250 Northern Ave, Boston, MA 02210
Phone: 844-448-1212

Email: info@rstudio.com
Web: http://www.rstudio.com

Work with Big Data in R

Ideas and Advice

Garrett Grolemund

Data Scientist and Master Instructor
November 2015
Email: garrett@rstudio.com

Lifecycle of an Analysis Project

Clarify
Become familiar with the data, template a solution

Develop
Create a working model

Productize
Automate and integrate

Publish
Socialize

R Markdown

Lifecycle of an Analysis Project

Clarify

Become familiar with the data, template a solution

A scripting **language** that is very useful for analyzing data.

Develop

Create a working model

Productize

Automate and integrate

Publish

Socialize

R Markdown

Velocity

Volume

Value

Variety

Veracity

Data > RAM

Lifecycle of an Analysis Project

Subset

Extract data to explore, work with

Clarify

Become familiar with the data, template a solution

Develop

Create a working model

Productize

Automate and integrate

Publish

Socialize

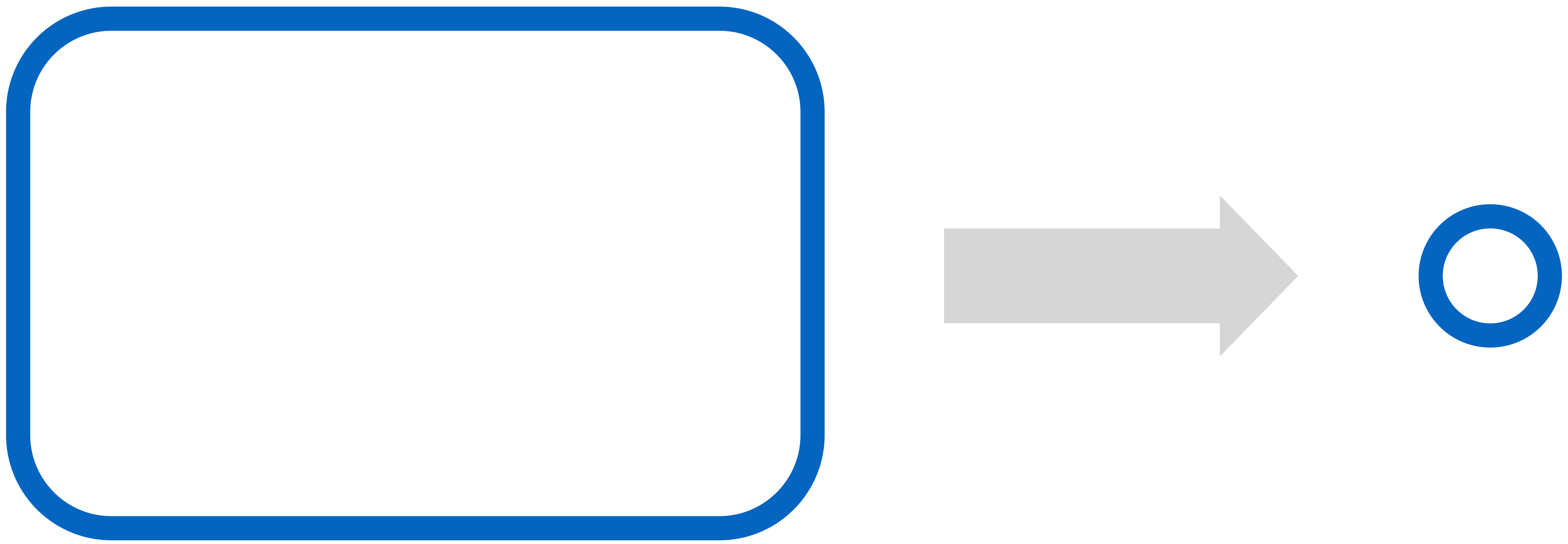
* sometimes

How do **analysts** use big data?

Analytic Big Data Problems

Class 1. Extract Data

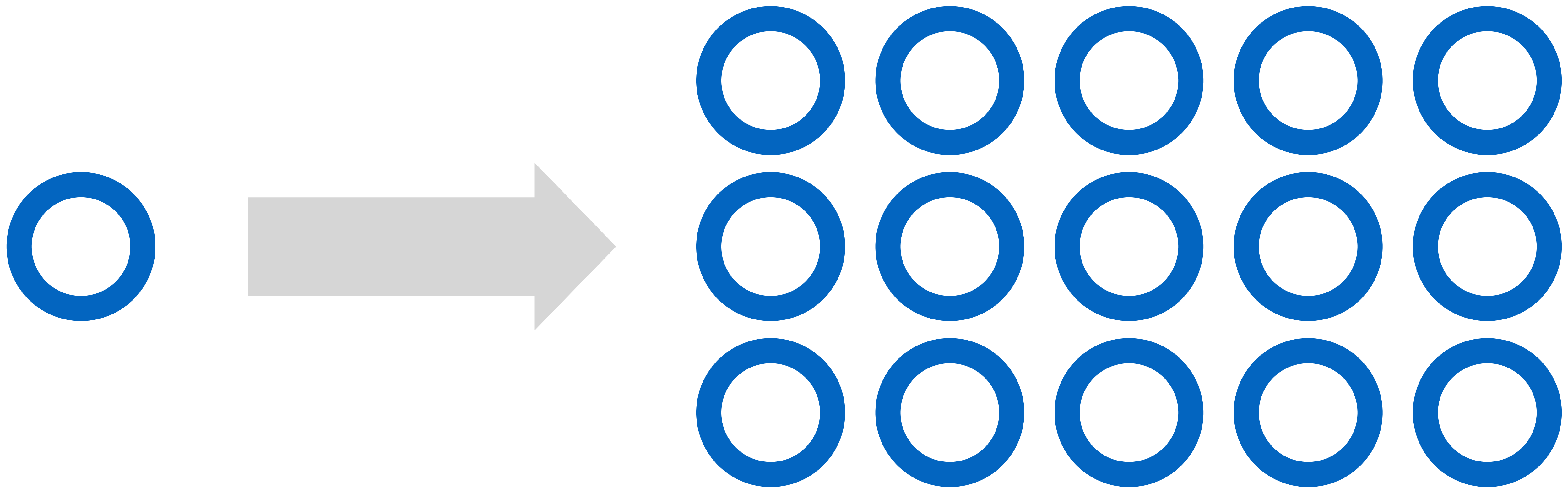
Problems that require you to extract a subset, sample, or summary from a Big Data source. You may do further analytics on the subset, and the subset might itself be quite large.



Analytic Big Data Problems

Class 2. Compute on the parts

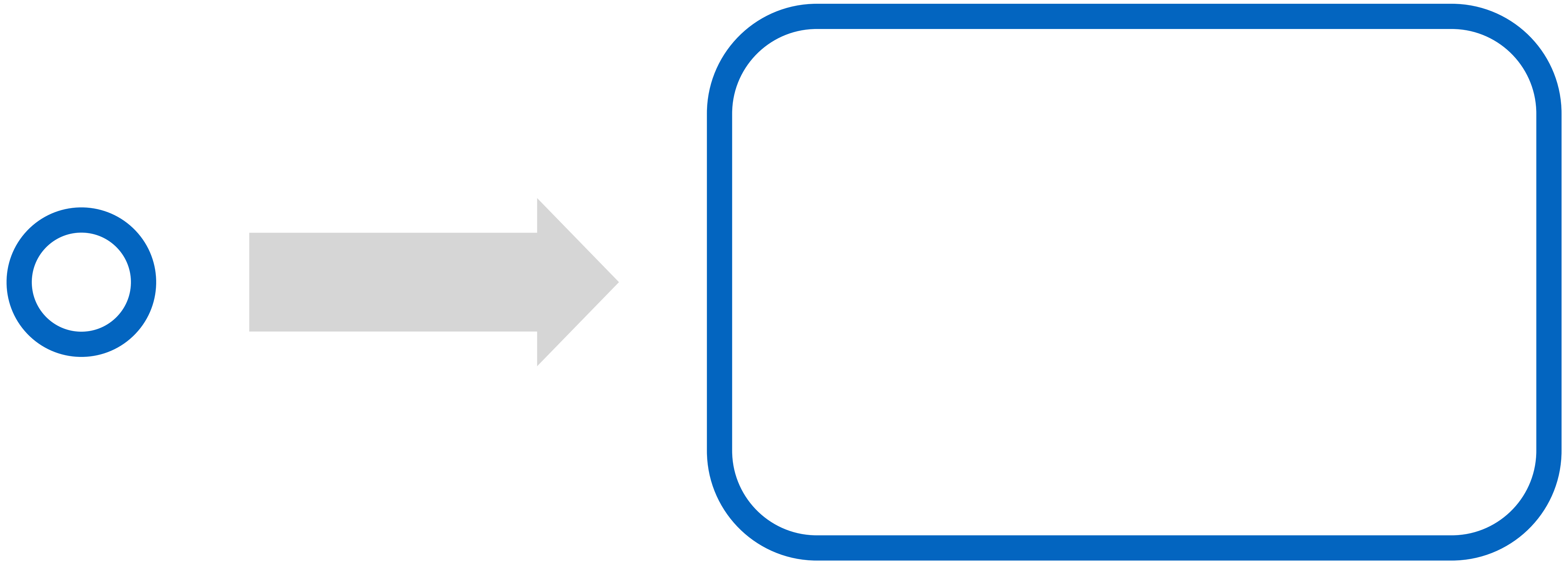
Problems that require you to repeat computation for many subgroups of the data, e.g. you need to fit one model per individual for thousands of individuals. You may combine the results once finished.



Analytic Big Data Problems

Class 3. Compute on the whole

Problems that require you to use all of the data at once. These problems are irretrievably big; they must be run at scale within the data warehouse.



