"Big Data" Management and Apache Flink

Volker Markl

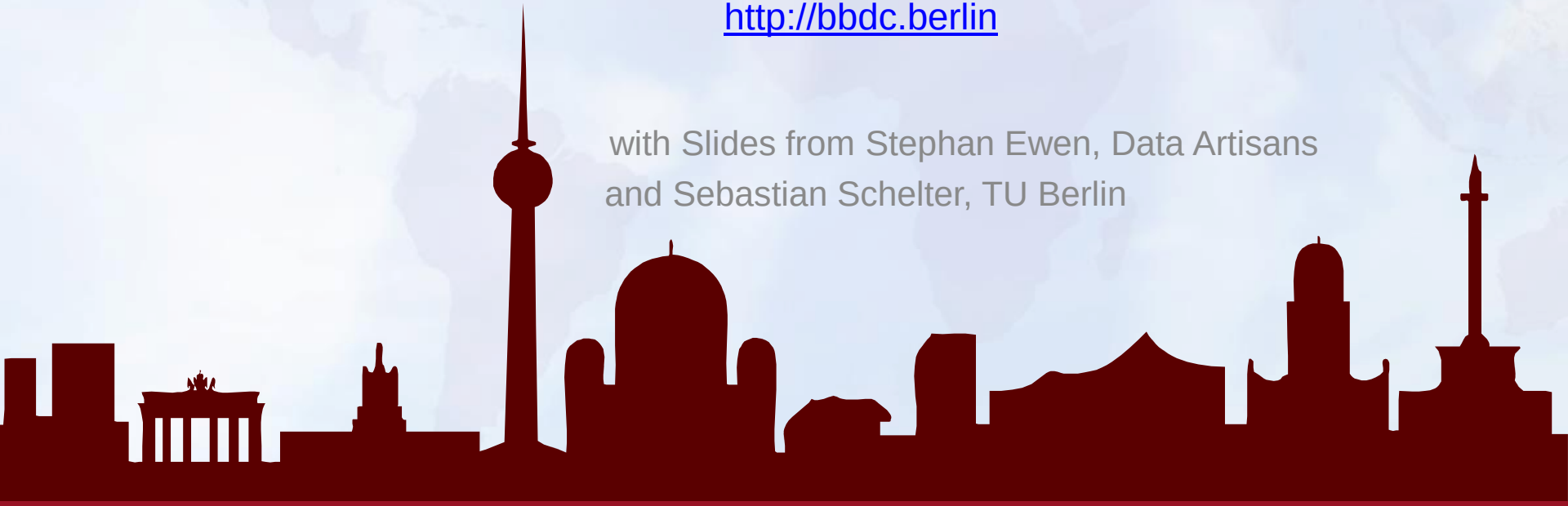
http://www.user.tu-berlin.de/marklv/

http://www.dima.tu-berlin.de

http://www.dfki.de/web/forschung/iam

http://bbdc.berlin

with Slides from Stephan Ewen, Data Artisans
and Sebastian Schelter, TU Berlin



About the Speaker

- Master‘s in 1996 (Exception Handling)
- PhD in 1999 (Multidimensional Indexing)
- IBM Research, Almaden, 2001 to 2008 (Self-Tuning Query Optimization, II)
- Full Professor at TU Berlin since 2008, Adjunct Prof at U Toronto since 2012
- Director of the Research Group „Intelligent Analysis of Massive Data“ at the German Research Center for Artificial Intelligence (DFKI)
- Speaker of the Berlin Big Data Center (BBDC)

- Research in parallel data processing, database programming languages for big data/data science, modern hardware, information marketplaces
- 19 patents, 2 more invention disclosures under evaluation
- Inventions integrated into several products, e.g., by IBM, Oracle, MSFT, SAP
- Started Stratosphere (now Apache Flink) in 2008
- Co-founded / advising three 3 Startups (Parstream - 2011, Internet Memory Research/Mignify - 2012, DataArtisans - 2014)
- Secretary of the Very Large Databases (VLDB) Endowment

Data & Analysis: Increasingly Complex!

scalability

DM

ML

algorithms

Data		Analysis	
data volume too large	***V**olume*	Reporting	aggregation, selection
data rate too fast	***V**elocity*	Ad-Hoc Queries	SQL, XQuery
data too heterogeneous	***V**ariety*	ETL/ELT	MapReduce
data too uncertain	***V**eracity*	Data Mining	MATLAB, R, Python
novel applications	***V**alue*	Predictive/Prescriptive	MATLAB, R, Python

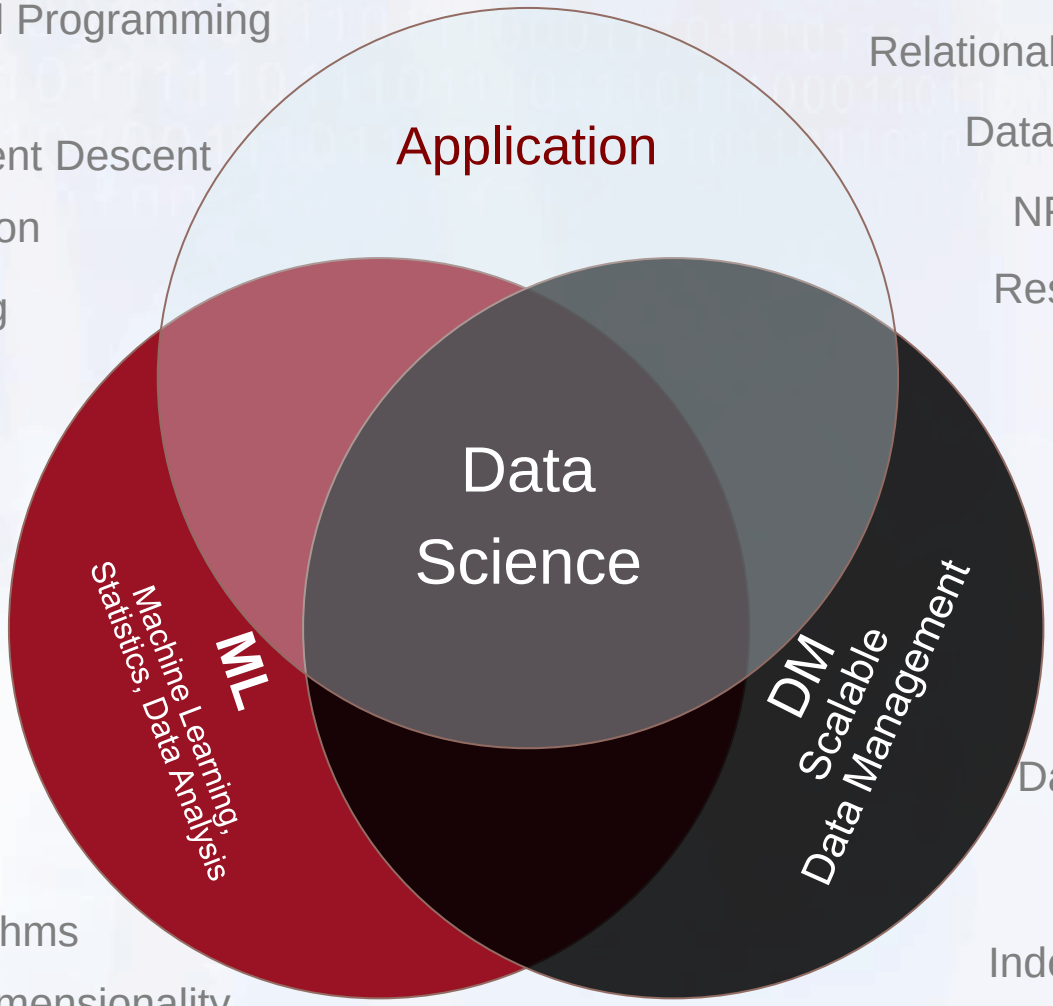
scalability

DM

ML

algorithms

“Data Scientist” – “Jack of All Trades!”



Domain Expertise (e.g., Industry 4.0, Medicine, Physics, Engineering, Energy, Logistics)

Mathematical Programming

Linear Algebra

Stochastic Gradient Descent

Error Estimation

Active Sampling

Regression

Monte Carlo

Statistics

Sketches

Hashing

Convergence

Decoupling

Iterative Algorithms

Curse of Dimensionality

Relational Algebra / SQL

Data Warehouse/OLAP

NF^2 /XQuery

Resource Management

Hardware Adaptation

Fault Tolerance

Memory Management

Parallelization

Scalability

Memory Hierarchy

Data Analysis Language Compiler

Query Optimization

Indexing

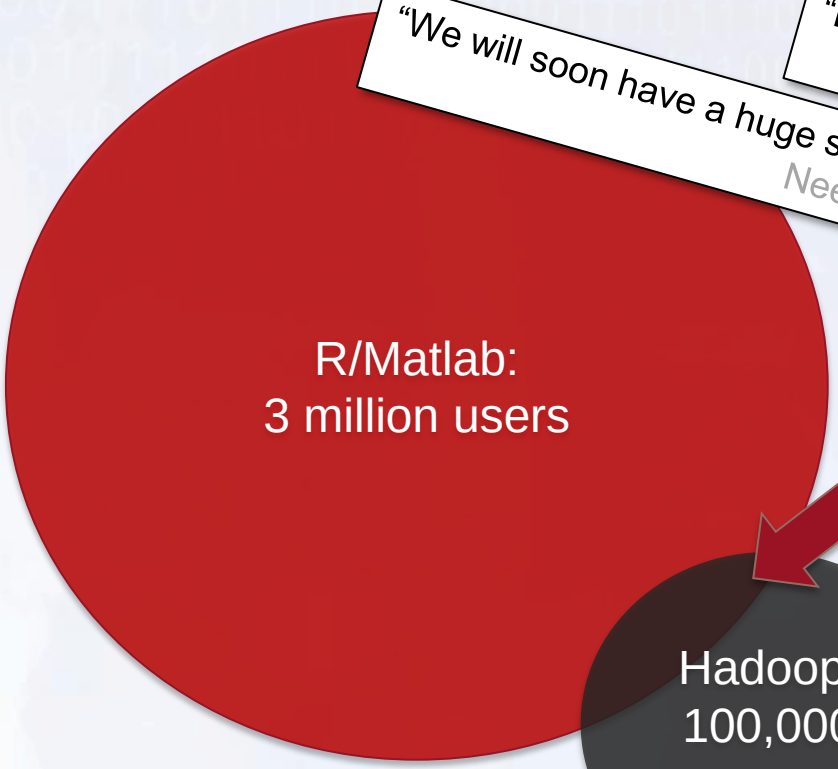
Data Flow

Control Flow

Real-Time

Big Data Analytics Requires Systems Programming

Data Analysis
Statistics
Algebra
Optimization
Machine Learning
NLP
Signal Processing
Image Analysis
Audio-,Video Analysis
Information Integration
Information Extraction
Data Value Chain
Data Analysis Process
Predictive Analytics



"We will soon have a huge skills shortage for data-related jobs."
Neelie Kroes (ICT 2013, Nov. 7, Vilnius)

"Big Data's Big Problem: Little Talent"
Wall Street Journal

People with Big Data Analytics Skills

Indexing
Parallelization
Communication
Memory Management
Query Optimization
Efficient Algorithms
Resource Management
Fault Tolerance
Numerical Stability

Big Data is now where database systems were in the 70s (prior to relational algebra, query optimization and a SQL-standard)! We address the complexities of big data science by merely teaching it.

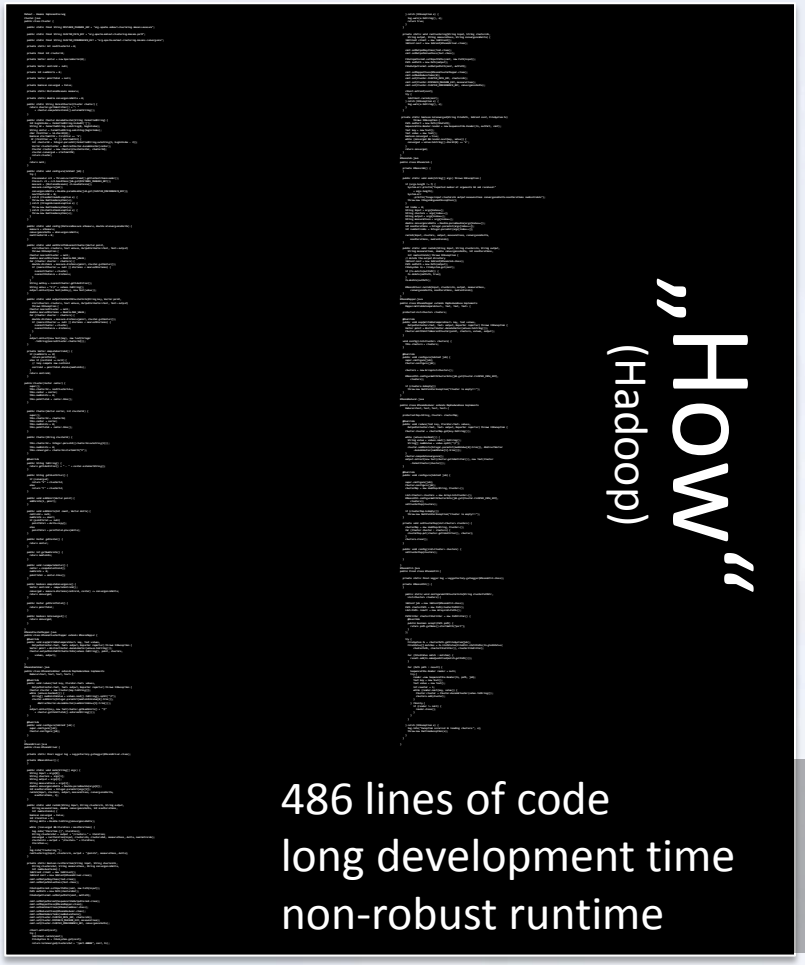
Novel Systems and declarative languages to the rescue!

„What“, not „how“ Example: k-Means Clustering

„What“
(Apache Flink)
(Scala frontend)

65 lines of code
short development time
robust runtime

Declarative data analysis program with automatic optimization, parallelization and hardware adaption



„How“
(Hadoop)

486 lines of code
long development time
non-robust runtime

Hand-optimized code
(data-, load- and system dependent)

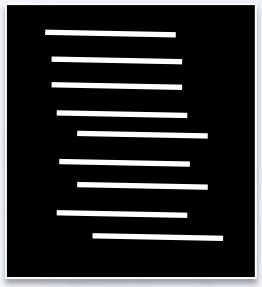
X = Big Data Analytics – System Programming! („What“, not „How“)

Data Analyst

Description of „What“?
(declarative specification)
New Technology

Larger human base of „data scientists“
Reduction of „human“ latencies
Cost reduction

Machine



Description of „How“?
(State of the art in scalable data analysis)
Hadoop, MPI

Deep Analysis of „Big Data“ is Key!

Deep Analytics

Simple Analysis

SQL

Small Data

Big Data (3V)

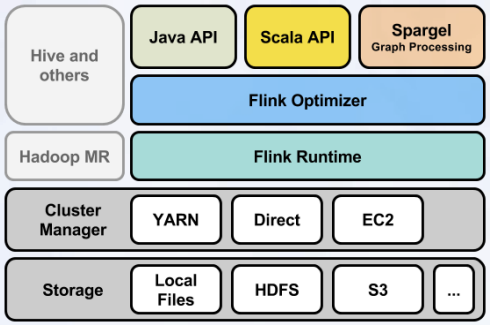
Introducing Stratosphere and Apache Flink

- Project started under the name “Stratosphere” late 2008 as a DFG funded research unit, lead by TU Berlin, in collaboration with HU Berlin, and the Hasso Plattner Institute Potsdam.

