TWJUG

Taiwan Java User Group

Java Meeting – 10/04/2014

Just another Key/Values store?

ORACLE NOSQL – A NEW ALTERNATIVE FOR THE ORACLE RDBMS DATABASE?

Gunther Pippèrr – IT-Architect – Consultant

Background

Gunther Pippèrr is working intensively for more than 15 years with the products of Oracle in database / application server and document management.

Mr. Pippèrr has deep knowledge about the structure of complex IT architecture and successfully implemented this in practice.

Mr. Pippèrr has a degree in computer science as Dipl. Ing. Technische Informatik (FH) - FH Weingarten.

Functional Expertise

- IT Systems Architect
- Technical project manager
- Design and implementation of database applications
- Design and implementation of IT infrastructures for data management

Industry Expertise

- High-Tech
- Real Estate
- Utility
- Communications

Web

- German: http://www.pipperr.de
- English: http://orapowershell.codeplex.com

Selected Experience

Software manufacture telecommunication
- Architect, database design and implementation of a mass data processing system for communication data

German technology group
- Architect and project management, database design and implementation of an order management for control of external contractors for the language service provider of a German technology group.

Utility Consulting
- Architect and technical project responsibility for a smart metering portal for meter data collection and asset management

Architect and technical project responsibility for IT infrastructure projects, such as:
- Central data storage for Munich hotel group with over 25 hotels worldwide
- Data collector for monitoring Russian cable operator
- Redundant database cluster infrastructure for various Web applications such as insurance portals , fond platforms, loyalty cards

Huge Munich building contractor
- CRM and tendering portal

LinkedIn Contact

LinkedIn

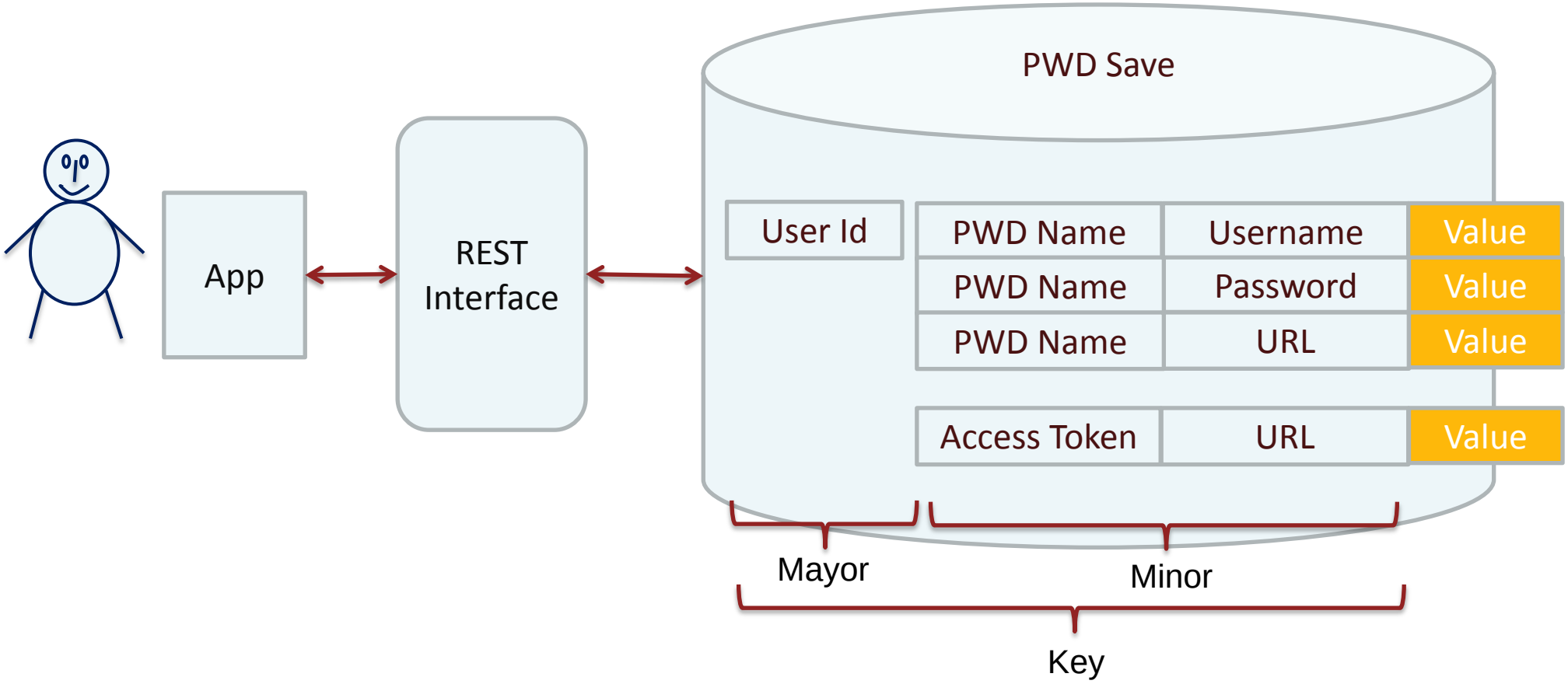
https://www.linkedin.com/pub/gunther-pipp%C3%A8rr/5b/623/23

Use Case

Our challenge to solve

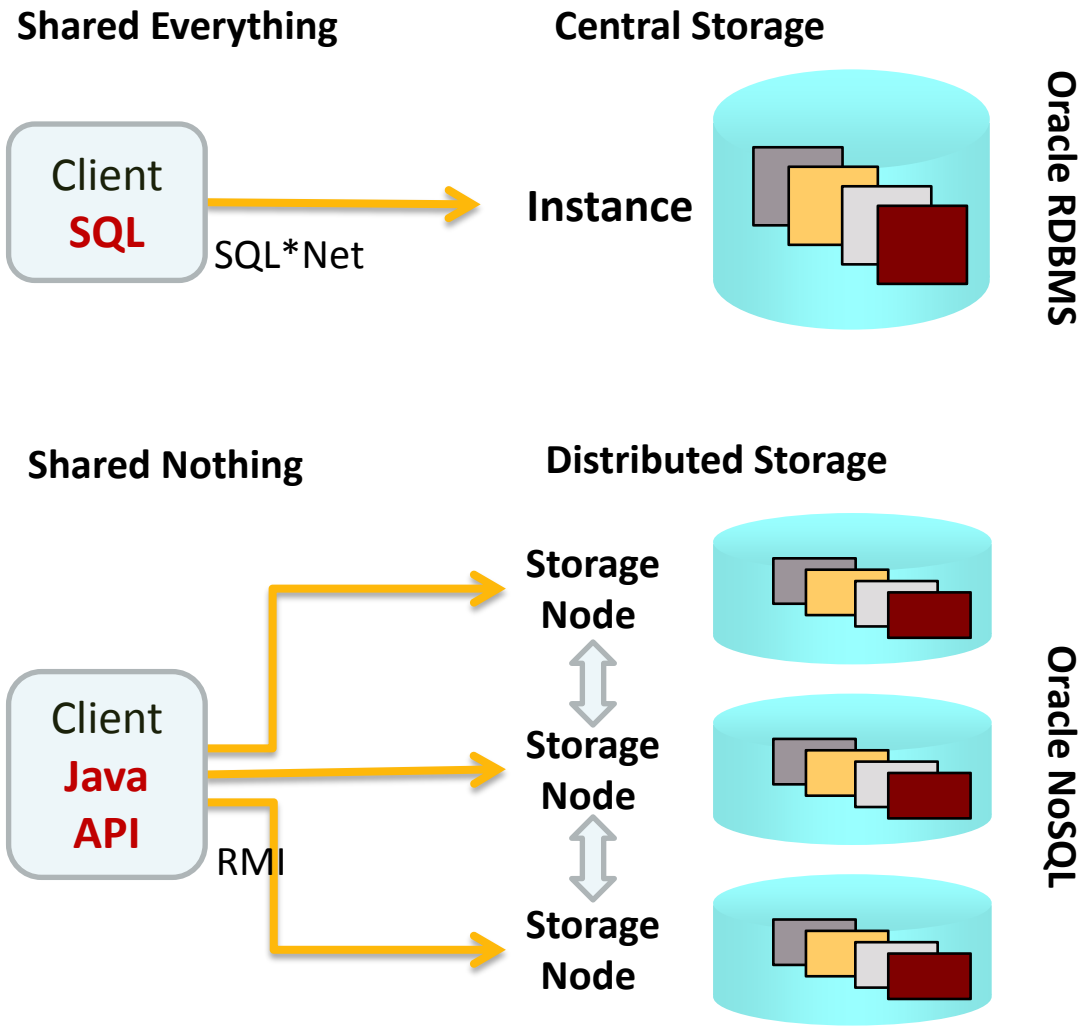
Our Use Case – Internet Simple Password Store

- We like to build:
  - A secure password store for identity credentials



How we can store the Data?

