Map Reduce

Background

- Google deals with very large amounts of data (petabytes)
  - need to process data fairly quickly
  - use very large numbers of commodity machines
  - Cheap nodes fail, especially if you have many
    - Mean time between failures for 1 node = 3 years
    - Mean time between failures for 1000 nodes = 1 day
    - Solution: Build fault-tolerance into the system

- Google developed an infrastructure consisting of
  - the Google distributed file system GFS
  - the MapReduce computational model
- MapReduce
  - functional programming model
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates
- Open source implementation Hadoop from Apache

Map Reduce

- Programming model for indexing and searching large data volumes over computer clusters
- Two Phases, Map and Reduce
  - Map
    - Extract sets of Key-Value pairs from underlying data
    - Potentially in Parallel on multiple machines
  - Reduce
    - Merge and sort sets of Key-Value pairs
    - Results may be useful for other searches

Google MapReduce

- Google’s MapReduce is implemented as a C++ library.
- Operates on commodity hardware and standard networking.
- Input data, intermediate results, and final results are stored in GFS.
- A master scheduler process distributes map, reduce tasks to workers.
- Fault tolerance:
  - The master pings workers periodically.
  - Workers that do not respond are marked as failed.
  - Jobs assigned to failed workers are rerun.
  - Master failure aborts the computation.

Data type: key-value *records*

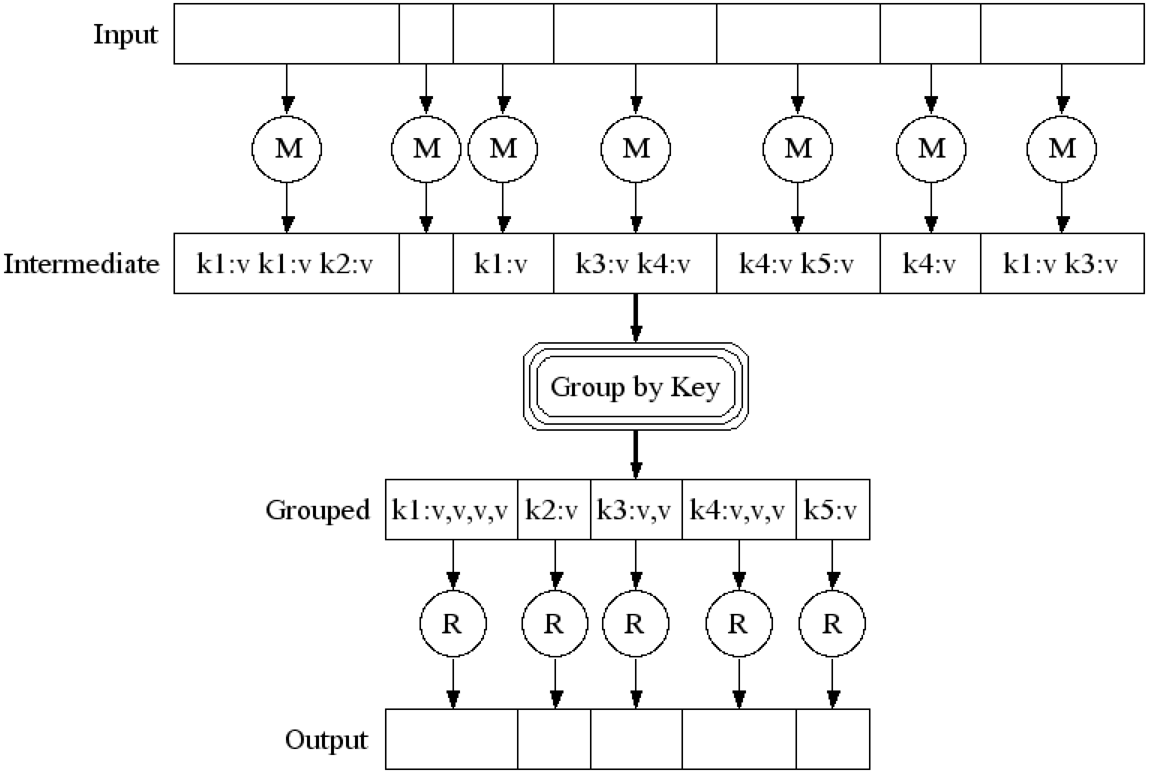
Map function:

$$(K_{in}, V_{in}) \Rightarrow \text{list}(K_{inter}, V_{inter})$$

Reduce function:

$$(K_{inter}, \text{list}(V_{inter})) \Rightarrow \text{list}(K_{out}, V_{out})$$

