

Windows Kernel Technologies

潘爱民

2010-9-11

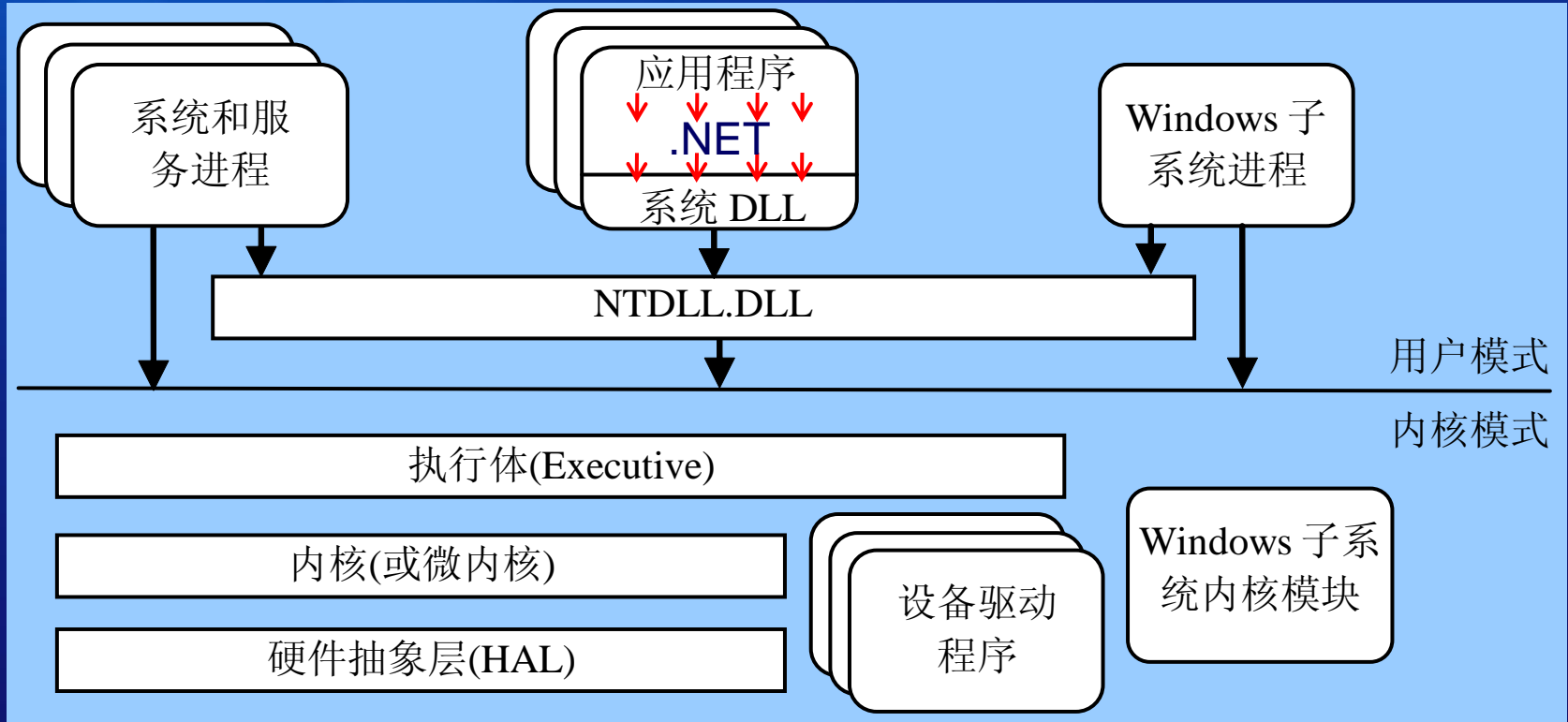
Outline

- Windows Architecture Overview
- System Trap
- Windows I/O Model & Device Drivers
- Windows Subsystem

NT Timeline: first 20 years

<u>2/1989</u>	<u>Design and Coding Begins</u>
7/1993	NT 3.1
9/1994	NT 3.5
5/1995	NT 3.51
7/1996	NT 4.0
12/1999	NT 5.0 Windows 2000
8/2001	NT 5.1 Windows XP
3/2003	NT 5.2 Windows Server 2003
8/2004	NT 5.2 Windows XP SP2
4/2005	NT 5.2 Windows XP 64 Bit Edition (WS03SP1)
10/2006	NT 6.0 Windows Vista (client)
2/2008	NT 6.0 Windows Server 2008 (Vista SP1)
10/2009	NT 6.1 Win7 & Server 2008 R2

Windows Architecture

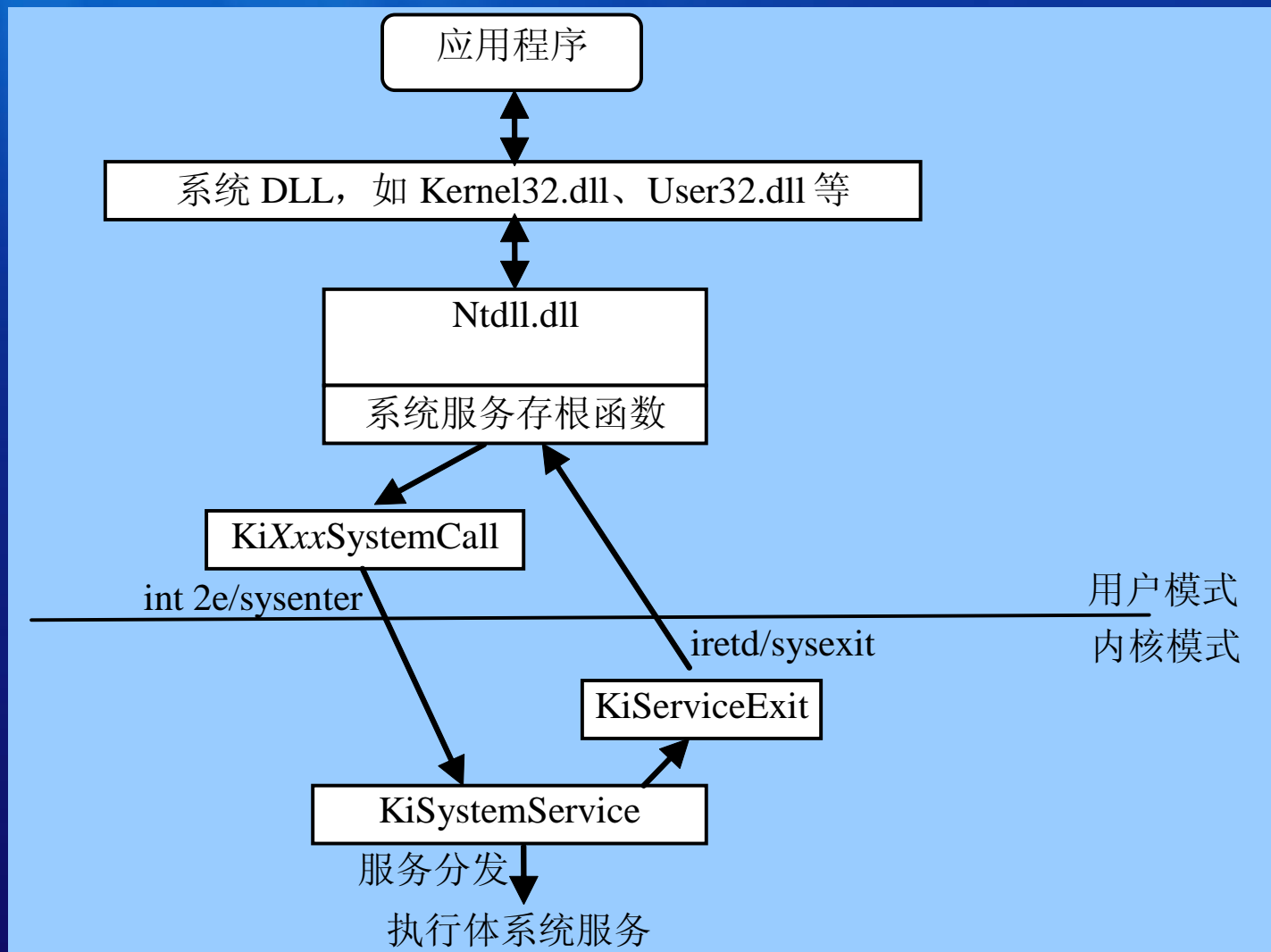


About Windows APIs

- Provided by Windows Subsystem DLLs
- Three types:
 - ① No kernel-mode interactions are needed. (e.g. PtInRect, IsRectEmpty, GetCurrentProcess);
 - ② Call into kernel once or more (e.g. CreateFile, PostMessage and BitBlt);
 - ③ Communications with the Windows subsystem process (csrss) are needed. (e.g. CreateProcess)



