

Information Engineering 2

Spark Best Practices and Applications

Prof. Dr. Kurt Stockinger

Semesterplan

SW	Datum	Vorlesungsthema	Praktikum
1	23.02.2022	Data Warehousing Einführung	Praktikum 1: KNIME Tutorial
2	02.03.2022	Dimensionale Datenmodellierung 1	Praktikum 1: KNIME Tutorial (Vertiefung)
3	09.03.2022	Dimensionale Datenmodellierung 2	Praktikum 2: Datenmodellierung
4	16.03.2022	Datenqualität und Data Matching	Praktikum 3: Star-Schema, Bonus: Praktikum 4: Slowly Changing Dimensions
5	23.03.2022	Big Data Einführung	DWH Projekt - Teil 1
6	30.03.2022	Spark - Data Frames	DWH Projekt - Teil 2 (Abgabe: 4.4.2022 23:59:59)
7	06.04.2022	Data Storage: Hadoop Distributed File System & Parquet	Praktikum 1: Data Frames
8	13.04.2022	Query Optimization	Praktikum 2: Data Storage
9	20.04.2022	Spark Best Practices & Applications	Praktikum 3: Query Optimization & Performance Analysis
10	27.04.2022	Machine Learning mit Spark 1	Praktikum 3: Query Optimization & Performance Analysis (Vertiefung)
11	04.05.2022	Machine Learning mit Spark 2 + Q&A	Praktikum 4: Machine Learning (Regression)
12	11.05.2022	NoSQL Systems	Big Data Projekt - Teil 1
13	18.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 2
14	25.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 3 (Abgabe: 30.5.2022 23:59:59)

Educational Objectives for Today

- You learn about **memory management**
- You understand the **impact** of memory on the **run time** of jobs
- Understand how **jobs** work
- Understand performance **improvements for filter queries** (Databricks Delta)
- Learn about various **big data applications**

Memory Management

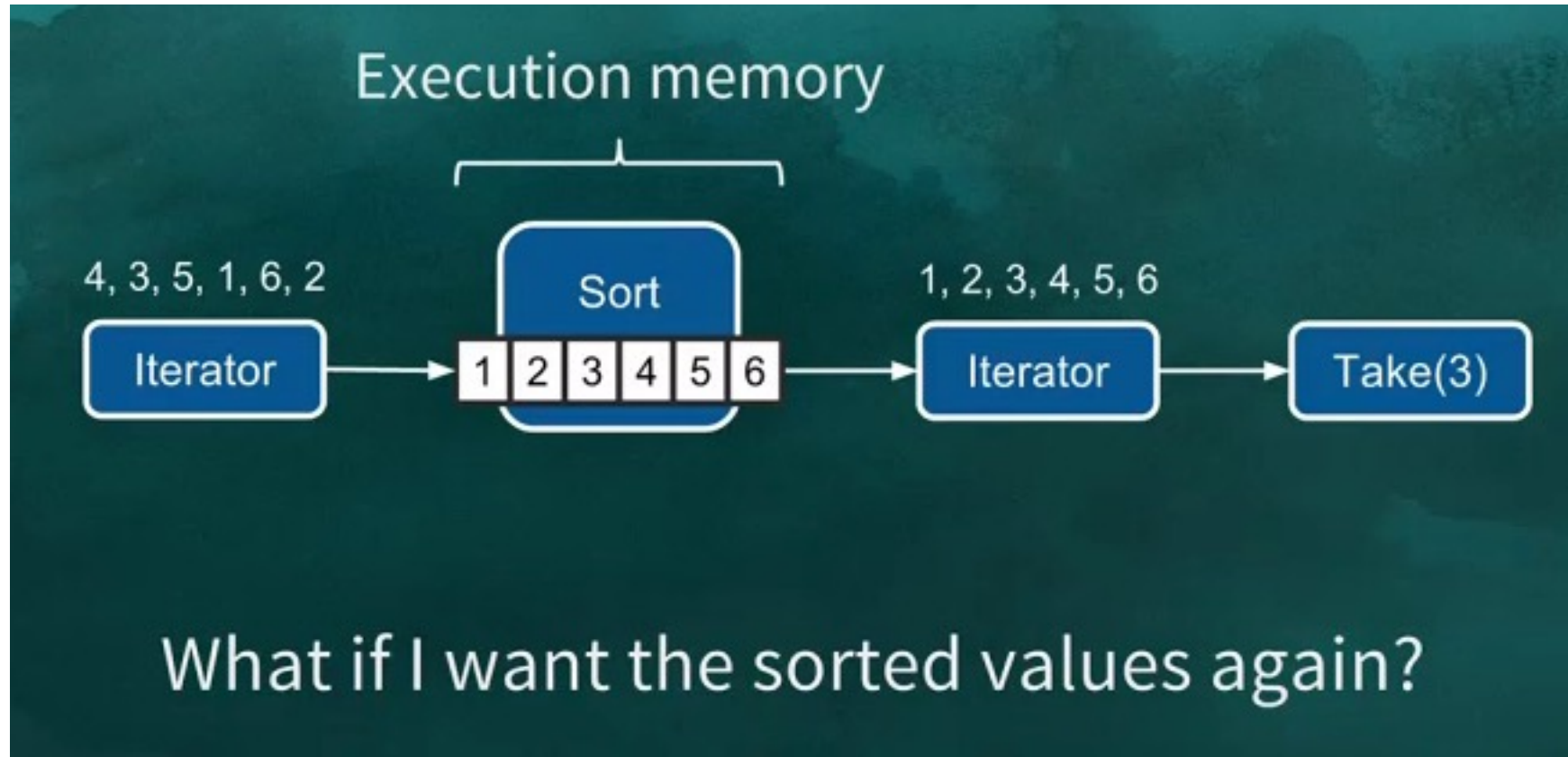
Why is Main Memory Management Important?

- Critical for **performance**
- Memory is a **scarce** resource
- **Memory contention** between:
 - Execution and storage
 - Tasks running in parallel
 - Across operators within a task (e.g. sort, group by)

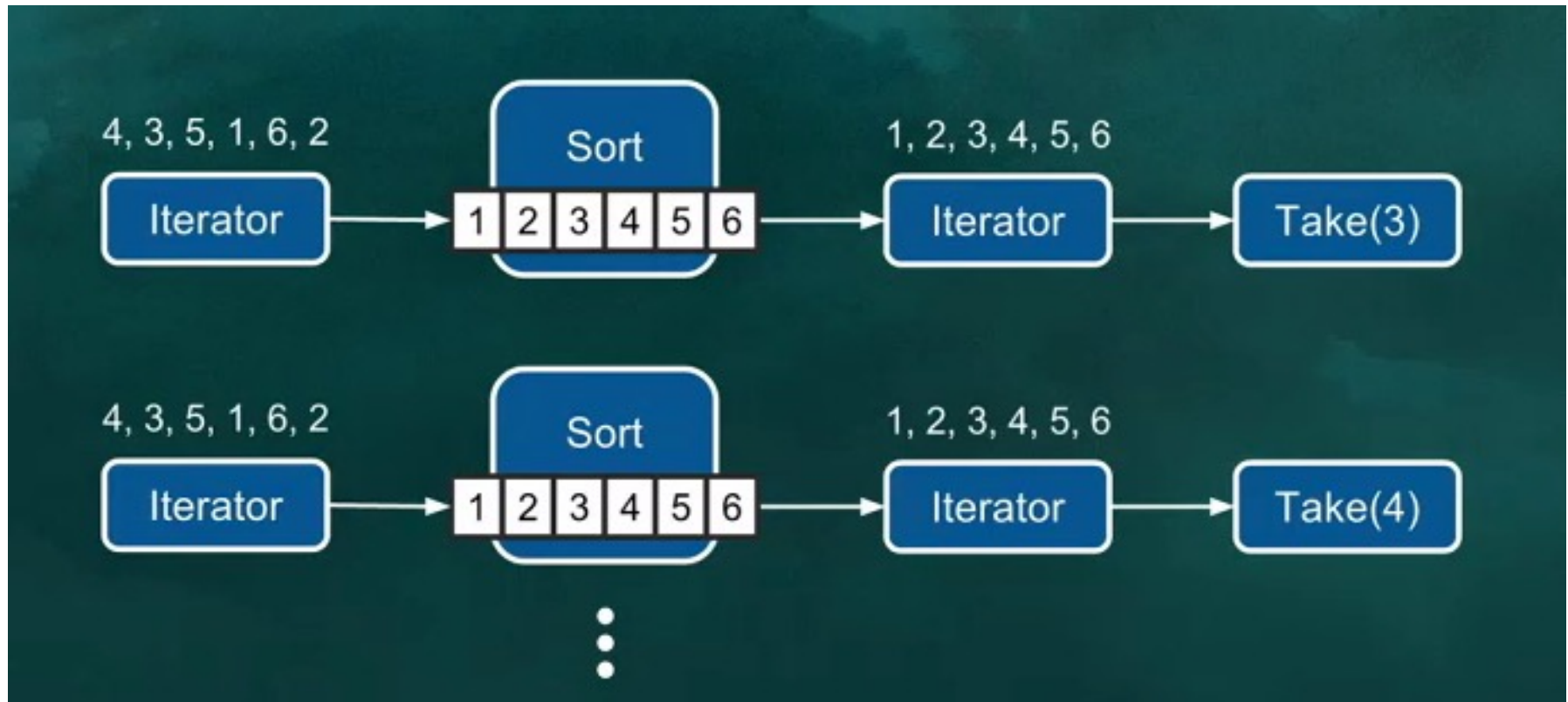
Two Usages of Memory in Spark

- **Execution:**
 - Memory used for **shuffles, joins, sorts and aggregations**
 - E.g. if “group by” is executed, it needs to be stored in hash map (needs memory)
 - Execution memory is **short-lived**:
 - After execution is finished, storage is released
- **Storage:**
 - Memory used to **cache** data that will be reused later
 - E.g. when user caches a data set
 - Storage is **needed for future** computation:
 - Needs to be explicitly released

