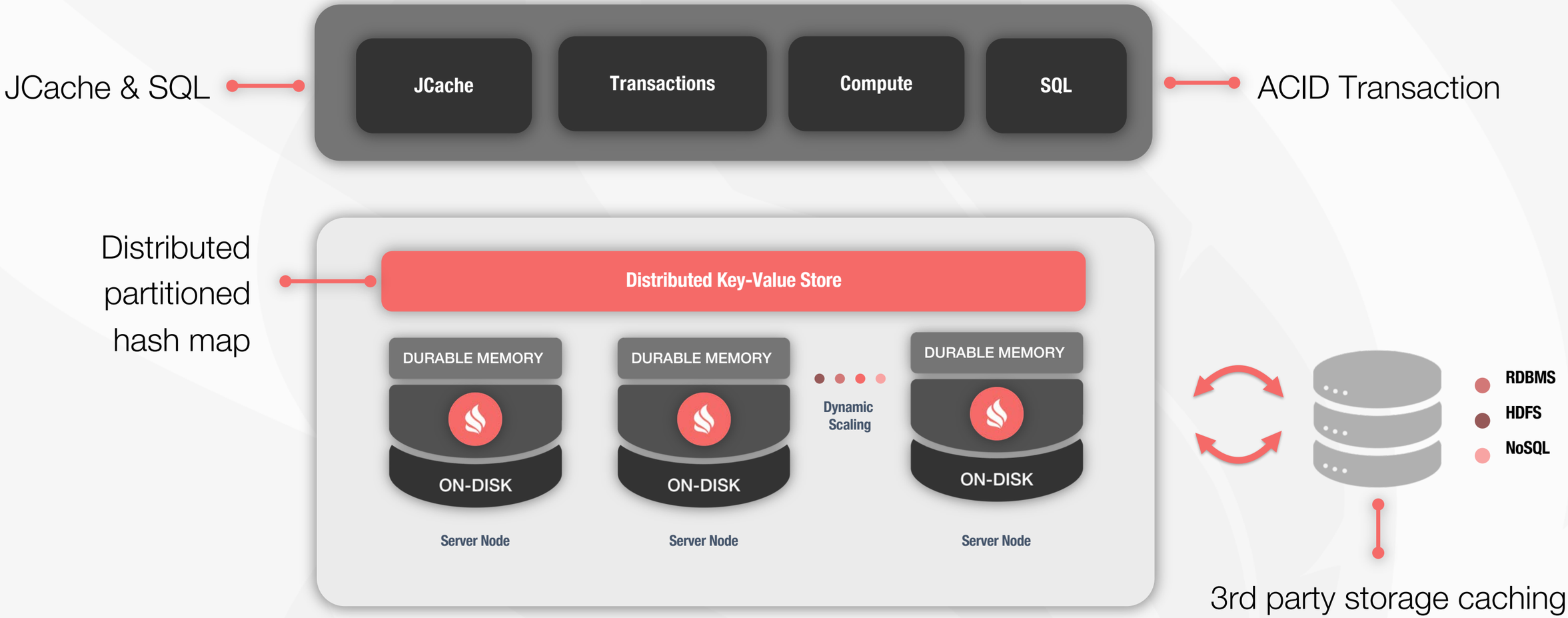
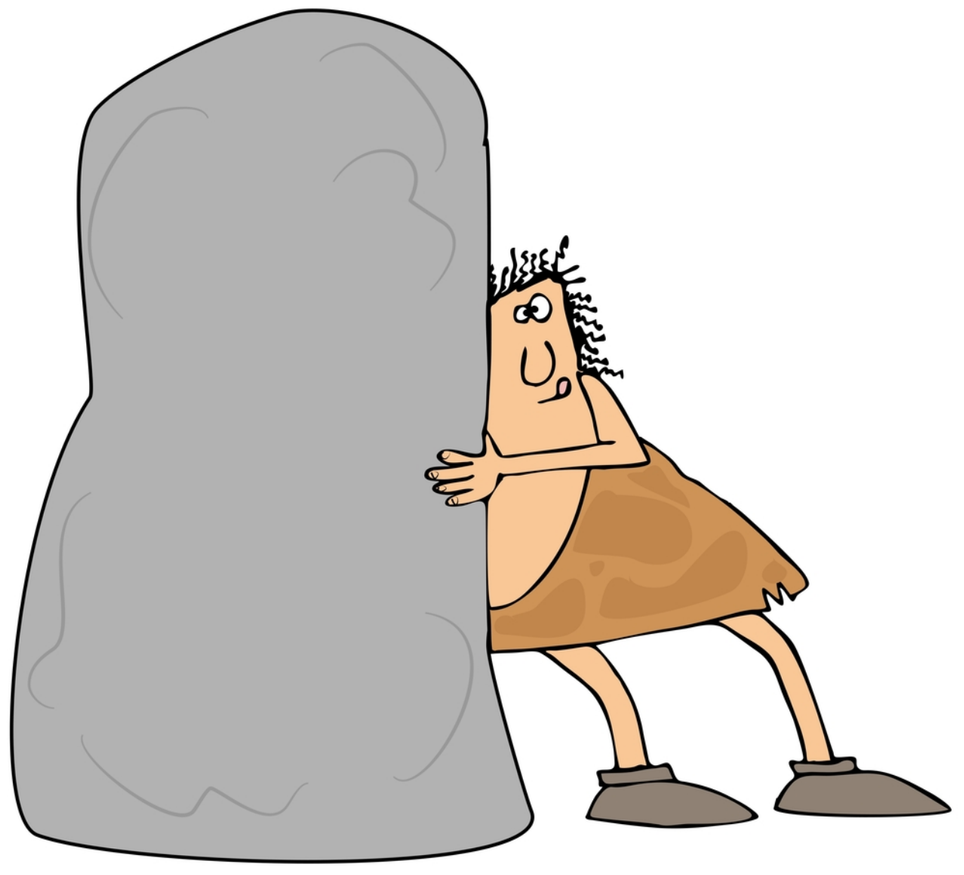
Want extreme performance at scale? Do distributed the RIGHT way!

