Sparklens: Understanding the scalability limits of Spark applications

Rohit Karlupia, Qubole

@qubole #Sparklens #SAISDev17

Agenda

PERFORMANCE TUNING PITFALLS

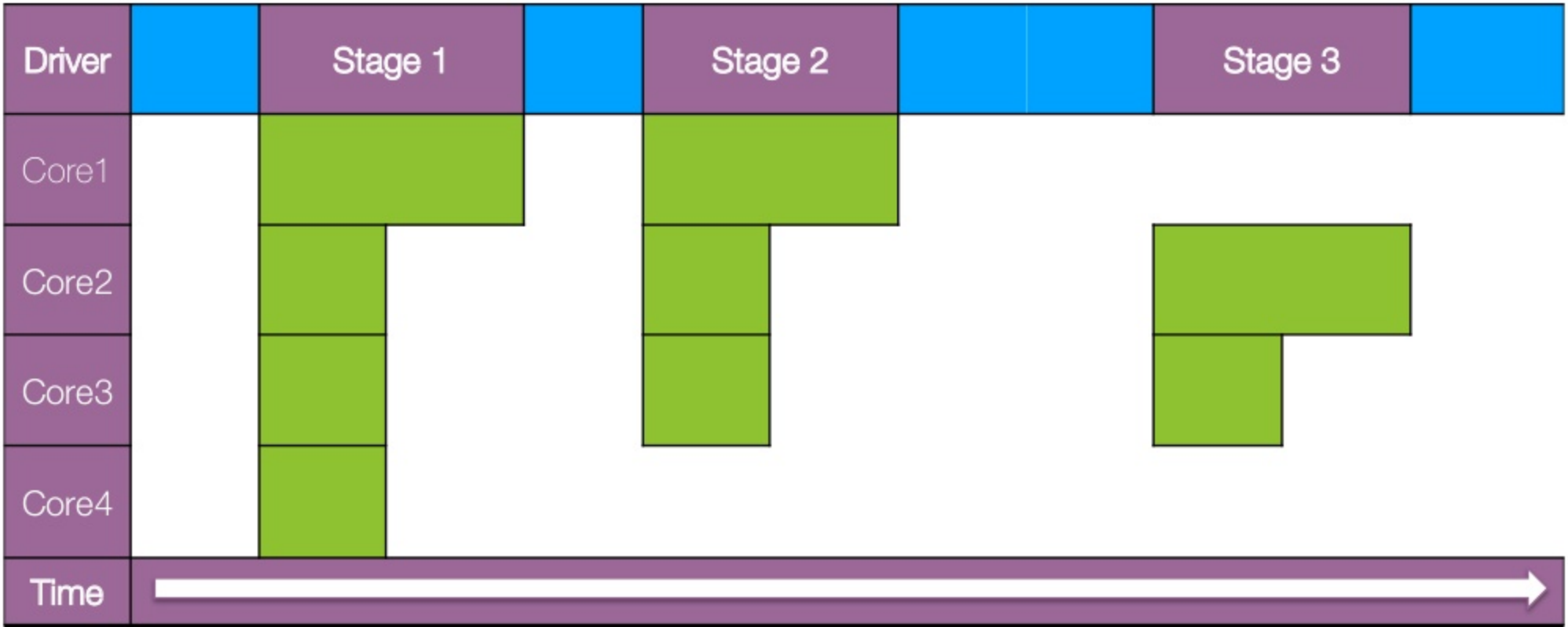
THEORY BEHIND SPARKLENS

QUBOLE SPARKLENS TUNING EXAMPLE