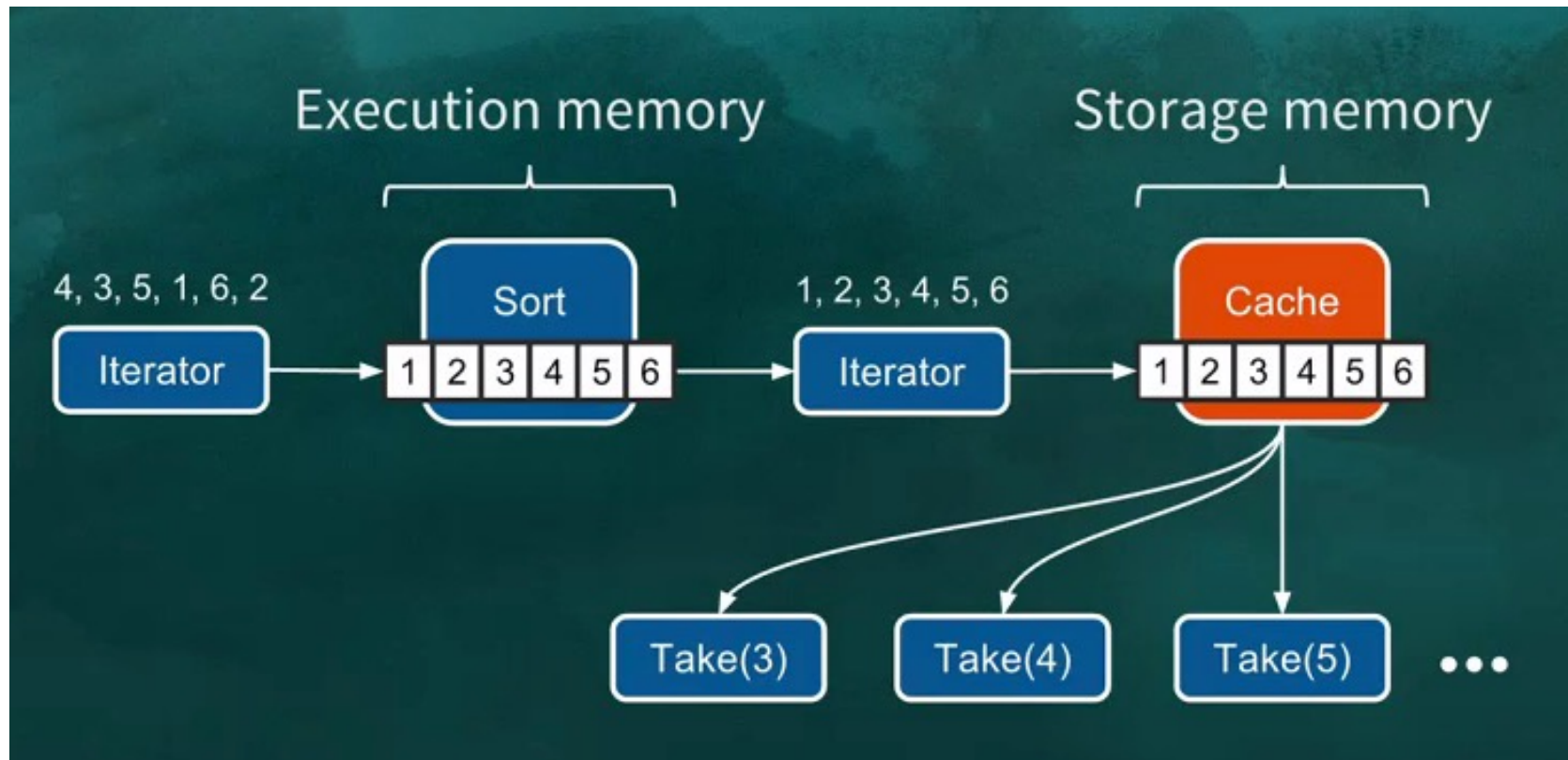
Example of Memory Usage



Iterators Can only be Traversed Once: Need to Redo Sorting

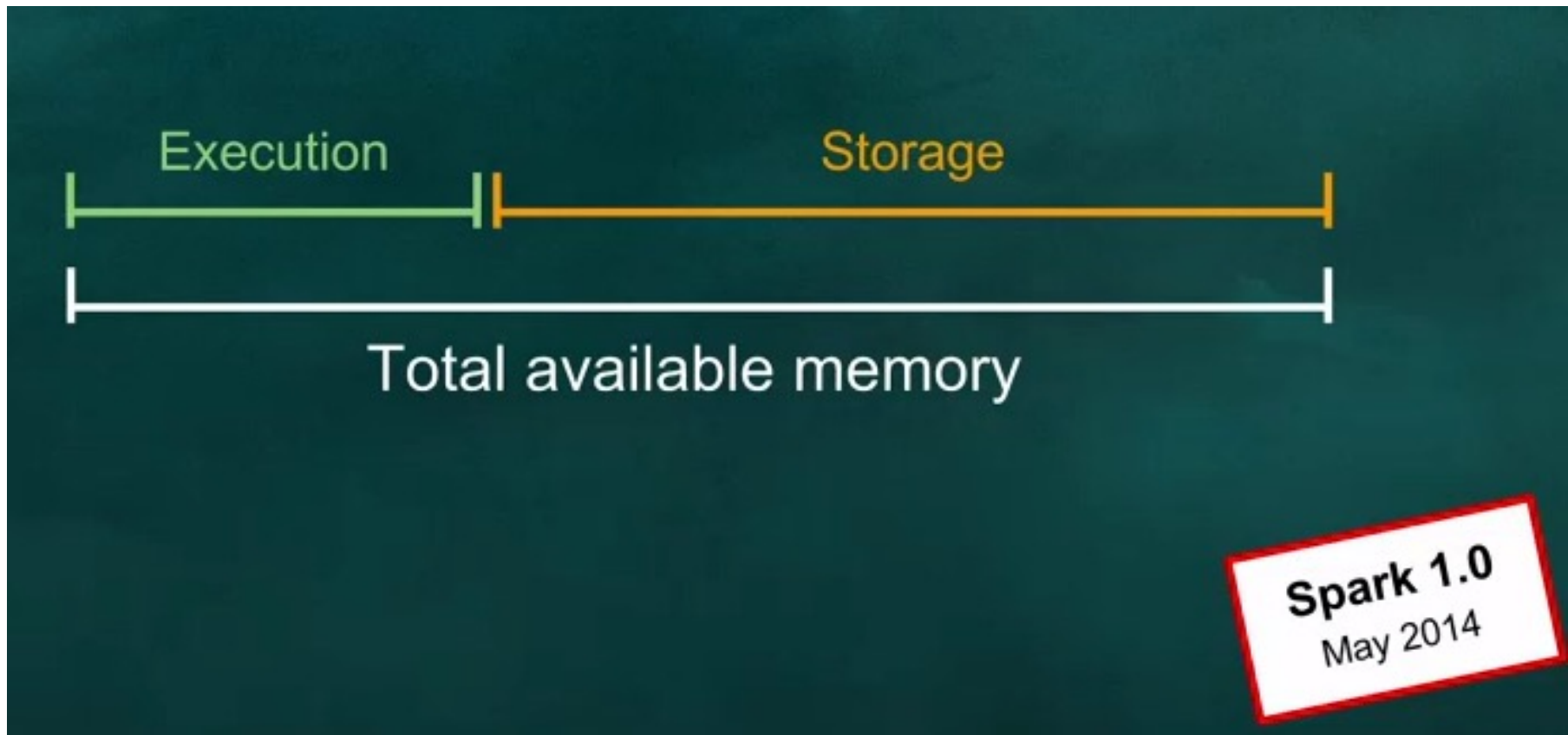


Better Approach: Cache the Result

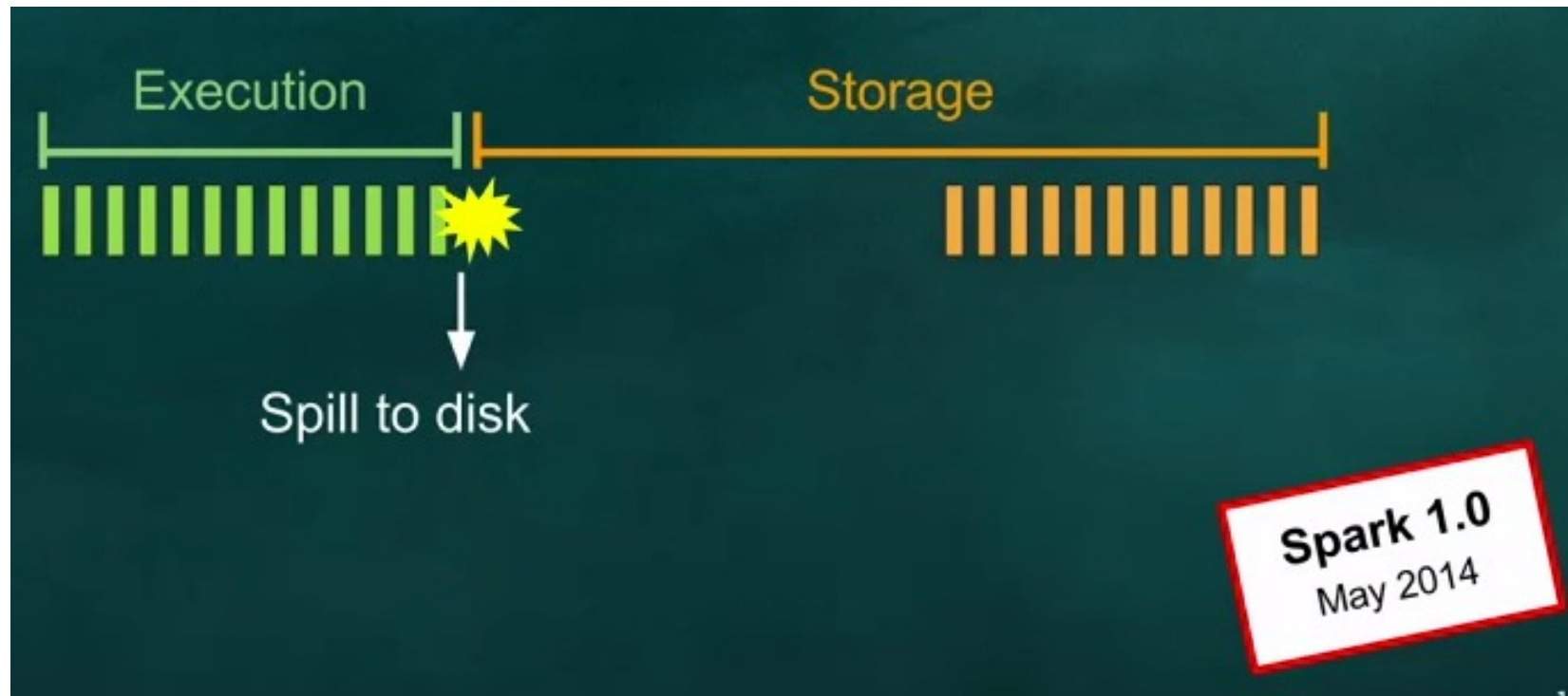


Only useful if results are reused several times since caching also takes time.

