

# IST346

Data and Database Systems

# Agenda

- Data
- Databases
- Backup

Data

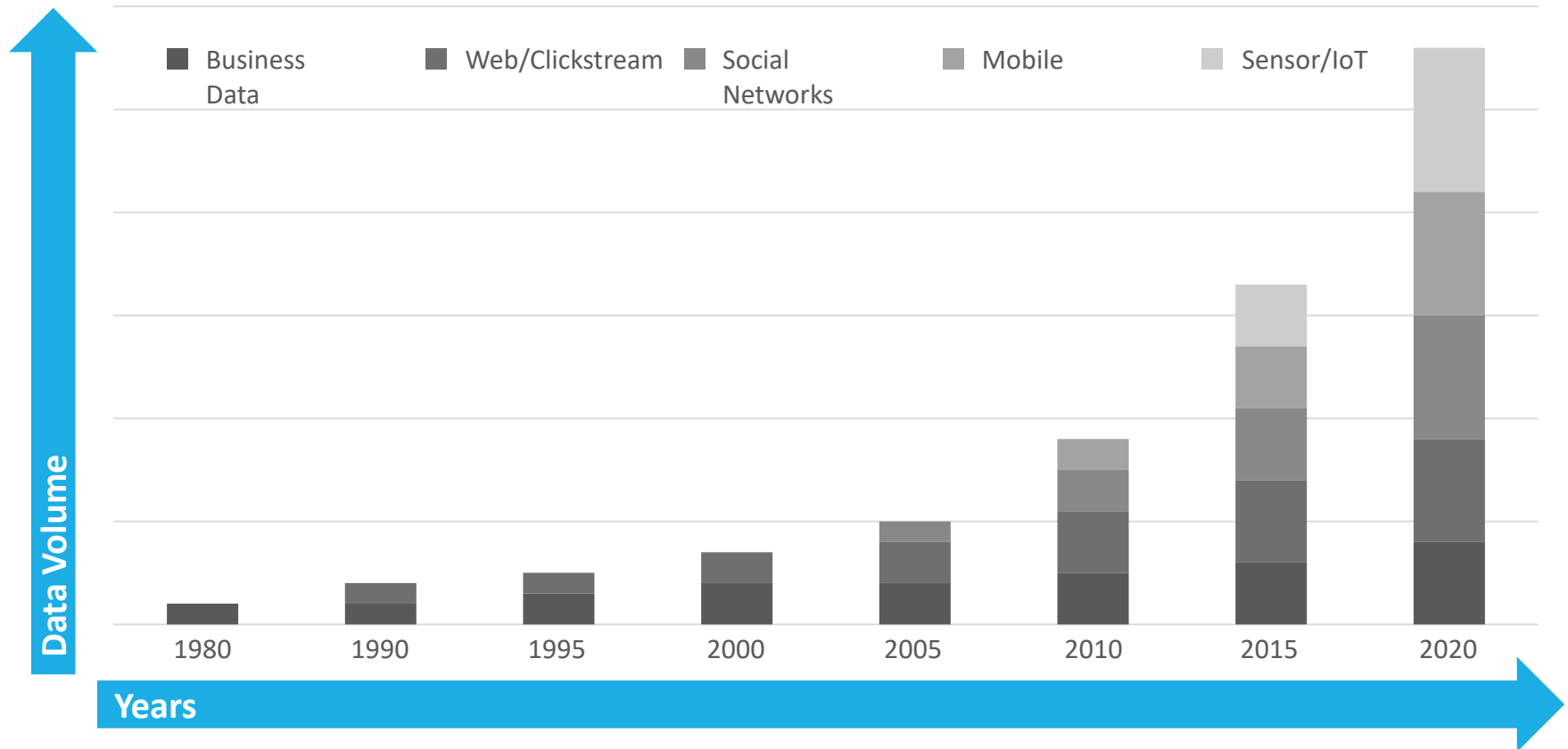
# A Organization's Most Important Asset... is It's Data!!!

