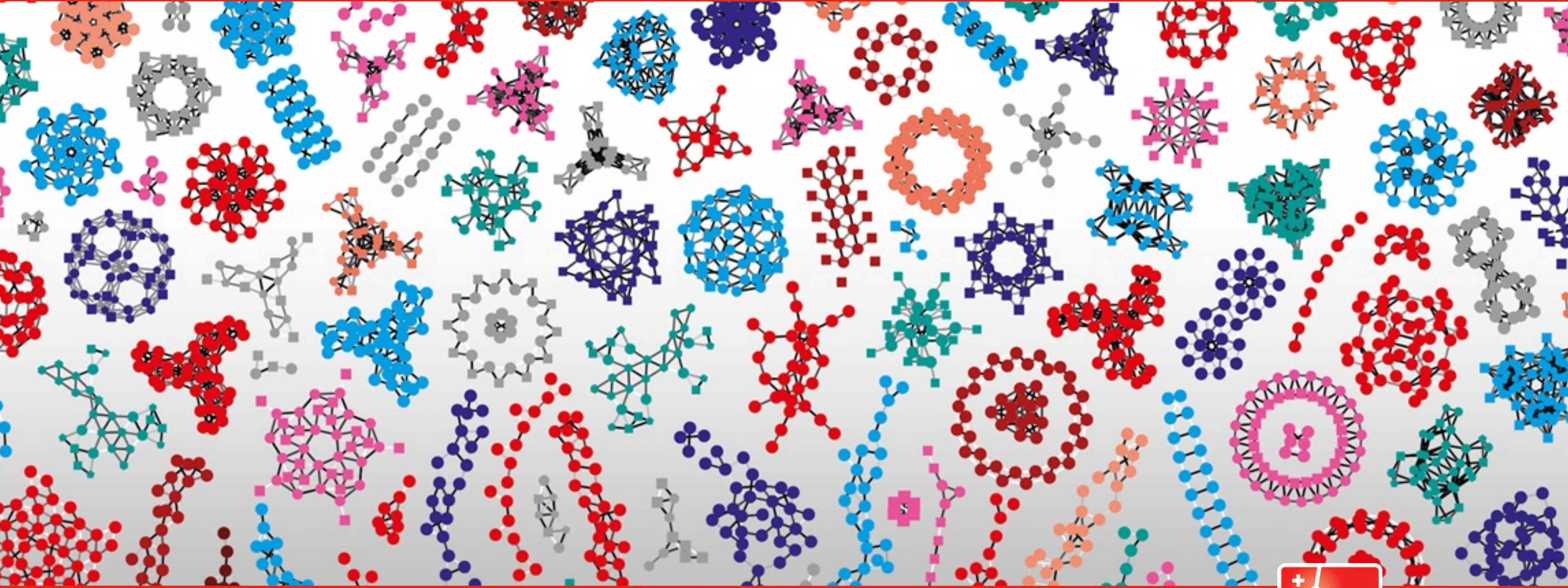


Statistical methods for big data in life sciences and health with R

Linda Dib, Frédéric Schütz

4th of June 2018

Swiss Institute of Bioinformatics

BCF

www.sib.swiss

Credits

- Who?
- This course worth 1 credits

Course web-page

- Course page:
- https://edu.sib.swiss/course/view.php?id=344

- Login: smbd18
- Password: SIB-smbd18

What is Big? (for this course)

When R doesn’t work

What is Big? (for this course)

What gets more difficult when data is big?

- Visualization
  - Visualizations get messy
- Memory issues
  - The data may not load into memory
- Computational time
  - Analyzing the data may take a long time
- Etc.

How much data can R load?

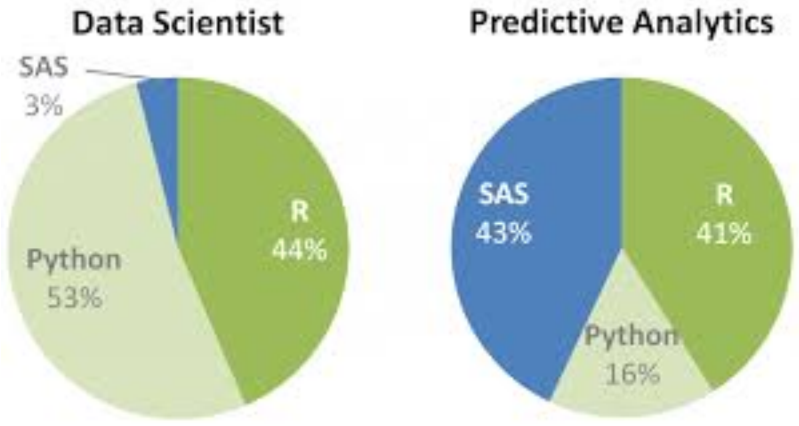
R sets a limit on the most memory it will allocate from the operating system

```
>memory.limit()
>?memory.limit
```

Comparing R to SAS

Under the hood:

- R loads all data into memory (by default)
- SAS allocates memory dynamically to keep data on disk (by default)



Changing the limit

memory.size() allows you to change R’s allocation limit.

Changing the limit

memory.size() allows you to change R’s allocation limit.

But…

Memory limits are dependent on your configuration

- If you're running 32-bit R on any OS, it'll be 2 or 3Gb
- If you're running 64-bit R on a 64-bit OS, the upper limit is effectively infinite,

Changing the limit

memory.size() allows you to change R’s allocation limit.

But…

Memory limits are dependent on your configuration

- If you're running 32-bit R on any OS, it'll be 2 or 3Gb
- If you're running 64-bit R on a 64-bit OS, the upper limit is effectively infinite,

  but…

  …you shouldn’t load huge datasets into memory and use

    Virtual memory, swapping, etc.

-

maximum 2,147,483,647
rows or columns

2GB of memory $\neq$ 2GB on disk

Making memory size meaningful

First example

Investigate object size

Smoking, Alcohol and Œsophageal Cancer

Breslow, N. E. and Day, N. E. (1980) Statistical Methods in Cancer Research. Volume 1: The Analysis of Case-Control Studies. IARC Lyon / Oxford University Press.

Smoking, Alcohol and Œsophageal Cancer
Data from a case-control study of œsophageal cancer in Ille-et-Vilaine, France.

Breslow, N. E. and Day, N. E. (1980) Statistical Methods in Cancer Research. Volume 1: The Analysis of Case-Control Studies. IARC Lyon / Oxford University Press.

Smoking, Alcohol and Œsophageal Cancer
Data from a case-control study of œsophageal cancer in Ille-et-Vilaine, France.

Size of data?

```
>data(esoph)
>object.size(esoph)
```

Breslow, N. E. and Day, N. E. (1980) Statistical Methods in Cancer Research. Volume 1: The Analysis of Case-Control Studies. IARC Lyon / Oxford University Press.