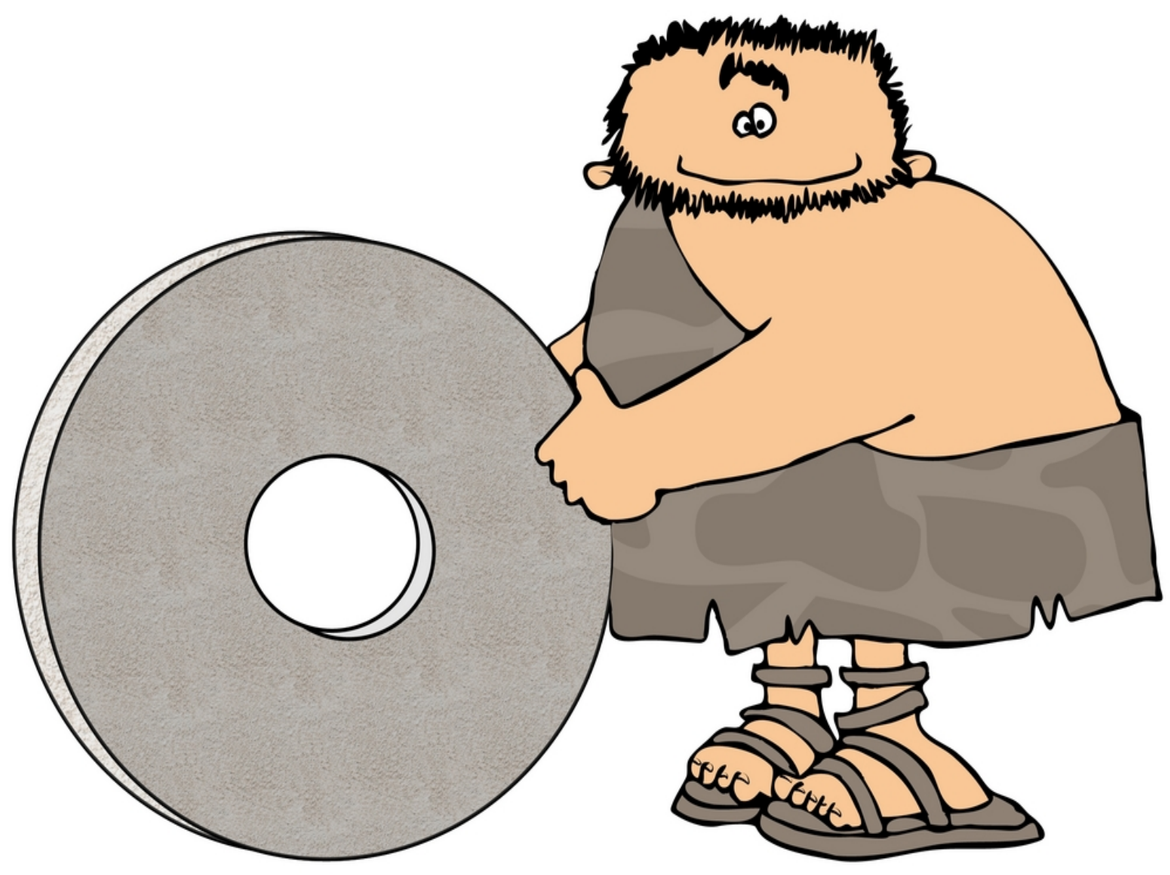
Valentin Kulichenko
Apache Ignite Committer
GridGain Solutions Architect

Distributed Storage



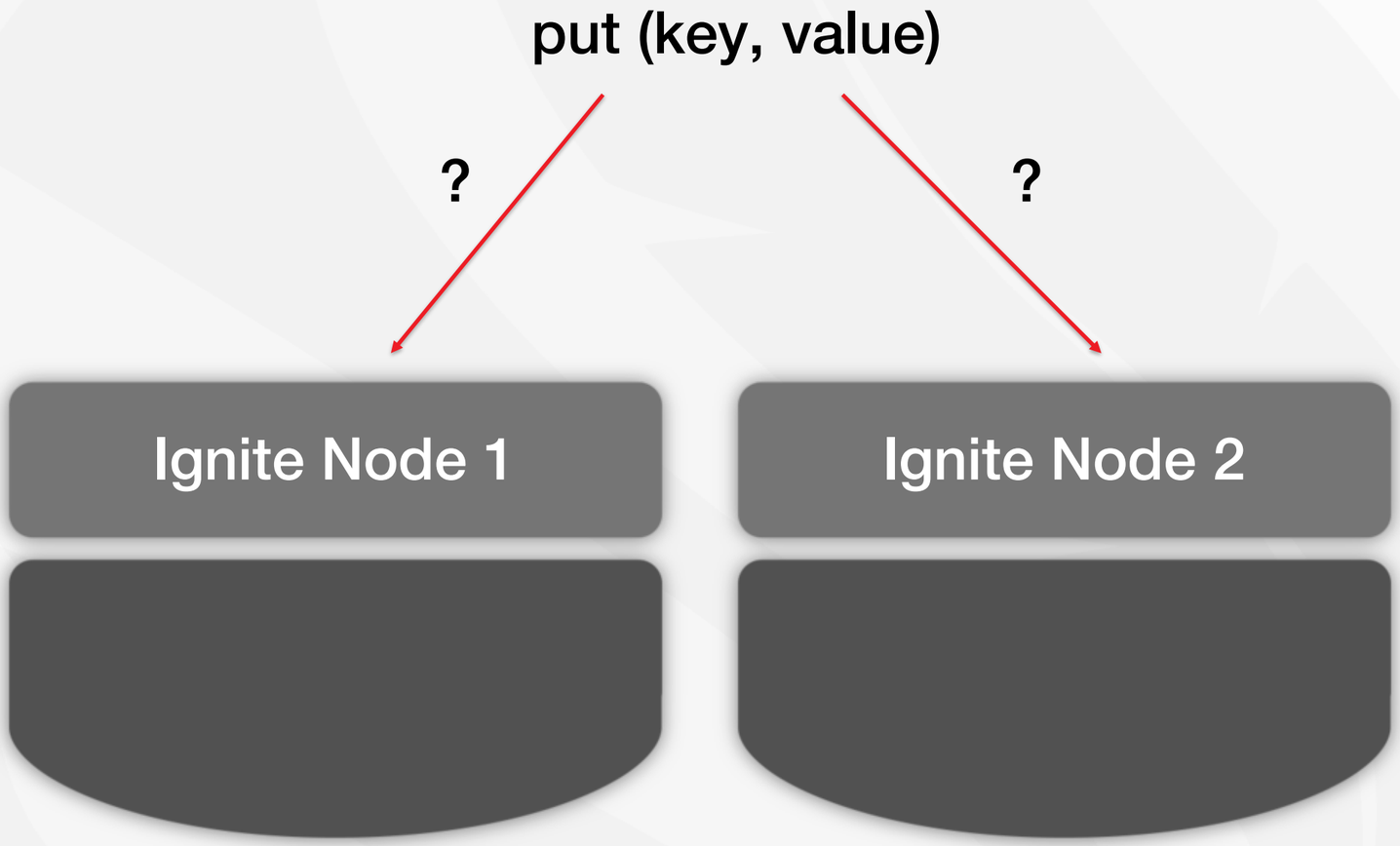
Where Is The Challenge?



