Big Data Analytics

Lecture 5:

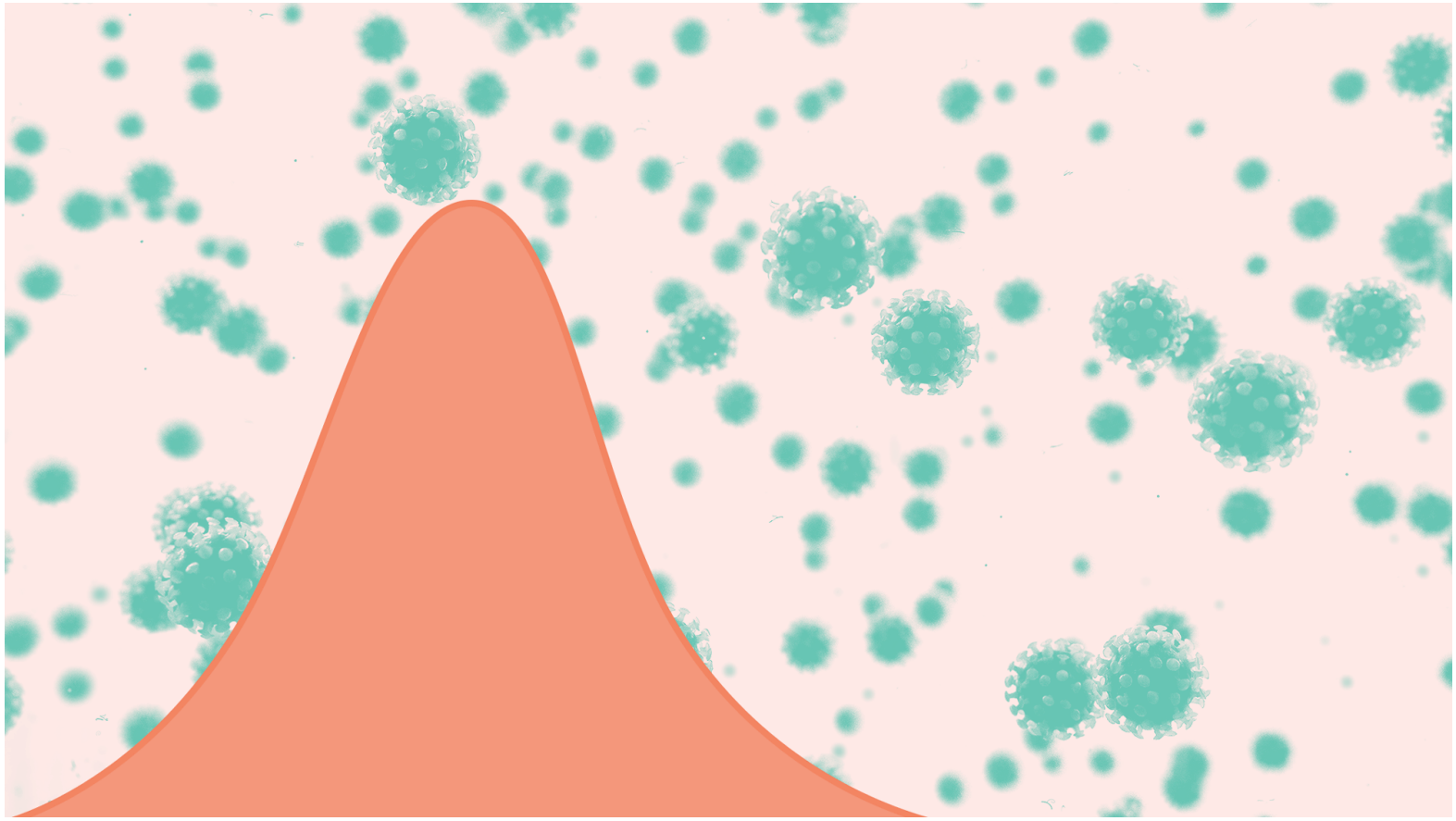
Cleaning and Transformation of Big Data

Prof. Dr. Ulrich Matter

25/03/2021

Updates

Welcome to Zoom



Online lecture mode

- All lectures **via Zoom**, same time/day as usual.
  - Lectures are recorded (30 days available).
  - Preferably no breaks, max. 90 minutes straight.
- Materials online, as usual.
- Last two sessions (7 May, 14 May): Q&A session in Zoom instead of presentations/discussion in classroom.

