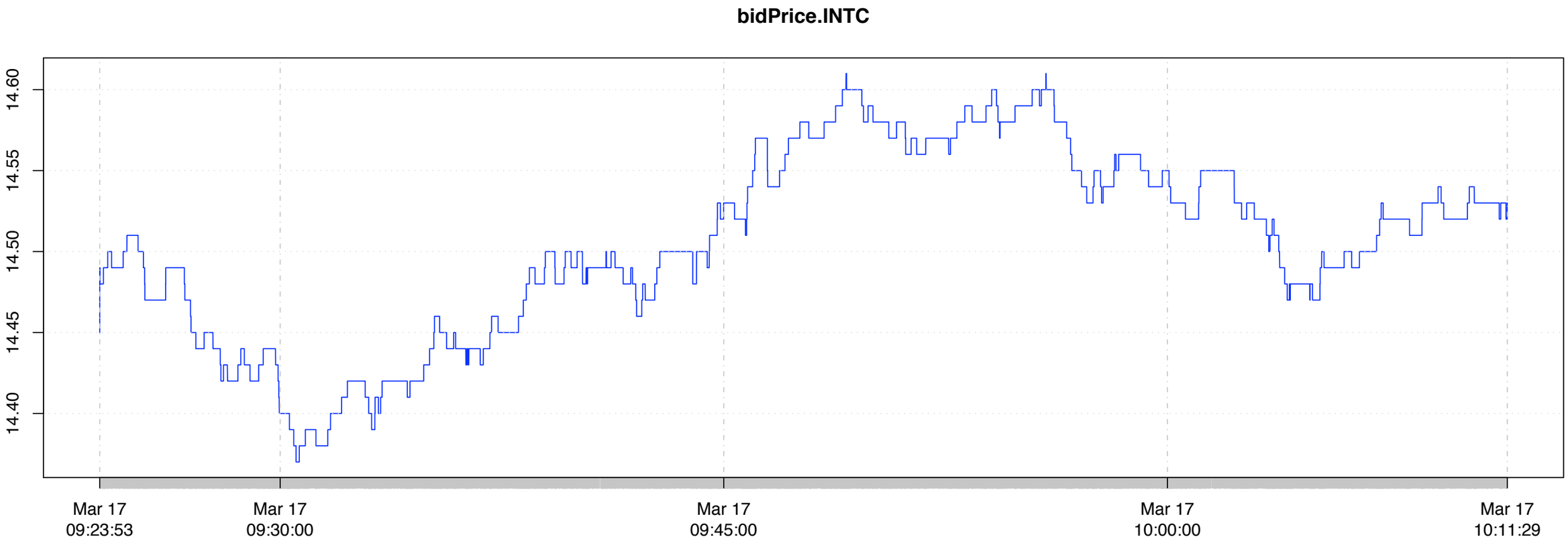
Real Time Trading in R



Jeffrey A. Ryan

jeffrey.ryan@insightalgo.com

Presented R/Rmetrics 2009

Meielisalp, Leissigen, Switzerland

30 June, 2009

Overview

1. Why use R?
2. Technical challenges and limitations
3. Working example: IBrokers

Why R?

Why use R as a real-time trading platform?

Why R?

Why use R as a real-time trading platform?

Advantages

Flexible, powerful language

Best of class tool-chain

Community

Open-source

Cross-platform

Fast

CRAN

Disadvantages

Not as fast as C

Single-threaded

Limited deployments

Why R?

Why use R as a real-time trading platform?

Advantages

Flexible, powerful language

Best of class tool-chain

Community

Open-source

Cross-platform

Fast

CRAN

Disadvantages

Not as fast as C

Single-threaded

Limited live deployments

Technical Details

Real time data in R

Connections

Threads

Data Persistence & Sharing

Throughput

Technical Details

Connections

COM, sockets, or external code?

R has a native socket interface which provides for relatively high performance, as well as assuring *maximum portability*.

Java/C/C++ connections are reasonable options.

Technical Details

Threads

Single-threaded R makes true "threading" impossible.

Most trading rules can be distilled into logic that can be executed linearly.

If one R process isn't sufficient, multiple processes can be used to simulate threaded behavior.

Technical Details

Data Persistence and Sharing

Closures allow for single process data persistence among calls. (*eWrappers*)

External sharing between R processes can be facilitated with files or shared memory, as implemented in the CRAN package bigmemory by Kane and Emerson

Technical Details

Throughput

Event loop is bound by costly
R loops.

Highly message structure dependent.

Restructure memory so messages are aggregated
to minimize looping.

Upper limit ~10k messages per second
per process.

