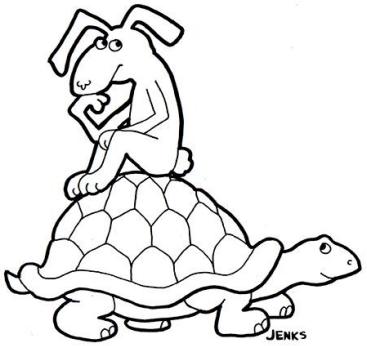
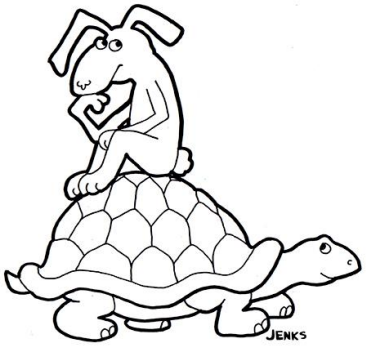
Programming Support for an Integrated Multi-Party Computation and MapReduce Infrastructure

Nikolaj Volgushev, Andrei Lapets, Azer Bestavros

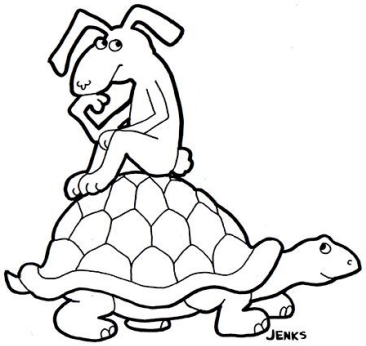
Programming Support for an Integrated Multi-Party Computation and MapReduce Infrastructure



Programming Support for an Integrated **Multi-Party Computation** and **MapReduce** Infrastructure



Programming Support for an Integrated **Multi-Party Computation** and MapReduce Infrastructure



What's multi-party computation (MPC)?

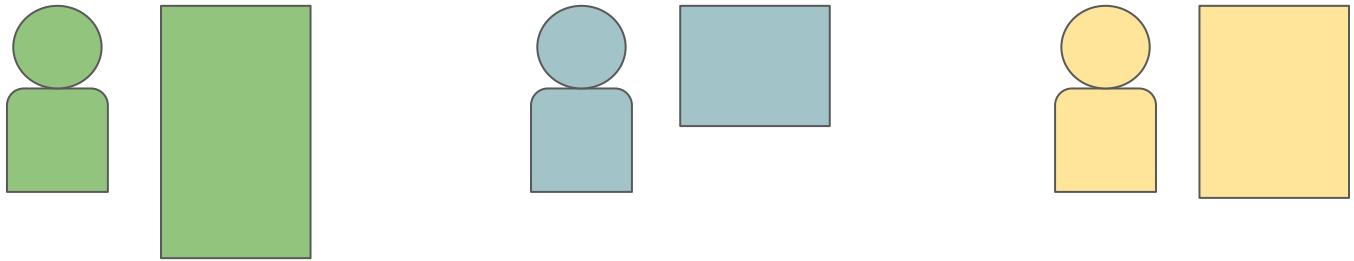
Given multiple parties $p_1, p_2, \ldots, p_n$ with private inputs $x_1, x_2, \ldots, x_n$

Need to compute $f(x_1, x_2, \ldots, x_n)$

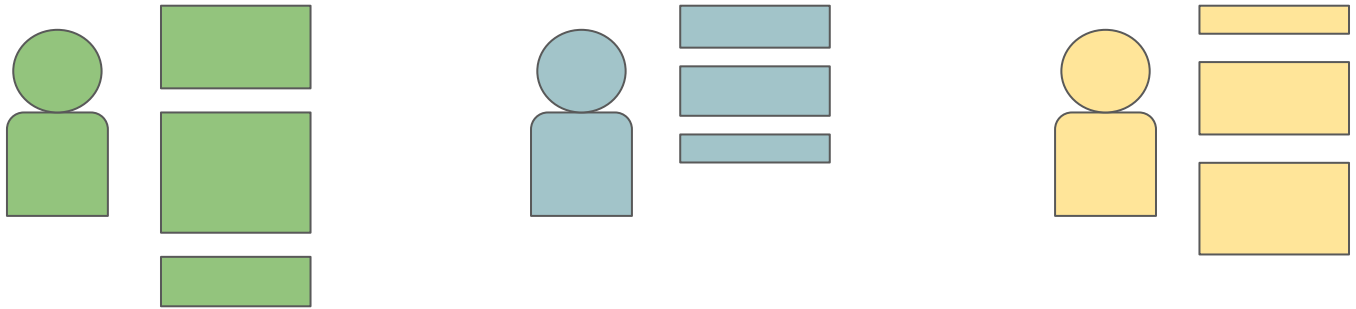
Without revealing more than the outputs of f

Sounds a bit like a magic trick.

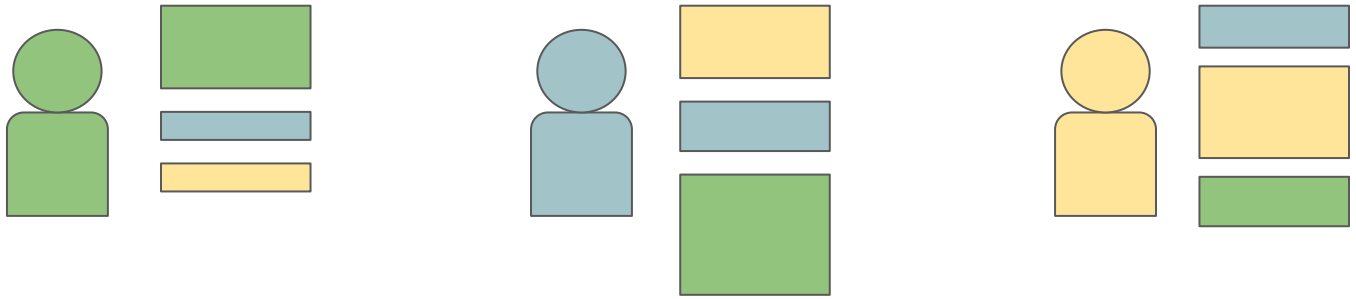
Quick example: the sum of secrets



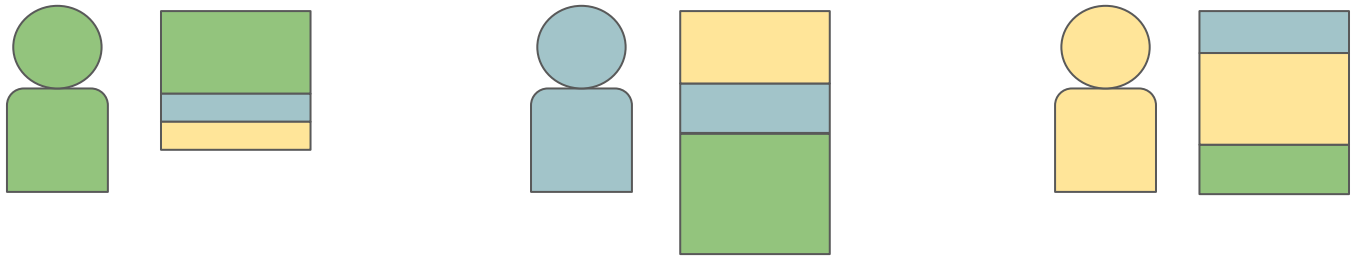
Players split their secrets into “shares”



