NoSQL

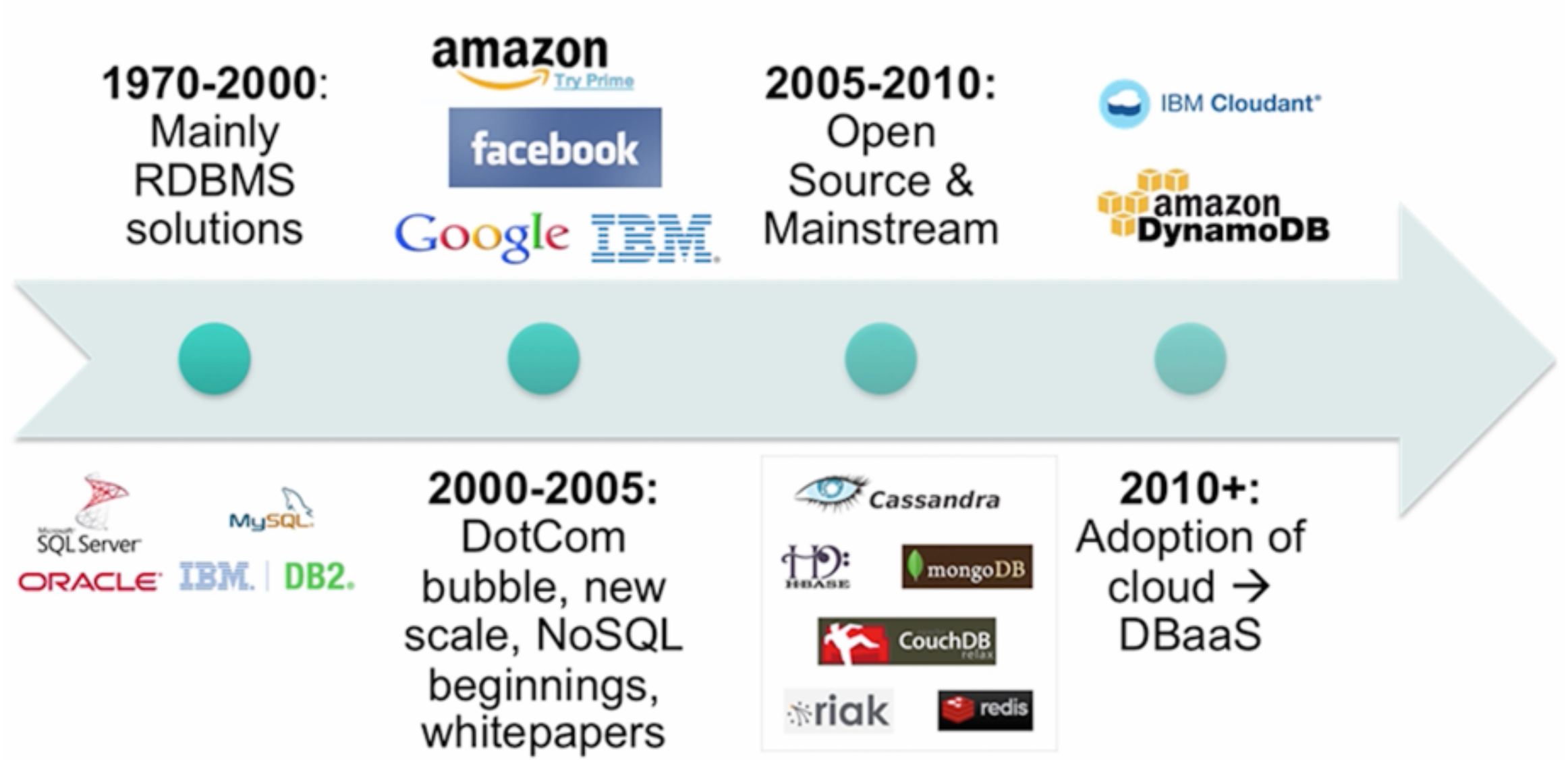
ภารุจ รัตนวรพันธุ์

ภาควิชาวิศวกรรมคอมพิวเตอร์ ม. เกษตรศาสตร์

NoSQL

- ในช่วงแรก NoSQL == No SQL
- ในปัจจุบัน NoSQL == Not only SQL
- ชื่อมาจาก Twitter’s Hashtag ของ Johan Oskarsson

