Bring Your Language (and libraries) to Your Data

Stefan Mandl, Oleksandr Kozachuk, Jens Graupmann

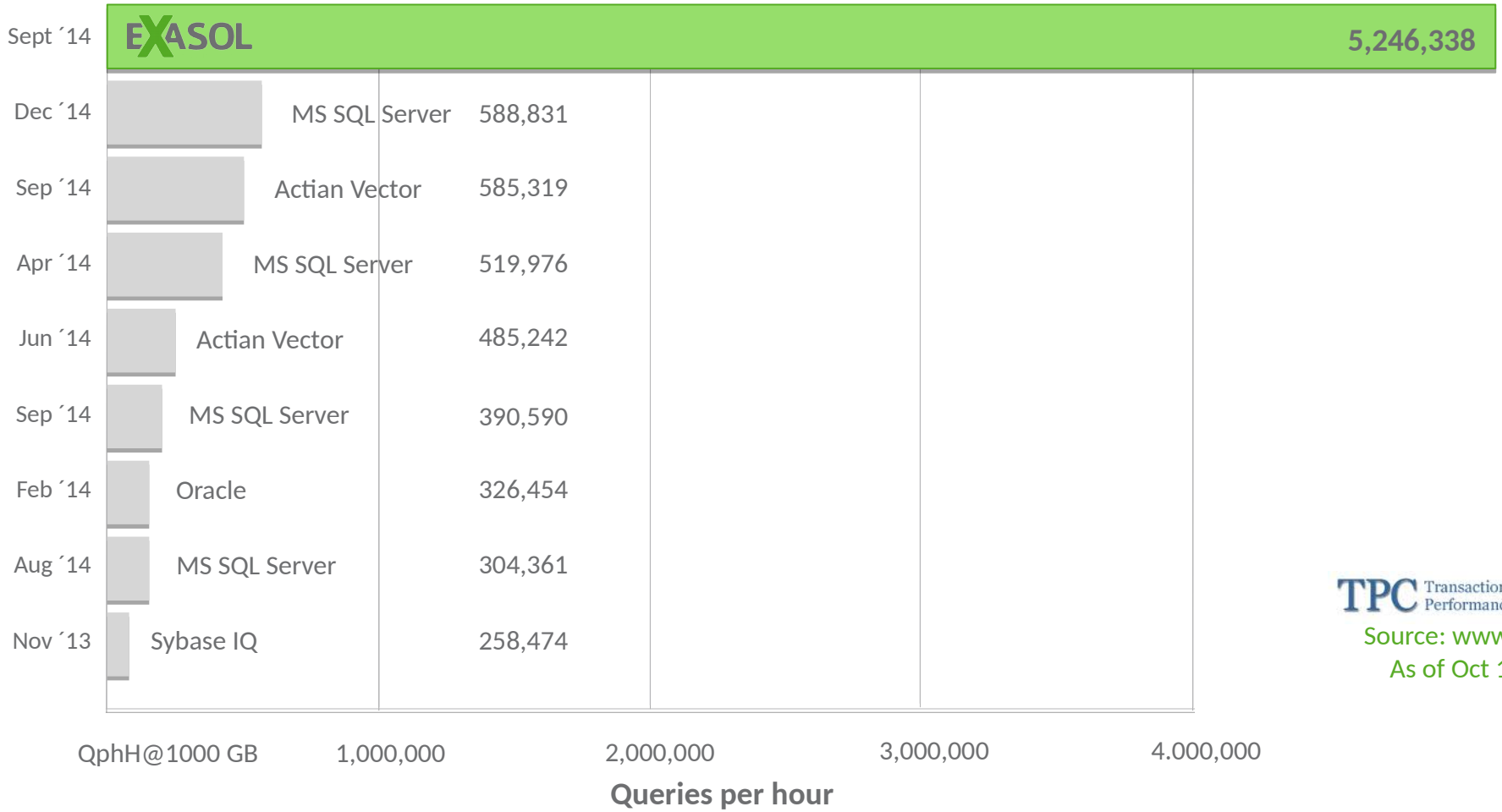
What is EXASOL?

- **a column store, massively parallel processing (MPP), in-memory analytic database**
  - modern software designed for analytics
  - runs on standard x86 hardware
  - ACID compliance
  - uses standard SQL language (with optional extensions)
  - suitable for any scale of data & any number of users
  - mature, proven & very cost effective
  - quick to implement & easy to operate

The World's Fastest Analytic Database

We are the benchmark leader (TPC-H)

On 1TB data - an order of magnitude faster than the nearest rival



TPC Transaction Processing Performance Council

Source: www.tpc.org
As of Oct 13, 2015

Full report can be viewed here: http://www.tpc.org/tpch/results/tpch_perf_results.asp

History of innovation: MPP, in-memory, performance

90ies
Early **Research Success** (University Erlangen-Nürnberg)