Lifecycle of an Analysis Project

Subset

Extract data to explore, work with

Clarify

Become familiar with the data,
template a solution

Develop

Create a working model

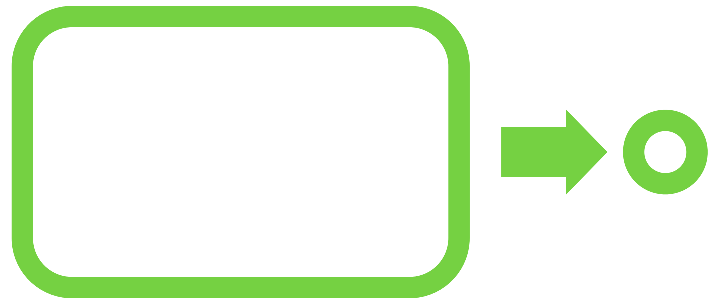
Productize

Automate and integrate

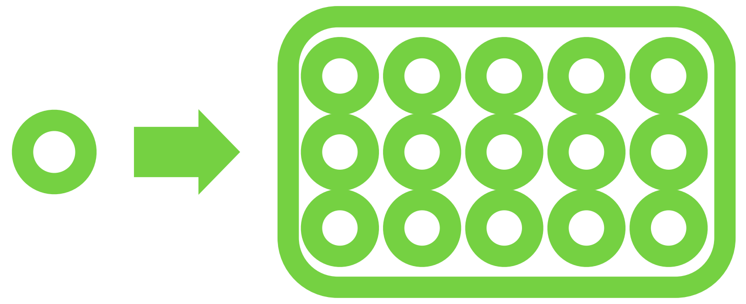
Publish

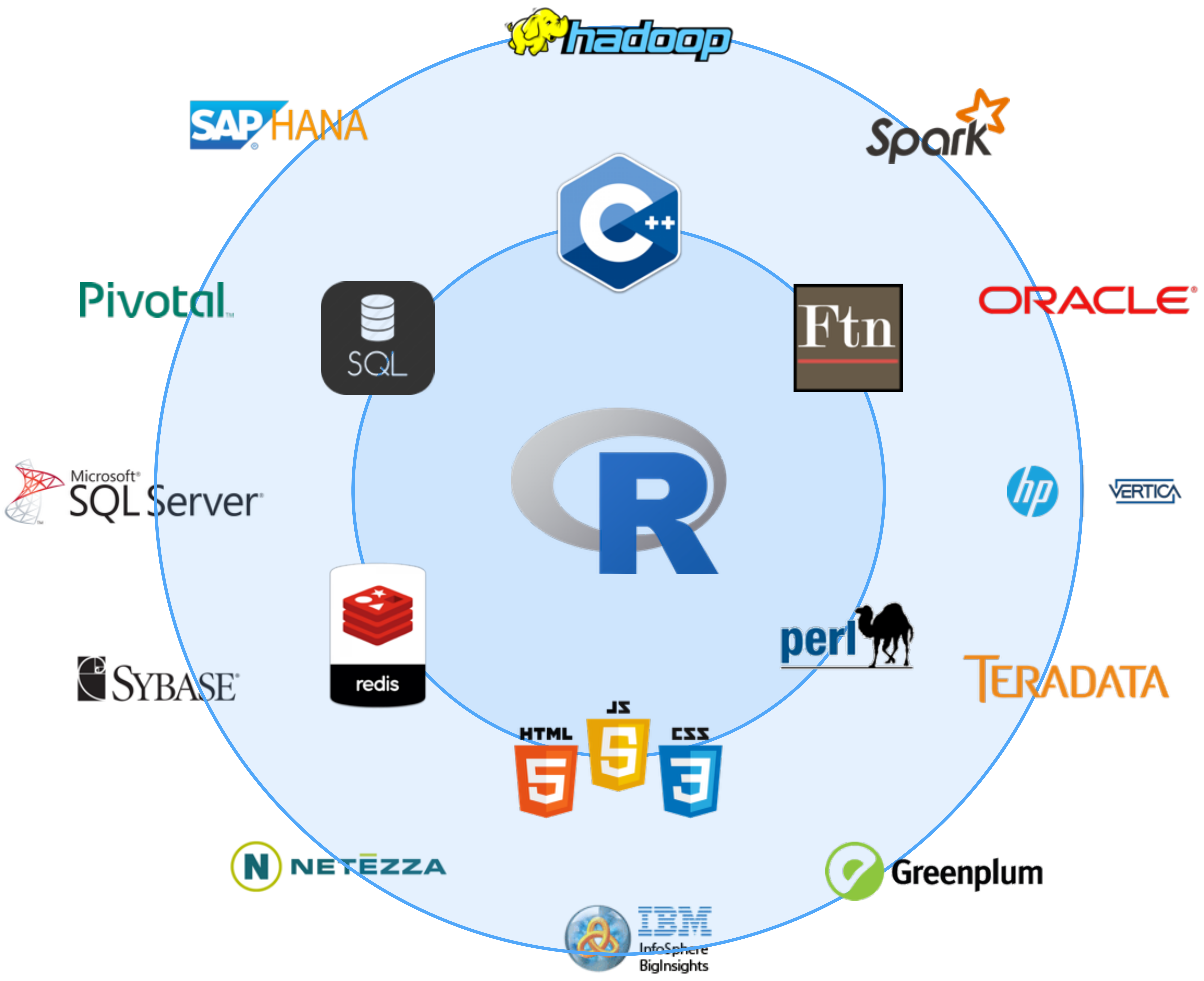
Socialize

Class 1



Class 2
Class 3

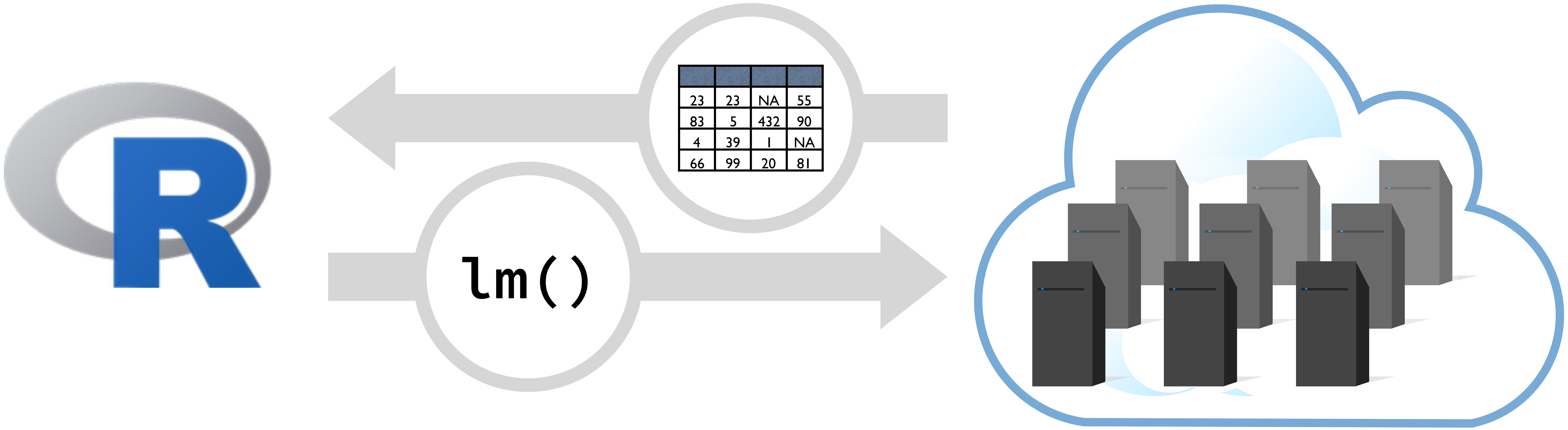



hadoop
SAP HANA
Spark
C++
Pivotal
SQL
Ftn
ORACLE
Microsoft SQL Server
R
hp
VERTICA
SYBASE
redis
perl
TERADATA
HTML
JS
CSS
NETEZZA
Greenplum
IBM
InfoSphere
BigInsights

General Strategy

Store big data in a data warehouse

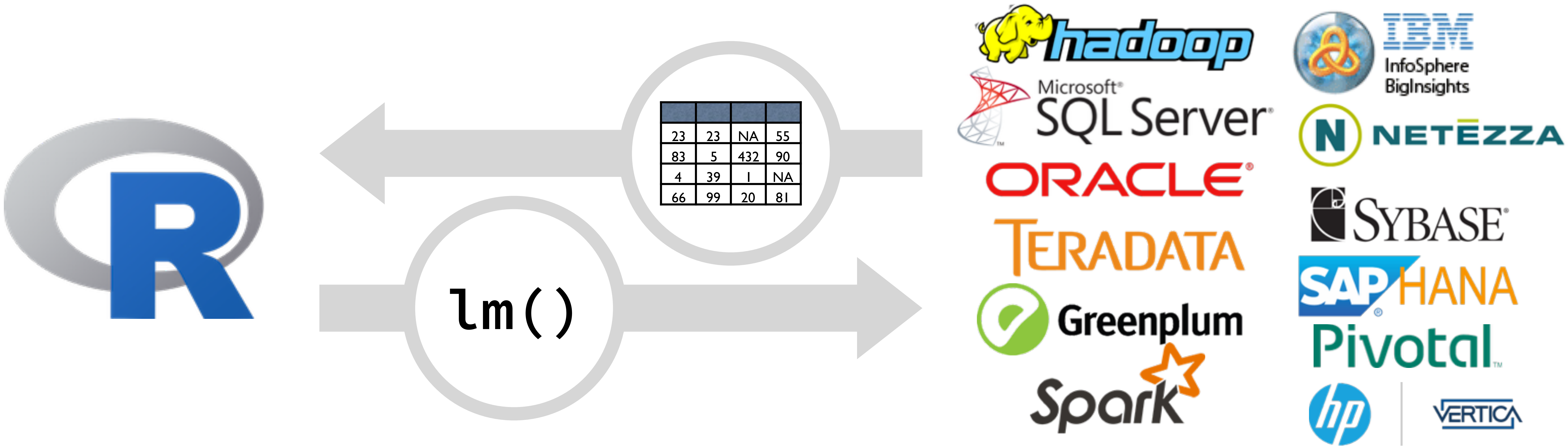
1. Pass subsets of data from warehouse to R
2. Transform R code, pass to warehouse.



General Strategy

Store big data in a data warehouse

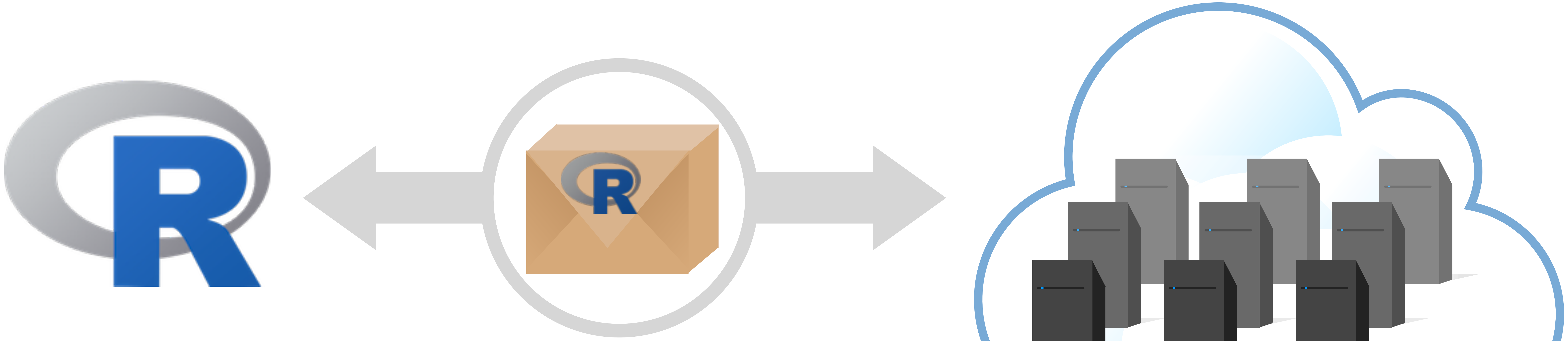
1. Pass subsets of data from warehouse to R
2. Transform R code, pass to warehouse.



General Strategy

Store big data in a data warehouse

1. Pass subsets of data from warehouse to R
2. Transform R code, pass to warehouse.



`dplyr, DBI, RHadoop, SparkR, RODBC, …`

dplyr/ Redshift

Airlines Data Set

Arrival and departure details for all commercial flights in US between October 1987 and April 2008.

120,000,000 records. **12 GB**

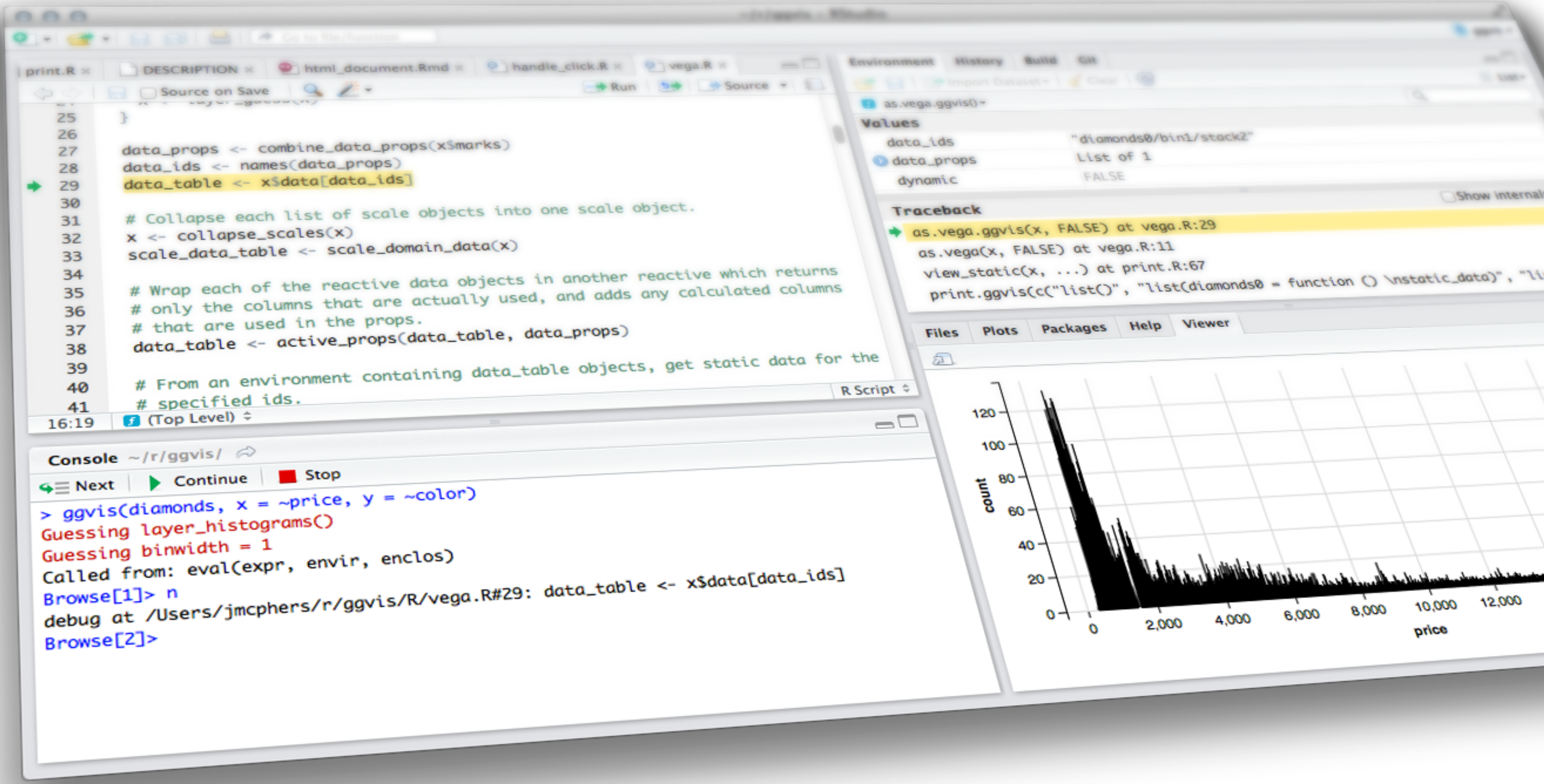
stat-computing.org/dataexpo/2009/

AirTran · Endeavor Air · Hawaiian Airlines · Southwest · Alaska Airlines · envoy · jetBlue Airways · United · American Airlines · ExpressJet · Mesa Airlines · US Airways · Delta · Frontier Airlines · SkyWest Airlines · Virgin atlantic