IBrokers

Unofficial R API to Interactive Brokers

```
<20090629 19:56:03.089257> id=1 symbol=QQQQ Volume: 978607
<20090629 19:56:03.091655> id=1 symbol=QQQQ highPrice: 36.75
<20090629 19:56:03.092825> id=1 symbol=QQQQ lowPrice: 36.12
<20090629 19:56:03.093818> id=1 symbol=QQQQ <default generic> 5 1 49 0.0
<20090629 19:56:03.095080> id=1 symbol=QQQQ shortable: 3.0
<20090629 19:56:03.095994> id=1 symbol=QQQQ bidPrice: 36.48  bidSize: 3
<20090629 19:56:03.097180> id=1 symbol=QQQQ bidSize: 3
<20090629 19:56:03.098035> id=1 symbol=QQQQ askPrice: 36.49  askSize: 80
<20090629 19:56:03.099086> id=1 symbol=QQQQ askSize: 80
<20090629 19:56:03.356604> id=1 symbol=QQQQ lastTimestamp: 1246319893
<20090629 19:56:03.358407> id=1 symbol=QQQQ lastPrice: 36.48
<20090629 19:56:03.359341> id=1 symbol=QQQQ lastSize: 50
<20090629 19:56:03.360228> id=1 symbol=QQQQ Volume: 978821
<20090629 19:56:03.361109> id=1 symbol=QQQQ highPrice: 36.75
<20090629 19:56:03.361938> id=1 symbol=QQQQ lowPrice: 36.12
<20090629 19:56:03.362782> id=1 symbol=QQQQ closePrice: 36.37
<20090629 19:56:03.363686> id=1 symbol=QQQQ openPrice: 36.47
<20090629 19:56:03.364639> id=1 symbol=QQQQ bidPrice: -1.0  bidSize: 0
<20090629 19:56:03.365668> id=1 symbol=QQQQ askPrice: -1.0  askSize: 0
<20090629 19:56:03.366758> id=1 symbol=QQQQ bidSize: 0
<20090629 19:56:03.367678> id=1 symbol=QQQQ askSize: 0
<20090629 19:56:03.368586> id=1 symbol=QQQQ optionHistoricalVol: 0.22668955978280078 NA
<20090629 19:56:03.369604> id=1 symbol=QQQQ optionImpliedVol: 0.24415731504169574
<20090629 19:56:03.370514> id=1 symbol=QQQQ optionCallVolume: 325860
<20090629 19:56:03.371377> id=1 symbol=QQQQ optionPutVolume: 261927
<20090629 19:56:03.372288> id=1 symbol=QQQQ averageVolume: 1351176
<20090629 19:56:03.373135> id=1 symbol=QQQQ 13-week High: 37.18610001
<20090629 19:56:03.374000> id=1 symbol=QQQQ 13-week Low: 29.5760994
<20090629 19:56:03.374881> id=1 symbol=QQQQ 26-week High: 37.18610001
<20090629 19:56:03.375764> id=1 symbol=QQQQ 26-week Low: 25.53730011
<20090629 19:56:03.376640> id=1 symbol=QQQQ 52-week High: 48.4056015
<20090629 19:56:03.377621> id=1 symbol=QQQQ 52-week Low: 24.9137001
<20090629 19:56:03.378510> id=1 symbol=QQQQ optionCallOpenInterest: 2321743
<20090629 19:56:03.379441> id=1 symbol=QQQQ optionPutOpenInterest: 1973620
```

IBrokers

DISCLAIMER

This software is in no way affiliated, endorsed, or approved by Interactive Brokers or any of its affiliates. It comes with absolutely no warranty and should not be used in actual trading unless the user can read and understand the source.

IBrokers

Design Requirements

Interactive Brokers

R Implementation

Future Direction

IBrokers

Design Requirements

Interactive Brokers

R Implementation

Future Direction

IBrokers

Design Requirements

API Consistency

Exploit the power of R

Minimize the weaknesses of R

Simplify interface to speed prototyping

IBrokers

Design Requirements

API Consistency

Exploit the power of R

Minimize the weaknesses of R

Simplify interface to speed prototyping

IBrokers

Design Requirements

API Consistency

Exploit the power of R

Minimize the weaknesses of R

Simplify interface to speed prototyping

IBrokers

Design Requirements

API Consistency

Exploit the power of R

Minimize the weaknesses of R

Simplify interface to speed prototyping

IBrokers

Design Requirements

Interactive Brokers

R Implementation

Future Direction

IBrokers

Interactive Brokers

PROS

- Comprehensive API
- Low commissions
- Good executions
- Cross-platform

CONS

- No tick data
- Not scalable
- API documentation (?)
- Java-based

IBrokers

Interactive Brokers

PROS

Comprehensive API

Low commissions

Good executions

Cross-platform

CONS

No tick data

Not scalable

API documentation (?)

Java-based

IBrokers

Interactive Brokers

PROS	CONS
Comprehensive API	No tick data
Low commissions	Not scalable
Good executions	API documentation (?)
Cross-platform	Java-based

