Windows Kernel Internals
Object Manager

David B. Probert, Ph.D.
Windows Kernel Development
Microsoft Corporation

Kernel Object Manager (OB)

- Provides underlying NT namespace
- Unifies kernel data structure referencing
- Unifies user-mode referencing via handles
- Simplifies resource charging
- Central facility for security protection

NT Name Space

- ¥
    - arcname¥
    - BaseNamedObjects¥
    - callback¥
    - device
        - HardDisk0¥
            - dr0
    - driver¥
    - FileSystem¥
        - ntfs
    - KernelObjects¥
    - KnownDlls¥
    - ObjectTypes¥
    - RPC Control¥
    - Windows¥
        - apiport
        - SbApiPort
        - WindowsStations¥
            - winsta0

¥ObjectTypes

Adapter
Callback
Controller
DebugObject
Desktop
Device
Directory
Driver
Event
EventPair

File
IoCompletion
Job
Key
KeyedEvent
Mutant
Port
Process
Profile
Section

Semaphore
SymbolicLink
Thread
Timer
Token
Type
WaitablePort
WindowsStation
WMIGuid

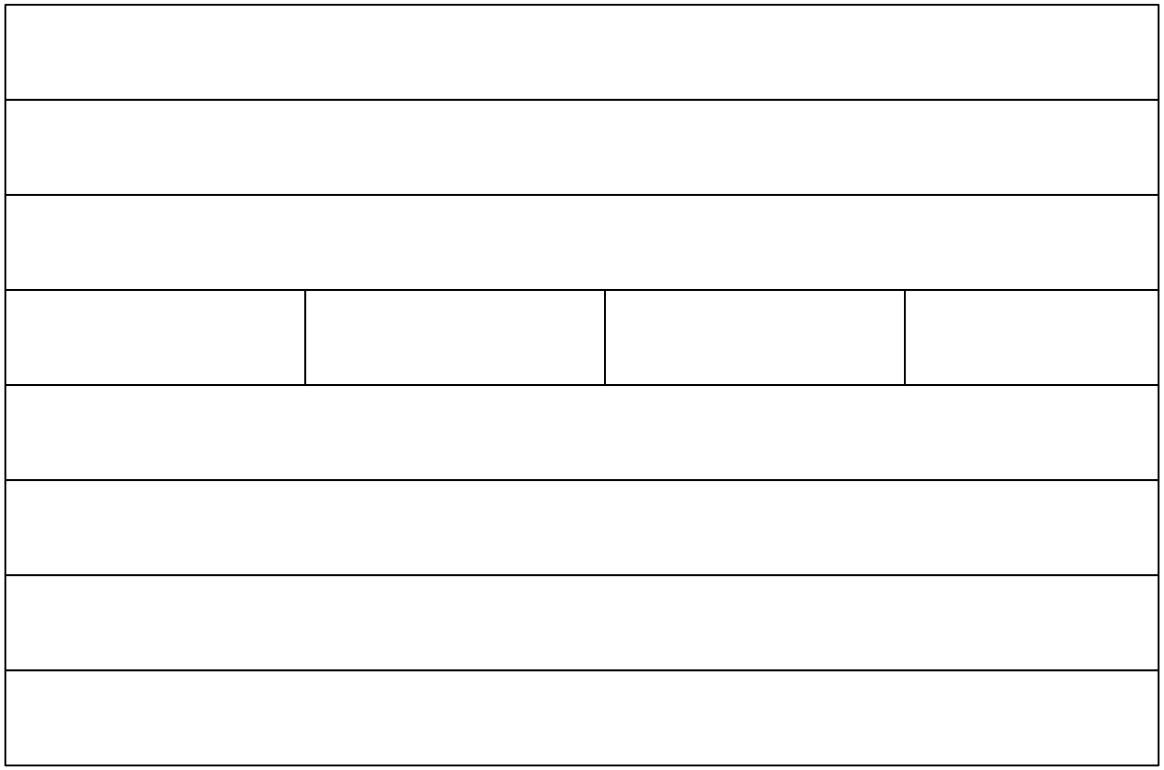
¥ObjectTypes

Adapter
Callback
Controller
DebugObject
Desktop
Device
Directory
Driver
Event
EventPair

