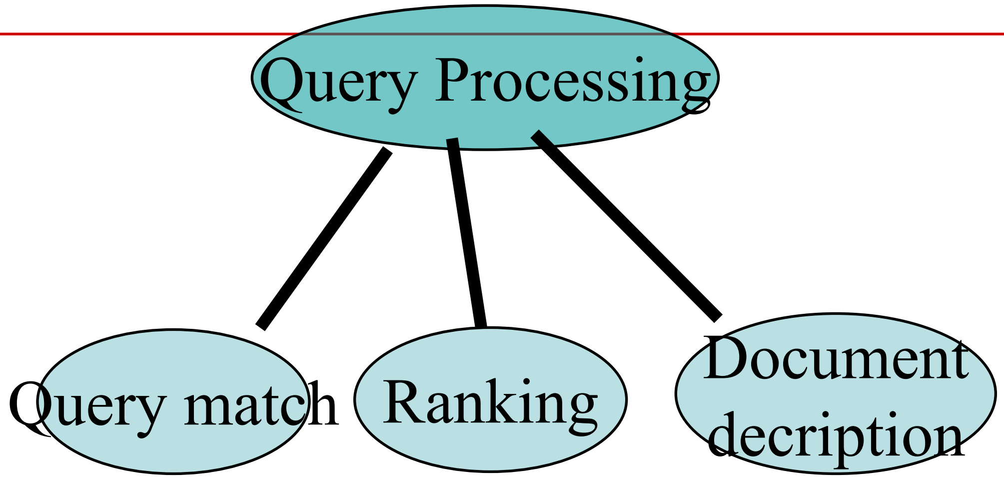
Design Tradeoffs in Query Processing and Online Architectures

- T. Yang 293S 2017

Content

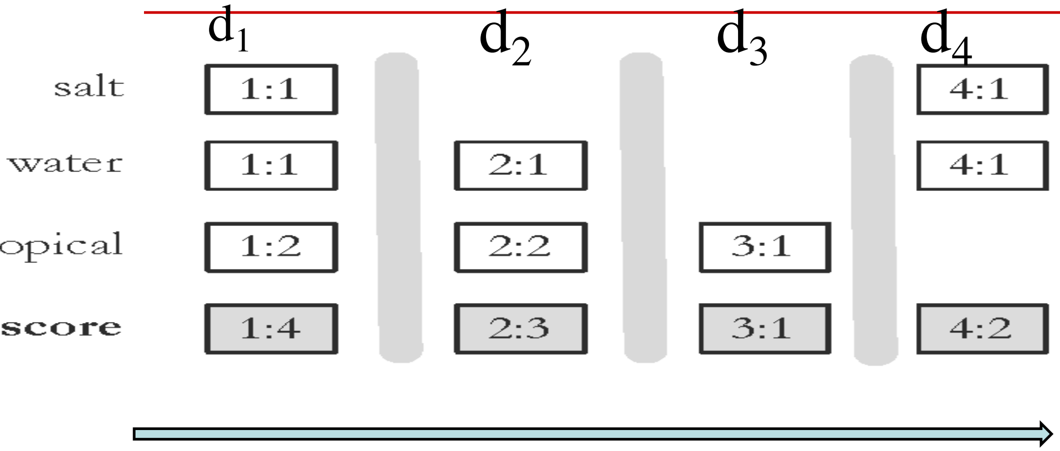
- **Example of design tradeoffs in query processing optimization**
- **Experience with Ask.com online architecture**
  - Service programming with Neptune.
  - Zookeeper

Query Processing



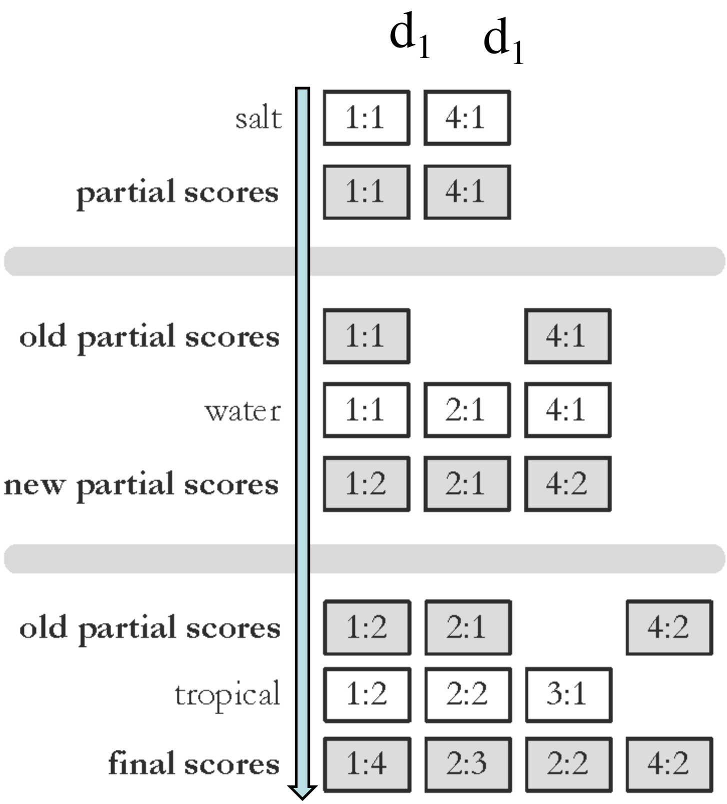
- **Query match to search a document set**
  - Document-at-a-time
    - Calculates complete scores for documents by processing all term lists, one document at a time
  - Term-at-a-time
    - Accumulates scores for documents by processing term lists one at a time

