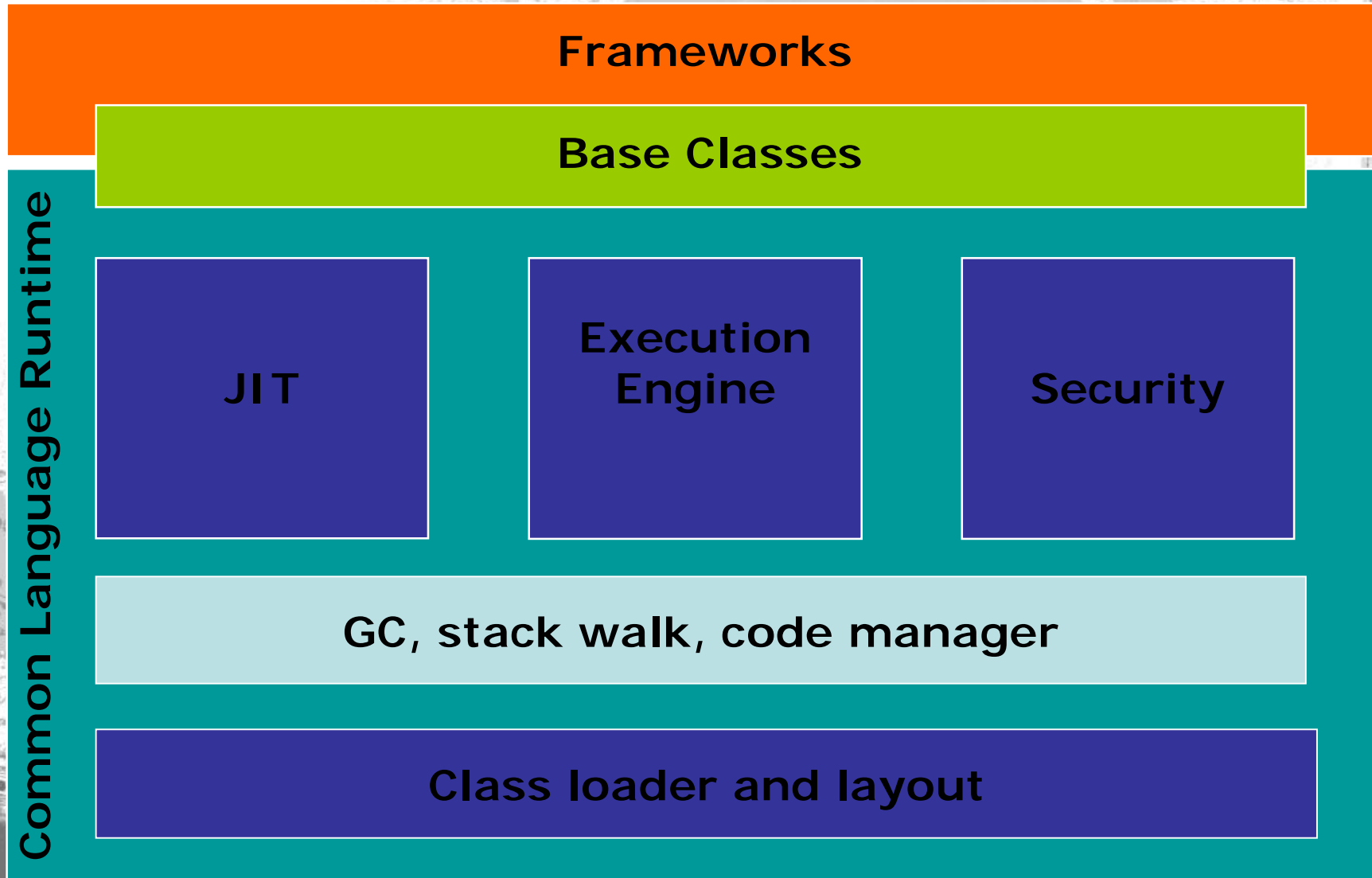




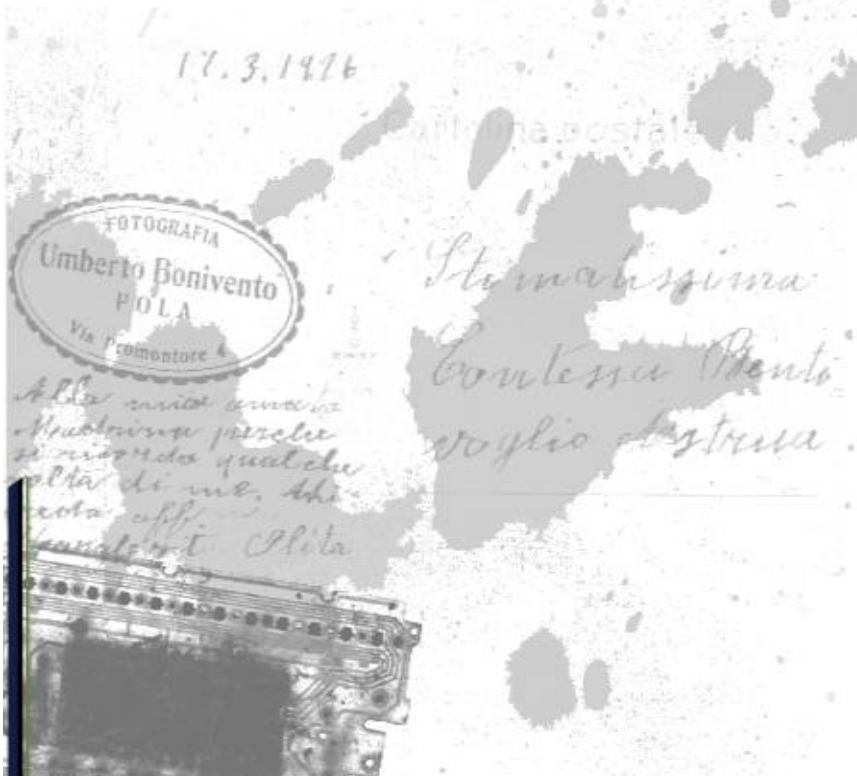
.NET Security

Flier@nsfocus.com

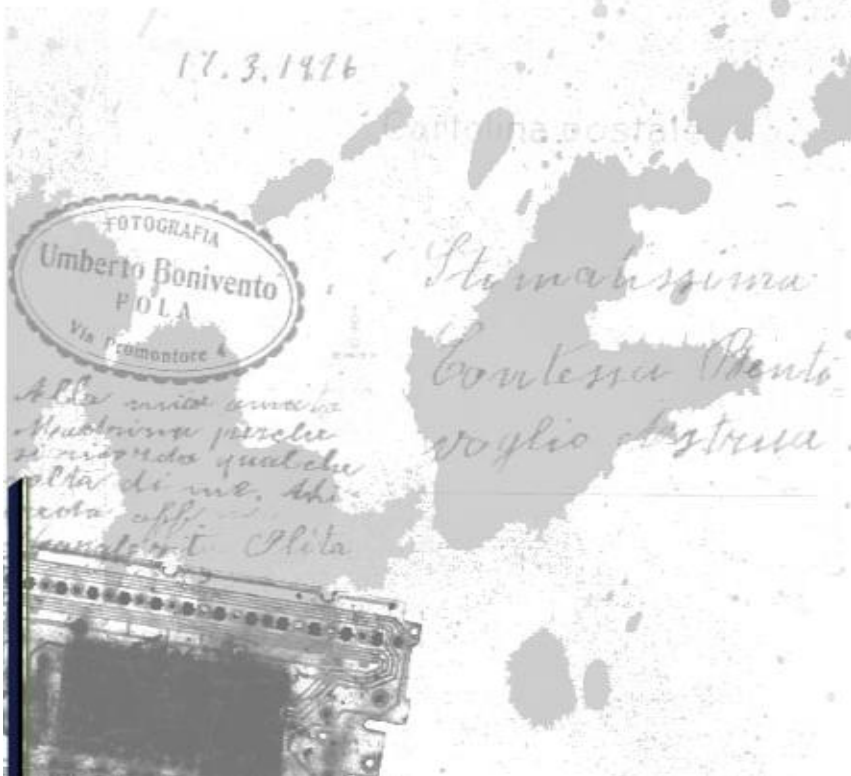




- Static Structure
- Dynamic Executing
- Security Framework

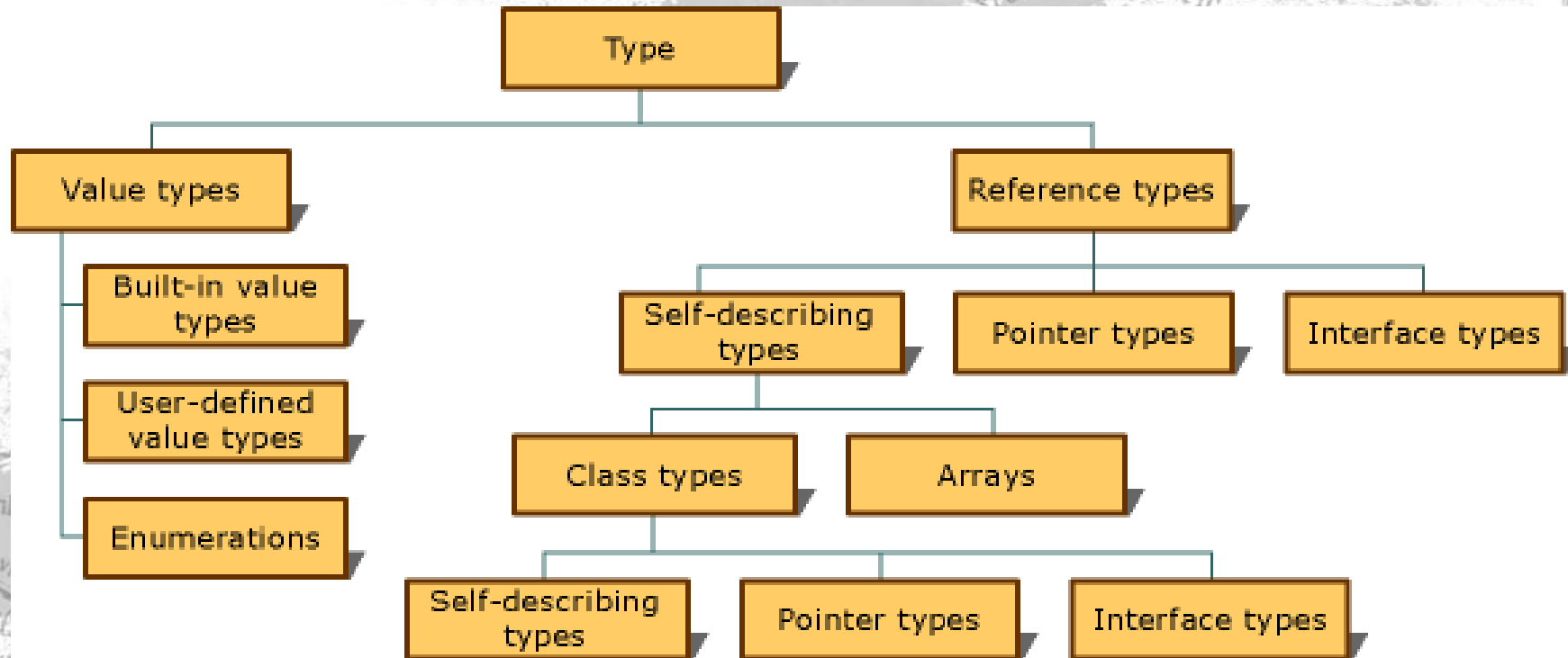


- CTS - Common Type System
- The Structure of a Managed Executable File
- IL Instructions (IL - Intermediate Language)