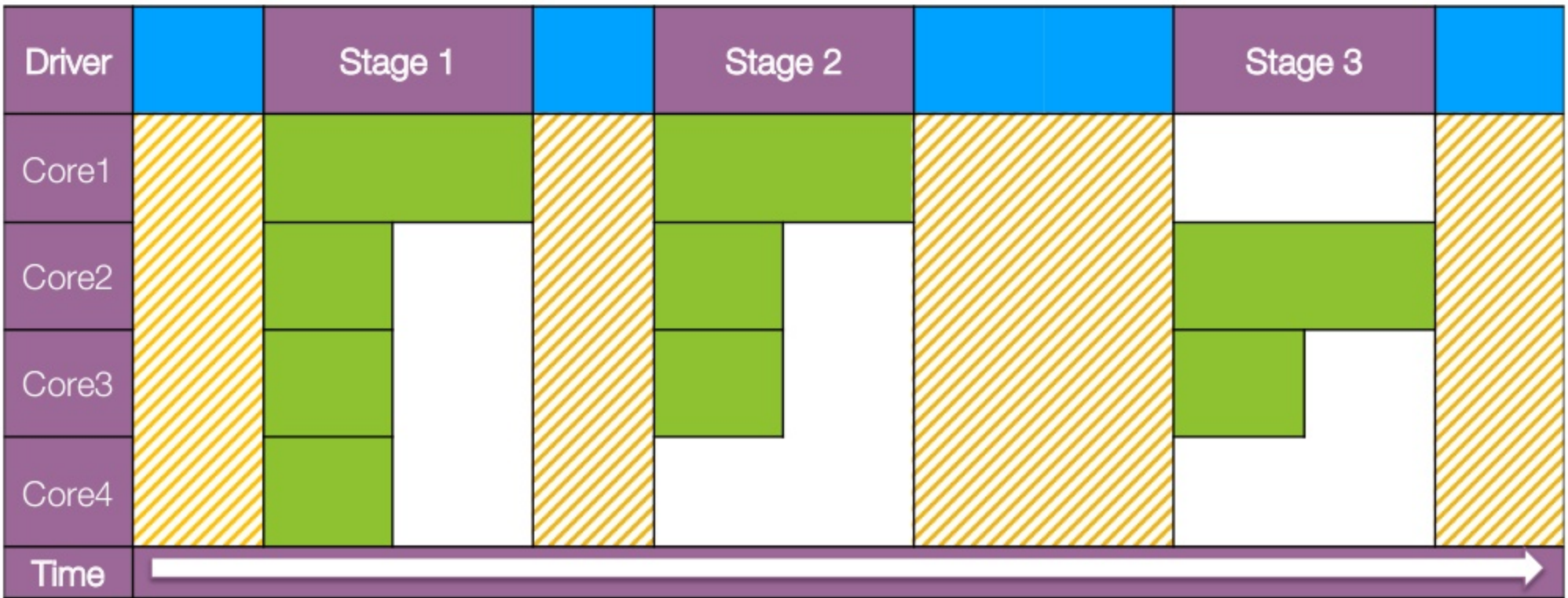
Spark Tuning: Usual Suspects

- Resource Utilization
- Experiments
- Profiling

Minimize Doing Nothing



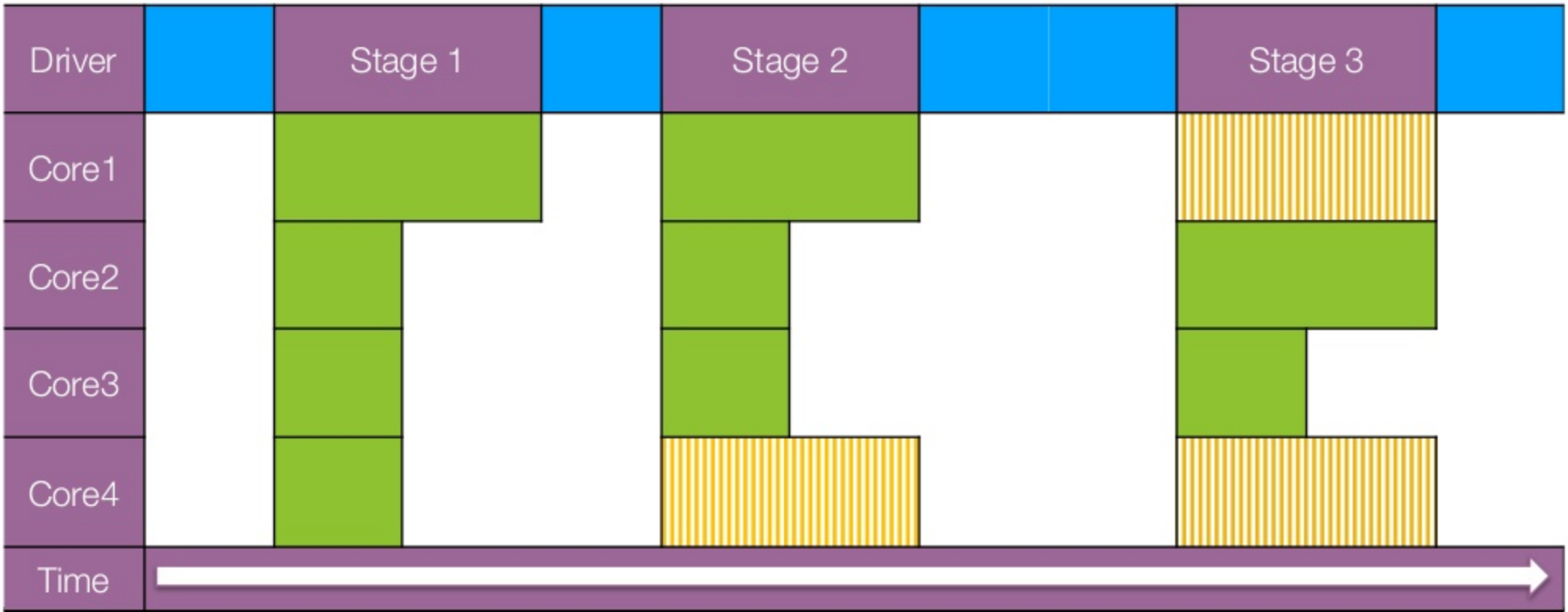
Driver Side Computations



Driver does...

