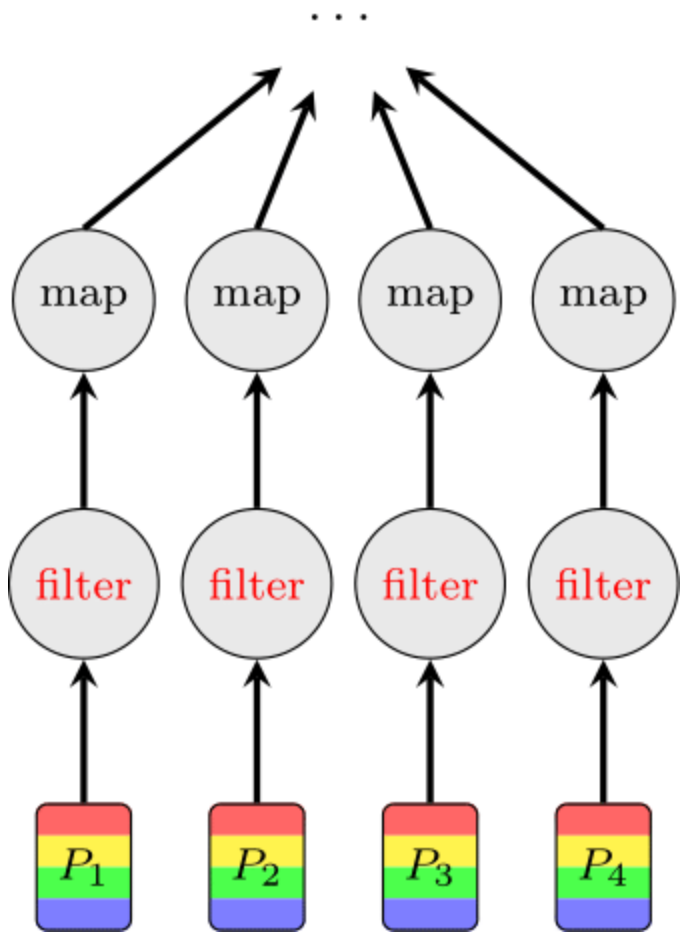
Caerus – Sampling of related techniques and open problems

Theodoros Gkountouvas, Hui Lei, Hongliang Tang, Yong Wang, Lin Zhang

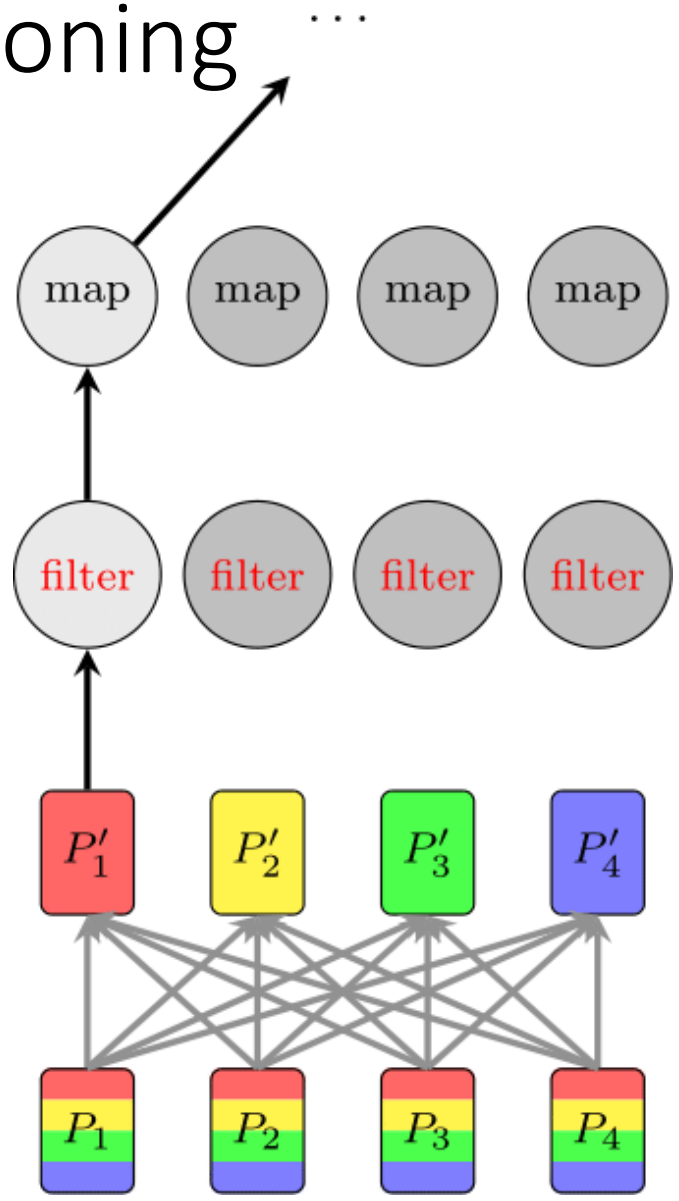
Semantic Cache - Motivation



- Query/Task example
  - Load whole dataset.
  - Filter in only all red data which is a small portion of the initial data.
  - Continue with the ensuing operations until your task/query is fully executed.
- Problem
  - Load whole data but operate only on a small portion of it.
  - Leads to excessive:
    - Storage I/O
    - Network I/O
    - CPU resources utilized
    - Memory space utilized
  - Can we leverage data/metadata from the execution of previous queries/tasks to minimize these overheads?

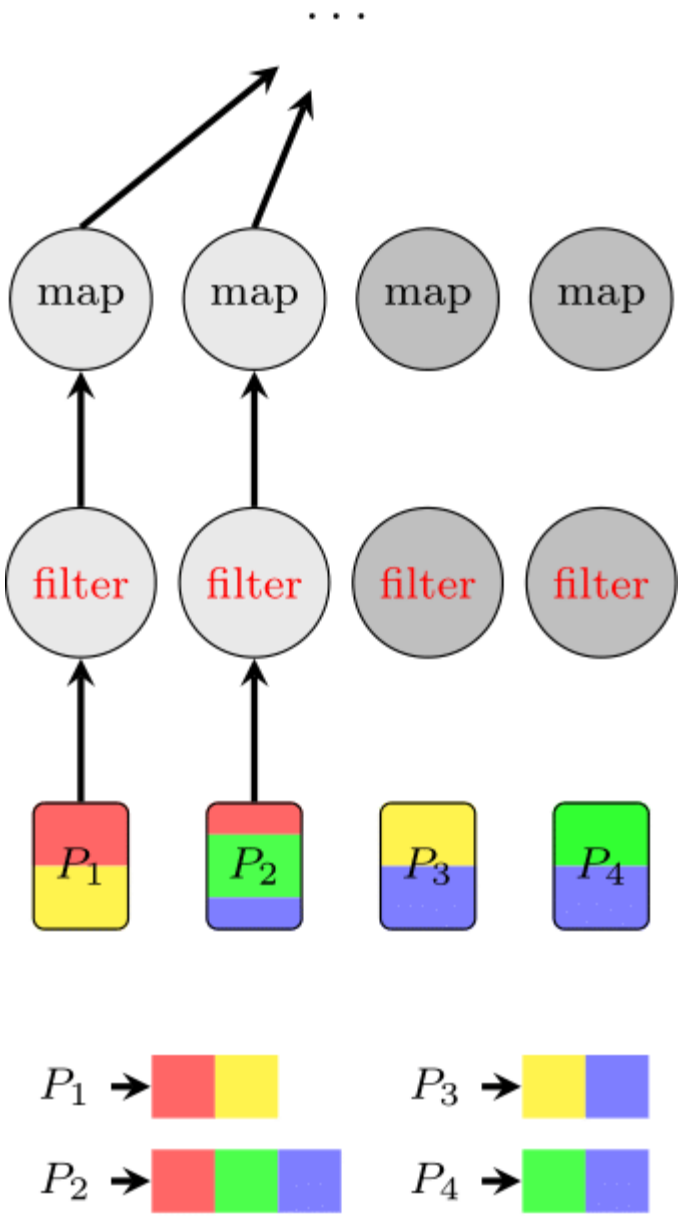
Semantic Cache - Adaptive Partitioning

- Change partitioning scheme of the underlying data.
- **Adaptive:** Dynamically adopts new partition schemes according to the executed workload needs.
- Requires costly operation which effectively requires a full shuffle operation.
- Make sure the benefits outweigh the costs:
  - Reduce the re-partitioning cost.
  - Execute multiple queries/tasks that benefit for this approach.
  - Eliminate multiple partitions from loading (partition pruning) per query/task.



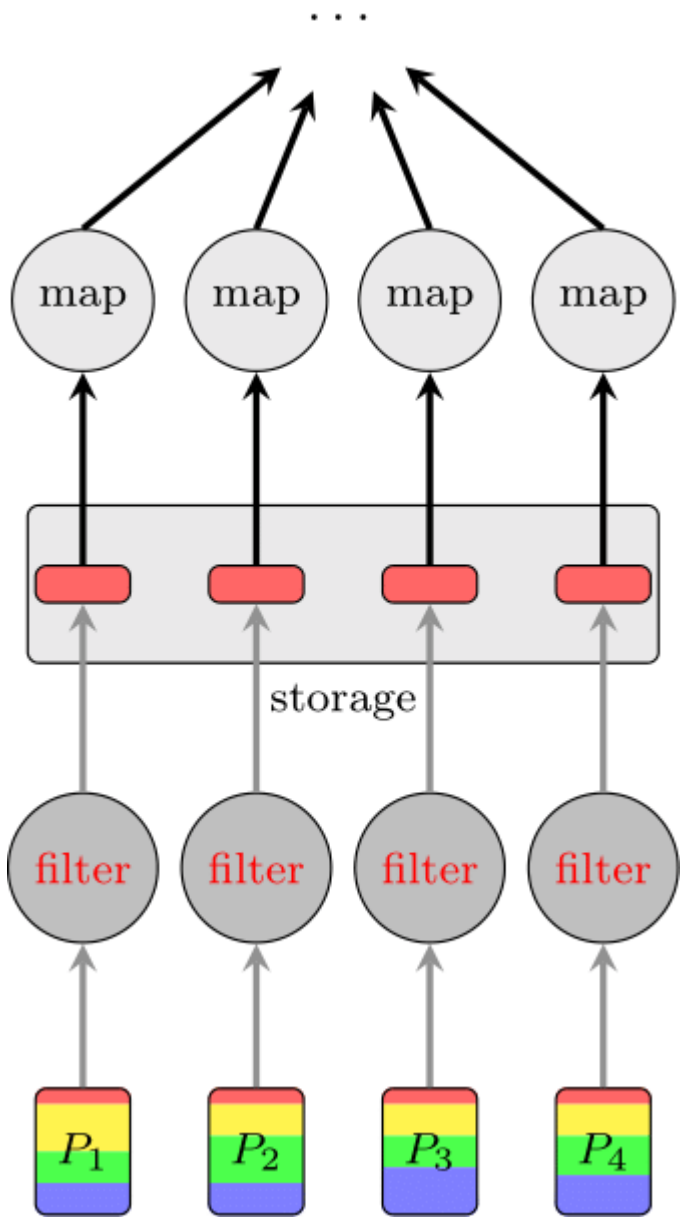
Semantic Cache - Data Skipping

- Maintain metadata for secondary attributes (not main partition attributes) per partition.
- **Skipping:** Metadata is used to skip partitions during the execution of specific query/task, that can safely eliminate all the elements of skipped partitions and still obtain the same result.
- Incurs less computation and storage overhead than *Adaptive Partitioning*.
- Less effective in data elimination than *Adaptive Partitioning*.

