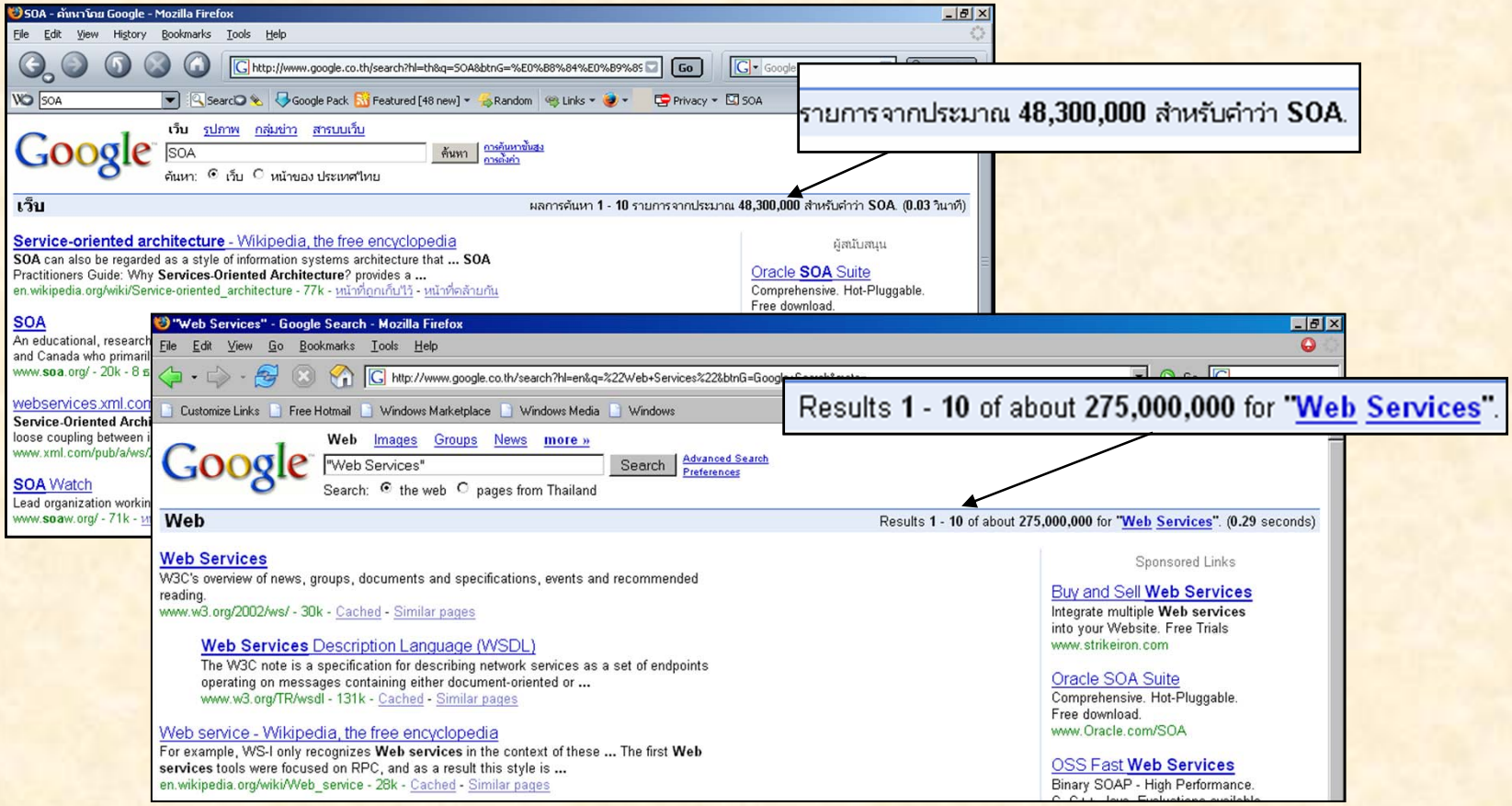
Session 1 : Intro duction to SOA

อาจารย์เชษฐ พัฒโนทัย
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
จุฬาลงกรณ์มหาวิทยาลัย

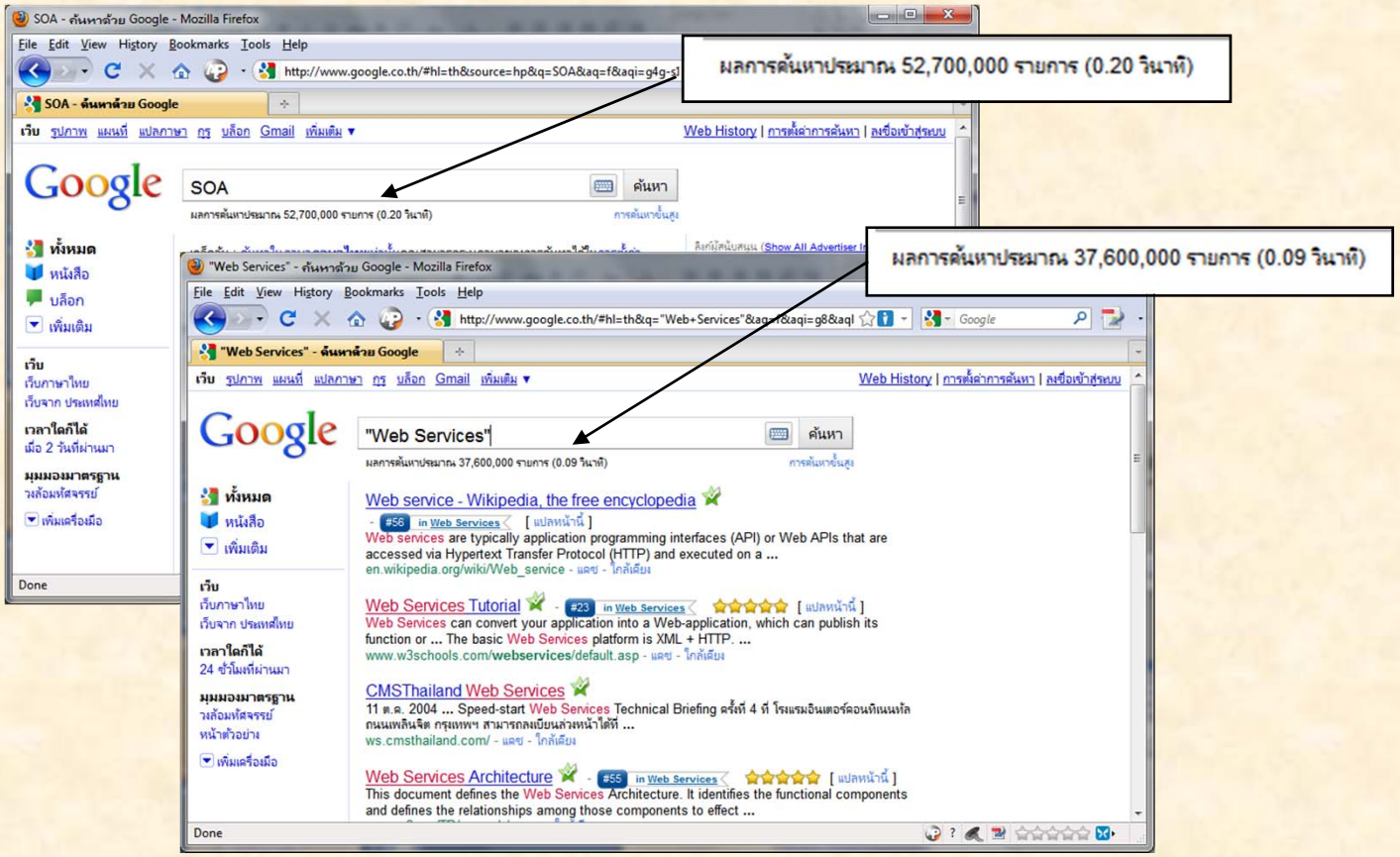
หัวข้อนำเสนอ

- คำถามเกี่ยวกับ SOA
- วิวัฒนาการสู่ SOA
- หลักการของ SOA
- หลักการของ Service
- หลักการของ Connection Technology
- ตอบคำถามเกี่ยวกับ SOA
- บทสรุป

ลองค้นคำว่า “SOA” (2550)



ค้นเมื่อ 9 ส.ค. 2553



คำถามเกี่ยวกับ SOA

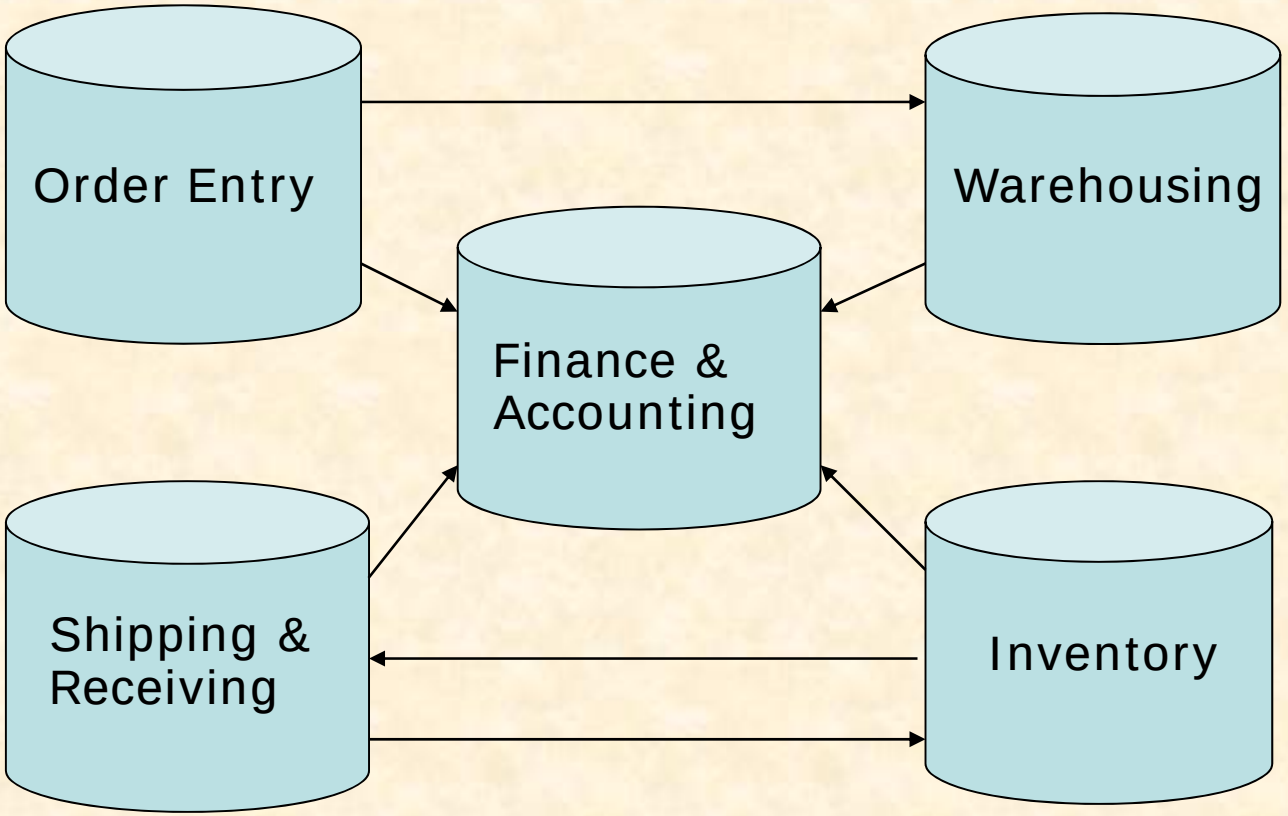
- SOA คืออะไร
- ทำไมต้องใช้ SOA
- ใช้ SOA เมื่อไร ไม่ใช้ได้หรือไม่
- พัฒนา SOA อย่างไร
- SOA กับ Web Services เหมือนกันหรือไม่
- จำเป็นต้องใช้ SOA ควบคู่กับ Web Services หรือไม่
- อื่น ๆ

