GOODBYE SQL

Master Program in Computer Science
University of Calabria

Prof. F. Ricca

From Web 1.0 to Web 2.0

- The Web 1.0
  - A collection of statically linked HTML pages
  - No database involved *at all*
- The Web 2.0
  - The three layer architectures
    - Client – Application Server – Database
    - Application Server
      - Easy to scale and distribute
    - Database server
      - **Usually the bottleneck for large systems**

Scale up vs Scale out

- Scale up
  - To **scale up (vertically)** means to add resources to a single node in a system
  - Typically involving the addition of CPUs or memory to a single computer
- Scale out
  - To **scale out (horizontally)** means to add more nodes to a system
  - Adding a new computer to a distributed software application

- Standard DBMS in 2005
  - More performance

    -> Buy a more powerful machine and more "efficient" DBMS
  - i.e., Scale up

First tries to Scale out

- Architectural changes
  - Memcached Servers
    - avoid database access as much is possible.
    - provides a distributed object cache
    - cache an object-oriented representation of database information across many servers
  - Replication
    - allows changes to one database to be copied to another database
    - *read requests* could be directed to any one of these replica databases
    - *write operations* still had to go to the master database

- *Both are good only if reads significantly outnumber writes*

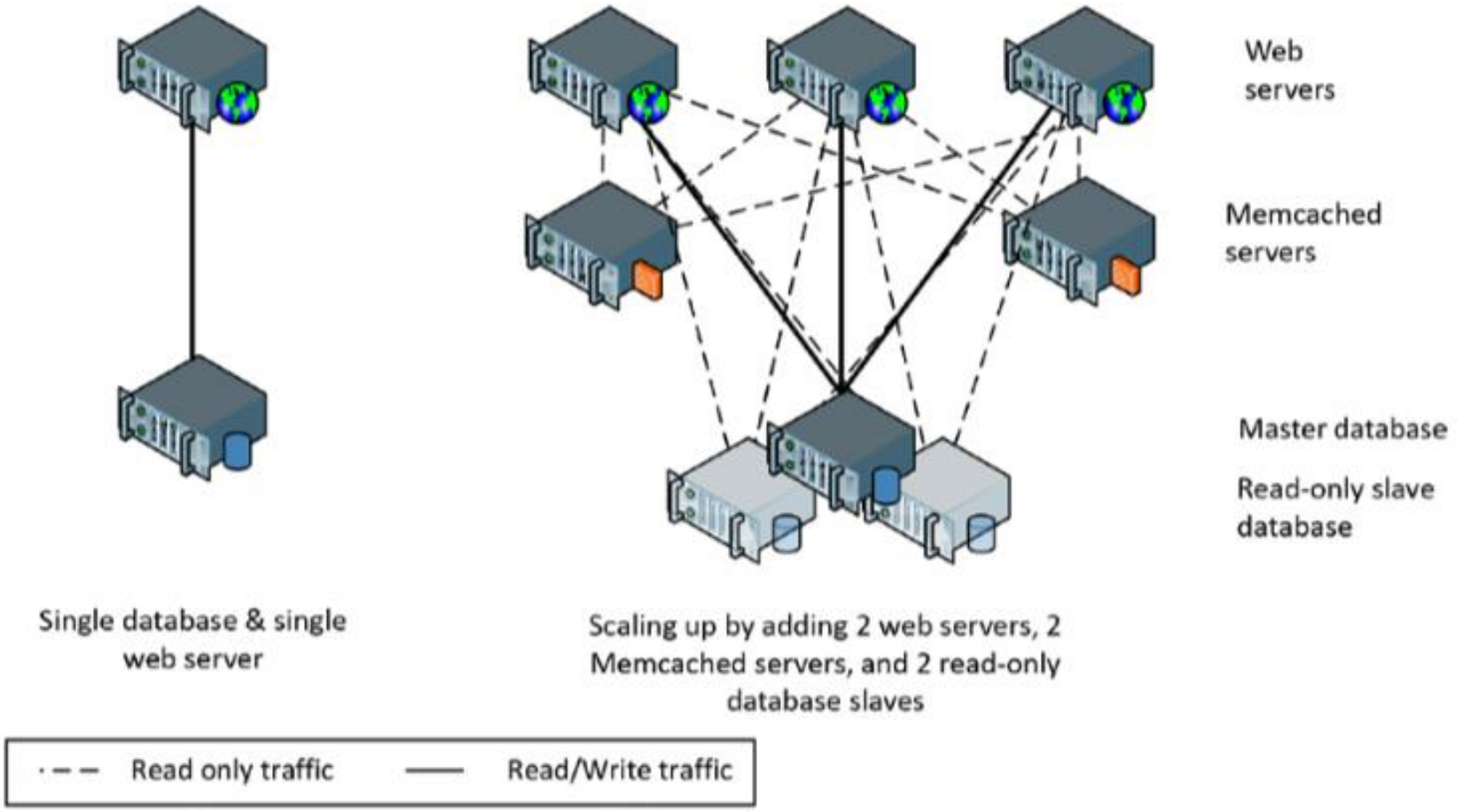


Figure 3-1. *Scaling up with Memcached servers and replication*

Sharding to write in large scale

- A database is partitioned across multiple physical servers
  - The largest tables are partitioned horizontally
  - E.g., partitioning is based on a Key Value, such as a user ID
- What is a Shard?
  - A database partition
- How it works?
  - The application must determine which shard will contain the data
  - Then send the SQL to the appropriate server
- Pro and contra:
  - Sharding is simple in concept but
  - The application logic must understands the location of data
  - **Requests accessing more than one shard need complex coding**

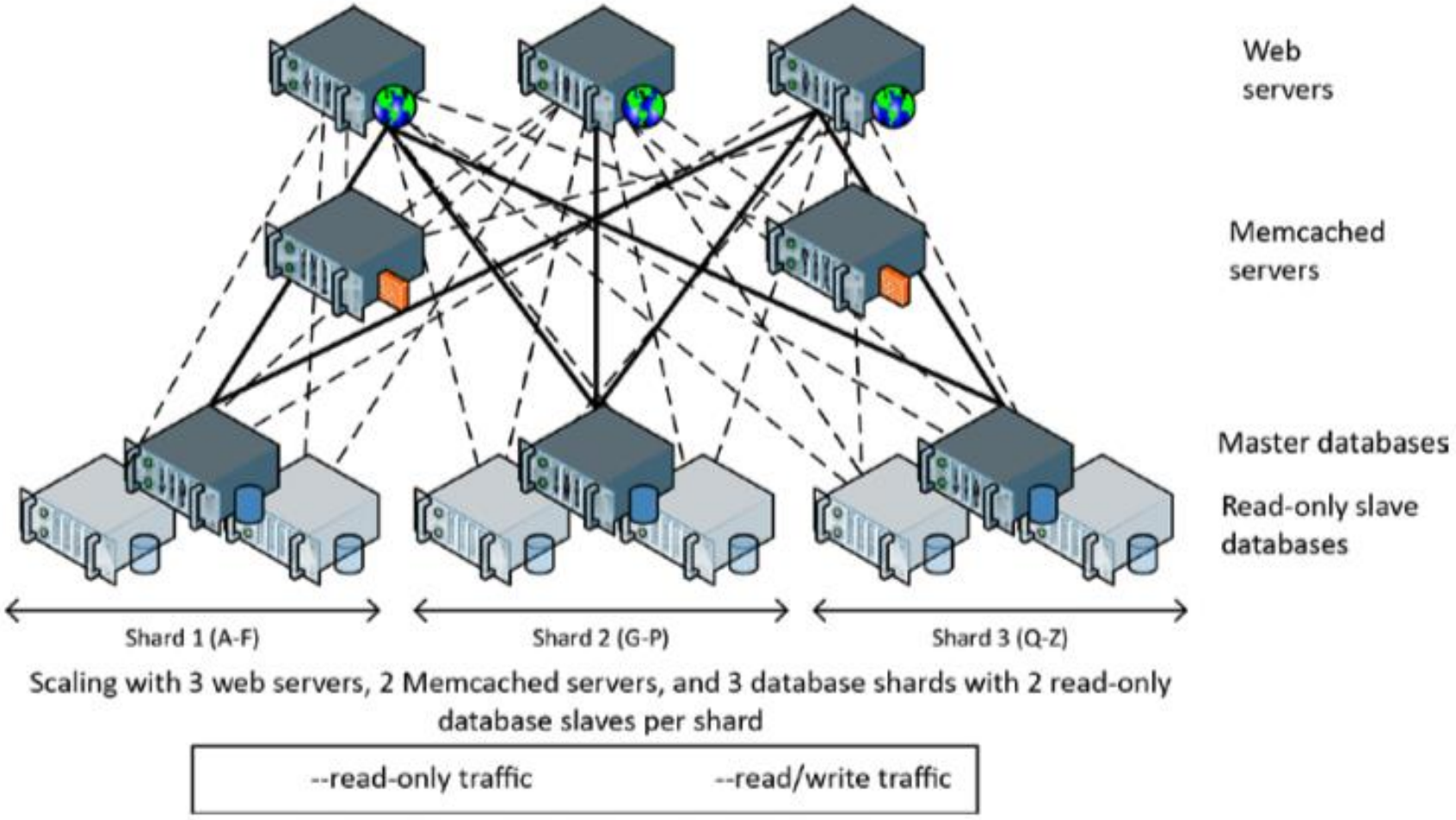


Figure 3-2. *Memcached/replication architecture from Figure 3-1, with sharding added*

