Algorithmic Trading with R

How to lose money without really trying.

Amis Consulting LLP

IT consultancy who reach the parts that other consultancies cannot reach.

- 1983-1997: Scientific computing and image processing.
- 1997-2003: Dotcom Boom (and bust) !!!
- 2003- Financial Systems.
- Currently involved in High Performance Computing and (*of course*) Big Data and well as all the other stuff.

Worked with a variety of technologies

- Languages (in anger):
  Fortran / C / Ada / Perl / Python / Lisp / Java / PHP /Groovy …
  … our ***GOTO*** languages remain C and Perl but ???
- Back-ends
  Unix (not just Linux) and Windows (so some .NET)
- Databases
  Both relational and the NoSQL stuff
- Moving into the cloud
  AWS: Map-reduce, (excited about) Redshift.

Then along came R …

- At Kings in late-2000's
- Interest was in HPC (mainly **CUDA**) applied to financial systems.
- Started using Matlab but was looking for a similar *type* package for personal/company usage .
- Gnu/Octave and R both fitted the bill, R won.
- One new option (-- *more later* --)

R in Finance

(My) Taxonomy of R financial packages

CRAN has a Finance view: http://cran.r-project.org/web/views/Finance.html

- Econometrics
- Rmetrics Group
- Quantmod
- "TITs"

A Couple of TITs

Algorithmic Trading

You will need:

- Data Feed
  Yahoo Financials, Pay for feeds, Platform API
- Analysis
  Python, C/C++, C#, R + … others
- Trade
  Metatrader, NinjaTrader, TradeStation, ThinkOrSwim …
- ***Cash !!!***

Real-time Trading (with IBrokers)

- ACL use Interactive Brokers (*not always in R*)
  https://www.interactivebrokers.co.uk/
- A feed to do real-time trading.
- Java Application (TWS) and a Web-Trader Interface
- Good documentation and phone support.
- API which can be accessed via C, Java, .Net, Excel
- Several *other* languages exist, including at least one R package.

