Accelerating Functional Verification Coverage Data Manipulation Using Map Reduce

Eman El Mandouh, Mentor Graphics Siemens Business

A. Gamal, A. Khaled, T. Ibrahim, Cairo University

B. Amr G. Wassal, Elsayed Hemayed , Cairo University

Mentor® A Siemens Business

Cairo University

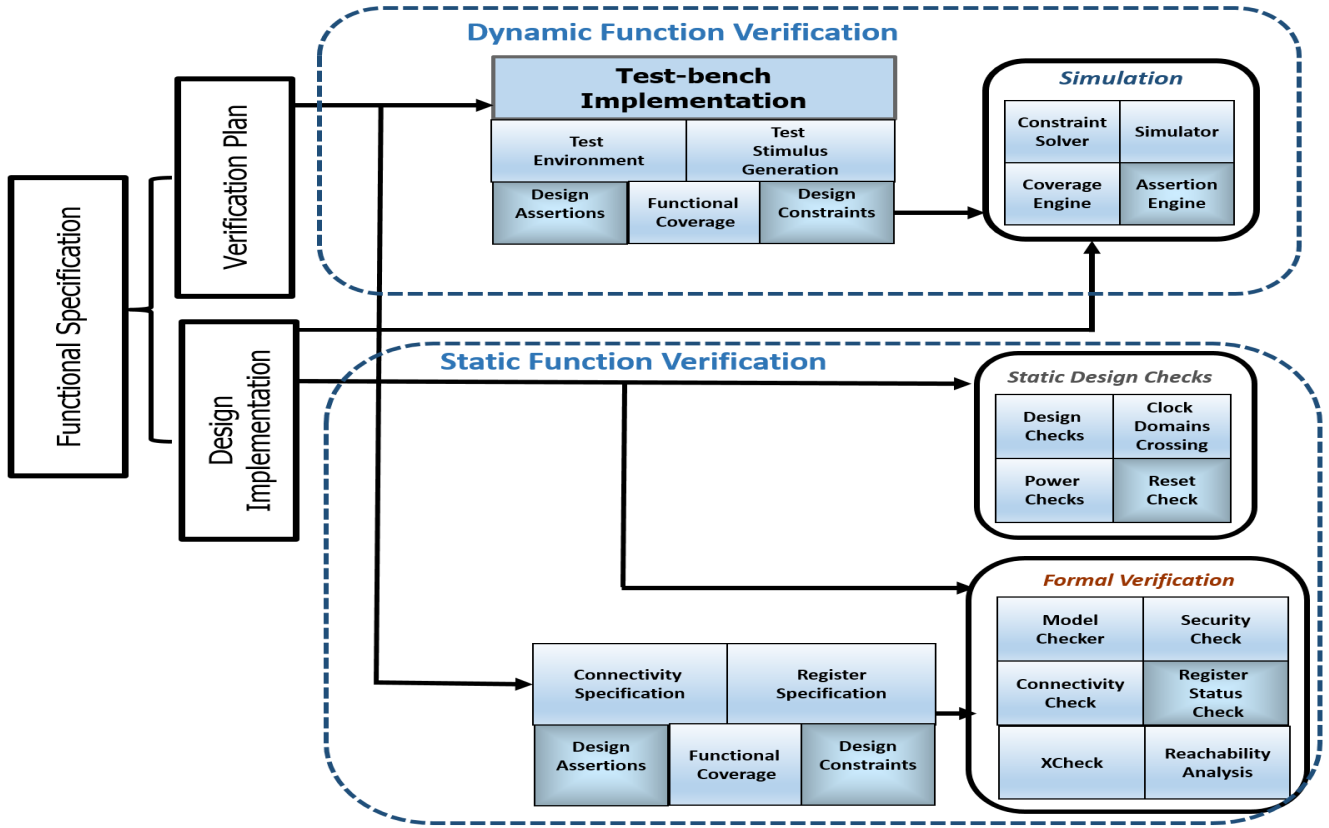
accellera SYSTEMS INITIATIVE

2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

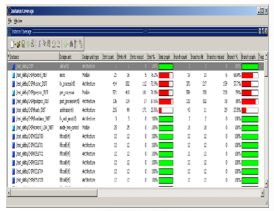
Functional Verification : An Overview

Hardware Verification is the process of checking if a design conforms to its specification of functionality, timing, testability, and power dissipation

The Primary Goal of Functional Verification (FV) is to establish confidence that the design intent was captured correctly by the implementation.



Coverage in Functional Verification



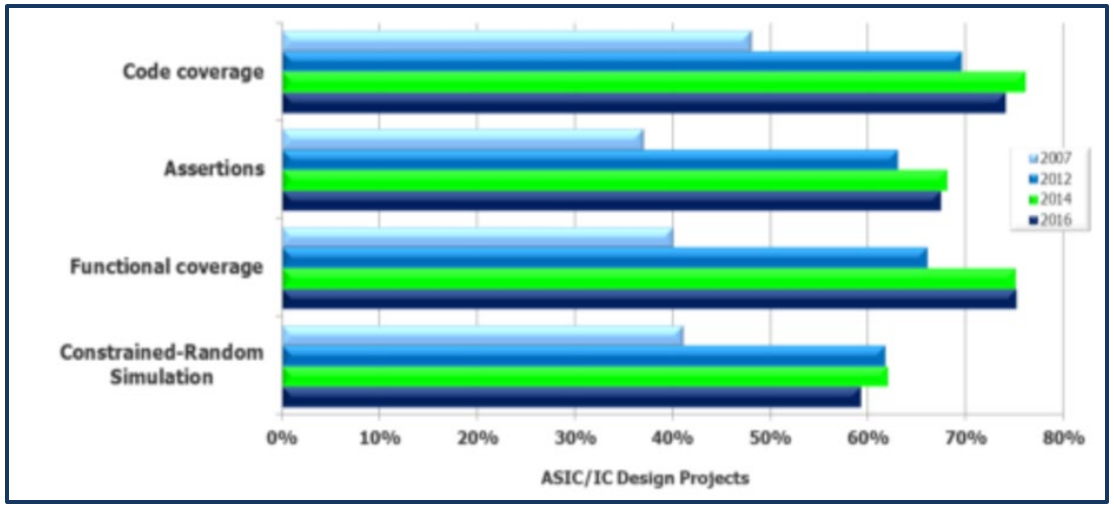
Coverage in Functional Verification

- ✓ **Is to measure the completeness of functional verification activities by judging the final coverage achieved from the tests execution**

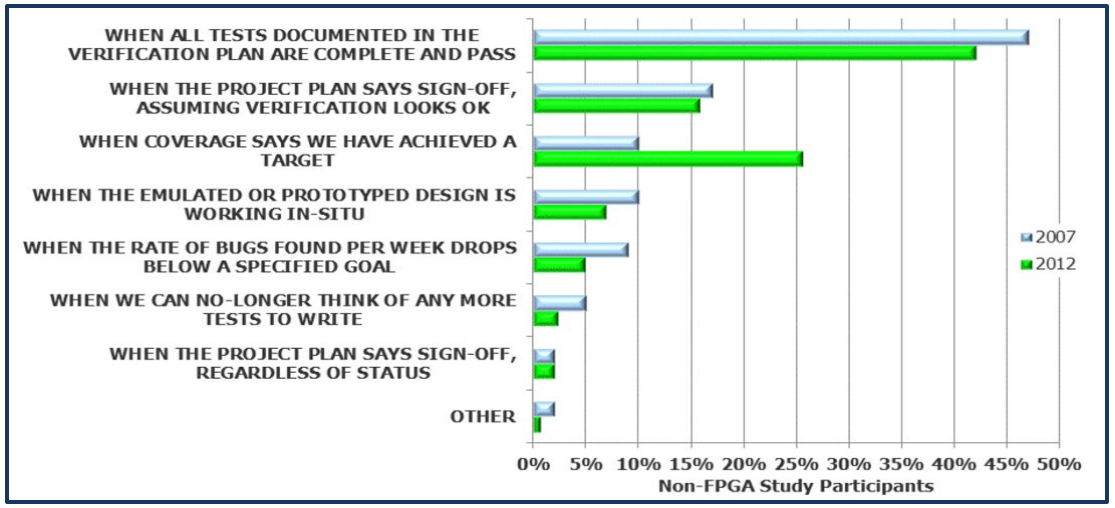
- Coverage Type
  - Code Coverage : Simplest and Easiest to gather
    - Statement, Branch, Toggle, Expression, FSM,..
  - Functional Coverage
    - Cover Directive, Cover Points

