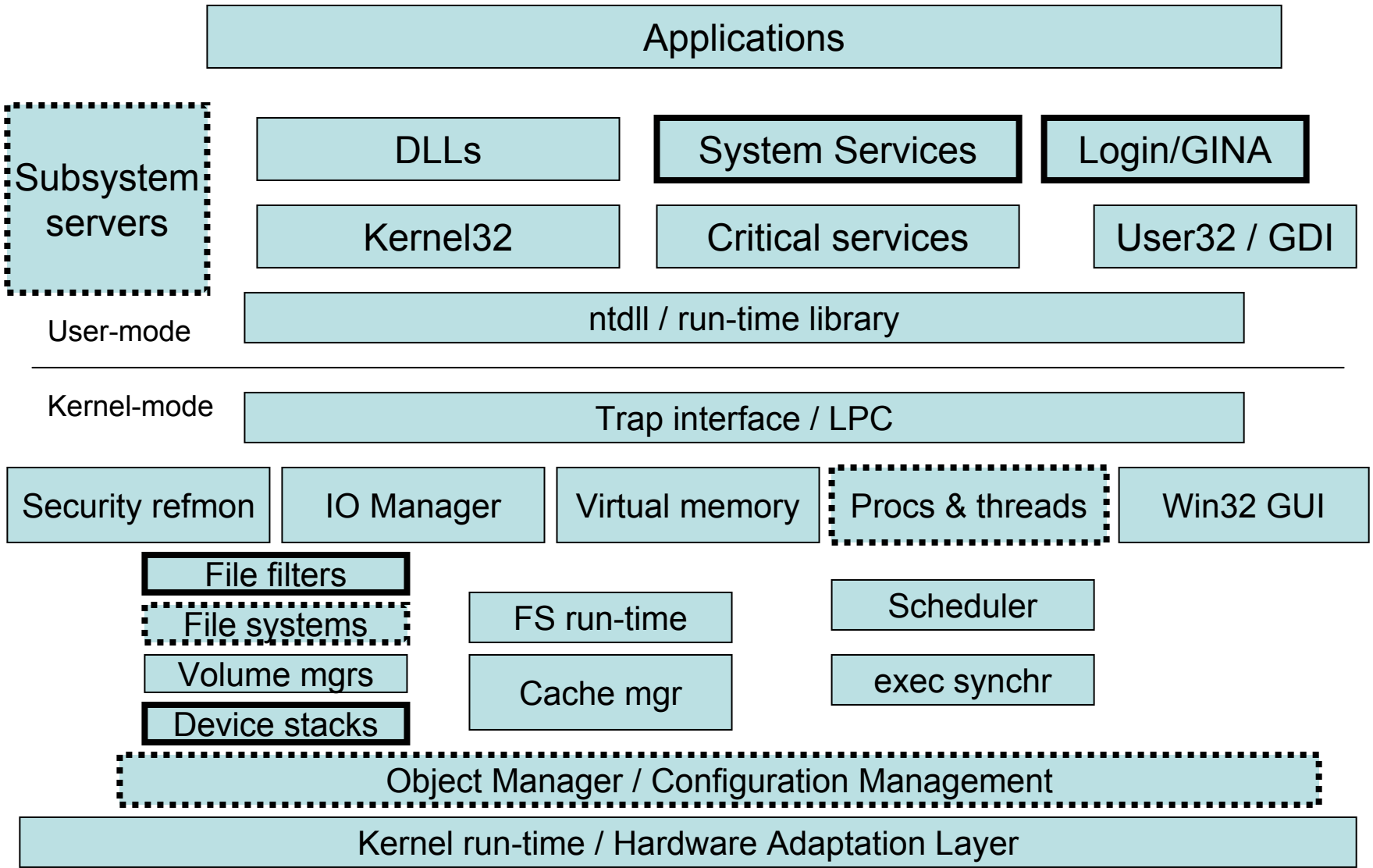
Windows Kernel Internals II

Processes, Threads, VirtualMemory

*University of Tokyo – July 2004**

Dave Probert, Ph.D.
Advanced Operating Systems Group
Windows Core Operating Systems Division
Microsoft Corporation

Windows Architecture



Process

Container for an address space and threads

Associated User-mode Process Environment Block (PEB)

Primary Access Token

Quota, Debug port, Handle Table etc

Unique process ID

Queued to the Job, global process list and Session list

MM structures like the WorkingSet, VAD tree, AWE etc

Thread

- Fundamental schedulable entity in the system
- Represented by ETHREAD that includes a KTHREAD
- Queued to the process (both E and K thread)
- IRP list
- Impersonation Access Token
- Unique thread ID
- Associated User-mode Thread Environment Block (TEB)
- User-mode stack
- Kernel-mode stack
- Processor Control Block (in KTHREAD) for cpu state when not running

Job

Container for multiple processes

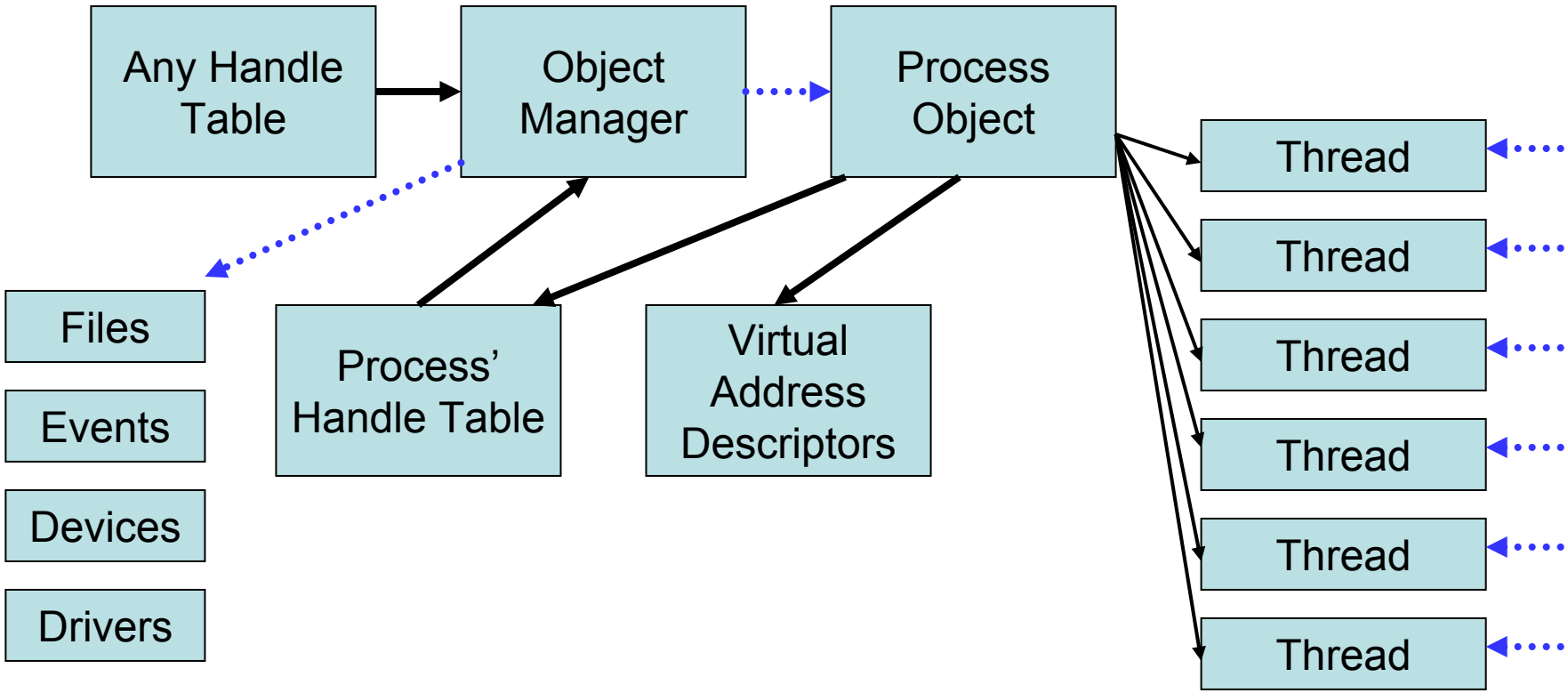
Queued to global job list, processes and jobs in the job set