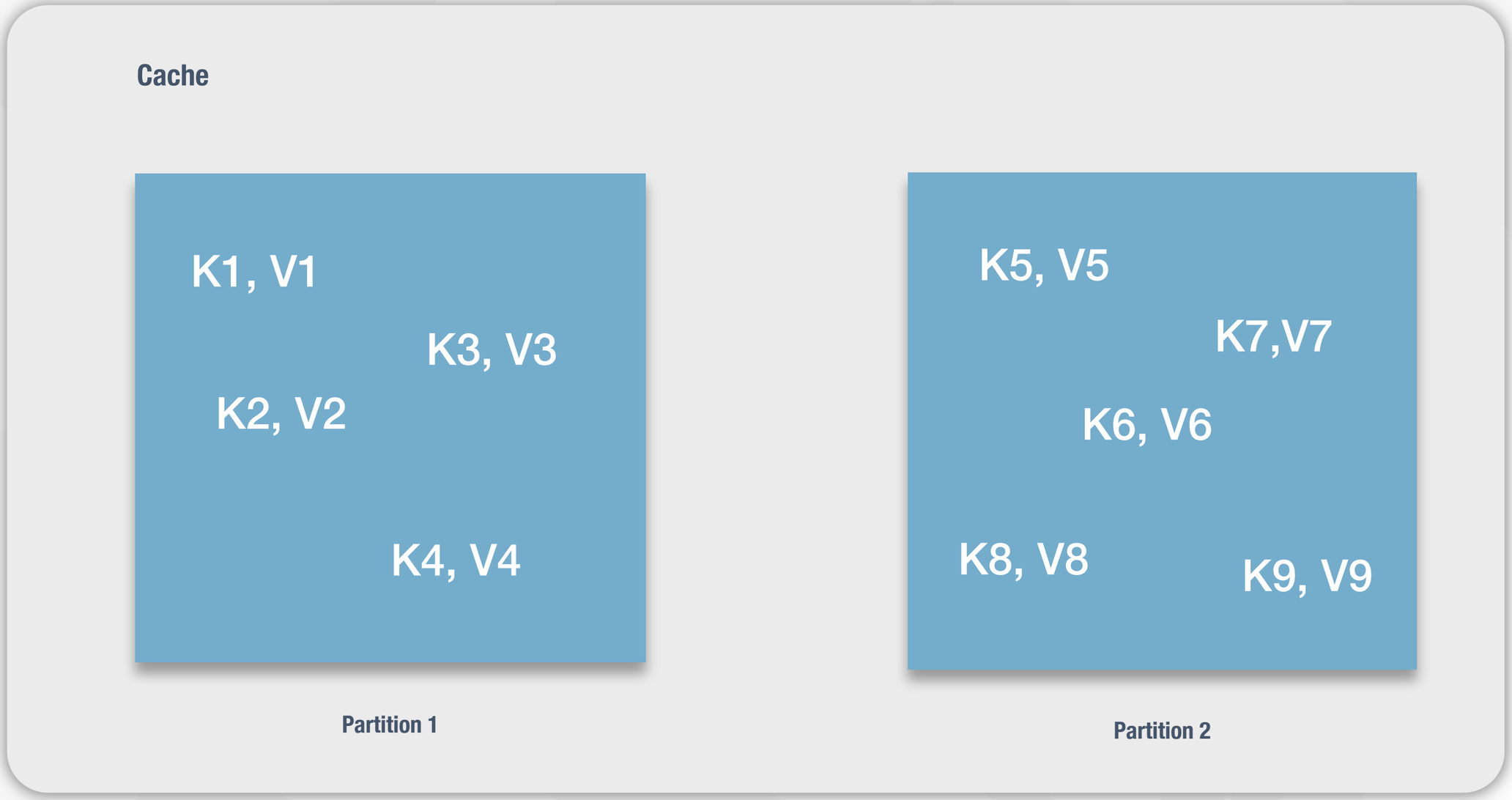


Data Affinity

Where Entry Goes?



