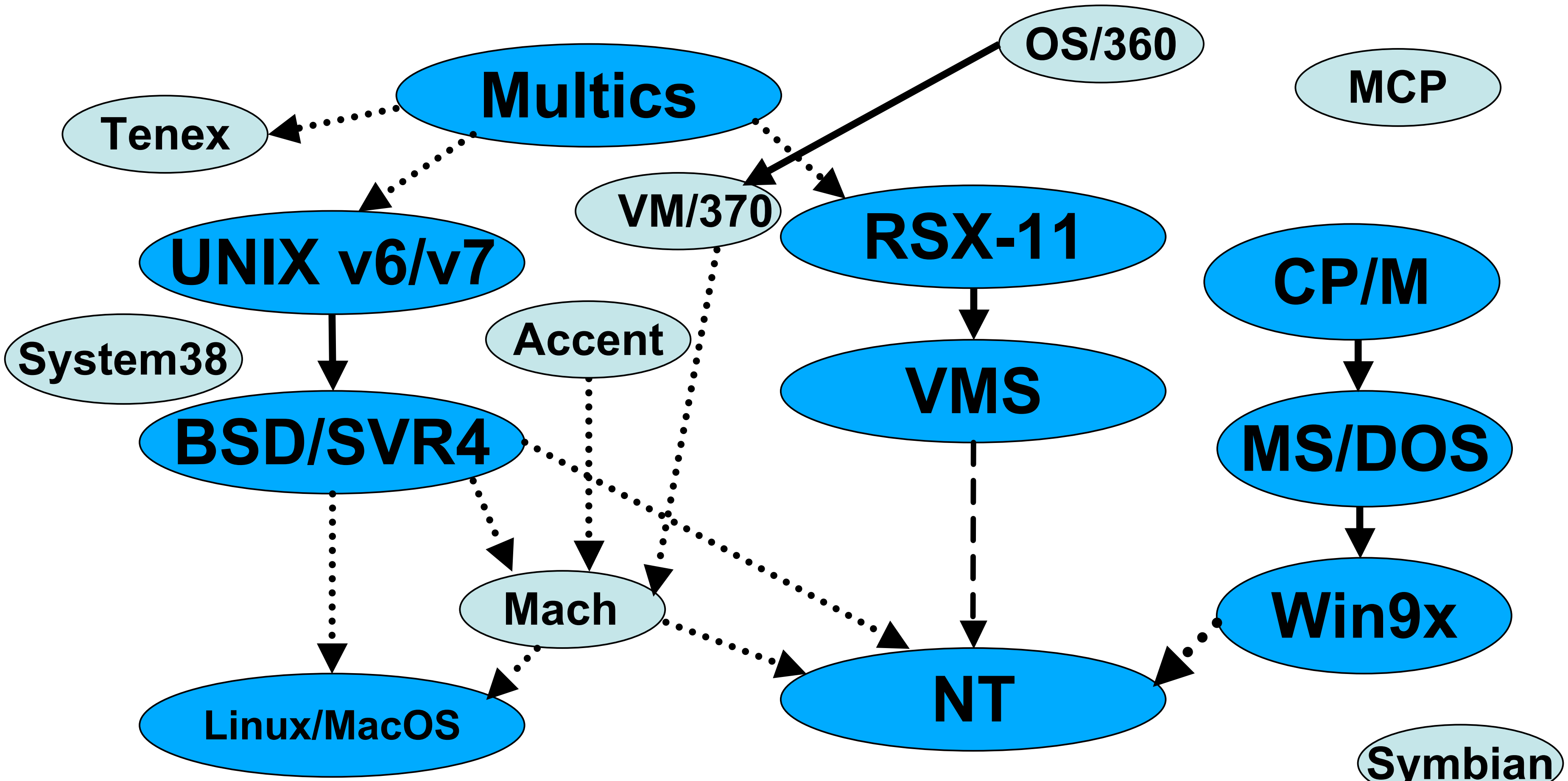
Architecture of the Windows Kernel

Berlin
April 2008

Dave Probert, Kernel Architect

Windows Core Operating Systems Division
Microsoft Corporation

Over-simplified OS history



Of all the interesting operating systems
only **UNIX** and **NT** matter (and maybe Symbian)

NT vs UNIX Design Environments

Environment which influenced fundamental design decisions	
Windows (NT)	**UNIX**
32-bit program address space	16-bit program address space
Mbytes of physical memory	Kbytes of physical memory
Virtual memory	Swapping system with memory mapping
Mbytes of disk, removable disks	Kbytes of disk, fixed disks
Multiprocessor (4-way)	Uniprocessor
Micro-controller based I/O devices	State-machine based I/O devices
Client/Server distributed computing	Standalone interactive systems
Large, diverse user populations	Small number of friendly users

Effect on OS Design

NT vs UNIX		
Although both Windows and Linux have adapted to changes in the environment, the original design environments (i.e. in 1989 and 1973) heavily influenced the design choices:		
Unit of concurrency:	Threads vs processes	Addr space, uniproc
Process creation:	CreateProcess() vs fork()	Addr space, swapping
I/O:	Async vs sync	Swapping, I/O devices
Namespace root:	Virtual vs Filesystem	Removable storage
Security:	ACLs vs uid/gid	User populations

Today's Environment

64-bit addresses Gbytes of physical memory Virtual memory, virtual processors Multiprocessors (64-128x) High-speed internet/intranet, Web Services Single user, but vulnerable to hackers worldwide
TV/PC Convergence Cellphone/Walkman/PDA/PC Convergence

Teaching unix AND Windows

“Compare & Contrast” drives innovation

- **Studying ‘foo’ is fine**
- **But if you also study ‘bar’, students will *compare & contrast***
- **Result is innovation:**
  - **Students mix & match concepts to create new ideas**
  - **Realizing there is not a single ‘right’ solution, students invent even more approaches**
  - **Learning to *think critically* is an important skill for students**

NT – the accidental secret

Historically little information on NT available

- Microsoft focus was end-users and Win9x
- Source code for universities was too encumbered

Much better internals information today

- Windows Internals, 4th Ed., Russinovich & Solomon
- Windows Academic Program (universities only):
  - CRK: Curriculum Resource Kit (NT kernel in PowerPoint)
  - WRK: Windows Research Kernel (NT kernel in source)
  - Design Workbook: soft copies of the original specs/notes
- Chapters in leading OS textbooks (Tanenbaum, Silberschatz, Stallings)

NT kernel philosophy

- Reliability, Security, Portability, Compatibility are all paramount
- Performance important
  - Multi-threaded, asynchronous
- General facilities that can be re-used
  - Support kernel-mode extensibility (for better or worse)
  - Provide unified mechanisms that can be shared
  - Kernel/executive split provides a clean layering model
  - Choose designs with *architectural headroom*

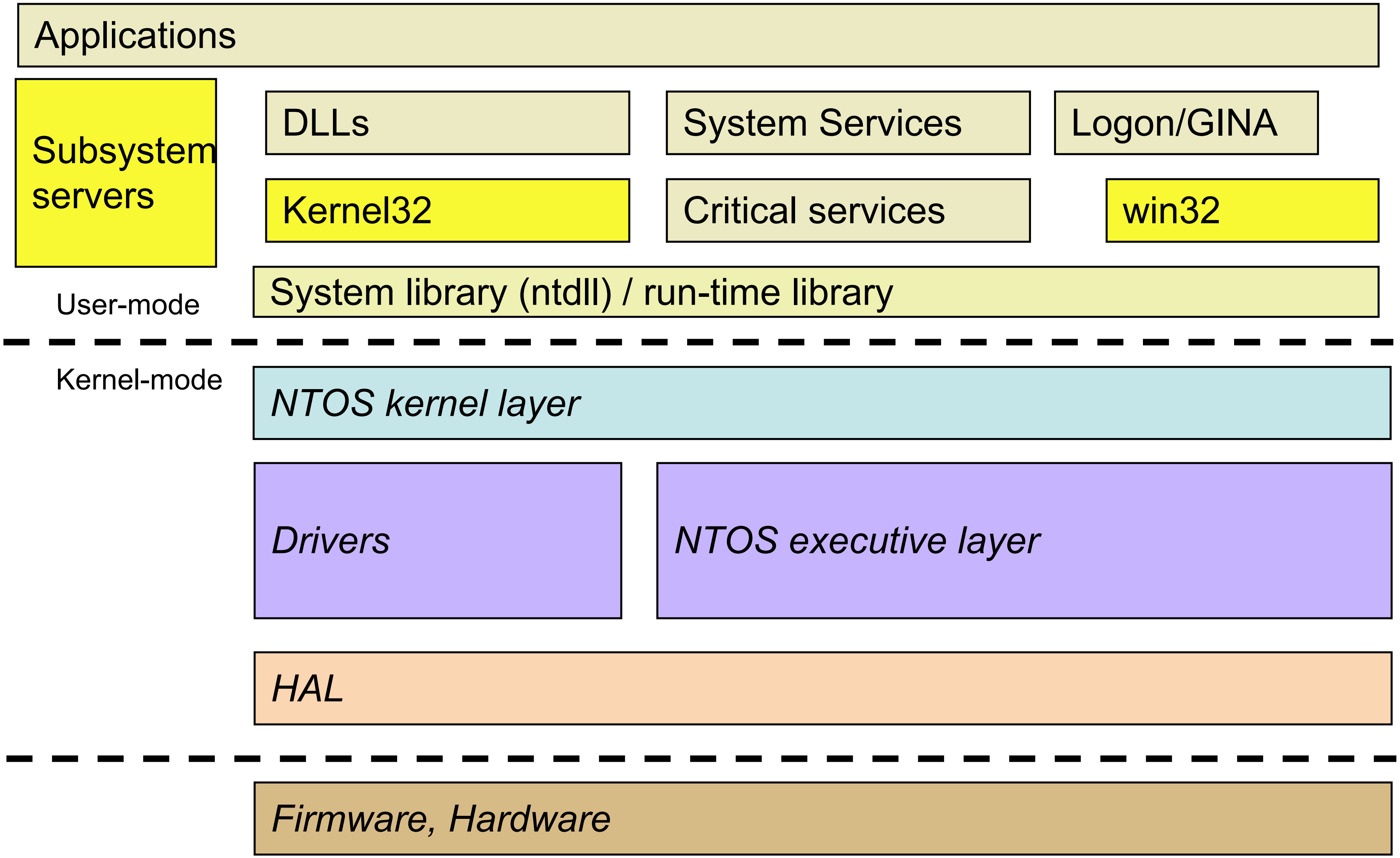
Important NT kernel features

- Highly multi-threaded in a process-like environment
- Completely asynchronous I/O model
- Thread-based scheduling
- Unified management of kernel data structures, kernel references, user references (handles), namespace, synchronization objects, resource charging, cross-process sharing
- Centralized ACL-based security reference monitor
- Configuration store decoupled from file system

Important NT kernel features (cont)

- Extensible filter-based I/O model with driver layering, standard device models, notifications, tracing, journaling, namespace, services/subsystems
- Virtual address space managed separately from memory objects
- Advanced VM features for databases (app management of virtual addresses, physical memory, I/O, dirty bits, and large pages)
- Plug-and-play, power-management
- System library mapped in every process provides trusted entrypoints