Interactive Brokers – Trader Workstation

TWS Mosaic Panels

Forex trading

AAPL stock movement

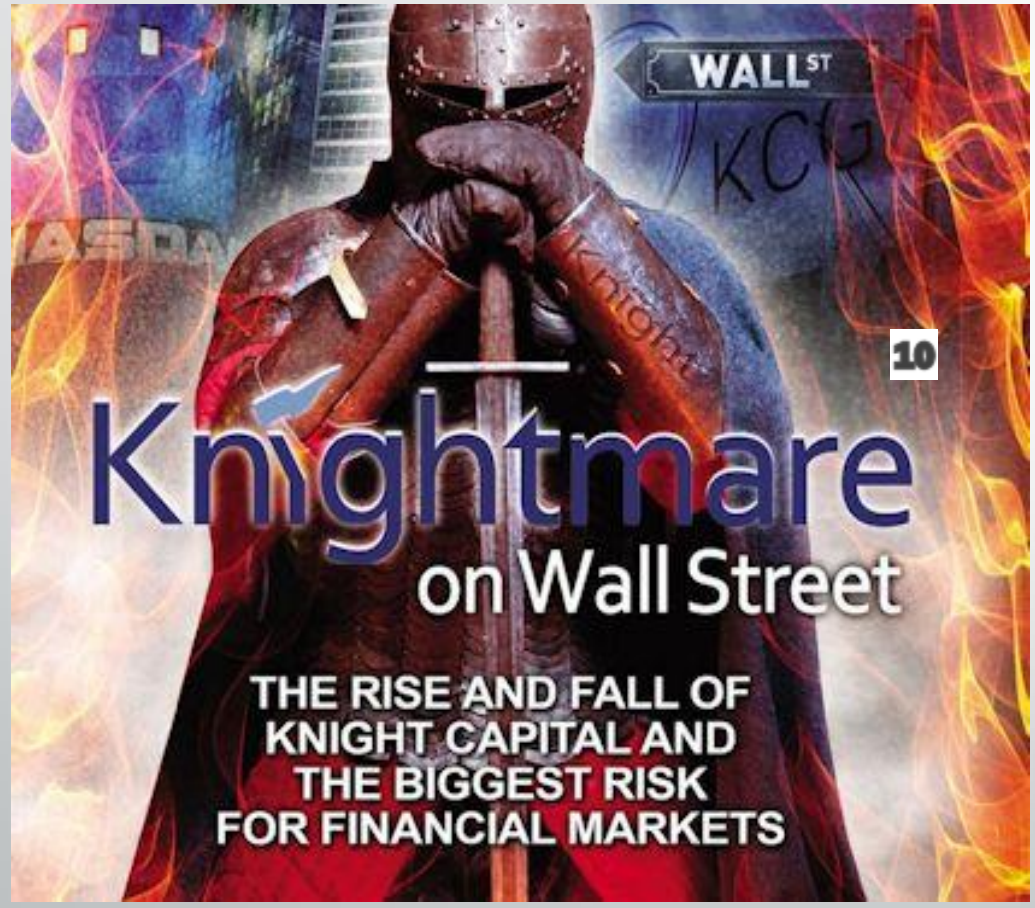
More on Algorithmic Trading

- Moving average
  Closing prices, Yahoo/Google Financials
- Historical data
  Back testing
- Real time
  Forex, Stocks, Options, Futures
- High frequency trading
  Black Arts, *not for us*

How do I start?

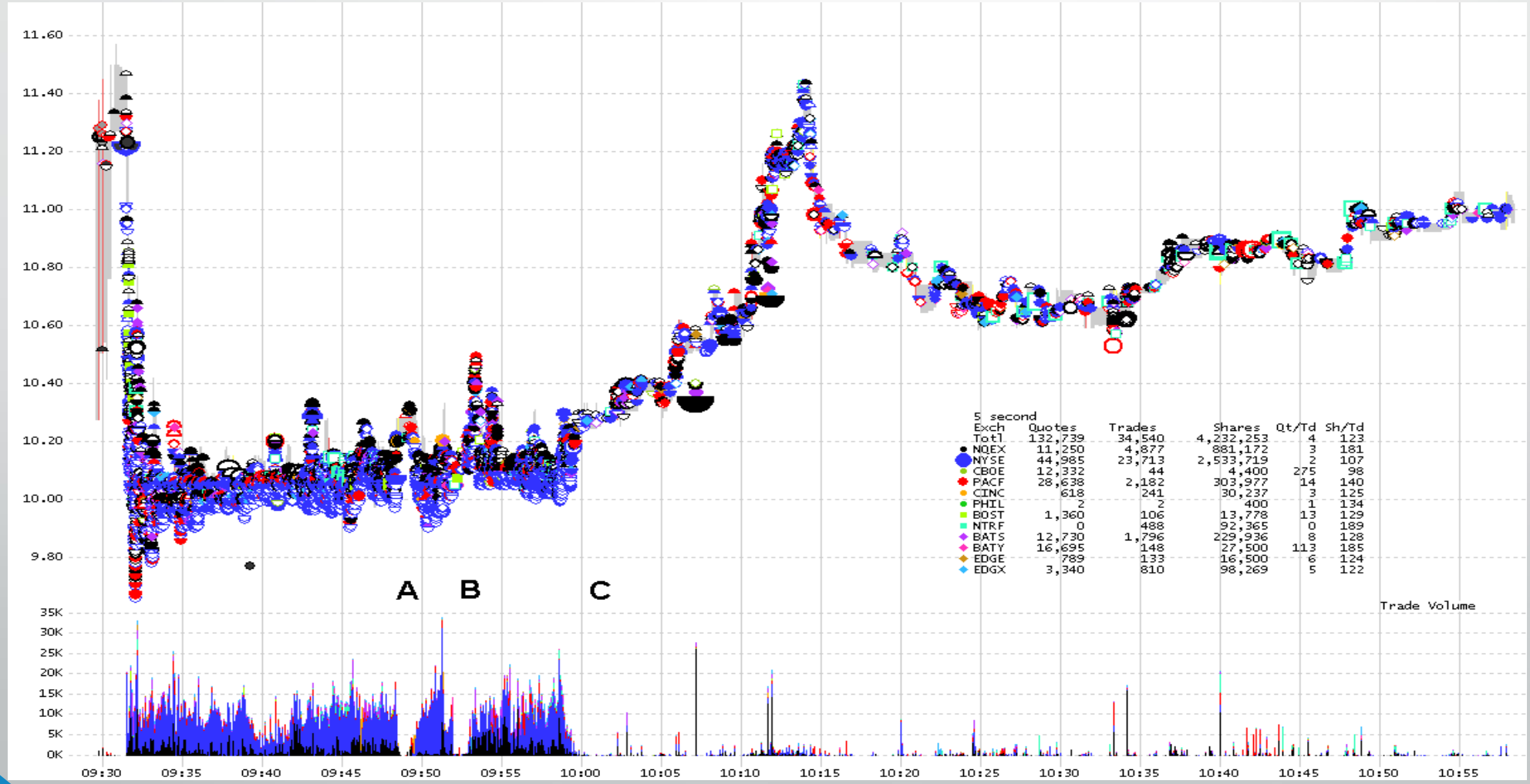
- Setup IB account.
  Read the documentation and look at the videos / webinars.
- Use the “Paper” Account .
  Forex, Stocks, Options …
- Use the TWS and/or Web-trader with real money.
- Formulate your algorithmic strategy.
- Program it.
- Goto the PUB.

