Claims-based Identity, Access Control & Personalization

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

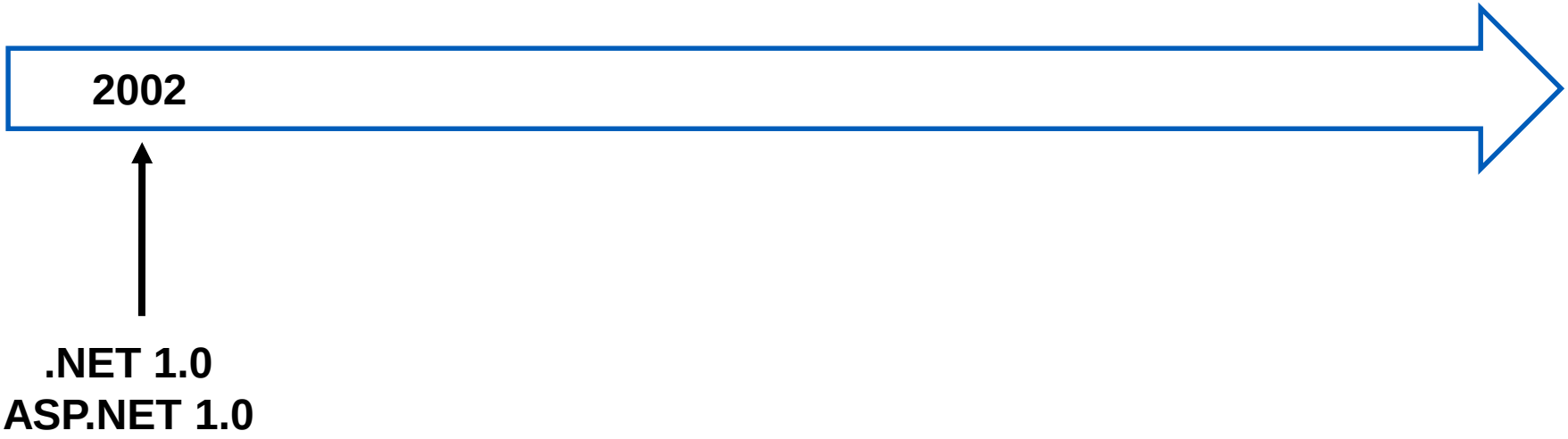
Objectives

- History of authentication/authorization infrastructure in .NET
- Security tokens and claims
- Windows Identity Foundation architecture & APIs
- Claims transformation
- Claims based authorization
- Federation
- Single sign-on