- File listing & split computation
- Loading of hive tables
- FOC
- Collect
- df.toPandas()

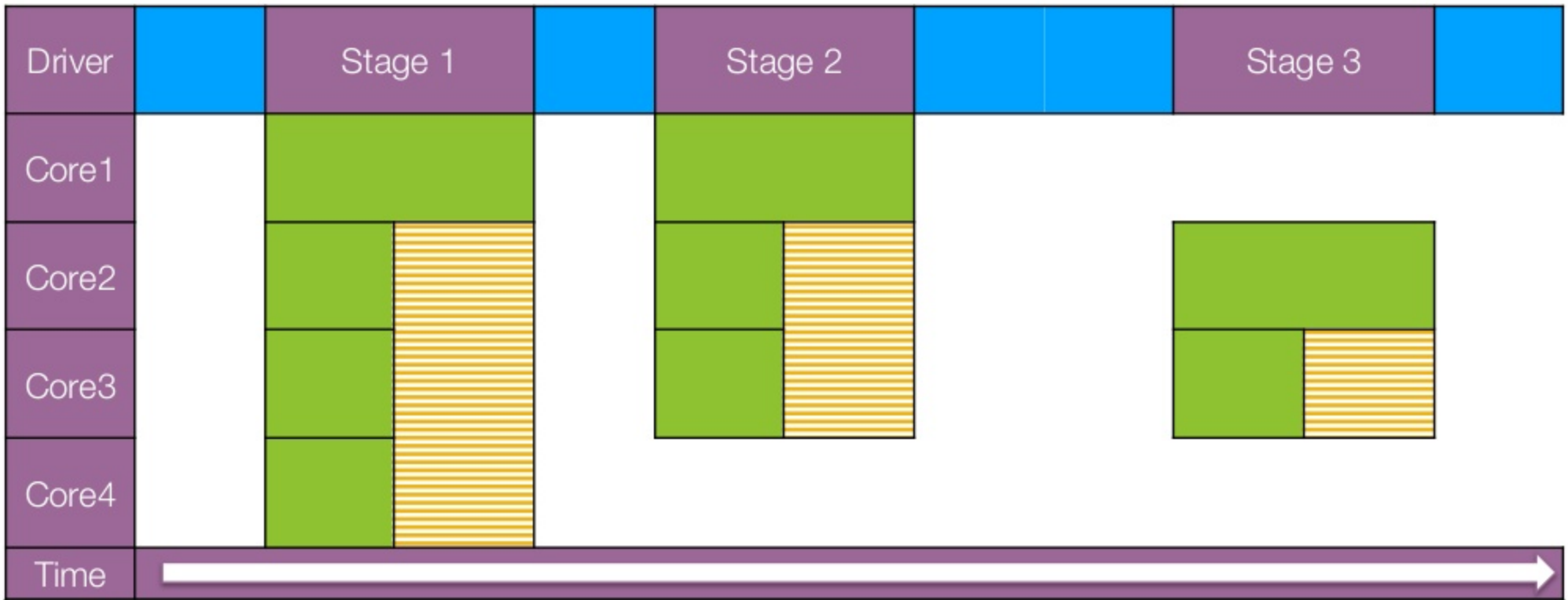
Not Enough Tasks



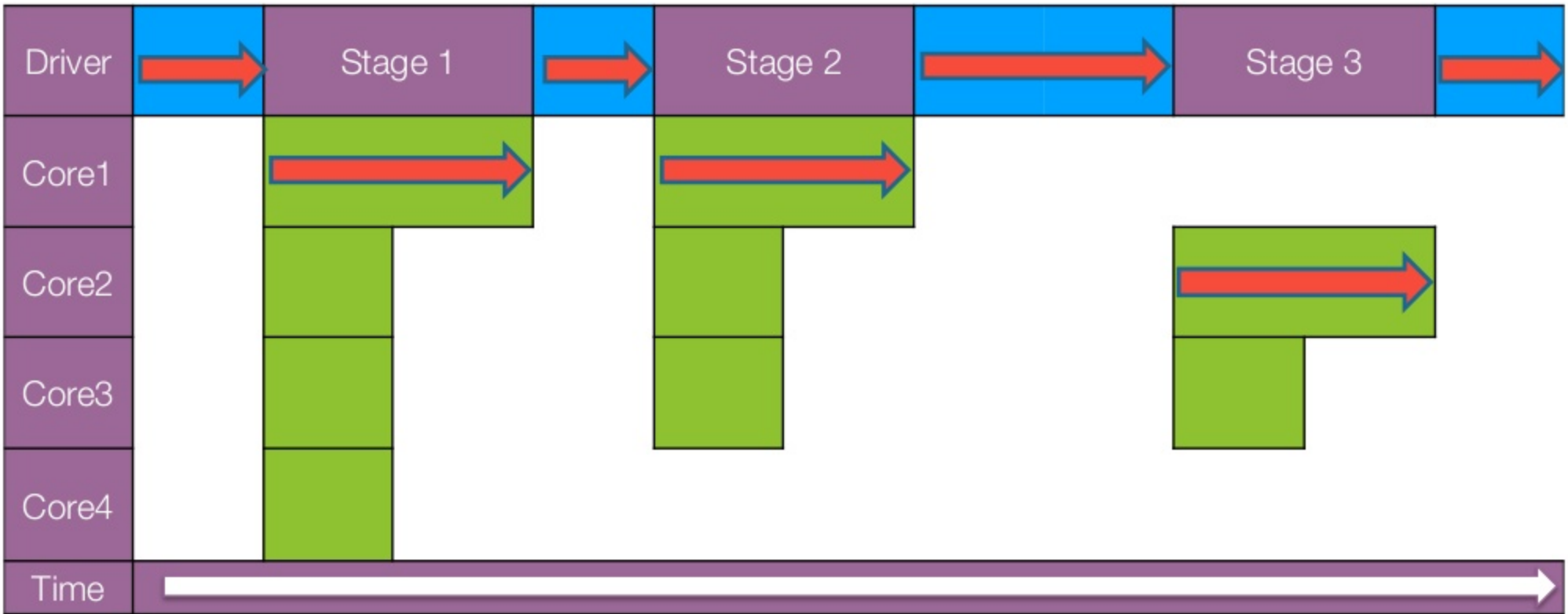
Controlling number of tasks

- HDFS block size
- Min/max split size
- Default Parallelism
- Shuffle Partitions
- Repartitions

