Integrated Forensic Platform

An Overview

2019-09-09

Introduction

- Digital forensics can combine a variety of data source to identify evidence.
- Different tools need to be combined, the tools may require specific OS, libraries, DB technology, etc.
- Existing approaches to integrate forensic tools is via Linux distribution, e.g., Kali Linux.
- We have developed number of various tools, how can be integrated?
- One of the main result of this project is an integrated digital forensic platform.

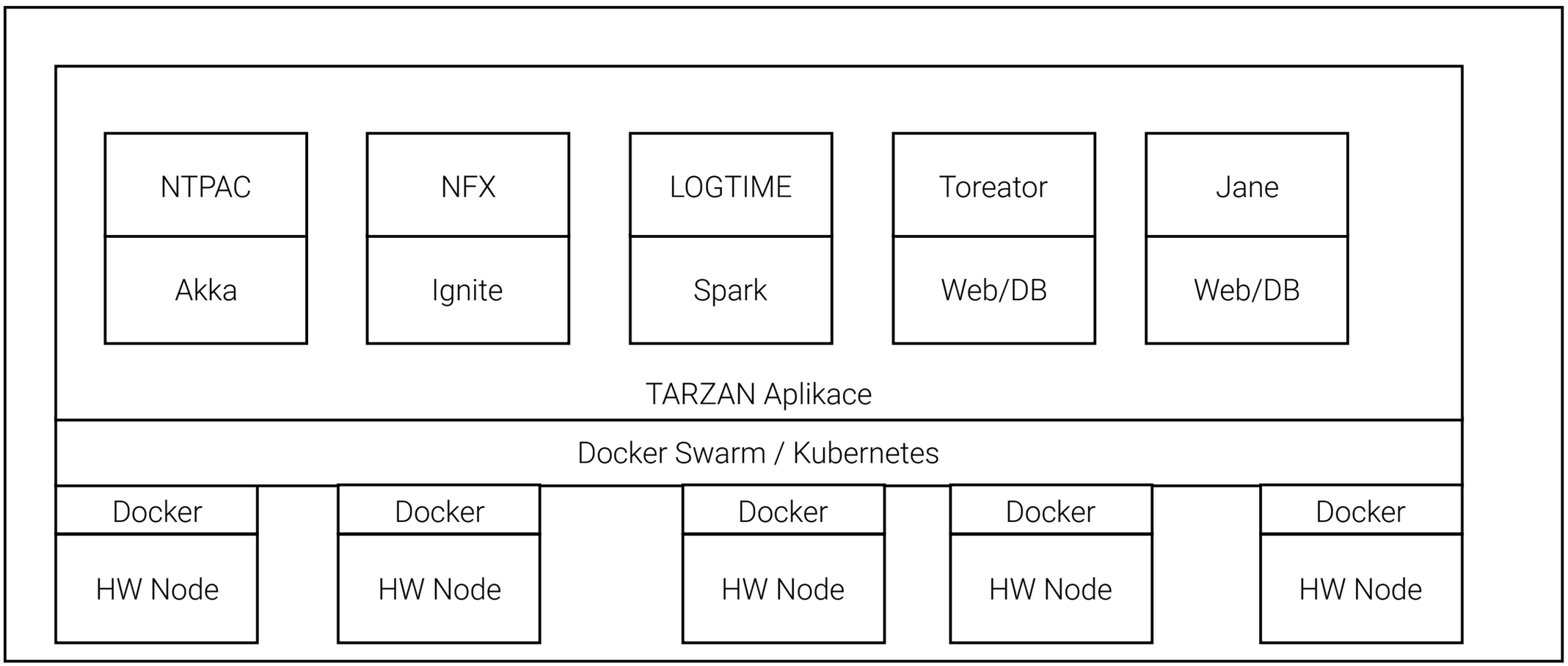
https://resources.infosecinstitute.com/category/computerforensics/introduction/free-open-source-tools/overview-of-computer-forensics-linux-distributions/

Areas/Tools in TARZAN

- **Network forensics**
  a tool for processing of captured network communication
- **Computer forensics**
  a tool for analysis of disk drive images
- **Social network forensics**
  a tool for identification of various events in social networks related to specified subject
- **Tor analysis and monitoring**
  a tool providing Tor network operational information
- **Cryptocurrency transaction analysis**
  a tool suitable for analysis and tracking transactions

Technology Enablers

- The platform is a computing cluster providing hardware resources for Docker Swarm
- Individual tools provide API for intercommunication
- Web-based user interface



{ REST }

React

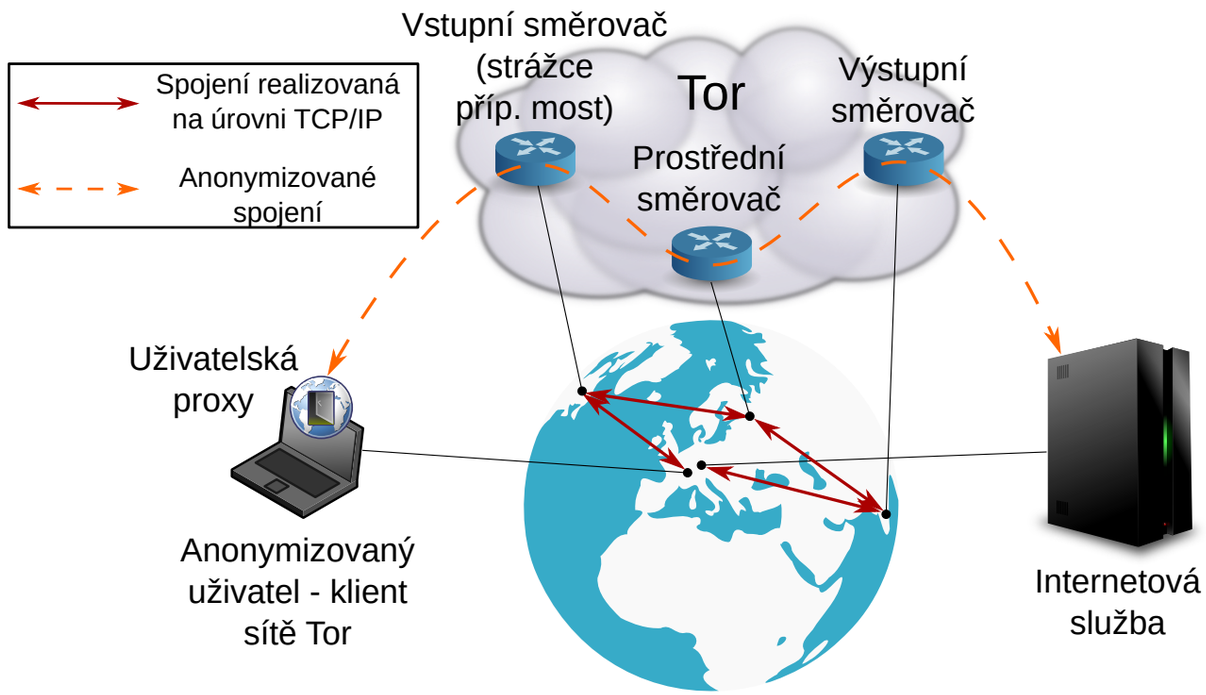
node JS

kubernetes

docker

Toreator

- An application for searching existence of IP addresses in Tor network.
  - Toreator proxy - GraphQL proxy implementation provides forwarding and caching for Toreator service.
  - Toreator web - Single-page React application.