Static Memory Assignment

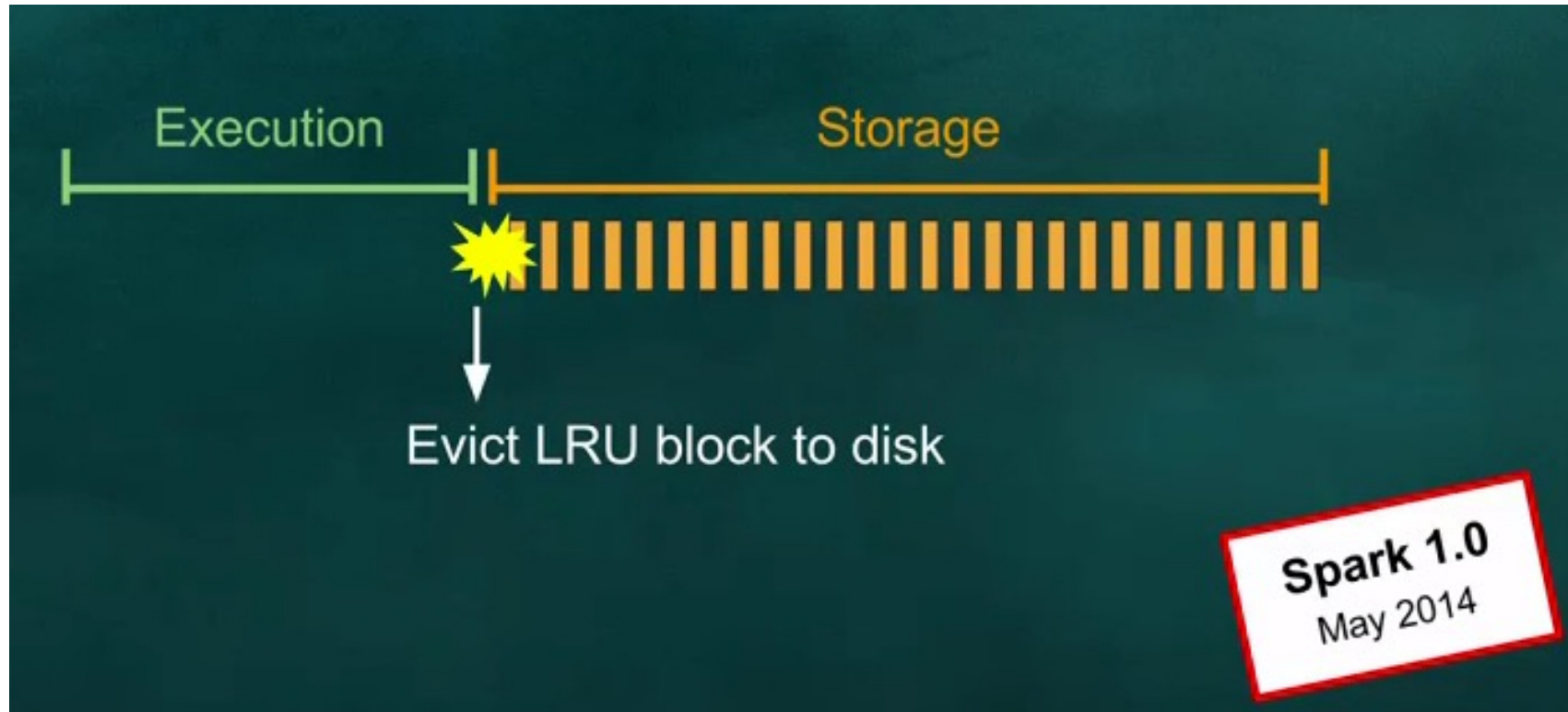


Problems



When execution memory runs out, data needs to be spilled to disk → expensive!

Problems



LRU = least recently used

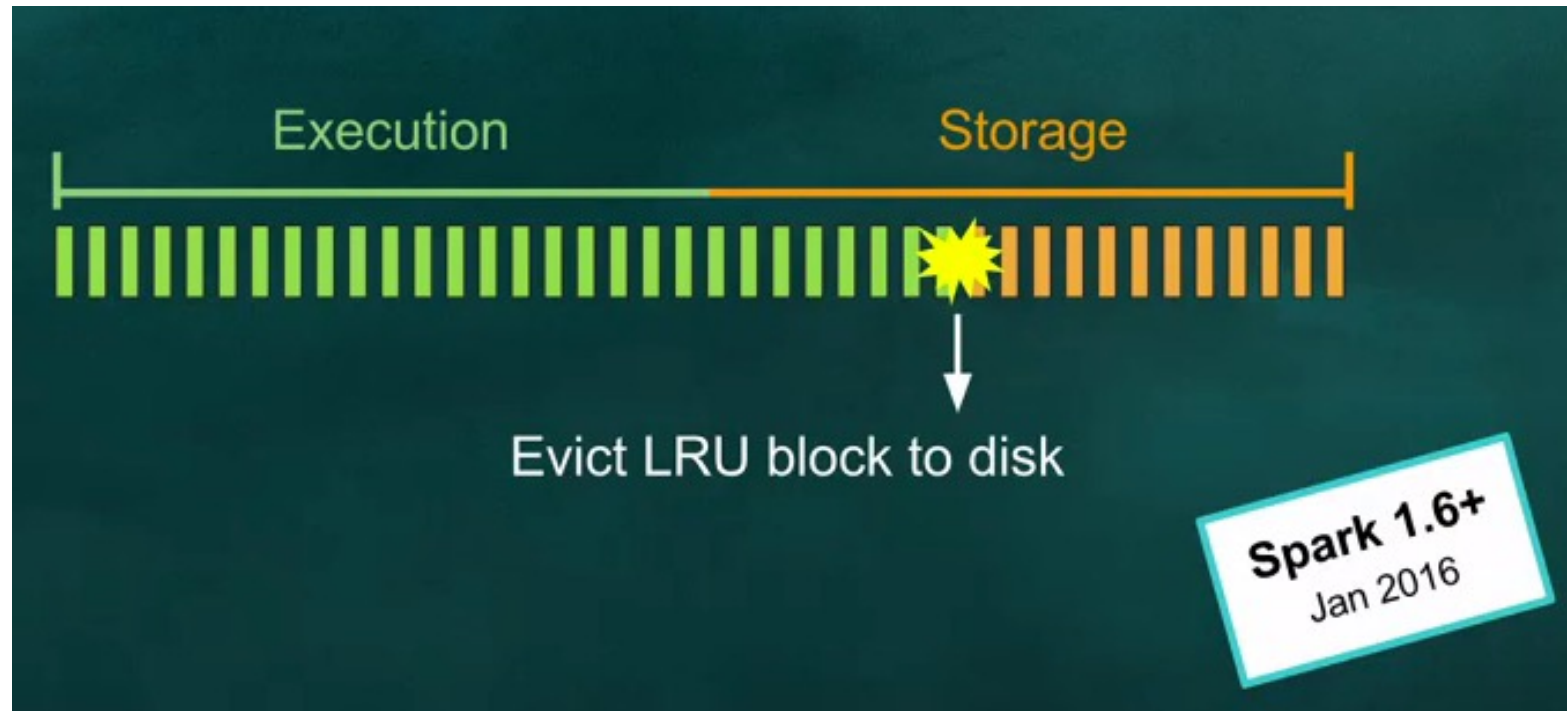
Block corresponds to RDD partition

Better Solution: Unified Memory Management



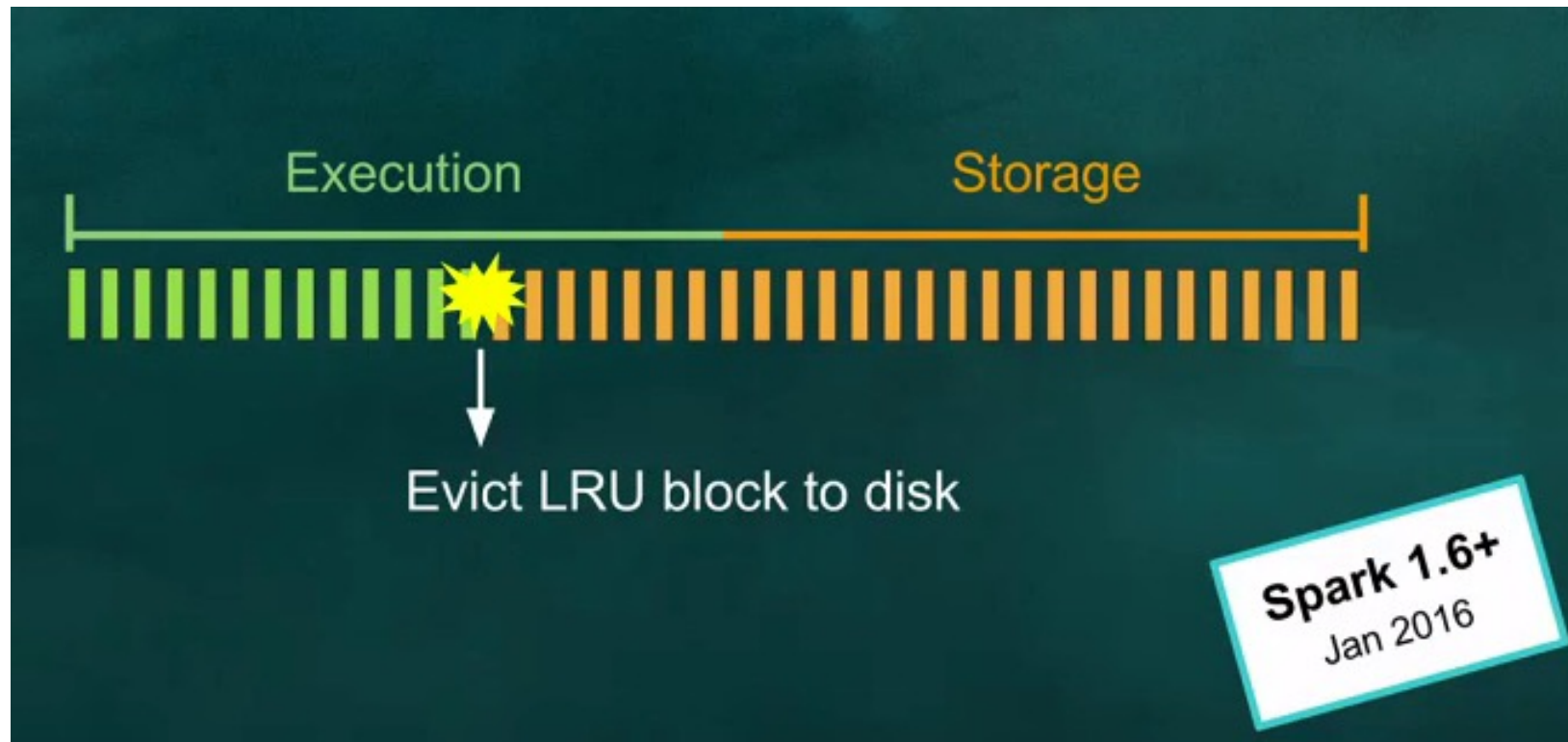
Express the two storages as one common region.

Better Solution: Unified Memory Management



Evict storage when execution needs more memory.

Better Solution: Unified Memory Management



Also evict storage when storage grows. Why?

Design Consideration

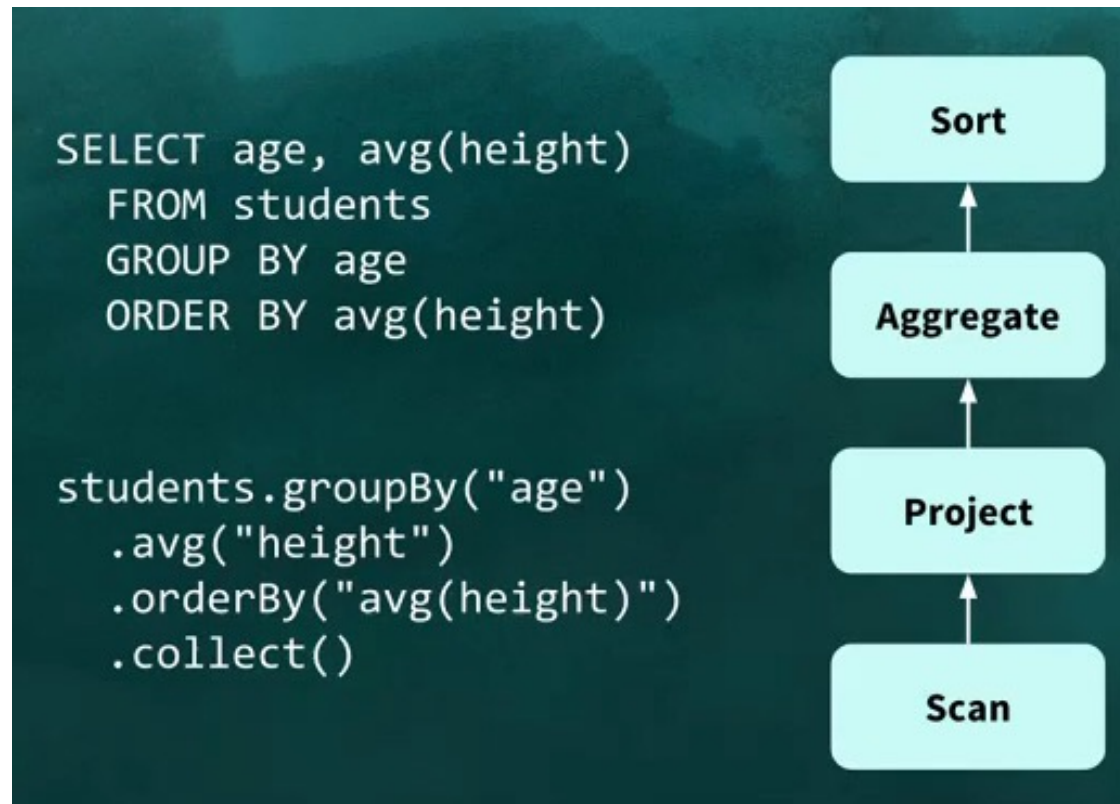
- Why evict storage and not execution?
 - Spilled execution data will always be read back from disk
 - Cached data may not be read back from disk