วิวัฒนาการสู่ SOA

Data Silos (1)

- หน่วยงานในองค์กรต่างมี Application และฐานข้อมูลของตนเอง
- ทำงานร่วมกันไม่สะดวกเพราะโครงสร้างหรือเทคโนโลยีในระบบต่างกัน
  - ไม่มี API ให้เรียกข้ามระบบ
  - ไม่สามารถใช้ข้อมูลร่วมกันได้โดยตรง
- เกิด Data Silos หรือ Islands of Data
- แลกเปลี่ยนข้อมูลกันโดย
  - Manual Rekey ข้อมูลของระบบอื่น
  - ส่งไฟล์ผ่าน Magnetic Tape หรือ Floppy Disk
  - ส่งไฟล์แบบ Electronic Transfer
- เกิดปัญหา Synchronization ระหว่างสำเนาของข้อมูล และแก้ไขแบบ Manual

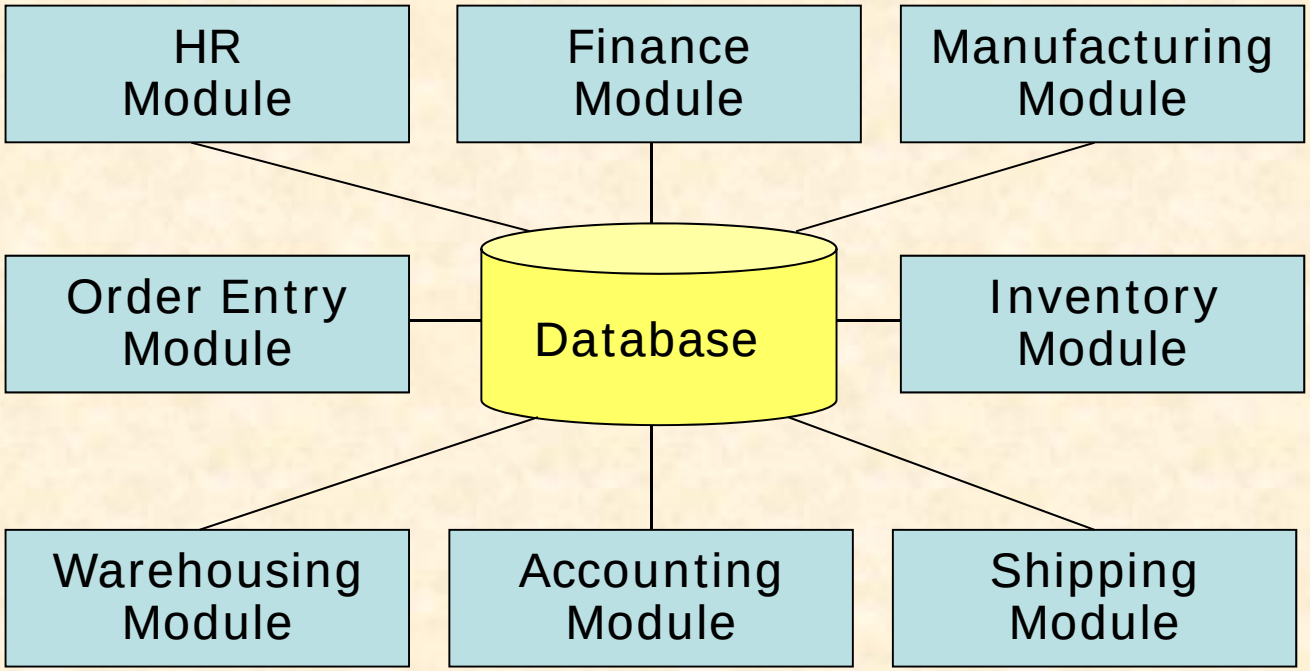
Data Silos (2)


Order Entry
Warehousing
Finance &
Accounting
Shipping &
Receiving
Inventory

ERP Application Suite (1)

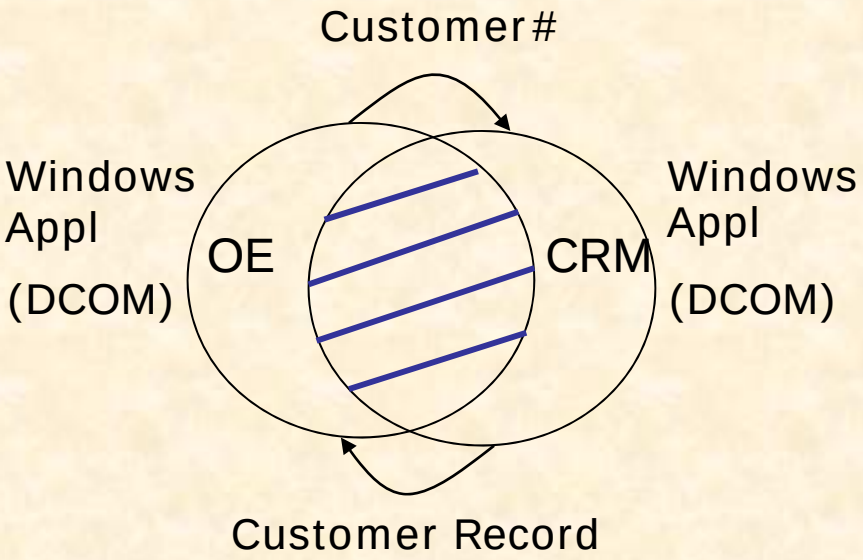
- Enterprise Resource Planning Application Suite เป็นที่นิยมตั้งแต่ปี 1980
- เป็น Integrated System ที่ประกอบด้วยหลาย Module มีฐานข้อมูลกลางร่วมกัน
- ทำให้เกิด Enterprise-Wide Integration ซึ่งแต่ก่อนทำไม่ได้
- ต้นทุนสูง ส่วนใหญ่เป็นค่า System Integration และ Customization
  - Module ไม่ยืดหยุ่น แก้ไขยาก
  - การปรับเปลี่ยนที่ส่วนหนึ่งมักกระทบส่วนอื่น
- External System ติดต่อด้วยยาก
  - การผนวกรวม ERP Systems สืบเนื่องจากการผนวกรวมองค์กรทำได้ยาก
  - เกิด Data Silos ใน Scale ที่ใหญ่ขึ้น

ERP Application Suite (2)



แนวทางการทำ Integration แบบที่ 1

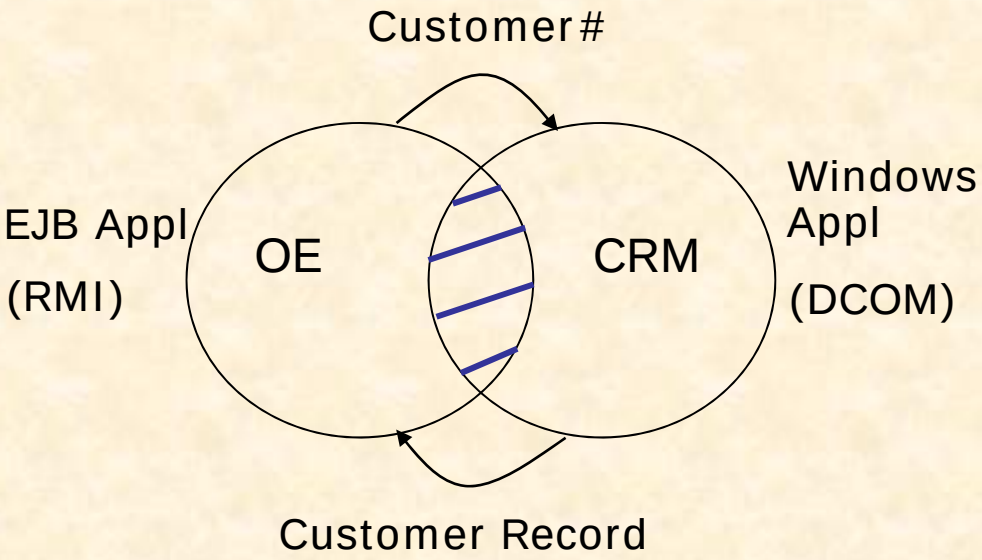
- ใช้เทคโนโลยีที่ Common กันของแต่ละระบบเป็นพื้นฐาน