Performance, Security, Availability


Shared Everything
Central Storage
Client
SQL
SQL*Net
Instance
Oracle RDBMS
Shared Nothing
Distributed Storage
Client
Java
API
RMI
Storage
Node
Storage
Node
Storage
Node
Oracle NoSQL

Use for each Problem the right tool

- NoSQL – General Concept
  - More a approach for data management and processing then a special technologies

- Basic Concept

  - Distributed, scalable solutions (Scale-out Concept)
    - **CAP** Theorem – **C**onsistency – **A**vailability - **P**artition Tolerance
    - **BASE - B**asically **Available**, **S**oft-state, **E**ventually
      versus
      ACID – Atomicity, **C**onsistency, **I**solation, **D**urability

  - Highly specialized solutions for exactly one problem with the handling of very large or very quickly required data

    - Dozens of solutions available in the market

NOSQL – Definition

ORACLE NOSQL DATABASE

- A real NoSQL database is:
  - none relational ☑
  - Schema free ☑
  - distributed multi-node architecture ☑
    - Goal: a shared nothing cluster solution
  - Simple replications mechanism ☑
  - A easy API (**Not only SQL = NoSQL**) ☑
  - Not necessarily a ACID Consistency Model ☑
  - Open Source ☑

But – the major disadvantage
=> Usually no easy query language
=> Complex joins are not so easy to implement

Classification Oracle NoSQL in the NoSQL World

- Key-Value
  - One key points to a set of data
    - Oracle 

- Document oriented
  - All data of a data set is organized in one document
    - Lotus Notes
- Graph
  - networked structures
    - Neo4J

Idea - Key-Value store database

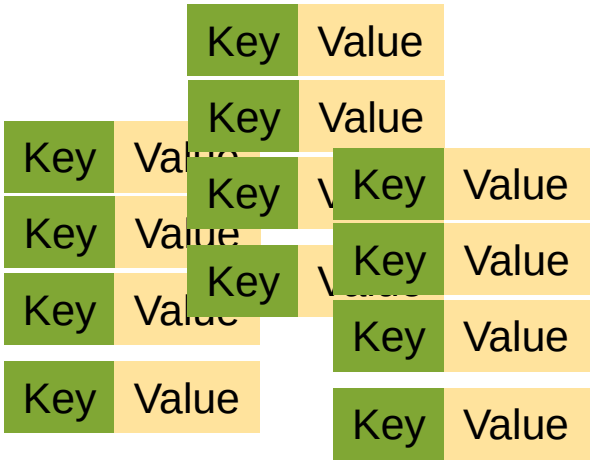
- Oracle NoSQL – a pure Key Value Store
  - A distributed and scalable high available database
  - Consistent hashing approach

Client Software

Intelligent driver

- Implementation of the Hash Mechanisms
- Knows the status of the store
- Read the partition Map at the first connect to the store

ORACLE NOSQL DATABASE

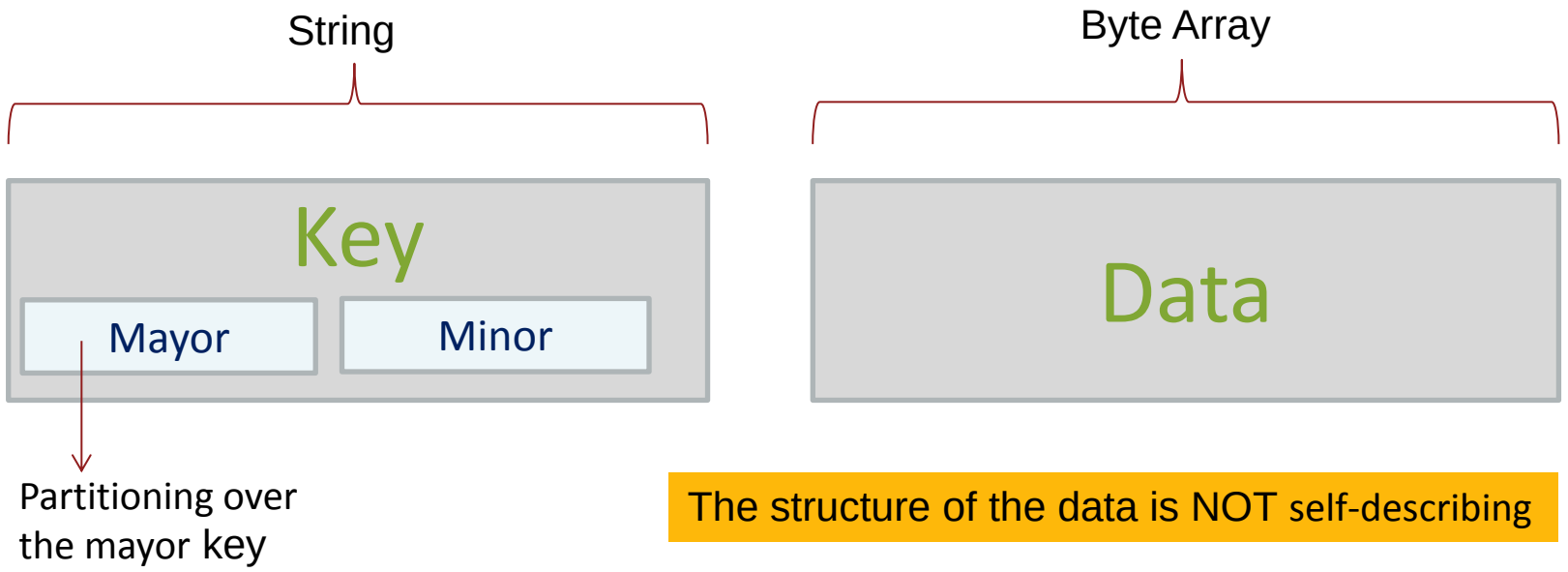


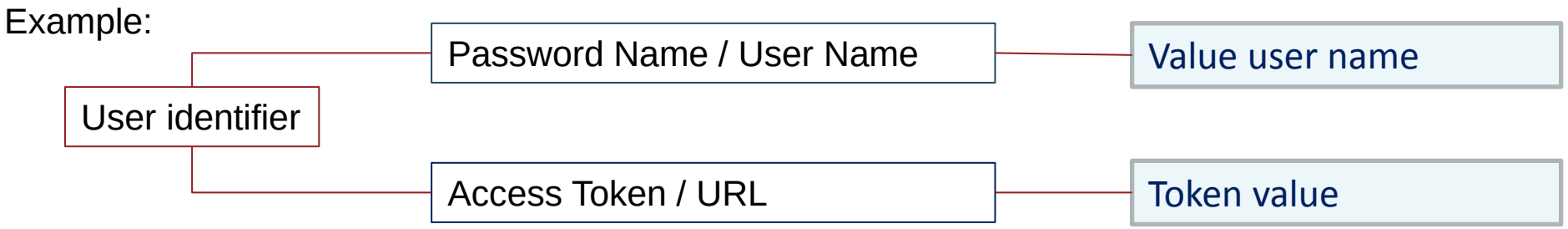
Data store is distributed over all nodes

One ACID Transaction is possible for all records of a mayor key with one Single API call.

The Mayor –Minor Key Concept

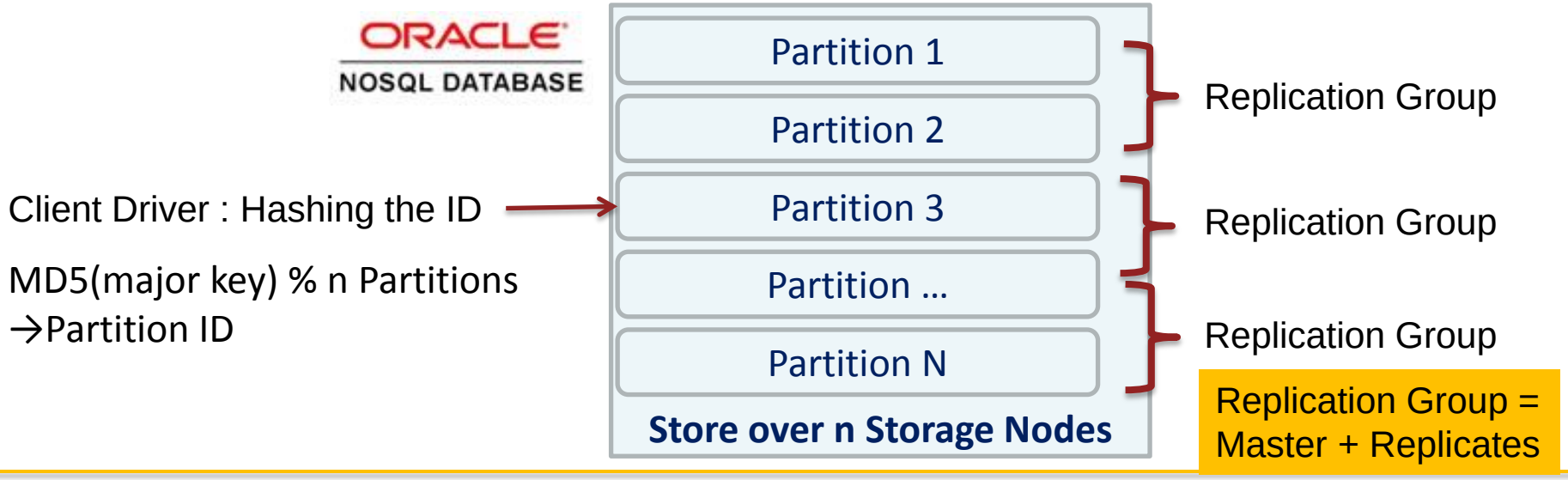
- Key – Composed of two components





Implementation

- Key = Hash Mayor Key => Partition Mapping
  - When you create the store in the first step a fixed count of partitions will be created
    - Each Partition corresponds essentially to his own Berkeley DB „Database“
    - The Hash of the key determine the partition
    - The partition will be stored redundant over the store



Query the store

- Java API
  - Alternative C API (Enterprise Feature!)
  - Command Line Interface for maintenance
- The key and ONLY the Key can be queried
  - For a search in the values all the data in the store have to be read
  - For a count you have to iterate over all values

Variants / versions of the Oracle NoSQL

- CC – Community Edition
  - Open Source Code
    - Open Source with the GNU Affero General Public License
  - Support is possible on a annual basis
- EE – Enterprise Edition
  - Full support
  - Support of Oracle external tables - exchange data with the normal Oracle database
  - Oracle Event Processing
  - RDF Adapter
  - SNMP Monitoring
  - Jena Adapter (Graph/SPARQL)
  - Since Version 3 - Security options with the Oracle Wallet

Integration in the Hadoop Ecosystem

Store results

Hadoop Map Reduce

Read the Data from the NoSQL Store

pure Java

Hive

Hadoop File system

ORACLE NOSQL DATABASE

Read as External Table

Read as External Table

11g
12c

ORACLE DATABASE 12c

Only EE Edition of the Oracle NoSQL!

