GOTC

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE , OPEN WORLD

综合技术专场

Vineyard：开源分布式数据管理引擎

何涛 2021年 8月 1日

阿里巴巴

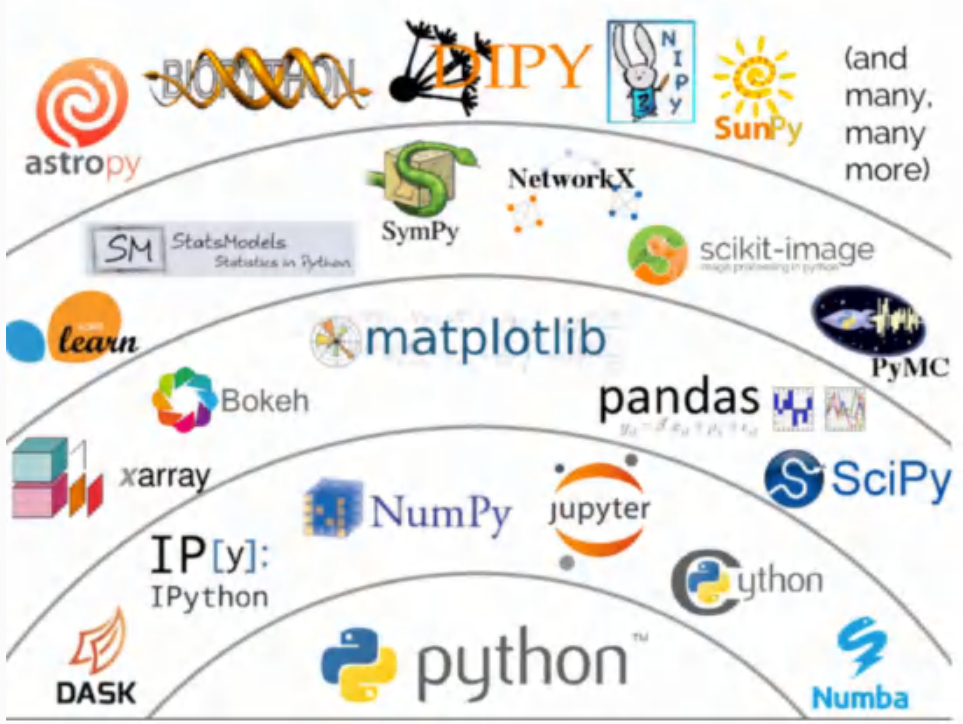
Why bother

GOTC

1. Sharing data efficiently (with “0-copy”) between libraries is easy within a single python process

```
>>> a = numpy.array([1, 2, 3])
>>> t = torch.from_numpy(a)
>>> t
tensor([ 1,  2,  3])
>>> t[0] = -1
>>> a
array([-1,  2,  3])
```

2. It is not as easy to do so across processes/runtimes on a single machine
   - Possible with plasma from Apache Arrow, a local object store using shared-memory
3. What about processing big data that cannot fit into a single machine, and involving different workloads?
   - Use vineyard + K8s!