Coverage in Functional Verification

Dynamic Verification Adoption Trends

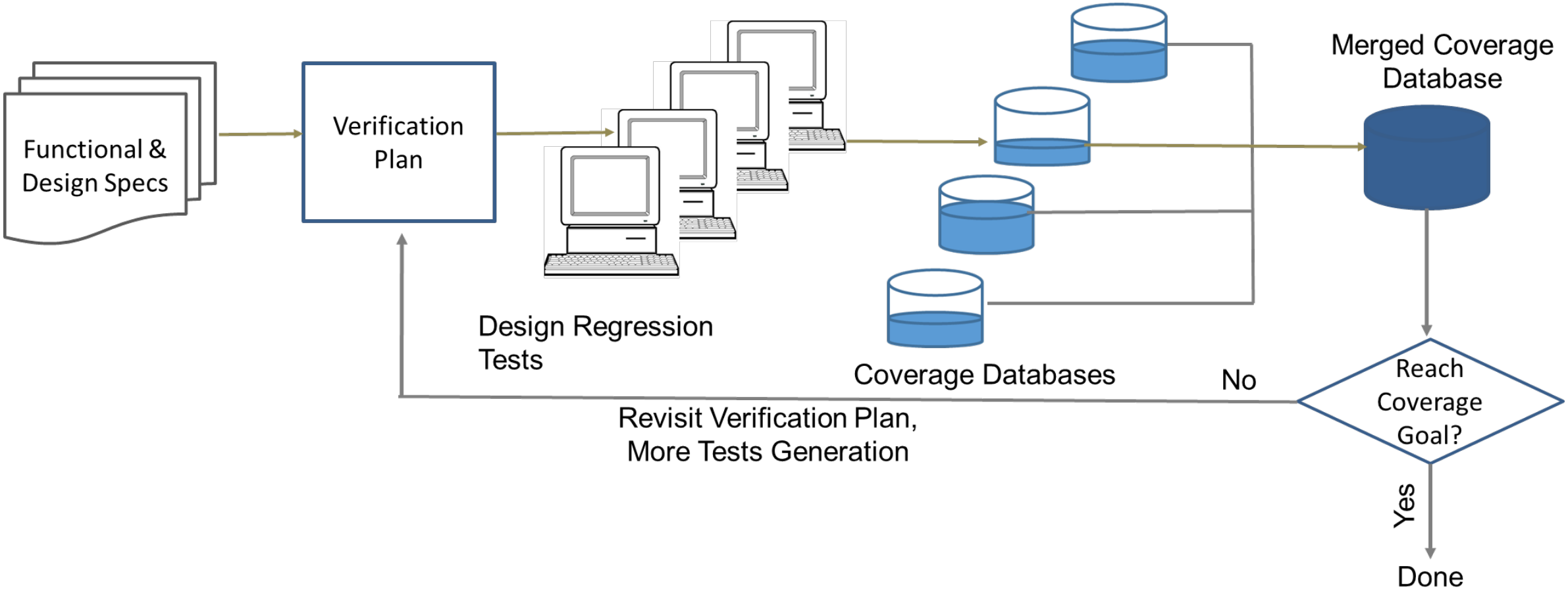


Sign Off Criteria Trends

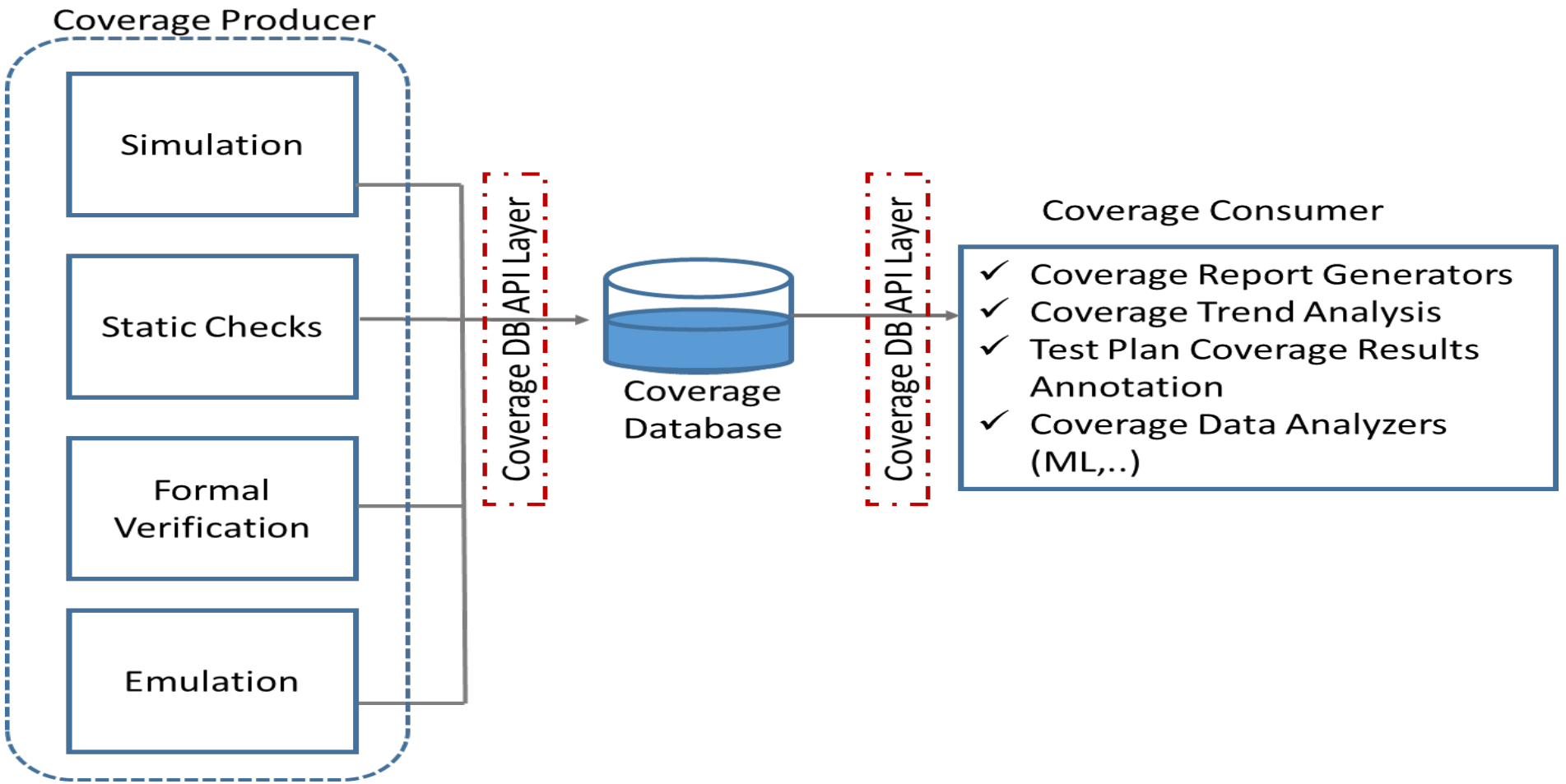


** Source Wilson Research Group Functional Verification Study

Coverage Driven Verification

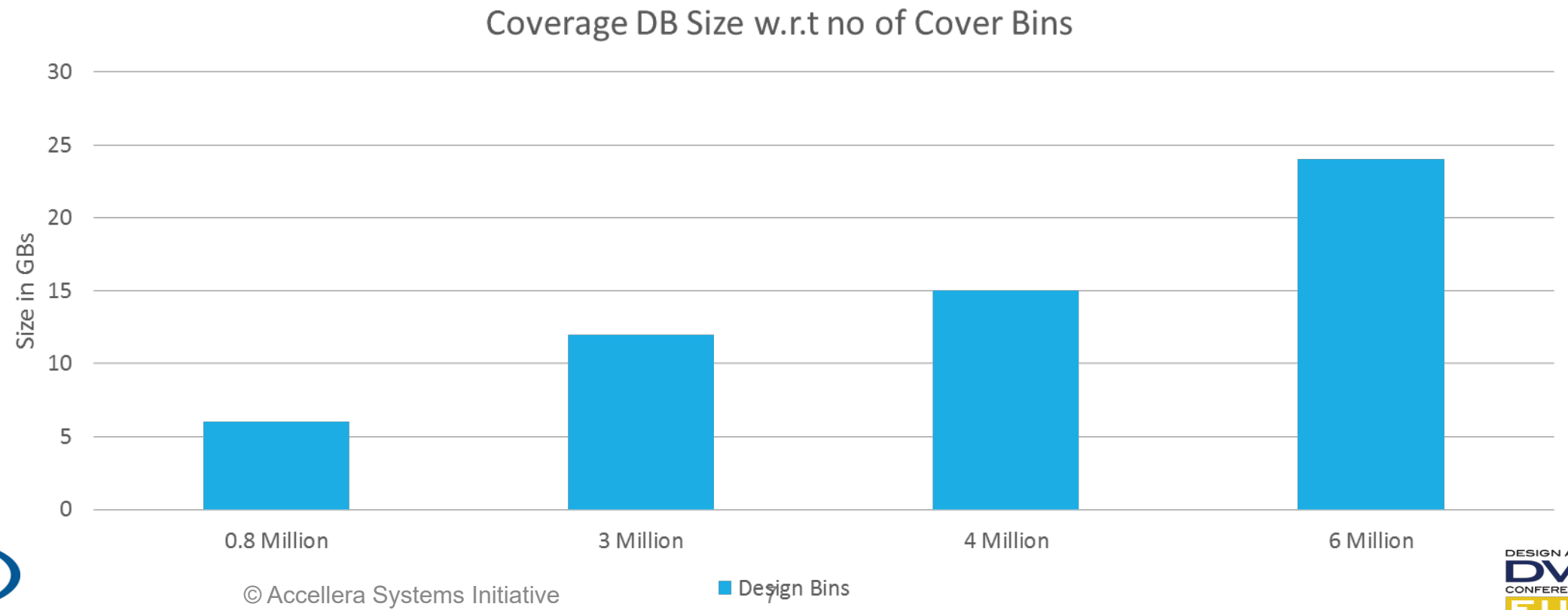


Universal Coverage Data Bases



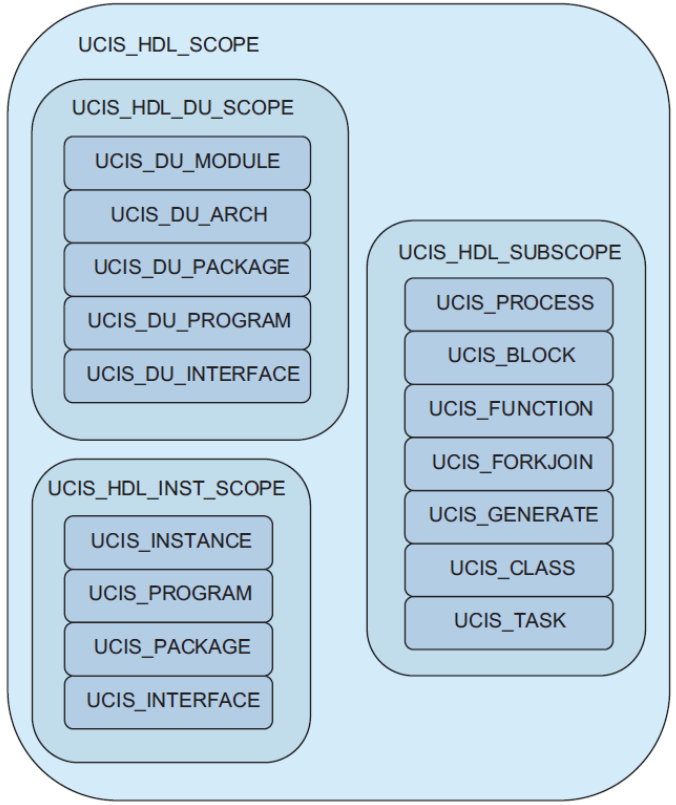
Challenges in Analyzing Coverage Data

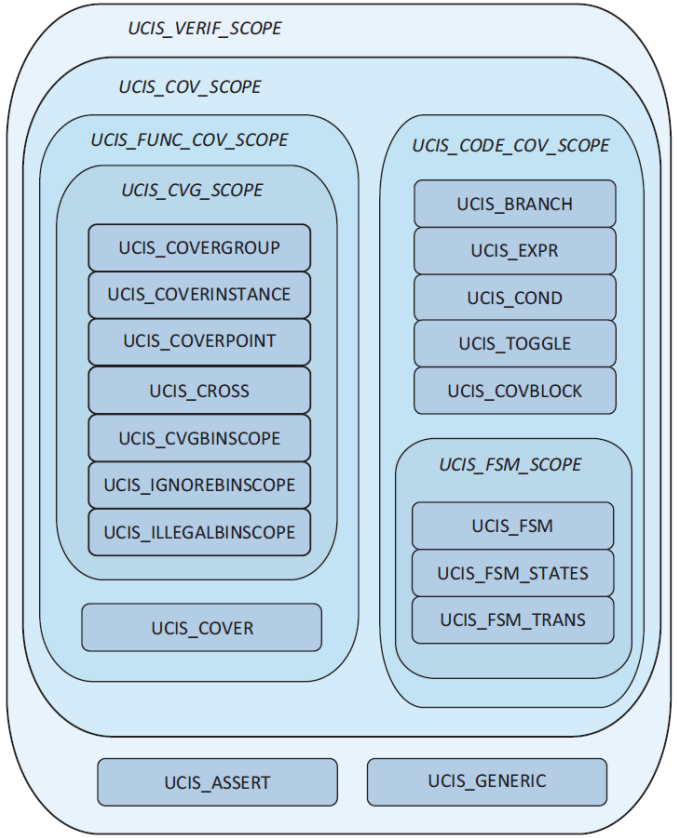
- Complex, Large HW Designs, Large Size of Coverage Data
- Multiple Coverage DBs from Different Verification Methods



