



elasticsearch

The Road to a
Distributed, (Near) Real Time, Search Engine

Shay Banon - @kimchy

Lucene Basics - Directory

- A File System Abstraction
- Mainly used to read and write “files”
- Used to read and write different index files

Lucene Basics - IndexWriter

- Used to add documents / delete documents from the index
- Changes are stored in memory (possibly flushing to maintain memory limits)
- Requires a commit to make changes “persistent”, which is expensive
- A single IndexWriter can write to an index, expensive to create (reuse at all cost!)

Lucene Basics - Index Segments

- An index is composed of internal segments
- Each segment is almost a self sufficient index by itself, immutable up to deletes
- Commits “officially” adds segments to the index, though internal flushing might create new segments as well
- Segments are merged continuously
- A lot of caching per segment (terms, field)

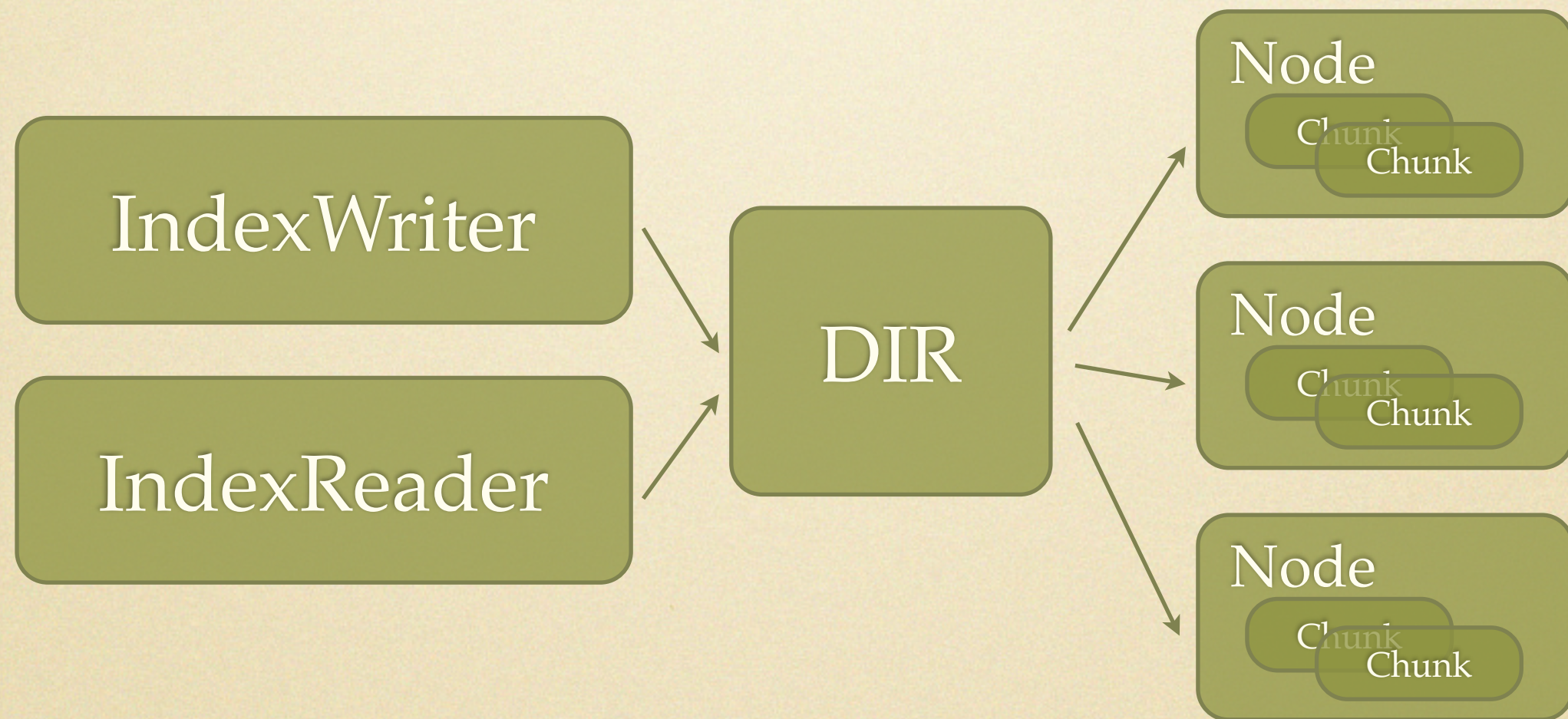
Lucene Basics - (Near) Real Time

- IndexReader is the basis for searching
- IndexWriter#getReader allows to get a refreshed reader that sees changes done to IW
 - Requires flushing (but not committing)
 - Can't call it on each operation, too expensive
- Segment based readers and search

Distributed Directory

- Implement a Directory that works on top of a distributed “system”
- Store file chunks, read them on demand
- Implemented for most (Java) data grids
 - Compass - GigaSpaces, Coherence, Terracotta
 - Infinispan

Distributed Directory



Distributed Directory

- “Chatty” - many network roundtrips to fetch data
- Big indices still suffer from a non distributed IndexReader
 - Lucene IndexReader can be quite “heavy”
- Single IndexWriter problem, can’t really scale writes

Partitioning

- Document Partitioning
 - Each shard has a subset of the documents
 - A shard is a fully functional “index”
- Term Partitioning
 - Shards has subset of terms for all docs

Partitioning - Term Based

- pro: K term query -> handled at most by K shards
- pro: $O(K)$ disk seeks for K term query
- con: high network traffic
 - data about each matching term needs to be collected in one place
- con: harder to have per doc information (facets / sorting / custom scoring)

Partitioning - Term Based

- Riak Search - Utilizing its distributed key-value storage
- Lucandra (abandoned, replaced by Solandra)
 - Custom IndexReader and IndexWriter to work on top of Cassandra
 - Very very “chatty” when doing a search
 - Does not work well with other Lucene constructs, like FieldCache (by doc info)

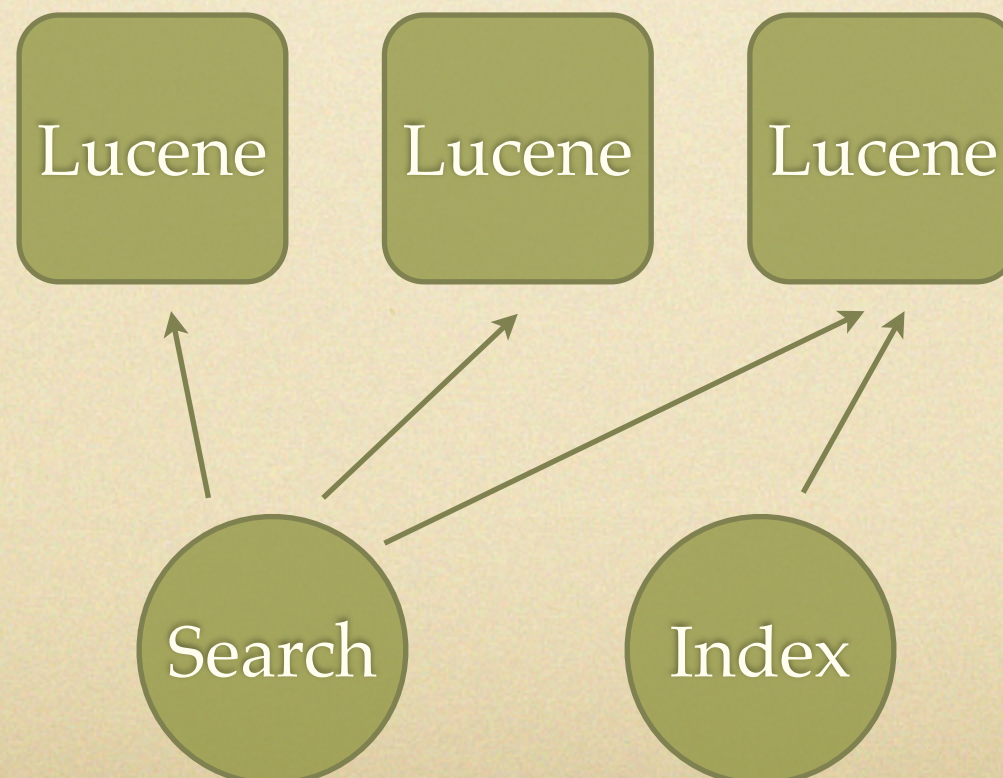
Partitioning - Document Based

- pro: each shard can process queries independently
- pro: easy to keep additional per-doc information (facets, sorting, custom scoring)
- pro: network traffic small
- con: query has to be processed by each shard
- con: $O(K*N)$ disk seeks for K term on N shard

Distributed Lucene

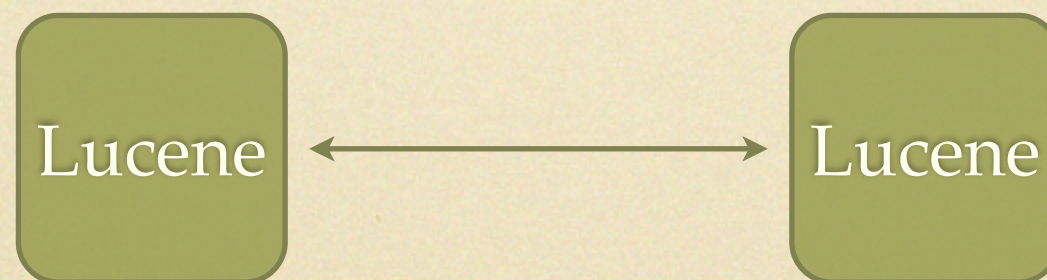
Doc Partitioning

- Shard Lucene into several instances
- Index a document to one Lucene shard
- Distribute search across Lucene shards



