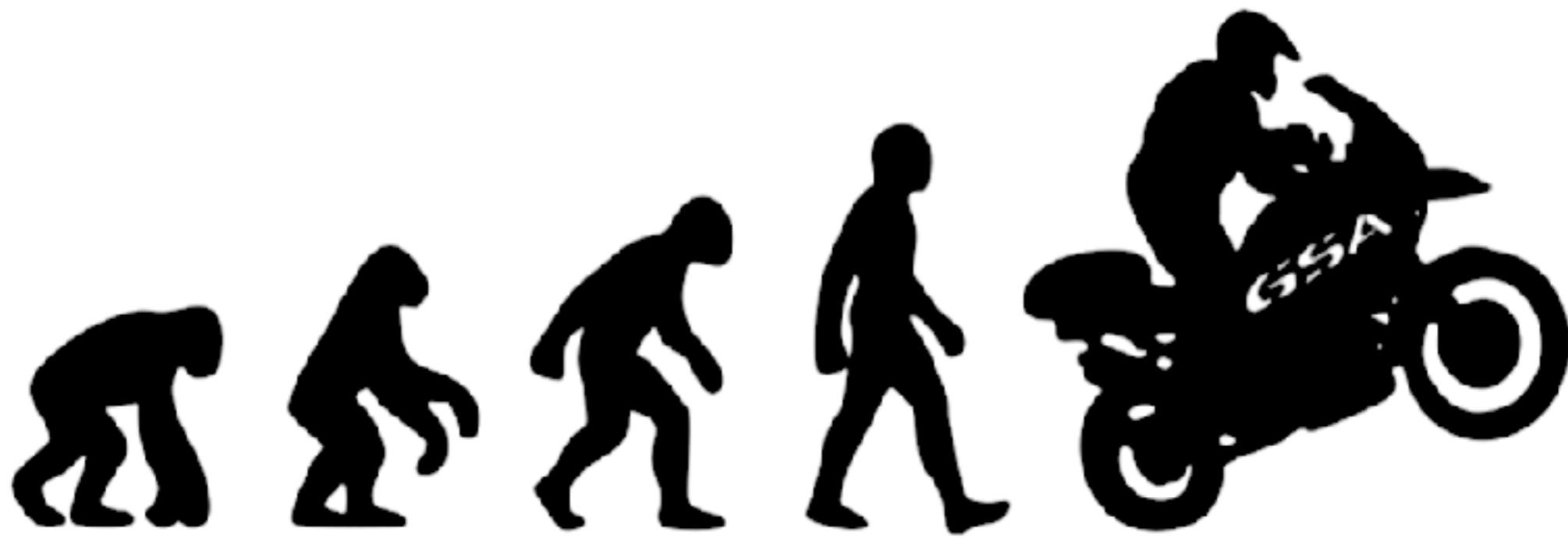


# Evolution of Architecture



monolith

n-tier

service  
oriented

microservices

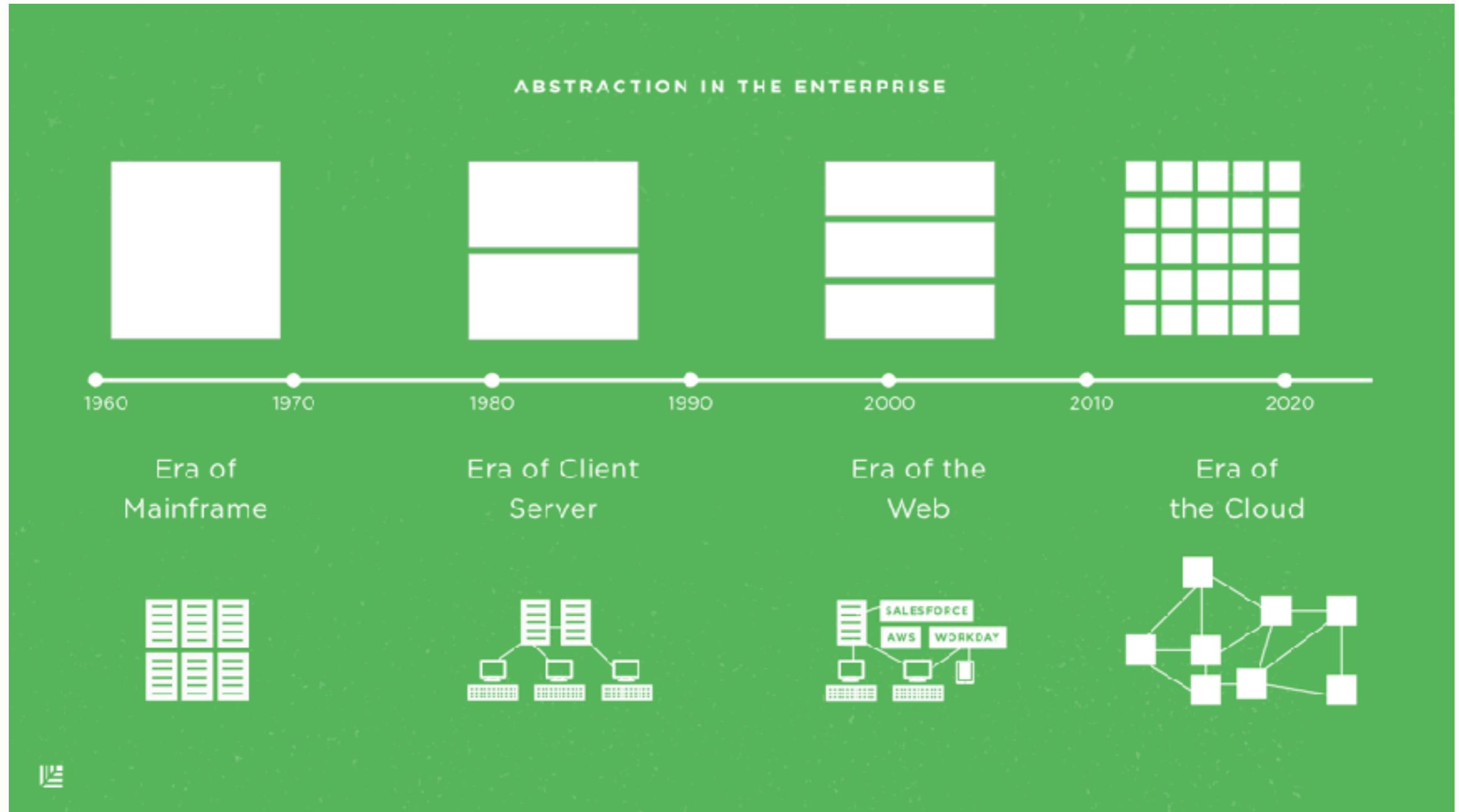
serverless  
 เช่น FaaS

1 Team/Service - silo'ed

@somkiat



# Evolution of Architecture



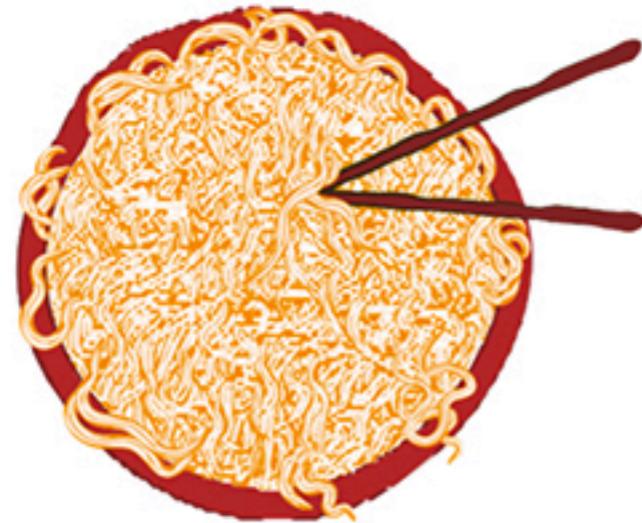
หัวใจของ Microservices คือการแบ่งออกจากกันเด็ดขาด



# Developer's perspective

1990s and earlier

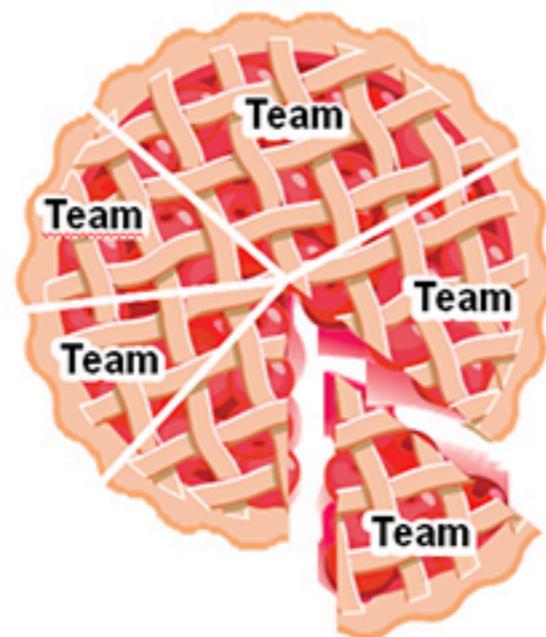
Pre-SOA (monolithic)  
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA  
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

