Mobile Software Technologies (SW8)

.NET Compact Framework

Hua Lu
Department of Computer Science
Aalborg University

Spring 2008

.NET Compact Framework

- .NET Framework
- .NET Compact Framework
- Basic Supports in .NET Compact Framework
- Smart Device Projects
- .NET Compact Framework 2.0
- Performance Issues

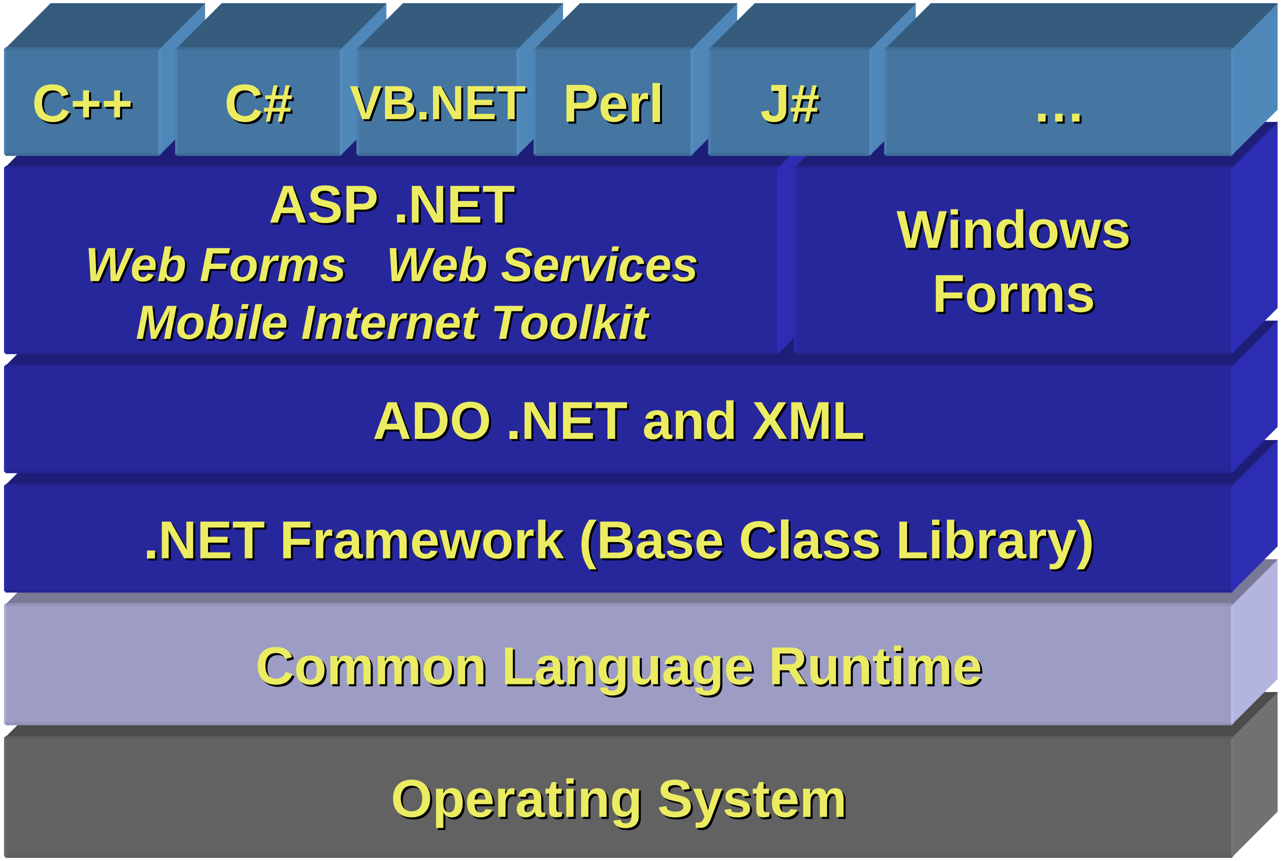
The Core of .NET Framework

- Framework Class Library (FCL)
  - Provides the core functionality:

    ASP.NET, Web Services, ADO.NET, Windows Forms, IO, XML, etc.
- Common Language Runtime (CLR)
  - Garbage collection
  - Language integration
  - Multiple versioning support
  - Integrated security

.NET Framework

- Programming Languages
  - Use your favorite language



C++ C# VB.NET Perl J# …

ASP .NET
Web Forms *Web Services*
Mobile Internet Toolkit

Windows Forms

ADO .NET and XML

.NET Framework (Base Class Library)

Common Language Runtime

Operating System

Common Type System (CTS)

- All .NET languages have the same primitive data types. An *int* in C# is the same as an *int* in VB.NET, COBOL.Net, Haskell, …
- When communicating between modules written in any .NET language, the types are guaranteed to be compatible on the binary level
- Types can be:
  - Value types – passed by value, stored in the stack
  - Reference types – passed by reference, stored in the heap
- Strings are a primitive data type now

Common Language Specification (CLS)

- Any language that conforms to the CLS is a .NET language
- A language that conforms to the CLS has the ability to take full advantage of the Framework Class Library (FCL)
- CLS is standardized by ECMA

.NET Languages

- Languages provided by Microsoft
  - C++, C#, J#, VB.NET, JScript
- Third-parties languages
  - Perl, Python, Pascal, APL, COBOL, Eiffel, Haskell, ML, Oberon, Scheme, Smalltalk…
- Advanced multi-language features
  - Cross-language inheritance and exceptions handling
- Object system is built in, not bolted on
  - No additional rules or API to learn
- All compile to .Net Assemblies
  - Contains MSIL and metadata
    - Intermediate Language

Intermediate Language

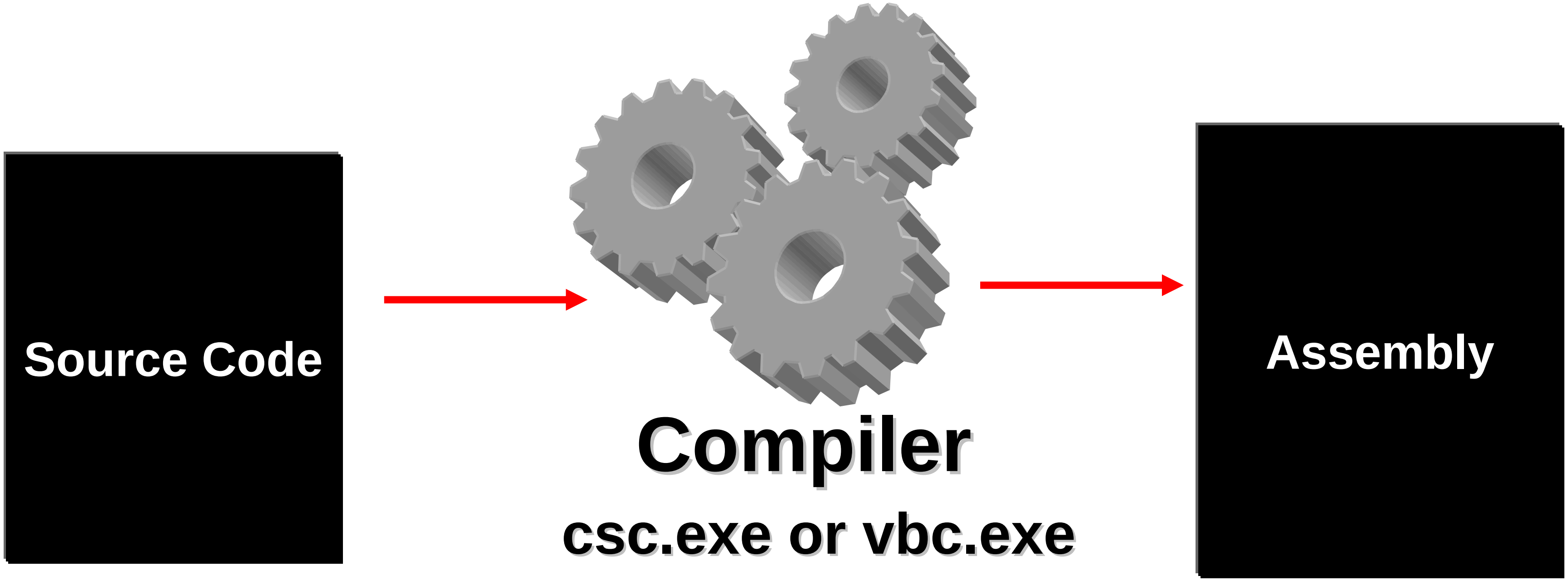
- .NET languages are compiled to an Intermediate Language (IL)
- IL is also known as MSIL or CIL
  - Microsoft IL or Common IL
- CLR compiles IL in just-in-time (JIT) manner – each function is compiled just before execution
- The JIT code stays in memory for subsequent calls
- Recompilations of assemblies are also possible