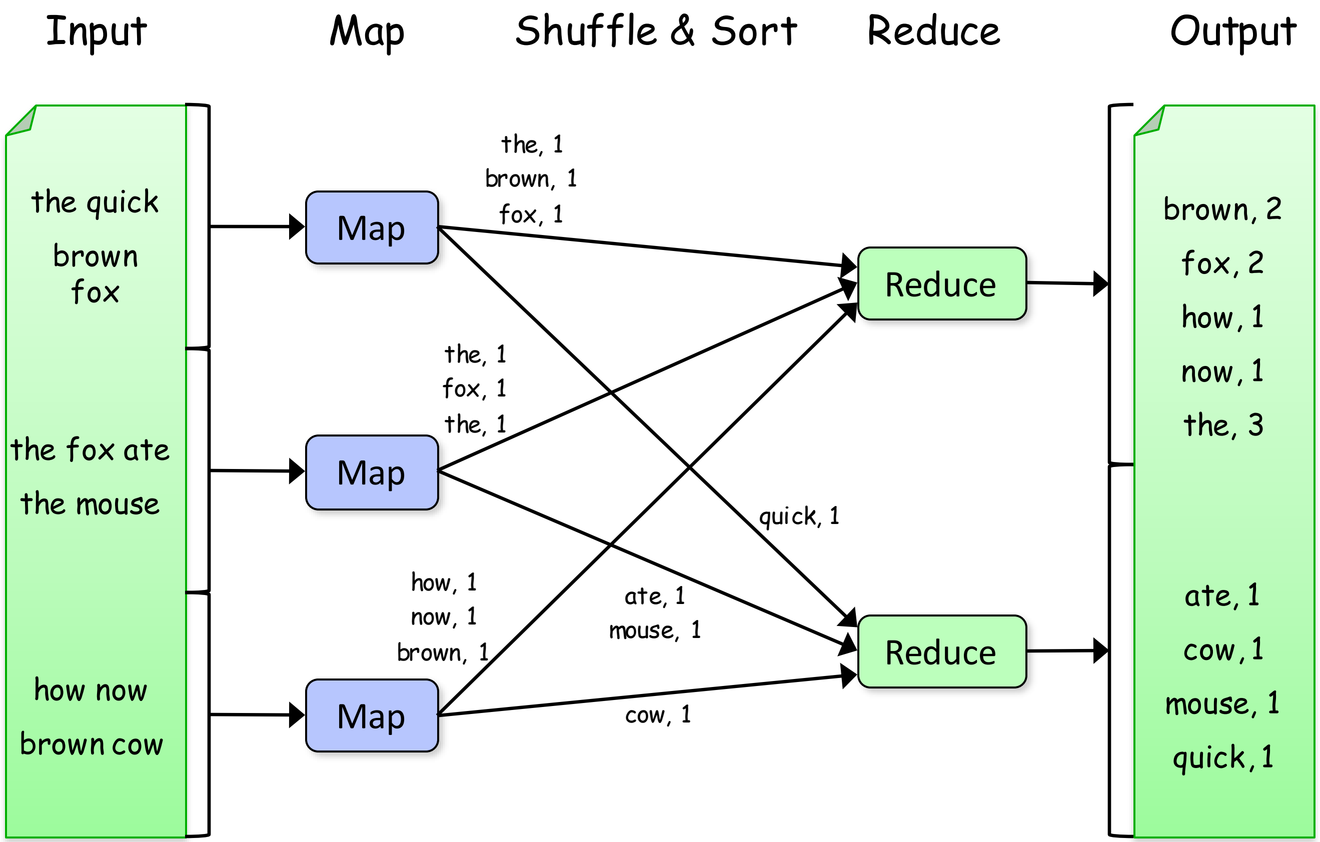
Execution


Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

Example: Word Count

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)

def reducer(key, values):
    output(key, sum(values))
```

Word Count Execution

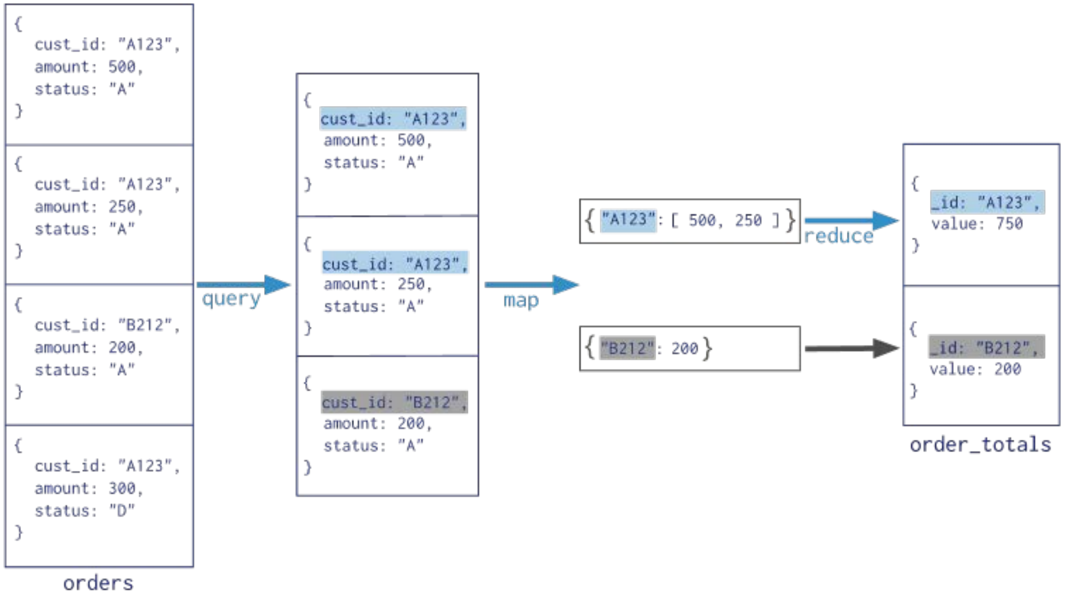


MapReduce Execution Details

- Single *master* controls job execution on multiple *slaves*
- Mappers preferentially placed on same node or same rack as their input block
  - Minimizes network usage
- Mappers save outputs to local disk before serving them to reducers
  - Allows recovery if a reducer crashes
  - Allows having more reducers than nodes

Map Reduce in MongoDB

```
  Collection
      ↓
db.orders.mapReduce(
         map    ──→   function() { emit( this.cust_id, this.amount ); },
         reduce ──→   function(key, values) { return Array.sum( values ) },
                      {
         query  ──→     query: { status: "A" },
         output ──→     out: "order_totals"
                      }
                    )
```



```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
{
  cust_id: "A123",
  amount: 300,
  status: "D"
}
```

orders

```
{
  cust_id: "A123",
  amount: 500,
  status: "A"
}
{
  cust_id: "A123",
  amount: 250,
  status: "A"
}
{
  cust_id: "B212",
  amount: 200,
  status: "A"
}
```

```
{ "A123": [ 500, 250 ] }
{ "B212": 200 }
```

```
{
  _id: "A123",
  value: 750
}
{
  _id: "B212",
  value: 200
}
```

order_totals

Map/Reduce in Mongo

```
db.collection.mapReduce(
        <mapfunction>,
        <reducefunction>,
        {
                out: <collection>,
                query: <>,
                sort: <>,
                limit: <number>,
                finalize: <function>,
                scope: <>,
                jsMode: <boolean>,
                verbose: <boolean>
        }
    )
```

Example: Tickets

```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

Sum(checkout)?

