

# R PROGRAMMING FOR FINANCIAL DATA

R/Finance 2013 Workshop  
May 17, 2013 Chicago, IL

Jeffrey A. Ryan [jeffrey.ryan@lemnica.com](mailto:jeffrey.ryan@lemnica.com)

# WHAT IS 'FINANCIAL DATA'

complex

unstructured

structured

shared resource

expensive

research dependent

production

big



## TWO MAIN TYPES

structured

ohlcv, earnings,  
TAQ, ...

unstructured

feeds, news,  
filings, ...

Reality: little choice in how you get it!

# MAPPING

The challenge is mapping from **raw data** sources into a format that is conducive to both **research** and **production** system. Possibly (re)used from different languages that R.



# MAPPING

Steps to success.

Understand your source data

Understand your production needs

Understand R structures

Create usable abstraction layers

# MAPPING

Steps to success.

Understand your source data

Understand your production needs

Understand R structures

Create usable abstraction layers



# MAPPING

Steps to success.

Understand your source data

Understand your production needs

Understand R structures

Create usable abstraction layers

# MAPPING

Steps to success.

Understand your source data

Understand your production needs

Understand R structures

Create usable abstraction layers

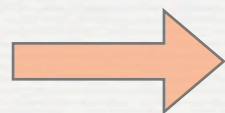


# MAPPING

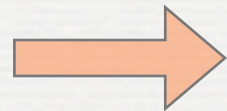
Steps to success.

Understand your source data

Understand your production needs



Understand R structures

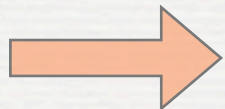


Create usable abstraction layers

Today we'll cover these **two** with examples

# MAPPING

Steps to success.



Understand your source data

Understand your production needs

Understand R structures

Create usable abstraction layers

But first, what **data** do you have?



# RAW DATA

Raw Text: SEC, News, Twitter

Database: Historical, Events, Earnings

APIs: Bloomberg, Reuters, Interactive Brokers

e.g.  
SQL                      CSV                      Messages

# RAW DATA

## Read Into R

Technically different talk.  
RT\_M?

e.g.

SQL

CSV

Messages



# R DATA STRUCTURES



Time

Data

Most of R programming in finance relies on time-based data. We need to understand this **very well**.

# R DATA STRUCTURES



Time



# R DATA STRUCTURES



Time

Date

~~Characters~~

POSIXct

POSIXlt

# R DATA STRUCTURES



Time

Date

**Date** objects in R  
allow for time-zone  
agnostic day  
representations.

**When to use:**  
Only care about days.  
i.e. not “time”



# R DATA STRUCTURES



Time

Date

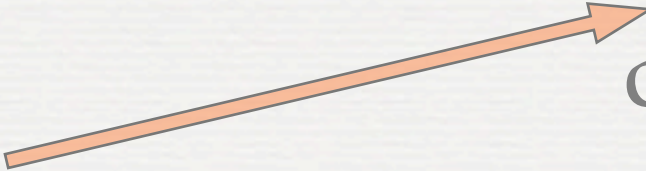
e.g.

```
as.Date("2013-05-17")
```

```
Sys.Date()
```

what's inside:

```
structure(15842,  
          class="Date")
```



number of days since 1970-01-01

# R DATA STRUCTURES



Time

POSIXct

**POSIXct** objects in R allow for date and times. Supports subseconds and time-zones.

**When to use:**  
Most often.



# R DATA STRUCTURES



Time

e.g.

```
as.POSIXct("2013-05-17")
```

```
Sys.time()
```

POSIXct

what's inside:

1368748800



number of seconds since 1970-01-01 in UTC\*

# R DATA STRUCTURES



Time

POSIXct

- watch TZ settings
- TZ is *machine* dependent
- use Sys.setenv(TZ=)
- *always* set a TZ

e.g.

```
Sys.setenv(TZ="UTC")
```



# R DATA STRUCTURES



Time

POSIXlt

**POSIXlt** objects in R allow for date and times. Supports subseconds and time-zones.

**When to use:**

need 'broken down' time.

