# Assemblies

code packaging, loading & versioning

# Overview

- **Managed assemblies**
  - Types & scoping
  - Assembly naming
  - Assembly resolution
    - private path probing
    - strong named resolution
  - Assembly caches
    - global assembly cache (GAC)
    - native code cache (ngen cache)

# Assemblies

- **Managed code is distributed in assemblies**
  - act as a scoping boundary for types
  - assigned location-independent names
  - may be independently versioned

# Assemblies & Types

- **Types are scoped by their containing assembly**
  - access control (`public` versus `internal`)
  - type names are assembly-qualified
    - same type name in different assemblies == different types

```
namespace Acme {
  public class Calc {
    ...
  }
}
```

**foo.dll**

```
namespace Acme {
  public class Calc {
    ...
  }
}
```

**bar.dll**

**Type name:** `"Acme.Calc, foo"`

**Type name:** `"Acme.Calc, bar"`

# Assembly Resolution & Loading

- **During managed execution, assembly loading involves**
  - version mapping (sometimes)
  - name resolution (a name-to-CODEBASE mapping operation)
  - validation (sometimes)
  - loading “resolved” version into memory

# Assembly Names

- **Name format drives sophistication of the resolution algorithm**
  - signed assemblies are referred to as “strongly named”
  - strongly named assemblies have a non-null PublicKeyToken

**asmname.exe**

```
using System;

class Program
{
  static void Main()
  {
    Console.WriteLine(typeof(Program).Assembly.FullName);
    Console.WriteLine(typeof(string).Assembly.FullName);
  }
}
```

VS.NET 2005 Shell

```
c:\dev\asmname>asmname.exe
asmname, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

c:\dev\asmname>_
```

# Simple Assembly Resolution

- **A simple set of rules govern loading of unsigned assemblies**
  - hosting application directory (APPBASE)
  - subdirectory named after assembly to load
    - eg: CLR will look for acme.dll in APPBASE\acme
  - additional *subdirectory(ies)* named in application's config file
    - eg: hello.exe's config file is named hello.exe.config
  - called private path probing
    - often referred to as "xcopy deployment"

```
<?xml version="1.0" ?>

<configuration xmlns:asm="urn:schemas-microsoft-com:asm.v1">
  <runtime>
    <asm:assemblyBinding>
      <asm:probing privatePath="subdir1;subdir2" />
    </asm:assemblyBinding>
  </runtime>
</configuration>
```

# Strong Names

- **Fully-specified assembly names have 4 parts**
  - friendly name
  - version
  - culture (language/region)
  - key token (uniquely identifies the publisher)
    - involves public/private key pair technology

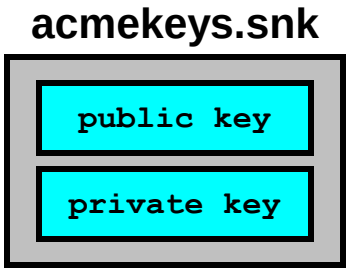
```
"calc, Version=0.0.0.0, Culture=neutral, PublicKeyToken=b7aa5c561934e089"
```

*Above format referred to as "display name"*

# Key Files

- **SN.EXE creates/manages key files**
  - -k command line option creates a new pair of keys

```
c:\> sn -k acmekeys.snk
```



# Strongly Naming an Assembly

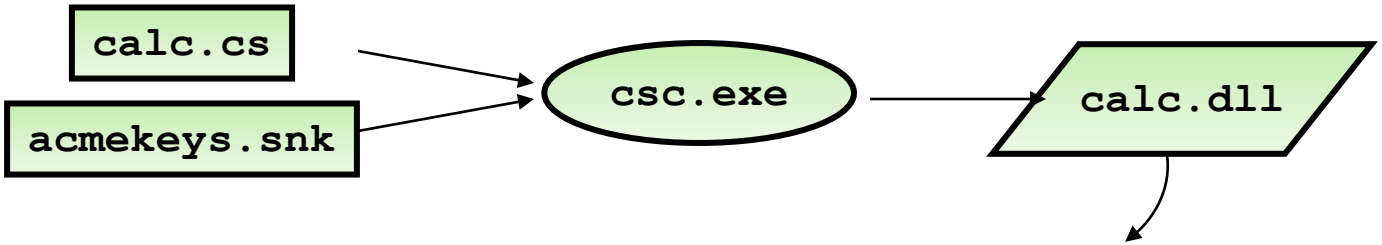
- **Assembly name is specified using attributes**
  - or VS.NET project settings

```
using System;
using System.Reflection;

[assembly: AssemblyVersion("1.0.0.1")]
[assembly: AssemblyCulture("neutral")]
```

