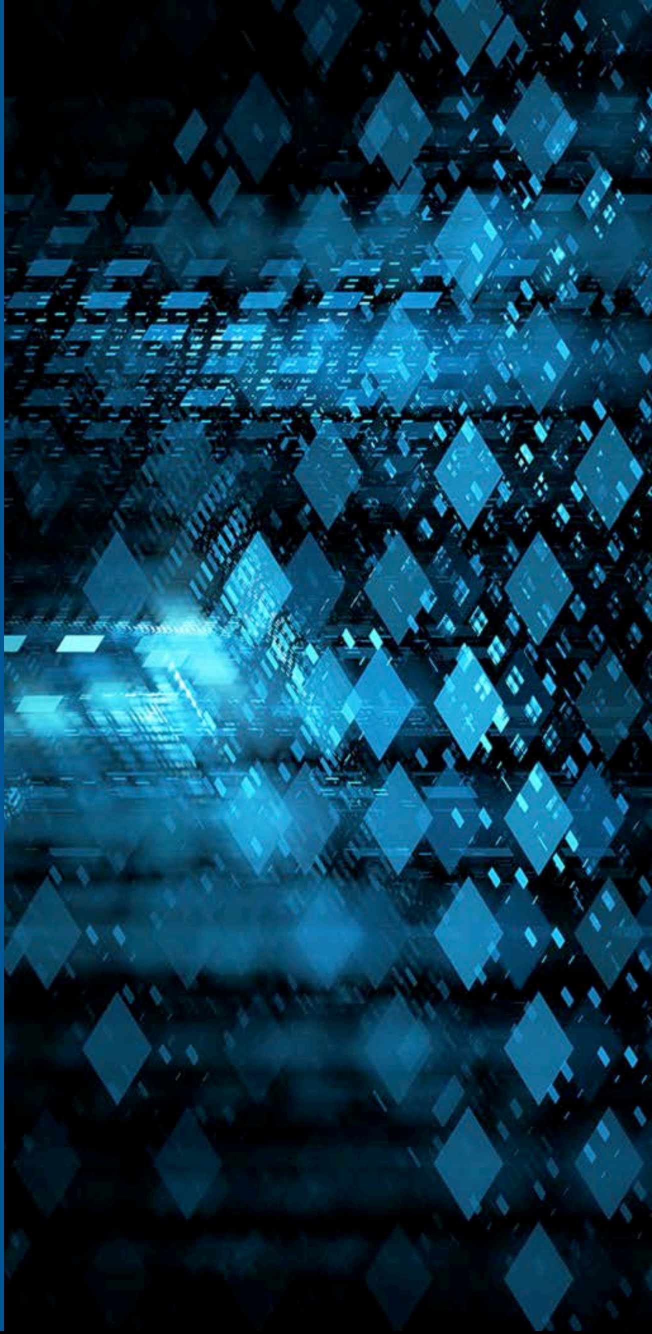
Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Tony Cebzanov

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



Agenda

- Introduction
- Architecture and Design
- Demonstration
- Future Work

Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Introduction

In the beginning... there was Netflow

Netflow was designed to retain the most important attributes of network conversations between TCP/IP endpoints on large networks without having to collect, store, and analyze all of the network's packet-level data

- For many years, this has been the most effective way to perform security analysis on large networks
- Over time, demand has increased for a platform that can support analytical workflows that make use of attributes beyond the transport layer

Modern flow collectors can export payload attributes at wire speed

- The challenge is scalable storage and analysis
- The current generation of distributed data processing platforms provides tools to address this challenge

Project Goals

The Mothra security analysis platform enables scalable analytical workflows that extend beyond the limitations of conventional flow records.

With the Mothra project, we aim to:

- facilitate bulk storage and analysis of cybersecurity data with high levels of flexibility, performance, and interoperability
- reduce the engineering effort involved in developing, transitioning, and operationalizing new analytics
- serve all major constituencies within the network security community, including data scientists, first-tier incident responders, system administrators, and hobbyists

SiLK: The Next Generation?

