Intro to NoSQL

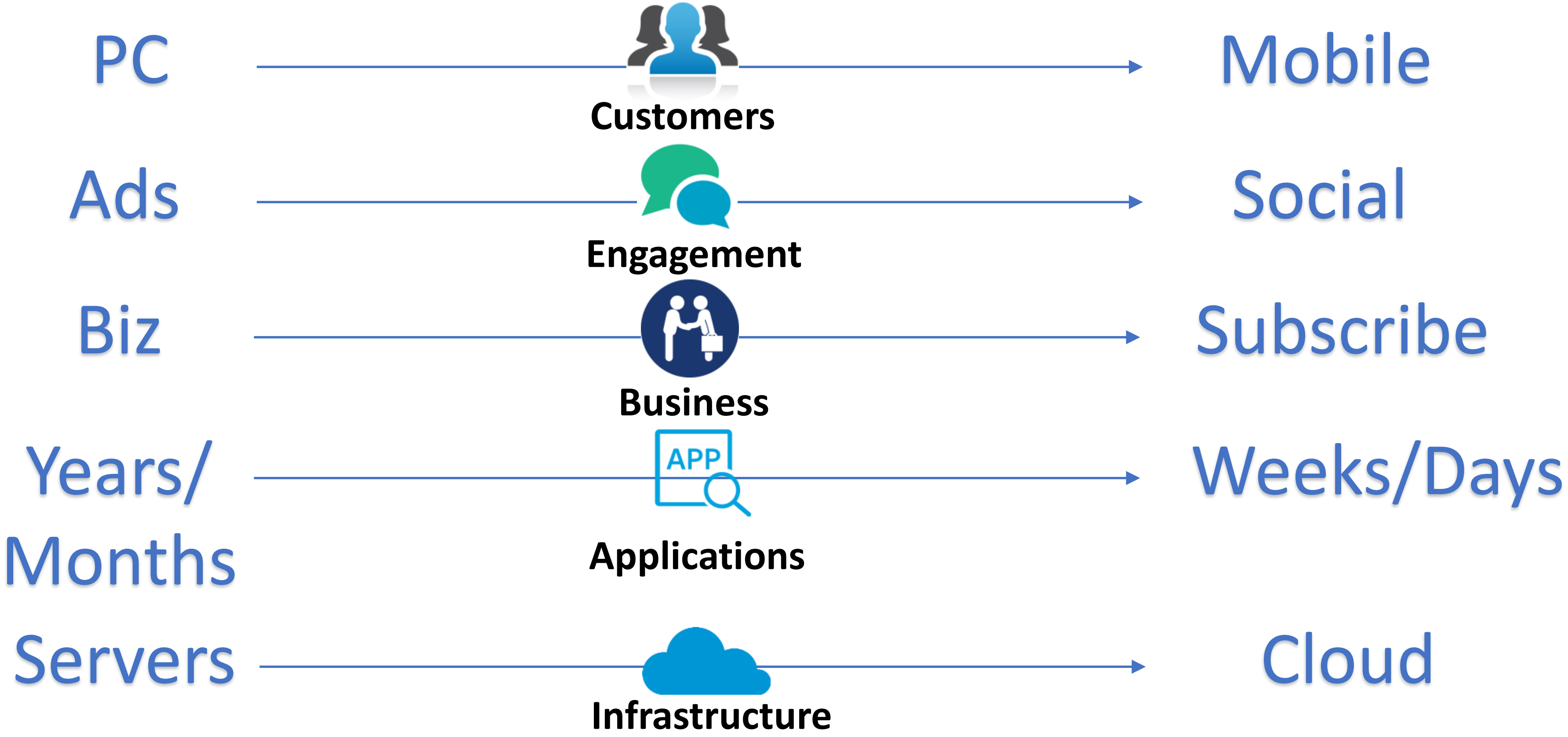
Poo Kuan Hoong

Introduction

- A NoSQL (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

Digital platforms have changed

- Digital platforms have changed over the past 5 years



PC → Mobile
Customers

Ads → Social
Engagement

Biz → Subscribe
Business

Years/ Months → Weeks/Days
Applications

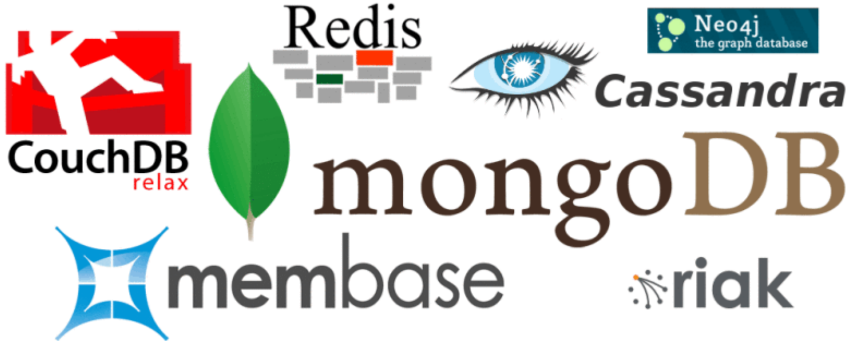
Servers → Cloud