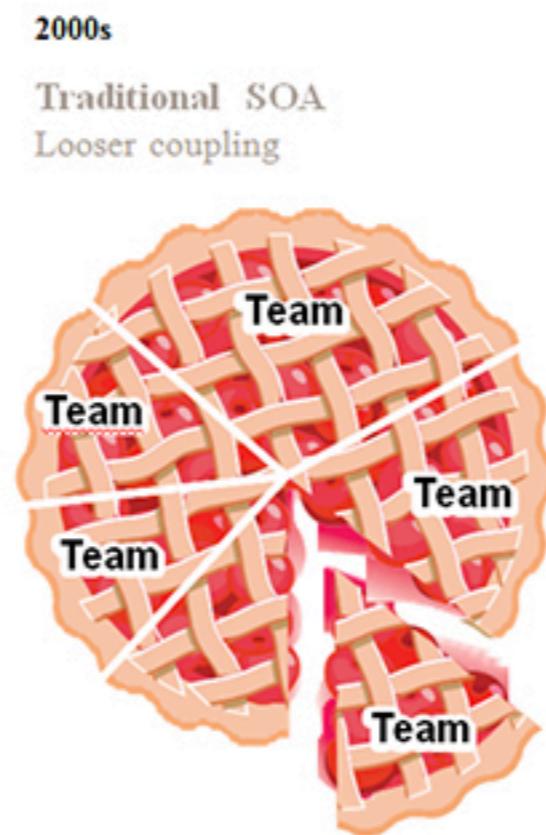
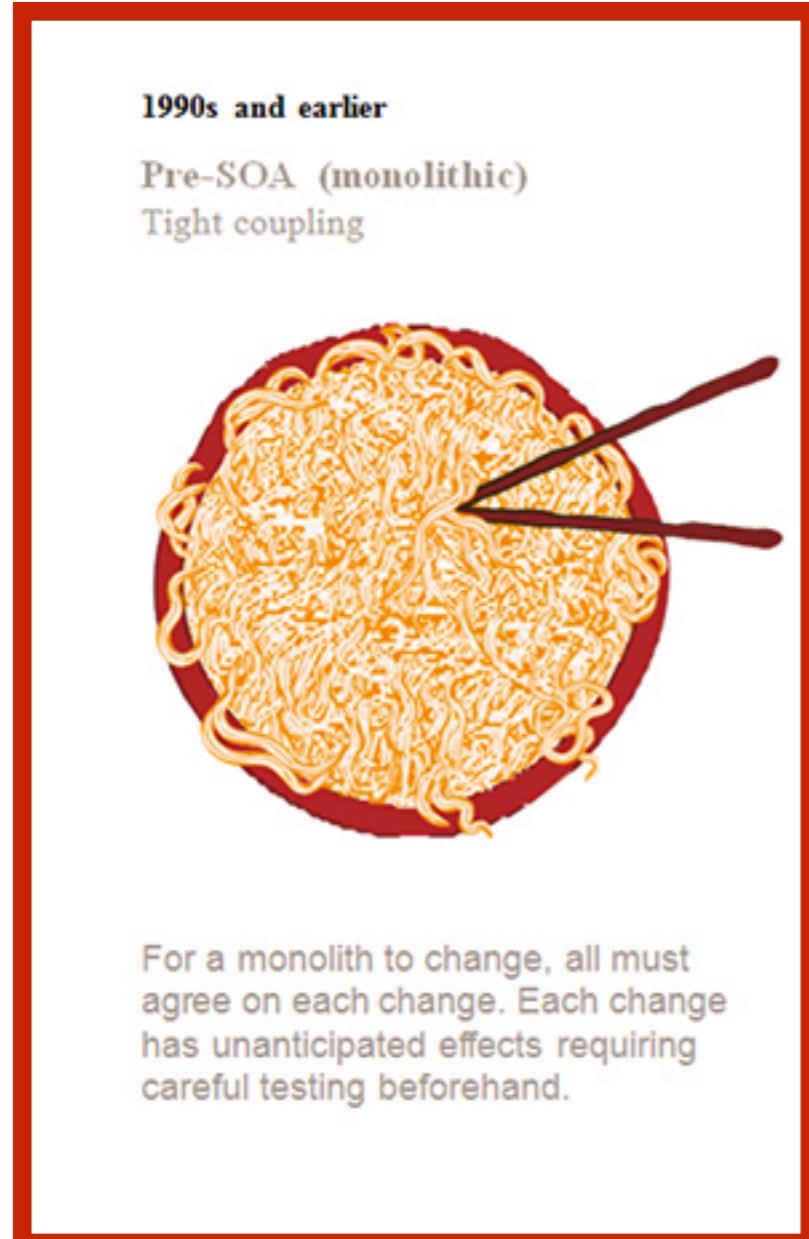
Microservices  
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



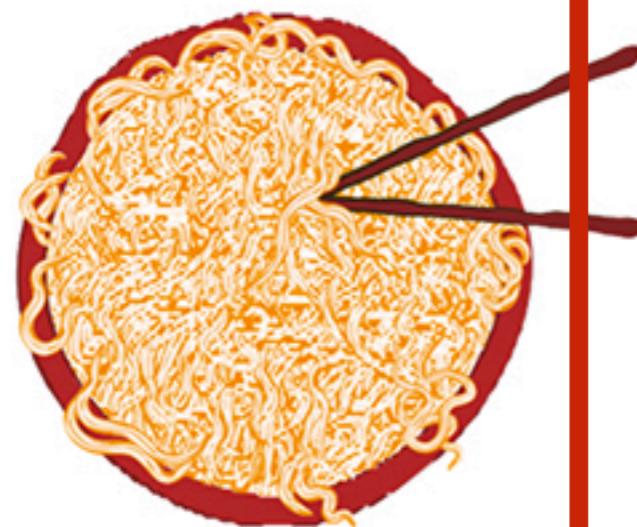
# Developer's perspective



# Developer's perspective

1990s and earlier

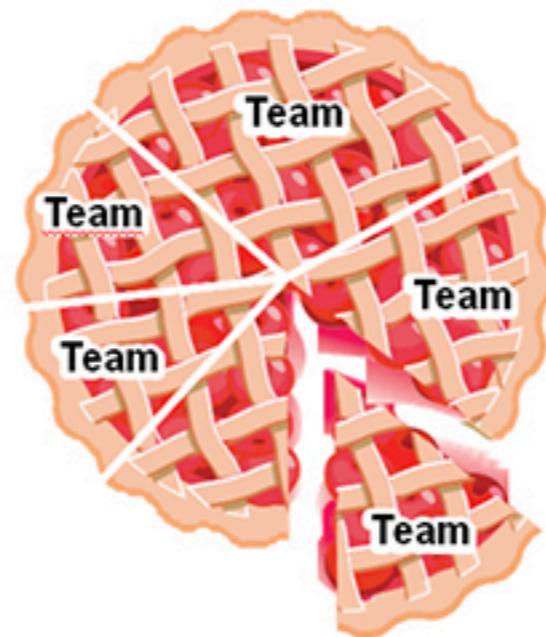
Pre-SOA (monolithic)  
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA  
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

Microservices  
Decoupled



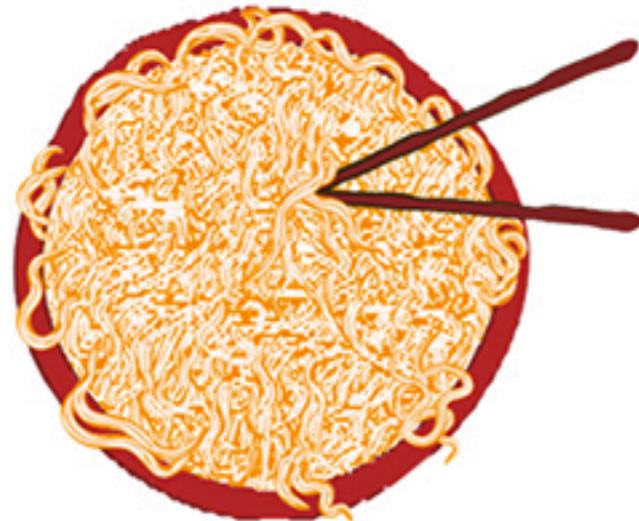
Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.



# Developer's perspective

1990s and earlier

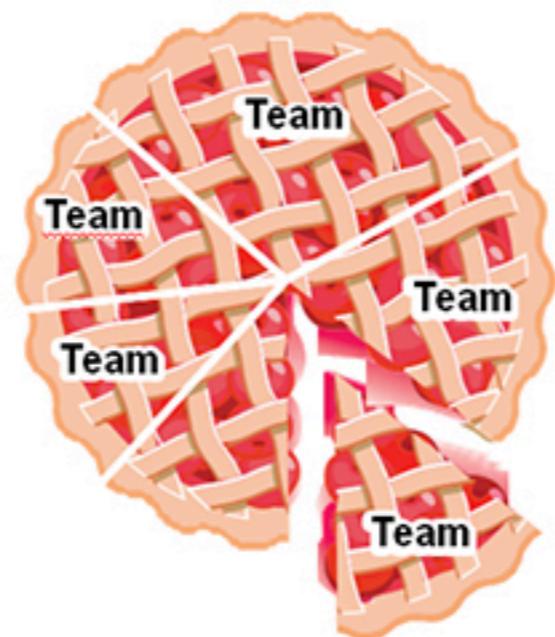
Pre-SOA (monolithic)  
Tight coupling



For a monolith to change, all must agree on each change. Each change has unanticipated effects requiring careful testing beforehand.

2000s

Traditional SOA  
Looser coupling



Elements in SOA are developed more autonomously but must be coordinated with others to fit into the overall design.

2010s

Microservices  
Decoupled



Developers can create and activate new microservices without prior coordination with others. Their adherence to MSA principles makes continuous delivery of new or modified services possible.

# Evolution of services

1990s and earlier

## Coupling

Pre-SOA (monolithic)