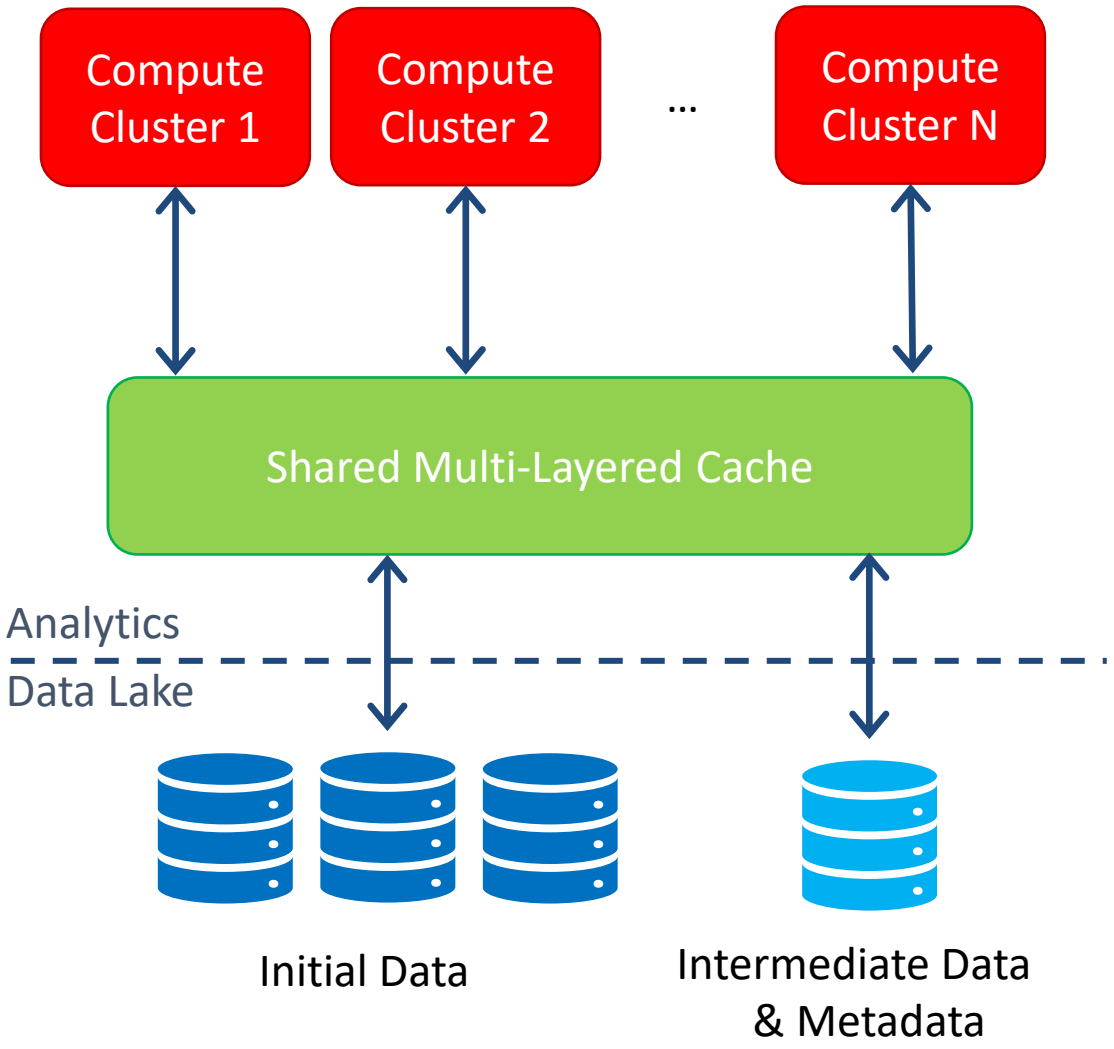


Semantic Cache - Caching Intermediate Data

- Cache intermediate results after processing the initial data (either to memory or storage).
- Any time an ensuing query/task runs, if there is no loss of information by loading the intermediate results, then only the cached data would be loaded (some operations might also be eliminated).
- Causes storage overhead commensurate to intermediate data size.
- Might cause computation overhead, since intermediate result needs to be materialized.
- Reduces storage I/O and computation overhead.