EXASOL is founded
2000

2006
Pilot Customer
Karstadt-Quelle uses EXASOL in production

Record in **TPC-H Benchmark** (“Oracle dethroned“)
2008

2010
Most successful vendor of analytic database systems **in Germany** (**BARC**)

Inclusion in **Gartner**’s “Magic Quadrant for DW DBMS Systems“
2012

2014
Successful global **expansion**, **100TB TPC-H** benchmark

100+ customers across 16 countries/5 continents
2015

100+ customers in 16 countries/5 continents and counting…

EXASOL

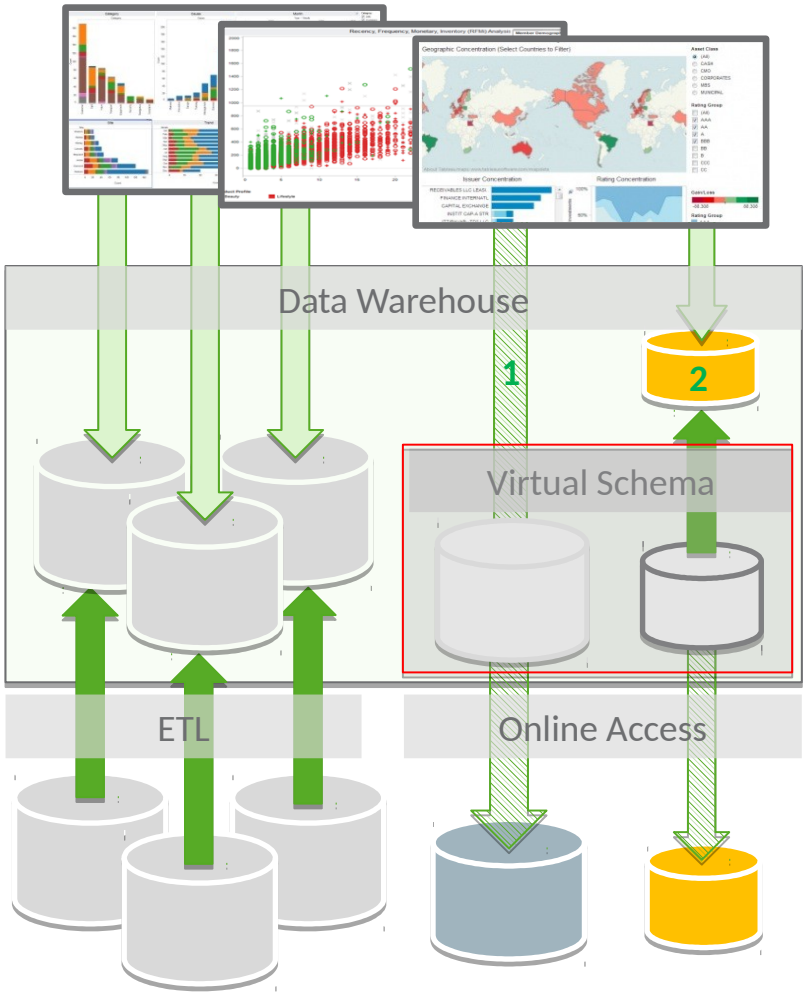
CCV: Analytics that redefines the customer experience

- **Leading supplier of payment solutions** in Europe
- Turned to EXASOL to **guarantee safe and fast transactions at payment terminals**
- Also use EXASOL to **analyze product development, product and process quality**
- **Speed, scalability** and great **EXASOL support** were defining factors for CCV
- Use EXASOL to run **in-depth analysis with "R" and user-defined functions (UDFs)**
- Company benefits from an **agile development environment**

let's make payment happen

Logical Data Warehouse with Virtual Schemas

(Transparent ecosystem integration framework)