Security token filters and job token

Completion ports

Counters, limits etc

Process/Thread structure



KPROCESS fields

DISPATCHER_HEADER Header
ULPTR DirectoryTableBase[2]
KGDTENTRY LdtDescriptor
KIDTENTRY Int21Descriptor
USHORT IopmOffset
UCHAR Iopl
volatile KAFFINITY ActiveProcessors
ULONG KernelTime
ULONG UserTime
LIST_ENTRY ReadyListHead
SINGLE_LIST_ENTRY SwapListEntry
LIST_ENTRY ThreadListHead
KSPIN_LOCK ProcessLock

KAFFINITY Affinity
USHORT StackCount
SCHAR BasePriority
SCHAR ThreadQuantum
BOOLEAN AutoAlignment
UCHAR State
BOOLEAN DisableBoost
UCHAR PowerState
BOOLEAN DisableQuantum
UCHAR IdealNode

EPROCESS fields

KPROCESS Pcb
EX_PUSH_LOCK ProcessLock
LARGE_INTEGER CreateTime
LARGE_INTEGER ExitTime
EX_RUNDOWN_REF RundownProtect
HANDLE UniqueProcessId
LIST_ENTRY ActiveProcessLinks
Quota Felds
SIZE_T PeakVirtualSize
SIZE_T VirtualSize
LIST_ENTRY SessionProcessLinks
PVOID DebugPort
PVOID ExceptionPort
PHANDLE_TABLE ObjectTable
EX_FAST_REF Token
PFN_NUMBER WorkingSetPage

KGUARDED_MUTEX AddressCreationLock
KSPIN_LOCK HyperSpaceLock
struct _ETHREAD *ForkInProgress
ULONG_PTR HardwareTrigger;
PMM_AVL_TABLE PhysicalVadRoot
PVOID CloneRoot
PFN_NUMBER NumberOfPrivatePages
PFN_NUMBER NumberOfLockedPages
PVOID Win32Process
struct _EJOB *Job
PVOID SectionObject
PVOID SectionBaseAddress
PEPROCESS_QUOTA_BLOCK QuotaBlock

EPROCESS fields

PPAGEFAULT_HISTORY WorkingSetWatch
HANDLE Win32WindowStation
HANDLE InheritedFromUniqueProcessId
PVOID LdtInformation
PVOID VadFreeHint
PVOID VdmObjects
PVOID DeviceMap
PVOID Session
UCHAR ImageFileName[16]
LIST_ENTRY JobLinks
PVOID LockedPagesList
LIST_ENTRY ThreadListHead
ULONG ActiveThreads
PPEB Peb
IO Counters

PVOID AweInfo
MMSUPPORT Vm
Process Flags
NTSTATUS ExitStatus
UCHAR PriorityClass
MM_AVL_TABLE VadRoot

KTHREAD fields

DISPATCHER_HEADER Header
LIST_ENTRY MutantListHead
PVOID InitialStack, StackLimit
PVOID KernelStack
KSPIN_LOCK ThreadLock
ULONG ContextSwitches
volatile UCHAR State
KIRQL WaitIrql
KPROC_MODE WaitMode
PVOID Teb
KAPC_STATE ApcState
KSPIN_LOCK ApcQueueLock
LONG_PTR WaitStatus
PRKWAIT_BLOCK WaitBlockList
BOOLEAN Alertable, WaitNext
UCHAR WaitReason
SCHAR Priority

UCHAR EnableStackSwap
volatile UCHAR SwapBusy
LIST_ENTRY WaitListEntry
NEXT SwapListEntry
PRKQUEUE Queue
ULONG WaitTime
SHORT KernelApcDisable
SHORT SpecialApcDisable
KTIMER Timer
KWAIT_BLOCK WaitBlock[N+1]
LIST_ENTRY QueueListEntry
UCHAR ApcStateIndex
BOOLEAN ApcQueueable
BOOLEAN Preempted
BOOLEAN ProcessReadyQueue
BOOLEAN KernelStackResident

KTHREAD fields cont.

UCHAR IdealProcessor
volatile UCHAR NextProcessor
SCHAR BasePriority
SCHAR PriorityDecrement
SCHAR Quantum
BOOLEAN SystemAffinityActive
CCHAR PreviousMode
UCHAR ResourceIndex
UCHAR DisableBoost
KAFFINITY UserAffinity
PKPROCESS Process
KAFFINITY Affinity
PVOID ServiceTable
PKAPC_STATE ApcStatePtr[2]
KAPC_STATE SavedApcState
PVOID CallbackStack
PVOID Win32Thread

PKTRAP_FRAME TrapFrame
ULONG KernelTime, UserTime
PVOID StackBase
KAPC SuspendApc
KSEMAPHORE SuspendSema
PVOID TlsArray
LIST_ENTRY ThreadListEntry
UCHAR LargeStack
UCHAR PowerState
UCHAR Iopl
CCHAR FreezeCnt, SuspendCnt
UCHAR UserIdealProc
volatile UCHAR DeferredProc
UCHAR AdjustReason
SCHAR AdjustIncrement