Second example

Investigate odds computation

BRFSS -Behavioral Risk Factor Surveillance System

BRFSS -Behavioral Risk Factor Surveillance System
Health-related telephone surveys collected in U.S

Download BRFSS as XPT file and unzip to a local file
URL: http://www.cdc.gov/brfss/annual_data/2013/files/LLCP2013XPT.ZIP
Universal Xpt File Viewer was previously known as the SAS Viewer.
In R two packages:

- Hmisc
- SASxport

```
>library(SASxport)
>brfss<- read.xport(”LLCP2013.xpt")
>head(brfss)
```

BRFSS™
Behavioral Risk Factor Surveillance System

Cholesterol Awareness

Section 6: Cholesterol Awareness

_RFCHOL	*Calculated variable for adults who have had their cholesterol checked and have been told by a doctor, nurse, or other health professional that it was high.* We derive _RFCHOL from BLOODCHO and TOLDHI2.	
1	No	Respondents who reported having had their blood cholesterol checked but had not been told it was high (BLOODCHO=1 and TOLDHI2=2)
2	Yes	Respondents who reported having had their blood cholesterol checked and had been told that they have high blood cholesterol (BLOODCHO=1 and TOLDHI2=1)
9	Don't Know/ Not Sure Or Refused /Missing	Respondents who reported they did not know if they had their blood cholesterol checked, those that reported they didn´t know if they have been told their blood cholesterol was high, those who refused to answer if they had their blood cholesterol checked, those who refused to answer if they had been told that their blood cholesterol was high, and those with missing responses (BLOODCHO=1 and TOLDHI2=7,9,or missing)
.	Missing	Respondents who reported they have not had their blood cholesterol checked (BLOODCHO=2,7,9,or missing)

Health Care Access

Section 3: Health Care Access

_HCVU651	*Calculated variable for respondents aged 18-64 who have any form of health care coverage.* We derive _HCVU651 from AGE and HLTHPLN1.	
1	Have Health Care Coverage	Respondents who reported having health care coverage (18 <= AGE <= 64 and HLTHPLN1 = 1)
2	Do Not Have Health Care Coverage	Respondents who reported not having health care coverage (18 <= AGE <= 64 and HLTHPLN1 = 2)
9	Don't Know/ Not Sure, Refused Or Missing	Respondents who reported that they did not know, were not sure, refused to report or had missing responses for having health care coverage (18 <= AGE <= 64 and HLTHPLN1 = 7, 9, or missing or AGE => 65)

SAS Code:

```
IF 18 LE AGE LE 64 THEN DO;
 IF HLTHPLN1=1 THEN _HCVU651=1;
 ELSE IF HLTHPLN1=2 THEN _HCVU651=2;
 ELSE _HCVU651=9;
 END;
 ELSE _HCVU651 = 9;
```

Cholesterol awareness &. health plan

Health plan?	Cholesterol aware	Cholesterol un-aware
YES	39	22
NO	61	78
Total	**100**	**100**

Cholesterol awareness &. health plan

Health plan?	Cholesterol aware	Cholesterol un-aware
YES	39	22
NO	61	78
Total	**100**	**100**

$$odds_{TMS} = \frac{39/100}{61/100} = \frac{39}{61} = 0.639$$

Cholesterol awareness &. health plan

Health plan?	Cholesterol aware	Cholesterol un-aware
YES	39	22
NO	61	78
Total	**100**	**100**

$$odds_{\text{aware}} = \frac{39/100}{61/100} = \frac{39}{61} = 0.639$$

$$odds_{\text{Not aware}} = \frac{22}{78} = 0.282$$

$$Odds\ ratio = \frac{0.639}{0.282} = 2.27$$

Odds are 2.27 times higher being aware than non aware when having a health care plan

BRFSS -Behavioral Risk Factor Surveillance System
Health-related telephone surveys collected in U.S

Download BRFSS as XPT file and unzip to a local file
URL: http://www.cdc.gov/brfss/annual_data/2013/files/LLCP2013XPT.ZIP

```
>library(epitools)
>oddsratio(as.factor(brfss$X_HCVU651),as.factor(brfss$X_RFCHOL))
```

BRFSS -Behavioral Risk Factor Surveillance System
Health-related telephone surveys collected in U.S

Download BRFSS as XPT file and unzip to a local file
URL: http://www.cdc.gov/brfss/annual_data/2013/files/LLCP2013XPT.ZIP

```
>library(epitools)
>oddsratio(as.factor(brfss$X_HCVU651),as.factor(brfss$X_RF
CHOL))
```

Error in fisher.test(xx) : FEXACT error 40.
Out of workspace.

Changing the amount of memory will NOT solve this

Solutions to bypass the limitation

- Get a bigger computer
- Format the data differently
- Make the data smaller

Solutions to bypass the limitation

- Get a bigger computer
- Format the data differently
- Make the data smaller

You may be lucky enough to have budget for a bigger PC

More likely, get some temporary space:

- Use one machine on the high-performance cluster
- Rent some cloud computing time

Solutions to bypass the limitation

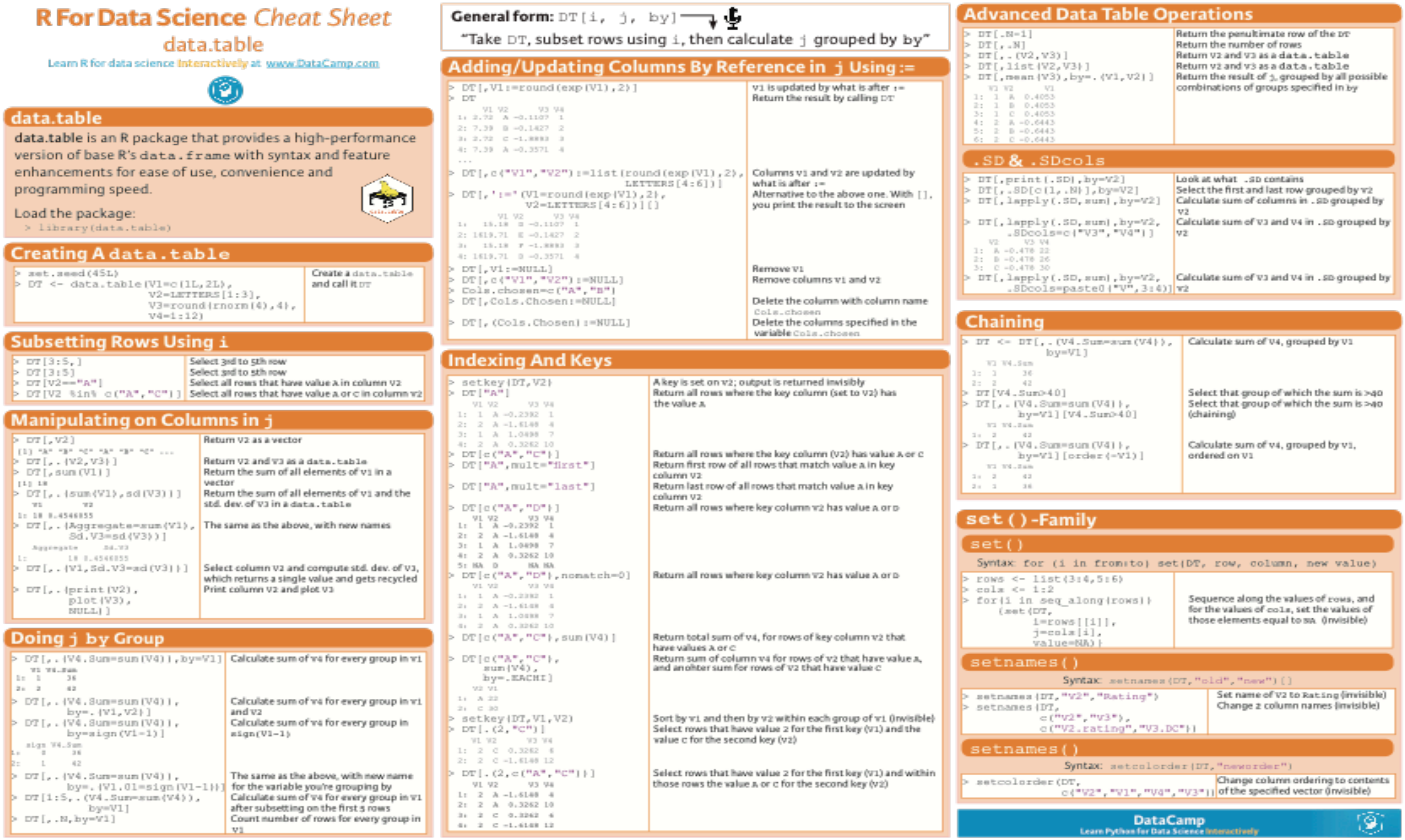
- Get a bigger computer
- Format the data differently
- Make the data smaller