Tight coupling



2000s

Traditional SOA

Looser coupling



2010s

Microservices

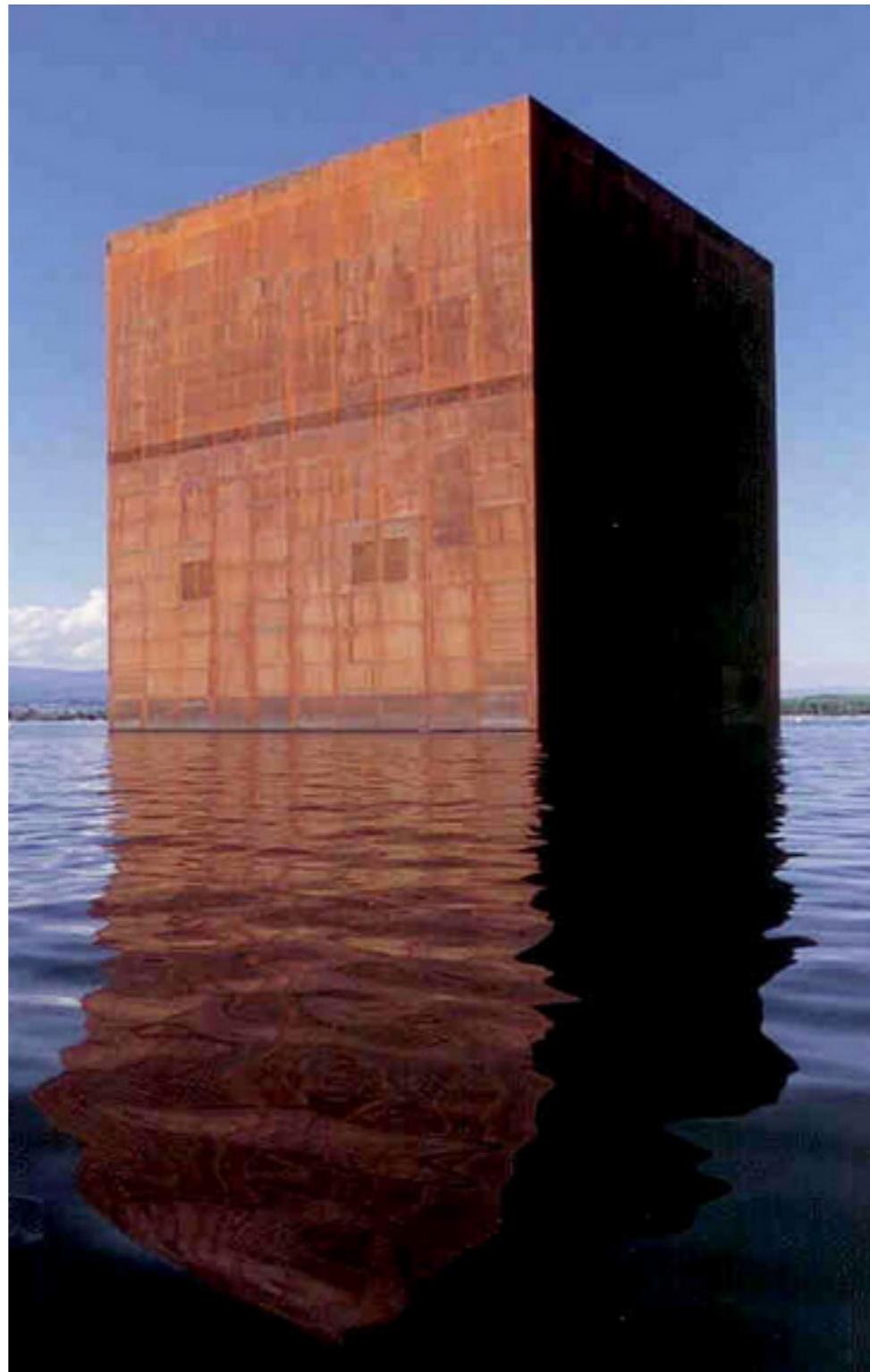
Decoupled



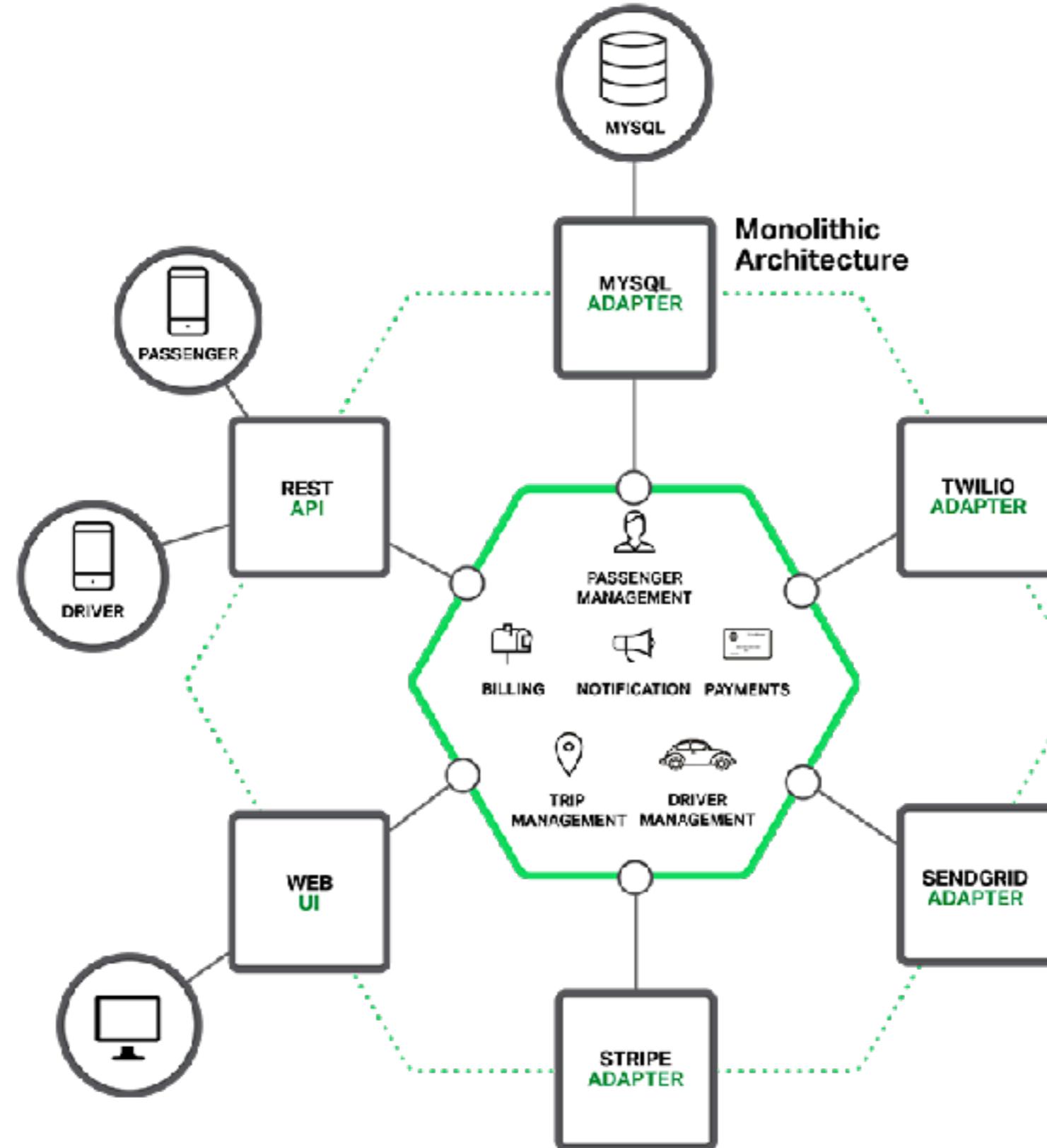
# Monolith



# Monolith



# Monolith



# Pros & Cons

?



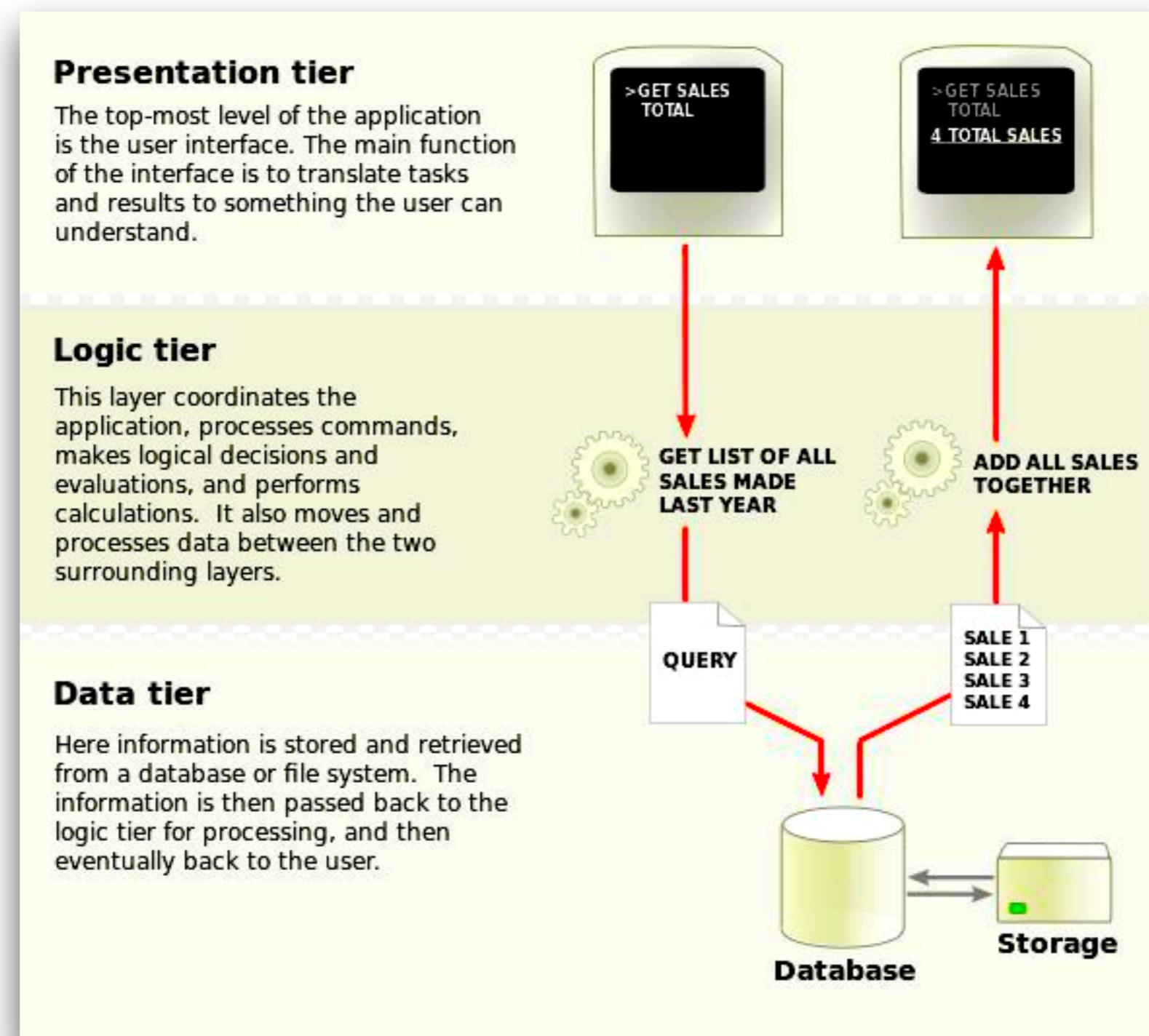
# N-tiers



# N-tiers



# N-tiers



# Pros & Cons

?