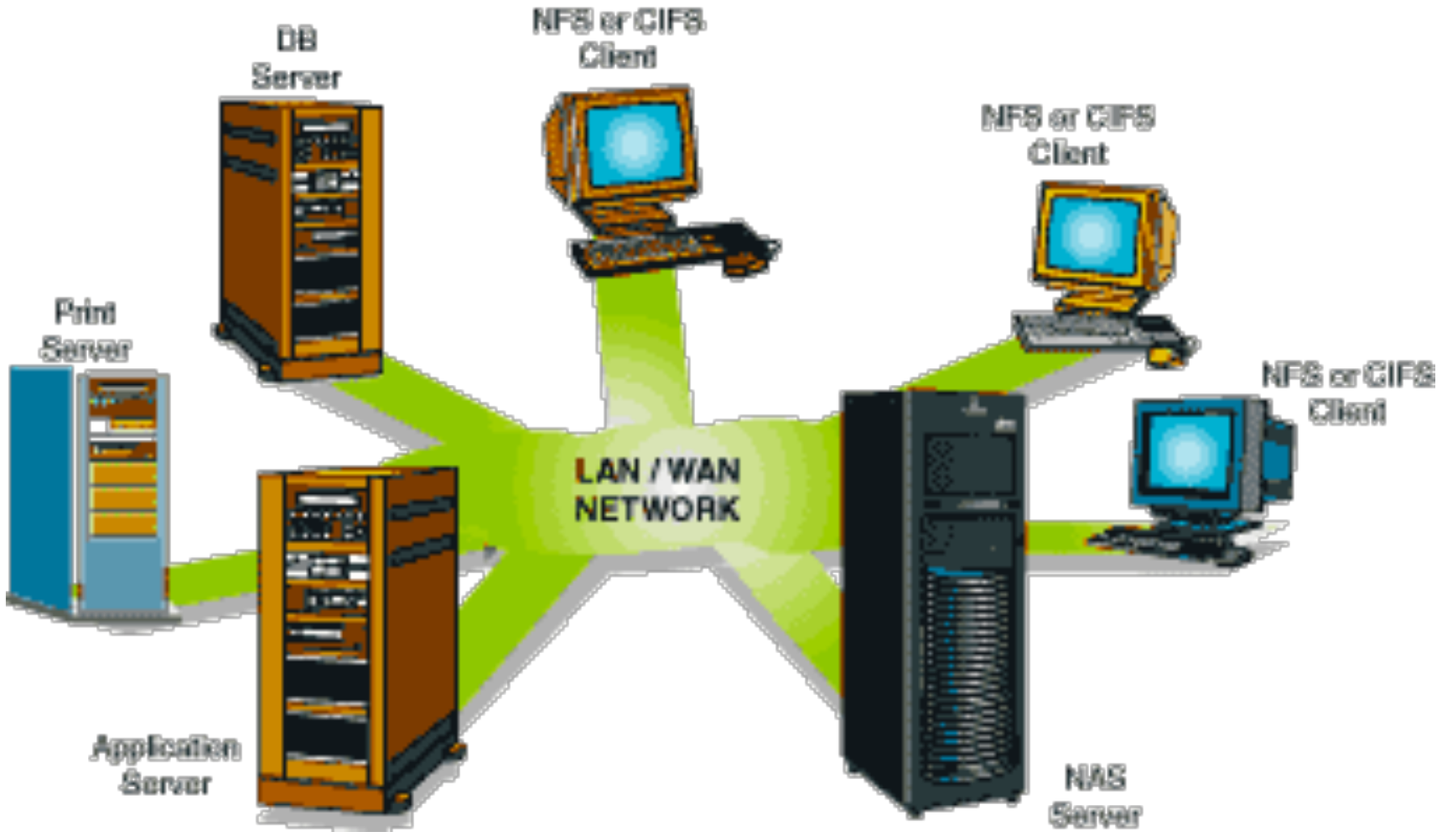
ความเป็นมาของ NoSQL



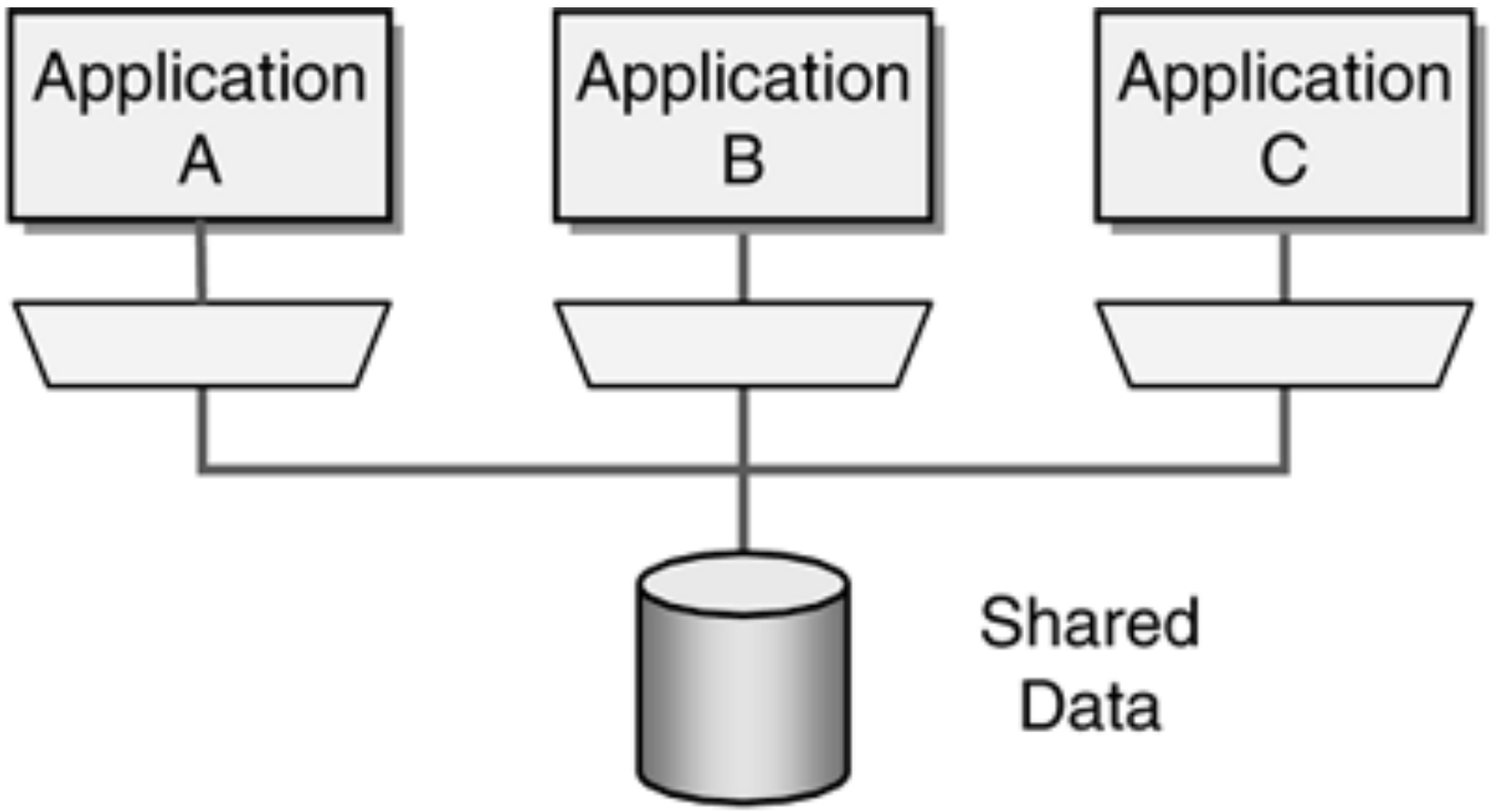
ความโดดเด่นของ relational database

- ACID transactions
- SQL ภาษาที่ทุกคนรู้จัก เชื่อมต่อกับภาษาโปรแกรมอื่นได้หลากหลาย
- สามารถทำ index ได้
- สามารถสร้างโมเดลข้อมูลที่รองรับได้ทุกงานการคำนวณ
  - Integration database

สถาปัตยกรรมของ enterprise computing



Integration Database

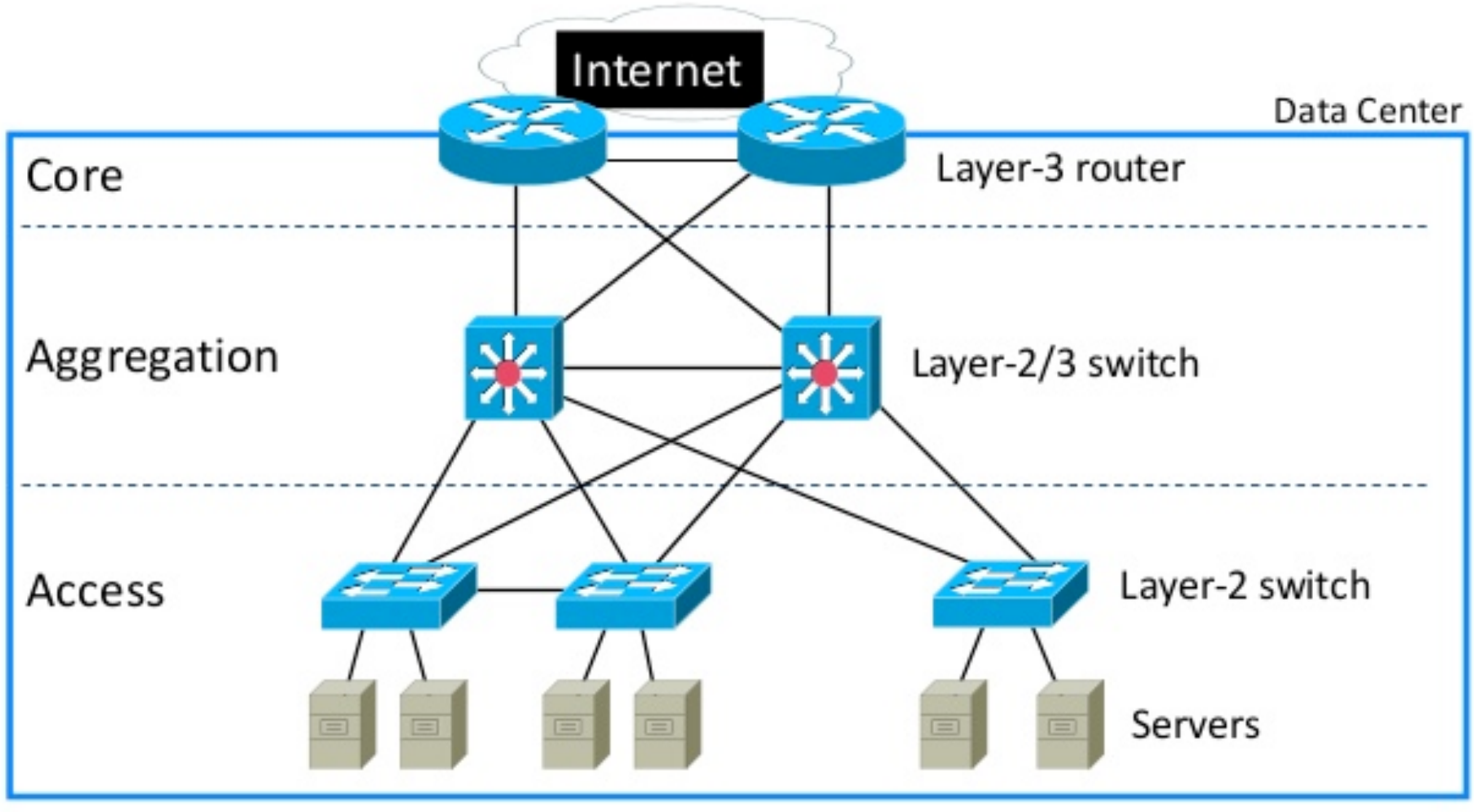


จาก SQL สู่ NoSQL

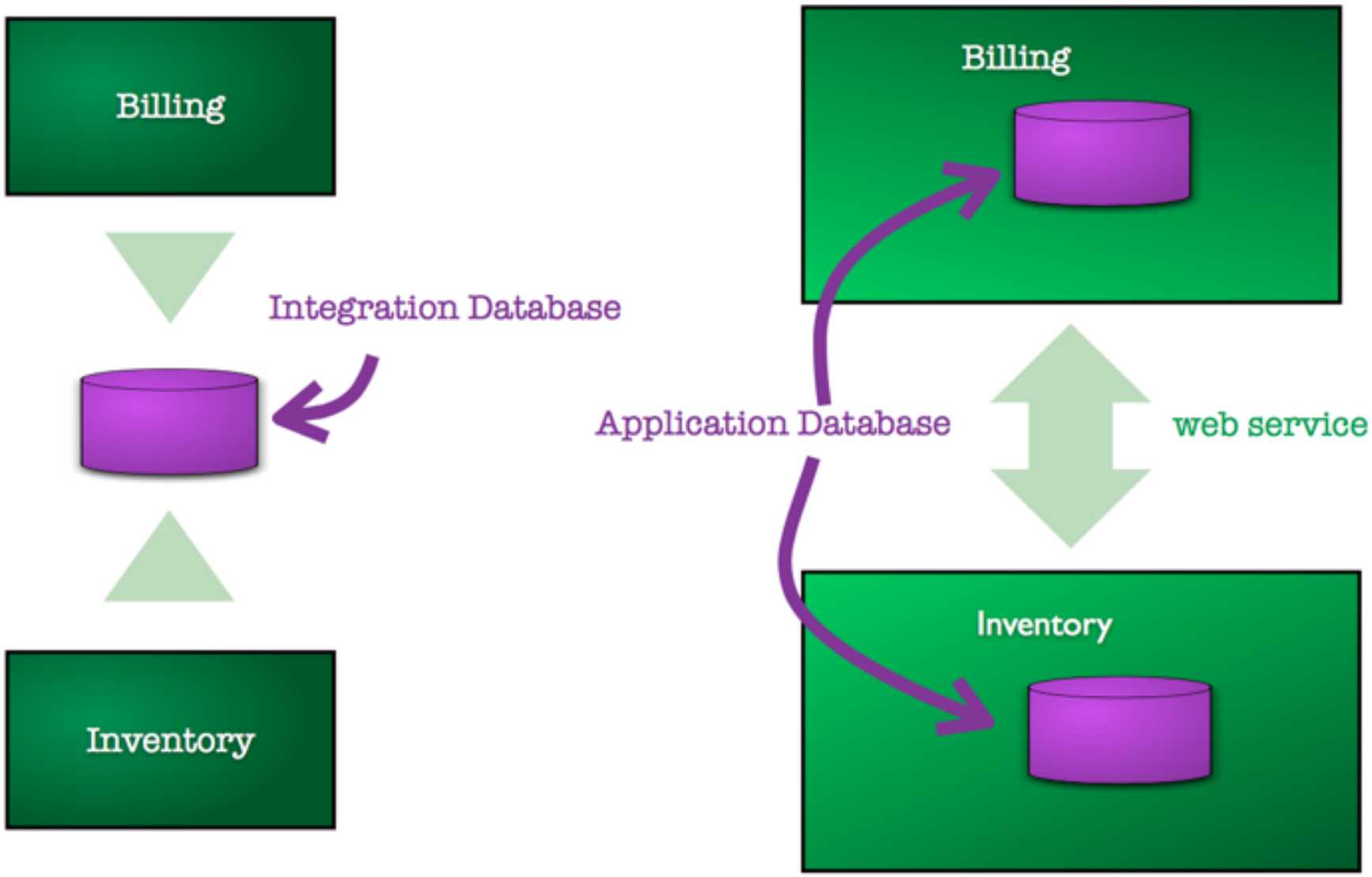
- ต้องการประมวลผลข้อมูลจำนวนมาก (big data) และรองรับผู้ใช้งานจำนวนมาก
  - Scale out ฐานข้อมูลต้องถูกกระจายออกไปอยู่ในหลายๆเซอเวอร์
- ต้องการความยืดหยุ่น ไม่ยึดติด schema (schema-less)
- ต้องการเพิ่มสมรรถนะของระบบ
- ปัจจัยสองอย่างหลังใช้ object database แก้ได้ ปัจจัยแรกเป็นตัวสำคัญสำหรับการปฏิวัติสู่ NoSQL

