Service Orientation

مهندسی نرم افزار ۲
ملیحه هاشمی

Service Orientation

- سرویس‌گرایی یک رویکرد برای جداسازی concern هاست که برای حل یک مساله‌ی بزرگ، ابتدا آن را به مجموعه‌ای از عناصر (یا concern ها) که هر کدام بخشی به خصوصی از مساله را در بردارند می‌شکنند. به این ترتیب این concern ها می‌توانند به صورت واحدهای مجزا تولید و در بعضی از موارد نیز جداگانه تست شوند. از این رو می‌توان مساله را به شکل بهتری مدیریت و حل نمود.
- بسیاری از رویکردهای تولید نرم افزار بر مبنای جداسازی concern ها کار می‌کنند مانند object oriented، Aspect oriented. service oriented نیز به یک روش متفاوت به جداسازی concern ها می‌پردازد.

Service Orientation

- سرویس‌گرایی یک پارادایم طراحی می‌باشد که در برگیرنده‌ی مجموعه‌ای از اصول طراحی است.
- در سرویس‌گرایی با به کارگیری این اصول، مساله به مجموعه‌ای از واحدهای منطقی مجزا تقسیم می‌شود، هر یک از این واحدها به طور مستقل تولید می‌شوند و از کنار هم قرار دادن آن‌ها می‌توان به اهداف سازمان دست یافت.
- علاوه بر این امکان استفاده مجدد از این واحدهای منطقی مورد تاکید است.
- این واحدهای منطقی سرویس نام دارند.

Service

- سرویس یک واحد منطقی مستقل با قابلیت استفاده مجدد و طراحی loosely-coupled می‌باشد که یک کارکرد مجزا را encapsulate می‌کند و یک بازنمایی انتزاعی از آن فراهم می آورد.

Service Models

- سرویس‌ها را بر مبنای موارد زیر می‌توان در چند دسته طبقه‌بندی نمود:
  - بر حسب منطق کارکردی که فراهم می‌آورند
  - برحسب میزان قابلیت استفاده مجدد از این منطق سرویس
  - برحسب این‌که سرویس چگونه به دامنه‌های موجود در یک سازمان مرتبط می‌شود.

- بر این مبنا، می‌توان سه دسته اصلی برای سرویس‌ها در نظر گرفت:
  - Entity services
  - Task service
  - Utility services

Service Models
Entity service

- سرویس‌هایی هستند که زمینه کارکردی آن‌ها با یک یا چند موجودیت کسب‌وکاری در ارتباط می‌باشد. برای مثال سرویس‌هایی مانند order، employee، client و ... نمونه‌هایی از این سرویس‌ها می‌باشند.
- به این ترتیب این گونه سرویس‌ها مرتبط با فرآیند کسب‌وکاری به خصوصی نمی‌باشند و می‌توان از آن‌ها به طور مکرر در فرآیندهای کسب‌وکاری و task های مختلف استفاده نمود. بنابراین این سرویس قابلیت استفاده مجدد زیادی دارند.

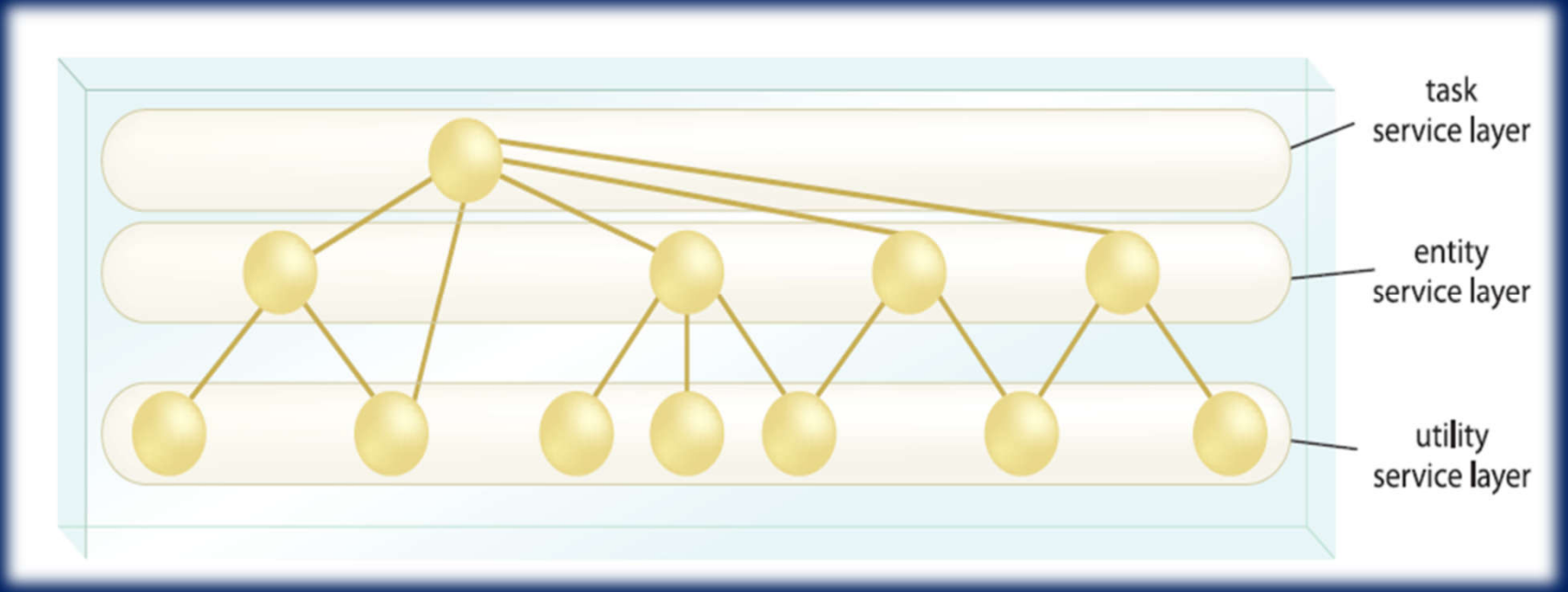
Service Models
Task service

- سرویسی است که منطق کسب‌وکاری یک task یا یک فرآیند کسب‌وکاری به خصوص را encapsulate می‌کند.
- این نوع سرویس، امکان کمتری برای استفاده مجدد دارد.
- این گونه سرویس‌ها معمولا منطق موردنیاز برای ترکیب سرویس‌های دیگر را به منظور تکمیل task موردنظر در بر می‌گیرند. (نقش کنترل‌کننده در ترکیب سرویس‌ها را ایفا می‌کنند.)

Service Models
Utility service

- برخلاف دو نوع قبل، که بر منطق کسب‌وکار تمرکز داشتند، زمینه کارکردی این گونه سرویس‌ها، متمرکز بر کسب‌وکار نمی‌باشد.
- این گونه سرویس‌ها، بر کارکردهای مبتنی بر تکنولوژی تمرکز دارند. برای مثال: logging، notification، security functions، exception handling
- این گونه سرویس‌ها، به کسب‌وکار و کاربرد خاصی وابسته نیستند و امکان استفاده مجدد از آن‌ها تا حد زیای وجود دارد.

Service Models

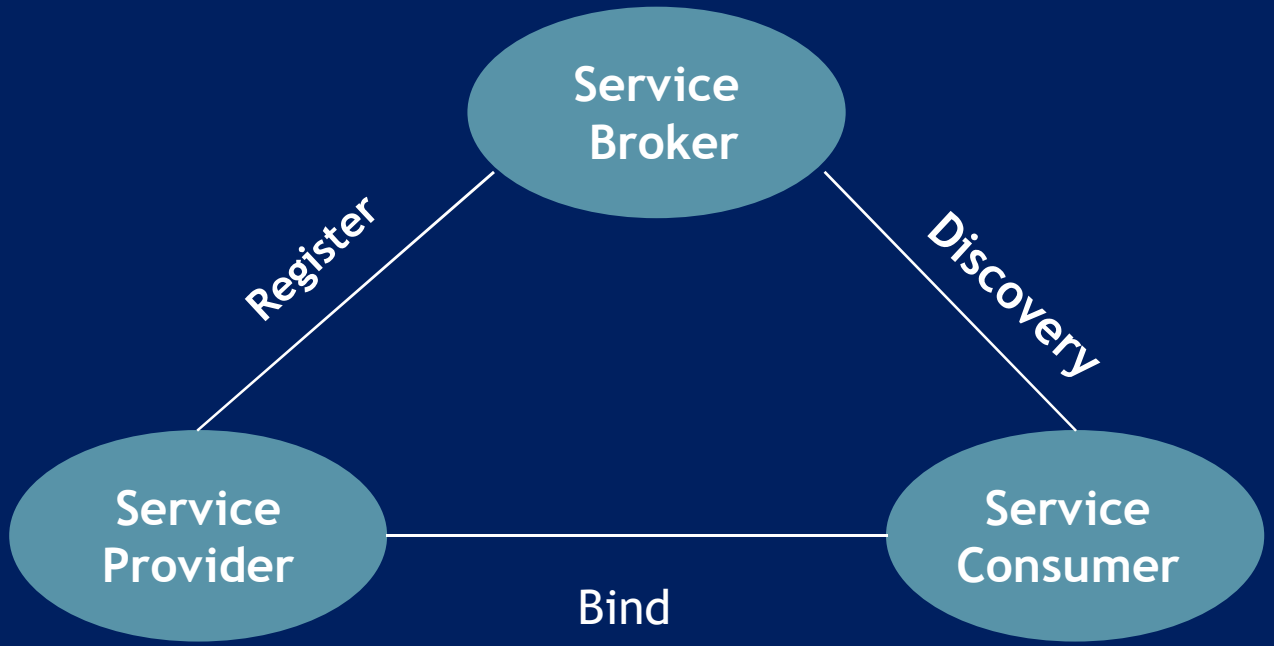


Service Oriented Architecture (SOA)

- معماری سرویس‌گرا، حاصل تلفیق سرویس‌گرایی و معماری نرم‌افزار می‌باشد.
- معماری نرم افزار یک سازمان‌دهی abstract است، که شامل مولفه های تشکیل دهنده، ارتباطات بین این مولفه ها، ویژگی‌های این مولفه ها و قواعد حاکم بر این ارتباطات می‌باشد.
- معماری سرویس‌گرا، کارکردهای سیستم‌های نرم افزاری را به عنوان یک واحد نرم افزاری به نام سرویس کپسوله‌سازی می‌کند و آن‌ها را برای مصرف کنندگان قابل کشف و استفاده می‌سازد.
- آنچه معماری سرویس‌گرا را از سایر معماری‌ها توزیع شده مجزا می‌کند، تاکید بر استقلال واحدهای منطقی تعریف شده و وجود قوانینی برای ایجاد ارتباط در بین آن‌هاست.
- به عبارتی هر یک از این واحدهای منطقی باید بتوانند به طور مستقل از هم تهیه شوند، در حالی که با یک سری از اصول مطابقت داشته و از استانداردهای مشترکی تبعیت کنند.
- معماری سرویس‌گرا از قابلیت‌های سرویس به‌منظور رفع نیازمندی‌های کسب‌وکار استفاده می‌کند.

Service Oriented Architecture (SOA)

- مدل عمومی معماری سرویس گرا



Service Oriented Architecture (SOA)

مدل عمومی معماری سرویس گرا

- فراهم آورنده‌ی سرویس
  - سرویس را ارائه کرده و آن سرویس را به همراه مشخصاتی همچون واسط سرویس به کارگزار سرویس معرفی می‌کند.
  - در نتیجه ی این کار، فراهم کننده سرویس امکان دستیابی به سرویس‌هایش را توسط مصرف کنندگان سرویس فراهم می‌کند.
- مصرف کننده سرویس :
  - نیاز به استفاده از سرویس‌های ارائه‌شده توسط فراهم کنندگان سرویس را دارد.
  - می‌تواند یک برنامه کاربردی یا سرویس دیگر باشد.
  - مصرف کننده می‌تواند از کارگزار درخواست کند تا سرویس مورد نیاز را جستجو کند.
  - در صورتی که سرویس یافت شود کارگزار این امکان را فراهم می آورد که مصرف کننده به فراهم آورنده سرویس متصل شود.

Service Oriented Architecture (SOA)

مدل عمومی معماری سرویس گرا

- کارگزار سرویس
  - به طور کلی مسئولیت ذخیره و بازیابی مشخصات سرویس‌های ارائه‌شده توسط فراهم کنندگان سرویس و فراهم آوردن امکان جستجوی سرویس‌های درخواست شده از طرف مصرف کنندگان سرویس را به عهده دارد.

Common Principle in Service Orientation

Service Reusability

- یکی از مسائل مهم جهت افزایش قابلیت استفاده مجدد یک سرویس عدم وابستگی آن به کسب و کار، تکنولوژی و پلتفرم به خصوصی است.
  - **در این صورت سرویس پتانسیل بیشتری برای استفاده مجدد دارد.**

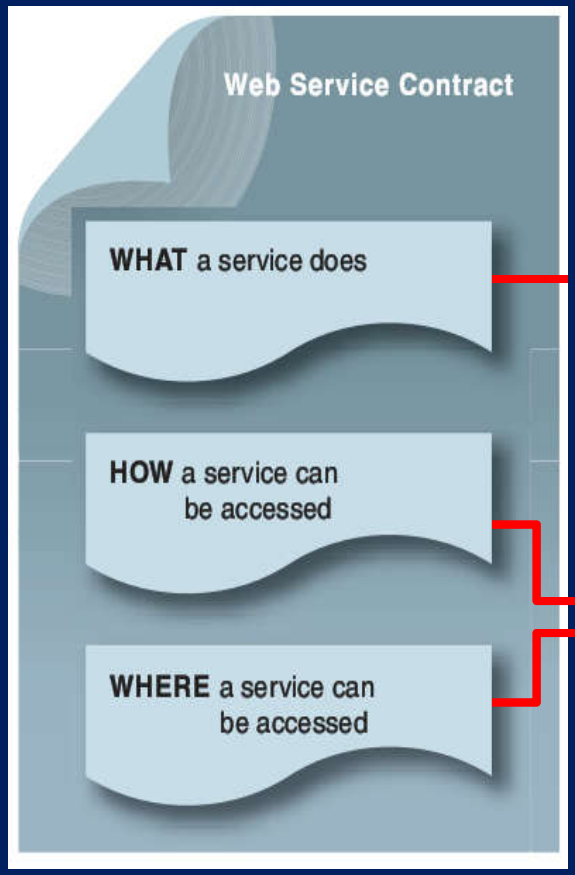
Service Contract

- اهداف و قابلیت‌های سرویس، با استفاده از قرارداد سرویس بیان می‌شود.
- با استفاده از قرارداد سرویس یک سرویس دهنده می‌تواند یک توصیف از عملکرد سرویس خود برای عموم قرار دهد. این اطلاعات باعث به وجود آمدن یک توافق بین فراهم آورنده‌ی سرویس و درخواست کننده‌های آن می‌شود.
- اطلاعاتی که در قرار داد سرویس بیان می شوند، عبارتند از:
  - اطلاعات مرتبط با تکنولوژی
  - اطلاعات مرتبط با کارکرد و عملیات
  - اطلاعات مرتبط با نحوه و منطق عملکرد
  - اطلاعات مرتبط باکیفیت سرویس

Web Service Contract

- مستندات توصیف‌کننده‌ی قرارداد برای سرویسی که به عنوان یک سرویس وب پیاده‌سازی شده است، در شکل زیر نمایش داده شده‌اند:

Web Service Contract
Fundamental Structure



Abstract description

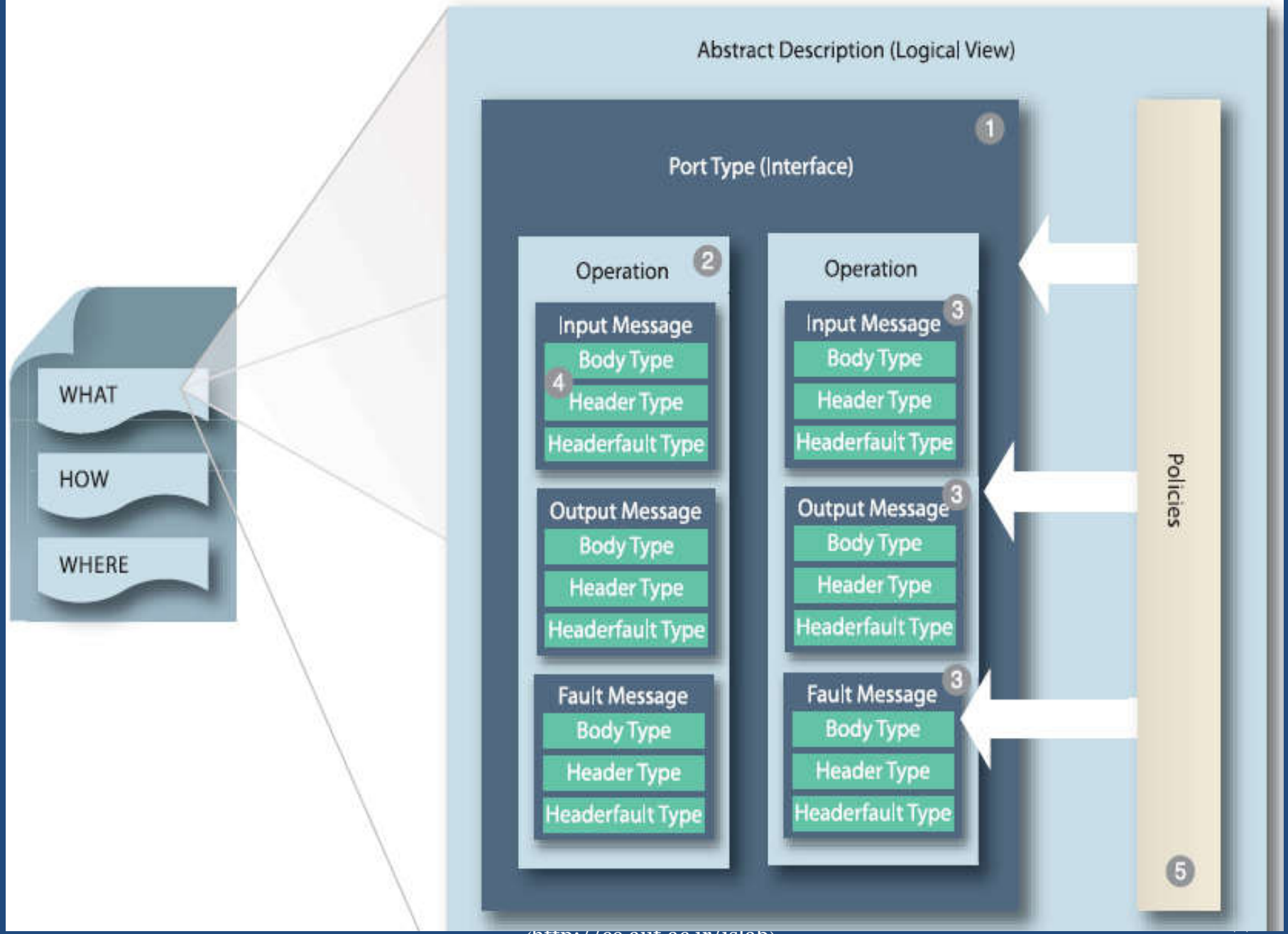
Concrete definition

What	اهداف و قابلیت‌های این سرویس چیست؟
HOW	چگونه می‌توان به این سرویس دسترسی داشت؟
Where	این سرویس در کجا قرار دارد؟

این بخش اساسا مسئول ارائه واسط عمومی قرارداد است.

- **این جداسازی این امکان را فراهم می‌کند تا بخش‌های مختلف یک سرویس در مراحل متفاوتی از چرخه حیات و توسط تیم‌های مختلف تولید شوند.**

این بخش جزئیات پیاده‌سازی و ارتباط با سرویس را فراهم می‌آورد که مورد نیاز مصرف کنندگان جهت دسترسی و استفاده از سرویس است.