IBrokers

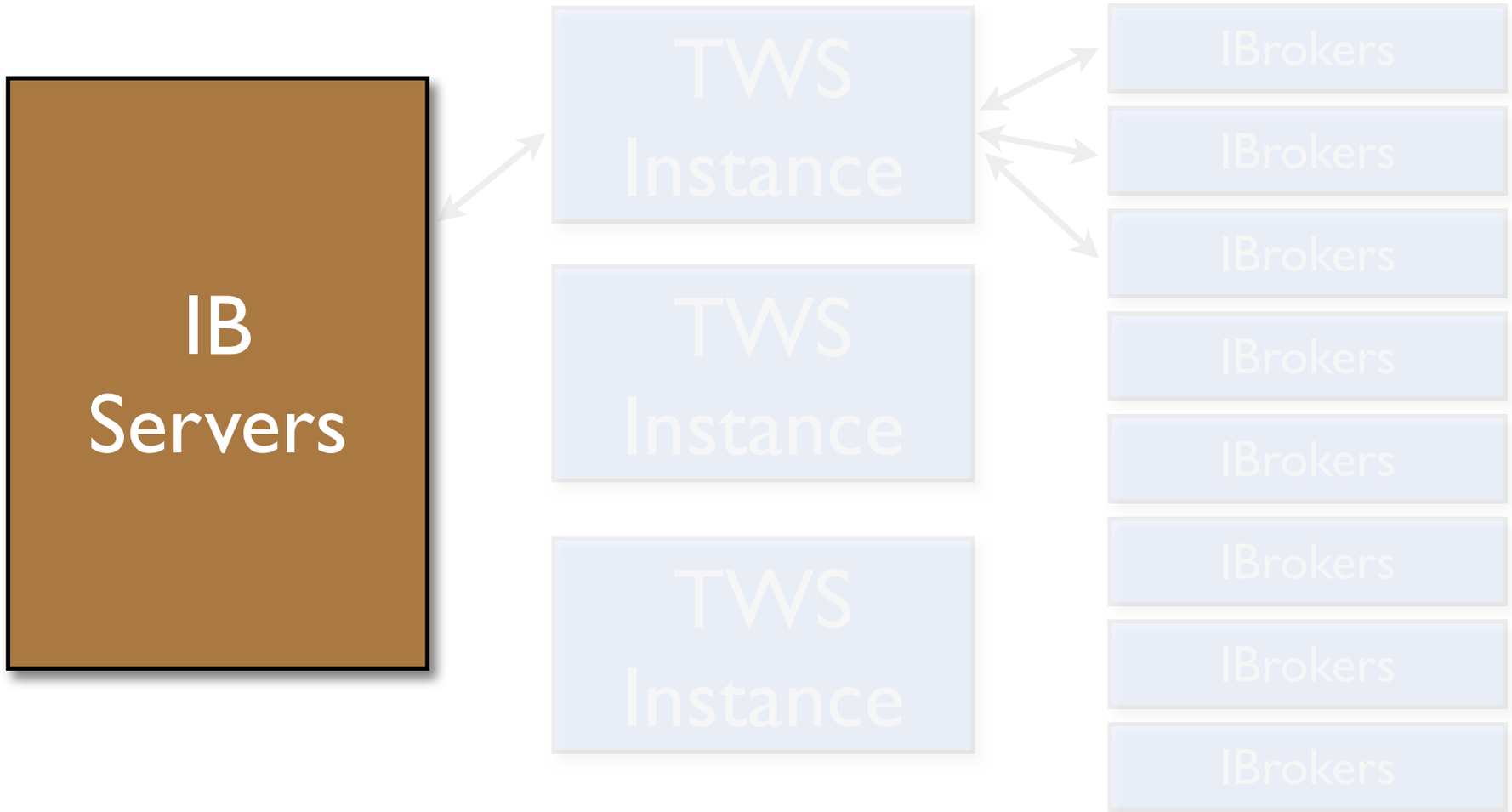
Interactive Brokers

PROS	CONS
Comprehensive API	No tick data
Low commissions	Not scalable
Good executions	API documentation (?)
Cross-platform	Java-based ;)

API interfaces: Java, Perl, Python, C, C++, DDE, R
(IBrokers & RIB)

