data.table

Köln R User Group

December 2013

Matt Dowle

Arun Srinivasan

30 minute outline

Matt	
▪ Lightning talk R/Finance Chicago	5 mins
▪ More detail for new users	10 mins
Arun	
▪ News from latest version	5 mins
Questions / whiteboard	10 mins

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 8GB+ 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"
]
```

```
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")                     #   450s
```

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

Further detail …

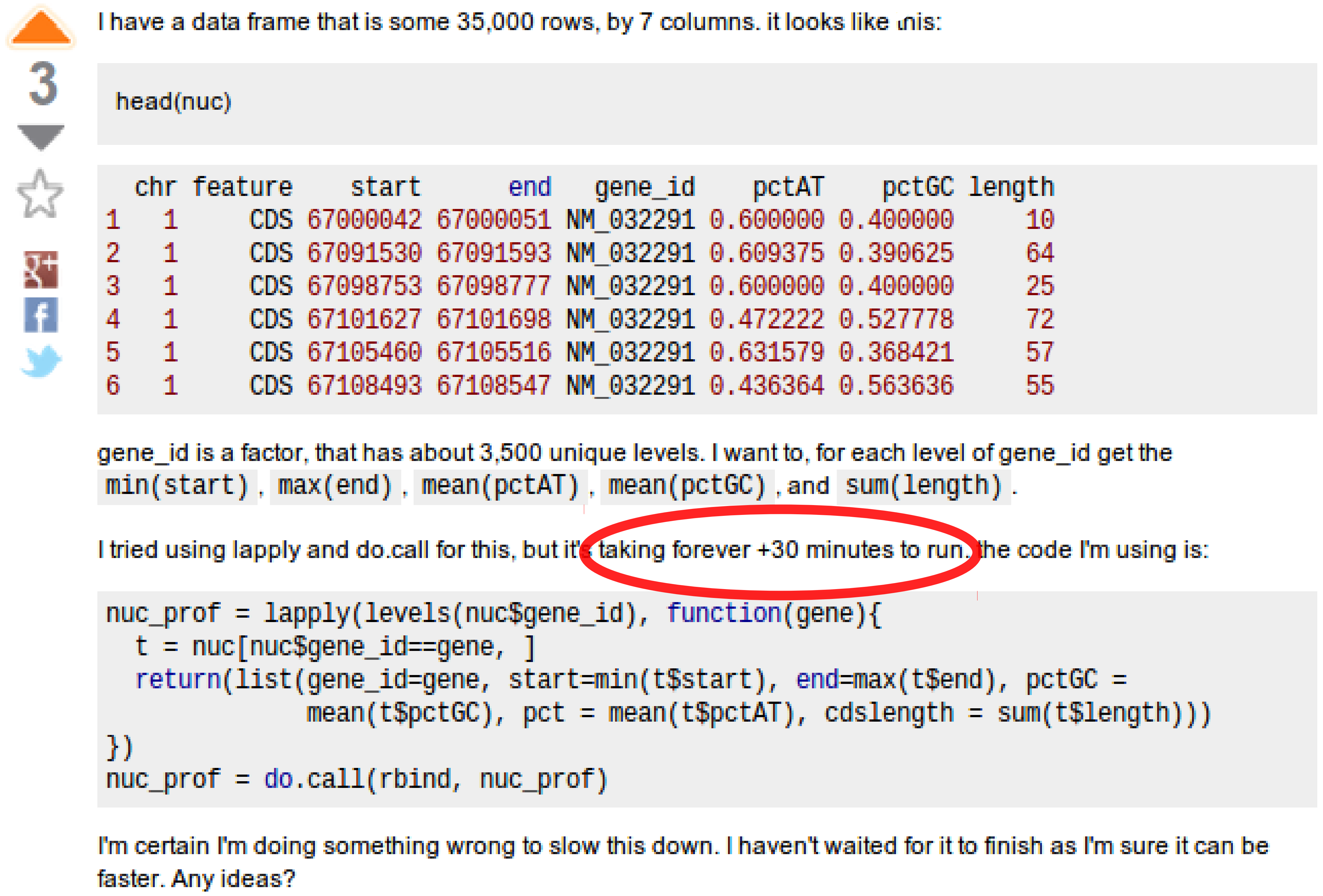
DT[i, j, by]

- 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



3

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
```