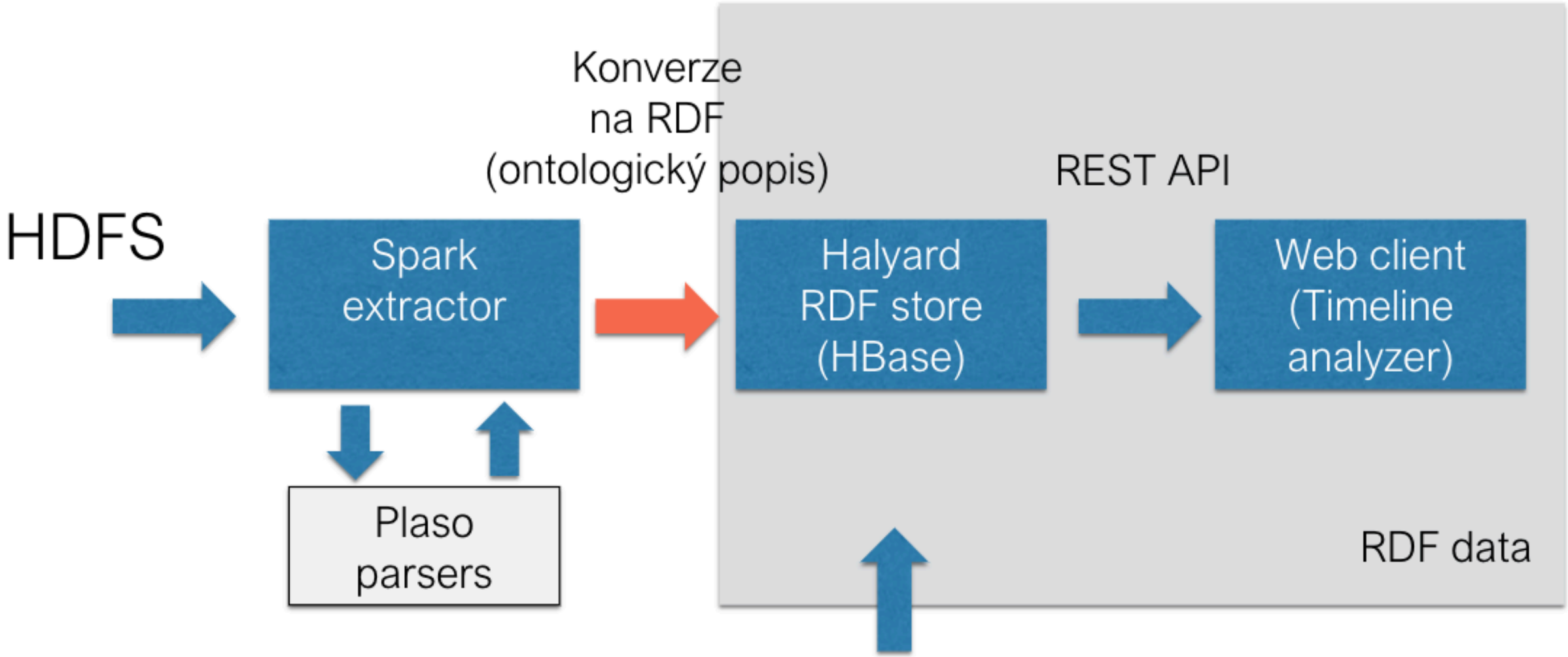
JANE

- A collection of modules/services:
  - jane-cryptoclients - a collection of 7 most popular cryptoclients for RPC access.
  - jane-cryptoalarm - a module that enables to raise an alarm on the appearance of particular transactions.
  - sMaSheD - a system for detection of cryptocurrency miners by network forensic approach.
- Implemented as services, deployed in containers
- Web interface + REST API

PLASO

- Plaso is a computer forensic tool for timeline generation and analysis.
- Log2Timeline extracts events from individual files
- Use cases:
  - Basic forensics investigation of a hard drive
  - Interactive timeline analysis of the content
  - Automated reporting based on predefined queries
- Issues of the integration with TARZAN
  - Plaso is a monolithical Python application
  - Plaso provides a large set of various parser/extractors

PLASO for TARZAN



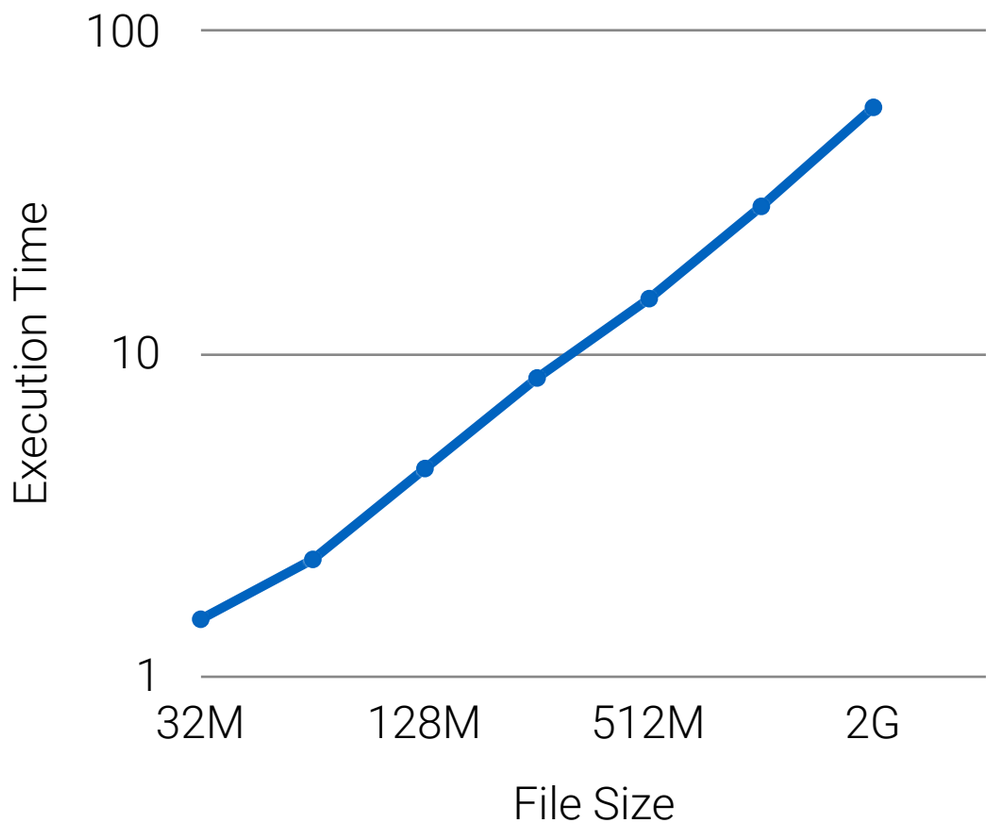
Network Forensics

- I have several of TB of PCAP files from various campaigns, test beds, honeypots, etc.
- It would be nice if it was possible to analyse them quickly.
  - I want to just know what data I have - IP addresses, applications and services, timeline, etc.
  - Found a security issue in one set, I can create a specification of this issue and test if such issues is not in other datasets too.
  - I want to extract only a specific content/metadata for further analysis by some other tool, e.g., Network Miner,ngrep, mailsnarf,smtpcap,tstat, dsniff, firesheep,nfex,driftnet).
  - and many other cases…

Too big data for Wireshark…

- Wireshark/tshark + scripting is fine, but when there is a lot of data it takes a lot of time.
  Get a log of all DNS packets:

File Size	Execution time (s)
32M	1.596
64M	2.283
128M	4.400
256M	8.431
512M	14.747
1G	28.627
2G	57.558
4G	161.566
25G	877.220. (~15m)

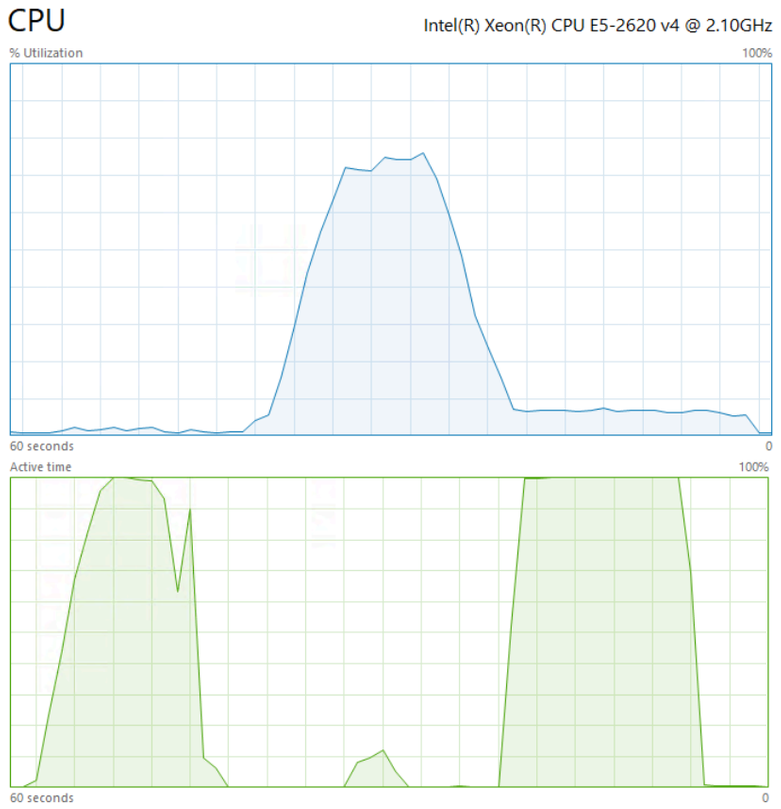


- Try the similar things with Scapy :)

Parallel TShark ?