Mothra **is not** the next version of SiLK

- SiLK's design philosophy was inspired by UNIX
  - Command-line tools that each focus on doing one thing well
  - Tools are composable into analytics via shell scripting
  - Fixed-length record formats for optimal performance
- With larger, variable-length records, this design can't scale
  - Solution? Throw more hardware at the problem ("big data")

We view SiLK and Mothra as complementary projects that will be developed in parallel for the foreseeable future

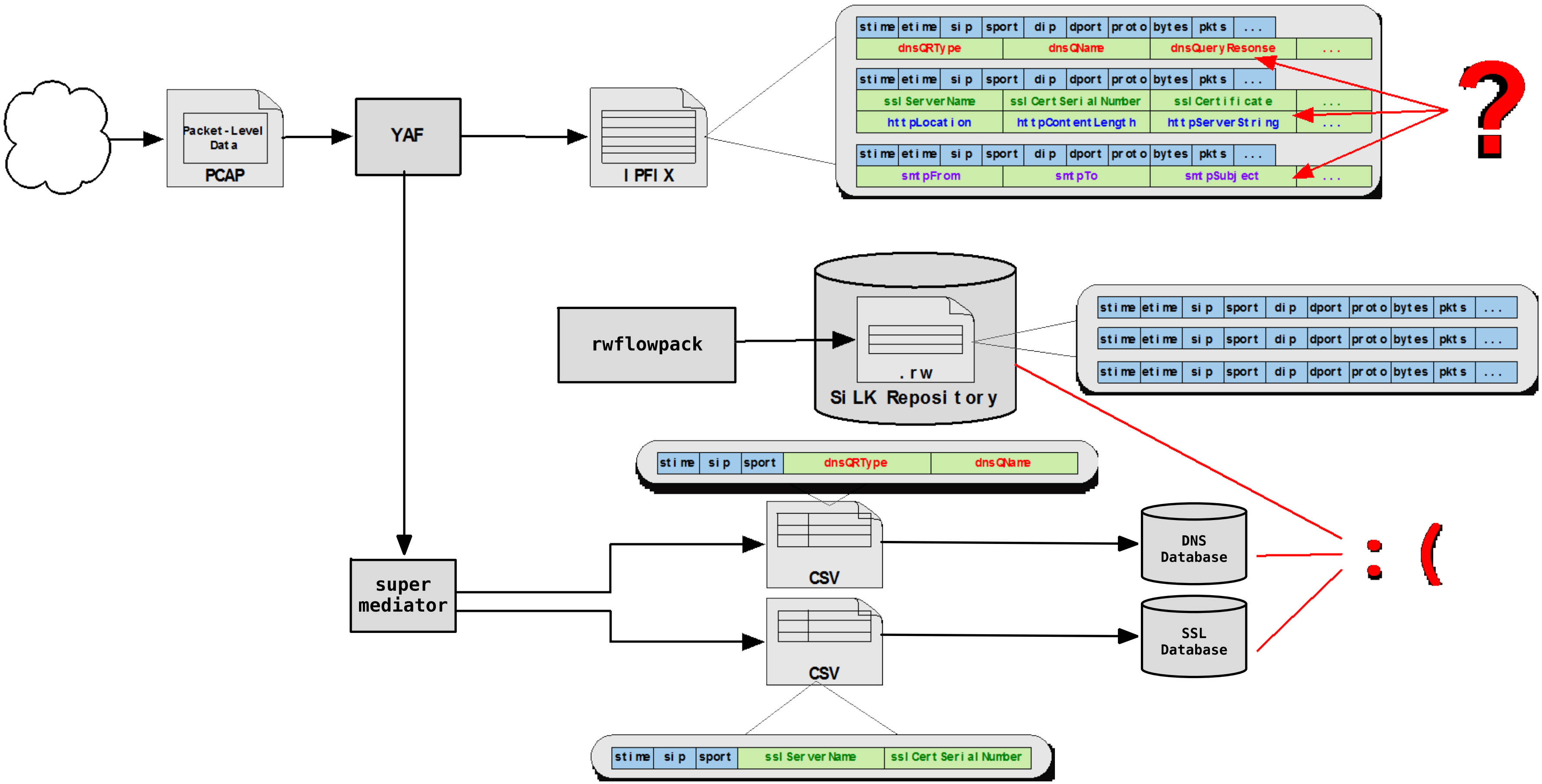
- SiLK still performs well for queries that don't look beyond layer 4
- Mothra enables more complex analyses at a scale beyond the capability of SiLK's single-node architecture

Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

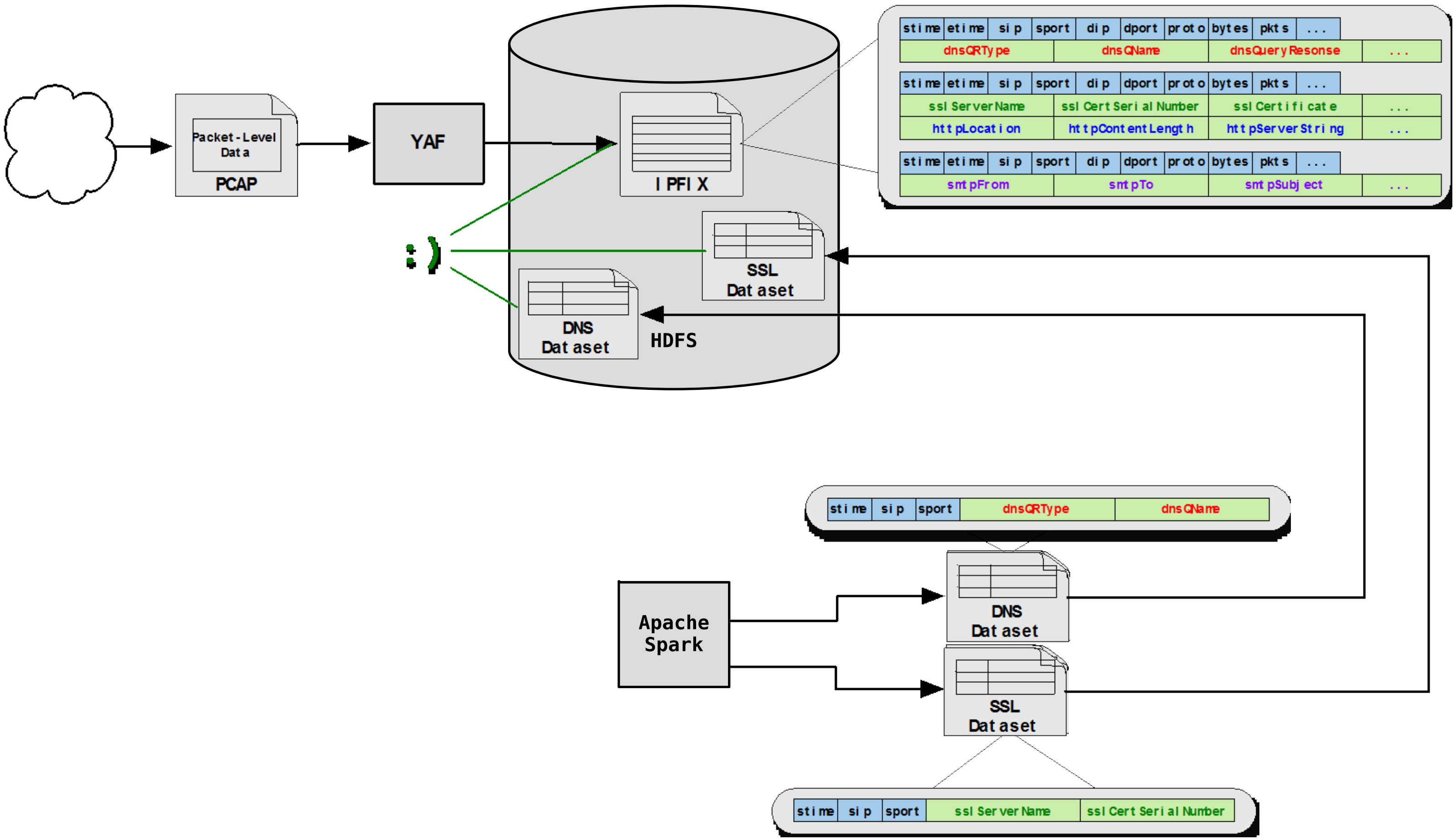
Architecture and Design

CERT | Software Engineering Institute | Carnegie Mellon University

YAF to SiLK Data Flow



YAF to Mothra Data Flow

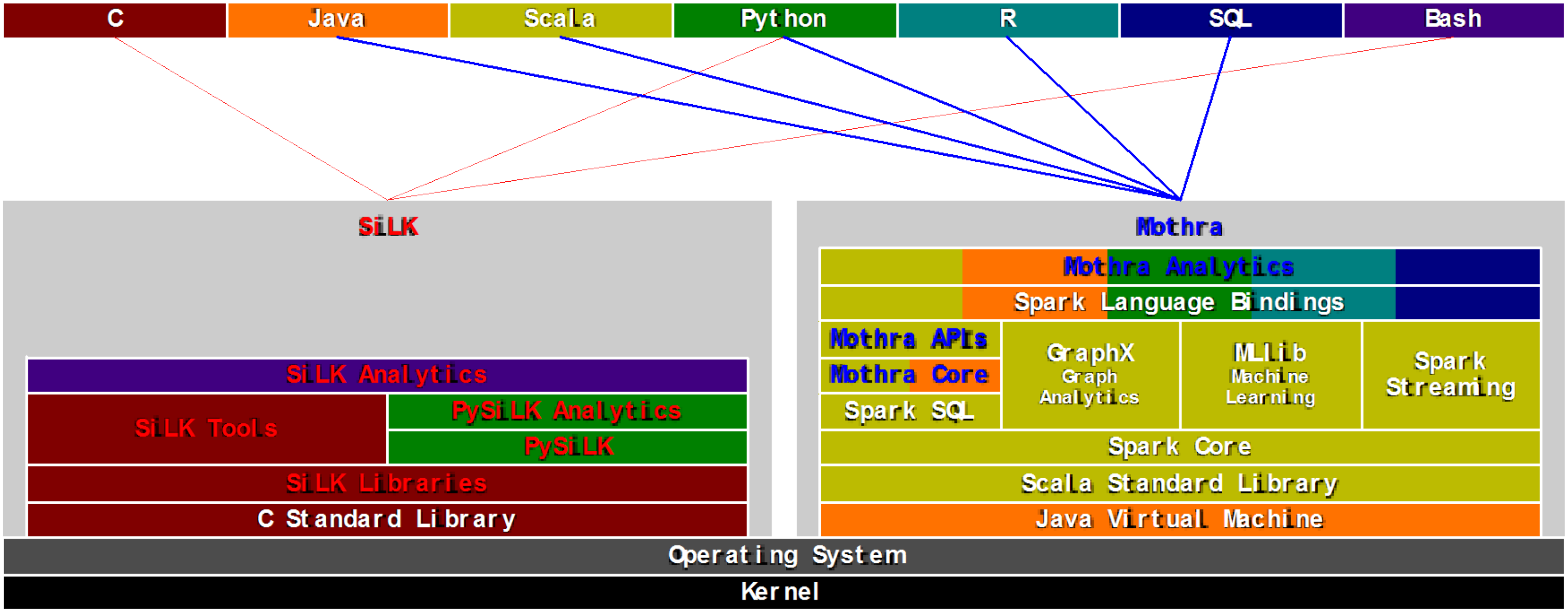


SiLK vs. Mothra

Mothra departs from SiLK's UNIX-like design in significant ways:

- SiLK
  - Command-line tools, mostly written in C, with some Python
  - Analytics are written as UNIX shell scripts
- Mothra
  - Built on Apache Spark, a cluster computing framework
    - Written primary in Scala, which runs on the Java Virtual Machine
    - Language bindings for Java, Python, R, and SQL
    - Runs standalone or on an existing cluster platform (e.g. Hadoop)
  - Mothra's core libraries are written in Scala and Java
  - Analytics can be written using any language Spark supports
  - A web notebook interface is provided for developing analytics

Platform Languages and Technologies (continued)



