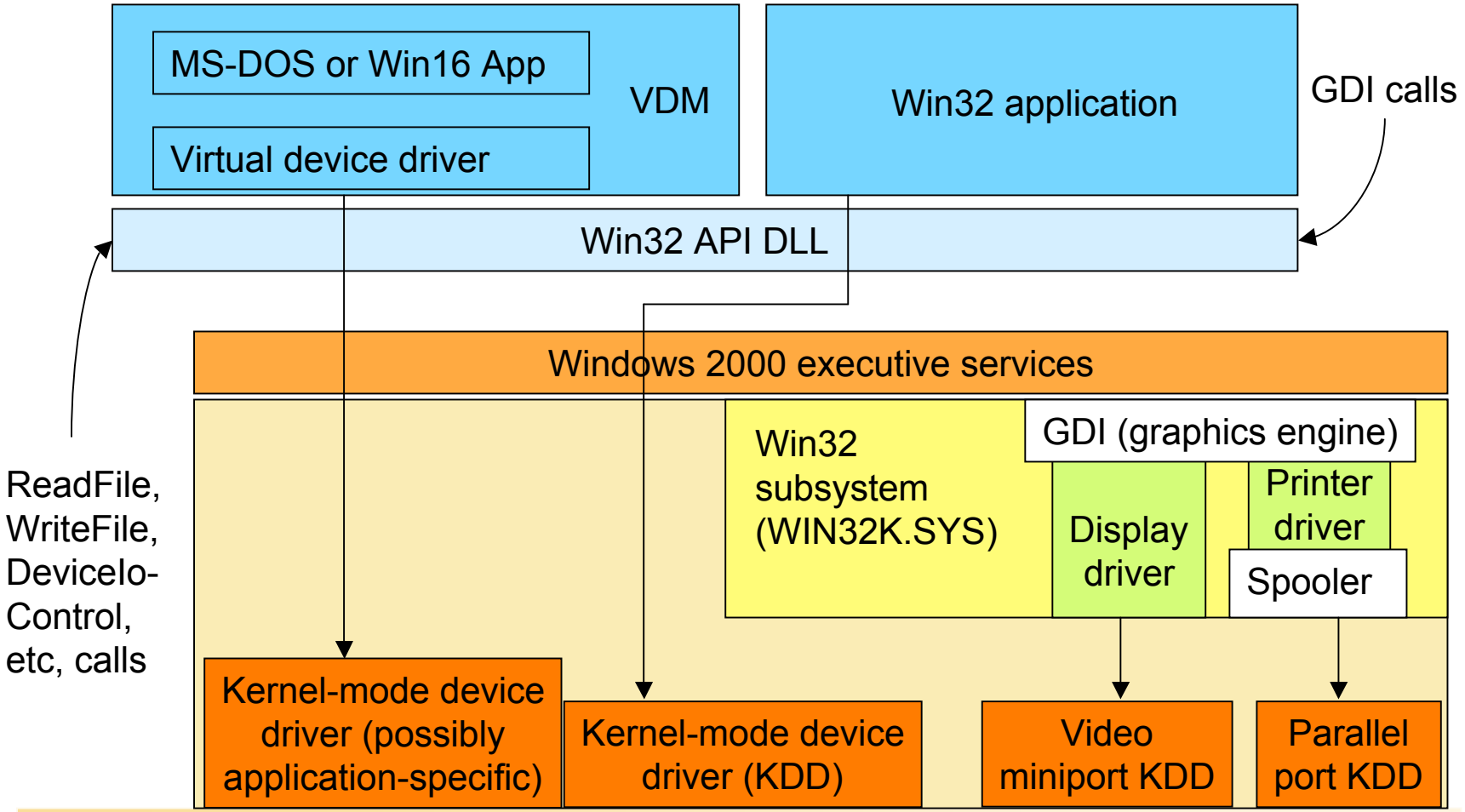
Control flow for an I/O operation



Advanced I/O Functions

- Fast I/O
  - Bypass generation of IRPs
  - Go directly to file system driver or cache manager to complete I/O
- Mapped File I/O and File Caching
  - Available through Win32 CreateFileMapping() / MapViewOfFile() func.
  - Used by OS for file caching and image activation
  - Used by file systems via cache manager to improve performance
- Scatter/Gather I/O
  - Win32 functions ReadFileScatter()/WriteFileScatter()
  - Read/write multiple buffers with a single system call
  - File must be opened for non-cached, asynchronous I/O; buffers must be page-aligned

Device Drivers



Types of Device Drivers (kernel-mode)

- File system drivers
  - Satisfy I/O requests to files by issuing requests to mass storage or network device drivers
- Windows 2000 drivers
  - Drivers for mass storage devices, protocol stacks, network adaptors
  - Integrate with Windows 2000 power manager and PnP manager
- Legacy drivers
  - Written for Windows NT, run unchanged on Windows 2000
  - PnP/power management not supported
- Win32 subsystem *display drivers*
  - Translate device-independent GDI requests into device-specific req.
  - Interaction with video miniport driver

Types of Device Drivers (kernel-mode) (contd.)