Call into Kernel



Example System Service: NtCreateFile

- The Call stack of creating a file in a user thread

```
f5484c94 808e3375 nt!IopCreateFile
f5484cf0 808e50ec nt!IoCreateFile
f5484d30 80882a2c nt!NtCreateFile
007deed8 7c8211f4 ntdll!KiFastSystemCallRet
007deedc 76cf2707 ntdll!ZwCreateFile+0xc
007def54 76cf276b iphlpapi!OpenTCPDriver+0xad
007def64 76cf278f iphlpapi!CheckTcpipState+0x62
007def98 76cf299a iphlpapi!GetIpStatsFromStack+0xd
007df00c 76cf32de iphlpapi!GetInterfaceInfo+0x38
007df028 76cf3230 iphlpapi!GetAdapterNameToIndexInfo+0x1e
007df060 76cf6a6b iphlpapi!GetAdapterInfo+0x18
007df0b4 4e7fdf36 iphlpapi!GetAdapterInfoEx+0x1c
007df504 4e7fe2f9 WINHTTP!CIPConfig::GetAdapterListOnNT5+0x6e
007dfac4 4e7fe938 WINHTTP!CIPConfig::GetAdapterList+0x58
007dfad0 4e7fd3ee WINHTTP!CIPConfig::CIPConfig+0x23
007dfaf0 4e7cd1cc WINHTTP!DetectAutoProxyUrl+0x26
007dfb28 4e7cd9b5 WINHTTP!CAutoProxy::DetectAutoProxyUrl+0xa4
007dfb58 4e7cdafb WINHTTP!CAutoProxy::GetProxyForURL+0x33
007dfb6c 4e7cdcd8 WINHTTP!InProcGetProxyForUrl+0x20
```

内核模式
用户模式

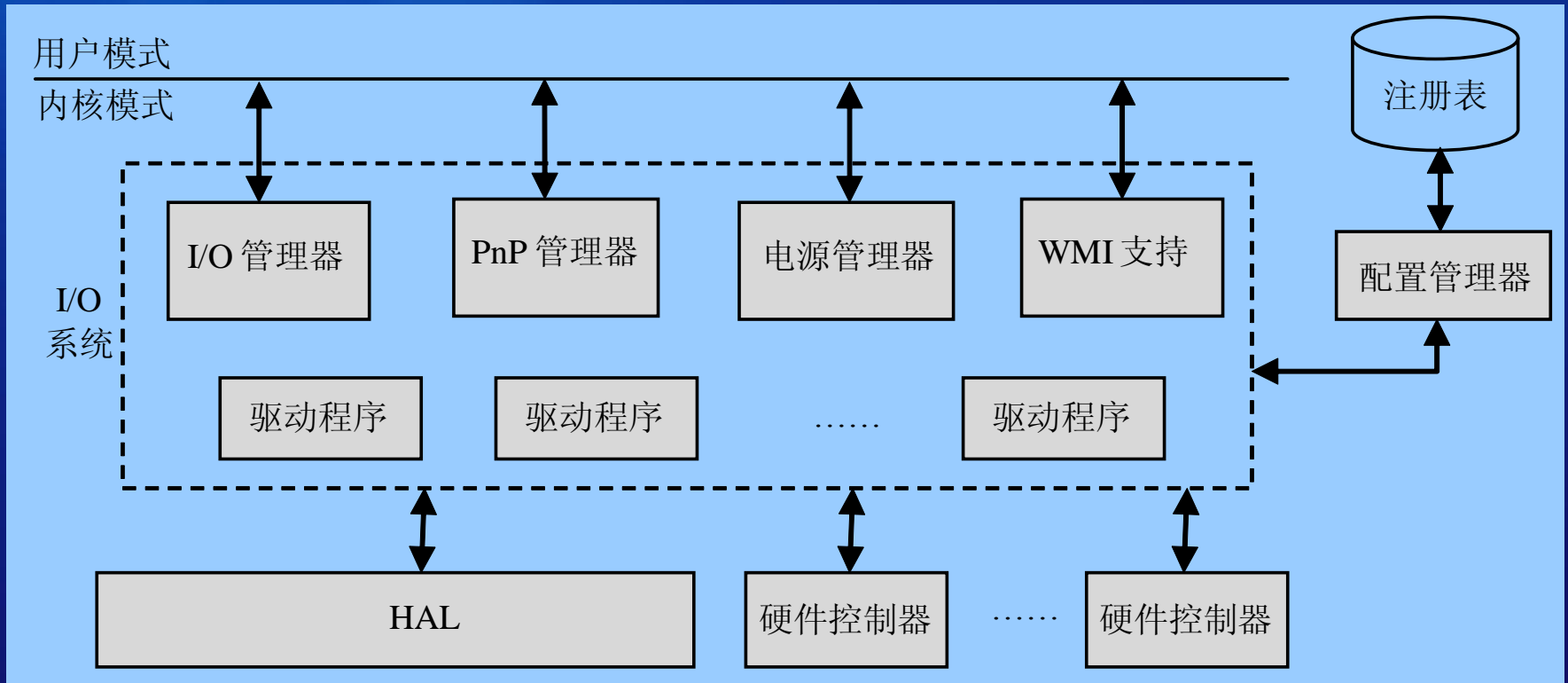
Example Call Stack with .NET

0012f2f4 77d19418 ntdll!KiFastSystemCallRet	用户模式
0012f38c 7b1d8997 user32!NtUserWaitMessage+0xc	
0012f3e4 7b1d87e1 System_Windows_Forms_ni+0x208997	
0012f414 7b6eddc6 System_Windows_Forms_ni+0x2087e1	.NET Runtime
0012f460 79e71b4c System_Windows_Forms_ni+0x71ddc6	Common Language
0012f470 79e821f9 mscorwks!CallDescrWorker+0x33	Runtime
0012f4f0 79e96571 mscorwks!CallDescrWorkerWithHandler+0xa3	
0012f634 79e965a4 mscorwks!MethodDesc::CallDescr+0x19c	
0012f650 79e965c2 mscorwks!MethodDesc::CallTargetWorker+0x1f	
0012f668 79f0788d mscorwks!MethodDescCallSite::Call+0x1a	
0012f7cc 79f077ad mscorwks!ClassLoader::RunMain+0x223	
0012fa34 79f07cfd mscorwks!Assembly::ExecuteMainMethod+0xa6	
0012ff04 79f07ee7 mscorwks!SystemDomain::ExecuteMainMethod+0x456	
0012ff54 79f07e17 mscorwks!ExecuteEXE+0x59	
0012ff9c 7900b77b mscorwks!_CorExeMain+0x15c	
0012ffac 7900b73d mscoree!_CorExeMain+0x2e	.NET Runtime
0012ffb8 79004de3 mscoree!ShellShim__CorExeMain+0x29	Execution Engine
0012ffc0 7c817077 mscoree!_CorExeMain_Exported+0x8	
0012fff0 00000000 kernel32!BaseProcessStart+0x23	

Windows I/O Model

- Asynchronous, Packet-based, Extensible
- Device discovery supports plug-and-play (PnP)
 - volumes automatically detected and mounted
 - power management support (ACPI)
- Drivers attach to per device driver stacks
 - Drivers can filter actions of other drivers in each stack
- Integrated kernel support
 - memory manager provides DMA support
 - HAL provides device access, PnP manages device resources
 - Cache manager provides file-level caching via MM file-mapping
- Multiple I/O completion mechanisms:
 - synchronous
 - update user-mode memory status
 - signal events
 - callbacks within initiating thread
 - I/O Completion Port