- Data Drives the Business
- Identify trends so you can respond
- Use knowledge as competitive advantage

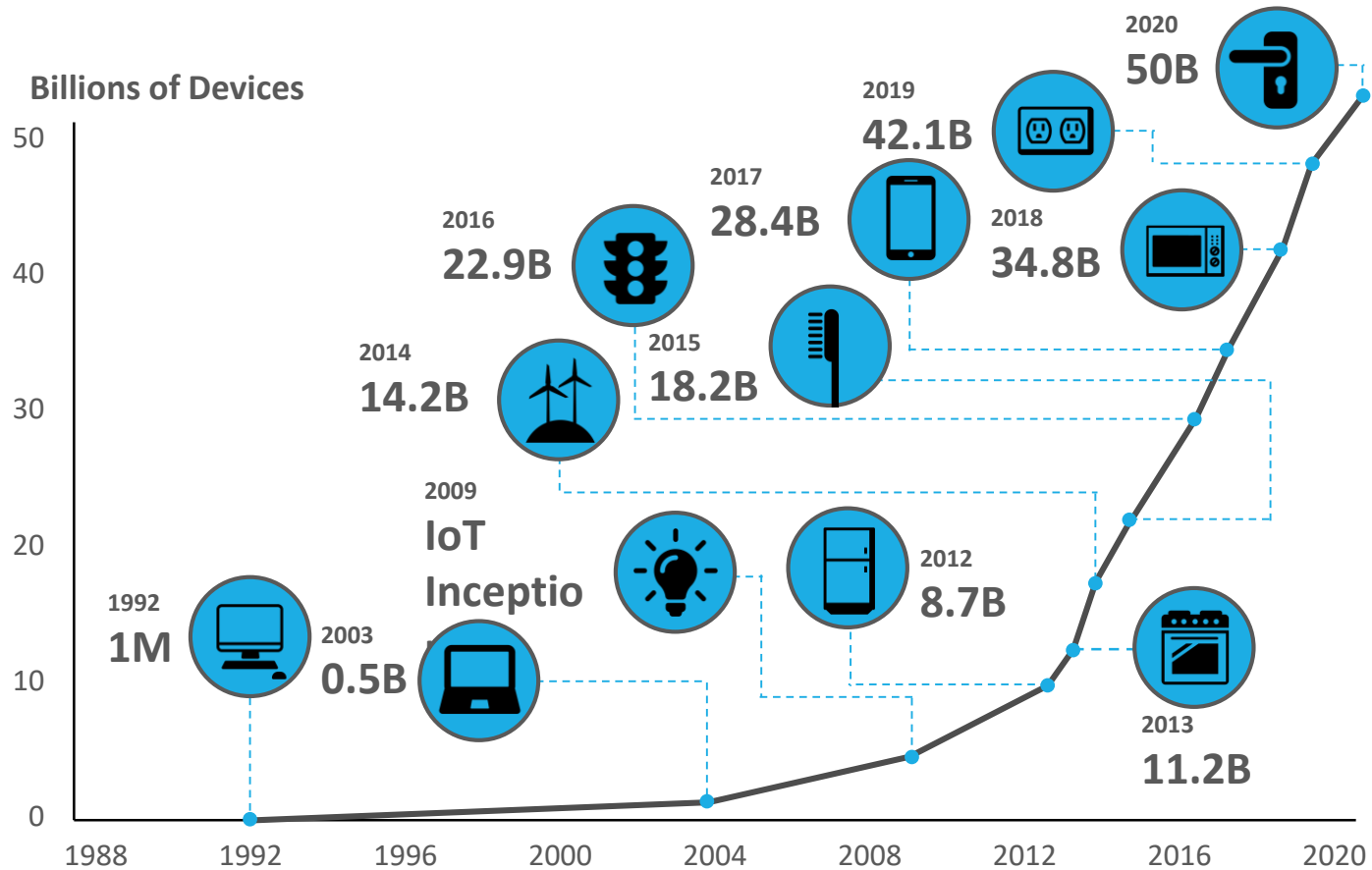
# Prevalence of Data

- Data are everywhere!
- There has been an explosion of data over the past few years.
- There are new varieties of data.
  - Business data → Sales and Orders
  - user generated → Tweets and Posts
  - device generated → Phones and IoT
- And there are different mediums than before.
  - Structured text → unstructured text → multimedia

