

Strata+ Hadoop

WORLD

PRES ENTED BY

O'REILLY®

cloudera®



strataconf.com

#StrataHadoop



for Big Data

Garrett Grolemund & Nathan Stephens

RStudio, Inc.

Sept 2016

HELLO

my name is

Garrett



@StatGarrett

HELLO

my name is

Nathan



@nwstephens

HELLO

my name is

Sean

HELLO
my name is

?

Warm up

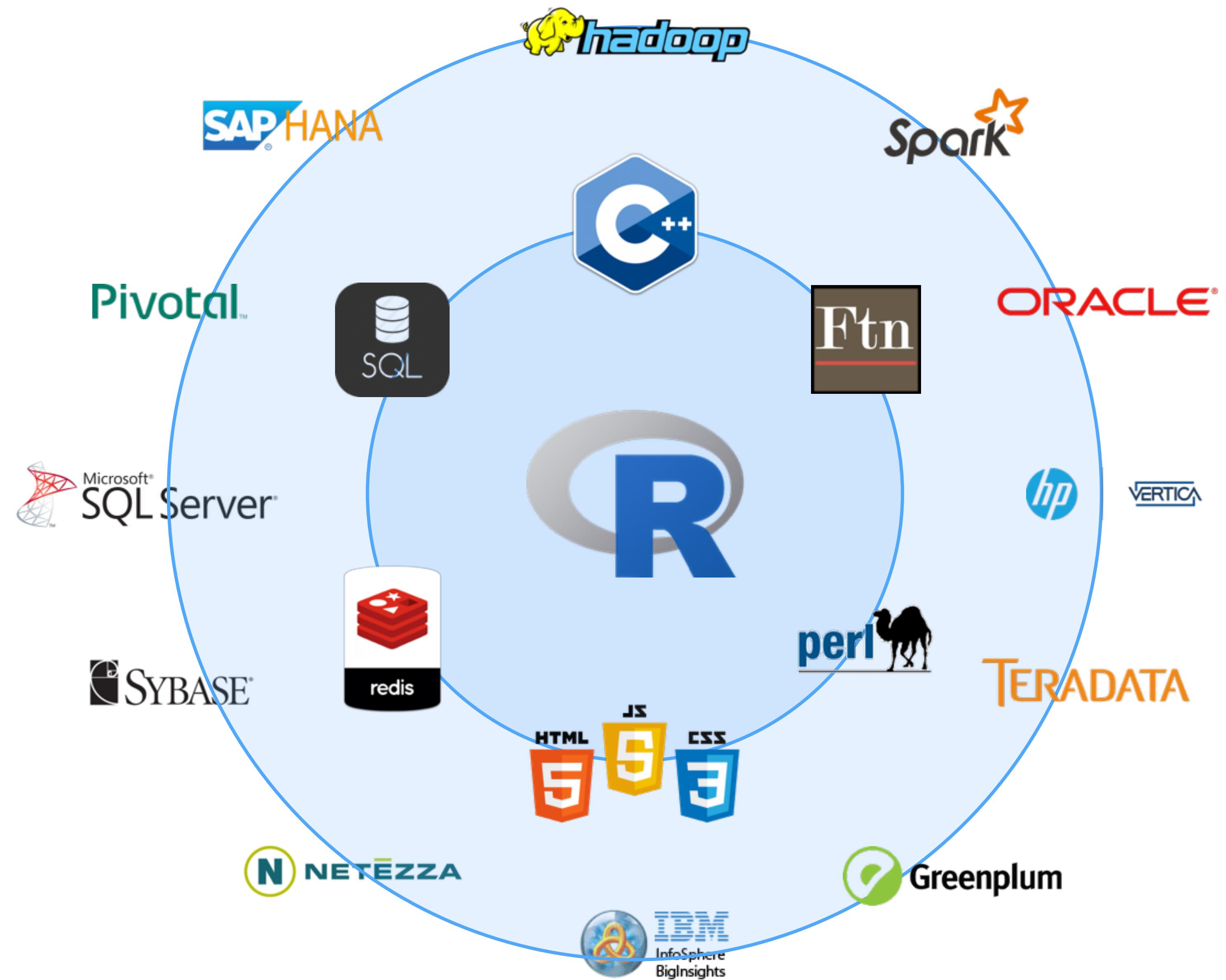
Introduce yourself to your table

- Name
- What do you do with data?
- For how long have you been using R?









Velocity

Volume

Big Data > 1/3 RAM

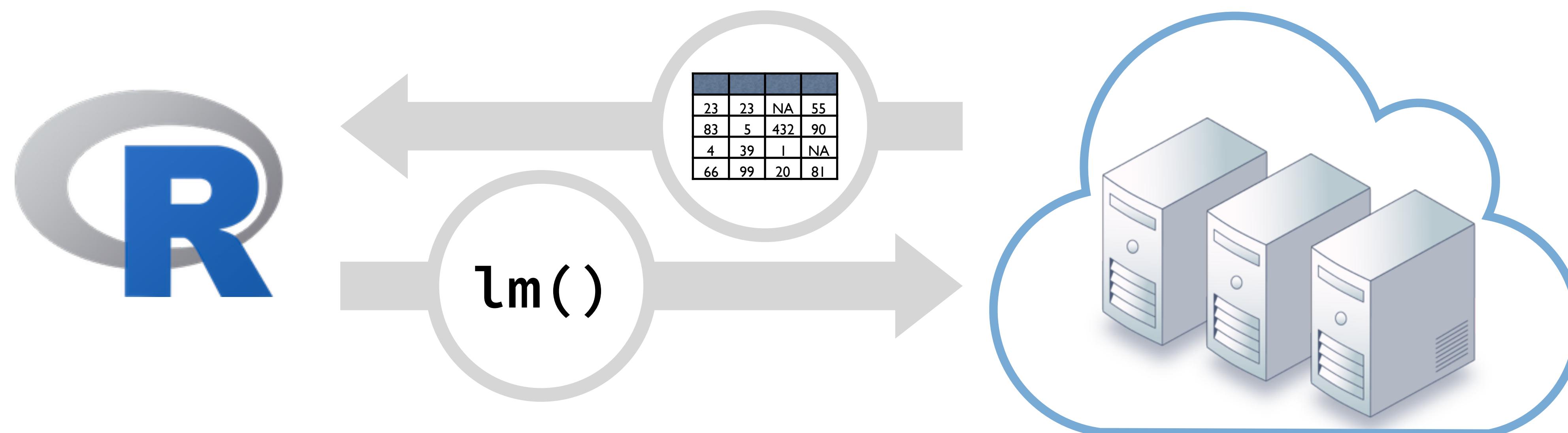
Variety

Veracity

General Strategy

Store big data in a data warehouse

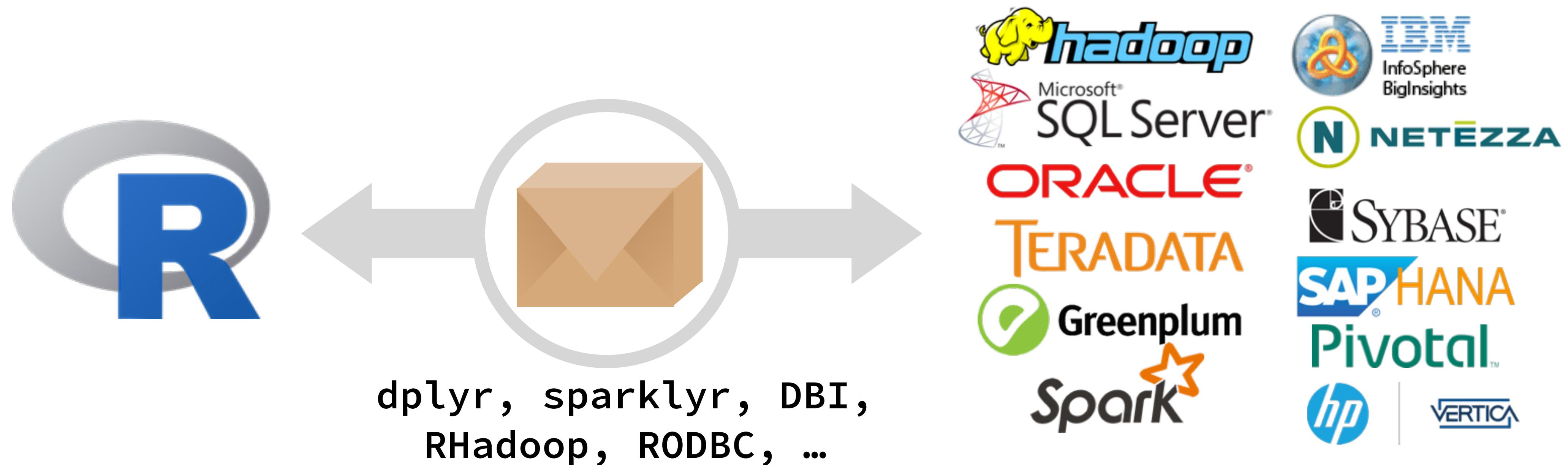
1. Pass subsets of data from warehouse to R
2. Transform R code, pass to warehouse.

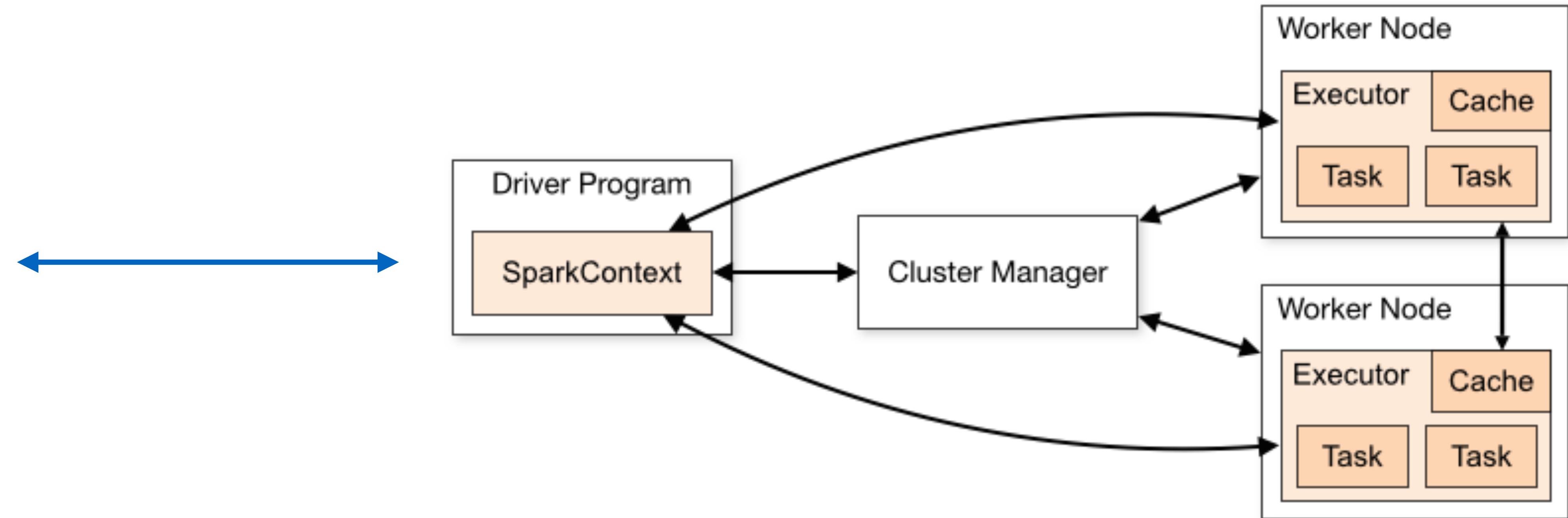


General Strategy

Store big data in a data warehouse

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





Demo

Outline

1. dplyr

The basic, universal commands

2. Databases

A quick aside, using dplyr with DBMS

3. sparklyr

Using dplyr and sparklyr with Spark

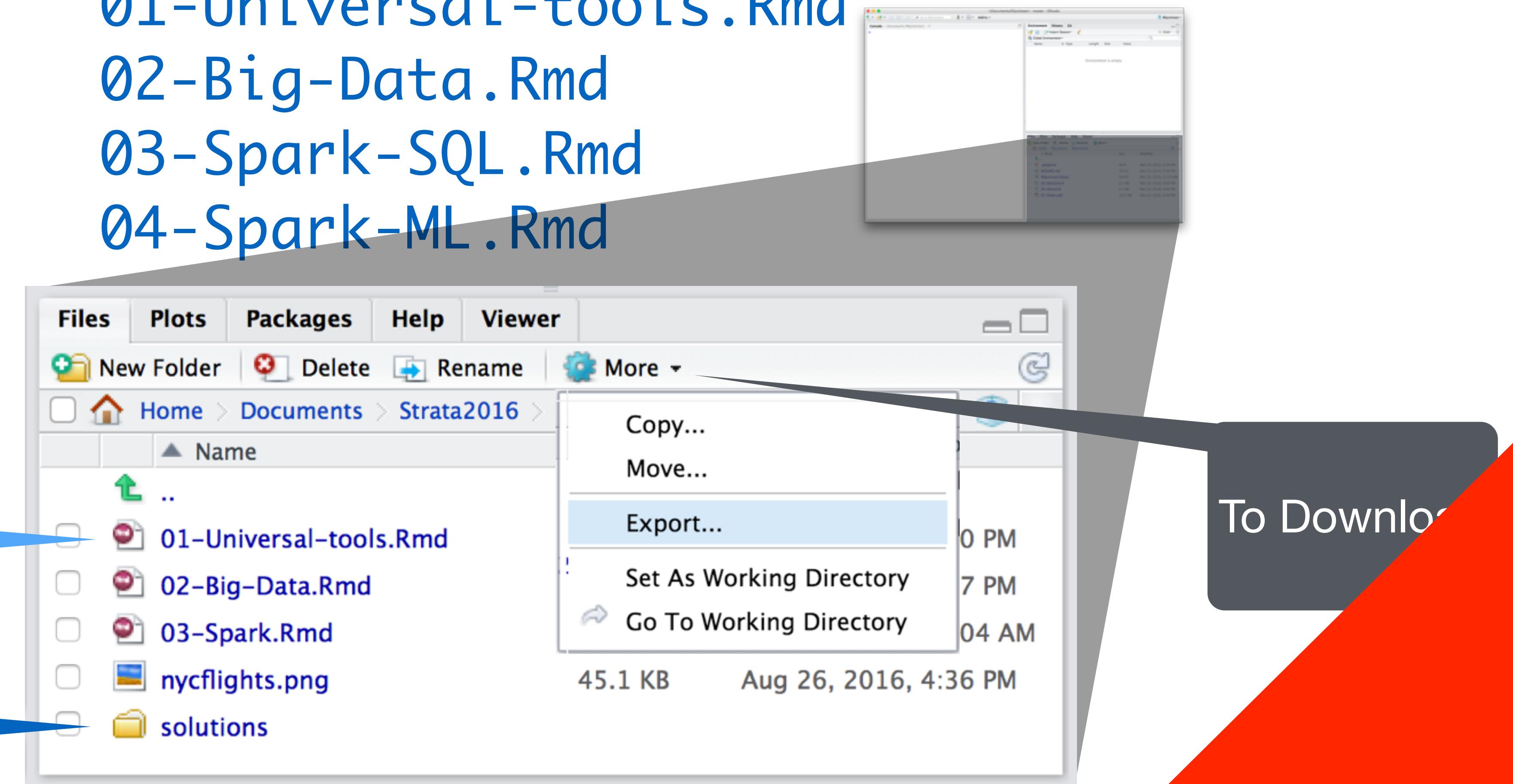
Class Materials

In the **Files** pane, you will find

- 01-Universal-tools.Rmd**
- 02-Big-Data.Rmd**
- 03-Spark-SQL.Rmd**
- 04-Spark-ML.Rmd**

Exercises

Solutions



To Download

**What is
R Markdown?**

Exercise 1

Open 01-Universal-tools.Rmd.