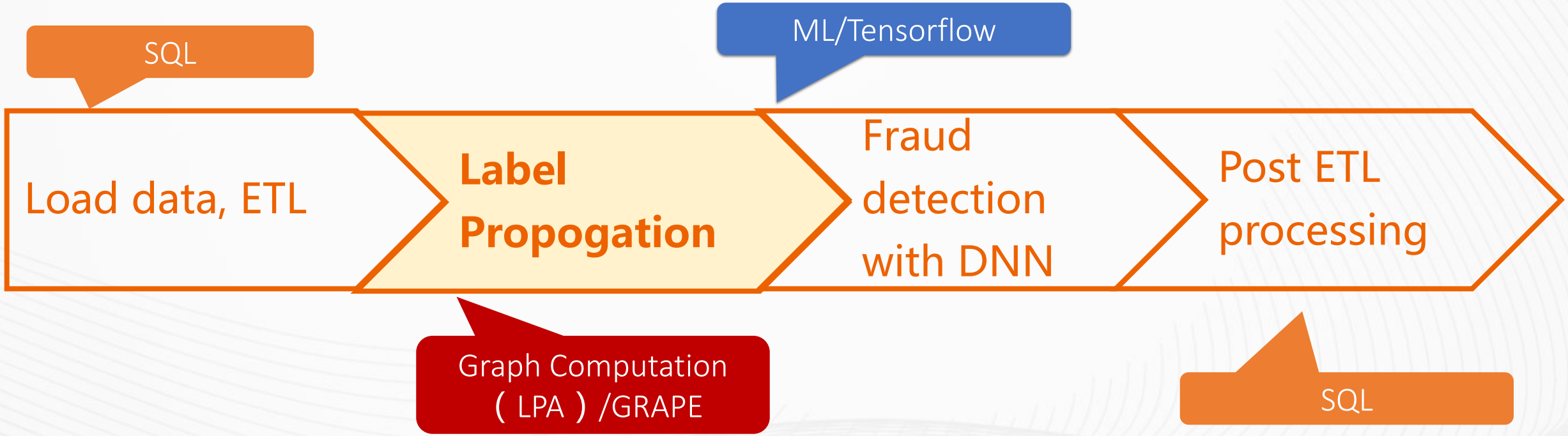


PyData is the de-facto standard for data analysis
There are lots of libraries for different workloads
(image credit: https://coiled.io/blog/pydata-dask/)

