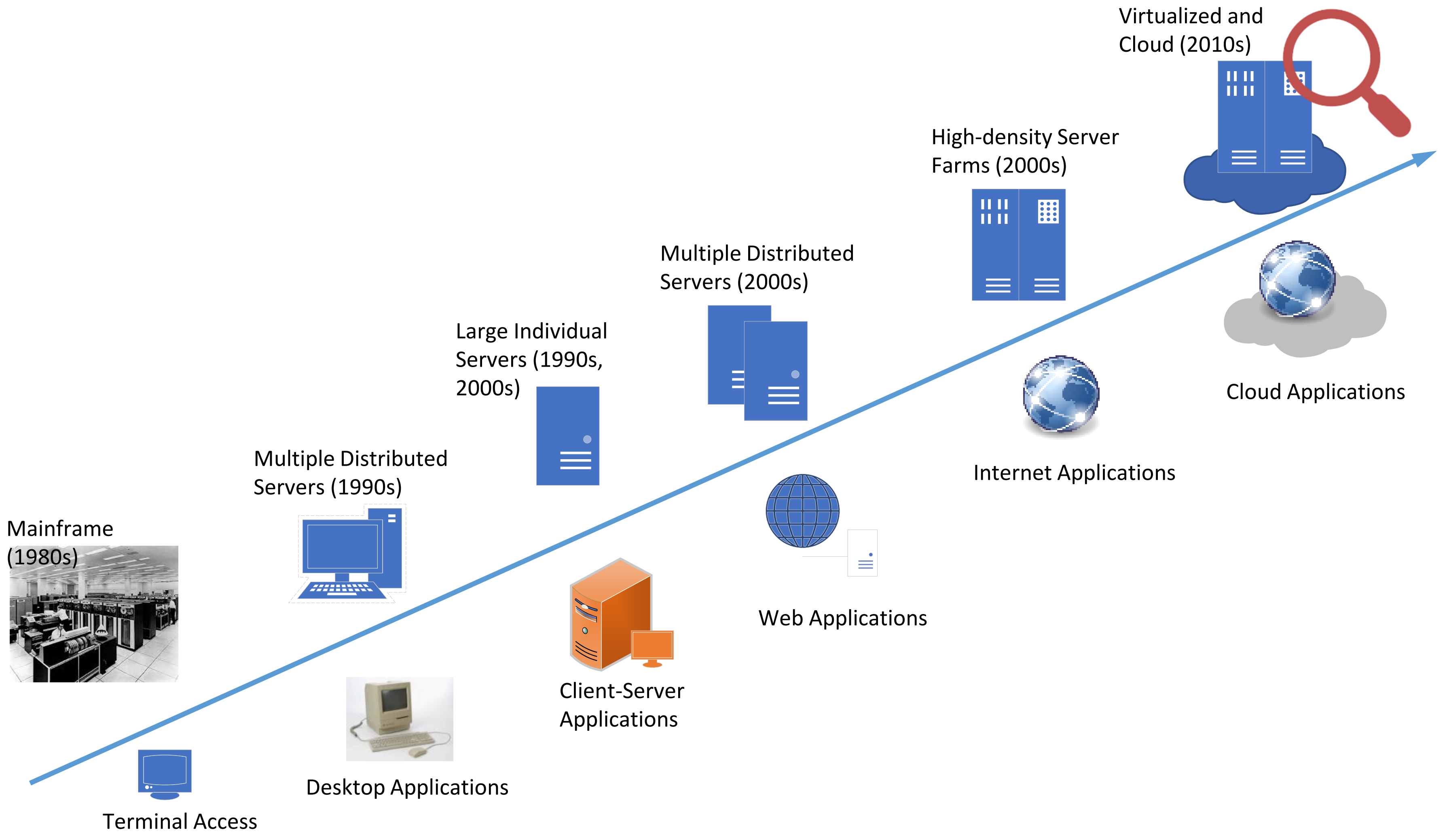
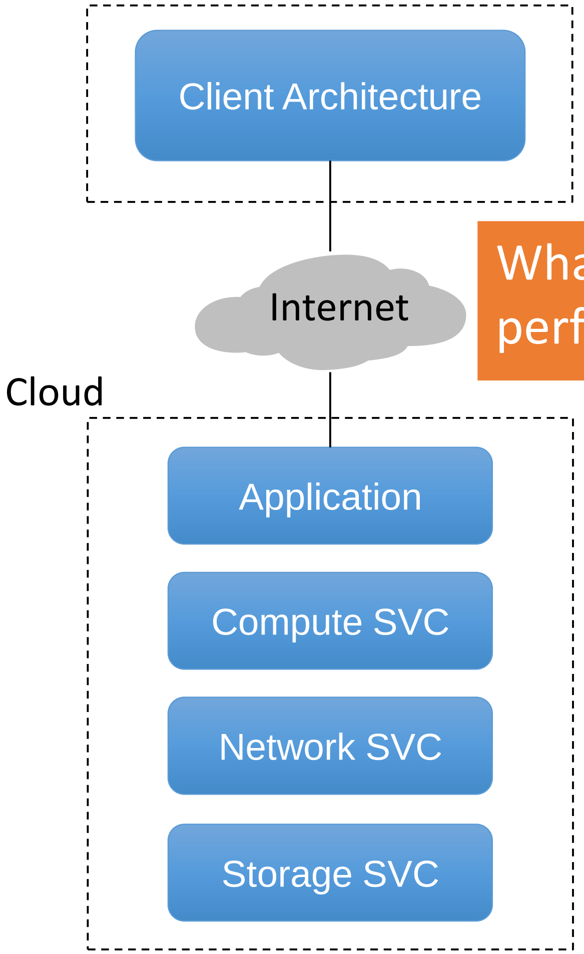
# Overview- Big Data Applications VM and Container

Csci 5980- Spring 2020

# Evolving Applications and Infrastructures



# A Look at Virtualized and Cloud Infrastructure




What's the impact on data access performance?

Computation:

- Powerful Units
- Large Scale
- Virtualized (VM)
- Containerized

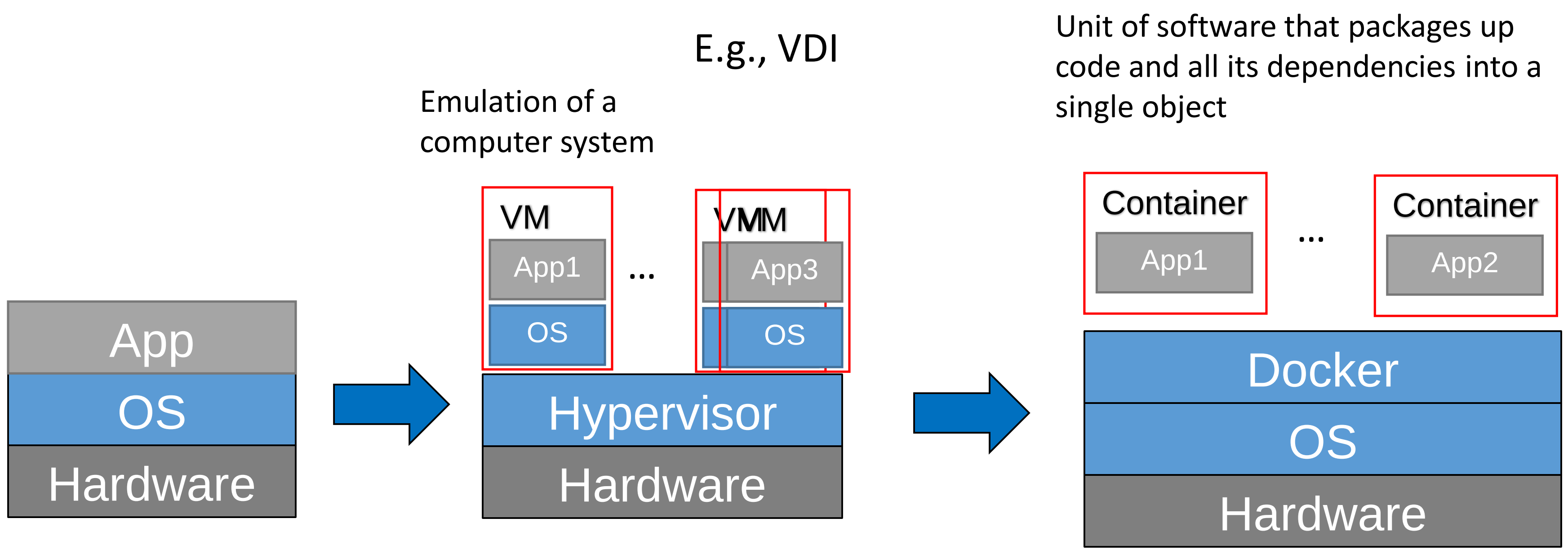
Network:

- Large (10K-100K switches)
- Software Defined
- On I/O path

Storage:

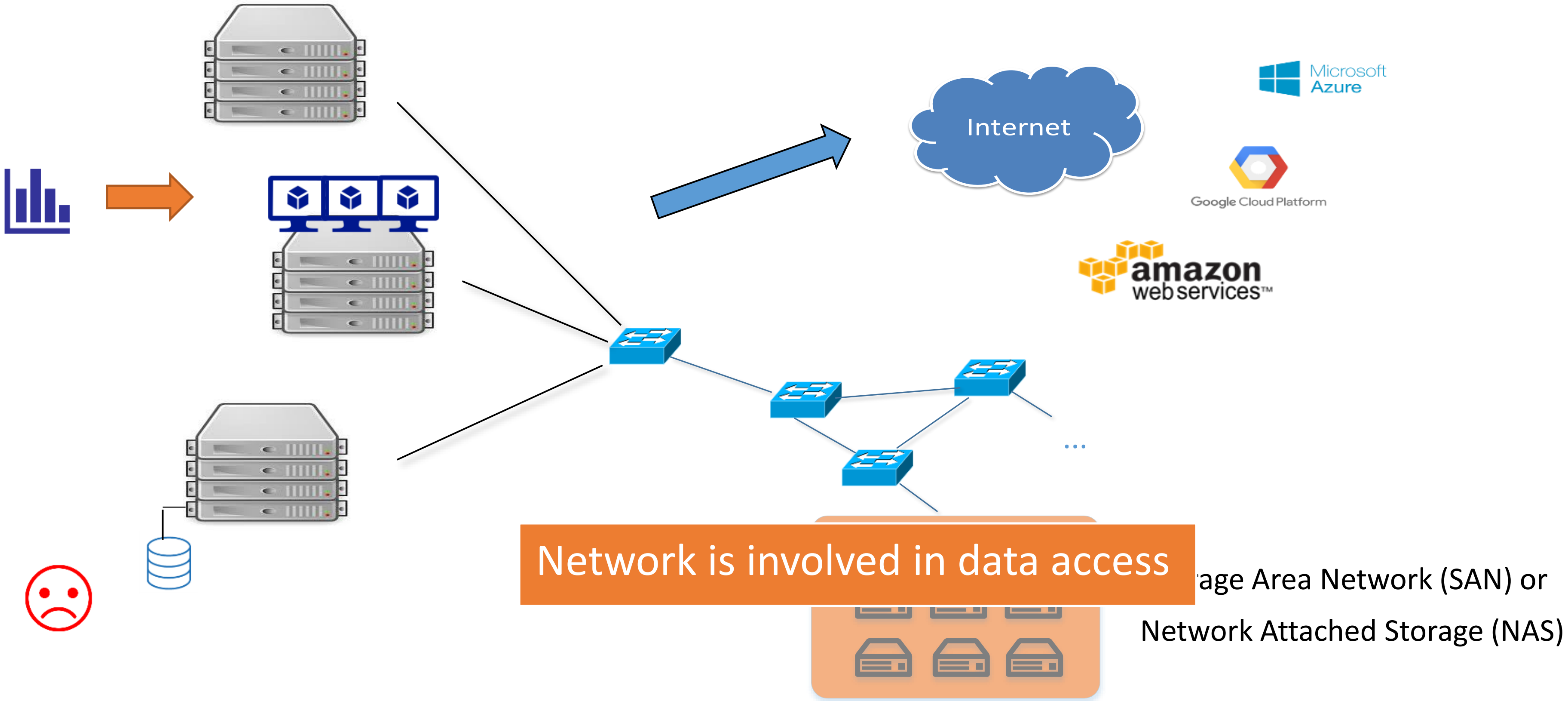
- Heterogeneous (HDD,SSD,SMR)
- High capacity
- Distributed

