Tutorial Description

Security Concepts for Distributed Component Systems

Instructor

Walt Smith, Director of Technology at Tekna, Inc

Summary of Topics to be addressed

Building distributed component systems introduces a new complexity to enterprise security planning. This seminar will introduce the participant to n-tier, distributed component systems, the technologies they are built on and the security concepts associated with them. We will focus on CORBA as a distribution architecture and look at some of the unique challenges it presents to the security planner.

Biography of Walt Smith

Mr. Smith is the Director of Technology for Tekna, Inc. and is located at their Richmond, VA headquarters. He is responsible for the technical strategies, methodologies, and technical staff management of Tekna and oversees their distributed component practice. Prior to joining Tekna, Mr. Smith was an object oriented systems designer for a major financial corporation. He has been a manufacturing, logistics and supply-chain systems consultant for 9 years.

The Web-Enabled Enterprise

Security Concepts for Distributed Component Systems

Agenda

- Introduction to n-tier distribution
- Introduction to CORBA
- Overview of CORBA security
- Questions and Answers

tekna

"In the near future, internet technologies will form the backbone of all enterprise applications development..."

Key Assumptions

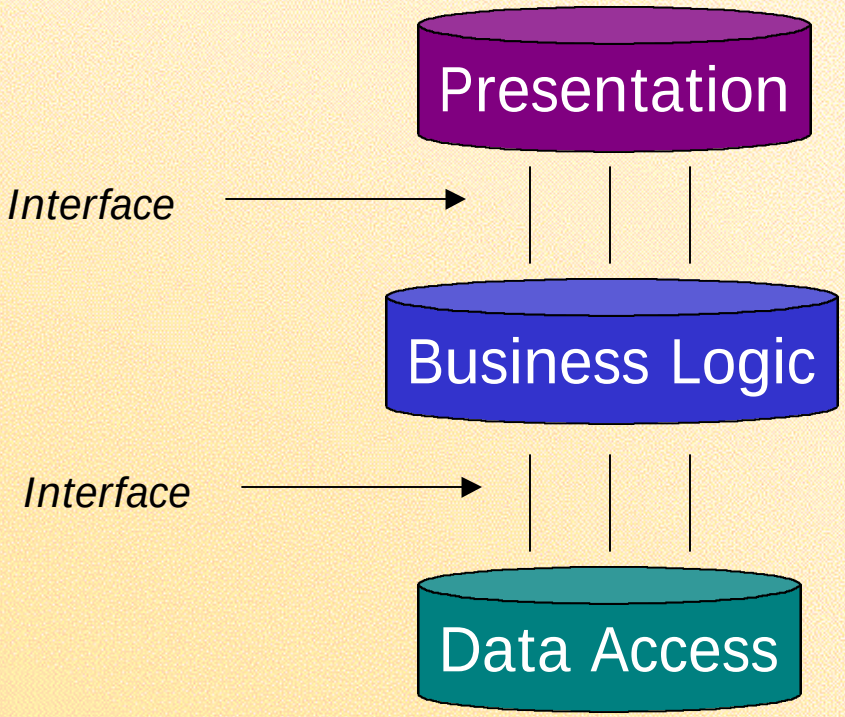
- N-tier application development is here to stay.
- In the near future, ALL n-tier application development will use Internet protocols as a transport mechanism.
- Distributed component technologies represent the most likely scenario for n-tier Internet application development.

tekna

Background

- What is an n-tier application?
- What is a component?
- What is a distributed component?
- What is CORBA?

What is an n-tier application?



- Displays information
- Accepts User feedback

- Enforces business rules
- Validates information
- Can be 1 to n tiers

- Persists data
- Interfaces to external systems
- Can be 0 to n tiers

Why n-tier is good

- Allows a clear separation of technology boundaries
- Encourages vendor independence at all levels
- Is extremely scalable

tekna

What is a component?

Any discrete chunk of business functionality that is accessed via a **well-defined interface**

“A component does one thing, and does it very well...”

Why components are good

- Maximize modularity of a system, improving maintainability
- Promote vendor independence
- Are extensible, improving reuse at all levels.

Components vs. Objects

Components

- Have a well-defined interface
- Act cooperatively to form a system

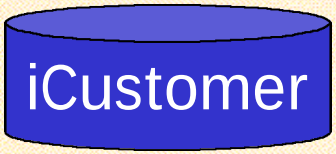
Objects

- Have a well-defined interface
- Hide their local state
- Encapsulate both data and procedures
- Use a hierarchy of inheritance
- Act cooperatively to form a system

A business component

A customer

=

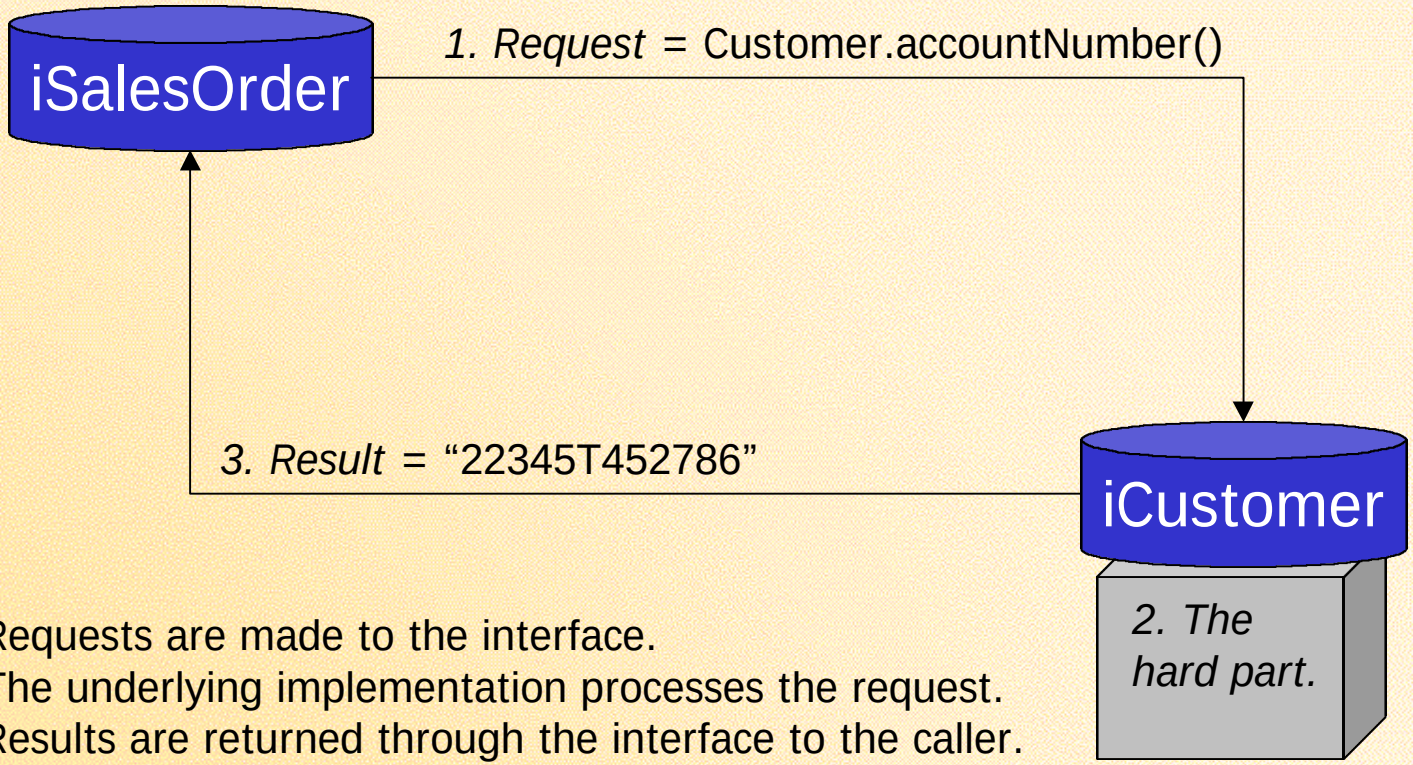


Interface

iCustomer.accountNumber()
iCustomer.name()
iCustomer.age()
iCustomer.balance()

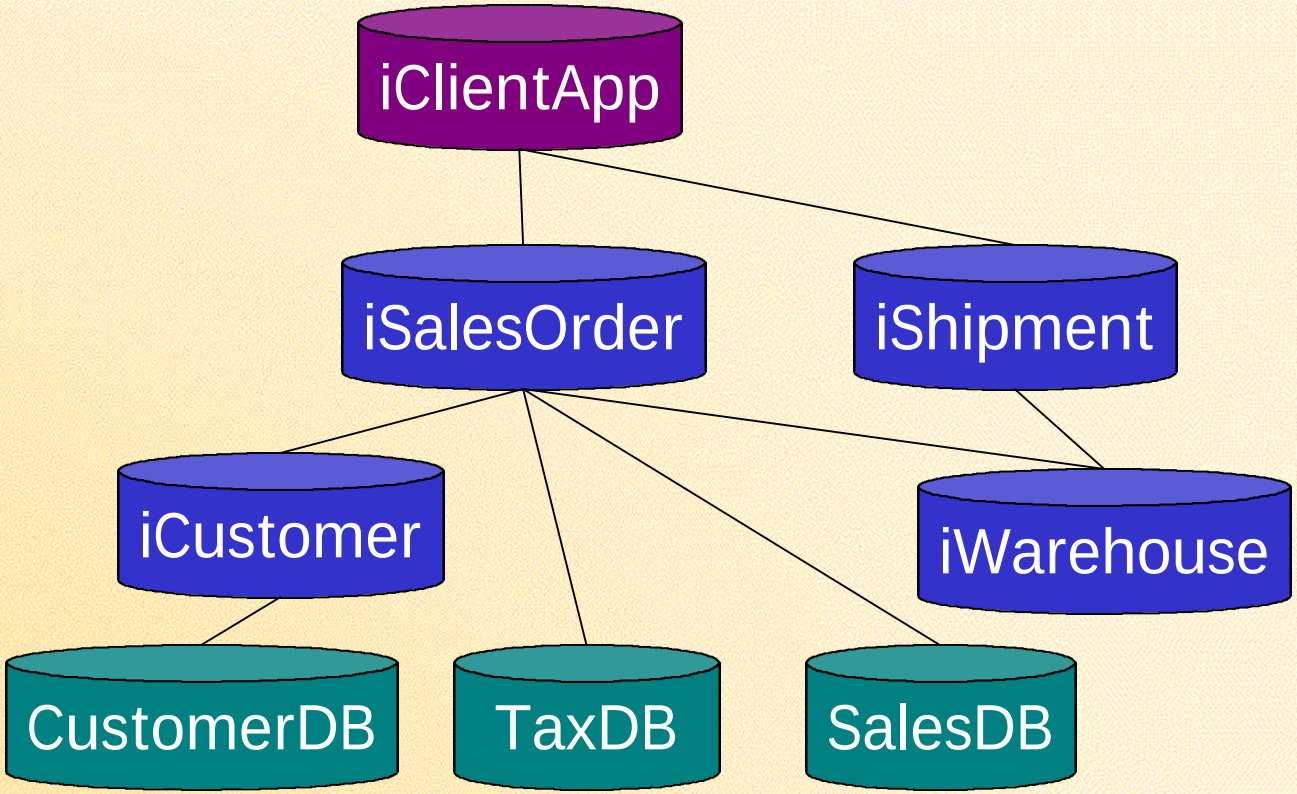
iCustomer.sendBill()