Abstract Description (Logical View)
Port Type (Interface)
1
Operation
2
Input Message
Body Type
4
Header Type
Headerfault Type
Output Message
Body Type
Header Type
Headerfault Type
Fault Message
Body Type
Header Type
Headerfault Type
Operation
Input Message
3
Body Type
Header Type
Headerfault Type
Output Message
3
Body Type
Header Type
Headerfault Type
Fault Message
3
Body Type
Header Type
Headerfault Type
Policies
5
WHAT
HOW
WHERE

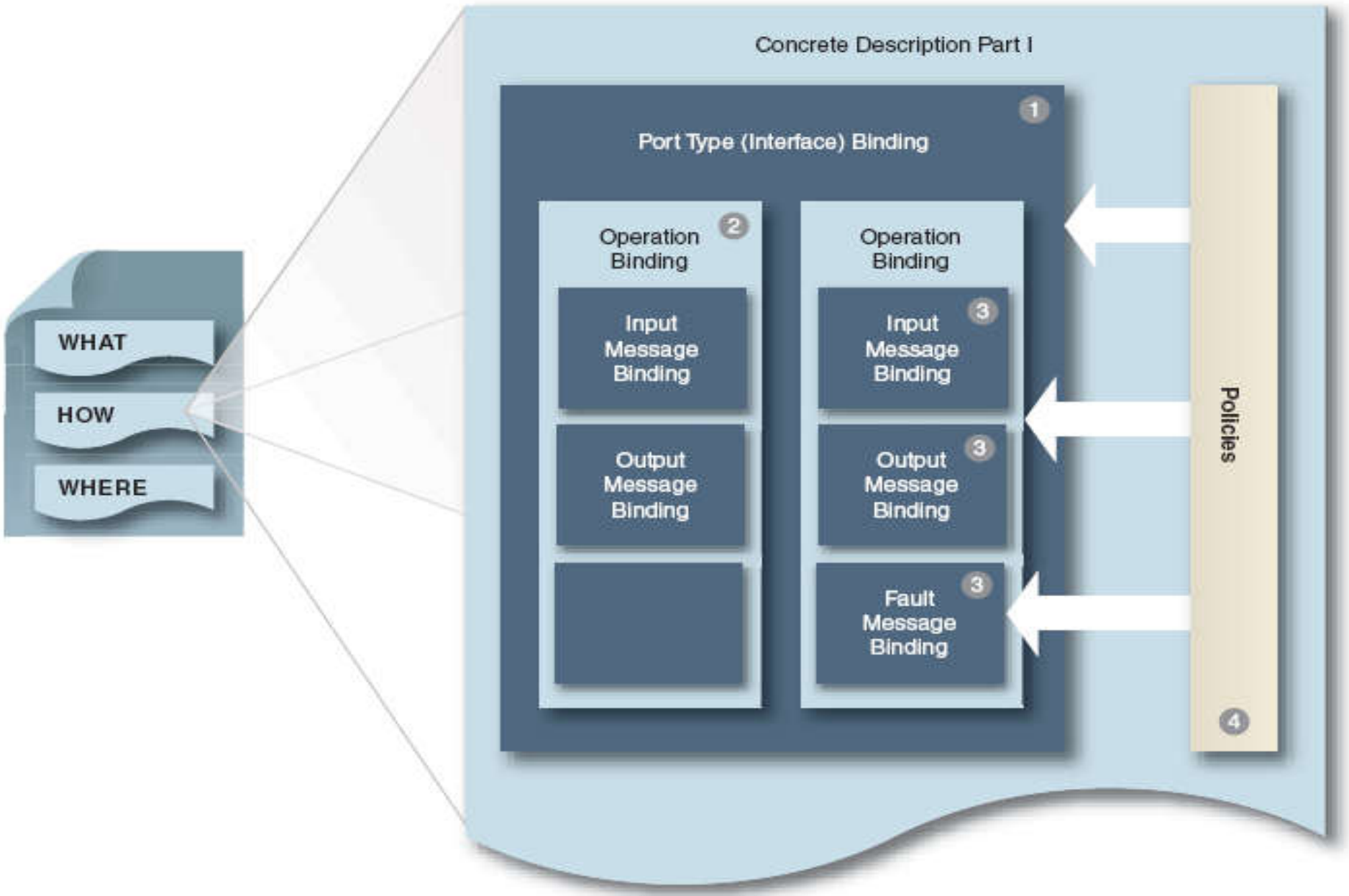
Web Service Contract
Fundamental Structure

- Port type
  - همانند interface در مفهوم شی‌گرایی است. در اینجا port type مجموعه ای از operationها را در بر می‌گیرد که برای هر operation می‌توان پیغام‌های ورودی و خروجی آن را تعریف نمود.
- operation
  - هر operation نشان دهنده‌ی عملیات به خصوصی است که توسط سرویس انجام می‌پذیرد. عملیات سرویس را می‌توان با مفهوم متد در شی گرایی مقایسه نمود. مانند متد، عملیات نیز دارای پارامترهای ورودی و خروجی می‌باشد. با توجه به اینکه در سرویس‌گرایی، ارتباطات تنها مبتنی بر پیغام است، در این‌جا پارامترها پیغام‌ها هستند. بنابراین هر عملیات شامل یک مجموعه از پیغام‌های ورودی و خروجی می‌باشد.
- یک port type مانند یک container برای یک مجموعه از عملیات‌های مرتبط با هم، عمل می‌کند.

Web Service Contract
Fundamental Structure

Message

- نوع پیغام‌ها می‌تواند به صورت ورودی، خروجی یا خطا باشند.
- هر پیغام از چندین بخش تشکیل می‌شود:
  - **Body**: ساختاری از پیش تعیین شده برای پیغام
  - **Header**: یک metadata که اطلاعاتی درباره‌ی پیغام است و پیغام را همراهی می‌کند.
  - **Header fault**: ممکن است انتقال پیغام یا پردازش فراداده با خطا مواجه شود، بنابراین لازم این نوع پیغام زمانی که باید پاسخی به یک شرایط خطا فرستاده شود، مورد استفاده قرار گیرد.


Concrete Description Part I
1
Port Type (Interface) Binding
Operation Binding
2
Input Message Binding
Output Message Binding
Operation Binding
Input Message Binding
3
Output Message Binding
3
Fault Message Binding
3
Policies
4
WHAT
HOW
WHERE

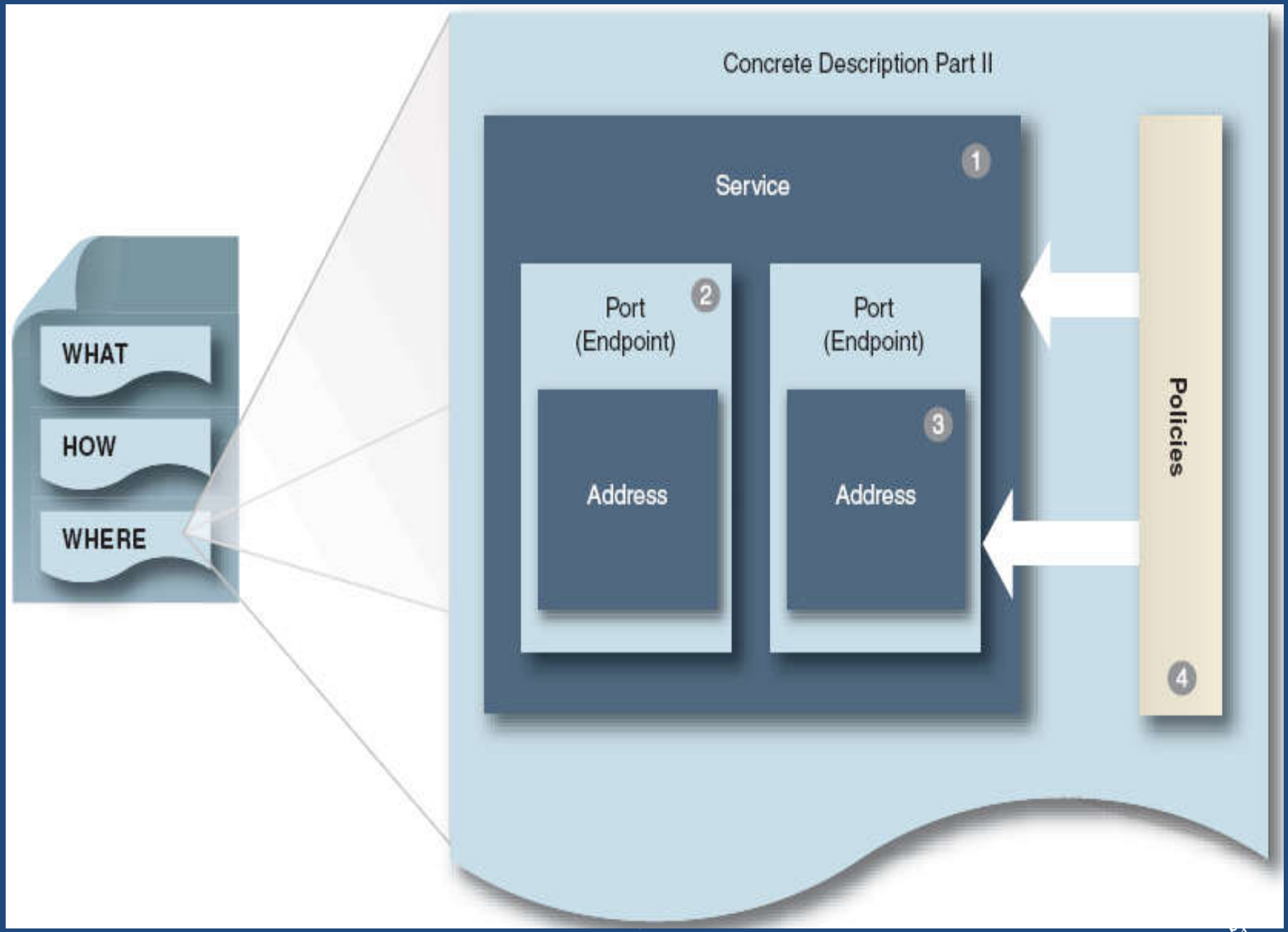
Web Service Contract
Fundamental Structure

Port Type Binding

- تعریف binding، مشخص‌کننده‌ی جزئیات تکنولوژی ارتباطی‌ای است که توسط مصرف‌کننده‌ی سرویس جهت فراخوانی و تعامل با آن مورد استفاده قرار می‌گیرد.
- تعریف port type در توصیف انتزاعی را می‌توان به یک یا چند تعریف bindingمرتبط نمود، که هرکدام شامل موارد زیر هستند:
  - پروتکل پیغام‌رسانی: مشخص کننده‌ی استاندارد قالب پیغام
  - پروتکل انتقال: مشخص کننده‌ی تکنولوژی ارتباطی برای انتقال پیغام در طول شبکه

Web Service Contract
Fundamental Structure

- operation binding definition
  - این تعریف همانند port type binding بوده اما محدوده‌ی آن به یک operationمربوط می‌شود.
  - اگر این binding برای یک operation تعریف نشده باشد، این operation تنظیمات مربوط به binding را از port type پدر خود به ارث می‌برد.
- Message binding definition:
  - این تعریف همانند دو تعریف قبل است اما در محدوده‌ی یک پیغام تعریف می‌شود.
  - در صورت عدم وجود این تعریف برای یک پیغام، تعریف binding آن از operationپدر و در صورت عدم وجود آن نیز از port type پدر به ارث می‌رسد.


Concrete Description Part II
Service
1
Port
(Endpoint)
2
Address
Port
(Endpoint)
3
Address
Policies
4
WHAT
HOW
WHERE

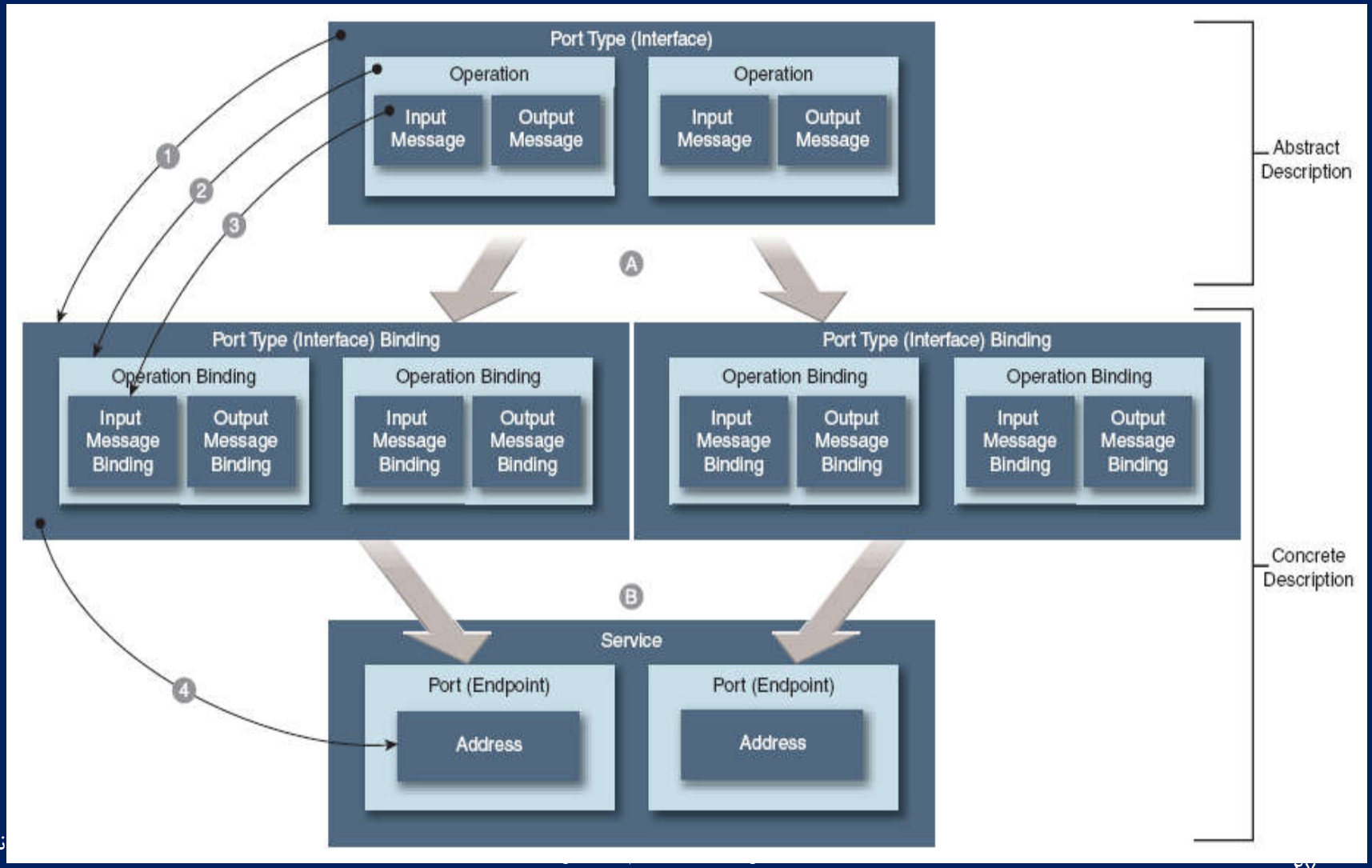
Web Service Contract
Fundamental Structure

Port (Endpoint)

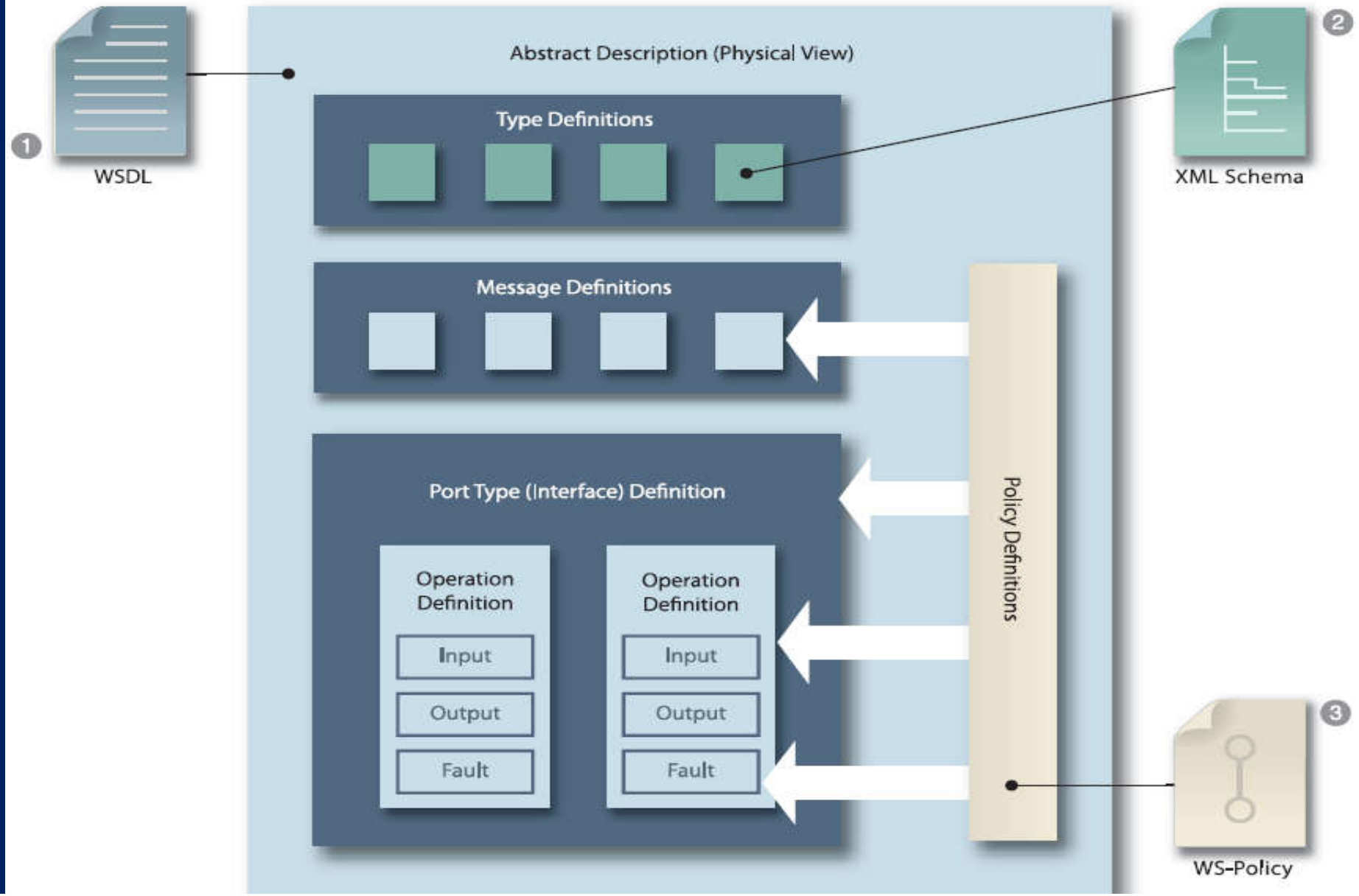
- هر تعریف binding باید به یک تعریف port مرتبط شود.
- هر port مانند یک container برای تعریف آدرس فیزیکی عمل می‌کند.
- هر تعریف port می‌تواند توسط یک port type، operationیا message مورد استفاده قرار گیرد.

Address definition

- مشخص کننده‌ی یک آدرس فیزکی در شبکه می‌باشد. این آدرس برای مثال می‌تواند یک URLباشد.