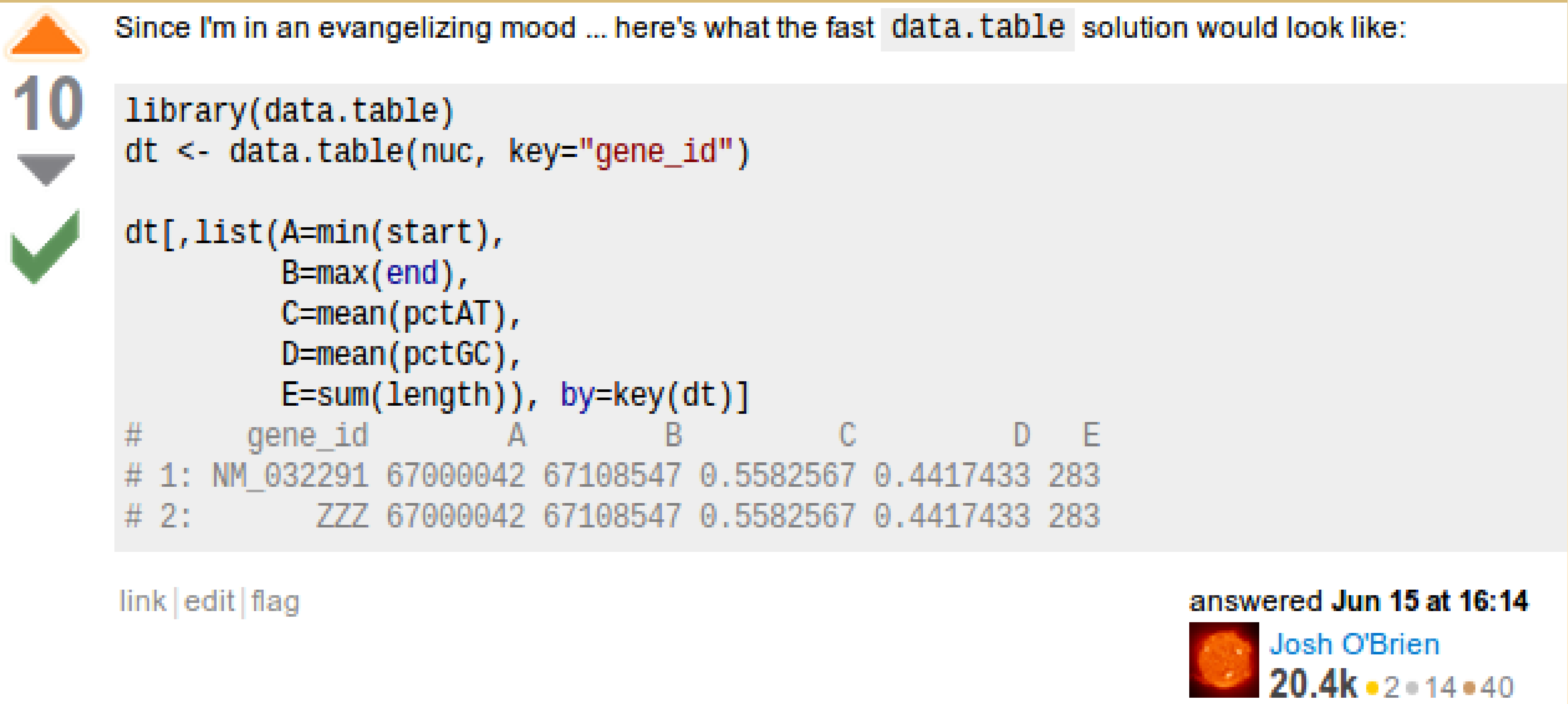
gene_id is a factor, that has about 3,500 unique levels. I want to, for each level of gene_id get the `min(start)`, `max(end)`, `mean(pctAT)`, `mean(pctGC)`, and `sum(length)`.