Windows Architecture



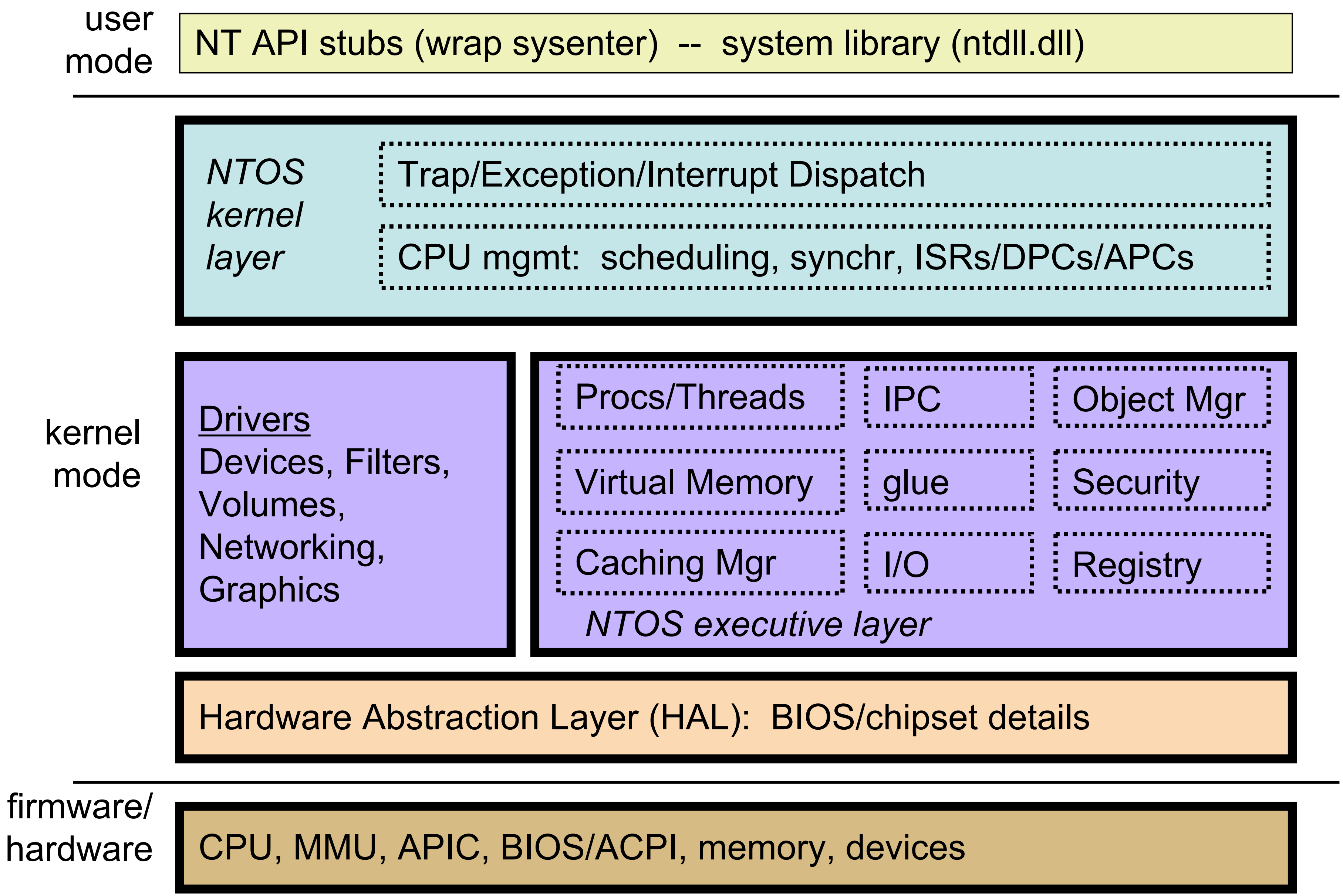
Windows user-mode

- Subsystems
  - OS Personality processes
  - Dynamic Link Libraries
  - *Why* NT mistaken for a microkernel
- System services (smss, lsass, services)
- System Library (ntdll.dll)
- Explorer/GUI (winlogon, explorer)
- Random executables (robocopy, cmd)

Windows kernel-mode

- NTOS (aka 'the kernel')
  - Kernel layer (abstracts the CPU)
  - Executive layer (OS kernel functions)
- Drivers (kernel-mode extension model)
  - Interface to devices
  - Implement file system, storage, networking
  - New kernel services
- HAL (Hardware Abstraction Layer)
  - Hides Chipset/BIOS details
  - Allows NTOS and drivers to run unchanged

Kernel-mode Architecture of Windows



Kernel/Executive layers

- Kernel layer – aka ‘ke’ (~ 5% of NTOS source)
  - Abstracts the CPU
    - Threads, Asynchronous Procedure Calls (APCs)
    - Interrupt Service Routines (ISRs)
    - Deferred Procedure Calls (DPCs – aka Software Interrupts)
  - Providers low-level synchronization
- Executive layer
  - OS Services running in a multithreaded environment
  - Full virtual memory, heap, handles
- Note: VMS had four layers:
  - Kernel / Executive / Supervisor / User

NT (Native) API examples

NtCreateProcess (&ProcHandle, Access, SectionHandle, DebugPort, ExceptionPort, …)

NtCreateThread (&ThreadHandle, ProcHandle, Access, ThreadContext, bCreateSuspended, …)

NtAllocateVirtualMemory (ProcHandle, Addr, Size, Type, Protection, …)

NtMapViewOfSection (SectHandle, ProcHandle, Addr, Size, Protection, …)

NtReadVirtualMemory (ProcHandle, Addr, Size, …)

NtDuplicateObject (srcProcHandle, srcObjHandle, dstProcHandle, dstHandle, Access, Attributes, Options)

Kernel Abstractions

Kernels implement abstractions

- Processes, threads, semaphores, files, …

Abstractions implemented as data and code

- Need a way of referencing instances

UNIX uses a variety of mechanisms

- File descriptors, Process IDs, SystemV IPC numbers

NT uses handles extensively

- Provides a unified way of referencing instances of kernel abstractions
- Objects can also be named (independently of the file system)

NT Object Manager

- **Generalizes access to kernel abstractions**
- **Provides unified management of:**
  - kernel data structures
  - kernel references
  - user references (handles)
  - namespace
  - synchronization objects
  - resource charging
  - cross-process sharing
  - central ACL-based security reference monitor
  - configuration (registry)

\ObjectTypes

Object Manager: Directory, SymbolicLink, Type

Processes/Threads: DebugObject, Job, Process, Profile, Section, Session, Thread, Token

Synchronization:

Event, EventPair, KeyedEvent, Mutant, Semaphore, ALPC Port, IoCompletion, Timer, TpWorkerFactory

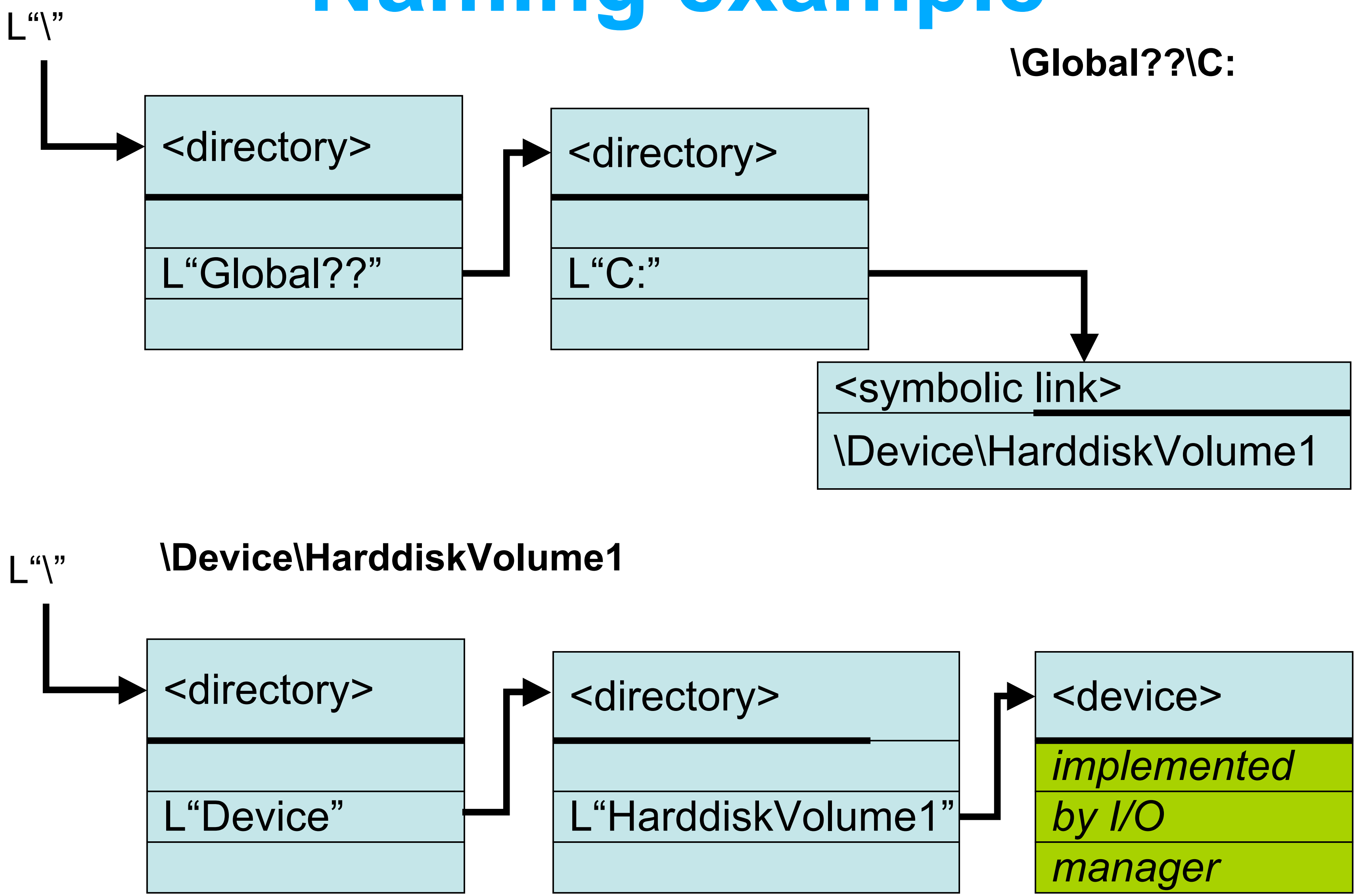
IO: Adapter, Controller, Device, Driver, File, Filter*Port

