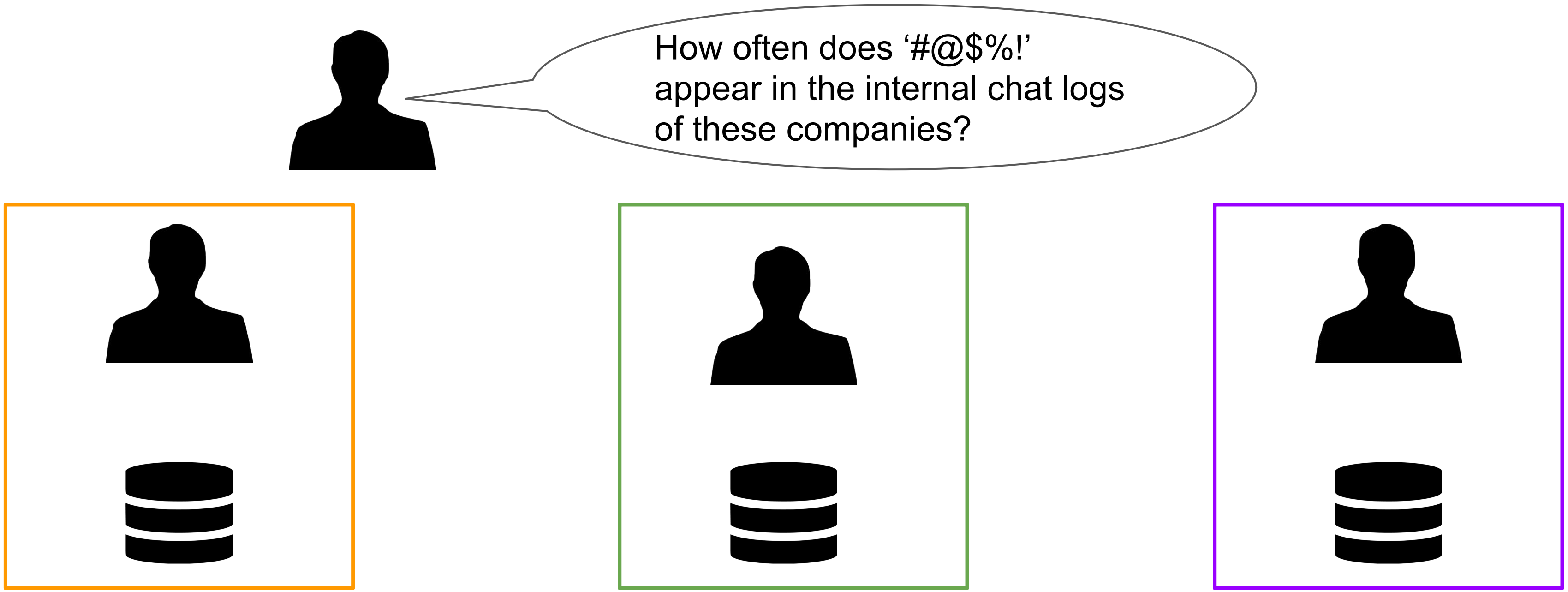
Integrating Multi-Party Computation in Big Data Workflows

Nikolaj Volgushev, Malte Schwarzkopf, Andrei Lapets, Mayank Varia, Azer Bestavros



How often does ‘#@$%!’ appear in the internal chat logs of these companies?

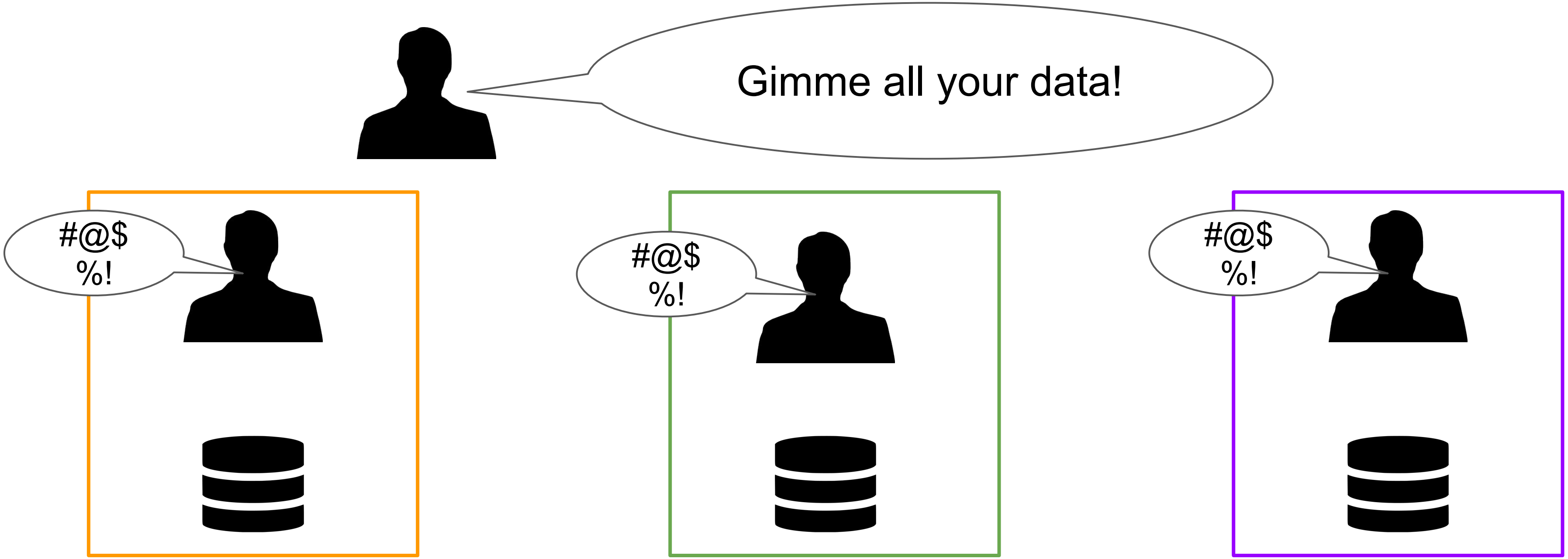
Sounds like a job for hadoop

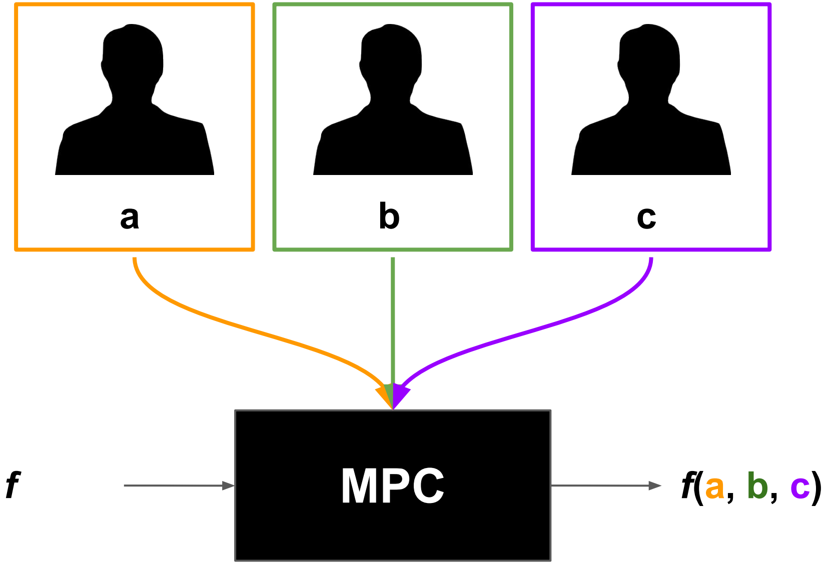
We’re talking Terabytes of data ⇒ a Python script won’t cut it.

