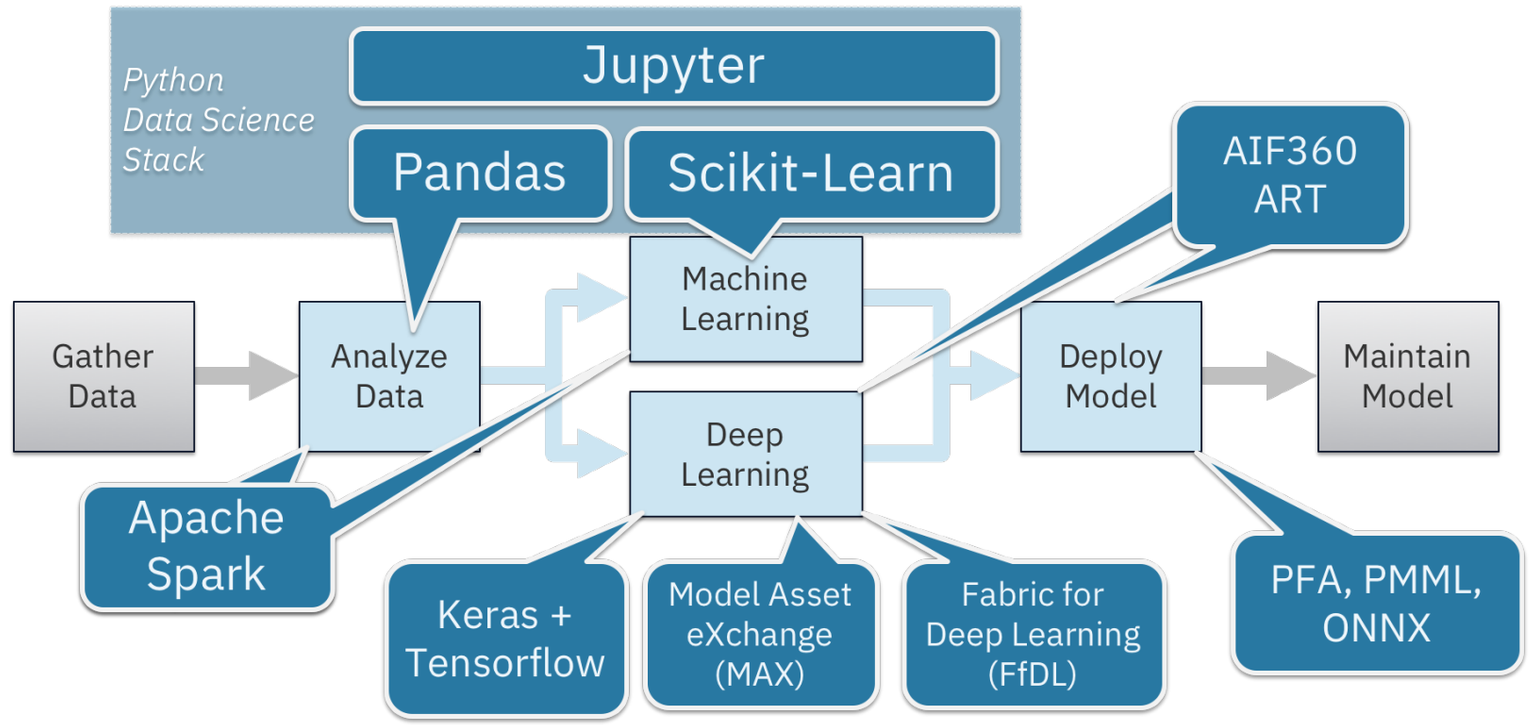
What's new in

Apache Spark 2.4

romeo kienzler

Center for Open-Source Data & AI Technologies

Improving the Enterprise AI Lifecycle in Open Source



- **contributes to over 10 open source projects**
- **17 Apache committers, 100+ contributors**
- **66,000+ LoC committed to Apache Spark**
- **65,000+ LoC committed to Apache SystemML**
- **Over 25 product lines within IBM using Apache Spark**
- **#2 contributor to KubeFlow**

IBM CODAIT

romeo kienzler

Spark in a nutshell

romeo kienzler

JVM
Process

Driver JVM

Compute Node

Driver JVM

Compute Node

Executor JVM

Driver JVM

Compute Node

Executor JVM

Executor JVM

Executor JVM

Driver JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Driver JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Driver JVM

Compute Node

Executor JVM

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Executor JVM

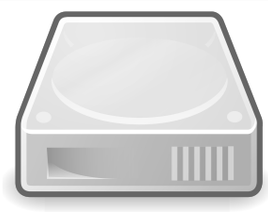
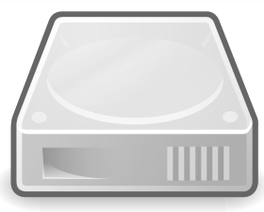
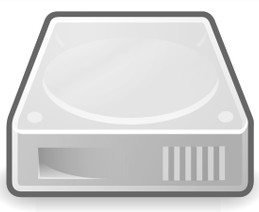
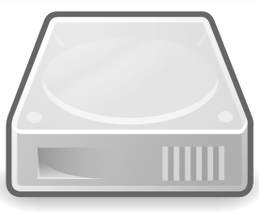
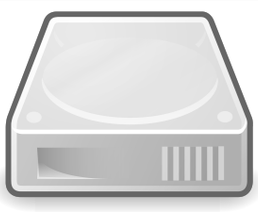
Compute Node

Executor JVM

Executor JVM

Executor JVM

HDFS

Compute Node	Compute Node	Compute Node	Compute Node	Compute Node
Executor JVM	Executor JVM	Executor JVM	Executor JVM	Executor JVM
Executor JVM	Executor JVM	Executor JVM	Executor JVM	Executor JVM
Executor JVM	Executor JVM	Executor JVM	Executor JVM	Executor JVM
Executor JVM	Executor JVM	Executor JVM	Executor JVM	Executor JVM
				
Part of File	Part of File	Part of File	Part of File	Part of File

HDFS

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

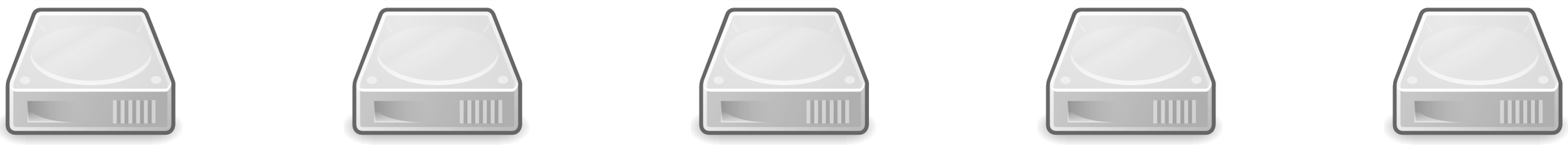
Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM



Part of File | Part of File | Part of File | Part of File | Part of File

Virtual File

RDD

- “Resilient Distributed Dataset“
- Distributed, immutable collection of data
- Created from HDFS, ObjectStore, Cloudant NoSQL, dashDB SQL, simple files, …
- In-memory, but spillable to disk
- lazy

Summary

- ApacheSpark programs are implicitly parallel
- Same code can process 1 KB or 1 PB
- RDD central API
- Data and task distribution transparent

Programming Language options on ApacheSpark

- ApacheSpark itself is implemented in Scala
- ApacheSpark runs on top if the JVM
- ApacheSpark jobs can be implemented in
  - Java
  - Scala
  - Python
  - R

Scala

- Scala is native to ApacheSpark
- Complete API set is available
- Scala code normally runs fastest

Java

- Java not a Data Science programming language
- Complete API set is available
- De-factor standard in Enterprise IT
- Language of “Hadoop”

R

- R is THE Data Science programming language
- Only subset of ApacheSpark API is available
- De-factor standard in academic research
- > 8000 add-on packages available
- Awesome plotting and charting capabilities
- Slow

Python

- as widely used in Data Science as R
- Only subset of ApacheSpark API is available
- nice plotting and charting but R is better here
- can get slow

Summary

	Scala	Java	R	Python
Spark API support	complete	complete	very limited	limited
ease of use	low	very low	high	very high
Speed	very high	high	very low	low
3rd party libraries	few	few	many	many

Functional Programming basics

Functional Programming (FP)

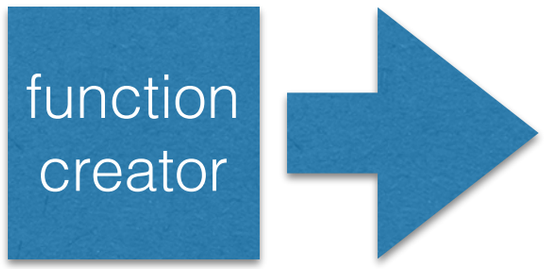
- mathematical concept of “Lambda Calculus
- 1st implementation was LISP in the 50s
- Haskell is part of every computer science curriculum since the 90s
- Scala most recent representative
- Python, R and Java also rudimentary support FP