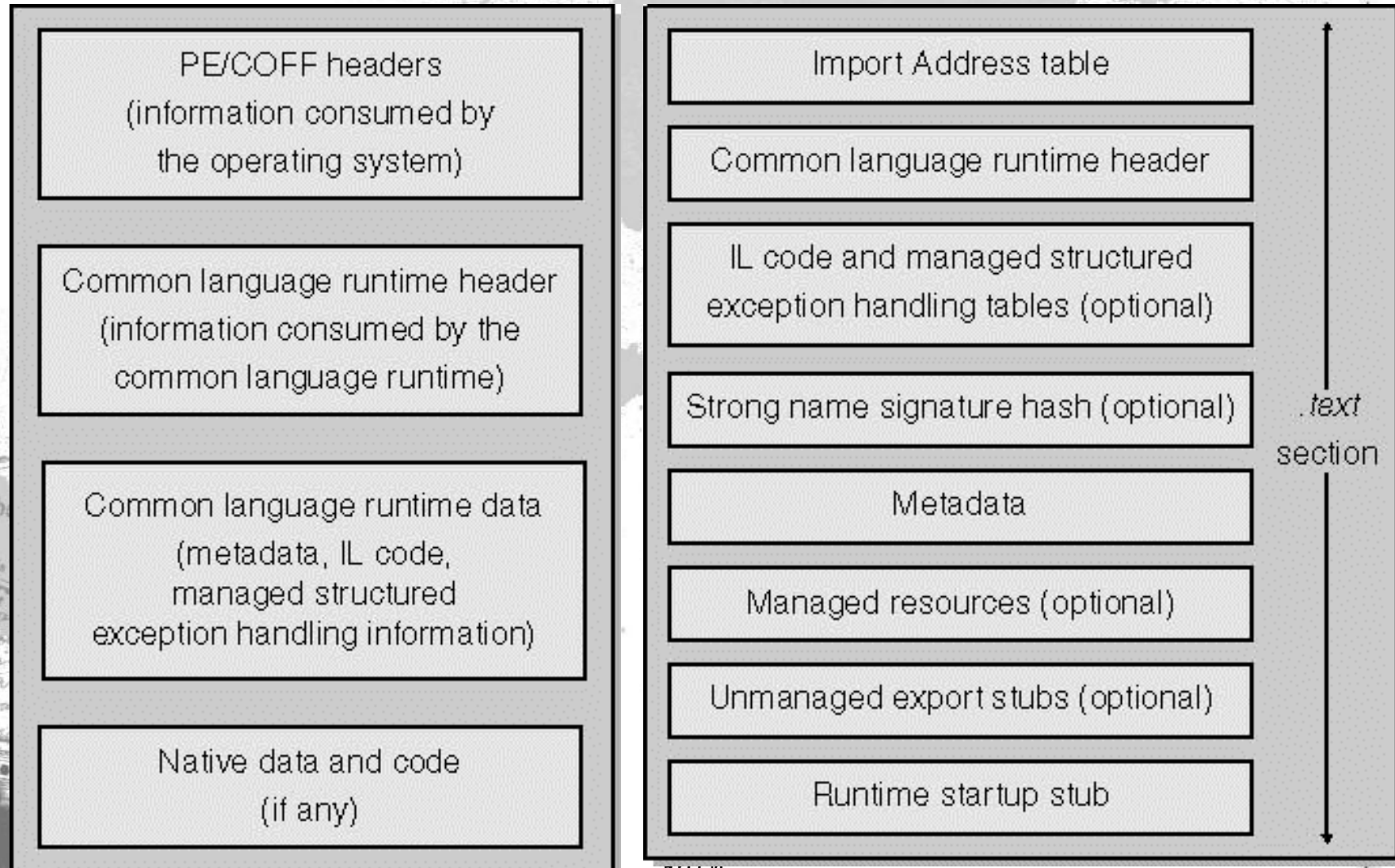


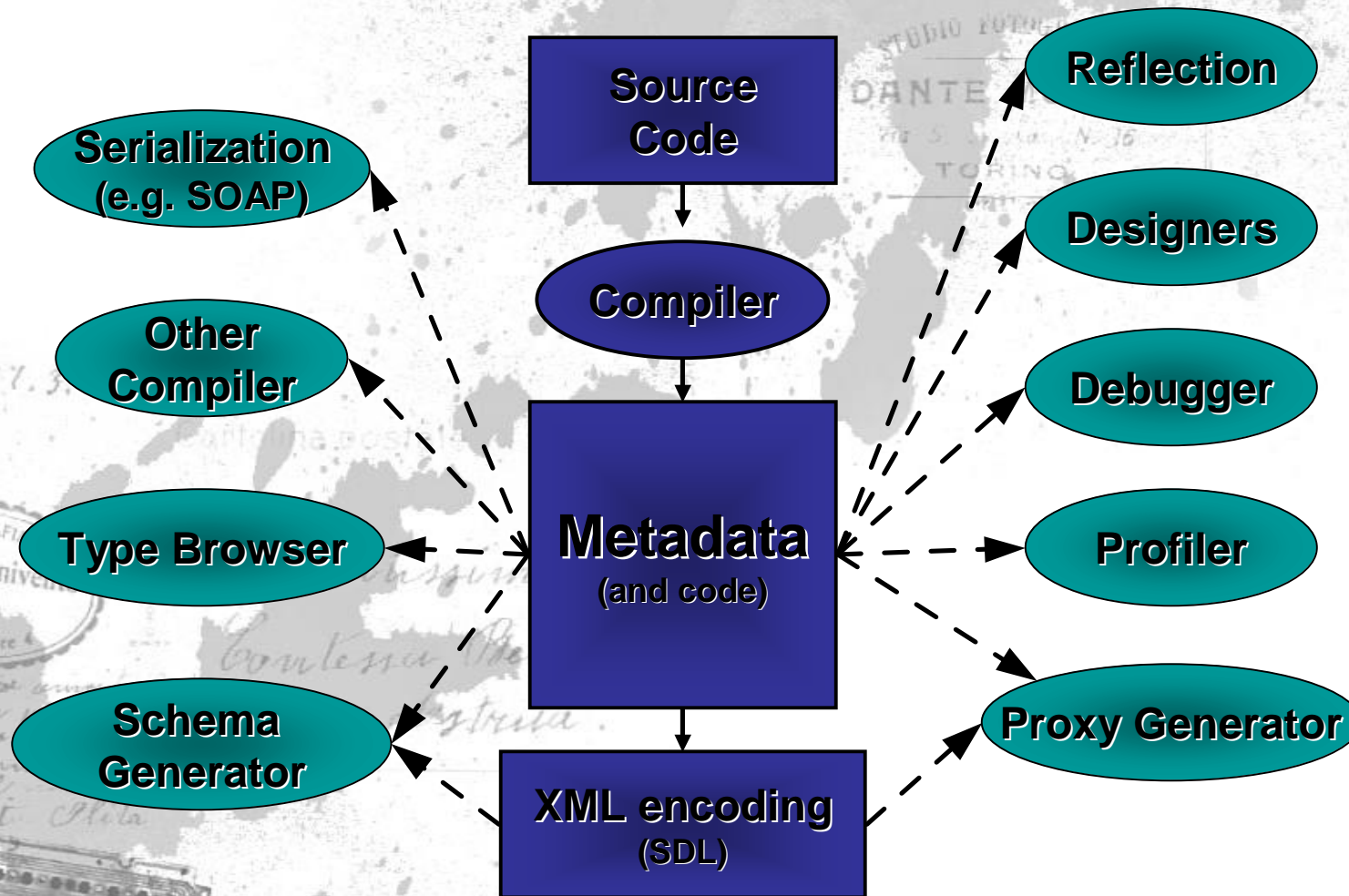
- Goals
 - Define how to declare, use and managed Type
 - Base of CLR Language Integration
- Features
 - Verification ensures type safety
 - High executing performance
 - Designed for Object-Oriented, Procedural, and Functional Languages
 - Define meta-rule of language

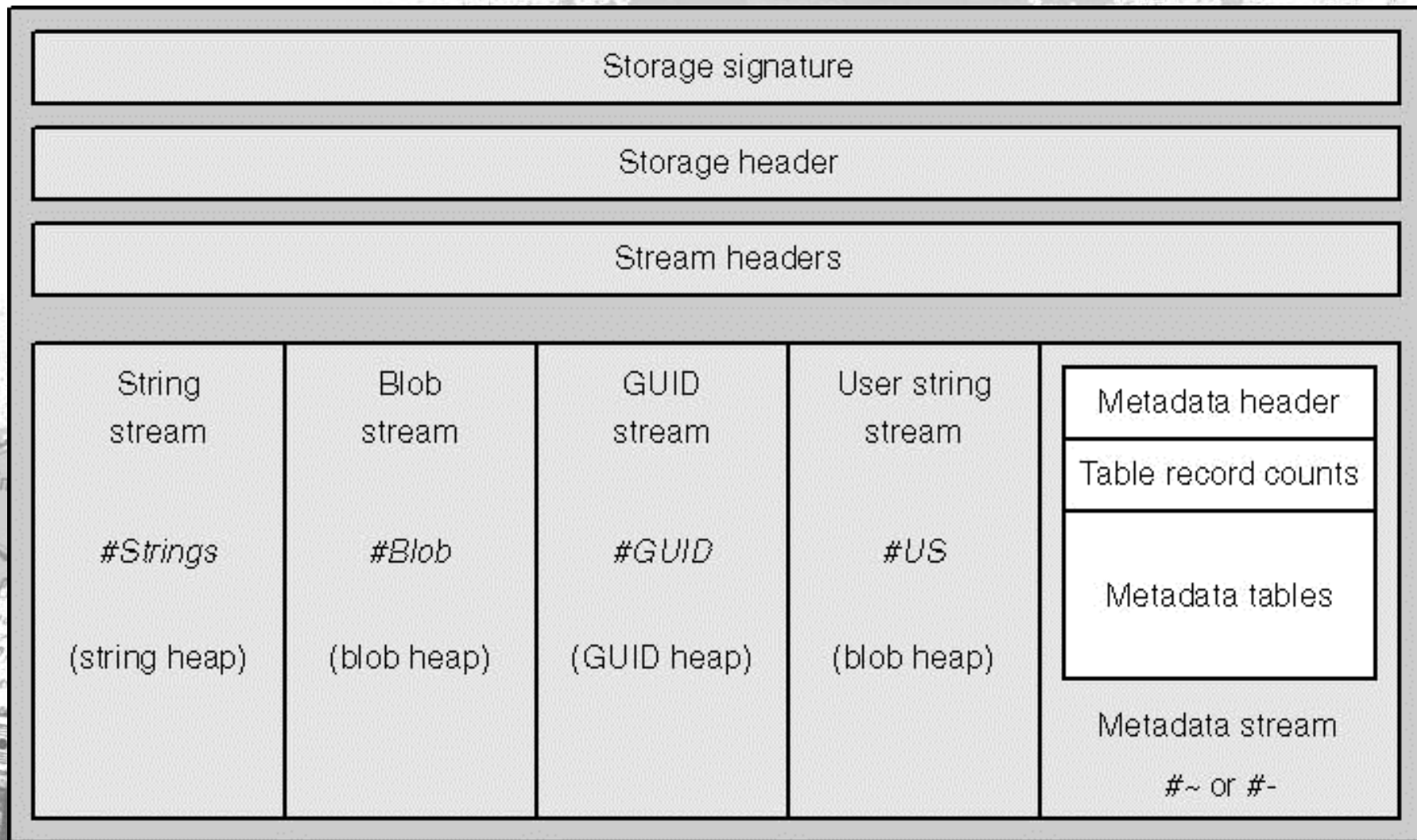
- Assemblies
- Types
- Members
 - Methods
 - Parameters and local variables
 - Fields
 - Nested Types
 - Properties
 - Events



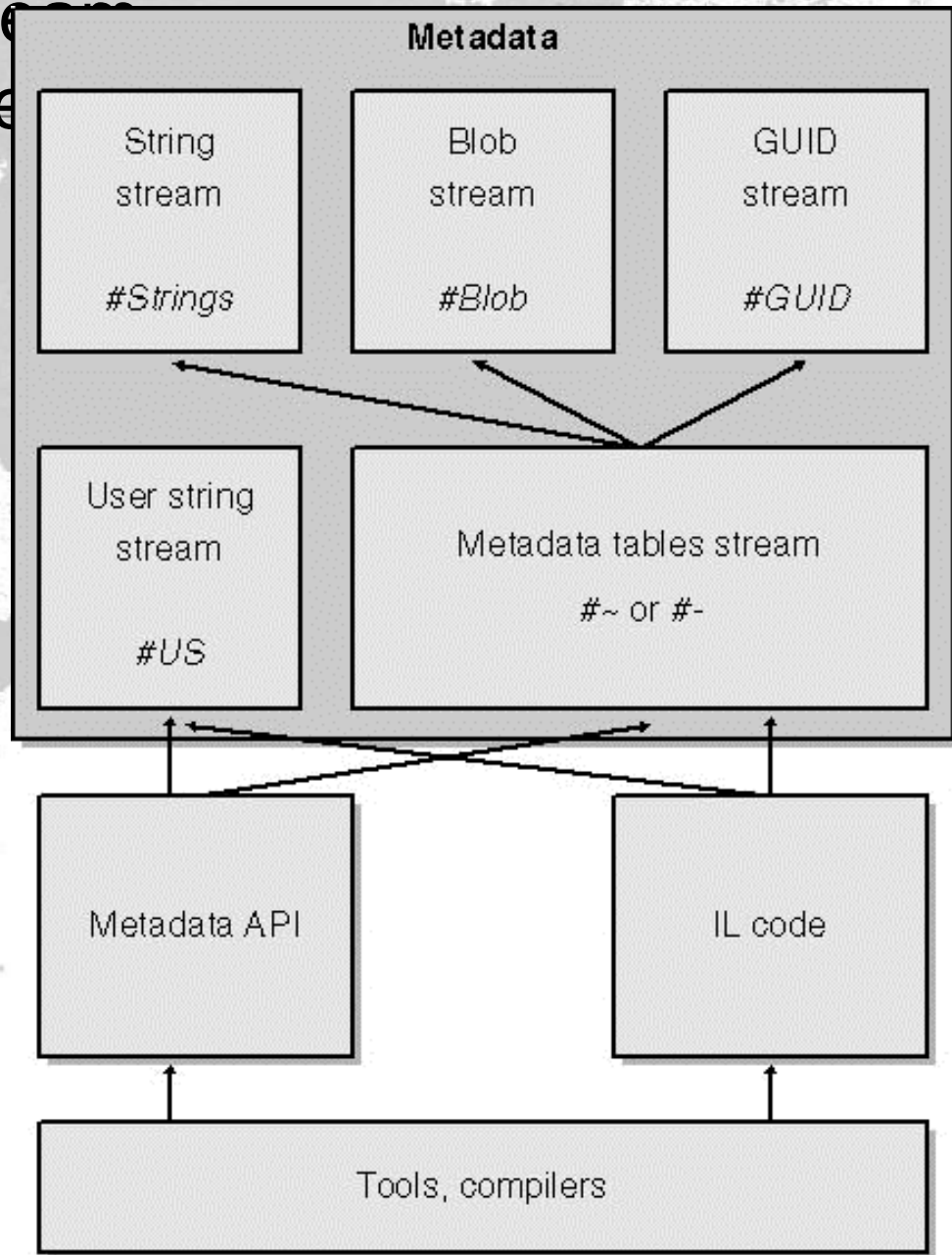
1.2. The Structure of a Managed Executable File





