SPARK+AI
SUMMIT 2019

WIFI SSID:SparkAISummit | Password: UnifiedAnalytics

ORGANIZED BY
databricks

SPARK+AI SUMMIT 2019

Cobrix: A Mainframe Data Source for Spark SQL and Streaming

Ruslan Iushchenko, ABSA
Jan Scherbaum, ABSA

#UnifiedAnalytics #SparkAISummit

About us

- ABSA is a Pan-African financial services provider
  - With Apache Spark at the core of its data engineering
- We fill gaps in the Hadoop ecosystem, when we find them
- Contributions to Apache Spark
- Spark-related open-source projects (https://github.com/AbsaOSS)

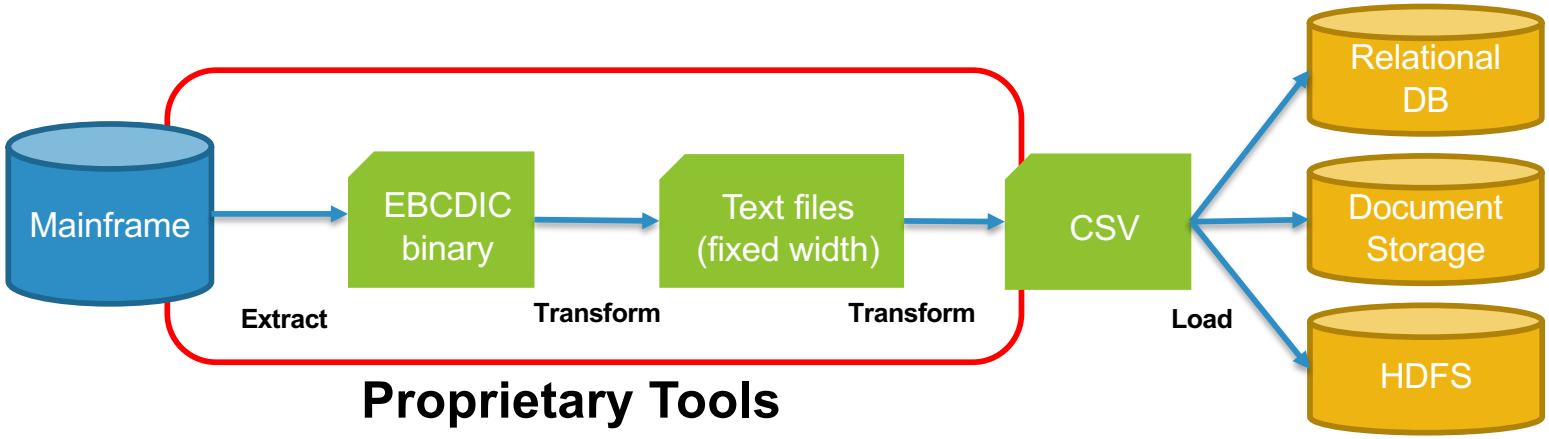
- **Spline** - a data lineage tracking and visualization tool
- **ABRiS** - Avro SerDe for structured APIs
- **Atum** - Data quality library for Spark
- **Enceladus** - A dynamic data conformance engine
- **Cobrix** - A Cobol library for Spark (focus of this presentation)

Business Motivation

- The market for Mainframes is strong, with no signs of cooling down. Mainframes
  - Are used by **71%** of Fortune **500** companies
  - Are responsible for **87%** of all **credit card transactions** in the world
  - Are part of the IT infrastructure of **92** out of the **100 biggest banks** in the world
  - Handle **68%** of the world's production **IT workloads**, while accounting for only **6%** of **IT costs**.
- For companies relying on Mainframes, becoming data-centric can be prohibitively expensive
  - High cost of hardware
  - Expensive business model for data science related activities

Source: http://blog.syncsort.com/2018/06/mainframe/9-mainframe-statistics/

Technical Motivation



- The process above takes 11 days for a 600GB file
- Legacy data models (hierarchical)
- Need for performance, scalability, flexibility, etc
- **SPOILER alert: we brought it to 1.1 hours**

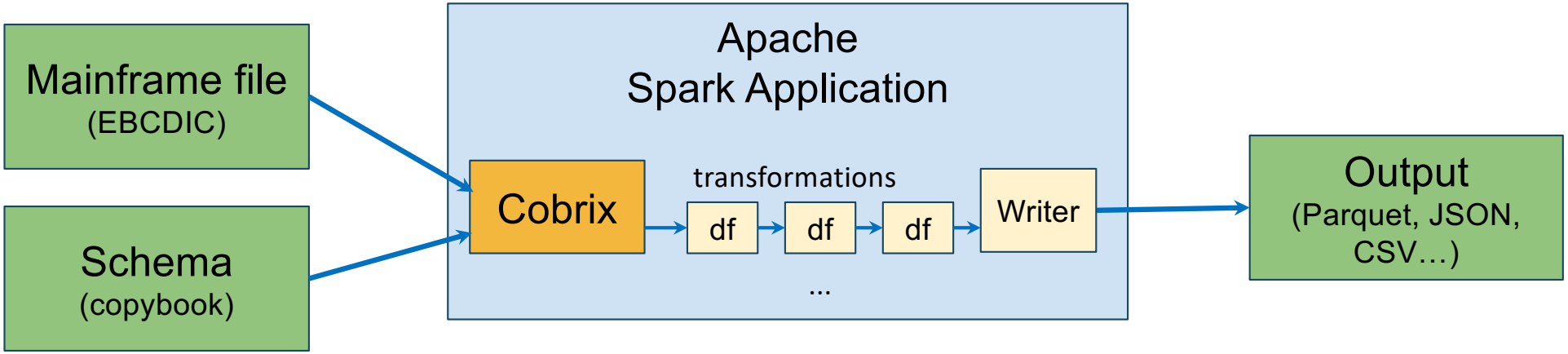
What can you do?

- Run analytics / Spark on mainframes
- Message Brokers (e.g. MQ)
- Sqoop
- Proprietary solutions
- But ...
  - Pricey
  - Slow
  - Complex (specially for legacy systems)
  - Require human involvement