ETHREAD fields

KTHREAD tcb

Timestamps

LPC locks and links

CLIENT_ID Cid

ImpersonationInfo

IrpList

pProcess

StartAddress

Win32StartAddress

ThreadListEntry

RundownProtect

ThreadPushLock

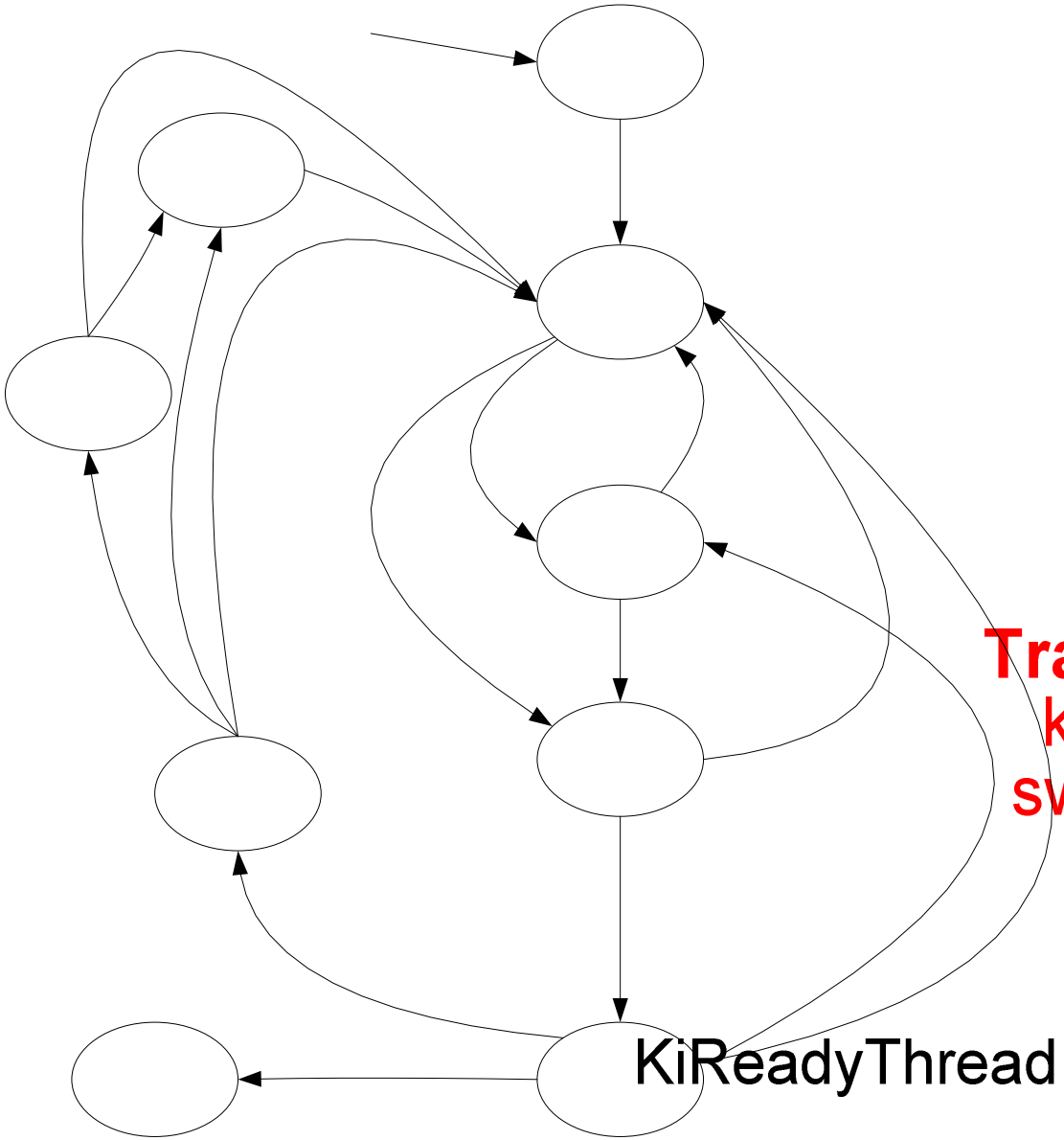
Process Synchronization

ProcessLock – Protects thread list, token

RundownProtect – Cross process address space, image section and handle table references

Token, Prefetch – Uses fast referencing

Token, Job – Torn down at last process dereference without synchronization



Thread scheduling states

KeInitThr

Transition
k stack
swapped

KiInsertDeferred

Ready

Thread scheduling states

- Main quasi-states:
  - Ready – able to run
  - Running – current thread on a processor
  - Waiting – waiting an event
- For scalability Ready is three real states:
  - DeferredReady – queued on any processor
  - Standby – will be imminently start Running
  - Ready – queue on target processor by priority
- Goal is granular locking of thread priority queues
- Red states related to swapped stacks and processes

Process Lifetime

Created as an empty shell

Address space created with only ntdll and the main image unless forked

Handle table created empty or populated via duplication from parent

Process is partially destroyed on last thread exit

Process totally destroyed on last dereference

Thread Lifetime

Created within a process with a CONTEXT record

Starts running in the kernel but has a trap frame to return to use mode

Kernel queues user APC to do ntdll initialization

Terminated by a thread calling NtTerminateThread/Process

Summary: Native NT Process APIs

NtCreateProcess()
NtTerminateProcess()
NtQueryInformationProcess()
NtSetInformationProcess()
NtGetNextProcess()
NtGetNextThread()
NtSuspendProcess()
NtResumeProcess()

NtCreateThread()
NtTerminateThread()
NtSuspendThread()
NtResumeThread()
NtGetContextThread()
NtSetContextThread()
NtQueryInformationThread()
NtSetInformationThread()
NtAlertThread()
NtQueueApcThread()

Virtual Memory Manager
Features

- Provides 4 GB flat virtual address space (IA32)
- Manages process address space
- Handles pagefaults
- Manages process working sets
- Manages physical memory
- Provides memory-mapped files
- Allows pages shared between processes
- Facilities for I/O subsystem and device drivers
- Supports file system cache manager

Virtual Memory Manager

NT Internal APIs

NtCreatePagingFile

NtAllocateVirtualMemory (Proc, Addr, Size, Type, Prot)

- Process: handle to a process
- Protection: NOACCESS, EXECUTE, READONLY, READWRITE, NOCACHE
- Flags: COMMIT, RESERVE, PHYSICAL, TOP_DOWN, RESET, LARGE_PAGES, WRITE_WATCH

NtFreeVirtualMemory(Process, Address, Size, FreeType)

- FreeType: DECOMMIT or RELEASE

NtQueryVirtualMemory

NtProtectVirtualMemory

Virtual Memory Manager

NT Internal APIs

Pagefault

NtLockVirtualMemory, NtUnlockVirtualMemory

- locks a region of pages within the working set list
- requires PROCESS_VM_OPERATION on target process and SeLockMemoryPrivilege

NtReadVirtualMemory, NtWriteVirtualMemory (

Proc, Addr, Buffer, Size)

NtFlushVirtualMemory

Virtual Memory Manager

NT Internal APIs

NtCreateSection

- **creates a section but does not map it**

NtOpenSection

- opens an existing section

NtQuerySection

- query attributes for section

NtExtendSection

NtMapViewOfSection (Sect, Proc, Addr, Size, …)

NtUnmapViewOfSection

Virtual Memory Manager

NT Internal APIs

APIs to support AWE (Address Windowing Extensions)