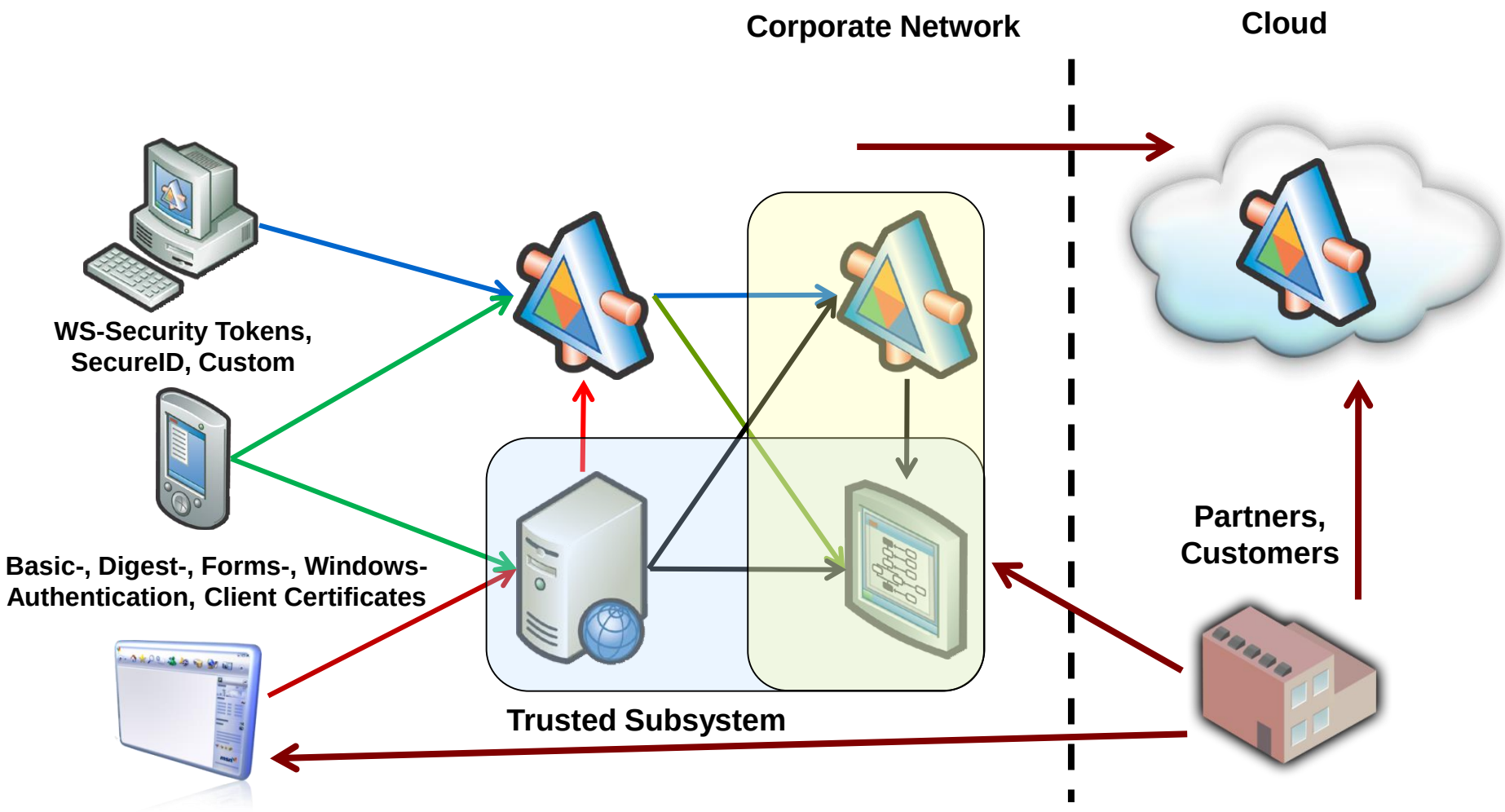
Once upon a time…

```
interface IIdentity
{
  bool IsAuthenticated { get; }
  string AuthenticationType { get; }
  string Name { get; }
}
```