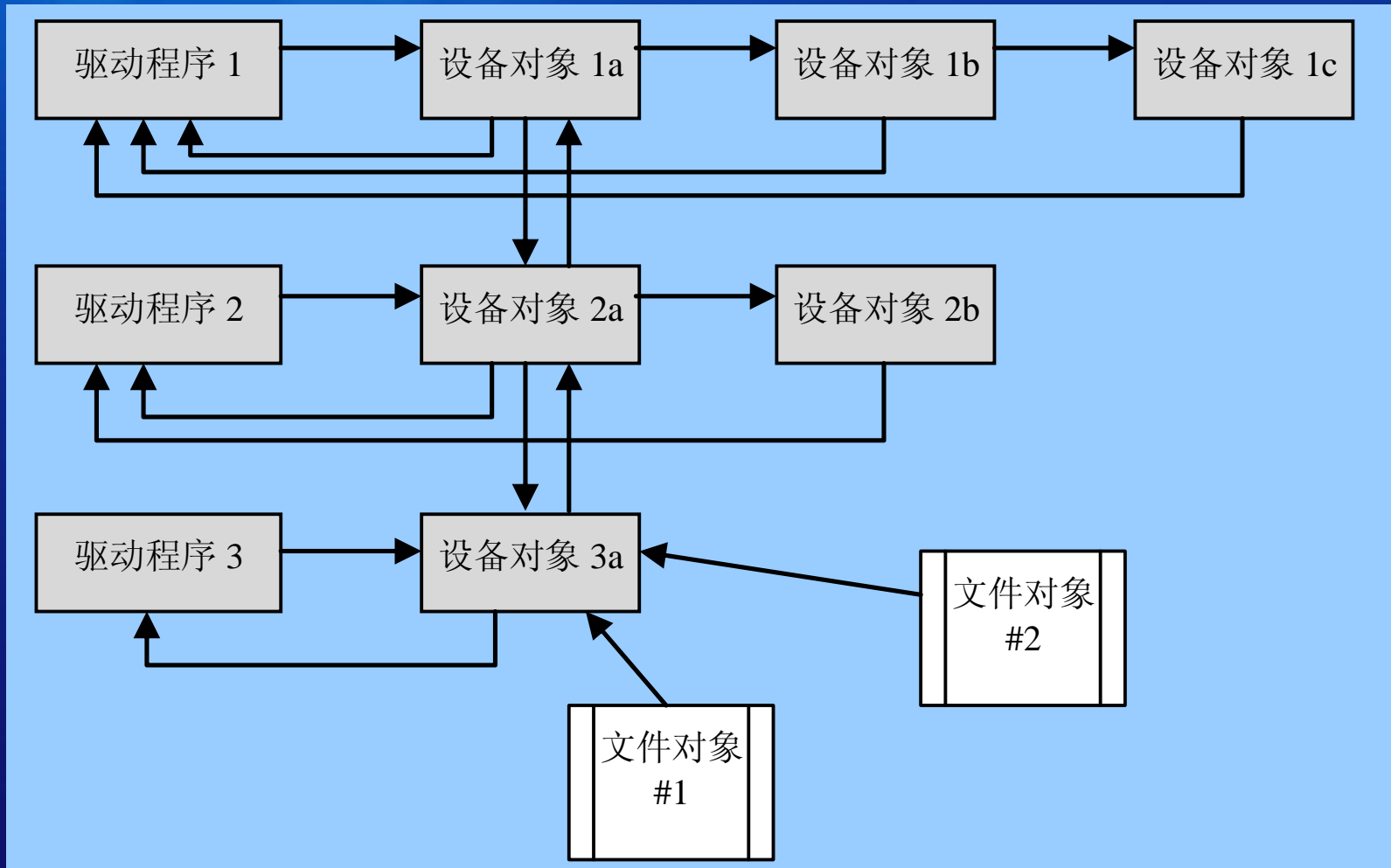
I/O Architecture



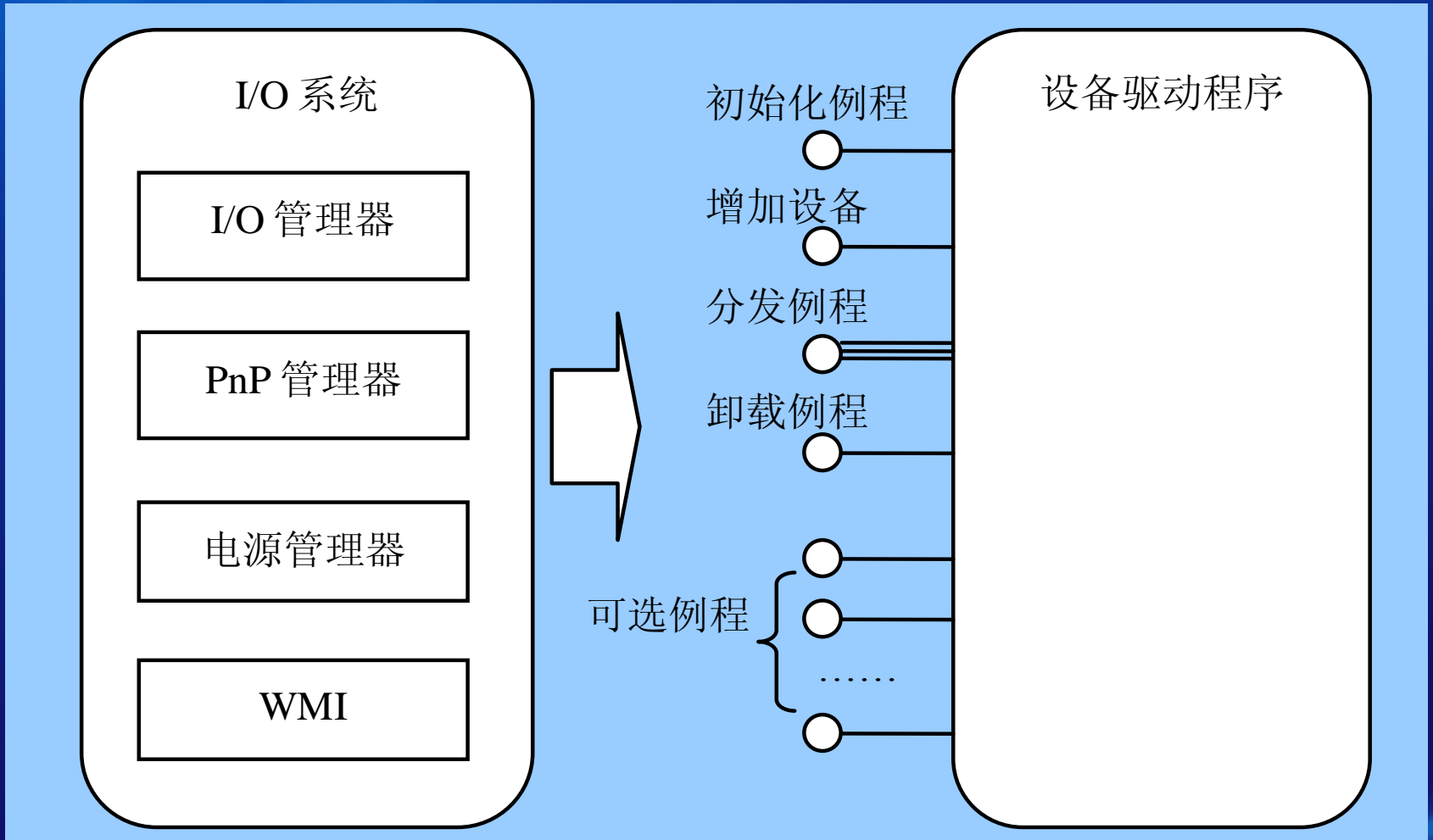
I/O Manager Objects

- Driver objects represent loaded drivers.
 - NtLoadDriver/NtUnloadDriver
 - IoCreateDriver
- Drivers create device objects to represent devices.
 - IoCreateDevice
- All I/O requests are made to device objects.
- File objects represent open instances of device objects.

Object Relationships



Structure of Windows Drivers



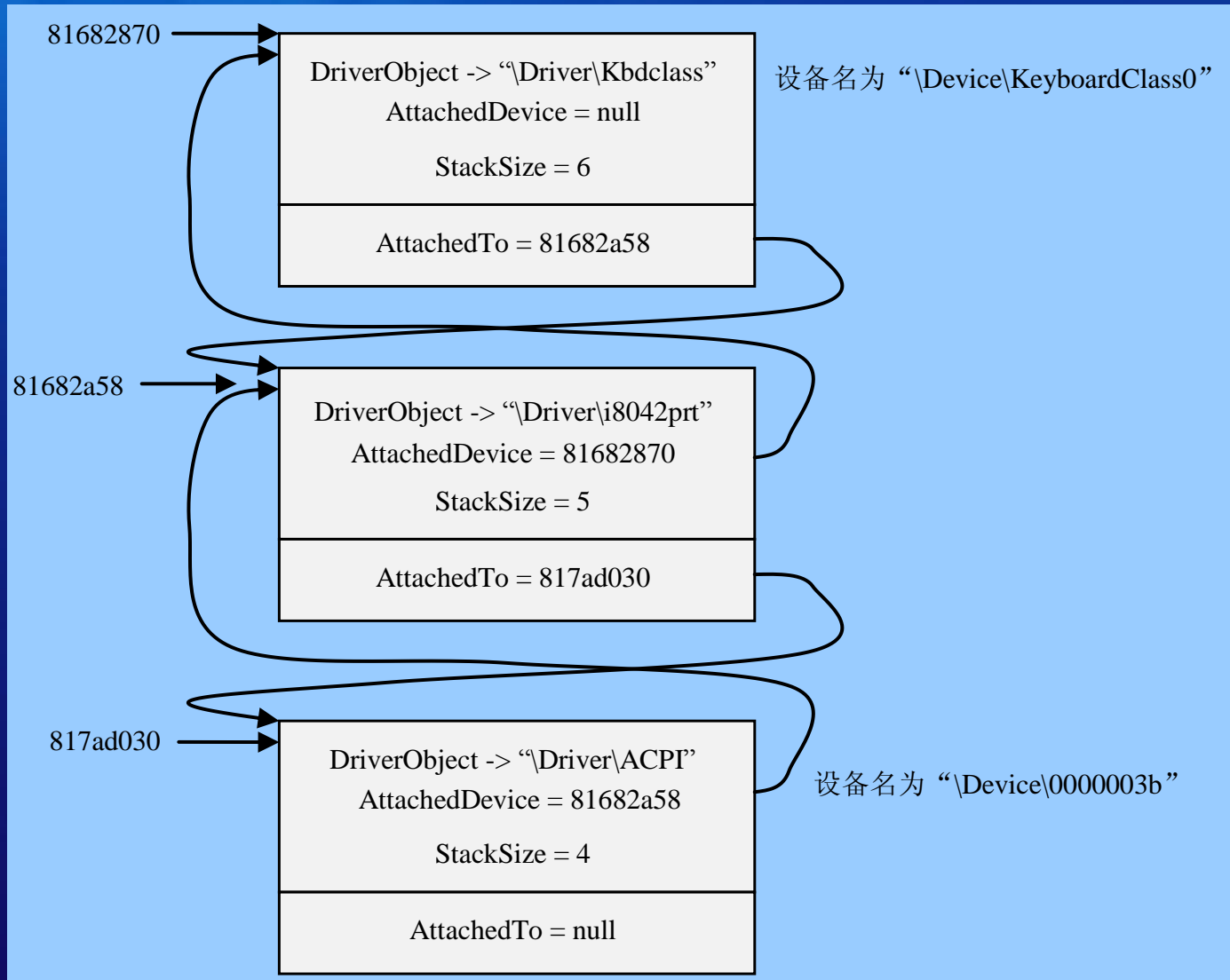
Loading Device Drivers

- Drivers can be loaded by,
 - The boot loader at boot time.
 - The I/O manager at system initialization.
 - The service control manager or PnP manager.
- Driver details are obtained from the registry.
- Driver object is created and DriverEntry for the driver is invoked.
- Drivers provide dispatch routines for various I/O operations. (Create, Read, Write, ...).
- Drivers can optionally provide fast path entry points.