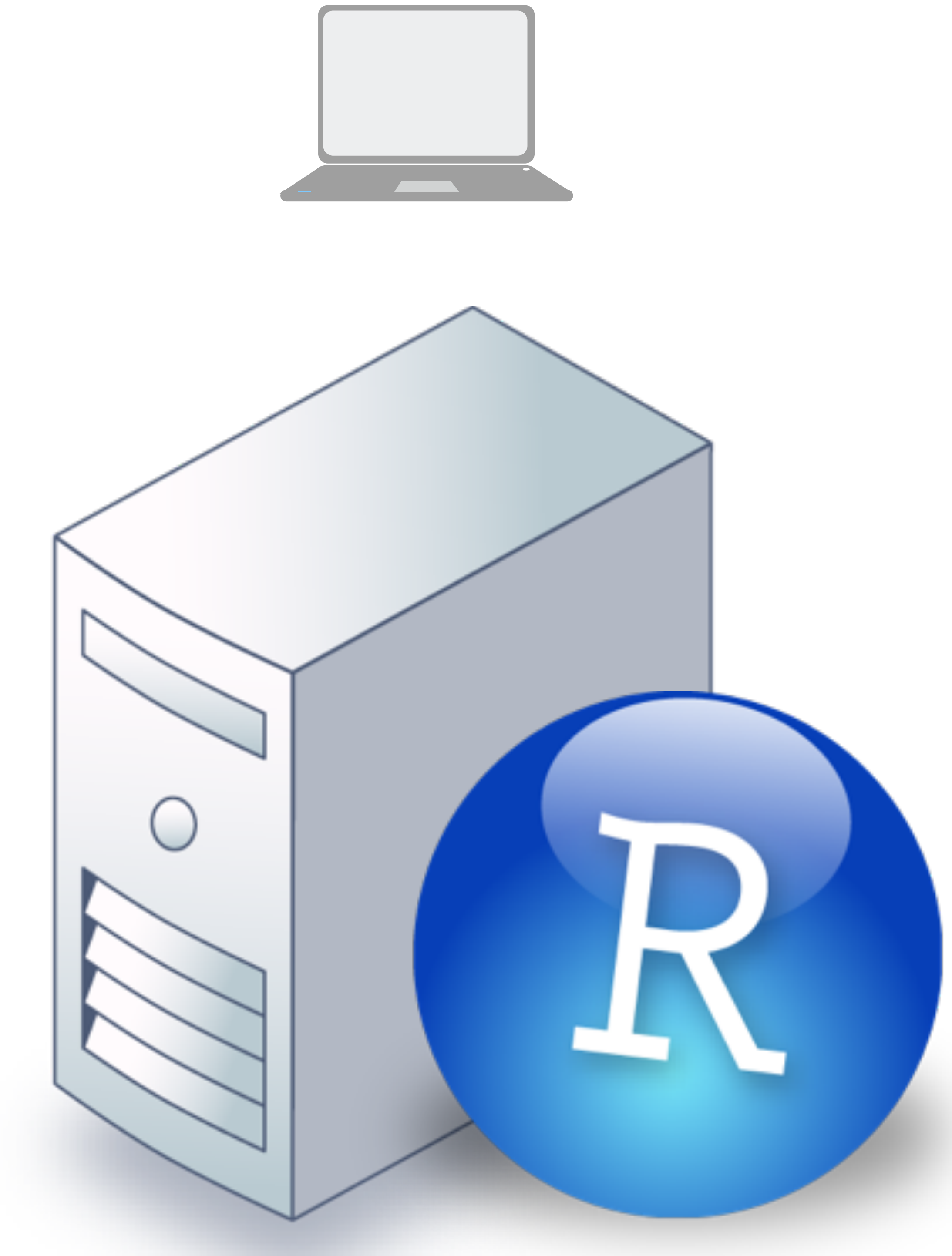
Data does not fit in memory

Example Task

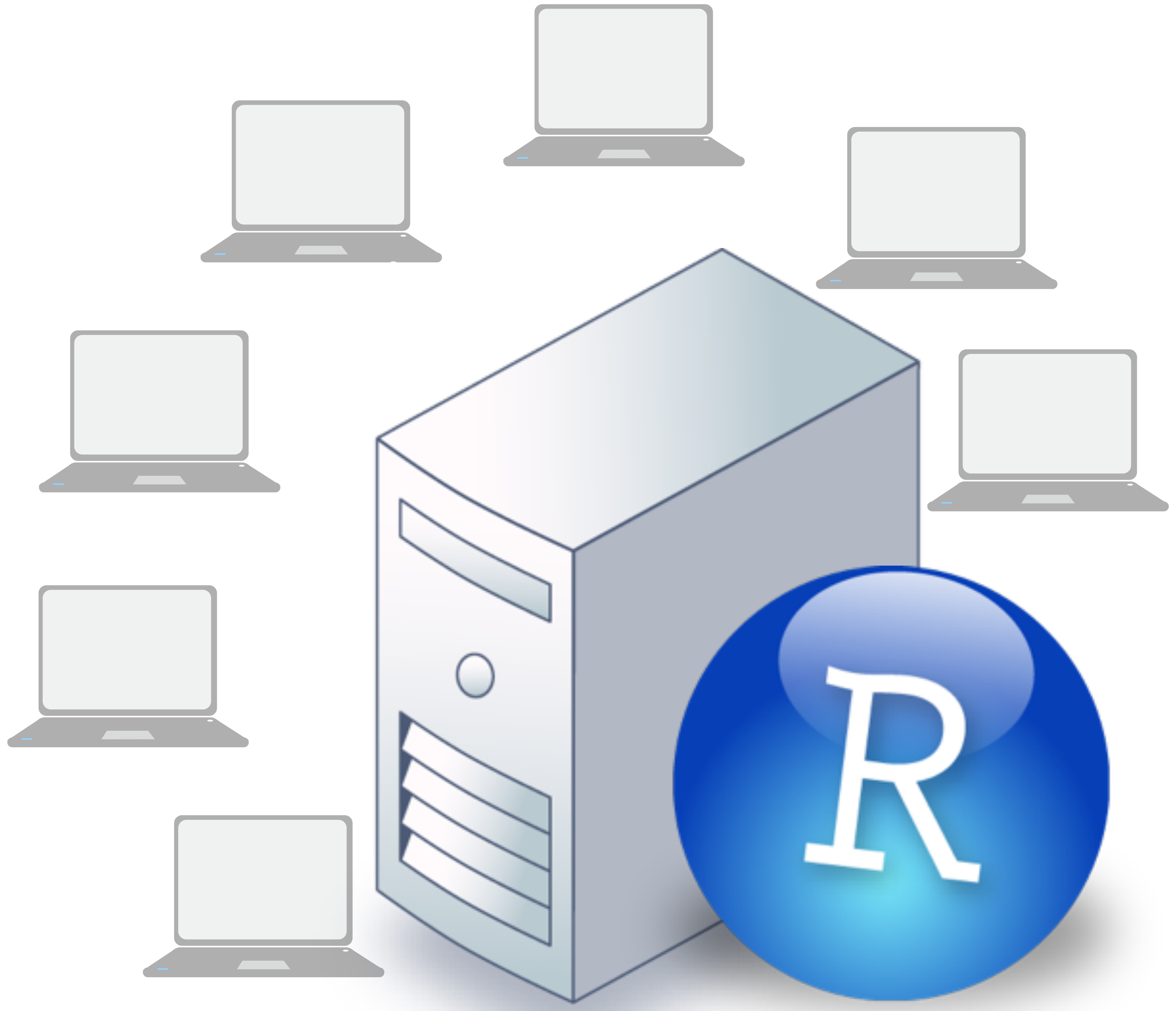
1. Collect random sample of training data
2. Fit a model to the sample (in R)
3. Score against test data (in DB)


print.R
DESCRIPTION
html_document.Rmd
handle_click.R
vega.R
Source on Save
Run
Source
data_props <- combine_data_props(x$marks)
data_ids <- names(data_props)
data_table <- x$data[data_ids]
# Collapse each list of scale objects into one scale object.
x <- collapse_scales(x)
scale_data_table <- scale_domain_data(x)
# Wrap each of the reactive data objects in another reactive which returns
# only the columns that are actually used, and adds any calculated columns
# that are used in the props.
data_table <- active_props(data_table, data_props)
# From an environment containing data_table objects, get static data for the
# specified ids.
16:19
(Top Level)
R Script
Console ~/r/ggvis/
Next
Continue
Stop
> ggvis(diamonds, x = ~price, y = ~color)
Guessing layer_histograms()
Guessing binwidth = 1
Called from: eval(expr, envir, enclos)
Browse[1]> n
debug at /Users/jmcphers/r/ggvis/R/vega.R#29: data_table <- x$data[data_ids]
Browse[2]>
Environment
History
Build
Git
Values
data_ids
"diamonds0/bin1/stack2"
data_props
List of 1
dynamic
FALSE
Traceback
as.vega.ggvis(x, FALSE) at vega.R:29
as.vega(x, FALSE) at vega.R:11
view_static(x, ...) at print.R:67
Files
Plots
Packages
Help
Viewer
count
price

RStudio
Desktop IDE



RStudio Server
Open Source



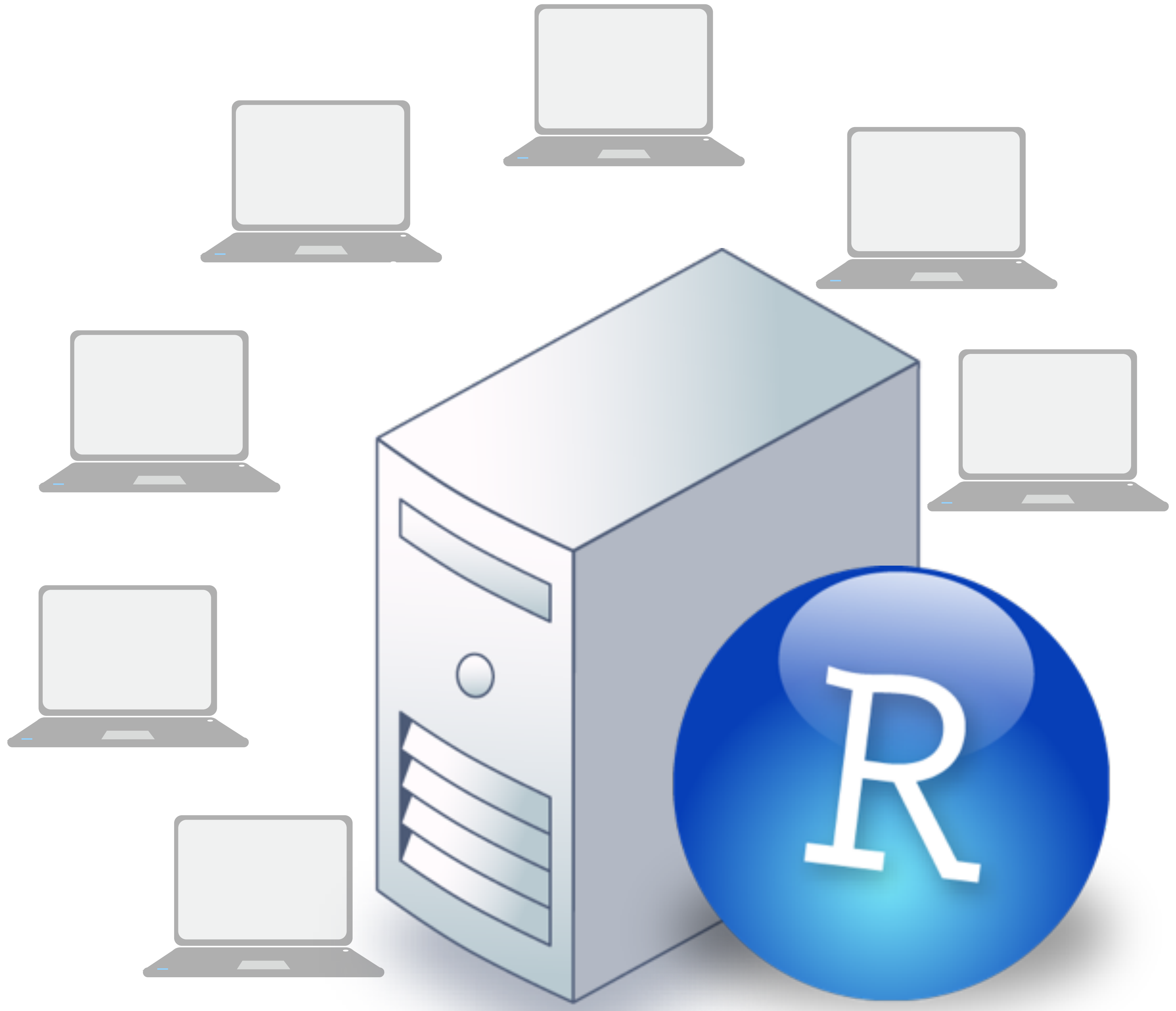
RStudio Server
Pro

- Security
- Load balancing
- Administrative tools
- Resource management
- Metrics and monitoring