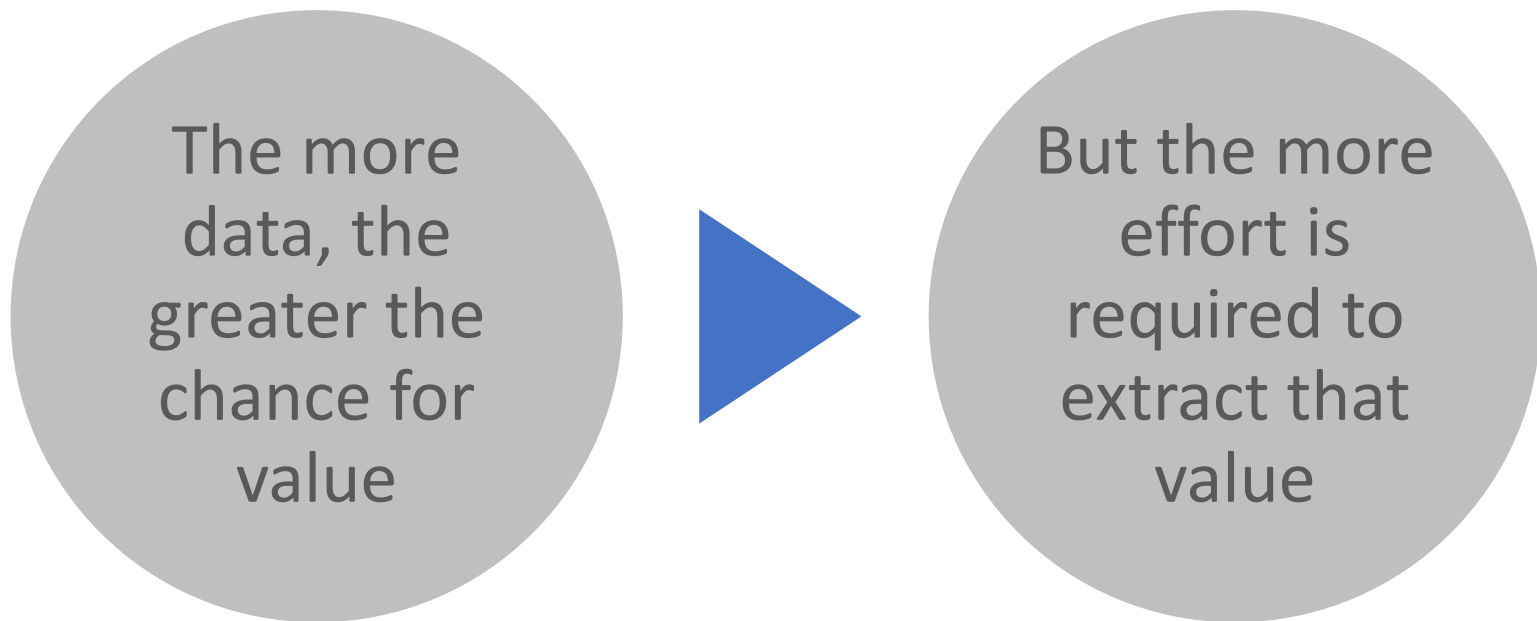
# Data Explosion: A Sample Growth of Organizational Data Over Time



