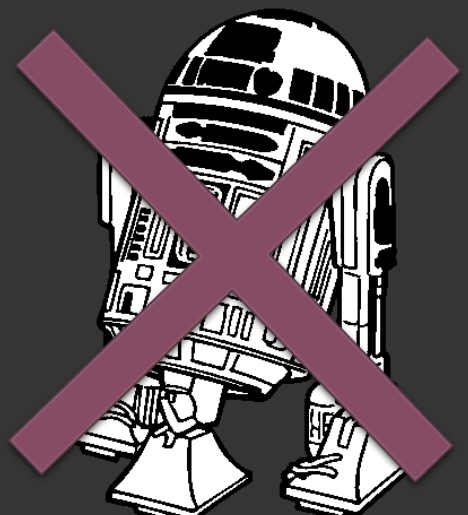
P2B2

Polyglot Persistence based Blockchain Analytics

https://github.com/p2b2



Simon Buchholz, Steffen Härtlein

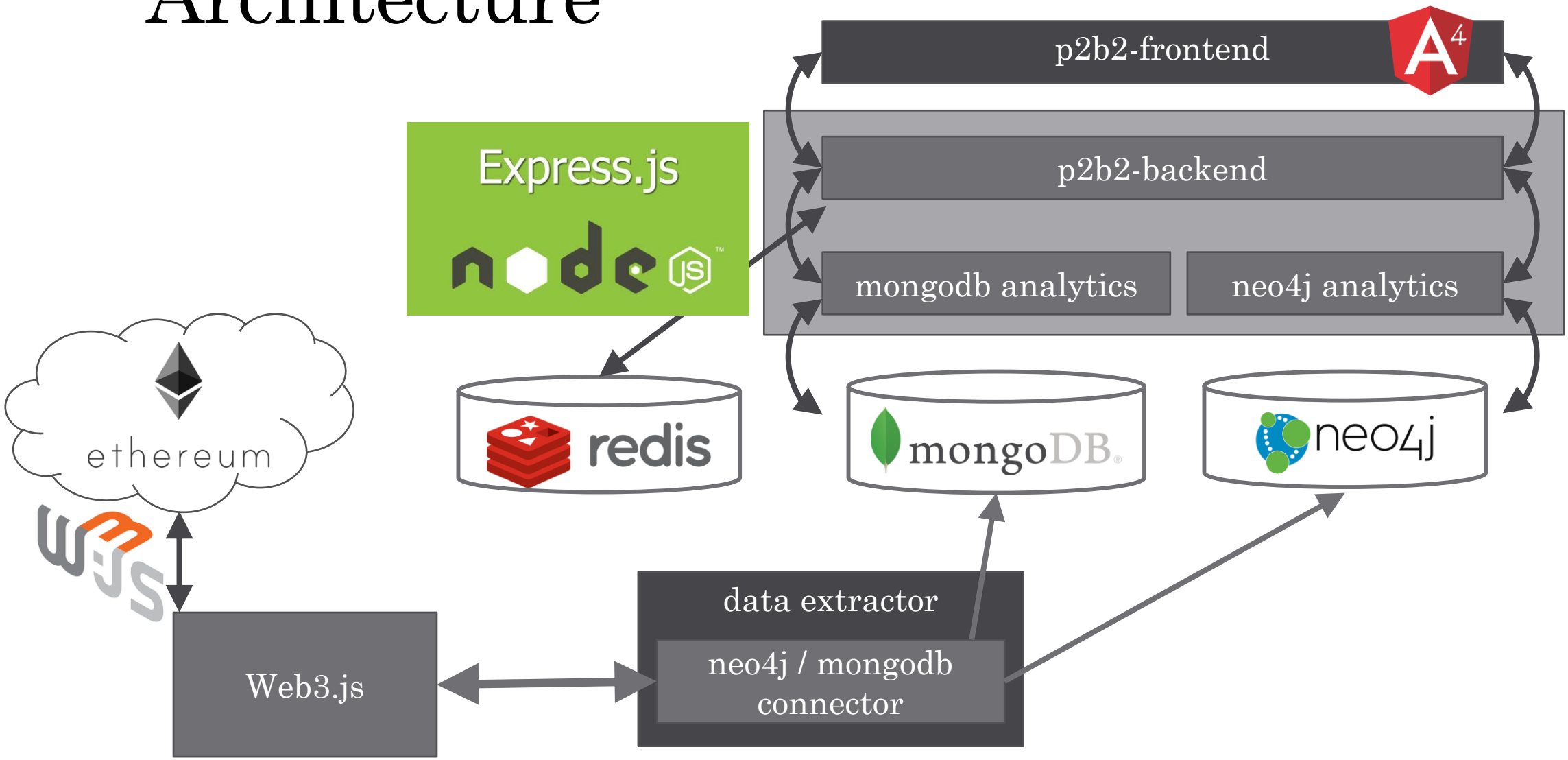