Scale-Up เทียบกับ Scale-Out

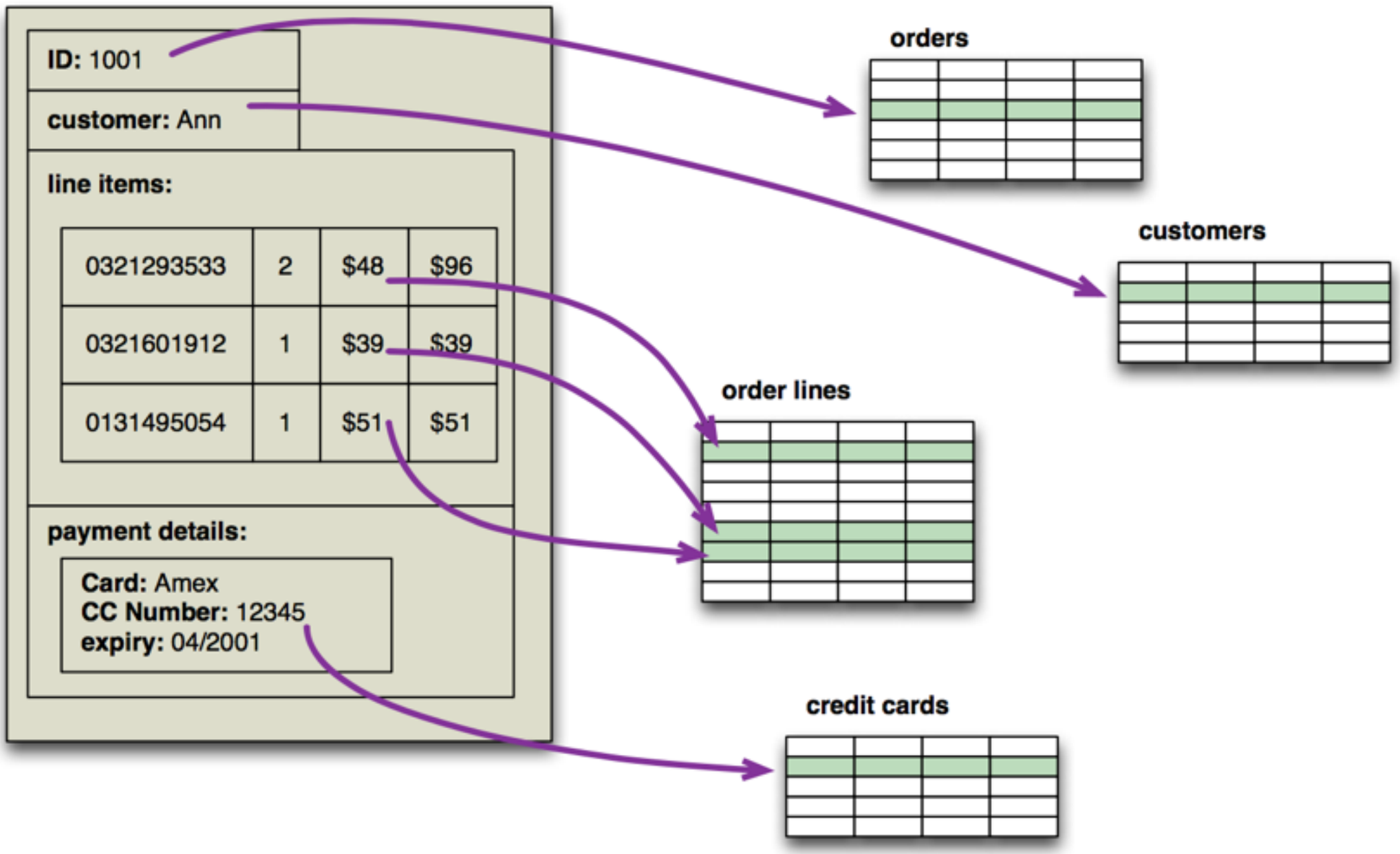
สถาปัตยกรรมสำหรับ data center



การใช้ฐานข้อมูลรวมกับการแยกฐาน



Impedance mismatch ในโมเดลแบบ relational

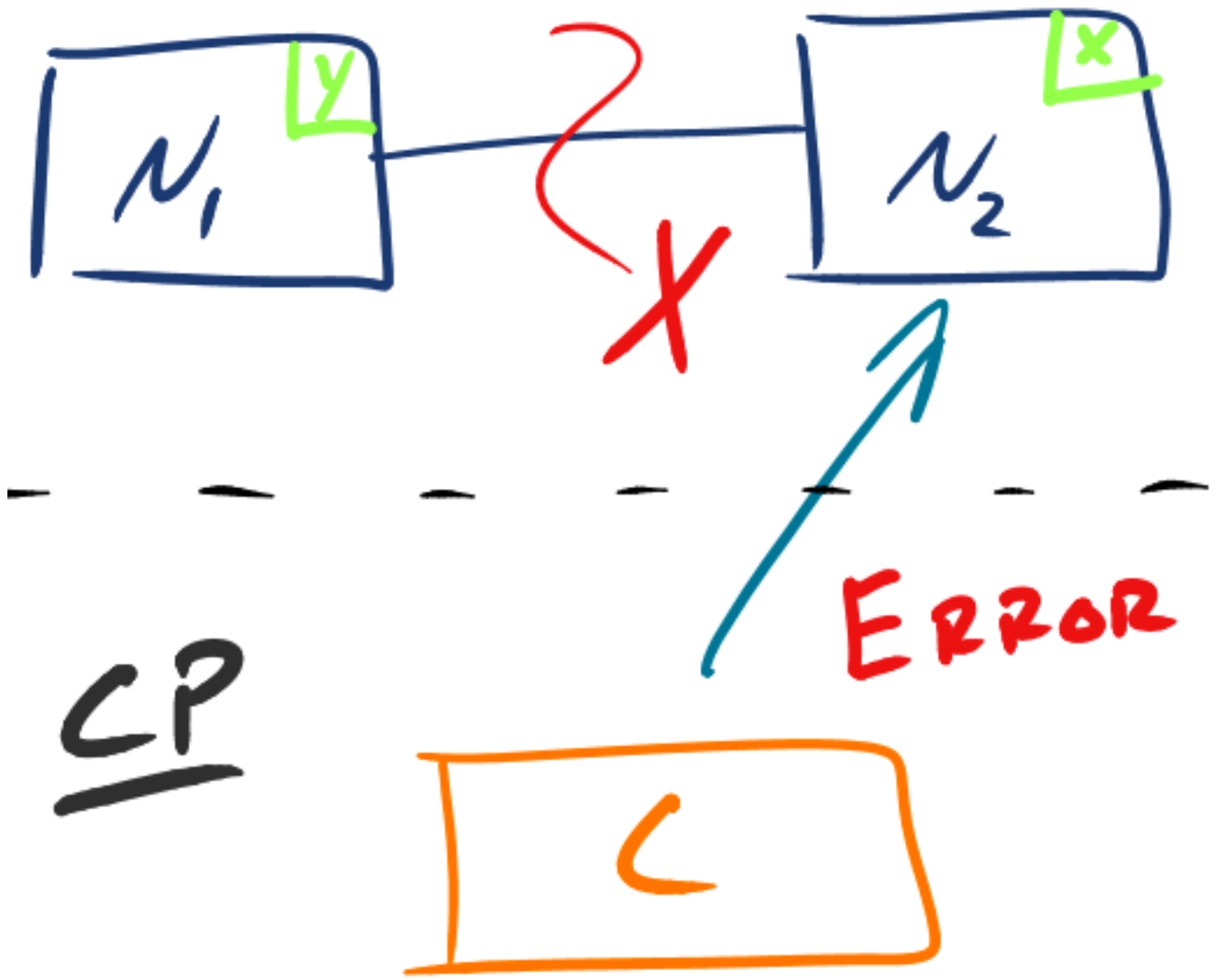


ทฤษฎี (ข้อสังเกต) CAP

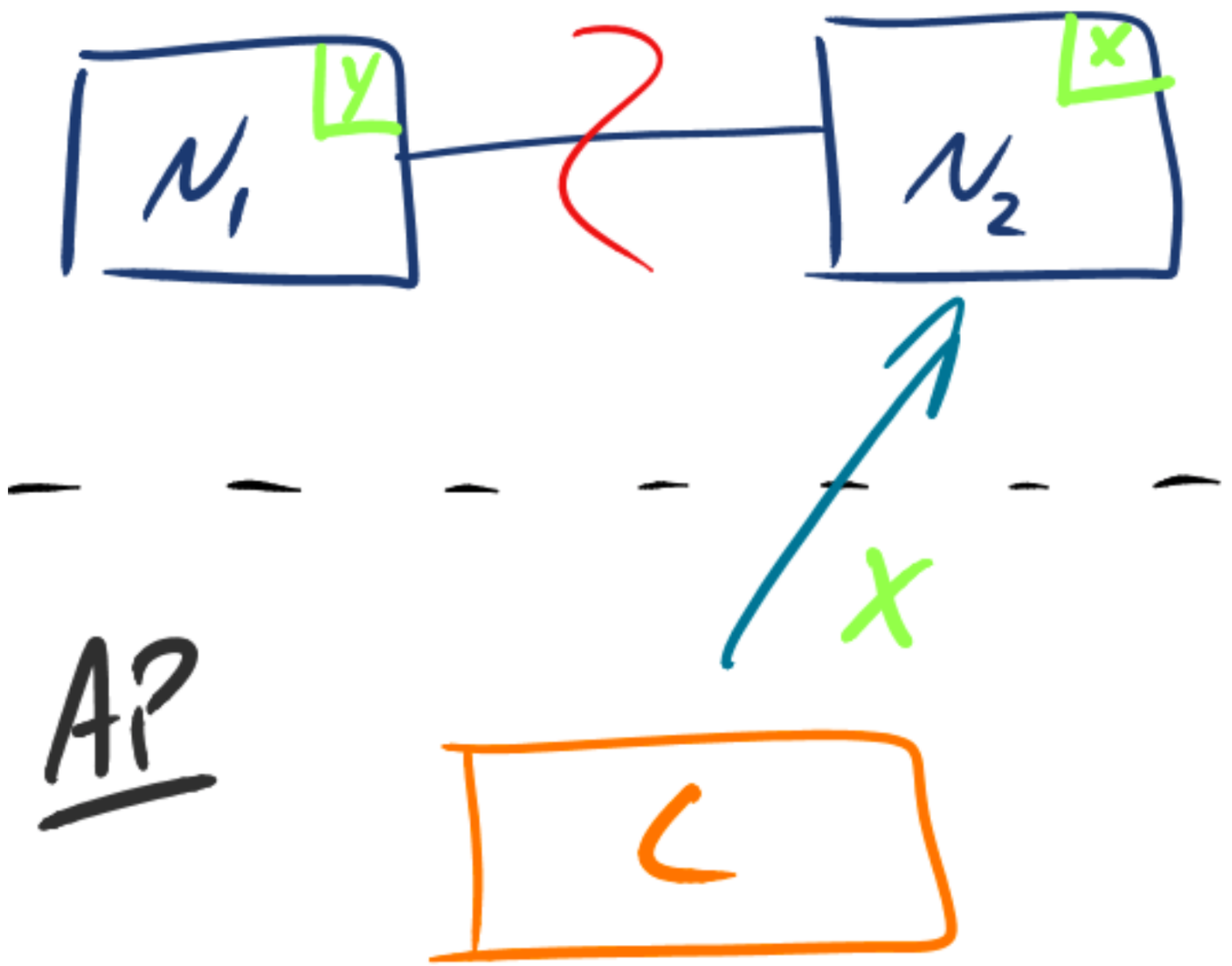
เป็นไปไม่ได้ที่ระบบคำนวณแบบกระจายจะมีคุณสมบัติครบถ้วนทั้งสามอย่างต่อไปนี้