Big data analytical pipelines

GOTC

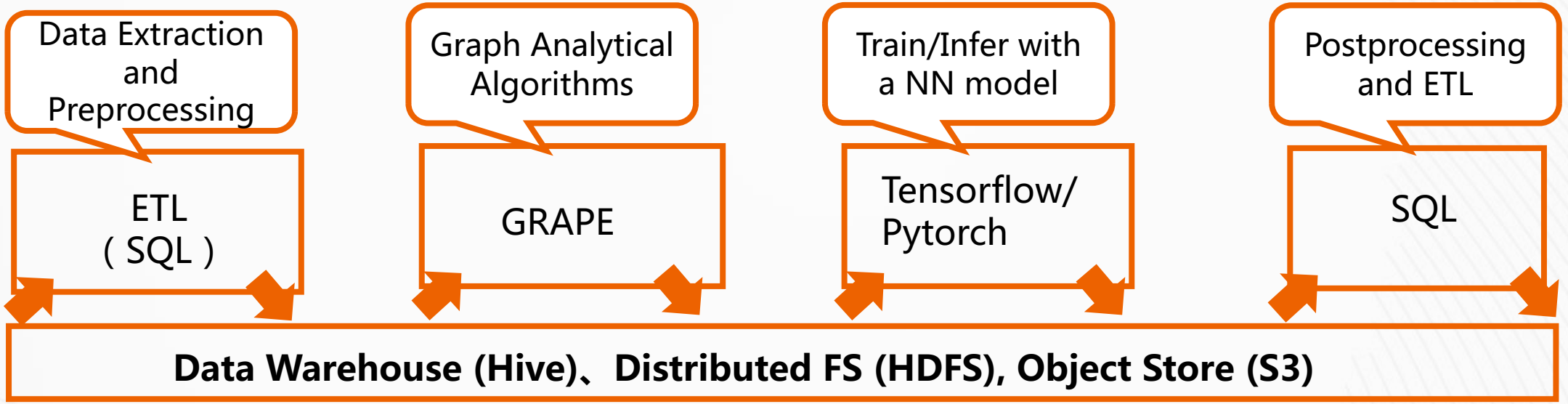
An anti-fraud pipeline



Big data analytical pipelines

GOTC

An anti-fraud pipeline



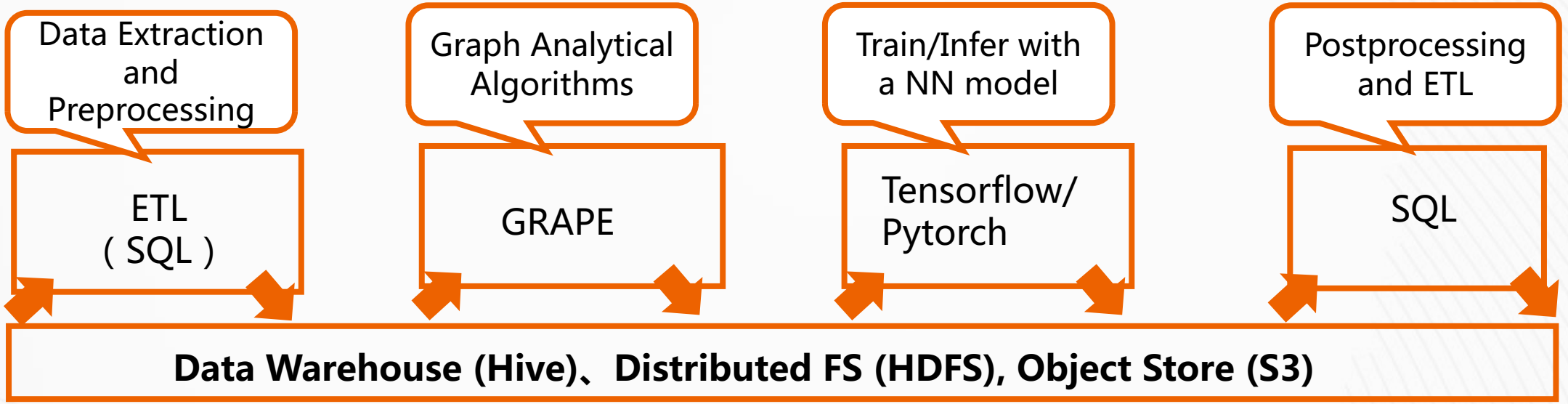
Obversation :

- A typical big data application involves various kinds of workloads, and thus involves multiple dedicated systems for each workload
- These dedicated systems typically shares intermediate data with external file systems
- The workflow is often organized as a chain/DAG, and each individual task only gets invoked after their prerequisite tasks are completed

Big data analytical pipelines

GOTC

An anti-fraud pipeline



Problem :

- Production-ready systems (Hive, Tensorflow, ...) are hard to develop ...
- Sharing data with external file systems has huge I/O cost ...
- Applying cross-task optimization (pipelining) on tasks is challenging ...

Big data analytical pipelines

GOTC

Hardness in developing production-ready systems

Problem :

- Many dedicated systems (e.g., for graph computing) are developed these years, but only a few are production-ready.
- Huge efforts are required just to implement
  - I/O adaptors
  - Data partition/chunking strategies
  - Fault-tolerance mechanisms
  - Scale in/out
  - Data sink/source

Big data analytical pipelines

GOTC

Huge I/O cost in workflows

Problem :

- Data could be polymorphic
  - Non-relational data, such as tensors, dataframes and graphs/networks are becoming increasingly prevalent.
  - Tables and SQL may not be best way to store/exchange or process them.
  - Having the data transformed from/to "tables" back and forth between different systems could be a huge overhead.
- Saving/loading the data to/from the external storage requires lots of memory-copies and IO costs.