Windows Driver Model (WDM) drivers

Implemented on Windows 2000/98/ME, PnP & power management

- Bus drivers
  - Manage a logical or physical bus (PCMCIA, PCI, USB, ISA, FireWire)
  - Responsible for device detection and powering the bus
- Function drivers
  - Manage a particular type of device (bus drivers use PnP manager to announce presence of devices to function drivers)
  - Export device‘s operational interface to OS
- Filter drivers
  - Augment or change behavior of a device or another driver

Types of Device Drivers (user-mode)

- *Virtual device drivers* (VDDs):
  - Used to emulate 16-bit MS-DOS applications
  - Translate MS-DOS references to I/O ports into native Win32 I/O func.
- Win32 subsystem *printer drivers*
  - Translate device-independent GDI requests into device-specific req.
  - Forward commands to kernel-mode drivers

Documentation: Device Driver Kit (DDK)

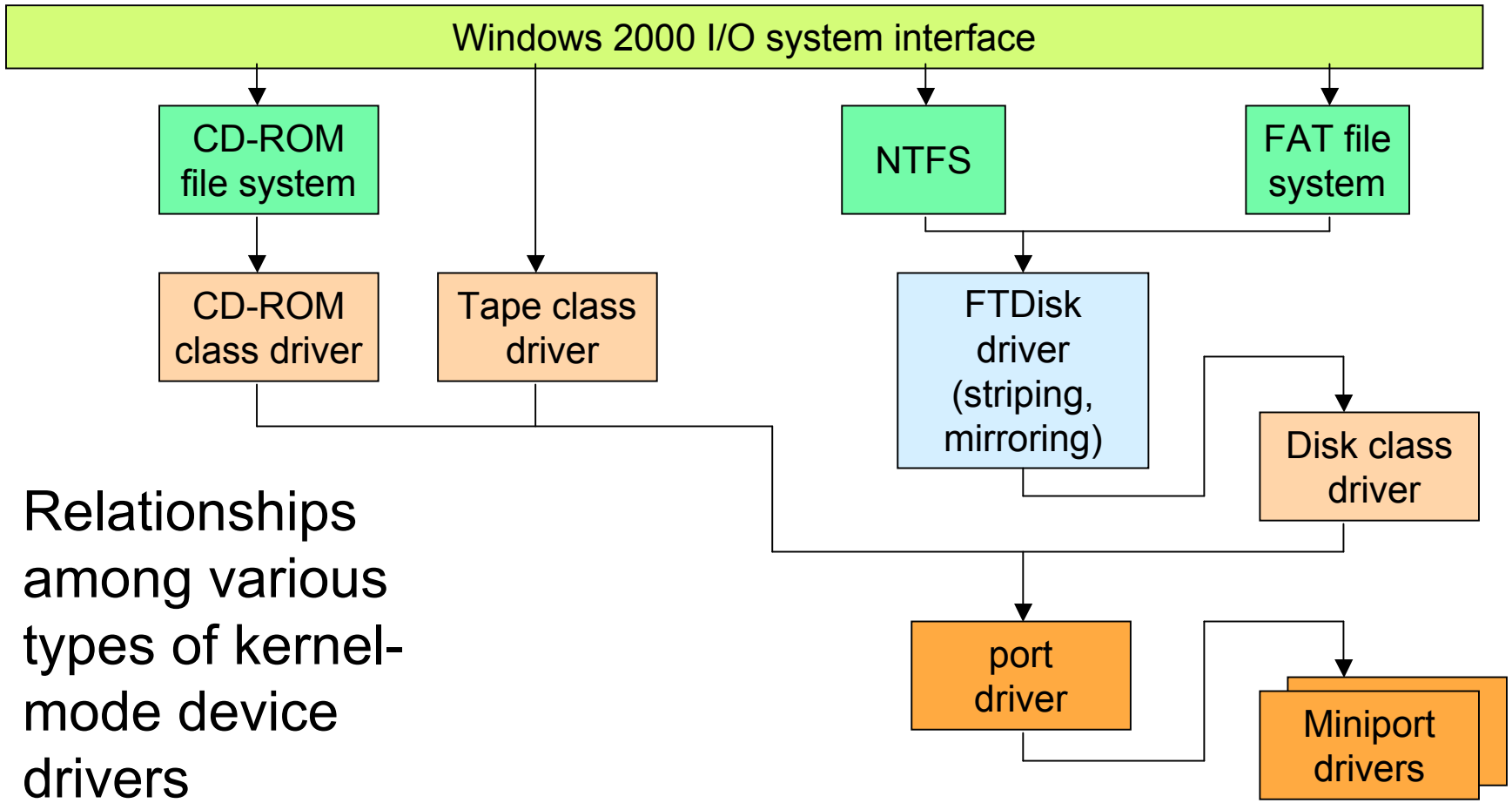
- *Kernel mode drivers* are the only type of driver that can directly control and access hardware devices