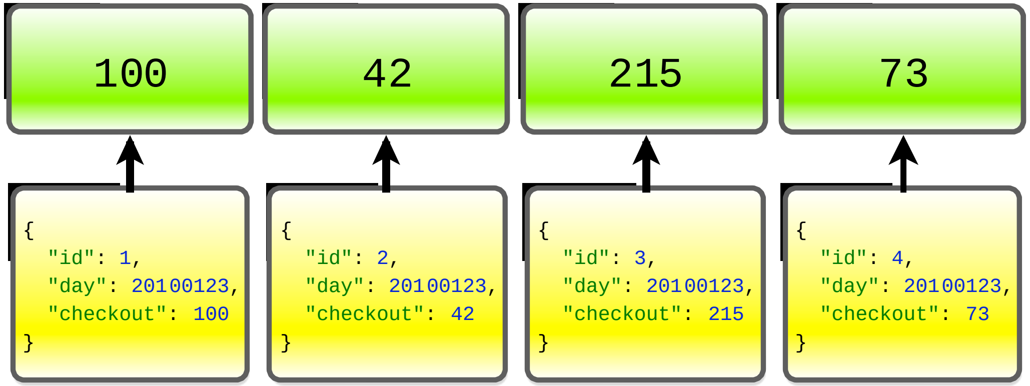
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

Map: emit(checkout)



100

42

215

73

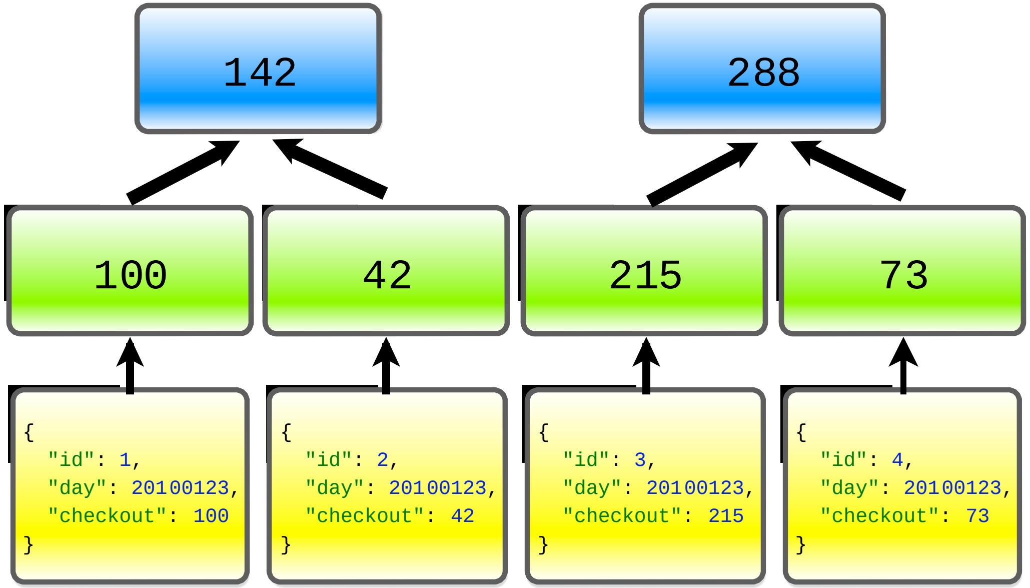
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

Reduce: sum(checkouts)



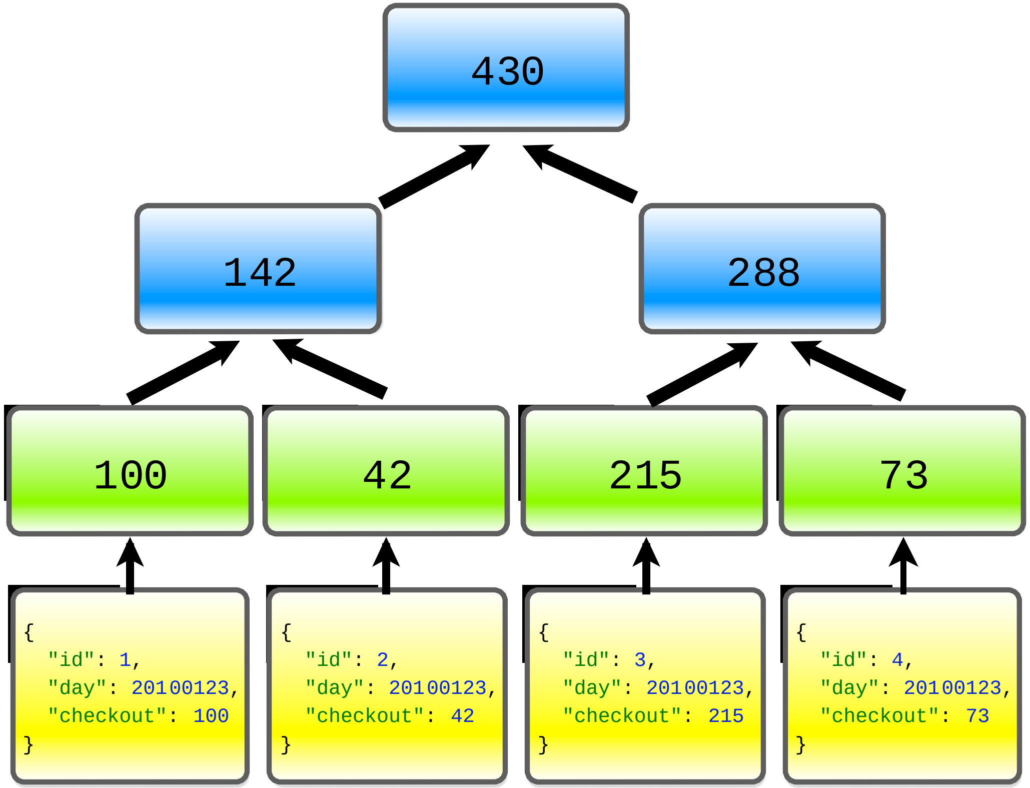
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