Document-At-A-Time vs Term-At-A-Time



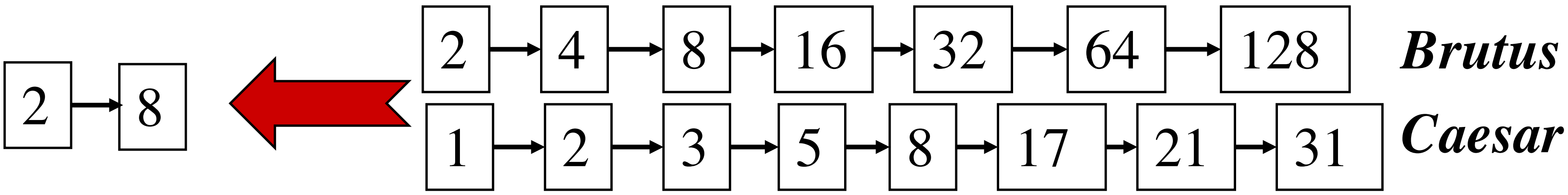
Term-at-a-time uses more memory for accumulators, data access is more efficient.

Less parallelism to exploit for parallel query processing

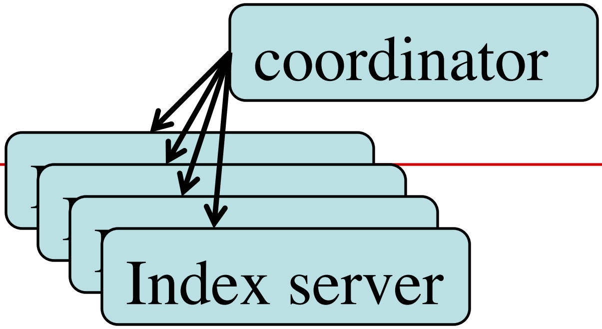


Tradeoff for shorter response time

- **Early termination of faster query processing**
  - Ignore lower priority documents at end of lists in doc-at-a-time
- **List ordering**
  - order inverted lists by quality metric (e.g., PageRank) or by partial score
  - makes unsafe (and fast) optimizations more likely to produce good documents
  - What about document ID ordering?

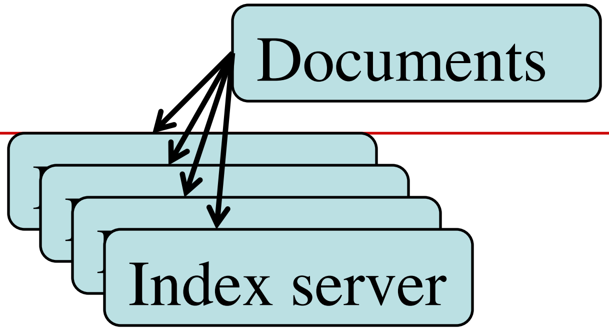


Distributed Matching



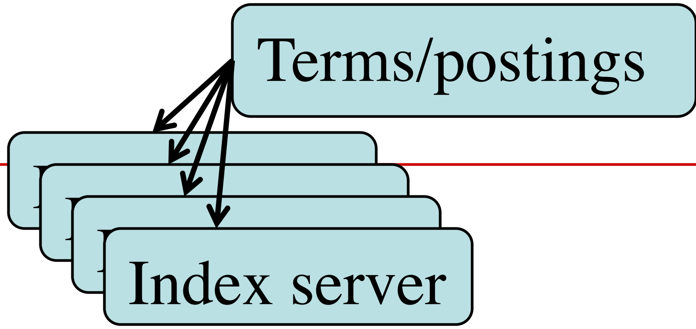
- **Basic process**
  - All queries sent to a *coordination machine*
  - The coordinator then sends messages to many *index servers*
  - Each index server does some portion of the query processing
  - The coordinator organizes the results and returns them to the user
- **Two main approaches**
  - Document distribution
    - by far the most popular
  - Term distribution

