Three hour tutorial

data.table

30 June 2014

useR! - Los Angeles

Matt Dowle

Overview

- data.table in a nutshell 20 mins
- Client server recorded demo 20 mins
- Main features in more detail 2 hours
- Q&A throughout 20 mins

Every question is a good question!

Feel free to interrupt.

What is data.table?

- Think `data.frame`, inherits from it
- `data.table()` and `?data.table`

Goals:

- Reduce programming time

  fewer function calls, less variable name repetition

- Reduce compute time

  fast aggregation, update by reference

- In-memory only, 64bit and 100GB routine
- Useful in finance but wider use in mind, too

  e.g. genomics

Reducing programming time

```
trades[
   filledShares < orderedShares,
   sum( (orderedShares-filledShares)
        * orderPrice / fx ),
   by = "date,region,algo"
]
```

Aside : could add database backend

```
R   :   i          j              by
SQL :   WHERE      SELECT         GROUP BY
```

Reducing compute time

e.g. 10 million rows x 3 columns x,y,v 230MB

```
DF[DF$x=="R" & DF$y==123,]   # 8     s
DT[.("R",123)]               # 0.008s

tapply(DF$v,DF$x,sum)        # 22    s
DT[,sum(v),by=x]             #  0.83s
```

See above in timings vignette (copy and paste)

Fast and friendly file reading

e.g. 50MB .csv, 1 million rows x 6 columns

```
read.csv("test.csv")              # 30-60s
read.csv("test.csv", colClasses=,
         nrows=, etc...)          #    10s
fread("test.csv")                 #     3s
```

e.g. 20GB .csv, 200 million rows x 16 columns

```
read.csv("big.csv", ...)          #  hours
fread("big.csv")                  #     8m
```

Update by reference using `:=`

Add new column ”sectorMCAP” by group :

```
DT[,sectorMCAP:=sum(MCAP),by=Sector]
```

Delete a column (0.00s even on 20GB table) :

```
DT[,colToDelete:=NULL]
```

Be explicit to really copy entire 20GB :

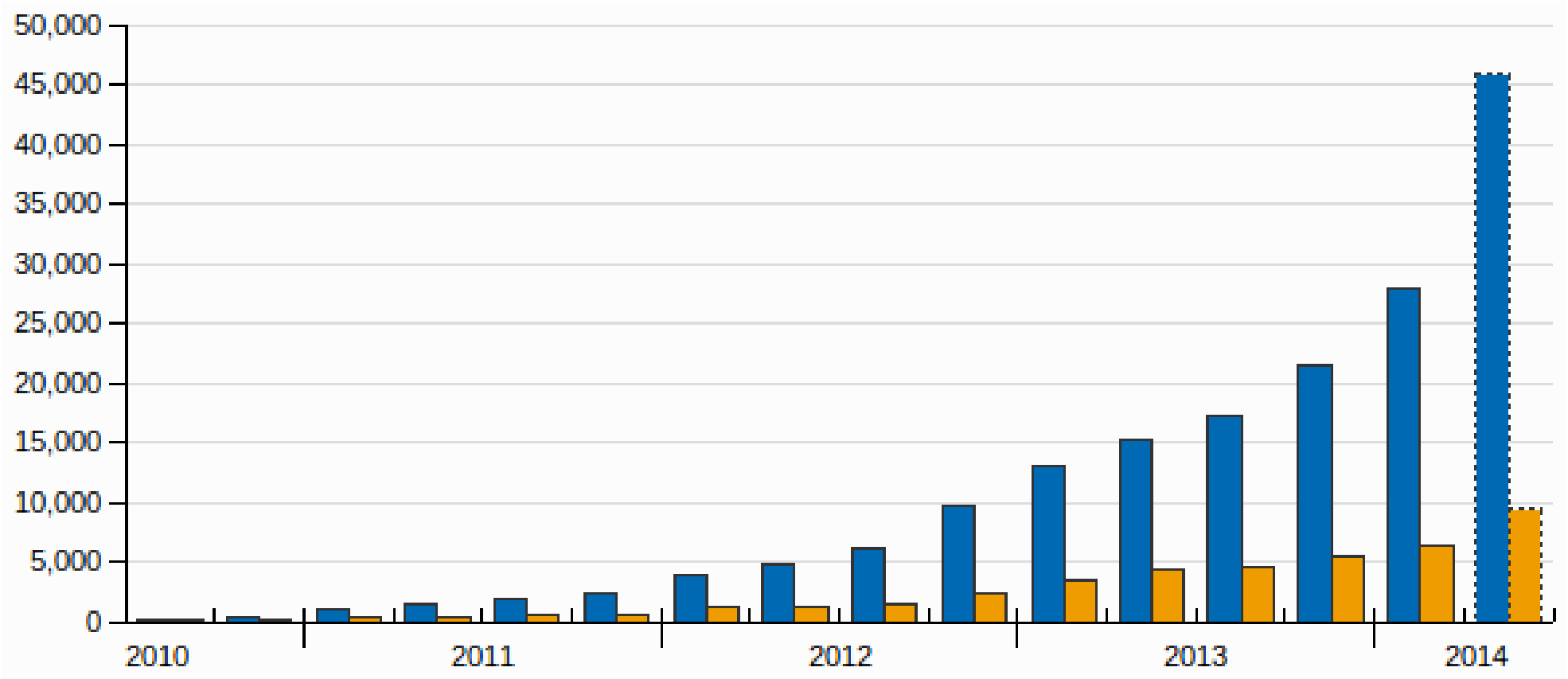
```
DT2 = copy(DT)
```

Why R?