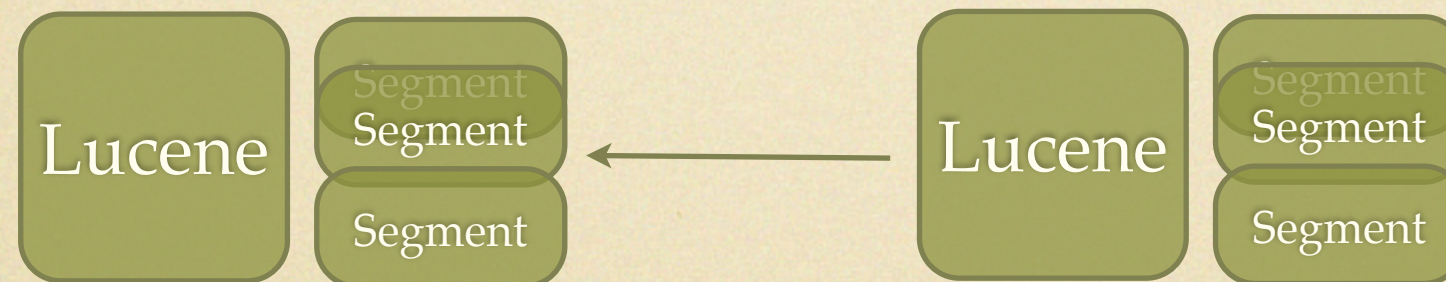
Distributed Lucene Replication

- Replicated Lucene Shards
 - High Availability
 - Scale search by searching replicas



Pull Replication

- Master - Slave configuration
- Slave pulls index files from the master (delta, only new segments)

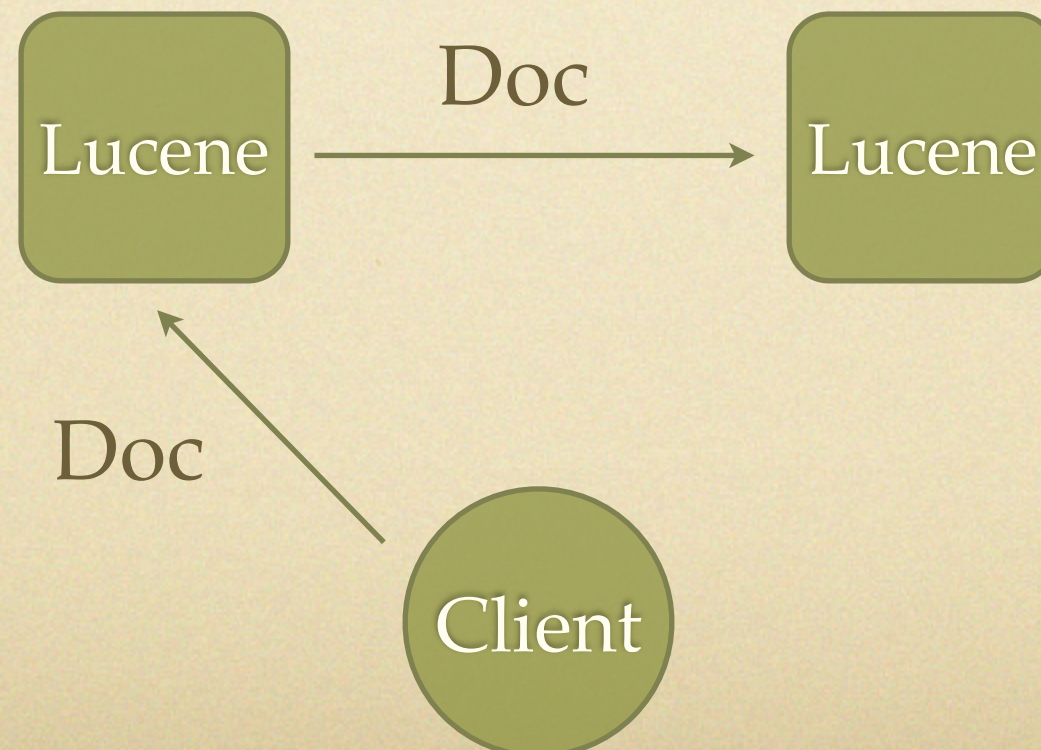


Pull Replication - Downsides

- Requires a “commit” on master to make changes available for replication to slave
- Redundant data transfer as segments are merged (especially for stored fields)
- Friction between commit (heavy) and replication, slaves can get “way” behind master (big new segments), loses HA
- Does not work for real time search, slaves are “too” behind

Push Replication

- “Master / Primary” push to all the replicas
- Indexing is done on all replicas



Push Replication - Downsides

- Indexing the document on all nodes
 - (though less data transfer over the wire)
- Delicate control over concurrent indexing operations
 - Usually solved using versioning

Push Replication - Benefits

- Documents indexed are immediately available on all replicas
- Improves High Availability
- Allows for (near) real time search architecture
- Architecture allows to switch “roles” ->
 - Primary dies, slave can become primary, and still allow indexing

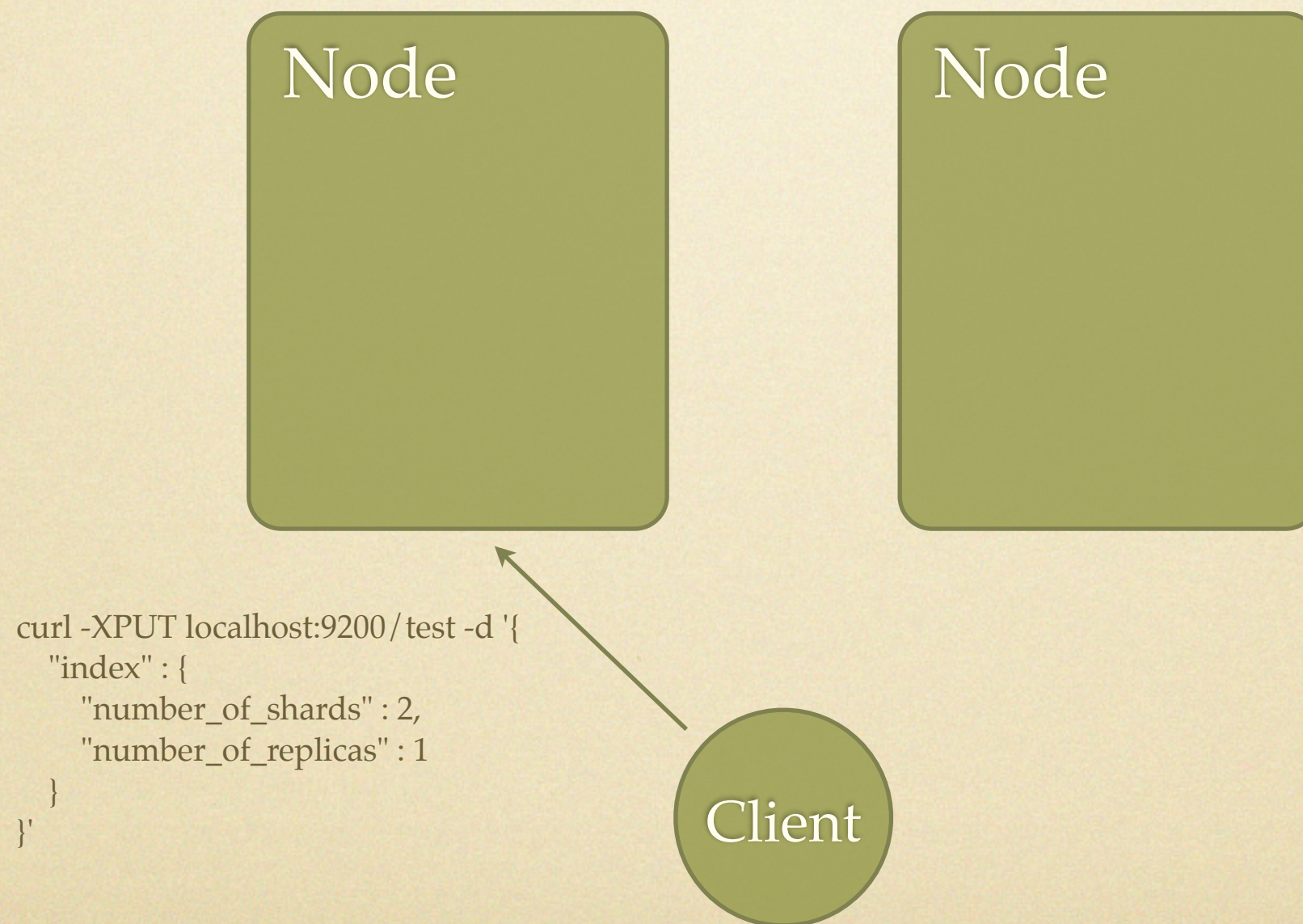
Push Replication - IndexWriter#commit

- IndexWriter#commit is heavy, but required in order to make sure data is actually persisted
- Can be solved by having a write ahead log that can be replayed on the event of a crash
- Can be more naturally supported in push replication