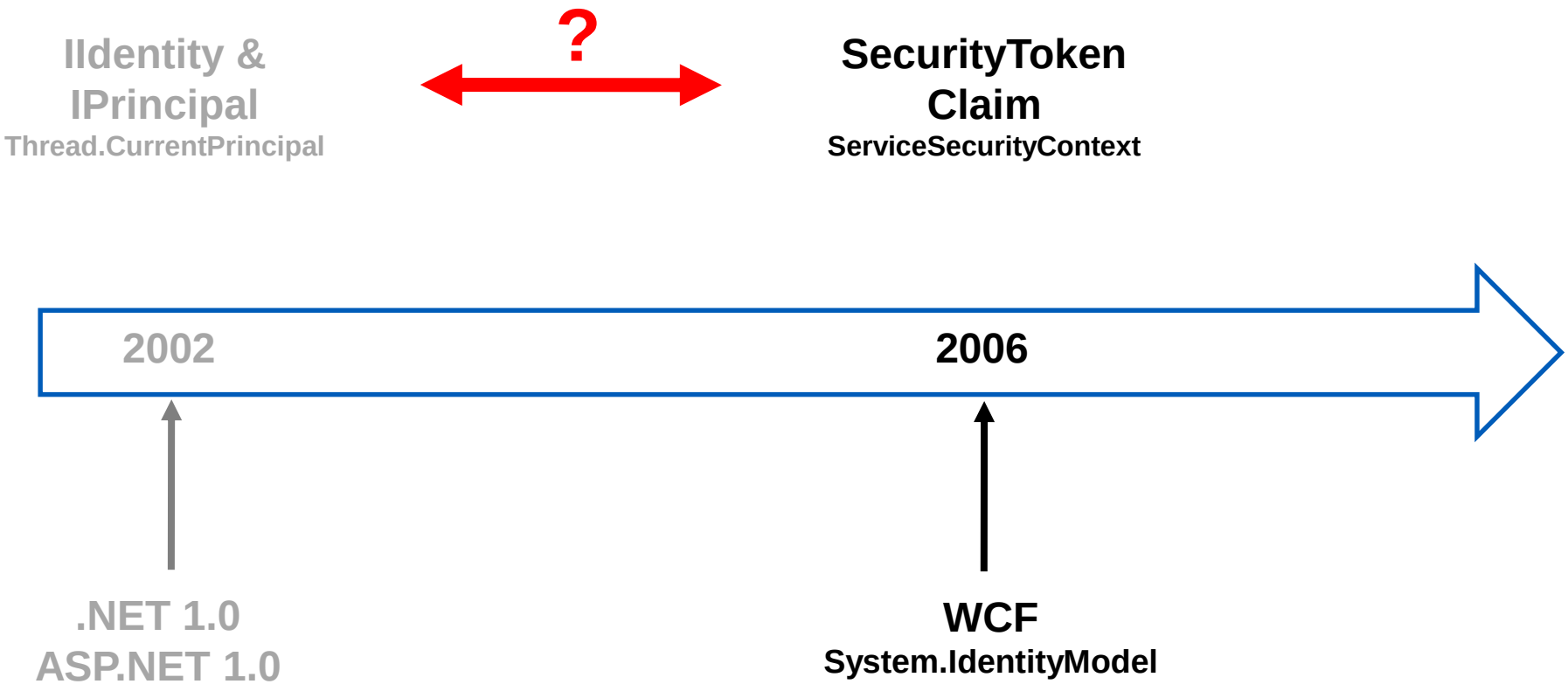
```
interface IPrincipal
{
  IIdentity Identity { get; }
  bool IsInRole(string roleName);
}
```



2002 - present



First attempt to solve the problem…



Claims

- **Many security systems out there**
  - groups, roles
  - permissions, capabilities
  - specialized (e.g. Bell LaPadula)

- **Examples**
  - Bob is an administrator
  - Jims email address is jim@foo.com
  - Alice is allowed to add new customers
  - Dave is allowed to write documents up to ‚confidential‘

Claims

- **Statement about an entity made by someone else**

```
public class Claim
{
    public virtual string ClaimType { get; }
    public virtual string Value { get; }
    public virtual string Issuer { get; }

    // rest omitted
}
```

Microsoft® .NET | Windows Identity Foundation

IIDentity & IPrincipal
Thread.CurrentPrincipal

SecurityToken Claim
ServiceSecurityContext

IClaimsIdentity & IClaimsPrincipal
Thread.CurrentPrincipal

.NET 1.0
ASP.NET 1.0

WCF
System.IdentityModel

Windows Identity Foundation
Microsoft.IdentityModel
(ASP.NET, WCF…)

IClaimsPrincipal & IClaimsIdentity

```
interface IIdentity
{
  bool IsAuthenticated       { get; }
  string AuthenticationType  { get; }
  string Name                { get; }
}
```