1) R's lazy evaluation enables the syntax :
   - **`DT[ filledShares < orderedShares ]`**
   - query optimization before evaluation
2) Pass `DT` to any package taking `DF`. It works.

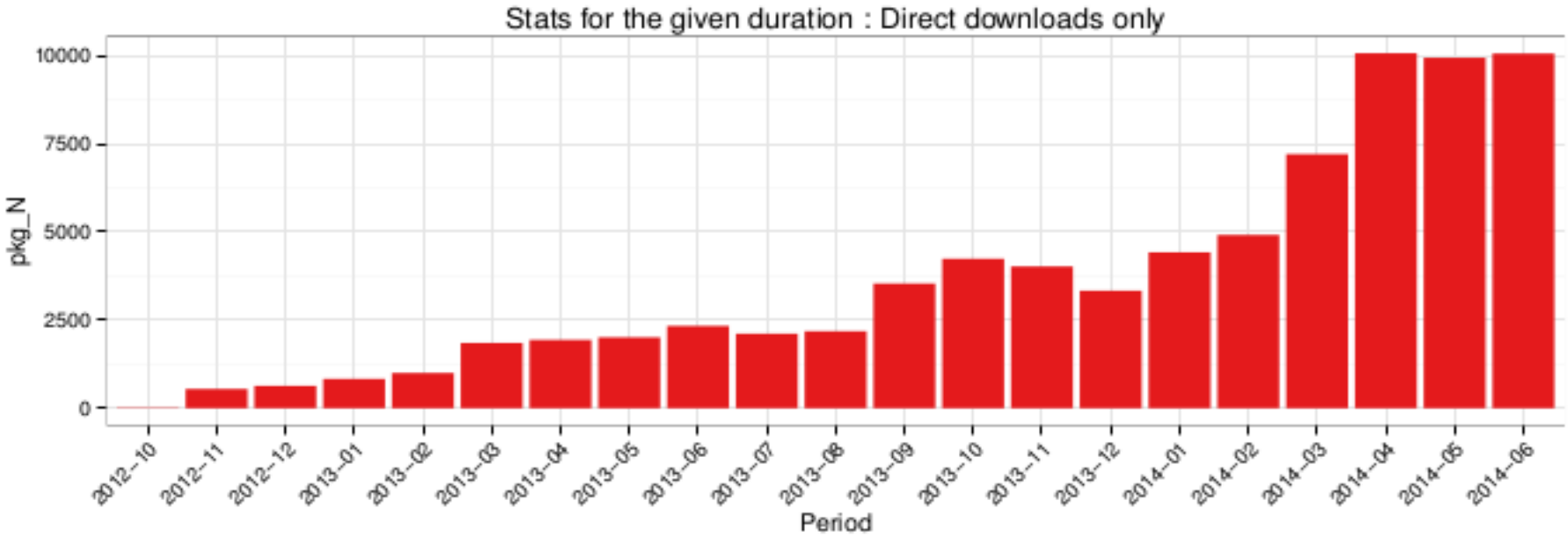
   `is.data.frame(DT) == TRUE`
3) CRAN (cross platform release, quality control)
4) Thousands of statistical packages to use with data.table