Distributed Evaluation



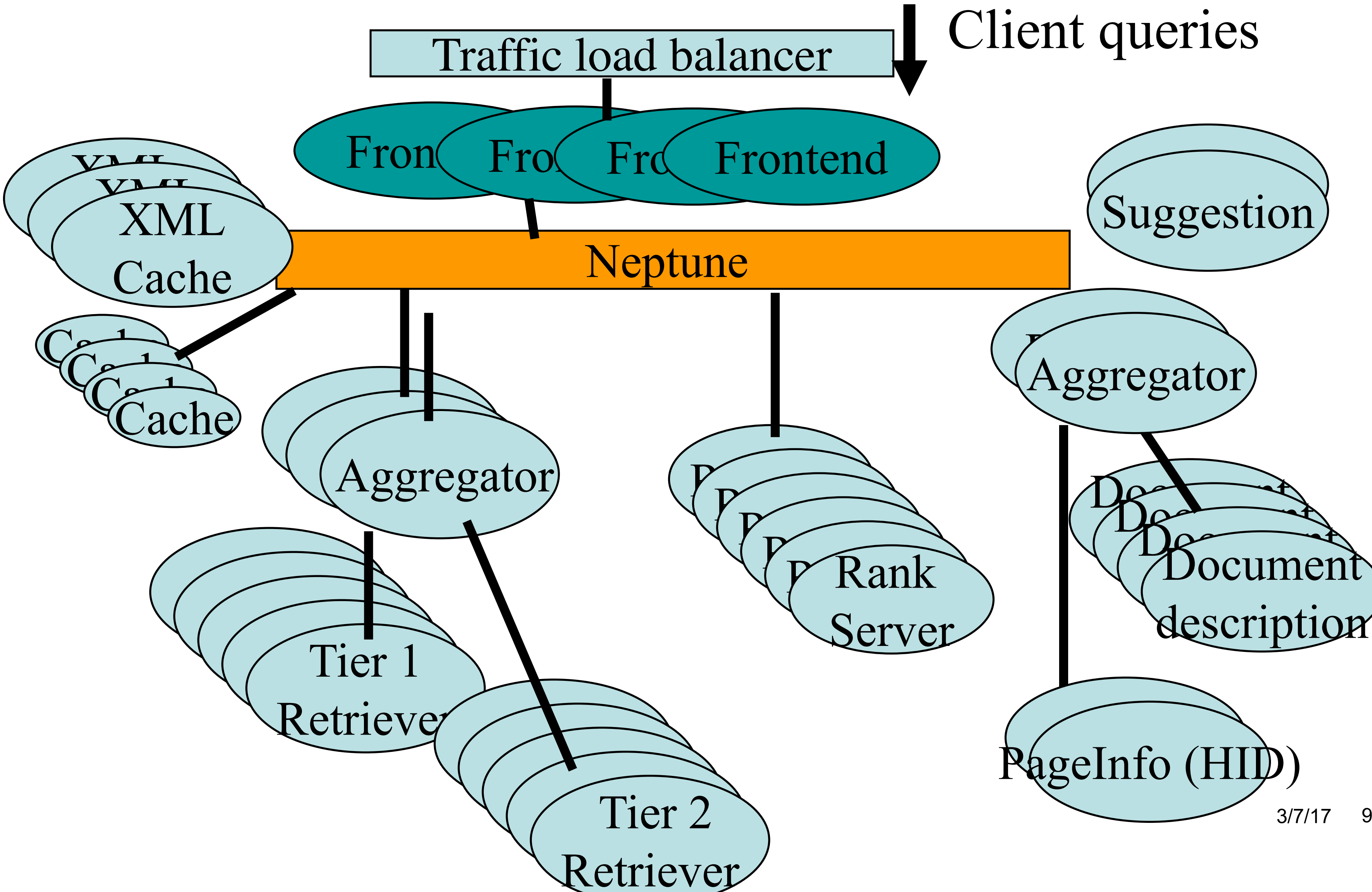
- **Document distribution**
  - Each index server acts as a search engine for a small fraction of the total collection
  - A coordinator sends a copy of the query to each of the index servers, each of which returns the top-*k* results
  - Results are merged into a single ranked list by the coordinator

Term-based distribution

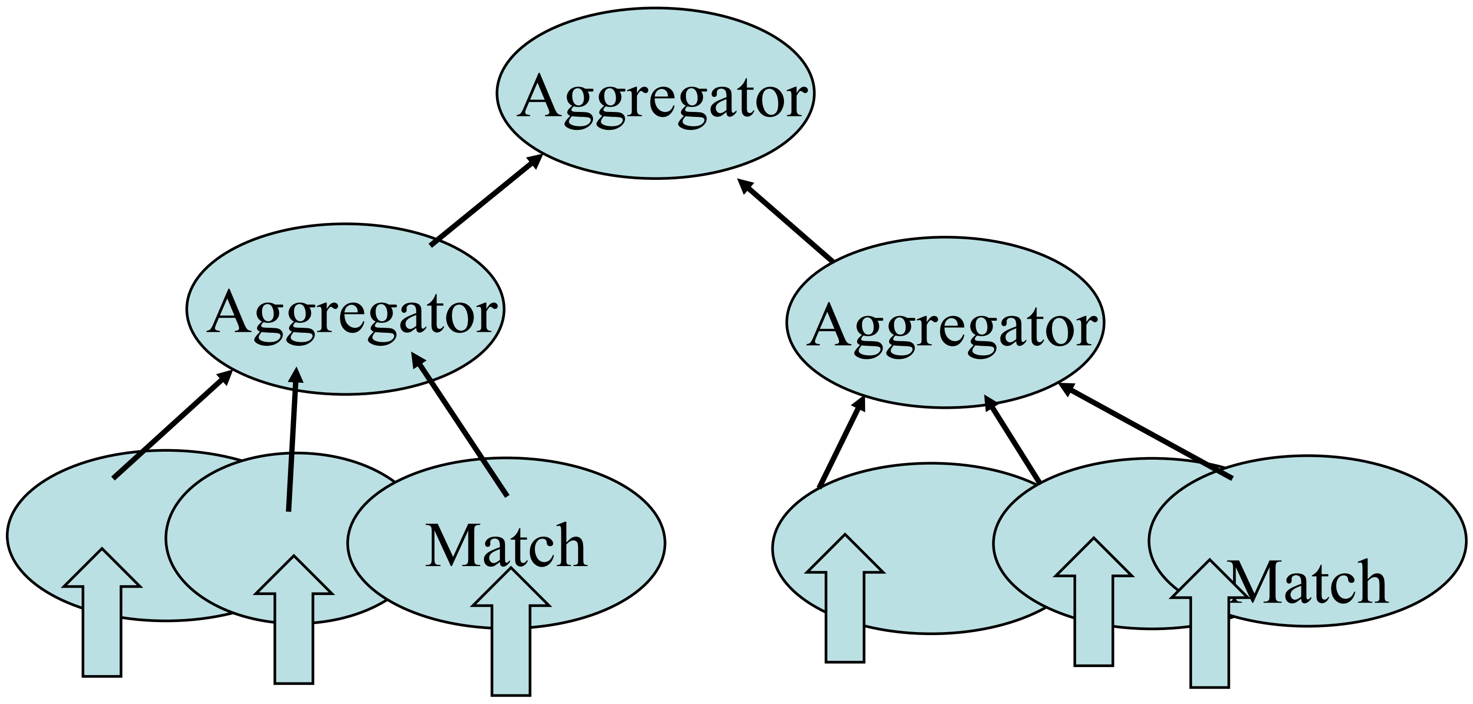


- **Single index is built for the whole cluster of machines**
- **Each inverted list in that index is then assigned to one index server**
  - in most cases the data to process a query is not stored on a single machine
- **One of the index servers is chosen to process the query**
  - usually the one holding the longest inverted list
- **Other index servers send information to that server**
- **Final results sent to director**

