Beyond traditional RDBMS

Historic view

- As of the rise of the Web 2.0:
  - Very large volumes of data being collected
    - Web logs first, then social media, web apps, etc.
    - Data from all type of sensors (phone, cars, …)
    - Metadata from communication networks
  - Analytics on this data, of great value

- *Big Data*: different from earlier DBs by…
  - **Volume**: much larger amounts of data stored
  - **Velocity**: much higher data ingest rate
  - **Variety**: many data types, beyond relational data

Alternative DB systems

- Many apps. willing to sacrifice standard DB features (e.g., ACID properties) to enhance scalability
- Usually, an alternative is required when you need…
  - very high scalability
    E.g., feed of a social network
  - to support non-relational data
    E.g., video, documents, etc.
- Parallel processing is critical
- Different *alternative* storage systems include:
  - Sharding across multiple databases
  - Parallel and distributed databases — Today
  - NoSQL models — Rest of the semester
  - Distributed file systems — November 23-30

Sharding

- **Sharding**: partition data across **multiple databases**
- Split the DB on some **partitioning attributes**
  - *Range*: e.g., records with key in 1 to 100,000 on DB1, records with key in 100,001 to 200,000 on DB2, ...
  - *Hashing*

- Properties:
  - Scales well
  - Easy to implement

- Drawbacks:
  - Not transparent: application has to deal with routing of queries, queries spanning multiple DBs, etc.
  - If DB is overloaded, moving part of its load out is not straightforward nor easy
  - Chance of failure increases as more DBs are used
    - Use replicas (more work for application code)

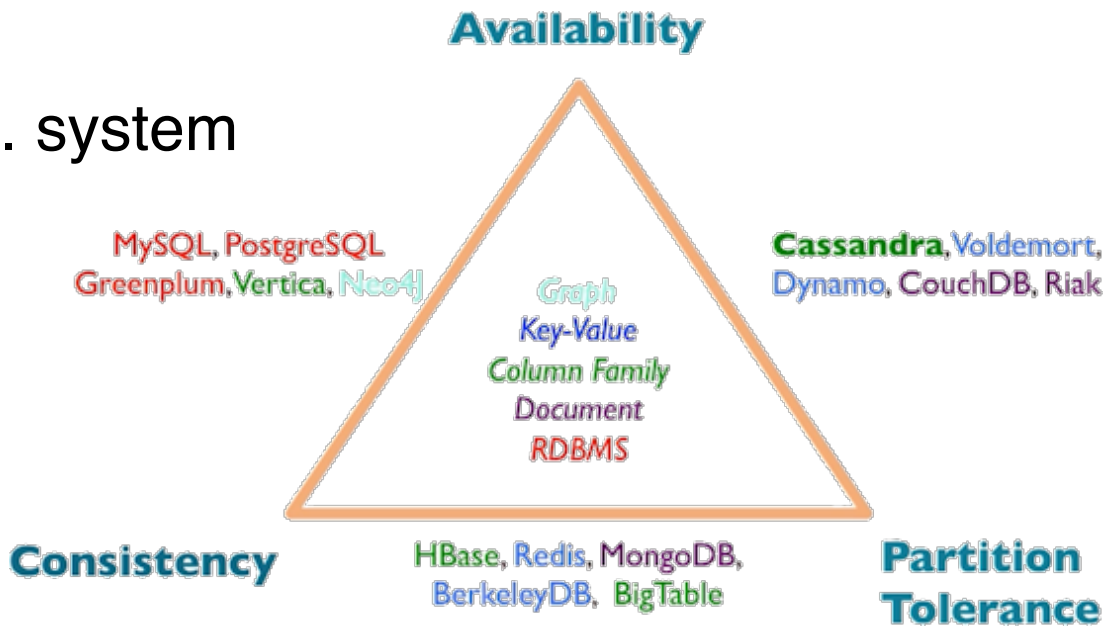
Availability vs. Consistency

- **Replication** is key for **availability** and essential in parallel/distributed DBs
- But, we would like to maintain replicated data **consistent**. Issues:
  - Atomic execution of update statement?
    - What if one of the systems that stores a copy is down?
  - You can use 3 replicas & read/write with 2+ replicas