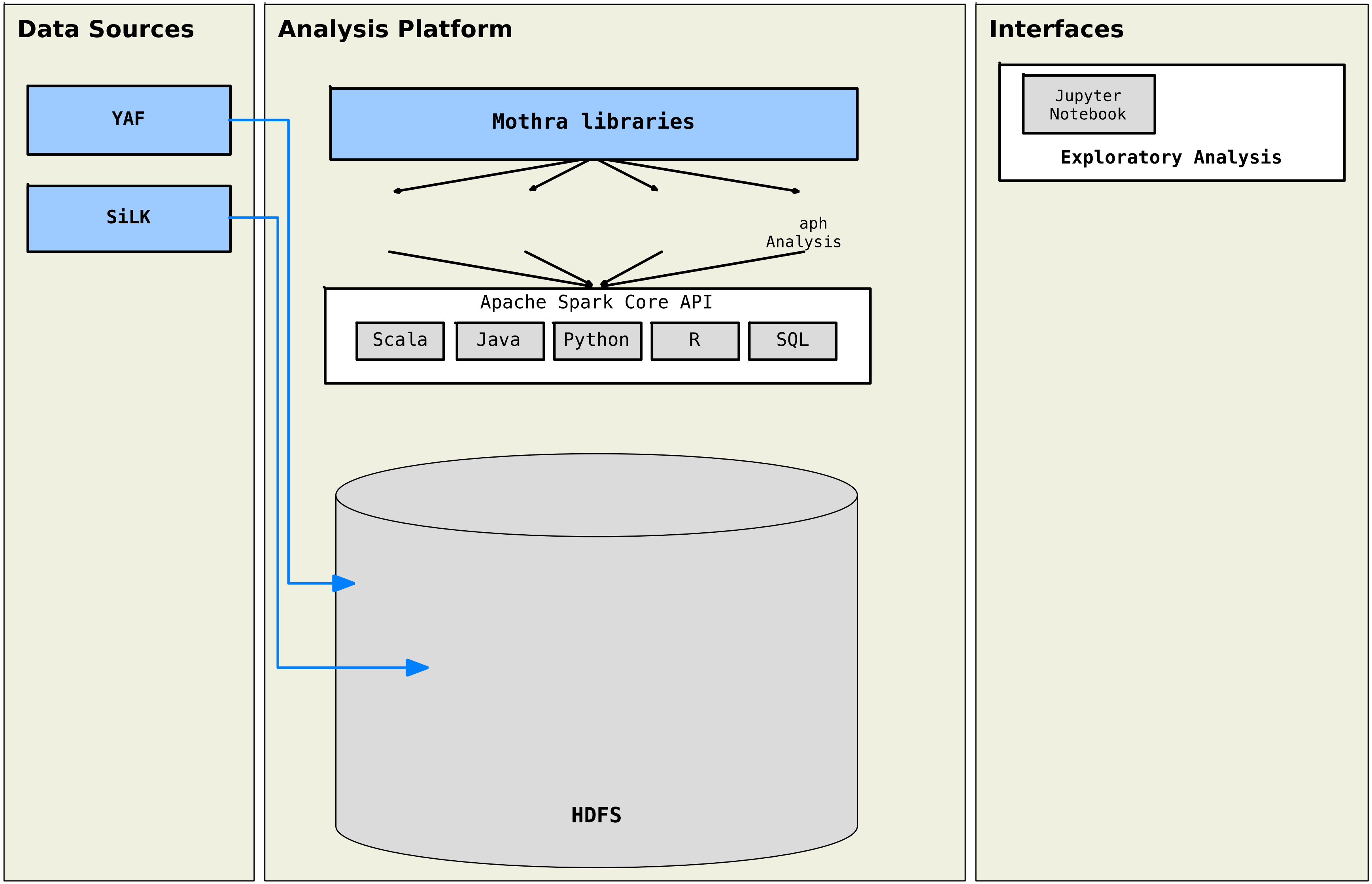
Why Spark?

Building Mothra on an established platform like Spark, with its active industry-sponsored open source development community, allows us to focus on components that deliver value to analysts.