Port Type (Interface)
Operation
Input Message
Output Message
Operation
Input Message
Output Message
Abstract Description
1
2
3
A
Port Type (Interface) Binding
Operation Binding
Input Message Binding
Output Message Binding
Operation Binding
Input Message Binding
Output Message Binding
Port Type (Interface) Binding
Operation Binding
Input Message Binding
Output Message Binding
Operation Binding
Input Message Binding
Output Message Binding
Concrete Description
B
Service
Port (Endpoint)
Address
Port (Endpoint)
Address
4

Technologies in Web service contract



Technologies in Web service contract

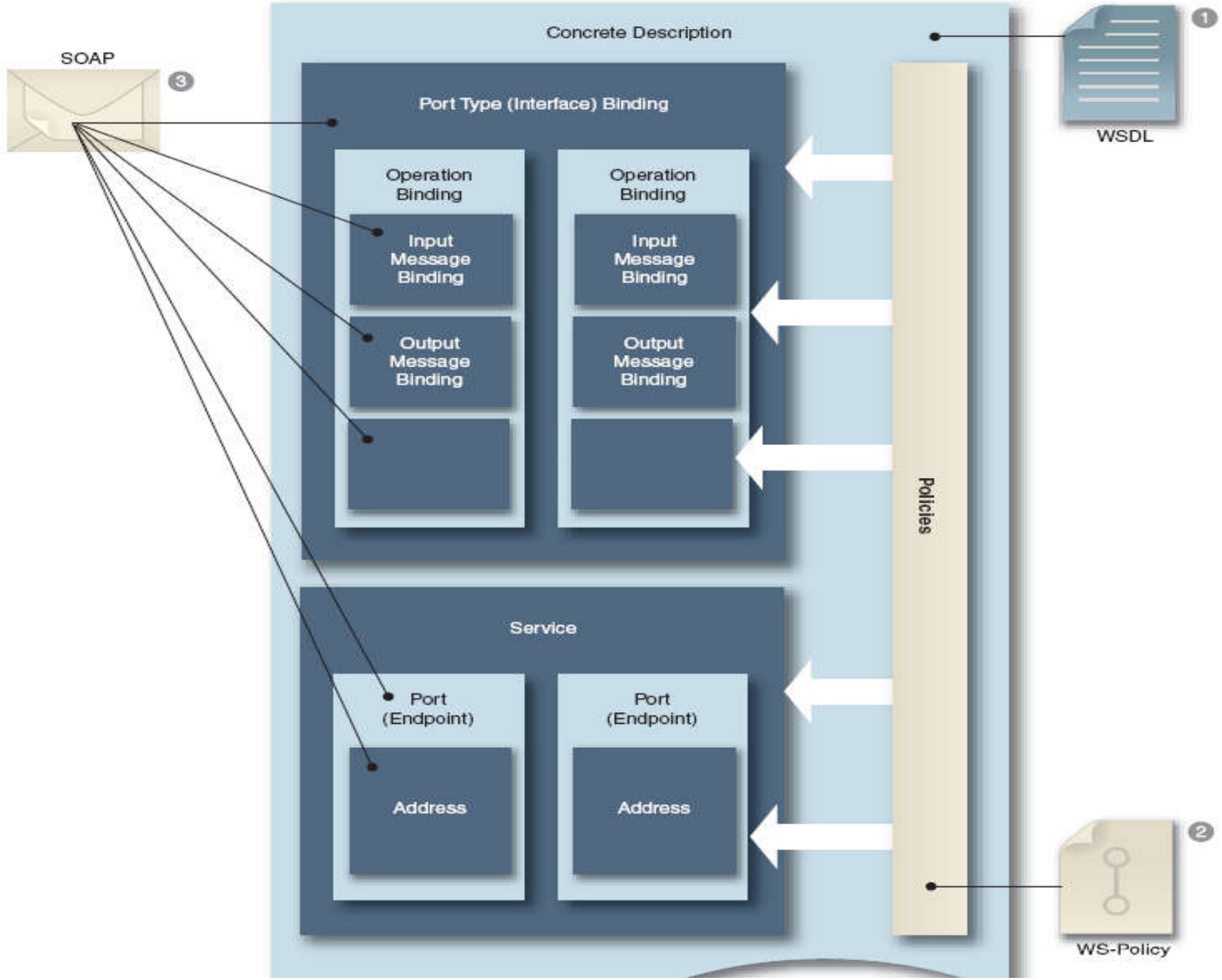
- WSDL مهم‌ترین و اساسی‌ترین تکنولوژی برای تعریف قرارداد سرویس است. با استفاده از WSDL ساختار قرارداد سرویس تعریف می‌شود و سپس با استفاده از تکنولوژی‌های دیگر جزئیات آن مشخص و بسط داده می‌شود.
- در حقیقت ساختاری که تاکنون برای قرارداد سرویس مشخص کردیم، تعریف WSDL را نشان می دهد.
- WSDL هم تعریف abstract و هم concrete را در بر می‌گیرد.

Technologies in Web service contract

- XML Scheme زبانی برای تعریف ساختار و اعتبارسنجی مستندات XML فراهم می‌آورد. با توجه به این‌که سرویس‌های وب با تبادل پیغام‌های مبتنی بر XML با یکدیگر ارتباط برقرار می کنند، XML یک تکنولوژی تکمیل‌کننده برای تولید قرارداد سرویس فراهم می‌آورد.
- Scheme ها به عنوان بخشی از قرارداد به شمار می‌روند و جزئیات ساختاری آن را توصیف می‌کنند. در حقیقت آن ها مشخص‌کننده‌ی عناصر و صفات ظاهر شونده در یک پیغام، ترتیب این اجزا و نوع داده‌ای هستند که این اجزا می توانند شامل شوند.
- کد XML Scheme را می‌توان به صورت مستقیم در مستند WSDL وارد نمود یا به صورت یک مستند جداگانه از آن بهره گرفت.

Technologies in Web service contract

- WS-Policy : استاندارد WS-Policy یک چارچوب مختص سرویس وب است که امکان بـیان ویژگی‌ها و قیود رفتاری سرویس را به عنوان یک فراداده‌ی machine readable فراهم می آورد. به این ترتیب می توان با استفاده از این زبان قرارداد سرویس را جهت بیان سایر نیازمندی‌های سرویس توسعه داد.


Concrete Description
SOAP
3
Port Type (Interface) Binding
Operation Binding
Input Message Binding
Output Message Binding
Operation Binding
Input Message Binding
Output Message Binding
Policies
1
WSDL
Service
Port (Endpoint)
Address
Port (Endpoint)
Address
2
WS-Policy

Technologies in Web service contract

- SOAP یک استاندارد قالب پیغام است که جهت تبادل پیغام بین برنامه های کامپیوتری استفاده می شود.
- SOAP رایج‌ترین پروتکل قالب پیغام است که همراه با WSDL مورد استفاده قرار می‌گیرد.
- پیغام های SOAP معمولا با استفاده از HTTP ارسال می شوند.

Service Contract

- با توجه به این‌که این قرارداد میان سرویس‌ها به اشتراک گذاشته می‌شود، طراحی آن دارای اهمیت زیادی است.
- درخواست کننده‌های سرویس که با این قرارداد توافق کرده‌اند، به این تعریف وابسته می‌شوند. بنابراین یکی از مواردی که باید مورد توجه قرار گیرد **نگه داری و نسخه بندی قراردادها** بعد از اولین ترخیص می‌باشد.

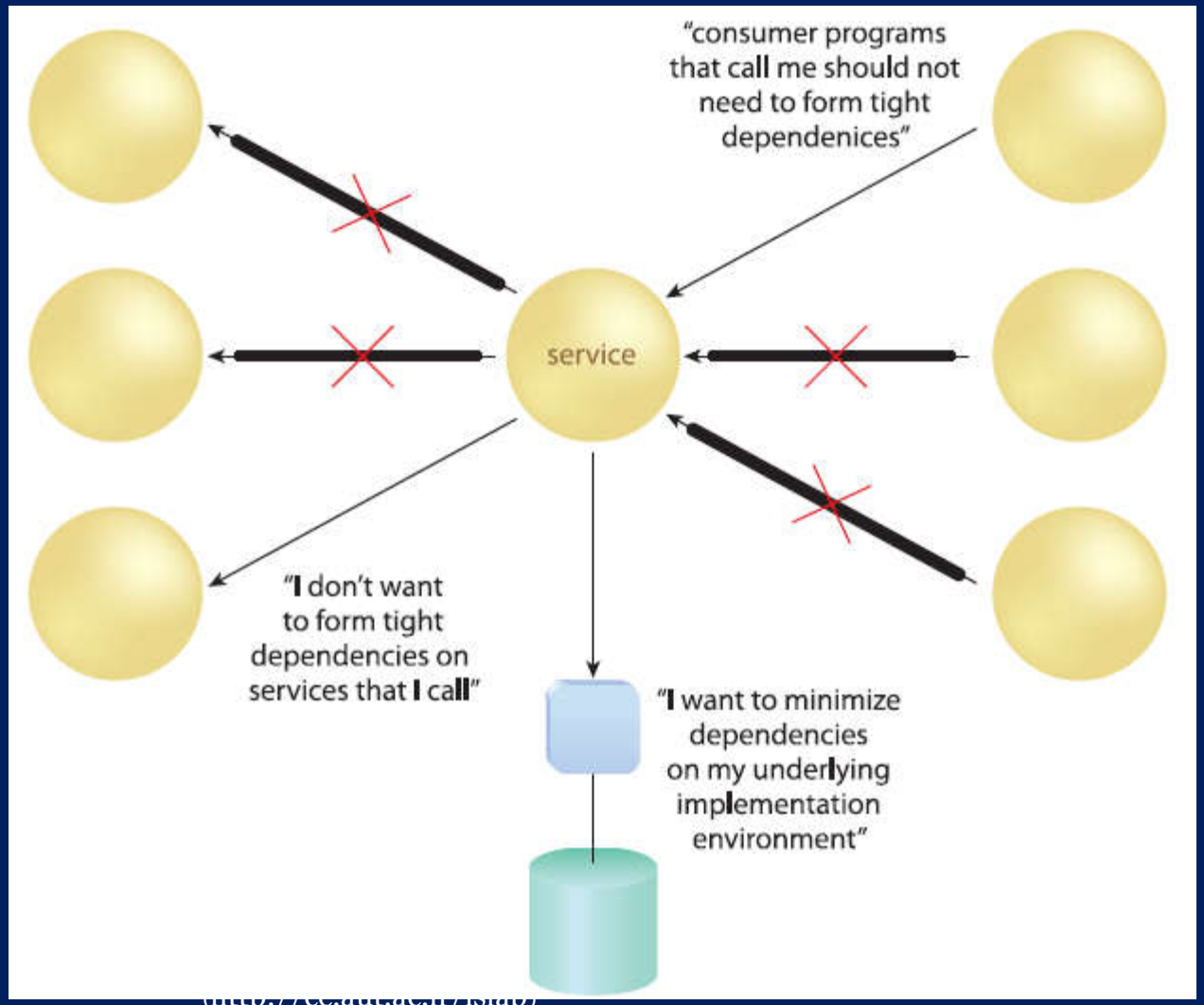
Service Loose Coupling

- اتصالات سست
  - یکی از اصول طراحی است که تضمین می کند قرارداد سرویس با مصرف کنندگان این سرویس، منطق سرویس و پیاده سازی آن دارای اتصالات تنگاتنگ (tightly coupling) نمی‌باشد. در نتیجه قرارداد سرویس بدون این که بر مصرف کنندگان سرویس یا پیاده‌سازی آن تاثیرگذار باشد می تواند تغییر و تکامل یابد.

Service Loose Coupling

این اصل از اصول طراحی بر کاهش اتصالات بین اجزای یک راه حل مبتنی بر سرویس گرایی اشاره دارد. خصوصا فراهم آوردن اتصالات سست بین اجزای زیر مورد تاکید است:

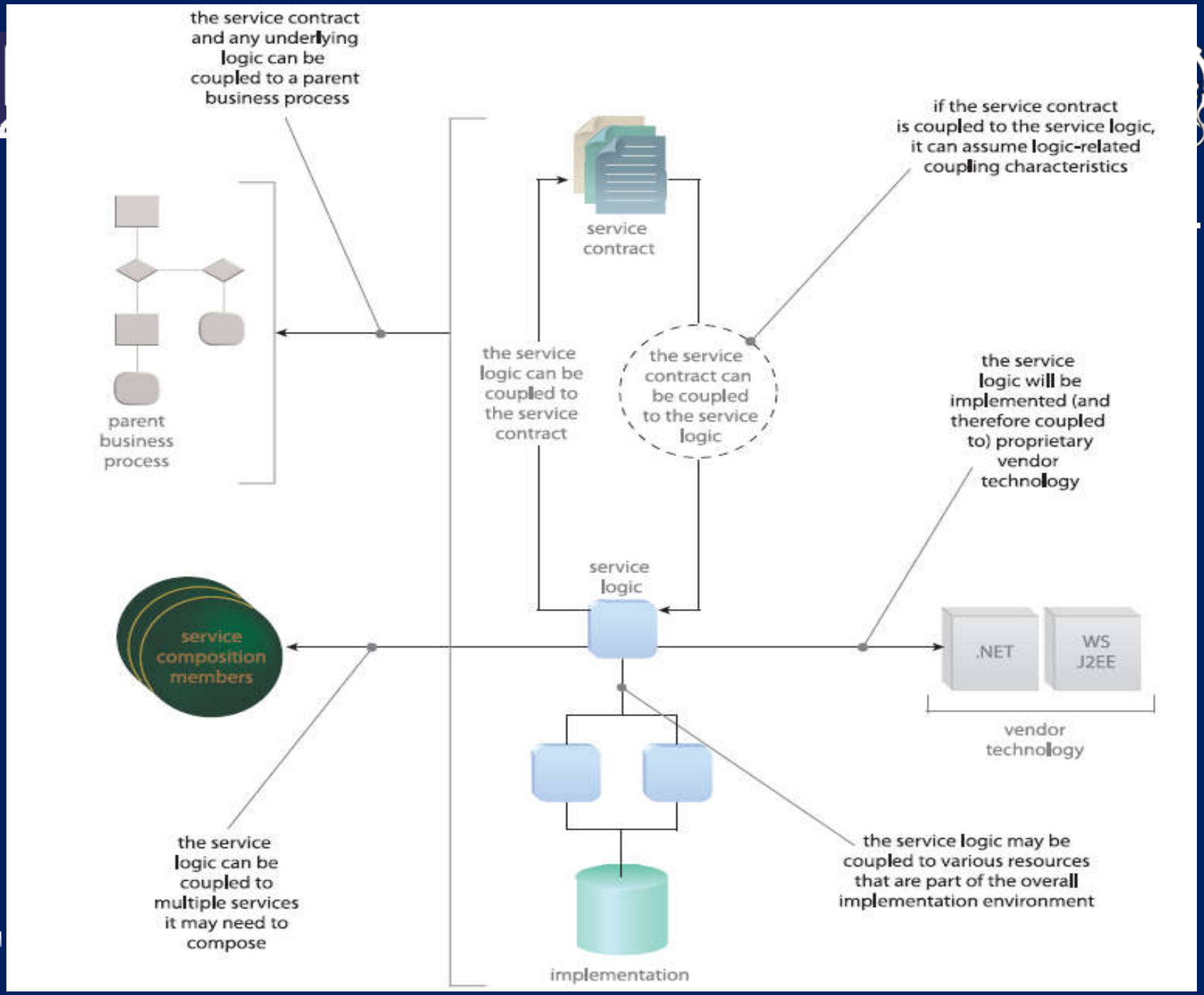
- بین قرارداد سرویس و مصرف کنندگان آن
- بین قرارداد سرویس، با منطق و پیاده سازی آن



Coupling Types

- Service Contract Coupling Types (intra-service coupling)
  - Logic-to-Contract
  - Contract-to-Logic
  - Contract-to-Implementation
  - Contract-to-technology
  - Contract-to-Functional
- Service Consumer Coupling Types (inter-service coupling)
  - Consumer-to-Implementation
  - Consumer-to-Contract

قرارداد سرویس عنصر اصلی است که اتصالات با آن شکل می‌گیرد.

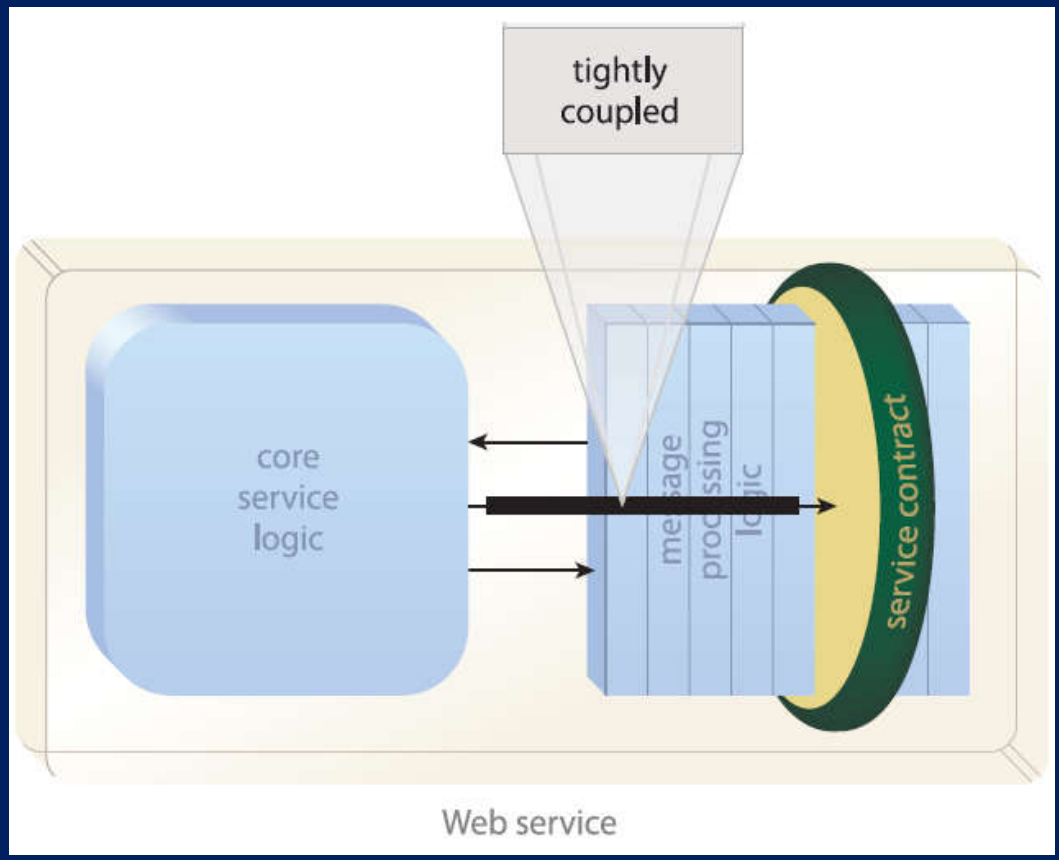

the service contract and any underlying logic can be coupled to a parent business process
if the service contract is coupled to the service logic, it can assume logic-related coupling characteristics
service contract
parent business process
the service logic can be coupled to the service contract
the service contract can be coupled to the service logic
the service logic will be implemented (and therefore coupled to) proprietary vendor technology
service logic
service composition members
.NET
WS J2EE
vendor technology
the service logic can be coupled to multiple services it may need to compose
the service logic may be coupled to various resources that are part of the overall implementation environment
implementation

Logic-to-Contract Coupling

- یکی از رویکردهای مورد توصیه، تولید قرارداد سرویس پیش از منطق آن است. به این ترتیب منطق زیرین به نوعی تهیه می‌شود که از قرارداد سرویس پشتیبانی کند.
- این رویکرد، موجب ایجاد اتصال تنگاتنگ منطق سرویس با قرارداد آن می‌باشد. با این وجود، این نوع از اتصال یک نوع مثبت از اتصال به شمار می‌رود.

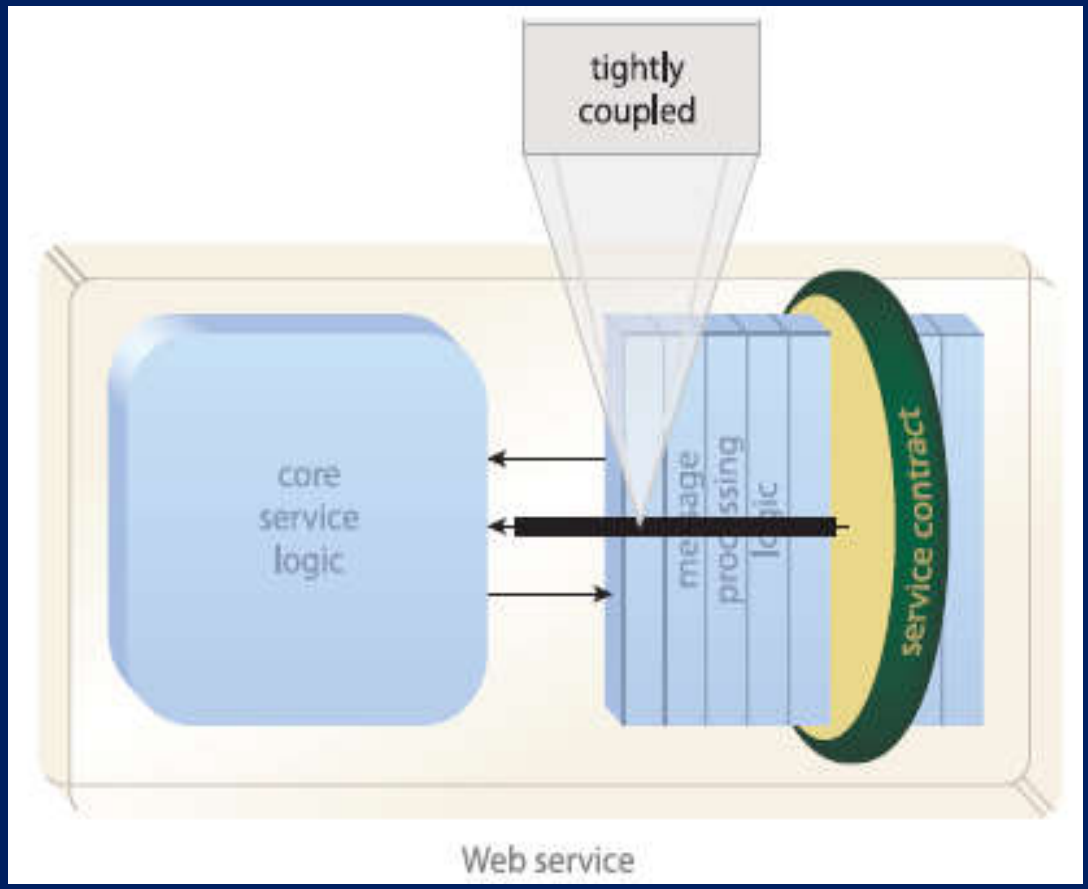
Logic-to-Contract Coupling

- در صورتی که ابتدا قرارداد سرویس تهیه شود و منطق سرویس بر مبنای آن ایجاد گردد؛ **منطق سرویس** دارای یک **اتصال تنگاتنگ** با **قرارداد سرویس** است.
- ✓ در این حالت قرارداد به منطق سرویس وابسته نبوده و به این ترتیب منطق سرویس در آینده می‌تواند جایگزین شود بدون اینکه مصرف کنندگان که به قرارداد وابسته هستند تحت تاثیر قرار گیرند.