Motivation

ethereum

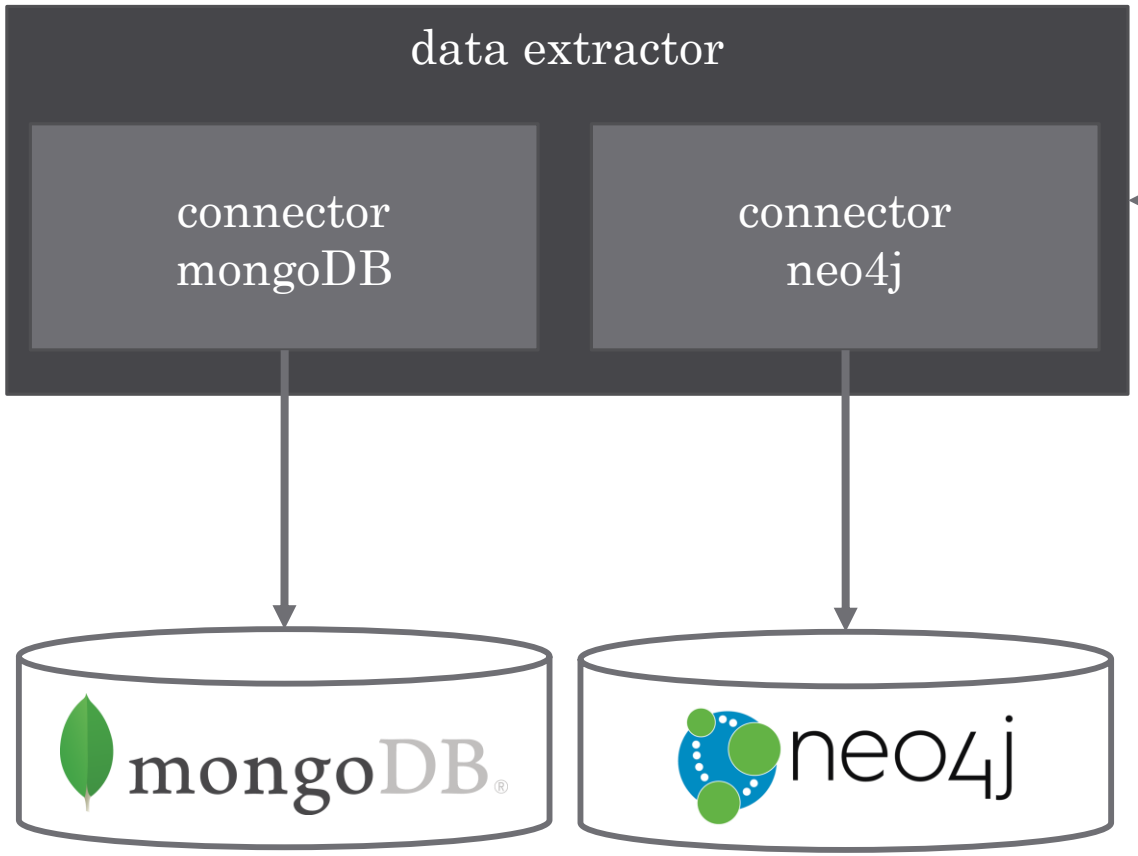
The Blockchain is basically a database itself, so why would we populate its' data to another database?