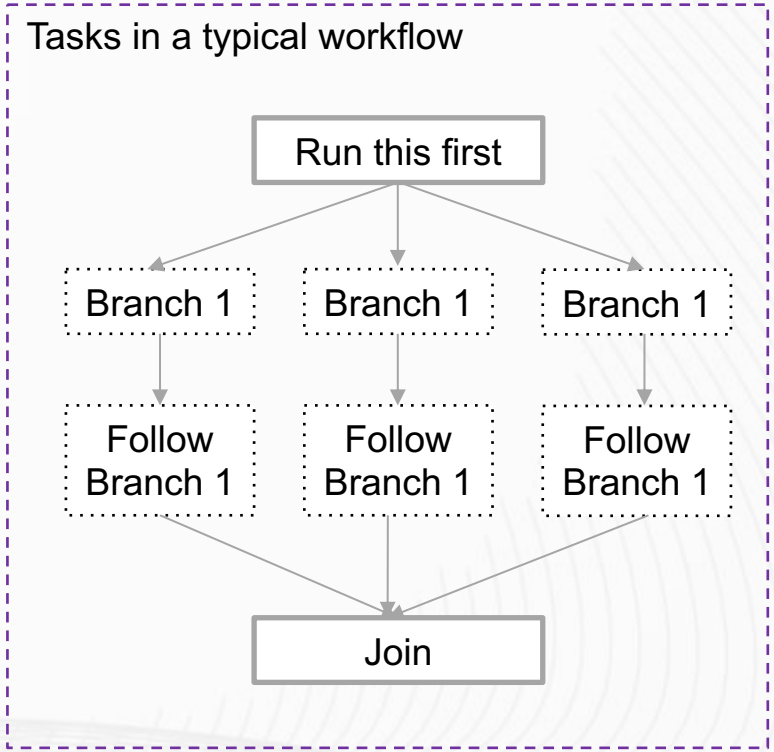
Big data analytical pipelines

GOTC

Hardness of cross-job optimization

Problem :

- Tasks in workflows has no information about other tasks
  - The immediate data cannot be placed in a optimized fashion for the dependent tasks
- The data transfer from one task to another is a barrier
  - Usually requires transformation of format and schema
  - It is hard to do cross-task pipelining



Motivation

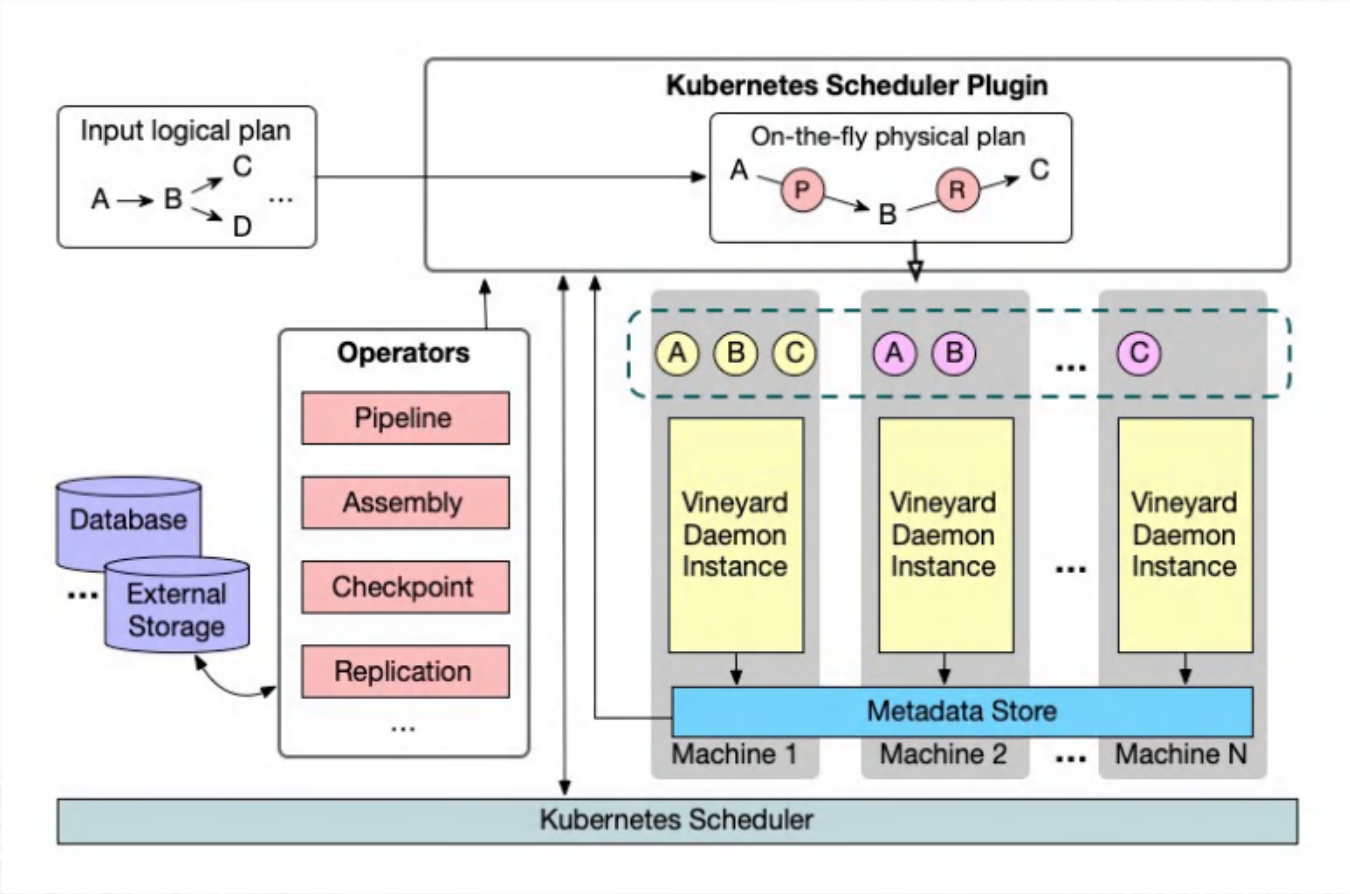
- Big data systems at production-ready quality are hard to develop
  - Vineyard has an extensible design, that supports pluggable routines for I/O, data partition, scaling and fault-tolerance
- I/O cost in workflows is usually high
  - Vineyard enables sharing in-memory immutable data in a zero-copy fashion
  - I/O flows tasks in a workflow don't require extra copy, and data can be accessed an in-memory data object.
- Cross-task optimization is challenging
  - Data in memory can be directly shared between different systems
  - Vineyard supported streams in shared memory, provides opportunity for pipelining between dedicated systems