Memory Contention among Parallel Tasks: Dynamic Assignment

Each task is now assigned $1/N$ of
the memory, where $N = 4$



Memory Contention within a Task



Executed within one task → memory contention between operators

Memory Contention within a Task

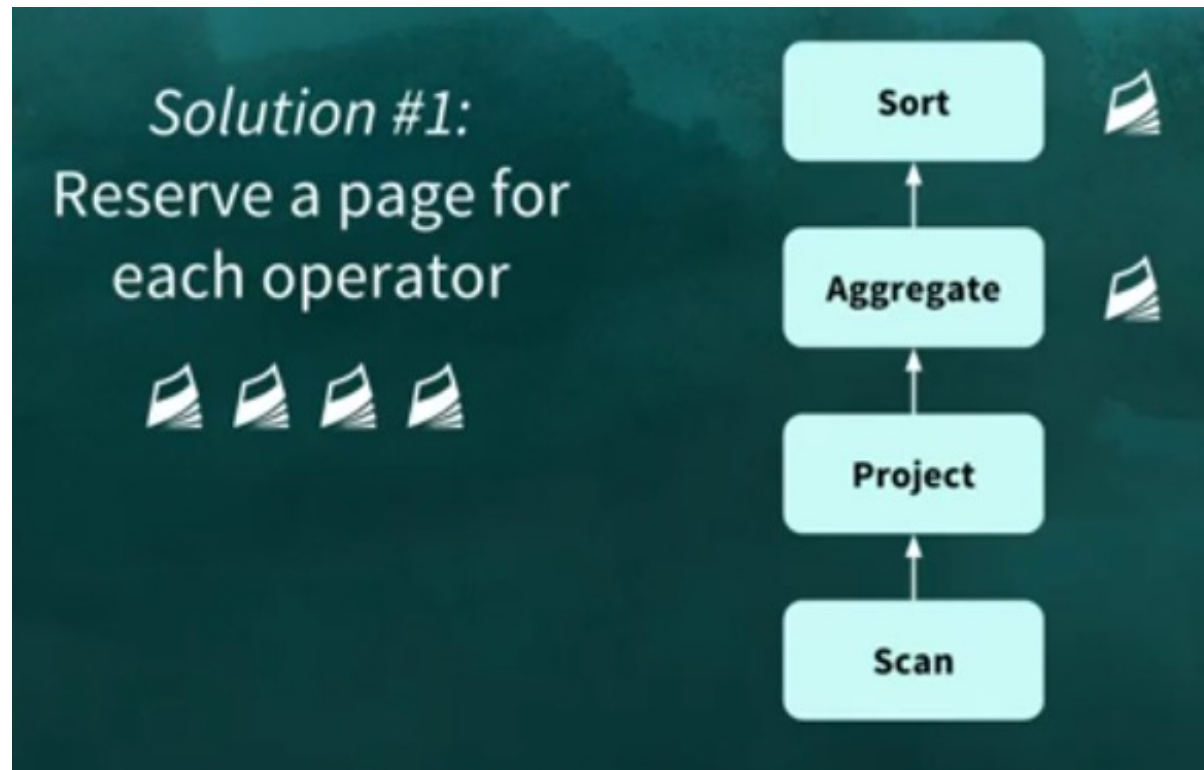


Assume: 6 memory pages available.

Aggregate operator uses up all of them for hash map.

Sort operator has no memory left.

Solution 1

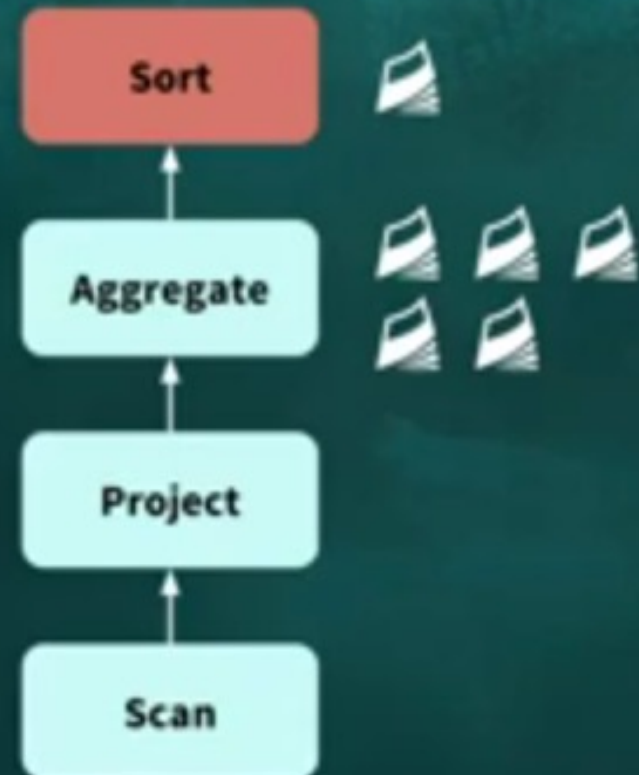


Not fair since aggregate operator can use up remaining four pages →
Sort operator will “starve”

Better Solution

Solution #2: Cooperative spilling

Sort needs more memory so
it forces *Aggregate* to spill
another page (and so on)



Understanding Jobs

RDD Example

Input file:

```
this is a little test file  
file1 file1 file2 file3  
aaa bbb ccc ddd  
file file777  
is this good?
```

Program code:

```
1  textFile = sc.textFile("/FileStore/tables/data_text.txt", 4)  
2  print 'collect: %s ' % textFile.collect()  
3  
4  lines = textFile.filter(lambda line: "file2" in line)  
5  print 'lines: %s ' % lines.collect()  
6  
7  wordCount = textFile.flatMap(lambda line: line.split()).map(lambda word: (word,1)).reduceByKey(lambda a,b: a+b)  
8  print 'wordCount: %s ' % wordCount.collect()
```

Even More Simplified

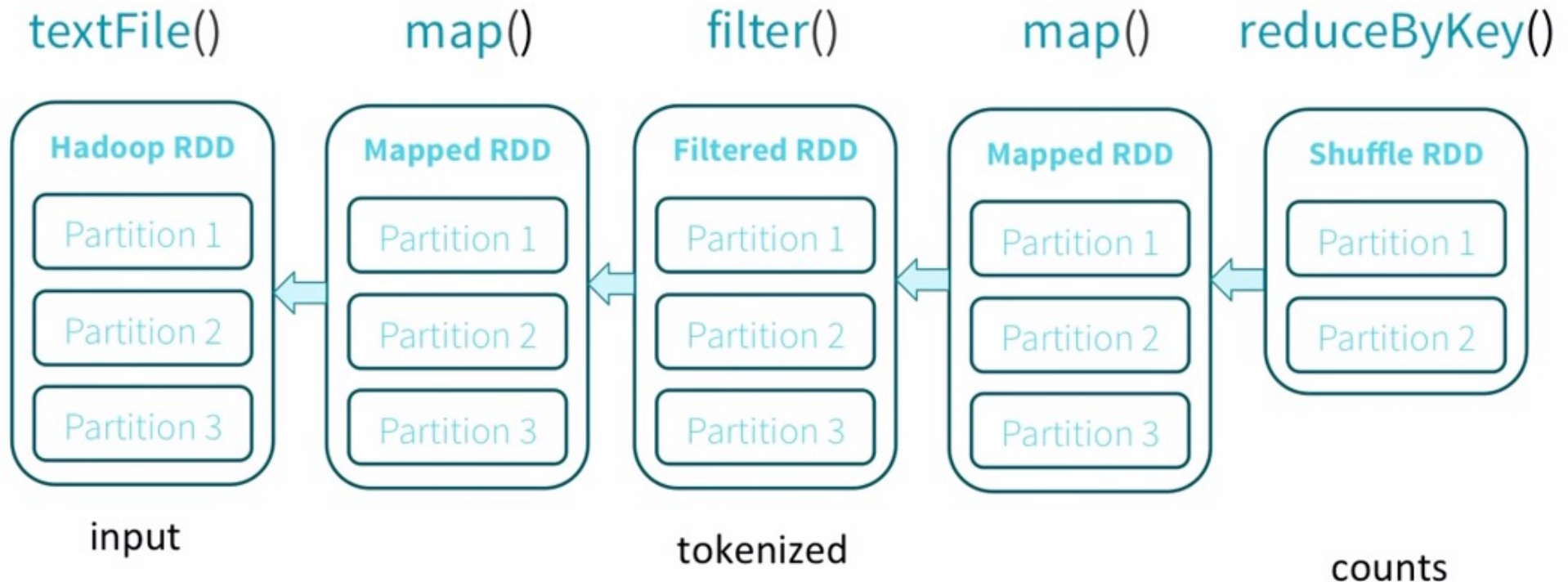
```
sc.textFile().map().filter().map().reduceByKey()
```

What is a good execution model?

Should one function be executed one after the other?