Mode of operation: distribute data across many machines, process in parallel.

Programming paradigm: specify data analytics tasks in high-level language.

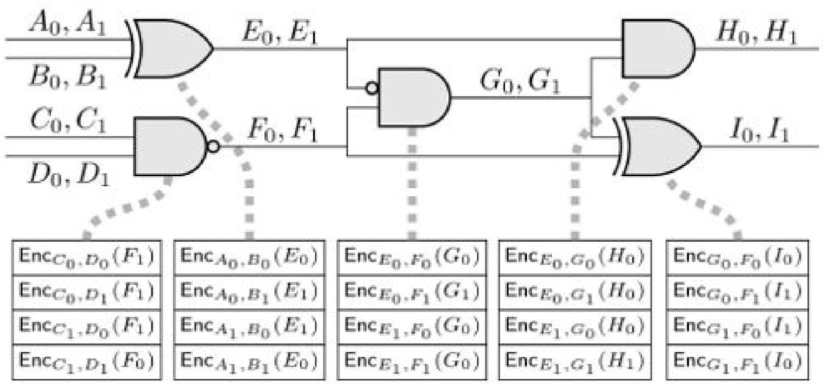
Backend infrastructure: cluster of machines.


Gimme all your data!
#@$ %!
#@$ %!
#@$ %!

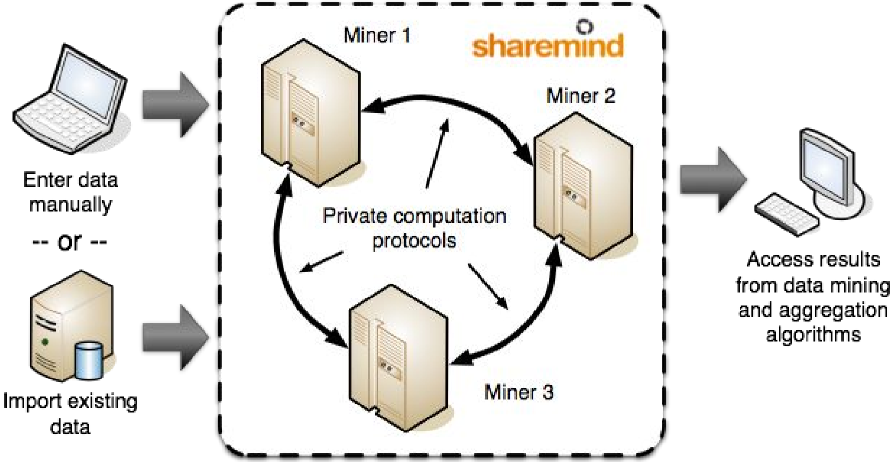


Multiparty computation (MPC) is a crypto tool for privacy preserving computation.

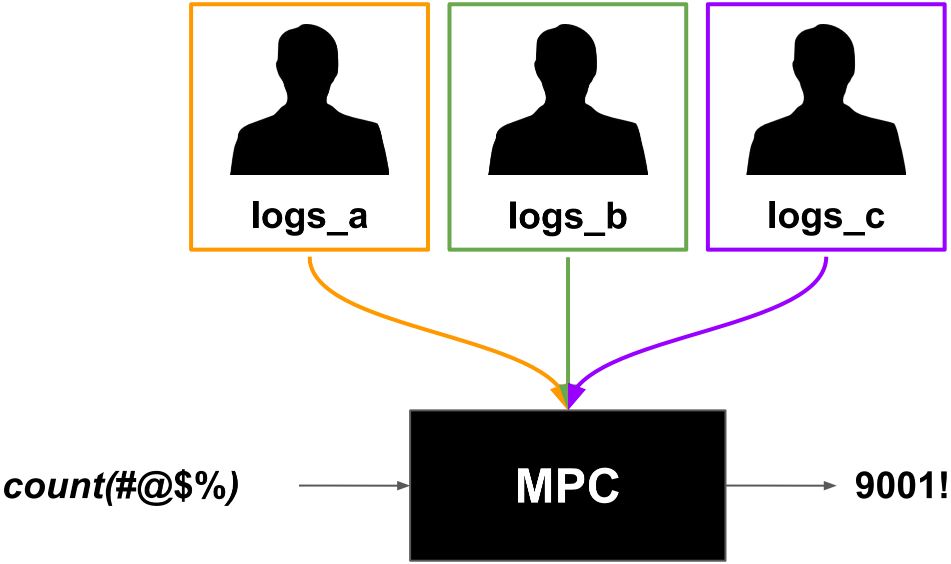
So much MPC!



$\mathsf{Enc}_{C_0,D_0}(F_1)$	$\mathsf{Enc}_{A_0,B_0}(E_0)$	$\mathsf{Enc}_{E_0,F_0}(G_0)$	$\mathsf{Enc}_{E_0,G_0}(H_0)$	$\mathsf{Enc}_{G_0,F_0}(I_0)$
$\mathsf{Enc}_{C_0,D_1}(F_1)$	$\mathsf{Enc}_{A_0,B_1}(E_1)$	$\mathsf{Enc}_{E_0,F_1}(G_1)$	$\mathsf{Enc}_{E_0,G_1}(H_0)$	$\mathsf{Enc}_{G_0,F_1}(I_1)$
$\mathsf{Enc}_{C_1,D_0}(F_1)$	$\mathsf{Enc}_{A_1,B_0}(E_1)$	$\mathsf{Enc}_{E_1,F_0}(G_0)$	$\mathsf{Enc}_{E_1,G_0}(H_0)$	$\mathsf{Enc}_{G_1,F_0}(I_1)$
$\mathsf{Enc}_{C_1,D_1}(F_0)$	$\mathsf{Enc}_{A_1,B_1}(E_0)$	$\mathsf{Enc}_{E_1,F_1}(G_0)$	$\mathsf{Enc}_{E_1,G_1}(H_1)$	$\mathsf{Enc}_{G_1,F_1}(I_0)$


Enter data manually
-- or --
Import existing data
Miner 1
sharemind
Miner 2
Private computation protocols
Miner 3
Access results from data mining and aggregation algorithms

So our data analyst should use MPC right?



Great in theory **but**...

Accessibility. MPC frameworks have a steep learning curve and don’t provide the high-level representations that data analysts use.