Infrastructure

The New Enterprise Stack

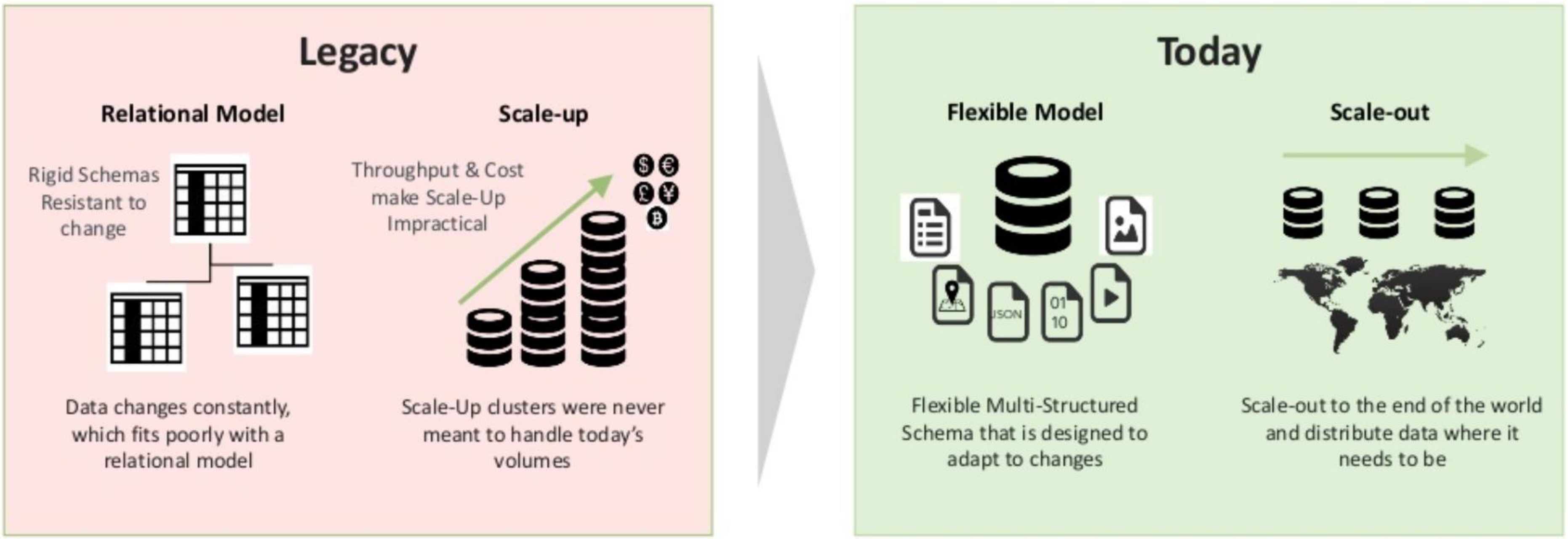
	Traditional	Modernized
Apps	On-premised, monoliths	SaaS, Microservices
Database	Relational (Oracle)	Non-relational (MongoDB)
EDW	Teradata, Oracle, etc	Hadoop
Compute	Scale-up server	Containers/Commodity Server/ Cloud
Storage	SAN	Local Storage & Data Lakes
Network	Routers & Switches	Software defined Networks

Why do we need NoSQL?

