Windows Kernel Internals
Overview

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

Contributors

Neill Clift
Adrian Marinescu
Nar Ganapathy
Jake Oshins
Andrew Ritz
Jonathan Schwartz
Mark Lucovsky
Samer Arafeh
Dan Lovinger
Landy Wang
David Solomon
Ben Leis
Brian Andrew
Jason Zions
Gerardo Bermudez
Dragos Sambotin
Arun Kishan
Adrian Oney

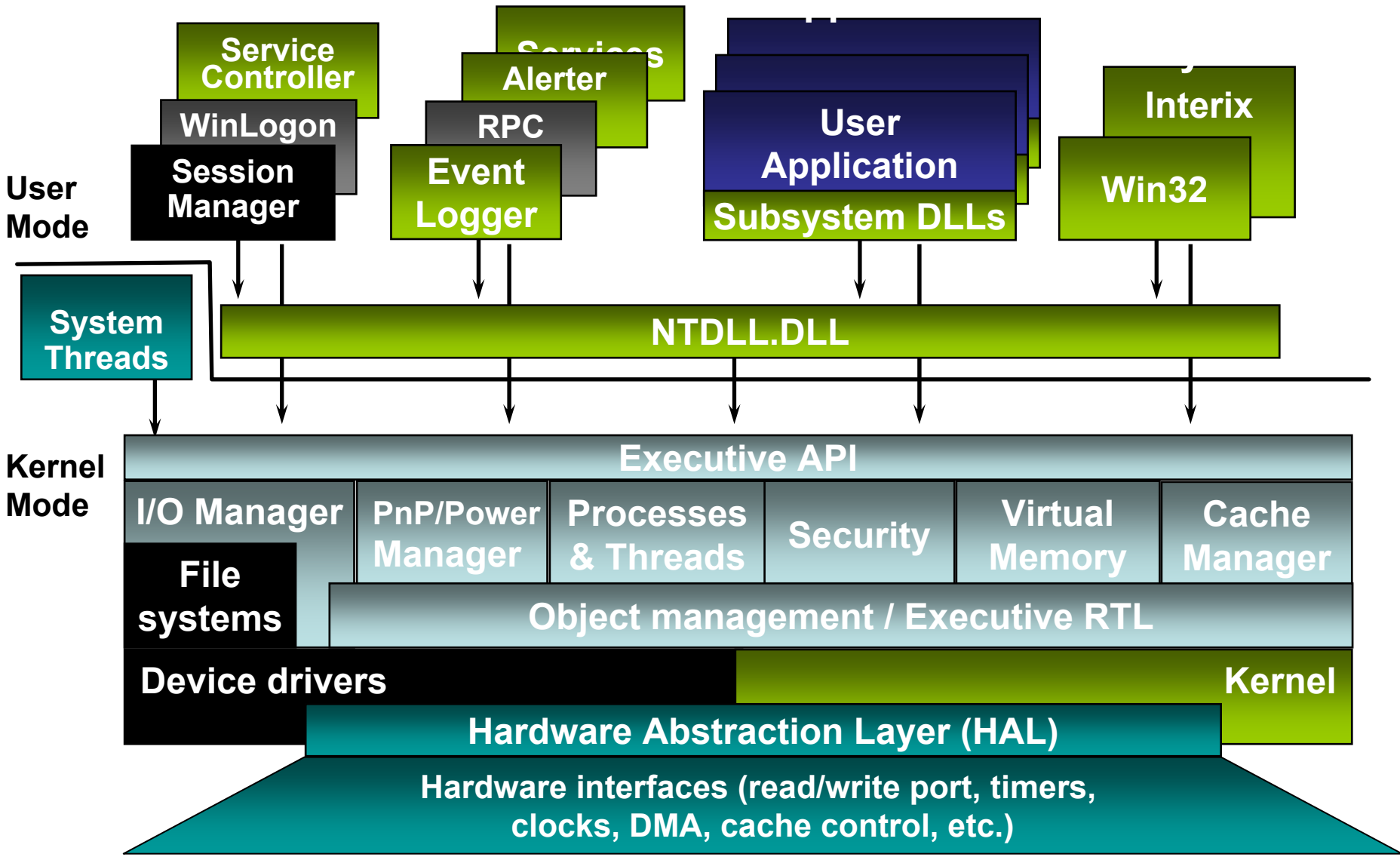
Windows History

- Team formed in November 1988
- Less than 20 people
- Build from the ground up
  - Advanced Operating System
  - Designed for desktops and servers
  - Secure, scalable SMP design
  - All new code
- Rigorous discipline – developers wrote very detailed design docs, reviewed/discussed each others docs and wrote unit tests