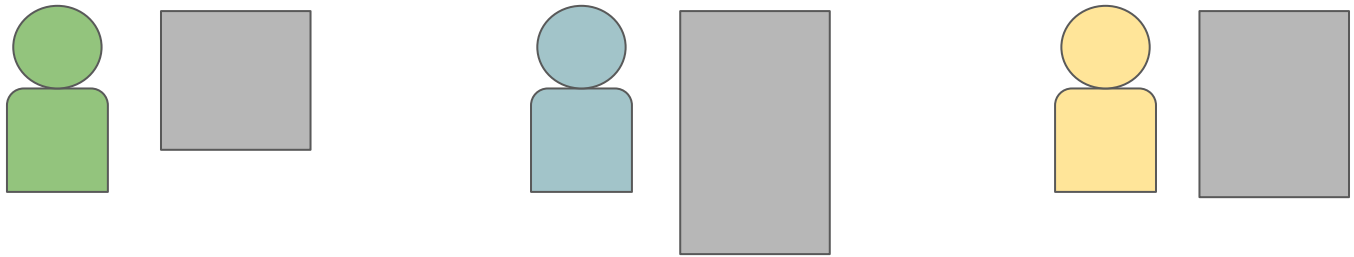
All players exchange shares



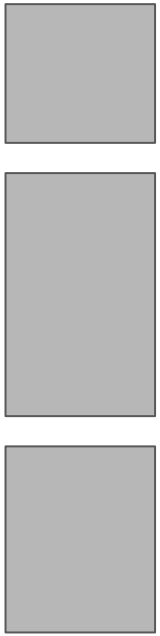
Each player computes the sum of their shares



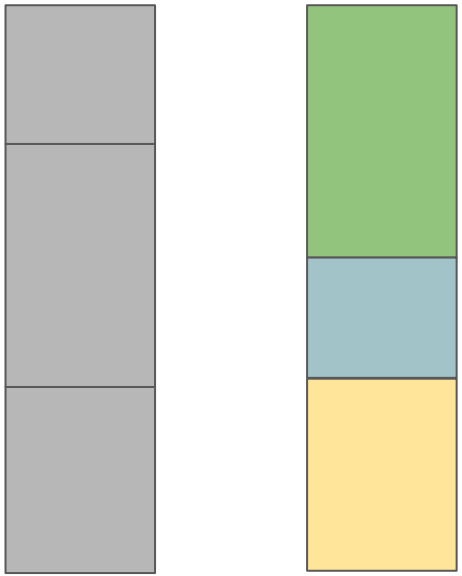
Each player now holds a share of the sum of secrets!



Exchange and recombine result shares



Lo and behold



MPC is great!

Let’s us run computations, while preserving privacy of the inputs.

A mathematical turtle shell for our data!

MPC is great (in theory)!

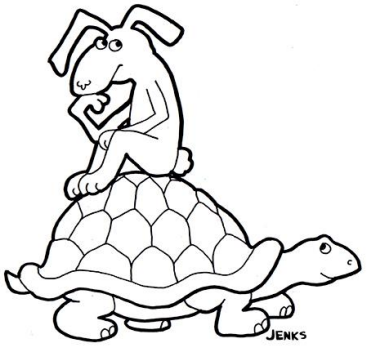
Let's us run computations, while preserving privacy of the inputs.

A mathematical turtle shell for our data!

Practical MPC frameworks exist but they are **slow** (not unlike turtles)

The learning curve is steep (trust me!)

Programming Support for an Integrated Multi-Party Computation and **MapReduce** Infrastructure



MapReduce: fast, like a hare!

Programming paradigm to specify data analytics tasks.

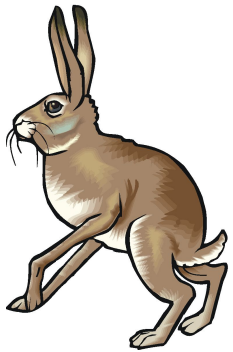
Backend infrastructure as a highly-distributed execution environment for those tasks.

Performance. Largest Apache Spark cluster is 8000 nodes**.**

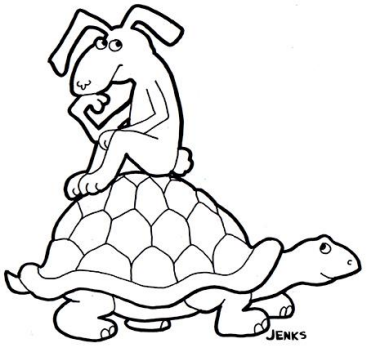
200 node Spark cluster sorted 100TB of data in 23 minutes**.**

Separation of concerns. Data analysts specifies analytics,

doesn’t worry about distributed nature of platform.



Programming Support for an **Integrated** Multi-Party Computation and MapReduce Infrastructure



To do big data analytics we need...

Data. But the interesting data is often private.

Performance. But MPC is slow!

An implementation of the analytics. But data analysts don’t know MPC...

A concrete example

Goal: Want to establish the difference between male and female salaries across the big companies around Boston.

Good: Companies like this idea. Participating makes them look good.

Bad: Until they’re asked to reveal their internal pay inequities to a “trusted” party.

If only they had our platform:

Each company can use **MapReduce** to find the salary differences in their own data.

→ Lots of computation

The companies can use **MPC** to find the collective difference without revealing their data.

→ Just one addition

The main components of our platform

Programming language to specify MapReduce and MPC operations.

Compiler to convert programs to tasks that are executable in existing MapReduce and MPC frameworks.

Backend platform running those MapReduce and MPC frameworks to act as an execution environment for a compiled program.

Let’s explore our platform top-down

Programming language to specify MapReduce and MPC operations.

Compiler to convert programs to tasks that are executable in existing MapReduce and MPC frameworks.

Backend platform running those MapReduce and MPC frameworks to act as an execution environment for a compiled Scatter program.

Pay equity: declaring our **key**-*value* store

1: type gender = str
2: type salary = int
3: data := store(**gender**, *salary*)

Our MapReduce operations

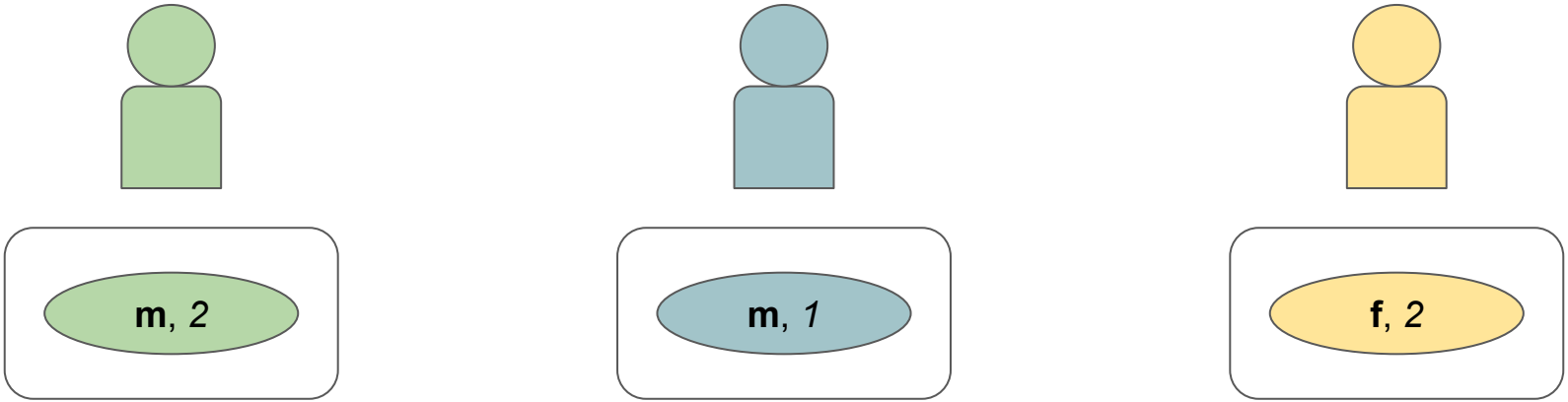
```
1: type gender = str
2: type salary = int
3: data := store(gender, salary)

4: m := reduce(+, filter("m", data))
5: f := reduce(+, filter("f", data))
6: d := m - f
```

What about MPC?