Reduce: sum(checkouts)



430

142

288

100

42

215

73

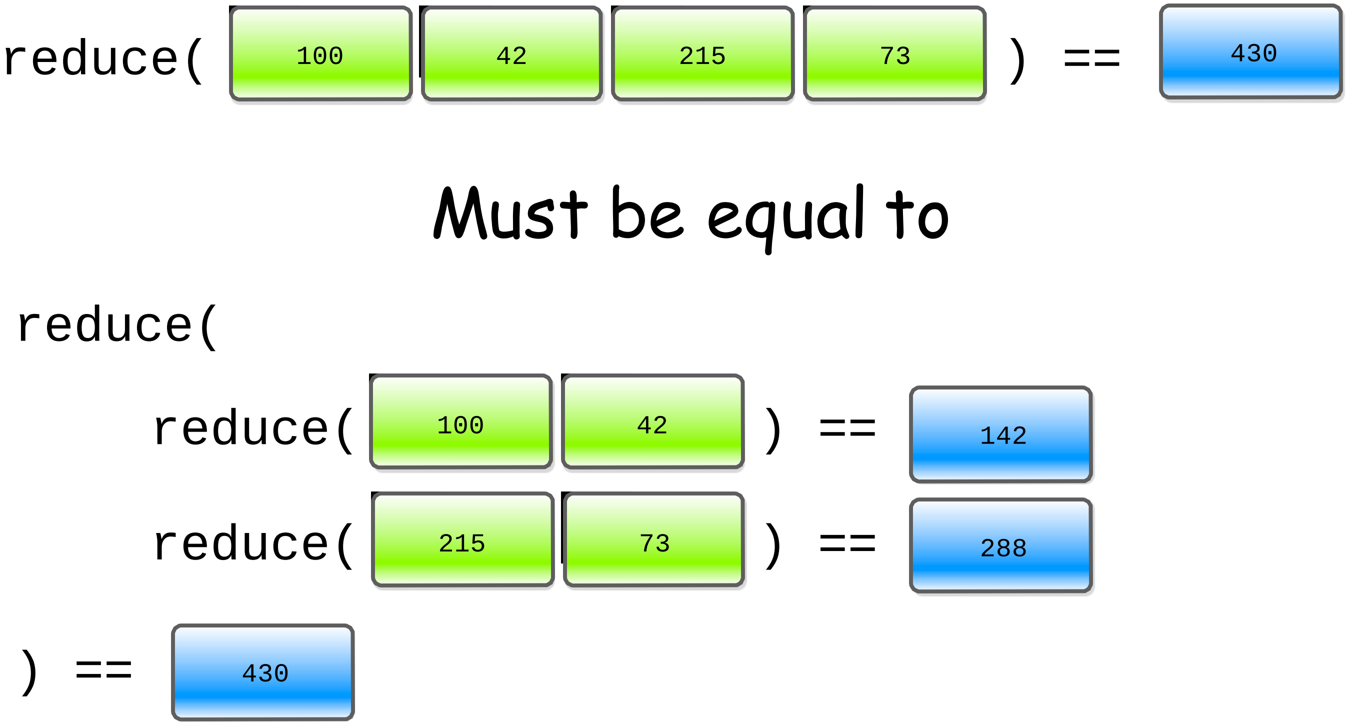
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