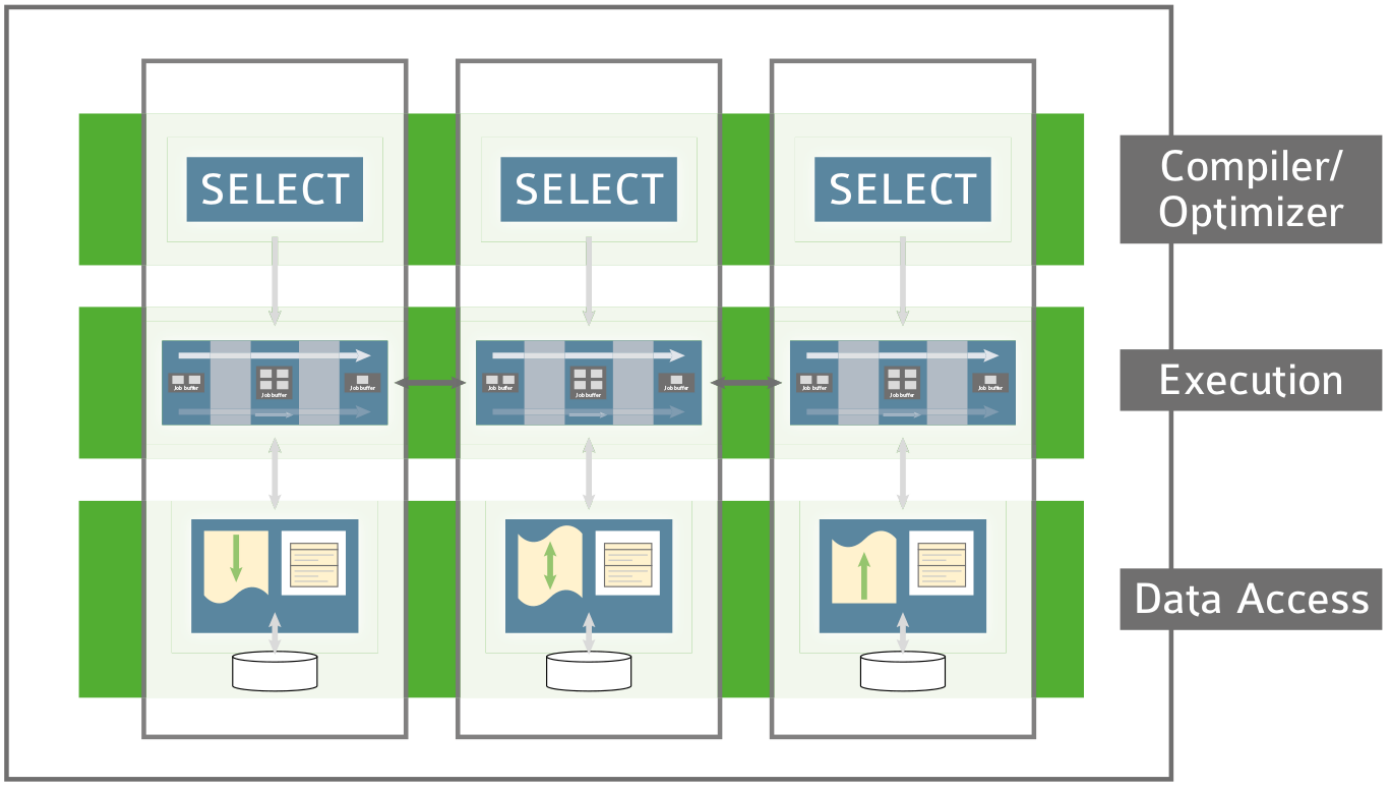


Virtual Schemas

- Only metadata of virtually connected data sources are visible.
- Whether virtual or "physical": fully transparent from application perspective.
- Access to these virtual schemas is dynamically forwarded to the connected data sources (1). Data is transferred on demand.
- If required, the data can be physically replicated into the DWH on demand without the need for additional ETL tools (2).
- Coexistence with ETL

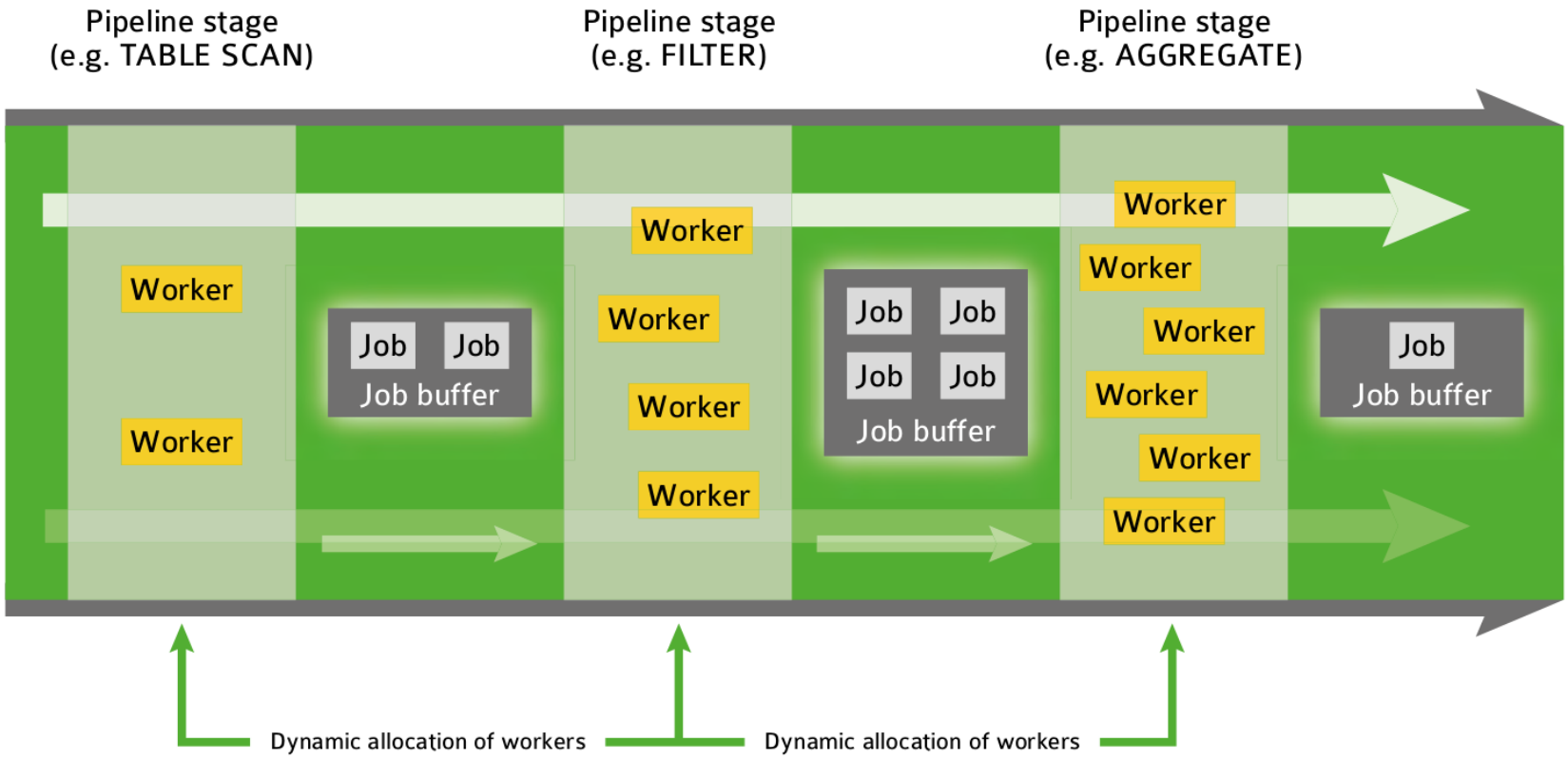
- ✓ Agile access to most recent information
- ✓ No/reduced redundancy
- ✓ Less ETL-jobs
- ✓ No waste of disk space