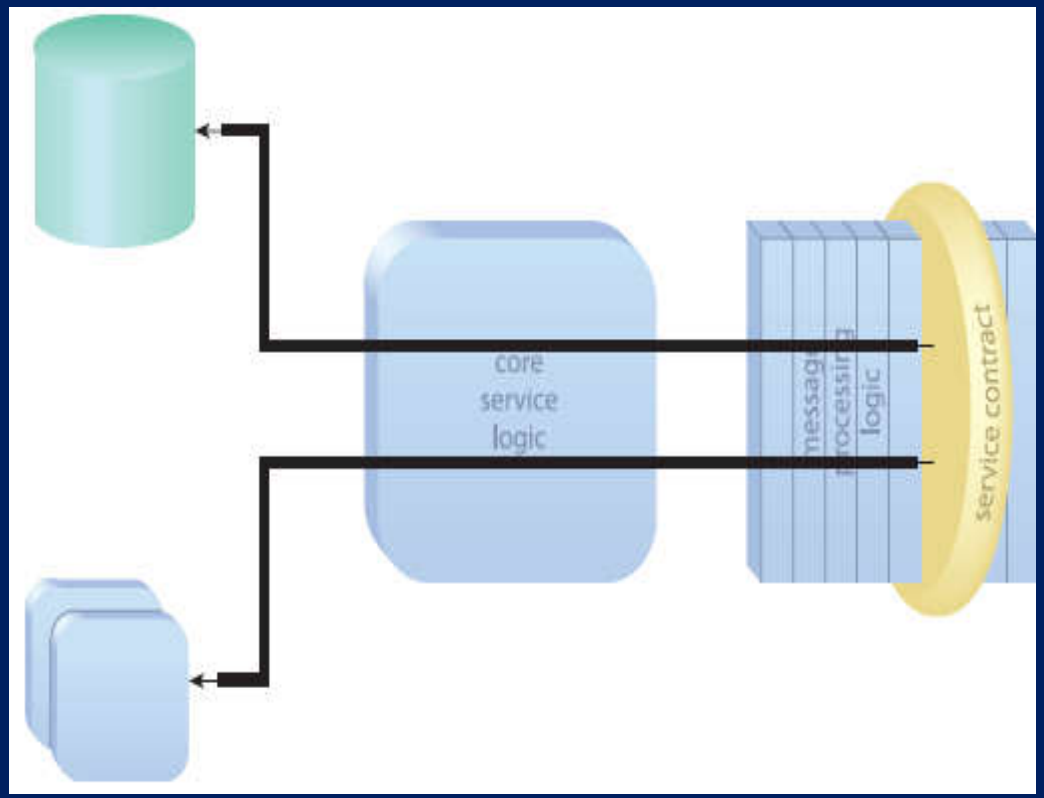
Contract-to-Logic Coupling

- در بسیاری از موارد به خصوص زمانی که منطق راه‌حل موجود است، قرارداد سرویس بر مبنای این منطق حاصل می‌گردد.
- تولید قرارداد معمولا با استفاده از ابزارهایی به صورت خودکار انجام می‌شود.
- در این حالت **قرارداد** دارای **اتصال تنگاتگ** با **منطق سرویس** است، در صورتی که منطق سرویس تغییر کند یک قرارداد جدید ایجاد می‌شود و یک نسخه جدید از سرویس منتشر می‌شود.
- ✘ این امر موجب ایجاد اتصالات زیادی بین مصرف‌کنندگان سرویس با آن می گردد.



Contract-to-Implementation Coupling

- منطق زیرین یک سرویس با منابع مورد نیاز خود مانند پایگاه داده، یا سایر منابع سیستمی در تعامل است. در صورتی که قرارداد سرویس از منطق خود که دارای چنین ارتباطاتی با محیط پیاده‌سازی است به دست آید، جزئیات پیاده‌سازی در قرارداد سرویس وارد می‌شوند.
- این امر موجب اتصال قرارداد سرویس با پیاده‌سازی خود می‌شود.



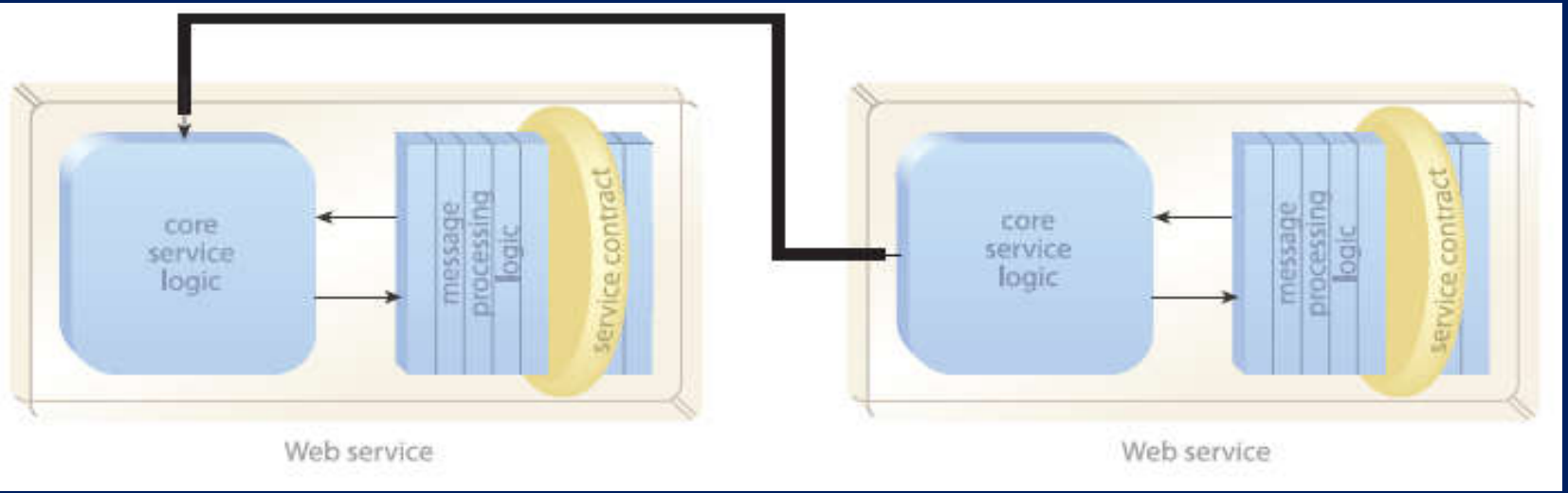
Contract-to-technology Coupling

- زمانی‌که قرارداد سرویس شامل جزئیات تکنولوژی مورد استفاده در تولید سرویس است، این نوع از اتصال ایجاد می‌گردد.
- این نوع از اتصال دارای تاثیرات منفی زیر است:
  - این امر موجب محدود نمودن مصرف کنندگان سرویس می‌شود. زیرا در این حالت مصرف کنندگان باید از تکنولوژی‌های ارائه شده پشتیبانی نمایند.
  - علاوه بر این تولیدکنندگان سرویس نمی‌توانند به راحتی تکنولوژی‌های به کار فته در پیاده سازی سرویس را تغییر دهند.

Contract-to-Functional Coupling

- این نوع از اتصال در حالات زیر ایجاد می‌گردد:
  - زمانی‌که منطق سرویس به منطق فرآیند کسب‌وکاری وابسته باشد.
  - زمانی‌که سرویس تنها برای یک مصرف کننده خاص تولید می گردد.

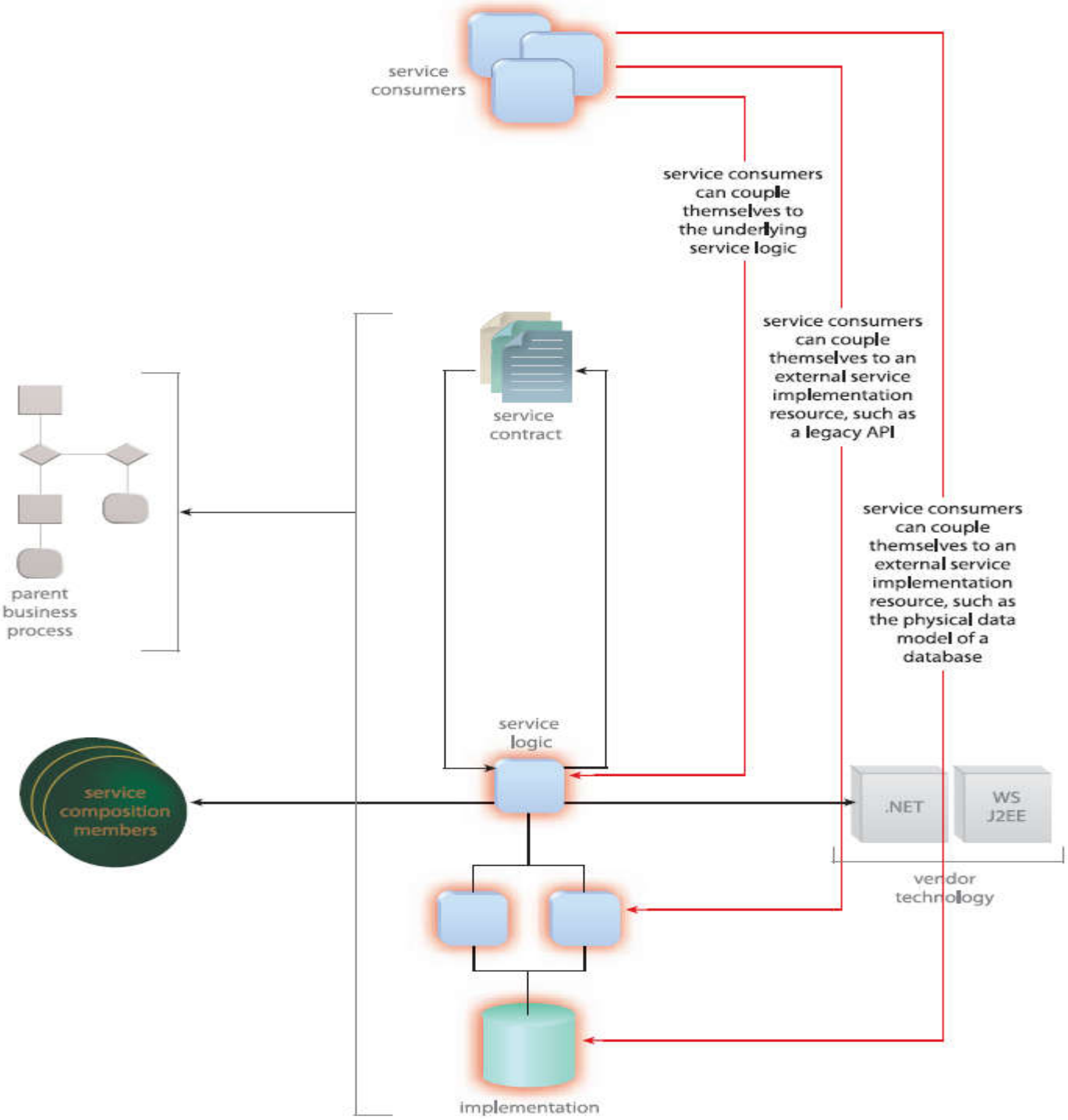
Consumer-to-Implementation Coupling

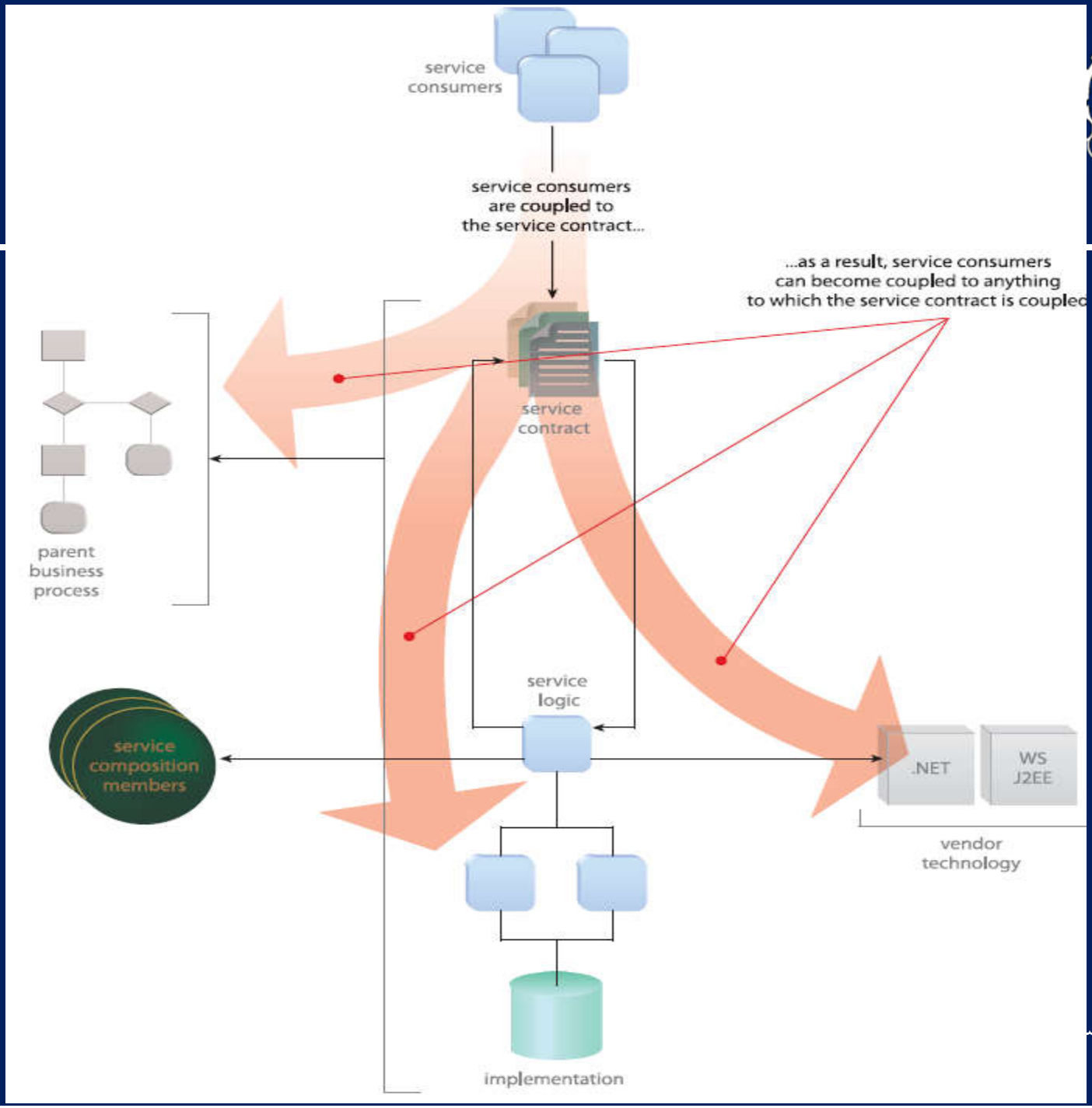


- مصرف کننده‌ی سرویس، قرارداد سرویس را نادیده می‌گیرد و مستقیما به منطق سرویس دسترسی پیدا می‌کند (برای مثال جهت افزایش کارایی) و با آن دارای **اتصال تنگاتنگ** است.

Consumer-to-Contract Coupling

- زمانی شکل می‌گیرد که مصرف‌کننده‌ی سرویس با استفاده از قرارداد سرویس به آن دسترسی پیدا می‌کند.
- این نوع از اتصال، شیوه‌ی مطلوب در ارتباط بین مصرف‌کننده سرویس و سرویس به شمار می‌رود، زیرا بیشترین میزان استقلال بین این دو را فراهم می‌آورد.
- البته این میزان استقلال وابسته به محتوای قرارداد سرویس است. به عبارتی مصرف‌کننده سرویس به هر آن‌چه که قرارداد سرویس با آن در اتصال است وابسته می‌باشد.


service consumers
service consumers can couple themselves to the underlying service logic
service consumers can couple themselves to an external service implementation resource, such as a legacy API
service consumers can couple themselves to an external service implementation resource, such as the physical data model of a database
service contract
parent business process
service logic
service composition members
.NET
WS J2EE
vendor technology
implementation


service consumers
service consumers are coupled to the service contract...
...as a result, service consumers can become coupled to anything to which the service contract is coupled
service contract
parent business process
service logic
service composition members
.NET
WS J2EE
vendor technology
implementation

Coupling types and their influences

Coupling Type	تاثیر منفی یا مثبت؟
Logic-to-Contract	+
Contract-to-Logic	–
Contract-to-Technology	–
Contract-to-Functional	–
Contract-to-Implementation	–
Consumer-to-Implementation	–
Consumer-to-Contract	+

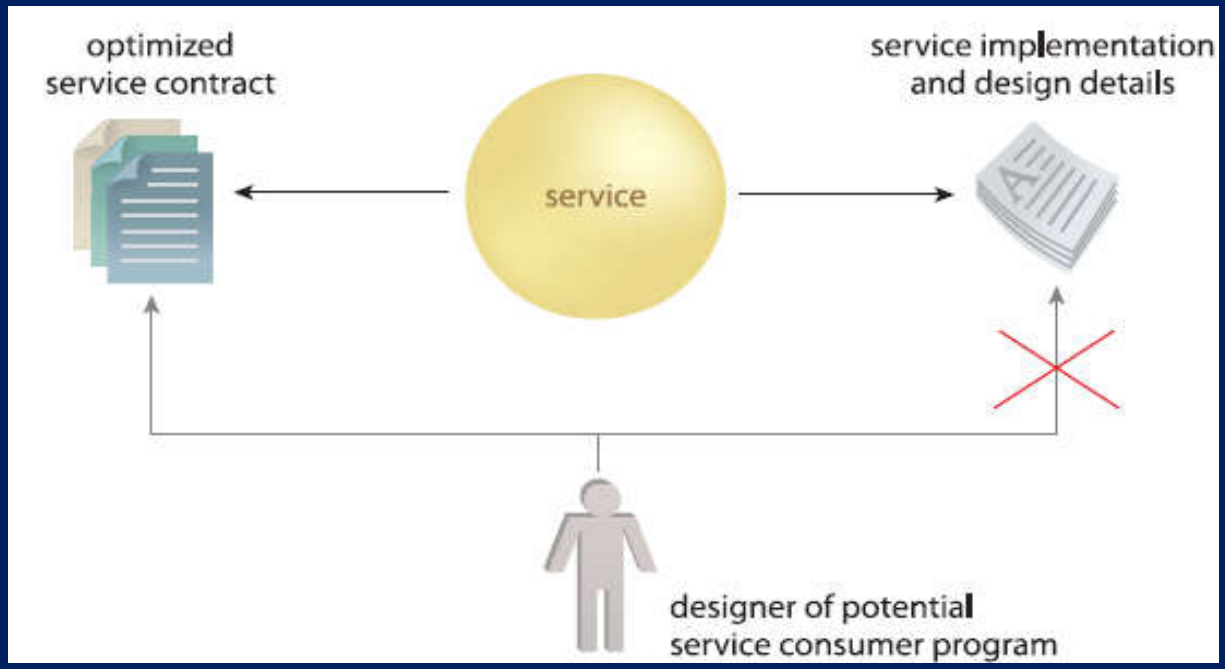
Service Abstraction

- تنها بخشی از سرویس که توسط مصرف‌کنندگان آن قابل مشاهده است، قرارداد سرویس است. منطق سرویس فراتر از آنچه در این قرارداد آمده است از دید مصرف‌کنندگان مخفی بوده و به آن‌ها مربوط نمی‌باشد.
- با مخفی نگه داشتن جزئیات به خصوص، امکان تکامل منطق و پیاده‌سازی سرویس در طول زمان وجود دارد، در حالی‌که هم‌چنان تمام آنچه در قرارداد سرویس ارائه شده است را برای مصرف کنندگان خود فراهم می‌آورد.
- به عبارتی سرویس‌ها مانند یک جعبه سیاه عمل کرده و جزئیات داخلی خود را مخفی می‌سازند.