Goals of the NT System

- Reliability – Nothing should be able to crash the OS. Anything that crashes the OS is a bug and we won't ship until it is fixed
- Security – Built into the design from day one
- Portability – Support more than one processor, avoid assembler, abstract HW dependencies.
- Extensibility – Ability to extend the OS over time
- Compatibility – Apps must run
- Performance – All of the above are more important than raw speed!

Windows Server 2003 Architecture



Windows Executive

- Upper layers of the operating system
- Provides “generic operating system” functions (“services”)
  - Creating and deleting processes and threads
  - Memory management
  - I/O initiation and completion
  - Interprocess communication
  - Security
- Almost completely portable C code
- Runs in kernel (“privileged”, ring 0) mode
- Many interfaces to executive services not documented

Windows Kernel

- Lower layers of the operating system
  - Implements processor-dependent functions (x86 vs. Alpha vs. etc.)
  - Also implements many processor-independent functions that are closely associated with processor-dependent functions
- Main services
  - Thread waiting, scheduling & context switching
  - Exception and interrupt dispatching
  - Operating system synchronization primitives (different for MP vs. UP)
  - A few of these are exposed to user mode
- Not a classic “microkernel”
  - shares address space with rest of kernel components

HAL - Hardware Abstraction Layer

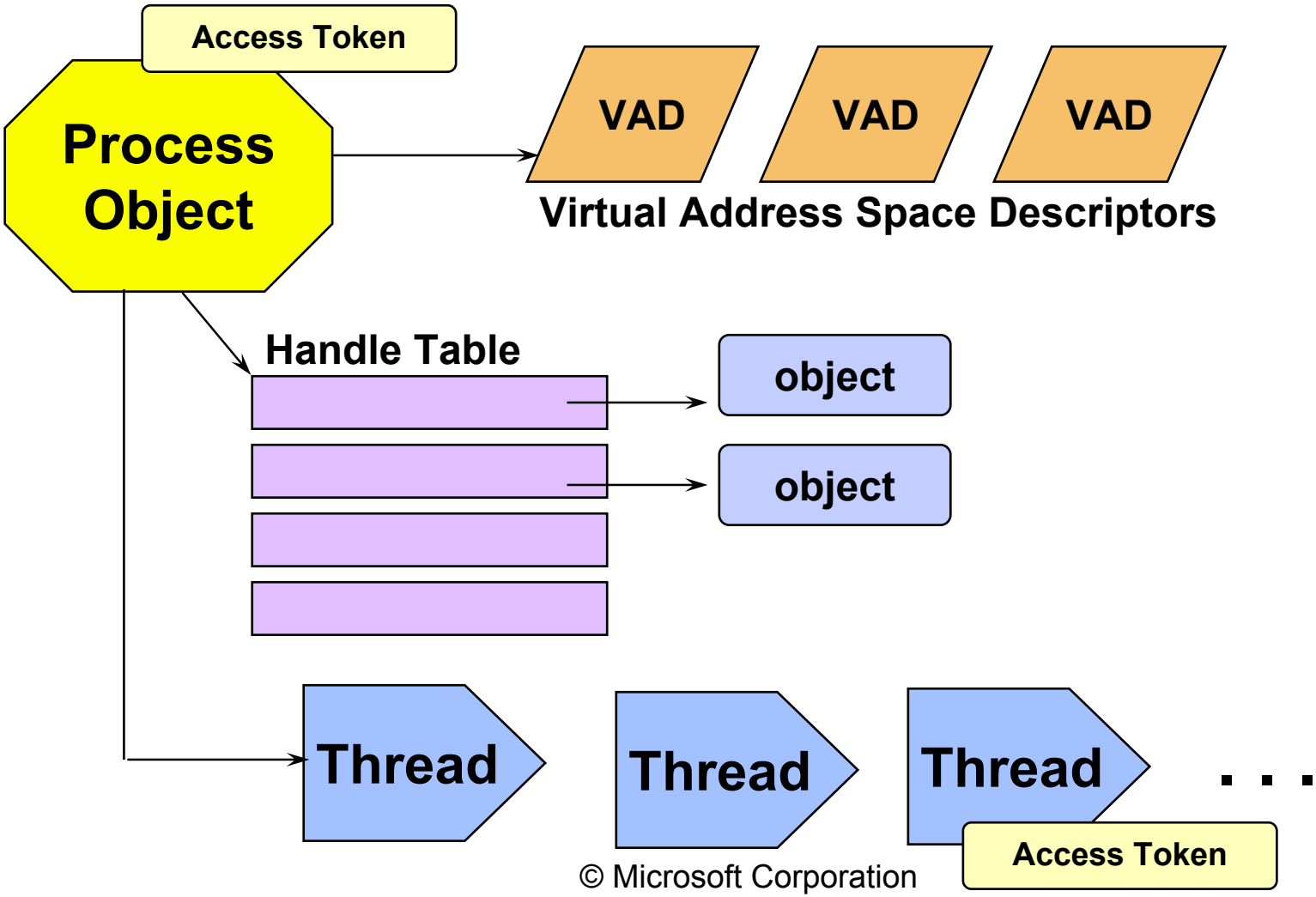
- Subroutine library for the kernel & device drivers
  - Isolates Kernel and Executive from platform-specific details
  - Presents uniform model of I/O hardware interface to drivers
- HAL abstracts:
  - System timers, Cache coherency & flushing
  - SMP support, Hardware interrupt priorities
  - HAL also implements some functions that appear to be in the Executive and Kernel

