Parallel programming - Polars and Dask

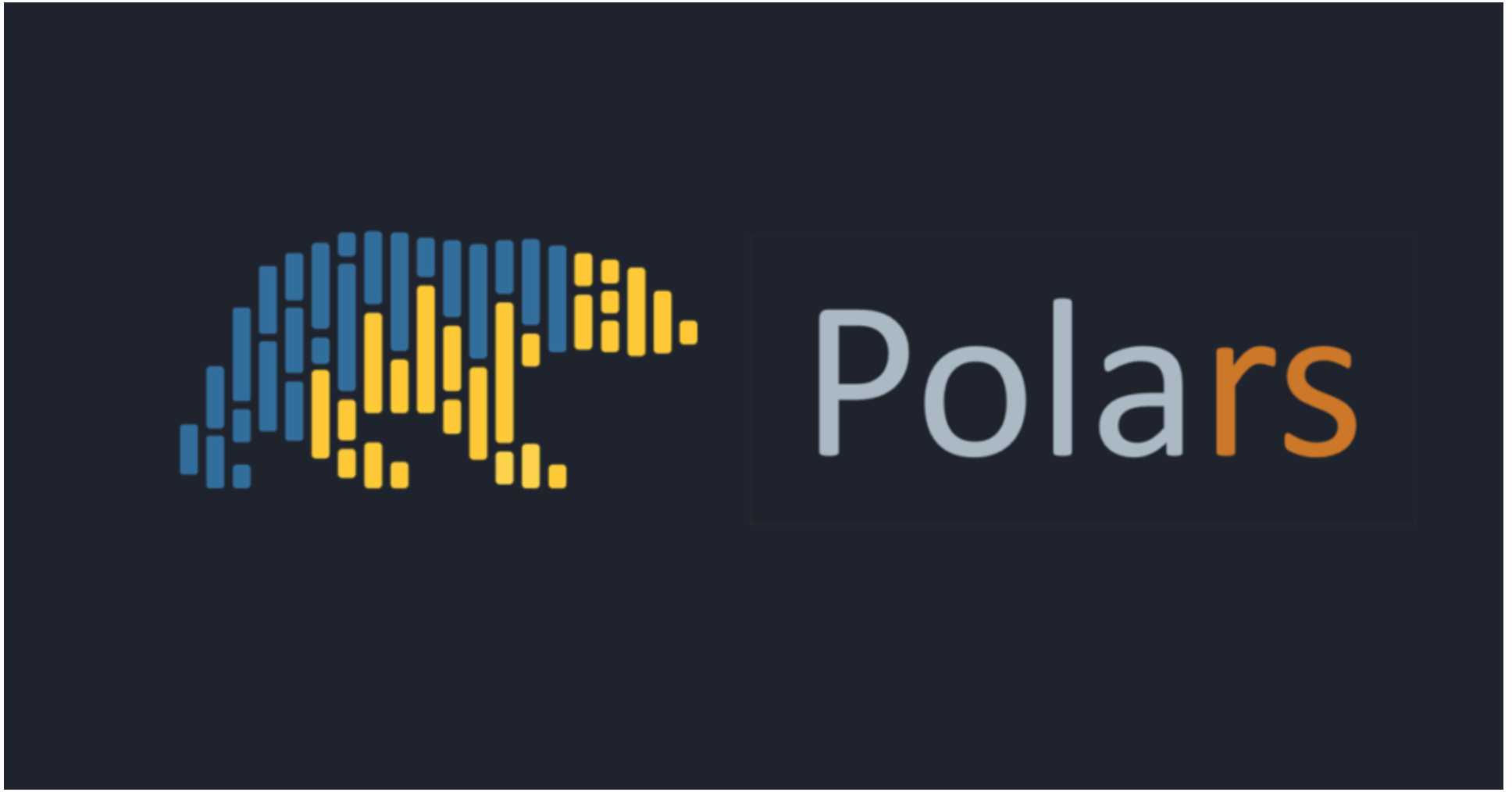
(For the curious reader)

Parallel programming

Breaking down larger problems into smaller, independen, that can be executed simultaneously.