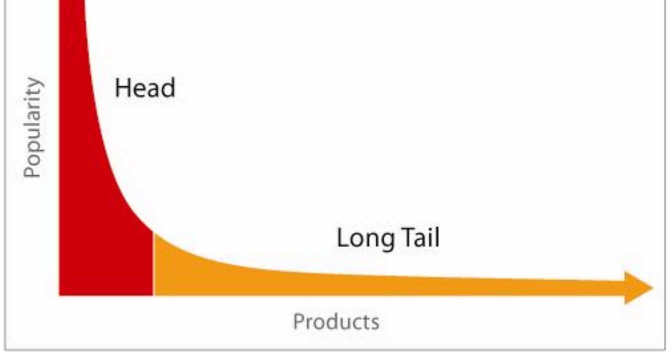
Ask.com Search Engine



Multi-tier aggregation for continus query stream processing



Frontends and Cache



Graph from http://www.longtail.com/about.html

- **Front-ends**
  - Receive web queries.
  - Direct queries through XML cache, compressed result cache, database retriever aggregators, page clustering/ranking,
  - Then present results to clients (XML).
- **XML cache :**
  - Save previously-queried search results (dynamic Web content).
  - Use these results to answer new queries. Speedup result computation by avoiding content regeneration
- **Result cache**
  - Contain all matched URLs for a query.
  - Given a query, find desired part of saved results. Frontends need to fetch description for each URL to compose the final XML result.

Index Matching and Ranking

- **Retriever aggregators (Index match coordinator)**
  - Gather results from online database partitions.
  - Select proper partitions for different customers.
- **Index database retrievers**
  - Locate pages relevant to query keywords.
  - Select popular and relevant pages first.
  - Cache popular index
- **Ranking server**
  - Classify pages into topics & Rank pages
- **Snippet aggregators**
  - Combine descriptions of URLs from different description servers.
- **Dynamic snippet servers**
  - Extract proper description for a given URL.

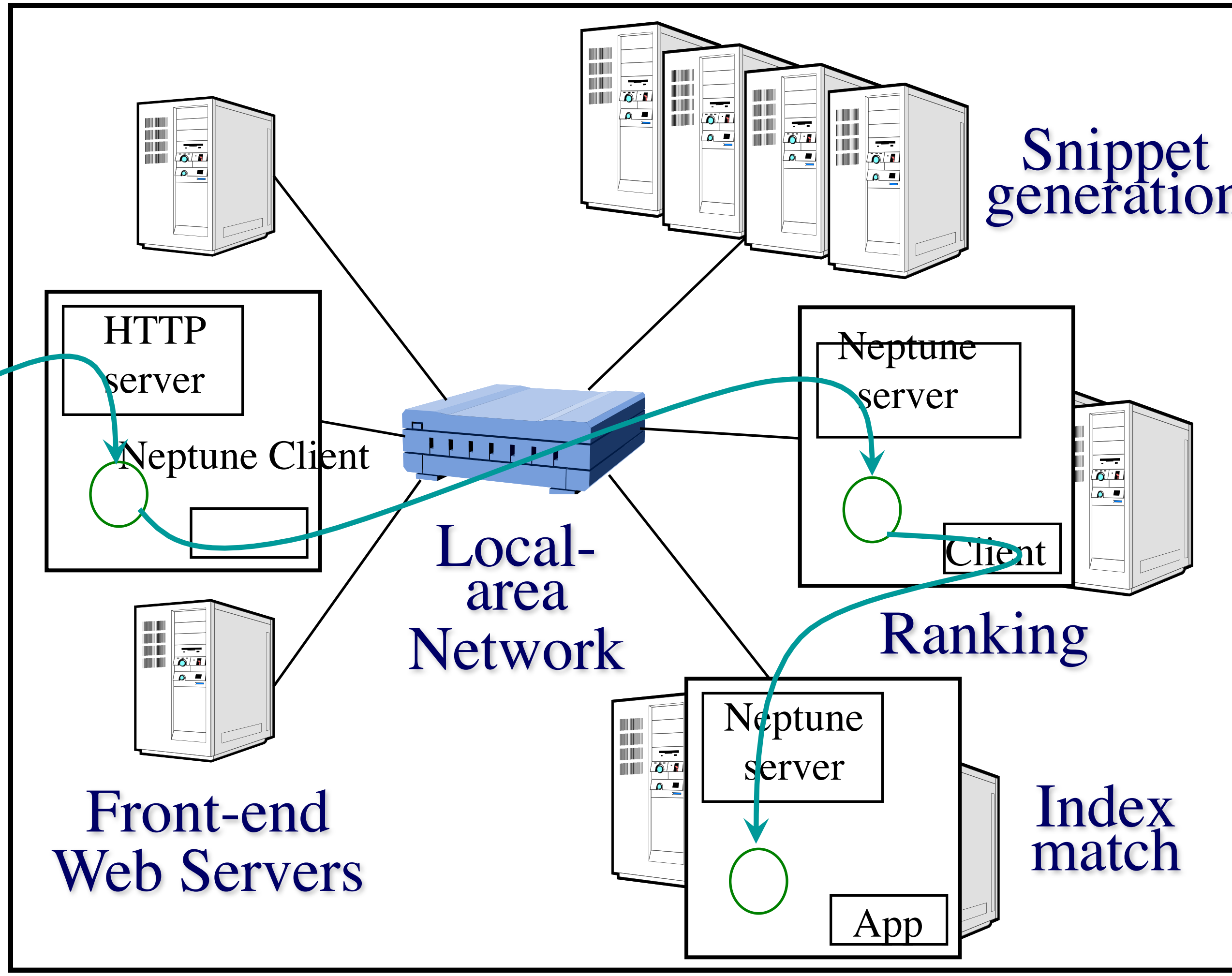
Programming Challenges for Online Services