Read through the Notebook Logistics section and try out the features described.

Stop when you get to the first horizontal rule (***)



dplyr

R tools for data
manipulation

Packages

A bundle of functions, data sets, and help pages that you install in addition to the base R installation, e.g.

```
install.packages("dplyr")
```

To use a package, you must load it with `library()`



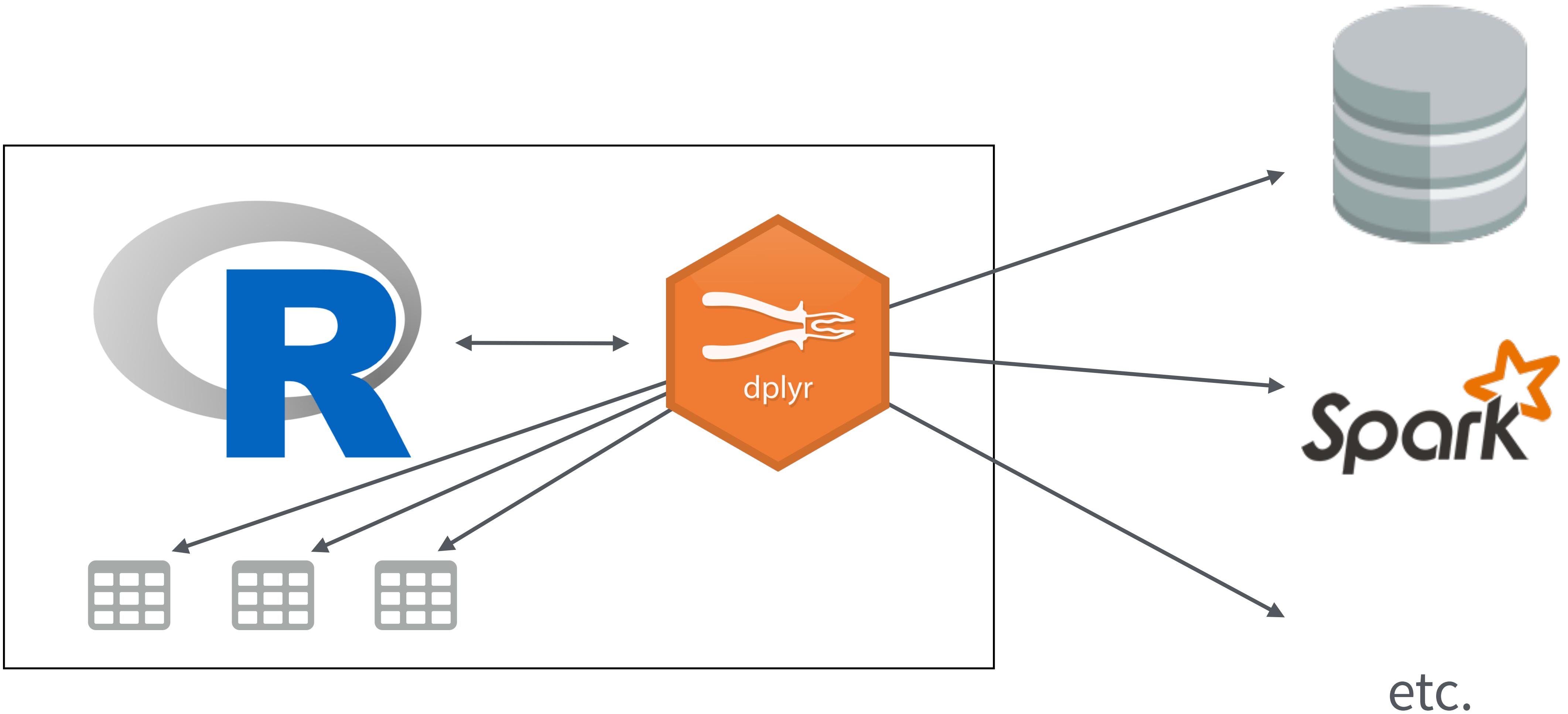
```
function1()  
function2()  
function3()  
function4()
```



dplyr

A grammar of data manipulation

select	left_join	bind_cols
filter	right_join	bind_rows
arrange	inner_join	union
mutate	full_join	intersect
summarise	semi_join	setdiff
group_by	anti_join	`%>%`



Single Table Verbs

Manipulate tabular data

`select`
`mutate`

`filter`
`arrange`



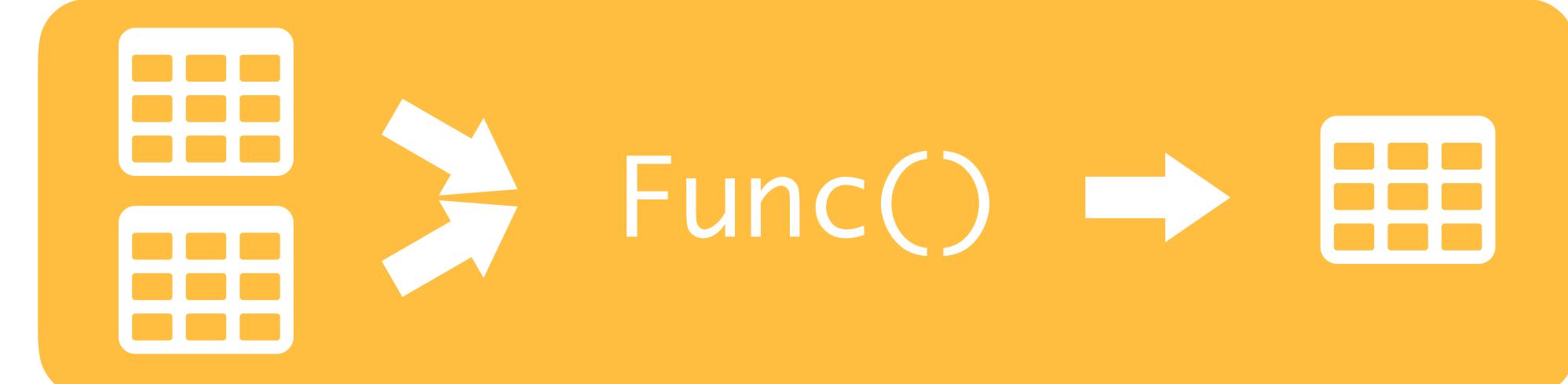
`summarise`
`group_by`

Two Table Verbs

Join together relational data

`left_join`
`right_join`
`inner_join`

`full_join`
`semi_join`
`anti_join`



`union` `bind_cols`
`intersect` `bind_rows`
`setdiff`

Single Table Verbs

Manipulate tabular data

`select`
`mutate`

`filter`
`arrange`



`summarise`
`group_by`

Two Table Verbs

Join together relational data

`left_join`
`right_join`
`inner_join`

`full_join`
`semi_join`
`anti_join`



`union` `bind_cols`
`intersect` `bind_rows`
`setdiff`

select()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	pressure
Alberto	1007
Alex	1009
Allison	1005
Ana	1013
Arlene	1010
Arthur	1010

`select(storms, storm, pressure)`

* These data sets are in the EDAWR package

mutate()

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date	ratio
Alberto	110	1007	2000-08-12	9.15
Alex	45	1009	1998-07-30	22.42
Allison	65	1005	1995-06-04	15.46
Ana	40	1013	1997-07-01	25.32
Arlene	50	1010	1999-06-13	20.20
Arthur	45	1010	1996-06-21	22.44

```
mutate(storms, ratio = pressure / wind)
```

* These data sets are in the EDAWR package

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alberto	110	1007	2000-08-12

```
filter(storms, wind == max(wind))
```

* These data sets are in the EDAWR package

logical tests in R

?Comparison

<	Less than
>	Greater than
==	Equal to
<=	Less than or equal to
>=	Greater than or equal to
!=	Not equal to
%in%	Group membership
is.na	Is NA
!is.na	Is not NA

?base::Logic

&	boolean and
	boolean or
xor	exactly or
!	not
any	any true
all	all true

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13

`filter(storms, wind >= 50)`

* These data sets are in the EDAWR package

filter()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alex	45	1009	1998-07-30
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21

`filter(storms, wind < 60, wind >= 40)`

* These data sets are in the EDAWR package

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

arrange(storms, **wind**)

* These data sets are in the EDAWR package

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Ana	40	1013	1997-07-01
Alex	45	1009	1998-07-30
Arthur	45	1010	1996-06-21
Arlene	50	1010	1999-06-13
Allison	65	1005	1995-06-04
Alberto	110	1007	2000-08-12

arrange(storms, wind)

* These data sets are in the EDAWR package

arrange()

storms

storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Alex	45	1009	1998-07-30
Allison	65	1005	1995-06-04
Ana	40	1013	1997-07-01
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21



storm	wind	pressure	date
Alberto	110	1007	2000-08-12
Allison	65	1005	1995-06-04
Arlene	50	1010	1999-06-13
Arthur	45	1010	1996-06-21
Alex	45	1009	1998-07-30
Ana	40	1013	1997-07-01

`arrange(storms, desc(wind))`

* These data sets are in the EDAWR package

summarise()

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



median
22.5

summarise(pollution, median = median(amount))

* These data sets are in the EDAWR package

summarise()

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56



mean	sum	n
42	252	6

```
summarise(pollution, mean = mean(amount), sum = sum(amount), n = n())
```

* These data sets are in the EDAWR package

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

* These data sets are in the EDAWR package

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

mean	sum	n
42	252	6

* These data sets are in the EDAWR package

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14



mean	sum	n
18.5	37	2

London	large	22
London	small	16



19.0	38	2
------	----	---

Beijing	large	121
Beijing	small	56



88.5	177	2
------	-----	---

group_by() + summarise()

* These data sets are in the EDAWR package

group_by()

city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
New York	small	14
London	large	22
London	small	16
Beijing	large	121
Beijing	small	56

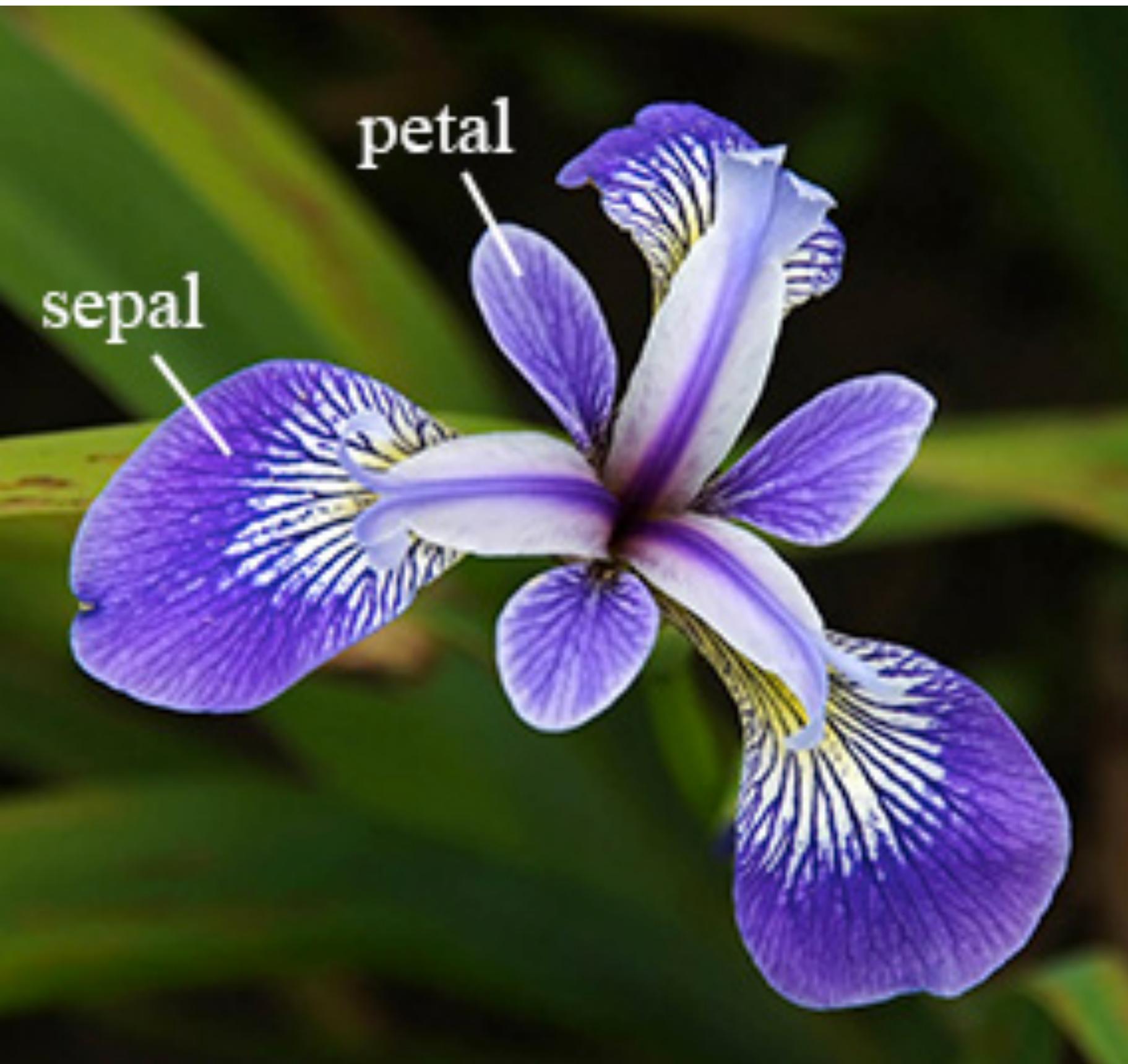
city	particle size	amount ($\mu\text{g}/\text{m}^3$)
New York	large	23
	small	14
London	large	22
	small	16
Beijing	large	121
	small	56