# R DATA STRUCTURES



The diagram shows a vertical stack of two boxes. The top box is orange and labeled 'Time'. The bottom box is white with a light orange gradient and labeled 'POSIXlt'. An orange arrow points from the text 'similar to C language *time\_t* struct' at the bottom left towards the 'POSIXlt' box.

Time

POSIXlt

similar to C language *time\_t* struct

e.g.

```
as.POSIXlt("2013-05-17")
```

what's inside:

```
list(sec=0,  
      min=0L,  
      hour=0L,  
      mday=17L,  
      mon=4L,  
      year=113L,  
      wday=5L,  
      yday=136L,  
      isdst=0L)
```



# R DATA STRUCTURES



Time

POSIXlt

- same TZ care required
- very large object
- very slow

# R DATA STRUCTURES

vector

matrix

list

data.frame

environment



Data



# R DATA STRUCTURES

vector

matrix

list

data.frame

environment



Data

# R DATA STRUCTURES

vector

matrix

list

data.frame

environment



Data

*vector* contains atomic types (integers, doubles, ...)

*list* and *environment* contain objects (vectors, lists, ...)

*matrix* and *data.frames* are vectors and lists, respectively



# R DATA STRUCTURES

## Additional Useful Data Objects

`data.table`

`xts`

`mmap`

A fast, scalable, data.frame. Use it. Carefully.

# R DATA STRUCTURES

Additional Useful Data Objects

data.table

xts

mmap

High performance time-series class.



# R DATA STRUCTURES

Additional Useful Data Objects

data.table

xts

mmap

Memory mapped objects. Fast. Scalable.

# ENVIRONMENTS

The **ultimate** R object.



# ENVIRONMENTS

The **ultimate** R object.

Store all other objects

Pass by *reference*

Hashed  $O(1)$  lookup performance

# ENVIRONMENTS

The **ultimate** R object.

```
e <- new.env(hash=TRUE)
```

```
e$a <- 100
```

```
e$a
```

```
[1] 100
```



# ENVIRONMENTS

The **ultimate** R object.

```
e <- new.env(hash=TRUE)
```

```
e$a <- 100
```

```
e$a
```

```
[1] 100
```

We will use often!

# ABSTRACTIONS

Interfaces in R should *behave like R*

Minimize the learning curve

Keep arguments consistent

Design with your use case in mind - *not* your data!!!!



# CACHES

Plan your data bottlenecks

Private, shared, read-optimized. *fast*

High-performance custom *local* stores

# PUTTING IT ALL TOGETHER

Learn By Example!