Idea

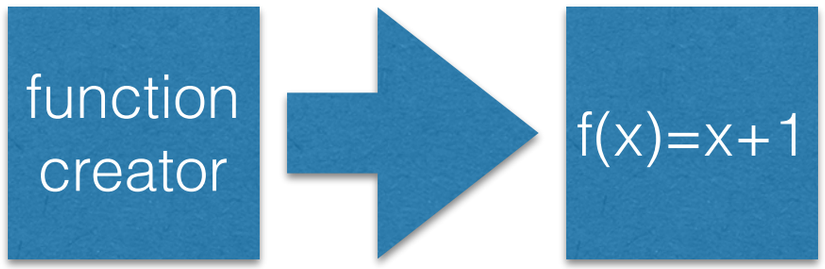
Idea

function creator

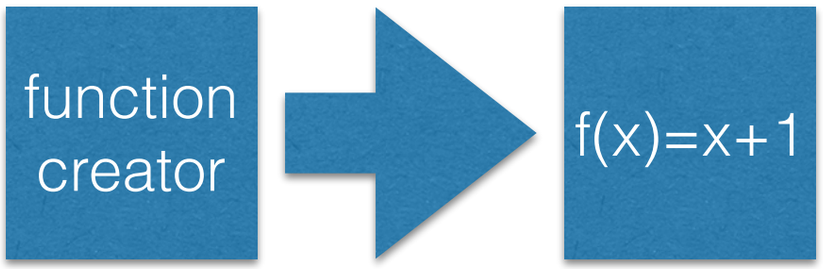
Idea

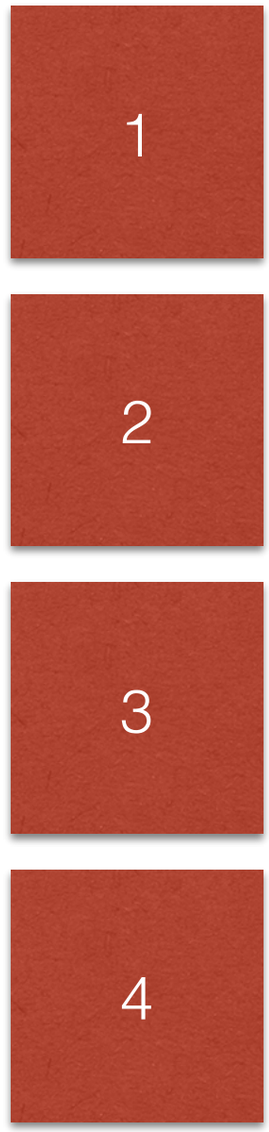


Idea

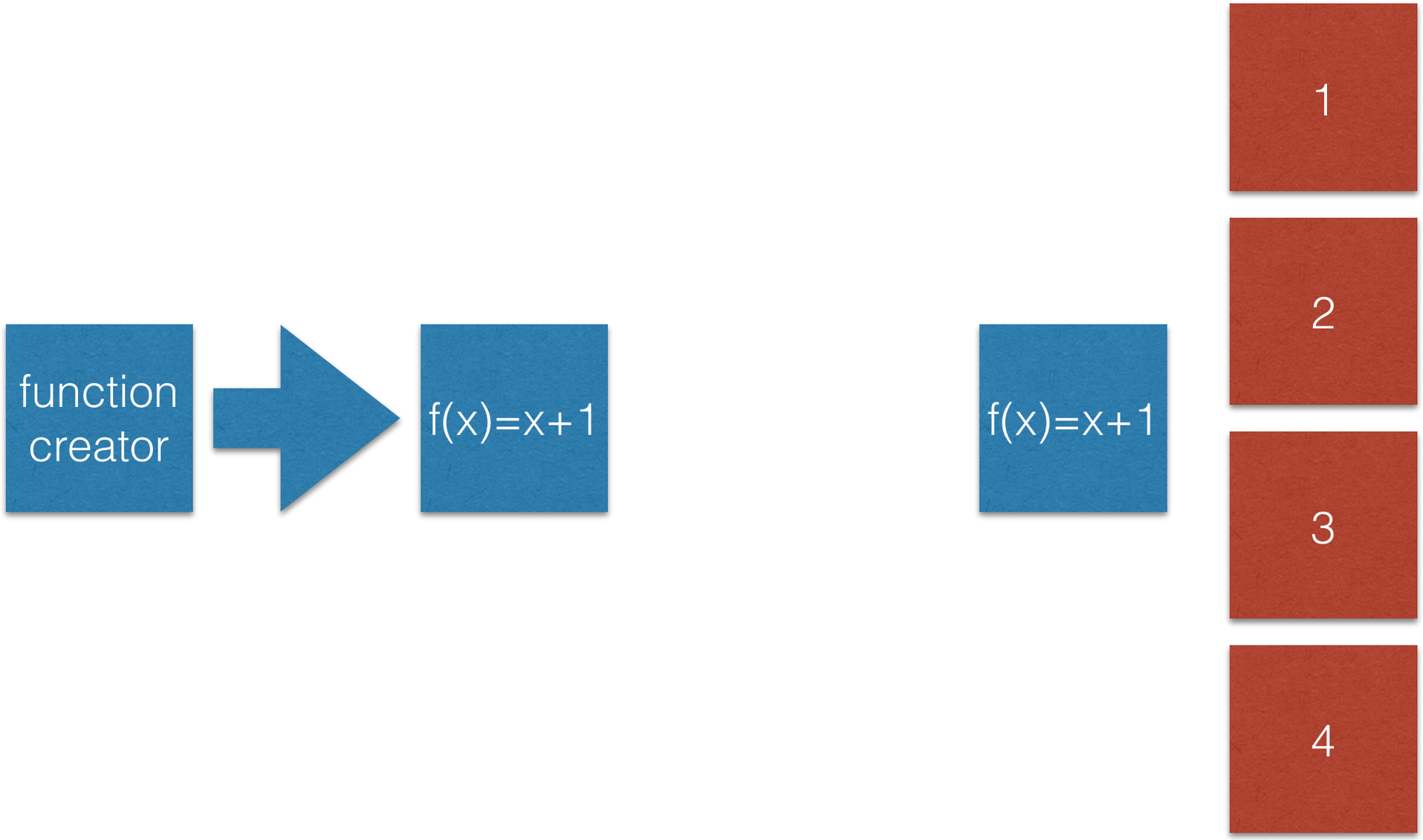


Idea

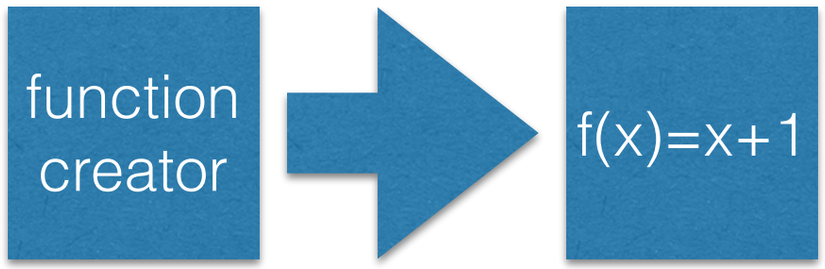


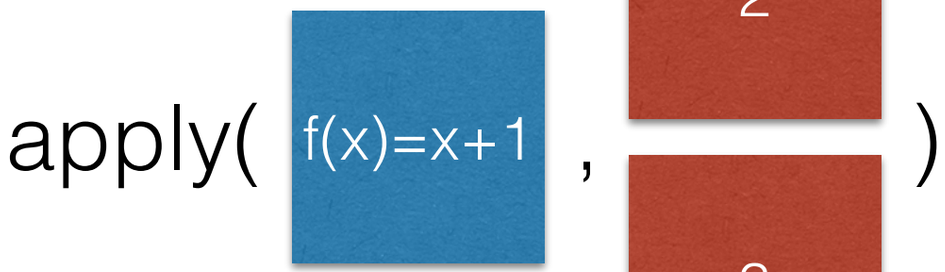


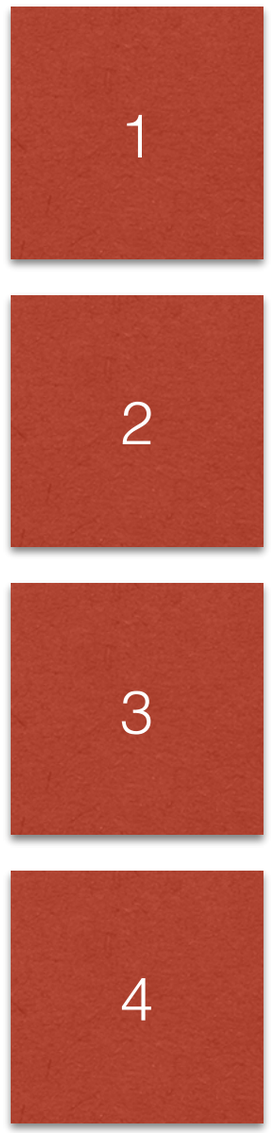
Idea



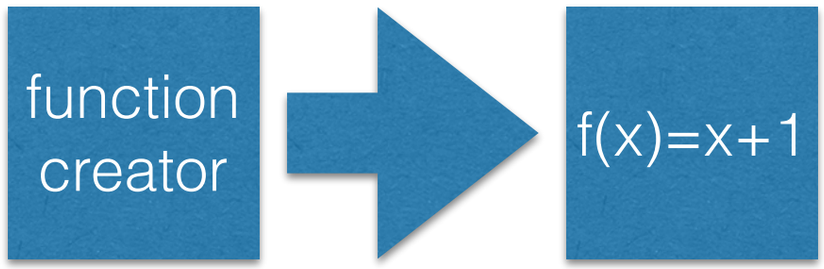
Idea

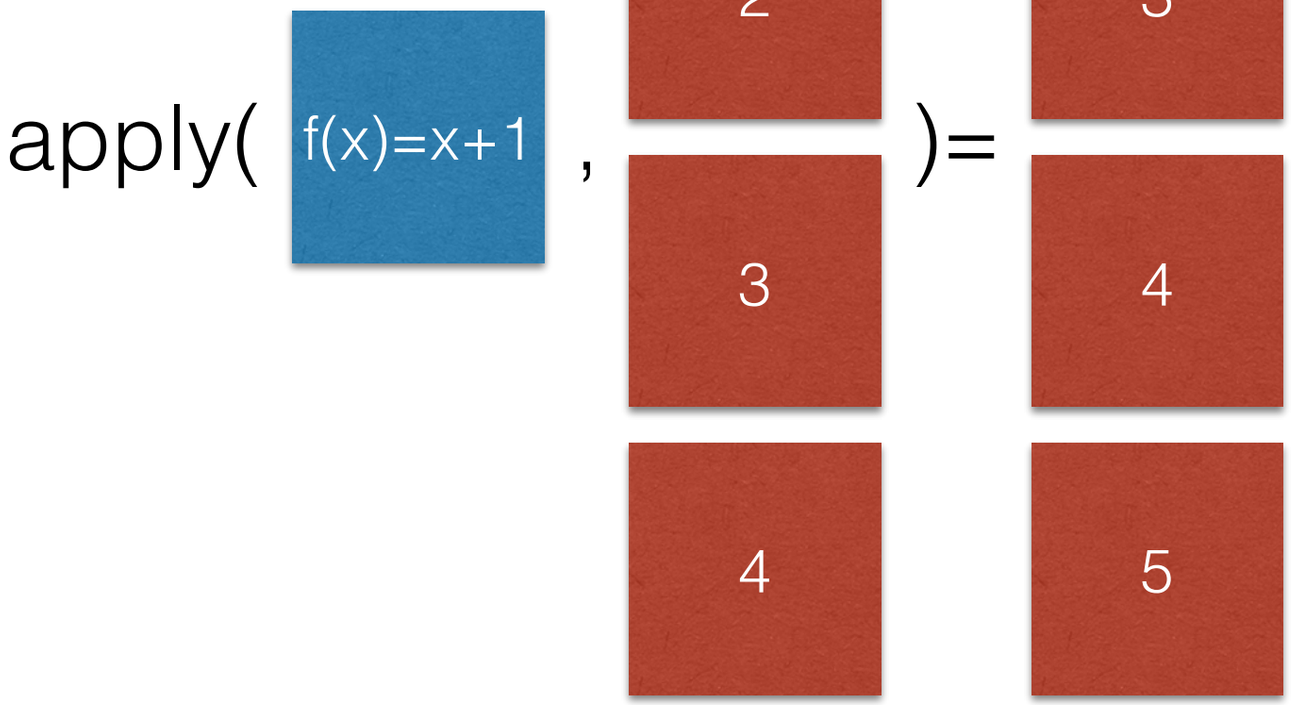






Idea





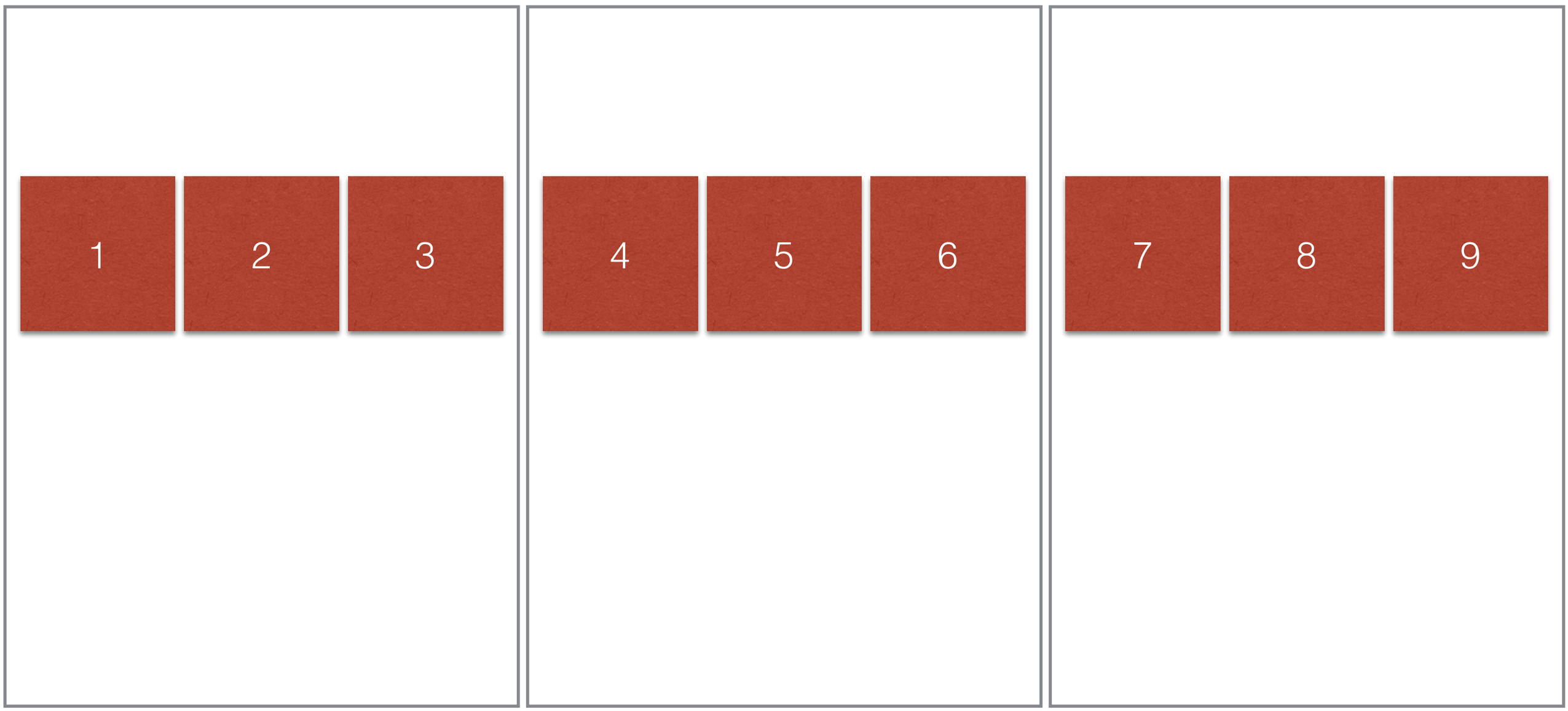
Parallelisation

f(x)=x+1

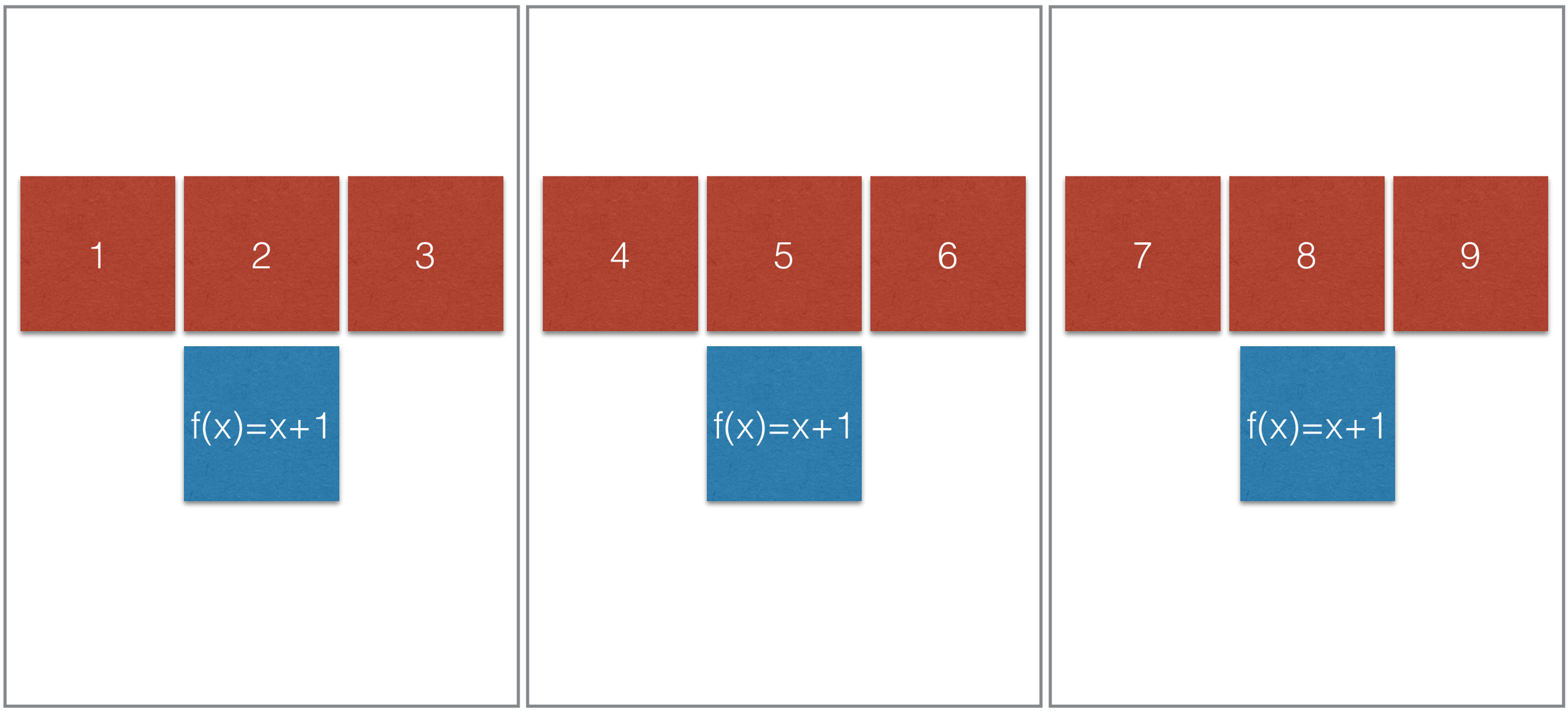
Parallelisation

f(x)=x+1

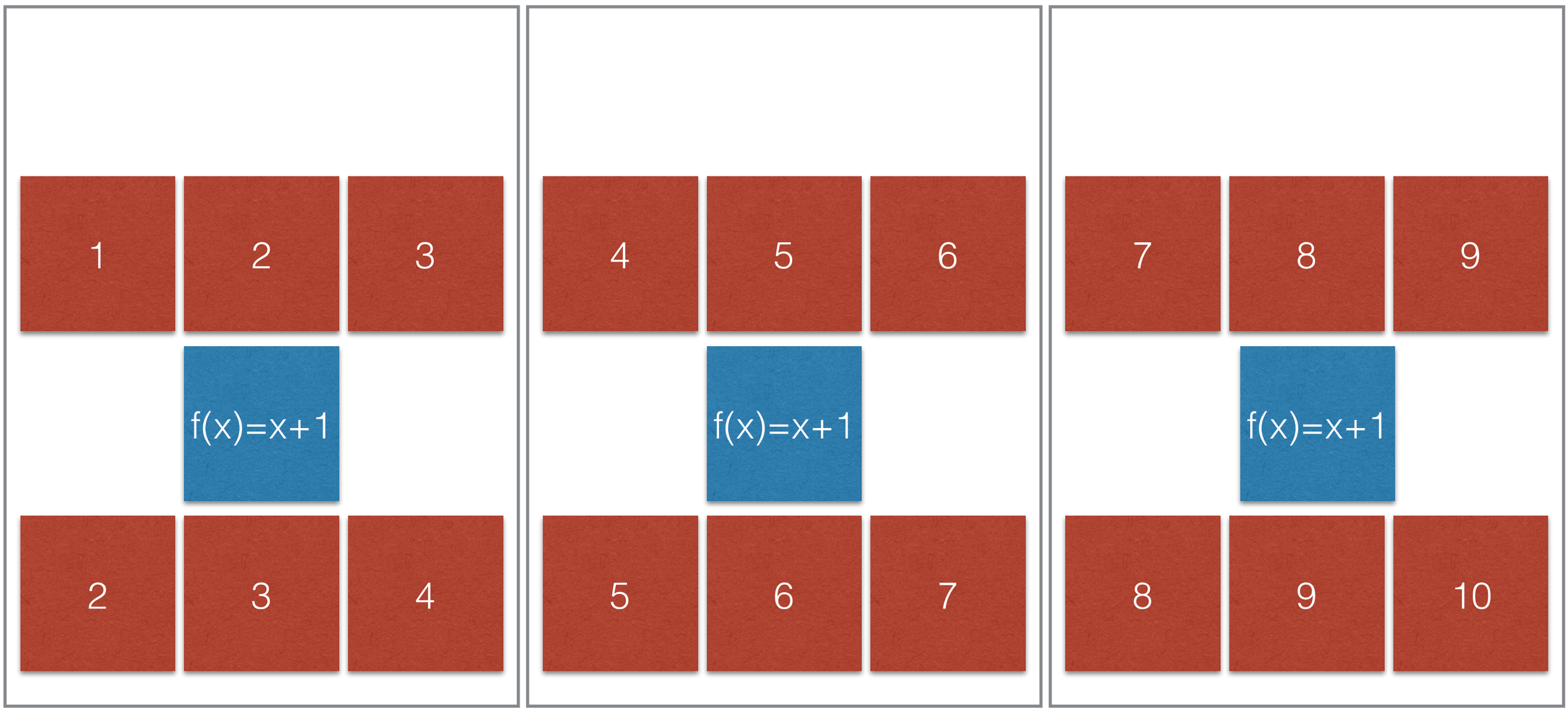
Parallelisation



Parallelisation



Parallelisation



Summary

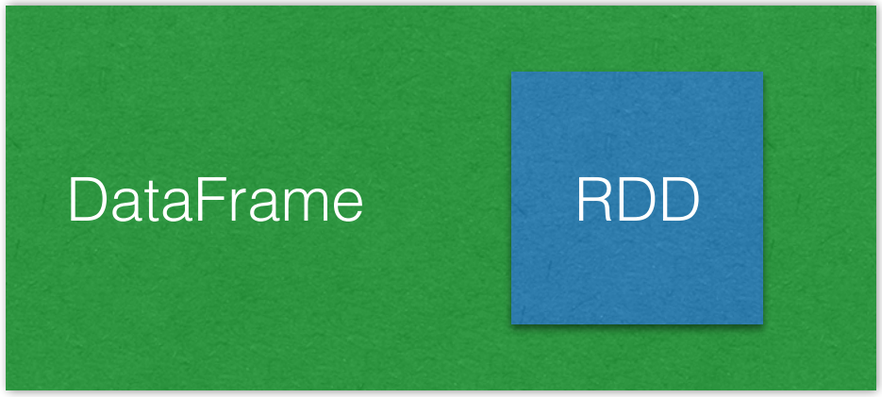
- ApacheSpark parallelises computations using the lambda calculus

- All functional spark programs are inherently parallel

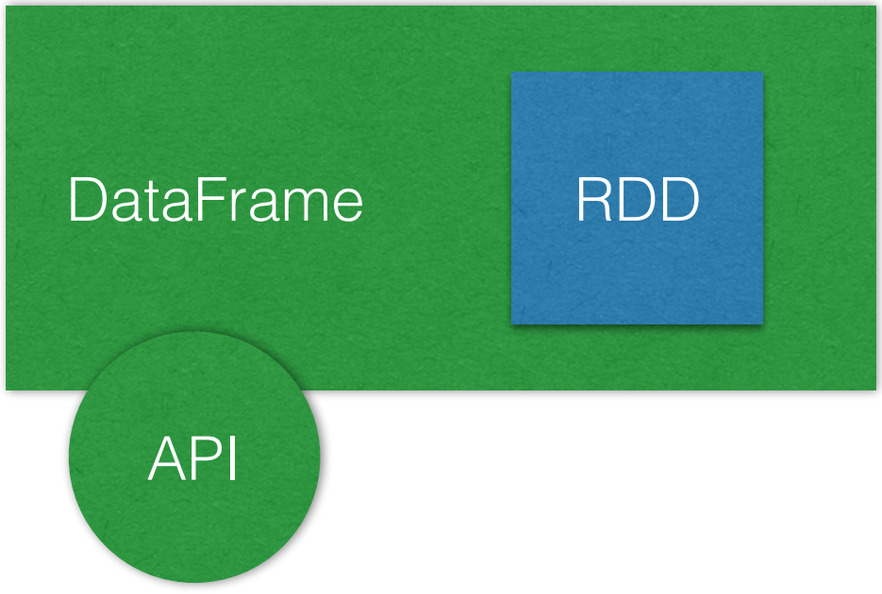
ApacheSparkSQL

ApacheSparkSQL

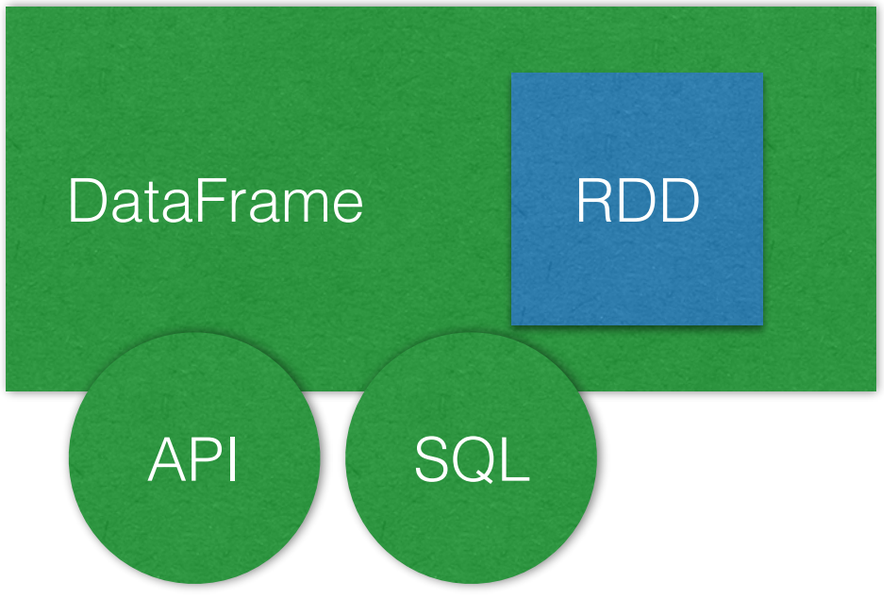
ApacheSparkSQL



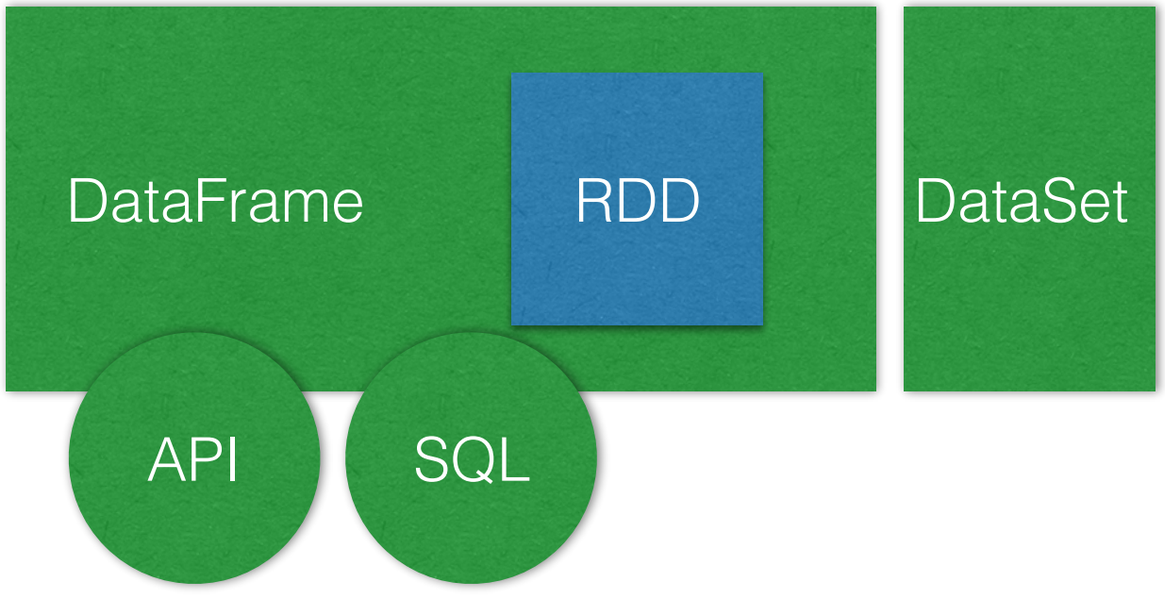
ApacheSparkSQL



ApacheSparkSQL



ApacheSparkSQL



Schemas

- RDDs are schema less (schema on read)
- DataFrames have a schema
  - lazy, inferred
  - explicitly defined