mean	sum	n
18.5	37	2
19.0	38	2
88.5	177	2

```
p <- group_by(pollution, city)
```

```
summarise(p, mean = mean(amount), sum = sum(amount), n = n())
```

iris

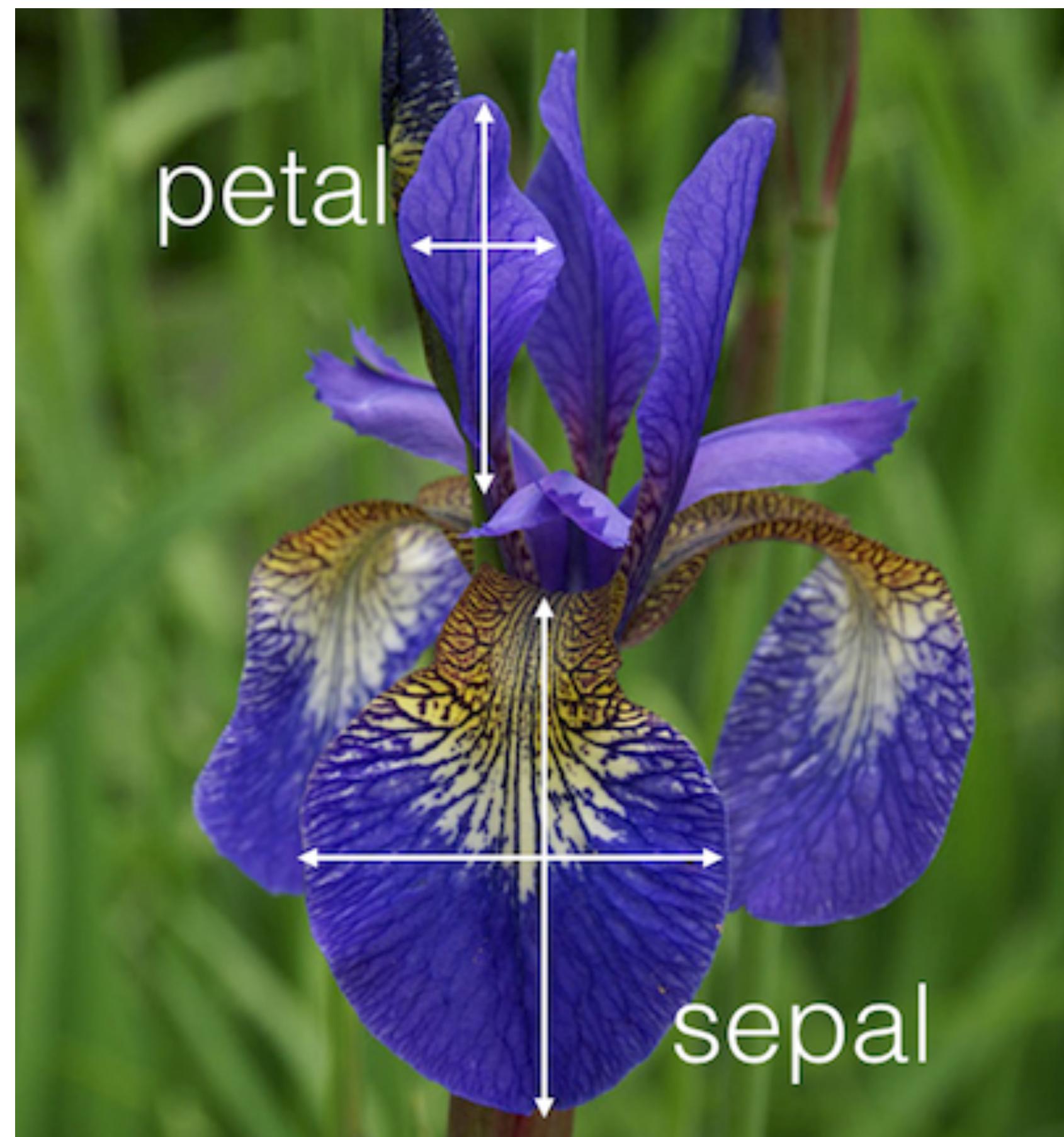


iris

Sepal.Length	Sepal.Width	Petal. length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
...
6.5	3.0	5.2	2.0	virginica
6.2	3.4	5.4	2.3	virginica
5.9	3.0	5.1	1.8	virginica

iris

On average, which species has the greatest difference in *petal width* and *petal length*?



Exercise 2

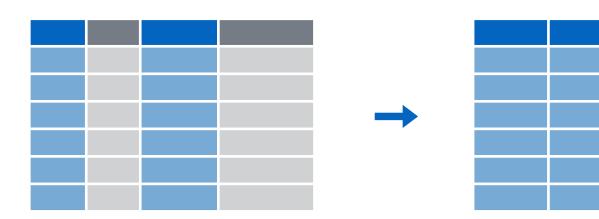
Species	avg_width	avg_length
?	?	?

1. **Group** iris by Species
2. For each group, return:
 - avg_width = mean Petal.Width
 - avg_length = mean Petal.Length
3. For each row, calculate diff = avg_length - avg_width
4. Return the **row** whose diff == the max diff
5. Return the **columns** above



```
iris1 <- group_by(iris, Species)
iris2 <- summarise(iris1,
  avg_width = mean(Petal.Width),
  avg_length = mean(Petal.Length))
iris3 <- mutate(iris2, diff = avg_length - avg_width)
iris4 <- filter(iris3, diff == max(diff))
select(iris4, Species, avg_width, avg_length)
```

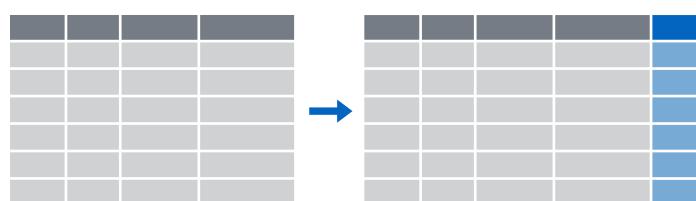
Recap: dplyr



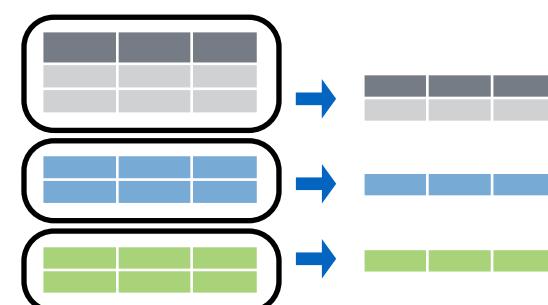
Extract columns and rows with **select()** and **filter()**



Arrange rows with **arrange()**.



Make new columns with **mutate()**.



Make groupwise summaries with **group_by()** and **summarise()**.

%>%

Ceci n'est pas une pipe.

```
iris1 <- group_by(iris, Species)
iris2 <- summarise(iris1,
  avg_width = mean(Petal.Width),
  avg_length = mean(Petal.Length))
iris3 <- mutate(iris2, diff = avg_length - avg_width)
iris4 <- filter(iris3, diff == max(diff))
select(iris4, Species, avg_width, avg_length)
```

The pipe %>% operator

%>%

These do the same thing

Try it!

```
filter(iris, Sepal.Length == max(Sepal.Length))
```

```
iris %>% filter(Sepal.Length == max(Sepal.Length))
```



iris filter_____, Sepal.Length == max(Sepal.Length))

%>%

Shortcut to type %>%

Cmd + **Shift** + **M** (Mac)

Ctrl + **Shift** + **M** (Windows)

A pipe

```
iris %>%  
  filter(Sepal.Length == max(Sepal.Length)) %>%  
  mutate(total = Sepal.Length + Sepal.Width) %>%  
  select(Species, total) %>%
```

...

Exercise 3

Use `%>%` to turn your code from Exercise 3 into a single long pipe.



```
iris1 <- group_by(iris, Species)
iris2 <- summarise(iris1,
  avg_width = mean(Petal.Width),
  avg_length = mean(Petal.Length))
iris3 <- mutate(iris2, diff = avg_length - avg_width)
iris4 <- filter(iris3, diff == max(diff))
select(iris4, Species, avg_width, avg_length)
```

```
iris %>%  
  group_by(Species) %>%  
  summarise(  
    avg_width = mean(Petal.Width),  
    avg_length = mean(Petal.Length)) %>%  
  mutate(diff = avg_length - avg_width) %>%  
  filter(diff == max(diff)) %>%  
  select(Species, avg_width, avg_length)
```

Single Table Verbs

Manipulate tabular data

`select`
`mutate`

`filter`
`arrange`



`summarise`
`group_by`

Two Table Verbs

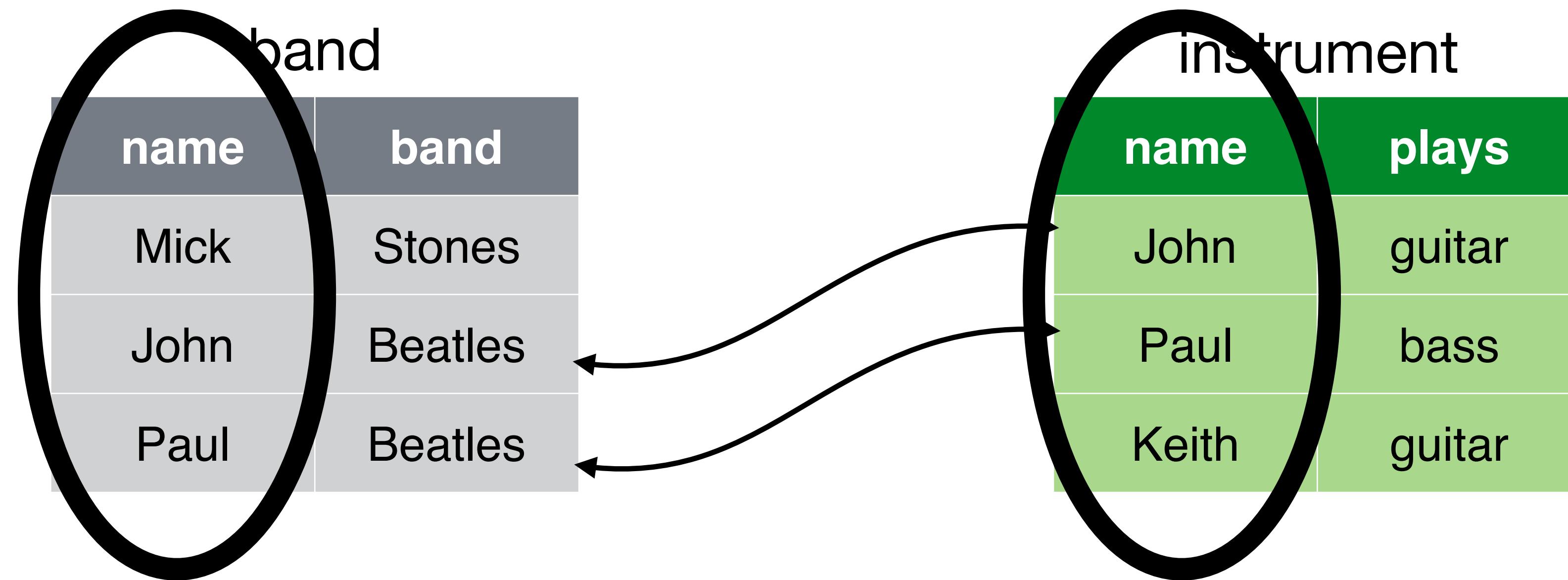
Join together relational data

`left_join`
`right_join`
`inner_join`

`full_join`
`semi_join`
`anti_join`



`union` `bind_cols`
`intersect` `bind_rows`
`setdiff`



left_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

```
left_join(band, instrument, by = "name")
```

left_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

```
left_join(band, instrument, by = "name")
```

Exercise 4

Recreate bands and instruments in your notebook.

