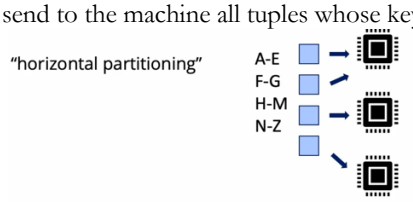
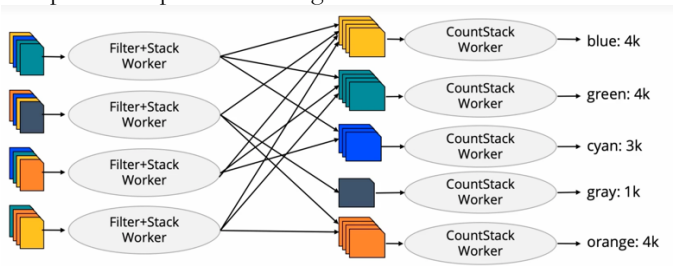
SQL: scale by typically getting bigger servers, scaling across different servers is difficult, no mechanism for parallelism
NoSQL: sharding, mapreduce

Sharding: distribute data across nodes/servers

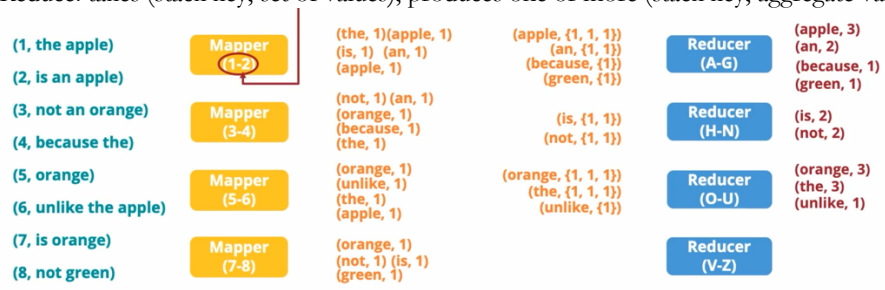
- map key range to a certain machine
- send to the machine all tuples whose key is in the range



MapReduce: spread computation among clusters



- MapReduce has to deal with workers failing, how to keep workers busy, how much workers should work, and how to monitor progress
- Two type of workers:
  - 1) take input item and produce output item for the stacks
    Map: takes (item key, value), produces one or more (stack key, value)
  - 2) take stacks and aggregate results to produce outputs on a per-stack basis
    Reduce: takes (stack key, set of values), produces one or more (stack key, aggregate value)



Transactions in Distributed Systems

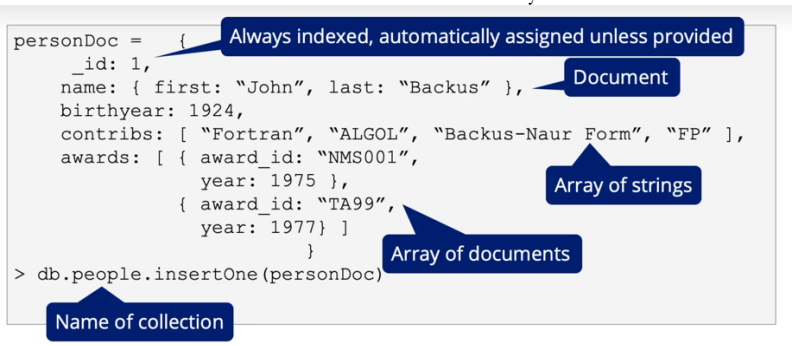
1) Two Phase Commit for Consistency
   a. Phase 1: ask each node if they are ready to commit
   b. Phase 2: if all nodes are ready to commit, coordinator signals commit, otherwise aborts
2) BASE
   a. Basically available: available most of the time
   b. Soft state: state may change periodically
   c. Eventual consistency: database will eventually become consistent over time

MongoDB

- Document oriented, collections of JSON format documents
- Joining is difficult but distributed, parallel query processing great for replication and sharding
- "Collection" at a time processing
- "Database" holds set of collections, "Collections" hold array of documents, "Documents" holds set of key-value pair

RDBMS	MongoDB
Table	Collection
Row(s)	JSON Document
Attribute	Key
Join	Embedding & Linking

Documents can have embedded documents or array of embedded documents




```
personDoc =   {
     _id: 1,
    name: { first: "John", last: "Backus" },
    birthyear: 1924,
    contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
    awards: [ { award_id: "NMS001",
                year: 1975 },
              { award_id: "TA99",
                year: 1977} ]
                          }
> db.people.insertOne(personDoc)
```

CRUD: Create, Read, Update, Delete

- Insert: db.collection.insertOne(), db.collection.insertMany()
- Delete: db.collection.deleteOne(), db.collection.deleteMany() (by default all documents in a collection, collection remains)
- Delete: db.collection.drop() (removes a collection)
- Update: db.collection.updateOne()

```
db.people.updateMany( {birthyear: 1924}, {$set: {birthyear: 1925}})
```

```
{ _id: 1,
  name: { first: "John", last: "Backus" },
  birthyear: 1925,
  contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
  awards: [ { award_id: "NMS001",
              year: 1975 },
            { award_id: "TA99",
              year: 1977} ]}
```

```
db.people.updateMany( {birthyear: 1924}, {$set: {type: "Deceased"}})
```

```
{ _id: 1,
  name: { first: "John", last: "Backus" },
  birthyear: 1924,  type: "Deceased",
  contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
  awards: [ { award_id: "NMS001",
              year: 1975 },
            { award_id: "TA99",
              year: 1977} ]}
```