- **Network partitions**: 2+ parts of a distr. system cannot communicate with each other

CAP theorem

  - In presence of **partitions**, no protocol ensure *both* **availability** and **consistency**
    - You need to decide which to ensure



Availability vs. Consistency

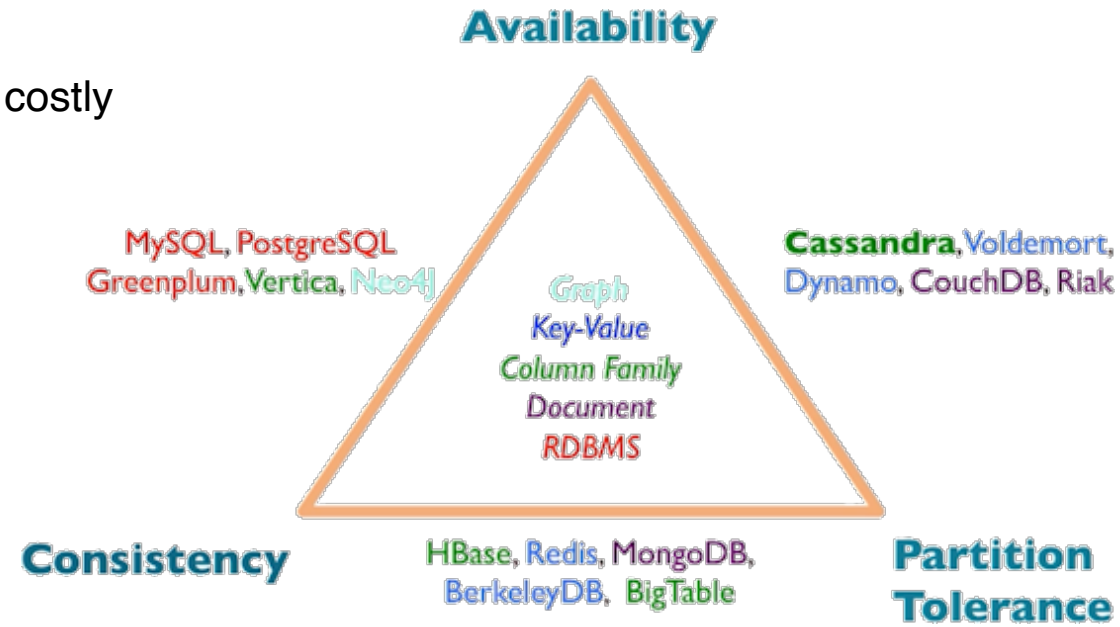
- **Network partitions**

CAP theorem

- In presence of **partitions**, no protocol ensure *both* **availability** and **consistency**

Really, **not a hard decision**
Consider *latency* instead of *availability*

- In practice, in large systems **partitions** cannot be prevented.
  - You need to decide which to ensure: *consistency* or *availability*
  - Traditional DBMSs choose **consistency**
  - For distributed DBMSs, ensuring **consistency** is costly
  - Many applications, willing to reach higher **availability** at the cost of consistency



Replication with Weak Consistency

- Key issues:
  - Some replicas may not get updated
    - Reads may get old versions
  - Different updates may be applied to different replicas
    - How to detect? Which is the newest version? How to resolve?

- Systems known as **BASE** (as opposed to ACID)

  Basically Available, Soft state, Eventual consistency
  - *Soft state*: copies of a data item may be inconsistent
  - *Eventually consistent*: copies allowed to become inconsistent, but after partitioning is resolved, eventually all copies become consistent

Beyond traditional RDBMS