Use them to test each join function and to complete the join definitions.



left_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

```
left_join(band, instrument, by = "name")
```

right_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

```
right_join(band, instrument, by = "name")
```

inner_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

```
inner_join(band, instrument, by = "name")
```

full_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

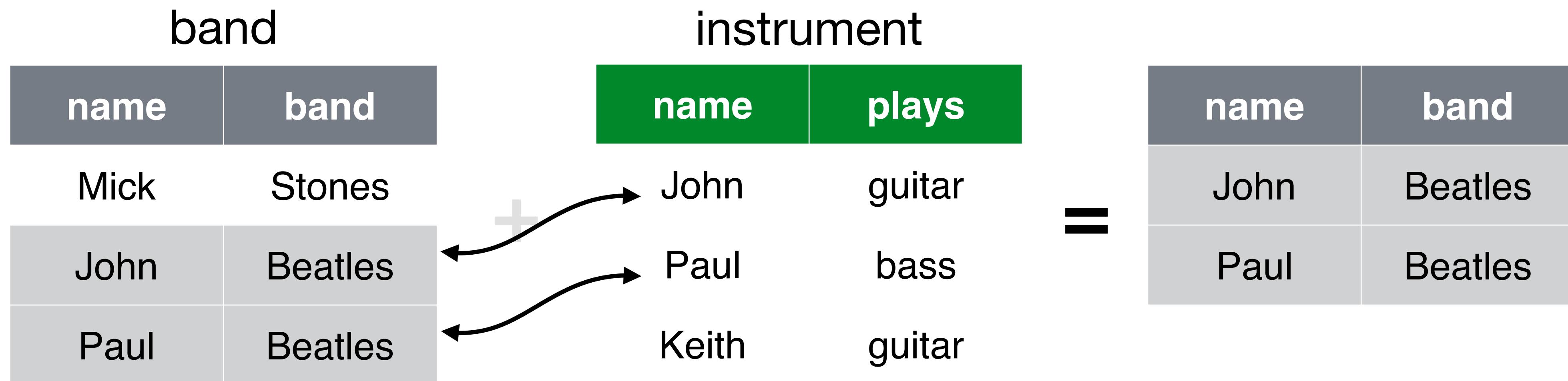
name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

```
full_join(band, instrument, by = "name")
```

semi_join()



```
semi_join(band, instrument, by = "name")
```

anti_join()

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

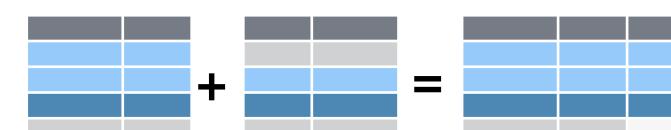
name	plays
John	guitar
Paul	bass
Keith	guitar

=

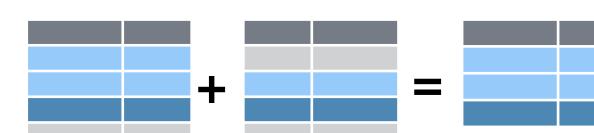
name	band
Mick	Stones

```
anti_join(band, instrument, by = "name")
```

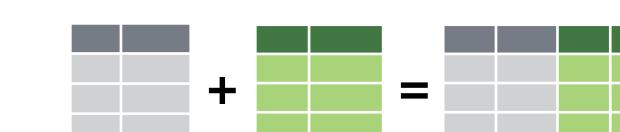
Recap: Two table verbs



Join together observations with **`left_join()`**,
`right_join()`, **`inner_join()`**, and **`full_join()`**



Filter one data set based on another with
`semi_join()` and **`anti_join()`**



Bind data sets together with **`bind_rows()`**
and **`bind_cols()`**



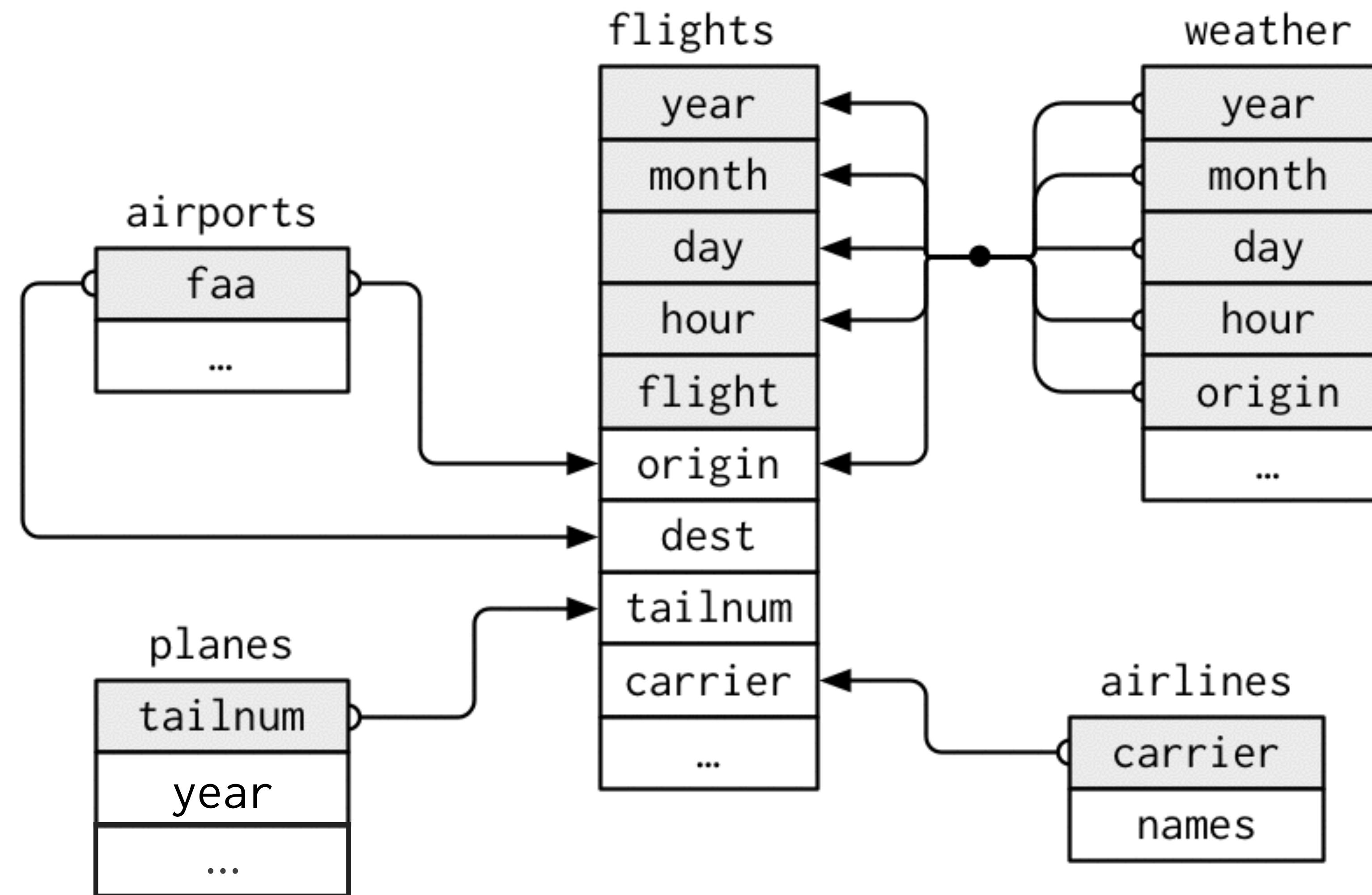
Do set operations on rows with dplyr's
`union()`, **`intersect()`**, and **`setdiff()`**

nycflights13



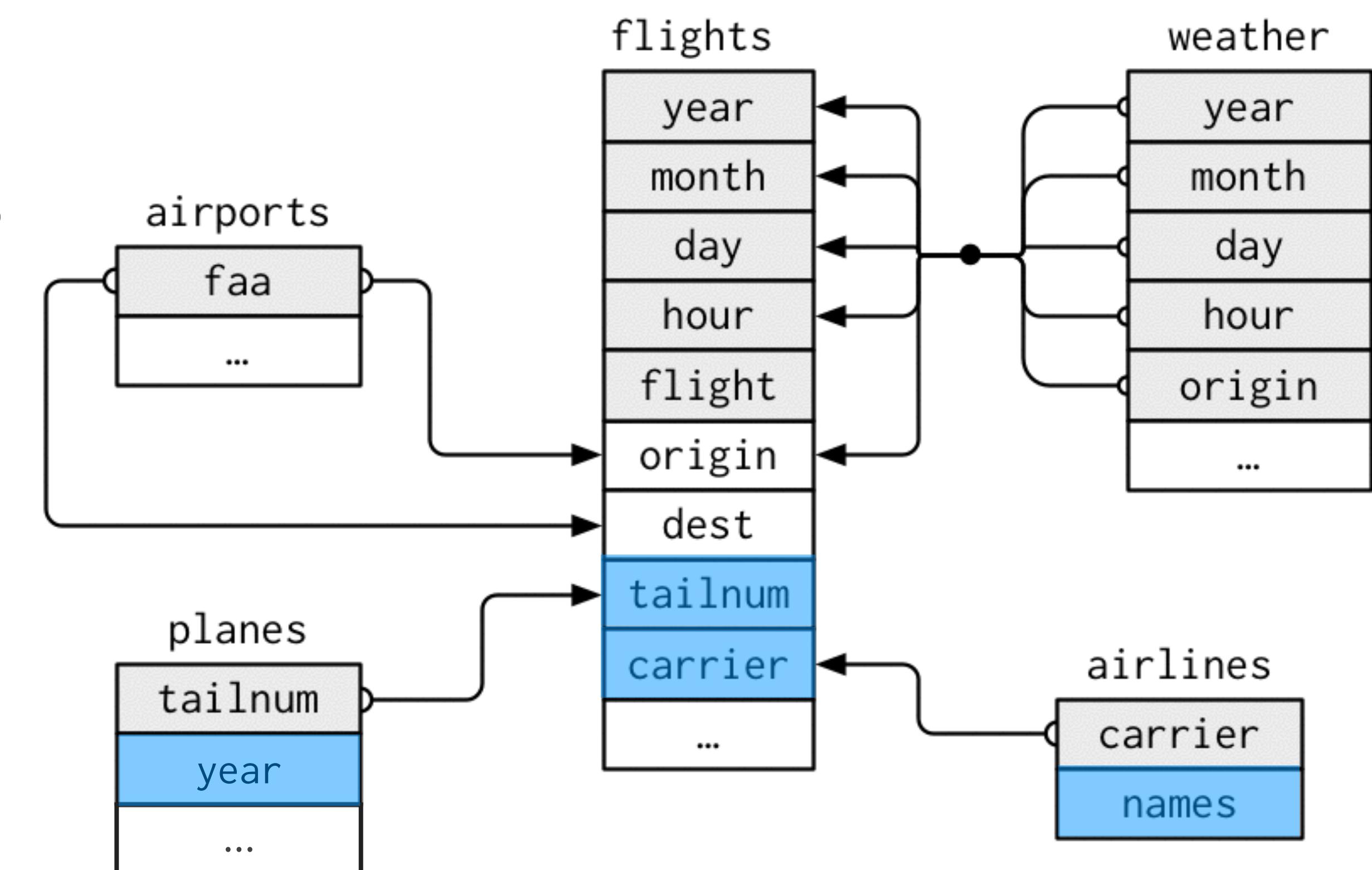
Data about every flight that departed La
Guardia, JFK, or Newark airports in 2013

nycflights13



nycflights13

On average, which airline has the newest planes (assigned to the NYC area)?



```
flights %>%  
  distinct(carrier, tailnum) %>%
```

Exercise 6

Determine which airline has the newest planes. Start with the code in the notebook.



```
flights %>%  
  distinct(carrier, tailnum) %>%  
  left_join(planes, by = "tailnum") %>%  
  group_by(carrier) %>%  
  summarise(avg = mean(year, na.rm = TRUE),  
            n = n(), nas = sum(is.na(year))) %>%  
  left_join(airlines, by = "carrier") %>%  
  select(name, avg, n, nas) %>%  
  arrange(desc(avg))
```

name	avg	n	nas
Hawaiian Airlines Inc.	2011.769	14	1
Virgin America	2008.712	53	1
Frontier Airlines Inc.	2008.000	26	3
Alaska Airlines Inc.	2007.843	84	6
JetBlue Airways	2006.503	193	0
...