- Consistency (ทุกโหนดมีมุมมองของข้อมูลแบบเดียวกัน)
- Availability (ทุกๆคำร้องได้รับการตอบสนองภายในระยะเวลาหนึ่ง)
- Partition tolerance (ระบบทำงานต่อไปได้แม้ว่าโหนดบางส่วนจะถูกตัดขาด)

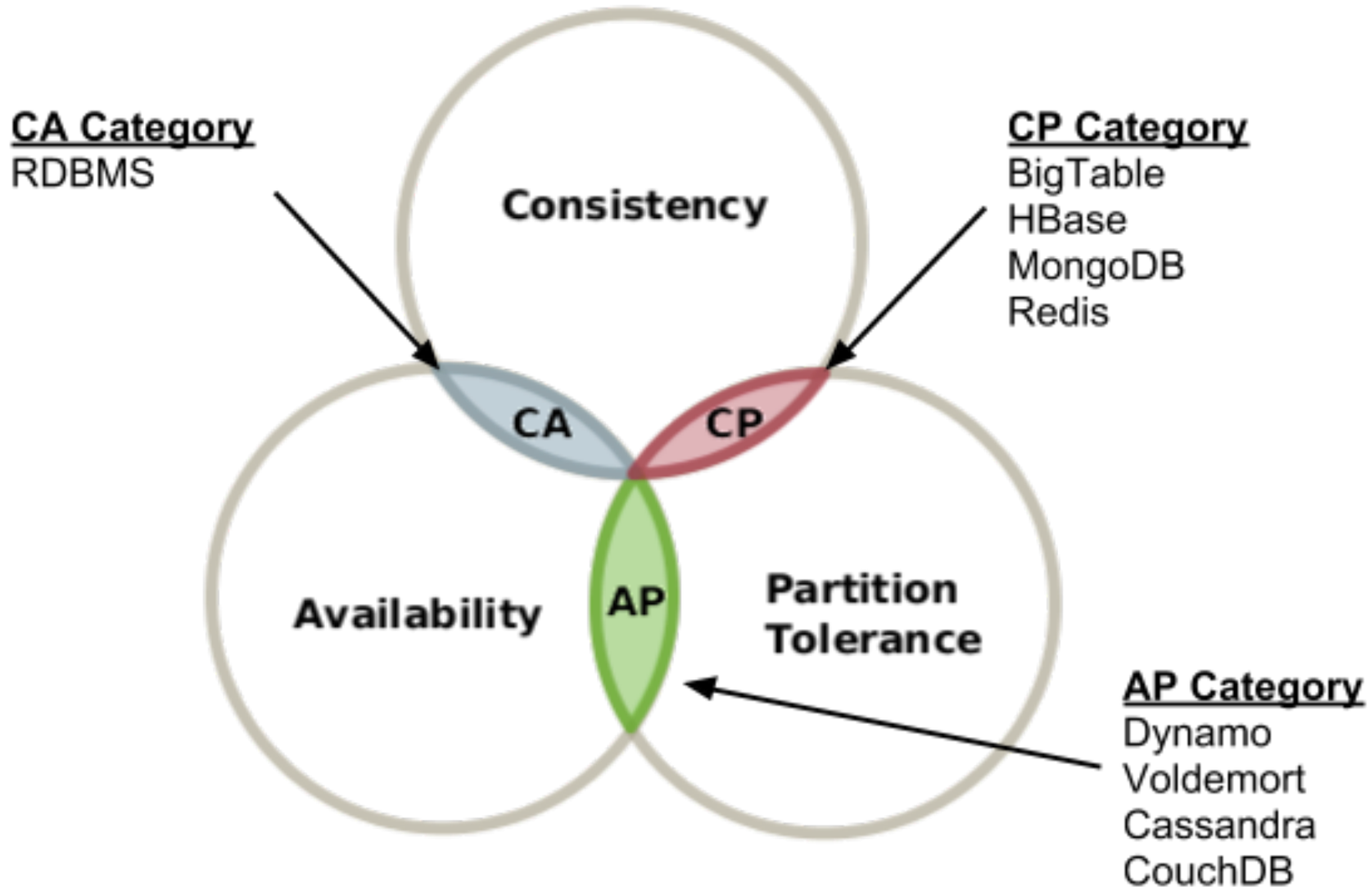
S:UU CP



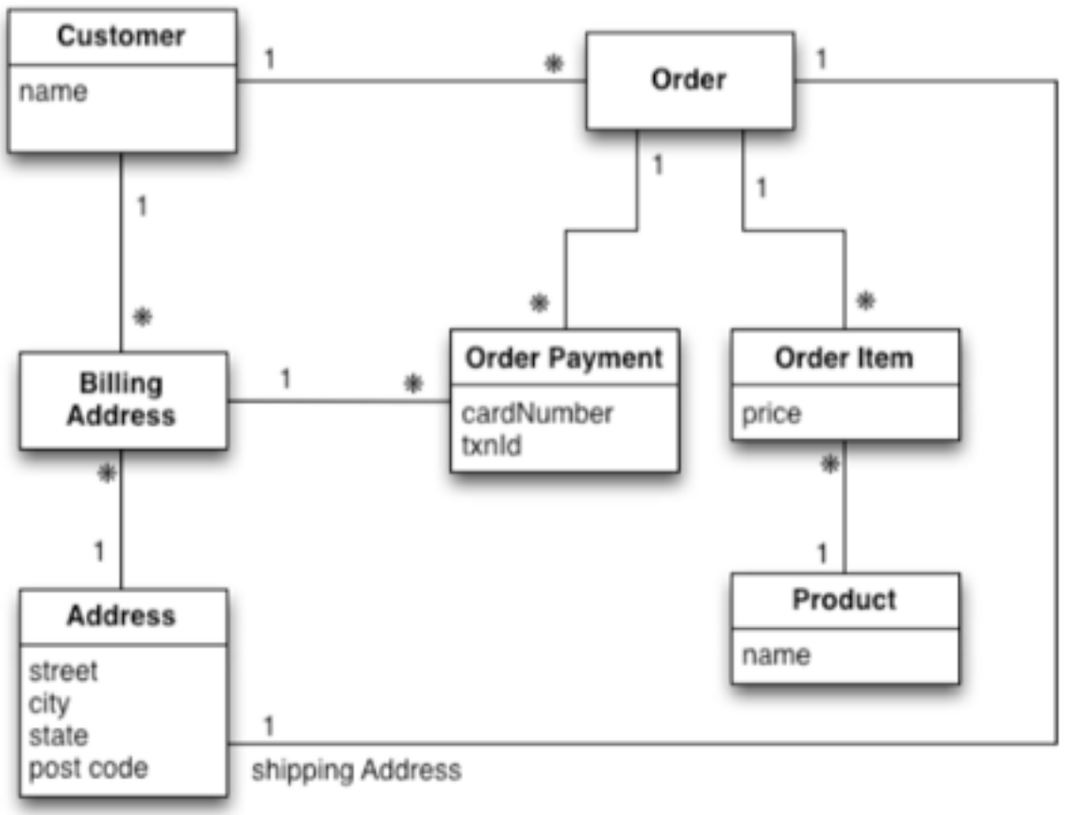
S:UU AP



แผนภาพ Venn ของ CAP



แนวคิดการรวมข้อมูล (Aggregate Data)



Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

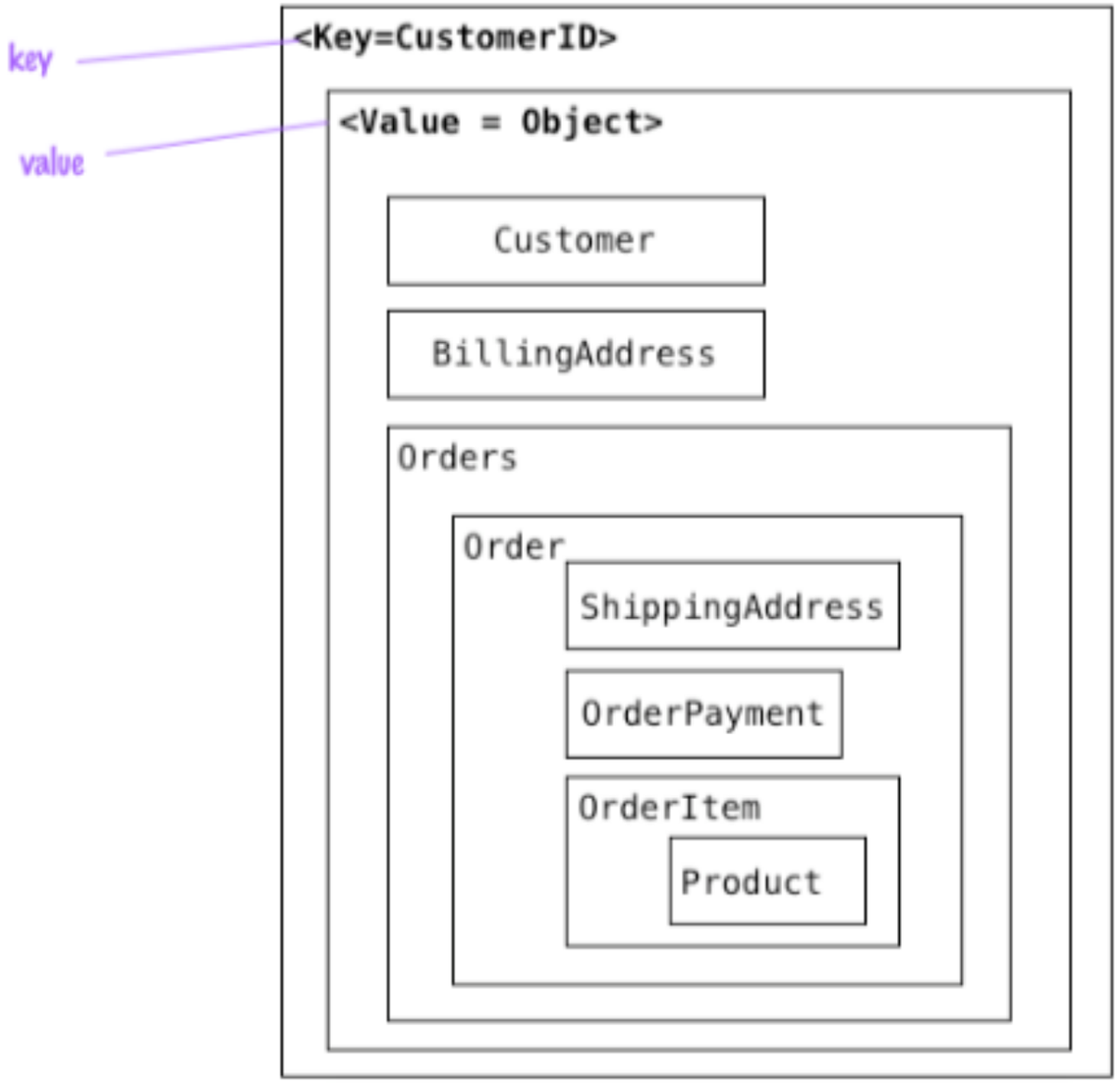
Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

จากระบบ RDBMS

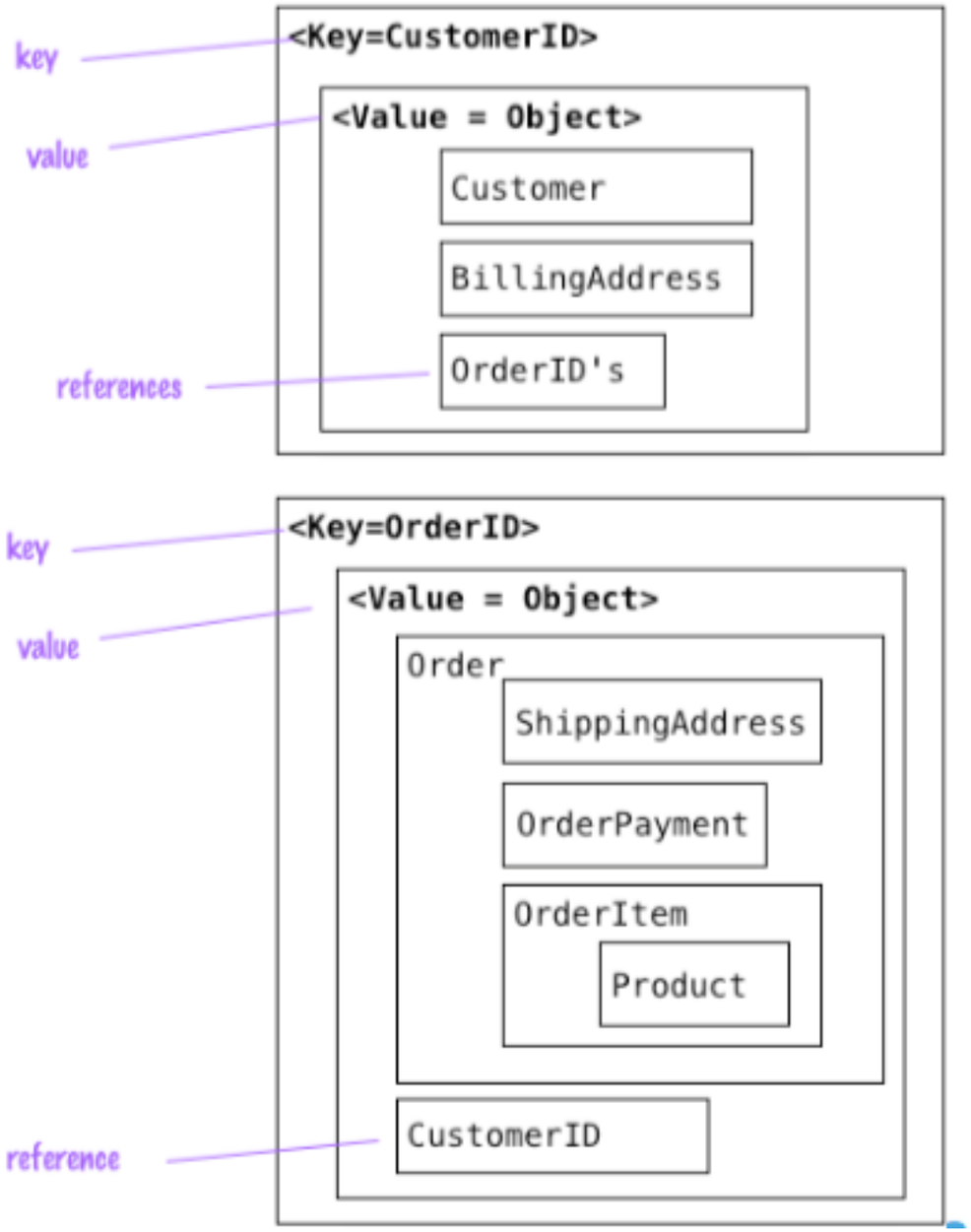
แนวคิดการรวมข้อมูล (Aggregate Data)




```
// in customers
{
  "customer": {
    "id": 1,
    "name": "Martin",
    "billingAddress": [{"city": "Chicago"}],
    "orders": [
        {
          "id":99,
          "orderItems":[
            {
              "productId":27,
              "price": 32.45,
              "productName": "NoSQL Distilled"
            }
          ],
          "shippingAddress":[{"city":"Chicago"}]
          "orderPayment":[
            {
              "ccinfo":"1000-1000-1000-1000",
              "txnId":"abelif879rft",
              "billingAddress": {"city": "Chicago"}
            }
          ],
        }
    ]
}}
```