# Virtualization and Containerization




Virtualization: more and more lightweight

# Network in Storage



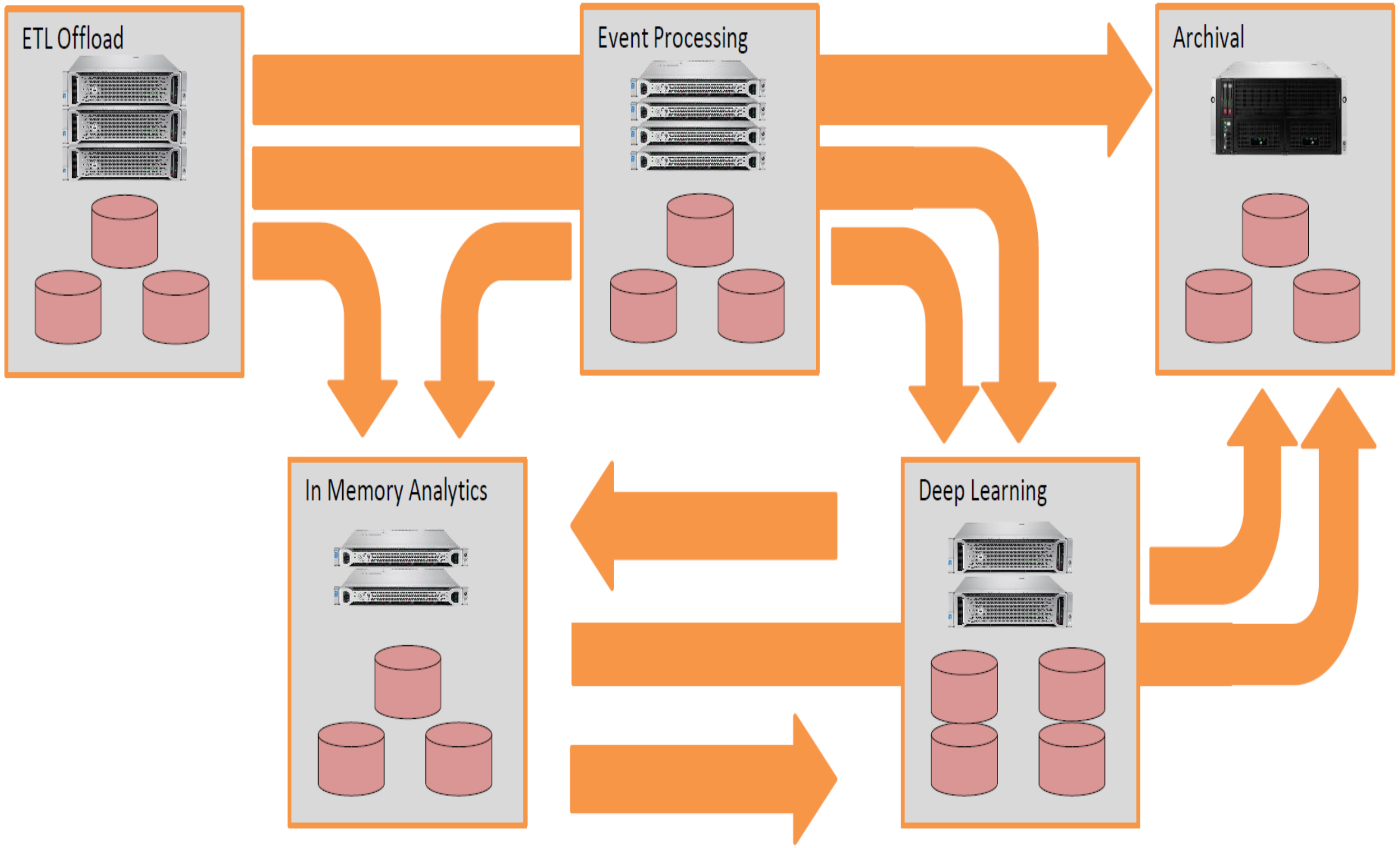
# Impact to Data Access Performance

- Data access in VM
  - Applications run in VMs. Data are stored in data center.
  - People can access data from anywhere at anytime.
  - How are storage allocated?
  - What are the storage requirements for such applications?
- Data access in Docker container
  - What is the current storage support for containerized applications?
  - How to allocate storage & manage storage based on users’ requirements?
- Data access over network
  - The dynamic network results in long I/O path and increased end-to-end management complexity.
  - A systematic view of client, network and storage is essential to improve data access performance.

# Hyperconverged Infrastructure

# A Typical Data Journey

- Data collected & transformed to different formats & offloaded to large scale distributed storage systems
- Simultaneously, through IoT and other event monitoring capabilities, collected data & real-time streamed data based on current events will be delivered to a large memory-based computing system to be analyzed (in-memory processing).
- Deep learning based AI & machine learning approaches will assist data analytics to support optimal decisions
- The original data as well as the analytic results are to be archived for future uses



# IT Infrastructure is Transforming

Goal: Data Processing → Information Retrieval → Knowledge Generation & Decision Making

+

White-Box Effect (Learned from Cloud Computing)

+

Open Source Effect

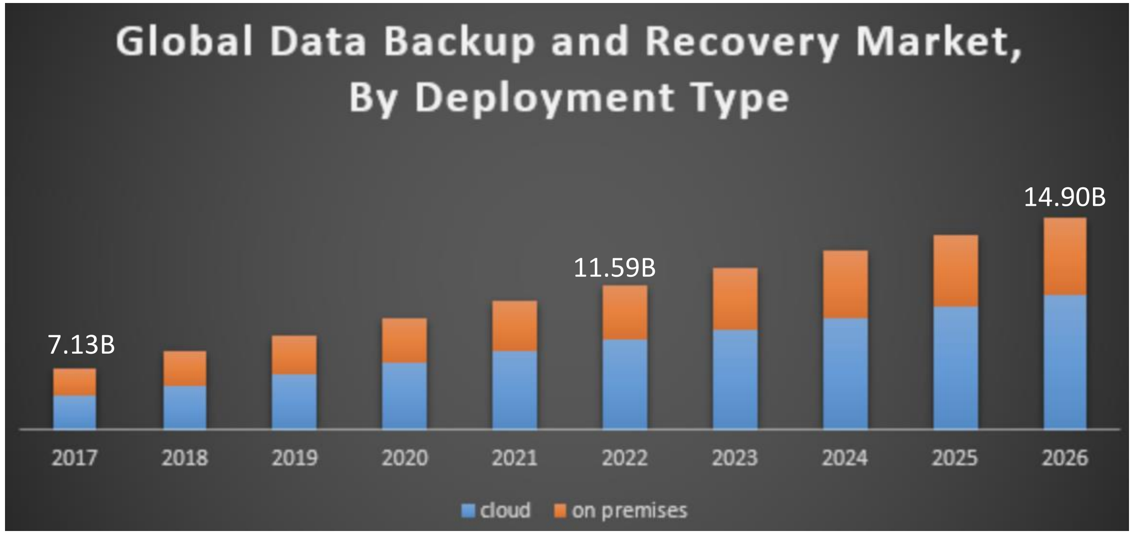
# Hyperconverged Infrastructure: Seamless integration of compute, network & storage in a distributed environment like the Internet

- We believe hyperconverged infrastructure (HI) is promising for the future Internet.
- In a hypercoverged infrastructure compute, storage and network are consolidated and fully integrated to support big data applications with increased efficiency, broad scalability, improved agility and reduced costs.
- Although hyperconvergence enables us to investigate the interactions between compute, network & storage, to realize all benefits, we need to leverage technology improvements of each component:
  - New architectures, Non-Volatile memory, VM & Containers for server compute.
  - Development of new optical networks, 5G cellular system, NFV (Network Functional Virtualization) & software-defined network for switches & routers.
  - Software-defined Storage, I/O stack revamping, multi-tier storage, long-term data preservation

# Data Deduplication

# Backup and Data Deduplication

Source: https://www.channelfutures.com/uncategorized/file-based-image-based-backup-selling-the-differences

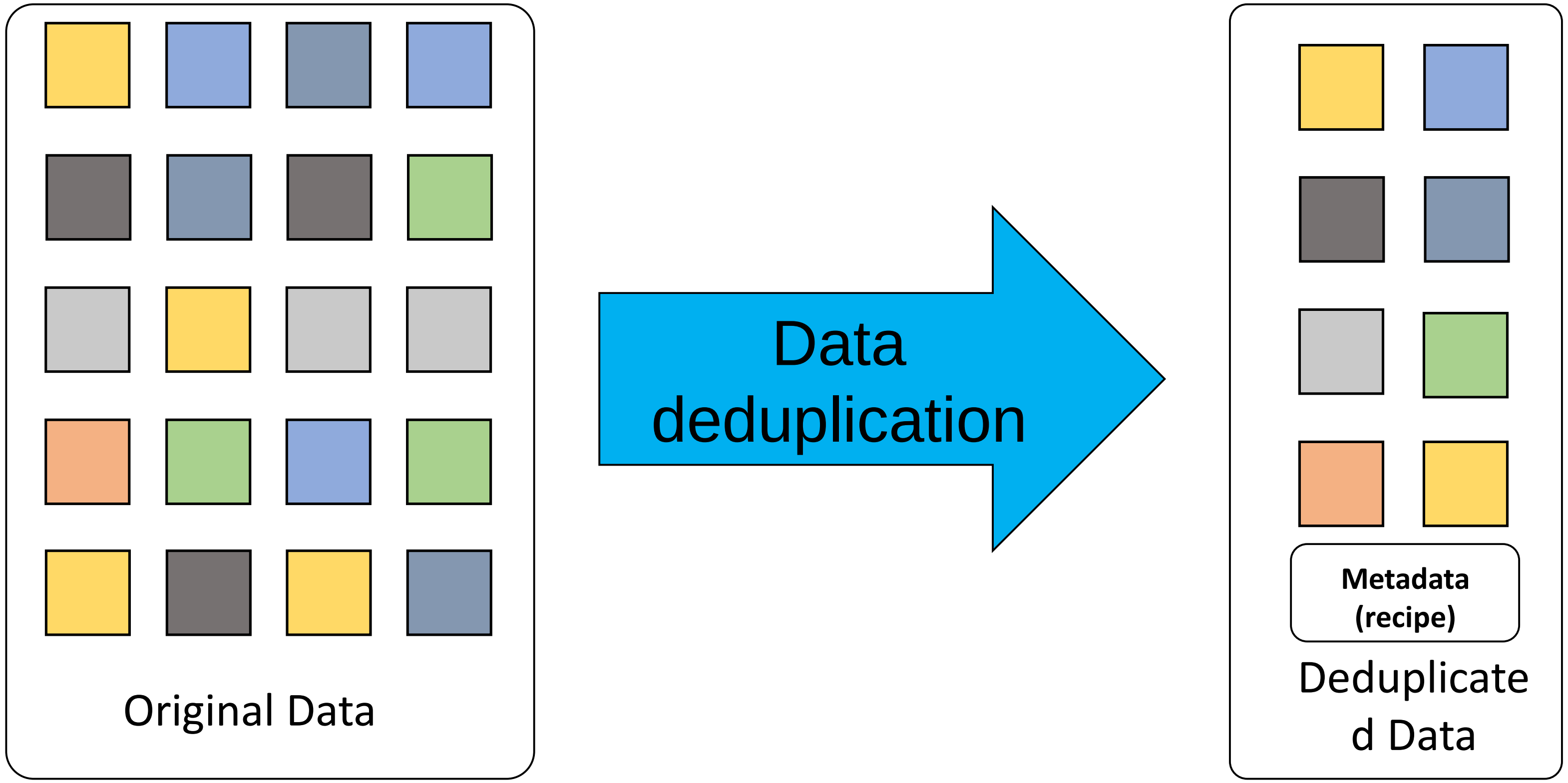



Source: https://www.maximizemarketresearch.com/market-report/data-backup-recovery-market/875/

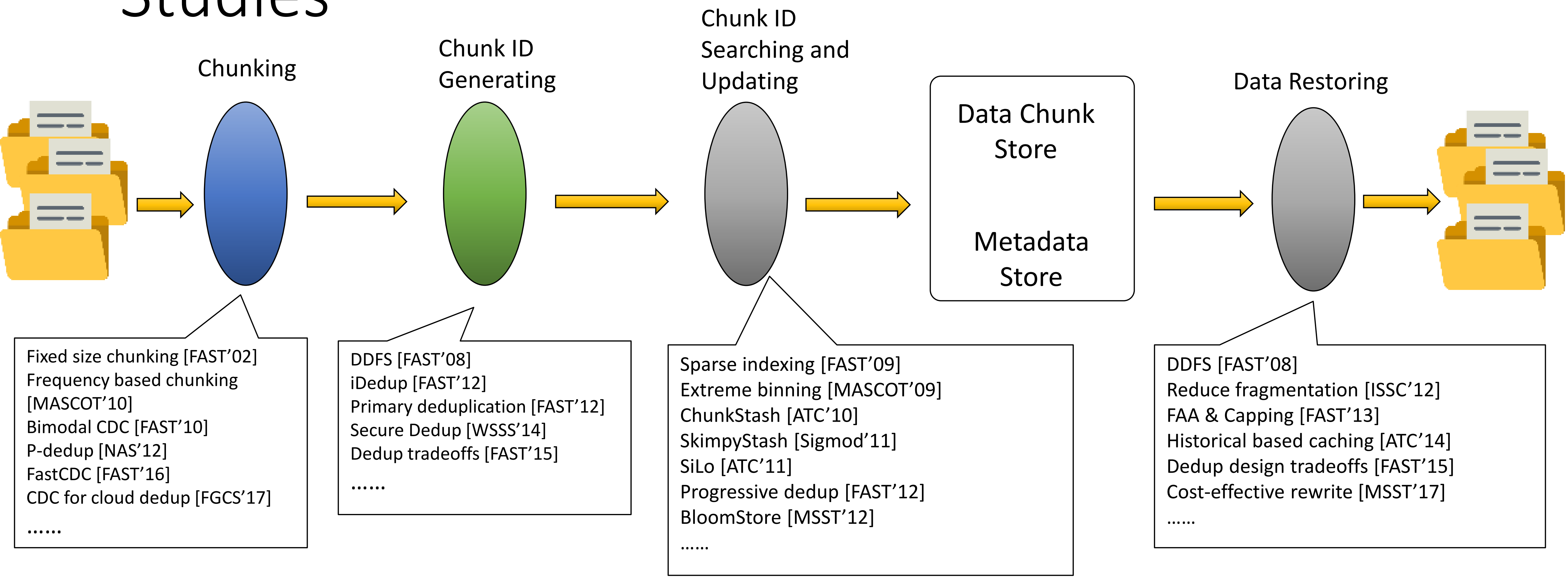
- Data deduplication is a very important technique in backup systems to efficiently reduce storage space utilization
- Due to the data content duplicates, a large portion of the data in different backup versions from the same backup source are the same. It is also true for data from different source (e.g., VM backup).
- After deduplication, some backup products can achieve 90% or even 95% more space saving