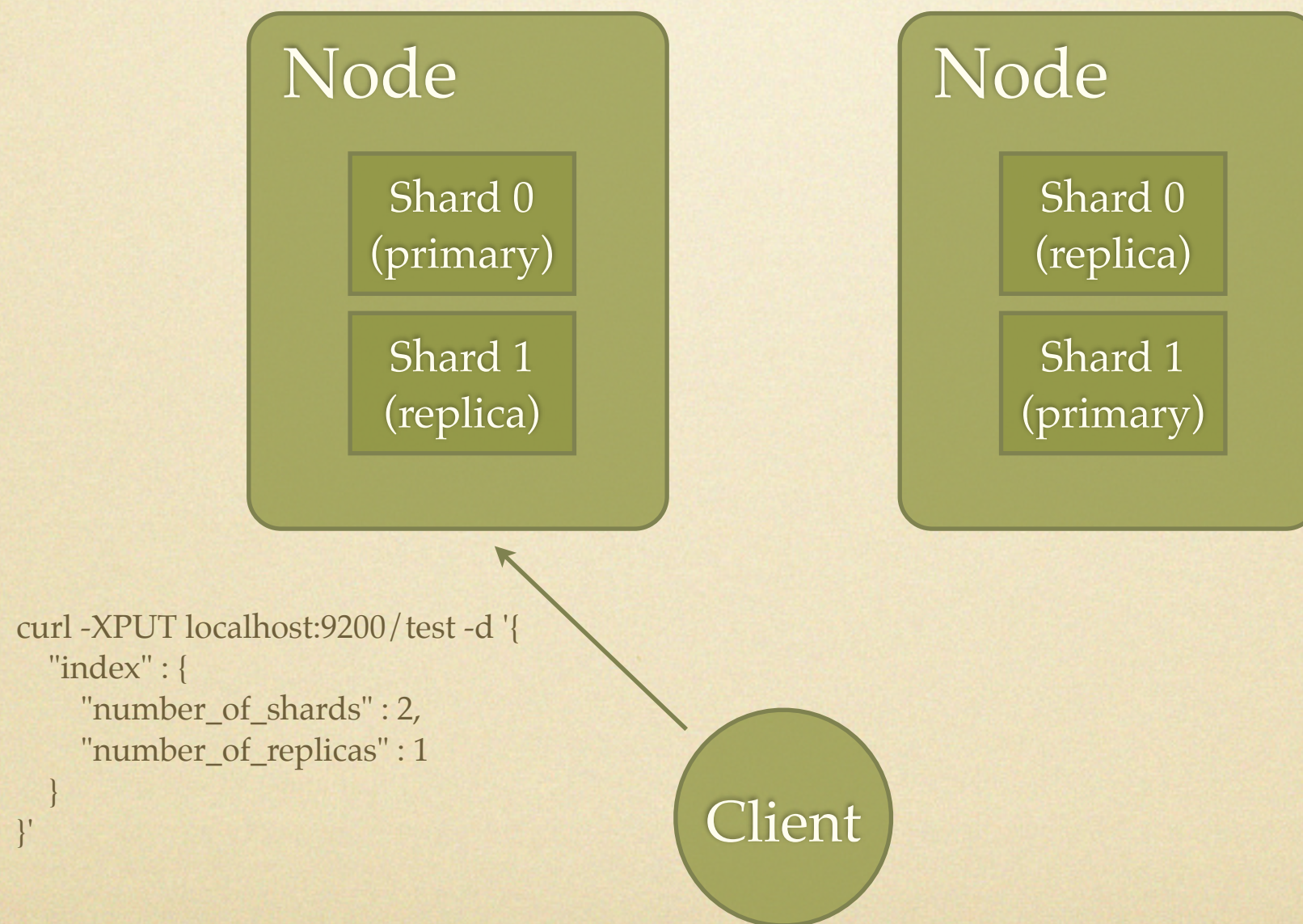
elasticsearch

<http://www.elasticsearch.org>

index - shards and replicas

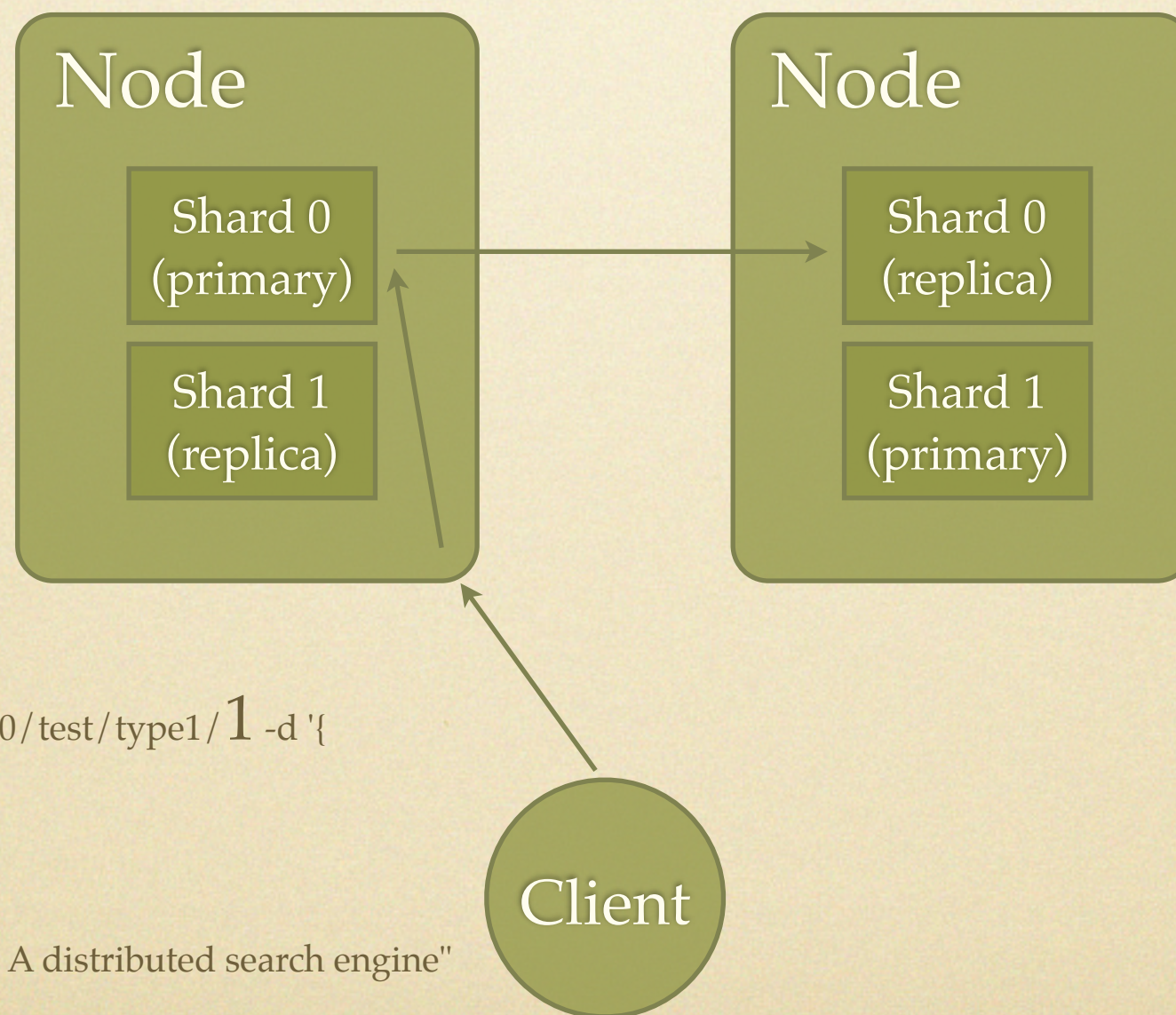


index - shards and replicas



indexing - 1

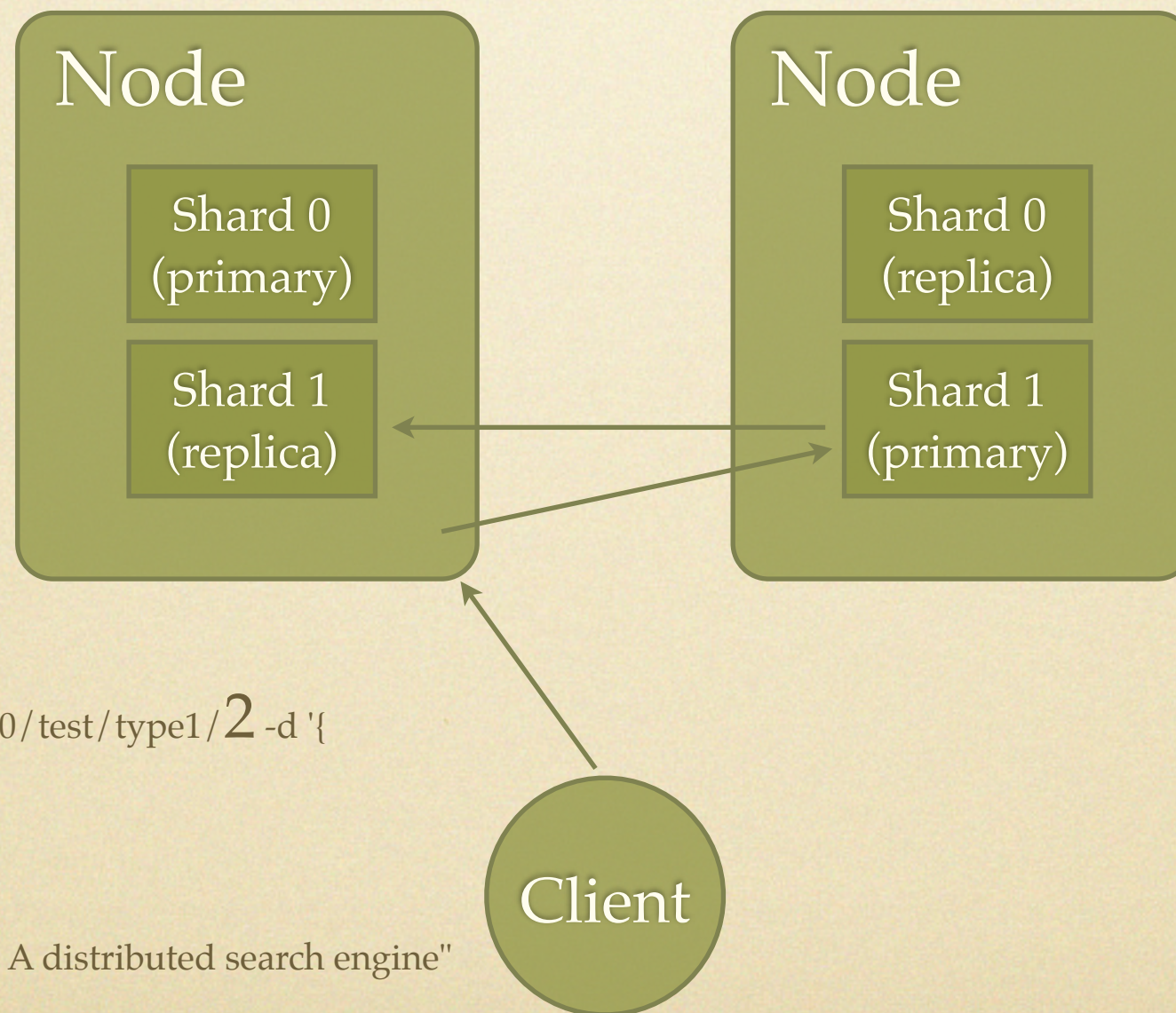
- Automatic sharding, push replication



```
curl -XPUT localhost:9200/test/type1/1 -d '{
  "name" : {
    "first" : "Shay",
    "last" : "Banon"
  },
  "title" : "ElasticSearch - A distributed search engine"
}'
```


