# *pR*: Introduction to Parallel R for Statistical Computing

## for people with little knowledge but infinite intelligence

CScADS Scientific Data and Analytics for Petascale Computing Workshop

**Nagiza F. Samatova**
samatovan@ornl.gov
North Carolina State University
Oak Ridge National Laboratory

8/03/2009

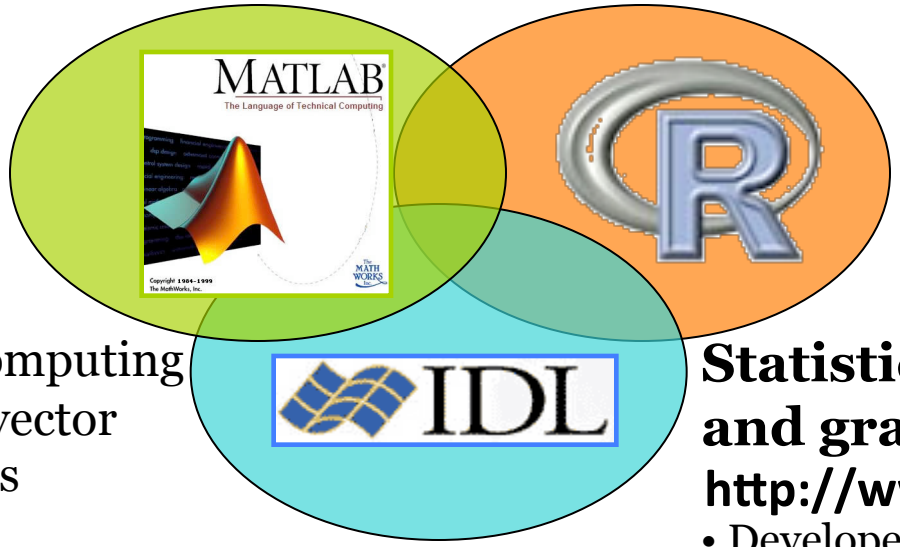
SDM CENTER

Office of Science
U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY
Department of Computer Science

OAK RIDGE
National Laboratory

# Why R?




- Technical computing
- Matrix and vector formulations

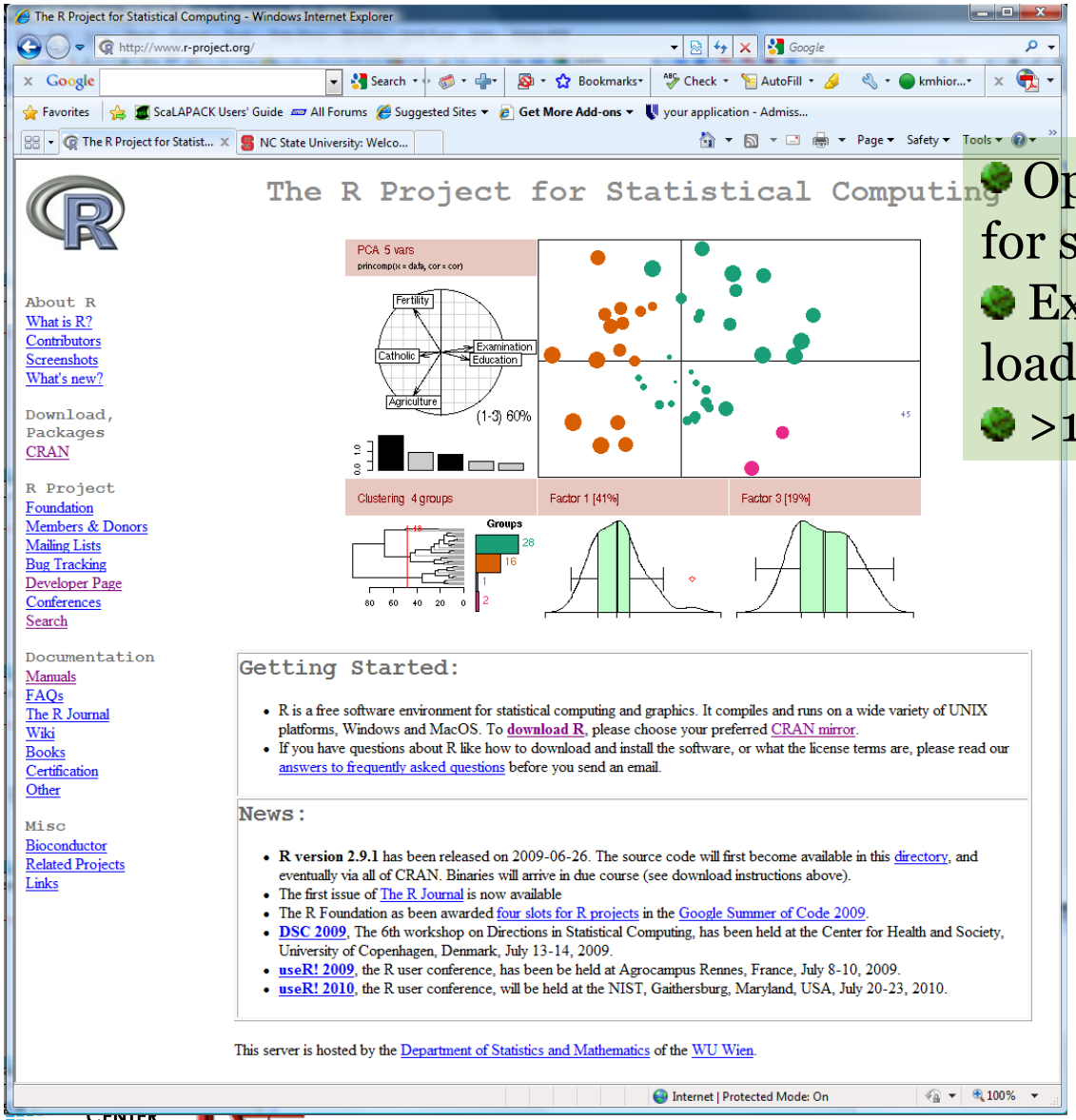
- Data Visualization and analysis platform
- Image processing, vector computing

**Statistical computing and graphics**

**http://www.r-project.org**

- Developed by **R**. Gentleman & **R**. Ihaka
- Expanded by community as open source
- Statistically rich

# Statistical Computing with *R* – http://www.r-project.org




- Open source, most widely used for statistical analysis and graphics
- Extensible via dynamically loadable add-on packages
- >1,800 packages on CRAN

```
> library (stats)
> pca = prcomp (data)
> summary (pca)
> …
> dyn.load( “foo.so”)
> .C( “foobar” )
> dyn.unload( “foo.so” )
```

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

# How to Get Started with R?

- Step 1: **Download R** (under Linux):
  - mkdir for RHOME; cd $RHOME
  - wget http://cran.cnr.berkeley.edu/src/base/R-2/R-2.9.1.tar.gz
- Step 2: **Install R** (under Linux):
  - tar –zxvf  R-2.9.1.tar.g
  - ./configure *--prefix=<RHOME>  --enable-R-shlib*
  - make
  - make install
- Step 3: **Run R** (under Linux):
  - Update env. variables in $HOME/.bash_profile:
    - *export PATH=<RHOME>/bin:$PATH*
    - *export R_HOME=<RHOME>*
  - R

# Session 1: Vectors and vector operations

To create a vector:

```
# c() command to create vector x
x=c(12,32,54,33,21,65)
# c() to add elements to vector x
x=c(x,55,32)
```

```
# seq() command to create
sequence of number
years=seq(1990,2003)
# to contain in steps of .5
a=seq(3,5,.5)
```

```
# rep() command to create data
that follow a regular pattern
b=rep(1,5)
c=rep(1:2,4)
```

To access vector elements:

```
# 2nd element of x
x[2]
# first five elements of x
x[1:5]
# all but the 3rd element of x
x[-3]
# values of x that are < 40
x[x<40]
# values of y such that x is < 40
y[x<40]
```

To perform operations:

```
# mathematical operations on vectors
y=c(3,2,4,3,7,6,1,1)
x+y; 2*y;  x*y;  x/y;  y^2
```

# Session 2: Matrices & matrix operations

To create a matrix:

```
# matrix() command to create matrix A with rows and cols
A=matrix(c(54,49,49,41,26,43,49,50,58,71),nrow=5,ncol=2))
```

To access matrix element:

```
# matrix_name[row_no, col_no]
 A[2,1] # 2nd row, 1st column element
 A[3,]  # 3rd row
 A[,2]  # 2nd column of the matrix
 A[2:4,1]  # submatrix of 2nd-4th elements
of the 1st column
 A["KC",] # access row by name, "KC"
```

Statistical operations:

```
 rowSums(A)
 colSums(A)
 rowMeans(A)
 colMeans(A)
# max of each columns
 apply(A,2,max)
# min of each row
 apply(A,1,min)
```

To perform element by element ops:

```
2*A+3; A+B; A*B; A/B;
```

# Help in R

- **At the package level**:
  - help (package=pkg_name) (e.g. help (package=stats))
  - pkg_name::<TAB><TAB> (stats:: )
  - CRAN Search:
  - CRAN Task Views: http://cran.cnr.berkeley.edu/web/views/
  - From R GUI: Help → Search help…
- **At the function level**:
  - help (function_name) (e.g. help(prcomp))
  - ?function_name (e.g.: ?prcomp)
  - ??function_name

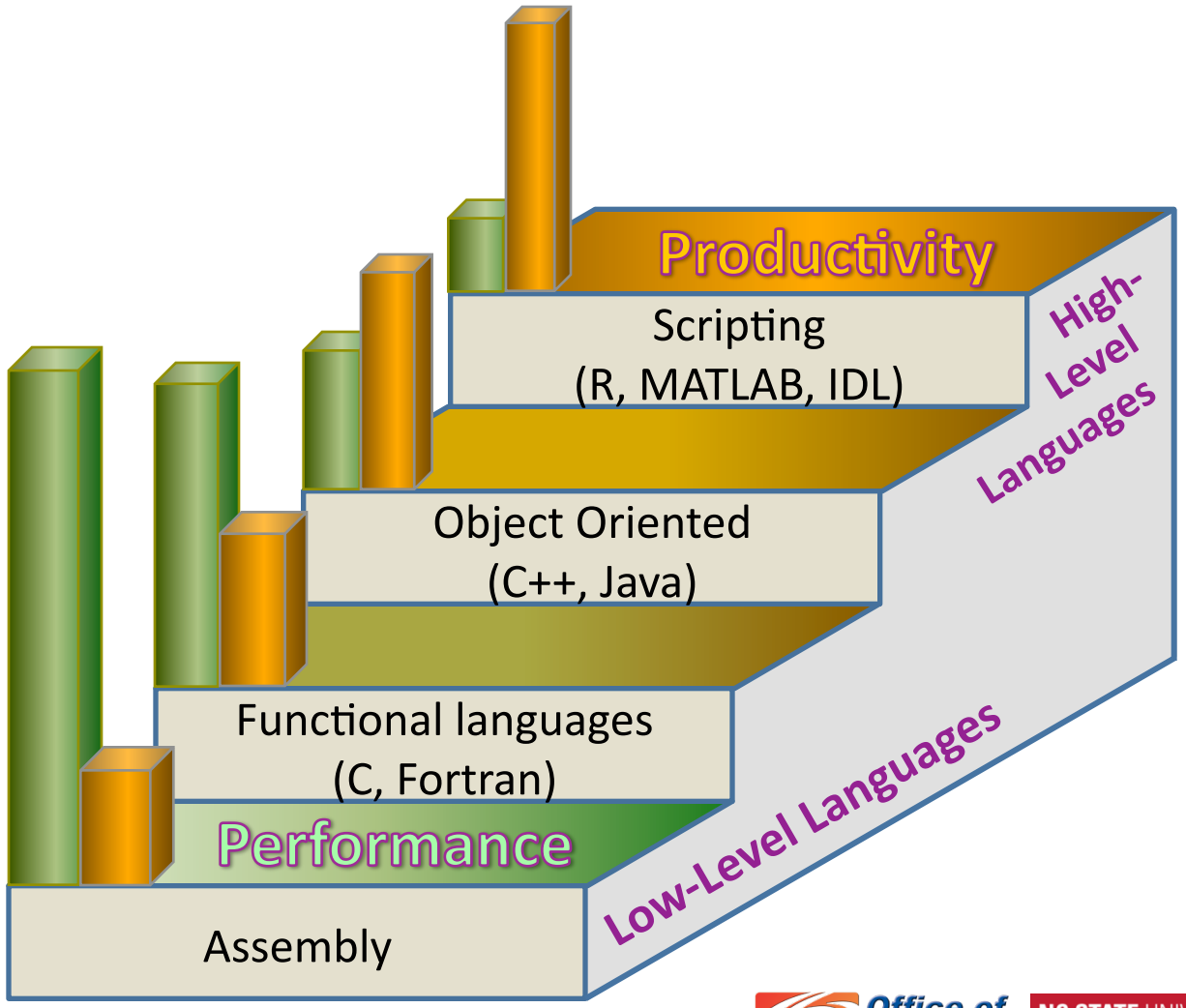
# How to Add a Package?