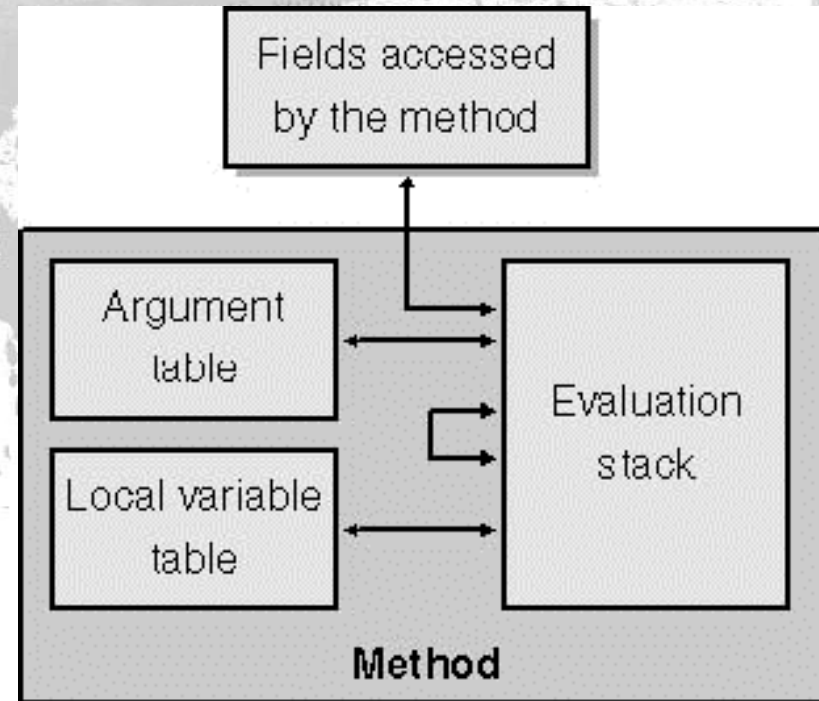


- #Strings
 - the names of metadata items
- #Blob
 - internal metadata binary objects, such as default values.
- #GUID
 - all sorts of globally unique identifiers.
- #US
 - user-defined strings.
- #~ 或 #-
 - compressed (uncompressed) metadata stream

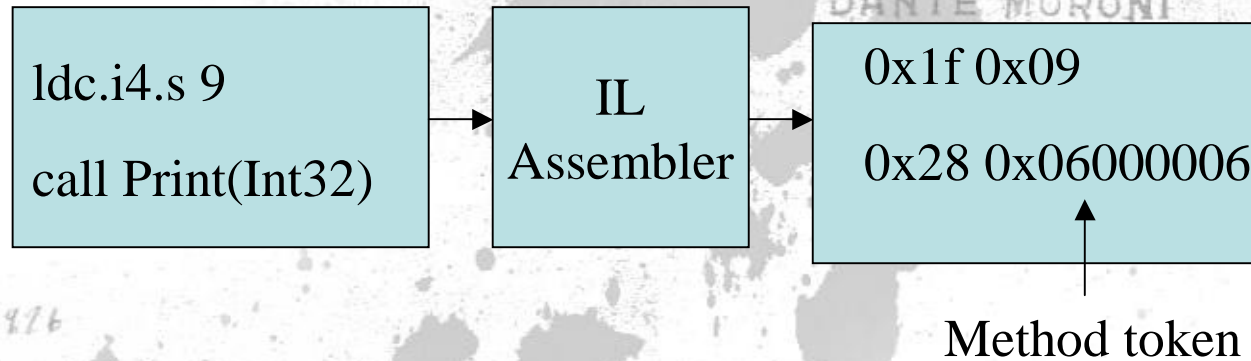


Module	TypeRef	TypeDef
FieldDef	MethodDef	ParamDef
InterfaceImpl	MemberRef	CustomAttribute
Permission	Signature	Event
Property	ModuleRef	TypeSpec
Assembly	AssemblyRef	File
ExportedType	ManifestResource	GenericParam
GenericParamConstraint	MethodSpec	

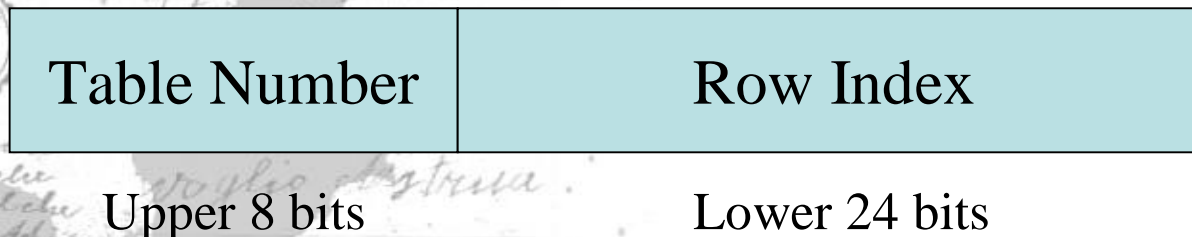
- Stack based model
- Code Verifiability
- Opcodes are either 1 byte or 2 bytes long
 - 0xFE = first byte of two byte opcodes
 - Pseudo-assembly
 - Uses "tokens" instead of offsets/pointers

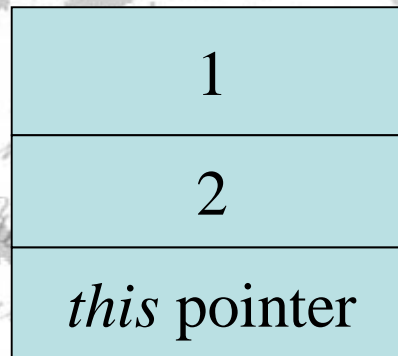
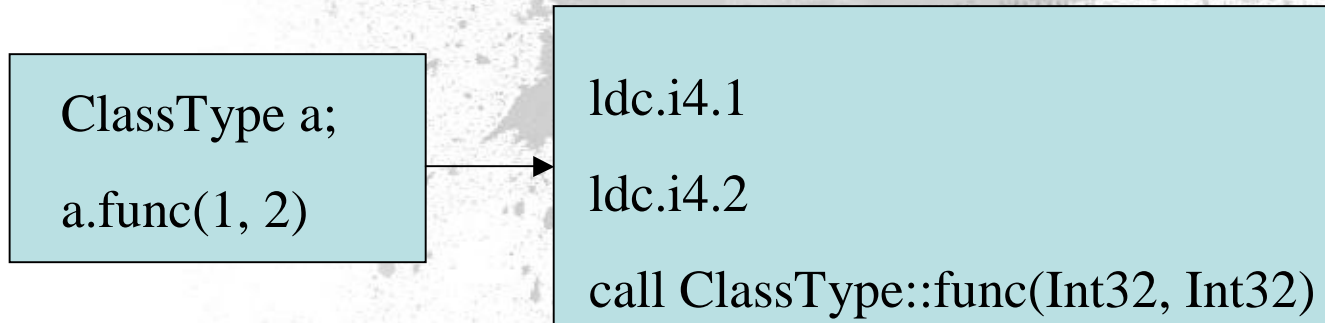


- Labels and Flow Control Instructions
 - br, brtrue, beq, leave, endfinally, ret
- Arithmetical Instructions
 - add, shl, ldc.i4, conv.i8, add.ovf
- Addressing Arguments and Local Variables
 - ldarg.0, starg, ldloc , stloc
- Addressing Fields
 - ldfld, ldsfld, stfld
- Calling Methods
 - call, callvirt, ldftn, ldvirtftn, calli, jmp
- Addressing Classes and Value Types
 - newobj, ldobj, ldstr, isinst, box
- Vector Instructions
 - newarr, ldlen, ldelem.r4, stelem.ref