Online examination mode

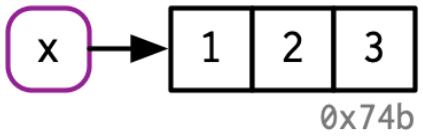
- Part I: take-home exercises: No changes. To be handed out on 7 May, **to be handed in on 8 June, 16:00**.
- Part II: project presentations: presentations recorded as 'screencast' (voice-over-slides).
  - Basically still the same requirements: use Rmd to create slides, presentations of 6-7 minutes max., etc. The only difference is **how** you deliver your presentation.
  - See here for tips on how to make a screencast.
  - Hand in your presentations by **14 May 2020, 23:59**.
  - See assignment in StudyNet/Canvas.

Recap Week 4

Bindings basics

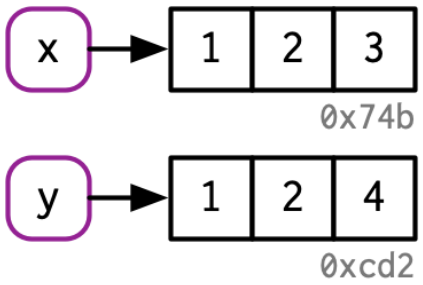
- Objects/values do not have names but **names have values**!
- Objects have a ‘memory address’/identifiers.

```
x <- c(1, 2, 3)
```



Copy-on-modify

- If we modify values in a vector, actual ‘copying’ is necessary (depending on the data structure of the object…).



Data structures and modify-in-place

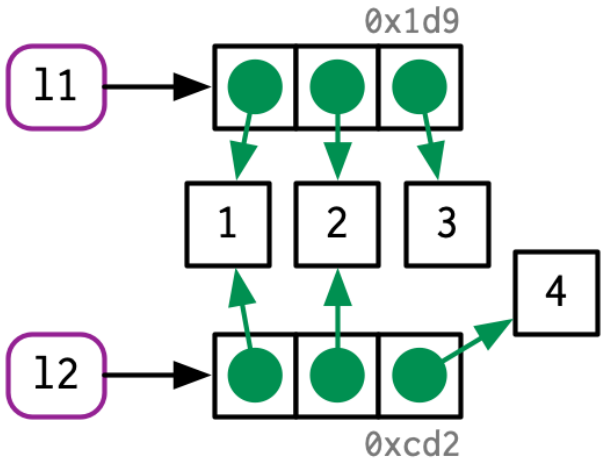


Figure by (???) (licensed under CC BY-NC-SA 4.0).

Improving performance

- Bottleneck(s) identified, what now?
- See previous examples for typical problems in a data analytics context.
- Vast variety of potential bottlenecks. Hard to give general advice.

Programming with Big Data

1. Which basic (already implemented) R functions are more or less suitable as building blocks for the program?
2. How can we exploit/avoid some of R's lower-level characteristics in order to implement efficient functions?
3. Is there a need to interface with a lower-level programming language in order to speed up the code? (advanced topic)

- Independent of **how** we write a statistical procedure in R (or in any other language, for that matter), is there an **alternative statistical procedure/algorithm** that is faster but delivers approximately the same result.

Issues to keep in mind

- Vectorization.
- Memory: avoid copying, pre-allocate memory.
- Use built in primitive (C) functions (caution: not always faster, if aim is precision).
- Existing solutions: load additional packages (`read.csv()` vs. `data.table::fread()`).
    - Focus of what follows in this course (approach taken in Walkowiak (2016)).

Procedural view and further reading

- Consider Hadley's advice: (???): Chapter 24
- Experienced coder? Have a look at R Inferno
- Further reading after this course: The Art of R Programming