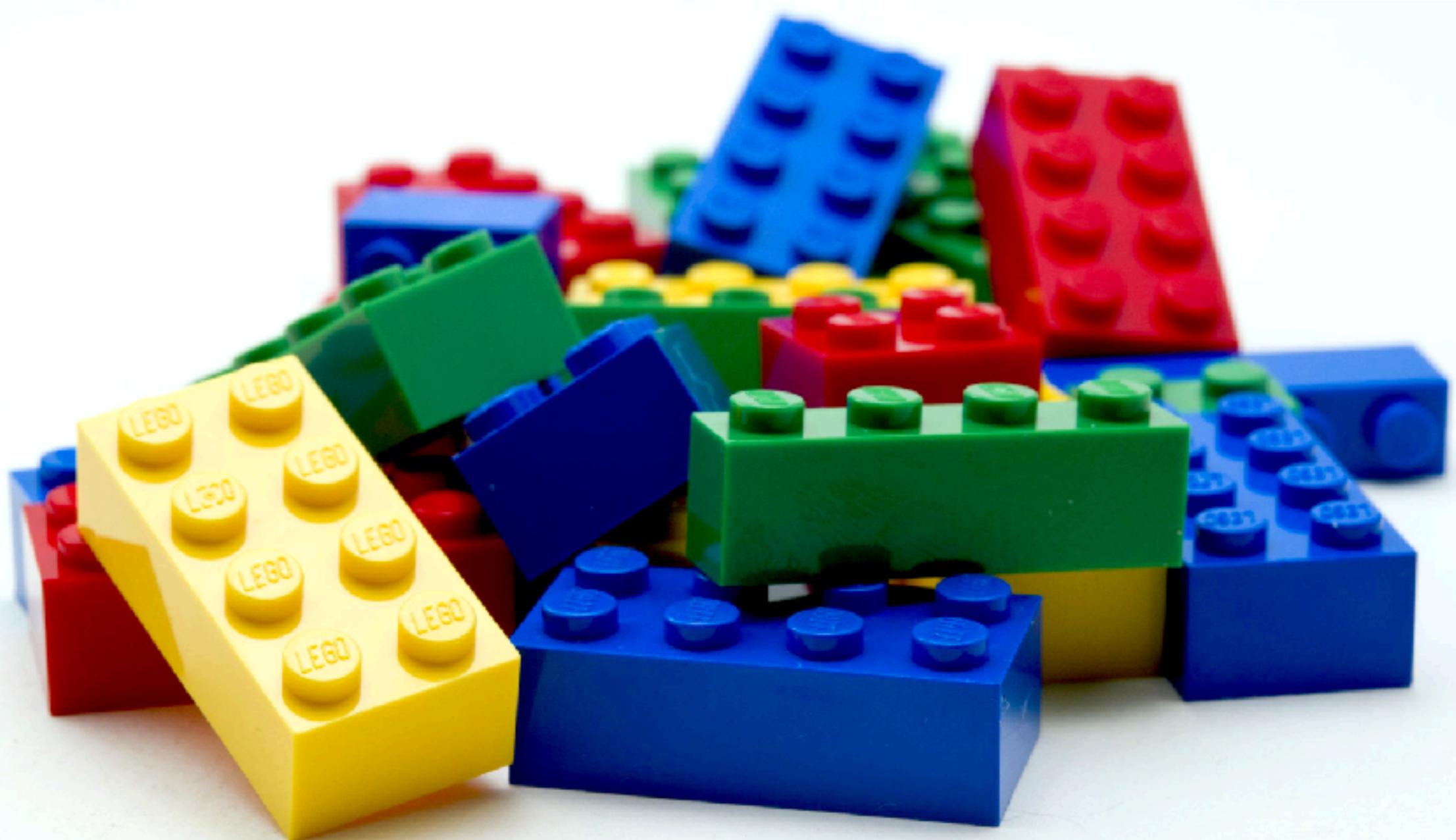


# **SOA**

# **Service Oriented Architecture**



# SOA

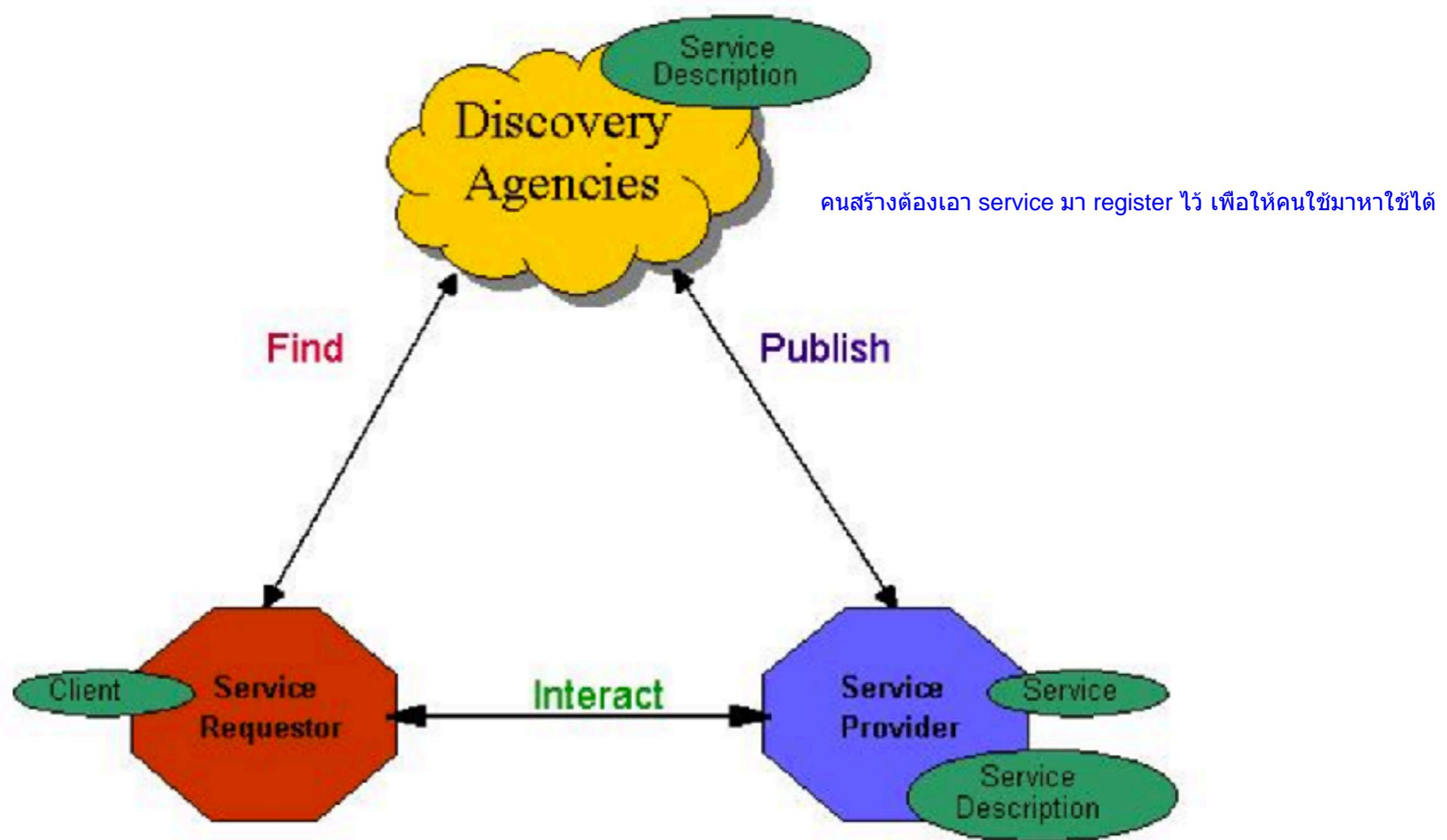


แต่ละ service เหมือน lego ชิ้นนึง แต่จะเอามาต่อ กันได้ ต้องมี standard

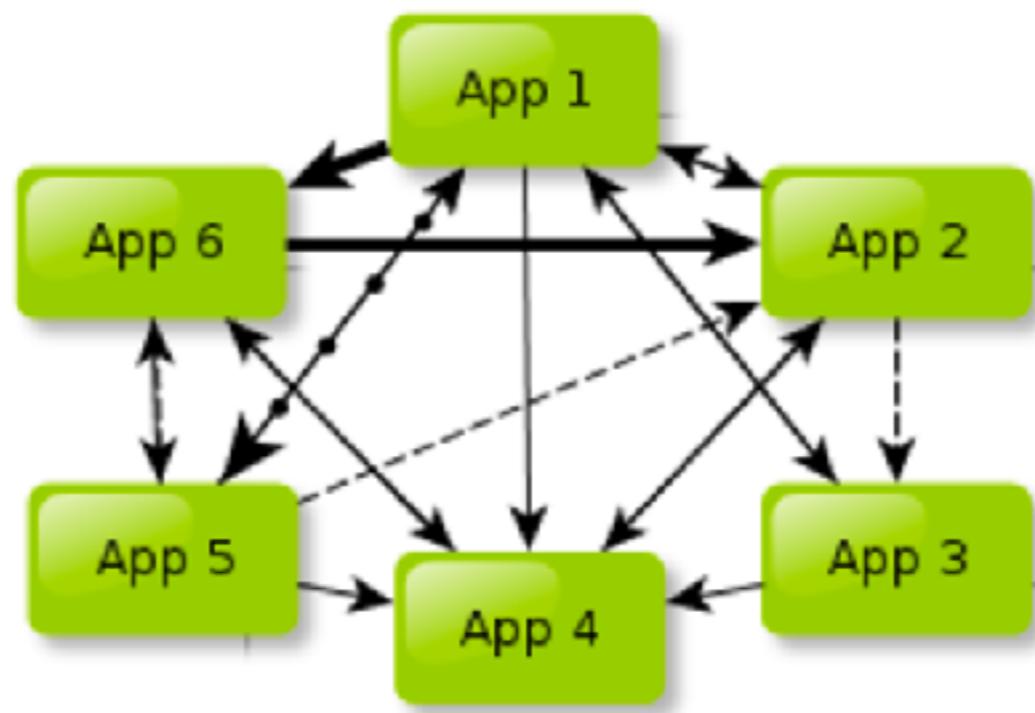
แต่มักมีปัญหาว่า standard ไม่ตรงกัน เพราะแต่ละเจ้ามักจะ extend จาก standard เลยมักมีปัญหาคุยกันไม่ได้



# SOA

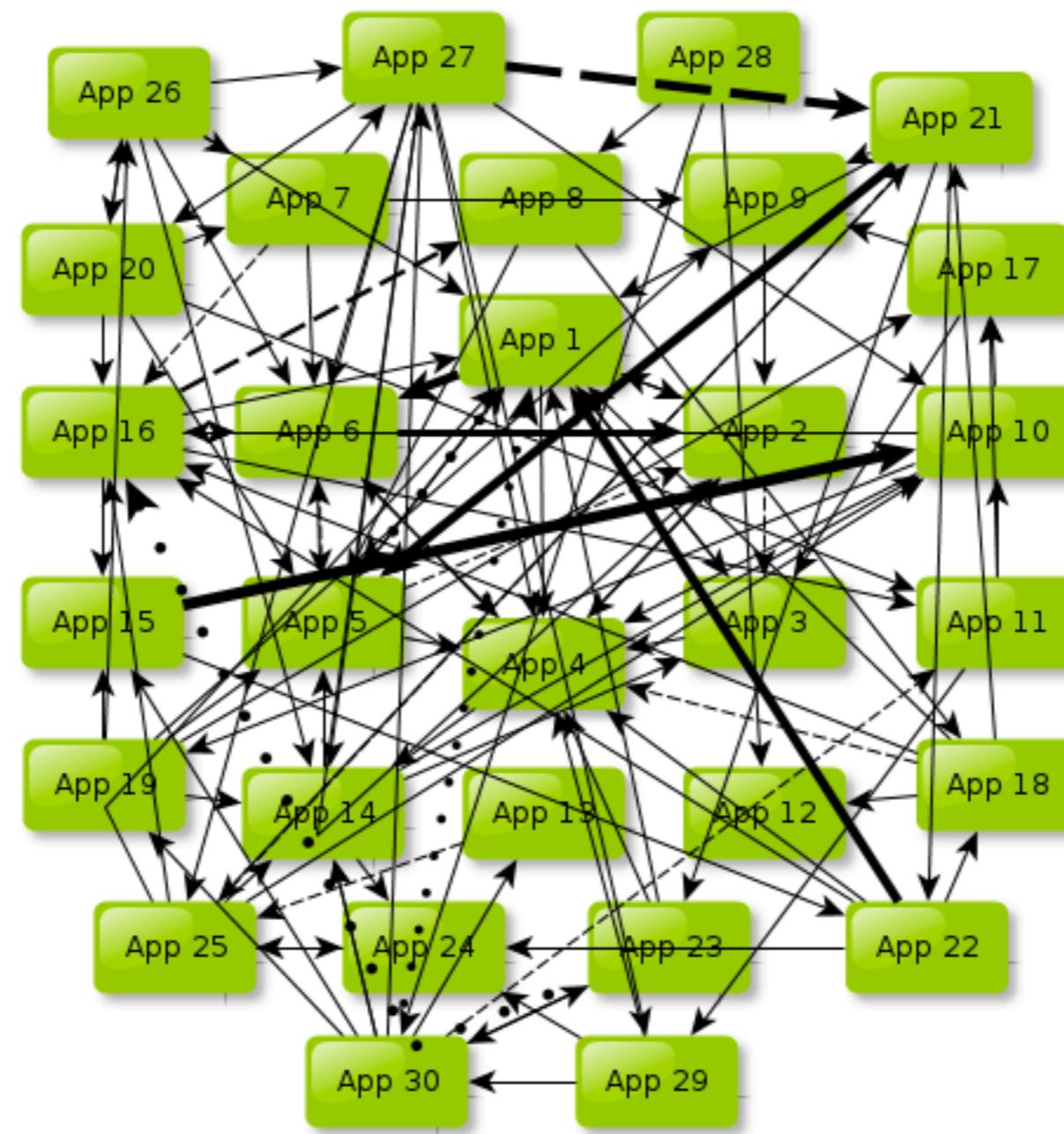


# SOA ?



หลักองค์กรมี SOA แบบนี้ แต่ไม่มีเครื่องสามารถ visualize ได้ว่า service มัน depend on กันยังไง หรือพ่อรู้ ก็พบว่ามี dependency/complexity สูงมาก ทำให้แก้ยาก หรือแก้ใช้เวลานาน