How components talk

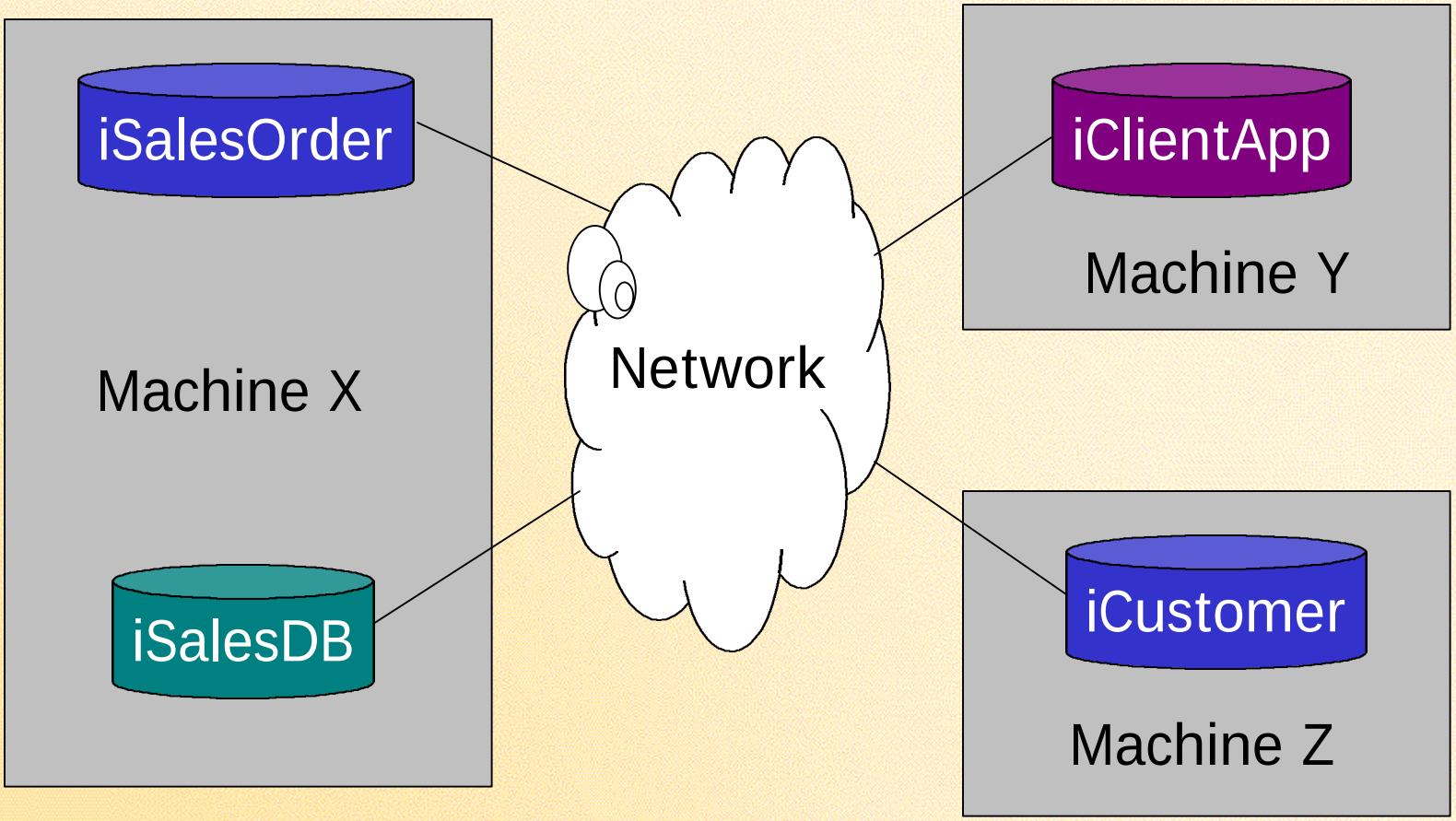


1. Requests are made to the interface.
2. The underlying implementation processes the request.
3. Results are returned through the interface to the caller.

A component based system



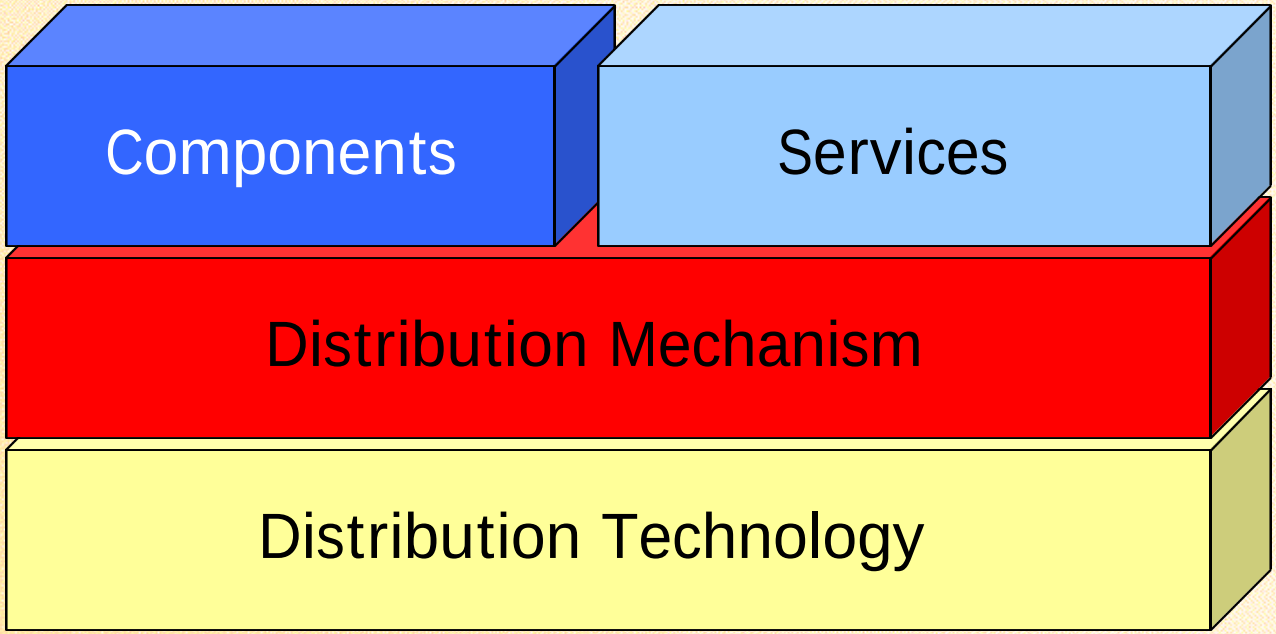
What are distributed components?



Why distribution is good

- Eases load balancing tasks
- Increases system scalability
- Promotes vendor independence
- Increases reliability through redundancy

Distribution Architecture

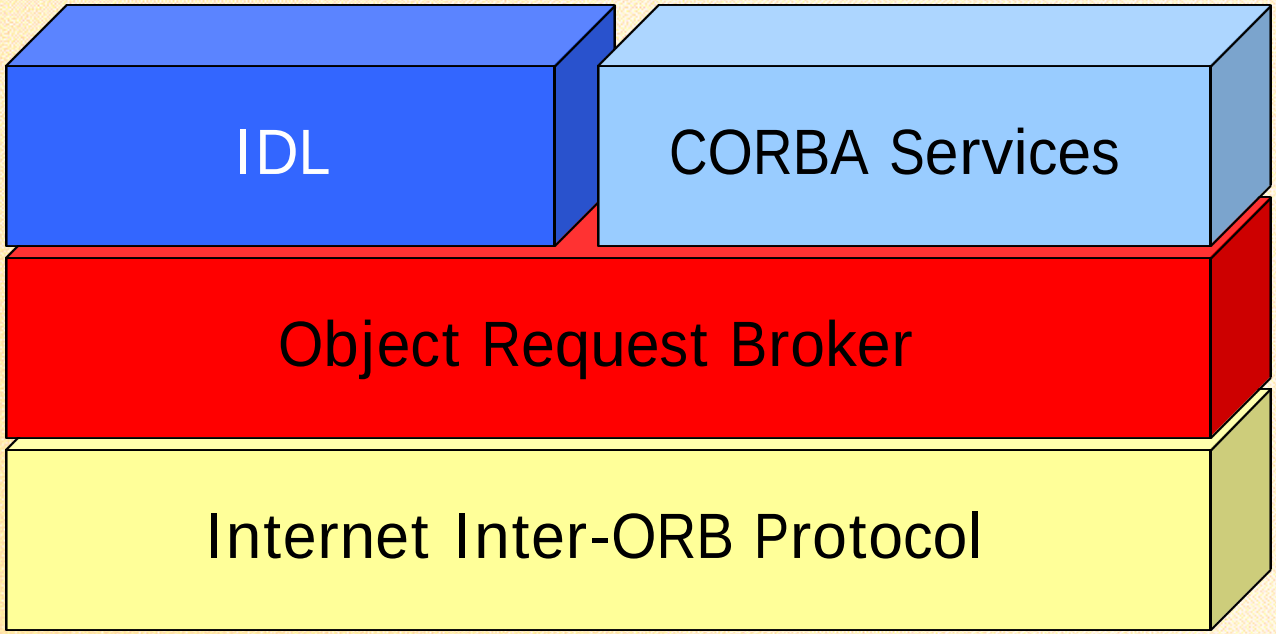


Some popular distribution mechanisms

Common Object Request Broker Architecture (CORBA)
Distributed Computing Environment (DCE)
Distributed Component Object Model (DCOM)**
Remote Method Invocation (RMI)*

* RMI is now a special case of CORBA
**DCOM now features inter-operability with CORBA

CORBA Architecture



What is CORBA?