How to Test Your New Market Making Software and Lose a Pile of Money, Fast

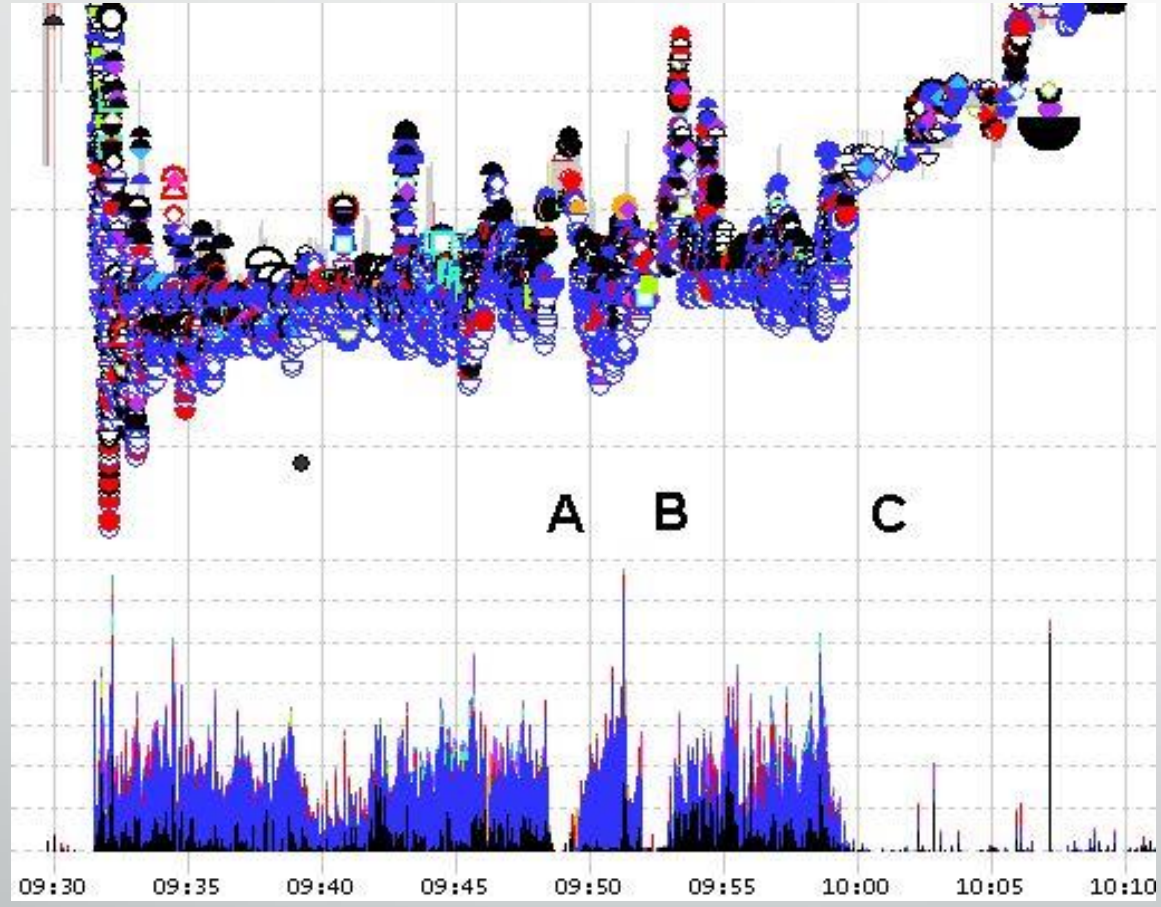


Wednesday, 1st August 2012

Nanex : 5 second interval chart showing all trades colour coded by exchange



Fatal First Forty minutes



5 second Exch	Quotes	Trades	Shares	Qt
Totl	132,739	34,540	4,232,253	
NQEX	11,250	4,877	881,172	
NYSE	44,985	23,713	2,533,719	
CBOE	12,332	44	4,400	
PACF	28,638	2,182	303,977	
CINC	618	241	30,237	