# What About Devices?

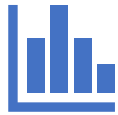


# Data Have Always Held Value





# Recall: Layers of a Modern Data-Oriented Application



## Presentation

Code and layout responsible for the user interface



## Business Logic

Transformational logic at the heart of what the application actually does



## Data Access

Create, read, update and delete (CRUD) operations

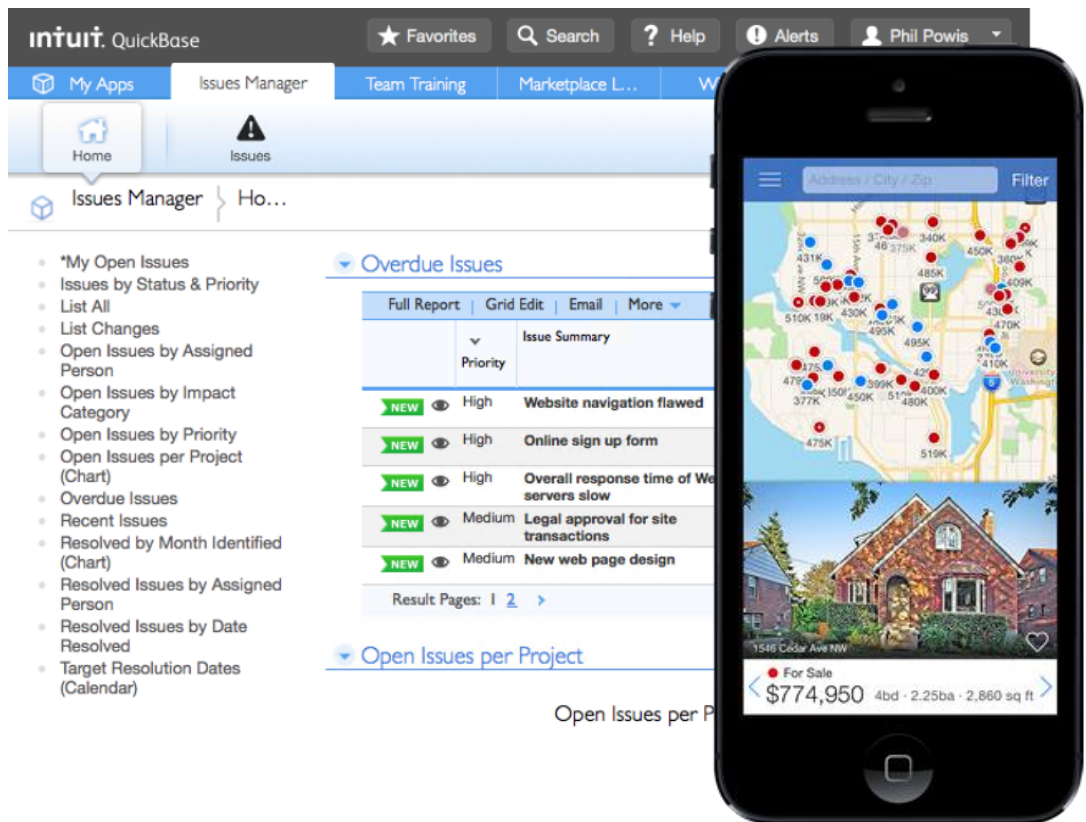


## Database

Data storage and retrieval

# Presentation Layer

- User interface concerns
- Web: HTML and CSS
- Mobile: Xcode interface builder/Android studio



# Business Logic Layer

- Main transformational logic of the application; part of the application's functionality
- Written in a programming language: Java, JavaScript, Python, C#, and so on



✓ 1 item added to Cart




Certified Refurbished Fire HD 8 Tablet  
with Alexa, 8" ...

\$64.99

☐ This is a gift  
Why is this important?

Order subtotal: \$64.99  
1 item in your Cart

 Edit your Cart

Proceed to checkout 

# Data Access Layer

- Responsible for CRUD operations
- Typically, this is code that transforms operations into the DSL (domain-specific language) to communicate with the database (typically SQL)

```
insert into fudgemart_customers ...
```

```
update fudgemart_customers set ...
```

```
delete from fudgemart_customers where...
```

```
select * from fudgemart_customers
```

# Database

- A generic term for the storage, management and retrieval of data
- Many Types Exist
- Serve Several Purposes

# Types of Databases

# Relational Databases

- Based on Relational theory, data are stored as rows in tables.
- Very proven database model with wide adoption in industry
- Uses custom query language SQL
- Does not scale easily horizontally
- Use cases
  - You need data consistency
  - Ad-Hoc Reporting and Querying
- Products
  - Oracle, Postgres, SQL Server, MySQL

# Key-Value Stores

- Basically a persistent hash map
  - Simple
  - Fast reads and writes
  - No secondary indexes
- Scales Horizontally
- Use cases
  - Data model is simple
  - All you need is CRUD (Create/Read/Update/Delete)
  - Caching frameworks
- Challenges
  - Difficult to perform complex queries
- Products
  - DynamoDB, Riak, Redis



# Columnar Stores

- Data in tables are stored by column instead of by row
  - Storage not wasted on null values
  - Fast operations on columns such as aggregation of data
  - Scales horizontally
- Use cases
  - Data analytics, big data sets
  - Timeseries data
- Challenges
  - Key design is not trivial
  - Need to split data according to how it will be queried
- Products
  - HBase, Cassandra, MemSQL