- From R prompt:
  - install.packages(“pkg_name") (install.packages(“ctv”)
- From R GUI:
  - Packages → Install package(s)…
- From Command Line:
  - R CMD INSTALL pkg_name.version.tar.gz

# Useful R Links

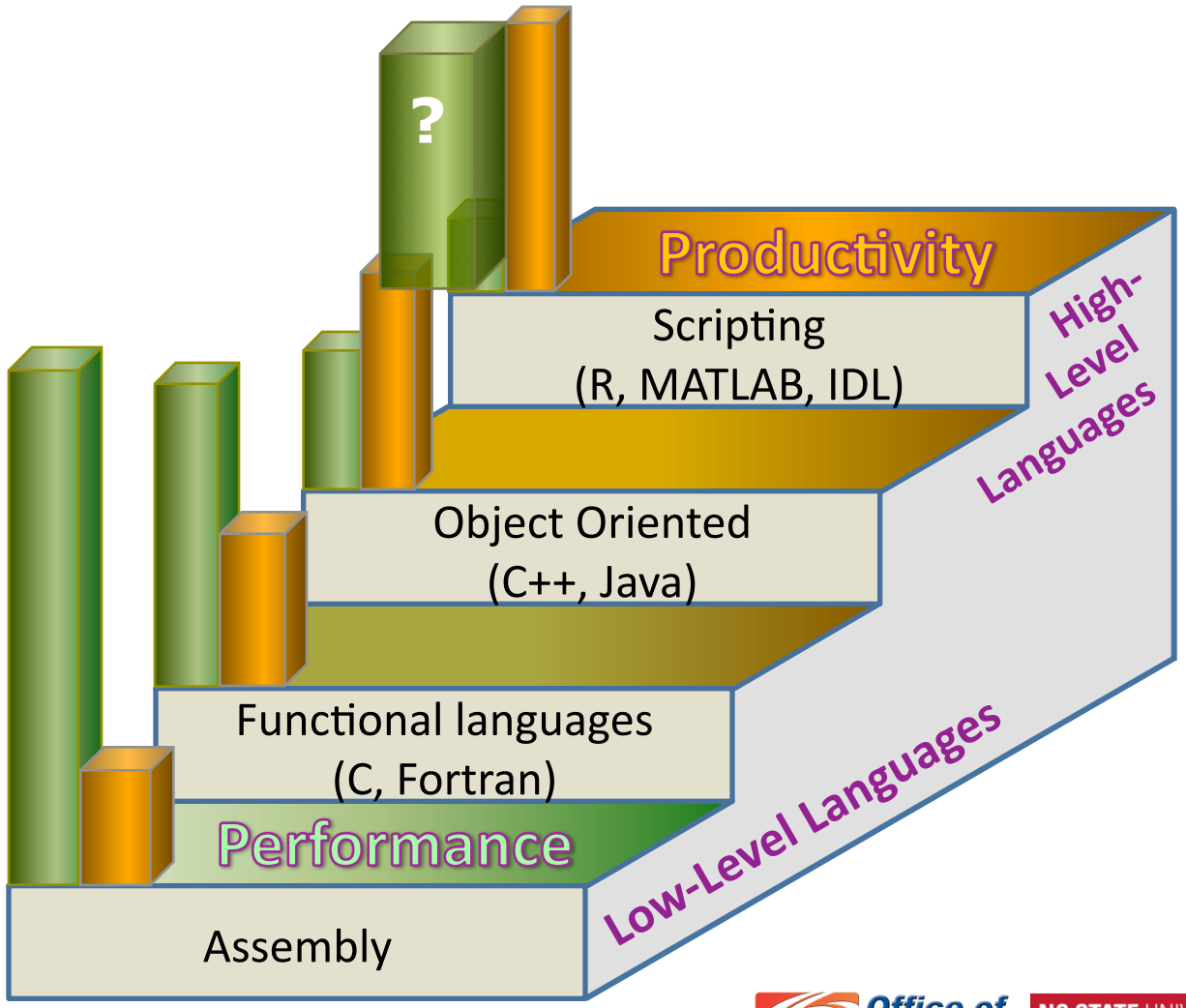
- R Home: http://www.r-project.org/
- R's CRAN package distribution: http://cran.cnr.berkeley.edu/
- Introduction to R manual: http://cran.cnr.berkeley.edu/doc/manuals/R-intro.pdf
- Writing R extensions: http://cran.cnr.berkeley.edu/doc/manuals/R-exts.pdf
- Other R documentation: http://cran.cnr.berkeley.edu/manuals.html

# The Programmer's Dilemma



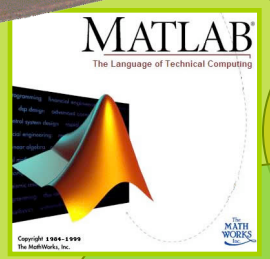
# Towards High-Performance High-Level Languages

## How do we get there? – Parallelization



# One Hat Does NOT Fit All

## Parallel R for Data Intensive Statistical Computing





**Data Intensive Statistical Computing**

- Technical computing
- Matrix and vector formulations

- Data Visualization and analysis platform
- Image processing, vector computing

**Statistical computing and graphics**

http://www.r-project.org

- Developed by **R**. Gentleman & **R**. Ihaka
- Expanded by community as open source
- Extensible via dynamically loadable libs

SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

# Towards Enabling Parallel Computing in *R*

http://cran.cnr.berkeley.edu/web/views/HighPerformanceComputing.html

- ***snow*** (Luke Tierney): general API on top of message passing routines to provide high-level (*parallel apply*) commands; mostly demonstrated for ***embarrassingly parallel*** applications.

| ***snow* API** | *High Level Routines* |
|---|---|
| `parLapply` | parallel `lapply` |
| `parSapply` | parallel `sapply` |
| `parApply` | parallel `apply` |
| | *Basic Routines* |
| `clusterExport` | export variables to nodes |
| `clusterCall` | call function on each node |
| `clusterApply` | apply function to arguments on nodes |
| `clusterApplyLB` | load balanced `clusterApply` |
| `clusterEvalQ` | evaluate explicit expression on nodes |
| `clusterSplit` | split vector into pieces for nodes |
| | *Administrative Routines* |
| `makeCluster` | create a new cluster of nodes |

- ***Rmpi*** (Hao Yu): ***R*** interface to ***MPI***.
- ***rpvm*** (Na Li and Tony Rossini): ***R*** interface to ***PVM***; requires knowledge of parallel programming.

```
> library (rpvm)
> .PVM.start.pvmd ()
> .PVM.addhosts (...)
> .PVM.config ()
```

SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

# Lessons Learned from R/Matlab Parallelization

## Interactivity and High-Level: Curse & Blessing

high

Abstraction
Interactivity
Productivity

low

**Back-end approach**
- **data parallelism**
- **C/C++/Fortran with MPI**
- **RScaLAPACK** (Samatova et al, 2005)

**Automatic parallelization**
- **task parallelism**
- **task-pR** (Samatova et al, 2004)

**Embarrassing parallelism**
- **data parallelism**
- **snow** (Tierney, Rossini, Li, Sevcikova, 2006)

**Manual parallelization**
- **message passing**
- **Rmpi** (Hao Yu, 2006)
- **rpvm** (Na Li & Tony Rossini, 2006)

**Compiled approach**
- **Matlab→C→automatic parallelization**

**pR**

Packages: http://cran.r-project.org/

**Parallel Performance**

high

SDM CENTER

# Types of Users

- **R end-user**:
  - Use R scripting language for statistical analysis tasks
- **R contributor**:
  - Contribute R packages
  - Use R (sometimes serial C/C++, Fortran)
- **HPC developer**:
  - MPI C/C++/Fortran codes
  - Linear algebra routines that underlie data analysis *f*()
  - Parallel machine learning and data mining algorithms:
    - Unsupervised learning: clustering, association rule mining,…
    - Supervised learning: SVM, NN, Decision Trees, Bayesian networks

# Our Philosophy

- From R end user's perspective:
  - Require ***NO*** (very trivial) changes to serial R code
  - Yet deliver HPC performance

- From HPC developer's perspective:
  - Provide ***native*** to HPC developer interface to R internals
  - With NO (constantly small) overhead

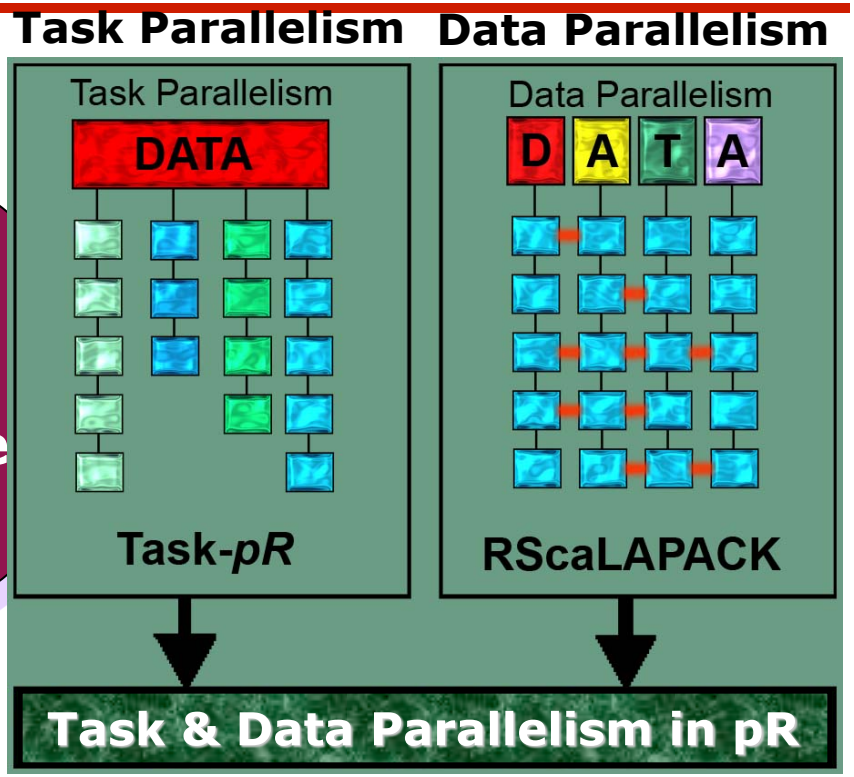
# Task and Data Parallelism in pR

**Goal: Parallel R (*pR*) aims:**

**(1) to automatically detect and execute *task-parallel* analyses;**

**(2) to easily plug-in *data-parallel* MPI-based C/C++/Fortran code**

**(3) to retain high-level of *interactivity, productivity* and *abstraction***




**Embarrassingly-parallel:**

- Likelihood Maximization
- Sampling: Bootstrap, Jackknife
- Markov Chain Monte Carlo
- Animations

**Data-parallel:**

- k-means clustering
- Principal Component Analysis
- Hierarchical clustering
- Distance matrix, histogram

SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

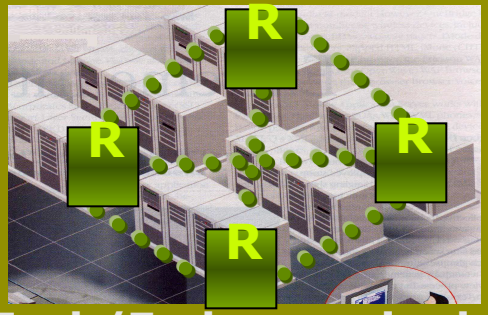
OAK RIDGE National Laboratory

# *pR* Multi-tiered Architecture

**Interactive R Client**

**Loosely Coupled**

**R Servers**



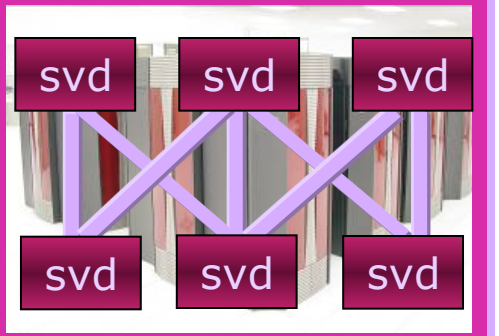

**Task/Embarrassingly parallel jobs**

A

S

**Tightly Coupled**

**MPI Servers**




**Data parallel jobs**

```
A ← matrix (1:10000, 100,100)
library (pR)

S ← sla.svd(A)        Data parallel
b ← list ()
for (k in 1:dim (A) [ 1 ] ) {
    b [ k ] ← sum ( A [ k, ] )
}          Embarrassingly parallel
m ← mean ( A )
d ← sum ( A )          Task parallel
```

getValue()

putValue()