Big Data looks tiny from Stratosphere

More information: http://stratosphere.eu

- Apache Open Source Incubation since April 2014,
  Apache Top Level Project since December 2014



- Fast growing community of open source users and developers in Europe and worldwide, in academia (e.g., SICS/KTH, INRIA, ELTE, U Modena, PoliMi) and companies (e.g., Researchgate, Spotify, Amadeus, Huawei)

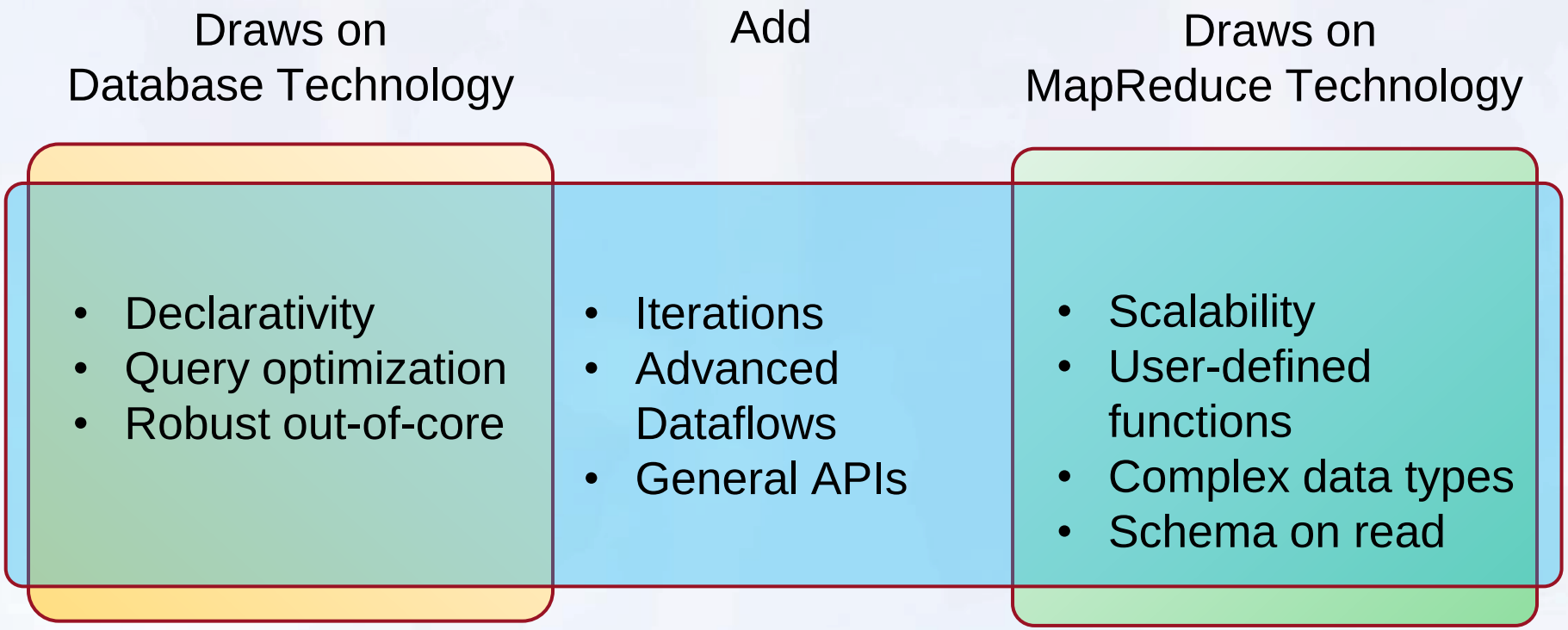
More information: http://flink.apache.org

Overly simplistic model of Map/Reduce results in complex code and prevents automatic optimization

- Map/Reduce is record at a time/group at a time single input; multiple inputs require tricks like input tagging or distributed caches (e.g., when implementing joins) instead of tested and tried methods of parallel databases (e.g., parallel hybrid hash joins, etc.)
- Lack of knowledge about first order function prevents exploitation of commutativity/associativity/distributivity (e.g., combiners vs. eager/lazy aggregation)
- Strict map/reduce pattern prevents optimization of complex analysis workflows (unnecessarily many writes of stages locally or in HDFS; network I/O) instead of projection/selection pushdown or join reordering
- Key/Values model vs. Tuple models looses partitioning, interesting orders, etc., introducing redundant unnecessary work
- No support for iterative programs

Effectively: Assembly level systems programming as opposed to problem oriented programming in SQL

Stratosphere (Flink): General Purpose Programming + Database Execution

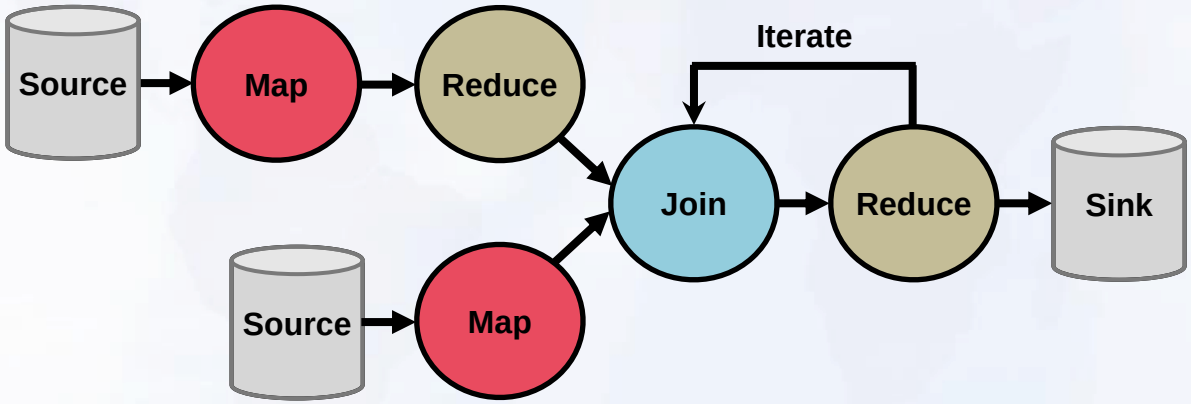


Alexandrov et al.: “The Stratosphere Platform for Big Data Analytics,” VLDB Journal 5/2014

12

Rich Set of Operators / PACT Programming Model

Map	Iterate	Project
Reduce	Delta Iterate	Aggregate
Join	Filter	Distinct
CoGroup	FlatMap	Vertex Update
Union	GroupReduce	Accumulators

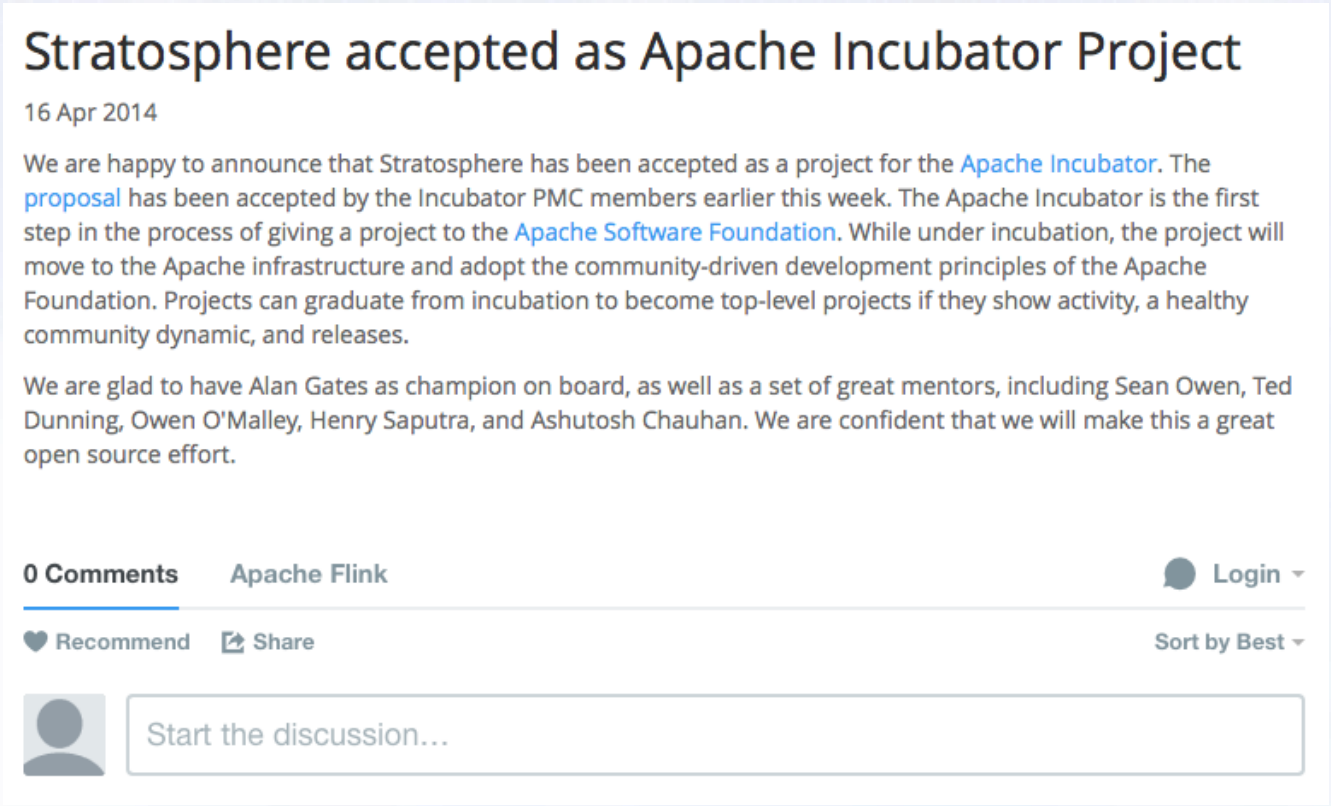


Alexandrov et al.: "The Stratosphere Platform for Big Data Analytics," VLDB Journal 5/2014
Battré et al.: „Nephele/PACTs: a programming model and execution framework for web-scale analytical processing," SoCC 2010: 119-130

LOOKING BACK 1.5 YEARS

The Stratosphere platform for big data analytics. VLDB J. 23(6): 939-964 (2014)

April 16, 2014



Stratosphere accepted as Apache Incubator Project

16 Apr 2014

We are happy to announce that Stratosphere has been accepted as a project for the Apache Incubator. The proposal has been accepted by the Incubator PMC members earlier this week. The Apache Incubator is the first step in the process of giving a project to the Apache Software Foundation. While under incubation, the project will move to the Apache infrastructure and adopt the community-driven development principles of the Apache Foundation. Projects can graduate from incubation to become top-level projects if they show activity, a healthy community dynamic, and releases.

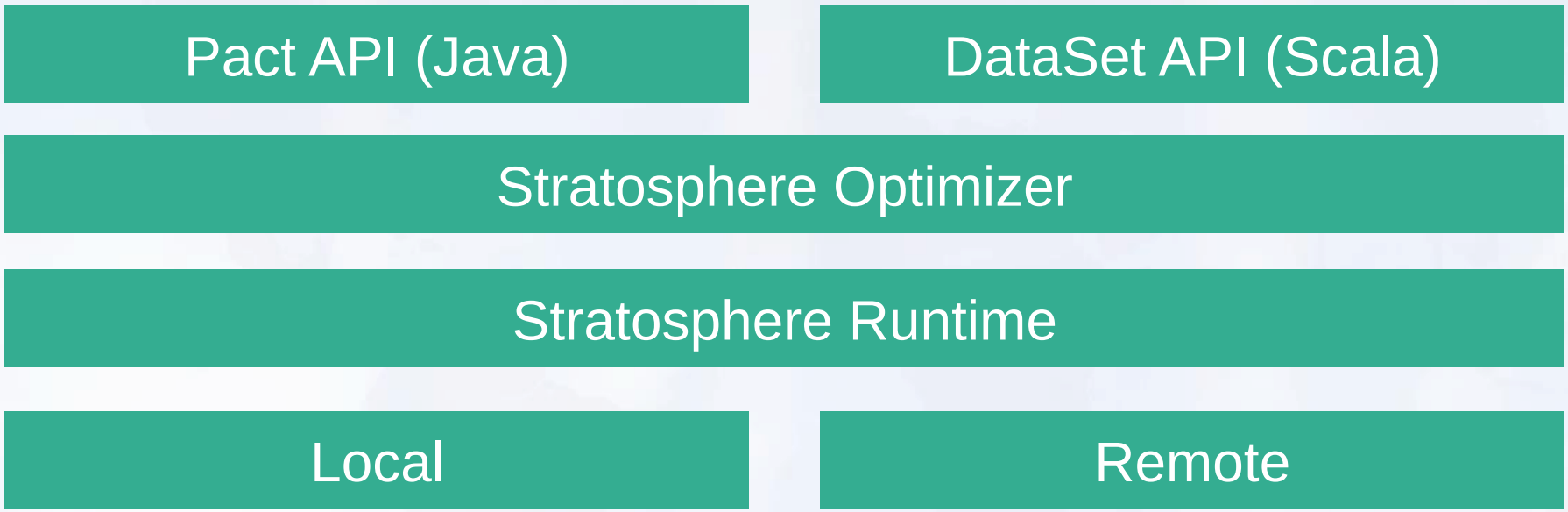
We are glad to have Alan Gates as champion on board, as well as a set of great mentors, including Sean Owen, Ted Dunning, Owen O'Malley, Henry Saputra, and Ashutosh Chauhan. We are confident that we will make this a great open source effort.

0 Comments Apache Flink Login

Recommend Share Sort by Best

Start the discussion…

Stratosphere 0.4



Batch processing on a pipelining engine, with iterations …

MORE RECENTLY …

Breaking the Chains: On Declarative Data Analysis and Data Independence in the Big Data Era. PVLDB 7(13): 1730-1733 (2014)

What is Apache Flink?

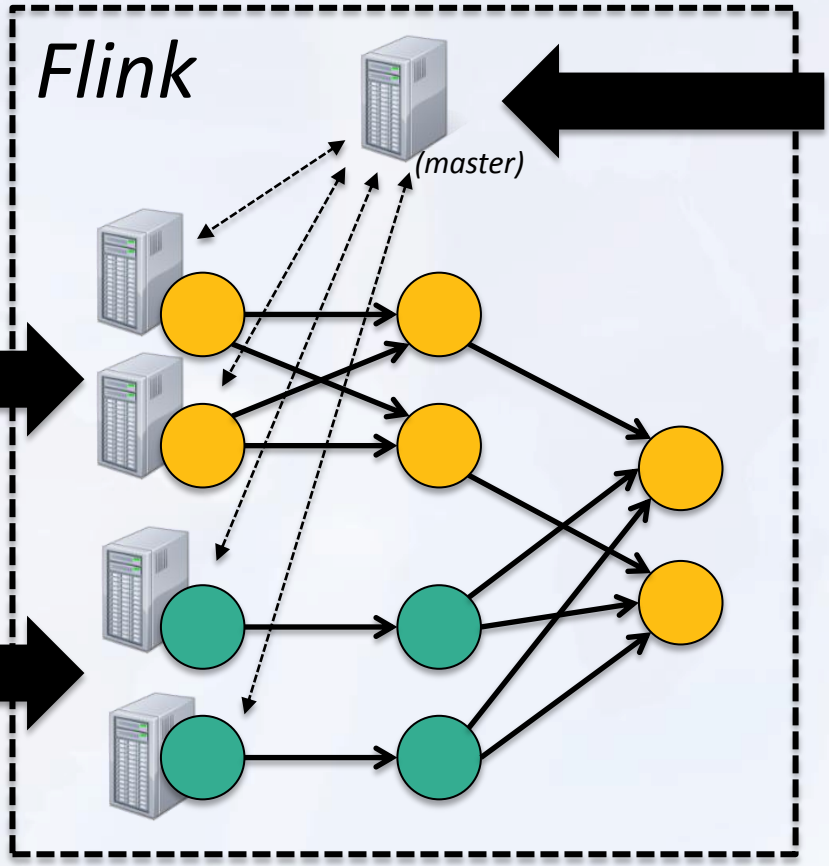
Real-time data streams

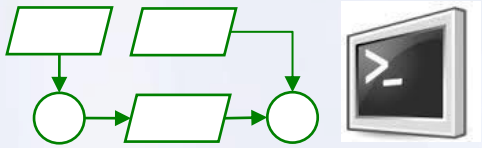
Event logs

Kafka, RabbitMQ, ...

Historic data

HDFS, JDBC, ...

