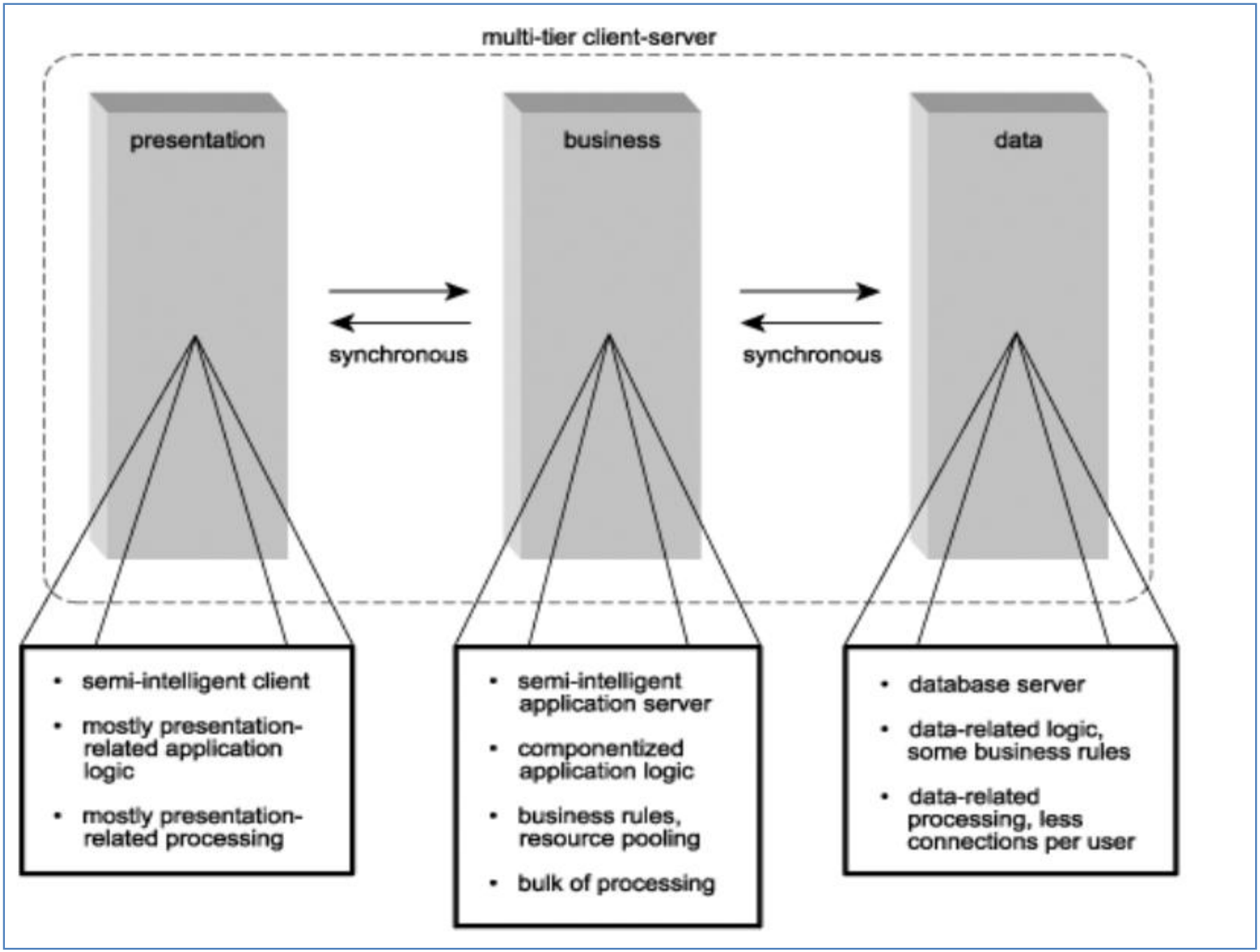
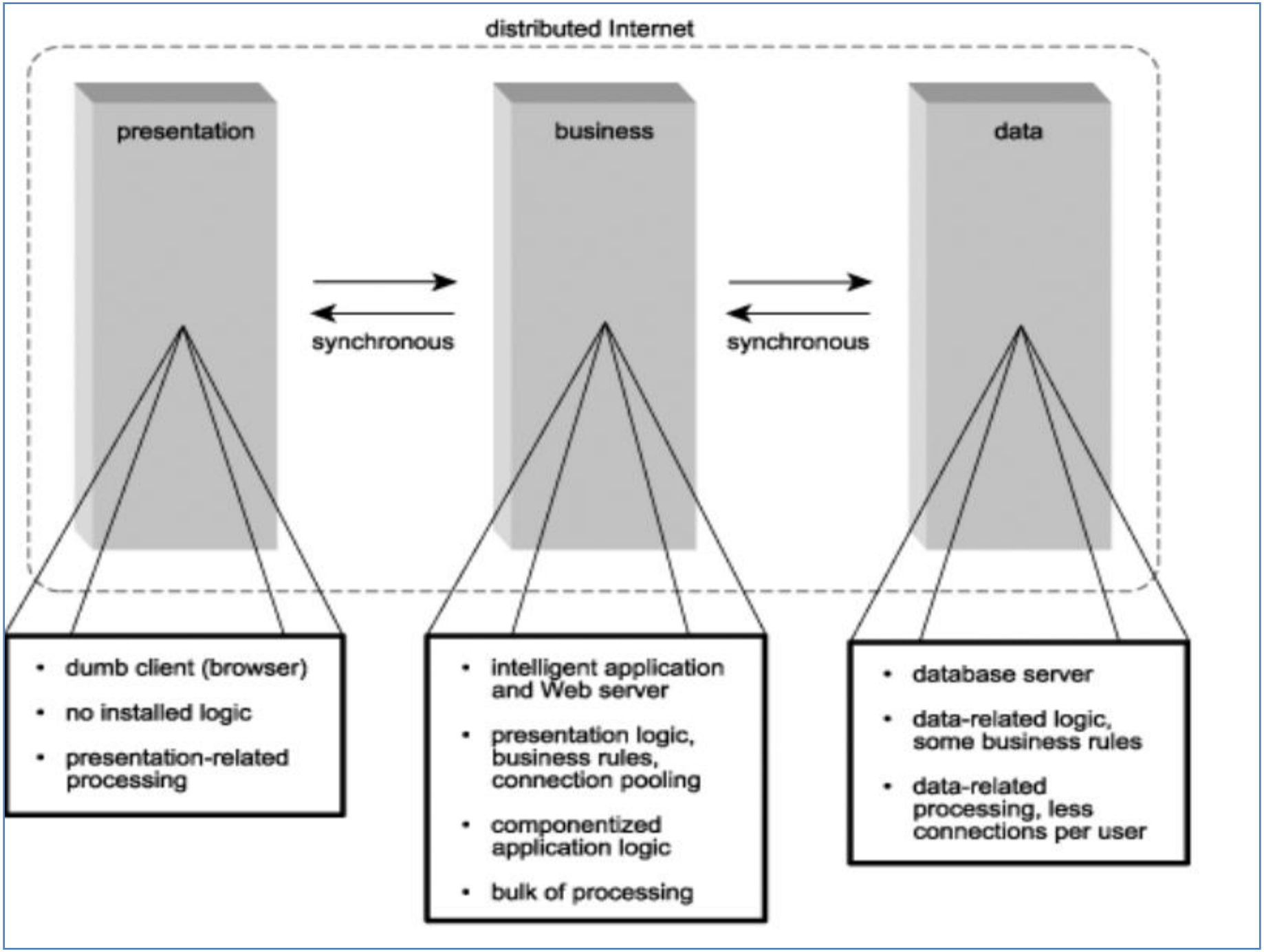
Distributed Internet architecture: a brief history