Use data table rather than data frame

data.table package = optimizations to data frame, but slightly different syntax

```
>brfss_dt <- data.table(brfss)
>object.size(brfss_dt)
>object.size(brfss)
```

data.table cheat sheet



R For Data Science *Cheat Sheet*

data.table

Learn R for data science Interactively at www.DataCamp.com

data.table

data.table is an R package that provides a high-performance version of base R's `data.frame` with syntax and feature enhancements for ease of use, convenience and programming speed.

Load the package:

```
> library(data.table)
```

Creating A data.table

```
> set.seed(45L)
> DT <- data.table(V1=c(1L,2L),
                   V2=LETTERS[1:3],
                   V3=round(rnorm(4),4),
                   V4=1:12)
```

Create a data.table and call it DT

Subsetting Rows Using i

`> DT[3:5,]`	Select 3rd to 5th row
`> DT[3:5]`	Select 3rd to 5th row
`> DT[V2=="A"]`	Select all rows that have value A in column V2
`> DT[V2 %in% c("A","C")]`	Select all rows that have value A or C in column V2

Manipulating on Columns in j

`> DT[,V2]`	Return V2 as a vector
`> DT[,.(V2,V3)]`	Return V2 and V3 as a data.table
`> DT[,sum(V1)]`	Return the sum of all elements of V1 in a vector
`> DT[,.(sum(V1),sd(V3))]`	Return the sum of all elements of V1 and the std. dev. of V3 in a data.table
`> DT[,.(Aggregate=sum(V1), Sd.V3=sd(V3))]`	The same as the above, with new names
`> DT[,.(V1,Sd.V3=sd(V3))]`	Select column V2 and compute std. dev. of V3, which returns a single value and gets recycled
`> DT[,.(print(V2), plot(V3), NULL)]`	Print column V2 and plot V3

Doing j by Group

`> DT[,.(V4.Sum=sum(V4)),by=V1]`	Calculate sum of V4 for every group in V1
`> DT[,.(V4.Sum=sum(V4)), by=.(V1,V2)]`	Calculate sum of V4 for every group in V1 and V2
`> DT[,.(V4.Sum=sum(V4)), by=sign(V1-1)]`	Calculate sum of V4 for every group in sign(V1-1)
`> DT[,.(V4.Sum=sum(V4)), by=.(V1.01=sign(V1-1))]`	The same as the above, with new name for the variable you're grouping by
`> DT[1:5,.(V4.Sum=sum(V4)), by=V1]`	Calculate sum of V4 for every group in V1 after subsetting on the first 5 rows
`> DT[,.N,by=V1]`	Count number of rows for every group in V1

General form: `DT[i, j, by]`

"Take DT, subset rows using i, then calculate j grouped by by"

Adding/Updating Columns By Reference in j Using :=

`> DT[,V1:=round(exp(V1),2)]` `> DT`	V1 is updated by what is after := Return the result by calling DT
`> DT[,c("V1","V2"):=list(round(exp(V1),2), LETTERS[4:6])]`	Columns V1 and V2 are updated by what is after :=
`> DT[,':='(V1=round(exp(V1),2), V2=LETTERS[4:6])][]`	Alternative to the above one. With [], you print the result to the screen
`> DT[,V1:=NULL]`	Remove V1
`> DT[,c("V1","V2"):=NULL]`	Remove columns V1 and V2
`> Cols.chosen=c("A","B")` `> DT[,Cols.Chosen:=NULL]`	Delete the column with column name Cols.chosen
`> DT[,(Cols.Chosen):=NULL]`	Delete the columns specified in the variable Cols.chosen

Indexing And Keys

`> setkey(DT,V2)`	A key is set on V2; output is returned invisibly
`> DT["A"]`	Return all rows where the key column (set to V2) has the value A
`> DT[c("A","C")]`	Return all rows where the key column (V2) has value A or C
`> DT["A",mult="first"]`	Return first row of all rows that match value A in key column V2
`> DT["A",mult="last"]`	Return last row of all rows that match value A in key column V2
`> DT[c("A","D")]`	Return all rows where key column V2 has value A or D
`> DT[c("A","D"),nomatch=0]`	Return all rows where key column V2 has value A or D
`> DT[c("A","C"),sum(V4)]`	Return total sum of V4, for rows of key column V2 that have values A or C
`> DT[c("A","C"), sum(V4), by=.EACHI]`	Return sum of column V4 for rows of V2 that have value A, and anohter sum for rows of V2 that have value C
`> setkey(DT,V1,V2)`	Sort by V1 and then by V2 within each group of V1 (invisible)
`> DT[.(2,"C")]`	Select rows that have value 2 for the first key (V1) and the value C for the second key (V2)
`> DT[.(2,c("A","C"))]`	Select rows that have value 2 for the first key (V1) and within those rows the value A or C for the second key (V2)

Advanced Data Table Operations

`> DT[.N-1]`	Return the penultimate row of the DT
`> DT[,.N]`	Return the number of rows
`> DT[,.(V2,V3)]`	Return V2 and V3 as a data.table
`> DT[,list(V2,V3)]`	Return V2 and V3 as a data.table
`> DT[,mean(V3),by=.(V1,V2)]`	Return the result of j, grouped by all possible combinations of groups specified in by

.SD & .SDcols

`> DT[,print(.SD),by=V2]`	Look at what .SD contains
`> DT[,.SD[c(1,.N)],by=V2]`	Select the first and last row grouped by V2
`> DT[,lapply(.SD,sum),by=V2]`	Calculate sum of columns in .SD grouped by V2
`> DT[,lapply(.SD,sum),by=V2, .SDcols=c("V3","V4")]`	Calculate sum of V3 and V4 in .SD grouped by V2
`> DT[,lapply(.SD,sum),by=V2, .SDcols=paste0("V",3:4)]`	Calculate sum of V3 and V4 in .SD grouped by V2