อาจใช้ภาษาโปรแกรมเดียวกัน

ใช้ Common Comm Protocol ในการส่งผ่านข้อมูลระหว่างกัน

การแปลงข้อมูลมีเล็กน้อย



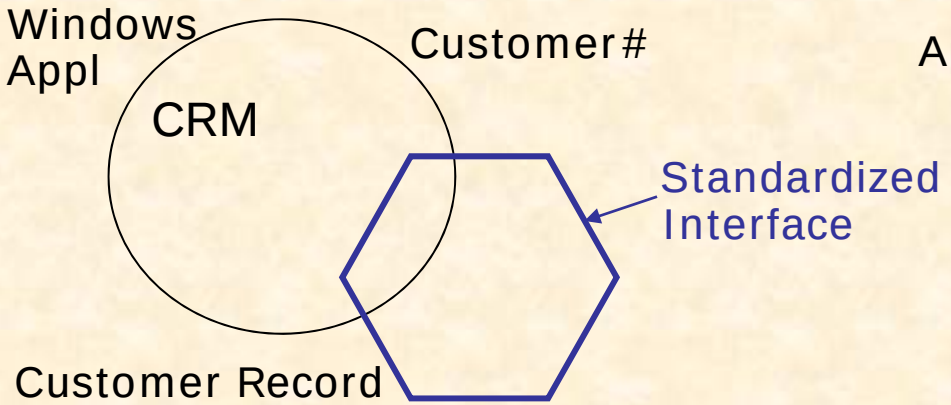
ต้องหา Compatible Comm Protocol

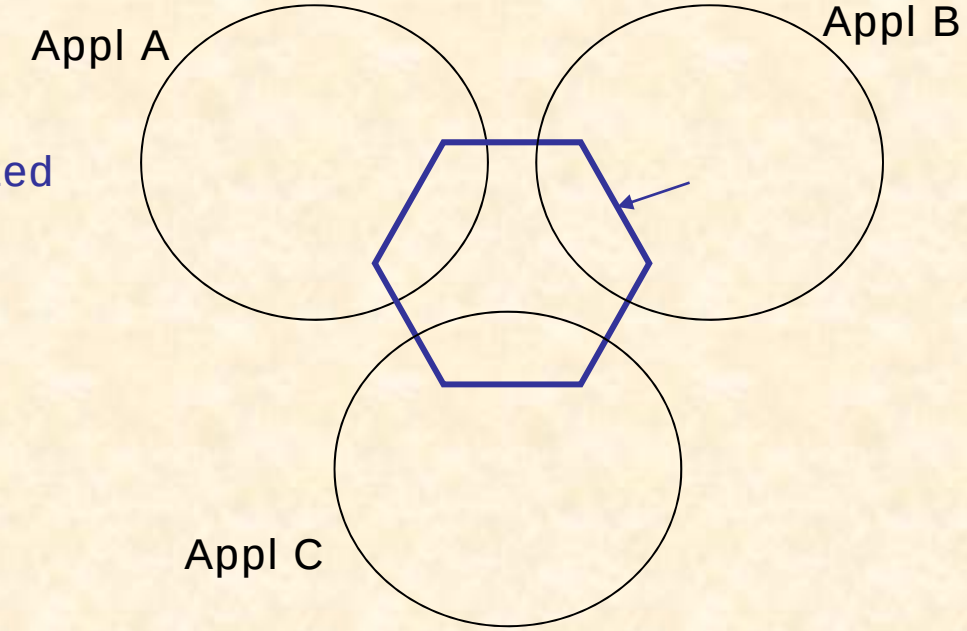
ต้องตีความ Data Representation

ยึดรูปแบบข้อมูลของฝั่งหนึ่งแล้วเขียนโค้ดที่อีกฝั่งเพื่อแปลงข้อมูล

แนวทางการทำ Integration แบบที่ 2

- ใช้เทคโนโลยีที่ไม่ขึ้นกับแต่ละระบบเป็นพื้นฐาน





ระบบกำหนด Interface ตามมาตรฐานเพื่อติดต่อกับระบบอื่นทั้งในและนอกองค์กร

มาตรฐานไม่ขึ้นกับเทคโนโลยีของระบบใดโดยเฉพาะ

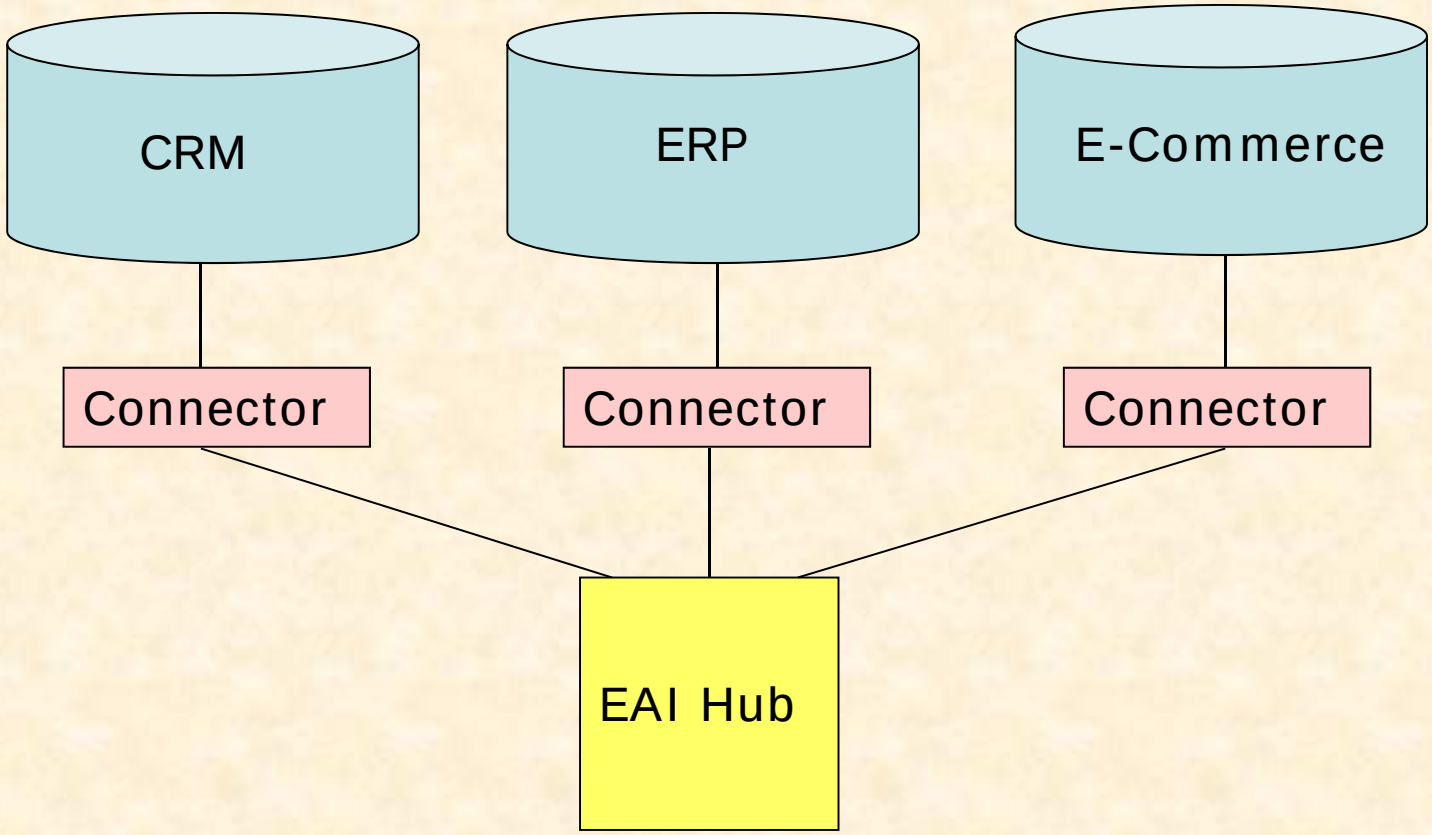
ไม่ต้องการสมมติฐานว่าระบบมีส่วน Common กัน

การแปลงข้อมูลทำระหว่างระบบกับมาตรฐานของ Interface

EAI Package (1)

- Enterprise Application Integration Package ใช้แนวทางการทำ Integration แบบที่ 2
- เป็นที่นิยมตั้งแต่ปี 1990
- เป็น Communication and Data Translation Platform ระหว่าง ERP System กับ External System
  - EAI Hub เป็น Broker Server
  - Connector เป็น Adapter ที่ Vendor มีให้ หรือใช้ Toolkit พัฒนาเอง
    - แปลงระหว่างข้อมูลของระบบกับข้อมูลในรูปแบบกลางที่กำหนด
- สนับสนุน Workflow ระหว่างระบบภายในองค์กรเท่านั้น
- มีปัญหาในการแปลงข้อมูลที่รูปแบบหรือความหมายต่างกัน
- ต้นทุนสูง ส่วนใหญ่เป็นค่า System Integration และ Professional Service

EAI Package (2)



โจทย์การพัฒนาแบบ Application Integration

- แก้ปัญหา Data Silos
- ใช้ Standardized API แทน Centralized Hub และ Connector
- สนับสนุนการพัฒนาแบบ Modular และ Incremental เพื่อการปรับเปลี่ยนองค์ประกอบใน Application ได้อย่างยืดหยุ่น
- ลดต้นทุน Consultant และการ Maintenance ระบบ
- สนับสนุน Business Process Integration ระหว่างองค์กร ไม่ใช่เพียง Batch Update Integration

SOA

หลักการของ SOA

SOA คืออะไร

- Software Architecture[1] คือ

  "The **fundamental organization** of a system, embodied in its **components**, their **relationships** to each other and the environment, and the **principles** governing its design and evolution"

- Service-Oriented Architecture[2] คือ

  "A **software architecture** within which all functions are defined as independent **services** with well-defined invokable **interfaces**, which can be called in defined sequences to form **business processes**"

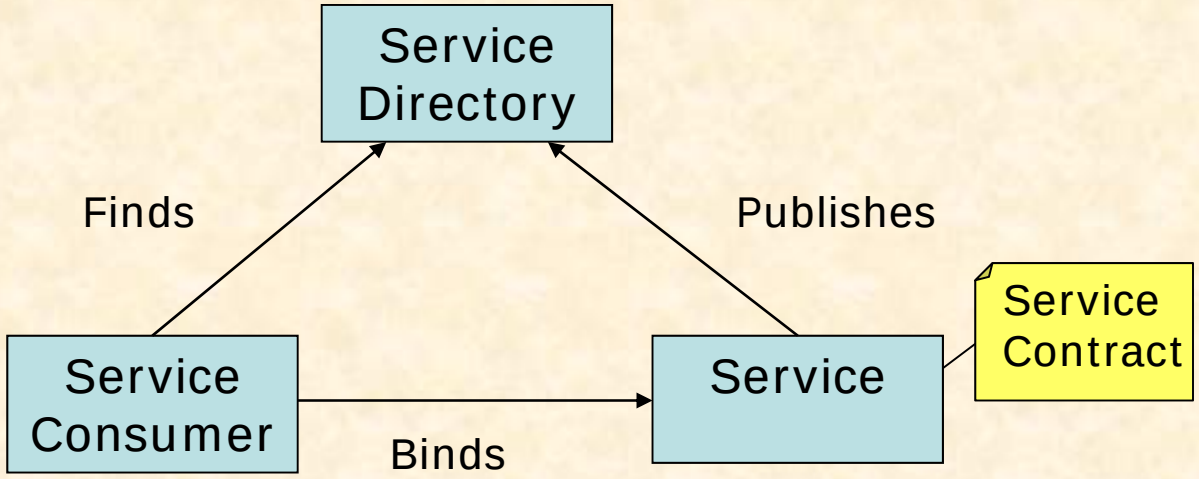
[1]IEEE 1471-2000

[2]Introduction to Service-Oriented Architecture (SOA), http://www.devshed.com/c/a/Web-Services/Introduction-to-Service-Oriented-Architecture-SOA/