Introduction to Coverage DB Data Model

- HDL scope data models
- Functional coverage data models
- Code coverage data models

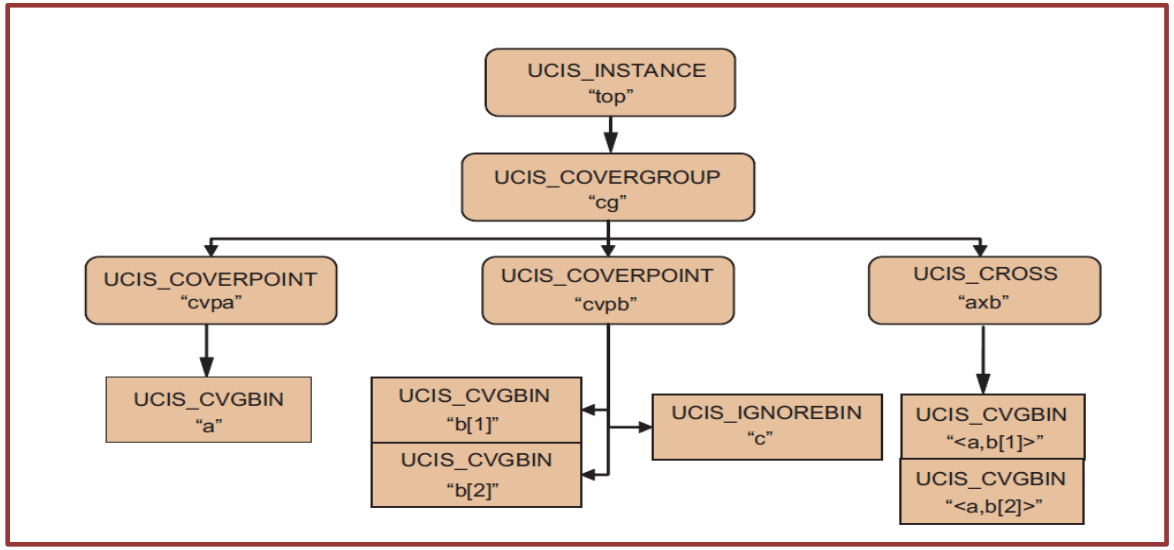




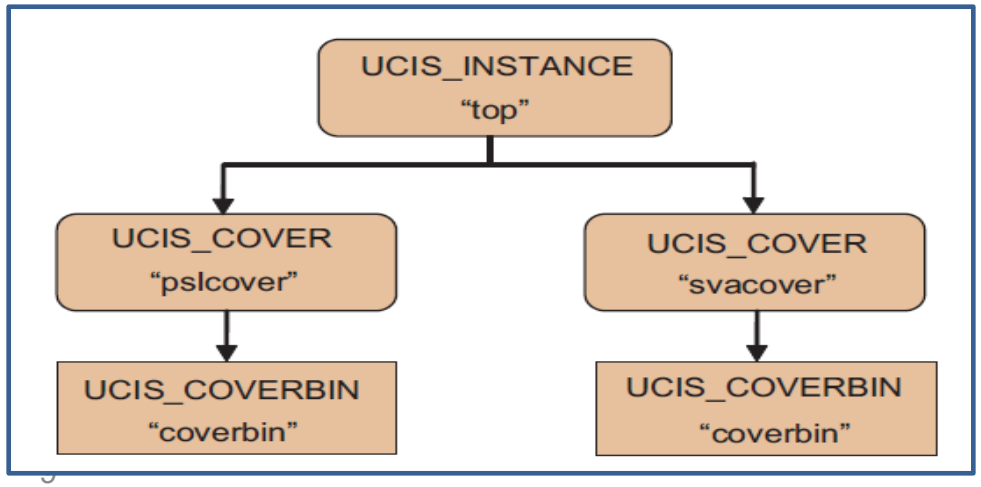
** Source Unified Coverage Interoperability Standard (UCIS) V1.0

Introduction to Coverage DB Data Model

```
covergroup cg;
 type_option.comment = Example;
 type_option.merge_instances = 1;
 option.at_least = 2; // becomes the 'goal' in the coveritems
 cvpa: coverpoint a {bins a = {0}; }
 cvpb: coverpoint b {bins b[] = {1,2}; ignore_bins c = {3}; }
 axb: cross cvpa, cvpb {type_option.weight = 2; }
endgroup
cg cv = new();
```




```
 // psl default clock = rose(clk);     // line 10
 // psl pslcover: cover {b;a};         // line 11
 sequence a_after_b;                   // line 12
  @ (posedge clk) b ##1 a;             // line 13
 endsequence                           // line 14
 svacover: cover property(a_after_b); // line 15
```



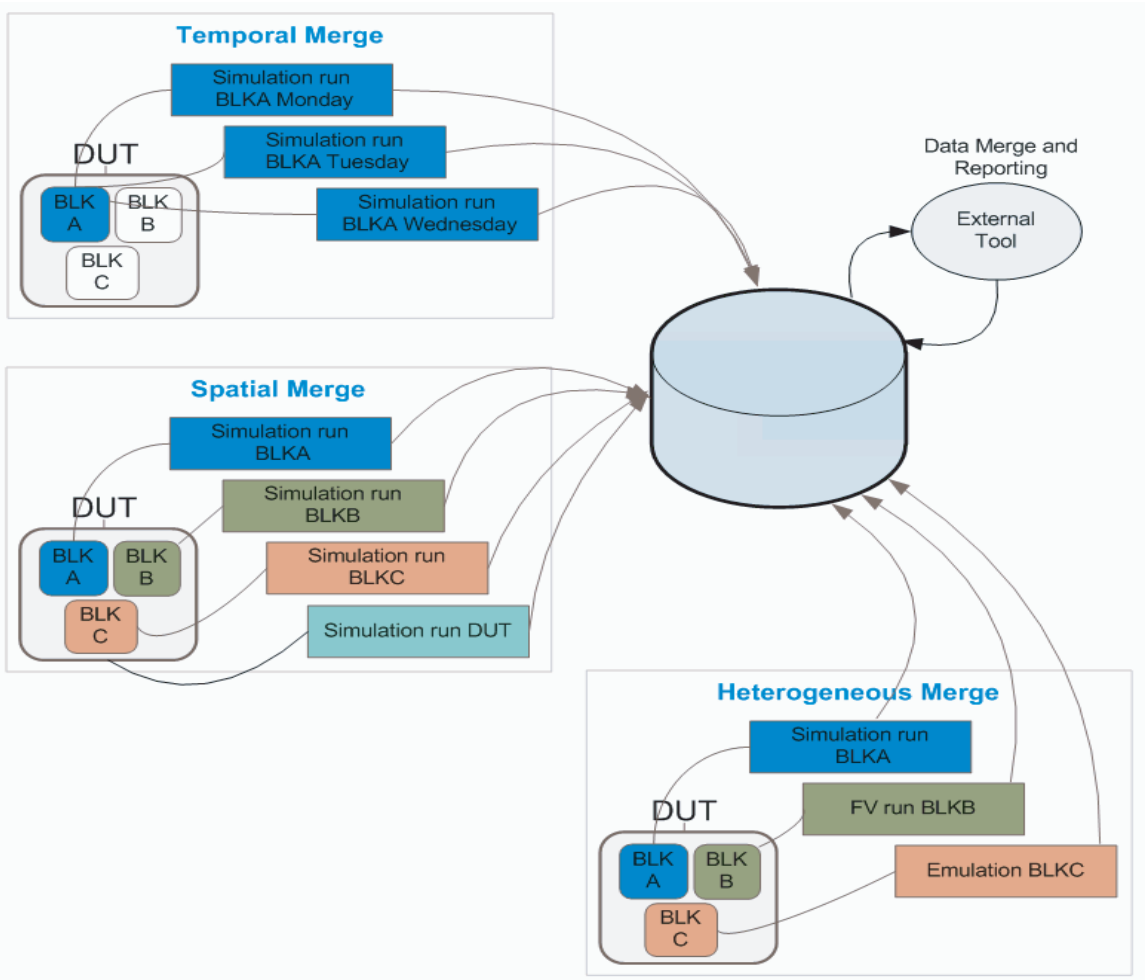
** Source Unified Coverage Interoperability Standard (UCIS) V1.0

Cover Object Data: An Overview

- **Coverage Database Handle**
- **Parent Scope**
- **Scope Name**
- **Scope Type**
- **File Info**
- **Design Unit Scope**
- **Count**
- **Attributes**
- **Flags**

Coverage Merging Algorithms

- Merge Use Cases
  - Temporal Merge
  - Spatial Merge
  - Heterogeneous Merge
- Merge Algorithm
  - Total Merge
  - Test Association Merge