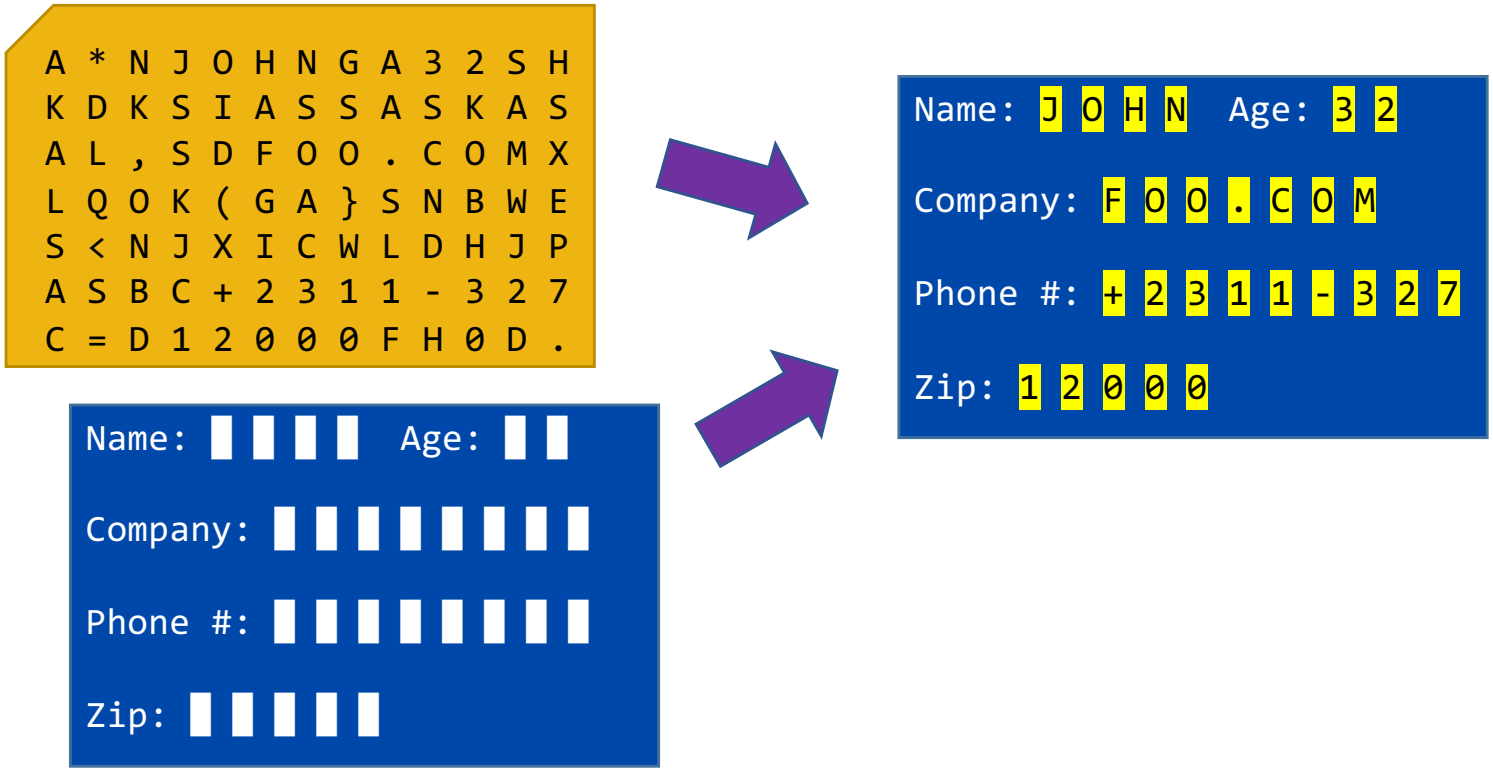
How Cobrix can help

- Decreasing human involvement
  - Fewer people are required…
  - No proprietary tool-specific knowledge
- Simplifying the manipulation of hierarchical structures
  - No intermediate data structures
- Providing scalability
- Open-source

Cobrix – a Spark data source



A copybook is a schema definition
A data file is a collection of binary records



Similar to IDLs

Thrift

```
struct Company {
  1: required i64          id,
  2: required string       name,
  3: optional list<string> contactPeople
}
```

protobuf
Protocol Buffers

```
message Company {
  required int64    id             = 1;
  required string   name           = 2;
  repeated string   contact_people = 3;
}
```

```
record Company {
  int64          id;
  string         name;
  array<string>  contactPeople;
}
```

COBOL

```
10 COMPANY.
  15 ID              PIC 9(12) COMP.
  15 NAME            PIC X(40).
  15 CONTACT-PEOPLE  PIC X(20)
       OCCURS 10.
```

Loading Mainframe Data

```
01 RECORD.
  05 COMPANY-ID     PIC 9(10).
  05 COMPANY-NAME   PIC X(40).
  05 ADDRESS        PIC X(60).
  05 REG-NUM        PIC X(8).
  05 ZIP            PIC X(6).
```

```
A * N J O H N G A 3 2 S H K D K S I
A S S A S K A S A L , S D F O O . C
O M X L Q O K ( G A } S N B W E S <
N J X I C W L D H J P A S B C + 2 3
```

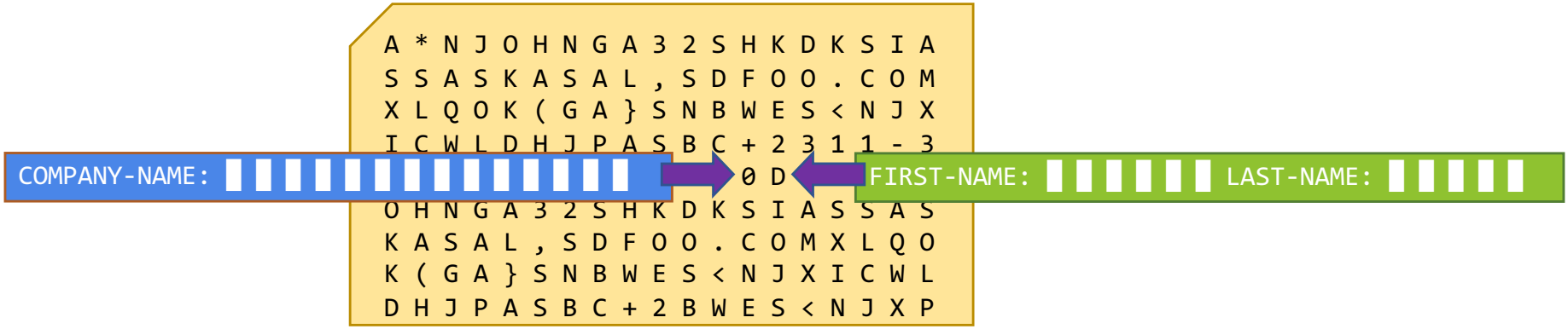
```
val df = spark
  .read
  .format("cobol")
  .option("copybook", "data/example.cob")
  .load("data/example")
```

COMPANY_ID	COMPANY_NAME	ADDRESS	REG_NUM	ZIP
100	ABCD Ltd.	10 Garden st.	8791237	03120
101	ZjkLPj	11 Park ave.	1233971	23111
102	Robotrd Inc.	12 Forest st.	0382979	12000
103	Xingzhoug	8 Mountst.	2389012	31222

Redefined Fields

- Redefined fields AKA
  - Unchecked unions
  - Untagged unions
  - Variant type fields
- Several fields occupy the same space

```
01 RECORD.
    05 IS-COMPANY                    PIC 9(1).
    05 COMPANY.
        10 COMPANY-NAME              PIC X(40).
    05 PERSON REDEFINES COMPANY.
        10 FIRST-NAME                PIC X(20).
        10 LAST-NAME                 PIC X(20).
    05 ADDRESS                       PIC X(50).
    05 ZIP                           PIC X(6).
```



Redefined Fields

- Cobrix applies all redefines for each record
- Some fields can clash
- It's up to the user to apply business logic to separate correct and wrong data

```
01 RECORD.
    05 IS-COMPANY                    PIC 9(1).
    05 COMPANY.
        10 COMPANY-NAME              PIC X(40).
    05 PERSON REDEFINES COMPANY.
        10 FIRST-NAME                PIC X(20).
        10 LAST-NAME                 PIC X(20).
    05 ADDRESS                       PIC X(50).
    05 ZIP                           PIC X(6).
```