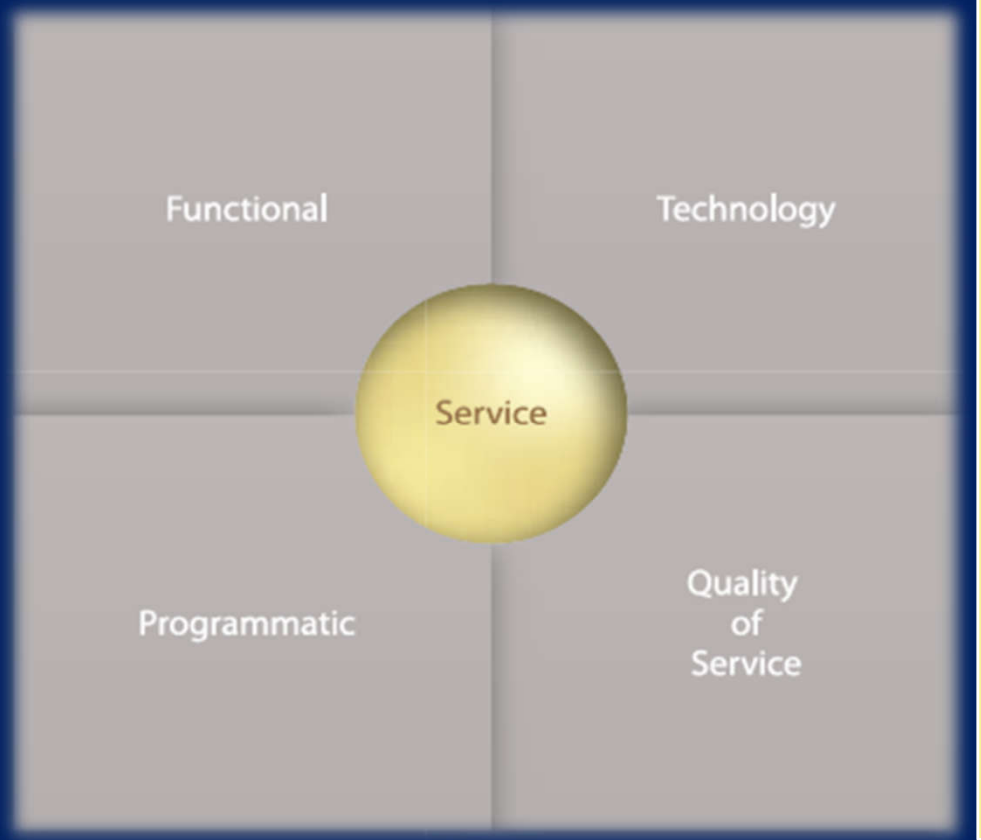
Service Abstraction

- به هر میزان اطلاعات ارائه شده به مصرف کنندگان بیشتر باشد، میزان آگاهی مصرف‌کننده به منطق سرویس، پلت فرم و جزئیات آن افزایش می‌یابد. بنابراین ممکن است مصرف کننده بر مبنای این اطلاعات پیش فرض‌هایی در تولید سرویس خود در نظر بگیرد و یک وابستگی consumer-to-implementation ایجاد شود.
- آن‌چه در این اصل مورد توجه قرار می‌گیرد، رسیدن به سطح مناسبی از پنهان سازی اطلاعات و جلوگیری از دسترسی به جزئیات اضافی است.

Service Abstraction



Service Abstraction

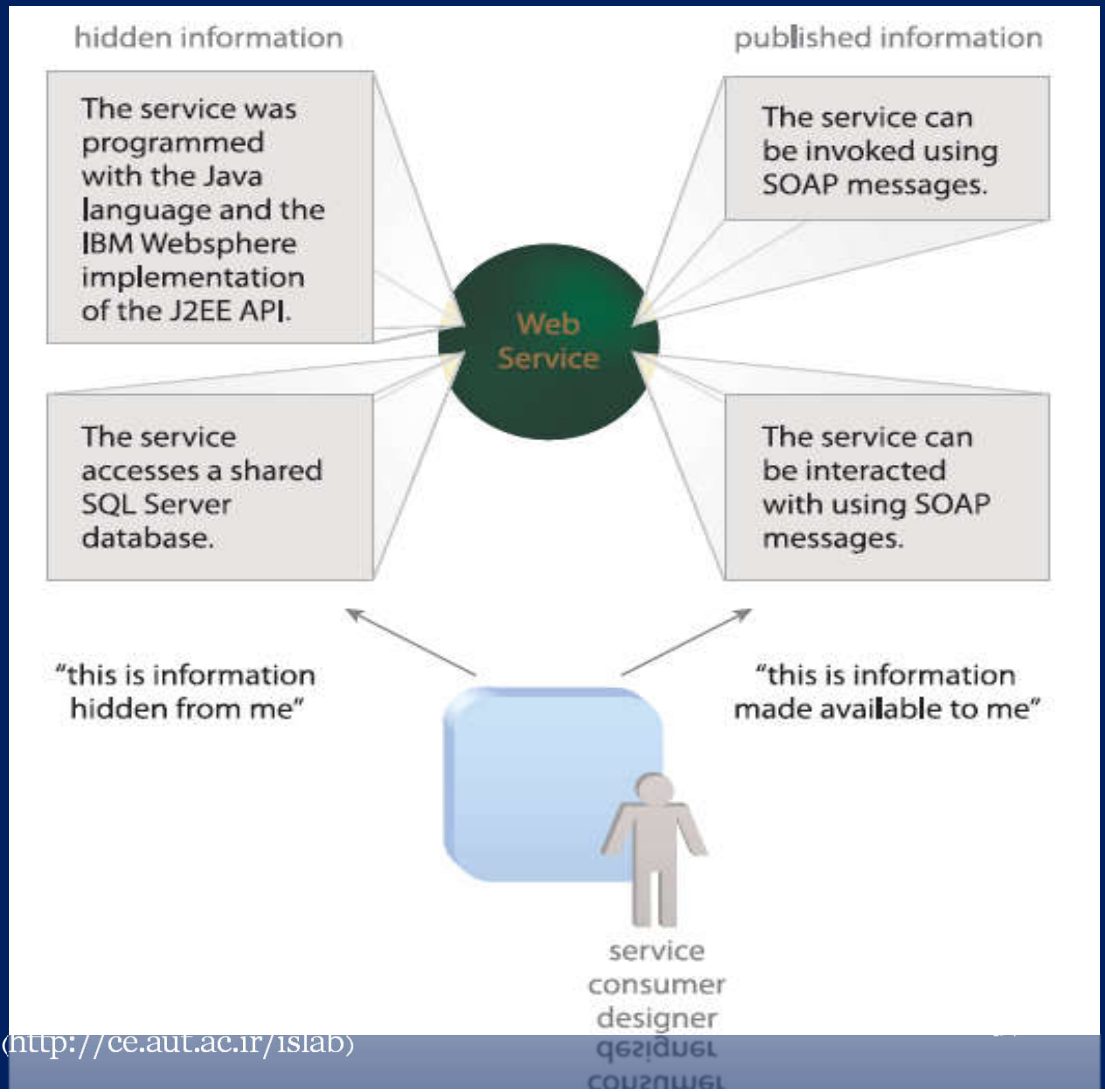


به طوری کلی اطلاعاتی که در مورد سرویس می‌توان فراهم کرد را می‌توان در دسته‌های مقابل طبقه بندی نمود. حال باید بررسی نمود که در رابطه با سرویس هر یک از این دسته اطلاعات را به چه میزان و به چه شیوه‌ای abstract نمود.

Service Abstraction

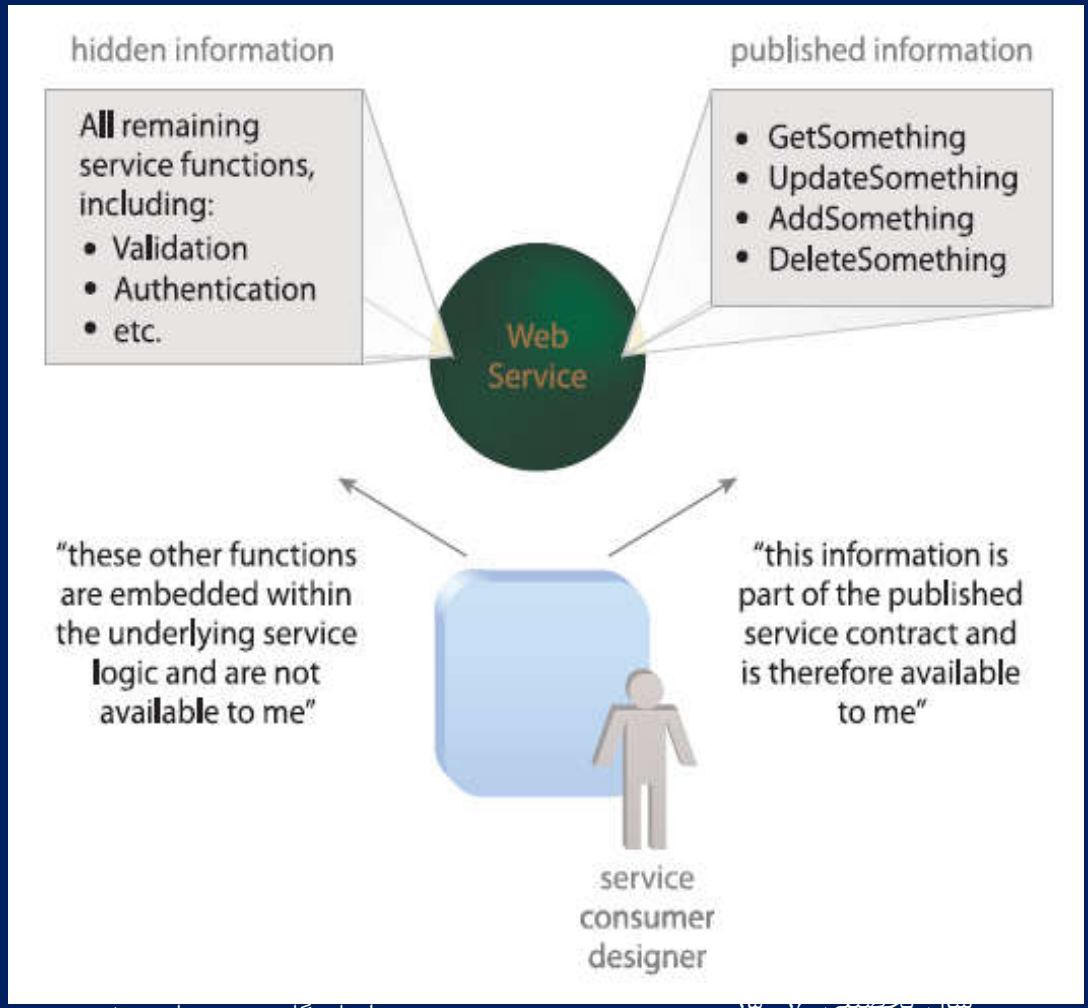
Technology Information Abstraction

- در رابطه با تکنولوژی باید اطلاعات زیر را منتشر نمود:
  - تکنولوژی مورد نیاز برای **فراخوانی سرویس**
  - تکنولوژی مورد نیاز برای **تعامل با سرویس**
- در رابطه با تکنولوژی باید اطلاعات زیر را مخفی نمود:
  - زبان برنامه نویسی مورد استفاده در سرویس
  - منابع مورد استفاده توسط سرویس



Service Abstraction
Functional Abstraction

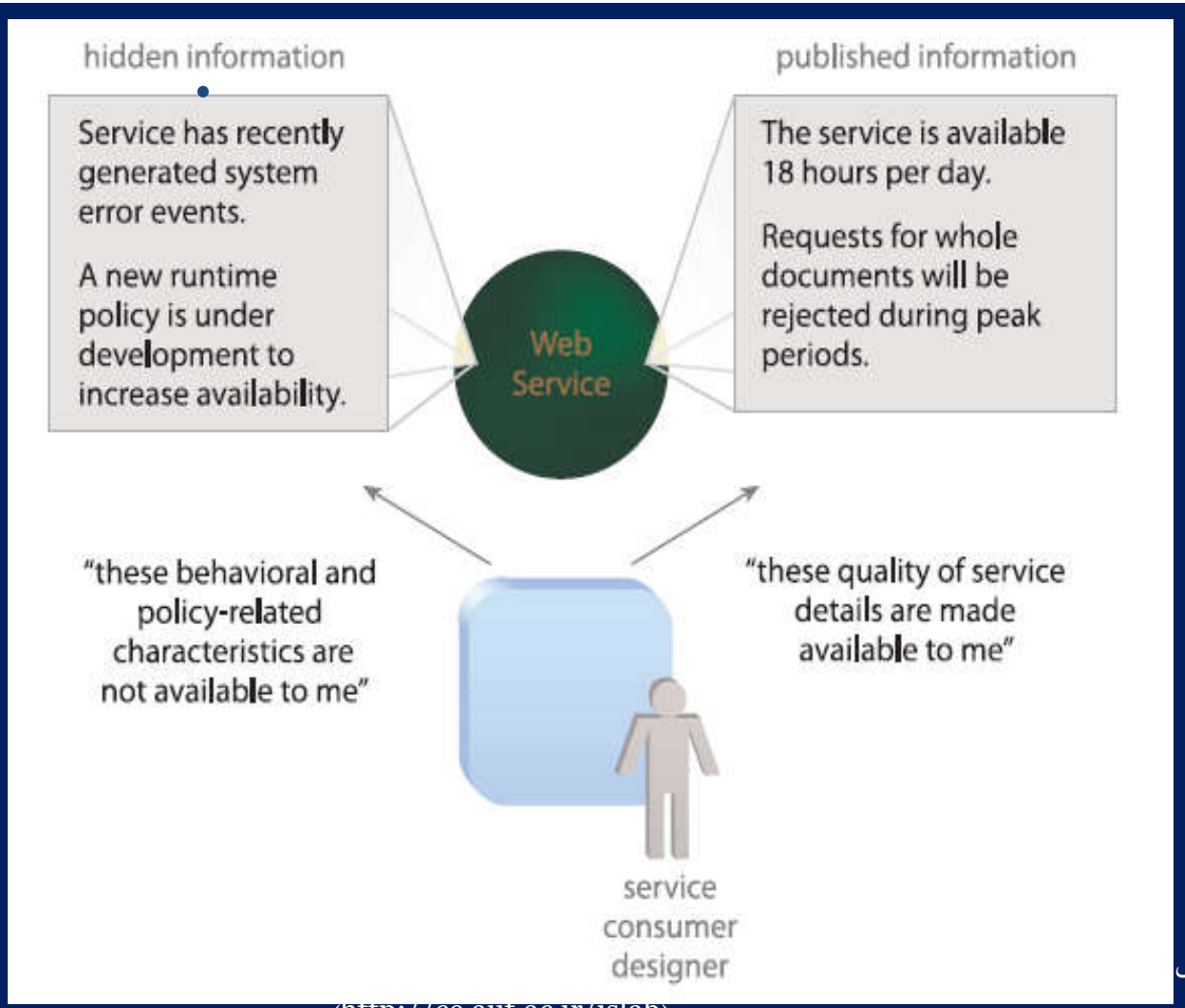
- این نوع انتزاع مشخص کننده‌ی این است که کدام یک از قابلیت‌های سرویس از طریق قرارداد تکنیکی ارائه گردد.
- تنها قابلیت‌هایی که به احتمال زیاد مورد استفاده برنامه‌های دیگر قرار می‌گیرند به صورت عمومی ارائه و مابقی پنهان می‌شوند.



Service Abstraction
Programmatic Abstraction

- این نوع از انتزاع، جزئیات سطح پایین طراحی مانند الگوریتم‌ها، کنترل خطا و استثناها و ... را پنهان می‌کند.
- بنابراین سرویس مصرف کننده به این اطلاعات دسترسی ندارد.

Service Abstraction
Quality of Service Abstraction

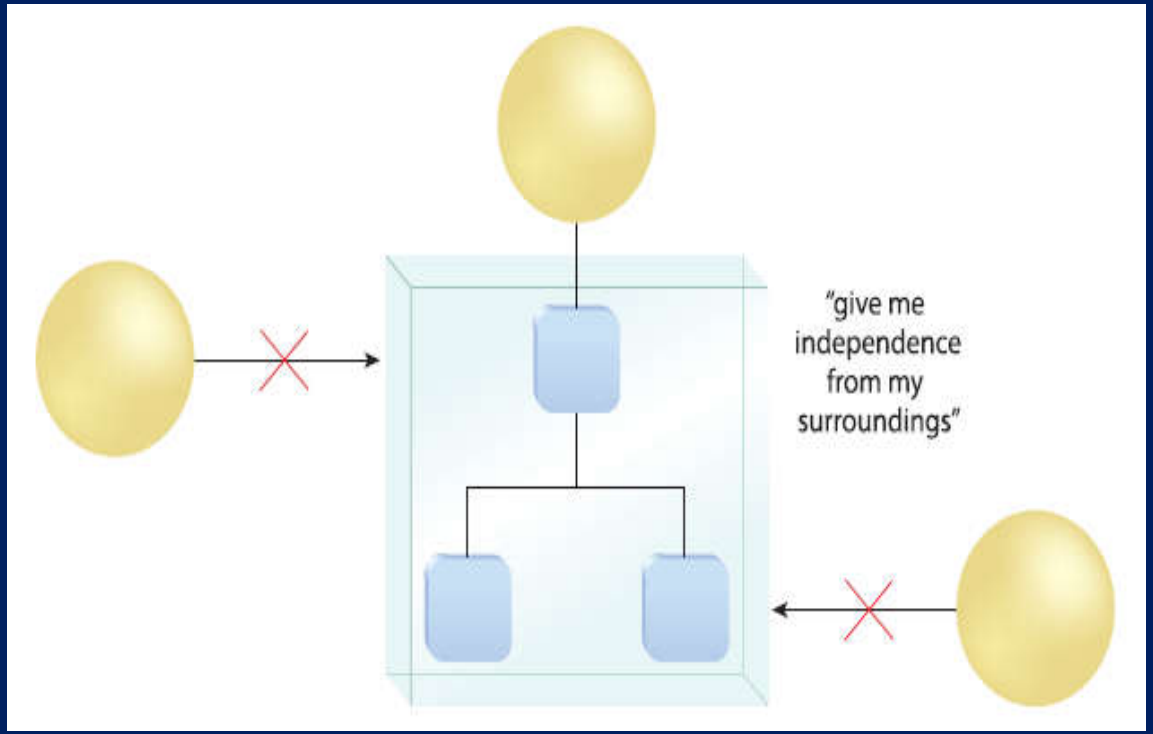


Service Autonomy

- خود مختاری سرویس نشان دهنده‌ی توانایی آن برای اجرای منطق پردازشی به صورت مستقل است.
- سرویس‌ها باید دارای یک درجه‌ای از کنترل بر محیط اجرا و منابع مورد مصرف خود داشته باشند. هر چه یک سرویس کنترل بیشتری بر روی محیط اجرای خود در زمان runtime داشته باشد، خودمختاری آن افزایش می‌یابد.
- هرچه یک سرویس از تاثیرات خارجی به میزان بیشتری مستقل و ایزوله باشد، قابلیت اطمینان آن بیشتر است. به عبارتی، از مهم‌ترین نتایج افزایش خودمختاری، افزایش قابلیت اطمینان (reliability) و پیش‌بینی پذیری (predictability) آن است

Service Autonomy

وقتی یک سرویس خودمختار باشد، منطق زیرین خود را به طور مستقل پیاده‌سازی می‌کند و تحت تاثیر موجودیت‌ها خارجی قرار نمی‌گیرد.



Service Autonomy

- به طور کلی برای سرویس دو نوع خودمختاری در نظر گرفته می‌شود:
  - خودمختاری در زمان اجرا (Runtime Autonomy): سطحی از کنترل که سرویس در زمان فراخوانی و اجرا بر روی منطق پردازشی خود دارد، خودمختاری در زمان اجرا خوانده می‌شود.
  - خودمختاری در زمان طراحی (Design-Time Autonomy): صرف نظر از اینکه سرویس بر محیط اجرای خود در زمان اجرا دارای کنترل است یا نه، استفاده کنندگان از این سرویس در رابطه با آن دارای وابستگی زمان طراحی هستند. این نوع وابستگی برای تکامل سرویس در پاسخ گویی نسبت به تغییرات آینده در نیازمندی‌ها محدودیت ایجاد می‌کند. به عبارتی زمانی که یک سرویس (بر مبنای قرارداد سرویس) دارای چند مصرف کننده است، کنترل خود را نسبت تغییرات و تکامل از دست می‌دهد. این نوع از خودمختاری با در نظر گرفتن اصل اتصالات سست فراهم می‌گردد.
- این دو نوع با یکدیگر دارای ارتباط مستقیم هستند به طوری که افزایش یکی، افزایش دیگری را نتیجه می‌دهد.