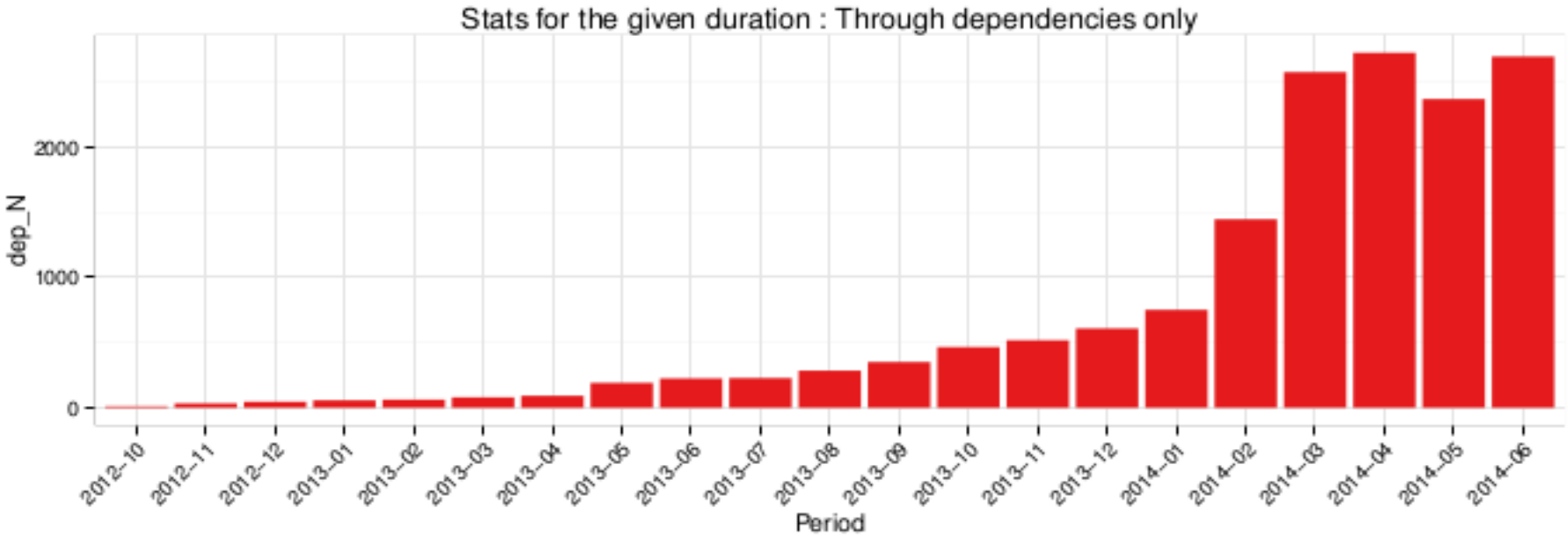
Web page visits



Downloads

RStudio mirror only





data.table support

21	Last 7 Days	19% unanswered
85	Last 30 Days	15.3% unanswered
1,542	All Time	8.6% unanswered

Client/server recorded demo

http://www.youtube.com/watch?v=rvT8XThGA8o

Main features in more detail ...

Essential!

- Given a 10,000 x 10,000 matrix in any language
- Sum the rows
- Sum the columns
- Is one way faster, and why?

setkey(DT, colA, colB)

- Sorts the table by colA then colB. That's all.
- Like a telephone number directory: last name then first name
- X[Y] is just binary search to X's key
- You **DO** need a key for joins X[Y]
- You **DO NOT** need a key for by= (but many examples online include it)

Example DT

x	y
B	7
A	2
B	1
A	5
B	9

DT[2:3,]

x	y
B	7
A	2
B	1
A	5
B	9

DT

x	y
B	7
A	2
B	1
A	5
B	9

setkey(DT, x)

x	y
A	2
A	5
B	7
B	1
B	9

DT[”B”,]

x	y
A	2
A	5
B	7
B	1
B	9

DT[”B”,mult=”first”]

x	y
A	2
A	5
B	7
B	1
B	9

DT[”B”,mult=”last”]

x	y
A	2
A	5
B	7
B	1
B	9

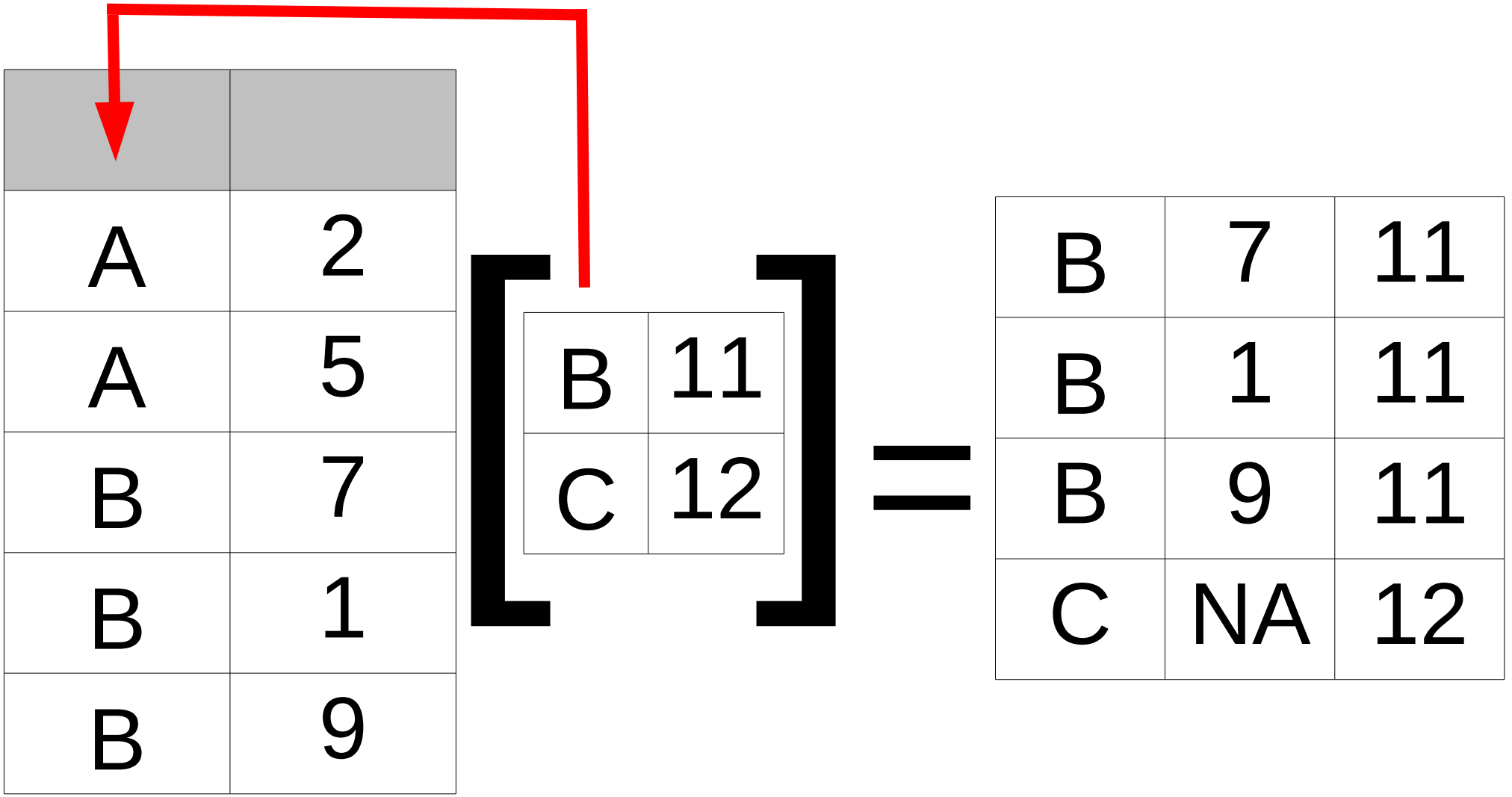
DT[”B”,sum(y)] == 17

x	y
A	2
A	5
B	7
B	1
B	9

DT[c(”A”,”B”),sum(y)] == 24

x	y
A	2
A	5
B	7
B	1
B	9

X[Y]



