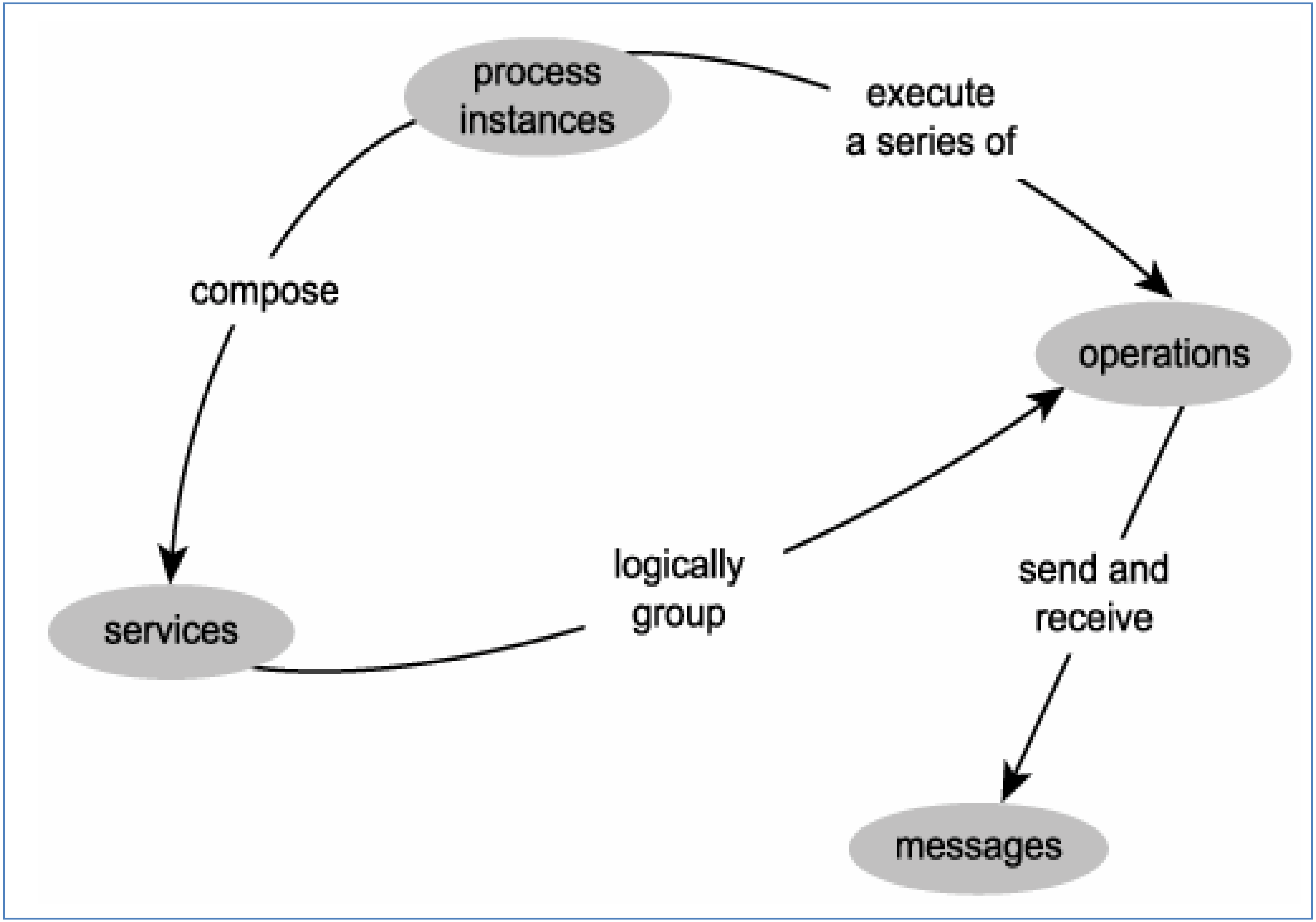
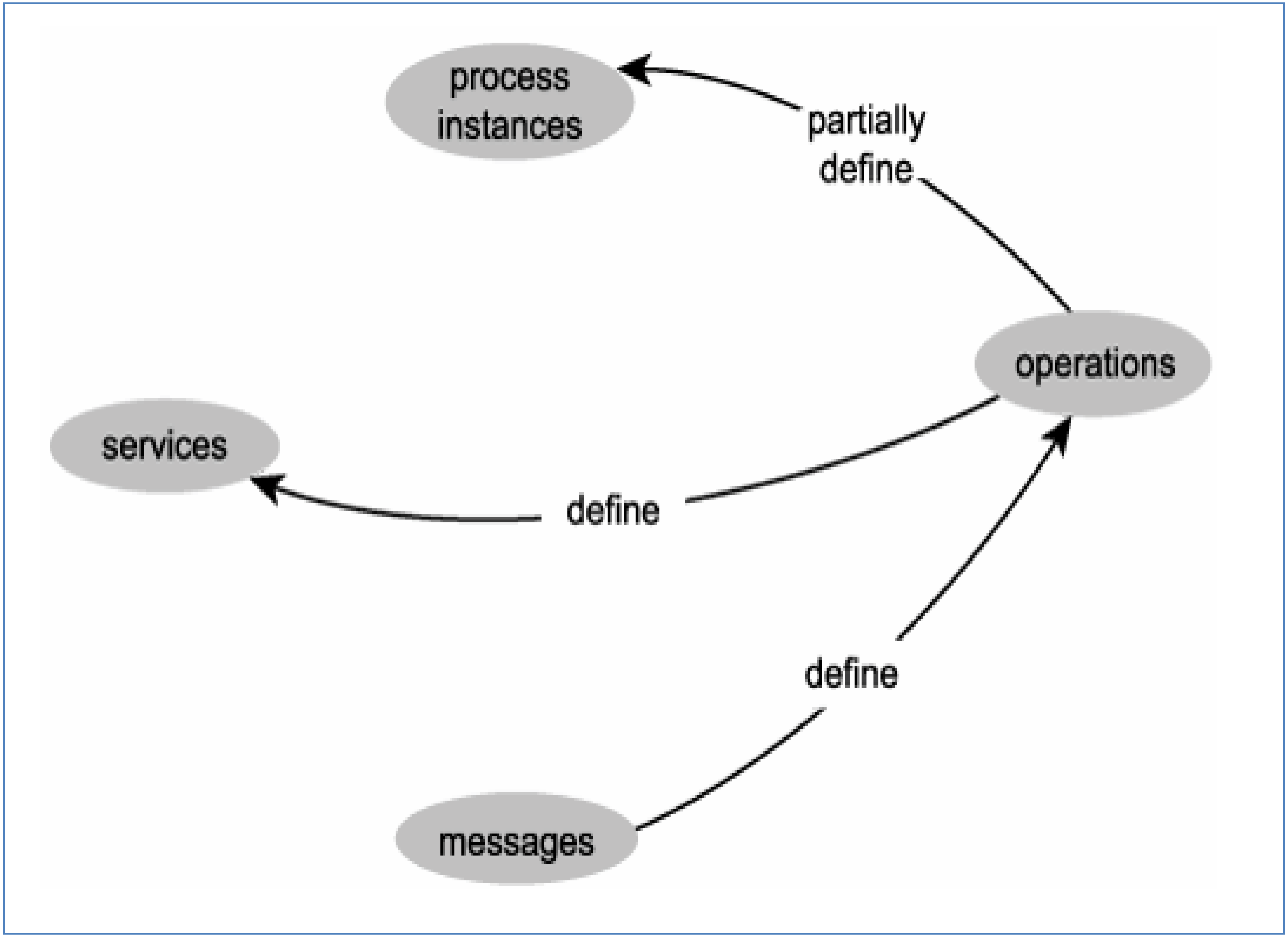
How components in an SOA Inter-relate?

- An operation sends and receives messages to perform work.
  - An operation is therefore mostly defined by the messages it processes.
- A service groups a collection of related operations.
  - A service is therefore mostly defined by the operations that comprise it.

How components in an SOA inter-relate

- A process instance can compose services.
  - A process instance is not necessarily defined by its services because it may only require a subset of the functionality offered by the services.
  - A process instance invokes a unique series of operations to complete its automation.
- Every process instance is therefore partially defined by the service operations it uses.


process instances
execute a series of
operations
compose
services
logically group
send and receive
messages


process instances
partially define
operations
services
define
define
messages

So far,

- The logical parts of an SOA can be mapped to corresponding components in the basic Web services framework.
- By viewing a service-oriented solution as a unit of automation logic, we establish that SOA consists of a sophisticated environment that supports a highly modularized separation of logic into differently scoped units.
- SOA further establishes specific characteristics, behaviors, and relationships among these components that provide a predictable environment in support of service-orientation.

Common Principles of Service-Orientation

- **Services are reusable**
  - Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.
- **Services share a formal contract**
  - For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.
- **Services are loosely coupled**
  - Services must be designed to interact without the need for tight, cross-service dependencies.

Common Principles of Service-Orientation

- **Services abstract underlying logic**
  - The only part of a service that is visible to the outside world is what is exposed via the service contract.
  - Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.
- **Services are composable**
  - Services may compose other services.
  - This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.

Common Principles of Service-Orientation

- **Services are autonomous**
  - The logic governed by a service resides within an explicit boundary.
  - The service has control within this boundary and is not dependent on other services for it to execute its governance.
- **Services are stateless**
  - Services should not be required to manage state information, as that can impede their ability to remain loosely coupled.
  - Services should be designed to maximize statelessness even if that means deferring state management elsewhere.