Chaining

`> DT <- DT[,.(V4.Sum=sum(V4)), by=V1]`	Calculate sum of V4, grouped by V1
`> DT[V4.Sum>40]`	Select that group of which the sum is >40
`> DT[,.(V4.Sum=sum(V4)), by=V1][V4.Sum>40]`	Select that group of which the sum is >40 (chaining)
`> DT[,.(V4.Sum=sum(V4)), by=V1][order(-V1)]`	Calculate sum of V4, grouped by V1, ordered on V1

set()-Family

set()

Syntax: `for (i in from:to) set(DT, row, column, new value)`

```
> rows <- list(3:4,5:6)
> cols <- 1:2
> for(i in seq_along(rows))
    {set(DT,
         i=rows[[i]],
         j=cols[i],
         value=NA)}
```

Sequence along the values of rows, and for the values of cols, set the values of those elements equal to NA (invisible)

setnames()

Syntax: `setnames(DT,"old","new")[]`

`> setnames(DT,"V2","Rating")`	Set name of V2 to Rating (invisible)
`> setnames(DT, c("V2","V3"), c("V2.rating","V3.DC"))`	Change 2 column names (invisible)

setnames()

Syntax: `setcolorder(DT,"neworder")`

`> setcolorder(DT, c("V2","V1","V4","V3"))`	Change column ordering to contents of the specified vector (invisible)

DataCamp

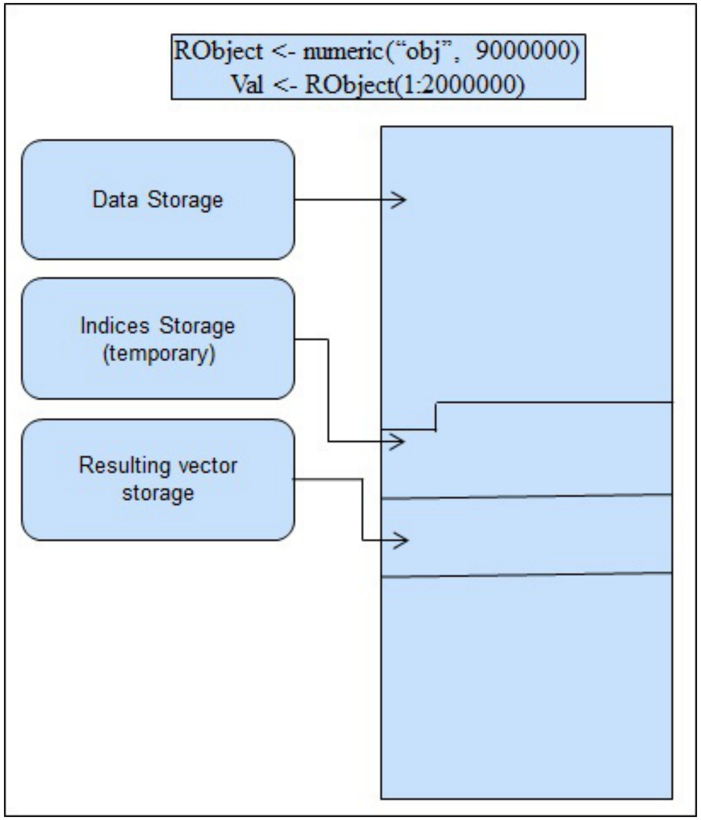
Learn Python for Data Science Interactively

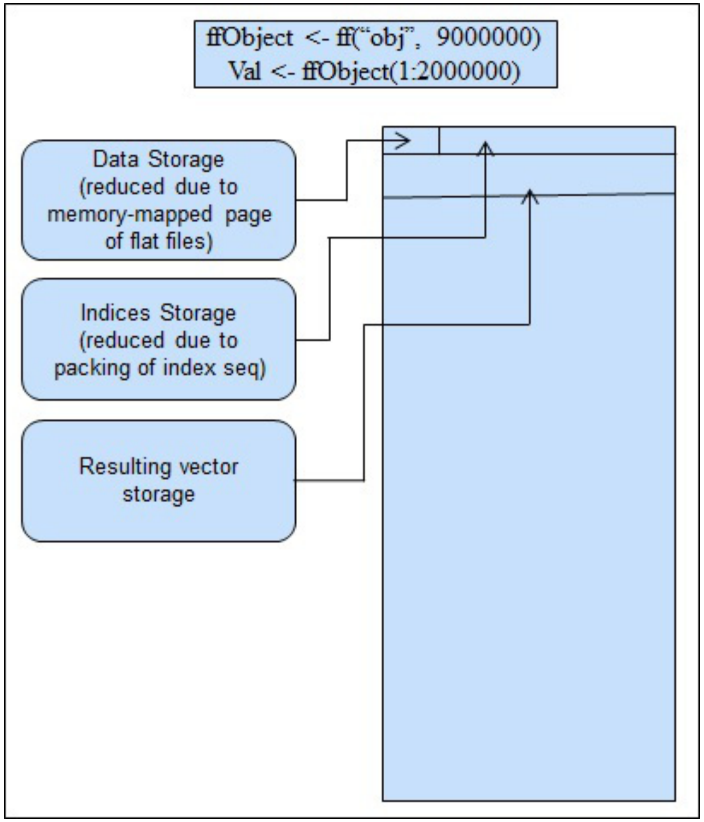
https://www.datacamp.com/community/tutorials/data-table-cheat-sheet

Karlijn Willems

Buffer the data set on disk as in SAS

ffdf object ff package





http://hsinay.blogspot.ch/p/big-data-analysis-using-ff-and-12-nov.html ffdf object from ff package

buffers the data set on disk as in SAS

ff Advantage:

Works a lot like a standard date frame, only reading in data only on demand

ff Drawback:

Proceed with caution when dealing with column types

Solutions to bypass the limitation

- Get a bigger computer
- Format the data differently
- Make the data smaller

data smaller = subsetting

data smaller = subsetting

+ remove unnecessary data

Pseudocode

```
>rows <- [1:500]
>columns <- [1:30]
>subset <- bigdata[rows, columns]
>rm(bigdata)
```

Challenge

Can you divide the brfss data into chunks of 500 random and try computing an odds ratio?

Challenge: Solution

Can you divide the brfss data into chunks of 500 random and try computing an odds ratio?

```
>rows_to_select <- sample(1:nrow(brfss), 500, replace=F)
>brfss_sample <- brfss[rows_to_select,]
>oddsratio(as.factor(brfss$X_HCVU651),as.factor(brfss$X_RF
CHOL))
```