- **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
- An internationally defined standard for component distribution
- Overseen by the Object Management Group
- Implemented by many vendors including IONA, Visigenic, SUN and IBM
- Platform, language and vendor independent

Why CORBA is good

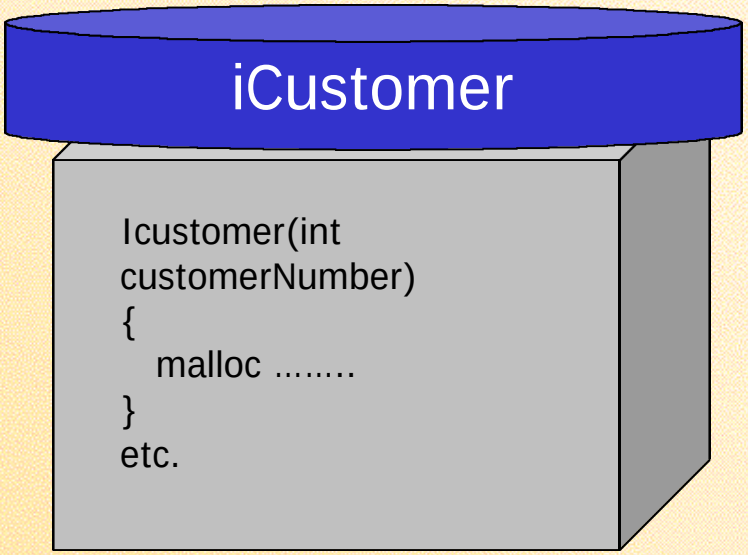
- Separates the definition of a component from it's implemenation
- Makes reuse available across all code bases (C++, JAVA, COBOL, ADA, etc)
- Allows a gradual migration of procedural code to component technology

Important Note

The CORBA standard refers to **components** as **objects**. A CORBA object may be either a component or a “true” object. From here on, we use the term object interchangeably with the word component. This is NOT an endorsement of the objectivity of CORBA.

CORBA Architecture

Interface Definition Language



Platform independent language to define interface.

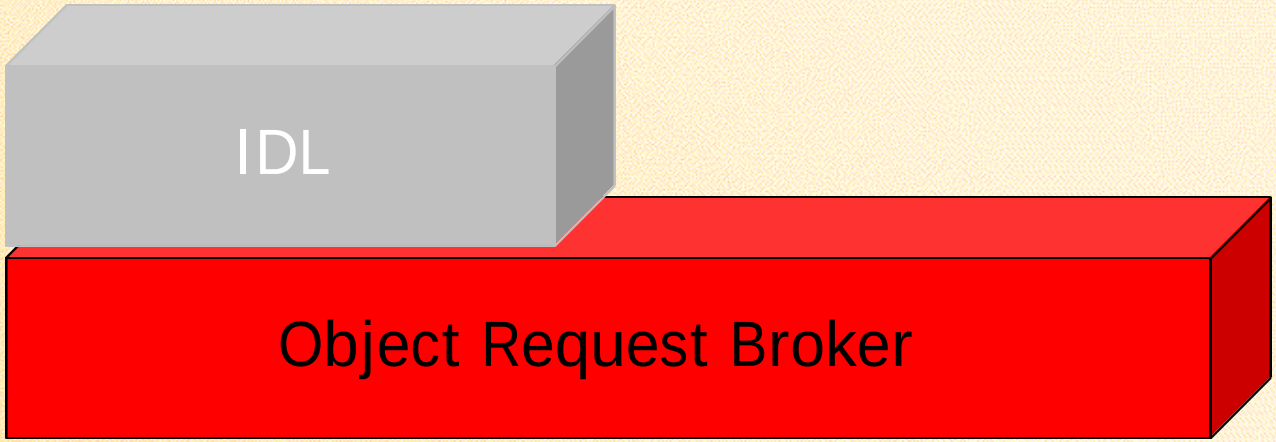
Hides the platform dependent implementation details from other components in the system.

IDL is usually be tied to the implementation language at compile time via a pre-complier.

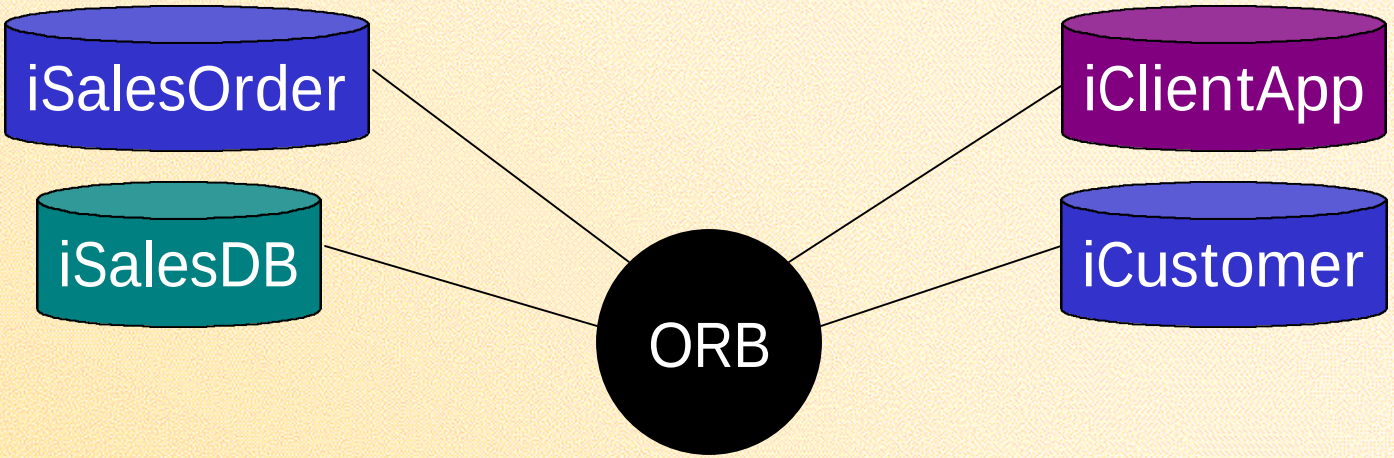
Example IDL

```
module mPartner{
        interface iCustomer{

                exception iCustomerException{
                        string reason;
                };

                string getCustomerName(
                        in long customerNumber)
                        raises (iCustomerException);
                void setCustomername(
                        in string customerName)
                        raises (iCustomerException);
        };
};
```