What is Vineyard

- Distributed in-memory object store for immutable data
- Zero-copy in-memory data sharing between different systems
- Out-of-the-box high-level abstraction for developing big data applications
- Local data access as native objects
- Drivers for data partitioning, I/O, checkpointing, migration,...

Architecture

- A vineyard object consists of data payload and metadata
  - Data payload is storing in shared memory
  - Metadata is synced through the cluster with ETCD
- Vineyard daemon instances are accessed via IPC/RPC connections
  - Data payload can only be accessed by IPC connections
- Pluggable drivers can provide certain functionalities to certain data formats

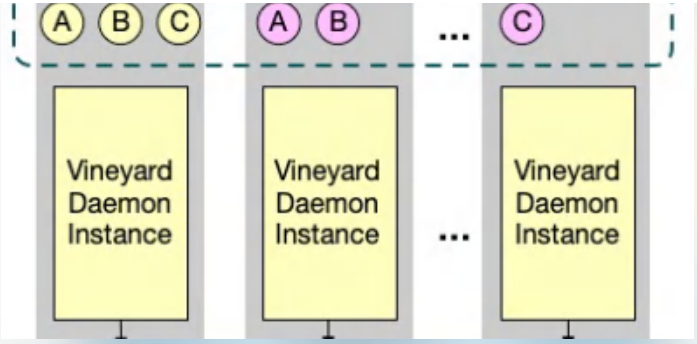


Vineyard

Efficient Object Sharing across Engines

- Object = Metadata + {Blob}
  - Decouple the *payload* and *semantics*
- Share by memory mapping
  - *Zero-copy*
- Share with the data structure abstractions
  - Shares the *data structure* directly
  - e.g., Tensors, DataFrames, Graphs
- Builders + Resolver
  - Interpret the vineyard objects to engine's *native* value type