Goals for today

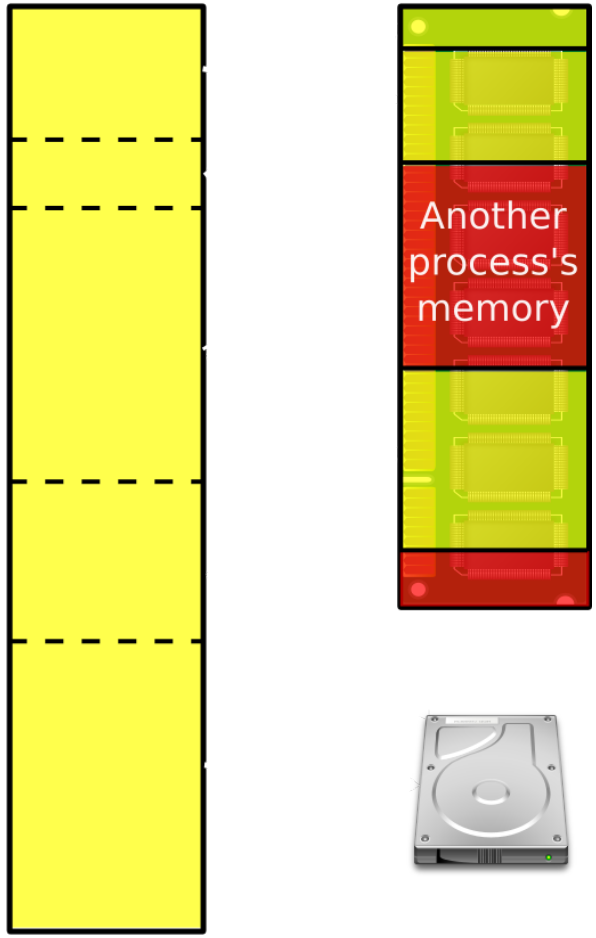
1. Know basic strategies for out-of-memory operations in R.
2. Know basic tools for local big data cleaning and transformation in R.
3. Understand (in simple terms) how these tools work.
4. (Recap of virtual memory concept)