- Many tasks can be executed independently, e.g., filtering all DNS traffic, extracting objects from all flows, etc.
- TShark does not utilize all available resources…
  ~5% on Intel Xeon E5-2620 2.1Ghz
- Executing many TShark instances in parallel (4GB input):

Number of Workers	Execution time (s)
2	72.564
3	54.234
4	40.123
5	40.860
9	41.136
18	43.334
36	44.244
70	45.828



Speed up the process

- CPU is not an issue, the problem is I/O
- We can buy a lot of memory - doing in memory as much as possible provides the significant performance improvement.
- Popular frameworks for in-memory data processing:

Use Cases of NFAT

- Network traffic capturing and analysis
- Evaluation of network performance
- Detection of anomalies and misuse of resources
- Determination of network protocols in use
- Aggregating data from multiple sources
- Security investigations and incident response
- Protection of intellectual property

Typical Workload

- Feature extraction
  - Packet-level
    timestamp, size, entropy, n-grams, inter packet delay, …
  - Flow-level
    usual flow information, extended information considering some application specific data, statistical information
- Content Analysis
  - Packet-level
    CoAP, MQTT messages, DNS data, …
  - Flow-level
    TCP/UDP Streams

Typical Workload

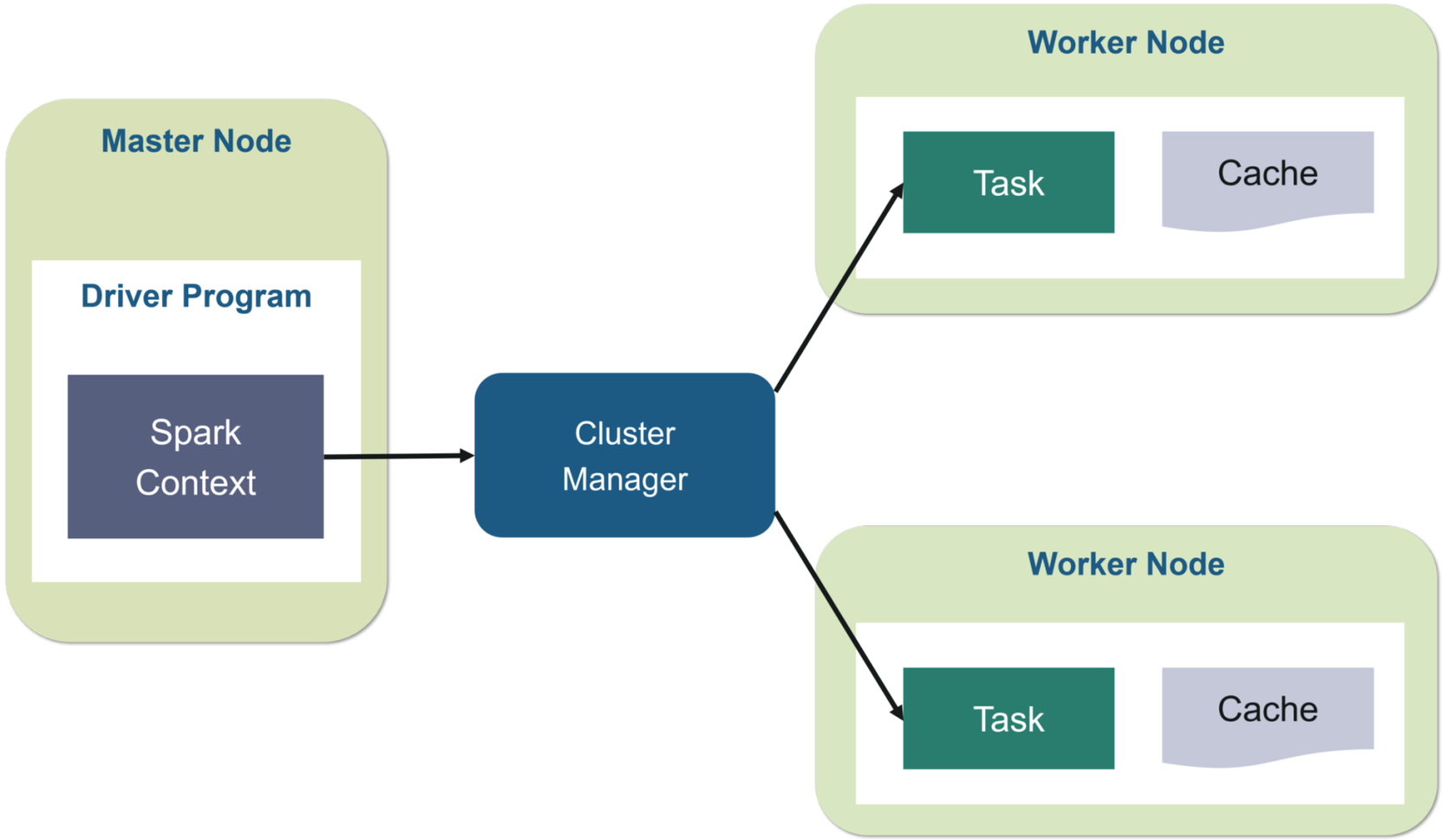
- Feature extraction
  - Packet-level
    timestamp, size, entropy, n-grams, inter packet delay, …
  - Flow-level
    usual flow information, extended information considering some application specific data, statistical information

APACHE SPARK

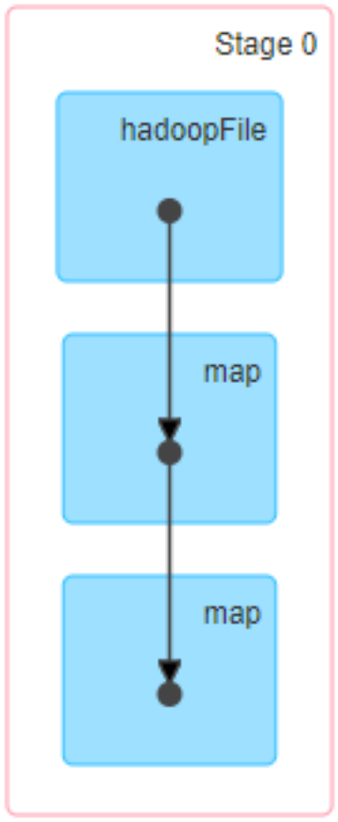
- Content Analysis
  - Packet-level
    CoAP, MQTT messages, DNS data, …
  - Flow-level
    TCP/UDP Streams