Kernel Transactions: TmEn, TmRm, TmTm, TmTx

Win32 GUI: Callback, Desktop, WindowStation

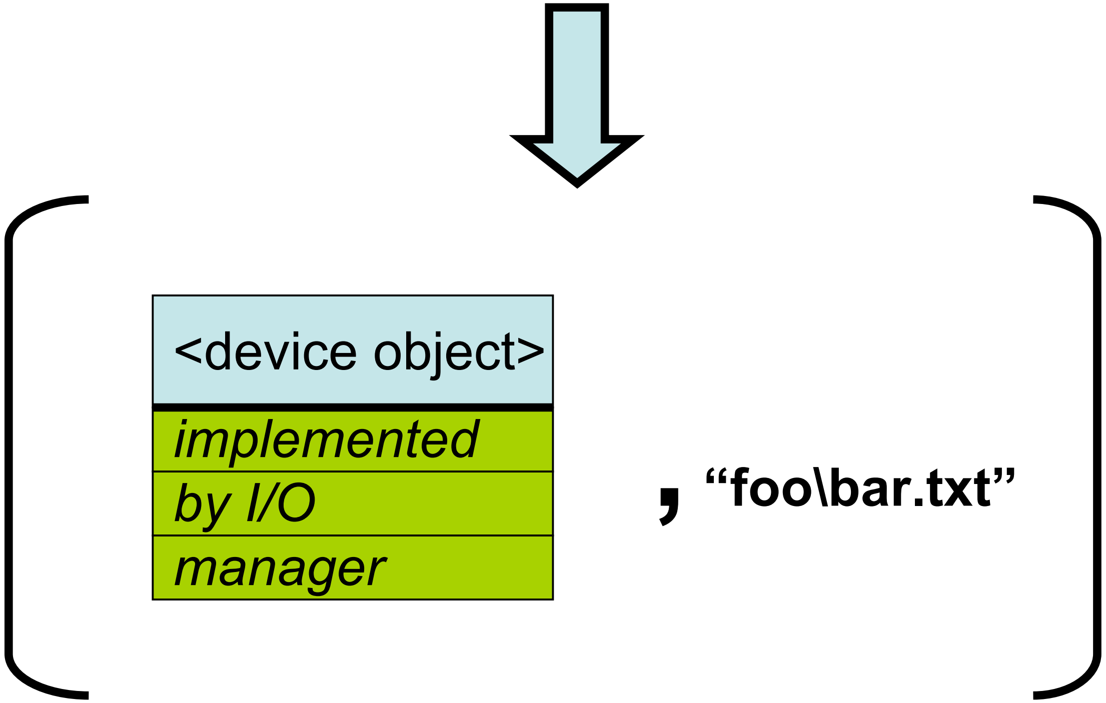
System: EtwRegistration, WmiGuid

Naming example



Object Manager Parsing example

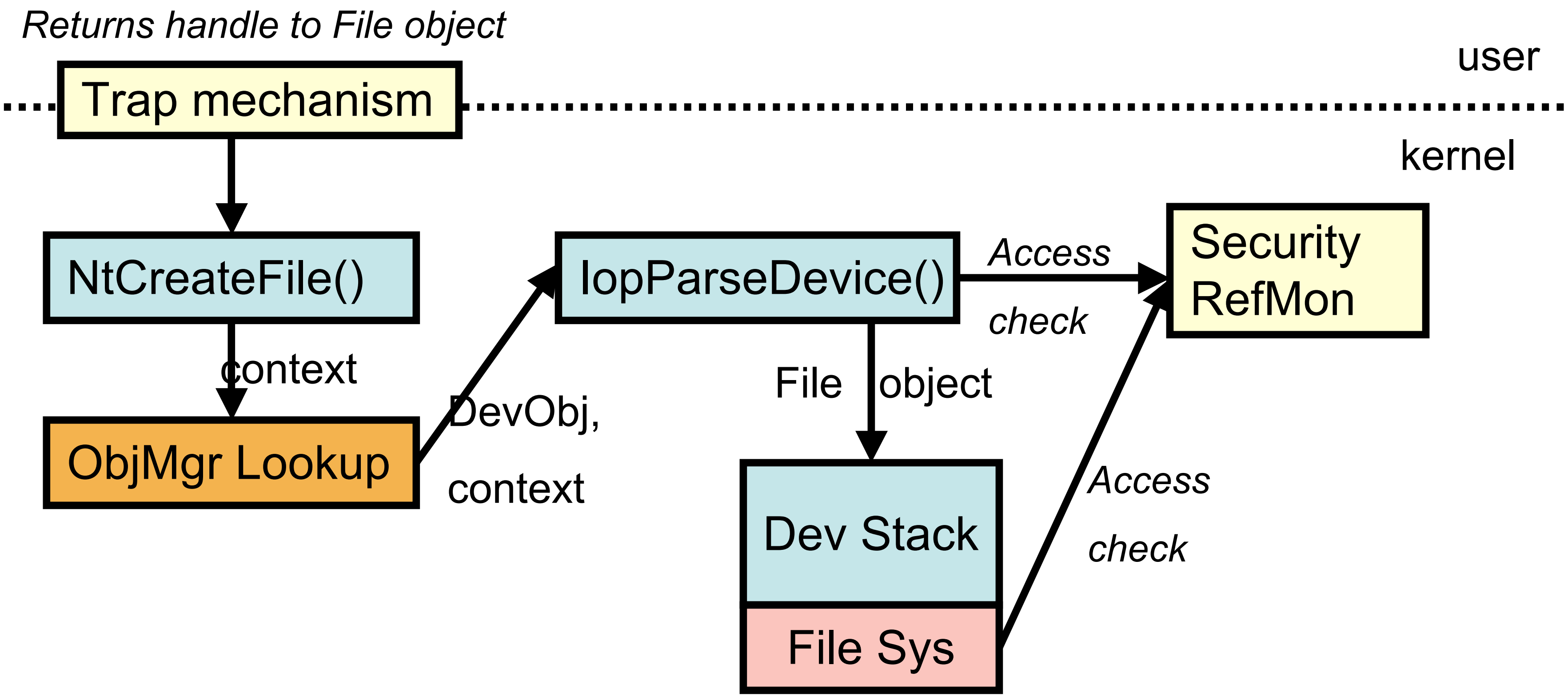
\Global??\C:\foo\bar.txt



deviceobject->ParseRoutine == IopParseDevice

Note: namespace rooted in object manager, not FS

I/O Support: IopParseDevice



File System Fills in File object

Why not root namespace in filesys?

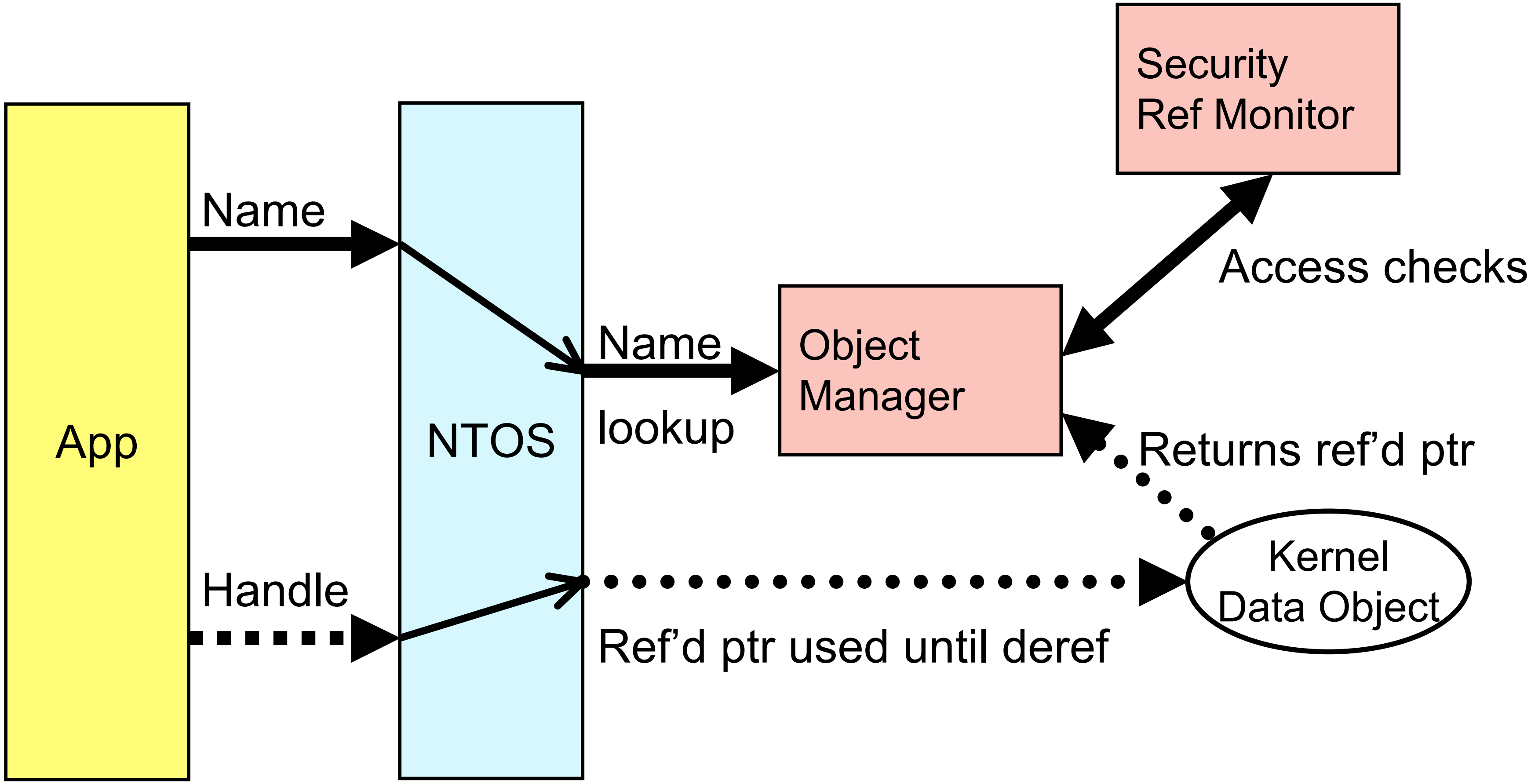
A few reasons…

- Hard to add new object types
- Device configuration requires filesys modification
- Root partition needed for each remote client
  - End up trying to make a tiny root for each client
  - Have to check filesystem *very* early

Windows uses object manager + registry hives

- Fabricates top-level namespace in kernel
- Uses config information from registry hive
- Only needs to modify hive after system stable

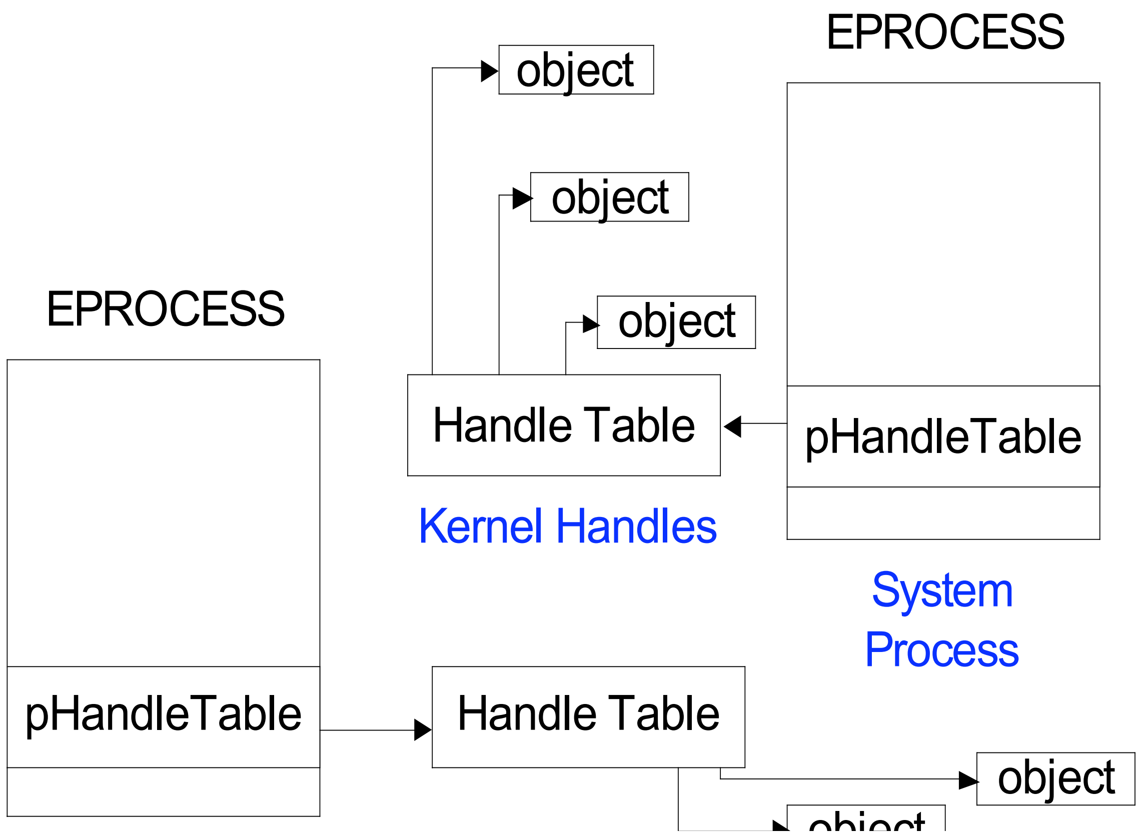
Object referencing



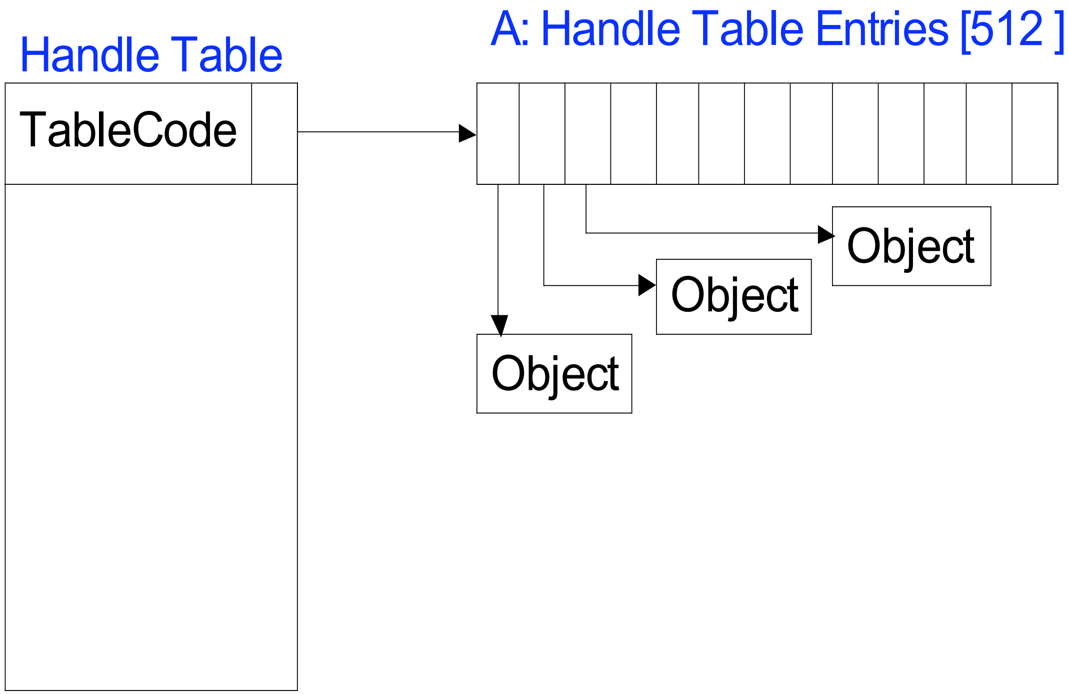
Handle Table

- NT handles allow user code to reference kernel data structures (similar, but more general than UNIX file descriptors)
- NT APIs use explicit handles to refer to objects (simplifying cross-process operations)
- Handles can be used for synchronization, including WaitMultiple
- Implementation is highly scalable