File
IoCompletion
Job
Key
KeyedEvent
Mutant
Port
Process
Profile
Section

Semaphore
SymbolicLink
Thread
Timer
Token
Type
WaitablePort
WindowsStation
WMIGuid

OBJECT_HEADER



Generic object services

- namespace ops: directories, symlinks
- NtQueryObject
- NtQuery/SetSecurityObject
- NtWaitForSingle/MultipleObjects
- ObOpenObjectByName/Pointer
- ObReferenceObjectByName/Handle
- NtDuplicateObject
- NtClose
- ObDereferenceObject

OBJECT_DIRECTORY

OBJECT_DIRECTORY

OBJECT_DIRECTORY_ENTRY *pHashBuckets[ ]
Lock
pDeviceMap
SessionId

OBJECT_DIRECTORY_ENTRY

OBJECT_DIRECTORY_ENTRY *pChainLink
pObject

ObpLookupDirectoryEntry(pD, s)

```
object = NULL
idx = HASH(s)
pE = pD->HashBuckets[idx]
LockDirectoryShared(pD)
while (pE && !eqs(s, pE->Object->Name))
    pE = pE->pChainLink
if (pE)
    ObpReferenceObject(object = pE->Object)
UnlockDirectory(pD)
return object
```

Object Methods

OPEN:	Create/Open/Dup/Inherit handle
CLOSE:	Called when each handle closed
DELETE:	Called on last dereference
PARSE:	Called looking up objects by name
SECURITY:	Usually *SeDefaultObjectMethod*
QUERYNAME:	Return object-specific name
OKAYTOCLOSE:	Give veto on handle close

Object Manager Types

Directory - namespace object

Implementation hardwired

SymbolicLink - namespace object

DeleteProcedure = ObpDeleteSymbolicLink

ParseProcedure = ObpParseSymbolicLink

Type **- represent object types**

DeleteProcedure = ObpDeleteObjectType

Object Manager lookups

ObpLookupObjectName(Name,Context)

- Search a directory for specified object name
- Use ObpLookupDirectoryEntry() on Directories
- Otherwise call object-specific ParseProcedure
  - Implements symbolic links (SymbolicLink type)
  - Implements file systems (DeviceObject type)

I/O Manager Types

Adapter - ADAPTER_OBJECT

Controller - CONTROLLER_OBJECT

Device - DEVICE_OBJECT

- **ParseProcedure = IopParseDevice**
- **DeleteProcedure = IopDeleteDevice**
- **SecurityProcedure = IopGetSetSecurityObject**

Driver - DRIVER_OBJECT

- **DeleteProcedure = IopDeleteDriver**

IoCompletion - KQUEUE

- **DeleteProcedure = IopDeleteIoCompletion**

I/O Manager File Type

File - FILE_OBJECT

CloseProcedure = IopCloseFile
DeleteProcedure = IopDeleteFile
ParseProcedure = IopParseFile
SecurityProcedure = IopGetSetSecurityObject
QueryNameProcedure = IopQueryName

IopParseDevice

(DeviceObject, Context, RemainingName)

- Call SeAccessCheck()
- If (!*RemainingName) directDeviceOpen = TRUE
- For file opens, get Volume from DeviceObject
- Update references on Volume and DeviceObject
- Construct an I/O Request Packet (IRP)
- FileObject = ObCreateObject(IoFileObjectType)
- Initialize FileObject
- Initiate I/O via IoCallDriver(VolumeDevice, IRP)
- Wait for I/O to signal FileObject->Event
- Return the FileObject to caller

FILE_OBJECT

pDeviceObject	Flags
pVolumeParameterBlock	CurrentByteOffset
pFsContext/pFsContext2	FinalNTStatus
pSectionObjectPointers	nWaiters
pPrivateCacheMap	nBusy
pRelatedFileObject	Lock
	Event
	pIOCompletionContext