# What Is Data Deduplication?

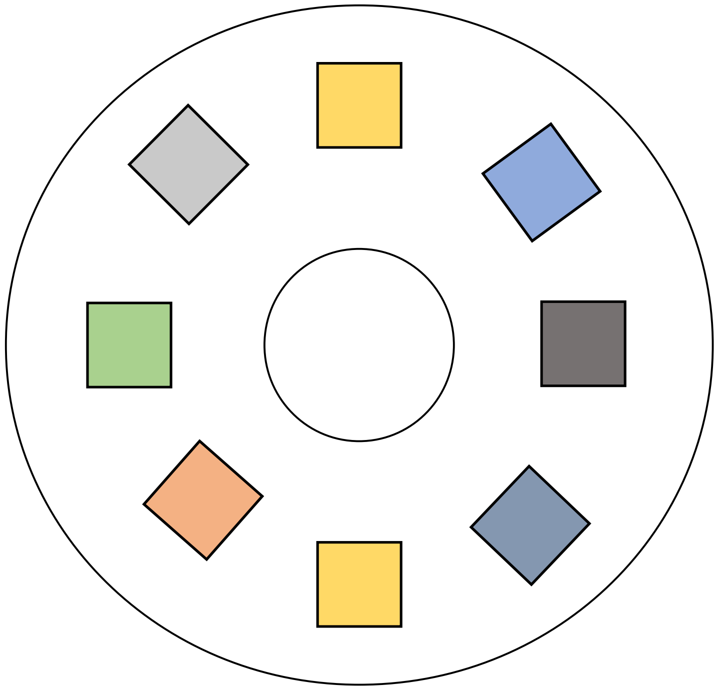
Data deduplication is a process to eliminate the redundant data content. Different from data compression (bytes level), data deduplication reduce the **block/chunk/file level** duplicates



# Data Deduplication/Restore and Related Studies



# Why Improving Restore Performance Is Important?

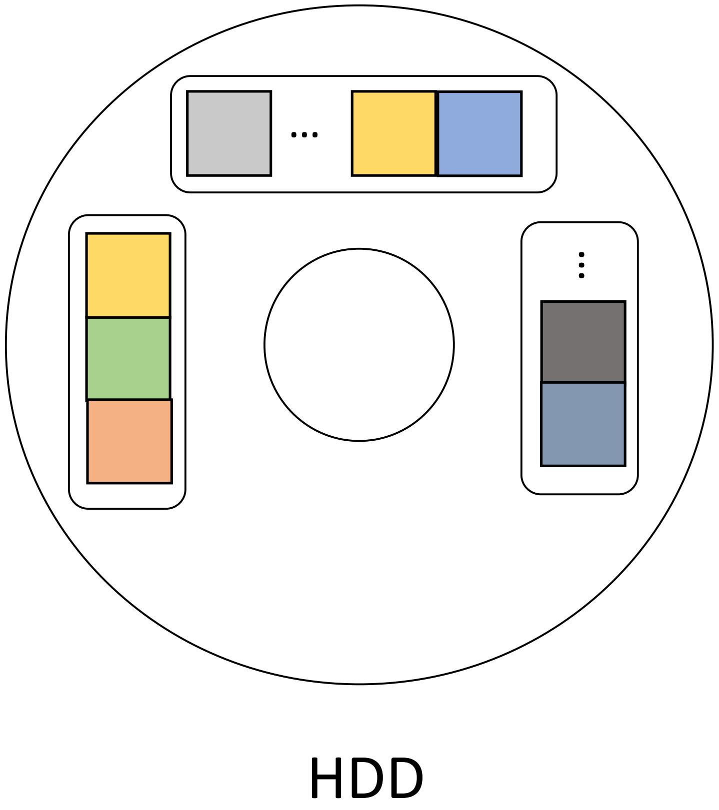


HDD

**Chunk-based I/O**

- After deduplication, the data chunks of original data are scattered in the whole storage system **[high data fragmentation]**
- Reads and writes consume high seeking time [**low read and write efficiency**]

# Why Improving Restore Performance Is Important?




**Chunk-based I/O**

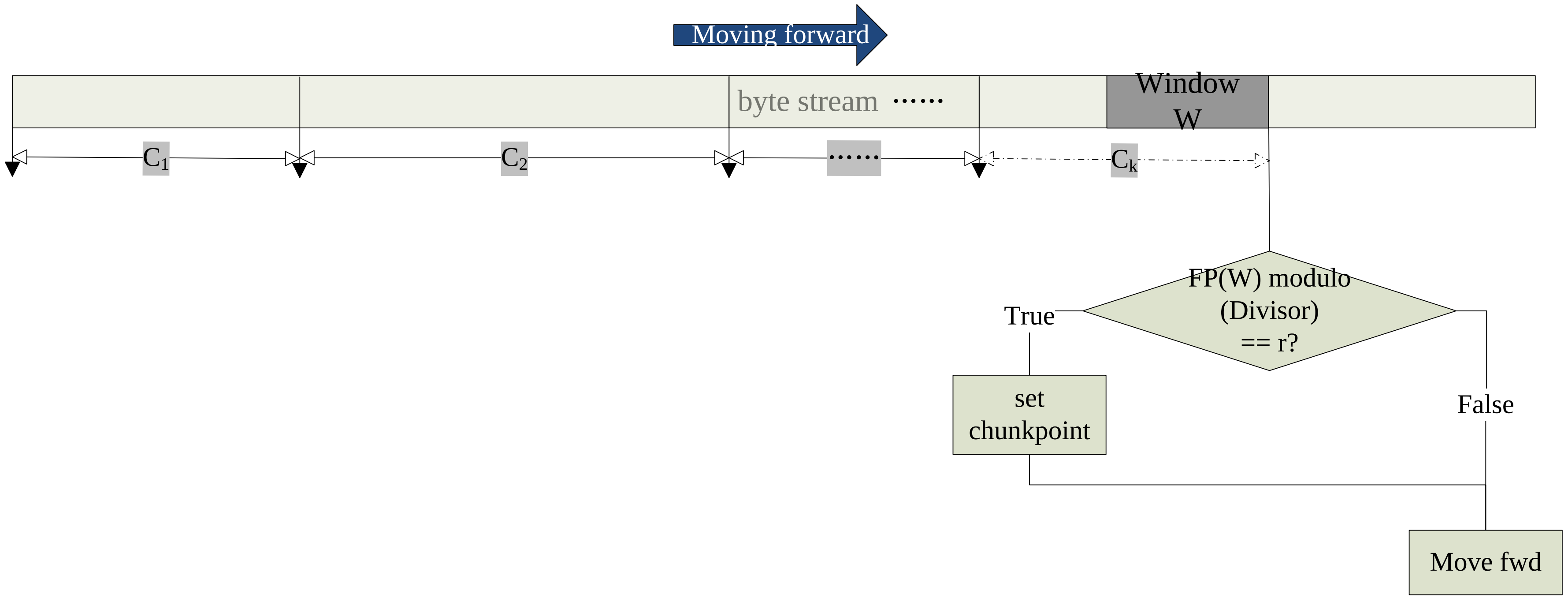
- After deduplication, the data chunks of original data are scattered in the whole storage system **[high data fragmentation]**
- Reads and writes consume high seeking time [**low read and write efficiency**]

**Container-based I/O**

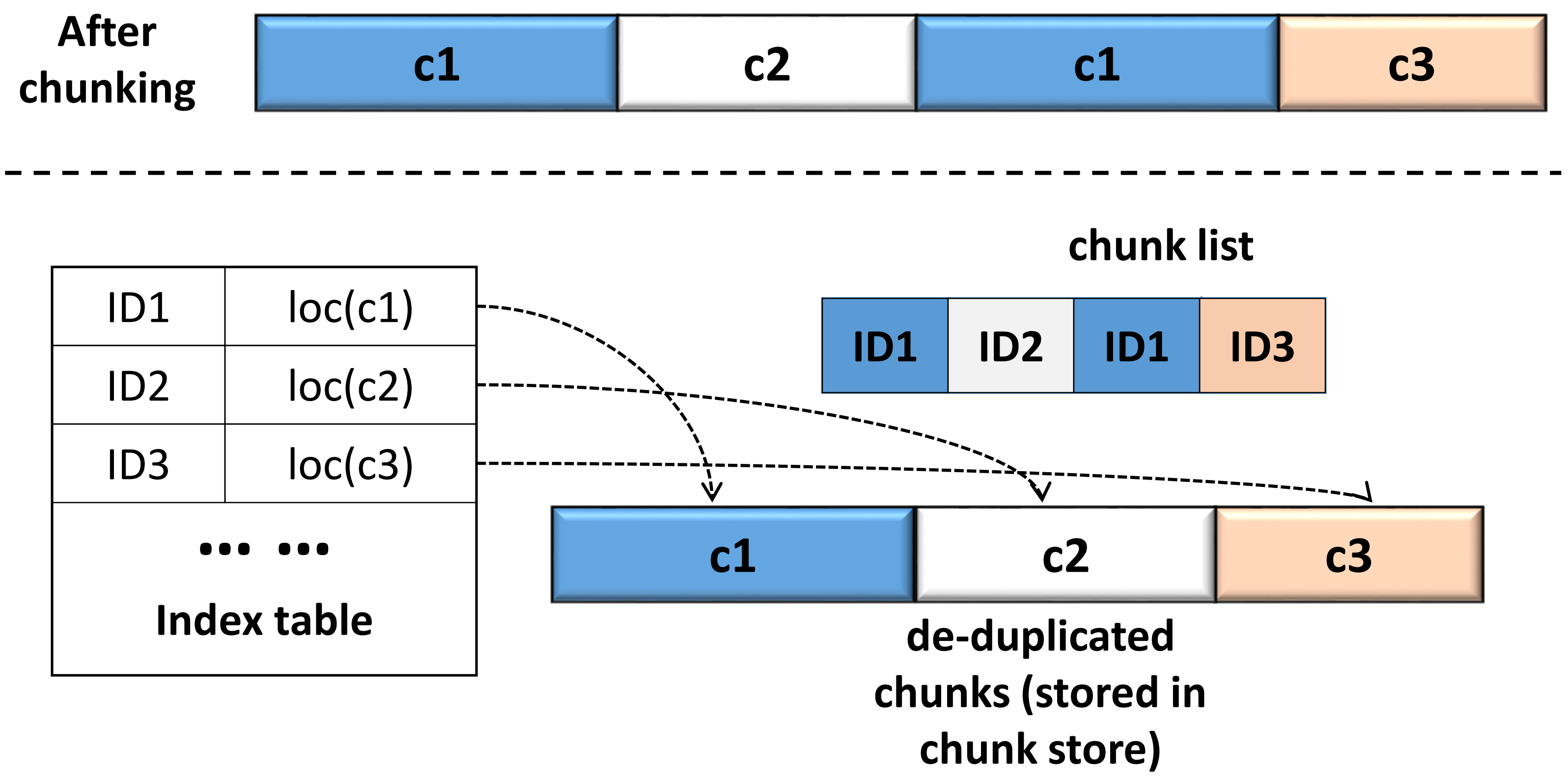
- After deduplication, the data chunks of original data are scattered in the whole storage system **[high data fragmentation]**
- When one or a small number of chunks are needed in one container, the whole container needs to be read out [**read amplification**]

# Overview of Chunking Algorithms

- Fixed-sized Chunking
- Content-Defined Chunking



# Data Structures Associated with Chunking Deduplication



# Dedupe Research Topics

- Read performance optimization
- Dedupe reliability
- Dedupe for checkpointing
- Scalable VM cloud storage
- Emerging storage hierarchy
- Checkpoint storage for exascale computing

# I/O Access Hints and Multi-Storage Pools

# Legacy I/O Stack w/ I/O Access Hints

## ❑Legacy I/O stack problems

- To adapt HDD, big performance gap (HDD vs. memory)
- Enterprise storage system=> multiple apps, parallel I/Os
- Many layers without proper coordination (app, vfs, fs, lvm…)
- Homogeneous fixed-size logical block address

## ❑I/O Access Hints in Hybrid Storage Systems

- A piece of tiny but useful information on top of block storage (e.g. stream ID, file metadata)
- Data management across diverse devices (data migration, data placement, space allocation, etc)
- Not like page level management (fadvise(), ionice())

# The Challenges of I/O Access Hints

- Industry (e.g.Intel, NetApp) has several standardization proposals based on T10/T13 without real outcome