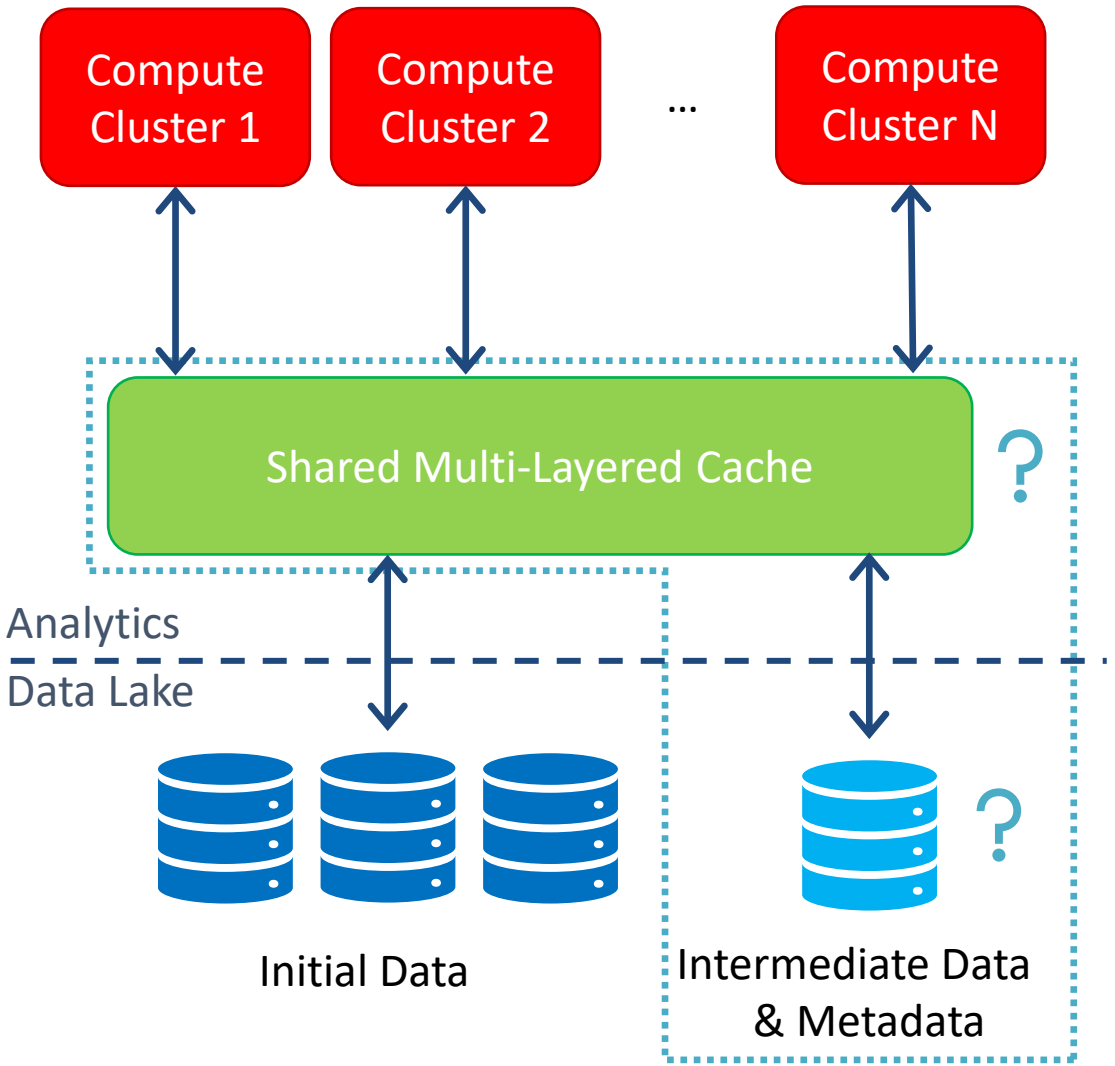


Semantic Cache - Architecture



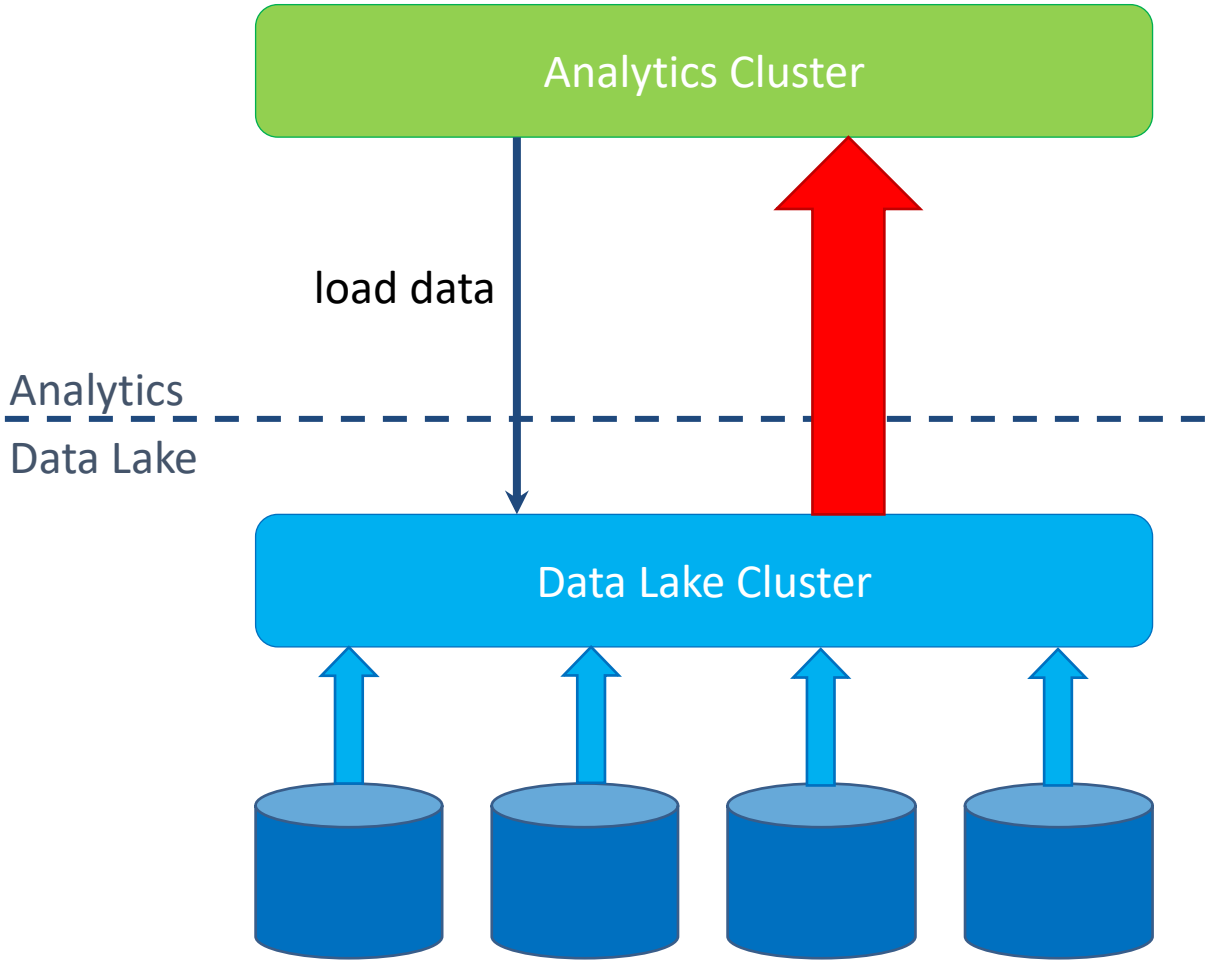
- Initial Data Storage
  - Replicates data to different storage nodes for fault-tolerance.
  - Holds a collection of files/objects/tables.
- Intermediate Data and Metadata Storage
  - Holds metadata and intermediate data, that results from processing of initial data.
  - Can reproduce content from initial data. Content is not required for correctness, only for performance improvement.
- Shared Multi-Layered Cache
  - Caches content from both initial data and intermediate data and metadata storage.
  - Utilizes different layers like conventional cache hierarchies in order to improve efficiency and enable sharing between different clusters.
- Compute Clusters
  - Perform data analysis using distributed processing engines (e.g. Spark).

Semantic Cache - Architecture



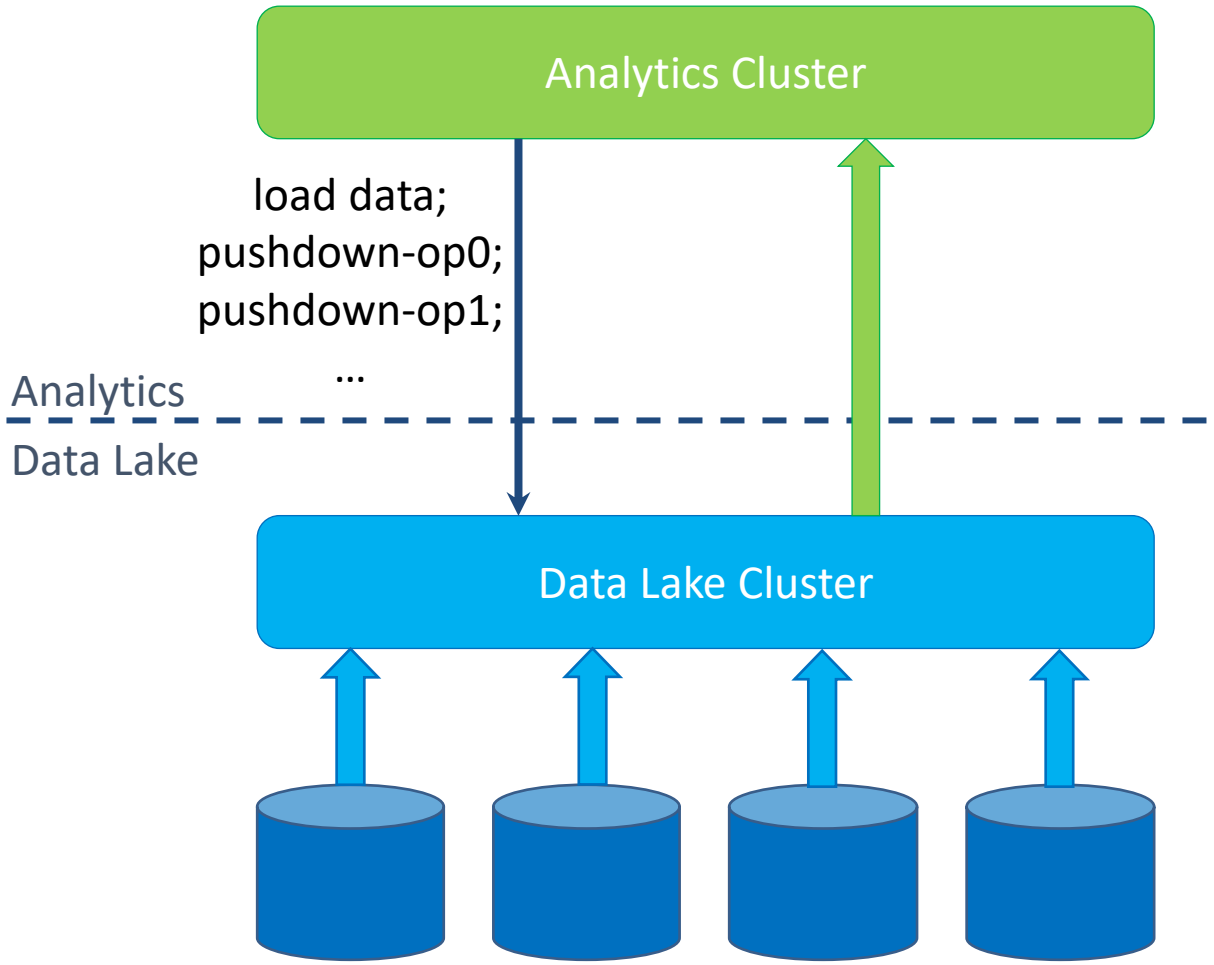
- What are the optimal configurations and policies?
  - Components
    - Multi-Layered Memory Cache
    - Intermediate Data/Metadata
  - Evaluation Metrics
    - Average query/task execution time
    - Total CPU time (compute resources)
    - Total storage I/O overhead (storage resources)
  - Principles
    - End-to-end approach is better than isolated/independent approach.
    - Use application-level semantics to improve performance.
    - Make automatic decisions without adding extra parameters and requiring modifications in the source code by users.
    - Cost-effective solutions
    - Secure and fair solutions

NDP - Motivation



1. Computations on top of whole amount of initial unprocessed data. Examples include:
   - selection, projection operations (simple operations)
   - aggregation, sample, top-k operations (Big Data)
   - classification, dataset shuffle, k-means (AI)
   - custom operations (UDFs)
2. Network I/O analogous to the amount of initial unprocessed data.
3. Memory resources required on Analysis Cluster side analogous to the amount of initial unprocessed data.
4. Data Lake processing capabilities are not utilized.