พื้นฐานของ SOA

- ใช้แนวทางการทำ Integration แบบที่ 2
- Service เป็น Building Block
  - Software Module ที่ให้บริการฟังก์ชันการทำงานเฉพาะอย่าง
  - กำหนด Interface มาตรฐานซึ่งมีลักษณะ Platform-, Language- และ OS-Independent ไว้เป็น Contract
- สนับสนุน Loose Coupling ระหว่าง Service กับซอฟต์แวร์ที่เรียกใช้
- ใช้ Connection Technology ในการติดต่อ Service
  - Web Services Technology ได้รับความนิยม
- ซอฟต์แวร์อื่นค้นหา Service ได้แบบ Dynamic
- Service ภายในและภายนอกองค์กรถูกเรียกใช้ต่อเนื่องกันตาม Business Process ได้
- รองรับ Legacy Application ที่มีอยู่แล้ว

องค์ประกอบของ SOA



Loose Coupling

- หลักการออกแบบ Application ที่เน้นการปรับเข้ากับสิ่งที่เปลี่ยนแปลงไปได้โดยง่าย (Agility)
  - ลดความขึ้นต่อกันระหว่างองค์ประกอบ มีความคล่องตัว
- เป็นเป้าหมายของ SOA
- มีหลายแง่มุม

Loose Coupling ใน SOA (1)

- แง่มุม Heterogeneous Technologies
  - SOA ใช้ Standardized Interface แทนการใช้ Standardized Code
  - อิสระในการพัฒนา Service ด้วยเทคโนโลยีใดก็ได้
    - ได้ Technology Decoupling โดยยอมเสีย Interface Coupling
- แง่มุม Data Exchange
  - SOA ใช้รูปแบบข้อมูลที่อิสระกับเทคโนโลยี
  - หลีกปัญหา Low-Level Data-Type Compatibility
- แง่มุม Published Schema
  - Service Directory ช่วยในการตกลงเรื่องการให้บริการอย่าง Dynamic
- แง่มุม Delayed Binding
  - Binding ทำตอนเรียกใช้ Service และยึด Standardized Interface
  - การเปลี่ยนแปลง Service Implementation ไม่กระทบผู้ใช้

Loose Coupling ใน SOA (2)

- แง่มุม Fault Tolerance
  - Connection Technology รองรับการติดต่อที่สามารถ Decouple Service ออกจาก Application ที่เรียกใช้
  - ลด Sensitivity ต่อ Latency และ Reliability ของ Service
- แง่มุม Data Semantics
  - SOA สนับสนุนการสร้าง Adapter Service แทนการ Recode
  - สองฝ่ายมี Data Semantics ที่ต่างกันได้
- แง่มุม Applicability
  - Service สามารถ Reuse ได้กับหลาย Application ในหลายประเภท

หลักการของ Service

Service คืออะไร

- Service เป็น Software Module ที่ให้บริการฟังก์ชันการทำงานเฉพาะอย่าง
- มีคุณลักษณะ เช่น
  - ชื่อ
  - ฟังก์ชันงานตามธุรกิจ
  - ที่ติดต่อ
  - นโยบายเกี่ยวกับ Security

Service Granularity

- ระดับรายละเอียดของขอบเขตงานที่กำหนด Service Interface
- ขอบเขตเล็ก เช่น การดึงข้อมูล
  - Customer Lookup
  - Account Lookup
- ขอบเขตใหญ่ เช่น การทำงานตาม Business Process
  - Order Processing
  - Hotel Reservation
  - Loan Processing
- Service ควรจะ Coarse-Grained

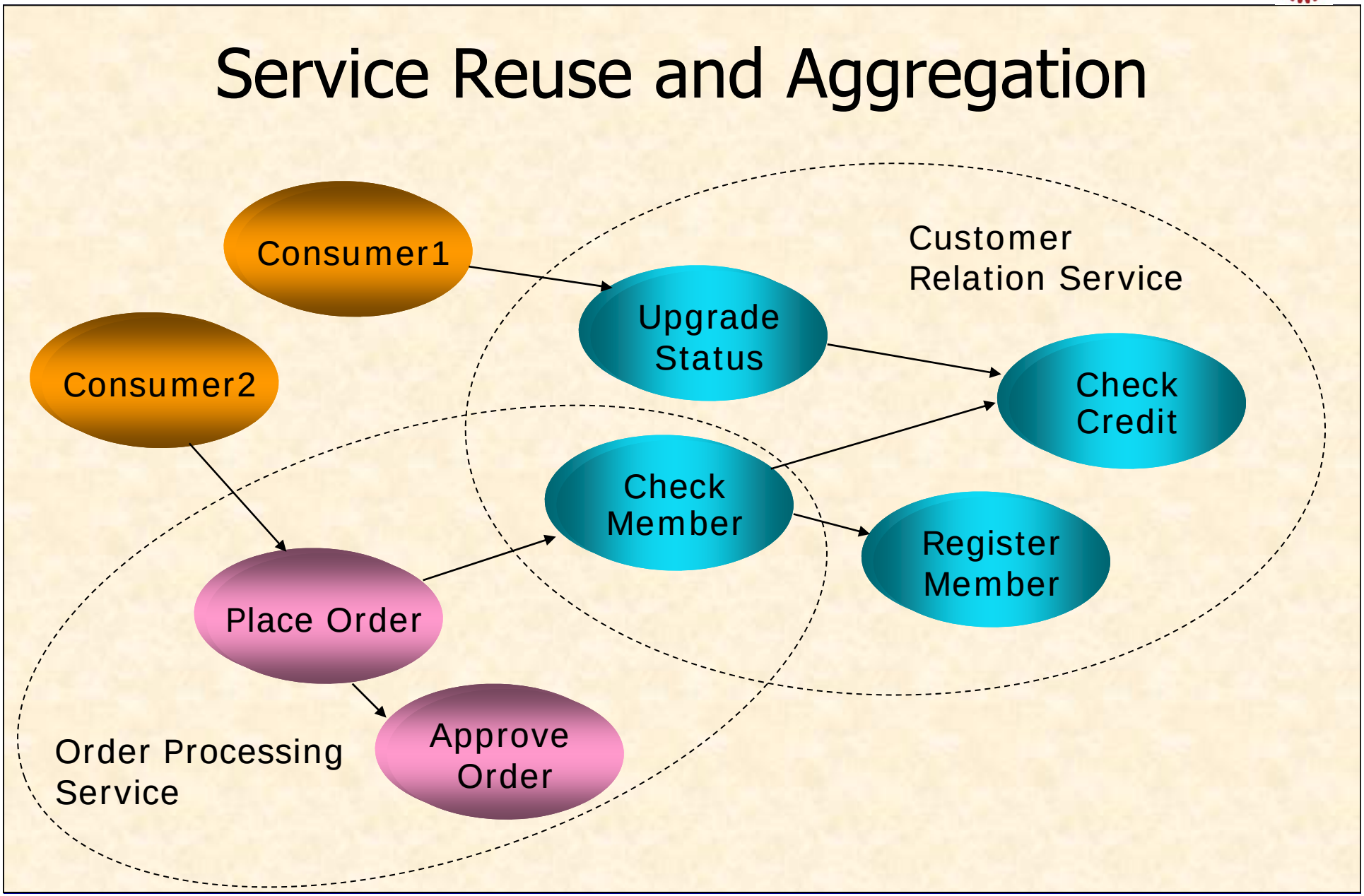
ประเภทของ Service

- Intra-System Service
  - ให้บริการภายในระบบหนึ่ง ๆ ซึ่ง Tightly-Coupled
  - อาจไม่ถือเป็น Service ก็ได้ เพราะไม่ช่วยการทำงานข้ามระบบ
- Internal Service
  - Behind-the-Firewall Service
  - ให้บริการระหว่างระบบภายใน Trust Domain หนึ่ง ๆ
  - ทั้ง Consumer และ Provider ควบคุมโดย Single Authority เช่น บริษัท หรือฝ่าย
- External Service
  - Beyond-the-Firewall Service
  - ให้บริการทั้งภายในและภายนอก Trust Domain หนึ่ง ๆ ทั้งแก่คู่ค้าและสาธารณะ
  - Consumer และ Provider ควบคุมโดย Authority ที่ต่างกัน
  - ต้องจัดการประเด็น Security, Business Agreement, Data Semantics