  - Many stakeholders

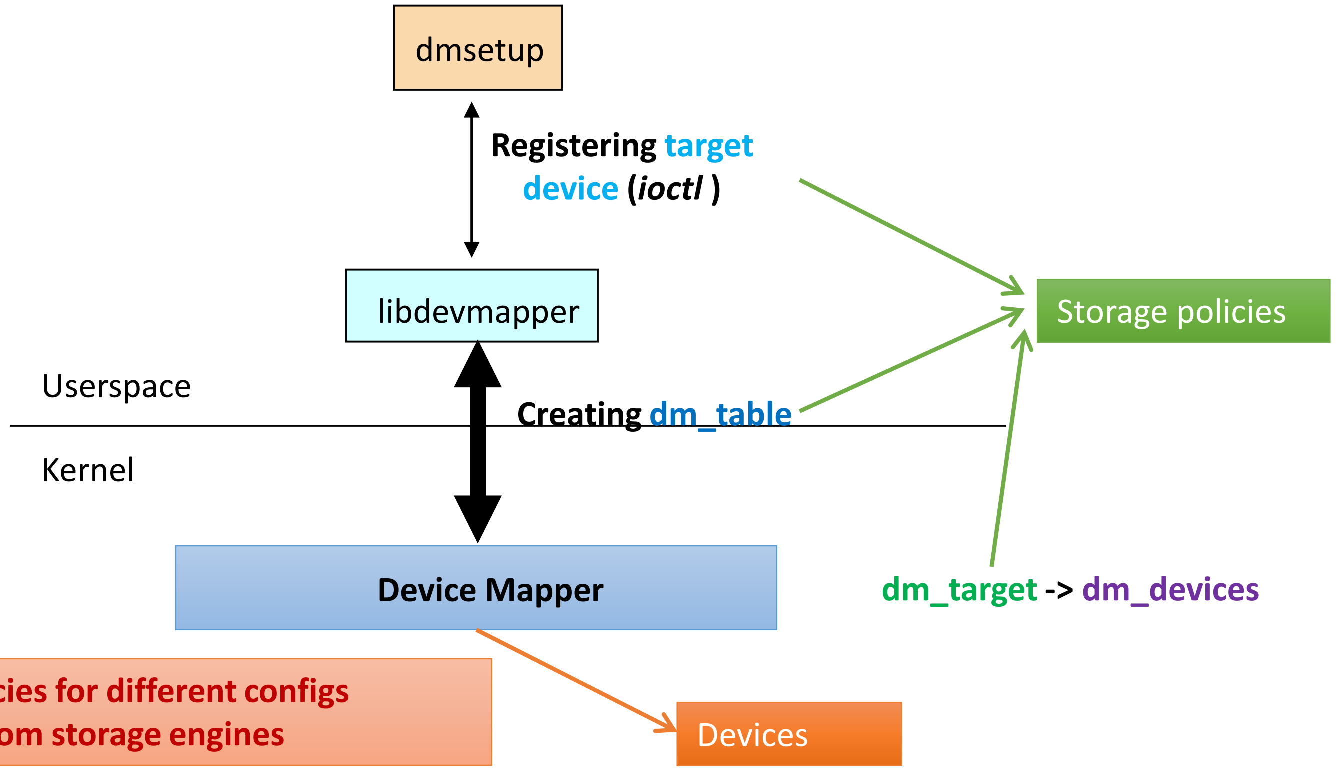
- To add and apply hints, different layers may require tedious modifications

  - Kernel level modification (block level management, file systems)

  - May invo

**Goal of HintStor => A flexible framework to study I/O access hints in heterogenous storage systems**

# Device Mapper in HintStor




1. Separate storage policies for different configs
2. Separate interfaces from storage engines

# Prerequisite of HintStor

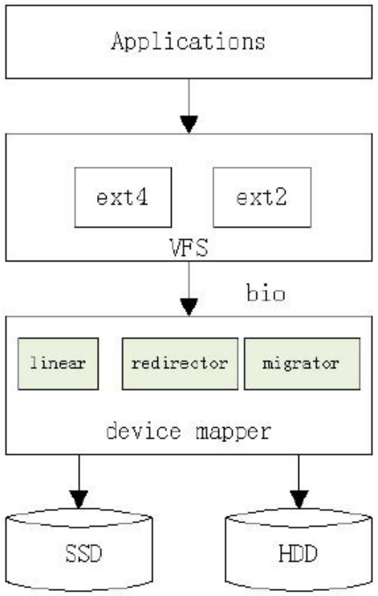
Two new drivers in Device Mapper

❑Redirector

The target device (bio->bdev) can be reset to the desired device

❑Migrator

Using the “kcopyd” policy to copy a fixed-size chunk (a set of blocks) from one device to another device




- 600~ LoC C code in Linux kernel

# Block Storage Data Manager

- Fixed-size chunk mapping table (1MB or more)

- Chunk-level I/O analyzer

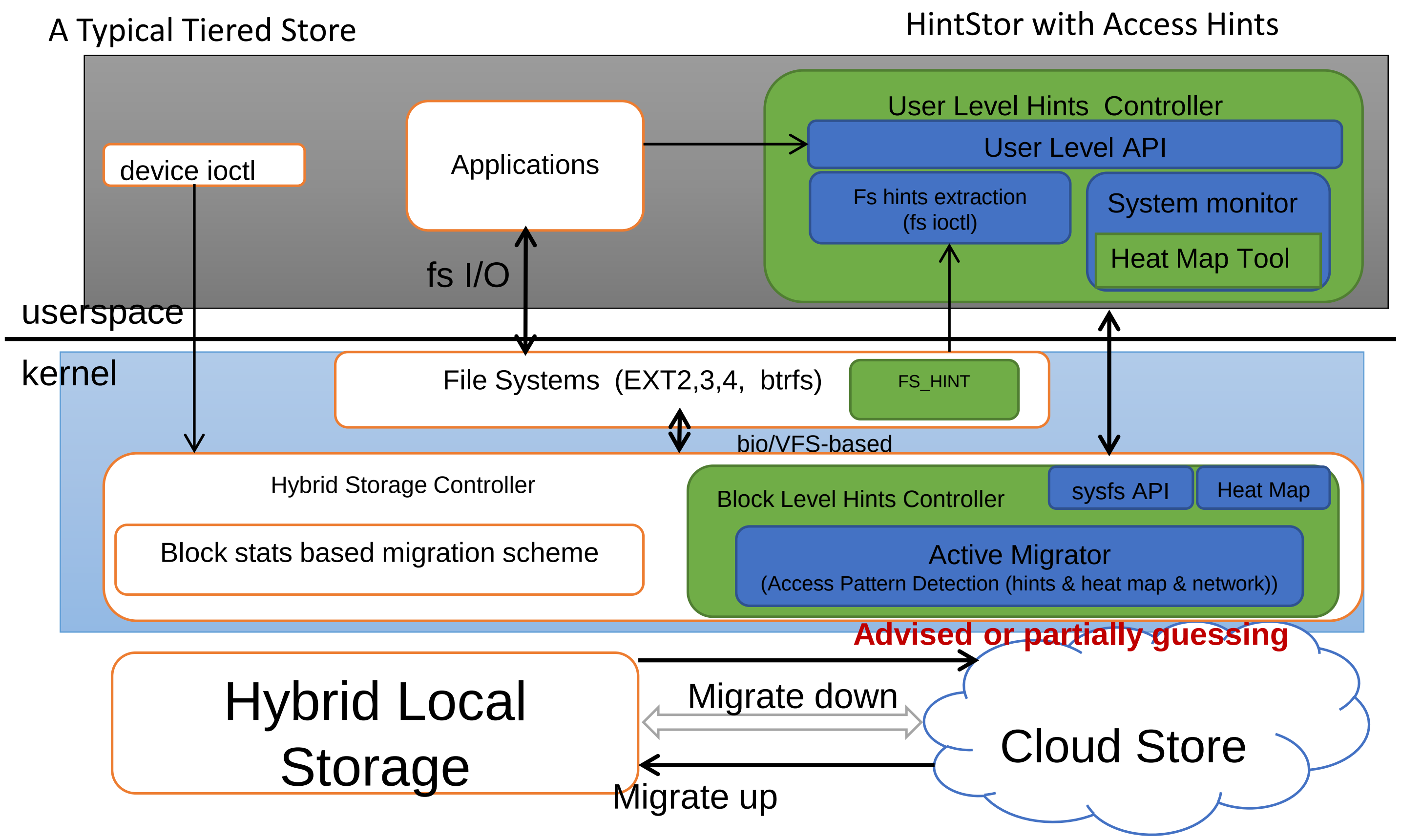
  - Monitor
  - Heatmap using Perl scripts

- Access hints atomic operations *(op, chunk id, src addr, dest addr)*

  - REDIRECT
  - MIGRATE
  - PREFETCH
  - REPLICATE

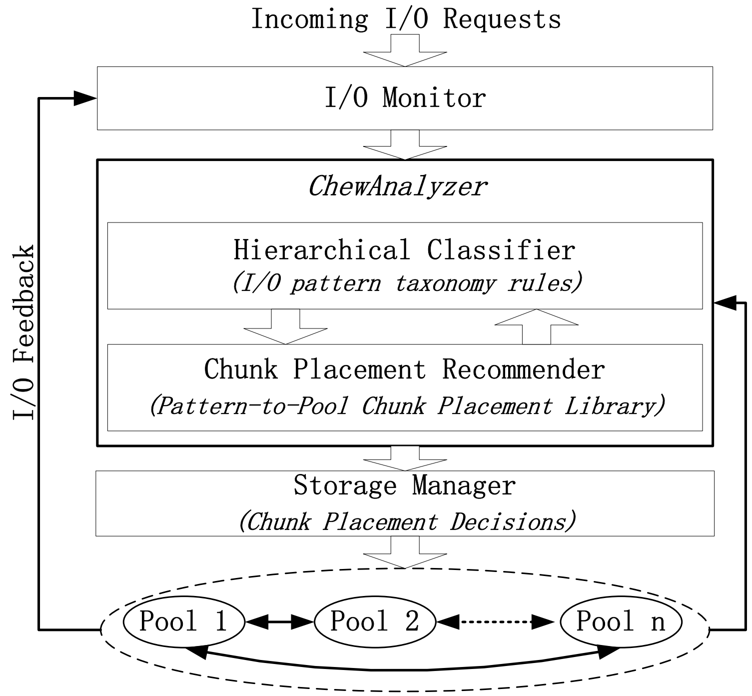
# HintStor Framework

- Prototyping in Ubuntu 14.04 (Kernel version, 3.13.0 )



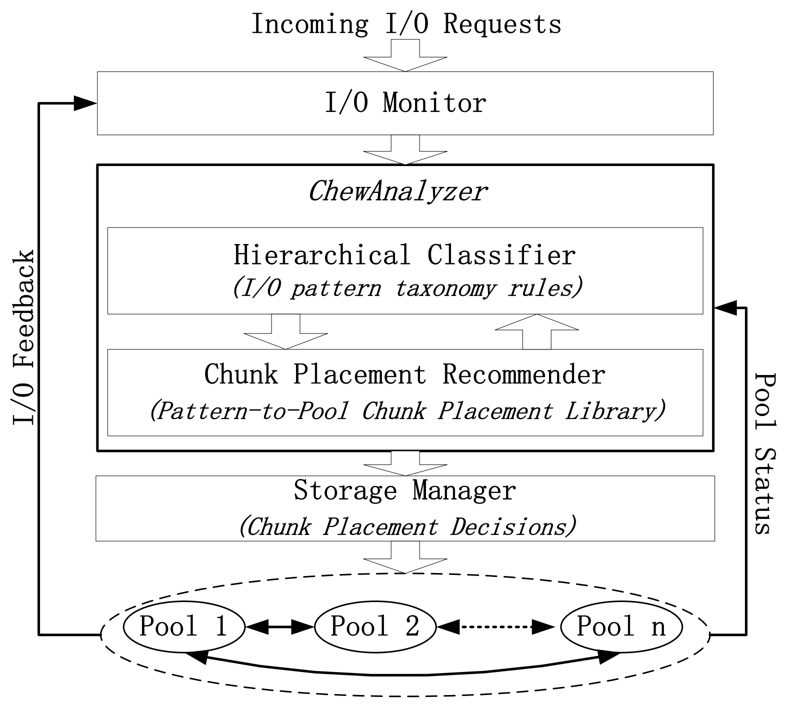
# ChewAnalyzer Framework

- **Data Path**
  - Chunk-level mapping table
    - Logical chunk number to physical chunk number
  - Current data location
    - <Physical chunk number, Offset>



# ChewAnalyzer Framework

- **Control Path**
  - I/O Monitor
    - Update I/O information of relevant chunk
  - If time window is full, for all chunks
    - Hierarchical Classifier for pattern detection
    - Chunk placement recommender
      - *Predefined referential Pattern-to-Pool library*
    - Chunk relocation decision maker
      - *Current status of each storage pool*