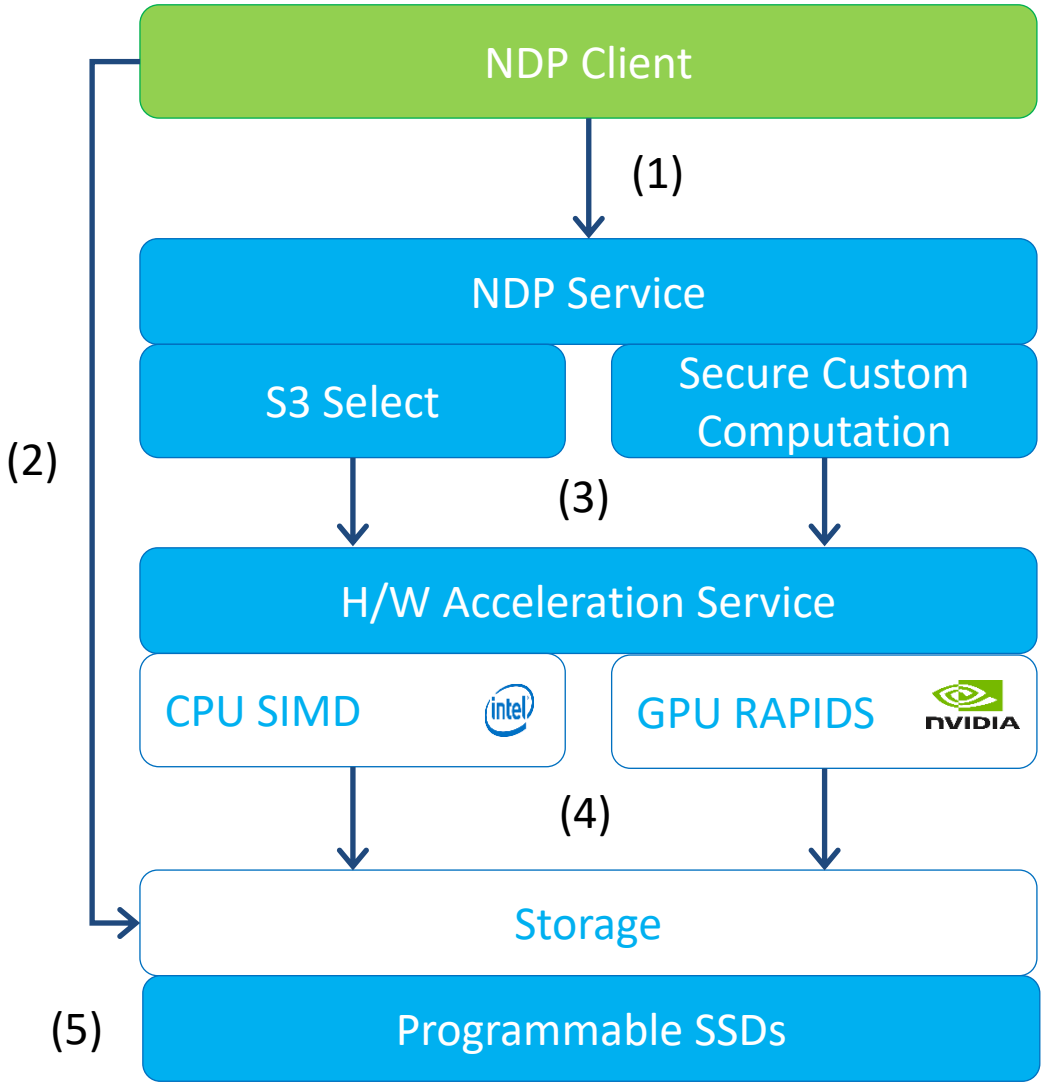
NDP - Motivation



Purpose of the near data processing pushdown operations is to **reduce** amount of operative data for ensuing operations, potentially by a large factor ($inputSize \gg outputSize$).

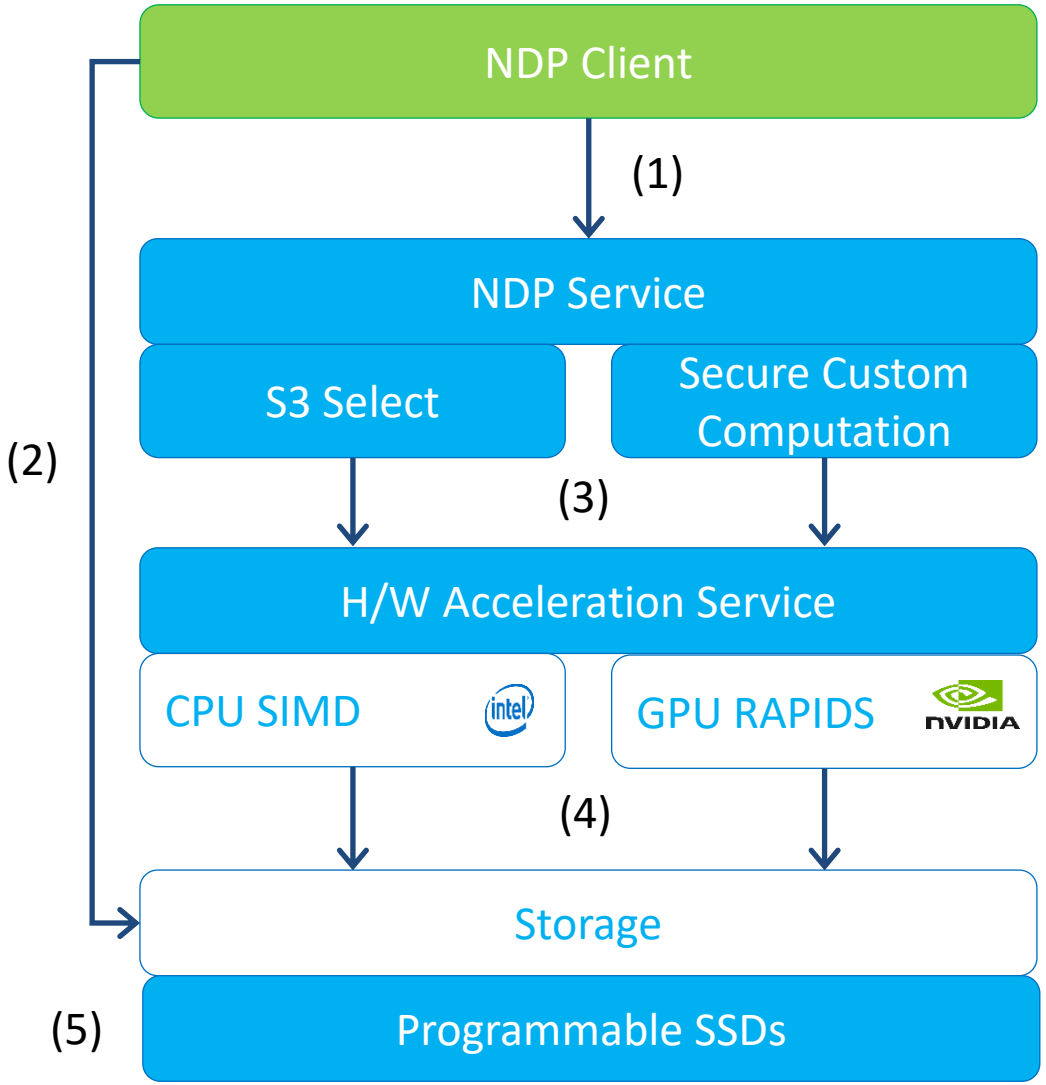
1. Ensuing computations only on top of reduced processed data.
2. Network I/O required analogous to the amount of reduced processed data.
3. Memory required on Analysis side analogous to the amount of reduced processed data.
4. Many other side benefits might include:
   - Improving the load and compute performance in Data Lake side by leveraging various software and hardware acceleration techniques.
   - Improving optimizations and decision making by utilizing execution semantics from pushdown operations (e.g. better caching decisions if optimizer is aware of pushdown capabilities).

NDP-Architecture



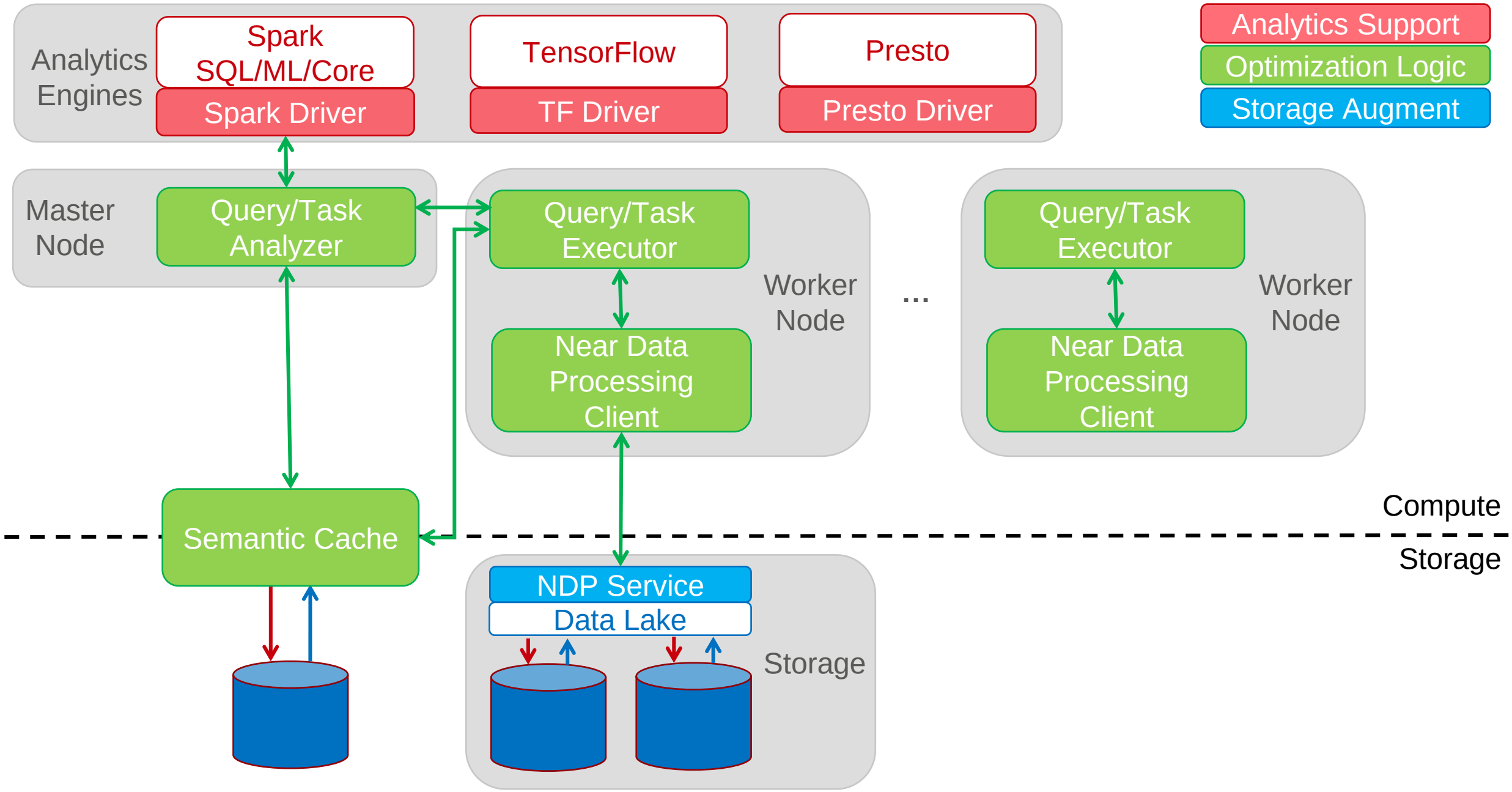
1. Object storage service using OS/POSIX interfaces. Use HTTP(s) based REST API, that can support custom request body to embed pushdown information. Three popular examples of similar approaches are:
   - AWS S3 (SelectObjectContent or S3 Select)
   - SNIA CDMI (Cloud Data Management Interface)
   - OpenStack Swift (Storlet)
2. If the request doesn't have the pushdown request body in the HTTP(s) request, the request should be forwarded directly to storage service (out-of-band approach).
3. The NDP service decides if the pushdown operations should be executed by the S3 Select module (SQL operations) or the more generic one (custom computations). Caerus S3 Select Service and Secure Custom Computations modules can be implemented as one module. However, due to increasing popularity of SQL query on storage, we plan to support it separately (performance, security).