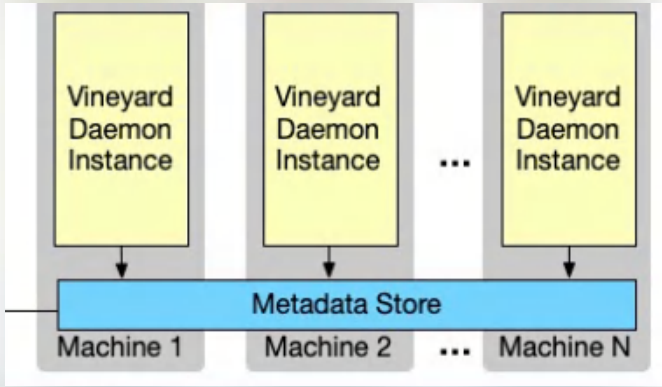
```
{
    'id': 'o0.....',
    'typename': 'vineyard::DataFrame',
    'columns_0_': 'a',
    'values_0_': {
        'id': 'o0....',
        ...                                  ard::Tensor<...>',
        'buffer': {
            'id': 'o8...',                                    --- blob
        },
    },
    ...                                                       --- blob
    ...                                                       --- blob
    ...
}
```



Distributed Objects Sharing

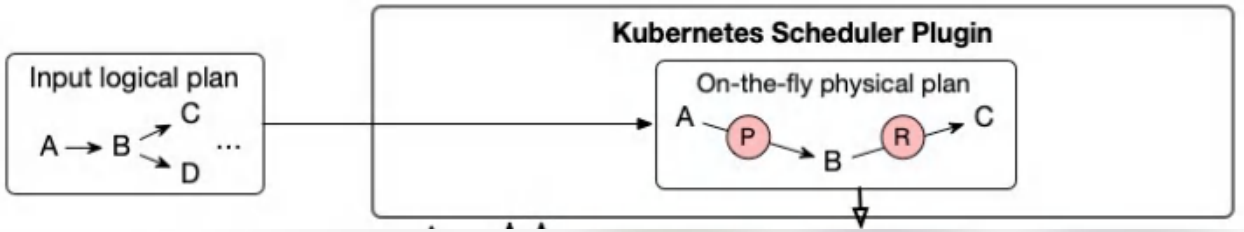
- Vineyard support distributed objects
  - A global object consists of a set of chunks
  - A client can accessing payload of local chunks
  - and metadata (only) of remote chunks

- Metadata is synced using *etcd*
  - Performance: only metadata of objects that are *refered* by a global object are synced to other instances



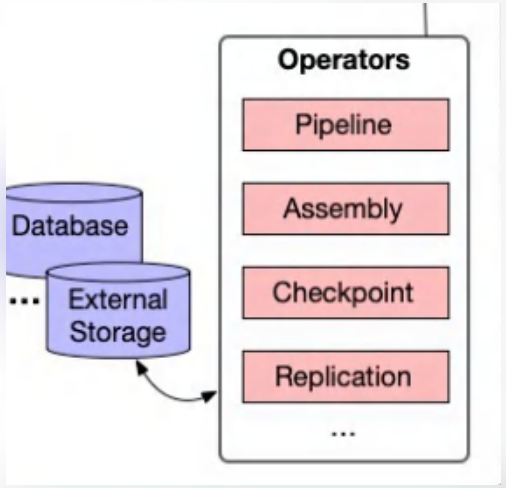
Pipelining between tasks in a workflow

- Zero-copy sharing unlocks new opportunity
  - The intermediate data sharing is not a barrier anymore
- Stream in Vineyard
  - Stream over chunks of *data structures*
  - e.g, tensor stream, dataframe stream
- Tasks can be pipelined using vineyard stream!