- **Challenges/requirements for online services:**
  - Data intensive, requiring large-scale clusters.
  - Incremental scalability.
  - 7×24 availability.
  - Resource management, QoS for load spikes.
- **Fault Tolerance:**
  - Operation errors
  - Software bugs
  - Hardware failures
- **Lack of programming support for reliable/scalable online network services and applications.**

The Neptune Clustering Middleware

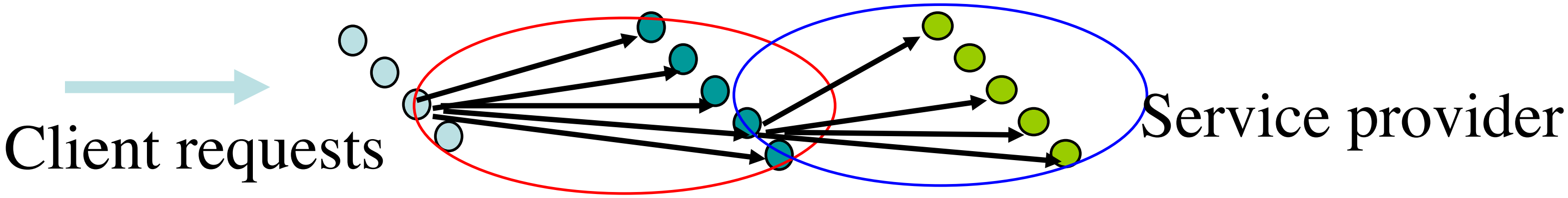
- Neptune: Clustering middleware for aggregating and replicating application modules with persistent data.
- A simple and flexible programming model to shield complexity of service discovery, load scheduling, consistency, and failover management
- www.cs.ucsb.edu/projects/neptune for code, papers, documents.
  - K. Shen, et. al, USENIX Symposium on Internet Technologies and Systems, 2001.
  - K Shen et al, OSDI 2002. PPoPP 2003.

Example: a Neptune Clustered Service: Index match service



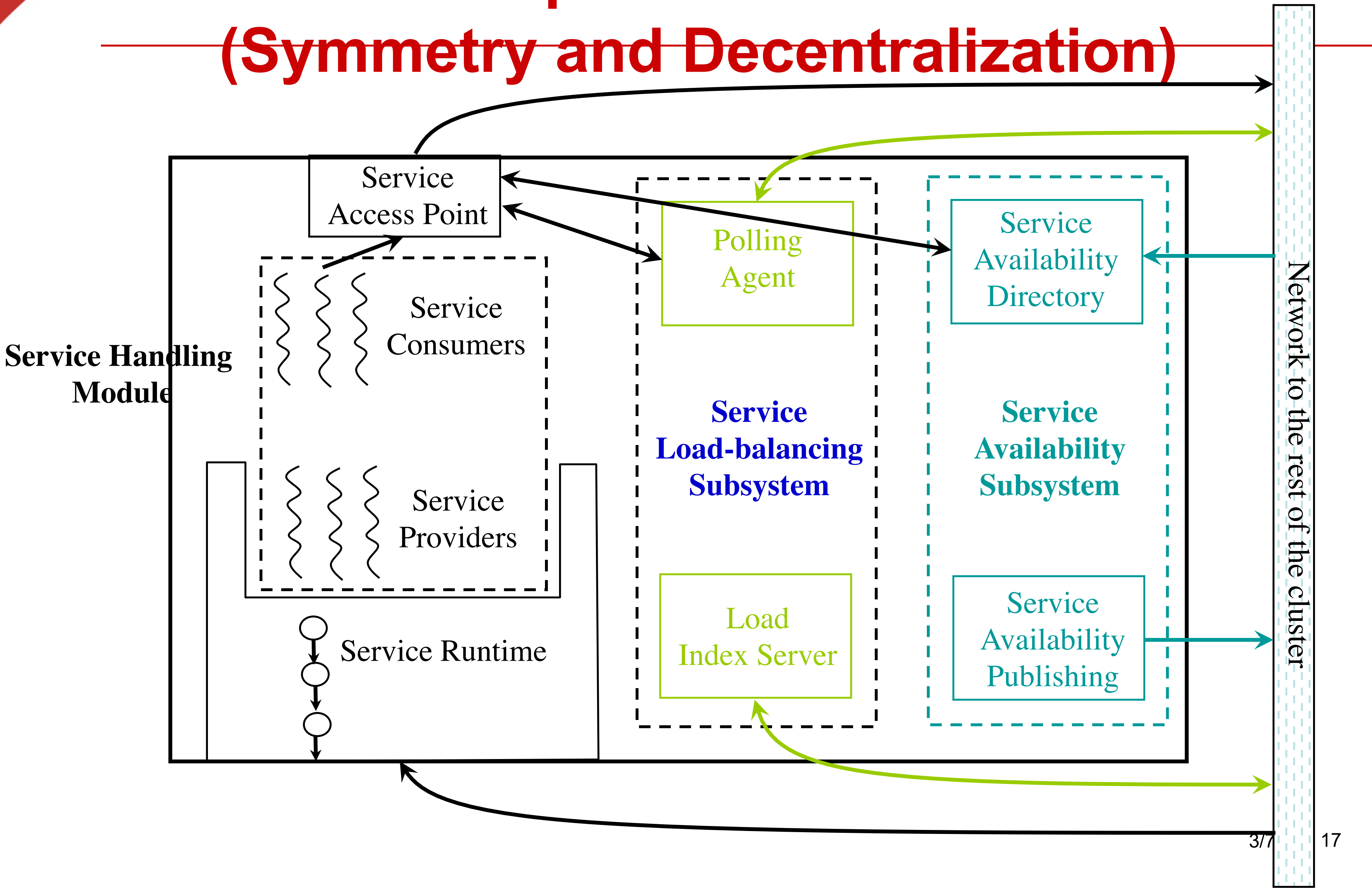
Neptune architecture for cluster-based services

- **Symmetric and decentralized**:
  - Each node can host multiple services, acting as a service provider (Server)
  - Each node can also subscribe internal services from other nodes, acting as a consumer (Client)
    - *Advantage: Support multi-tier or nested service architecture*



- **Neptune components at each node**:
  - Application service handling subsystem.
  - Load balancing subsystem.
  - Service availability subsystem.

