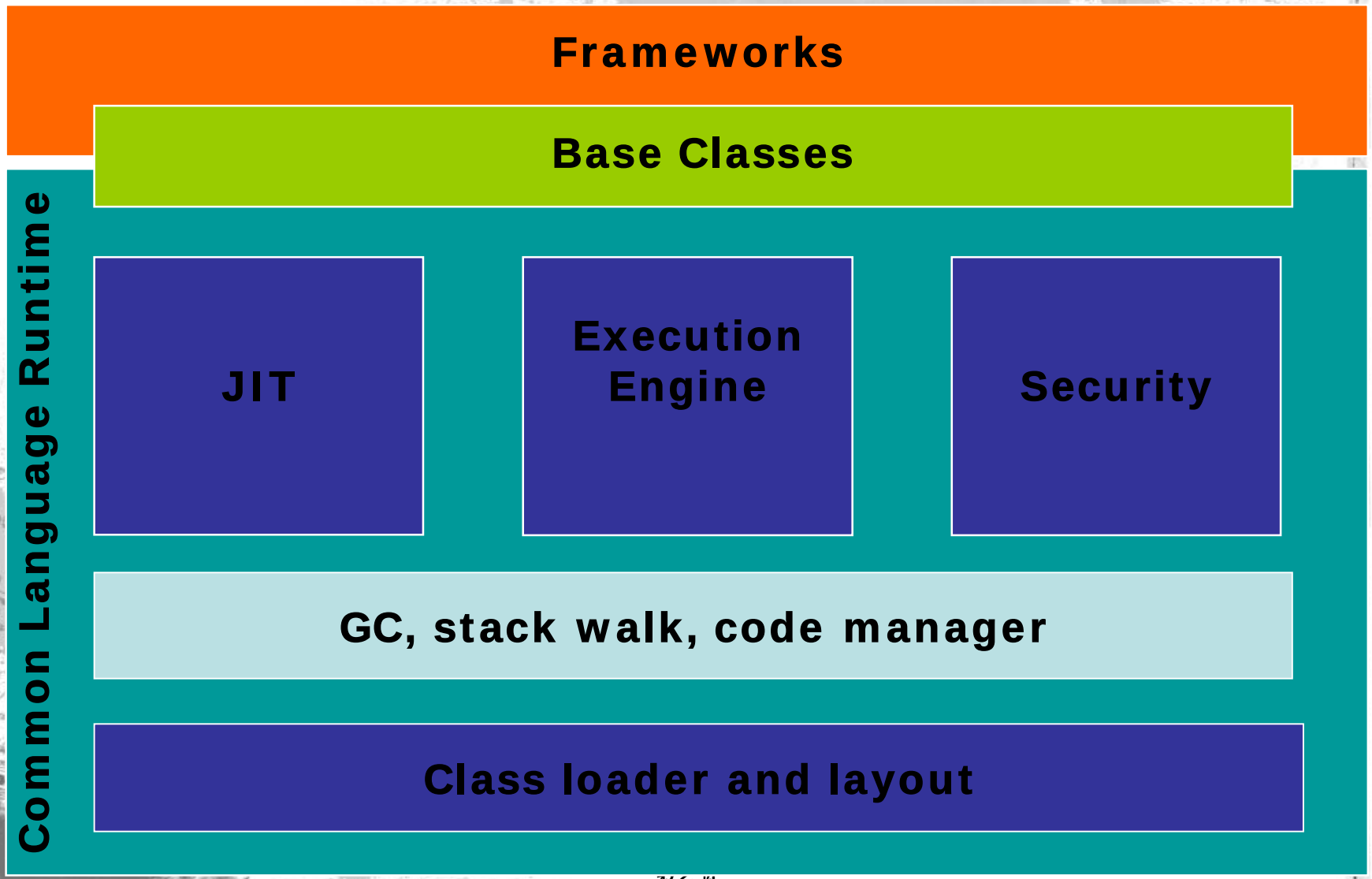
.NET Security

Flier@nsfocus.com

X'con 2004

.NET Architecture



X'con 2004

Agenda

- Static Structure
- Dynamic Executing
- Security Framework

1. Static Structure

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

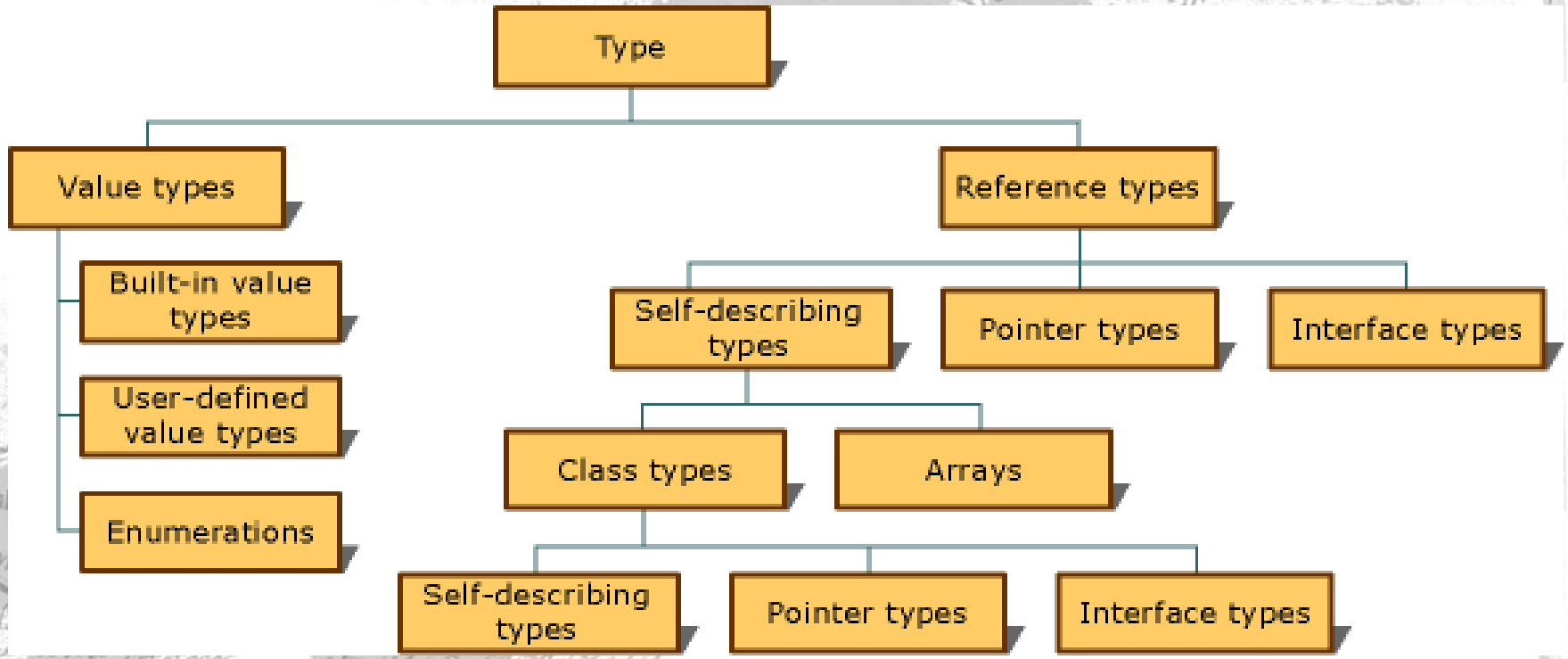
1.1.CTS - Common Type System