A	2
A	5
B	7
B	1
B	9

[

B	11
C	12

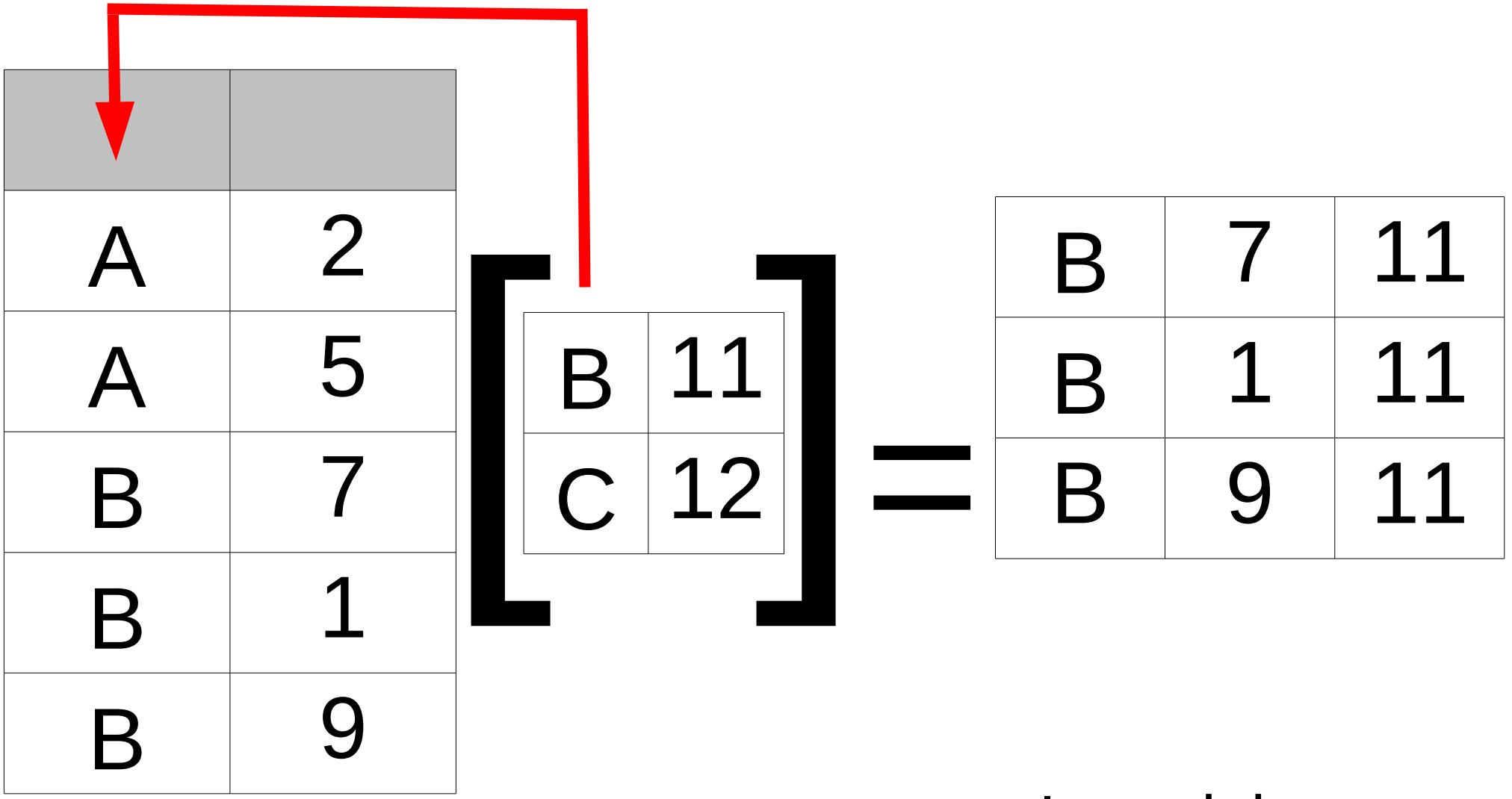
]

=

B	7	11
B	1	11
B	9	11
C	NA	12

Outer join by default (in SQL parlance)

X[Y, nomatch=0]



Inner join

`X[Y,head(.SD,n),by=.EACHI]`

x	y
A	2
A	5
B	7
B	1
B	9

Join inherited column

z	n
A	1
B	2

=

x	y
A	2
B	7
B	1

i.e. select a *data driven* topN for each i row

”Cold” by (i.e. without setkey)

Consecutive calls unrelated to key are fine and common practice :

```
> DT[, sum(v), by="x,y"]
> DT[, sum(v), by="z"]
> DT[, sum(v), by=colA%%5]
```

Also known as "ad hoc by"

Programatically vary by

```
bys = list ( quote(y%%2),
             quote(x),
             quote(y%%3) )

for (this in bys)
  print(X[,sum(y),by=eval(this)])

   this V1
1:    0  2
2:    1 22
   this V1
1:    A  7
2:    B 17
   this V1
1:    2  7
2:    1  8
3:    0  9
```

DT[i, j, by]

- Out loud: ”Take **`DT`**, subset rows using **`i`**, then calculate **`j`** grouped by **`by`**”

- Once you grok the above reading, you don't need to memorize any other functions as all operations follow the same intuition as base.

lapply and do.call running very slowly?