Evolution form Key-Value / Avro to table

- Release 1 – Key Value

  - Pure Key-Value concept
  - No „Data Dictionary“ – Entire logic in the application
- Release 2 – Avro

  - Schema Definition with Apache Avro Standard
  - Declaration of the structure of the Key-Value Data inside the database AND in the application necessary
- Release 3 – Tables

  - Structure definition as table
    - A secondary index on columns of the table is possible

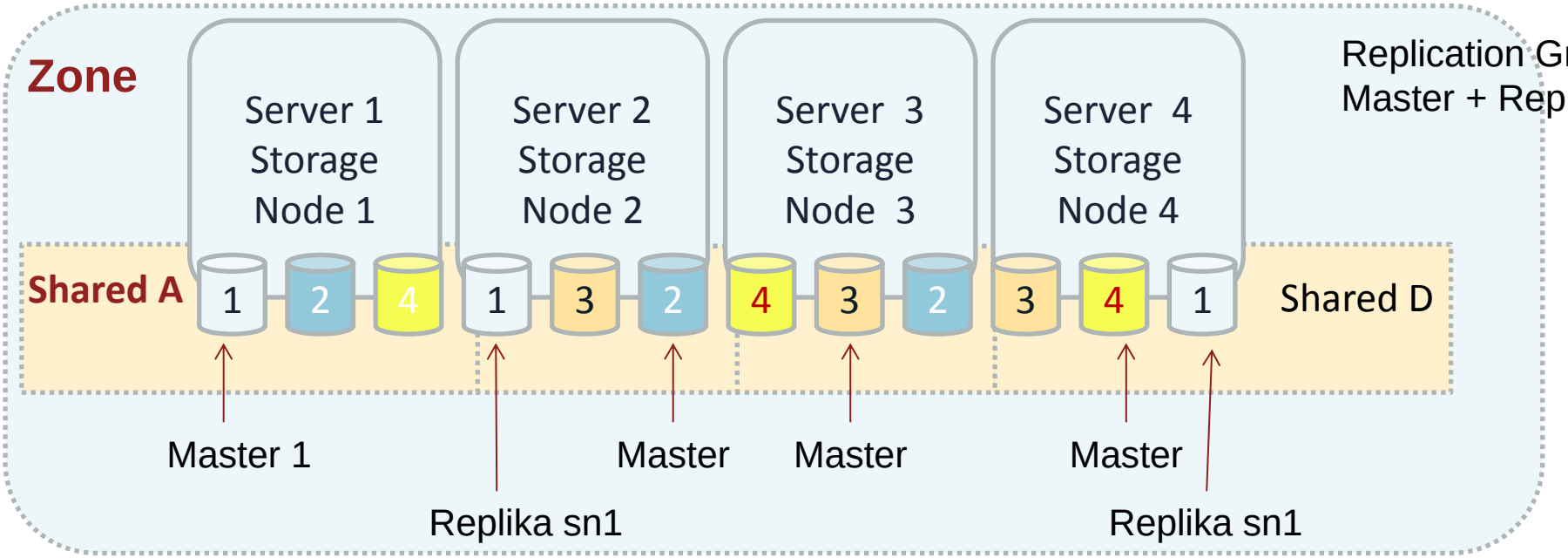
The architecture of the Oracle NoSQL database in detail

A distributed key value store

The store = the database

- Store – all nodes together over all storage nodes

  Example of a store with a replication factor of 3



The replication factor is defined as „master + count replicates“.
In our example one master and two replicates = replication factor 3
Zone = one store node environment

Overview

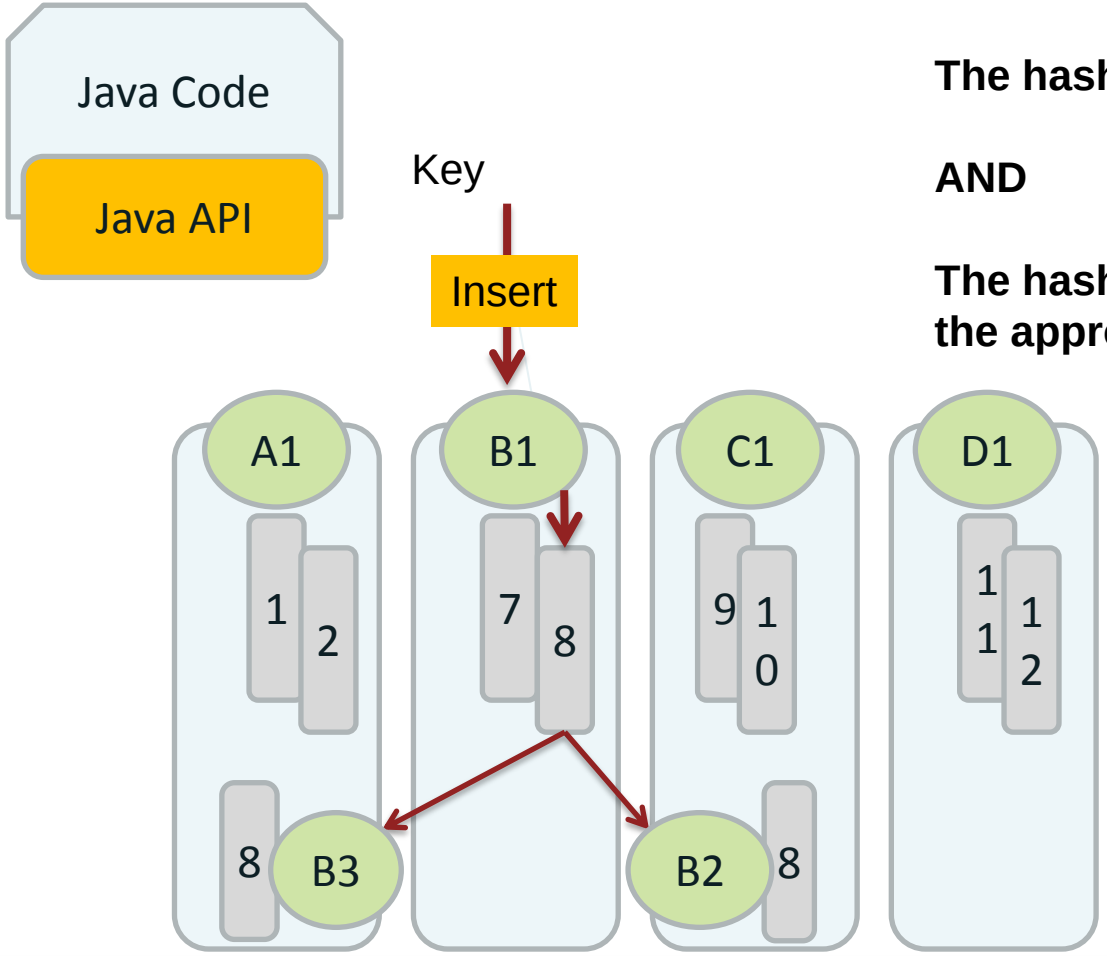
- Depending on the replications factor the data will duplicated over the store
- The count of partitions will be distributed over all master nodes
- A topology describes the whole store

- Example– 4 nodes - replication factor 3 – 800 partitions
  - One Master per node + two replica per storage node
  - 800 partitions distributed over the four master Nodes
    - 200 for each master
  - Inserts only over the master node
    - The Master Node distribute the data to the replicas
  - It is possible to read the data from each node

The count of partition can not be changed after the create of the store!

The distribution of the data in the store

- The client API hashes the key und distributes the Key over the count of partitions



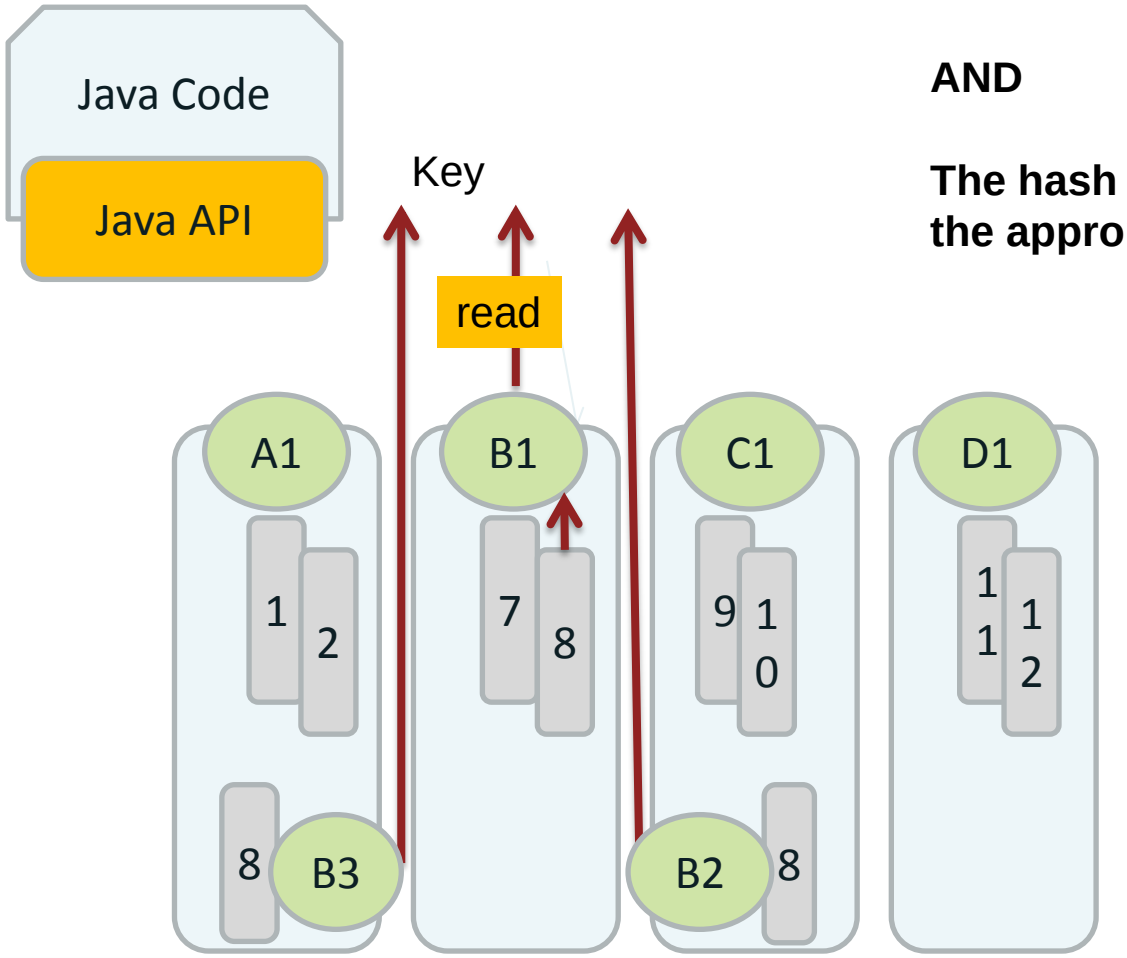
The hash of the key defines the partition

AND

The hash defines the master node holds the appropriate partition.