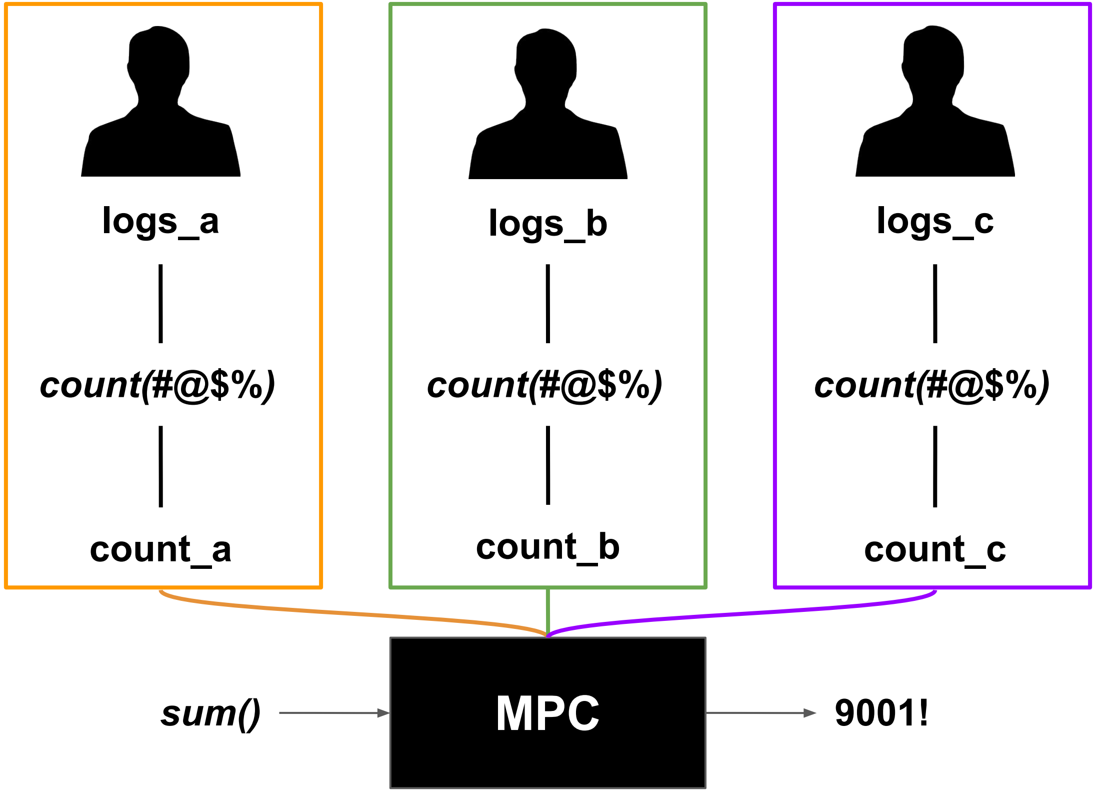
Scalability. MPC is slow.

Bottom line:

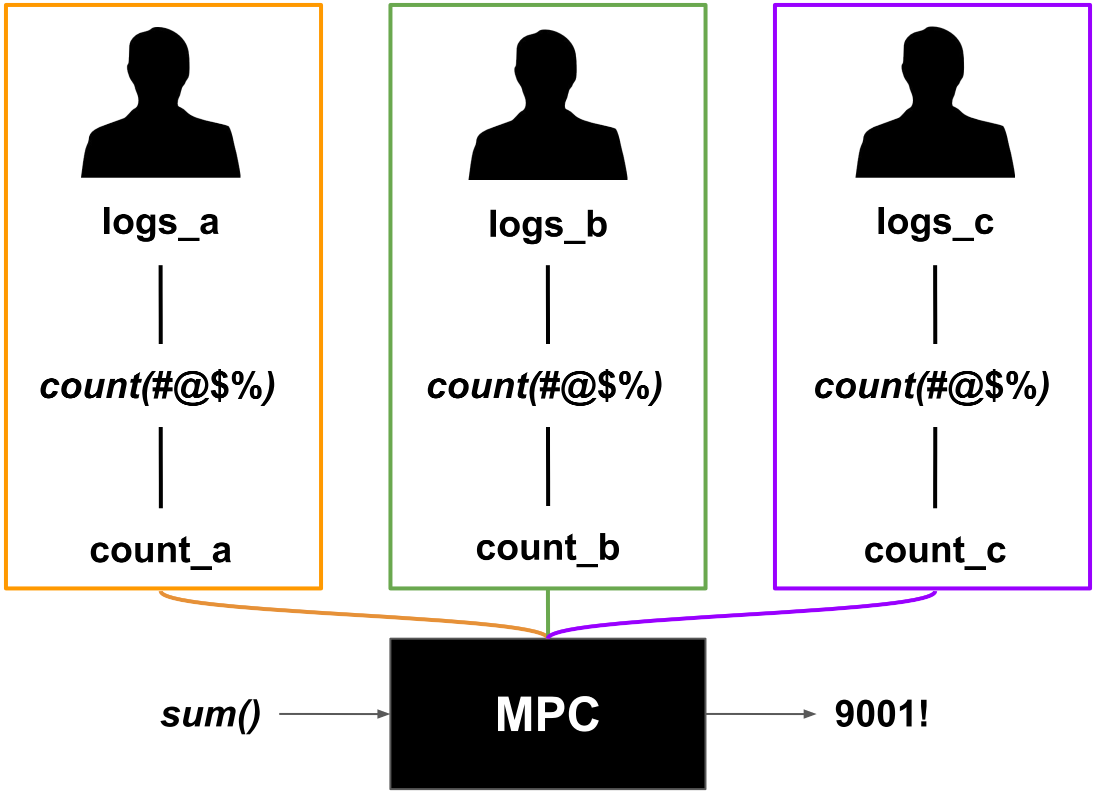
Our analyst probably doesn’t know MPC, or how to use it.

Any MPC framework is **far** too slow to process GBs of data.

What about a *hybrid* approach?



A **lot** of work and expert knowledge required



Good news!

We have just the system for you:

- Relational front-end language to specify workflow
- Automatic detection of which part of the workflow requires MPC
- Automatic code generation and execution
- Directive: “Do as much locally as possible.”
- Leverages existing frameworks as backends

The main components of our system

SQL-like programming language to specify analytics using standard relational operators.

Compiler that converts programs to jobs that are executable in existing data processing frameworks and MPC frameworks.

Dispatcher to execute the generated jobs automatically and seamlessly on the available backends.

Let’s explore top-down

SQL-like programming language to specify analytics using standard relational operators.

Compiler that converts programs to jobs that are executable in existing data processing frameworks and MPC frameworks.

Dispatcher to execute the generated jobs automatically and seamlessly on the available backends.

This is what the analyst writes

```
select count(log_message)
   from logs
 where log_message like'#@$%';
```

The main components of our system

SQL-like programming language to specify analytics using standard relational operators.

Compiler that converts programs to jobs that are executable in existing data processing frameworks and MPC frameworks.