File Object (FO) flags

FO_FILE_OPEN
FO_SYNCHRONOUS_IO
FO_ALERTABLE_IO
FO_REMOTE_ORIGIN
FO_WRITE_THROUGH
FO_SEQUENTIAL_ONLY
FO_CACHE_SUPPORTED
FO_NAMED_PIPE
FO_STREAM_FILE
FO_MAILSLOT
FO_FILE_MODIFIED
FO_FILE_SIZE_CHANGED
FO_CLEANUP_COMPLETE
FO_TEMPORARY_FILE
FO_DELETE_ON_CLOSE
FO_OPENED_CASE_SENSITIVE
FO_HANDLE_CREATED
FO_FILE_FAST_IO_READ
FO_RANDOM_ACCESS
FO_FILE_OPEN_CANCELLED
FO_VOLUME_OPEN
FO_FILE_OBJECT_HAS_EXTENSION
FO_NO_INTERMEDIATE_BUFFERING
FO_GENERATE_AUDIT_ON_CLOSE
FO_DIRECT_DEVICE_OPEN

Process/Thread Types

Job - JOB

DeleteProcedure = PspJobDelete

CloseProcedure = PspJobClose

Process - EPROCESS

DeleteProcedure = PspProcessDelete

Profile - EPROFILE

DeleteProcedure = ExpProfileDelete

Section - SECTION

DeleteProcedure = MiSectionDelete

Thread - ETHREAD

DeleteProcedure = PspThreadDelete

Token - TOKEN

DeleteProcedure = SepTokenDeleteMethod

Job methods - Close

PspJobClose - called by OB when a handle is closed

```
Return unless final close
Mark Job as closed
Acquire the job's lock
If job marked PS_JOB_FLAGS_CLOSE_DONE
    Release the JobLock
    Call PspTerminateAllProcessesInJob()
    Reacquire the JobLock
Acquire the job's MemoryLimitsLock
Remove any completion port from the job
Release the MemoryLimitsLock
Release the JobLock
Dereference the completion port
```

Job methods - Delete

PspJobDelete - called by OB at final dereference

- Holding the Joblock callout to ntuser
- Acquire the PspJobListLock
- If part of a jobset then we are the job pinning the jobset
    - tJob = next job in set and remove current job
- Release the PspJobListLock
- If (tJob) ObDereferenceObjectDeferDelete (tJob)
- If (Job->Token) ObDereferenceObject (Job->Token)
- Free pool allocated for job filters
- Unlink our JobLock from the global list

Synchronization Types

Event	- KEVENT
EventPair	- EEVENT_PAIR
KeyedEvent	- KEYED_EVENT_OBJECT
Mutant	- KMUTANT
	DeleteProcedure = ExpDeleteMutant
Port	- LPCP_PORT_OBJECT
	DeleteProcedure = LpcpDeletePort
	CloseProcedure = LpcpClosePort
Semaphore	- KSEMAPHORE
Timer	- ETIMER
	DeleteProcedure = ExpDeleteTimer