# Network Re-Design: Software-Defined Networks

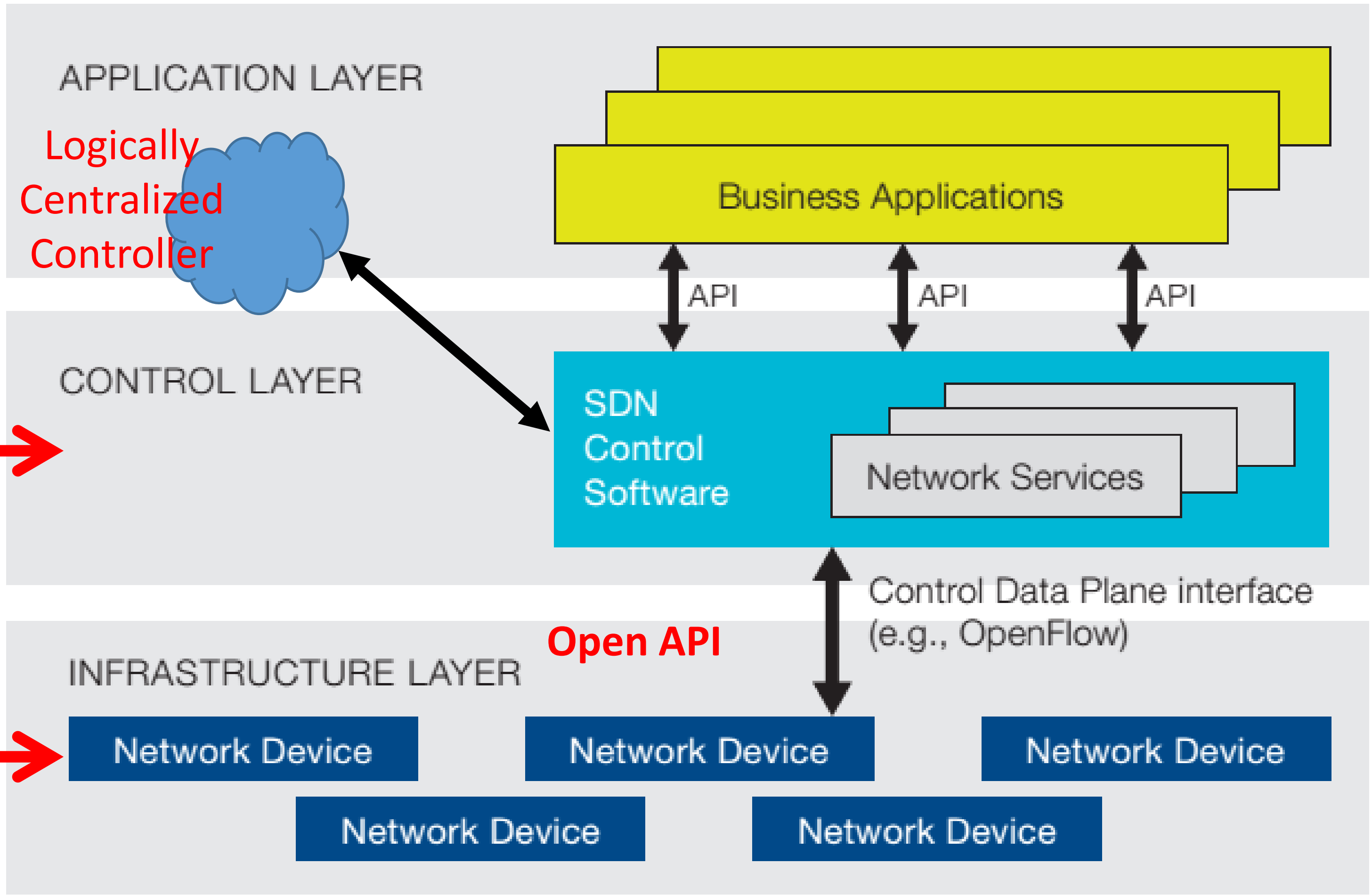
# Proposed SDN Solution

Standard API to
Enable
Programmable

Control Plane

Separation of
Control Plane and
Data Plane

Data Plane



# Goals of Using Software-Defined Networks

- How to Use White-Box Switches and Re-Programmable Routers?
- Integrating Required Network Functions (NFV) with Data Storage Using Docker Container
- Creating A Unified Management Platform for Compute, Network, and Storage
- Supporting Data Analytics and Decision Making with Integrated Hyperconverged Infrastructure

# Platform for Big Data Analysis and Its Performance Evaluation

**Understand the workloads in storage systems of big data**

**Key value store workload characterization** of big graph in Facebook

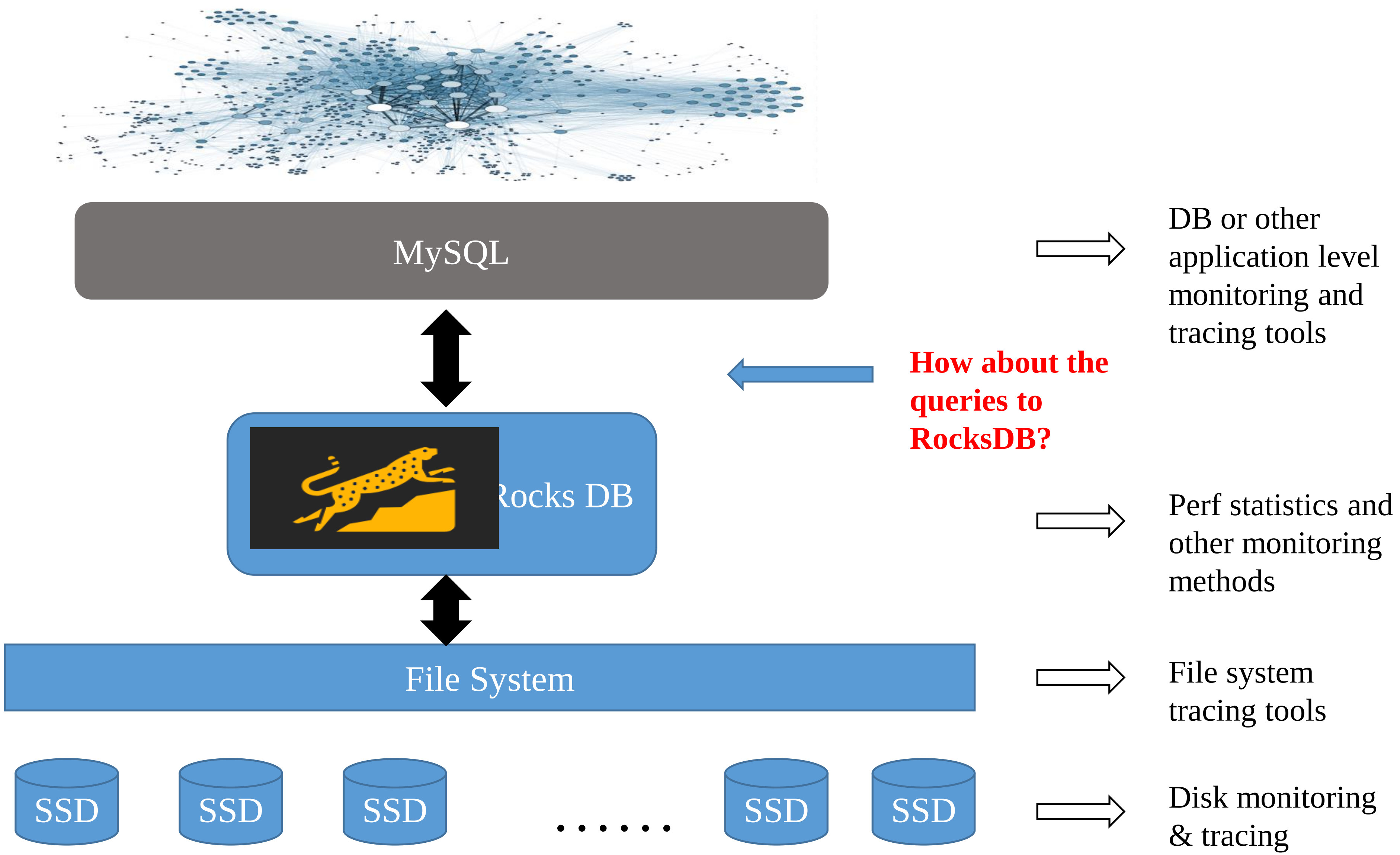
# Background and Motivations

- Key Value Store (KVS). is more and more widely used by applications as backend storage for structured/unstructured data, or even supporting file system

- RocksDB is a flash adaptive high performance KVS

- Existing studies about how to collect, characterize, and model KVS workloads is limited

- People has limited understanding of the workload in storage layer that supporting the big data.


MySQL
Rocks DB
File System
SSD
SSD
SSD
SSD
SSD
DB or other application level monitoring and tracing tools
How about the queries to RocksDB?
Perf statistics and other monitoring methods
File system tracing tools
Disk monitoring & tracing

# Current Contributions and Future Direction

- Propose the tracing and trace analyzing methodologies for key-value store
- Model the workload and develop a real-workload like workload generator for key-value store developers to evaluate and optimize the storage engine
- Help us to understand the workloads of key value store which supports the largest big graph in the world
- How to construct efficient big data platform for data analytics and big graph processing (future work)?

# Integrating SDN with Distributed Data Storage

Existing KVS

- Distributed Key-Value Store for Collecting Data from  IoT and Big Data Applications
- Query Distributed Key-Value Store without Using Meta-Data Servers

Research Goal:

- How to Efficiently Store, Manage, and Access Data from KVS?

# SDKinetic: A Software Defined Kinetic-Based Key-Value Store using The Programmable Switch and P4

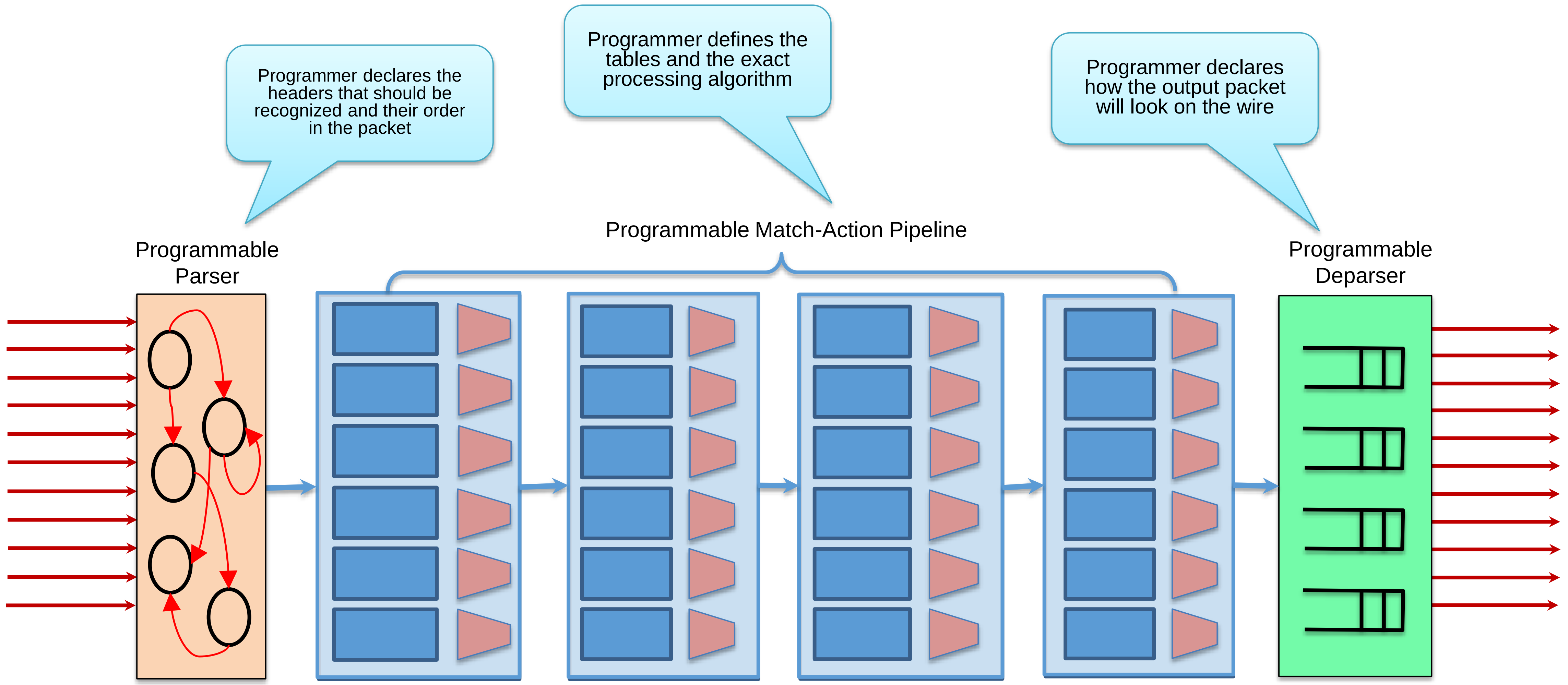
# Programmable Switches and P4

P4 is a high-level language for programming protocol-independent packet processors designed to achieve 3 goals.