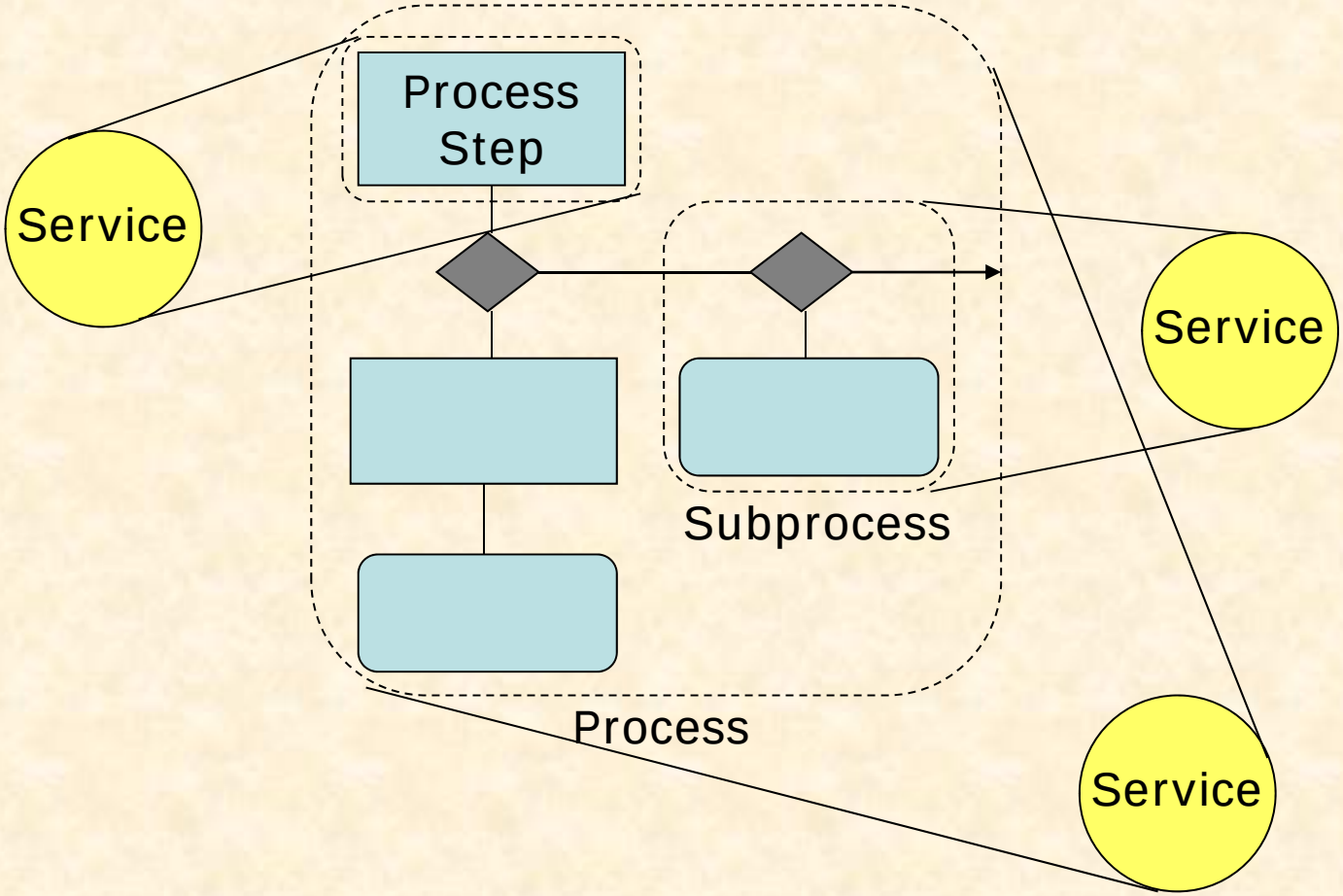
Service Collaboration

- Orchestration
  - Primary Service เรียกใช้ Service อื่น ๆ ตามลำดับ
  - ลำดับการทำงานตาม Business Logic ฝังอยู่ใน Primary Service
- Business Interaction
  - Coordination Mechanism จัดการลำดับการทำงานของ Long-Lived Service
  - Coordination Mechanism อาจเป็น Business Process Execution Engine, Work Flow Engine หรือ Enterprise Service Bus
- Interception
  - Intermediary Service รับ-ส่งข้อความระหว่างผู้ส่งกับ Service ปลายทางอย่าง Transparent

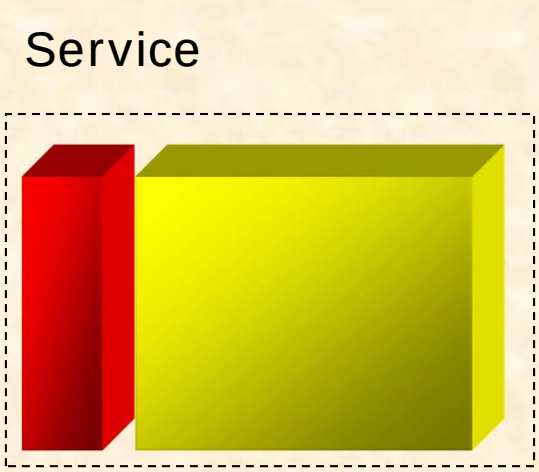
Service Reuse and Aggregation



Service Encapsulates Logic

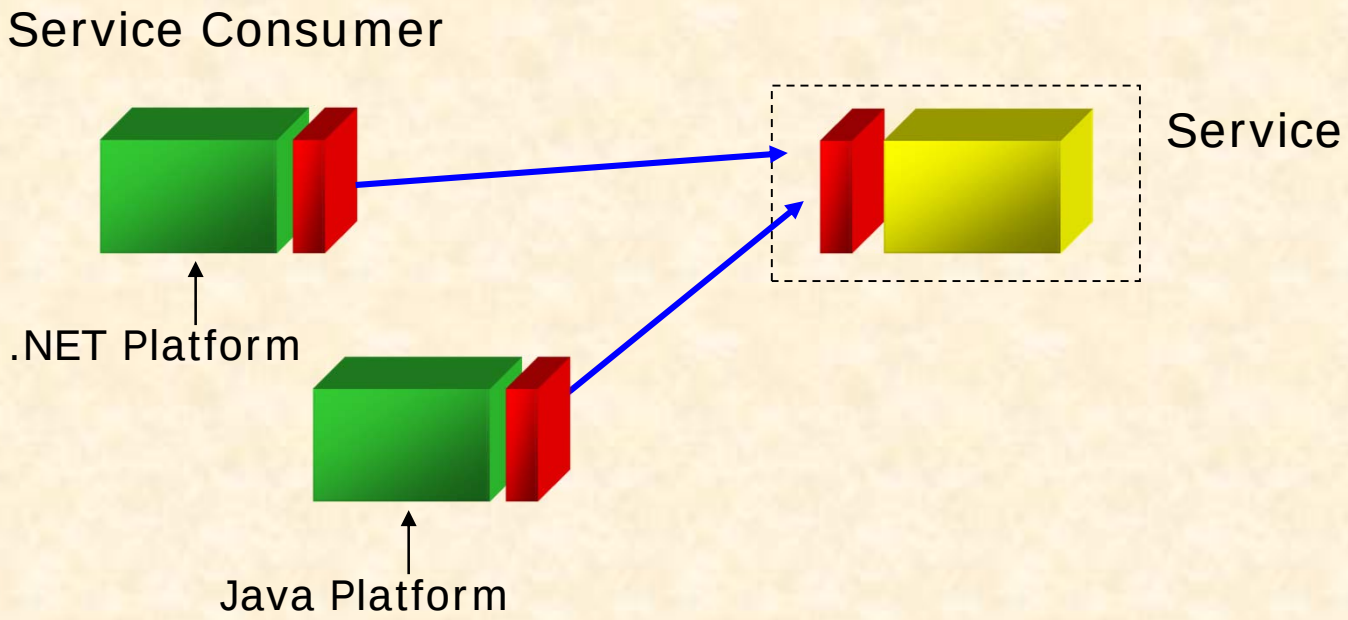


Interface vs Application Implementation



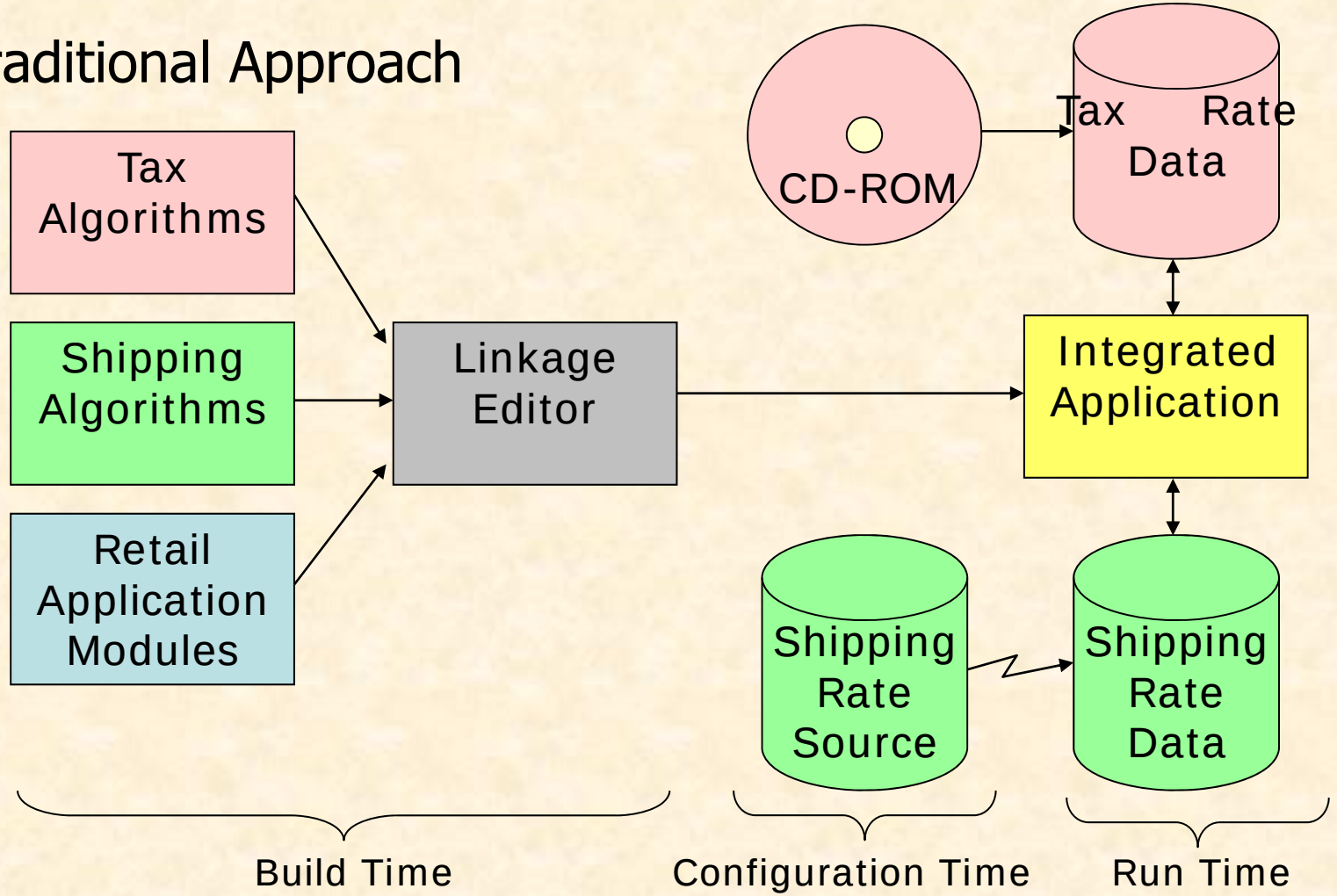
สามารถเปลี่ยน Implementation
โดยไม่กระทบ Service Interface