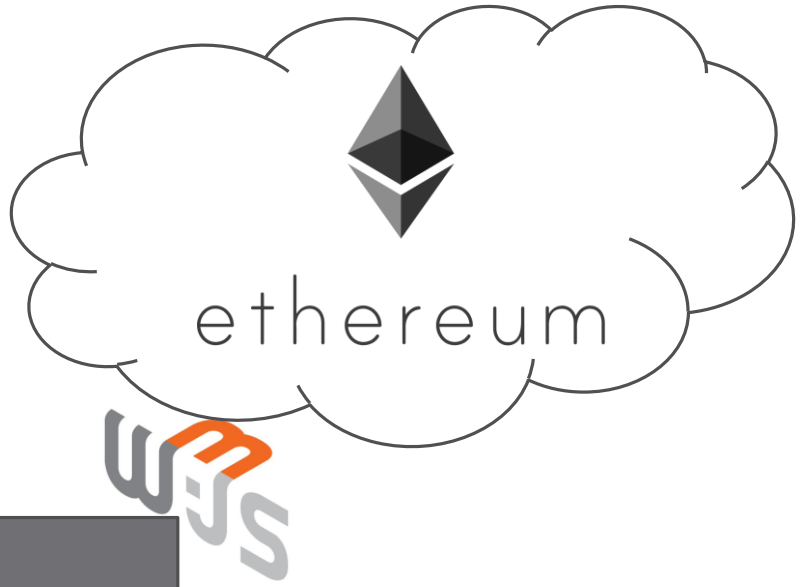
- The ethereum blockchain is stored in the `chaindata`
  - Currently, depending on the implementation, the Ethereum Blockchain is about 30GB
  - Optimized for validating transactions and maintaining a valid state
  - **NOT good for historical lookups or analytics**

Architecture



Data extraction






```
{
  "number": 3,
  "hash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bk
  "miner": "0x8888f1f195afa192cfee860698584c030f4c9c
  "difficulty": BigNumber,
  "size": 616,
  "extraData": "0x",
  "gasLimit": 3141592,
  "gasUsed": 21662,
  "timestamp": 1429287689,
  "transactions": [
    {    "hash": "0x9fc76417374aa880d4449a1f7f31ec597
         "nonce": 2,
         "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde
         "blockNumber": 3,
         "transactionIndex": 0,
         "from": "0xa94f5374fce5edbc8e2a8697c15331677
```