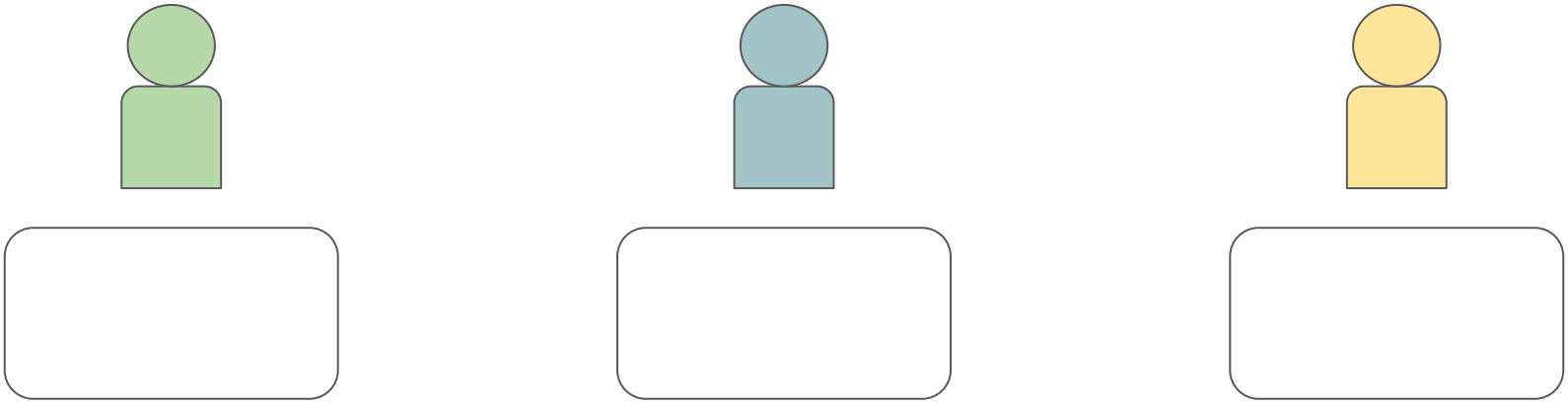
Two main constructs:

Scatter, and **Gather**.

