

99 Problems, But The Search Ain't One

Andrei Zmievski • ConFoo • Mar 9, 2011

who am i?

* `curl http://localhost:9200/speaker/info/andrei`

```
{“name”: “Andrei Zmievski”,  
  “works”: “Analog Co-op”,  
  “projects”: [“PHP”, “PHP-GTK”, “Smarty”, “Unicode/i18n”],  
  “likes”: [“coding”, “beer”, “brewing”, “photography”],  
  “twitter”: “@a”,  
  “email”: “andrei@zmievski.org”}
```

what is elasticsearch?

- * a search engine for the NoSQL generation
 - * domain-driven
 - * distributed
 - * RESTful
 - * Hitchhiker's Guide to the Galaxy (no, really)

document model

- * document-oriented
- * JSON-based
- * schema-free

engine

- * based on Lucene
- * multi-tenancy
- * distributed, out of the box

3 easy steps

1. index

request

```
curl -XPOST http://localhost:9200/conf/speaker/1 -d '{
  "name": "Andrei Zmievski",
  "talk": "99 Problems, but the Search Ain't One",
  "likes": ["coding", "beer", "photography"],
  "twitter": "a",
  "height": 187
}'
```

response

```
{
  "ok": true
  "_index": "conf"
  "_type": "speaker"
  "_id": "1"
}
```

2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
    {
      "name": "Andrei Zmievski",
      "talk": "99 Problems, but the Search Ain't One",
      "likes": ["coding", "beer", "photography"],
      "twitter": "a",
      "height": 187
    }
  } ] } }
```

2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
    {
      "name": "Andrei Zmievski",
      "talk": "99 Problems, but the Search Ain't One",
      "likes": ["coding", "beer", "photography"],
      "twitter": "a",
      "height": 187
    }
  } ] } }
```



total number of hits

2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
      {
        "name": "Andrei Zmievski",
        "talk": "99 Problems, but the Search Ain't One",
        "likes": ["coding", "beer", "photography"],
        "twitter": "a",
        "height": 187
      }
    } ]
  }
}
```

the index of the doc



2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
      {
        "name": "Andrei Zmievski",
        "talk": "99 Problems, but the Search Ain't One",
        "likes": ["coding", "beer", "photography"],
        "twitter": "a",
        "height": 187
      }
    } ]
  }
}
```

the type of the doc



2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
      {
        "name": "Andrei Zmievski",
        "talk": "99 Problems, but the Search Ain't One",
        "likes": ["coding", "beer", "photography"],
        "twitter": "a",
        "height": 187
      }
    } ] } }
```

the id of the doc



2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
      {
        "name": "Andrei Zmievski",
        "talk": "99 Problems, but the Search Ain't One",
        "likes": ["coding", "beer", "photography"],
        "twitter": "a",
        "height": 187
      }
    } ] } }
```

the hit score



2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
      {
        "name": "Andrei Zmievski",
        "talk": "99 Problems, but the Search Ain't One",
        "likes": ["coding", "beer", "photography"],
        "twitter": "a",
        "height": 187
      }
    } ]
  }
}
```

the original doc contents



2. search

request