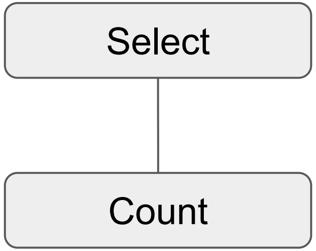
Dispatcher to execute the generated jobs automatically and seamlessly on the available backends.

Relational

```
select count(msg)
   from logs
 where msg like '#@$%';
```

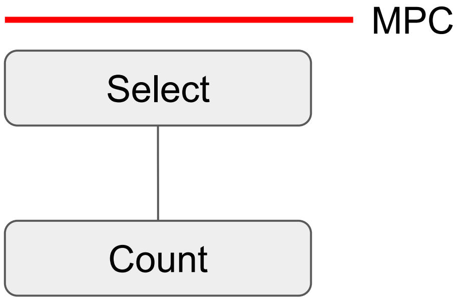
Relational ⇒ IR

```
select count(msg)
   from logs
 where msg like '#@$%';
```



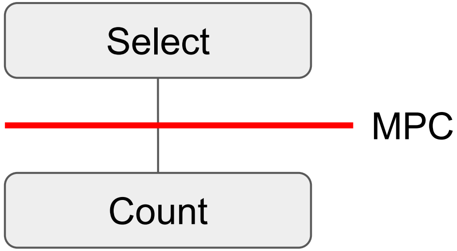
Relational ⇒ IR ⇒ MPC-IR

```
select count(msg)
   from logs
 where msg like '#@$%';
```



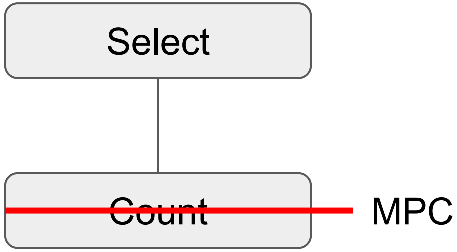
We don’t need MPC for selections

```
select count(msg)
   from logs
 where msg like '#@$%';
```



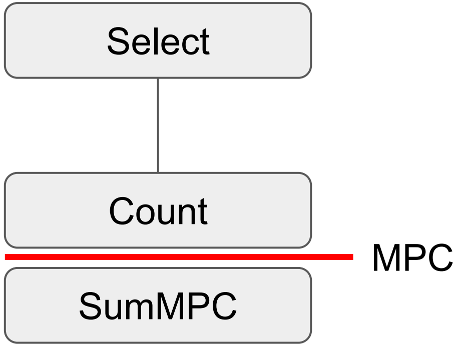
But what about aggregations?

```sql
select count(msg)
   from logs
 where msg like '#@$%';
```



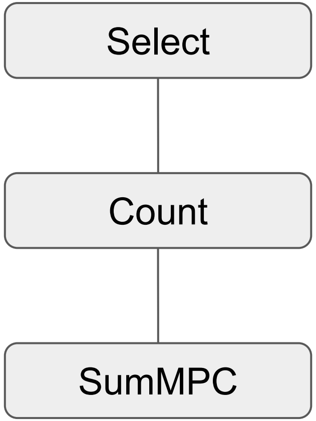
count(whole) = sum(count(parts))

```
select count(msg)
   from logs
 where msg like '#@$%';
```



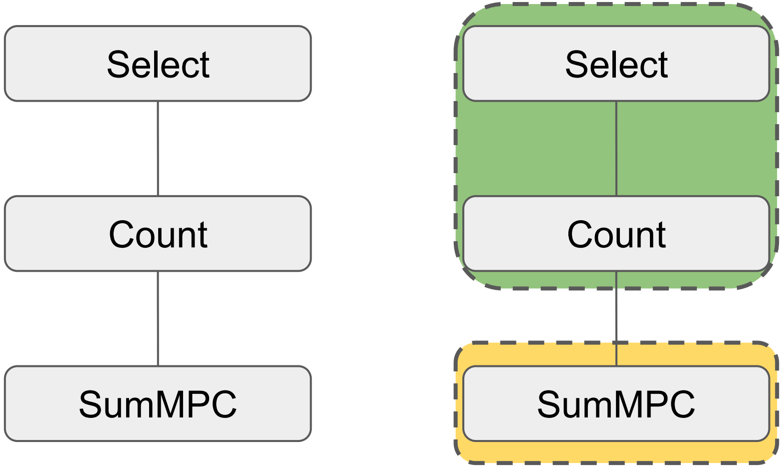
Relational ⇒ IR ⇒ MPC-IR

```
select count(msg)
   from logs
 where msg like '#@$%';
```



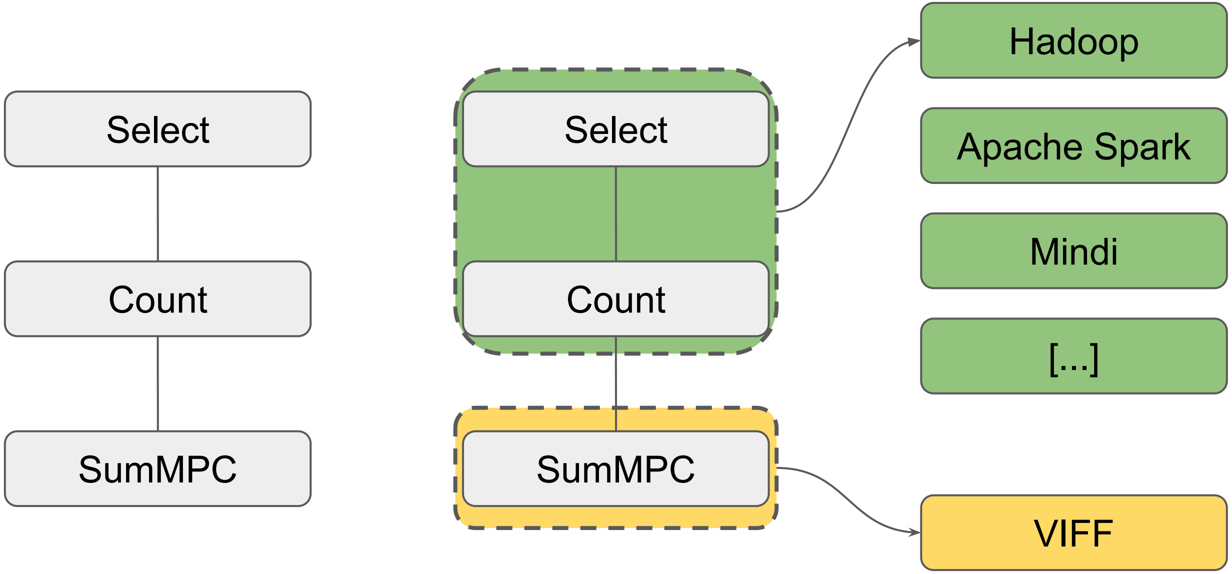
Relational ⇒ IR ⇒ MPC-IR ⇒ Partitions

```
select count(msg)
   from logs
 where msg like '#@$%';
```



Relational ⇒ IR ⇒ MPC-IR ⇒ Partitions ⇒ Backends

```
select count(msg)
   from logs
 where msg like '#@$%';
```



Select

Count

SumMPC

Select

Count

SumMPC

Hadoop

Apache Spark

Mindi

[...]