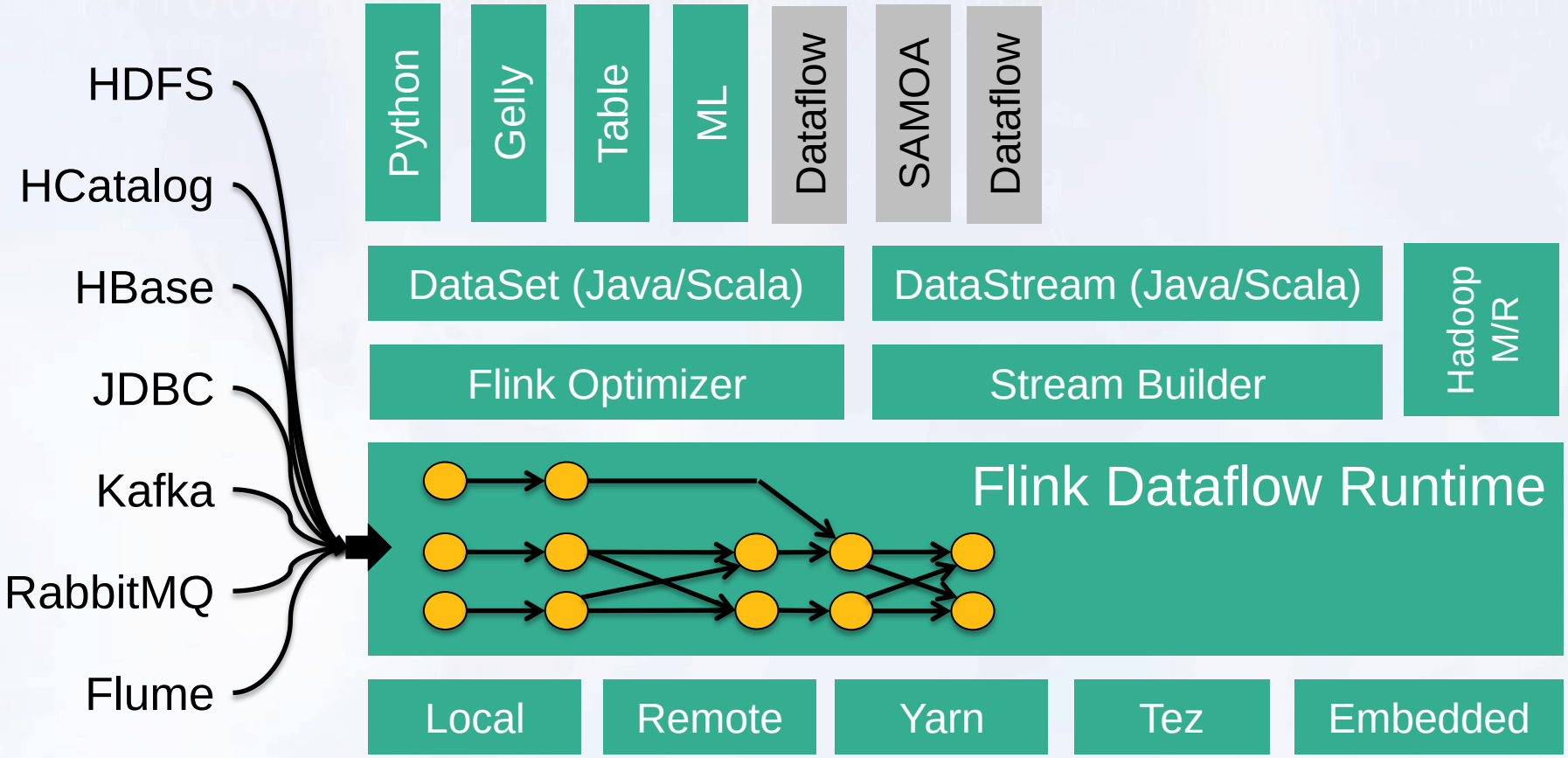




ETL, Graphs,
Machine Learning
Relational, …

Low latency,
windowing,
aggregations, ...

What is Apache Flink?



Batch / Streaming APIs

```
case class Word (word: String, frequency: Int)
```

DataSet API (batch):

```
val lines: DataSet[String] = env.readTextFile(...)

lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .groupBy("word").sum("frequency")
     .print()
```

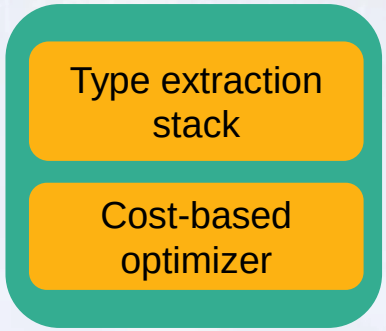
DataStream API (streaming):

```
val lines: DataStream[String] = env.fromSocketStream(...)

lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .window(Count.of(1000)).every(Count.of(100))
     .groupBy("word").sum("frequency")
     .print()
```

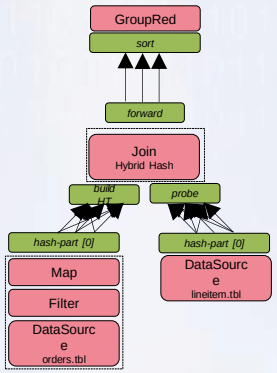
Technology inside Flink

```
case class Path (from: Long, to:
Long)
val tc = edges.iterate(10) {
  paths: DataSet[Path] =>
    val next = paths
      .join(edges)
      .where("to")
      .equalTo("from") {
        (path, edge) =>
          Path(path.from, edge.to)
      }
      .union(paths)
      .distinct()
    next
  }
```

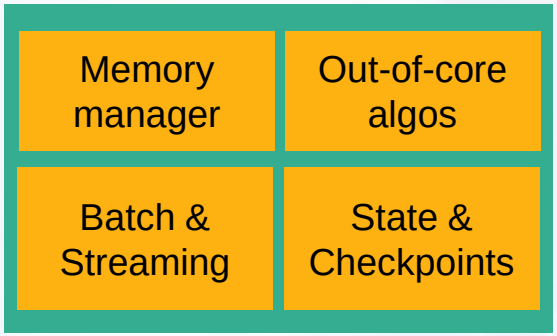
Program

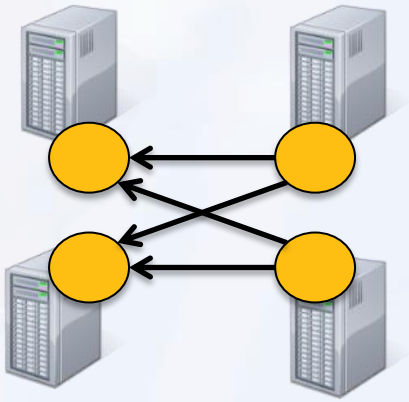


Pre-flight (Client)



Dataflow Graph

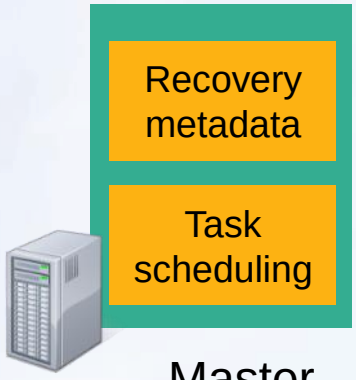




Workers

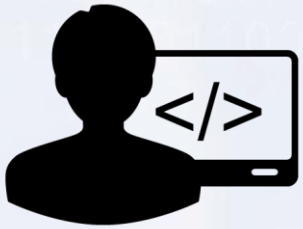
deploy operators

track intermediate results



Master

Effect of optimization



Hash vs. Sort
Partition vs. Broadcast
Caching
Reusing partition/sort

Execution Plan A

Execution Plan B

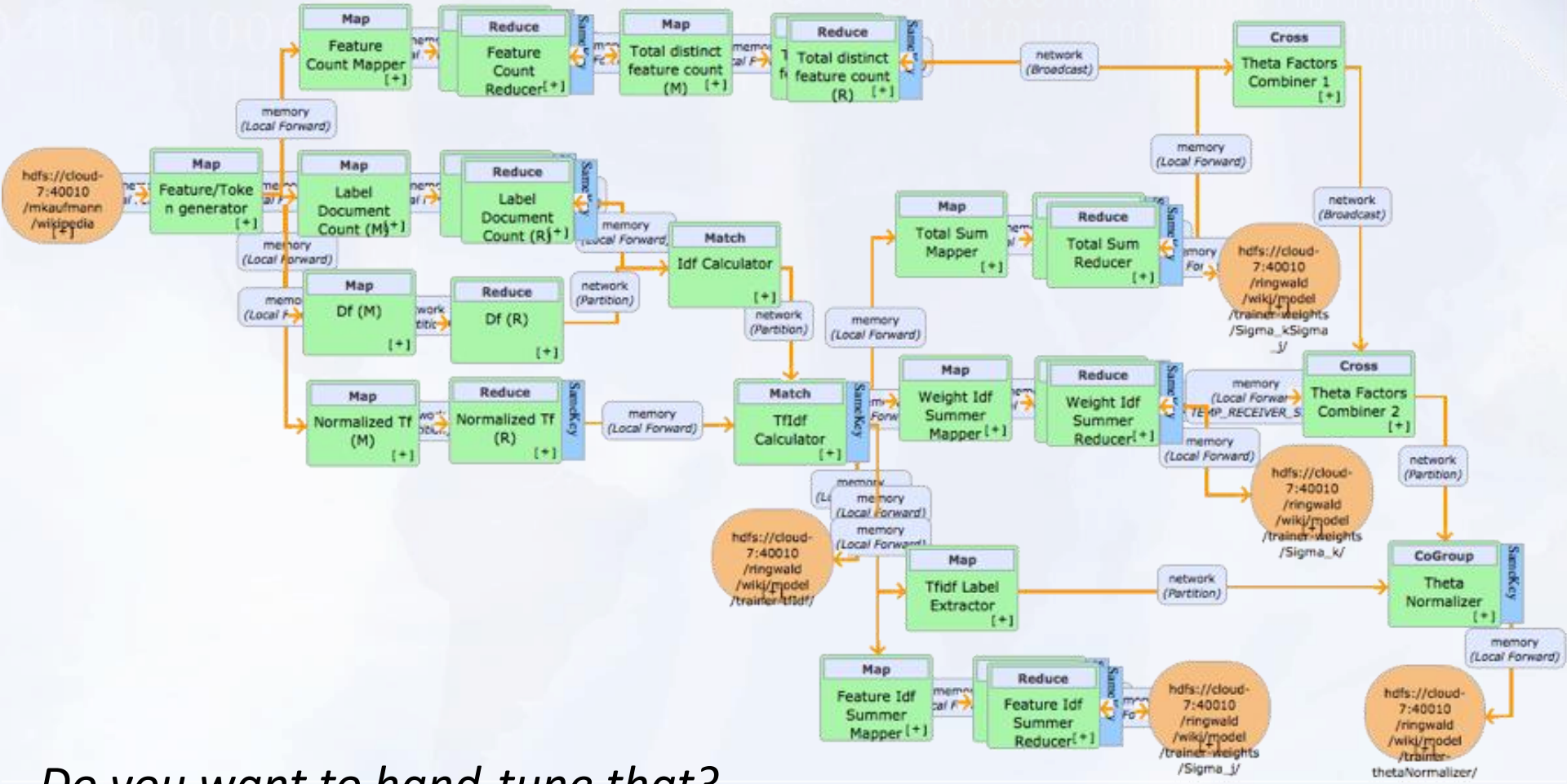
Execution Plan C

Run on a sample on the laptop

Run on large files on the cluster

Run a month later after the data evolved

Why optimization ?

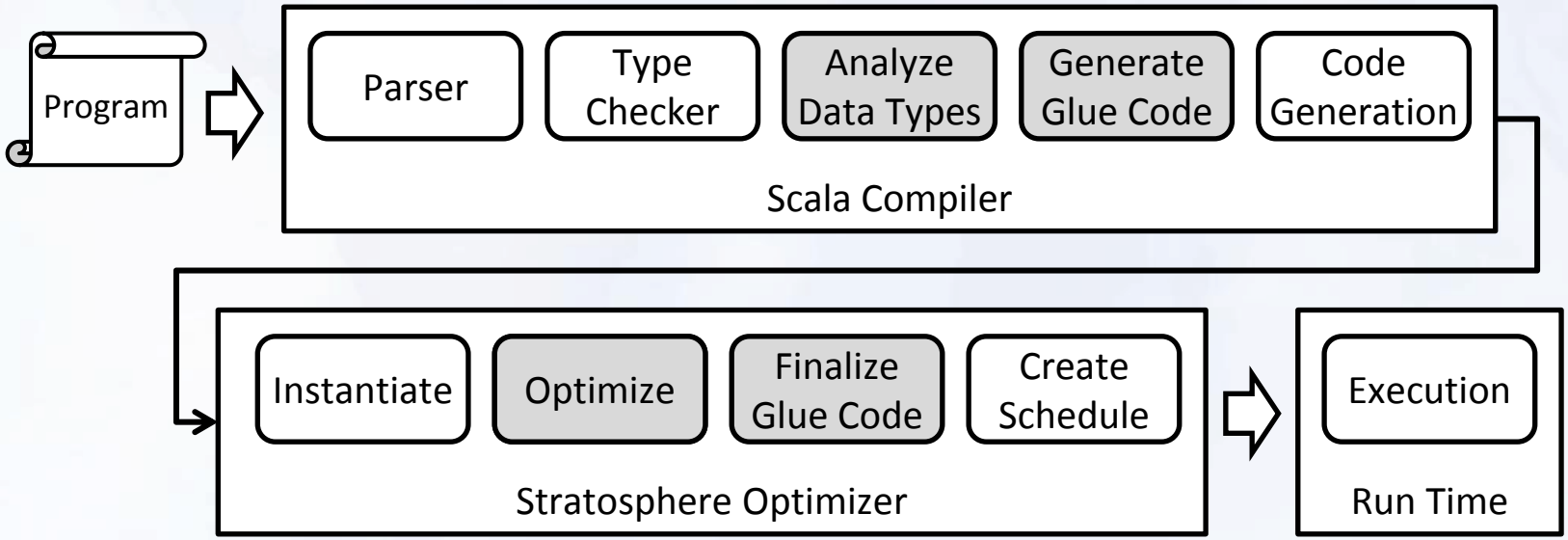


Do you want to hand-tune that?

Optimizing Programs

- Program optimization happens in two phases

1. Data type and function code analysis inside the Scala Compiler
2. Relational-style optimization of the data flow

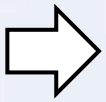


Type Analysis/Code Gen

- Types and Key Selectors are mapped to flat schema
- Generated code for interaction with runtime

Primitive Types, Arrays, Lists

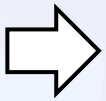
```
Int, Double,
Array[String],
...
```



Single Value

Tuples / Classes

```
(a: Int, b: Int, c: String)
class T(x: Int, y: Long)
```

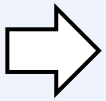


Tuples

```
(a: Int, b: Int, c: String)
(x: Int, y: Long)
```

Nested Types

```
class T(x: Int, y: Long)
class R(id: String, value: T)
```

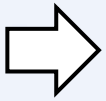


Recursively flattened

```
(x: Int, y: Long)
(id:String, x:Int, y:Long)
```

recursive types

```
class Node(id: Int, left: Node,
                    right: Node)
```



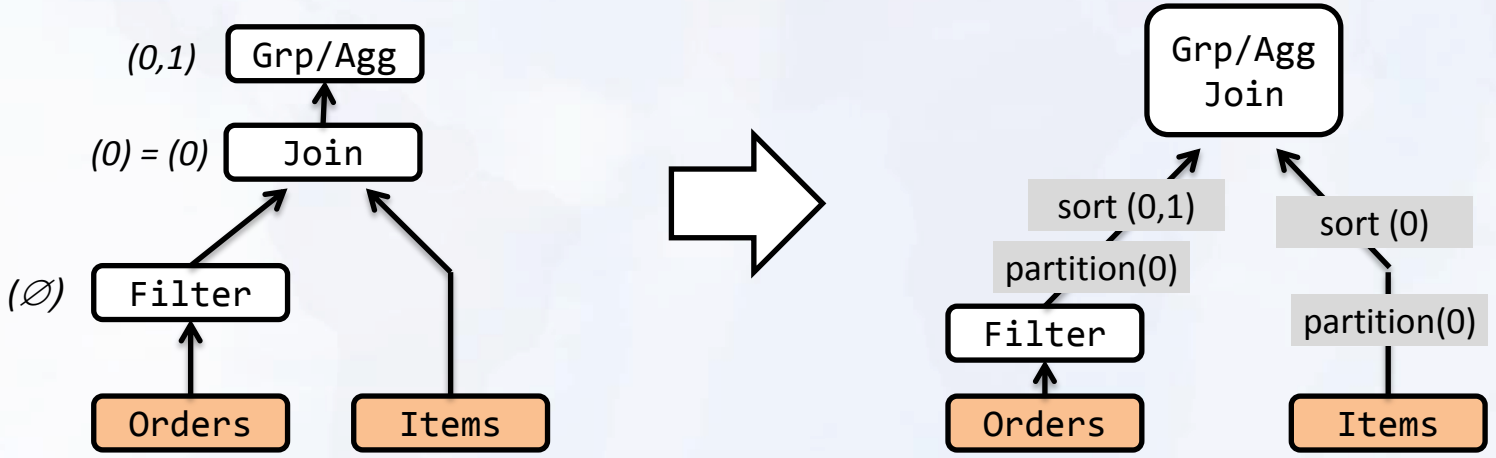
Tuples (w/ BLOB for recursion)

```
(id:Int, left:BLOB,
         right:BLOB)
```

