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

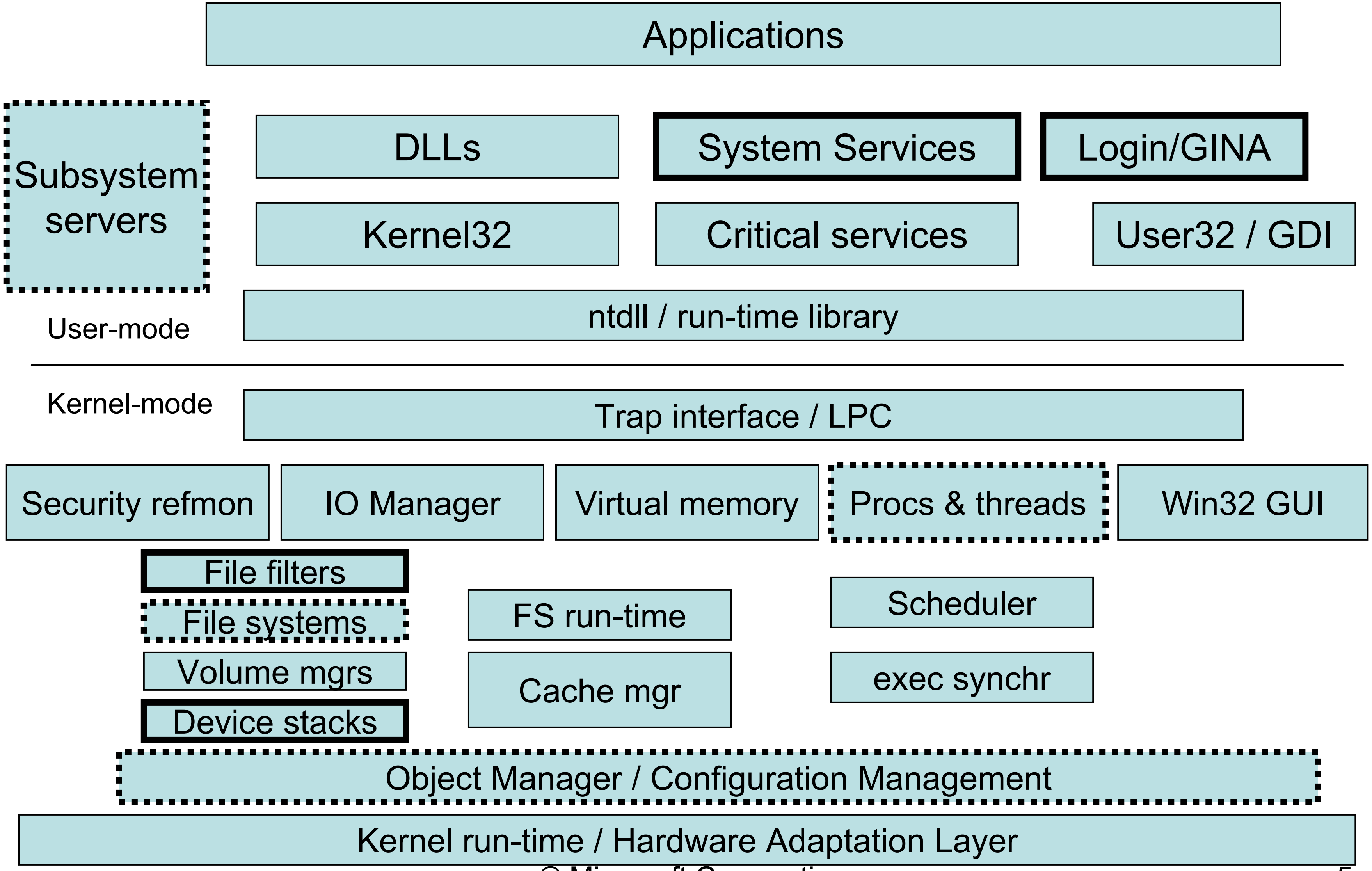
Windows History

- 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 Architecture



Applications

Subsystem servers

DLLs

System Services

Login/GINA

Kernel32

Critical services

User32 / GDI

ntdll / run-time library

User-mode

Kernel-mode

Trap interface / LPC

Security refmon

IO Manager

Virtual memory

Procs & threads

Win32 GUI

File filters

File systems

Volume mgrs

Device stacks

FS run-time

Cache mgr

Scheduler

exec synchr

Object Manager / Configuration Management

Kernel run-time / Hardware Adaptation Layer

Windows Kernel Organization

Kernel-mode organized into

- NTOS (kernel-mode services)
  - Run-time Library, Scheduling, Executive services, object manager, services for I/O, memory, processes, …
- Hal (hardware-adaptation layer)
  - Insulates NTOS & drivers from hardware dependencies
  - Providers facilities, such as device access, timers, interrupt servicing, clocks, spinlocks
- Drivers
  - kernel extensions (primarily for device access)

Major Kernel Services

Process management

- Process/thread creation

Security reference monitor

- Access checks, token management

Memory manager

- Pagefaults, virtual address, physical frame, and pagefile management
- Services for sharing, copy-on-write, mapped files, GC support, large apps

Lightweight Procedure Call (LPC)

- Native transport for RPC and user-mode system services.