- Private memory only
- Map only in current process
- Requires LOCK_VM privilege

NtAllocateUserPhysicalPages (Proc, NPages, &PFNs[])

NtMapUserPhysicalPages (Addr, NPages, PFNs[])

NtMapUserPhysicalPagesScatter

NtFreeUserPhysicalPages (Proc, &NPages, PFNs[])

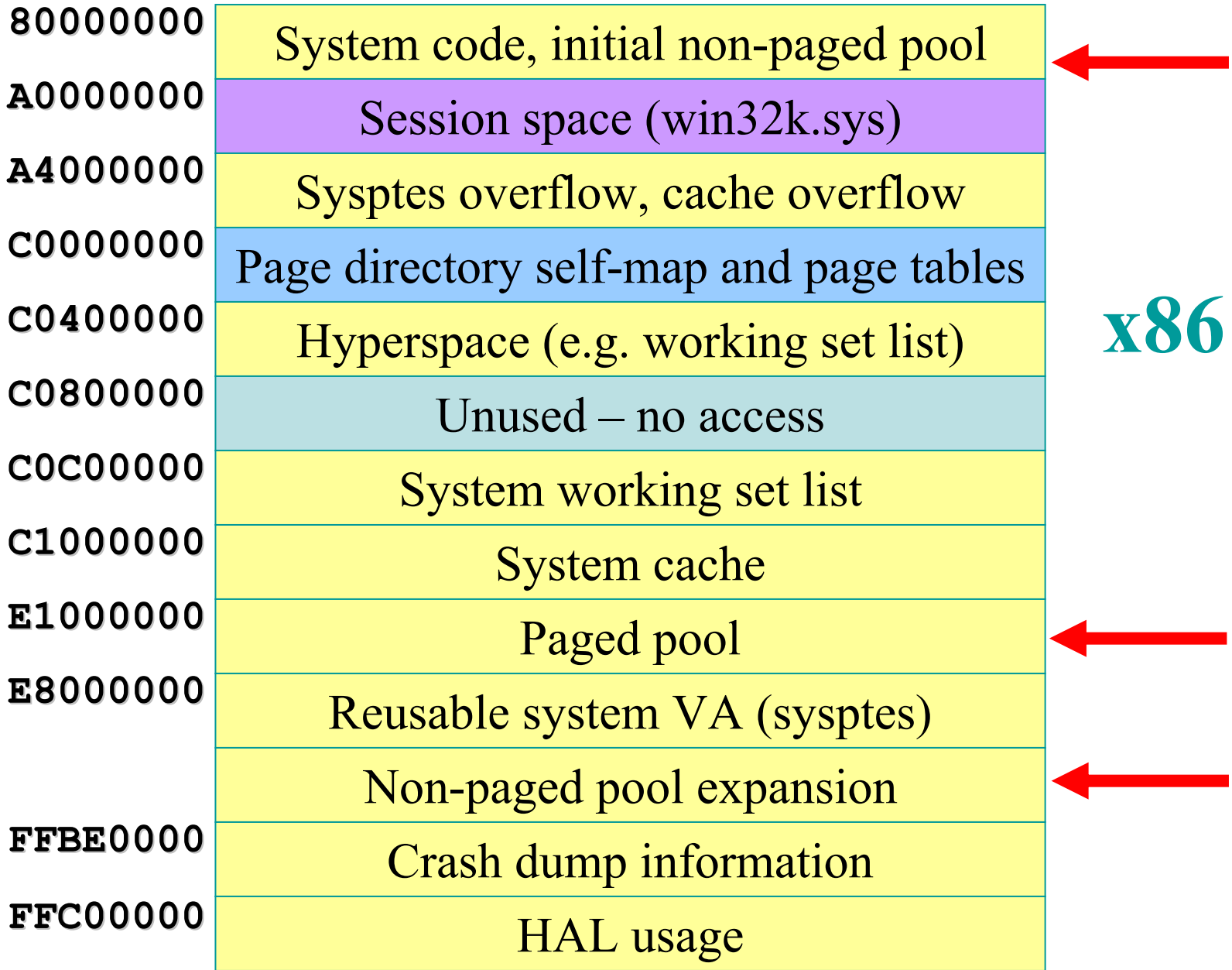
NtResetWriteWatch

NtGetWriteWatch

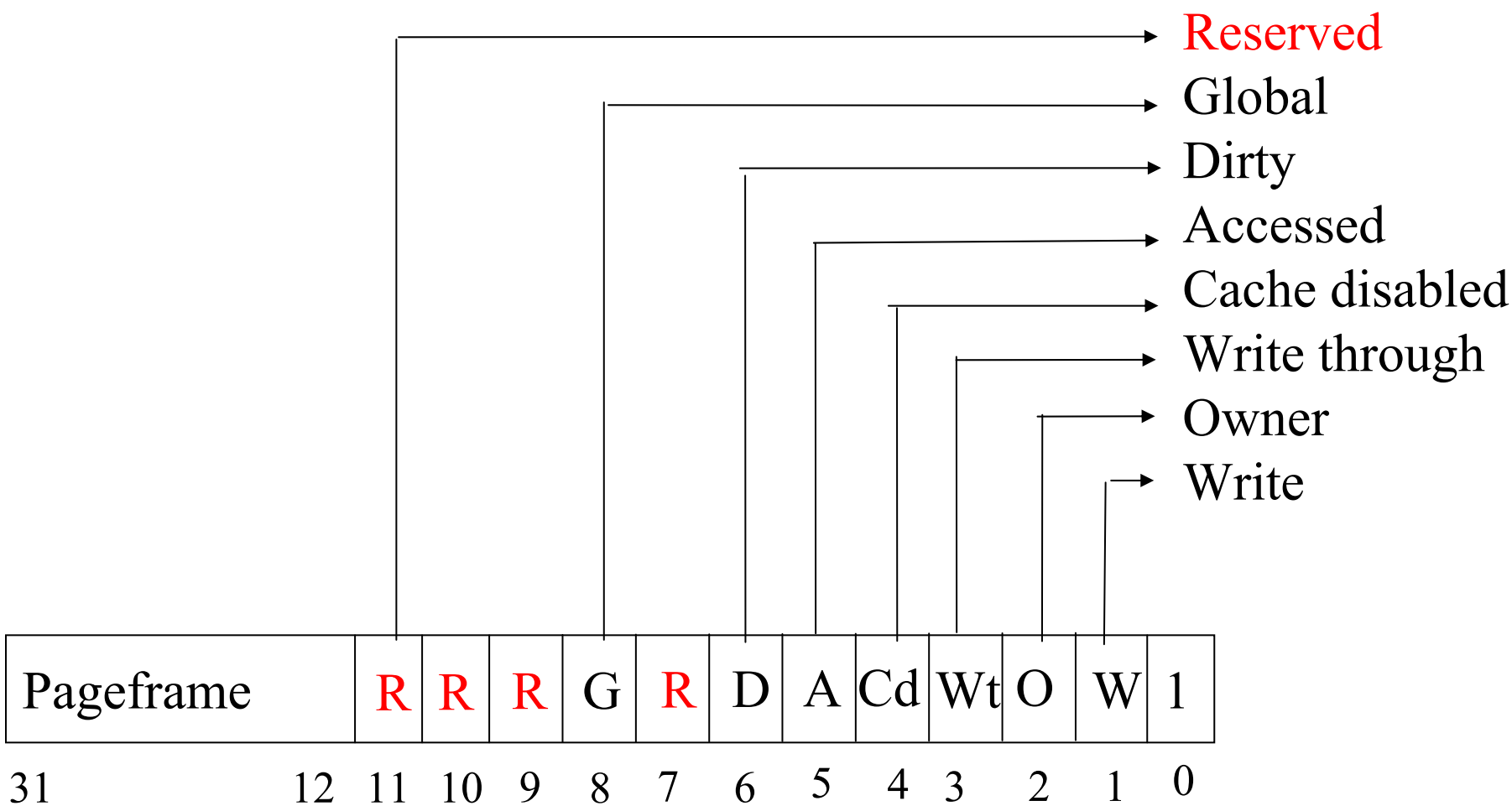
Read out dirty bits for a section of memory since last reset