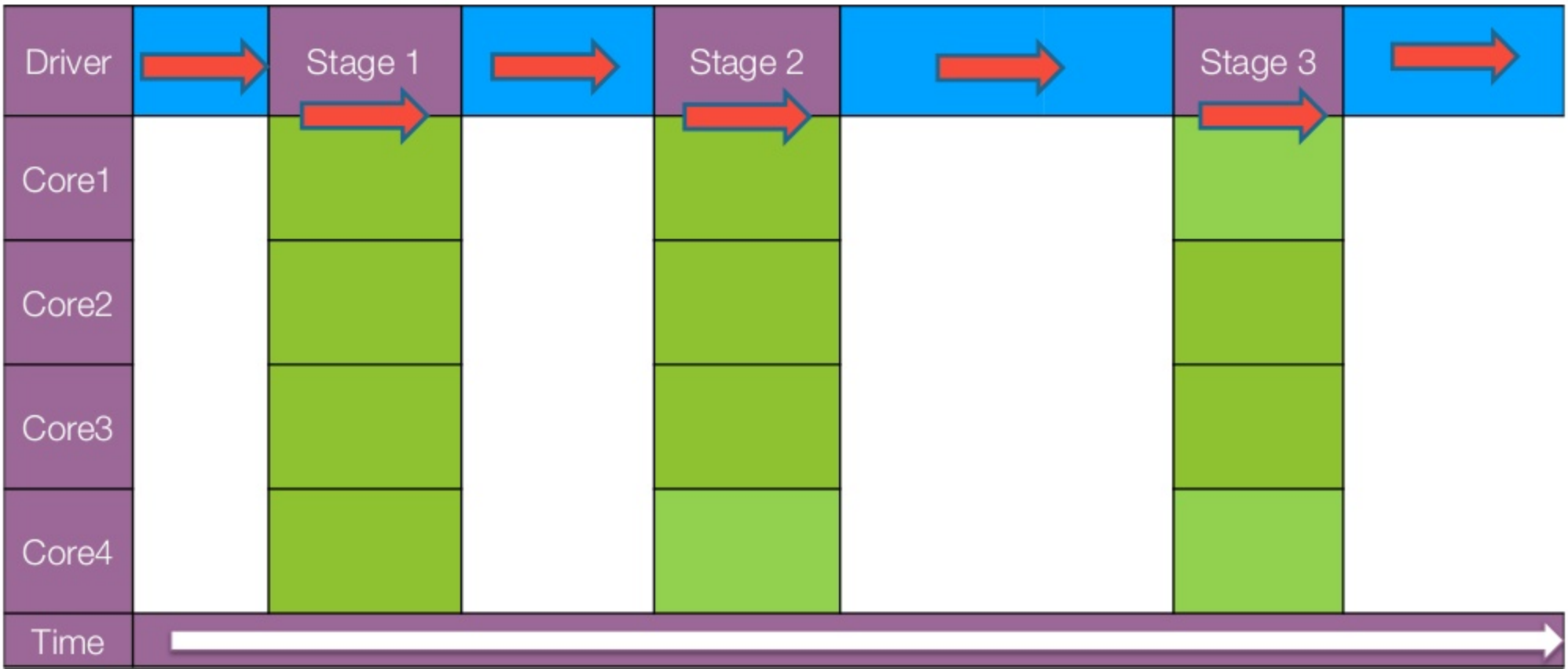
Non-Uniform Tasks: Skew



Critical Path: Limit to scalability



Ideal Application Time



Latency Vs Compute vs Developer

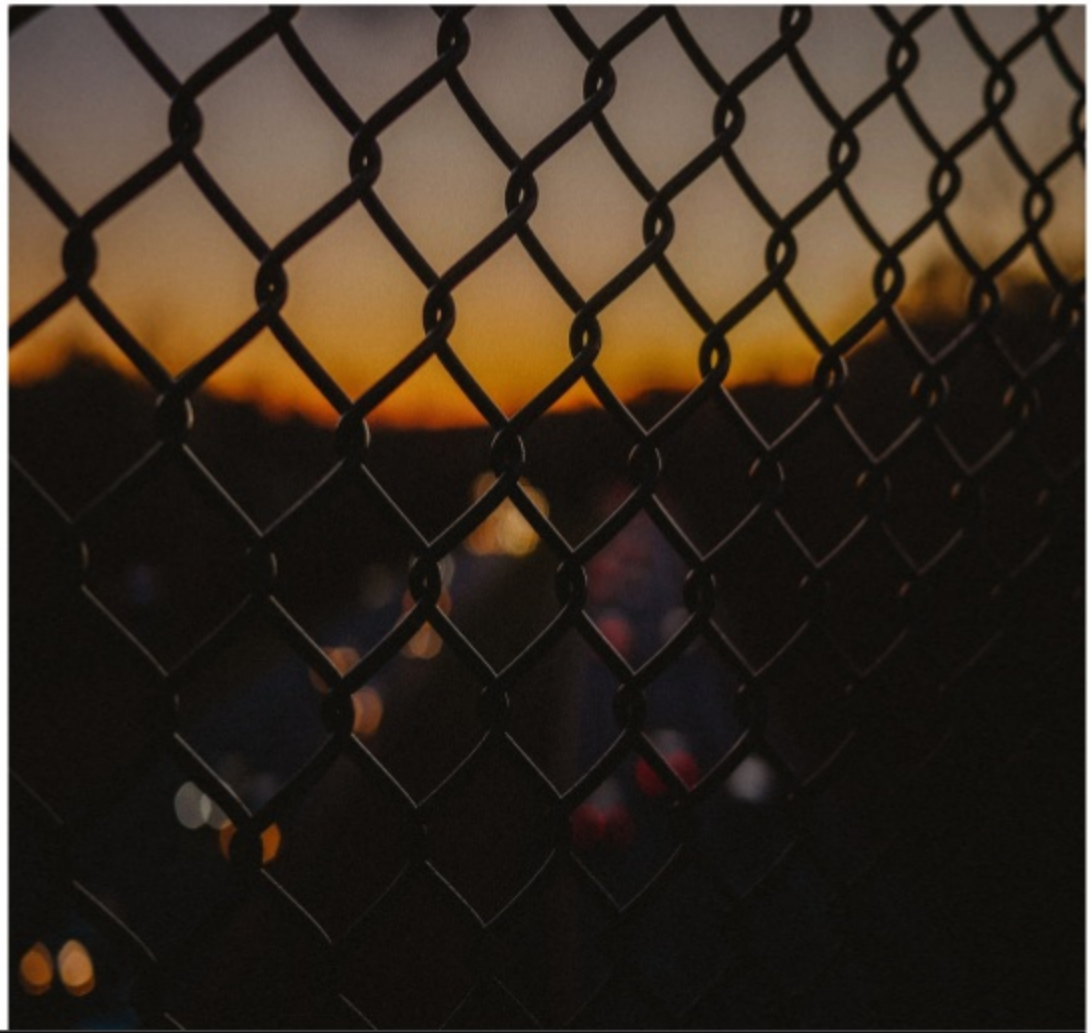
Wall Clock Time

Critical Path Time

Ideal* Application Time

Structure of a Spark application

- Spark application is either executing in driver or in parallel in executors
- Child stage is not executed until all parent stages are complete
- Stage is not complete until all tasks of stage are complete