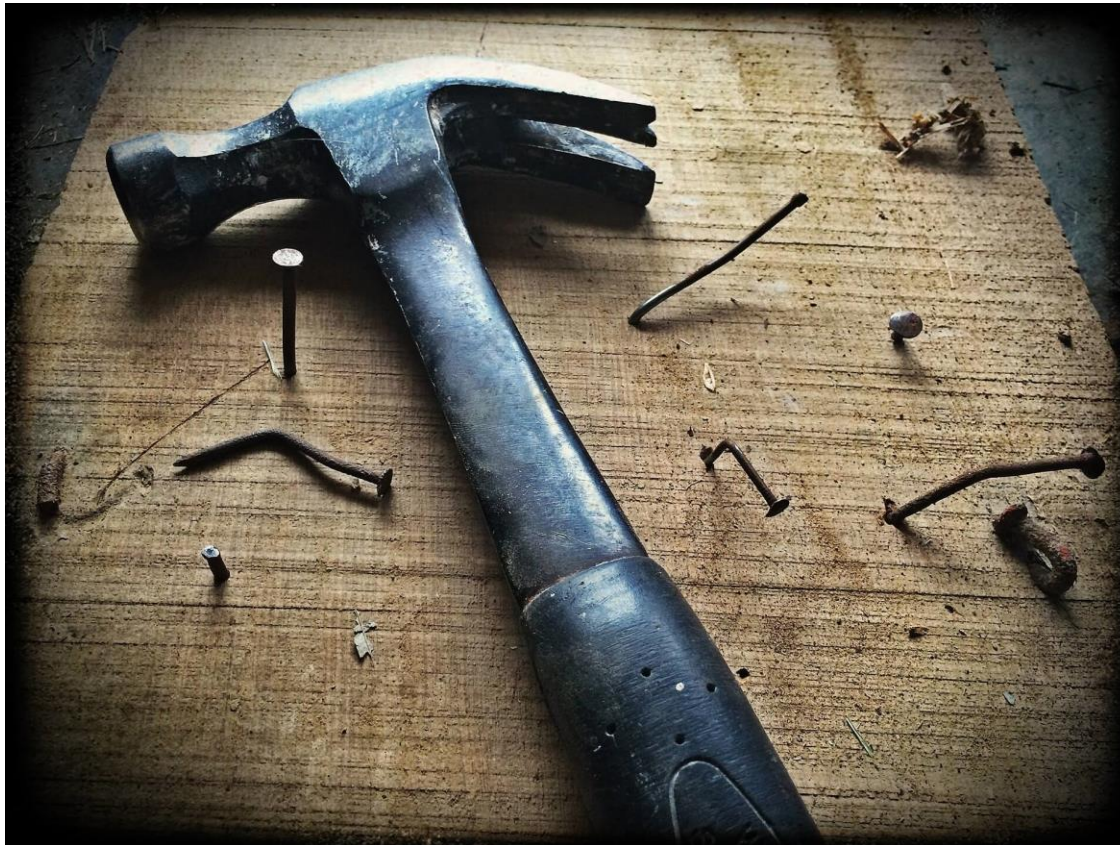
# Document Stores

- Nested structures of hashes and their corresponding values
  - Very flexible schema
  - No need to normalize
- Use cases
  - Applications where the schema is likely to change
  - When you don't need the flexible query of relational, but need better performance
- Challenges
  - Complex queries with joins are slow
  - Documents which reference themselves and other circular dependencies
- Products
  - MongoDB, CouchDB, RavenDB

# Graph Data Stores

- Nodes and edges
  - Good fit for highly interconnected data
  - Allows for explicit relationships among data items
  - Based on Graph theory
- Use cases
  - When your data look like a graph or hierarchy
- Challenges
  - Does not scale well horizontally
  - Very specific use cases—know when to use it!
- Products
  - Neo4j, SQL Server Graph Tables

When You Treat Every Data Problem Like a Nail,  
The Same Database System Becomes a Hammer



# One Size Does Not Fit All

- There is no one database management system that can handle the complexity and variety of data found today.
- This is why different systems exist to manage different types of data that vary in:
  - Structure
  - Size
  - Rate of change
- We've learned the hard way that not all data problems are nails, and there is more than just the hammer.

# So How Do You Choose?

- What type of data model is it? Graph? Timeseries? Real-time? Historical?
- Do you need a schema (structure)? Does it require schema on write or on read?
- Would you benefit from integrity constraints, or do you need to guarantee writes?
- Do you have cross-cutting query concerns?
- Do you need to scale out?

# Don't Settle for One— Polyglot Persistence!

Today's enterprises have many databases:

- Product catalog and blog in MongoDB
  - Redis for your shopping cart and web page cache
  - Cassandra for audit and activity logs
  - MySQL for order processing and payments
  - Hadoop for data analytics
  - Neo4j for your social graph
- 
- Hammer. Nail. Screwdriver. Screw. Use the right tool for the job!

# Polyglot Persistence

## Pros

- There is better performance and scalability.
- It is cloud friendly in today's world of microservices.
- Using the best tool for the job! Stop forcing that square peg into that round hole!

## Cons

- No interoperability: You must connect the databases together.
- You must decide where data should be stored.
- There is increased complexity and administrative burden.

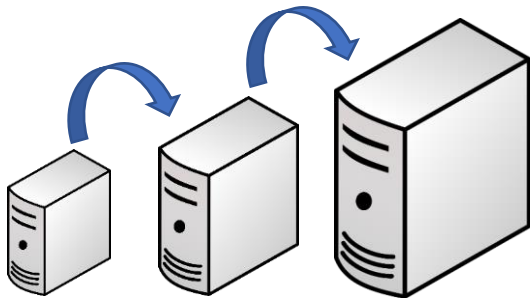


# Databases and Scalability

# DBMS Scaling: Up vs. Out

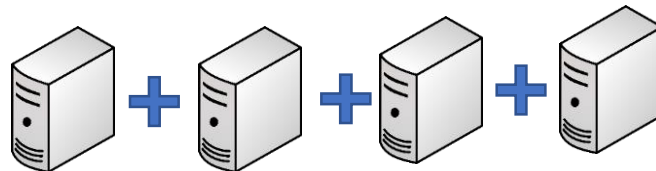
## Vertical “Scale Up”

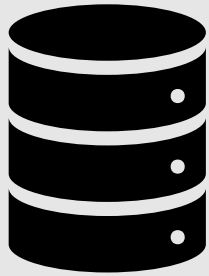
- Add more resources to an existing system running the service
- Easier, but limited scale
- Single point of failure



## Horizontal “Scale Out”

- Run the service over multiple systems, and orchestrate communication between them
- Harder, but massive scale
- Overhead to manage nodes





It's very hard to scale  
out a relational  
database.