Virtual Memory

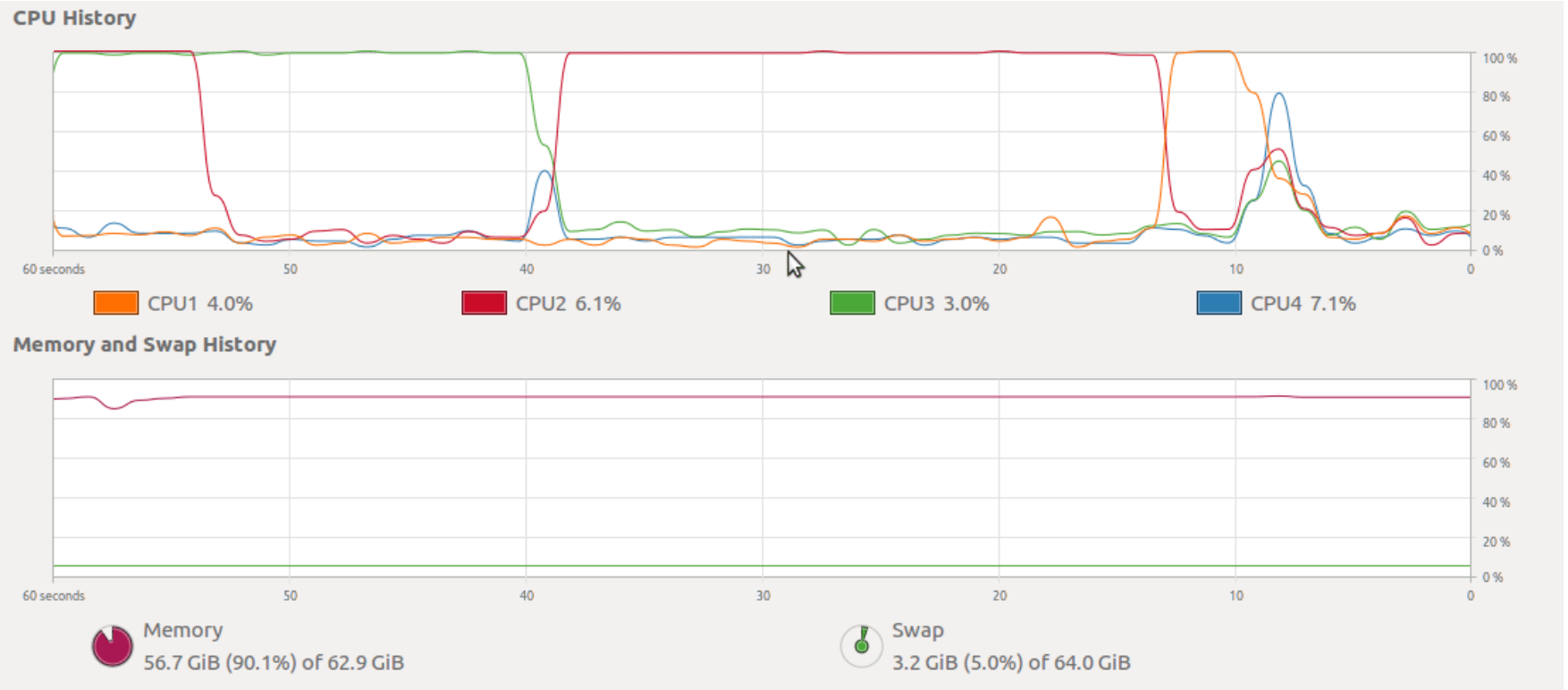
Virtual memory

- Operating system allocates part of mass storage device (hard-disk) as **virtual memory**.
- Process/application uses up too much RAM, OS starts **swapping** data between RAM and virtual memory.
- Processes slow down due to swapping.
- Default (OS) usage of virtual memory concept is not necessarily optimized for data analysis tasks.

Virtual memory



Virtual memory: example (linux)



'Out-of-memory' strategies

- Use virtual memory idea for specific data analytics tasks.
- Two approaches:
  - **Chunked data files on disk**: partition large data set, map and store chunks of raw data on disk. Keep mapping in RAM. (`ff`-package)
  - **Memory mapped files and shared memory**: virtual memory is explicitly allocated for one or several specific data analytics tasks (different processes can access the same memory segment). (`bigmemory`-package)

Chunking data with the ff-package

Preparations

```
# SET UP --------------

# install.packages(c("ff", "ffbase"))
# load packages
library(ff)
library(ffbase)
library(pryr)

# create directory for ff chunks, and assign directory to ff
system("mkdir ffdf")
options(fftempdir = "ffdf")
```

Chunking data with the ff-package

Import data, inspect change in RAM.

```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   1393123  74.5    2149367  114.8   2149367  114.8
## Vcells 122509759 934.7  213343868 1627.7 211038278 1610.1
```

```
mem_change(
flights <-
     read.table.ffdf(file="../data/flights.csv",
                     sep=",",
                     VERBOSE=TRUE,
                     header=TRUE,
                     next.rows=100000,
                     colClasses=NA)
)
```

```
## read.table.ffdf 1..100000 (100000)  csv-read=0.402sec ffdf-write=0.039sec
## read.table.ffdf 100001..200000 (100000)  csv-read=0.411sec ffdf-write=0.03sec
## read.table.ffdf 200001..300000 (100000)  csv-read=0.424sec ffdf-write=0.031sec
## read.table.ffdf 300001..336776 (36776)  csv-read=0.158sec ffdf-write=0.017sec
##  csv-read=1.395sec  ffdf-write=0.117sec  TOTAL=1.512sec
```

```
## -31.6 MB
```