Reduce must be associative

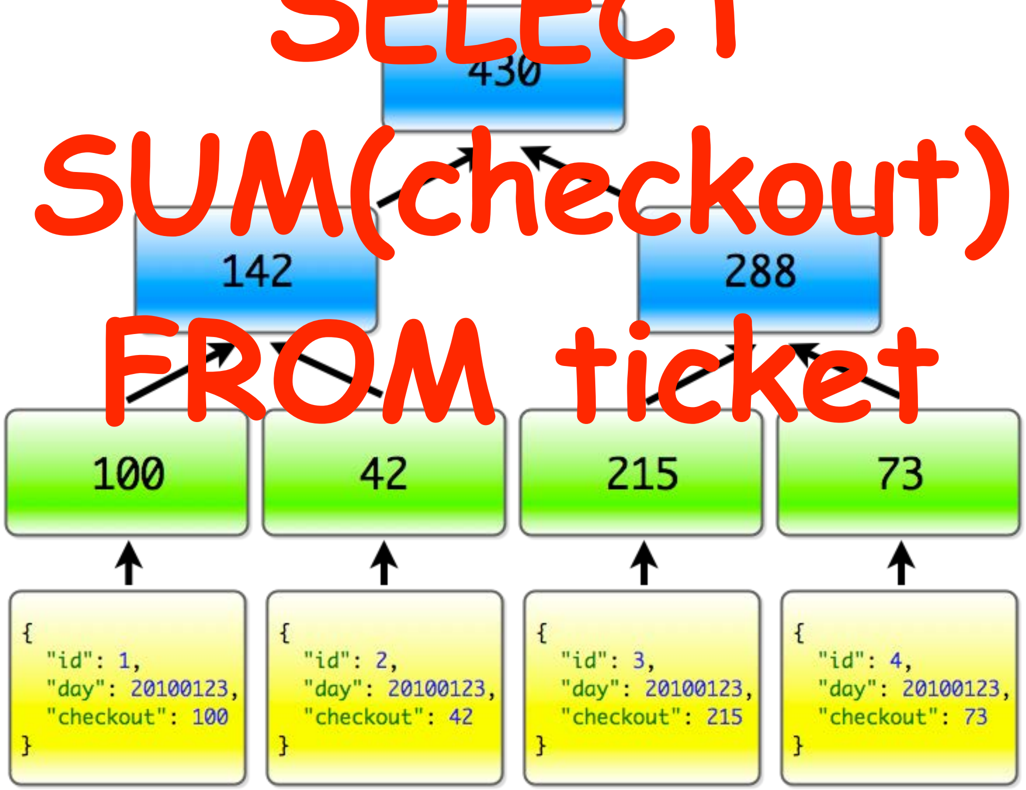


```
reduce( 100 42 215 73 ) == 430
```

Must be equal to

```
reduce(
      reduce( 100 42 ) == 142
      reduce( 215 73 ) == 288
) == 430
```

SELECT SUM(checkout) FROM ticket



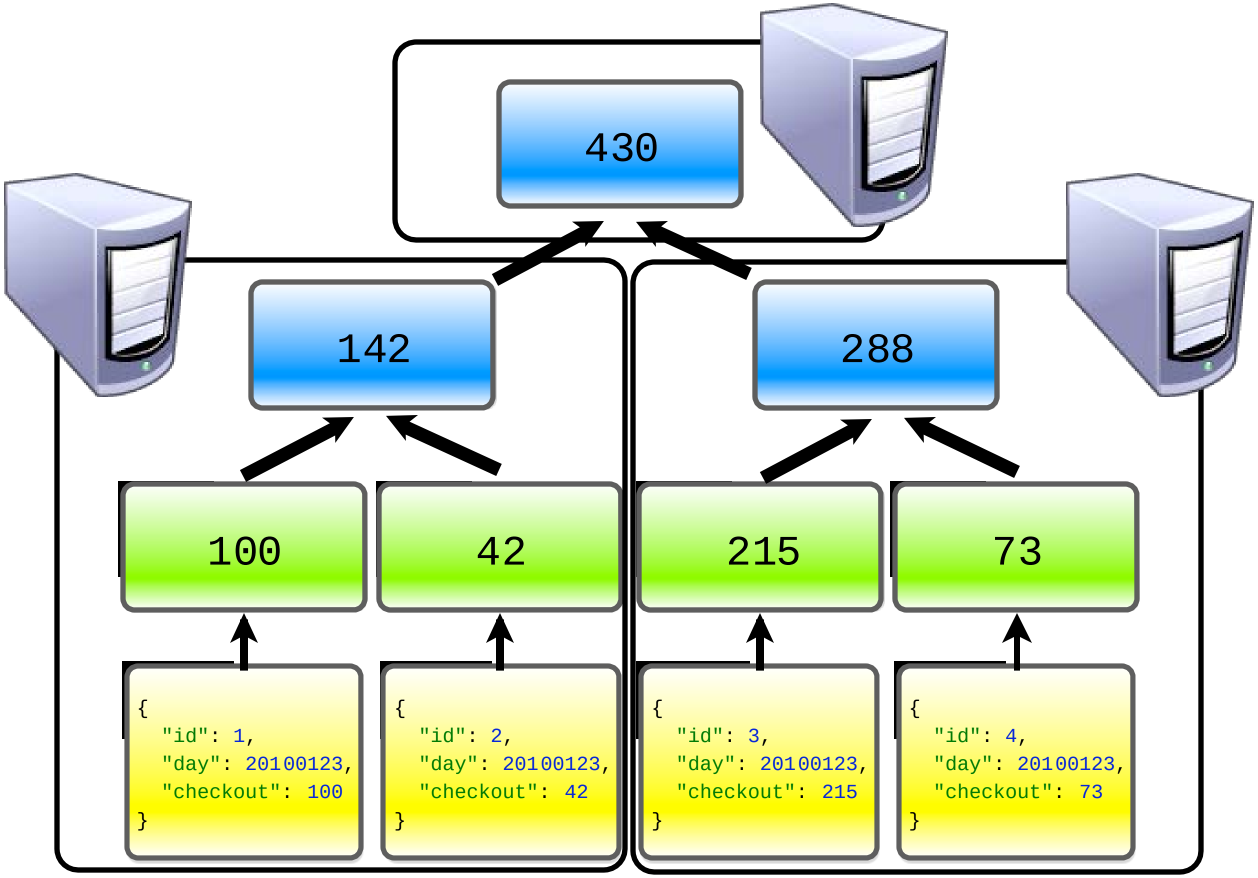
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

Inherently distributed



```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100123,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100123,
  "checkout": 73
}
```