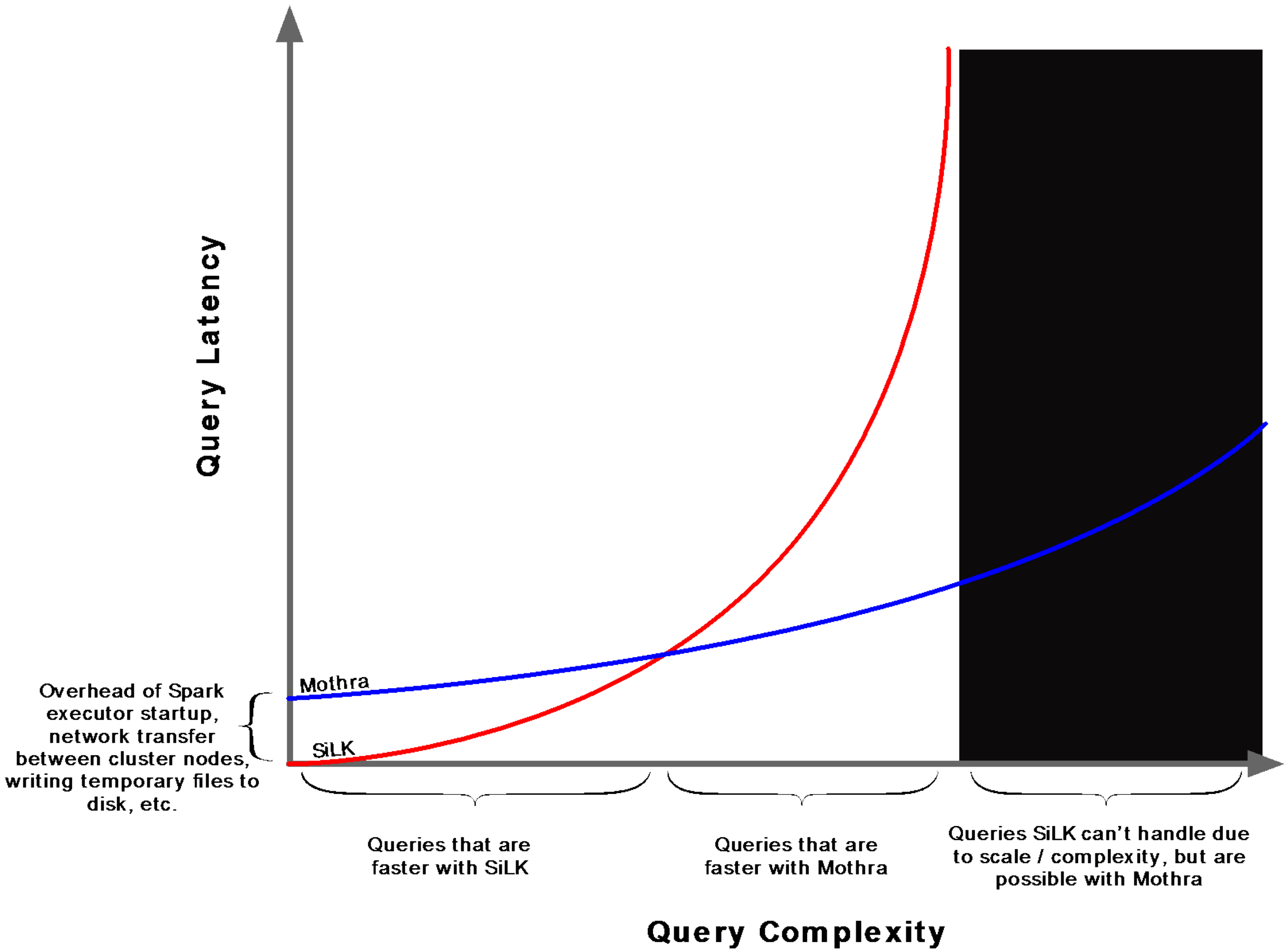
The Spark platform:

- enables a degree of scalability not possible with SiLK
- supports higher-level languages for faster development and transition of core functionality and analytics
- provides consistent interfaces to a variety of data sources
- includes libraries for graph analysis and machine learning
- integrates well with other big data platforms and technologies

Mothra Architecture



SiLK vs. Mothra Scalability



User Interfaces

Mothra uses Jupyter Notebook as an exploratory analysis UI:

- Rich web-based interface
  - Input cells for developing and executing code
  - Output cells display analysis results, including visualizations
  - Markdown for annotations with rich text
- Simple sharing and publishing of analytics and results
- Less daunting for novices to learn than the UNIX command-line

For CLI fans, **`jupyter-console`** and **`spark-shell`** are available

Jupyter Notebook

jupyter mothra-demo.platform.cooked Last Checkpoint: 10/31/2016 (unsaved changes)

File Edit View Insert Cell Kernel Navigate Help | Apache Toree - Scala

Markdown CellToolbar

3.3.4 Filtering

Note that there are a lot of `null` values in the SSL columns. Let's filter those out and display only the flows that have SSL fields:

In [11]:
```
https_flows.filter($"sslCertificate".isNotNull).show()
```

sip	dip	sport	dport	proto	packets	bytes	sslCipher	sslCertificate
192.168.202.80	192.168.28.103	54151	443	6	6	412	[22, 19, 10, 102,...	[MD8BPzA/AQgCCQA/...
192.168.202.80	192.168.28.103	54153	443	6	12	1036	[5, 4, 65664, 2, ...	[MD8BPzA/AQgCCQA/...
192.168.202.76	192.168.22.254	61187	443	6	11	1359	[4, 5, 47, 51, 50...	[MD8BPzA/ARI/AwIB...
192.168.202.76	192.168.22.254	61222	443	6	11	1359	[4, 5, 47, 51, 50...	[MD8BPzA/ARI/AwIB...
192.168.202.80	192.168.28.253	54380	443	6	6	412	[22, 19, 10, 102,...	[MD8EDjA/A3c/AwIB...
192.168.202.80	192.168.28.253	54381	443	6	6	654	[5, 4, 65664, 2, ...	[MD8EDjA/A3c/AwIB...