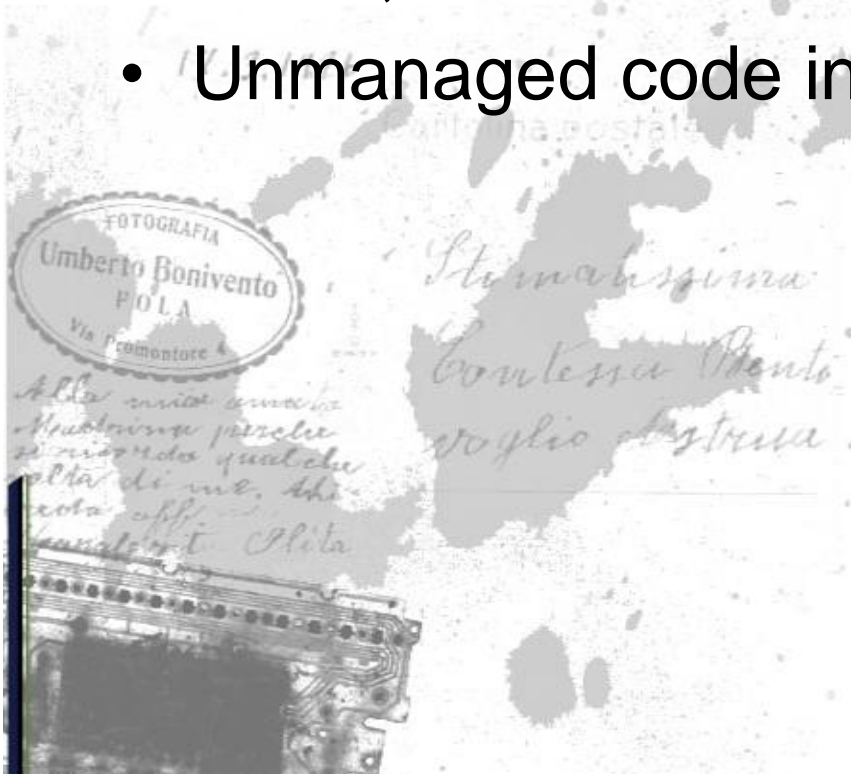
Token

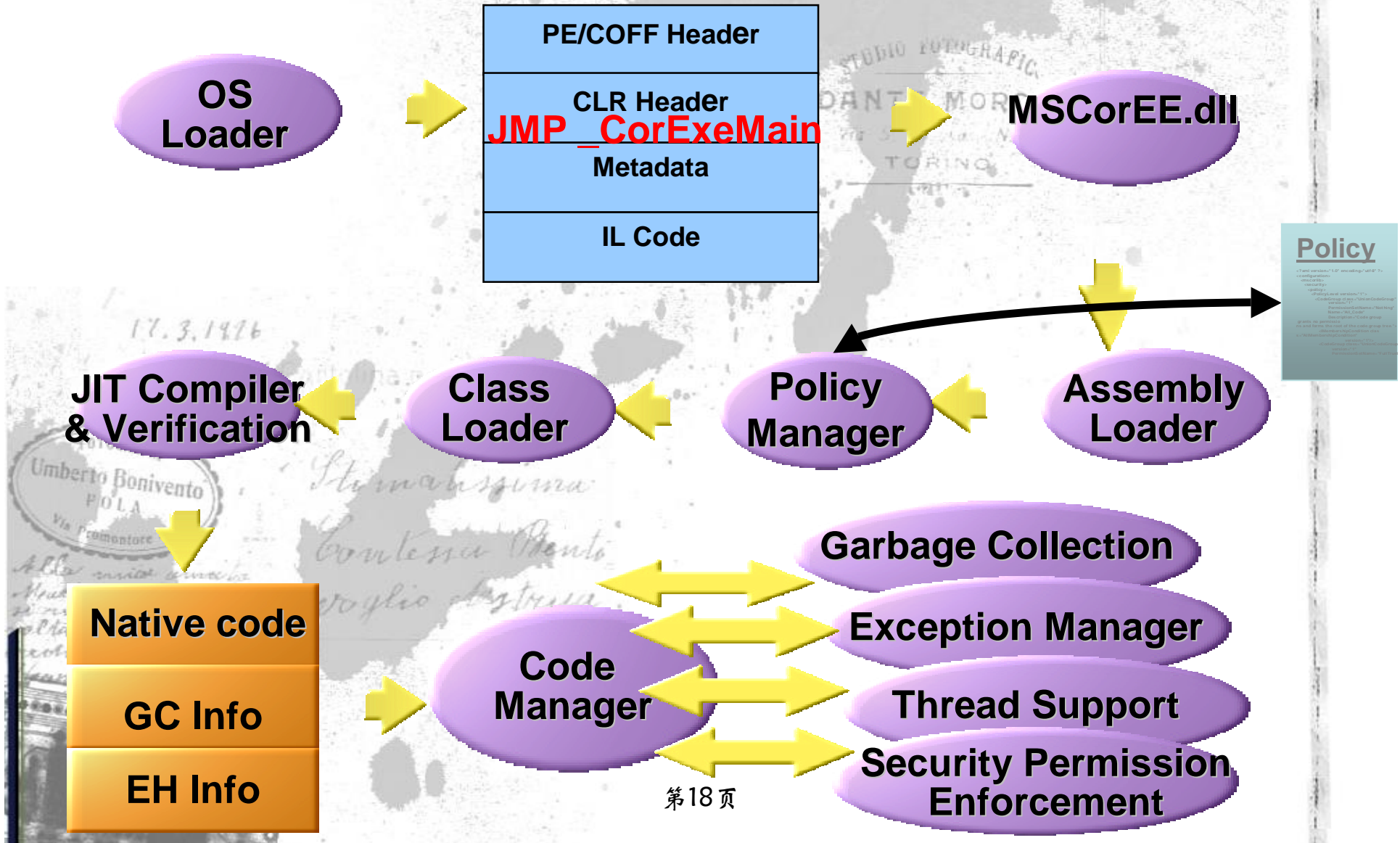




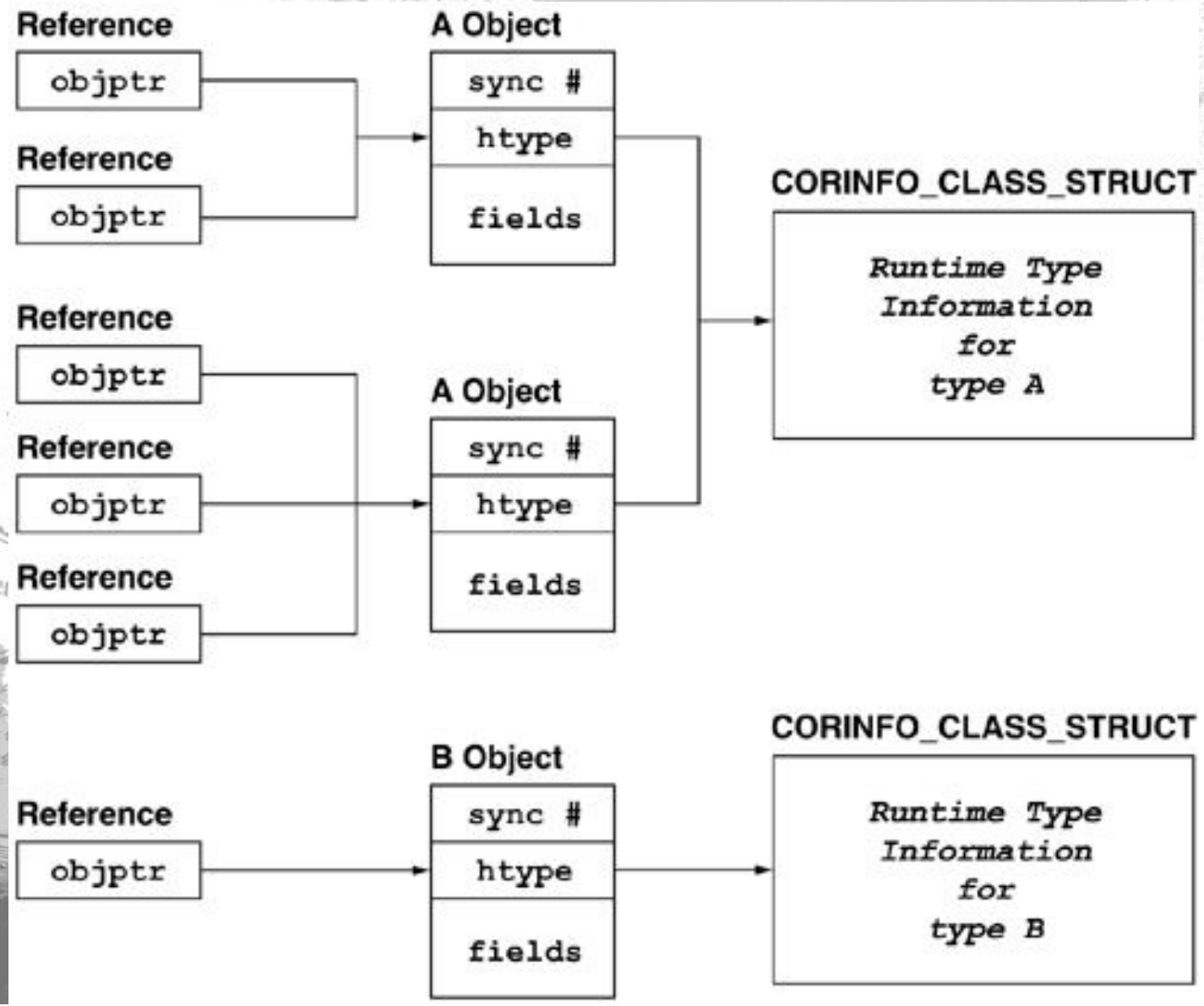
Stack top

- CLR Loader (fusion)
- Type and object memory layout
- JIT Compilation and Method Invocation
- Stack, Frame and Stack Walk
- Unmanaged code interop

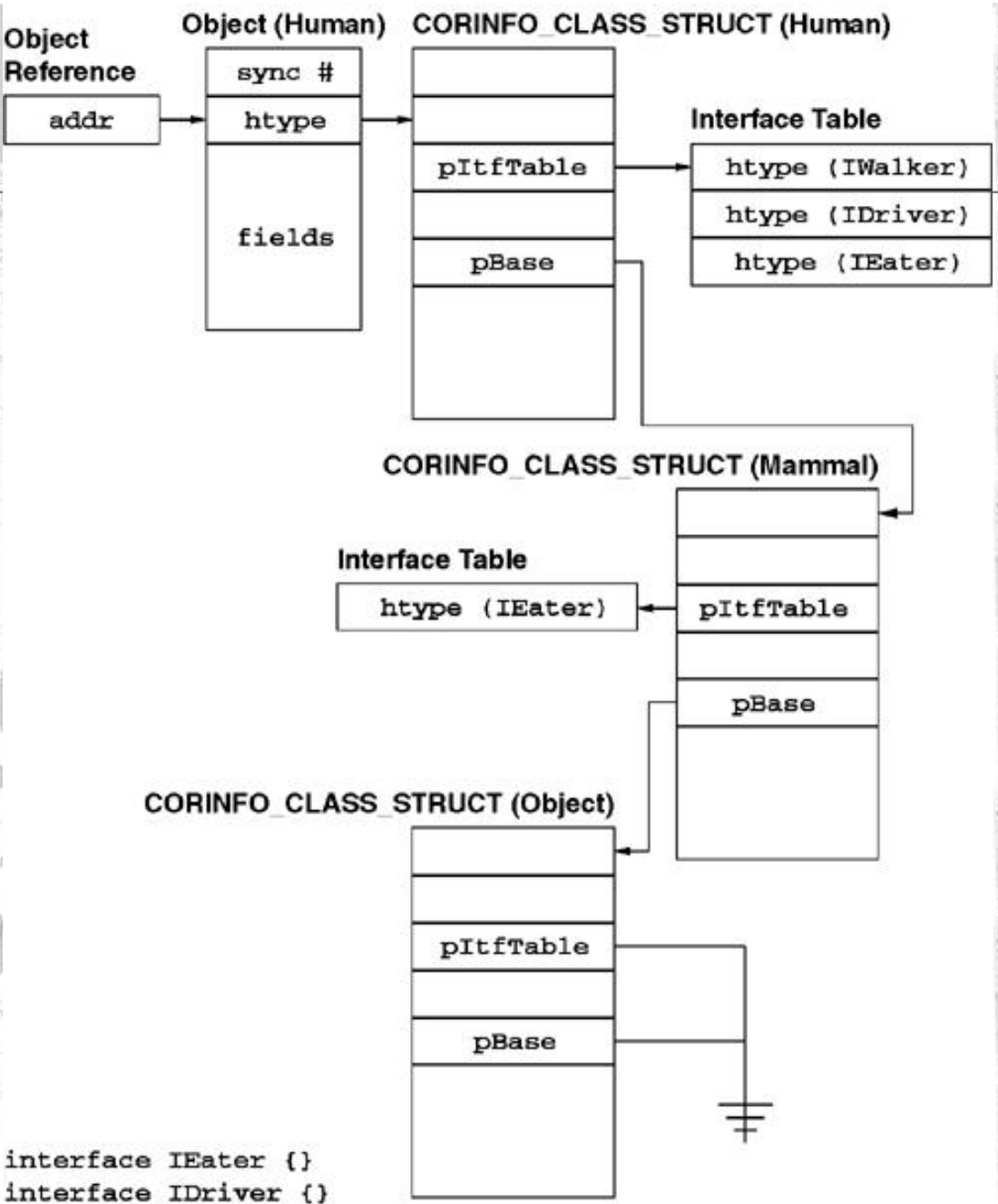




2.2. Type and object memory layout

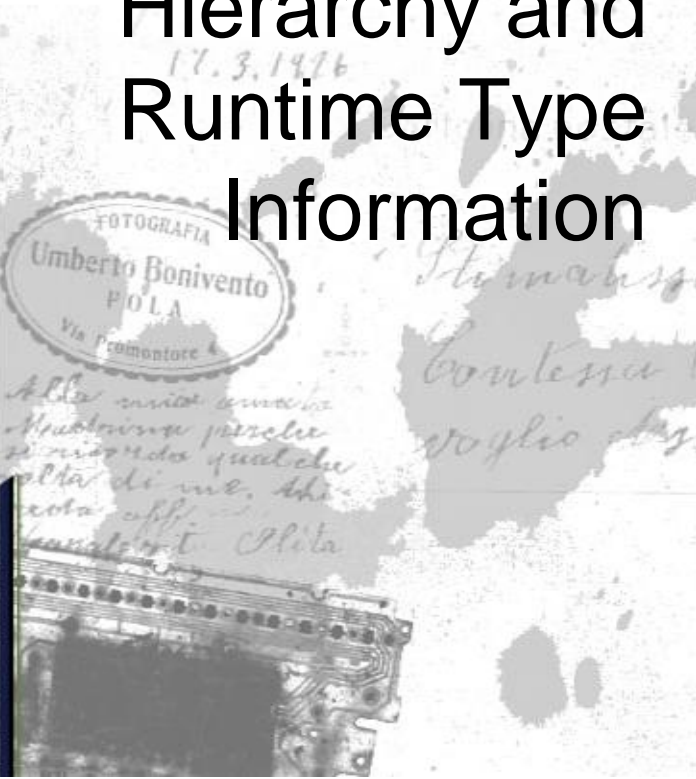


2.2.2. Type Hierarchy and Runtime Type Information



```

interface IEater {}
interface IDriver {}
interface IWalker {}
class Mammal : IEater {}
class Human : Mammal, IWalker, IDriver {}
  
```



- 2.3.1.JIT Compilation and Method Table
- 2.3.2.Virtual Functions Using Class-Based References
- 2.3.3.Virtual Functions Using Interface-Based References
- 2.3.4.Asynchronous Method Invocation
- 2.3.5.Internal Method Invocation

CORINFO_CLASS_STRUCT for Bob

CORINFO_CLASS_STRUCT for Bob	
Bob	
cMethods (9)	
Tostring	→ call 0013EA50
Equals	→ call 0013EA50
GetHashCode	→ call 0013EA50
Finalize	→ call 0013EA50
.ctor	→ call 0013EA50
a	→ call 0013EA50
b	→ call 0013EA50
c	→ jmp 038C01CA
f	→ jmp 038C0100