Win32k.sys

Callback - CALLBACK_OBJECT

DeleteProcedure = ExpDeleteCallback

WindowsStation, Desktop

CloseProcedure = ExpWin32CloseProcedure

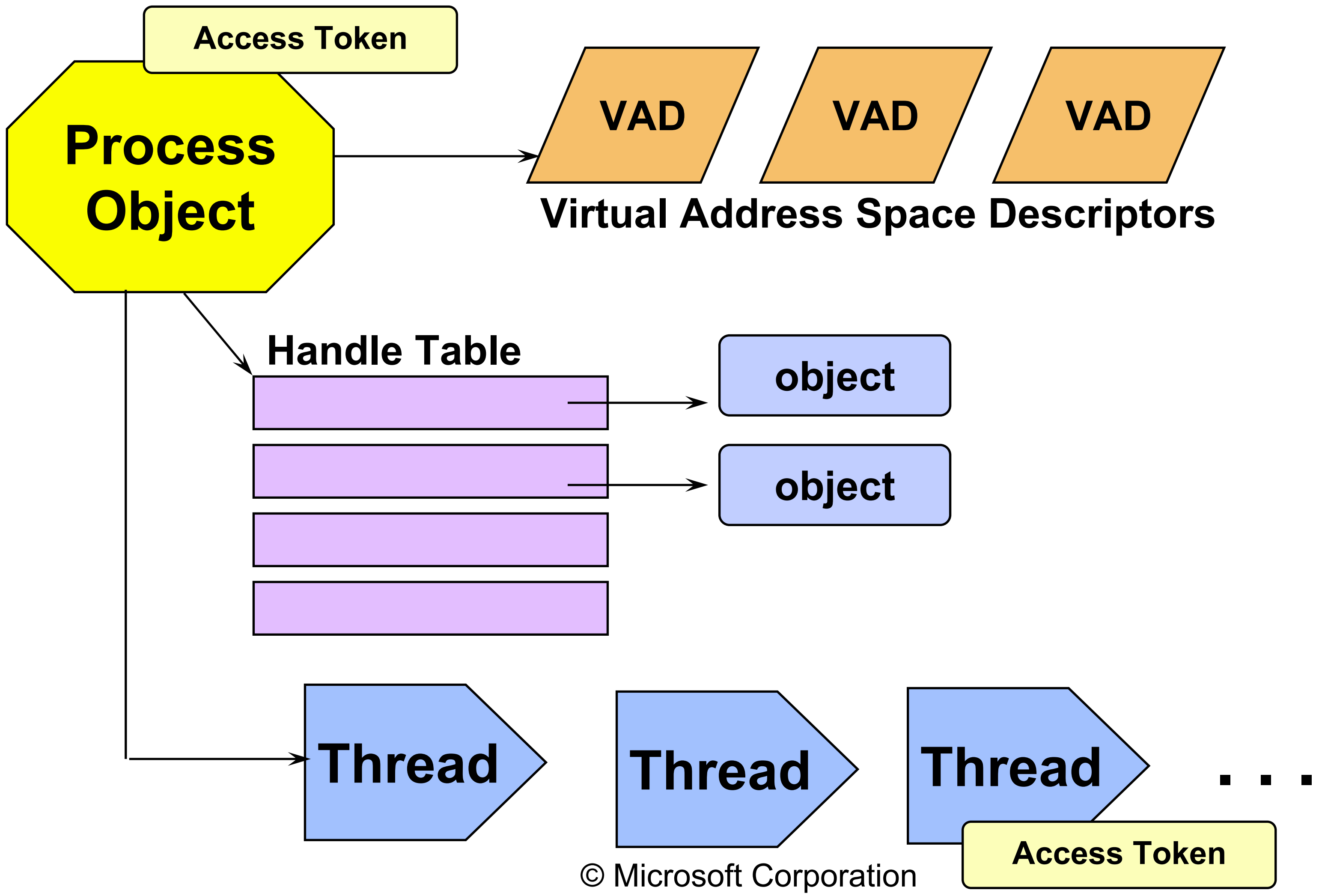
DeleteProcedure = ExpWin32DeleteProcedure