The “Catalyst”

- Creates “logical execution plan” (LEP) from SQL
- Optimises LEP to “physical execution plans” (PEPs)
  - based on statistics chooses best PEP to execute
  - similar to cost based optimisers in RDBMs

Project Tungsten

- Java Virtual Machine (JVM) is an art piece
- General purpose byte code execution engine
- JVM objects & Garbage Collection (GC) overhead
  - 4 byte string is 48 byte on the JVM
  - GC optimises on object life time estimation
  - Spark knows this better than JVM

Project Tungsten

- L1/L2/L3 Cache friendly data structures
- Code generation to remove
  - boxing of primitive types
  - polymorphic function dispatching

Summary

- ApacheSpark supports SQL via data frame API
- Internally still RDDs are used
- Makes writing ApacheSpark jobs easier
- Performance benefits through Catalyst & Tungsten

Spark 2.4

Core and Spark SQL

- **Major features**
  - **Barrier Execution Mode**: [SPARK-24374] Support Barrier Execution Mode in the scheduler, to better integrate with deep learning frameworks.
  - **Scala 2.12 Support**: [SPARK-14220] Add experimental Scala 2.12 support. Now you can build Spark with Scala 2.12 and write Spark applications in Scala 2.12.
  - **Higher-order functions**: [SPARK-23899] Add a lot of new built-in functions, including higher-order functions, to deal with complex data types easier.
  - **Built-in Avro data source**: [SPARK-24768] Inline Spark-Avro package with logical type support, better performance and usability.
- **API**
  - [SPARK-24035] SQL syntax for Pivot
  - [SPARK-24940] Coalesce and Repartition Hint for SQL Queries
  - [SPARK-19602] Support column resolution of fully qualified column name
  - [SPARK-21274] Implement EXCEPT ALL and INTERSECT ALL
- **Performance and stability**
  - [SPARK-16406] Reference resolution for large number of columns should be faster
  - [SPARK-23486] Cache the function name from the external catalog for lookupFunctions
  - [SPARK-23803] Support Bucket Pruning
  - [SPARK-24802] Optimization Rule Exclusion
  - [SPARK-4502] Nested schema pruning for Parquet tables
  - [SPARK-24296] Support replicating blocks larger than 2 GB
  - [SPARK-24307] Support sending messages over 2GB from memory
  - [SPARK-23243] Shuffle+Repartition on an RDD could lead to incorrect answers
  - [SPARK-25181] Limited the size of BlockManager master and slave thread pools, lowering memory overhead when networking is slow
- **Connectors**
  - [SPARK-23972] Update Parquet from 1.8.2 to 1.10.0
  - [SPARK-25419] Parquet predicate pushdown improvement
  - [SPARK-23456] Native ORC reader is on by default
  - [SPARK-22279] Use native ORC reader to read Hive serde tables by default
  - [SPARK-21783] Turn on ORC filter push-down by default
  - [SPARK-24959] Speed up count() for JSON and CSV
  - [SPARK-24244] Parsing only required columns to the CSV parser
  - [SPARK-23786] CSV schema validation - column names are not checked
  - [SPARK-24423] Option query for specifying the query to read from JDBC
  - [SPARK-22814] Support Date/Timestamp in JDBC partition column
  - [SPARK-24771] Update Avro from 1.7.7 to 1.8
- **Kubernetes Scheduler Backend**
  - [SPARK-23984] PySpark bindings for K8S
  - [SPARK-24433] R bindings for K8S
  - [SPARK-23146] Support client mode for Kubernetes cluster backend
  - [SPARK-23529] Support for mounting K8S volumes
- **PySpark**
  - [SPARK-24215] Implement eager evaluation for DataFrame APIs
  - [SPARK-22274] User-defined aggregation functions with Pandas UDF
  - [SPARK-22239] User-defined window functions with Pandas UDF
  - [SPARK-24396] Add Structured Streaming ForeachWriter for Python
  - [SPARK-23874] Upgrade Apache Arrow to 0.10.0
  - [SPARK-25004] Add spark.executor.pyspark.memory limit
  - [SPARK-23030] Use Arrow stream format for creating from and collecting Pandas DataFrames
  - [SPARK-24624] Support mixture of Python UDF and Scalar Pandas UDF
- **Other notable changes**
  - [SPARK-24596] Non-cascading Cache Invalidation
  - [SPARK-23880] Do not trigger any job for caching data
  - [SPARK-23510] Support Hive 2.2 and Hive 2.3 metastore
  - [SPARK-23711] Add fallback generator for UnsafeProjection
  - [SPARK-24626] Parallelize location size calculation in Analyze Table command

Programming guides: Spark RDD Programming Guide and Spark SQL, DataFrames and Datasets Guide.

Structured Streaming

- **Major features**
  - [SPARK-24565] Exposed the output rows of each microbatch as a DataFrame using foreachBatch (Python, Scala, and Java)
  - [SPARK-24396] Added Python API for foreach and ForeachWriter
  - [SPARK-25005] Support "kafka.isolation.level" to read only committed records from Kafka topics that are written using a transactional producer.
- **Other notable changes**
  - [SPARK-24662] Support the LIMIT operator for streams in Append or Complete mode
  - [SPARK-24763] Remove redundant key data from value in streaming aggregation
  - [SPARK-24156] Faster generation of output results and/or state cleanup with stateful operations (mapGroupsWithState, stream-stream join, streaming aggregation, streaming dropDuplicates) when there is no data in the input stream.
  - [SPARK-24730] Support for choosing either the min or max watermark when there are multiple input streams in a query
  - [SPARK-25399] Fixed a bug where reusing execution threads from continuous processing for microbatch streaming can result in a correctness issue
  - [SPARK-18057] Upgraded Kafka client version from 0.10.0.1 to 2.0.0

Programming guide: Structured Streaming Programming Guide.

MLlib

- **Major features**
  - [SPARK-22666] Spark datasource for image format
- **Other notable changes**
  - [SPARK-22119] Add cosine distance measure to KMeans/BisectingKMeans/Clustering evaluator
  - [SPARK-10697] Lift Calculation in Association Rule mining
  - [SPARK-14682] Provide evaluateEachIteration method or equivalent for spark.ml GBTs
  - [SPARK-7132] Add fit with validation set to spark.ml GBT
  - [SPARK-15784] Add Power Iteration Clustering to spark.ml
  - [SPARK-15064] Locale support in StopWordsRemover
  - [SPARK-21741] Python API for DataFrame-based multivariate summarizer
  - [SPARK-21898] Feature parity for KolmogorovSmirnovTest in MLlib
  - [SPARK-10884] Support prediction on single instance for regression and classification related models
  - [SPARK-23783] Add new generic export trait for ML pipelines
  - [SPARK-11239] PMML export for ML linear regression

Programming guide: Machine Learning Library (MLlib) Guide.

SparkR

- **Major features**
  - [SPARK-25393] Adding new function from_csv()
  - [SPARK-21291] add R partitionBy API in DataFrame
  - [SPARK-25007] Add array_intersect/array_except/array_union/shuffle to SparkR
  - [SPARK-25234] avoid integer overflow in parallelize
  - [SPARK-25117] Add EXCEPT ALL and INTERSECT ALL support in R
  - [SPARK-24537] Add array_remove / array_zip / map_from_arrays / array_distinct
  - [SPARK-24187] Add array_join function to SparkR
  - [SPARK-24331] Adding arrays_overlap, array_repeat, map_entries to SparkR
  - [SPARK-24198] Adding slice function to SparkR
  - [SPARK-24197] Adding array_sort function to SparkR
  - [SPARK-24185] add flatten function to SparkR
  - [SPARK-24069] Add array_min / array_max functions
  - [SPARK-24054] Add array_position function / element_at functions
  - [SPARK-23770] Add repartitionByRange API in SparkR

Programming guide: SparkR (R on Spark).

GraphX

- **Major features**
  - [SPARK-25268] run Parallel Personalized PageRank throws serialization Exception

Barrier Execution

Spark / SPARK-24374

SPIP: Support Barrier Execution Mode in Apache Spark

Comment | Agile Board | More

Details

Type:	Epic	Status:	IN PROGRESS
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	2.4.0	Fix Version/s:	None
Component/s:	ML, Spark Core		
Labels:	Hydrogen, SPIP		
Epic Name:	Support Barrier Execution Mode		

Description

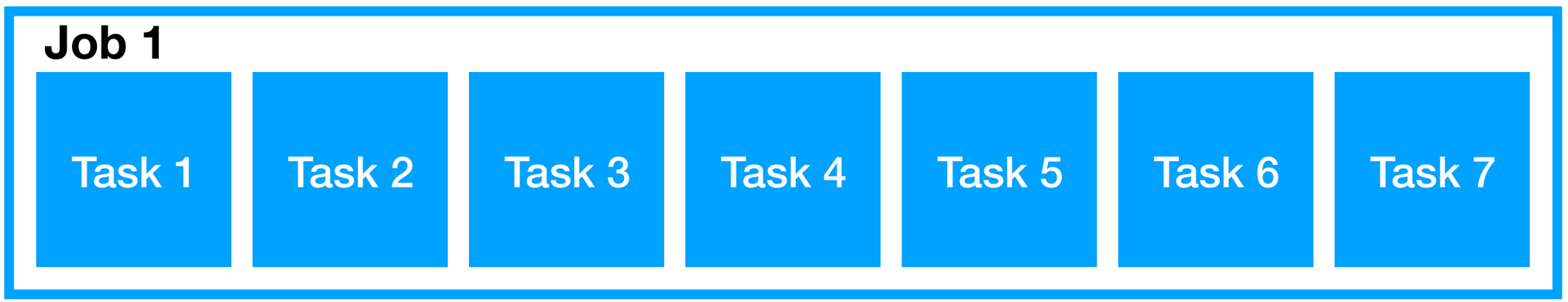
(See details in the linked/attached SPIP doc.)

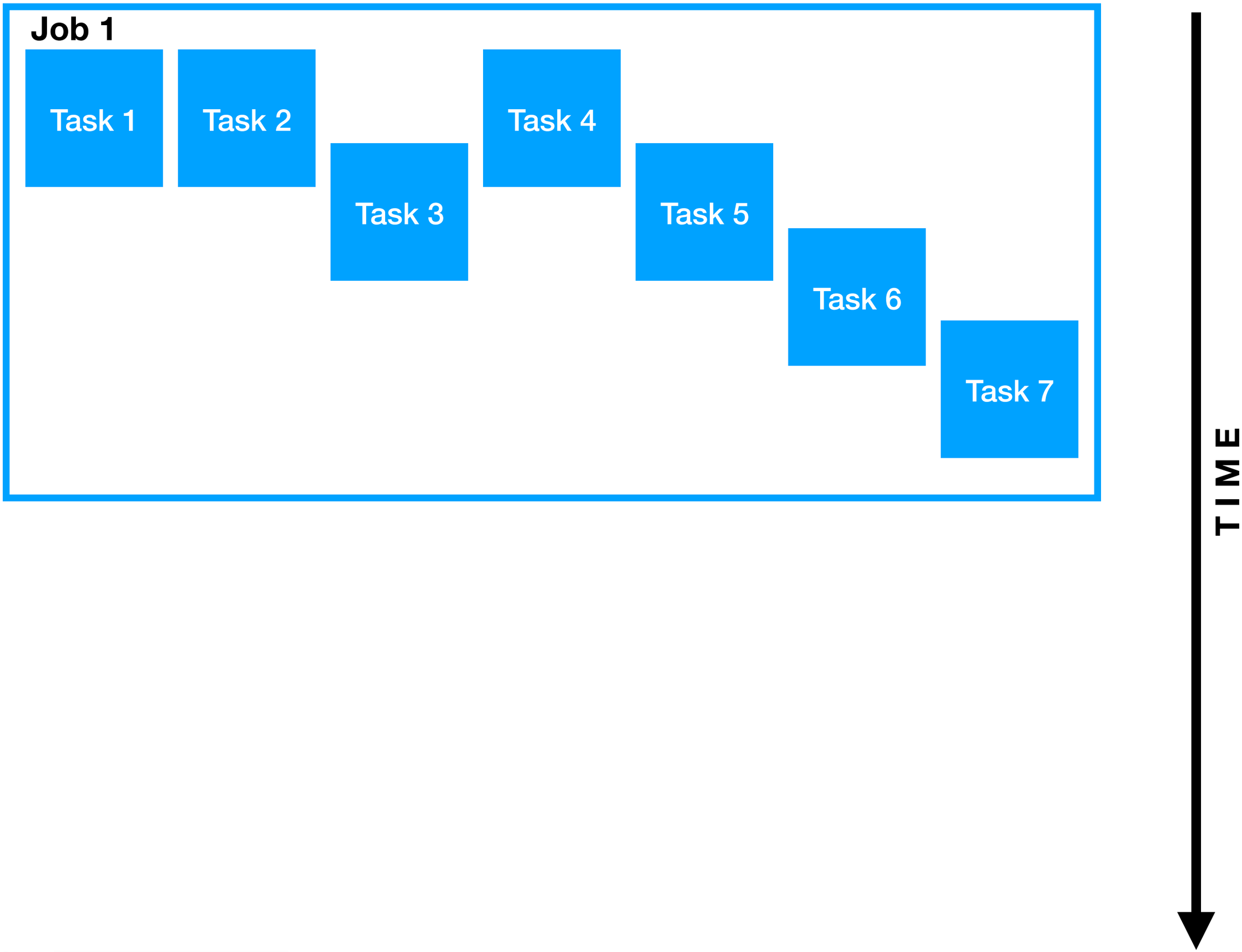
> The proposal here is to add a new scheduling model to Apache Spark so users can properly embed distributed DL training as a Spark stage to simplify the distributed training workflow. For example, Horovod uses MPI to implement all-reduce to accelerate distributed TensorFlow training. The computation model is different from MapReduce used by Spark. In Spark, a task in a stage doesn't depend on any other tasks in the same stage, and hence it can be scheduled independently. In MPI, all workers start at the same time and pass messages around. To embed this workload in Spark, we need to introduce a new scheduling model, tentatively named "barrier scheduling", which launches tasks at the same time and provides users enough information and tooling to embed distributed DL training. Spark can also provide an extra layer of fault tolerance in case some tasks failed in the middle, where Spark would abort all tasks and restart the stage.