Scatter: make secret and share

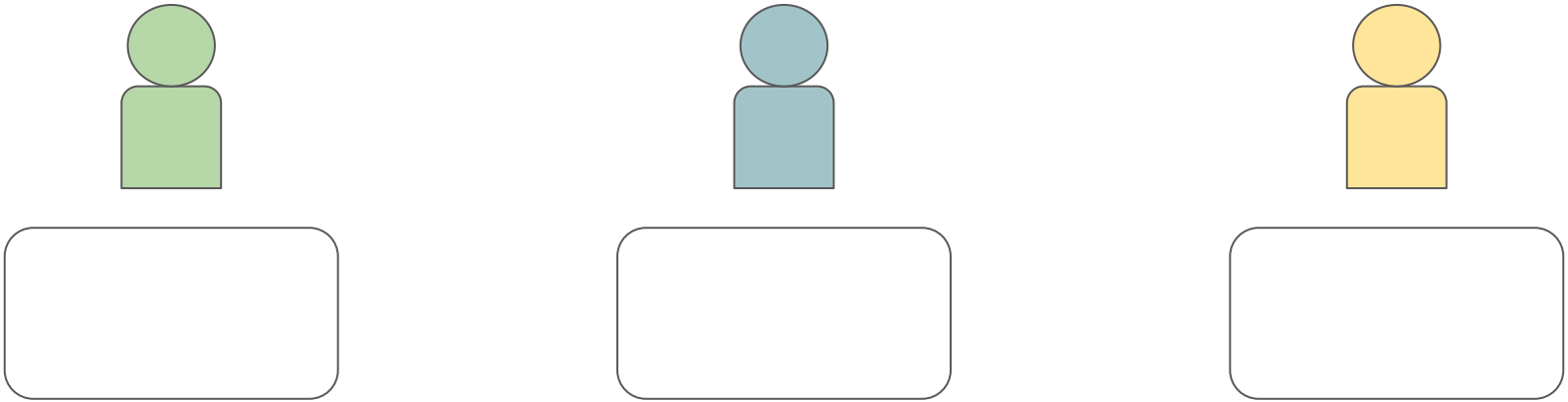


Scatter: make secret and share

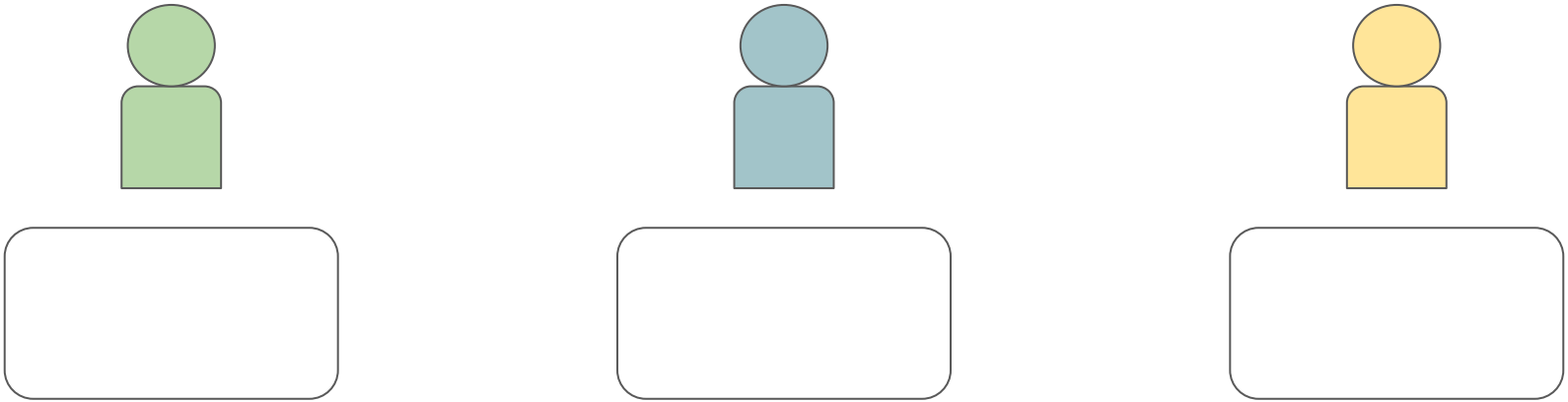


Secret but shared key value store

Scatter: make secret and share



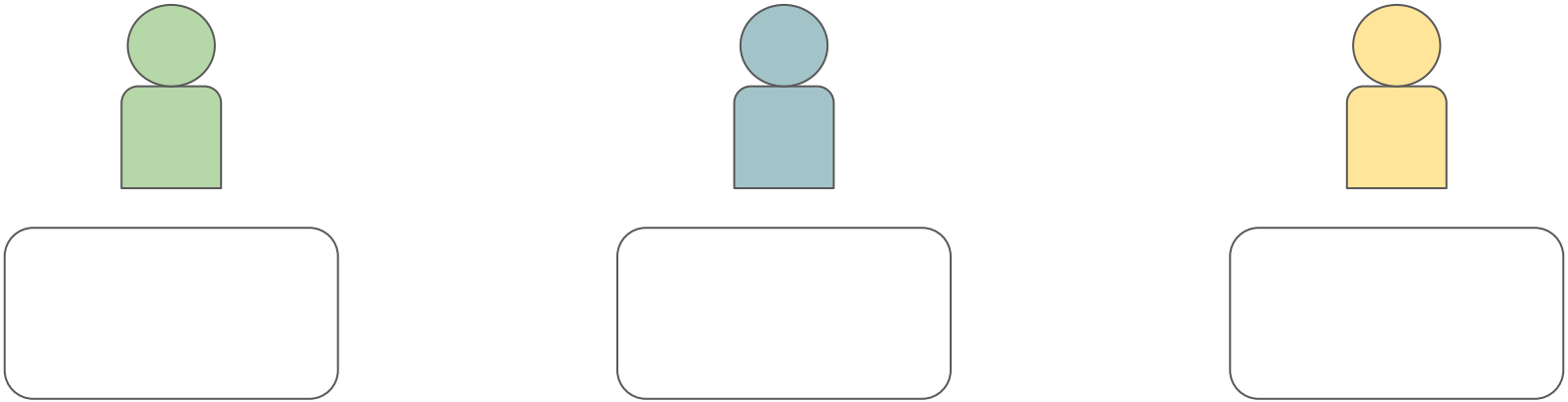
Let’s say we want to perform a *reduce*(+)



Secret but shared key value store

m, *[2]* **m**, *[1]* **f**, *[2]*

Let’s say we want to perform a *reduce*(+)