Write Once, Access from Anywhere



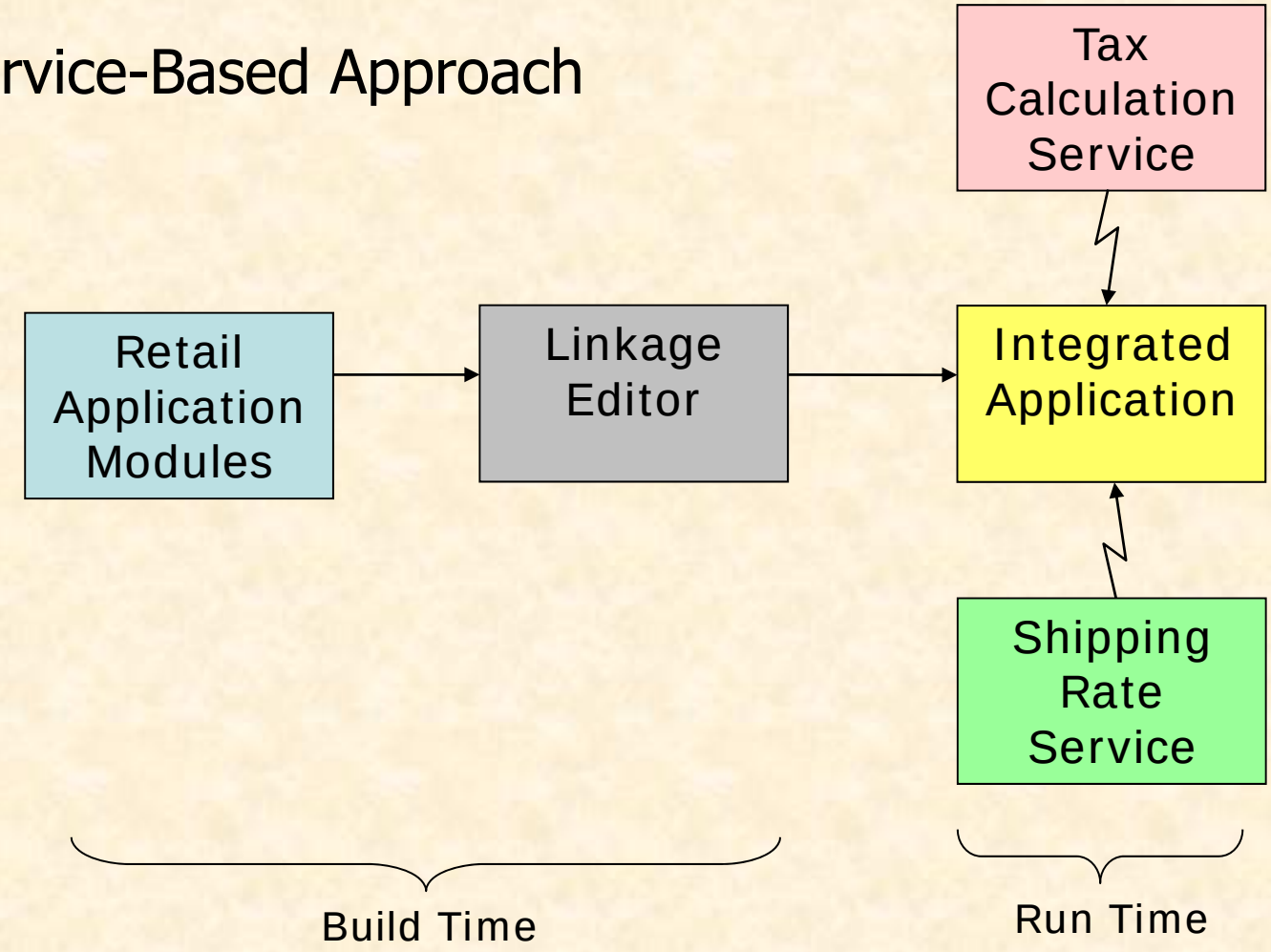
Application Architecture ที่เปลี่ยนไป (1)

- Traditional Approach

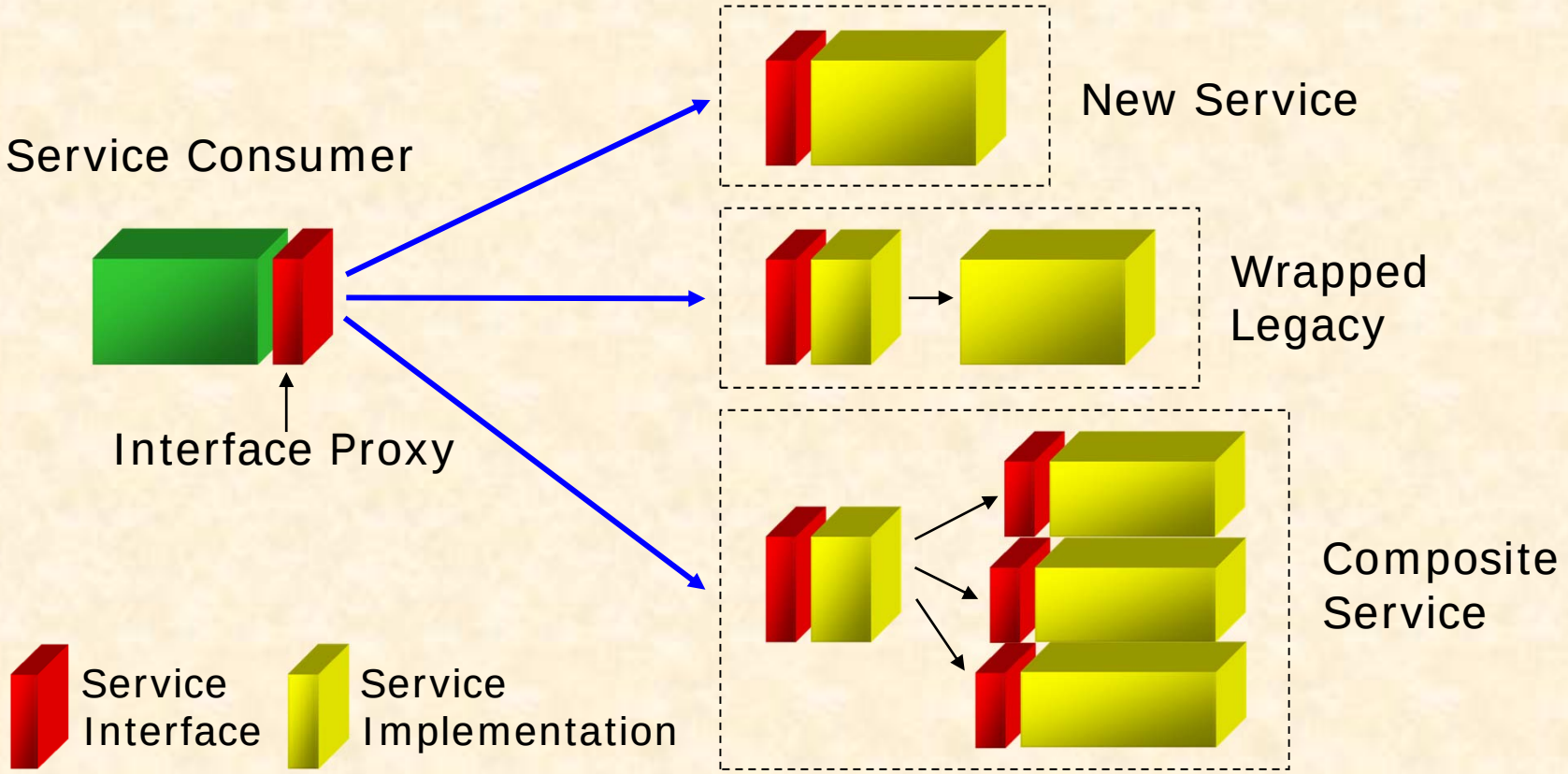


Application Architecture ที่เปลี่ยนไป (2)

- Service-Based Approach



สร้าง Service ได้หลายแบบ



การออกแบบ Service

- ออกแบบ Interface
- ออกแบบ Implementation
- ออกแบบข้อมูลที่แลกเปลี่ยน
- ออกแบบ Service Contract
- ออกแบบความสัมพันธ์กับ Service อื่น

หลักการของ Connection Technology

Connection Technology คืออะไร

- เทคโนโลยีเพื่อการเชื่อมต่อและสื่อสารระหว่าง Disparate Systems
- วิวัฒนาการ
  - CORBA (Common Object Request Broker Architecture)
    - Platform-, Language- และ OS-independent
    - ต้องใช้ CORBA Protocol (IIOP) และ Application ส่วนใหญ่เป็น OO
    - ซับซ้อนและต้นทุนสูง
  - DCOM (Distributed Component Object Model)
    - เชื่อมต่อ Object ใน Windows Application บนคอมพิวเตอร์ต่างเครื่อง
    - Application พัฒนาด้วยหลายภาษาได้
  - RMI (Remote Method Invocation)
    - เชื่อมต่อ Object บนคอมพิวเตอร์ต่างเครื่องซึ่งพัฒนาด้วยภาษา Java
    - Object อยู่บนหลายแพลตฟอร์มได้
- อาศัยหลักการของ Service ที่ให้บริการผ่าน Interface

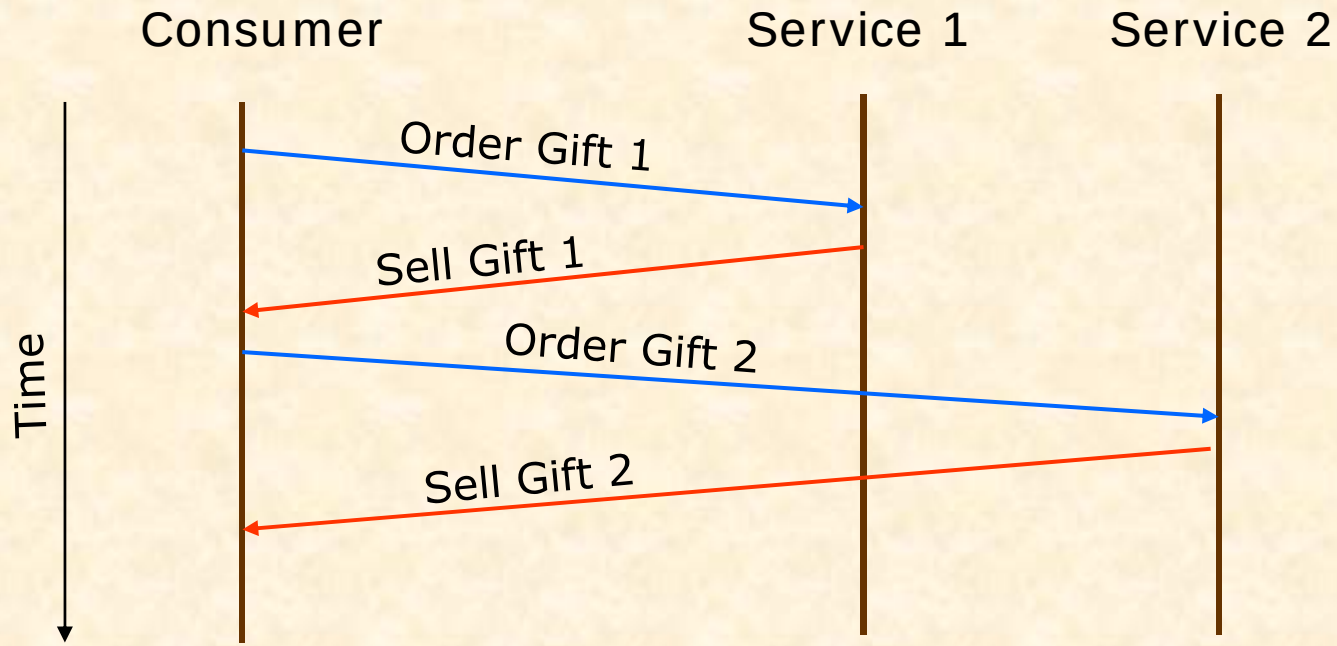
Web Services

- เป็น Connection Technology ที่ได้รับความนิยม
- ช่วย SOA ให้บรรลุเป้าหมาย Loose Coupling ได้ดี
  - ใช้หลายมาตรฐานที่เกี่ยวกับการสื่อสารและเป็นที่แพร่หลายแล้ว ได้แก่ HTTP, XML (แก้ปัญหา CORBA)
  - Platform-Independent เพราะ Service Interface สามารถแยกจาก Service Implementation (แก้ปัญหา DCOM)
  - Language-Independent เพราะ Service Interface สามารถแยกจาก Service Implementation (แก้ปัญหา RMI)