NDP-Architecture

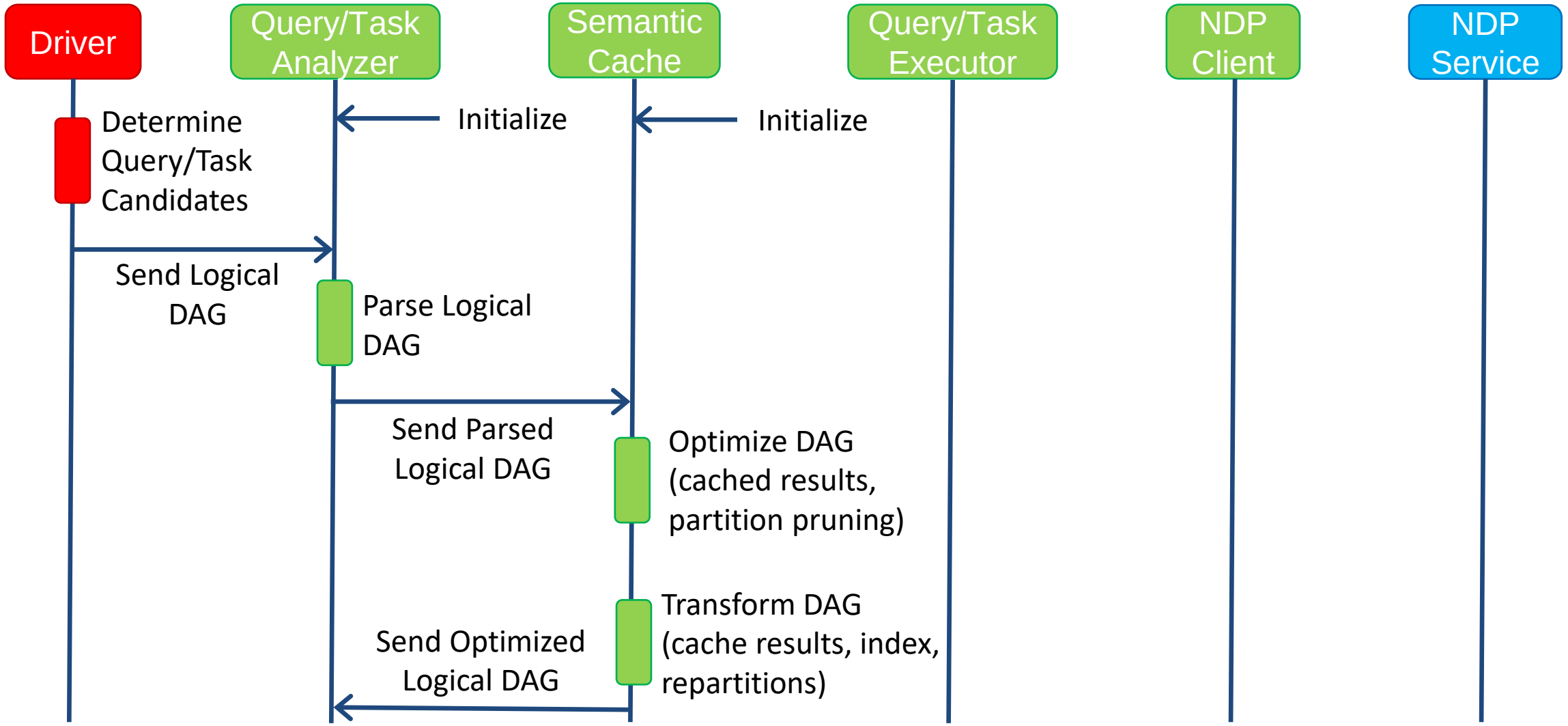


4. The Caerus Hardware Acceleration Service will apply acceleration when input or output data objects are processed. The two major techniques in the consideration are:
   - CPU SIMD Technology
   - Open GPU RAPIDS Data Science Technology
5. In normal cases, there are storage media layer like HDDs and SDDs. Caerus plans to extend pushdown further into storage media, more specifically, into programmable SSDs (in I/O context) or SCM (in memory context). Caerus aims to significantly reduce storage system resource consumption and improve performance in some cases by getting advantage of programmable SSDs.
6. Return path is omitted for simplicity.