- Multiple sessions
- Collaborative editing
- Easy R versioning
- Audit history

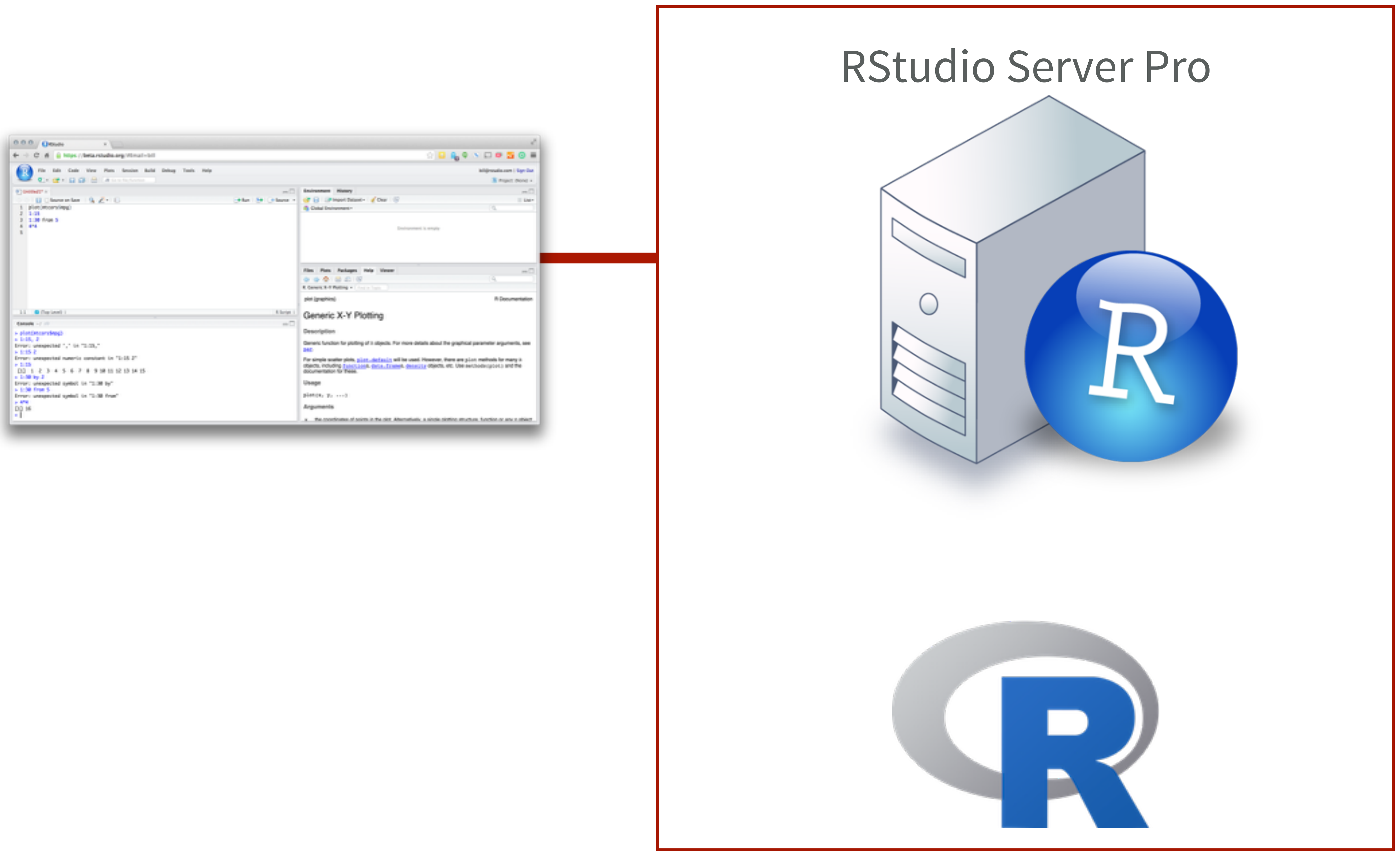
RStudio Desktop IDE

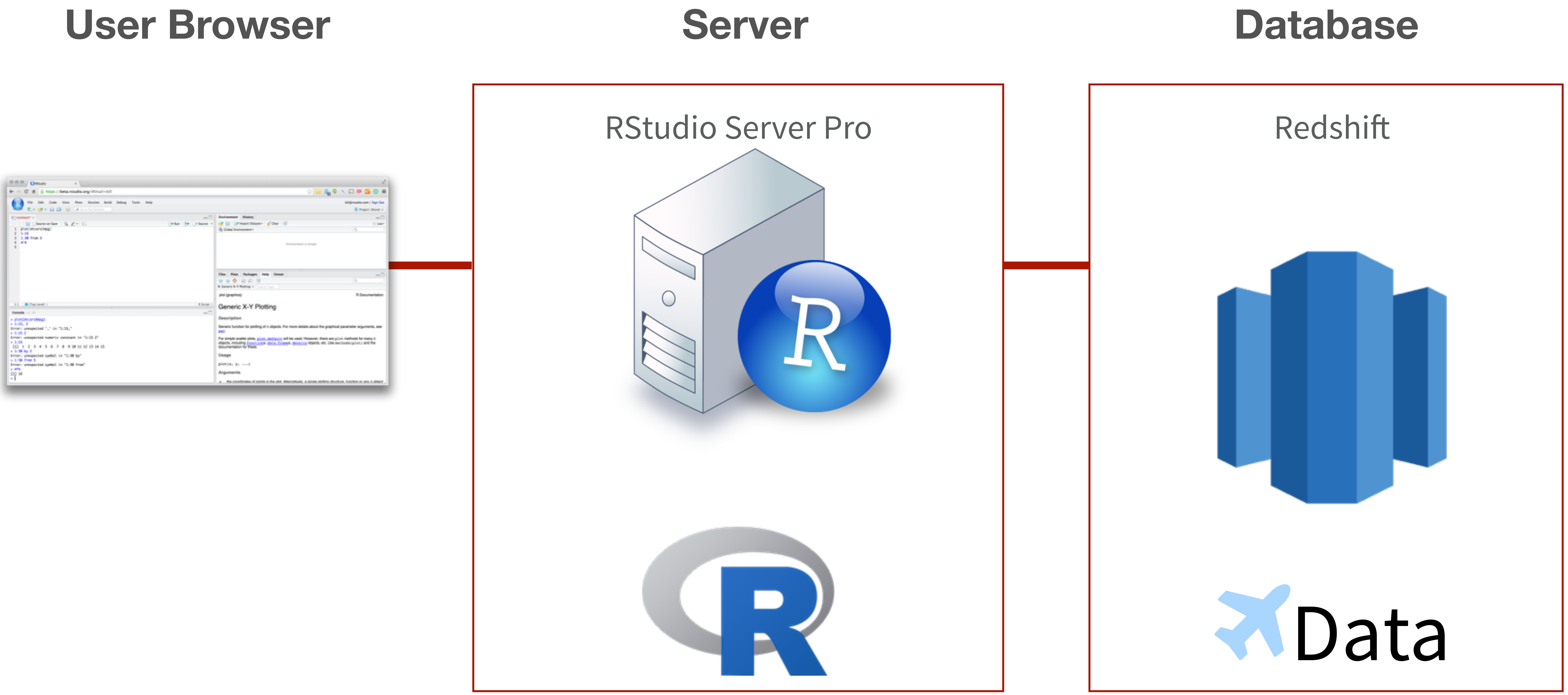
RStudio Server Open Source



RStudio Server Pro

User Browser **Server**

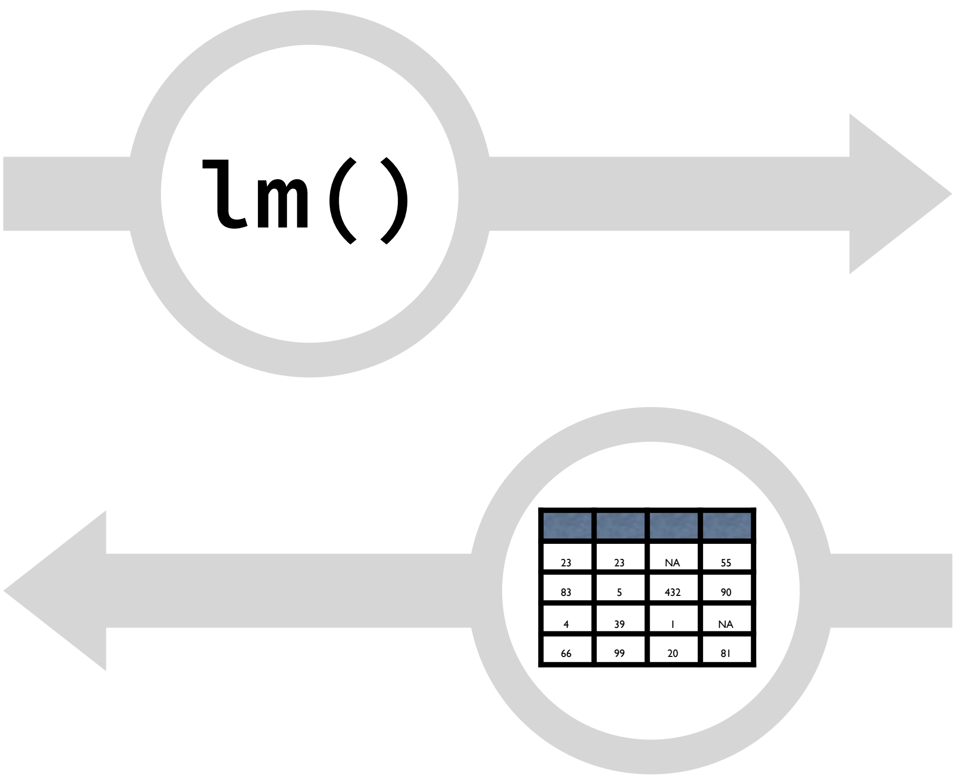



User Browser
Server
Database
RStudio Server Pro
Redshift
Data

dplyr

Package that provides data manipulation syntax for R. Comes with **built-in SQL backend:**

1. **Connects** to DBMS's
2. **Transforms R code** to SQL, sends to DBMS to run in DBMS
3. **Collect results** into R



Connections

Make with src_ function:
src_postgres, src_sqlite
src_mysql, src_bigquery

```
db <- src_postgres(
        dbname = 'DATABASE_NAME',
        host = 'HOST',
        port = 5432,
        user = 'USERNAME',
        password = 'PASSWORD')
```

Save to use

Driver specific arguments.

For PostgreSQL: dbname, host, port, user, password

```r
library(dplyr)

# Create connection to the database
air <- src_postgres(
  dbname = 'airontime',
  host = 'sol-eng.cjku7otn8uia.us-west-2.redshift.amazonaws.com',
  port = '5439',
  user = 'redshift_user',
  password = 'ABCd4321')

# List table names
src_tbls(air)

# Create a table reference with tbl
flights <- tbl(air, "flights")
carriers <- tbl(air, "carriers")
```

RStudio

Dplyr driver functions

Package	DBMS
src_sqlite()	SQLite
src_mysql()	MySQL, MariaDB
src_postgres()	PostgreSQL
library(bigrquery) src_bigquery()	Google BigQuery

https://cran.r-project.org/web/packages/dplyr/vignettes/databases.html

Adding a new SQL backend

Open guide with

```
vignette("new-sql-backend", package = "dplyr")
```

Adding a new SQL backend

2015-06-15

This document describes how to describe a new SQL backend to dplyr. It's a work in progress, but will hopefully get started. If you're familiar with how your database works, and at least one other database that dplyr already supports, this should be reasonably simple, and I'm happy to help if you get stuck. It's also possible that a new database backend may need new methods - I'm also happy to add those as needed. If you need more help, please email the mailing list and I'll help you get

```r
# Manipulate the reference as if it were the actual table
clean <- flights %>%
  filter(!is.na(arrdelay), !is.na(depdelay)) %>%
  filter(depdelay > 15, depdelay < 240) %>%
  filter(year >= 2002 & year <= 2007) %>%
  select(year, arrdelay, depdelay, distance, uniquecarrier)
```

The pipe operator %>%

```
dd <- 1:1000
mean(dd, na.rm = TRUE)
dd %>% mean(na.rm = TRUE)
```

These do the same thing
Try it!

%>%

```
dd                 mean( _____, na.rm = TRUE)
```

show_query

To see the SQL that dplyr will run.

```
show_query(clean)
```

show_query

tbl made from table reference and dplyr functions

```
show_query(clean)
##  <SQL>
##  SELECT "year" AS "year",
##    "arrdelay" AS "arrdelay",
##    "depdelay" AS "depdelay",
##    "distance" AS "distance",
##    "uniquecarrier" AS "uniquecarrier"
##  FROM "flights"
##  WHERE NOT("arrdelay"IS NULL) AND NOT("depdelay"IS NULL)
##    AND "depdelay" > 15.0 AND "depdelay" < 240.0
##    AND "year" >= 2002.0 AND "year" <= 2007.0
```

* dplyr can convert all of the following to SQL
** other functions will be passed as is into SQL

dplyr functions	`arrange, filter, group_by. mutate, select, summarize, %>%, left_join, etc.`
Operators	`+, -, *, /, %%, ^`
Math functions	`abs, acos, cosh, sin, asinh, atan, atan2, atanh, ceiling, cos, cosh, cot, coth, exp, floor, log, log10, round, sign, sin, sinh, sqrt, tan, tanh`
Comparisons	`<, <=, !=, >=, >, ==, %in%`
Booleans	`&, &&, |, ||, !, xor`
Aggregations	`mean, sum, min, max, sd, var`

Lazy Execution 1

```
q1 <- filter(flights, year < 2007)
q2 <- filter(q1, depdelay > 15)
q3 <- filter(q2, depdelay < 240)
q4 <- select(q3, arrdelay, depdelay, yea
q4
```

When should dplyr query the database?