APACHE IGNITE

Apache Spark



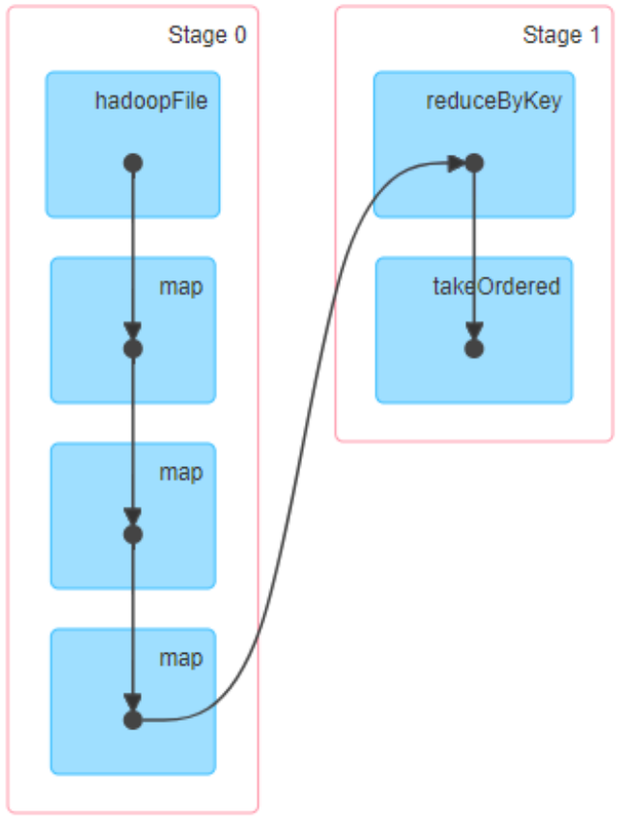
Simple operation (capinfo)




```scala
import org.ndx.model.Packet;
import org.ndx.model.PacketModel.RawFrame;
import org.ndx.model.Statistics;

val frames = sc.hadoopFile("hdfs://neshpc1.fit.vutbr.cz/user/rysavy/cap/*.cap",
classOf[org.ndx.spark.pcap.PcapInputFormat],
classOf[org.apache.hadoop.io.LongWritable],
classOf[org.apache.hadoop.io.ObjectWritable])

val packets = frames.map(x=> Packet.parsePacket(x._2.get().asInstanceOf[RawFrame]))

val capinfo = packets.map(x => Statistics.fromPacket(x)).reduce(Statistics.merge)
```

```
File type:           Wireshark/tcpdump/... - pcap
File encapsulation:  Ethernet
File timestamp precision:  microseconds (6)
Packet size limit:   file hdr: 65535 bytes
Number of packets:   50 M
File size:           11 GB
Data size:           10 GB
Capture duration:    24789795.090000 seco
First packet time:   2015-04-20 13:49:56.140000
Last packet time:    2016-02-01 10:53:11.230000
Data byte rate:      417 bytes/s
Data bit rate:       3337 bits/s
Average packet size: 203.39 bytes
Average packet rate: 2 packets/s
SHA1:                97efe62aaa42402d3b84292d1fc010a5090f3332
RIPEMD160:           e711effc6f6c47490f886df5fd7941cfc5df1b47
MD5:                 54844e4f6d9fa58f30b293d64a6b4150
```

Little bit more…(get-flows)

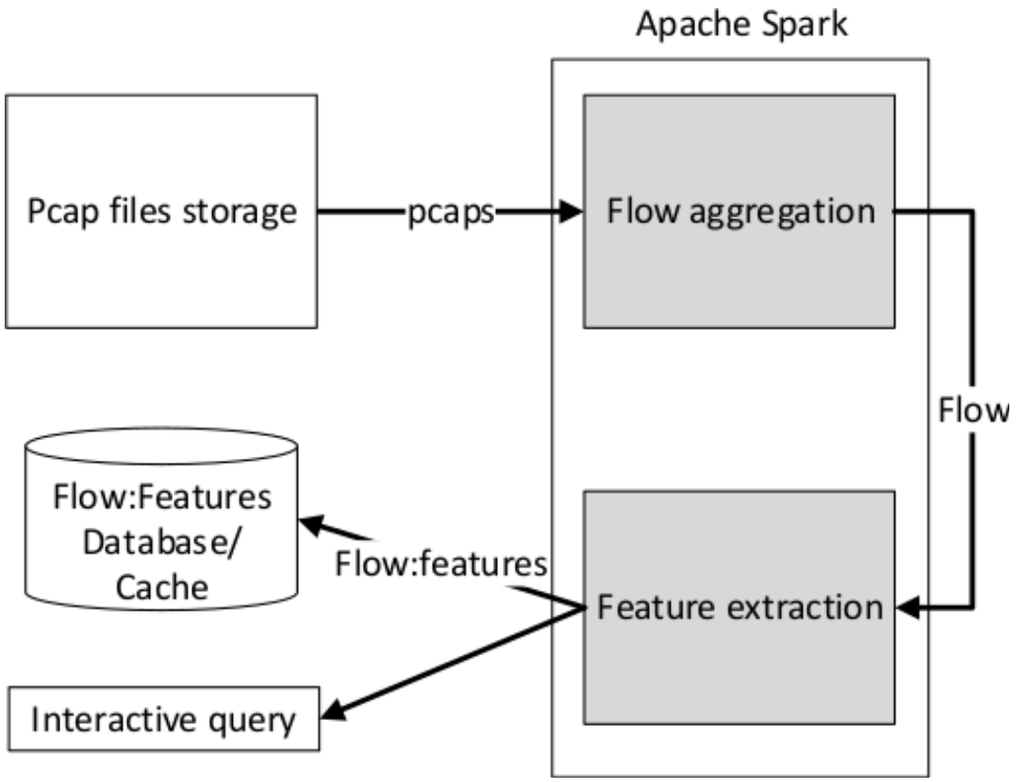



```
import org.ndx.model.Packet;
import org.ndx.model.PacketModel.RawFrame;
import org.ndx.model.Conversations;

val frames = sc.hadoopFile("hdfs://neshpc1.fit.vutbr.cz/user/rysavy/cap/*.cap",
classOf[org.ndx.spark.pcap.PcapInputFormat],
classOf[org.apache.hadoop.io.LongWritable],
classOf[org.apache.hadoop.io.ObjectWritable]);
val packets = frames.map(x=> Packet.parsePacket(x._2.get().asInstanceOf[RawFrame]));
val flows = packets.map(x=>(x.getFlowString(),x));
val stats = flows.map(x=>
(x._1,Conversations.fromPacket(x._2))).reduceByKey(Conversations.merge);
println("Date flow start          Duration Proto   Src IP Addr:Port      Dst IP
Addr:Port     Packets    Bytes Flows")
stats.takeOrdered(10)(Ordering[Int].reverse.on(x=>
x._2.getPackets())).map(c=>Conversations.format(c._1, c._2)).foreach(println)
println("Done.")
```

Completed Stages (2)

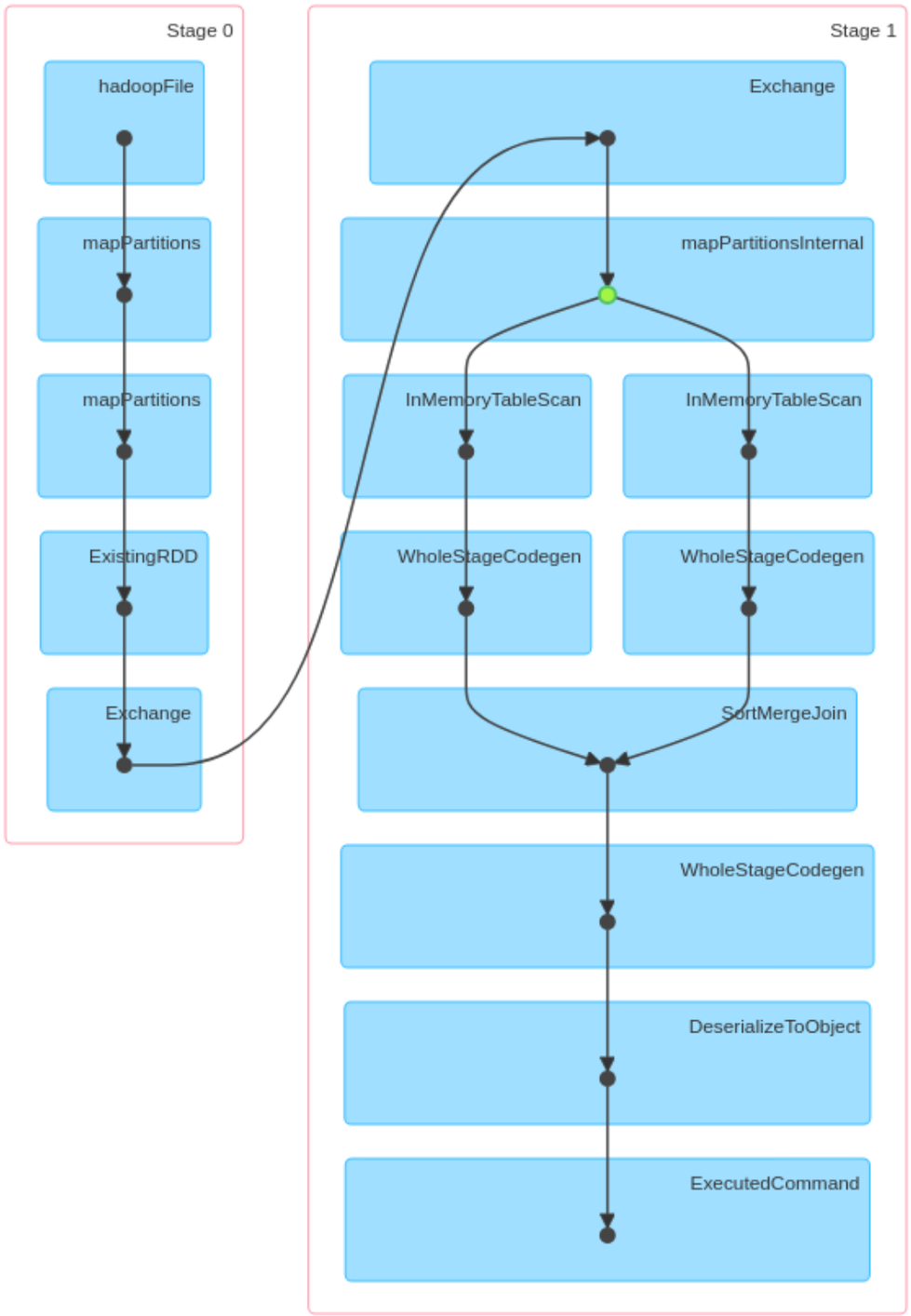
Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
1	takeOrdered at <console>:36	+details	2017/08/23 14:45:06	3 s	102/102			191.5 MB	
0	map at <console>:33	+details	2017/08/23 14:44:48	17 s	102/102	10.4 GB			191.5 MB