The distribution of the data in the store

- Read is possible from each replicate



The hash of the key defines the partition

AND

The hash defines the master node, holds the appropriate partition.

Internal management

- Admin Service for the internal management
  - Separate admin database
  - Separate port
- The Admin Service is responsible for implementing of the Master – Slave concept
  - If one Node fails a new master will be defined

Availability

- What happens in case of a failure of a storage node?
  - If the Master is unavailable – it can now not longer be written to the store?
    - Admin Process supervise the whole store
      - Has his own Admin Database
      - Recognizes errors
      - Select from the remaining Replicas the replica with the highest LSN (log sequence number) for a new master

The developer has to choose :

- **Consistency** and **Availabitlity**

OR

- **Availabitlity** and **Partition Tolerance**

Consistency => all nodes see the same data at the same time
Availability => a guarantee that every request receives a response about whether it was successful or failed
Partition tolerance => the system continues to operate despite arbitrary message loss or failure of part of the system

If the developer decide to use **ReplicaAckPolicy.ALL** you will get an error message if a node fails!

The engine (1)

- The Java Version of the Berkeley DB
  - JE Version 5.0.83 - NoSQL 2.2.18
  - JE Version 6.0.14 - NoSQL 3.0.14
  - Each partition is a database
  - Berkeley DB Replication for the master slave concept
  - The DB files can be analyzed with the Berkeley DB classes

Show the version with:
java -classpath ".\lib\kvclient.jar;.\lib\kvstore.jar;.\lib\je.jar" com.sleepycat.je.util.DbVerify -**V**

Tipp:
You can configure the original Berkeley DB Mechanism per Storage Node
See: http://www.pipperr.de/dokuwiki/doku.php?id=nosql:log_file_verhalten_oracle_nosql_db_11gr2

The engine (2)

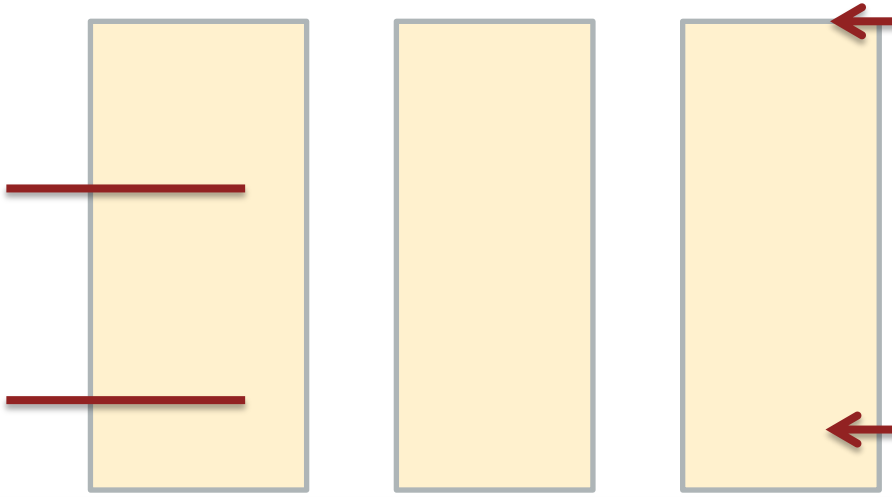
ORACLE BERKELEY DB

- Particularity of the Berkeley DB:
  - Redo Log entries and the data are stored to the same „data“ file
  - Background Jobs take care about the cleaning of the unnecessary entries

je.cleaner.threads
Number of Cleaner Threads

je.cleaner.bytesInterval
Start cleaning if x Bytes written

je.cleaner.minAge
Only clean if x files are older then the latest



je.log.fileMax
Maximal Size of the log

je.cleaner.minUtilization
% must be used of this file

je.cleaner.minFileUtilization
Clean Logfile, if this value is reached

je.log.bufferSize
Minimal Size of the log

Example – Analyze the Data files

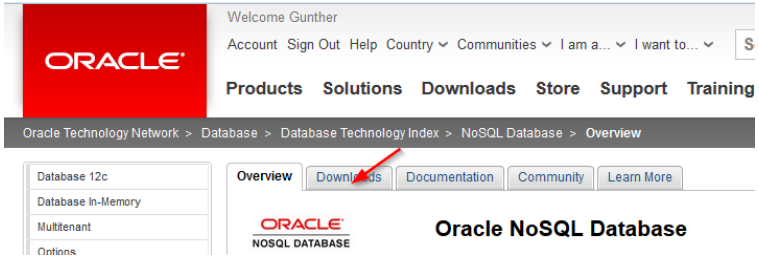
- Read the size of the DB with the Berkeley classes:
  - Ensure the correct class path to the jar files
    - export KVLIB=/opt/oracle/produkt/11.2.0/kv-2.1.8/lib
    - export KVCLASS=$KVLIB/kvclient.jar:$KVLIB/kvstore.jar:$KVLIB/je.jar
  - Path to the data files
    - export JEENV=/opt/oracle/kvdata/GPIDB/sn1/rg1-rn1/env/
  - Analyze with:

```
java -classpath $KVCLASS com.sleepycat.je.util.DbSpace -h  $JEENV


File     Size (KB)   % Used
--------  ---------  ------
00000000     110808      21

 TOTALS      110808      21
(LN size correction factor: NaN)
```

Installation

Unzip Java archive, create directory , ready?

Download Software from :
http://www.oracle.com/technetwork/database/database-technologies/nosqldb/overview/index.html



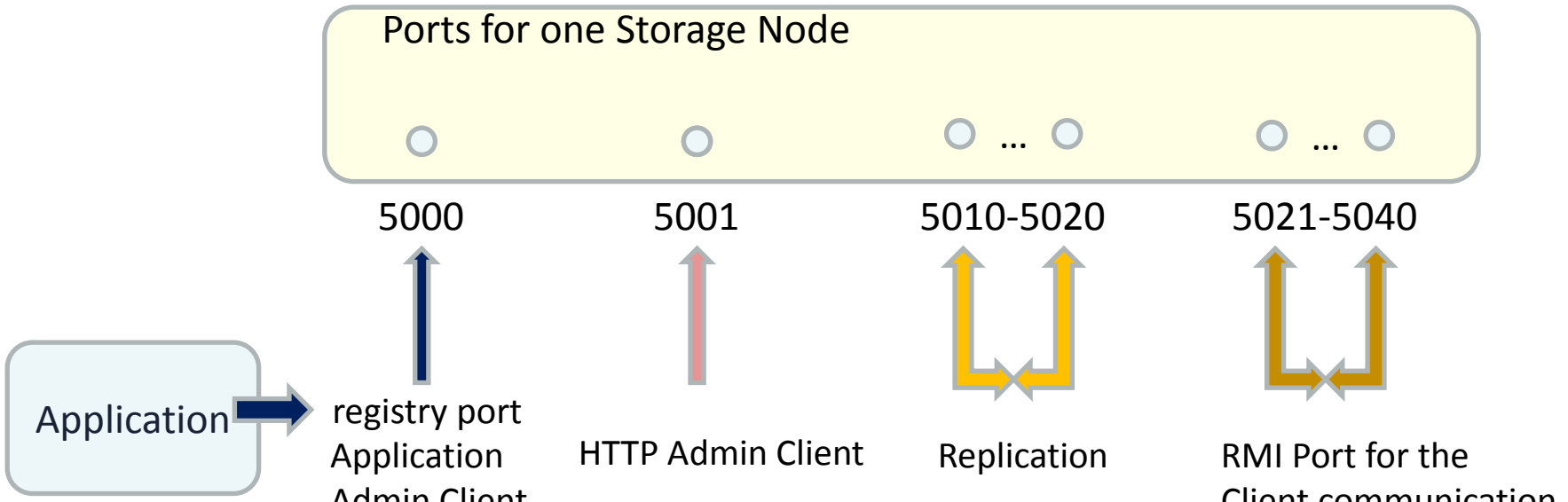
The planning of a NoSQL Installation - Network

- Network Infrastructure
  - Physic
    - High performante internal communication between the nodes necessary
      - pay attention for low latencies in the network
        - » Keyword Infiband
        - » Own 10Gbit Network?
  - Define TCP Ports
    - For the internal communication of the Storage nodes
    - For the communication of the Clients with the Master nodes
      - One Port for external connection + one Port for the Admin Console + ServicePortRange for the RMI Connection from the client
  - Maintainability
    - SSH between the den Storage Nodes
      - Script for maintenance over many nodes

Define some company standard , with the time you will get a lot of environments!

