The Roots of Service-Oriented Architecture

Slides are based on the book:
"Service-Oriented Architecture: Concepts, Technology, and Design"
By Thomas Erl,
Publisher: Prentice Hall PTR

Comparing SOA to past architectures

Application architecture

- Application architecture is to an application development team what a blueprint is to a team of construction workers.
- Different organizations document different levels of application architecture.
- Some keep it high-level, providing abstract physical and logical representations of the technical blueprint.
- Others include more detail, such as
  - common data models,
  - communication flow diagrams,
  - application-wide security requirements, and
  - aspects of infrastructure.

Application architecture

- It is not uncommon for an organization to have several application architectures. A single architecture document typically represents a distinct solution environment.
  - For example, an organization that houses both .NET and J2EE solutions would very likely have separate application architecture specifications for each.
- A key part of any application-level architecture is that it reflects immediate solution requirements, as well as long-term, strategic IT goals.
- It is for this reason that when multiple application architectures exist within an organization, they are almost always accompanied by and kept in alignment with a governing enterprise architecture.

Enterprise architecture

- In larger IT environments, the need to control and direct IT infrastructure is critical.
- When numerous, disparate application architectures co-exist and sometimes even integrate, the demands on the underlying hosting platforms can be complex and onerous.
- Therefore, it is common for a master specification to be created, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure.
- An enterprise architecture specification is to an organization what an urban plan is to a city.
  - Therefore, the relationship between an urban plan and the blueprint of a building are comparable to that of enterprise and application architecture specifications.

Enterprise architecture

- Typically, changes to enterprise architectures directly affect application architectures, which is why architecture specifications often are maintained by the same group of individuals. Further, enterprise architectures often contain a long-term vision of how the organization plans to evolve its technology and environments.
  - For example, the goal of phasing out an outdated technology platform may be established in this specification.
- Finally, this document also may define the technology and policies behind enterprise-wide security measures. However, these often are isolated into a separate security architecture specification.

Service-oriented architecture

- SOA spans both enterprise and application architecture domains.
- The benefit potential offered by SOA can only be truly realized when applied across multiple solution environments.
- This is where the investment in building reusable and interoperable services based on a vendor-neutral communications platform can fully be leveraged.
- This does not mean that the entire enterprise must become service-oriented. SOA belongs in those areas that have the most to gain from the features and characteristics it introduces.

Service-oriented architecture

- An SOA can refer to an application architecture or the approach used to standardize technical architecture across the enterprise.
- Because of the composable nature of SOA (meaning that individual application-level architectures can be comprised of different extensions and technologies), it is absolutely possible for an organization to have more than one SOA.
- The Web services platform offers one of a number of available forms of implementation for SOA.
- Other approaches, such as those provided by traditional distributed platforms, also exist.
- An important aspect of the terminology used is that our use of the term "SOA" implies the contemporary SOA model (based on Web services and service-orientation principles)