IS_COMPANY	COMPANY	PERSON	ADDRESS	ZIP
1	{"COMPANY_NAME": "September Ltd."}	{"FIRST_NAME": "**Septem**", "LAST_NAME": "**ber Ltd.**"}	74 Lawn ave., Denver	39023
0	{"COMPANY_NAME": "**Beatrice Gagliano**"}	{"FIRST_NAME": "Beatrice", "LAST_NAME": "Gagliano"}	10 Garden str.	33113
1	{"COMPANY_NAME": "January Inc."}	{"FIRST_NAME": "**Januar**", "LAST_NAME": "**y Inc.**"}	122/1 Park ave.	31234

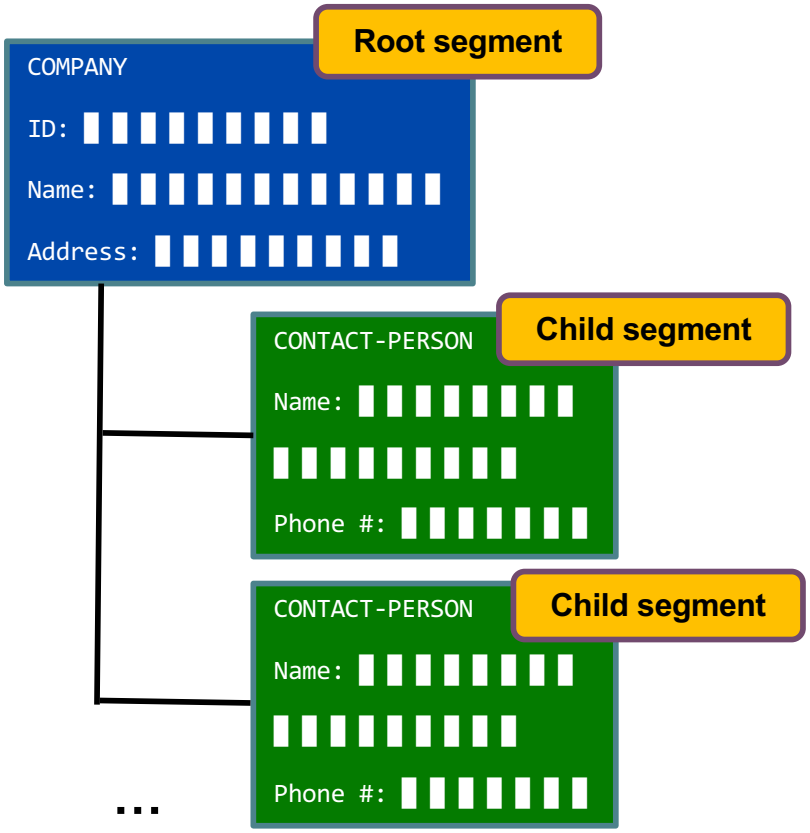
Redefined Fields clean up

```
df.select($"IS_COMPANY",
  when($"IS_COMPANY" === true, "COMPANY_NAME")
    .otherwise(null).as("COMPANY_NAME"),
  when($"IS_COMPANY" === false, "CONTACTS")
    .otherwise(null).as("FIRST_NAME")),
...
```

IS_COMPANY	COMPANY_NAME	FIRST_NAME	LAST_NAME	ADDRESS	ZIP
1	September Ltd.			74 Lawn ave., Denver	39023
0		Beatrice	Gagliano	10 Garden str.	33113
1	January Inc.			122/1 Park ave.	31234

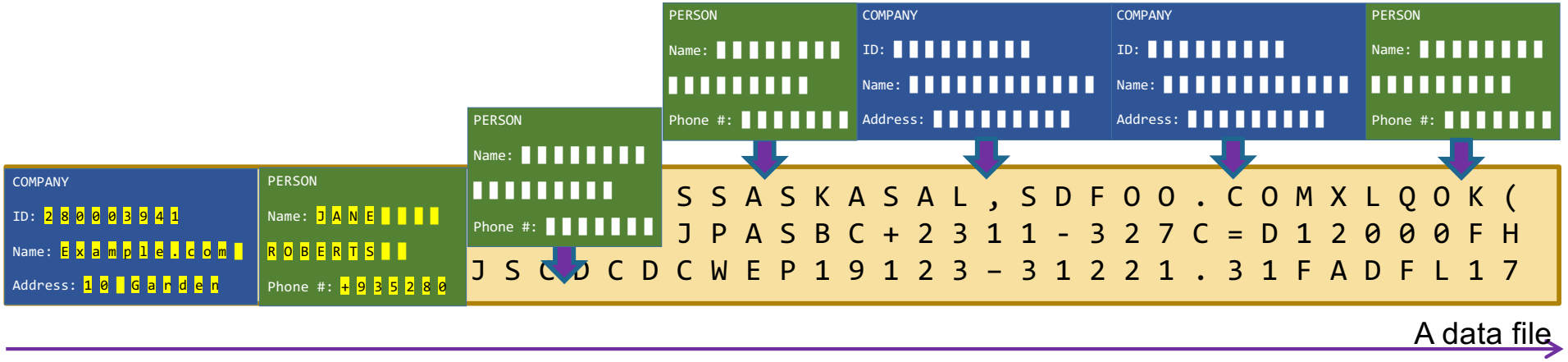
Hierarchical DBs

- Several record types
  - AKA segments
- Each segment type has its own schema
- Parent-child relationships between segments



Variable Length Records (VLRs)

- When transferred from a mainframe a hierarchical database becomes
  - A sequence of records
- To read next record a previous record should be read first
- A sequential format by it's nature



Step 1. Define a copybook

The combined copybook has to contain all the segments as redefined fields:

```
01  COMPANY-DETAILS.
        05  SEGMENT-ID          PIC X(5).
        05  COMPANY-ID          PIC X(10).
        05  COMPANY.
            10  NAME            PIC X(15).
            10  ADDRESS         PIC X(25).
            10  REG-NUM         PIC 9(8) COMP.
        05  CONTACT REDEFINES COMPANY.
            10  PHONE-NUMBER    PIC X(17).
            10  CONTACT-PERSON  PIC X(28).
```

Common fields

Segment 1

Segment 2

redefines

Step 2. Specify the VLR option

- **The code snippet for reading the data:**

```
val df = spark
  .read
  .format("cobol")
  .option("copybook", "/path/to/copybook.cpy")
  .option("is_record_sequence", "true")
  .load("examples/multisegment_data")
```

Step 3. Reading all the segments

- **The dataset for the whole copybook:**

SEGMENT_ID	COMPANY_ID	COMPANY	CONTACT
C	1005918818	[ABCD Ltd.]	[invalid]
P	1005918818	[invalid]	[Cliff Wallingford]
C	1036146222	[DEFG Ltd.]	[invalid]
P	1036146222	[invalid]	[Beatrice Gagliano]
C	1045855294	[Robotrd Inc.]	[invalid]
P	1045855294	[invalid]	[Doretha Wallingford]
P	1045855294	[invalid]	[Deshawn Benally]
P	1045855294	[invalid]	[Willis Tumlin]
C	1057751949	[Xingzhoug]	[invalid]
P	1057751949	[invalid]	[Mindy Boettcher]