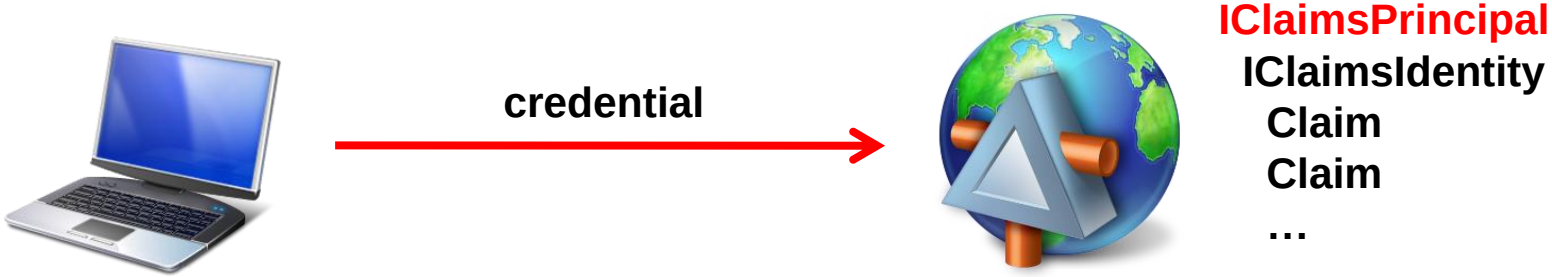
```
interface IPrincipal
{
  IIdentity Identity { get; }
  bool IsInRole(string roleName);
}
```

```
interface IClaimsIdentity : IIdentity
{
  ClaimCollection Claims { get; }
  string NameClaimType   { get; set; }
  string RoleClaimType   { get; set; }
}
```

```
interface IClaimsPrincipal : IPrincipal
{
  ClaimsIdentityCollection Identities { get; }
}
```

What can WIF do for you?

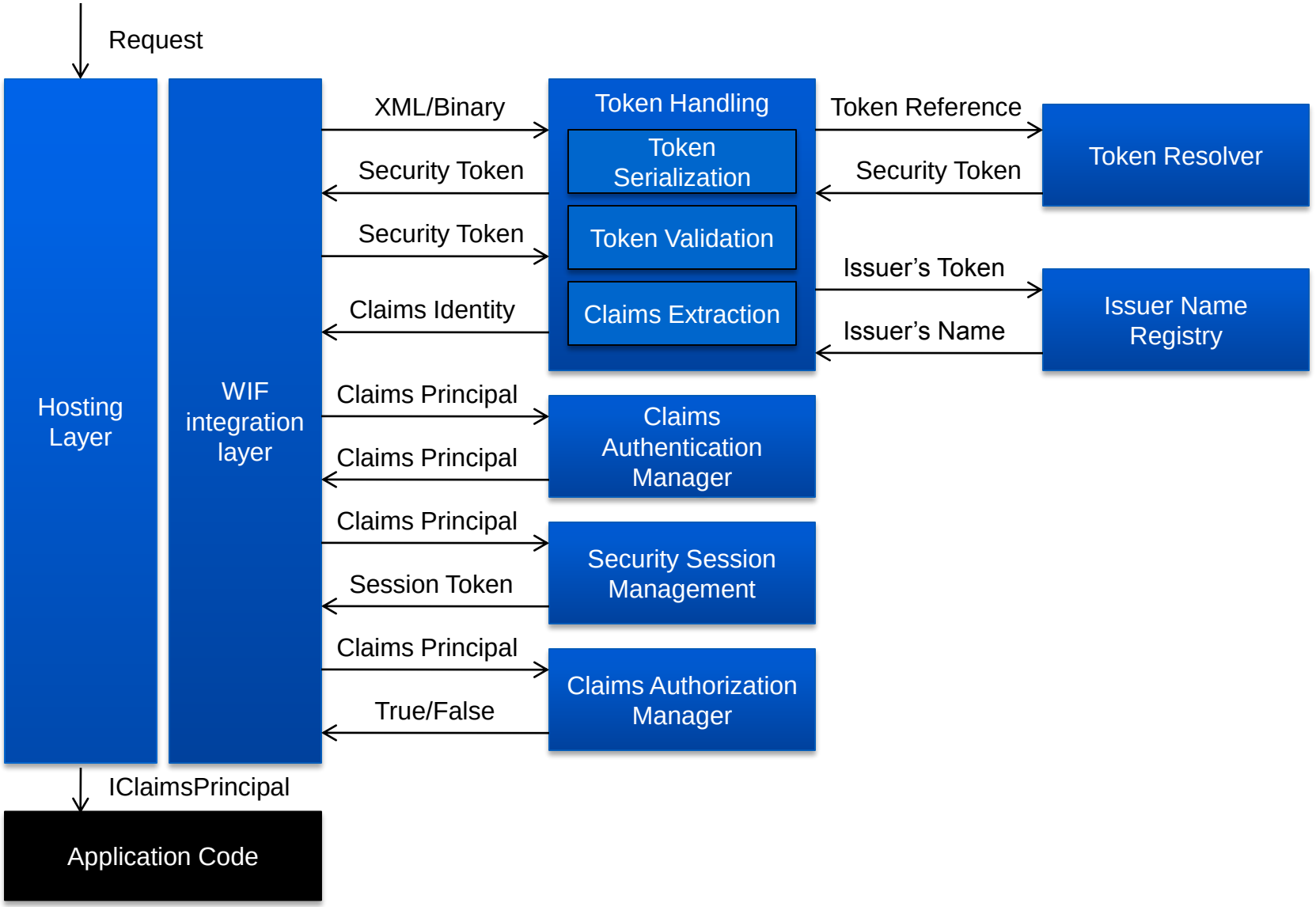
- **Conversion of various credential formats to common IClaimsPrincipal representation**
  - Kerberos
  - HTTP authentication
  - SSL client certificates
  - WS-Security tokens
  - SAML
  - extensible