Aggregate เป็น object ที่เข้าถึงด้วย key

แนวคิดการรวมข้อมูล (Aggregate Data)




```
    // in Customers
{
    "id":1,
    "name":"Martin",
    "billingAddress":[{"city":"Chicago"}]
}
// in Orders
{
    "id":99,
    "customerId":1,
    "orderItems":[
      {
        "productId":27,
        "price": 32.45,
        "productName": "NoSQL Distilled"
      }
    ],
    "shippingAddress":[{"city":"Chicago"}]
    "orderPayment":[
      {
        "ccinfo":"1000-1000-1000-1000",
        "txnId":"abelif879rft",
        "billingAddress": {"city": "Chicago"}
      }
    ],
}
```

Aggregate ที่มีการทำ reference

ผลของการ aggregate data

- สามารถกระจายข้อมูลในคลัสเตอร์ได้ง่าย
- ลดภาระการทำ ACID transaction
  - กระทำในระดับ aggregate เท่านั้น
- การอัพเดทที่กระทำข้าม aggregate ค่อนข้างยุ่งยาก
  - ต้องจัดการในระดับโปรแกรมเพื่อให้ได้ ACID ข้าม aggregate

NoSQL และ Aggregation

- NoSQL ที่มีพื้นฐานมาจาก data aggregation
  - Key-Value
  - Document
  - Column-Family
- NoSQL ที่ไม่ได้มาจาก data aggregation
  - Graph

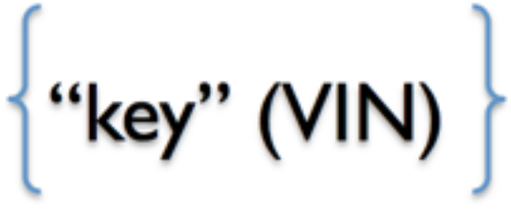
ฐานข้อมูลแบบ Key-Value

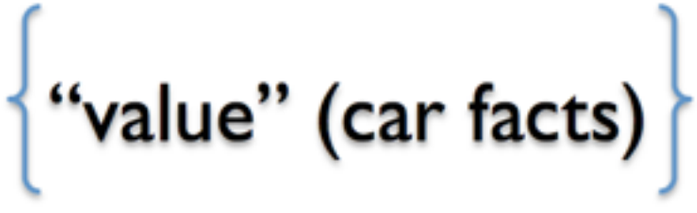
- หนึ่ง key เข้าคู่กับ value เพียงหนึ่งเดียว
- ฐานข้อมูลไม่สามารถกระทำการโดยตรงกับ value ได้ (Value is opaque to the database)
- ลักษณะใกล้เคียง hash และเทียบเคียงได้กับ RDBMS ที่มี 2 คอลัมน์
- ทำ CRUD แบบง่ายๆได้เร็วมากแต่ query ที่ซับซ้อนทำได้ไม่ดี
- กระทำการแบบ atomic กับ single key เท่านั้น
- ทำ sharding เพื่อ scale out ได้ง่าย

ตัวอย่างของฐานข้อมูล Key-Value

riak

redis





Project Voldemort
A distributed database.

MEMCACHED

การใช้งาน Key-Value

- ใช้ได้ดีในงานต่อไปนี้
  - ข้อมูลเซสชั่นสำหรับ web application
  - ข้อมูลผู้ใช้และความชอบ
  - ข้อมูลตะกร้าซื้อของสำหรับ e-commerce web
- ไม่เหมาะสมในงานต่อไปนี้
  - มีความสัมพันธ์ระหว่างข้อมูลมากอย่างข้อมูล social network
  - ต้องการ consistency สำหรับปฏิบัติการที่มี key หลายๆอันมาเกี่ยวข้อง
  - ต้องการทำ query ผ่าน value ด้วย

ฐานข้อมูลแบบ Document

- หนึ่ง key เข้าคู่กับ value เพียงหนึ่งเดียว
- ฐานข้อมูลกระทำการโดยตรงกับ value ได้ (Value is visible to the database)
  - สามารถทำ query โดยใช้วัตถุที่อยู่ใน value ได้
- ส่วนใหญ่ value ของฐานข้อมูล document จะอยู่ในรูป XML หรือ JSON และมีโครงสร้างเป็น tree กระทำการแบบ atomic กับ single key เท่านั้น
- กระทำการแบบ atomic กับ single document เท่านั้น
- ทำ sharding เพื่อ scale out ได้ง่าย

ตัวอย่างของฐานข้อมูล Document

mongoDB

RAVEN DB

“id” (VIN)

=

“document” (car facts)

CouchDB relax

terrastore

OrientDB

JTTDR…

=

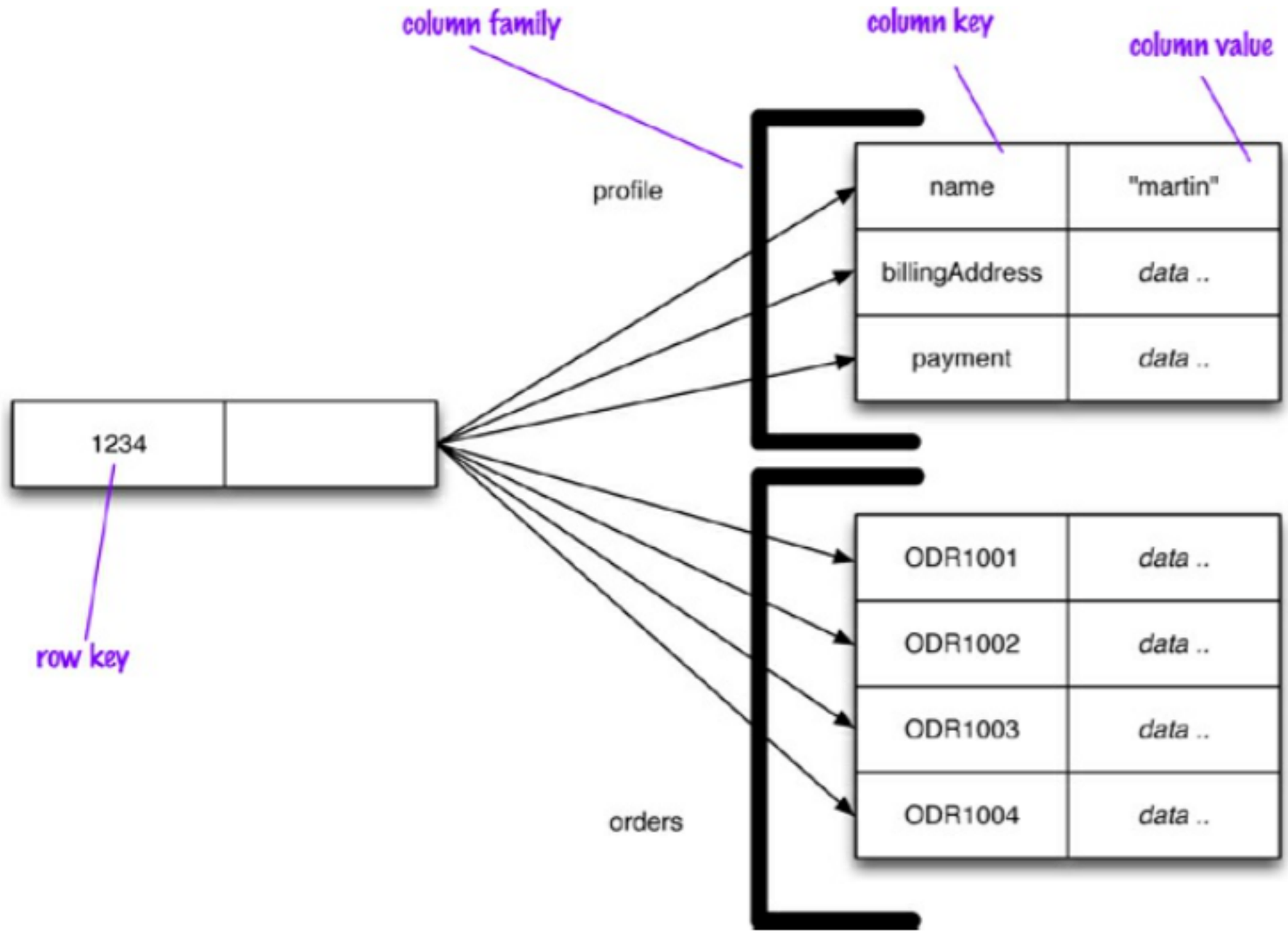
{…
“make”: “Ford”,
“model”: “Mustang”,
“year”: 2011,
…
}

การใช้งาน Document

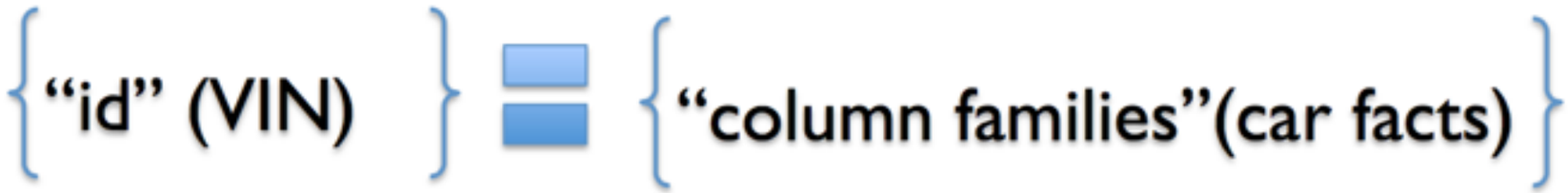
- ใช้ได้ดีในงานต่อไปนี้
  - ข้อมูลการบันทึกเหตุการณ์ที่เกิดขึ้น (event logging)
  - Web application ประเภท blogging
  - Web Analytics หรือ Real-Time Analytics
    - อัพเดท document เฉพาะบางส่วน เพิ่มหรือลด metric ที่ต้องการวัดและวิเคราะห์ได้ง่าย
- ไม่เหมาะสมในงานต่อไปนี้
  - ต้องการทำ query ที่ซับซ้อนและไม่เข้ากับโครงสร้างของ document ที่ aggregate มา
  - ต้องการ consistency สำหรับปฏิบัติการที่มี document หลายๆอันมาเกี่ยวข้อง