**Data Bank Server(s)**

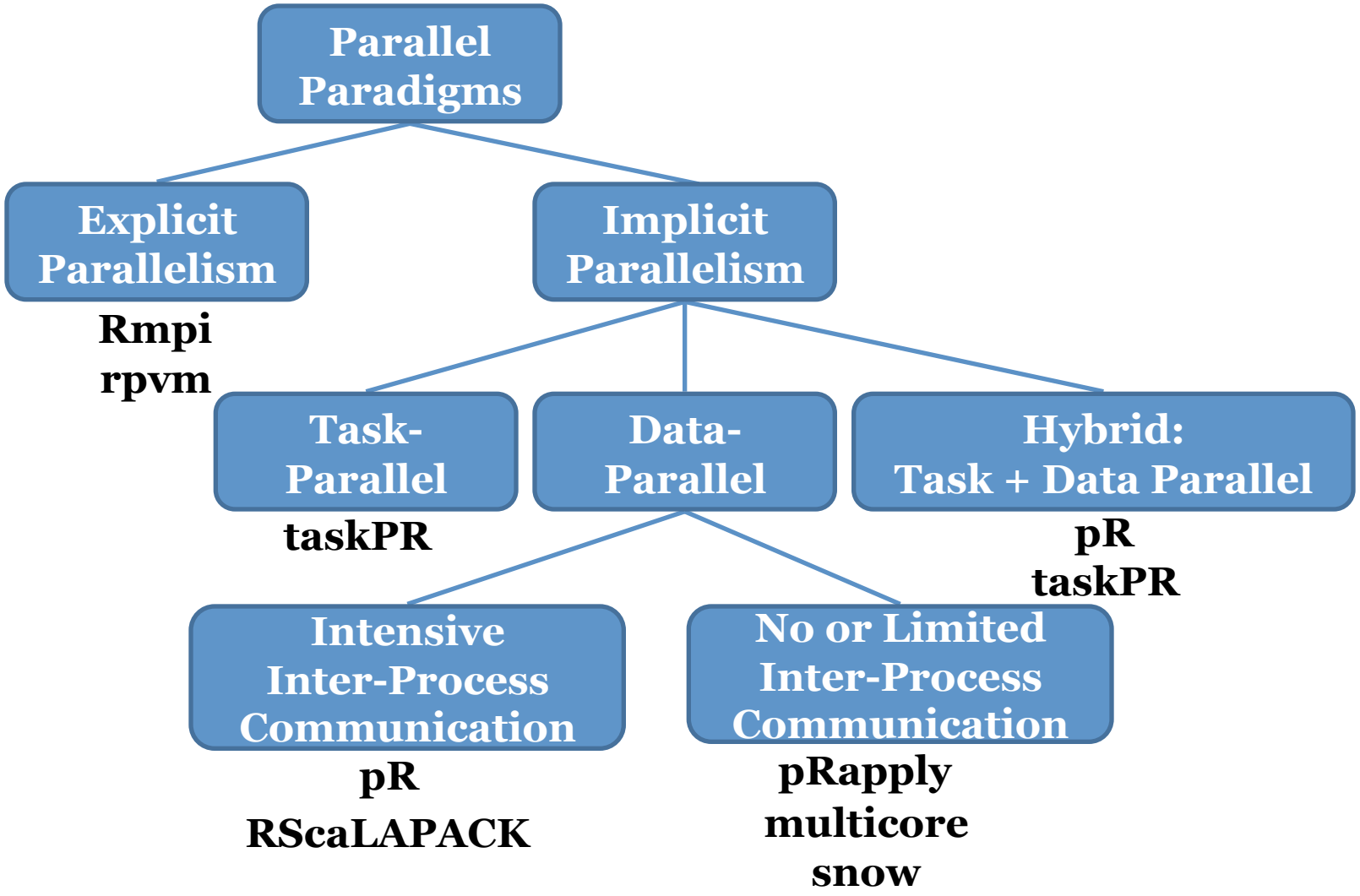
**Memory & I/O Management**

U.S. DEPARTMENT OF ENERGY

CENTER

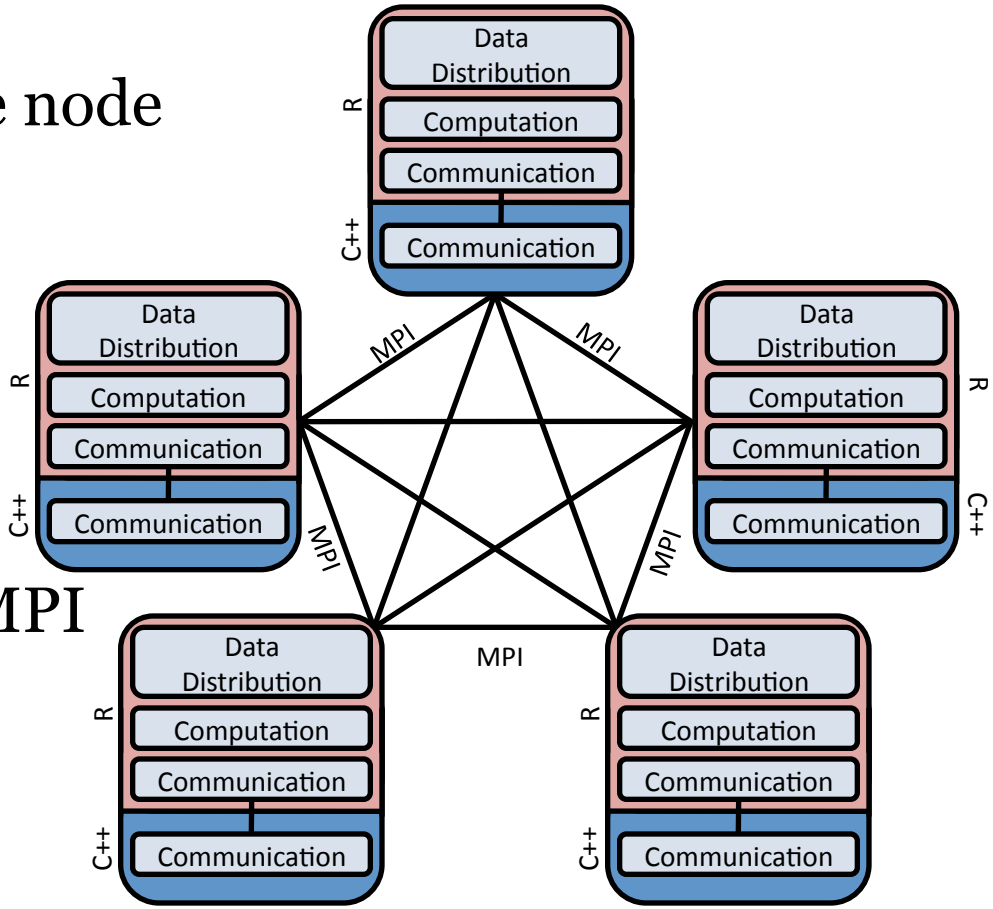
OAK RIDGE National Laboratory

# Parallel Paradigm Hierarchy



# Rmpi May Not Be Ideal for All End-Users

- R-wrapper around MPI
- R is required at each compute node
- Executed as interpreted code, which introduces noticeable overhead
- Supports ~40 of >200 MPI-2 functions
- Users must be familiar with MPI details
- Can be especially useful for prototyping



# Rmpi Matrix Multiplication Requires Parallel Programming Knowledge and is Rmpi Specific

```
mm_Rmpi <- function(A, B, n_cpu = 1) {
  da <- dim(A) ## dimensions of matrix A
  db <- dim(B) ## dimensions of matrix B

  ## Input validation
  matrix_mult_validate( A, B, da, db )
  if( n_cpu == 1 )
    return(A %*% B)

  ## spawn R workers
  mpi.spawn.Rslaves( nslaves = n_cpu )

  ## broadcast data and functions
  mpi.bcast.Robj2slave( A )
  mpi.bcast.Robj2slave( B )
  mpi.bcast.Robj2slave( n_cpu )

  ## how many rows on workers ?
  nrows_workers <- ceiling( da[ 1 ] / n_cpu )
  nrows_last <- da[ 1 ] - ( n_cpu - 1 ) *
    nrows_workers

  ## broadcast number of rows and foo to apply
  mpi.bcast.Robj2slave( nrows_workers )
  mpi.bcast.Robj2slave( nrows_last )
  mpi.bcast.Robj2slave( mm_Rmpi_worker )

  ## start partial matrix multiplication
  mpi.bcast.cmd( mm_Rmpi_worker() )

  ## gather partial results from workers
  local_results <- NULL
  results <- mpi.gather.Robj(local_results)
  C <- NULL

  ## Rmpi returns a list
  for(i in 1:n_cpu)
    C <- rbind(C, results[[ i + 1 ]])

  mpi.close.Rslaves()
  C
}
```

Worker

```
mm_Rmpi_worker <- function(){
  commrank <- mpi.comm.rank() - 1
  if(commrank == ( n_cpu - 1 ))
    local_results <- A[ (nrows_workers * commrank + 1): (nrows_workers * commrank + nrows_last),] %*% B
  else
    local_results <- A[ (nrows_workers * commrank + 1): (nrows_workers * commrank + nrows_workers),] %*% B

  mpi.gather.Robj(local_results, root = 0, comm = 1)
}
```

# pR Matrix Multiplication

pR example:

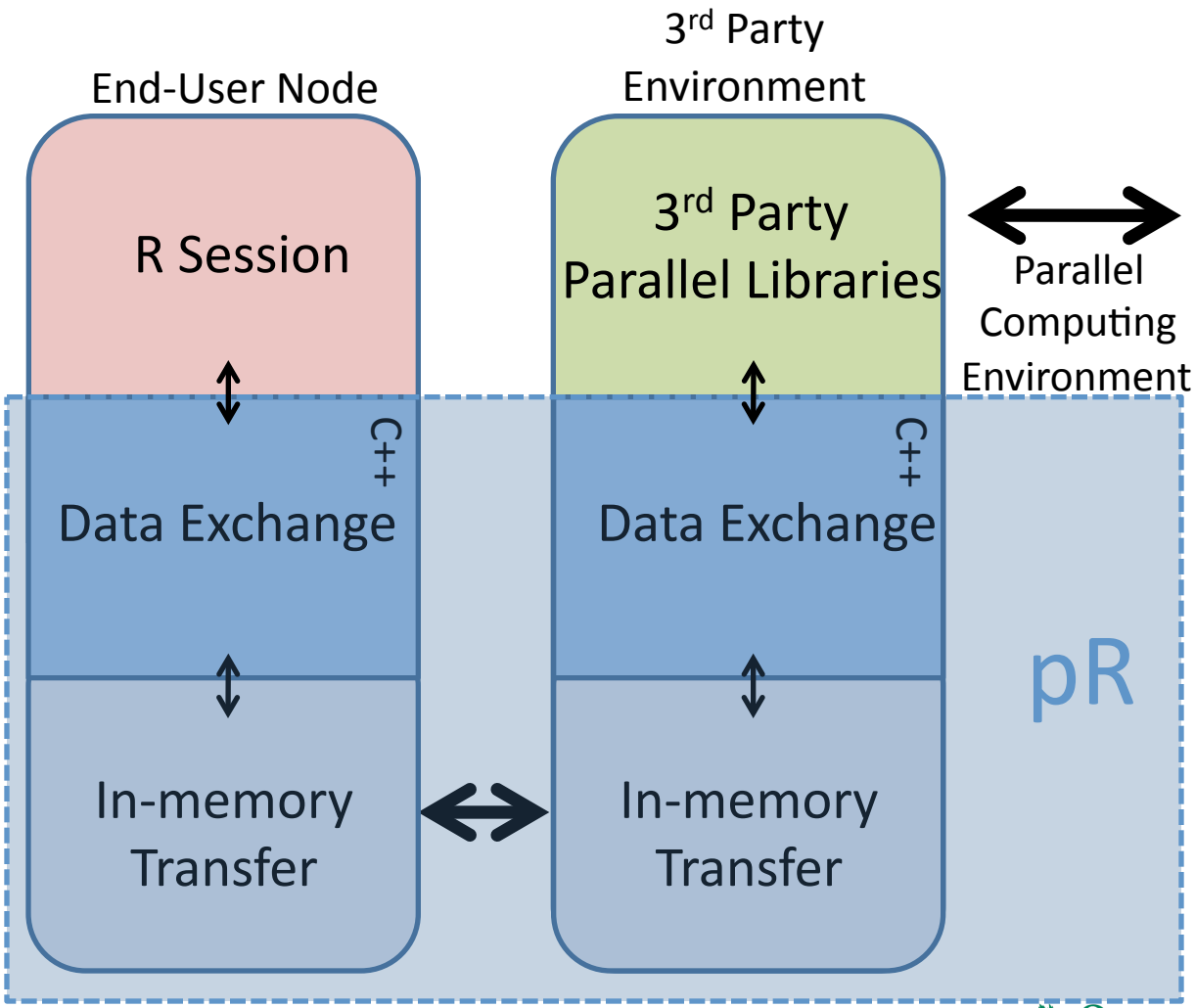
```
library (RScaLAPACK)
A = matrix (c(1:256),16,16)
B = matrix (c(1:256),16,16)
C = sla.multiply (A, B)
```

Using R:

```
A = matrix (c(1:256),16,16)
B = matrix (c(1:256),16,16)
C = A % * % B
```

# pR Parallel Plugin: Goals & Software Stack

- Couple general parallel computation capabilities within R
- Shield end-users from the complexities of parallel computing
- Enable execution of compiled and scripting codes together



# pR Example: Do it Yourself

R End-User

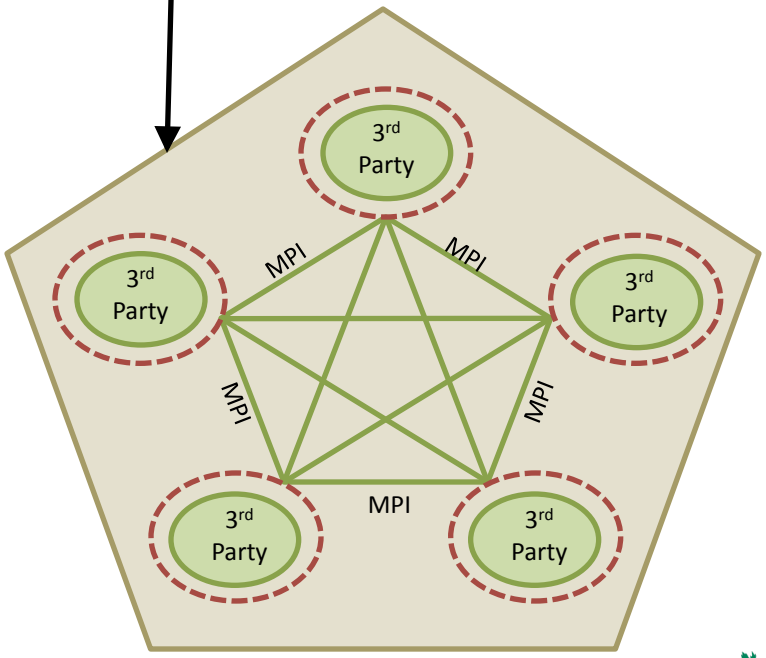
```
A = matrix(c(1:256),16,16)
B = matrix(c(1:256),16,16)
C = mm(A, B)
```

pR glue code

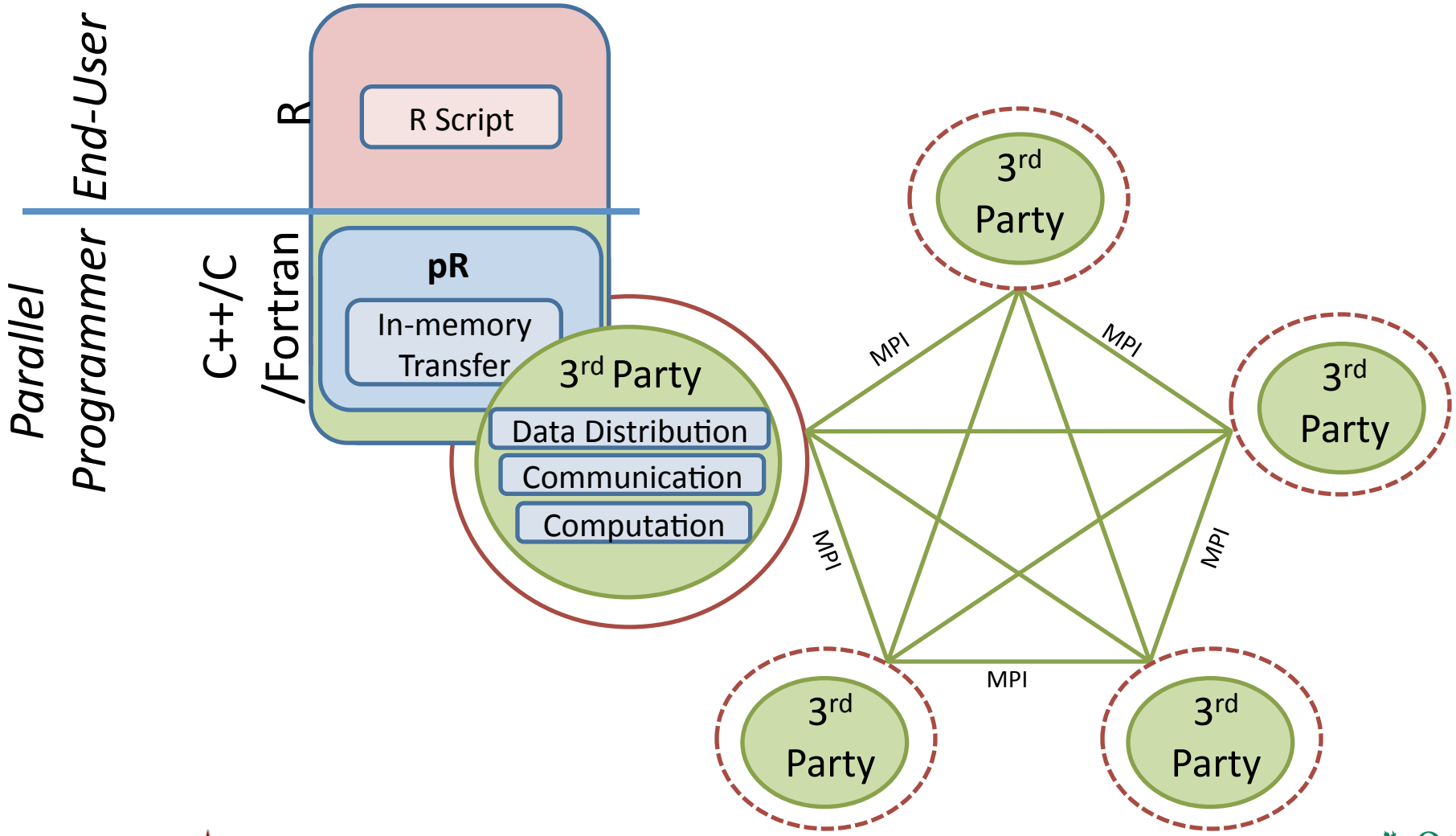
```
mm(SEXP args) {
  double *a, *b;
  a = args.getPointer();
  b = args.getPointer();
  c = matrix_multiply(a,b);
  return c.getRObject();
}
```