Common Principles of Service-Orientation

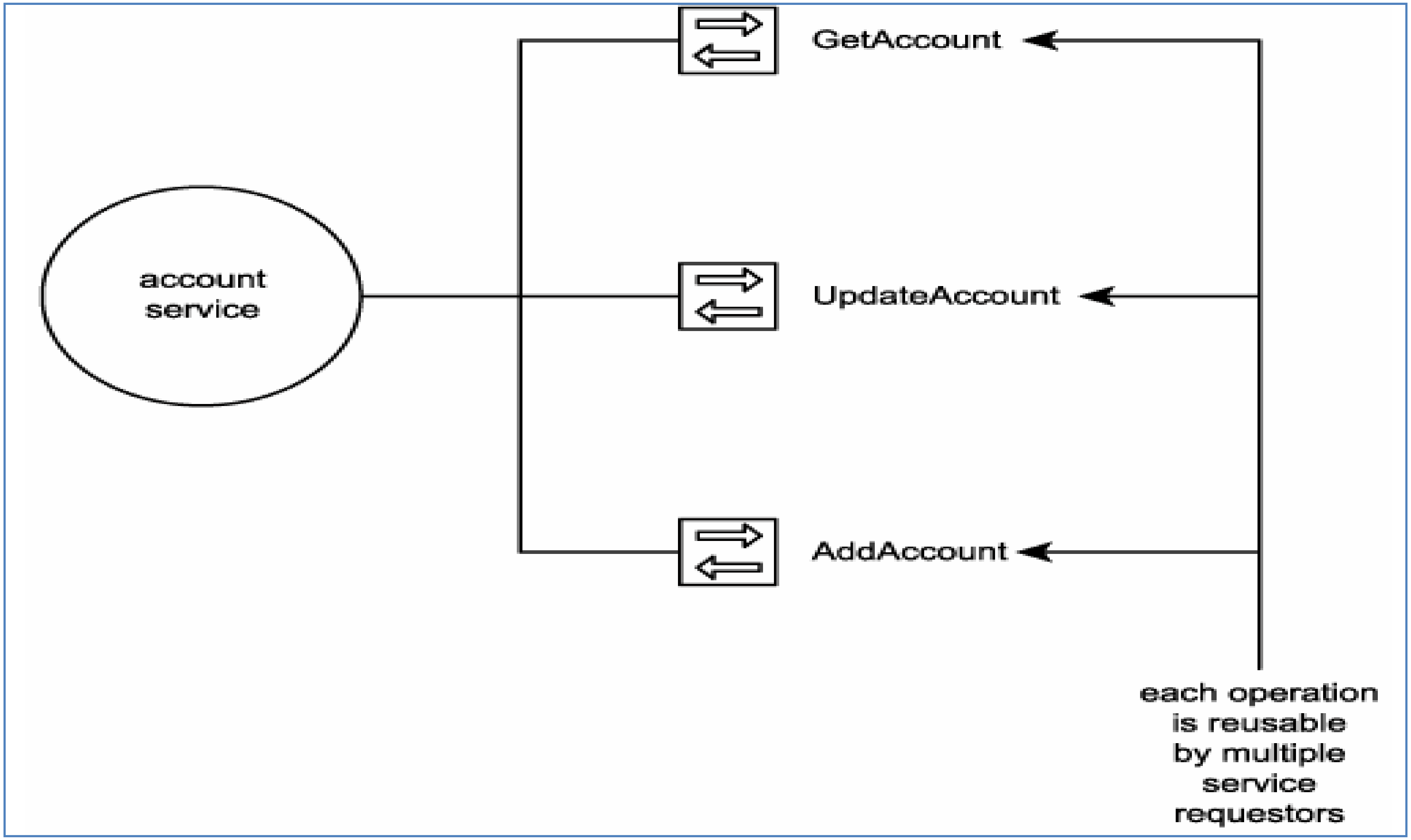
- **Services are discoverable**
  - Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

- Of these eight, autonomy, loose coupling, abstraction and the need for a formal contract can be considered the **core principles** that form the baseline foundation for SOA.

Services are Reusable

- Service-orientation encourages reuse in all services, regardless if immediate requirements for reuse exist.
- Application of design standards is required so that future requirements can be easily accommodated
- No need to create wrapper services if the services are reusable
- This principle facilitates **all forms of reuse**, including inter-application interoperability, composition and the creation of utility services.

Services are Reusable



Services are Reusable

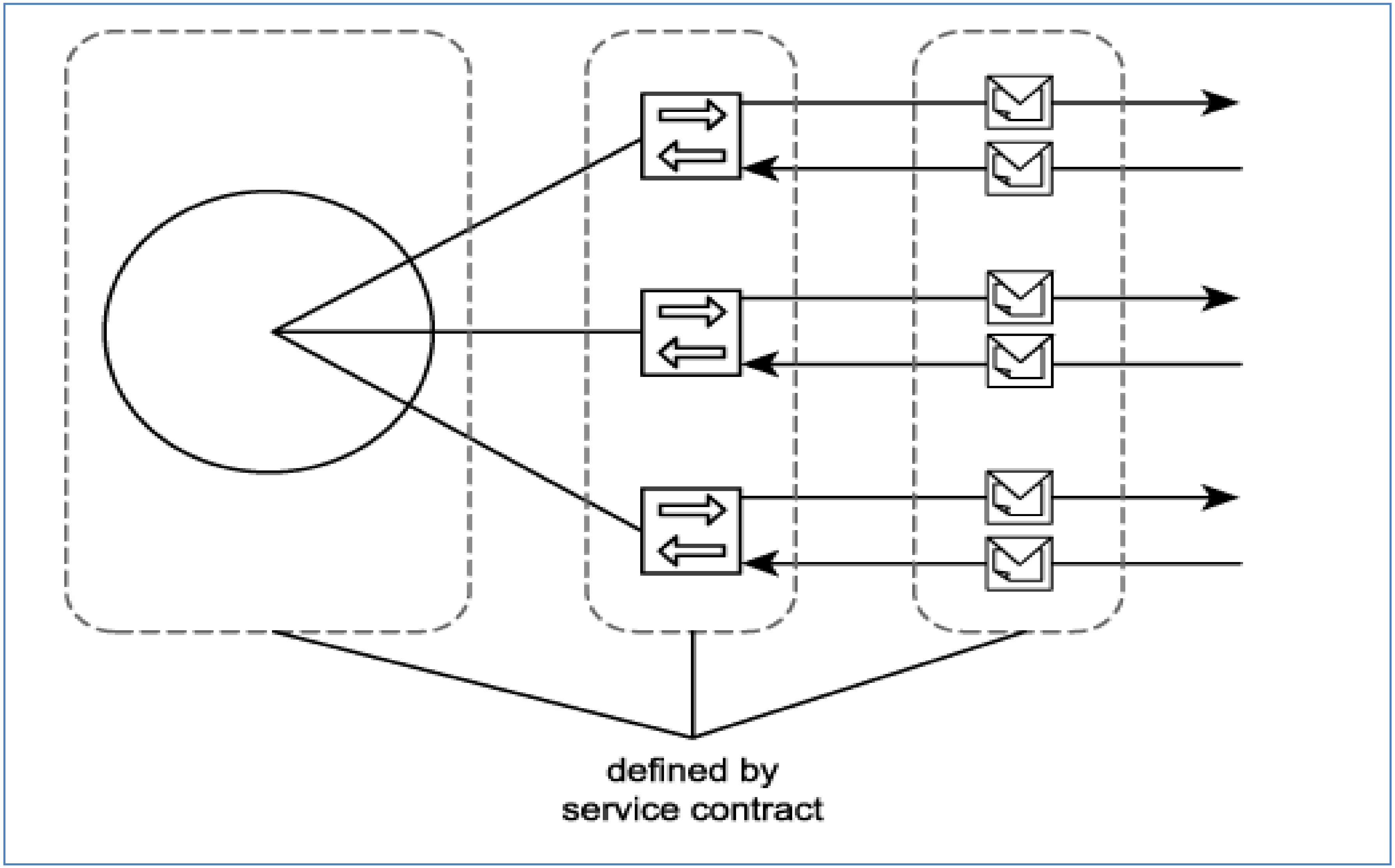
- "Messaging indirectly supports service reusability through the use of SOAP headers.“
  - The processing-specific logic embedded in a message alleviates the need for a service to contain this logic.
  - More importantly, it imposes a requirement that service operations become less activity-specific.

- "The more generic a service's operations are, the more reusable the service."

Services share a Formal Contract

- **Service contracts** provide a formal definition of:
  - The service endpoint
  - Each service operation
  - Every input and output message supported by each operation
  - Rules and characteristics of the service and its operations
- Hence it covers **all primary components** of SOA

Services share a Formal Contract



Services share a Formal Contract

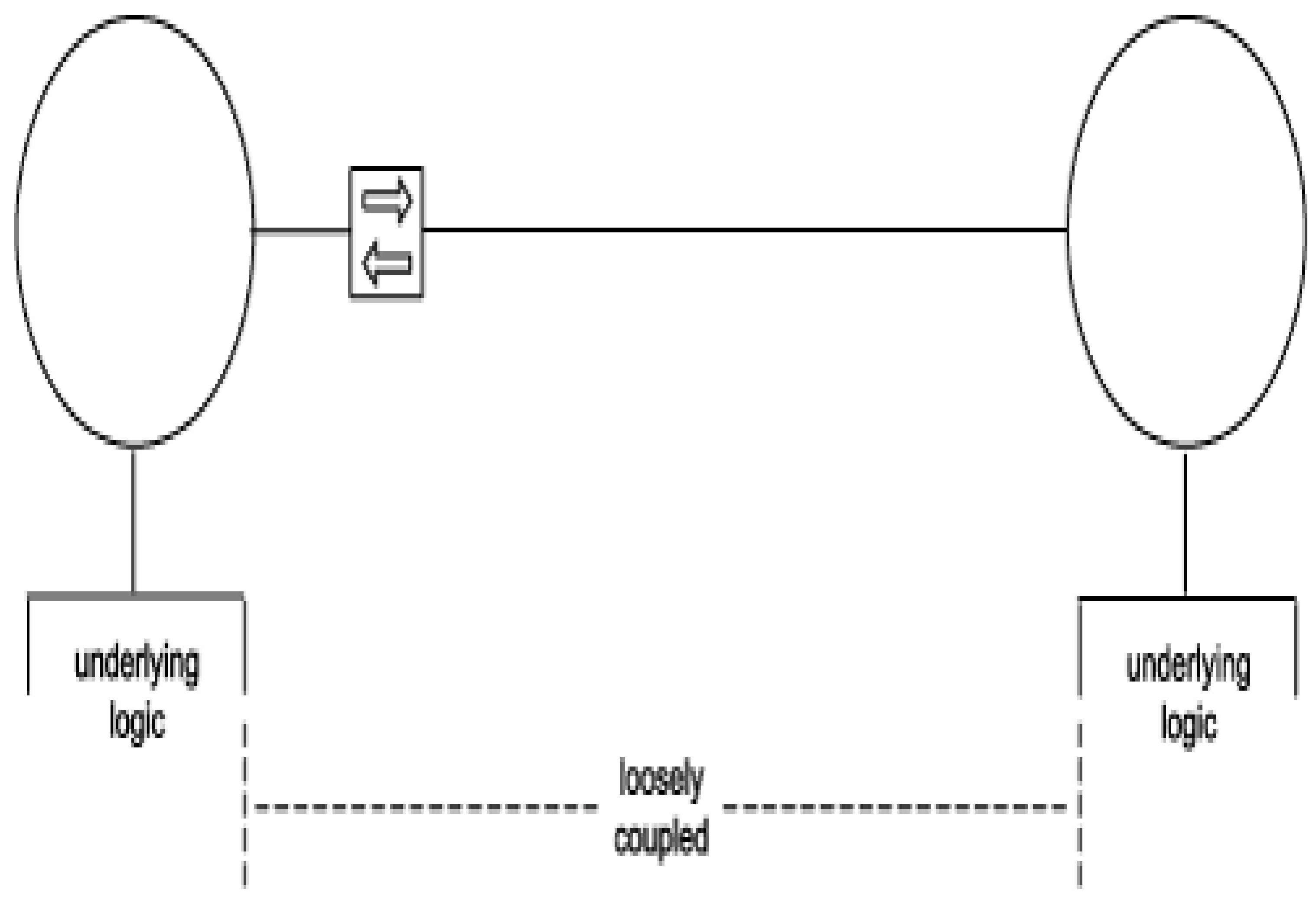
- “Contract design is extremely important.”
- “Contracts need to be carefully maintained and versioned after their initial release.”
- WSDL definition, XSD schemas, and policies, can be viewed collectively as a communications contract that expresses exactly how a service can be programmatically accessed.