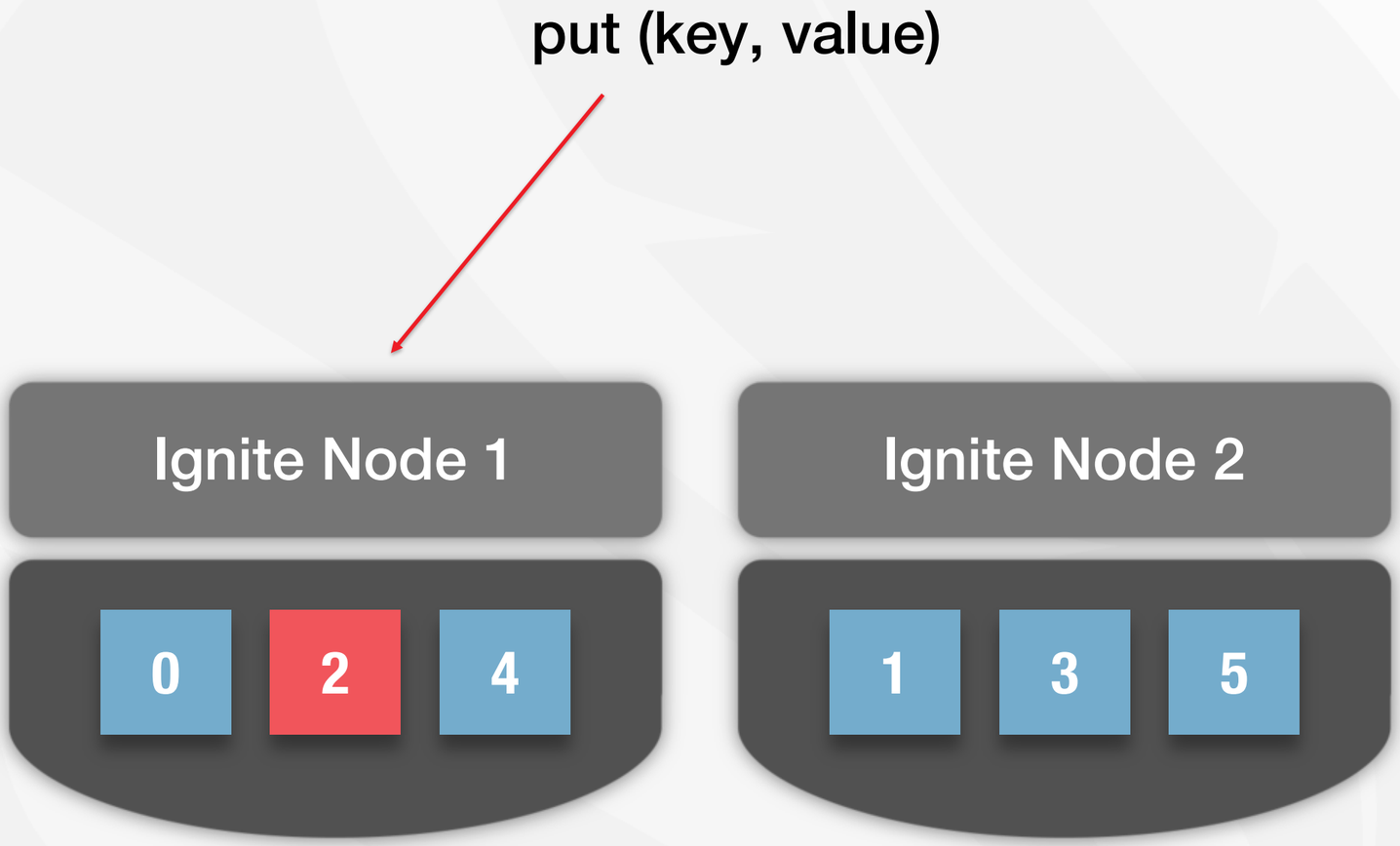
Caches and Partitions



Partitions Distribution

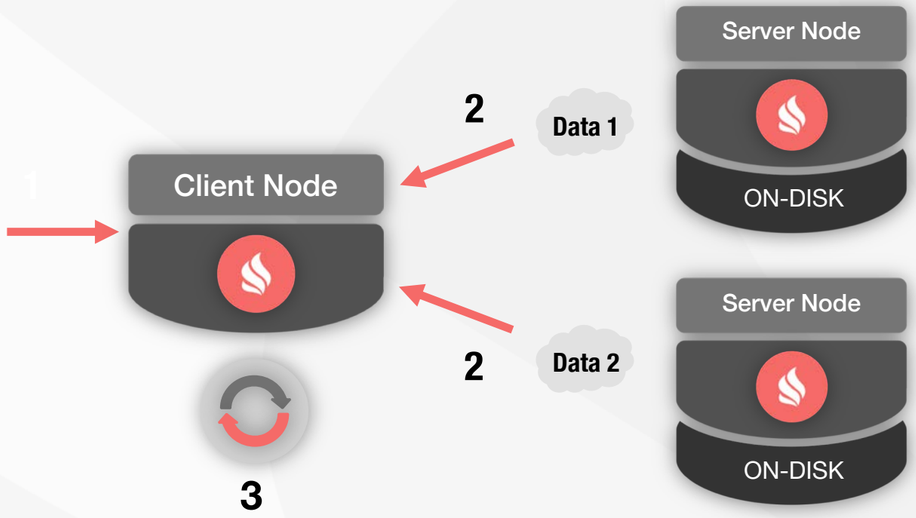
Affinity Function

Where Entry Goes?



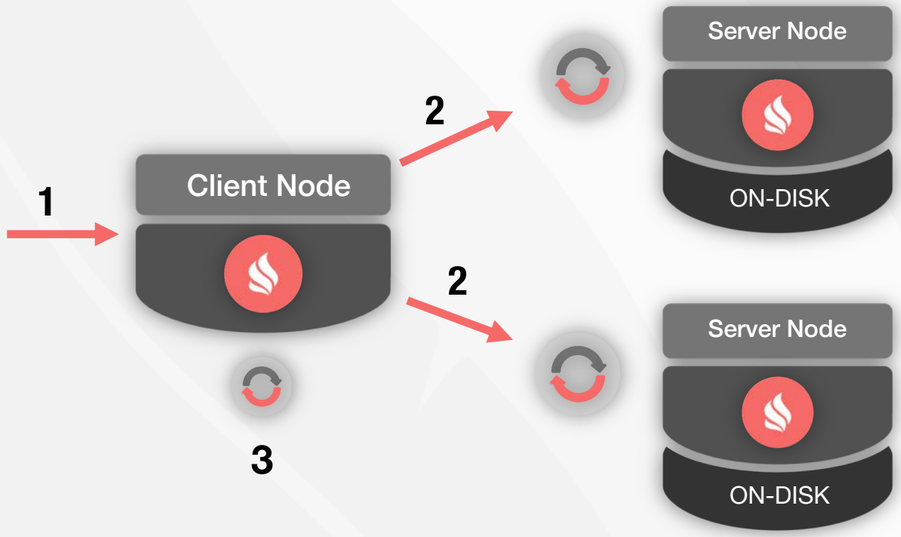
Co-located Processing

Client-Server Processing



1. Initial Request
2. Fetch data from remote nodes
3. Process entire data-set

Co-located Processing



1. Initial Request
2. Co-located processing with data
3. Reduce multiple results in one

Use Case: Account Balance Update

```
class Account {
    String firstName;
    String lastName;
    String address;

    ...

    double balance;
}
```

```
Account account = cache.get(123);

account.balance -= 100;

cache.put(123, account);
```

Use Case: Account Balance Update

```
cache.invoke(123, new EntryProcessor<Integer, Account, Object>() {
    @Override public Object process(MutableEntry<Integer, Account> entry,
                                    Object... args) {
        Account account = entry.getValue();

        account.balance -= 100;

        entry.setValue(account);

        return null;
    }
});
```

Co-located Data

Use Case: Payment Transaction Authorization

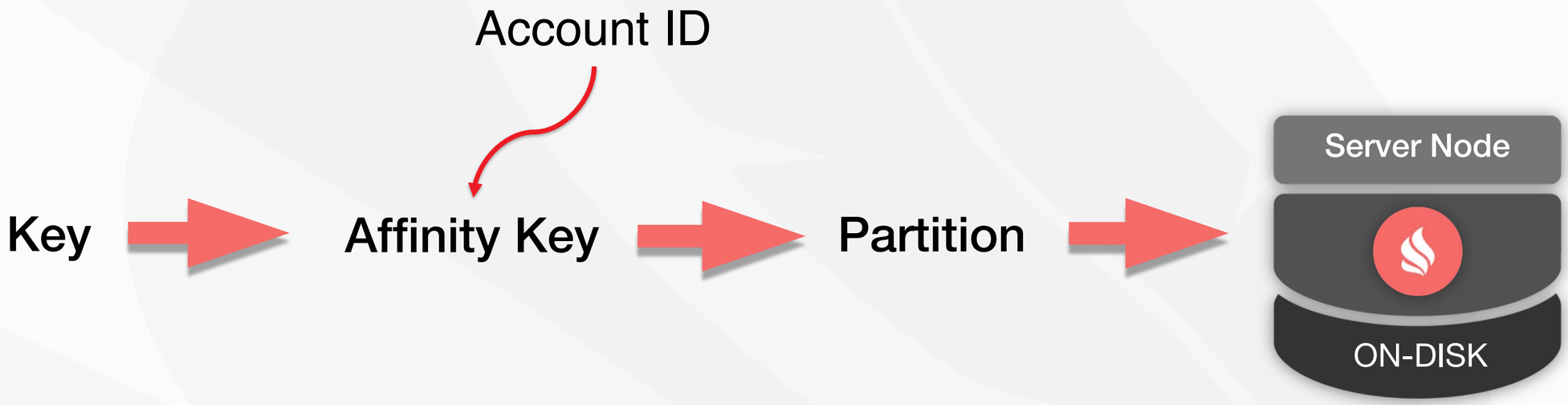
```
class Transaction {
    int accountId;
    String storeName;
    double amount;
}
```

For each new transaction:

- Find all transactions for the account ID
- Go through the list, calculate authorization variables
- If transaction is authorized, add it to the list

Affinity Key

Affinity Key



Affinity Key

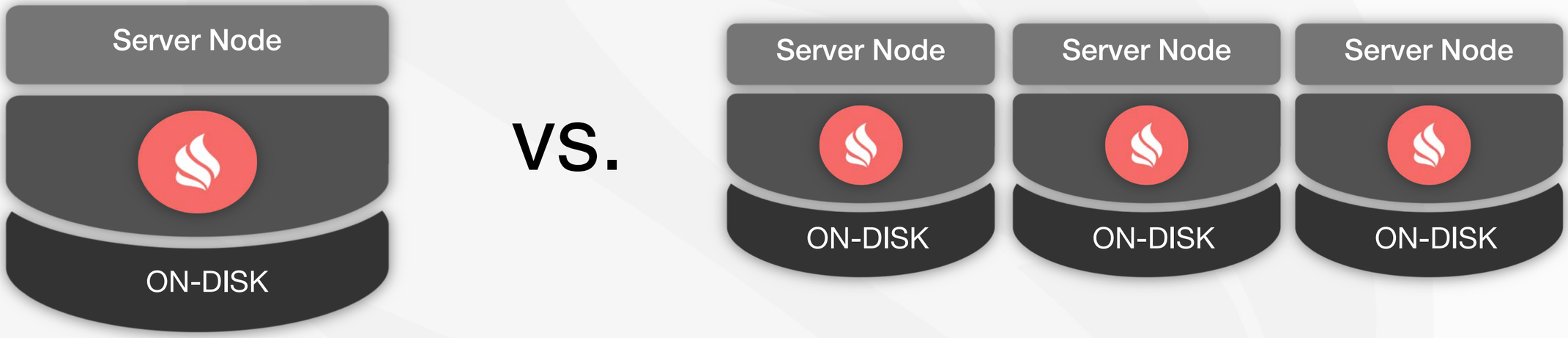
```
class TransactionKey {
    int transactionId;

    @AffinityKeyMapped
    int accountId;
}
```

```
ignite.compute().affinityRun(
    "transactions", // Cache name.
    123,            // Account ID.
    () -> { ... }   // Computation.
);
```