Optimization

```
case class Order(id: Int, priority: Int, ...)
case class Item(id: Int, price: double, )
case class PricedOrder(id, priority, price)
```

```
val orders = DataSource(...)
val items = DataSource(...)

val filtered = orders filter { ... }

val prio = filtered join items where { _.id } isEqualTo { _.id }
                    map {(o,li) => PricedOrder(o.id, o.priority, li.price)}

val sales = prio groupBy {p => (p.id, p.priority)} aggregate ({_.price},SUM)
```

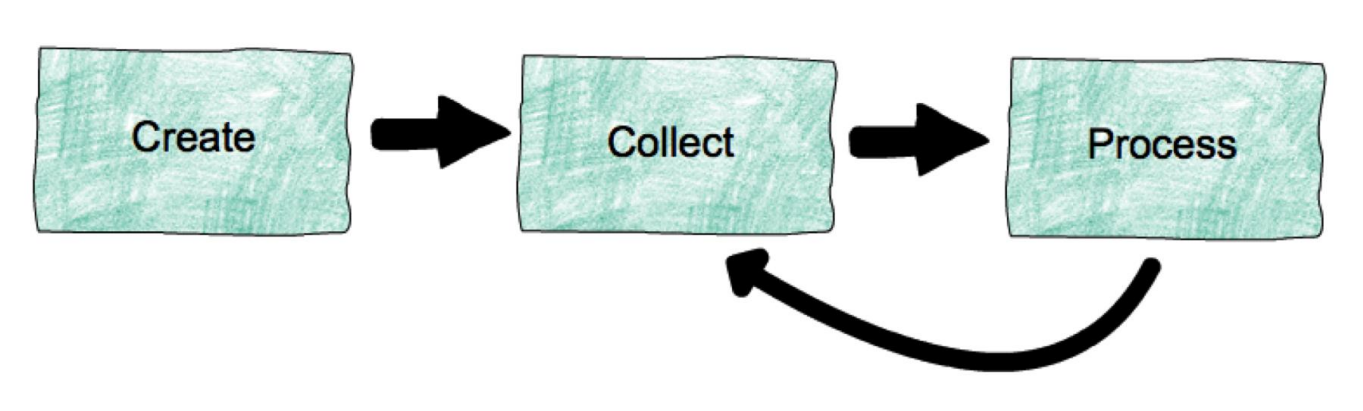


Opening the Black Boxes in Data Flow Optimization. PVLDB 5(11) 2012

Peeking into the optimization of data flow programs with MapReduce-style UDFs. ICDE 2013

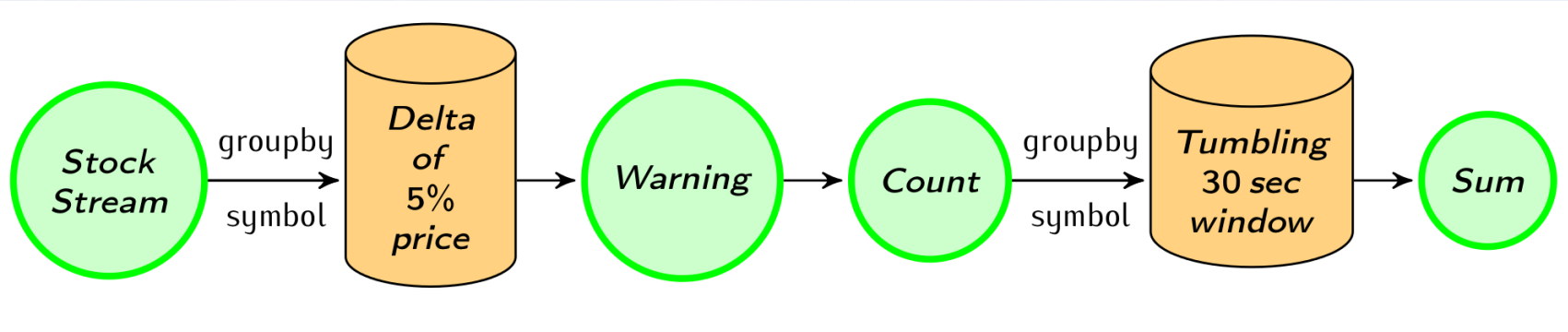
DATA STREAMING ANALYSIS

Life of data streams



- **Create:** create streams from event sources (machines, databases, logs, sensors, …)
- **Collect:** collect and make streams available for consumption (e.g., Apache Kafka)
- **Process:** process streams, possibly generating derived streams (e.g., Apache Flink)

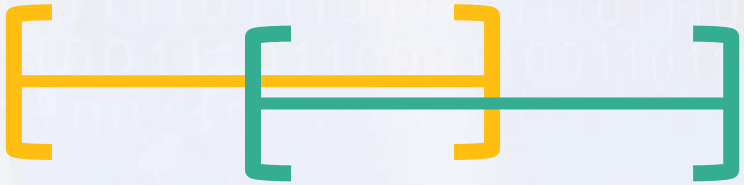
Stream Analysis in Flink




```
case class Count(symbol: String, count: Int)
val defaultPrice = StockPrice("", 1000)

//Use delta policy to create price change warnings
val priceWarnings = stockStream.groupBy("symbol")
  .window(Delta.of(0.05, priceChange, defaultPrice))
  .mapWindow(sendWarning _)

//Count the number of warnings every half a minute
val warningsPerStock = priceWarnings.map(Count(_, 1))
  .groupBy("symbol")
  .window(Time.of(30, SECONDS))
  .sum("count")
```

More at: http://flink.apache.org/news/2015/02/09/streaming-example.html

Defining windows in Flink



- Trigger policy
  - When to trigger the computation on current window
- Eviction policy
  - When data points should leave the window
  - Defines window width/size
- E.g., count-based policy
  - evict when #elements > n
  - start a new window every n-th element
- Built-in: Count, Time, Delta policies

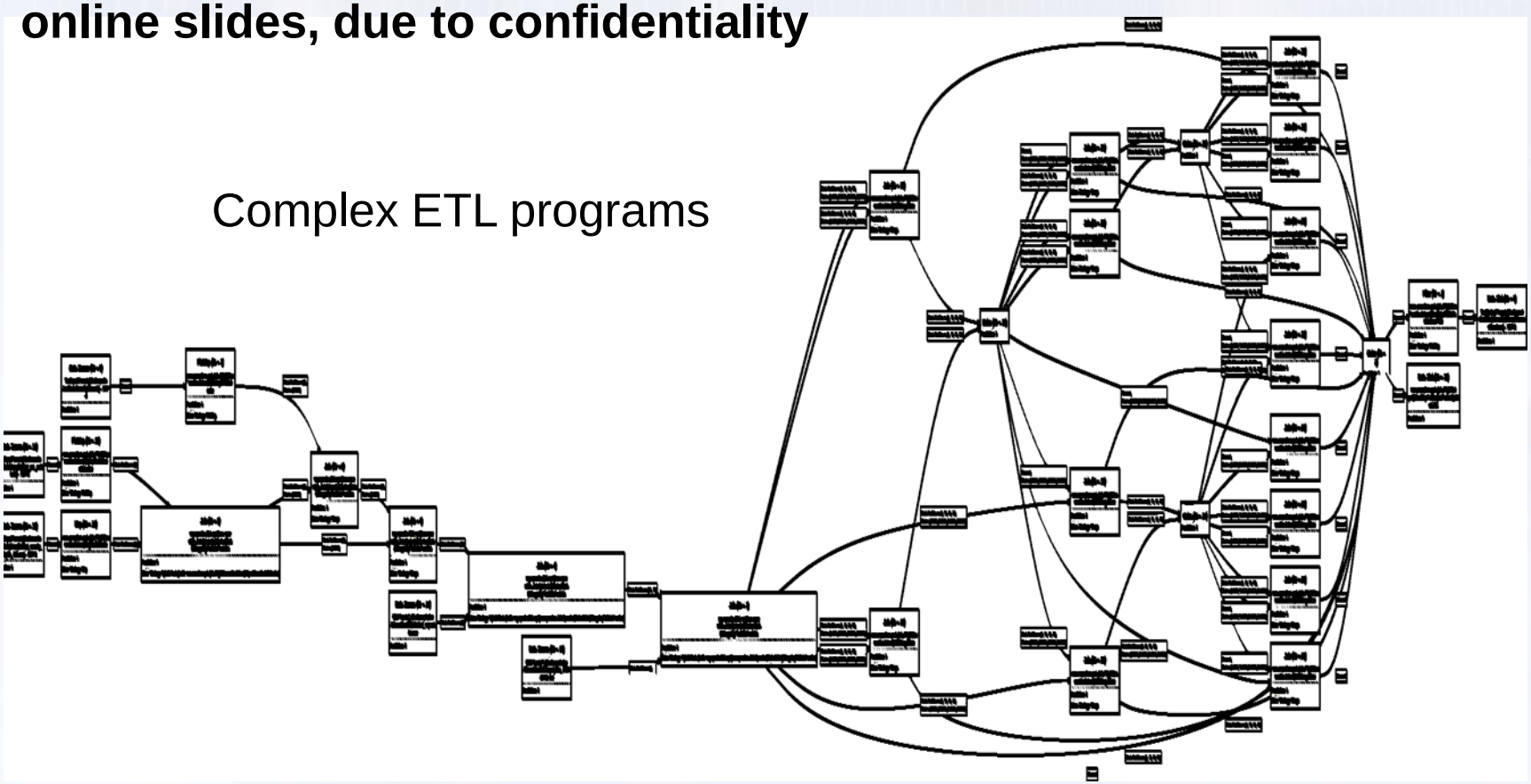
HEAVY ETL PIPELINES

31

Heavy Data Pipelines

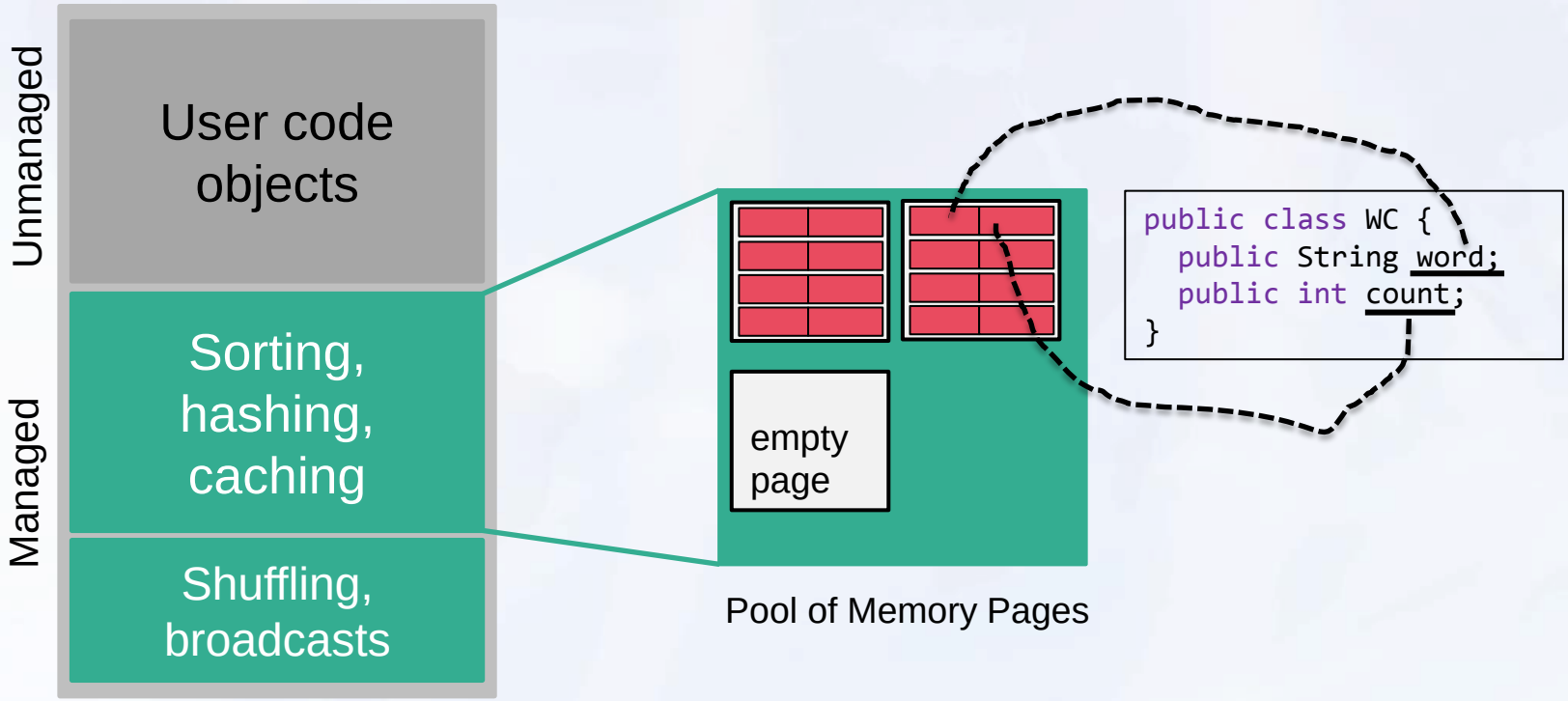
Apology: Graph had to be blurred for online slides, due to confidentiality

Complex ETL programs



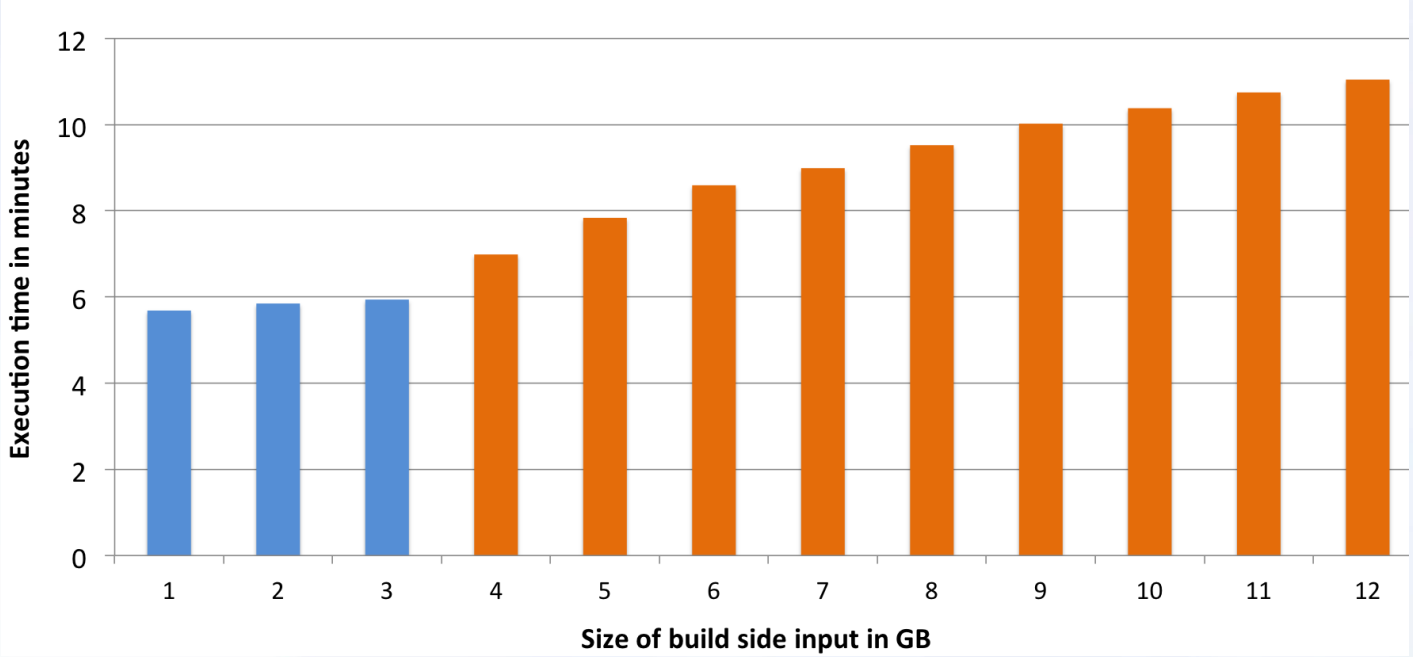
Memory Management

Flink contains its own memory management stack. Memory is allocated, de-allocated, and used strictly using an internal buffer pool implementation. To do that, Flink contains its own type extraction and serialization components.