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/

Data does not
fit in memory

Airlines database



Amazon **Redshift**

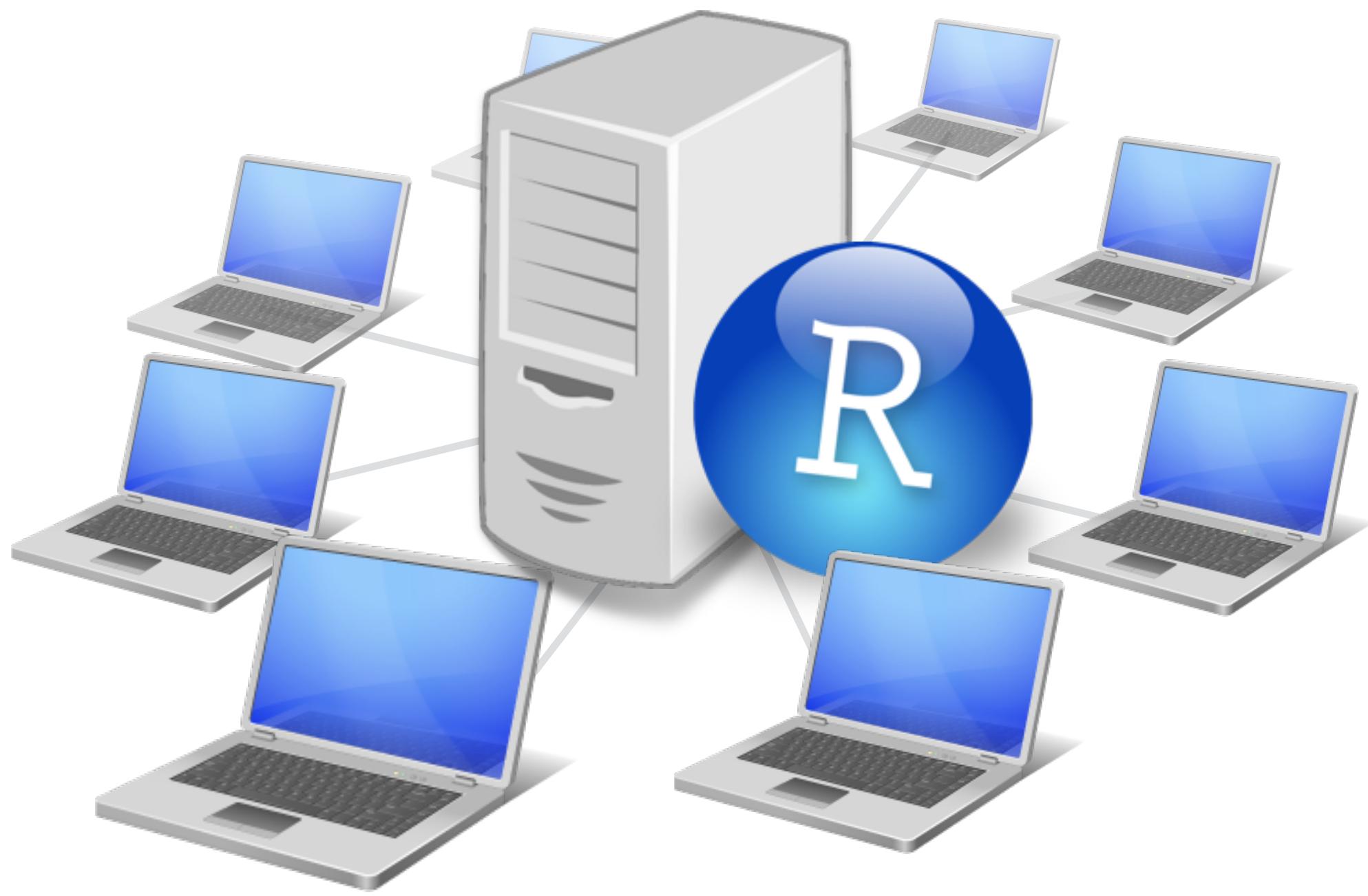
database: **airontime**
host: **sol-eng-sparklyr.cyii7eabibhu.us-east-1.redshift.amazonaws.com**
port: **5439**
user: **redshift_user**
password: **ABCd4321**

databases

manipulating big data with R

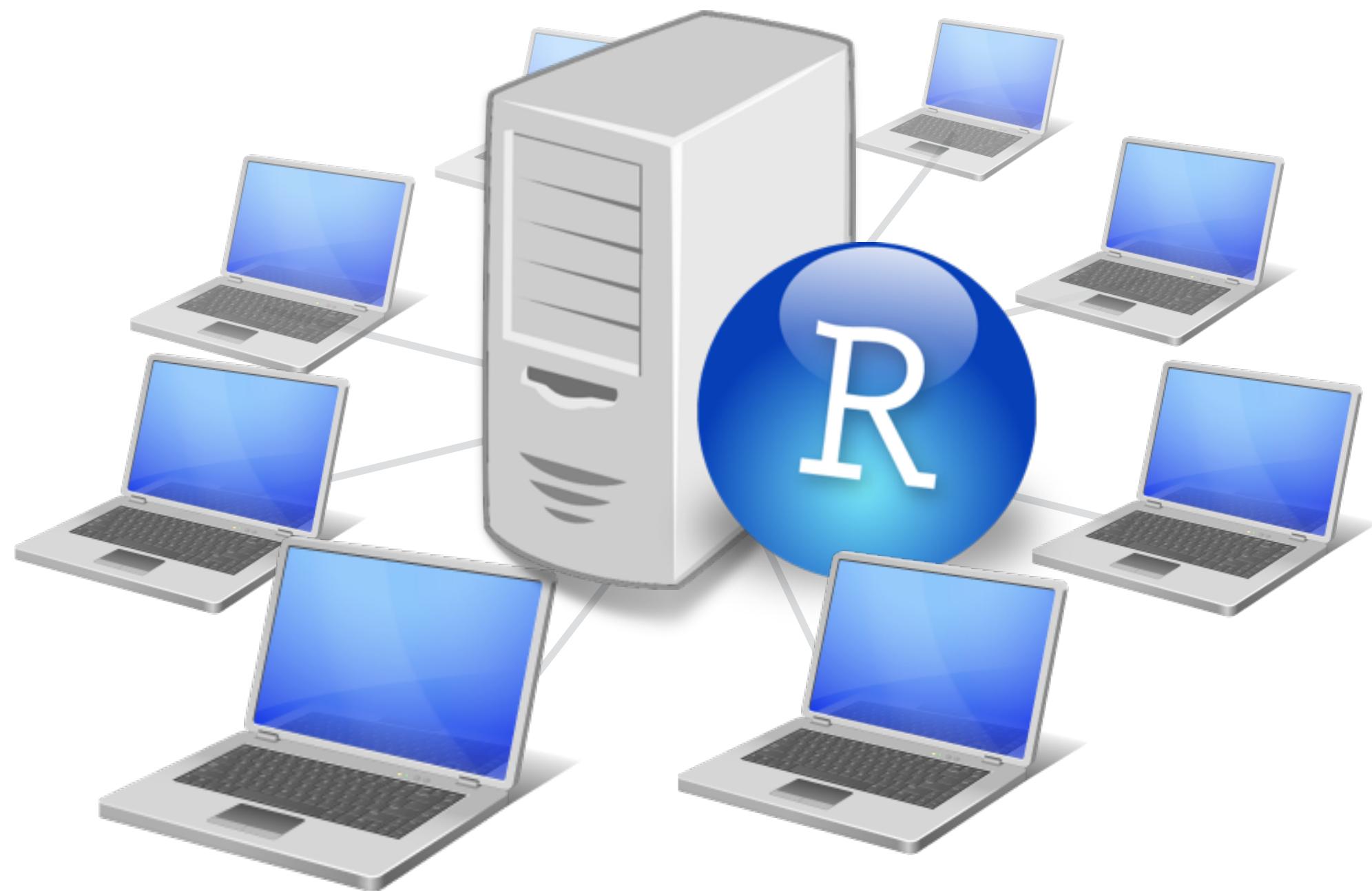


RStudio
Desktop IDE



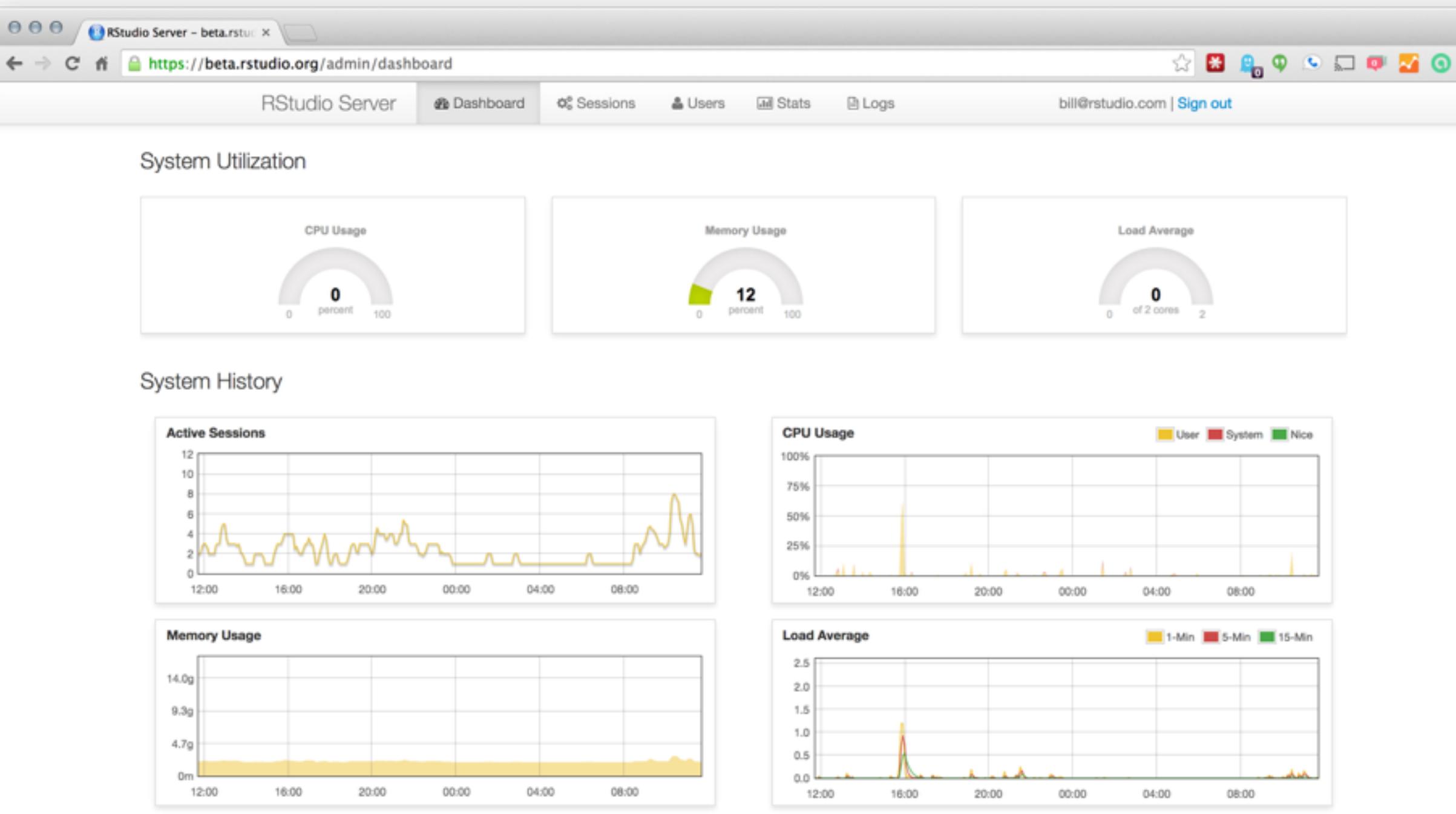
RStudio Server Pro

- Security
- Load balancing
- Administrative tools
- Resource management
- Metrics and monitoring
- Multiple sessions
- Collaborative editing
- Easy R versioning
- Audit history



RStudio Server Pro

Administrative Dashboard



View and manage resource utilization

The screenshot shows the 'Users' page of the RStudio Server administrative dashboard. It displays a table of user statistics with 50 entries per page. The columns include Username, Last Session, 30-Day Use, Avg Mem, and Avg CPU %. The table lists various email addresses and their corresponding usage metrics.

Username	Last Session	30-Day Use	Avg Mem	Avg CPU %
weisuxiaobaba@gmail.com	Wed Sep 24 05:58 AM	3 hr, 48 min	138m	14.0
jocelynondohue@gwmail.gwu.edu	Tue Sep 09 11:10 PM	53 min	189m	4.8
hhaider@gwmail.gwu.edu	Thu Sep 25 04:11 PM	3 hr, 6 min	377m	3.6
leyoung2@gmail.com	Fri Sep 26 10:40 PM	17 hr, 34 min	1.2g	3.3
jnothnagel@gwmail.gwu.edu	Tue Sep 09 11:10 PM	6 min	159m	3.3
jhbang0518@gwmail.gwu.edu	Tue Sep 09 11:09 PM	50 min	198m	3.2
divinemeho@gmail.com	Mon Sep 29 01:06 PM	5 hr, 54 min	294m	2.2
jdinh@gwmail.gwu.edu	Tue Sep 09 11:10 PM	48 min	167m	2.1
jakeklein94@gmail.com	Wed Sep 17 02:16 PM	53 min	143m	1.6
aamayerson@gmail.com	Sat Sep 13 05:46 AM	2 hr, 19 min	883m	1.4
sarahreswow@gwmail.gwu.edu	Wed Sep 24 05:54 PM	2 hr, 16 min	121m	1.4
bmartin1@macalester.edu	Fri Sep 05 03:48 PM	1 hr, 5 min	177m	1.1
jober7@gwmail.gwu.edu	Tue Sep 09 11:08 PM	1 hr	207m	1.1
goodwin.dh@gmail.com	Wed Sep 24 06:00 PM	4 hr, 56 min	79m	0.8
wangjilayi@gwmail.gwu.edu	Tue Sep 09 11:09 PM	55 min	157m	0.7
mansewerz@gmail.com	Wed Sep 03 04:46 PM	1 min	70m	0.7
ellenlempres@gwmail.gwu.edu	Thu Sep 25 06:07 PM	2 hr, 38 min	201m	0.6
grady.lenkin@gmail.com	Wed Sep 24 05:23 AM	4 hr, 24 min	182m	0.5

View and manage active sessions

The screenshot shows the 'Sessions' page of the RStudio Server administrative dashboard. It displays a table of active sessions with 50 entries per page. The columns include Pid, Username, Duration, CPU Time, CPU %, Mem, Mem %, and Actions. The table lists two sessions: one for kvu001@gmail.com and one for mingwen.an@gmail.com.