CORBA Architecture

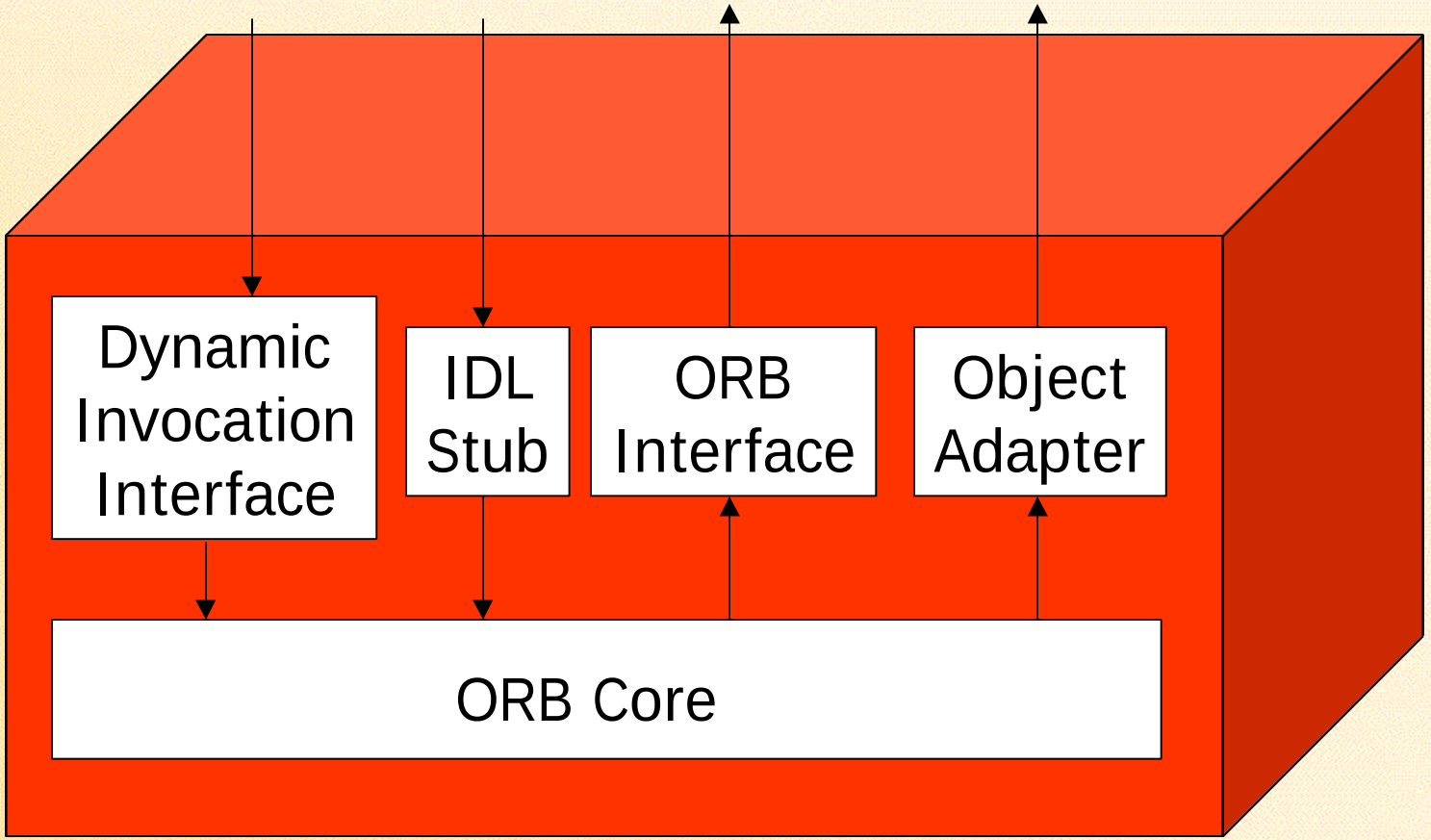


Object Request Broker

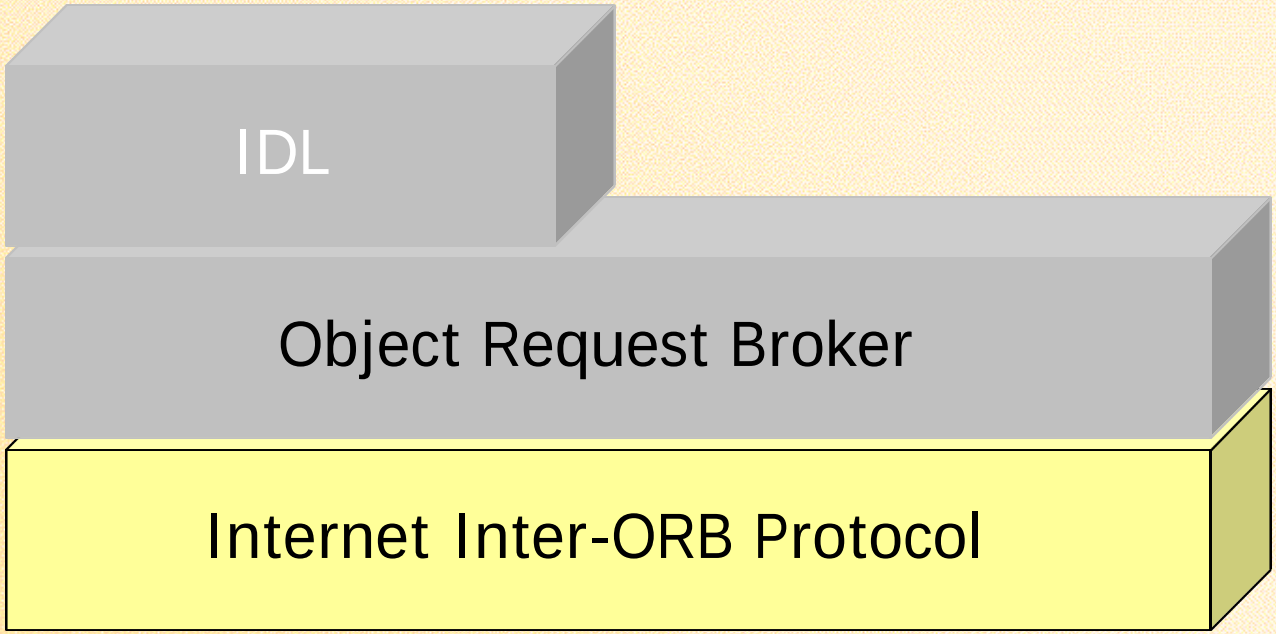


- Facilitates connections between objects
- Accepts requests to use objects, locates those objects and facilitates the connection to them.
- A facilitator, NOT a mediator! Does not mediate connections once they are established.

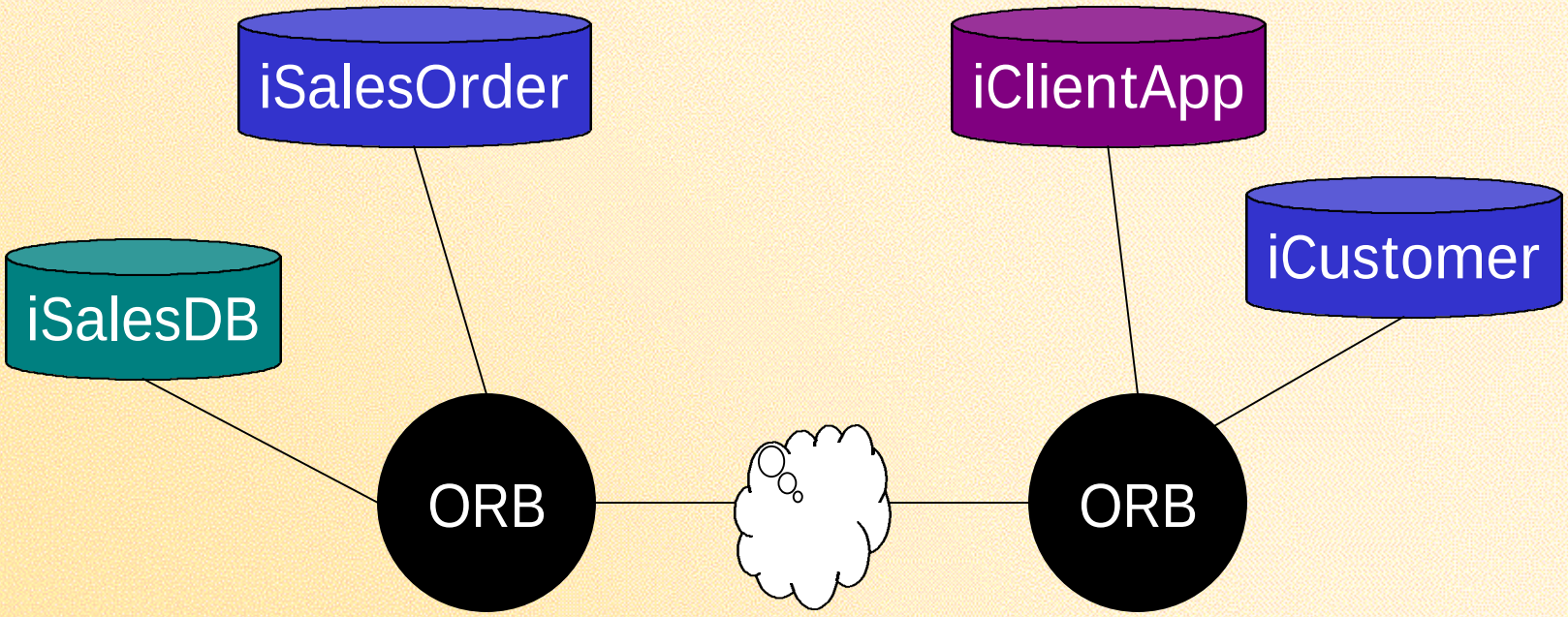
Interfaces to the ORB



CORBA Architecture



Internet Inter-ORB Protocol



Allows components and ORBs to communicate over a TCP/IP wire protocol

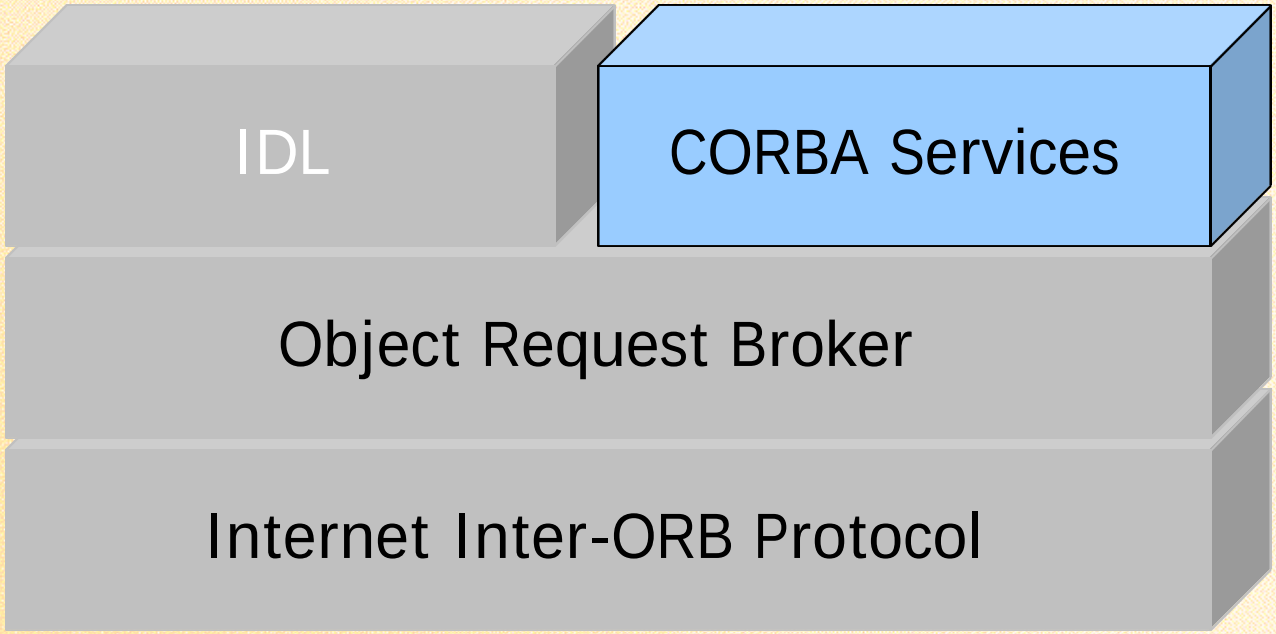
IIOP Factoids

- A special case of GIOP, the General Inter-ORB Protocol. The GIOP can be mapped onto any connection oriented transport layer.

- Uses Interoperable Object References (IOR's) to bridge requests between ORB's. These are basically platform independent data structures.

- Has become ubiquitous, even being used for intra-ORB communications. "The de-facto ORB communication protocol."

CORBA Architecture



CORBA Services

Provide a definition of infra-structure functionality for CORBA based systems to insulate the developer from some details of implementing a CORBA application. Examples;

Object Naming
Transaction Control
Persistence of Data
Event Notification
Concurrency (Object locking)
Security

The CORBA Security Service

•Defined by the OMG in Chapter 15 of the CORBA services white paper.

•Covers all aspects of security and defines the implementation details for each aspect.

•Based on KERBEROS, the DCE security model.

•Not all-encompassing (Read Appendix E, Guidelines to a trustworthy system)

Aspects of Security

- **Confidentiality -** Information is disclosed only to users authorized to access it.
- **Integrity -** Information is modified only by users who have the right to do so, and only in authorized ways. It is transferred only between intended users and in intended ways.
- **Accountability -** Users are accountable for their security-relevant actions. A particular case of this is non-repudiation, where responsibility for an action cannot be denied.
- **Availability -** Use of the system cannot be maliciously denied to authorized users.