VIFF

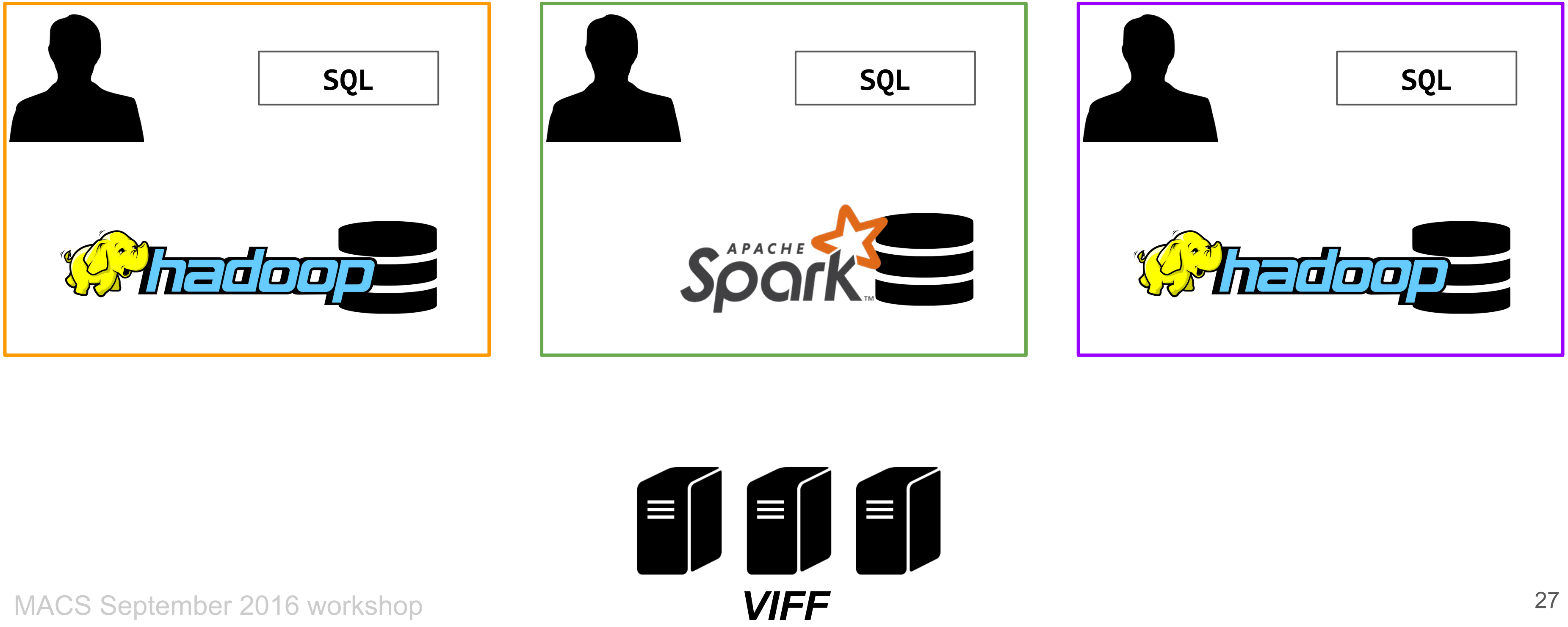
The main components of our system

SQL-like programming language to specify analytics using standard relational operators.

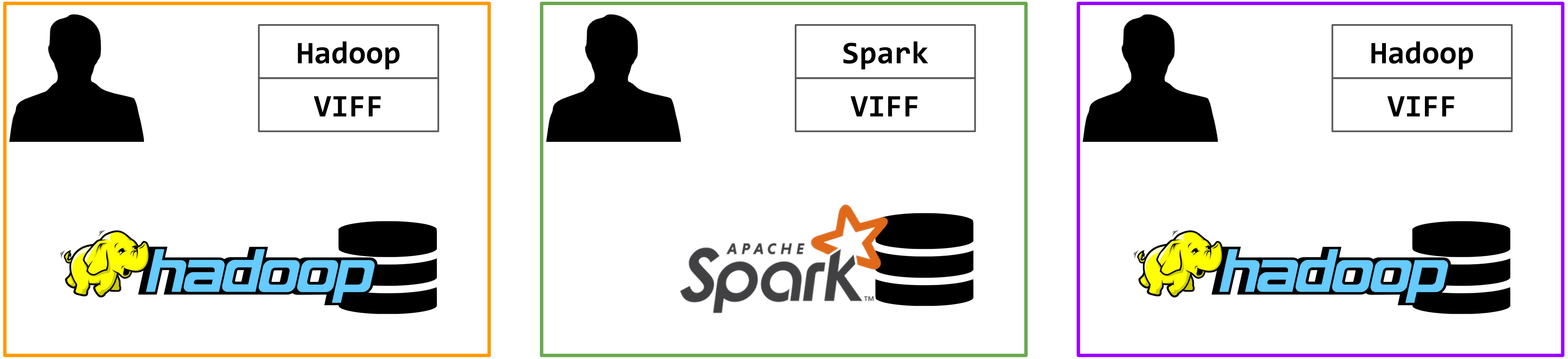
Compiler that converts programs to jobs that are executable in existing data processing frameworks and MPC frameworks.

Dispatcher to execute the generated jobs automatically and seamlessly on the available backends.