Lazy Execution 1

```
q1 <- filter(flights, year < 2007)
q2 <- filter(q1, depdelay > 15)
q3 <- filter(q2, depdelay < 240)
q4 <- select(q3, arrdelay, depdelay, year)
q4
```

dplyr will not retrieve data until last possible moment. It combines all necessary work into a single query.

```
show_query(q4)
## <SQL>
## SELECT "arrdelay" AS "arrdelay",
     "depdelay" AS "depdelay",
     "year" AS "year"
## FROM "flights"
## WHERE "year" > 2007.0
     AND "depdelay" > 15.0
     AND "depdelay" < 240.0
```

Lazy Execution 2

dplyr will only retrieve the **first 10 rows** of a query when you look at the output.

```
clean
## Source: postgres 8.0.2 […]
## From: flights [6,517,621 x 4]
## Filter: !is.na(arrdelay), !is.na(depdelay), …

##      year arrdelay depdelay uniquecarrier
##     (int)    (int)    (int)         (chr)
## 1    2007       42       40            9E
## 2    2007       90       94            9E
## 3    2007       19       20            9E
## 4    2007      184      167            9E
## 5    2007       21       30            9E
## 6    2007      178      179            9E
## 7    2007       56       59            9E
## 8    2007       21       21            9E
## 9    2007       50       57            9E
## 10   2007       56       23            9E
## ..    ...      ...      ...           ...
```

collect()

Forces dplyr to retrieve all results into R.

```
q5 <- flights %>%
  filter(year > 2007, depdelay > 15) %>%
  filter(depdelay == 240) %>%
  collect()
```

collect() returns entire result as a tbl_df

collect()

query to run

collapse()

Forces execution in DBMS

```
q6 <- flights %>%
    mutate(adjdelay = depdelay - 15) %>%
    collapse() %>%
    filter(adjdelay > 0)
```

collapse() turns the preceding queries into a table expression

remaining queries are run against the table described in the collapsed expression

```r
# Extract random 1% sample of training data
random <- clean %>%
  mutate(x = random()) %>%
  collapse() %>%
  filter(x <= 0.01) %>%
  select(-x) %>%
  collect()
```

Fit a model (in R)

Do some carriers make up lost time better than others?

$$gain = depdelay - arrdelay$$

```r
# make gain
random$gain <- random$depdelay - random$arrdelay

# build model
mod <- lm(gain ~ depdelay + distance + uniquecarrier,
          data = random)

# carrier coefficients table
coef(mod)
```

```r
# make coefficients lookup table
coefs <- dummy.coef(mod)

coefs_table <- data.frame(
  uniquecarrier = names(coefs$uniquecarrier),
  carrier_score = coefs$uniquecarrier,
  int_score = coefs$`(Intercept)`,
  dist_score = coefs$distance,
  delay_score = coefs$depdelay,
  row.names = NULL,
  stringsAsFactors = FALSE
)
```

```
coefs_table
##    uniquecarrier carrier_score int_score  dist_score delay_score
## 1             9E     0.0000000 -1.540312 0.003083624 -0.01359926
## 2             AA    -1.7131012 -1.540312 0.003083624 -0.01359926
## 3             AQ     0.6153050 -1.540312 0.003083624 -0.01359926
## 4             AS     1.4143664 -1.540312 0.003083624 -0.01359926
## 5             B6    -1.9714287 -1.540312 0.003083624 -0.01359926
## 6             CO    -1.5865993 -1.540312 0.003083624 -0.01359926
## 7             DH     3.1367039 -1.540312 0.003083624 -0.01359926
## 8             DL    -2.6404154 -1.540312 0.003083624 -0.01359926
## 9             EV     2.3434536 -1.540312 0.003083624 -0.01359926
## 10            F9     0.5341914 -1.540312 0.003083624 -0.01359926
## 11            FL    -0.8888280 -1.540312 0.003083624 -0.01359926
## 12            HA     1.6712540 -1.540312 0.003083624 -0.01359926
## 13            HP     3.3742529 -1.540312 0.003083624 -0.01359926
## 14            MQ    -1.3632398 -1.540312 0.003083624 -0.01359926
## 15            NW    -2.0416490 -1.540312 0.003083624 -0.01359926
```

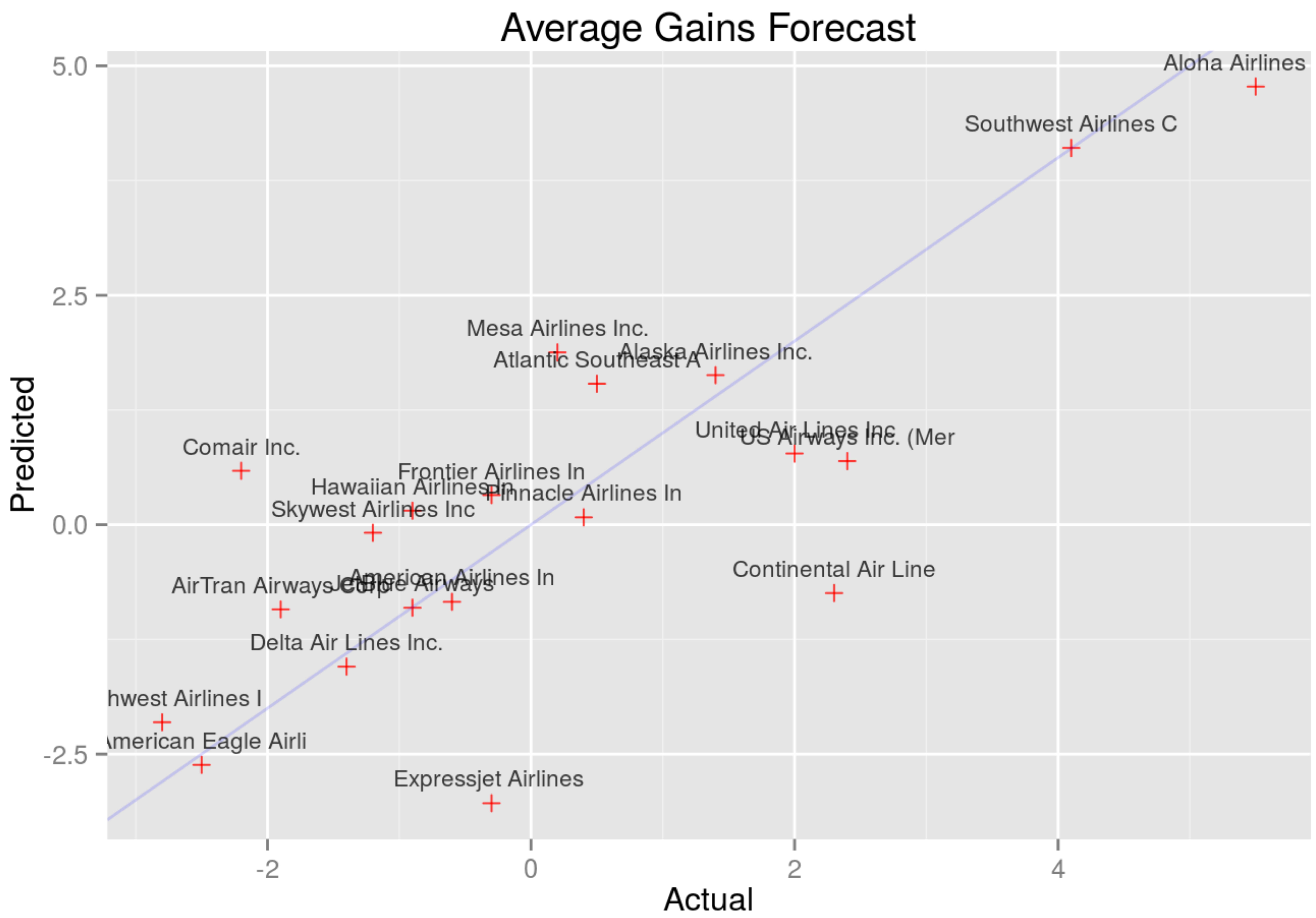
RStudio

Score data (in DBMS)

```
# Score test data
score <- flights %>%
  filter(year == 2008) %>%
  filter(!is.na(arrdelay) & !is.na(depdelay) & !is.na(distance)) %>%
  filter(depdelay > 15 & depdelay < 240) %>%
  filter(arrdelay > -60 & arrdelay < 360) %>%
  select(arrdelay, depdelay, distance, uniquecarrier) %>%
  left_join(carriers, by = c('uniquecarrier' = 'code')) %>%
  left_join(coefs_table, copy = TRUE) %>%
  mutate(gain = depdelay - arrdelay) %>%
  mutate(pred = int_score + carrier_score + dist_score * distance +
         delay_score * depdelay) %>%
  group_by(description) %>%
  summarize(gain = mean(1.0 * gain), pred = mean(pred))
scores <- collect(score)
```

Visualize scores

```
library(ggplot2)
ggplot(scores, aes(gain, pred)) +
  geom_point(alpha = 0.75, color = 'red', shape = 3) +
  geom_abline(intercept = 0, slope = 1, alpha = 0.15, color = 'blue') +
  geom_text(aes(label = substr(description, 1, 20)),
    size = 4, alpha = 0.75, vjust = -1) +
  labs(title='Average Gains Forecast', x = 'Actual', y = 'Predicted')
```



Average Gains Forecast
Predicted
Actual
5.0
2.5
0.0
-2.5
-2
0
2
4
Aloha Airlines
Southwest Airlines C
Mesa Airlines Inc.
Atlantic Southeast A
Alaska Airlines Inc.
United Air Lines Inc.
US Airways Inc. (Mer
Comair Inc.
Frontier Airlines In
Hawaiian Airlines In
Pinnacle Airlines In
Skywest Airlines Inc
Continental Air Line
American Airlines In
AirTran Airways Corp
JetBlue Airways
Delta Air Lines Inc.
hwest Airlines I
merican Eagle Airli
Expressjet Airlines

copy_to

```
copy_to(air, query5, name = "gains")
```

copy_to()

connection object

data frame to copy to data database

table name in database

`copy_to()` creates a table in the database from a local data frame.

Close Connection

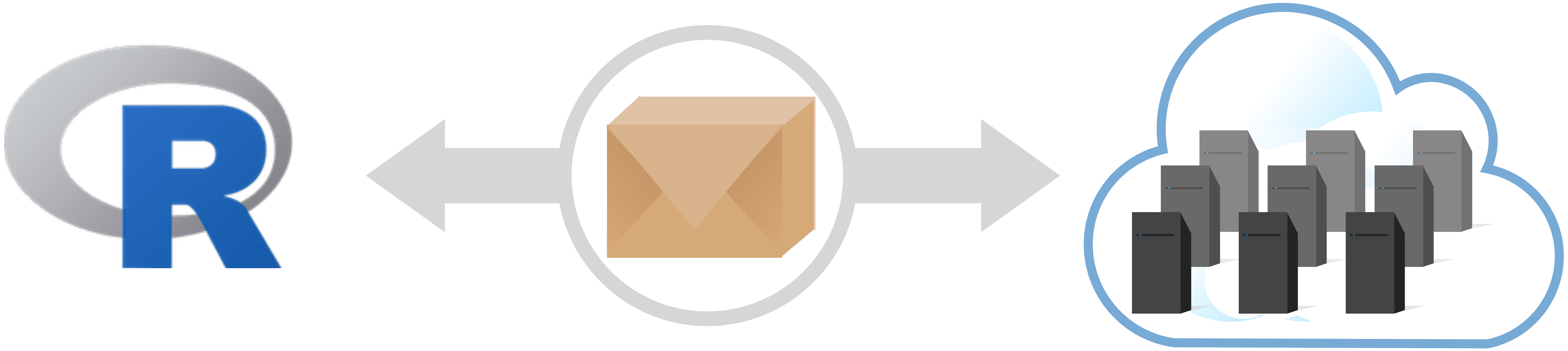
```
rm(air)
gc()
```

Remove connection object

Run garbage collector

dplyr automatically closes connections when you remove the connection object *and then run the garbage collector, `gc()`.*

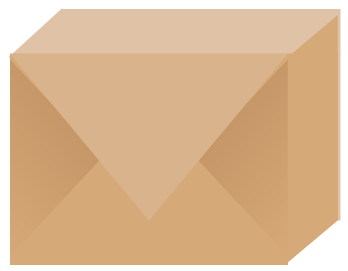
Alternative APIs



- RHadoop
- SparkR
- RevoScaleR
- PivotalR
- RODBC
- RJDBC, etc…

Recap: Access Big Data

Store data outside of memory in data warehouse



Use an R package as an API to the data warehouse. dplyr, DBI, sparkR, others.

```
db <-
```

