



REvolution
computing

We do the math

foreach + iterators

foreach

- A for-loop/lapply hybrid
- Similar to foreach and list comprehensions in Python and other languages

iterators

- Similar to Java iterators
- nextElem ()

```
foreach (iterator) %dopar%  
  
  {  
    statements  
  }
```

```
> foreach (j=1:4) %dopar% {sqrt (j) }
```

```
[[1]]
```

```
[1] 1
```

```
[[2]]
```

```
[1] 1.414214
```

```
[[3]]
```

```
[1] 1.732051
```

```
[[4]]
```

```
[1] 2
```

Aggregation/reduction with .combine:

```
> foreach(j=1:4, .combine=c) %dopar% {sqrt(j)}  
[1] 1.000000 1.414214 1.732051 2.000000
```

```
> foreach(j=1:4, .combine='+') %dopar% sqrt(j)  
[1] 6.146264
```

Foreach is more general than most implementations of parallel lapply. The following typically doesn't work with miscellaneous parLapplies:

```
> z <- 2
> f <- function (x) sqrt (x + z)
> foreach (j=1:4, .combine='+') %dopar% f(j)
[1] 8.417609
```

Here is a simple simulation:

```
birthday <- function(n) {  
  ntests <- 1000  
  pop <- 1:365  
  anydup <- function(i)  
    any(duplicated(  
      sample(pop, n, replace=TRUE)))  
  sum(sapply(seq(ntests), anydup)) / ntests  
}  
  
x <- foreach (j=1:100) %dopar% birthday (j)
```

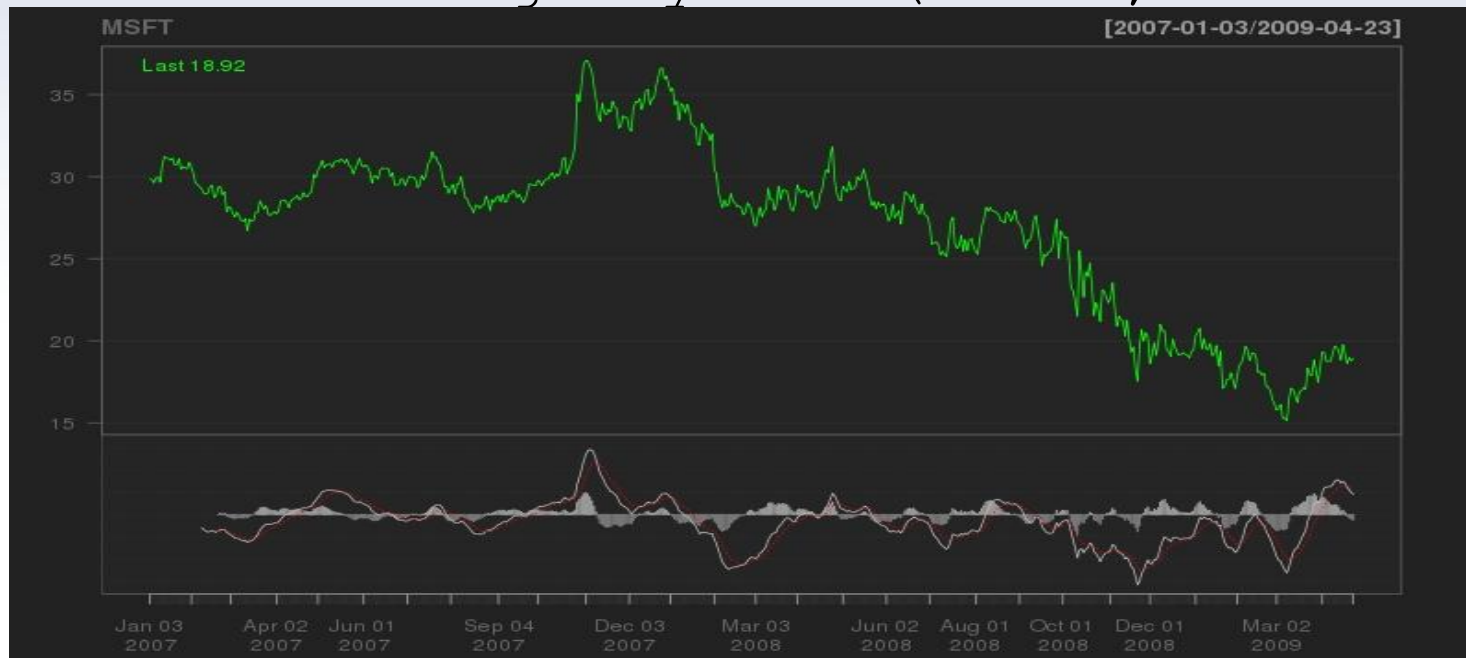
`%dopar%`

Modular parallel backends:

- doSEQ (the default)
- doNWS (NetWorkSpaces)
- doSNOW
- doRMPI
- doSMP
- doMulticore

A simple example: backtesting a technical trading rule
(with TTR, quantmod, PerformanceAnalytics and foreach):

```
startDate <- "2007-01-01"  
endDate   <- Sys.Date()  
MSFT      <- getSymbols("MSFT", ...  
IEF       <- getSymbols("IEF", ...
```



```
Ra <- Return.calculate (Cl (MSFT) )  
Rb <- Return.calculate (Cl (IEF) )  
chart.CumReturns (cbind (Ra, Rb) )
```



A very simple trading rule:

```
simpleRule <- function (z, fast=12, slow=26,  
                      signal=9, instr, benchmark)  
{  
  x <- MACD (z, nFast=fast, nSlow=slow,  
            nSig=signal, maType="EMA")  
  position <- sign(x[,1]-x[,2])  
  s <- xts(position, order.by=index(z))  
  return (instr*(s>0) + benchmark*(s<=0))  
}
```

Brute-force optimization of the fast and slow parameters:

```
M <- 100
S <- matrix(0,M,M)

for (j in 1:(M-1)) {
  for (k in min((j+2),M):M) {
    R <- simpleRule (Cl (MSFT),j,k,9, Ra, Rb)
    Dt <- na.omit (R - Rb)
    S[j,k] <- mean (Dt) / sd(Dt)
  }
}
```

With foreach:

```
M <- 100
```

```
S <- foreach (j=1:(M-1), .combine=rbind,  
             .packages=c('xts','TTR')) %dopar% {  
  x <- rep(0,M)  
  for (k in min ((j+2),M):M) {  
    R <- simpleRule (Cl (MSFT), j, k, 9, Ra, Rb)  
    Dt <- na.omit (R - Rb)  
    x[k] <- mean (Dt) / sd (Dt)  
  }  
  return(x)  
}
```

```

j <- which (S==max(S), arr.ind=TRUE)
Ropt <- simpleRule (Cl (MSFT), j[1], j[2], 9, Ra, Rb)
chart.CumReturns (cbind (Ra, Rb, Ropt))
    
```