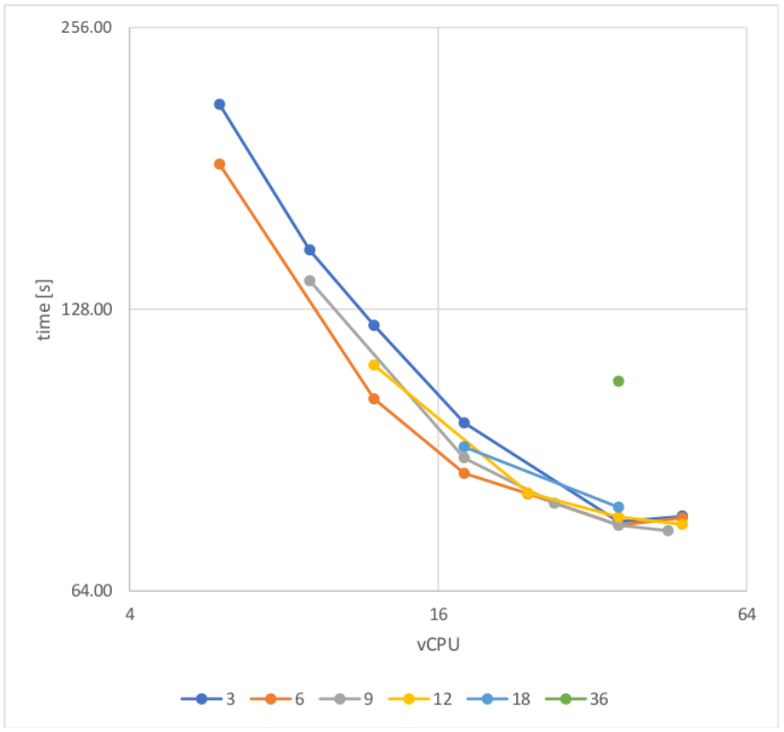
Feature Extraction

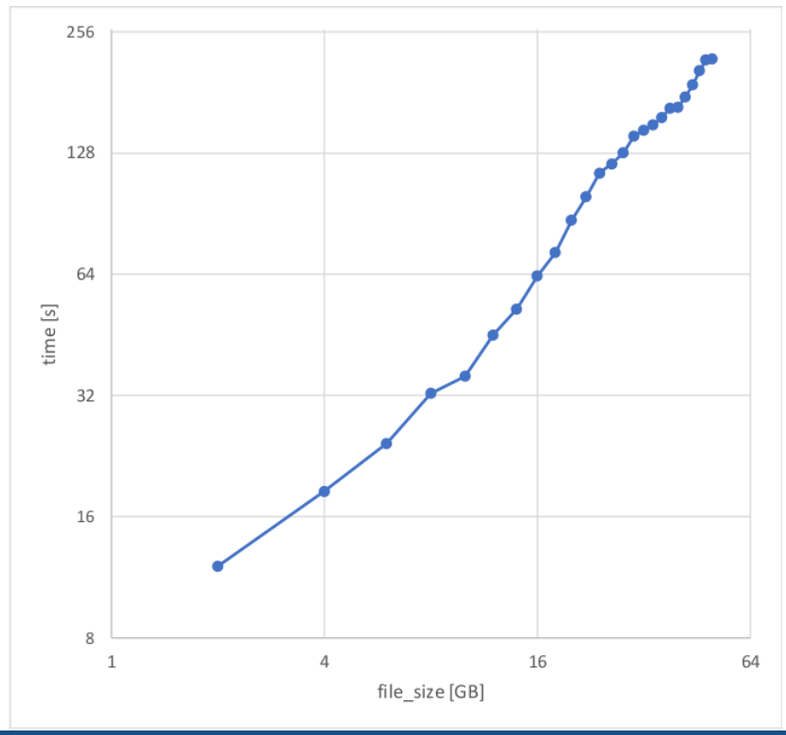


Name	Description
proto	Protocol
srcp	Source port
dstp	Destination port
packets	Number of packets in flow
size	Size of all packets in flow
paysize	Total size of payload in all packets in flow
duration	Flow duration
nopay	Number of packets without payload
avgps	Average packet size
minps	Minimal packet size
maxps	Maximal packet size
stddps	Standard deviation of packet size
avgpays	Average payload size
minpays	Minimal payload size
maxpays	Maximal payload size
stddpays	Standard deviation of payload size