indexing - 2

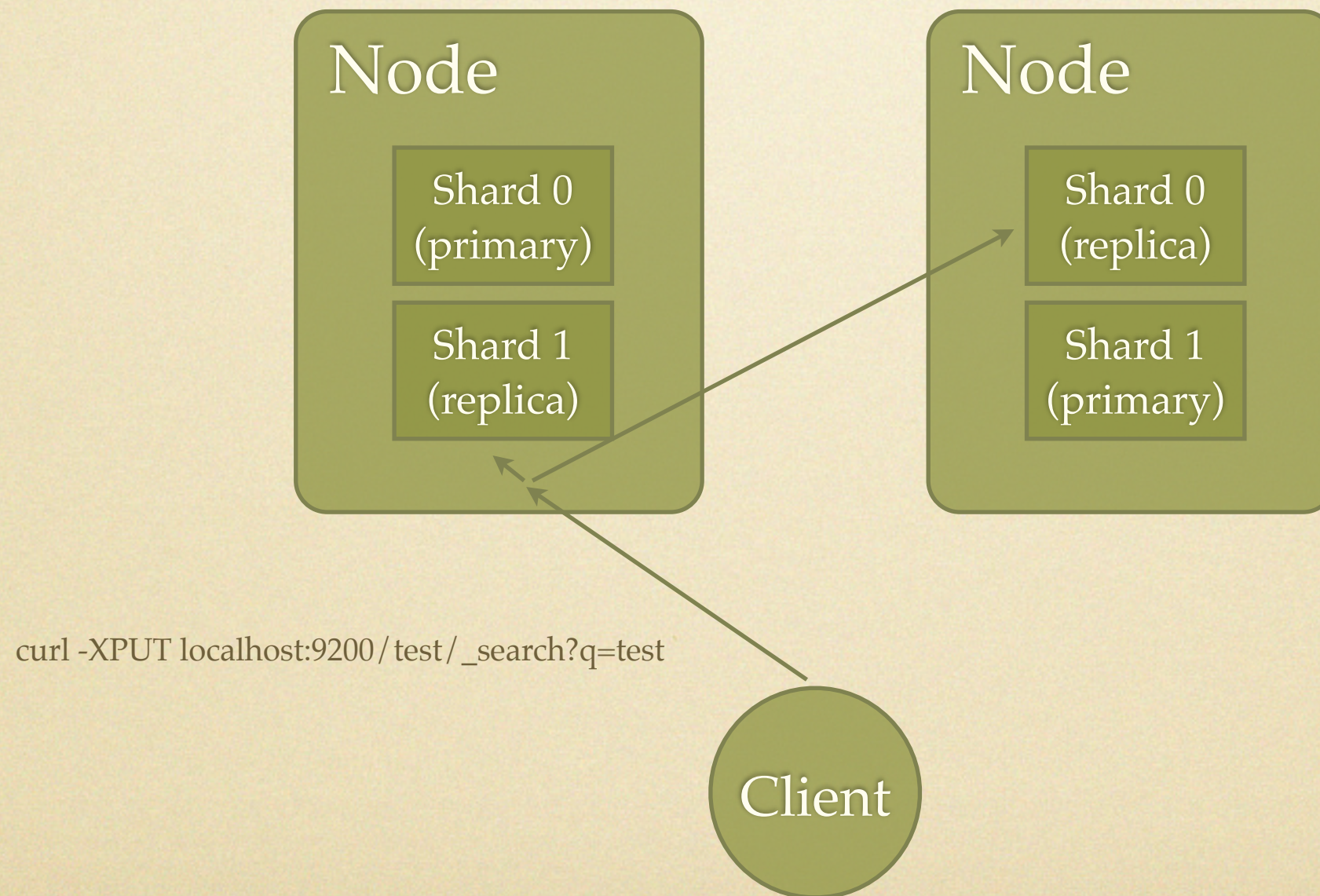
- Automatic request “redirection”



```
curl -XPUT localhost:9200/test/type1/2 -d '{  
  "name" : {  
    "first" : "Shay",  
    "last" : "Banon"  
  },  
  "title" : "ElasticSearch - A distributed search engine"  
}'
```