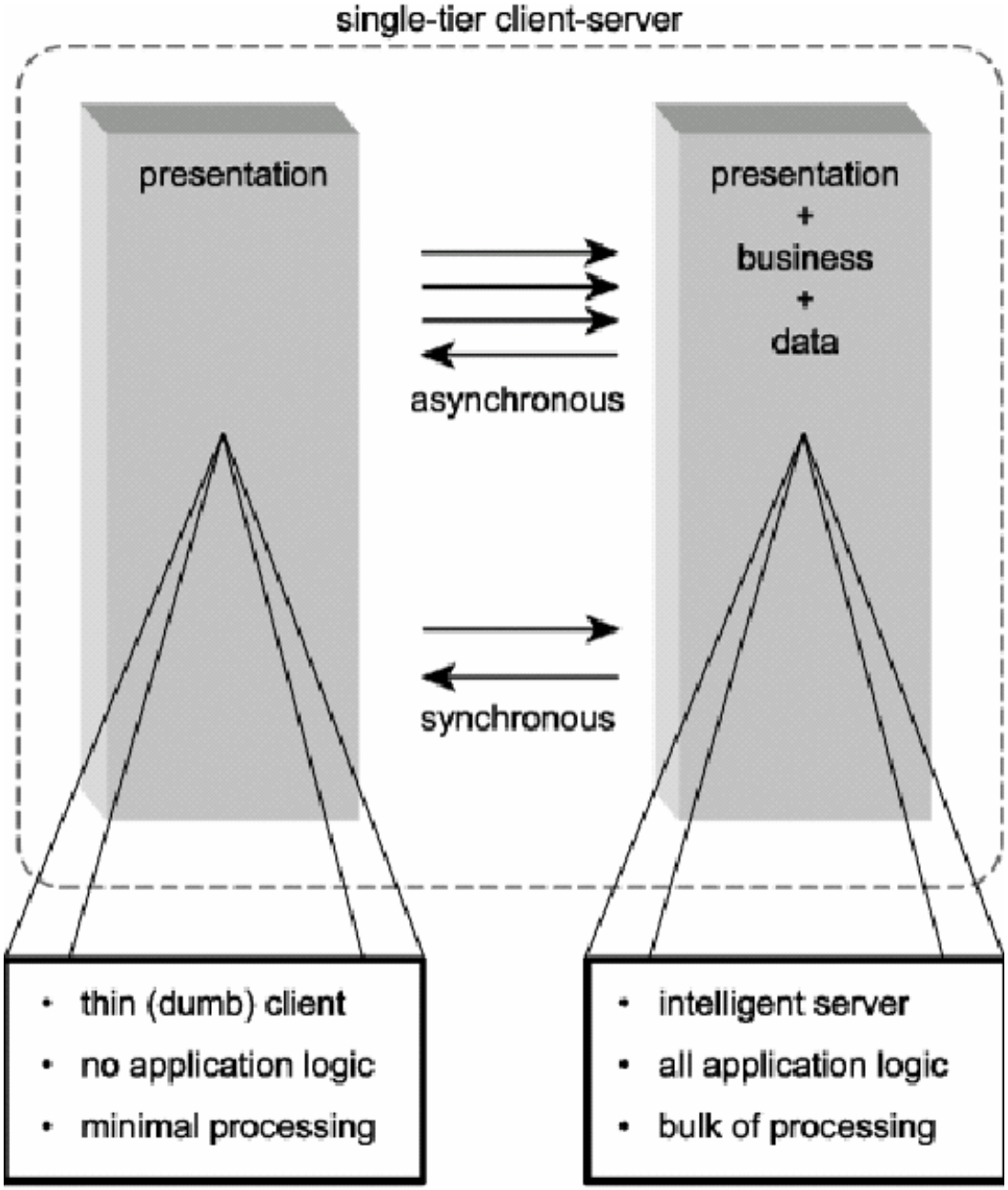
SOA vs. client-server architecture

- Just about any environment in which one piece of software requests or receives information from another can be referred to as "client-server.“
- Pretty much every variation of application architecture that ever existed (including SOA) has an element of client-server interaction in it.
- However, the industry term "client-server architecture" generally refers to a particular generation of early environments during which the client and the server played specific roles and had distinct implementation characteristics.

Client-server architecture

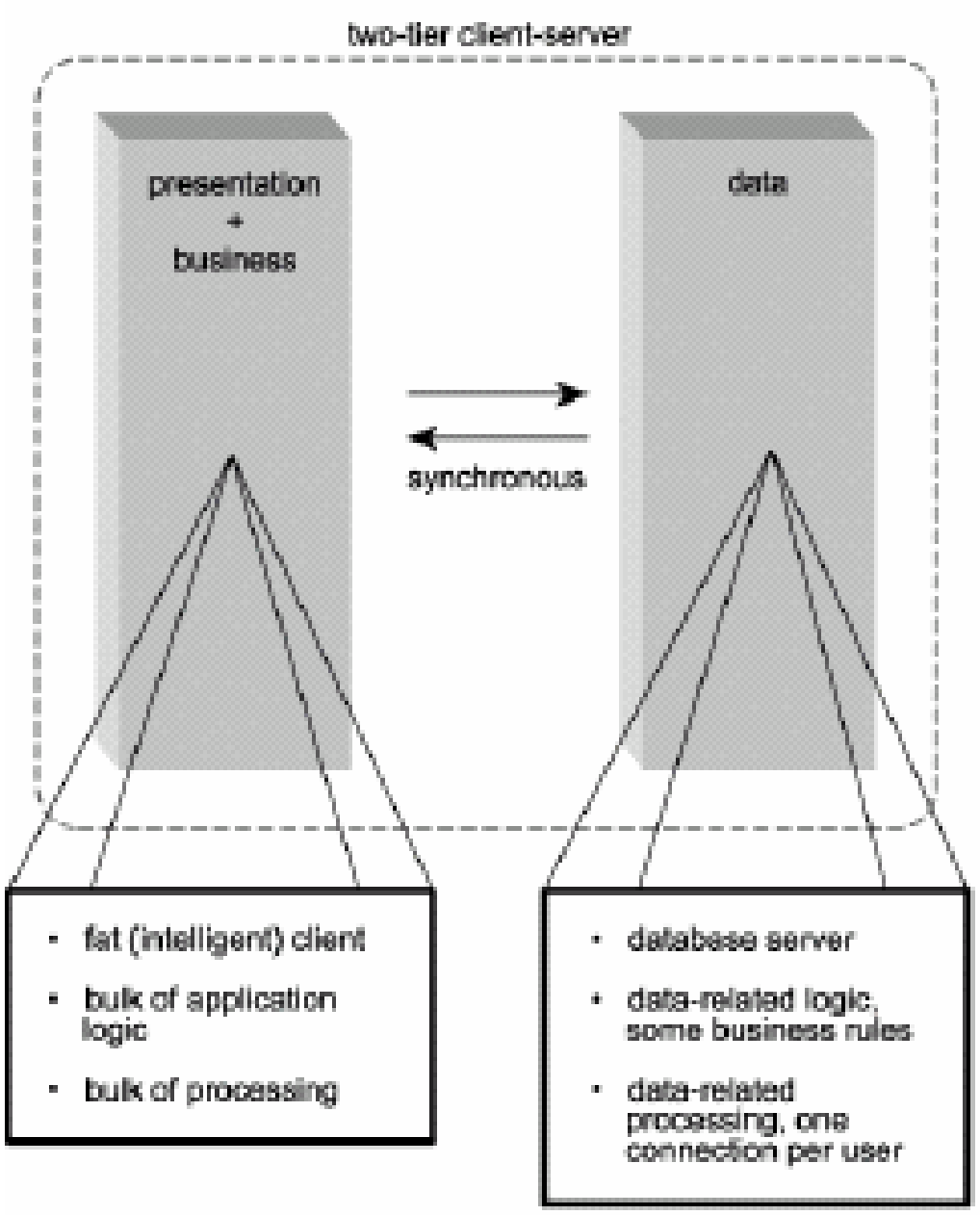
A brief history

- The original monolithic mainframe systems that empowered organizations to get seriously computerized often are considered the first inception of client-server architecture.
- These environments, in which bulky mainframe back-ends served thin clients, are considered an implementation of the single-tier client-server architecture.