DAG View of RDDs

DAG = Directed Acyclic Graph

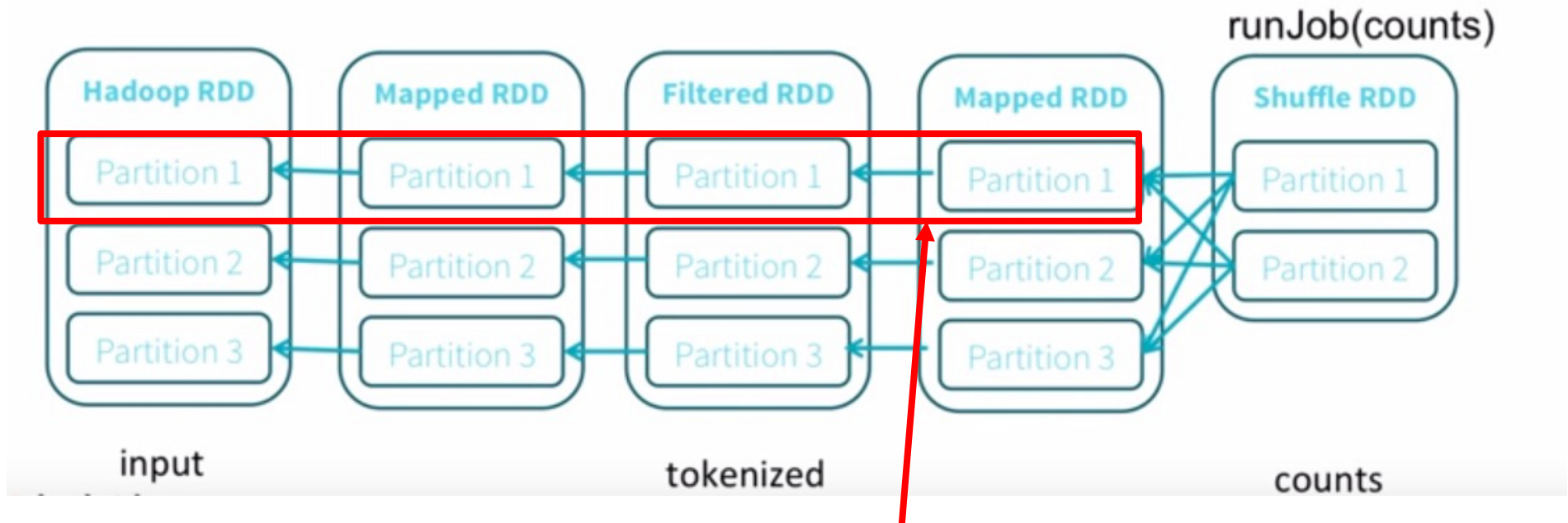


Partition corresponds to Hadoop's input block size = unit of parallelism

Physical Optimizations

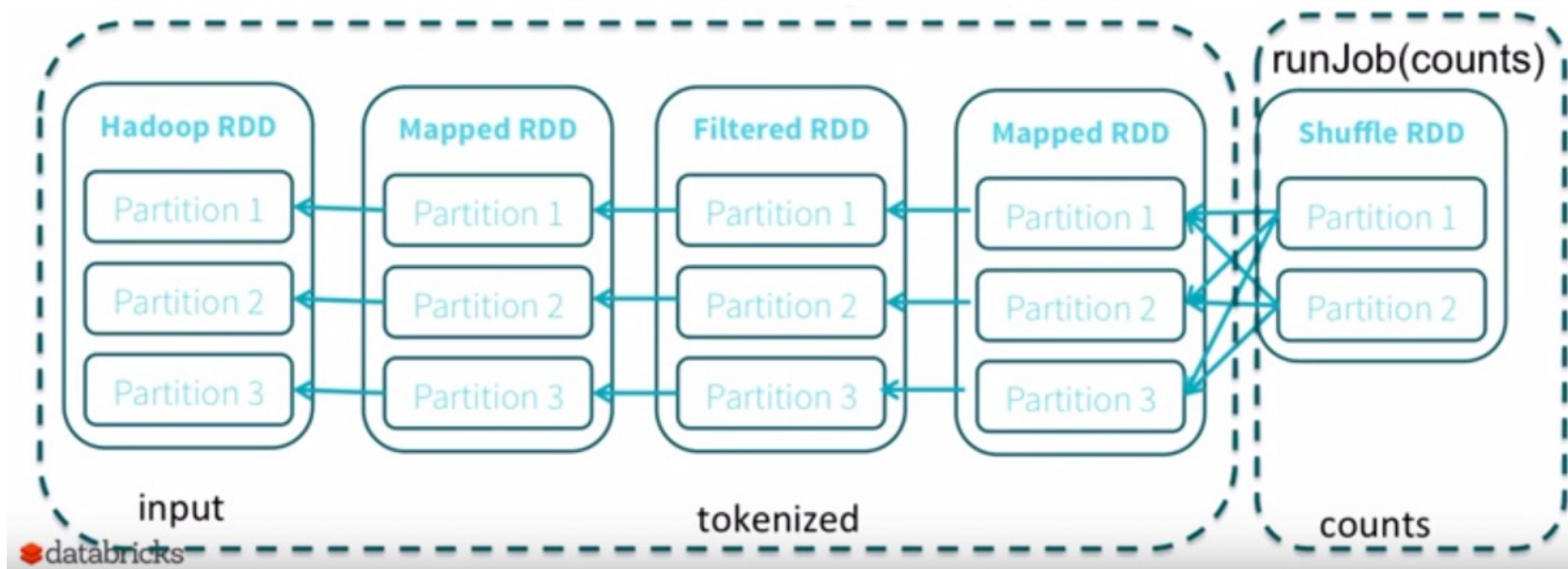
- Certain types of operations can be **pipelined**
- If dependent RDDs have already been cached (or persisted in a shuffle), the graph can be **truncated**
- Spark produces a set of **stages**:
 - Each stage consists of **tasks**

How runJob Works



Can be **pipelined** and **performed locally** (in-place computation without data movement)
Map → Filter can be run on each partition independently

Optimized Execution Consists of 2 Stages



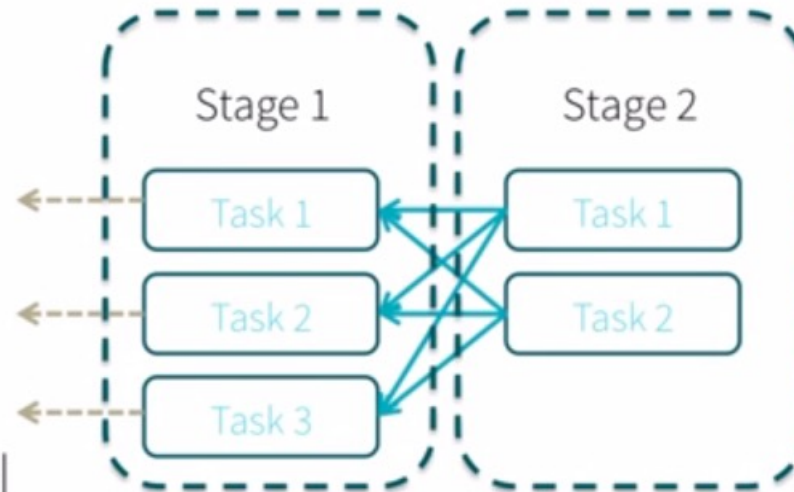
Stage 1

Stage 2

Stage Graph Consisting of Pipelined Tasks

Each task will:

1. Read Hadoop input
2. Perform maps and filters
3. Write partial sums (per partition)



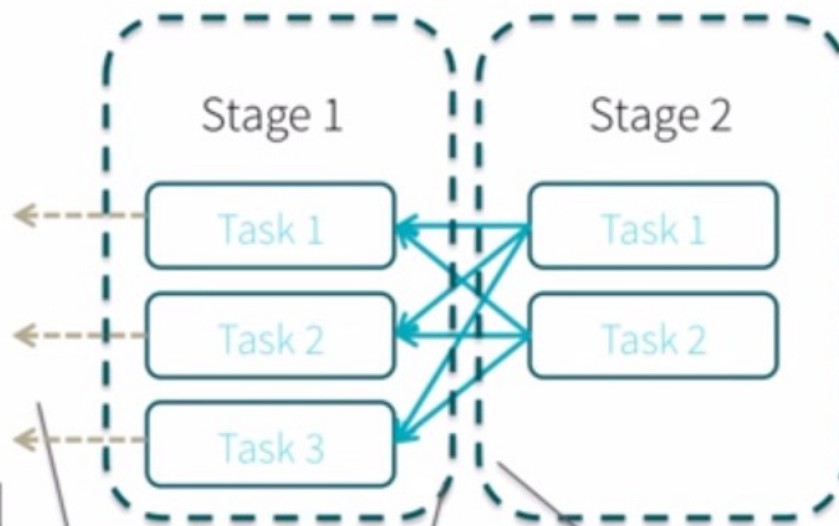
Each task will:

1. Read partial sums
2. Invoke user function passed to runJob.

Stage Graph Consisting of Pipelined Tasks #2

Each task will:

1. Read Hadoop input
2. Perform maps and filters
3. Write partial sums



Each task will:

1. Read partial sums
2. Invoke user function passed to runJob.

Check this information in Spark UI

Units of Physical Execution

- **Jobs:**
 - Work required to compute RDD in a runJob
- **Stages:**
 - Work within a job corresponding to one or more **pipelined RDDs**.
- **Tasks:**
 - Unit of work within a stage corresponding to one **RDD partition**
- **Shuffle:** Transfer of data between stages

Spark UI Output

Hostname: ec2-34-217-123-45.us-west-2.compute.amazonaws.com Spark Version: 4.2.x-scala2.11

Jobs Stages Storage Environment Executors SQL JDBC/ODBC Server

Details for Job 8

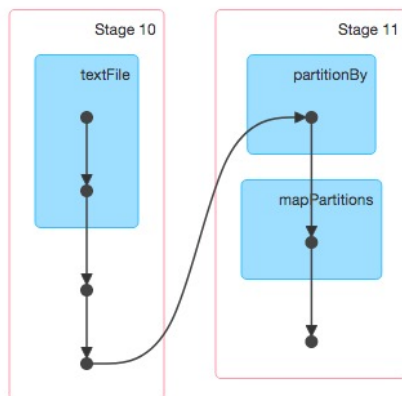
Status: SUCCEEDED

Job Group: 6930014662452935705_4735908371609143054_b5806fcf409b4279ae35ed15624dec92

Completed Stages: 2

► Event Timeline

▼ DAG Visualization



Completed Stages (2)

Stage Id ▾	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
11	6930014662452935705	textFile = sc.textFile("/FileStore/tables/data_... collect at <command-594278169302577>:8 +details	2018/09/19 13:46:46	0.1 s	5/5			905.0 B	
10	6930014662452935705	textFile = sc.textFile("/FileStore/tables/data_... reduceByKey at <command-594278169302577>:7 +details	2018/09/19 13:46:46	0.3 s	5/5				905.0 B

Speeding up Filter-Queries

Example Data Frame

A1	A2	A3	A3	A4	A5	A6	A7	A8	A9
3	8	6	3	2	8	8	9	2	8
...			4	5					
			5	2					
			3	3					
			8	7					
			1	8					
			0	2					...

Query: SELECT A1 WHERE A3 < 5 AND A4 > 9

What happens in Spark?

Spark Query Processing

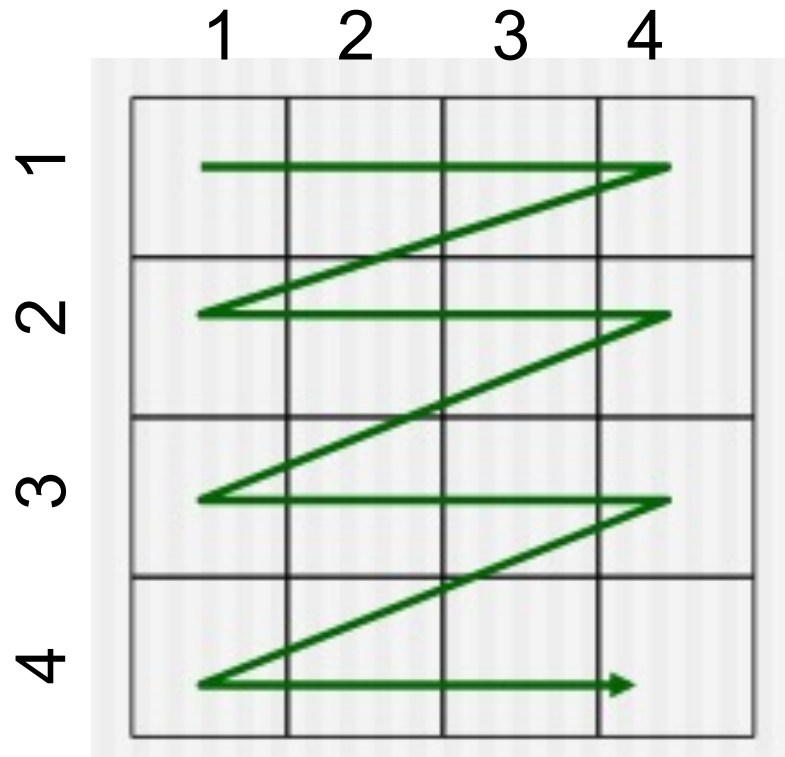
- Parquet storage:
 - Each attribute (column) is stored in a separate file
- Query “SELECT A1 WHERE A3 < 5 AND A4 > 9” scans two complete files
- Can’t we do better?
- What do traditional databases do?