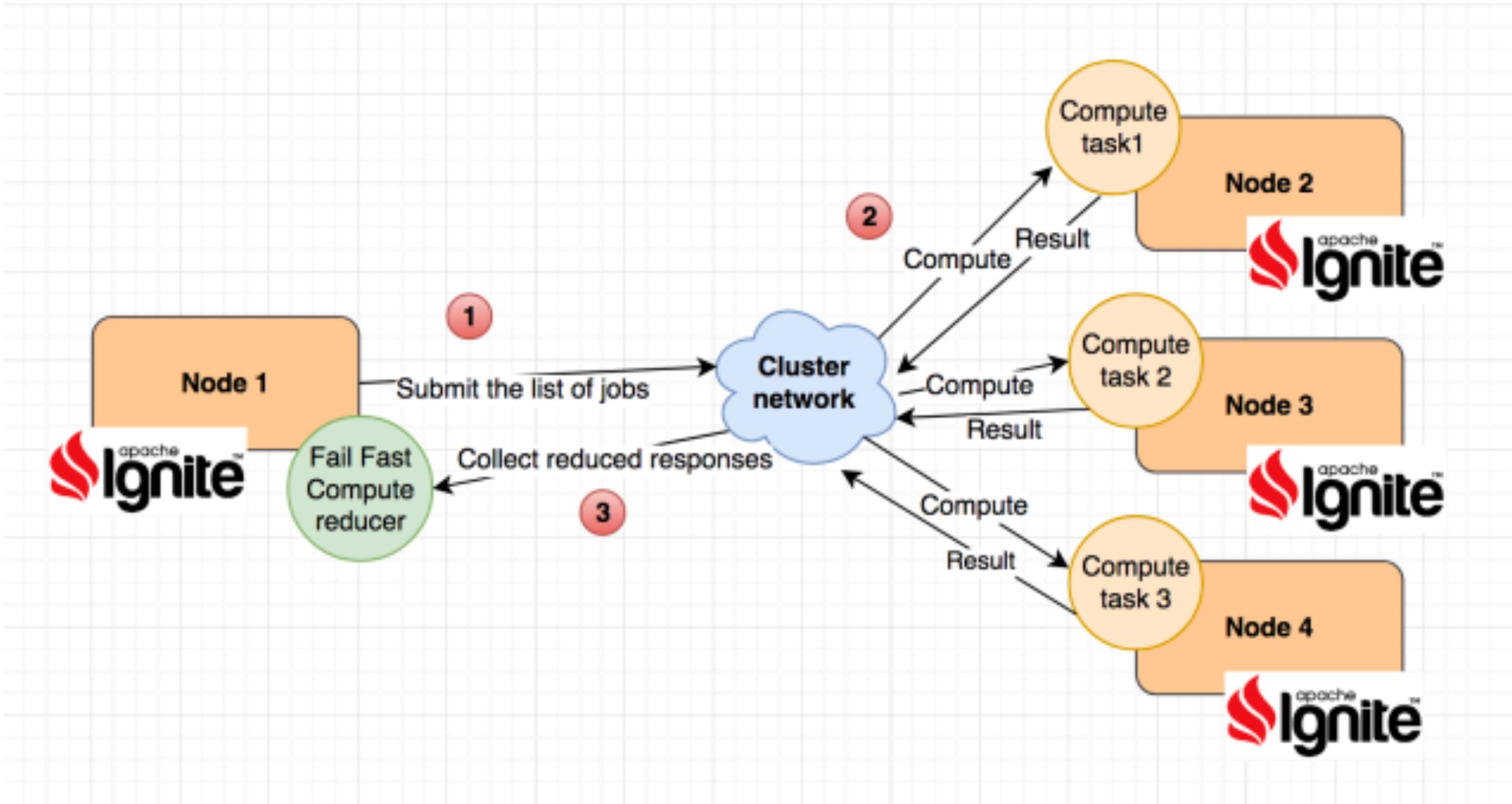
Processing pipeline



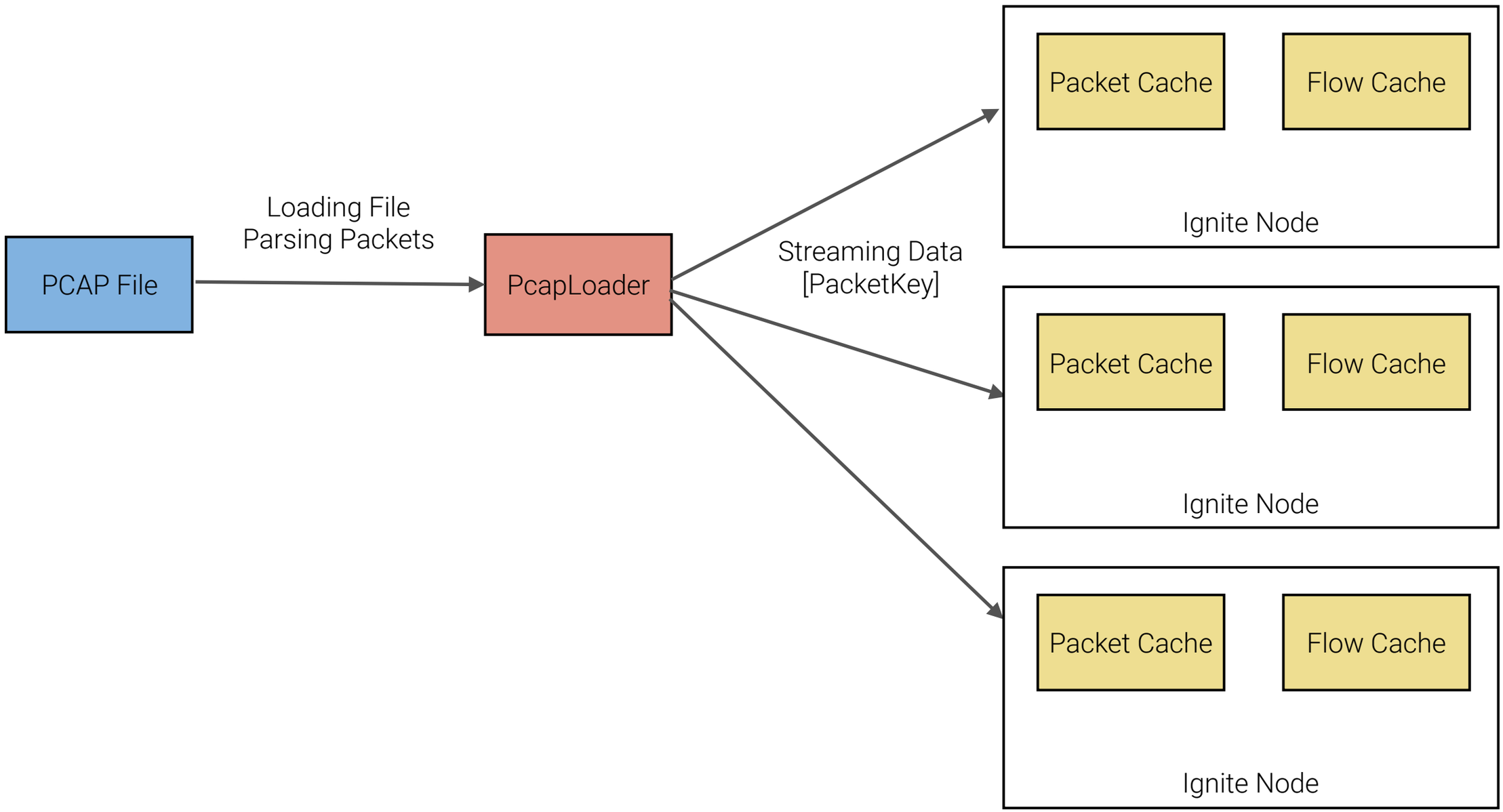




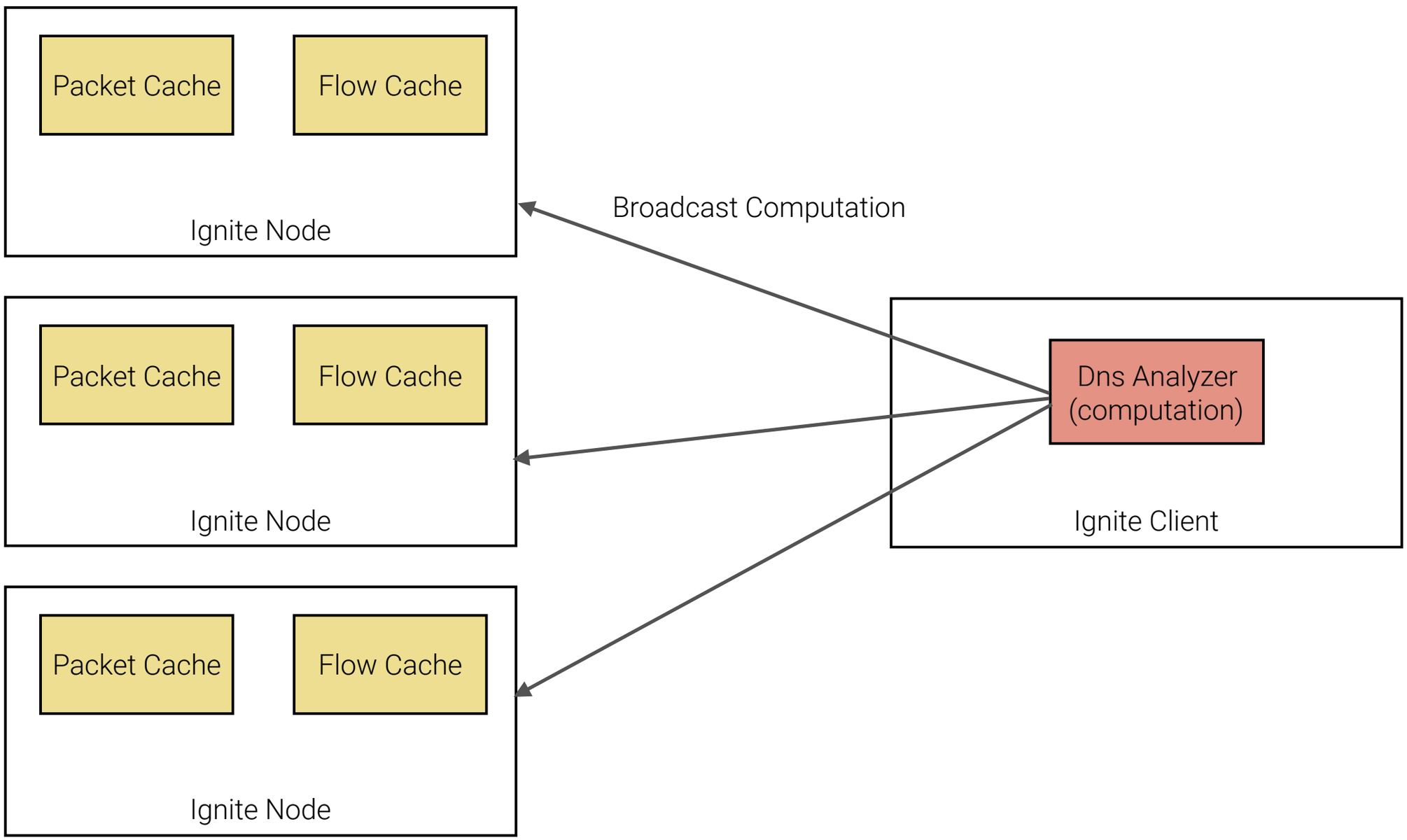
Apache Ignite



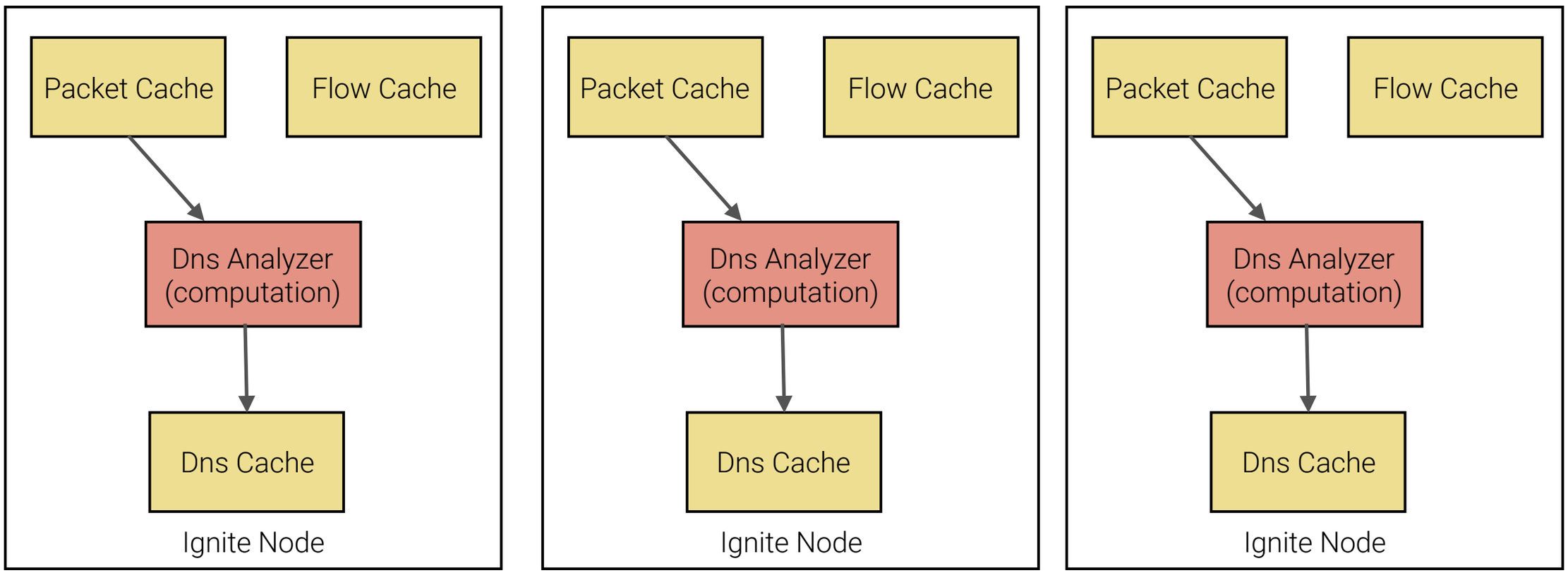
Ingesting data



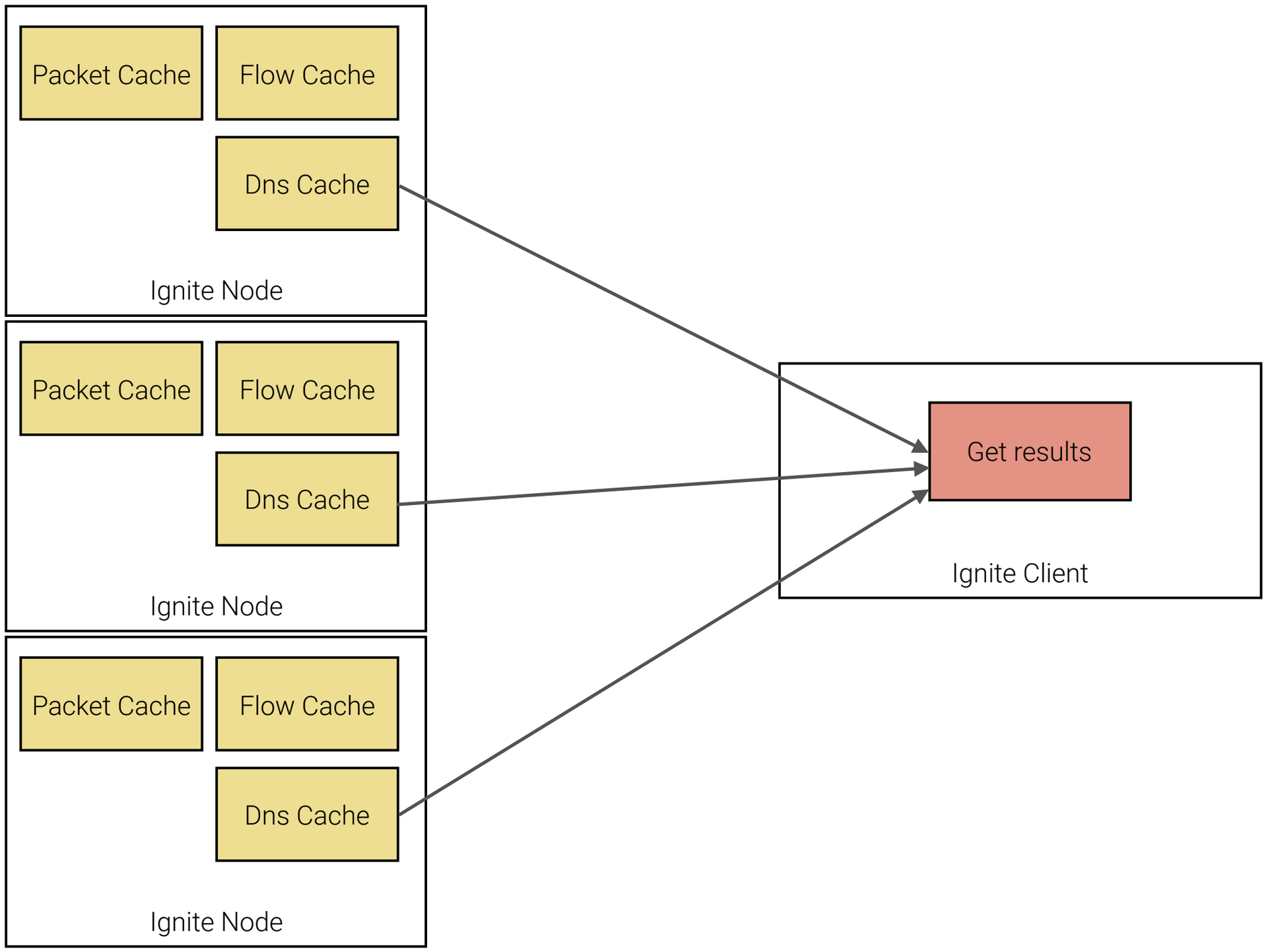
Broadcast computation



Executing computation

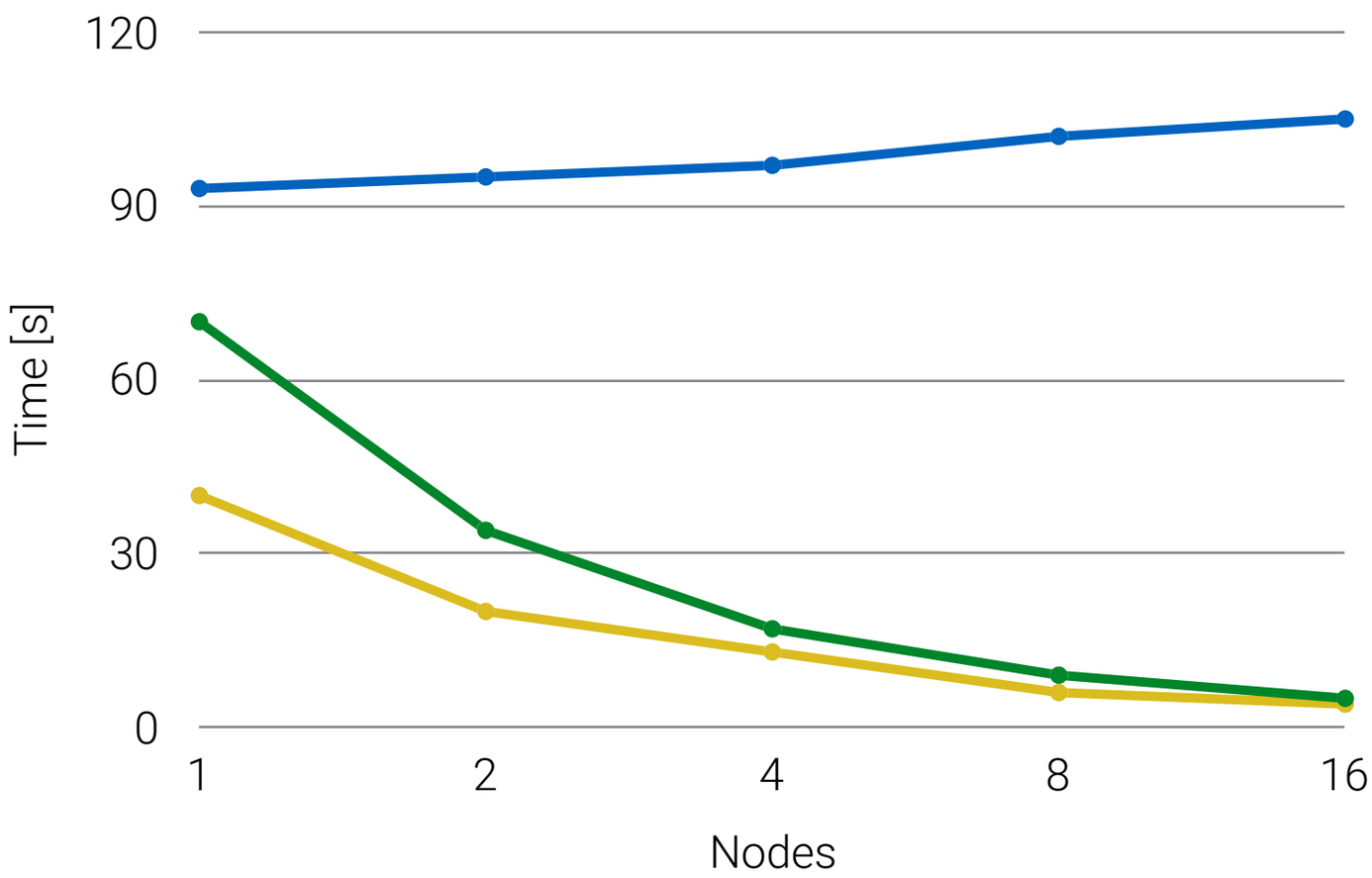


Get results


Packet Cache
Flow Cache
Dns Cache
Ignite Node
Packet Cache
Flow Cache
Dns Cache
Ignite Node
Packet Cache
Flow Cache
Dns Cache
Ignite Node
Get results
Ignite Client

Some numbers…