# SOA ?



# Solution ?



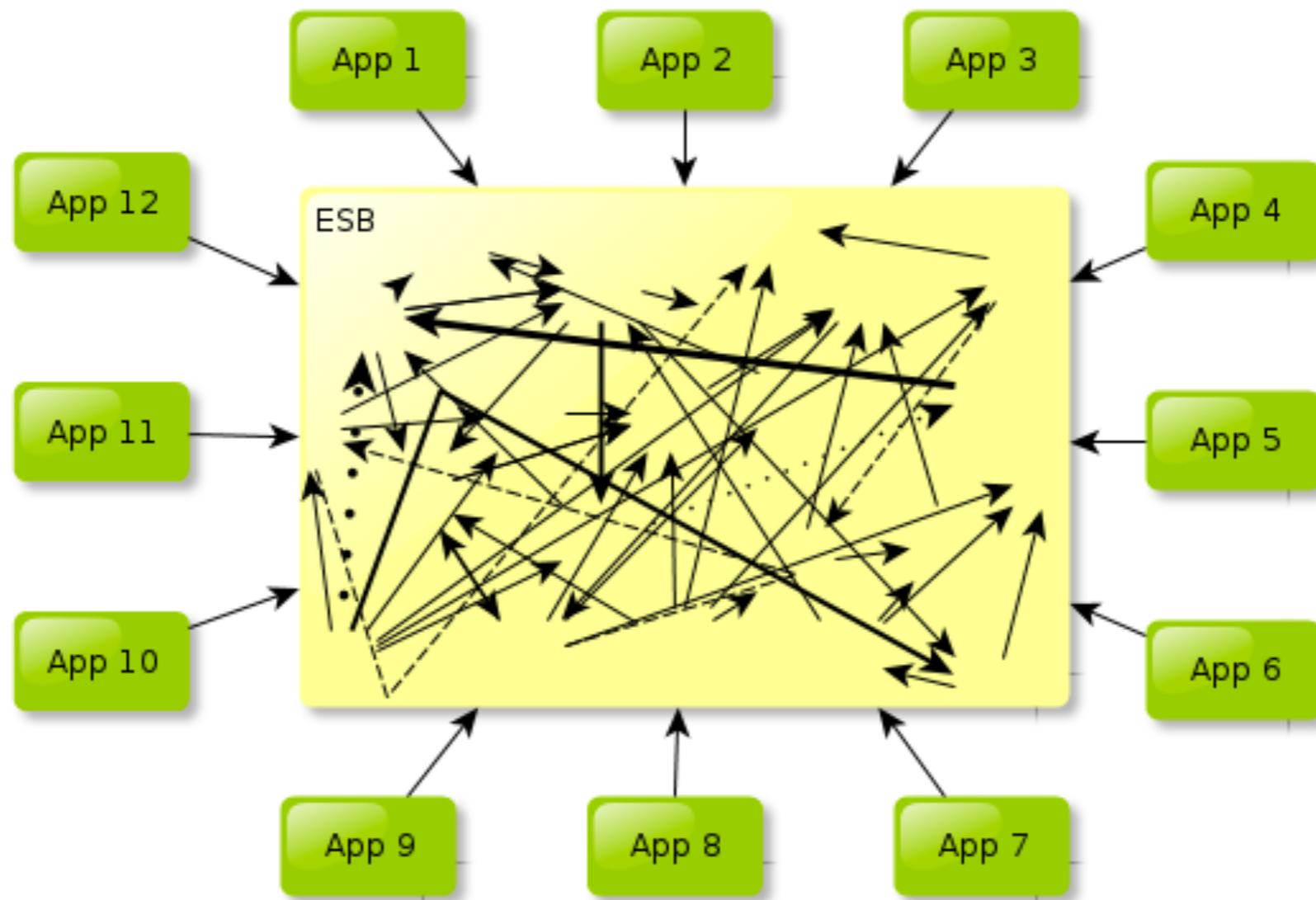
# SOA with ESB

ตอนหลัง มีการคิดค้น ESB เข้ามา ให้ทุก service ต้องมาใช้ผ่าน ESB ด้วยโนಡ์ pub/sub



# SOA with ESB

ในทางปฏิบัติ ESB ใหญ่มาก และทำให้ ESB กลายเป็น single point of failure  
แฉน ESB มักจะขึ้นกับ tool/vendor