But why?

---

**WE OFFER 3 KINDS OF SERVICES**  
**GOOD - CHEAP - FAST**

**BUT YOU CAN ONLY PICK TWO**

---

**GOOD & CHEAP** WON'T BE **FAST**

---

**FAST & GOOD** WON'T BE **CHEAP**

---

**CHEAP & FAST** WON'T BE **GOOD**

---

# CAP Theorem of Distributed Data Stores

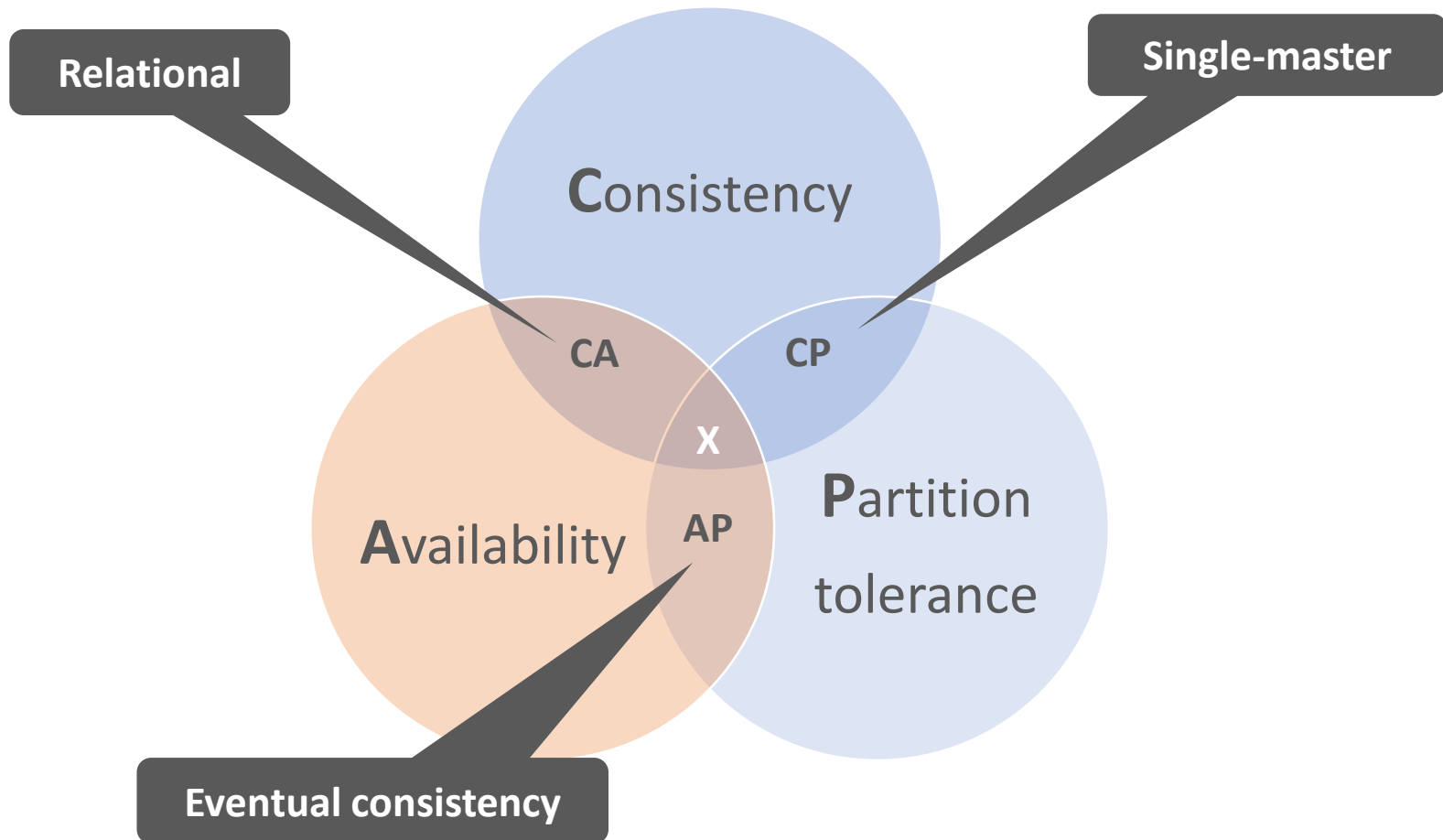
- Eric Brewer: You can only have two of the following three guarantees:
  1. **Data consistency:** all nodes see the same data at the same time
  2. **Data availability:** assurances that every request can be processed
  3. **Partition tolerance:** network failures are tolerated, the system continues to operate
- Relational systems are designed to be ***consistent*** and ***available*** and therefore cannot be ***partition tolerant***.
- If I deposit money in an ATM that is disconnected from the network, how can my bank know about that deposit?