Sparklens in Action

Performance Tuning 603 lines of unfamiliar Scala code

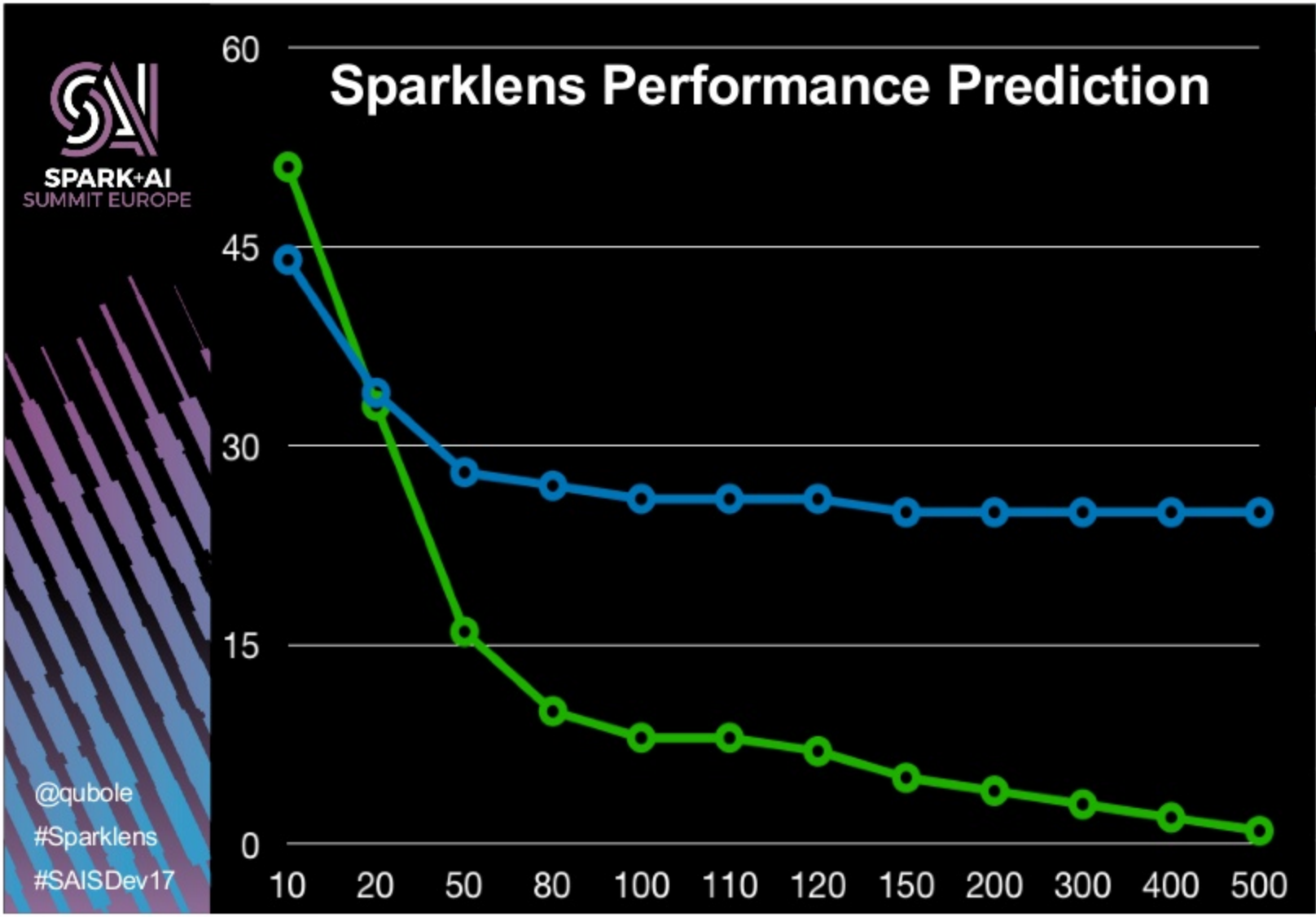

SPARK+AI
SUMMIT EUROPE
Sparklens: First Pass
Driver WallClock 41m 40s 26%
Executor WallClock 117m 03s 74%
Total WallClock 158m 44s
Critical Path 127m 41s
Ideal Application 43m 32s
@qubole
#Sparklens
#SAISDev17

Observations & Actions

- The application had too many stages (697)
- The Critical Path Time was 3X the Ideal Application Time
- Instead of letting spark write to hive table, the code was doing serial writes to each partition, in a loop
- We changed the code to let spark write to partitions in parallel

Sparklens: Second Pass

Driver WallClock	02m 28s	9%
Executor WallClock	24m 03s	91%
Total WallClock	26m 32s	
Critical Path	25m 27s	
Ideal Application	04m 48s	

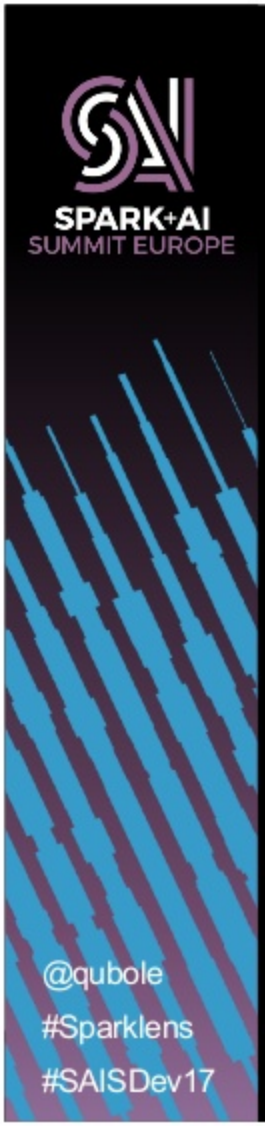


Count	Time	Utilisation
10	44m	51%
20	34m	33%
50	28m	16%
80	27m	10%
100	**26m**	8%
110	26m	8%
120	26m	7%
150	25m	5%
200	25m	4%
300	25m	3%
400	25m	2%
500	25m	1%