single-tier client-server
presentation
presentation
+
business
+
data
asynchronous
synchronous
• thin (dumb) client
• no application logic
• minimal processing
• intelligent server
• all application logic
• bulk of processing

Client-server architecture

- Mainframe systems natively supported both synchronous and asynchronous communication. The latter approach was used primarily to allow the server to continuously receive characters from the terminal in response to individual key-strokes. Only upon certain conditions would the server actually respond.
- While its legacy still remains, the reign of the mainframe as the foremost computing platform began to decline when a two-tier variation of the client-server design emerged in the late 80s.
- This new approach introduced the concept of delegating logic and processing duties onto individual workstations, resulting in the birth of the fat client. Further supported by the innovation of the graphical user-interface (GUI), two-tier client-server was considered a huge step forward and went on to dominate the IT world for years during the early 90s.
- The common configuration of this architecture consisted of multiple fat clients, each with its own connection to a database on a central server. Client-side software performed the bulk of the processing, including all presentation-related and most data access logic. One or more servers facilitated these clients by hosting scalable RDBMSs.


two-tier client-server
presentation
+
business
data
synchronous
• fat (intelligent) client
• bulk of application logic
• bulk of processing
• database server
• data-related logic, some business rules
• data-related processing, one connection per user

Client-server architecture: Application logic

- Client-server environments place the majority of application logic into the client software.
- This results in a monolithic executable that controls the user experience, as well as the back-end resources. One exception is the distribution of business rules.
- A popular trend was to embed and maintain business rules relating to data within stored procedures and triggers on the database.
- This somewhat abstracted a set of business logic from the client and simplified data access programming. Overall, though, the client ran the show.

Client-server architecture: Application logic

- The presentation layer within contemporary service-oriented solutions can vary.
- Any piece of software capable of exchanging SOAP messages according to required service contracts can be classified as a service requestor.
- While it is commonly expected for requestors to be services as well, presentation layer designs are completely open and specific to a solution's requirements.

Client-server architecture: Application logic

- Within the server environment, options exist as to where application logic can reside and how it can be distributed. These options do not preclude the use of database triggers or stored procedures.
- However, service-oriented design principles come into play, often dictating the partitioning of processing logic into autonomous units. This facilitates specific design qualities, such as service statelessness and interoperability, as well as future composability and reusability.
- Additionally, it is more common within an SOA for these units of processing logic to be solution-agnostic. This supports the ultimate goal of promoting reuse and loose coupling across application boundaries.

Client-server architecture: Application processing

- Because most client-server application logic resides in the client component, the client workstation is responsible for the bulk of the processing.
- The 80/20 ratio often is used as a rule of thumb, with the database server typically performing twenty percent of the work.
- Despite that, though, it is the database that frequently becomes the performance bottleneck in these environments.

Client-server architecture: Application processing

- A two-tier client-server solution with a large user-base generally requires that each client establish its own database connection. Communication is predictably synchronous, and these connections are often persistent (meaning that they are generated upon user login and kept active until the user exits the application).
- Proprietary database connections are expensive, and the resource demands sometimes overwhelm database servers, imposing processing latency on all users.
- Additionally, given that the clients are assigned the majority of processing responsibilities, they too often demand significant resources. Client-side executables are fully stateful and consume a steady chunk of PC memory.
- User workstations therefore often are required to run client programs exclusively so that all available resources can be offered to the application.

Client-server architecture: Application processing

- Processing in SOA is highly distributed.
- Each service has an explicit functional boundary and related resource requirements. In modeling a technical service-oriented architecture, you have many choices as to how you can position and deploy services.
- Enterprise solutions consist of multiple servers, each hosting sets of Web services and supporting middleware. There is, therefore, no fixed processing ratio for SOAs.
- Services can be distributed as required, and performance demands are one of several factors in determining the physical deployment configuration.
- Communication between service and requestor can be synchronous or asynchronous.
- This flexibility allows processing to be further streamlined, especially when asynchronous message patterns are utilized. Additionally, by placing a large amount of intelligence into the messages, options for achieving message-level context management are provided.
- This promotes the stateless and autonomous nature of services and further alleviates processing by reducing the need for runtime caching of state information.

Client-server architecture: Technology

- The emergence of client-server applications promoted the use of 4GL programming languages.
- Traditional 3GL languages, such as C++, were also still used, especially for solutions that had more rigid performance requirements.
- The technology set used by SOA actually has not changed as much as it has expanded. Newer versions of older programming languages, such as Visual Basic, still can be used to create Web services, and the use of relational databases still is commonplace.
- The technology landscape of SOA, though, has become increasingly diverse. In addition to the standard set of Web technologies (HTML, CSS, HTTP, etc.) contemporary SOA brings with it the absolute requirement that an XML data representation architecture be established, along with a SOAP messaging framework, and a service architecture comprised of the ever-expanding Web services platform.