Kernel Mode Execution

Code is run in kernel mode for one of three reasons:

1. Requests from user mode (system calls)
   - Via the system service dispatch mechanism
   - Kernel-mode code runs in the context of the requesting thread
2. Interrupts from external devices
   - Interrupts (like all traps) are handled in kernel mode
   - NT-supplied interrupt dispatcher invokes the interrupt service routine
   - ISR runs in the context of the interrupted thread (so-called “arbitrary thread context”)
   - ISR often requests the execution of a “**DPC routine**”, which also runs in kernel mode
3. Dedicated kernel-mode threads
   - Some threads in the system stay in kernel mode at all times (mostly in the “System” process)
   - Scheduled, preempted, etc., like any other threads

Processes & Threads



Each process has its own…

- Virtual address space (including program global storage, heap storage, threads' stacks)
  - processes cannot corrupt each other's address space by mistake
- Working set (physical memory "owned" by the process)
- Access token (includes security identifiers)
- Handle table for Win32 kernel objects
- These are common to all threads in the process, but separate and protected between processes

Each thread has its own…

- Stack (automatic storage, call frames, etc.)
- Instance of a top-level function
- Scheduling state (Wait, Ready, Running, etc.) and priority
- Current access mode (user mode or kernel mode)
- Saved CPU state if it isn't Running
- Access token (optional -- overrides process's if present)

Windows Past, Present, Future

PAST: Personal computer, 16->32 bits, MSDOS, Windows 9x code base, desktop focus

- Features, usability, compatibility, platform
- Windows 98

PRESENT: Enterprise computing, 32/64 bits, NT code base, solid desktop, datacenter