```
curl http://localhost:9200/conf/speaker/_search?q=beer
```

response

```
{ "took" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 0.5908709,
    "hits" : [ {
      "_index" : "conf",
      "_type" : "speaker",
      "_id" : "1",
      "_score" : 0.5908709,
      "_source" :
      {
        "name": "Andrei Zmievski",
        "talk": "99 Problems, but the Search Ain't One",
        "likes": ["coding", "beer", "photography"],
        "twitter": "a",
        "height": 187
      }
    } ]
  }
}
```

the execution time



3. profit

✱ that's up to you

demo

distributed model

- * provides:
 - * performance
 - * resiliency (high-availability)

shards

- * a portion of the document space
- * each one is a separate Lucene index
 - * thus, many per-index settings are available
- * document is sharded by its `_id` value
 - * but can be assigned (routed) to a shard deterministically

zero-conf discovery

- * zen (multicast and unicast)
- * cloud (EC2 via API)

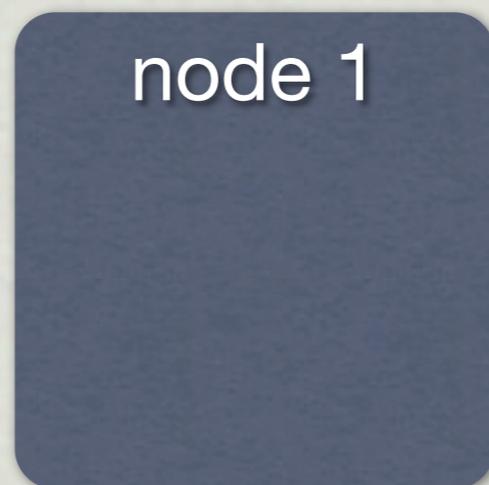
auto-routing

- * master node:
 - * maintains cluster state
 - * reassigns shards if nodes leave/join cluster
- * any node can process the search request
- * the query is handled via scatter-gather mechanism

replicas

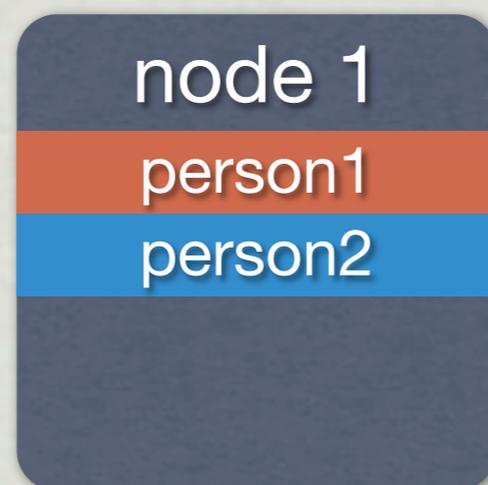
