SCALABLE WEB PROGRAMMING

CS193S - Jan Jannink - 2/02/10

Weekly Syllabus

1. Scalability: *(Jan.)*

2. Agile Practices

3. Ecology/Mashups

4. Browser/Client

5. Data/Server: *(Feb.)*

6. Security/Privacy

7. Analytics*

8. Cloud/Map-Reduce

9. Publish APIs: *(Mar.)**

10. Future

* assignment due

Data is the Core

* Maybe I should just go back and rename the course
* Data Storage, Access, Transport, Presentation
    * keep it generic
    * design for incremental system growth
    * avoid unbounded growth at any layer
        * duplicate elimination, query filtering

Data Storage

* Reliable Persistence
    * almost every other DB feature is overkill in web apps
* Simplicity/Genericity
    * avoid a system that grows more complex over time
* Recoverability
    * Backups are great but not the first line of defense

Data Scalability

* Data access spreading
    * Balance reads to writes
* Data set partitioning
    * Parallel Access
* Hot Spots
    * Data Caching, Randomizing Keys

Database Scalability

* Keep schemas ultra generic
    * consider storing all data in a single table
* Constraint management often works against availability
    * increases the number of query errors
* Caching is key
    * commonly accessed data accounts for majority of requests

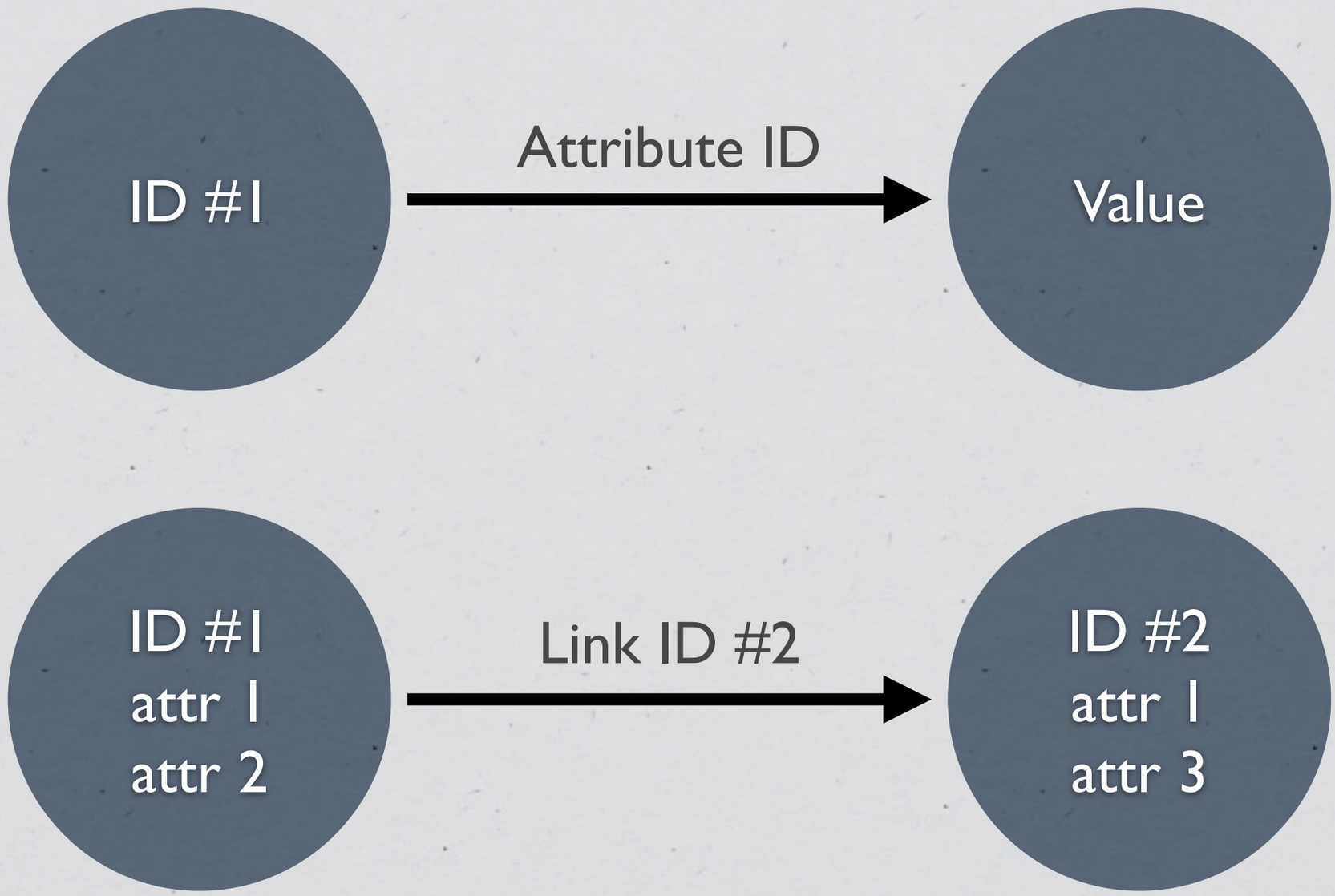
Flat File to Data Center

* Single table, single server
* Distributed in memory cache
* Master - Slave single master
* Table partitioning, multiple masters
* Read partitioning, multiple locations

Attribute Data Store

* Basic data tuples
* (ID, Content)
    * equivalent to a hashtable
* (ID1, ID2, Content)
    * complete for representation of semi structured data
    * similar to RDF data model

Attribute Graph Model