Types of Kernel-mode Drivers (another categorization)

Support for a device might be split among low-level *hardware device drivers*

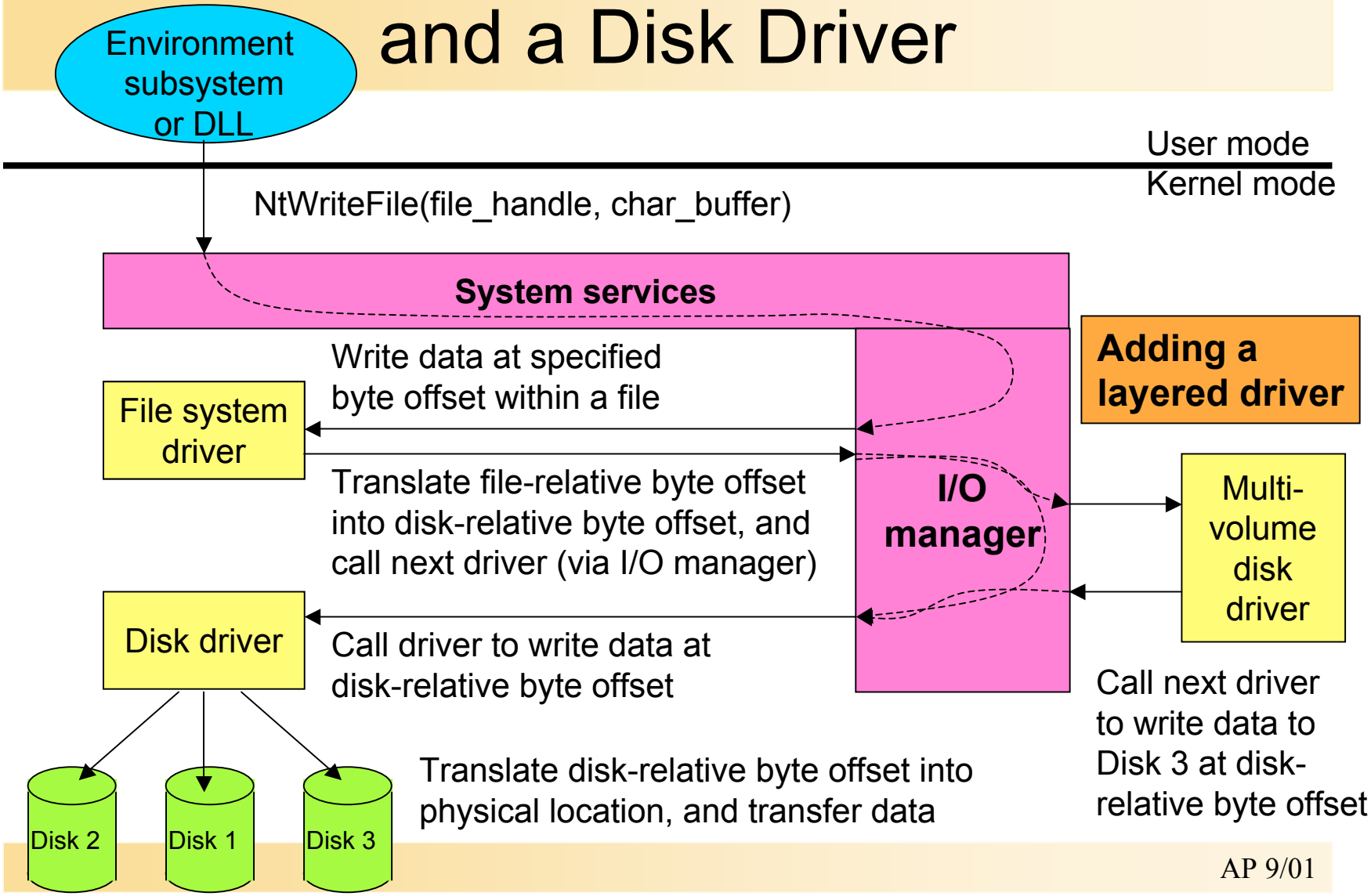
- *Class drivers*
  - Implement I/O processing for a particular class of devices (disk, tape, CD-ROM)
- *Port drivers*
  - Implement I/O processing specific to the type of I/O port (SCSI,...)
- *Miniport drivers*
  - Map generic I/O request to a port type into an adapter type (a specific SCSI adapter,...)