- 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

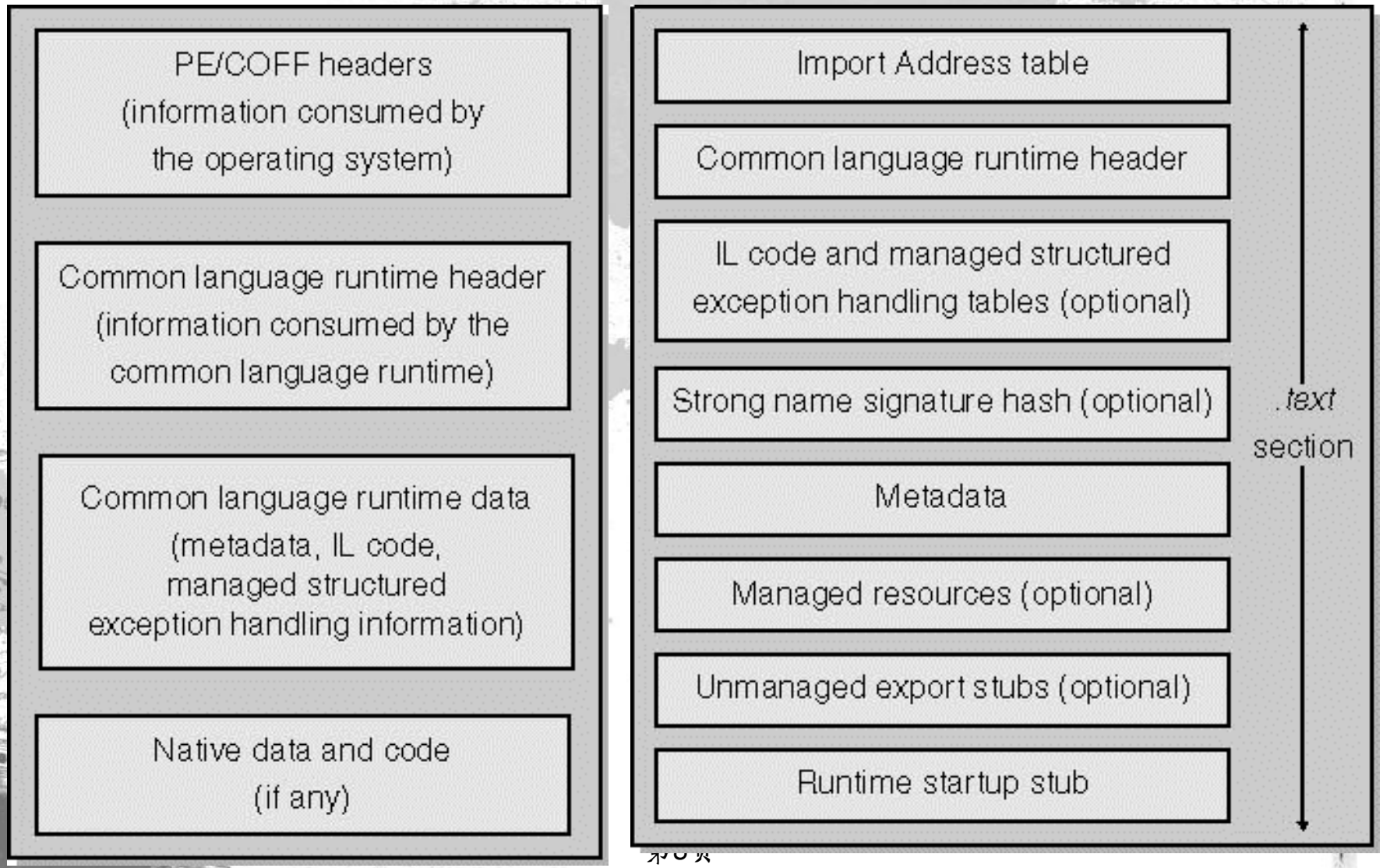
1.1.2. Naming Hierarchy

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

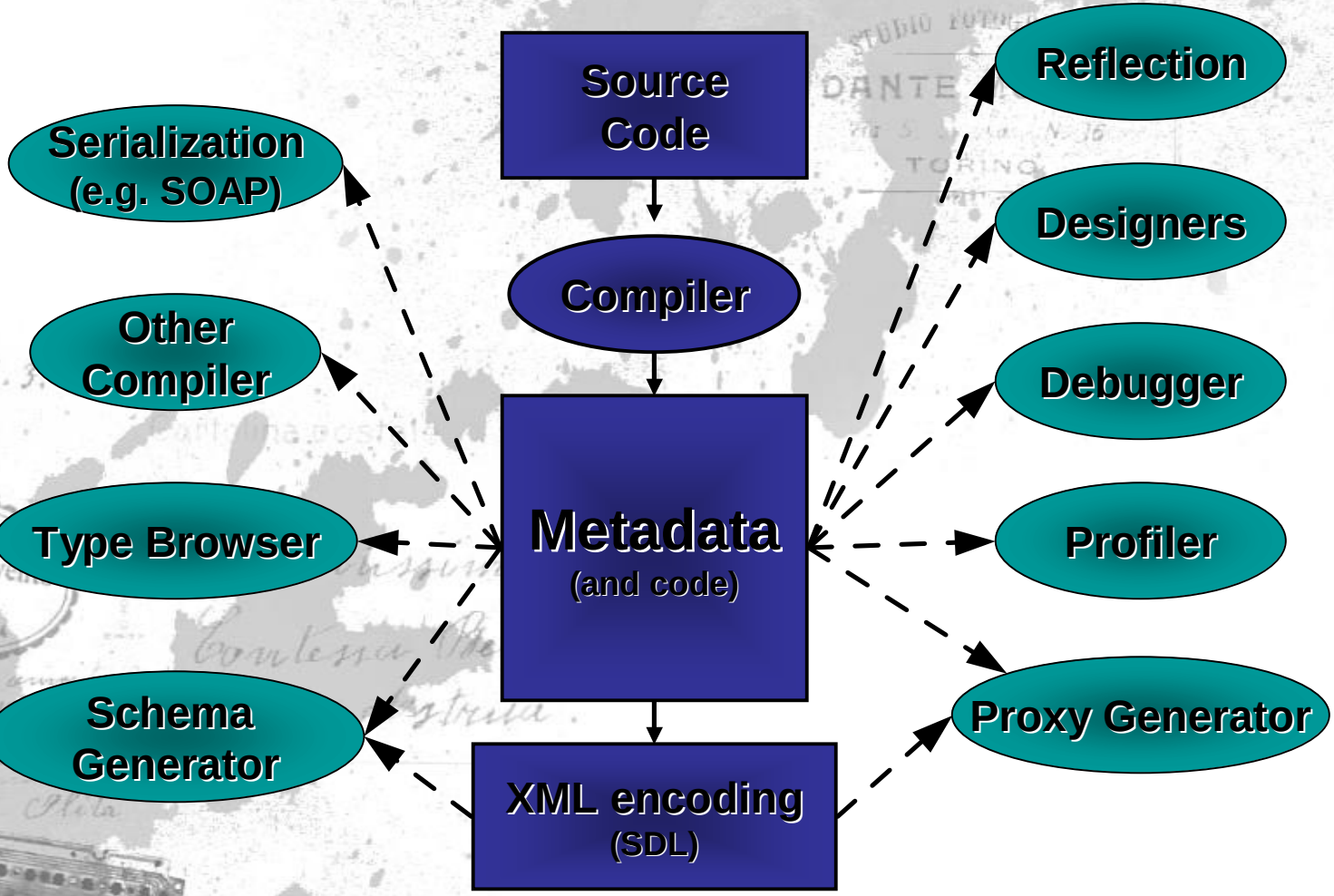
1.1.3.Type Categories



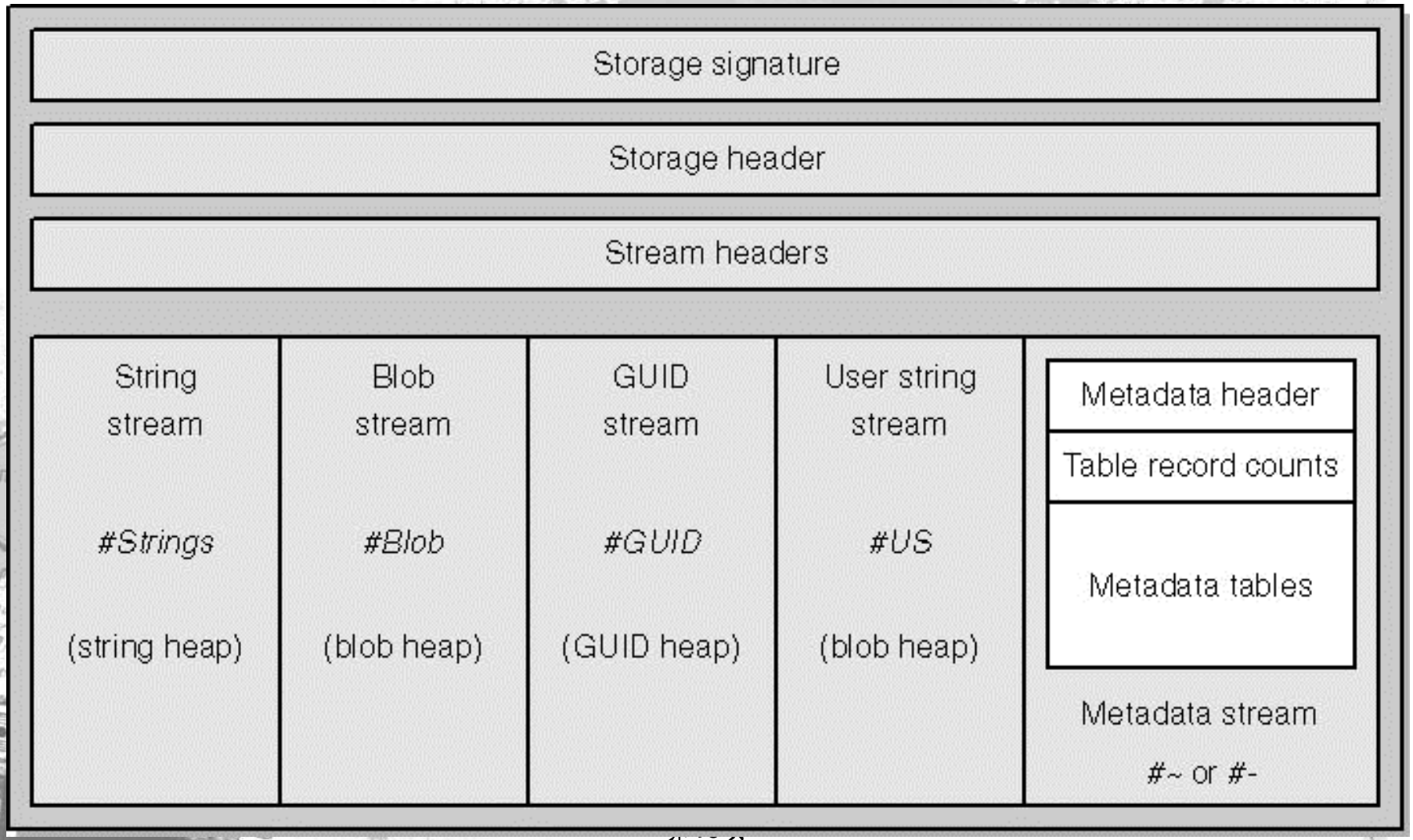
1.2. The Structure of a Managed Executable File