PHIL	2	2	400	
BOST	1,360	106	13,778	
NTRF	0	488	92,365	
BATS	12,730	1,796	229,936	
BATY	16,695	148	27,500	
EDGE	789	133	16,500	
EDGX	3,340	810	98,269	

http://www.nanex.net/aqck2/3522.html

Why Use R as a trading platform

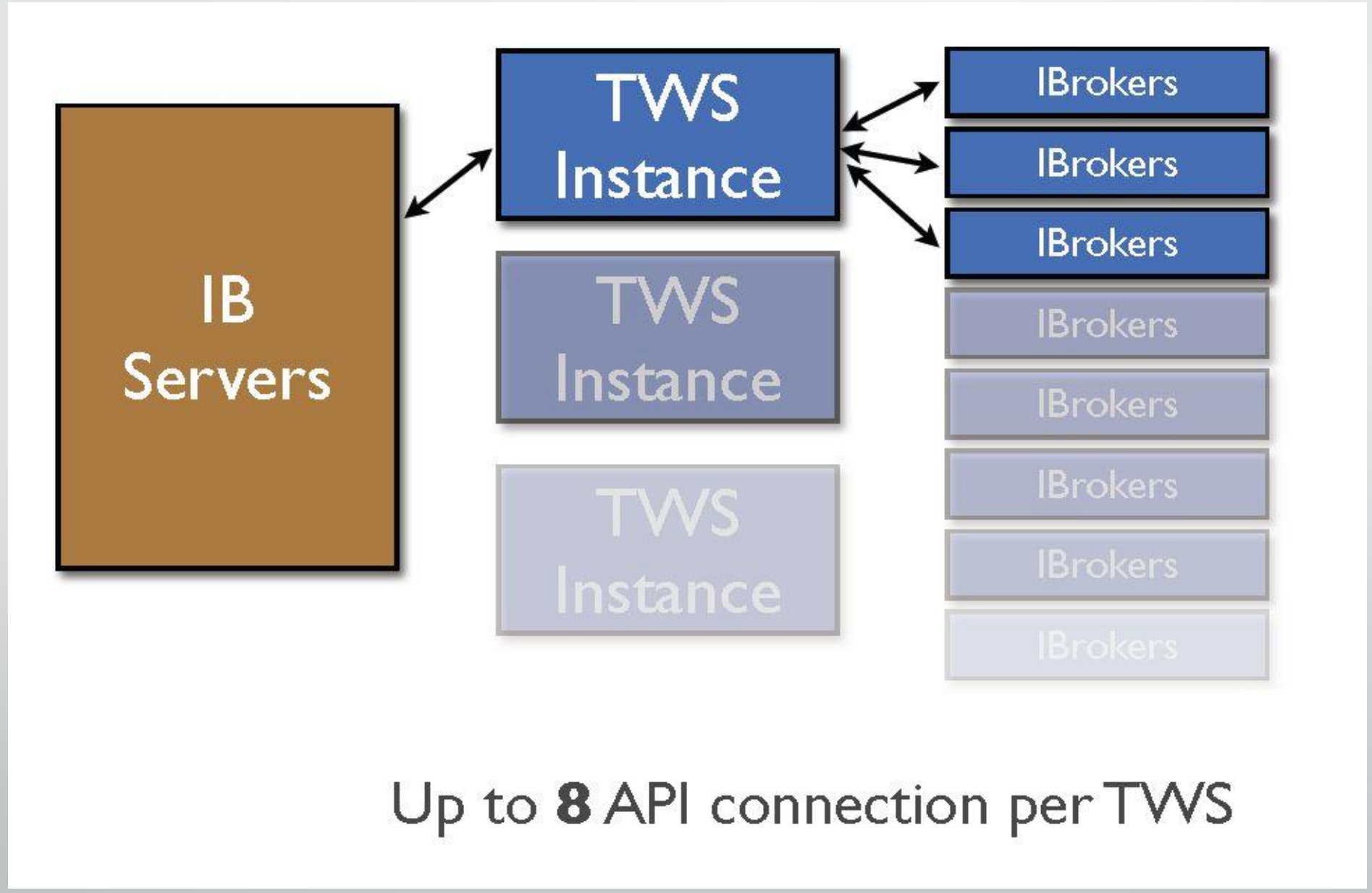
- Advantages
  - Flexible, powerful language
  - Best of class tool chain
  - Community, open source, cross-platform
- Disadvantages
  - Single threaded
  - Not as quick as C/C++