OkayToCloseProcedure = ExpWin32OkayToCloseProcedure

ParseProcedure = ExpWin32ParseProcedure

OpenProcedure = ExpWin32OpenProcedure

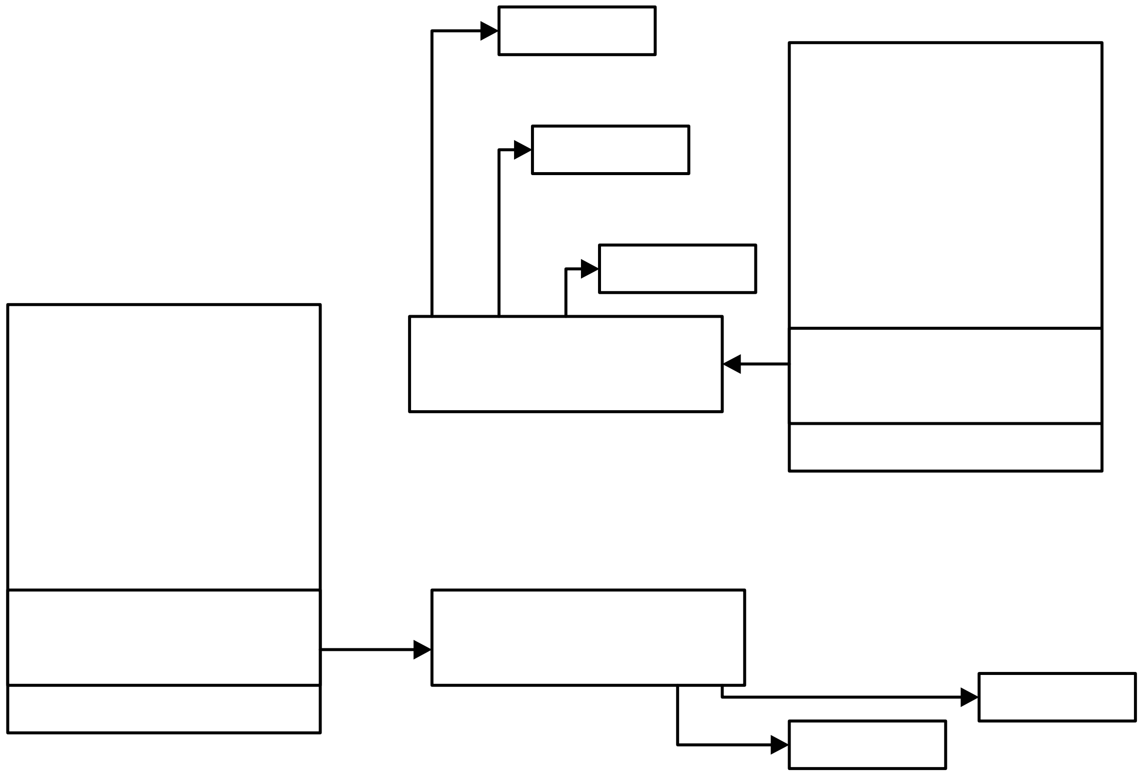
Processes & Threads



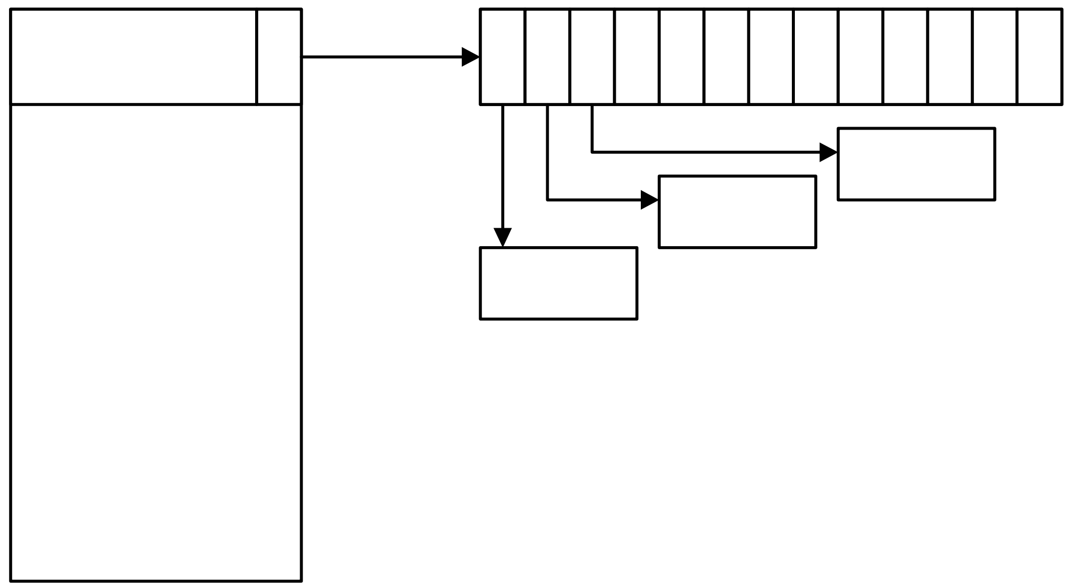
Handle Table (Executive)