# ESB

## Enterprise Service Bus

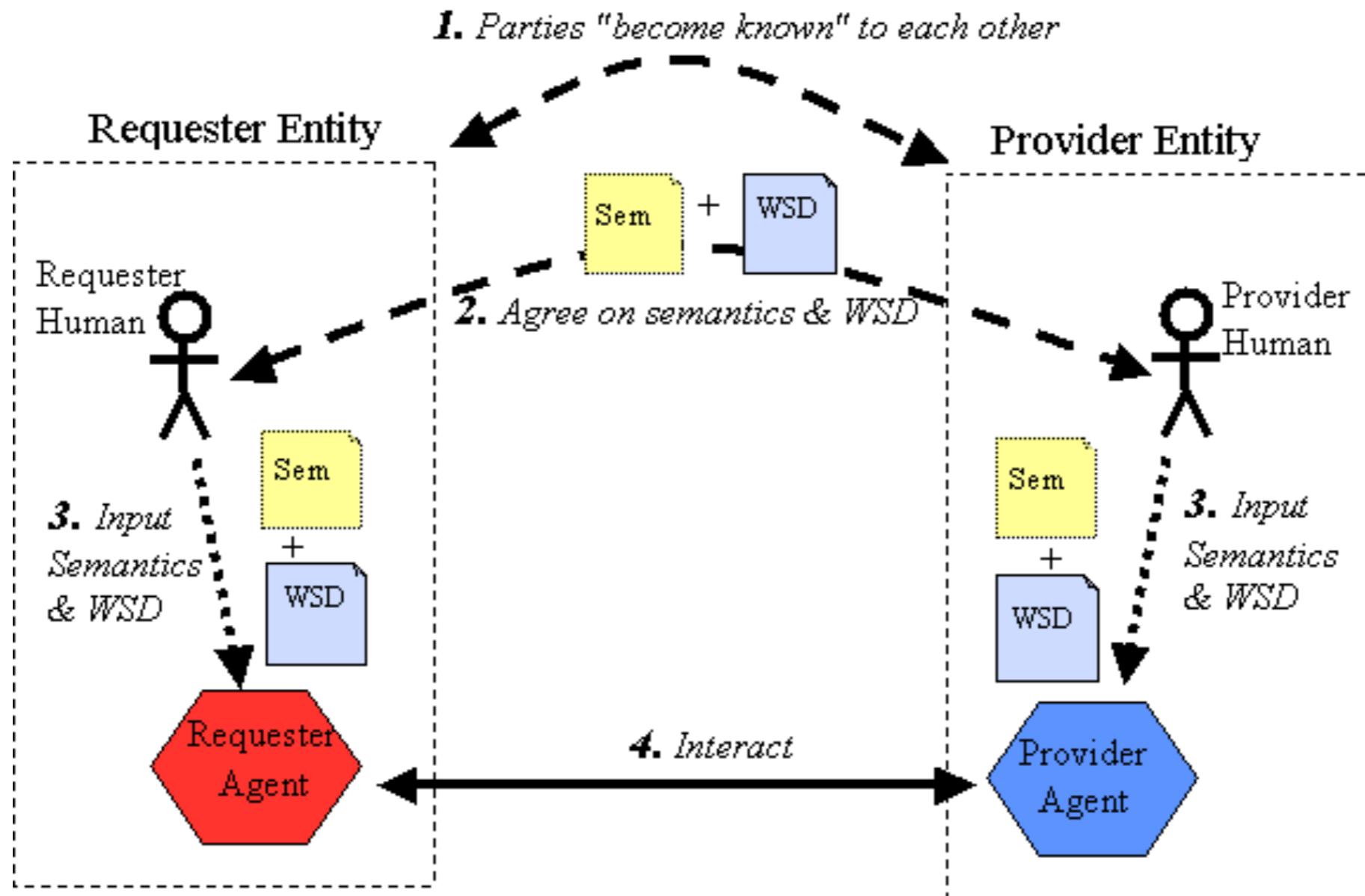


# WebService



บริษัท สยามชนาญกิจ จำกัด และเพื่อนพ้องน้องพี่

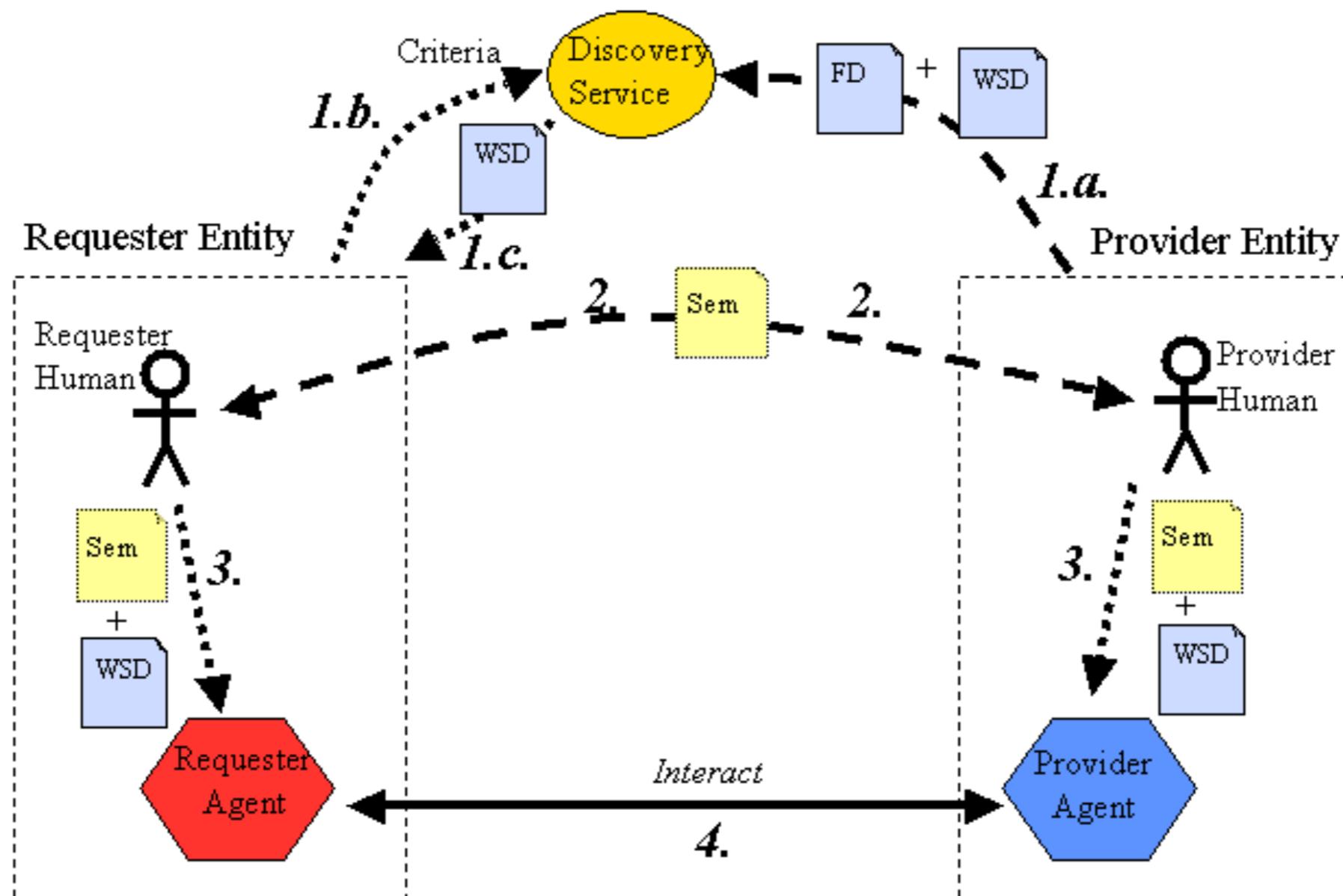
# WebService



<https://www.w3.org/TR/ws-arch>



# Service Discovery

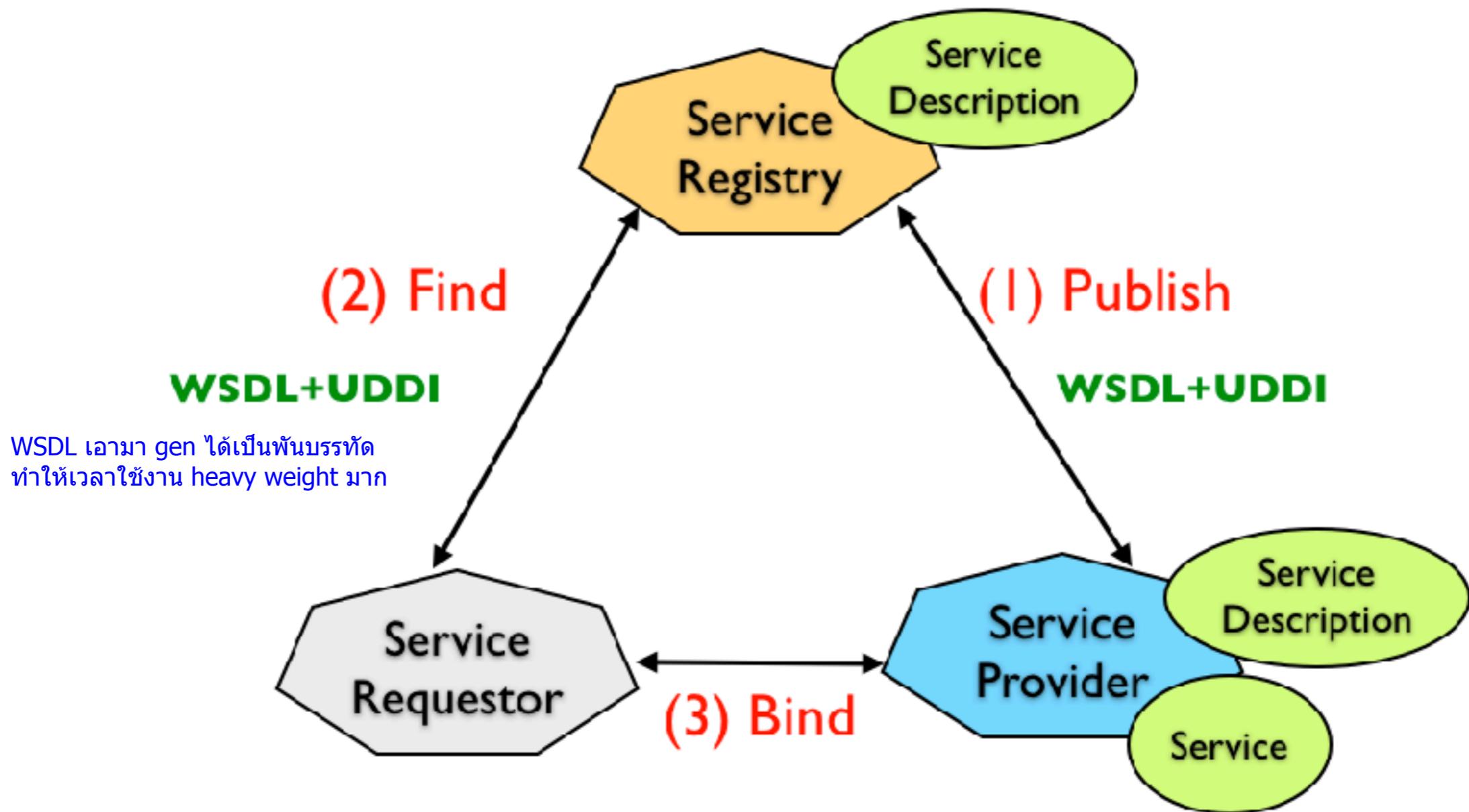


<https://www.w3.org/TR/ws-arch>



# WebService

Web Service เป็น implementation หนึ่งของ SOA



# WebService

WSDL  
UDDI  
SOAP



# WebService

## WebService Description Language Universal Descriptor Discovery Integration Simple Object Access Protocol

SOAP มันໂວເດ ແຕ່ SOAP with Attachments ມັນ...

ດ້ວຍໆຢ່າງ:

SMS = SOAP

MMS = SOAP + Attachment

ແຄ່ຈະສັງ HelloWorld ທ່ານໃຫ້ SOAP message ມັນໃຫຍ່ມາກ  
ທ່ານໃຫ້ server ມັນຮອງຮັບ load ໄດ້ໄນ່ເຍຂະ

ປັຈຸບັນ ບ.ໃໝ່ ຈະໄມ່ໃຊ້ SOAP/Web Services ໃນການ implement ເລຍ ມີແຕ່ໃນ Enterprise ໃຫຍ່າ

Enterprise ເລຍພາຍານປັບດັບມານ ບ. ໃໝ່



# Pros & Cons

?



# **REST API**

# **REpresentational State Transfer**

หมายองค์กรพยายามทำ RESTful แค่เปลี่ยนชื่อ แต่การออกแบบ การทำงานยังเหมือนเดิม

REST ตรงข้ามกับ SOA โดยสิ้นเชิง

REST เป็นของเก่ามาเล่าใหม่ แต่ design ต่างออกไป



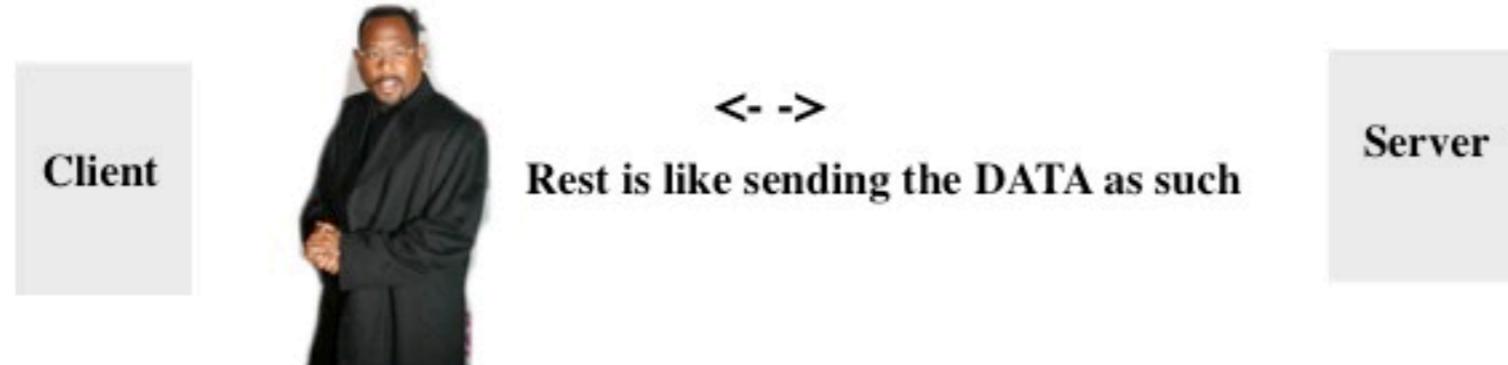
# REST

Consider "Martin Lawrence" as your data

## SOAP



## REST



# REST



Rides directly on HTTP. Plain and simple. In reality, this is all you need to send data from point A to point B and get the required response. Catch: Until something that represents a service contract is put in to place, it's kinda "anything goes".



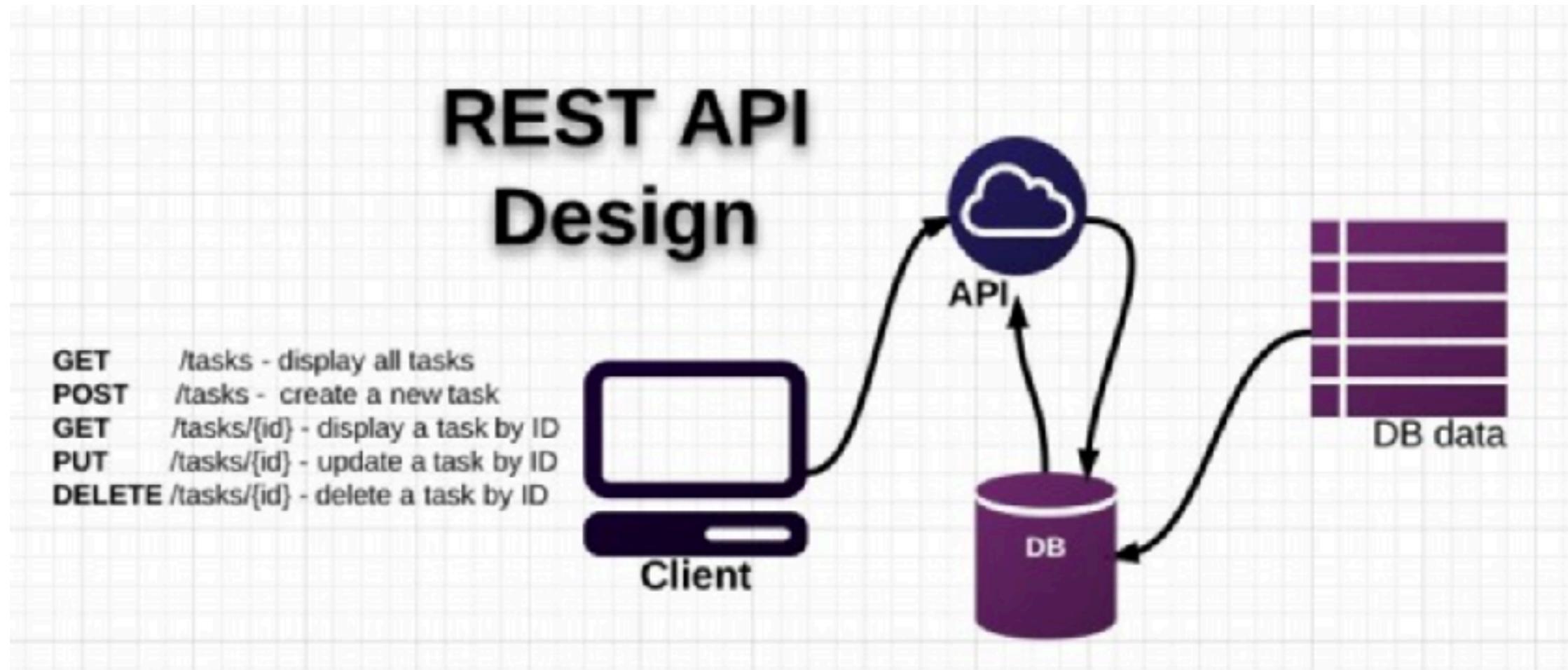
The coach is your SOAP envelope: it wraps your data. Main strength is the presence of a contract: the WSDL. Gives you the "comfort" of easily generating artifacts. Catch: look at the complexity and added weight.