IBrokers: R package (1)

- Connections
  R has a native socket interface providing relatively high performance.
- Threads
  Single-threaded R makes true "*threading*" impossible.
- Data Persistence and Sharing
  Closures allow for single process data persistence among calls.
- Thorough-put
  Event REPL bounded by costly R loops
  Message structure dependent

IBrokers: R package (2)

- Authored by Jeff Ryan
  Member of the "Quant Mods", his first cut 2008, revised 2012
- Works via the Trader Workstation API
  Uses this for authentication and for the data feed
- Establishes a socket to socket local connection
- Written in almost native R, not a wrapper
- Revised version almost complete implementation of IB API



Up to **8** API connection per TWS

IBrokers : R Implementation

- Historical Data : reqHistoricalData
  Direct access to IB historic data

- Real Time Data : reqMktData, reqRealTimeBars
  Live market data – stocks, options, futures, FX

- Live Execution : placeOrder, cancelOrder, reqAccountUpdates
  Place and cancel orders
  View account information

Using IBrokers from R

A quick example of capturing data to disk:

```
library (IBrokers)
tws  <-  twsConnect()
aapl.csv  <-  file("AAPL.csv", open="w")
reqMktData (tws,
                    twsSTK("AAPL"),
                    eventWrapper = eWrapper.MktData.CSV(1),
                    file = aapl.csv)
close(aapl.csv)
close(tws)
```

Be familiar with the IB API documentation on a call as well as the R implementation (viz.)
https://www.interactivebrokers.com/en/software/api/apiguide/c/reqmktdata.htm