I/O manager (& plug-and-play & power)

- Maps user requests into IRP requests, configures/manages I/O devices, implements services for drivers

Cache manager

- Provides file-based caching for buffer file system I/O
- Built over the memory manager

Scheduler (aka ‘kernel’)

- Schedules thread execution on each processor

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)

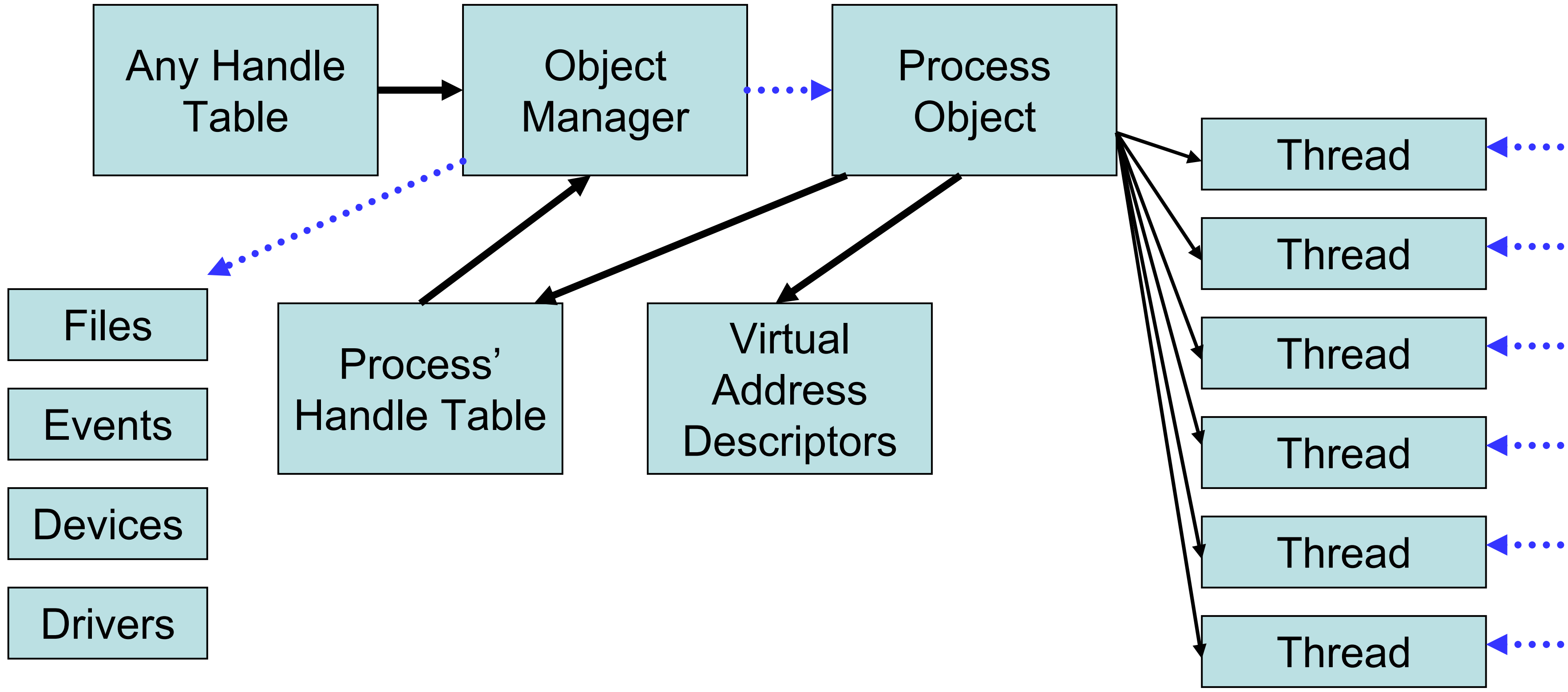
Interrupt Service Routines run at IRL > 2

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)