Death by a Thousand Shards

- the operational costs of sharding are huge
  - **Application complexity**
    - i.e., complex code being required to maintain the system
  - **Crippled SQL**
    - it is not possible to issue a SQL statement that operates across shards
    - only programmers can query the database as a whole
  - **Loss of transactional integrity**
    - ACID transactions against multiple shards are not possible
    - The *Two Phase Commit Protocol* can create bottlenecks
  - **Operational complexity**
    - Load balancing across shards becomes extremely problematic
    - Adding new shards requires a complex rebalancing of data
    - Transitory inconsistencies in the schema are possible

Users and alternatives to sharding

- Early adopters of sharding: Facebook, twitter
  - FB In 2011 used 4k shards of MySQL with 9k Memcached servers
  - 1.4 billion reads/sec – 3.5 million row chenges/second

- Alternatives by ORACLE
  - *Real Application Clusters* (*RAC*)
    - ***transparently scalable, ACID compliant, relational cluster***
    - each database node works with data located on shared storage devices
    - *shared disk* clustering vs *shared nothing* clusters
    - a sort of distributed memory cache iacross database nodes
    - it failed as an alternative to the MySQL sharded model
      - it was too expensive
      - **Maybe a try to get something "impossible"**

The CAP Theorem

In a distributed system you have at most only two of Consistency, Availability, and Partition tolerance

- Consistency
  - Any user has an identical view of the system at any given instant
- Availability
  - In the event of a failure, the system remains operational
- Partition Tolerance
  - Maintain operations in the event of the network failure

A conjecture become a Theorem

- Conjuctured by Brewer in 2000
  - Eric A. Brewer:
    Towards robust distributed systems (abstract). PODC 2000: 7
- Proved formally in 2004 by Gilbert and Lynch
  - Seth Gilbert, Nancy Lynch
    Brewers Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

CAP by example (1)

- This example is by Kaushik Sathupadi
  - Available at: http://ksat.me/a-plain-english-introduction-to-cap-theorem/

- **Chapter 1: "Remembrance Inc" Your new venture :**
  - Remembrance Inc! - Never forget, even without remembering!

    - Customer : Hey, Can you store my neighbor's birthday?
    - You: Sure.. when is it?
    - Customer : 2nd of jan
    - You: (write it down against the customer's page in your paper note book) Stored. Call us any time for knowing your neighbor's birthday again!
    - Customer : Thank you! You: No problem! We charged your credit card with $0.1

CAP by example (2)

- **Chapter 2 : You scale up:**
  - You and your wife both get an extension phone
  - Customers still dial (555)–55-REMEM and need to remember only one number
  - A pbx will route the a customers call to whoever is free and equally
- **Chapter 3 : You have your first "Bad Service"**
  - Jhon: Hey
  - You: Glad you called "Remembrance Inc!". What can I do for you?
  - Jhon: Can you tell me when is my flight to New Delhi?
  - You: Sure.. 1 sec sir
    (You look up your notebook)
    (wow! there is no entry for "flight date" in Jhon's page)!!!!!
  - You: Sir, I think there is a mistake. You never told us about your flight to delhi
  - Jhon: What! I just called you guys yesterday!(cuts the call!)

CAP by example (3)

- How did that happen?
  - Could John's call yesterday reached your wife?
  - Sure enough it’s there.
  - **Your distributed system is not consistent!**
- **Chapter 4: You fix the Consistency problem:**
  - Whenever any one of us get a call for an update (when the customer wants us to remember something) before completing the call we tell the other person
  - This way both of us note down any updates
  - When there is call for search we don’t need to talk with the other person
  - Since both of us have the latest updated information in both of our note books we can just refer to it.

CAP by example (4)

- This is sure, right?
  - BUT: an “update” request has to involve both and no work in parallel during write
  - **Availability problem!**
- **Chapter 5: You come up with the greatest solution Ever:**
  - "This is what we can do to be consistent and available"
    - i) Whenever any one of us get a call for an update before completing the call, if the other person is available we tell the other person.
      This way both of us note down any updates
    - ii) But if the other person is not available we send the other person an email about the update
    - iii) The next day when the other person comes to work after taking a day off, He first goes through all the emails, updates his note book accordingly.. before taking his first call

CAP by example (5)

- Consistent and available, right?
  - BUT: what if both of you report to work and one of you doesn't update the other person?
    - You can decide to be partition tolerant by deciding not to take any calls until you patch up with your wife
  - You lost **Partition Tolerance!**
- **Chapter 7: Conclusion:**
  - You can get cannot achieve all three of Consistency, Availability and Partition tolerance.
    - Consistency: You customers, once they have updated information with you, will always get the most updated information when they call subsequently. No matter how quickly they call back
    - Availability: Remembrance Inc will always be available for calls until any one of you (you or your wife) report to work.
    - Partition Tolerance: Remembrance Inc will work even if there is a communication loss between you and your wife!