data smaller = subsetting

data smaller = subsetting

directly from a database

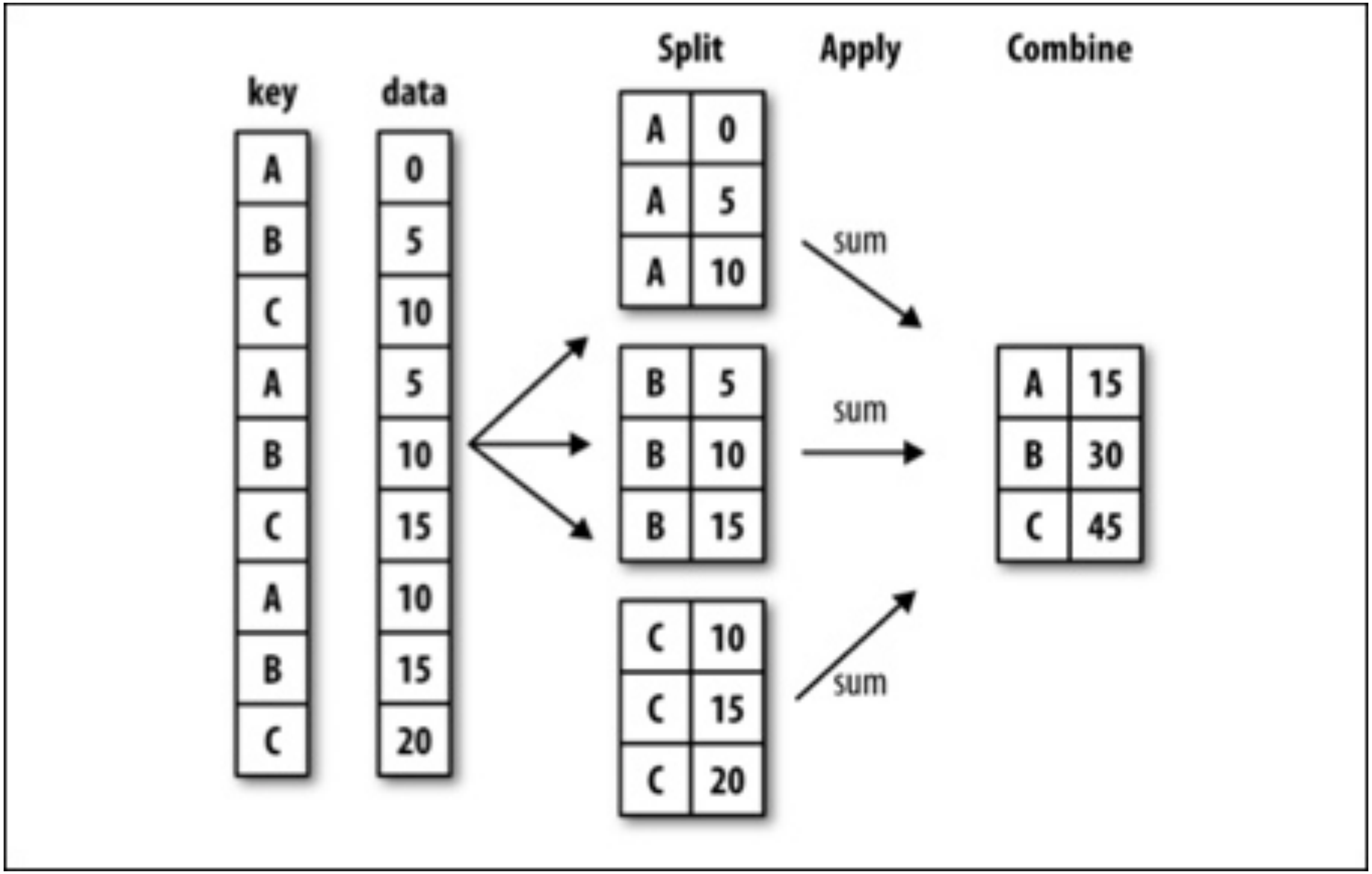
Subset using SQL query

R packages: “RODBC” or “RMySQL”

Once you split

Once you split you need to combine

split-apply-combine



split-apply-combine

There are many ways to do this in R.

Specifically:

by, aggregate, split, and plyr, cast, tapply, data.table, dplyr, and so forth.

Data set to illustrate the different functions

```
#Calculate mean per group (mean by group)
>df <- data.frame(
         group=factor(sample(c("g1","g2"), 10,
                     replace=TRUE)),
         mortality=runif(10))
```

10 rows, 2 groups

Data set to illustrate the different functions

```
#Calculate mean per group (mean by group)
>df <- data.frame(
         group=factor(sample(c("g1","g2"), 10,
                     replace=TRUE)),
         mortality=runif(10))
>df
>dt<- data.table(df)
>setkey(dt, mortality)
```

	group	mortality
1	g1	0.80668490
2	g1	0.53349584
3	g2	0.07571784
4	g2	0.39518628
5	g1	0.84557955
6	g1	0.69121443
7	g1	0.38124950
8	g2	0.22536126
9	g1	0.04704750
10	g2	0.93561651

split-apply-combine: Tapply function

```
>tapply(df$mortality, df$group, mean)
```

split-apply-combine: Aggregate function

aggregate takes in data.frames, outputs data.frames, and uses a formula interface.

```
>aggregate(mortality~ group, df, mean)
```

split-apply-combine: By function

In its most user-friendly form, it takes in vectors and applies a function to them. However, its output is not in a very manipulable form

```
>res.by <- by(df$mortality, df$group, mean)
>res.by
```

To get around this, for simple uses of by the as.data.frame method in the taRifx library works:

```
>library(taRifx)
>as.data.frame(res.by)
```

split-apply-combine: Split function

As the name suggests, it performs only the "split" part of the split-apply-combine strategy.

To test it here is the a small function that uses sapply for apply-combine.

```
>splitmean <- function(df) {
    s <- split( df, df$group)
    sapply( s, function(x)
    mean(x$mortality) )
}
>splitmean(df)
```

split-apply-combine: data.table structure

```
>library(data.table)
>setDT(df)[ , .(mean_mortality = 
mean(mortality)), by = group]
```

split-apply-combine: Reshape2 function

The reshape2 library is not designed with split-apply-combine as its primary focus. Instead, it uses a two-part melt/cast strategy to perform a wide variety of data reshaping tasks. However, since it allows an aggregation function it can be used for this problem

```
>library(reshape2)
>dcast( melt(df), variable ~ group,
mean)
```

split-apply-combine: plyr and dplyr packages

Hadley Wickham in plyr package adresses performance on very large datasets

plyr (the pre-cursor of dplyr)

split-apply-combine: Dplyr package

```
>library(dplyr)
>group_by(df,group) %>%
summarize(m=mean(mortality))
```

split-apply-combine: Plyr package

If you have to learn one tool for split-apply-combine manipulation it should be plyr.

```
>library(plyr)
>res.plyr <- ddply( df, .(group),
function(x) >mean(x$mortality) )
>res.plyr
```

Wrap up on memory

If your data is just too big, there are several things you can do:

- Get a bigger computer
- Format the data differently
- Make the data smaller : split & combine

What is Big? (for this course)

What gets more difficult when data is big?

- Visualization
  - Visualizations get messy
- Memory issues
  - The data may not load into memory
- Computational time
  - Analyzing the data may take a long time
- Etc.

Computational time

Modeling and computational time

Sometimes you can load the data, but analyzing it is slow

Sometimes you can load the data, but analyzing it is painfully slow

Implementation matters

R was built by statisticians,
not by data miners.