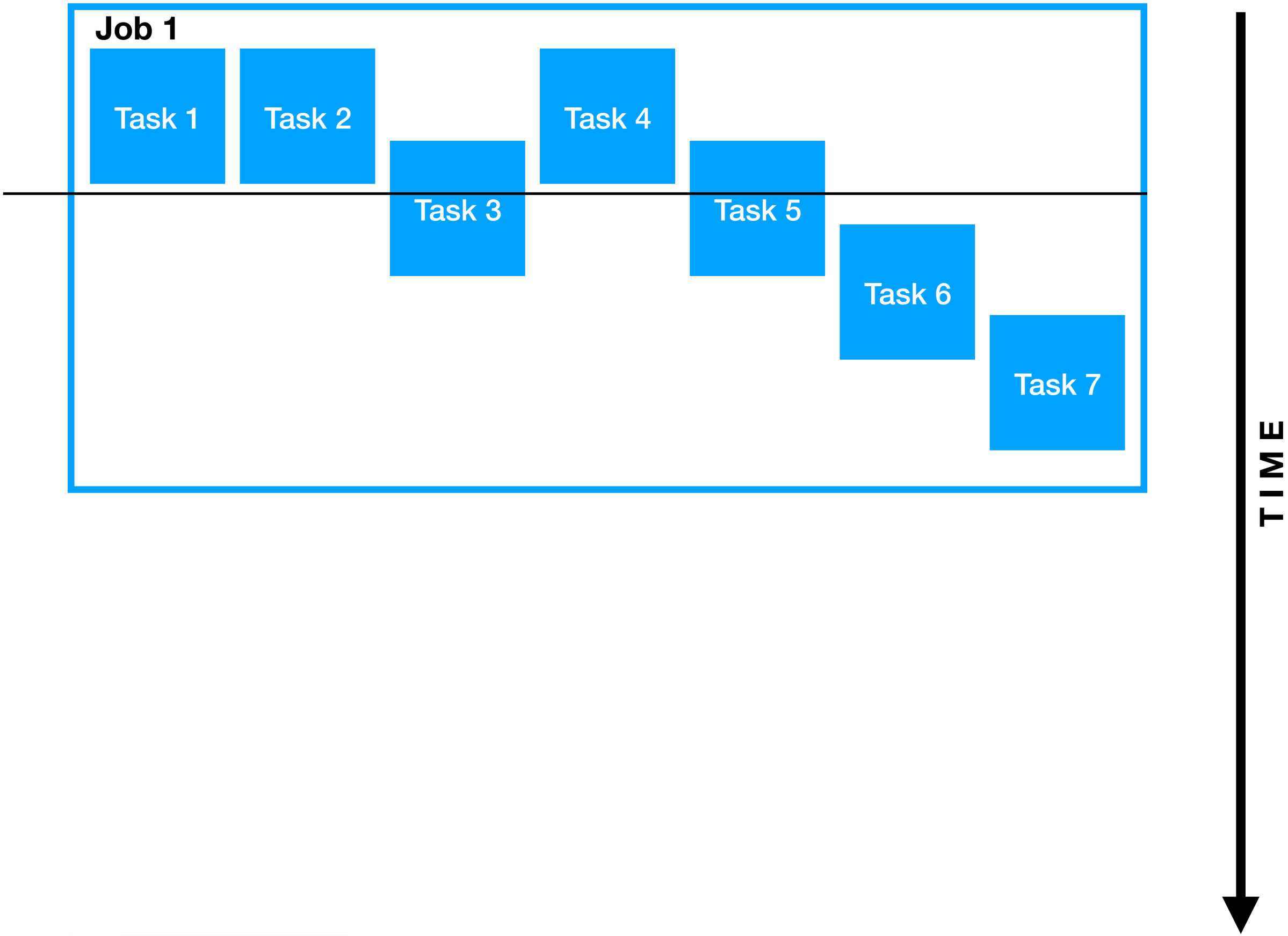
Job vs. Task

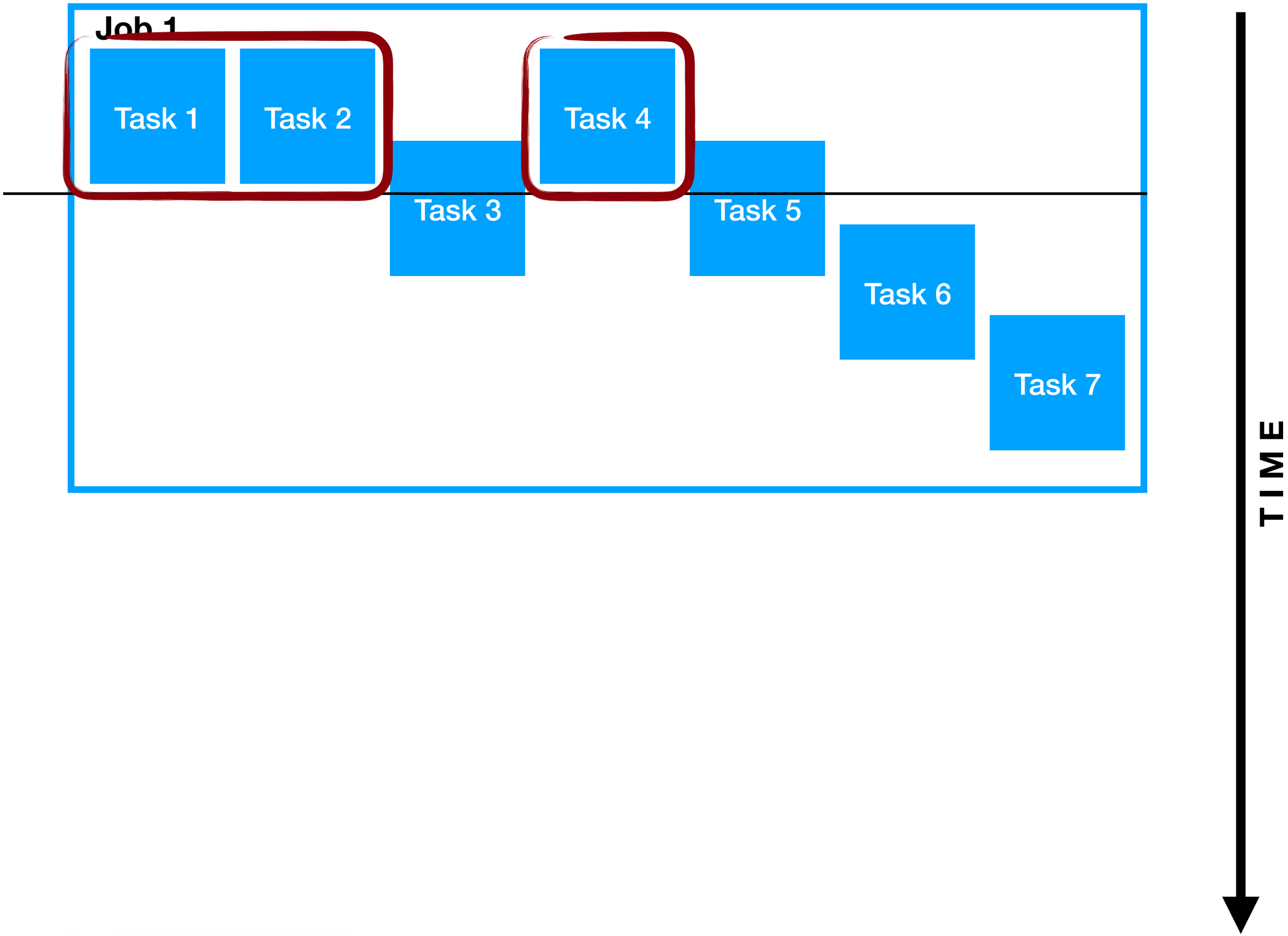
romeo kienzler

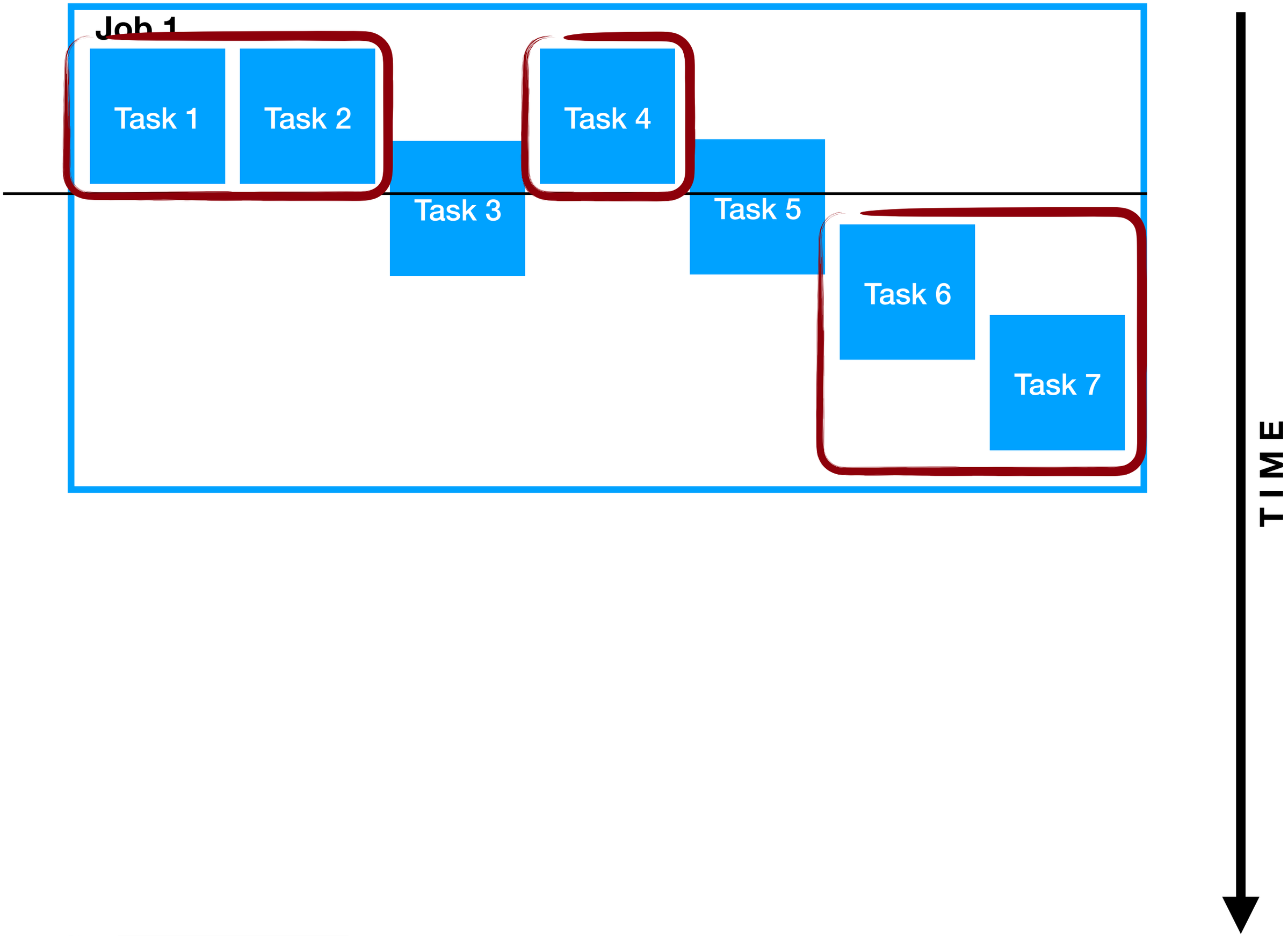
Job vs. Task



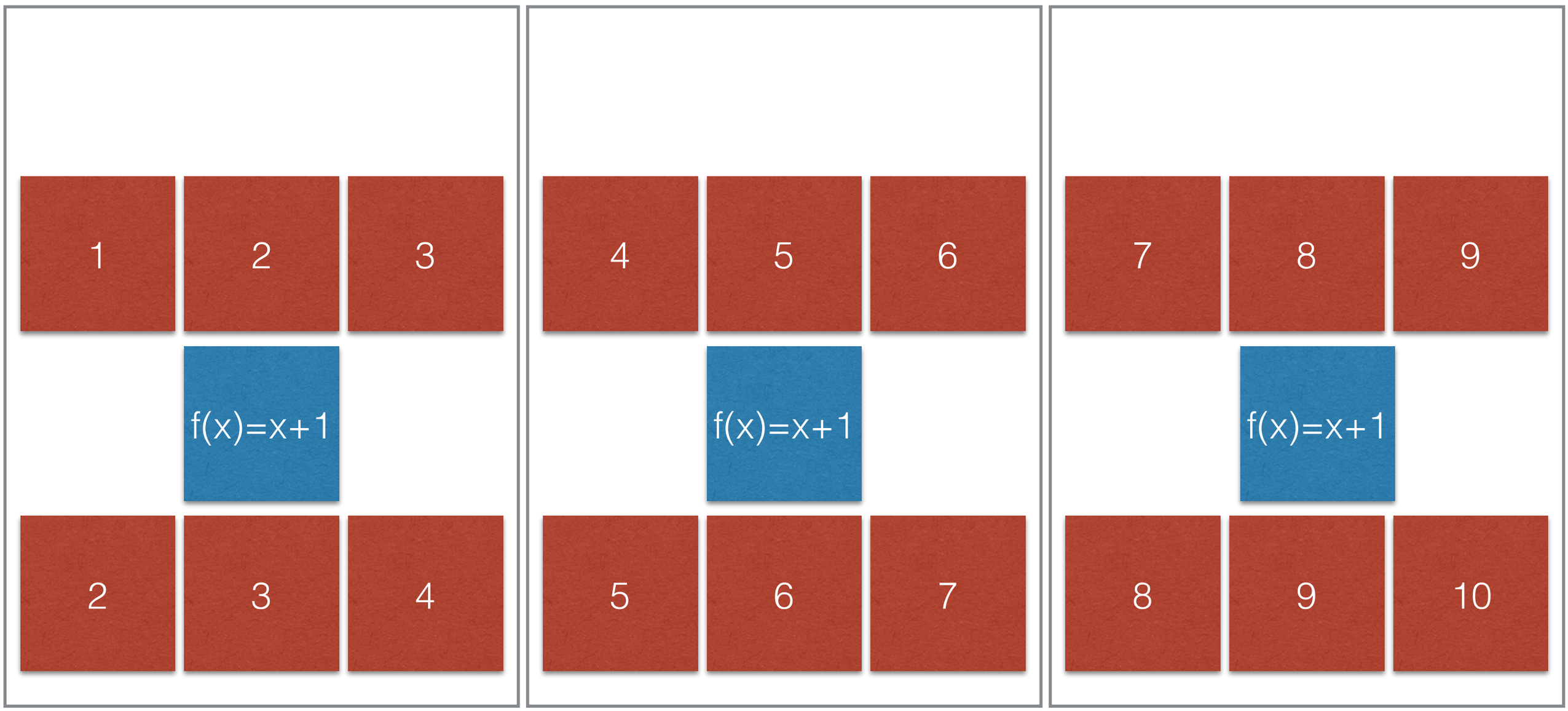


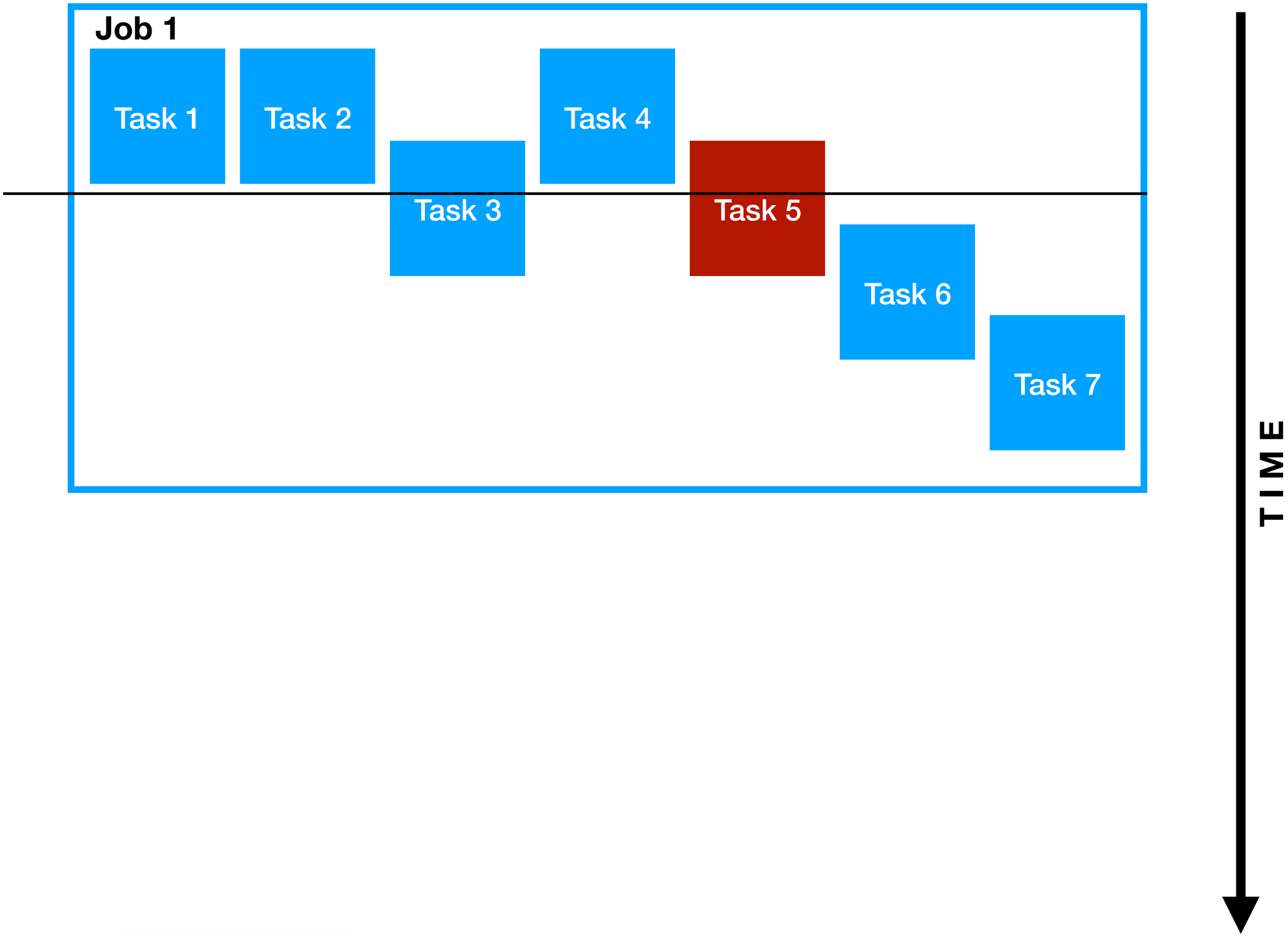


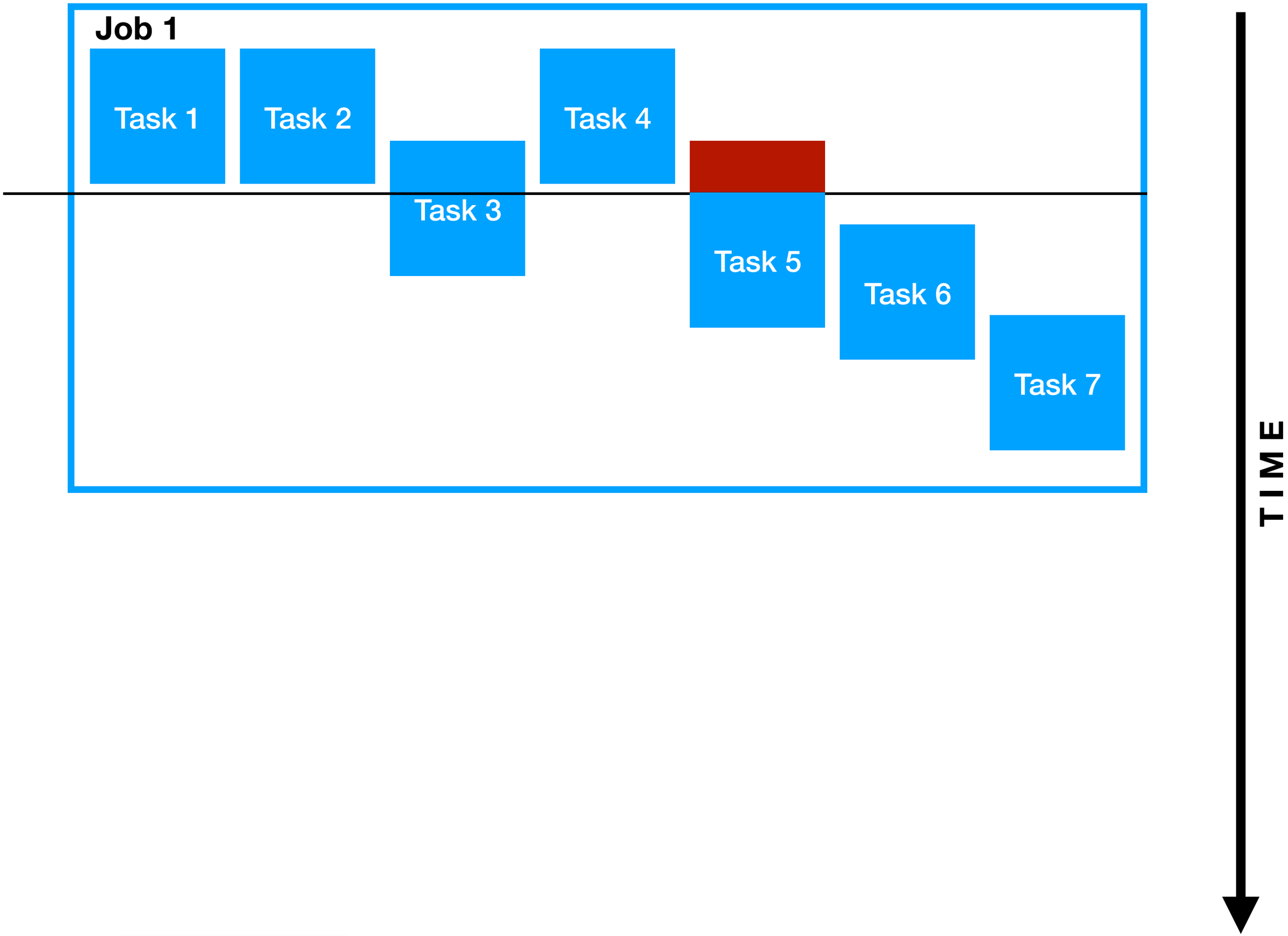




Parallelisation

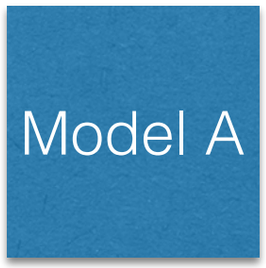






data parallelism

aka. “Jeff Dean style” parameter averaging

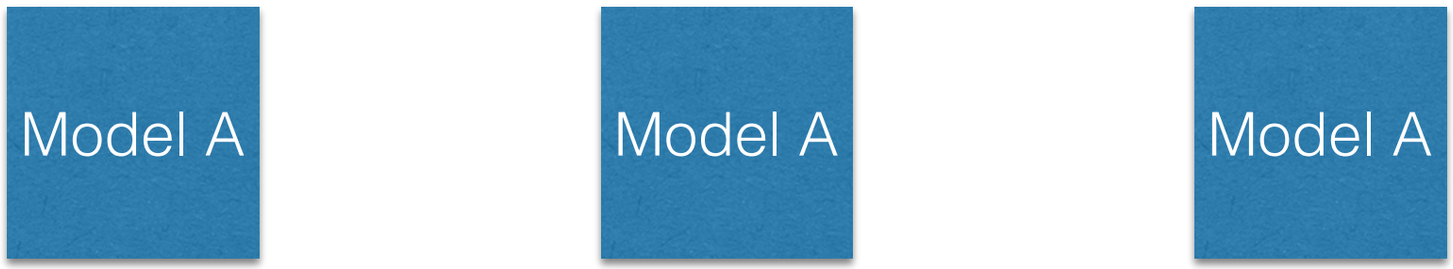


data parallelism

aka. “Jeff Dean style” parameter averaging