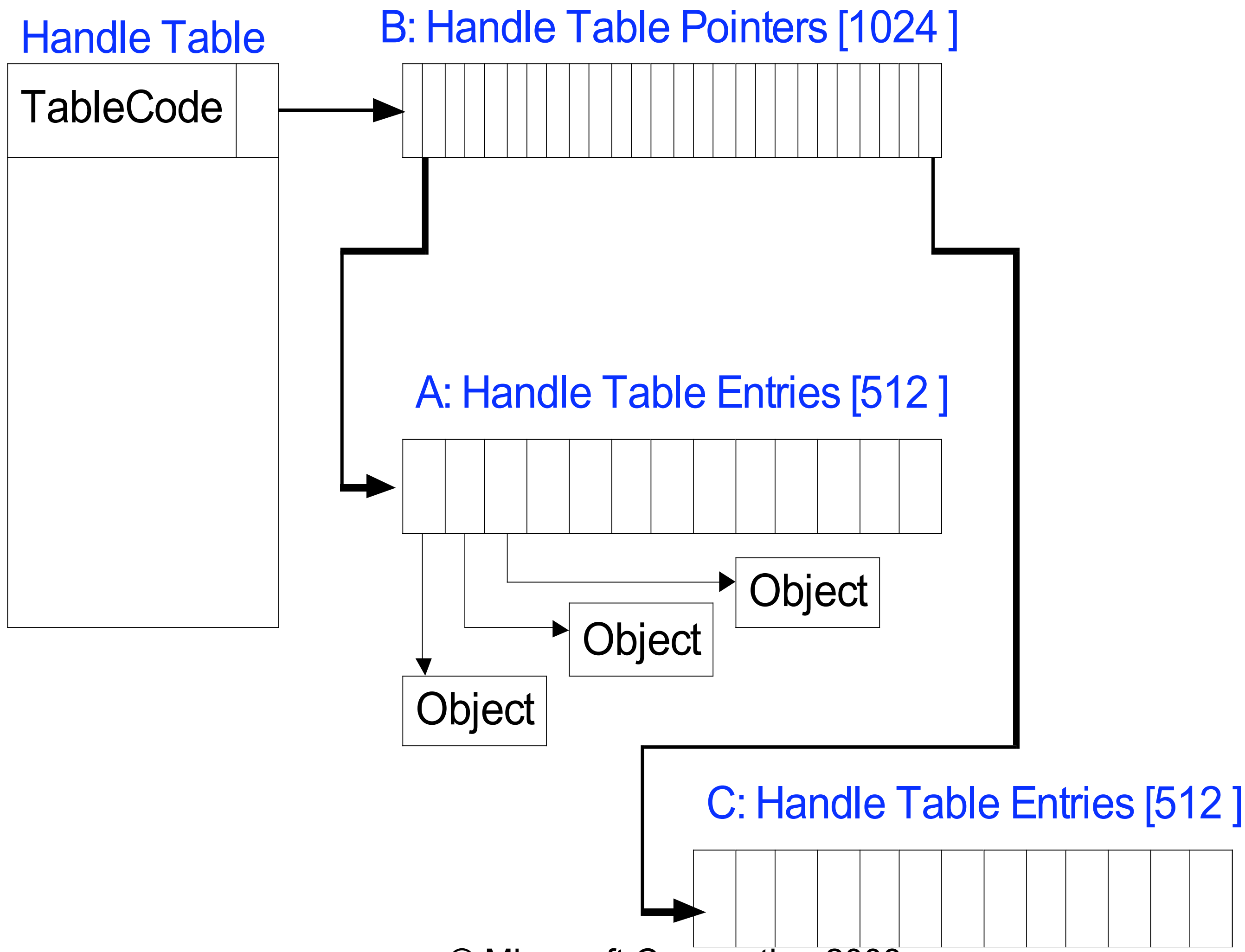
Process Handle Tables



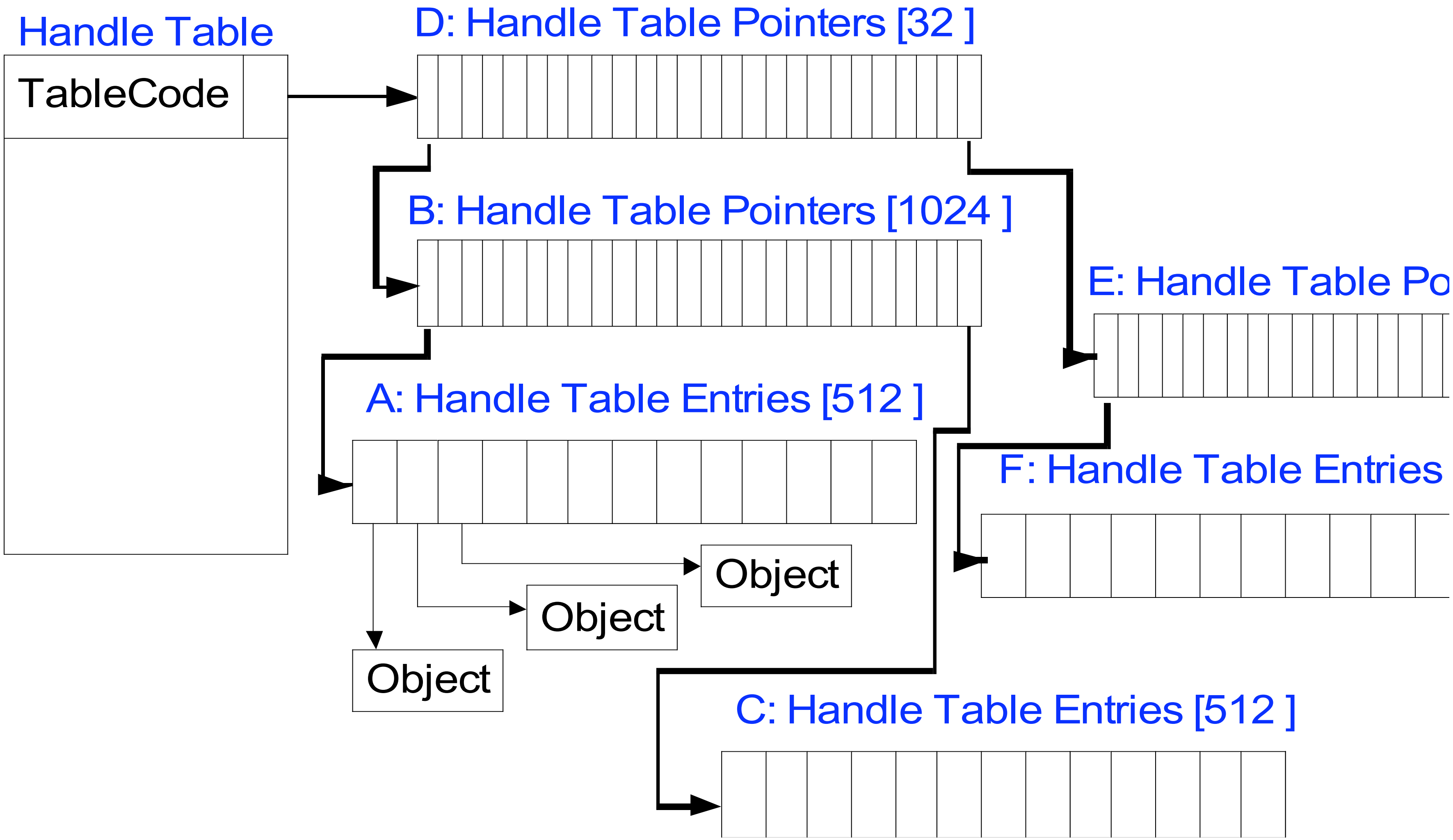
One level: (to 512 handles)



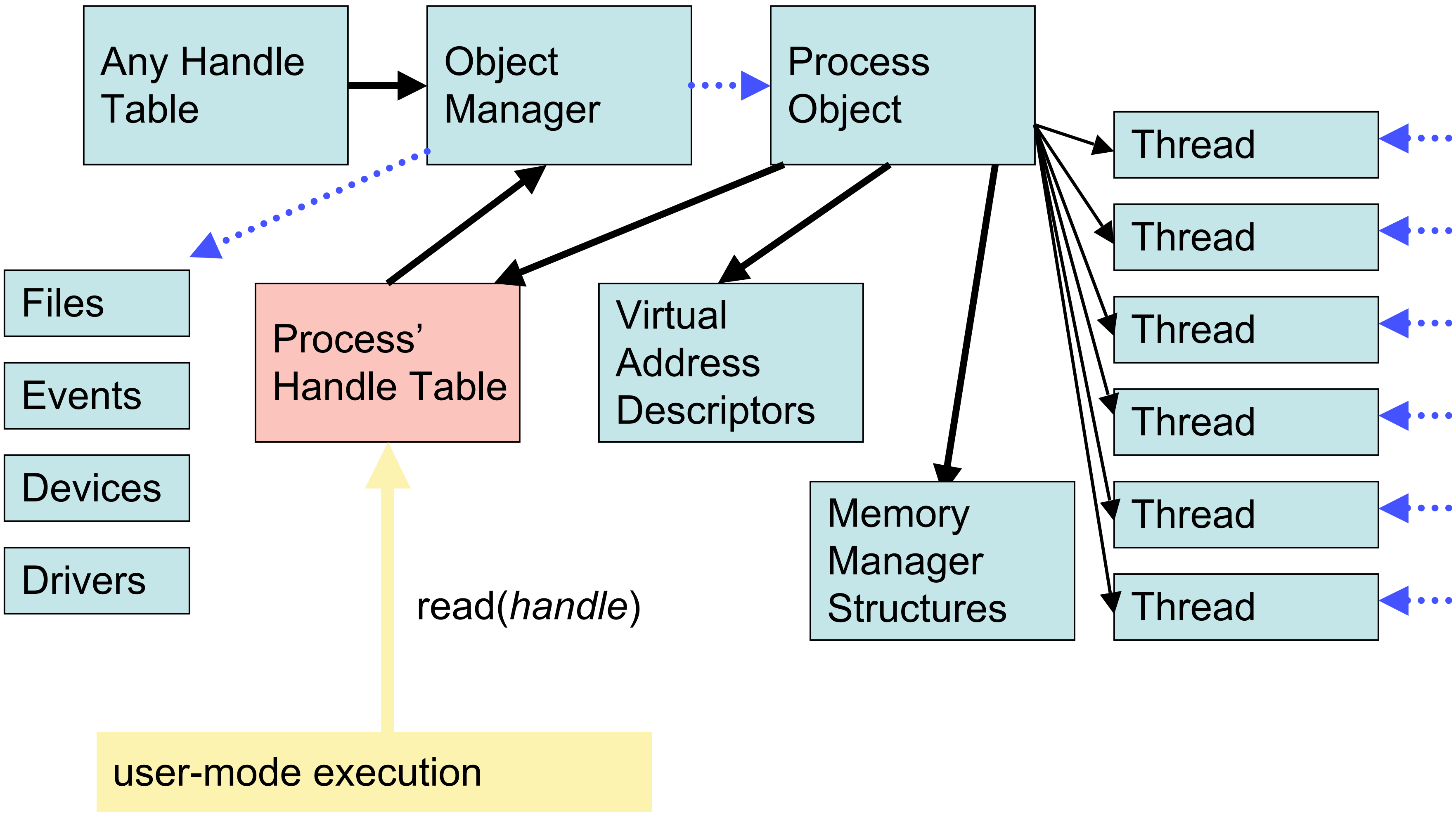
Two levels: (to 512K handles)



Three levels: (to 16M handles)