- In response to the costs and limitations associated with the two-tier client server architecture, the concept of building component-based applications hit the mainstream.
- Multi-tier client-server architectures surfaced, breaking up the monolithic client executable into components designed to varying extents of compliance with object-orientation.


multi-tier client-server
presentation
business
data
synchronous
synchronous
• semi-intelligent client
• mostly presentation-related application logic
• mostly presentation-related processing
• semi-intelligent application server
• componentized application logic
• business rules, resource pooling
• bulk of processing
• database server
• data-related logic, some business rules
• data-related processing, less connections per user


distributed Internet
presentation
business
data
synchronous
synchronous
• dumb client (browser)
• no installed logic
• presentation-related processing
• intelligent application and Web server
• presentation logic, business rules, connection pooling
• componentized application logic
• bulk of processing
• database server
• data-related logic, some business rules
• data-related processing, less connections per user

SOA vs. Distributed Internet Architecture

- Contradiction!
- SOA - distributed Internet architecture

Application logic

- Where the Partitioned units of logic reside?
  - Provider logic resides on servers which is broken down into pieces for **both SOA and DIA**
  - The **difference** is how to application logic is partitioned and how it allows communication
  - **Reuse is common** in both approaches but SOA improves reuse by adding interoperability

Application logic

- How is the application logic partitioned and How does it allow communication?
- **DIA:**
  - Components reside on one or more servers
  - Designed with varying degree of functional granularity
  - Same server components communicate via proprietary APIs
  - Actual references are embedded in code
  - RPC is used for intra server communication

Application logic

- **SOA:**
  - Relies on components, but services encapsulate them in standard way
  - Services are designed based on SO principles
  - Services provide a standard interface to any kind of applications – Legacy or non-legacy
  - SOA implements loose coupling with the use of discovery mechanism
  - Services rely on document style messages with intelligent information

Application Processing

- **DIA:**
  - Promotes use of proprietary communication protocols – DCOM or CORBA
  - Requires active connection , supports both stateful and stateless components
  - Synchronous data exchange is used more often

Application Processing

- **SOA:**
  - Relies on message based communication – Serialization, transmission, deserialization, validation, parsing required
  - Supports autonomous and stateless services
  - Asynchronous communication is encouraged

Technology

- **Distributed Internet architecture**
  - HTML, HTTP, XML, Web services
- **SOA**
  - XML and Web services
- **Difference:**
  - CSOA is built upon XML data representation and Web service platform
  - For DIA these are optional

Security

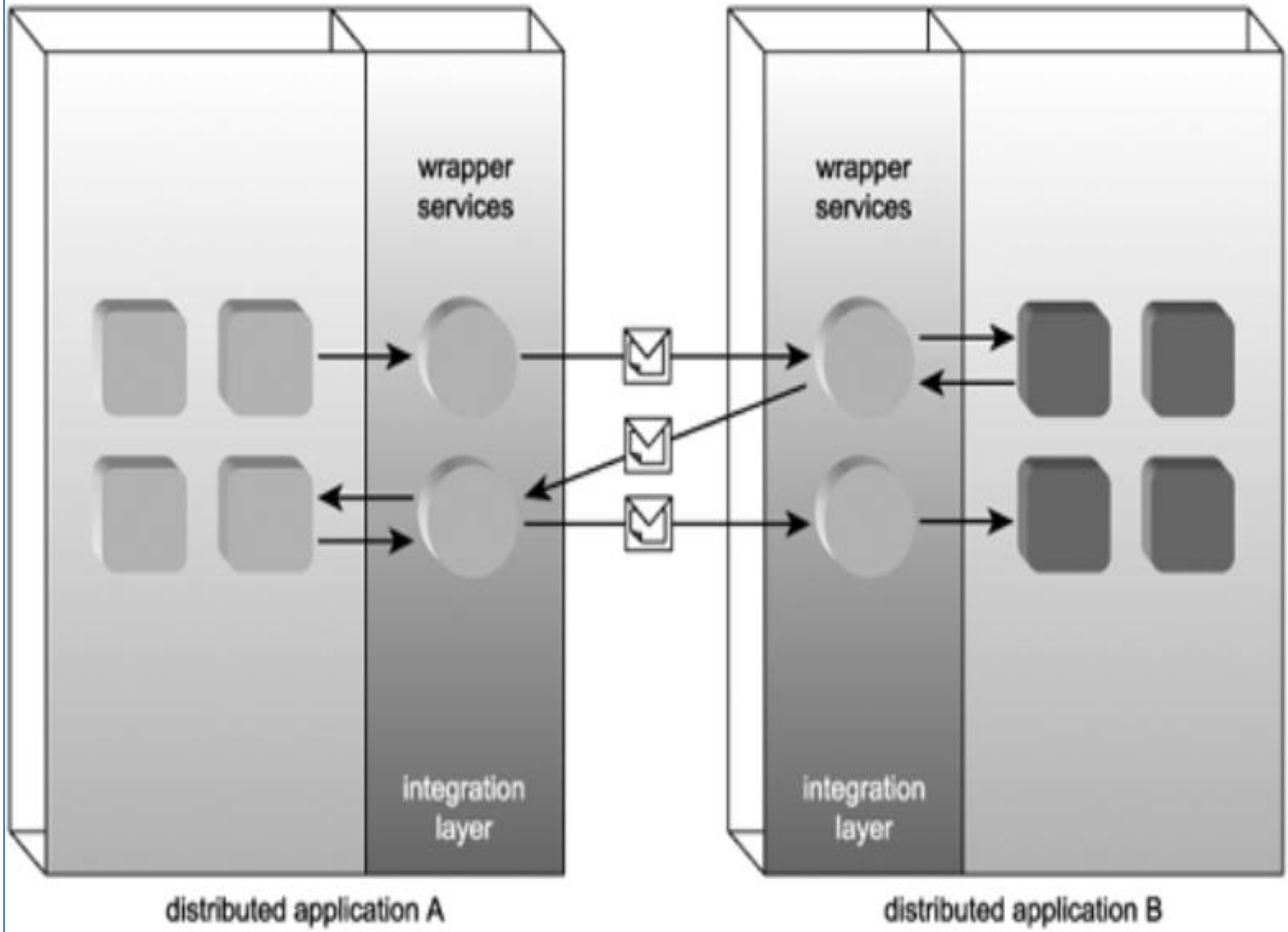
- **Distributed Internet architecture**
  - Delegation and impersonation
  - Encryption
- **SOA**
  - WS-Security framework