Synchronous Messaging (1)

- Consumer ส่ง Request แล้วหยุดรอ Response
  - พัฒนาง่าย ไม่ต้อง Correlate Request กับ Response
- ใช้กับการ Query และ Update ง่าย ๆ เช่น
  - Stock Quote
  - Weather Report
  - Delivery Tracking
- ไม่เหมาะกับ Service ที่ต้องการ Decouple Request จาก Response

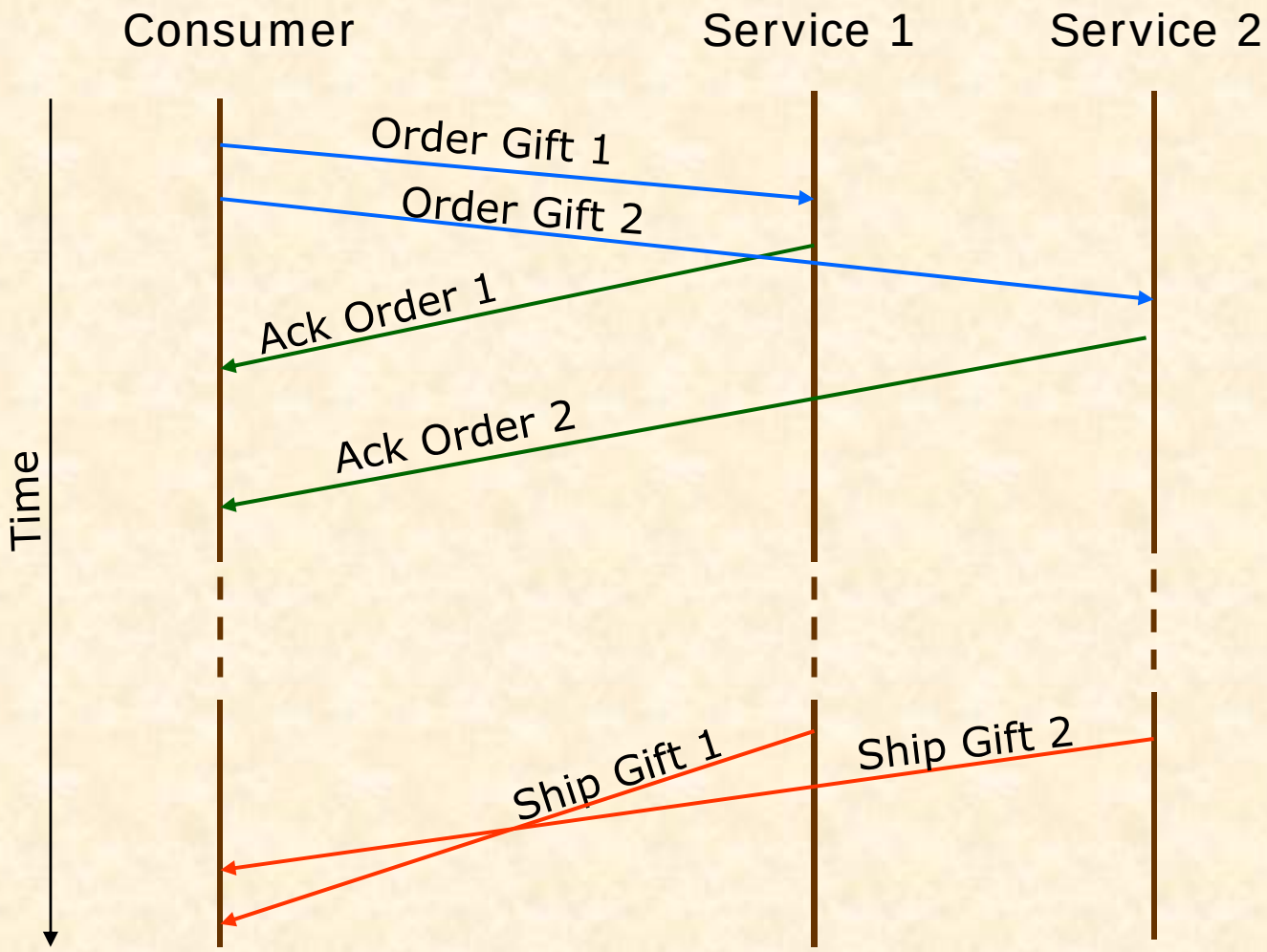
Synchronous Messaging (2)



Asynchronous Messaging (1)

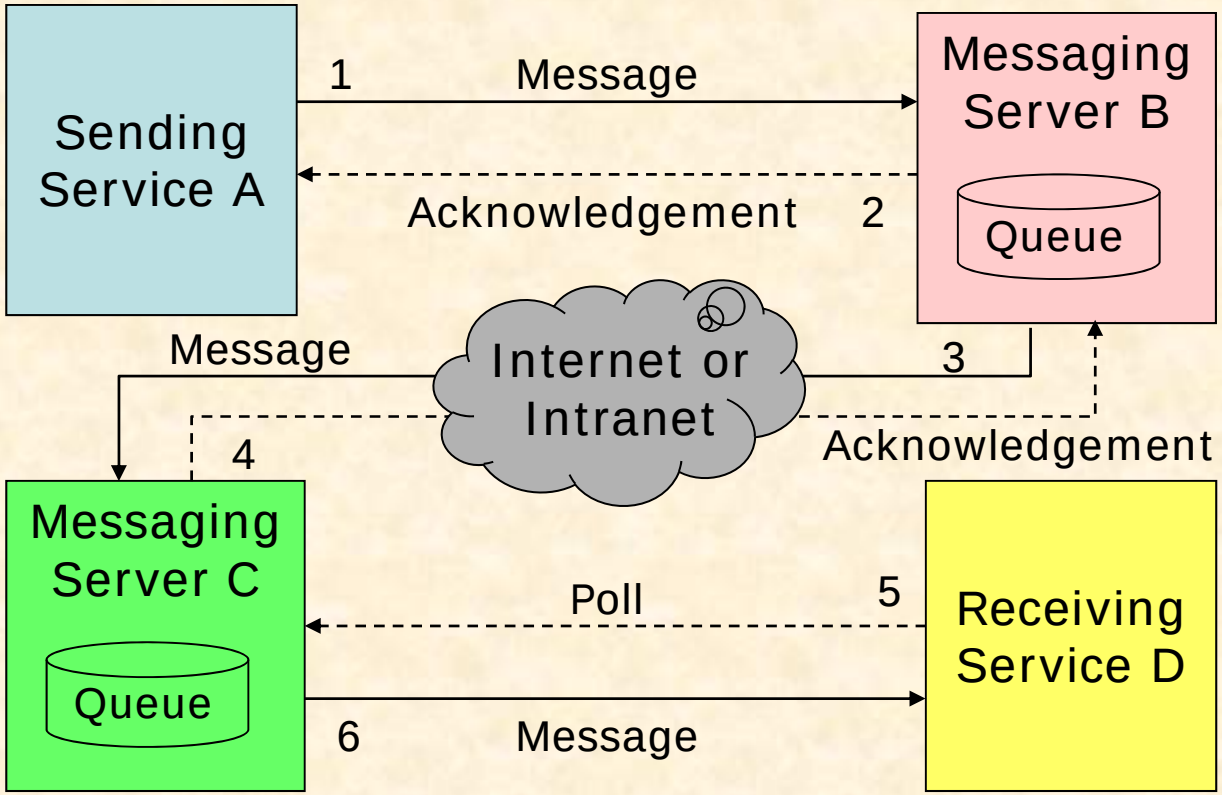
- Consumer ส่ง Request แล้วไม่หยุดรอ Response
  - อาจจะได้รับ Acknowledgement
- Service ประมวลผลในภายหลัง และส่ง Response กลับ
- Consumer ต้อง Correlate Response กับ Request
- สอดคล้องกับการเรียกใช้ Service ใน Real-World Business Process

Asynchronous Messaging (2)



Asynchronous Messaging (3)

- Message Queuing ช่วยจัดการเรื่อง Load และ Failure ของ Service



RPC Style

- มักใช้ติดต่อ Service แบบ Synchronous Request/Response (แต่อาจเป็น Asynchronous และอาจไม่มี Response)
- Message ระบุชนิดข้อมูลทั้งแบบพื้นฐาน และแบบ Compound ที่แลกเปลี่ยนกันในการเรียกใช้ฟังก์ชันของ Service
- อาจจะ Fine-Grained เกินไปสำหรับการติดต่อในธุรกิจจริง

Document Style

- ใช้ติดต่อ Service โดยส่ง Self-Contained Document ให้
- เหมาะกับการติดต่อในเชิงธุรกิจ เช่น ส่งทั้ง Purchase Order
- เหมาะกับการติดต่อแบบ Asynchronous (แต่เป็น Synchronous ก็ได้)
- Service ใช้ข้อมูลเท่าที่จำเป็นใน Document
- Service ส่งต่อ Document ให้ Service อื่นได้
- จำนวน Message สำหรับการทำงานที่ซับซ้อน จะน้อยลงกว่าแบบ RPC

ตอบคำถาม

Q1: ถ้าไม่ใช้ Web Services จะทำ SOA ได้หรือไม่

A1: เทคโนโลยี เช่น CORBA และ RMI อาศัยหลักการของ Service และการ Publish Standard Interface รวมทั้งสามารถทำ Service Aggregation ได้ แต่เนื่องจากมี Technology Coupling จึงตอบสนองหลักการของ SOA ได้ระดับหนึ่งเท่านั้น

Q2: อะไรที่ต้องมีใน Application จึงจะถือว่าเป็นไปตาม SOA ถ้าไม่มีจะถือว่าไม่เป็น

A2: อย่างน้อยต้องมี Service เพราะเป็น Building Block อย่างไรก็ตาม การออกแบบ Service ต้องคำนึงถึงหลักการต่าง ๆ ของ SOA ด้วย เช่น การมี Loose Coupling การมี Granularity ที่เหมาะสมกับการให้บริการ