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



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
```

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

User's reaction

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

”I think that congratulations are well in order for the frankly amazingly well written quick start guide and FAQ. Seriously.”

Davy Kavanagh, 15 Jun 2012

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)

”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"

but ...

- Example a few slides back 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).

Prevailing join (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***

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 intend to auto convert order() to the internal fastorder() so you don't have to know.
- We already do quite a bit of optimization, but order() in i hasn't been done yet.

split-apply-combine

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

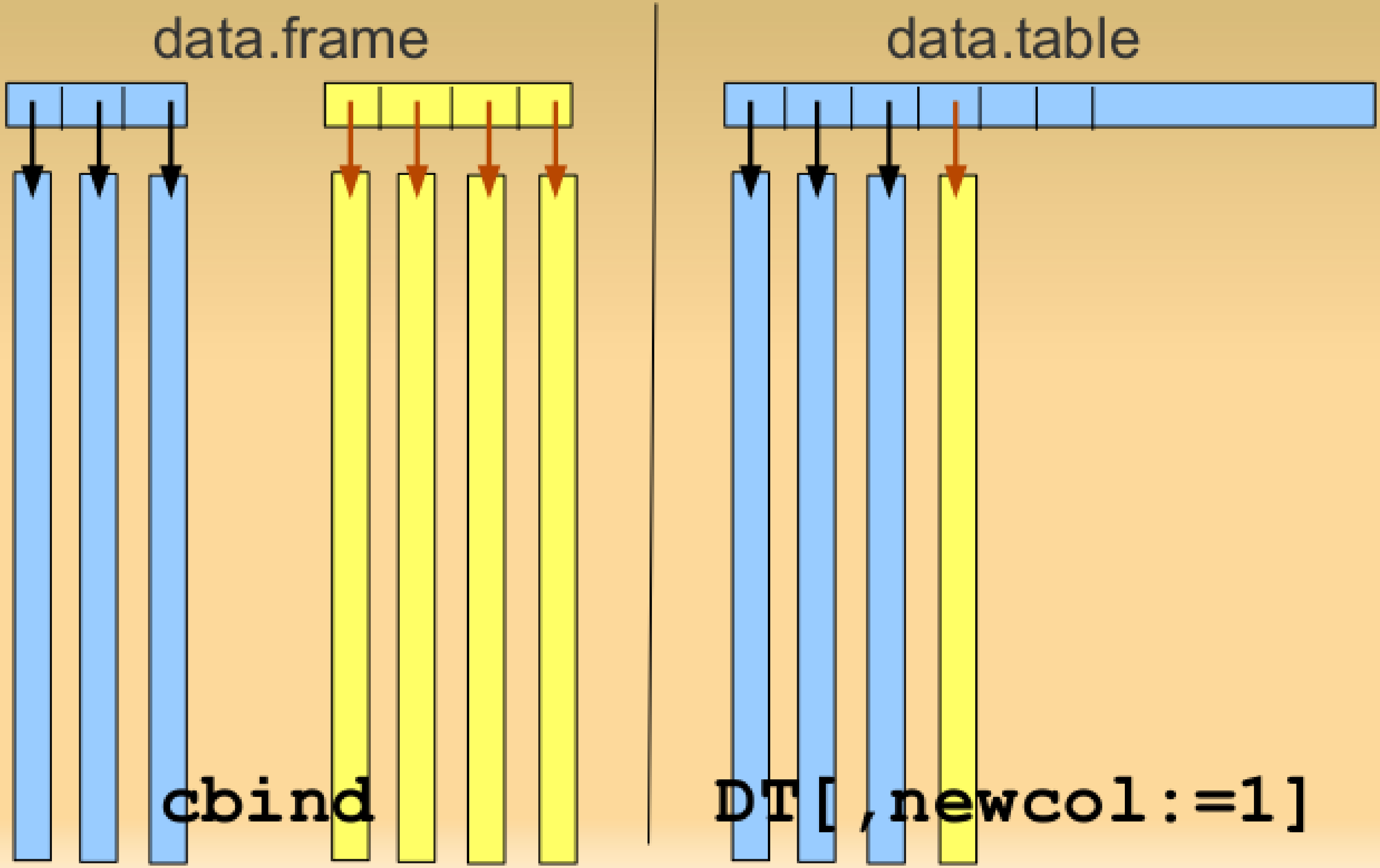
Since 2010, data.table :

- 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 innovation

- By Romain Francois and Hadley Wickham in dplyr
- Instead of the eval(j) from C, they convert to an Rcpp function and call that from C. Skipping the R eval step.
- Very fast!
- Very promising!
- We'll take a close look.

data.table over-allocates



:= and `:=`()

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


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

Analogous to SQL

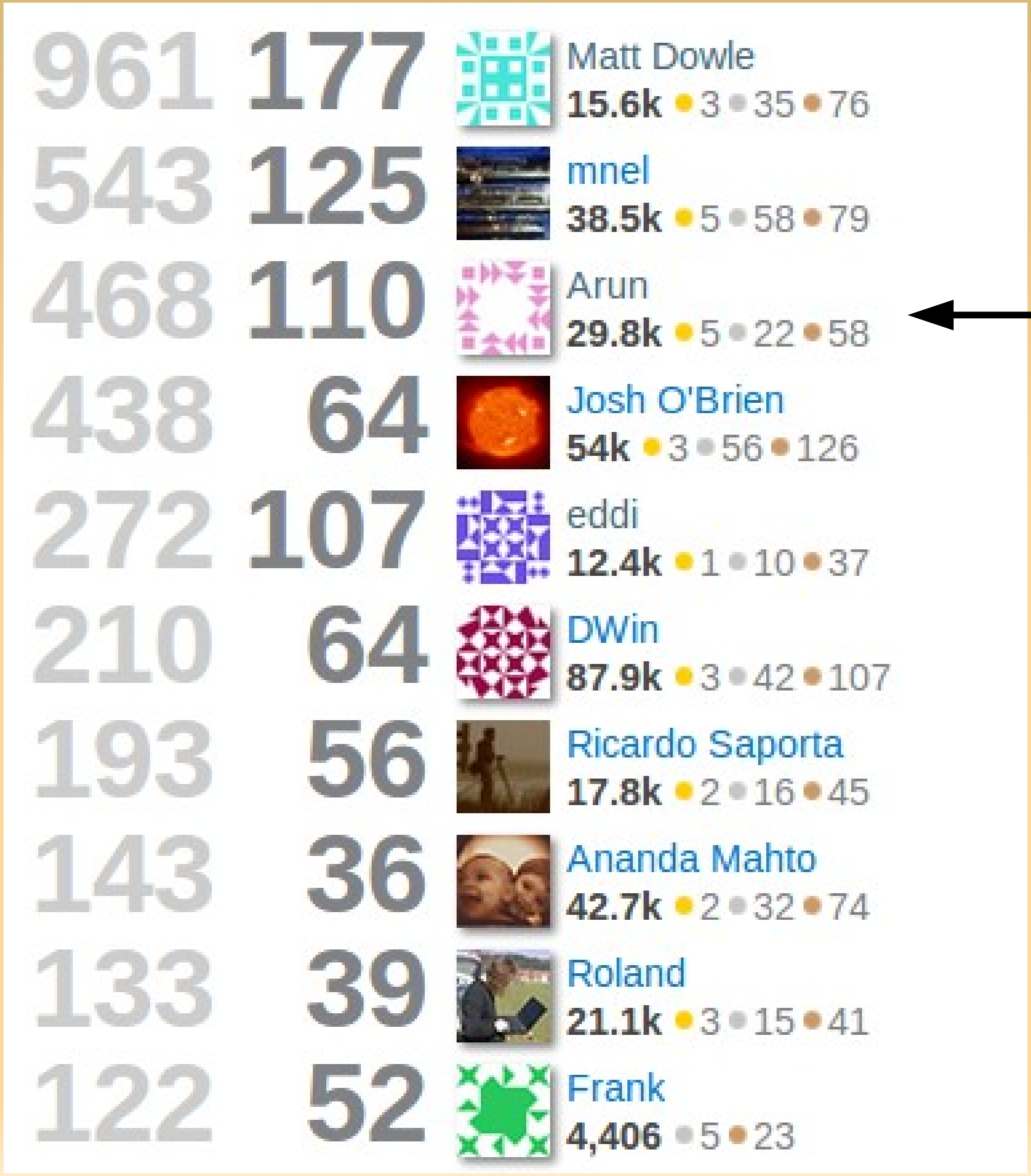
```
DT[ where,
    select | update,
    group by ]
  [ having ]
  [ order by ]
  [ i, j, by ] ... [ i, j, by ]
```

Dankeschön

Handover to Arun ...

Vibrant data.table community

950+ questions



961	177	Matt Dowle	15.6k ●3 ●35 ●76