The baseline



Our system compiles programs into jobs



Hadoop

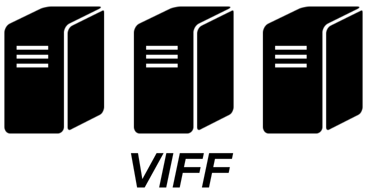
VIFF

Spark

VIFF

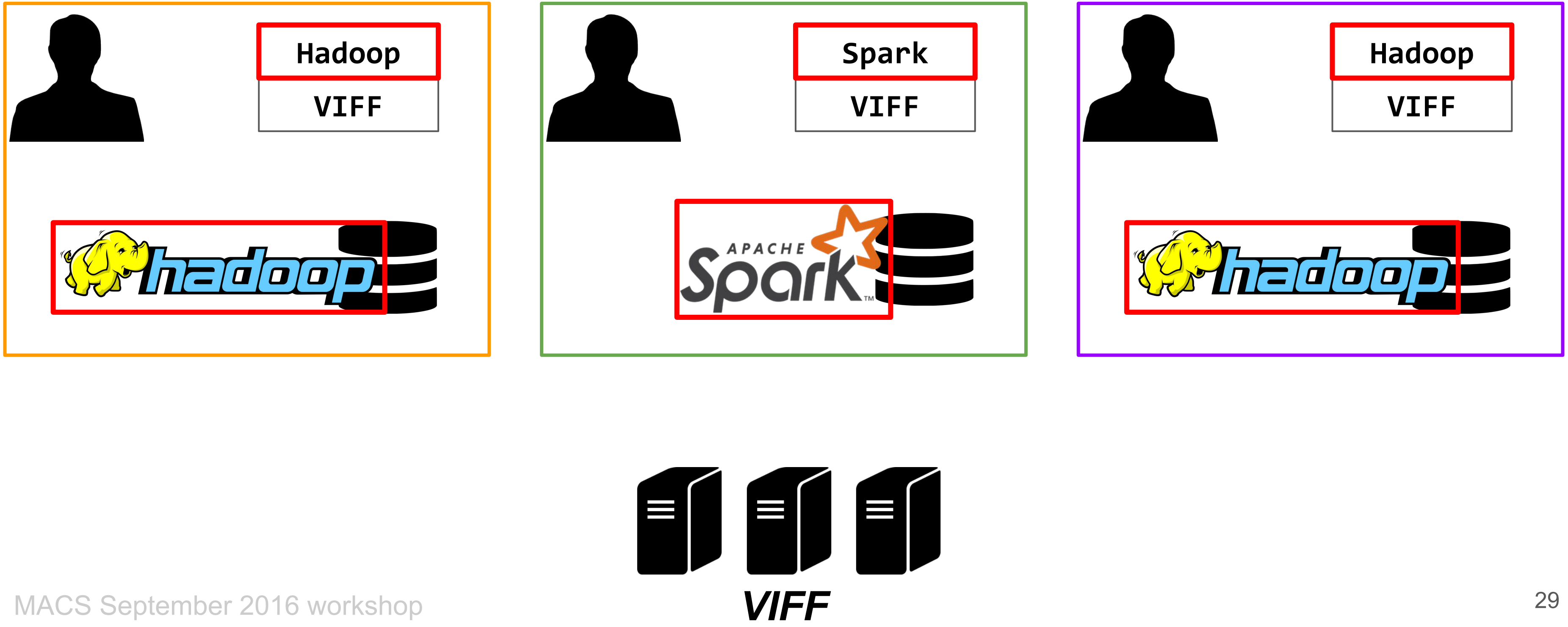
Hadoop

VIFF

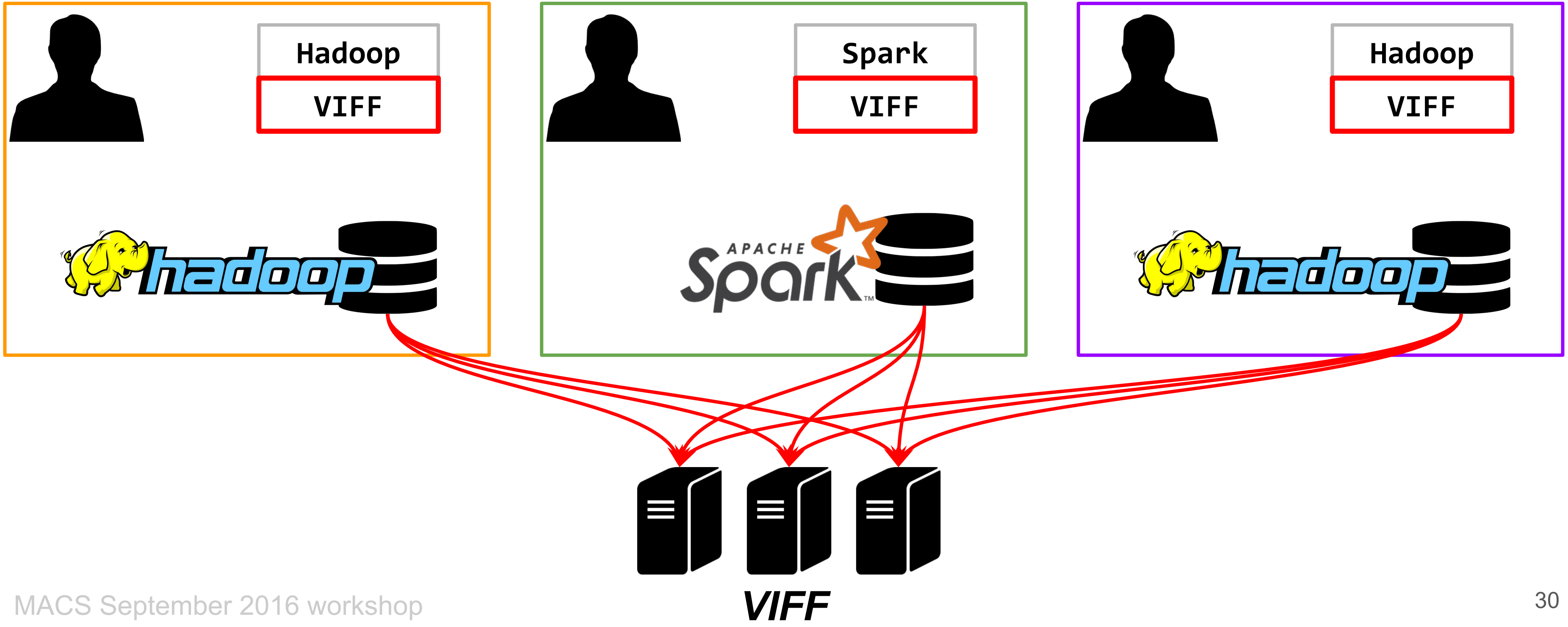


VIFF

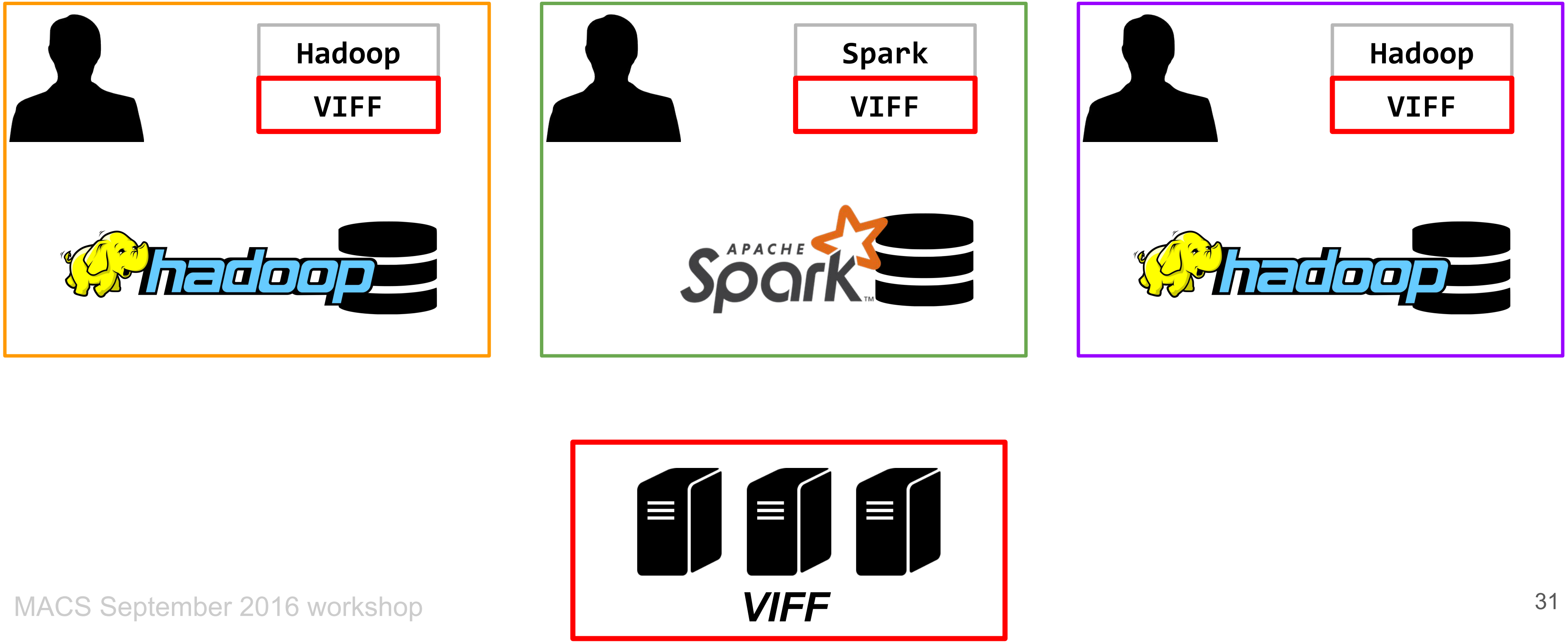
The subtasks are dispatched to the available backends and executed there