Hadoop

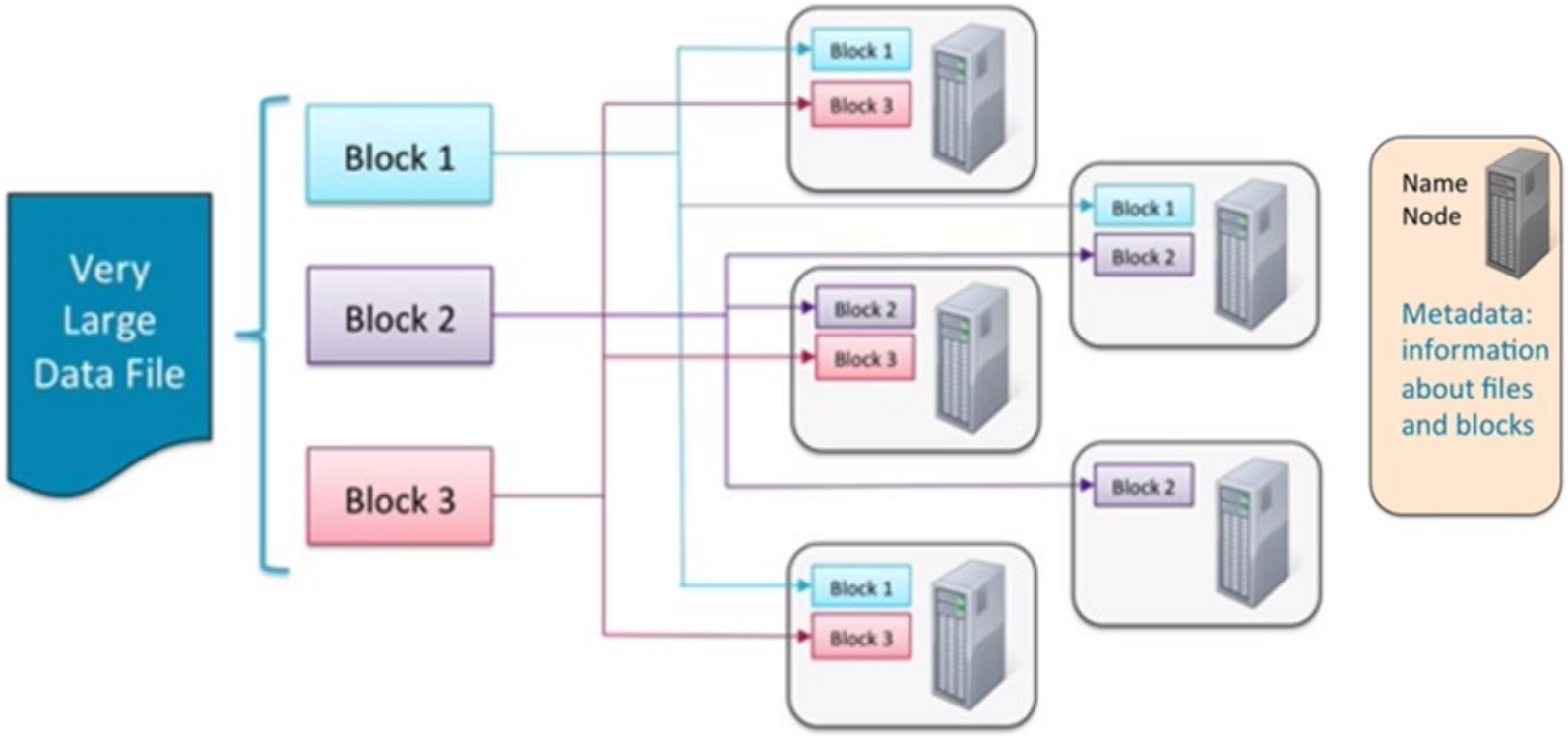
- It is an open-source software framework used for distributed storage and processing of dataset of big data
- Hadoop used components:
  - **Hadoop Distributed File System (HDFS)** – the Java-based scalable system that stores data across multiple machines
  - **YARN** – (Yet Another Resource Negotiator) provides resource management for the processes running on Hadoop.
  - **MapReduce** – a parallel processing software framework. It is comprised of two steps. Map step is a master node that takes inputs and partitions them into smaller subproblems and then distributes them to worker nodes.

Hadoop Distributed File System : HDFS

- HDFS is a file system written in Java
- Sits on the top of native file system
- Storage of massive amount of data across clusters
  - Scalable
  - Fault Tolerant
  - Designed specifically to Support Efficient Processing with MapReduce
- Command Line Interface or Java API

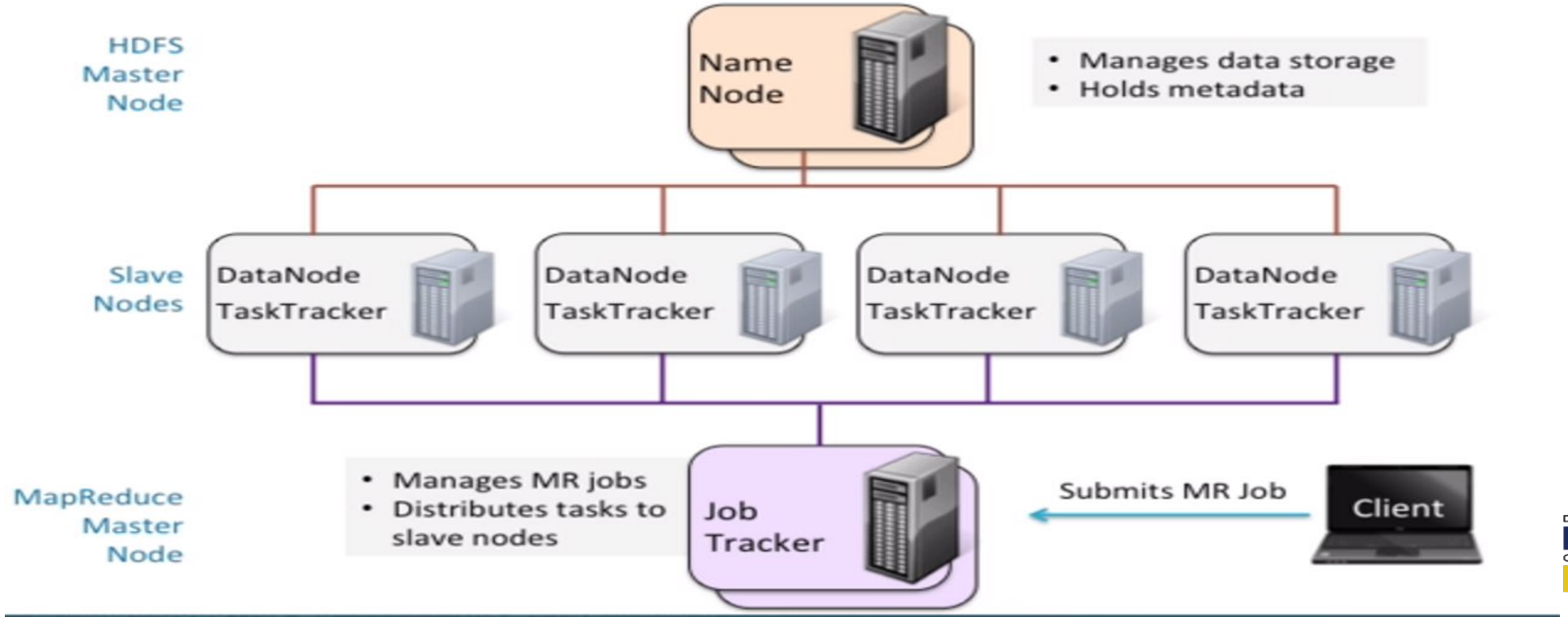
How Files are Stored in HDFS?

- Large Data Files are Split into Blocks (64MB, 128MB)
- File Blocks are Distributed to Data Nodes
- Each Block is Replicated on Multiple Nodes (Default 3x), Fault Tolerant
- Name Node (Master Node) stores metadata about Data Nodes, files, blocks



Anatomy of Hadoop Cluster

- Hadoop Cluster is group of machines/nodes work together to processes map-reduce jobs.
- Each Cluster has
  - Group of ***Data-Nodes (Slaves)*** , Holds Data Blocks of HDFS File and Processes Map/Reduce actual Tasks (Task Tracker)
  - HDFS Master Node ***Name Node***, keeps metadata about cluster data nodes, files, blocks
  - MapReduce Master Node ***Job Tracker*** manage MapReduce tasks, identify individual tasks in MapReduce job, running them across slave nodes.
  - Each cluster should maintain ***(active-standby)*** version of Master Data Node and Master Job Trackers



Hadoop MapReduce

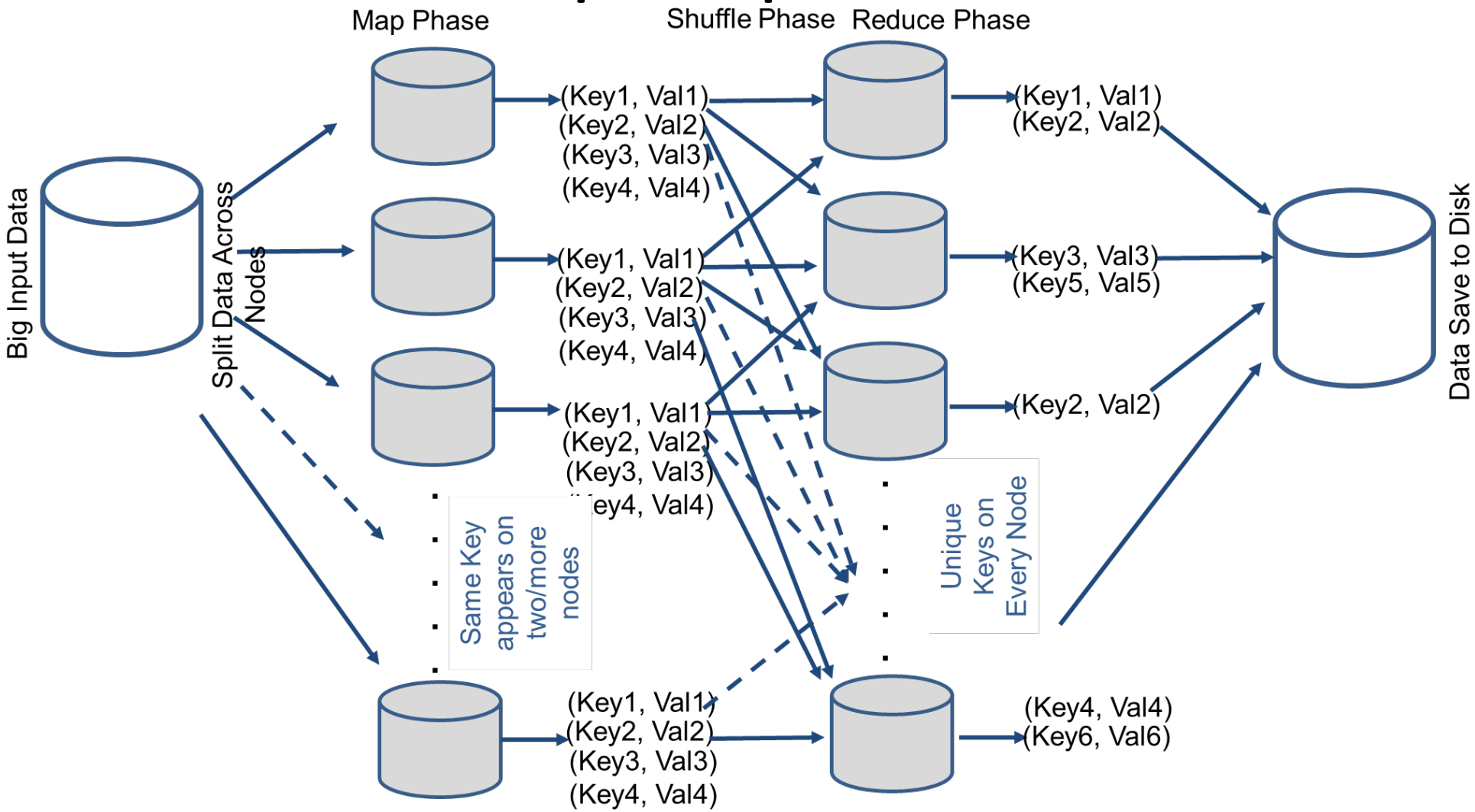
How Data is Possessed within Hadoop?