Step 4. Reading root segments

- Filter segment #1 (companies)

```
val dfCompanies =
      df.filter($"SEGMENT_ID"=="C")
        .select($"COMPANY_ID",
                $"COMPANY.NAME".as($"COMPANY_NAME"),
                $"COMPANY.ADDRESS",
                $"COMPANY.REG_NUM")
```

Company_Id	Company_Name	Address	Reg_Num
100	ABCD Ltd.	10 Garden st.	8791237
101	ZjkLPj	11 Park ave.	1233971
102	Robotrd Inc.	12 Forest st.	0382979
103	Xingzhoug	8 Mountst.	2389012
104	Example.co	123 Tech str.	3129001

Step 5. Reading child segments

- Filter segment #2 (people) using segment filter pushdown

```
val dfContacts = spark.read.format("cobol")…
  .option("segment_filter”, "P")
  .select($"COMPANY_ID",
          $”CONTACT.CONTACT_PERSON",
          $”CONTACT.PHONE_NUMBER")
```

Company_Id	Contact_Person	Phone_Number
100	Marry	+32186331
100	Colyn	+23769123
102	Robert	+12389679
102	Teresa	+32187912
102	Laura	+42198723

Step 6. Joining two segments by Company_Id

Company_Id	Company_Name	Address	Reg_Num
100	ABCD Ltd.	10 Garden st.	8791237
101	ZjkLPj	11 Park ave.	1233971
102	Robotrd Inc.	12 Forest st.	0382979
103	Xingzhoug	8 Mountst.	2389012
104	Example.co	123 Tech str.	3129001

Company_Id	Contact_Person	Phone_Number
100	Marry	+32186331
100	Colyn	+23769123
102	Robert	+12389679
102	Teresa	+32187912
102	Laura	+42198723

Company_Id	Company_Name	Address	Reg_Num	Contact_Person	Phone_Number
100	ABCD Ltd.	10 Garden st.	8791237	Marry	+32186331
100	ABCD Ltd.	10 Garden st.	8791237	Colyn	+23769123
102	Robotrd Inc.	12 Forest st.	0382979	Robert	+12389679
102	Robotrd Inc.	12 Forest st.	0382979	Teresa	+32187912
102	Robotrd Inc.	12 Forest st.	0382979	Laura	+42198723

Step 7. Denormalize data

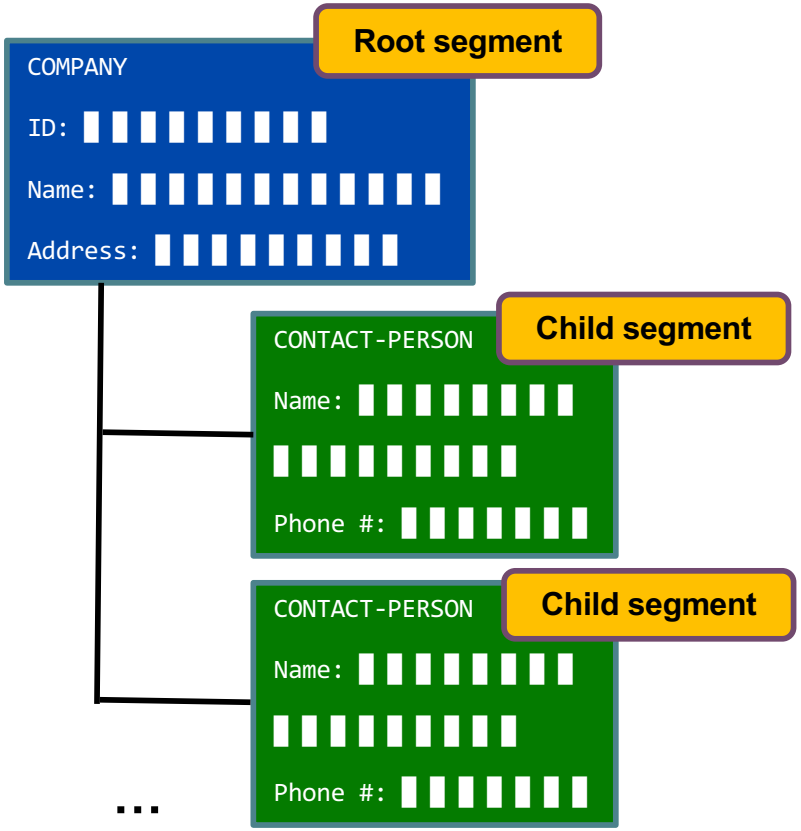
- The joined table can also be denormalized for document storage

```
val dfCombined =
    dfJoined
     .groupBy($"COMPANY_ID",
              $"COMPANY_NAME",
              $"ADDRESS",
              $"REG_NUM")
     .agg(
       collect_list(
         struct($"CONTACT_PERSON",
                $"PHONE_NUMBER"))
       .as("CONTACTS"))
```


```
{
 "COMPANY_ID": "8216281722",
 "COMPANY_NAME": "ABCD Ltd.",
 "ADDRESS": "74 Lawn ave., New York",
 ”REG_NUM": "33718594",
 "CONTACTS": [
   {
     "CONTACT_PERSON": "Cassey Norgard",
     "PHONE_NUMBER": "+(595) 641 62 32"
   },
   {
     "CONTACT_PERSON": "Verdie Deveau",
     "PHONE_NUMBER": "+(721) 636 72 35"
   },
   {
     "CONTACT_PERSON": "Otelia Batman",
     "PHONE_NUMBER": "+(813) 342 66 28"
   }
 ]
}
```

Restore parent-child relationships

- In our example we had COMPANY_ID field that is present in all segments
- In real copybooks this is not the case
- What can we do?



Id Generation

- If COMPANY_ID is not part of all segments
  Cobrix can generate it for you

No COMPANY-ID

```
01  COMPANY-DETAILS.
   05  SEGMENT-ID                PIC X(5).
   05  COMPANY.
      10  NAME                   PIC X(15).
      10  ADDRESS                PIC X(25).
      10  REG-NUM                PIC 9(8) COMP.
   05  CONTACT REDEFINES COMPANY.
      10  PHONE-NUMBER           PIC X(17).
      10  CONTACT-PERSON         PIC X(28).
```

```
val df = spark
  .read
  .format("cobol")
  .option("copybook", "/path/to/copybook.cpy")
  .option("is_record_sequence", "true")
  .option("segment_field", "SEGMENT-ID")
  .option("segment_id_level0", "C")
  .option("segment_id_prefix", "ID")
  .load("examples/multisegment_data")
```

Id Generation

- **Seg0_Id** can be used to restore parent-child relationship between segments

No COMPANY-ID

```
01  COMPANY-DETAILS.
   05  SEGMENT-ID              PIC X(5).
   05  COMPANY.
      10  NAME                 PIC X(15).
      10  ADDRESS              PIC X(25).
      10  REG-NUM              PIC 9(8) COMP.
   05  CONTACT REDEFINES COMPANY.
      10  PHONE-NUMBER         PIC X(17).
      10  CONTACT-PERSON       PIC X(28).
```