Client-server architecture: Security

- Besides the storage and management of data and the business rules embedded in stored procedures and triggers, the one other part of client-server architecture that frequently is centralized at the server level is security.
- Databases are sufficiently sophisticated to manage user accounts and groups and to assign these to individual parts of the physical data model.
- Security also can be controlled within the client executable, especially when it relates to specific business rules that dictate the execution of application logic (such as limiting access to a part of a user-interface to select users).
- Additionally, operating system-level security can be incorporated to achieve a single sign-on, where application clearance is derived from the user's operating system login account information.

Client-server architecture: Security

- Though one could boast about the advantages of SOA, most architects envy the simplicity of client-server security.
- Corporate data is protected via a single point of authentication, establishing a single connection between client and server. In the distributed world of SOA, this is not possible.
- Security becomes a significant complexity directly relational to the degree of security measures required. Multiple technologies are typically involved, many of which comprise the WS-Security framework

Client-server architecture: Administration

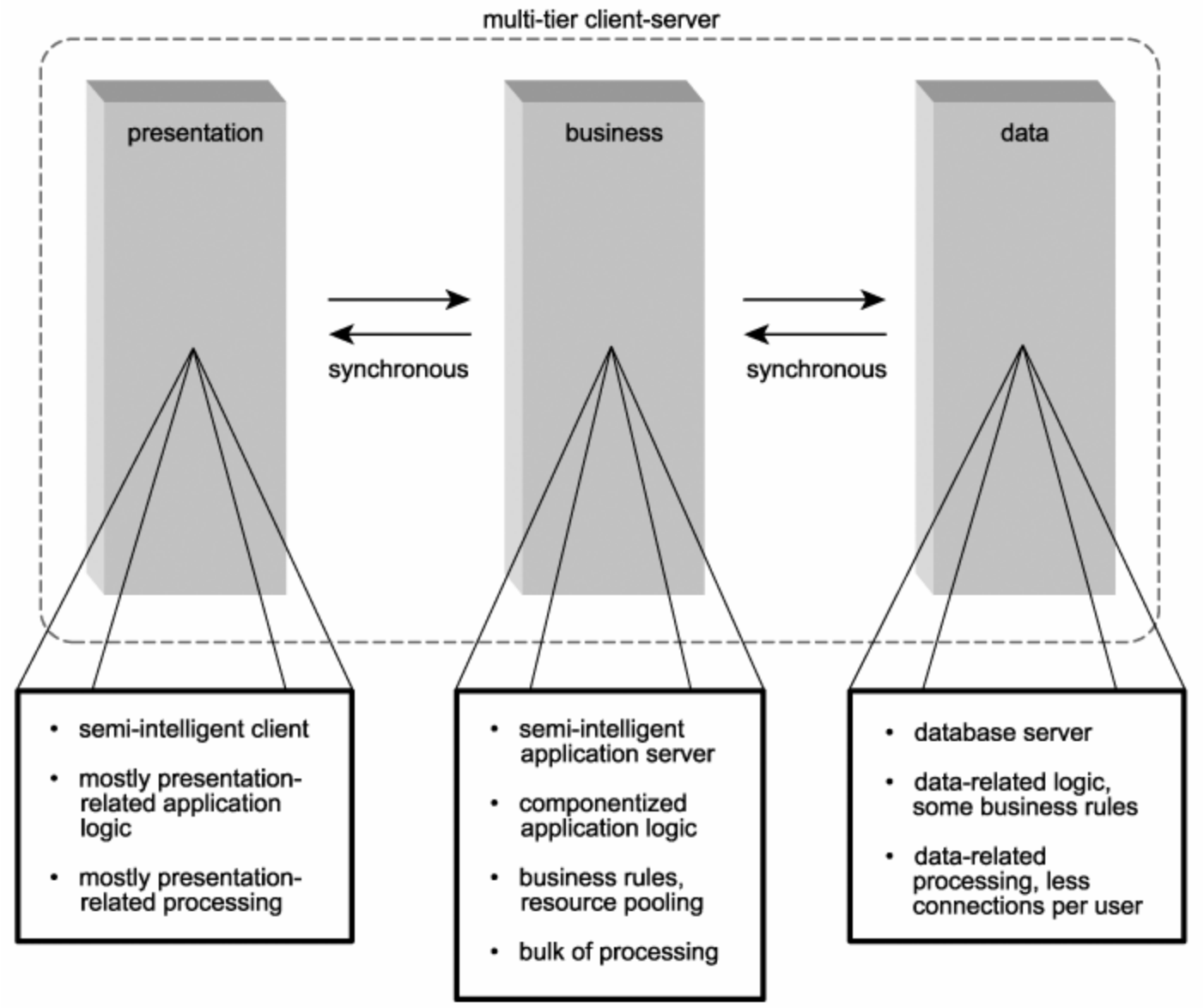
- One of the main reasons the client-server era ended was the increasingly large maintenance costs associated with the distribution and maintenance of application logic across user workstations.
- Because each client housed the application code, each update to the application required a redistribution of the client software to all workstations. In larger environments, this resulted in a highly burdensome administration process.
- Maintenance issues spanned both client and server ends.
- Client workstations were subject to environment-specific problems because different workstations could have different software programs installed or may have been purchased from different hardware vendors.
- Further, there were increased server-side demands on databases, especially when a client-server application expanded to a larger user base.