Services are Loosely Coupled

- *“No one can predict how an IT environment will evolve.”*

- Being able to ultimately respond to unforeseen changes in an efficient manner is a key goal of applying service-orientation.

- **Loose Coupling:**
  - “Having knowledge of other services still remain independent”


underlying logic
underlying logic
loosely coupled

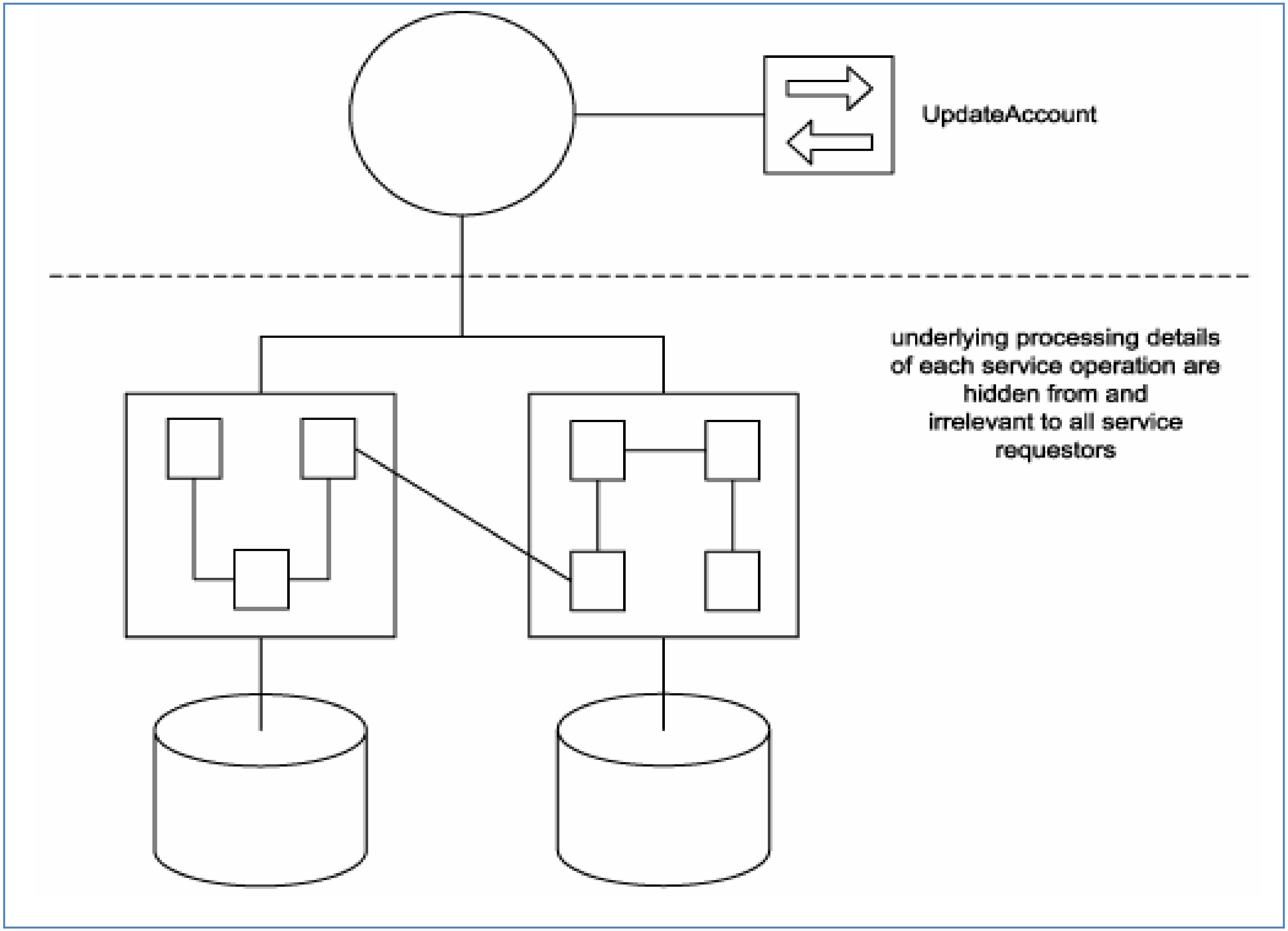
Services are Loosely Coupled

- Loose coupling is achieved through the use of service contracts that allow services to interact within predefined parameters.

- *“It is interesting to note that within a loosely coupled architecture, service contracts actually tightly couple operations to services.“*
  - When a service describes its operations, other services become dependent on that.

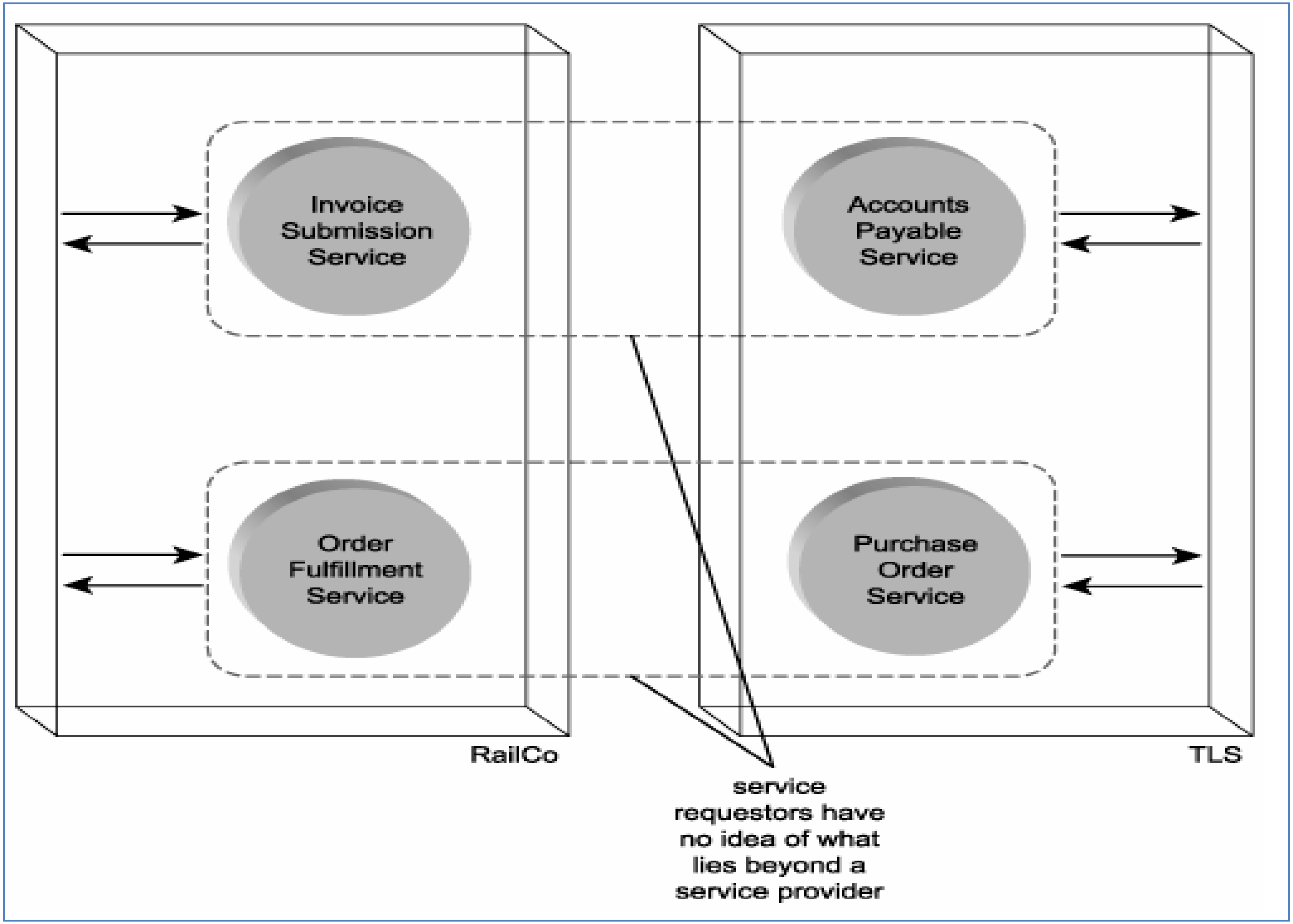
Services Abstract underlying logic

- Also referred to as *service interface-level abstraction*, it is this principle that allows services to act as black boxes, hiding their details from the outside world.
- The scope of logic represented by a service significantly influences the design of its operations and its position within a process.
- Services can encapsulate varying amounts of logic


UpdateAccount
underlying processing details of each service operation are hidden from and irrelevant to all service requestors