Attribute Model Benefits

* Trivial to manage objects
* Easy to repair broken constraints
* Trivial to partition tables
* Natural to support huge data graphs
* Automatically support every new feature
    * future proof

Drawbacks

* Schema does not guide query style
* Semantics buried in object and attribute definitions
* Need to encode these semantics in the server code
* Some advance planning needed for data path design

Agile Data

* Read/write ratio is near 10/1
* 80-20 access pattern
  * 20% of data accounts for 80% of access
* Construct pages from no more than 2 DB queries
  * reassess page or data design otherwise
* Future proof your design by not locking into a complex schema

Agile App Design

* Make the data path the core of the system
* Design data access API to allow different backends
  * ease transition to different clouds
* Centralize access methods into a few classes at most
  * simplify addition of an in memory cache

Rapid Prototyping vs. Scale

* Most sites are built front to back, UI first, back end last
    * pressure to demo by investors
    * we know better what we can see in front of us
* Ruby on Rail ‘magically’ generates DB schemas
    * gets apps out the door fast
    * difficult to start from data centered design

Extreme Programming Conundrum

* Main Principle: don't design more than immediate needs
* Main Caveat: don't make the same mistake twice
* Main Compromise
  * don't build more than what you need
  * learn how to design minimalist systems that don't dead end

Twitter Example

* Basic idea: put IM status on the web
  * extreme case of long tail data access
* Largest Ruby on Rails system
  * scheduled downtime
  * limited feature growth
  * data access APIs are all throttled