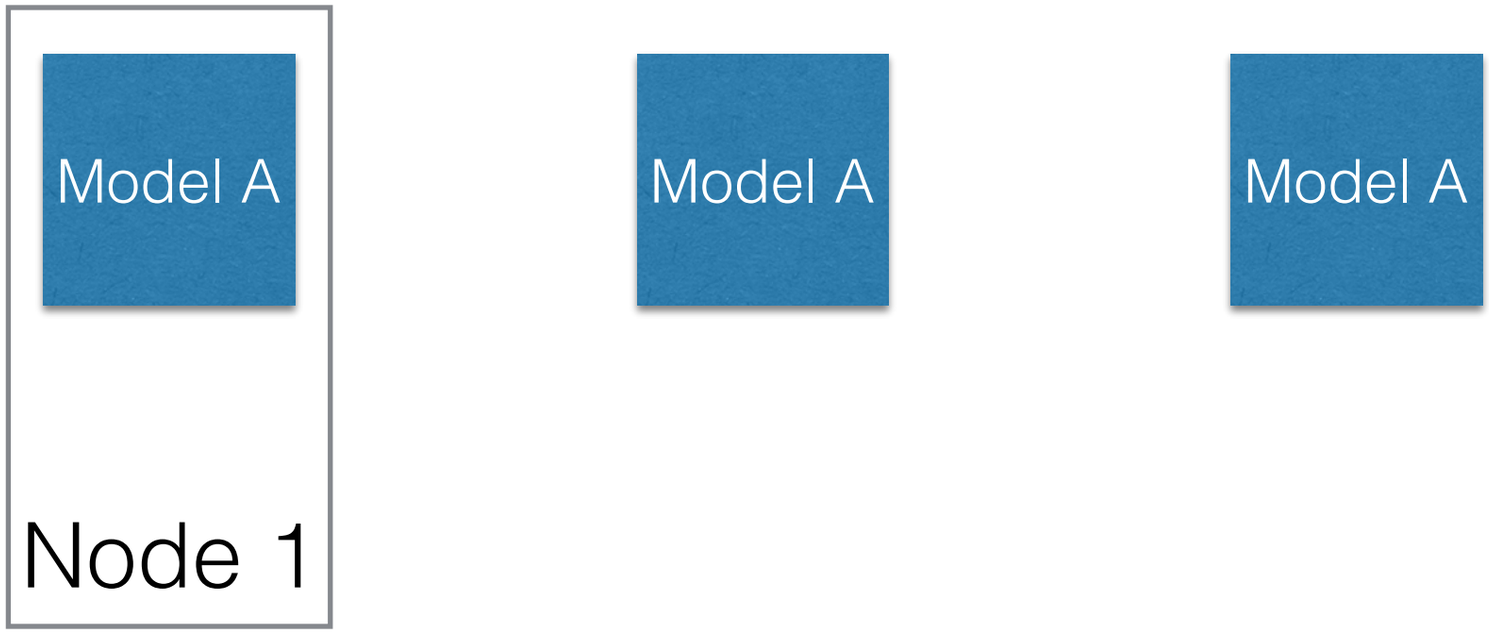
data parallelism

aka. “Jeff Dean style” parameter averaging



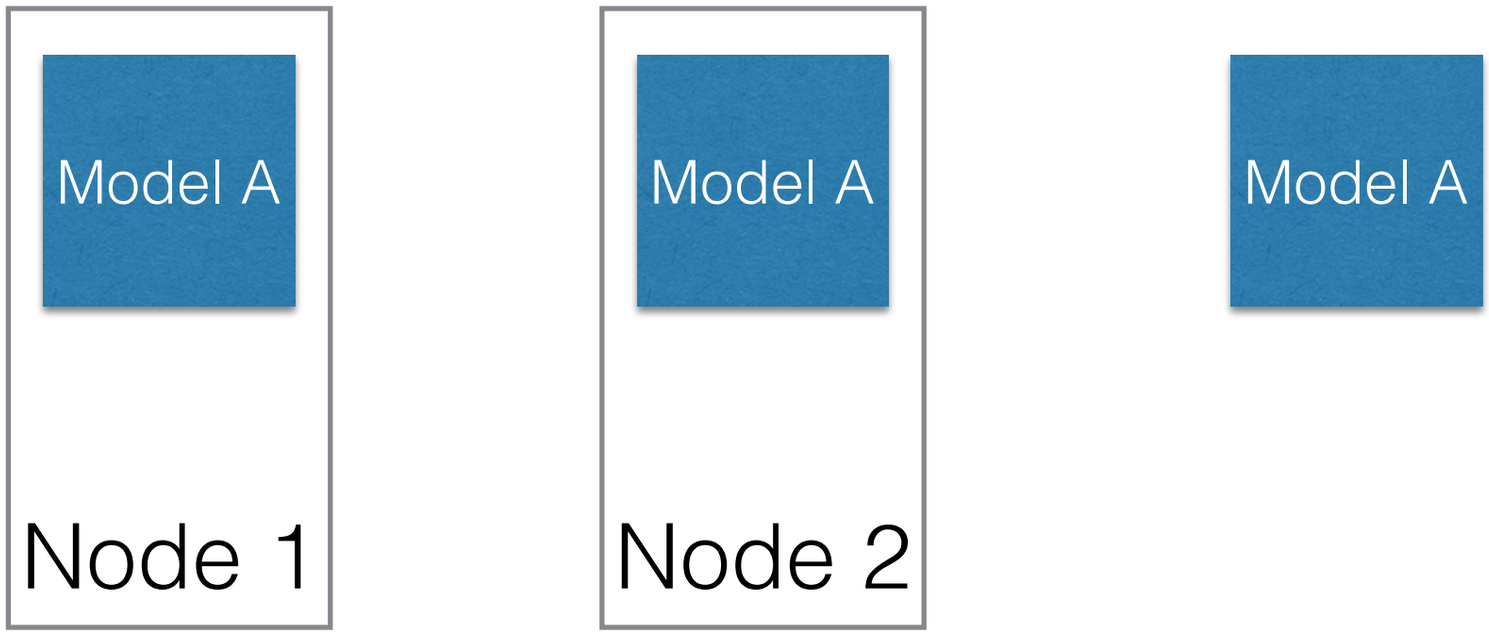
data parallelism

aka. “Jeff Dean style” parameter averaging



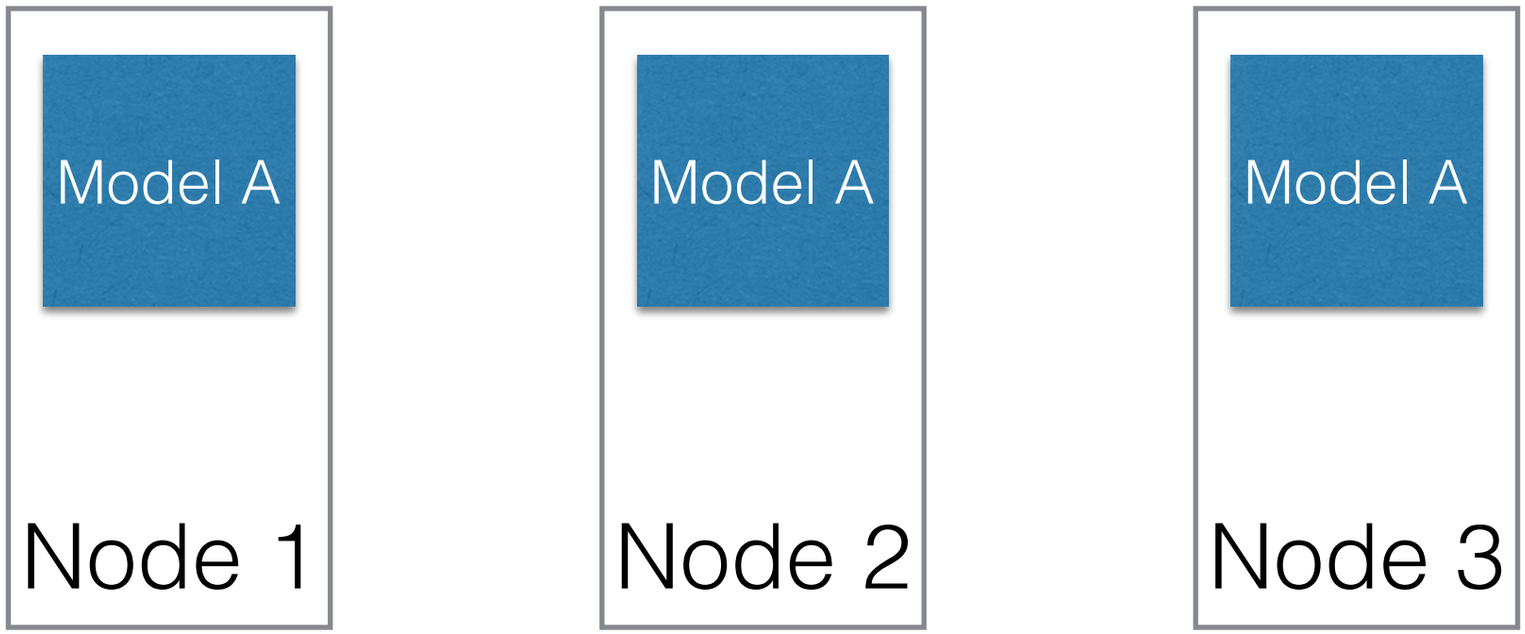
data parallelism

aka. “Jeff Dean style” parameter averaging



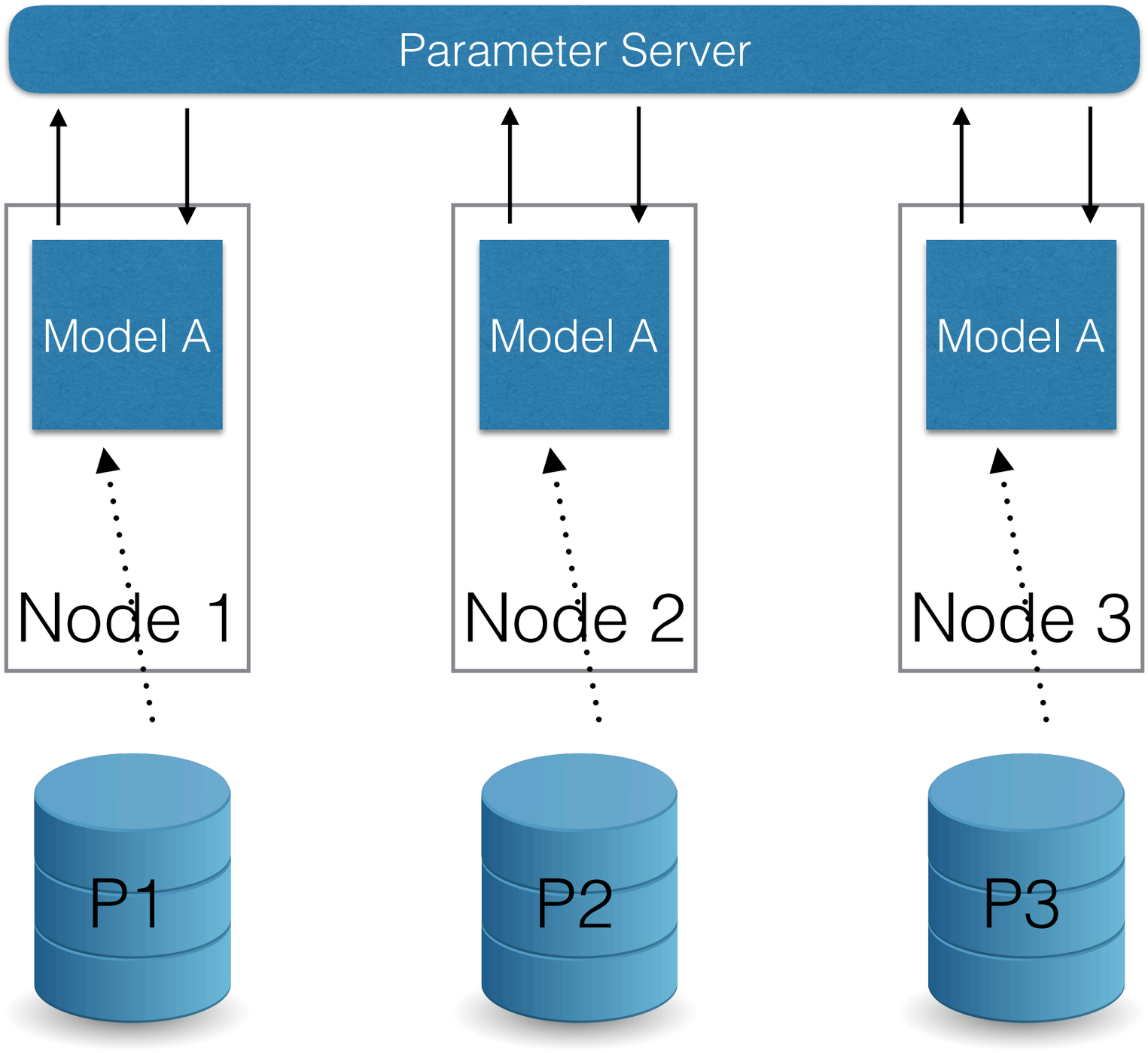
data parallelism

aka. “Jeff Dean style” parameter averaging



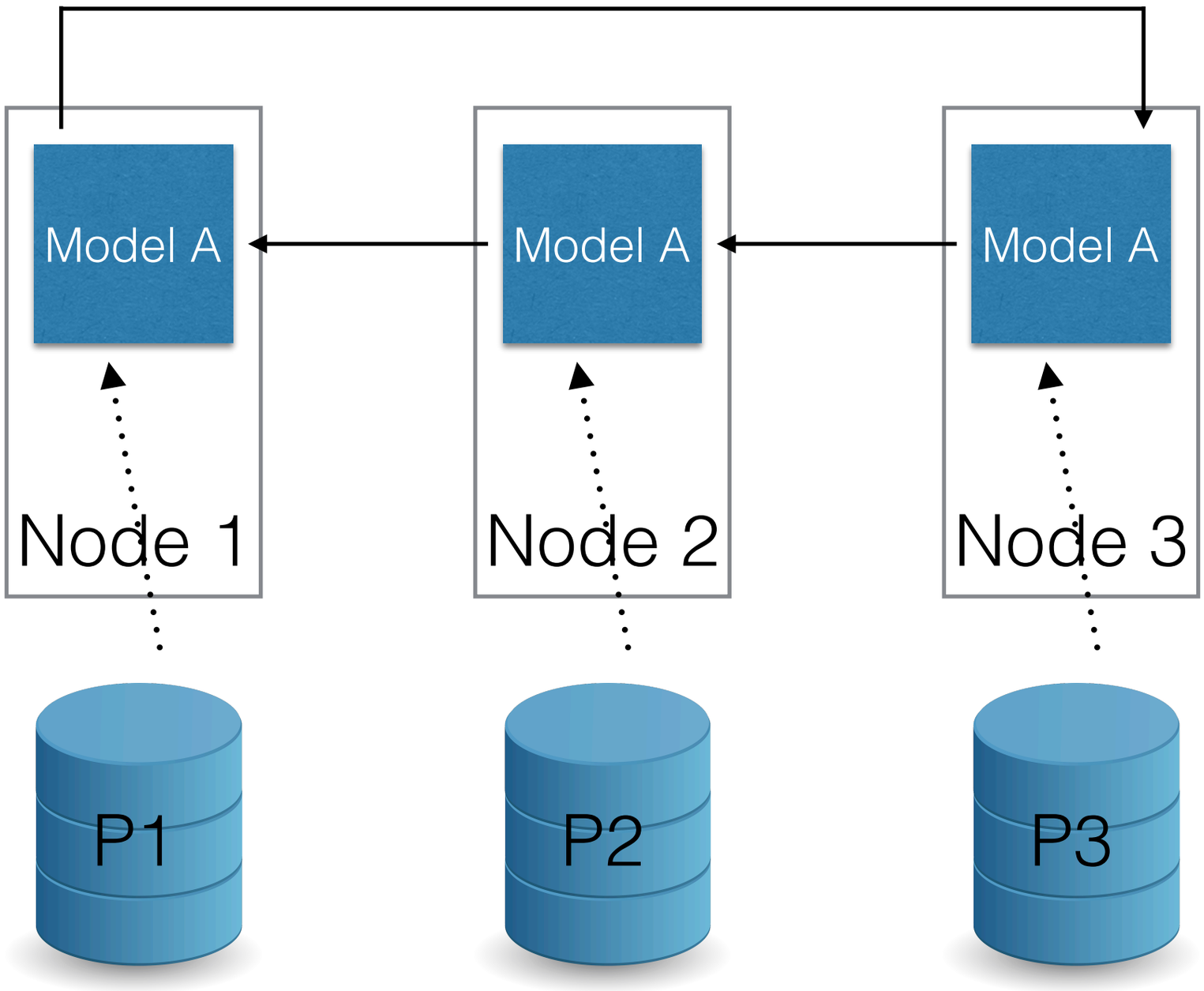
data parallelism

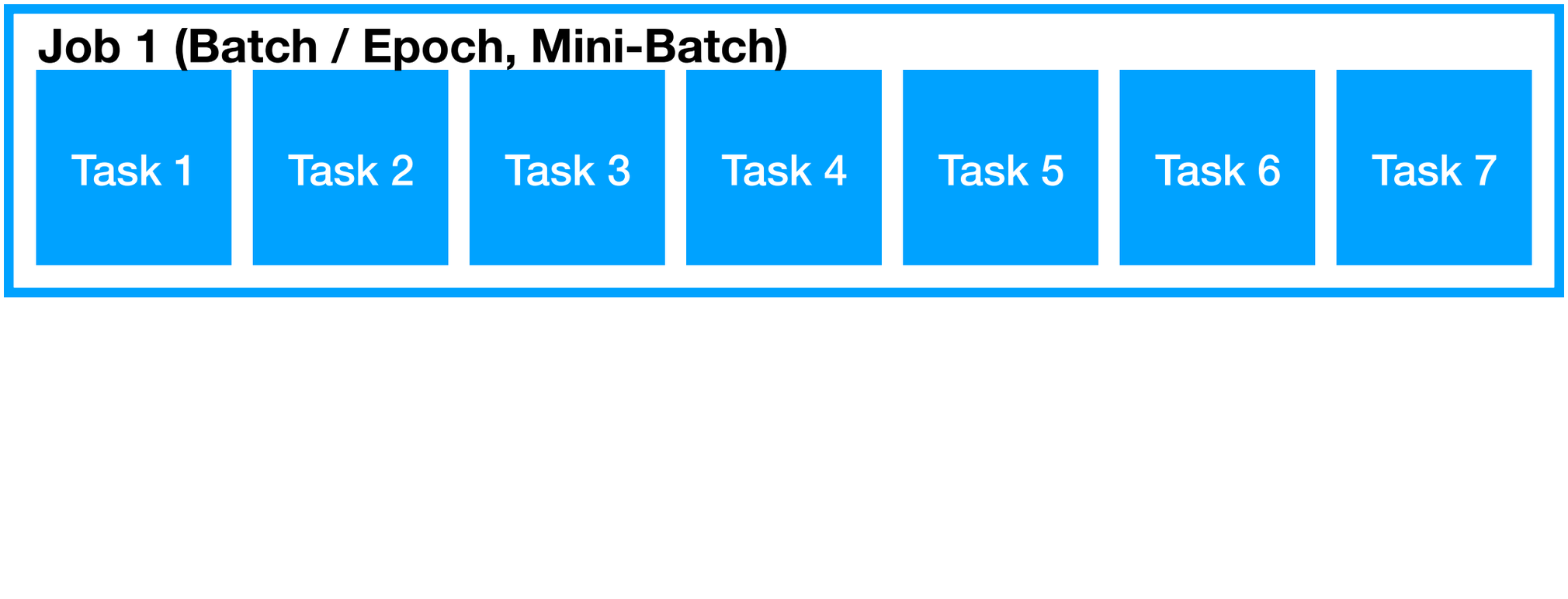
aka. “Jeff Dean style” parameter averaging