Eventual consistency

- *This relates more to performance than to availability*
- **For banks consistency is all**
  - They will pay the cost
- **For other globally distributed system is not!**
  - ACID transactions were increasingly untenable in large-scale distributed websites
- For Facebook, Twitter, etc
  - Worldwide synchronous consistency is unnecessary
    - *Does matter if my friend in Australia can see my tweet a few seconds before my friend in America?*
  - Eventual consistency (updates in background, some latency)
    - a key characteristic of many NoSQL databases
    - was implemented in Amazon's *Dynamo* key-value store

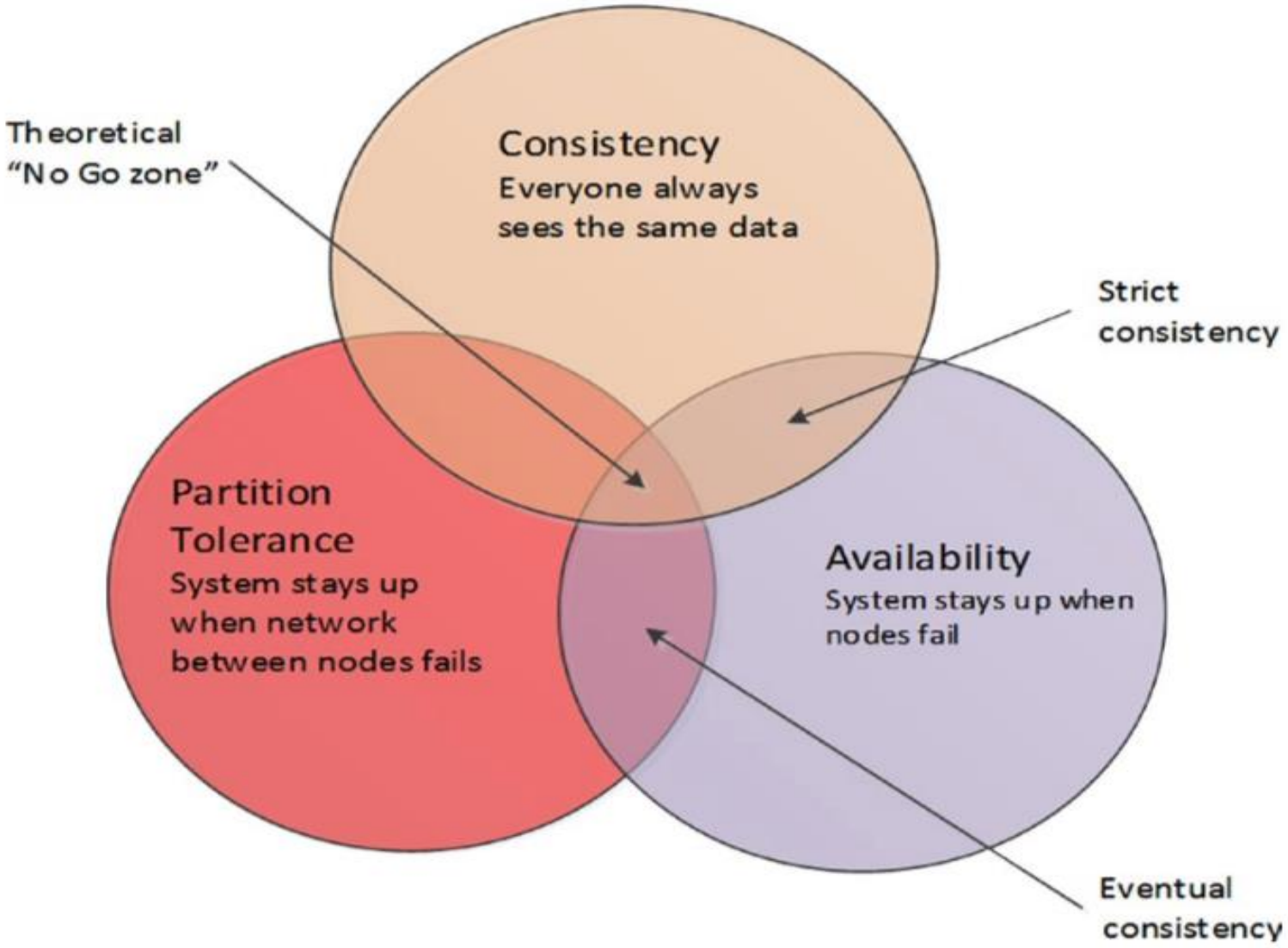


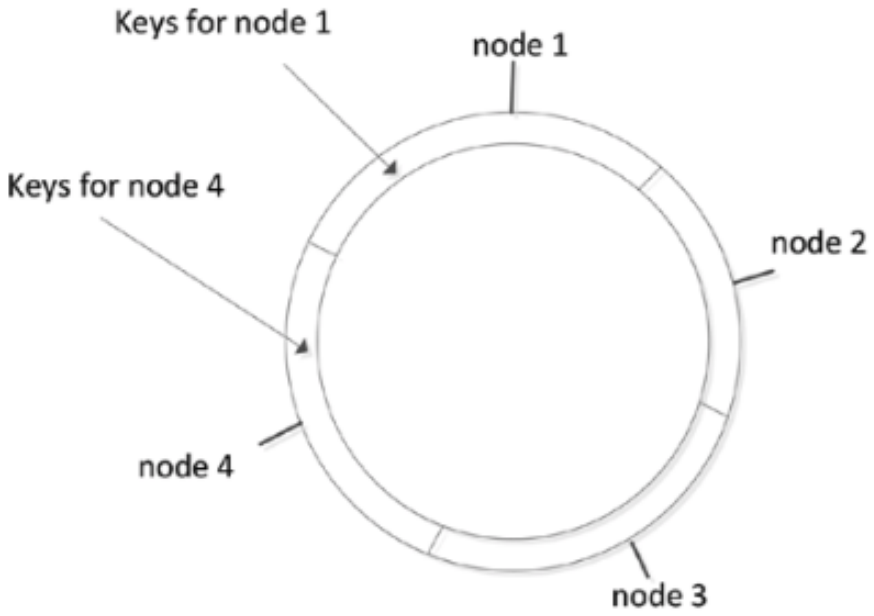
Figure 3-4. *Dynamo and ACID RDBMS mapped to CAP theorem limits*

Amazon's Dynamo

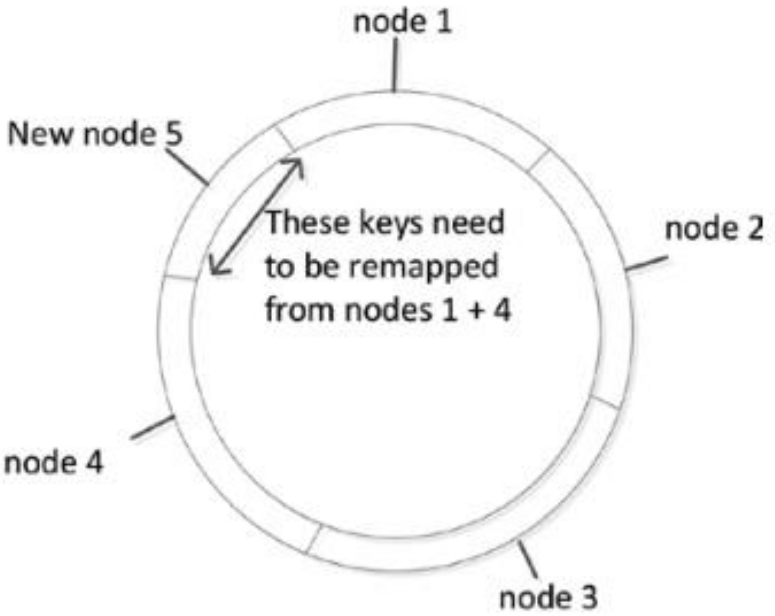
- An alternative non relational database
  - the data store need only support primary key-based access
  - the values retrieved by a key lookup would be unstructured binary objects
  - Most objects would be small, under 1 MB