ใน Web Service มักตั้งชื่อ function ดังนี้:

getData  
updateData  
deleteData  
เป็นลักษณะ RPC (Remote Procedure Call)

บาง service เป็น god service ทำทุกอย่าง ส่ง command ผ่าน parameter เข้าไป

# REST



REST คิดใหม่ มองว่าตัวเองมี resource ถ้า มี HTTP request เข้ามา

GET data คือการเรียกดูข้อมูลทั้งหมด

GET data/1 คือการเรียกดูข้อมูลตัวที่ 1

PUT data คือการสร้างข้อมูล

PUT data/1 คือการอัพเดตข้อมูลตัวที่ 1

DELETE data/1 คือการลบข้อมูลตัวที่ 1

1 endpoint ควบมีการทำงานบน resource ตัวเดียว

REST ต้อง stateless <> stateful  
stateless = ระบบไม่สนใจคุณเป็นใคร แต่คุณต้องมีตัว จึงจะเข้าถึง resource ได้  
stateful = server ต้องเก็บ session ของ client ทำให้ server ต้อง load หนักขึ้น

REST มี token ใช้แทนตัว เพื่อป้องกันการขโมยใช้ ตัวจึงมีอายุจำกัดเสมอ

หลังๆมี OAuth มาช่วยเรื่อง Authentication และ Authorization

1. Request Token รู้จักกัน
  2. Access Token ไว้ใช้กัน - มีอายุจำกัด
- ทำให้ security มั่นคงขึ้นมา



**Monolith**

ค่ายาเล็กลง

**N-tier**

**SOA/  
WebService**

**REST API**



# Microservice

หลายๆ คนออกแบบเป็น Microlith ซึ่งน่ากลัวมาก

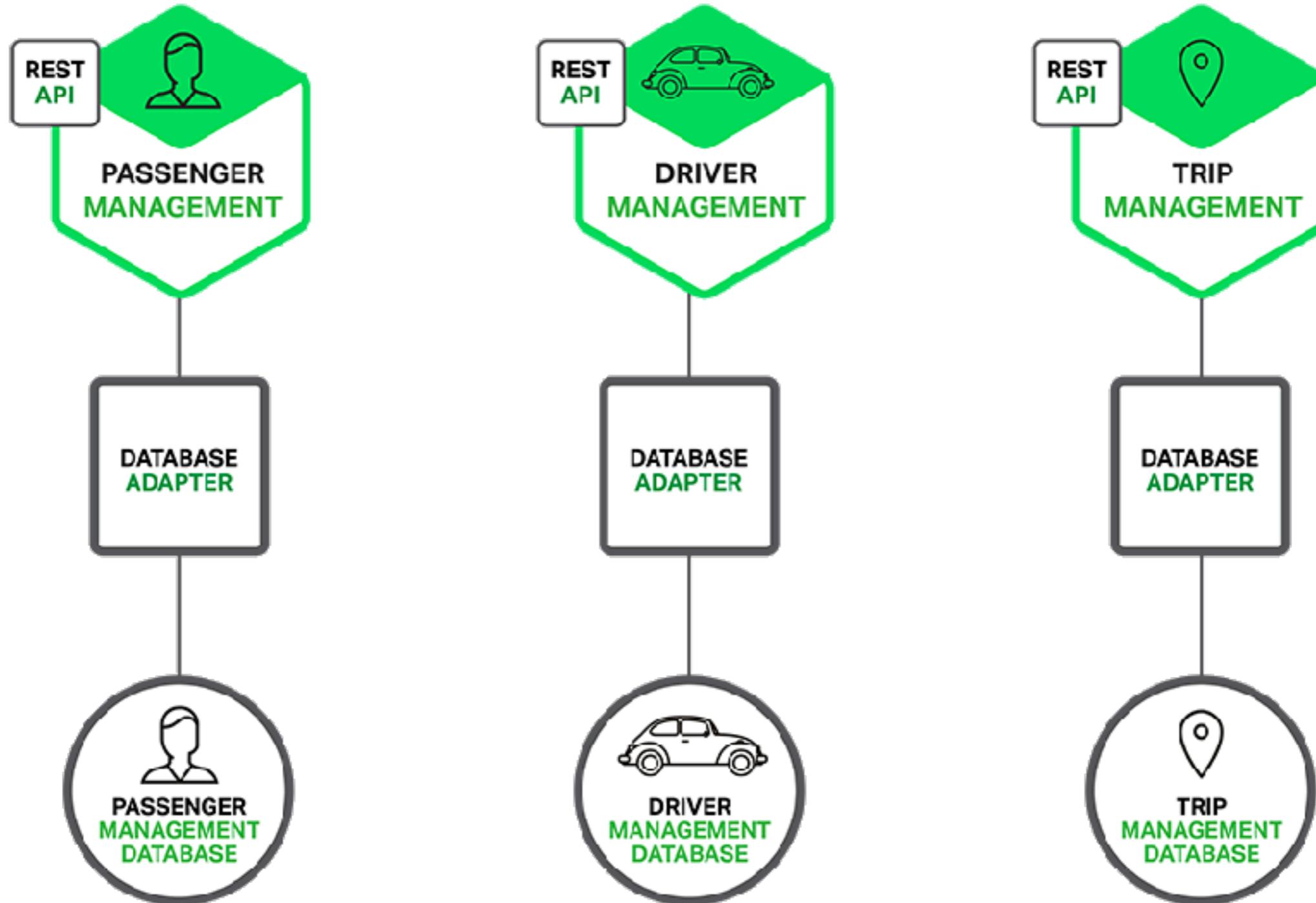


บริษัท สยามชนาญกิจ จำกัด และเพื่อนพ้องน้องพี่

# Microservice

แต่ละ service มีหลาย API ได้ (ใช้ REST API)

1 service จัดการจบในตัวเอง = ถ้า service พังก็พังแค่ตัวเดียว คือต้อง isolate ทั้งในระดับ physical และ logical  
ซึ่งทำให้ต้องการ server เยอะขึ้นมาก ทำให้ cloud เข้ามามีบทบาท

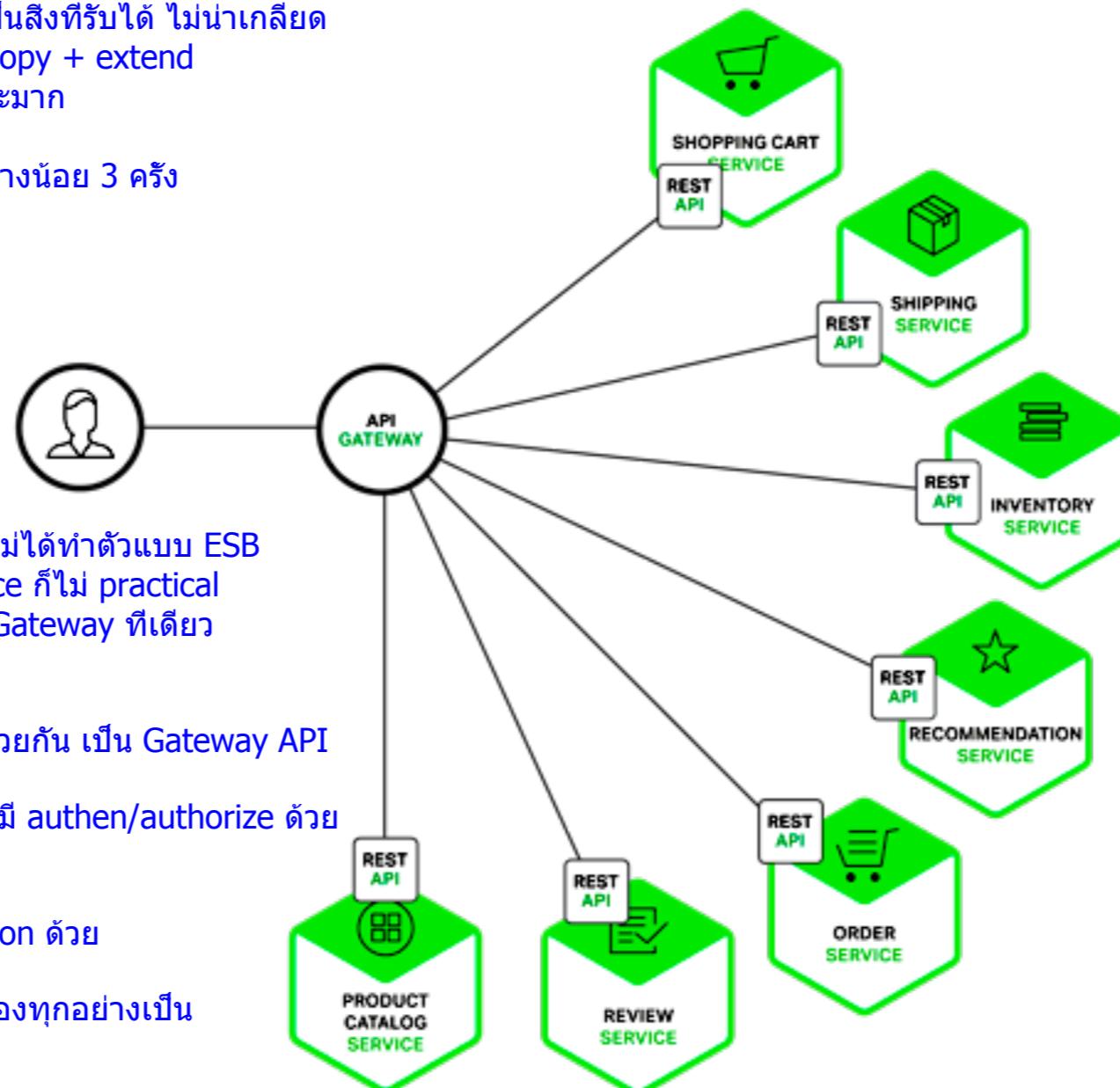


# Microservice

การแตก service ละเอียดเกินไป ทำให้ client ต้อง call API หลายครั้ง ทำให้ response time แย่ เป็นการออกแบบที่ไม่ดี architect มักบวกกวนกว่าดี เพราะมี reusability แต่พ่อวิเคราะห์ดูแล้วกลับมักไม่รู้ว่ามีใครใช้งาน ทำให้แก้ไขลำบาก

ใน Microservice มองว่า duplication เป็นสิ่งที่รับได้ ไม่น่าเกลียด  
ถ้า feature 2 ต้องการ reuse ก็อาจจะ copy + extend  
ไม่ใช่ทำเป็น library ที่มี if...else... เยอะมาก

การ reuse ควรทำเมื่อมี duplication อย่างน้อย 3 ครั้ง



ESB กลับมาในรูปของ API Gateway แต่ไม่ได้ทำตัวแบบ ESB  
การจะใส่ authen/authorize ในทุก service ก็ไม่ practical  
เลยเอา authen/authorize มาใส่ใน API Gateway ที่เดียว

และ client ก็เรียกใช้จากที่เดียว  
นอกจากรายงานที่ยังสามารถช่วยร้อย service เข้าด้วยกัน เป็น Gateway API