- ***Mapper***
  - Maps are the individual tasks that transform input records into intermediate records
  - Each Map Task Typically Operates on single HDFS Block
  - Map Tasks run usually on nodes where data block is stored

- ***Sort and Shuffle***
  - Hadoop does it automatically to run map tasks across resulted data from mapper.
  - i.e sort and consolidate intermediate data from all mappers and before reduce tasks start.

- ***Reducer***
  - Operates on sorted/shuffled intermediate data
  - Do Final Processing to collect all results and produce final output.

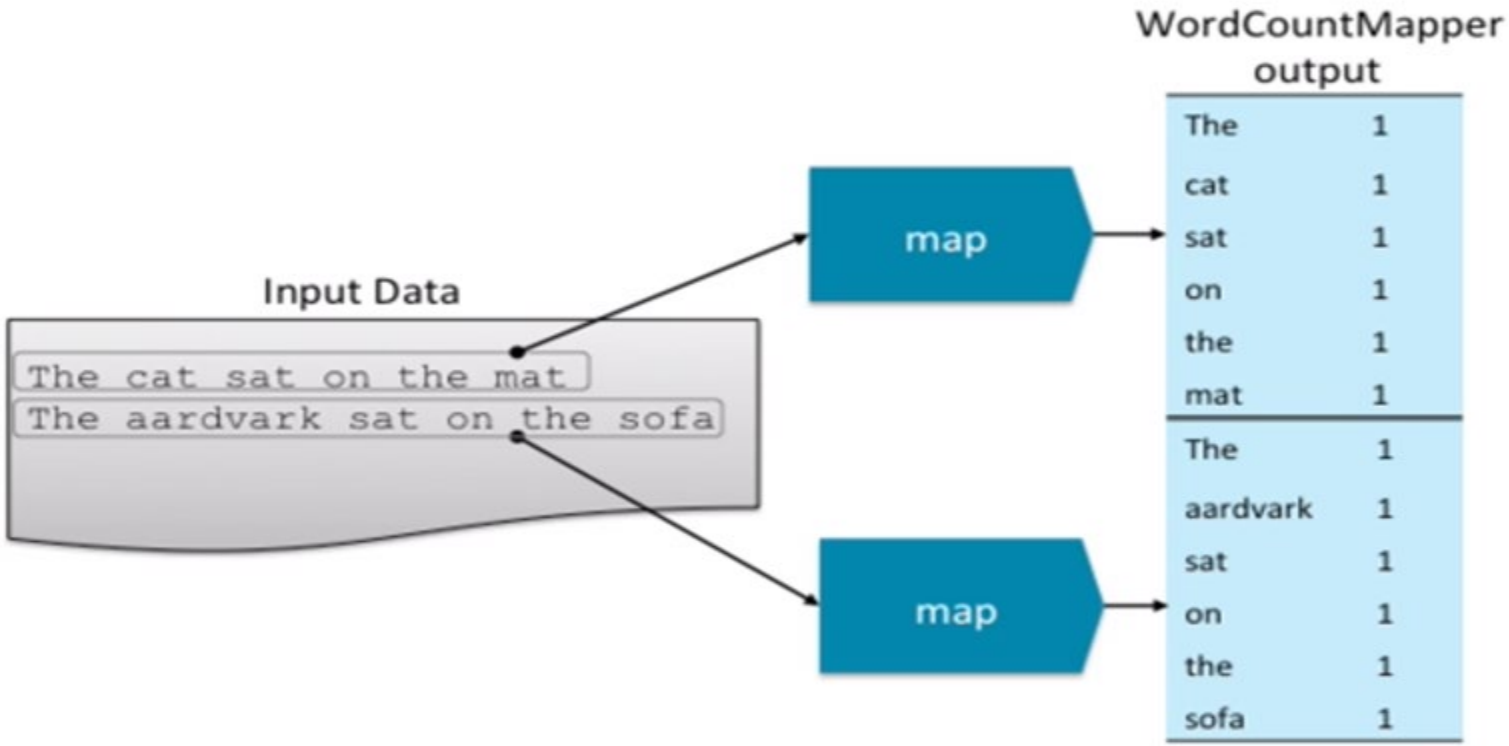
Hadoop Map Reduce



MapReduce Example :

Word Occurrence Count in Log file

- Start by writing map function takes each line do some function, count # word occurrence



MapReduce Example :

Word Occurrence Count in Log file

- Sort/Shuffle: Hadoop takes o/p of Mapper, long list of words with single occurrence, not adding them but sort and consolidate them across all Mappers output.