Example of MSIL Code

```
.method private hidebysig static void  Main()
  cil managed
{
  .entrypoint
  // Code size       11 (0xb)
  .maxstack  8
  IL_0000:  ldstr      "Hello, world!"
  IL_0005:  call       void
  [mscorlib]System.Console::WriteLine(string)
  IL_000a:  ret
} // end of method HelloWorld::Main
```

.Net Assemblies

- Compilation



Source Code

Compiler

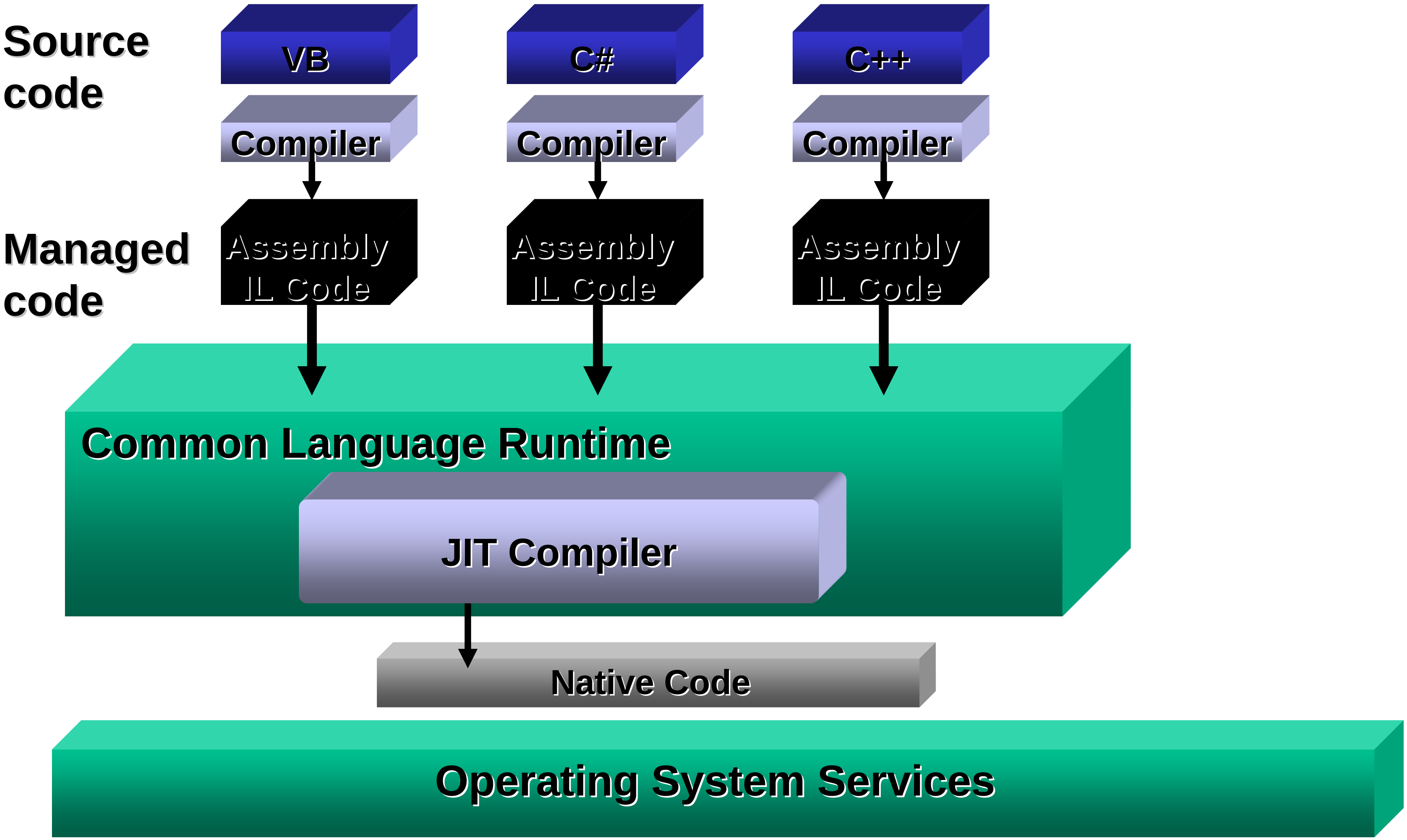
csc.exe or vbc.exe

Assembly

C++, C#, VB or any .NET language

DLL or EXE

.Net Execution Model



.NET Compact Framework

- .NET Framework
- .NET Compact Framework
- Basic Supports in .NET Compact Framework
- Smart Device Projects
- .NET Compact Framework 2.0
- Performance Issues