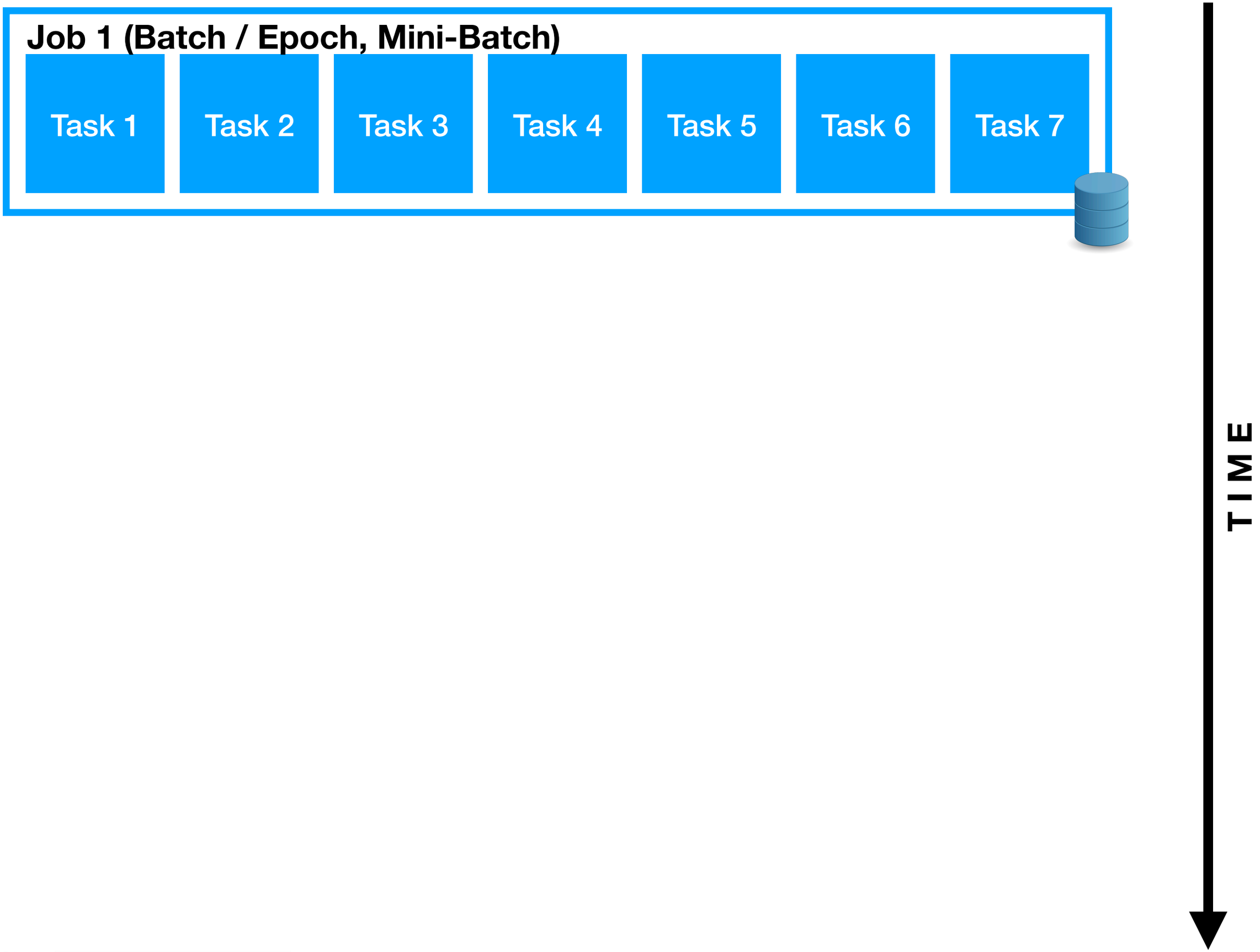


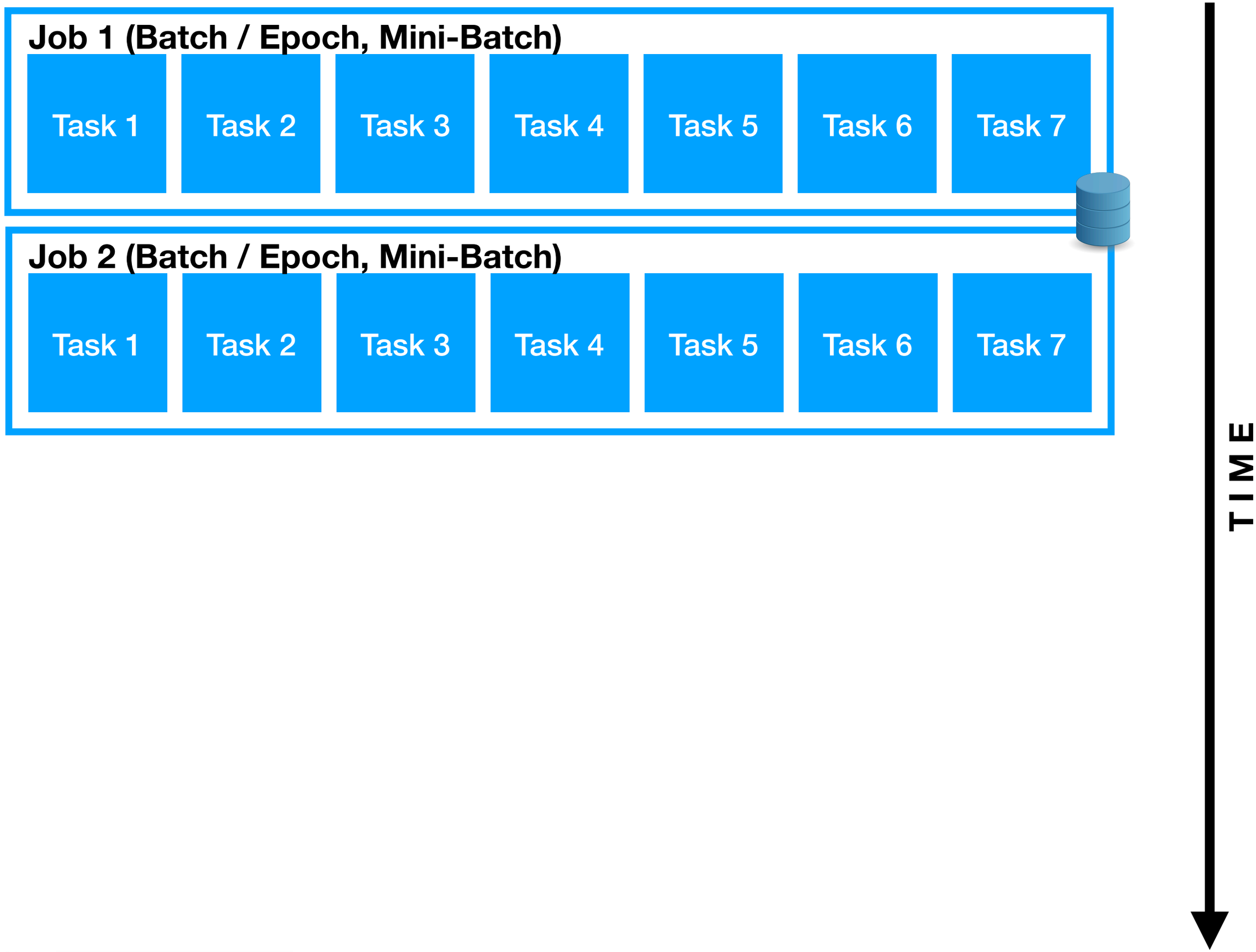
data parallelism

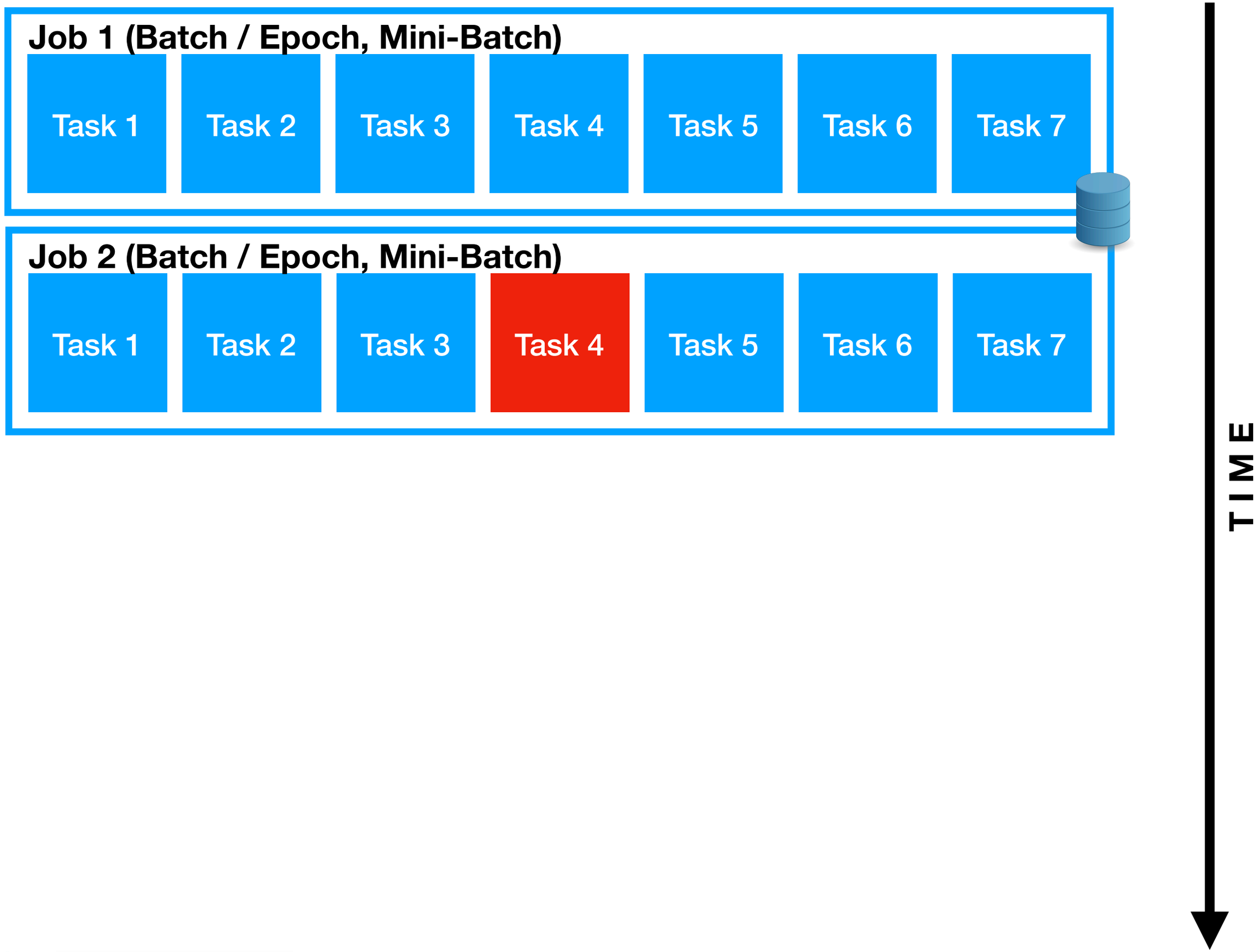
aka. “ReduceAll” parameter averaging

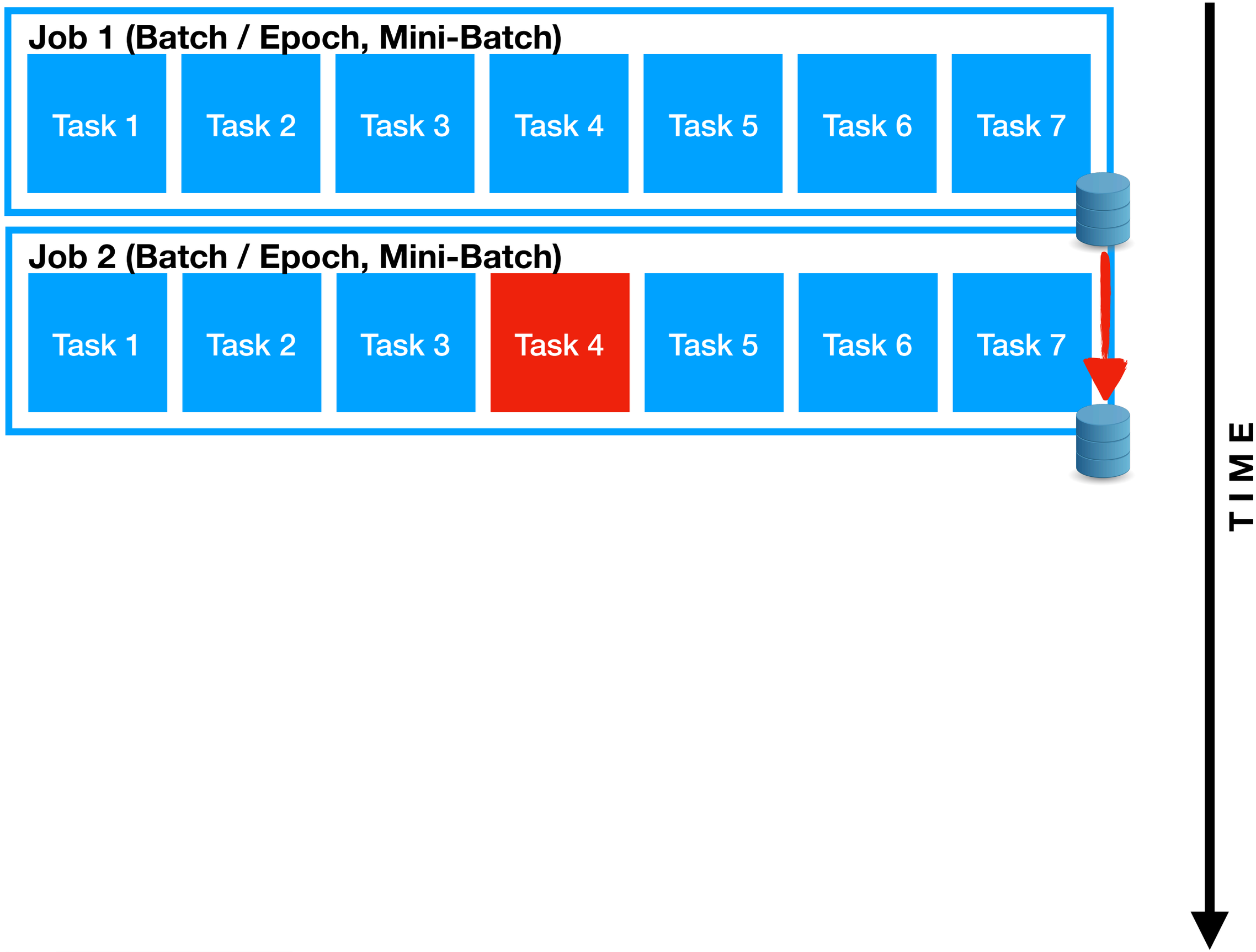












Job 1 (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

Job 2 (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

Job 2a (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

TIME

Job 1 (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

Job 2 (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

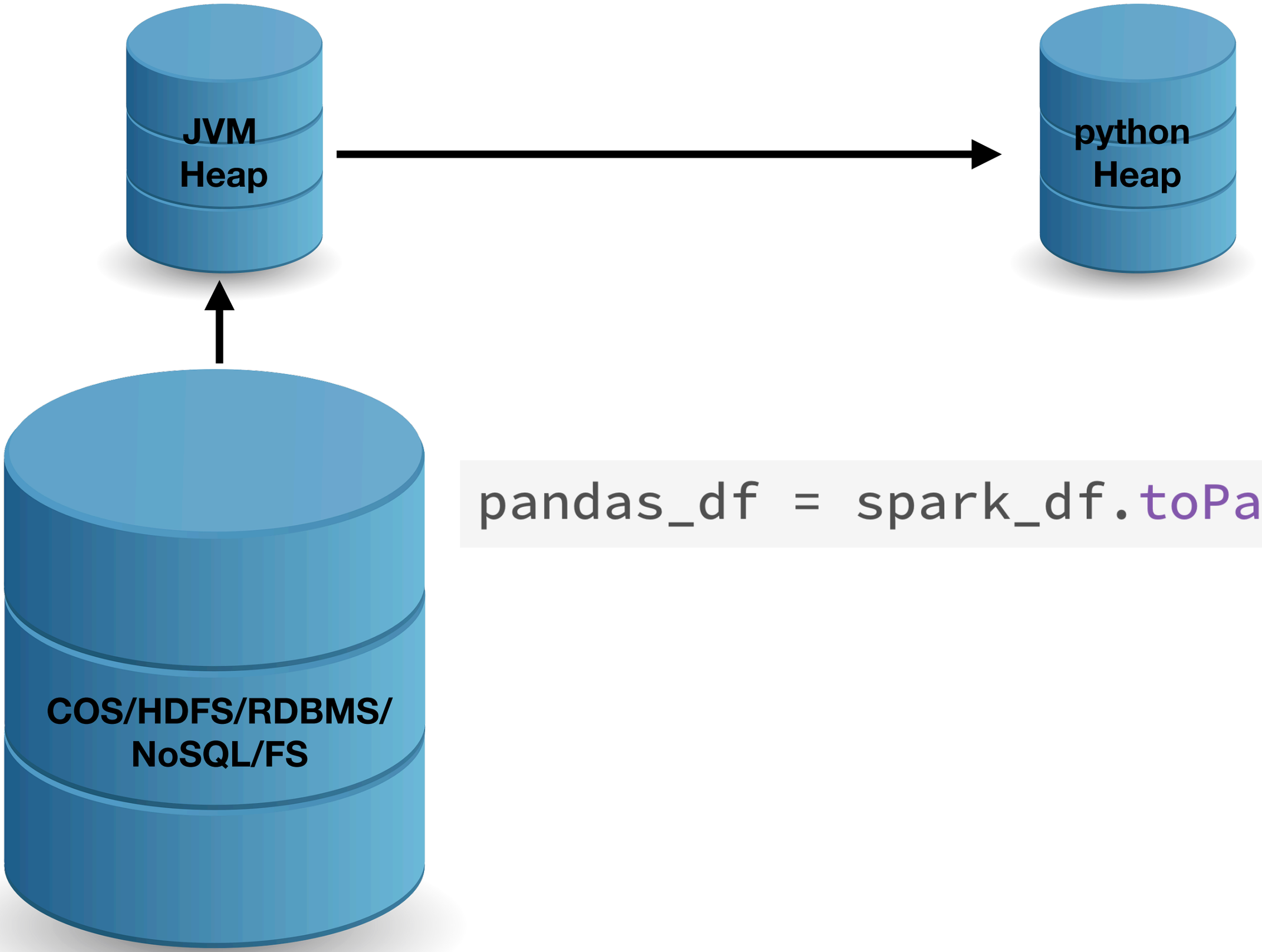
Job 2a (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

Job 3 (Batch / Epoch, Mini-Batch)

Task 1 | Task 2 | Task 3 | Task 4 | Task 5 | Task 6 | Task 7

TIME




```
pandas_df = spark_df.toPandas()
```

Decrease memory consumption with toPandas() collection using Arrow

Details

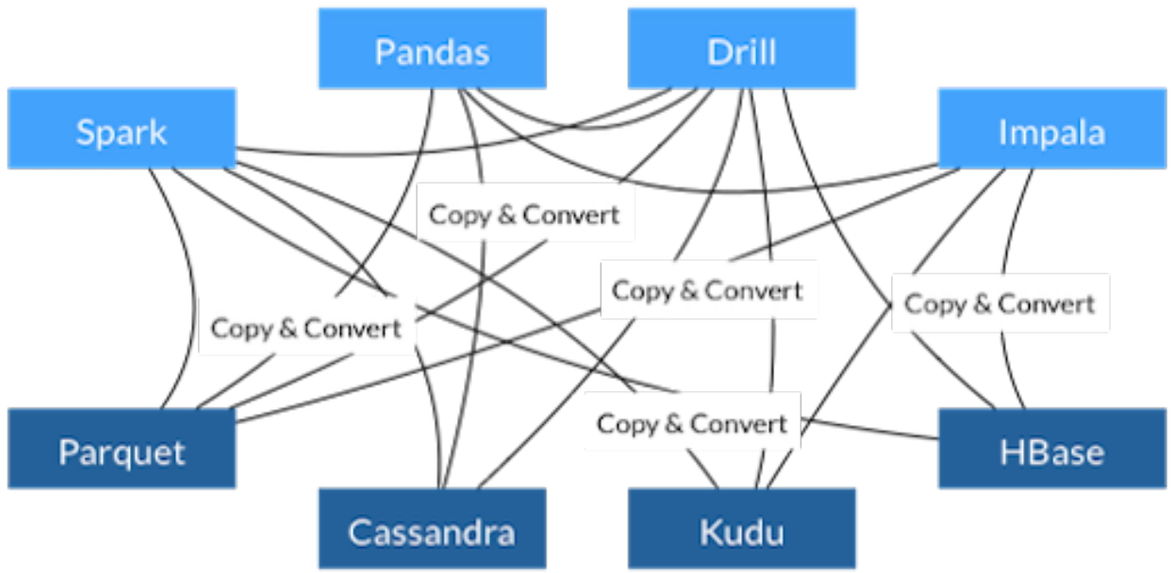
Type:	Sub-task	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.3.0	Fix Version/s:	2.4.0
Component/s:	PySpark, SQL		
Labels:	None		

Description

Currently with Arrow enabled, calling `toPandas()` results in a collection of all partitions in the JVM in the form of batches of Arrow file format. Once collected in the JVM, they are served to the Python driver process.