- We are storing more data now than we ever have before
- Connections between our data are growing all the time
- We don't make things knowing the structure from day 1
- Server architecture is now at a stage where we can take advantage of it



The challenges of using RDBMS



Legacy

Relational Model

Rigid Schemas Resistant to change

Data changes constantly, which fits poorly with a relational model

Scale-up

Throughput & Cost make Scale-Up Impractical

Scale-Up clusters were never meant to handle today's volumes

Today

Flexible Model

Flexible Multi-Structured Schema that is designed to adapt to changes

Scale-out

Scale-out to the end of the world and distribute data where it needs to be

NoSQL Use Cases

- Large data volumes
  - Massively distributed architecture
  - Required to store the data
  - Google, Amazon, Facebook, 100k servers
- Extreme query workload
  - Impossible to efficiently do joins at that
  - Scale with an RDBMS
- Schema evolution
  - Schema flexibility is not trivial at a large
  - Scale but it can be with NoSQL

NoSQL: Pros and Cons

- **Pros**
  - Massive scalability
  - High availability
  - Lower cost
  - Schema flexibility
  - Sparse and semi structured data
- **Cons**
  - Limited query capabilities
  - Not standardised (portability may be an issue)
  - Still a developing technology

BIGTABLE

KEY VALUE

FOUR
EMERGING TRENDS IN NOSQL DATABASES

GRAPHDB

DOCUMENT

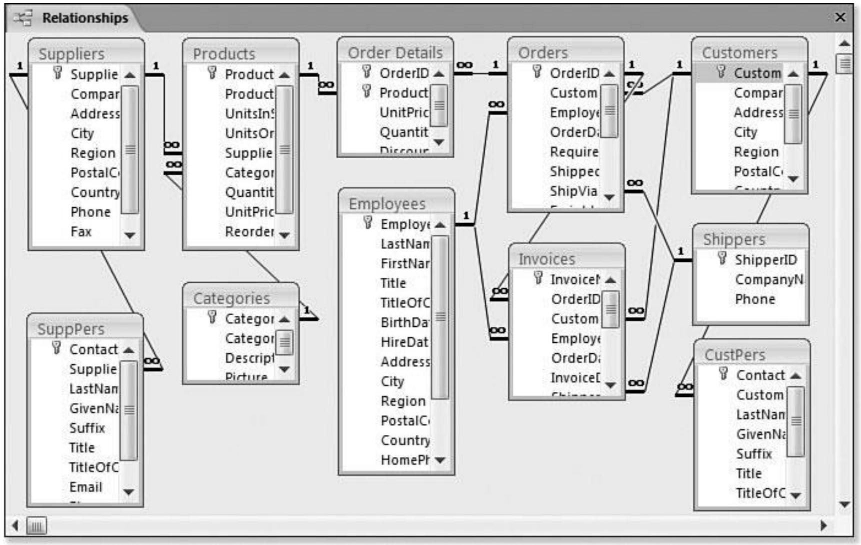
Big Table

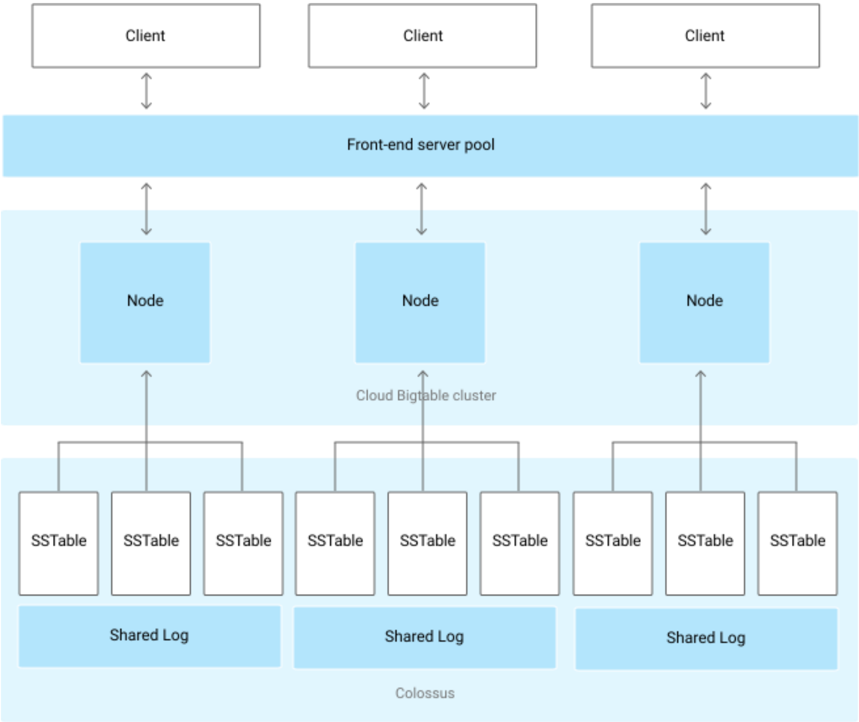
- Behaves like a standard relational database but with a slight change
- Designed to work with a lot of data…a really big crap ton
- Created by google and now used by lots of others