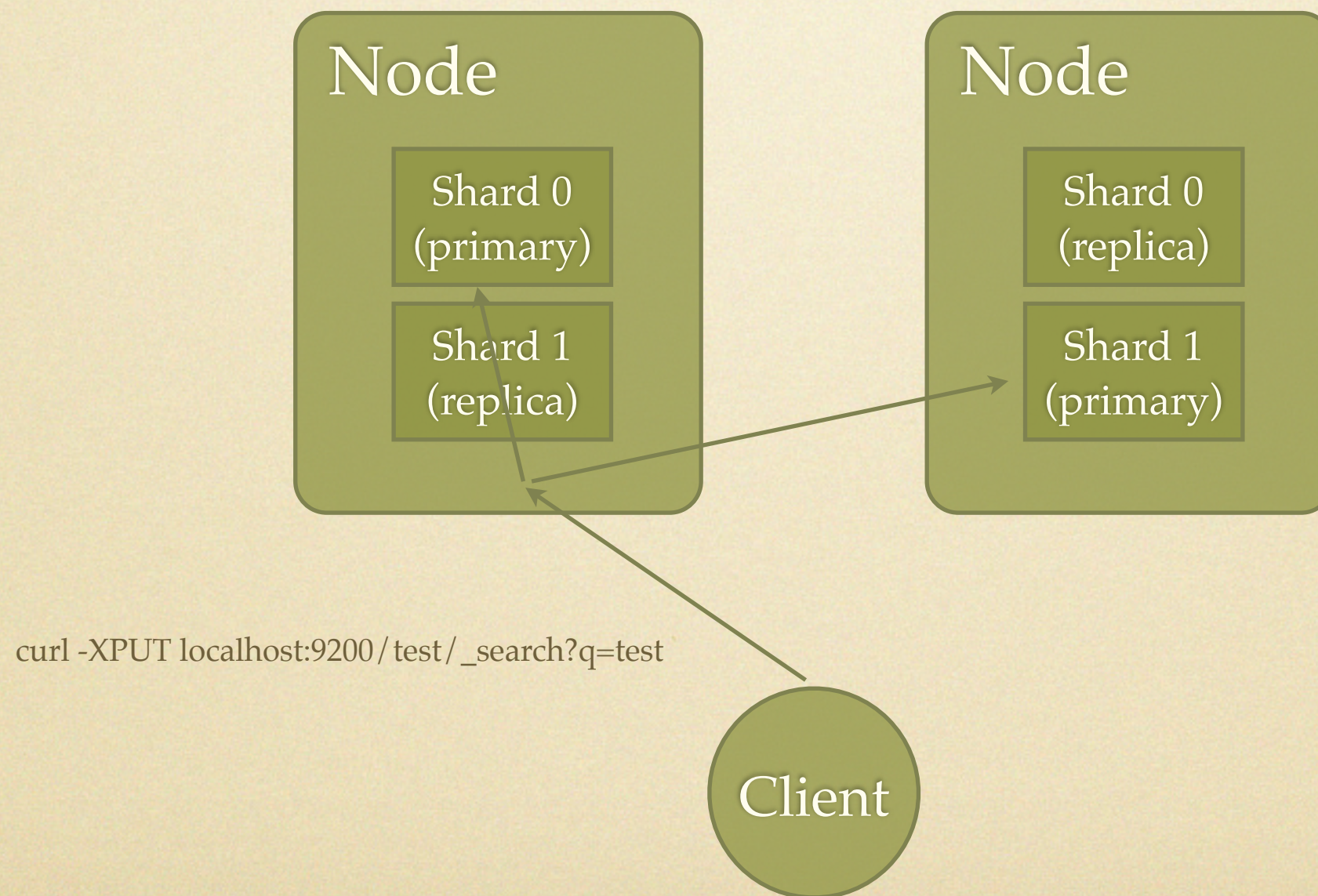

search - 1

- Scatter / Gather search



search - 2

- Automatic balancing between replicas

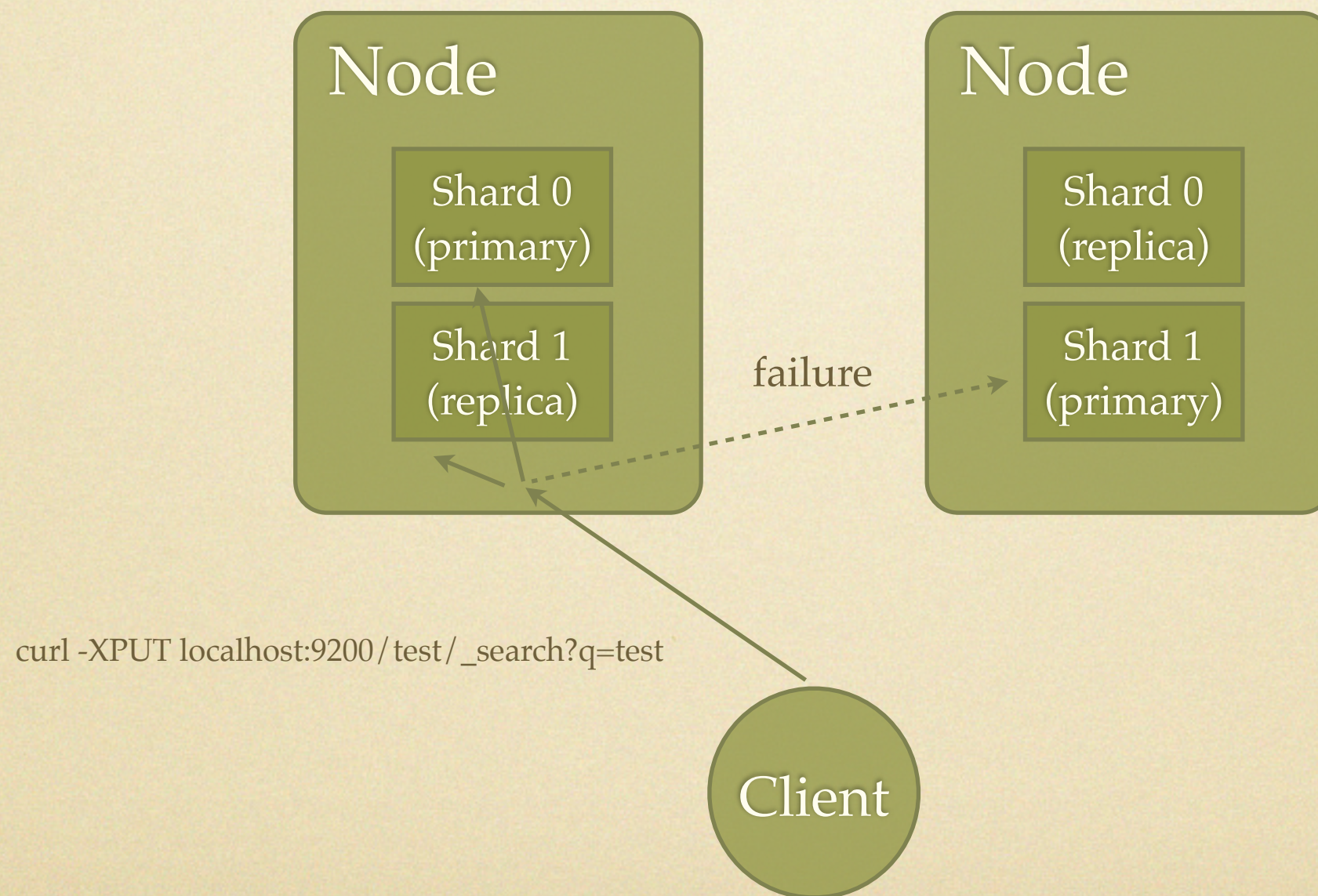


`curl -XPUT localhost:9200/test/_search?q=test`

Client

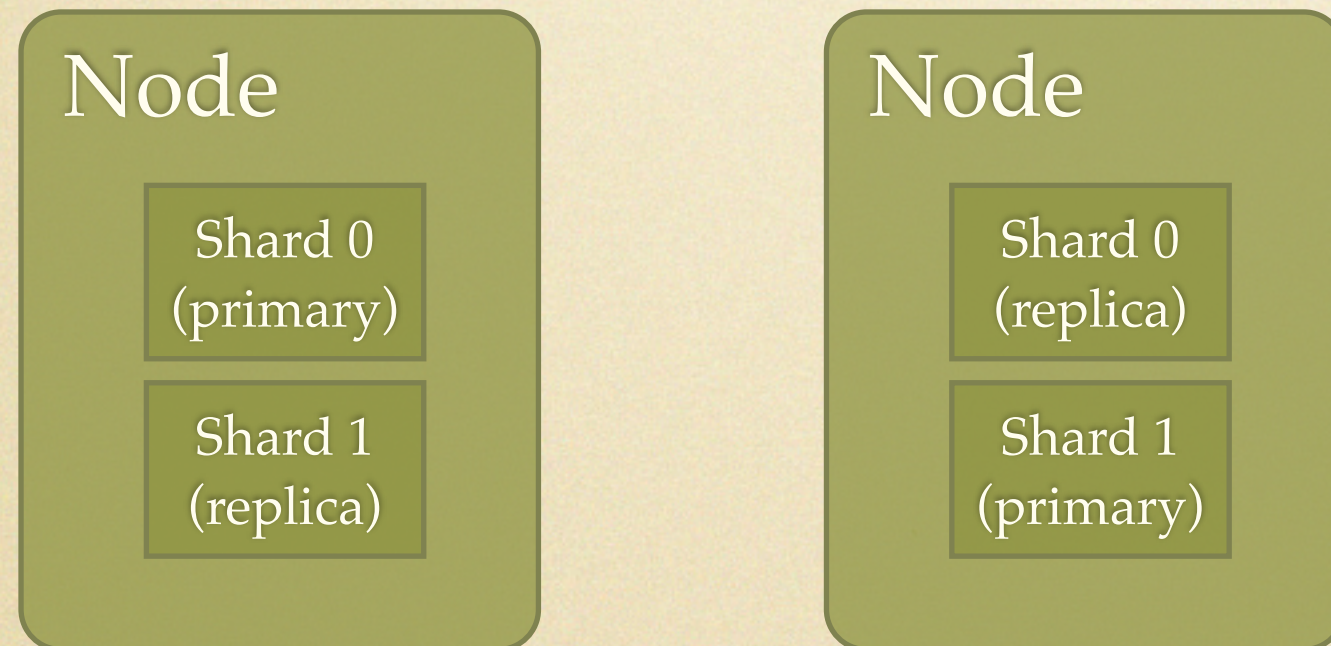
search - 3

- Automatic failover