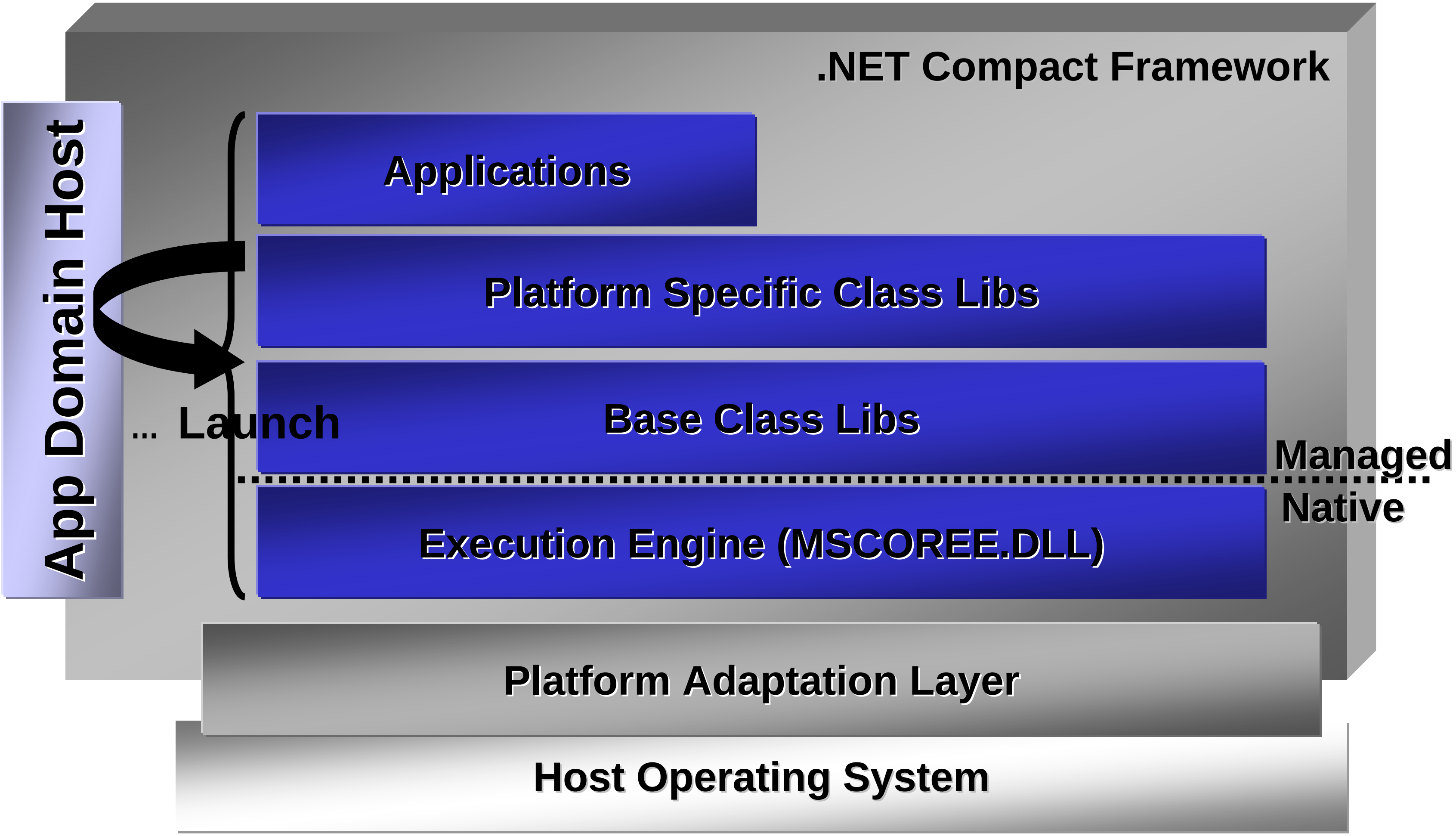
What is the .NET CF?

- Essentially, the .NET CF is an “extended subset” of the .NET Framework
  - “Subset”: Some non-essential classes are not included
  - “Extended”: Functionality specific to the Windows Mobile platform
- High-level goal: Extend the .NET developer platform to the Windows Mobile device platform

Design Goals

- Target mobile and embedded devices
- Portable subset of .NET Framework
  - No new ‘compact’ namespaces
  - Visual Basic .NET and C# compiler support in v1
- Leverage Visual Studio .NET
  - Run managed .EXEs and .DLLs directly
  - Debug with Visual Studio .NET
- Peacefully co-exist with host OS
  - Run on native threads, P/Invoke to call native code
    - Platform Invoke service. This service allows managed code to invoke unmanaged functions residing in DLLs.

Architecture



.NET Compact Framework

App Domain Host

… Launch

Applications

Platform Specific Class Libs

Base Class Libs

Managed

Native

Execution Engine (MSCOREE.DLL)

Platform Adaptation Layer

Host Operating System

Execution Engine Commonalities

- Verifiable type safe execution
  - No uninitialized variables, unsafe casts, bad array indexing, bad pointer math
- Garbage Collection
  - No ref-counting, no leaks
- JIT compilation
- Error handling with exceptions
- Common type system
  - Call, inherit, and source-level debug across different languages

Supported in .NET CF

- Common Base Classes
  - IO, collections, reflection, math, drawing
- Connectivity
  - Networking, HTTP classes, calling XML Web services
- Data Access
  - ADO.NET, SQL Server CE, SQL Server
- XML
  - XmlDocument, XmlReader/Writer
- Windows Forms

Execution Engine Differences

- No ASP.NET
- COM Interop
  - Good support for calling native DLLs
    - • P/Invoke – PlatformInvoke enables calls to Win32 DLLs
  - Support for calling COM objects using dll wrappers
  - No support for writing COM/ActiveX objects
  - No Install-time JIT (nGen)
- No Reflection Emit
- No Remoting
  - Client web services is fully supported
- No Generic Serialization
  - Datasets can be serialized to XML
  - No binary Serialization

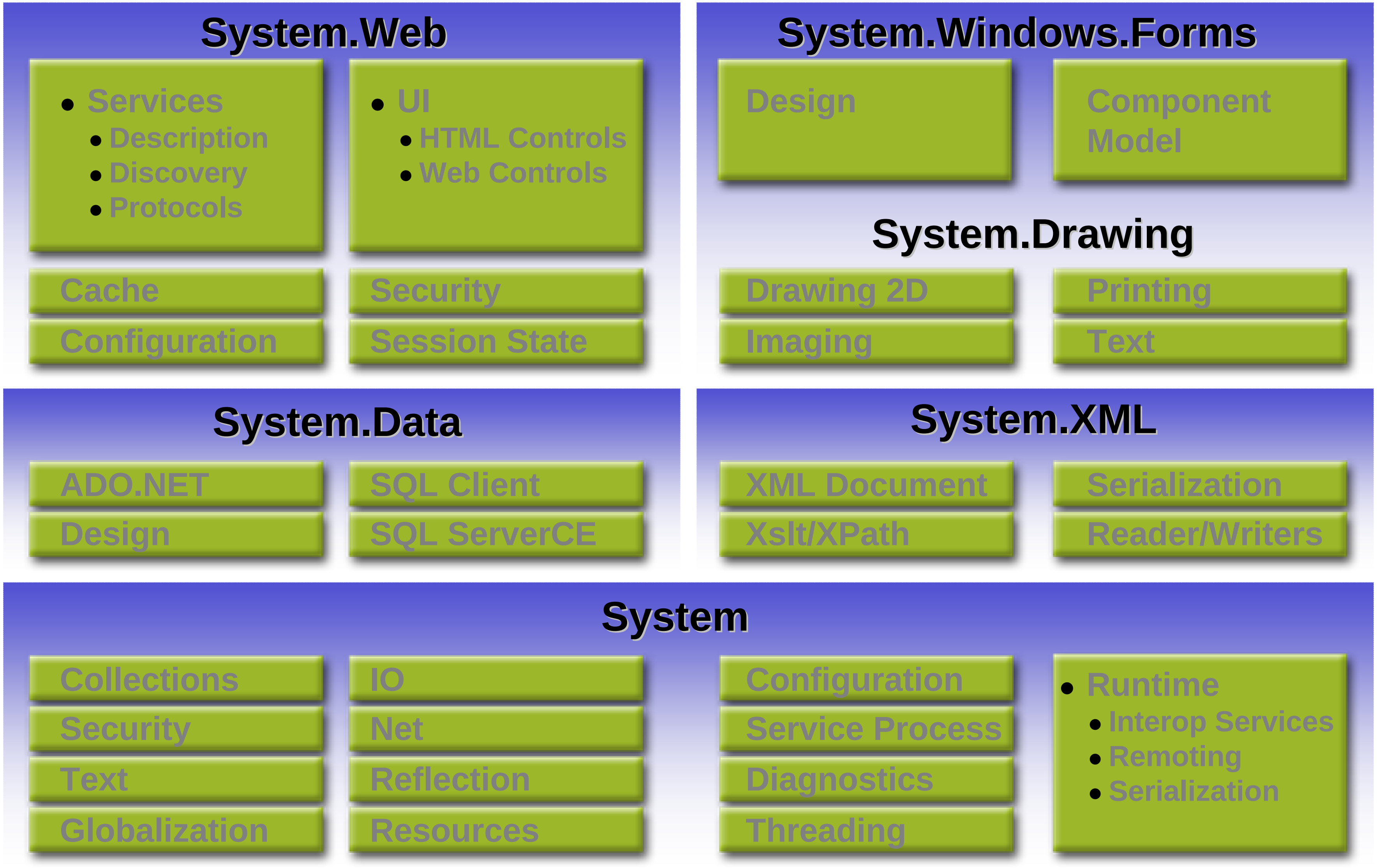
Other Differences

- Class libraries are a subset (about 25%)
- Different size and scalability characteristics
- Compact Additions
  - IrDA support
  - SQL Server CE managed classes
  - Device-specific controls