Create and work with connection object

```
rm(db)
gc()
```

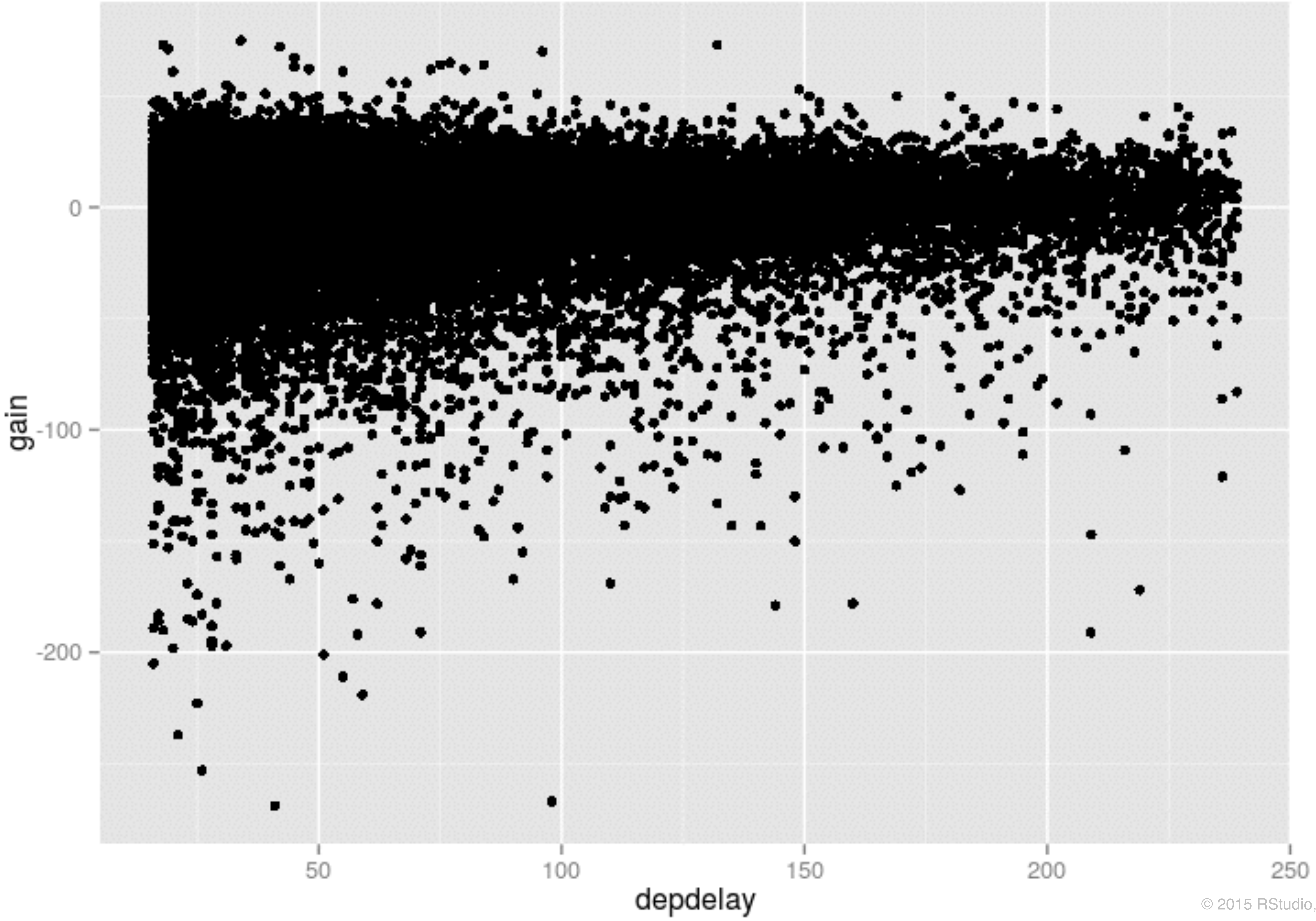
Close connection when finished

Applying big data

Big Data and Visualization

1. Plot **summaries**, not raw data

Big data ❤’s overplotting



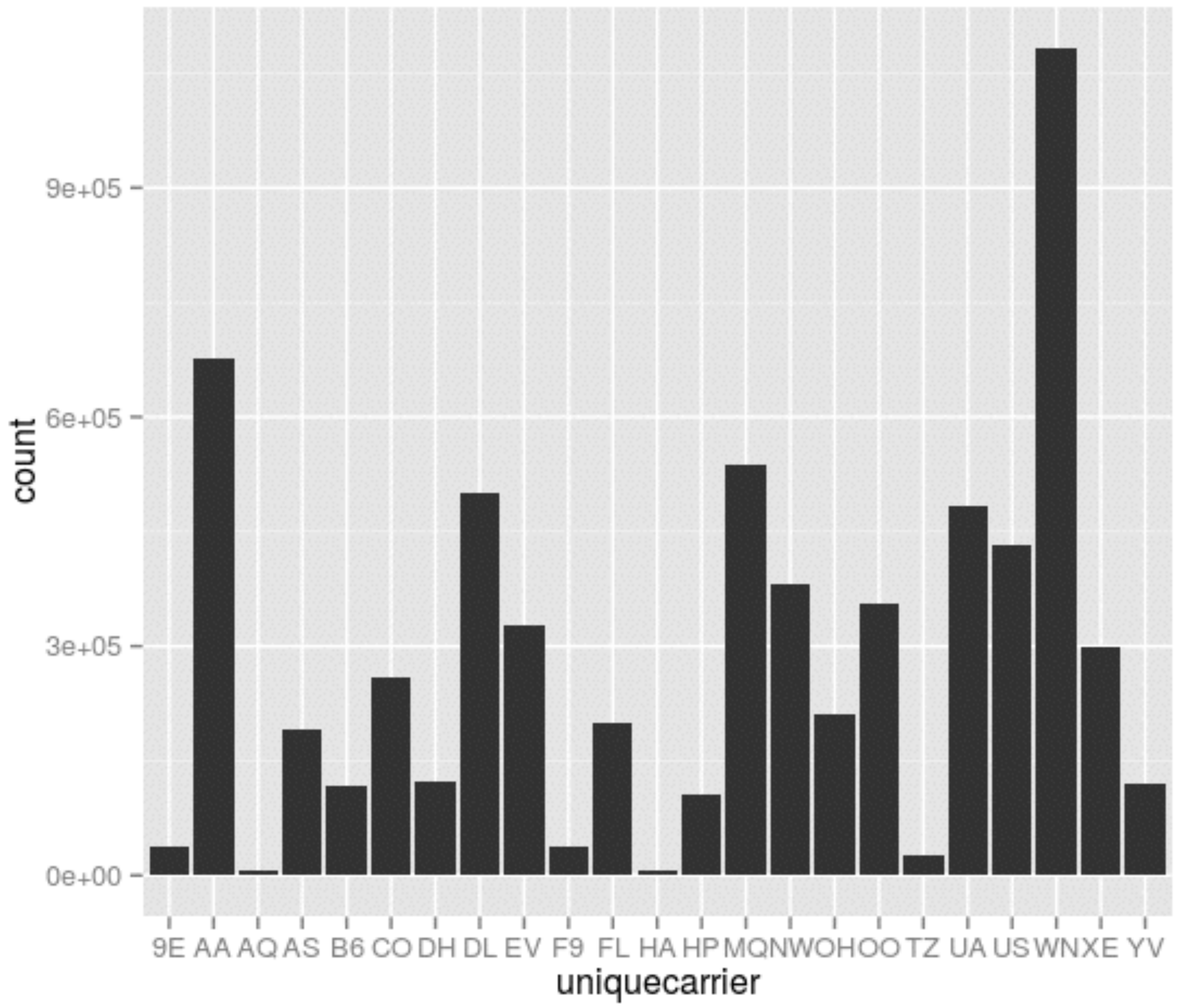
Big Data and Visualization

1. Plot **summaries**, not raw data
2. Calculate summaries **manually** in data store

```
cldata <- collect(clean)
ggplot(cldata) +
  geom_bar(aes(x = uniquecarrier))
```

year	arrdelay	depdelay	distance	unique carrier
2007	42	40	424	9E
2007	90	94	424	9E
2007	19	20	424	9E
2007	184	167	424	9E
2007	21	30	424	9E
2007	178	179	424	9E
2007	56	59	424	9E
2007	21	21	424	9E
2007	50	57	424	9E
…	…	…	…	…

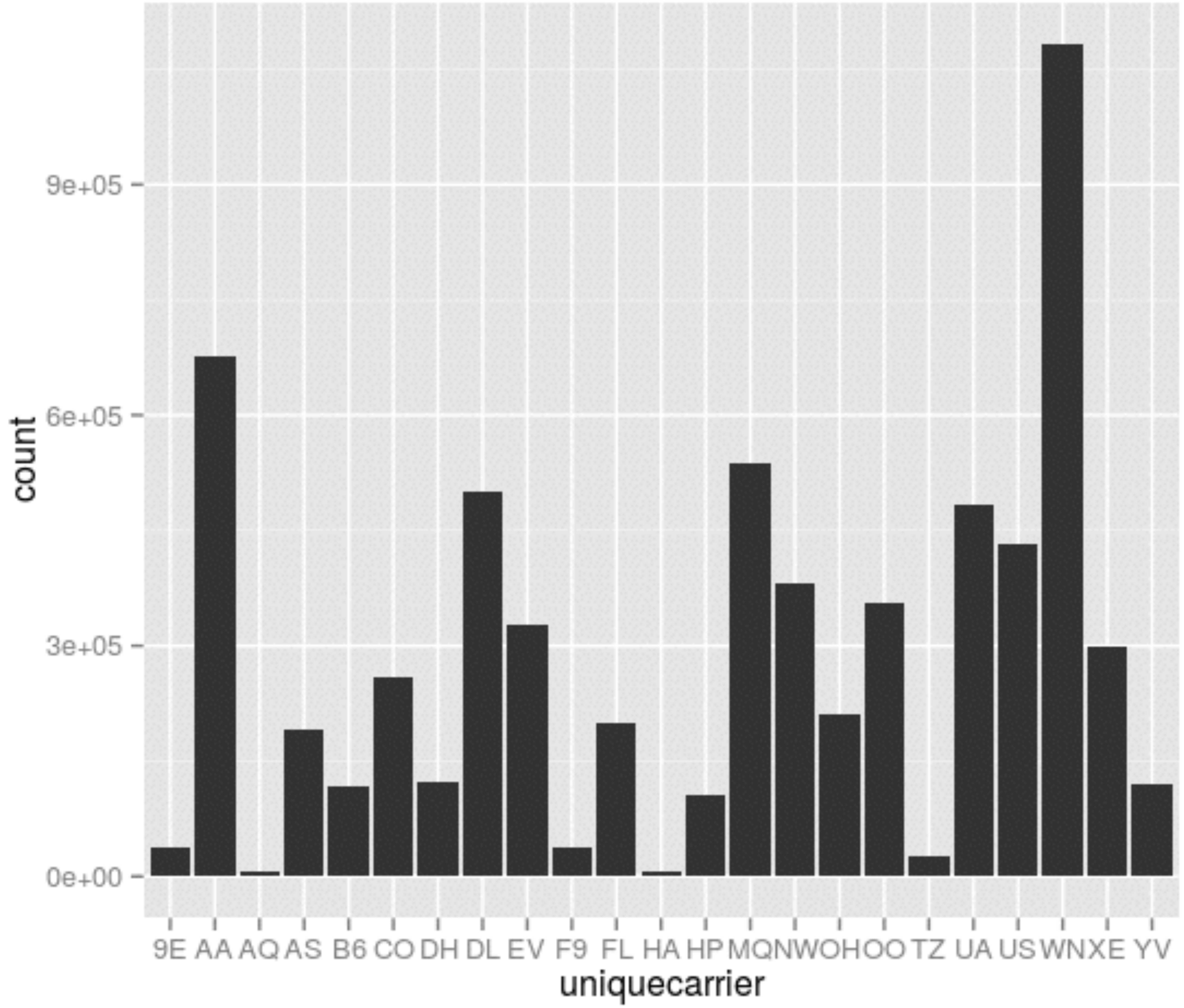
unique carrier	count
DH	123752
US	431913
AA	677471
F9	37710
HP	105926
AS	189748
AQ	5368
9E	38367
EV	326694
NW	381213



```
cldata <- collect(clean)
ggplot(cldata) +
  geom_bar(aes(x = uniquecarrier),
           stat = "bin")
```

year	arrdelay	depdelay	distance	unique carrier
2007	42	40	424	9E
2007	90	94	424	9E
2007	19	20	424	9E
2007	184	167	424	9E
2007	21	30	424	9E
2007	178	179	424	9E
2007	56	59	424	9E
2007	21	21	424	9E
2007	50	57	424	9E
…	…	…	…	…

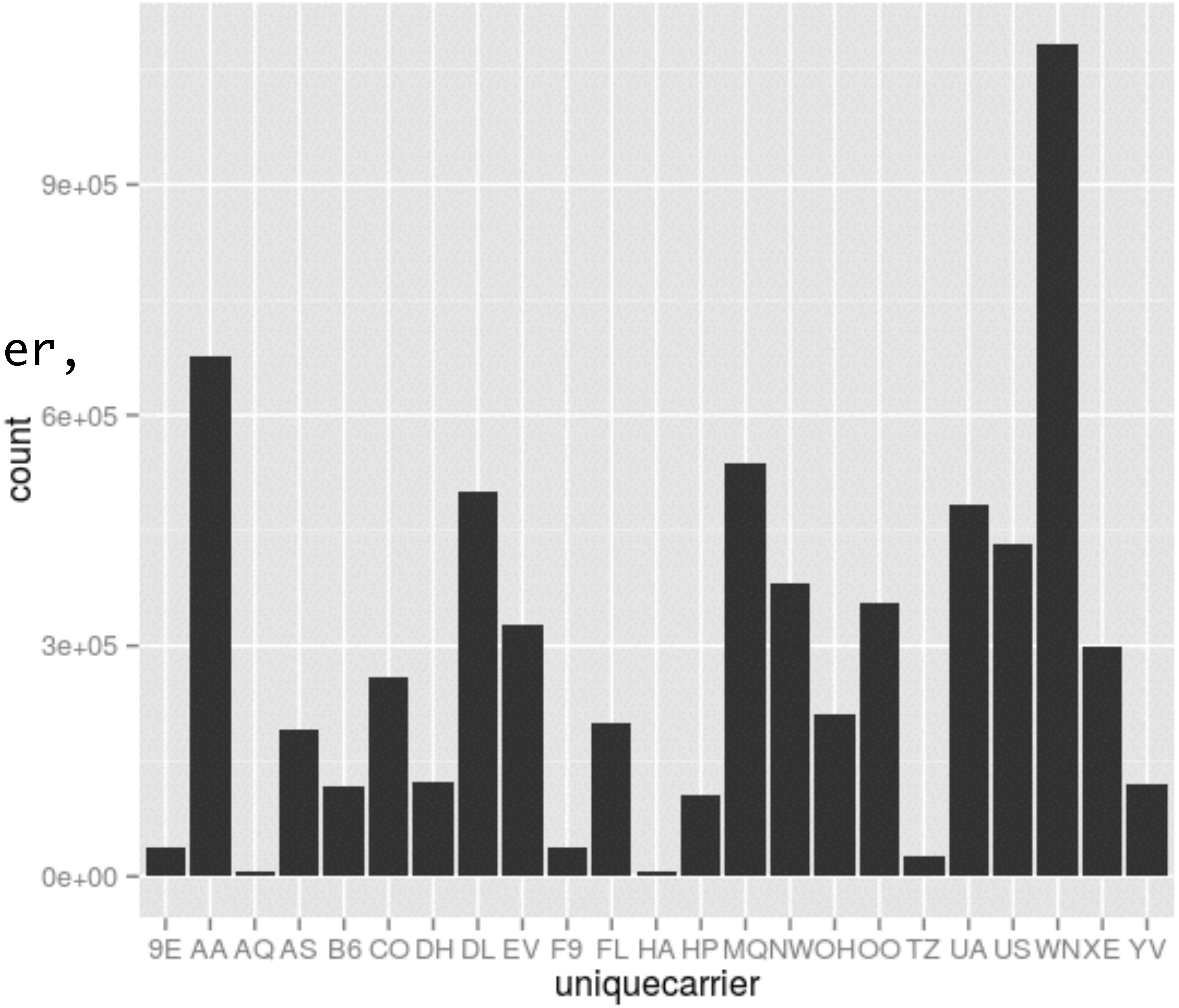
unique carrier	count
DH	123752
US	431913
AA	677471
F9	37710
HP	105926
AS	189748
AQ	5368
9E	38367
EV	326694
NW	381213



```
clsummary <- clean %>%
  group_by(uniquecarrier) %>%
  summarise(count = n()) %>%
  collect()
```

unique carrier	count
DH	123752
US	431913
AA	677471
F9	37710
HP	105926
AS	189748
AQ	5368
9E	38367
EV	326694
NW	381213

```
ggplot(clsummary) +
  geom_bar(aes(x = uniquecarrier,
               y = count),
           stat = "identity")
```