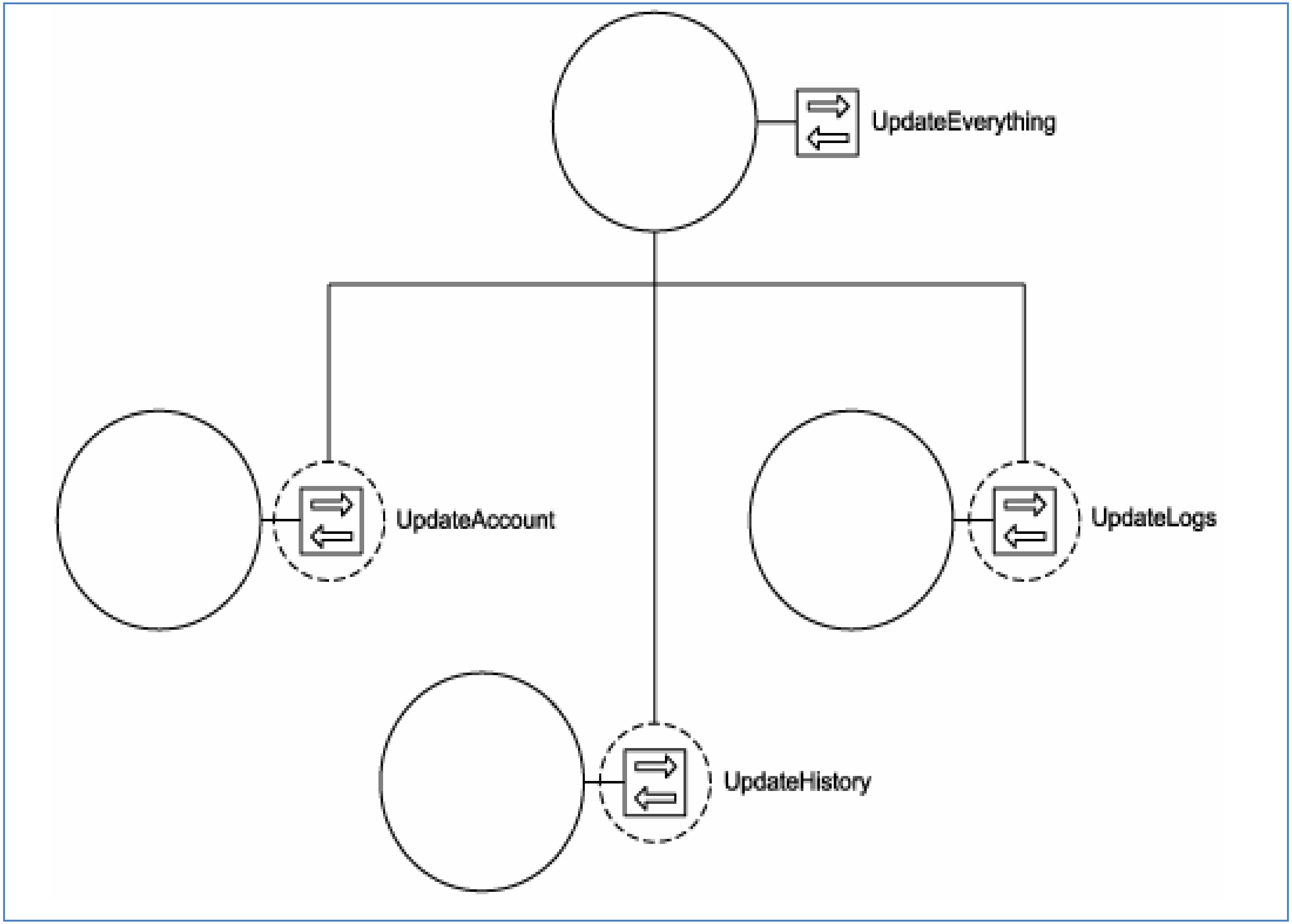
Services Abstract underlying logic

- *"Operation granularity is a primary design consideration"*
  - It is the individual operations that collectively abstract the underlying logic.
  - Services simply act as containers for these operations.
- The loosely coupled communications structure requires that the only piece of knowledge services need to interact is each others' service descriptions.


Invoice Submission Service
Accounts Payable Service
Order Fulfillment Service
Purchase Order Service
RailCo
TLS
service requestors have no idea of what lies beyond a service provider

Services are Composable

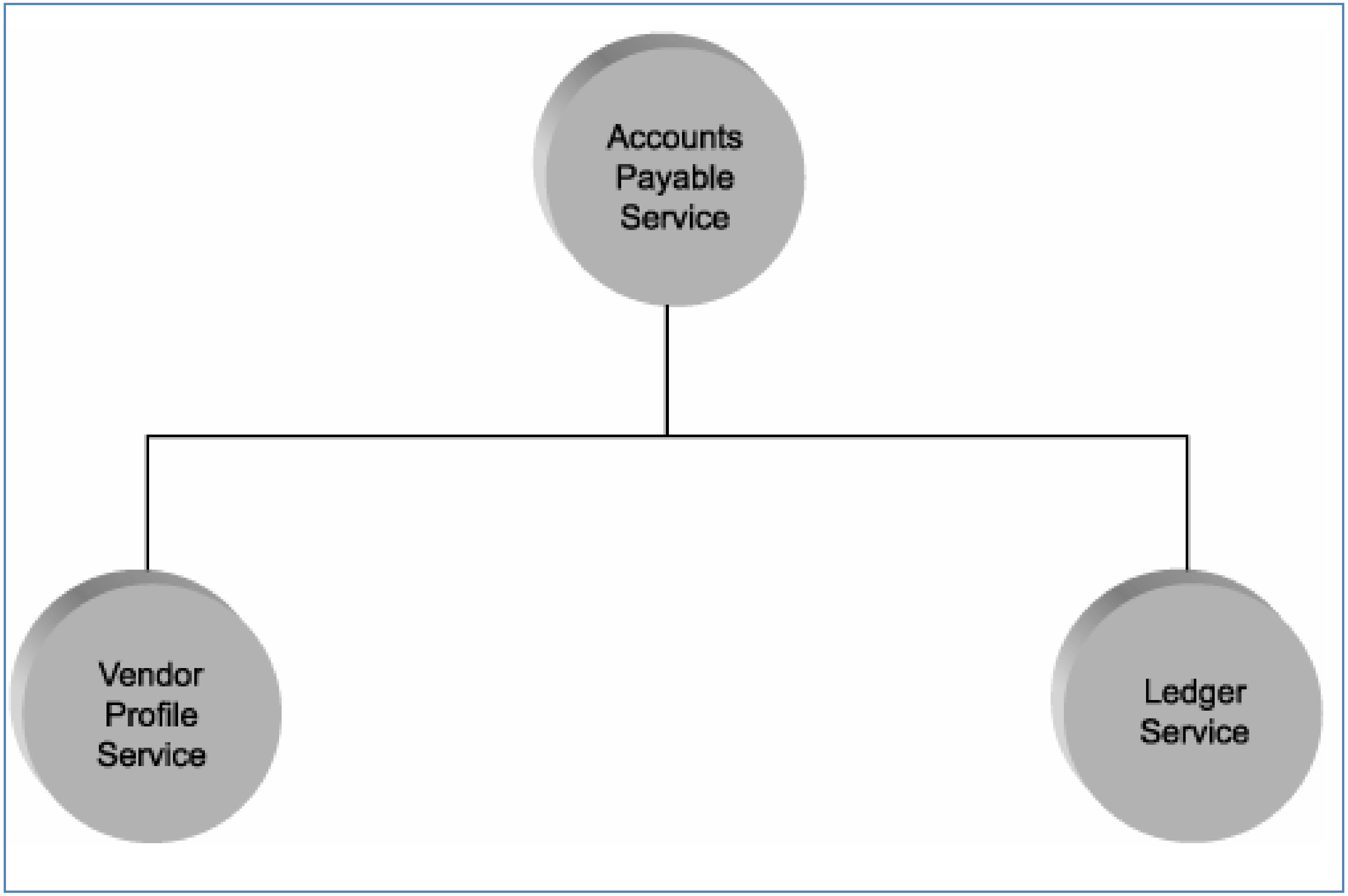
- Services should be able to take part in compositions as and when required
- A common SOA extension that underlines composability is the concept of **orchestration**.
  - Here, a service-oriented process is controlled by a parent process service that composes process participants.


UpdateEverything
UpdateAccount
UpdateLogs
UpdateHistory

Services are Composable

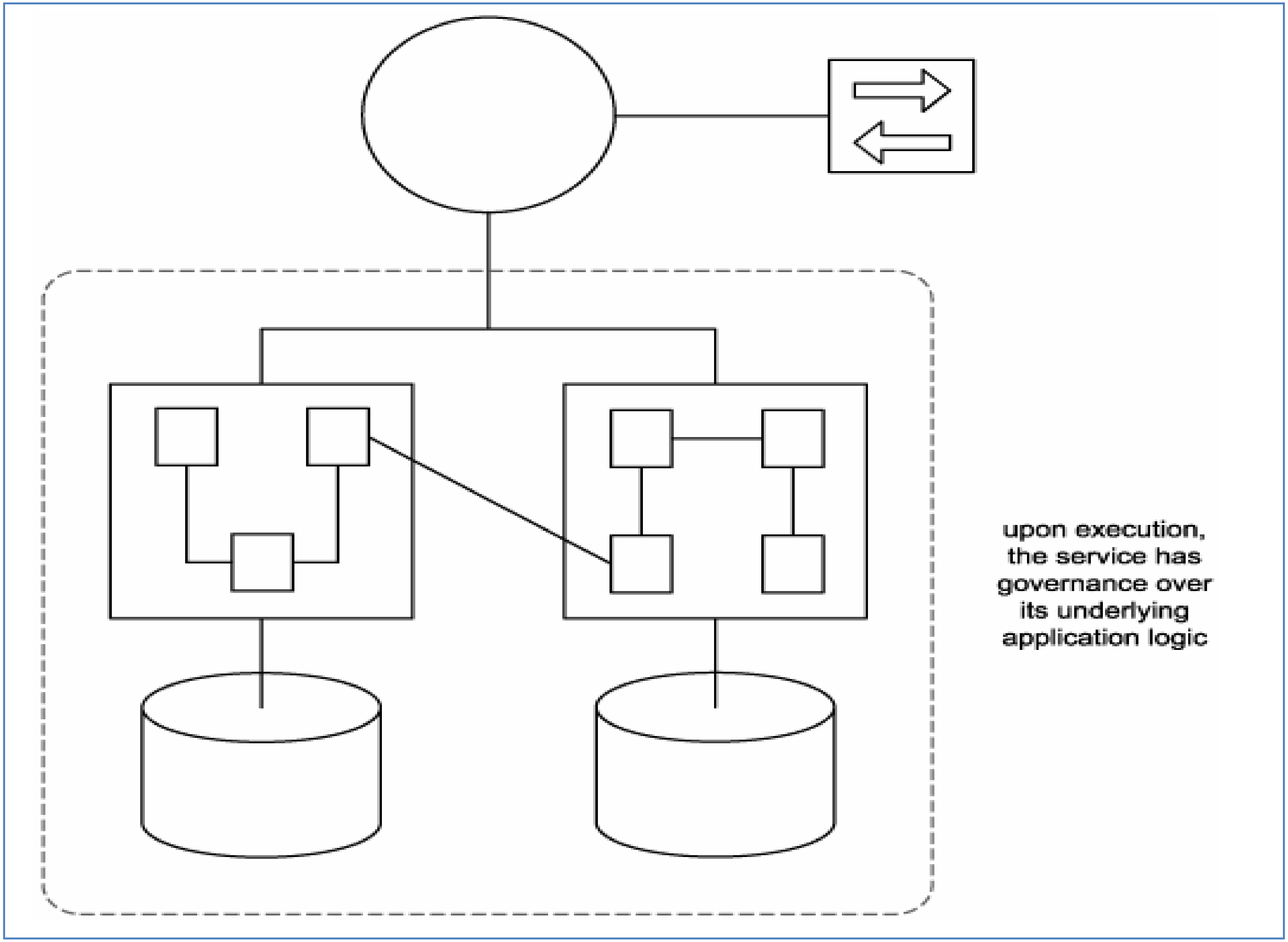
- "Composability is simply another form of reuse"
  - Therefore operations need to be designed in a standardized manner
  - and with an appropriate level of granularity to maximize composition opportunities.