Execution model



- Shared nothing

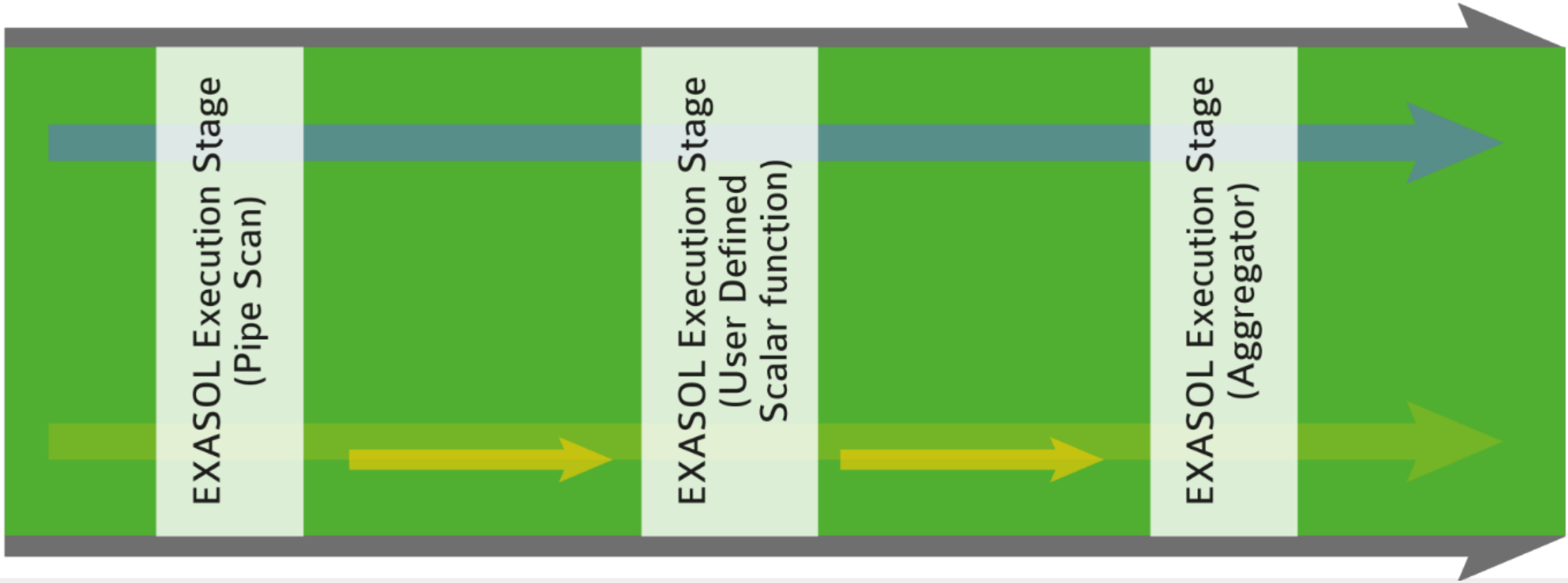
Execution model



Trade-Offs when Designing a UDF Feature

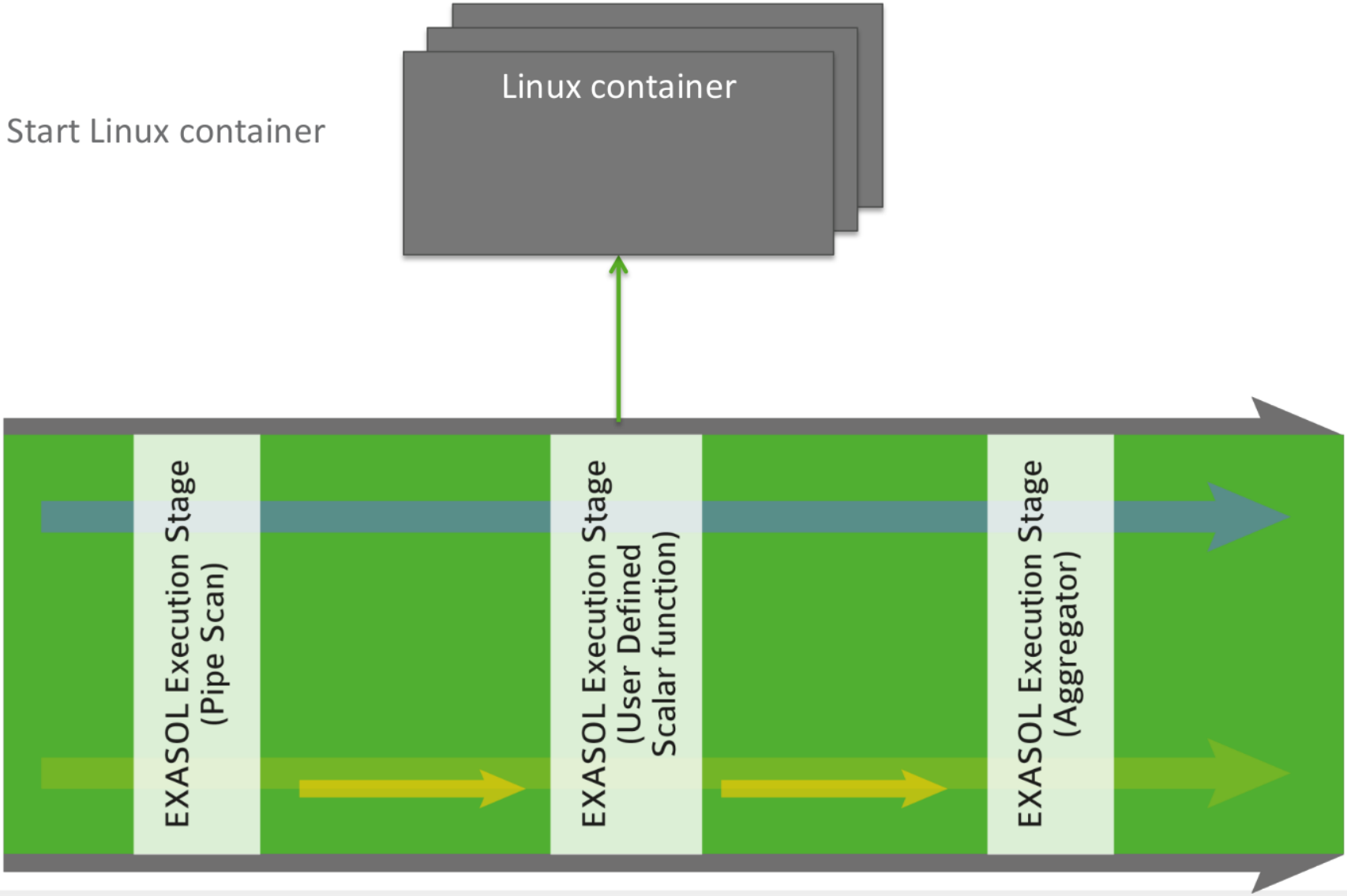
- **The language**
  - Role-your-own
  - Some form of PL/SQL
  - **General purpose programming languages**
- **Integration with the DB**
  - Tight coupling
  - **Loose coupling**