3rd Party Code: HPC MPI code

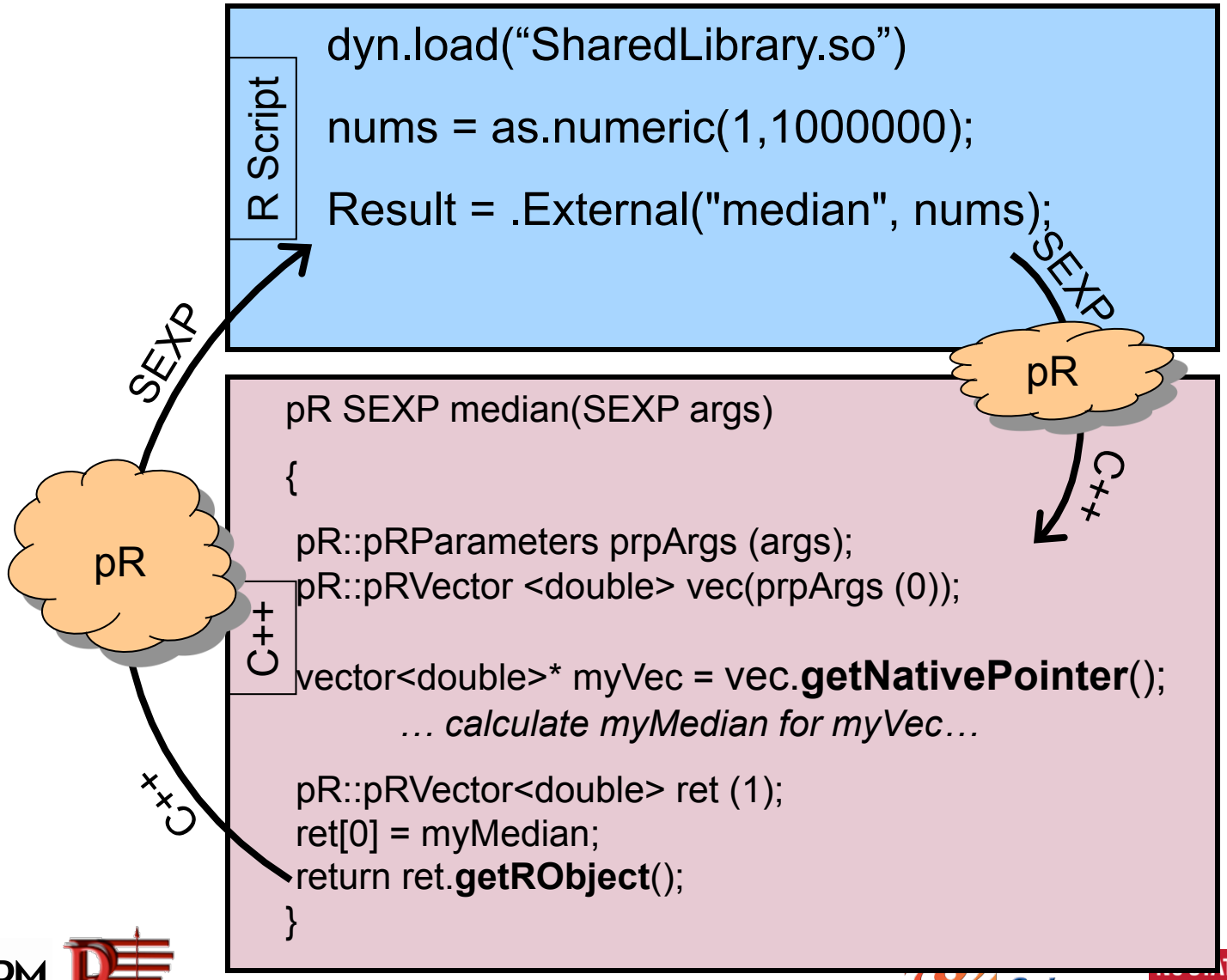
```
matrix_multiply(a,b) {
  ...
  return c;
}
```



# pR Overview


End-User
Parallel Programmer
R
R Script
C++/C /Fortran
pR
In-memory Transfer
3rd Party
Data Distribution
Communication
Computation
MPI
MPI
MPI
MPI
MPI
3rd Party
3rd Party
3rd Party
3rd Party

# C/C++/Fortran Plug-in to pR



# Serial pR Performance over Python and R

| Method | Python | R | pR | pR Improv. over Python | pR Improv. over R |
|---|---|---|---|---|---|
| Median | 0.300 | 0.108 | 0.030 | 10.0 | 3.6 |
| Geometric Mean | 0.323 | N/A | 0.128 | 2.9 | N/A |
| Average | 0.071 | 0.049 | 0.012 | 5.9 | 4.1 |
| Standard Deviation | 0.420 | 0.020 | 0.019 | 22.1 | 1.1 |
| **Average** | | | | 18.1 | 2.8 |

| Method | C++ | R | pR | R Overhead | pR Overhead | Times Imrov. over R |
|---|---|---|---|---|---|---|
| ddot | 0.003 | 0.131 | 0.003 | 0.128 | 0.001 | 43.7 |
| dnrm2 | 0.024 | 0.044 | 0.026 | 0.014 | 0.000 | 1.7 |
| dchdc | N/A | 0.001 | 0.001 | 0.001 | 0.000 | 1.0 |
| dsvdc | N/A | 0.661 | 3.864 | 0.026 | 0.000 | 0.2 |
| dgeev | 0.002 | 0.557 | 0.004 | 0.027 | 0.002 | 139.3 |
| **Average** | | | | | | 37.2 |

# pR Often Lowers Overhead Compared to R

| Method | C++ | *R* | pR | *R* Overhead | pR Overhead | Times Imrov. over *R* |
|---|---|---|---|---|---|---|
| ddot | 0.003 | 0.131 | 0.003 | 0.128 | 0.001 | 43.7 |
| dnrm2 | 0.024 | 0.044 | 0.026 | 0.014 | 0.000 | 1.7 |
| dchdc | N/A | 0.001 | 0.001 | 0.001 | 0.000 | 1.0 |
| dsvdc | N/A | 0.661 | 0.516 | 0.026 | 0.000 | 1.3 |
| dgeev | 0.002 | 0.557 | 0.004 | 0.027 | 0.002 | 139.3 |
| Average | | | | | | 37.2 |

Comparing C++, R and BridgeR External Method Calls in Seconds

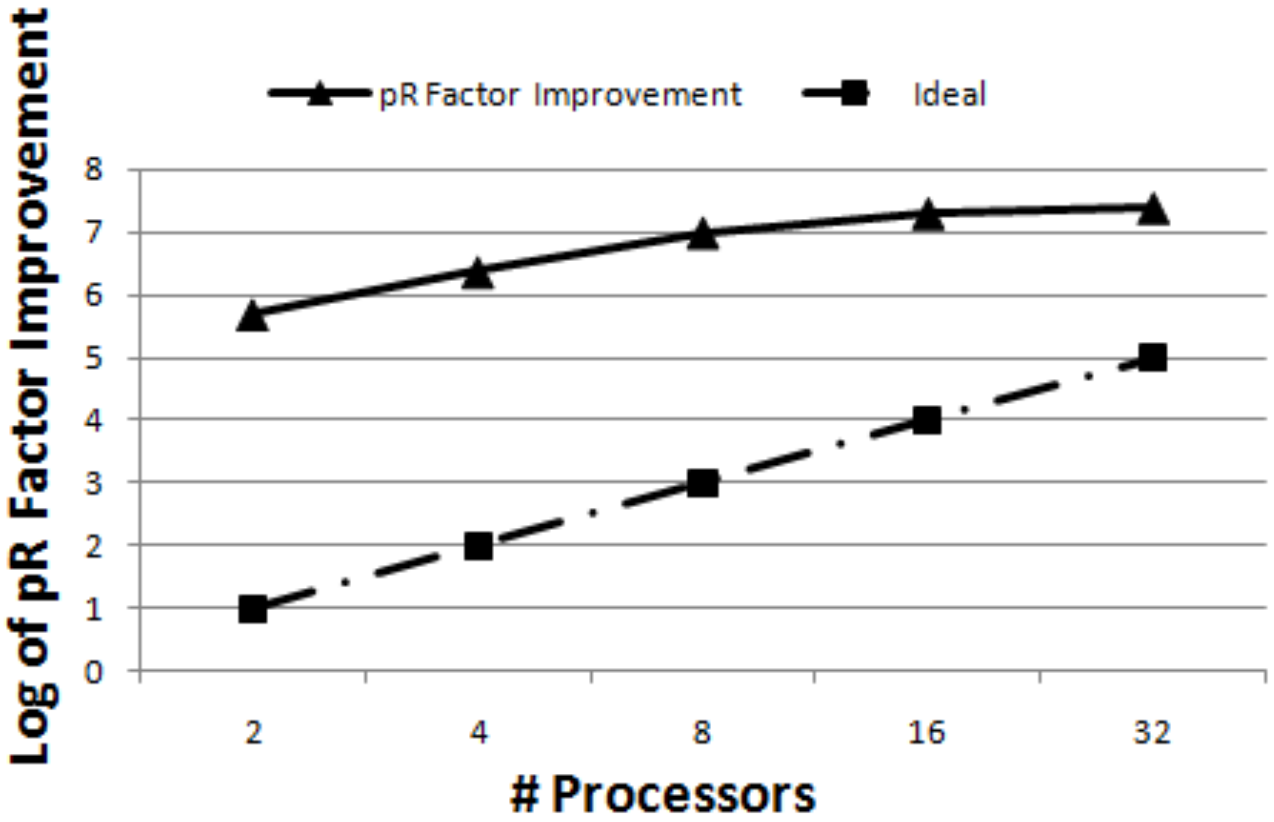
# pR to R Performance Comparison

| Matrix Size | $R$ (sec) | $pR$ (sec) | $pR$ Factor Improvement |
|---|---|---|---|
| 32 | $< 1 \cdot 10^{-3}$ | $1 \cdot 10^{-3}$ | 0.0 |
| 64 | $1 \cdot 10^{-3}$ | $8 \cdot 10^{-3}$ | 0.01 |
| 128 | $1 \cdot 10^{-3}$ | $8 \cdot 10^{-2}$ | 0.07 |
| 256 | $5 \cdot 10^{-2}$ | $1 \cdot 10^{-1}$ | 0.41 |
| 512 | $8 \cdot 10^{-1}$ | $2 \cdot 10^{-1}$ | 5.63 |
| 1024 | $1 \cdot 10^{1}$ | $6 \cdot 10^{-1}$ | 17.18 |
| 2048 | $9 \cdot 10^{1}$ | $4 \cdot 10^{0}$ | 19.78 |
| 4096 | $7 \cdot 10^{2}$ | $2 \cdot 10^{1}$ | 28.32 |

Single processor matrix-multiplication testcase.

System: 32 node Intel-based Infiniband cluster running Linux. Each node contains two 3.4 GHz Pentium IV processors and 6GB of memory.

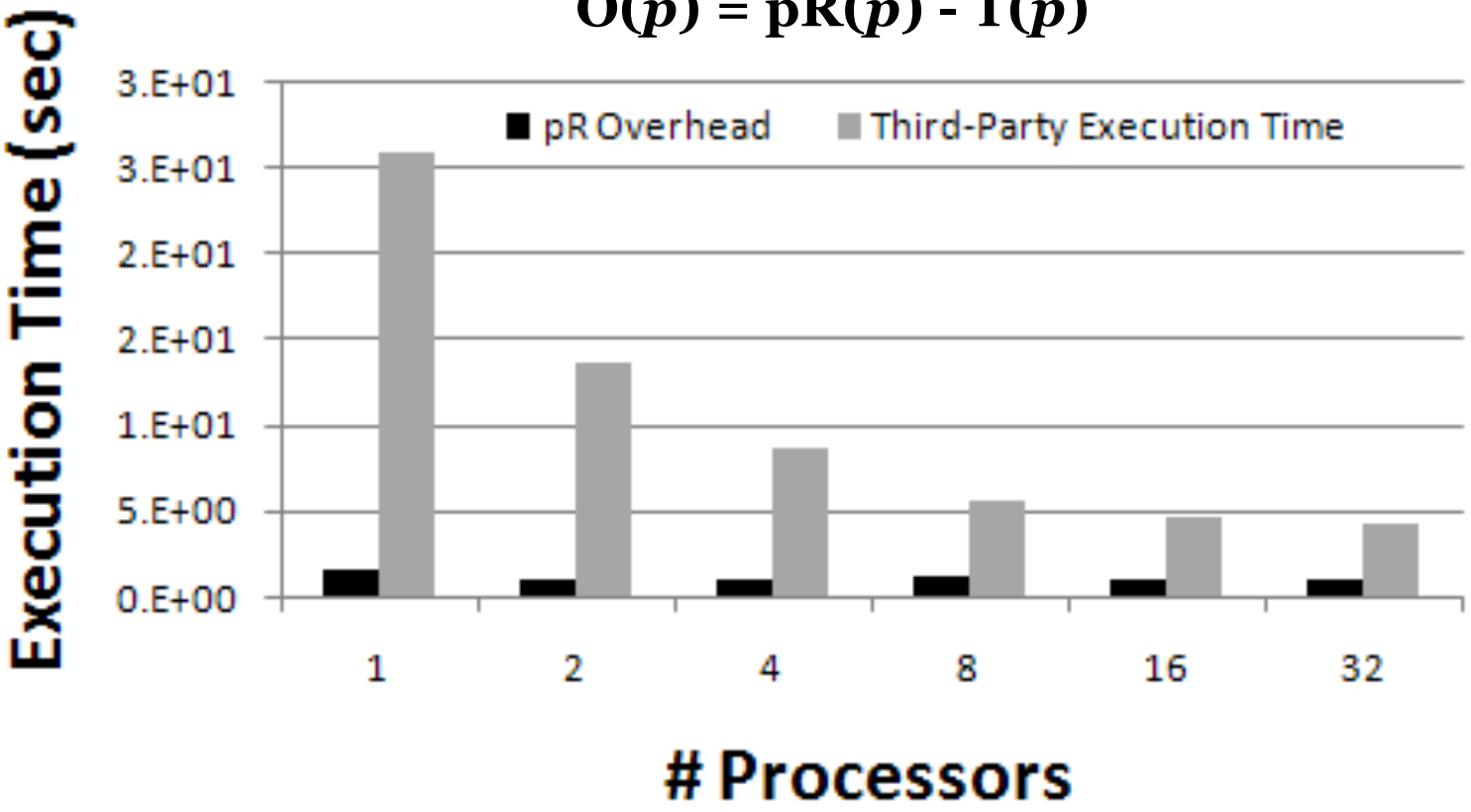
# pR Achieves Superlinear Speedup vs. Serial R




Using a matrix-multiplication testcase on 4096x4096 matrices.