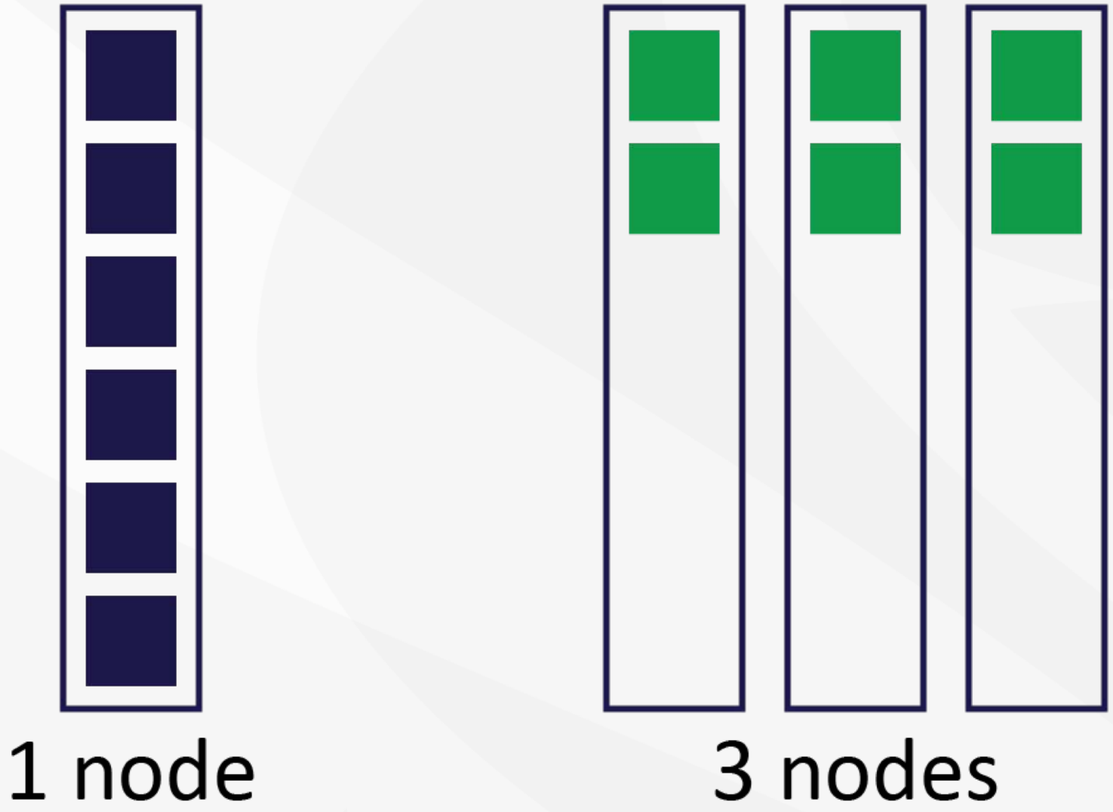
Co-location and SQL: Indexing

Let’s Run a SQL Query!

```
SELECT AVG(amount) FROM Transaction WHERE accountId = ?
```



Executing SQL: Full Scan



- **1/3x** latency
- **3x** capacity

But What If We Use Index?

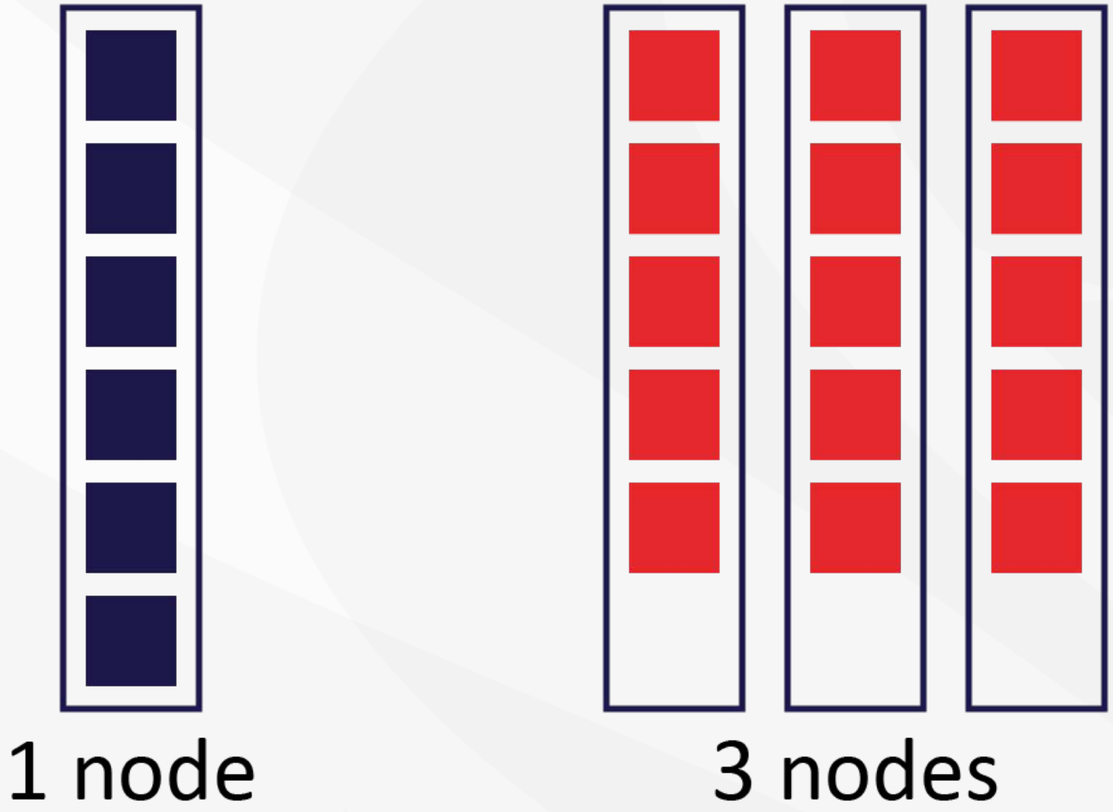
Indexed Search Complexity

```
log 1_000_000 ≈ 20
```