Case Study


Accounts Payable Service
Vendor Profile Service
Ledger Service

Services are Autonomous

- Autonomy requires that the range of logic exposed by a service exist within an explicit boundary.
- It also eliminates dependencies on other services, which frees a service from ties that could inhibit its deployment and evolution.
- “Service autonomy is a primary consideration when deciding how application logic should be divided up into services and which operations should be grouped together within a service context.”


upon execution,
the service has
governance over
its underlying
application logic

Services are Autonomous

- "Deferring the location of business rules is one way to strengthen autonomy and keep services more generic."
  - Orchestration helps to achieve this aspect

- "Autonomy does not necessarily grant a service exclusive ownership of the logic it encapsulates."
  - It only guarantees that at the time of execution, the service has control over whatever logic it represents.
  - We therefore can make a distinction between two types of autonomy.

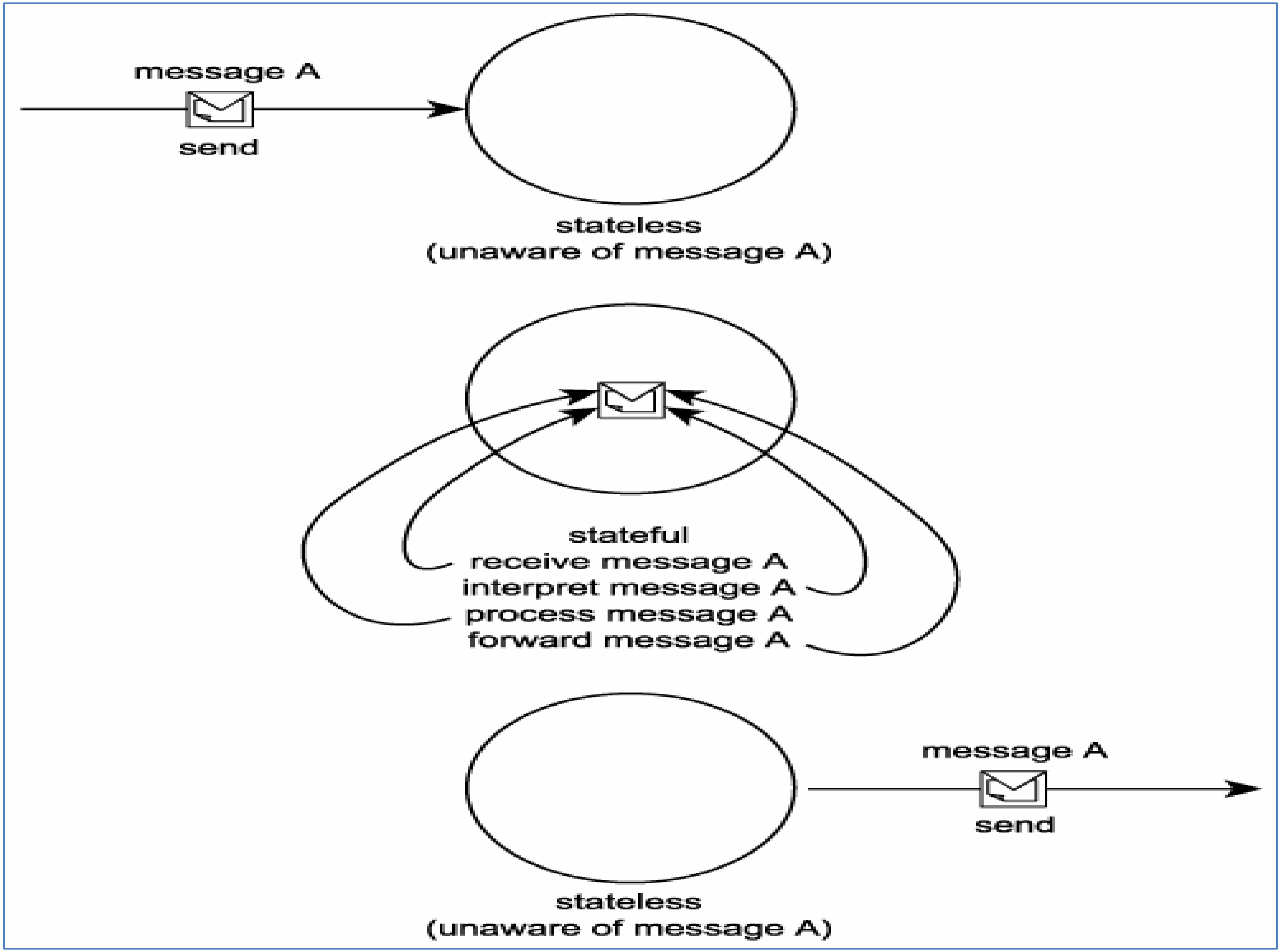
Services are Autonomous

1. **Service-level Autonomy**
   - Service boundaries are distinct from each other, but the service may share underlying resources.

2. **Pure Autonomy**
   - The underlying logic is under complete control and ownership of the service.
   - Used when a new service is being built

Services are Stateless

- Services should minimize the amount of state information they manage and the duration for which they hold it.
- *“While a service is processing a message, for example, it is temporarily stateful.”*
- If a service is responsible for retaining state for longer periods of time, its ability to remain available to other requestors will be impeded.


message A
send
stateless
(unaware of message A)
stateful
receive message A
interpret message A
process message A
forward message A
message A
send
stateless
(unaware of message A)

Services are Stateless

- “Statelessness is a preferred condition for services and one that promotes reusability and scalability”.

- For a service to retain as little state as possible, its individual operations need to be designed with stateless processing considerations.

- A primary quality of SOA that supports statelessness is the use of document-style messages.
  - The more intelligence added to a message, the more independent and self-sufficient it remains.

Services are Discoverable

- Discovery helps avoid the accidental creation of redundant services or services that implement redundant logic.

So far,

- Different organizations have published their own versions of service-oriented principles. As a result, many variations exist.
- The most common principles relate to loose coupling, autonomy, discoverability, composability, reuse, service contracts, abstraction, and statelessness.

?

- What's the difference between reusability and composability?
- What's the difference between autonomy and statelessness?
- What's the difference between loose coupling and the use of a service contract?

Service-Orientation Principles

- Services are reusable = service reusability
- Services share a formal contract = service contract
- Services are loosely coupled = service loose coupling
- Services abstract underlying logic = service abstraction
- Services are composable = service composability
- Services are autonomous = service autonomy
- Services are stateless = service statelessness
- Services are discoverable = service discoverability

Service Reusability

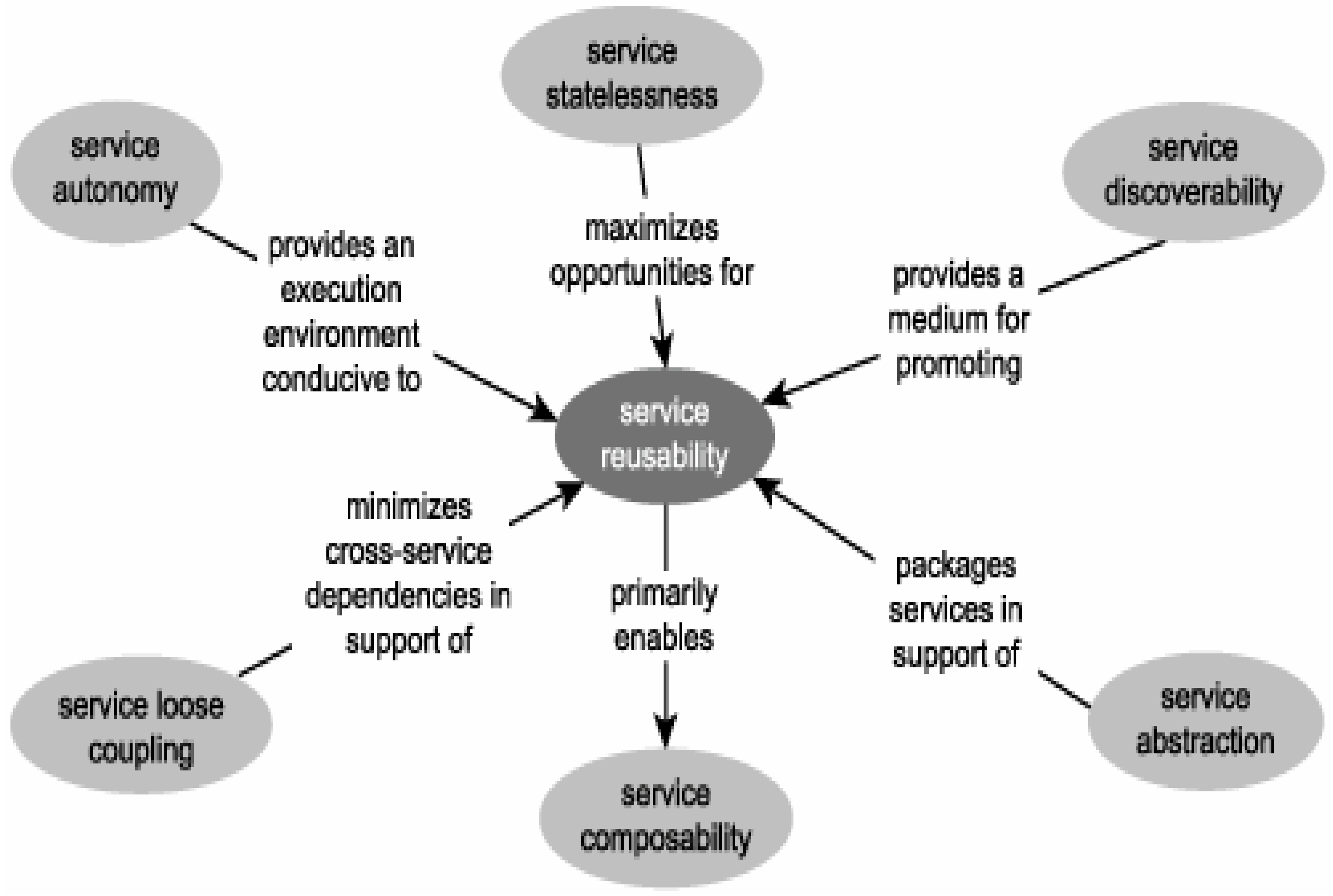
- “When a service encapsulates logic that is useful to more than one service requestor, it can be considered reusable.“
- **Supported by:**
  - Service autonomy establishes an execution environment that facilitates reuse because the service has independence and self-governance.