1.2.2. What Is Metadata?

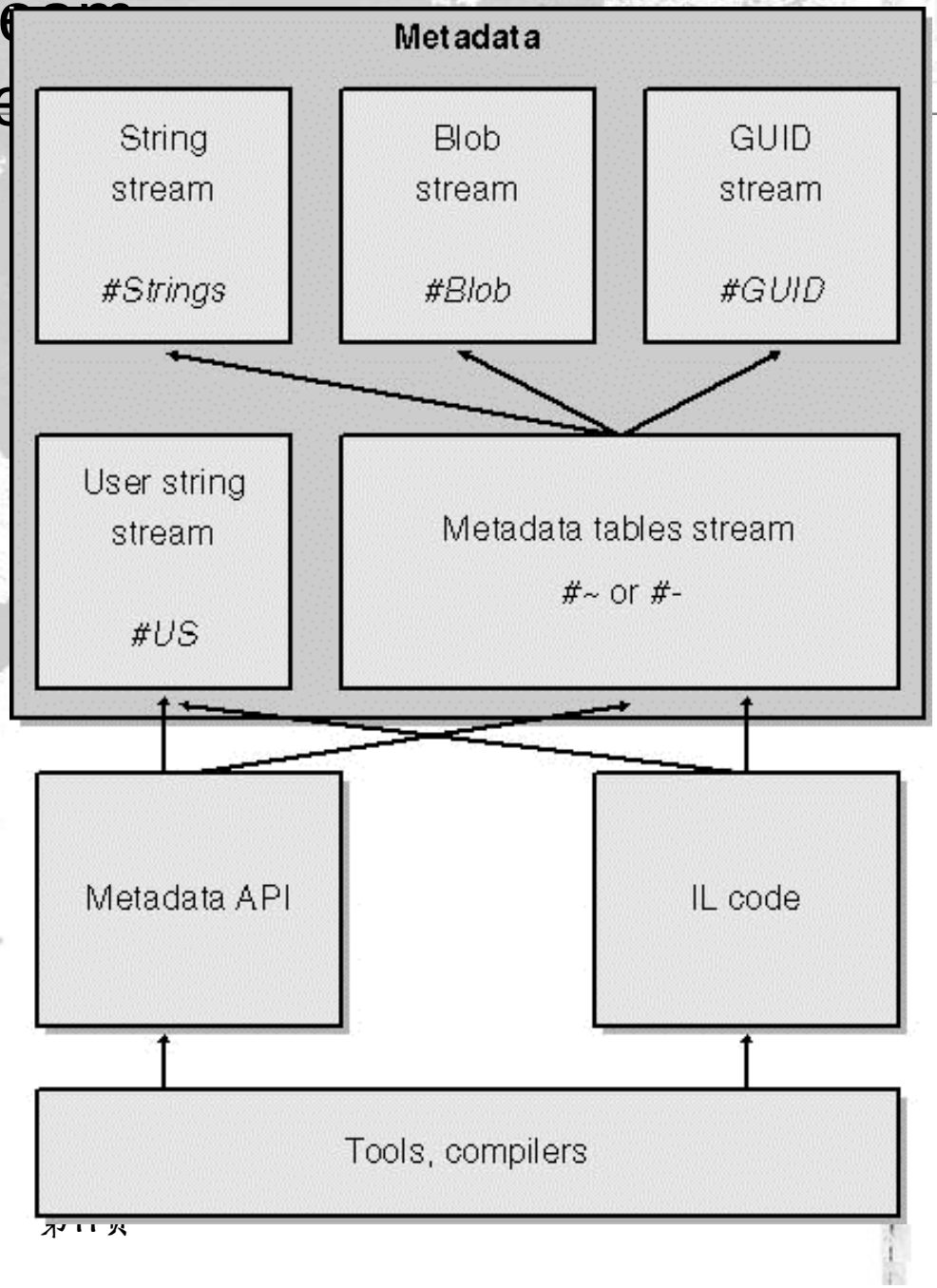


1.2.3Metadata Structure



1.2.4. Stre… refe…

- #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

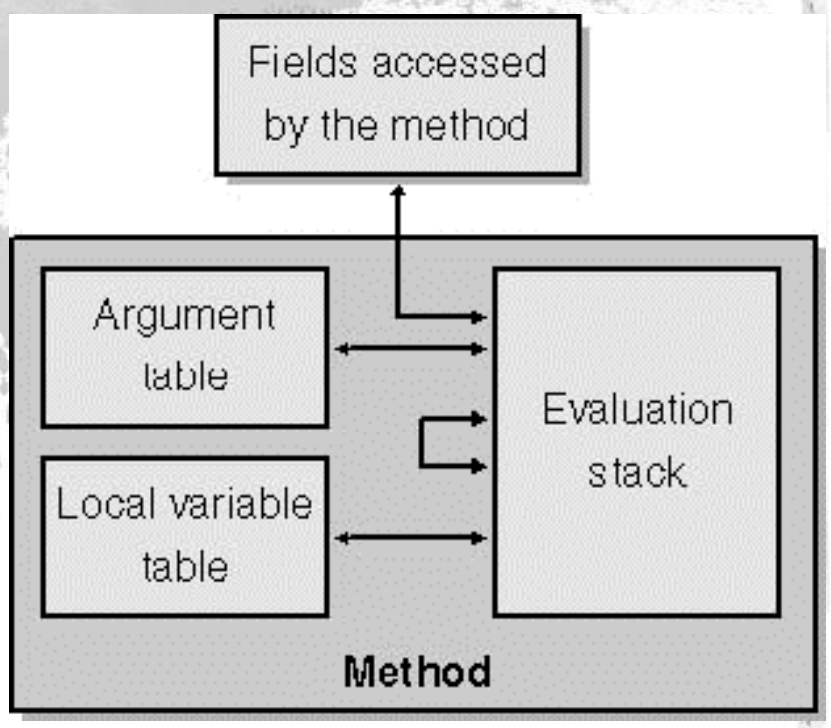


1.2.5.Metadata Tables

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

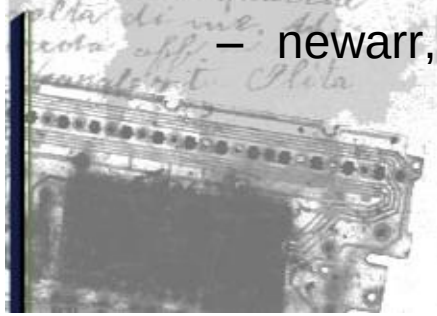
1.3. IL Instructions

- 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

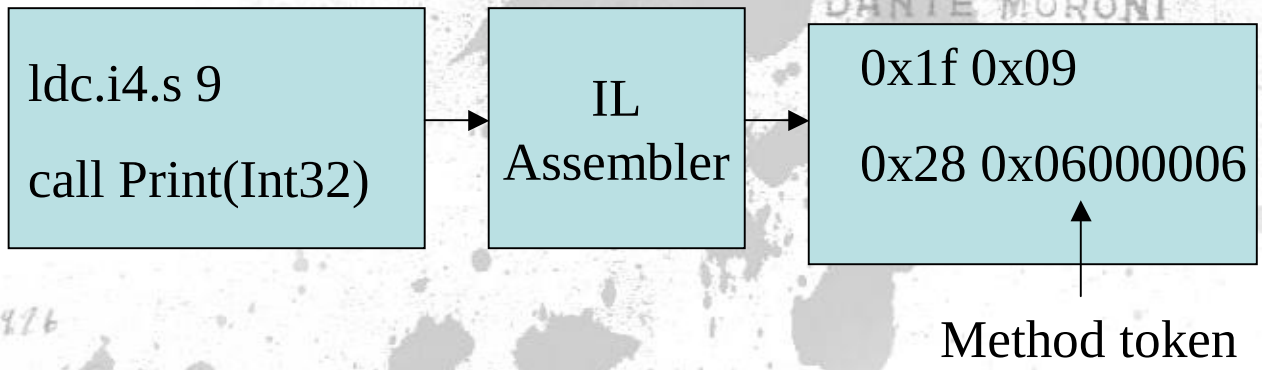


1.3.2.IL Instruction Categories

- 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



1.3.3.Uses "Token"



Token

1.3.4.Method Call Stack

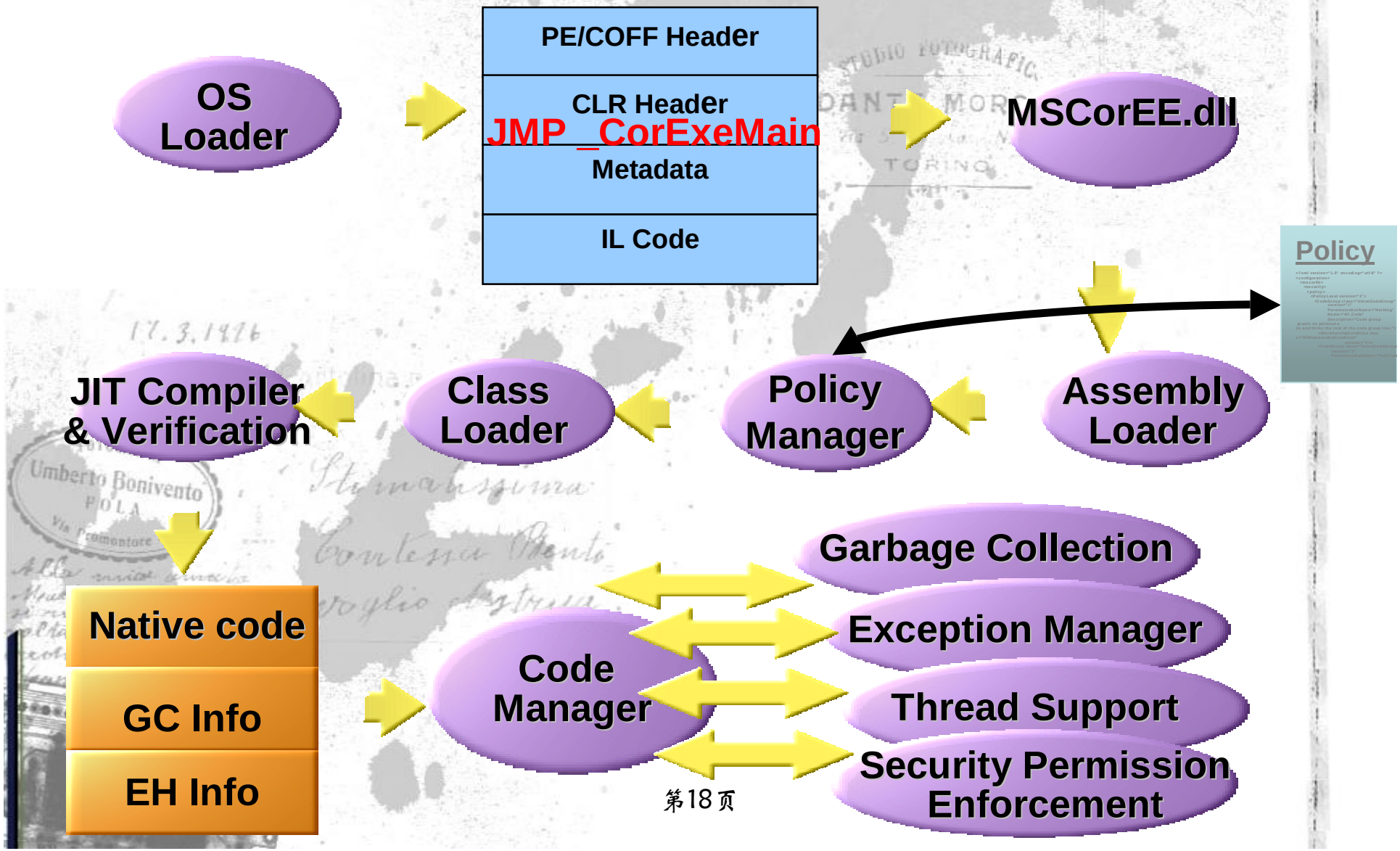
```
ClassType a;
a.func(1, 2)
```

```
ldc.i4.1
ldc.i4.2
call ClassType::func(Int32, Int32)
```