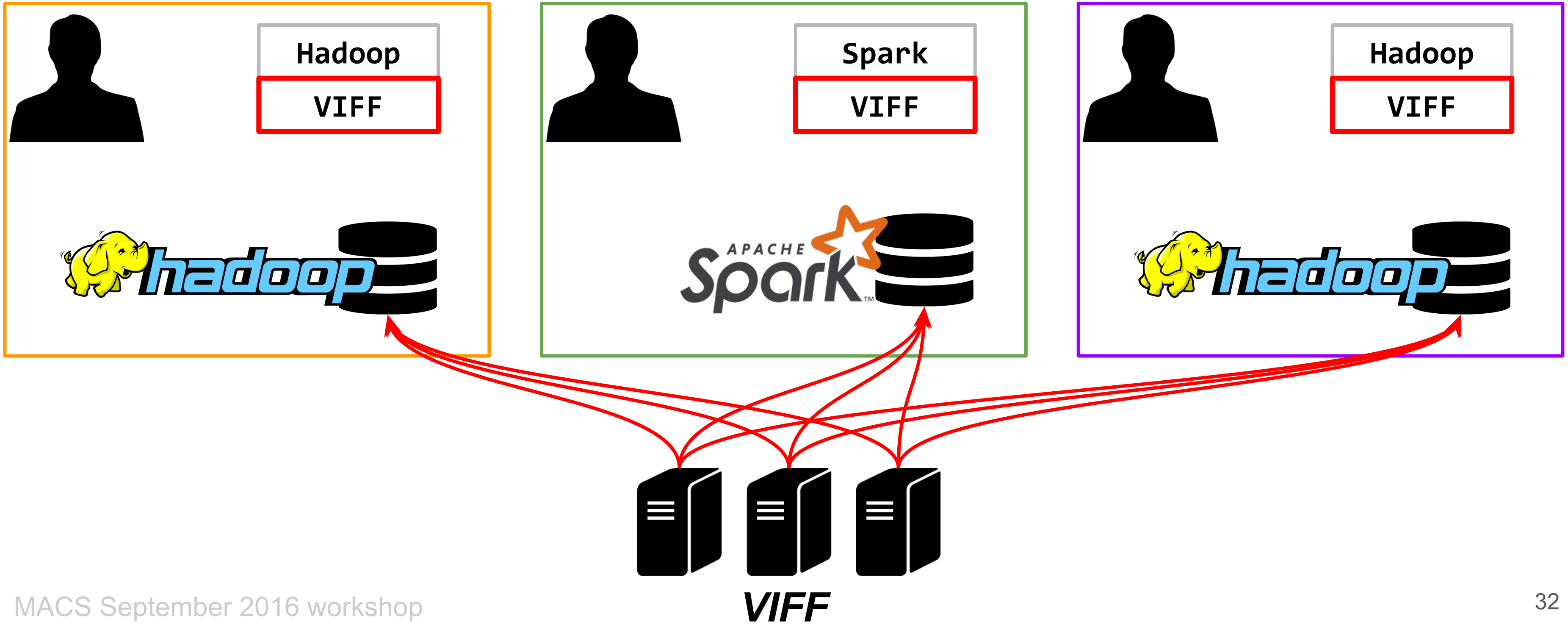
The MPC step involves delivering data to the MPC service



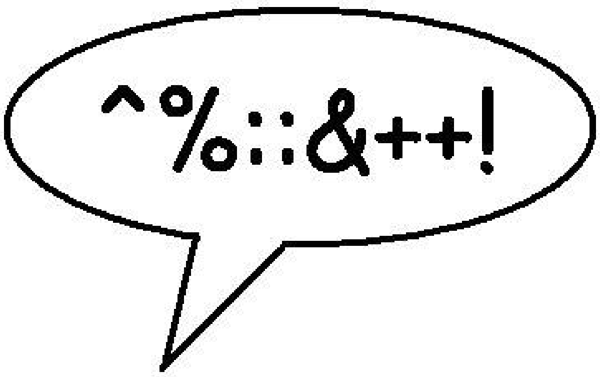
Executing the analytics on the secret data



And finally retrieving the results



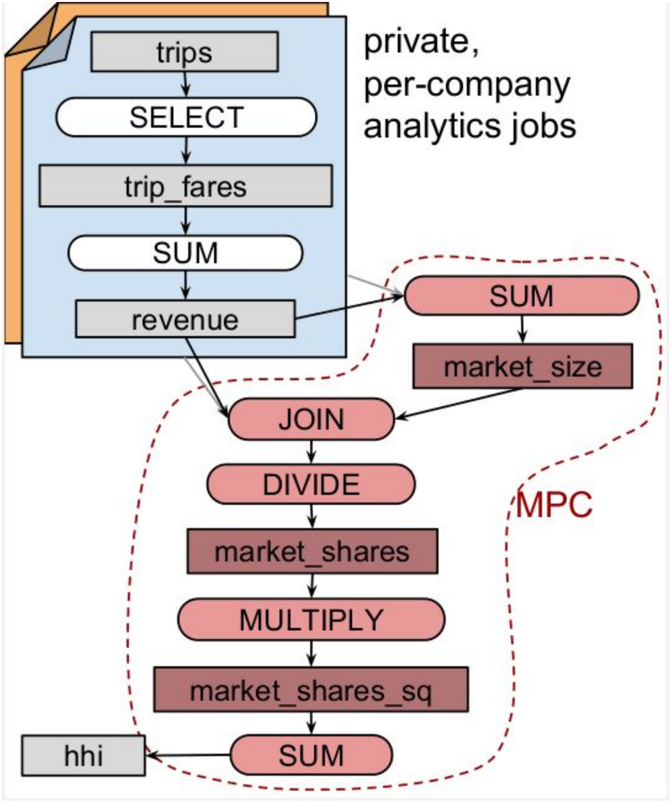
Okay, but did you actually count swear words?