Client-server architecture: Administration

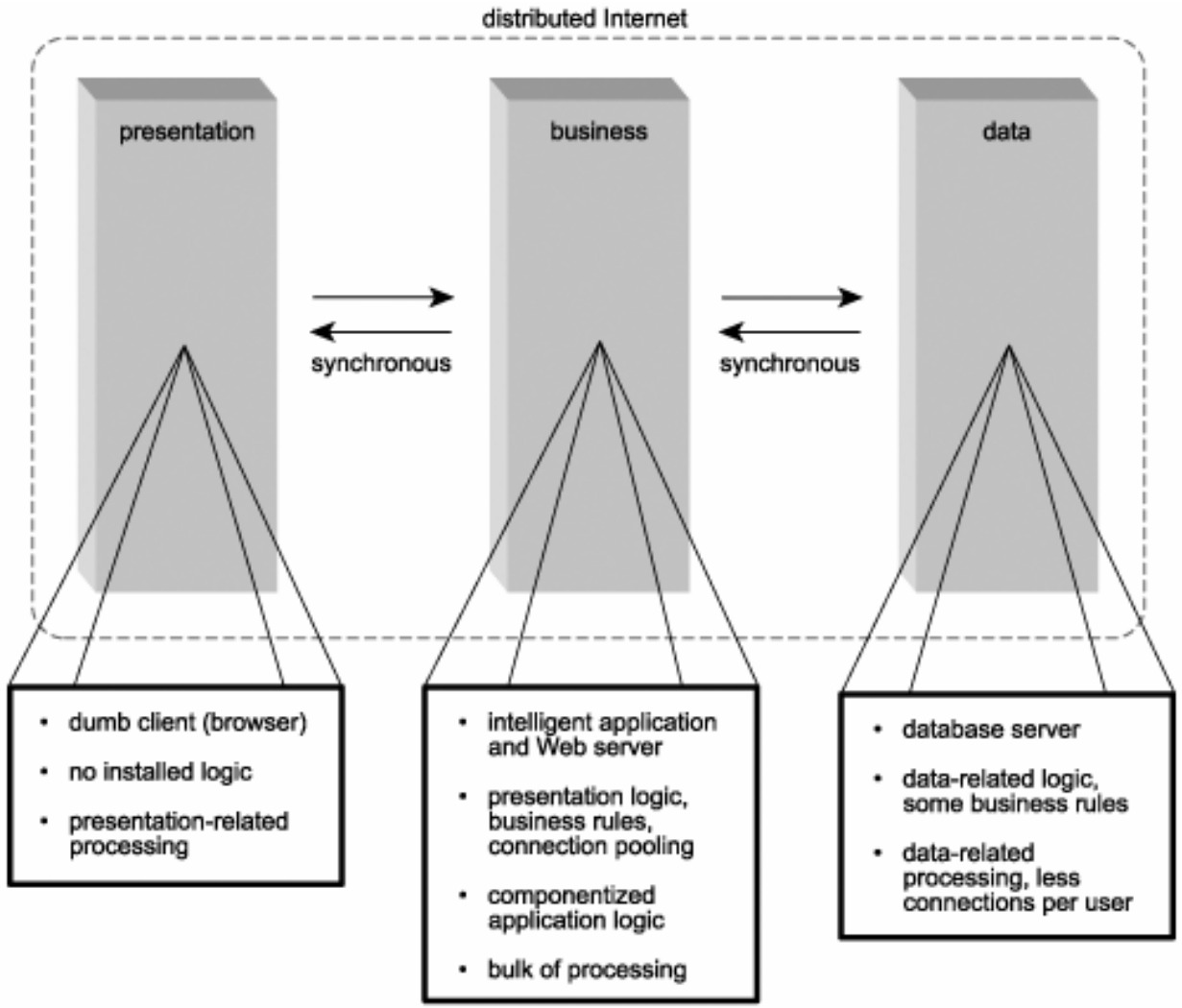
- Because service-oriented solutions can have a variety of requestors, they are not necessarily immune to client-side maintenance challenges.
- While their distributed back-end does accommodate scalability for application and database servers, new administration demands can be introduced.
- For example, once SOAs evolve to a state where services are reused and become part of multiple service compositions, the management of server resources and service interfaces can require powerful administration tools, including the use of a private registry.
- *Refer: Case Study 02*

SOA vs. Distributed Internet architecture

- Given that SOA can be viewed as a form of distributed Internet architecture and because we established earlier that previous types of distributed architecture also could be designed as SOAs.
- Though possible, and although there are distributed environments in existence that may have been heavily influenced by service-oriented principles, this variation of SOA is still a rarity.
- Consider the comparison provided here as one that contrasts traditional distributed Internet architecture in the manner it was most commonly designed.