# QUANTMOD

Abstraction Example

# getSymbols()

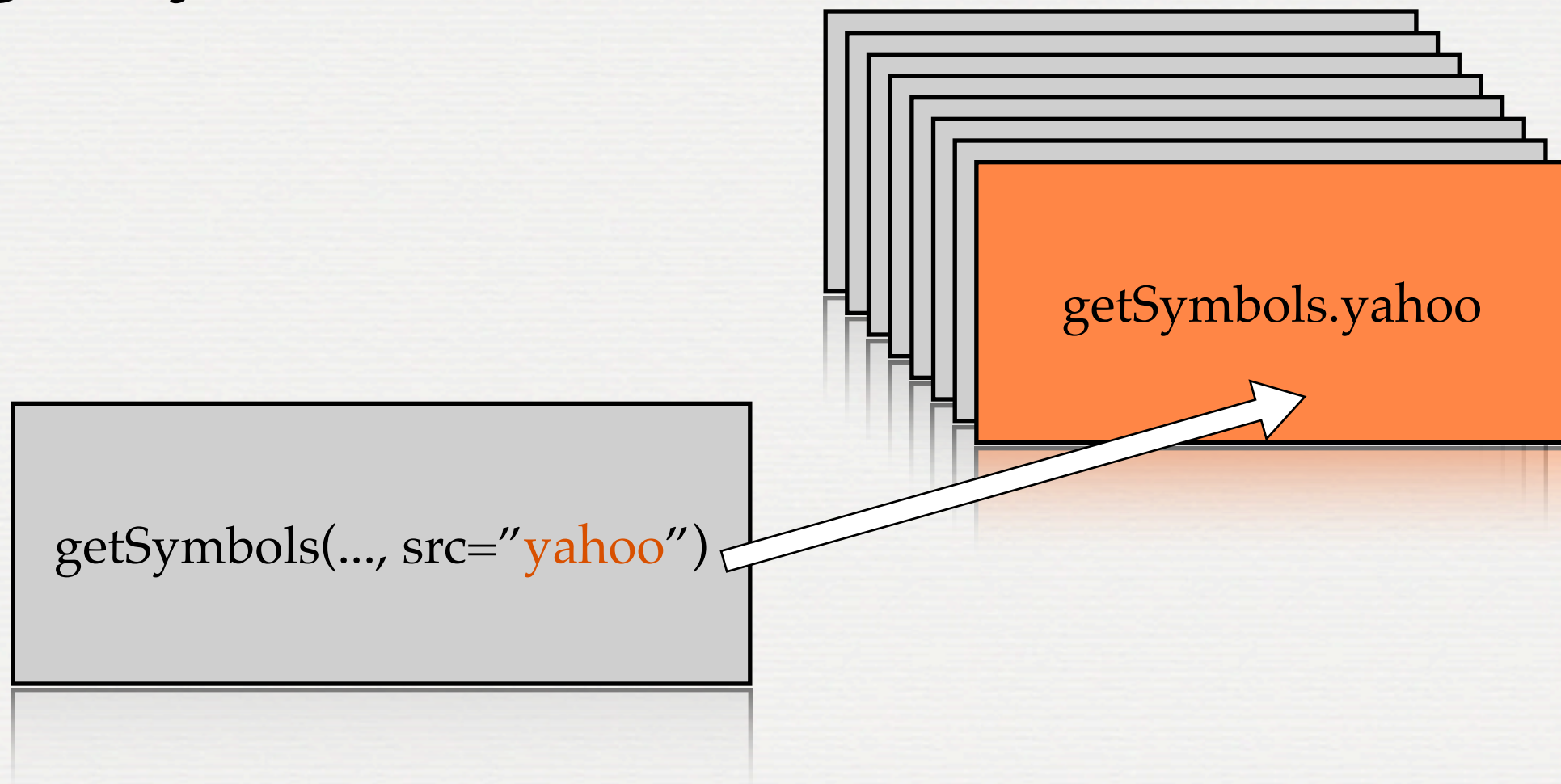
Designed to provide a **uniform interface** to various data sources, while maintaining a single entry point and **hiding the data access internals**.