Pluggable drivers

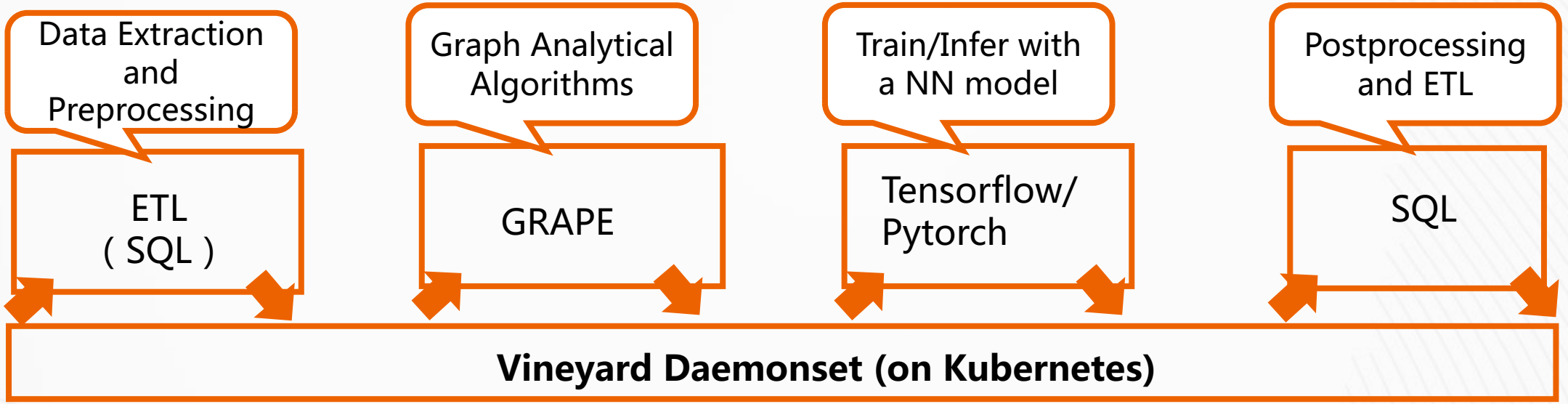
- Engines usually are hard to be connected to production systems
  - Integration with *internal I/O*
  - Integration with *other internal engines*
- Vineyard serves as a *bridge*
  - I/O is delegated to vineyard
  - Engines consume *data structures* in vineyard directly
  - Engine talks to other engines via shared intermediate objects in vineyard



Vineyard on Kubernetes

GOTC

Vision: a new cloud-native paradigm for bigdata tasks



The end-to-end big data task is deployed on Kubernetes

- Intermediate data is abstracted as a Kubernetes resource(*CRDs*), and is sharing with vineyard through memory mapping
- "Data" lives *in memory*, and the scheduler optimizes the data flow among cluster nodes

Vineyard on Kubernetes

GOTC

Memory Sharing on Kubernetes

- Vineyard requires IPC communication between vineyard server pods and application pods for memory sharing

- The domain socket of vineyard server could be mounted on *hostPath* or *PersistentVolumeClaim*

- When users bundle vineyard and the workload to the same pod, the domain socket could be shared using an *emptyDir*

Vineyard on Kubernetes

GOTC

Vineyard objects as Kubernetes resources (CRDs)