Allocating kernel memory (pool)

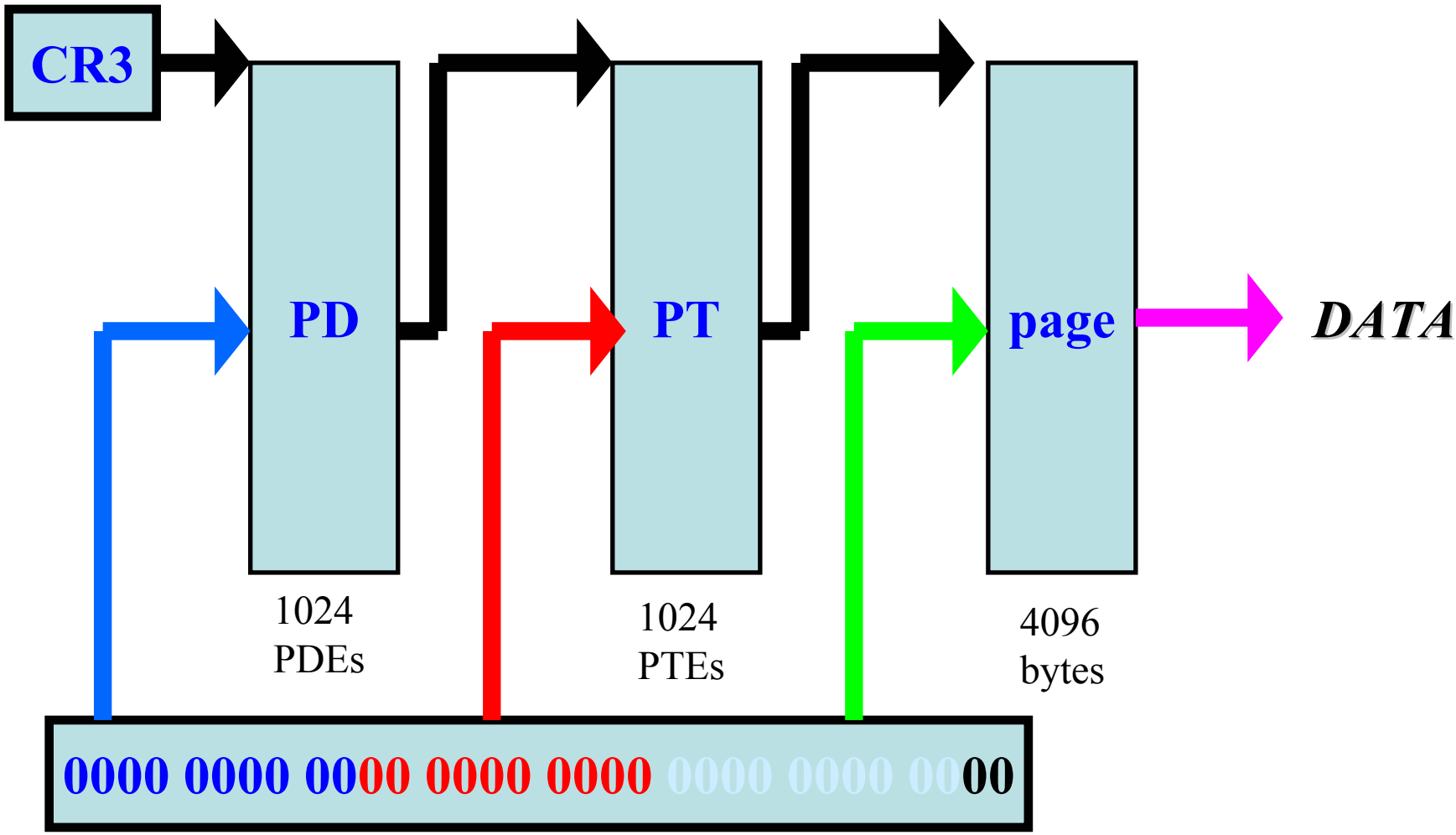
- Tightest x86 system resource is KVA
  - Kernel Virtual Address space
- Pool allocates in small chunks:
  - < 4KB: 8B granulariy
  - >= 4KB: page granularity
- Paged and Non-paged pool
  - Paged pool backed by pagefile
- Special pool used to find corruptors
- Lots of support for debugging/diagnosis