More at: https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=53741525

Smooth out-of-core performance



Single-core join of 1KB Java objects beyond memory (4 GB)

Blue bars are in-memory, orange bars (partially) out-of-core

More at: http://flink.apache.org/news/2015/03/13/peeking-into-Apache-Flinks-Engine-Room.html

34

Benefits of managed memory

- More reliable and stable performance (less GC effects, easy to go to disk)

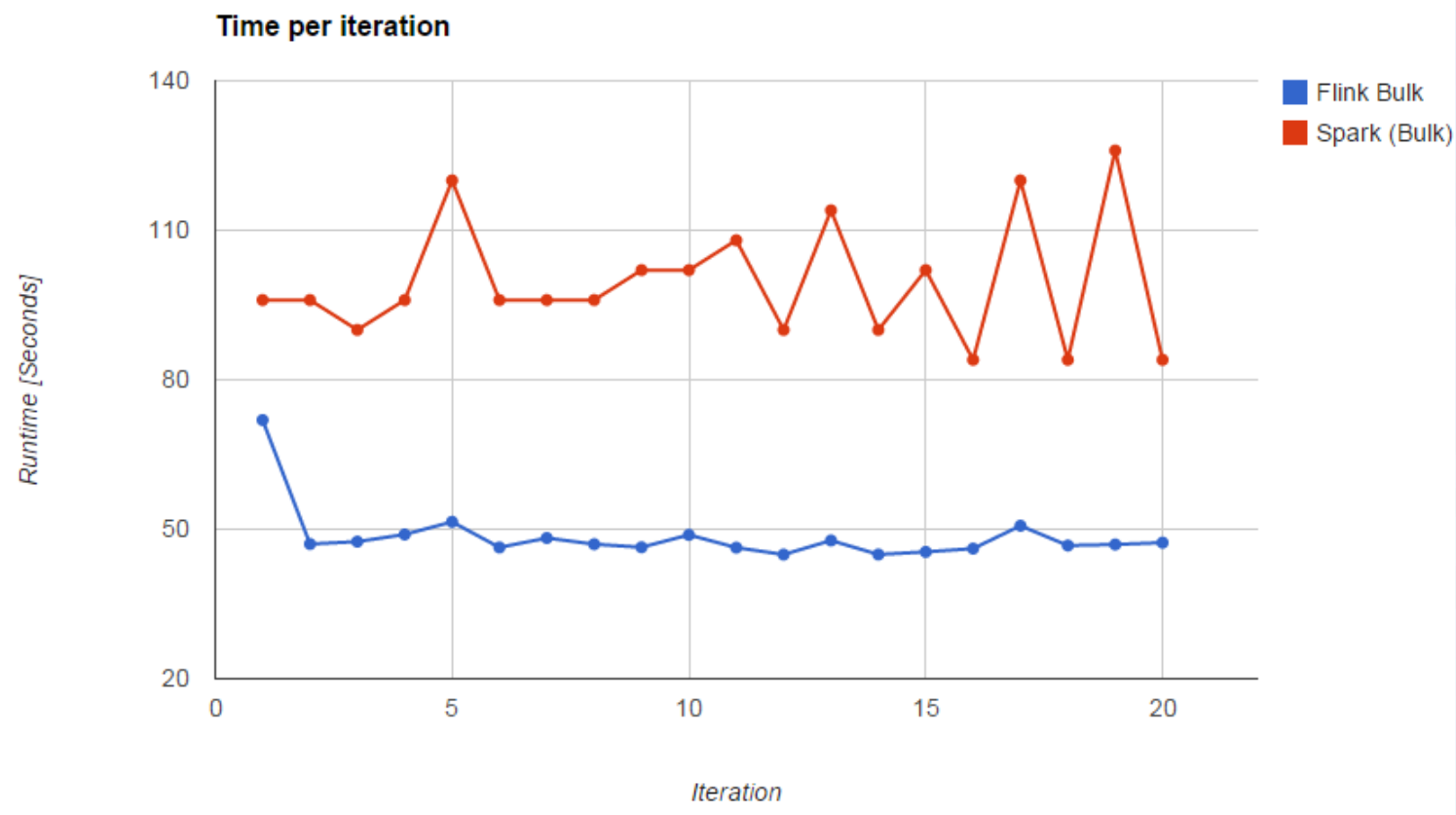
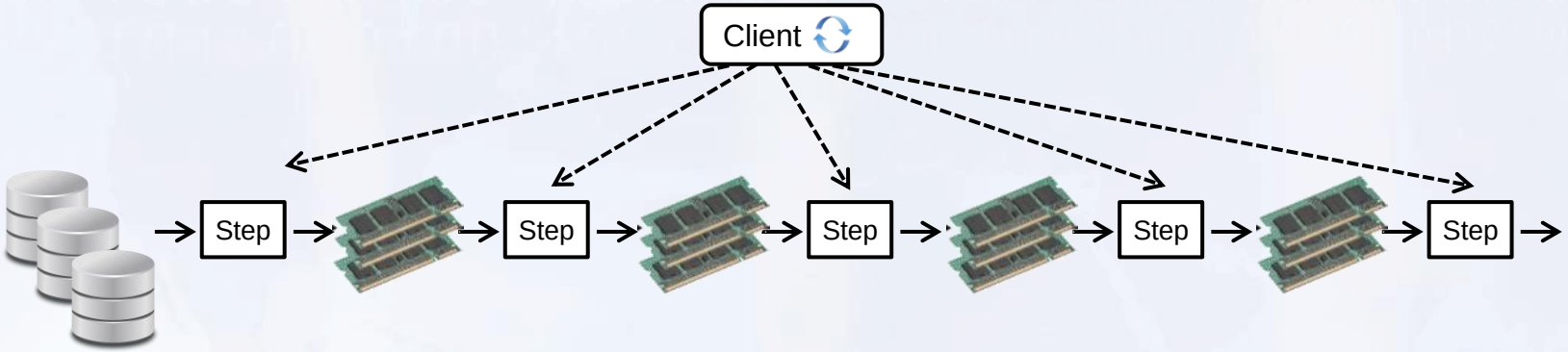


Table API

```
val customers = envreadCsvFile(…).as('id, 'mktSegment)
      .filter( 'mktSegment === "AUTOMOBILE" )

val orders = env.readCsvFile(…)
      .filter( o => dateFormat.parse(o.orderDate).before(date) )
      .as('orderId, 'custId, 'orderDate, 'shipPrio)

val items = orders
      .join(customers).where('custId === 'id)
      .join(lineitems).where('orderId === 'id)
      .select('orderId,'orderDate,'shipPrio,
          'extdPrice * (Literal(1.0f) - 'discount) as 'revenue)

val result = items
      .groupBy('orderId, 'orderDate, 'shipPrio)
      .select('orderId, 'revenue.sum, 'orderDate, 'shipPrio)
```

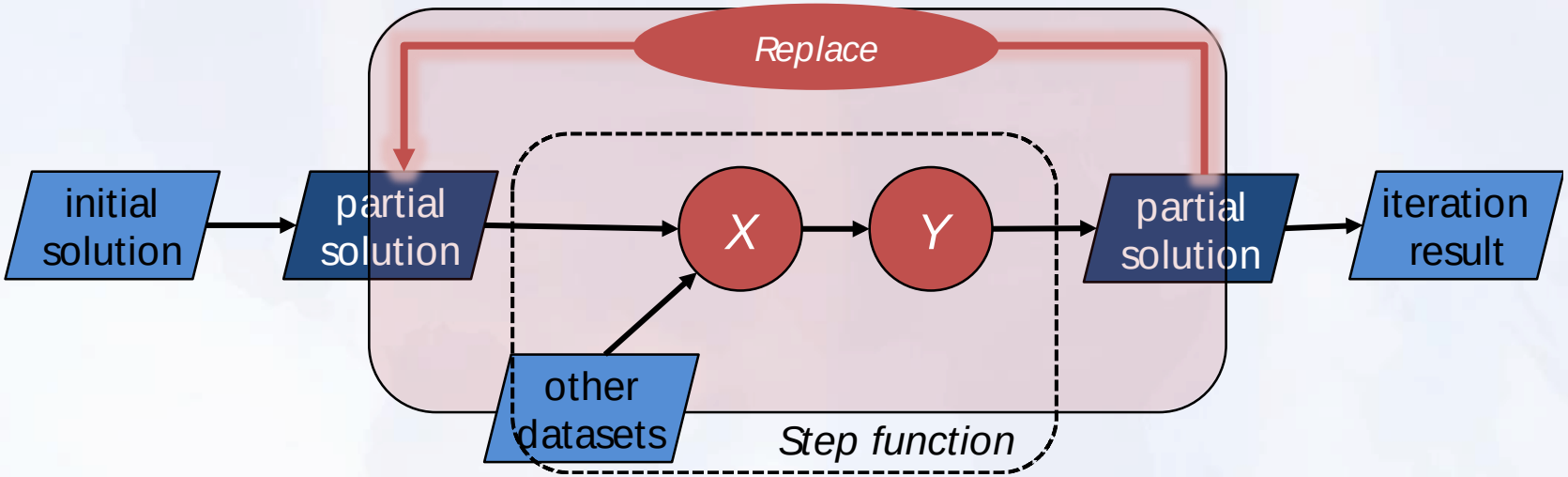
ITERATIONS IN DATA FLOWS → MACHINE LEARNING ALGORITHMS

Iterate by looping



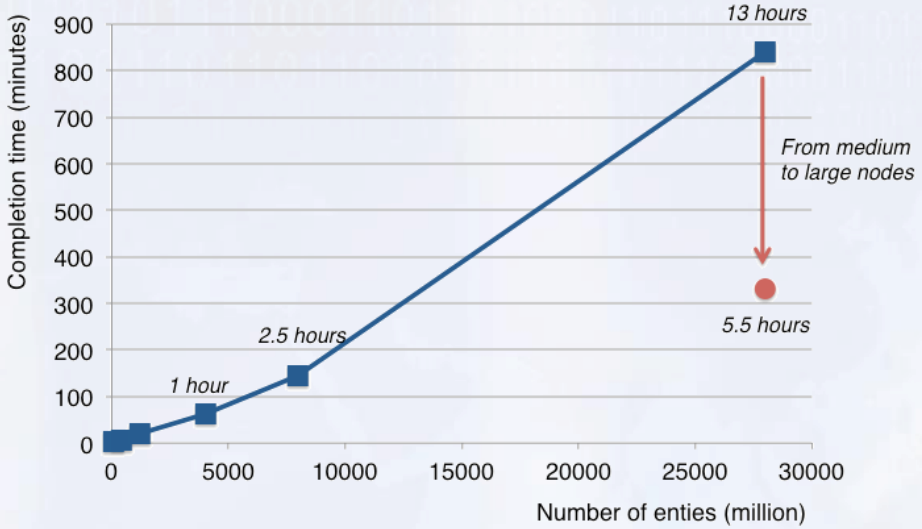
- for/while loop in client submits one job per iteration step
- Data reuse by caching in memory and/or disk

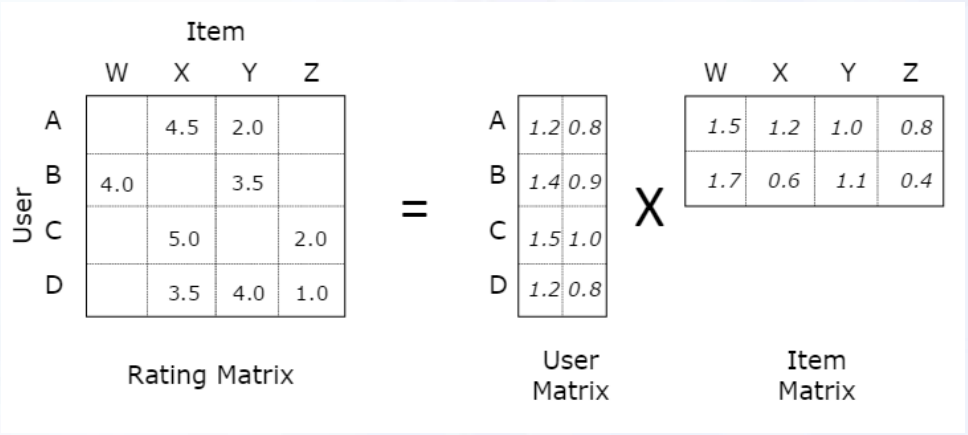
Iterate in the Dataflow



Large-Scale Machine Learning

Factorizing a matrix with 28 billion ratings for recommendations





(Scale of Netflix or Spotify)

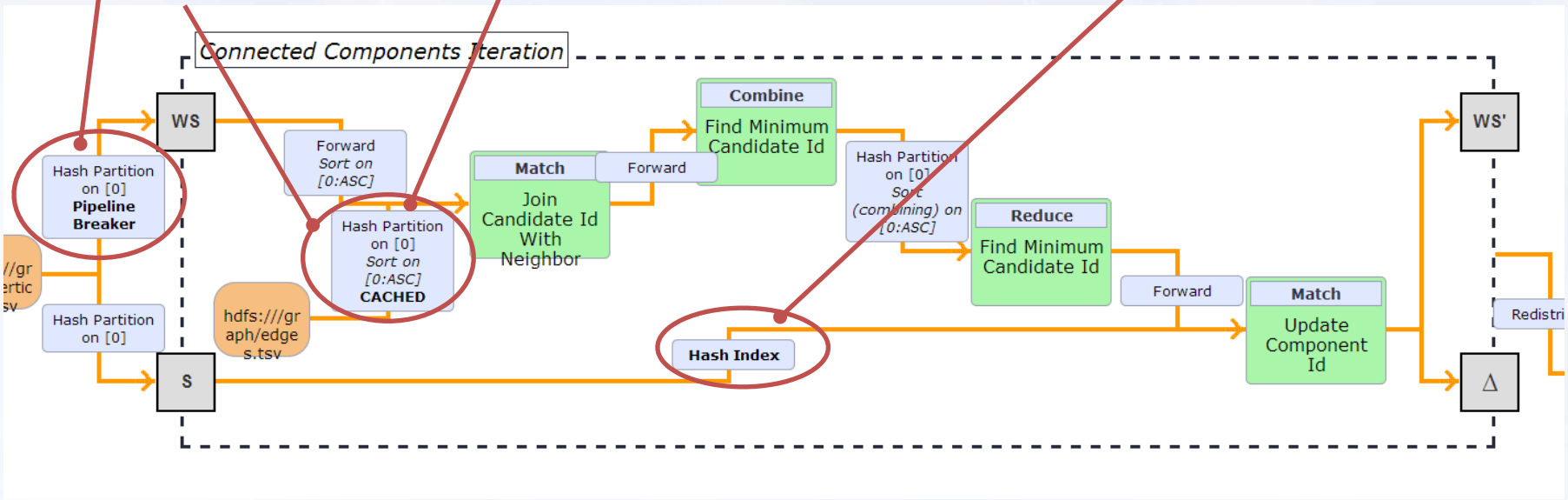
More at: http://data-artisans.com/computing-recommendations-with-flink.html