3.3.5 Aggregation

The syntax for aggregate queries is relatively simple. In this query, we'll group the `https_flows` DataFrame by destination IP address with `https_flows.groupBy($"dip")`, calculating the average packets and bytes with `.avg("packets", "bytes")`, and sort the resulting rows by the average byte count with `.sort($"avg(bytes)").desc)`.

In [12]:
```
https_flows.groupBy($"dip").avg("packets", "bytes").sort($"avg(bytes)".desc).show()
```

dip	avg(packets)	avg(bytes)
192.168.28.253	6.0	470.6666666666667
192.168.28.103	6.0	448.5
192.168.22.254	4.571428571428571	448.2857142857143
192.168.4.86	9.0	432.0
157.55.130.153	3.0	152.0
65.55.223.14	3.0	152.0
157.56.52.12	3.0	152.0
64.4.23.154	3.0	152.0
157.56.52.28	3.0	152.0
194.165.188.82	3.0	152.0
157.55.56.157	3.0	152.0
64.4.23.140	3.0	152.0
213.199.179.158	3.0	152.0
157.55.56.140	3.0	152.0
10.21.6.40	3.0	152.0
157.55.56.143	3.0	152.0
157.55.130.159	3.0	152.0
157.55.56.152	3.0	152.0
64.4.23.145	3.0	152.0
149.5.45.140	2.0	104.0

only showing top 20 rows

Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Demonstration

Field Specifier Syntax

IPFIX fields are specified using strings of the following format:

`[path][/][operator]name[:format][=alias]`

where:

- `path` (optional) is a path to the desired information element
  - paths are made up of template names, delimited by `/` characters
  - if `path` is empty, the field specifier will look for the given element in all top-level IPFIX records
- `operator` (optional) is a character indicating how the information element should be treated
  - currently, the only operator is @, indicating that the IE is a basicList field
- `name` (required) is the name of the IPFIX informtaion element
- `format` (optional) is a string indicating how the field should be formatted in the data frame
  - current formats are:
    - `str` – format IE as a string
    - `iso8601` – format IE as an ISO-8601 date string
    - `base64` – format IE as as base64-encoded string
- `alias` (optional) is a string to be used for the data frame column name instead of the IPFIX IE name
  - if `alias` is unspecified, the column name will default to the IPFIX IE name

Field Spec Examples

- **`flowStartMilliseconds`**
  - the flow start time in milliseconds at the top level of any IPFIX record
- **`flowStartMilliseconds:iso8601`**
  - same, but formatted as an ISO-8601 string
- **`flow_total_ip6/flowStartMilliseconds:iso8601`**
  - same, but only if the top-level record matches the IPv6 template
- **`flow_total_ip6/flowStartMilliseconds:iso8601=stime`**
  - same, but rename the field to stime
- **`/http/@httpUserAgent`**
  - the HTTP user agent list within the http template
- **`/ssl_cert_full/@sslCertificate:base64`**
  - a list of SSL certificates in the flow, each base64-encoded

Spark DataFrames

From the Spark documentation:

- "A DataFrame is a distributed collection of data organized into named columns. **It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood**."

To build a DataFrame in Mothra with the Scala language interface:

```
In [4]: val input_df = spark_ipfix.load(sqlContext, input_data, all_fields)
```

To count the number of records:

```
In [5]: input_df.count()
Out[5]: 30299
```

Queries

Spark DataFrame API maps reasonably well to `rw*` commands:

- Filtering (`rwfilter`)

```
In [6]: var https_flows = input_df.filter($"dport" === 443)
```

- Syntax is similar with the Python API:

```
In [48]: https_flows = df.filter(df["sport"] == 443)
         https_flows.show()
```

Queries

- Column selection & display (`rwcut`)