Ignite Nodes (Local)	Ingesting Frames	Tracking Flows	Extracting DNS
1	93s	71s	39s
2	95s	34s	22s
4	97s	17s	13s
8	102s	9s	6s
16	105s	5s	4s



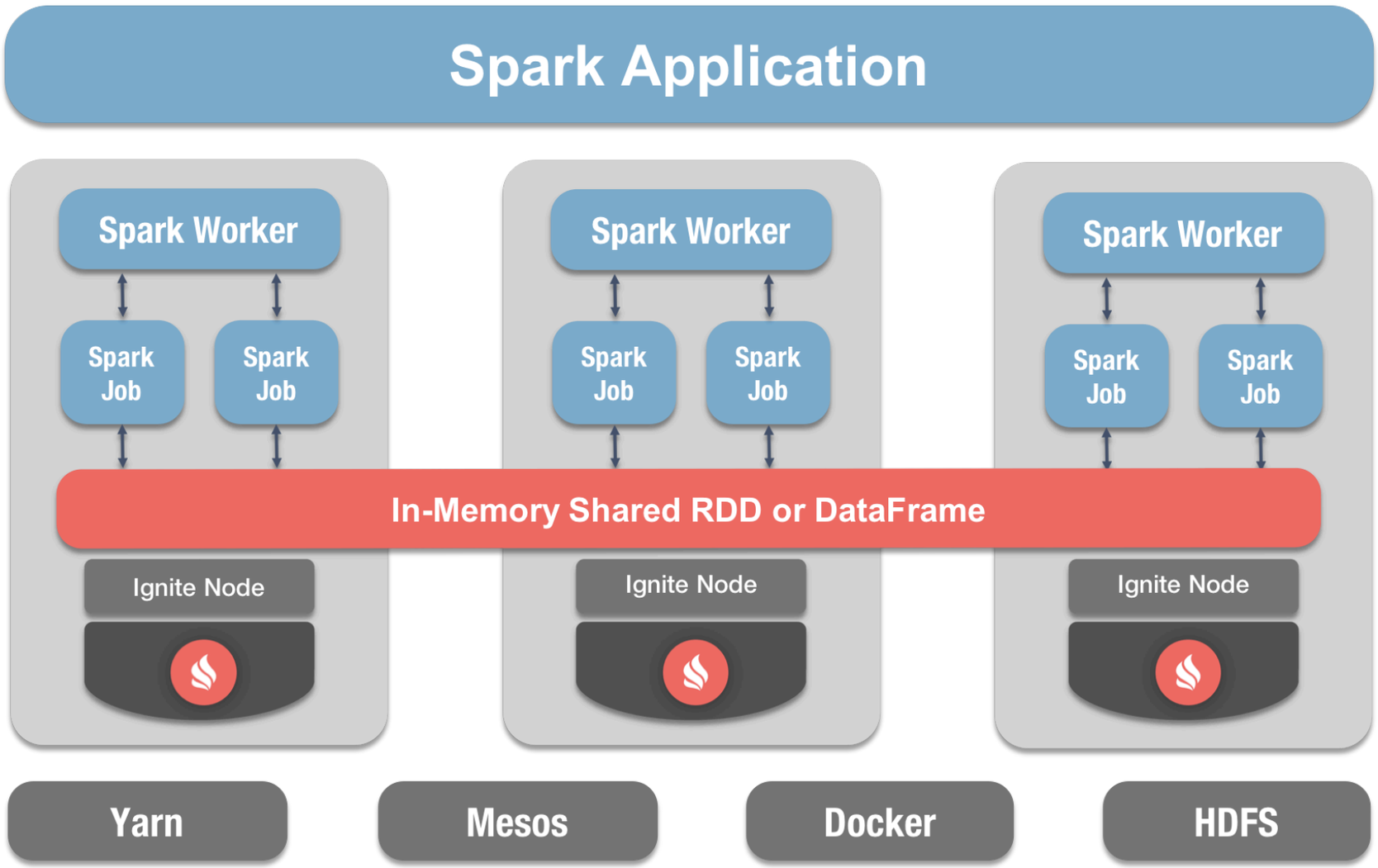
File size ~ 4GB

Observations

- Apache Spark is suitable for a workload that can be represented as a processing pipeline (DAG).
  - It works well with HDFS.
  - It scales quite well.
- Apache Ignite can do an arbitrary computation on data.
  - Data collocation computations
  - Sending computation not data.
- Bottleneck - data ingestion, once the data are in HDFS or memory fabric, the computation is fast.

Spark + Ignite

- Share state and data among Spark jobs
- No data movement
- In-place query execution
- Boost DataFrame and SQL Performance
- SQL on top of RDDs



Summary

- Integrated platform is a collection of various tools for several digital forensic domains.
- Their integration is only possible thanks to the use of state-of-the-art technology for service virtualization.
- Several case studies demonstrate the implemented functionality and considered usage scenarios.
  TO BE SEEN THIS AFTERNOON
- The platform is open for other extensions and customisations depending on the specific needs.