Framework Size

- Framework size
  - 1.35MB (ROM) on
    Windows CE .NET Device
- Running RAM needs
  - 1 MB+ (depends on app)
- Typical application sizes
  - 5 - 100 KB
  - Apps often smaller due to use of platform features in the framework

.NET Framework



System.Web

- Services
  - Description
  - Discovery
  - Protocols
- UI
  - HTML Controls
  - Web Controls

Cache

Configuration

Security

Session State

System.Windows.Forms

Design

Component Model

System.Drawing

Drawing 2D

Imaging

Printing

Text

System.Data

ADO.NET

Design

SQL Client

SQL ServerCE

System.XML

XML Document

Xslt/XPath

Serialization

Reader/Writers

System

Collections

Security

Text

Globalization

IO

Net

Reflection

Resources

Configuration

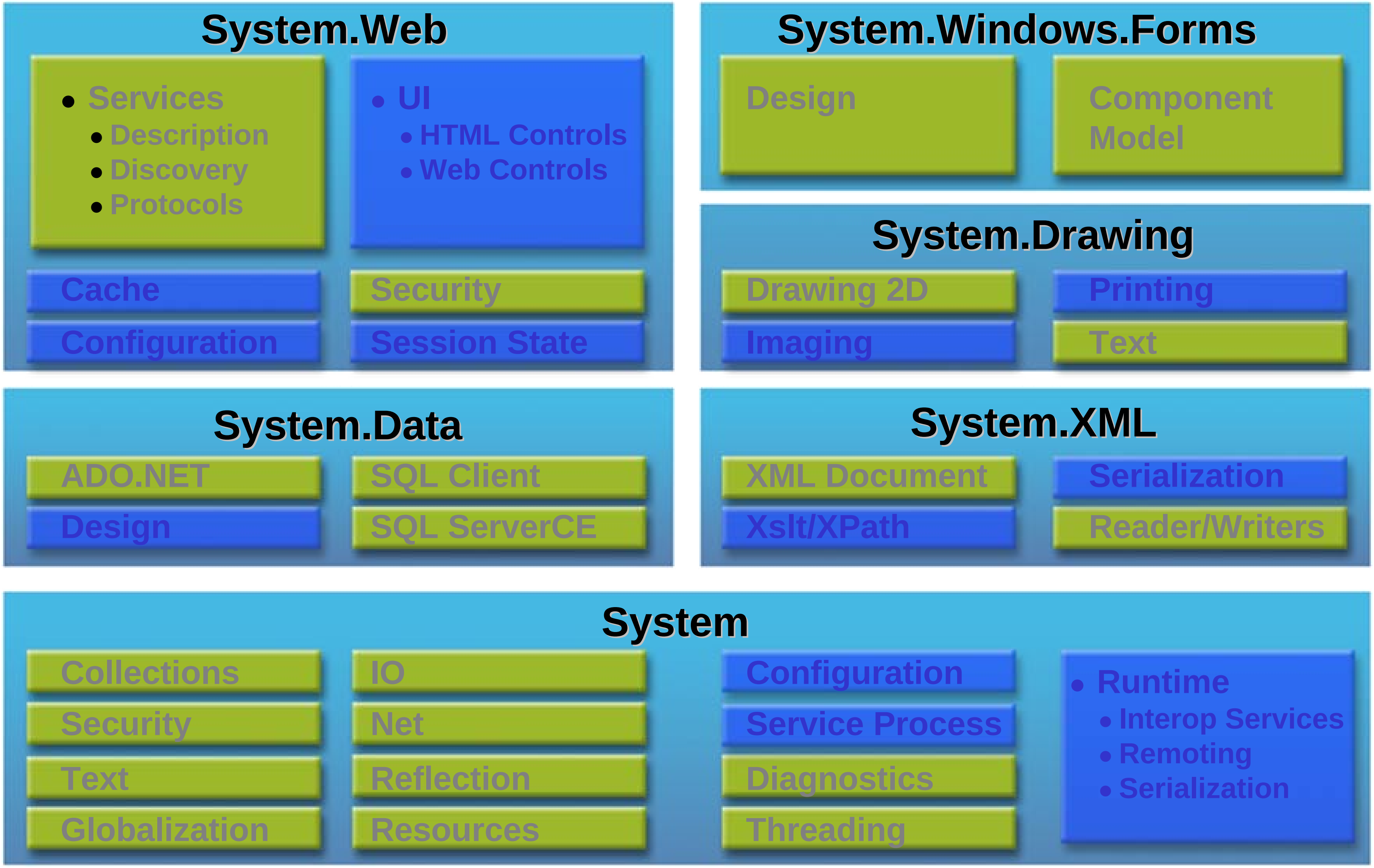
Service Process

Diagnostics

Threading

- Runtime
  - Interop Services
  - Remoting
  - Serialization

.NET Compact Framework



System.Web

- Services
  - Description
  - Discovery
  - Protocols
- UI
  - HTML Controls
  - Web Controls