Layering Drivers

- **Device objects** can be attached one on top of another using IoAttachDevice* APIs to create **device stacks**.
- I/O manager sends IRP to the top of the stack.
- Drivers store next lower device object in their private data structure.
- Stack tear down done using IoDetachDevice and IoDeleteDevice.

Device Stack Example



Device Deletion and Driver Unload

- Drivers delete devices using `IoDeleteDevice`.
- Drivers are unloaded by calling `NtUnloadDriver` or by PnP.
- No further opens/attaches allowed after a device is marked for deletion or unload.
- Driver unload function is invoked when all its device objects have no handles/attaches.
- Driver is unloaded when the last reference to driver object goes away.

File Objects

- Also managed by Windows Object Manager, its type is `IoFileObjectType`
 - Applications and drivers “open” devices by name
 - The name is parsed by the Object Manager
- Representation of an open instance of a device object
 - **files on a volume are virtual devices**
- Created by `IoCreateFile` function, which is invoked by `NtCreateFile`
 - When an open succeeds a file handle is added to the process handle table

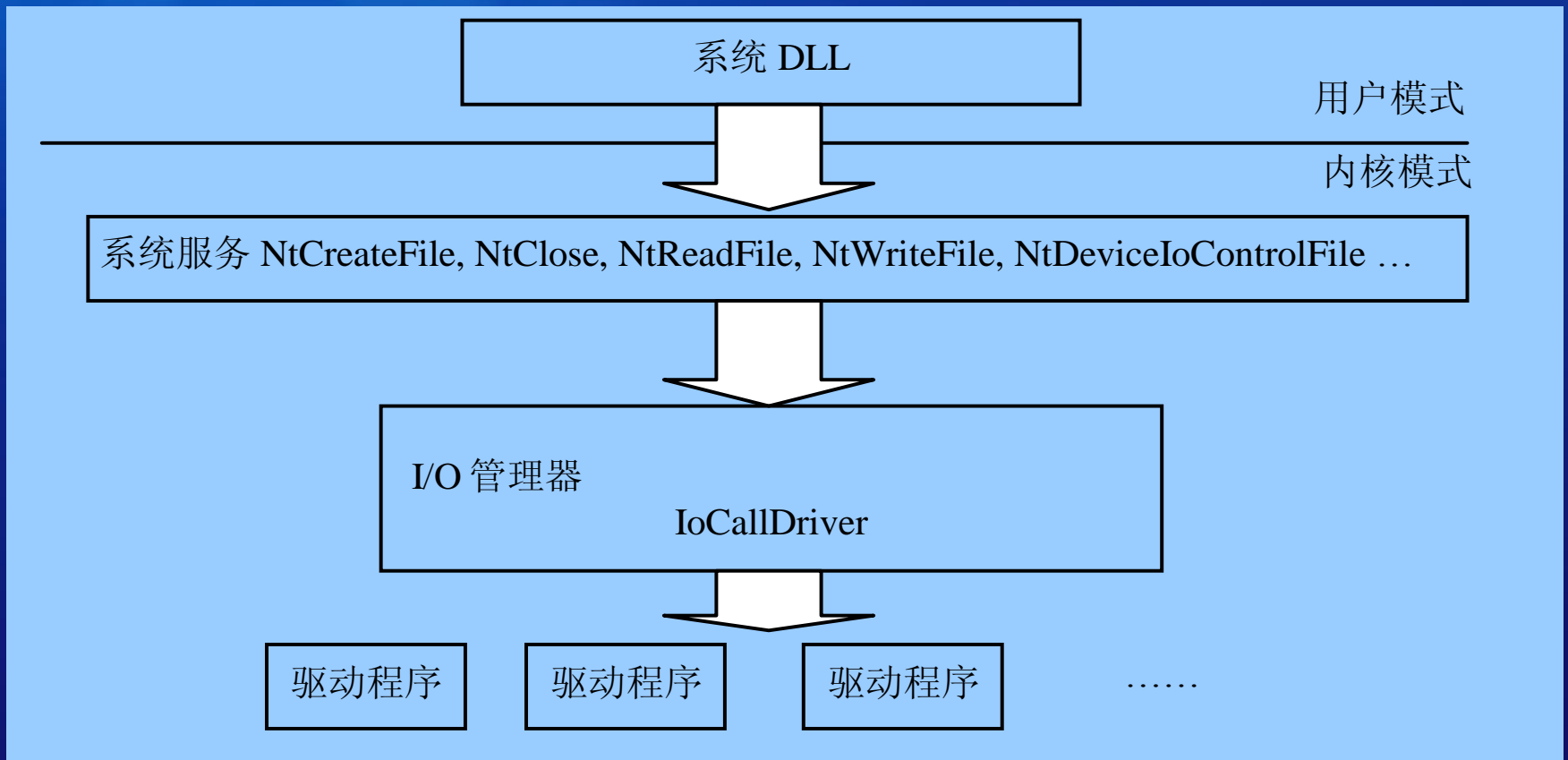
I/O Processing

- IRP
- Issues in I/O processing
 - Fast I/O
 - Buffer management
 - I/O completion
- I/O Completion port

I/O Request Packet (IRP)

- I/O operations are encapsulated in IRPs.
- I/O requests travel down a device stack in an IRP.
- Each driver gets a stack location which contains parameters for that I/O request.
- IRP has major and minor codes to describe I/O operations.
- Major codes include create, read, write, PnP, devioctl, cleanup and close.
- Irps are associated with a thread that made the I/O request.

I/O Requests from Applications



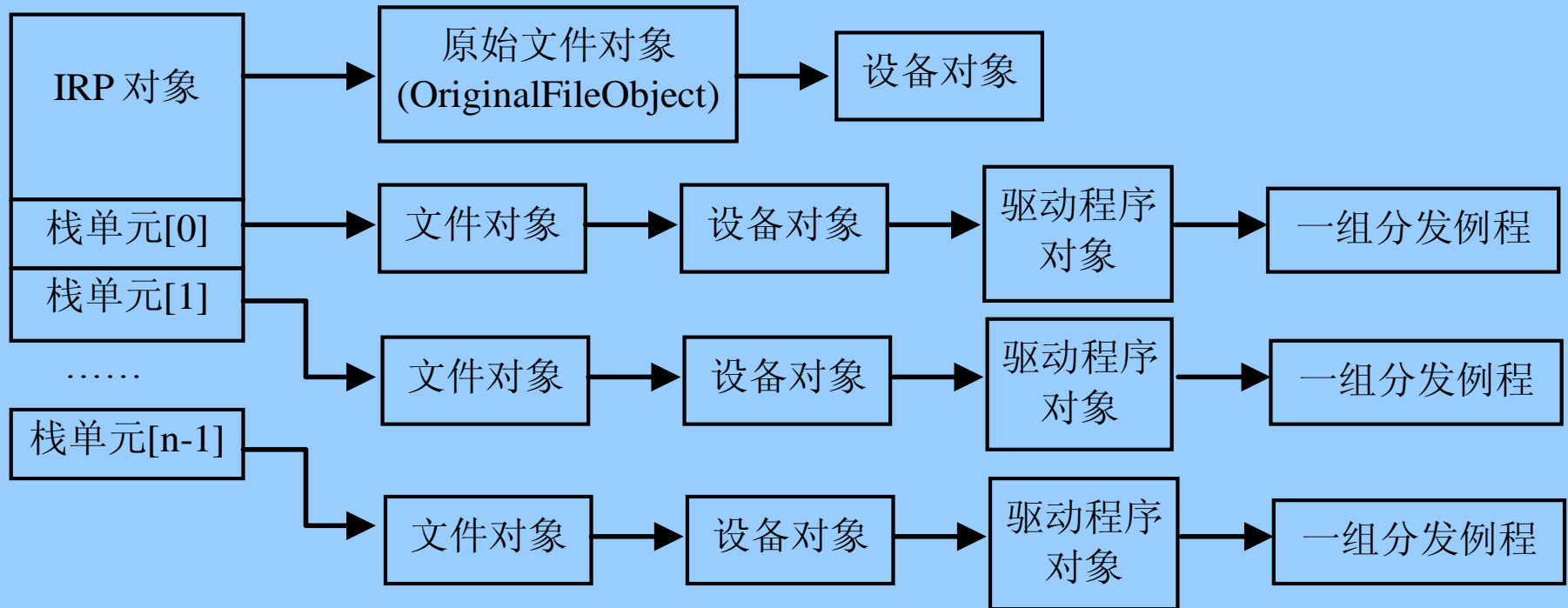
IRP Major Codes

```
#define IRP_MJ_CREATE 0x00
#define IRP_MJ_CREATE_NAMED_PIPE 0x01
#define IRP_MJ_CLOSE 0x02
#define IRP_MJ_READ 0x03
#define IRP_MJ_WRITE 0x04
#define IRP_MJ_QUERY_INFORMATION 0x05
#define IRP_MJ_SET_INFORMATION 0x06
#define IRP_MJ_QUERY_EA 0x07
#define IRP_MJ_SET_EA 0x08
#define IRP_MJ_FLUSH_BUFFERS 0x09
#define IRP_MJ_QUERY_VOLUME_INFORMATION 0x0a
#define IRP_MJ_SET_VOLUME_INFORMATION 0x0b
#define IRP_MJ_DIRECTORY_CONTROL 0x0c
#define IRP_MJ_FILE_SYSTEM_CONTROL 0x0d
#define IRP_MJ_DEVICE_CONTROL 0x0e
#define IRP_MJ_INTERNAL_DEVICE_CONTROL 0x0f
```