Valid x86 Hardware PTEs



Virtual Address Translation



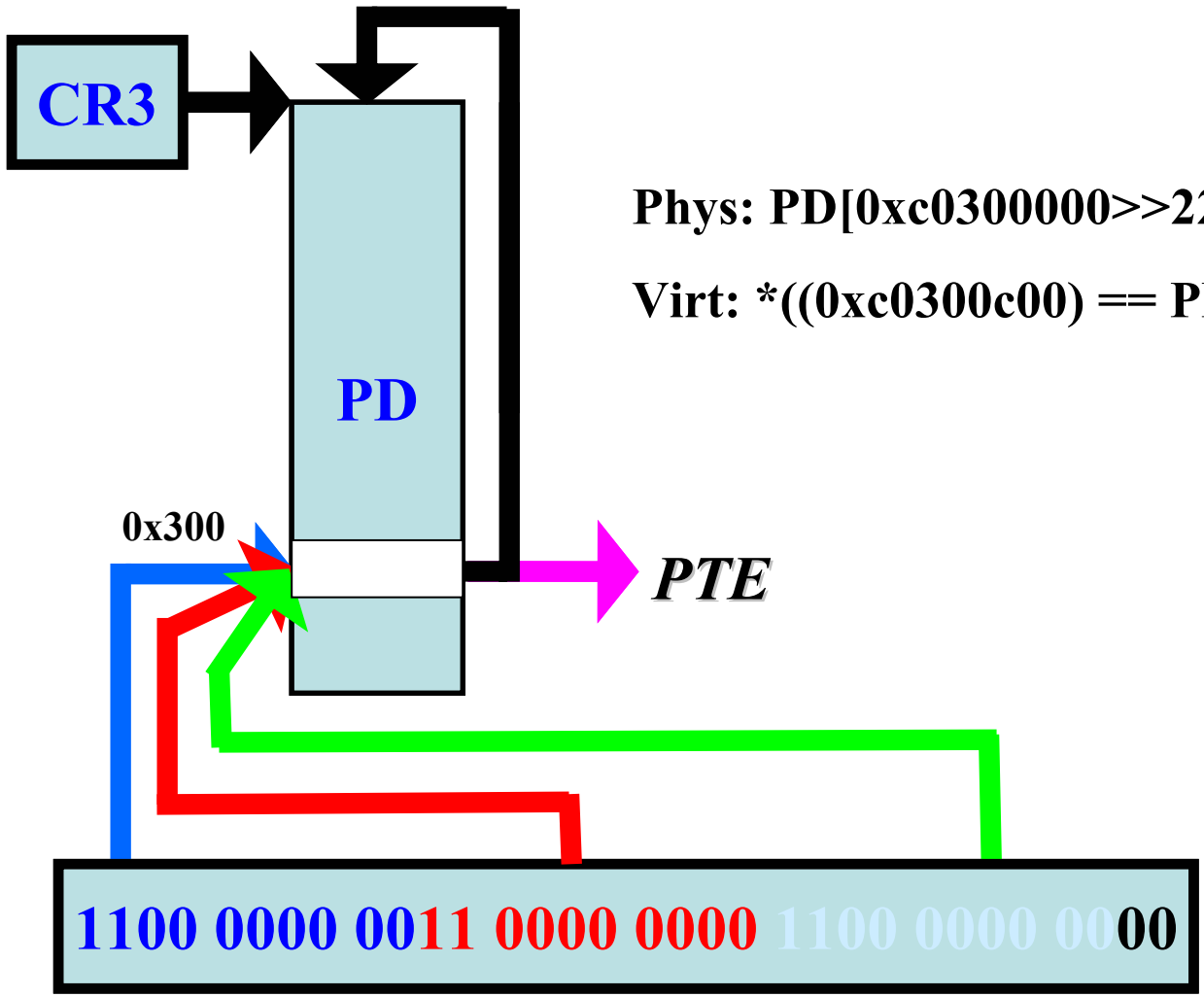
Self-mapping page tables

- Page Table Entries (PTEs) and Page Directory Entries (PDEs) contain **Physical Frame Numbers (PFNs)**
  - But Kernel runs with **Virtual Addresses**
- To access PDE/PTE from kernel use the self-map for the current process:

  PageDirectory[0x300] uses PageDirectory as PageTable
  - GetPdeAddress(va): 0xc0300000[va>>20]
  - GetPteAddress(va): 0xc0000000[va>>10]
- PDE/PTE formats are compatible!
- Access another process VA via thread ‘attach’

Self-mapping page tables

Virtual Access to PageDirectory[0x300]