Data extraction - MongoDB

For each (cleaned) block, including transactions, insert object into the database


```
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001dfa"), "difficulty" : 17171480576, "extraData" : "0x476574682f76312e302e302f6c696e75782f676f312e342e32
", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0x05a56e2d52c817161883f50c441c3228cfe54d9f", "number" : 1, "size" : 537, "timestamp" : 14382699
88, "totalDifficulty" : 34351349760, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001dfb"), "difficulty" : 17163096064, "extraData" : "0x476574682f76312e302e302d30636463373634372f6c696e75
782f676f312e34", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0xdd2f1e6e498202e86d8f5442af596580a4f03c2c", "number" : 2, "size" : 544, "timesta
mp" : 1438270017, "totalDifficulty" : 51514445824, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001dfc"), "difficulty" : 17154715646, "extraData" : "0x476574682f76312e302e302d666337396433332642f6c696e75
782f676f312e34", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0x5088d623ba0fcf0131e0897a91734a4d83596aa0", "number" : 3, "size" : 1079, "timest
amp" : 1438270048, "totalDifficulty" : 68669161470, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001dfd"), "difficulty" : 17146339321, "extraData" : "0x59617465732052616e64616c6c202d2045746865724e696e6a
61", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0xc8ebccc5f5689fa8659d83713341e5ad19349448", "number" : 4, "size" : 1079, "timestamp" : 14382
70077, "totalDifficulty" : 85815500791, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001dfe"), "difficulty" : 17154711556, "extraData" : "0x476574682f76312e302e302f6c696e75782f676f312e342e32
", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0x05a56e2d52c817161883f50c441c3228cfe54d9f", "number" : 5, "size" : 537, "timestamp" : 14382700
83, "totalDifficulty" : 102970212347, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001dff"), "difficulty" : 17146335232, "extraData" : "0x476574682f76312e302e302f6c696e75782f676f312e342e32
", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0x0193d941b50d91be6567c7ee1c0fe7af498b4137", "number" : 6, "size" : 537, "timestamp" : 14382701
07, "totalDifficulty" : 120116547579, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001e00"), "difficulty" : 17154707466, "extraData" : "0x476574682f76312e302e302d30636463373634372f6c696e75
782f676f312e34", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0xdd2f1e6e498202e86d8f5442af596580a4f03c2c", "number" : 7, "size" : 1078, "timest
amp" : 1438270110, "totalDifficulty" : 137271255045, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001e01"), "difficulty" : 17163083788, "extraData" : "0x476574682f76312e302e302f6c696e75782f676f312e342e32
", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0x0193d941b50d91be6567c7ee1c0fe7af498b4137", "number" : 8, "size" : 537, "timestamp" : 14382701
12, "totalDifficulty" : 154434338833, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001e02"), "difficulty" : 17171464200, "extraData" : "0x476574682f76312e302e302d30636463373634372f6c696e75
782f676f312e34", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0xdd2f1e6e498202e86d8f5442af596580a4f03c2c", "number" : 9, "size" : 544, "timesta
mp" : 1438270115, "totalDifficulty" : 171605803033, "transactions" : [ ] }
{ "_id" : ObjectId("593fa7a4a9bd9b2d56001e03"), "difficulty" : 17163079696, "extraData" : "0x476574682f76312e302e302f6c696e75782f676f312e342e32
", "gasLimit" : 5000, "gasUsed" : 0, "miner" : "0x0193d941b50d91be6567c7ee1c0fe7af498b4137", "number" : 10, "size" : 537, "timestamp" : 1438270
128, "totalDifficulty" : 188768882729, "transactions" : [ ] }
```