Optimizing iterative programs

Pushing work „out of the loop“

Caching Loop-invariant Data

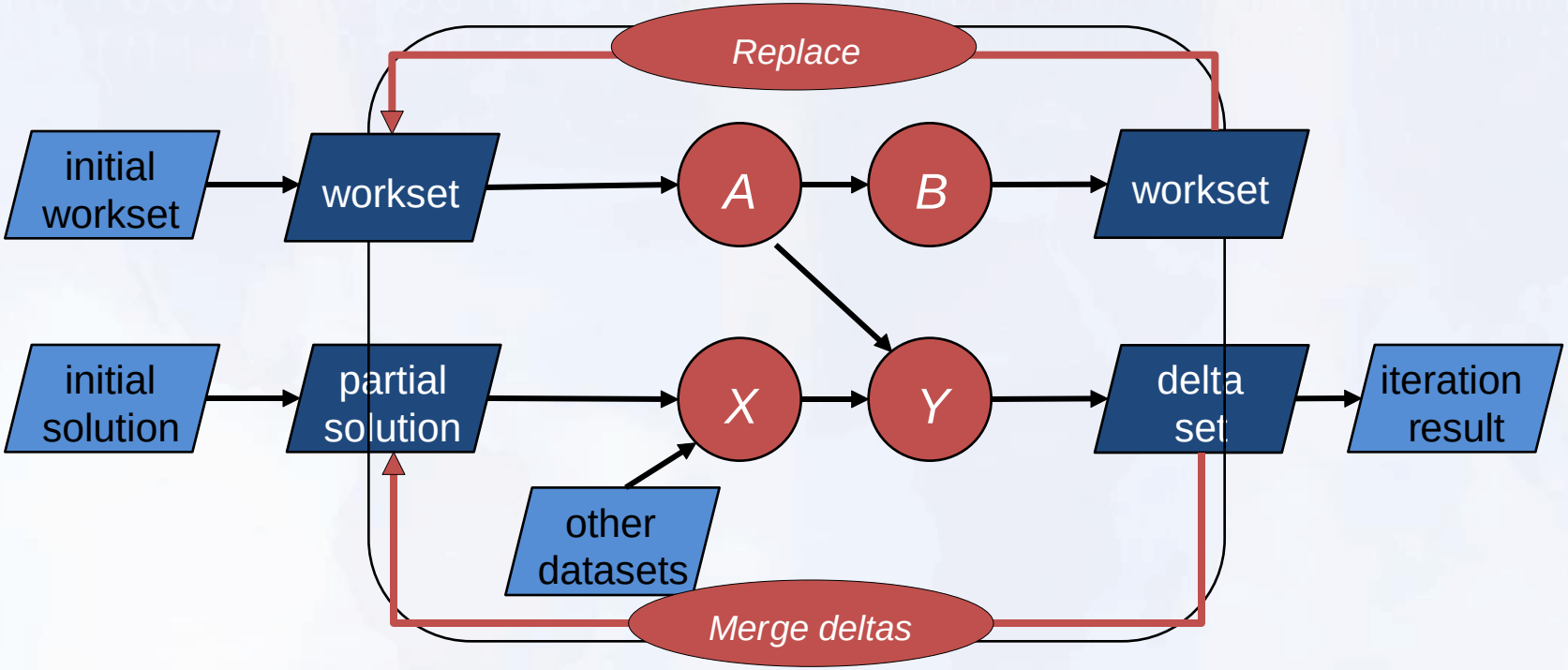
Maintain state as index



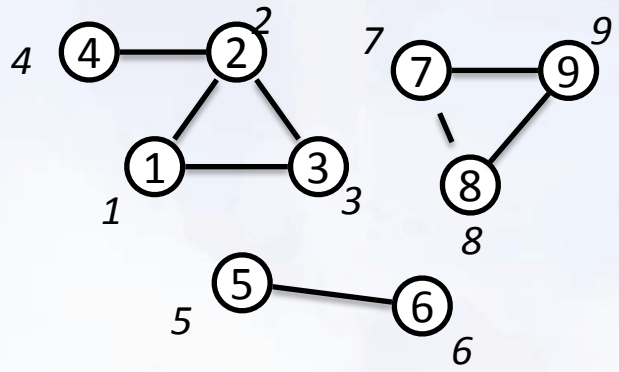
Spinning Fast Iterative Data Flows. PVLDB 5(11): 1268-1279 (2012)

STATE IN ITERATIONS → GRAPHS AND MACHINE LEARNING

Iterate natively with deltas



Workset Algorithm Illustrated



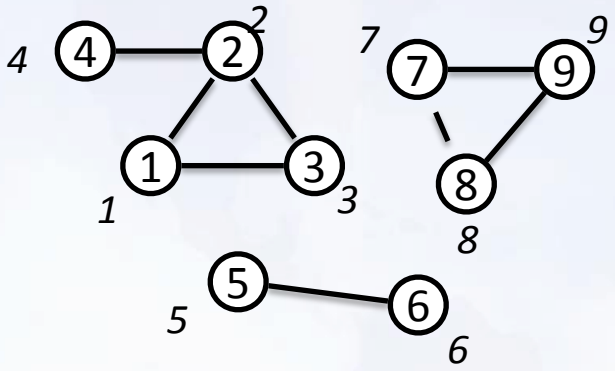
Algorithm: Find connected components of a graph.

Start: Vertices have IDs that respresent the component they belong to. Initially, every vertex has its own id (is its own component).

Step: Each vertex tells its neighbors its component id. Vertices take the min-ID of all candidates from their neighbors. A vertex that did not adopt a new ID needs not participate in the next step, as it has nothing new to tell its neighbors.

Workset Algorithm Illustrated

Solution Set



Solution Set Delta

Workset

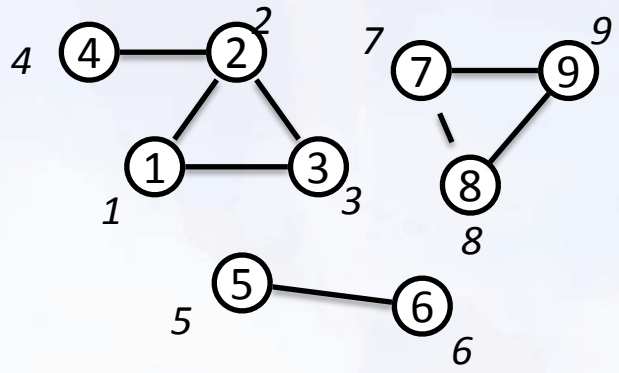
1 (2,2)	3 (1,1)	8 (7,7)
(3,3)	(2,2)	(9,9)
2 (1,1)	4 (2,2)	9 (7,7)
(3,3)		(8,8)
(4,4)	5 (6,6)	
	6 (5,5)	

Messages sent to neighbors:

1 (4, 3) means that vertex 1 receives a candidate id of 3 from vertex 4

Workset Algorithm Illustrated

Solution Set



Workset

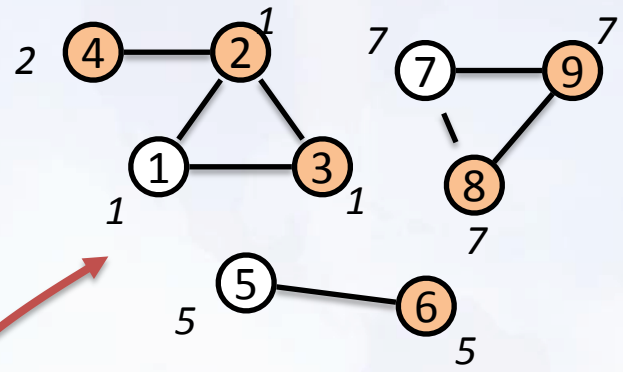
1 (2,2)	3 (1,1)	8 (7,7)
(3,3)	(2,2)	(9,9)
2 (1,1)	4 (2,2)	9 (7,7)
(3,3)		(8,8)
(4,4)	5 (6,6)	
	6 (5,5)	

Solution Set Delta

(2,1)	(6, 5)
(3,1)	(8,7)
(4,2)	(9,7)

Workset Algorithm Illustrated

Solution Set



Workset

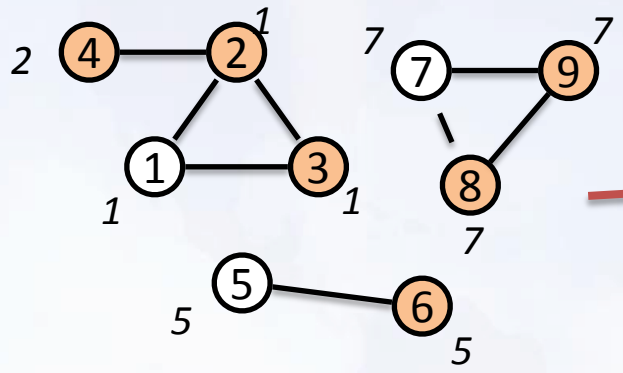
1 (2,2)	3 (1,1)	8 (7,7)
(3,3)	(2,2)	(9,9)
2 (1,1)	4 (2,2)	9 (7,7)
(3,3)		(8,8)
(4,4)	5 (6,6)	
	6 (5,5)	

Solution Set Delta

(2,1)	(6, 5)
(3,1)	(8,7)
(4,2)	(9,7)

Workset Algorithm Illustrated

Solution Set



Workset

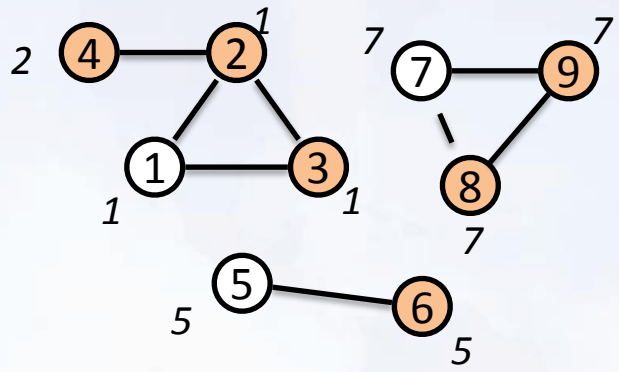
1 (2,1)	3 (2,1)	7 (8,7)
(3,1)		(9,7)
	4 (2,1)	
2 (3,1)		8 (9,7)
(4,2)	5 (6,5)	
		9 (8,7)

Solution Set Delta

(2,1)	(6, 5)
(3,1)	(8,7)
(4,2)	(9,7)

Workset Algorithm Illustrated

Solution Set



Workset

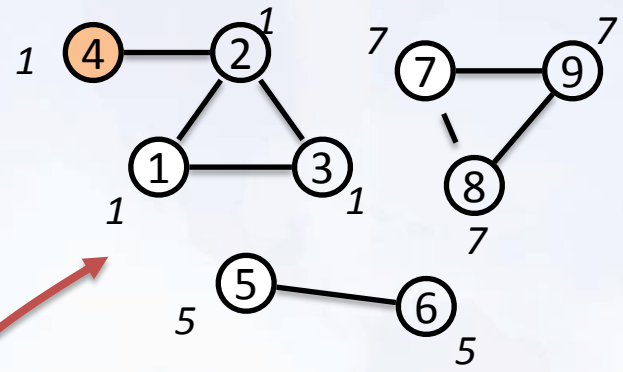
1 (2,1)	3 (2,1)	7 (8,7)
(3,1)		(9,7)
	4 (2,1)	
2 (3,1)		8 (9,7)
(4,2)	5 (6,5)	
		9 (8,7)

Solution Set Delta

(4,1)

Workset Algorithm Illustrated

Solution Set



Workset

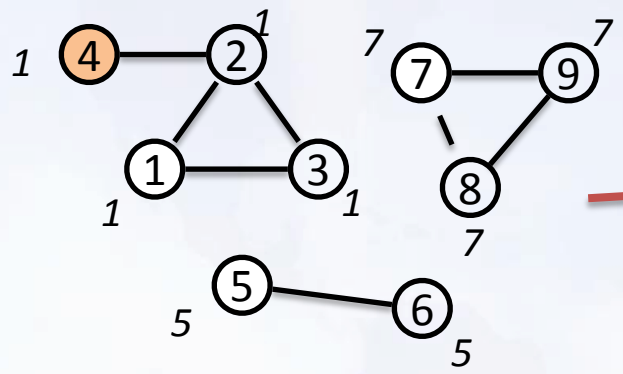
1 (2,1)	3 (2,1)	7 (8,7)
(3,1)		(9,7)
	4 (2,1)	
2 (3,1)		8 (9,7)
(4,2)	5 (6,5)	
		9 (8,7)

Solution Set Delta

(4,1)

Workset Algorithm Illustrated

Solution Set



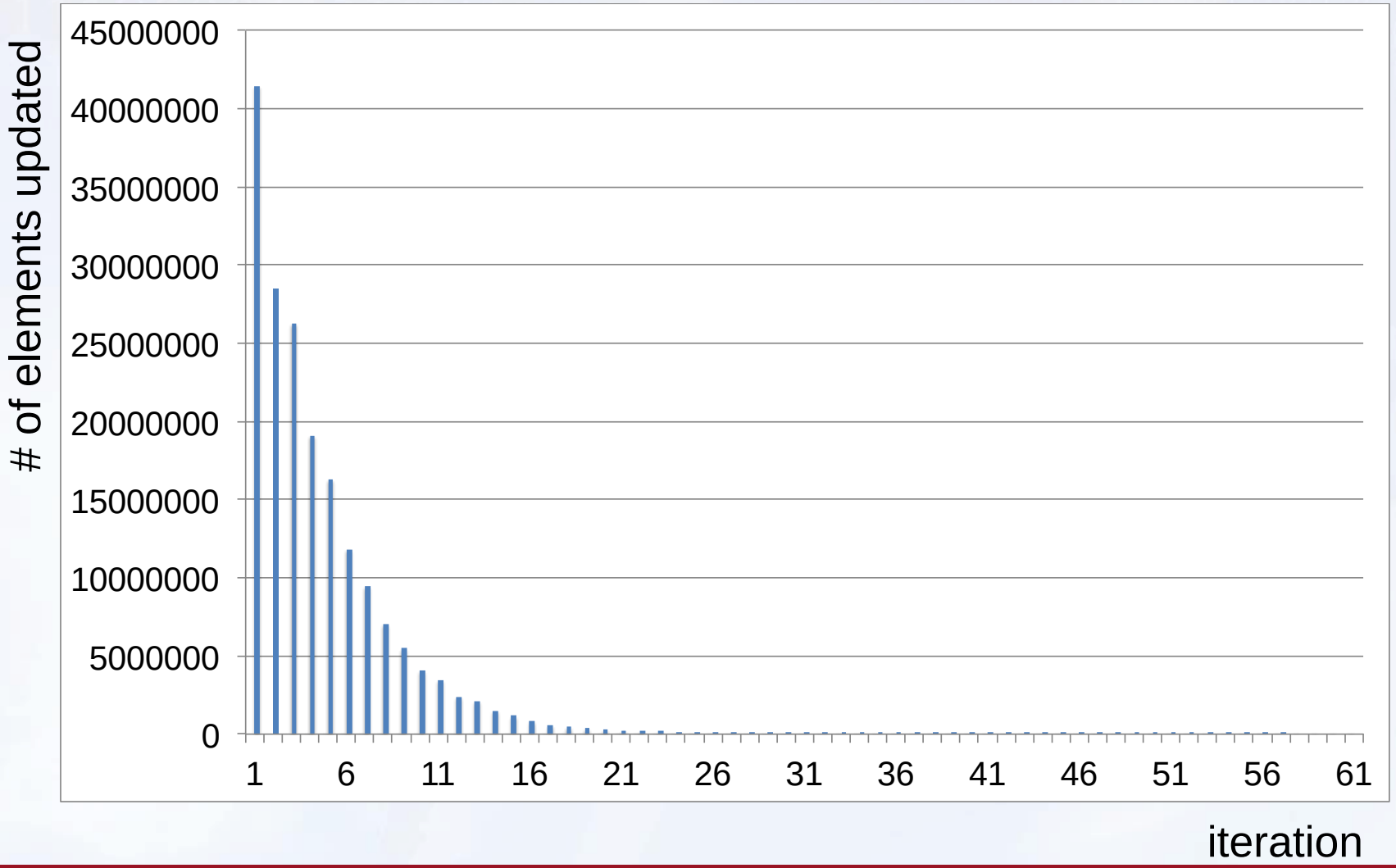
Workset

2 (4,1)

Solution Set Delta

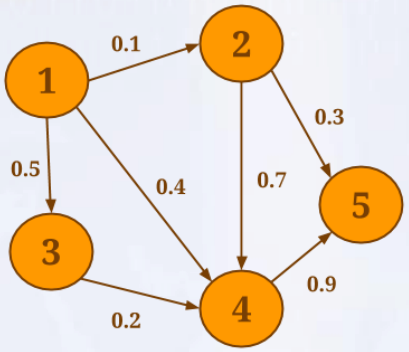
(4,1)