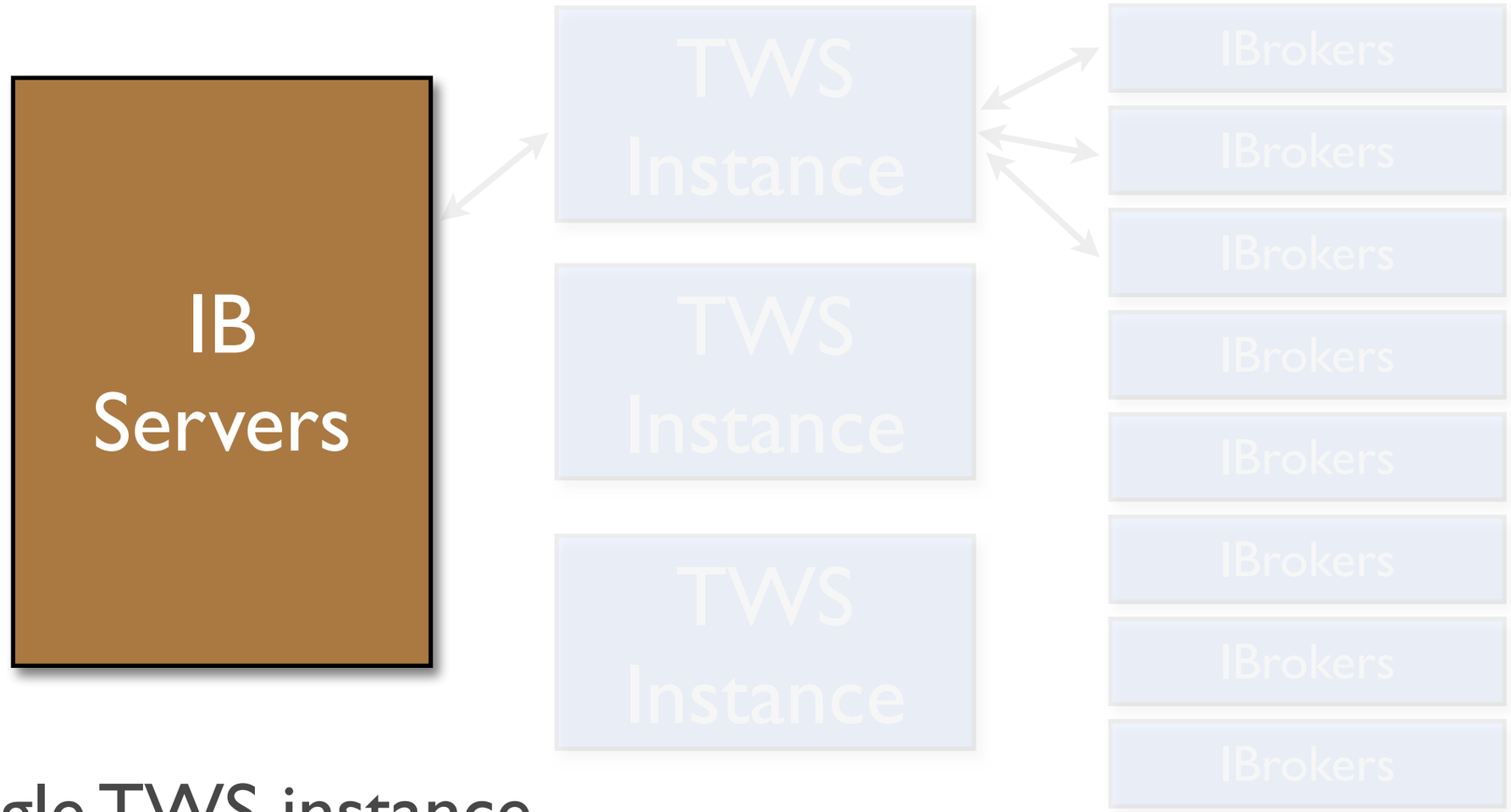
IBrokers

Interactive Brokers



IBrokers

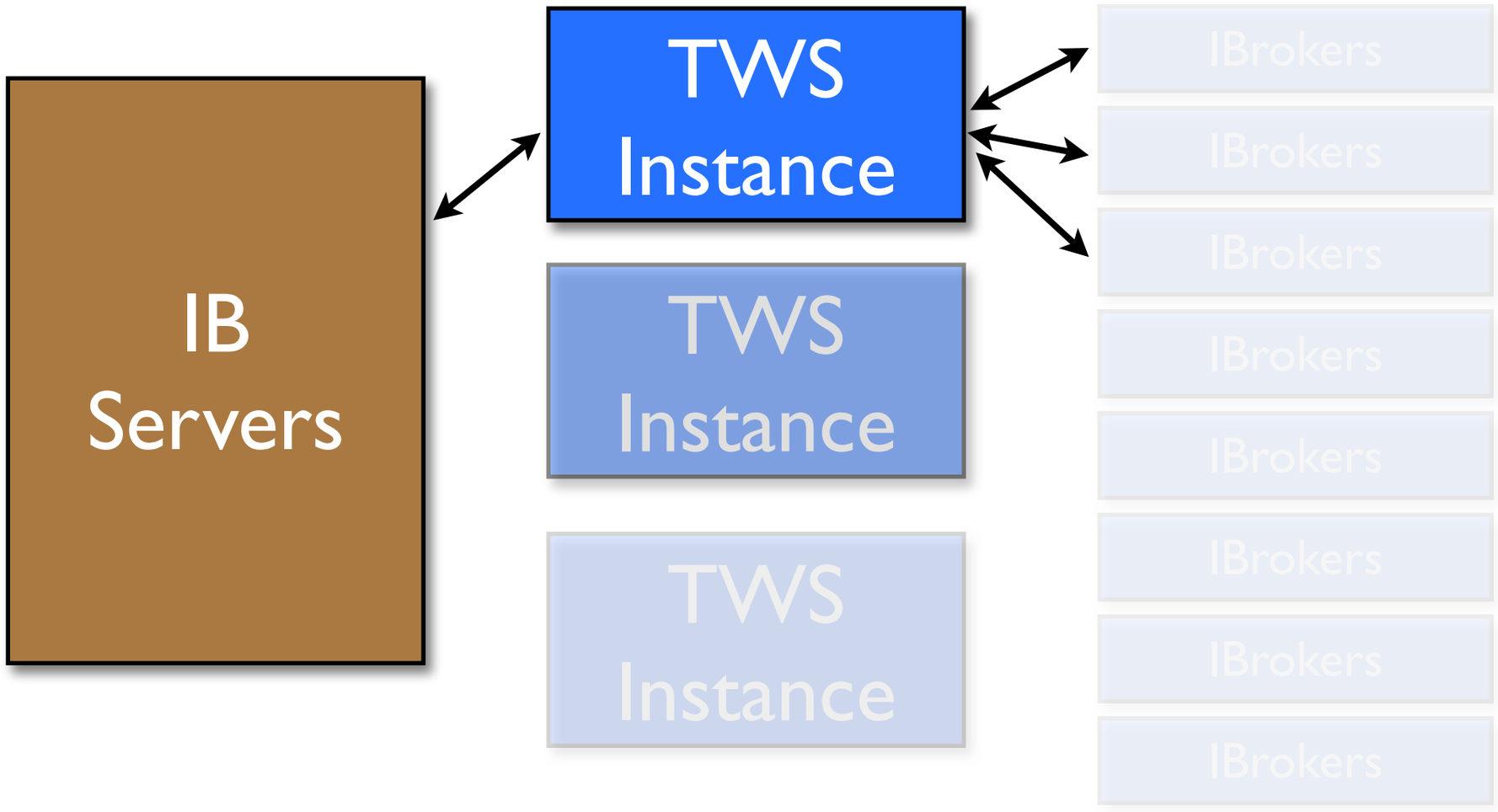
Interactive Brokers



Single TWS instance connects via internet or direct connection to IB

IBrokers

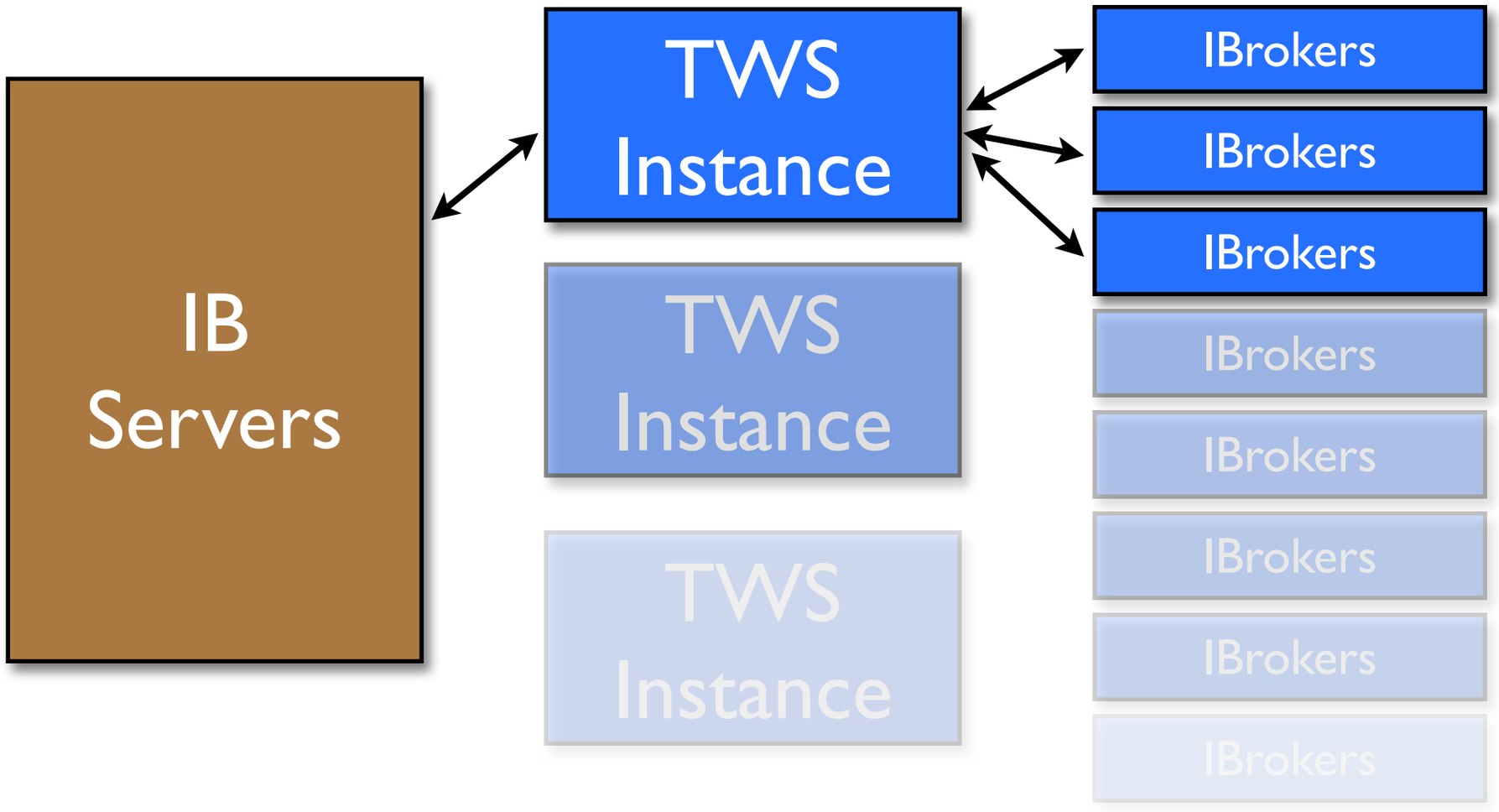
Interactive Brokers



Multiple TWS instances can
run on one machine.

IBrokers

Interactive Brokers



Up to **8** API connection per TWS

IBrokers

Design Requirements

Interactive Brokers

R Implementation

Future Direction

IBrokers

R Implementation

