

# The **esperr** R package

Bryan Lewis<sup>1</sup>   Michael J. Kane<sup>2</sup>

<sup>1</sup><http://illposed.net>

<sup>2</sup>Yale University, Department of Statistics

April 16, 2010

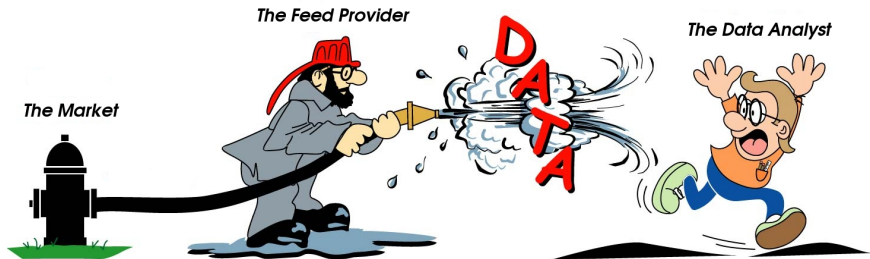
# Outline

- 1 Motivation: Analyzing Stock Market Feeds in R
- 2 Complex Event Processing with Esper
- 3 **esperr** for Event-based Analytics in R
- 4 Conclusions

# Stock Market Feeds

- Stock ticker information (price, volume, etc.) for a set of companies (MSFT, GOOG, etc.)
- Received asynchronously
- High frequency
- Combination of information for multiple companies at a high frequency results in high volume

# Like Drinking from a Firehose



## Can't we do this in R?

```
while (1)
{
  tickInfo <- GetTickInfo(stockSymbols)
  analysisResults <- TickAnalysis(tickInfo)
  MakeTrades(analysisResults)
}
```

# This doesn't work very well.

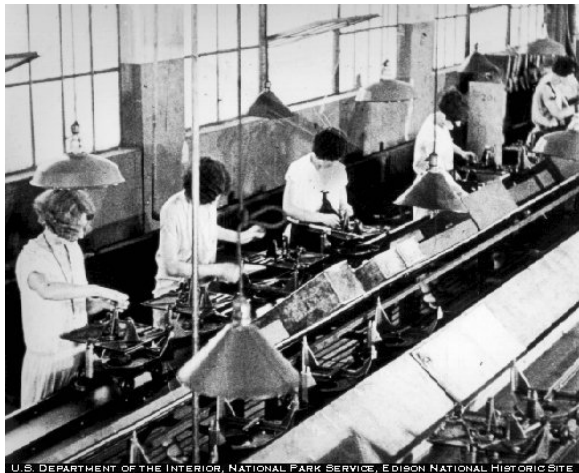
- 1 Ticks are not received asynchronously
  - Wait until tick information for all symbols is received
  - Ticks are dropped during analysis and trade
- 2 The loop is executed sequentially
  - Only one function executes at a time
  - Want to perform analysis while waiting for new ticks

# Problem 1: R doesn't do callbacks

We resort to polling.



## Problem 2: R doesn't do pipeline concurrency



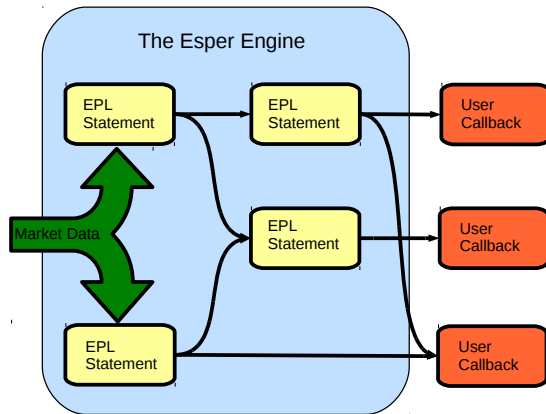
# What is Esper?

- Esper is a Java library for high-throughput/low latency event stream processing.
  - Events are immutable structured data objects (XML or JavaBean)
  - An event stream is a sequence of events
- The Esper library provides an SQL-like “Event Processing Language” (EPL).
  - EPL statements map input events to an output.
  - Maps can filter and perform simple processing operations.