Driver Structure

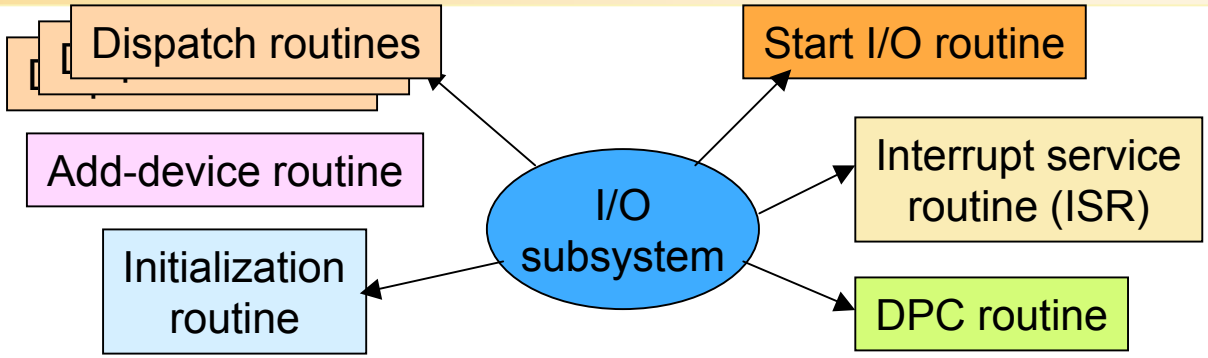


Relationships among various types of kernel-mode device drivers

Layering a File System Driver and a Disk Driver



Structure of a Driver



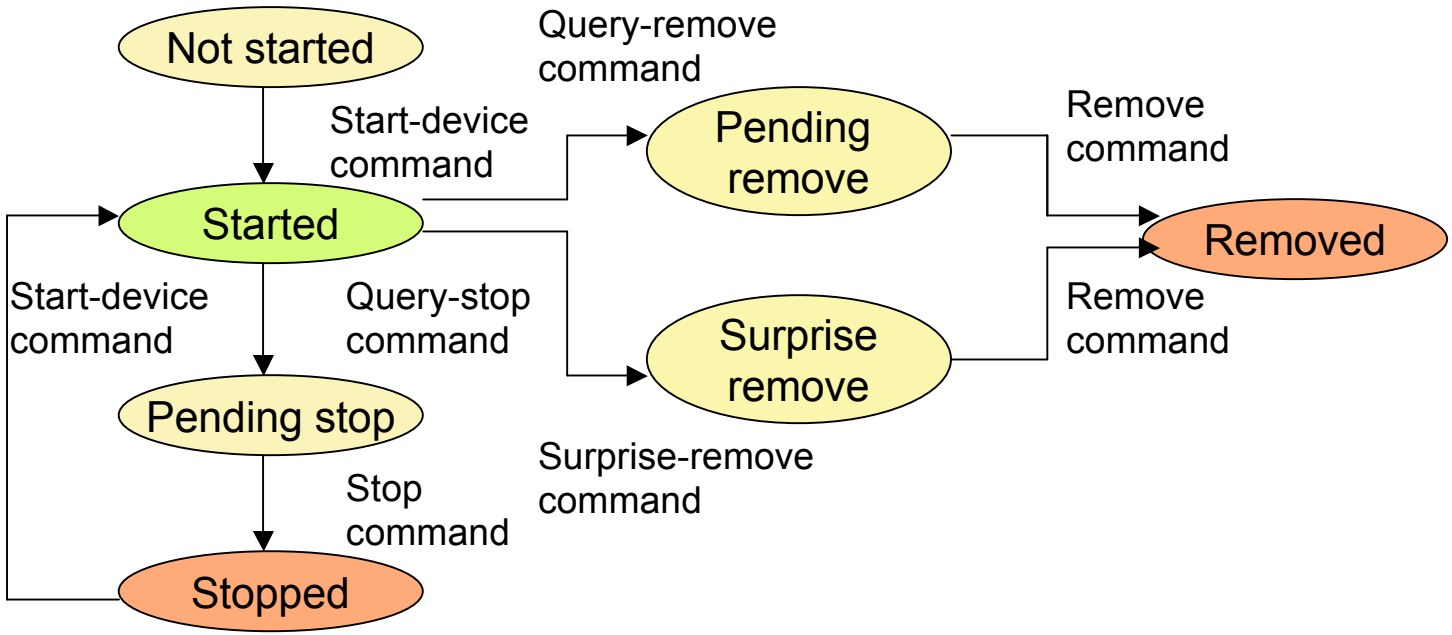
- I/O manager executes initialization routine when loading a driver
- PnP manager calls add-device routine on device detection
- Dispatch routines: open(), close(), read(), write()
- Start I/O routine initiates transfer from/to a device
- ISR runs in response to interrupt; schedules DPC
- DPC routine performs actual work of handling interrupt; starts next queued I/O operation on device