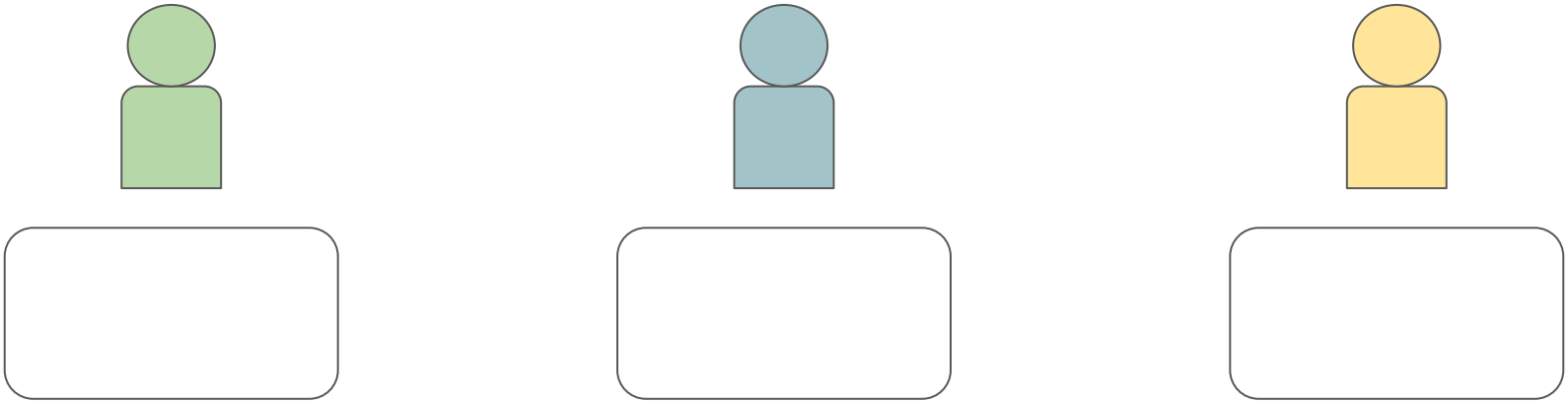


Secret but shared key value store

m, *[2+1]*

f, *[2]*

Gather: collect and reveal

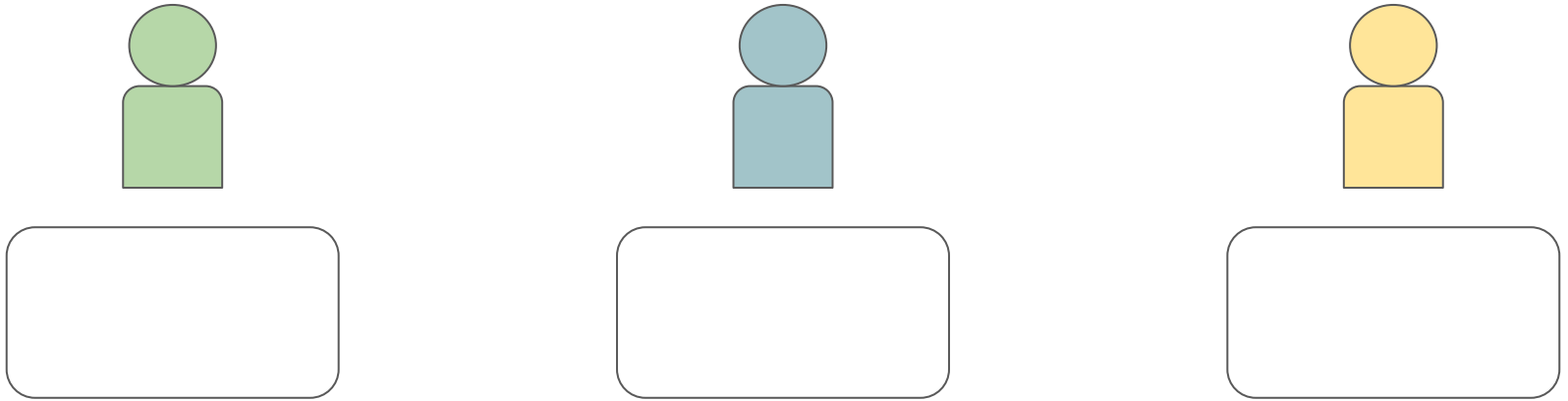


Secret but shared key value store

m, *[2+1]*

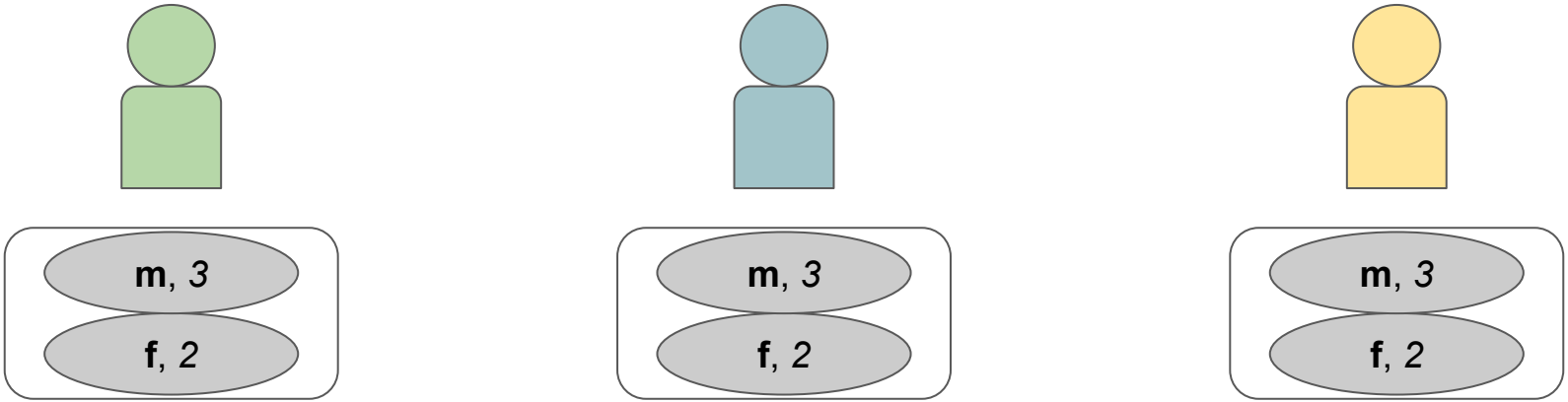
f, *[2]*

Gather: collect and reveal



Secret but shared key value store

Gather: collect and reveal



Secret but shared key value store

Complete pay equity program

```
1: type gender = str
2: type salary = int
3: data := store(gender, salary)

4: m := reduce(+, filter("m", data))
5: f := reduce(+, filter("f", data))
6: d := m - f

8: s := gather(reduce(+, scatter(d)))
```

Each company will execute this locally

1: type gender = str
2: type salary = int
3: data := store(gender, salary)

4: m := reduce(+, filter("m", data))
5: f := reduce(+, filter("f", data))
6: d := m - f

8: s := gather(reduce(+, scatter(d)))

The companies will need to perform an MPC

```
1: type gender = str
2: type salary = int
3: data := store(gender, salary)

4: m := reduce(+, filter("m", data))
5: f := reduce(+, filter("f", data))
6: d := m - f

8: s := gather(reduce(+, scatter(d)))
```

What do we do with a program?

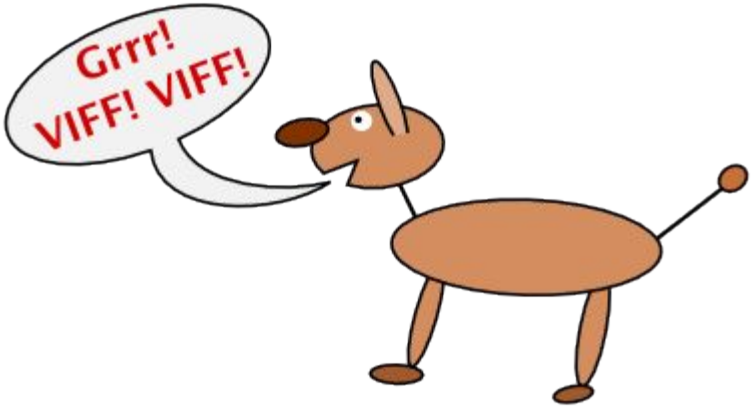
Programming language to specify MapReduce and MPC operations.

Compiler to convert Scatter programs to tasks that are executable in existing MapReduce and MPC frameworks.

Backend platform running those MapReduce and MPC frameworks to act as an execution environment for a compiled program.

Our current target frameworks

“a fast and general engine for large-scale data processing”



MPC framework that allows for Shamir secret sharing, arithmetic, and comparison over secret shares

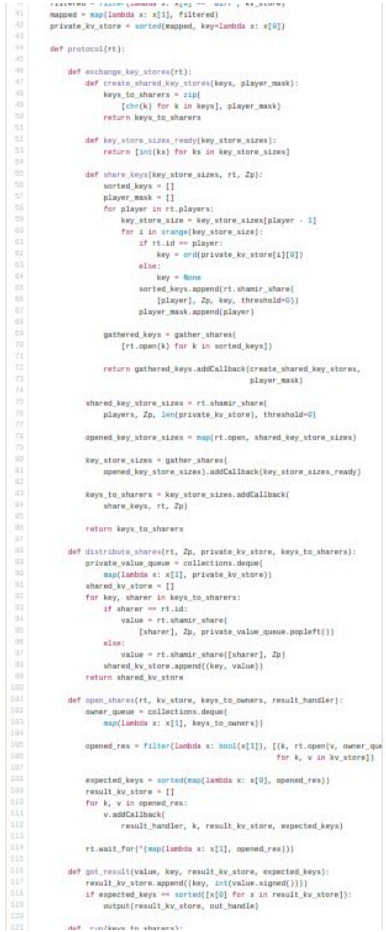
4: m := **reduce**(+, **filter**("m", data))

5: f := **reduce**(+, **filter**("f", data))

6: d := m - f

```
m = data.filter(lambda x: x[0] == 'm')\
        .reduceByKey(lambda x, y: x + y)\
        .collect()
f = data.filter(lambda x: x[0] == 'f')\
        .reduceByKey(lambda x, y: x + y)\
        .collect()
d = ('d', m[0][1] - f[0][1])
```

8: s := **gather**(**reduce**(+, **scatter**(d)))



What to do with executable code?

Programming language to (unified) specify MapReduce and MPC operations.

Compiler to convert programs to tasks that are executable in existing MapReduce and MPC frameworks.

Backend platform running those MapReduce and MPC frameworks to act as an execution environment for a program.

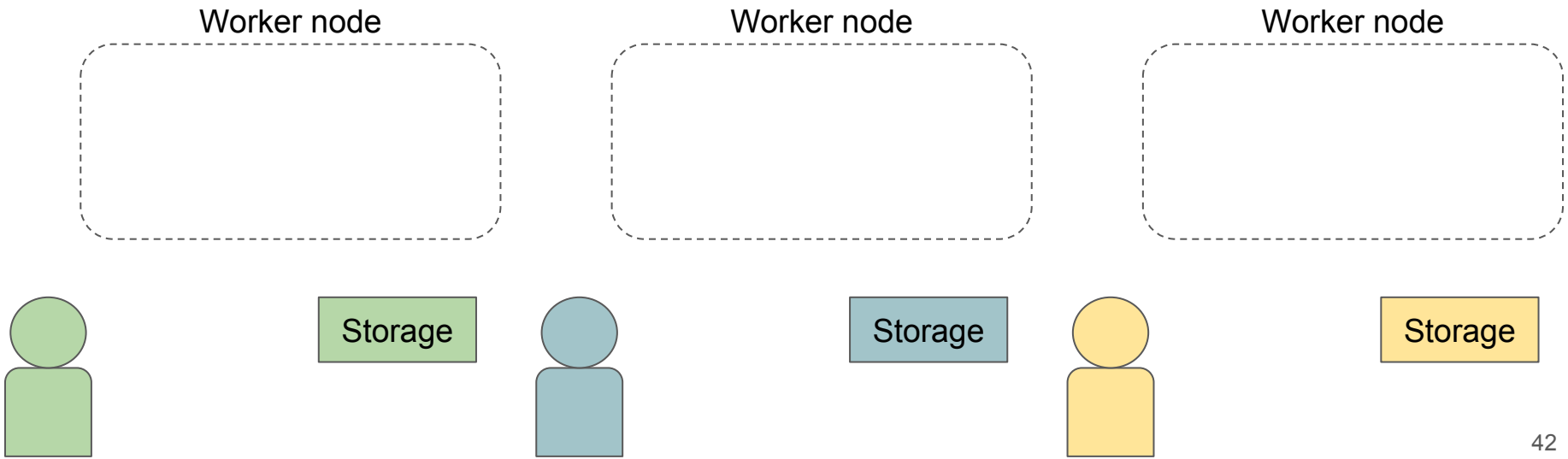
Let’s build our backend.