Data extraction – Neo4j

Pseudocode for `insertBlock(block)`

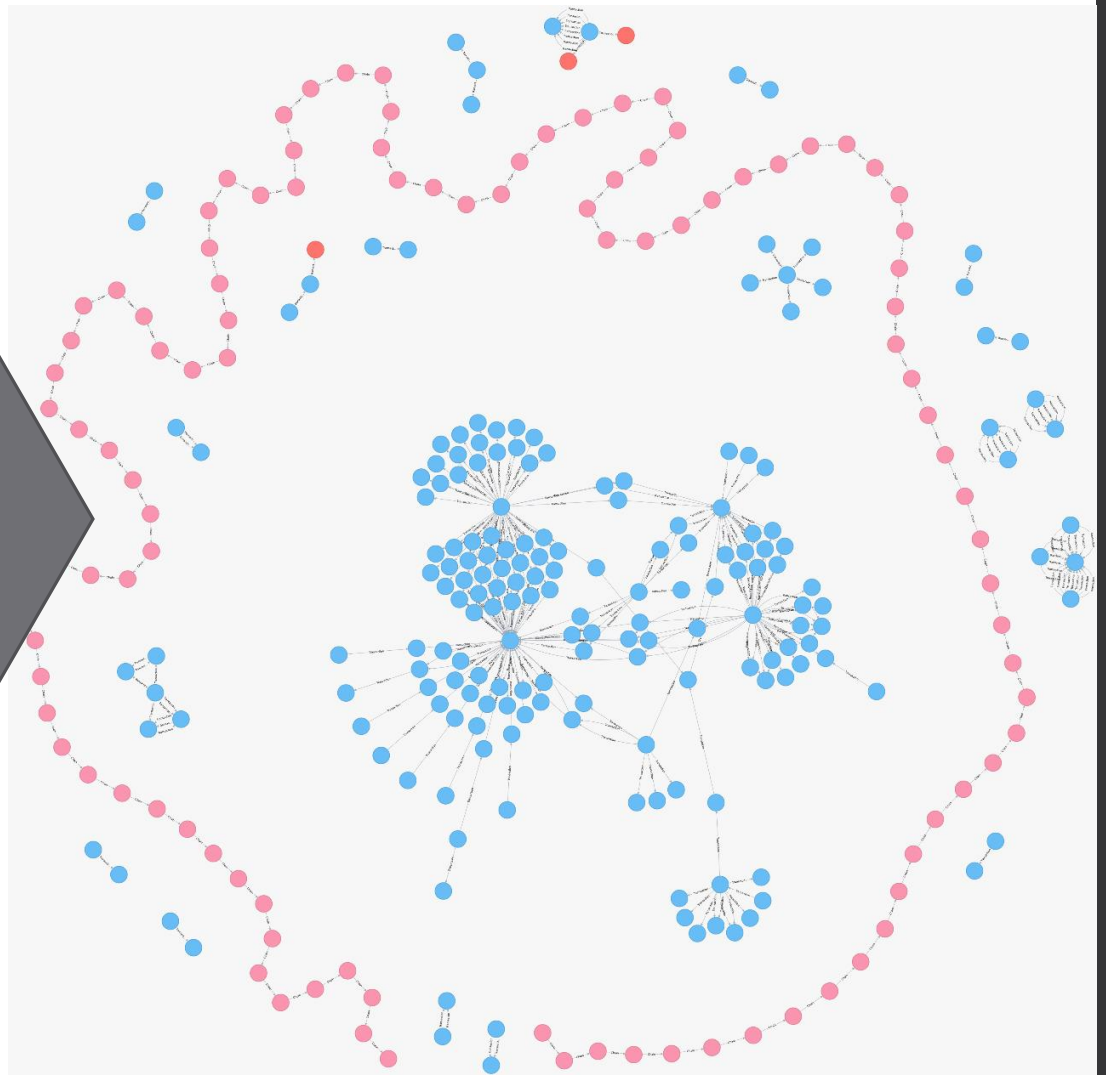
1. If: `lastInsertedBlock == null`
   1. Create Scheme: Unique constraint on `account.address` and `block.blockNumber`
2. Save `block` as node in graph
   1. Chain the `block` to its predecessor
   2. Check if `block.miner` is inserted in the graph as an account node already
      1. If yes → Insert edge `mined`
      2. Else → Insert `block.miner` as a node in graph + Insert edge `mined`
3. For each `transaction in block.transactions`
   1. Check if the `transaction` is a contract creation
      1. If yes → Create a new contract node in the graph
   2. Check if `transaction.from` and `transaction.to` are inserted in the graph as account nodes already
      1. If yes → Insert `transaction` as edge
      2. Else → Create the missing account node(s)
   3. Insert `transaction` itself as an edge into the graph