Planning of a NoSQL installation - Ports

- Necessary ports - Network Infrastructure



Parameter when equipped with the "makebootconfig"

-port -admin -harange -servicerange

Parameter for changing with the "change-policy -params "

haPortRange servicePortRange

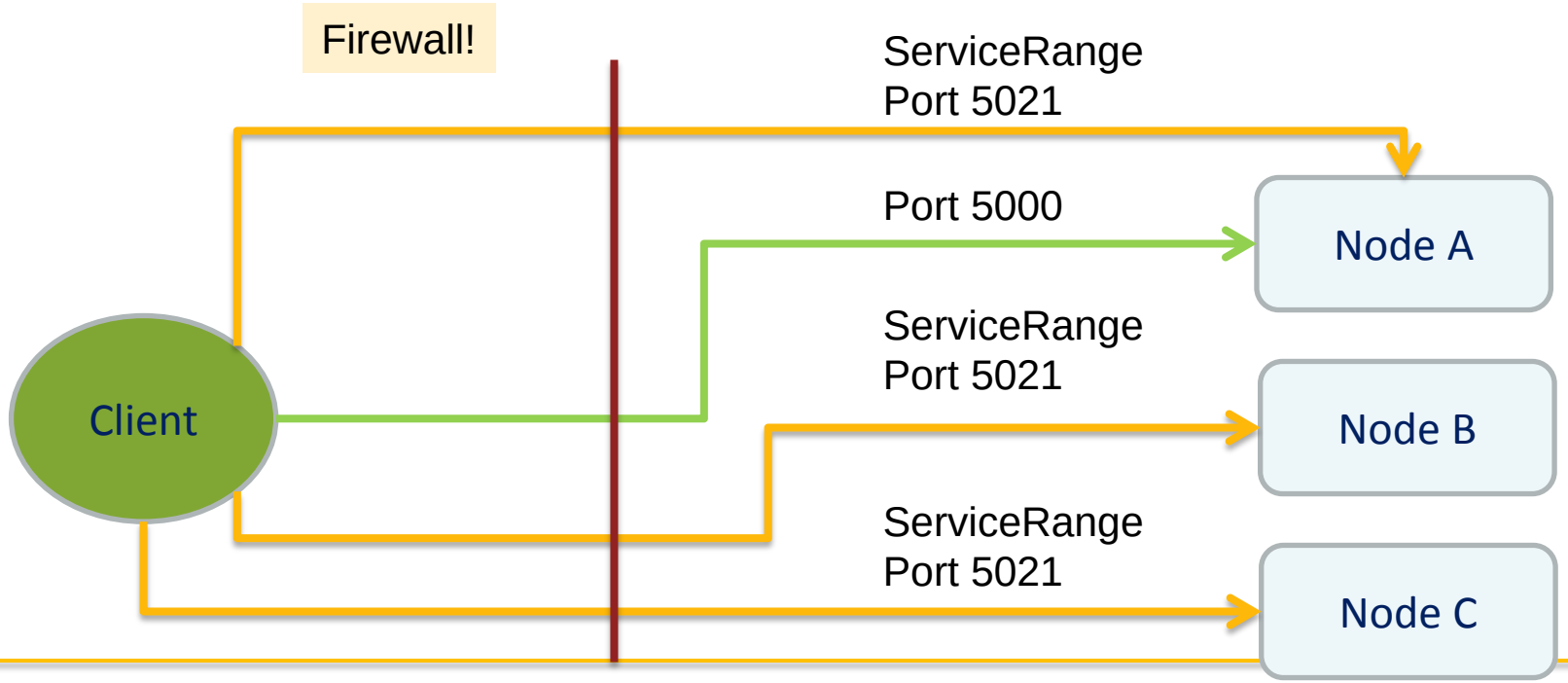
Oracle NoSQL Client Connect

- Access from the Client to the „registry port“
  - For example Port 5000 on node A

-port

- The query is take through over the Java RMI protocol on the auf den respective node
  - Define enough remote ports!

-servicerange



Planning of a NoSQL Installation - Hardware

- CPU and Memory
  - Cache Size can be adjusted from 2 until n GB
    - Watch carefully GC effects of Java
      - Oracle NoSQL is explicit optimized for clean, small objects sizes to not overwhelm the Java garbage collector
- Storage Infrastructure
  - One Store Node can server also more then one Master und replicates
    - If you have more then one Storage node on a server you should use separate disks/luns to have best I/O on the server
      - In the Background the “Cleaner” processes need some I/O resources

Planning of a NoSQL Installation – Disk space

- Disk space
  - Calculation is not entirely trivial
    - The Berkeley DB store all actions like the redo data in the same file
    - Not until some Thresholds are reached the data files will be optimized
    - => The delete of data enlarge the store in the first step!

Rule of thumb:
Pro Master/ Replika => Count of Transactions (Insert/Update/Delete) * (Net Data Size + Overhead)

=> Filling of the data files per Default optimized to 40%

Installation

- Preparation on each Node
  - Java JDK installation and create Store User
  - Setup SSH Connection
  - Create the KVROOT (Data Storages!)
- Oracle NoSQL installation
  - Extract the code – create the KVHOME
  - Create the Store Node Base directory (KVROOT)
  - Create the Boot configuration
  - Start the Admin Node
  - Define a plan for the new store
    - Define the database topologies
  - Rollout the plan
  - test

Per Script: 5min

Install our own local test system

- Download from oracle.com and extract the zip file to a directory
- Create a local store and start the NoSQL Database

```
REM go to your DB directory
cd R:\KVSTORE
REM set the libraries Path
set KVHOME=D:\kv-3.0.14

REM Create the Store
java  -jar %KVHOME%\lib\kvstore.jar kvlite -root R:\KVSTORE
         -store kvstore -host localhost -port 5000 -admin 5001
```

- Start a admin Shell

```
java -jar %KVHOME%\lib\kvstore.jar runadmin -port 5000 -host localhost

kv-> ping
… Pinging components of store kvstore based upon topology sequence #14
… Rep Node [rg1-rn1] Status: RUNNING,MASTER at sequence number: 39 ha
```

Work with the NoSQL Store

Maintenance and operation

Maintenance (1) – command Line

- Kv – For the Store Administration

```
-- -- Command java -jar /opt/oracle/produkt/11.2.0/kv-2.1.57/lib/kvstore.jar runadmin -port 5000 -host nosqldb01
Redirecting to master at rmi://nosqldb03:5000
kv-> ping
Pinging components of store GPIDB based upon topology sequence #516
Time: 2014-06-02 15:12:14 UTC
GPIDB comprises 500 partitions and 3 Storage Nodes
Storage Node [sn1] on nosqldb01:5000    Datacenter: DCGPIDB [1]    Status: RUNNING   Ver: 12cR1.2.1.57 2014-01-10 06:21:46 UTC  Build id: 4d323d1b5495
        Rep Node [rg1-rn1]      Status: RUNNING,REPLICA at sequence number: 1,937,521 haPort: 5011
        Rep Node [rg2-rn1]      Status: RUNNING,MASTER at sequence number: 1,949,809 haPort: 5012
        Rep Node [rg3-rn1]      Status: RUNNING,REPLICA at sequence number: 1,934,227 haPort: 5013
Storage Node [sn2] on nosqldb02:5000    Datacenter: DCGPIDB [1]    Status: RUNNING   Ver: 12cR1.2.1.57 2014-01-10 06:21:46 UTC  Build id: 4d323d1b5495
        Rep Node [rg1-rn2]      Status: RUNNING,REPLICA at sequence number: 1,937,521 haPort: 5011
        Rep Node [rg2-rn2]      Status: RUNNING,REPLICA at sequence number: 1,949,809 haPort: 5012
        Rep Node [rg3-rn2]      Status: RUNNING,MASTER at sequence number: 1,934,227 haPort: 5013
Storage Node [sn3] on nosqldb03:5000    Datacenter: DCGPIDB [1]    Status: RUNNING   Ver: 12cR1.2.1.57 2014-01-10 06:21:46 UTC  Build id: 4d323d1b5495
        Rep Node [rg1-rn3]      Status: RUNNING,MASTER at sequence number: 1,937,521 haPort: 5011
        Rep Node [rg2-rn3]      Status: RUNNING,REPLICA at sequence number: 1,949,809 haPort: 5012
        Rep Node [rg3-rn3]      Status: RUNNING,REPLICA at sequence number: 1,934,227 haPort: 5013
```