- Dynamo would allow (though not require) consistency to be sacrificed to Availability and Partition Tolerance
  - Think about Amazon's business and you will understand why!!
- Key Architectural features
  - **Consistent hashing**
  - **Tunable consistency**
  - **Data versioning**

Dynamo main features

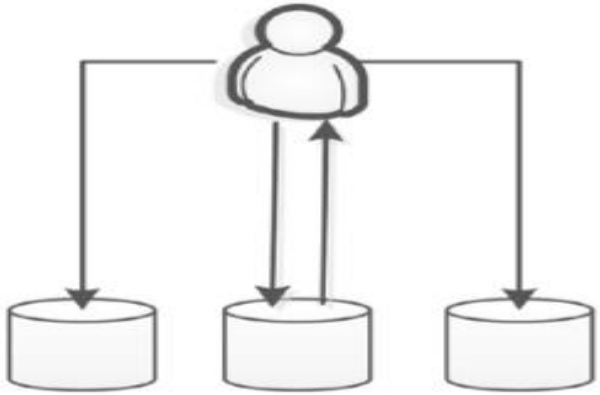
- **Consistent hashing**:
  - a scheme that uses the hash value of the primary key to determine the nodes in the cluster responsible for that key
  - allows nodes to be added or removed from the cluster with minimal rebalancing overhead
- **Tunable consistency**:
  - The application can specify trade-offs between consistency, read performance, and write performance
- **Data versioning**:
  - Since write operations will never be blocked, it is possible that there will be multiple versions of an object in the system.
  - They will need to be resolved by the application or user.
  - For instance, if the buyer updates his or her shopping cart from two computers, the resulting cart may have duplicate items that he or she may need to remove.