IBrokers builds a *new* API in R based on the official Java API distributed by Interactive Brokers.

IBrokers

R Implementation

IBrokers builds a *new* API in R based on the official Java API distributed by Interactive Brokers.

Socket-based written entirely in R
Historical, Real Time, and Order Execution
Additional interface *niceties* using R

IBrokers

R Implementation

Historical Data reqHistoricalData

Direct access to IB historical data from R

Real Time Data reqMktData, reqRealTimeBars

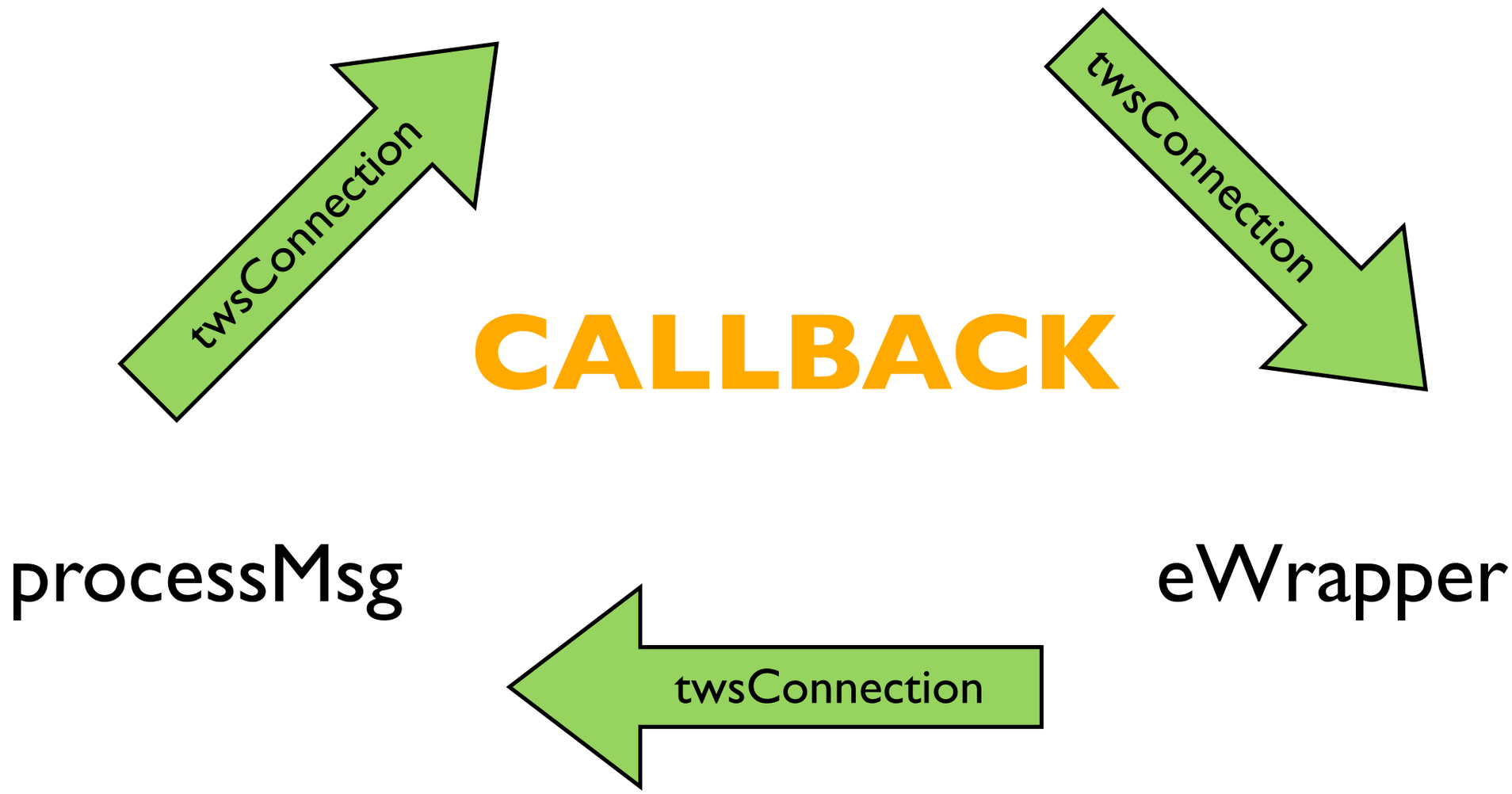
Live market data - stock, option, future, FX.