UDFs in EXASOL



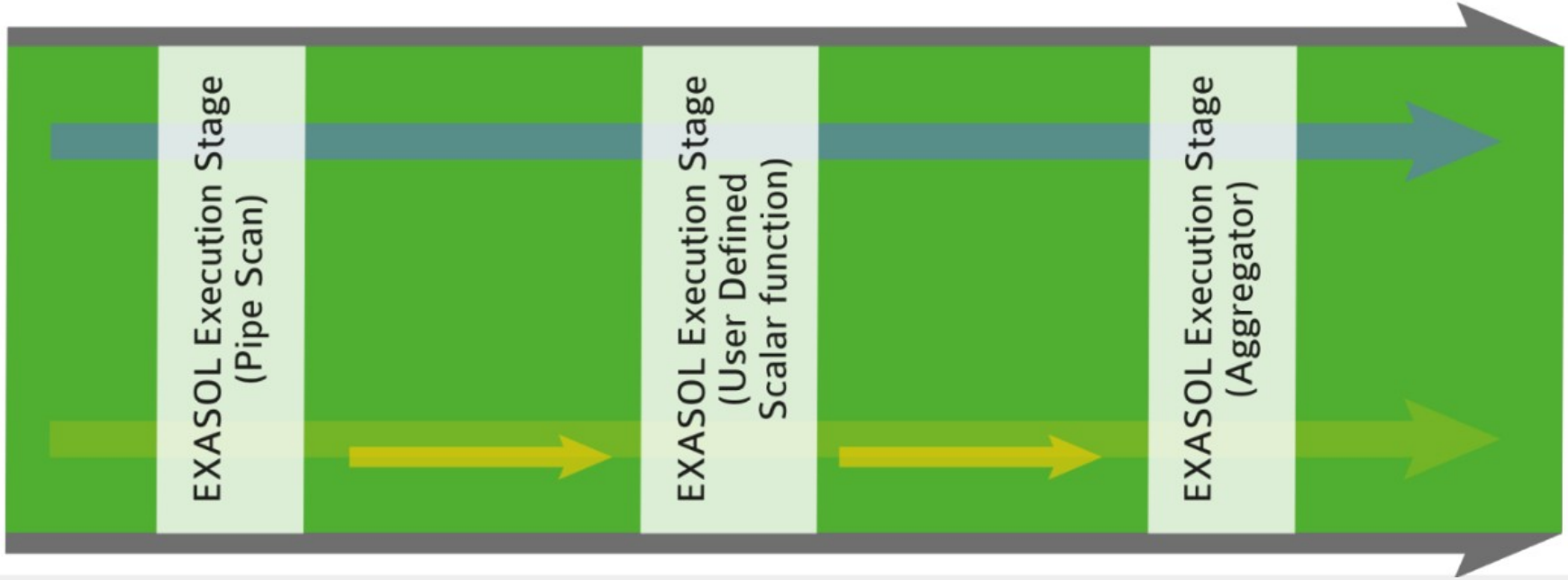
UDFs in EXASOL

1) Start Linux container



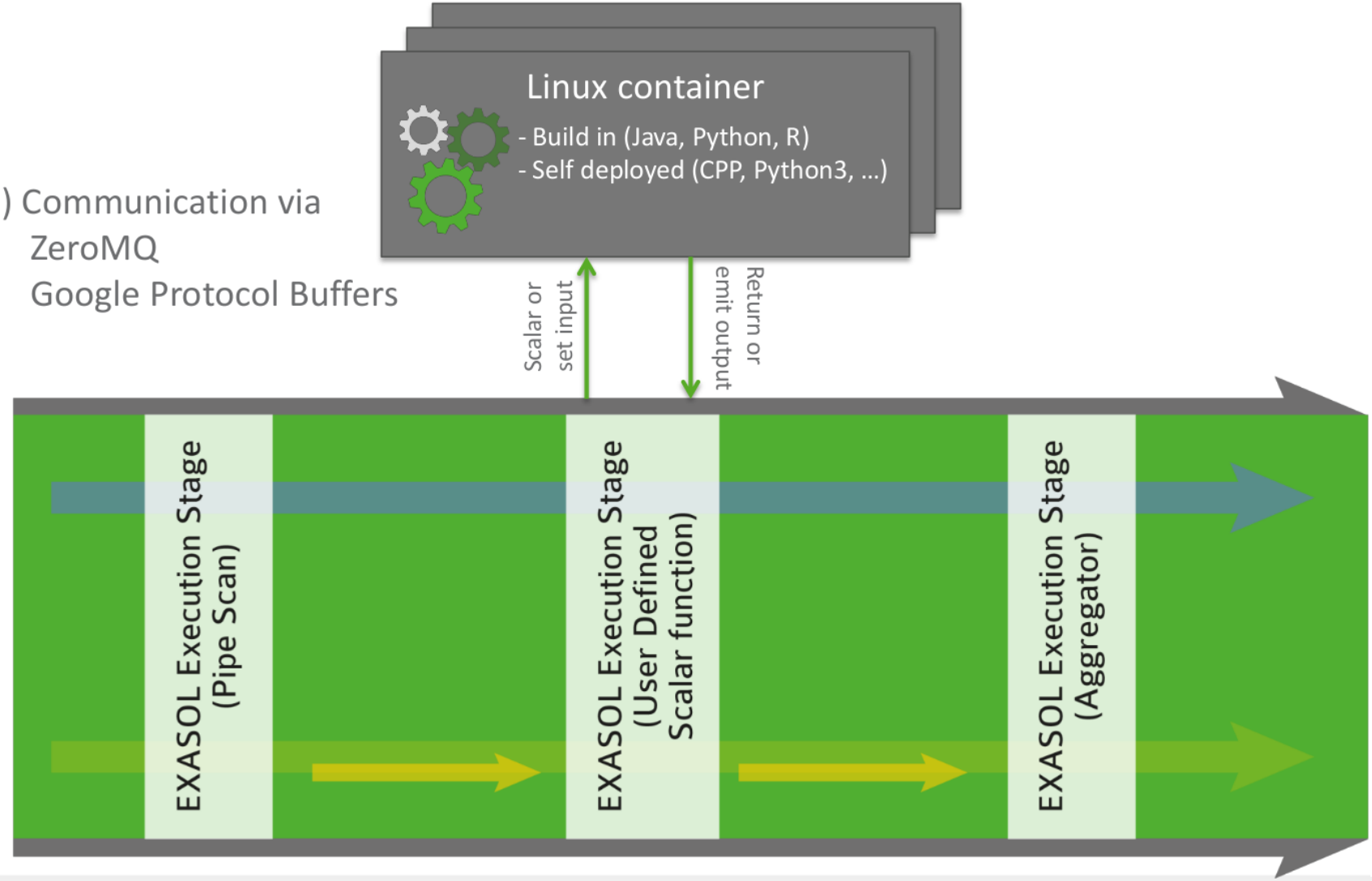
UDFs in EXASOL

2) Start script client



Bring Your Own Language

3) Communication via
- ZeroMQ