vs.

```
log   333_333 ≈ 18
log   333_333 ≈ 18
log   333_333 ≈ 18
```

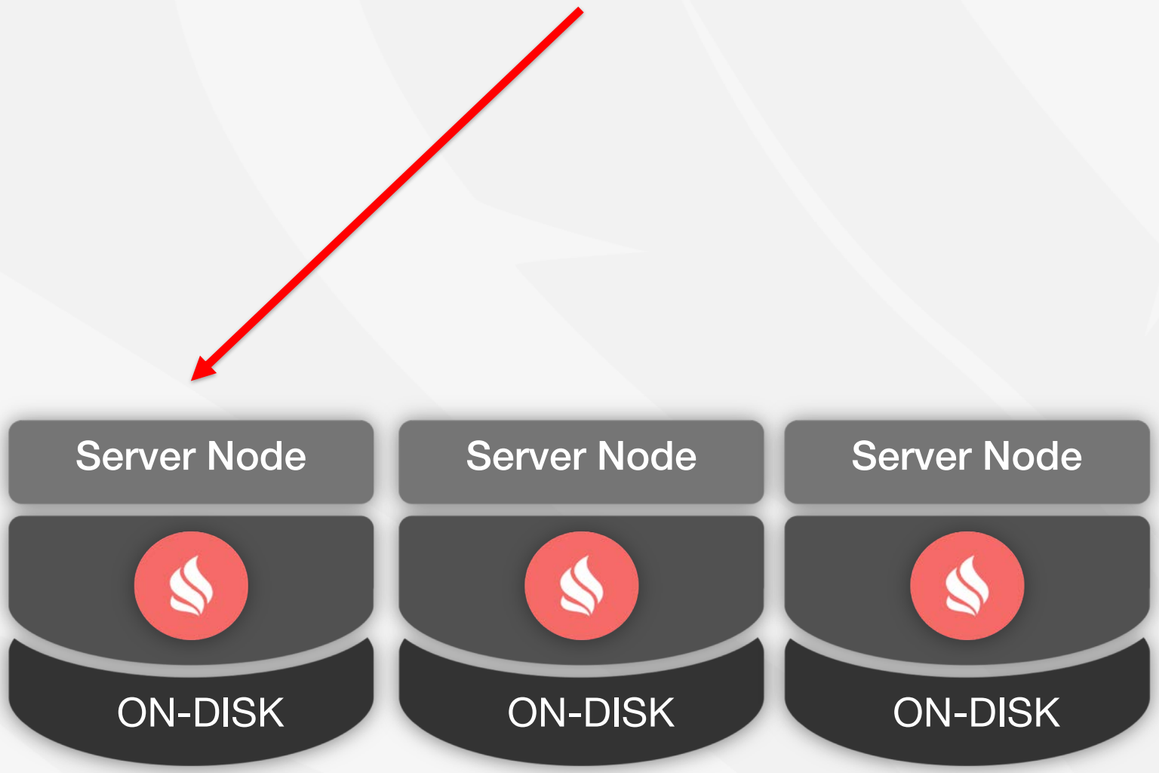
Executing SQL: Indexed Search



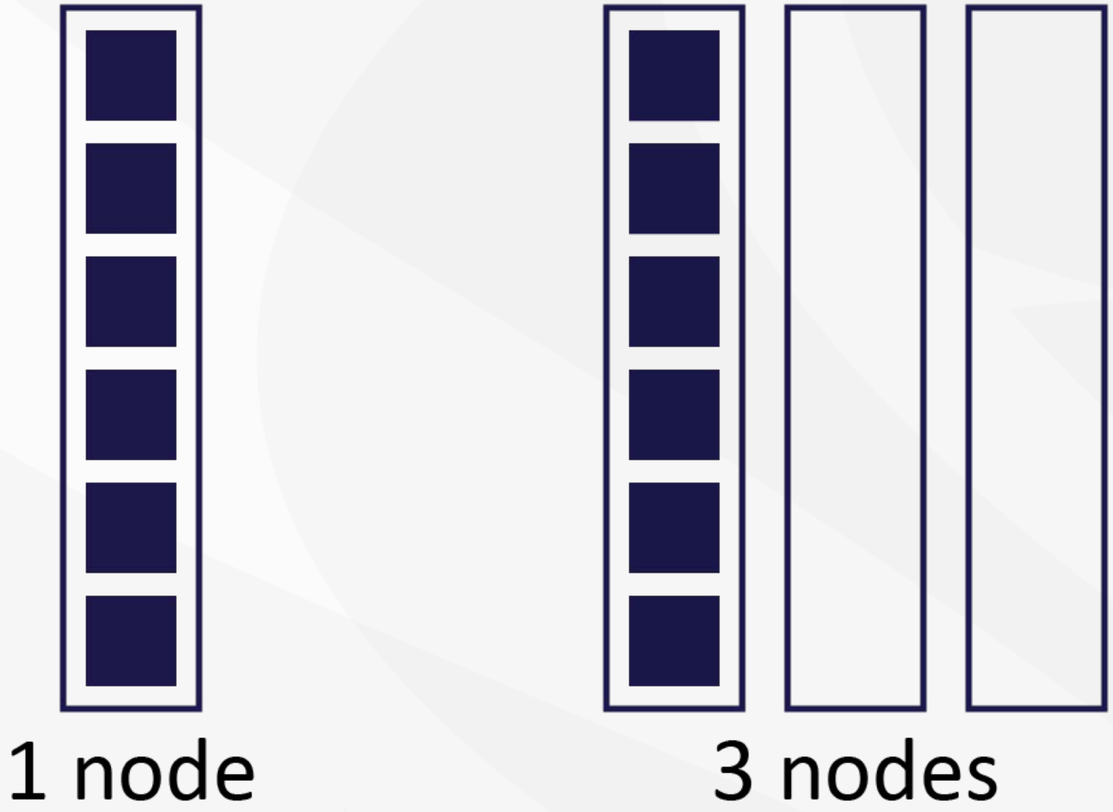
- **~same** latency
- **~same** capacity

Let’s Co-locate

```
SELECT AVG(amount) FROM Transaction WHERE accountId = ?
```



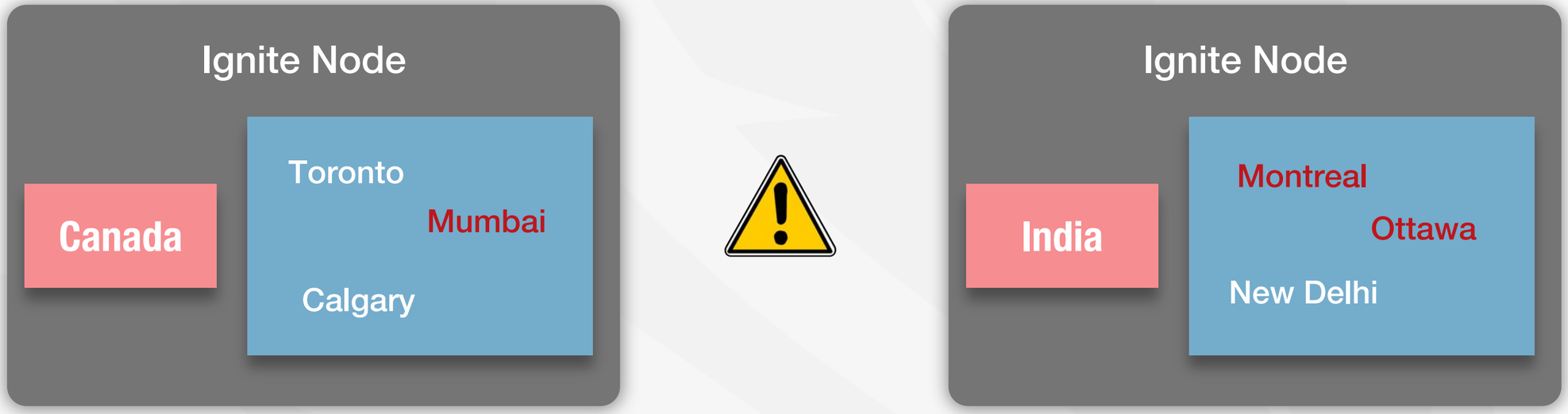
Executing SQL: Indexed Search With Co-location