Perceived Threats

- An authorized user of the system gaining access to information that should be hidden from him.
- A user masquerading as someone else, and so obtaining access to whatever that user is authorized to do, so that actions are being attributed to the wrong person.
- In a distributed system, a user may delegate his rights to other objects, so they can act on his behalf. This adds the threat of rights being delegated too widely, again causing a threat of unauthorized access.
- Security controls being bypassed.
- Eavesdropping on a communication line, so gaining access to confidential data.
- Tampering with communication between objects - modifying, inserting and deleting items.
- Lack of accountability due, for example, to inadequate identification of users.

Key Features

- **Identification** and **authentication** of principals (human users and objects which need to operate under their own rights) to verify they are who they claim to be.

- **Authorization** and **access control** - deciding whether a principal can access an object, normally using the identity and/or other privilege attributes of the principal (such as role, groups, security clearance) and the control attributes of the target object (stating which principals, or principals with which attributes) can access it.

- **Security auditing** to make users accountable for their security related actions. It is normally the human user who should be accountable. Auditing mechanisms should be able to identify the user correctly, even after a chain of calls through many objects.

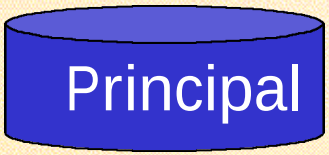
Key Features (continued)

•**Security of communication** between objects, which is often over insecure lower layer communications. This requires trust to be established between the client and target, which may require **authentication of clients to targets** and **authentication of targets to clients**. It also requires **integrity protection** and (optionally) **confidentiality protection** of messages in transit between objects.

•**Non-repudiation** provides irrefutable evidence of actions such as proof of origin of data to the recipient, or proof of receipt of data to the sender to protect against subsequent attempts to falsely deny the receiving or sending of the data.

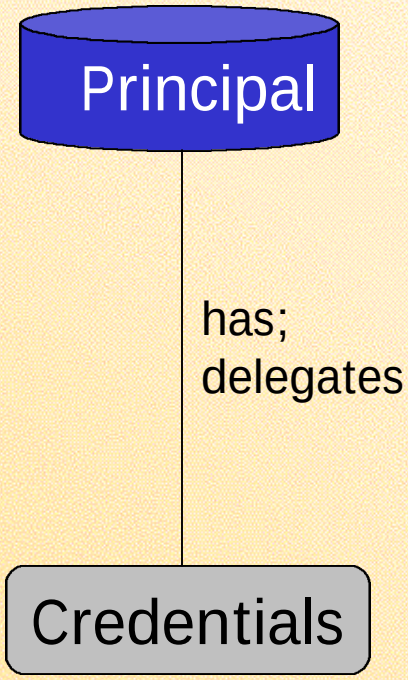
•**Administration** of security information (for example, security policy) is also needed.

Security model entities



A principal is a **human** user **or system entity** that is registered in and authentic to the system. Initiating principals are the ones that initiate activities. An initiating principal may be authenticated in a number of ways, the most common of which for human users is a password. For systems entities, the authentication information such as its long-term key, needs to be associated with the object.

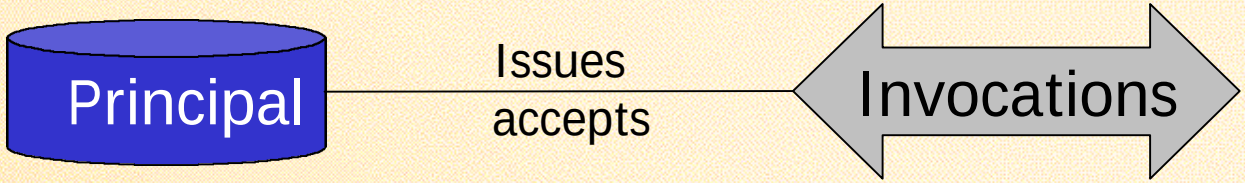
Security model entities



Credentials include:

- Unathenticated or Public attributes
- Authenticated Attributes:
  - identity attributes
  - privilege attributes

Security model entities



An invocation may involve:

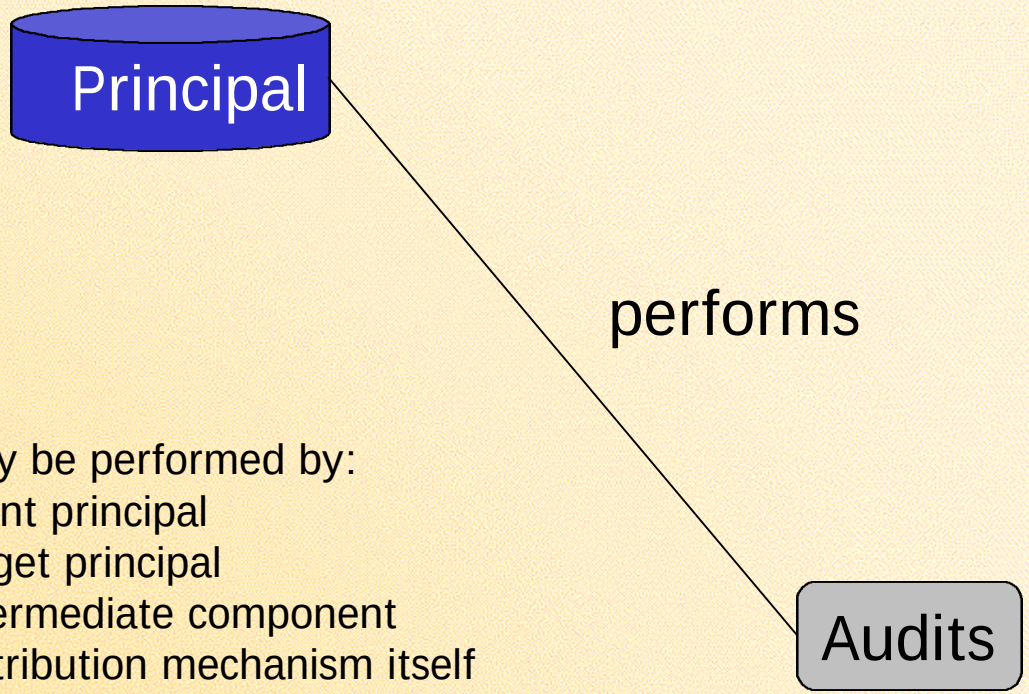
• Establishing a **security association** between the client and target object so that each has the required trust that the other is who it claims to be. In many implementations, associations will normally persist for many interactions, not just a single invocation. (Within some environments, the trust may be achieved by local means, without use of authentication and cryptography.)

• Deciding whether this client (acting for this principal) can perform this operation on this object according to the access control policy

• Auditing this invocation if required

• Protecting the request and response from modification or eavesdropping in transit, according to the specified quality of protection.

Security model entities



Audits may be performed by:

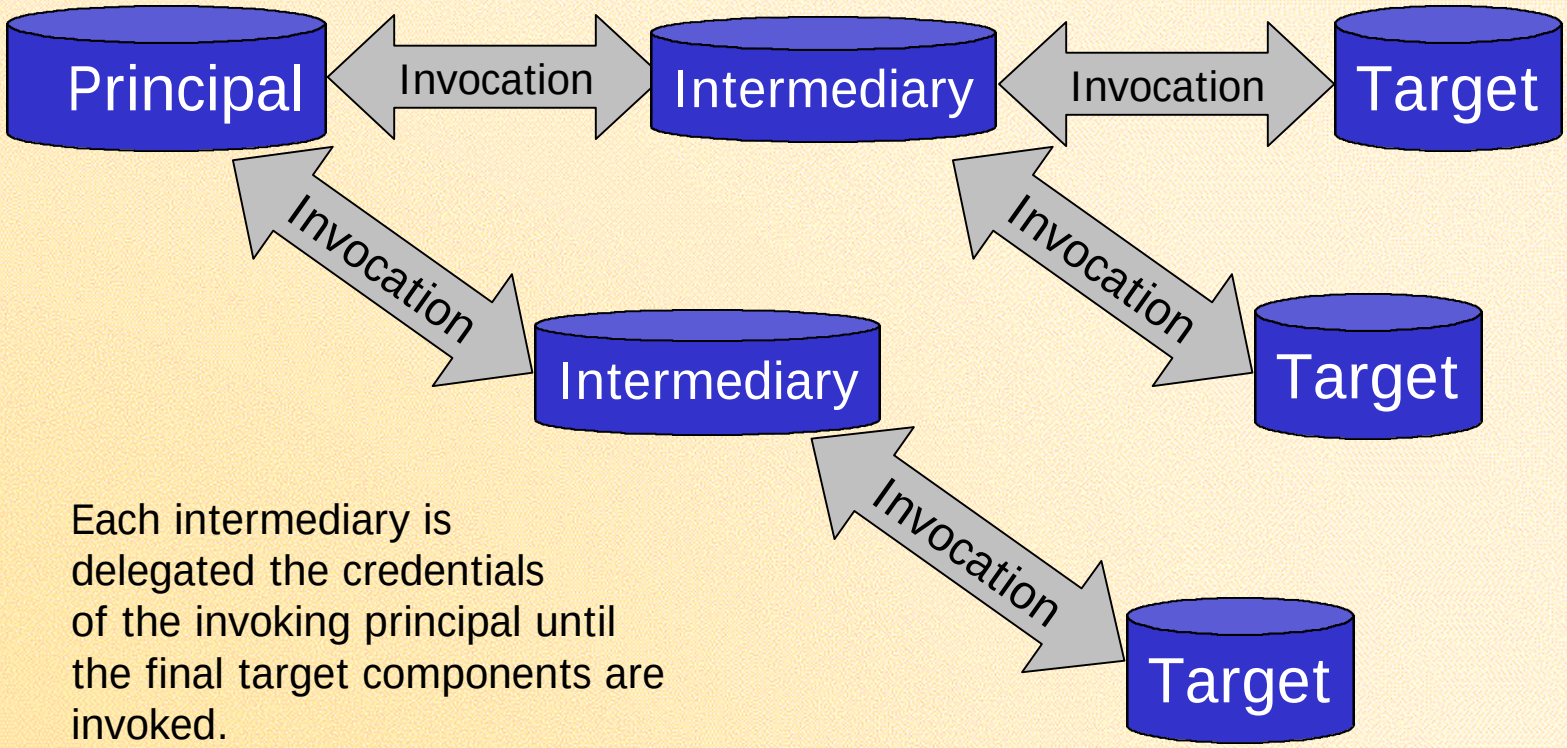
- The client principal
- The target principal
- Any intermediate component
- The distribution mechanism itself

(during invocation)

A closer look at...