Live Execution placeOrder, cancelOrder, reqAccountUpdates

Place and cancel orders, view account information

IBrokers

Real Time Data Model



IBrokers

Real Time Data Model

IBrokers

Real Time Data Model

Make a connection to the TWS client

twsConnect

IBrokers

Real Time Data Model

Request market data on one or more instruments

IBrokers

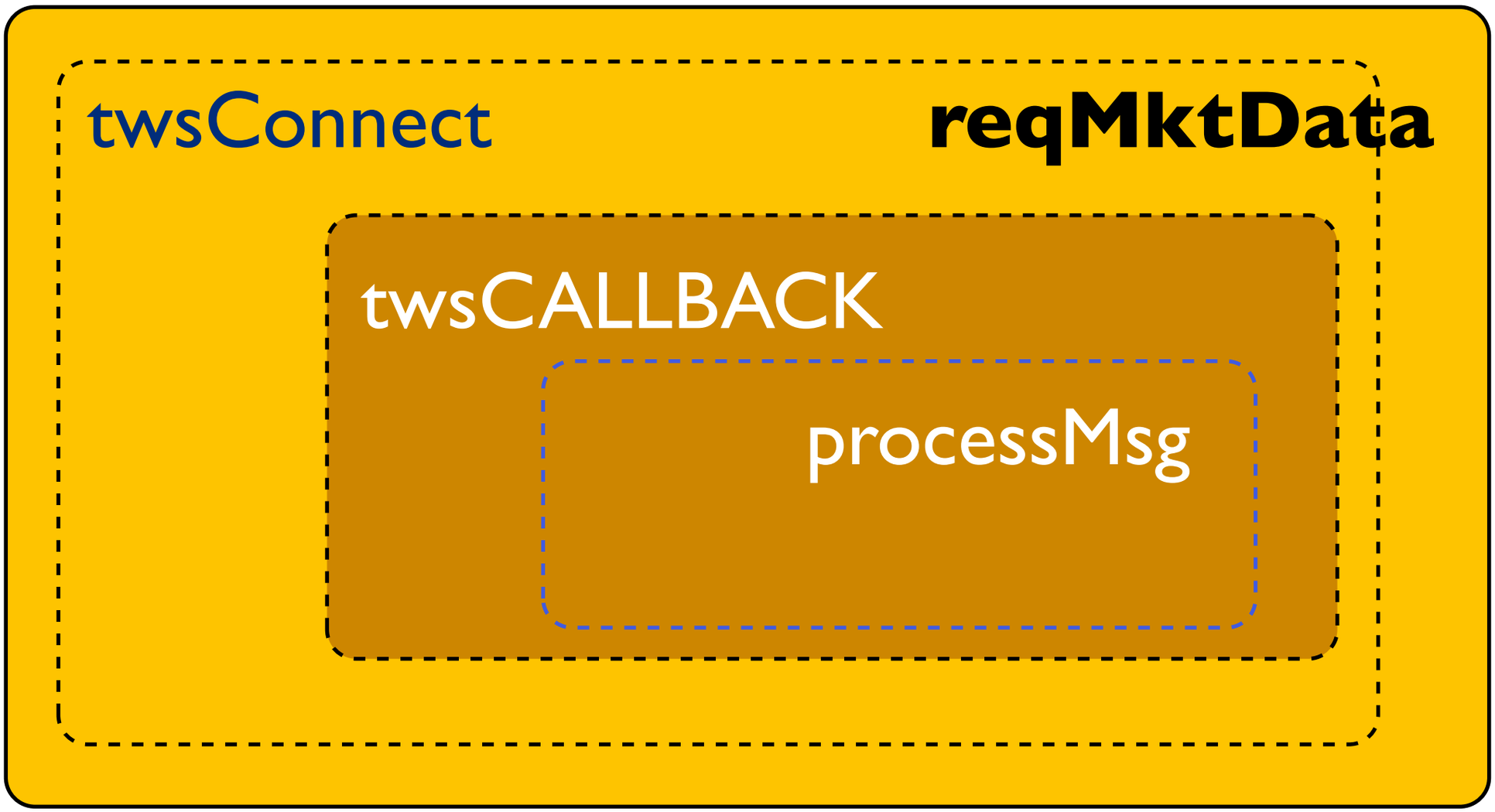
Real Time Data Model

Internally dispatch to main loop

IBrokers

Real Time Data Model

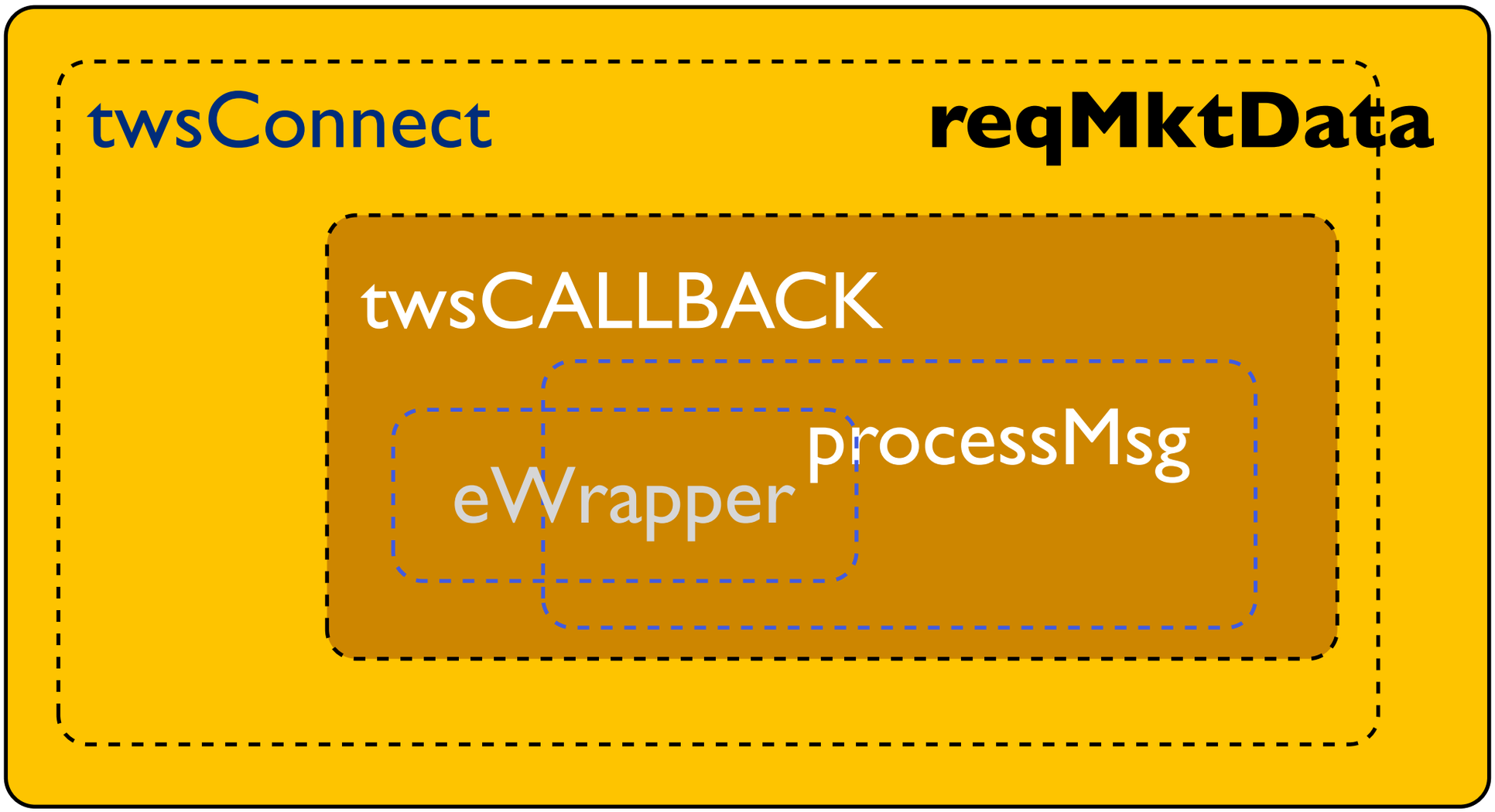
Process each message individually



IBrokers

Real Time Data Model

Use eWrapper embedded callback methods



IBrokers

Real Time Data Model

Example:

Request Real Time Market Data

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))
```

Create connection to TWS

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))
```

twsConnection

twsContract (Future)

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))

<20090629 17:17:13.117602> id=1 symbol=NQ Volume: 0
<20090629 17:17:15.323245> id=1 symbol=NQ bidPrice: 1480.75  bidSize: 8
<20090629 17:17:15.324615> id=1 symbol=NQ askPrice: 1481.25  askSize: 19
<20090629 17:17:15.325706> id=1 symbol=NQ bidSize: 8
<20090629 17:17:15.326655> id=1 symbol=NQ askSize: 19
<20090629 17:17:15.327537> id=1 symbol=NQ lastPrice: 1480.75
<20090629 17:17:15.328335> id=1 symbol=NQ lastSize: 1
<20090629 17:17:15.329518> id=1 symbol=NQ lastTimestamp: 1246313805

...
```

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))

<20090629 17:17:13.117602> id=1 symbol=NQ Volume: 0
<20090629 17:17:15.323245> id=1 symbol=NQ bidPrice: 1480.75  bidSize: 8
<20090629 17:17:15.324615> id=1 symbol=NQ askPrice: 1481.25  askSize: 19
<20090629 17:17:15.325706> id=1 symbol=NQ bidSize: 8
<20090629 17:17:15.326655> id=1 symbol=NQ askSize: 19
<20090629 17:17:15.327537> id=1 symbol=NQ lastPrice: 1480.75
<20090629 17:17:15.328335> id=1 symbol=NQ lastSize: 1