IA-32 Native Code for JIT Compiler

```
mscorlib.dll!PreStubWorker
```

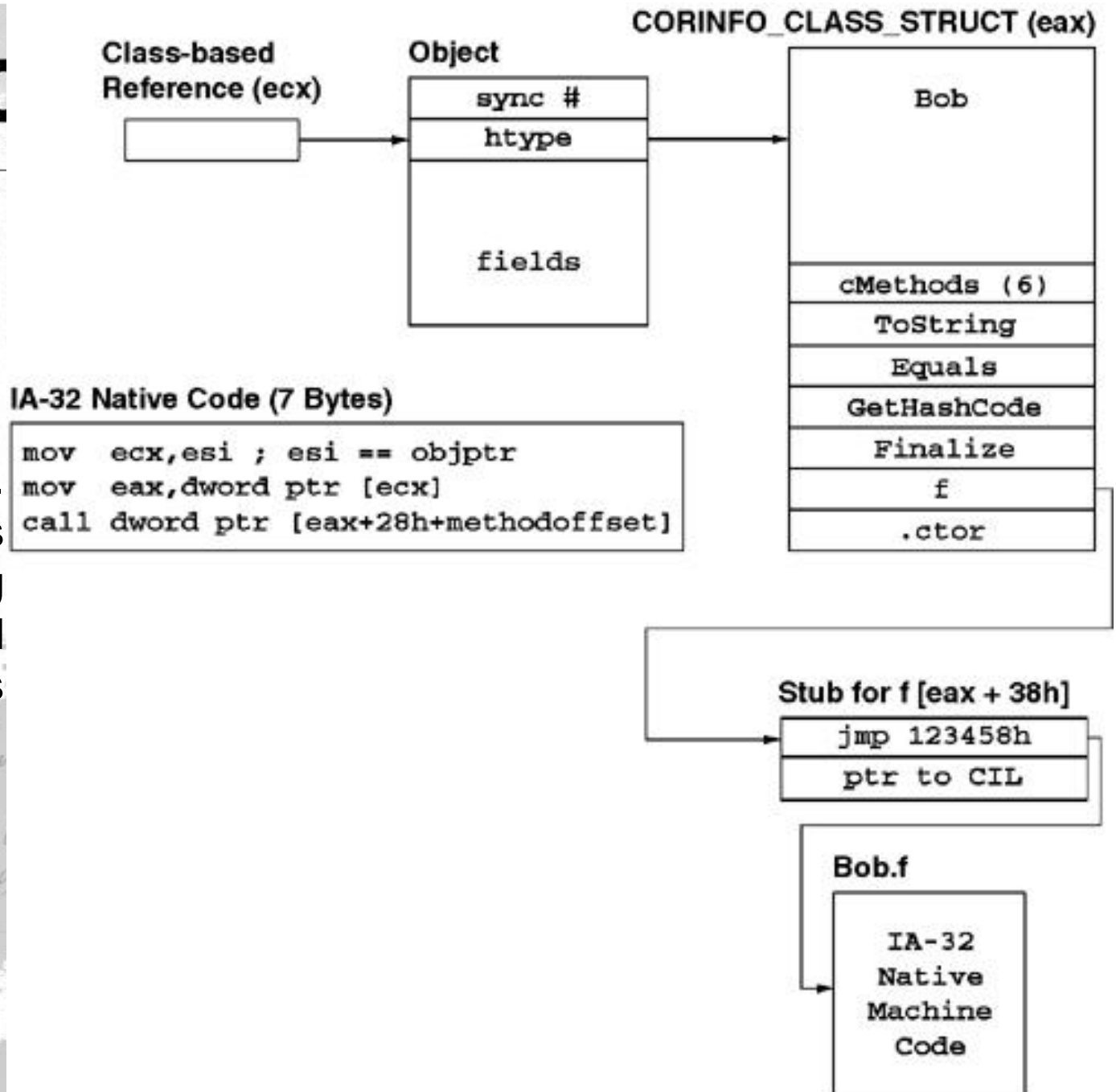
IA-32 Native Code for Bob.c

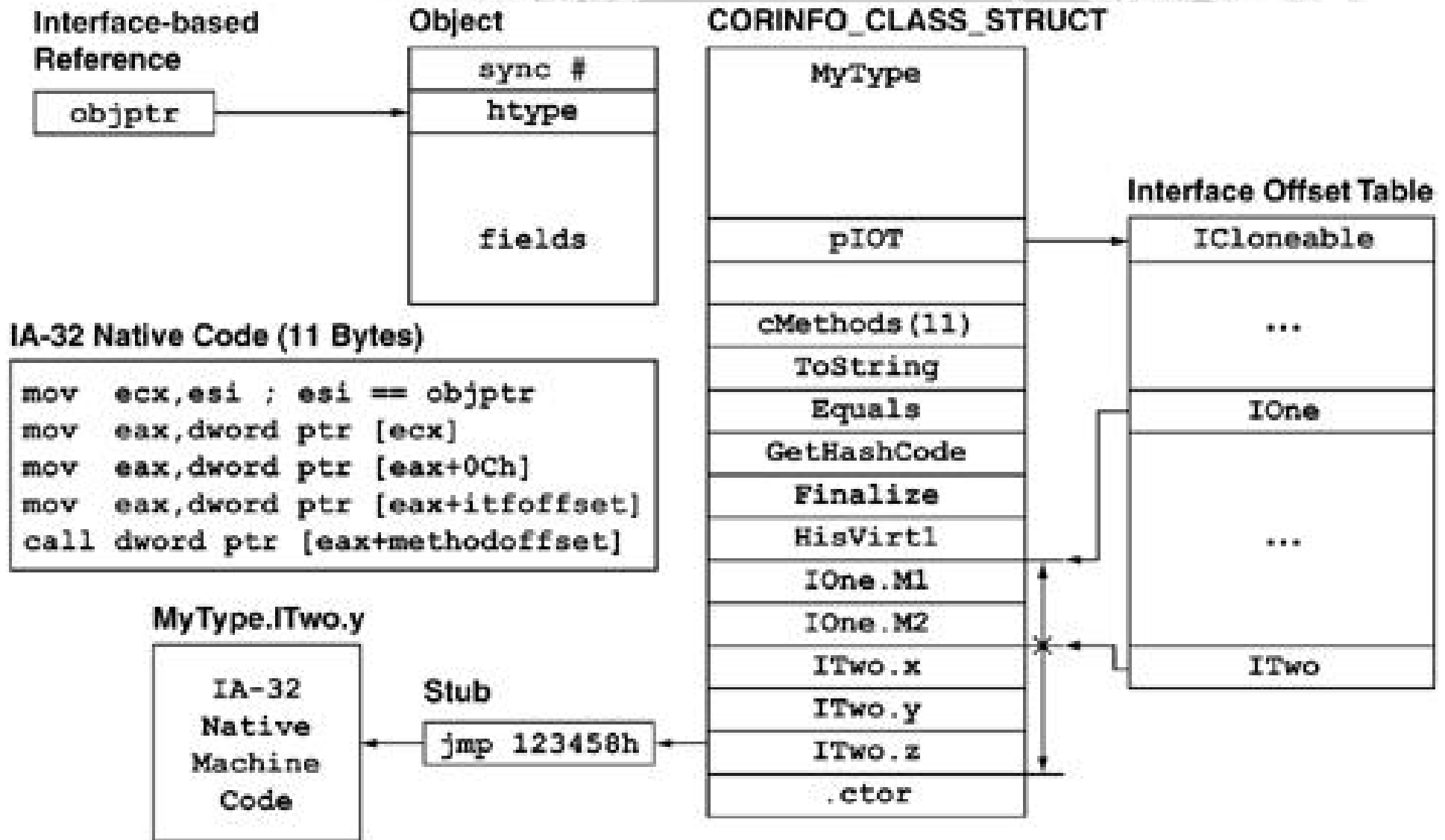
```
push ebp
mov  ebp, esp
    add  dword ptr ds : [41BC68h], 4
pop  ebp
ret
```

IA-32 Native Code for Bob.f

```
push ebp
mov  ebp, esp
    call dword ptr ds:[37565Ch]
    call dword ptr ds:[375658h]
    call dword ptr ds:[375654h]
pop  ebp
ret
```

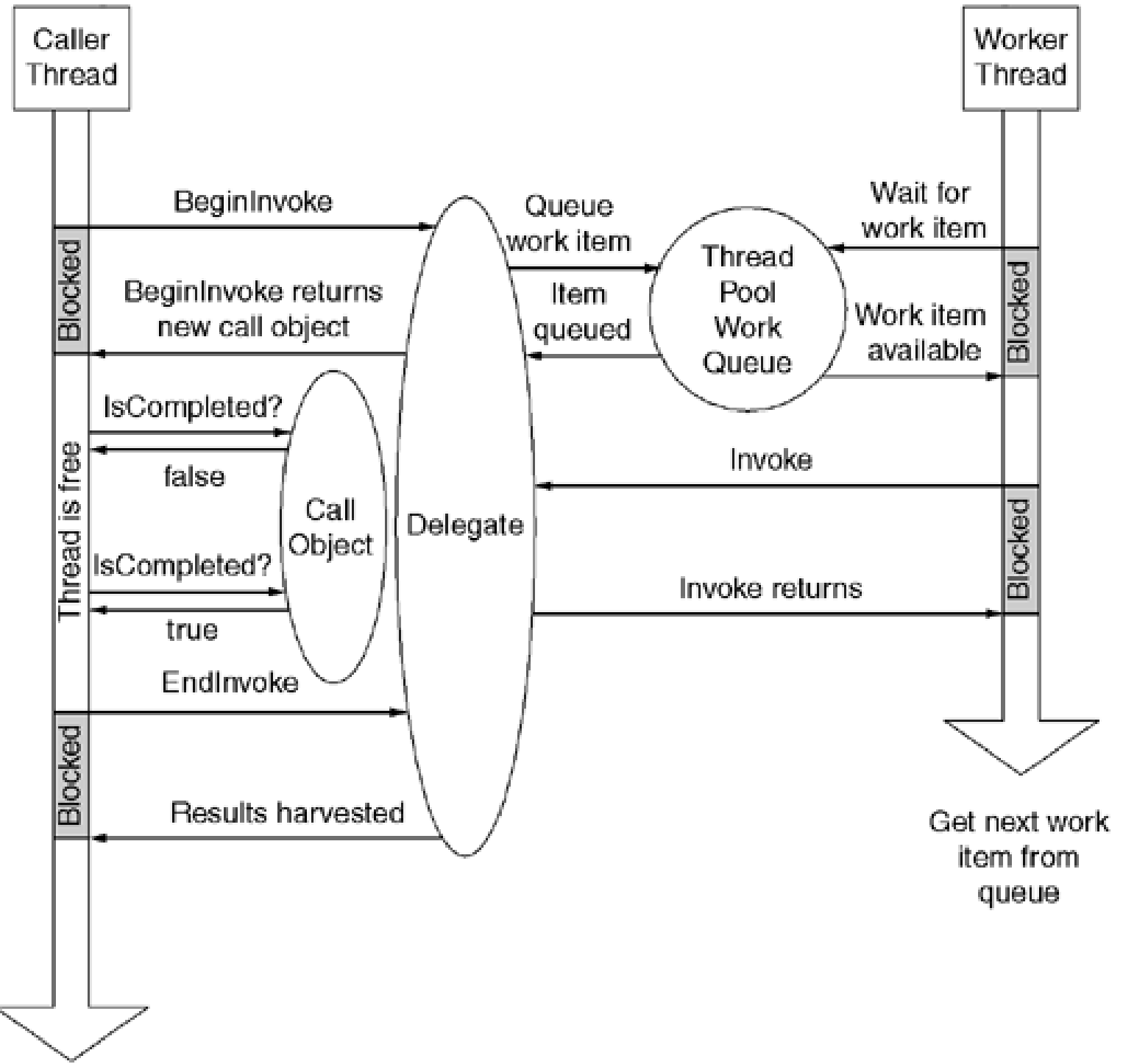
2.3.2. Virtual Functions Using Class-Based References