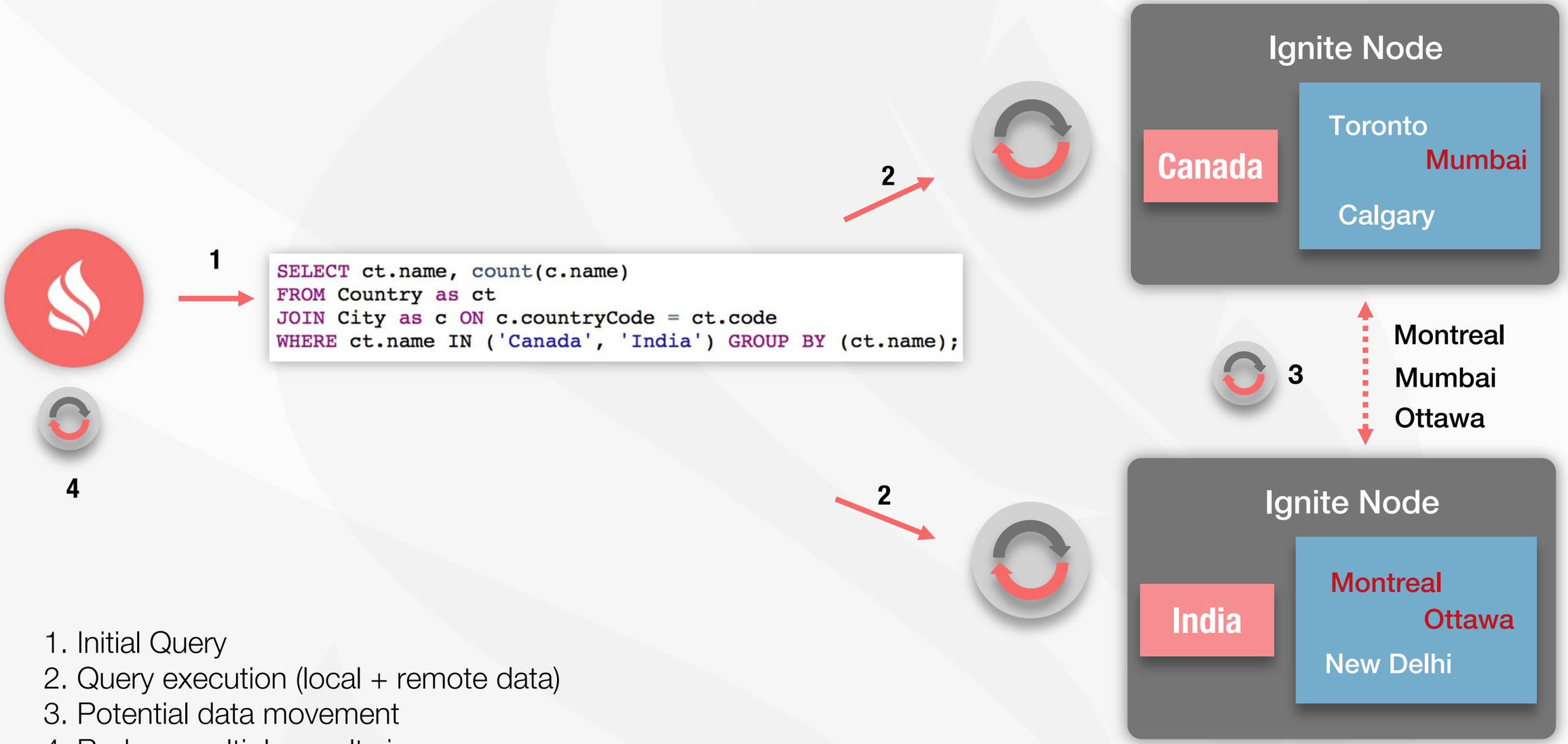
- **same** latency
- **3x** capacity

Co-location and SQL: Joins

Random Distribution

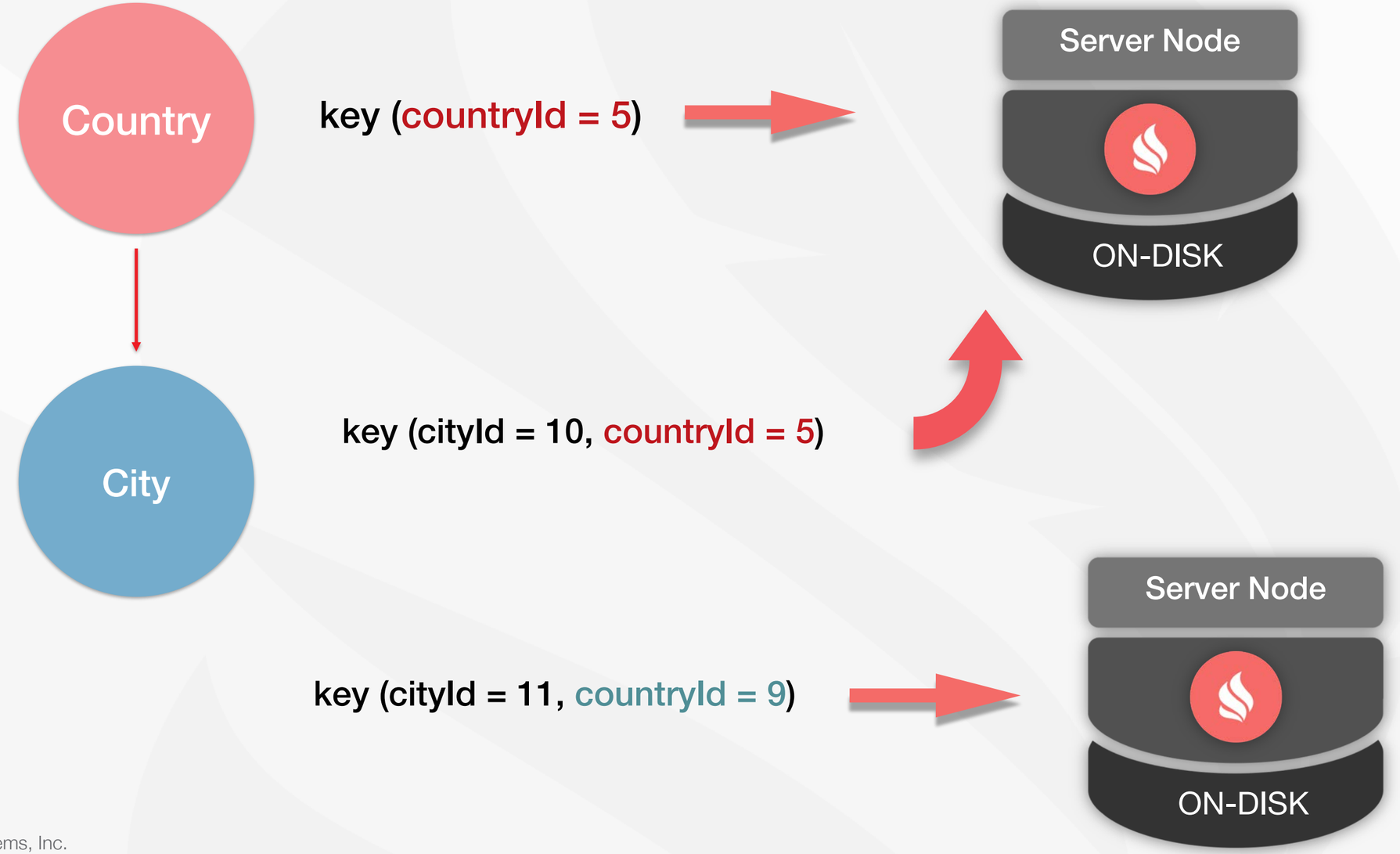


Non-Collocated Joins

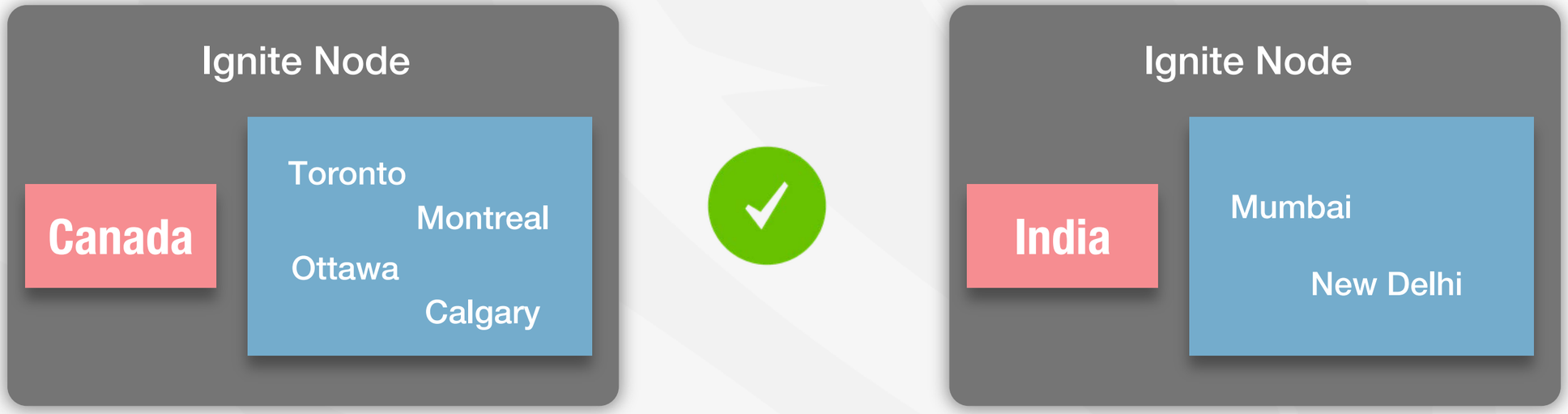


1. Initial Query
2. Query execution (local + remote data)
3. Potential data movement