**contains public & private key pair**

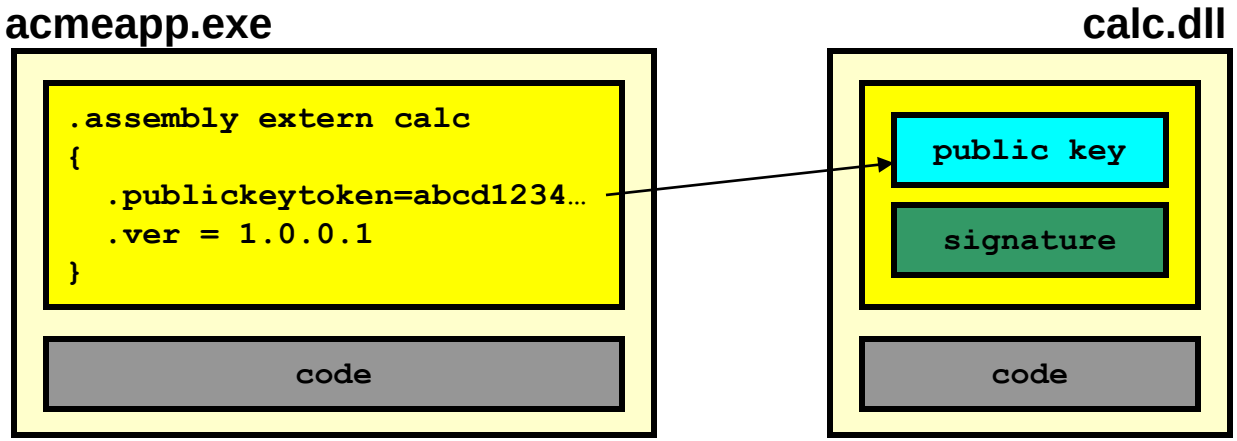
```
C:>csc /keyfile:acmekeys.snk /t:library calc.cs
```




**"calc, Version=1.0.0.1, Culture=neutral, PublicKeyToken=abcd1234.."**

# Signed Assemblies & References

- **Strongly named assemblies contain a signature**
  - assembly hash encrypted with publisher's private key
  - copy of public key embedded in manifest for clients
    - referencing assemblies note the key *token* for future use

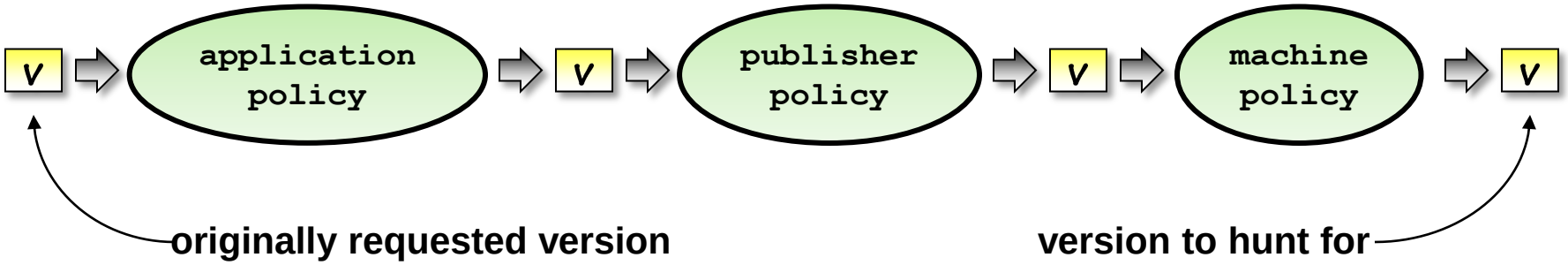


# Strongly Named Assembly Resolution

- **Loading of signed assemblies is more involved**
  - requested version optionally mapped to alternate version
  - is the target version already in memory?
  - is the target version installed in the Global Assembly Cache (GAC)
    - local file system repository in support of side-by-side (SxS) versioning
  - has a CODEBASE hint been provided
    - an arbitrary URI (file system, network share, HTTP)
  - fallback on private path probing
  - strong name validation

# Version Policy

- **Assembly version is configurable via config files**
  - per-application
  - per-publisher (strongly named assembly developer)
  - per-machine
  - (any/all of which may be a no-op)



# Global Assembly Cache (GAC)

- **The GAC supports side-by-side deployment of assemblies**
  - multiple versions of the 'same' assembly present on a machine
  - clients indicate desired version of assembly to load
  - if resolved version exists in the GAC, it's used as-is
    - strong name validation occurs at GAC-install time, *not* load time
    - a point of vulnerability if write access to GAC can be attained