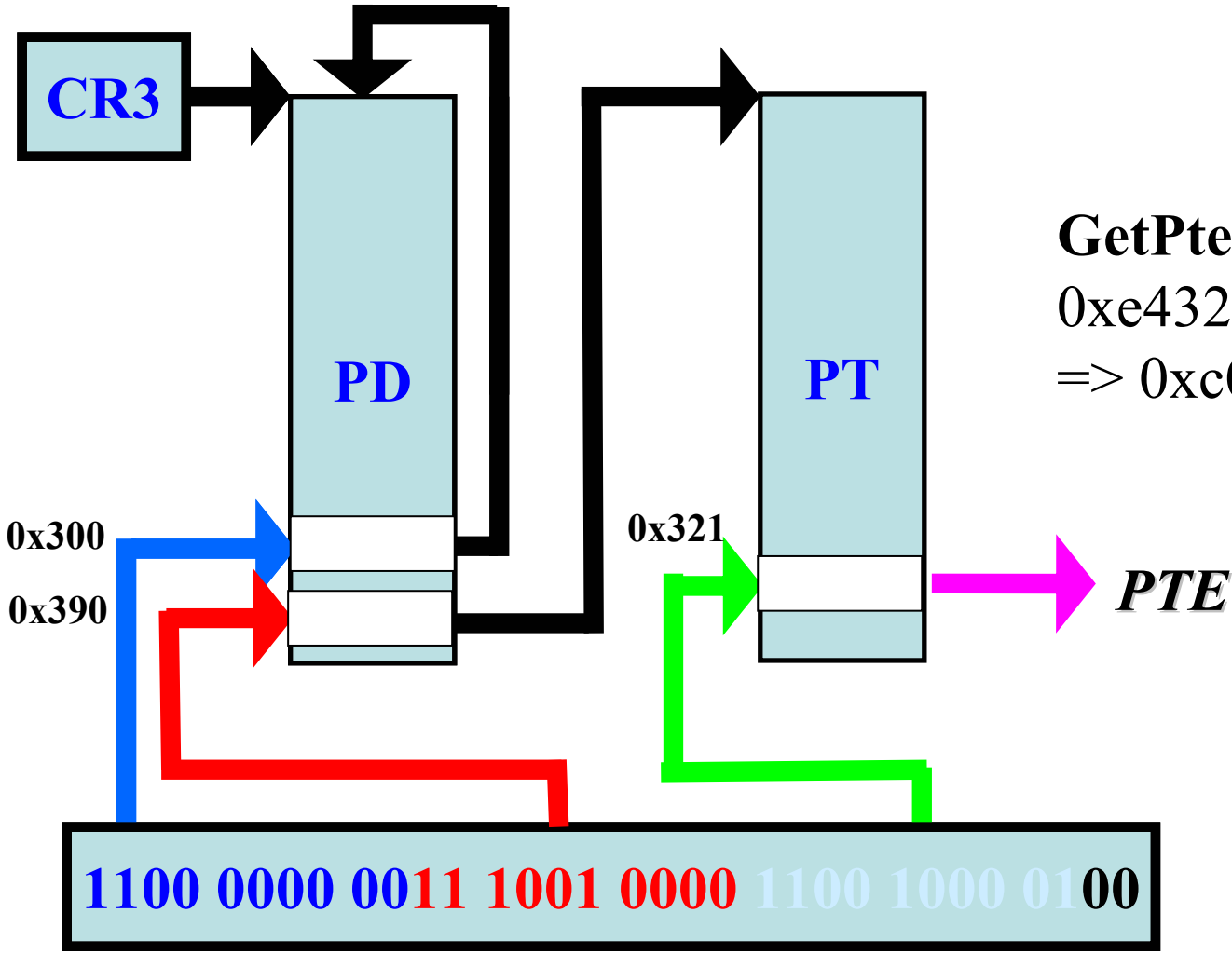


Phys: PD[0xc0300000>>22] = PD

Virt: *((0xc0300c00) == PD

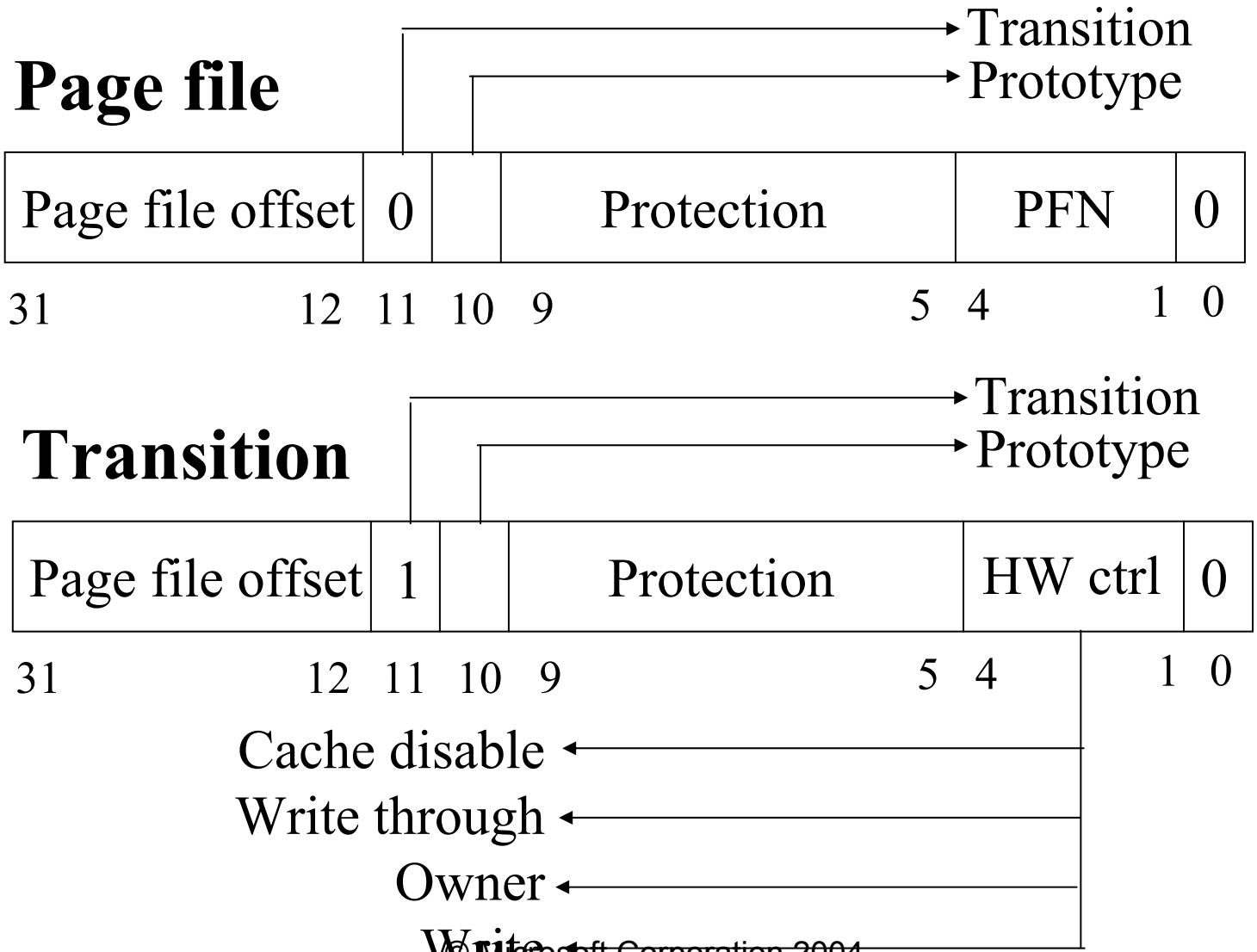
Self-mapping page tables

Virtual Access to PTE for va 0xe4321000



GetPteAddress:
0xe4321000
=> 0xc0390c84

x86 Invalid PTEs

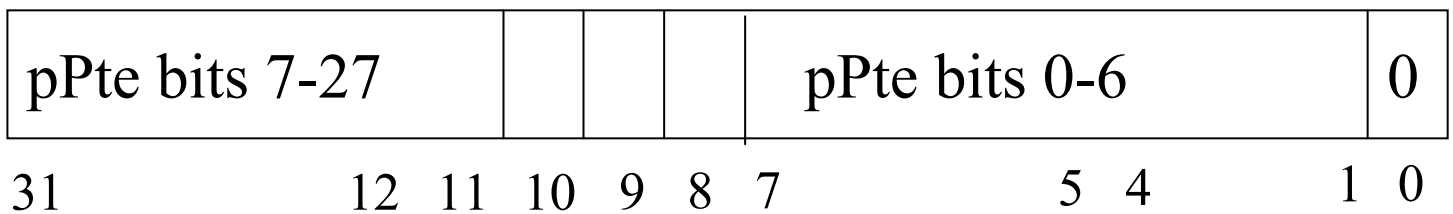


x86 Invalid PTEs

Demand zero: Page file PTE with zero offset and PFN

Unknown: PTE is completely zero or Page Table doesn't exist yet. Examine VADs.