R aren't the best IMHO

If you’re doing a lot of computation

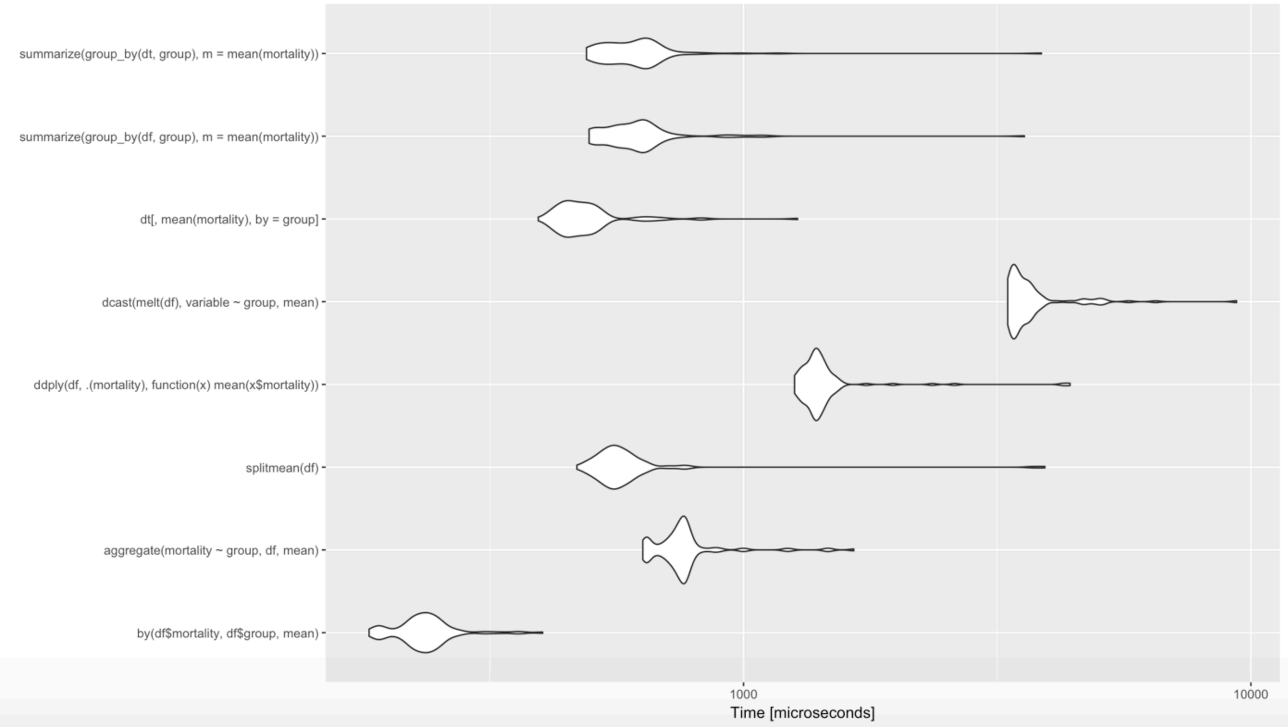
PROFILE your code

If you're doing a lot of computation

PROFILE your code

(i.e. time your code)

Benchmark the different methods



Benchmark the different methods

```
>library(microbenchmark)
>m1 <- microbenchmark(
  by( df$mortality, df$group, mean),
  aggregate(mortality~ group, df, mean ),
  splitmean(df),
  ddply( df, .(group), function(x) mean(x$mortality) ),
  dcast( melt(df), variable ~ group, mean),
  dt[, mean(mortality), by = group],
  summarize( group_by(df, group), m = mean(mortality) ),
  summarize( group_by(dt, group), m = mean(mortality) )
)
>print(m1, signif = 3)
>autoplot(m1)
```

Benchmark the different methods

```
>library(microbenchmark)
>m1 <- microbenchmark(
  by( df$mortality, df$group, mean),
  aggregate(mortality~ group, df, mean ),
  splitmean(df),
  ddply( df, .(group), function(x) mean(x$mortality) ),
  dcast( melt(df), variable ~ group, mean),
  dt[, mean(mortality), by = group],
  summarize( group_by(df, group), m = mean(mortality) ),
  summarize( group_by(dt, group), m = mean(mortality) )
)
>print(m1, signif = 3)
>autoplot(m1)
```

Benchmark the different methods

```
>library(microbenchmark)
>m1 <- microbenchmark(
  by( df$mortality, df$group, mean),
  aggregate(mortality~ group, df, mean ),
  splitmean(df),
  ddply( df, .(group), function(x) mean(x$mortality) ),
  dcast( melt(df), variable ~ group, mean),
  dt[, mean(mortality), by = group],
  summarize( group_by(df, group), m = mean(mortality) ),
  summarize( group_by(dt, group), m = mean(mortality) )
)
>print(m1, signif = 3)
>autoplot(m1)
```

Benchmark the different methods

```
>library(microbenchmark)
>m1 <- microbenchmark(
  by( df$mortality, df$group, mean),
  aggregate(mortality~ group, df, mean ),
  splitmean(df),
  ddply( df, .(group), function(x) mean(x$mortality) ),
  dcast( melt(df), variable ~ group, mean),
  dt[, mean(mortality), by = group],
  summarize( group_by(df, group), m = mean(mortality) ),
  summarize( group_by(dt, group), m = mean(mortality) )
)
>print(m1, signif = 3)
>autoplot(m1)
```

Benchmark the different methods

```
>library(microbenchmark)
>m1 <- microbenchmark(
  by( df$mortality, df$group, mean),
  aggregate(mortality~ group, df, mean ),
  splitmean(df),
  ddply( df, .(group), function(x) mean(x$mortality) ),
  dcast( melt(df), variable ~ group, mean),
  dt[, mean(mortality), by = group],
  summarize( group_by(df, group), m = mean(mortality) ),
  summarize( group_by(dt, group), m = mean(mortality) )
)
>print(m1, signif = 3)
>autoplot(m1)
```

Benchmark the different methods

```
>library(microbenchmark)
>m1 <- microbenchmark(
  by( df$mortality, df$group, mean),
  aggregate(mortality~ group, df, mean ),
  splitmean(df),
  ddply( df, .(group), function(x) mean(x$mortality) ),
  dcast( melt(df), variable ~ group, mean),
  dt[, mean(mortality), by = group],
  summarize( group_by(df, group), m = mean(mortality) ),
  summarize( group_by(dt, group), m = mean(mortality) )
)
>print(m1, signif = 3)
>autoplot(m1)
```

Wrap up

- Plyr is always worth learning for its flexibility
- data.table is worth learning if you plan to analyze huge datasets
- by and aggregate and split are all base R functions and thus universally available

Challenge

Benchmark the different split and merge methods available in R when dataframe is composed of 1,000 groups and has 10000 then 10^7 rows:

What is Big? (for this course)

What gets more difficult when data is big?

- Visualization
  - Visualizations get messy
- Memory issues
  - The data may not load into memory
- Computational time
  - Analyzing the data may take a long time
- Etc.