Process/Thread structure



Any Handle Table

Object Manager

Process Object

Files

Events

Devices

Drivers

Process' Handle Table

Virtual Address Descriptors

Thread

Thread

Thread

Thread

Thread

Thread

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

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

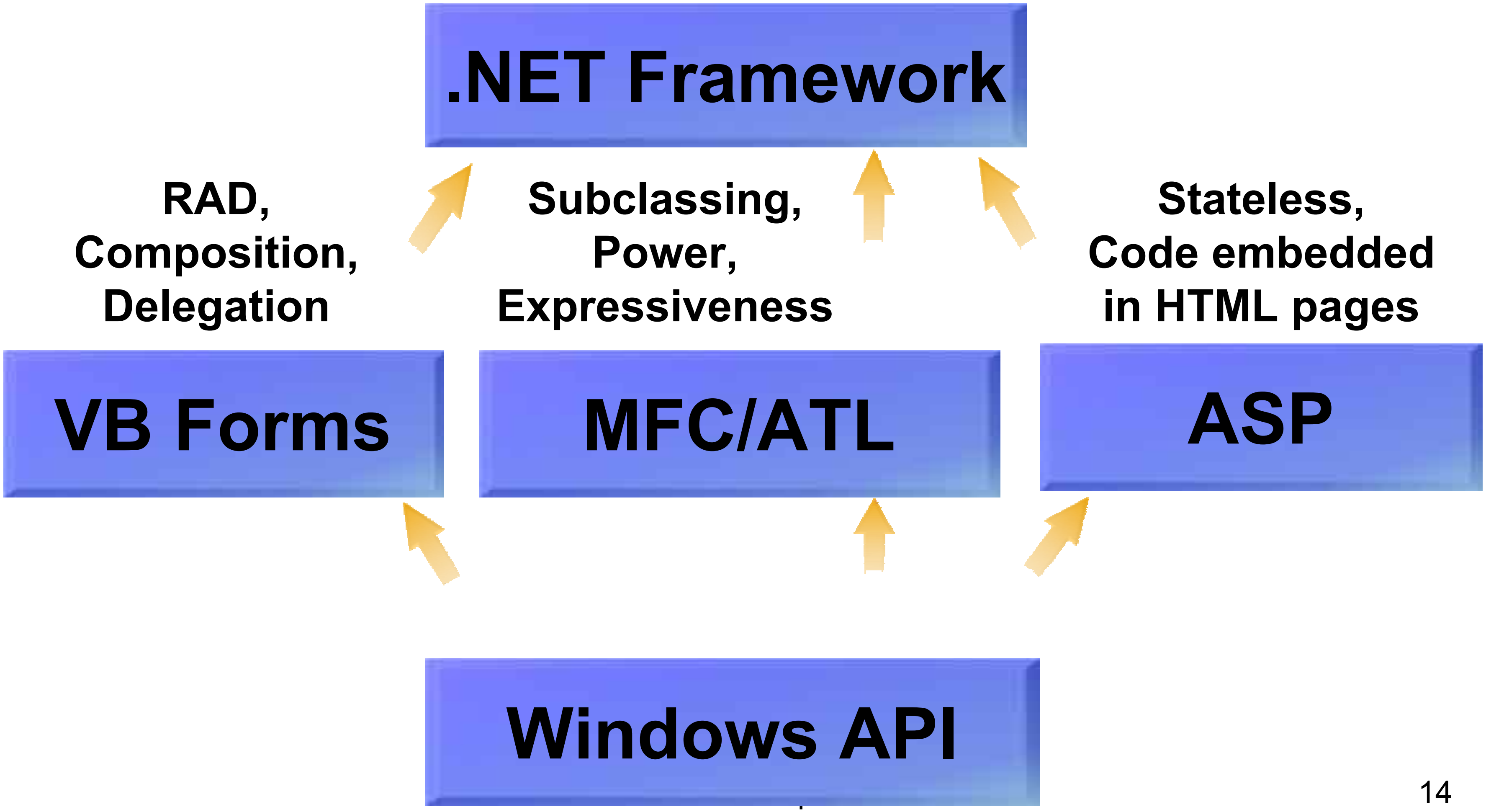
```
HWND hwndMain = CreateWindowEx(
    0, "MainWClass", "Main Window",
    WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,
    CW_USEDEFAULT, CW_USEDEFAULT,
    CW_USEDEFAULT, CW_USEDEFAULT,
    (HWND)NULL, (HMENU)NULL, hInstance, NULL);
ShowWindow(hwndMain, SW_SHOWDEFAULT);
UpdateWindow(hwndMain);
```