IRP Major Codes (cont.)

```
#define IRP_MJ_SHUTDOWN          0x10
#define IRP_MJ_LOCK_CONTROL      0x11
#define IRP_MJ_CLEANUP          0x12
#define IRP_MJ_CREATE_MAILSLOT  0x13
#define IRP_MJ_QUERY_SECURITY    0x14
#define IRP_MJ_SET_SECURITY      0x15
#define IRP_MJ_POWER             0x16
#define IRP_MJ_SYSTEM_CONTROL    0x17
#define IRP_MJ_DEVICE_CHANGE     0x18
#define IRP_MJ_QUERY_QUOTA       0x19
#define IRP_MJ_SET_QUOTA         0x1a
#define IRP_MJ_PNP               0x1b
#define IRP_MJ_PNP_POWER         IRP_MJ_PNP // Obsolete...
#define IRP_MJ_MAXIMUM_FUNCTION  0x1b
```

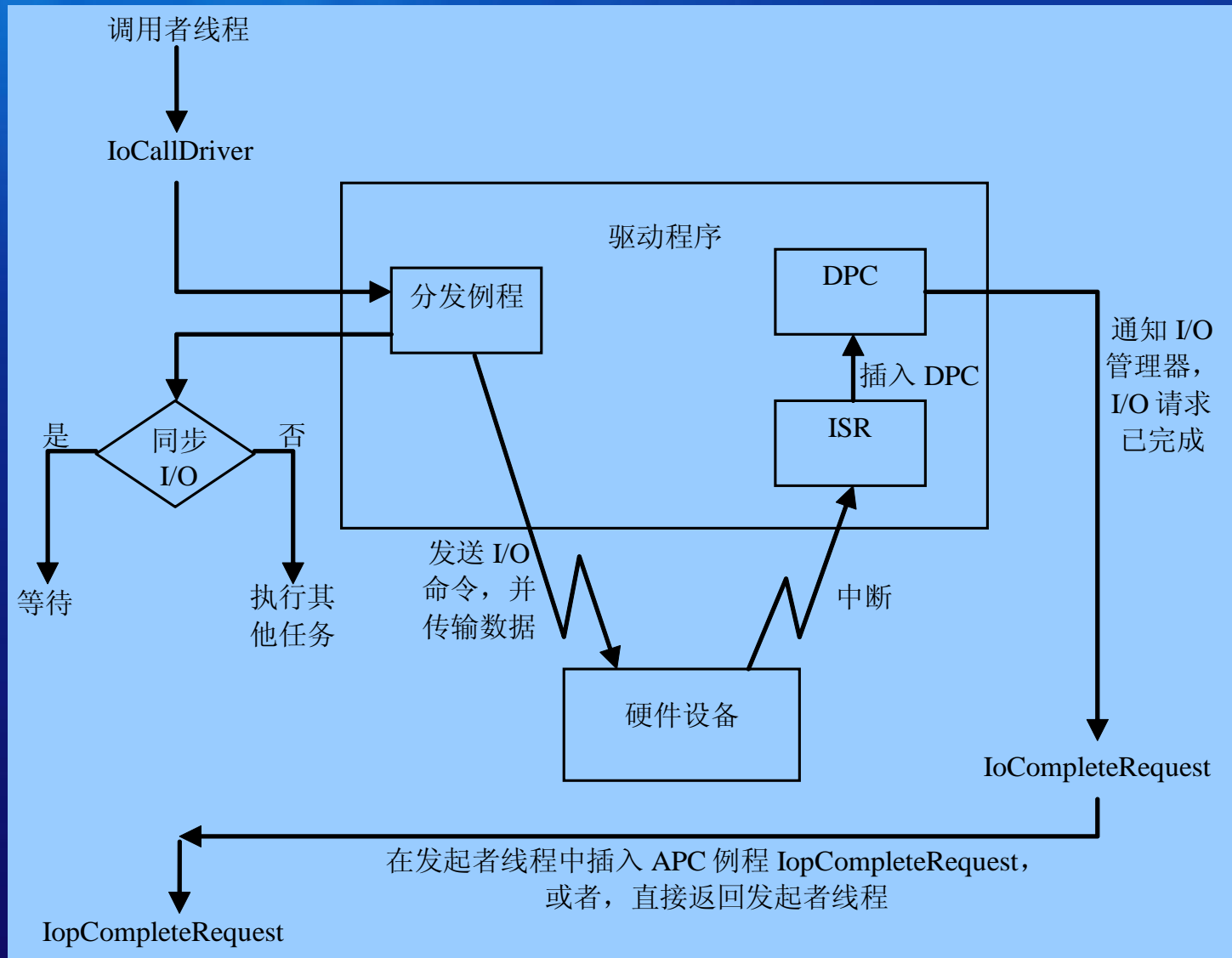
I/O Stack Locations in IRPs



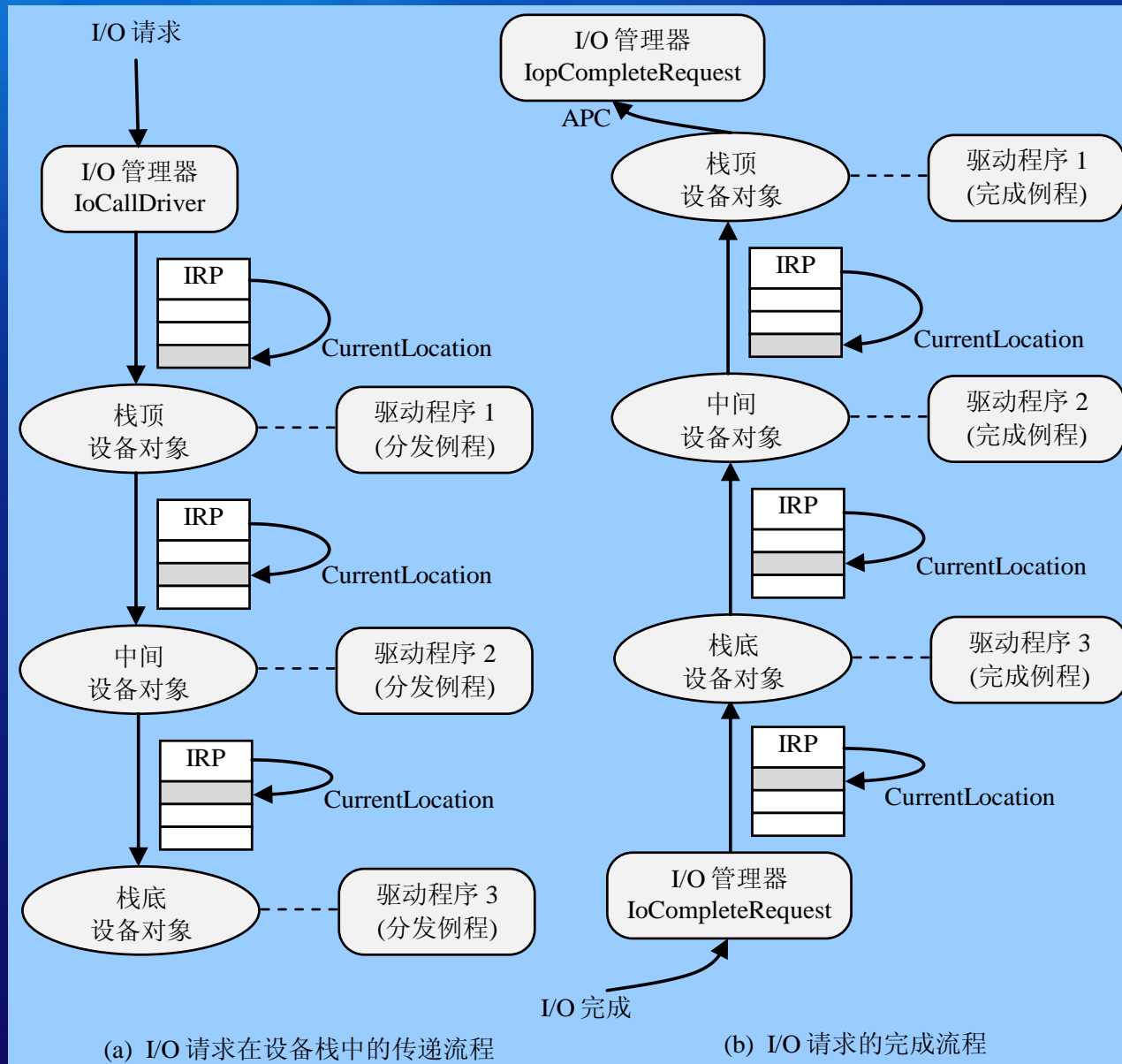
Completing an I/O Request

