Spark.

Spark Core ⇒

⇒ ① Definition ② advantages over hadoop ; Spark Architecture ③ features of spark ④ ecosystem components ⑤ RDD meaning, ways to create, features of RDD ⑥ RDD Transformations

- Narrow & wide
  - Narrow (within partition): Map, flatmap, MapPartition, filter, Sample, Union
  - wide (data may be in different partition so shuffle involved): Intersection, Distinct, ReduceByKey, GroupByKey, Join, Cartesian, Repartition, Coalesce

⑦ Actions - first, take, reduce, count, collect, cache → not an action, saveAsTextFile

⑧ Limitation of RDD
- no optimization, no structure
- garbage collection issue,
- Performance degrade with bigger RDD.
- No static typing

⑨ RDD Persistence & caching :- benefits & modes & unpersist
- MO, MOS, MAD, MADS, DO, MO2 & MAD2, OFF_HEAP (serialized)

⑩ Overcome the limitations of RDD.
① Use dataset & dataframes (catalyst optimizer & Tungsten Exe. Engine)
② compile time check using DS & DF
③ DS & DF → more control over data using structure

⑪ Transformations - map, flatmap, mapPartition, mapPartitionWithIndex, filter, union, intersection, distinct, reduceByKey, GroupByKey, SortByKey, join, coalesce, cogroup, cartesian, pipe, repartition, combineByKey, repartitionAndSortWithinPartitions, mapValues, flatMapValues,

(schema must be same; pairRDD only; process RDD with shell script)

Spark Actions:- count, collect, take, top, countByValue, reduce, fold, aggregate, foreach, takeSample, takeOrdered, saveAsTextFile, SaveAsSequenceFile, saveAsObjectFile, countByKey, collectAsList, forEachPartition,
(top → unordered → ordered, head)

- Spark Job flow ⇒ Job → Stage → Task
- Spark performance tuning,
- broadcast variables, accumulators
- RDD lineage → reconstruct the RDD if lost
- UDFs,
- Challanges faced ⇐ transaction like → ACID (incremental approach)
  - outOfMemory → solved with foreachPartition

Transformations on PairRDD:- Internal working

ReduceByKey, foldByKey, combineByKey, PartitionBy(new HashPartitioner(2)), AggregateByKey.

Actions on PairRDD:- countByKey, collectAsMap, lookup(key)

Spark Partitioning -

Spark – spark architecture, RDD, transformation & actions., advantages,, features, (Narrow, wide) limitation of RDD, persistence

Spark SQL – Spark dataframes, datasets, difference., Optimization engine, Java API, Spark Shuffling Data partitioning, Row formats, broadcasts, accumulators

Spark Configuration properties., compression technique

Spark Streaming –

Challanges –

⇒ OOM ⇒ forEachPartition

MapEachPartitionWithIndex → collect.

Hive ⇒ Comma separated values ⇒ used Lateral view & explode

Code redundancy ⇒ modularised code → common utility

Spark tuning ⇒ persist RDD, removed collects used Accumulators V2, calculation of executor memory

Latest data ⇒ Ranking windowing

~~10.97-101. 885: 9~~

Core Java - Concepts, multithreading, collections, design patterns, jdbc, SpringBoot

J2EE - jsp objects, Servelet lifecycle, framework MVC

Rest - Go through interview questions.

Design Patterns - Singleton, Factory Pattern,

MVC - Struts & hibernate

HDFs - Big data concepts,
file read write & configurations
hdfs vs old style
Storage policy.

Mapreduce - functioning
Configuration
parallel processing
interview question

Yarn - Mapreduce vs YARN
uber, components of hadoop & hadoop2

Hive - Architecture, modes, Tez vs MapR
hive vs SQL / Traditional RDBMS
hive vs MapReduce
hive vs Pig
hive queries, UDAF Evaluator method.
Map join, self join, cross join,
purpose ← UDF, UDAF, aggr. functn
Hcatalog, complex types, Serde