Profiling several lines of code in R

Simple profiling

–Option 1:

```
system.time(<call>)
```

Profiling several lines of code in R

Simple profiling

–Option 2:

```
start_time <- proc.time()
<call>
end_time <- proc.time()
end_time – start_time
```

More advanced profiling options

Rprof is a function in the utils library that creates an external file with deep profiling results

Tricks to go faster

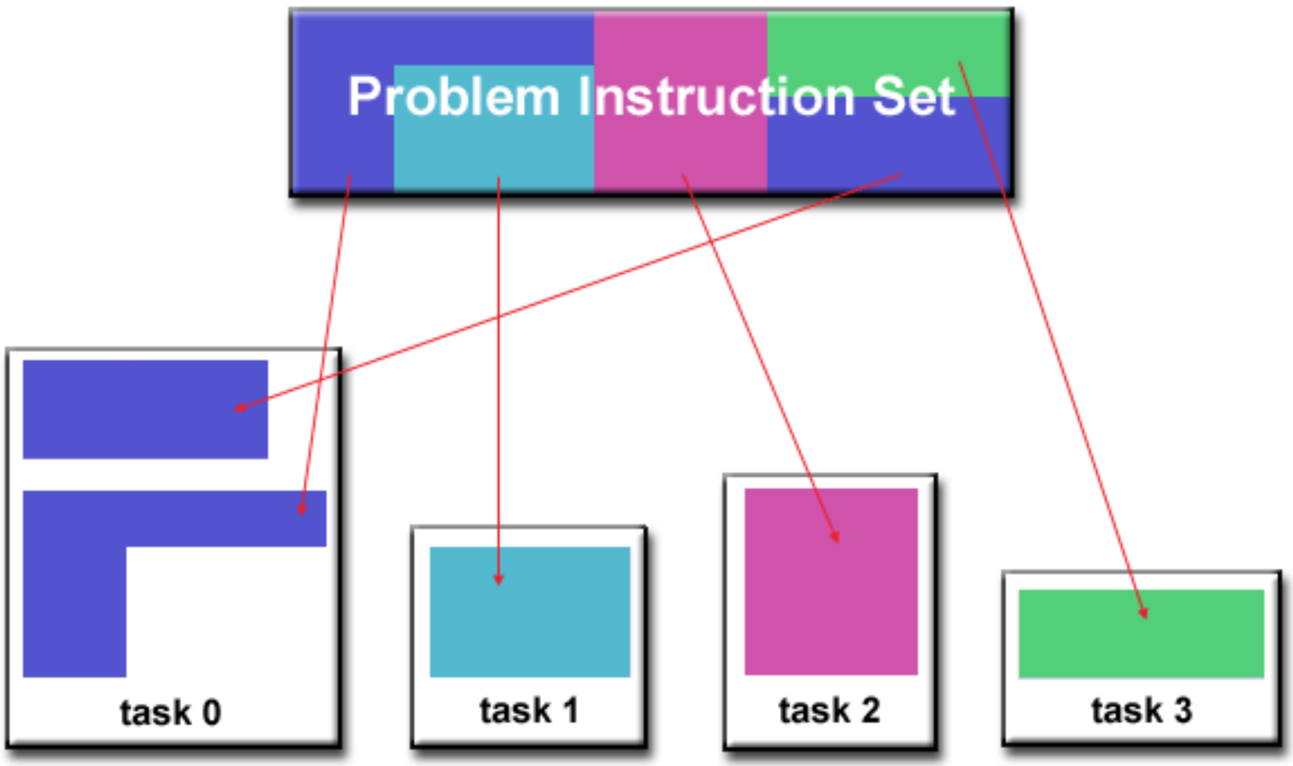
The compiler package

compile()   compiles a specific function
enableJIT() auto-compiles every function at first use

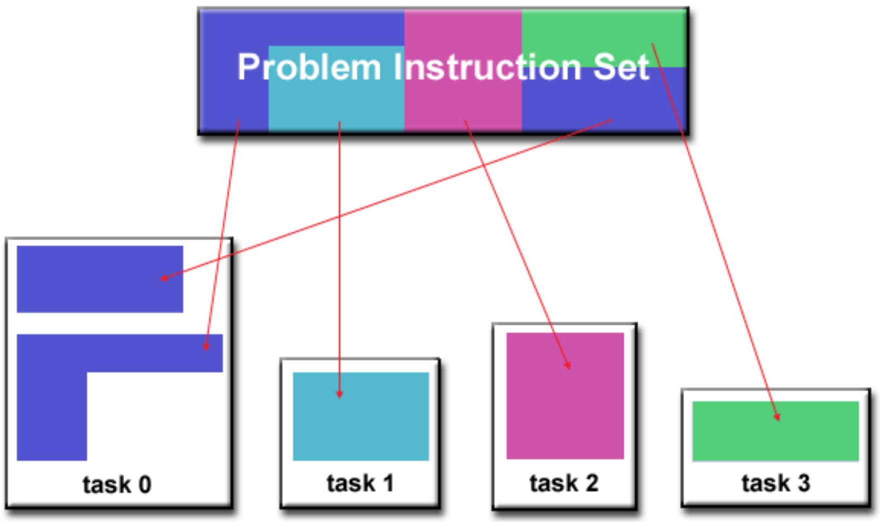
Tricks to go faster

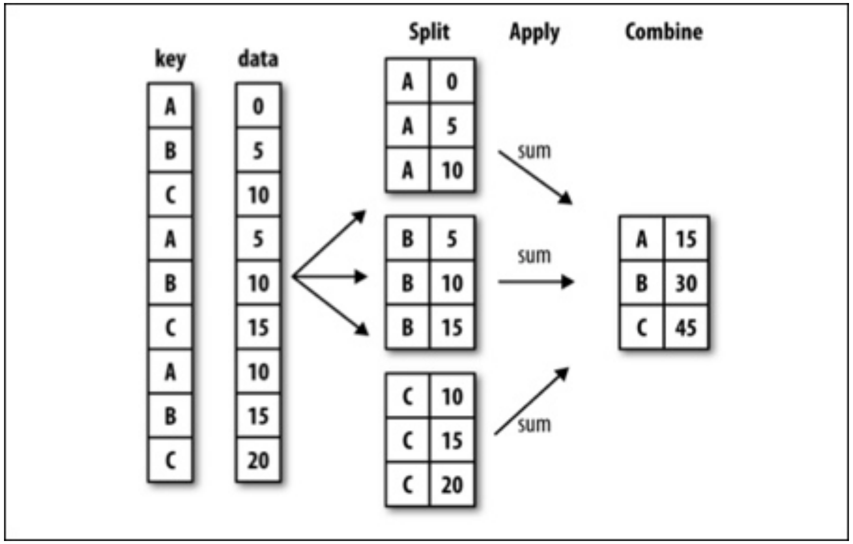
Go parallel

Parallel processing is basically splitting subtasks to independent processors, then merging results



Go parallel ≠ split combine





Fragment the instruction set

Fragment data

How to go parallel without explicitly doing parallel programming?

– **aaply**:

in Plyr package: like apply, but with an option to parallelize

– **foreach**:

allows you to write loops that can be parallelized

– **mclapply**:

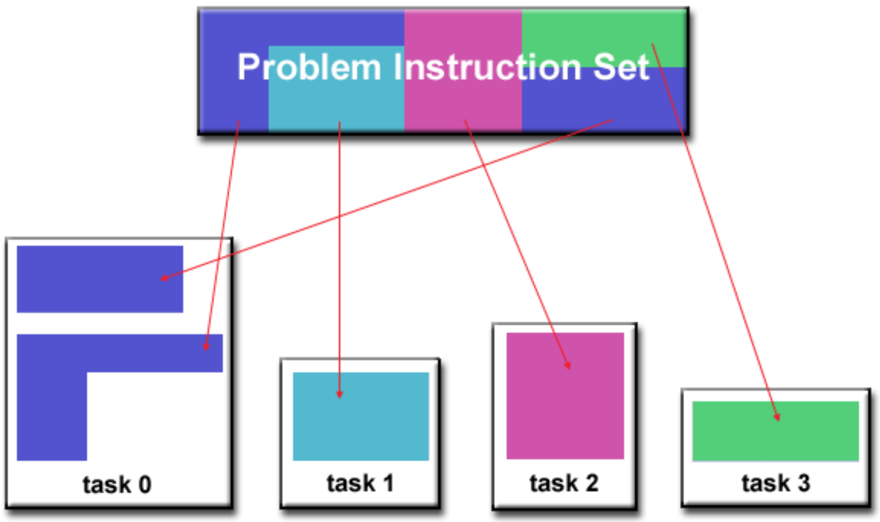
uses apply and themulticore of the machine