Fundamentals of Parallel Computer Architecture

- **Multi-core computing**: A multi-core processor has multiple processing cores on a single chip, which can execute instructions in parallel.
- **Symmetric multiprocessing (SMP)**: In SMP, two or more similar processors controlled by a single OS have equal access to a shared main memory and all common resources.
- **Distributed computing**: In this architecture, system components on different networked computers coordinate their actions through communication, often using HTTP or message queues.
- **Massively parallel computing**: This involves using a large number of computers or processors to execute computations in parallel.


Polars

Short Introduction to Polars

https://pola.rs/

Polars

Polars is a library for parallel data processing. It is useful for working with large datasets. Polars is imported using the `import` keyword. Polars is usually imported using the alias `pl`.

```
import polars as pl
```

How is it coded? Polars is written in Rust, a low-level programming language. Rust is similar to C++, but it is safer and more efficient. Rust is used to write high-performance applications, including web browsers and operating systems.

Polars - Dataframes

Dataframes are the main data structure in Polars. They are used to store tabular data. They can be created using the `pl.DataFrame` function. Dataframes can be created from lists, tuples, dictionaries, and other dataframes. It is common to name dataframes `df`.

```
df = pl.DataFrame({'a': [1], 'b': [4]})
```

```
shape: (3, 2)
┌─────┬─────┐
│ a   ┆ b   │
│ --- ┆ --- │
│ i64 ┆ i64 │
╞═════╪═════╡
│ 1   ┆ 4   │
└─────┴─────┘
```

Polars - Lazy evaluation

Polars uses lazy evaluation to improve performance. This means that operations are not executed until they are needed. This is useful for working with large datasets, because it allows the program to only load the data that is needed. Lazy evaluation is done using the `lazy` method. The `lazy` method has one argument: a function. The function is applied to the dataframe, and the result is returned.

```
df = pl.DataFrame({'a': [1, 2, 3], 'b': [4, 5, 6]})
df.lazy()
```

Polars - General Instructions

Some instructions are similar to Pandas, however some other operations have a different syntax to connect to the Rust API.

Creating dataframes

```
df = pl.read_csv('file.csv')
df = pl.read_parquet('file.parquet')
```

Polars - Expressions

Expressions can be performed in sequence, this improves readability. You can use the `\` character to split code, or use parenthesis (**recommended**).

- Use `filter` to mask observations.
- Use `pl.col` to select a column.
- Use methos inside `pl` to perform operations on columns. E.g. `pl.col('a').sum()`.

Example of Polars code

```
df = (df
    .filter(pl.col('a') > 0)
    .select(pl.col('a'), pl.col('b'))
    .groupby(pl.col('a'))
    .agg([pl.col('b').sum().alias('sum_b')])
)
```

Subset Observations - rows

Filter: Extract rows that meet logical criteria.

```
df.flter(pl.col("random") > 0.5)
df.flter(
(pl.col("groups")  "B")
& (pl.col("random") > 0.5)
)
```

Sample

```
# Randomly select fraction of rows.
df.sample(frac=0.5)
# Randomly select n rows.
df.sample(n=2)
```

Subset Observations - rows (2)

```
#Select first and last rows
# Select frst n rows
df.head(n=2)
# Select last n rows.
df.tail(n=2)
```

Make New Columns

```
df.with_columns(
    [
    (pl.col("a") * 2).alias("a_doubled"),
    (pl.col("b") + pl.col("a")).alias("a_plus_b"),
    (pl.col("c") / 2).alias("c_halved"),
    ]
)
```

Dask

https://www.dask.org/

What is Dask?

Dask is a flexible parallel computing library for analytic computing. It is designed to scale from single machines to clusters of machines. Dask is useful for working with large datasets. Dask is imported using the `import` keyword and the `dd` alias for the `dask.dataframe` module.

```
import dask.dataframe as dd
```

Example

```
import dask.dataframe as dd

df = dd.read_parquet("s3://data/uber/")

# How much did NYC pay Uber?
df.base_passenger_fare.sum().compute()

# And how much did drivers make?
df.driver_pay.sum().compute()
```

The `compute()` method is used to execute the code after a lazy evaluation.

Dask - Loops

```
from dask.distributed import Client

client = Client()

# Define your own code
def f(x):
    return x + 1

# Run your code in parallel
futures = client.map(f, range(100))
results = client.gather(futures)
```