- Google Protocol Buffers



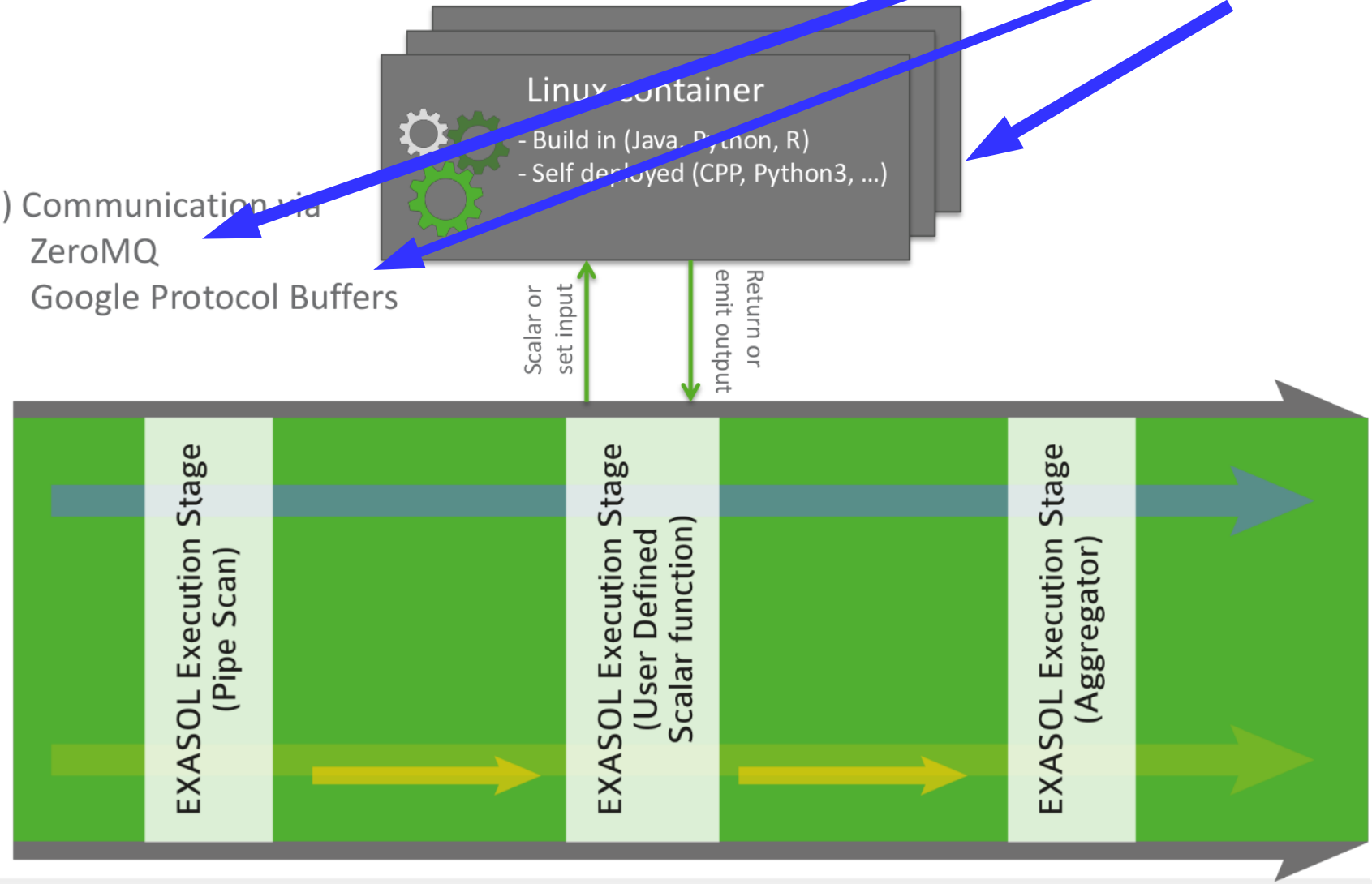
UDF-Types and Parallelism

	Scalar output	Set output
Scalar input	Scalar function	Table generating function
Set input	Aggregate function	m:n mapping Special case: AF