June 2012

I have a data frame that is some 35,000 rows, by 7 columns. it looks like this:

```
head(nuc)
```

```
  chr feature    start      end   gene_id    pctAT    pctGC length
1   1     CDS 67000042 67000051 NM_032291 0.600000 0.400000     10
2   1     CDS 67091530 67091593 NM_032291 0.609375 0.390625     64
3   1     CDS 67098753 67098777 NM_032291 0.600000 0.400000     25
4   1     CDS 67101627 67101698 NM_032291 0.472222 0.527778     72
5   1     CDS 67105460 67105516 NM_032291 0.631579 0.368421     57
6   1     CDS 67108493 67108547 NM_032291 0.436364 0.563636     55
```

gene_id is a factor, that has about 3,500 unique levels. I want to, for each level of gene_id get the `min(start)`, `max(end)`, `mean(pctAT)`, `mean(pctGC)`, and `sum(length)`.

I tried using lapply and do.call for this, but it's taking forever +30 minutes to run. the code I'm using is:

```
nuc_prof = lapply(levels(nuc$gene_id), function(gene){
  t = nuc[nuc$gene_id==gene, ]
  return(list(gene_id=gene, start=min(t$start), end=max(t$end), pctGC =
              mean(t$pctGC), pct = mean(t$pctAT), cdslength = sum(t$length)))
})
nuc_prof = do.call(rbind, nuc_prof)
```

I'm certain I'm doing something wrong to slow this down. I haven't waited for it to finish as I'm sure it can be faster. Any ideas?

data.table answer

10

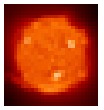
Since I'm in an evangelizing mood ... here's what the fast `data.table` solution would look like:

```
library(data.table)
dt <- data.table(nuc, key="gene_id")

dt[,list(A=min(start),
         B=max(end),
         C=mean(pctAT),
         D=mean(pctGC),
         E=sum(length)), by=key(dt)]
#      gene_id        A        B         C         D   E
# 1: NM_032291 67000042 67108547 0.5582567 0.4417433 283
# 2:       ZZZ 67000042 67108547 0.5582567 0.4417433 283
```

link | edit | flag

answered **Jun 15 at 16:14**



Josh O'Brien

20.4k ● 2 ● 14 ● 40

NB: It isn't just the speed, but the simplicity. It's easy to write and easy to read.

User's reaction

”data.table is awesome! That took about 3 seconds for the whole thing!!!”

Davy Kavanagh, 15 Jun 2012

but ...

- Example had **`by=key(dt)`** ?
- Yes, but it didn't need to.
- If the data is very large (1GB+) and the groups are big too then getting the groups together in memory can speed up a bit (cache efficiency).

by= and keyby=

- Both **by** and **keyby** retain row order within groups – important, often relied on
- Unlike SQL
- **by** retains order of the groups (by order of first appearance) - important, often relied on

---

```
setkeyv(DT[,,by=],by)
DT[,,keyby=]                    # same, shortcut
```

Prevailing joins (roll=TRUE)

- One reason for setkey's design.
- Last Observation (the prevailing one) Carried Forward (LOCF), efficiently
- Roll forwards or backward
- Roll the last observation forwards, or not
- Roll the first observation backwards, or not
- Limit the roll; e.g. 30 days (roll = 30)
- Join to nearest value (roll = ”nearest”)
- i.e. ***ordered joins***

… continued

- **`roll = [-Inf,+Inf] | TRUE | FALSE | "nearest"`**
- **`rollends = c(FALSE,TRUE)`**
- By example ...

PRC

id	date	price
SBRY	20080501	380.50
SBRY	20080502	391.50
SBRY	20080506	389.00
VOD	20080501	159.30
VOD	20080502	163.30
VOD	20080506	160.80

setkey(PRC, id, date)

1. PRC[.(“SBRY”)] # all 3 rows
2. PRC[.(“SBRY”,20080502),price] # 391.50
3. PRC[.(“SBRY”,20080505),price] # NA
4. PRC[.(“SBRY”,20080505),price,roll=TRUE] # 391.50
5. PRC[.(“SBRY”,20080601),price,roll=TRUE] # 389.00
6. PRC[.(“SBRY”,20080601),price,roll=TRUE,rollends=FALSE] # NA
7. PRC[.(“SBRY”,20080601),price,roll=20] # NA
8. PRC[.(“SBRY”,20080601),price,roll=40] # 389.00