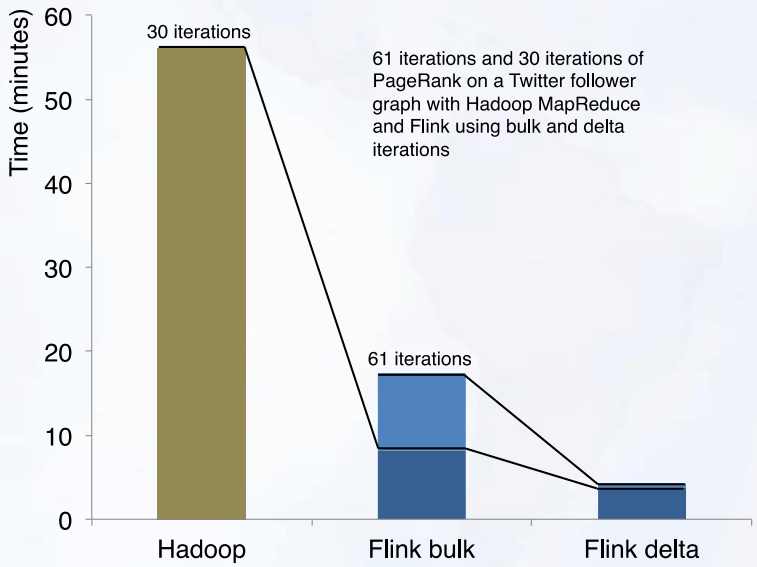
Effect of delta iterations…



… very fast graph analysis

Performance competitive with dedicated graph analysis systems

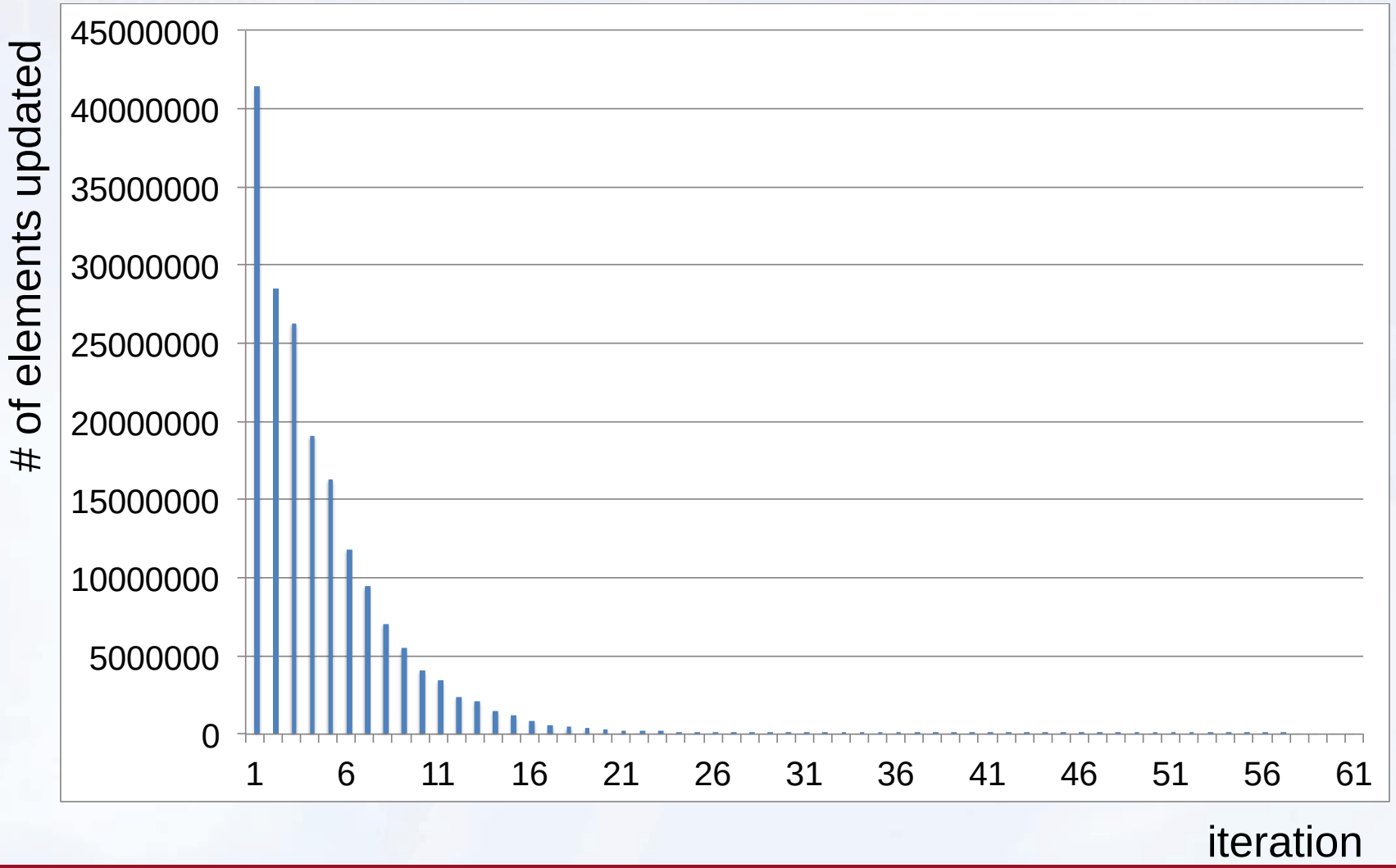




… and mix and match ETL-style and graph analysis in one program

More at: http://data-artisans.com/data-analysis-with-flink.html

Effect of delta iterations…

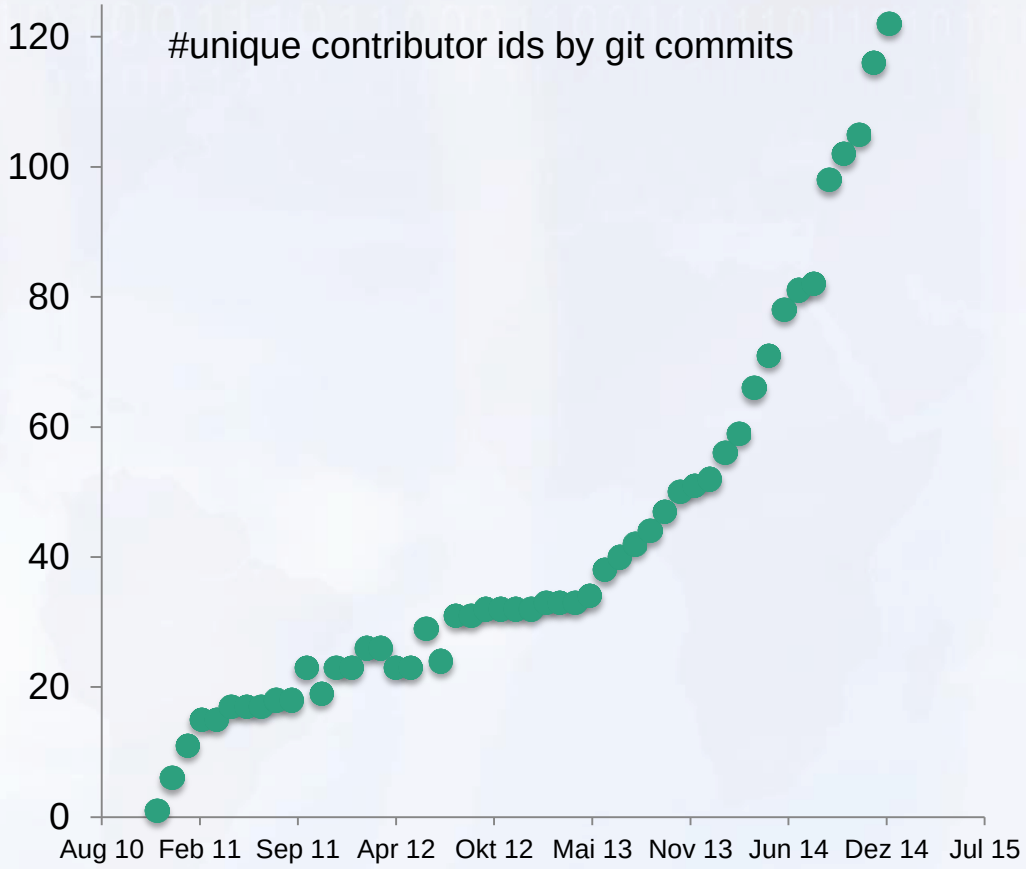


CLOSING

Flink Roadmap for 2015

- Out-of-core state in Streaming
- Monitoring and scaling for streaming
- Streaming Machine Learning with SAMOA
- More additions to the libraries
  - Batch Machine Learning
  - Graph library additions (more algorithms)
- SQL on top of expression language
- Master failover

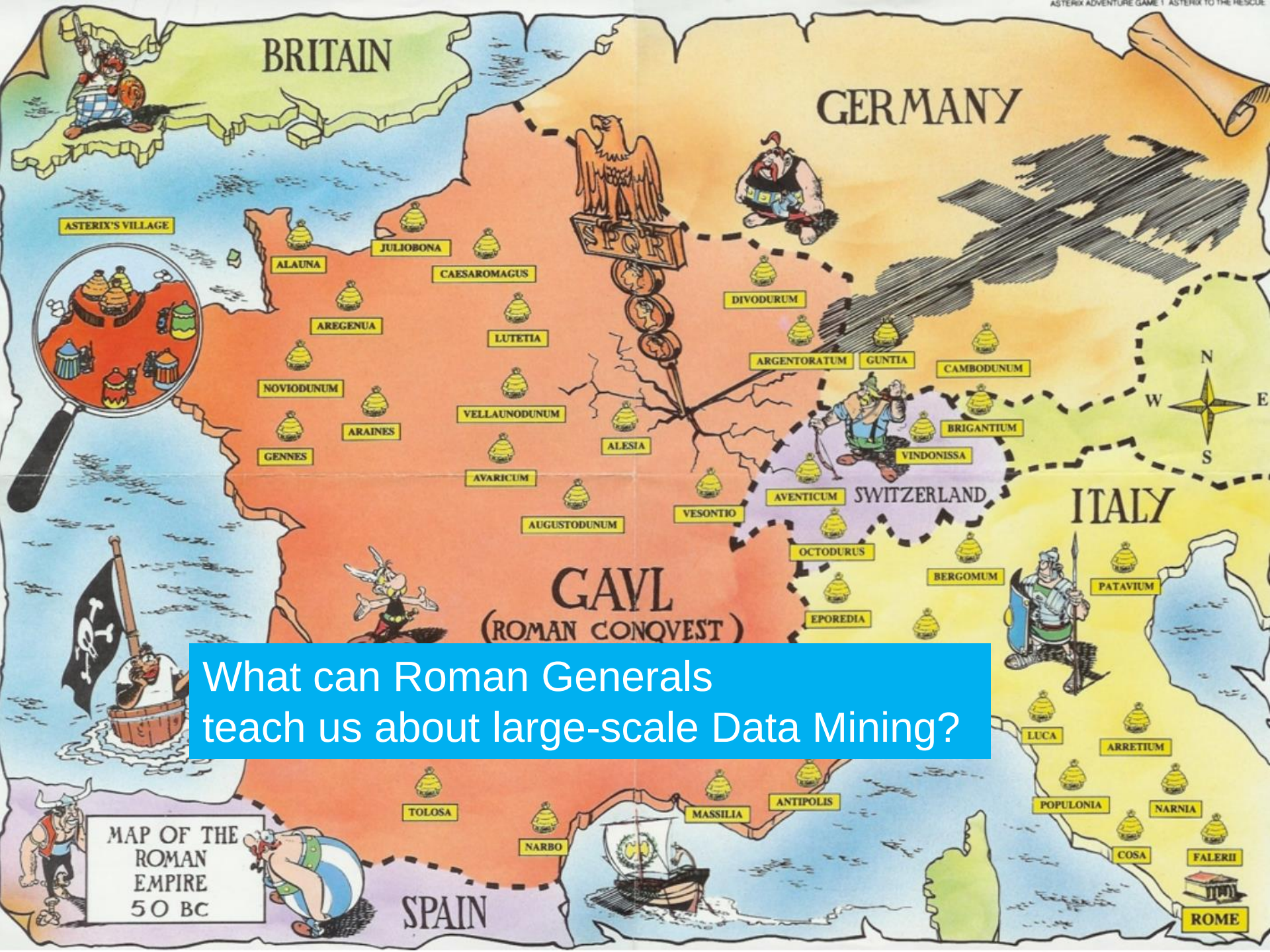
Flink community



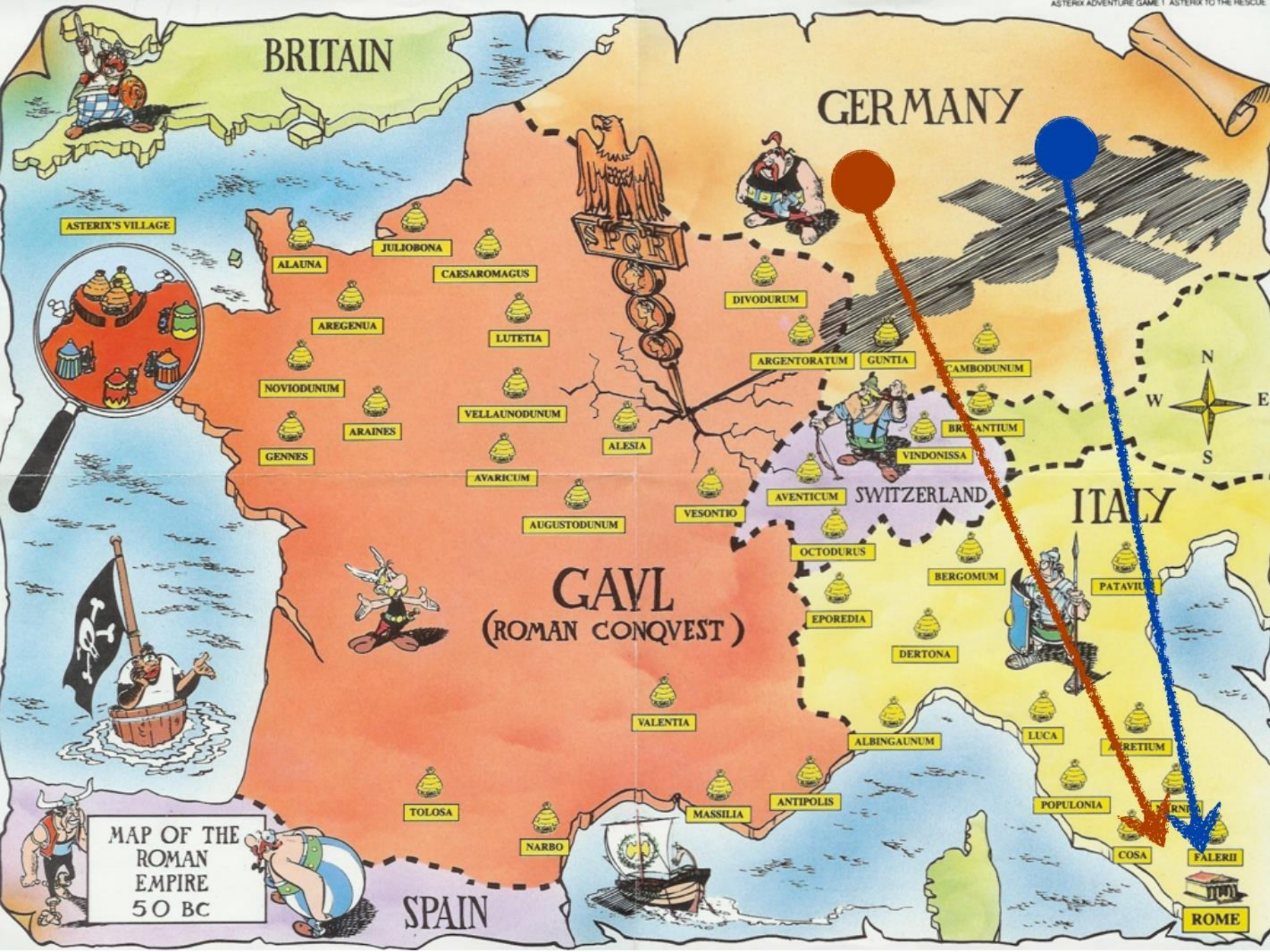
What can we learn from Roman Generals about Big Data?