- * each shard can have 1 or more replicas
- * # of replicas can be updated dynamically after index creation
- * replicas can be used for querying in parallel

shard allocation



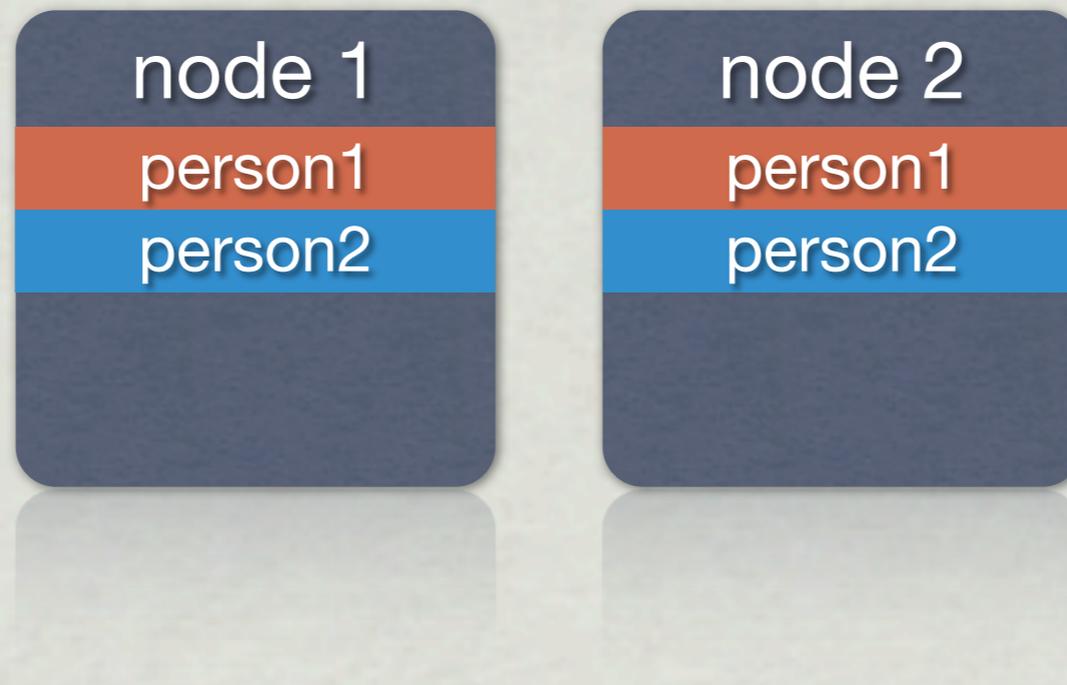
start with a single node

shard allocation



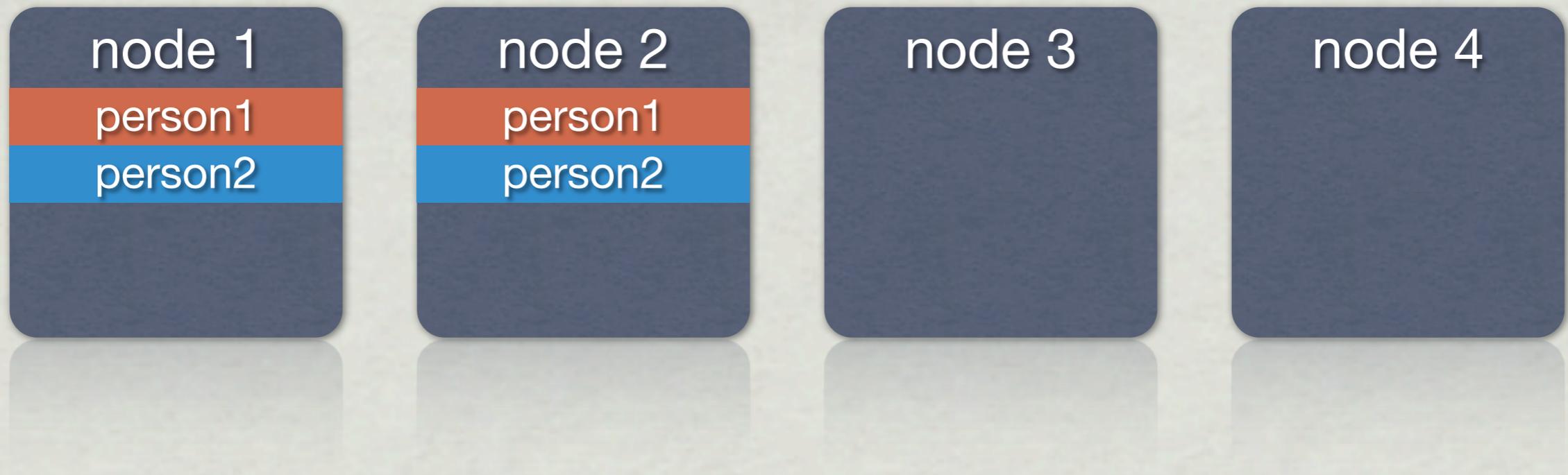
```
PUT /person {  
  "index": {  
    "number_of_shards": 2,  
    "number_of_replicas": 1  
  }  
}
```

shard allocation



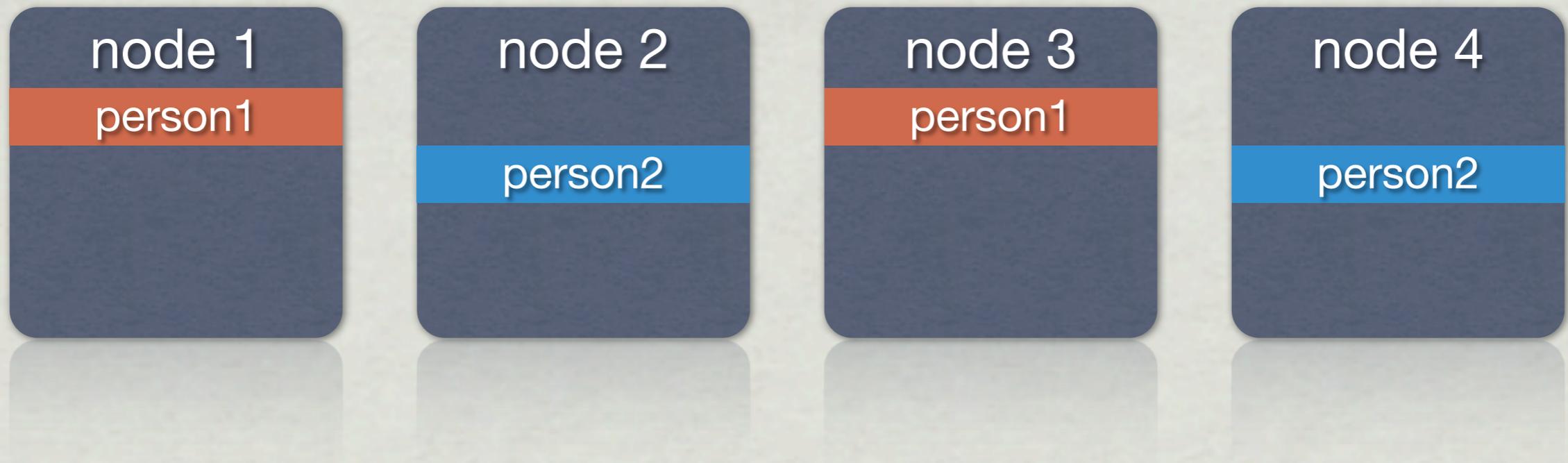
start the second node

shard allocation



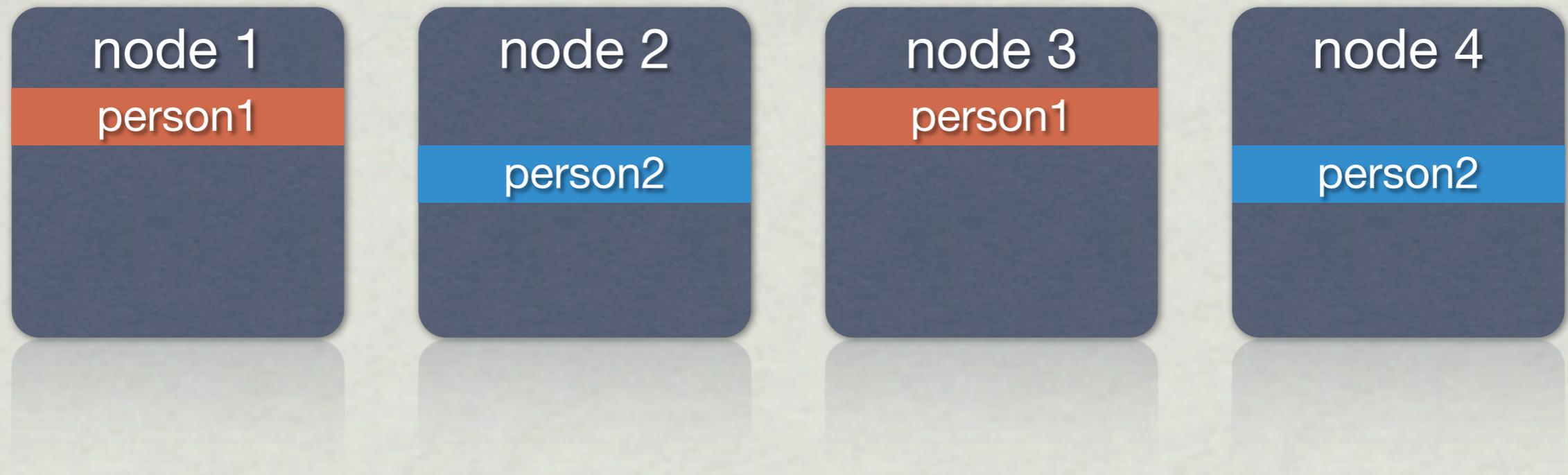
start 2 more nodes

shard allocation