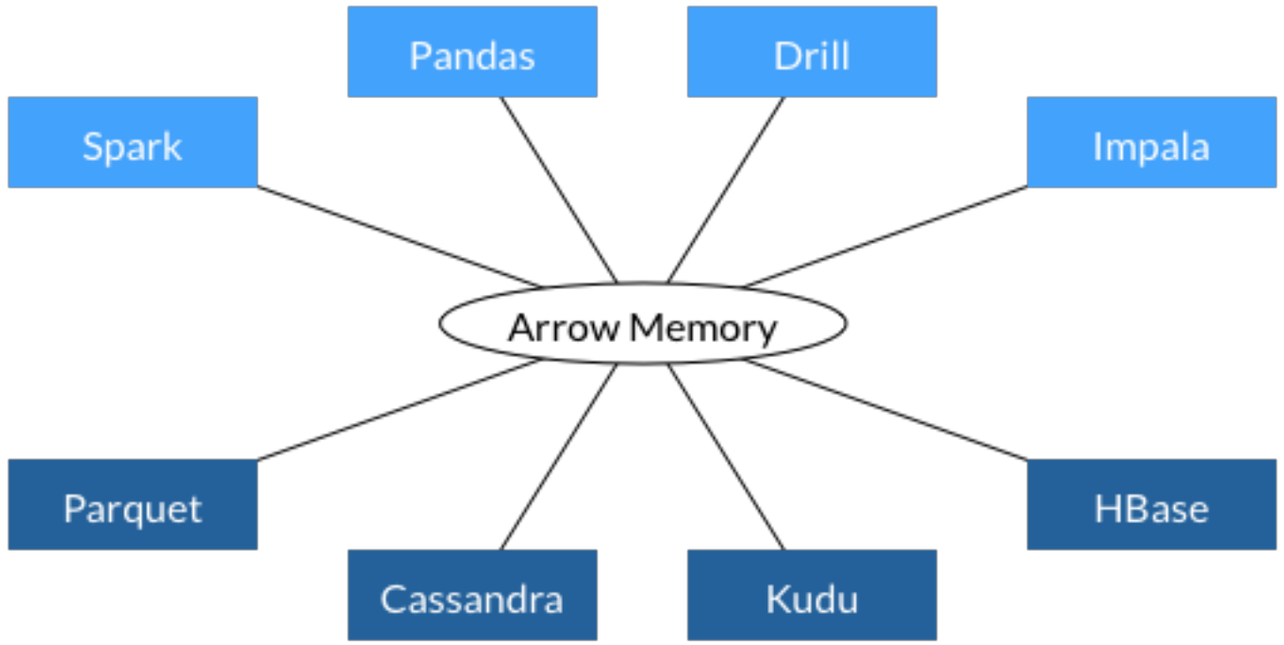
I believe using the Arrow stream format can help to optimize this and reduce memory consumption in the JVM by only loading one record batch at a time before sending it to Python. This might also reduce the latency between making the initial call in Python and receiving the first batch of records.

Apache Arrow



romeo kienzler

Apache Arrow



Apache Arrow

	session_id	timestamp	source_ip
Row 1	1331246660	3/8/2012 2:44PM	99.155.155.225
Row 2	1331246351	3/8/2012 2:38PM	65.87.165.114
Row 3	1331244570	3/8/2012 2:09PM	71.10.106.181
Row 4	1331261196	3/8/2012 6:46PM	76.102.156.138

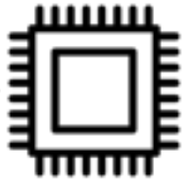
Traditional Memory Buffer

Row 1	1331246660
	3/8/2012 2:44PM
	99.155.155.225
Row 2	1331246351
	3/8/2012 2:38PM
	65.87.165.114
Row 3	1331244570
	3/8/2012 2:09PM
	71.10.106.181
Row 4	1331261196
	3/8/2012 6:46PM
	76.102.156.138

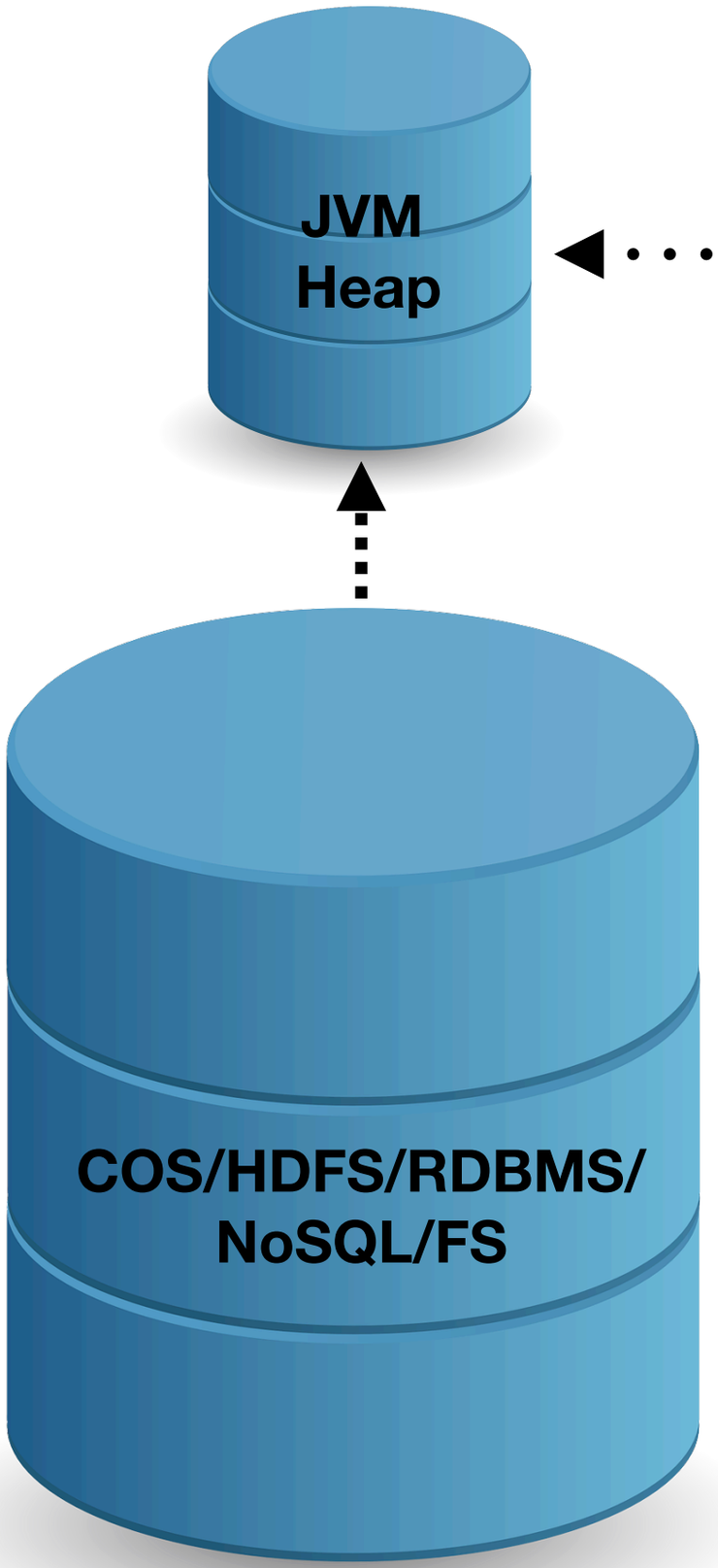
Arrow Memory Buffer

session_id	1331246660
	1331246351
	1331244570
	1331261196
timestamp	3/8/2012 2:44PM
	3/8/2012 2:38PM
	3/8/2012 2:09PM
	3/8/2012 6:46PM
source_ip	99.155.155.225
	65.87.165.114
	71.10.106.181
	76.102.156.138

```
SELECT * FROM clickstream
WHERE session_id = 1331246351
```



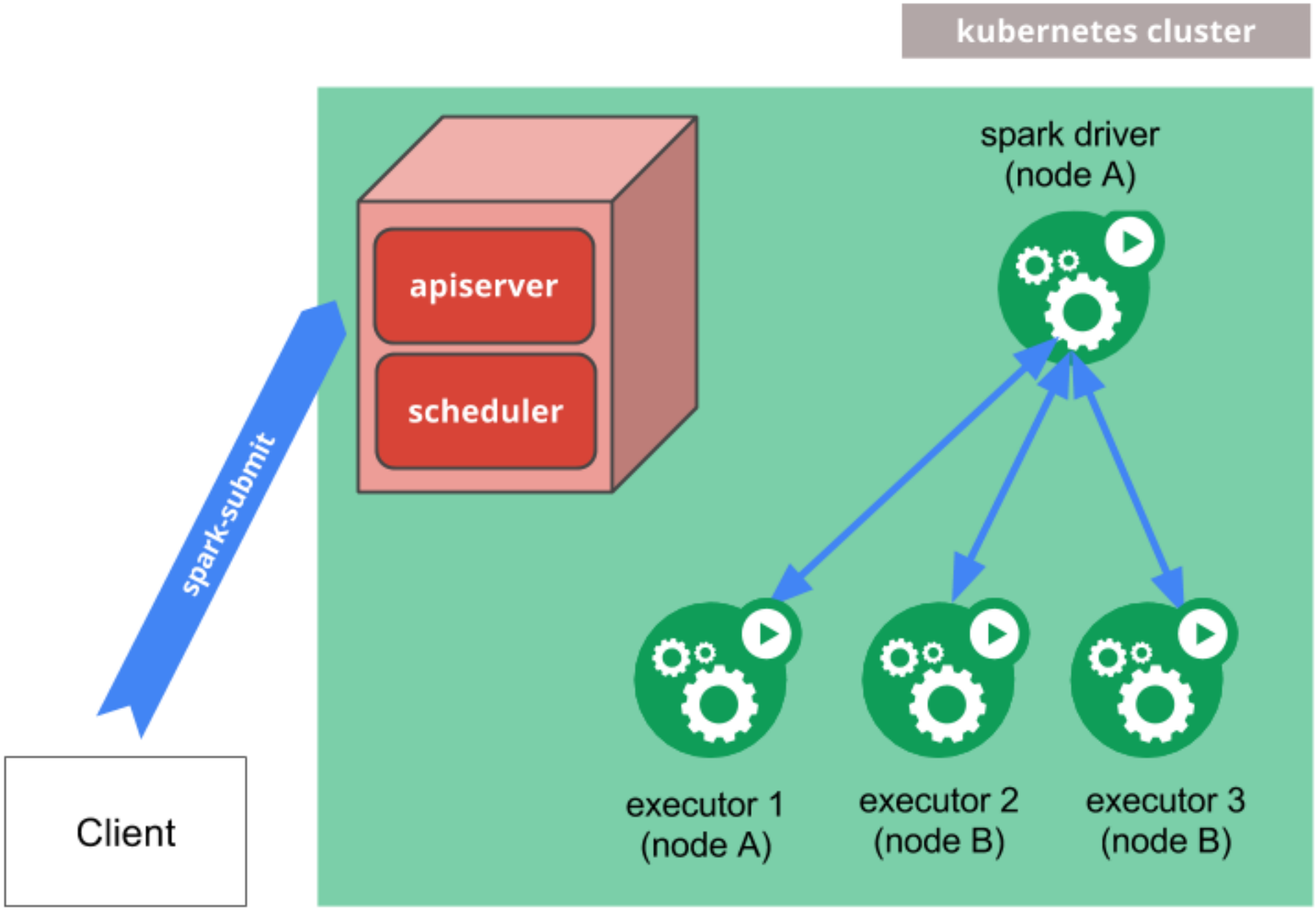
Intel CPU




```
pandas_df = spark_df.toPandas()
```

Spark on Kubernetes

[SPARK-18278] / [K8S-34377] since Spark 2.3

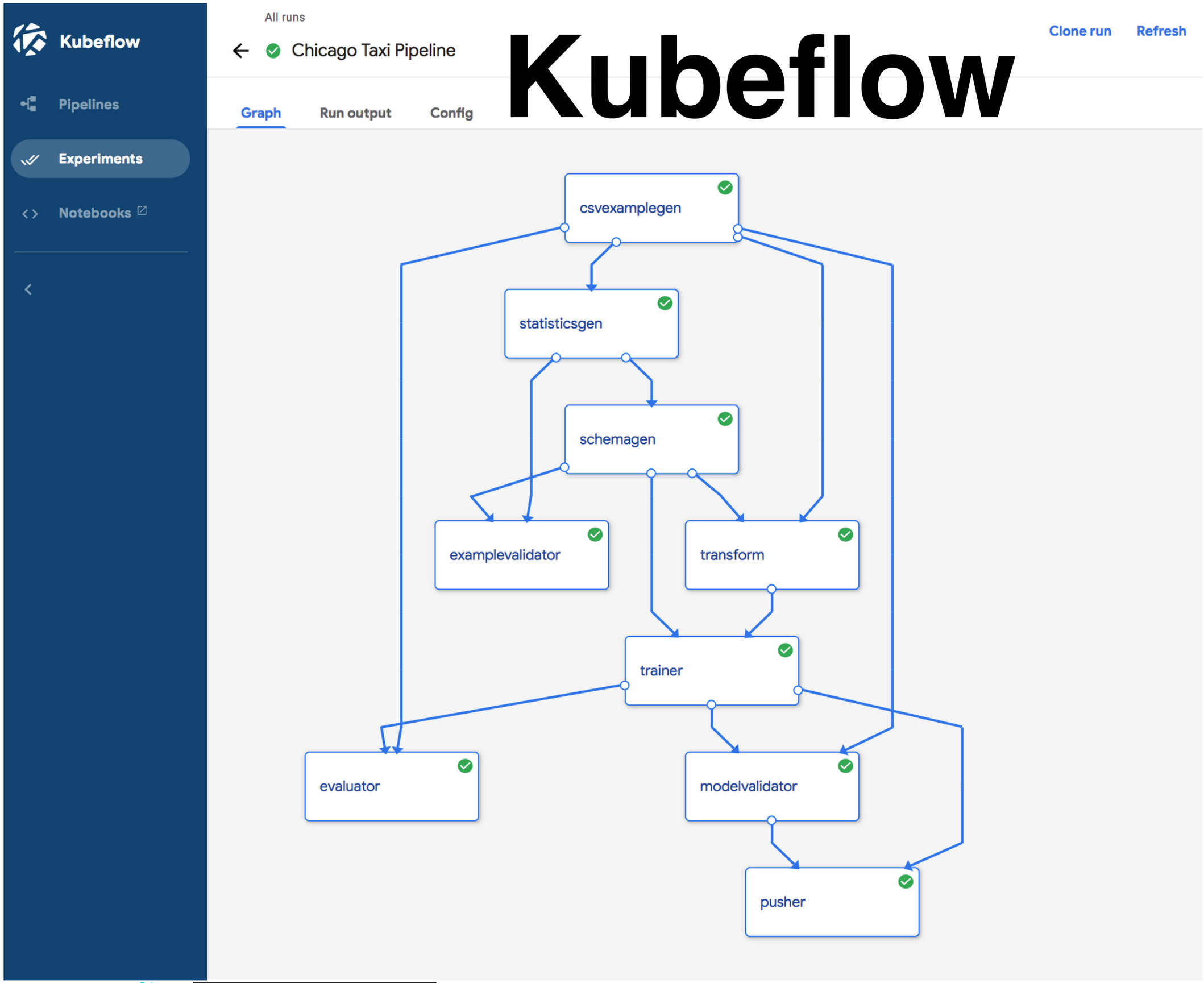


Spark on Kubernetes

[SPARK-18278] / [K8S-34377] since Spark 2.3

```
$ bin/spark-submit \
    --master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
    --deploy-mode cluster \
    --name spark-pi \
    --class org.apache.spark.examples.SparkPi \
    --conf spark.executor.instances=5 \
    --conf spark.kubernetes.container.image=<spark-image> \
    local:///path/to/examples.jar
```

Kubeflow



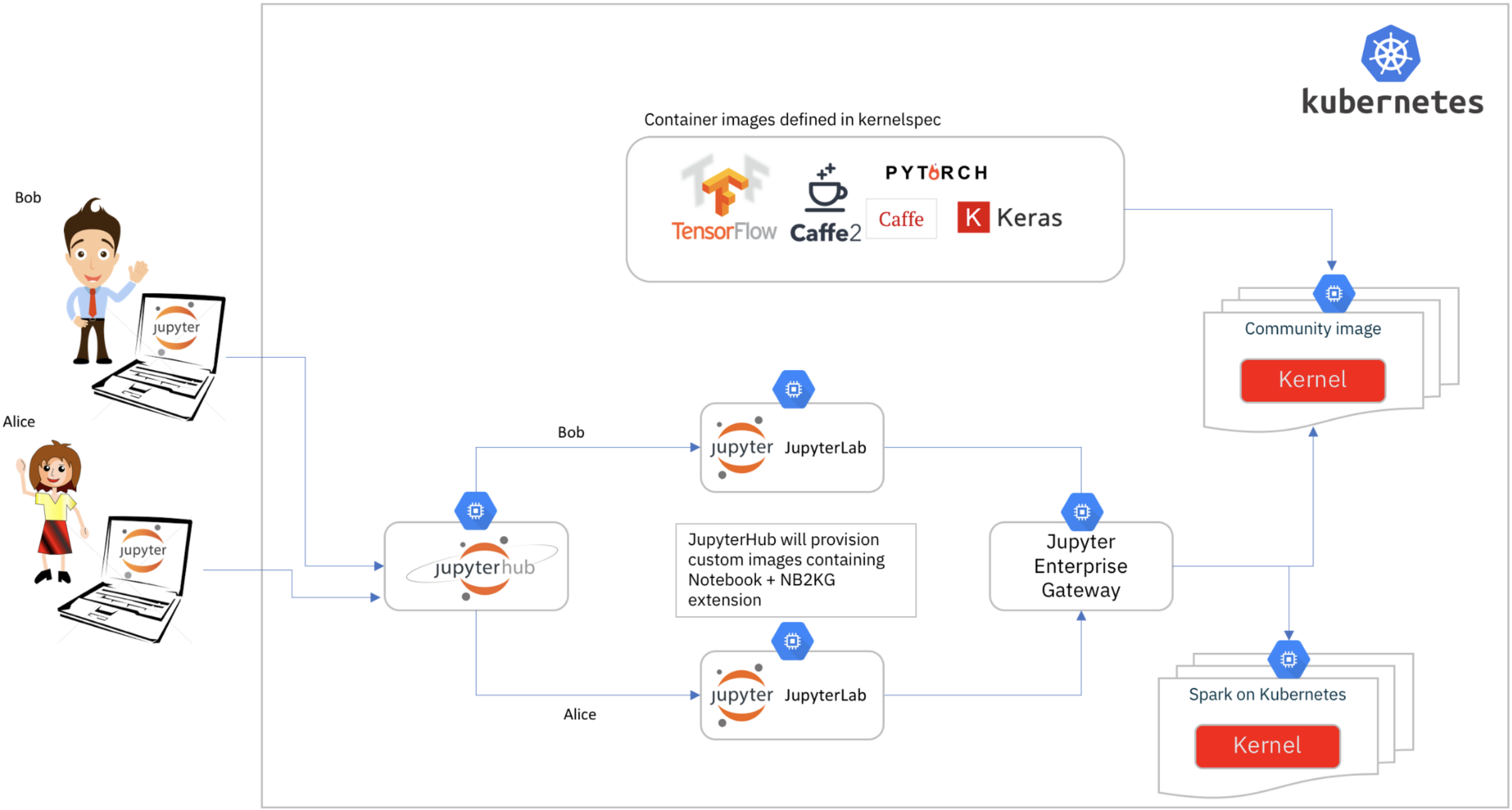
Spark on Kubernetes

Interactive??