compression SNAPPY & file format ORC

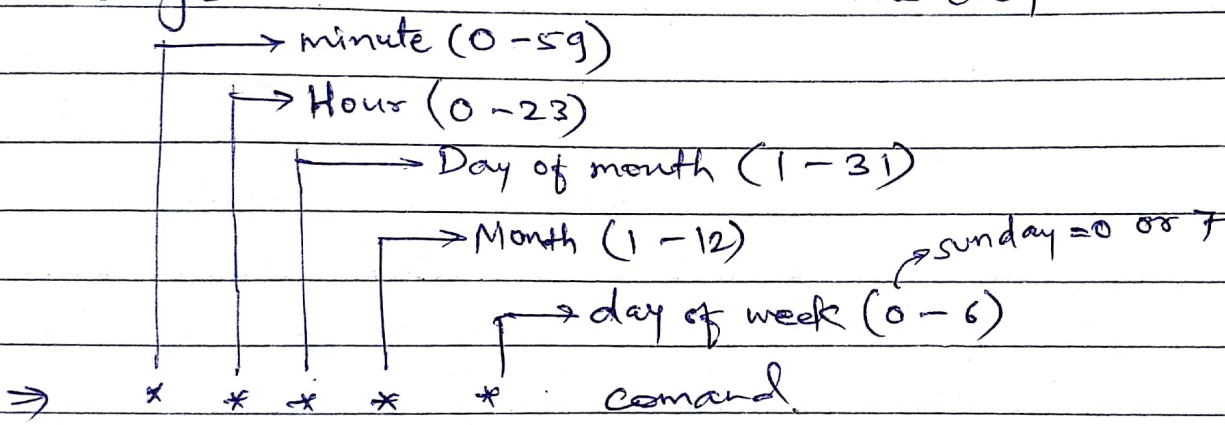
Evaluate method validity

Pig - Architecture
modes, functions, complex types, UDF
↳ extend EvalFunc
exec() method

*/10 11-13 * * * ⇒ Every 10 min bet^n 11 to 13.59 pm

0 * * * * ⇒ 0th minute every hour.

Cronjob –



Spark SQL

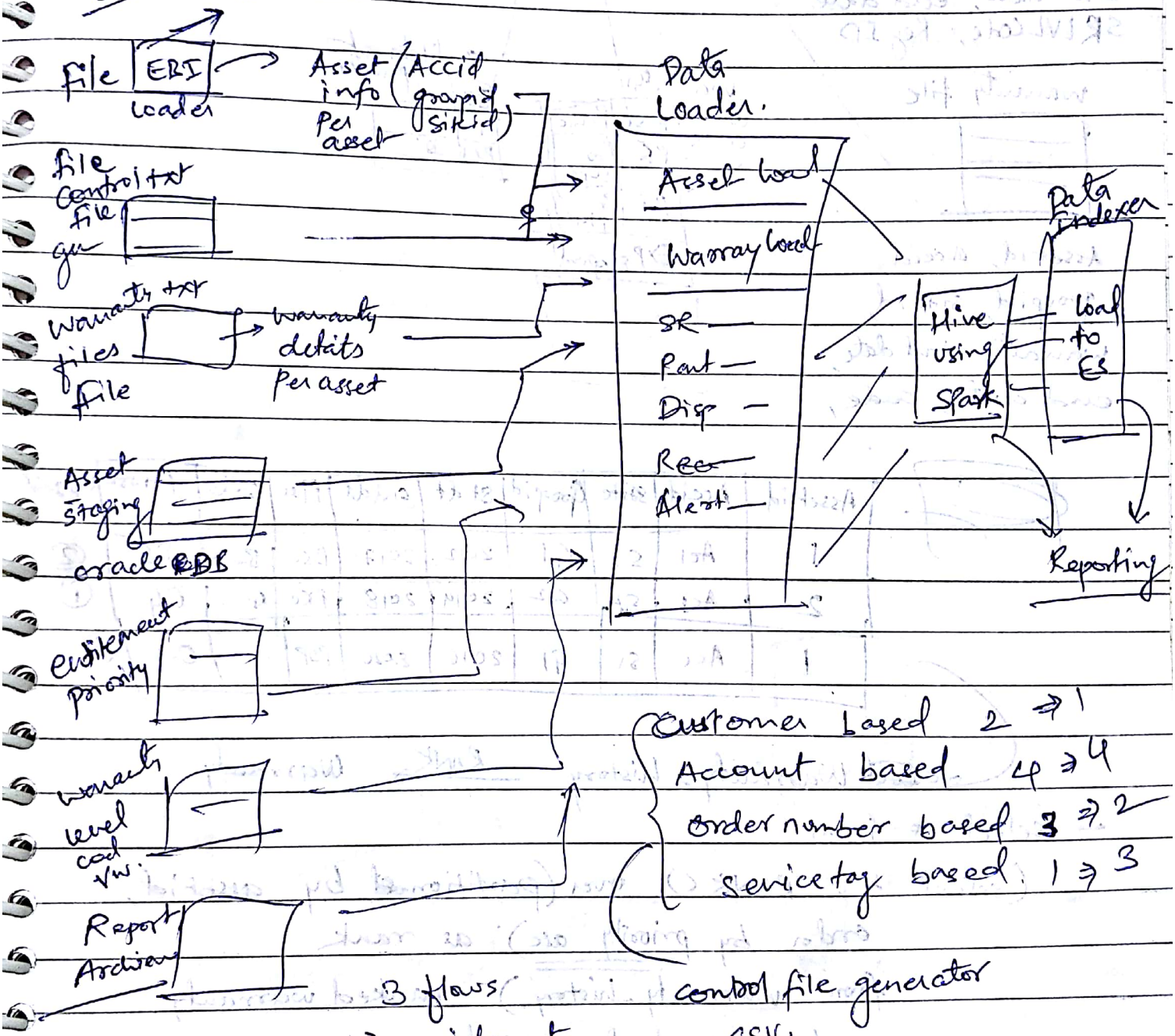
- optimization possible → Tungsten → off-heap as binary so serialization & garbage coll° avoided
  → Catalyst Optimizer
- control over structure of data
- files, JDBC, Hive, Parquet
- Dataset<Row> ⇒ Dataframes → limitation
  - Compile time safety
  - DF to domain object not possible
- Encoder ⇒ works like SerDe
  - Generate a bytecode which can interact with off-heap data.
  - no need to deserialize whole object
- Dataset ⇒ compile time safety.