Inside a Neptune Server Node (Symmetry and Decentralization)

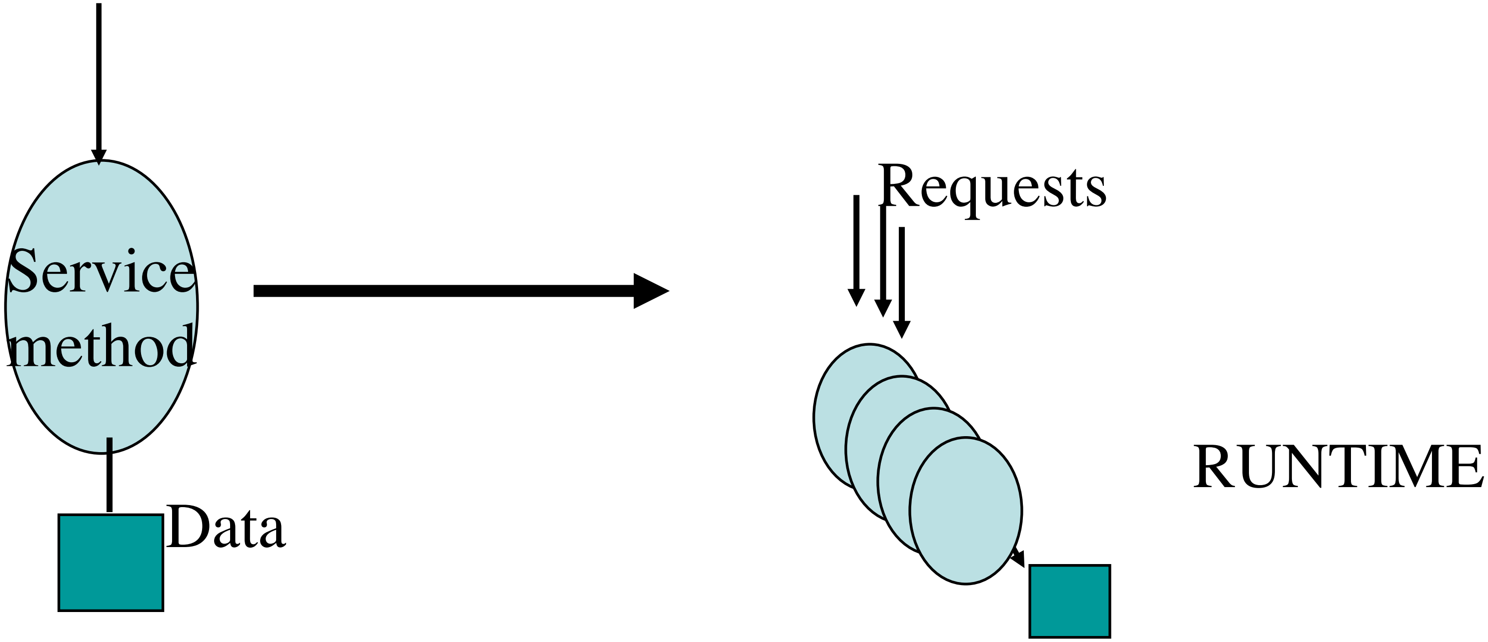


Availability and Load Balancing

- **Availability subsystem**:
  - Announcement once per second through IP multicast;
  - Availability info kept as soft state, expiring in 5 seconds;
  - Service availability directory kept in shared-memory for efficient local lookup.
- **Load-balancing subsystem**:
  - Challenging: medium/fine-grained requests.
  - Random polling with sampling.
  - Discarding slow-responding polls