- 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

Flow of I/O Processing



I/O Processing in a Device Stack



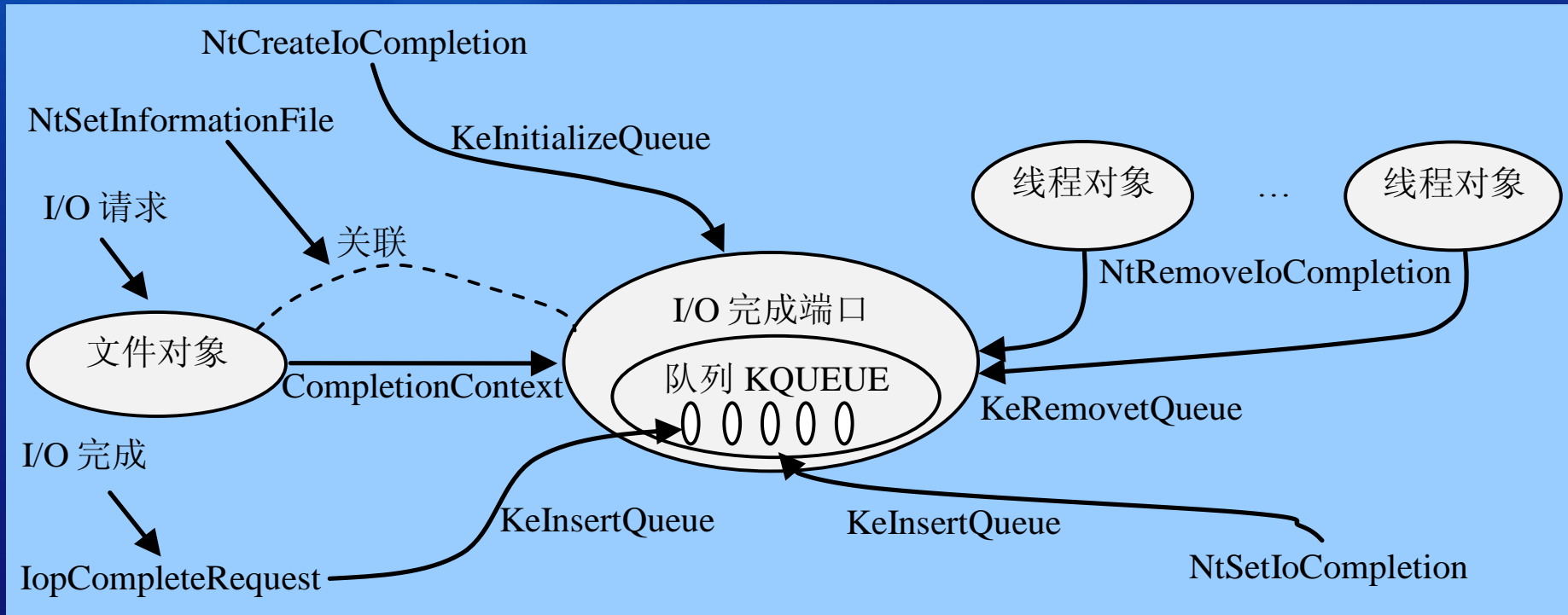
(a) I/O 请求在设备栈中的传递流程

(b) I/O 请求的完成流程

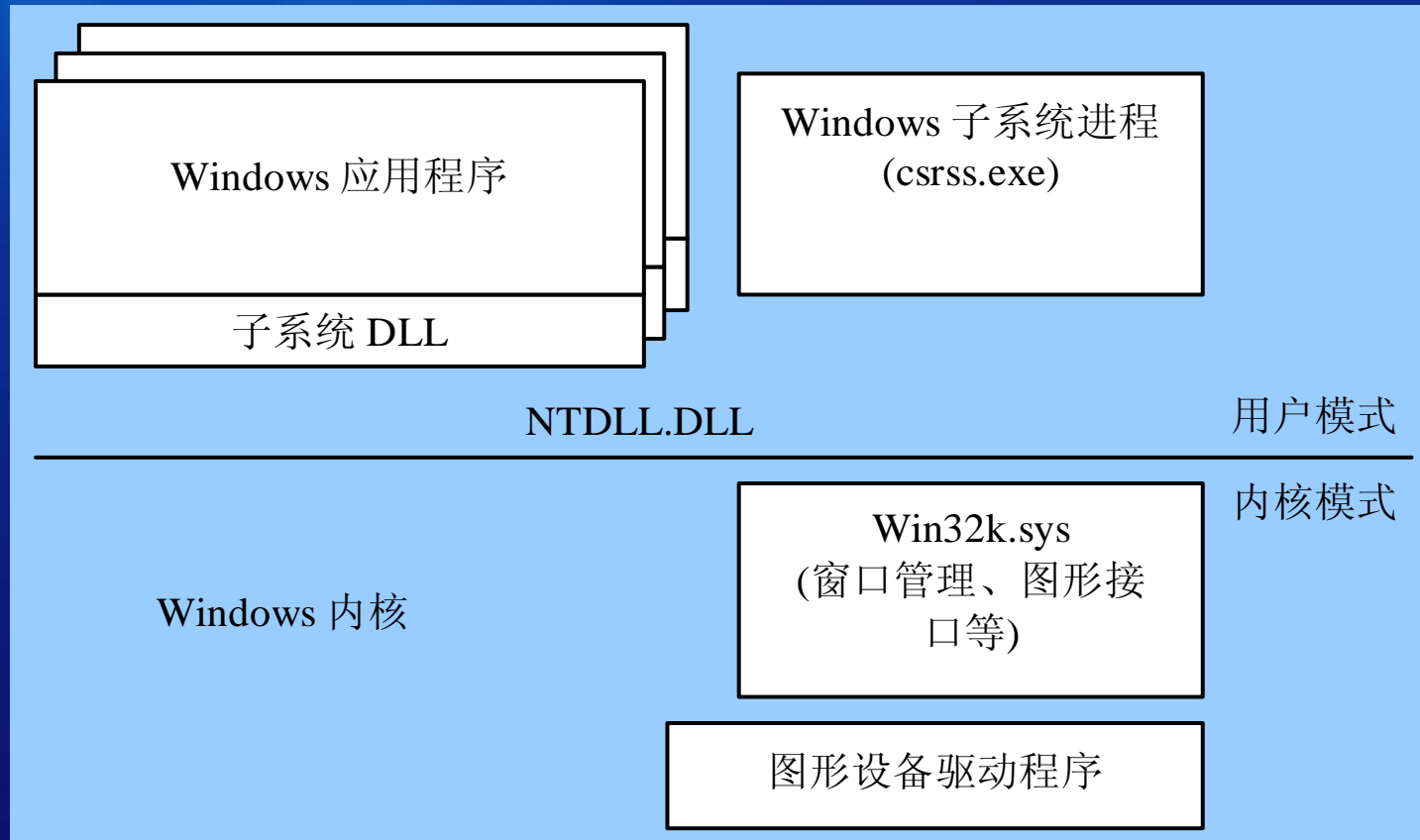
I/O Completion Port

- A kind of executive objects called **I/O Completion Port or IoCompletion**, its type is `IoCompletionObjectType`
- For balancing the I/O throughput and thread-trashing
- The completion port is essential a queue object
- A file can be associated with an `IoCompletion` object, so I/O manager will queue a completion packet to the completion port
- The number of active threads which process I/O completion are controlled by the queue object.