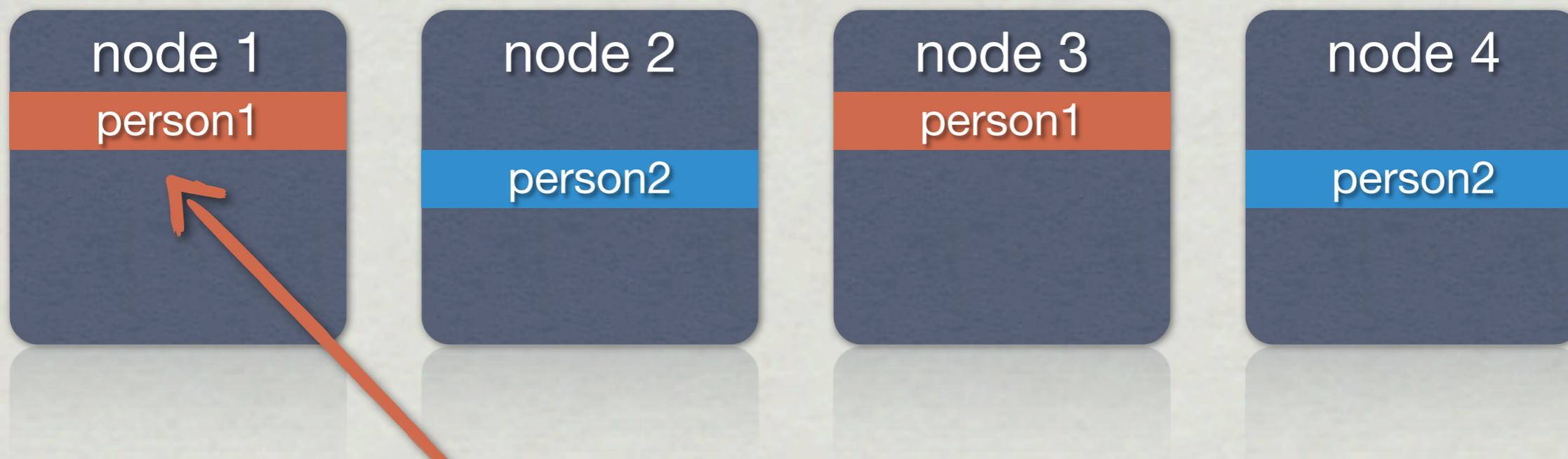
start 2 more nodes

document sharding



PUT /person/info/1
{ ... }

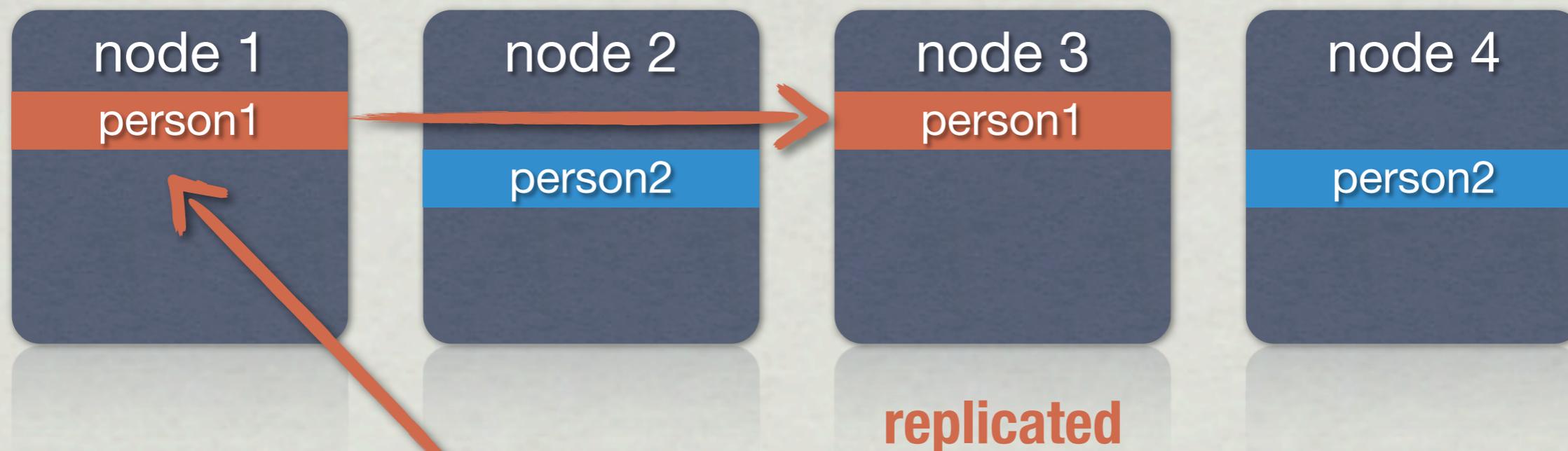
document sharding



hashed to shard 1

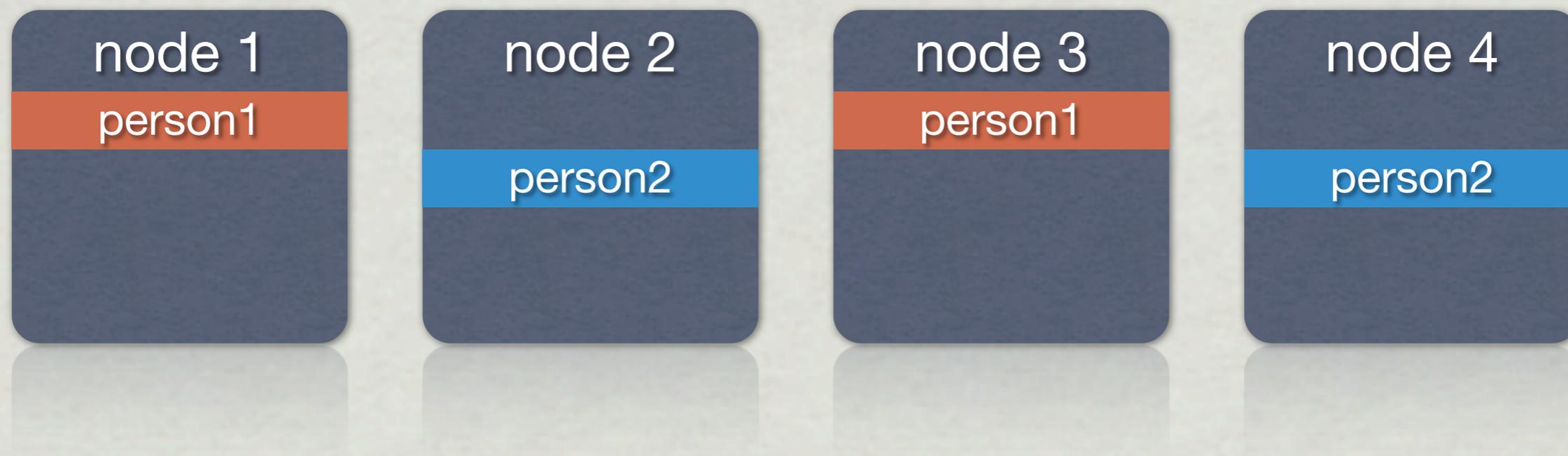
PUT /person/info/1
{ ... }

document sharding



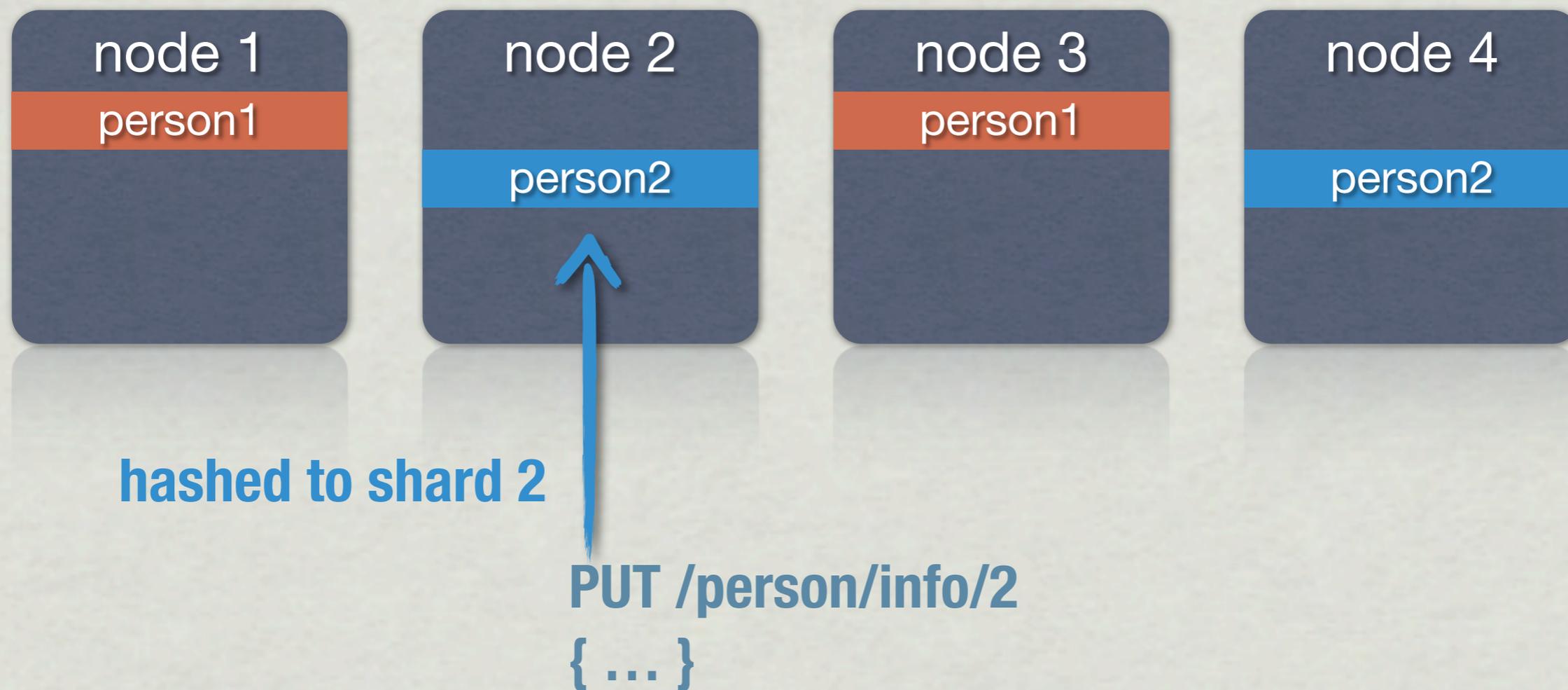
PUT /person/info/1
{ ... }

document sharding

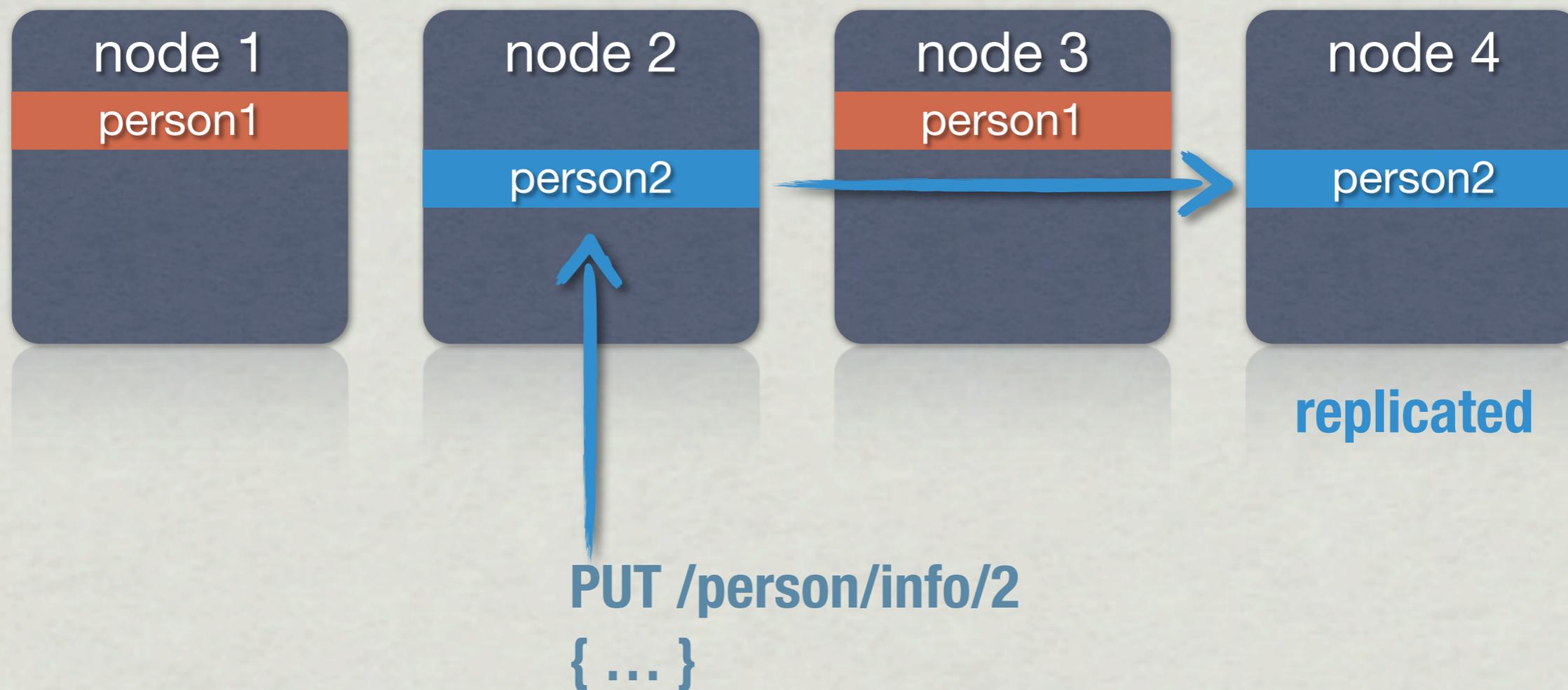


PUT /person/info/2
{ ... }

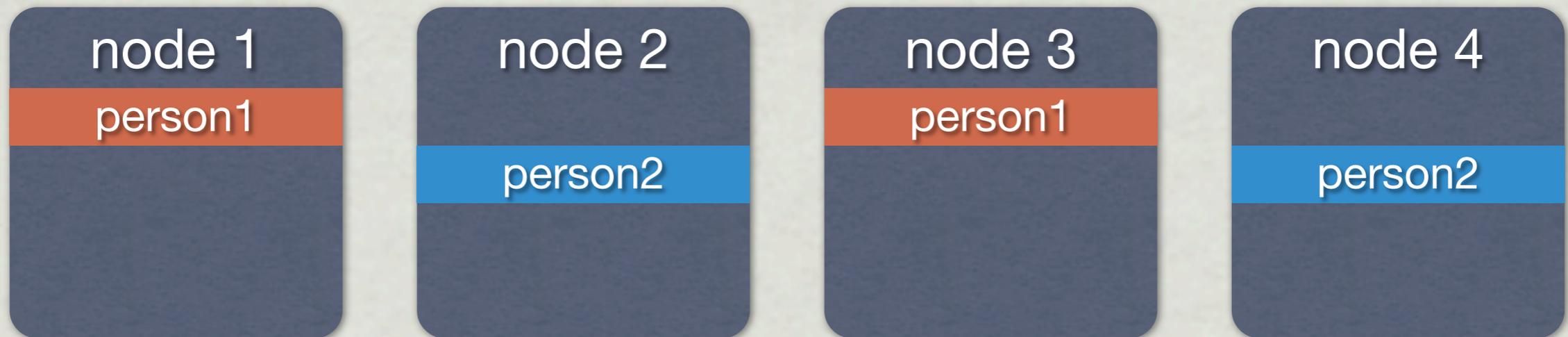
document sharding



document sharding

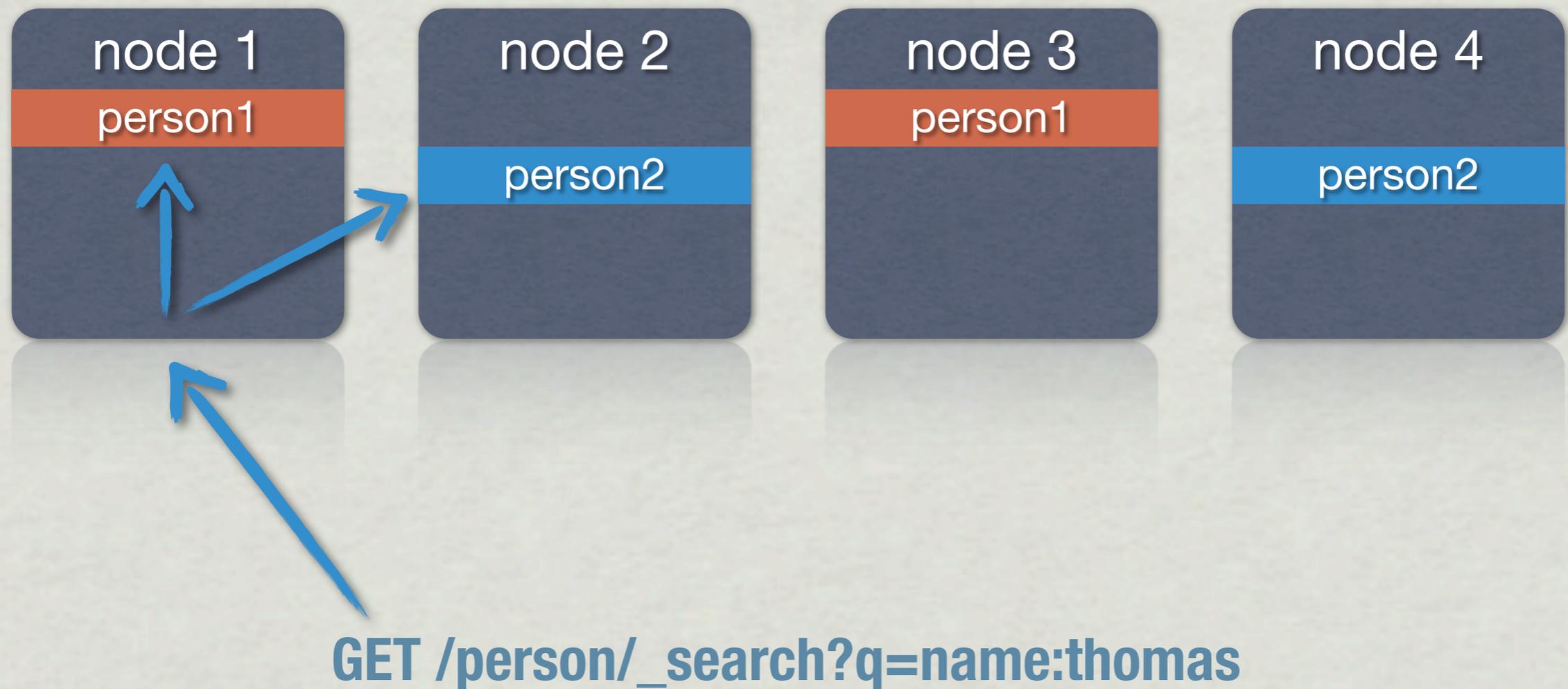


scatter-gather

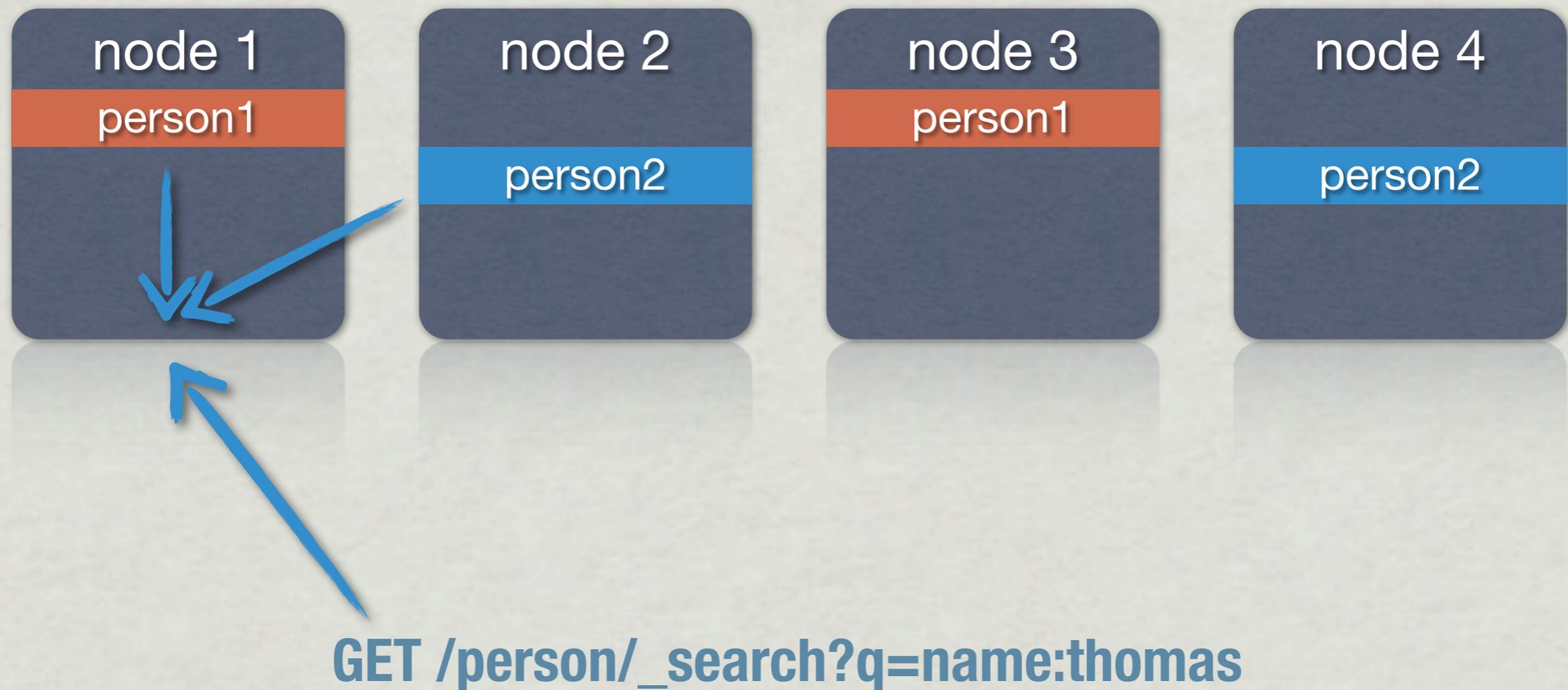


GET /person/_search?q=name:thomas