- **Protocol independence.**
- **Target independence.**
- **Re-configurability in the field.**

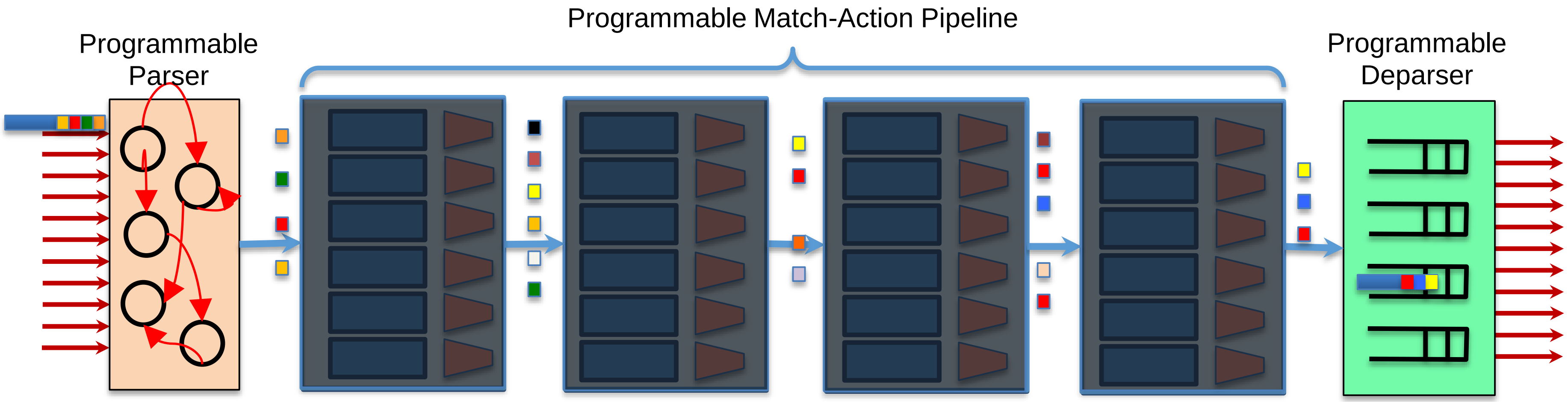
*Think programming rather than protocols…*

# PISA: Protocol-Independent Switch Architecture



# PISA in Action

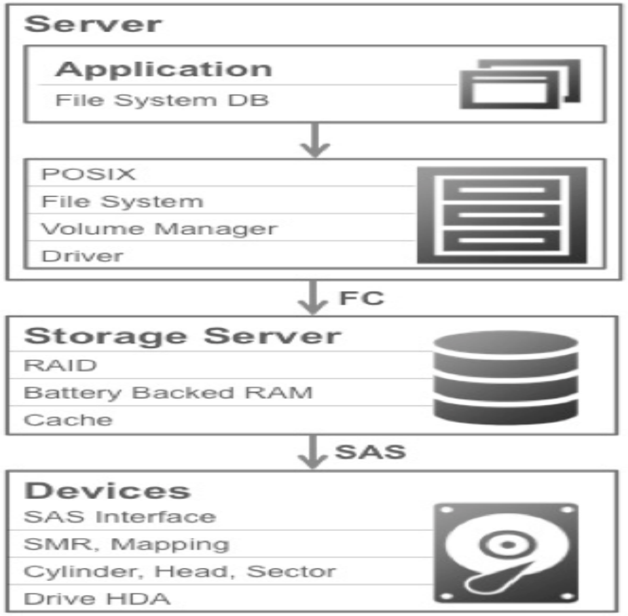
- **Packet is parsed into individual headers (parsed representation)**
- **Headers and intermediate results can be used for matching and actions.**
- **Headers can be modified, added or removed.**
- **Packet is deparsed (serialized).**



# Key-Value Store

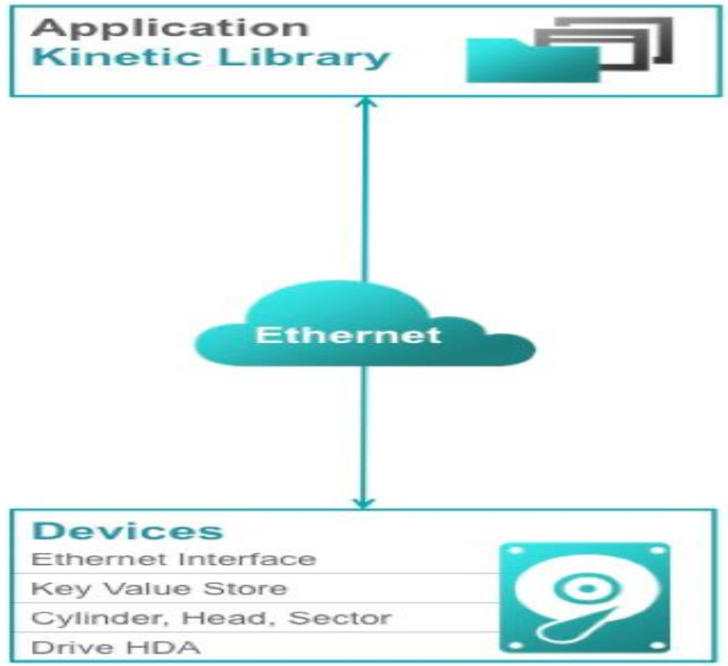
- The record is represented by two attributes:
  - **Key (identifier):** retrieve, modify, delete the record.
  - **Value:** the data itself like files, database records, images, graphs, or multimedia.

**Traditional Stack**




- All implementation is on the storage server.
- The storage server manages all the connected HDD/SDD with multiple of legacy layers that may introduce latency.

**Kinetic Stack**




kinetic drive is an independent and active device connected to the Internet.

# Our Goal

Building a Kinetic Drive or Server based large scale Key-Value Store with SDN to satisfy user requests and to improve the performance of the storage system by exploiting parallelism and embedding index table in SDN

Challenges:

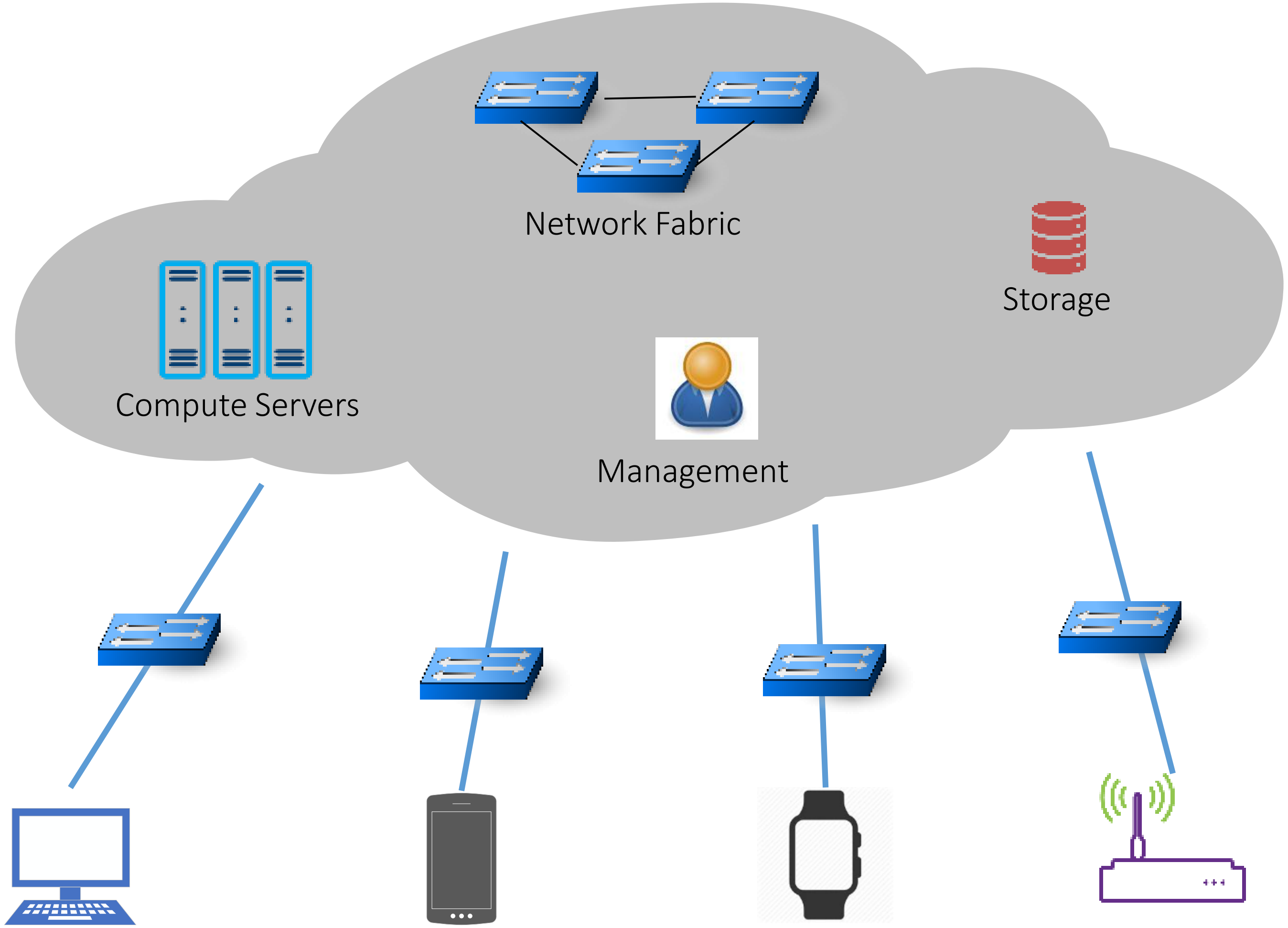
- Removing Metadata server
  - Metadata server forms a single point of failure.
  - Potential server bottleneck (All requests are sent to the metadata server for index searching).
- How to allocate data (key-value pairs)
  - Kinetic Drive has limited bandwidth (60 MB/sec) and limited size.
  - Data popularity and size keep changing (fixed allocation will not be enough)
- Improving Average Response Time
  - 2RTT for satisfying the request  with metadata server (1 RTT for getting IP + 1 RTT for getting data)
  - Contacting multiple drives for getting the data ( increase the response time)
- Cashing in Network and Load Balancing with SDN
- Reliability Issue (disk drive or switch failure)

# Proposed Solution

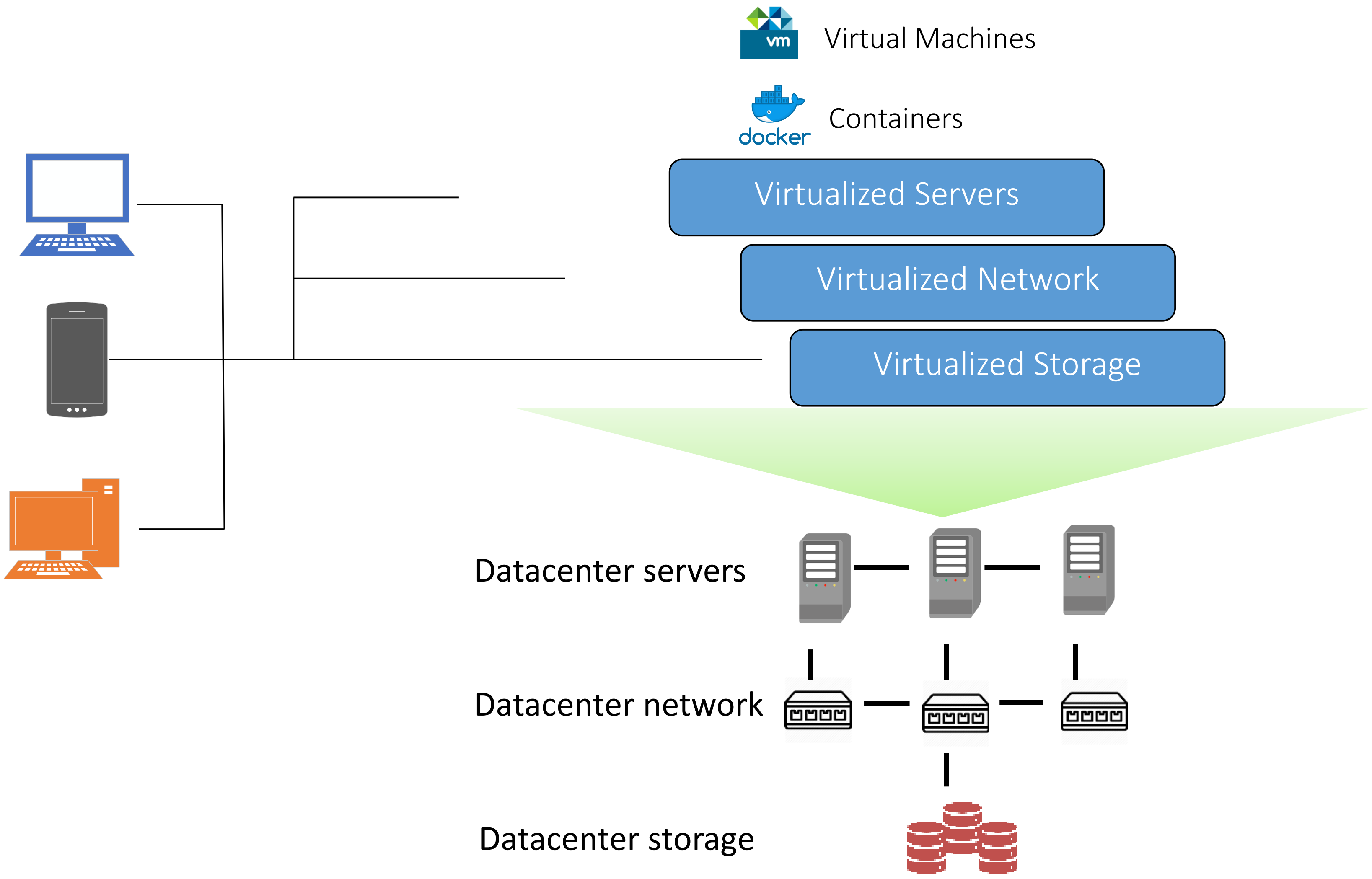
- Use the logically centralized design in SDN to collect performance parameters of each component
- Use the P4 switches instead of normal switches inside the distibured network
- Build and distribute the index table as rules on the switch with match-action table
- Using a key-range routing approach instead of the normal IP routing to route the request from a client to the target drive without contacting any server at the beginning to know the drive IP address
- Using the normal IP routing to route the data back from the drive back to the client.