Executor Utilisation

```
ECCH available 320h 50m
ECCH used       31h 00m   9%
ECCH wasted    289h 50m 91%
```

ECCH: Executor Core Compute Hour

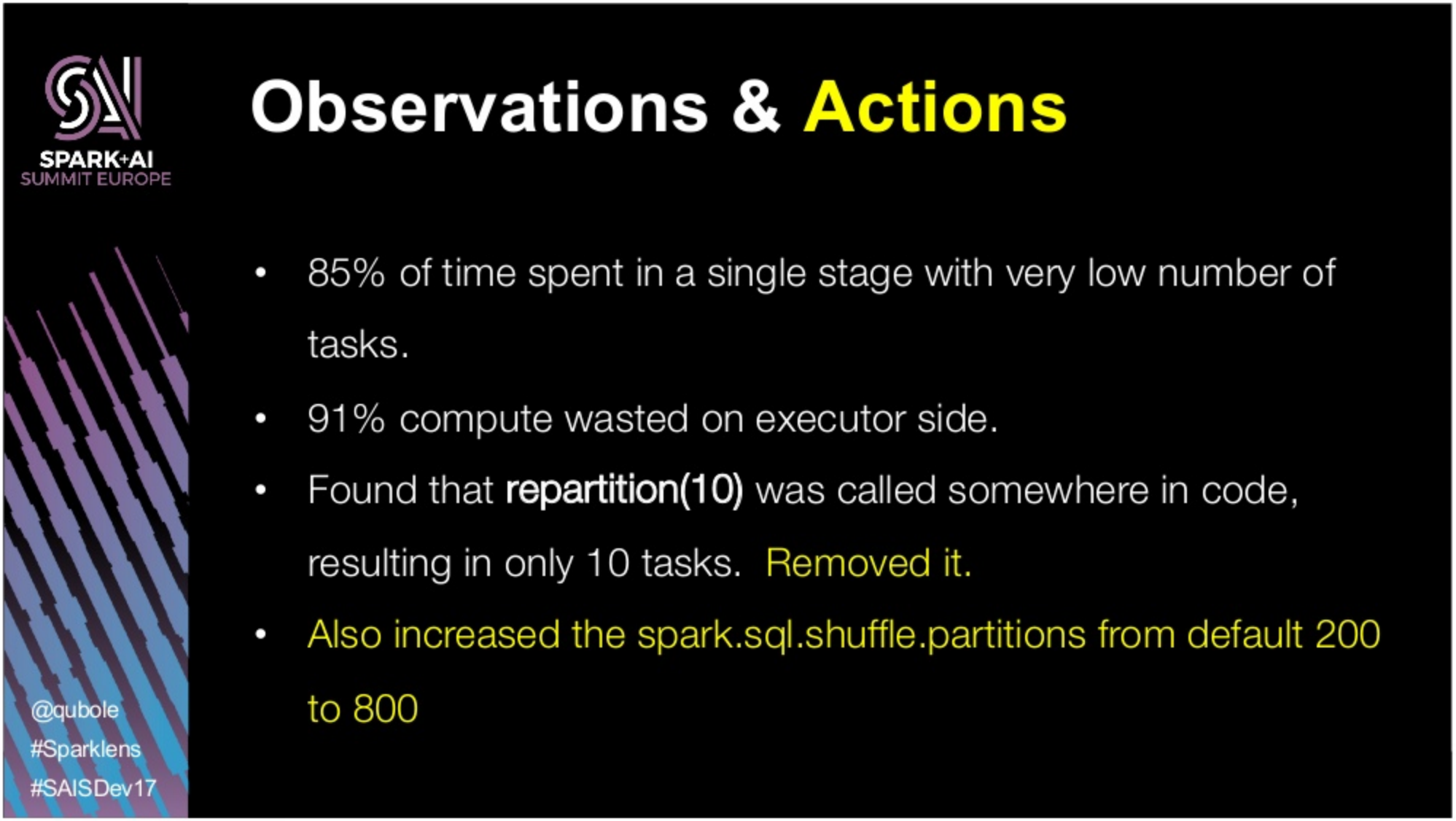


Per Stage Metrics

Stage-ID	WallClock Stage%	Core ComputeHours	Task Count	PRatio	-----Task------ Skew	StageSkew
0	0.27	00h 00m	2	0.00	1.00	0.78
1	0.37	00h 00m	10	0.01	1.05	0.85
33	85.84	03h 18m	10	0.01	1.07	1.00

Stage-ID	OIRatio	|* ShuffleWrite%	ReadFetch%	GC% *|
0	0.00	|* 0.00	0.00	3.03 *|
1	0.00	|* 0.00	0.00	2.02 *|
33	0.00	|* 0.00	0.00	0.23 *|

CCH	3h 18m
Task Count	10
Total Cores	800


SPARK+AI
SUMMIT EUROPE
Observations & Actions
• 85% of time spent in a single stage with very low number of tasks.
• 91% compute wasted on executor side.
• Found that repartition(10) was called somewhere in code, resulting in only 10 tasks. Removed it.
• Also increased the spark.sql.shuffle.partitions from default 200 to 800
@qubole
#Sparklens
#SAISDev17