- Delegation
- Non-repudiation
- Domains
- Auditing

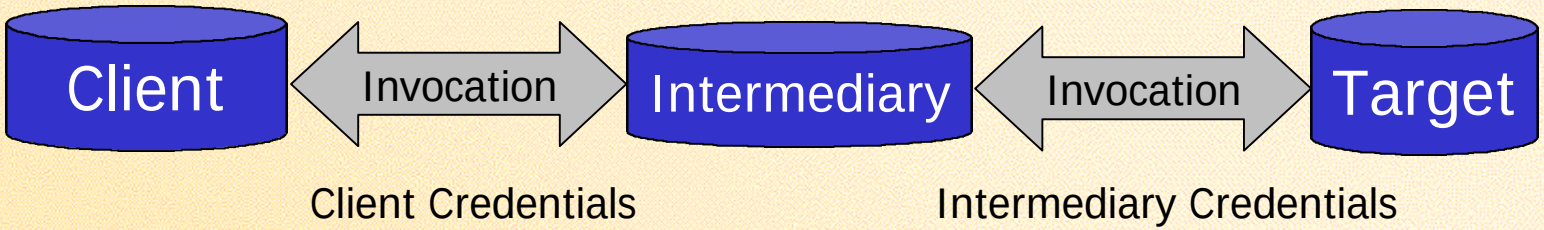
Delegation model



Each intermediary is delegated the credentials of the invoking principal until the final target components are invoked.

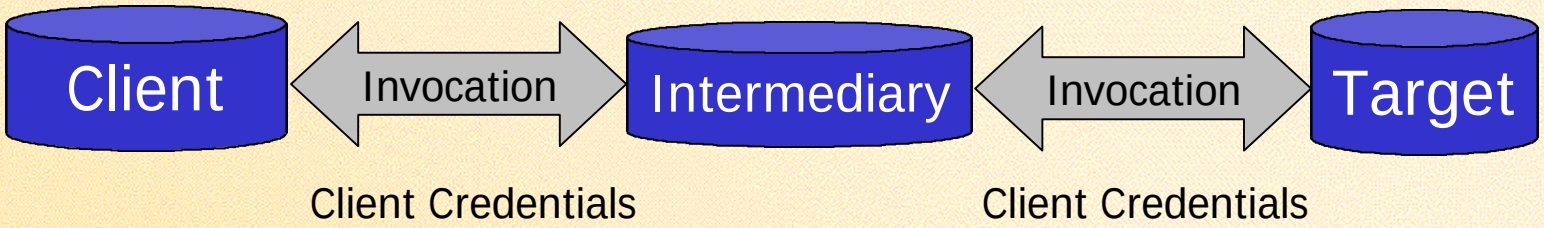
Types of Delegation

1. No Delegation



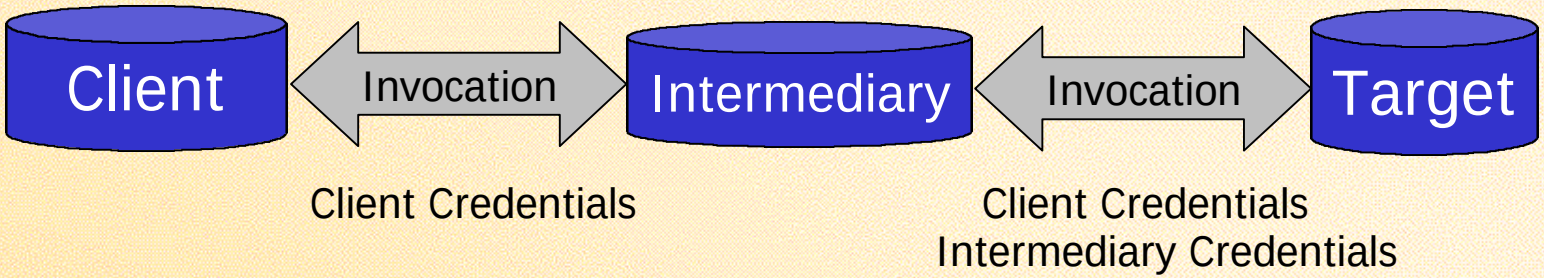
The client permits the intermediary to use its privileges for access control decisions, but does not permit them to be delegated, so the intermediary component cannot use these privileges when invoking the next component in the chain.

Types of Delegation
2. Simple Delegation



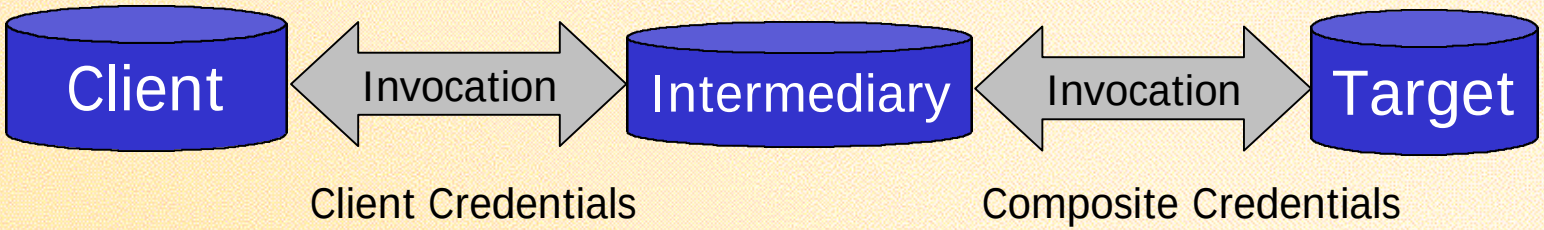
The client permits the intermediate to assume its privileges, both using them for access control decisions and delegating them to other others. The target object receives only the client's privileges, and does not know who the intermediate is.

Types of Delegation
3. Composite Delegation



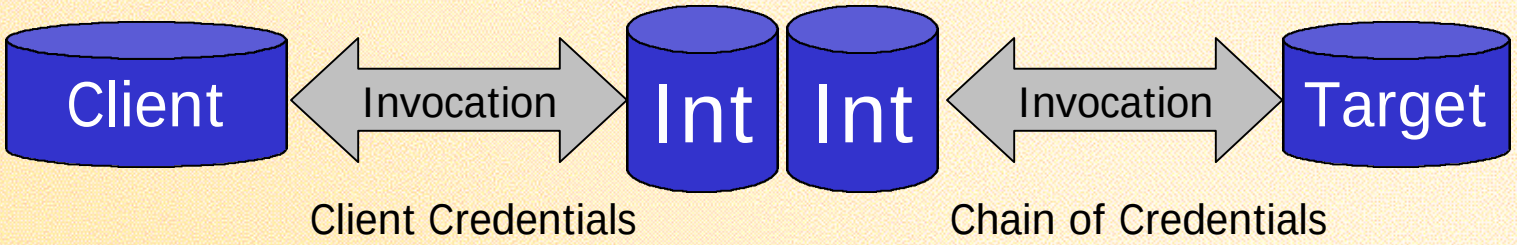
The client permits the intermediary component to use its credentials and delegate them. Both the client privileges and the immediate invoker's privileges are passed to the target, so that both the client privileges and the privileges from the immediate source of the invocation can be individually checked.

Types of Delegation
3. Combined Delegation



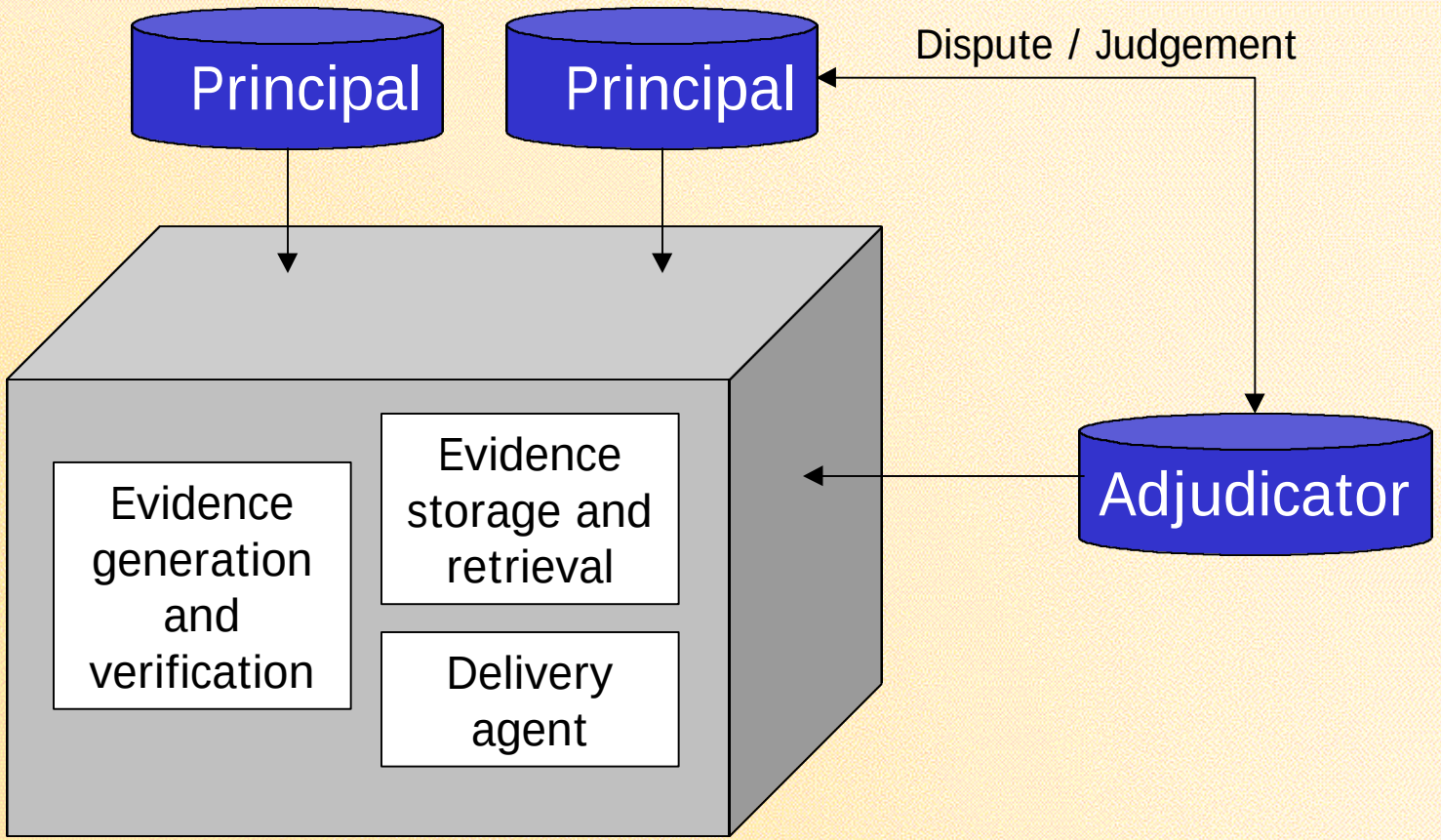
The client permits the intermediate object to use its privileges. The intermediate converts these privileges into credentials and combines them with its own credentials. In that case, the target cannot distinguish which privileges come from which principal.