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

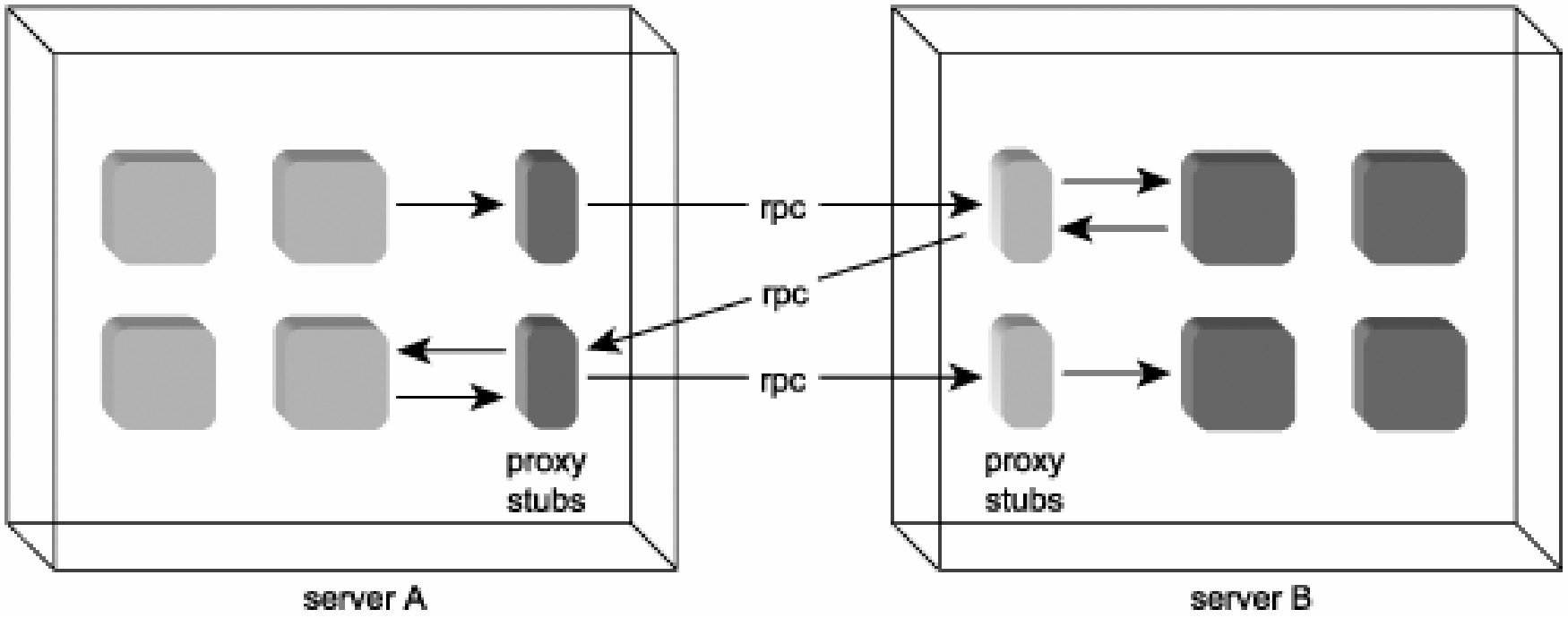
A typical distributed Internet architecture



Distributed Internet architecture
Application logic

- Except for some rare applications that embed proprietary extensions in browsers, distributed Internet applications place all of their application logic on the server side.
- Even client-side scripts intended to execute in response to events on a Web page are downloaded from the Web server upon the initial HTTP request. With none of the logic existing on the client workstation, the entire solution is centralized.
- The emphasis is therefore on:
  - how application logic should be partitioned
  - where the partitioned units of processing logic should reside
  - how the units of processing logic should interact

Components rely on proxy stubs for remote communication



Distributed Internet architecture

Application logic

- Another significant shift related to the design and behavior of distributed application logic is in how services exchange information.
- While traditional components provide methods that, once invoked, send and receive parameter data, Web services communicate with SOAP messages.
- Even though SOAP supports RPC-style message structures, the majority of service-oriented Web service designs rely on document-style messages.
- Also messages are structured to be as self-sufficient as possible. Through the use of SOAP headers, message contents can be accompanied by a wide range of meta information, processing instructions, and policy rules.
- In comparison to data exchange in the pure component world, the messaging framework used by SOA is more sophisticated, bulkier, and tends to result in less individual transmissions.
- Finally, although reuse is also commonly emphasized in traditional distributed design approaches, SOA fosters reuse and cross-application interoperability on a deep level by promoting the creation of solution-agnostic services.

Distributed Internet architecture

Application processing

- SOA, on the other hand, relies on message-based communication. This involves the serialization, transmission, and deserialization of SOAP messages containing XML document payloads.
- Processing steps can involve the conversion of relational data into an XML-compliant structure, the validation of the XML document prior and subsequent to transmission, and the parsing of the document and extraction of the data by the recipient.
- Although advancements, such as the use of enterprise parsers and hardware accelerators are on-going, most still rank RPC communication as being noticeably faster than SOAP.
- Because a network of SOAP servers can effectively replace RPC-style communication channels within service-oriented application environments, the incurred processing overhead becomes a significant design issue.
- Document and message modeling conventions and the strategic placement of validation logic are important factors that shape the transport layer of service-oriented architecture.

Distributed Internet architecture

Application processing

- This messaging framework promotes the creation of autonomous services that support a wide range of message exchange patterns.
- Though synchronous communication is fully supported, asynchronous patterns are encouraged, as they provide further opportunities to optimize processing by minimizing communication.
- Further supporting the statelessness of services are various context management options that can be employed, including the use of WS-* specifications, such as WS-Coordination and WS-BPEL, as well as custom solutions.

Distributed Internet architecture
Technology

- The technology behind distributed Internet architecture went through a number of stages over the past few years. Initial architectures consisted of components, server-side scripts, and raw Web technologies, such as HTML and HTTP. Improvements in middleware allowed for increased processing power and transaction control.
- The emergence of XML introduced sophisticated data representation that actually gave substance to content transmitted via Internet protocols.
- The subsequent availability of Web services allowed distributed Internet applications to cross proprietary platform boundaries.

Distributed Internet architecture
Technology

- Because many current distributed applications use XML and Web services, there may be little difference between the technology behind these solutions and those based on SOA. One clear distinction, though, is that a contemporary SOA will most likely be built upon XML data representation and the Web services technology platform.
- Beyond a core set of Internet technologies and the use of components, there is no governance of the technology used by traditional Internet applications.
- Thus XML and Web services are optional for distributed Internet architecture but not for contemporary SOA.