Service Reusability

- Service statelessness supports reuse because it maximizes the availability of a service.
- Service abstraction fosters reuse because it establishes the black box concept.
- Service discoverability promotes reuse, as it allows requestors to search for and discover reusable services.
- Service loose coupling establishes an inherent independence that frees a service from immediate ties to others.

Service Reusability

- **Supports:**
  - Service composability is primarily possible because of reuse.


service statelessness
maximizes opportunities for
service autonomy
provides an execution environment conducive to
service discoverability
provides a medium for promoting
service reusability
minimizes cross-service dependencies in support of
service loose coupling
primarily enables
service composability
packages services in support of
service abstraction

Service Contract

- "A service contract is a representation of a service's collective metadata."

- It standardizes the expression of rules and conditions that need to be fulfilled by any requestor wanting to interact with the service.

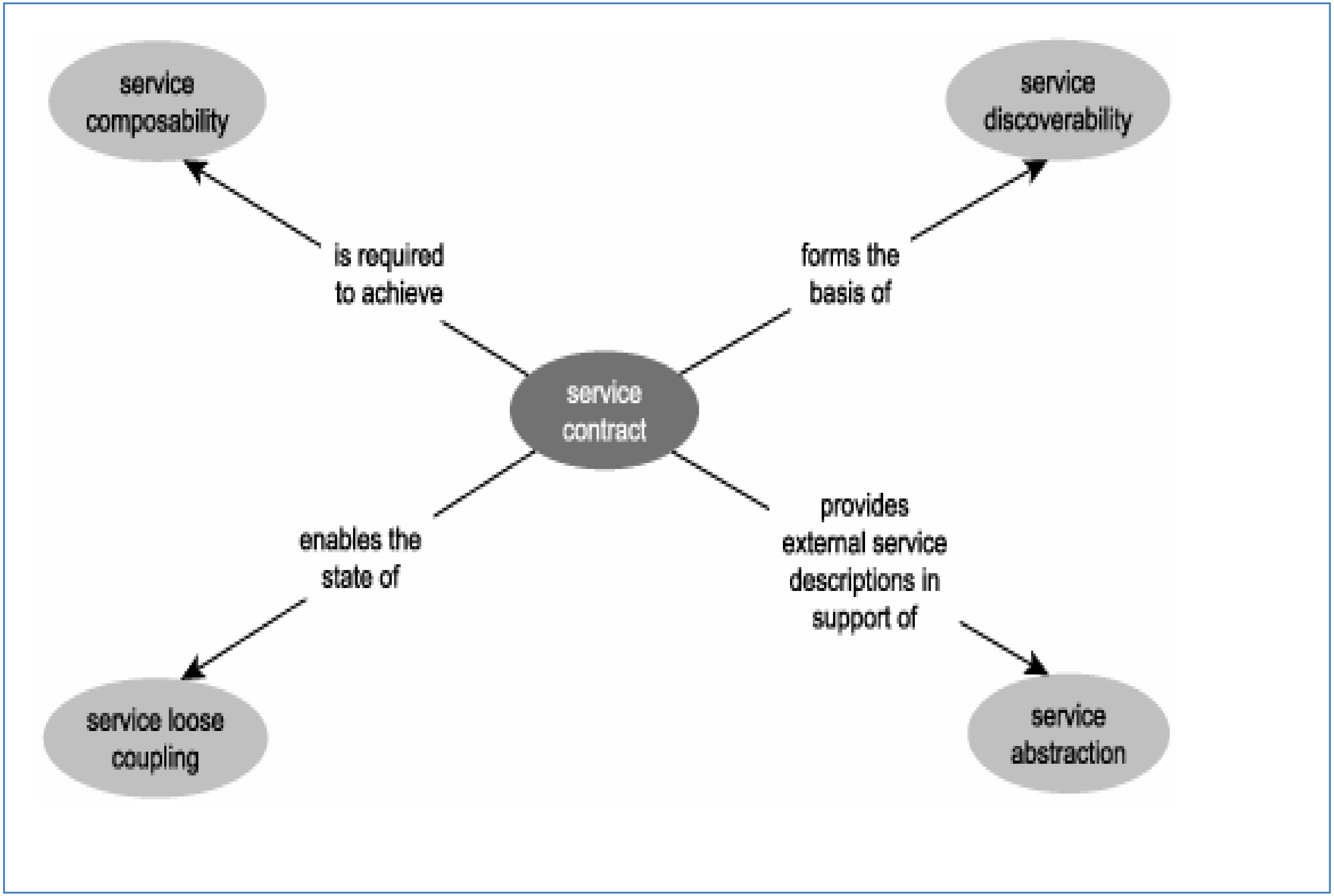
- **Core Principle**

Service Contract

- **Supports:**
  - Service abstraction is realized through a service contract, as it is the metadata expressed in the contract that defines the only information made available to service requestors.
  - Service loose coupling is made possible through the use of service contracts.
    - Processing logic from different services do not need to form tight dependencies; they simply need an awareness of each other's communication requirements

Service Contract

- Service composability is enabled through the use of service contracts as a controller service needs contract of member services.
- Service discoverability is based on the use of service contracts.


service composability
service discoverability
is required to achieve
forms the basis of
service contract
enables the state of
provides external service descriptions in support of
service loose coupling
service abstraction

Service Loose Coupling

- “Loose coupling is a state that supports a level of independence between services.”

- Independence or non-dependency is a fundamental aspect of services and SOA as a whole.

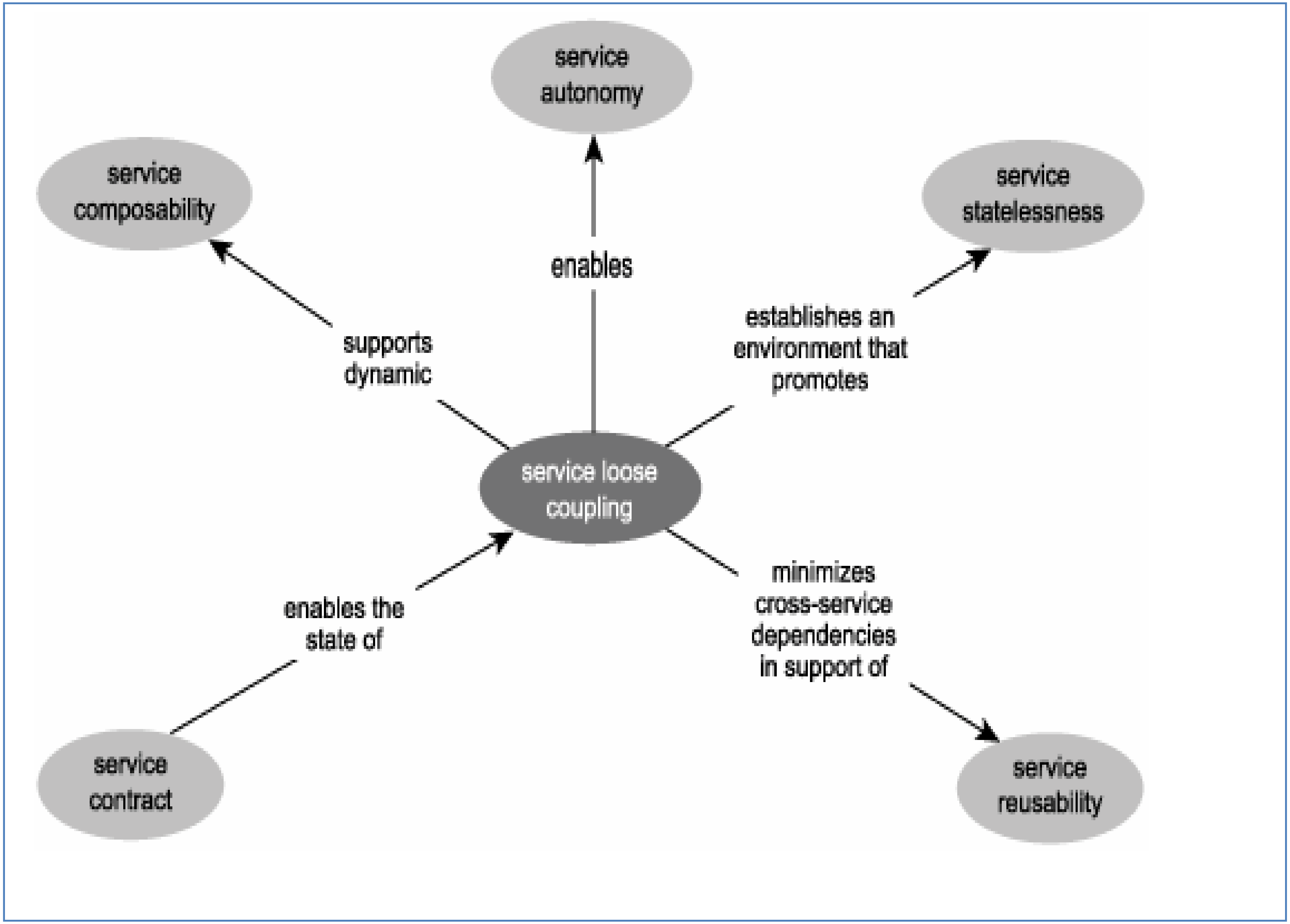
Service Loose Coupling

- **Supports:**
  - Service reusability is supported through loose coupling because services are freed from tight dependencies on others. This increases their availability for reuse opportunities.
  - Service composability is fostered by the loose coupling of services, especially when services are dynamically composed.
  - Service statelessness is directly supported through the loosely coupled communications framework established by this principle.

Service Loose Coupling

- Service autonomy is made possible through this principle, as it is the nature of loose coupling that minimizes cross-service dependencies.