Execute all in a Neo4j transaction
If error → `rollback`, else→ `commit`

Data extraction – Neo4j

Block received from Web3.js

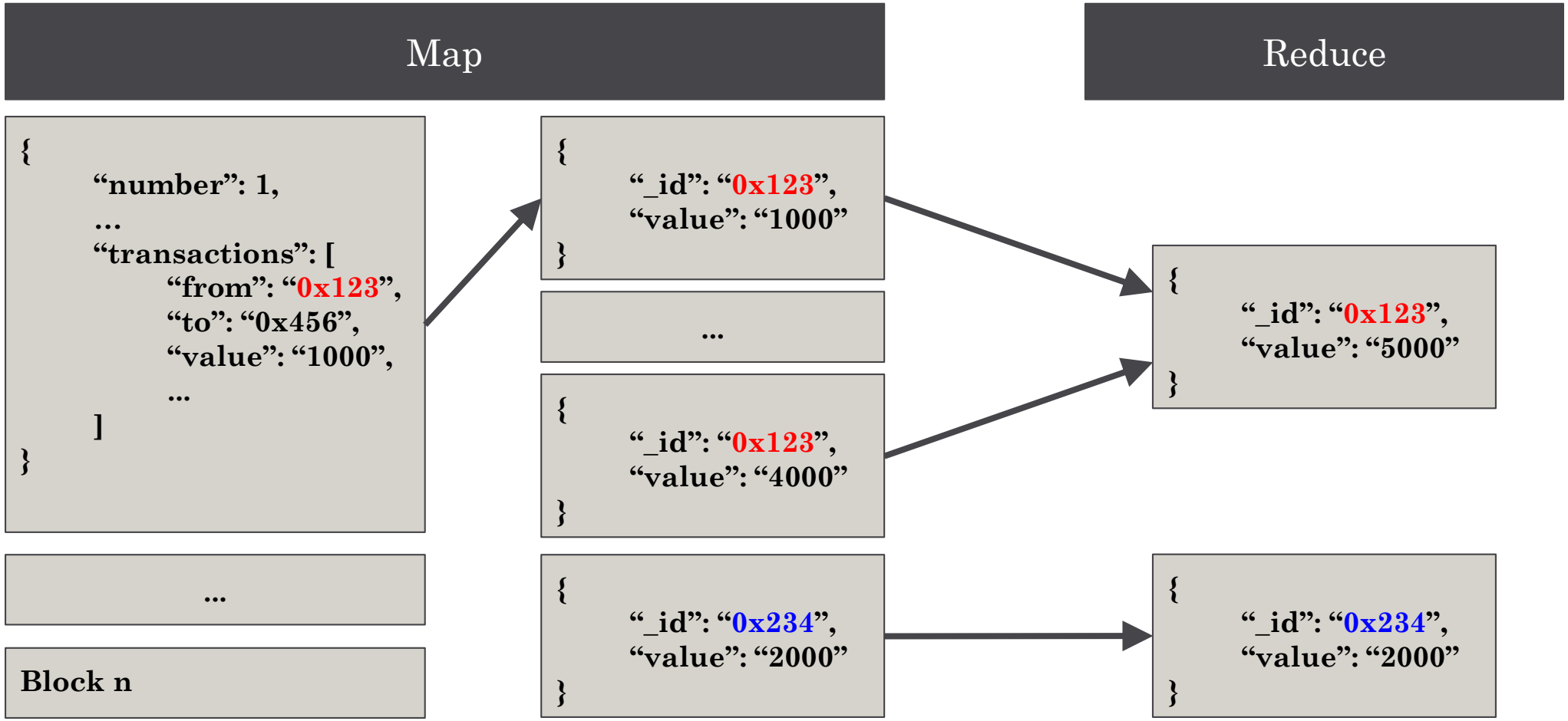

```
"number": 3,
"hash": "0xef95f2f1ed3ca60b048b4bf67cde2195961e0bba6f70bcbea9a1
"miner": "0x8888f1f195afa192cfee860698584c030f4c9db1",
"difficulty": BigNumber,
"size": 616,
"extraData": "0x",
"gasLimit": 3141592,
"gasUsed": 21662,
"timestamp": 1429287689,
"transactions": [
  {   "hash": "0x9fc76417374aa880d4449a1f7f31ec597f00b1f6
      "nonce": 2,
      "blockHash": "0xef95f2f1ed3ca60b048b4bf67cde219596
      "blockNumber": 3,
      "transactionIndex": 0,
      "from": "0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b
      "to": "0x6295ee1b4f6dd65047762f924ecd367c17eabf8f",
      "value": BigNumber,
      "gas": 314159,
      "gasPrice": BigNumber,
      "input": "0x57cb2fc4"
  }, {...}, ...
],
```

Analytics - Scenarios

- MapReduce Analytics
  - **Which accounts have the highest revenue?**
  - **How high is the revenue for a given account?**
  - How high are transaction fees in average?
  - …
- GraphDB Analytics
  - **Given an account address, who sent transactions to that account?**
  - Are there different clusters, where most of transactions are done?
  - **What are the most important accounts or contracts in the network?**
  - …

Analytics – MapReduce Jobs



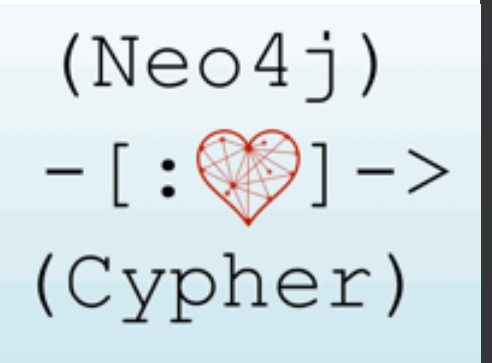
Analytics – Graph theory

- Different metrics from Graph theory give interesting insights
  - **Degree Centrality**
  - **Betweenness Centrality**
  - PageRank
  - etc.