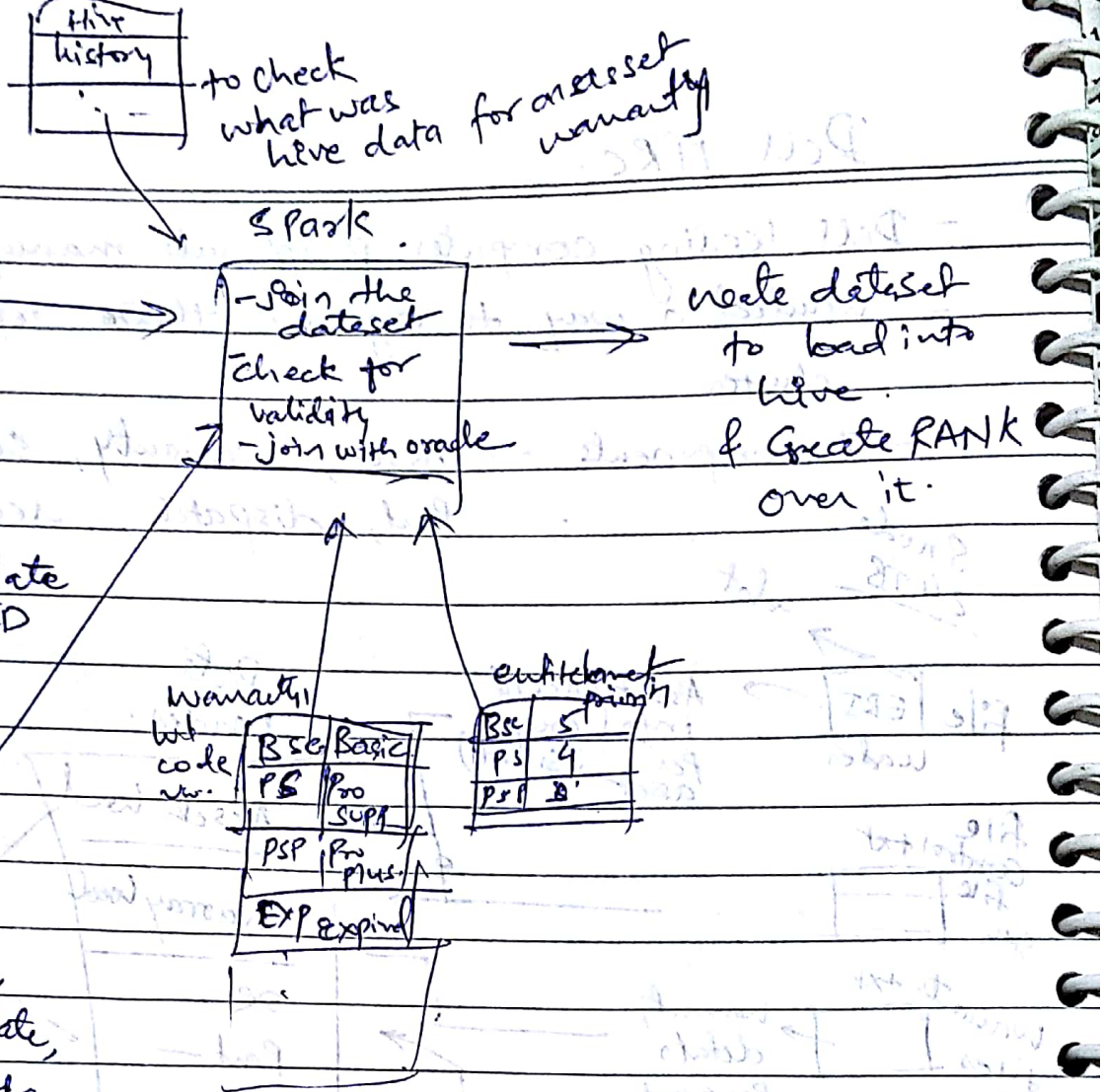
  Features ⇒ ① Optimization
  ② Compile time check
  ③ Persistence
  ④ convertible ⇒ toDF(), toDS()
  ⑤ fast ⑥ less memory
  ⑦ Sample API for Java & Scala
- RDD vs DF vs DS