.Net Framework

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

.Net: Unify Programming Models

Consistent API availability regardless of language and programming model



.NET Framework

RAD,
Composition,
Delegation

Subclassing,
Power,
Expressiveness

Stateless,
Code embedded
in HTML pages

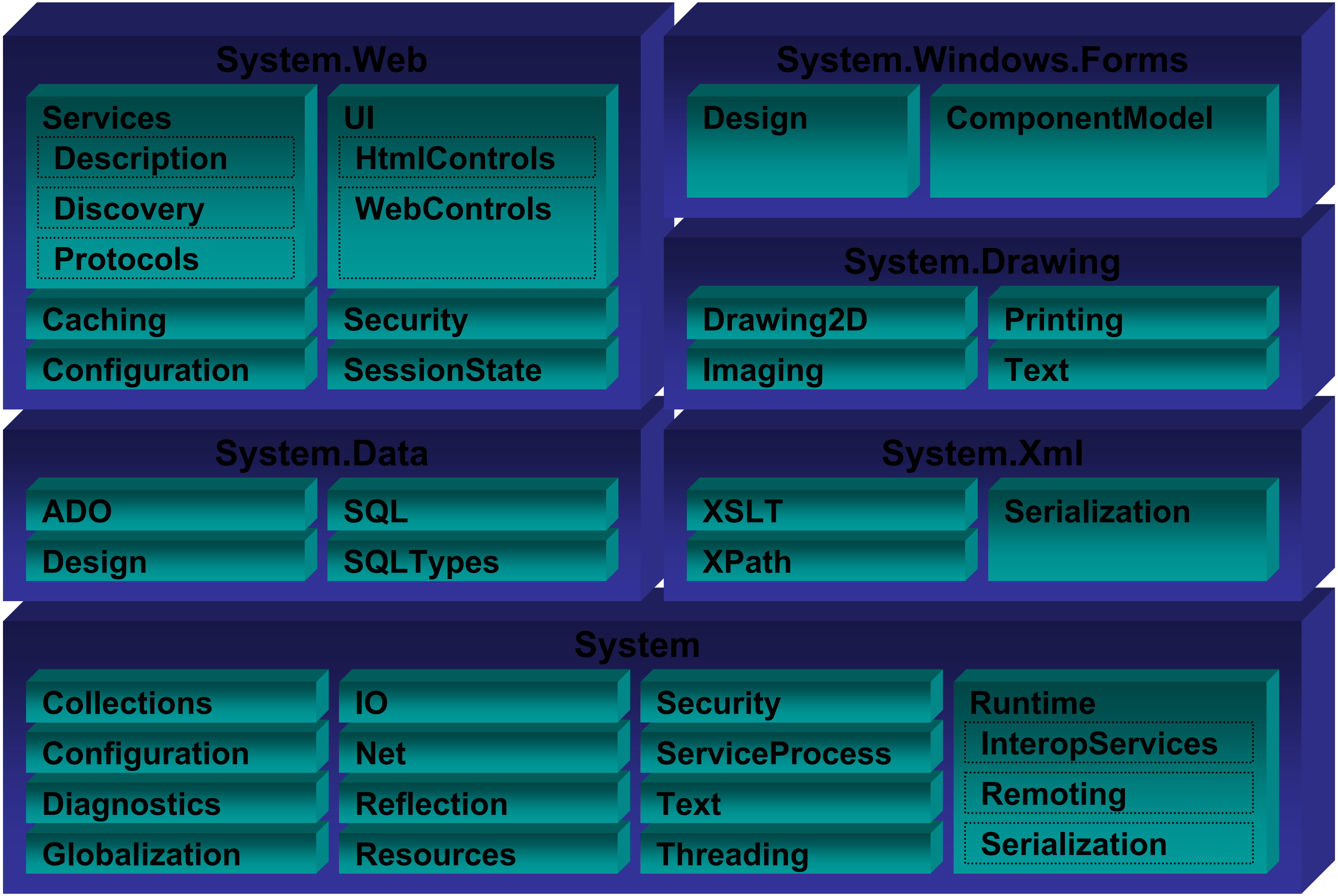
VB Forms

MFC/ATL

ASP

Windows API

.Net: API Organization



System.Web

- Services
  - Description
  - Discovery
  - Protocols
- UI
  - HtmlControls
  - WebControls
- Caching
- Security
- Configuration
- SessionState

System.Windows.Forms

- Design
- ComponentModel

System.Drawing

- Drawing2D
- Printing
- Imaging
- Text

System.Data

- ADO
- SQL
- Design
- SQLTypes

System.Xml

- XSLT
- Serialization
- XPath

System

- Collections
- Configuration
- Diagnostics
- Globalization
- IO
- Net
- Reflection
- Resources
- Security
- ServiceProcess
- Text
- Threading
- Runtime
  - InteropServices
  - Remoting
  - Serialization

.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