Real Time Data Model (1a)

```
> reqMktData(tws, twsFUT("NQ","GLOBEX","201309"))
<20130629 11:17:13.117602> id=1 symbol=NQ Volume: 0
<20130629 11:17:15.323245> id=1 symbol=NQ bidPrice: 1480.75 bidSize: 8
<20130629 11:17:15.324615> id=1 symbol=NQ askPrice: 1481.25 askSize: 19
<20130629 11:17:15.325706> id=1 symbol=NQ bidSize: 8
<20130629 11:17:15.326655> id=1 symbol=NQ askSize: 19
<20130629 11:17:15.327537> id=1 symbol=NQ lastPrice: 1480.75
<20130629 11:17:15.328335> id=1 symbol=NQ lastSize: 1
<20130629 11:17:15.329518> id=1 symbol=NQ lastTimestamp: 1246313805
```

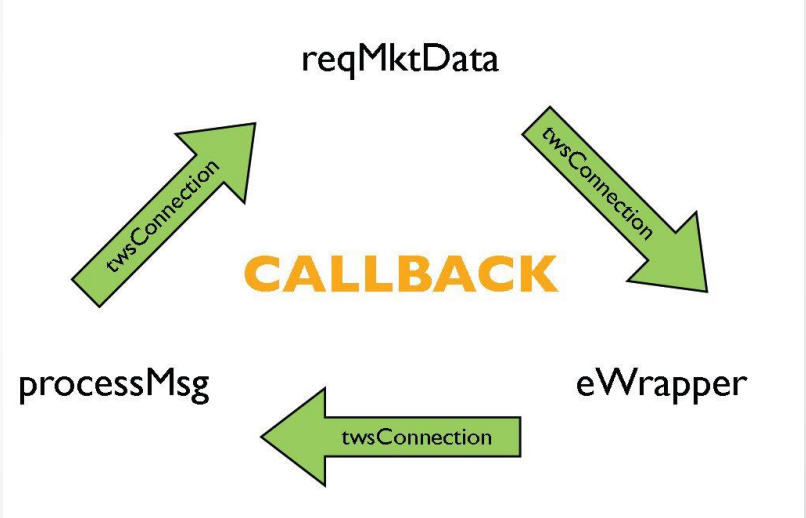
Real Time Data Model (1b)

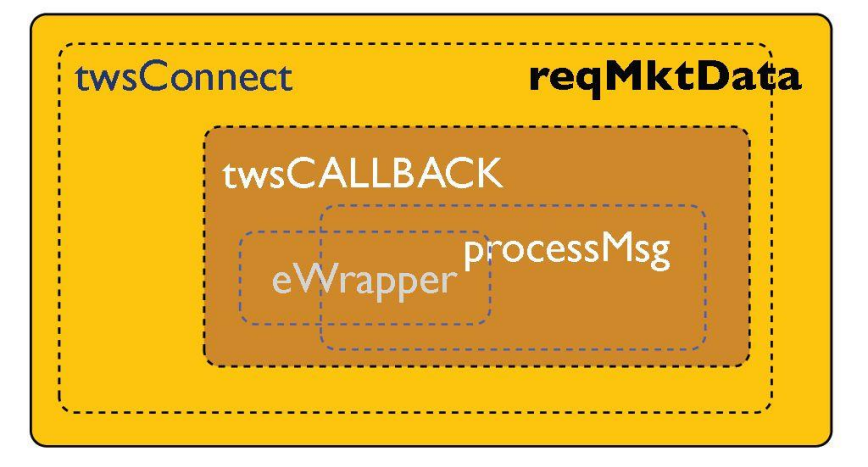
```
> reqMktData(tws, list(twsSTK("MSFT"),twsSTK("AAPL")))
<20131029 13:34:55.649897> id=1  symbol=MSFT  Volume: 622549
<20131029 13:34:55.652436> id=1  symbol=MSFT  highPrice: 24.03
<20131029 13:34:55.653531> id=1  symbol=MSFT  lowPrice: 23.55
<20131029 13:34:55.656226> id=1  symbol=MSFT  shortable: 3.0
<20131029 13:34:55.657129> id=1  symbol=MSFT  bidPrice: 23.89 bidSize: 2
<20131029 13:34:55.658781> id=1  symbol=MSFT  bidSize: 2
<20131029 13:34:55.659603> id=1  symbol=MSFT  askPrice: 23.9 askSize: 30
<20131029 13:34:55.660695> id=1  symbol=MSFT  askSize: 30
<20131029 13:34:55.850867> id=2  symbol=AAPL  bidPrice: 141.82 bidSize: 1
<20131029 13:34:55.852082> id=2  symbol=AAPL  askPrice: 141.9 askSize: 6
<20131029 13:34:55.853202> id=2  symbol=AAPL  lastPrice: 141.95
<20131029 13:34:55.854040> id=2  symbol=AAPL  bidSize: 1
<20131029 13:34:55.854956> id=2  symbol=AAPL  askSize: 6
```