Cache

Security

Configuration

Session State

System.Windows.Forms

Design

Component Model

System.Drawing

Drawing 2D

Printing

Imaging

Text

System.Data

ADO.NET

SQL Client

Design

SQL ServerCE

System.XML

XML Document

Serialization

Xslt/XPath

Reader/Writers

System

Collections

IO

Configuration

Security

Net

Service Process

Text

Reflection

Diagnostics

Globalization

Resources

Threading

- Runtime
  - Interop Services
  - Remoting
  - Serialization

.NET Compact Framework

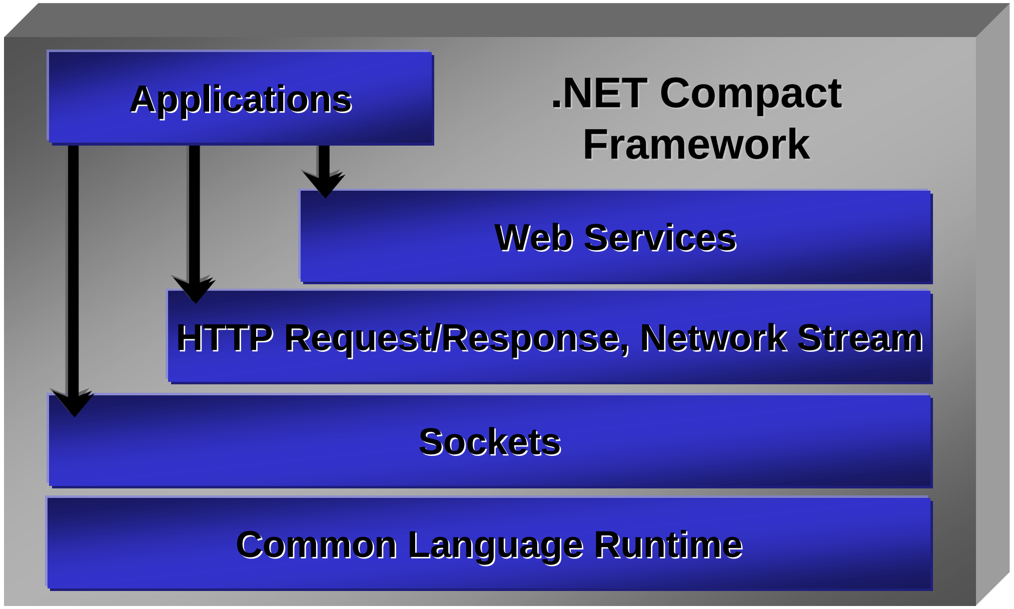
- .NET Framework
- .NET Compact Framework
- Basic Supports in .NET Compact Framework
- Smart Device Projects
- .NET Compact Framework 2.0
- Performance Issues

Basic Data Types

- Base data types are the same as the desktop
  - Formatting
  - StringBuilder
    - More efficient when string length changes
  - Arrays
  - Value types (Int16, Int32, Int64, UInt16, etc.)
  - Floats and doubles
- Collections
  - Classes for storing sets of objects
  - Arraylists and Hashtables

Base: Networking

- Sockets
  - Synchronous and asynchronous
  - Multiple protocols
- Streams
  - Built on top of sockets
  - Synchronous and asynchronous
- HTTP request and response
  - Use stream model
  - Requires no user knowledge of HTTP



Base: Threading

- Applications start with an initial thread
- Applications can start new threads
- Using threads
  - Responsive UI
  - Program function segregation
- Thread synchronization primitives
- App domains exist until all threads exit

Windows Forms Support

- Layout
  - Manual positioning
- Drawing
  - Polygons, lines, arcs, ellipses, rectangles
  - JPEG, BMP images
- Text and images
  - TrueType bitmap fonts on Mobile
- Most desktop controls
- Designer support

Supported Controls

- Supported controls

Button
CheckBox
ComboBox
ContextMenu
DataGrid
DomainUpDown
FileOpenDialog
HScrollBar
ImageList
Label
ListBox
ListView
TreeView
FileSaveDialog
MainMenu
NumericUpDown
Panel
PictureBox
ProgressBar
RadioButton
StatusBar
TabControl
TextBox
Timer
ToolBar
VScrollBar

- Unsupported controls

GroupBox
Printing Controls
RichTextBox
NotificationBubble (PPC)

- Unsupported controls – not available in CE

CheckedListBox
ColorDialog
ErrorProvider
HelpProvider
LinkLabel
NotifyIcon
ToolTip
Splitter
FontDialog

Data Choices

- Remote data
  - XML Web Services, ADO.NET
    (.NET Data Providers), Networking
- On Device data
  - Handle with XML, ADO.NET (DataSet)
  - Cache for use offline with SQL CE, ADO.NET (DataSet persistence as XML)
- Intelligent synchronization of data when connected
  - SQL CE Synchronization, ActiveSync

XML

- XmlTextReader and XmlTextWriter
  - Forward-only parsers of XML data
  - Better performance, no in-memory caching
  - Low memory requirements
- XmlDocument
  - Parse entire document
  - In memory traversal
  - Higher memory requirements; more functionality
- Unsupported:
  - XMLDataDocument, XPath, XSL/T, Validation

ADO.NET Support

- Handling data offline with DataSet
- Communicating DataSet with XML
- Common data model from server to PC to device
- Extensible ADO.NET provider model
- Included data providers
  - SQL Server (System.Data.SqlClient)
  - SQL Server CE (System.Data.SqlServerCe)

XML Web Services Support

- Calling XML Web Services
- All encoding types
- Synchronous and asynchronous invocation
- Basic and Digest authentication
- Secure Sockets Layer support for encryption (SSL)
- Custom SOAP headers
- SOAP Extension Framework

.NET Compact Framework

- .NET Framework
- .NET Compact Framework
- Basic Supports in .NET Compact Framework
- Smart Device Projects
- .NET Compact Framework 2.0
- Performance Issues

What Are Smart Device Projects?

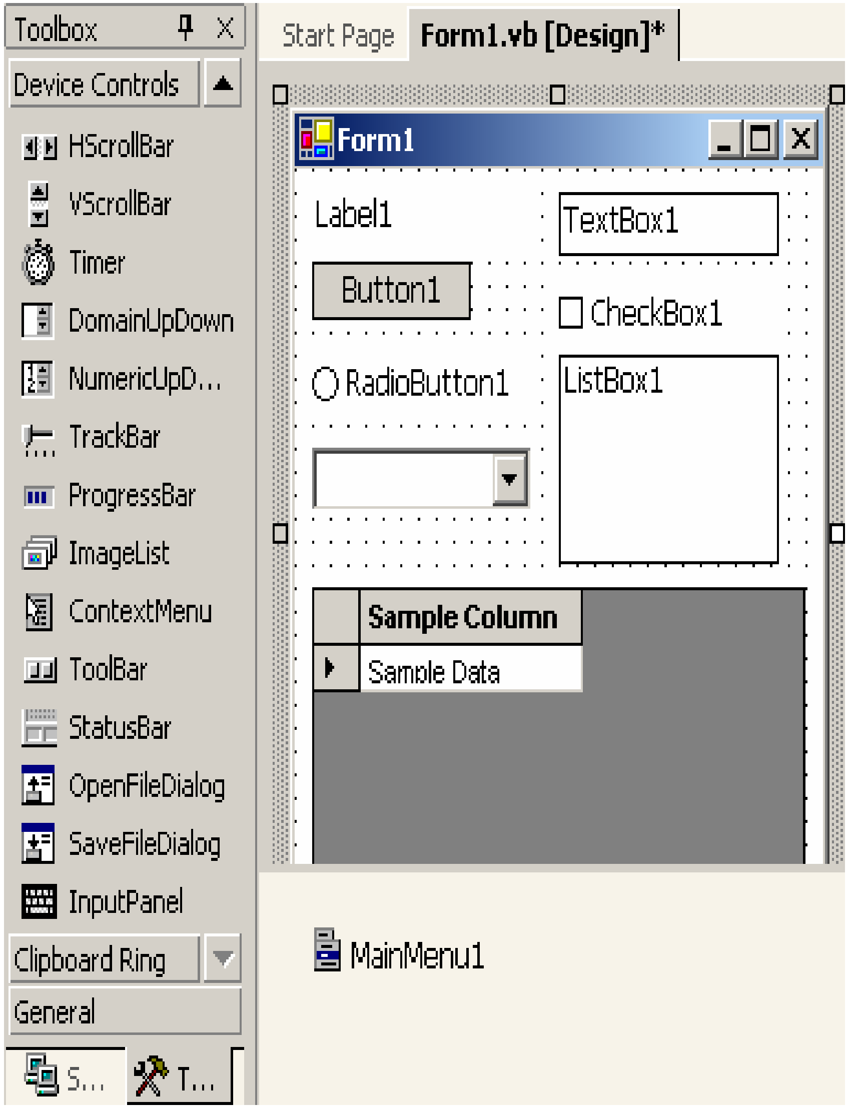
- Smart Device Projects are used to develop applications that target the .NET Compact Framework
- Supported devices include:
  - Pocket PC 2000, 2002 and 2003
  - Pocket PC 2002 Phone Edition
  - SmartPhone 2003
  - Custom-designed embedded devices built with the Windows CE .NET 4.1 operating system
  - Windows Mobile 2003, v5.0, v6.0
- Supported languages are Visual Basic and C#
- Even if you don't have a smart device, you can create and test your smart device applications using emulation technology without leaving the Visual Studio integrated development environment.
- Smart Device Development
  - http://msdn2.microsoft.com/en-us/library/sa69he4t(VS.80).aspx