R Markdown
and big data

Big Data and R Markdown

1. **Cache** code chunks that manipulate big data.

cache

R Markdown will cache the result of the code chunk to reuse (and thus avoid computation) when **cache = TRUE**

````
Here's some code
that takes a "long"
time to run.
```{r cache=TRUE}
Sys.sleep(5)
rnorm(1)
```
````

Here's some code that takes a "long" time to run.

```
Sys.sleep(5)
rnorm(1)
```

```
## [1] 1.582407
```

Here's some code that takes a "long" time to run.

```
Sys.sleep(5)
rnorm(1)
```

```
## [1] 1.582407
```

cache

R Markdown will cache the result of the code chunk to reuse (and thus avoid computation) when **cache = TRUE**

```
```{r cache=TRUE}
d <- flights %>%
  select(…) %>%
  filter(…) %>%
  mutate(…) %>%
  collect()
```
```

Here's some code that takes a "long" time to run.

```
Sys.sleep(5)
rnorm(1)
```

```
## [1] 1.582407
```

Here's some code that takes a "long" time to run.

```
Sys.sleep(5)
rnorm(1)
```

```
## [1] 1.582407
```

Big Data and R Markdown

1. **Cache** code chunks that manipulate big data.
2. Set **engine** to use another language if sensible.

engine

To embed non R code, set the engine option to the language you want to embed.

```
Some python code,
```{r engine='python'}
x = 'hello, python world!'
print(x)
print(x.split(' '))
```
```

Some python code:

```
x = 'hello, python world!'
print(x)
print(x.split(' '))
```

```
## hello, python world!
## ['hello,', 'python', 'world!']
```

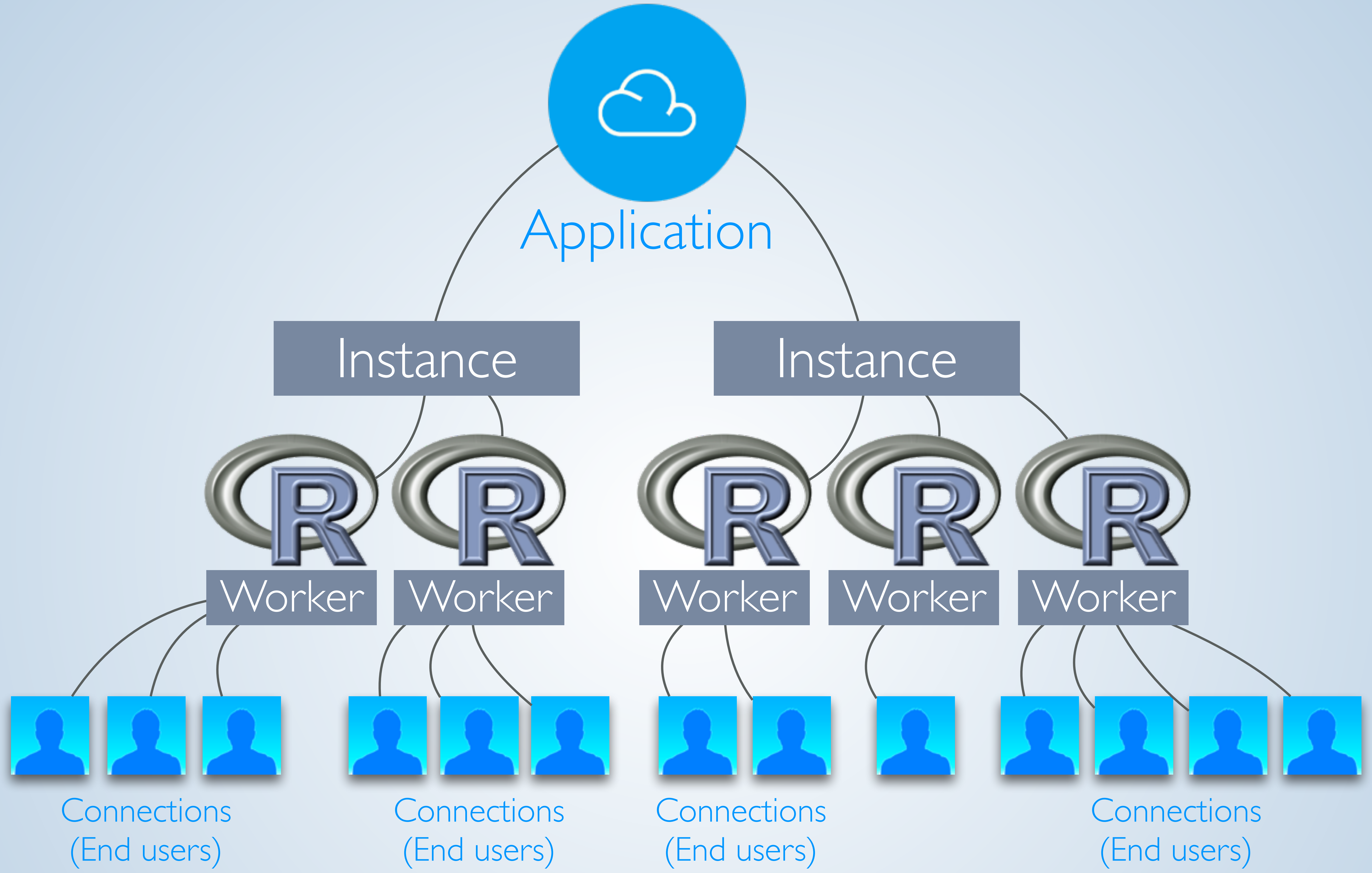
knitr comes with engines for the following languages, and can be extended to other languages

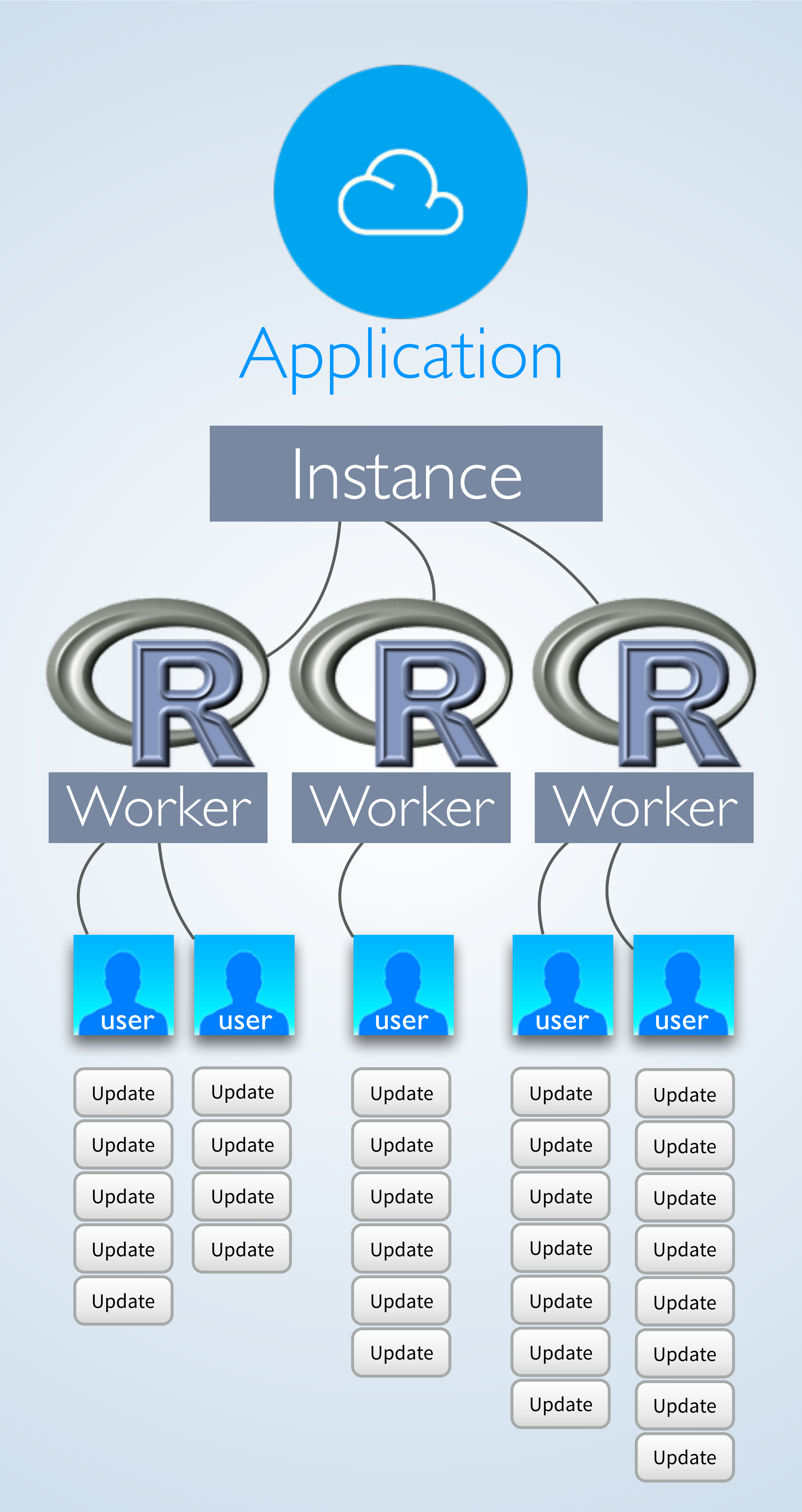
asis	**fortran**	perl	sed
asy	gawk	**psql**	sh
awk	groovy	python	stan
bash	haskell	**Rcpp**	stata
c	highlight	Rscript	tikz
cat	lein	ruby	zsh
coffee	**mysql**	sas	
dot	node	**scala**	

Shiny
and big data

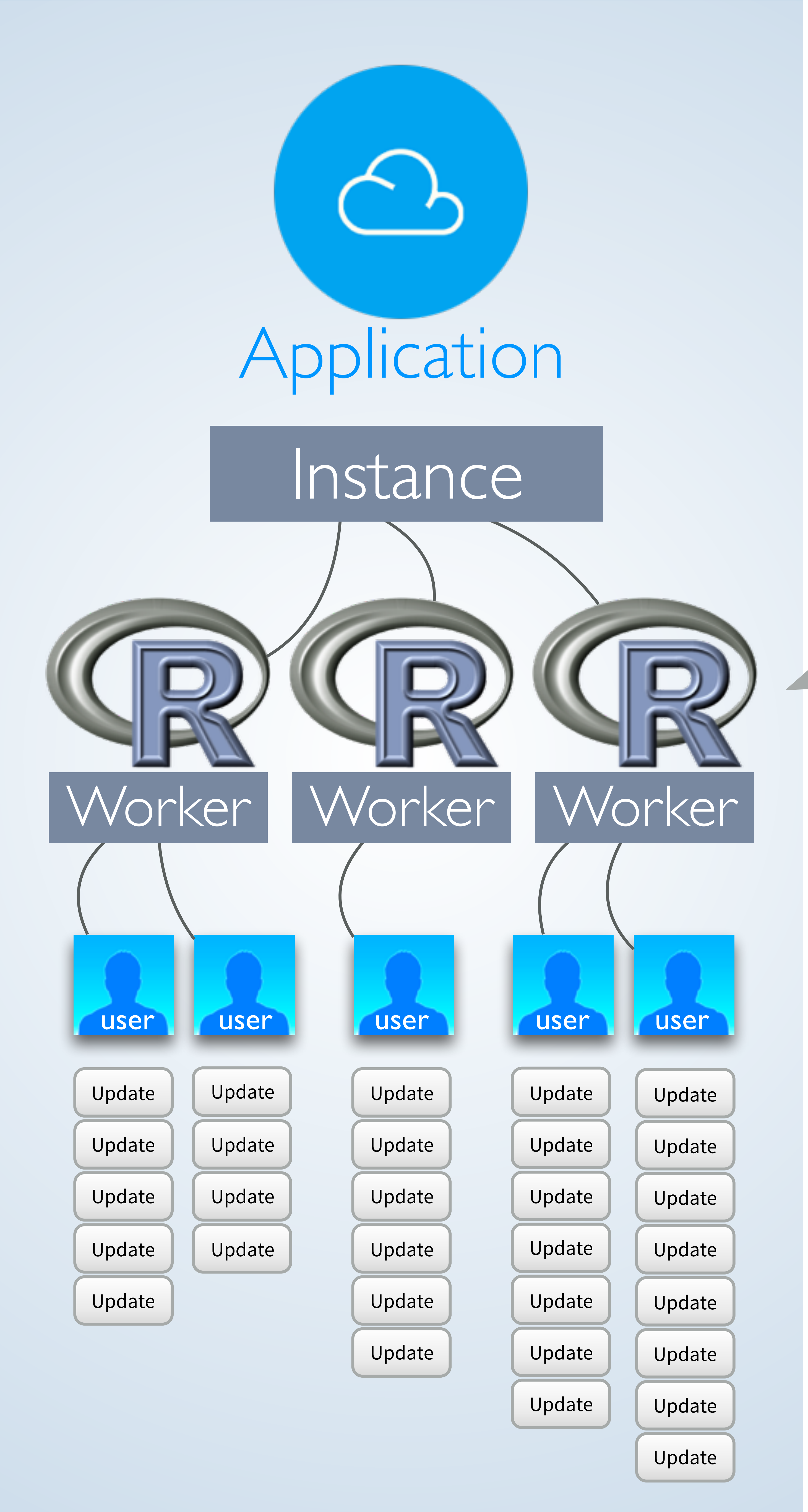
Big Data and Shiny

1. **Avoid** unnecessary repetitions


Application
Instance
Instance
Worker
Worker
Worker
Worker
Worker
Connections
(End users)
Connections
(End users)
Connections
(End users)
Connections
(End users)