Administration

- **DIA**
  - Keep track of component instances
  - Local and remote problem tracking
  - Monitor server demands
  - DB administration
  - Scalability of web server
- **SOA**
  - Above all plus runtime administration
  - Messaging framework problems
  - Maturity of WS-* extensions

SOA vs. hybrid Web service architecture

- Recent variations of Distributed Internet architecture have incorporated Web services
- Web services are used as component wrappers


wrapper
services
wrapper
services
integration
layer
integration
layer
distributed application A
distributed application B

SOA vs. hybrid Web service architecture

- **Hybrid Web service architecture**
  - Does not qualify as a true SOA
    - Use of point to point connections
    - Does not use principles of SOA
- **SOA**
  - Support for a variety of messaging models
  - Service-orientation principles
  - Never limited to point-to-point communication

Service-Orientation and Object-Orientation

- Object-oriented programming is commonly used to build the application logic encapsulated within Web services.
  - Fundamentally OOP differs from service orientation

Service-Orientation and Object-Orientation

- SO is based on design of services.
- OO is centered around creation of objects
- SO emphasizes loose coupling between services.
- OO is based on predefined dependencies of classes / tightly bound objects.
- In SO, scope of units of processing logic (services) can vary.
- In OO, units of logic (objects) tend to be smaller in scope.

Service-Orientation and Object-Orientation

- SO prefers units of processing logic to be as stateless as possible
- OO binds data and logic together, hence it tends to be stateful
- SO supports composition of loosely coupled units of processing logic
- OO supports composition but also encourages inheritance – can lead to tightly coupled dependencies

Common misperceptions about SOA

- "An application that uses Web services is service-oriented"

  – For classifying as being Service Oriented,

    - Web services must be positioned and designed according to Service Orientation Principles

Common misperceptions about SOA

- "SOA is just a marketing term used to re-brand Web services"
  - SOA has become high profile buzzword with the rise of web services.
  - It is just that CSOA implies the use of web service technology to realize SOA principles
  - So web services is just the platform chosen for SOA implementation

Common misperceptions about SOA

- "SOA is just a marketing term used to re-brand distributed computing with Web services"
  - False SOA syndrome
  - SOA has design principles which are related but very different from distributed computing

Common misperceptions about SOA

- "SOA simplifies distributed computing"
  - Transition from theory to in practice is complex
  - Typical SOA implementations require more up-front research than solutions created under previous platform paradigms
  - SOA can only be realized by strictly following principles of service-orientation
    - Quality of simplicity, well integration of composable services, standardized access, etc…