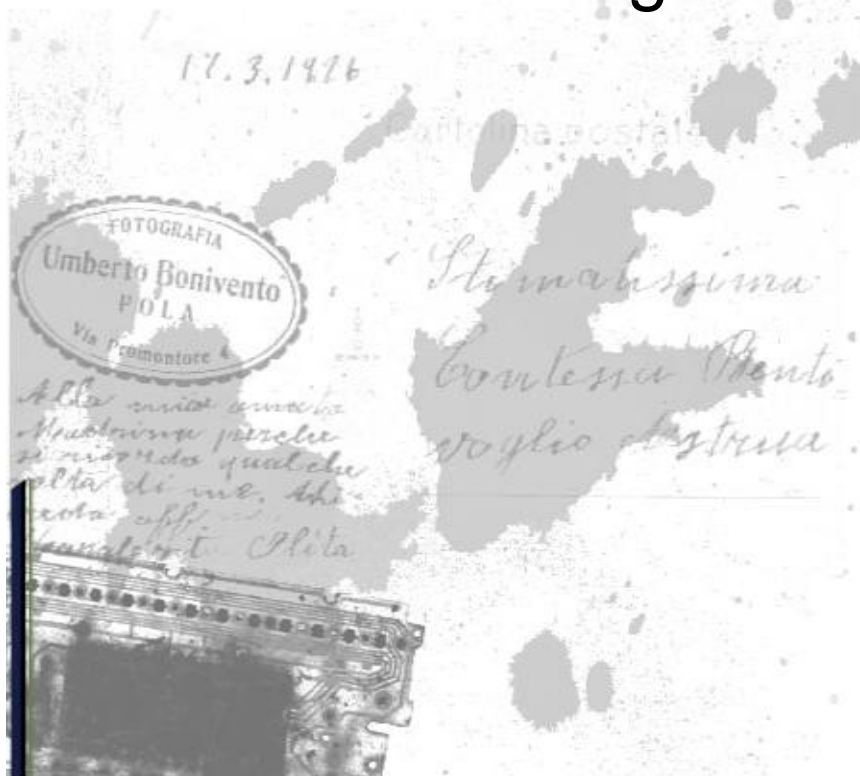




2.3.4 Asynchronous Method Invocation



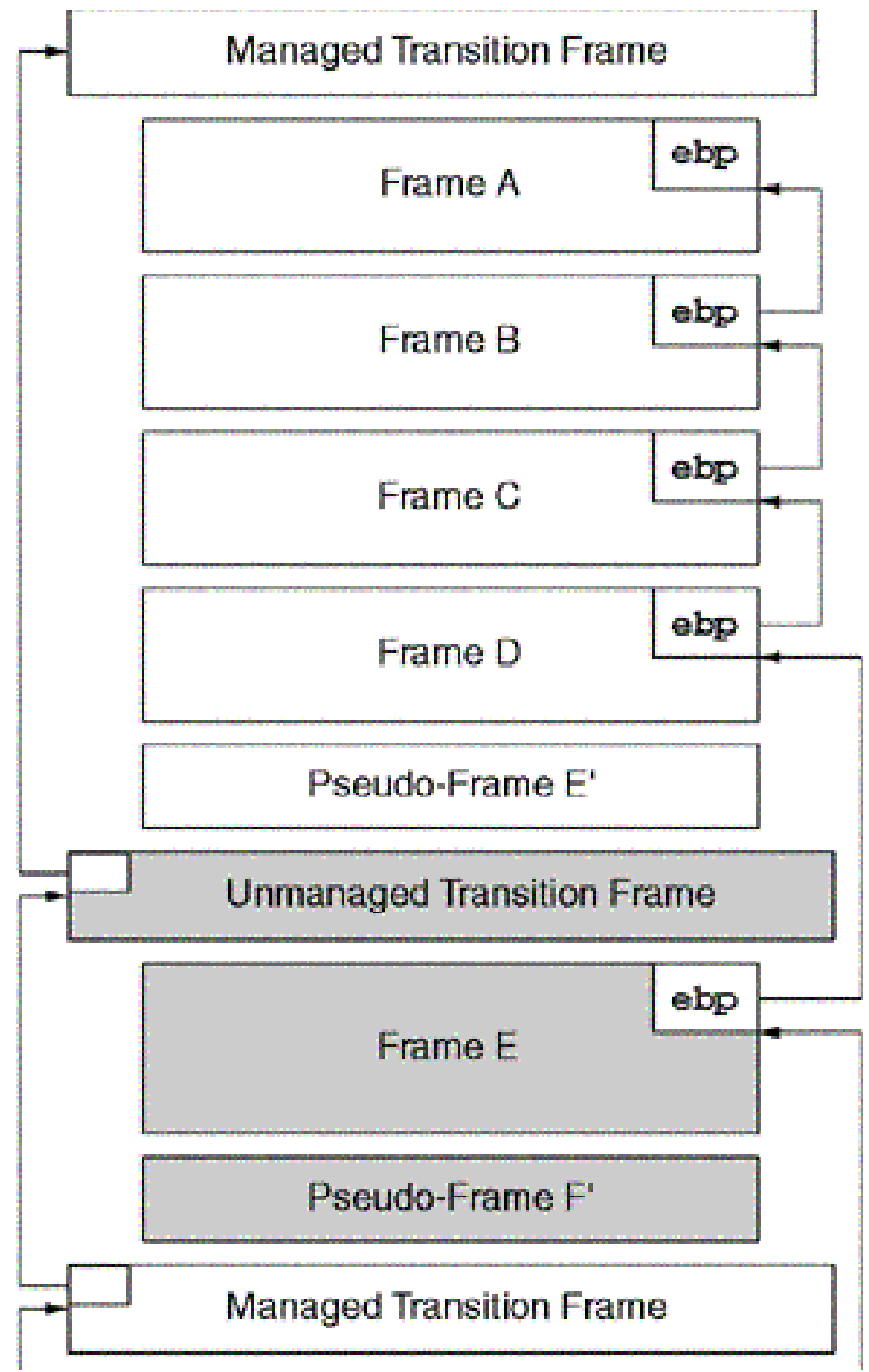
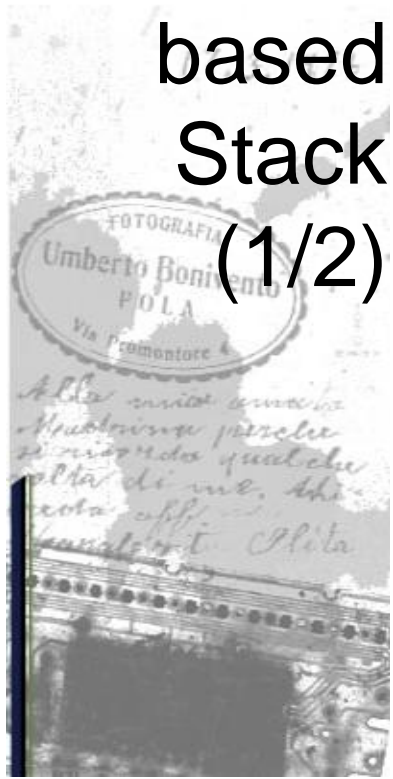
- 2.4.1. Frame-based Stack
- 2.4.2. Common Frame Type Hierarchy
- 2.4.3. Stack Walk Mechanics
- 2.4.4. Walking the stack





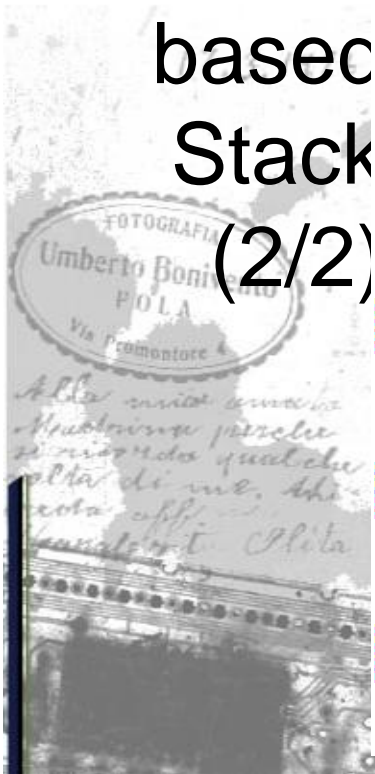
```
#pragma unmanaged
void H() {}
#pragma managed
void G() { H(); }
void F() { G(); }
#pragma unmanaged
void E() { F(); }
#pragma managed
void D() { E(); }
void C() { D(); }
void B() { C(); }
void A() { B(); }
```

2.4.1. Frame-based Stack (1/2)





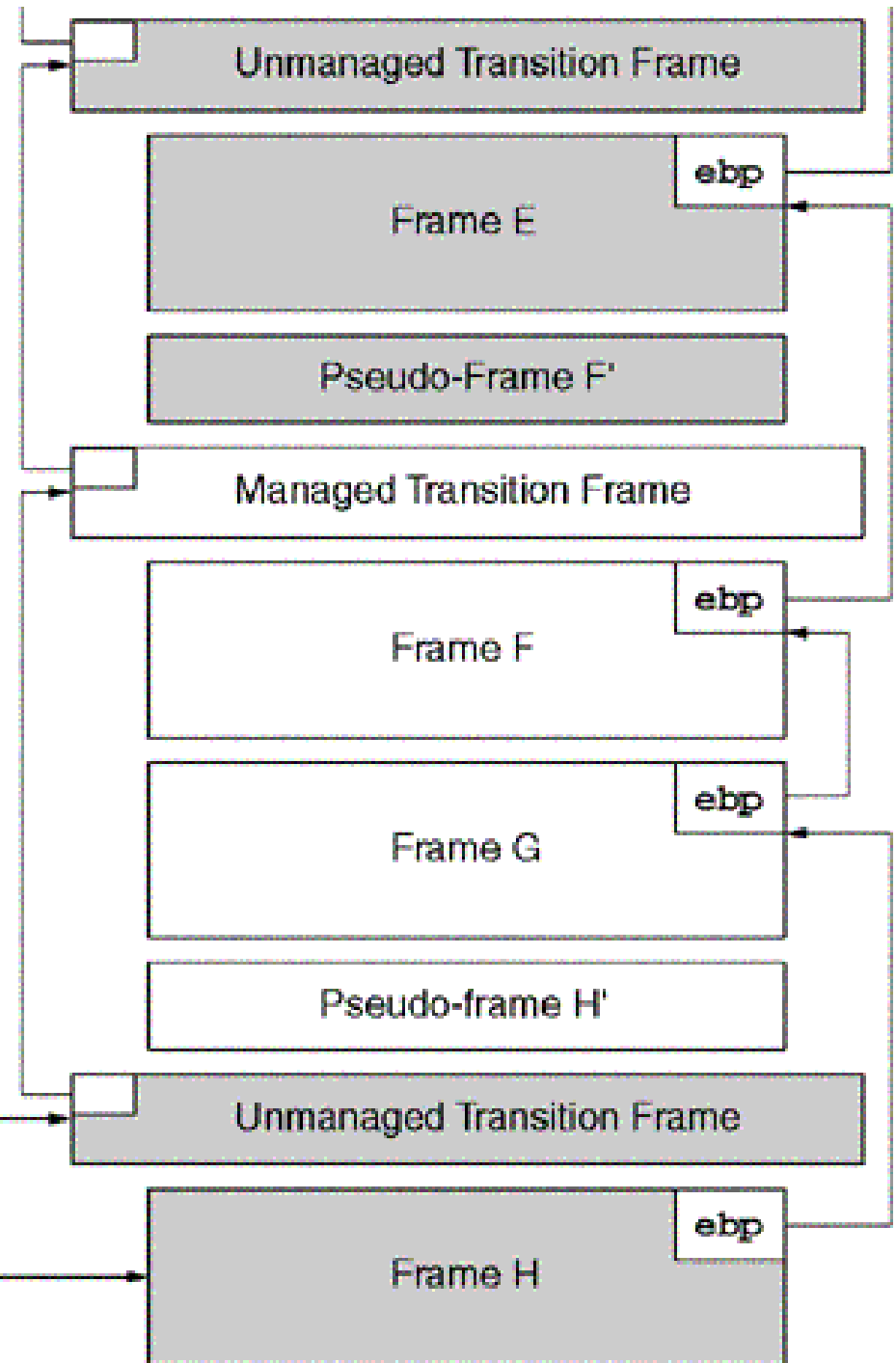
2.4.1. Frame-based Stack (2/2)



