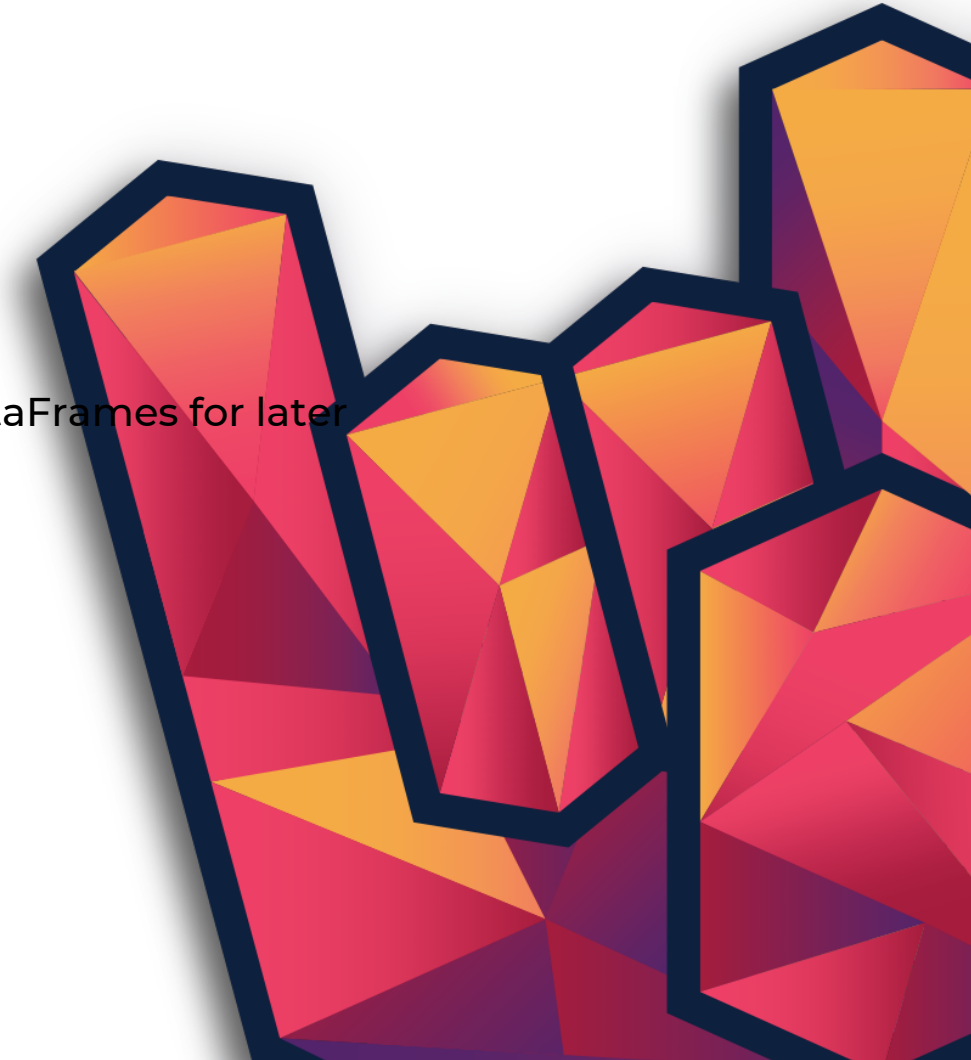
Checkpointing

Objective

Different technique for saving RDDs/DataFrames for later

vs Caching

Checkpointing recommendations



Checkpointing vs Caching

No main memory used, only disk

Takes more space and is slower than caching

Dependency graph is erased

Disk location is usually a cluster-available file system e.g. HDFS

Node failure with caching => partition is lost & needs to be recomputed

Node failure with checkpointing => partition is reloaded on another executor

Checkpointing

Saves the RDD/DF to external storage and forgets its lineage

Makes an intermediate RDD/DF available to other jobs

Takes more space and is slower than caching

Does not use Spark memory

Does not force recomputation of a partition if a node fails

When to Use What

Use checkpointing if you can't afford a recomputation

- example: a huge incremental dataset

If a job is slow, use caching

If a job is failing, use checkpointing

- OOMs are reduced as checkpoints don't use executor memory
- network/other errors are mitigated by breaking the job into several segments

```
sc.setCheckpointDir("data/checkpoints/")
```

config external location to store checkpoints

```
mySuperExpensiveRDD.checkpoint()
```

Spark rocks