- **Parallelism**
  - For scalar input: start as many UDF containers as convenient (e.g. one per core), and send arbitrarily buffered rows to the UDFs. For the user it appears as one row at a time.
  - For set input: start as many UDF containers as convenient. Data is partitioned per group and processed by UDF per group.
  - → The level of parallelism can be controlled by appropriate GROUP BY clauses

Bring Your Own Language

Open source



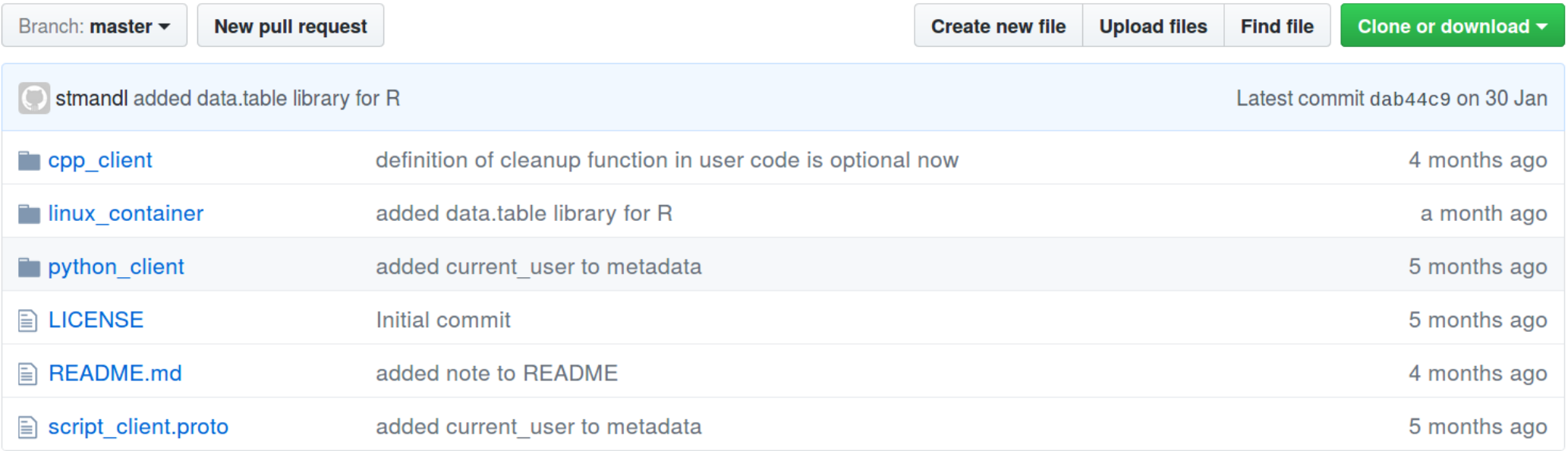
3) Communication via
- ZeroMQ
- Google Protocol Buffers

Bring Your Own Language

- **More precisely**
  - Open source the involved components
  - Provide means to store custom compontents in the cluster
  - Provide means to link the new components to SQL

Open Sourcing the Involved Components

https://github.com/EXASOL/script-languages



Open UDF Language Protocol

MT_CLIENT	The script language implementation is alive and requests more information
MT_INFO	Basic information about the EXASOL system and cluster configuration and the UDF script code
MT_META	Names and data types of the data to send between EXASOL and the script language implementation
MT_CLOSE	Terminates the connection to EXASOL
MT_IMPORT	Request the source code of other scripts or information stored in CONNECTION objects
MT_NEXT	Request more data to be sent from EXASOL to the UDF
MT_RESET	Restart the input data iterator to the beginning
MT_EMIT	Send results from the UDF to EXASOL
MT_RUN	Change status to indicate the start of data transfers
MT_DONE	Indicate that the UDF will send no more results for the current group of data
MT_CLEANUP	Send to indicate that no more groups of data will have to be processed by the script language implementation and that it may stop
MT_FINISHED	Sent when the script language implementation successfully stopped
MT_CALL	Used to call a certain function in the UDF when in Single-Call mode
MT_RETURN	Used to send the result of the Single-Call function call

```
// --- data definition -------------------------------------------------------

enum column_type {
    PB_UNSUPPORTED = 0;  // represents following SQL types:
    PB_DOUBLE = 1;       // FLOAT
    PB_INT32 = 2;        // DECIMAL(4, 0)
    PB_INT64 = 3;        // DECIMAL(8, 0)
    PB_NUMERIC = 4;      // all other numeric types
    PB_TIMESTAMP = 5;    // TIMESTAMP
    PB_DATE = 6;         // DATE
    PB_STRING = 7;       // CHAR or VARCHAR
    PB_BOOLEAN = 8;      // BOOL
}

// type of iteration:
//   scalar, returns  -> PB_EXACTLY_ONCE
//   set, emits       -> PB_MULTIPLE
enum iter_type { PB_EXACTLY_ONCE = 1; PB_MULTIPLE = 2; };

message exascript_metadata {
    message column_definition {
        // Name of column as UTF-8 String
        required string name = 1;
        // Type of column (see "enum column_type")
        optional column_type type = 2;
        // Type name how it is shown in EXASolution
        required string type_name = 3;
        // Size of Type, for CHAR and VARCHAR types
        optional uint32 size = 4;
        // Precision, for DECIMAL types
        optional uint32 precision = 5;
        // Scale, for DECIMAL types
        optional uint32 scale = 6;
    }
    required iter_type input_iter_type = 1;
    required iter_type output_iter_type = 2;

    repeated column_definition input_columns = 3;
    repeated column_definition output_columns = 4;
```

BucketFS: Store arbitrary data in the cluster

Node 1

Node 2

Node N

- A filesystem that is automatically synchronized on all nodes
- Support straightforward REST API for external or write access
- Manages non-transactional resources (models/libraries/languages)
- In UDFs: Read-Only access via filesystem