romeo kienzler

Spark on Kubernetes

jupyther on Kubernetes before Apache Spark 2.3 using jupyter enterprise gateway



IBM **Watson Studio** Upgrade

My Projects / coursera_deg / assignment4.1_spark2.3_python3.5

File Edit View Insert Cell Kernel Help

Run Format Markdown

In [2]:
```
df = spark.read.parquet('washing.parquet')
df.createOrReplaceTempView('washing')
df.show()
```

```
+--------------------+--------------------+-----+--------+----------+---------+--------+-----+-----------+-------------+-------+
|                 _id|                _rev|count|flowrate|fluidlevel|frequency|hardness|speed|temperature|           ts|voltage|
+--------------------+--------------------+-----+--------+----------+---------+--------+-----+-----------+-------------+-------+
|0d86485d0f88d1f9d...|1-57940679fb8a713...|    4|      11|acceptable|     null|      77| null|        100|1547808723923|   null|
|0d86485d0f88d1f9d...|1-15ff3a0b304d789...|    2|    null|      null|     null|    null| 1046|       null|1547808729917|   null|
|0d86485d0f88d1f9d...|1-97c2742b68c7b07...|    4|    null|      null|       71|    null| null|       null|1547808731918|    236|
|0d86485d0f88d1f9d...|1-eefb903dbe45746...|   19|      11|acceptable|     null|      75| null|         86|1547808738999|   null|
|0d86485d0f88d1f9d...|1-5f68b4c72813c25...|    7|    null|      null|       75|    null| null|       null|1547808740927|    235|
|0d86485d0f88d1f9d...|1-cd4b6c57ddbe77e...|    5|    null|      null|     null|    null| 1014|       null|1547808744923|   null|
|0d86485d0f88d1f9d...|1-a35b25b5bf43aaf...|   32|      11|acceptable|     null|      73| null|         84|1547808752028|   null|
|0d86485d0f88d1f9d...|1-b717f7289a8476d...|   48|      11|acceptable|     null|      79| null|         84|1547808768065|   null|
|0d86485d0f88d1f9d...|1-c2f1f8fcf178b2f...|   18|    null|      null|       73|    null| null|       null|1547808773944|    228|
|0d86485d0f88d1f9d...|1-15033dd9eebb4a8...|   59|      11|acceptable|     null|      72| null|         96|1547808779093|   null|
|0d86485d0f88d1f9d...|1-753dae825f9a6c2...|   62|      11|acceptable|     null|      73| null|         88|1547808782113|   null|
|0d86485d0f88d1f9d...|1-b168089f44f03f0...|   13|    null|      null|     null|    null| 1097|       null|1547808784940|   null|
|0d86485d0f88d1f9d...|1-403b687c6be0dea...|   23|    null|      null|       80|    null| null|       null|1547808788955|    236|
|0d86485d0f88d1f9d...|1-195551e0455a24b...|   72|      11|acceptable|     null|      77| null|         87|1547808792134|   null|
|0d86485d0f88d1f9d...|1-060a39fc6c2ddee...|   26|    null|      null|       62|    null| null|       null|1547808797959|    233|
|0d86485d0f88d1f9d...|1-2234514bffee465...|   27|    null|      null|       61|    null| null|       null|1547808800960|    226|
|0d86485d0f88d1f9d...|1-4265898bb401db0...|   82|      11|acceptable|     null|      79| null|         96|1547808802154|   null|
|0d86485d0f88d1f9d...|1-2fbf7ca9a0425a0...|   94|      11|acceptable|     null|      73| null|         90|1547808814186|   null|
|0d86485d0f88d1f9d...|1-203c0ee6d7fbd21...|   97|      11|acceptable|     null|      77| null|         88|1547808817190|   null|
|0d86485d0f88d1f9d...|1-47e1965db94fcab...|  104|      11|acceptable|     null|      75| null|         80|1547808824198|   null|
+--------------------+--------------------+-----+--------+----------+---------+--------+-----+-----------+-------------+-------+
only showing top 20 rows
```

Spark on Kubernetes

Spark / SPARK-23146

Support client mode for Kubernetes cluster backend

Comment | Agile Board | More

Details

Type:	New Feature	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.3.0	Fix Version/s:	2.4.0
Component/s:	Kubernetes		
Labels:	None		
Target Version/s:	2.4.0		

Description

This issue tracks client mode support within Spark when running in the Kubernetes cluster backend.

Spark on Kubernetes

[SPARK-23146] since Spark 2.4

Lets you decide on Apache Spark Driver placement
=> interactive shells
=> jupyter notebooks

romeo kienzler

Eager Mode

Spark / SPARK-24215

Implement eager evaluation for DataFrame APIs

Comment | Agile Board | More

Details

Type:	Improvement	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	2.3.0	Fix Version/s:	2.4.0
Component/s:	PySpark, Spark Core, SQL		
Labels:	None		
Target Version/s:	2.4.0		

Description

To help people that are new to Spark get feedback more easily, we should implement the repr methods for Jupyter python kernels. That way, when users run pyspark in jupyter console or notebooks, they get good feedback about the queries they've defined.

This should include an option for eager evaluation, (maybe spark.jupyter.eager-eval?). When set, the formatting methods would run dataframes and produce output like `show`. This is a good balance between not hiding Spark's action behavior and getting feedback to users that don't know to call actions.

Here's the dev list thread for context: http://apache-spark-developers-list.1001551.n3.nabble.com/eager-execution-and-debuggability-td23928.html

Eager Mode

(on DataFrame) - before

jupyter Untitled21 Last Checkpoint: 10 分钟前 Autosave Failed!

Logout

File Edit View Insert Cell Kernel Help

Not Connected error Trusted Python 2

Run Code

In [1]:
```
spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .getOrCreate()
```

In [2]:
```
df = spark.read.json("/Users/XuanYuan/Source/spark_build/examples/src/main/resources/people.json")
```

In [3]:
```
df.select("name")
```

Out[3]: DataFrame[name: string]

In [4]:
```
df.select(df['name'], df['age'] + 1)
```

Out[4]: DataFrame[name: string, (age + 1): bigint]

In [5]:
```
df.filter(df['age'] > 21)
```

Out[5]: DataFrame[age: bigint, name: string]

In [6]:
```
df.groupBy("age").count()
```

Out[6]: DataFrame[age: bigint, count: bigint]

Eager Mode

(on DataFrame) - before

jupyter Untitled20 Last Checkpoint: 13 分钟前 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help

Not Connected | Trusted | Python 2

Run | Code

In [1]:
```
spark = SparkSession \
        .builder \
        .appName("Python Spark SQL basic example") \
        .config("spark.jupyter.eagerEval.enabled", "true") \
        .getOrCreate()
```

In [2]:
```
df = spark.read.json("/Users/XuanYuan/Source/spark_build/examples/src/main/resources/people.json")
```

In [3]:
```
df.select("name")
```

Out[3]:

name
Michael
Andy
Justin

In [4]:
```
df.select(df['name'], df['age'] + 1)
```

Out[4]:

name	(age + 1)
Michael	null
Andy	31
Justin	20

In [5]:
```
df.filter(df['age'] > 21)
```

Out[5]:

age	name
30	Andy

Outlook Spark v3.0

Spark / SPARK-24579

SPIP: Standardize Optimized Data Exchange between Spark and DL/AI frameworks

Comment | Agile Board | More

Details

Type:	Epic	Status:	OPEN
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	3.0.0	Fix Version/s:	None
Component/s:	ML, PySpark, SQL		
Labels:	Hydrogen		
Epic Name:	Project Hydrogen: Data Exchange		

Description

(see attached SPIP pdf for more details)

At the crossroads of big data and AI, we see both the success of Apache Spark as a unified analytics engine and the rise of AI frameworks like TensorFlow and Apache MXNet (incubating). Both big data and AI are indispensable components to drive business innovation and there have been multiple attempts from both communities to bring them together.

We saw efforts from AI community to implement data solutions for AI frameworks like tf.data and tf.Transform. However, with 50+ data sources and built-in SQL, DataFrames, and Streaming features, Spark remains the community choice for big data. This is why we saw many efforts to integrate DL/AI frameworks with Spark to leverage its power, for example, TFRecords data source for Spark, TensorFlowOnSpark, TensorFrames, etc. As part of Project Hydrogen, this SPIP takes a different angle at Spark + AI unification.

None of the integrations are possible without exchanging data between Spark and external DL/AI frameworks. And the performance matters. However, there doesn't exist a standard way to exchange data and hence implementation and performance optimization fall into pieces. For example, TensorFlowOnSpark uses Hadoop InputFormat/OutputFormat for TensorFlow's TFRecords to load and save data and pass the RDD records to TensorFlow in Python. And TensorFrames converts Spark DataFrames Rows to/from TensorFlow Tensors using TensorFlow's Java API. How can we reduce the complexity?

Outlook Spark v3.0

Spark / SPARK-24615

SPIP: Accelerator-aware task scheduling for Spark

Comment | Agile Board | More

Details

Type:	Epic	Status:	OPEN
Priority:	Major	Resolution:	Unresolved
Affects Version/s:	2.4.0	Fix Version/s:	None
Component/s:	Spark Core		
Labels:	Hydrogen SPIP		
Epic Name:	GPU-aware Scheduling		

Description

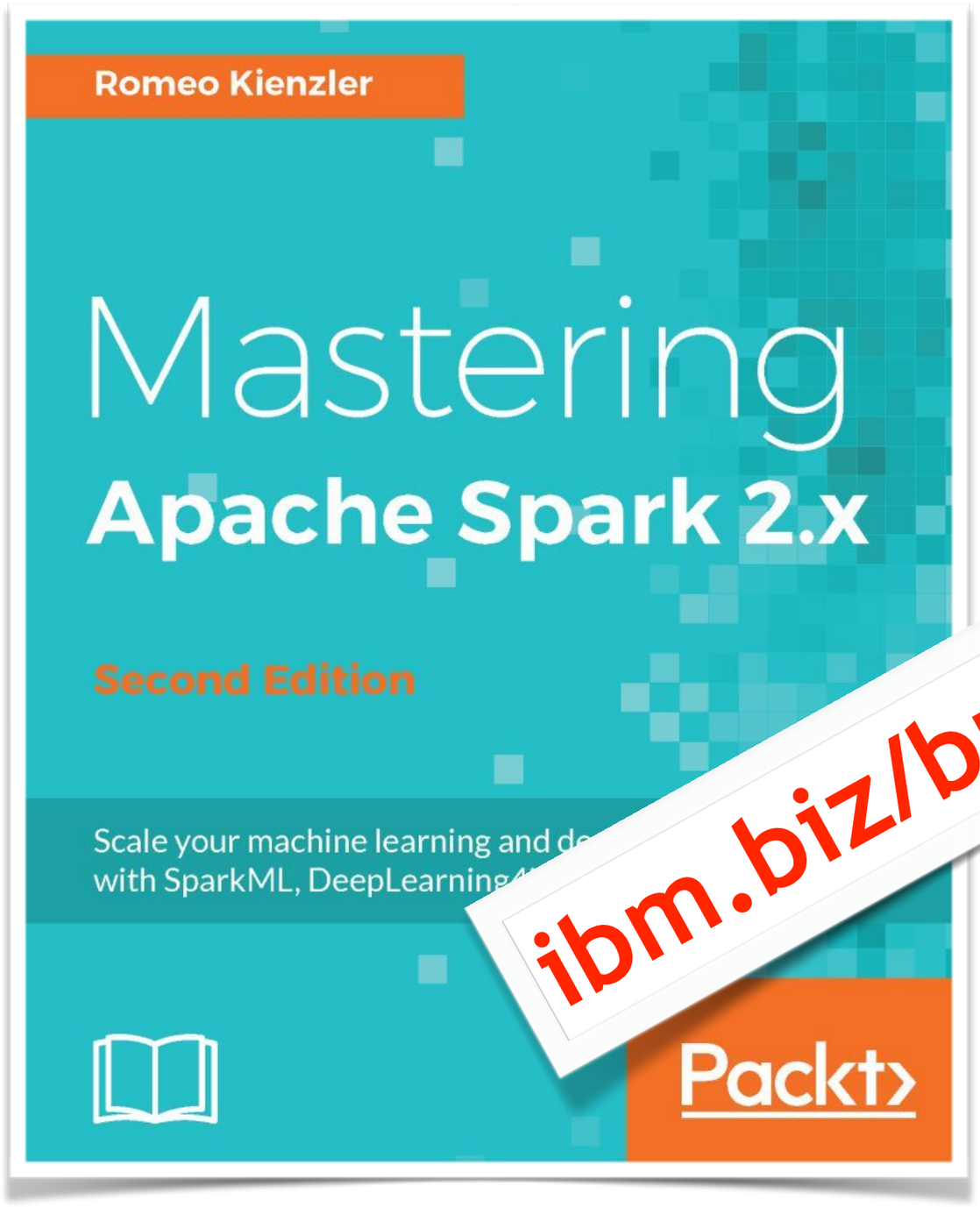
(The JIRA received a major update on 2019/02/28. Some comments were based on an earlier version. Please ignore them. New comments start at comment-16778026.)

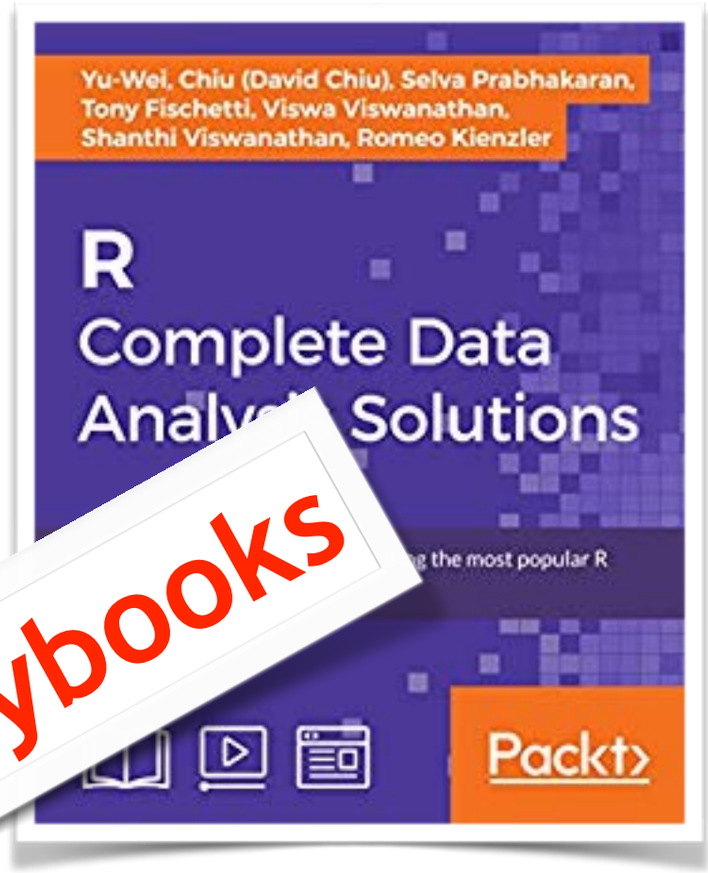
Background and Motivation

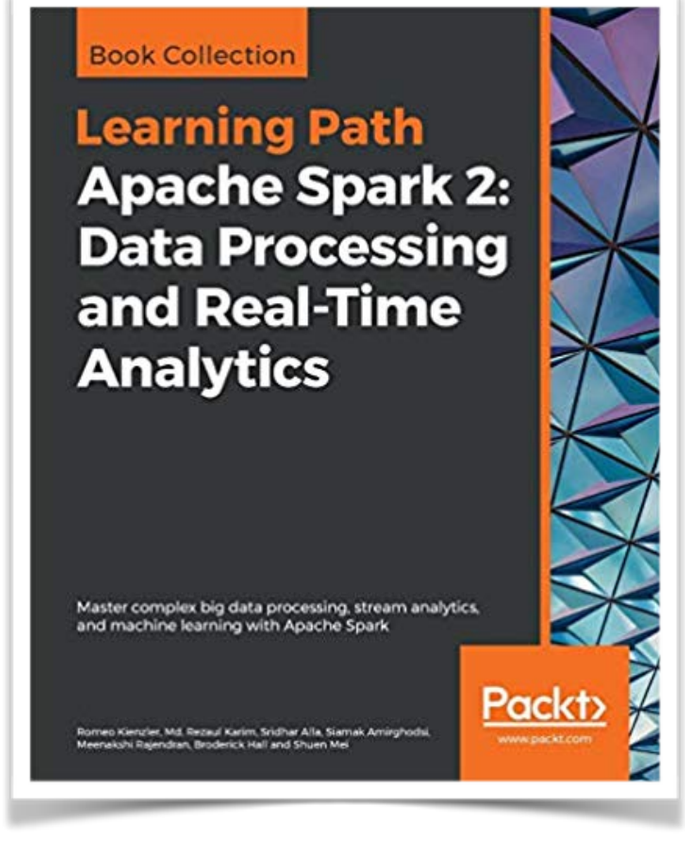
GPUs and other accelerators have been widely used for accelerating special workloads, e.g., deep learning and signal processing. While users from the AI community use GPUs heavily, they often need Apache Spark to load and process large datasets and to handle complex data scenarios like streaming. YARN and Kubernetes already support GPUs in their recent releases. Although Spark supports those two cluster managers, Spark itself is not aware of GPUs exposed by them and hence Spark cannot properly request GPUs and schedule them for users. This leaves a critical gap to unify big data and AI workloads and make life simpler for end users.

To make Spark be aware of GPUs, we shall make two major changes at high level:

- At cluster manager level, we update or upgrade cluster managers to include GPU support. Then we expose user interfaces for Spark to request GPUs from them.
- Within Spark, we update its scheduler to understand available GPUs allocated to executors, user task requests, and assign GPUs to tasks properly.







IBM Advanced Data Science Specialization Certificate on C sera

Fundamentals of Scalable Data
www.coursera.org/le

Advanced g and Signal Processing
www.coursera ed-machine-learning-signal ssing

Applied AI wit
www.co

Ac Data Science Capstone Project
https://www.c rsera.org/learn/advanced-data-science-capstone