- http://research.google.com/archive/bigtable.html
- http://research.google.com/archive/spanner.html

Big Table

- “A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.”





Relational Database vs Big Table

Key value database

- Again, designed to work with a lot of data
- Each bit of data is stored in a single collection
- Each collection can have different types of data

cassandra

amazon
DynamoDB

ORACLE®

NOSQL DATABASE

Document Oriented Database

- A document-oriented database, or document store, is a computer program designed for storing, retrieving and managing document-oriented information, also known as semi-structured data. Document-oriented databases are one of the main categories of NoSQL databases
- Very similar to a key value database, where main difference is that you can actually see the values
- CRUD operations

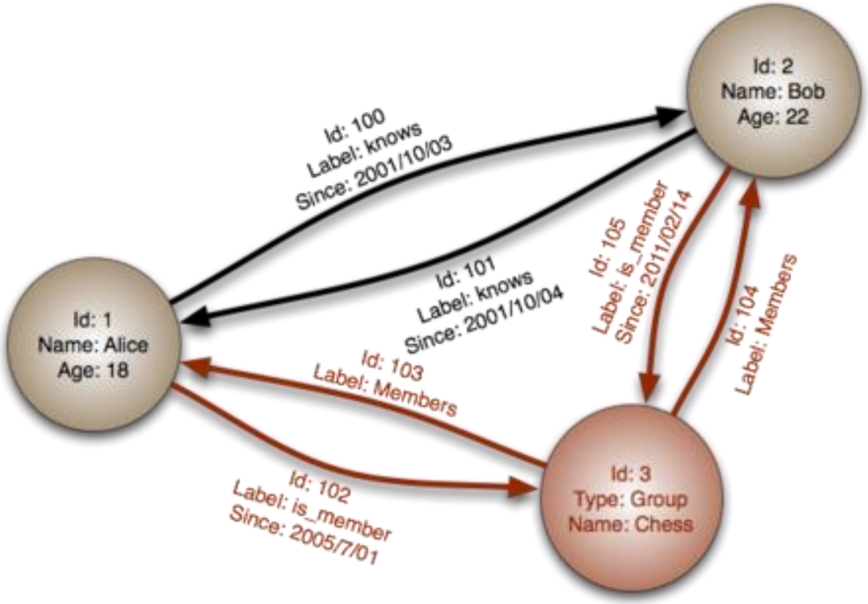
CouchDB
relax

mongoDB

SAP® HANA

Graph Database

- Focus here is on modelling the structure of the data
- Inspired by graph theory
- Scales really well to the structure of the data



Graph Database

- Neo4j is a graph database management system developed by Neo4j, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing
- Neo4j is the most popular graph database according to DB-Engines ranking

NoSQL: Tips

- High availability and disaster recovery are a must
- Understand the pros and cons of each design model
- Don’t pick something just because it is new
- Don’t pick something based JUST on performance

SQL: The good

- High performance for transactions. Think ACID
- Highly structured, very portable
- Small amounts of data
- SMALL IS LESS THAN 500GB
- Supports many tables with different types of data
- Can fetch ordered data
- Compatible with lots of tools

SQL: ACID

- Atomicity
- Consistency
- Isolation
- Durability

SQL: The bad

- Complex queries take a long time
- The relational model takes a long time to learn
- Not really scalable
- Not suited for rapid development

NoSQL : The good

- Fits well for volatile data
- High read and write throughput
- Scales really well
- Rapid development is possible
- In general it’s faster than SQL

NoSQL: The bad

- Key/Value pairs need to be packed/unpacked all the time
- Still working on getting security for these working as well as SQL
- Lack of relations from one key to another

Summary

SQL

works great, can’t scale for large data

NoSQL

works great, doesn’t fit all situations

so use both, but think about when you want to use them!