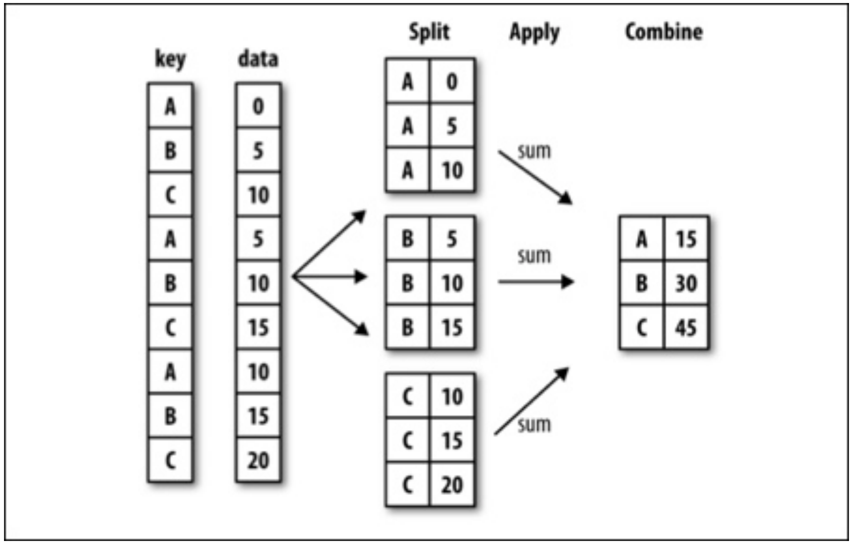
Going parallel issues

-How do you know when subsections of a task are independent?

-How do you know when you are done?

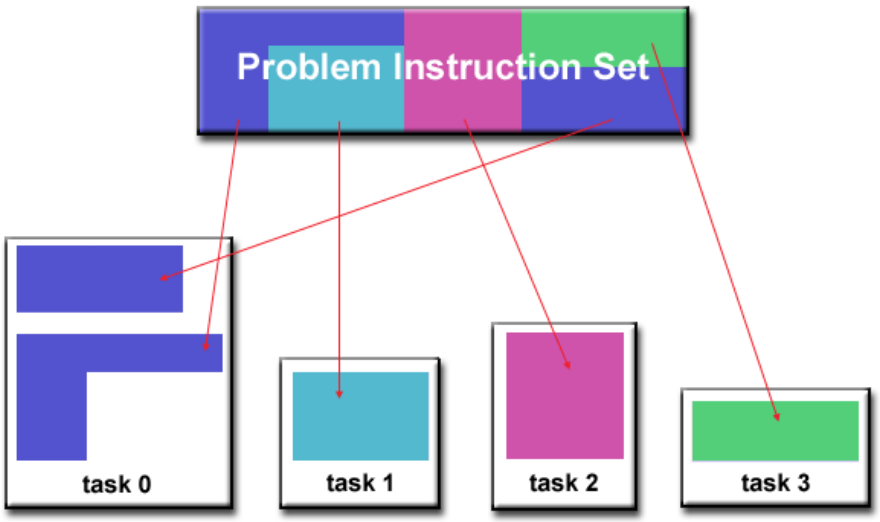
-New classes of potential mistakes: Race conditions, mutual exclusion, and deadlocks

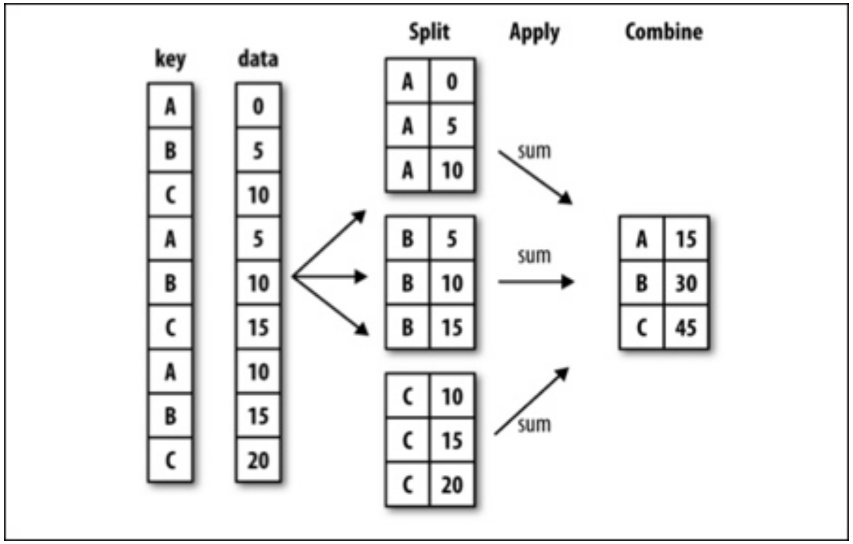




Fragment the instruction set

Fragment data





Fragment the instruction set

Fragment data

Fragment the instruction set + Fragment data = **MapReduce**

MapReduce is behind Hadoop concept in Big data

STOP!

What is **hadoop**?

Hadoop

Hadoop is a framework which was used to solve Big Data management challenges and it was introduced by Apache Software Foundation.

Current distributions:

- Apache Hadoop
- Cloudera
- Hortonworks
- MapR
- AWS
- Windows Azure HDInsignts

Hadoop

Hadoop is an open-source

It contains two modules

- **MapReduce**
- Hadoop Distributed File System (HDFS):

used to store and process the datasets.

Hadoop

Hadoop is an open-source

It contains two modules

- MapReduce
- **Hadoop Distributed File System** (HDFS):

used to store and process the datasets.

MapReduce in R

MapReduce library in R:

```
>library(mapReduce)
>mapReduce(map, reduce, data)
```

Takeaway message:

if you think your data needs MapReduce scale processing, talk to us

Challenge : profiling

ggplot2 dataset
>library(gg[plot2)
>diamonds

Compare the profiling time of *for* and *apply* functions that return TRUE

when color value == E

in diamonds dataset

Which case is faster?

Case1: Do some operation on every row using apply (which pre-allocates memory):

```
start_time <- proc.time()
apply(diamonds, 1, function(row) { row['color'] == 'E' })
proc.time() - start_time
```

Case2: Do the same operation but build the response vector through concatenation:

```
start_time <- proc.time()
e_diamonds <- c()
for (row in 1:nrow(diamonds)) {
e_diamonds <- c(e_diamonds, diamonds[row, 'color'] == 'E') }
e_diamonds
proc.time() - start_time
```

IMPORTANT

Course room

Monday, Tuesday, Wednesday:

– Génopode Building 2020

Thursday:

– Amphipôle Building 321

Thank you for your attention