Sparklens: Third Pass

Driver WallClock	02m 34s	26%
Executor WallClock	07m 13s	74%
Total WallClock	09m 48s	
Critical Path	07m 18s	
Ideal Application	07m 09s	

Using Sparklens

For inline processing, add following extra command line options to spark-submit

```
–packages qubole:sparklens:0.2.0-s_2.11
–conf spark.extraListener=com.qubole.sparklens.QuboleJobListener
```

Old event log files (history server)

```
–packages qubole:sparklens:0.2.0-s_2.11 --class
com.qubole.sparklens.app.ReporterApp dummy-arg <eventLogFile>
source=history
```

Special Sparklens output files (very small file with all the relevant data)

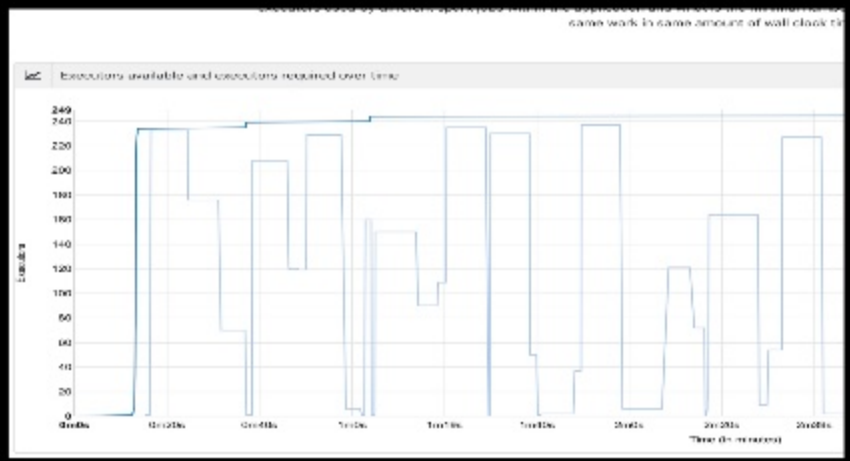
```
–packages qubole:sparklens:0.2.0-s_2.11 --class
com.qubole.sparklens.app.ReporterApp dummy-arg <eventLogFile>
```

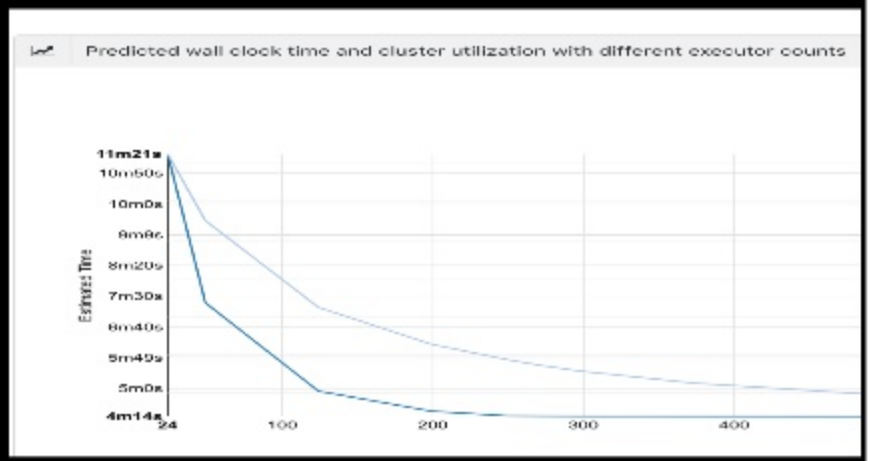
@qubole
#Sparklens
#SAISDev17

https://github.com/qubole/sparklens

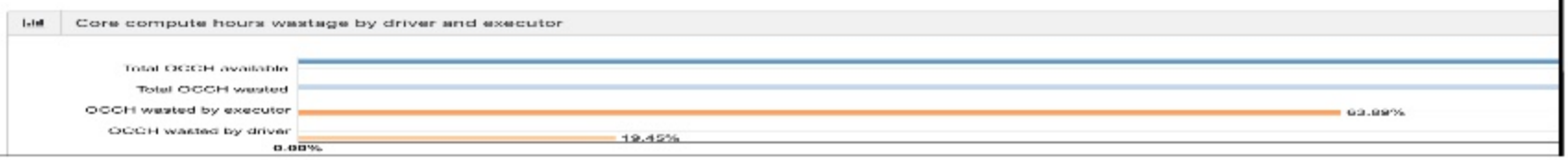
SPARK+AI SUMMIT EUROPE

http://sparklens.qubole.net





computed by assuming ideal partitioning (tasks == cores and no skew) of data in all stages of the application.



@qubole
#Sparklens
#SAISDev17

Future Work

- Sparklens for spark pipelines
- Elasticity aware autoscaling