Tackling Big Data with R

New features and old concepts for handling large and streaming data in practice

Simon Urbanek

R Foundation

Overview

- Motivation
- Custom connections
- Data processing pipelines
- Parallel processing
- Back-end experiments: Hadoop, RDFS
- Call for participation

Motivation

- R’s in memory model is fast
  - RAM prices declining steadily (unlike CPUs), [ca. $8/Gb for server RAM now]
  - Billion+ rows in R workable
- Problem 1: parallelization

```
s <- split(df, ...) ### slow and ineffcient!
y <- mclapply(s, function(x) ...)
```

  - splitting up data is expensive
- Problem 2: streaming
  - conceptually cannot have all data at once

Old, simple idea: chunking

- Process data in (big) chunks
- Parallelization:
  - feed each process/worker with chunks, collect results
  - can process chunks in parallel (if the processing can be independent); no copying
- Streaming:
  - keep a mutable state
  - process chunks as they come in, modifying state and creating results
- Issue: R has no explicit framework/API for this

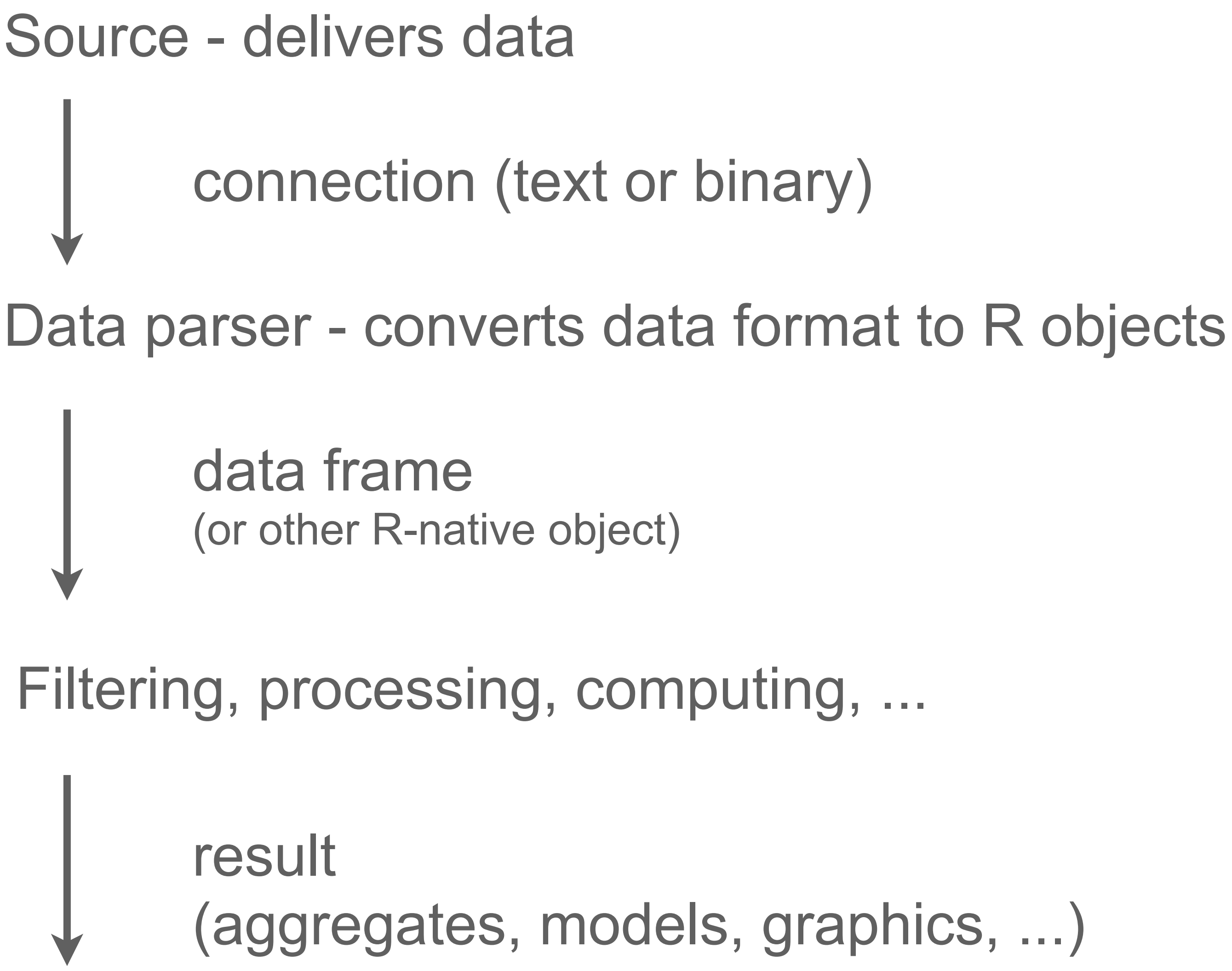
Connections

- R has connections: abstraction for data access and transport - completely back-end opaque!
- New in R 3.0.0: custom connection support
  - packages can create new connection implementations
  - some examples:
    - zmqc - 0MQ PUB/SUB connections - read from 0MQ streaming feeds directly
    - hdfsc - read files from HDFS - just like any other file

```
f = HDFS(“/data/foo”)
d = read.table(f)
```

Data pipeline

```
mean(read.table(HDFS("foo"))$x)
```



Source - delivers data

connection (text or binary)

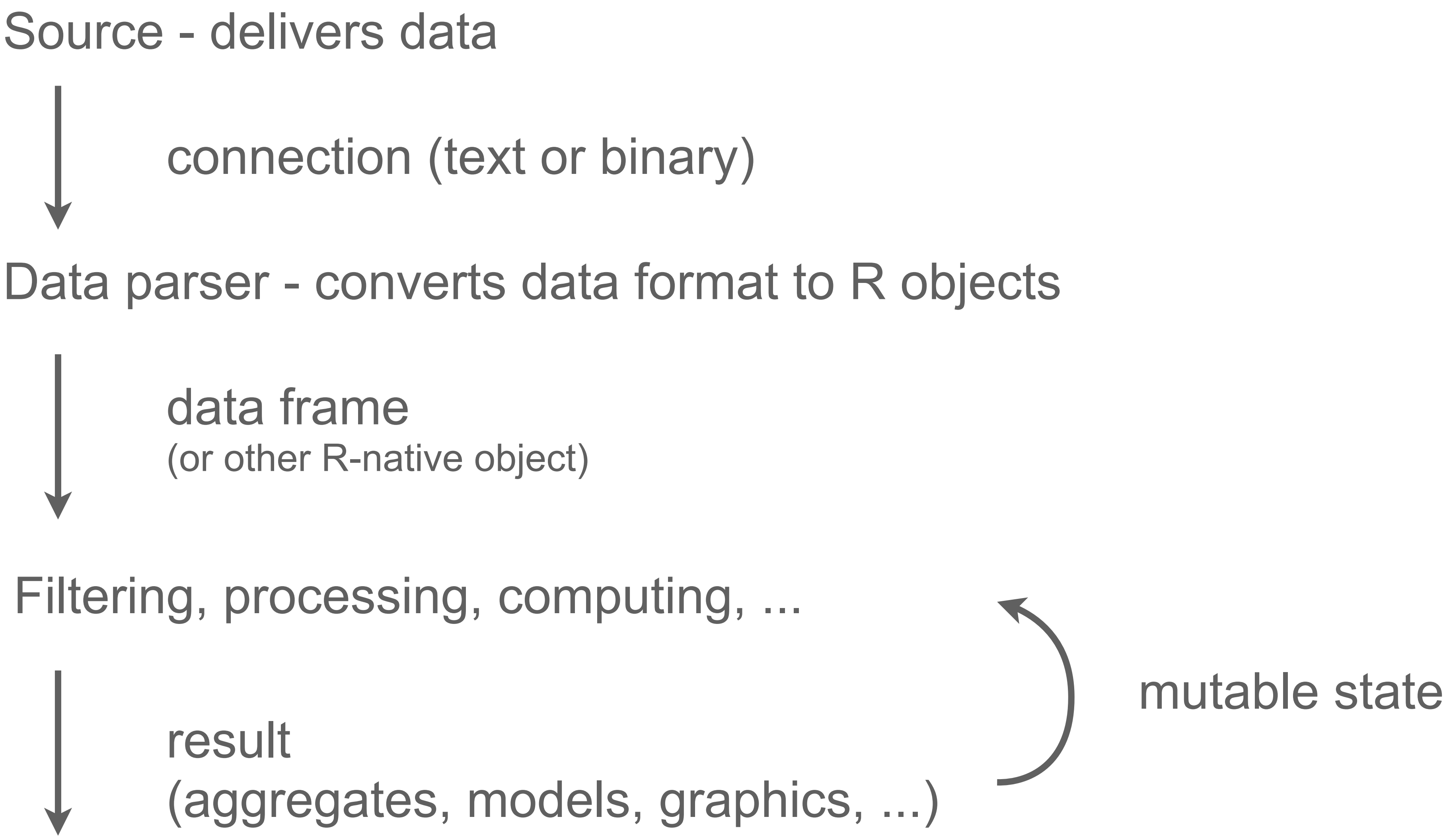
Data parser - converts data format to R objects

data frame
(or other R-native object)

Filtering, processing, computing, ...

result
(aggregates, models, graphics, ...)