- kvshell – Select / insert data

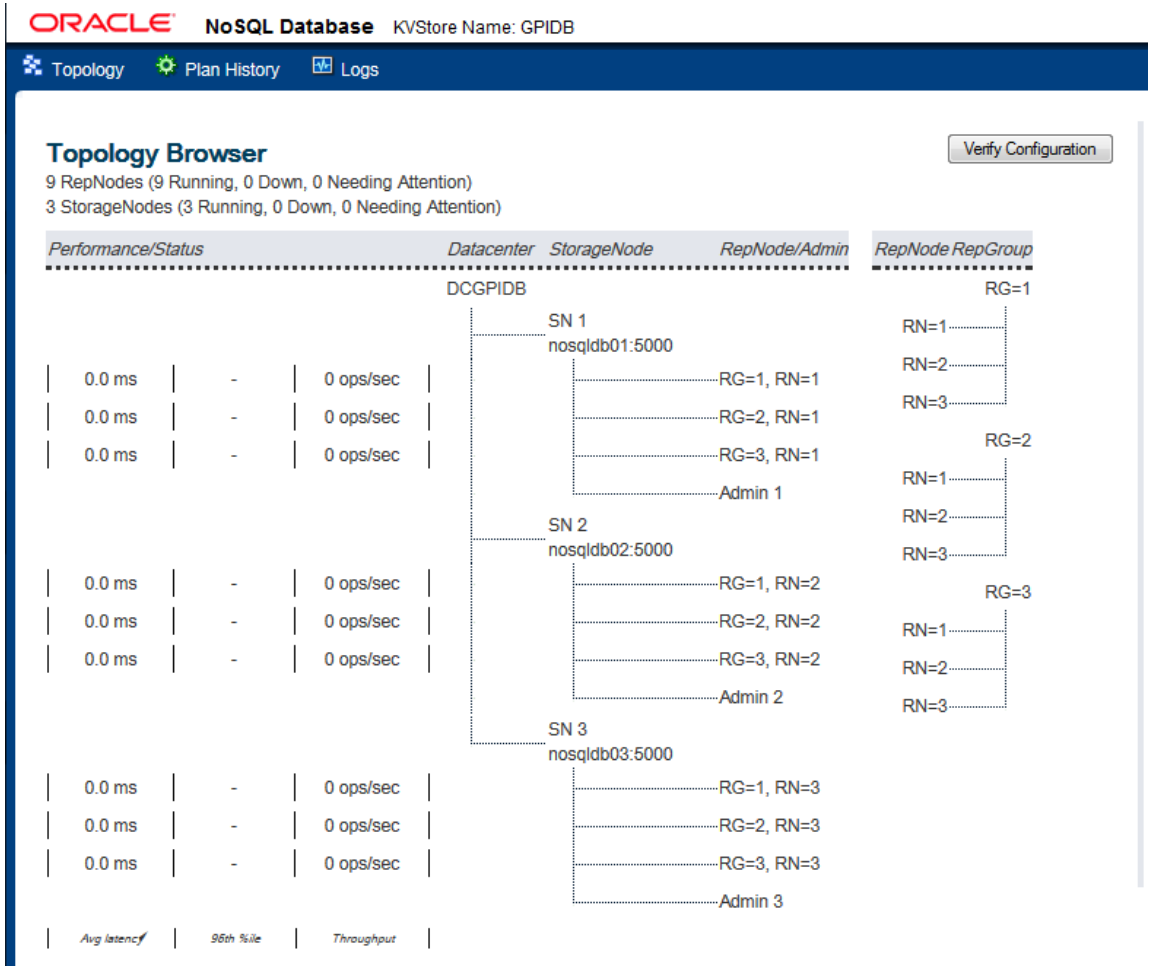
```
-- -- Command java -jar /opt/oracle/produkt/11.2.0/kv-2.1.57/lib/kvcli.jar -host nosqldb01 -port 5000 -store GPIDB
kvshell-> aggregate -count
count: 1010204
```

```
kvshell-> put -key "/Gunther/name" -value "Pipperr"
Operation successful, record inserted.

kvshell-> get -key "/Gunther/name"
Pipperr
```

Maintenance (2) – Web interface

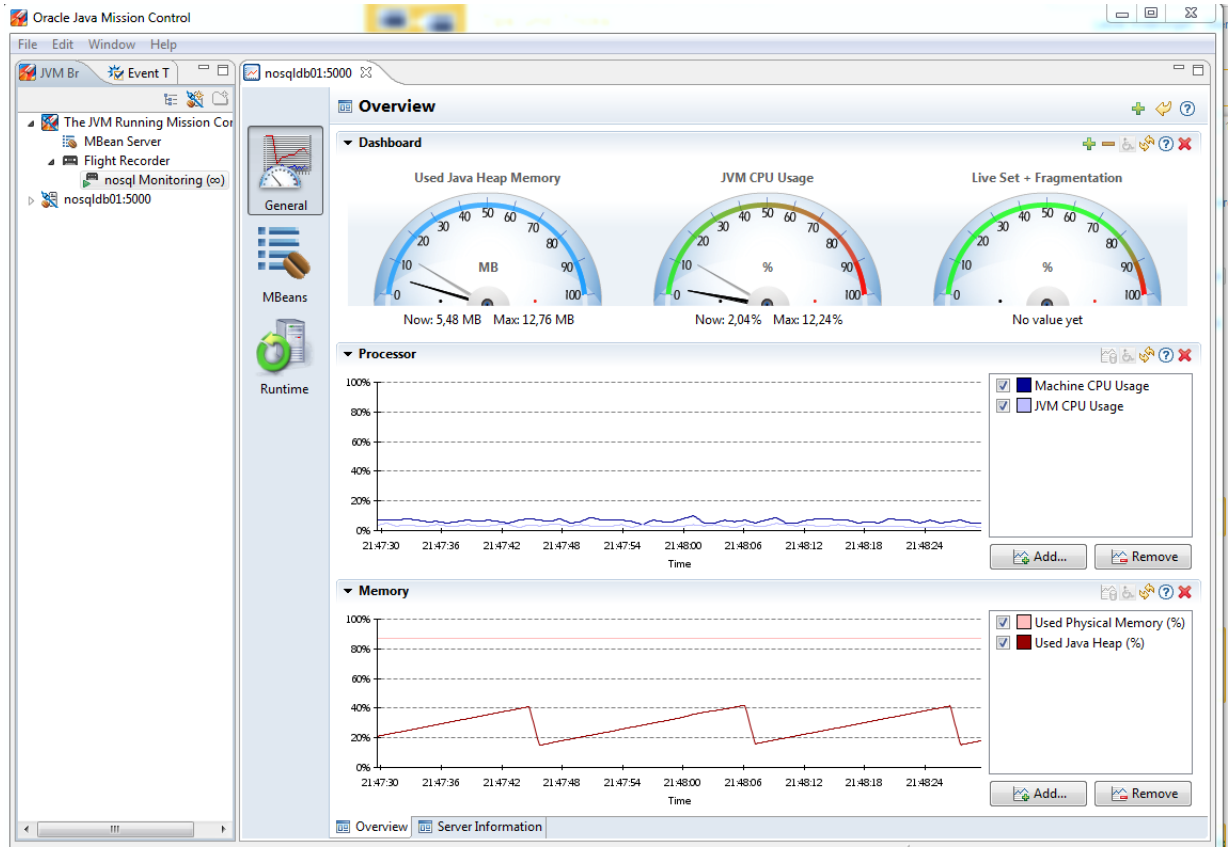
– Web interface based on a internal jetty server

http://nosqldb03:5005/



JMX Interface

- Start JMC – Java Mission Control
  - JAVA_HOME/bin/jmc.exe
  - Connect over the „registry port“



Reliability

Availability and Backup

Backup per Snapshot possible

- Backup:
  - A complete Store can be saved consistent per Snapshot
  - pay attention for enough disk space!
- Restore
  - A Store can be completely reset
    - Create a Store snapshot
    - Stop the Store
    - Provide the Snapshot (Link in the File system for example)
    - Start the Store, Storage Nodes read die Data, create the data again and DELETE the snapshot!
  - Use as a kind of import
    - The generated files can be read from an other store

Performance

I/O and Network

Performance considerations

- read
  - Network
    - All records must be read over the network
- write
  - I/O
    - Background process need enough recourse for the optimization of the data files!

A simple Count(*) have to read all data!

ALL record are processed on the client side!

Security?

Oracle NoSQL and Security?

Security?

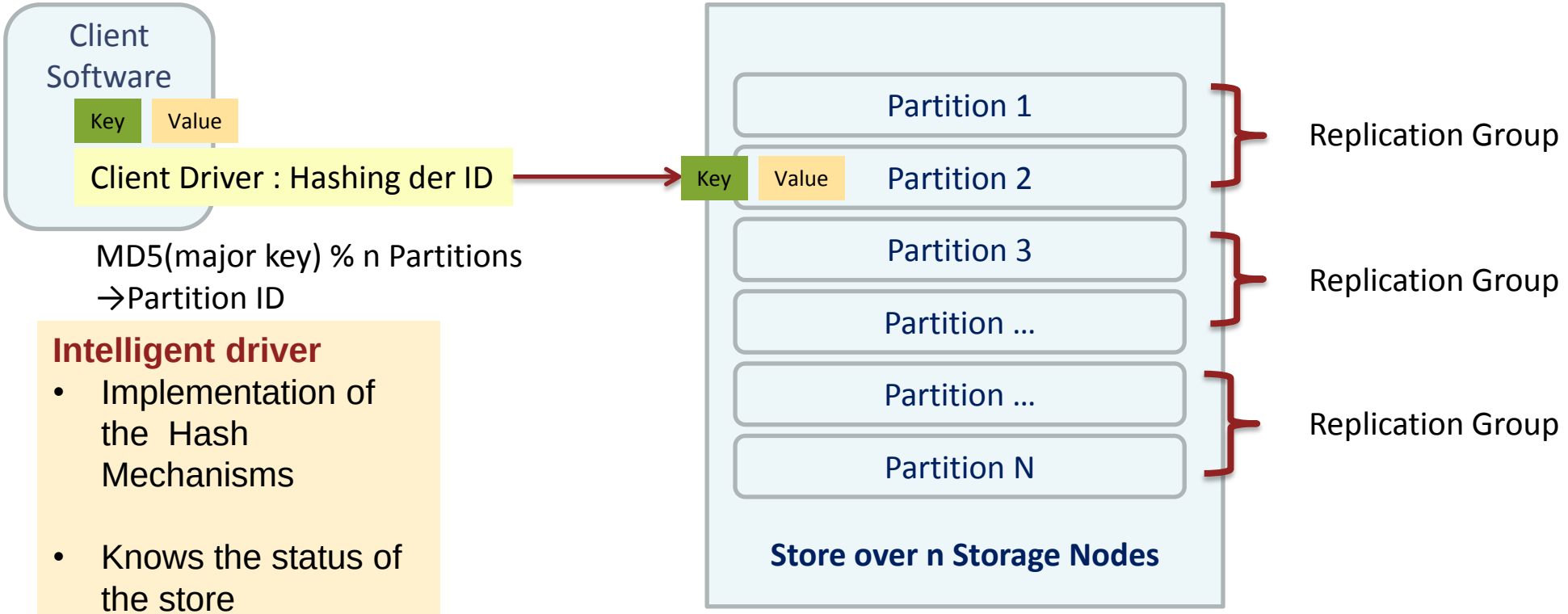
- Version 2 ?
  - The developer has to care about it – not implemented

- Since Version 3
  - SSL encoding
  - Password security
    - But Wallet usage is a EE feature!