# pR Overhead is ~Constant and Small

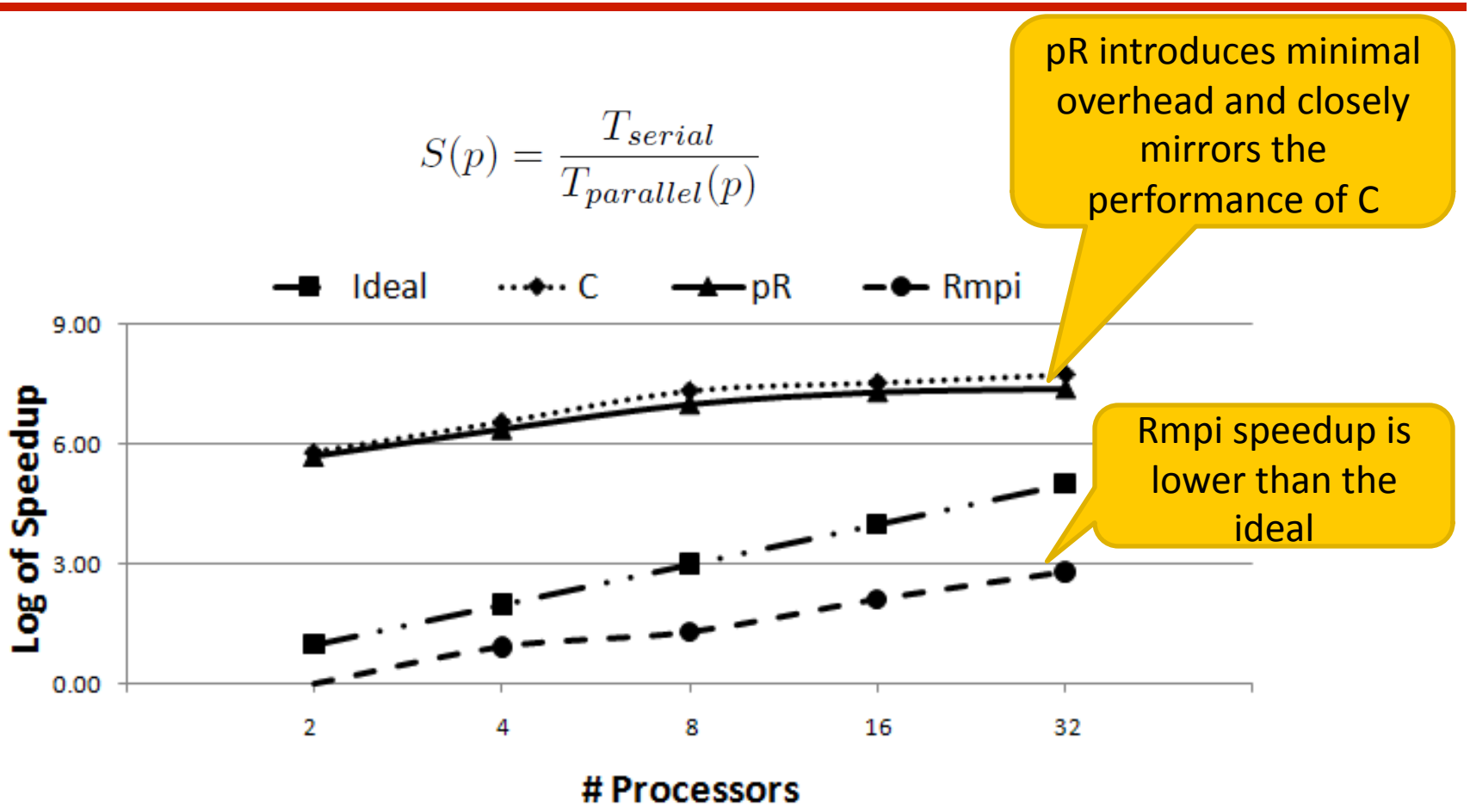
$$O(p) = pR(p) - T(p)$$




Matrix-multiplication testcase on 4096x4096 matrices

# pR Offers Parallel *Scripting* Computing with the Same Performance as Parallel *Compiled* Codes

$$S(p) = \frac{T_{serial}}{T_{parallel}(p)}$$

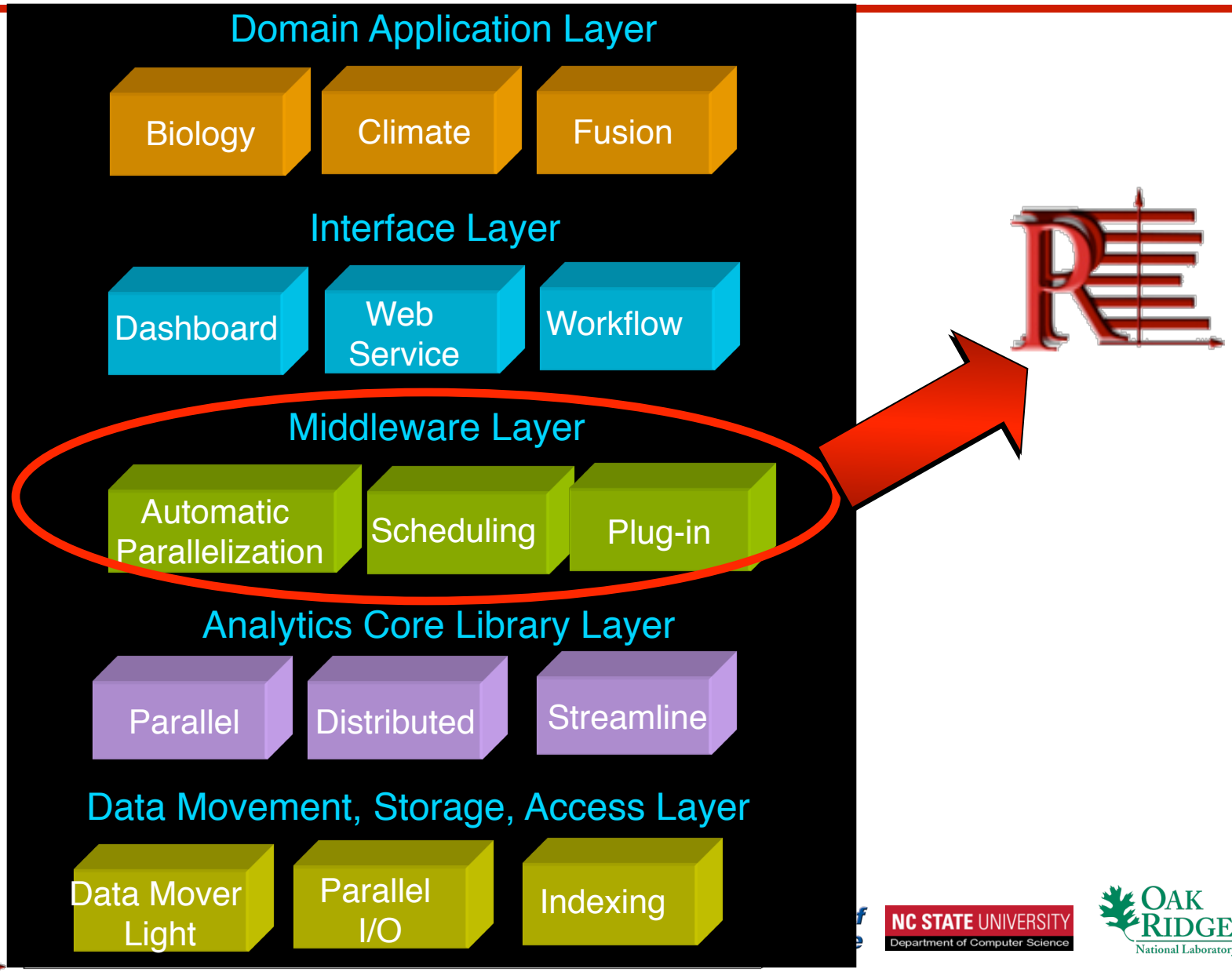



Using a matrix-multiplication testcase on 4096 x 4096 matrices, and comparing against a serial R implementation.

# pR Provides Simple, Efficient Third-Party Access to R

- Tightly-Coupled R Interface Between R and Third-Party Code
- Bidirectional Translation of Data Objects
- Memory Management: Direct Memory Access to R objects
- Compared to R: Average Speedup of 37x (with large variability)
- Compared to C: Negligible Overhead Induced

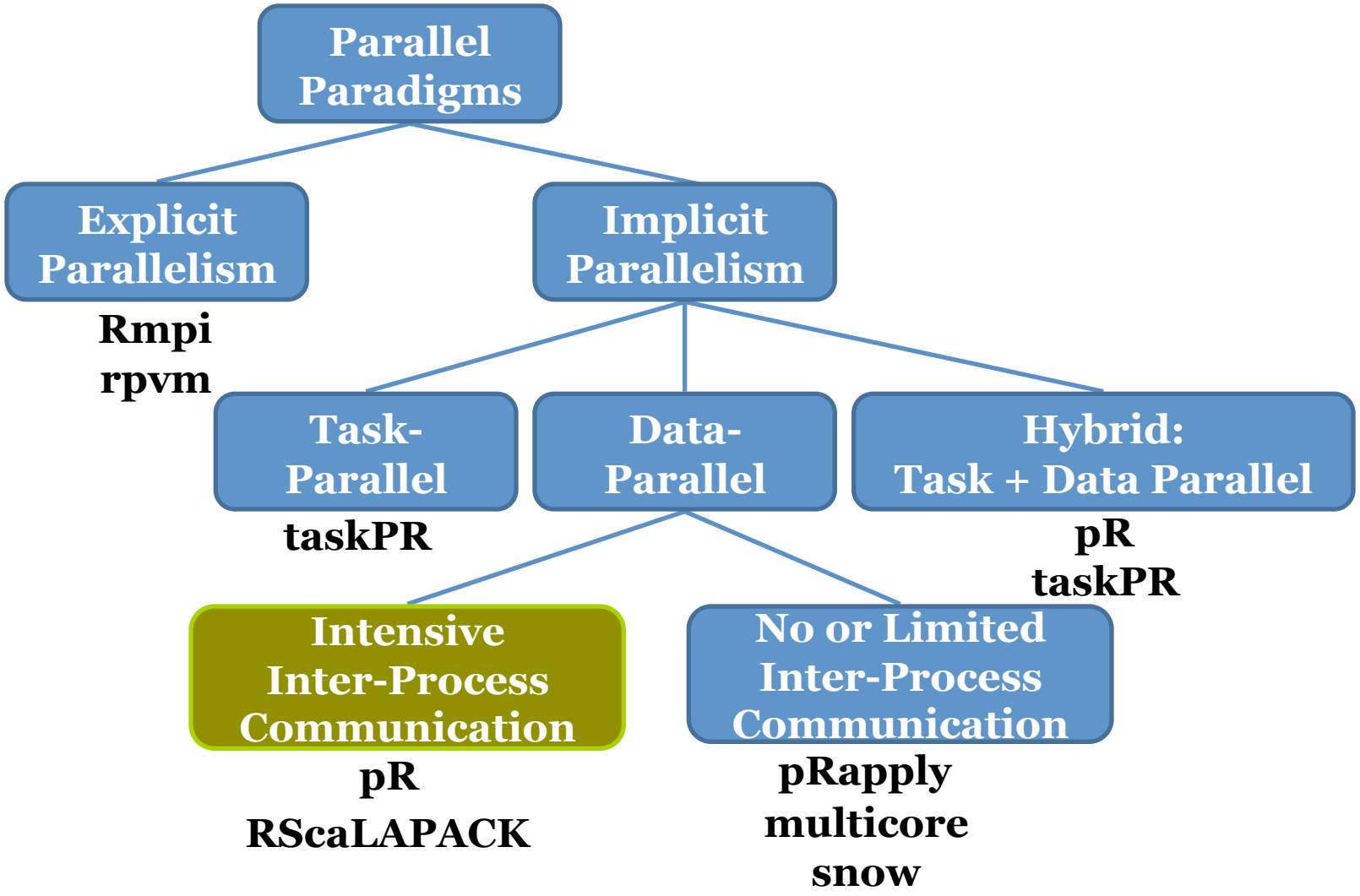
# End-to-End Data Analytics



# pR

- Lightweight middleware to bridge 3rd party codes
- pR-based packages:
  - RScaLAPACK
  - pRapply
  - pRstats
  - ….
- You can add your own libraries using pR

# Parallel Paradigm Hierarchy



# What is RScaLAPACK?

- Motivation:
  - Many data analysis routines call linear algebra functions
  - In R, they are built on top of **serial** LAPACK library: http://www.netlib. org/lapack

- ScaLAPACK:
  - **parallel** LAPACK: http://www.netlib. org/scalapack

- RScaLAPACK is a wrapper library to ScaLAPACK:
  - Also allows to link with ATLAS: http://www.netlib.org/atlas

# Ex: RScaLAPACK Examples

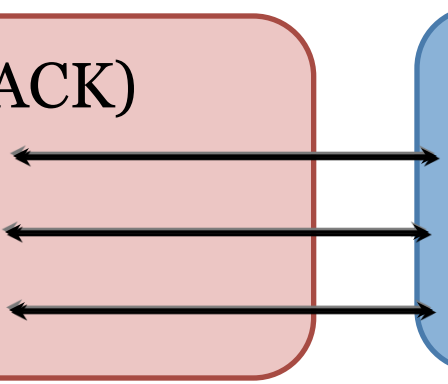
```
A = matrix(rnorm(256),16,16)
b = as.vector(rnorm(16))
```