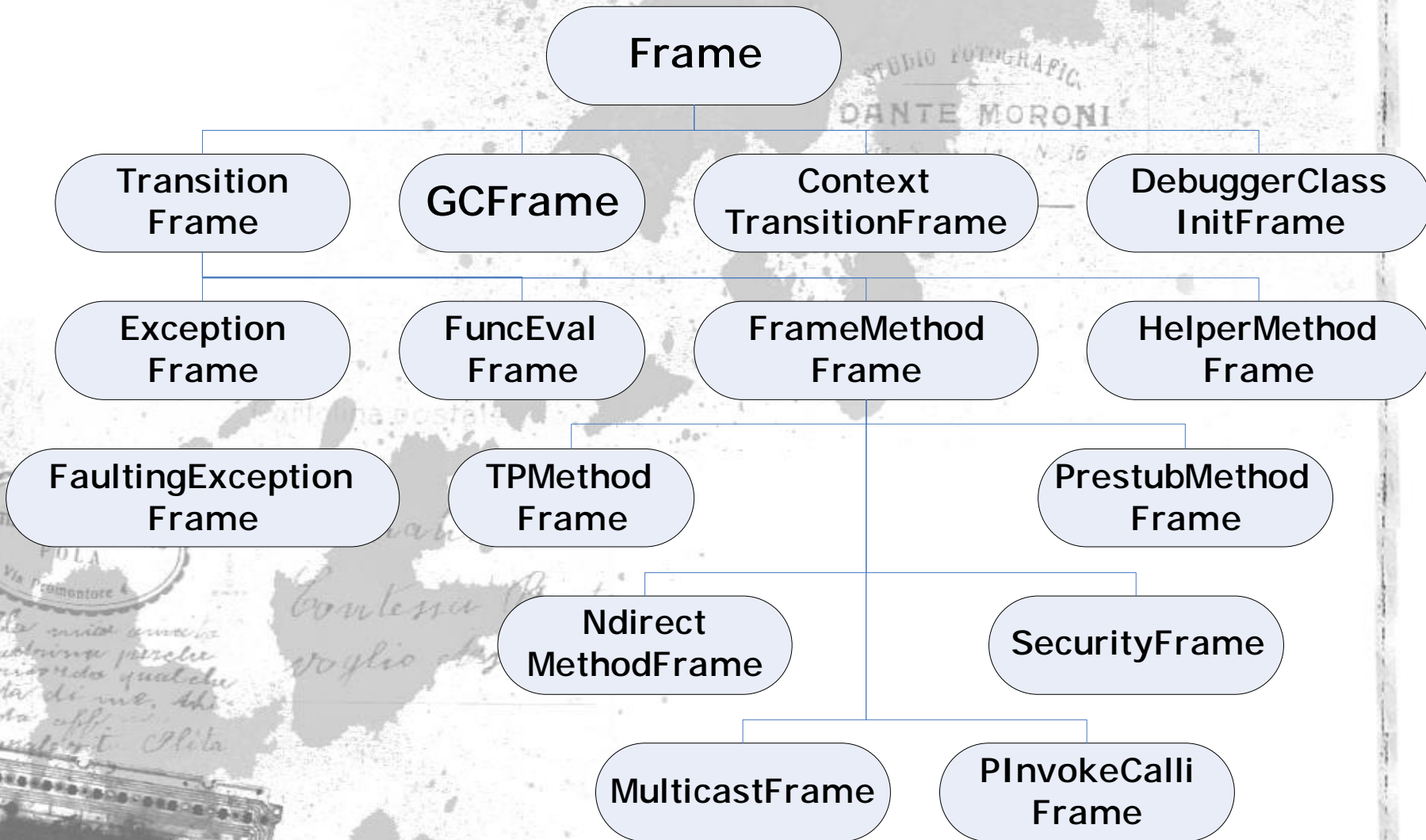
`tls.mode == unmanaged`

`tls.currentChain`

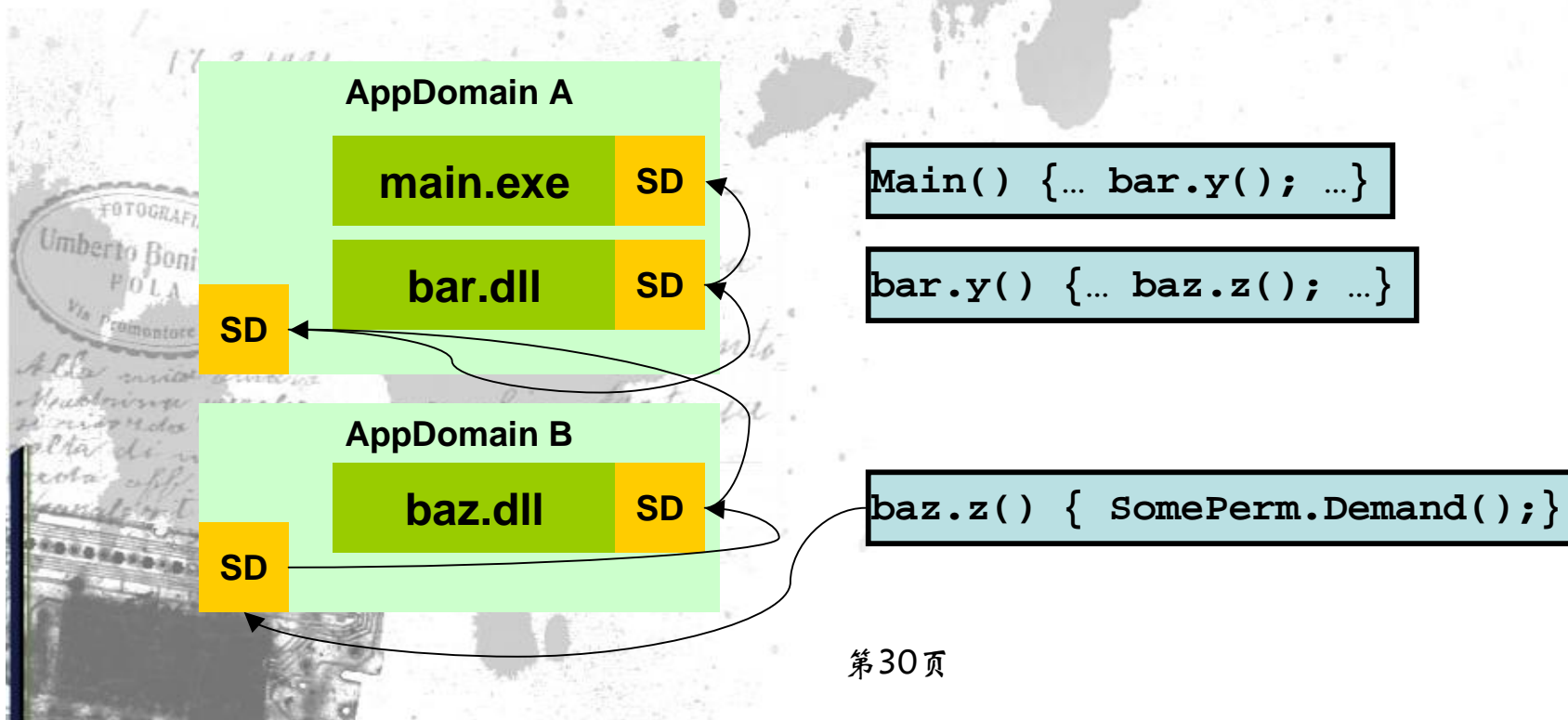
`esp`

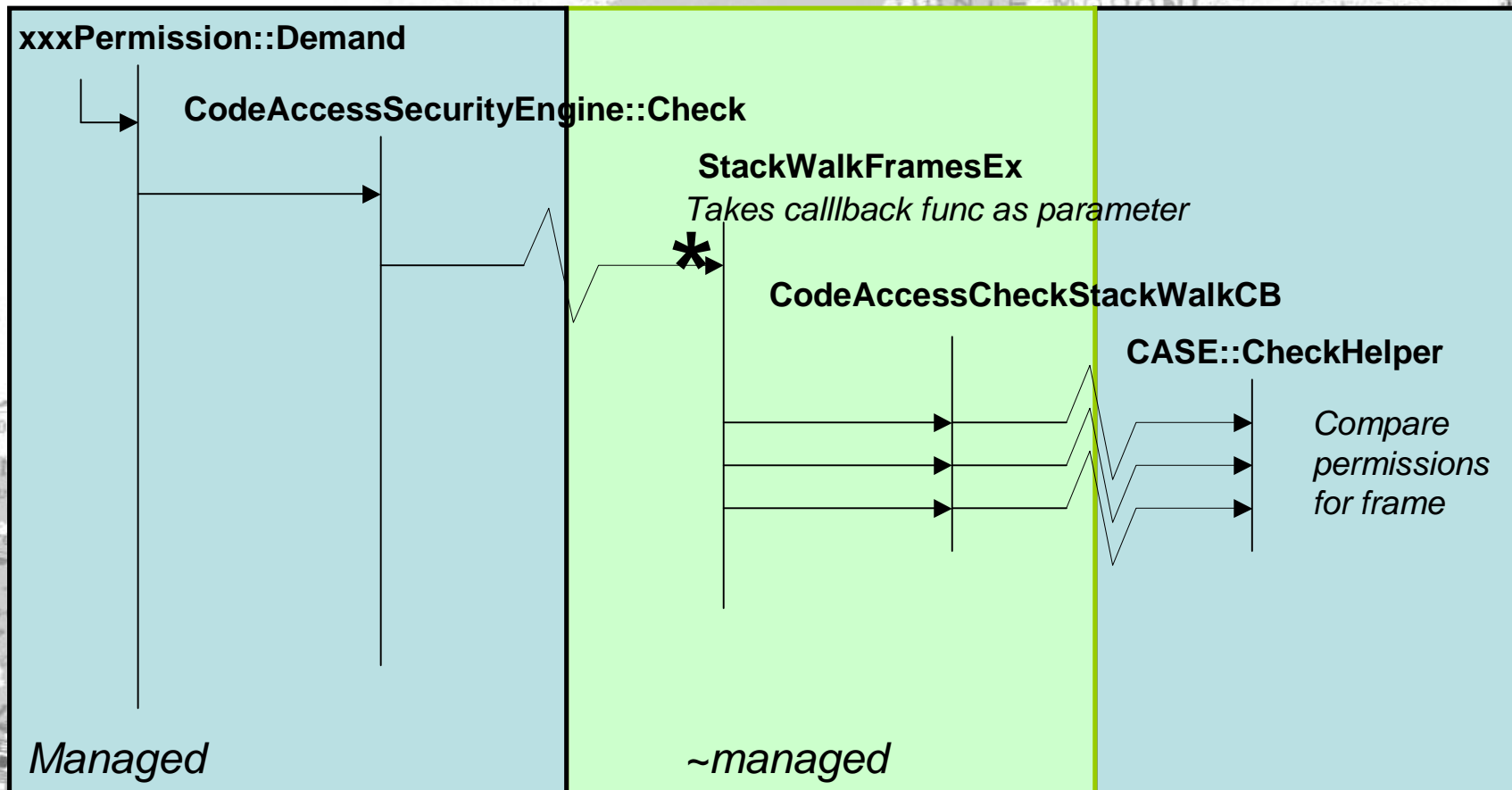


2.4.2. Common Frame Type Hierarchy

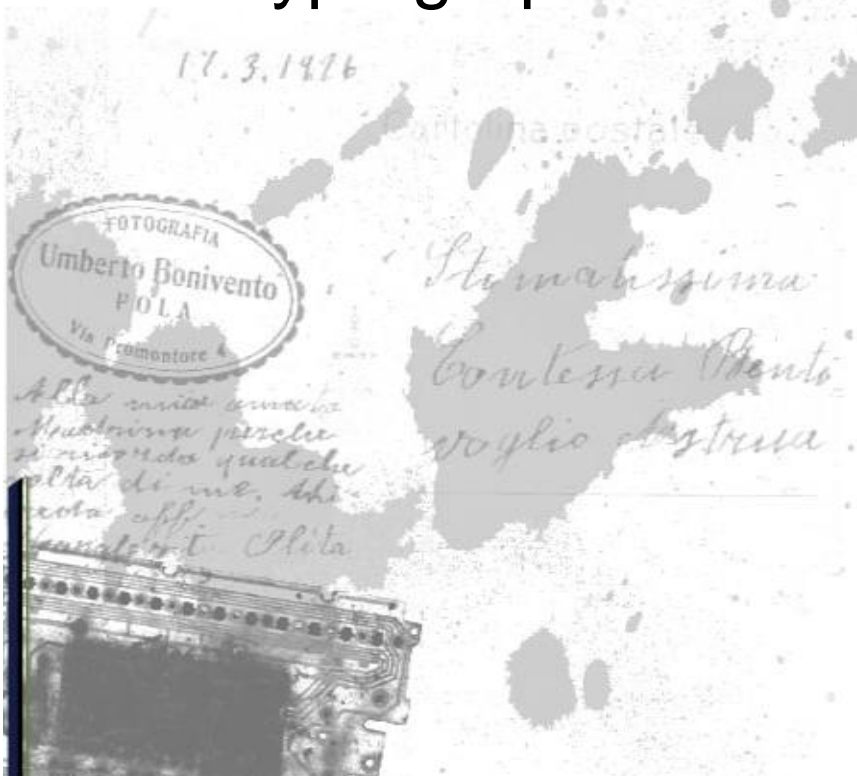


- Frame-based Stack Reverse Walk Algo.
- Mainly used in security and GC operations
- Done via unmanaged code