- **Supported By:**
  - Service contracts are what enable loose coupling between services, as the contract is the only piece of information required for services to interact.


service autonomy
service composability
service statelessness
enables
supports dynamic
establishes an environment that promotes
service loose coupling
enables the state of
minimizes cross-service dependencies in support of
service contract
service reusability

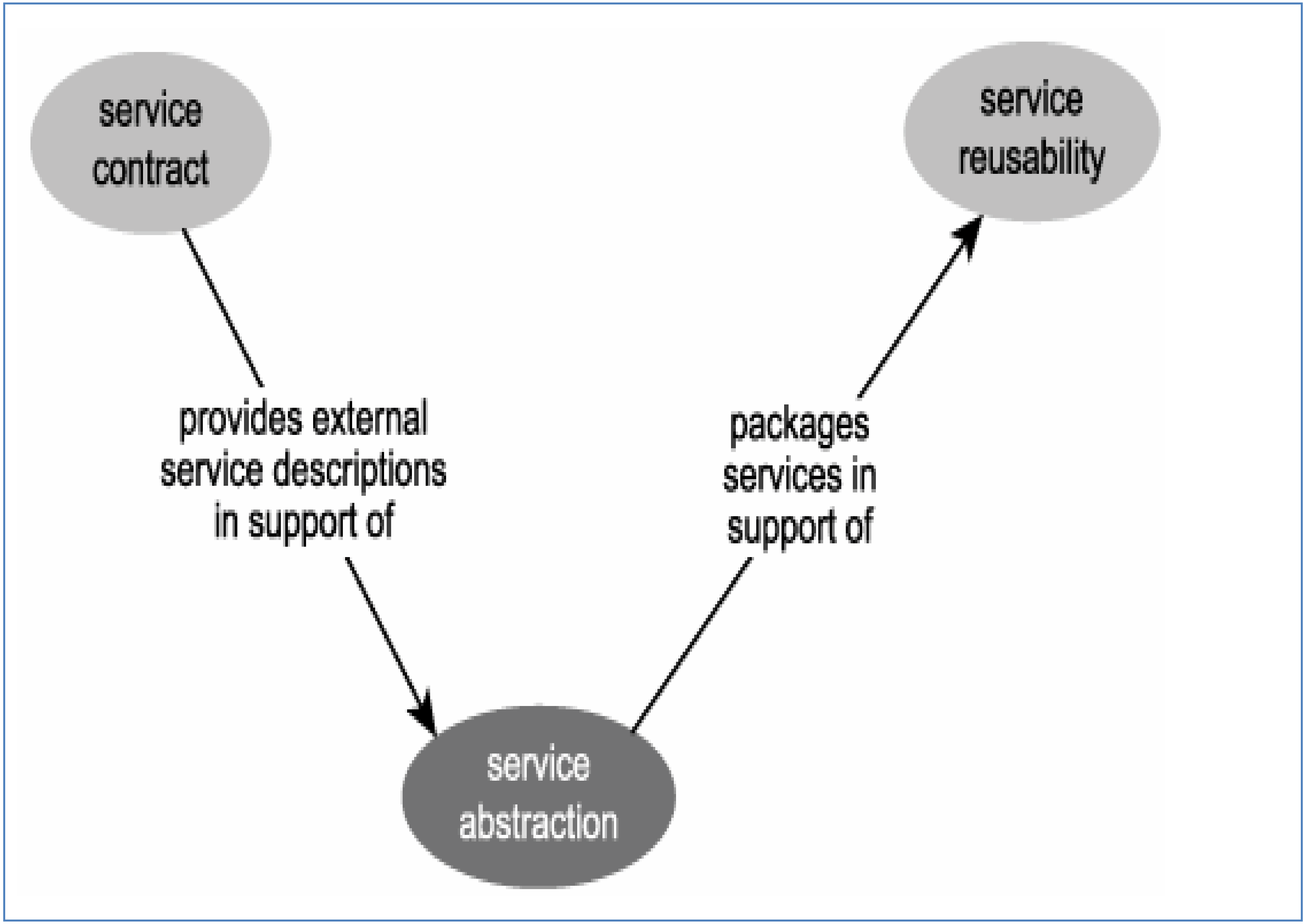
Service Abstraction

- Part of building solutions with independent services is allowing those services to encapsulate potentially complex processing logic
  - and exposing that logic through a generic and descriptive interface.

Service Abstraction

- **Supported by:**
  - Service contracts, in a manner, implement service abstraction by providing the official description information that is made public to external service requestors.

- **Supports:**
  - Service reusability is supported by abstraction, as long as what is being abstracted is actually reusable.


service contract
service reusability
provides external service descriptions in support of
packages services in support of
service abstraction

Service Composability

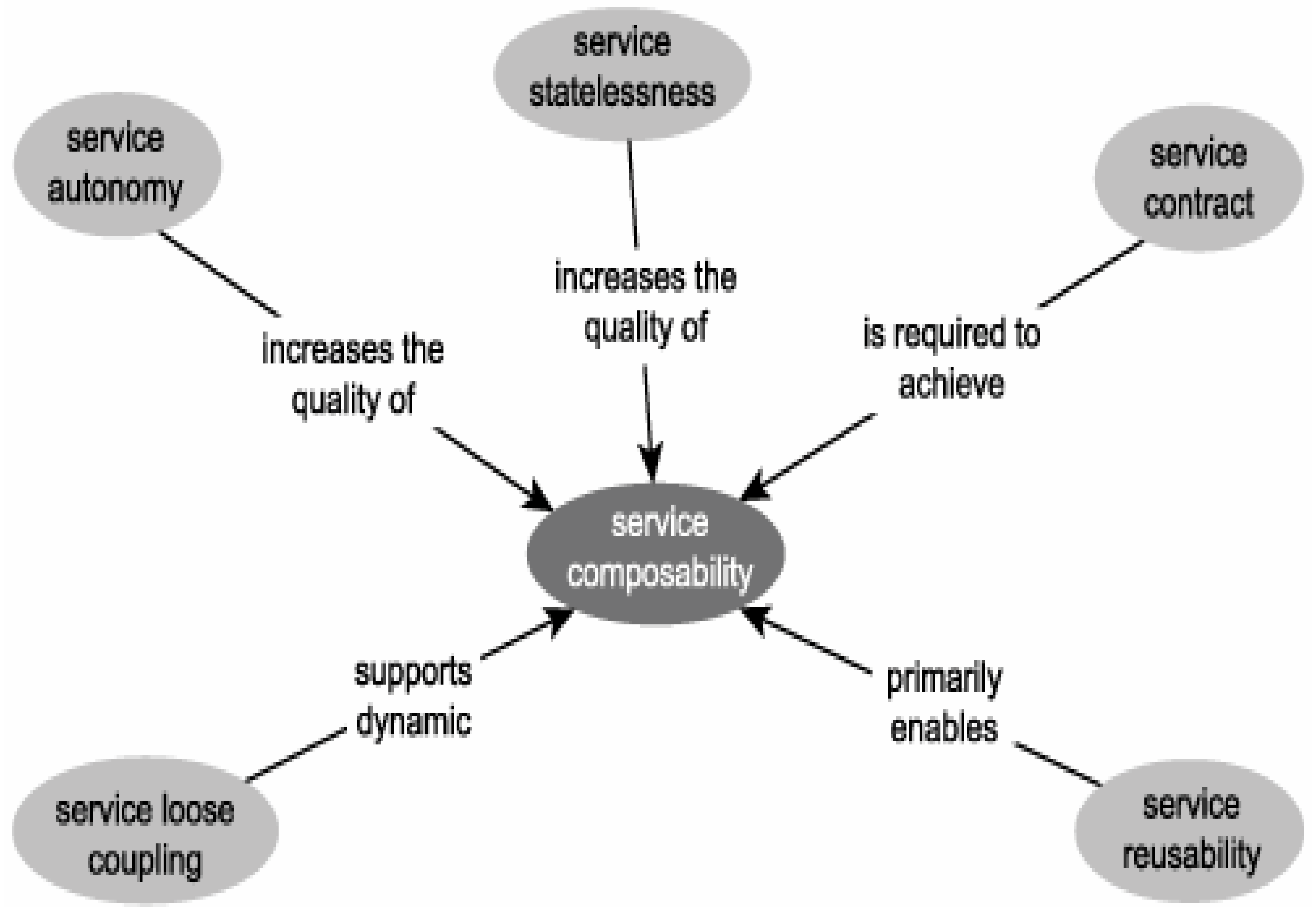
- “Designing services so that they support composition by others is fundamental to building service-oriented solutions.”

Service Composability

- **Supported by:**
  - Service reusability is what enables one service to be composed by numerous others.
  - Service loose coupling establishes a communications framework that supports the concept of dynamic service composition.
  - Service statelessness supports service composability, especially in larger compositions. If all services are stateless, the overall composition executes more harmoniously.

Service Composability

- Service autonomy held by composition members strengthens the overall composition.
- Service contracts enable service composition by formalizing the runtime agreement between composition members.


service statelessness
service autonomy
service contract
increases the quality of
increases the quality of
is required to achieve
service composability
supports dynamic
primarily enables
service loose coupling
service reusability

Service Autonomy

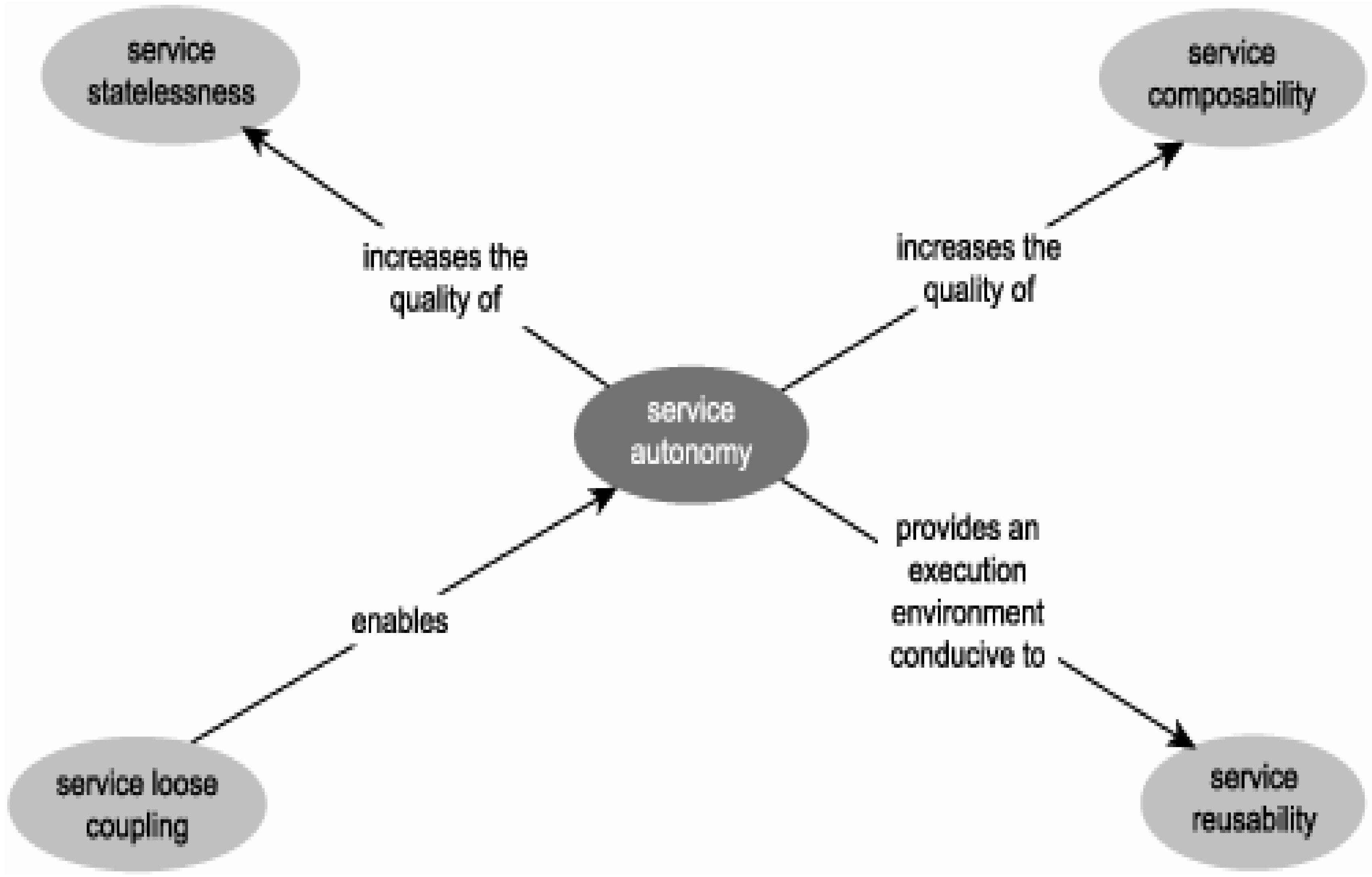
- This principle applies to a service's underlying logic.
- “By providing an execution environment over which a service has complete control, service autonomy relates to several other principles.”