Using RScaLAPACK:

```
library (RScaLAPACK)
sla.solve (A,b)
sla.svd (A)
sla.prcomp (A)
```

Using R:

```
solve (A,b)
La.svd (A)
prcomp (A)
```



# RScaLAPACK Functions

- library (RScaLAPACK)
- help (package=RScaLAPACK)
- help (sla.solve)  or ?sla.solve
- example (sla.solve)
- demo (RScaLAPACK)

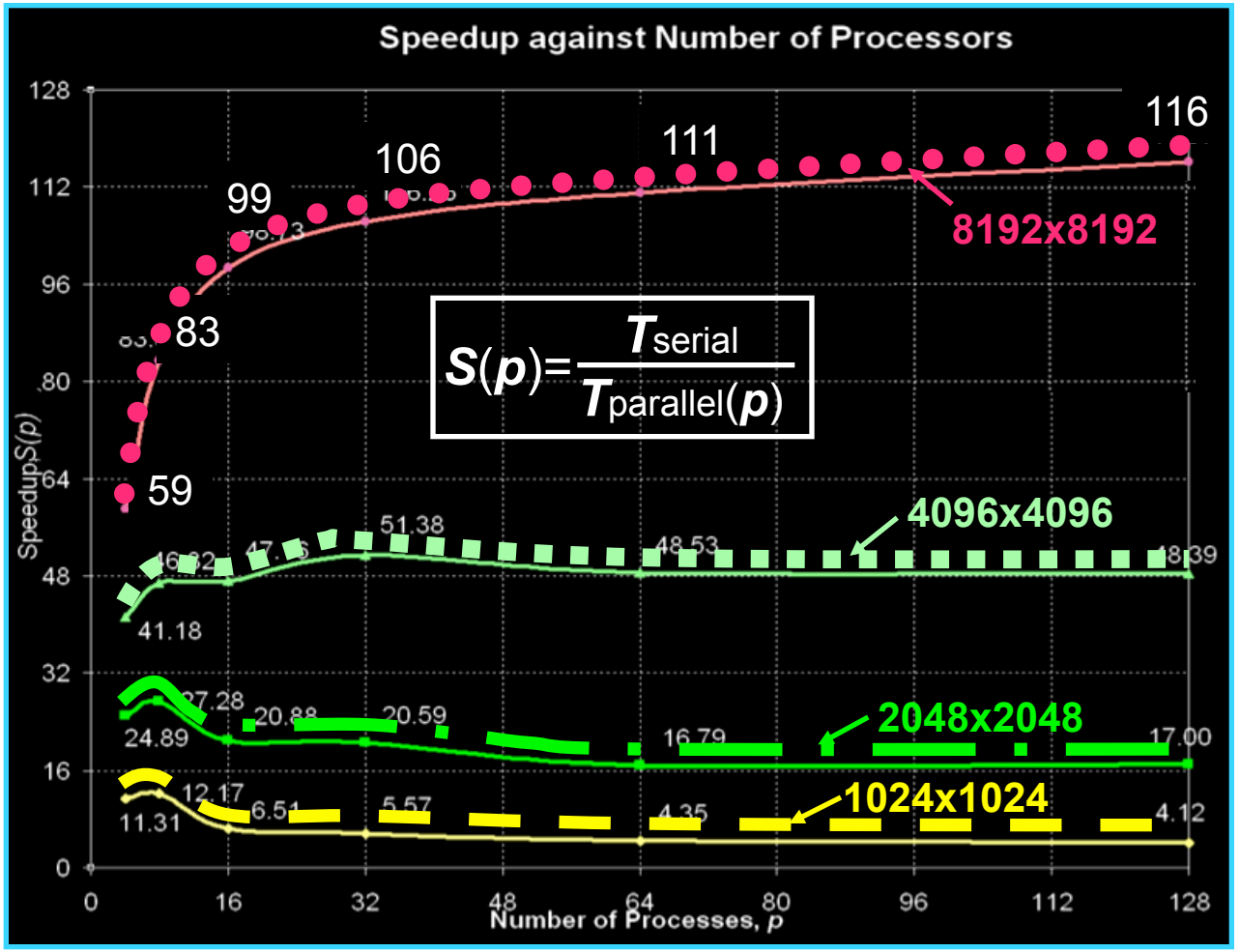
# Currently Supported Functions

| Serial R Functions | Parallel RScaLAPACK | RScaLAPACK Function Description |
|---|---|---|
| **svd** | **sla.svd** | **Compute a singular value decomposition of a rectangular matrix** |
| **eigen** | **sla.eigen** | **Computes the Eigen values and Eigen vectors of symmetric square matrix** |
| **chol** | **sla.chol** | **Computes the Choleski factorization of a real symmetric positive definite square matrix** |
| **chol2inv** | **sla.chol2inv** | **Invert a symmetric, positive definite, square matrix from its Choleski decomposition** |
| **solve** | **sla.solve** | **This generic function solves the equation a*x=b for x** |
| **qr** | **sla.qr** | **computes the QR decomposition of a matrix** |
| **factanal** | **sla.factanal** | **Perform maximum-likelihood factor analysis on a covariance matrix or data matrix using RScaLAPACK functions** |
| **factanal.fit.mle** | **sla.factanal.fit.mle** | **Perform maximum-likelihood factor analysis on a covariance matrix or data matrix using RScaLAPACK functions** |
| **prcomp** | **sla.prcomp** | **performs a principal components analysis on the given data matrix using RScaLAPACK functions** |
| **princomp** | **sla.princomp** | **performs a principal components analysis on the given data matrix using RScaLAPACK functions** |
| **varimax** | **sla.varimax** | **These functions rotate loading matrices in factor analysis using RScaLAPACK functions** |
| **promax** | **sla.promax** | **These functions rotate loading matrices in factor analysis using RScaLAPACK** |

# Scalability of *pR: RScaLAPACK*

***R> solve (A,B)*** ***pR> sla.solve (A, B, NPROWS, NPCOLS, MB)***

***A, B*** are input matrices; ***NPROWS*** and ***NPCOLS*** *are process grid specs;* ***MB*** is block size




Architecture: SGI Altix at CCS of ORNL with **256 Intel Itanium2 processors** at **1.5 GHz**; 8 GB of memory per processor (**2 TB** system memory); 64-bit Linux OS; 1.5 TeraFLOPs/s theoretical total peak performance.

# Changing the Processor Grid in RScaLAPACK

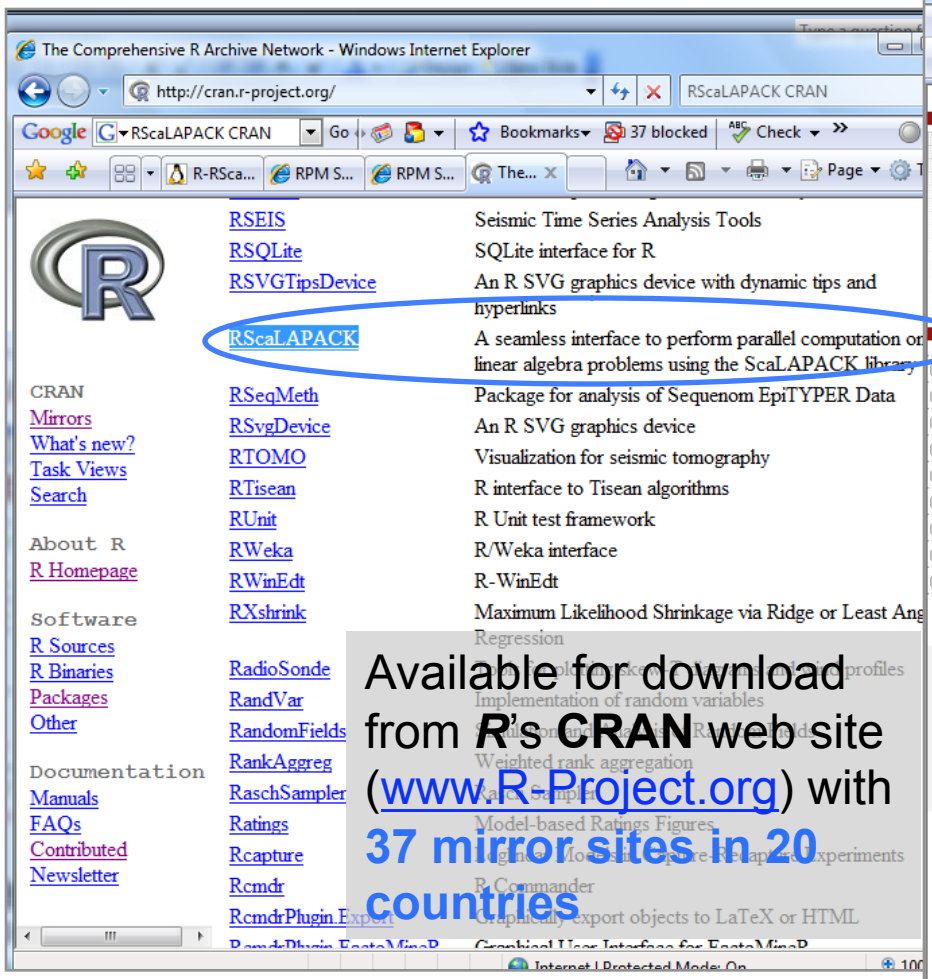
```
library (RScaLAPACK)
A = matrix(rnorm(128*128),128,128)
?sla.gridInit
```

Changing processor grid:

```
sla.gridInit(NPROCS=8)
x = sla.solve (A, NPROWS=4)
sla.gridExit()
```

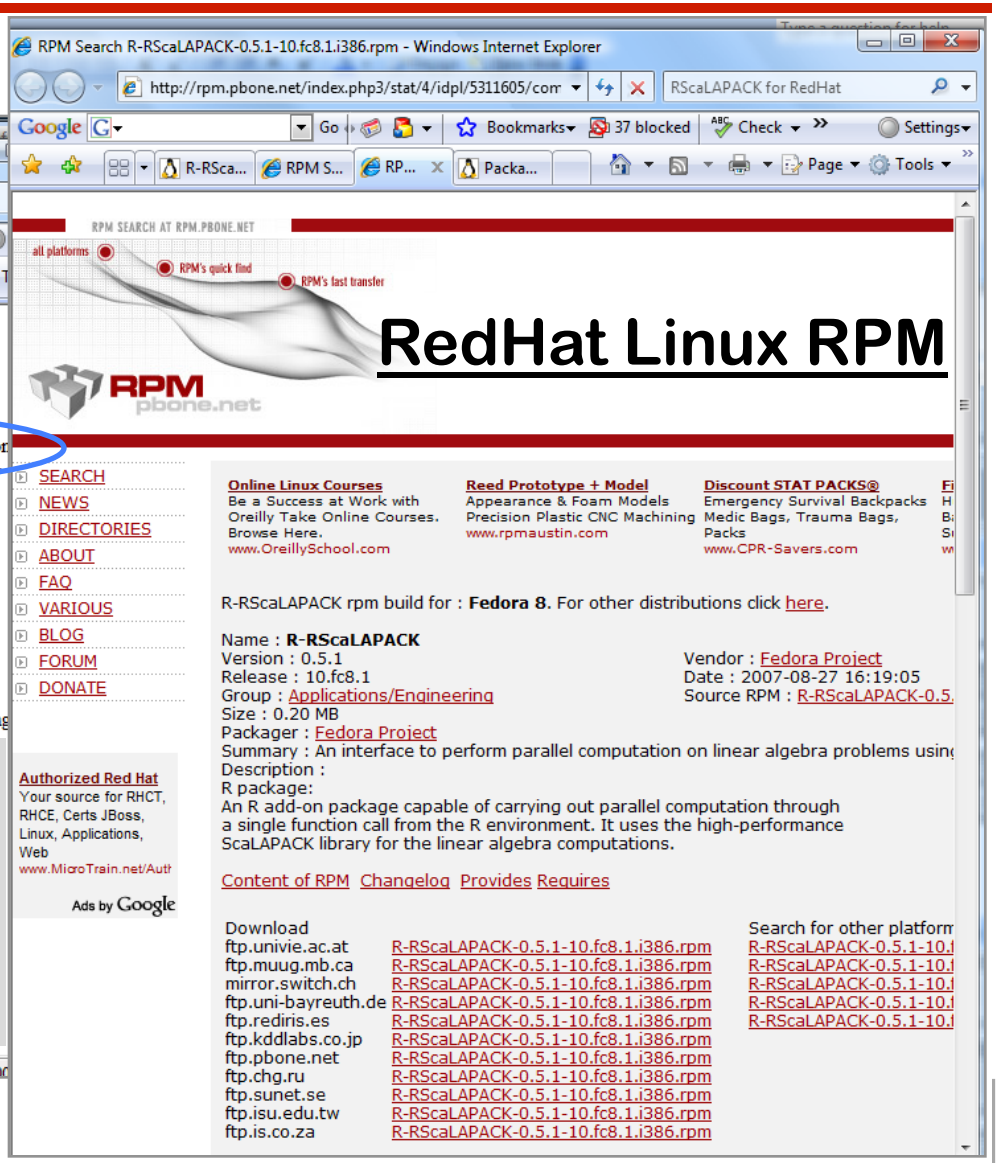
# RedHat and CRAN Distribution

## CRAN R-Project




Available for download from ***R***'s **CRAN** web site (www.R-Project.org) with **37 mirror sites in 20 countries**