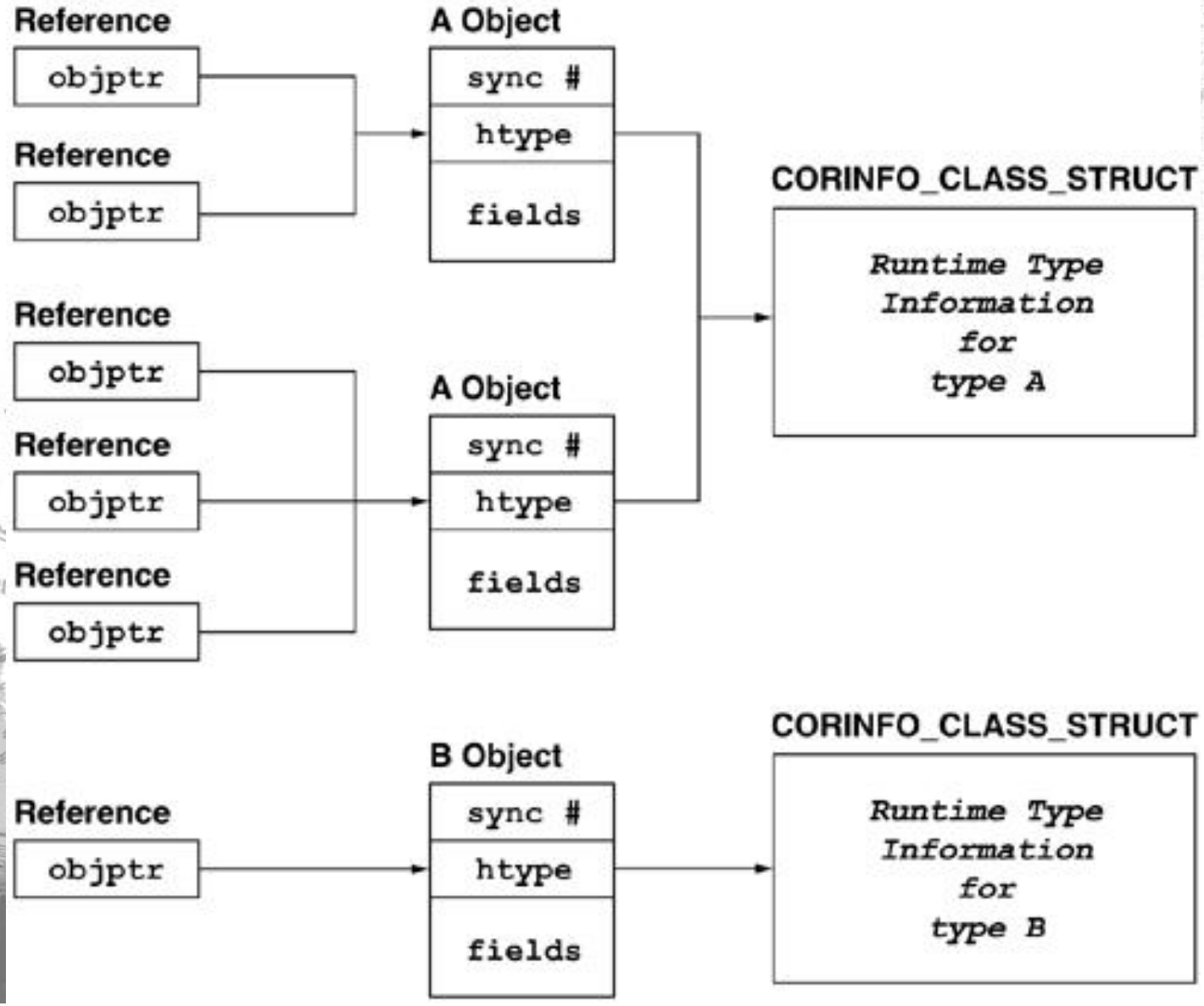
2.Dynamic Executing

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

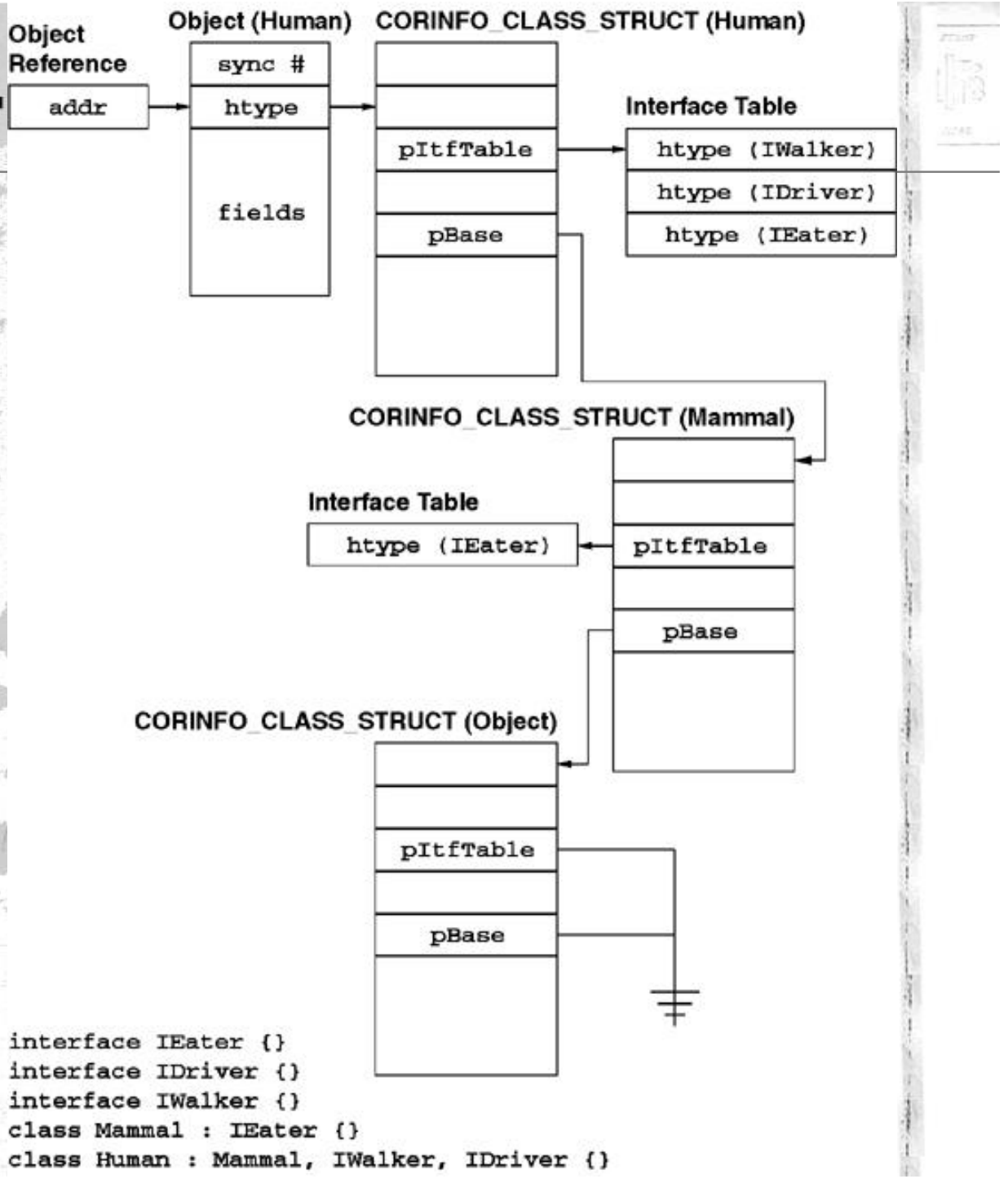
2.1.CLR Loader



2.2. Type and object memory layout



2.2.2. Type Hierarchy and Runtime Type Information

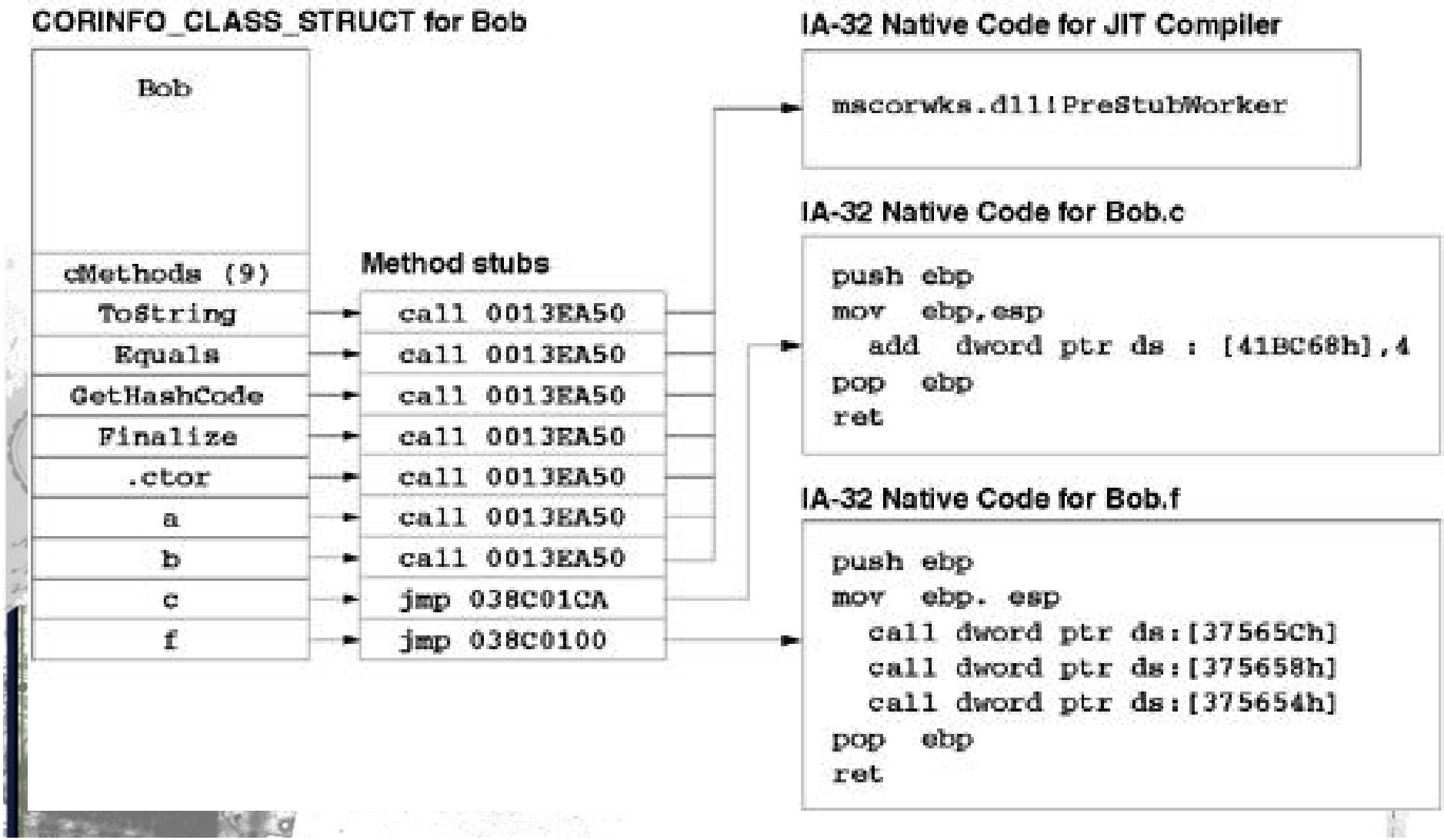



```
interface IEater {}
interface IDriver {}
interface IWalker {}
class Mammal : IEater {}
class Human : Mammal, IWalker, IDriver {}
```