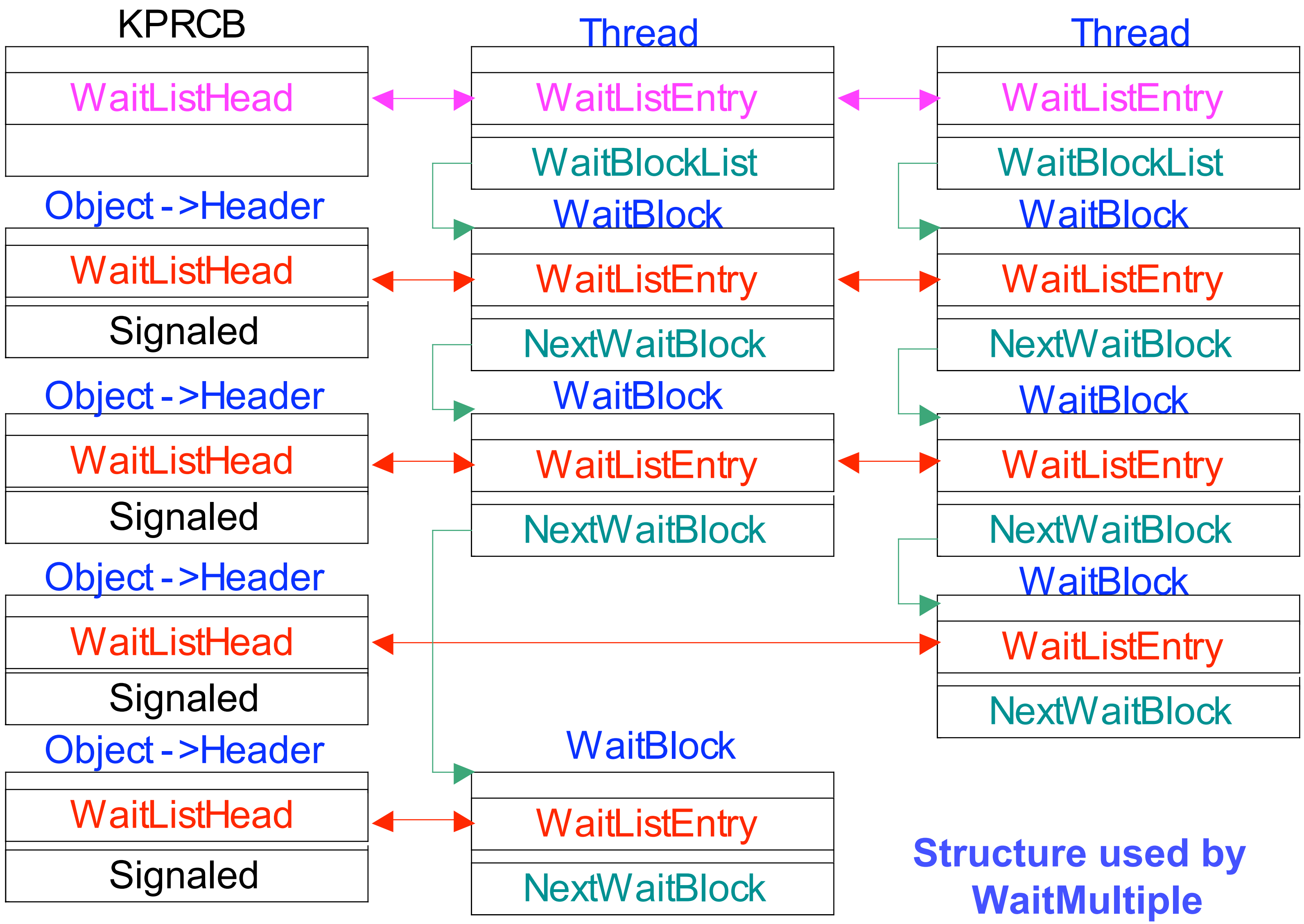
Process/Thread structure



OBJECT_HEADER

PointerCount			
HandleCount			
pObjectType			
oNameInfo	oHandleInfo	oQuotaInfo	Flags
pQuotaBlockCharged			
pSecurityDescriptor			
CreateInfo + NameInfo + HandleInfo + QuotaInfo			
OBJECT BODY [optional DISPATCHER_HEADER]			

Signaled
Event Type: Notification or Synchronization
Waiter List



Structure used by WaitMultiple

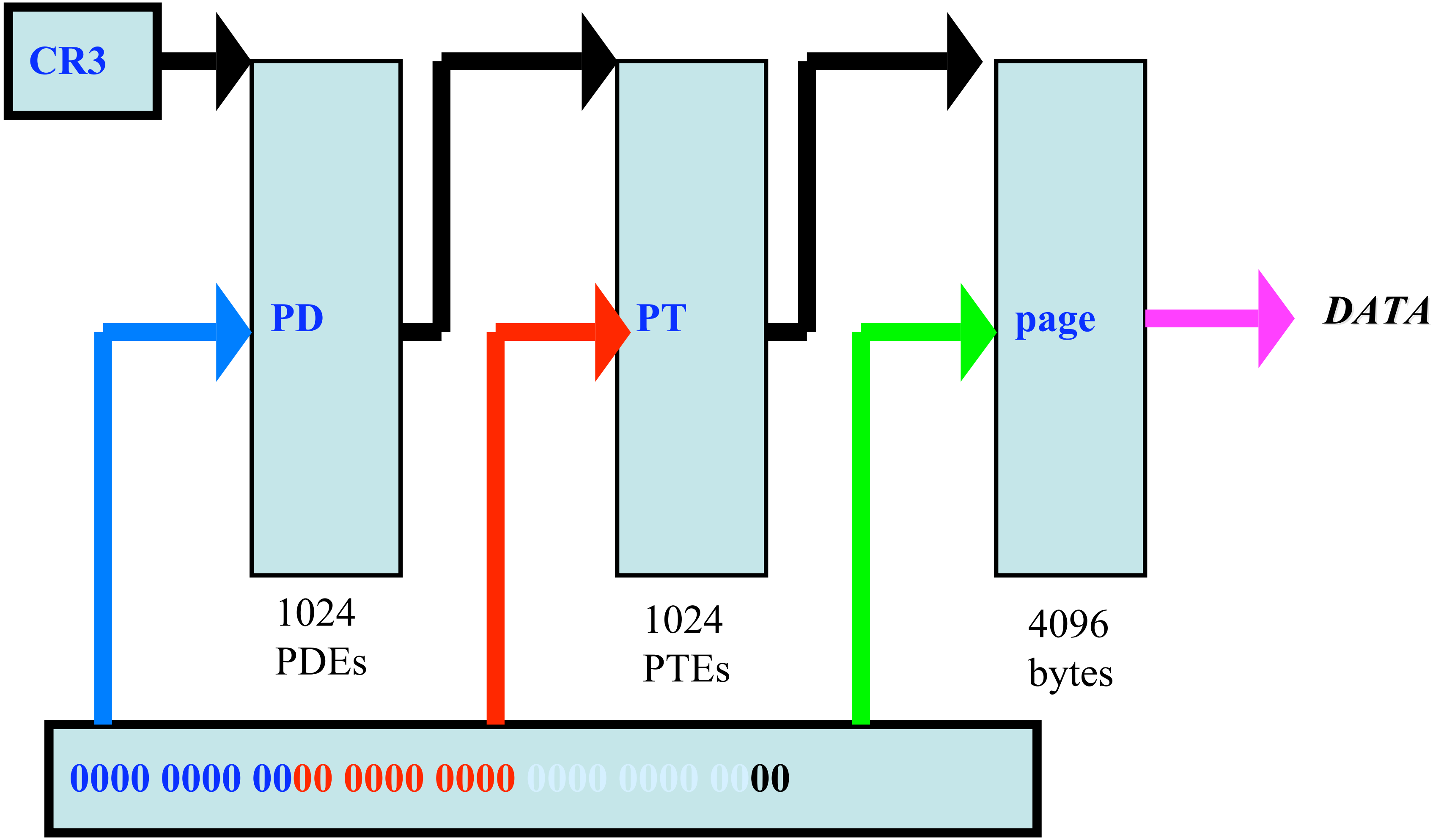
Summary: Object Manager

- Foundation of NT namespace
- Unifies access to kernel data structures
  – Outside the filesystem (initialized form registry)
  – Unified access control via Security Ref Monitor
  – Unified kernel-mode referencing (ref pointers)
  – Unified user-mode referencing (via handles)
  – Unified synchronization mechanism (events)

Processes

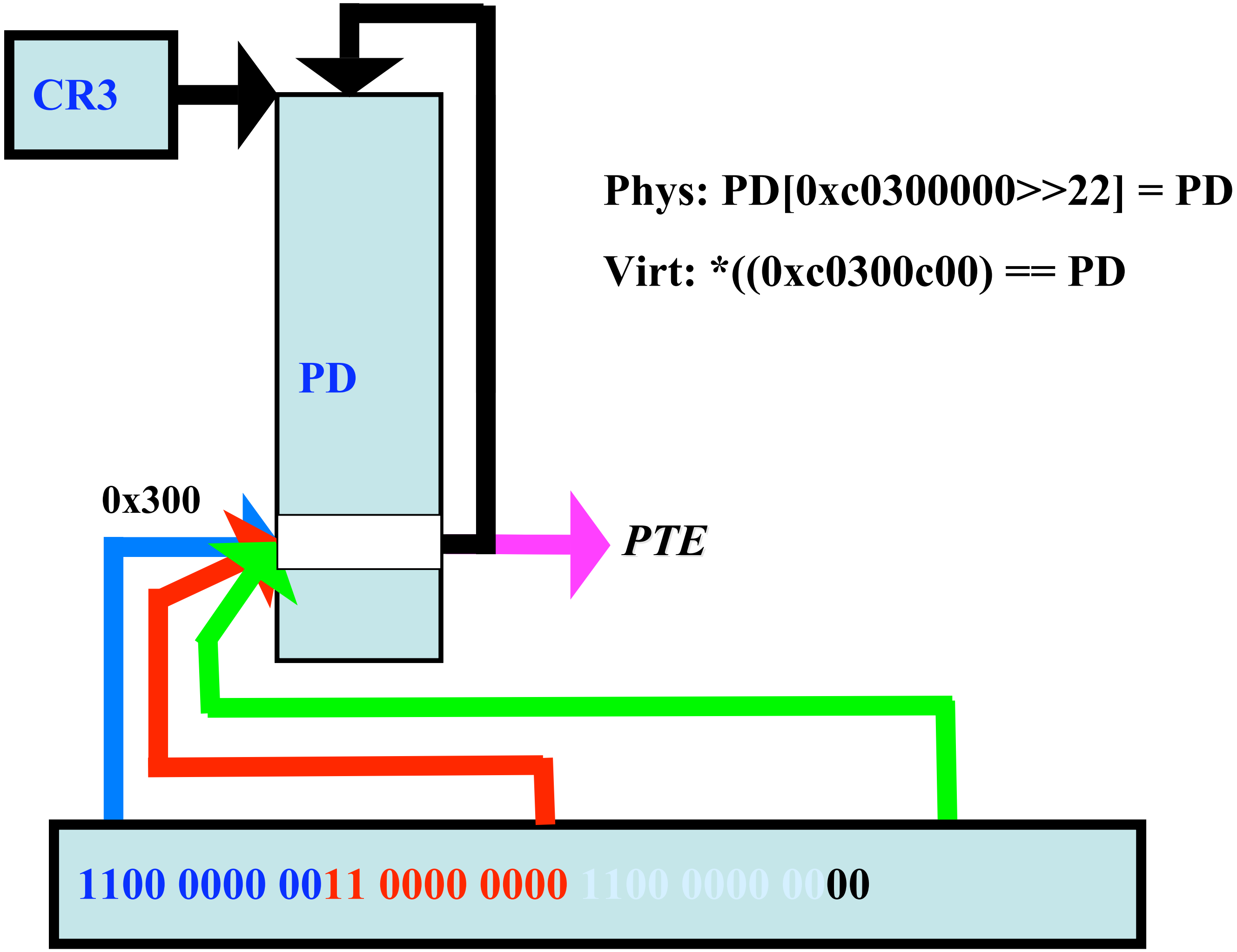
- An environment for program execution
- Binds
  - namespaces
  - virtual address mappings
  - ports (debug, exceptions)
  - threads
  - user authentication (token)
  - virtual memory data structures
- Abstracts the MMU, not the CPU