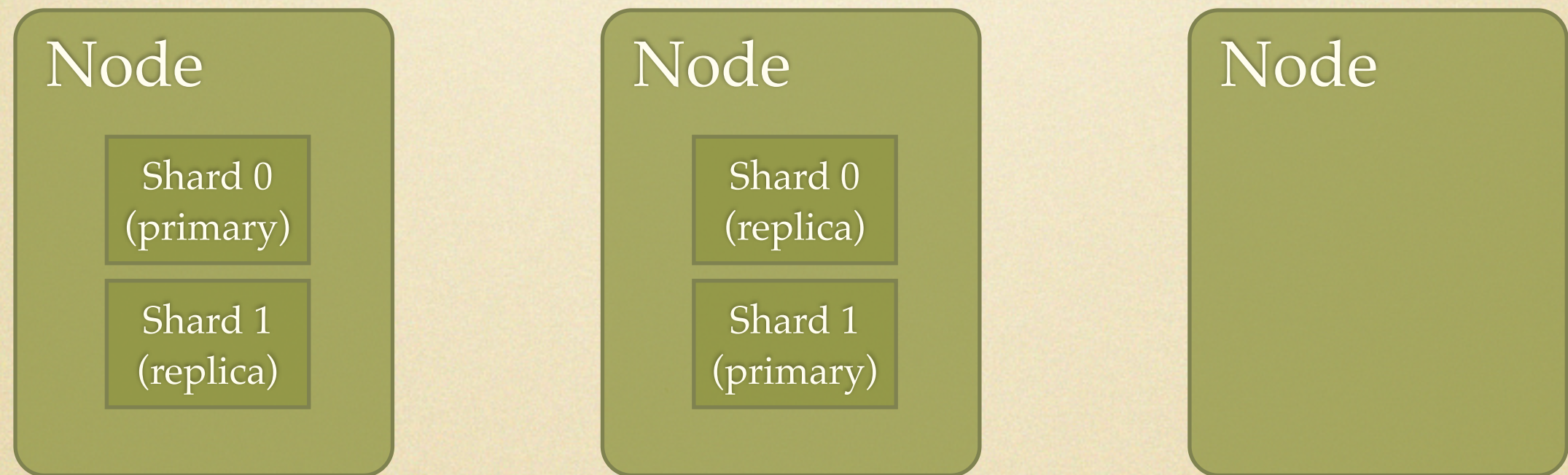
adding a node

- “Hot” relocation of shards to the new node



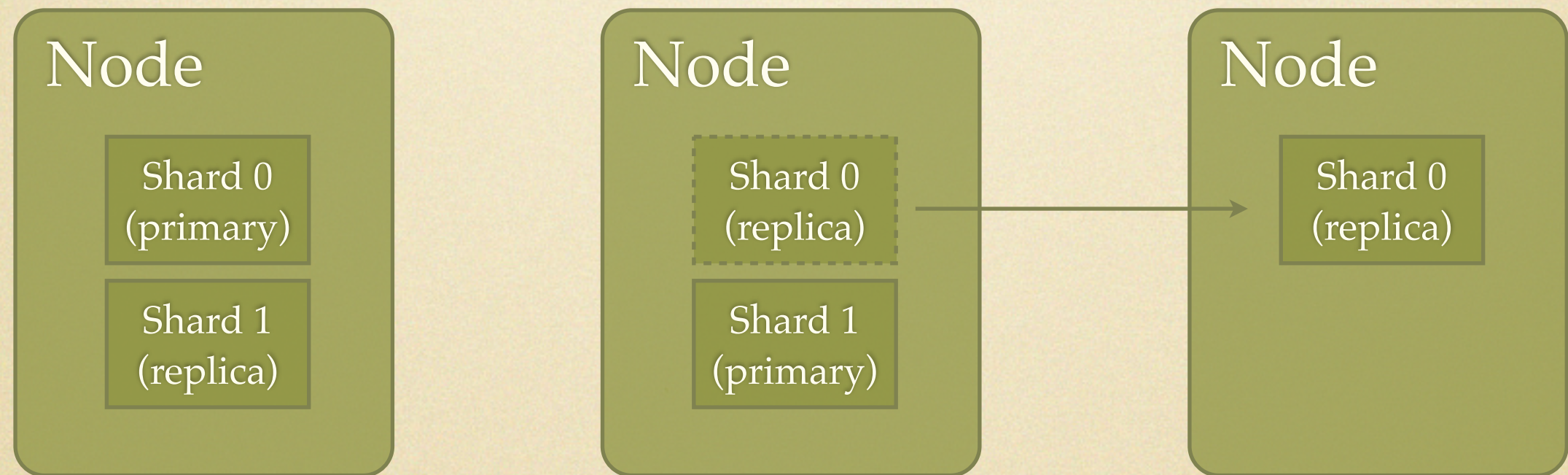
adding a node

- “Hot” relocation of shards to the new node

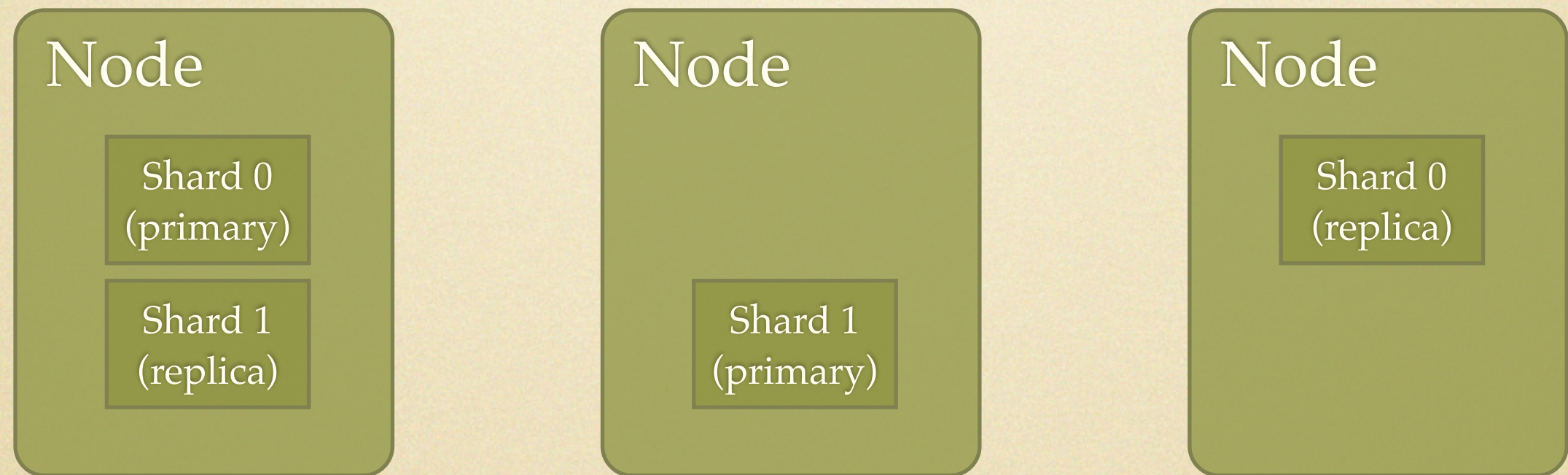


adding a node

- “Hot” relocation of shards to the new node