543	125	mnel	38.5k ●5 ●58 ●79
468	110	Arun	29.8k ●5 ●22 ●58
438	64	Josh O'Brien	54k ●3 ●56 ●126
272	107	eddi	12.4k ●1 ●10 ●37
210	64	DWin	87.9k ●3 ●42 ●107
193	56	Ricardo Saporta	17.8k ●2 ●16 ●45
143	36	Ananda Mahto	42.7k ●2 ●32 ●74
133	39	Roland	21.1k ●3 ●15 ●41
122	52	Frank	4,406 ●5 ●23

Me

Next version (v1.8.11)

- 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 fixes edge cases :
  - Range > 100,000
  - Negatives
- 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

Faster for those cases

20 million rows x 4 columns, 539MB

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

	v1.8.10	v1.8.11
setkey(DT, a)	54.9s	7.2s
setkey(DT, c)	48.0s	7.0s
setkey(DT, a, b)	102.3s	16.9s
”Cold” grouping (no setkey first) :		
DT[, mean(b), by=c]	47.0s	8.7s

https://gist.github.com/arunsrinivasan/7997273

New feature: melt/cast

i.e. reshape2 for data.table

20 million rows x 6 columns (a:f)	768MB
melt(**DF**, id=”d”, measure=1:2)	191 sec
melt(**DT**, id=“d”, measure=1:2)	3 sec
dcast(**DF**, d~e, ..., fun=sum)	184 sec
dcast(**DT**, d~e, ..., fun=sum)	28 sec

https://gist.github.com/arunsrinivasan/7839891

Similar to melt_ in Kmisc by Kevin Ushey

… melt/cast 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.

Closing comments

- data.table vs dplyr preliminary benchmarks including time to group_by() not just summarise() : https://gist.github.com/arunsrinivasan/7997521
- Syntax, speed and memory efficiency.
- A feature that’ll make data.table “complete” for genomics – ‘interval trees’ e.g. IRanges, GenomicRanges from Bioconductor.   Some day!

Dankeschön

http://datatable.r-forge.r-project.org/

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

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

Learn by example :

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