- Reliability, performance, IT Features
- Windows XP, Windows Server 2003

FUTURE: Managed code **(.NET Framework)**

- Productivity, innovation, empowerment
- Longhorn

.Net: Making it Simple

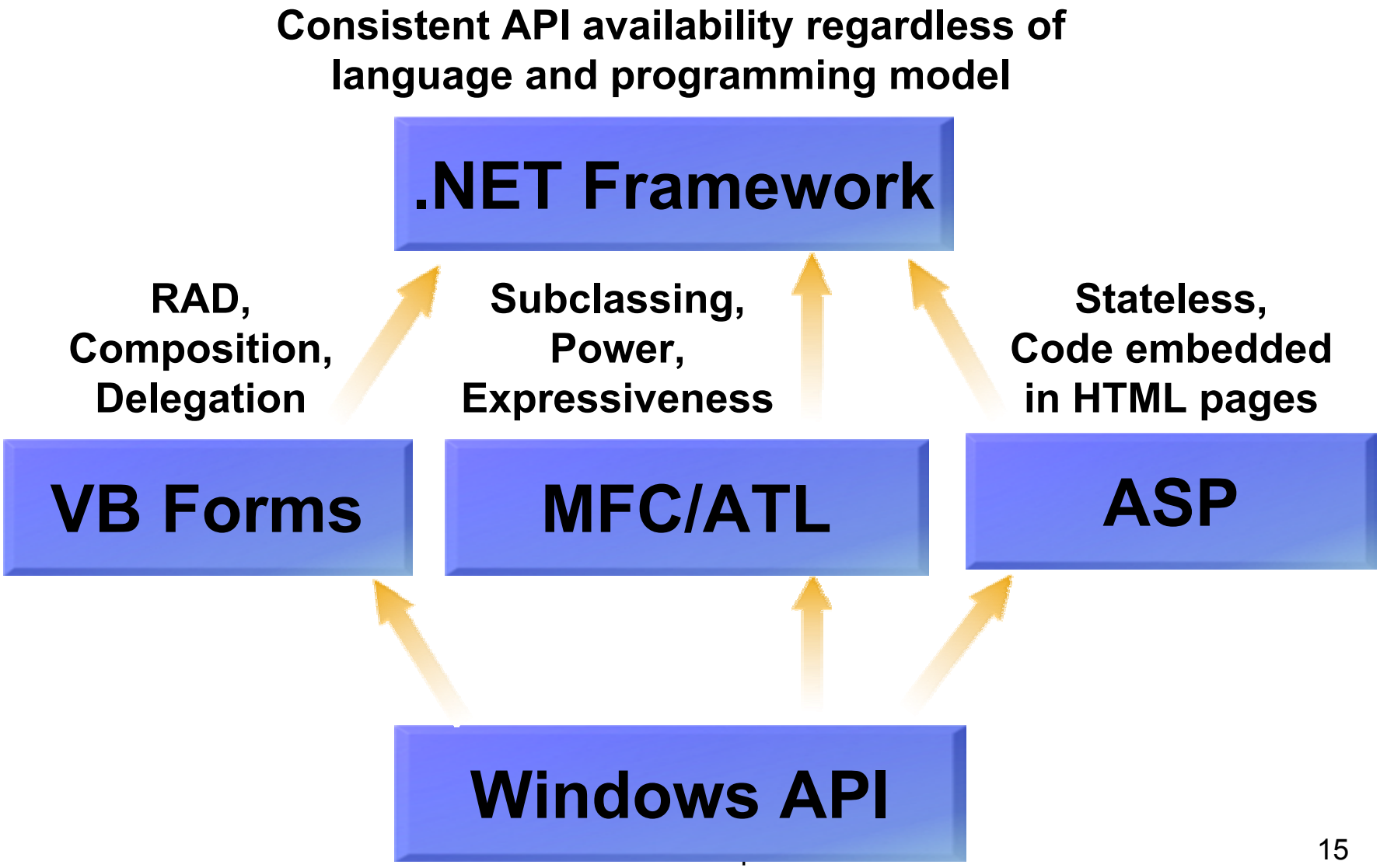
Windows API

```
HWND hwndMain = CreateWindowEx(
    0, "MainWClass", "Main Window",
    WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    (HWND)NULL, (HMENU)NULL, hInstance, NULL);
ShowWindow(hwndMain, SW_SHOWDEFAULT);
UpdateWindow(hwndMain);
```