Other components of device drivers

- Completion routines
  - A layered driver may have completion routines that will notify it when a lower-level driver finishes processing an IRP (**I**/O **R**equest **P**acket)
- Cancel I/O routine
- Unload routine
  - Releases system resources
- System shutdown notification routine
- Error-logging routines
  - Notify I/O manager to write record to error log file (e.g., bad disk block)
- Win2000: Windows Driver Model (WDM)
  - Plug & Play support
  - Source compatible between Win98/ME and Win2000

Plug and Play (PnP)

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



Device Plug and Play state transitions

Power Manager

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

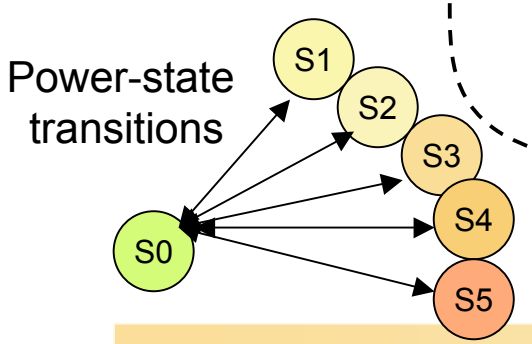
State	Power Consumption	Software Resumption	HW Latency
S0 (fully on)	Maximum	Not applicable	None
S1 (sleeping)	Less than S0, more than S2	System resumes where it left off (returns to S0)	Less than 2 sec.
S2 (sleeping)	Less than S1, more than S3	System resumes where it left off (returns to S0)	2 or more sec.
S3 (sleeping)	Less than S2, processor is off	System resumes where it left off (returns to S0)	Same as S2
S4 (sleeping)	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

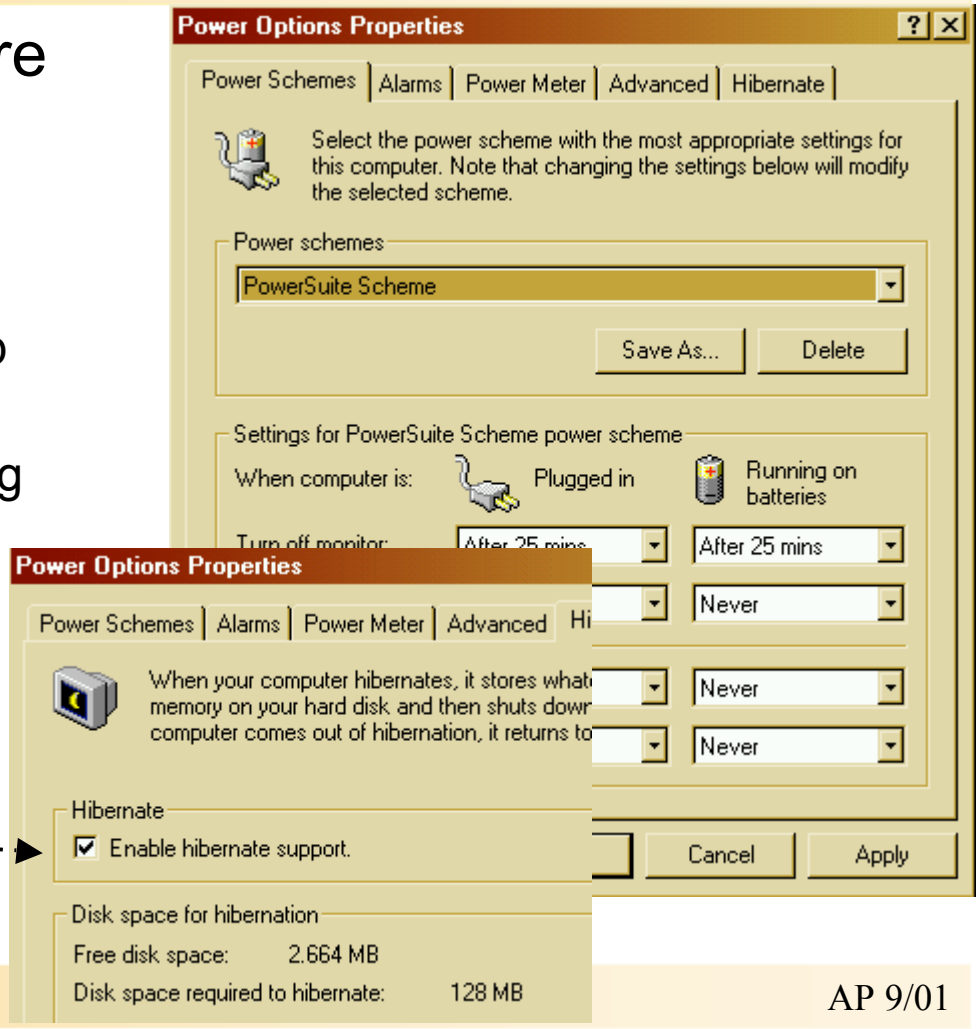
System Power-State Definitions

Power Manager Operation

Power-state transitions are triggered by:

- System activity level
- System battery level
- Shutdown, hibernate, or sleep requests from application
- User actions, such as pressing the power button
- Control Panel power settings





Synchronization

Drivers must synchronize accesses to global driver data:

- Execution of a driver can be preempted by higher-prio threads; quantum may expire; interrupts
- Windows 2000 can run driver code simultaneously on multiprocessor systems

W2K kernel provides special synchronization routines:

- Raise IRQL on single processor machines
- Use spinlocks on multiprocessors

Data structures – File objects

- Memory-based representation of physical resource

Attribute	Purpose
Filename	Identifies physical file that the file object refers to
Byte offset	Current location in the file (for synchronous I/O)
Share mode	Exclusive vs. Shared
Open mode	Synch. vs. asynch, cached vs. non-cached, sequential, random
Pointer to device object	Type of device on which the file resides
Pointer to volume parameter block	Volume/partition, that the file resides on
Pointer to section object pointers	Indicates a root structure that describes a mapped file
Pointer to private cache map	Identifies which parts of the file are cached by the cache manager and where they reside in the cache

Driver objects and device objects

- When a thread opens a handle to a file object, the I/O manager must determine which driver(s) it should call

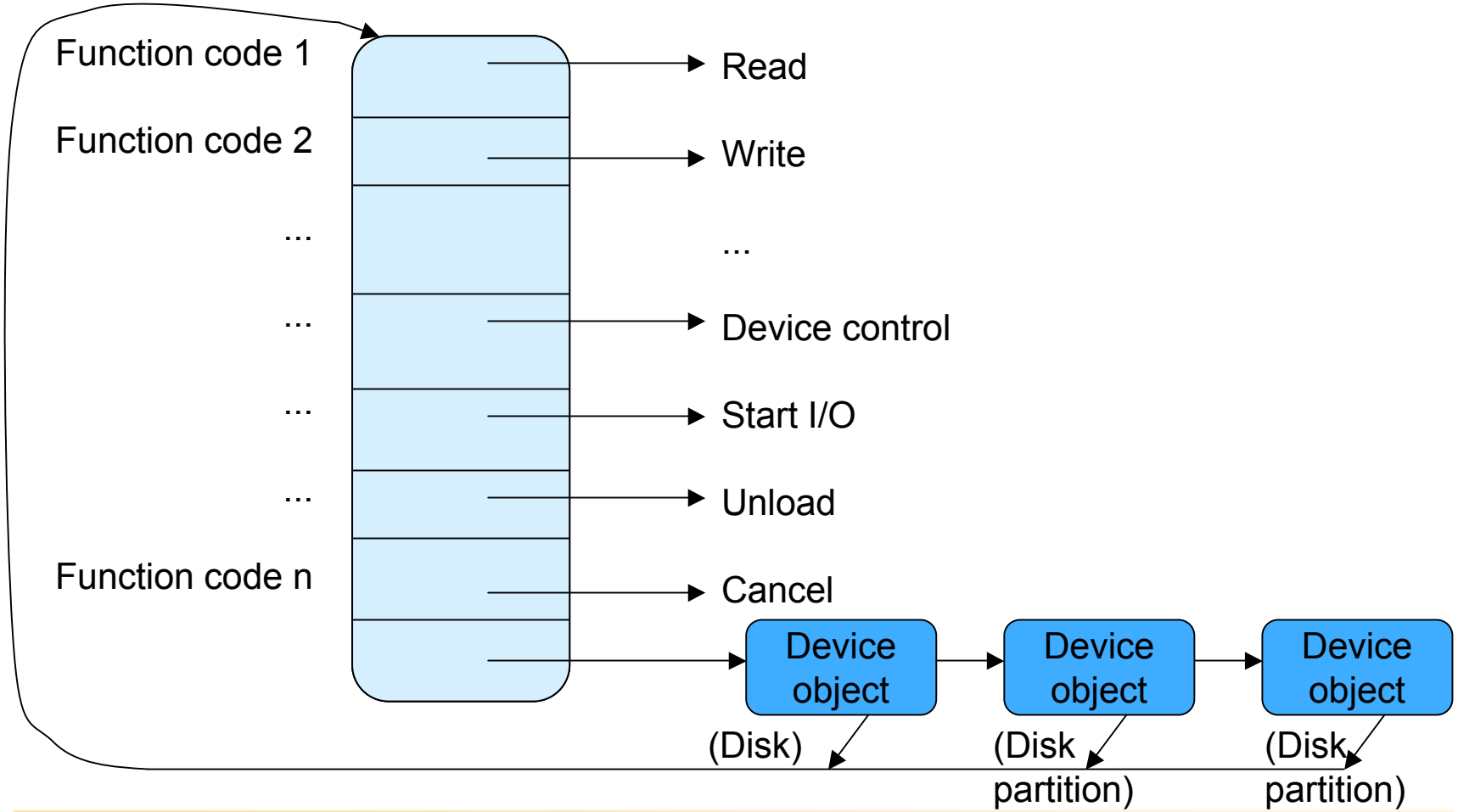
Driver object:

- Represents individual driver in the system
- Records for I/O manager the address of each of the drivers dispatch routines (entry points)

Device object:

- Represents physical/logical/virtual device on the system
- Describes characteristics such as buffer alignment, location of device queue for incoming I/O request packets (IRPs)

The driver object

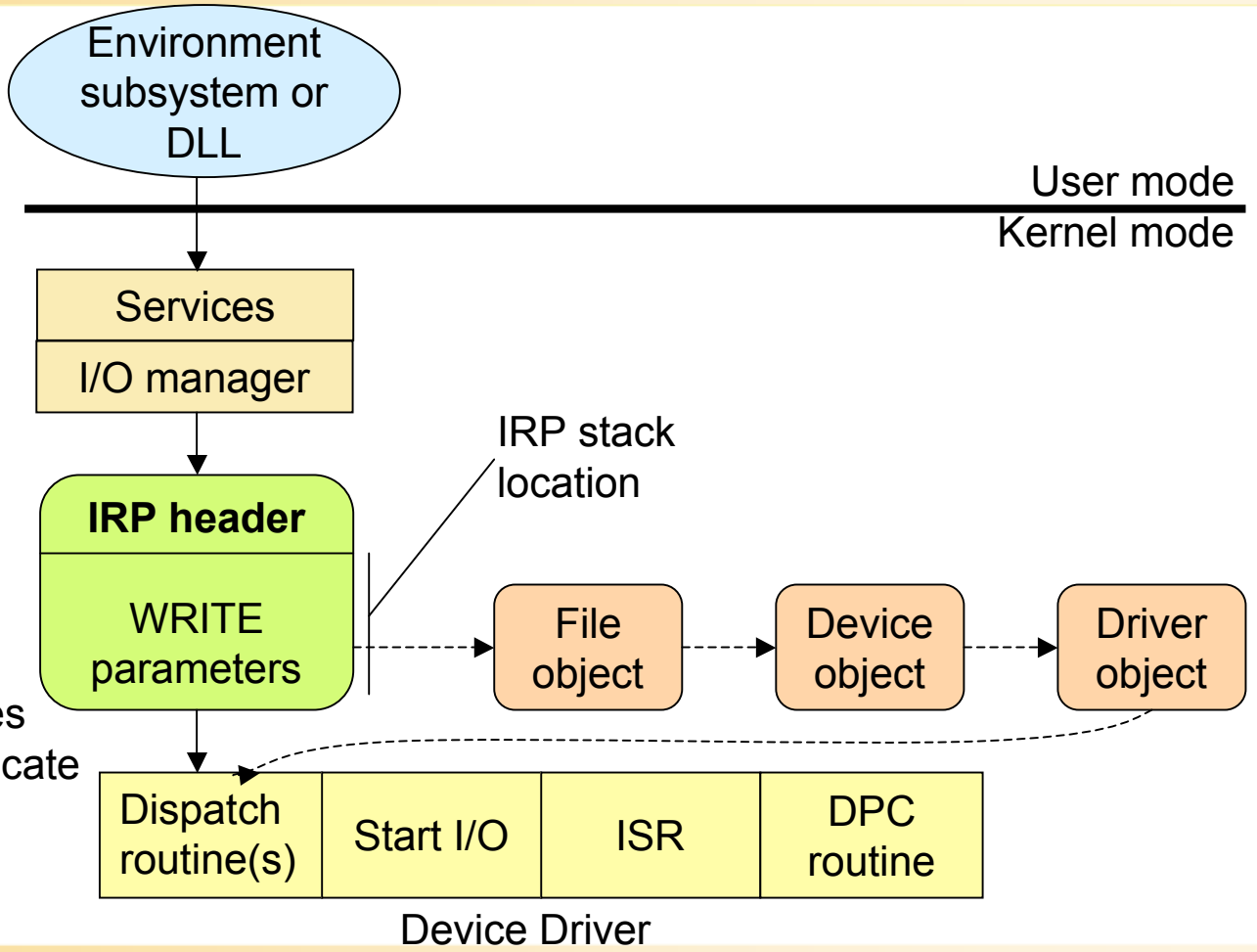


I/O Request Packet

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 – synch. 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 int.
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 int.
- 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