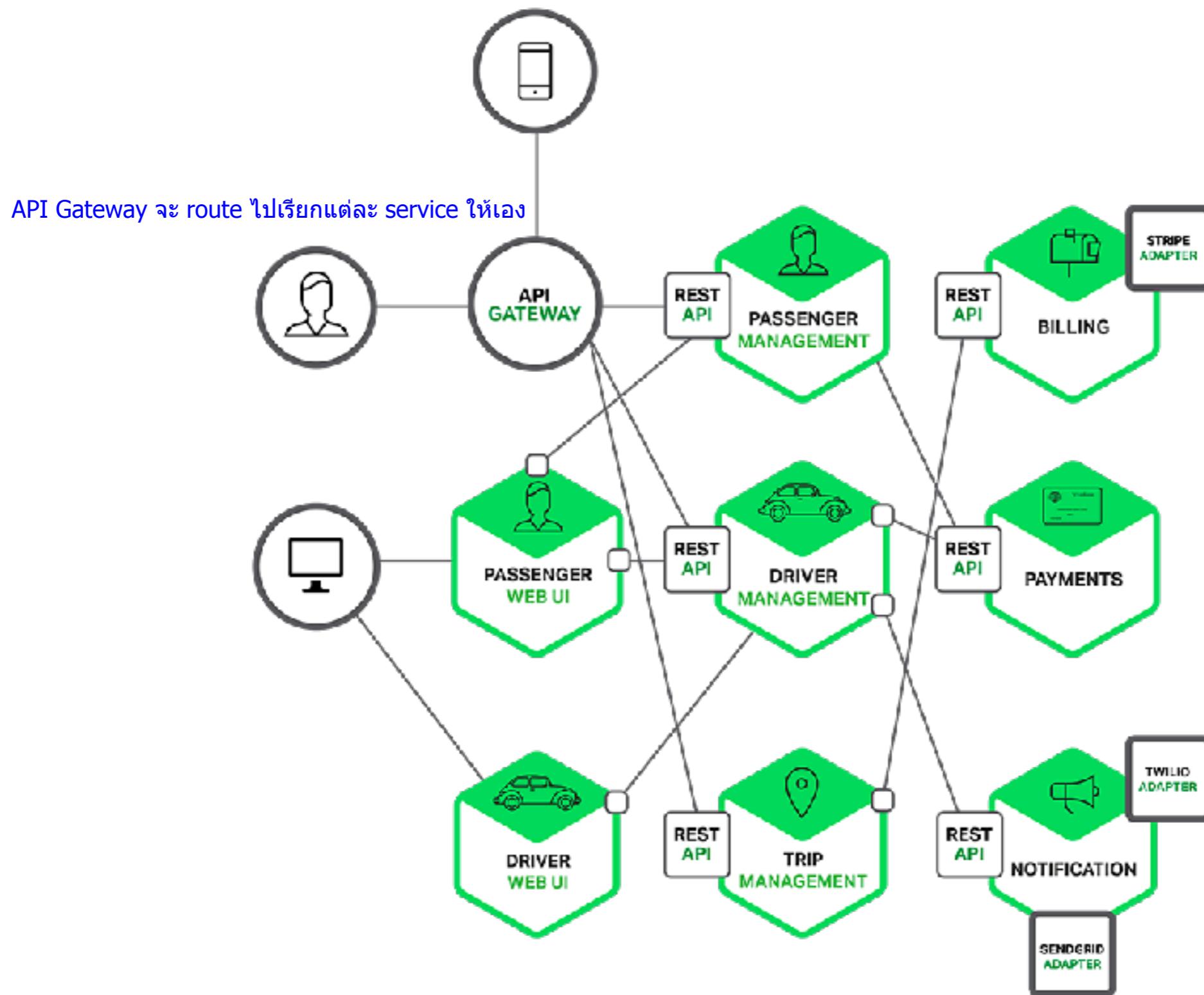
ส่วน private services ข้างหลัง ก็ควรต้องมี authen/authorize ด้วย  
แต่อาจไม่ต้องหนักเท่าที่ตัว gateway

ถ้า sensitive data ก็อาจต้องทำ encryption ด้วย

Don't trust your private network! ให้มองทุกอย่างเป็น  
public



# Microservice



# Pros & Cons

Microservices = Design for Failure (Learning) เพราะมันเล็ก ถ้าพังมันกลับง่าย ถ้ากลับยากแสดงว่าออกแบบผิด  
ไม่ต้องกลัวผิด เพราะผิดแน่นอน เจ็บแน่นอน แต่ต้องรู้ให้เร็ว (รู้ก่อนคนใช้ vs รู้จากคนใช้)

บางที่ยุบ service ดีเกินไป ทำให้บาง service โดมาก มีหลาย API มีการใช้งานเยอะ แต่มักมี API ที่ใช้งานเฉพาะฯ อยู่อันเดียว  
ถ้าจะ scale เราก็อยากร scale แค่ API นั้นๆ ไม่ใช่ทั้งหมด ทำให้แยก API ตัวนันออกมารูปเป็น service ใหม่

...ต้อง redesign ใหม่ไปเรื่อยๆ เพื่อไม่ให้กลัวการ redesign ก็ต้องมี test มี monitoring ด้วย



tradeoff Coupling vs. Cohesion --> Pros/Cons

Microservices หากมี 10 choices of design ก็ໄล์ไปเลยทีละ choice ขอให้รู้เร็ว จะเร็วได้ scope ต้องเลือก

การจัด Team ก็แบ่งตาม service ใน team ก็ cross functional ลดการ communication ลง  
ลดความเป็นเจ้าของลง และเป็นการกระจาย knowledge ในองค์กร

Microservices ต้องเลือกขนาดไหน?

> rewrite in 1 day --> ทำให้คนไม่กลัวที่จะ design ใหม่ สร้างใหม่

แต่ละ service ทำงานกันผ่าน REST ทำให้แต่ละ service สร้างด้วยอะไรก็ได้  
ภาษาต่างๆมักถูกสร้างขึ้นมาเพื่อแก้ปัญหาเฉพาะทาง ทำให้สามารถลองได้

ใน Microservices จะใช้ DB แยกกัน ทำให้มี data duplication กันได้ เป็นของคู่กันกับ Microservices  
สามารถ share DB กันได้ แต่ต้องแยกกันให้ดี ระวังเรื่อง single point of failure

Deployment lead time ควรให้สั้นๆ เพื่อให้เรียนรู้เร็วขึ้น เลยมีการเอา container เข้ามาใช้ เพื่อให้ provisioning/configuration mgmt. ง่ายขึ้น  
และทำให้ automation ได้

Monitoring ในระดับ App/services เพื่อดู success/failure rate สถานะของ service เพื่อเตรียมการ scale  
log เป็นสิ่งที่ดี แต่จะดีกว่าถ้านำมา visualize ได้ (เช่น splunk, kafana)  
log เป็น time series data เมนะจะเก็บด้วย Prometheus, ElasticSearch, etc. ไม่เหมาะกับ RDB

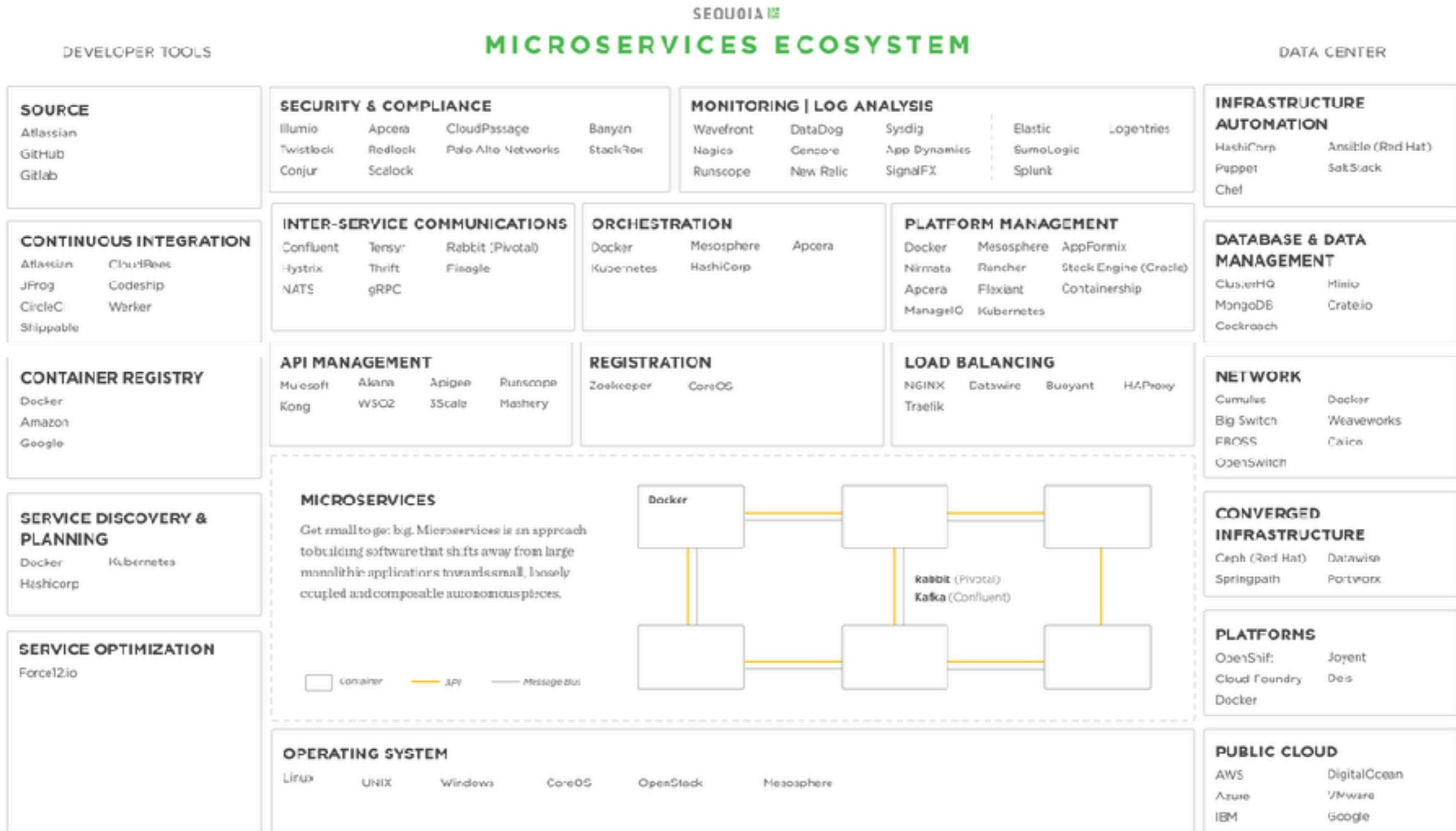
Alert and Notifications, Security (e.g. OWASP) ควรทำ security req. ตั้งแต่ต้น อย่าทำตอนท้าย

Testing เป็นเรื่องที่ໂหนดร้ายมาก เพราะแก้ service นึง จะกระทบ service อื่นๆ ด้วย

<http://12factor.net>  
เป็นพื้นฐานการออกแบบ  
ควรต้องตอบโจทย์ 12 ข้อ



# Microservice Ecosystem



Includes both companies and open source projects.

Version 1.2 | GI.79.76

<https://twitter.com/mcmiller00/status/690894999370633216>



DDD = Domain Driven Design ช่วยแยก service ออกจากกันมี system boundary

Asynchronous vs. Synchronous ก็สำคัญ

# Serverless



# FaaS

# Function as a Service



# Let's workshop