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



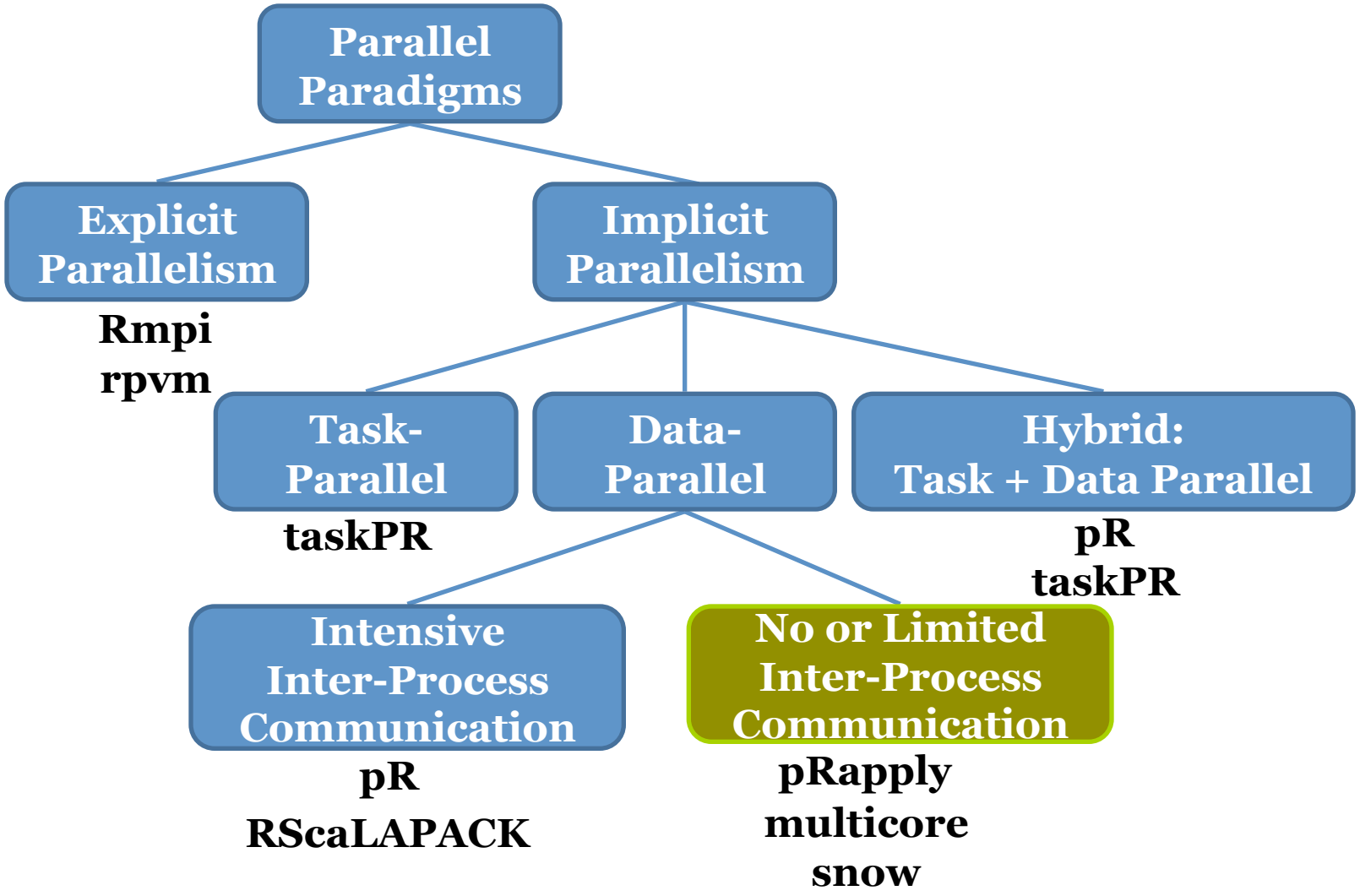

http://rpmfind.net/linux/RPM/RByName.html

# RScaLAPACK Installation

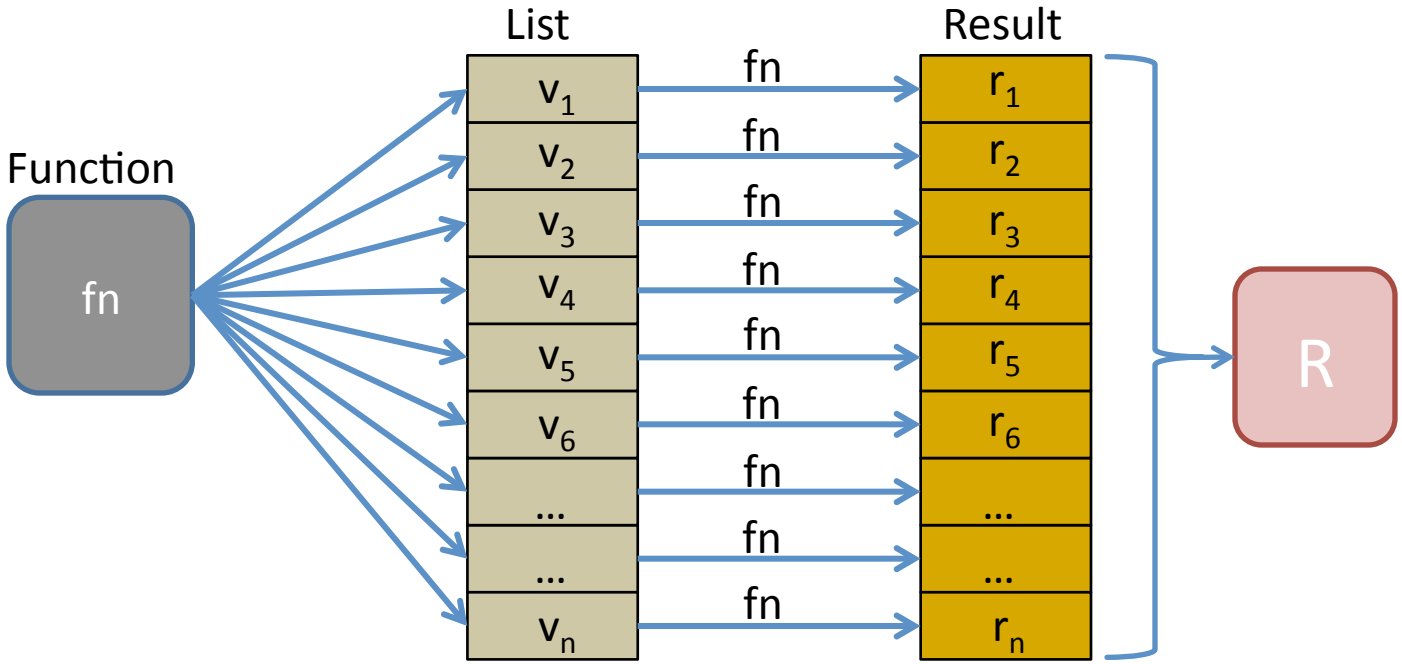
- **Download** RscaLAPACK from R's CRAN web-site
- **Install dependency packages**:
  - Install R
  - MPI (Open MPI, MPICH, LAM MPI)
  - ScaLAPACK (with the proper MPI distribution)
  - Setup environment variables

    *export LD_LIBRARY_PATH=<path2deps>/lib:$LD_LIBRARY_PATH*
- **Install RScaLAPACK**:
  - R CMD INSTALL *--configure-args="--with-f77 --with-mpi=<MPI install home directory> --with-blacs=<blacs build>/lib --with-blas=<blas build>/lib --with-lapack=<lapack build>/lib --with-scalapack=<scalapack build>/lib" RScaLAPACK_0.6.1.tar.gz*

# Parallel Paradigm Hierarchy



# R's *lapply* Method is a Natural Candidate for Automatic Parallelization




- Examples: Bootstrapping, Monte Carlo, etc.

# Existing R Packages with Parallel *lapply*

- multicore
  - Limited to single-node execution
  - *mclapply*

- snow
  - Built on Rmpi – uses MPI for communication
  - Requires users to explicitly manage R dependencies (libraries, variables, functions)
  - *clusterApply*

# snow Example

R End-User

```
1  library(snow);
2  library(abind);
3  x = as.list(1:16);
4  axis=0;
5  y = matrix(1:12,3,4);
6  fn <- function(){
7     z = y+100;
8     b = dim(abind(y,z,along=axis))
9  }
10
11 cl = makeCluster(numProcs, type = "MPI")
12 clusterApply(cl, x, fn);
13 stopCluster(c1);
```

Explicitly send libraries, functions, and variables

```
1 clusterExport(cl, list(axis, y));
2 clusterEvalQ(cl, library(abind));
```

# pRapply Example

**pRlapply (list, fn, procs=2, cores=2)**

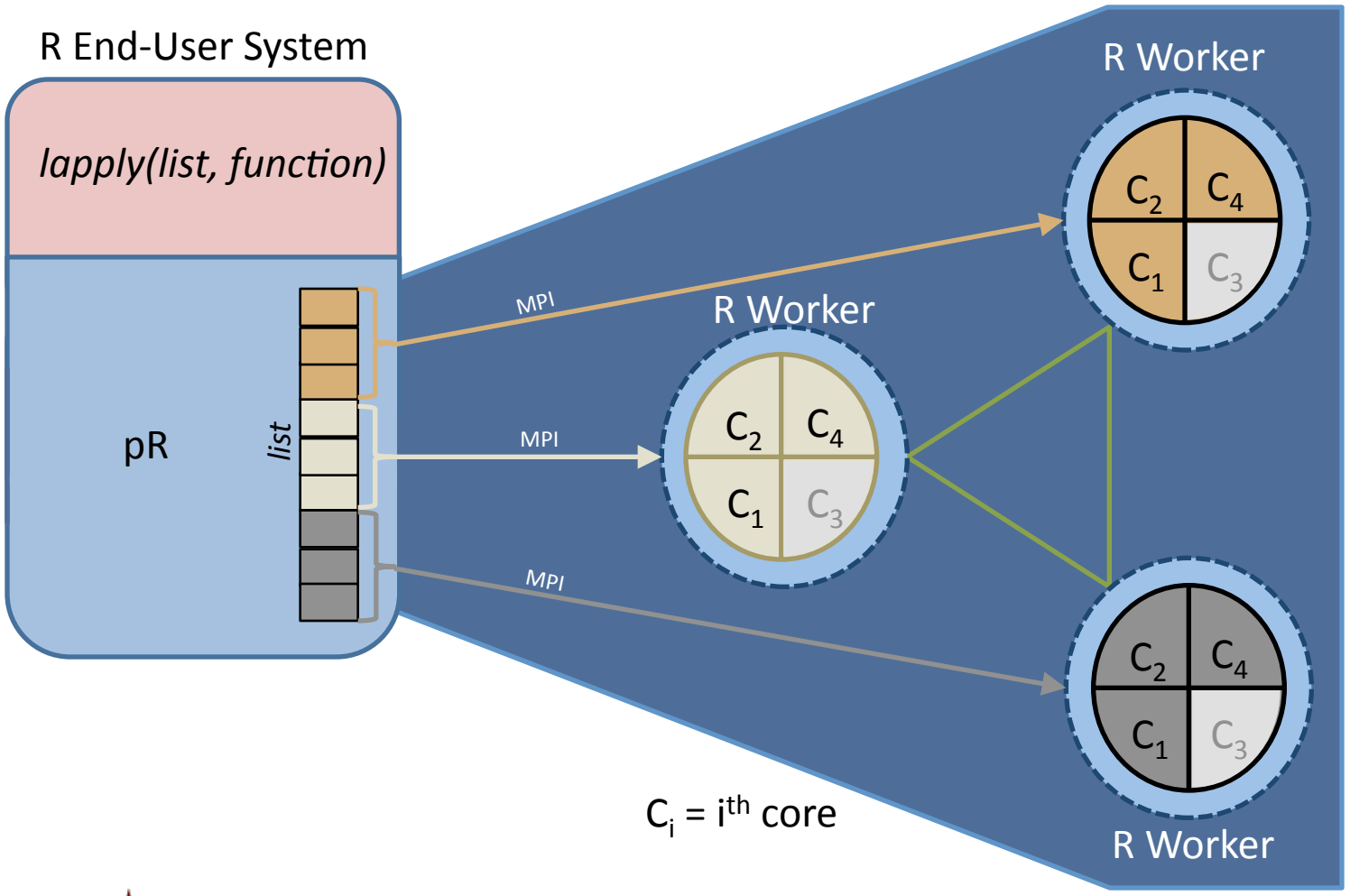
Using pRapply:

```
1  library(pRapply);
2  library(abind);
3  x = as.list(1:16);
4  axis=0;
5  y = matrix(1:12,3,4);
6  fn <- function(){
7     z = y+100;
8     b = dim(abind(y,z,along=axis))
9  }
10
11 pRlapply(x, fn)
```

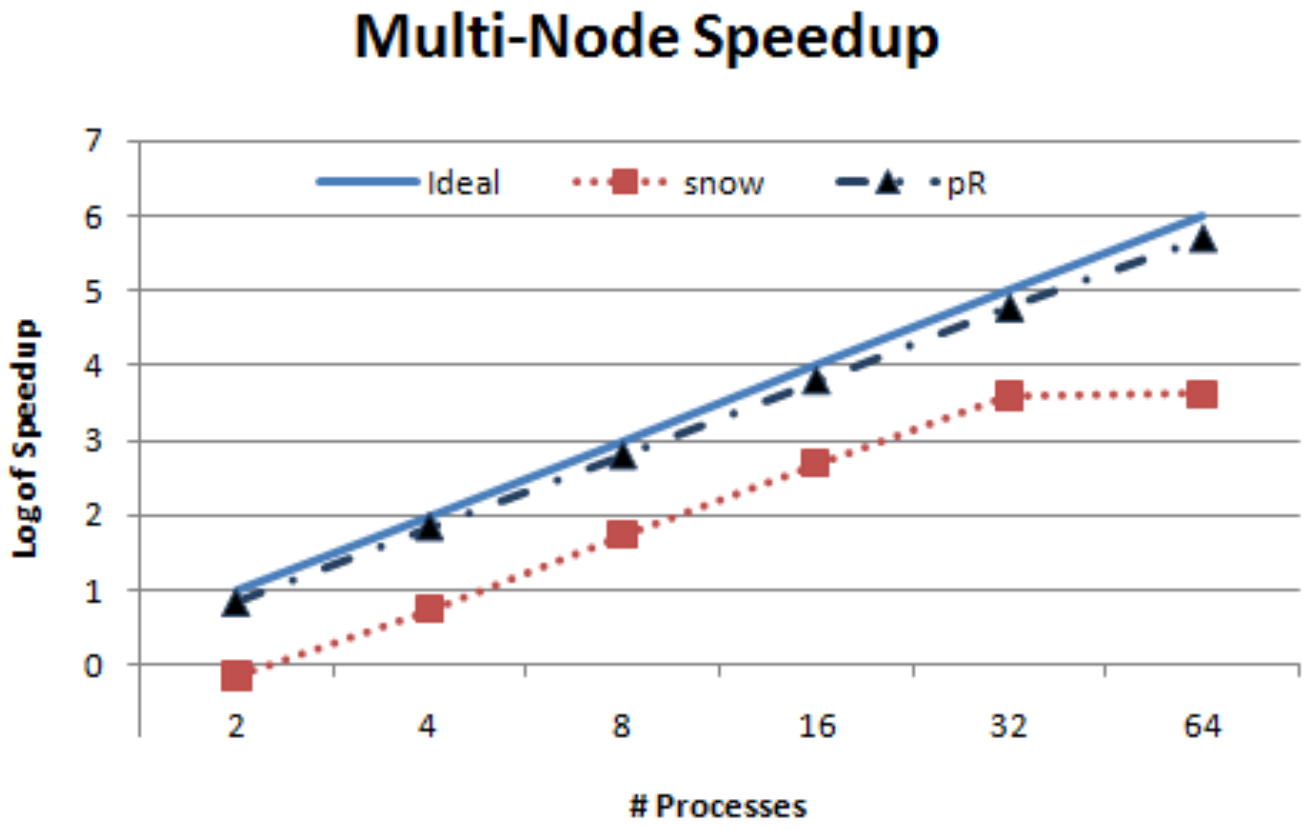
Using R:

```
1
2  library(abind);
3  x = as.list(1:16);
4  axis=0;
5  y = matrix(1:12,3,4);
6  fn <- function(){
7     z = y+100;
8     b = dim(abind(y,z,along=axis))
9  }
10
11 lapply(x, fn)
```

# pR Automatic Parallelization Uses a 2-Tier Execution Strategy



# pRapply: ~Ideal Speedup vs. snow




Multi-node speedup for pR and snow compared to the serial R implementation using a testcase that sums 4096 vectors, each of size $2^{24}$.