Mapper Output

The	1
cat	1
sat	1
on	1
the	1
mat	1
The	1
aardvark	1
sat	1
on	1
the	1
sofa	1

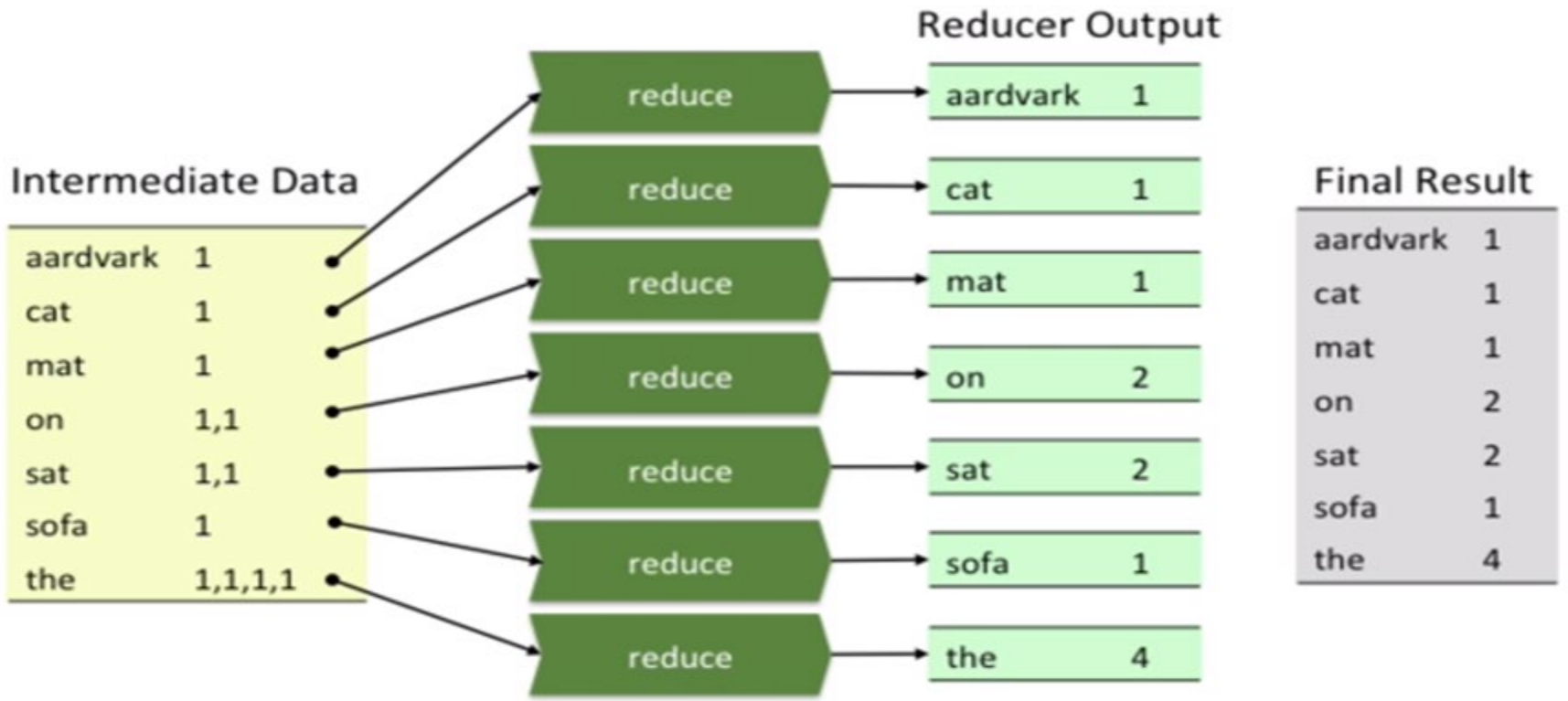
Intermediate Data

aardvark	1
cat	1,1
mat	1
on	1,1
sat	1,1
sofa	1
the	1,1,1,1

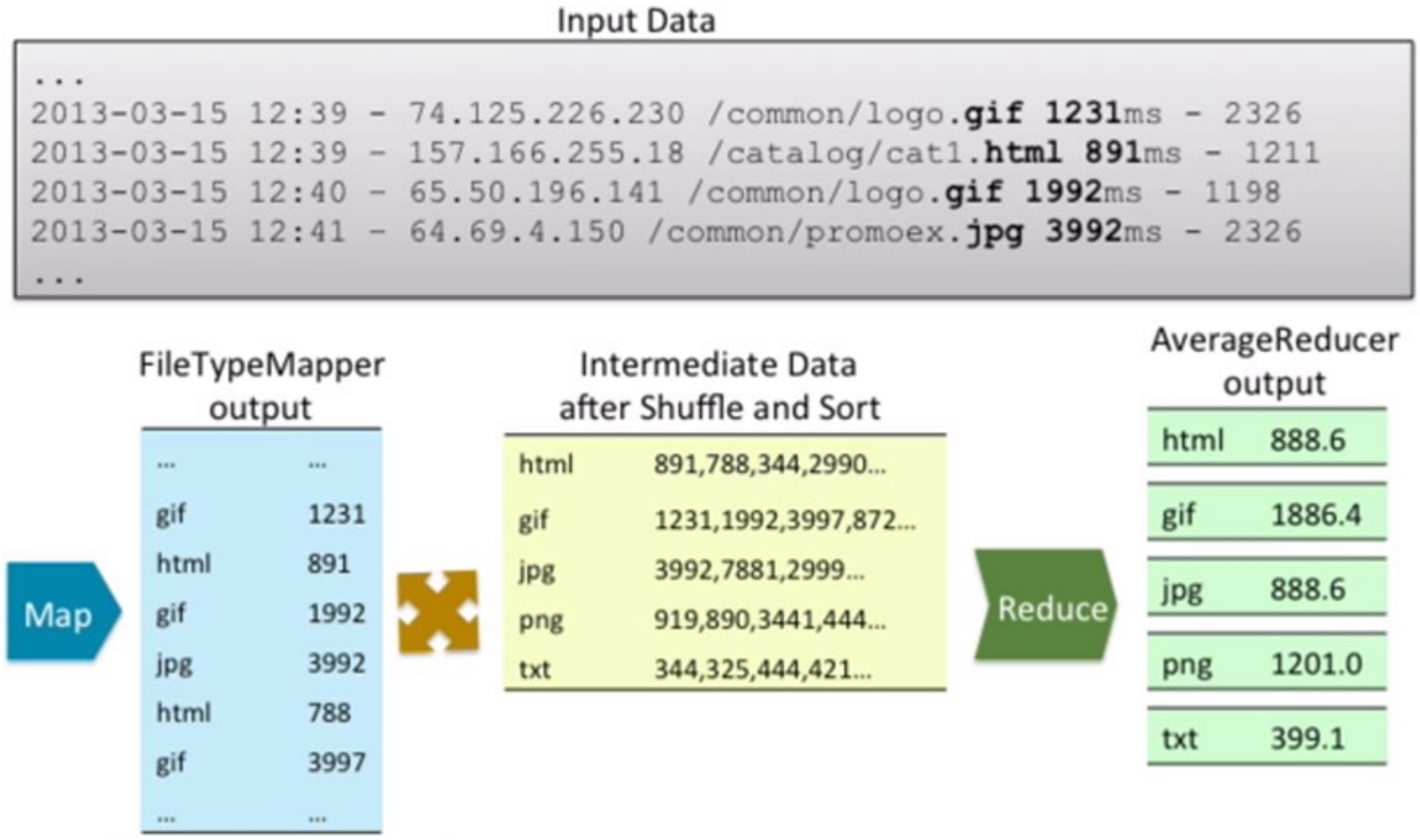
MapReduce Example :

Word Occurrence Count in Log file

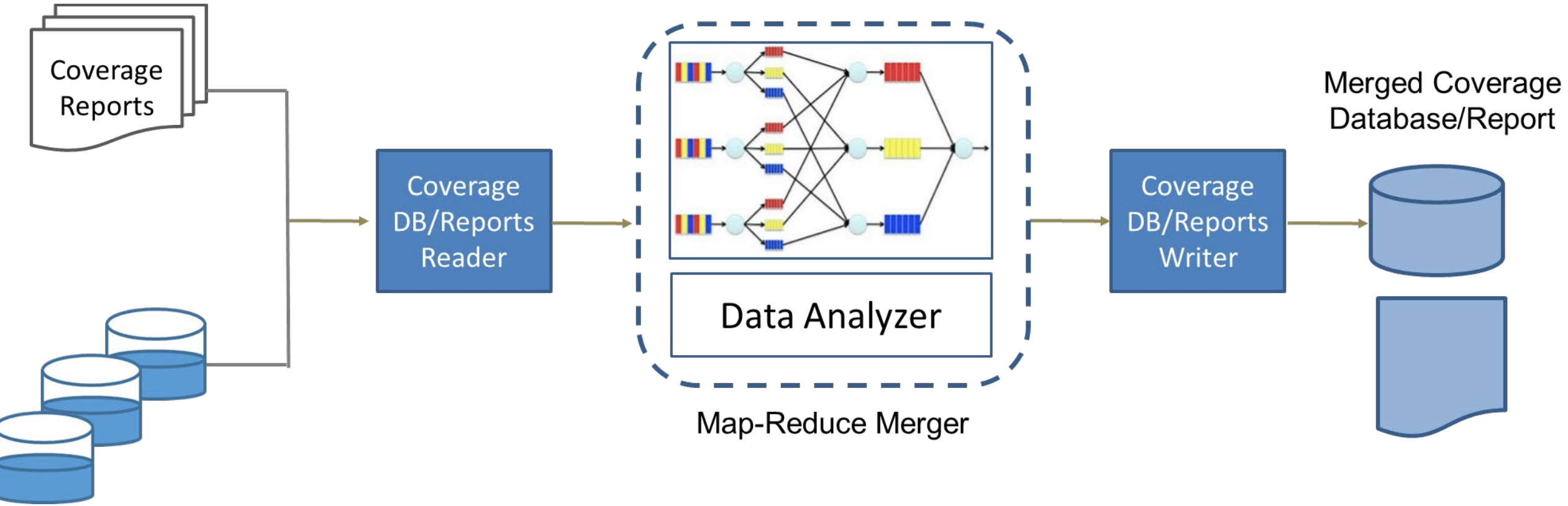
- Reducers : run in parallel by performing reduce function (sum reduce, avg, min, max,..) on the intermediate data resulted from sort and shuffle step.



MapReduce : Example2 Analyzing Logfiles



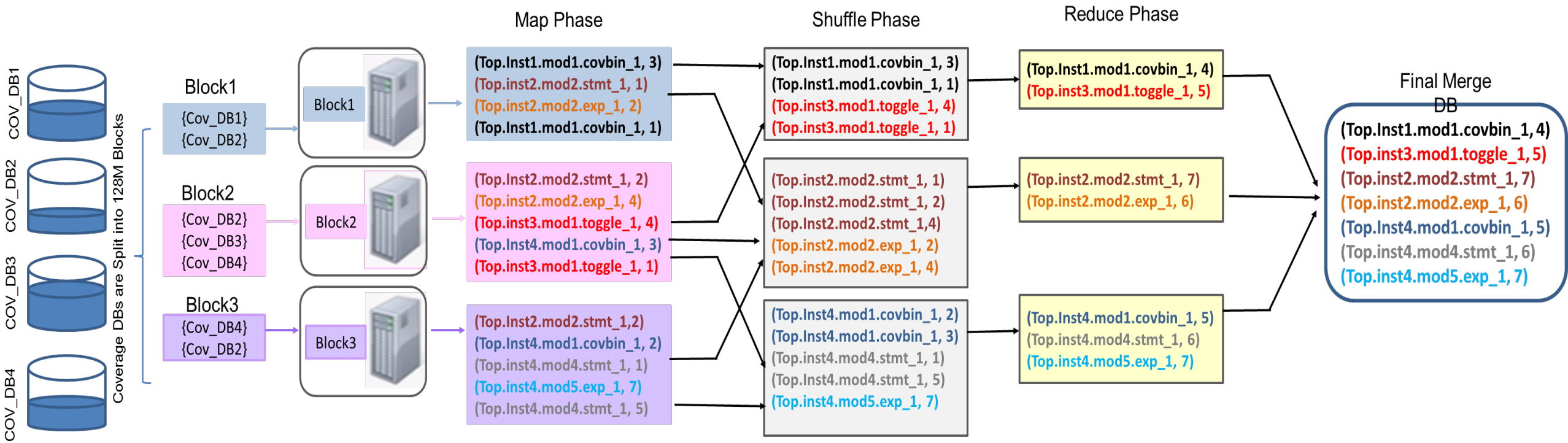
Coverage Data Merging Using Map Reduce



Coverage DB Reader

- **Read-streaming model versus in-memory model**
- Read-streaming mode allows limited-access to a narrow win the coverage database as it is traversed.
- Read-Streaming DB access mode has been selected because aligned with the need to handle DBs chunks by Hadoop Distributed File System that splits the file into 128 MB blocks.
- Read-streaming mode doesn't have a big memory footprint and it can be used to stream the data to the mappers easily with some modifications on its source code to be compatible with HDFS.
- Additionally, all merging related data can be accessed in this mode

Coverage Data Merging Proposed Algorithm



Coverage Data Merging Proposed Algorithm

Algorithm 1. Coverage-Data Merging Map-Function

Inputs: *Cover-Item Record. Cov_Rec*
Outputs: *(Key, Val) Pairs for every Cover-Item in Coverage DB*
//Input *Cov_Rec* consists of Cover Item :
// Hierarchal Scope Path, Name, Source Info,
//Coverage DB File Path, Coverage Count, Type
//Tag associated with it and the Flags

1. *Key <- Concatenate { Hier Path, Type, Name, Source Info, Tag}*
2. *Val <- Concatenate {Count, Flags, DB File Path}*
3. *Emit (Key, Val)*

Algorithm 2. Coverage-Data Merging Reduce-Function

Inputs: *P (Key, Val) Cover-Items with Unique Key Value from Map Function*
Outputs: *(Key, Total_Cov_Val) Pairs for every Cover-Item in Coverage DB and the final aggregated merged Coverage Count/Flags*

1. *Cov_Count_Sum <- 0*
2. *Cov_Aggregated_Flags <- 0*
3. **foreach** *item ∈ P* **do**
4. *{Cov_Count, Cov_Flags, Cov_Path} <- Split (item.Cov_Val)*
5. *Cov_Type <- Split (item.Key)*
6. **If** *(Cov_Type in (Assert,))* **then**
7. *Cov_Count_Sum = Logical_OR(Cov_Count_Sum, Cov_Count)*
8. **else**
9. *Cov_Count_Sum += Cov_Count*
10. **endif**
11. *Cov_Aggregated_Flags <- MergFlags(Cov_Flags)*
12. **end**
13. *Total_Cov_Val <- Concatenate (Cov_Count_Sum, Cov_Aggregated_Flags , Cov_Path}*
14. *Emit (Key, Total_Cov_Val)*

Coverage DB Writer

- The coverage DB writing starts by picking up a master DB, i.e. Largest DB from the input DBs that contains the largest number of cover-items.
- This class is the base class to which the merged values from UCTB is annotated.
- For any missing items or scopes, a clone is triggered for them from their source coverage DB file, as the path of the file is stored in the value emitted by the reducer operation, this is followed by the annotation of the merged coverage value from UCTB file.