How to Design a Smart Device Application

- Usability is a key consideration:
  - Avoid requiring intensive data entry
  - Provide access to the Software Input Panel (SIP)
  - Enable device hardware buttons
  - Avoid presenting too many options
  - Use large buttons

How to Create a Smart Device Application

- Create a New Smart Device Application Project
- Choose the platform and type of project
- Add additional forms, controls, and code



How to Test a Smart Device Application

- Visual Studio .NET 2003 includes device emulators that let you test your application
  - Pocket PC and SmartPhone
  - Windows CE .NET 4.1
- You should also test with an actual device
- Debugging
  - Set breakpoints
  - Step through executing code in emulators or on device

How to Deploy a Smart Device Application

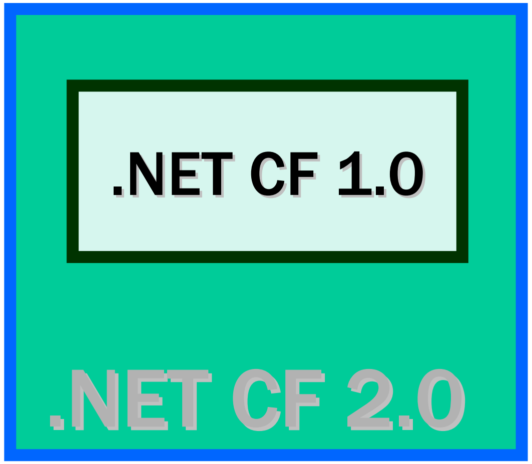
- You can use Microsoft ActiveSync from a desktop computer to manually deploy applications
- You can also use automated distribution mechanisms such as:
  - Downloading CAB files from a Web site
  - Microsoft Systems Management Server (SMS)

.NET Compact Framework

- .NET Framework
- .NET Compact Framework
- Basic Supports in .NET Compact Framework
- Smart Device Projects
- .NET Compact Framework 2.0
- Performance Issues

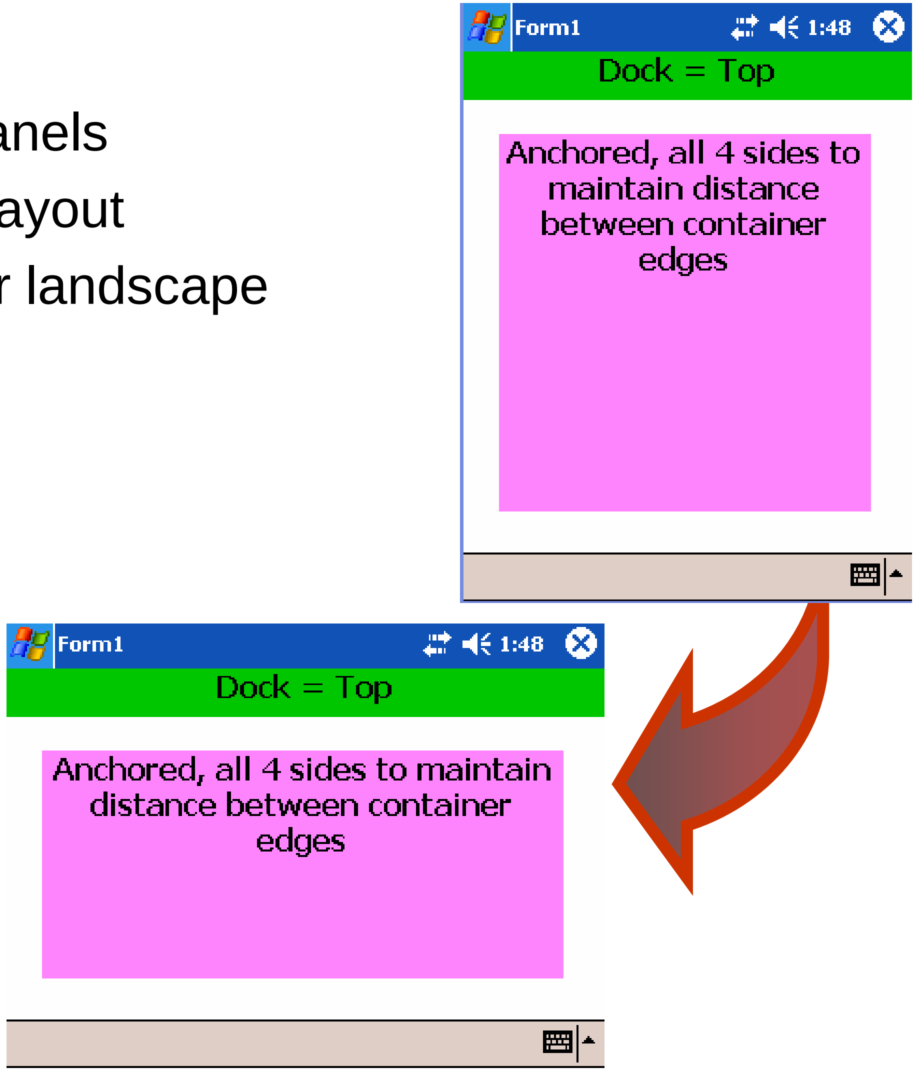
.NET Compact Framework 2.0

- Compatible with full .NET Framework
  - Interoperability, protocols
- Compatible with .NET CF 1.0
  - Side-by-side execution
  - Application compatibility
- Enhanced performance
  - Unified JIT
  - Improved string handling
  - XML improvements
  - Improved ADO.NET with SQL Mobile



.NET CF2.0: Displays and Layout

- Orientation support
  - Docking and anchoring
  - AutoScroll property – forms, panels
  - SuspendLayout and ResumeLayout
  - ChangeOrientation – portrait or landscape
- Resolution support
  - Automatic scaling
  - Graphics
    - DpiX
    - DpiY



Dock = Top

Anchored, all 4 sides to maintain distance between container edges

Dock = Top

Anchored, all 4 sides to maintain distance between container edges

.NET CF2.0: Smartphone Support

- Data access
  - SQL Mobile
  - DataGridView
- Textbox IME switching
  - InputModeEditor
- Enable multiple menu items on left softkey