Performance

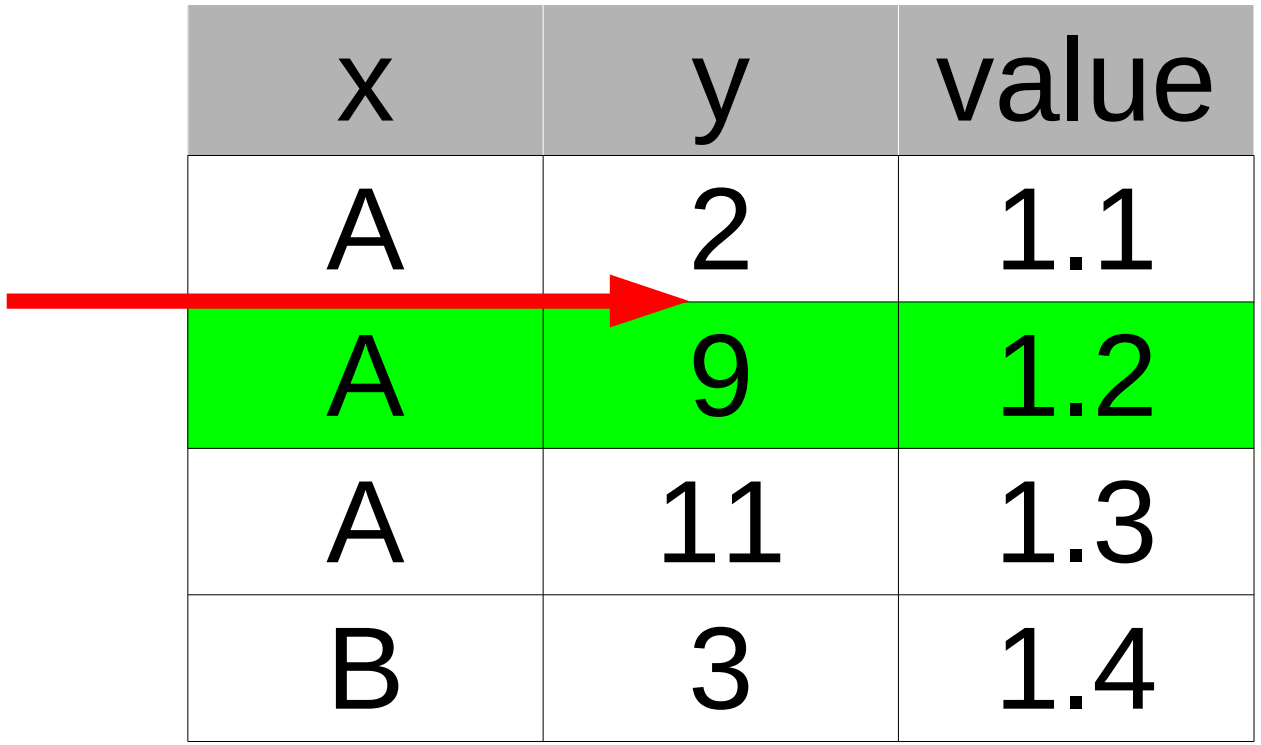
All daily prices 1986-2008 for all non-US equities
- 183,000,000 rows (id, date, price)
- 2.7 GB

```
system.time(PRICES[id==”VOD”])    # vector scan
user      system      elapsed
66.431    15.395      81.831

system.time(PRICES[“VOD”])        # binary search
user      system      elapsed
0.003     0.000       0.002
```

setkey(PRICES, id, date) needed first (one-off apx 20 secs)

roll = ”nearest”



x	y	value
A	2	1.1
A	9	1.2
A	11	1.3
B	3	1.4

```
setkey(DT, x, y)

DT[.(”A”,7), roll=”nearest”]
```

Variable name repetition

- The 3rd highest voted [R] question (of 43k)

  How to sort a dataframe by column(s) in R (*)

- DF[with(DF, order(-z, b)),]
  - vs -
  DT[order(-z, b)]

- quarterlyreport[with(lastquarterlyreport,order(-z,b)),]
  - vs -
  quarterlyreport[order(-z, b)]

Silent incorrect results due to using a similar variable by mistake. Easily done when this appears on a page of code.

(*) Click link for more information

but ...

- Yes order() is slow when used in i because that's base R's order().
- That's where ”optimization before evaluation” comes in. We now auto convert order() to the internal forder() so you don't have to know.
- Available in v1.9.3 on GitHub, soon on CRAN

split-apply-combine

Why ”split” 10GB into many small groups???