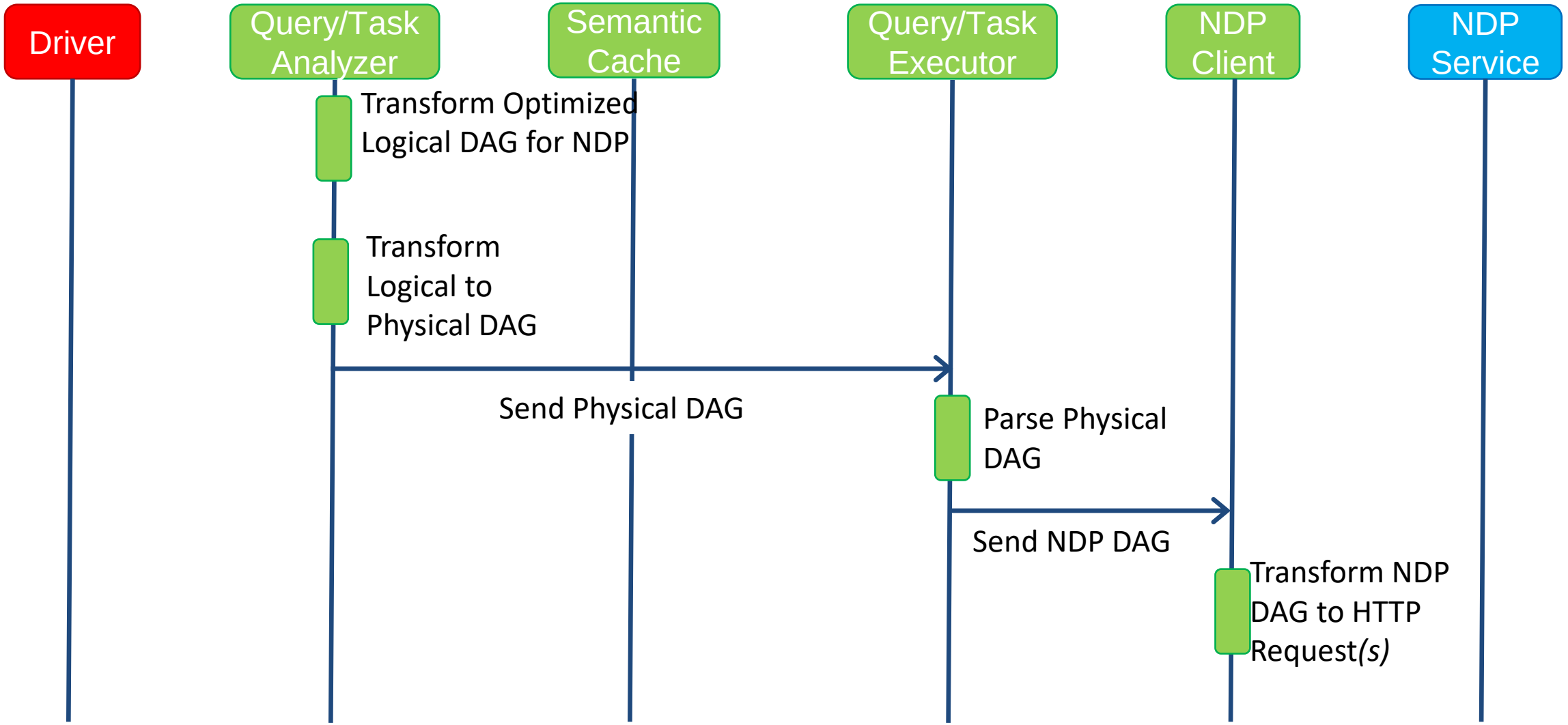
Holistic Approach - Architecture



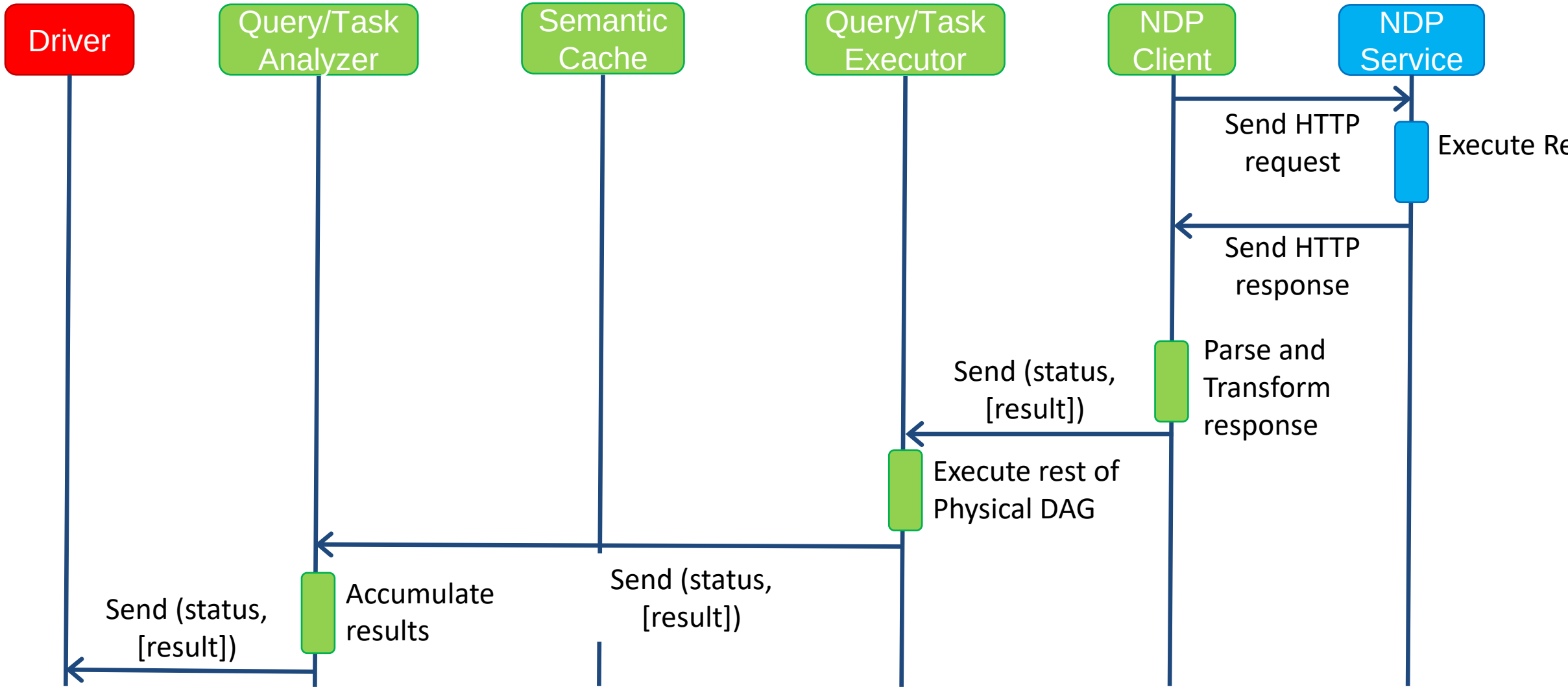
Holistic Approach - Design



Holistic Approach - Design

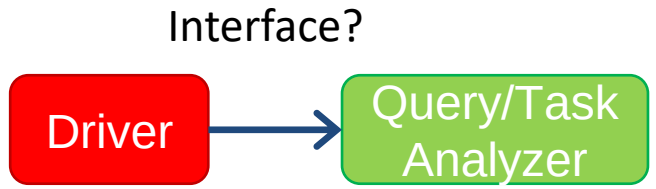


Holistic Approach - Design



Open Problems

- **Holistic Approach**
  - Optimizations are applicable to multiple compute engines. What is a good representation for communication between compute engines and our platform? (Peregrine, SOCC '19)
    - Support Big Data and ML workloads.
  - How do we optimize execution of multiple workloads (Sparkcruise, VLDB '19)?
    - Multiple Techniques into Consideration
      - **Adaptive Partitioning**
      - **Data Skipping Metadata**
      - **Caching Intermediate Data**
      - Prefetching/Precomputing
      - Scheduling
      - NDP execution
    - Explore sophisticate evaluation goals.
      - Performance (query/task execution time)
      - Resource usage (storage I/O, network I/O, CPU usage, etc.)
      - Cost-effectiveness (cost in $$)



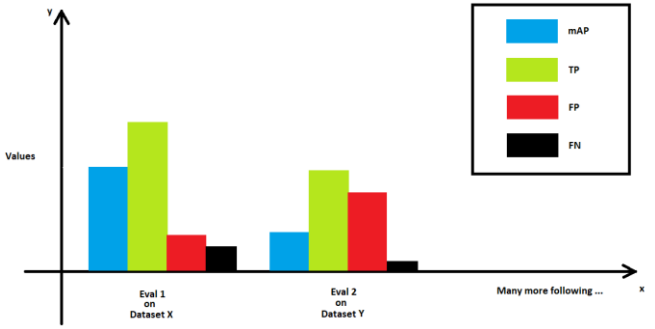
Minimize *???*

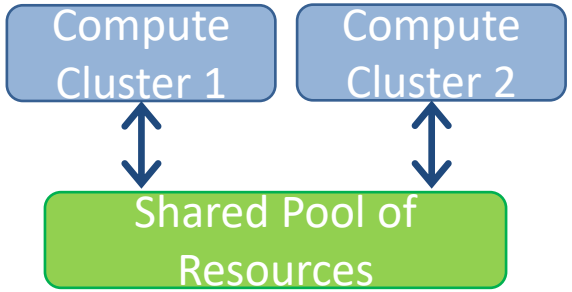
Constrained To $storage\ size \leq storage\ capacity$

$memory\ size \leq memory\ capacity$

Open Problems

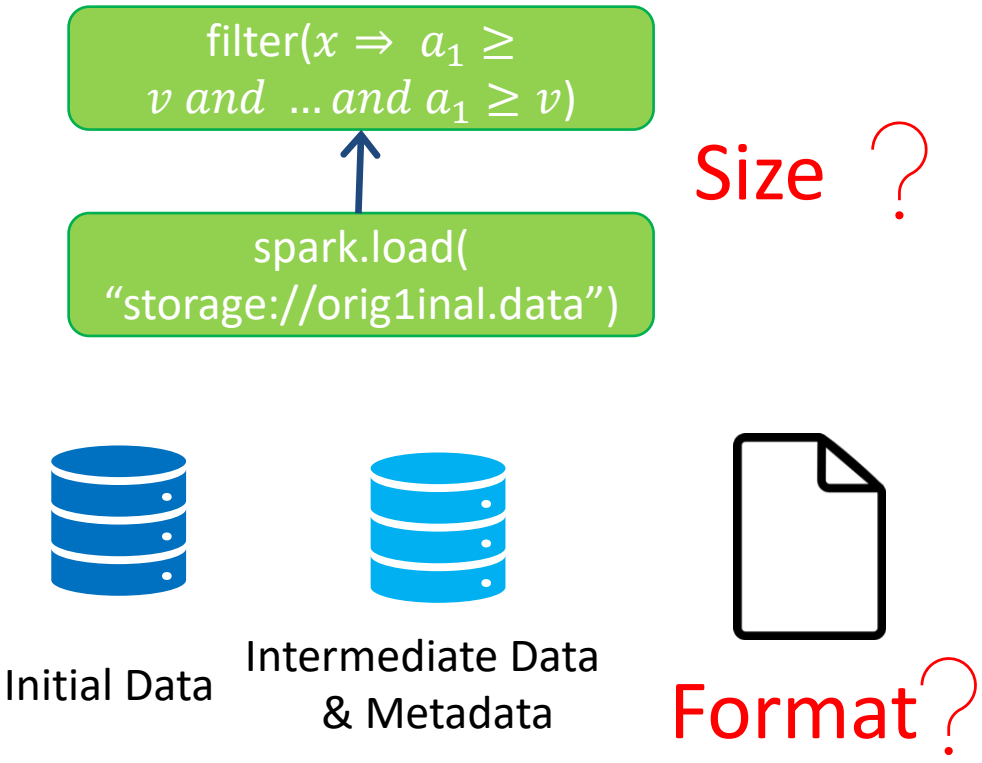
- **Holistic Approach**
  - Evaluation is critical. We need traces of query/task executions (not just different types of queries/tasks like TPC benchmarks). How do we create one?
    - Randomly generate a sequence of query tasks with different parameters from existing benchmarks.
    - Rely on frequencies for different type of queries to generate sequences. Parameterization is still random.
  - We work in multi-tenant environments. What is fair use of these shared resources?
    - Provide meaningful fairness guarantees (e.g. perform at least as well as running in isolation with the fair amount of cache and NDP resources) for the multi-cluster environment.
    - Accommodate priority scheduling. Some jobs might be more critical compared to others.
  - Security guarantees are important in the multi-tenant environment. How Caerus can provide important security properties during the execution of multiple workloads in the presence of adversaries, like confidentiality, integrity and availability?





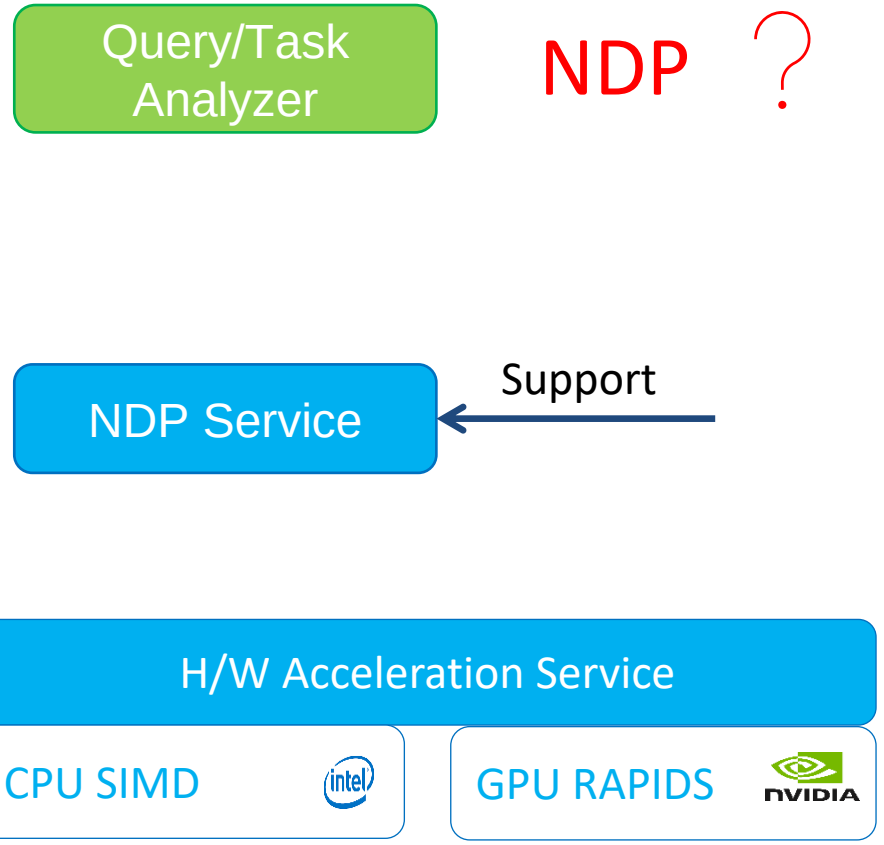
Open Problems

- **Semantic Cache**
  - Better information leads to better decisions. How can we improve this information?
    - Intermediate size estimation
    - Runtime estimation
    - Resource usage estimation
  - What is an appropriate data format for initial data and intermediate data/metadata?
    - Try to support as many formats as possible for initial data.
    - Investigate what is the most performant format for Semantic Cache.
    - Explore data ingestion issues.
  - Can all the utilized techniques be further improved?
    - Better cost for repartitioning?
    - **Better candidate selection (indexing, caching, repartitioning)?**
    - More sophisticated indexing (**workload-aware indexing**)?
    - More sophisticated intermediate data caching (workload-aware)?