⇒ RDD ⇒ RDD.toDF ⇒ define case class
gives DF or Dataset<Row>
DF.as[Employee] gives dataset[Employee]

Dell PPRC.

- Dell leading computer peripherals manufacturer
- wanted a way to organize their repository in hadoop cluster
- 7 components → Asset, warranty, Service Request, Part, dispatch, recomendation, alerts





Assetid	Accid	Siteid	Grpid	St dt	Endt	Plan	code	Priority	RANK
1	Ac1	S1	G1	2012	2017	Bsc	Bscxxx	05	②
2	Ac2	S2	G2	2014	2018	PKo	KP...	04	①
1	Ac1	S1	G1	2016	2020	PSP	---	03	①

~~Get~~ Warranty_history → RANK → Warranty.

```
← select * from
    (select *, RANK() over (partitioned by assetid,
    order by priority asc) as rank
    from warranty_history) ranked_warranty
    where ranked_warranty.rank = 1;
```

⇒

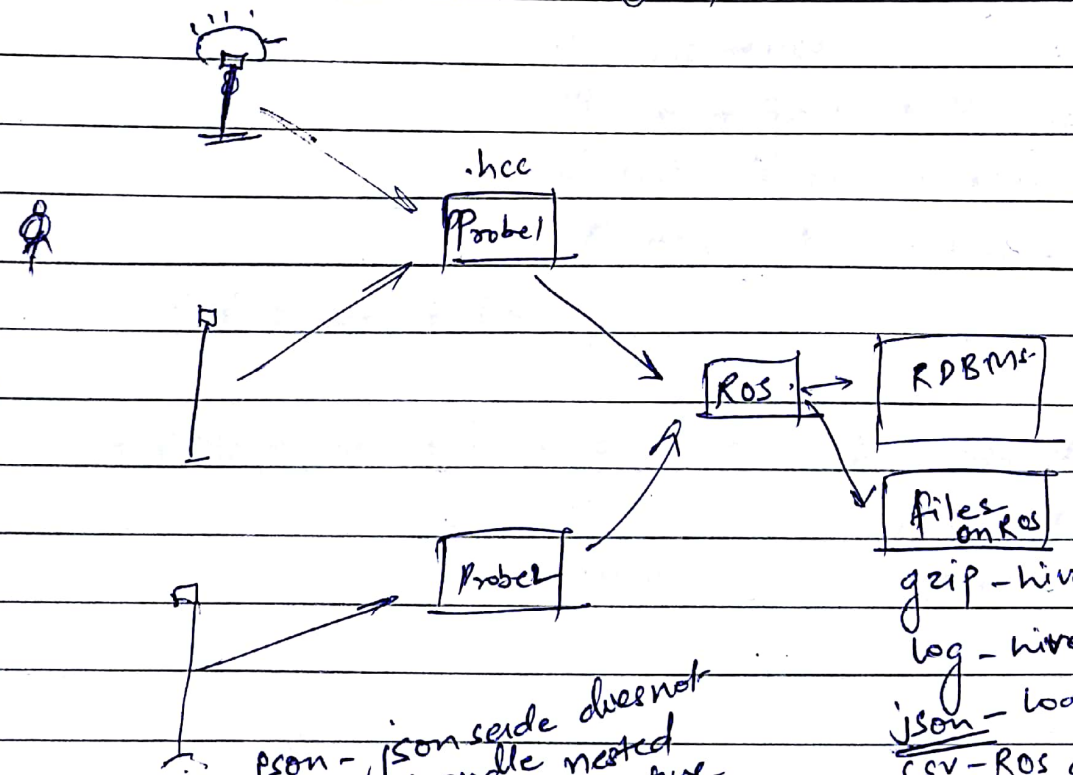
Priority asc → priority Asc, Stdt DESC, EndDT DESC.