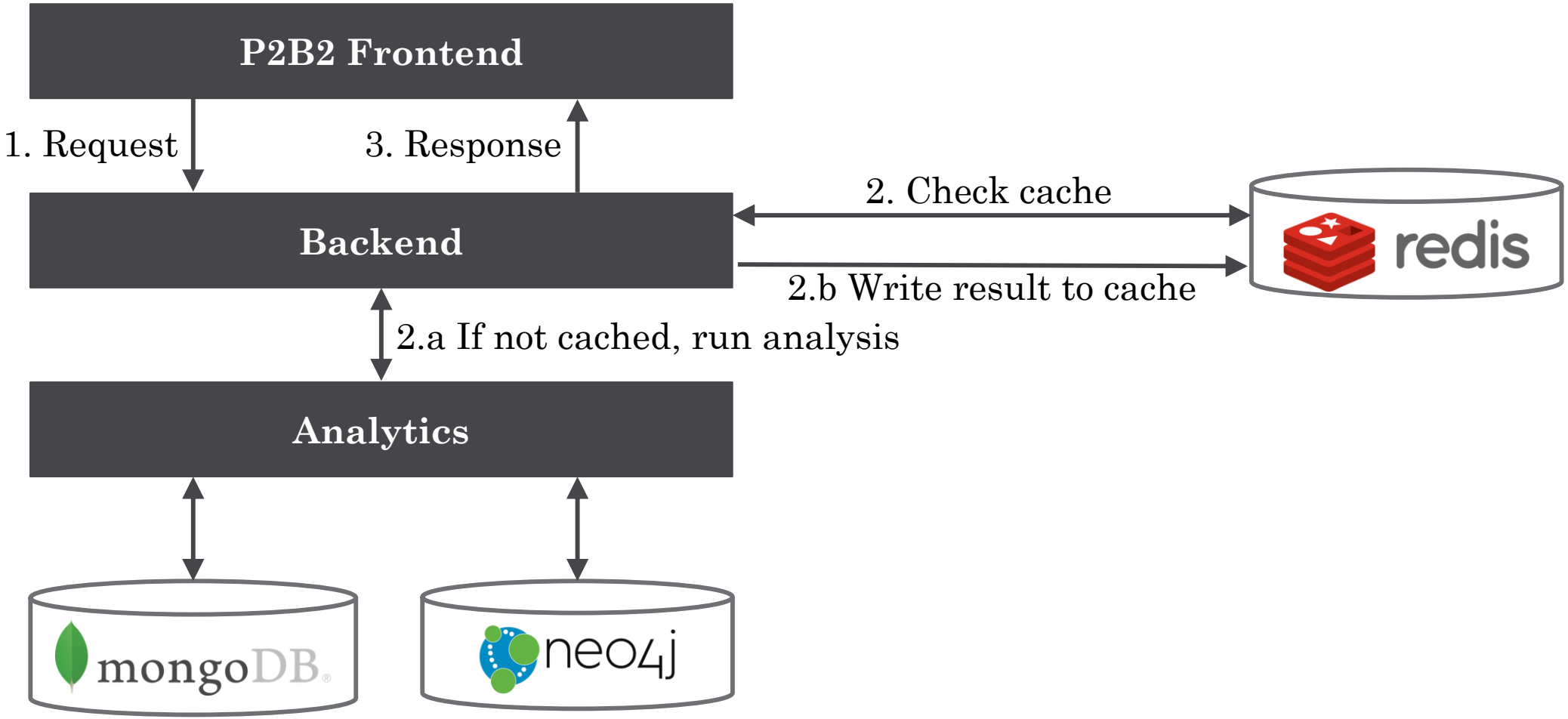
- Example:

Degree Centrality:

```
MATCH (n:Account)-[r:Transaction]-(m:Account)
RETURN n.address, count(r) AS DegreeScore
ORDER BY DegreeScore DESC
LIMIT 10;
```



Backend and Caching



Front end

P2B2 - Ethereum Analyzer

Account info

Enter an Ethereum address to display details for it *

0xc8ebccc5f5689fa8659d83713341e5ad19349448

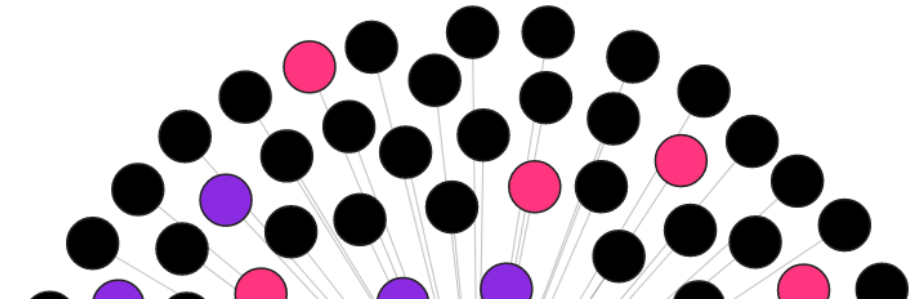
Get Details

Total revenue: 12362.258602078503 ether

Transaction Record

Block Number	From	To	Value	Gas	Gas price	Input
1067639	0xc8ebccc5f5689fa8659d83713341e5ad19349448	0xb26842086a407eb4fef0465f502285f3d25d168b	3822457805882795520	21000	50000000000	0x
1045768	0xc8ebccc5f5689fa8659d83713341e5ad19349448	0x00fc572550f3bdfe84f72e3eaa99d02a43f69733	9223372036854775807	21000	50000000000	0x
1030933	0xc8ebccc5f5689fa8659d83713341e5ad19349448	0xceafeaa5e4f736d64119073fc34c7f30930a6248	10000000000000000000	90000	10000000000000	0x

Account Graph



Questions?

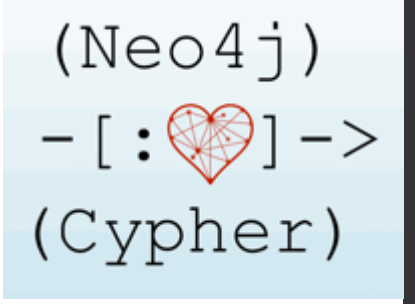
Backup / Notes

Analytics improvements – Graph

- **Cypher** is a **declarative** graph query language that allows querying and updating the graph store
  - In **declarative** graph query, you are specifying **what you want**
- **Traversal** language is an **imperative** way of accessing a graph
  - In imperative access, you are instead telling the database exactly **how to get the graph**
  - E.g. I want to do a depth first search, prune these branches, stop when I hit certain nodes, etc.

➔ In general: Cypher is well optimized and faster for most queries

➔ But for some analytical processing (OLAP), traversal access could bring performance improvements (not evaluated because out of time scope)



Analytics – Graph theory

- Different metrics from Graph theory give interesting insights
  - **Degree Centrality**
  - **Betweenness Centrality**
  - PageRank
  - etc.
- Examples:

Degree Centrality:

```
MATCH (n:Account)-[r:Transaction]-(m:Account)
RETURN n.address, count(r) AS DegreeScore
ORDER BY DegreeScore DESC
LIMIT 10;
```

Betweenness Centrality:

```
MATCH p=allShortestPaths((source:Account)-
[:Transaction*]-(target:Account))
WHERE id(source) < id(target) and length(p) > 1
UNWIND nodes(p)[1..-1] AS n
RETURN n.address, count(*) as betweenness
ORDER BY betweenness DESC
LIMIT 10;
```