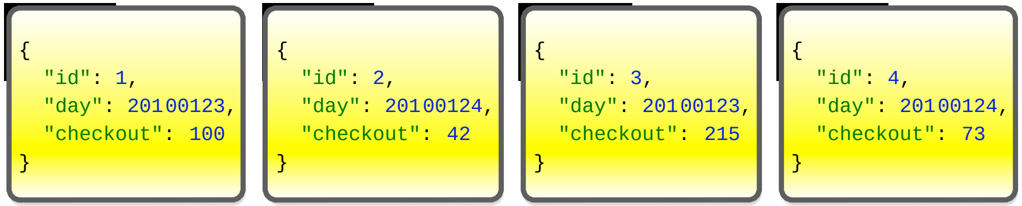
Calculate total checkout

```
#Aggregate alternative
db.tickets.aggregate ({
        "$group": {_id: null, "value": {$sum: "$checkout"}}},
        {$out: "sumOfCheckouts_agg“
        })
```

Persistent Collection

```
#Map-reduce alternative
var map = function() {
        emit(null, this.checkout)
}

var reduce = function (key, values) {
   var sum=0
   for (var idx = 0; idx< values.length; idx++)
        sum+=values[idx];
   return sum;
}

db.tickets.mapReduce (map, reduce, {"out":  "sumOfCheckouts"})

db.sumOfCheckouts.findOne().value
```

Sum(checkout) Group By day



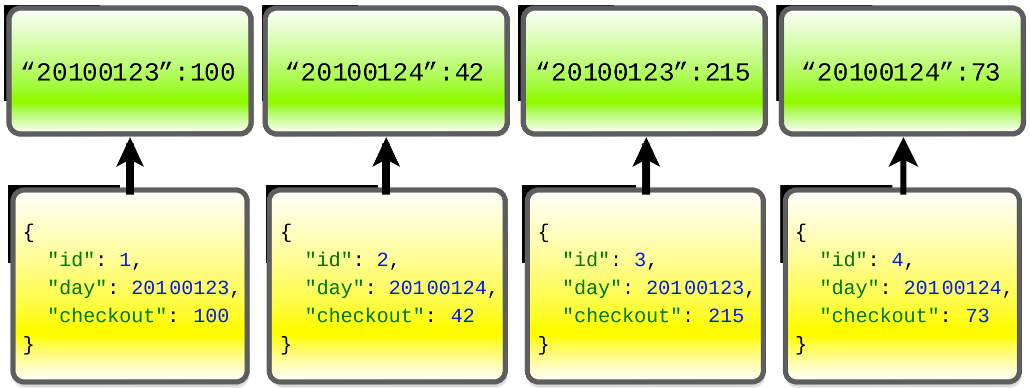
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100124,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100124,
  "checkout": 73
}
```

Map: emit(day,checkout)



“20100123”:100

“20100124”:42

“20100123”:215

“20100124”:73

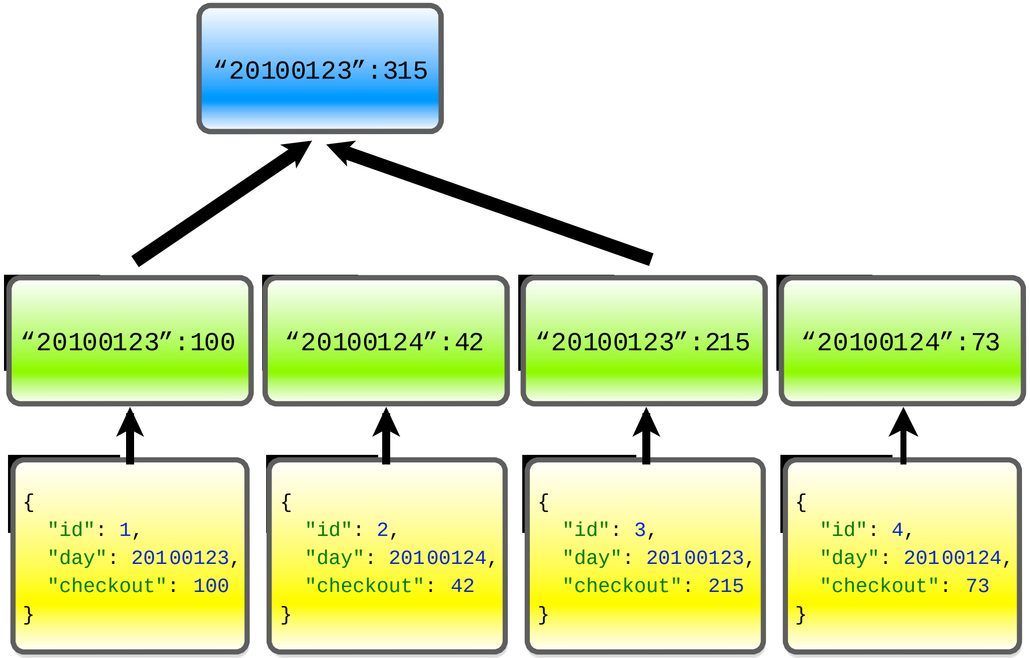
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100124,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100124,
  "checkout": 73
}
```

Reduce: sum(checkouts)