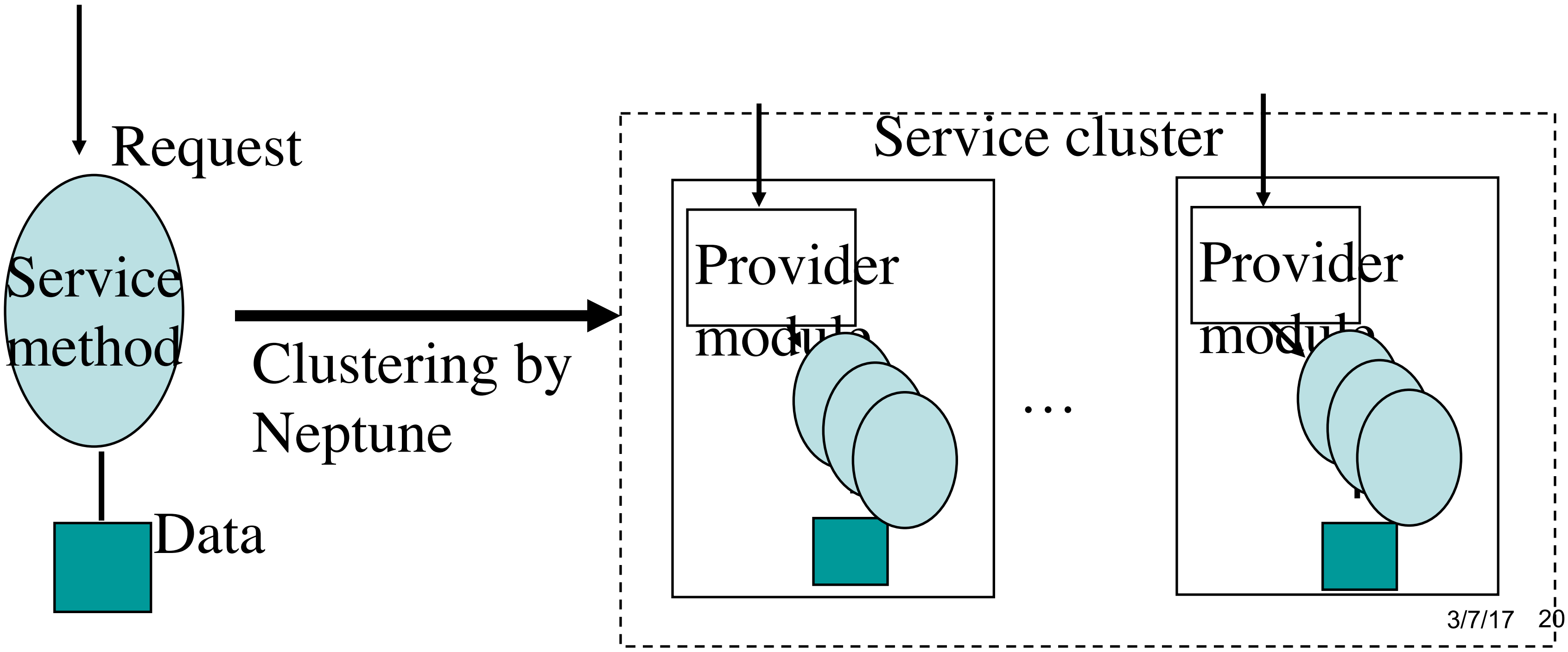
Programming Model in Neptune

- **Request-driven processing model**: programmers specify service methods to process each request.
- **Application-level concurrency:** Each service provider uses a thread or a process to handle a new request and respond.

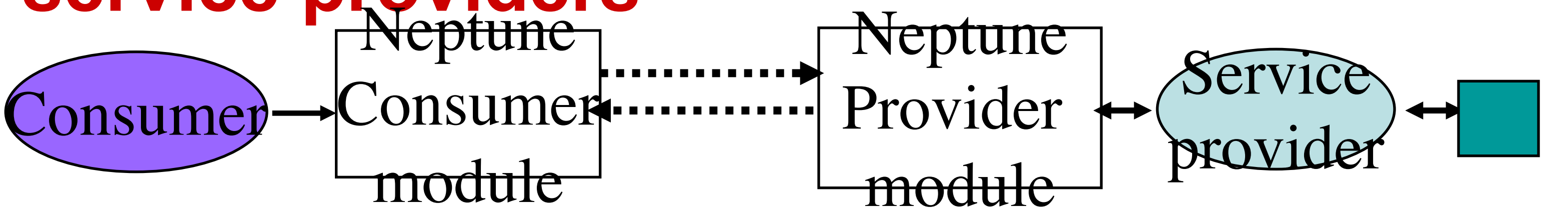


Cluster-level Parallelism/Redudancy

- **Large data sets** can be partitioned and replicated.
- **SPMD model** (single program/multiple data).
- **Transparent service access**: Neptune provides runtime modules for service location and consistency.



Service invocation from consumers to service providers



- **Request/response messages:**
  - *Consumer side:* NeptuneCall(service_name, partition_ID, service_method, request_msg, response_msg);
  - *Provider side:* "service_method" is a library function. Service_method(partitionID, request_msg, result_msg);
  - *Parallel invocation with aggregation*
- **Stream-based communication:** Neptune sets up a bi-directional stream between a consumer and a service provider. Application invocation uses it for socket communication.

Code Example of Consumer Program

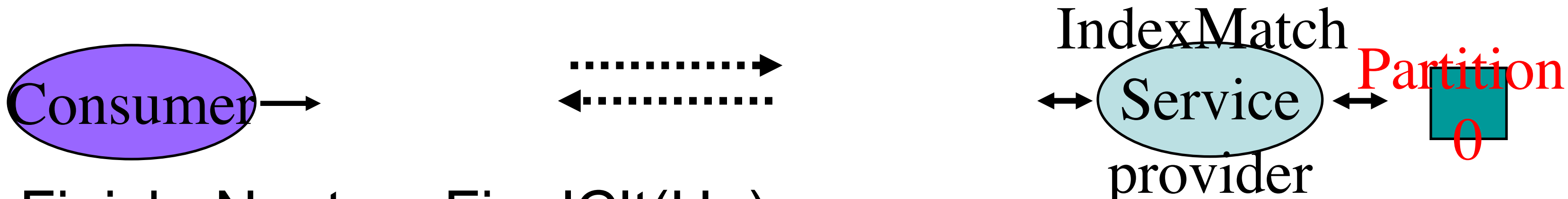
1. Initialize

Hp=NeptuneInitClt(LogFile);

2. Make a connection

NeptuneConnect (Hp, “IndexMatch”, 0, Neptune_MODE_READ, “IndexMatchSvc”, &fd, NULL);

3. Then use fd as TCP socket to read/write data



4. Finish. NeptuneFinalClt(Hp);

Example of server-side API with stream-based communication

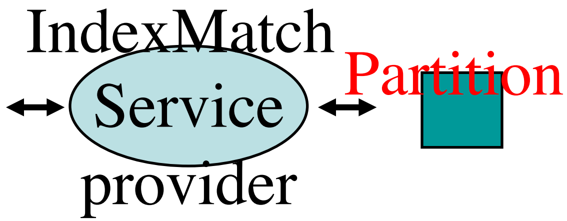
- **Server-side functions**

Void IndexMatchInit(Handle)

Initialization routine.