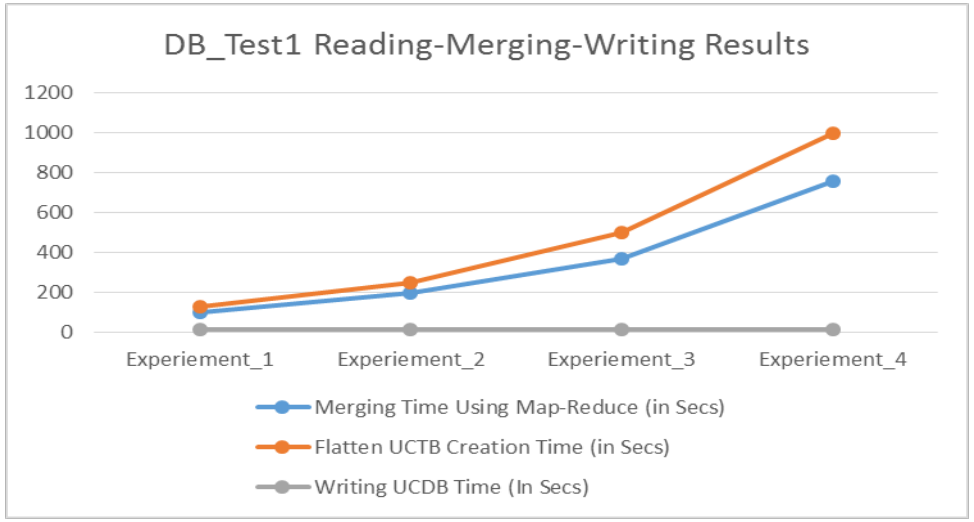
Experimental Results

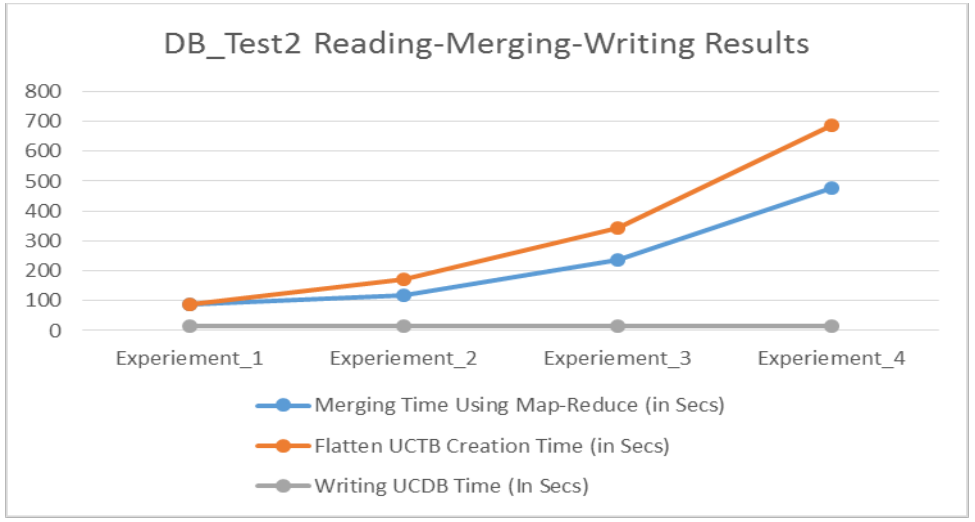
- List of Designs with Branch, Expression, Statement, Toggle, Assertions Cover Items
- The MapReduce merge results is compared against Totals Merge Algorithm
- The correctness and completeness of coverage merging results is checked by comparing the coverage data from MapReduce merger vs traditional resulted merged coverage DB from simulation.

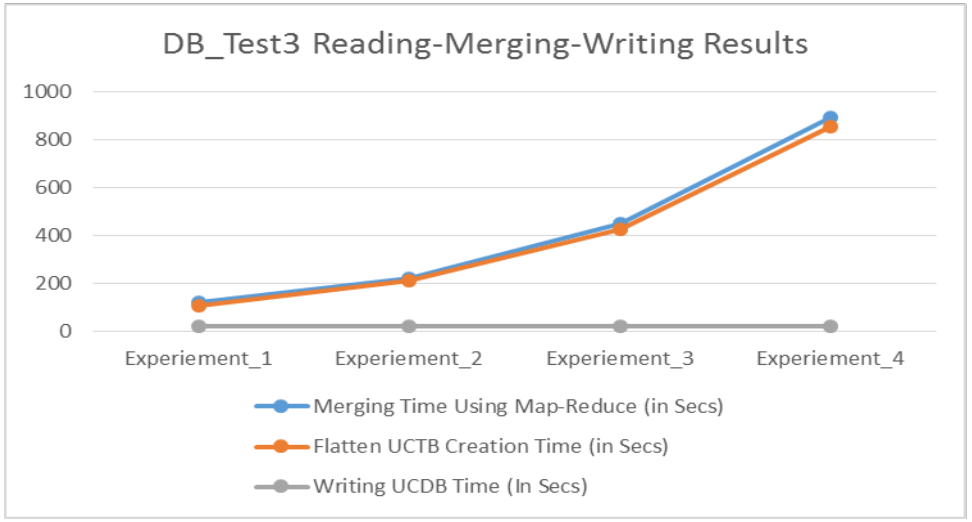
Design Name	DB_Test1	DB_Test2	DB_Test3	DB_Test4
Coverage DB Size in (MBs)	2.5	17	20	44
Total Numbers of input Coverage DBs	1000	120	120	52
Total Size of Coverage DBs in (GBs)	2.5	2.04	2.4	2.3
No. of Assertions Bins	N/A	803	803	978
No. of Statement Bins	N/A	621,406	621,406	13,105
No. of Branches Bins	N/A	383,861	383,861	5,501
No. of Toggle Bins	N/A	0	3,097,204	13,646
No. of Focused Expression Coverage Condition Bins	N/A	33,831	33,831	1,815
No. of Focused Expression Coverage Expression Bins	N/A	858,673	858,673	10,917
No. of FSM States	N/A	5,013	5,013	0
No. of FSM Transitions	N/A	15,457	15,457	0
No. of CoverPoints Bins	524,288	0	0	754,154
Total No. of Cover Items	524,288	1,919,044	5,016,248	800,116