Chunking data with the ff-package

Inspect file chunks on disk and data structure in R environment.

```
# show the files in the directory keeping the chunks
list.files("ffdf")
```

```
##   [1] "clone1664b7fbd953f.ff" "clone1664b9b8cca9.ff"  "clone1e7014c0a1cd8.ff"
##   [4] "clone1e7015a4f712e.ff" "clone2aea22211d9e1.ff" "clone2aea2360c6703.ff"
##   [7] "clone2aea2566ab42d.ff" "clone2aea25e1c1f75.ff" "clone2d49618dbfbf6.ff"
##  [10] "clone2d4965ee3349a.ff" "clone2d49664b07745.ff" "clone2d49672b82b88.ff"
##  [13] "clone308112a4ca401.ff" "clone308113d044b7c.ff" "clone308113d22fb5f.ff"
##  [16] "clone3081149714ed4.ff" "clone399cd5627eb1f.ff" "clone399cd72c6506d.ff"
##  [19] "clone399cd78f6c4e6.ff" "clone399cd8b1f075.ff"  "clone3c3ef1e38eca1.ff"
##  [22] "clone3c3ef4ac46441.ff" "clone3c3ef514956e9.ff" "clone3c3efcb5fb24.ff"
##  [25] "ff1664b222c38f0.ff"    "ff1664b4d23ee78.ff"    "ff1664b4d7f1e3e.ff"
##  [28] "ff1e7011754e092.ff"    "ff1e7011a76d5a6.ff"    "ff1e7017084631e.ff"
##  [31] "ff2aea22c3703b9.ff"    "ff2aea2664ee33.ff"     "ff2aea26b164ce7.ff"
##  [34] "ff2d49627cd458.ff"     "ff2d49631ca5a34.ff"    "ff2d4964237cc21.ff"
##  [37] "ff30811207897c4.ff"    "ff308115699c1a.ff"     "ff30811b3430e.ff"
##  [40] "ff399cd1a2ffc0e.ff"    "ff399cd1e963877.ff"    "ff399cd5477c29.ff"
##  [43] "ff3c3ef17300293.ff"    "ff3c3ef2229a09b.ff"    "ff3c3ef765d6fb7.ff"
##  [46] "ffdf1664b11f63957.ff"  "ffdf1664b12d379a7.ff"  "ffdf1664b16a5a516.ff"
##  [49] "ffdf1664b16cc8da8.ff"  "ffdf1664b16d72904.ff"  "ffdf1664b178a08cd.ff"
##  [52] "ffdf1664b17c18654.ff"  "ffdf1664b23c24fe0.ff"  "ffdf1664b24c84c7c.ff"
##  [55] "ffdf1664b25a0763c.ff"  "ffdf1664b2663b0d0.ff"  "ffdf1664b283091fe.ff"
##  [58] "ffdf1664b2b44b5a4.ff"  "ffdf1664b2c262d7d.ff"  "ffdf1664b2cd1a62.ff"
##  [61] "ffdf1664b2e3a4c4e.ff"  "ffdf1664b2ed055dd.ff"  "ffdf1664b3054a5d7.ff"
```

Memory mapping with `bigmemory`

Preparations

```
# SET UP ----------------

# load packages
library(bigmemory)
library(biganalytics)
```

Memory mapping with `bigmemory`

Import data, inspect change in RAM.

```
# import the data
flights <- read.big.matrix("../data/flights.csv",
                    type="integer",
                    header=TRUE,
                    backingfile="flights.bin",
                    descriptorfile="flights.desc")
```

Memory mapping with `bigmemory`

Inspect the imported data.

```
summary(flights)
```

```
##                      min          max         mean          NAs
## year          2013.000000  2013.000000  2013.000000     0.000000
## month            1.000000    12.000000     6.548510     0.000000
## day              1.000000    31.000000    15.710787     0.000000
## dep_time         1.000000  2400.000000  1349.109947  8255.000000
## sched_dep_time 106.000000  2359.000000  1344.254840     0.000000
## dep_delay      -43.000000  1301.000000    12.639070  8255.000000
## arr_time         1.000000  2400.000000  1502.054999  8713.000000
## sched_arr_time   1.000000  2359.000000  1536.380220     0.000000
## arr_delay      -86.000000  1272.000000     6.895377  9430.000000
## carrier          9.000000     9.000000     9.000000 318316.000000
## flight           1.000000  8500.000000  1971.923620     0.000000
## tailnum                                               336776.000000
## origin                                                336776.000000
## dest                                                  336776.000000
## air_time        20.000000   695.000000   150.686460  9430.000000
## distance        17.000000  4983.000000  1039.912604     0.000000
## hour             1.000000    23.000000    13.180247     0.000000
## minute           0.000000    59.000000    26.230100     0.000000
## time_hour     2013.000000  2014.000000  2013.000261     0.000000
```

Memory mapping with `bigmemory`

Inspect the object loaded into the R environment.

```
flights
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x5608bde76480>
```

Memory mapping with `bigmemory`

- `backingfile`: The cache for the imported file (holds the raw data on disk).
- `descriptorfile`: Metadata describing the imported data set (also on disk).

Memory mapping with `bigmemory`

Understanding the role of `backingfile` and `descriptorfile`.

First, import a large data set without a backing-file:

```
# import data and check time needed
system.time(
     flights1 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer")
)
```

```
##    user  system elapsed
##   1.050   0.019   1.069
```

```
# import data and check memory used
mem_change(
     flights1 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer")
)
```

```
## 528 B
```

Memory mapping with `bigmemory`

Understanding the role of `backingfile` and `descriptorfile`.