Indexes for Query Processing

- Traditional databases use **indexes**:
 - One-dimensional queries: B-tree
 - Multi-dimensional queries: bitmap indexes
- Until recently Spark did not support indexes

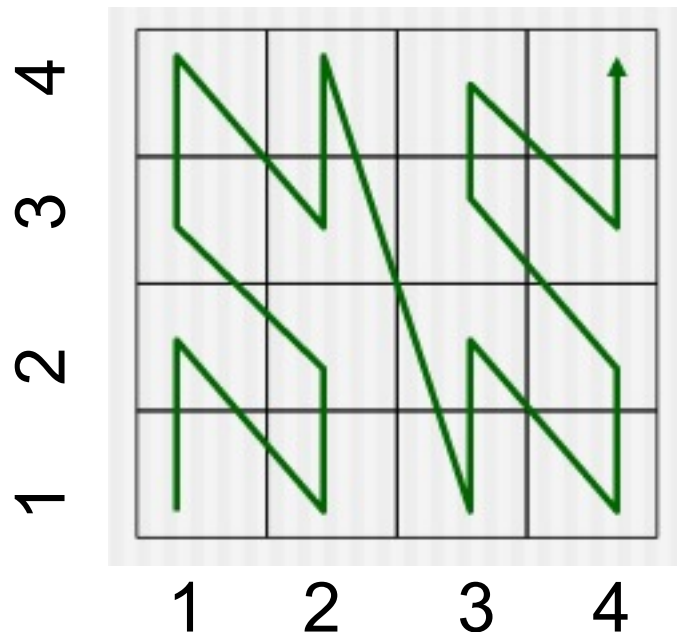
Row-Order Representation for Table with 2 Columns

- Good for x-axis (column 1), bad for y-axis (column 2)

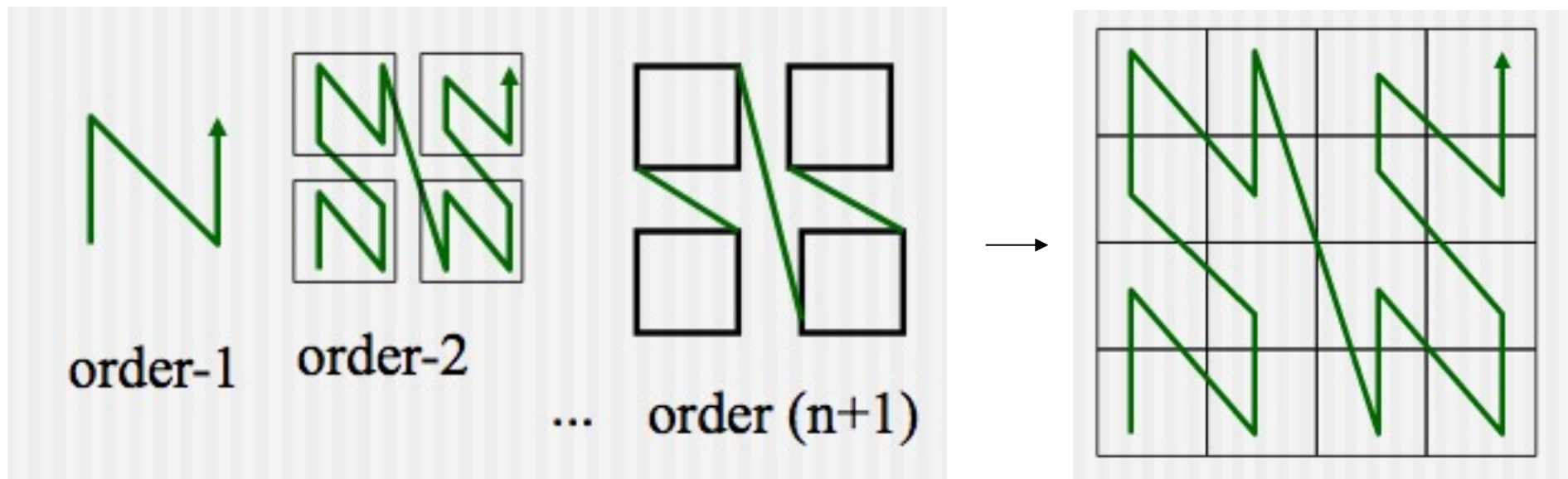


Z-Ordering

- Main idea:
 - Represent n-dimensional data in 1 dimension
 - Need to map n-d space to 1-d space



Z-Ordering: Recursive Construction



Advantage of Z-Order

- A multi-dimensional clustering technique
- Preserves data locality (co-locate related information in the same file)
- Good for multi-dimensional queries

Databricks Delta

- **Transactional storage manager** based on Apache Spark and Databricks DBFS
- “Optimized” Spark table
- Stores data as Parquet files
- Fast read access based on Z-ordering

Example

```
%sql
-- NOTE: This step may take a few minutes depending on your cluster configuration.
-- CLEANUP: Drop table
DROP TABLE IF EXISTS flights;

-- Create a standard table and import US based flights for year 2008
-- USING Clause: Specify "delta" format instead of the standard parquet format
-- PARTITIONED BY clause: Organize data based on "Origin" column (Originating Airport code).
-- FROM Clause: Import data from a csv file.
CREATE TABLE flights
USING delta
PARTITIONED BY (Origin)
SELECT _c0 as Year, _c1 as Month, _c2 as DayofMonth, _c3 as DayOfWeek, _c4 as DepartureTime, _c5 as CRSDepartureTime, _c6 as
ArrivalTime,
_c7 as CRSArrivalTime, _c8 as UniqueCarrier, _c9 as FlightNumber, _c10 as TailNumber, _c11 as ActualElapsedTime, _c12 as
CRSElapsedTime,
_c13 as AirTime, _c14 as ArrivalDelay, _c15 as DepartureDelay, _c16 as Origin, _c17 as Destination, _c18 as Distance,
_c19 as TaxiIn, _c20 as TaxiOut, _c21 as Cancelled, _c22 as CancellationCode, _c23 as Diverted, _c24 as CarrierDelay,
_c25 as WeatherDelay, _c26 as NASDelay, _c27 as SecurityDelay, _c28 as LateAircraftDelay
FROM csv.`dbfs:/databricks-datasets/asa/airlines/2008.csv`;

%sql
-- OPTIMIZE consolidates files and orders the Databricks Delta table data by DayOfWeek under each partition for faster retrieval
OPTIMIZE flights ZORDER BY (DayOfWeek);
```

Evaluation of Data Partitioning and Z-Ordering

Lehmann et al. *J Big Data* (2020) 7:61
<https://doi.org/10.1186/s40537-020-00340-7>


 Journal of Big Data

RESEARCH

Open Access



Big Data architecture for intelligent maintenance: a focus on query processing and machine learning algorithms

Claude Lehmann^{1*} , Lilach Goren Huber¹, Thomas Horisberger², Georg Scheiba², Ana Claudia Sima¹ and Kurt Stockinger¹

*Correspondence:
claude.lehmann@zhaw.ch
¹ Zurich University of Applied
Sciences, Obere Kirchgasse 2,
8400 Winterthur, Switzerland
Full list of author information
is available at the end of the
article

Abstract

Exploiting available condition monitoring data of industrial machines for intelligent maintenance purposes has been attracting attention in various application fields. Machine learning algorithms for fault detection, diagnosis and prognosis are popular and easily accessible. However, our experience in working at the intersection of academia and industry showed that the major challenges of building an end-to-end system in a real-world industrial setting go beyond the design of machine learning algorithms. One of the major challenges is the design of an end-to-end data management solution that is able to efficiently store and process large amounts of heterogeneous data streams resulting from a variety of physical machines. In this paper we present the design of an end-to-end Big Data architecture that enables intelligent maintenance in a real-world industrial setting. In particular, we will discuss various physical design choices for optimizing high-dimensional queries, such as partitioning and Z-ordering, that serve as the basis for health analytics. Finally, we describe a concrete fault detection use case with two different health monitoring algorithms based on machine learning and classical statistics and discuss their advantages and disadvantages. The paper covers some of the most important aspects of the practical implementation of such an end-to-end solution and demonstrates the challenges and their mitigation for the specific application of laser cutting machines.

Keywords: Prognostics and health management, Intelligent maintenance, Big data architecture, Heterogeneous data integration, Stream processing, Query processing, Machine learning

Source: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00340-7>

ZHAW Research on Big Data



The screenshot shows the Springer Open website interface. At the top, there is a search bar and the Springer Open logo. Below the header, the article title "Scalable architecture for Big Data financial analytics: user-defined functions vs. SQL" is prominently displayed. The authors listed are Kurt Stockinger, Nils Bundi, Jonas Heitz, and Wolfgang Breymann. The article is from the "Journal of Big Data", volume 6, article number 46, published in 2019. It has 2954 accesses and 4 citations. The abstract section follows, starting with "Large financial organizations have hundreds of millions of financial contracts on their balance sheets. Moreover, highly volatile financial markets and heterogeneous data sets within and across banks world-wide make near real-time financial analytics very challenging and their handling thus requires cutting edge financial algorithms. However, due to a lack of data modeling standards, current financial risk algorithms are typically inconsistent and non-scalable. In this paper, we present a novel implementation of a real-world use case for performing large-scale financial analytics leveraging Big Data technology. We first provide detailed background information on the financial underpinnings of our framework along with the major financial calculations. Afterwards we analyze the performance of different parallel implementations in Apache Spark based on existing computation kernels that apply the ACTUS data and algorithmic standard for financial contract modeling. The major contribution is a detailed discussion of the design trade-offs between applying user-defined functions on existing computation kernels vs. partially re-writing the kernel in SQL and thus taking advantage of the underlying SQL query optimizer. Our performance evaluation demonstrates almost linear scalability for the best design choice."

Springer Open

Search

Journal of Big Data

About Articles Submission Guidelines

Research | Open Access | Published: 06 June 2019

Scalable architecture for Big Data financial analytics: user-defined functions vs. SQL

Kurt Stockinger, Nils Bundi, Jonas Heitz & Wolfgang Breymann

Journal of Big Data 6, Article number: 46 (2019) | Cite this article

2954 Accesses | 4 Citations | Metrics

Abstract

Large financial organizations have hundreds of millions of financial contracts on their balance sheets. Moreover, highly volatile financial markets and heterogeneous data sets within and across banks world-wide make near real-time financial analytics very challenging and their handling thus requires cutting edge financial algorithms. However, due to a lack of data modeling standards, current financial risk algorithms are typically inconsistent and non-scalable. In this paper, we present a novel implementation of a real-world use case for performing large-scale financial analytics leveraging Big Data technology. We first provide detailed background information on the financial underpinnings of our framework along with the major financial calculations. Afterwards we analyze the performance of different parallel implementations in Apache Spark based on existing computation kernels that apply the ACTUS data and algorithmic standard for financial contract modeling. The major contribution is a detailed discussion of the design trade-offs between applying user-defined functions on existing computation kernels vs. partially re-writing the kernel in SQL and thus taking advantage of the underlying SQL query optimizer. Our performance evaluation demonstrates almost linear scalability for the best design choice.