- Efficient, scalable object index structure
- One per process containing ‘open’ objects
- Kernel handle table (system process)
- Also used to allocate process/thread IDs

Process Handle Tables



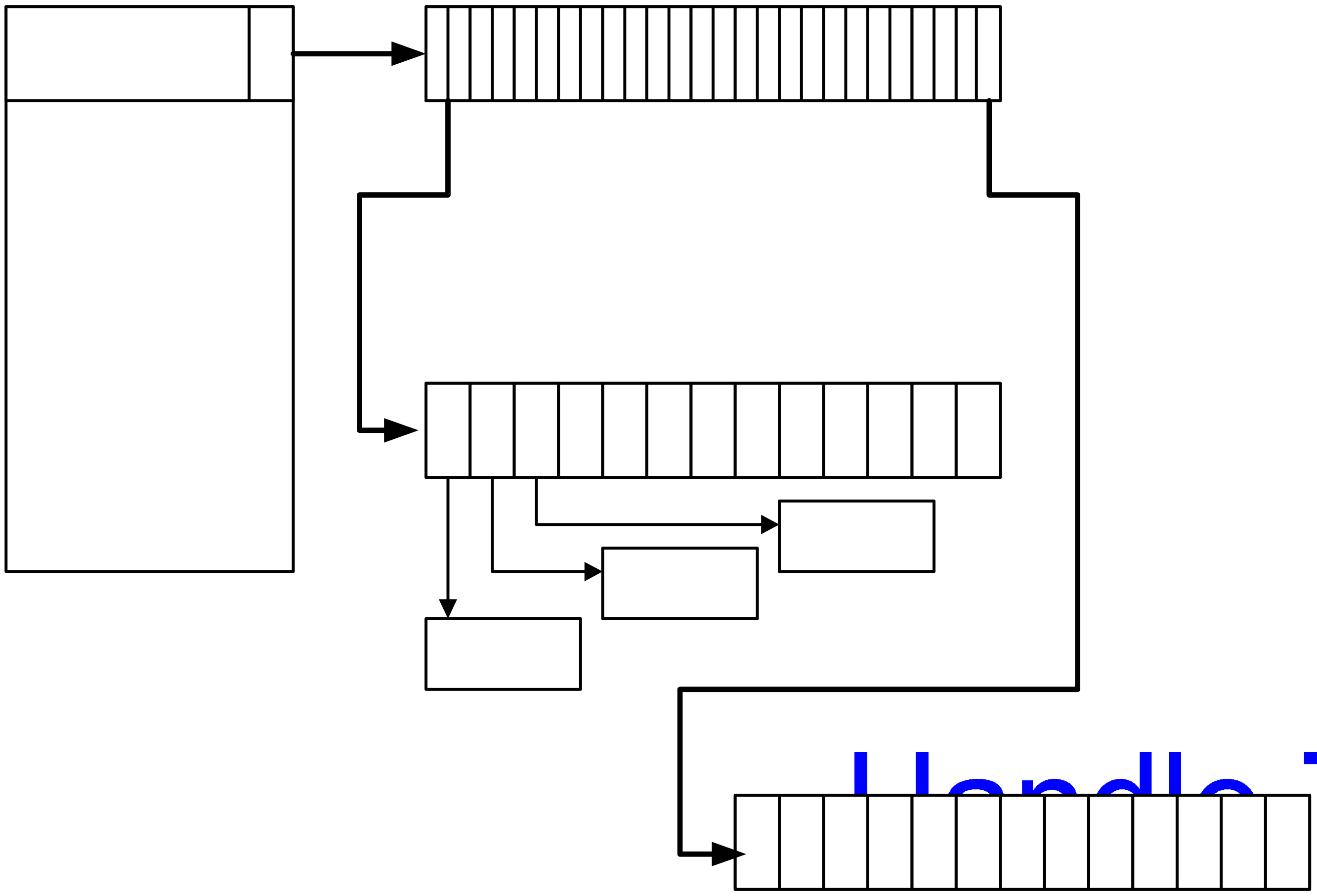
One level: (to 512 handles)