Defining Languages in SQL Sesssions

```
create or replace PYTHON set script train_tree(tree_name varchar(1000),
                                               x1 int, x2 int, x3 int, y int)
returns varchar(1000)
as
from sklearn import tree
from sklearn.datasets import make_blobs
import pickle
import redis
redis = redis.StrictRedis(host="[Hostname or IP of your redis DB]", port=6379, db=0)

def run(ctx):
    X = []
    Y = []
    tree_name = None
    while True:    # read all the data from the group
        if not tree_name: tree_name = ctx.tree_name
        assert(tree_name == ctx.tree_name)
        X.append([ctx.x1, ctx.x2, ctx.x3])
        Y.append(ctx.y)
        if not ctx.next(): break
    clf = tree.DecisionTreeClassifier()
    clf = clf.fit(X, Y)
    redis.set(tree_name, pickle.dumps(clf))
    return tree_name
/
```

Defining Languages in SQL Sesssions

```
PYTHON=localzmq+protobuf:///bfsdefault/default/EXAClusterOS/ScriptLanguages-6.0.0?lang=python#buckets/bfsdefault/default/EXASolution-6.0.0/exaudfclient
```

Defining Languages in SQL Sesssions

```
PYTHON=
    localzmq+protobuf:///

    bfsdefault/default/EXAClusterOS/ScriptLanguages-6.0.0

    ?lang=python

    #buckets/bfsdefault/default/EXASolution-6.0.0/exaudfclient
```

Use Case: Supply C++ language support

- Strategy:
  - Compile the UDF‘s code on the fly
  - Dynamically link and call the generated code
- Alternative:
  - Precompile all desired functions „into the language implementation“
  - At runtime: only dispatch to the required function implementation

Use Case: Supply C++ language support

Recipe:

- Download Linux Container from BucketFS
- Import into Docker
- Mount your source code
- Run docker
- Build your container
- Package as archive
- Upload into cluster
- Define it as new language in SQL

Use Case: Supply C++ language support

```
ALTER SESSION SET SCRIPT_LANGUAGES='…
…  …     …  …     …
CPP=localzmq+protobuf:///bfsdefault/de
fault/EXAClusterOS/ScriptLanguages-
6.0.0#buckets/bfsdefault/cpp/cppclient
/cppclient'
```

Use Case: Supply C++ language support

```
CREATE cpp SCALAR SCRIPT duplicateAndScale(n INT, x DOUBLE,
                                           y DOUBLE, z DOUBLE)
EMITS (x DOUBLE, y DOUBLE, z DOUBLE) AS
%compilerflags -lblas;
#include <cblas.h>
using namespace UDFClient;

void run_cpp(Metadata* meta, InputTable* in, OutputTable* out) {
   double x[] = {in->getDouble(1), in->getDouble(2),
                 in->getDouble(3)};
   cblas_dscal(3, 4.323, x, 1);

   for (size_t n = 0; n < in->getInt64(0); ++n) {
      for (size_t i = 0; i < 3; ++i)
         out->setDouble(i,x[i]);
      out->next();
   }
}
/
```

Use Case: Supply C++ language support

```
SQL_EXA> SELECT duplicateAndScale(3,1,2,3);
EXA: SELECT duplicateAndScale(3,1,2,3);


X                  Y                  Z
----------------- ----------------- -----------------
            4.323 8.646000000000001            12.969
            4.323 8.646000000000001            12.969
            4.323 8.646000000000001            12.969

3 rows in resultset.
```

Use Case 2 – Supply a new R library from CRAN

Recipe:

- Download Linux Container from BucketFS
- Import into Docker
- Mount your a target directory
- Run docker, run R
- Use R to install the library
- Package as archive
- Upload into cluster
- Use it in R UDFs

Use Case 2 – Supply a new R library

```
$> docker import http://10.60.1.11:2580/default/EXAClusterOS/ScriptLanguages-6.0.0.tar.gz
   rphonenumber
$> mkdir r_pkg
$> docker run -v `pwd`/r_pkg:/r_pkg  --name=rphonenumber  -it rphonenumber /bin/bash
```

```
$> export R_LIBS="/r_pkg/"
$> R
```

```
> install.packages('phonenumber', repos="http://cran.r-project.org")
```

```
Installing package into '/r_pkg'
(as 'lib' is unspecified)
trying URL 'http://cran.r-project.org/src/contrib/phonenumber_0.2.2.tar.gz'
Content type 'application/x-gzip' length 10516 bytes (10 KB)
==================================================
downloaded 10 KB

* installing *source* package 'phonenumber' ...
** package 'phonenumber' successfully unpacked and MD5 sums checked
** R
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** installing vignettes
** testing if installed package can be loaded
* DONE (phonenumber)

The downloaded source packages are in
        '/tmp/RtmpZyGTtk/downloaded_packages'
```

Use Case 2 – Supply a new R library

```
$> tar zcf r_pkg.tgz r_pkg/
```

```
$> curl -X PUT -T r_pkg.tgz http://w:<writepw>@10.60.1.11:2580/languages/r_pkg.tgz
```

```
CREATE R SCALAR SCRIPT tophone(letters VARCHAR(2000000)) RETURNS int AS
.libPaths( c( .libPaths(), "/buckets/bucketfs1/languages/r_pkg/r_pkg"))
library(phonenumber)

run <- function(ctx) {
   letterToNumber(ctx$letters, qz = 1)
}
/

SELECT tophone('EXASOL');
```

Summary

Build your own Languages

Easy Integration via BucketFS

Use Your Language in EXASOL

EXASOL: Bring your own language!

http://www.exasol.com/testdrive

Download a copy of
EXASOL Community Edition
or sign up for a personal
demo system in the cloud.