SEGMENT_ID	Seg0_Id	COMPANY	CONTACT
C	ID_0_0	[ABCD Ltd.]	[invalid]
P	ID_0_0	[invalid]	[Cliff Wallingford]
C	ID_0_2	[DEFG Ltd.]	[invalid]
P	ID_0_2	[invalid]	[Beatrice Gagliano]
C	ID_0_4	[Robotrd Inc.]	[invalid]
P	ID_0_4	[invalid]	[Doretha Wallingford]

Segment-Redefine Filter Pushdown

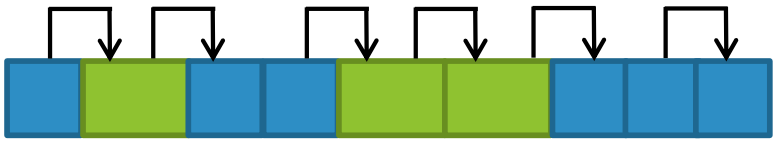
- Allows to resolve segment redefines on parsing stage for performance

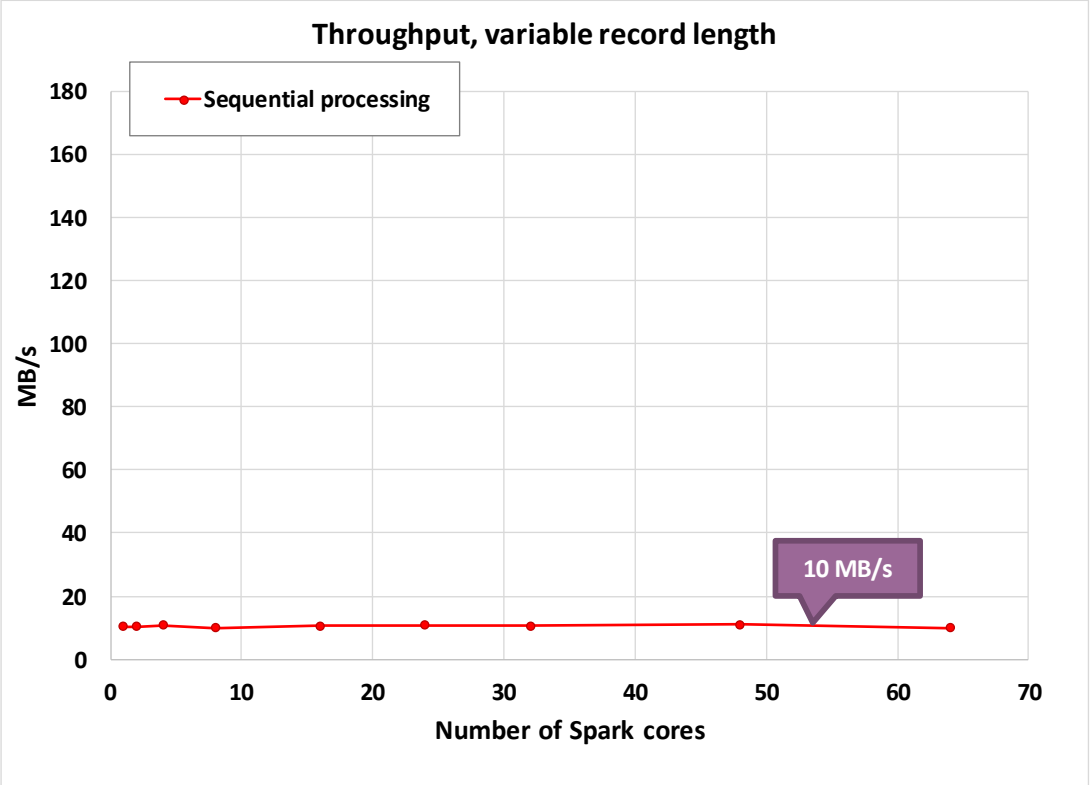
```
val df = spark
  .read
  .format("cobol")
  .option("copybook", "/path/to/copybook.cpy")
  .option("is_record_sequence", "true")
  .option("segment_field", "SEGMENT-ID")
  .option("redefine_segment_id_map:0", ”COMPANY => C")
  .option("redefine_segment_id_map:1", ”CONTACT => P")
  .load("examples/multisegment_data")
```

Performance challenge of VLRs

- **Naturally sequential files**
  - To read next record the prior
    record need to be read first
- **Each record had a length field**
  - Acts as a pointer to the next record
- **No record delimiter when reading a file from the middle**

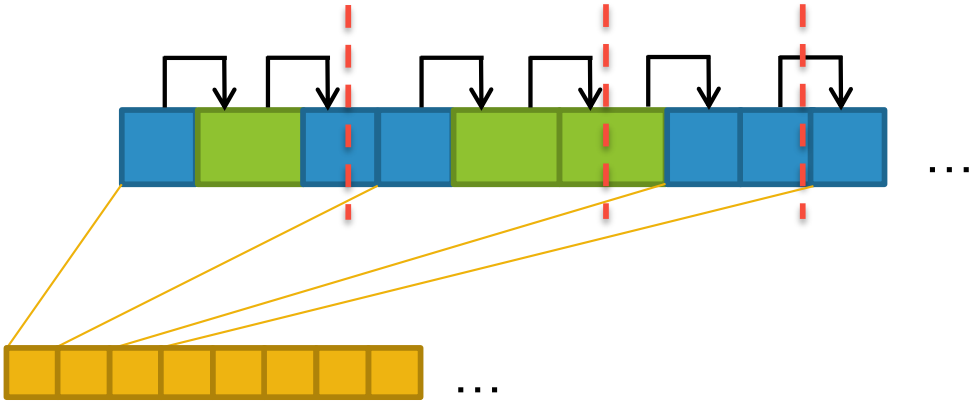
VLR structure





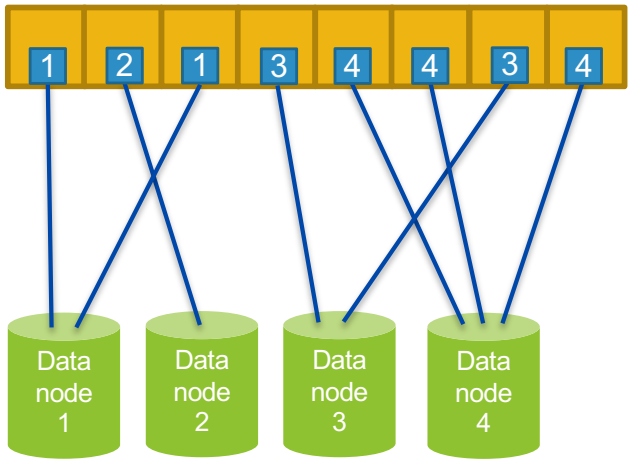
3-phase processing

- **Phase 1**. Extract record boundaries into a sparse index
  - Index chunk size is aligned to HDFS block size