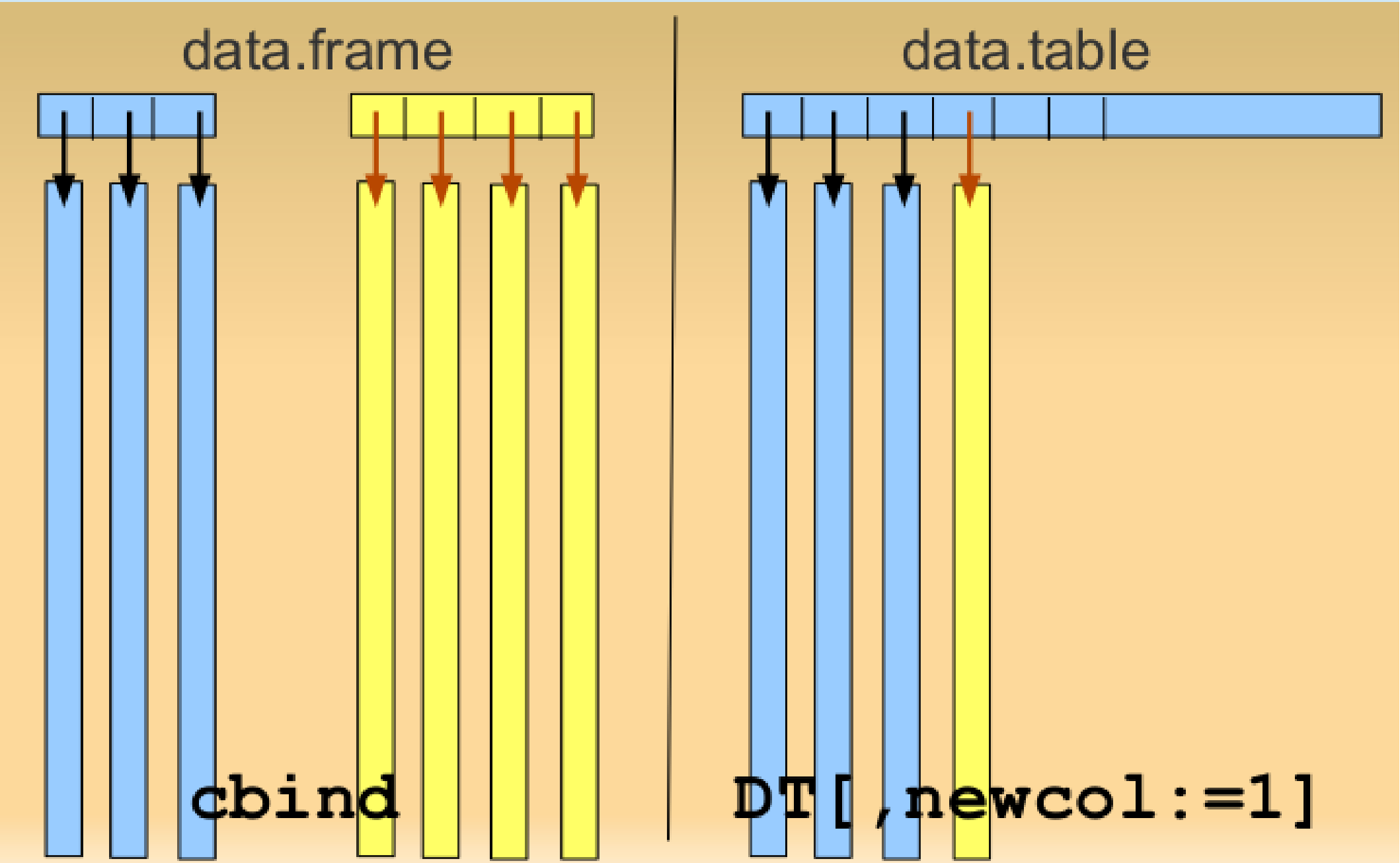
Since 2010, data.table :

- Allocates memory for largest group
- Reuses that same memory for all groups
- Allocates result data.table up front
- Implemented in C
- eval() of j within each group

Recent innovations

- Instead of the eval(j) from C, dplyr converts to an Rcpp function and calls that from C. Skipping the R eval step.
- In response, data.table now has **GForce**: one function call that computes the aggregate across groups. Called once only so no need to speed up many calls!
- Both approaches limited to simple aggregates: sum, mean, sd, etc. But often that's all that's needed.

data.table over-allocates



Assigning to a subset

In R I find myself doing something like this a lot:

```
 adataframe[adataframe$col==something]<-
adataframe[adataframe$col==something)]+1
```

This way is kind of long and tedious. Is there some way for me to reference the object I am trying to change such as

```
adataframe[adataframe$col==something]<-$self+1
```

?

continued

Try package data.table and its `:=` operator. It's very fast and very short.

```
DT[col1==something, col2:=col3+1]
```

The first part `col1==something` is the subset. You can put anything her
as if they are variables; i.e., no need to use `$` . Then the second part `co`
to the LHS within that subset, where the column names can be assigned t

---

Easy to write, easy to read

Multiple `:=`

```
DT[, `:=`(newCol1=mean(colA),
          newCol2=sd(colA)),
     by=sector]
```

- Can combine with a subset in `i` as well
- `` `:=` `` is functional form and standard R e.g. `` `<-` `` and `` `=`(x,2) ``

`set*` functions

- `set()`
- `setattr()`
- `setnames()`
- `setcolorder()`
- `setkey()`
- `setkeyv()`
- `setDT()`
- `setorder()`

copy()

- data.table **IS** copied-on-change by `<-` and `=` as usual in R. Those ops aren't changed.
- No copy by `:=` or `set*`
- You have to use those, so it's clear to readers of your code
- When you need a copy, call `copy(DT)`
- Why copy a 20GB data.table, even once.
- Why copy a whole column, even once.

list columns

- Each cell can be a different type
- Each cell can be vector
- Each cell can itself be a data.table
- Combine list columns with i and by

list column example

```
data.table(
   x = letters[1:3],
   y = list(  1:10,
              letters[1:4],
              data.table(a=1:3,b=4:6)
))
```

```
    x   y
1:  a   1,2,3,4,5,6,
2:  b   a,b,c,d
3:  c   <data.table>
```

All options

```
datatable.verbose                FALSE
datatable.nomatch                NA_integer_
datatable.optimize               Inf
datatable.print.nrows            100L
datatable.print.topn             5L
datatable.allow.cartesian        FALSE
datatable.alloccol               quote(max(100L,ncol(DT)+64L))
datatable.integer64              "integer64"
```