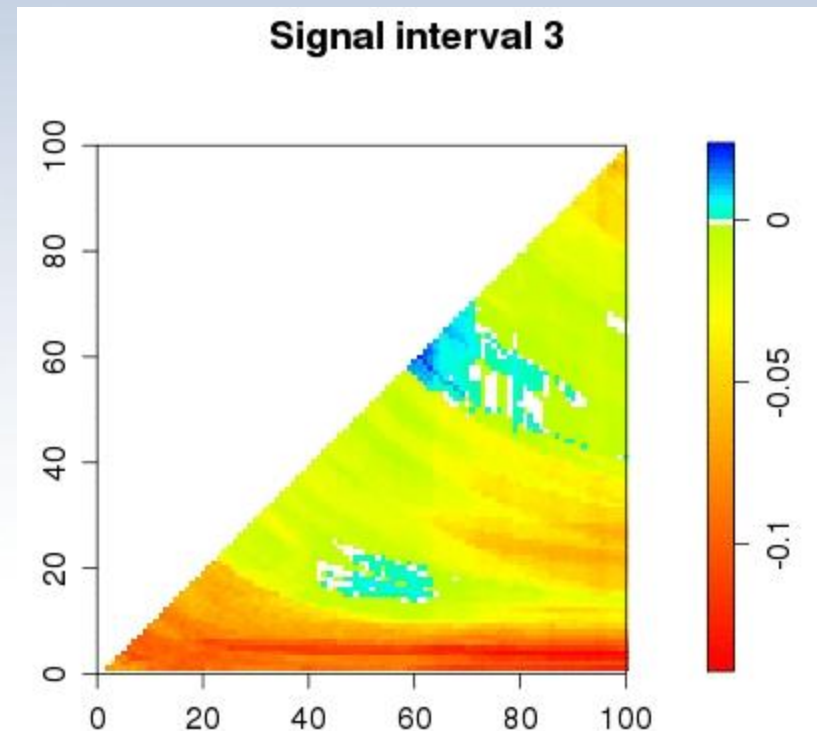


These parameters are nicely visualized with the spatstat package

```
require ("spatstat")

function showIm (S) {
  (wrapper for image) ... }

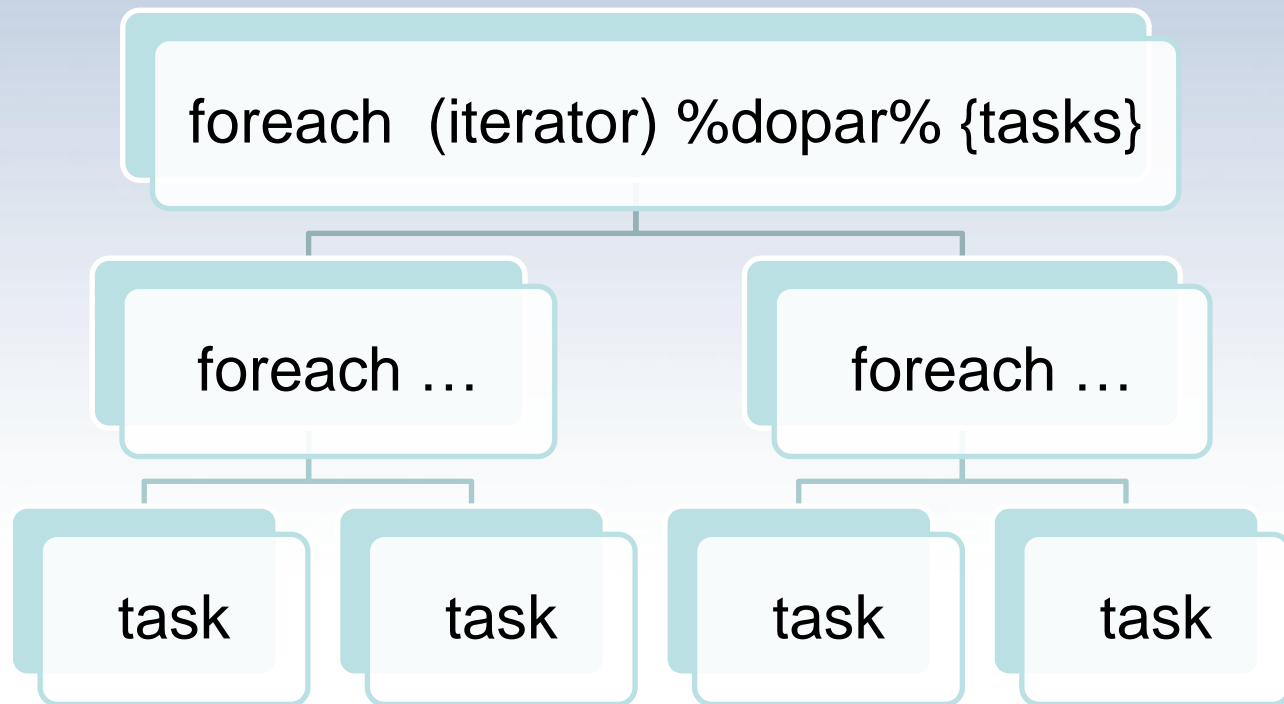
for (j in 3:20) {
  S <- S3[, , j]
  showIm(S)
}
```



An example of explicit multi-paradigm ||ism

CLUSTER

SMP




```
require ( 'snow' )
require ( 'foreach' )
require ( 'doSNOW' )

cl <- makeCluster (c ( 'n1', 'n2' ))
registerDoSNOW ( )

foreach (iterator,
        .packages=c ( 'foreach', 'doMETHOD' )
        %dopar%
        {
            registerMETHOD ( )
            foreach (iterator) %dopar% {
                tasks...
            }
        }
    )
```

Summary

Foreach is a simple approach to parallel computing with R that maps naturally on to a number of existing systems for distributed computing.