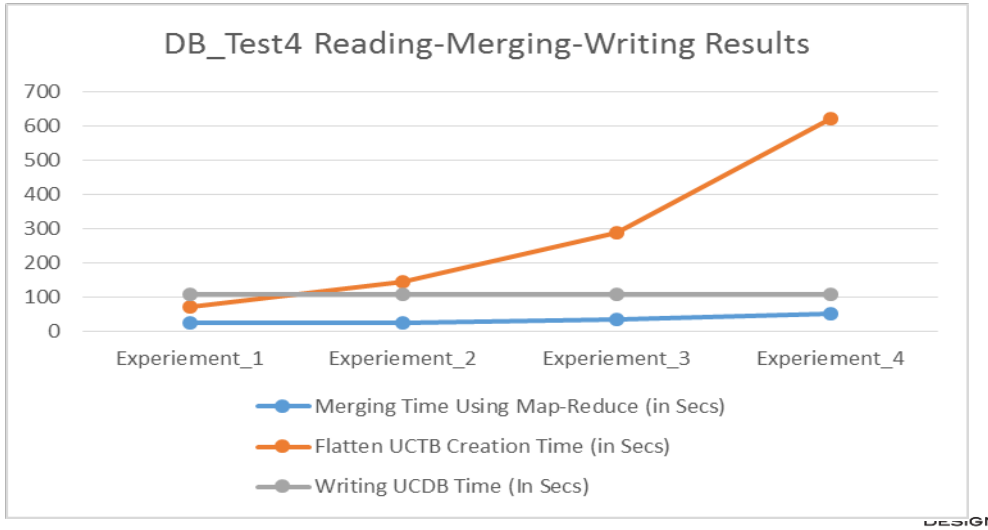
Experimental Results

Coverage Data Conversion, Merging and Writing-back Times



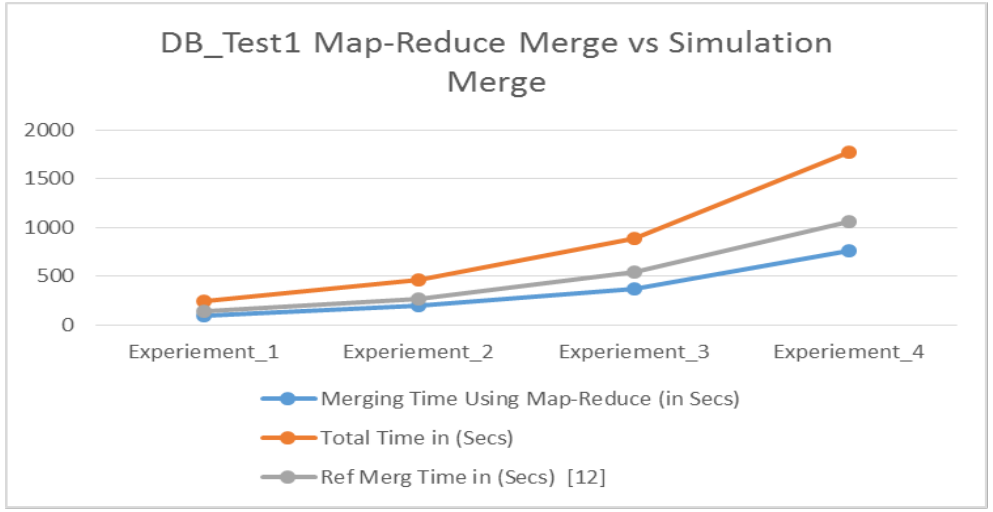


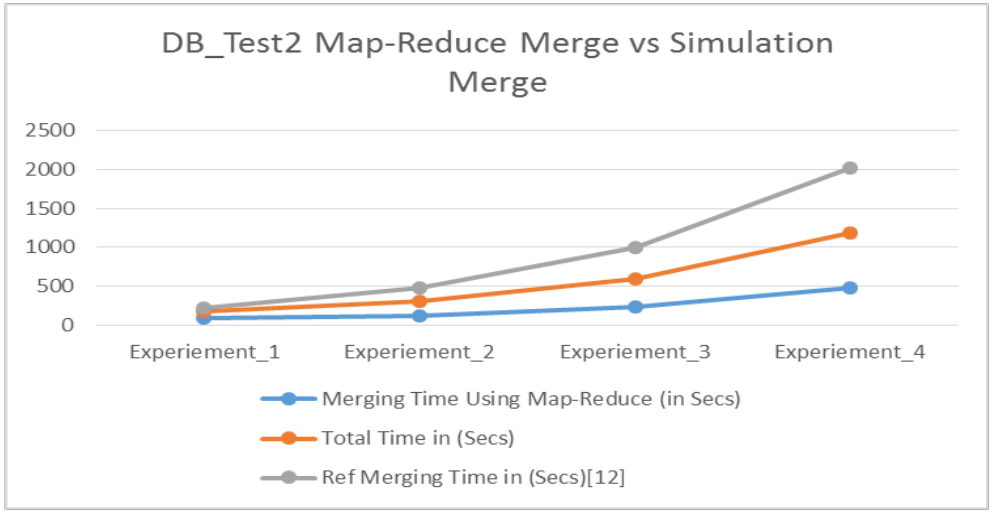


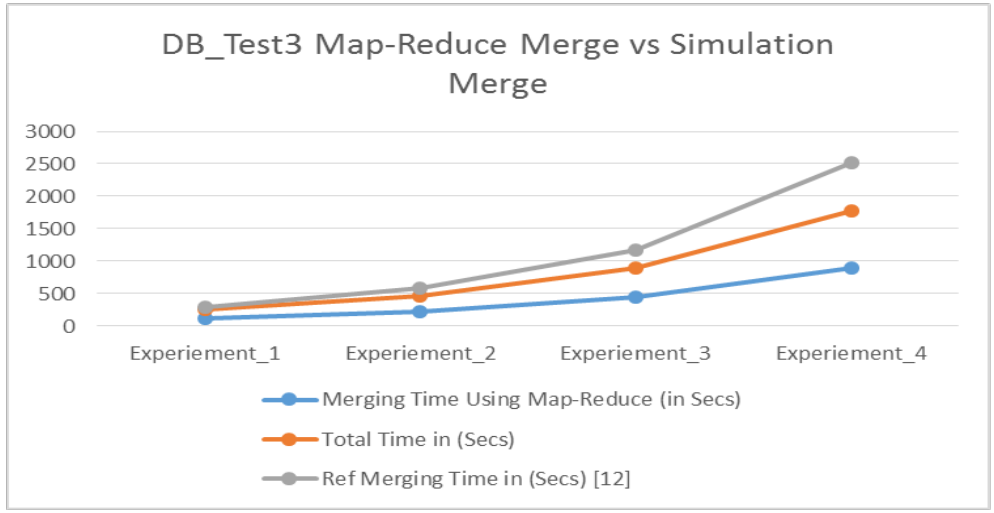


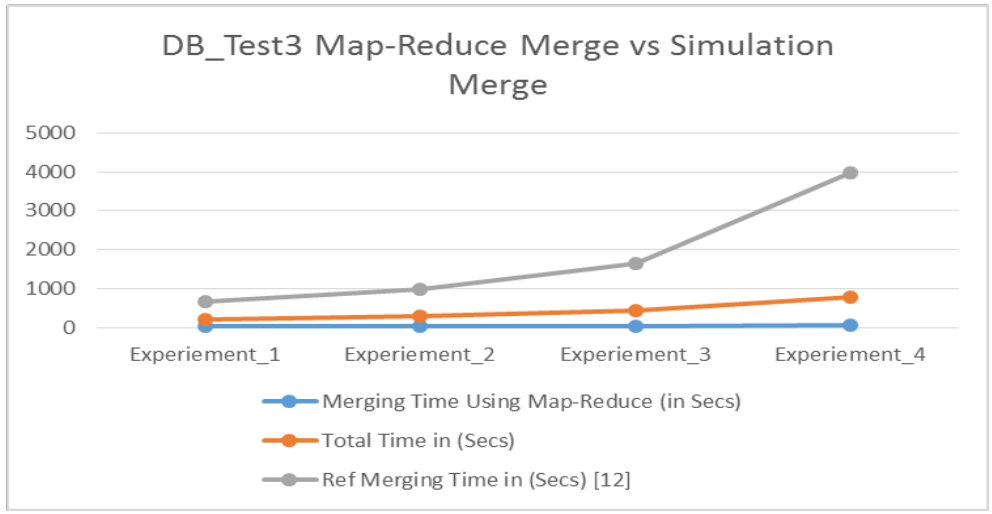
Experimental Results

MapReduce Coverage Merger with Traditional Simulation Merging Approach

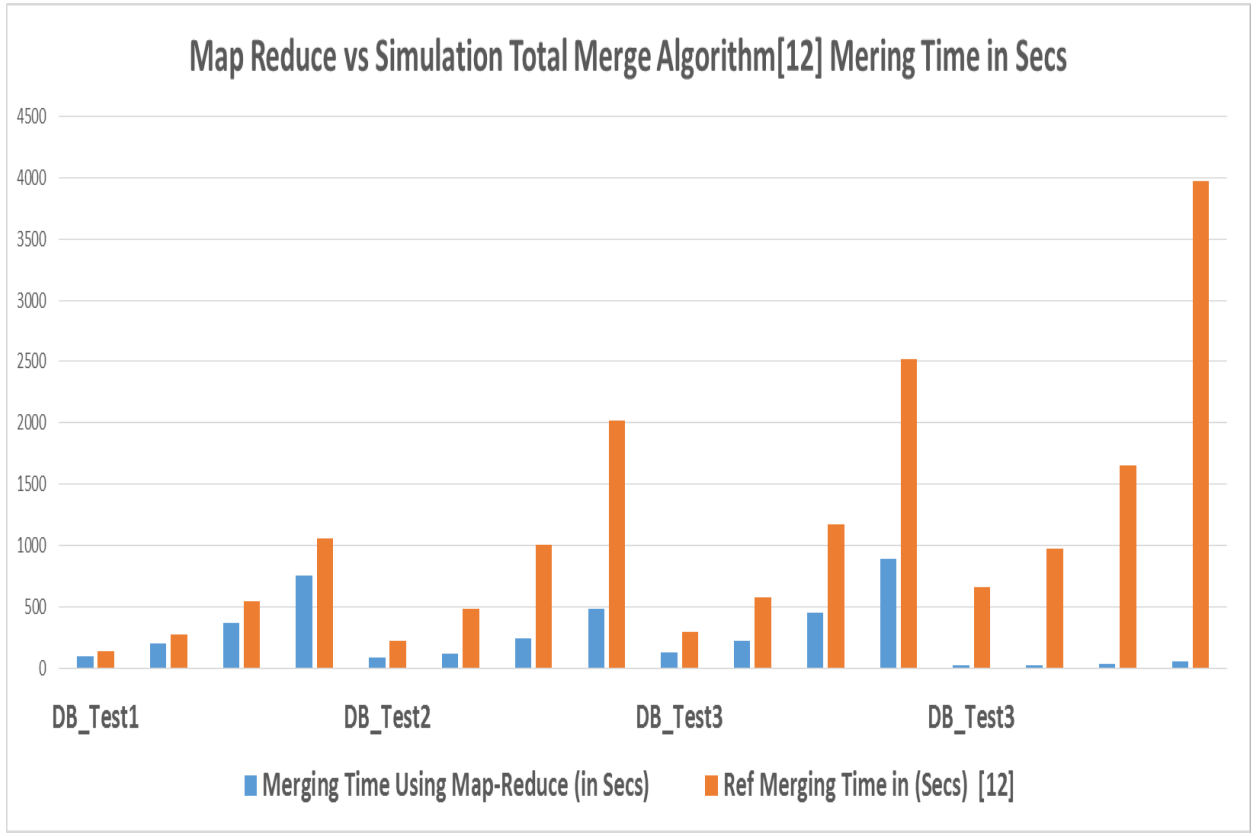


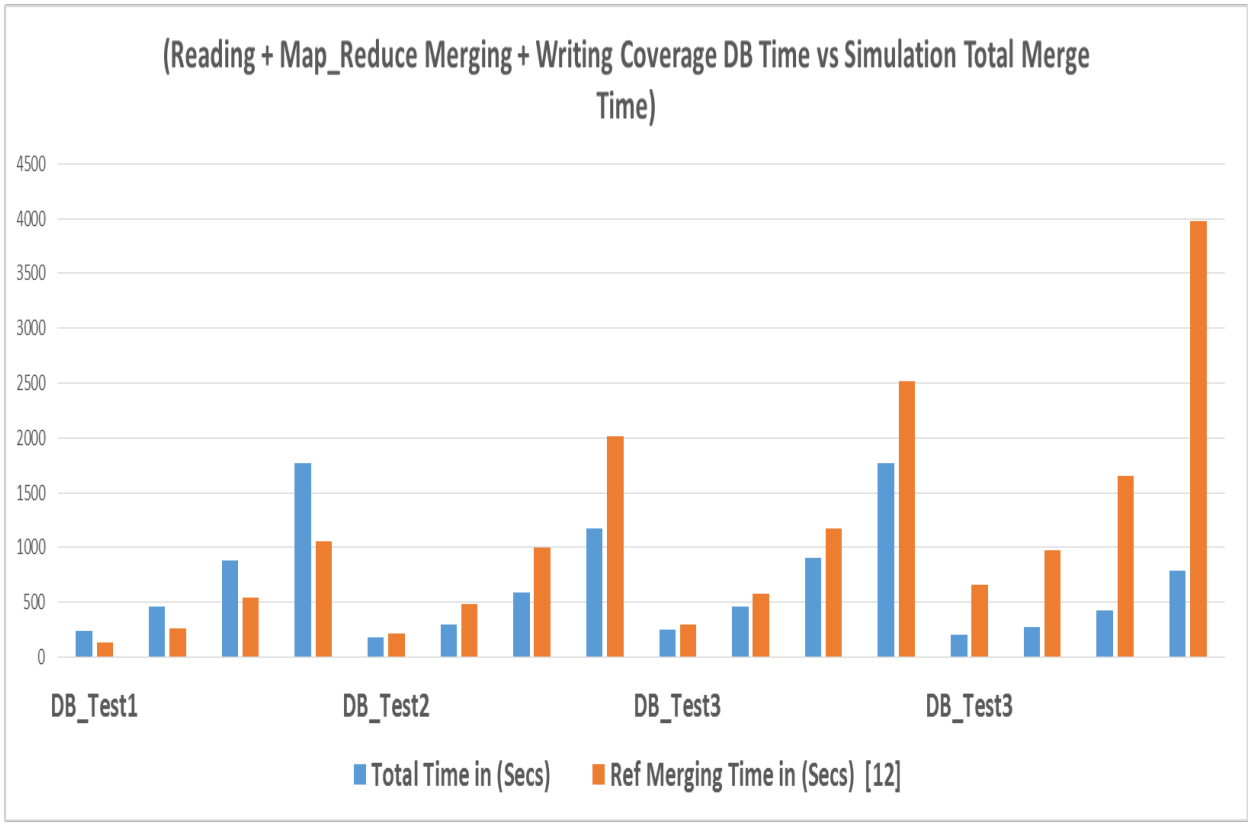






Experimental Results





Conclusion & Future Work

- Our approach demonstrates up to 1.8x speedup in the merging time of big coverage data.
- Using Big-Data Method with Coverage Data is a concept still valid to accelerate other coverage data processing steps like coverage report generation or mining coverage data for coverage trend construction.
- Having a flattened unified coverage format, that inherits the storage optimizations of UCIS representation, but at the same time allows independent proceeding for the cover-items, should be the next step in coverage data presentation.
- Our future work aims to explore how the unified coverage DBs can be proceed directly as HDFS file.
- Another direction for our future work is to extend the system to include data analyzing and mining that can result useful conclusions about the coverage data and hence help further analysis of coverage information.

Questions

Finalize slide set with questions slide