Types of Delegation
4. Traced Delegation



The client permits the intermediary components to use its privileges and delegate them. However, at each intermediate component in the chain, the intermediate's privileges are added to privileges propagated to provide a trace of the delegates in the chain.

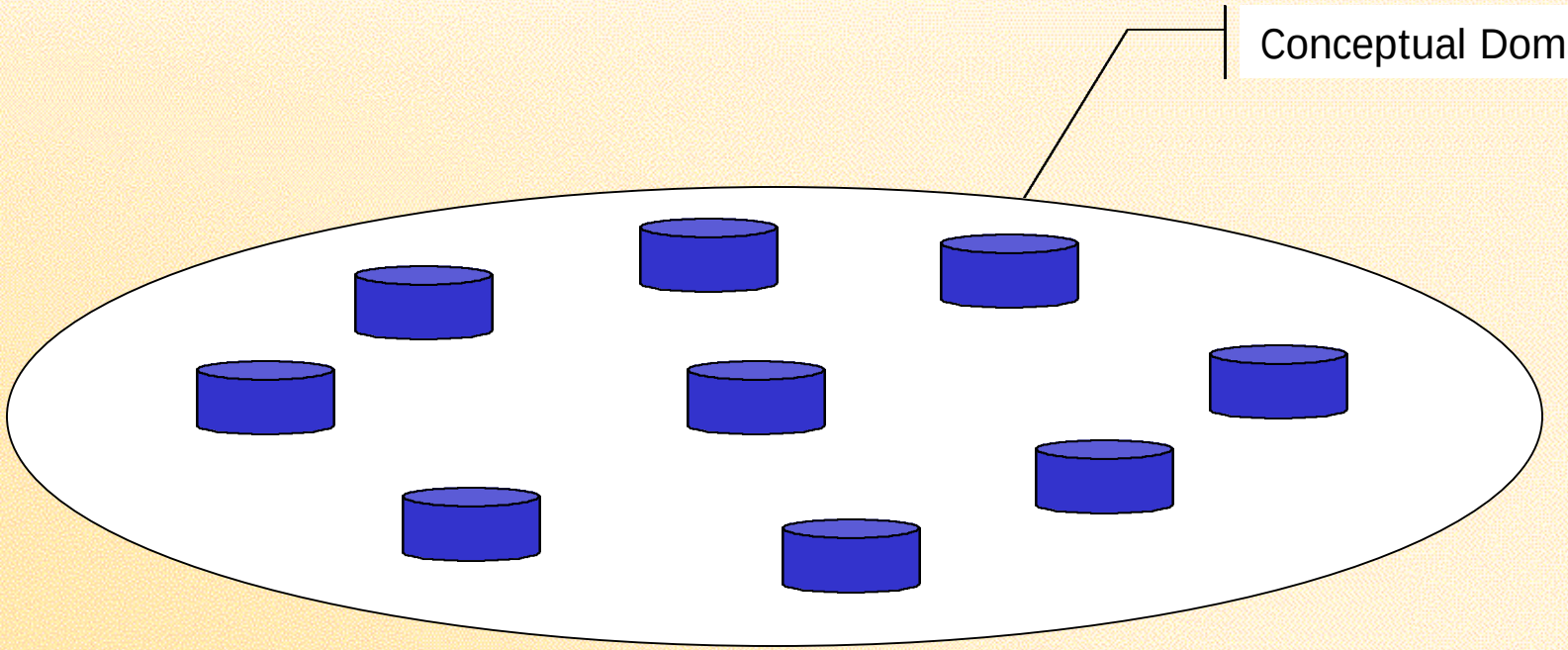
Non-repudiation model



Non-repudiation services

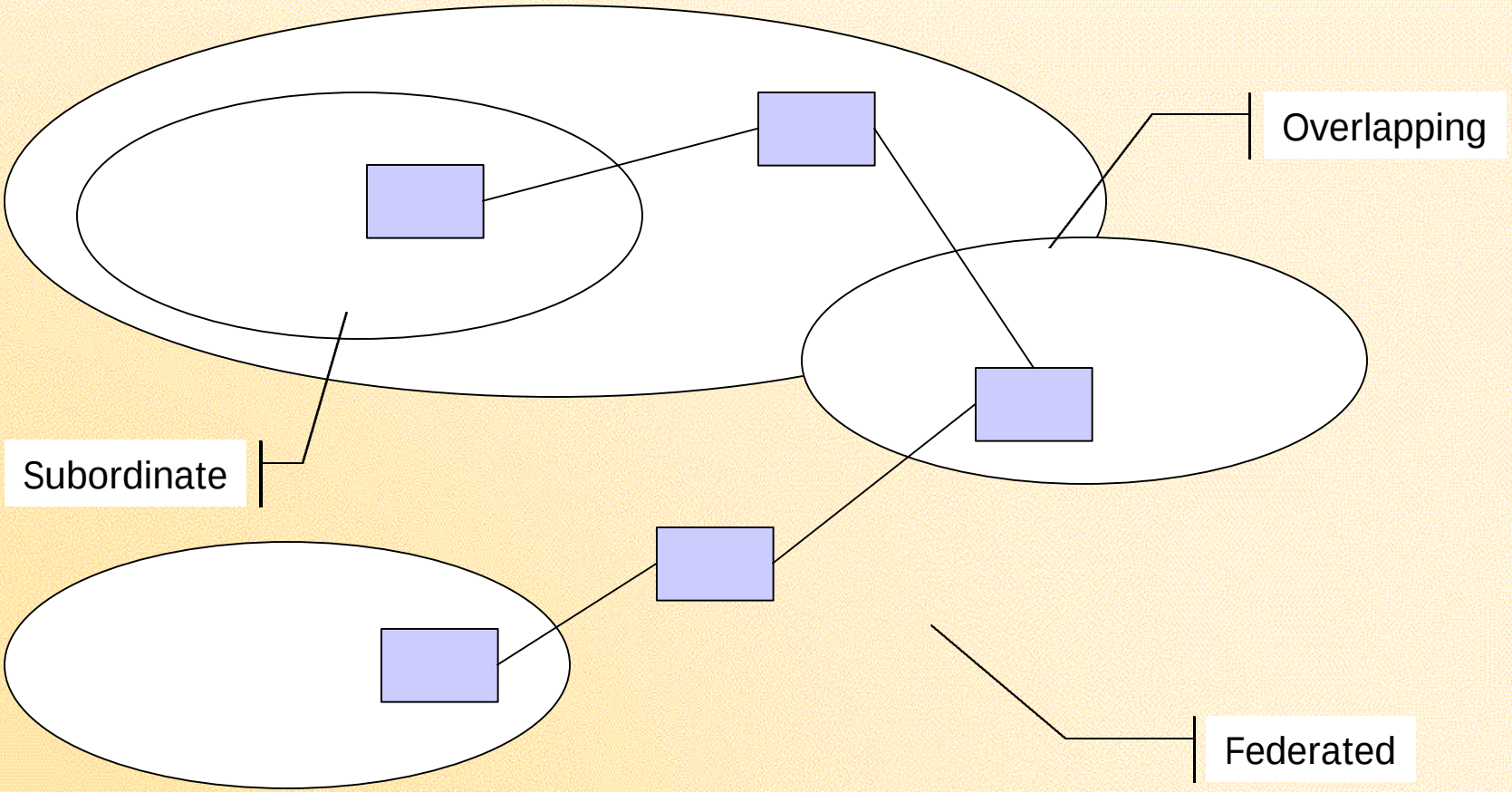
- Generation of evidence of an action.
- Verification of evidence of an action.
- Generation of a request for evidence related to a message sent to a recipient.
- Receipt of a request for evidence related to a message received.
- Analysis of details of evidence of an action.
- Collection of the evidence required for long term storage. In this case, more complete evidence may be needed.

Domain Model



A distinct scope, within which certain common characteristics are exhibited and common rules observed.