Read and write to the store with Java

No SQL necessary – all is done in Java

Overview



Intelligent driver

- Implementation of the Hash Mechanisms
- Knows the status of the store
- Read the partition Map at the first connect to the store

Replication Group = Master + Replication Node

First – Connect to the store (1)

- Create the connection to the store
  - Set the store properties with the class **KVStoreConfig**

```
// Node liste
String[] kvhosts = {"nosqldb01:5000","nosqldb02:5000","nosqldb03:5000"}
```

Node

Port

```
// Config erzeugen
KVStoreConfig kvconfig = new KVStoreConfig("GPIDB", kvhosts);
```

Store Name

First – Connect to the store (2)

- Create the connection to the store
  - Define the Read and Write Rules

Rule read

```
// Define Consistency
kvconfig.setConsistency(Consistency.NONE_REQUIRED);

// Define Durabiltiy
kvconfig.setDurability(Durability.COMMIT_NO_SYNC);
```

Rules write

```
// Connect to the store
KVStore kstore = KVStoreFactory.getStore(kvconfig);
```

Open connection to the Store of a Factory

Do not forget to close the Store!
kstore.close();

First Example - Write the data

- Address the DB object over the key
- Create the key

```
Key k = Key.createKey("USER_ID", "PWD_NAME_GOOGLE/USERNAME");
```

Minor Key

Mayor Key

- Write the data

```
// Wert als Byte Array anlegen
byte[] b = ("gpipperr@googelmail.com").getBytes();

// Wert in den Store schreiben
kvstore.put(k, Value.createValue(b));
```

Write the data to the store

First Example - Read the data over the key

- Create the key

```
Key k = Key.createKey("USER_ID", "PWD_NAME_GOOGLE/USERNAME");
```

Minor Key

Mayor Key

- Read data

```
// Read the Data from the Store with the key
ValueVersion vv = kvstore.get(k);

// Read the Daten Value
Value vdata = vv.getValue();

// Recreate the Values
String username= new String(vdata.getValue());

// Work with the data
System.out.println(username);
```

read

Example

- Demo

Are we reading the last version of data?

TRANSACTION CONCEPTS

The consistency of the data – Transactionality

write

- In one session transactionality for write one mayor key is possible
- The developer has to decide when a record is apply as written!

Durability

Save: On all nodes to disk

Fast: On one node to the cache

Performance

read

- The developer has to decide if the read record is actual

Consistency

Save: Latest Version

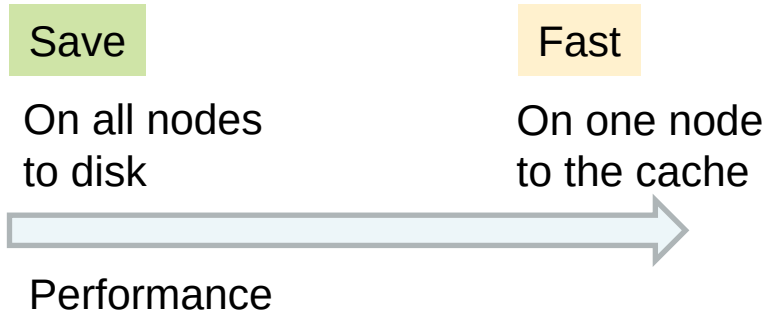
Fast: first available answer

Performance

The write consistency in the Store - Durability

- The developer is responsible for the write consistency!
- Will be defined at the connect to the store
  - Class oracle.kv.Durability
    - Commit-Policy for the Master
    - Commit-Policy for the replicas
    - Replica Acknowledgement-Policy

```
kvconfig.setDurability(
    new Durability(
        // master sync
        Durability.SyncPolicy.NO_SYNC
        // replicat sync
        ,Durability.SyncPolicy.NO_SYNC
        // replica acknowledge
        ,Durability.ReplicaAckPolicy.NONE
    )
)
```



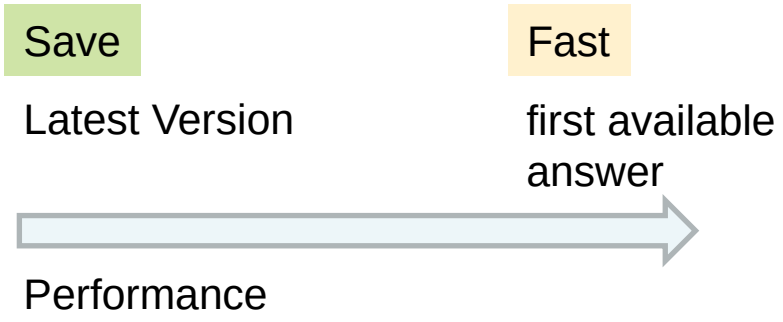
Durability Policies

- **Commit-Policy for the Master and the replicates**
  - **SyncPolicy.SYNC**
    - Wait until the Storage Node write the transaction to the log file and synchronized to the disk
  - **SyncPolicy.WRITE_NO_SYNC**
    - Wait until the Storage Node write the transaction to the log file in the cache (not written to disk!)
  - **SyncPolicy.NO_SYNC**
    - Don’t wait
- **Replica Acknowledgement**
  - **ReplicaAckPolicy.ALL**
    - Wait until all replicates confirm, that all transaction has been written
  - **ReplicaAckPolicy.SIMPLE_MAJORITY**
    - Wait until the simple majority the replicates confirm the transaction
  - **ReplicaAckPolicy.NONE**
    - Don’t wait for the answers of the replicates

The read consistency in the store - Consistency

- The developer is responsible for the read consistency!
- Each key object owns a version information
  - A kind of transaction ID
- The behavior will be defined with the connect to the store
  - read – Class oracle.kv.Consistency

```
kvconfig.setConsistency(Consistency.NONE_REQUIRED);
```



Consistency

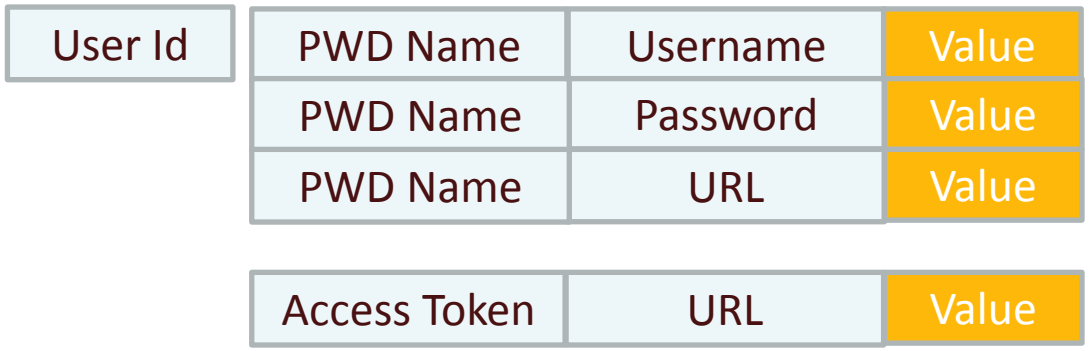
- Consistency.ABSOLUTE
  - The session can read only from the master for this key.
  - This ensures that the data is relay valid
- Consistency.Time
  - The session can read from a replica
  - With the Parameter you can define how „old“ the data can be to accepted as valid data
- Consistency.Version
  - The session can read from a replica
  - The version information will be used instead of the time
- Consistency.NONE_REQUIRED
  - Very result is valid, can be any old

Structured data access

THE TABLE IS BACK

Use the table construct (1)

- Our Key Model



- Our Table Model

Table USER_PASSWORDS

USER_ID | PWD_NAME | USER_NAME | PASSWORD| URL

Table USER_ACESS_TOCKENS

USER_ID | ACESS_TOCKEN | URL | Value

Use the table construct (1.1)

- Create a table definition in the Store with the client utility

```
set KVHOME=D:\entwicklung\libraries\kv-ee-3.0.14

java -jar %KVHOME%\lib\kvcli.jar -host localhost -port 5000
-store kvstore


#Create table
table create -name USER_PASSWORDS

# Add table fields
add-field -name USER_ID          -type STRING
add-field -name PWD_NAME         -type STRING
add-field -name USER_NAME        -type STRING
add-field -name PASSWORD         -type STRING
add-field -name URL              -type STRING
```

Use the table construct (1.2)

```
# Add the primary Key
primary-key -field USER_ID

#Leave table Mode
exit
#create the Table in the Store
plan add-table -name USER_PASSWORDS -wait

#check the Table Definition
show table -name USER_PASSWORDS

#insert first Record of data in to the table with the client tools
put table -name USER_PASSWORDS -json
"{\"USER_ID\":\"gunther@pipperr.de\",\"PWD_NAME\":\"google.de\",\"USER_NA
ME\":\"gpipperr\",\"PASSWORD\":\"Password01\",\"URL\":\"www.google.de\"}"

#create index on the table
plan add-index -name IDX_USER_PASSWORDS -table
USER_PASSWORDS -field USER_ID –wait

#read the data
get table -name USER_PASSWORDS -index IDX_USER_PASSWORDS
```

Use the table construct (1)

- Write the source code to use the table defintion

```
// get the Table API
TableAPI tableAPI = kvstore.getTableAPI();
```

```
//get a reference to the table
Table empTable = tableAPI.getTable("USER_PASSWORDS");
```

```
// Get a Row instance
Row empRow = empTable.createRow();
```

```
// put data in the row
empRow.put("empno", 10);
```

```
// write the data to the store
tableAPI.put(empRow, null, null);
```

Use the table construct (2)