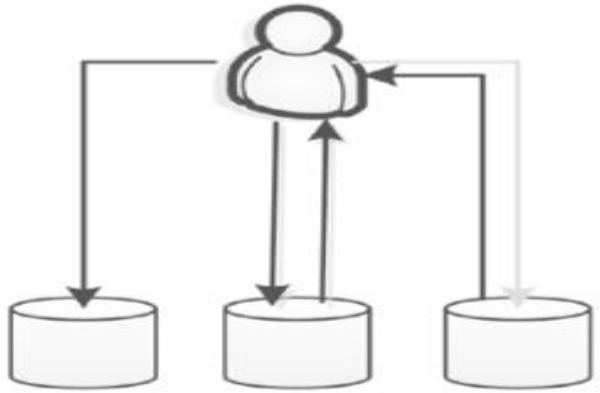
Split of keys in 4-node cluster



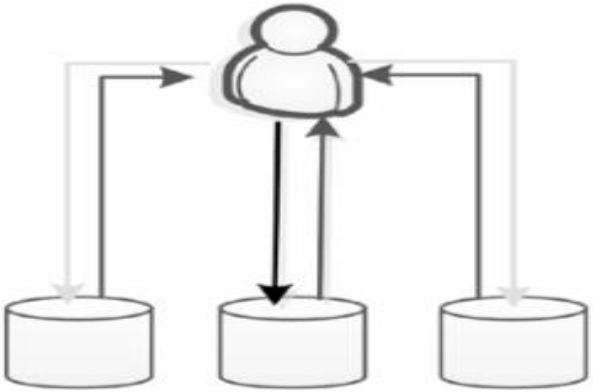
Adding a new node to the cluster



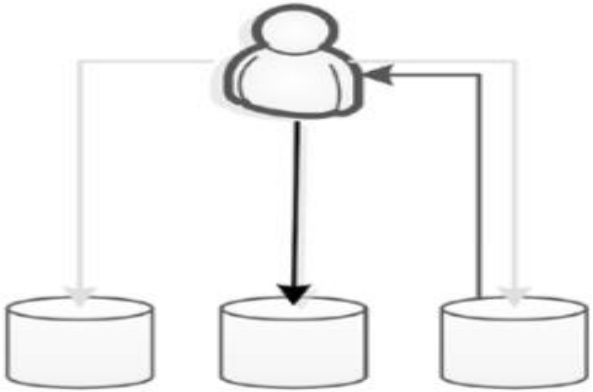
N=3 W=3 R=1
Slow writes, fast reads, consistent
There will be 3 copies of the data.
A write request only returns when all 3 have written to disk.
A read request only needs to read one version.



N=3 W=2 R=2
Faster writes, still consistent (quorum assembly)
There will be 3 copies of the data.
A write request returns when 2 copies are written – the other can happen later.
A read request reads 2 copies make sure it has the latest version.



N=3 W=1 R=N
Fastest write, slow but consistent reads



N=3 W=1 R=1
Fast, but not consistent
There will be 3 copies of the data.

No SQL databases

- The founding paper:
  - Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels Dynamo: Amazon's Highly Available Key-value Store
    - http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
- Amazon's DynamoDB inspired the first no SQL DBs
  - Riak
  - LinkedIn's Voldemort
  - Cassandra
  - …
- They enforce no structure on their payload
  - **This makes them impenetrable to all but programmers**

Document Databases

A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers

—Object-oriented data community, mid-1990s

- A document database is a nonrelational database that stores data as structured documents
  - Mostly in XML or JSON formats
  - Document databases are free to implement ACID transactions
  - The dominant document databases provide relatively modest transactional support
  - **a happy medium between the rigid schema of the relational database and the completely schema-less key-value stores**
    - Started to become influent in 2008

XML and XML Databases

- The first document databases were XML-DBs
- XML Tools and Standards
  - **XPath**: A syntax for retrieving specific elements from an XML
  - **XQuery**: *"SQL for XML"* A query language for interrogating XML
  - **XML schema**: A special type of XML document that describes the elements that may be present in a specified
  - **XSLT** (**Extensible Stylesheet Language Transformations**): A language for transforming XML documents into alternative formats, including non-XML formats such as HTML
  - **DOM** (**Document Object Model**): An object-oriented API that programs can use to interact with XML, XHTML