# Why Can't You Have It All?



- **Suppose we lose partition tolerance between nodes.**
  - We must ignore any updates the nodes receive, or sacrifice consistency, or we must deny service until it becomes **available** again.
- If we guarantee **availability** of requests, despite the failure:
  - We gain **partition tolerance** (the system still works), but lose **consistency** (nodes will get out of sync).
- If we guarantee **consistency** of data, despite the failure:
  - We gain **partition tolerance** (again, system works) but lose **availability** (data on nodes cannot be changed, failure is resolved).

# Database Systems and CAP



# CAP: All Kinds of Database Systems

- RDBMSs like Oracle, MySQL, and SQL Server:
  - Focus on consistency and availability (ACID principles), sacrificing partition tolerance (and thus they don't scale well horizontally)
  - Use cases: business data, when you don't need to scale out
- Single-master systems like MongoDB, HBase, Redis, and HDFS:
  - Provide consistency at scale, but data availability runs through a single node
  - Use cases: read-heavy; caching, document storage, product catalogs
- Eventual Consistency systems like CouchDB, Cassandra, Redis and Dynamo:
  - Provide availability at scale but do not guarantee consistency
  - Use cases: write heavy, isolated activities: shopping carts, orders, social media



# ACID vs. BASE

## ACID

- Atomic: Everything in a transaction succeeds, or the entire transaction is rolled back.
- Consistent: A transaction cannot leave the database in an inconsistent state.
- Isolated: Transactions cannot interfere with each other.
- Durable: Completed transactions persist, even when servers restart and so on.

## BASE

- Basic availability: Data can be read and written to any node.
- Soft-state: Nodes may change over time, even without direct updates.
- Eventual consistency: At some point all nodes will have the same data.

# Backups and Data Integrity

# 3-2-1 Data Strategy

- 3 Copies of your data
- 2 Copies are backups (one is “live”)
- 1 Backup Copy is Off-Site

# Why Backups?

- Data gets lost
- People delete data by mistake (or on purpose)
- Archival Purposes
- Legal Issues / Subpoenas
- Data gets corrupted
- Systems crash / Disks fail
- Notebooks get lost / stolen

**You need your backups to be *reliable*.**

# Multiple Ways to Backup

- Image Backups
- “Classic” Tape Backups
- Disk-to-disk-to-tape backups (D2D2T)
- Disk-to-disk-to-disk backups (disks are cheap)
- Offsite Backup services (backup over the internet)

# Why restores?

- Most Common: Accidental File Deletion / corrupt data
  - So common that snapshot technology is used.
  - Mac time machine / Windows File History / system restore
- Pull from Archives
  - Historical snapshots of data.
  - Need recovery of user's files or email after they've left the org.
- Least Common: Storage Failure
  - Fault-Tolerant system (RAID) failure
  - Loss of data and loss of service, too

# Data Integrity

- **Data Integrity** – ensuring your data is accurate.
- How does it become corrupted?
  - Viruses / Malware
  - Buggy Software
  - Hardware failures
  - User Error
- How to you ensure data integrity?
  - Hashing – compare file to its checksum MD5/SHA256
  - Keep anti-malware software current
- Backing up inaccurate data is useless!

# Types of Backups

- A **full backup** (level 0) is a complete copy of a partition.
- A **differential backup** (level 1) is an archive of only the files that have changed since the last full backup.
- An **incremental backup** (level 2, 3, etc) is an archive of only the file that have changed since the last backup (not necessarily full backup).

Backup	Sun (F)	Mon	Tue	Wed	Thu	Fri	Sat
Full	2TB	2TB	2TB	2TB	2TB	2TB	2TB
Diff.	2TB	1GB	1.2GB	1.6GB	1.9GB	2.3GB	2.8GB
Incr.	2TB	1GB	0.2GB	0.4GB	0.3GB	0.4GB	0.5GB



# Differential or Incremental?

- Differential backup

Advantages: quicker recovery time, requiring only a full backup and the latest differential backup to restore the system.