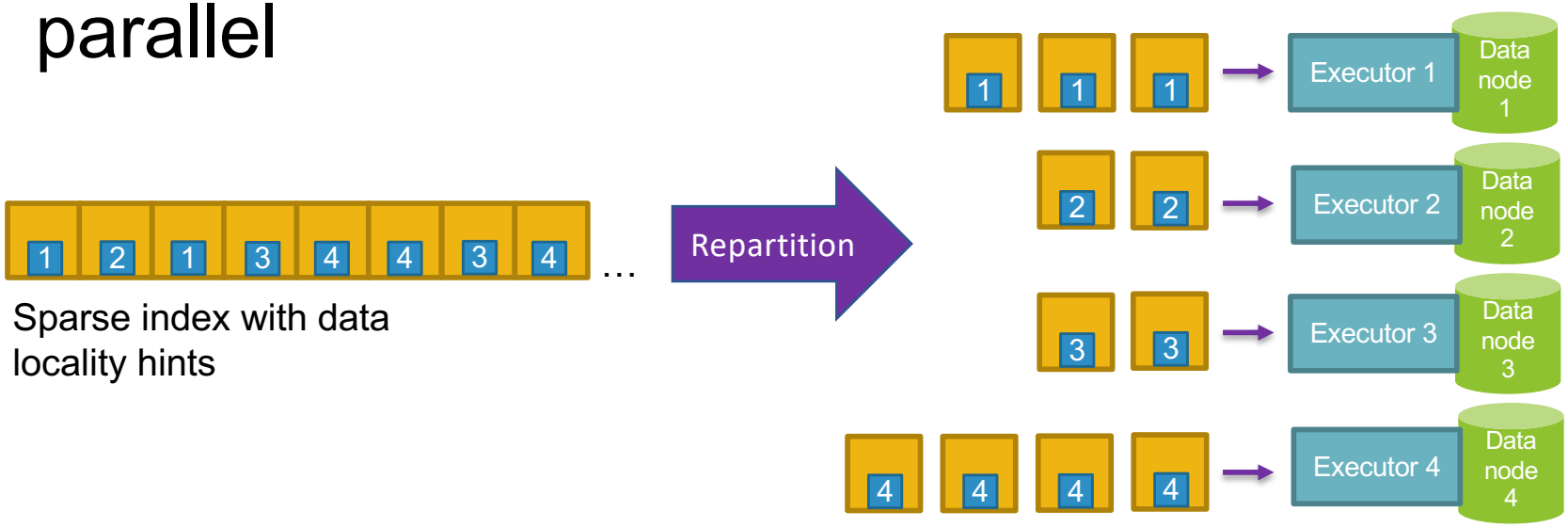
3-phase processing

- **Phase 2**. Query HDFS blocks for each index chunk for data locality awareness
  - Add localist 'hints' to the index elements



3-phase processing

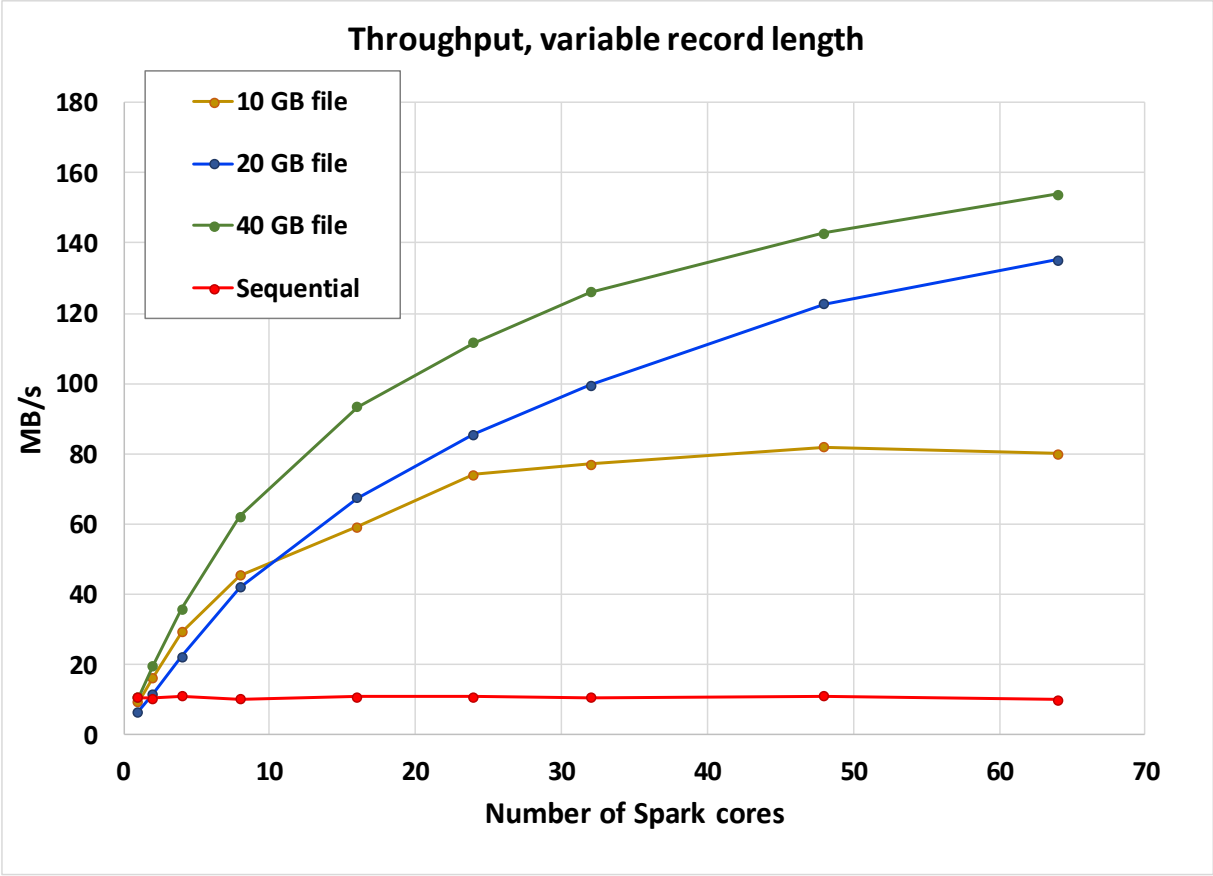
- **Phase 3**. Use the sparse index and the data locality hints to process the mainframe file in parallel