Empirix Intellisight & Holistix.

Empirix provides carrier grade products & services to major network providers. Their clients ~~are~~ were vodafone, O2, Tuk. So this project was to design an initial migration project of the application from tradition RDBMS to hadoop infra.

- Tasks :- identified what all sources for data.
  - what all types of input data.) probe, ROS, RDBMS.
- Steps - shell scripts to read them & check its validity → HDFS
  - Hive scripts to load them into hive & process.
  - RDBMS → sqoop scripts → Hive tables
  - Java module to create ladder diagram out of this data.
  - coon job → nightly

6 nodes 5 TB



gzip - hive directly reads it
log - hive
json - load to hive.
csv - ROS generated reports
hcc - call data - shell converts them into csv.
tarballs - directly load on hdfs.

json - json serde does not handle nested structure well so we use LATERAL VIEW to achieve this

guide wire policy center

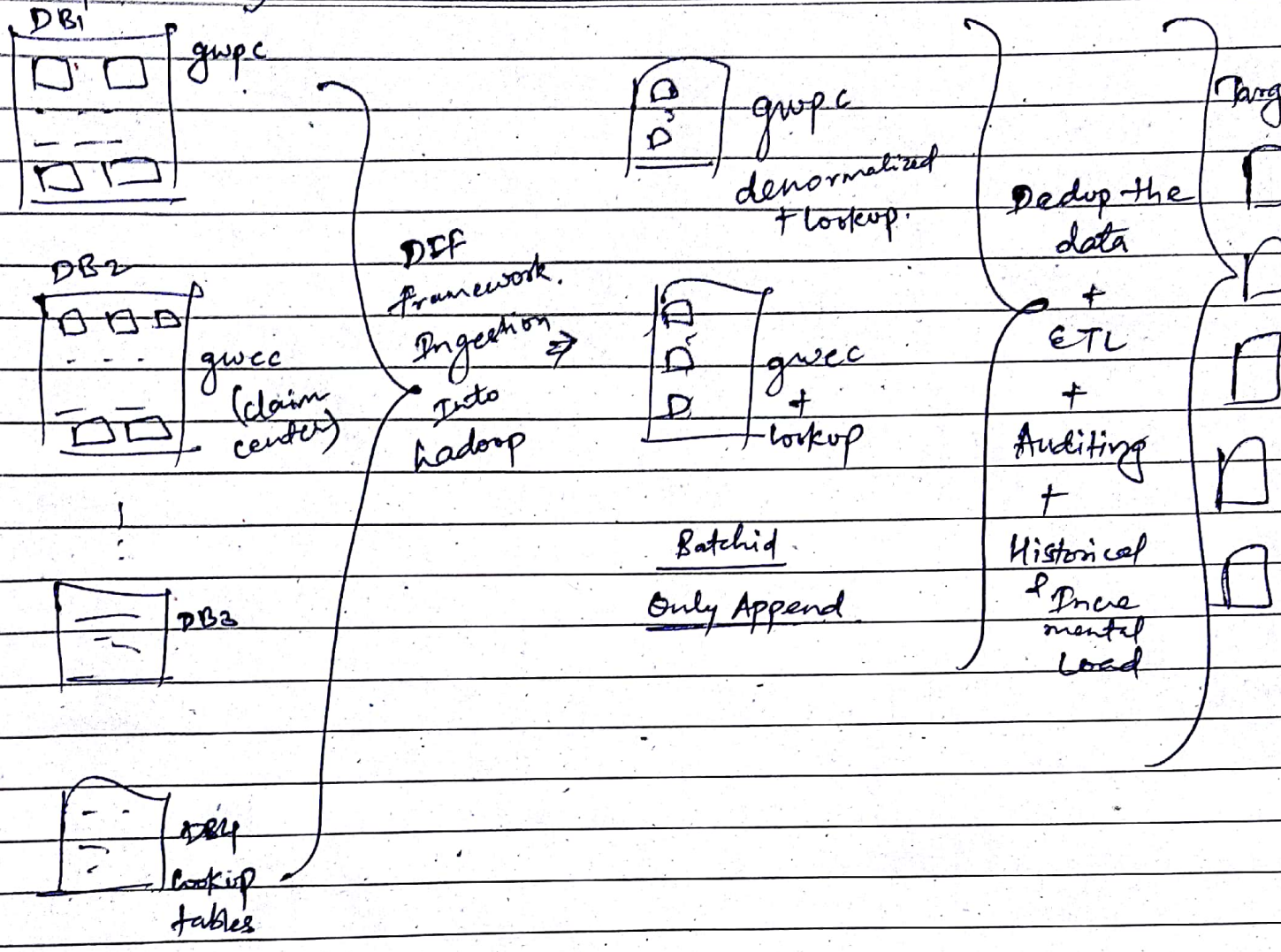
Guidewire

SA Ingestion

Denormalized form DATE: ☐☐ ☐☐ ☐☐

SA Curation

ETL



DATE: ☐☐☐☐☐☐

Challenges

① – Stateauto – orc file reading from spark vs reading from hive
– Spark printed column values as Null due to orc version was older
Action Solution – we installed newer orc version packages to spark Libraries.

②

② Spark Jobs was taking too long,
Solution – Performance enhancements using broadcast join, better spark config. & StreamTable hints in Spark SQL
GroupbyKey removed by reducebykey

③ OOM errors – removed collects wherever possible
– Also used mapPartitionsWithIndex & then collected at partition level & saved data

④ Spark version mismatch in dev vs UAT environment.
– This caused the job to fail due to difference in Lineage (DAG) created by job
– Made UAT cluster on Spark version as Dev

⑤ OOM – due to improper join sequences (Largest table last)

⑥ Architecture challenge – Most cost on cluster, but ~~job~~ ETL time was 6 hours –
Solution – Suggested dataproc cluster transient one & a small permanant ~~hive~~ dataproc cluster for hive

⑦ Difficult to debug the code – Solution :- we added isDebugMode flag & printed 20 records for better debugging.