How does IOCompletion work



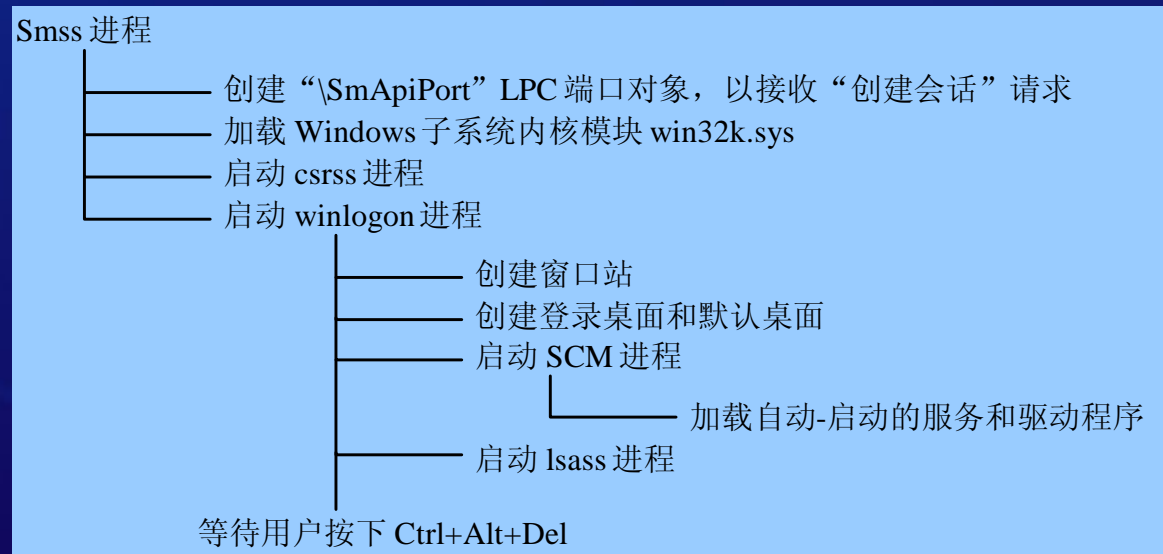
Windows Subsystem



Windows Subsystem Initialization

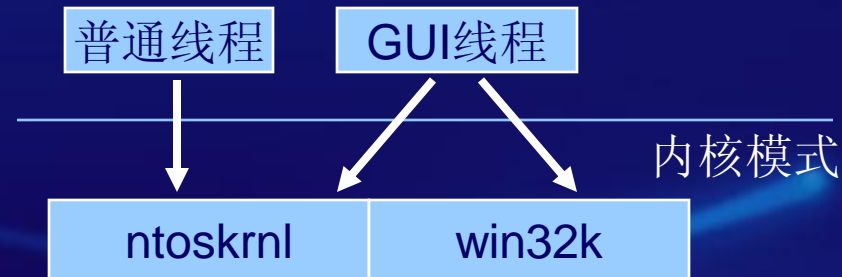
- DriverEntry in Win32k.sys
 - KeAddSystemServiceTable
 - PsEstablishWin32Callouts
 - MmPageEntireDriver
 - InitializeGre
 - Win32UserInitialize
 - Returns Win32KDriverUnload

- Smss
- Winlogon
- Csrss



Convert to a GUI thread

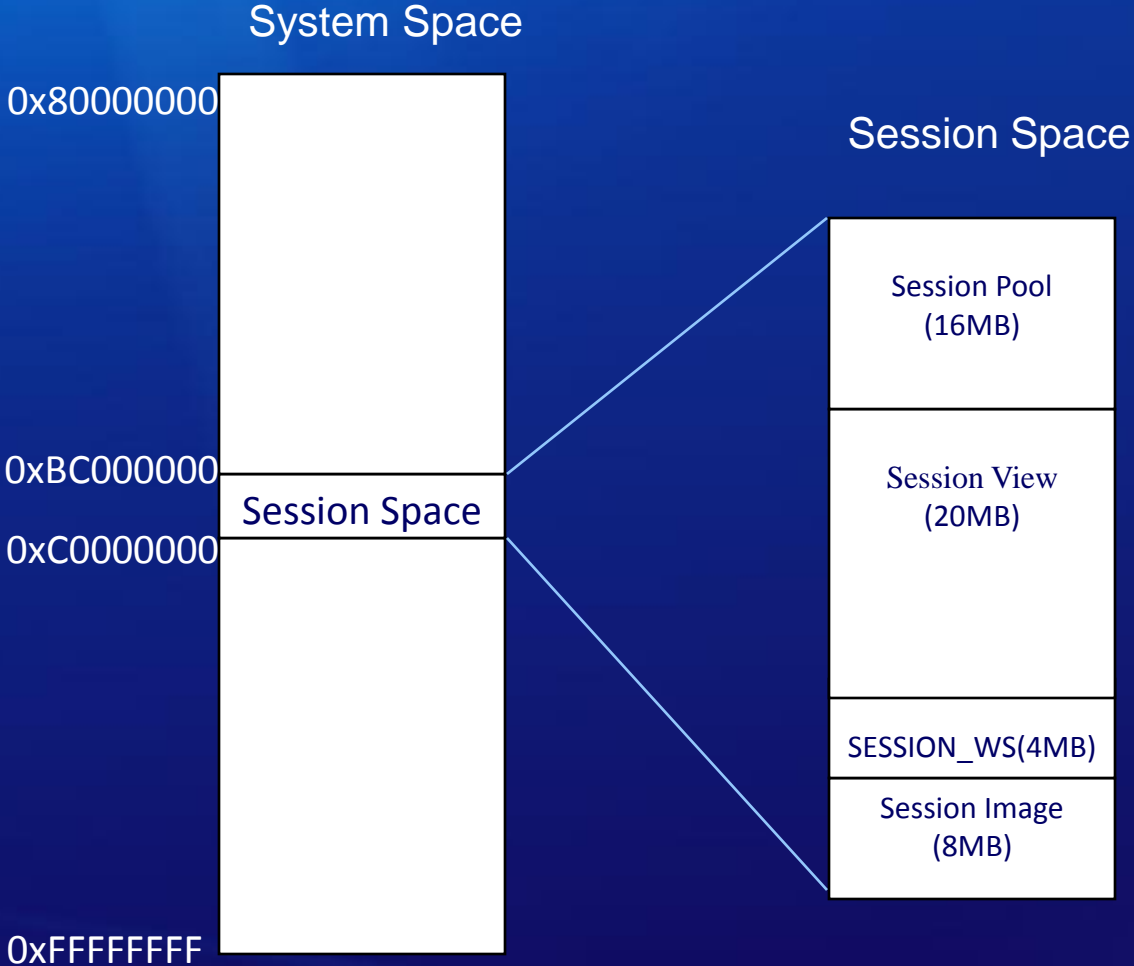
- What is a GUI thread?
 - Thread are non-GUI when created
 - Converted on first call to win32k.sys
 - Bigger Stack
 - Win32k.sys notified of creation and destruction
 - Converts process to GUI
- PsConvertToGuiThread
 - MmCreateKernelStack & KeSwitchKernelStack
 - KTHREAD->ServiceTable initialized to ntkrnl!KeServiceDescriptorTable, replaced with ntkrnl!KeServiceDescriptorTableShadow
 - Call PspW32ProcessCallout
 - Call PspW32ThreadCallout