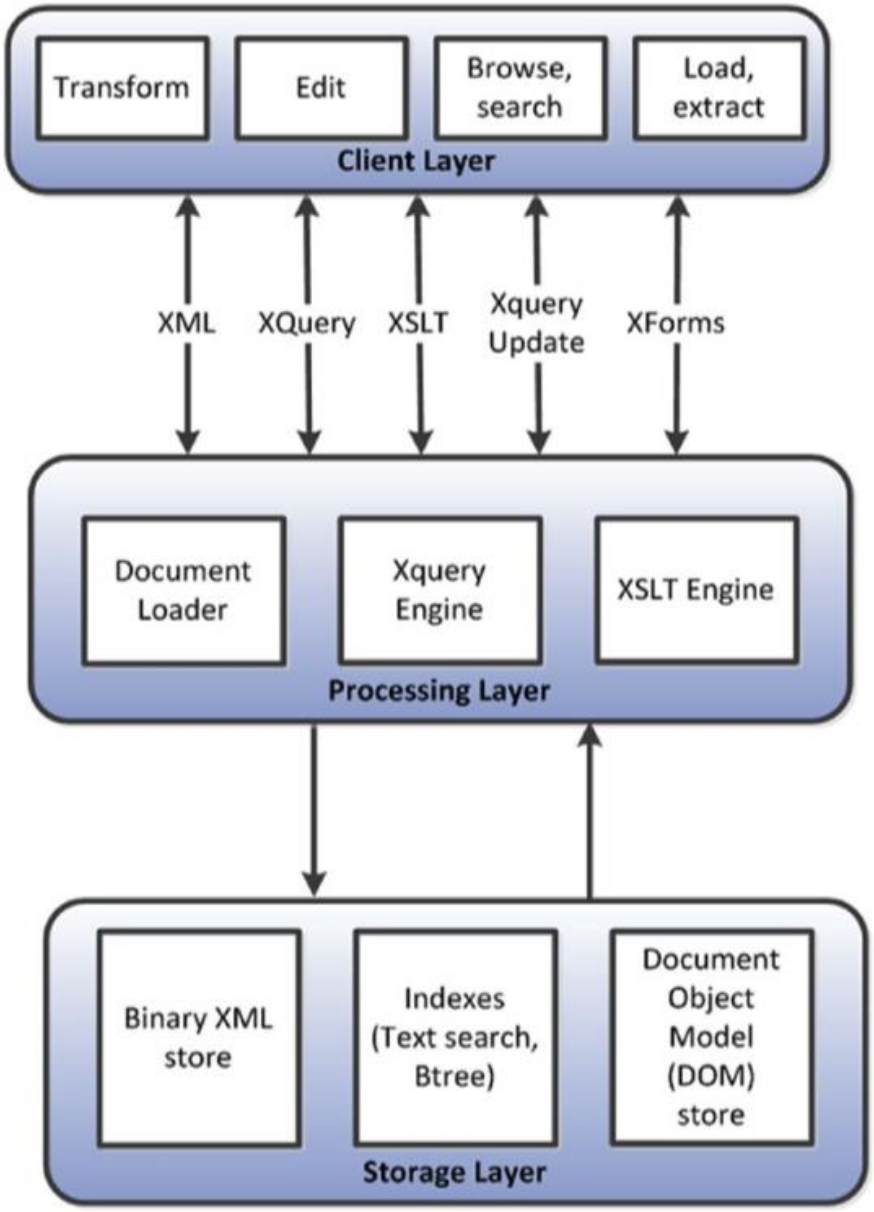


Figure 4-2. *Generic XML database architecture*

XML Databases and DBMS with XML

- XML databases in 2000s
  - They were not positioned as alternatives to the RDBMS
  - A means of managing the XML documents
- Two of the more significant XML
  - *eXist* and *MarkLogic*
- Relational database vendors all introduced XML support
  - Allowing XML documents to be stored within long/BLOB columns and
  - Providing support for the various XML standards
  - XML support was added to the ANSI SQL definition (SQL/XML)
  - Support for these SQL extensions in
    - Oracle, Postgres, DB2, MySQL, and SQL Server

JSON vs XML

- XML problems
  - XML is justifiably criticized as being wasteful of space and computationally expensive to process
  - XML tags are verbose and repetitious, typically increasing the amount of storage required by several factors
  - XML has long been the predominant format for structured files, but
- The diffusion of the *JavaScript Object Notation* (*JSON*)
  - Mostly support web-based operational workloads
  - JSON was created by JavaScript pioneer Douglas Crockford
    - JSON was deliberately intended to be a lightweight substitute for XML
    - Replaced XML in *AJAX (Asynchronous JavaScript And XML)*
  - Can remove the need of ORM in web applications!