Common misperceptions about SOA

- "An application with Web services that uses WS-* extensions is service-oriented"
  - Just by making WS-* extensions as a part of architecture does not make it service oriented
  - The important aspect is how the architecture itself is designed

Common misperceptions about SOA

- "If you understand Web services you won't have a problem building SOA"
  - A technical and conceptual knowledge of Web services is certainly helpful
  - Fundamental service-orientation principles are pretty much technology agnostic
  - The manner in which Web services are utilized in SOA is significantly different
  - It is best to assume that realizing contemporary SOA requires a separate skill set that goes beyond a knowledge of Web services technology

Common misperceptions about SOA

- "Once you go SOA, everything becomes interoperable"
  - Though this ultimate goal is attainable, it requires investment, analysis, and above all, a high degree of standardization

Common benefits of SOA

- **Improved integration (and intrinsic interoperability)**
  - SOA can result in the creation of solutions that consist of inherently interoperable services
  - Vendor-neutral communications framework established by Web services-driven SOAs + highly standardized service descriptions + message structures
  - The cost and effort of cross-application integration is significantly lowered when applications being integrated are SOA-compliant

Common benefits of SOA

- **Inherent reuse**
  - Service-orientation promotes the design of services that are inherently reusable
  - fulfill immediate application-level requirements + supporting a degree of reuse by future potential requestors
  - Building services to be inherently reusable results in a moderately increased development effort and requires the use of design standards
  - Subsequently leveraging reuse within services lowers the cost and effort of building service-oriented solutions

Common benefits of SOA

- **Streamlined architectures and solutions**
  - The concept of composition is fundamental part of SOA → highly optimized automation environments
  - Adherence to design standards that govern allowable extensions within each application environment
    - Reduced processing overhead and reduced skill-set requirements

Common benefits of SOA

- **Leveraging the legacy investment**
  - The Web services technology set has spawned a large adapter market
    - To work towards a state of federation
  - The cost and effort of integrating legacy and contemporary solutions is lowered.
  - The need for legacy systems to be replaced is potentially lessened

Common benefits of SOA

- **Focused investment on communications infrastructure**
  - Web services establish a common communications framework
    - SOA can centralize inter-application and intra-application communication as part of standard IT infrastructure

Common benefits of SOA

- **"Best-of-breed" alternatives**
  - IT departments are frequently required to push back and limit or even reject requests to alter or expand upon existing automation solutions
  - SOA is expected to increase empowerment of both business and IT communities
    - "best-of-breed" technology + vendor-neutral communications framework
  - The potential scope of business requirement fulfillment increases, as does the quality of business automation

Common benefits of SOA

- **Organizational agility**
  - Much of service-orientation is based on the assumption that what you build today will evolve over time
  - Change has became the norm in distributed solution design
    - Qualities such as reuse and interoperability become commonplace
    - The predictability of these qualities within the enterprise leads to a reliable level of organizational agility.

Common benefits of SOA

- **Organizational agility**
  - The cost and effort to respond and adapt to business or technology-related change is reduced

Common pitfalls of adopting SOA

- **Building service-oriented architectures like traditional distributed architectures**
  - The number one obstacle
    - Due to various misperceptions
    - RPC Style descriptions
    - Improper partitioning
    - Non-composability
    - Non-standardization

Common pitfalls of adopting SOA

- **Not standardizing SOA**
  - Due to concurrent project development, many custom standards are designed for different projects
  - Future integration can be costly and fragile

Common pitfalls of adopting SOA

- **Not creating a transition plan**
  - Proper migration should be planned when moving to SOA

Common pitfalls of adopting SOA

- **Not starting with an XML foundation architecture**
  - In the world of contemporary SOA, everything, in fact, begins with XML
  - Proper implementation of a persistent XML data representation layer within SOAs is must
  - Standardizing XML technologies lay the groundwork for a robust, optimized, and interoperable SOA

Common pitfalls of adopting SOA

- **Not understanding SOA performance requirements**
  - Environments begin experiencing processing latency
    - Contemporary SOA introduces layers of data processing

Common pitfalls of adopting SOA

- **Not understanding Web services security**
  - SSL: Developers' choice
    - Not the technology of choice for SOA
  - Message-level security is important

Common pitfalls of adopting SOA

- **Not keeping in touch with product platforms and standards development**
  - A transition to SOA alters everything