<20090629 17:17:15.329518> id=1 symbol=NQ lastTimestamp: 1246313805
```

R timestamp

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))

<20090629 17:17:13.117602> id=1 symbol=NQ Volume: 0
<20090629 17:17:15.323245> id=1 symbol=NQ bidPrice: 1480.75  bidSize: 8
<20090629 17:17:15.324615> id=1 symbol=NQ askPrice: 1481.25  askSize: 19
<20090629 17:17:15.325706> id=1 symbol=NQ bidSize: 8
<20090629 17:17:15.326655> id=1 symbol=NQ askSize: 19
<20090629 17:17:15.327537> id=1 symbol=NQ lastPrice: 1480.75
<20090629 17:17:15.328335> id=1 symbol=NQ lastSize: 1
<20090629 17:17:15.329518> id=1 symbol=NQ lastTimestamp: 1246313805
```

request ID

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))

<20090629 17:17:13.117602> id=1 symbol=NQ Volume: 0
<20090629 17:17:15.323245> id=1 symbol=NQ bidPrice: 1480.75  bidSize: 8
<20090629 17:17:15.324615> id=1 symbol=NQ askPrice: 1481.25  askSize: 19
<20090629 17:17:15.325706> id=1 symbol=NQ bidSize: 8
<20090629 17:17:15.326655> id=1 symbol=NQ askSize: 19
<20090629 17:17:15.327537> id=1 symbol=NQ lastPrice: 1480.75
<20090629 17:17:15.328335> id=1 symbol=NQ lastSize: 1
<20090629 17:17:15.329518> id=1 symbol=NQ lastTimestamp: 1246313805
```

Contract Symbol

IBrokers

Real Time Data Model

```
> tws <- twsConnect(1)

> reqMktData(tws, twsFUT("NQ","GLOBEX","200909"))

<20090629 17:17:13.117602> id=1  symbol=NQ Volume: 0
<20090629 17:17:15.323245> id=1  symbol=NQ bidPrice: 1480.75   bidSize: 8
<20090629 17:17:15.324615> id=1  symbol=NQ askPrice: 1481.25   askSize: 19
<20090629 17:17:15.325706> id=1  symbol=NQ bidSize: 8
<20090629 17:17:15.326655> id=1  symbol=NQ askSize: 19
<20090629 17:17:15.327537> id=1  symbol=NQ lastPrice: 1480.75
<20090629 17:17:15.328335> id=1  symbol=NQ lastSize: 1
<20090629 17:17:15.329518> id=1  symbol=NQ lastTimestamp: 1246313805
```

Data

IBrokers

Real Time Data Model

```
> reqMktData(tws, list(twsSTK("MSFT"),twsSTK("AAPL")))
```