Give each client the computational resources to:

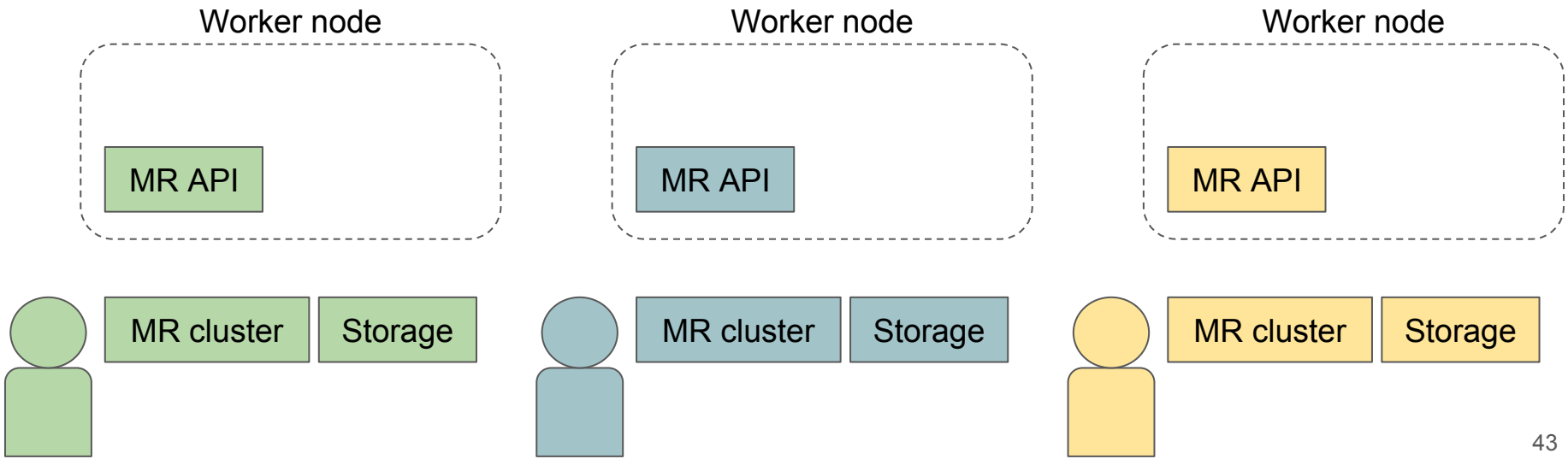
- run local MapReduce tasks on their data
- participate in MPC rounds to process data across companies
- coordinate those two actions

Let’s build a **worker node**.

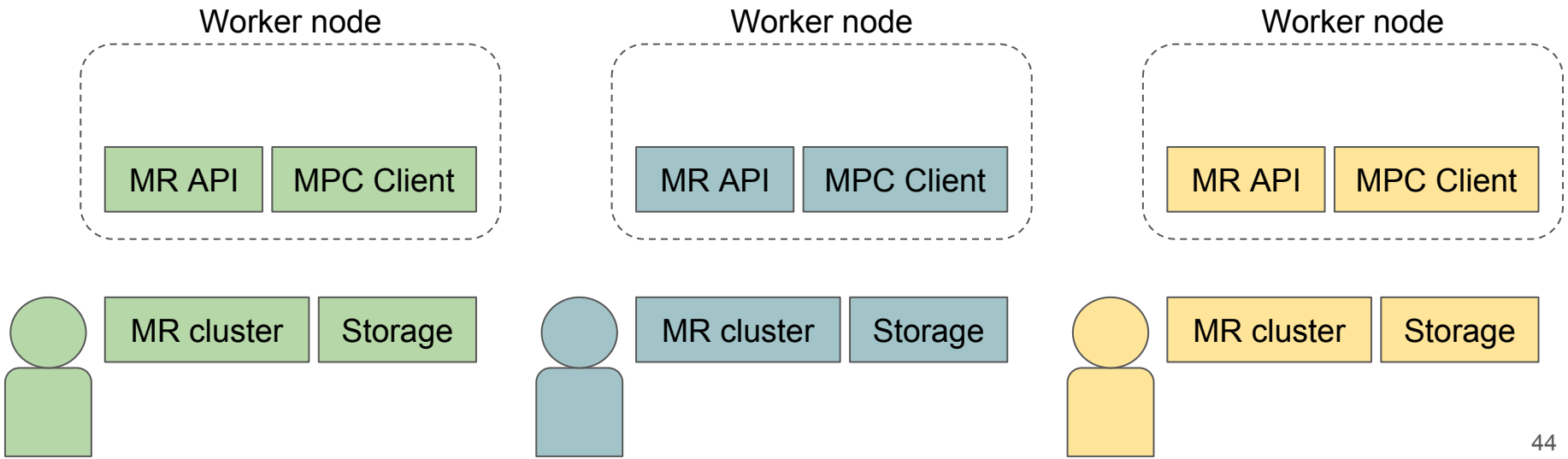
What does each company start with?



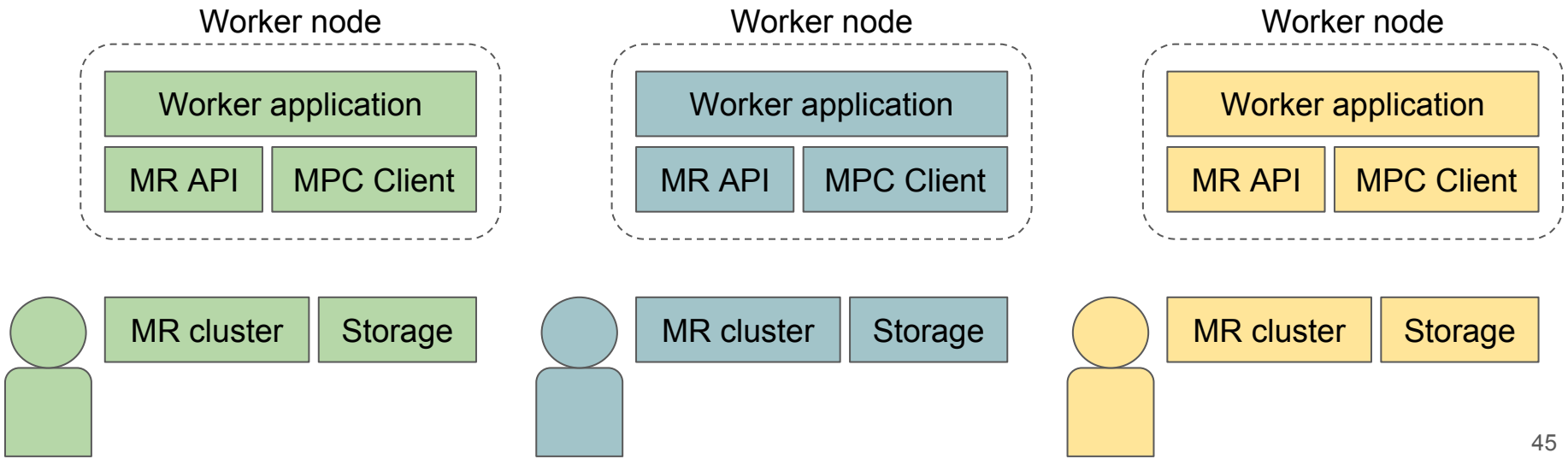
What do companies need to run MapReduce code?