JSON Databases

- There is not a single specification of a JSON database.
- In a JSON document database
  - A *document* is the basic unit of storage (corresponding approximately to a row in an RDBMS)
    - A document comprises one or more key-value pairs,
    - May also contain nested documents and arrays
      - Arrays may also contain documents (complex hierarchical structures)
  - A *collection (*or *data bucket)*
    - *A set of documents corresponding to a relational table*
    - Documents in a collection don't have to be of the same type
    - Represents a common category of information
- A smaller number of collections w.r.t. RDBs
  - Nested documents representing master-detail relationships

Figure 4-4. *Simple Relational movie database*

```
{ "_id" : 97, "Title" : "BRIDE INTRIGUE",
              "Category" : "Action",
    "Actors" :
      [ { "actorId" : 65, "Name" : "ANGELA HUDSON" } ]
}

{ "_id": 115,"Title": "CAMPUS REMEMBER",
             "Category": "Action",
    "Actors" :
        [
          { "actorId": 8,"Name": "MATTHEW JOHANSSON" },
          { "actorId": 45,"Name": "REESE KILMER" },
          { "actorId": 168,"Name": "WILL WILSON" }
        ]
}

{ "_id" : 105, "Title" : "BULL SHAWSHANK",
               "Category" : "Action",
     "Actors" :
       [ { "actorId" : 2, "Name" : "NICK WAHLBERG" },
         { "actorId" : 23, "Name" : "SANDRA KILMER" } ]
}
```

Film Document

Actor document

Array of actors

Collection

Figure 4-5. *JSON movie collection*

The document model

- JSON Databases relax the first normal form
- Theoretically similar to Nested Relational Schemas
- Are redundant to be efficient
  - Like some join queries are "materialized" in the data
  - No equivalent of third normal form that defines a “correct” model

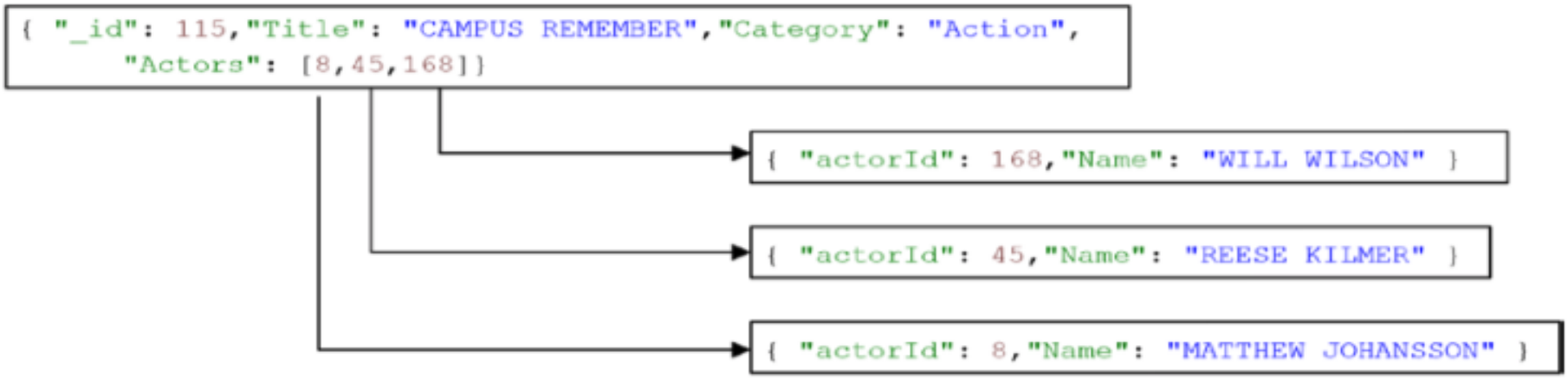


Figure 4-6. *Linking documents in a document database*

Some JSON Databases

- Early JSON Databases
  - CouchDB, created by Damien Katz (the first notable JSON-based database)
    - First supported XML then JSON
    - Hadoop, a (JavaScript-based) MapReduce,
    - Eventual consistency and multiple versioning model
- MemBase and CouchBase
  - Membase: a persistent variation on *Memcached* framework
    - Farmville online game
  - CouchBase (Extension of CouchDB)
- MongoDB (started in 2007)
  - a scalable and elastic data storage engine
  - internally it uses a binary encoded variant of JSON called BSON.
  - JavaScript-based query capability that is reasonably easy to learn
    - Query Language of the same power of Nested Relational Algebra
    - MongoDB established a strong lead in the NoSQL database
- Today ORACLE provides a document database!