# Ensure Application Performance with Docker Containers by Considering Hyperconverging

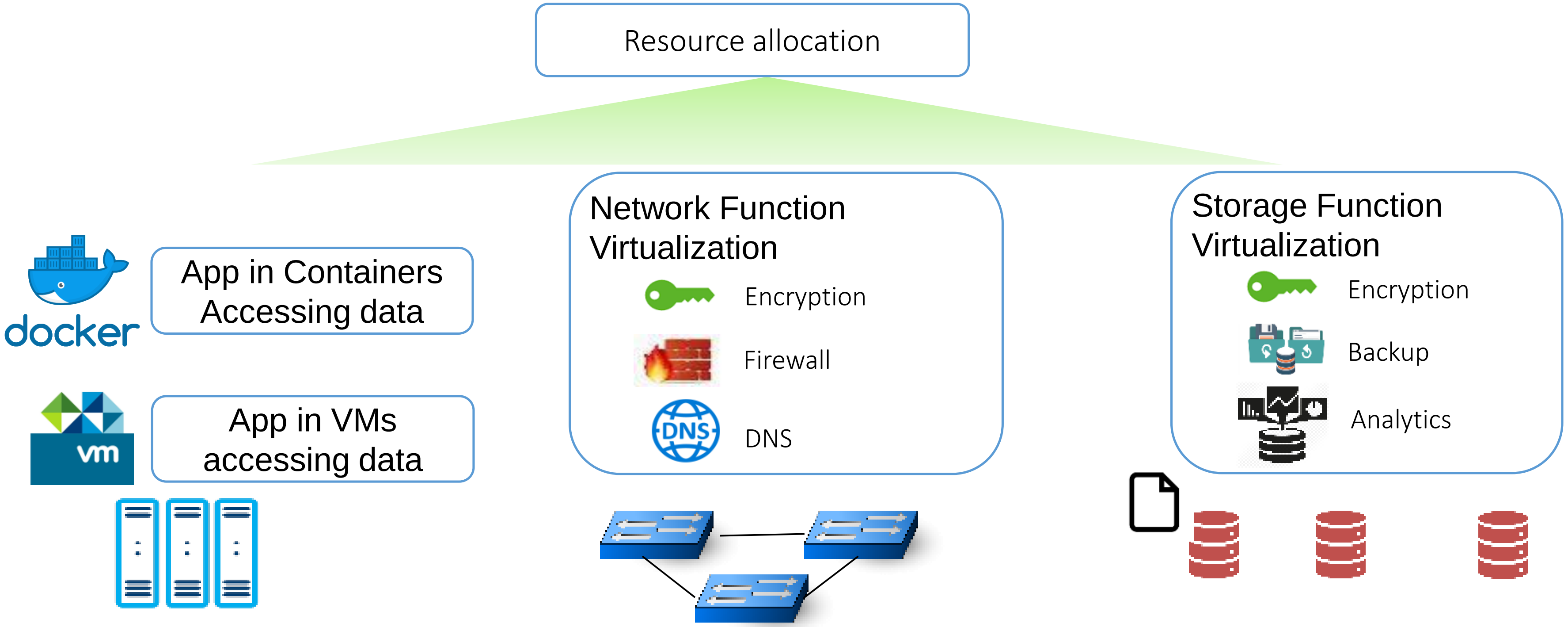
# Today’s Cloud Infrastructure is hyperconverged



# Virtualization is the Building Block

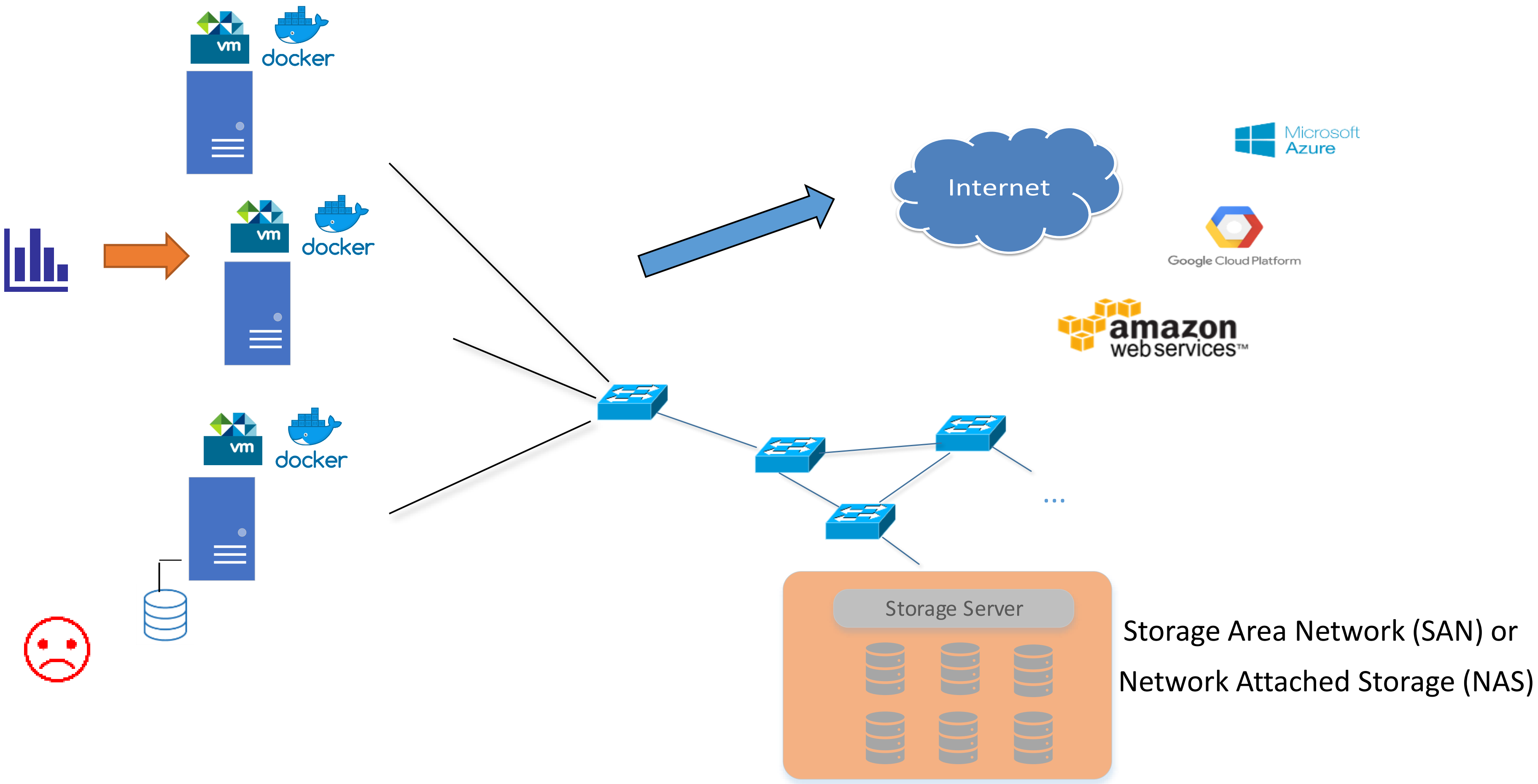


# Improve Application Performance in Emerging Hyper-converged Infrastructure



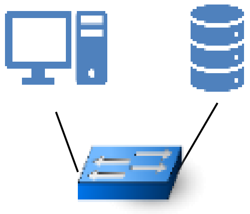

Ability to control all resources → Systematic control over client, network, storage for app in networked storage

# What is Networked Storage



# Two Research Projects

- Enhance storage support in container
  - Applications run in containers in the hyper-converge infrastructure. Propose a system that can support applications with various storage requirements deployed in the Kubernetes environment based on Docker containers. [Under submission]



- Improve I/O latency in the networked storage environment
  - Propose a system that coordinates different components along the I/O path to ensure latency SLO for applications in networked storage environment. [MASCOTS'18]

# Kubernetes - Distributed OS of Containers

An orchestrator is essential to deploy and manage applications in containers across multiple hosts.

- Application scheduling
- Resource management
- Mainstream: Docker swarm, Mesos, and Kubernetes (k8s)[7] [Verma et al. EuroSys '15, Burns et al. Queue 14, 1]

Kubernetes is the most popular container orchestration platform according to surveys from Cloud Native Computing Foundation (CNCF) [8,9]

In this research, we focus on Kubernetes environment based on Docker.

[7]Kubernetes concepts. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.
[8]Survey Shows Kubernetes Leading as Orchestration Platform. https://www.cncf.io/blog/2017/06/28/survey-shows-kubernetes-leading-orchestration-platform/.
[9]CNCF Survey: Use of Cloud Native Technologies in Production Has Grown Over 200%. https://www.cncf.io/blog/2018/08/29/cncf-survey-use-of-cloud-native-technologies-in-production-has-grown-over-200-percent.

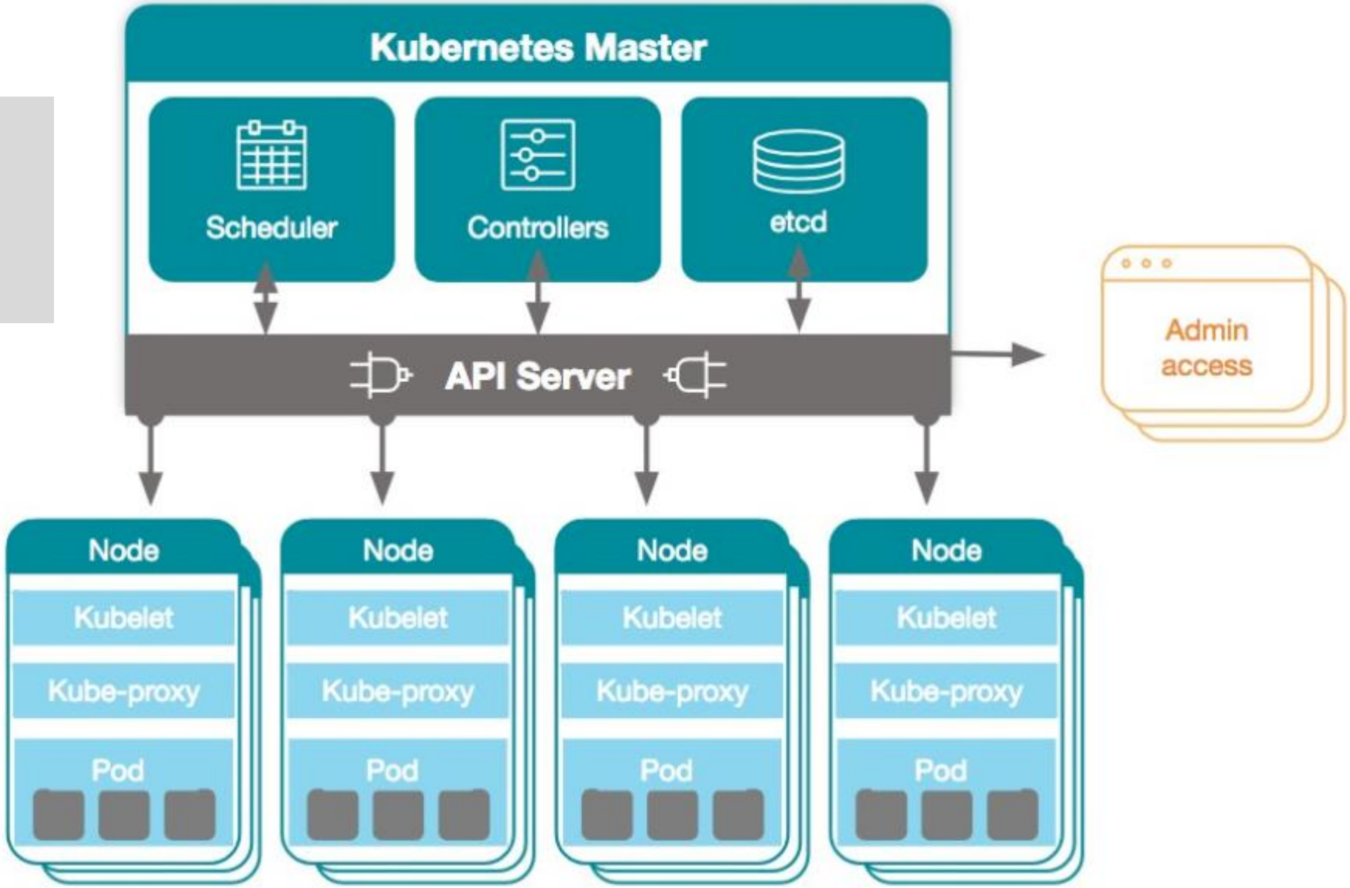
# Issues of Kubernetes in Storage Allocation

Storage allocation is static

CPU, Mem, Affinities to apps/nodes ✓

Storage resources ✗

Error-prone, not resource efficient storage allocation



# Static Storage Allocation in K8s

- K8s allocates storage based on *StorageClass* (SC)

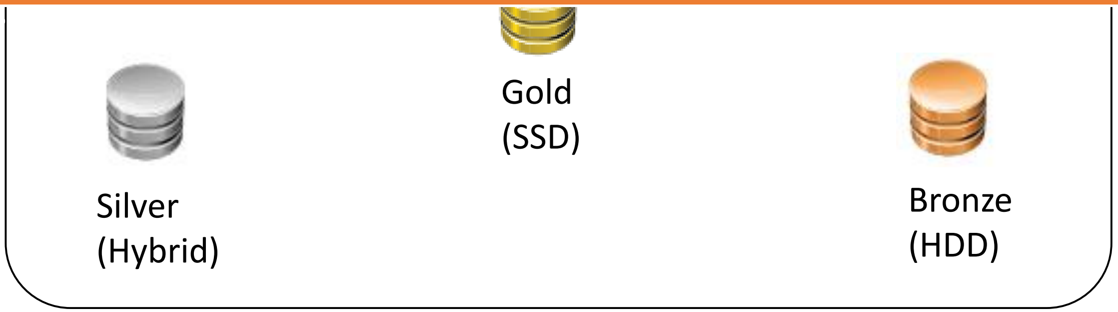
Admins create SCs

Users choose SCs

Limitations:

- SC is static. Storage performance is changing
- Few SCs -> Over provisioning
  Lots of SCs -> Hard to maintain
- Advanced storage requirements, e.g., rate limiting, caching, etc.
- Not user friendly and error-prone

How can we make k8s better meet users' storage requirements & all other requirements, and at the same time



# Our Contributions

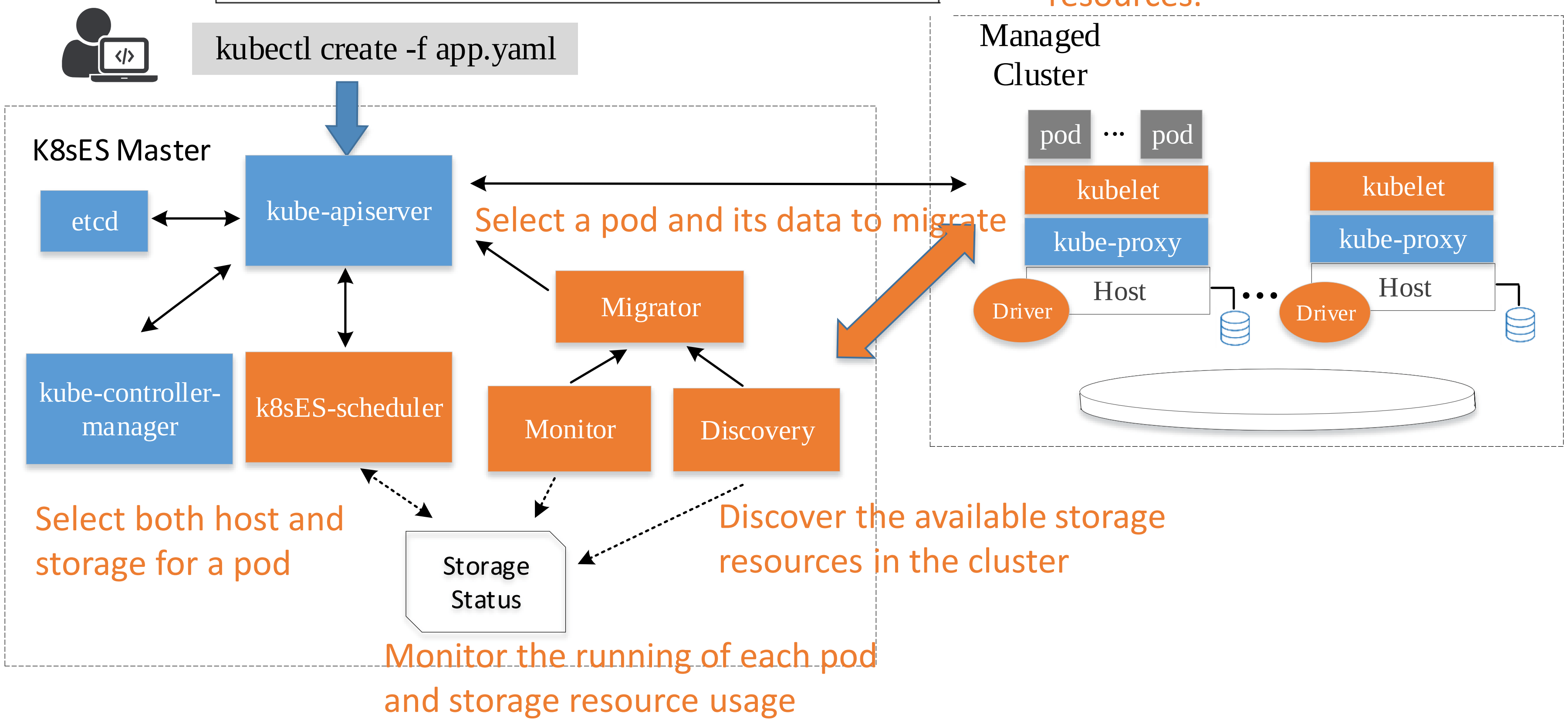
We propose *K8sES* (k8s Enhanced Storage), a system that can dynamically allocate storage to applications in Kubernetes based on users' storage requirements.

- Initial storage allocation
  - **Storage monitoring capabilities**: performance of storage devices
  - **User friendly.** Allow users to specify storage requirements directly in config.
  - **No limitations of SC.** Admins don't create SC.
  - **Strengthened scheduling.** Select storage with other k8s related requirements
  - **Automatic storage provisioning** based on users' requirements
- Storage adjustment at runtime
  - **Storage monitoring capabilities:** enforcement of storage SLOs of a pod
  - **Migration**
- **Improves storage utilization efficiency** in k8s: thin provisioning, multiplexing, balancing utilization between storage and non-storage

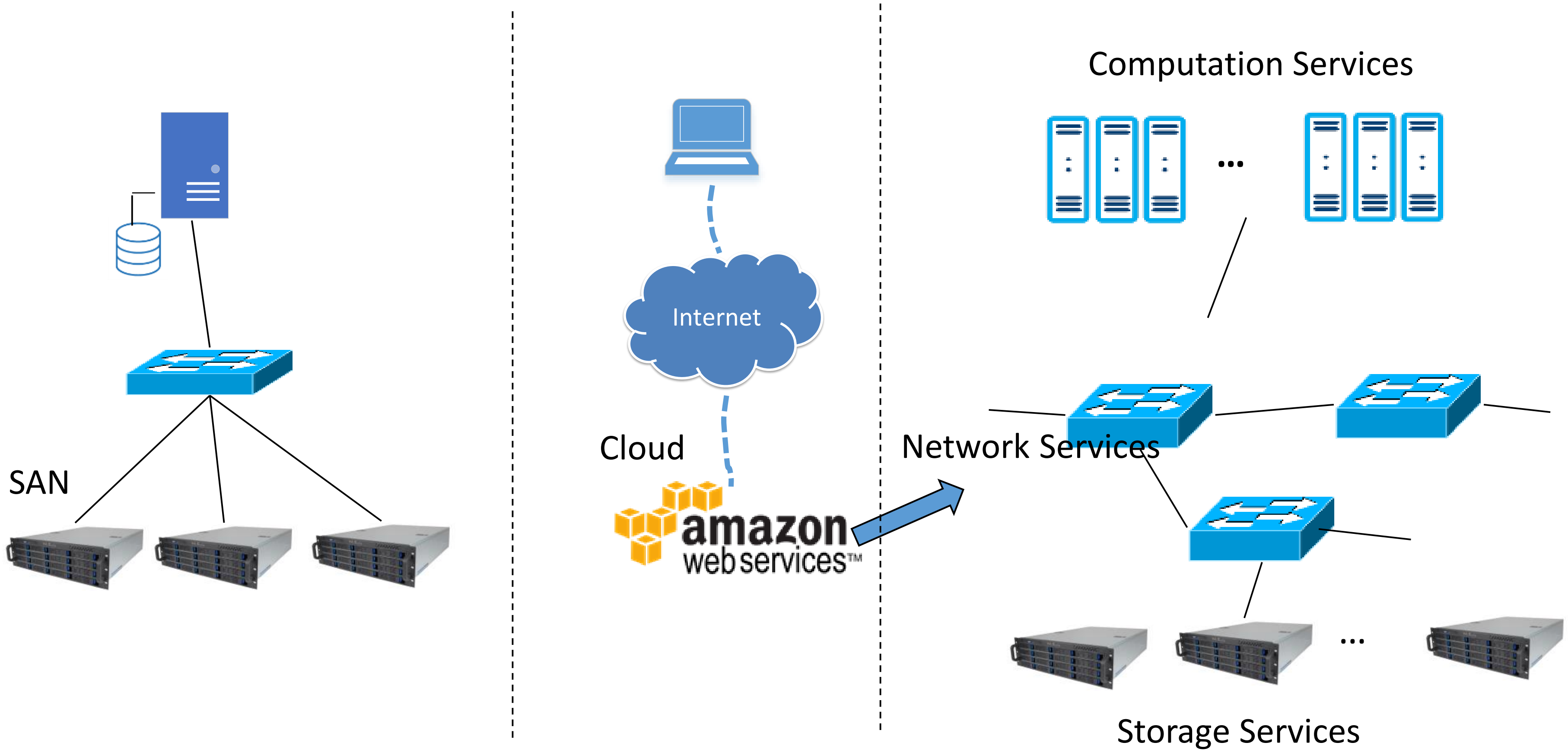
# Pod Creation

```
<k8sES volume>
  size: x GB
  sustained bw: y MB/s
  sharing: False
  reclaim: Retain
  policy: WHEN GETS/s > z, SET CACHING
```

The kubelet receives the storage decision from k8es-scheduler and call the ***Driver*** to carve out storage resources.



# Network is Important in Data Access




E.g., OpenStack (VM), Kubernetes (containers)

# Problem and Challenges

In the networked storage environment, how can we **coordinate different components** in network and storage to improve latency SLOs for applications?

Challenges:

- Different components involved, e.g., clients, network switches, storage servers, disks, etc.
- Status of the components are dynamically changing
- Each component performs different functions on I/Os

# Our Contributions

- We identify the need to consider all the components along the I/O path to ensure latency SLO.
- We design a controller-based mechanism to coordinate the control on different components dynamically based on the status of network and storage.
- We design an approach to control I/O packets with little overhead based on the asymmetry property in read and write.
- We build a real system called JoiNS, to coordinate clients, network, and storage, and demonstrate the effectiveness in ensuring latency SLO.

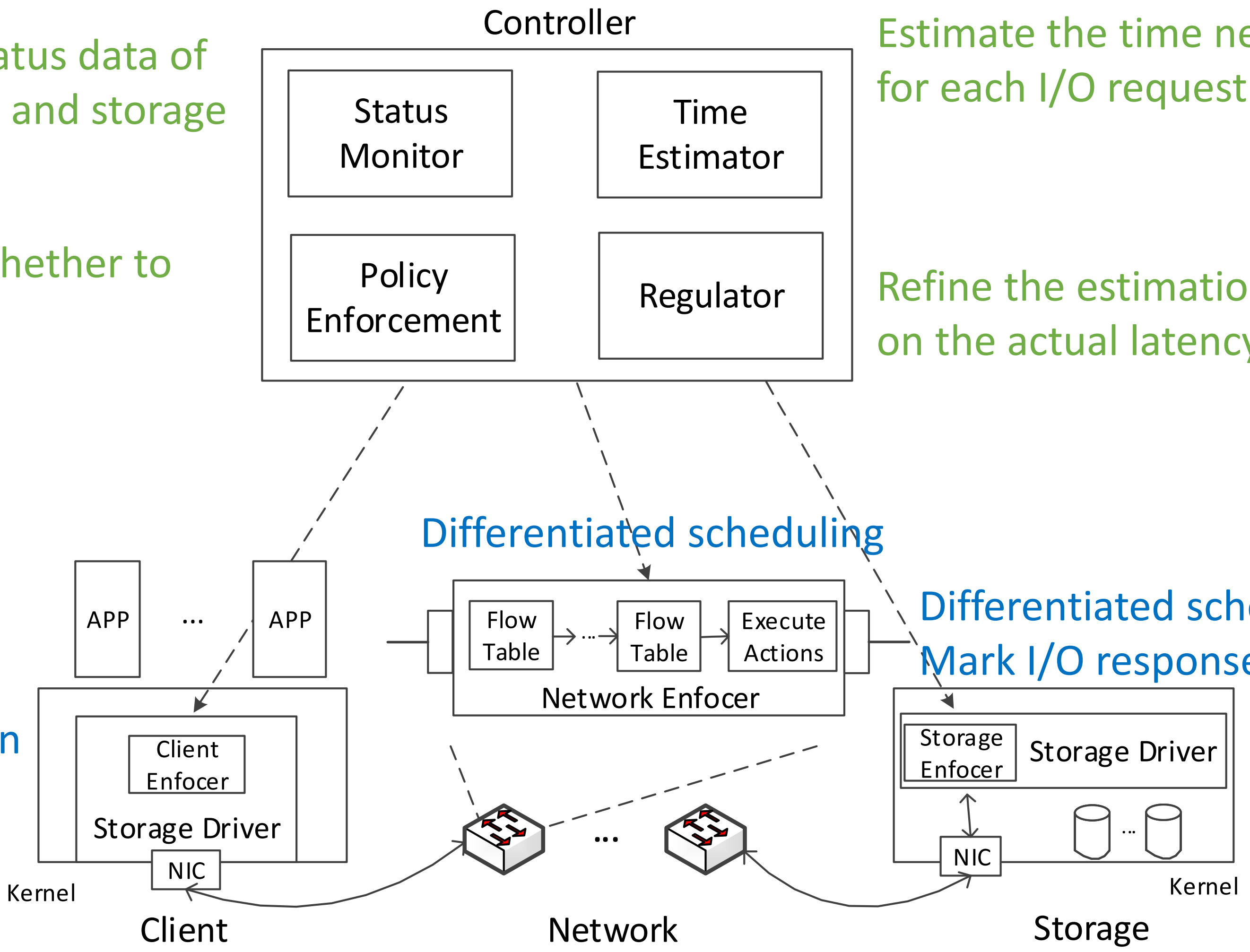
# JoiNS Architecture

Collect the status data of each network and storage node

Determine whether to control I/Os

Estimate the time needed for each I/O request

Refine the estimation based on the actual latency




Differentiated scheduling

Differentiated scheduling

Mark I/O responses

Admit I/Os

Mark I/O requests in packet headers and storage commands

# Cost-effective Control

- Distinguish Read from Write
  - Based on the asymmetry property in read and write along its I/O path.
  - Read requests can be prioritized on request path with little penalty.
  - Write responses can be prioritized on return path with little penalty.