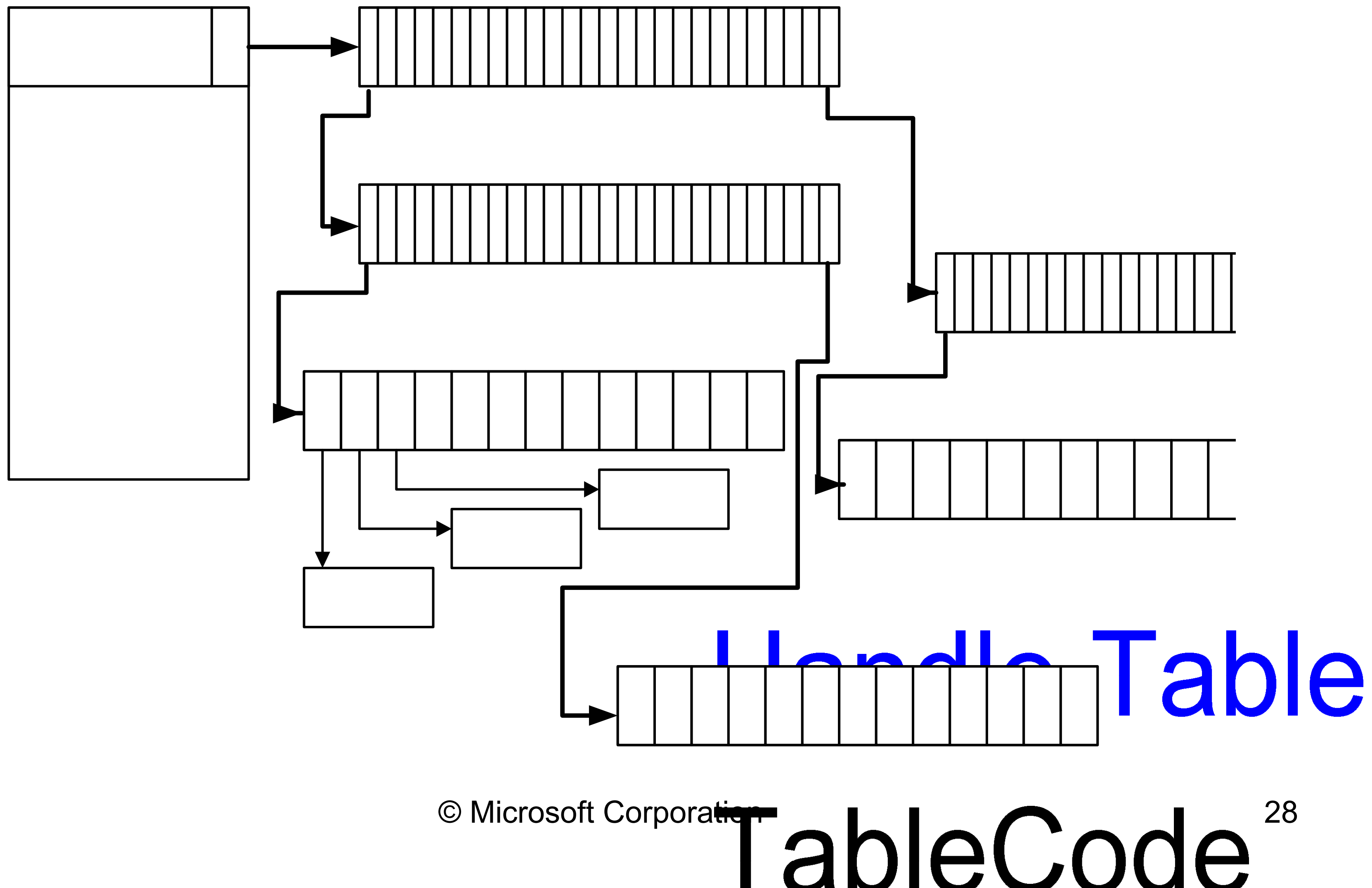
Handle Table

TableCode

Two levels: (to 512K handles)