Pid	Username	Duration	CPU Time	CPU %	Mem	Mem %	Actions
32219	kvu001@gmail.com	10:51:52	11:36:32	0.0	718m	4.1	
12676	mingwen.an@gmail.com	00:06:38	00:00:03	0.0	73m	0.4	

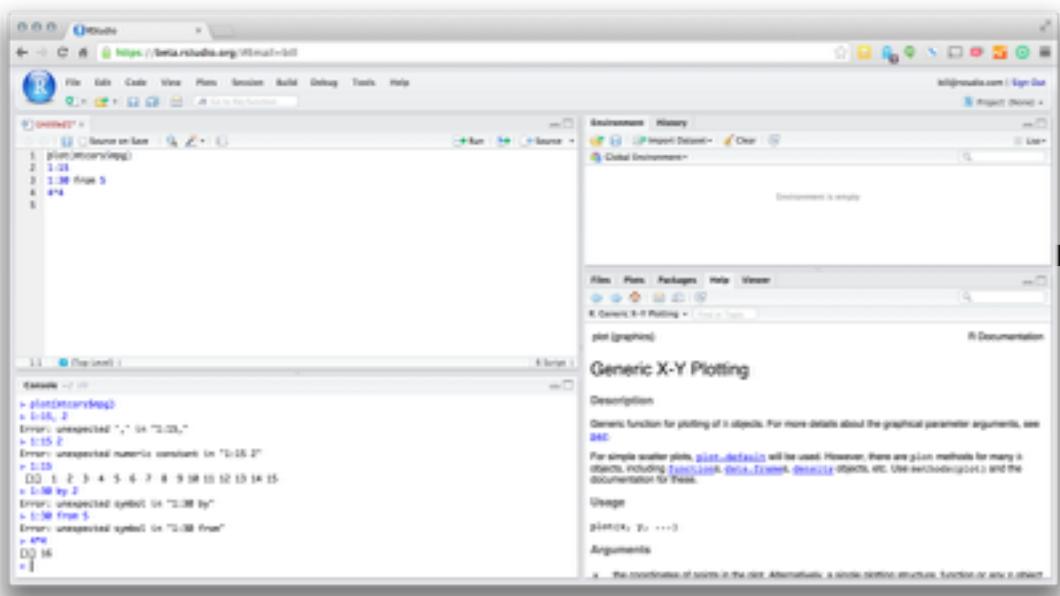
Showing 1 to 2 of 2 entries

← Previous 1 Next →



User Browser

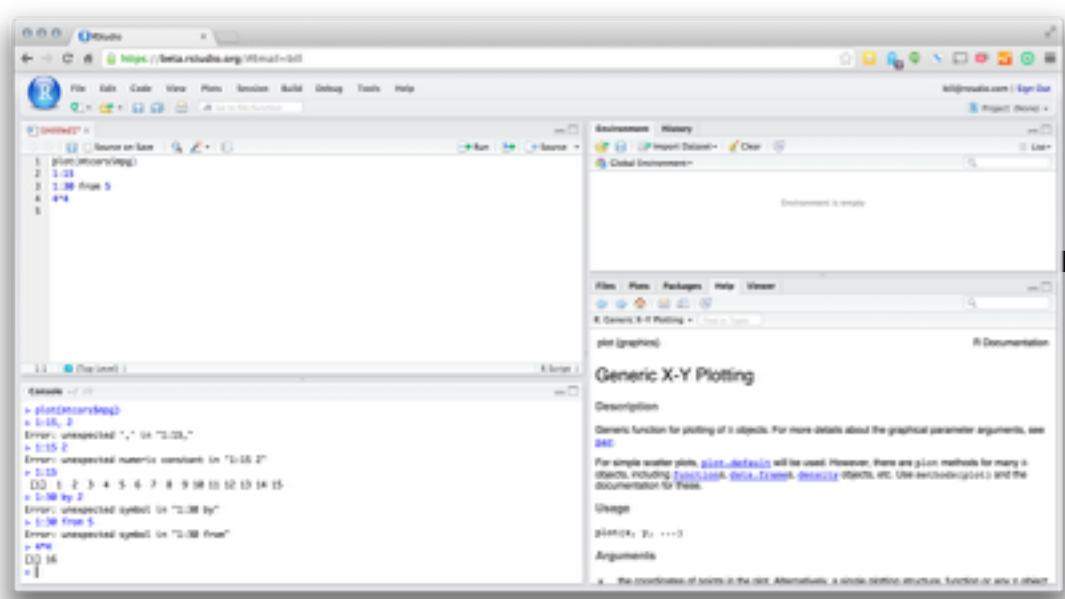
Server



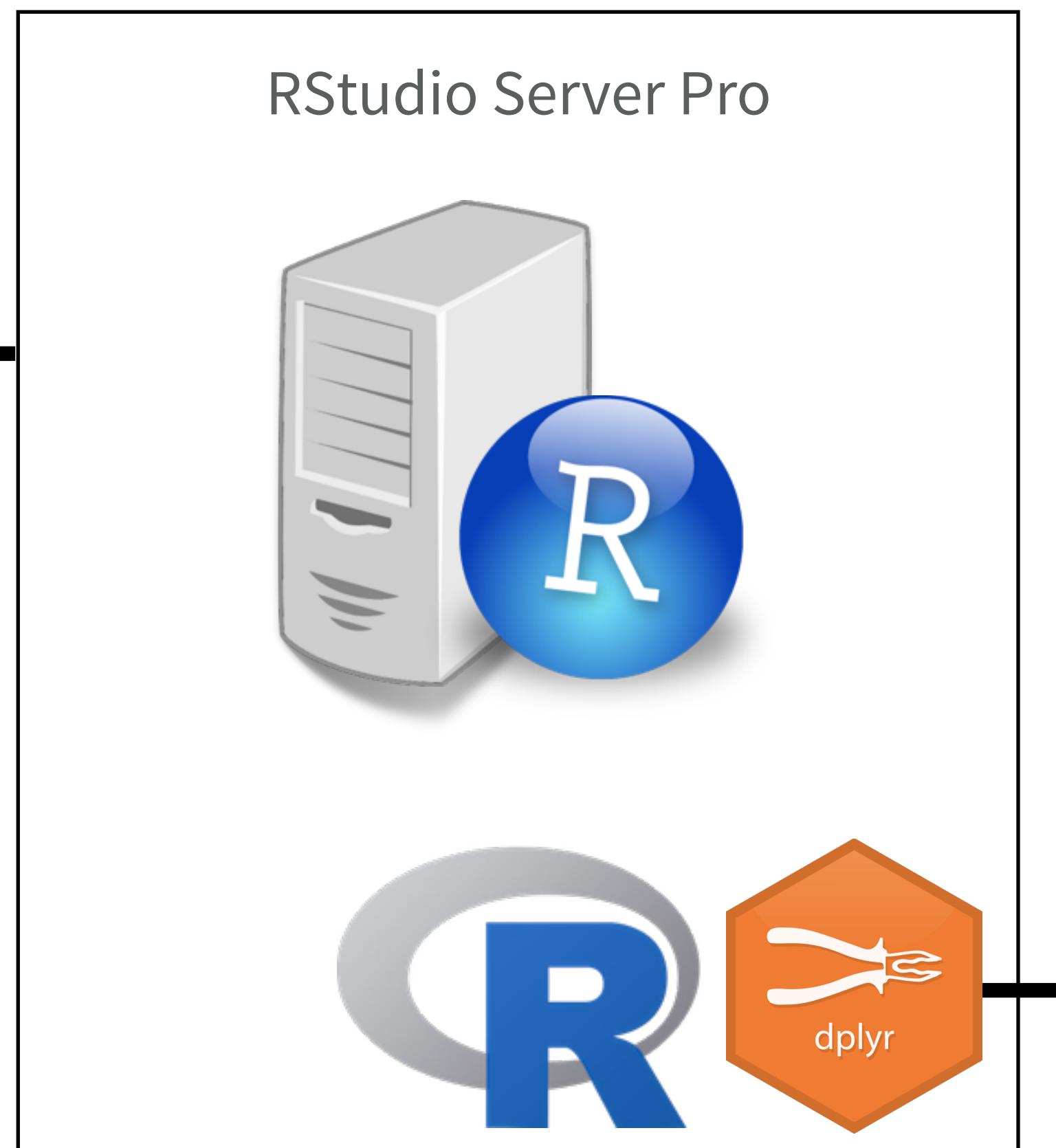
RStudio Server Pro



User Browser



Server



Database



1. Create a connection

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

Save
to use

src_driver
function

Lists tables
in DB

driver
specific args

```
src_tbls(con)
```

```
## "iris"    "iris2"   "iris3"
```

dplyr driver functions

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

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

Exercise 1

Open 2-Big-Data.Rmd in the class materials. Use the information in the Airlines Database section to create a `dplyr` connection to the database.

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



```
air <- src_postgres(  
  dbname = 'airontime',  
  host='sol-eng-sparklyr.cyii7eabibhu.us-east-1.redshift.amazonaws.com',  
  port = '5439',  
  user = 'redshift_user',  
  password = 'ABCd4321')
```

2. Create a table reference

connection
to DB

name of
table in DB

```
tab <- tbl(con, "table_name")
```

Use `tbl()` to create objects that refer to tables
in the database

3. Manipulate the reference

Treat the reference as if it were a table in R. dplyr will translate your code to SQL and execute it in the DBMS.*

```
flights <- tbl(air, "flights")
flights %>%
  distinct(uniquecarrier, tailnum) %>%
...

```

Exercise 1

1. Open 2-Big-Data.Rmd in the class materials. Run the first code chunk to connect to the database.
2. The second code chunk applies your nycflights13 analysis to the new data sets. Create table references named:
 1. flights for the flights table
 2. planes for the planes table
 3. airlines for the airlines table

Then re-run the analysis to see which airline used the newest planes.



```
flights <- tbl(air, "flights")
airlines <- tbl(air, "carriers")
planes <- tbl(air, "planes")

flights %>%
  distinct(uniquecarrier, tailnum) %>%
  mutate(tailnum = substring(tailnum, 1L, 5L)) %>%
  left_join(planes, by = "tailnum") %>%
  group_by(uniquecarrier) %>%
  summarise(avg = mean(year, na.rm = TRUE),
            n = n(), nas = sum(is.na(year))) %>%
  left_join(airlines, by = "uniquecarrier") %>%
  select(name, avg, n, nas) %>%
  arrange(desc(avg))
```

show_query

To see the SQL that dplyr will run.

```
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)  
  
show_query(clean)
```

```
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, year)
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 optimized 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
```

collapse()

Forces execution in **DBMS**

```
q5 <- 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

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
## ...     ...     ...     ...
```

4. Collect the results

Use `collect()` to import the entire set of results into R.

```
q6 <- flights %>%  
  filter(year > 2007, depdelay > 15) %>%  
  filter(depdelay == 240) %>%  
  collect()
```

5. Close the connection

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

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

dplyr database workflow

1. Create a connection

```
con <- src_postgres(dbname, host, port, user, pass, ...)
```

2. Create a reference

```
tab <- tbl(con, "tablename")
```

3. Manipulate the reference

```
query <- tab %>% filter(x > 1) %>% select(x, y, z)
```

4. Collect the results

```
results <- collect(query)
```

5. Close the connection

```
rm(con); gc()
```

sparklyr

manipulating big data with
Apache Spark

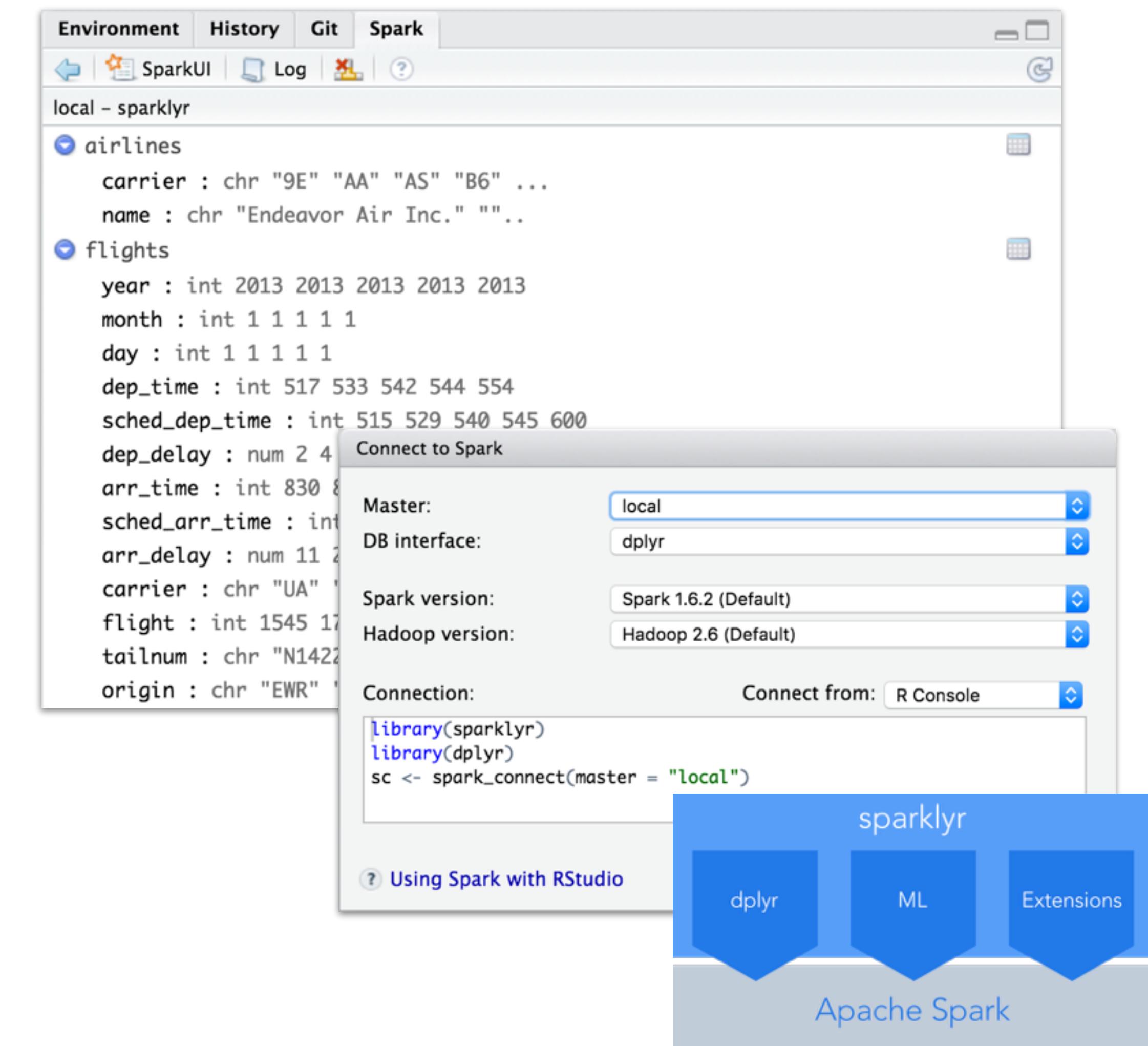
What is Spark?