4. Reduce multiple results in one

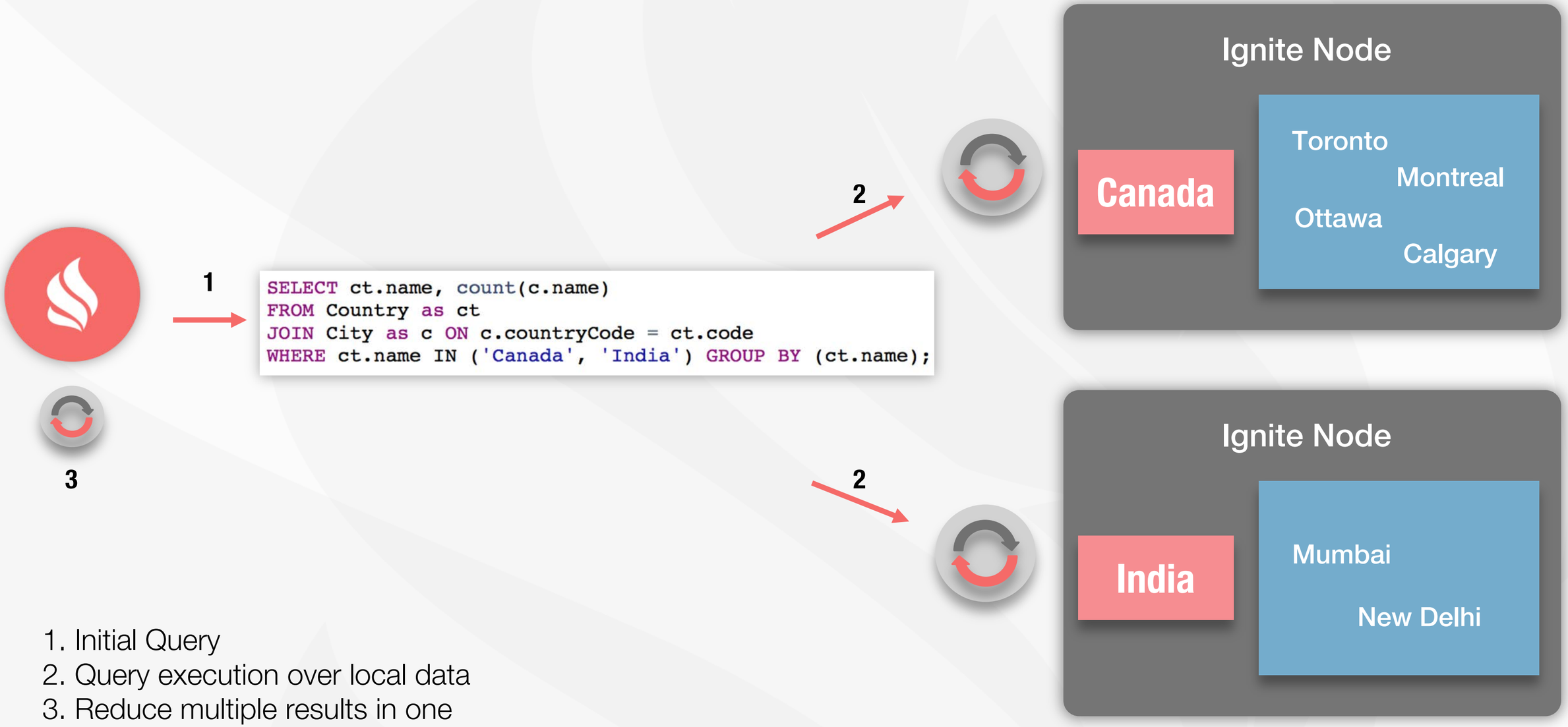
Affinity Collocation



Collocated Distribution



Collocated Joins



1. Initial Query
2. Query execution over local data
3. Reduce multiple results in one

Any Questions?

Thank you for joining us. Follow the conversation.
http://ignite.apache.org

@vkulichenko
#apacheignite