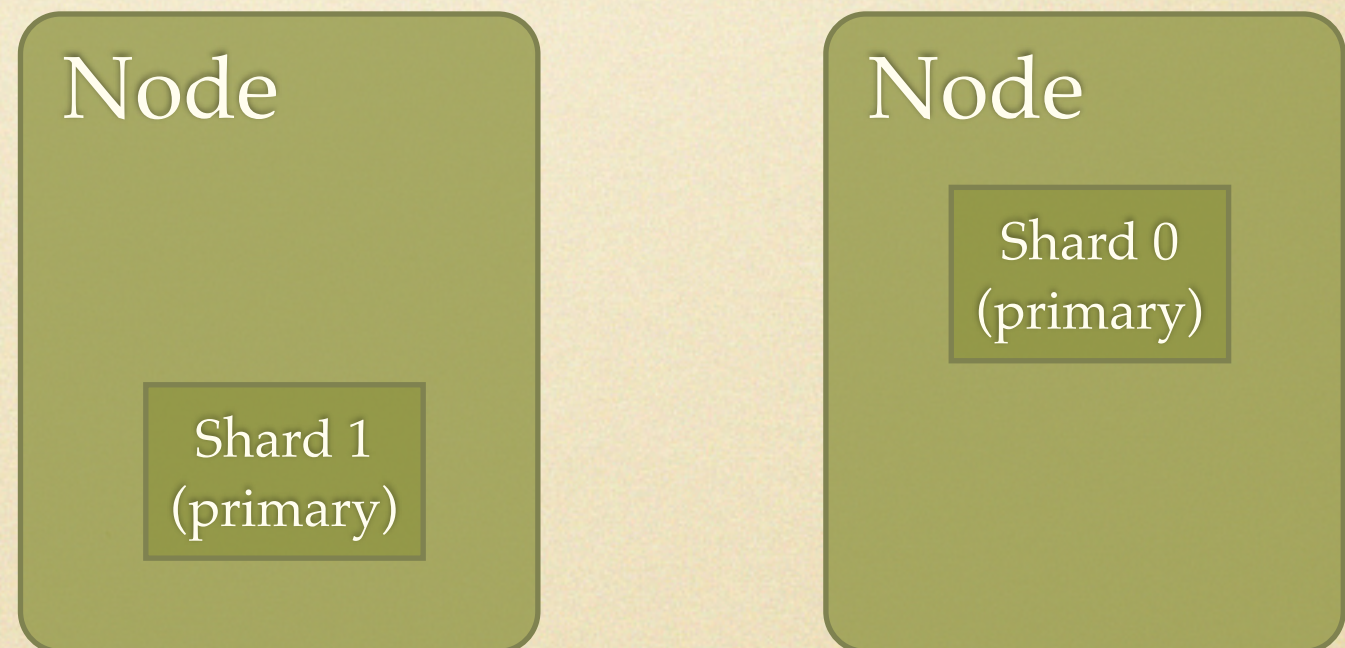


node failure



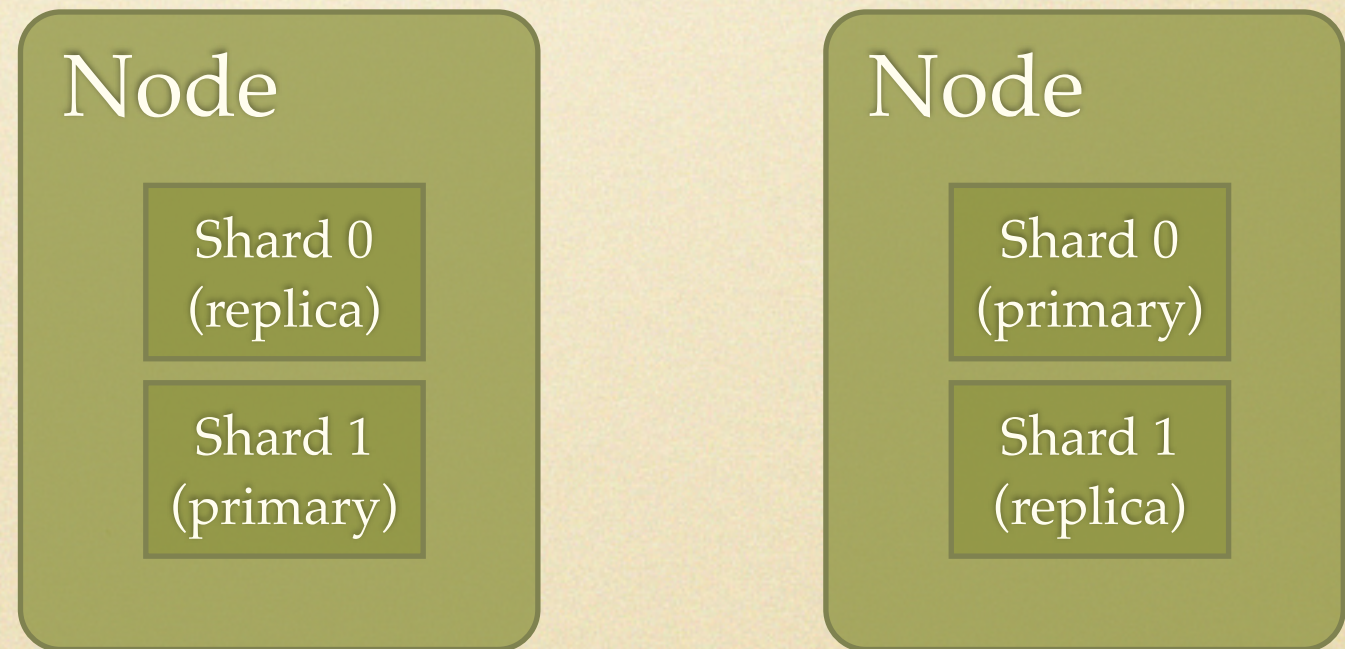
node failure - 1

- Replicas can automatically become primaries

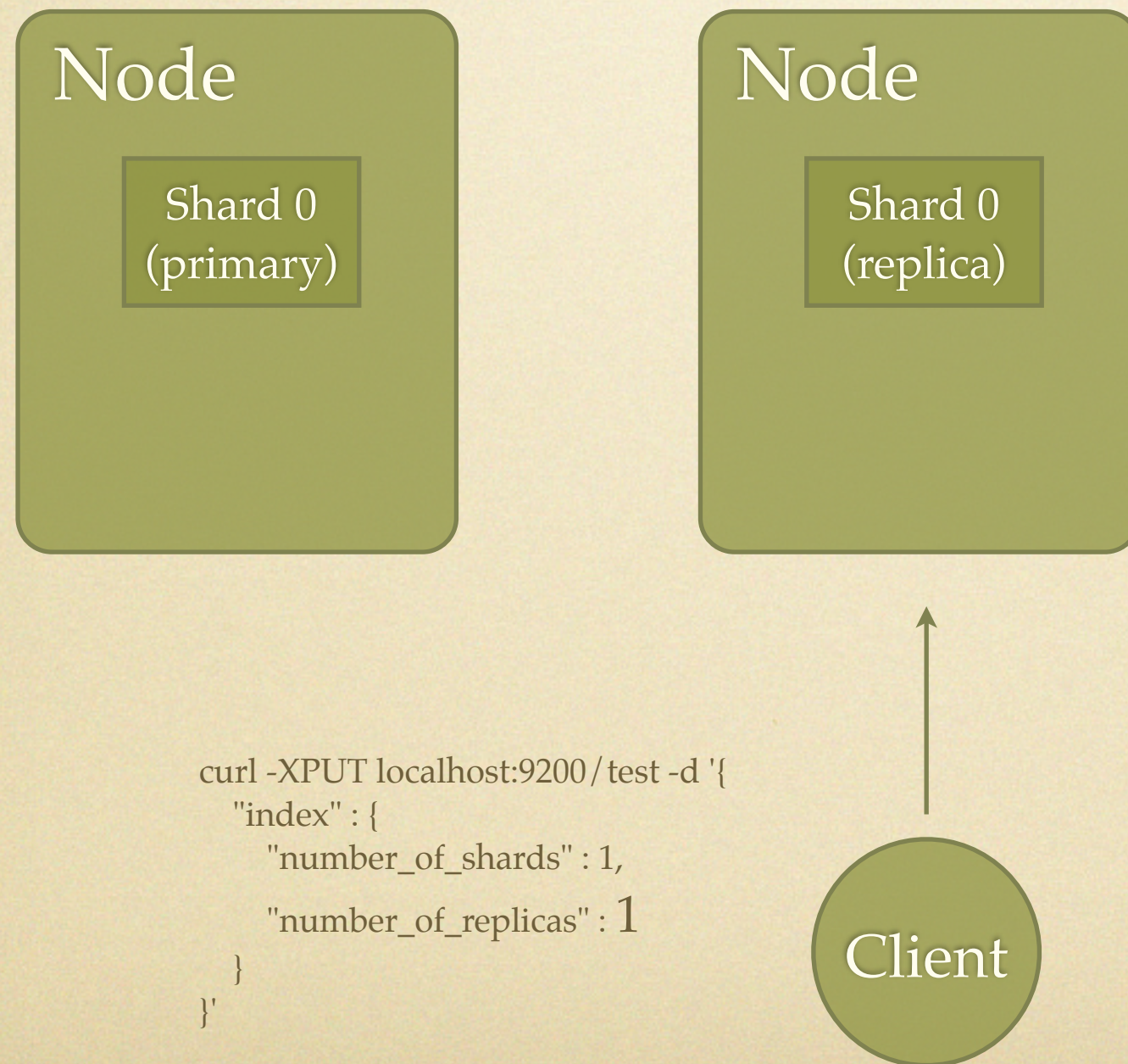


node failure - 2

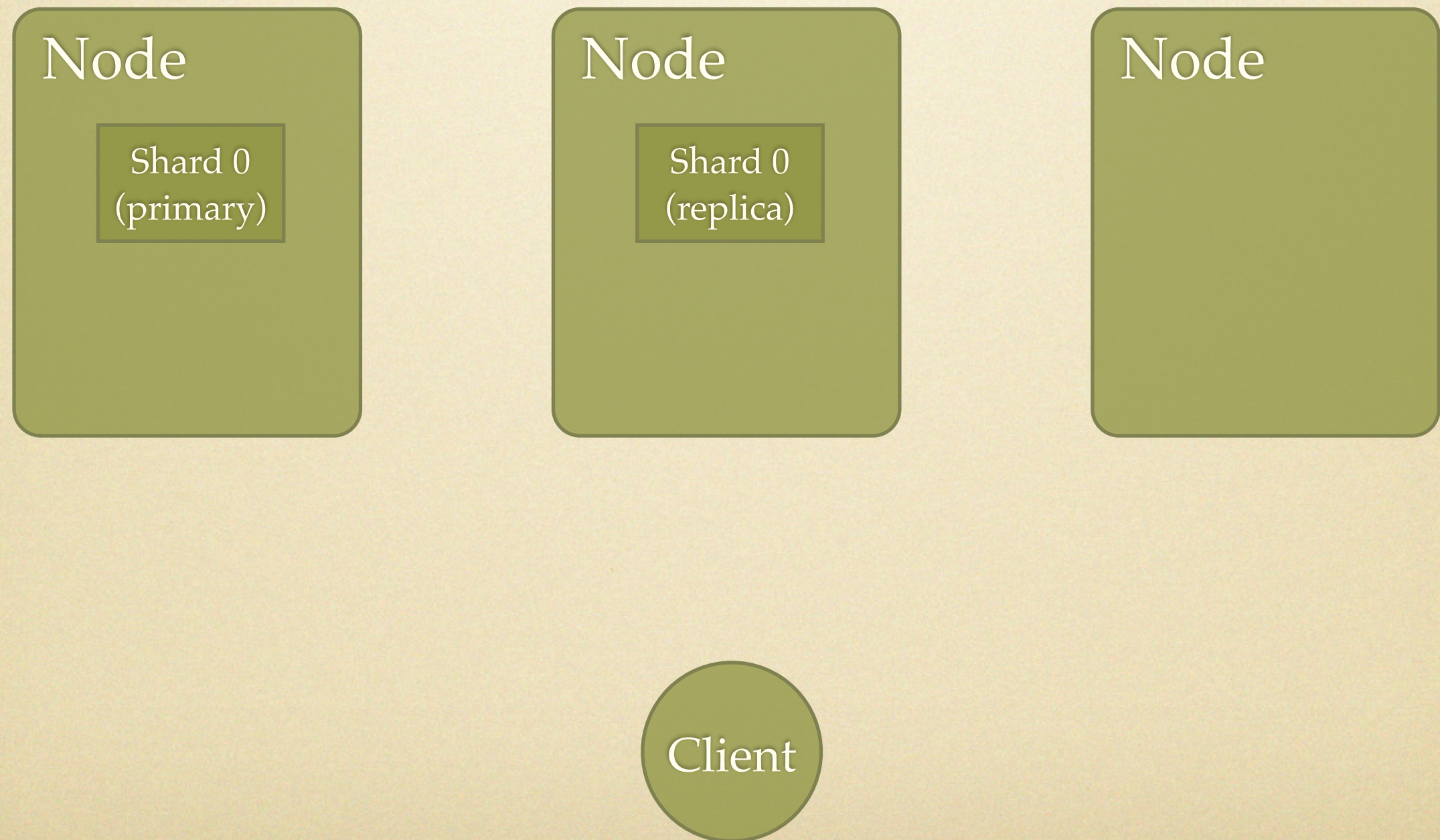
- Shards are automatically assigned, and do “hot” recovery