Herfindahl-Hirschman Index

A measure of market concentration.

The sum of squares of a market shares.

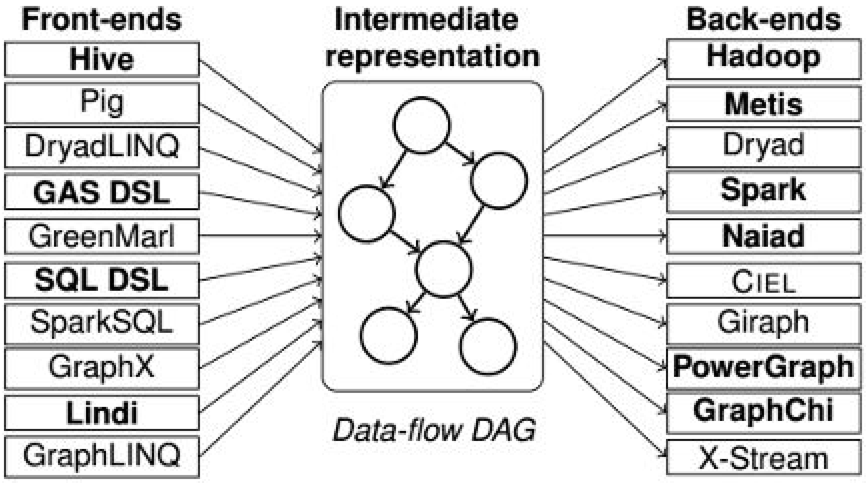


Market concentration of NYC cab trip data

Setup	Data Volume	Runtime
Insecure, trusted Hadoop (8 nodes)	156 GB	16 min 10 s (970s)
Our system with MPC (5 parties, 1+1+1+1+4 nodes)	**{16,16,16,28,80} GB**	**17 min 31 s (1,051s)**
MPC framework only (VIFF, 5 parties, 5 nodes)	156 GB	>2 hours (>7,200s)

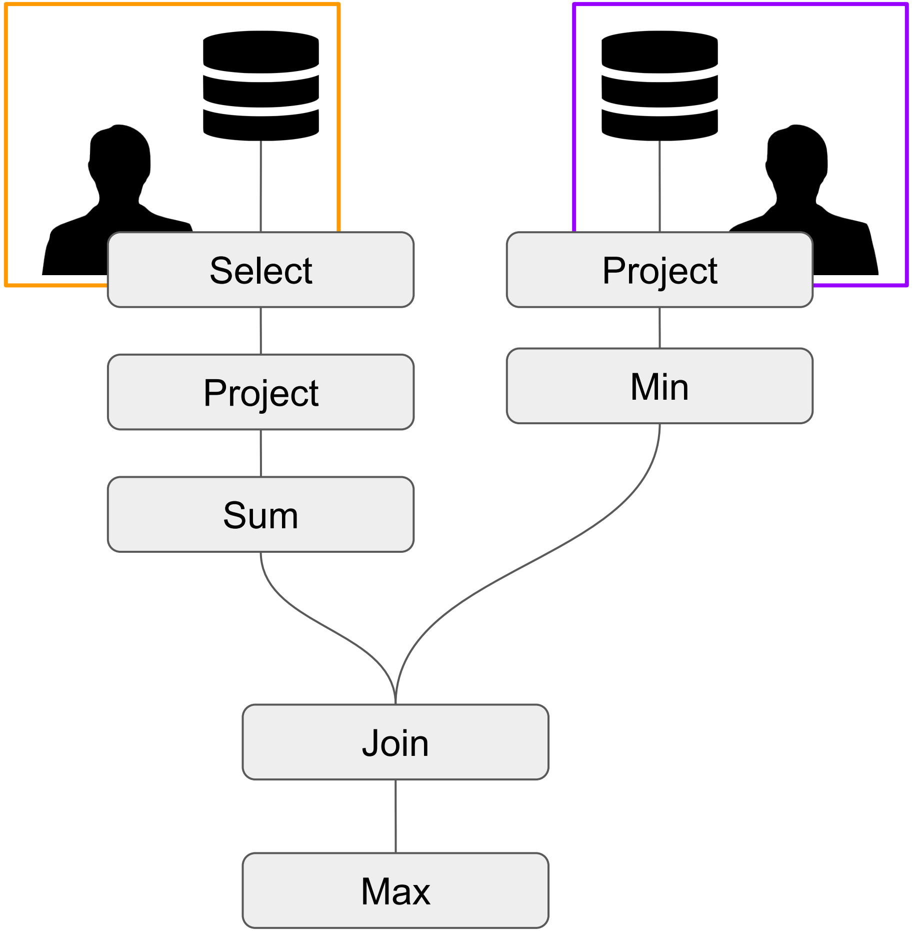
Implementation

We extended ***Musketeer***, a big data workflow manager, to incorporate MPC.



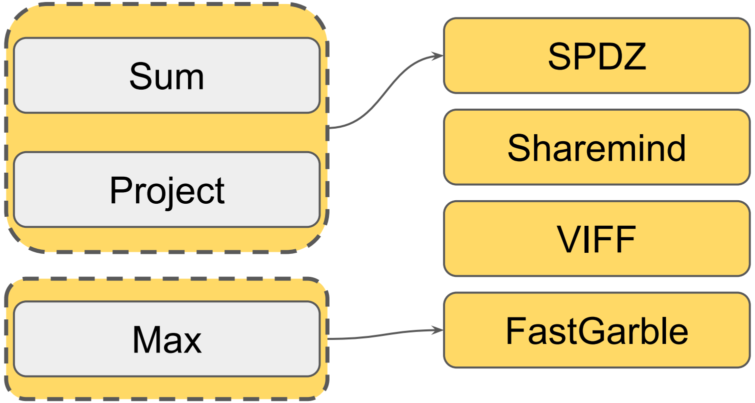
Future directions

- **Ownership provenance**
- More MPC backends!
- Multiple MPC backends in single workflow
- Repeated MPC (iterative/separate cliques)


Select
Project
Project
Min
Sum
Join
Max

Future directions

- Ownership provenance
- **More MPC backends!**
- **Multiple MPC backends in single workflow**
- Repeated MPC (iterative/separate cliques)



Future directions

- Ownership provenance
- More MPC backends!
- Multiple MPC backends in single workflow
- **Repeated MPC (iterative/separate cliques)**

Summary

SQL-like programming language to specify analytics using relational operators.
⇒ ***No MPC experience required!***

Compiler detects MPC boundaries, converts programs to parallel data processing and MPC jobs, and generates code for individual jobs.
⇒ ***No manual implementation required.***

Dispatcher executes the generated jobs automatically on the available backends, choosing the best strategy.
⇒ ***No new infrastructure or "glue scripts" required.***