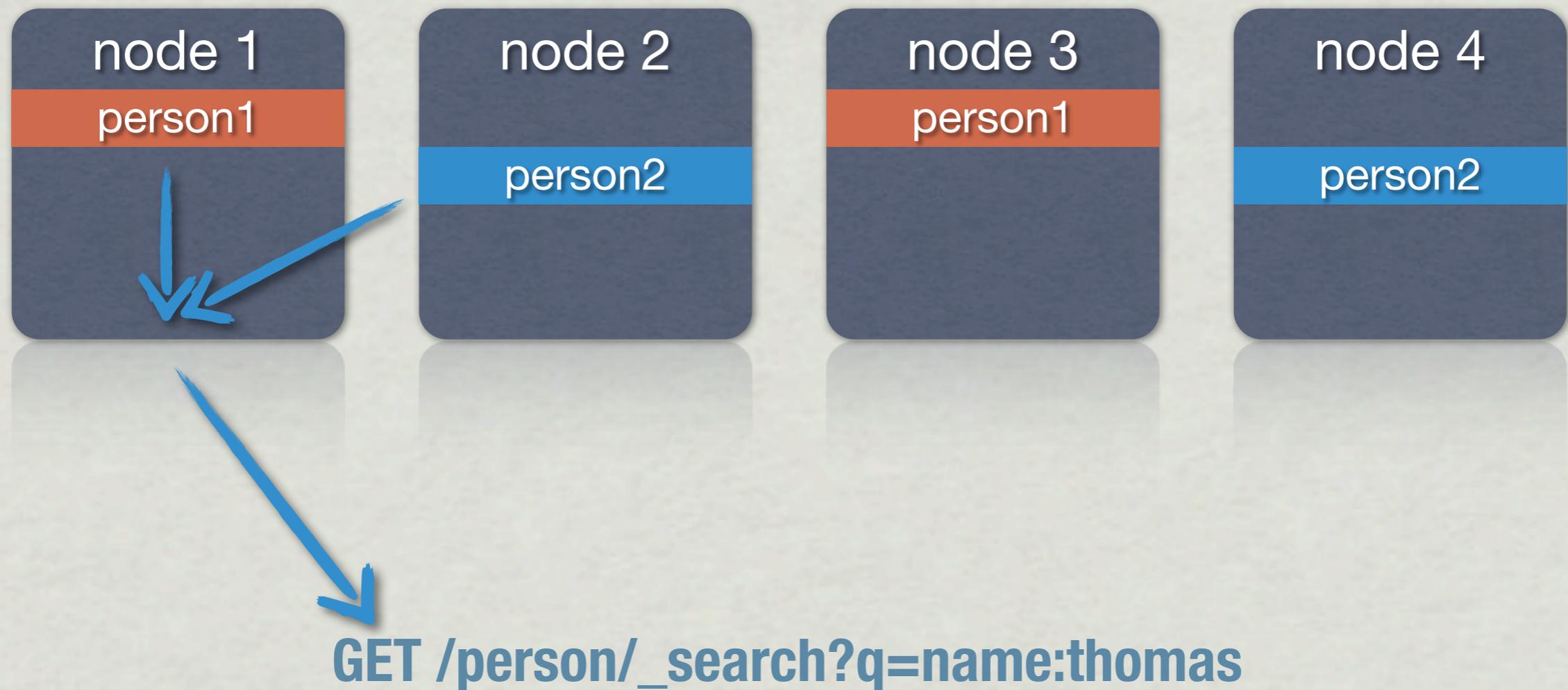
shard allocation



shard allocation



shard allocation



demo

transactional model

- * per-document consistency
- * no need to commit/flush
- * uses write-ahead transaction log
- * write consistency (W) can be controlled
 - * one, quorum, or all

(near) real-time search

- * 1 second refresh rate by default
- * `_refresh` API also

index storage

- * node data considered transient
- * can be stored in local file system, JVM heap, native OS memory, or FS & memory combination
- * persistent storage requires a gateway

gateways

- * persistent store for cluster state and indices
- * asynchronous, translog-based write strategy
- * allows full recovery if a cluster restart is needed
- * supported gateways:
 - * local
 - * shared FS
 - * Hadoop via HDFS
 - * S3

mapping

- * describes document structure to the search engine
- * automatically created with sensible defaults
- * explicit mapping can be provided (generally, a good idea)
- * can run into merge conflicts

mapping

- * important meta fields:

- * `_source`

- * `_all`

- * there are more

mapping types

- * simple:

- * string, integer/long, float/double, boolean, and null)

- * complex:

- * array, object

sample mapping

document