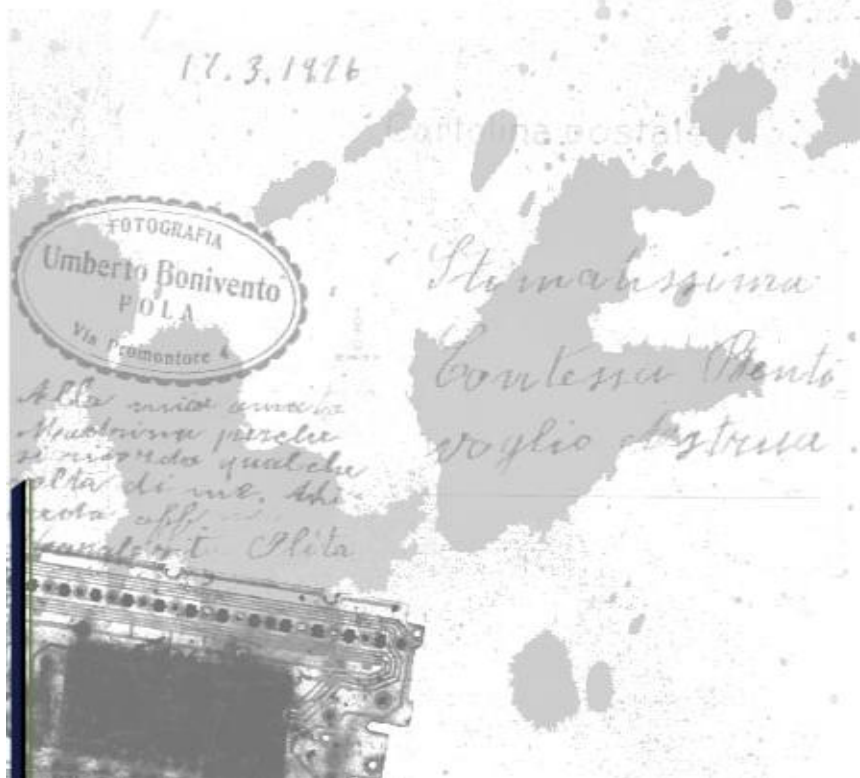




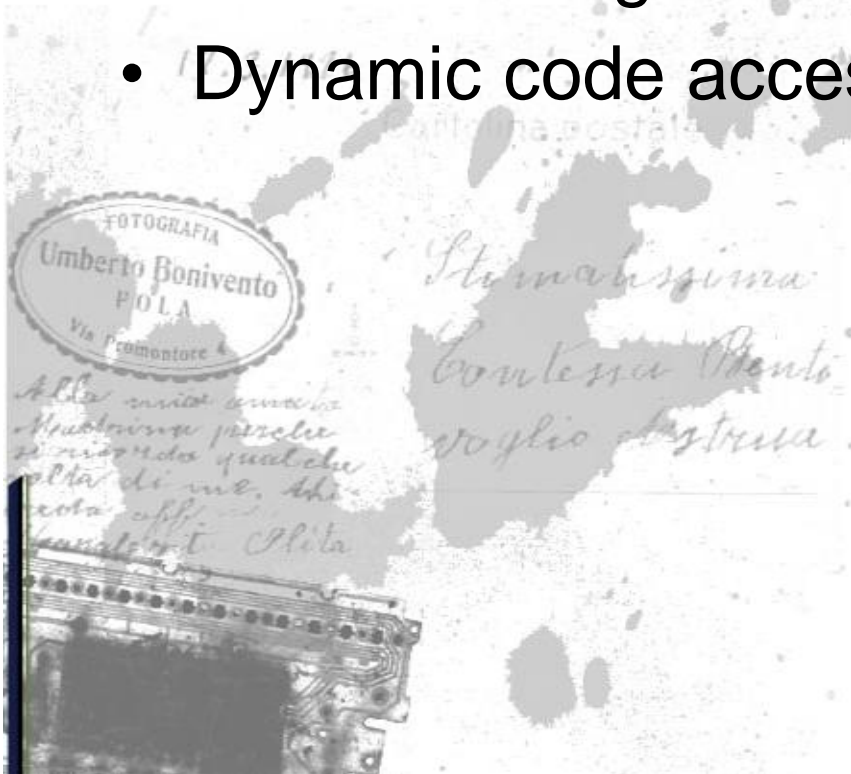
- Code-Access Security
- Role-Based Security
- Isolated Storage
- Cryptographic Services

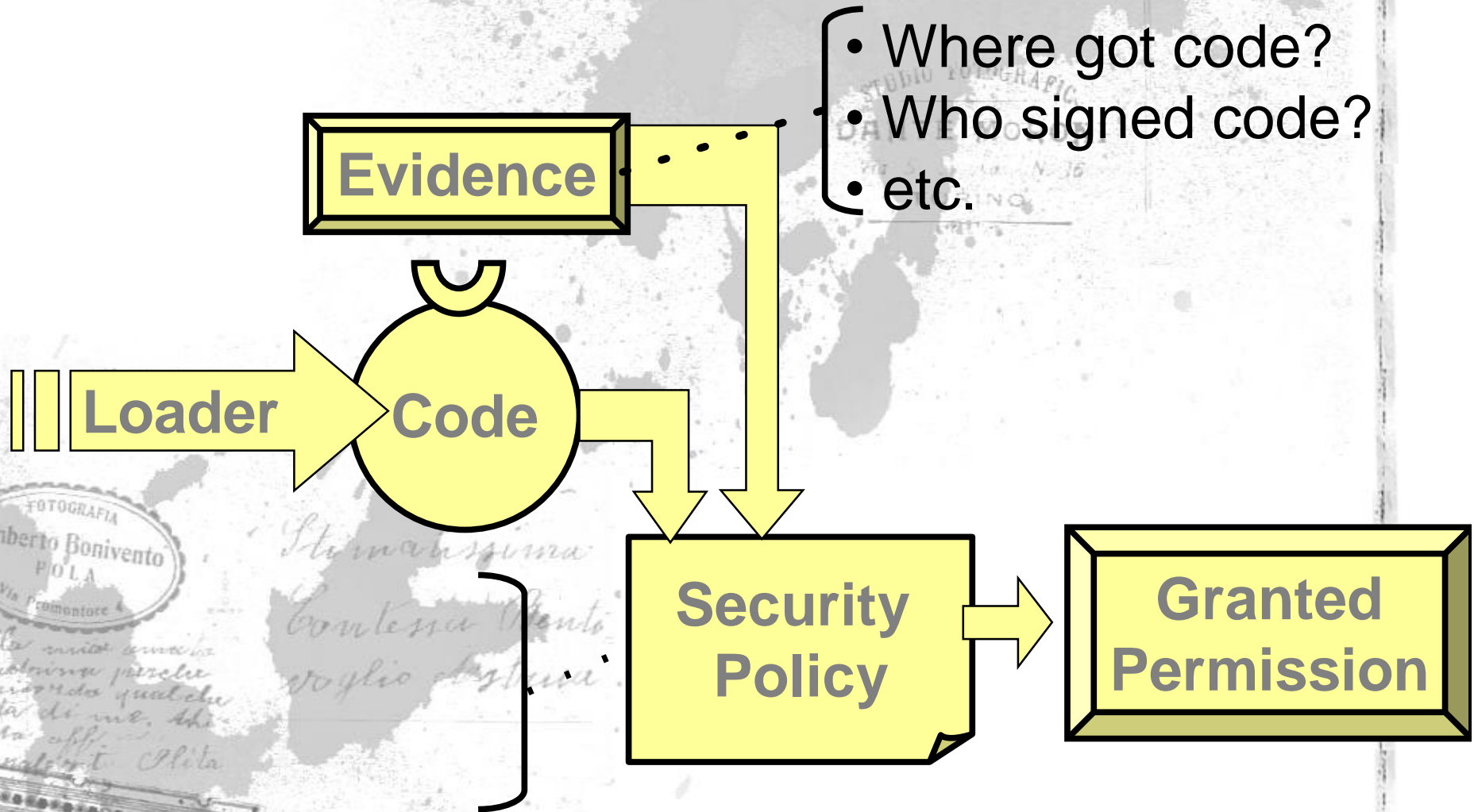


- Security Policy
- Code Evidence
- Permission



- Define permission
- Manage security policy
- Granted permission for Code
- Permit code grant caller certain permission
- Dynamic code access security (Stackwalking)





- Policy level
 - Enterprise
 - Machine
 - User
 - Application Domain
- Use caspol.exe or mscorcfg.msc
- Intersecting policy level grant sets

- Standard .NET evidence types
 - Zone
 - Site
 - Url
 - Publisher
 - StrongName
 - ...
- Custom Evidence

- .NET code-access permission classes

- CodeAccessPermission
- DBDataPermission
- PrintingPermission
- DnsPermission
- WebPermission
- EnvironmentPermission
- FileIOPermission
- RegistryPermission
- UIPermission

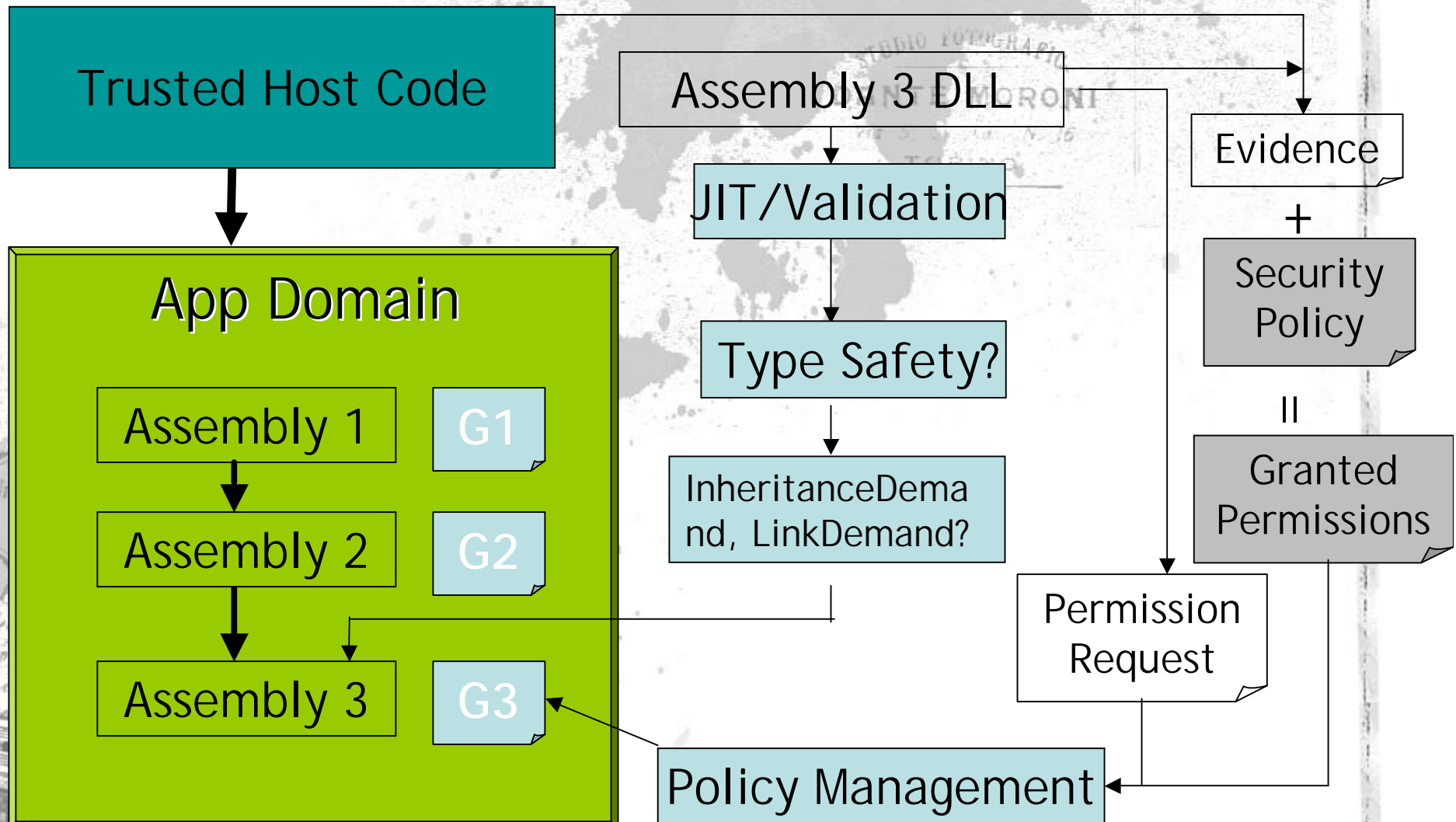
- Custom Code-Access Permissions

3.1.7. Enforcing Code-Access Security

permissions granted to each assembly

A, B			
A, C			
A, C	Deny A		
A, B, C		Assert B	
A, B, C			
A, B, C			
	Demand A	Demand B	Demand C
	denied	granted	denied





- .NET Role-Based Security Classes

- GenericPrincipal
- GenericIdentity
- WindowsPrincipal
- WindowsIdentity
- ...
- PrincipalPermission

- **IPrincipal interface**

```
namespace System.Security.Principal {  
    public interface IPrincipal {  
        Identity Identity { get; }  
        bool IsInRole( String role );  
    }  
}
```

- **IIdentity interface**

```
namespace System.Security.Principal {  
    public interface IIdentity {  
        String AuthenticationType { get; }  
        bool IsAuthenticated { get; }  
        String Name { get; }  
    }  
}
```

```
[ PrincipalPermissionAttribute( SecurityAction.Demand,  
                               Name = "MyUser",  
                               Role = "User" ) ]
```

```
public static void PrivateInfo()  
{
```

```
    // print secure data
```

```
    Console.WriteLine( "\n\nYou have access to the private  
data!" );
```

```
}
```

```
String id1 = "Bob";  
String role1 = "Manager";  
PrincipalPermission PrincipalPerm1 =  
    new PrincipalPermission(id1, role1);  
String id2 = "Louise";  
String role2 = "Supervisor";  
PrincipalPermission PrincipalPerm2 =  
    new PrincipalPermission(id2, role2);  
(PrincipalPerm1.Union(PrincipalPerm2)).Demand();
```

Thanks !