Source: <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0209-0>

Conclusions

- Good to be aware of **memory management** to understand potential performance problems
- Understand the difference between **jobs, stages and tasks**
- **Analyze** the performance of jobs



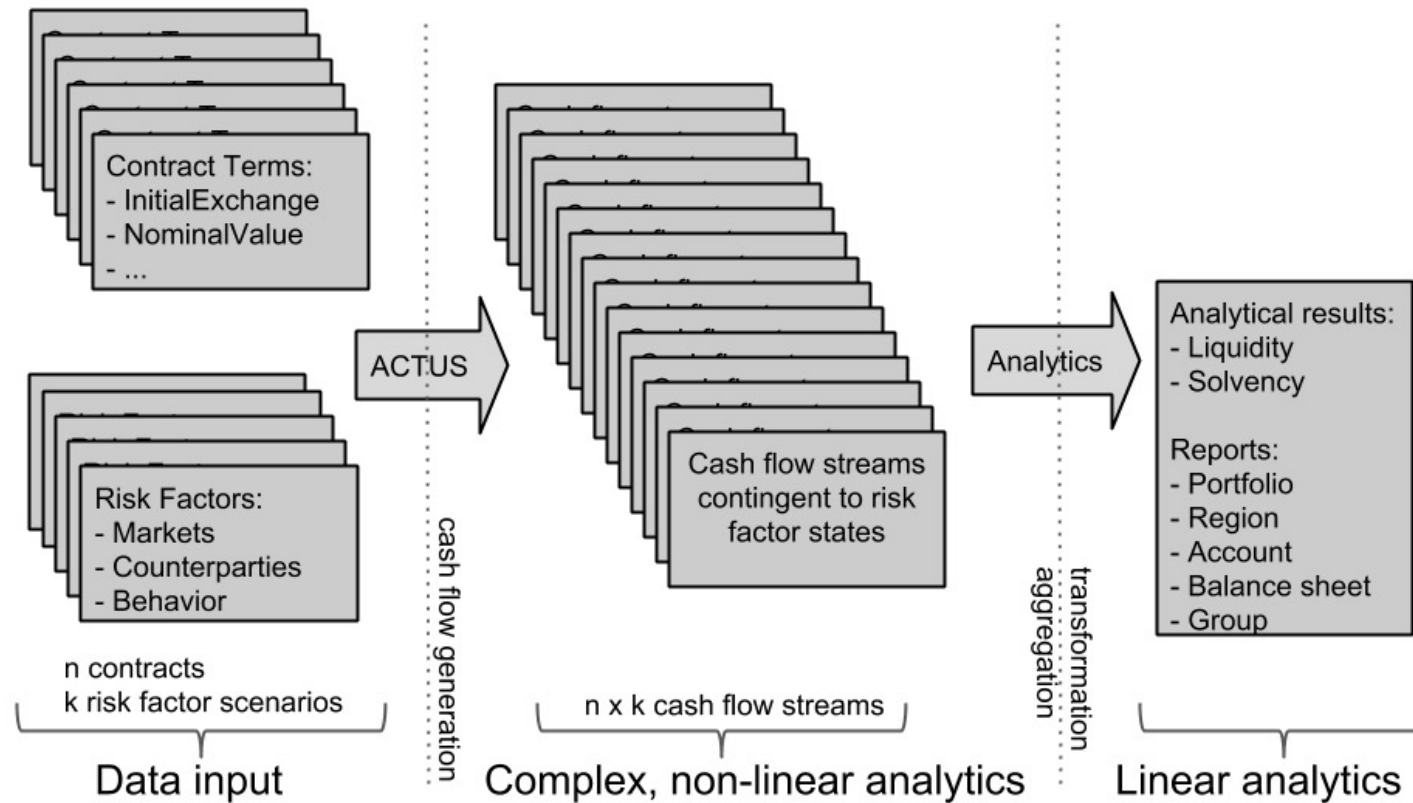
Applications

Large-Scale Data-Driven Financial Risk Modeling using Big Data Technology

- **Financial crises** of 2008 demonstrated that neither banks nor regulators could assess financial risks appropriately
- **No standard** for describing contractual cash flow obligations in financial contracts – the basis for financial risk assessment
- **ACTUS (Algorithmic Contract Type Unification Standard)**
 - Standardization of financial contracts and algorithms
 - Input:
 - Contract data
 - Risk factor data
 - Output:
 - Cash flows of financial contracts
 - Financial analytics
- Challenge: How to **scale risk calculations** for major international banks?

Data Flows in ACTUS

ALGORITHMIC CONTRACT TYPES UNIFIED STANDARDS

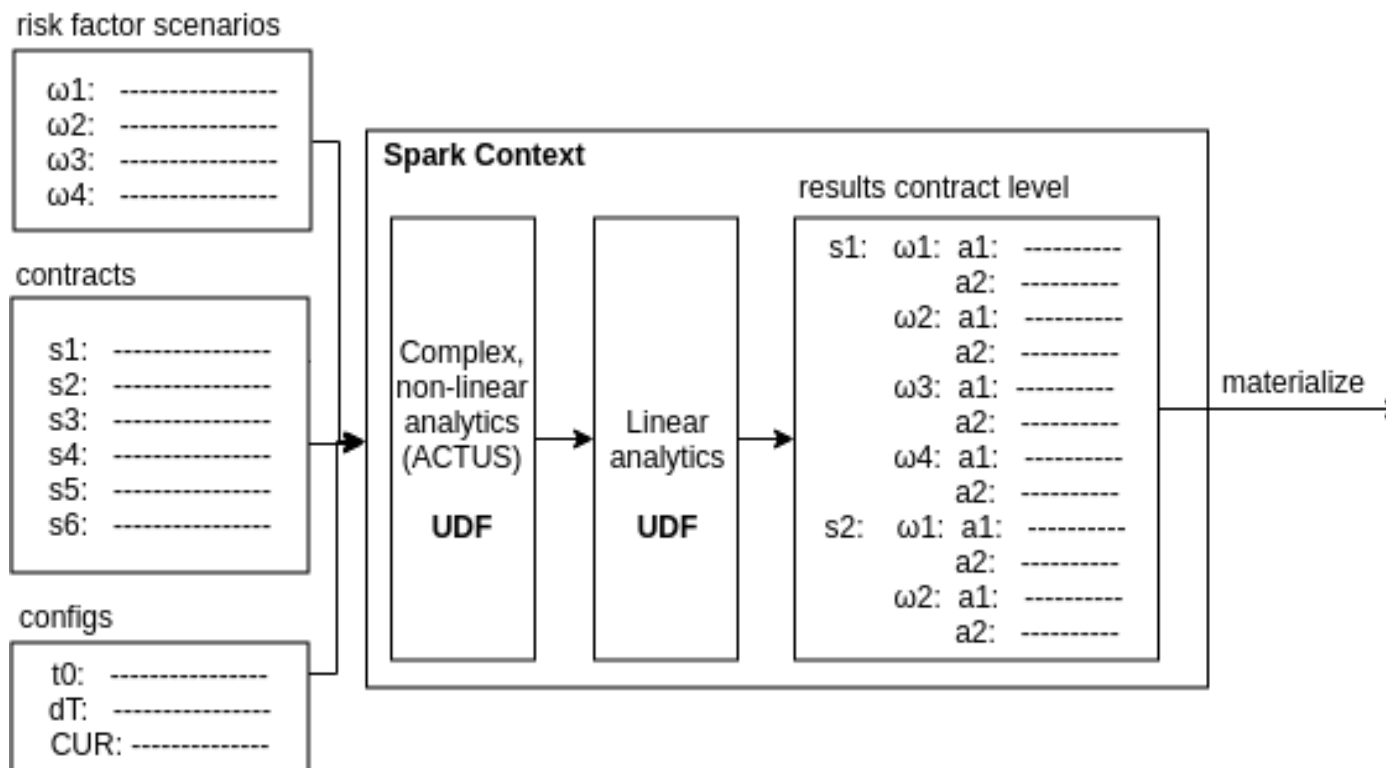


Financial Analytics

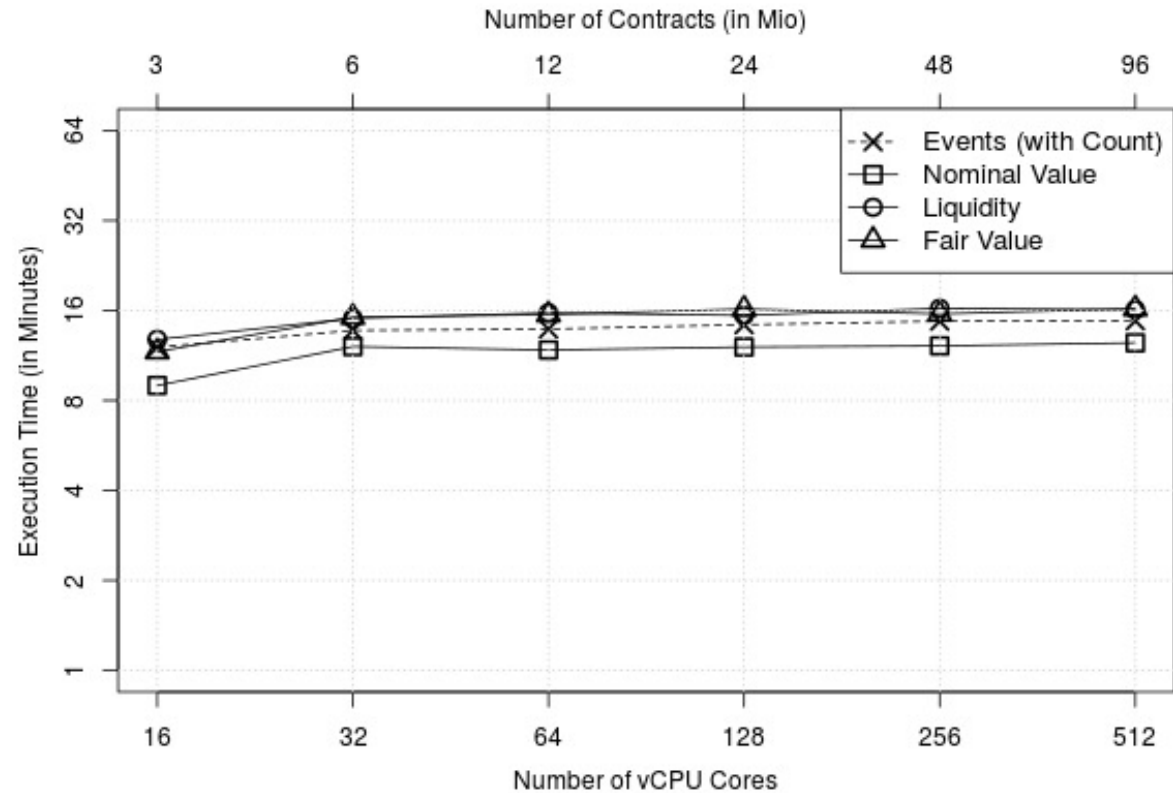
- **Nominal value:**
 - Measures the (current) notional outstanding of, e.g., a loan
 - Provides basis for exposure calculations in credit- risk departments
- **Fair value:**
 - Quantifies the price of a contract that could be realized in a market transaction at current market conditions
- **Liquidity:**
 - Expected net liquidity flows over some future time periods

Basic measurements necessary for analyzing and managing different types of financial risks

System Architecture



Performance Analysis



Software: Apache Spark 2.3 running on AWS; Hardware: Up to 512 vCPU cores, 960 GB distributed RAM

Kurt Stockinger, Nils Bundi, Jonas Heitz, Wolfgang Breymann: Scalable Architecture for Big Data Financial Analytics: User-Defined Functions vs. SQL. *Journal of Big Data* 6: 46 (2019)

Conclusions of Financial Analytics

- We could **parallelize** an existing financial kernel with Apache Spark
- We demonstrated that financial analytics of up to 96 million contracts on 512 vCPU cores show **almost linear scalability**
- Large global bank has on the order of 10 million financial contracts
- .
- We can **calculate key risk analytics of about 10 large banks** in only about **16 minutes**
- **More information:**
 - Kurt Stockinger, Nils Bundi, Jonas Heitz, Wolfgang Breymann: **Scalable Architecture for Big Data Financial Analytics: User-Defined Functions vs. SQL**. *J. Big Data* 6: 46 (2019)

Spark AI Summits

- <https://databricks.com/sparkaisummit/north-america>



- <https://databricks.com/sparkaisummit>





More Applications

Which Data Sources are Interesting?