Virtual Address Translation



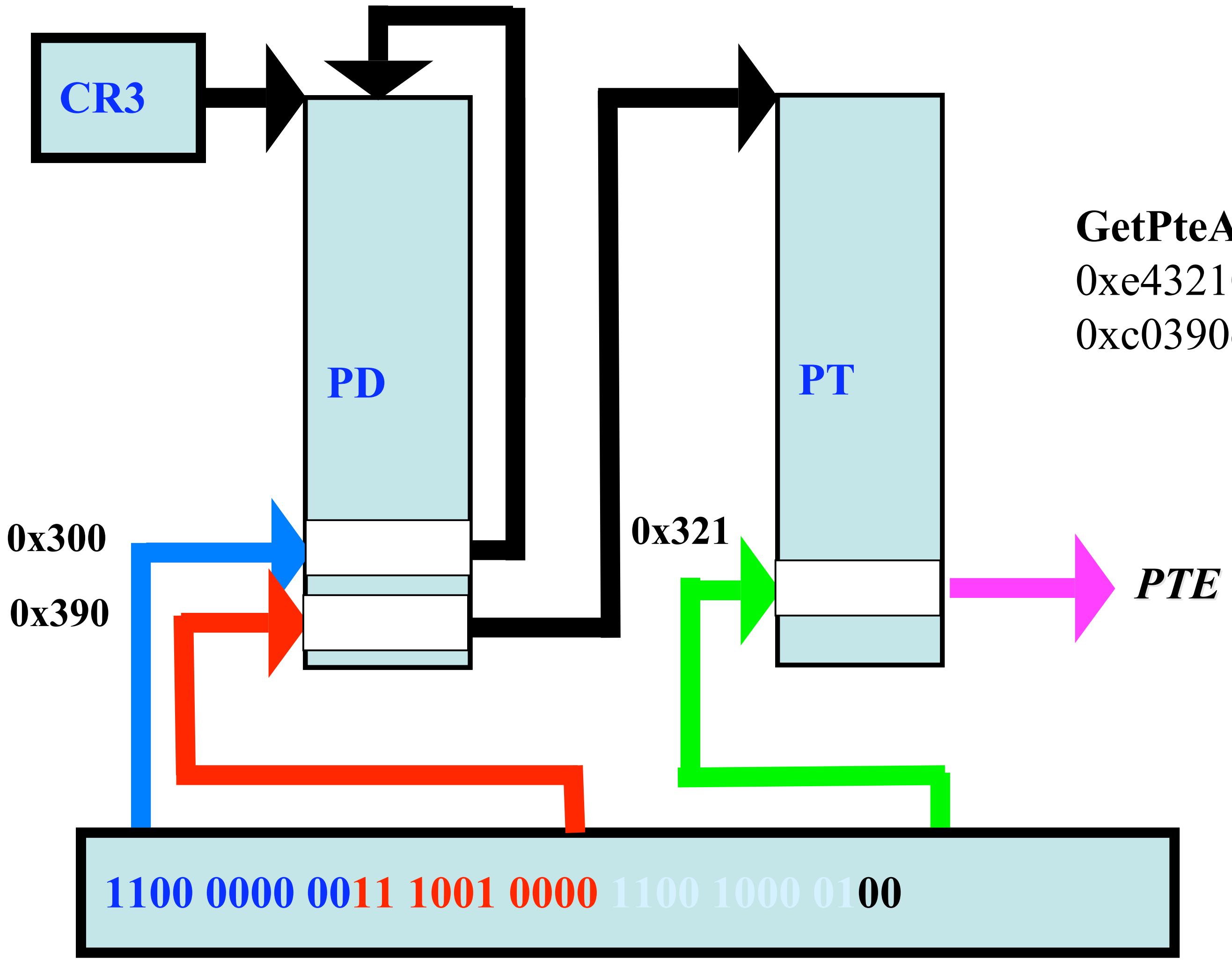
Self-mapping page tables

Virtual Access to PageDirectory[0x300]



Self-mapping page tables

Virtual Access to PTE for va 0xe4321000



GetPteAddress:
0xe4321000 =>
0xc0390c84

Virtual Address Descriptors

- Tree representation of an address space
- Types of VAD nodes
  - invalid
  - reserved
  - committed
  - committed to backing store
  - app-managed (large pages, AWE, physical)
- Backing store represented by section objects

Physical Frame Management

Page Tables

- hierarchical index of page directories and tables
- leaf-node is *page table entry* (PTE)
- PTE states:
  - Active/valid
  - Transition
  - Modified-no-write
  - Demand zero
  - Page file
  - Mapped file

Table of _PFN data structures

- represent all pageable pages
- synchronize page-ins
- linked to management lists: standby, modified, free, zero

Paging Overview

Working Sets: list of valid pages for each process (and the kernel)

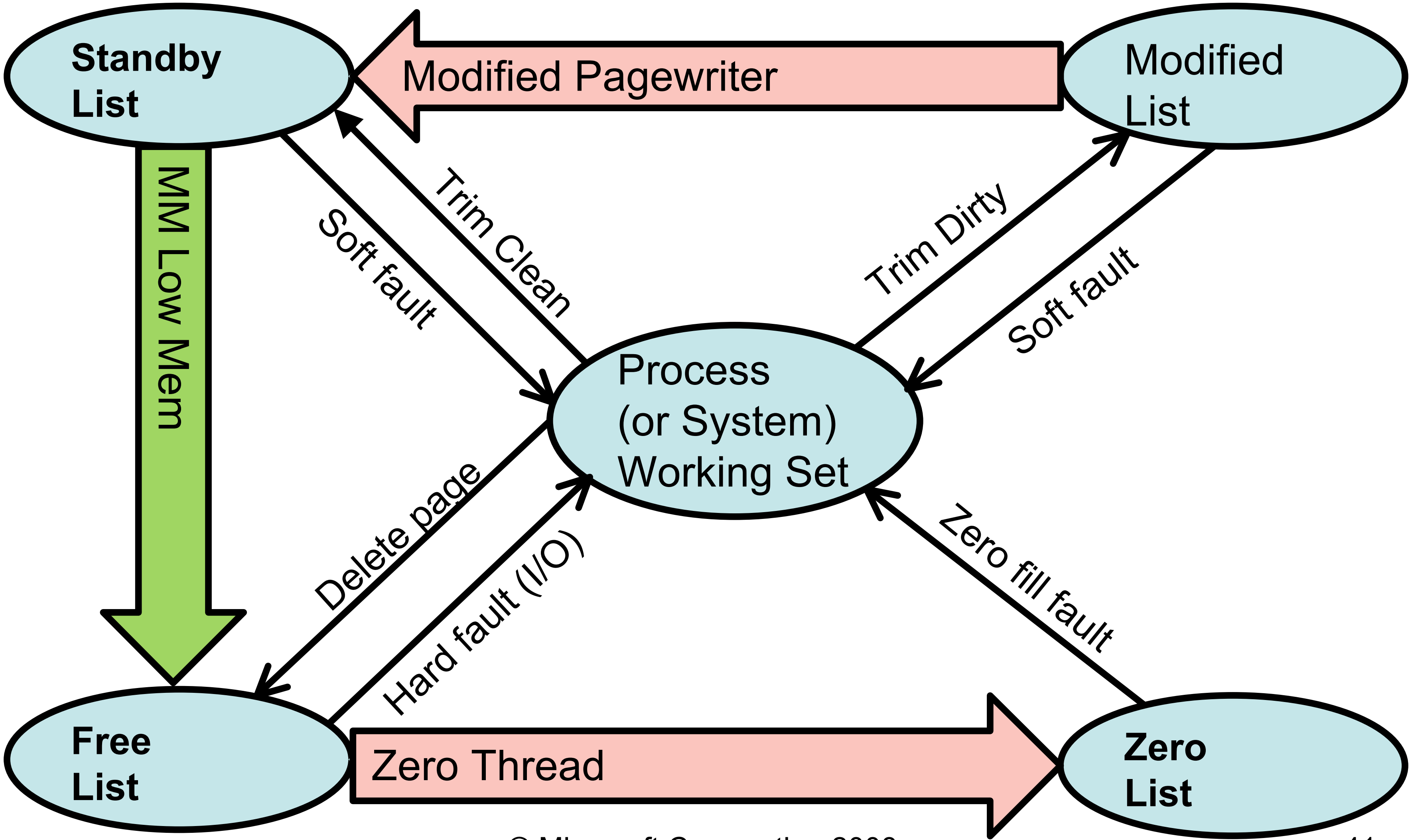
Pages ‘trimmed’ from working set on lists

- **Standby list**: pages backed by disk
- **Modified list**: dirty pages to push to disk
- **Free list**: pages not associated with disk
- **Zero list**: supply of demand-zero pages

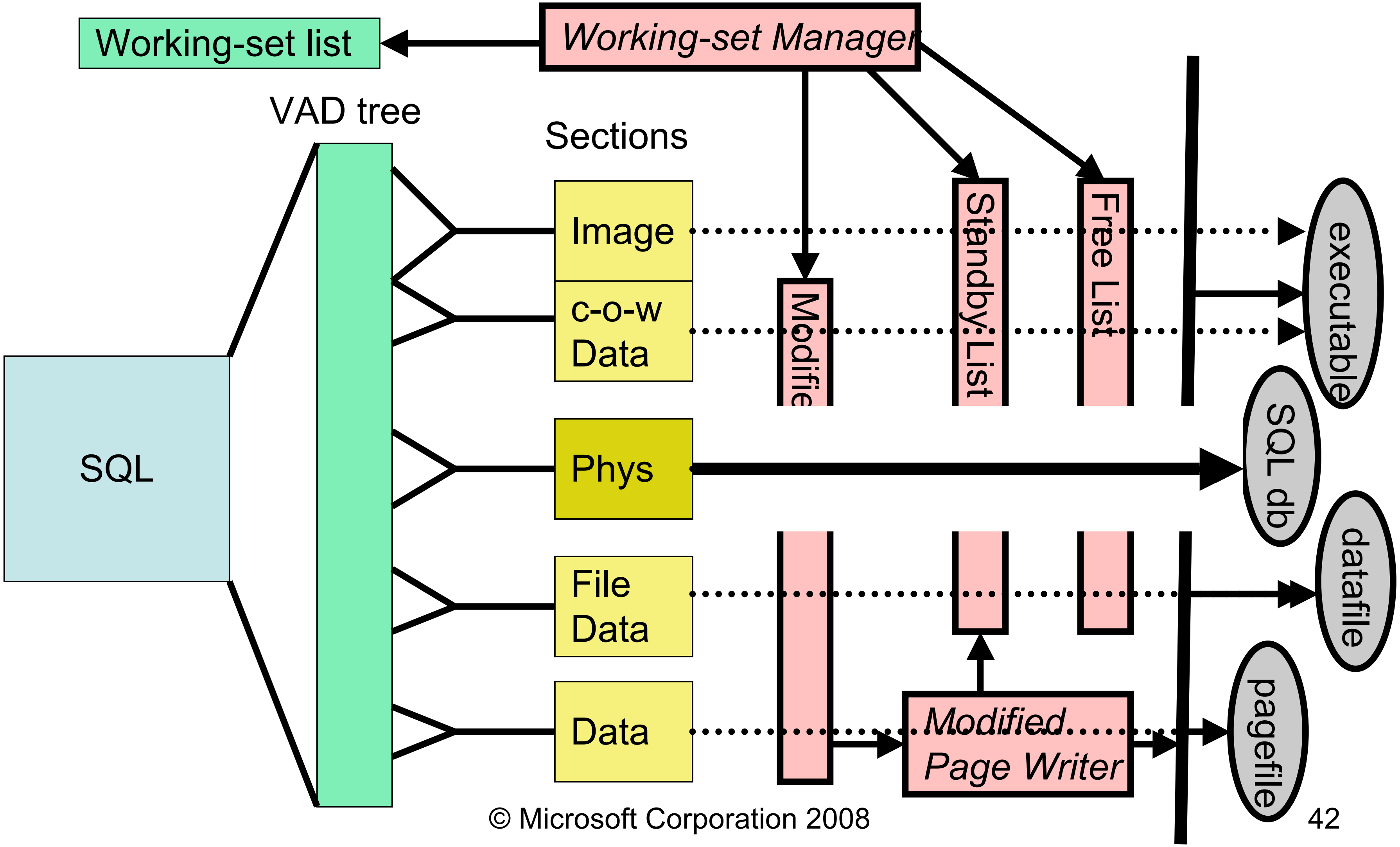
Modify/standby pages can be faulted back into a working set w/o disk activity (soft fault)

Background system threads trim working sets, write modified pages and produce zero pages based on memory state and config parameters

Physical Frame State Changes



32-bit VA/Memory Management



Threads

Unit of concurrency (abstracts the CPU)

Threads created within processes

System threads created within system process (kernel)