Service Autonomy
Autonomy level

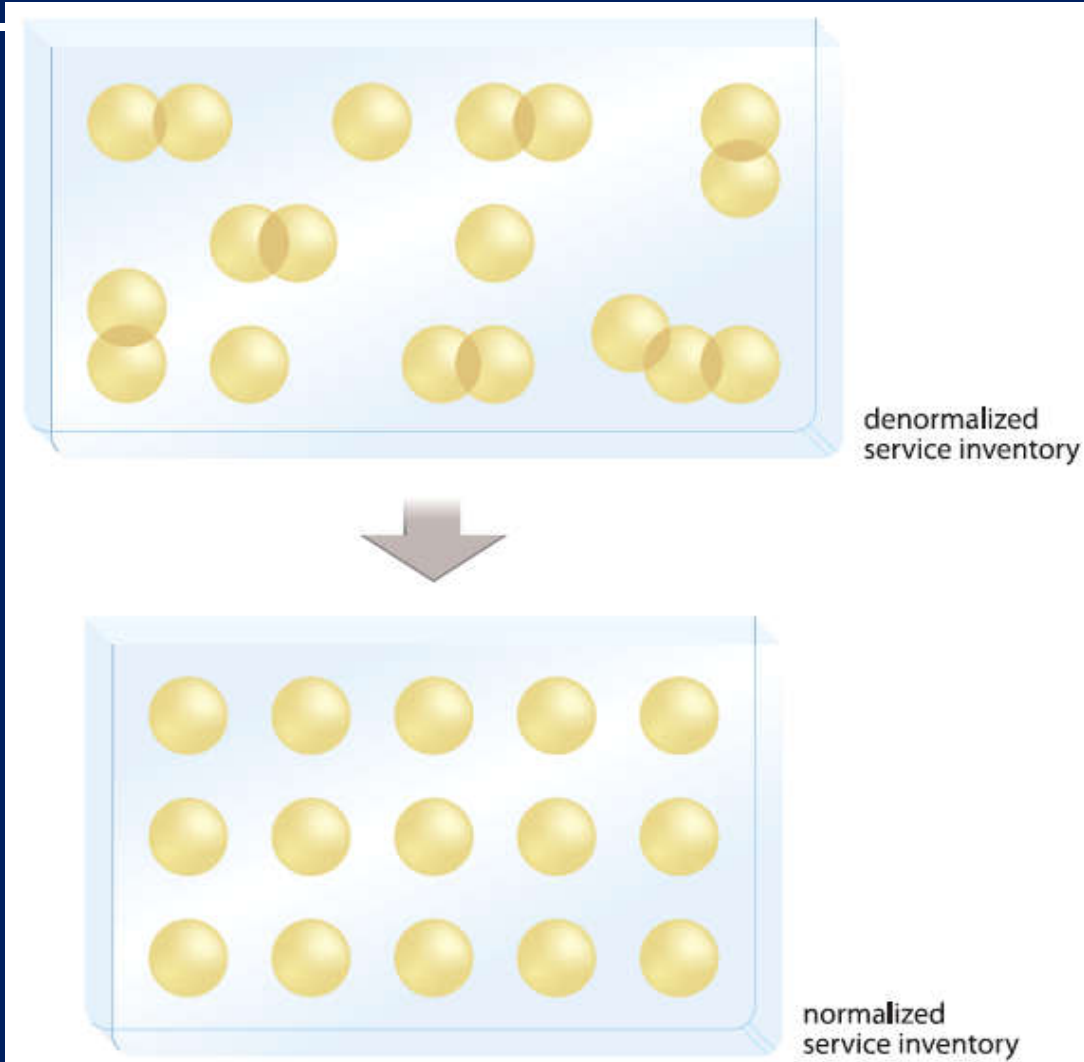
- سطوح خودمختاری را می‌توان به صورت زیر دسته بندی نمود:
  - Service Contract Autonomy
  - Shared Autonomy
  - Service Logic Autonomy
  - Pure Autonomy

Service Contract Autonomy

- یکی از مواردی که در تهیه یک مجموعه از سرویس‌ها دارای اهمیت است، اطمینان از این است که محدوده‌ی کارکردهایی که یک سرویس ارائه می‌کند، تنها در اختیار خود آن است. به عبارتی قابلیت هایی که در قرارداد یک سرویس بیان شده است نباید با سایر سرویس ها هم پوشانی داشته باشد.
- البته این نوع از خودمختاری بر خودمختاری زمان اجرا دلالت نمی کند، به عبارتی حتی اگر کارکردهای ارائه شده با هم هم پوشانی نداشته باشند، ممکن است پیاده سازی آن ها با یکدیگر هم پوشانی داشته باشد.

Service Contract Autonomy

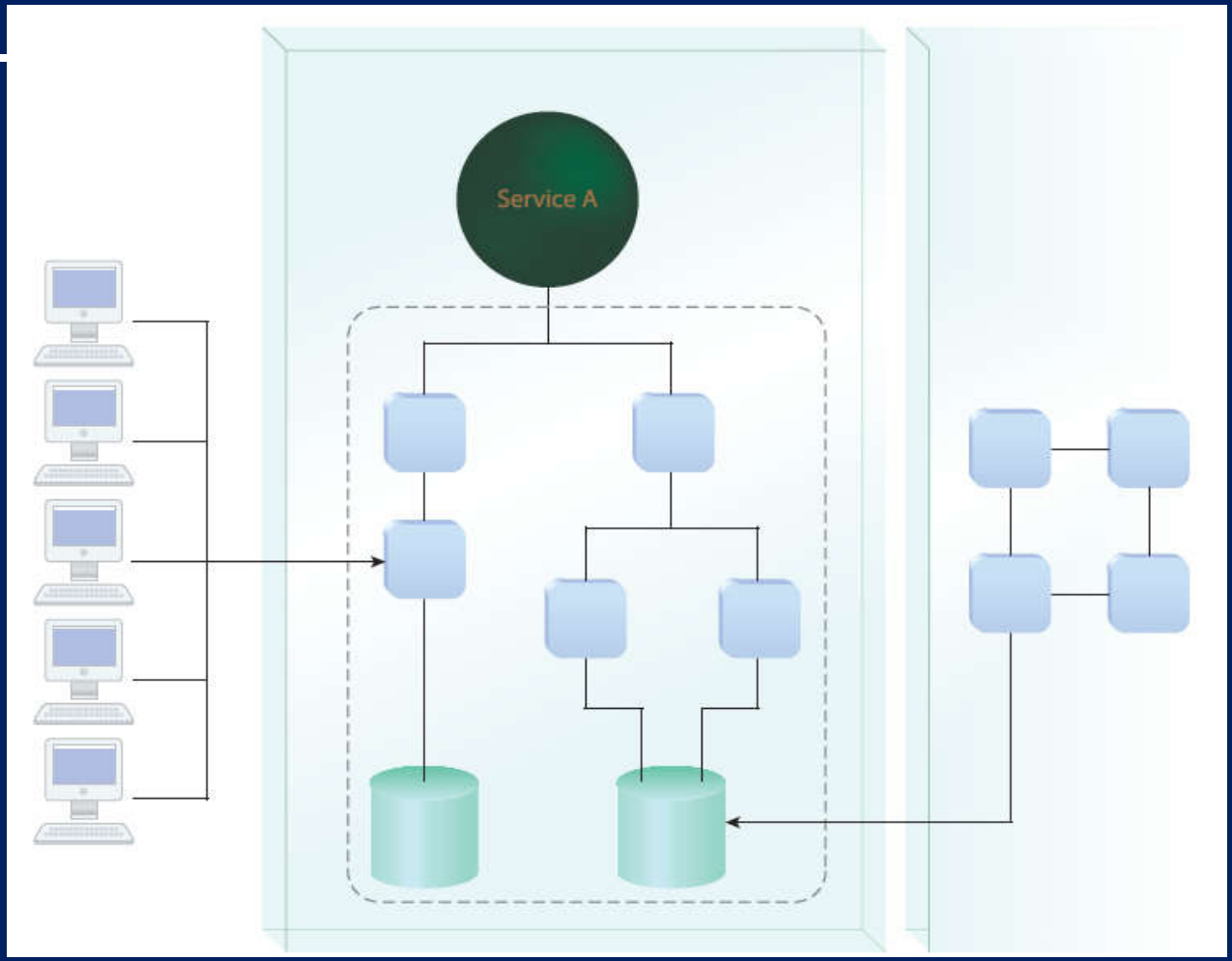
- در حقیقت این نوع از خودمختاری با مفهوم **نرمال‌سازی** در ارتباط می‌باشد.
- مجموعه سرویس‌ها باید نرمال باشد و در آن افزونگی وجود نداشته باشد.



(http://ce.aut.ac.ir/islab)

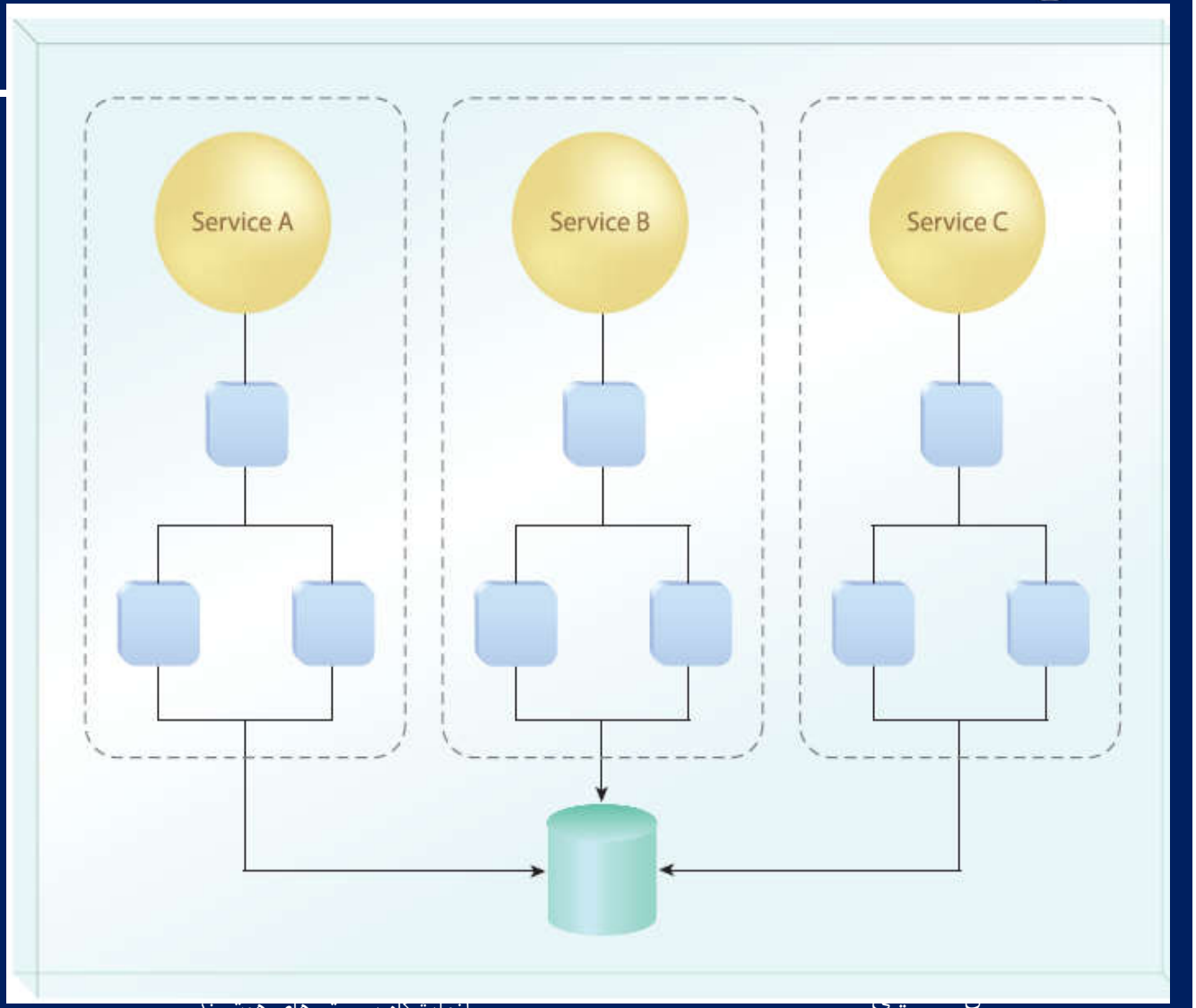
Shared Autonomy

- تمامی سرویس‌ها از ابتدا ساخته نمی‌شوند، بلکه ممکن است بعضی از آن‌ها برنامه‌های کاربردی legacy را کپسوله نمایند. بنابراین در این حالت سرویس از خودمختاری بالایی برخوردار نیست.



Service Logic Autonomy

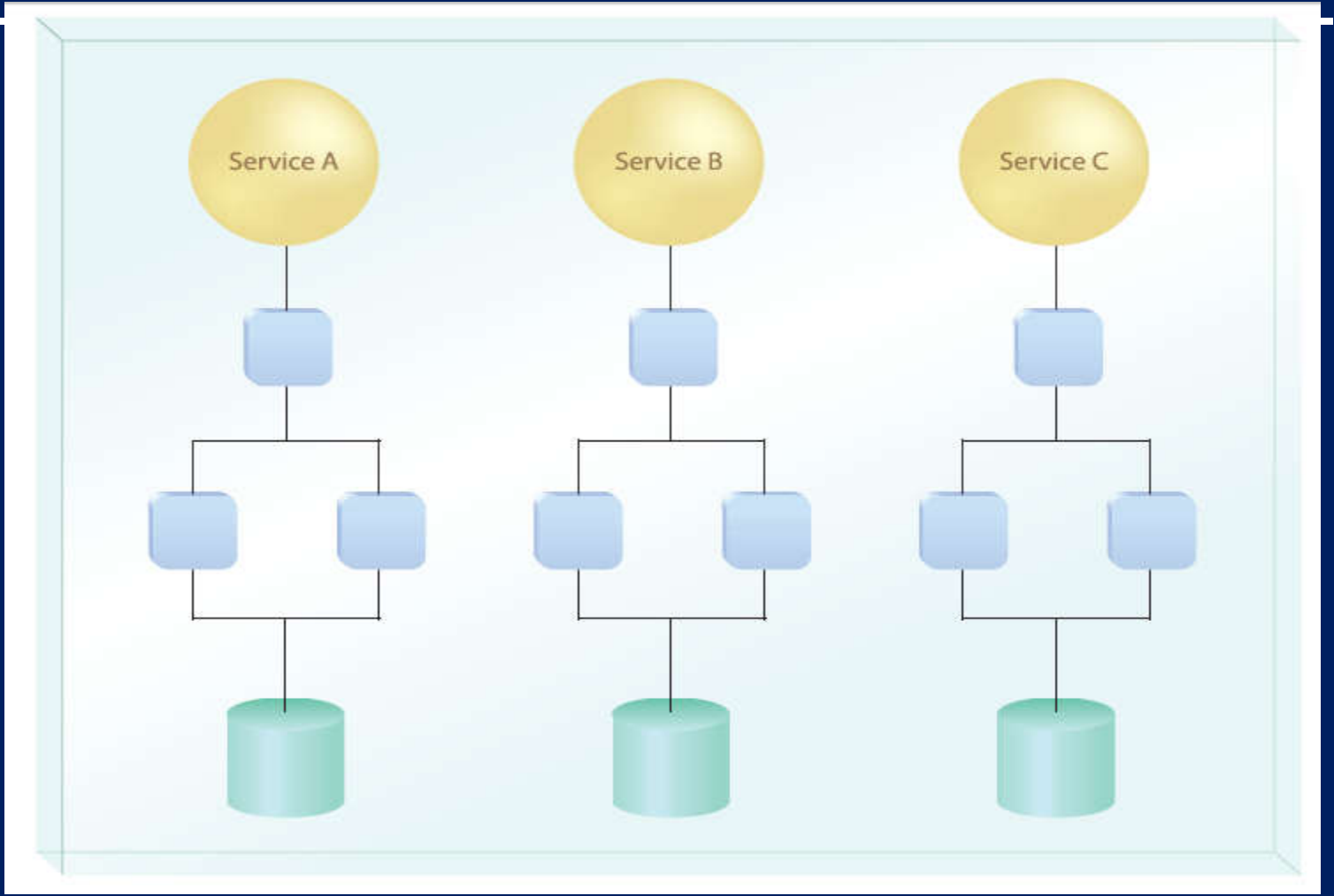
در این حالت همانطور که مشاهده می‌شود، مولفه‌های هر سرویس نسبت به سایر سرویس‌ها ایزوله هستند. اما در این حالت با توجه به این که داده‌ها (پایگاه داده، فایل و ...) و سایر منابع بین سرویس‌ها و یا اجزای دیگر محیط مشترک هستند، سرویس از خودمختاری جزئی برخوردار است.