- Open-source Apache computing engine
- Bigger-than-memory data, low-latency distributed computing
- Can integrate with the Hadoop ecosystem
- Built-in machine learning



sparklyr

- **R package.** New, open-source package from RStudio
- **dplyr.** Complete dplyr back-end for Spark
- **IDE.** Integrated with the RStudio IDE
- **Extensible.** Extensible foundation for Spark and R



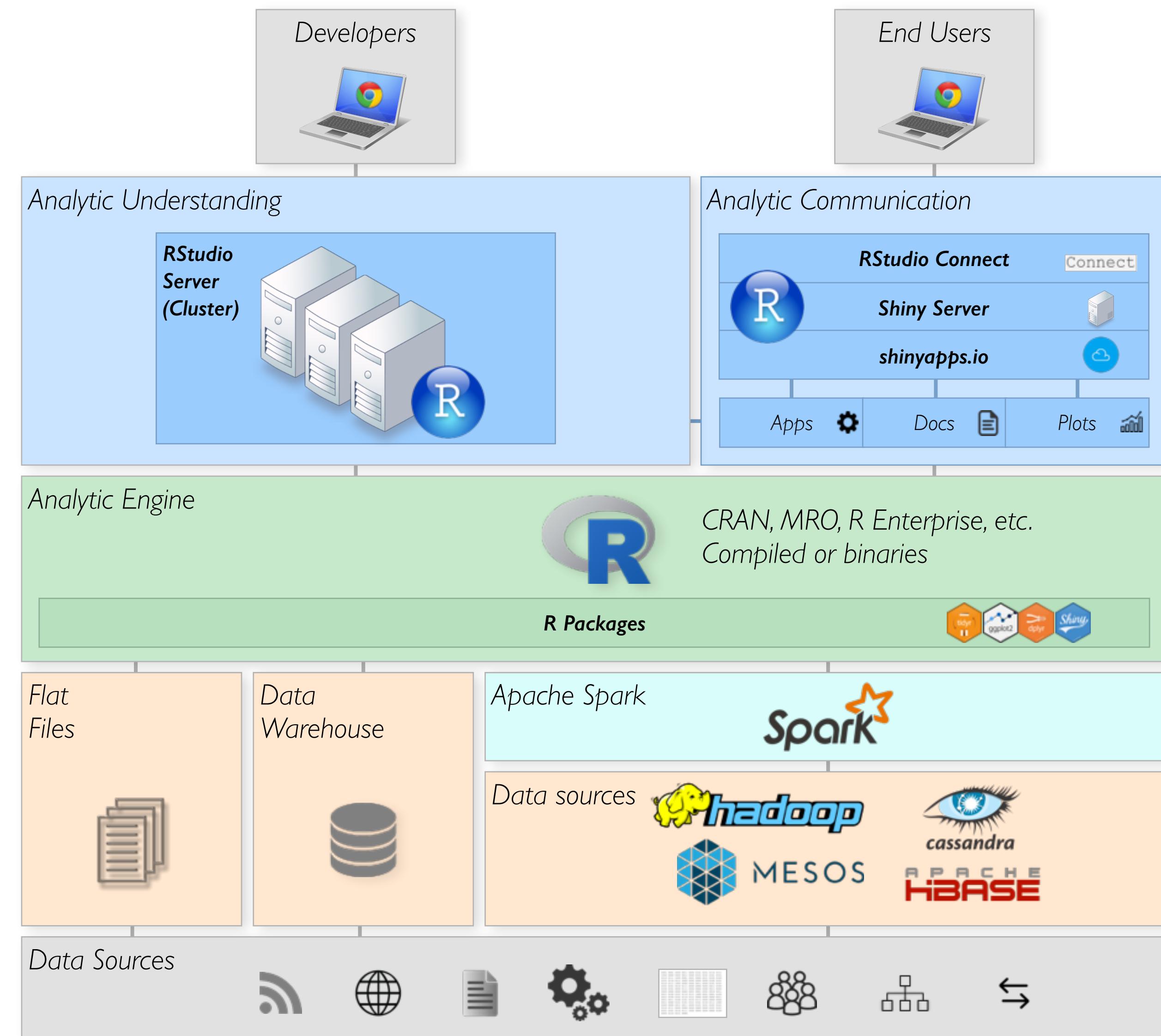
Use dplyr to write spark sql

```
library(dplyr)
```

```
# use standard verbs to filter and aggregate
select(
  filter(my_tbl, Petal_Width < 0.3),
  Petal_Length, Petal_Width
)
```

```
# use magrittr pipes for a cleaner syntax
my_tbl %>%
  filter(Petal_Width < 0.3) %>%
  select(Petal_Length, Petal_Width)
```

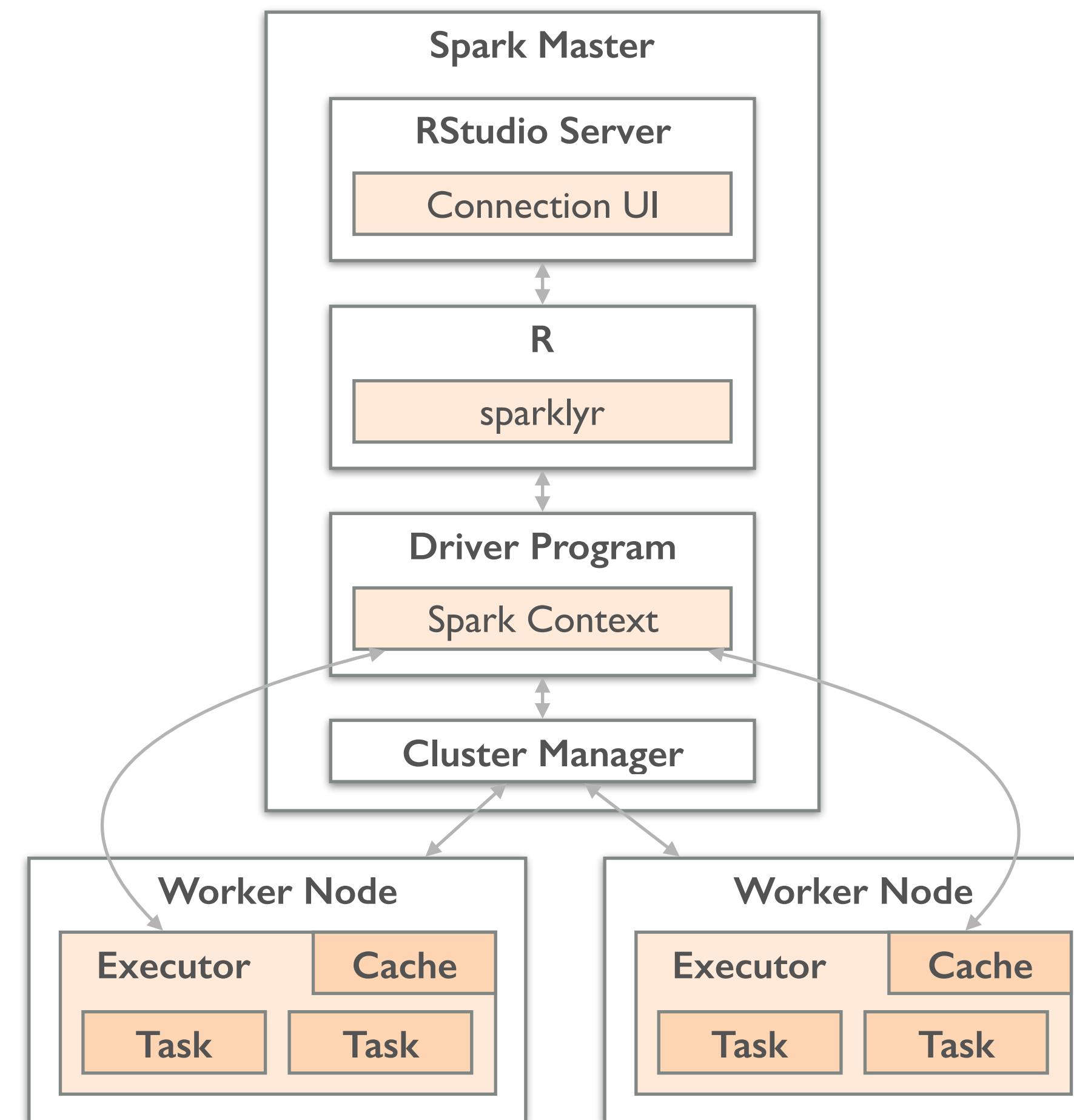
Setup



Cluster Mode

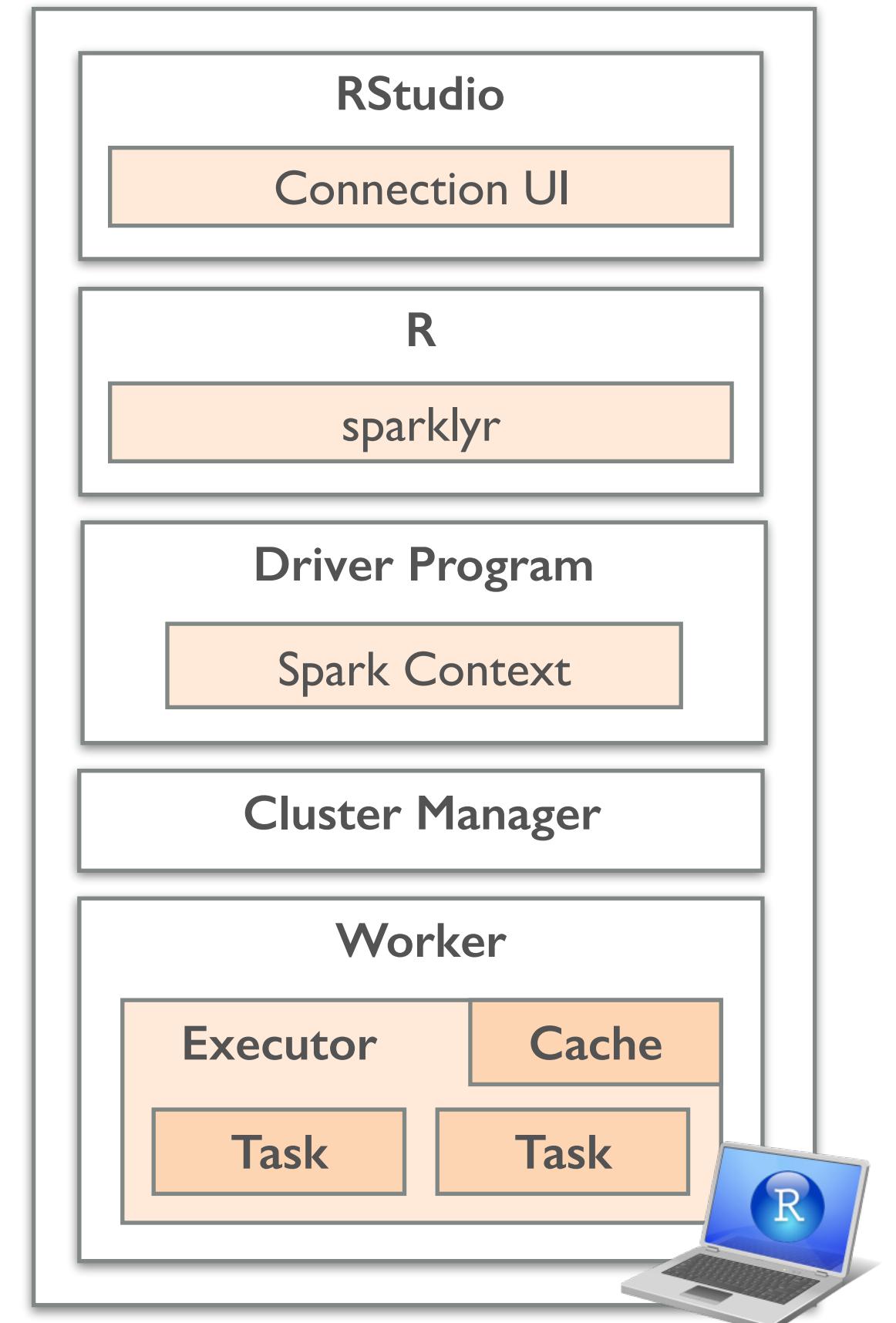
Use RStudio Server on the Spark cluster master node

```
spark_connect("spark://spark.company.org:7077")
my_tbl <- tbl(sc, "tblname")
```



Local Mode

```
library(sparklyr)  
spark_install()  
sc <- spark_connect("local")  
my_tbl <- copy_to(sc, iris)
```



Relationship to SparkR

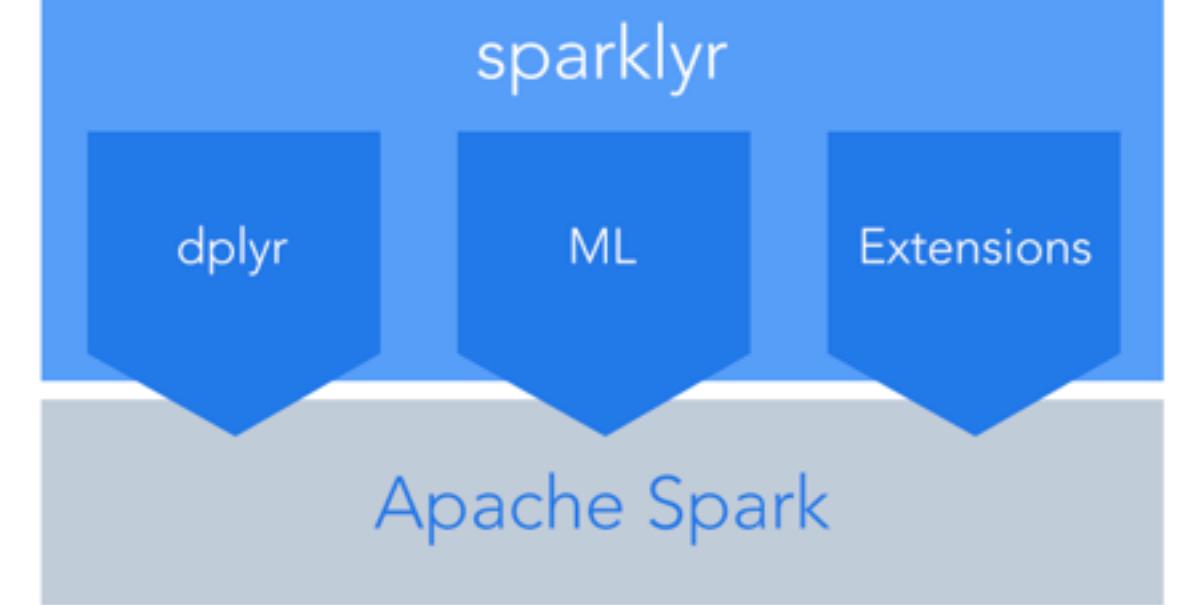
- Some differences in approach:
 - CRAN distribution
 - dplyr compatibility
- Working together to establish a common extension API

sparklyr.rstudio.com

sparklyr Home dplyr ML Extensions Deployment Reference

sparklyr — R interface for Apache Spark

- Connect to [Spark](#) from R — the sparklyr package provides a complete [dplyr](#) backend.
- Filter and aggregate Spark datasets then bring them into R for analysis and visualization.
- Use Spark's distributed [machine learning](#) library from R.
- Create [extensions](#) that call the full Spark API and provide interfaces to Spark packages.