Service Autonomy

- **Supports:**
  - Service reusability is more easily achieved when the service offering reusable logic has self-governance over its own logic.
  - Service composability is also supported by service autonomy as service composition consisting of autonomous services is much more robust and collectively independent.
  - Service statelessness is best implemented by a service that can execute independently.

Service Autonomy

- **Supported by:**
  - Service loose coupling is a primary enabler of this principle.


service statelessness
service composability
increases the quality of
increases the quality of
service autonomy
enables
provides an execution environment conducive to
service loose coupling
service reusability

Service Statelessness

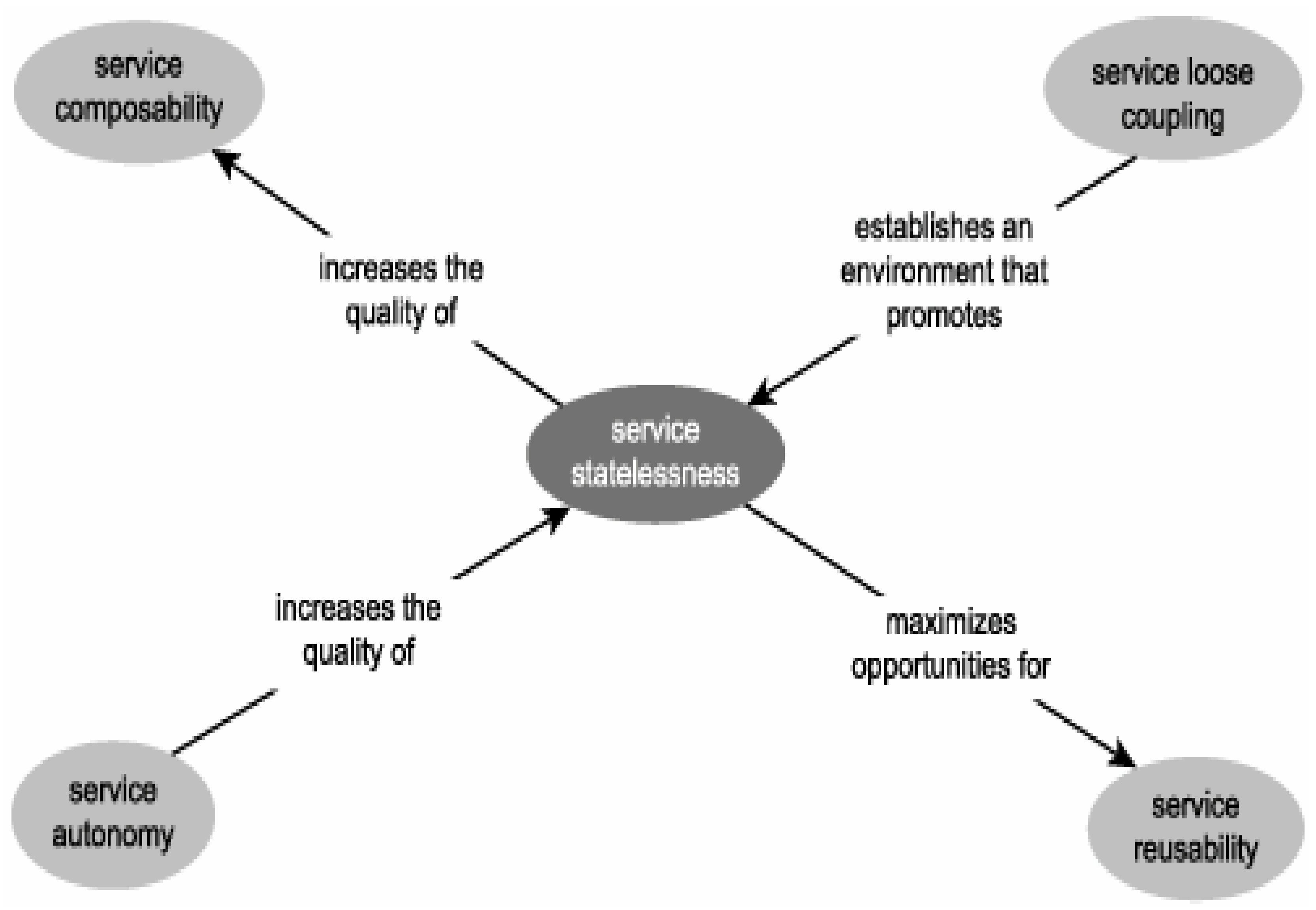
- “To successfully design services not to manage state requires the availability of resources surrounding the service to which state management responsibilities can be delegated.”

Service Statelessness

- **Supported by:**
  - Service autonomy provides the ability for a service to control its own execution environment, which reduces dependencies
  - Service loose coupling establishes a communication paradigm that is fully realized through messaging and messages enable statelessness.

Service Statelessness

- **Supports:**
  - Service composability benefits from stateless composition members, as they reduce dependencies and minimize the overhead of the composition as a whole.
  - Service reuse becomes more of a reality for stateless services, as availability of the service to multiple requestors is increased.


service composability
service loose coupling
increases the quality of
establishes an environment that promotes
service statelessness
increases the quality of
maximizes opportunities for
service autonomy
service reusability

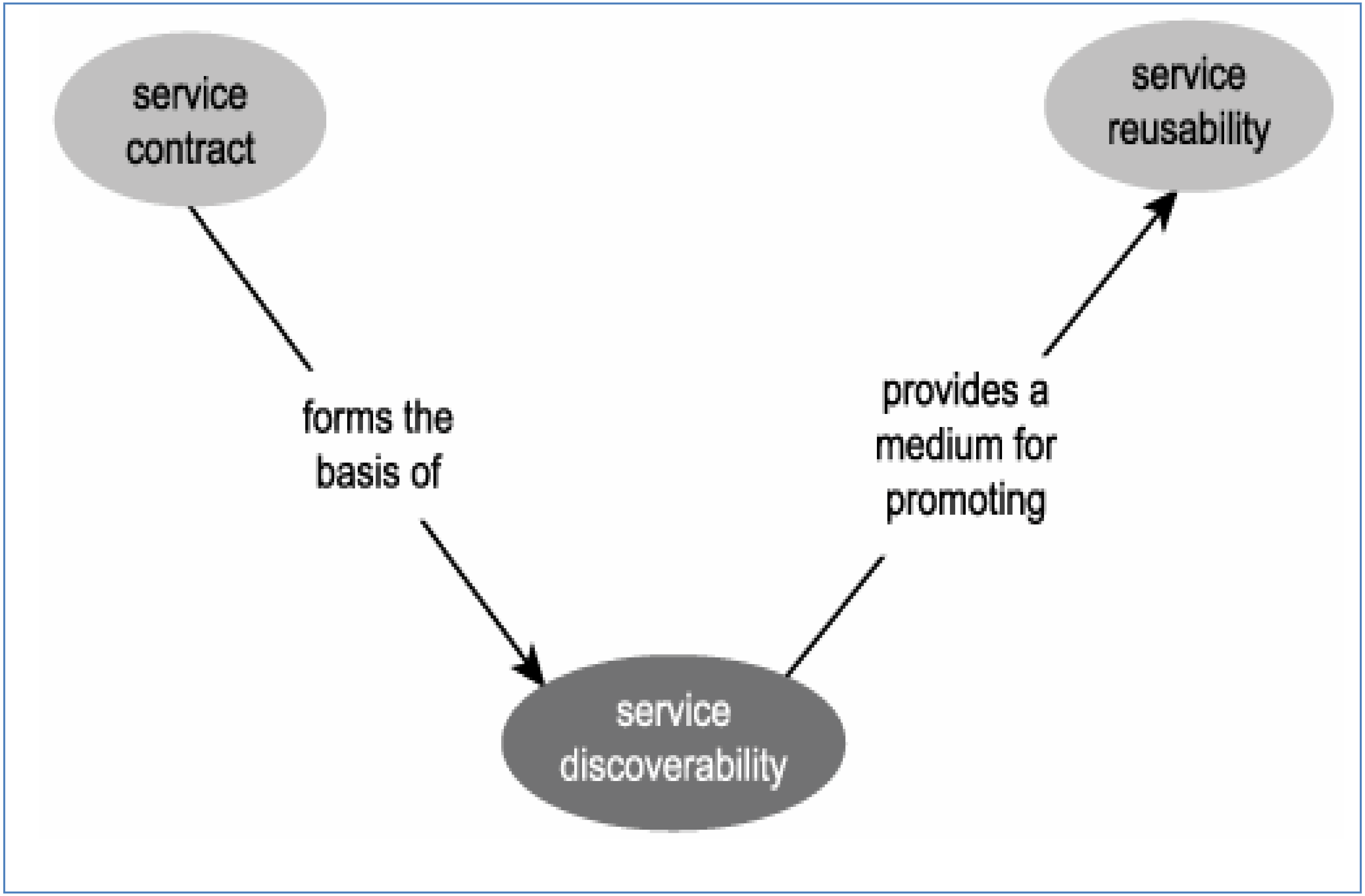
Service Discoverability

- “Designing services so that they are naturally discoverable enables an environment whereby service logic becomes accessible to new potential service requestors.”

Service Discoverability

- **Supported by:**
  - Service contracts are what service requestors (or those that create them) actually discover and subsequently assess for suitability.

- **Supports:**
  - Discoverability is a medium to promote reusability


service contract
forms the basis of
service discoverability
provides a medium for promoting
service reusability

So far,

- Service-orientation principles are ***not realized in isolation***; principles relate to and support other principles in different ways.
- Principles, such as service reusability and service composability, benefit from the support of other implemented principles.
- Principles, such as service loose coupling, service contract, and service autonomy, provide significant support for the realization of other principles.