ฐานข้อมูลแบบ Column-Family

- การจัดการข้อมูลทำในระดับคอลัมน์
- แต่ละแถวมี key เฉพาะสำหรับการเข้าถึงข้อมูลในคอลัมน์
- แถวที่อยู่ใน column-family เดียวกันไม่จำเป็นต้องมีจำนวนคอลัมน์เท่ากัน
- เป็นเหมือน two-level map โดย key แรก (row key) นำเข้าหา column-family แล้ว column key นำเข้าหา column value อีกต่อหนึ่ง
- กระทำการแบบ atomic ในระดับแถวเท่านั้น


column family
column key
column value
profile
name
"martin"
billingAddress
data ..
payment
data ..
1234
row key
ODR1001
data ..
ODR1002
data ..
ODR1003
data ..
ODR1004
data ..
orders

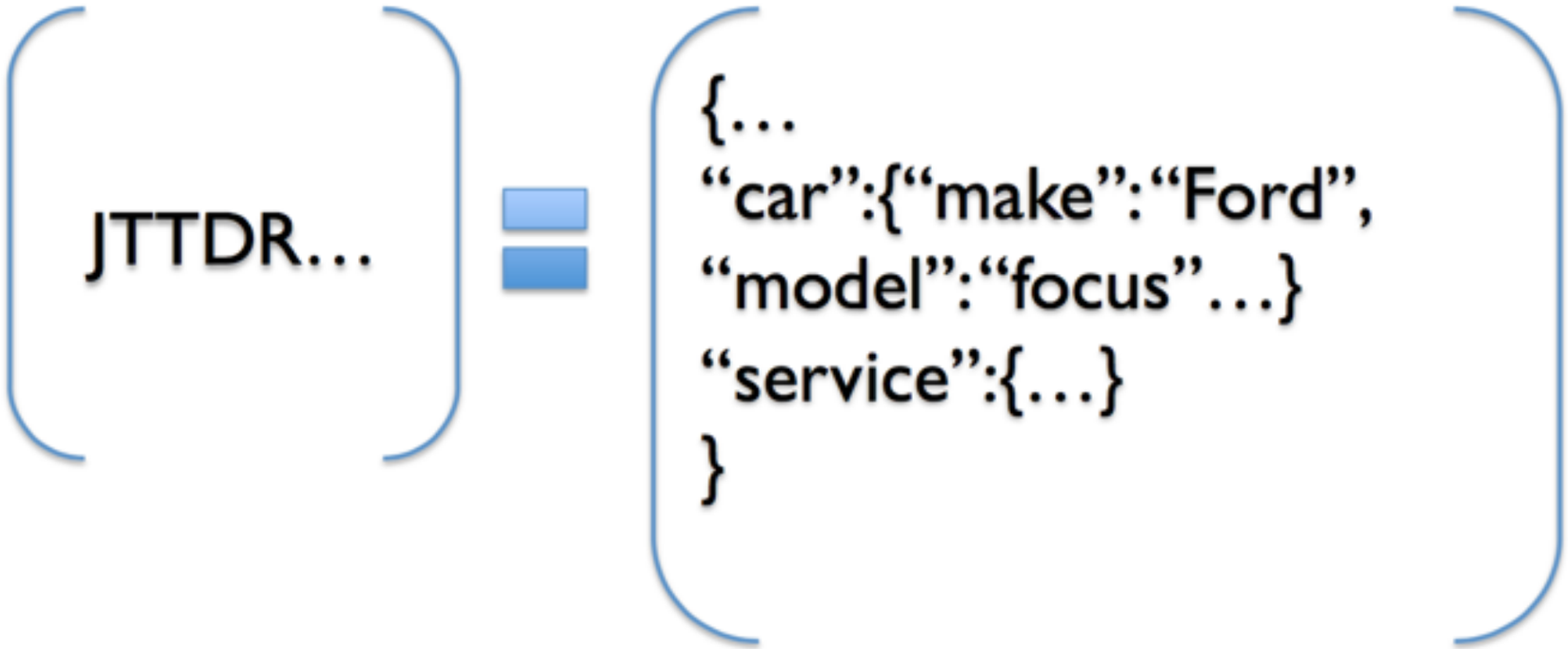
ตัวอย่างของฐานข้อมูล Column-Family



Cassandra

HYPERTABLE INC

APACHE HBASE

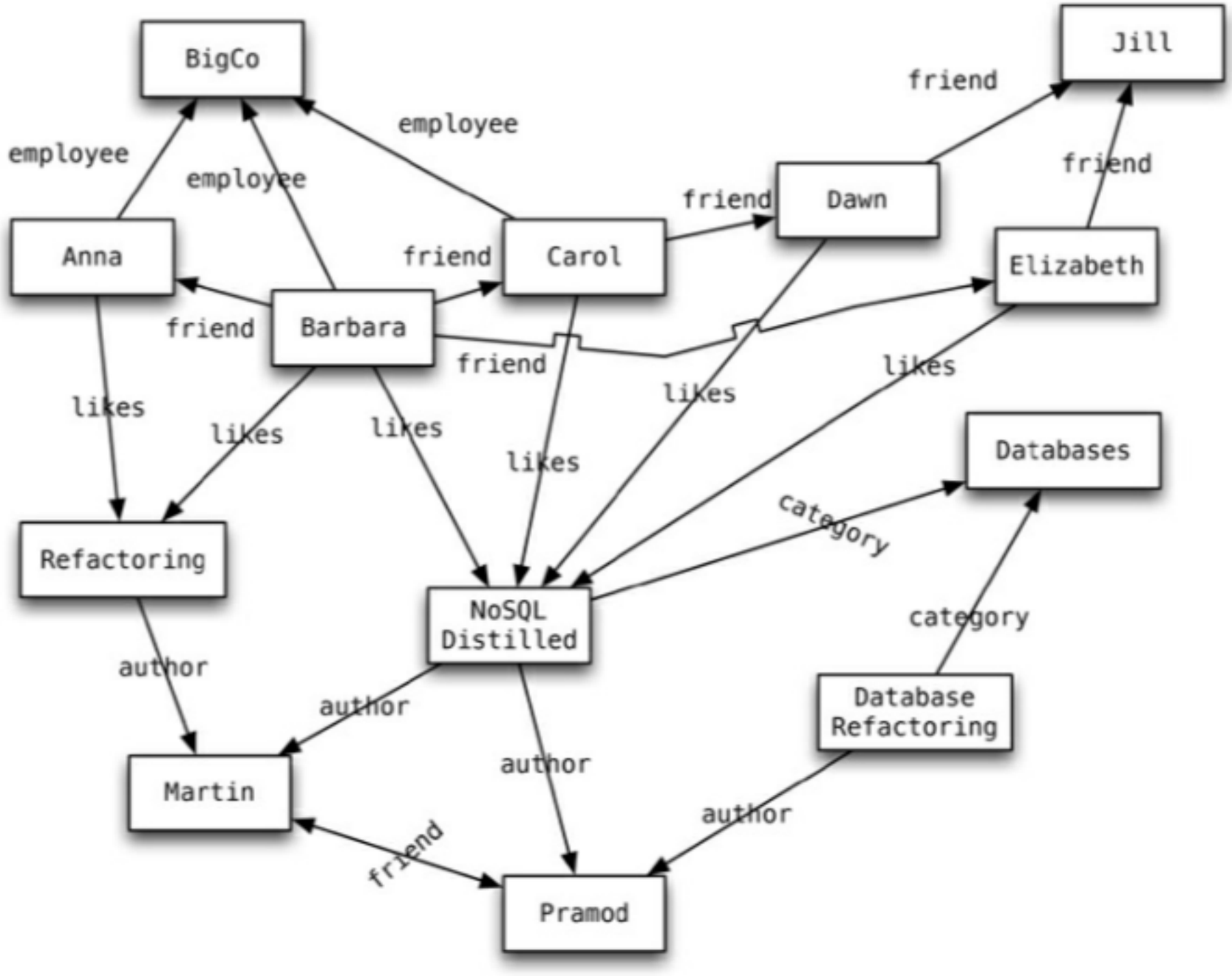


การใช้งาน Column-Family

- ใช้ได้ดีในงานต่อไปนี้
  - ข้อมูลการบันทึกเหตุการณ์ที่เกิดขึ้น (event logging)
  - Web application ประเภท blogging
  - Web Analytics หรือ Real-Time Analytics
- ไม่เหมาะสมในงานต่อไปนี้
  - งาน prototype ที่ต้องมีการเปลี่ยนแบบคอลัมน์บ่อย
  - ต้องการ consistency ข้าม column-family

ฐานข้อมูลแบบ Graph

- ไม่ได้มีพื้นฐานมาจากการทำ aggregation เหมือน NoSQL ทั้งสามแบบที่กล่าวมาแล้ว
- เน้นข้อมูลที่มีขนาดเล็กแต่มีความสัมพันธ์ (relationship) ที่ซับซ้อน
- มีโครงสร้างเหมือน graph data structure มี node มี edges
- ปฏิบัติการพื้นฐานคือการทำ graph traversal
- ได้ ACID เหมือน RDBMS; ไม่กระจายข้อมูลไปตามโหนดต่างๆ