New with Visual Studio 2005

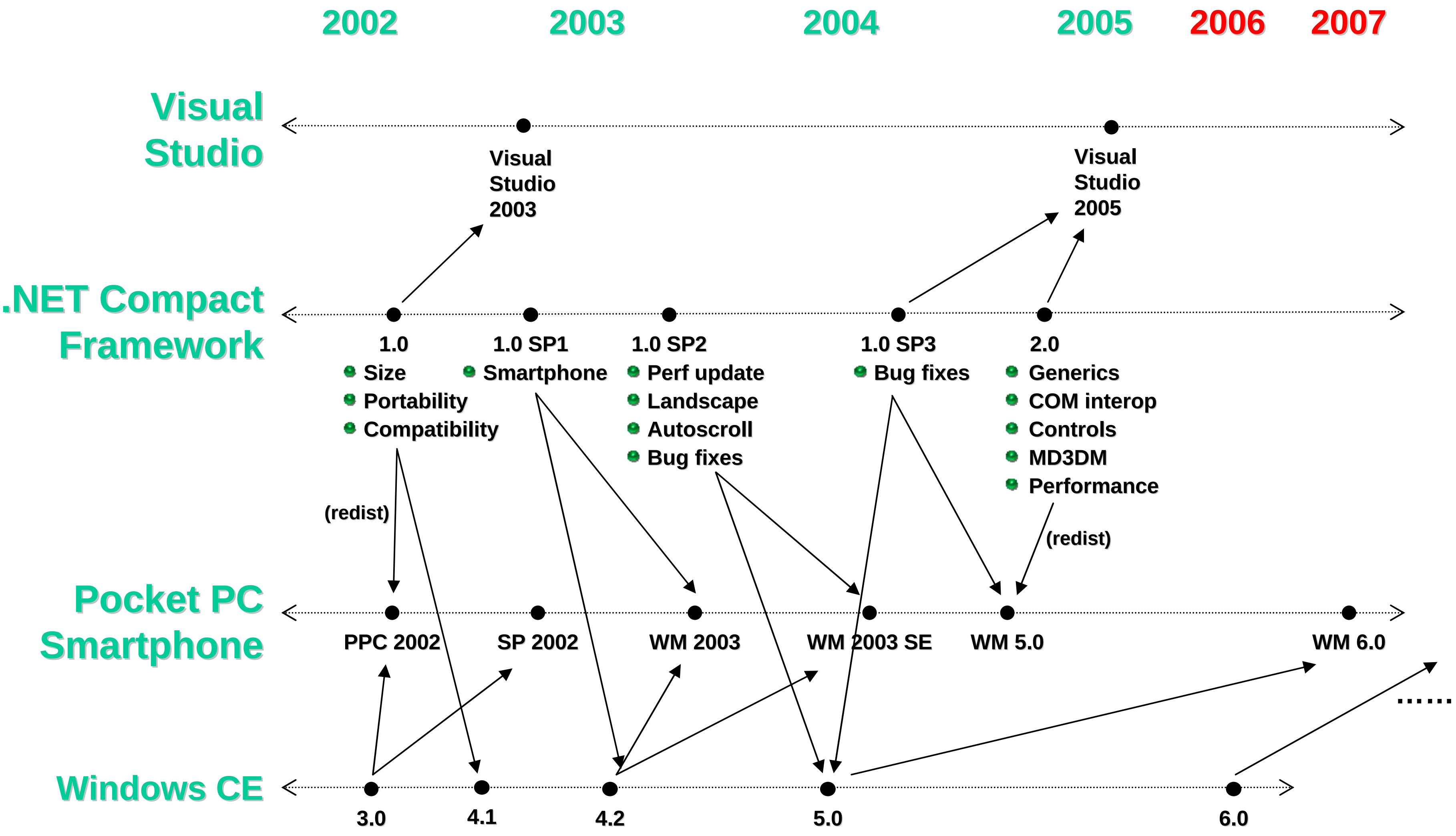
- True ARM emulator with higher fidelity
  - Same executable/CAB for device and emulator
  - Realistic device performance
  - Direct3D and GAPI support
- New debugger
  - Brand new architecture rewritten from line 0
  - Optimized for USB 2.0 performance
- New designers
  - Improved UI designers (docking and anchoring)
  - Data designers (drag, drop, bind SQL to forms)
  - Improved CAB designer support (new project type)

Compatibility

- Applications written using prior versions of the .NET Compact Framework just work on new versions
  - Goal is full backward compatibility
- New versions of .NET Compact Framework run on previous versions of Windows CE and Windows Mobile
  - Windows Mobile support policy: n-2
  - Windows CE support policy: n-1

Supported Devices	
V1	V2
PocketPC 2000 PocletPC 2002 PocketPC 2003, SE SmartPhone 2003 WinCE 4.1 WinCE 4.2 WinCE 5.0	Pocket PC 2003 SE Pocket PC 2005 Smart Phone 2005 WinCE 5.0

Release Roadmap



2002 2003 2004 2005 2006 2007

Visual Studio

Visual Studio 2003

Visual Studio 2005

.NET Compact Framework

1.0
- Size
- Portability
- Compatibility

1.0 SP1
- Smartphone

1.0 SP2
- Perf update
- Landscape
- Autoscroll
- Bug fixes

1.0 SP3
- Bug fixes

2.0
- Generics
- COM interop
- Controls
- MD3DM
- Performance

(redist)

(redist)

Pocket PC Smartphone

PPC 2002 SP 2002 WM 2003 WM 2003 SE WM 5.0 WM 6.0

……

Windows CE

3.0 4.1 4.2 5.0 6.0

Generics

- Classes and methods that work similarly on values of different types
  - Variables are specific types, not objects
  - No casting required
- Benefits
  - Re-use common code
  - Find bugs at compile time

```
class Stack<T>
{
   private T[] store;
   private int size;

   public Stack() {
      store = new T[10];
      size = 0;
   }

   public void Push(T x) {
      // push code goes here
   }

   public T Pop() {
      return store[--size];
   }
}

void Add(Stack<int> s) {
   int x = s.Pop();
   int y = s.Pop();
   s.Push(x+y);
}
```

More New C# Language Features

- Anonymous Methods – Code blocks encapsulated in a delegate

Before

```
// in constructor
     button.Click += new
     EventHandler(ProcessClick);
// separate method
void ProcessClick(object sender,EventArgs e)
{
    // increment a counter or other action
}
```

After

```
// in constructor
button.Click += new EventHandler(sender,args)
{
// increment a counter or other action
};
```

- Partial Types – Split definitions for types and class members across multiple files
  - C# compiler combines all definitions to make a single class

Foo-Part1.cs

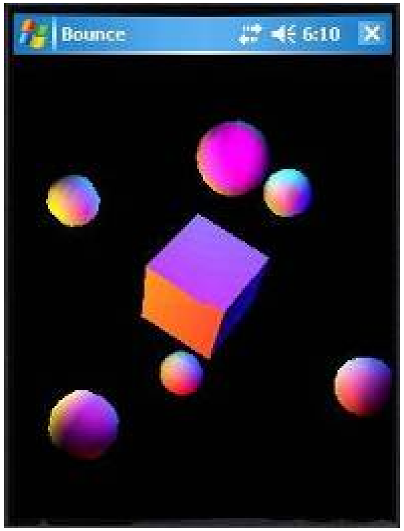
```
public partial class Foo
{
   public void CodeGenFunc()
   {
       // emitted by tool
   }
}
```

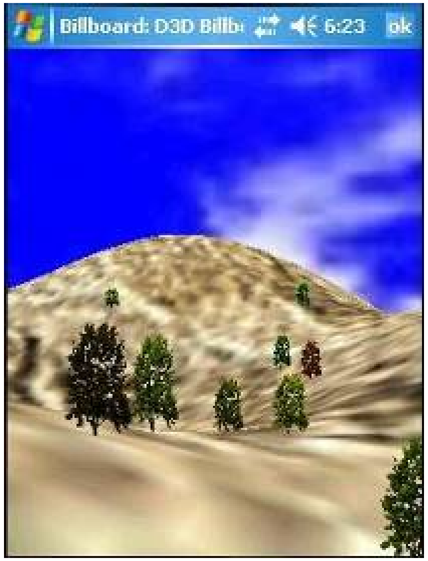
Foo-Part2.cs

```
public partial class Foo
{
   public void UserFunc()
   {
       // user code
   }
}
```