Marcus
Pessimus

Gaius
Optimus

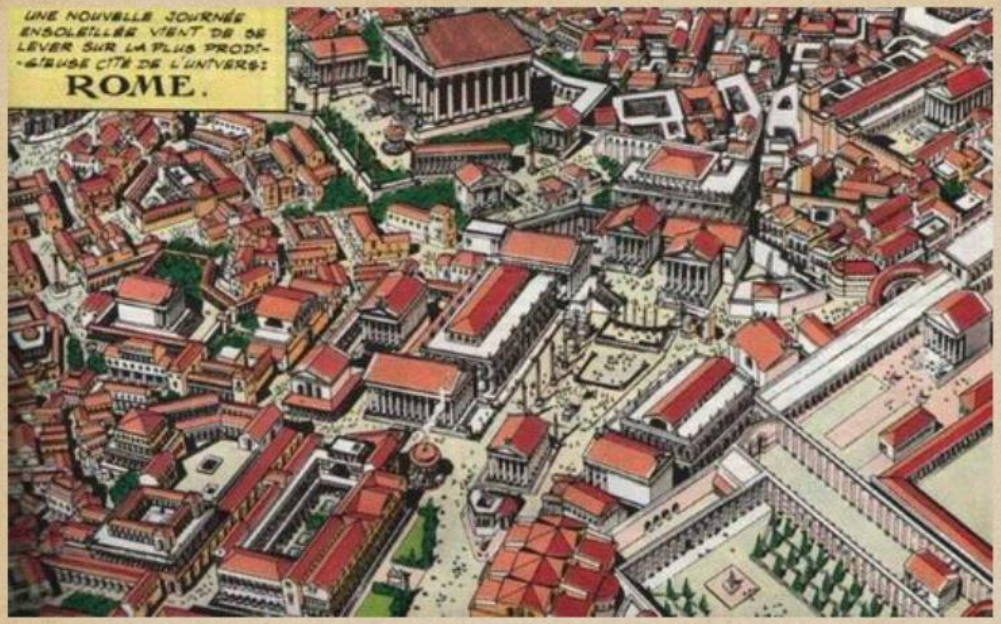


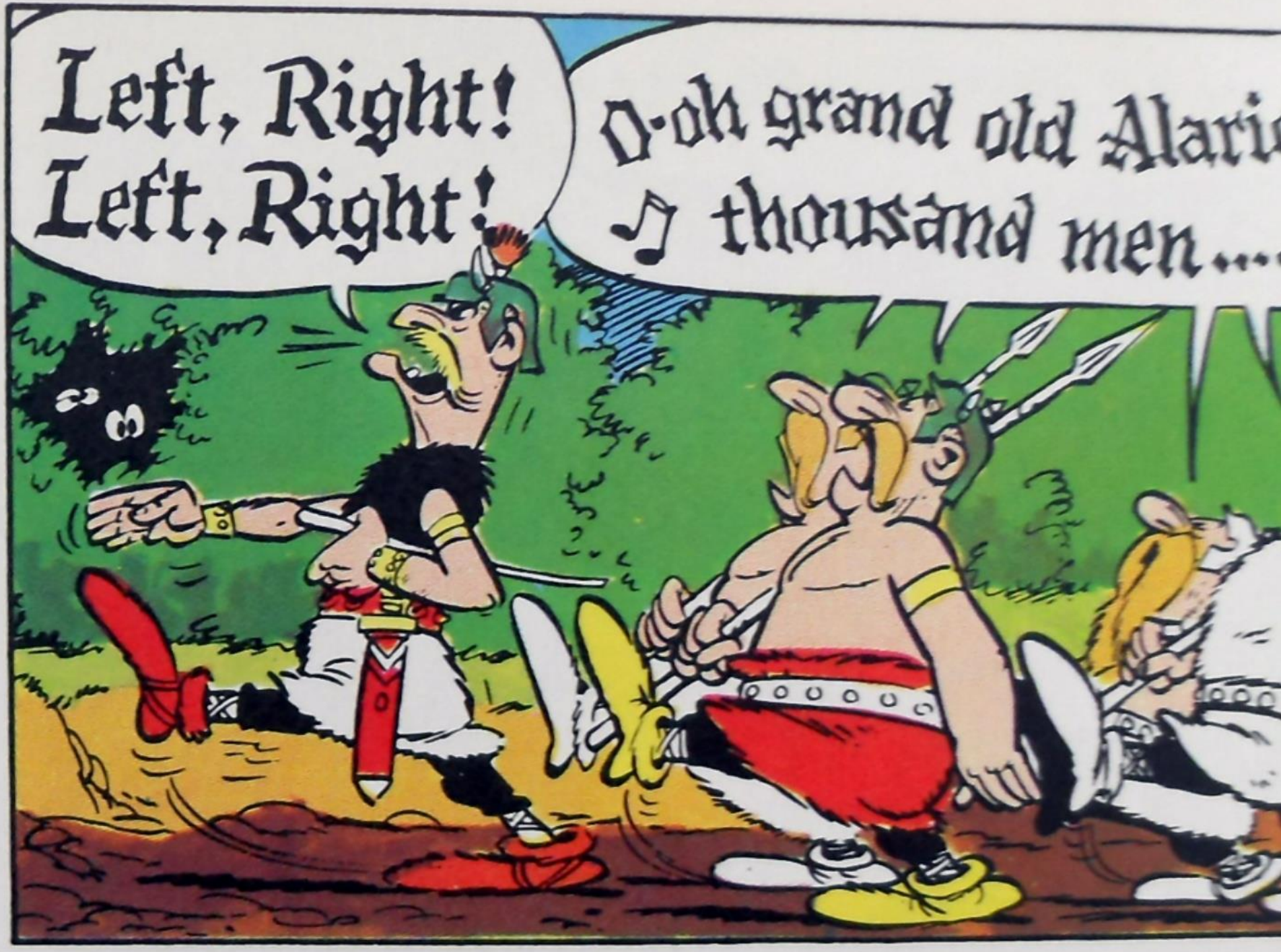
What can Roman Generals
teach us about large-scale Data Mining?


BRITAIN
GERMANY
ASTERIX'S VILLAGE
ALAUNA
JULIOBONA
CAESAROMAGUS
AREGENUA
LUTETIA
NOVIODUNUM
ARAINES
VELLAUNODUNUM
GENNES
AVARICUM
ALESIA
AUGUSTODUNUM
VESONTIO
DIVODURUM
ARGENTORATUM
GUNTIA
CAMBODUNUM
VINDONISSA
AVENTICUM
SWITZERLAND
ITALY
OCTODURUS
BERGOMUM
EPOREDIA
DERTONA
VALENTIA
ALBINGAUNUM
LUCA
POPULONIA
COSA
FALERII
ROME
TOLOSA
NARBO
MASSILIA
ANTIPOLIS
GAUL
(ROMAN CONQUEST)
SPAIN
N
W
E
S
MAP OF THE ROMAN EMPIRE 50 BC

Marcus
Pessimus

Gaius
Optimus


UNE NOUVELLE JOURNÉE ENSOLEILLÉE VIENT DE SE LEVER SUR LA PLUS PRODI-
-GIEUSE CITÉ DE L'UNIVERS:
ROME.


Left, Right!
Left, Right!
O-oh grand old Alari
♫ thousand men....

1. Build Fortified camp

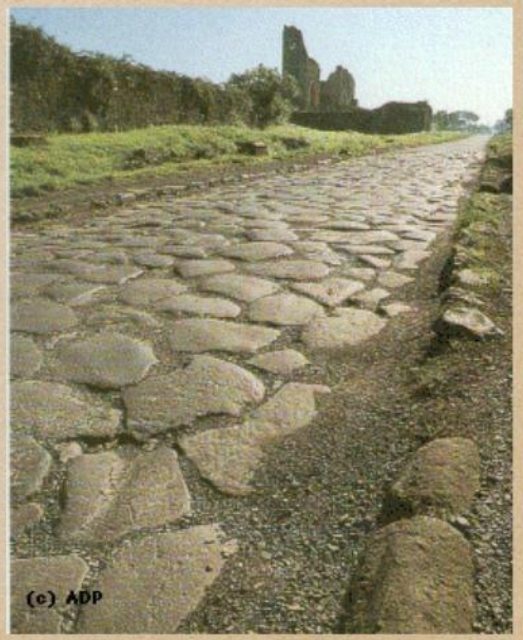
2. Deal with attack

3. Retreat to last night's camp

1. No camps

2. Chase barbarians

3. Wander and follow next roman road



All Roads Lead to Rome

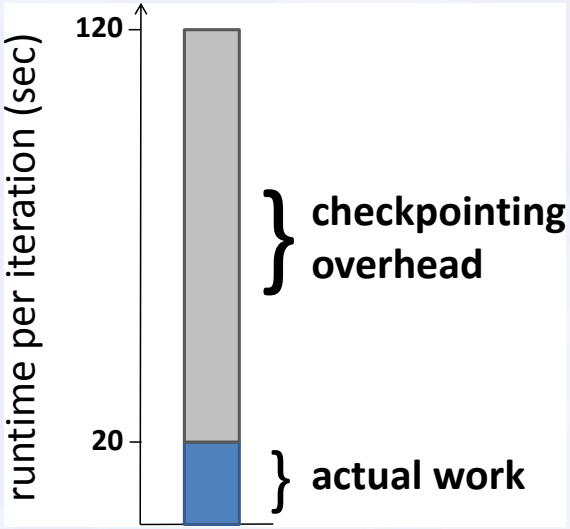
Fault tolerance

Pessimistic Recovery:

- Write intermediate state to stable storage
- Restart from such a checkpoint in case of a failure

Problematic:

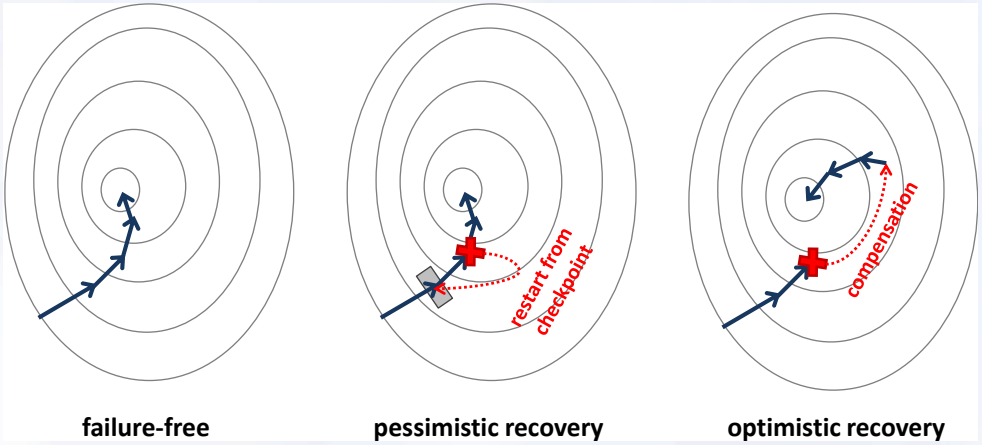
- High overhead, checkpoint must be replicated to other machines
- Overhead always incurred, even if no failures happen!



➢ **How can we avoid this overhead in failure-free cases**

Optimistic recovery

- Many data mining algorithms are **fixpoint algorithms**
- **Optimistic Recovery**: jump to a different state in case of a failure, still converge to solution



- No checkpoints → **No overhead in absense of failures**!
- algorithm-specific **compensation function** must restore state

All Roads lead to Rome

If you are interested more, read our CIKM 2013 paper:

Sebastian Schelter, Stephan Ewen, Kostas Tzoumas, Volker Markl: **"All roads lead to Rome": optimistic recovery for distributed iterative data processing**. *CIKM 2013*: 1919-1928

Sergey Dudoladov, Chen Xu, Sebastian Schelter, Asterios Katsifodimos, Stephan Ewen, Kostas Tzoumas, Volker Markl: **Optimistic Recovery for Iterative Dataflows in Action**. *SIGMOD 2015*

Further current work:

- **Self-Tuning, GPU-Accelerated Kernel Density Models for Multidimensional Selectivity Estimation. SIGMOD Conference 2015: 1477-1492**

- **Efficient sample generation for scalable meta learning. ICDE 2015: 1191-1202**

- **Implicit Parallelism through Deep Language Embedding. SIGMOD Conference 2015: 47-61**

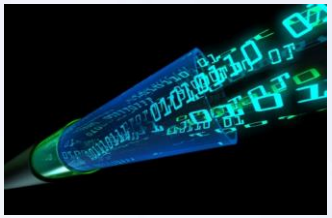
Further Challenges for Deep Analysis on Big Data

Optimizing Access on Raw Data:
in-situ data analysis

Adaptive Seamless Deployment:
Scale from laptop to cluster

Evolving Datasets:
First results fast, stream mining

Advanced Data Analysis Programs:
Declarative specification and optimization of programs with iteration and state



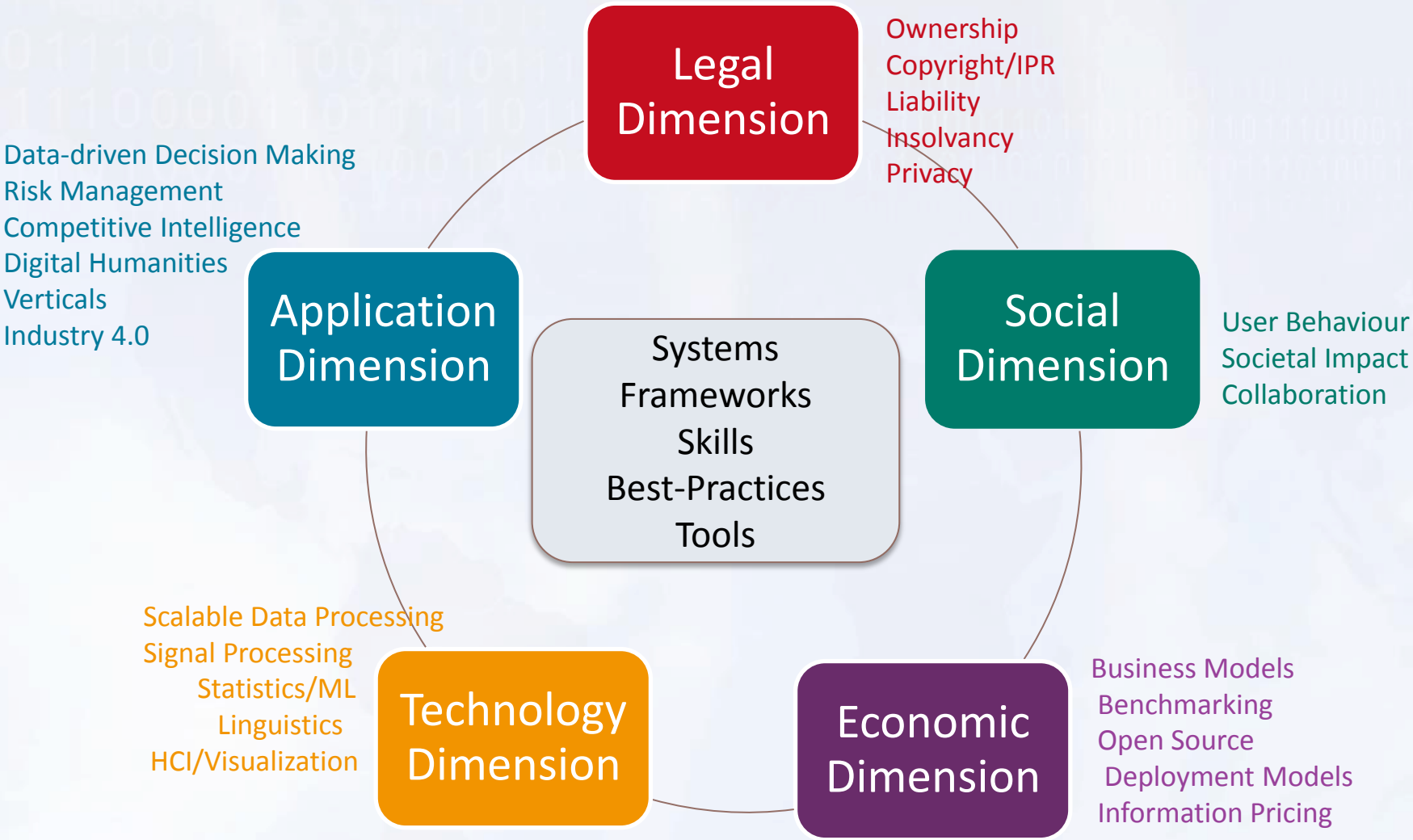
Low Latency:
Trading-off virtualization, heterogeneous CPUs

Multi-tenancy:
Continuous, workload-aware optimizations

Engines: one size does not fit all - pluggable engines and libraries

More information: http://bbdc.berlin

The 5 Dimensions of Big Data



More information: http://bbdc.berlin

Apache Flink

Fast and reliable large-scale data processing engine

Download | View on Github

Fast

State-of-the art performance exploiting in-memory processing and data streaming.

Reliable

Flink is designed to perform very well even when the cluster's memory runs out.

Expressive

Write beautiful, type-safe code in Java and Scala. Execute it on a cluster.

Easy to use

Few configuration parameters required. Cost-based optimizer built in.

Scalable

Tested on clusters of 100s of machines, Google Compute Engine, and Amazon EC2.

Hadoop-compatible

Flink runs on YARN and HDFS and has a Hadoop compatibility package.

Contributors wanted

http://flink.apache.org

Open Positions on the Post-Doc, PhD and Master Student level

- Research positions on Flink, Streaming, declarative languages for scalable data science, with applications in Digital Economy and IoT/Industrie 4.0
- Further Information: http://www.dima.tu-berlin.de/ and http://www.dfki.de/web/forschung/iam
- PhD and Postdoc Applications can be directed at: seniors@dima.tu-berlin.de
- Master students please have a look at: http://www.dima.tu-berlin.de/menue/theses/

Flink Forward registration & call for abstracts is open now

- 12/13 October 2015
- Meet developers and users of Flink!
- With Flink Workshops / Trainings!

http://flink-forward.org/ Discount Code: FlinkEITSummerSchool25

Flink Forward

BERLIN 12/13 OCT 2015

- **Two day developer conference** with in-depth talks from
  - developers and contributors
  - industry and research users
  - related projects
- **Flink training** sessions (in parallel)
  - System introduction, real-time stream processing, APIs on top

http://flink-forward.org/

Discount Code: FlinkEITSummerSchool25

78