“20100123”:315

“20100123”:100

“20100124”:42

“20100123”:215

“20100124”:73

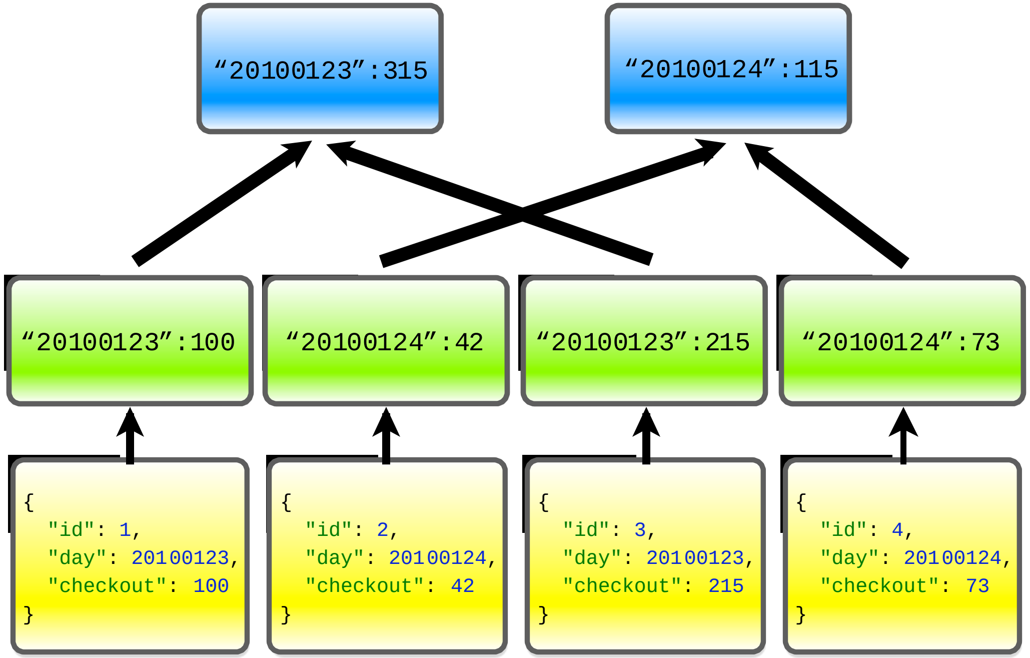
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100124,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100124,
  "checkout": 73
}
```

Reduce: sum(checkouts)



“20100123”:315

“20100124”:115

“20100123”:100

“20100124”:42

“20100123”:215

“20100124”:73

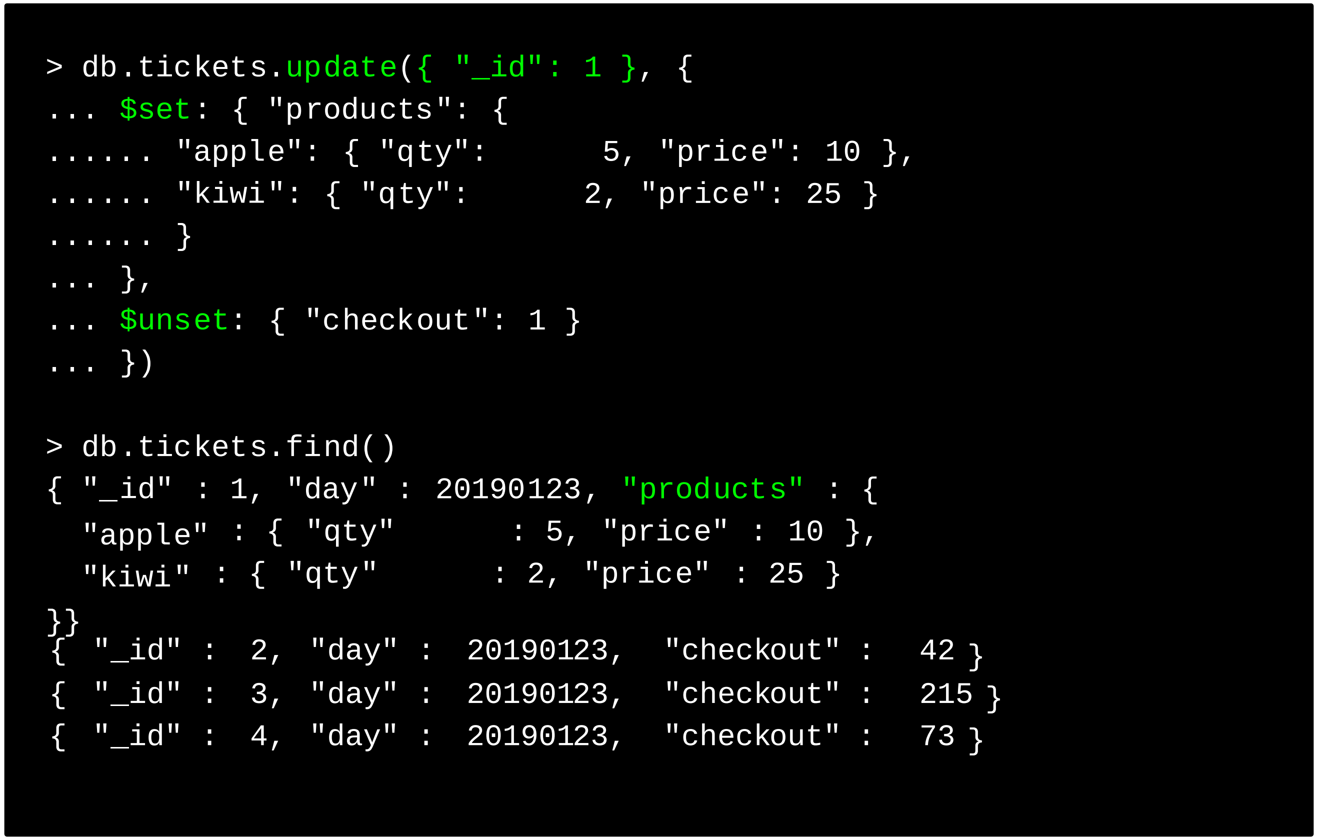
```
{
  "id": 1,
  "day": 20100123,
  "checkout": 100
}
```