Agile Cache Design

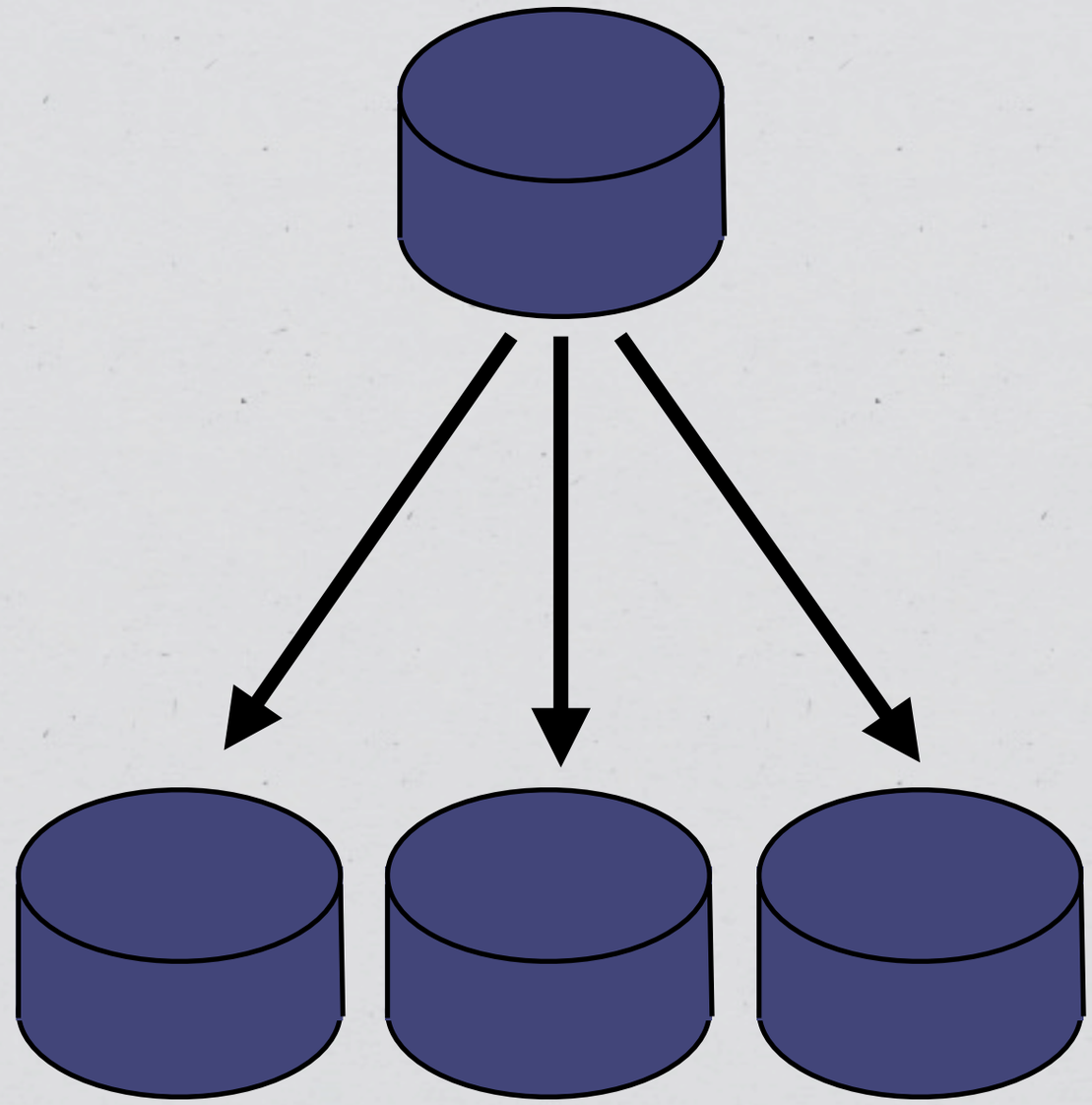
* Store objects either as raw DB rows or as server objects (or both)
    * use ID as key
    * optimize for access pattern
    * read only => DB rows, frequent updates => server objects
* Store entire query results too
    * use query string or hash as key

Agile Parallelization

* Worth starting at the Webserver level
* round robin routing is usually sufficient
* lock users to a given server
* associate closely linked content to closer web servers
* extend CDN (Content Delivery Network) concept

Master Slave DB Concepts

* Start with one DB server
    * about 10 reads per write
* Add extra DBs
    * writes copied by log file
* End with 10 identical DBs
    * 1 read per write at full load



New Frontier: Autopartition

* Route queries to DB servers by key
* When Server reaches access or query speed threshold
  * Bring up standby DB servers
    * Copy tables
    * Split DB key space evenly
  * Update DB client routing table

Worth Checking Out

* Memcached
  * http://memcached.org/
* MySQL replication
  * http://dev.mysql.com/doc/refman/5.5/en/replication.html
* RDF
  * http://www.w3.org/TR/rdf-concepts/

Q & A Topics

* Data Loss, Downtime, Backups
* Index and query optimizing
    * when to do it
* Other architectures
    * document oriented DBs
    * column oriented DBs