Streaming



Source - delivers data

connection (text or binary)

Data parser - converts data format to R objects

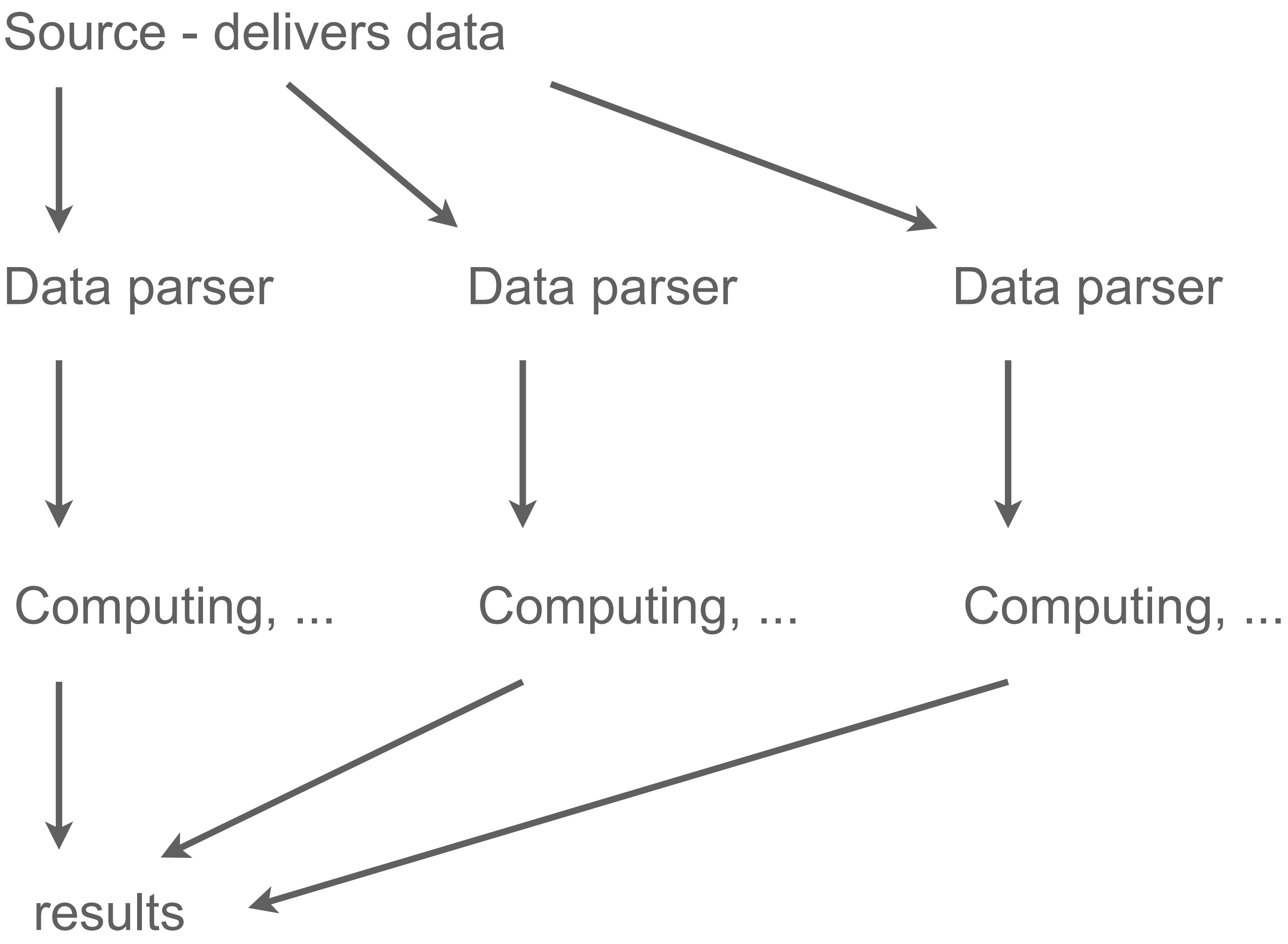
data frame
(or other R-native object)

Filtering, processing, computing, ...

result
(aggregates, models, graphics, ...)

mutable state

Parallel processing



Source - delivers data

Data parser

Data parser

Data parser

Computing, ...