Real Time Data Model (2)

```
reqMktData ( tws,
             Contract = twsSTK("QQQQ"),
             eventWrapper = eWrapper(TRUE),
             timeStamp=NULL)
```

```
1 5 1 6 36.75 0 0
1 5 1 7 36.12 0 0
45 5 1 49 0.0
45 5 1 46 3.0
1 5 1 1 36.48 3 1
2 5 1 0 3
2 5 1 2 36.49 80 1
2 5 1 3 80
2 5 1 8 978607
```





Callback routine

```
> twsCALLBACK
function (twsCon, eWrapper, timestamp, file, playback = 1, ...)
{
    if (missing(eWrapper))
        eWrapper <- eWrapper() con <- twsCon[[l]]
        if (inherits(twsCon, "twsPlayback")) {

        }
    }
   else {
        while (TRUE) {
            socketSelect(list(con), FALSE, NULL)
           curMsg <- readBin(con, character(), 1)
            if ( !is.null(timestamp) ) {
                processMsg(curMsg, con, eWrapper, format(Sys.time(), timestamp), file, ...)
            }
        }
    }
}
```

Process Message

```
> processMsg
function (curMsg, con, eWrapper, timestamp, file, ...)
{
   if (curMsg == .twsIncomingMSG$TICK_PRICE)  {
       msg <- readBin(con, character(), 6)
       eWrapper$tickPrice(curMsg, msg, timestamp, file, ...)
   }
  else if (curMsg == .twsIncomingMSG$TICK_SIZE) {
       msg <- readBin(con, character(), 4)
       eWrapper$tickSize(curMsg, msg, timestamp, file, ...)
   }
}
```

... # Dispatch to custom *eWrapper* method

eWrapper

Define custom message handling functions quickly and easily

```
> str(eWrapper())
List of 37
$ .Data :<environment: 0x307cbf0>
$ get.Data :function (x)
$ assign.Data :function (x, value)
$ remove.Data :function (x)
$ tickPrice :function (curMsg, msg, timestamp, file, ...)
$ tickSize :function (curMsg, msg, timestamp, file, ...)
$ tickOptionComputation :function (curMsg, msg, timestamp, file, ...)
$ tickGeneric :function (curMsg, msg, timestamp, file, ...)
$ tickString :function (curMsg, msg, timestamp, file, ...)
$ tickEFP :function (curMsg, msg, timestamp, file, ...)
$ orderStatus :function (curMsg, msg, timestamp, file, ...)
$ errorMessage :function (curMsg, msg, timestamp, file, ...)
$ openOrder :function (curMsg, msg, timestamp, file, ...)
$ openOrderEnd :function (curMsg, msg, timestamp, file, ...)
$ updateAccountValue :function (curMsg, msg, timestamp, file, ...)
$ updatePortfolio :function (curMsg, msg, timestamp, file, ...)
$ updateAccountTime :function (curMsg, msg, timestamp, file, ...)
$ accountDownloadEnd :function (curMsg, msg, timestamp, file, ...)
$ nextValidId :function (curMsg, msg, timestamp, file, ...)
$ contractDetails :function (curMsg, msg, timestamp, file, ...)
$ bondContractDetails :function (curMsg, msg, timestamp, file, ...)
$ contractDetailsEnd :function (curMsg, msg, timestamp, file, ...)
$ execDetails :function (curMsg, msg, timestamp, file, ...)
$ updateMktDepth :function (curMsg, msg, timestamp, file, ...)
$ updateMktDepthL2 :function (curMsg, msg, timestamp, file, ...)
$ updateNewsBulletin :function (curMsg, msg, timestamp, file, ...)
$ managedAccounts :function (curMsg, msg, timestamp, file, ...)
$ receiveFA :function (curMsg, msg, timestamp, file, ...)
$ historicalData :function (curMsg, msg, timestamp, file, ...)
$ scannerParameters :function (curMsg, msg, timestamp, file, ...)
$ scannerData :function (curMsg, msg, timestamp, file, ...)
$ scannerDataEnd :function (curMsg, msg, timestamp, file, ...)
$ realtimeBars :function (curMsg, msg, timestamp, file, ...)
$ currentTime :function (curMsg, msg, timestamp, file, ...)
$ fundamentalData :function (curMsg, msg, timestamp, file, ...)
$ deltaNeutralValidation:function (curMsg, msg, timestamp, file, ...)
$ tickSnapshotEnd :function (curMsg, msg, timestamp, file, ...)
```