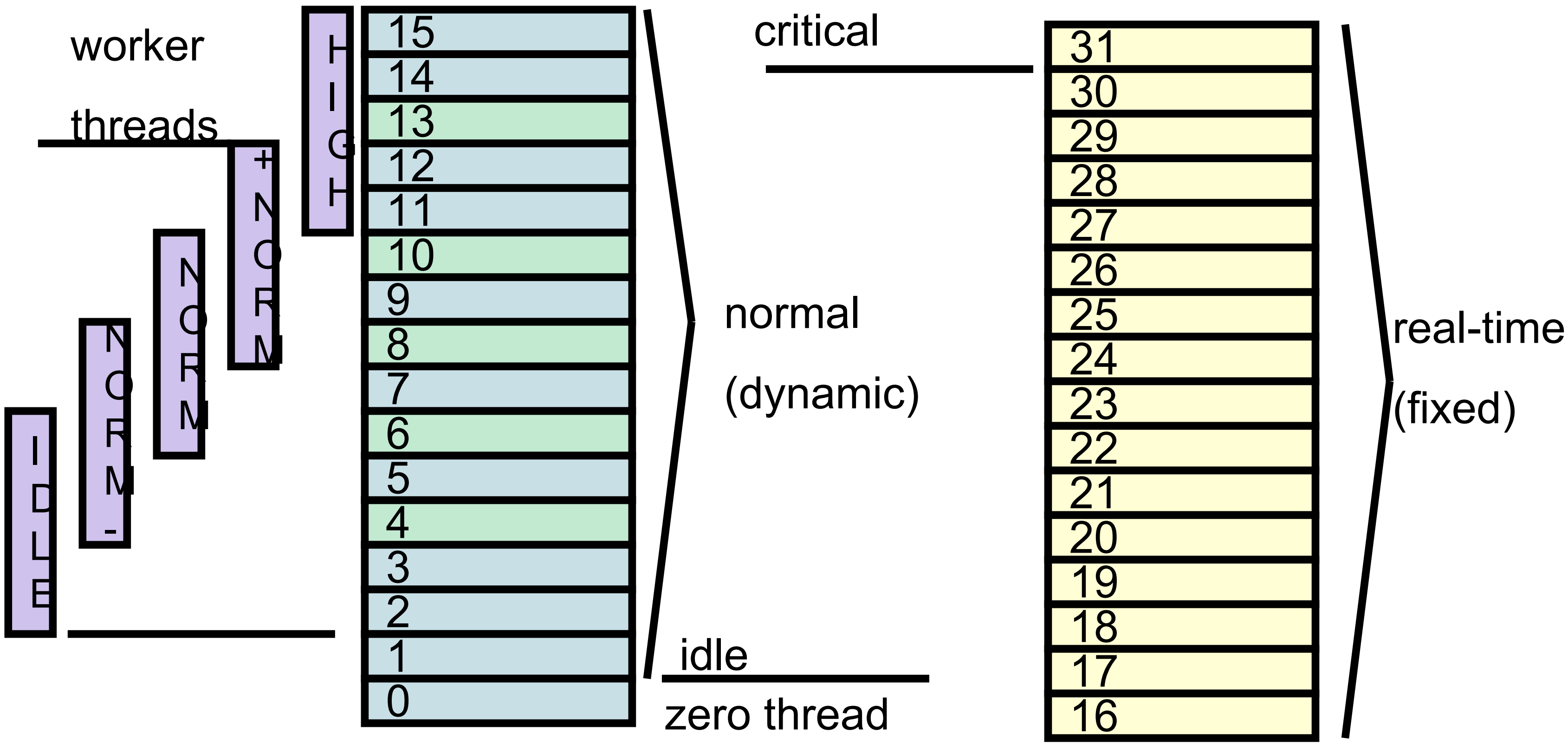
System thread examples:

- Dedicated threads
  - Lazy writer, modified page writer, balance set manager, mapped pager writer, other housekeeping functions
- General worker threads
  - Used to move work out of context of user thread
  - Must be freed before drivers unload
  - Sometimes used to avoid kernel stack overflows
- Driver worker threads
  - Extends pool of worker threads for heavy hitters, like file server

Scheduling

- **Windows schedules threads, not processes**
  - Scheduling is preemptive, priority-based, and round-robin at the highest-priority
  - 16 real-time priorities above 16 normal priorities
  - Scheduler tries to keep a thread on its ideal processor/node to avoid perf degradation of cache/NUMA-memory
  - Threads can specify affinity mask to run only on certain processors
- Each thread has a current & base priority
  - Base priority initialized from process
  - Non-realtime threads have priority boost/decay from base
  - Boosts for GUI foreground, waking for event
  - Priority decays, particularly if thread is CPU bound (running at quantum end)
- **Scheduler is state-driven by timer, setting thread priority, thread block/exit, etc**
- Priority inversions can lead to starvation
  - balance manager periodically boosts non-running runnable threads

NT thread priorities



CPU Control-flow

Thread scheduling occurs at PASSIVE or APC level (IRQL < 2)

APCs (Asynchronous Procedure Calls) deliver I/O completions, thread/process termination, etc (IRQL == 1)

- Not a general mechanism like unix signals (user-mode code must explicitly block pending APC delivery)

Interrupt Service Routines run at IRL > 2

ISRs defer most processing to run at IRQL==2 (DISPATCH level) by queuing a **DPC** to their current processor

A pool of ***worker threads*** available for kernel components to run in a normal thread context when user-mode thread is unavailable or inappropriate

Normal thread scheduling is round-robin among priority levels, with priority adjustments (except for fixed priority real-time threads)

Summary: CPU

- Multiple mechanisms for getting CPU
  - Integrated with the I/O system
- Thread is basic unit of scheduling
- Highly preemptive kernel environment
- Real-time scheduling priorities
- *Interesting part is locking/scalability*

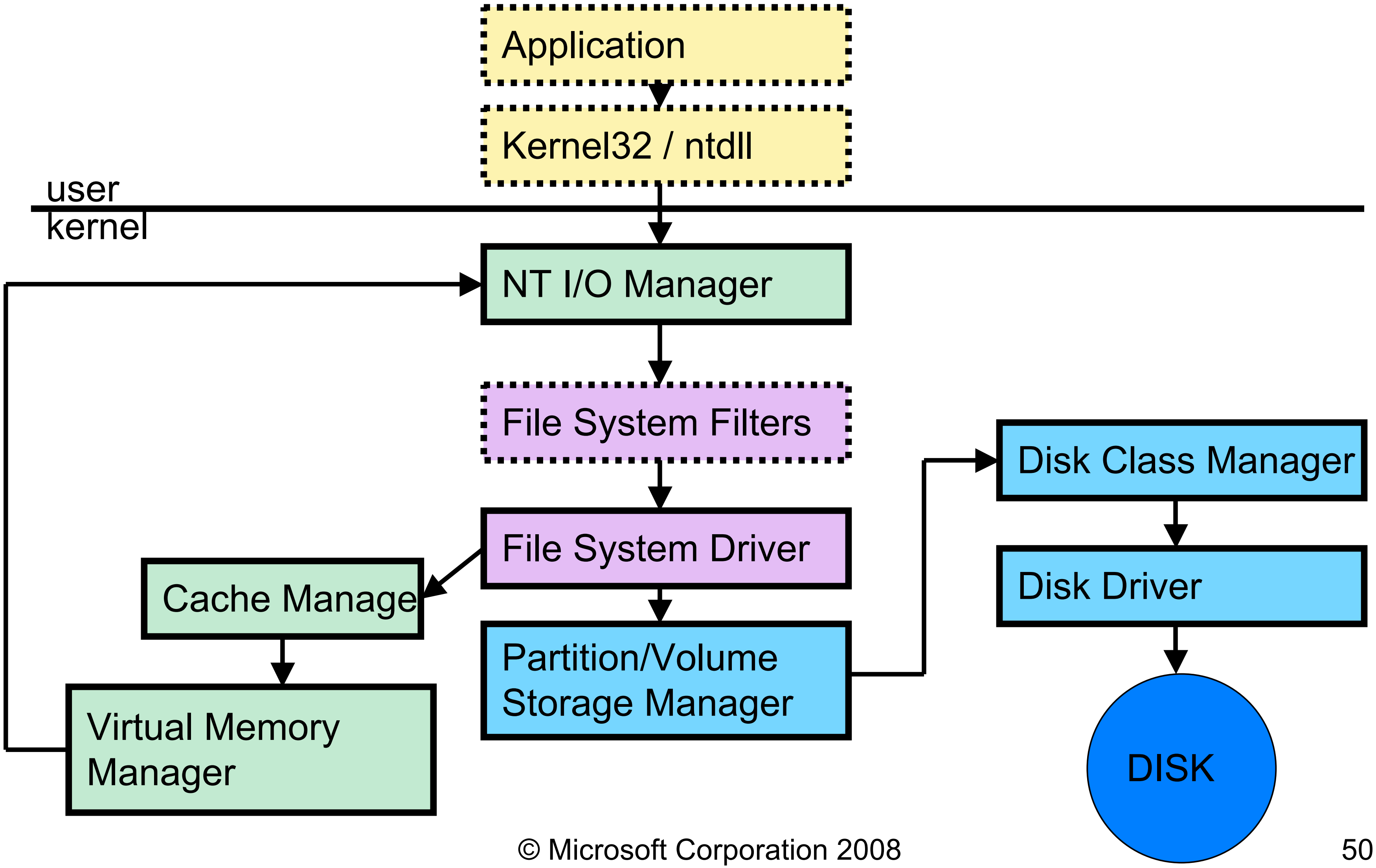
I/O Model

- Extensible filter-based I/O model with driver layering
- Standard device models for common device classes
- Support for notifications, tracing, journaling
- Configuration store remembers PnP decisions
- File caching is virtual, based on memory mapping
- Completely asynchronous model (with cancellation)
  - Multiple completion models:
    - wait on the file handle
    - wait on an event handle
    - specify a routine to be called at I/O completion (User-mode APC)
    - **use an I/O completion port**
    - poll status variable

Layering Drivers

- **Device objects attach one on top of another using IoAttachDevice* APIs creating “device stacks”**
  - I/O manager sends IRP to top of a stack
  - drivers store next lower device object in their private data structure
  - stack tear down done using IoDetachDevice and IoDeleteDevice
- **Device objects point to driver objects**
  - driver represent driver state, including dispatch table
  - drivers have device objects in multiple device stacks
- **File objects point to open files**
- **File systems are drivers which manage file objects for volumes (described by VolumeParameterBlocks)**