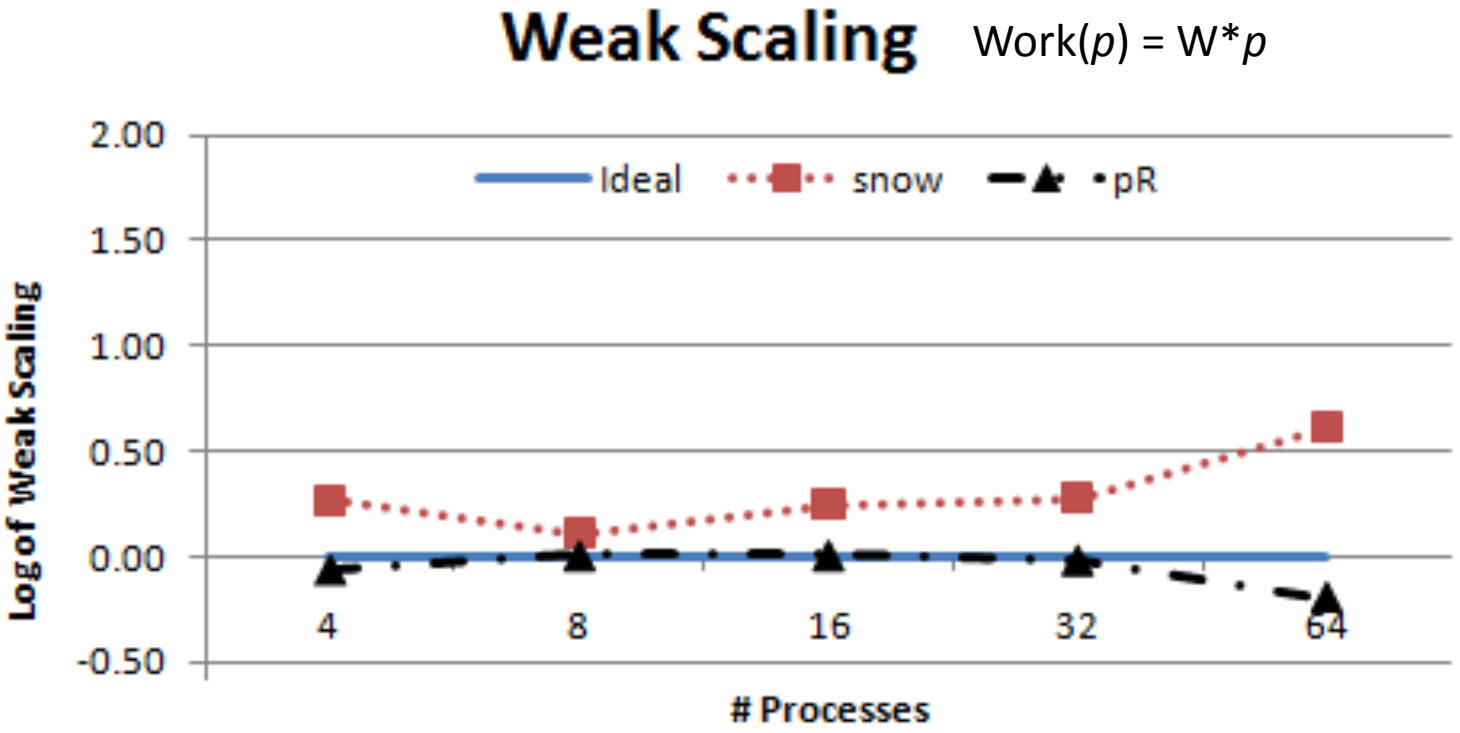
SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

# pRapply: Ideal or Better Weak Scaling




Weak scaling comparison of snow and pR in a 64-node hybrid multi-node multi-core environment; each process sums a $2^{24}$ element vector.

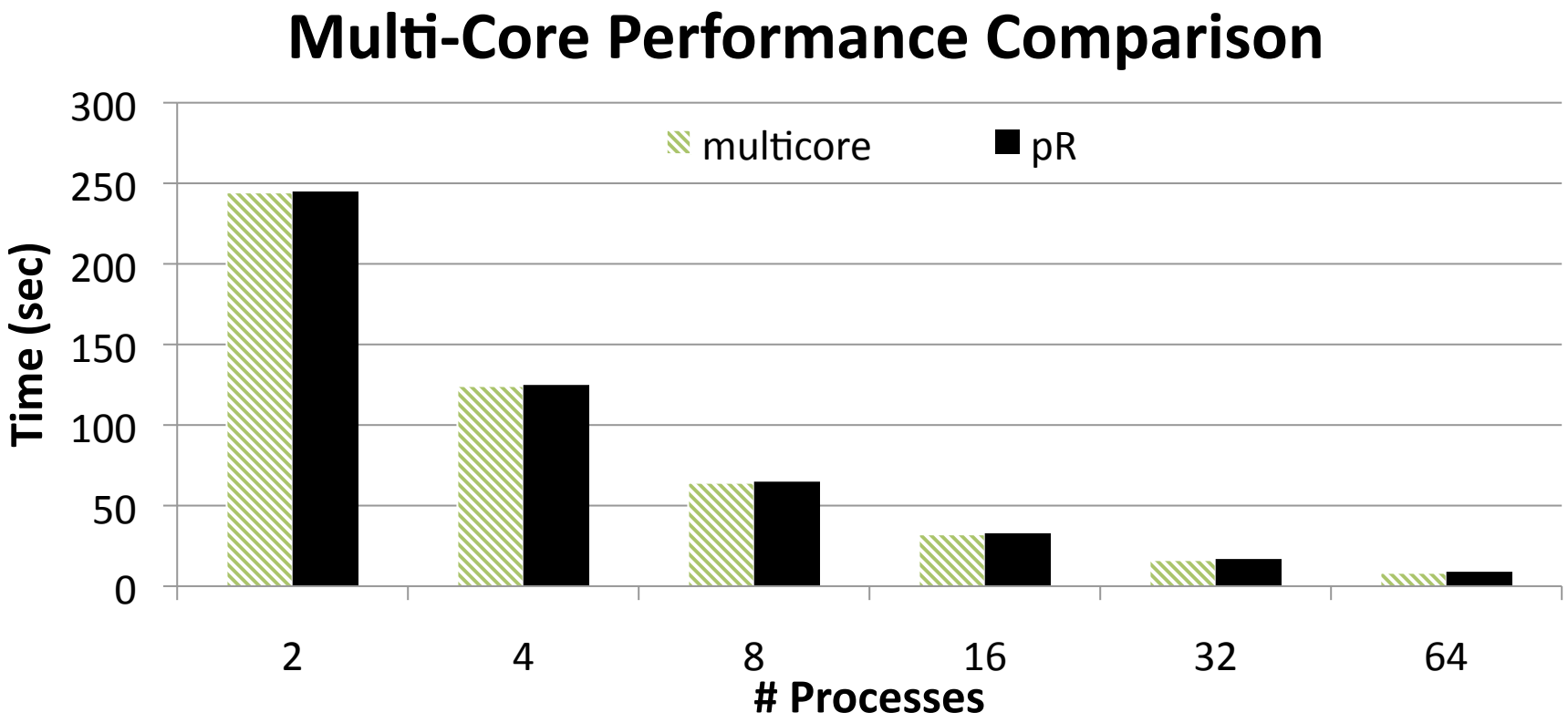
SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

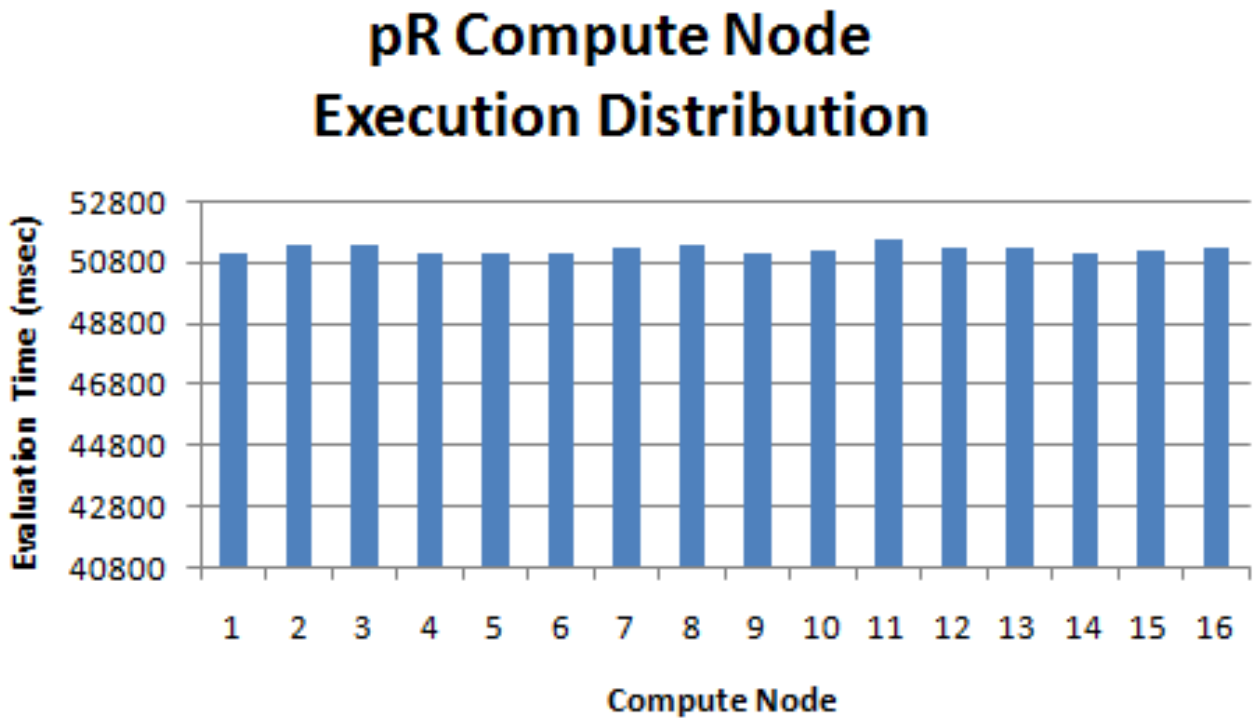
OAK RIDGE National Laboratory

# pRapply: Like *multicore*, but Supports Hybrid Multi-Node, Multi-Core Execution




The pR overhead and total execution time compared to the multicore package over powers of two processes (1-64) using a testcase that *sleeps* 120 times for 4 seconds.

# pR Achieves Load-Balanced Performance




pR compute node execution heterogeneity in a hybrid multi-node multi-core Environment using 16 nodes to each sum a $2^{24}$ element vector.

# pR Major Contributions

- Enables integrating efficient serial codes into R using in-memory data transfers
- Provides parallel computing capabilities within R
- Enables automatic parallelization of data-parallel codes in hybrid multi-core and multi-node environments

# Acknowldgements


- Guruprasad Kora, ORNL
- Paul Breimyer, Lincoln Lab
- Srikanth Yoginath, ORNL
- David Bauer, GA Tech

# Publications


1. *pR: Efficient and Scalable Parallel R for High-Performance Statistical Computing;* The International Conference on Information and Knowledge Engineering (IKE) 2009; Paul Breimyer, Guruprasad Kora, William Hendrix, Nagiza F. Samatova
2. *Parallel R for High Performance Analytics: Applications to Biology*; Scientific Data Management (Book In Press), 2008; Nagiza F. Samatova, Paul Breimyer, Guruprasad Kora, Chongle Pan, Srikanth Yoginath; Editors: Arie Shoshani, Doron Rotem, Chandrika Kamath
3. *pR: Automatic Parallelization of Data-Parallel Statistical Computing Codes for R in Hybrid Multi-Node and Multi-Core Environments*; International Association for Development of Information Society (IADIS) 2009; Paul Breimyer, Guruprasad Kora, William Hendrix, Nagiza F. Samatova
4. RScaLAPACK on CRAN: http://cran.r-project.org/mirrors.html
5. *High performance statistical computing with parallel R: applications to biology and climate modelling,* Journal of Physics: Conference Series 46 (2006) 505–509; Samatova NF, Branstetter M, Ganguly AR, Hettich R, Khan S, Kora G, Li J, Ma X, Pan C, Shoshani A, Yoginath S,
6. *RScaLAPACK: High-performance parallel statistical computing with R and ScaLAPACK.* Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems (PDCS-2005), Sep 12-14, 2005, Las Vegas, Nevada; Yoginath S, Samatova NF, Bauer D, Kora G, Fann G, Geist A.


SDM
CENTER


Office of
Science
U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY
Department of Computer Science

OAK
RIDGE
National Laboratory

# Session 2: Input and Output

## To input data:

```
# scan() command to scan data
 x=scan()
# enter numbers, one or more on a
line, and type a blank line to end
```

```
# read.table() command to read
data from file
 y=read.table("myfile")
```

```
 # rep() command to create data
that follow a regular pattern
 b=rep(1,5)
 c=rep(1:2,4)
```

## To access vector elements:

```
# 2nd element of x
 x[2]
# first five elements of x
 x[1:5]
# all but the 3rd element of x
 x[-3]
 # values of x that are < 40
 x[x<40]
# values of y such that x is < 40
 y[x<40]
```

## To perform operations:

```
 # mathematical operations on vectors
y=c(3,2,4,3,7,6,1,1)
 x+y; 2*y;  x*y;  x/y;  y^2
```

# Session 4: Plotting in R

- To create a vector:
  - x=c(12,32,54,33,21,65) # help (c)