DATE: ☐☐☐☐☐☐

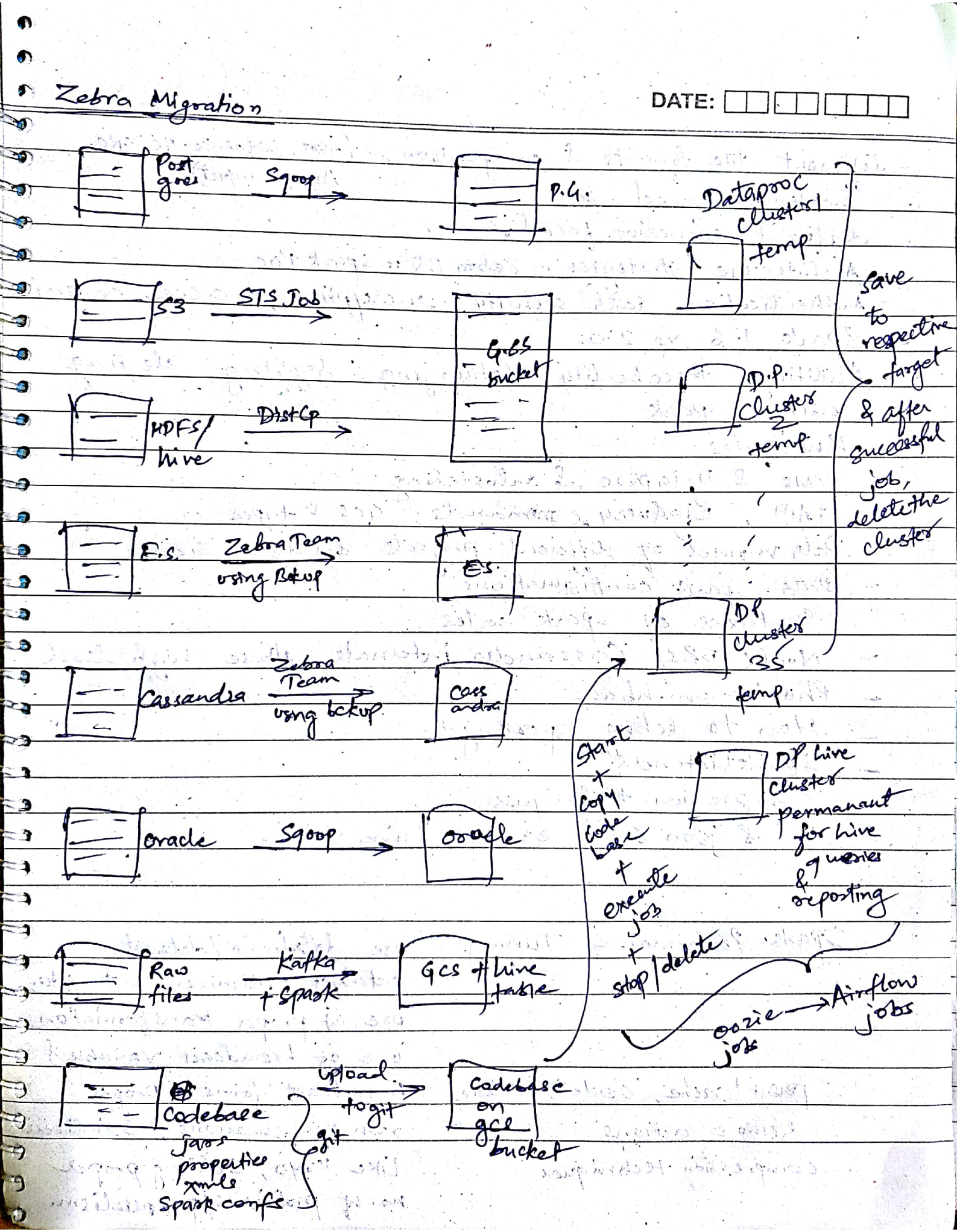
- different file formats & comparison ✓ (text, sequence, RC, ORC, Avro, Parquet)
- challenges faced ✓
- different scenarios faced ✓
- Architecture strategies – Zebra BQ vs spark Poc
- Authentication, data security, encryption, access control
- Spark 1.6 vs 2.0
- Auditing, traceability, debugging, logging, alerting using spark
- Kubernetes
- VMs & Dataproc & autoscaling
- IAM, BigQuery commands, GCS & types
- Data volumes of different projects & cluster sizes.
- ~~Data~~ Spark configurations
- Syntaxes of spark codes
- NoSQL DBs, Cassandra internals, Hbase high level
- Phoenix on hbase
- How to debug spark job.
- SparkListeners.
- Compression techniques
- xml & json serde example hive