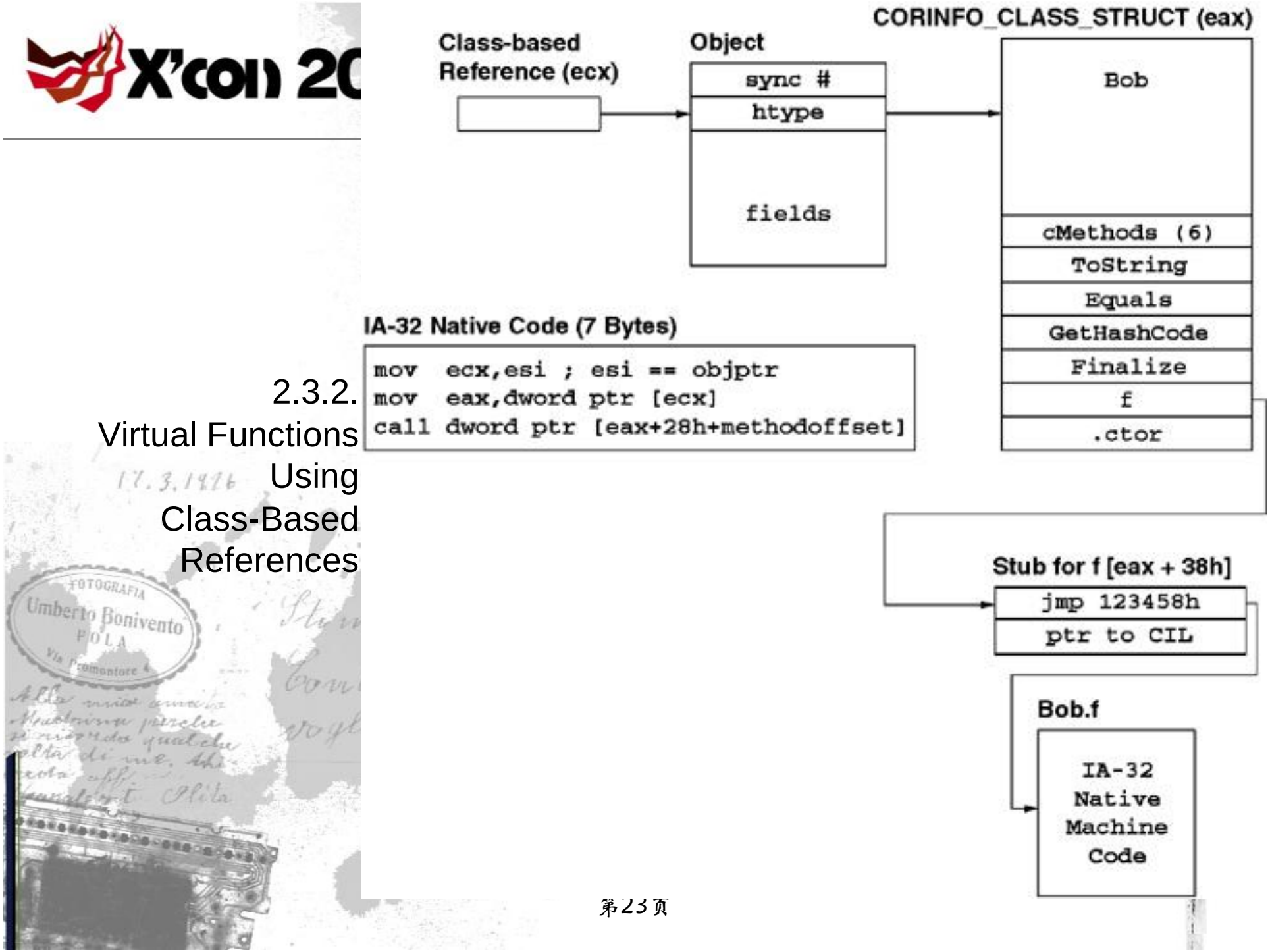
2.3.JIT Compilation and Method Invocation

- 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

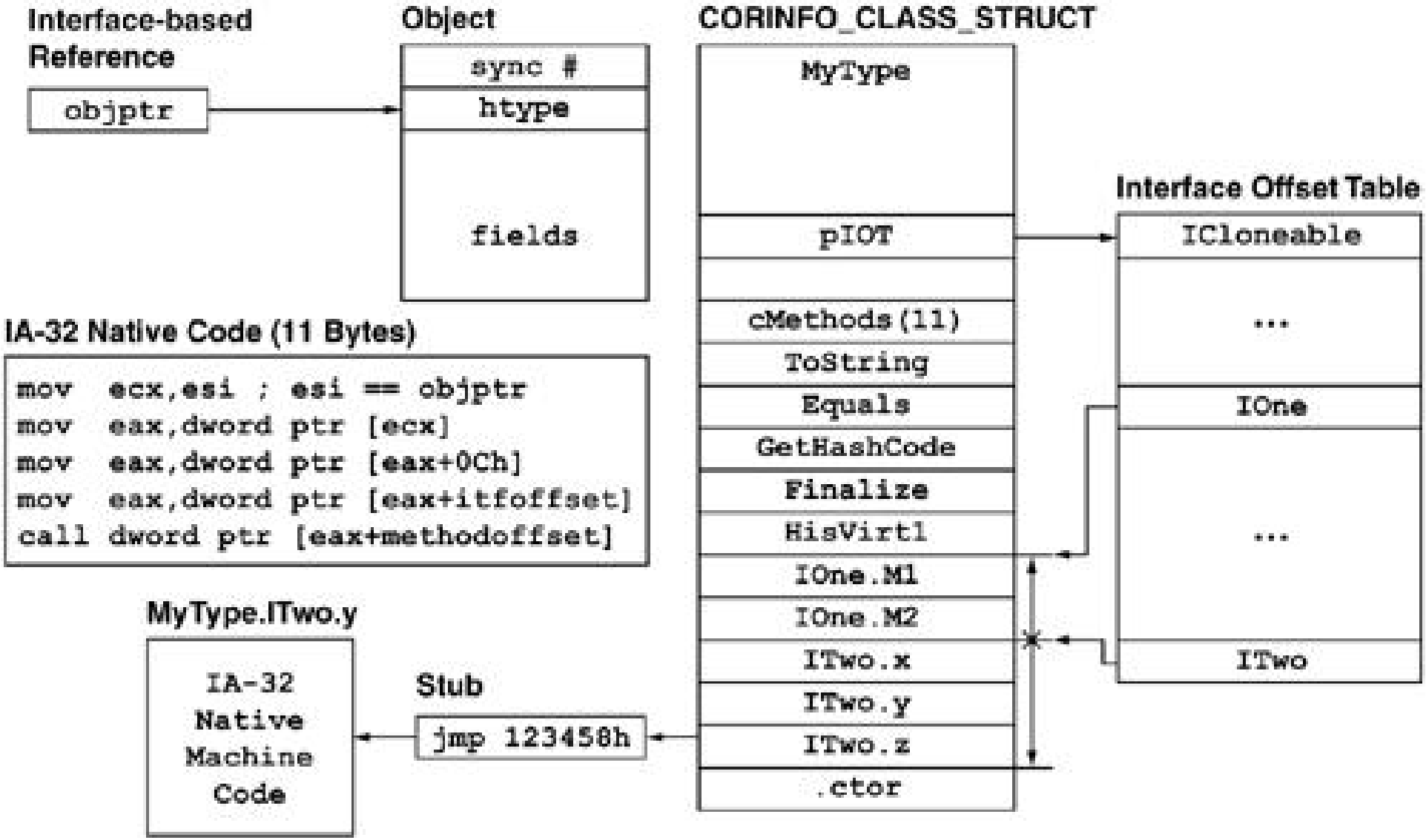
2.3.1.JIT Compilation and Method Table



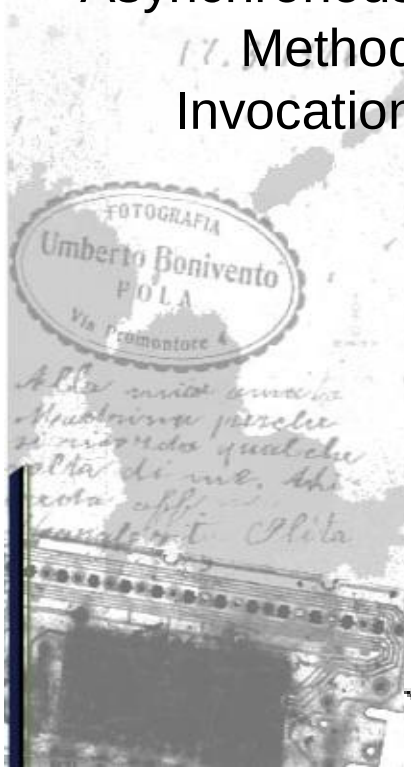
2.3.2. Virtual Functions Using Class-Based References