Enabling WIF

- **WIF has an extensible hosting API**
  - built-in support for ASP.NET and WCF
- **ASP.NET**
  - ClaimsPrincipalHttpModule
  - ClaimsAuthorizationModule
  - WSFederationAuthenticationModule
  - SessionAuthenticationModule
- **WCF**
  - ConfigureServiceHostBehavior
  - custom ServiceHostFactory

WIF pipeline



Claims authentication

- **ClaimsAuthenticationManager allows to**
  - add, transform, reject claims

```
public class ClaimsAuthNManager : ClaimsAuthenticationManager
{
  public override IClaimsPrincipal Authenticate(
    string resourceName, IClaimsPrincipal incomingPrincipal)
  {
    if (incomingPrincipal.Identity.IsAuthenticated)
    {
      return TranformClaims(incomingPrincipal);
    }

    return incomingPrincipal;
  }
}
```

Session management

- **Result of claims transformation can be cached in a sesion**
  - cookies for ASP.NET
  - SecureConversation for WCF
- **Extensible mechanism**
  - session token protection
  - web farm support
  - round trip optimization

Claims authorization

- **ClaimsAuthorizationManager provides central extensibility point for**
  - loading/parsing authorization policy
  - mapping operations/resources to required claims
- **Can be auto-invoked during request processing**
  - with HTTP method / URL (ASP.NET)
  - with SOAP action / URL (WCF)
- **Application code should not check for claims directly**

```
public class ClaimAuthZManager : ClaimsAuthorizationManager
{
  public override bool CheckAccess(AuthorizationContext context)
  {
    // inspect context and make authorization decision
  }
}
```

Authorization context

- **Allows complex description of resource access**
  - resource / action pair
  - can be claims