Installation

You can install **sparklyr** using the [devtools](#) package as follows:

```
install.packages("devtools")
devtools::install_github("rstudio/sparklyr")
```

You should also install a local version of Spark for development purposes:

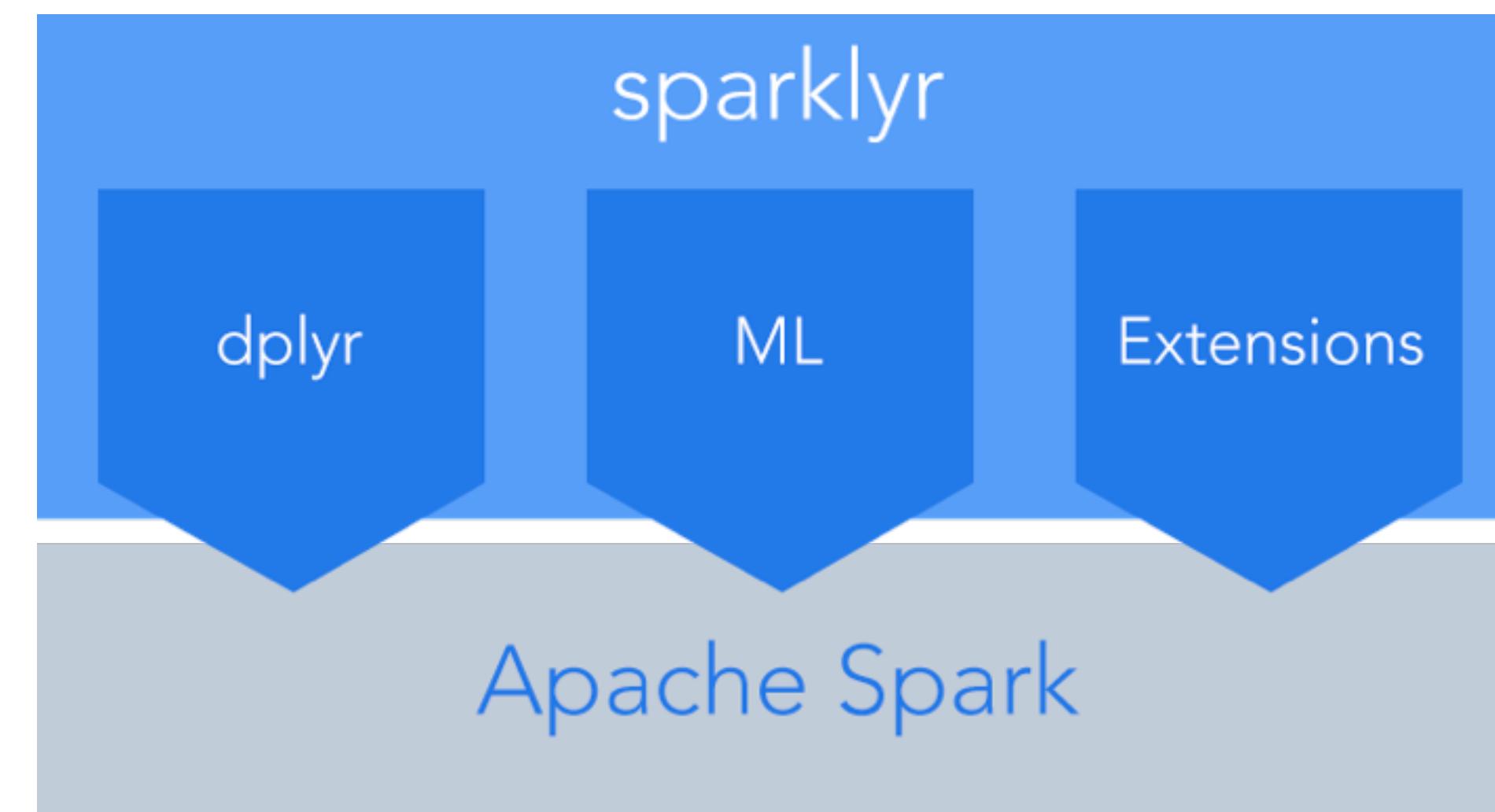
```
library(sparklyr)
spark_install(version = "1.6.1")
```

If you use the RStudio IDE, you should also download the latest [preview release](#) of the IDE which includes several enhancements for interacting with Spark (see the [RStudio IDE](#) section below for more details).

Connecting to Spark

reserved.

sparklyr Interface for Apache Spark



Spark SQL

Use dplyr syntax to translate R code
into Spark SQL (HiveQL)

DPLYR

```
my_tbl %>%  
  filter(Petal_Width < 0.3) %>%  
  select(Petal_Length, Petal_Width) %>%  
  arrange(Petal_Length)
```

SPARK SQL

```
select Petal_Length, Petal_Width  
from iris  
where Petal_Width < 0.3  
order by Petal_Length
```

Spark SQL

Open 03-Spark-SQL.Rmd for notes and exercises.

Spark ML

Execute analytic commands inside Spark

- **sdf.** Use spark data frame commands to manipulate data frames.
Examples: sdf_partition, sdf_predict, sdf_sample
- **ft.** Use feature transforms to manipulate features.
Examples: ft_bucketizer, ft_index_to_string
- **ml.** Use machine learning algorithms (ml_*) to train models.
Examples: ml_kmeans, ml_logistic_regression, ml_pca

Spark ML

Open 04-Spark-ML.Rmd for notes and exercises.

Spark Extensions

Write your own Spark extensions in R. Use extensions in Spark much like you use CRAN packages in R.

The screenshot shows a web browser displaying the sparklyr documentation. The navigation bar at the top includes links for sparklyr, Home, dplyr, ML, Extensions (which is the active tab), Deployment, and Reference. The main content area has a sidebar on the left with links to Introduction, Core Types (which is highlighted in blue), Calling Spark from R, Wrapper Functions, and Dependencies. The main content area features a section titled "Core Types" with a sub-section "Three classes are defined for representing the fundamental types of the R to Java bridge:". It lists three functions with their descriptions:

Function	Description
<code>spark_connection</code>	Connection between R and the Spark shell process
<code>spark_job</code>	Instance of a remote Spark object
<code>spark_dataframe</code>	Instance of a remote Spark DataFrame object

Below this, it states: "S3 methods are defined for each of these classes so they can be easily converted to or from objects that contain or wrap them. Note that for any given `spark_job` it's possible to discover the underlying `spark_connection` ."

The next section is titled "Calling Spark from R" with the sub-section "There are several functions available for calling the methods of Java objects and static methods of Java classes:". It lists three functions with their descriptions:

Function	Description
<code>invoke</code>	Call a method on an object
<code>invoke_new</code>	Create a new object by invoking a constructor
<code>invoke_static</code>	Call a static method on an object

At the bottom, there is a note: "For example, to create a new instance of the `java.math.BigInteger` class and then call the `longValue()` method on it you would use code like this:"

sparklyr
examples

Titanic Machine Learning

<https://beta.rstudioconnect.com/content/1518/>

The screenshot shows a web browser window with the title "Comparison of ML Classifiers L X". The address bar displays the URL <https://beta.rstudioconnect.com/content/1518/>. The page content is titled "Comparison of ML Classifiers Using Sparklyr" and includes an "Overview" section. On the left, there is a sidebar with a "Overview" tab selected, containing links to various steps: Load the data, Tidy the data, Spark SQL transforms, Spark ML transforms, Train-validation split, Train the models, Logistic regression, Other ML algorithms, Validation data, Compare results, Model lift, AUC and accuracy, Feature importance, Compare run times, and Discuss.

Comparison of ML Classifiers Using Sparklyr

Overview

You can use `sparklyr` to fit a wide variety of machine learning algorithms in Apache Spark. This analysis compares the performance of six classification models in Apache Spark on the [Titanic](#) data set.

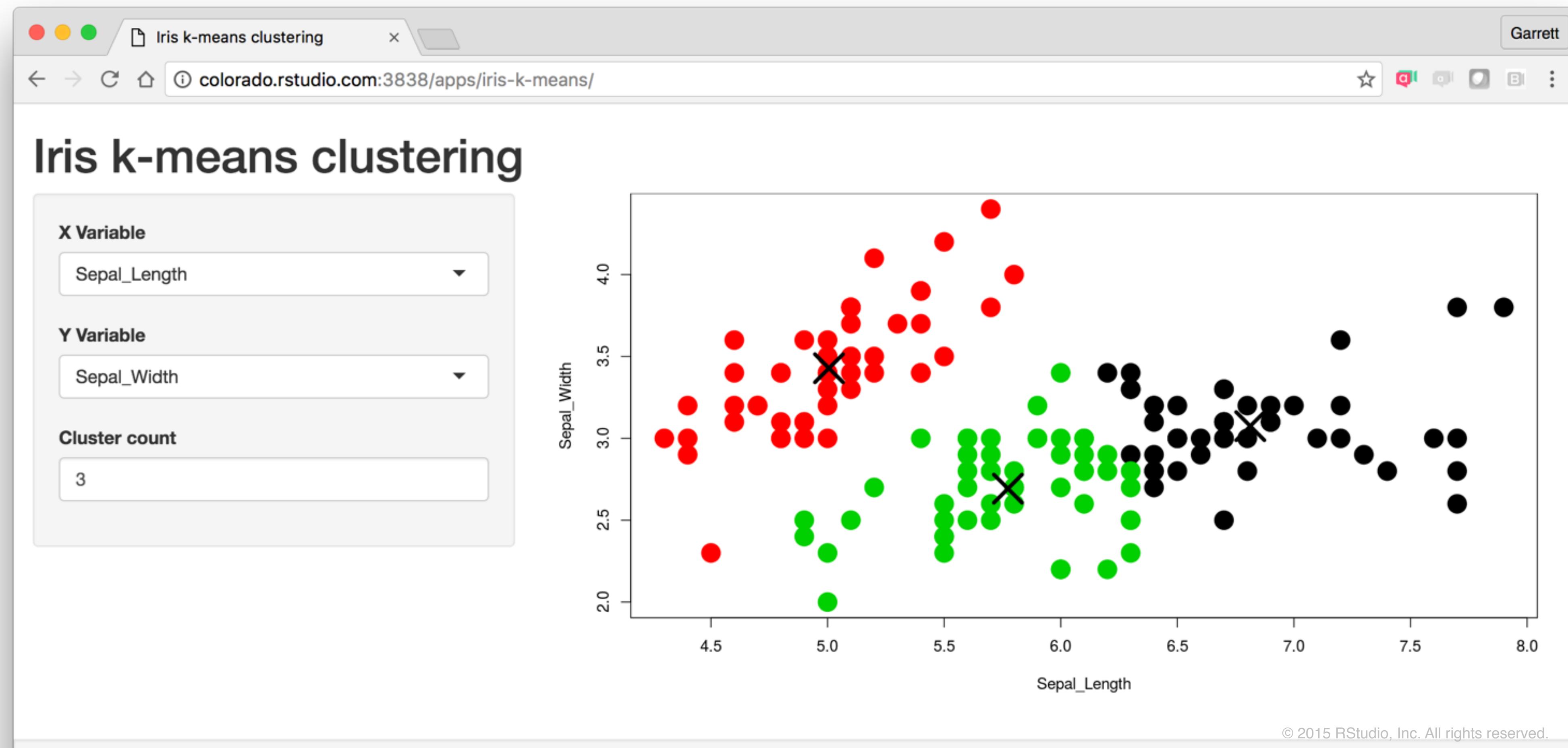
For the Titanic data, decision trees and random forests performed the best and had comparatively fast run times. See [results](#) for a detailed comparison.

ID	Function	Description	AUC Rank	Run time Rank
1	Random forest	ml_random_forest	1	3
2	Decision tree	ml_decision_tree	2	2
3	Gradient boosted tree	ml_gradient_boosted_trees	3	6
4	Logistic regression	ml_logistic_regression	4	4
5	Multilayer perceptron (neural net)	ml_multilayer_perceptron	5	5
6	Naive Bayes	ml_naive_bayes	6	1

Load the data

Iris K Means Clustering

<http://sparkdemo.rstudio.com/apps/iris-k-means/>



Diamonds Dashboard

sparkdemo.rstudio.com/dashboards/diamonds-explorer/