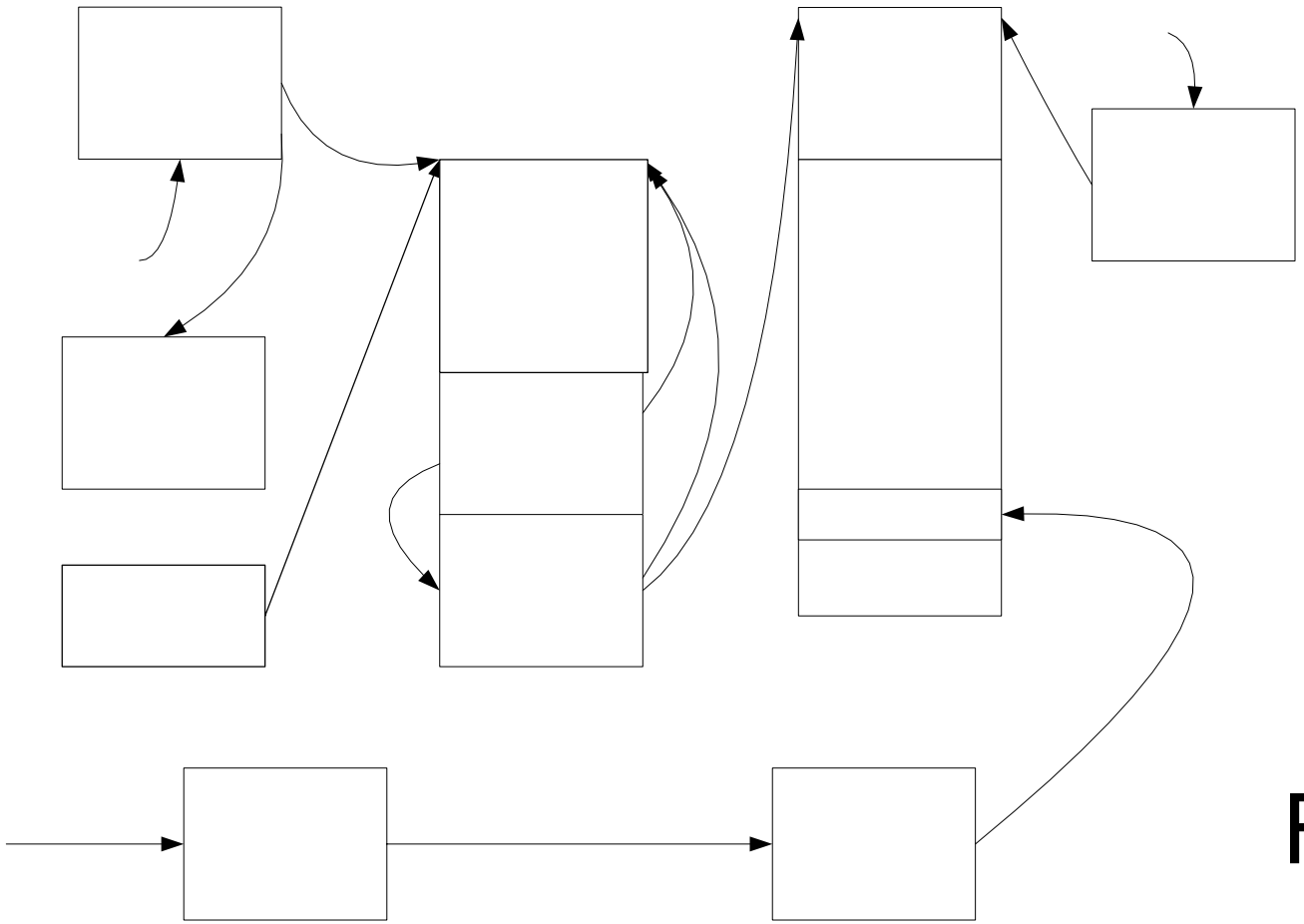
Pointer to Prototype PTE



Prototype PTEs

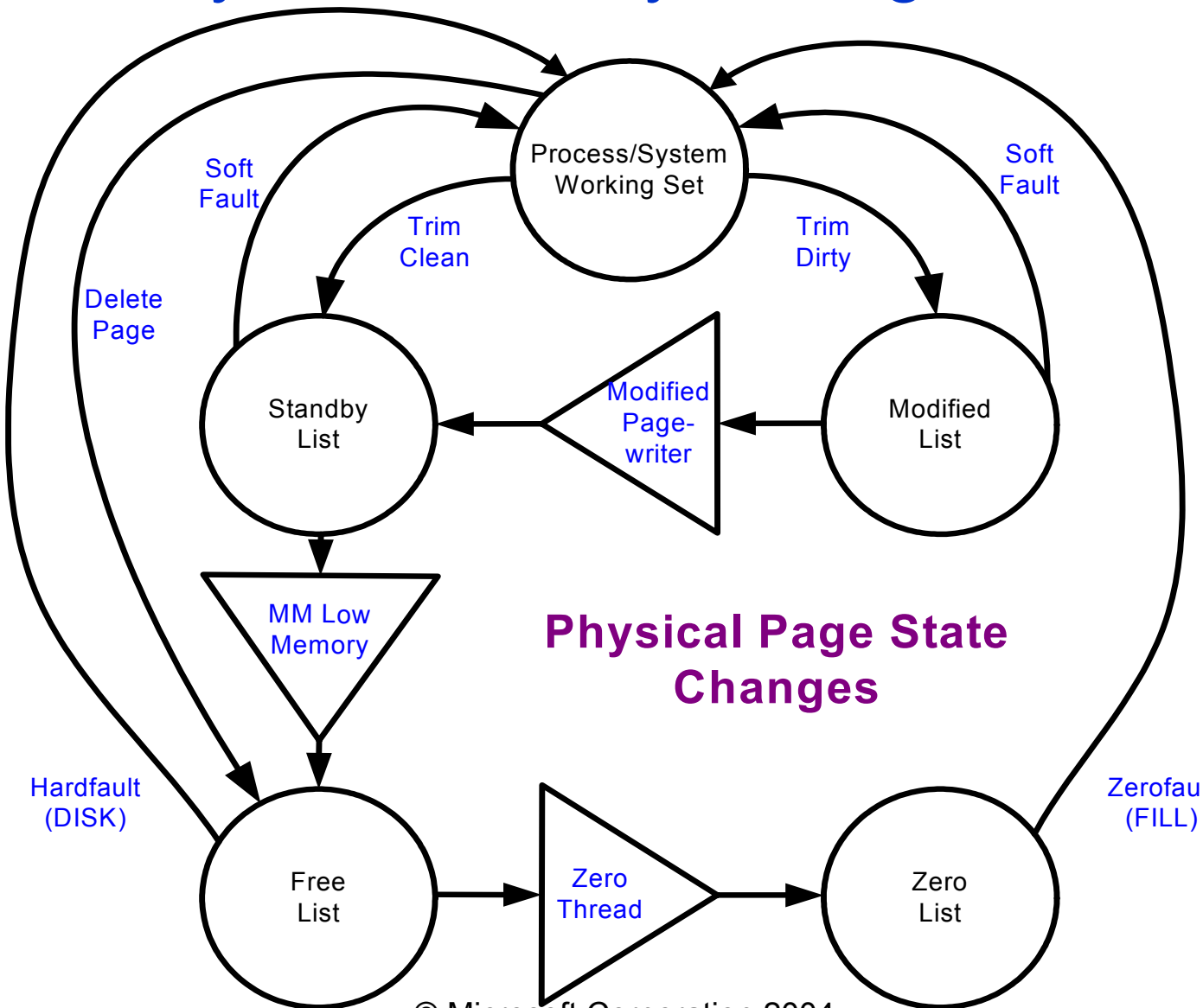
- Kept in array in the *segment* structure associated with section objects
- Six PTE states:
  - Active/valid
  - Transition
  - Modified-no-write
  - Demand zero
  - Page file
  - Mapped file

Shared Memory Data Structures



File Ob

Physical Memory Management



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

Managing Working Sets

Aging pages: Increment age counts for pages which haven't been accessed

Estimate unused pages: count in working set and keep a global count of estimate

When *getting* tight on memory: replace rather than add pages when a fault occurs in a working set with significant unused pages

When memory *is* tight: reduce (trim) working sets which are above their maximum

Balance Set Manager: periodically runs Working Set Trimmer, also swaps out kernel stacks of long-waiting threads

Discussion