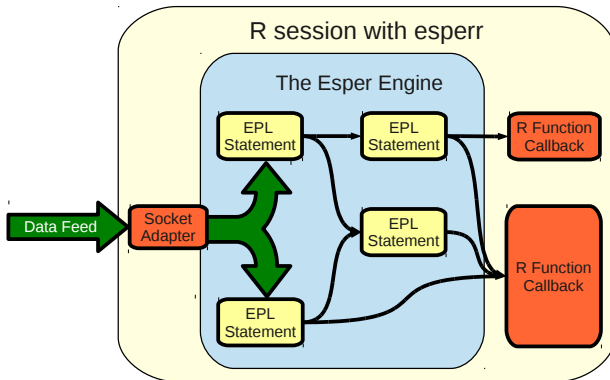
# EPL Capabilities

- Event streams “flow through” EPL statements, producing new output events when the statement conditions are satisfied.
- EPL statements
  - filter
  - aggregate
  - tabulate
  - window
  - process

# Flowing Market Data



# Flowing Market Data with **esperr**



## Example

```
schema <- "tick.xsd"  
esperSchema(schema, 'tick', 'TickStream')  
data(tickXMLEvent)  
s <- esperStatement(  
  'SELECT symbol, price FROM TickStream')  
f <- function(event) {  
  id <- getEventString(event, 'symbol')  
  command <- getEventString(event, 'price')  
  cat(symbol, price, '\n')  
}  
registerEventListener(s, 'f')  
sendEvent(tickXMLEvent)
```

# Using **esperr**

- Procedure
  - 1 Plug Esper into your favorite stock feed
  - 2 Use EPL statements to grab interesting market data
  - 3 Use R to analyze the data and trade
  - 4 Profit
- Also generates Esper events
  - Useful for backtesting
  - Create other streams (not just stock data)

# VWAP Example 1: Calculation is done in R

## The VWAP Mutable Closure

```
vwapGenerator=function(size=100) {  
  .prices=c(); .volumes=c(); .size=size  
  function(event) {  
    .prices <<- c(.vals,  
      as.numeric(getEventString(event, "price"))  
    .volumes <<- c(.vals,  
      as.numeric(getEventString(event, "volume"))  
    if (length(.prices) > .size) {  
      .prices <<- .prices[-1]  
      .volumes <<- .volumes[-1]  
    }  
    cat ("VWAP_for", getEventString(event, "symbol"),  
      "is", sum(.prices*.volumes)/sum(.volumes), "\n")  
  }  
}
```

# VWAP Example 1: Calculation is done in R

Associate the EPL statement with the mutable closure.

```
s <- esperStatement(  
  'SELECT_*_FROM_TickStream_WHERE_symbol="GOOG"')  
googVWAP <- vwapGenerator()  
registerEventListener(s, googVWAP)
```

## VWAP Example 2: Calculation is done in Esper

```
vwap <-function(event) {  
  cat("VWAP_for", getEventString(event, "symbol"),  
      "is", getEventString(event, "vwap"), "\n")  
}  
  
s <- esperStatement(paste(  
  "SELECT_sum(price*volume)/sum(volume)",  
  "as_VWAP_from_TickStream.win:length(100)",  
  "where_symbol='GOOG'"))  
  
registerEventListener(s, vwap)
```

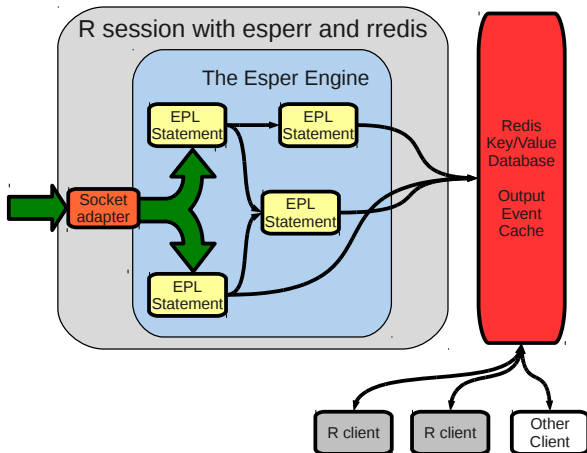
# Implementation Guidelines

- Use Esper
  - Filter on the events of interest
  - Perform simple processing (VWAP, MA, etc.)
- Use R
  - High-level analysis
  - “Strategies” implemented with mutable closures

# Performance

- VWAP on 5 minute intervals  $\approx$  4,000 events/s on generic hardware with current implementation
- Next version will include Apache Axiom streaming XML protocol to improve performance by at least an order of magnitude (maybe two)
- Can use Redis as an output event cache

# R, Esper and Redis



# Conclusions

## `esperr`

- Handles asynchronous events
  - Events are processed in Esper
  - Callbacks are defined in R
- Provides pipeline concurrency
  - Esper processing and R analysis runs on different processes
  - Increased throughput

## For Further Reading



[The Esper Website.](http://esper.codehaus.org)

`http://esper.codehaus.org`



[The \*\*esperr\*\* R package.](http://github.com/bwlewis/esperr)

`http://github.com/bwlewis/esperr`



[The redis Website.](http://code.google.com/p/redis/)

`http://code.google.com/p/redis/`



[The \*\*rredis\*\* R package.](http://cran.r-project.org/web/packages/rredis/index.html)

`http://cran.r-project.org/web/packages/rredis/  
index.html`