All symbols

- `.N`
- `.SD`
- `.I`
- `.BY`
- `.GRP`

.SD

```
stocks[, head(.SD,2), by=sector]
```

```
stocks[, lapply(.SD, sum), by=sector]
```

```
stocks[, lapply(.SD, sum), by=sector,
.SDcols=c("mcap",paste0(revenueFQ",1:8))]
```

.I

```
if (length(err <- allocation[,
                if(length(unique(Price))>1) .I,
                by=stock ]$V1 )) {

  warning("Fills allocated to different
accounts at different prices! Investigate.")

  print(allocation[err])

} else {

  cat("Ok   All fills allocated to each
account at same price\n")

}
```

Analogous to SQL

```
DT[ where,
    select | update,
    group by ]
  [ having ]
  [ order by ]
  [ i, j, by ] ... [ i, j, by ]
```

i.e. chaining

New in v1.9.2 on CRAN

- 37 new features and 43 bug fixes
- set() can now add columns just like :=
- .SDcols “de-select” columns by name or position; e.g.,

```
DT[,lapply(.SD,mean),by=colA,.SDcols=-c(3,4)]
```

- fread() a subset of columns
- fread() commands; e.g.,

```
fread("grep blah file.txt")
```

- Speed gains

Radix sort for integer

- R's method=”radix” is not actually a radix sort … it's a counting sort. See ?setkey/Notes.
- data.table liked and used it, though.
- A true radix sort caters for range > 100,000
- (Negatives was a one line change to R we suggested and was accepted in R 3.1)
- Adapted to integer from Terdiman and Herf's code for float …

Radix sort for numeric

- R reminder: numeric == floating point numbers
- Radix Sort Revisited, Pierre Terdiman, 2000

  http://codercorner.com/RadixSortRevisited.htm
- Radix Tricks, Michael Herf, 2001

  http://stereopsis.com/radix.html
- Their C code now in data.table with minor changes; e.g., NA/NaN and 6-pass for double

Radix sort for numeric

- R reminder: numeric == floating point numbers
- Radix Sort Revisited, Pierre Terdiman, 2000

  http://codercorner.com/RadixSortRevisited.htm
- Radix Tricks, Michael Herf, 2001

  http://stereopsis.com/radix.html
- Their C code now in data.table with minor changes; e.g., NA/NaN and 6-pass for double

Faster for those cases

20 million rows x 4 columns, 539MB

a & b (numeric), c (integer), d (character)

	v1.8.10	v1.9.2
setkey(DT, a)	54.9s	5.3s
setkey(DT, c)	48.0s	3.9s
setkey(DT, a, b)	102.3s	6.9s
”Cold” grouping (no setkey first) :		
DT[, mean(b), by=c]	47.0s	3.4s

https://gist.github.com/arunsrinivasan/451056660118628befff

New feature: melt

i.e. reshape2 for data.table

20 million rows x 6 columns (a:f) 768MB

melt(DF, id="d", measure=1:2) 4.1s (*)

melt(DT, id="d", measure=1:2) 1.7s

(*) including Kevin Ushey's C code in reshape2, was 190s

melt(DF, ..., na.rm=TRUE) 39.5s

melt(DT, ..., na.rm=TRUE) 2.7s

https://gist.github.com/arunsrinivasan/451056660118628befff

New feature: dcast

i.e. reshape2 for data.table

20 million rows x 6 columns (a:f) 768MB

dcast(DF, d~e, ..., fun=sum) 76.7 sec

dcast(DT, d~e, ..., fun=sum) 7.5 sec

reshape2::dcast hasn't been Kevin'd yet

https://gist.github.com/arunsrinivasan/451056660118628befff

… melt/dcast continued

Q: Why not submit a pull request to reshape2 ?

A: This C implementation calls data.table internals at C-level (e.g. fastorder, grouping, and joins). It makes sense for this code to be together.

Miscellaneous 1

```
DT[, (myvar):=NULL]
```

Spaces and specials; e.g., `by="a, b, c"`

```
DT[4:7,newCol:=8][]
```

- extra `[]` to print at prompt
- auto fills rows 1:3 with NA

```
rbindlist(lapply(fileNames, fread))
```

`rbindlist` has `fill` and `use.names`

Miscellaneous 2

Dates and times

Errors & warnings are deliberately very long

Not joins **`X[!Y]`**

Column plonk & non-coercion on assign

by-without-by => **`by=.EACHI`**

Secondary keys / merge

R3, singleton logicals, reference counting

bit64::integer64

Miscellaneous 3

Print method vs typing DF, copy fixed in R-devel

How to benchmark

mult = ”all” | ”first” | ”last”   (may expand)

with=FALSE

which=TRUE

CJ() and SJ()

Chained queries: DT[...][...][...]

Dynamic and flexible queries (eval text and quote)

Miscellaneous 4

fread **`drop`** and **`select`** (by name or number)

fread **`colClasses`** can be ranges of columns

fread **`sep2`**

Vector search vs binary search

One column == is ok, but not 2+ due to temporary logicals (e.g. slide 5 earlier)

Not (that) much to learn

- Main manual page: `?data.table`
- Run `example(data.table)` at the prompt (53 examples)
- No methods, no functions, just use what you're used to in R

Thank you

https://github.com/Rdatatable/datatable/

http://stackoverflow.com/questions/tagged/data.table

```
> install.packages("data.table")
> require(data.table)
> ?data.table
> ?fread
```

Learn by example :

```
> example(data.table)
```