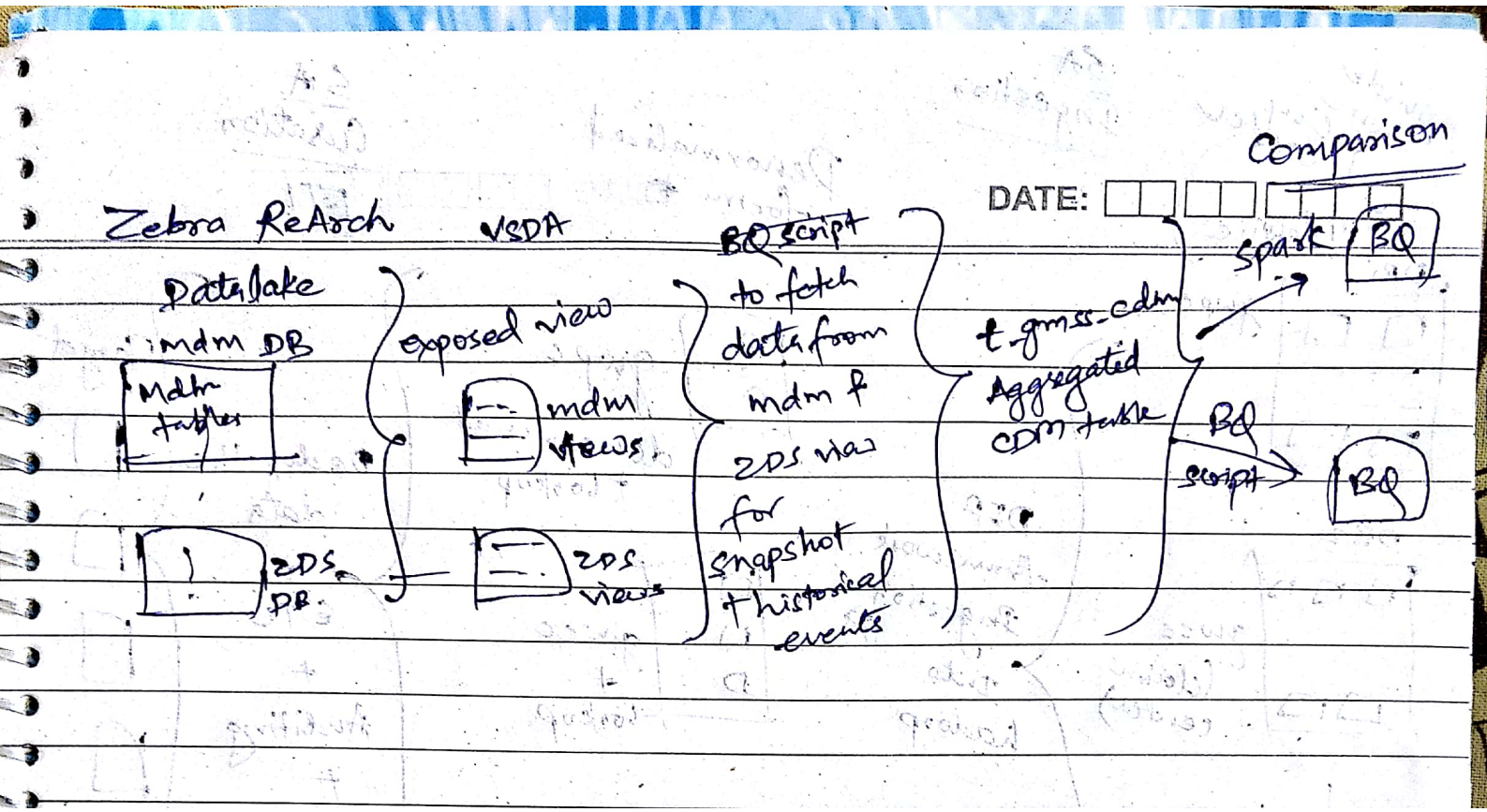
Spark Performance Tuning – use dataframe/dataset, catalyst optimizer & tungsten, use of proper transformation, use of broadcast variable & broadcast joins, proper run configurations, Serializers like Kryo, Setting proper no. of partitions/parallelism

- persist/cache, code modularize
- Remove actions
- compression techniques

Zebra Migration

DATE: ☐☐ ☐☐ ☐☐☐☐





Hadoop vs Dataproc :- Support by google, one place of everything, can be shutdown, GS + dataproc combination

Dataflow vs Spark

① Resource mgmt – Dataflow allocates on demand
② Interactive shell – Spark has this
③ Streaming feature – dataflow better
④ Complexity of coding - dataflow for simpler transform

GS, Dataproc, BigQuery, Cloud sql, Cloud-Spanner, Cloud BigTable, Cloud datastore, Dataflow, Dataprep, Stackdriver, Kubernetes, Compute Engine, App Engine, Pub-Sub, Interconnect, STS, Load balancing, Cloud Functions (trigger based), Google cloud storage, types & Billing, BQ billing,