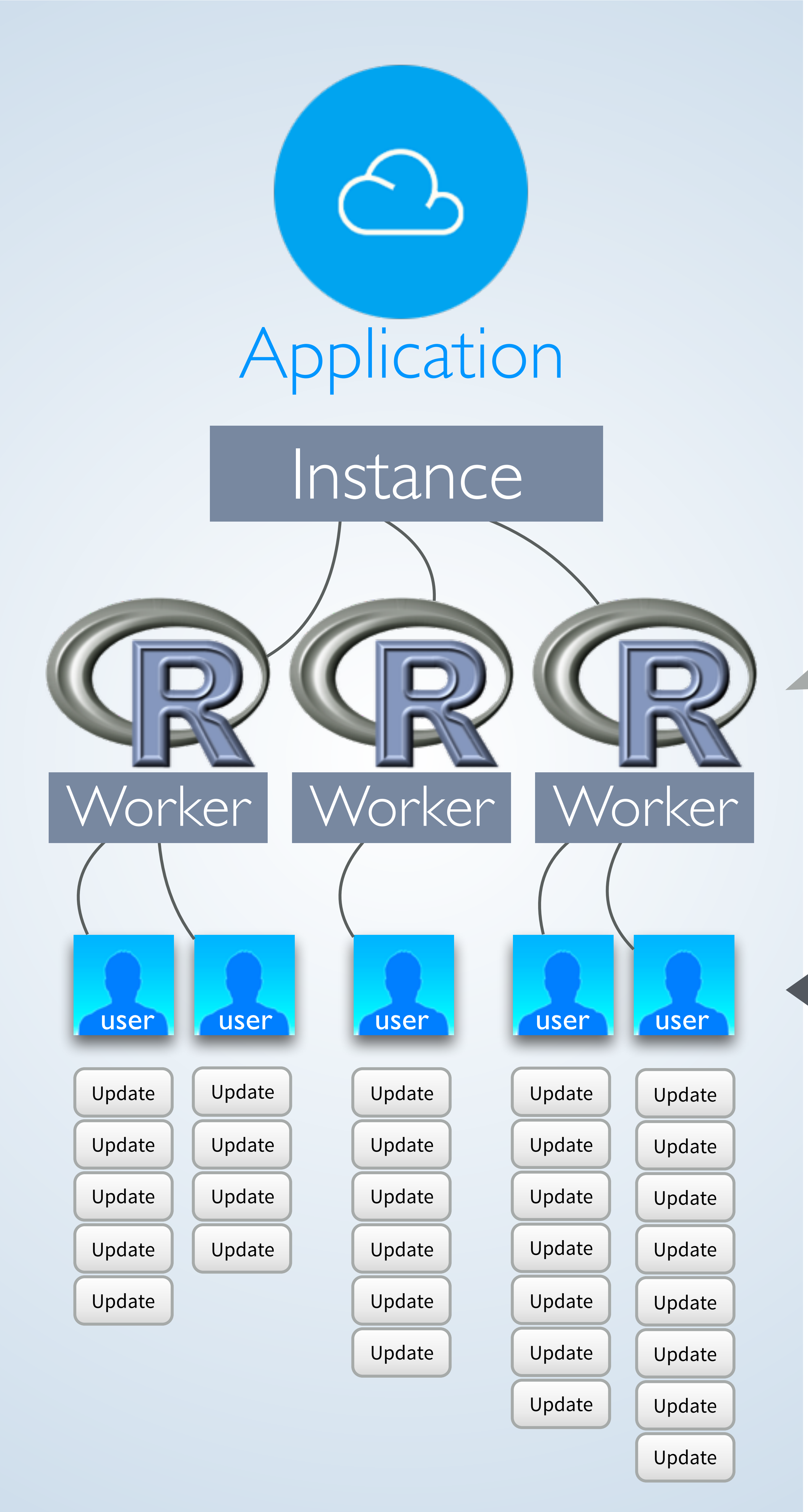



```r
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



Code outside the server function will be run once per R worker

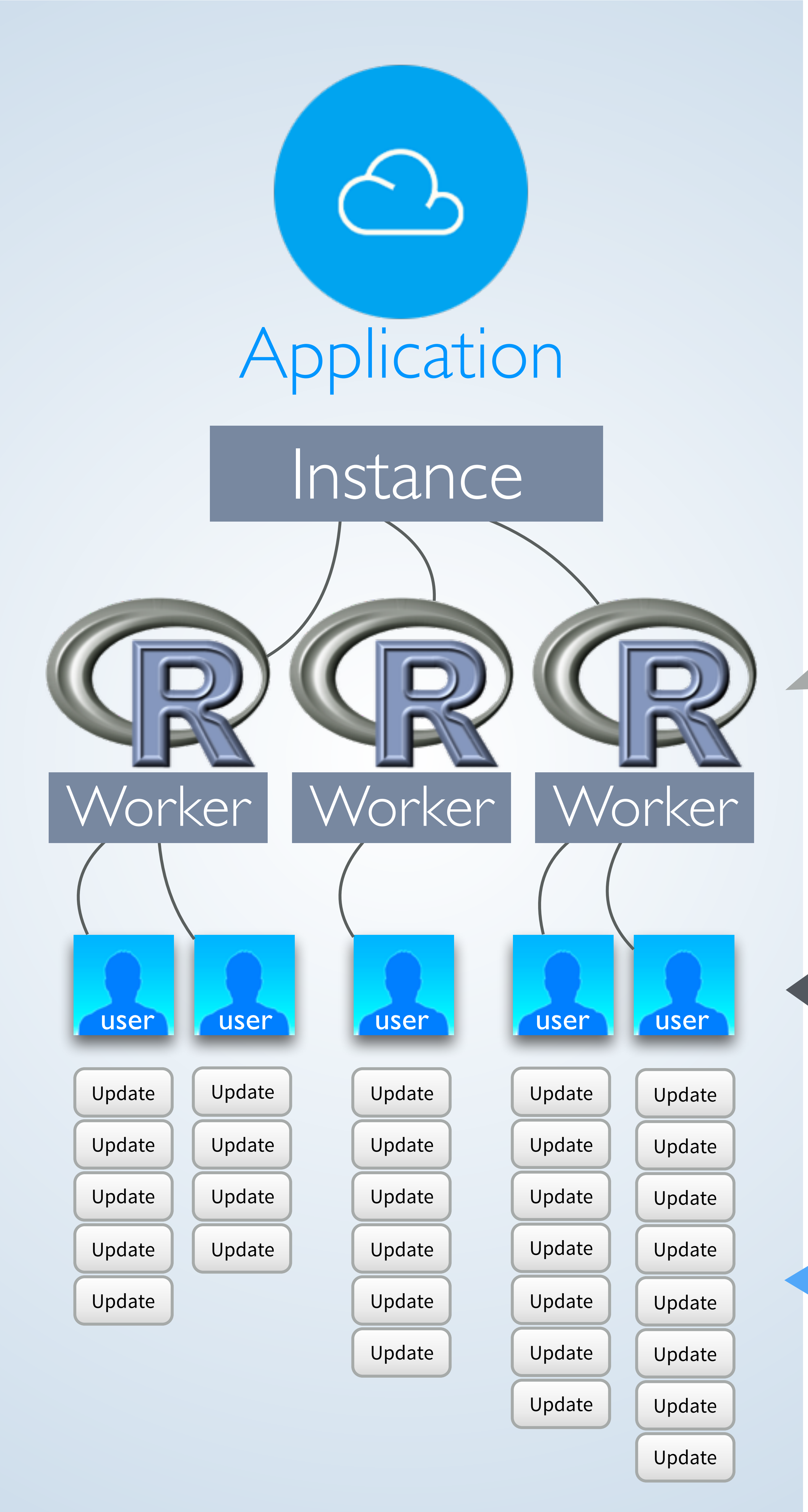
```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



Code outside the server function will be run once per R worker

Code inside the server function will be run once per connection

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```



Code outside the server function will be run once per R worker

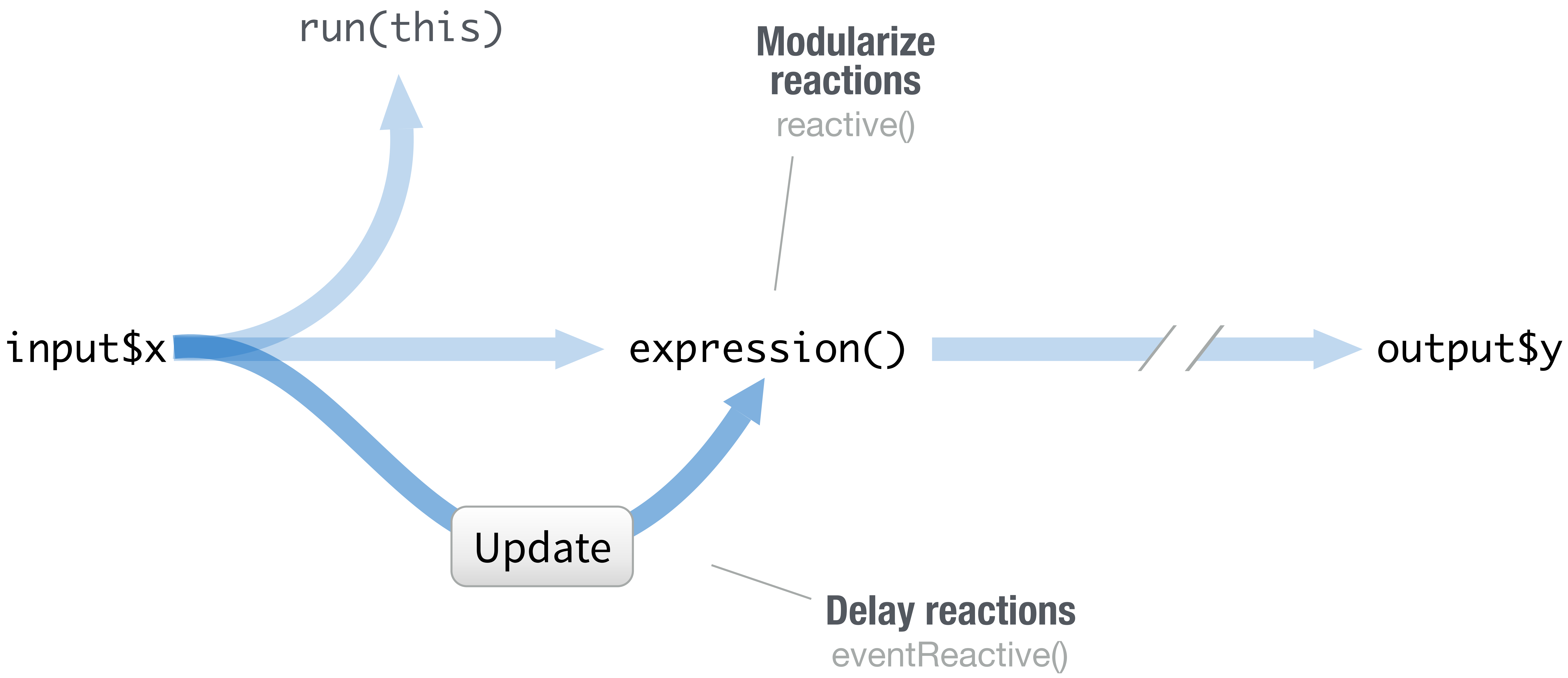
Code inside the server function will be run once per connection

Code inside of a reactive function will be run once per reaction

```
library(shiny)

ui <- fluidPage(
  sliderInput(inputId = "num",
    label = "Choose a number",
    value = 25, min = 1,
    max = 100),
  plotOutput("hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}

shinyApp(ui = ui, server = server)
```

Big Data and Shiny

1. **Avoid** unnecessary repetitions
2. **Cache** expensive operations with reactive expressions
3. **Delay** expensive operations, e.g. with an action button


run(this)
Modularize reactions
reactive()
input$x
expression()
output$y
Update
Delay reactions
eventReactive()

Thank You