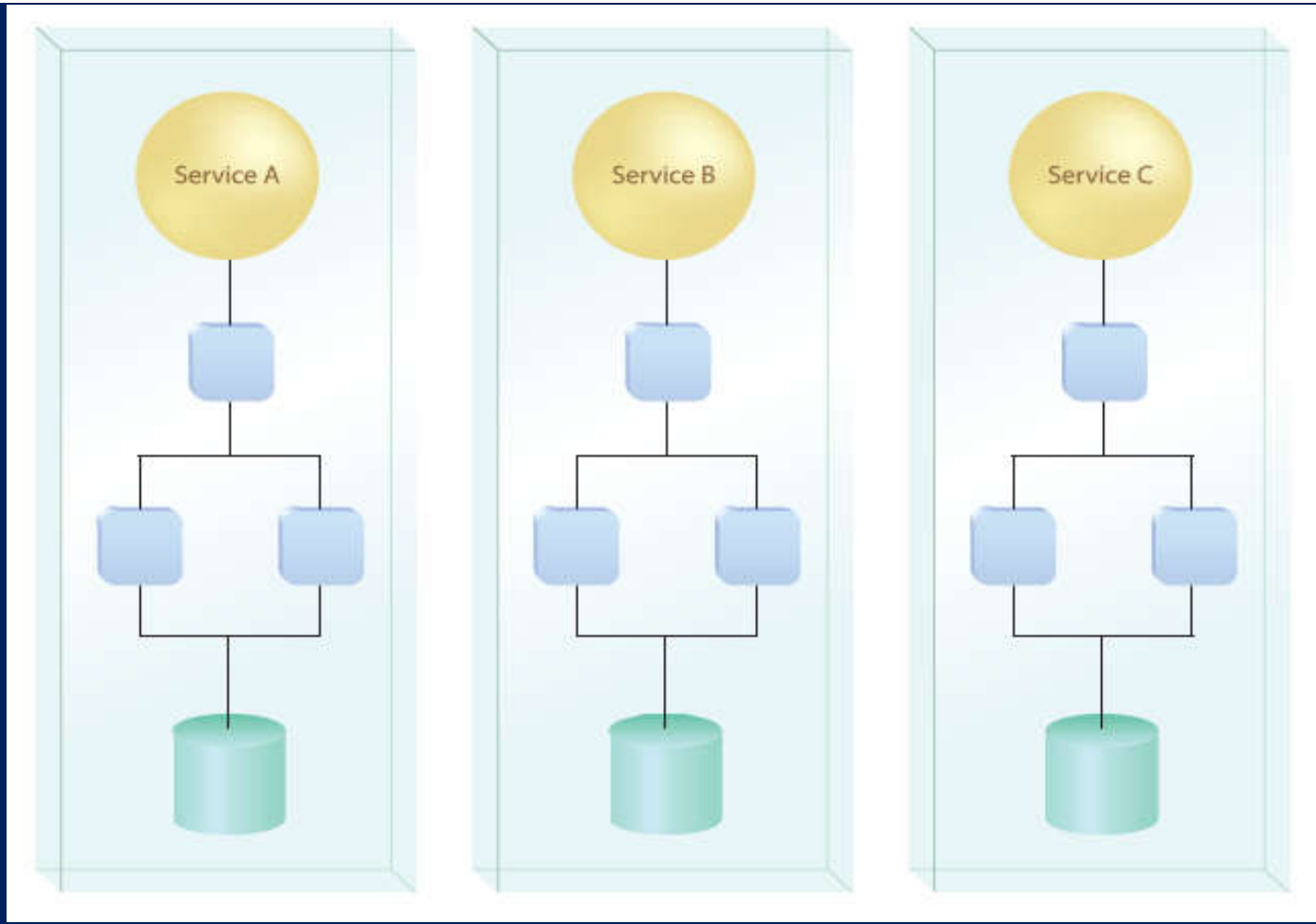
Pure Autonomy

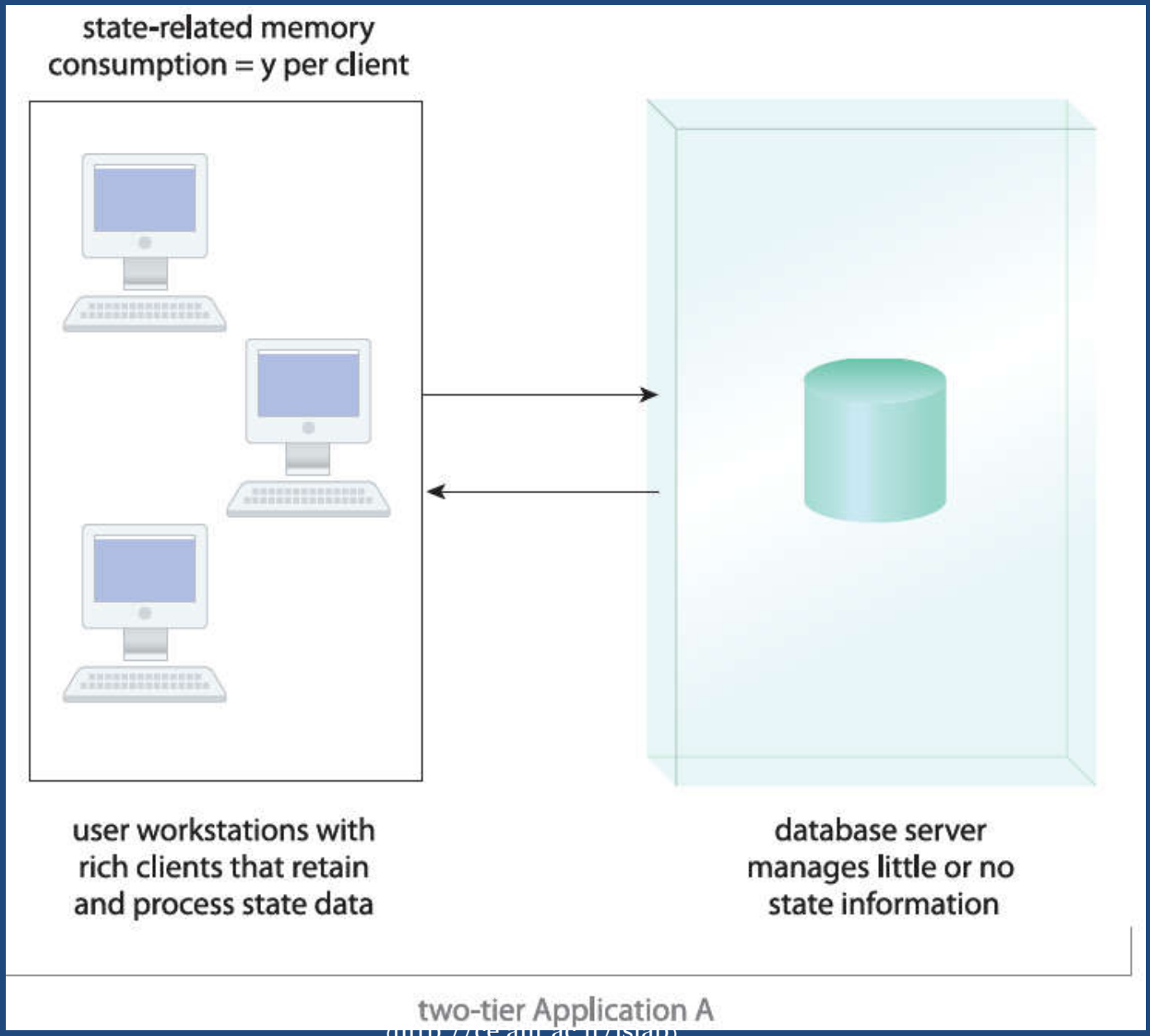
- این نوع از خودمختاری، یک محیط پیاده‌سازی ایده‌آل برای سرویس فراهم می‌آورد، که در آن سرویس دارای کنترل کامل بر محیط اجرا و منابع موردنیاز خود است.
- این نوع از خود مختاری به دو صورت است:
  - Functional Isolation: مولفه‌های تشکیل دهنده سرویس و داده ها برای هر سرویس اختصاصی هستند اما تمام سرویس ها در یک سرور قرار دارند. (منابع سیستمی یکسانی را به اشتراک میگذارند)
  - Absolute Isolation: مولفه‌های تشکیل دهنده سرویس، داده ها و محیط سرویس برای هر سرویس اختصاصی هستند.

Functional Isolation

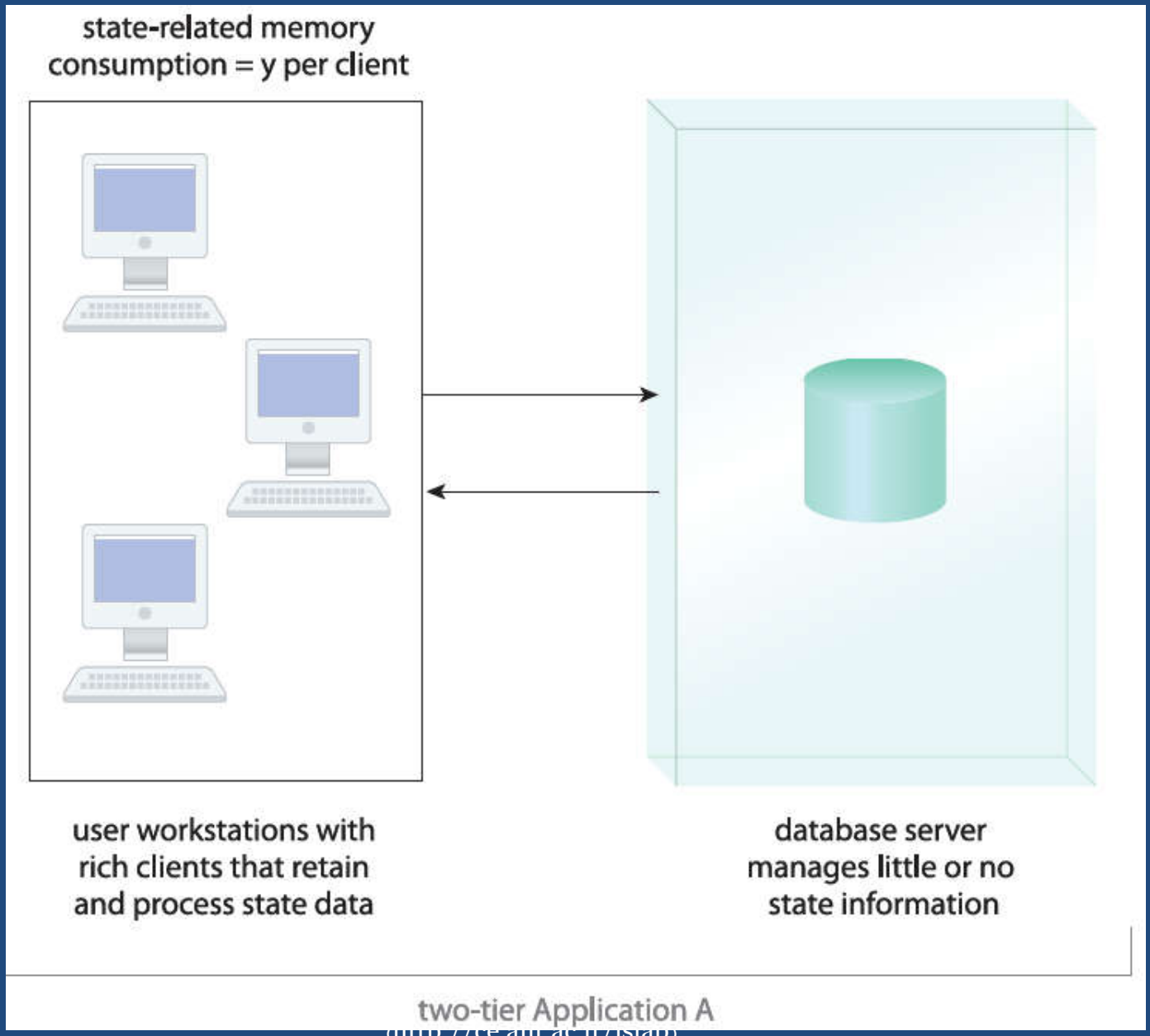


Absolute Isolation

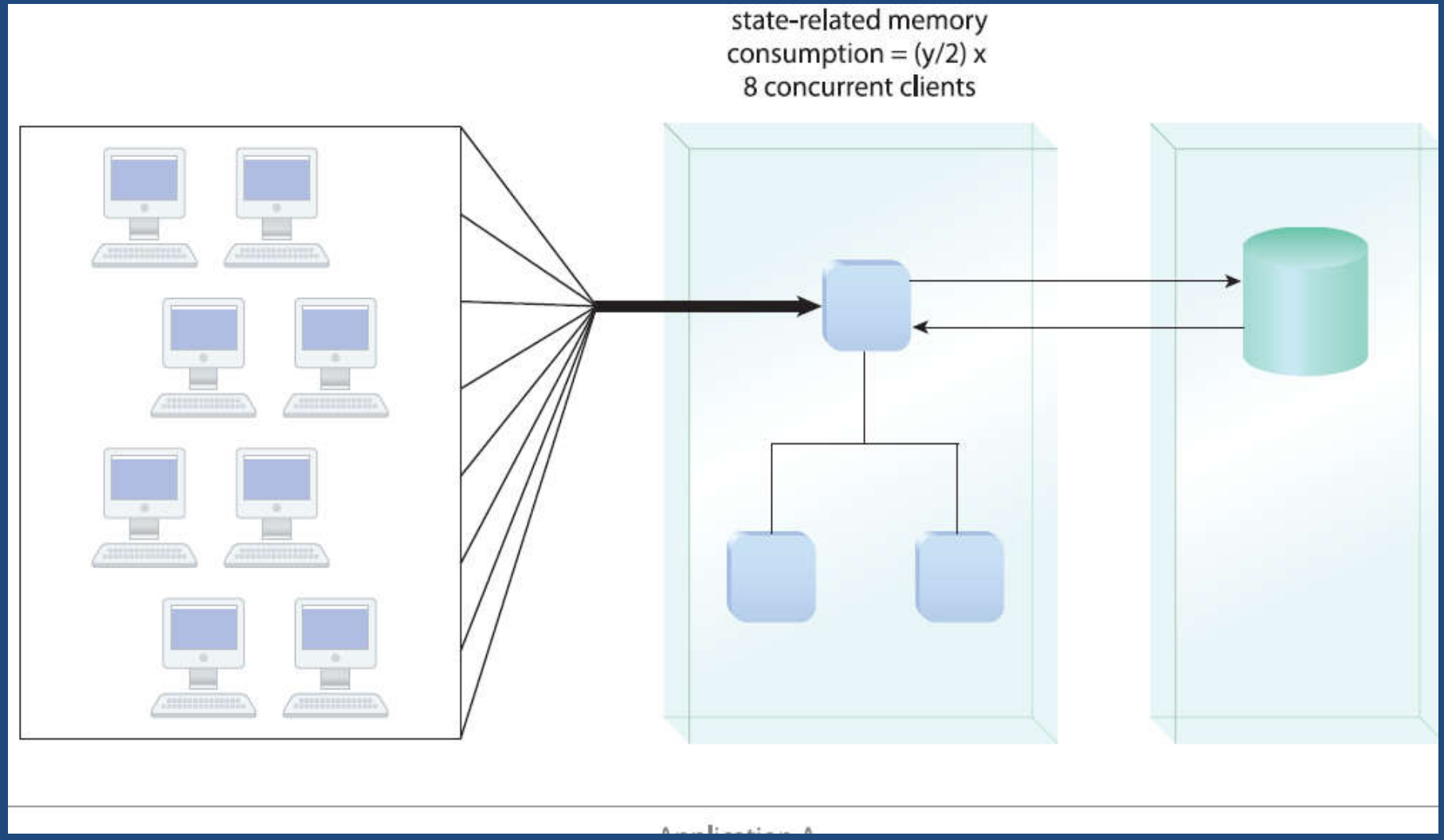




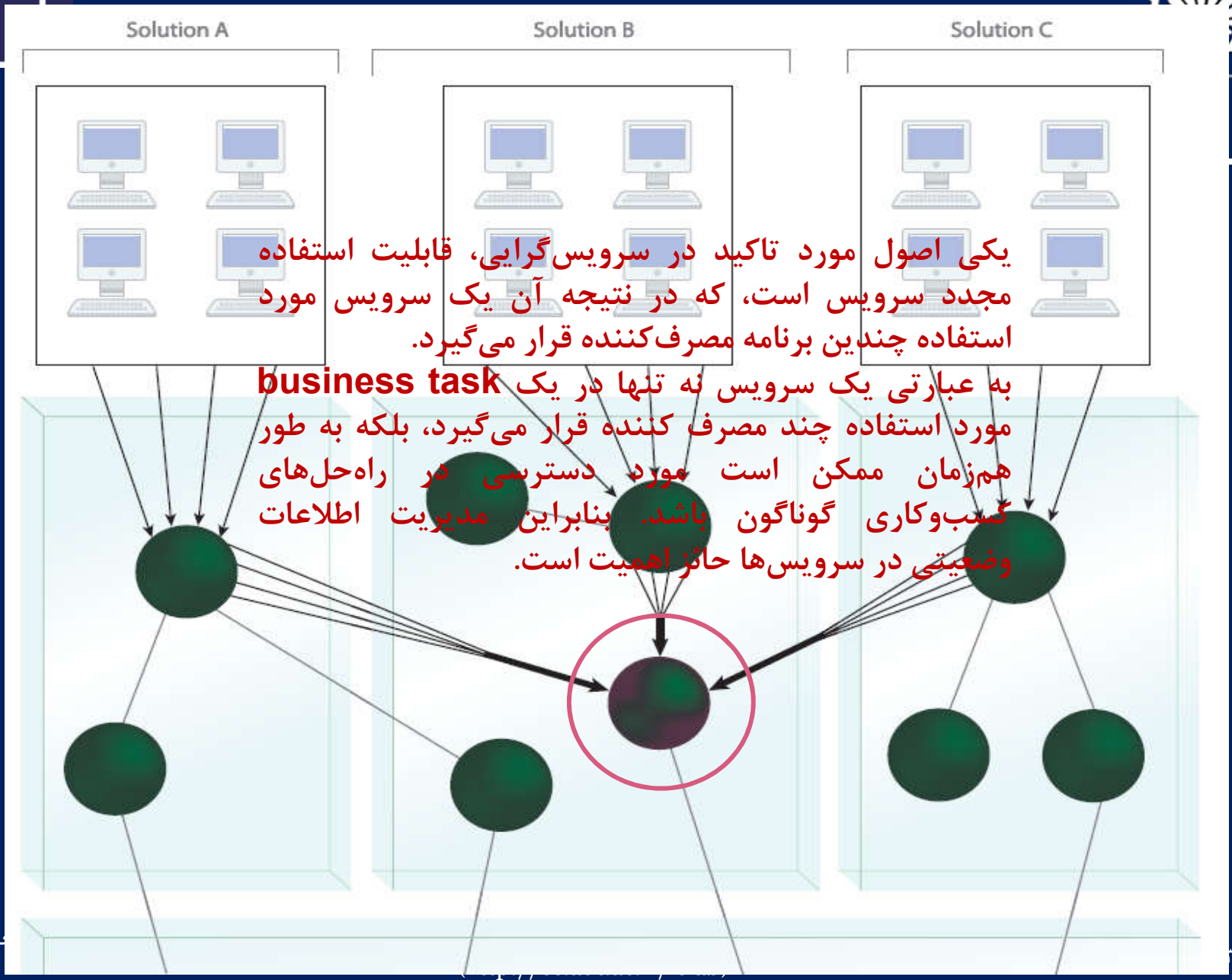
two-tier Application A



two-tier Application A


state-related memory
consumption = (y/2) x
8 concurrent clients

زمانی که حجم زیادی از اطلاعات وضعیتی پردازش و یا نگه داری می شود، میزان استفاده از حافظه و CPU در سمت سرور بالا می رود.
با توجه به این که سناریوهای استفاده در زمان اجرا همیشه قابل پیش بینی نیستند، دسترسی همزمان به برنامه های سمت سرور، موجب ایجاد گلوگاه کارایی در این نقطه می شود.
البته تنها نگهداری اطلاعات وضعیتی موجب مصرف حافظه سیستم اما به طور ملاحظه ای در این امر نقش دارد. یک راه حل برای حل این مساله، واگذاری وضعیت یا state delegation است.
به این ترتیب می توان از یک پایگاه داده‌ی اختصاصی برای نگه داری و بازیابی اطلاعات وضعیتی استفاده نمود.



یکی اصول مورد تاکید در سرویس‌گرایی، قابلیت استفاده مجدد سرویس است، که در نتیجه آن یک سرویس مورد استفاده چندین برنامه مصرف‌کننده قرار می‌گیرد.

به عبارتی یک سرویس نه تنها در یک **business task** مورد استفاده چند مصرف کننده قرار می‌گیرد، بلکه به طور هم‌زمان ممکن است مورد دسترسی در راه‌حل‌های کسب‌وکاری گوناگون باشد. بنابراین مدیریت اطلاعات وضعیتی در سرویس‌ها حائز اهمیت است.

Service Statelessness

- تولید سرویس‌هایی که اطلاعات وضعیتی را مدیریت نمی‌کنند مستلزم در دسترس بودن منابعی در محیط سرویس است تا بتوان وظایف مدیریت وضعیت سرویس را به آن‌ها واگذار نماید. به عبارتی سرویس مدیریت اطلاعات وضعیتی را به تاخیر می‌اندازد. (این واگذاری به صورت موقت است.)
- به این ترتیب مقیاس‌پذیری سرویس افزایش می‌یابد.
- همچنین از طراحی سرویس‌هایی با منطق agnostic پشتیبانی شده و امکان استفاده مجدد بیشتر می‌گردد.

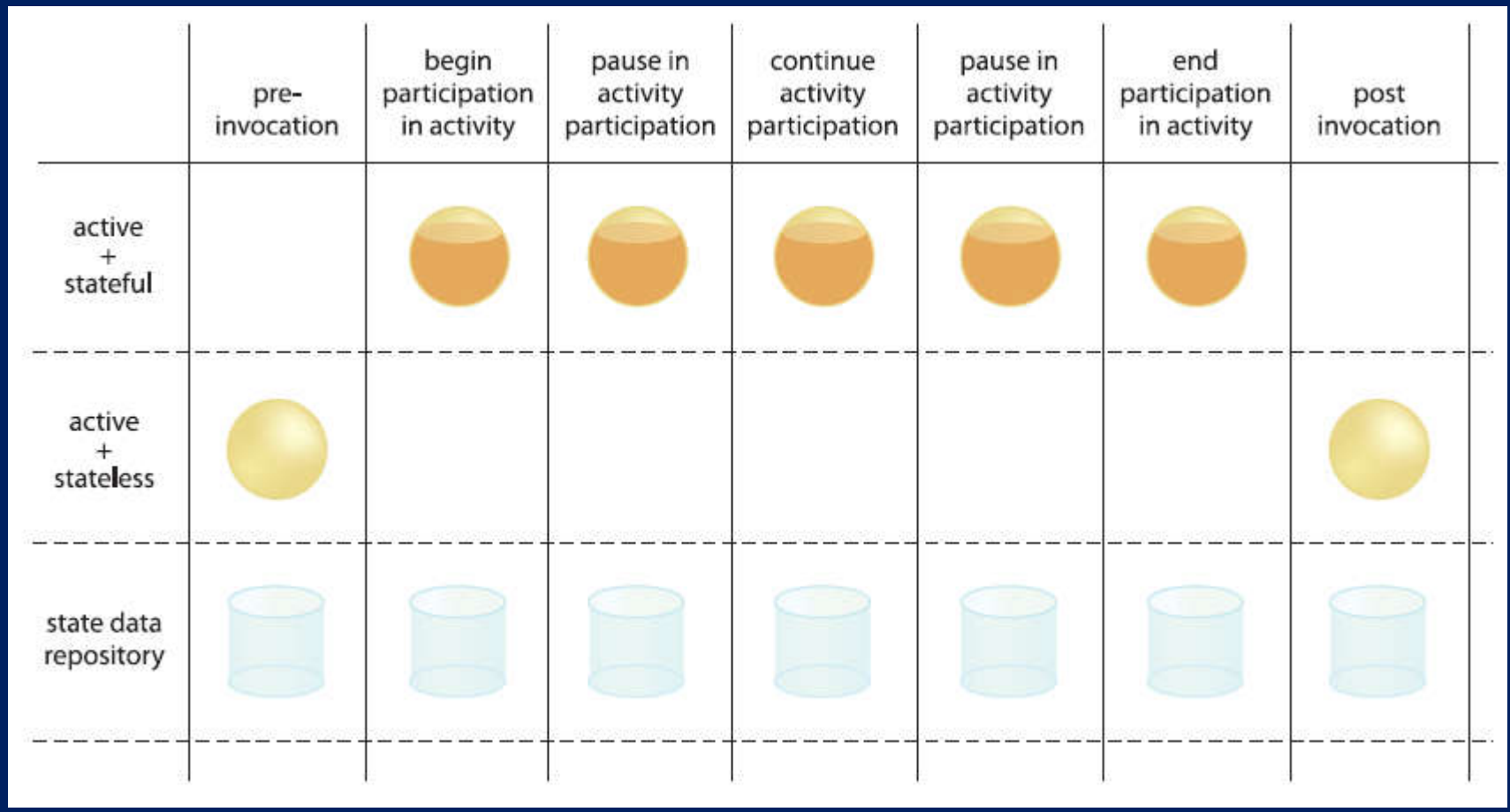
Service Statelessness

- یک سرویس زمانی‌که مورد استفاده قرار نمی‌گیرد در وضعیت passive و زمانی‌که فراخوانی و اجرا می‌شود در وضعیت active قرار دارد.
- زمانی‌که یک سرویس در وضعیت active ممکن است در یکی از دو وضعیت زیر به سر ببرد:
  - Stateful: زمانی یک سرویس در حال پردازش یا نگه داری اطلاعات وضعیتی یک task به خصوص می‌باشد، در وضعیت stateful است.
  - Stateless: زمانی یک سرویس در وضعیت فعال است اما هیچ داده‌ای را پردازش و نگه‌داری نمی‌کند.