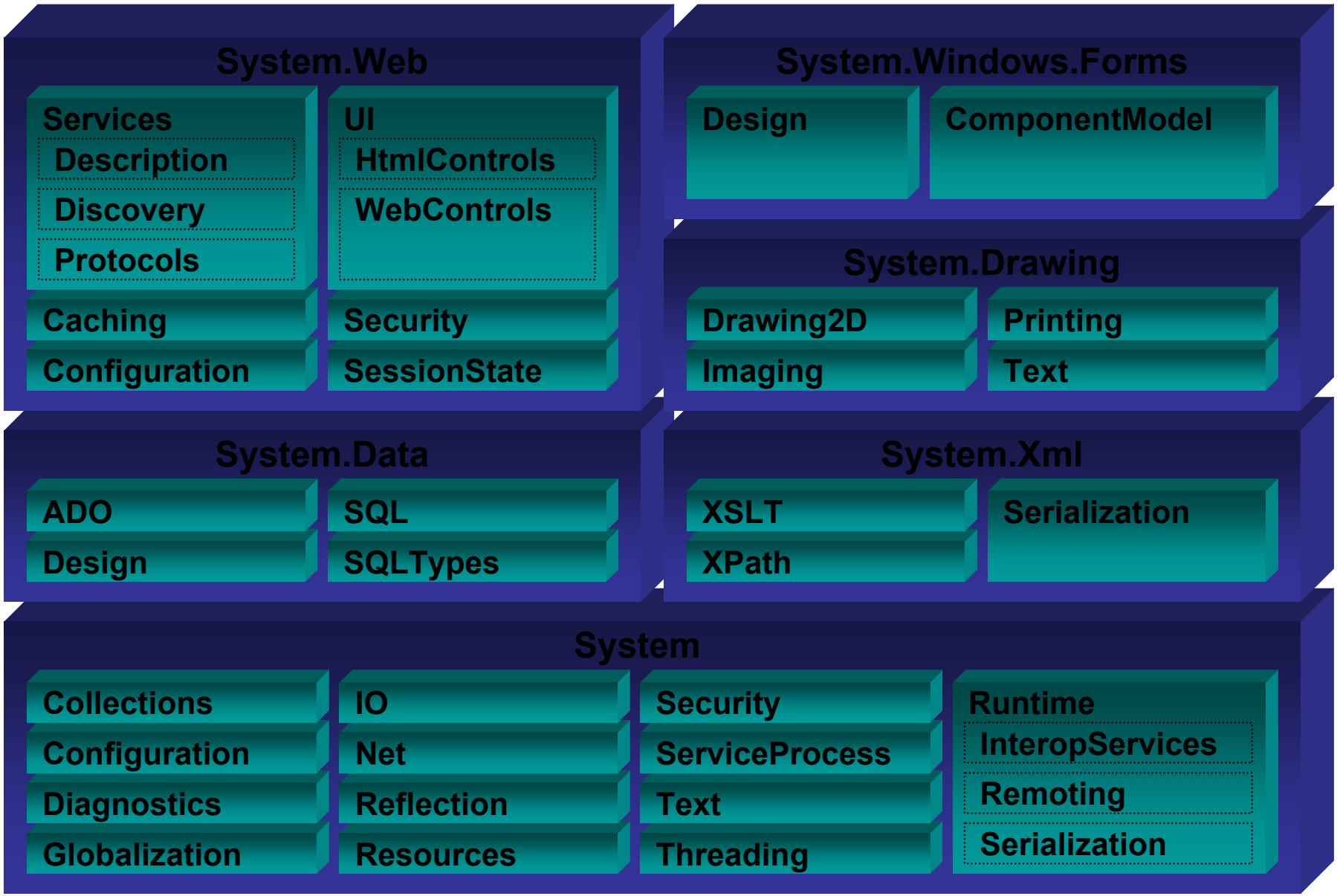
.Net Framework

```
Window w = new Window();
w.Text = "Main Window";
w.Show();
```

.Net: Unify Programming Models



.Net: API Organization



.Net: Languages

- The Managed Platform is Language Neutral
  - All languages are first class players
  - You can leverage your existing skills
- Common Language Specification
  - Set of features guaranteed to be in all languages
  - C# enforcement: [assembly:CLSCompliant(true)]
- We are providing
  - VB, C++, C#, J#, JScript
- Third-parties are building
  - APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk…

Unmanaged vs. Managed

Unmanaged Code	Managed Code
Binary standard	**Type standard**
Type libraries	**Assemblies**
Immutable	**Resilient bind**
Reference counting	**Garbage collection**
Type unsafe	**Type safe**
Interface based	**Object based**
HRESULTs	**Exceptions**
GUIDs	**Strong names**

University of Tokyo
Windows Kernel Internals

Lectures

- Object Manager
- Virtual Memory
- Thread Scheduling
- Synchronization
- I/O Manager
- I/O Security
- Power Management
- NT File System
- Registry
- Lightweight Proc Calls
- Windows Services
- System Bootstrap
- Traps / Ints / Exceptions
- Processes
- Adv. Virtual Memory
- Cache Manager
- User-mode heap
- Win32k.sys
- WoW64
- Common Errors

University of Tokyo
Windows Kernel Internals

Projects

Device Drivers and Registry Hooking

Dragos Sambotin – Polytech. Inst. of Bucharest

Using LPC to build native client/server apps

Adrian Marinescu – University of Bucharest

Threads and Fibers

Arun Kishan – Stanford University

Doing virtual memory experiments from user-mode

Arun Kishan – Stanford University

Discussion