Query: “หาหนังสือในกลุ่มฐานข้อมูลที่เพื่อนของ Barbara ที่เป็นพนักงานของ BigCo ชอบ”

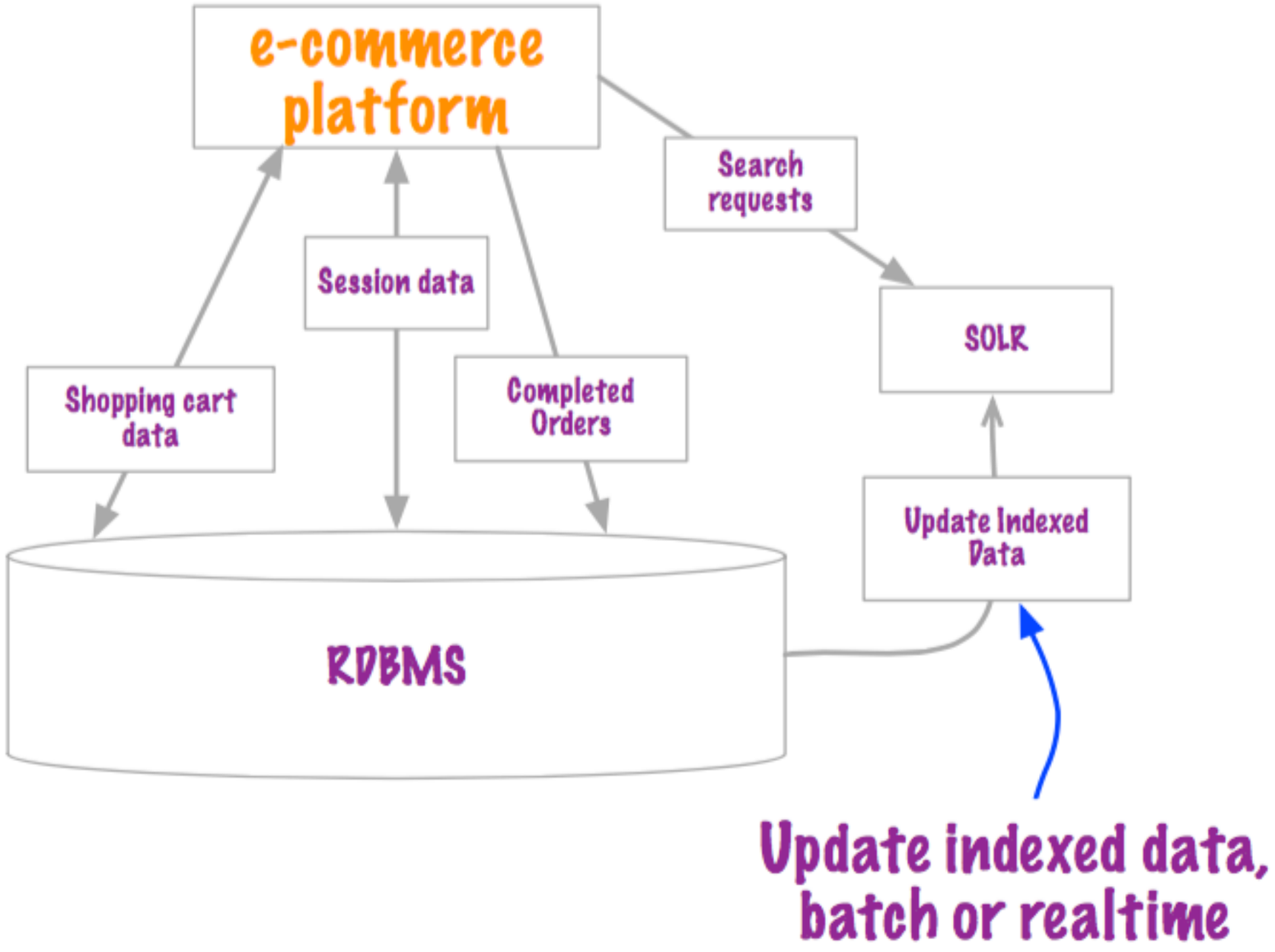
การใช้งาน Graph

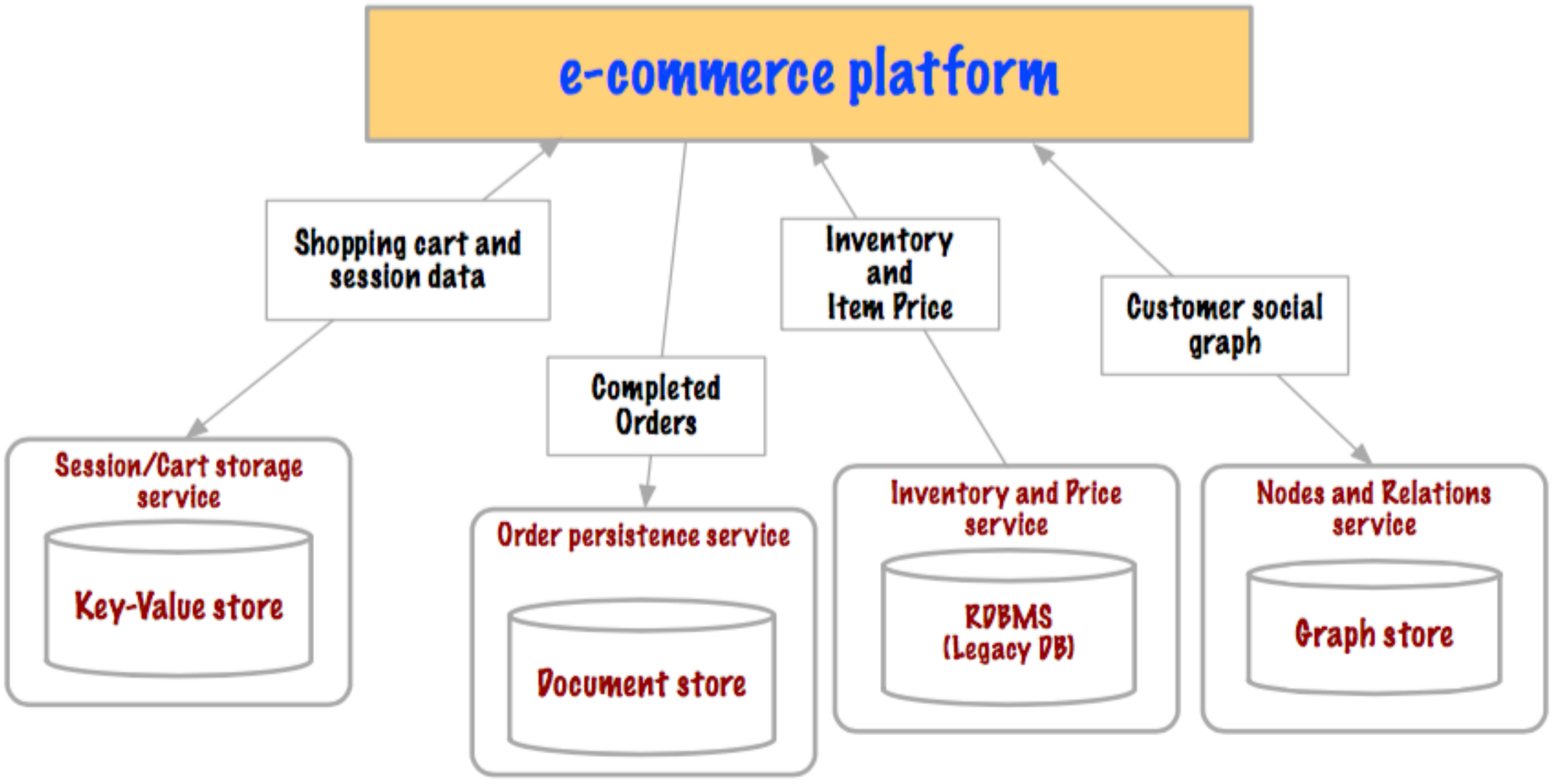
- ใช้ได้ดีในงานต่อไปนี้
  - ข้อมูลที่มีการเชื่อมต่อกันสูง (social networking)
  - ระบบแนะนำ (recommendation engine)
- ไม่เหมาะสมในงานต่อไปนี้
  - การอัพเดท node ใน graph บ่อยๆ
  - ต้องการ scale out เพื่อรองรับข้อมูลและผู้ใช้งานจำนวนมาก

โลกของ Polyglot Persistence

- ใช้เทคโนโลยีฐานข้อมูลที่เหมาะสมสำหรับแต่ละงานและความต้องการ

- ห่อหุ้ม (encapsulate) ข้อมูลและสื่อสารข้อมูลผ่าน web service


e-commerce platform
Search requests
Session data
SOLR
Shopping cart data
Completed Orders
Update Indexed Data
RDBMS
Update indexed data, batch or realtime


e-commerce platform
Shopping cart and session data
Completed Orders
Inventory and Item Price
Customer social graph
Session/Cart storage service
Key-Value store
Order persistence service
Document store
Inventory and Price service
RDBMS (Legacy DB)
Nodes and Relations service
Graph store

บทสรุป

- เราได้กล่าวถึงสิ่งต่อไปนี้
  - NoSQL ที่ทำ data aggregation
    - Key-Value
    - Document
    - Column-Family
  - NoSQL ที่เน้นเรื่อง relationship
    - Graph
- แนวทางในปัจจุบันไปทาง polyglot resistance
  - เลือก database ให้เหมาะกับ application
  - ทำ integration ผ่าน web service