Throughput with sparse indexes

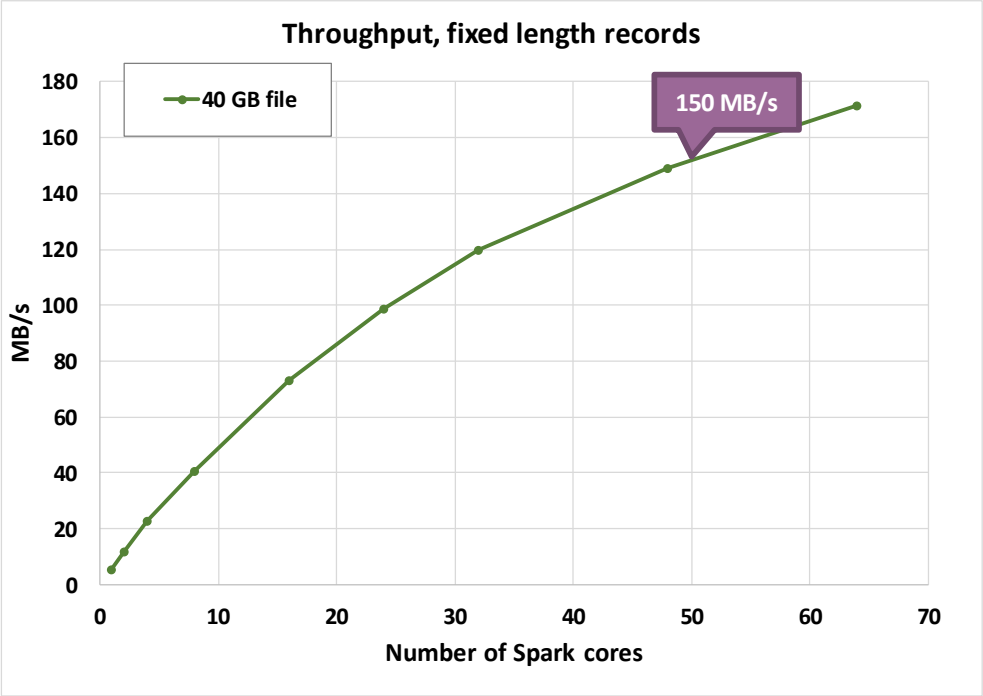
- Experiments were ran on our lab cluster
- 4 nodes
- 380 cores
- 10 Gbit network

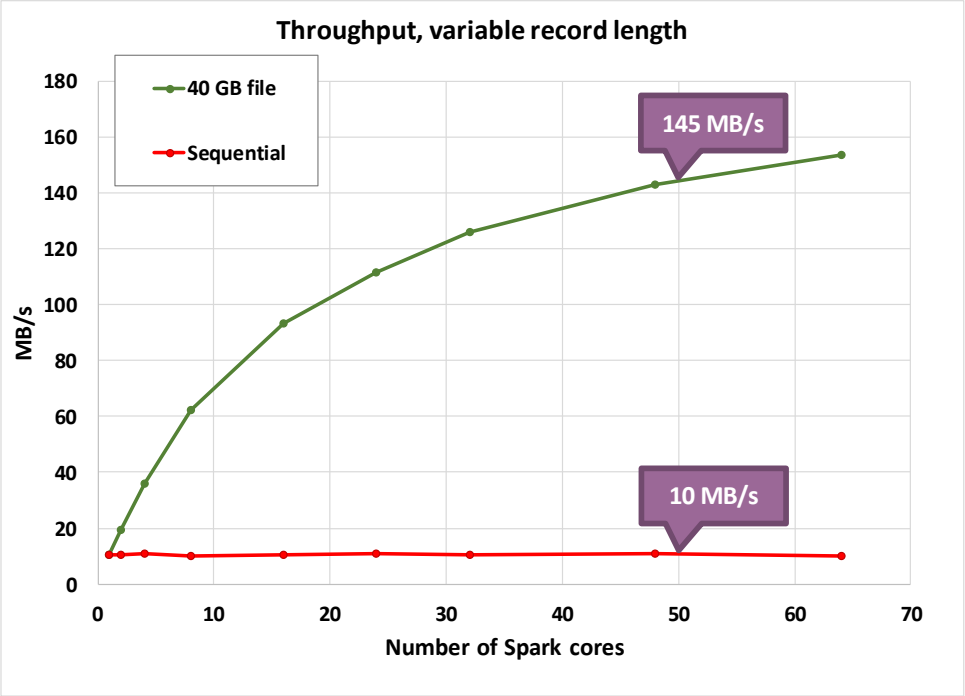
- Scalable – for bigger files when using more executors the throughput is bigger



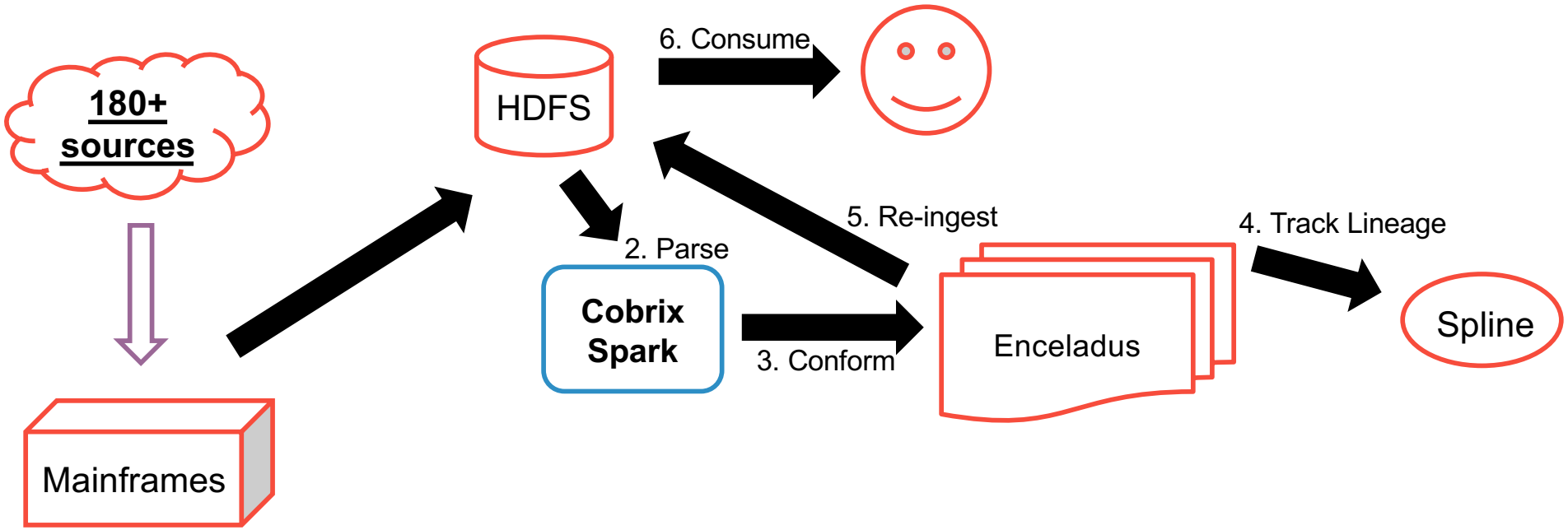
Comparison versus fixed length records

- Distribution and locality is handled completely by Spark
- Parallelism is achieved using sparse indexes