Domain hierarchies



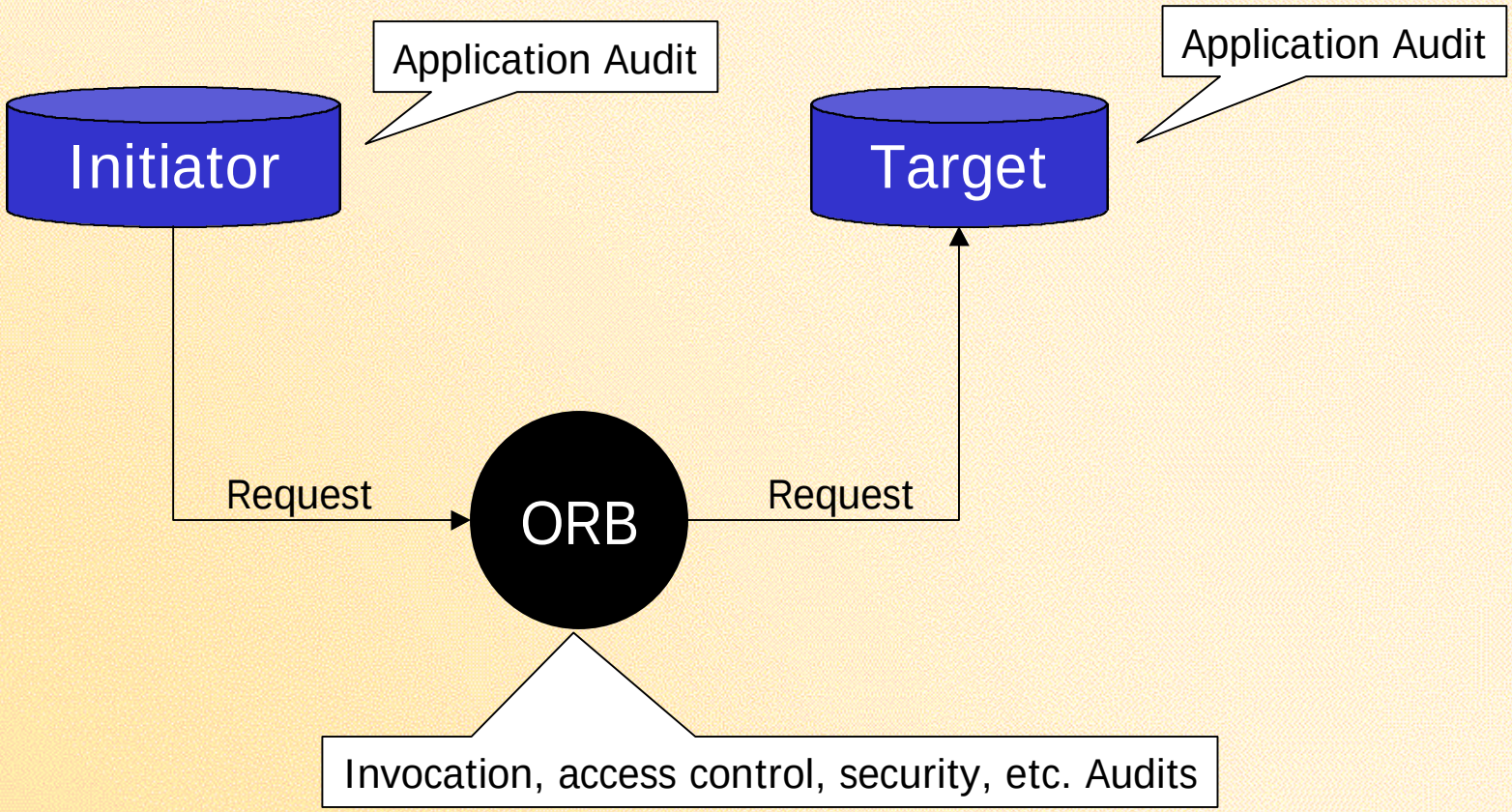
Domain Types

- Security policy domain - The scope over which a security policy is enforced. There may be subdomains for different aspects of this policy.

- Security environment domain - The scope over which the enforcement of a policy may be achieved by some means local to that environment, so does not need to be enforced within the object system. For example, messages will often not need cryptographic protection to achieve the required integrity when being transferred between objects in the same machine.

- Security technology domain - Where common security mechanisms are used to enforce the policies.

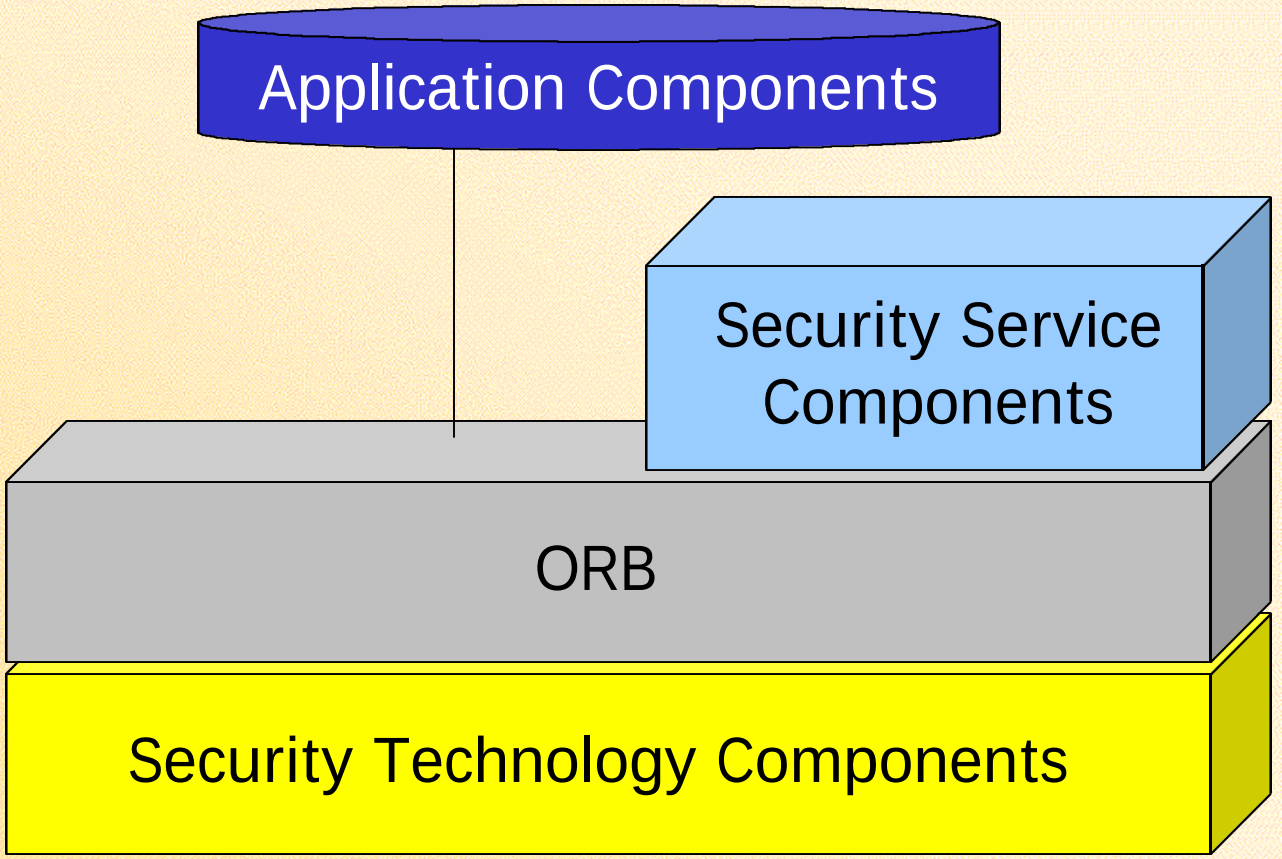
Auditing Model



Auditing Factoids

- Audit policies are used to restrict what types of events to audit under which circumstances to reduce audit record sizes.
- System audit policies are enforced automatically for all applications, even security unaware ones.
- Events can either be recorded on audit trails for later analysis or, if they are deemed to be serious, alarms can be sent to an administrator.
- Application audit trails may be separate from system ones.

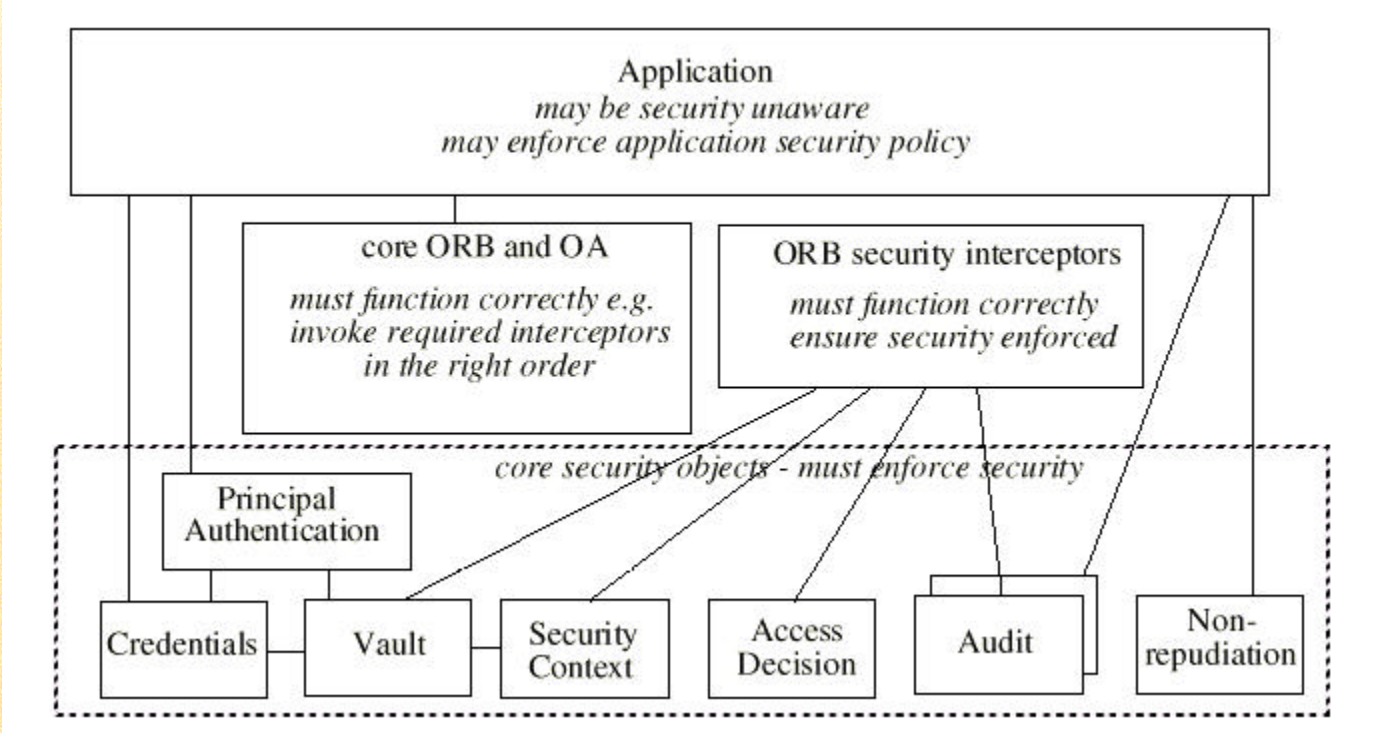
Security implementation



Security Interface

- Via function calls to the security service interface
- Interface and type definitions are fully defined in the OMG standard
- Interface is implementation independent, allowing multiple technology implementations to co-exist, even in the same environment

Security responsibilities



Questions

And possibly answers,
definitely references.

Bibliography

CORBA standard white paper, Chp 15 - Security Services.
November, 1996 version 1.0
http://www.omg.org

The CERF CORBA FAQ
http://www.cerfnet.com/~mpcline/Corba-FAQ/index.html

CORBA on the Web
Ron Ben-Natan
McGraw-Hill ISBN 0-07-006724-4

Contact Information

Walt Smith
wsmith@tekna.com
(804) 217-8888

Tekna
4600 Cox Road, Suite 320
Glen Allen, VA 23060
www.tekna.com