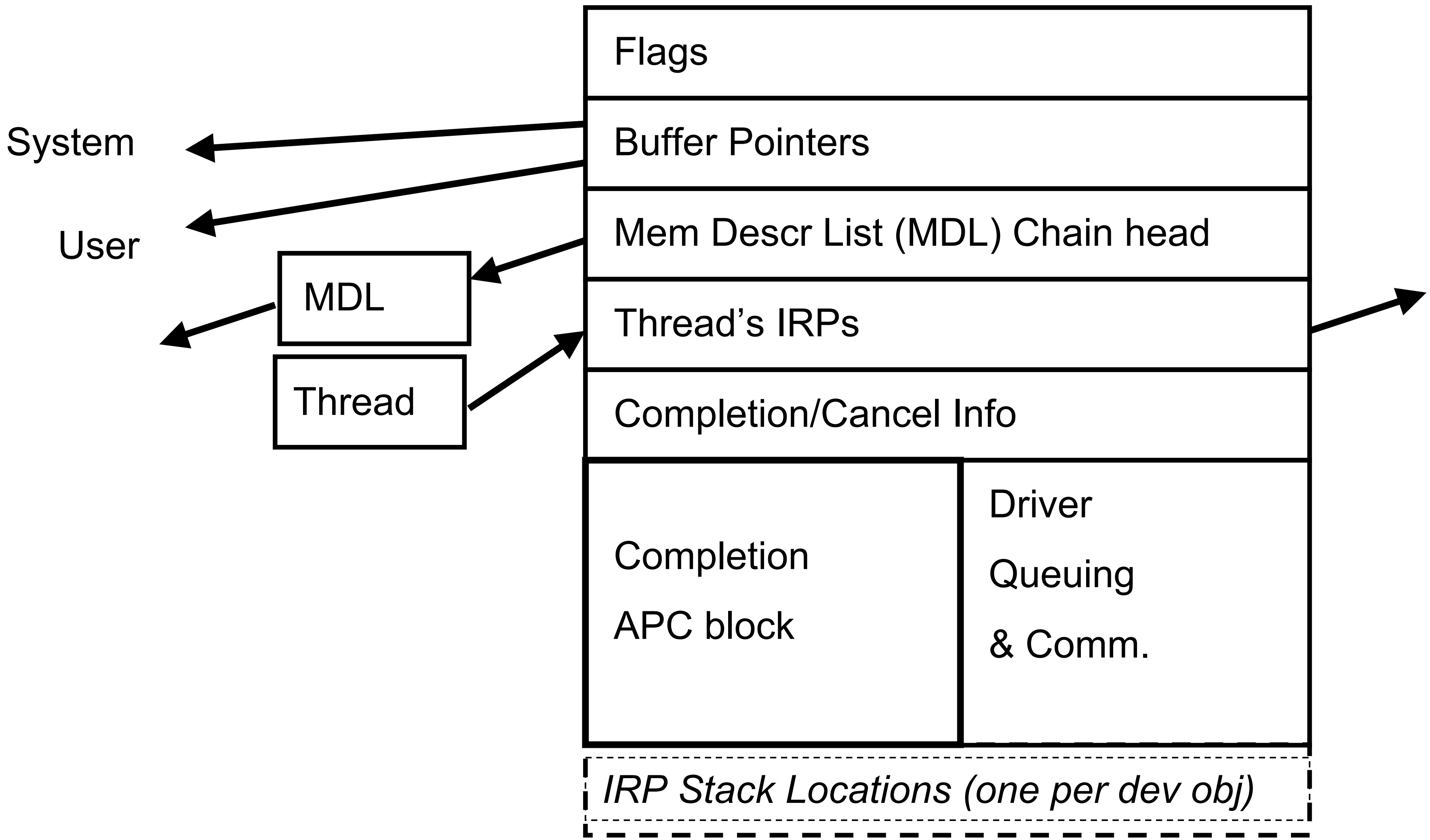
File System Device Stack



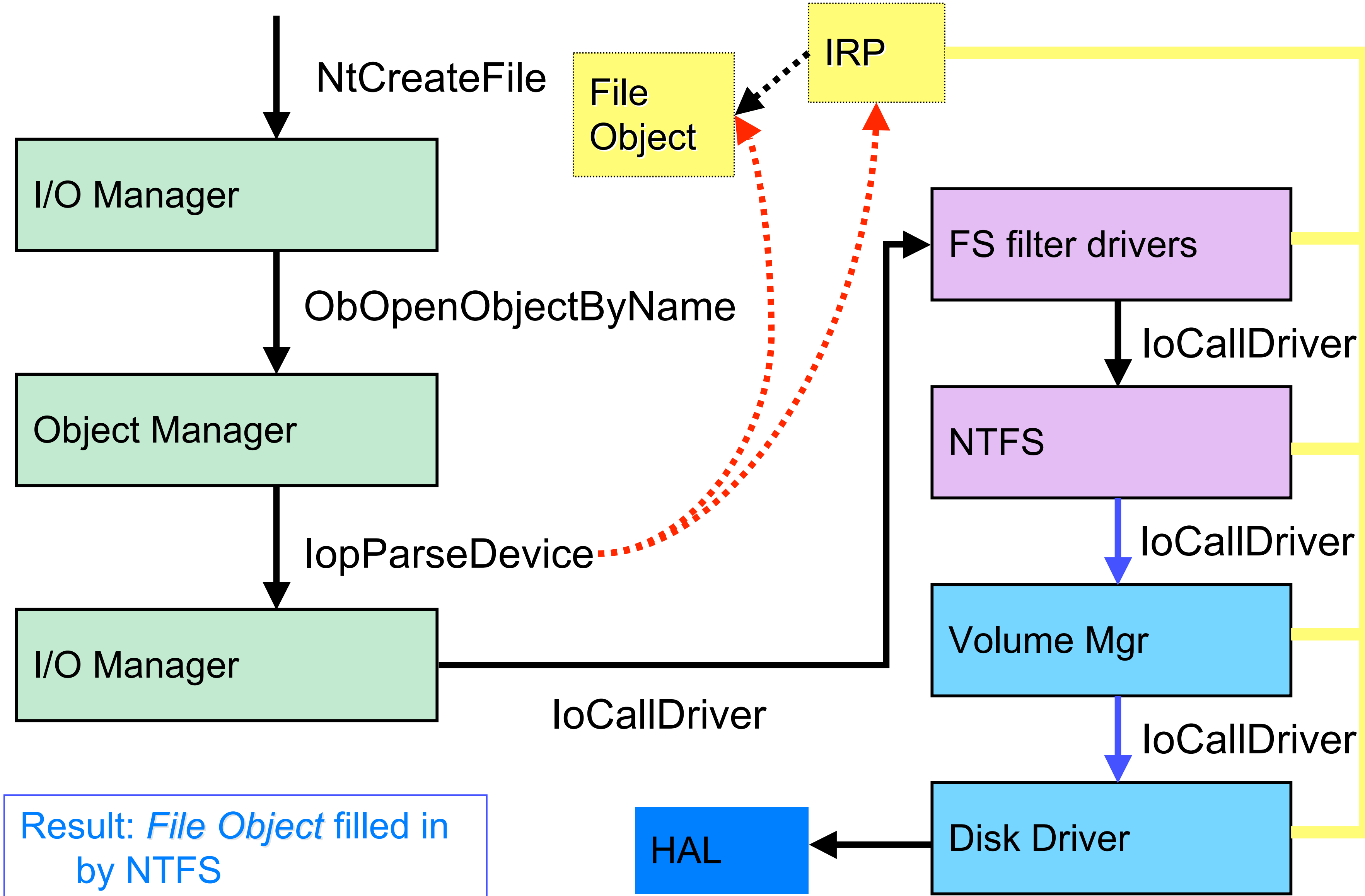
I/O Request Packet (IRP)

- I/O operations encapsulated in IRPs
- I/O requests travel down a driver stack in an IRP
- Each driver gets a stack location which contains parameters for that IO 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 – ***and can be cancelled***

IRP Fields



Flags
Buffer Pointers
System
User
Mem Descr List (MDL) Chain head
MDL
Thread's IRPs
Thread
Completion/Cancel Info
Completion
APC block
Driver
Queuing
& Comm.
IRP Stack Locations (one per dev obj)



Result: *File Object* filled in by NTFS

Asynchronous I/O

- I/O manager called to perform a standard operation
  - Open/create, read/write, ioctl, cleanup/close, …
- I/O operations represented by I/O Request Packet (IRP)
- I/O system uses IoCallDriver to call into a device stack
  - Figures out which device stack from name or top device object
- Drivers call IoCallDriver for next device object
  - Device object links to driver object, which has dispatch table
- Drivers keep calling down the device stack until:
  - I/O operation completes synchronously, or
  - Device driver decides to continue operation asynchronously
    - IRP queued to interrupt driven facilty or *posted* to a worker thread

IRP flow of control (asynchronous)

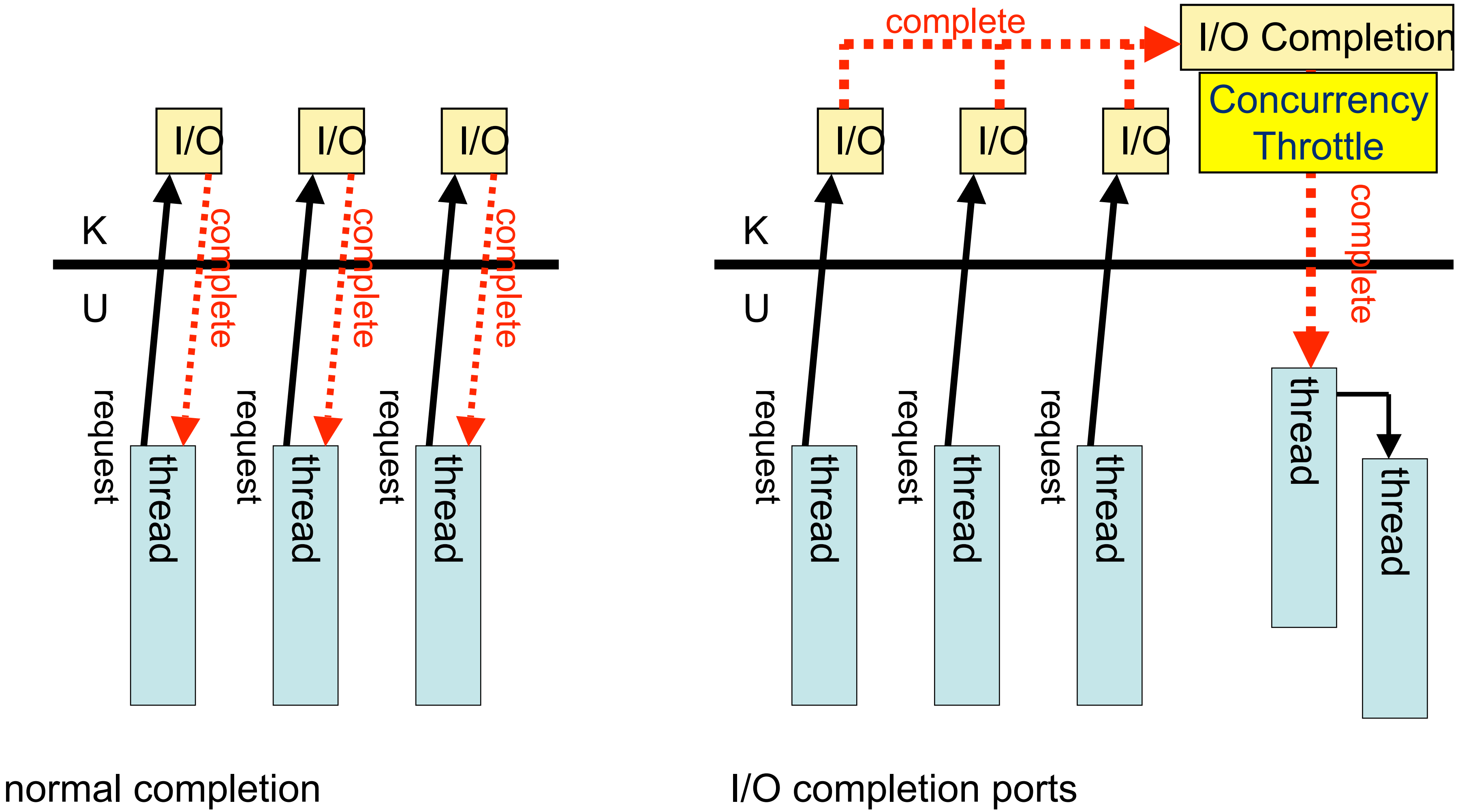
Eventually a driver decides to be asynchronous…

- Driver queues IRP for further processing
- Driver returns STATUS_PENDING up call stack
- Higher drivers may return all the way to user, or may wait for I/O to complete (synchronizing the stack)

Eventually a driver decides I/O is complete…

- Usually due to an interrupt/DPC completing I/O
- Each completion routine in device stack is called, possibly at DPC or in arbitrary thread context
- IRP turned into APC request delivered to original thread
- APC runs final completion, accessing process memory

I/O Completion Ports



NTFS Features

- Native file system for NT (replaced FAT and FAT32)
- Extends object manager / security reference monitor ACLs to files
- Many advanced features:
  - Quotas, journaling, objectids, encryption, compression, sparse files
- Supports multiple data streams per file
  - This is why ':' is not allowed in file names
  - Used primarily for MacOS resource forks on servers
  - NTFS implementation itself uses these data streams
- Directories use special $Index streams
- Common metadata duplicated
  - 'ls –l' very fast
- Equivalent of inodes has embedded data
- Integrity of metadata based on transaction logging
- Supports legacy
  - short names, attribute tunneling, Posix, hard links, symlinks?
- Unicode-based

NT Timeline

2/1989	**Design/Coding Begins**	
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 Ed. (& WS03SP1)
10/2006	**NT 6.0**	**Windows Vista (client)**
2/2008	**NT 6.0**	**Windows Server 2008**

Vista Kernel Security Changes

Code Integrity (x64) and BitLocker Encryption

- Signature verification of kernel modules
- Drives can be encrypted

Protected Processes

- Secures DRM processes

User Account Control (Allow or Deny?)

- Signature verification of kernel modules

Integrity Levels

- Provides a backup for ACLs by limiting write access to objects (and windows) regardless of permission
- Used by “low-rights” Internet Explorer

Vista Process/Memory Changes

Process Management changes

- Protected processes: move many steps into kernel and use for isolation (for DRM)

Memory Management improvements

- Improved prefetch at app launch/swap-in and resume from hibernation/sleep
- Kernel Address Space dynamically configured
- Support use of flash as write-through cache
- Address Space Randomization (executables and stacks) for improved virus resistance

Vista I/O

Memory Management improvements

- Improved prefetch at app launch/swap-in and resume from hibernation/sleep
- Kernel Address Space dynamically configured
- Support use of flash as write-through cache (compressed/encrypted)
- Session 0 is now isolated (runs systemwide services)
- Address Space Randomization (executables and stacks) for improved virus resistance

Vista Boot & Startup changes

Boot changes

- Boot.ini replaced by Boot Configuration Data registry hive
- BootMgr & Winload/WinResume replace NTLDR
- MemTest included as boot option

Startup changes

- Session Manager (SMSS) starts sessions in parallel
- Winlogon role ⇒Wininit & LSM (local session mgr)
- Console now runs in Session 1 not 0

Questions