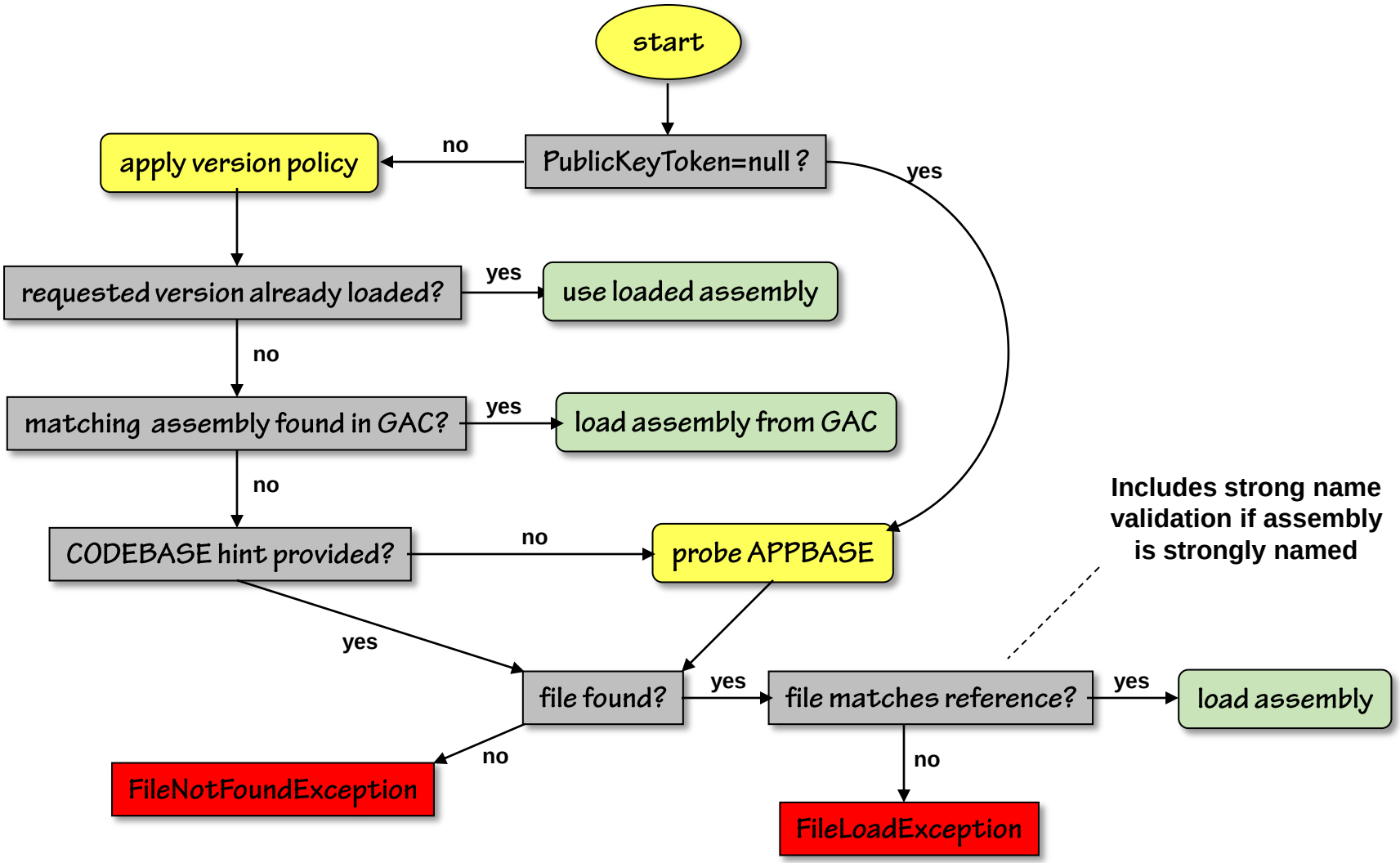
# Codebase Hints

- **Assemblies can be loaded from off-host via CODEBASE hints**
  - Used after assembly not found in memory or GAC
  - Used before private path probing

# Native Code Cache

- **Assemblies can be "preJITed" on machines where deployed**
  - IL-based assembly deployed on target machine normally
  - ngen.exe utility run to compile IL to processor-specific code
    - ngen == 'native code generator' or 'native image generator'
    - run on target machine, not dev machine
    - image generated is named assembly.ni.extension
      - (eg: mscorlib.ni.dll)
  - the ngen cache is consulted when an assembly is resolved
    - does a native image exist for the target assembly to be loaded?
    - does it's MVID (module version ID) match?
      - the MVID is a GUID in each assembly's metadata
      - generated anew when assembly image is emitted by a compiler
      - recorded for future reference when native image is generated

# Summary


start
PublicKeyToken=null ?
no
yes
apply version policy
requested version already loaded?
yes
use loaded assembly
no
matching assembly found in GAC?
yes
load assembly from GAC
no
CODEBASE hint provided?
no
probe APPBASE
yes
file found?
yes
file matches reference?
yes
load assembly
no
FileNotFoundException
no
FileLoadException
Includes strong name validation if assembly is strongly named

# References

- *Portable Executable and Common Object File Format Specification*
  - http://www.microsoft.com/whdc/system/platform/firmware/PECOFFdwn.mspx
- Common Language Infrastructure (CLI) Specification
  - http://link.pluralsight.com/netspecs
  - Partition IIA: Metadata Semantics
  - Partition IIB: Metadata File Format
- *The Common Language Infrastructure Annotated Standard*
  - by James S. Miller & Susann Ragsdale; Addison-Wesley
  - Miller was the lead software architect of the CLR
- *Expert .NET 2.0 IL Assembler*
  - by Serge Lidin; APress
  - CLR team member in charge of
    - IL assembler (ilasm.exe) & disassembler (ildasm.exe)
    - metadata validator (peverify.exe)
    - run-time metadata validation component of the EE