Managed Direct 3D Mobile

- Included in WinCE 5.0 and Windows Mobile 5.0
  - Native API is DX8 inspired
  - Managed APIs are DX9 inspired
- Features
  - Complete access to the underlying native D3DM API
  - Fixed point support (Vertex Data, Matrices, Lights, and Materials)
  - Sprite
  - Font
  - Mesh
  - TextureLoader
  - Tutorials and Samples





Security

- Managed apps have identical security experience as native apps
  - Mobile Operators can restrict app install/start to signed apps only
  - Operators can control cert chain of trust, and/or rely on Mobile2Market
  - Malicious applications can be revoked if device is restricted
- Security features added:

	V1	V1 SP1	V2
Permissions		Integration with WM load-time infrastructure for run/no-run decision	
Cryptography	• Certificates ▪ ASN1/X.509		• Hashing ▪ MD5, SHA1 • Symmetric key encryption ▪ RC2, RC4, 3DES, DES • Asymmetric key encryption ▪ RSA, DSA
Network Protocols	• Authentication ▪ Digest • HTTPS (Server auth only)		• Authentication ▪ Negotiate ▪ NTLM ▪ Kerberos

.NET Compact Framework

- .NET Framework
- .NET Compact Framework
- Basic Supports in .NET Compact Framework
- Smart Device Projects
- .NET Compact Framework 2.0
- Performance Issues

Performance: Garbage Collector

- What triggers a GC?
  - Memory allocation failure
  - 1M of GC objects allocated (v2)
  - Application going to background
  - GC.Collect() (Avoid “helping” the GC!)
- What happens at GC time?
  - Freezes all threads at safe point
  - Finds all live objects and marks them
    - An object is live if it is reachable from root location
  - Unmarked objects are freed and added to finalizer queue
    - Finalizers are run on a separate thread
  - GC pools are compacted if required (less than 750K of free space)
  - Return free memory to the operating system
- In general, if you don’t allocate objects, GC won’t occur
  - Beware of side-effects of calls that may allocate objects
- http://blogs.msdn.com/stevenpr/archive/2004/07/26/197254.aspx

Where Garbage Comes From? (1)

- Unnecessary string copies
  - Strings are immutable
  - String manipulations (Concat(), etc.) cause copies
  - Use StringBuilder

```
String result = "";
for (int i=0; i<10000; i++) {
   result +=
   ".NET Compact Framework";
    result += " Rocks!";
}
```

```
StringBuilder result =
   new StringBuilder();
for (int i=0; i<10000; i++){
   result.Append(".NET Compact
        Framework");
   result.Append(" Rocks!");
}
```

A Note on StringBuilder

- Remember it's all about reducing memory traffic
- If you roughly know the expected length of your final string – allocate that much before hand (StringBuilder constructor)
- Getting the string out of a StringBuilder doesn't cause a new alloc, the existing buffer is converted into a string
- http://weblogs.asp.net/ricom/archive/2003/12/02/40778.aspx

Where Garbage Comes From? (2)

- Unnecessary boxing
  - Value types allocated on the stack (fast to allocate)
  - Boxing causes a heap allocation and a copy
  - Use strongly typed arrays and collections

    (framework collections are NOT strongly typed)

```
class Hashtable {
   struct bucket {
       Object key;
       Object val;
   }
   bucket[] buckets;
   public Object this[Object key] { get; set; }
 }
```

CLR: Generics

- Fully specialized implementation in .NET Compact Framework v2
  - Pros
    - Strongly typed
    - No unnecessary boxing and type casts
    - Specialized code is more efficient than shared
  - Cons
    - Internal execution engine data structures and JIT-compiled code aren’t shared

      List<int>, List<string>, List<MyType>
  - http://blogs.msdn.com/romanbat/archive/2005/01/06/348114.aspx

CLR: Execution Engine

- Call path
  - Managed calls are more expensive than native
    - Instance call: ~2-3X the cost of a native function call
    - Virtual call: ~1.4X the cost of a managed instance call
    - Platform invoke: ~5X the cost of managed instance call (*Marshal int parameter)
  - Properties are calls
- JIT compilers
  - All platforms have the same optimizing JIT compiler architecture in v2
  - Optimizations
    - Method inlining for simple methods
    - Variable enregistration

CLR: Call Path Sample (1)

```
public class Shape
{
  protected int m_volume;
  public virtual int Volume
  {
   get {return m_volume;}
  }
}
public class Cube:Shape
{
   public MyType(int vol)
   {
     m_volume = vol;
   }
}
```

```
public class Shape
{
  protected int m_volume;
  public int Volume
  {
   get {return m_volume;}
  }
}
public class Cube:Shape
{
   public MyType(int vol)
   {
     m_volume = vol;
   }
}
```

CLR: Call Path Sample (2)

```
public class MyCollection
   {
         private const int m_capacity = 10000;
         private Shape[] storage = new Shape[m_capacity];
         …
         public void Sort()
         {
            Shape tmp;
            for (int i=0; i<m_capacity-1; i++) {
             for (int j=0; j<m_capacity-1-i; j++)
                 if (storage[j+1].Volume < storage[j].Volume){
                            tmp = storage[j];
                            storage[j] = storage[j+1];
                            storage[j+1] = tmp;
                 }
            }
         }
   }
```

```
callvirt instance int32 Shape::get_Volume()
```

CLR: Call Path Sample (3)

57 sec

```
public class Shape
{
  protected int m_volume;
  public virtual int Volume
  {
   get {return m_volume;}
  }
}
public class Cube:Shape
{
   public MyType(int vol)
   {
     m_volume = vol;
   }
}
```

39 sec

```
public class Shape
{
  protected int m_volume;
  public int Volume
  {
   get {return m_volume;}
  }
}
public class Cube:Shape
}
```

- No virtual call overhead
- Inlined (no call overhead at all)

~ Equal to accessing field

CLR: Reflection

- Reflection can be expensive
- Reflection performance cost
  - Type comparisons (for example: typeof())
  - Member enumerations (for example: Type.GetFields())
  - Member access (for example: Type.InvokeMember())
  - ~10-100x slower
- Working set cost
  - Runtime data structures
    - ~100 bytes per loaded type, ~80 bytes per loaded method
- Be aware of APIs that use reflection as a side effect
- Override
  - Object.ToString()
  - GetHashCode() and Equals() (for value types)

Best Practices for Windows Forms

- Load and cache Forms in the background
  - Populate data separate from Form.Show()
    - Pre-populate data, or
    - Load data async to Form.Show()
- Use BeginUpdate/EndUpdate when it is available
  - e.g. ListView, TreeView
- Use SuspendLayout/ResumeLayout when repositioning controls
- Keep event handling code tight
  - Process bigger operations asynchronously
  - *Blocking* in event handlers will affect UI responsiveness
- Form load performance
  - Reduce the number of method calls during initialization

Best Practices for Graphics And Games

- Compose to off-screen buffers to minimize direct to screen blitting
  - Approximately 50% faster
- Avoid transparent blitting in areas that require performance
  - Approximate 1/3 speed of normal blitting
- Consider using pre-rendered images versus using System.Drawing rendering primitives
  - Need to measure on a case-by-case basis