Distributed Internet architecture

Security

- When application logic is strewn across multiple physical boundaries, implementing fundamental security measures such as authentication and authorization becomes more difficult.
- In a two-tiered client-server environment, an exclusive server-side connection easily facilitates the identification of users and the safe transportation of corporate data. However, when the exclusivity of that connection is removed, and when data is required to travel across different physical layers, new approaches to security are needed.
- To ensure the safe transportation of information and the recognition of user credentials, while preserving the original security context, traditional security architectures incorporate approaches such as delegation and impersonation.
- Encryption also is added to the otherwise wide open HTTP protocol to allow data to be protected during transmission beyond the Web server.

Distributed Internet architecture

Security

- SOAs depart from this model by introducing wholesale changes to how security is incorporated and applied.
- Relying heavily on the extensions and concepts established by the WS-Security framework, the security models used within SOA emphasize the placement of security logic onto the messaging level. SOAP messages provide header blocks in which security logic can be stored.
- That way, wherever the message goes, so does its security information. This approach is required to preserve individual autonomy and loose coupling between services, as well as the extent to which a service can remain fully stateless.

Distributed Internet architecture
Administration

- Maintaining component-based applications involves keeping track of individual component instances, tracing local and remote communication problems, monitoring server resource demands, and, of course, the standard database administration tasks.
- Distributed Internet architecture further introduces the Web server and with it an additional physical environment that requires attention while solutions are in operation. Because clients, whether local or external to an organization, connect to these solutions using HTTP, the Web server becomes the official first point of contact.
- It must therefore be designed for scalability requirement that has led to the creation of Web server farms that pool resources.

Distributed Internet architecture

Administration

- Enterprise-level SOAs typically require additional runtime administration. Problems with messaging frameworks (especially when working with asynchronous exchange patterns) can more easily go undetected than with RPC-based data exchanges. This is because so many variations exist as to how messages can be interchanged. RPC communication generally requires a response from the initiating component, indicating success or failure.
- Upon encountering a failure condition, an exception handling routine kicks in. Exception handling with messaging frameworks can be more complex and less robust. Although WS-* extensions are being positioned to better deal with these situations, administration effort is still expected to remain high.

Distributed Internet architecture
Administration

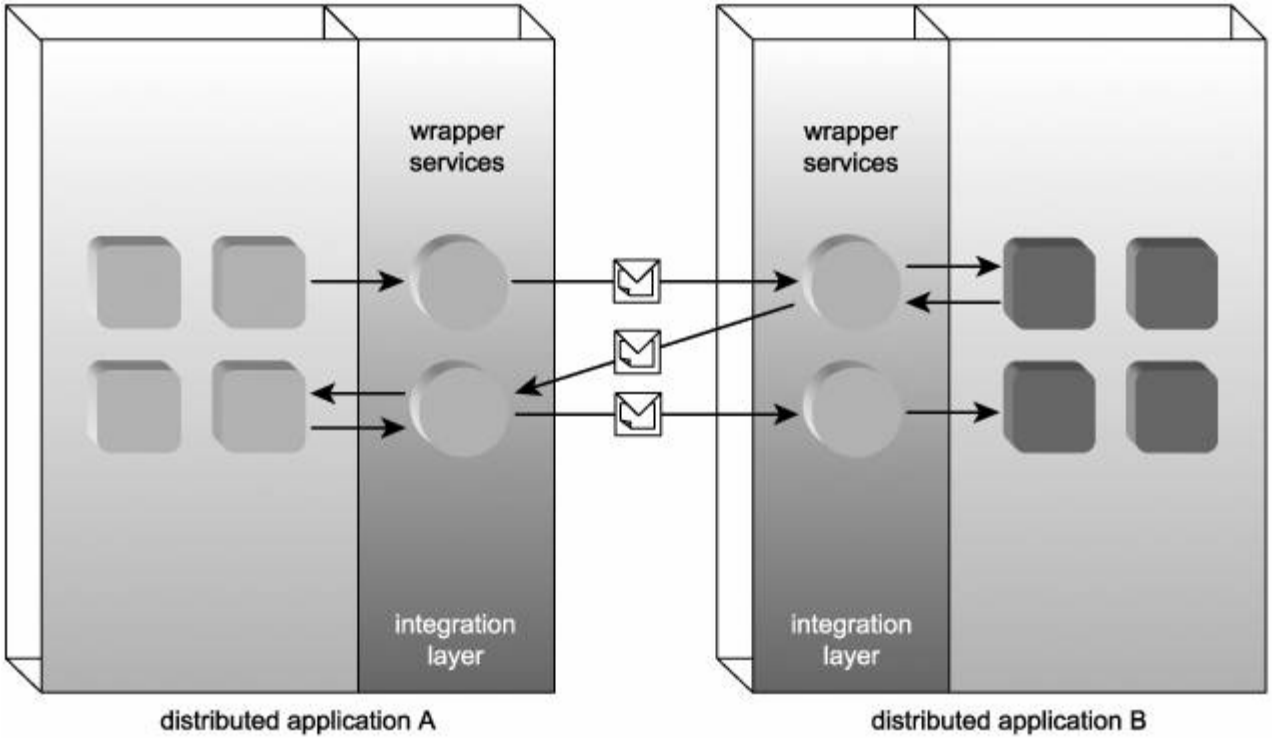
- Other maintenance tasks, such as resource management (similar to component management), are also required. However, to best foster reuse and composability, a useful part of an administration infrastructure for enterprises building large amounts of Web services is a private registry.
- UDDI is one of the technologies used for standardizing this interface repository, which can be manually or programmatically accessed to discover service descriptions.
- *Refer: Case Study 03*