- Social media data:
 - Twitter
 - Facebook
 - LinkedIn
 - Xing
- Internet:
 - E-commerce platforms (search and buying behavior)
 - Information about companies
- Goal:
 - Better understand customers to build customer profile

Facebook

- World's largest social network with more than **2 billion users**
 - Customer profile based on likes and posts
- Facebook has **advantages of TV and Google**:
 - Broad coverage (TV)
 - Customizable (Google)
- Targeted **advertizing**:
 - Facebook is a fantastic tool for doing personalized marketing“
 - „We can tell you within the first 15 minutes of a post whether it's a good post or a bad post“

Source: „How Facebook Sold You Krill Oil“, New York Times, 2. August 2014

Example: Scrap Rate Reduction

Manufacturing

Business Objective:

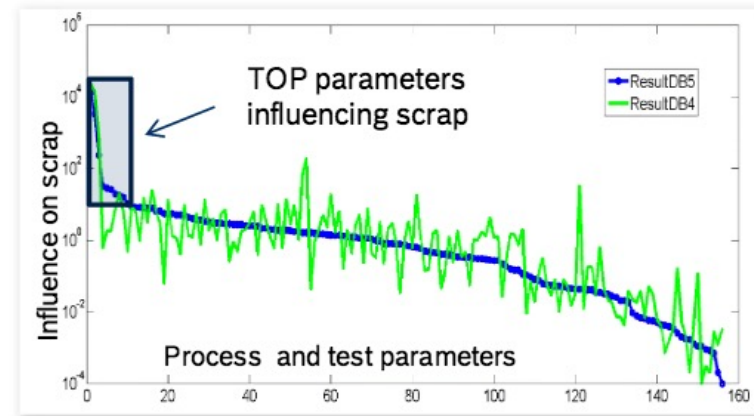
Reduce the scrap rate of a Pump Module

Solution

65% reduction in scrap rate by identifying top influencing parameters in root cause analysis

Business Benefits

Cost savings of 1.5m USD over 6 months



Source: ISC Cloud & Big Data 2015, Frankfurt

Other Popular Examples

- Purchase recommendations: Bspw. «Netflix Prize» (2009)
Quelle: <http://www.netflixprize.com/>
- Churn prediction: Telcos, Insurance companies, ...
Quelle: <http://engineering.zhaw.ch/de/engineering/forschung/publikationen-t.html?pi=206434&qu=96884>
- Basket analysis: e.g. beer and diapers, books, ...
Quelle: <http://web.onetel.net.uk/~hibou/Beer%20and%20Nappies.html>
- Risk and fraud analysis: e.g. life insurances, credit card usage,...
Quelle: Eric Siegel, «Predictive Analytics», John Wiley & Sons, 2013



...

Even More Examples

Was?	Wer?
Grippe	Google Flu Trends: Sagt anhand Suchtrends Zunahme von Grippefällen 7-10 Tage vor der zuständigen staatlichen Seuchenstelle voraus
Systemfehler	Argonne National Laboratory: Vorhersage von Rissen im Kühlsystem von AKWs BNSF Railway: Vorhersage defekter Gleise 85% → Verhindert schwere Zugunglücke TTX: Vorhersage von Fehlern in hunderttausenden von Eisenbahnradern zur Abschätzung des jährlichen Wartungsaufwands (98.5% Genauigkeit)
Ölfördermenge	National Iranian South Oil Company: Vorhersage der Tagesproduktion an Öl
Flugverspätung	Continental Airlines: Vorhersage der Verspätung von Flügen via Radardaten (Einsparung mehrerer 10 Millionen Dollar)
Fahrer- ablenkung	Ford Motor Company, TWT GmbH: Grad der Fahrerablenkung für Autolenker (86% Genauigkeit für Ford-System)

Quellen:

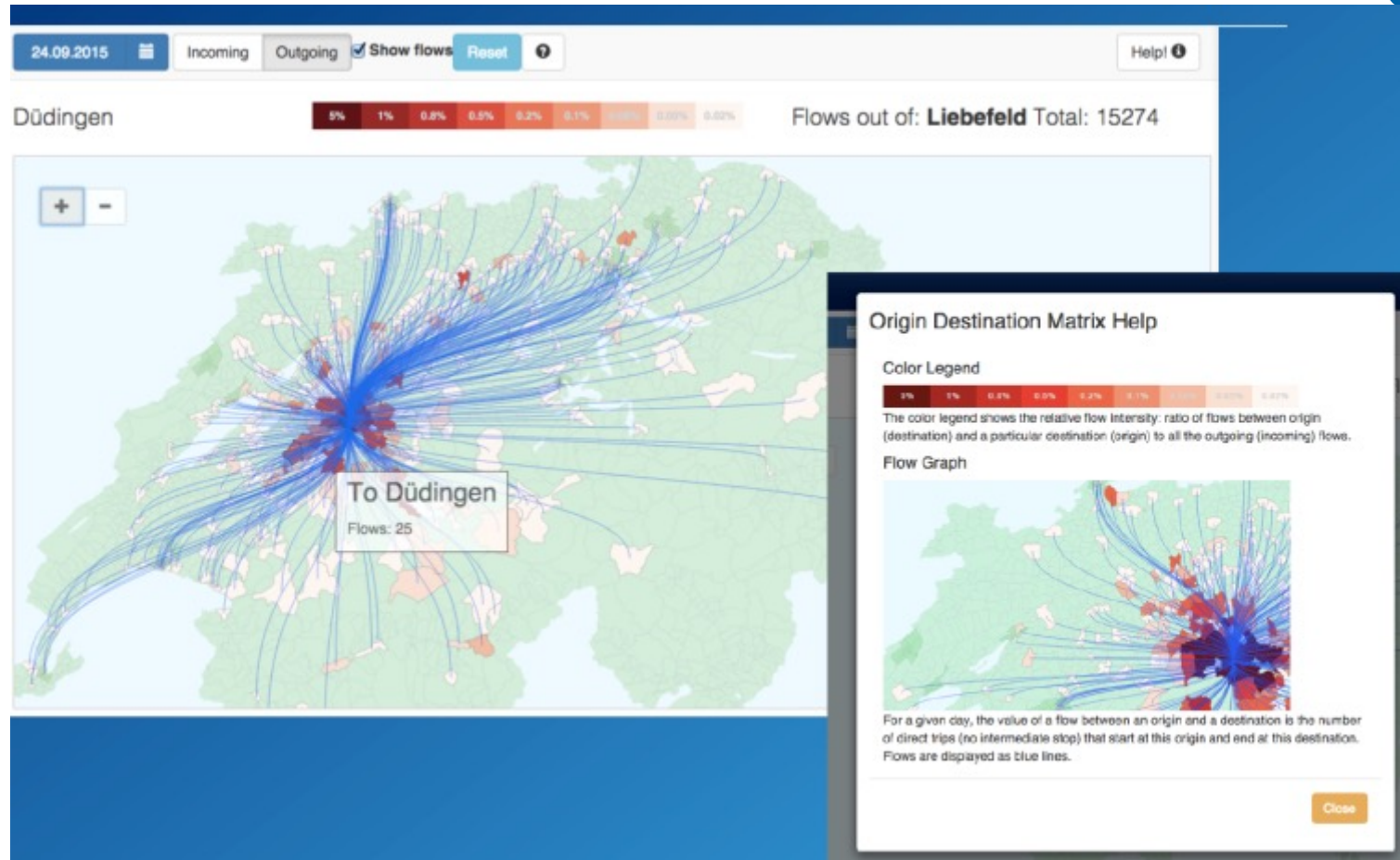
- Eric Siegel, «Predictive Analytics», John Wiley & Sons, 2013
- Thilo Stadelmann et al., «FABELHAFT - Fahrerablenkung: Entwicklung eines Meta-Fahrerassistenzsystems durch Echtzeit-Audioklassifikation», VDI Wissensforum, 2012

Examples in Switzerland

- Banks
 - Fraud analysis
 - Were at forefront in early days of data mining
- Telcos
 - Evaluation of mobile phone usage
- Retail:
 - Shopping behavior
- Pharma
 - Personalized medicine
- Industry:
 - Predictive maintenance



Smart Cities: Example Swisscom



Source: ISC Cloud & Big Data 2015, Frankfurt

The Future: Internet of Things and Robots

- Interconnection of various devices and machines
 - Cars
 - Washing machines
 - Lawn mowers
 - Thermostats
 - Service robots
- Machines are connected via internet



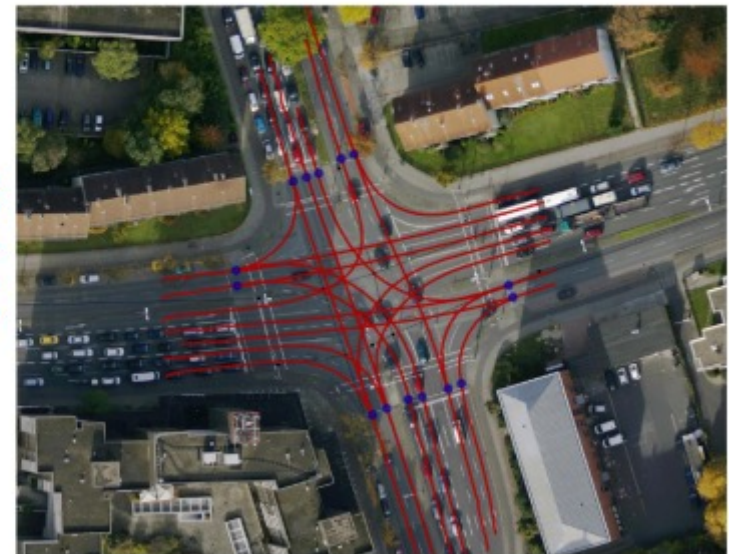
Self-Driving Cars: Example Audi

TECHNOLOGY – TODAY



LEARNING OF AN INTERSECTION GEOMETRY (4)

Step 4
Calculation of
the
intersection
geometry



Source: ISC Cloud & Big Data 2015, Frankfurt

Final Example: IBM Research

- Integration of various data sources
- Social analytics (ca. 5.5 min):

https://www.youtube.com/watch?v=3Qe_Med9Jyw