*in other words...*

Make data management idiot-proof



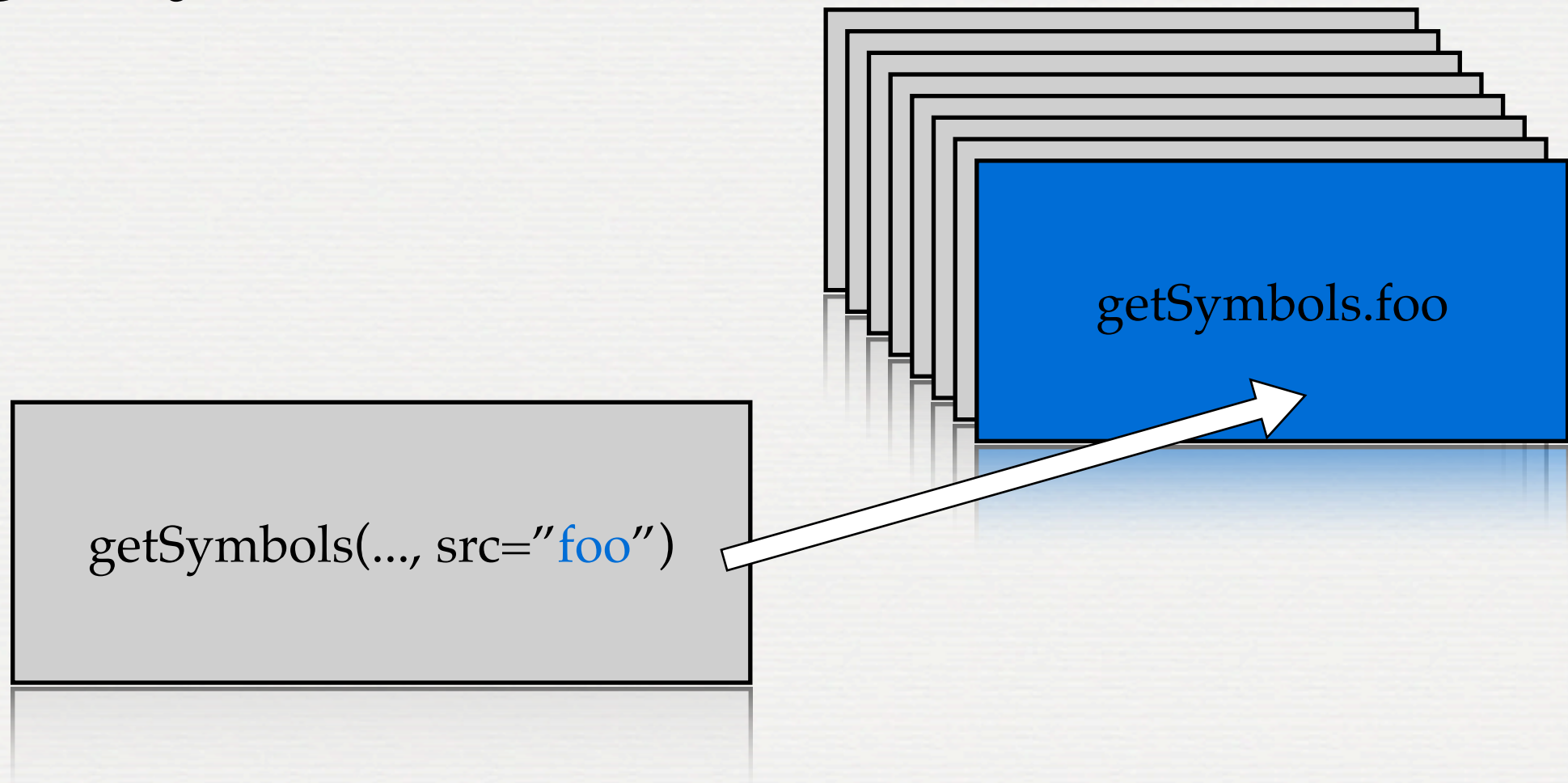
# getSymbols()



S3 'style' dispatch. Easy to extend.

.yahoo, .google, .rds, .csv ...

# getSymbols()



FUNCTION

```
getSymbols.foo <- function(...) {  
  ...  
}
```

CODE

```
SRC <- "foo"  
getSymbols(..., src=SRC)
```



# EQUITY EXAMPLE.

rds. attach. makeActiveBinding

# OPTIONS EXAMPLE

mmap + indexing



# 13F EXAMPLE

mmap struct

# PUTTING IT ALL TOGETHER

A recipe for data success:

1. What do you need the data for?
2. What sources do you use now?
3. Abstract - a la quantmod
4. Leverage firm / existing solutions
5. Build high performance caches



# Thanks!