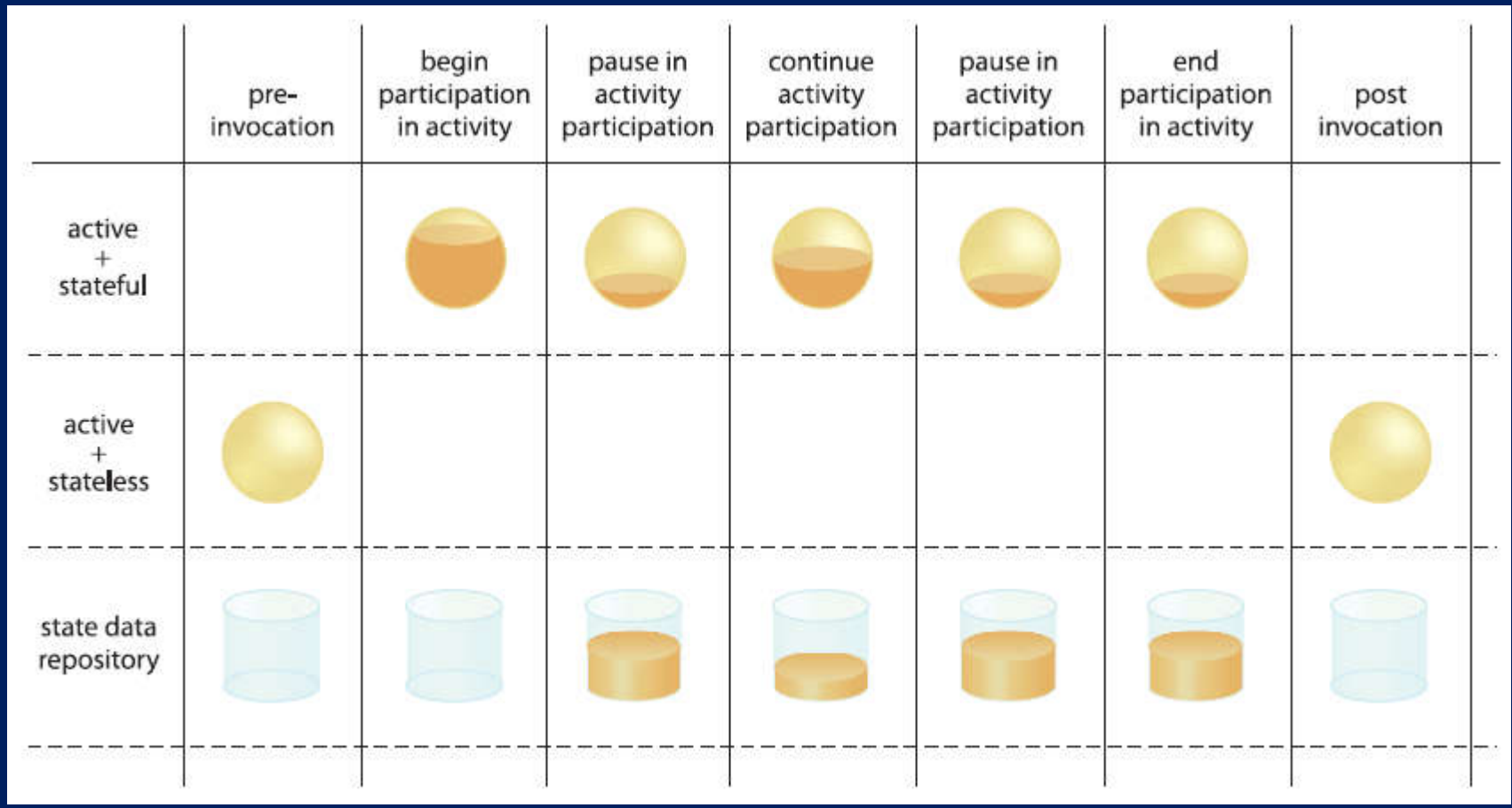
Service Statelessness

- سرویس‌ها باید مقدار اطلاعات وضعیتی که توسط آن‌ها مدیریت می‌شود و همچنین بازه‌ی نگه‌داری این اطلاعات را حداقل نمایند.
- وقتی یک سرویس در حال پردازش یک پیغام است، به صورت موقت state full است.
- اگر یک سرویس مسئول نگه داری اطلاعات وضعیتی برای بازه‌ی طولانی باشد، قابلیت دسترسی آن تحت تاثیر قرار می‌گیرد و توان مقیاس‌پذیری آن کاهش می‌یابد. بنابراین سرویس‌ها طوری طراحی می‌شوند که تنها در صورت نیاز state full باشند.
- برای اینکه یک سرویس اطلاعات وضعیتی اندکی را نگه داری کند، operation های آن باید با در نظر گرفتن پردازش‌های stateless طراحی شوند.

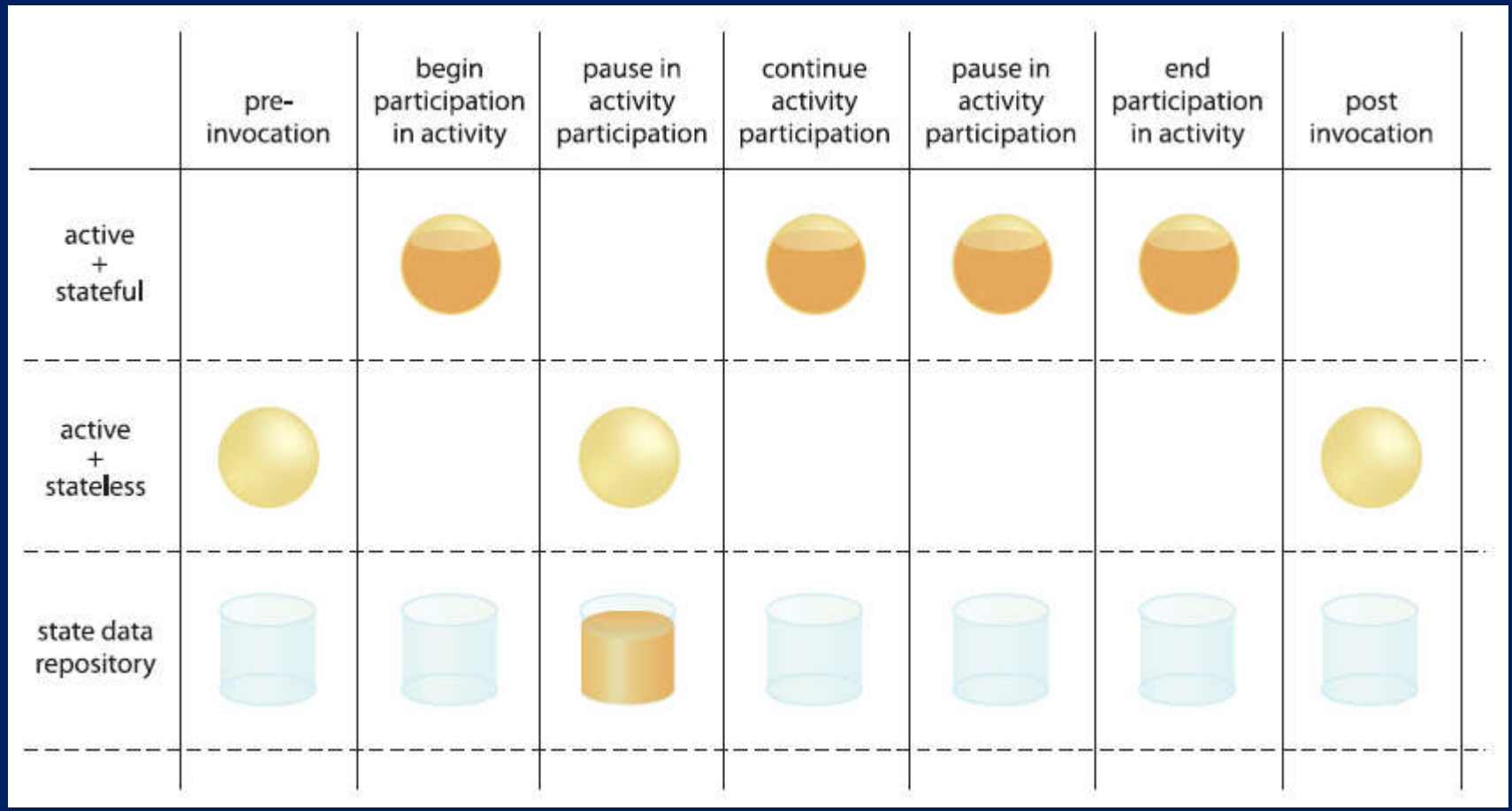
Non-Deferred State Management (low-to-no statelessness)



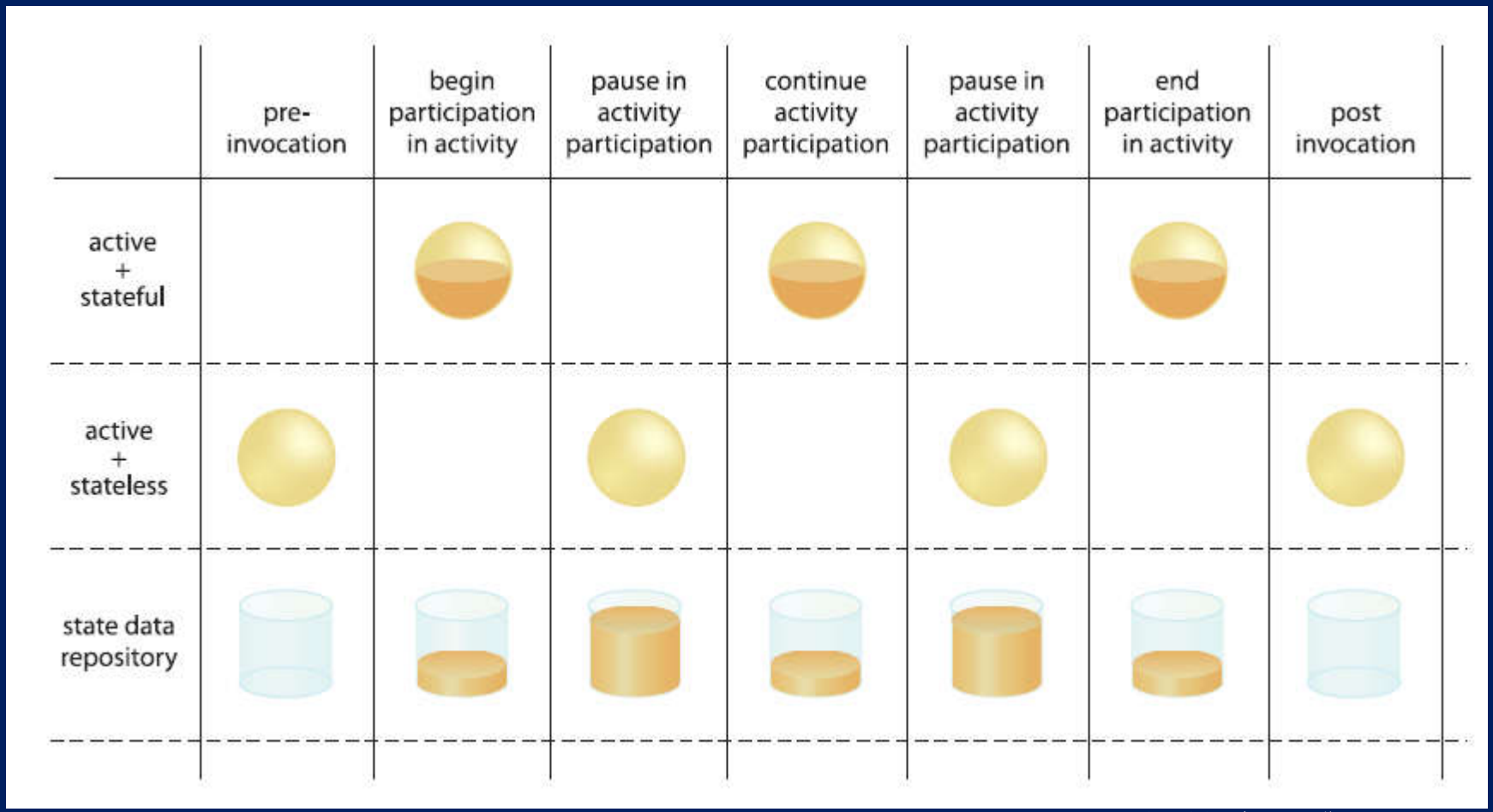
Partially Deferred Memory (reduced statefulness)



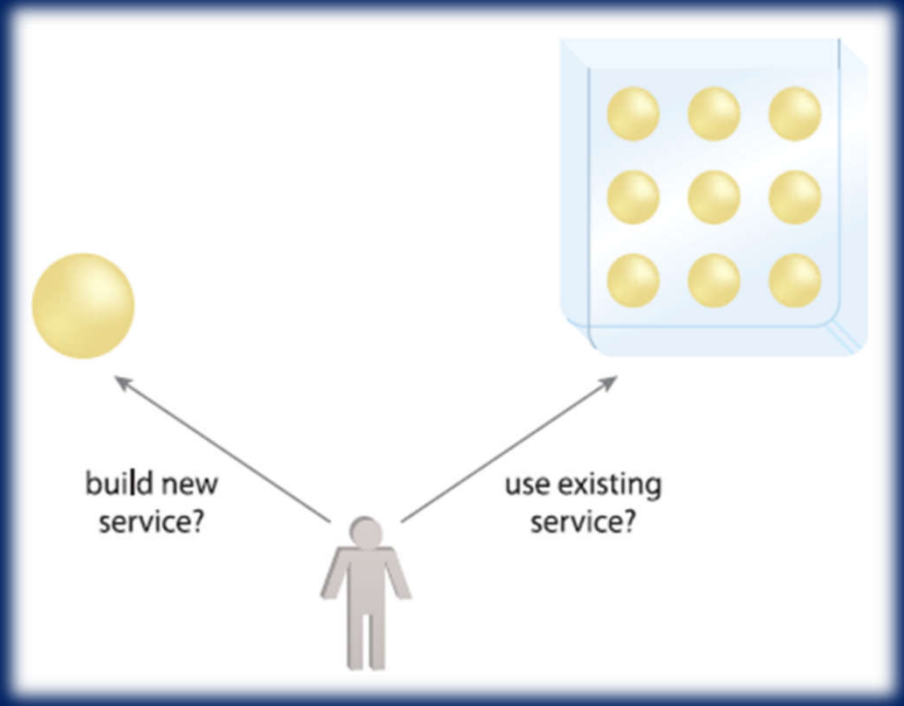
Partial Architectural State Management Deferral (moderate statelessness)



Internally Deferred State Management (high statelessness)



Service Discoverability

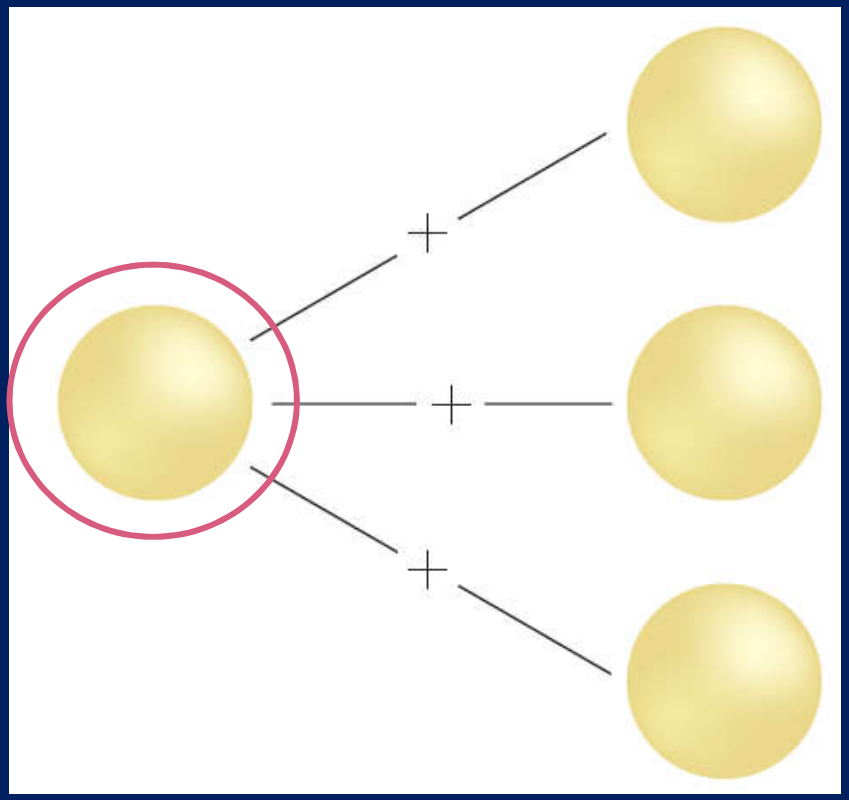


- با کمک قابلیت کشف سرویس، می‌توان فهمید که آیا کارکرد مورد نظر ما در یک مجموعه از سرویس‌ها موجود است یا باید آن را تولید کرد.
- از اینرو کشف سرویس به منظور اجتناب از تولید سرویس‌های افزونه و زاید دارای اهمیت است.

Service Discoverability

- کشف سرویس از طریق metadata منتشر شده در مورد سرویس امکان پذیر است. این فراداده‌ها عبارتند از:
  - هدف یک سرویس
  - قابلیت‌های یک سرویس و محدودیت های آن
- برای ارزیابی سرویس‌ها، این فراداده‌ها بر مبنای معیارهای انتخاب مورد بررسی قرار می‌گیرند.
  - کیفیت این فراداده‌ها، نحوه‌ی دسترسی به آن‌ها و ... همه در تصمیم‌گیری درباره‌ی مناسب بودن یک سرویس تاثیرگذارند.
- کشف سرویس معمولا به صورت manual و human based انجام می پذیرد.
- روش هایی اتوماتیک برای کشف سرویس نیز وجود دارد (برای مثال UDDI) و ارائه روش هایی برای این منظور جز حوزه های تحقیقاتی فعال است.
- لازم به ذکر است، کشف سرویس با کیفیت مشخص یک مساله NP-hard است.

Service Composability



- همان‌طور که بیان شد، قابلیت استفاده مجدد از سرویس‌ها یکی از اصول اصلی و مورد تاکید در سرویس‌گرایی است. اما تحقق اهداف این اصل بسیار وابسته به ترکیب موثر و مکرر سرویس‌ها در کنار سرویس‌های دیگر است. بنابراین ترکیب سرویس‌ها (service composition) دارای اهمیت زیادی است.

Service Composability