SOA vs. hybrid Web service architecture

- In the previous section we mentioned how more recent variations of the distributed Internet architecture have come to incorporate Web services. This topic is worth elaborating upon because it has been (and is expected to continue to be) at the root of some confusion surrounding SOA.
- First, the use of Web services within traditional architectures is completely legitimate. Due to the development support for Web services in many established programming languages, they easily can be positioned to fit in with older application designs. And, for those legacy environments that do not support the custom development of Web services, adapters are often available.

Web services as component wrappers

- The primary role of Web services in this context has been to introduce an integration layer that consists of wrapper services that enable synchronous communication via SOAP-compliant integration channels.
- In fact, the initial release of the SOAP specification and the first generation of SOAP servers were specifically designed to duplicate RPC-style communication using messages.


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

Web services as component wrappers

- These integration channels are primarily utilized in integration architectures to facilitate communication with other applications or outside partners. They also are used to enable communication with other (more service-oriented) solutions and to take advantage of some of the features offered by third-party utility Web services.
- Regardless of their use or purpose within traditional architectures, it is important to clarify that a distributed Internet architecture that incorporates Web services in this manner does not qualify as a true SOA. It is simply a distributed Internet architecture that uses Web services.

Web services as component wrappers

- Instead of mirroring component interfaces and establishing point-to-point connections with Web services, SOA provides strong support for a variety of messaging models (based on both synchronous and asynchronous exchanges).
- Additionally, Web services within SOAs are subject to specific design requirements, such as those provided by service-orientation principles.
- These and other characteristics support the pursuit of consistent loose coupling. Once achieved, a single service is never limited to point-to-point communication; it can accommodate any number of current and future requestors.
- *Refer: Case Study 04*

Web services within SOA

- While SOAs can vary in size and quality, there are tangible characteristics that distinguish an SOA from other architectures that use Web services. Much of this book is dedicated to exploring these characteristics.
- For now it is sufficient to state that fundamentally, SOAs are built with a set of Web services designed to collectively automate (or participate in the automation of) one or more business processes and that SOA promotes the organization of these services into specialized layers that abstract specific parts of enterprise automation logic.
- Also by standardizing on SOA across an enterprise, a natural interoperability emerges that transcends proprietary application platforms. This allows for previously disparate environments to be composed in support of new and evolving business automation processes.

Service-orientation and object-orientation

- That distinction was made to stress the fact that the relationship between these two schools of thought is not necessarily a competitive one.
- In fact, object-oriented programming is commonly used to build the application logic encapsulated within Web services. However, how the object-oriented programming methodology differs fundamentally from service-orientation is worth exploring. An understanding of their differences will help you make them work together.
- Below is a list comparing aspects of these design approaches. (Whereas service-orientation is based on the design of services, object-orientation is centered around the creation of objects. Because comparing services to objects can be confusing, the term "units of processing logic" is used.)

Service-orientation and object-orientation

- Service-orientation emphasizes **loose coupling** between units of processing logic (services). Although object-orientation supports the creation of reusable, loosely coupled programming routines, much of it is based on predefined class dependencies, resulting in more tightly bound units of processing logic (objects).
- Service-orientation encourages **coarse-grained** interfaces (service descriptions) so that every unit of communication (message) contains as much information as possible for the completion of a given task. Object-oriented programming fully supports **fine-grained** interfaces (APIs) so that units of communication (RPC or local API calls) can perform various sized tasks.

Service-orientation and object-orientation

- Service-orientation expects the **scope** of a unit of processing logic (service) to **vary significantly**. Object-oriented units of logic (objects) tend to be **smaller and more specific** in scope.
- Service-orientation promotes the creation of **activity-agnostic units** of processing logic (services) that are driven into action by intelligent units of communication (messages). Object-orientation encourages the **binding of processing logic with data**, resulting in highly intelligent units (objects).

Service-orientation and object-orientation

- Service-orientation prefers that units of processing logic (services) be designed to remain **as stateless as possible**. Object-orientation promotes the binding of data and logic, resulting in the creation of **more stateful units** (objects). (However, more recent component-based design approaches deviate from this tendency.)
- Service-orientation supports the **composition of loosely coupled** units of processing logic (services). Object-orientation supports composition but also encourages **inheritance** among units of processing logic (objects), which can lead to **tightly coupled dependencies**.

Summary of Key Points

- SOA is a radical departure from client-server architecture. Current SOAs still employ some of the technologies originally used to build client-server applications. Though more sophisticated, SOAs introduce complexity that sharply contrasts the simplicity of a two-tier client-server architecture.
- Distributed Internet architecture has much in common with SOA, including a significant amount of its technology. However, SOA has distinct characteristics relating to both technology and its underlying design principles. For example, SOA introduces processing and security requirements that differ from distributed Internet architecture, and SOA administration is typically more complex due to its reliance on messaging-based communication.
- Traditional architectures have and can continue to use Web services within their own design paradigms. It's important to not confuse these architectures with SOA. Non-SOA use of Web services is typically found within distributed Internet architectures, where Web services are employed to mirror RPC-style communication.