```
{"user":      "derick",  
 "title":    "Don't Panic",  
 "tags":     ["profiling", "debugging", "php"],  
 "postDate": "2010-12-22T17:14:12",  
 "priority": 2}
```

mapping

```
{"post": {  
  "properties" : {  
    "user":      {"type": "string", "index": "not_analyzed"},  
    "message":   {"type": "string", "boost": 1.5},  
    "tags":      {"type": "string", "include_in_all": "no"},  
    "postDate"  : {"type": "date", "store": "no"},  
    "priority"  : {"type": "integer"}  
  }  
}}
```

analyzers

- * break down (tokenize) and normalize fields during indexing and query strings at search time
- * analyzer = tokenizer + token filters (0 or more)

```
Standard Analyzer =  
  Standard Tokenizer +  
    Standard Token Filter +  
    Lowercase Token Filter +  
    Stop Token Filter
```

analyzers

- * analyzers, tokenizers, and filters can be customized

elasticsearch.yml

```
index:
  analysis:
    analyzer:
      eu1ang:
        type: custom
        tokenizer: standard
        filter: [standard, lowercase, stop,
                asciifolding, porterStem]
```

mapping

```
...
"title": {"type": "string", "analyzer": "eu1ang"},
...
```

API

API conventions

- * append `?pretty=true` to get readable JSON
- * boolean values: `false/0/off` = false, rest is true
- * JSONP support via `callback` parameter

API structure

- * `http://host:port/[index]/[type]/[_action/id]`
- * GET `http://es:9200/_status`
- * GET `http://es:9200/twitter/_status`
- * POST `http://es:9200/twitter/tweet/1`
- * GET `http://es:9200/twitter/tweet/1`

API structure

- * [http://host:port/\[index\]/\[type\]/\[_action/id\]](http://host:port/[index]/[type]/[_action/id])
- * GET http://es:9200/twitter/tweet/_search
- * GET http://es:9200/twitter/user/_search
- * GET http://es:9200/twitter/tweet,user/_search
- * GET http://es:9200/twitter,facebook/_search
- * GET http://es:9200/_search

API query example

```
{
  "query": {
    "filtered": {
      "query": {
        "query_string": {
          "query": "foo bar",
          "default_operator": "AND",
          "fields": ["title", "description"],
          "boost": 2.0
        }
      },
      "filter": {
        "range": {"date": {"gt": "2011-03-09"}}
      }
    }
  },
  "from": 10,
  "size": 10
}
```

API {core}

- * index

- * bulk

- * delete

- * delete by query

- * get

- * count

- * search

- * query

- * from/size paging

- * sort

- * highlighting

- * selective fields

API {indices}

- * create
- * delete
- * open/close
- * get/put/delete
mapping
- * refresh
- * optimize
- * snapshot
- * update settings
- * analyze
- * status
- * flush

Query DSL

- * term / terms
- * range
- * prefix
- * bool
- * fuzzy
- * wildcard
- * query_string
- * default_operator
- * analyzer
- * phrase_slop
- * etc

filters

- ✱ share some similar features with queries (term, range, etc)
- ✱ why use a filter?

filters

- * faster than queries
- * cached (depends on the filter)
 - * the cache is used for different queries against the same filter
- * no scoring
- * more useful ones: term, terms, range, prefix, and, or, not, exists, missing, query

facets

- * provide aggregated data based on the search request
- * terms, histogram, date histogram, range, statistical, and more

geo search

- * implemented as filters (and a facet)
 - * `geo_distance`
 - * `geo_bounding_box`
 - * `geo_polygon`

interfaces

- * REST
- * Java / Groovy
- * Language clients (REST/Thrift):
 - * pyes, PHP (standalone and symfony), Ruby, Perl
- * Flume sink implementation

data import

- * ES is not the primary data store (usually)
- * to import/synchronize data:
 - * write an agent (Gearman, message queues, etc)
 - * use rivers (CouchDB, RabbitMQ, Twitter)

10 more features

- * versioning
- * index aliases
- * parent/child docs
- * scripting
- * dynamic mapping templates
- * load balancing nodes
- * plugins
- * more_like_this
- * multi_field mapping
- * percolation

References

- * <http://github.com/elasticsearch/elasticsearch>
- * <http://groups.google.com/a/elasticsearch.com/group/users/>
- * IRC: #elasticsearch on irc.freenode.net
- * twitter: @elasticsearch