- Read from this table again

```
// get the Table API
TableAPI tableAPI = kvstore.getTableAPI();
```

```
//get a reference to the table
Table empTable = tableAPI.getTable("emp");
```

```
//Create a primary key and assign the field value
PrimaryKey key = table.createPrimaryKey();
key.put("userID", 1);
//Get the matching row
Row row = tableAPI.get(key, new ReadOptions(null, 0, null));
if (row == null) {
        throw new Error("No matching row found");
}
//Print the full row as JSON
System.out.println(row.toJsonString(true));
//Access a specific field
System.out.println("firstName field as JSON: " +
row.get("firstName").toJsonString(false));
```

Example

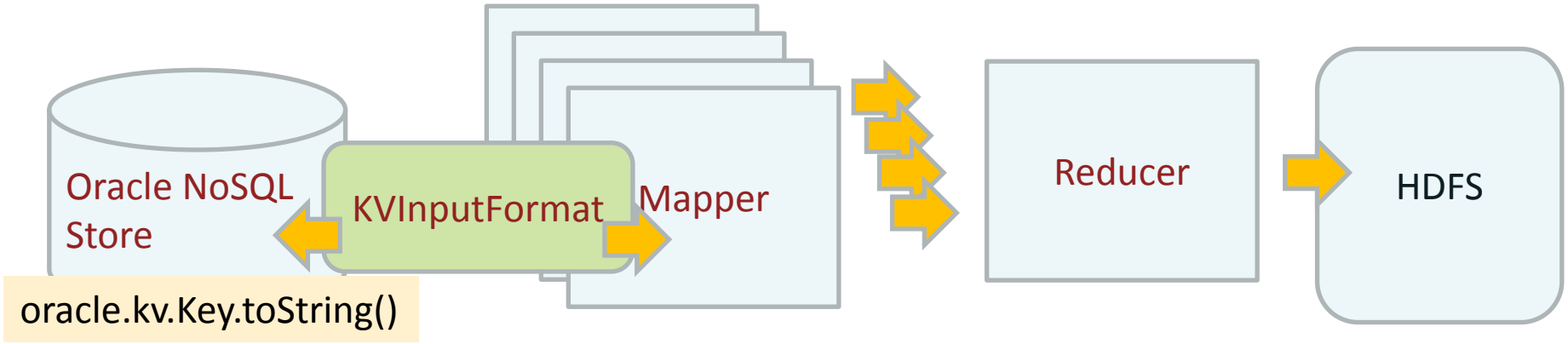
- Demo

Archive your data

INTEGRATION INTO THE HADOOP DATA WAREHOUSE

Oracle NoSQL – MapReduce Hadoop

- Traditional MapReduce Pattern
- Input Format Class – KVInputFormat.class



Error: java.lang.RuntimeException: java.lang.ClassNotFoundException: Class oracle.kv.hadoop.KVInputFormat not found at

lib/kvclient.jar must in the class path!
Like:
hadoop jar secure-pwd-store.jar gpi.secstore.PWDStoreHadoop
-libjars $KVHOME/lib/kvclient.jar
/tmp/nosqlTest2

Better solution -put all in one jar fie

Main Part – Job Configuration

- Define the Input Format Class in the Job Configuratoin

```
// set the Input Format Classe
job.setInputFormatClass(KVInputFormat.class);

//Parameter for the Input Format Class
KVInputFormat.setKVStoreName("KVStore");

String [] kvhostList = {"StoreDB01:5000", "StoreDB02:5000",};
KVInputFormat.setKVHelperHosts(kvhostList);
```

Set the Input Class for the MapReduce Job

Define the connection to the Store

Default: Key and Value are read as Text Format

Example

- Demo

Conclusion

Are the benefits worth the effort?

Why we should use the Oracle NoSQL? (1)

- What is really important for a strategic development?
  - Maintainability
    - Is the product easy to use?
  - Long-term strategy of the manufacturer
    - Is a product cycle from 4-5 year possible?
  - Release and Update Problematic
    - Java 8 and higher? Linux 7?

Why we should use the Oracle NoSQL? (2)

- What is for a strategic development really important?
  - Fits the base concept to my technical needs?
    - Are my requirements changing with the time?
  - Costs

Oracle promises this:
– Version 3 ready
– The Software Berkeley DB since year in use

Use Cases

- Use Cases
  - Personalization of web pages
  - Authentication
  - Stream Processing
    - machine data

Goal:

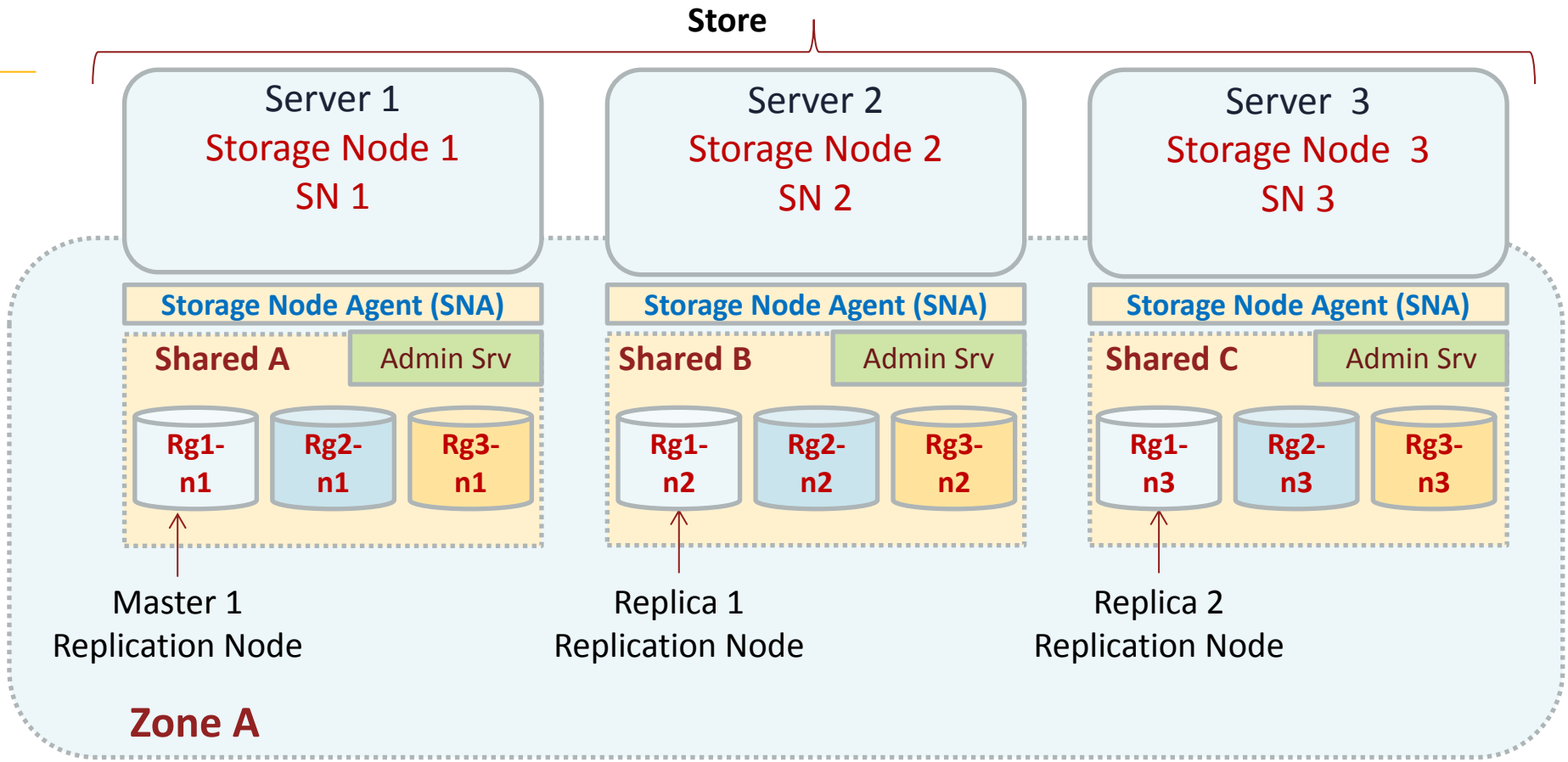
Get for one Key fast as possible the results

Requirement: Provide large amounts of data with low latency – Persistent Cache

Case Study: Web Application

- In a Web application each user can have a very high number of different "bonus offers, vouchers"
- This bonus offers are calculated individually according to the profiles of a user on a weekly base
  - The NoSQL Store holds the UserID as key with a list of all metadata for the bonus offers
    - Avoid complex joins over the DWH with the user profiles
    - Immediate display results on the smart phone or in the Web without a time delay – very low latencies

Requirement: Provide large amounts of data with low latency – Persistent Cache



Architecture of a NoSQL Stores

3 Storage Nodes (SN) on three servers
with a Replication Factor of three = three Replication Groups (RG)

Direct Comparison Oracle RDBMS ⇔NoSQL

Oracle RDBMS	Oracle NoSQL
▪ ~35 years of development	▪ ~2-3 years
▪ Powerful SQL API	▪ Simple Java API
▪ Joins	▪ Joins only over the application
▪ Constraints	▪ ?
▪ Many Features	▪ Pure data store
▪ OLTP	▪ Mostly only read operation
▪ Shared anything approach	▪ Shared Nothing Cluster
▪ A general all purpose weapon	▪ Specialized niche solution

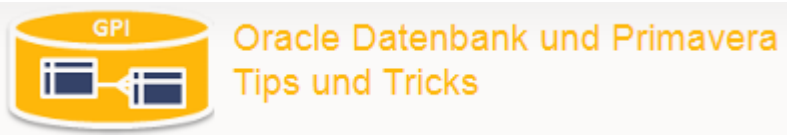
NO:SQL

Oracle NoSQL

Sources

- Oracle - ORACLE
  - English
    http://www.oracle.com/technetwork/database/database-technologies/nosqldb/documentation/index.html
  - German
    http://www.oracle.com/webfolder/technetwork/de/community/dojo/index.html

- See also:



  - German http://www.pipperr.de/dokuwiki/doku.php?id=nosql_datenbank