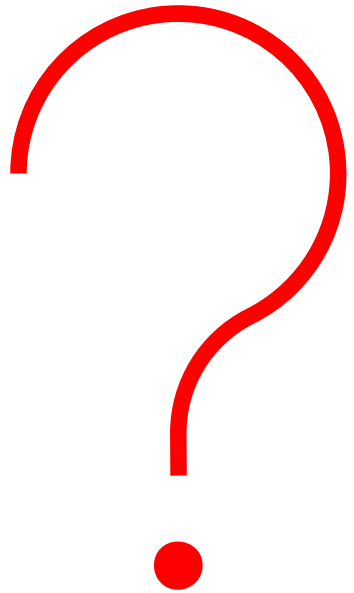


Open Problems

- **Near Data Processing**
  - Can we automatically determine NDP operations to pushdown based on storage characteristics/capabilities on compute side?
  - **What type of operations provide benefits when pushed to storage side?**
    - Filter
    - Project
    - Aggregation
    - Sample
    - Top-K
    - …
  - How can we benefit from H/W acceleration?
    - More operations can be supported (only pushdown predicates are supported now).
      - Range predicates
      - Substring predicates



Thank You



Competitive Analysis – Semantic Caching

Company	Product	Storage System	Distributed Caching	Multi-Modal Caching	Predictive Caching	Workload-Aware Caching	Multi-Tier Caching
Alluxio	Alluxio	Multiple	Centralized	Manual	Manual	Manual	Memory, SSD, HDD
Databricks	Spark Cache	Multiple	Decentralized	Source Data, Intermediate Data	No	Spark DAG	No
IBM	Big Data Skipping	COS	Centralized	Indices	Yes	No	Storage
Microsoft	Microsoft Azure – Databricks	Multiple	Centralized	Source Data, Intermediate Data (including re-partitions)	No	Yes	Memory, Storage

Research Innovations – Semantic Caching

University	Project Name	Conference	Storage System	Distributed Caching	Multi-Modal Caching	Predictive Caching	Workload-Aware Caching	Multi-Tier Caching
Texas, Microsoft Research India	INSTalytics	FAST 2019	Filesystem (HDFS)	Centralized	Source Data, Re-partitioned Data	No	Selection Operation	Only Storage
MIT CSAIL, Microsoft	Amoeba	SoCC 2017	Filesystem (HDFS)	Centralized	Source Data, Re-partitioned Data	No	Selection Operation	Only Storage
UC Berkeley	Partitioning for Aggressive Data Skipping	SIGMOD 2014	Filesystem (HDFS)	Centralized	Indices	No	Selection Operation	Only Storage
Hong Kong	LRC, LERC	INFOCOM 2017	Multiple	Centralized	Intermediate Data	No	Spark DAG	Memory /Storage
Boston, Northeastern	Caching in the Multiverse	HotStorage 2019	Ceph	Centralized	Intermediate Data	Yes	Pig, Hadoop DAG	Memory /Storage

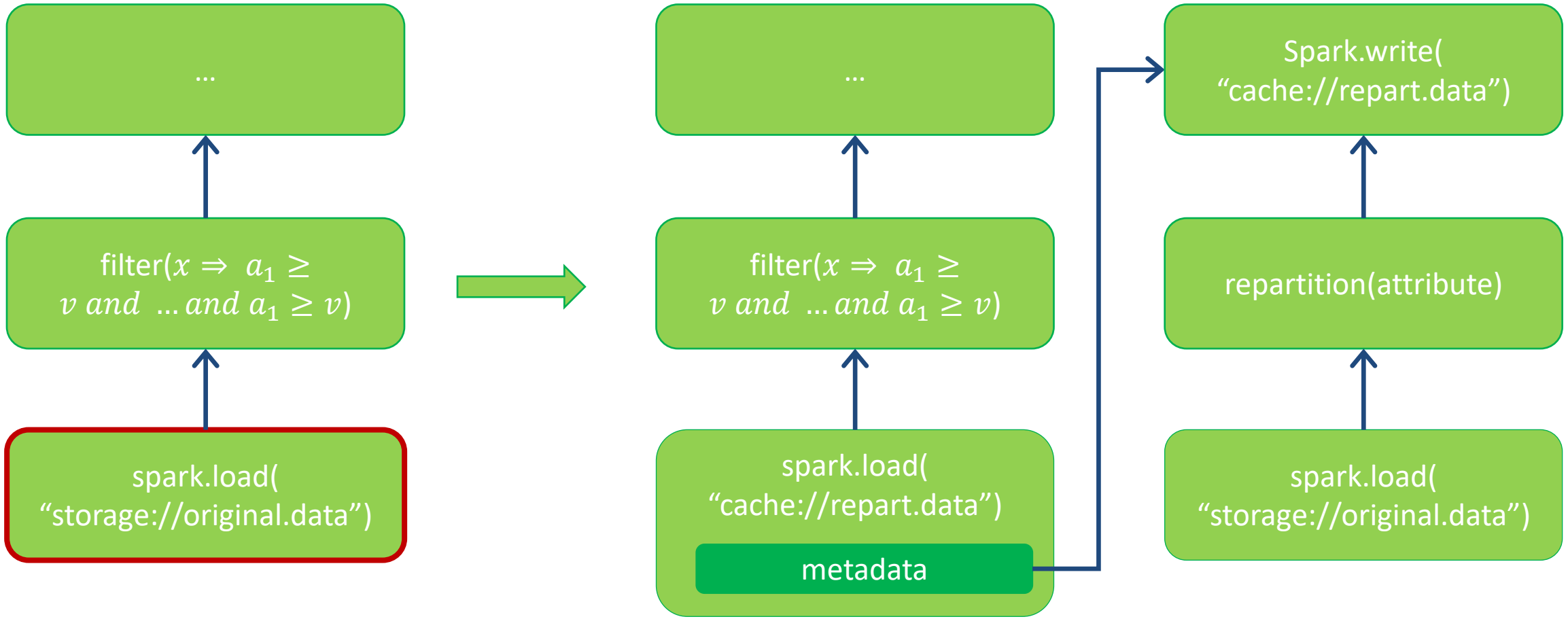
Competitive Analysis - NDP

Company	Product	Analytics Engine	Storage System	What to Pushdown	How to Pushdown	Where to Pushdown to	How to Execute	Notes
MinIO	S3 Select	Spark (Spark-Select)	Object Storage: MinIO	Simple projections, predicates, and aggregation	SQL	Storage processors (SPs)	SPs ingest, parse and filter CSV and Json files with limited hw/sw acceleration	• Better performance than AWS S3 • Not support complex operations • No I/O optimization due to lack of indexing • No UDFs
Amazon AWS	S3 Select and Glacier Select	Up to users	Object Storage: AWS S3	Simple projections, predicates, and aggregation	SQL	Information unavailable, probably similar to MinIO	Information unavailable, probably similar to MinIO	• Become object storage standard • Great APIs and SDKs • Not support complex operations • No UDFs
RedHat (IBM)	Ceph Object Classes (CLS)	UDFs and Postgres	Unified Storage (block/file/object) based on Object Storage: Ceph	UDFs and most SQL operations	UDFs and SQL	SPs to object store (OSD)	LUA VM on storage side can talk to OSDs via extension interfaces	Need extensive work to port CLS like implementation into other storage systems
IBM	OpenStack SWIFT Storlet	Spark etc.	Object Storage: SWIFT	UDFs and most of SQL operations	UDFs and SQL	Via SWIFT proxy to storage nodes	Storlet Dockers on storage side can support UDFs and SQL	• Specific to SWIFT, hard to port to other systems • No sw/hw accelerations
Oracle, IBM etc.	Exadata (Oracle) Netezza (IBM)	RDMS	Non-disaggregated appliances	Wide range of SQL operations	SQL	Information unavailable, all-in-one box or vendor-controlled storage servers	Information unavailable, all-in-one-box or vendor-controlled storage servers	Traditional RDMS appliances or vendor controlled storage server, hard to reuse in generic storage systems
MongoDB	MongoDB (NoSQL)	NoSQL	Proprietary storage	Wide range of query (Json) operations	NoSQL and SQL (wrapper)	Storage engines	Storage engine manages how data is stored both in memory and drives	Proprietary storage specific to this database, hard to reuse in generic storage systems