Computing, ...

Computing, ...

results

Proposal: Chunks in a pipeline

- Connections
  - define available classes of data sources *contribute!*
- Read from sources in big chunks
- Parsers
  - transform data representation to R objects *contribute!*
- Compute
  - algorithms that work on chunks *contribute!*

    (serial processing + mutable state = streaming, independence = parallel)
- Collect
  - algorithms to combine parallel chunks *contribute!*

Example: streaming

- Use 0MQ PUB/SUB: buffered per subscriber (slow subscribers don’t affect others; can detect dropped recor
- Read
- Upda
- Serve Rhttp

```
feed  = zmqc(“ipc:///my–feed.0mq”, “r”)
max   = 1000
state = numeric()
while (TRUE) {
  d     = read.table(feed, FALSE, nrows=max)
  mix   = c(state ∗ 0.9, table(d[,2]))
  state = tapply(mix, names(mix), sum)
  if (any(state <= 1)) state = state[state > 1]
}
```

Parallel processing

- At least three stages:
  - split (often implicit)
  - compute
  - combine
- Define functions using this paradigm simple examples:

```
cc.sum   <- function(x) cc(x, sum, sum)

cc.table <- function(x) cc(x, table, function(x) tapply(x, names(x), sum))

cc.mean  <- function(x) cc(x, function(x) c(sum(x), length(x)),
                              function(x) sum(x[1,]) / sum(x[2,]))
```

Practical considerations

- The implementation can be seamless: use special “distributed vector” class and dispatch on it
- Typically source is big, so splitting is implicit since the data does not reside in R (e.g. sequence in a file)
- Leverage distributed storage: run computing where the chunks are stored

  Examples:

  - Hadoop
  - RDFS

Hadoop

- A lot of companies invest in Hadoop clusters
  (we have to live with it even if there are many better solutions)
- Literal map/reduce based on key/value is very inefficient for R since it is not a vector operation
- Hadoop can be (ab)used for chunk-wise processing: streaming mode - use HDFS chunks as input, compute is map on the entire chunk, combine is reduce

Example

- Aggregate point locations by ZIP code (match points against ZCTA US/Census 2010 shapefiles)

```
r <- read.table(hmr(
  hinput("/data/2013/06"),
  function(x)
    table(zcta2010.db()[
       inside(zcta2010.shp(), x[,4], x[,5]), 1]),
  function(x) ctapply(x, names(x), sum)))
```

- Fairly native R programming
- Implicit defaults (read.table parser, conversion of named vectors to key/value entries)
- Result is an HDFS connection

R Distributed File System - Experiment

- Purely R-based (R client, R server, R code)
- Uses Rserve for fast access
  (no setup cost, optional authentication, users switching, transport encryption for free)
- Any storage available (RData, ASCII, ...), all storage is R-native - parsing step can be removed
- No name node, all nodes are equal
- Scales only to moderate cluster sizes (hundreds of nodes), but is very fast (milliseconds for job setup, no need to leave R)

Call for Participation

- More users, more use cases
  - is this powerful enough?
  - if not, what is missing?
- Make it part of R
  - so developers can rely on it
- Start writing functions and packages
  - help to create critical mass
- Theoretical work
  - methods and approaches that give bounds for approximation error, necessary assumptions etc.

Related work

- Purdue Univ: Divide/Recombine
  - results for linear model approximations
  - RHipe - very specialized vehicle for the above using specific version and brand of Hadoop
- Iterators (also used by foreach)
  - idea of running code in iterations; does include chunks
  - focused on inner code (chunk processing)

Conclusions

- New in R 3.0.0: custom connections, to be used as building blocks for data pipelines
- Read from connections in chunks, compute and collect
- Generic framework that can be applied to streaming and parallel processing
- Let us work together to see if it is powerful enough to build an official R interface that everyone can use and contribute to
- Back-end agnostic - testing on Hadoop and RDFS

Contact

- Most packages available on **RForge.net** (source also on GitHub)
  - http://RForge.net/zmqc
  - http://RForge.net/hdfsc
- Remaining packages (iotools, rdfs, …) in the process of being pushed, check RForge.net and
  - https://github.com/s-u

Simon URBANEK
simon.urbanek@R-project.org
http://urbanek.info