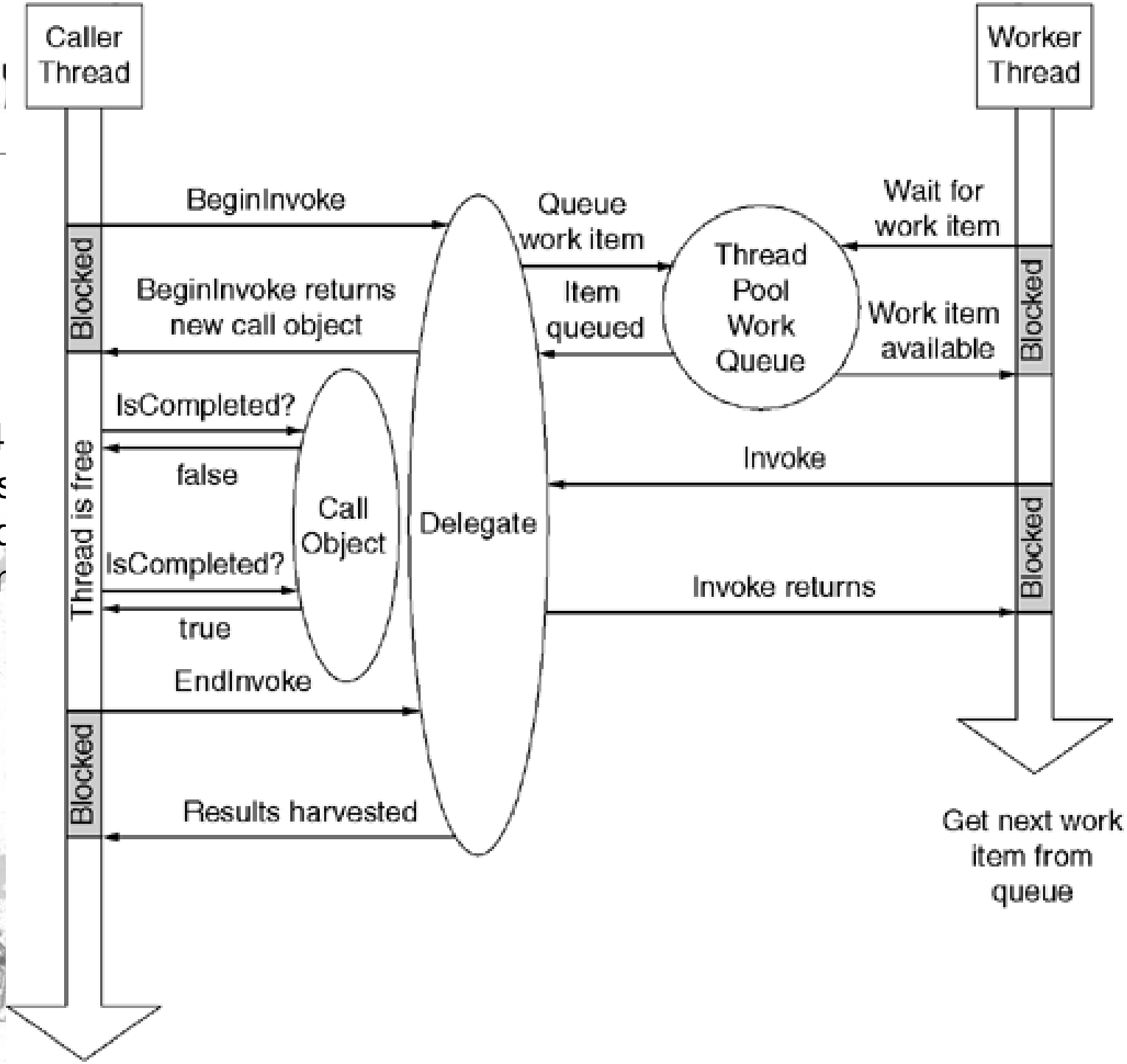


2.3.3.Virtual Functions Using Interface-Based References



2.3.4
Asynchronous
Method
Invocation



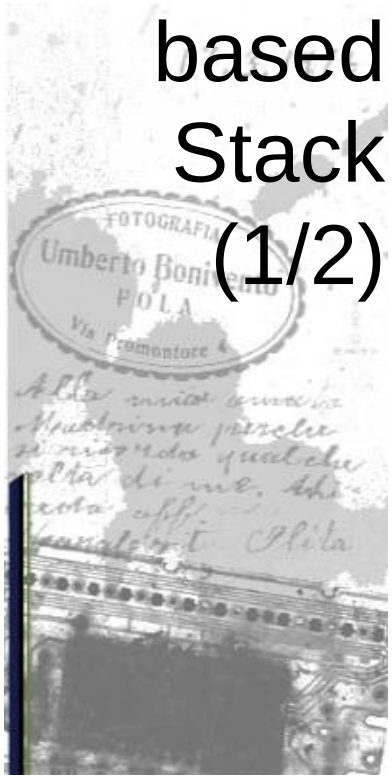


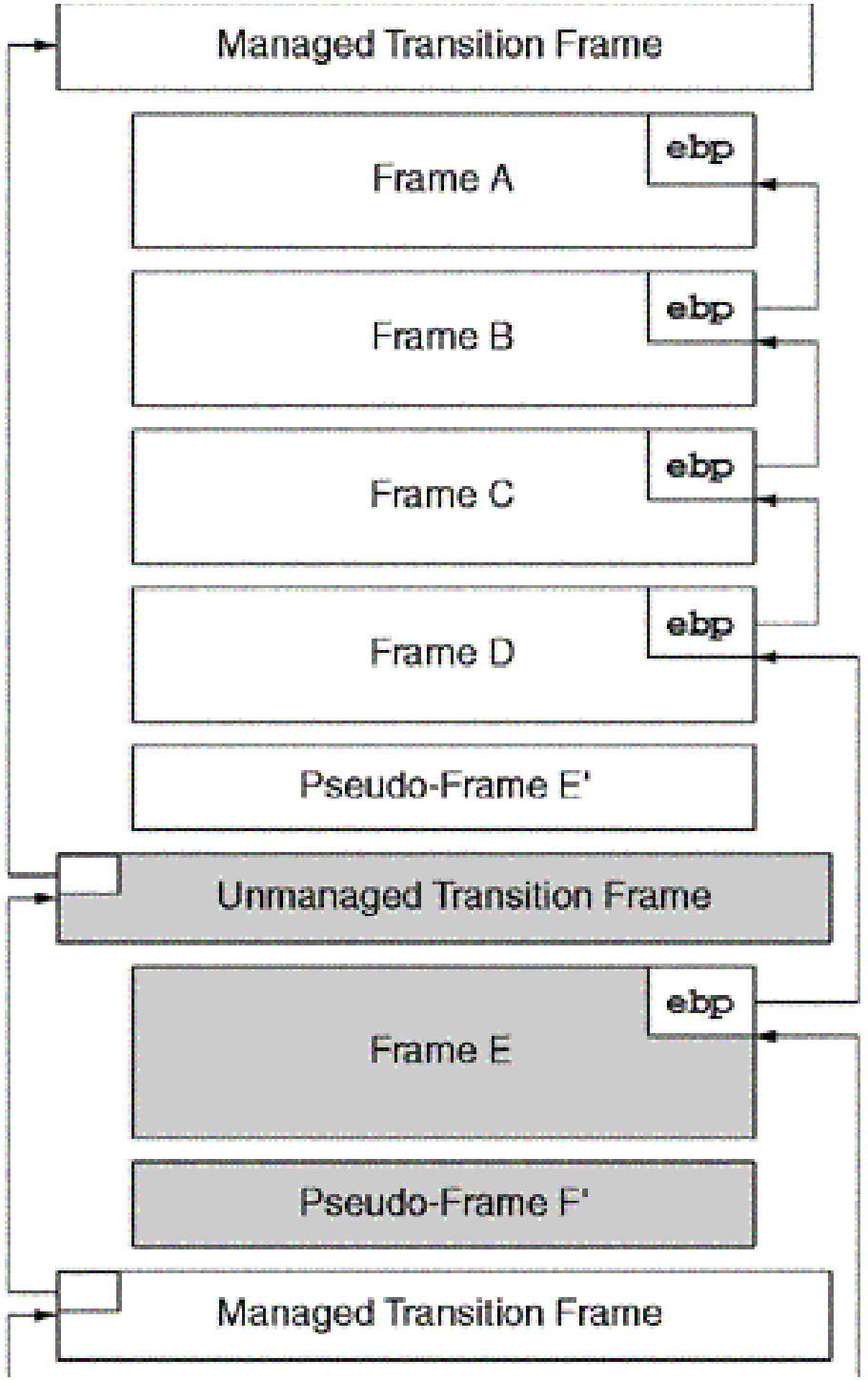
2.4. Stack, Frame and Stack Walk

- 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

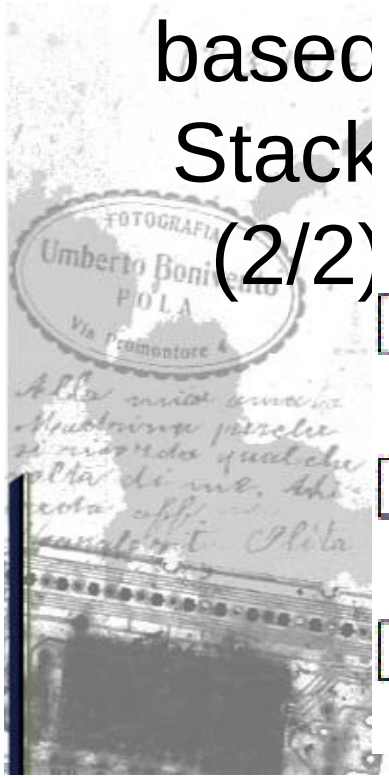
2.4.1. Frame-based Stack (1/2)

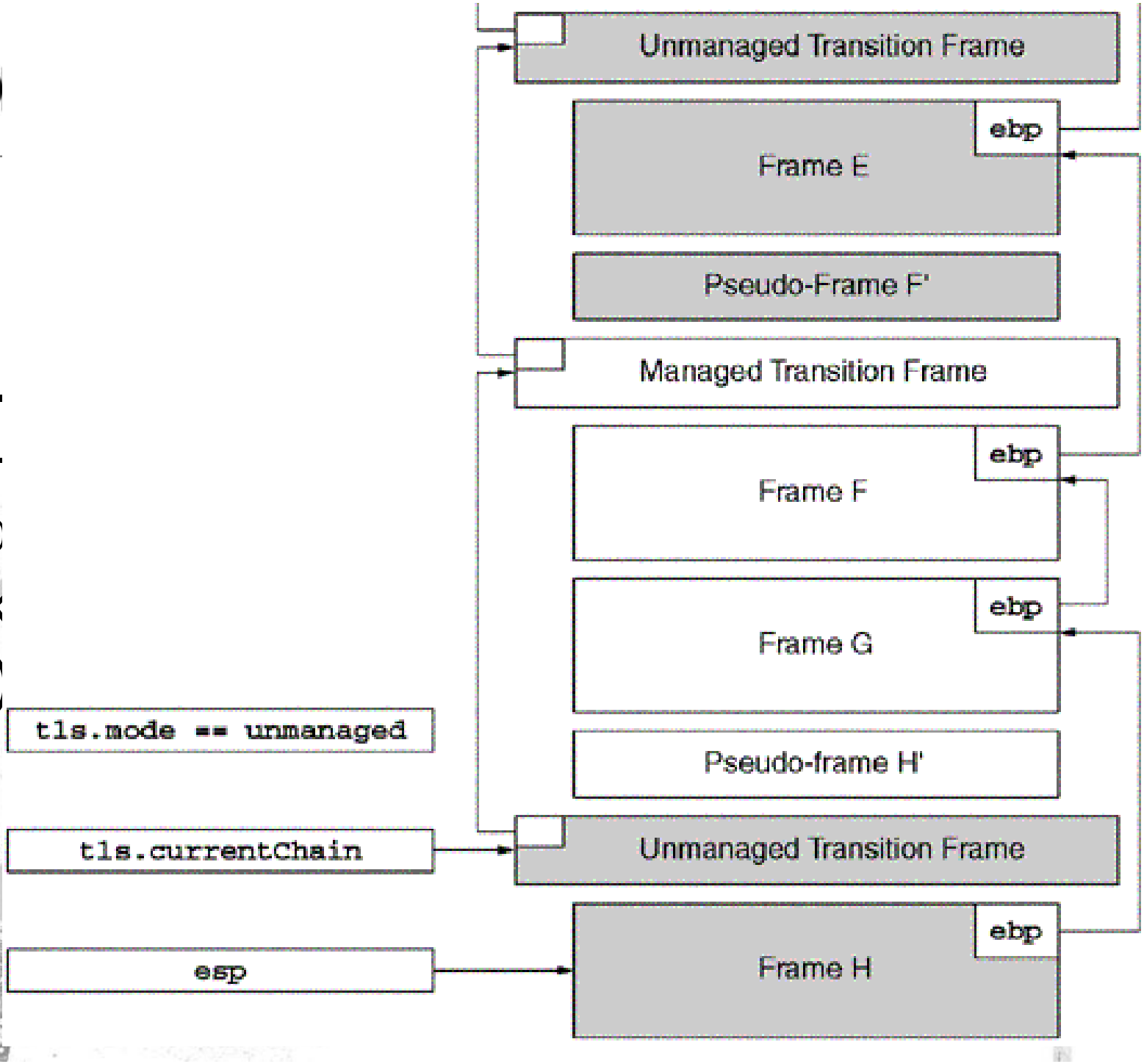
```
#pragma unmanaged
void H(){}
#pragma managed
void G() { H(); }
void F() { G(); }
#pragma unmanaged
void E() { F(); }
#pragma managed
void D() { E(); }
void C() { D(); }
void B() { C(); }
void A() { B(); }
```