Request multiple instruments

IBrokers

Real Time Data Model

```
> reqMktData(tws, list(twsSTK("MSFT"),twsSTK("AAPL")))
```

Request multiple instruments

IBrokers

Real Time Data Model

```
> reqMktData(tws, list(twsSTK("MSFT"),twsSTK("AAPL")))

<20090629 17:34:55.649897> id=1 symbol=MSFT Volume: 622549
<20090629 17:34:55.652436> id=1 symbol=MSFT highPrice: 24.03
<20090629 17:34:55.653531> id=1 symbol=MSFT lowPrice: 23.55
<20090629 17:34:55.654973> id=1 symbol=MSFT <default generic> 5 1 49 0.0
<20090629 17:34:55.656226> id=1 symbol=MSFT shortable: 3.0
<20090629 17:34:55.657129> id=1 symbol=MSFT bidPrice: 23.89  bidSize: 2
<20090629 17:34:55.658781> id=1 symbol=MSFT bidSize: 2
<20090629 17:34:55.659603> id=1 symbol=MSFT askPrice: 23.9  askSize: 30
<20090629 17:34:55.660695> id=1 symbol=MSFT askSize: 30
<20090629 17:34:55.850867> id=2 symbol=AAPL bidPrice: 141.82  bidSize: 1
<20090629 17:34:55.852082> id=2 symbol=AAPL askPrice: 141.9  askSize: 6
<20090629 17:34:55.853202> id=2 symbol=AAPL lastPrice: 141.95
<20090629 17:34:55.854040> id=2 symbol=AAPL bidSize: 1
<20090629 17:34:55.854956> id=2 symbol=AAPL askSize: 6
```

IBrokers

Real Time Data Model

What is happening *inside* IBrokers?

IBrokers

Real Time Data Model

The TWS raw message.

IBrokers

Real Time Data Model

```
> 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
1 5 1 2 36.49 80 1
2 5 1 3 80
2 5 1 8 978607
```

IBrokers

Real Time Data Model

```
> 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
1 5 1 2 36.49 80 1
2 5 1 3 80
2 5 1 8 978607
```

Methods to handle messages

IBrokers

Real Time Data Model

```
> 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
1 5 1 2 36.49 80 1
2 5 1 3 80
2 5 1 8 978607
```

1 5 1 1 36.48 3 1

IBrokers

Real Time Data Model

```
> 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
1 5 1 2 36.49 80 1
2 5 1 3 80
2 5 1 8 978607
```

1 5 1 1 36.48 3 1

Header

(new tickPrice update)

IBrokers

Real Time Data Model

```
> 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
1 5 1 2 36.49 80 1
2 5 1 3 80
2 5 1 8 978607
```

1 5 1 1 36.48 3 1

Body

IBrokers

Real Time Data Model

So *how* is the data being processed?

IBrokers

Real Time Data Model

```
> twsCALLBACK
function (twsCon, eWrapper, timestamp, file, playback = 1, ...)
{
    if (missing(eWrapper))
        eWrapper <- eWrapper()
    con <- twsCon[[1]]
    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, ...)
            }
            else {
                processMsg(curMsg, con, eWrapper, timestamp,
                  file, ...)
            }
        }
}
```

CALLBACK

IBrokers

Real Time Data Model

```
> twsCALLBACK
function (twsCon, eWrapper, timestamp, file, playback = 1, ...)
{
    if (missing(eWrapper))
        eWrapper <- eWrapper()
    con <- twsCon[[1]]
    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, ...)
                }
                else {
                processMsg(curMsg, con, eWrapper, timestamp,
                  file, ...)
                }
        }
}
```

Wait on connection

IBrokers

Real Time Data Model

```
> twsCALLBACK
function (twsCon, eWrapper, timestamp, file, playback = 1, ...)
{
    if (missing(eWrapper))
        eWrapper <- eWrapper()
    con <- twsCon[[1]]
    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, ...)
            }
            else {
                processMsg(curMsg, con, eWrapper, timestamp,
                  file, ...)
            }
        }
}
```

Read header

IBrokers

Real Time Data Model

```
> twsCALLBACK
function (twsCon, eWrapper, timestamp, file, playback = 1, ...)
{
    if (missing(eWrapper))
        eWrapper <- eWrapper()
    con <- twsCon[[1]]
    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, ...)
            }
            else {
                processMsg(curMsg, con, eWrapper, timestamp,
                  file, ...)
            }
        }
}
```

Process message

IBrokers

Real Time Data Model

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, ...)
    }
...
...
...
}
```

IBrokers

Real Time Data Model

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, ...)
    }
...
...
...
}
```

New tickPrice message

IBrokers

Real Time Data Model

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, ...)
    }
...
...
...
}
```

Read fixed size message from connection

IBrokers

Real Time Data Model

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

IBrokers

Real Time Data Model

eWrapper

```
> 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, ...)
```

IBrokers

Real Time Data Model

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, ...)
```

IBrokers

Real Time Data Model

Putting it all together:

Separate data and order connection,
persistent data storage,
automated execution.

IBrokers

Real Time Data Model

New CALLBACK

```
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]) {
  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, ...)
  }
}             # TRADE LOGIC HERE
curBID <- as.numeric(eWrapper$.Data$data[[1]][3])
if(!traded && !is.na(curBID) && curBID > 141.00) {
  IBrokers:::.placeOrder(twsOC, twsSTK("AAPL"), twsOrder(1053, "BUY", "10", "MKT"))
  traded <- TRUE
}
```

IBrokers

Real Time Data Model

New CALLBACK

order connection

data messages

order messages

trade logic

```
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]) {
  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, ...)
  }
}             # TRADE LOGIC HERE
curBID <- as.numeric(eWrapper$.Data$data[[1]][3])
if(!traded && !is.na(curBID) && curBID > 141.00) {
  IBrokers:::.placeOrder(twsOC, twsSTK("AAPL"), twsOrder(1053, "BUY", "10", "MKT"))
  traded <- TRUE
}
```

IBrokers

Real Time Data Model

Future?

IBrokers

Real Time Data Model

Secondary data feeds (possible now)

Trade logic extraction: plug and play

External data pre-processing engine

Data sharing across processes

Generalize API as vendor neutral