Three levels: (to 16M handles)



Handle Table Data Structure

TablePointer/Level	Points at handles
QuotaProcess	Who to charge
UniqueProcessId	Passed to callbacks
HandleTableLocks[N]	Locks for handles
HandleTableList	Global list of tables
HandleContentionEvent	Event to block on
DebugInfo	Stacktraces
ExtraInfoPages	Parallel table for audits
FirstFree/LastFree	The two handle free lists
NextHandleNeedingPool	Handles w/ memory
HandleCount	Handles in use

Handle Table Functions

ExCreateHandleTable – create non-process tables
ExDupHandleTable – called creating processes

ExSweepHandleTable – for process rundown
ExDestroyHandleTable – called destroying processes

ExCreateHandle – setup new handle table entry
ExChangeHandle – used to set inherit and/or protect
ExDestroyHandle – implements CloseHandle
ExMapHandleToPointer – reference underlying object

ExReferenceHandleDebugInfo – tracing handles
ExSnapShotHandleTables – handle searchers (oh.exe)

ExCreateHandle(table, entry)

```
NewHandleTableEntry = ExpAllocateHandleTableEntry()
KeEnterCriticalRegionThread()
*NewHandleTableEntry = *HandleTableEntry
ExUnlockHandleTableEntry()
KeLeaveCriticalRegionThread()
```

ExpAllocateHandleTableEntry()

```
while (1) {
    while (! (OldValue = Table->FirstFree)) {
        ExAcquirePushLockExclusive(TableLock[0]);
        If (OldValue = Table->FirstFree)  break;
        If (OldValue = ExpMoveFreeHandles())  break;
        ExpAllocateHandleTableEntrySlow();
        ExReleasePushLockExclusive(TableLock[0]);
    }
    ExpUnlockHandleTableExclusive();
    Handle.Value = (OldValue & FREE_HANDLE_MASK);
    Entry = ExpLookupHandleTableEntry();
    Idx = ((Handle.Value)>>2) % HANDLE_LOCKS;
    ExAcquirePushLockExclusive(TableLock[idx]);
    if (OldValue != *(volatile)&Table->FirstFree) {
        ExReleasePushLockExclusive(TableLock[idx]);
        continue;
    }
    KeMemoryBarrier ();
    NewValue = *(volatile)&Entry->NextFreeTableEntry;
    Expected = InterlockedCompareExchange (&Table-
      >FirstFree, NewValue, OldValue);
        ExReleasePushLockExclusive(Lock[idx]);
    if (Expected == OldValue) break;
}

InterlockedIncrement (HandleCount);
*pHandle = Handle;
```

ExpLookupHandleTableEntry

```
If Handle.Value >= NextHandleNeedingPool
    return NULL;
CapturedTable = *(volatile)&Table->TableCode;
CapturedTable = CapturedTable - TableLevel;
switch (CapturedTable & LEVEL_CODE_MASK) {
   … index into tables according to level …
}

return Entry;
```

ExpMoveFreeHandles

```
// Move all free entries from the delayed free list
Old = InterlockedExchange (&Table->LastFree, 0);
Acquire and immediately release all the TableLocks to synch
if (! StrictFIFO) {
    // If FirstFree list is empty just stash the delayed list
    if (InterlockedCompareExchange (&Table->FirstFree,
                    Old + GetNextSeq(), 0) == 0) return Old;
}
Reverse the chain to get:  FirstEntry -> … -> LastEntry
New = FirstEntry + GetNextSeq();
while (1) {
    tmp = Table->LastFree;
    Entry->NextFreeTableEntry = tmp;
    if (tmp == InterlockedCompareExchange (Index, New, tmp))
        break;
}
return Old;
```

Object Manager Summary

- Manages the NT namespace
- Common scheme for managing resources
- Extensible method-based model for building system objects
- Memory management based on reference counting
- Uniform/centralized security model
- Support handle-based access of system objects
- Common, uniform mechanisms for using system resources

Discussion