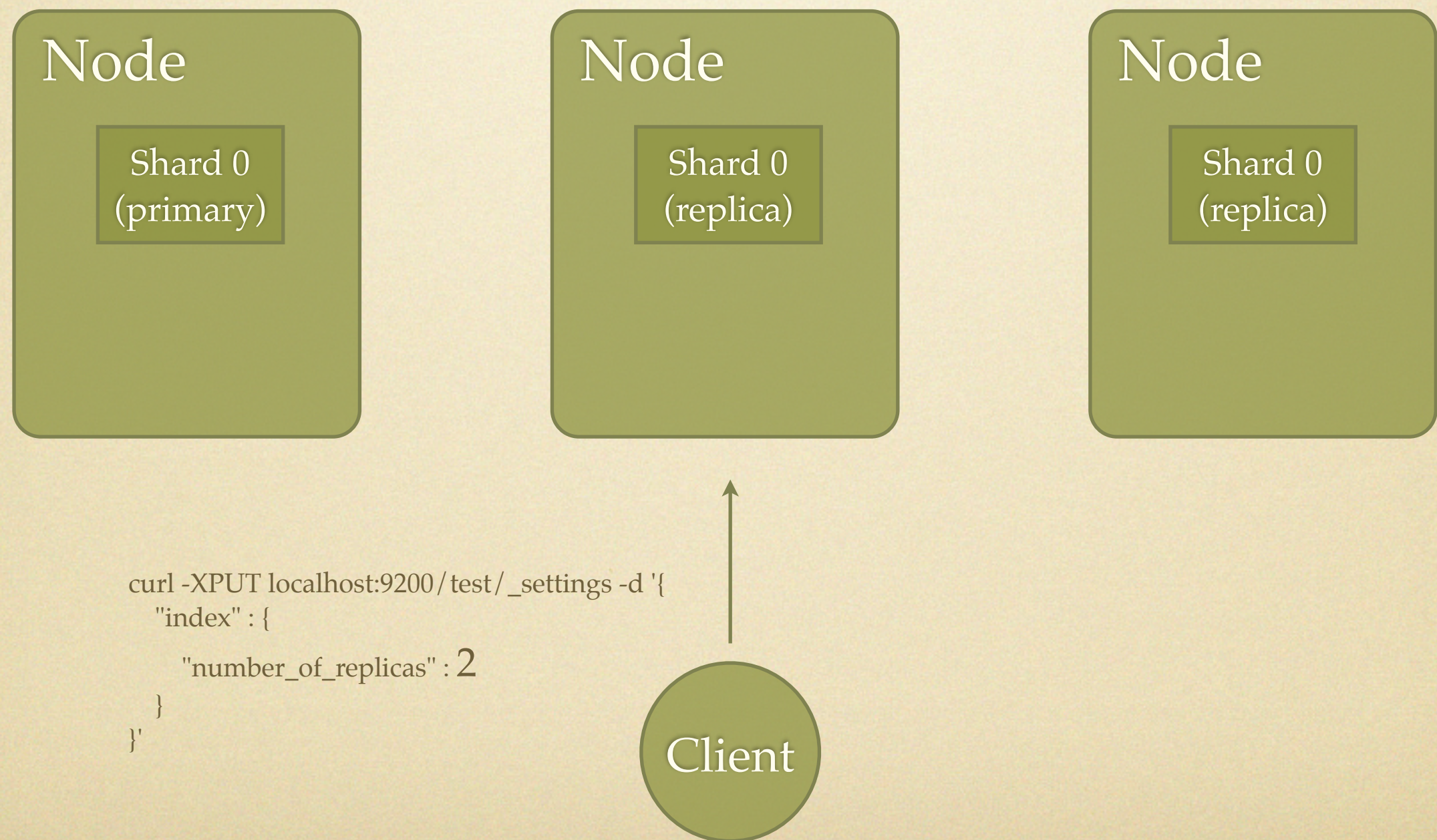
dynamic replicas



dynamic replicas



dynamic replicas



multi tenancy - indices

Node

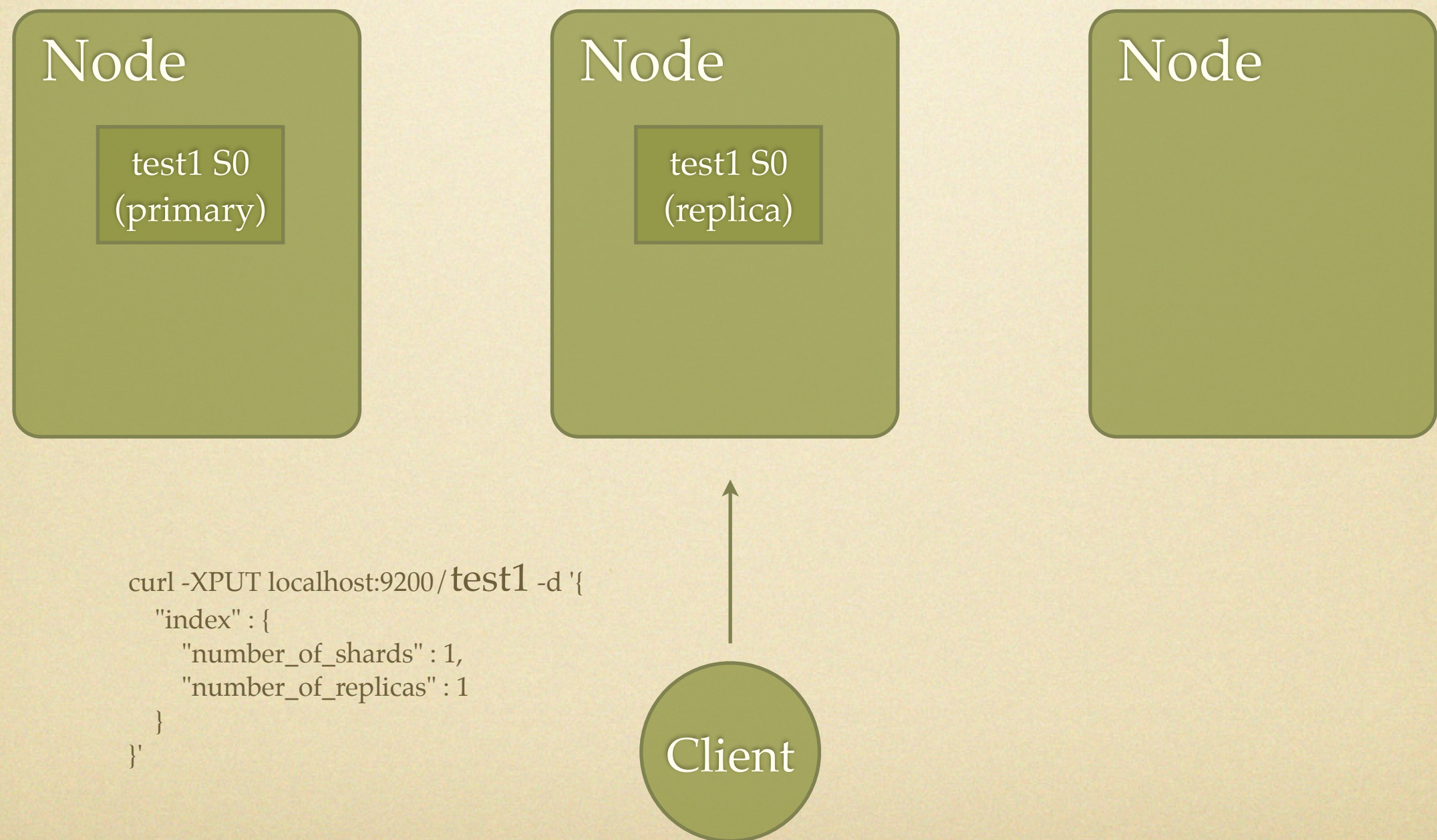
Node

Node

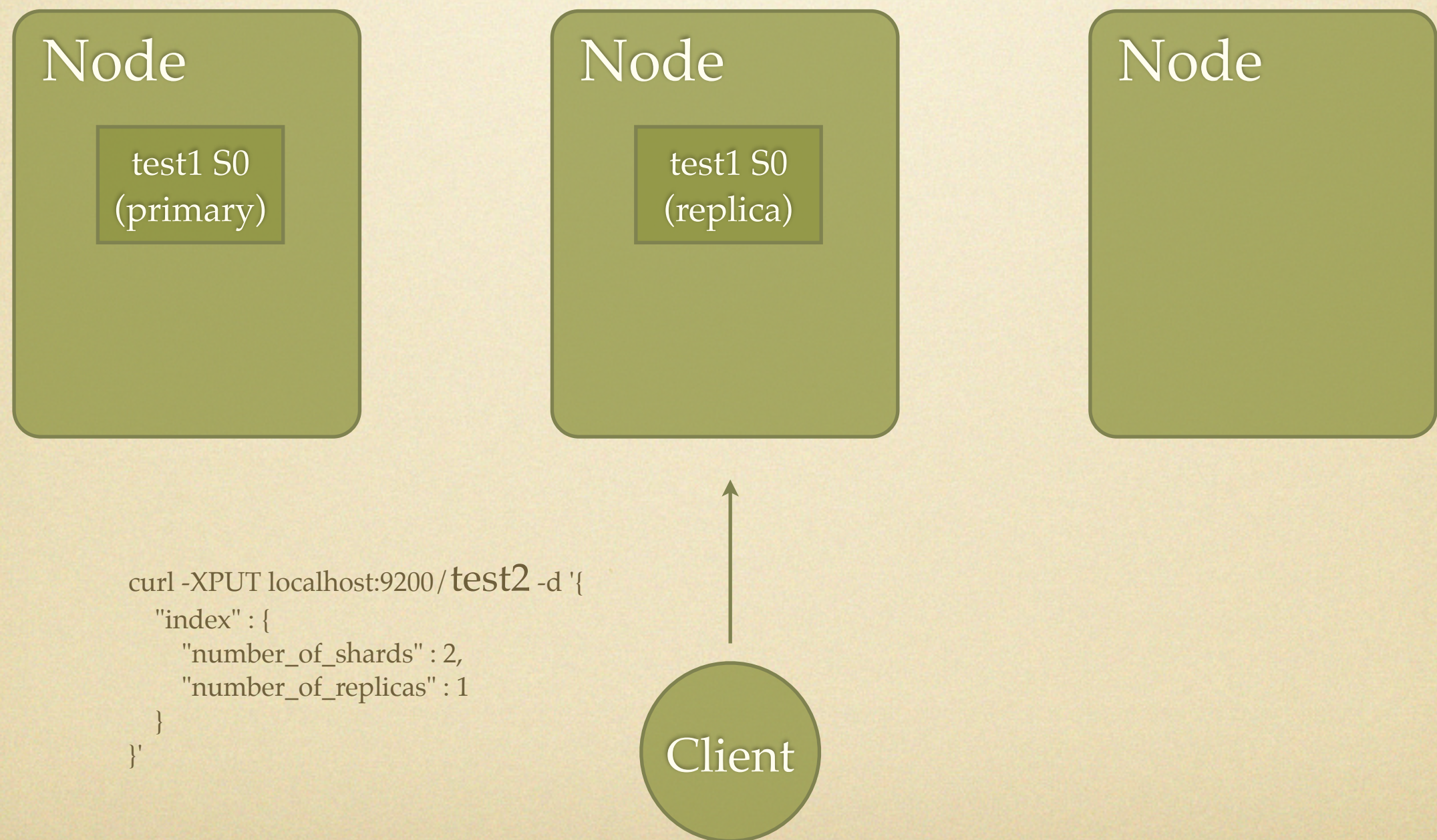
```
curl -XPUT localhost:9200/test1 -d '{  
  "index": {  
    "number_of_shards": 1,  
    "number_of_replicas": 1  
  }  
'
```

Client

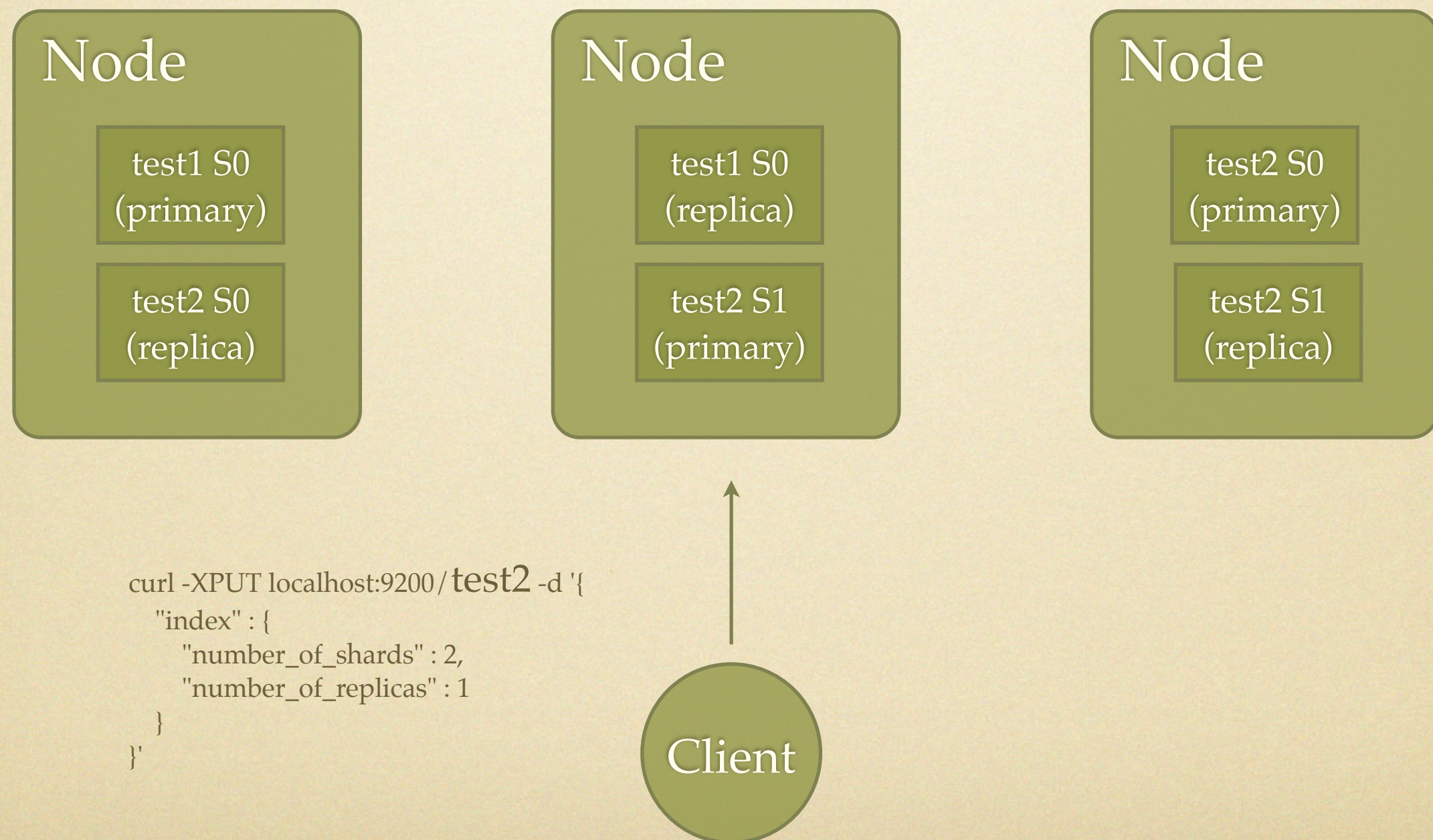
multi tenancy - indices



multi tenancy - indices



multi tenancy - indices



multi tenancy - indices

- Search against specific index
 - `curl localhost:9200 / test1 / _search`
- Search against several indices
 - `curl localhost:9200 / test1,test2 / _search`
- Search across all indices
 - `curl localhost:9200 / _search`
- Can be simplified using aliases

transaction log

- Indexed / deleted doc is fully persistent
 - No need for a Lucene IndexWriter#commit
- Managed using a transaction log / WAL
- Full single node durability (kill dash 9)
- Utilized when doing hot relocation of shards
- Periodically “flushed” (calling IW#commit)

many more...

(dist. related)

- Custom routing when indexing and searching
- Different “search execution types”
 - dfs, query_then_fetch, query_and_fetch
- Complete non blocking, event IO based communication (no blocking threads on sockets, no deadlocks, scalable with large number of shards / replicas)

Thanks

- Shay Banon, twitter: @kimchy
- elasticsearch
 - <http://www.elasticsearch.org/>
 - twitter: @elasticsearch
 - github: <https://github.com/elasticsearch/elasticsearch>