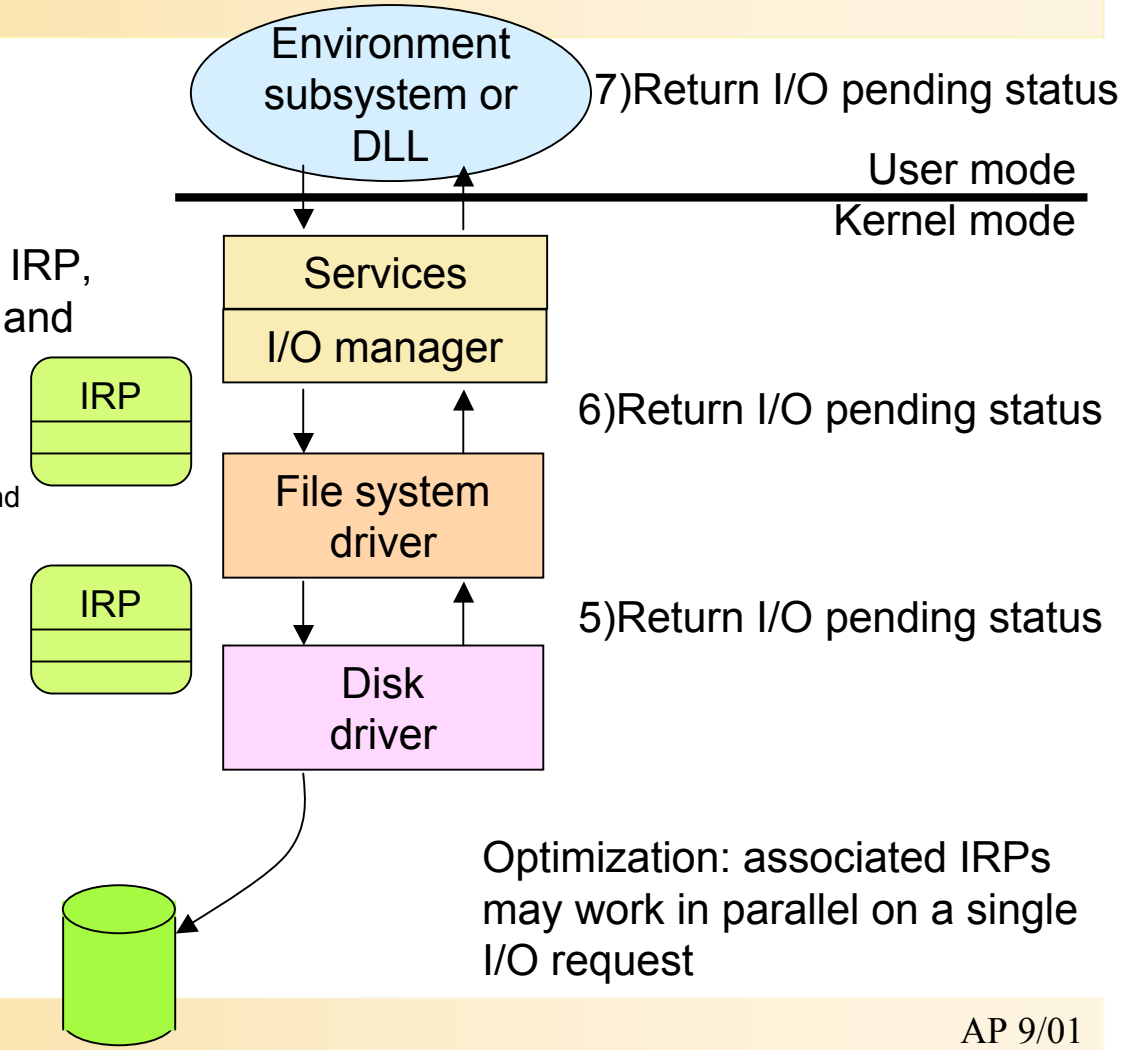
Layered Drivers

1)Call I/O service

2)The I/O manager creates an IRP, initializes first stack location and calls file system driver

3)File system driver fills in a 2nd IRP stack location and calls the disk driver

4)Send IRP data to device (or queue IRP), and return



Optimization: associated IRPs may work in parallel on a single I/O request