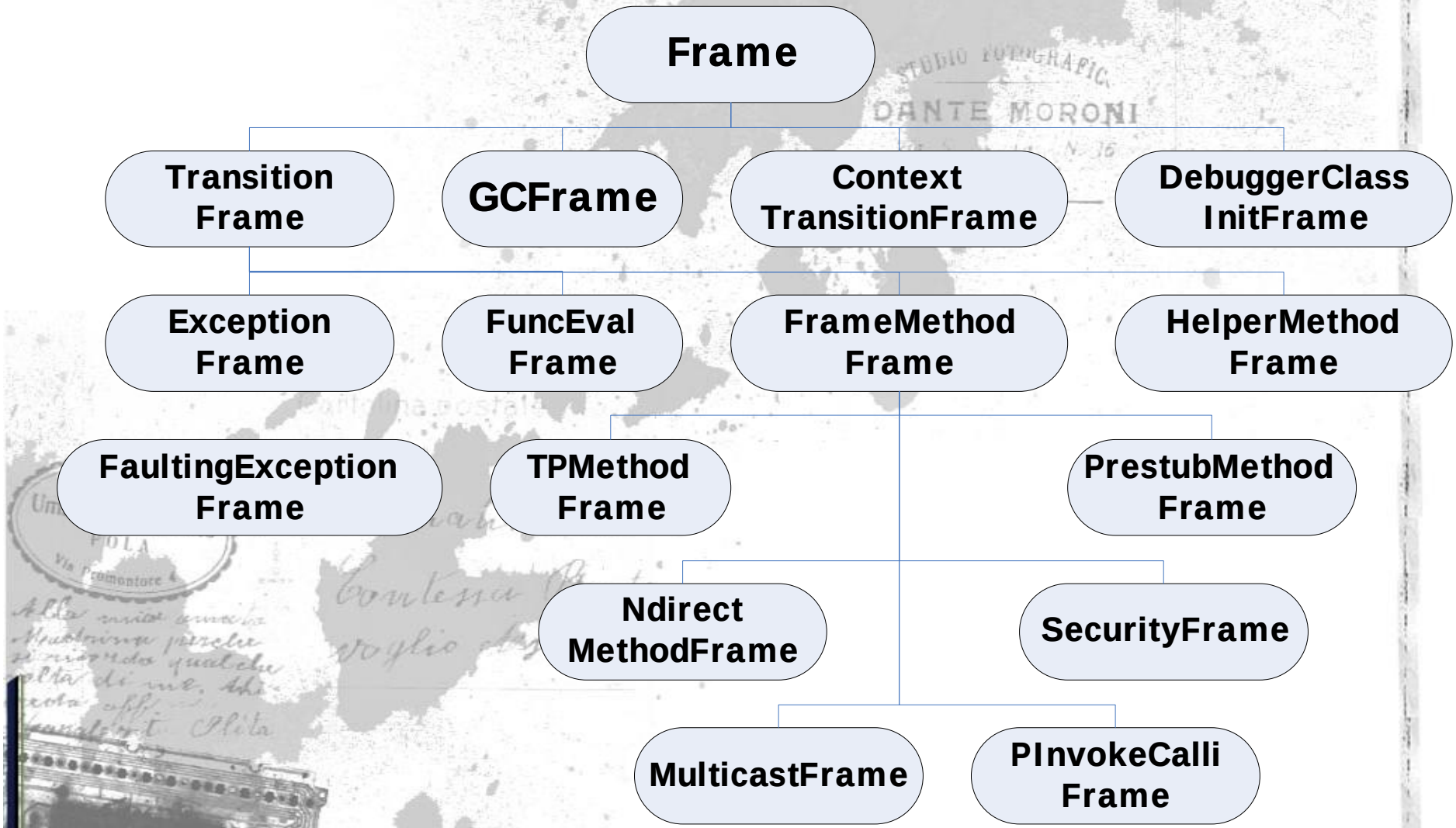


2.4.1. Frame-based Stack (2/2)



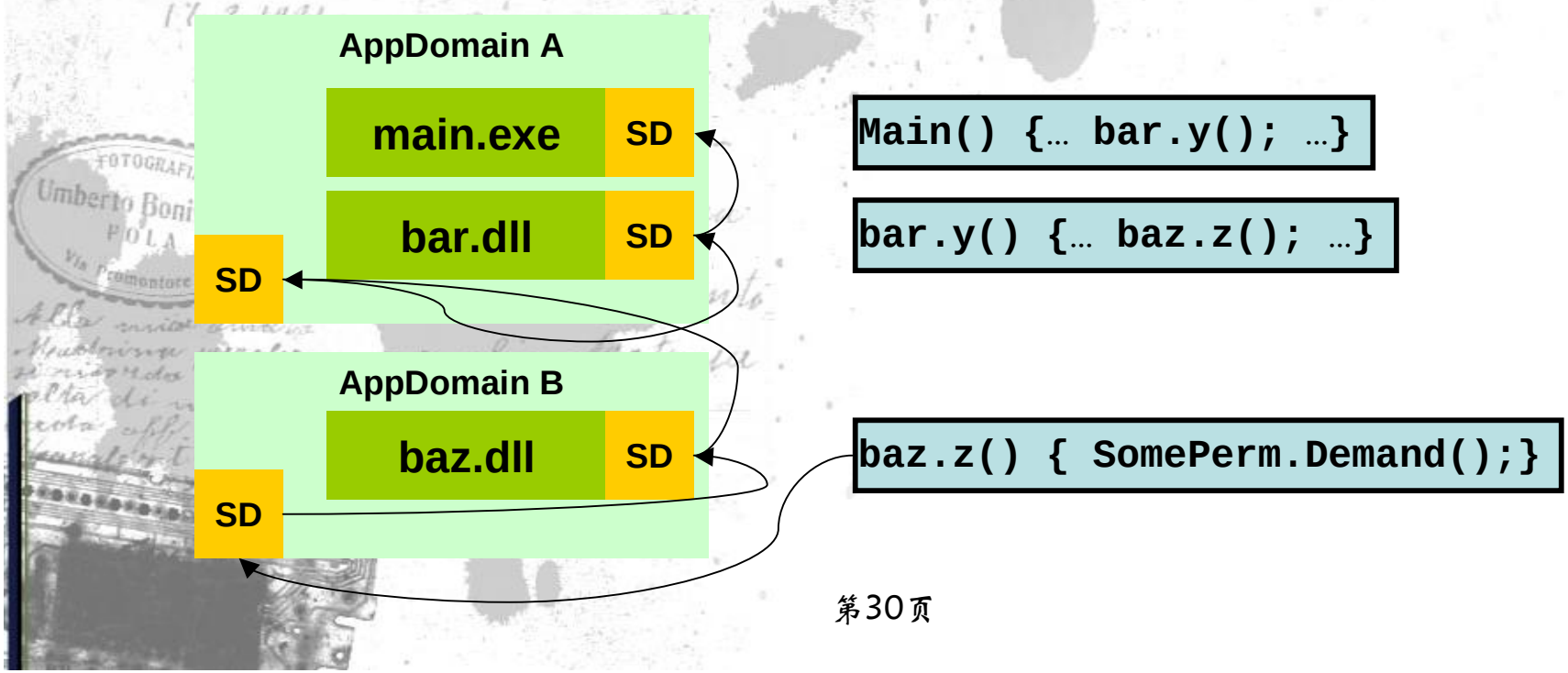


2.4.2. Common Frame Type Hierarchy

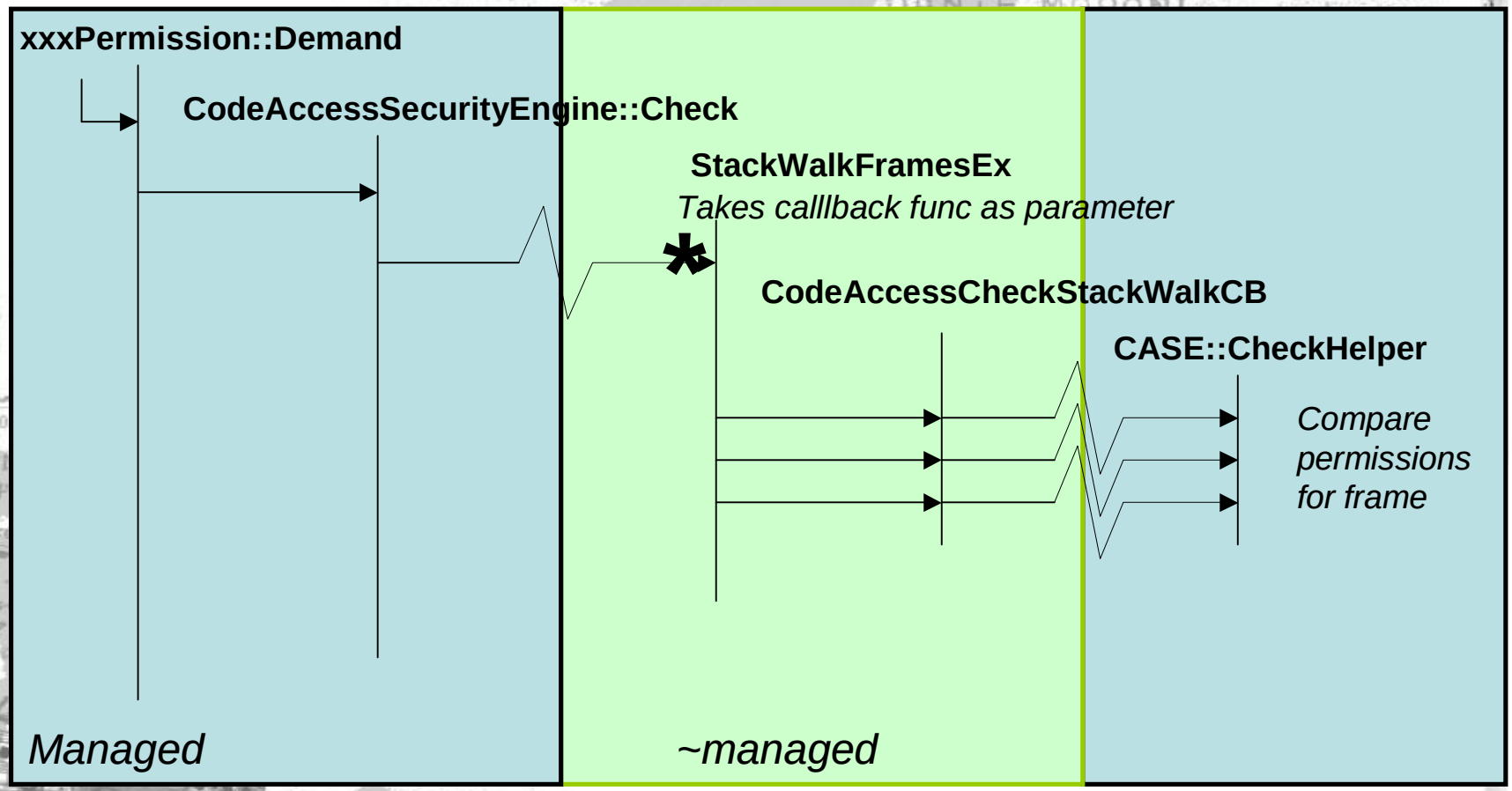


2.4.3. Stack Walk Mechanics

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



2.4.4. Walking the stack



3.Security Framework

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

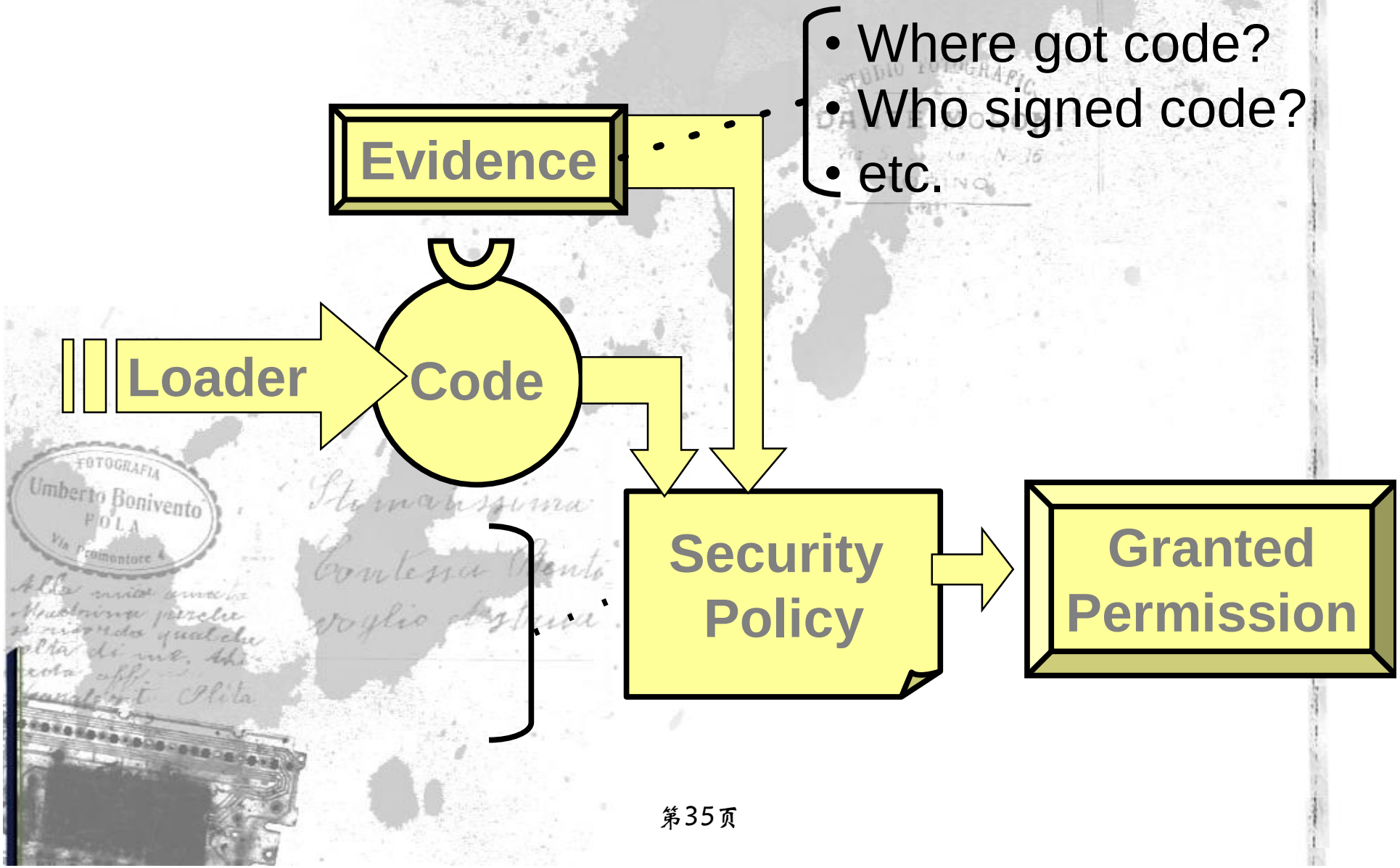
3.1. Code-Access Security

- Security Policy
- Code Evidence
- Permission

3.1.2.CAS Mechanics

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

3.1.3. Granting



3.1.4.Security Policy

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

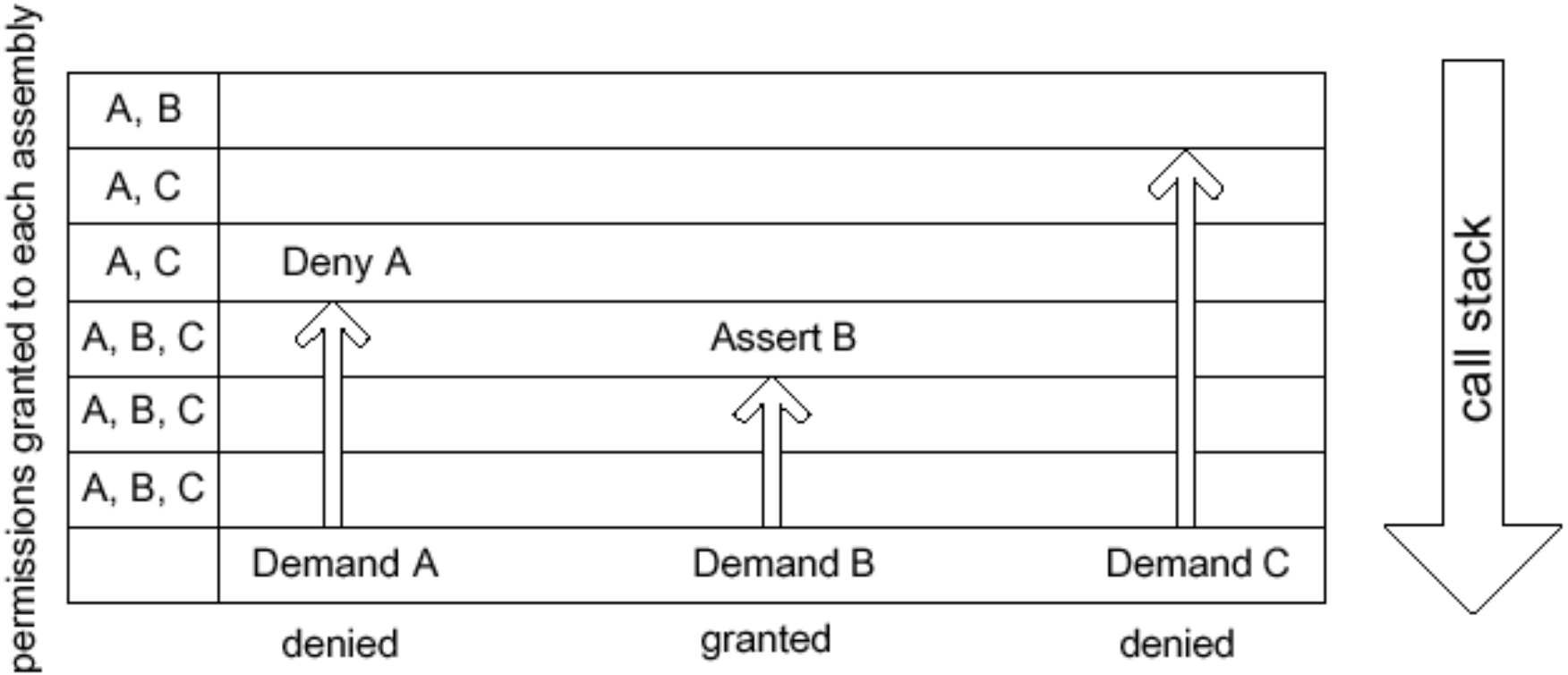
3.1.5.Evidence

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

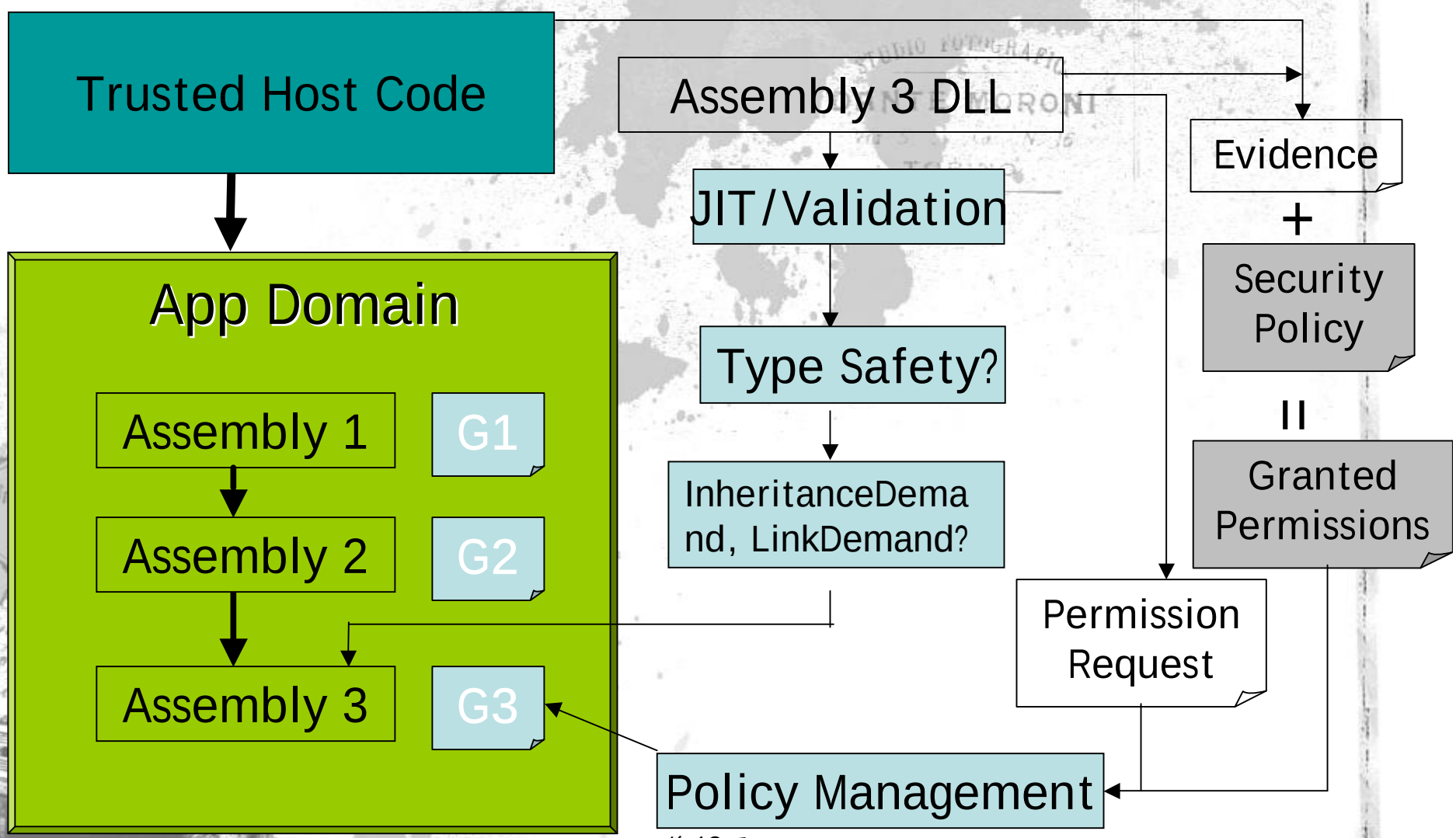
3.1.6.Permission

- .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



3.1.8. Granted Permissions



3.2. Role-Based Security

- .NET Role-Based Security Classes
  - GenericPrincipal
  - GenericIdentity
  - WindowsPrincipal
  - WindowsIdentity
  - …
  - PrincipalPermission

3.2.1.Interfaces

- IPrincipal interface

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

- IIdentity interface

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

3.2.2.Static Demand

```
[ PrincipalPermissionAttribute( SecurityAction.Demand,
                                Name = “MyUser”,
                                Role = “User” ) ]
public static void PrivateInfo()
{
        // print secure data
        Console.WriteLine( “\n\nYou have access to the private
data!” );
}
```

3.2.3.Dynamic Demand

```
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();
```

X'con 2004

Thanks !

X'CON 2003