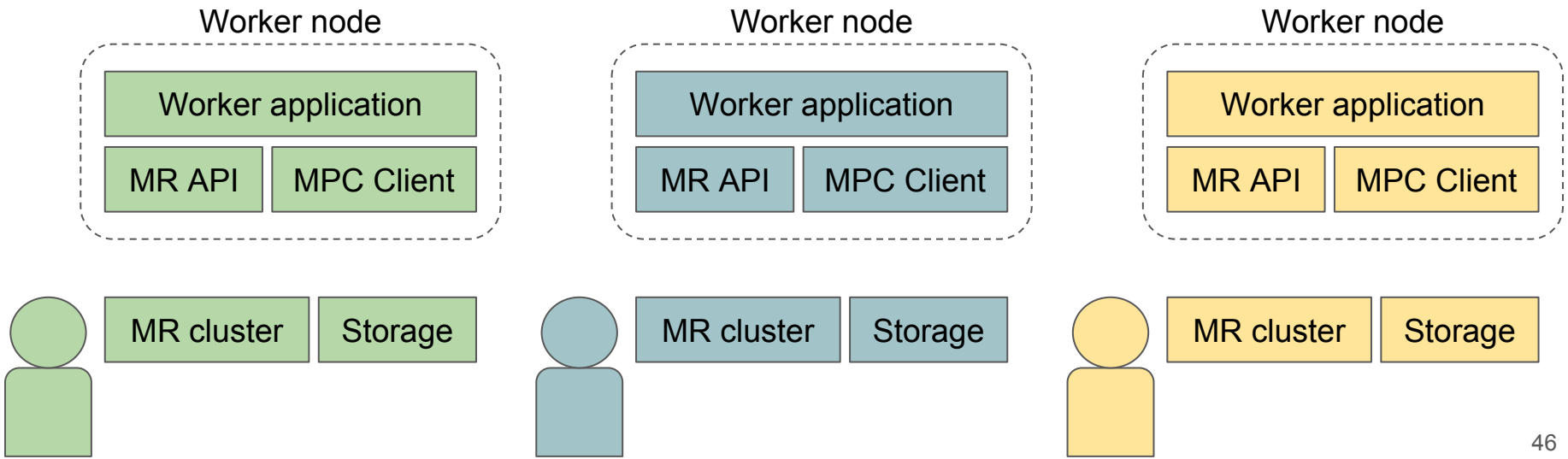
What do companies need to run MPC code?



What about coordinating program execution?



Bundle up the software, we have our **worker nodes**!



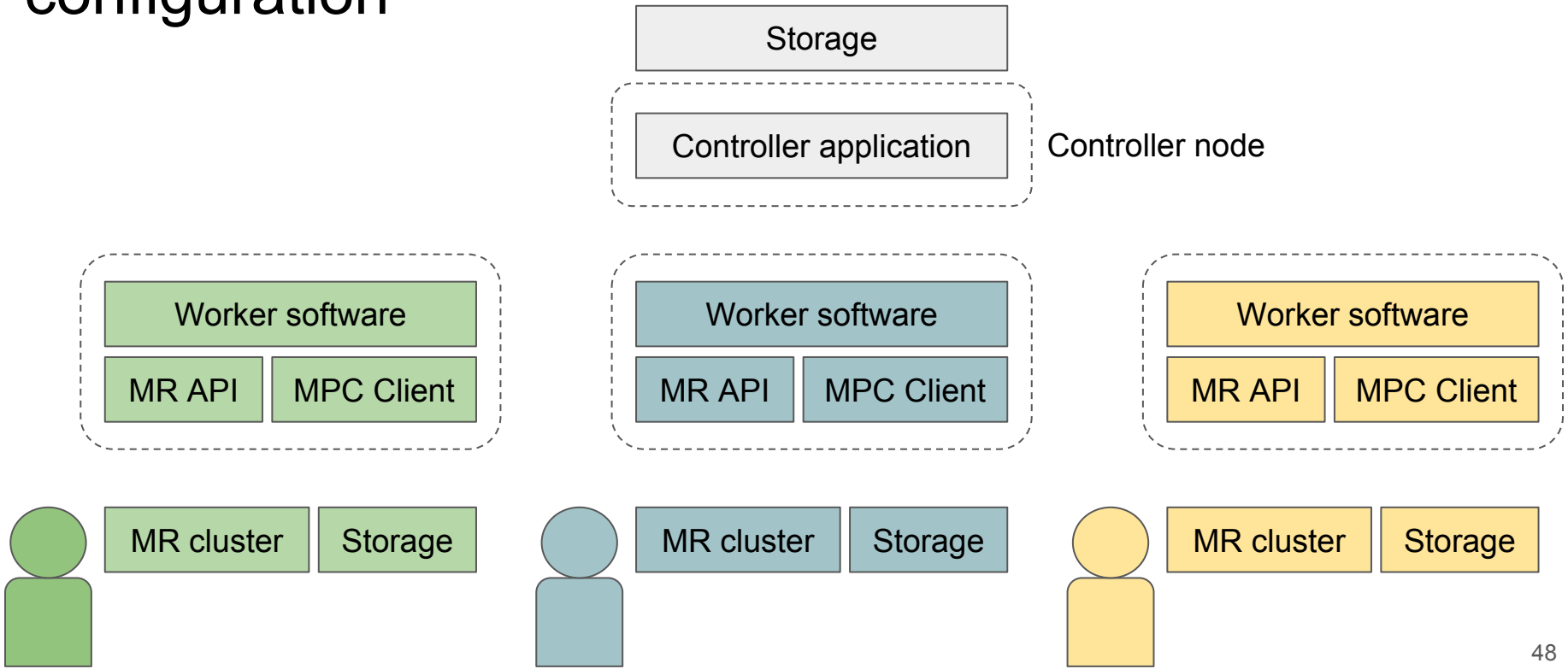
Almost done...

We have a distributed system.