Void IndexMatchFinal(Handle)

Final processing routine.



Void IndexMatchSvc(Handle, parititionID, ConnSd)

Processing routine for each indexMatch request.

Publishing Index Search Service

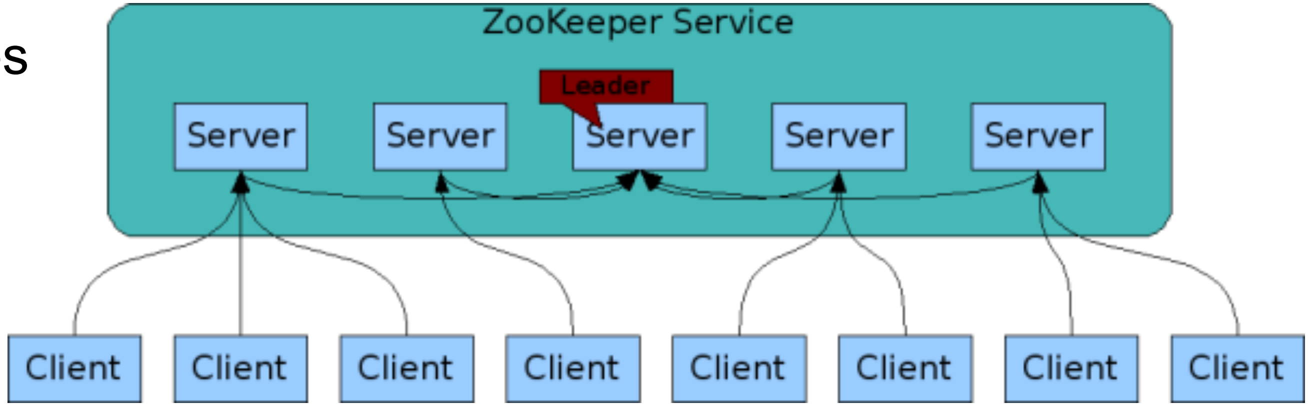
- **Example of configuration file**

```
[IndexMatch]
SVC_DLL = /export/home/neptune/IndexTier2.so
LOCAL_PARTITION = 0,4           # Partitions hosted
INITPROC=IndexMatchInit
FINALPROC=IndexMatchFinal
STREAMPROC=IndexMatchSvc
```

ZooKeeper

- **Coordinating distributed systems as “zoo” management**
  - http://zookeeper.apache.org

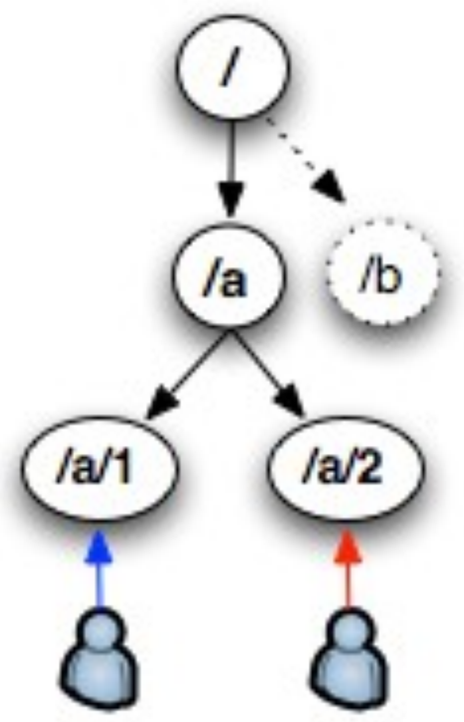
- **Open source high-performance coordination service for distributed applications**
  - Naming
  - Configuration management
  - Synchronization
  - Group services



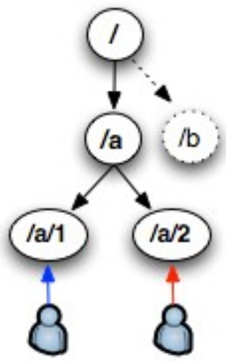
Data Model

- **Hierarchal namespace (like a metadata file system)**
- **Each znode has data and children**
- **data is read and written in its entirety**

The znode will be deleted when the creating client's session times out or it is explicitly deleted



Zookeeper Operations



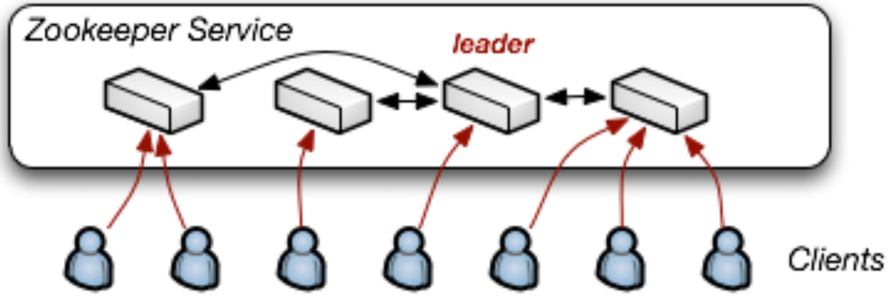
Operation	Description
create	Creates a znode (the parent znode must already exist)
delete	Deletes a znode (the znode must not have any children)
exists	Tests whether a znode exists and retrieves its metadata
getACL, setACL	Gets/sets the ACL (access control list) for a znode
getChildren	Gets a list of the children of a znode
getData, setData	Gets/sets the data associated with a znode
sync	Synchronizes a client's view of a znode with ZooKeeper

Zookeeper: Distributed Architecture

Start with support for a file API

1) Partial writes/reads

2) Rename



- **Ordered updates and strong persistence guarantees**
- **Conditional updates**
- **Watches for data changes and ephemeral nodes**