Disadvantage: for each day elapsed since the last full backup, more data needs to be backed up, especially if a majority of the data has been changed.

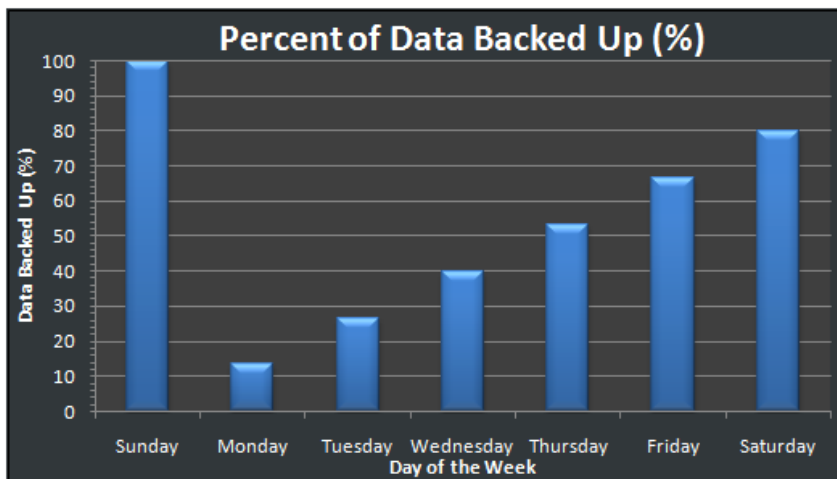
- Incremental backup

Advantages: quicker backup times, as only changed files need to be saved.

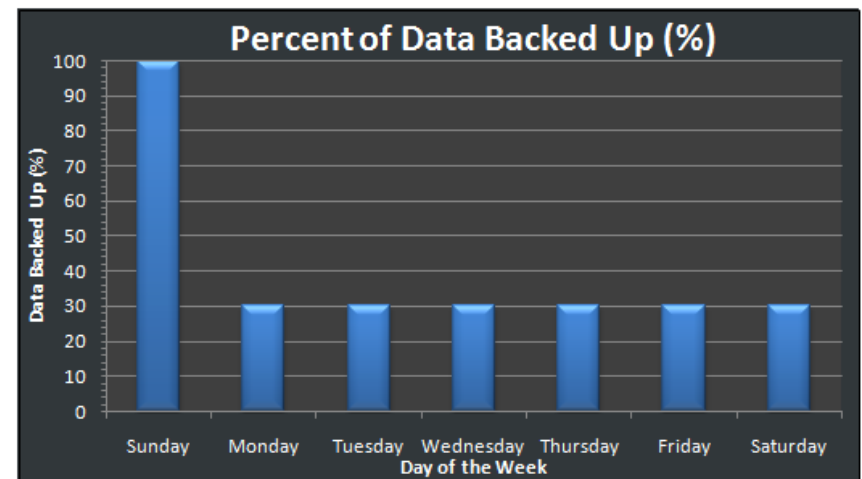
Disadvantage: longer recovery times, as the latest full backup, and all incremental backups up to the date of data loss need to be restored.

# Differential vs. Incremental

## Differential



## Incremental



# Testing Backups

- Periodically test your backups by performing restores.
- Why?  
The only way you know your backups are working is to restore data from them and test.
- Backups are no good if you can't restore from them.
- Backups are one of the most understated processes in IT management but one of the more important.

# Backup Strategy

- You can't backup everything all the time and keep it around forever.
  - It's just not realistic.
- You need a combination of short-term and long-term backups.
  - What if you need files from 12 months ago?
- You should draft a backup and restore SLA
  - Through the SLA, customers know what to expect
  - Plan your backups around the SLA
- Mitigate risk
  - Don't store your backups next to your servers!
- **The restore requirements govern your backup strategy.**

# Backup Strategy #1

- Backup
  - Sunday L0
  - Monday – Saturday L1 (Diff)
  - Each week, an L0 is saved for a year.
  - Week 52 is saved as year-end backup (not reused)
- Can this strategy Restore
  - A file from 4 days ago?
  - A file from 5 weeks ago?
  - A file from Last July, that was deleted in August?

# Backup Strategy #2

- Backup
  - Full 1<sup>st</sup> Day of each month
  - Differential each remaining day of the month.
  - Media on 1<sup>st</sup> day of the month not reused.
- Can this strategy Restore
  - A file from 25 days ago?
  - A file from 60 days ago?
  - A file from 1 year ago that was around for 2 months.