In [8]:

```
https_flows = https_flows.select(
    "sip", "dip", "sport", "dport",
    "proto", "packets", "bytes", "sslCipher", "sslCertificate")
https_flows.show()
```

```
+-------------+-------------+-----+-----+-----+-------+-----+--------------------+--------------------+
|          sip|          dip|sport|dport|proto|packets|bytes|           sslCipher|      sslCertificate|
+-------------+-------------+-----+-----+-----+-------+-----+--------------------+--------------------+
|192.168.202.80|192.168.28.254|53708|  443|    6|      2|   84|                null|                null|
|192.168.202.80|192.168.28.254|53709|  443|    6|      2|   84|                null|                null|
|192.168.202.80|192.168.28.254|53710|  443|    6|      2|   84|                null|                null|
|192.168.202.80|192.168.28.254|53711|  443|    6|      2|   84|                null|                null|
|192.168.202.83|192.168.206.44|58624|  443|    6|      1|   60|                null|                null|
|192.168.202.83|192.168.206.44|58628|  443|    6|      1|   60|                null|                null|
|192.168.202.80|192.168.28.103|54128|  443|    6|      5|  294|                null|                null|
|192.168.202.80|192.168.28.103|54151|  443|    6|      6|  412|[22, 19, 10, 102,...|[MD8BPzA/AQgCCQA/...|
|192.168.202.80|192.168.28.103|54153|  443|    6|     12| 1036|[5, 4, 65664, 2, ...|[MD8BPzA/AQgCCQA/...|
|192.168.202.76|192.168.22.254|61187|  443|    6|     11| 1359|[4, 5, 47, 51, 50...|[MD8BPzA/ARI/AwIB...|
```

Queries (continued)

- Sorting:

```
In [10]: https_flows.sort($"bytes".desc).show()
```

- Aggregation:

```
In [12]: https_flows.groupBy($"dip").avg("packets", "bytes")
                   .sort($"avg(bytes)".desc).show()
```

```
+---------------+----------------+-----------------+
|            dip|    avg(packets)|       avg(bytes)|
+---------------+----------------+-----------------+
| 192.168.28.253|             6.0|470.6666666666667|
| 192.168.28.103|             6.0|            448.5|
| 192.168.22.254|4.571428571428571|448.2857142857143|
|   192.168.4.86|             9.0|            432.0|
| 157.55.130.153|             3.0|            152.0|
|   65.55.223.14|             3.0|            152.0|
```

Queries (continued)

- Full SQL syntax

In [17]:

```
input_df.registerTempTable("df")
sqlContext.sql("""SELECT dnsQName,
                      AVG(packets) AS avg_packets,
                      SUM(packets) AS sum_packets,
                      AVG(bytes) AS avg_bytes,
                      SUM(bytes) AS sum_bytes
                FROM df
                WHERE dnsQName IS NOT NULL
                GROUP BY dnsQName
                ORDER BY sum_bytes DESC
                """).show()
```

dnsQName	avg_packets	sum_packets	avg_bytes	sum_bytes
version.bind.	5.372093023255814	231	321.86046511627904	13840
www.apple.com.	4.0	16	236.0	944
www.squid-cache.org.	1.0	8	65.0	520
time.apple.com.	4.0	8	240.0	480
http.	2.0	6	140.0	420
44.206.168.192.in...	1.0	3	73.0	219
fs-1.one.ubuntu.com.	1.0	2	65.0	130

Queries (continued)

- Compound query example
  - Build dataframe of SSL flows with a known bad SSL cert
  - Build dataframe of DNS flows with non-null qname
  - Join two dataframes on the `sip` field
  - Select the `sip`, `sslCertificate`, and `dnsQName` fields

In [19]:
```
(
    https_flows.filter(array_contains($"sslCertificate", bad_ssl_cert))
    .join(dns_flows.filter($"dnsQName".isNotNull), "sip")
).select("sip", "sslCertificate", "dnsQName").show()
```

```
+--------------+--------------------+-------------+
|           sip|      sslCertificate|     dnsQName|
+--------------+--------------------+-------------+
|192.168.202.80|[MD8BPzA/AQgCCQA/...|version.bind.|
|192.168.202.80|[MD8BPzA/AQgCCQA/...|version.bind.|
|192.168.202.80|[MD8BPzA/AQgCCQA/...|version.bind.|
```

Mothra: A Large-Scale Data Processing Platform for Network Security Analysis

Future Work

Future Work

On the horizon:

- Streaming ingest from sensors
- Operational analyst console integration
- Simplified deployment and configuration
- Open source release
- Integration of useful components into other FOSS projects

Related Projects

Apache Metron (incubating)

- Sponsored by Hortonworks, Rackspace, Cisco, and others

Apache Spot (incubating):

- Sponsored by Cloudera, Intel, EBay, and others

Similar in scope and scale, different in emphasis and design

As these projects mature and grow in popularity, we may pursue integration opportunities

Questions?

CERT NetSA Tools Home
http://tools.netsa.cert.org

Contact
netsa-help@cert.org
tonyc@cert.org

Mailing list
netsa-tools-discuss@cert.org