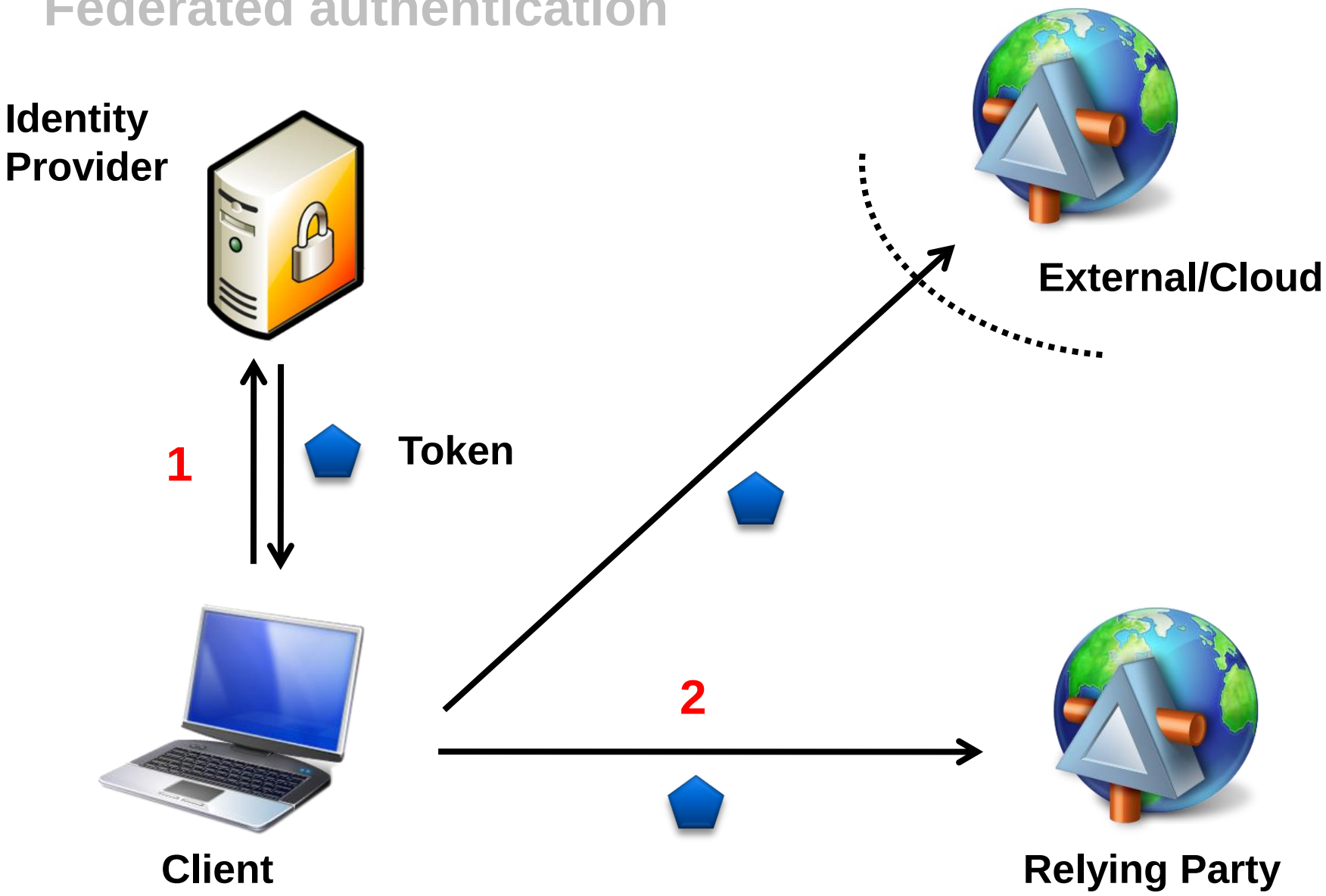
```
[ClaimsPrincipalPermission(SecurityAction.Demand,
  Operation = "Add", Resource = "Customer")]
public void AddCustomer(Customer customer) { … }
```

```
void Print(Document document)
{
  if (ClaimsPrincipalPermission.CheckAccess(
    document.Printer, "Print"))
    { … }
}
```

Federated identity

- **Authentication is moved to a central authentication service**
  - called a security token service (STS)
- **STS turns credential into a standard token type**
  - typically SAML 1.1/2.0
- **Applications establish trust with STS**
- **Enables many interesting scenarios**
  - rich claims
  - single sign-on
  - federation
  - identity delegation

Federated authentication



Active token request (WS-Trust)