Second, import the same data set with a backing-file:

```
# import data and check time needed
system.time(
     flights2 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer",
                                 backingfile = "flights2.bin",
                                 descriptorfile = "flights2.desc"
                                 )
)
```

```
##    user  system elapsed
##   1.067   0.012   1.081
```

```
# import data and check memory used
mem_change(
     flights2 <- read.big.matrix("../data/flights.csv",
                                 header = TRUE,
                                 sep = ",",
                                 type = "integer",
                                 backingfile = "flights2.bin",
```

Memory mapping with `bigmemory`

Understanding the role of `backingfile` and `descriptorfile`.

Third, re-import the same data set with a backing-file.

```
# remove the loaded file
rm(flights2)

# 'load' it via the backing-file
system.time(flights2 <- attach.big.matrix("flights2.desc"))
```

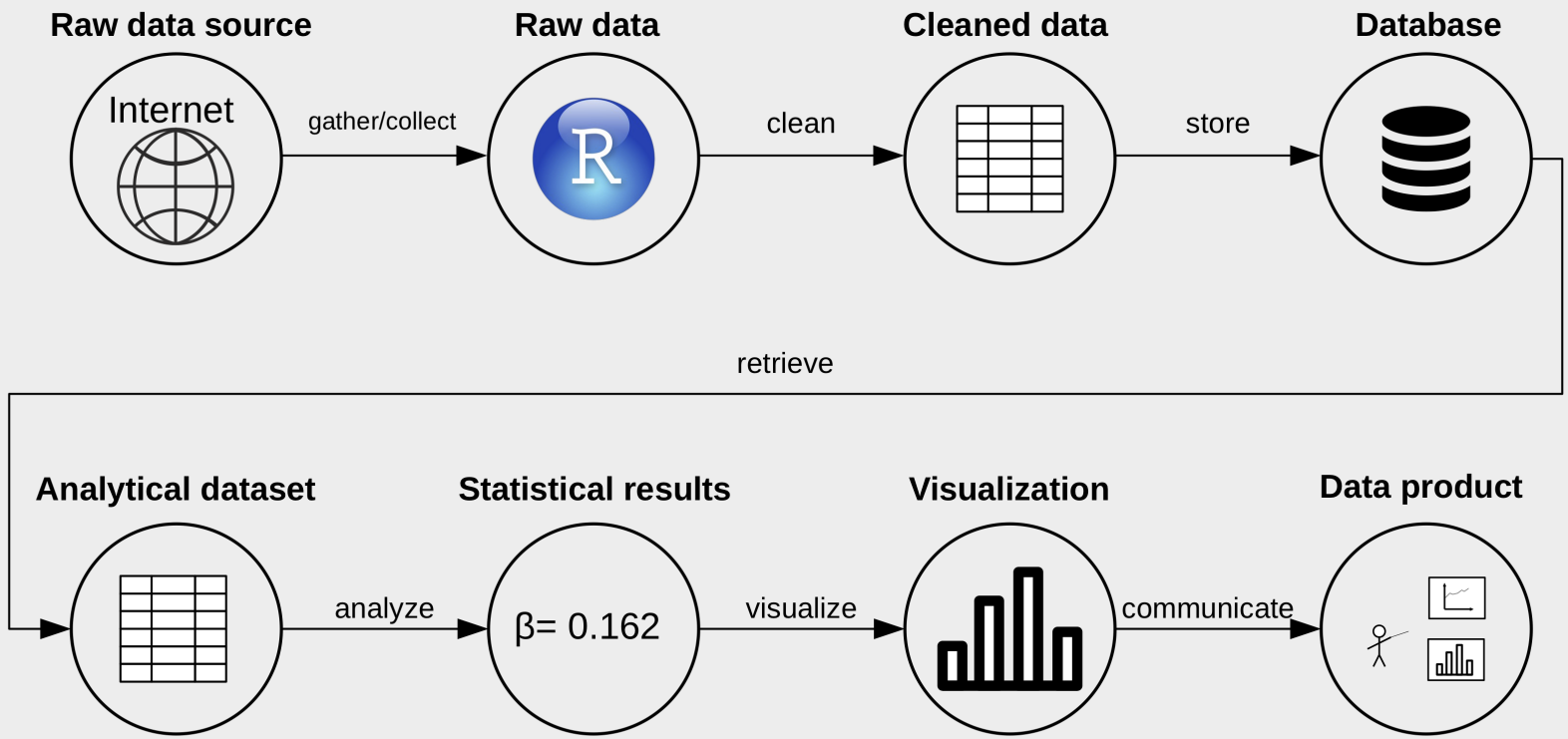
```
##    user  system elapsed
##   0.000   0.000   0.001
```

```
flights2
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x5608c166c350>
```

Cleaning and Transformation

Data (science) pipeline



Typical tasks (independent of data set size)

- Normalize/standardize.
- Code additional variables (indicators, strings to categorical, etc.).
- Remove, add covariates.
- Merge data sets.
- Set data types.