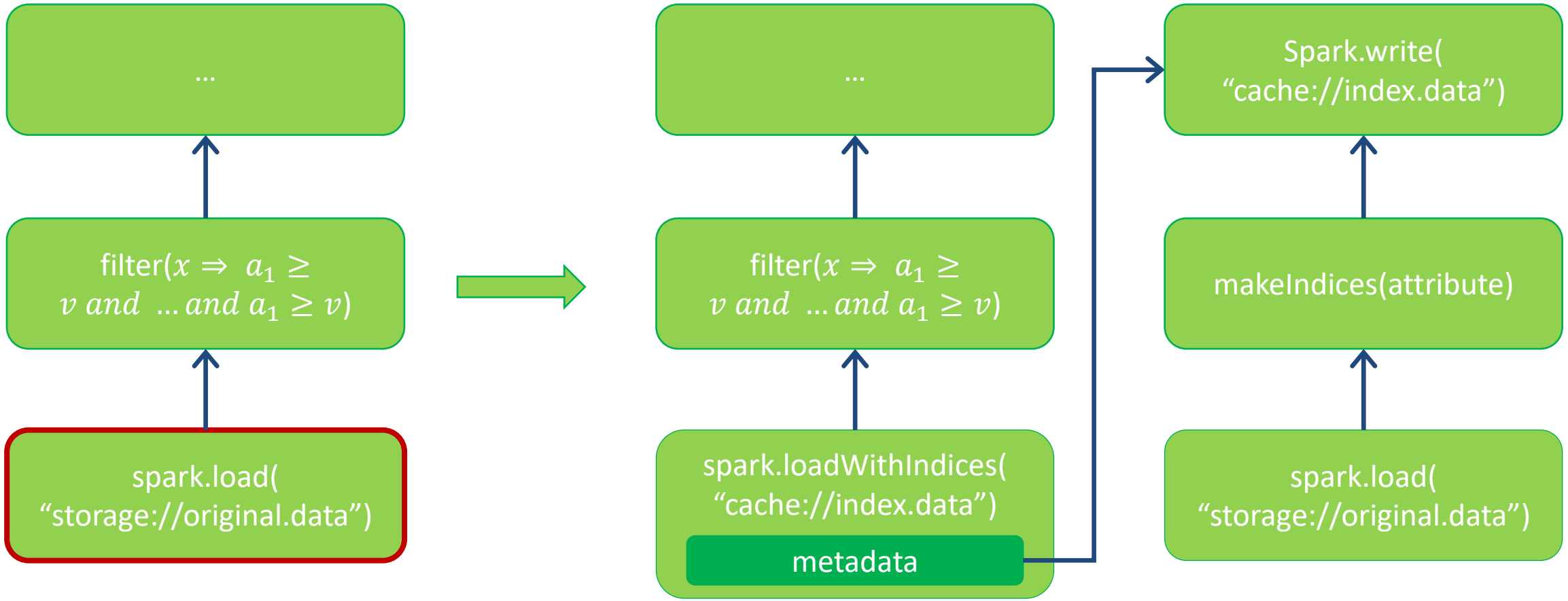
Related Research - NDP

University	Project Name	Conference	Analytics Engine	Storage System	What to Pushdown	How to Pushdown	Where to Pushdown to	How to Execute
University of California Santa Cruz	SkyhookDM	Ongoing research project, many conferences: Vault'19, '20, FAST19 etc.	Postgres	Ceph	UDFs and most SQL operations	UDFs and SQL via Postgres FDW (Foreign Data Wrappers)	SPs to object store (OSD) via CLS	LUA VM on storage side can talk to OSDs via extension interfaces CLS.
University of Wisconsin-Madison & MIT	PushdownDB	ICDE2020	Homegrown PushdownDB on top of AWS S3	AWS S3 (Select)	Extended SQL operations like join, group-by, and top-K	Used native S3 Select operations	AWS S3 object storage	Used multiple-steps commercial available AWS S3 Select to support more SQL operations
Reutlingen University & TU Darmstadt, Germany	NativeNDP	ADBIS 2019	R Platform	Ceph	Data transformation	Used UDF via Ceph CLS to push R-native operations on to their storage abstraction DataFrame; similar to SkyhookDM	Pushdown to Ceph object store (OSDs); Then further push down to SCM storage devices; similar to SkyhookDM	Implemented Ceph CLS for R by using Ceph RADOS Object APIs on storage side, so it can run UDFS to transform data on storage side; similar to SkyhookDM
University of California, Irvine	Catalina ("Catalina: In-Storage Processing Acceleration for Scalable Big Data Analytics")	Journal of Big Data, 2019 & Euro Conf on Parallel, Distributed and Network-based Processing 2019	MPI (Message Passing Interface) & Hadoop	Edge Computational Storage Devices, SNIA term "CSD"	Special example: image similarity search using MPI	Embedded MPI slaves in storage devices, so that everything is a pushdown	ISP (In-Stroage-Processing) Engine: including ISP OS (user space + kernel), firmware, FPGA and storage	Using ARM and FPGA to process and accelerate; using programmable SSD (Flash interface) to pushdown operations
KTH Royal Institute of Technology, Sweden	Not applicable ("Identifying the potential of Near Data Processing for Apache Spark")	International Symposium on Memory Systems 2017	Spark	Server Storage	Not applicable, but they did great job to measure compute, memory and storage distribution in wide range of Spark workloads like Spark core, SQL, ML, Graph X, and Streaming, to see what are the best workloads for ISP pushdown, PIM (processing-in-memory, in relate to caching) pushdown or hybrid: ISP matches well with normal Spark SQL and Spark core etc. non-iterative batch; PIM suits streaming processing and iterative batch processing workload in Spark; Machine Learning workloads in Spark are best for hybrid of ISP and PIM. This can provide guideline for our holistic orchestration.			

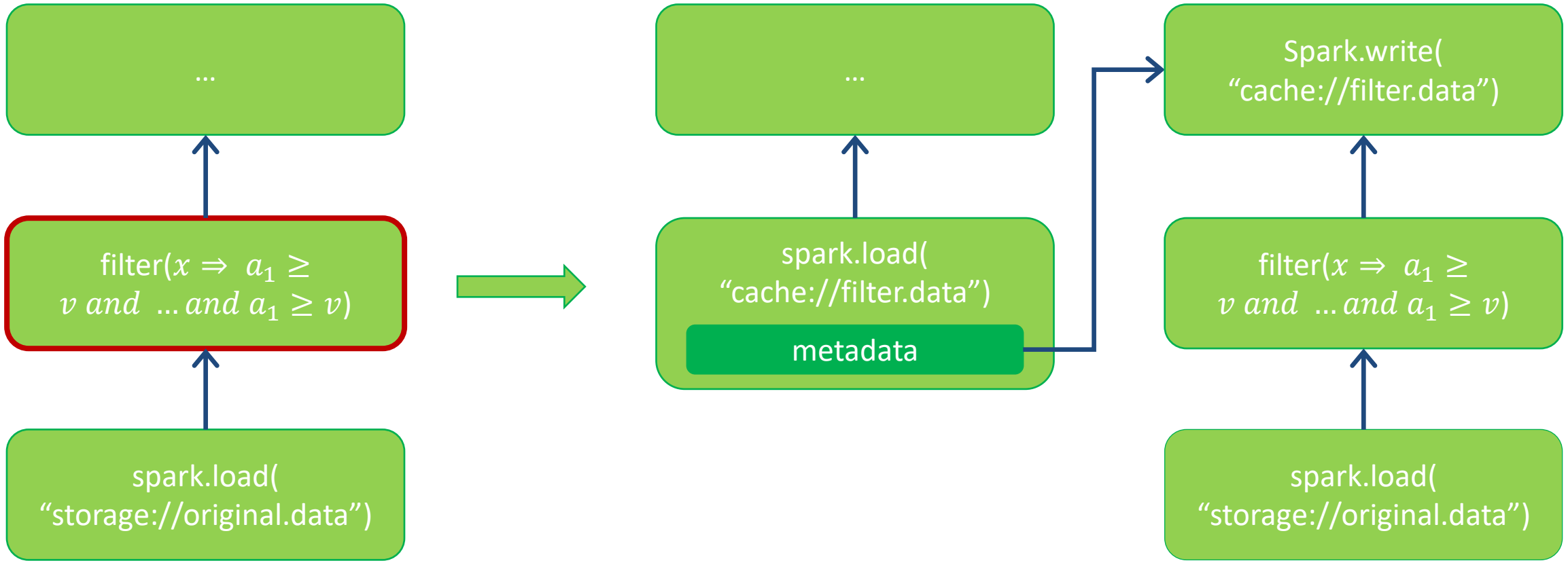
Example: Repartitioning



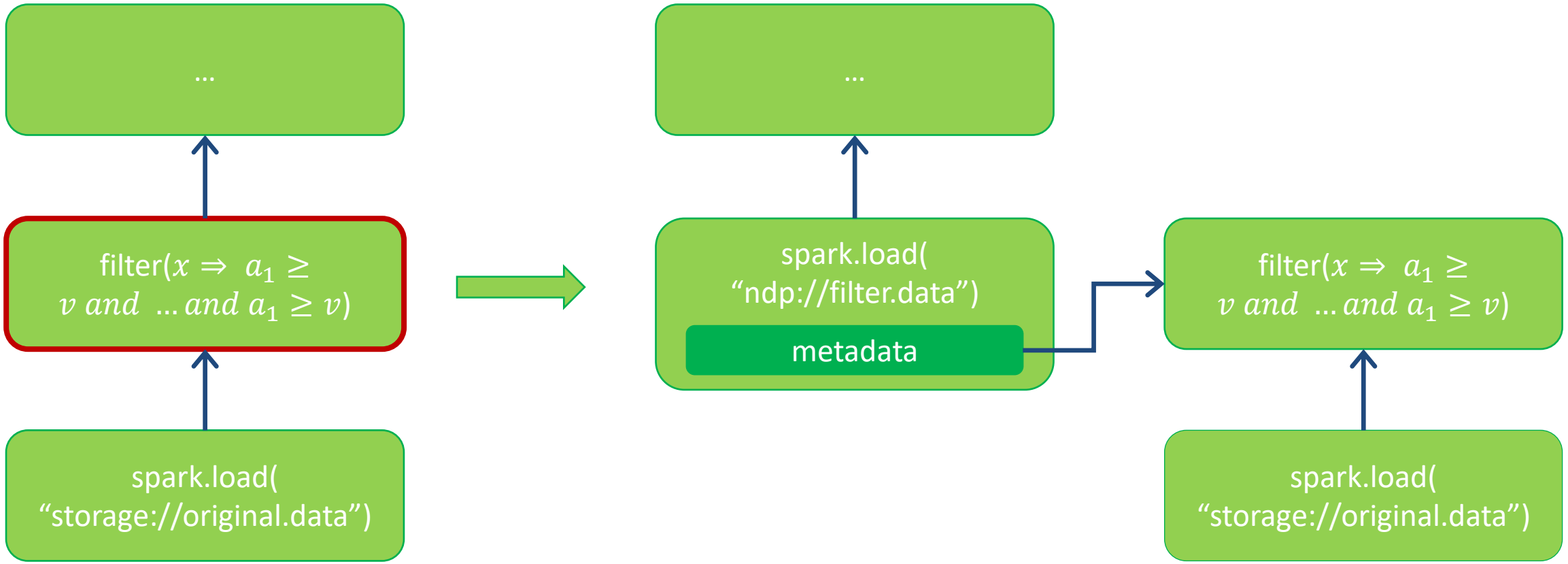
Example: Indexing



Example: Caching



Example: NDP


...
filter($x \Rightarrow a_1 \geq v$ and ... and $a_1 \geq v$)
spark.load("storage://original.data")
...
spark.load("ndp://filter.data")
metadata
filter($x \Rightarrow a_1 \geq v$ and ... and $a_1 \geq v$)
spark.load("storage://original.data")