Identity Provider

RST/ RSTR

```
<RequestSecurityToken>
  <RequestType>Issue</RequestType>
  <TokenType>SAML#1.1</TokenType>
```

```
<RequestSecurityTokenResponse>
  <saml:Assertion>
    …
  </saml:Assertion>
<RequestSecurityTokenResponse>
```

```
  </AppliesTo>
<RequestSecurityToken>
```

Client

SOAP w/ security header

Relying Party

Passive token request (WS-Federation)

Identity Provider

auth.aspx?wa=wsignin1.0&wtrealm=address_of_rp

```
<form method="POST" action="http://app/default.aspx">
  <input name="wresult" value="<saml:Assertion…" />
  …
  <script >
      window.setTimeout('document.forms[0].submit()', 0);
  </script>
</form>
```

GET /auth.aspx

Client

GET /default.aspx

POST /default.aspx

Relying Party

SAML token

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
  <saml:AttributeStatement>
    <saml:Attribute AttributeName="userid"
                    AttributeNamespace="http://...">
      <saml:AttributeValue>42</saml:AttributeValue>
    </saml:Attribute>

    <saml:Attribute AttributeName="name"
                    AttributeNamespace="http://... ">
      <saml:AttributeValue>Dominick</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute AttributeName="department"
                    AttributeNamespace="http://... ">
      <saml:AttributeValue>Research</saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>

  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#" />
</saml:Assertion>
```

Setting up federation – step 1

- **Establish trust with identity provider**
  - only accept tokens from issuers you trust

```
<microsoft.identityModel>
  <service>
    <issuerNameRegistry type="....ConfigurationBasedIssuerNameRegistry, …">
      <trustedIssuers>
        <add thumbprint="032f5976187b18d9f9514381196531 22fd45172c"
             name="CorporateIdP"/>
      </trustedIssuers>
    </issuerNameRegistry>
  </service>
</microsoft.identityModel>
```

Setting up federation – step 2

- **Set token expectations**
  - audience is a URI embedded in the token
  - specifies the intended receiver
  - you only want to accept tokens with a known value

```
<microsoft.identityModel>
  <service>

    <audienceUris>
      <add value="http://www.company.com/app" />
    </audienceUris>


  </service>
</microsoft.identityModel>
```

Setting up federation – step 3

- **Set token decryption key**
  - identity provider can encrypt token
  - additional security
  - client cannot inspect token

```
<microsoft.identityModel>
  <service>

  <serviceCertificate>
    <certificateReference storeName="My"
                          storeLocation="LocalMachine"
                          x509FindType="FindByThumbprint"
                          findValue="abc" />
  </serviceCertificate>


  </service>
</microsoft.identityModel>
```

Setting up federation – step 4 (ASP.NET)

- **Set up WS-Federation**
  - login URL
  - application identifier

```
<microsoft.identityModel>
  <service>

    <federatedAuthentication>
      <wsFederation passiveRedirectEnabled="true"
                    issuer="https://login.company.com//issue.aspx"
                    realm="http://www.company.com/app" />
      </federatedAuthentication>


  </service>
</microsoft.identityModel>
```

Setting up federation – step 4 (WCF)

- **Set up WS-Trust**
  - WCF federation binding
  - link to identity provider's metadata endpoint
  - svcutil.exe can generate client code and configuration

```
<system.serviceModel>
  <bindings>
    <ws2007FederationHttpBinding>
      <binding name="fed">
        <security>
          <message>
            <issuerMetadata address="https://login.company.com/issue.svc/mex" />
          </message>
        </security>
      </binding>
    </ws2007FederationHttpBinding>
  </bindings>
</system.serviceModel>
```

Summary

- **Claims-based identity is the new .NET security model**
  - will gradually move into all major products
- **Windows Identity Foundation is the toolkit for claims**
  - token handling
  - protocol support
  - application framework integration
- **Claims are especially attractive when you need to**
  - cross security boundaries
  - bridge authentication protocols or credential types
  - interop with other security systems