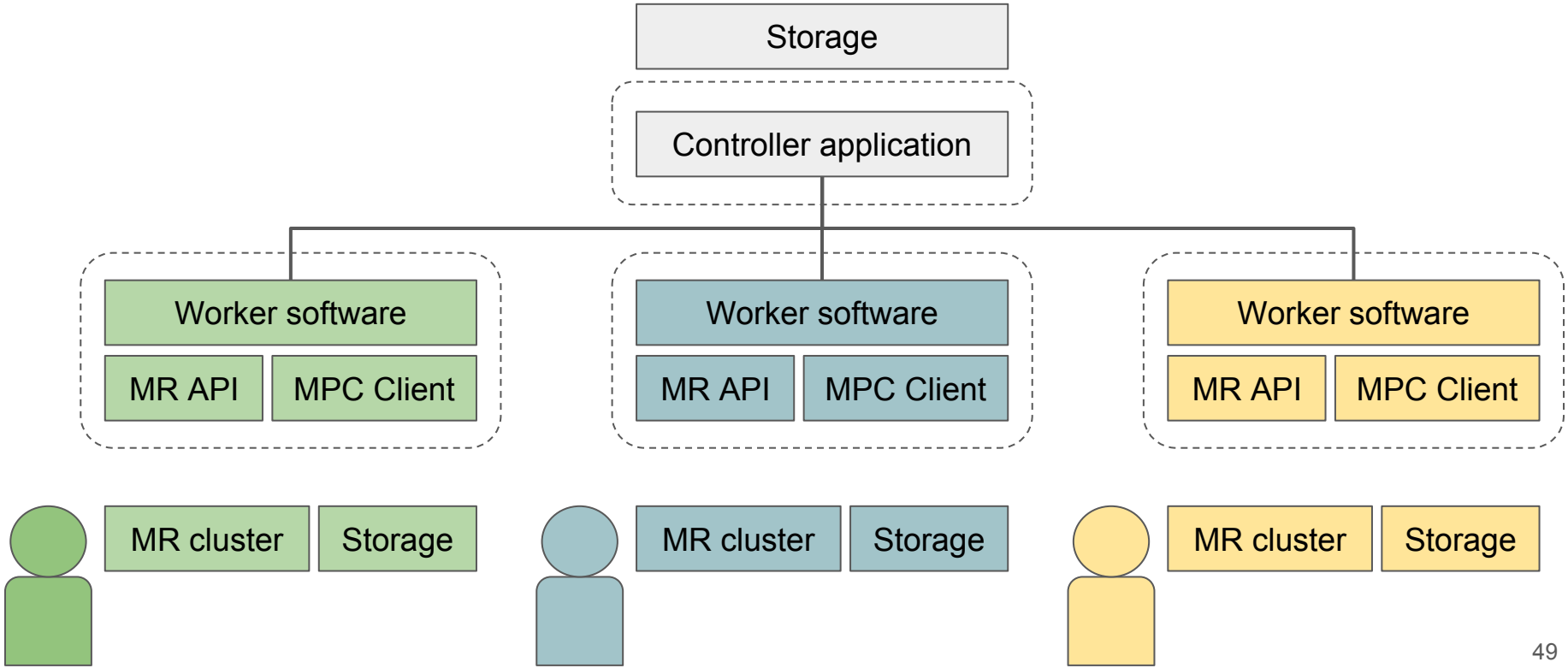
We need to coordinate task execution not only within worker nodes but also **across** worker nodes.

Let’s build a **controller node**.

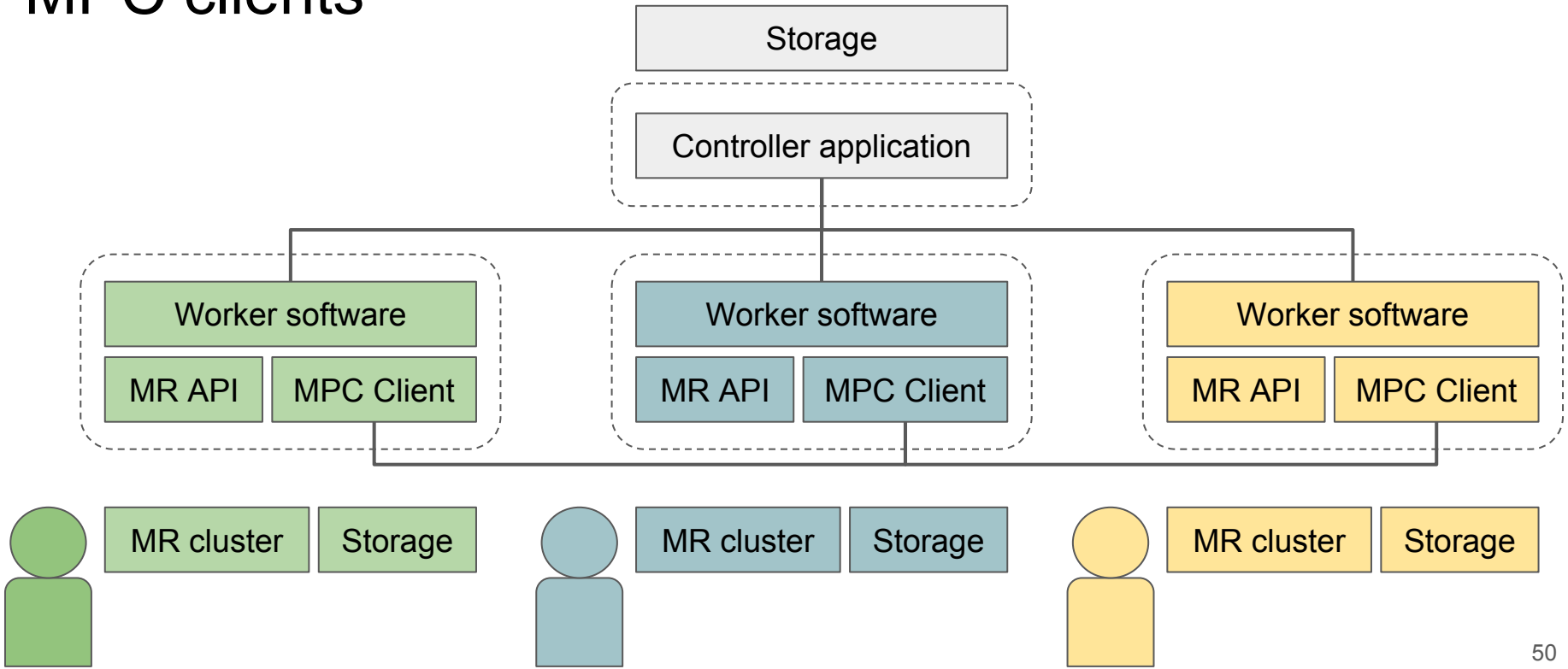
Controller: orchestrate task execution + worker configuration



Workers connect to Controller over HTTPS



Workers use controller to configure and connect MPC clients



And there we have it!

Programming language to specify MapReduce and MPC operations.

Compiler to convert programs to tasks that are executable in existing MapReduce and MPC frameworks.

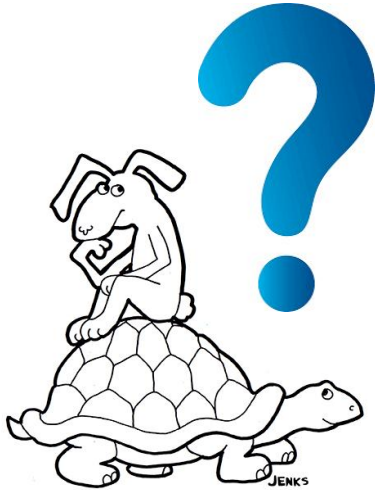
Backend platform running those MapReduce and MPC frameworks to act as an execution environment for a compiled program.

Future work

Extending the support to more MPC and MapReduce backends

Separation of concerns

So many plans, so little time!


JENKS