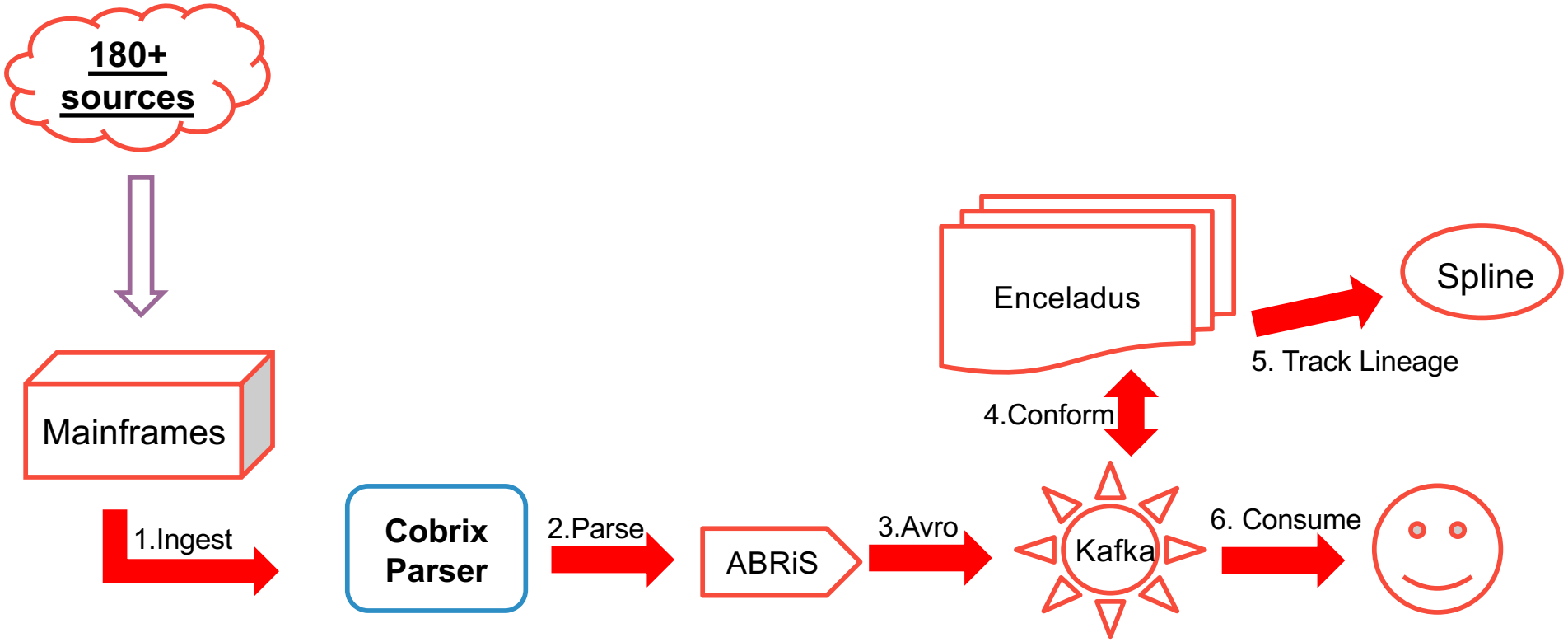




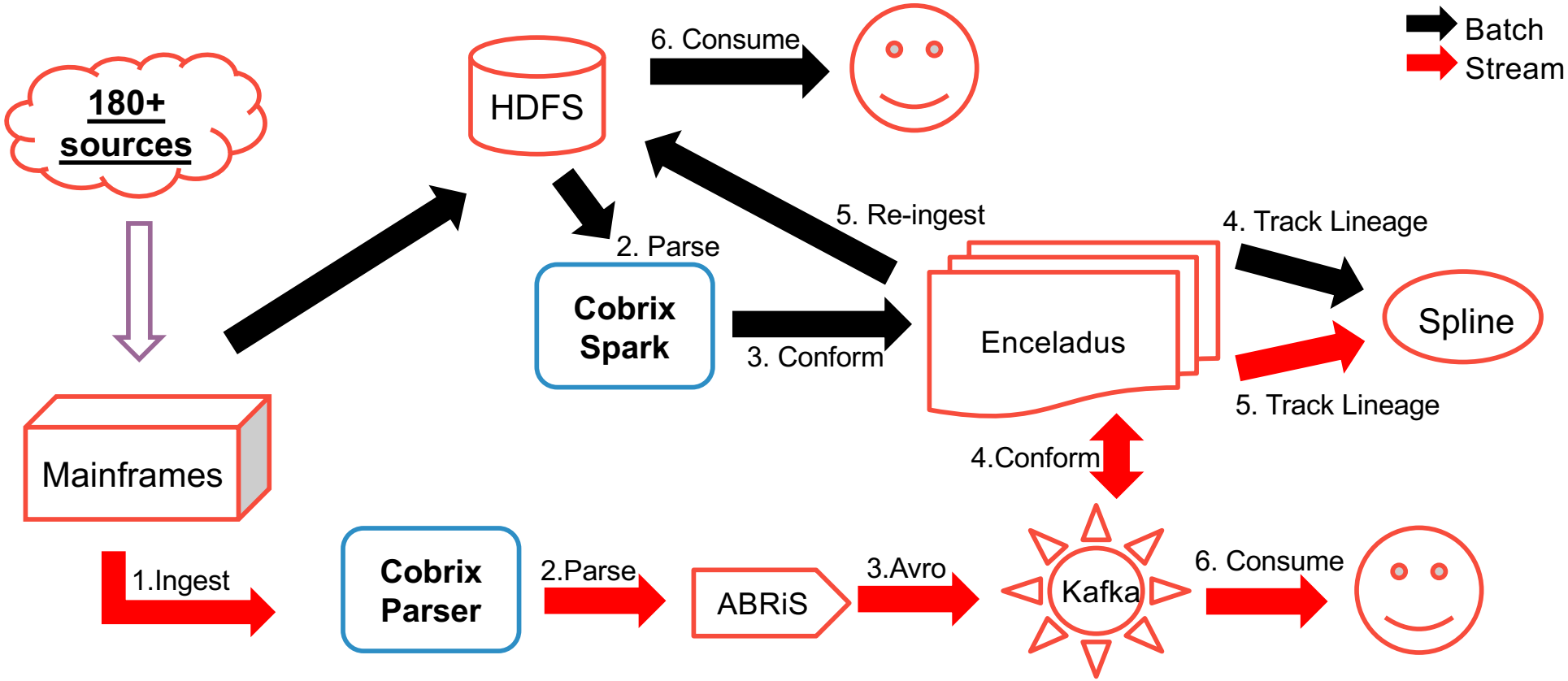
Cobrix in ABSA Data Infrastructure - Batch



Cobrix in ABSA Data Infrastructure - Stream



Cobrix in ABSA Data Infrastructure



Acknowledgment

- Big thanks to **Felipe Melo**, a co-author of the project who couldn't make it to the summit!
- Thanks to the following people the project was made possible and for all the help along the way:
  - **Andrew Baker, Francois Cillers, Adam Smyczek, Peter Moon, Clifford Lategan, Rekha Gorantla, Mohit Suryavanshi, Niel Steyn**
- Thanks to the authors of the original COBOL parser:
  - **Ian De Beer, Rikus de Milander** (https://github.com/zenaptix-lab/copybookStreams)

Your Contribution is Welcome

- Combine expertise to make access mainframe data in Hadoop seamless
- Our goal is to support the widest range of use cases possible

- Report a bug 😀
- Request new feature 😀
- Create a pull request 😀 👍 🎉 🍺

Our home: https://github.com/AbsaOSS/cobrix

SPARK+AI SUMMIT 2019

DON'T FORGET TO RATE AND REVIEW THE SESSIONS

SEARCH SPARK + AI SUMMIT

Download on the App Store

GET IT ON Google Play