Indexes: Unique index on _id field

- Additional indexes: db.collection.createIndex({birthyear: -1, name: "YOB-Index"}), give index to birthyear and name it YOB-Index
- Delete indexes: db.collection.dropIndex()
- Validator: specifies a schema that is required

Simple Find: queries return a database cursor which allows us to limit, skip, sort results

```
mydoc={
      _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
      author : "Kevin",
      date : new Date("February 2, 2012"),
      text : "About MongoDB...",
      birthyear:  1980,
      tags : [ "tech", "databases" ]
      }

> db.posts.insert(mydoc)
```

- Return all collections: db.posts.find()
- Return collections based on condition: db.posts.find({author: "Kevin", birthyear: 1980})
- Pretty print: db.posts.find(…).pretty()

Comparison operators: $lt, $lte, $gt, $gte

```
db.posts.find({birthyear: {$gte: 1970, $lte: 1990}})
```

Negation: $ne

```
db.posts.find({birthyear: {$ne: 1982}})
```

$in (single key), $or (different keys)

```
db.posts.find({birthyear: {$in: [1980, 1985]}})
```

```
db.posts.find({$or: [{birthyear: 1982}, {author: "Kevin"}]})
```

1 shows what to display, 0 suppresses, id will always display if not suppressed

```
db.posts.find({}, {author:1, birthyear:1})
```

```
{_id: ObjectId("4c4ba5c0672c685e5e8aabf3"), author: "Kevin",
birthyear:  1980 }
```

```
db.posts.find({}, {_id: 0, author:1})
```

```
{ author: "Kevin" }
```

```
db.posts.find({}, {_id:0, date:0, tags:0})
```

```
{ author: "Kevin", birthyear: 1980, text: "About MongoDB..." }
```

Limit, skip, sort

Limit the number of results to 3

```
db.posts.find(…).limit(3)
```

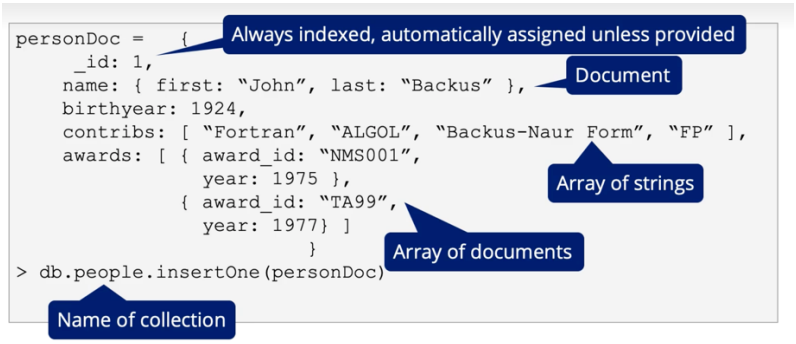
Skip the first three results and return the rest

```
db.posts.find(…).skip(3)
```

Sort the result by author ascending (1) and then title descending (-1)

```
db.posts.find(…).sort({author:1, title: -1})
```

Arrays 1




```
personDoc =   {
     _id: 1,
    name: { first: "John", last: "Backus" },
    birthyear: 1924,
    contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
    awards: [ { award_id: "NMS001",
                year: 1975 },
              { award_id: "TA99",
                year: 1977} ]
                        }
> db.people.insertOne(personDoc)
```

Elements of an array are **ordered**, so the following:

```
db.people.find({contribs:[ "Fortran", "ALGOL", "Backus-Naur Form", "FP"]})
```

would return mydoc, but the following would not:

```
db.people.find({contribs:[ "ALGOL", "Fortran", "Backus-Naur Form", "FP"]})
```

To query without specifying order (returns mydoc):

```
db.people.find({contribs: {$all: ["ALGOL", "Fortran", "Backus-Naur Form",
"FP" ]}})
```

To find documents with an array containing a single tag (returns mydoc):

```
db.people.find({contribs:"Fortran"})
```

You can test the value in an array at an index position:

```
db.people.find({"contribs.0": "Fortran" })
```

or the size of an array:

```
db.people.find({contribs: {$size: 4}})
```

To find people with more than 4 contribs, use $where:

```
db.people.find({$where:'this.contribs.length>4'})
```

Arrays 2

```
db.inventory.insertMany([
   { item: "journal", dim_cm: [ 14, 21 ] },
   { item: "planner", dim_cm: [ 22.85, 30 ] },
   { item: "postcard", dim_cm: [ 10, 15.25 ] }  ])
```

For the latter, one element must be greater than 15, and another (or the same) element must be less than 20

At least one array element satisfies all conditions: $elemMatch: {...}

```
db.inventory.find( { dim_cm: { $elemMatch: { $gt: 22, $lt: 30 } } } )
```

```
{ item: "planner", dim_cm: [ 22.85, 30 ] }
```

Some combination of array elements meets the conditions:

```
db.inventory.find( { dim_cm: { $gt: 15, $lt: 20 } }  )
```

```
{ item: "journal", dim_cm: [ 14, 21 ] },
{ item: "postcard", dim_cm: [ 10, 15.25 ] }  ])
```

Embedded Documents

people:

```
[{ _id: 1,
   name: { first: "John", last: "Smith" },…},
  {_id: 2,
   name: { first: "John", middle: "P.", last: "Smith" },… }]
```

```
[{ _id: 3,
  name: [{ first: "Sally", last: "Smith" },
         { first: "John", last: "Jones" }],… },…]
```

This will return all 3 documents

Find all people with first name "John"

```
db.people.find({"name.first": "John"})
```

Find all people with first name "John" and last name "Smith".

```
db.people.find({"name.first": "John", "name.last": "Smith"})
```

This will return only the first document

With the exact match: Find all people with name "John Smith".

```
db.people.find({"name": {"first": "John", "last": "Smith"}})
```