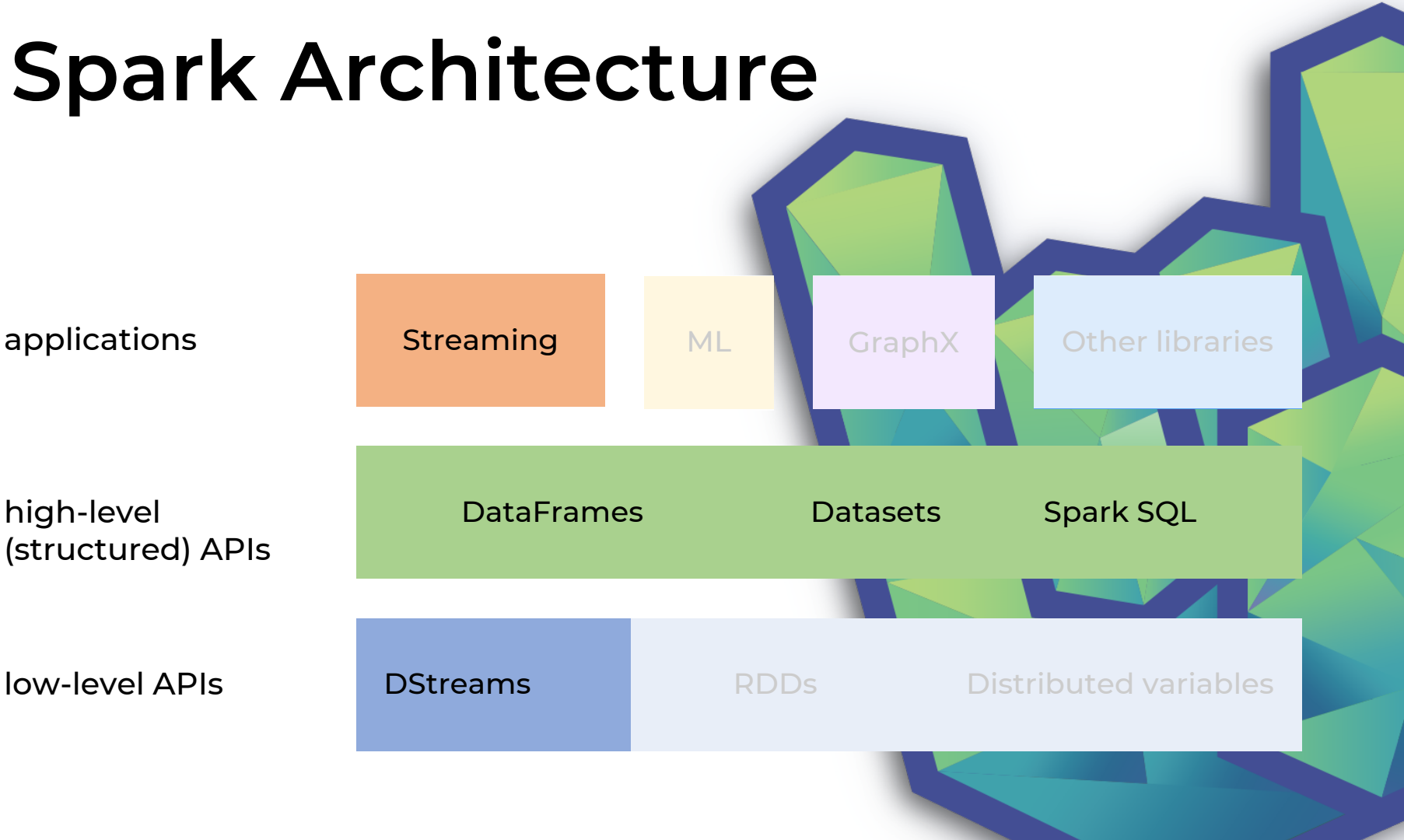
First Principles

Spark Architecture



Why

Once we compute something valuable, we want updates

We need continuous big data processing

=> Spark Streaming

Stream Processing

Include new data to compute a result

No definitive end of incoming data

Batch processing = operate on a fixed (if big) dataset, compute result once

In practice, stream & batch interoperate

- incoming data joined with a fixed dataset
- output of a streaming job periodically queried by a batch job
- consistency between batch/streaming jobs (e.g. a bank account balance)

Stream Processing

Real life examples/use-cases:

- sensor readings
- interactions with an application/website
- credit card transactions
- real-time dashboards
- alerts and notifications
- incremental big data
- incremental transactional data e.g. analytics or accounts
- interactive machine learning

Data is better when consumed fresh

Pros/Cons

Pros

- Much lower latency than batch processing
- Greater performance/efficiency (especially with incremental data)

Difficulties

- Maintaining state and order for incoming data ← event time processing
- Exactly-once processing in the context of machine failures ← fault tolerance
- Responding to events at low latency
- Transactional data at runtime
- Updating business logic at runtime

Spark Streaming Principles

Declarative API

- write "what" needs to be computed, let the library decide "how"
- alternative: RaaT (record-at-a-time)
    - set of APIs to process each incoming element as it arrives
    - low-level: maintaining state & resource usage is your responsibility
    - hard to develop

Event time vs Processing time API

- event time = when the event was produced
- processing time = when the event arrives
- event time is critical: allows detection of late data points

Continuous vs micro-batch execution

- continuous = include each data point as it arrives ← lower latency
- micro-batch = wait for a few data points, process them all in the new result ← higher throughput

Low-level (DStreams) vs High-level API (Structured Streaming)

Spark Streaming Principles

Spark Streaming operates on micro-batches

(continuous execution is experimental)

Spark rocks