Typical workflow

1. Import raw data.
2. Clean/transform.
3. Store for analysis.
    - Write to file.
    - Write to database.

Bottlenecks

- RAM:
  - Raw data does not fit into memory.
  - Transformations enlarge RAM allocation (copying).
- Mass Storage: Reading/Writing
- CPU: Parsing (data types)

Data Preparation with ff

Set up

The following examples are based on Walkowiak (2016), Chapter 3.

```
## SET UP ------------------------

#Set working directory to the data and airline_id files.
# setwd("materials/code_book/B05396_Ch03_Code")
system("mkdir ffdf")
options(fftempdir = "ffdf")

# load packages
library(ff)
library(ffbase)
library(pryr)

# fix vars
FLIGHTS_DATA <- "../code_book/B05396_Ch03_Code/flights_sep_oct15.txt"
AIRLINES_DATA <- "../code_book/B05396_Ch03_Code/airline_id.csv"
```

Data import

```
# DATA IMPORT ------------------

# 1. Upload flights_sep_oct15.txt and airline_id.csv files from flat files.

system.time(flights.ff <- read.table.ffdf(file=FLIGHTS_DATA,
                                          sep=",",
                                          VERBOSE=TRUE,
                                          header=TRUE,
                                          next.rows=100000,
                                          colClasses=NA))
```

```
## read.table.ffdf 1..100000 (100000)  csv-read=0.521sec ffdf-write=0.075sec
## read.table.ffdf 100001..200000 (100000)  csv-read=0.517sec ffdf-write=0.05sec
## read.table.ffdf 200001..300000 (100000)  csv-read=0.521sec ffdf-write=0.05sec
## read.table.ffdf 300001..400000 (100000)  csv-read=0.517sec ffdf-write=0.054sec
## read.table.ffdf 400001..500000 (100000)  csv-read=0.513sec ffdf-write=0.049sec
## read.table.ffdf 500001..600000 (100000)  csv-read=0.512sec ffdf-write=0.05sec
## read.table.ffdf 600001..700000 (100000)  csv-read=0.524sec ffdf-write=0.043sec
## read.table.ffdf 700001..800000 (100000)  csv-read=0.512sec ffdf-write=0.047sec
## read.table.ffdf 800001..900000 (100000)  csv-read=0.506sec ffdf-write=0.057sec
## read.table.ffdf 900001..951111 (51111)  csv-read=0.261sec ffdf-write=0.038sec
##  csv-read=4.904sec  ffdf-write=0.513sec  TOTAL=5.417sec
```

```
##    user  system elapsed
##   5.246   0.172   5.420
```

Comparison with read.table

```
##Using read.table()
system.time(flights.table <- read.table(FLIGHTS_DATA,
                                        sep=",",
                                        header=TRUE))
```

```
##    user  system elapsed
##   5.077   0.116   5.197
```

```
gc()
```

```
##            used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   1396908   74.7    2149367  114.8   2149367  114.8
## Vcells 136559299 1041.9  213343868 1627.7 212429013 1620.8
```

```
system.time(airlines.table <- read.csv(AIRLINES_DATA,
                                       header = TRUE))
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```
# check memory used
mem_used()
```

Inspect imported files

```
# 2. Inspect the ffdf objects.
## For flights.ff object:
class(flights.ff)
```

```
## [1] "ffdf"
```

```
dim(flights.ff)
```

```
## [1] 951111     28
```

```
## For airlines.ff object:
class(airlines.ff)
```

```
## [1] "ffdf"
```

```
dim(airlines.ff)
```

```
## [1] 1607    2
```