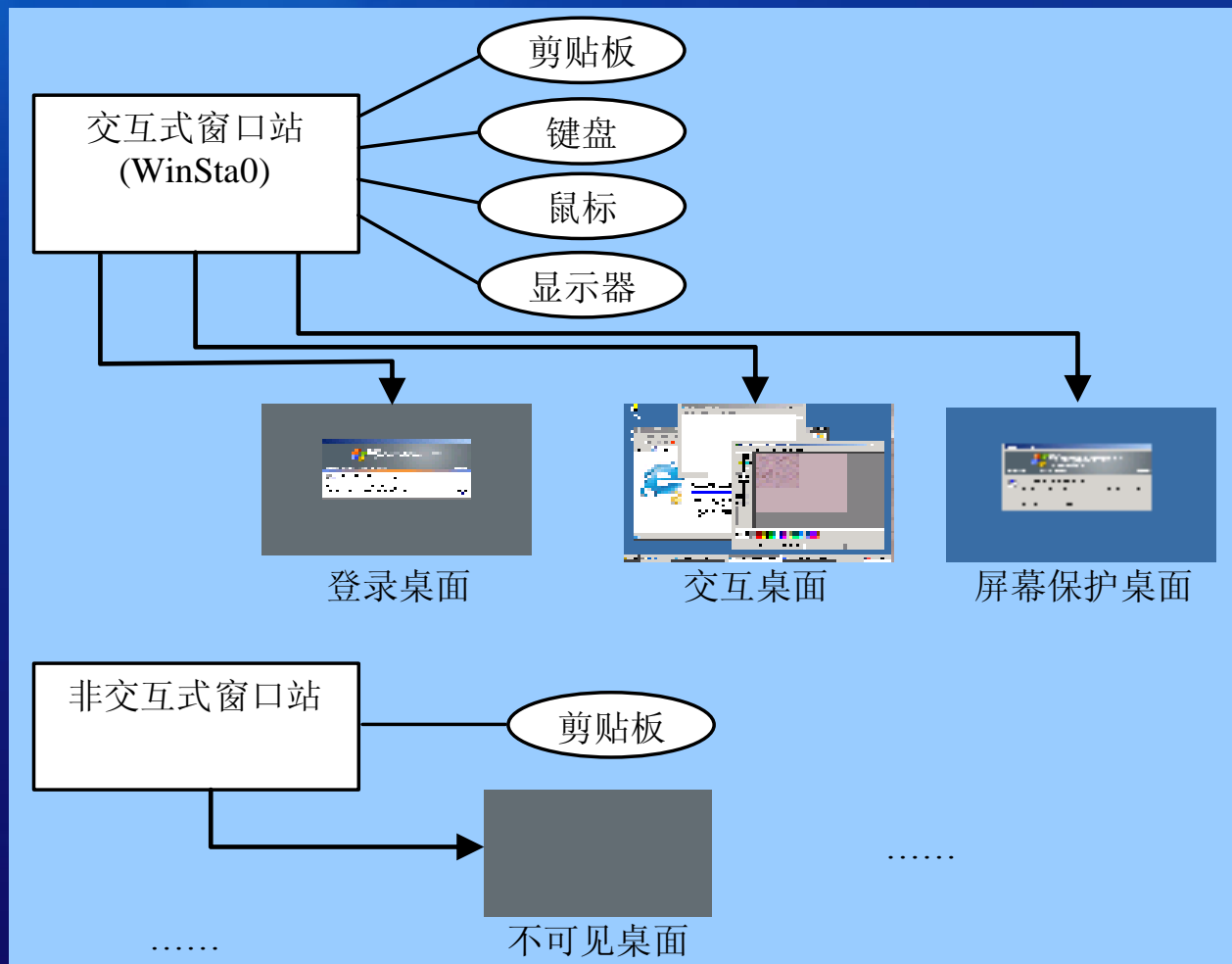
Terminal Services/Multiple Sessions

- Multiple Sessions
 - Console Session
 - Session 0 in Vista/Win7
 - Terminal Sessions
 - by FUS (Fast User Switching)
- For each session
 - Created by smss (session manager process)
 - Session space in system space
 - Its own copies of Win32k.sys, csrss.exe, Winlogon.exe, video driver, print driver, etc.

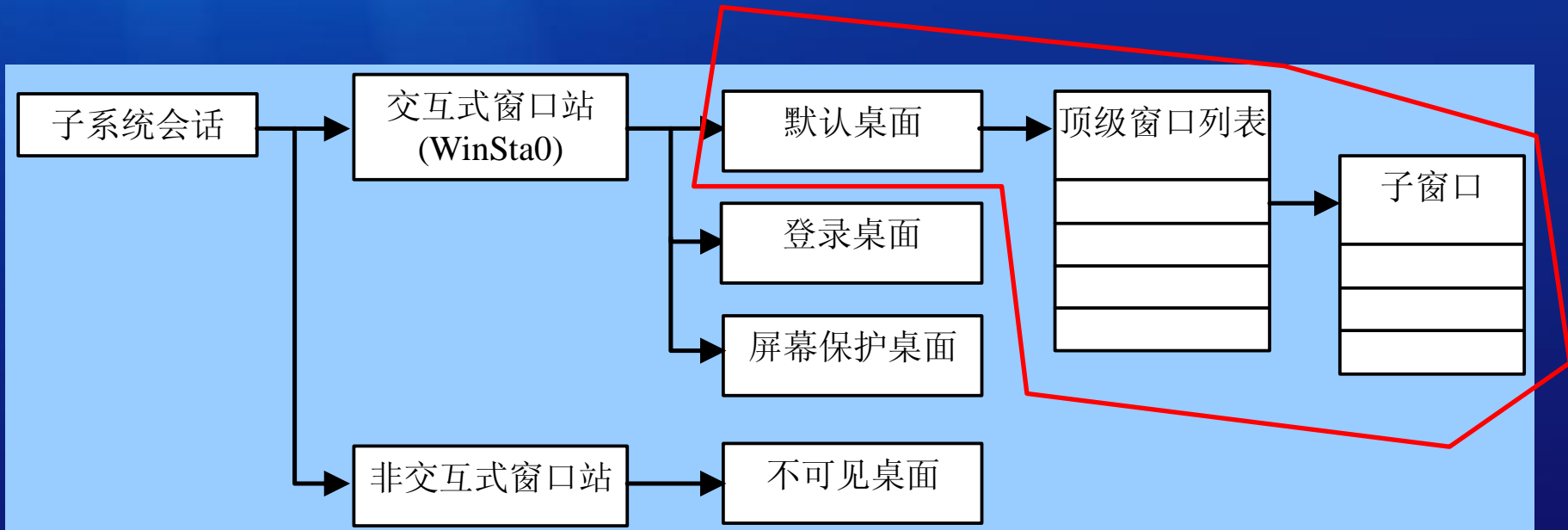
Session Spaces



Window Management: Window Stations and Desktops



Window Hierarchy

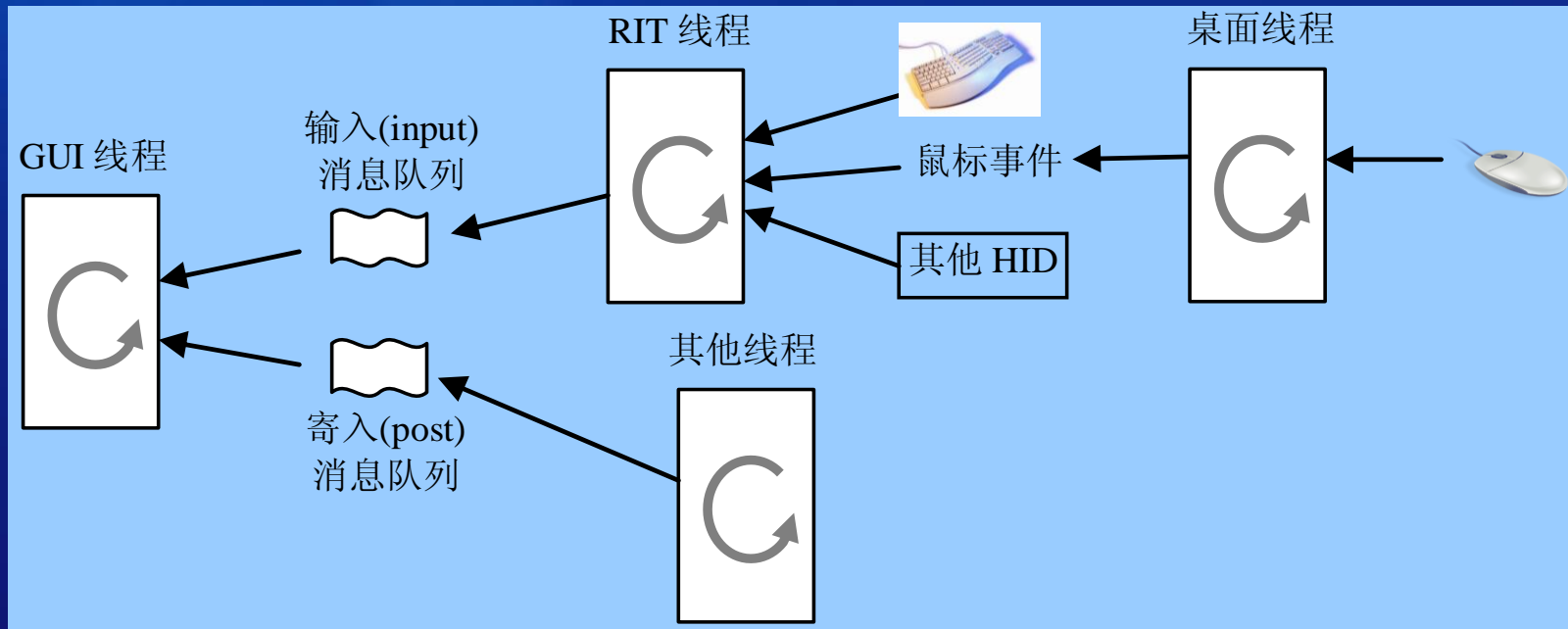


Desktop

- An surface for output
- In a WindowStation
- Have a Desktop Heap (assigned from the session view)

```
c:\kktools\dheapmon8.1\x86>dheapmon.exe
Desktop Heap Information Monitor Tool (Version 8.1.2925.0)
Copyright (c) Microsoft Corporation. All rights reserved.
-----
Session ID:    0 Total Desktop: ( 5312 KB - 7 desktops)
-----
WinStation\Desktop      Heap Size(KB)    Used Rate(%)
-----
WinSta0\Default         3072              2.5
WinSta0\Disconnect      64                4.0
WinSta0\Winlogon        128               6.1
Service-0x0-3e7$\Default 512               3.2
Service-0x0-3e4$\Default 512               3.2
Service-0x0-3e5$\Default 512               0.4
SAWinSta\SADesktop      512               0.4
-----
```

Message Path in Win32



References

- Mark E. Russinovich and David A. Solomon, *Windows Internals (4th/5th Edition)*, Microsoft Press, 2004/2009.
- 潘爱民, *Windows内核原理与实现*, 电子工业出版社, 2010. 4
- WRK, *Windows Research Kernel* (by Microsoft)
- *Microsoft WDK Documents & Samples*

Thanks!

Q&A