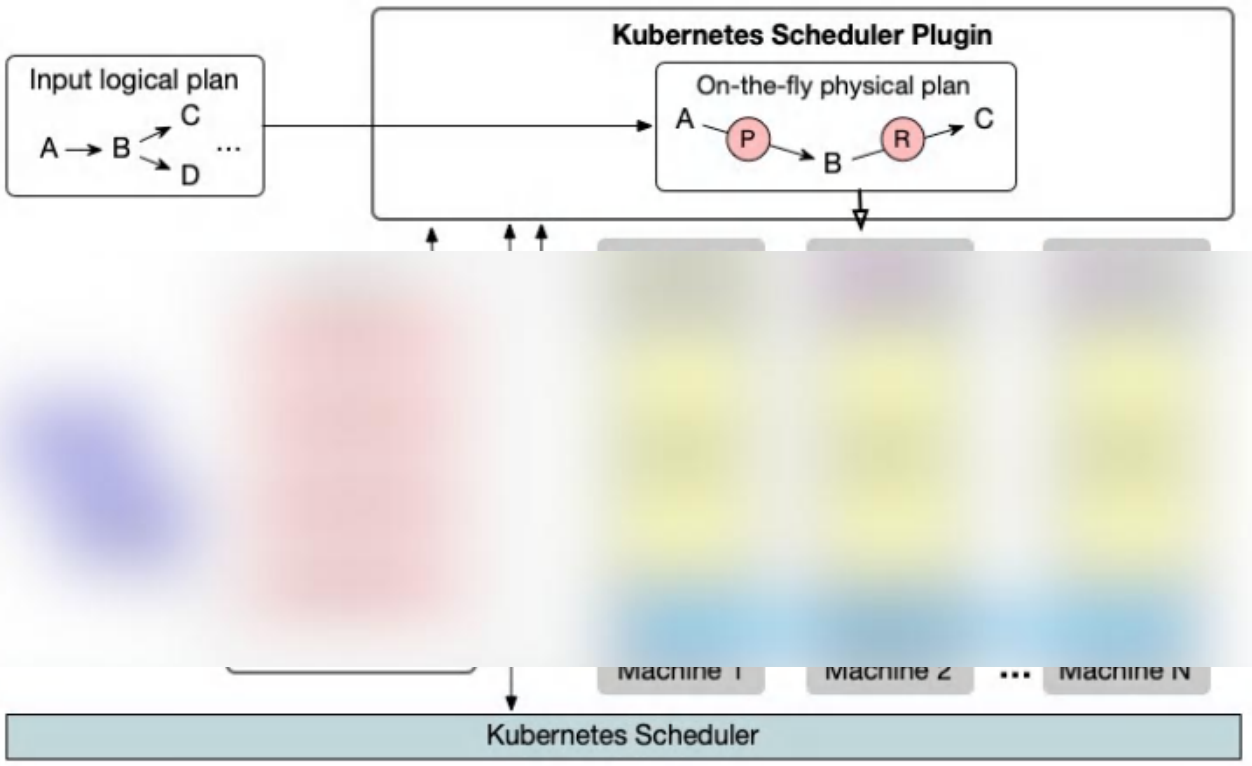
- Vineyard objects are abstracted as Kubernetes resources (i.e., CRDs)
- Each CRD contains the metadata of the represented vineyard object
- Location specs that describe which node an object is located are added to the CRDs of local objects

Vineyard on Kubernetes

GOTC

Scheduling on Kubernetes

- Job and its required data cannot be always aligned
  - The cluster environment is dynamic and constrained
  - The requirements of different workloads is different
- The location information can be used to guide the scheduling process:
  - A *vineyard scheduler plugin*!
- It still can be unaligned
  - Auto migration in *intiContainer*



Vineyard on Kubernetes

GOTC

Deploying Vineyard on Kubernetes

- Deployment
  - Vineyard is deployed as a DaemonSet in Kubernetes cluster

- Deploy using Helm
  - Vineyard can be easily deployed in Kubernetes cluster using Helm:

```
helm repo add vineyard https://dl.bintray.com/libvineyard/charts/

helm install --namespace vineyard--name vineyard stable/vineyard
```

Roadmap

GOTC

Onging

- Connecting to machine learning frameworks
  - Integration with Tensorflow/Pytorch to share objects in vineyard to machine learning frameworks
- SDK in more languages
  - Python
  - Java
  - Rust
  - Go
- Integration with workflow engines
  - Integration with airflow: brings better immediate data sharing solution for workflows orchestrated by airflow

Roadmap

GOTC

Further ahead

- Vineyard Operator for Kubernetes
  - Better cluster management and monitor on Kubernetes cluster
  - Better data-aware scheduler policy within the scheduler plugin
- Application-aware Far Memory
  - Vineyard supports global object abstractions, e.g,, GlobalDataFrame
  - Support for application-aware far memory will enables single-machine applications to leverage remote memory resources
  - Better performance than raw RPC
- Storage hierarchy
  - In-memory objects can be swapped out in certain cases
  - Snapshot the objects and restore back to memory later benefits the end-to-end performance

Vineyard Community

GOTC

Opensource

- Vineyard is open source under the Apache-2.0 License

- Any contribution from the community are welcomed
  - Issues about bugs and feature requests
  - Pull requests for bugfix, enhancement, feature implementation and extensions
  - Discussion about the installation, deployment, usage of vineyard

- We have comprehensive documentation for the underlying design and how to build application on vineyard
  - https://v6d.io/

加入我们

一起构建 Vineyard

https://v6d.io

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

GOTC

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

Vineyard