Q3: ถ้า Application เรียกใช้เพียง 1 Service จะถือว่าเป็น SOA แล้วหรือไม่

A3: ยังไม่พบข้อกำหนดเกี่ยวกับจำนวน Service ที่ใช้ กับความเป็น SOA ดังนั้น น่าจะถือว่าเป็น SOA ได้ แต่สิ่งที่สำคัญกว่าจำนวน ก็คือการสร้าง Application และ Service นั้นได้คำนึงถึงหลักการของ SOA แล้วหรือไม่

Q4: Service จำเป็นต้อง Reuse โดยผู้อื่นหรือไม่ ถ้าสร้างไว้ใช้เอง จะถือว่าเป็นไปตามหลักการของ SOA หรือไม่

A4: การกำหนด Service สามารถใช้ Reusability เป็นแนวทางได้ คือหากฟังก์ชันหนึ่งถูกใช้ซ้ำ ๆ โดยหลาย Application เราก็จะกำหนดเป็น Service ขึ้นมา แต่ Reusability ไม่ใช่สาเหตุเดียวในการสร้าง Service สมมุติว่ามีเพียง Application เดียวที่ใช้ฟังก์ชันหนึ่ง แต่ฟังก์ชันนั้นมีการเปลี่ยนแปลงบ่อยมาก หรือมีปัญหาเรื่อง Heterogeneous Technology ในการติดต่อใช้งาน การสร้างฟังก์ชันนั้นเป็น Service ก็น่าจะเหมาะสม เพื่อช่วย Decouple ส่วน Interface ของการใช้งานฟังก์ชันออกจากส่วน Implementation

Q5: Service, Object และ Software Component เหมือนหรือต่างกันอย่างไร

A5: หลักการของ Service คล้ายคลึงกับหลักการของ Object และ Software Component ในแง่ที่การติดต่อส่วนประกอบในระบบทำผ่าน API ที่แยกออกจากส่วน Implementation และสนับสนุน Reuse แต่ก็มีความแตกต่างกัน

- แง่มุม Granularity:
  - Object จะ Fine-Grained โดยเป็นหน่วยเล็ก ๆ ในระดับโค้ด ส่วน Software Component จะ Abstract กว่า Object โดยเป็น Packaging ของ Object ภายใน
  - Service จะ Coarse-Grained กว่า และเป็นส่วนประกอบที่อยู่ระดับบนกว่า
- แง่มุม Reuse:
  - Object และ Software Component ทำให้เกิดการ Reuse Code แต่จะ Tightly-Coupled เพราะ Application ต่อ ๆ ไปที่จะพัฒนาด้วย Object หรือ Software Component นี้ จะพัฒนาด้วยภาษาเดิมหรือบนแพลตฟอร์มเดิม
  - Service ไม่ต้องการ Port Source Code หรือใช้ Library ในการสร้าง Application แต่ Code ของ Service จะ Execute ในที่ที่มันอยู่

Q6: Service จะมาแทนที่ Object และ Software Component หรือไม่

A6: หลักการของ Service เกี่ยวกับการมี Interface มาตรฐาน ไม่ได้เกี่ยวกับ Implementation ดังนั้น Service จะไม่ได้มาแทนที่ Object แต่สามารถใช้ Object เป็นเทคโนโลยีในการพัฒนาส่วน Implementation ของ Service ได้ เช่นเดียวกับแบบ Non-OO อย่างไรก็ตาม ในการใช้ OO ในการพัฒนา หากทำเพียงแค่สร้าง Wrapper ครอบ Fine-Grained Object แล้วเรียกว่า Service ก็จะไม่เหมาะสม

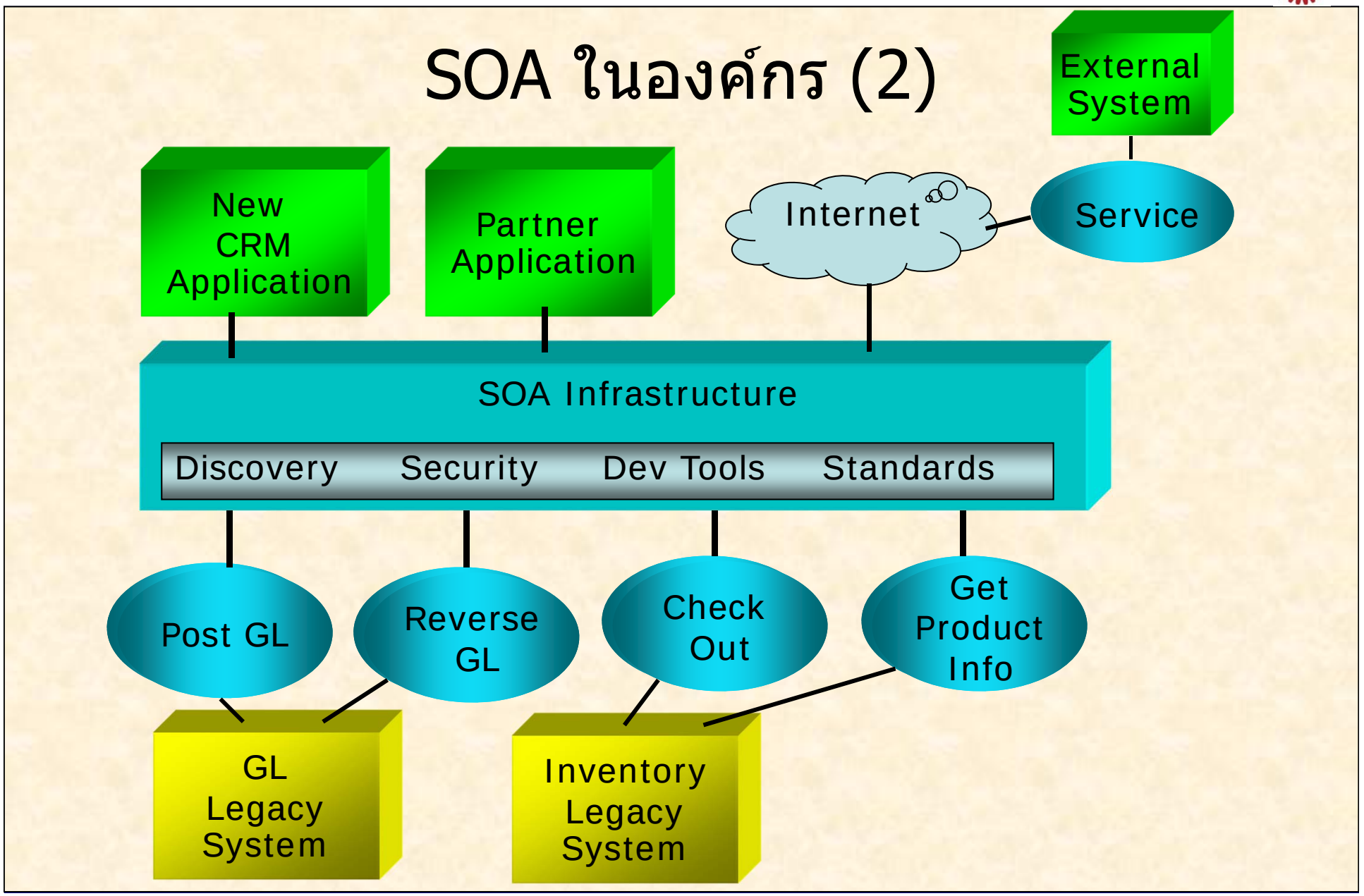
Q7: ควรเริ่มทำ SOA เมื่อไร และอย่างไร

A7: องค์กรสามารถเริ่มปรับตัวเพื่อทำ SOA ได้เลย ขณะนี้เทคโนโลยีมาตรฐานพื้นฐานเริ่มนิ่งแล้ว แต่ก็ยังมีอีกหลายส่วนที่ยังพัฒนาอยู่ การทำ SOA ในช่วงที่แนวคิดนี้ยังค่อนข้างใหม่อยู่ จะทำให้องค์กรมีความได้เปรียบ แต่ต่อไปเมื่อแนวคิดนี้เป็นที่แพร่หลายแล้วและองค์กรไม่ทำ SOA ก็จะกลายเป็นเสียประโยชน์ไป การเริ่มต้นอาจเริ่มจากการทำ Pilot Project เล็ก ๆ โดยทำเป็น Intra-System Service หรือ Internal Service แล้วค่อยขยายเป็น Internal Service ที่มีขอบเขตใหญ่ขึ้น หรือแม้กระทั่งให้บริการ External Service

บทสรุป

SOA ในองค์กร (1)

- เน้นการรวมระบบซอฟต์แวร์เข้าด้วยกันแบบองค์รวม โดยกำหนด Service และ Collaboration ระหว่างกัน
- ไม่ใช่เป็นเพียงการเรียกใช้ฟังก์ชันโดยอิสระจากเทคโนโลยีเท่านั้น แต่นำไปสู่การพัฒนา Long-Running Business Process ข้ามองค์กร
- มีเทคโนโลยีในการพัฒนา SOA หลากหลายแบบที่ควรเลือกให้เหมาะสม


SOA ในองค์กร (2)
External System
New CRM Application
Partner Application
Internet
Service
SOA Infrastructure
Discovery
Security
Dev Tools
Standards
Post GL
Reverse GL
Check Out
Get Product Info
GL Legacy System
Inventory Legacy System

ประโยชน์ของ SOA

- Application มีความยืดหยุ่น ปรับเปลี่ยนง่าย เพราะอิงมาตรฐานและถอดเปลี่ยน Service ได้
- สามารถพัฒนา Application ได้เร็ว โดยการประกอบ Service เข้าด้วยกัน
- ลดต้นทุนและความเสี่ยง ไม่ต้องพัฒนาและบำรุงรักษาทุกส่วนของ Application เอง
- สามารถนำระบบที่มีอยู่แล้วมาใช้ร่วมกันแบบองค์รวมได้ การลงทุนไม่สูญเปล่า

เอกสารอ้างอิง

- Debu Panda, "An Introduction to Service-Oriented Architecture from a Java Developer's Perspective", 26 January 2005. Available: http://www.onjava.com/pub/a/onjava/2005/01/26/soa-intro.html
- Jagadish Chatarji, "Introduction to Service-Oriented Architecture (SOA)", 13 October 2004. Available: http://www.devshed.com/c/a/Web-Services/Introduction-to-Service-Oriented-Architecture-SOA/
- Doug Kaye, "Loosely Coupled: The Missing Pieces of Web Services", RDS Press, 2003.
- Douglas K. Barry, "Web Services and Service-Oriented Architectures: The Savvy Manager's Guide", Morgan Kaufmann Publishers, 2003.