Real Time Data Model (3)

```
twsOC <- twsConnect(2)                                                    # Our order connection
ocWrapper <- eWrapper(TRUE)
traded <- FALSE
while (TRUE) {
        cons <- socketSelect(list(con, twsOC[[1]]), FALSE, 0.01)
        if(cons[1]) {                                                     ## Data ##
                curMsg <- readBin(con, character(), 1)
                if ( !is.null(timestamp)) {
                        processMsg(curMsg, con, eWrapper, format(Sys.time(), timestamp), file, ...)
                }
                else {
                        processMsg(curMsg, con, eWrapper, timestamp, file, ...)
                }
        }
        else if(cons[2]) {                                                ## Order messages ##
                curMsg <- readBin(twsOC[[1]], character(), 1)
                if (!is.null(timestamp)) {
                        processMsg(curMsg, twsOC[[1]], ocWrapper, format(Sys.time(), timestamp), file, ...)
                }
                else {
                        processMsg(curMsg, twsOC[[1]], ocWrapper, timestamp, file, ...)
                }
        }
        curBID <- as.numeric(eWrapper$.Data$data[[1]][3])                 ## Trade Logic ##
        if( !traded && !is.na(curBID) && curBID > 140.00) {
                IBrokers:::.placeOrder(twsOC, twsSTK("AAPL"), twsOrder(1053, "BUY", "10", "MKT"))
                traded <- TRUE
        }
}
```

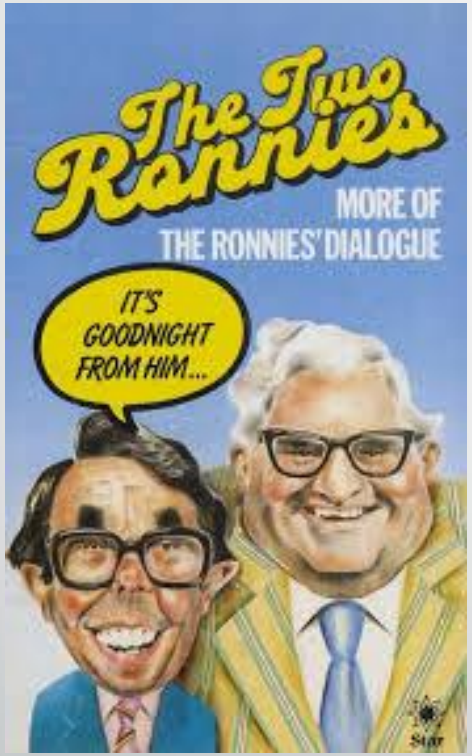
Conclusions

- R can be used from real-time trading.
- Do some online trading first.
- If you still have any cash left -- try programming your strategy.

- The IBrokers package is a good example of what can be achieved natively in R.

- There is a new kid(-*ess*) on the block: ***Julia***

And it's goodnight from him



- Malcolm Sherrington
  Amis Consulting LLP

- malcolm@amisllp.com
- http://www.amisllp.com
- mobile: 07931 287043