```
{
  "id": 2,
  "day": 20100124,
  "checkout": 42
}
```

```
{
  "id": 3,
  "day": 20100123,
  "checkout": 215
}
```

```
{
  "id": 4,
  "day": 20100124,
  "checkout": 73
}
```

Update the tickets to remove checkout

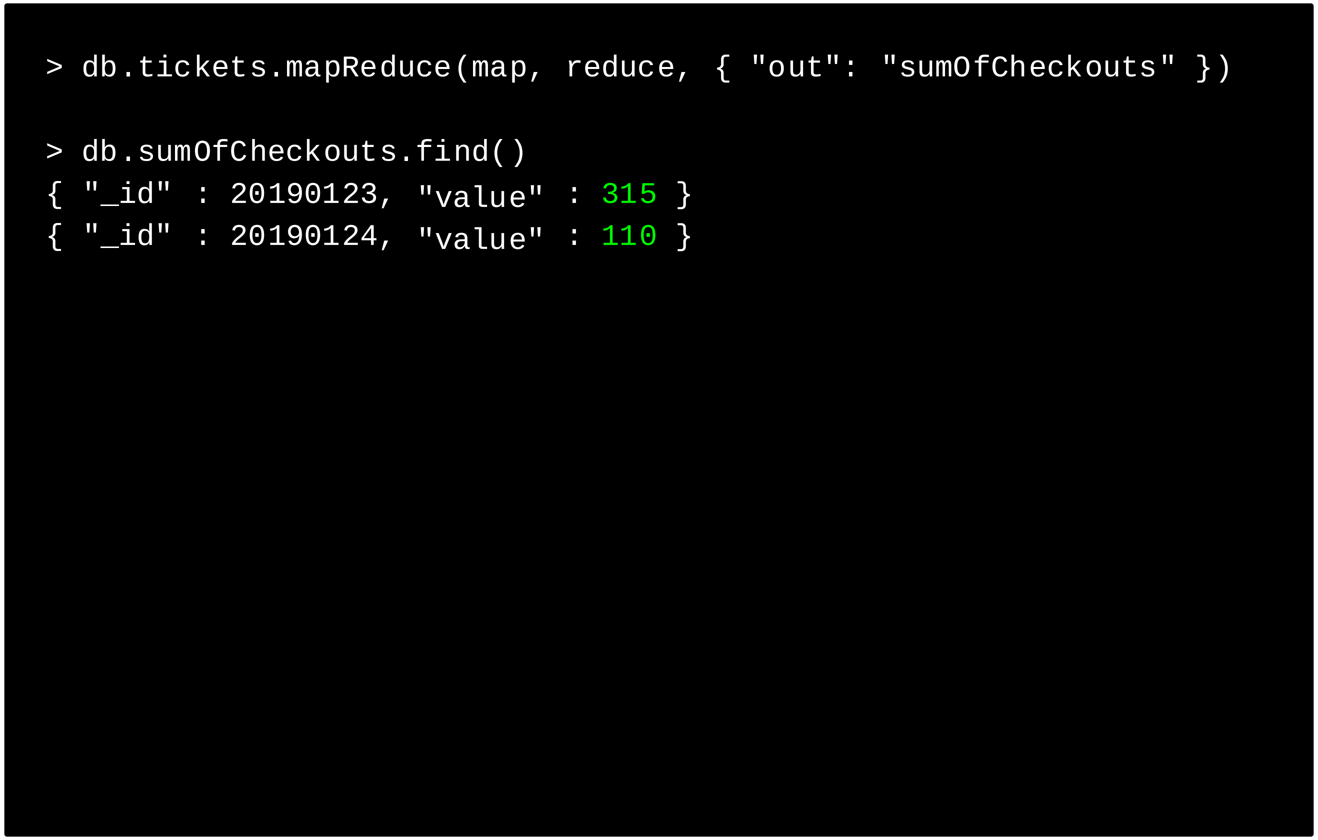


```
> db.tickets.update({ "_id": 1 }, {
... $set: { "products": {
...... "apple": { "qty":      5, "price": 10 },
...... "kiwi": { "qty":      2, "price": 25 }
...... }
... },
... $unset: { "checkout": 1 }
... })

> db.tickets.find()
{ "_id" : 1, "day" : 20190123, "products" : {
  "apple" : { "qty"      : 5, "price" : 10 },
  "kiwi" : { "qty"      : 2, "price" : 25 }
}}
{  "_id" :  2,  "day" :  20190123,  "checkout" :   42 }
{  "_id" :  3,  "day" :  20190123,  "checkout" :   215 }
{  "_id" :  4,  "day" :  20190123,  "checkout" :   73 }
```

Sum(Checkout) by day Calculate Checkout

```
> var map = function() {
... var checkout = 0
... for (var name in this.products) {
...... var product = this.products[name]
...... checkout += product.qty  * product.price
...... }
... emit(this.day, checkout)
}

> var reduce = function(key, values) {
... var sum = 0
... for (var index in values) sum += values[index]
... return sum
}
```

Sum(Checkout) by day Calculate Checkout



```
> db.tickets.mapReduce(map, reduce, { "out": "sumOfCheckouts" })

> db.sumOfCheckouts.find()
{ "_id" : 20190123, "value" : 315 }
{ "_id" : 20190124, "value" : 110 }
```

For today

Follow the map-reduce examples linked below and create the orders collection:
https://docs.mongodb.com/manual/tutorial/map-reduce-examples/

Write a map-reduce aggregation that returns the number of items bought (the number of elements in the items array) per costumer and submit it to ICON.