Data cleaning and transformation

Goal: merge airline data to flights data

```
# step 1:
## Rename "Code" variable from airlines.ff to "AIRLINE_ID" and "Description" into "AIRLINE_NM".
names(airlines.ff) <- c("AIRLINE_ID", "AIRLINE_NM")
names(airlines.ff)
```

```
## [1] "AIRLINE_ID" "AIRLINE_NM"
```

```
str(airlines.ff[1:20,])
```

```
## 'data.frame':    20 obs. of  2 variables:
##  $ AIRLINE_ID: int  19031 19032 19033 19034 19035 19036 19037 19038 19039 19040 ...
##  $ AIRLINE_NM: Factor w/ 1607 levels "40-Mile Air: Q5",..: 945 1025 503 721 64 725 1194 99 1395 2
```

Data cleaning and transformation

Goal: merge airline data to flights data

```
# merge of ffdf objects
mem_change(flights.data.ff <- merge.ffdf(flights.ff, airlines.ff, by="AIRLINE_ID"))
```

```
## 780 kB
```

```
class(flights.data.ff)
```

```
## [1] "ffdf"
```

```
dim(flights.data.ff)
```

```
## [1] 951111     29
```

```
dimnames(flights.data.ff)
```

```
## [[1]]
## NULL
##
## [[2]]
##  [1] "YEAR"              "MONTH"             "DAY_OF_MONTH"      "DAY_OF_WEEK"
```

Inspect difference to in-memory operation

```
##For flights.table:
names(airlines.table) <- c("AIRLINE_ID", "AIRLINE_NM")
names(airlines.table)
```

```
## [1] "AIRLINE_ID" "AIRLINE_NM"
```

```
str(airlines.table[1:20,])
```

```
## 'data.frame':    20 obs. of  2 variables:
##  $ AIRLINE_ID: int  19031 19032 19033 19034 19035 19036 19037 19038 19039 19040 ...
##  $ AIRLINE_NM: chr  "Mackey International Inc.: MAC" "Munz Northern Airlines Inc.: XY" "Cochise A
```

```
# check memory usage of merge in RAM
mem_change(flights.data.table <- merge(flights.table,
                                       airlines.table,
                                       by="AIRLINE_ID"))
```

```
## 160 MB
```

Subsetting

```
mem_used()
```

```
## 1,331,350,816 B
```

```
# Subset the ffdf object flights.data.ff:
subs1.ff <- subset.ffdf(flights.data.ff, CANCELLED == 1,
                        select = c(FL_DATE, AIRLINE_ID,
                                   ORIGIN_CITY_NAME,
                                   ORIGIN_STATE_NM,
                                   DEST_CITY_NAME,
                                   DEST_STATE_NM,
                                   CANCELLATION_CODE))
```

```
dim(subs1.ff)
```

```
## [1] 4529    7
```

```
mem_used()
```

```
## 1,331,633,840 B
```

Save to ffdf-files

(For further processing with `ff`)

```
# Save a newly created ffdf object to a data file:

save.ffdf(subs1.ff, overwrite = TRUE) #7 files (one for each column) created in the ffdb directory
```

Load ffdf-files

```
# Loading previously saved ffdf files:
rm(subs1.ff)
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   1417399   75.7    4479337  239.3   3298975  176.2
## Vcells 156549594 1194.4  256092641 1953.9 212429013 1620.8
```

```
load.ffdf("ffdb")
str(subs1.ff)
```

```
## List of 3
##  $ virtual: 'data.frame':    7 obs. of  7 variables:
##  .. $ VirtualVmode     : chr  "integer" "integer" "integer" "integer" ...
##  .. $ AsIs             : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  .. $ VirtualIsMatrix  : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  .. $ PhysicalIsMatrix : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
##  .. $ PhysicalElementNo: int  1 2 3 4 5 6 7
##  .. $ PhysicalFirstCol : int  1 1 1 1 1 1 1
##  .. $ PhysicalLastCol  : int  1 1 1 1 1 1 1
##  .. - attr(*, "Dim")= int [1:2] 4529 7
##  .. - attr(*, "Dimorder")= int [1:2] 1 2
##  $ physical: List of 7
##  .. $ FL_DATE          : list()
##  ..  ..- attr(*, "physical")=Class 'ff_pointer' <externalptr>
```

Export to CSV

```
#  Export subs1.ff into CSV and TXT files:
write.csv.ffdf(subs1.ff, "subset1.csv")
```

References

Walkowiak, Simkon. 2016. **Big Data Analytics with R**. Birmingham, UK: PACKT Publishing.