pR: Introduction to Parallel R for Statistical Computing

CScADS Scientific Data and Analytics for Petascale Computing Workshop

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

7/27/2010

Outline

- Application Driver
- Introduction to Parallel R
- Analytics-Aware Parallel I/O
- In situ Analytics in Staging

Contributors & Collaborators

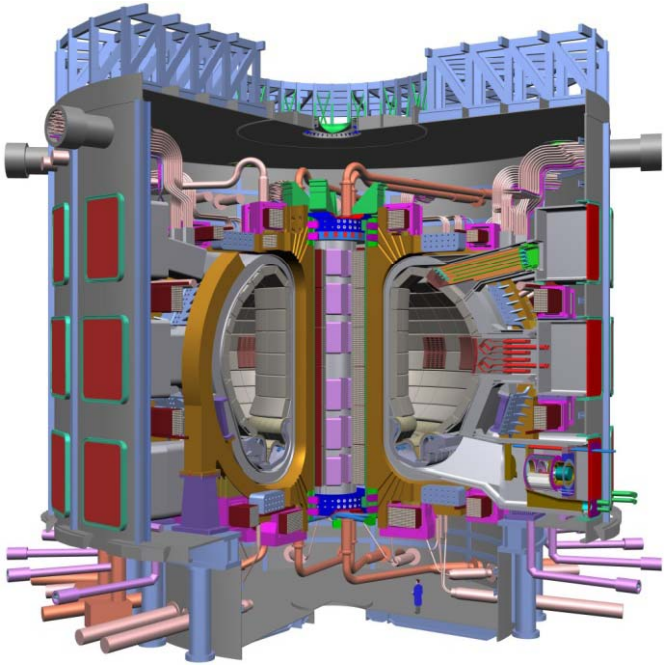
- **Front detection/tracking**: CS Chang (NYU), Neil Shah and Katie Shpanskaya (High School Students)
- **pR**: Guru Kora (ORNL), Paul Breimyer (NCSU)
- **ADIOS**: Scott Klasky, Nobert Podhorszki, Qing (Gary) Liu (ORNL); Jay Lofstead (GA Tech); Sriram Lakshminarasimhan, Abhijit Sachidananda, Michael Warren (NCSU)

Outline

- Application Driver
- Introduction to Parallel R
- Analytics-Aware Parallel I/O
- In situ Analytics in Staging

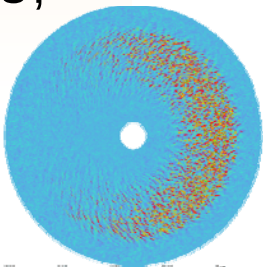
The Fusion Energy Challenge
How to "Hold the Sun?"

Challenge: **How to sustain fusion reaction to produce net energy profit?**



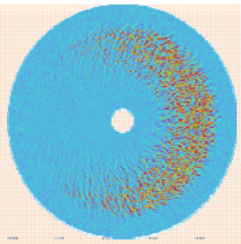
The ITER Tokamak

- **Maintain high enough temperature of plasma for prolonged time.**
- **Maintain magnetic confinement of plasma.**
- **Eliminate detrimental** plasma instabilities, or turbulence.



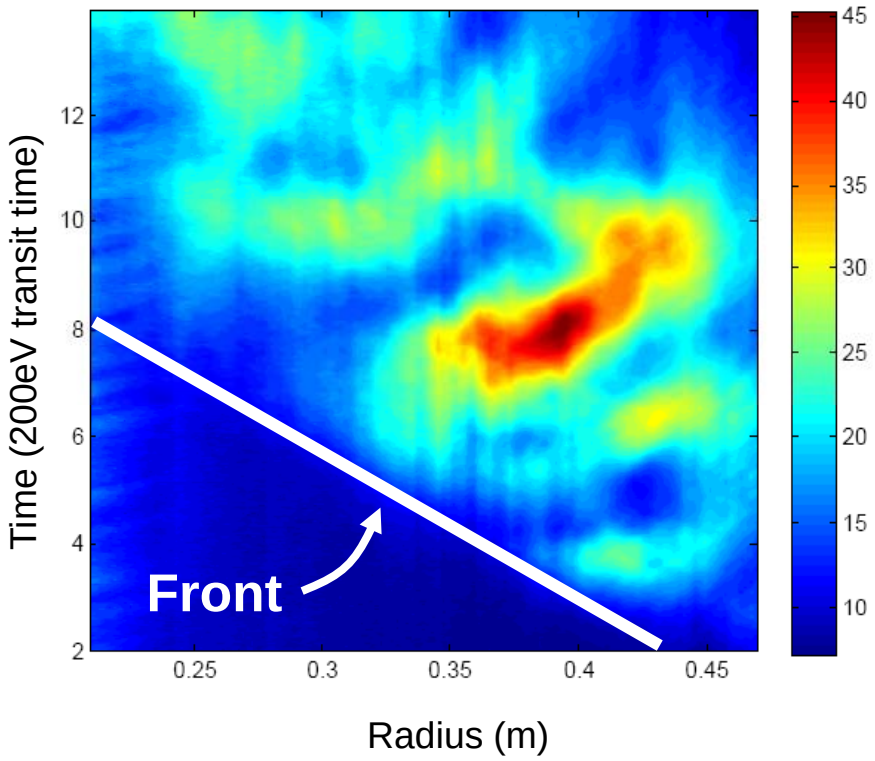
What is a Dynamic Turbulent Front?

Spatio-Temporal Front Propagation

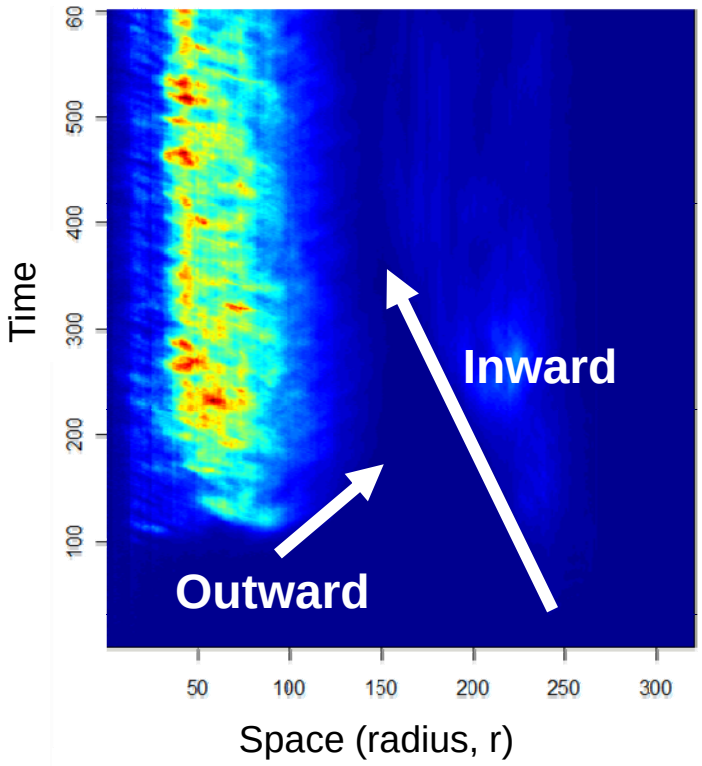


Informally, turbulent fronts are patterns, where the turbulence starts.

Potential Energy Fluctuation



Propagation in Space and Time

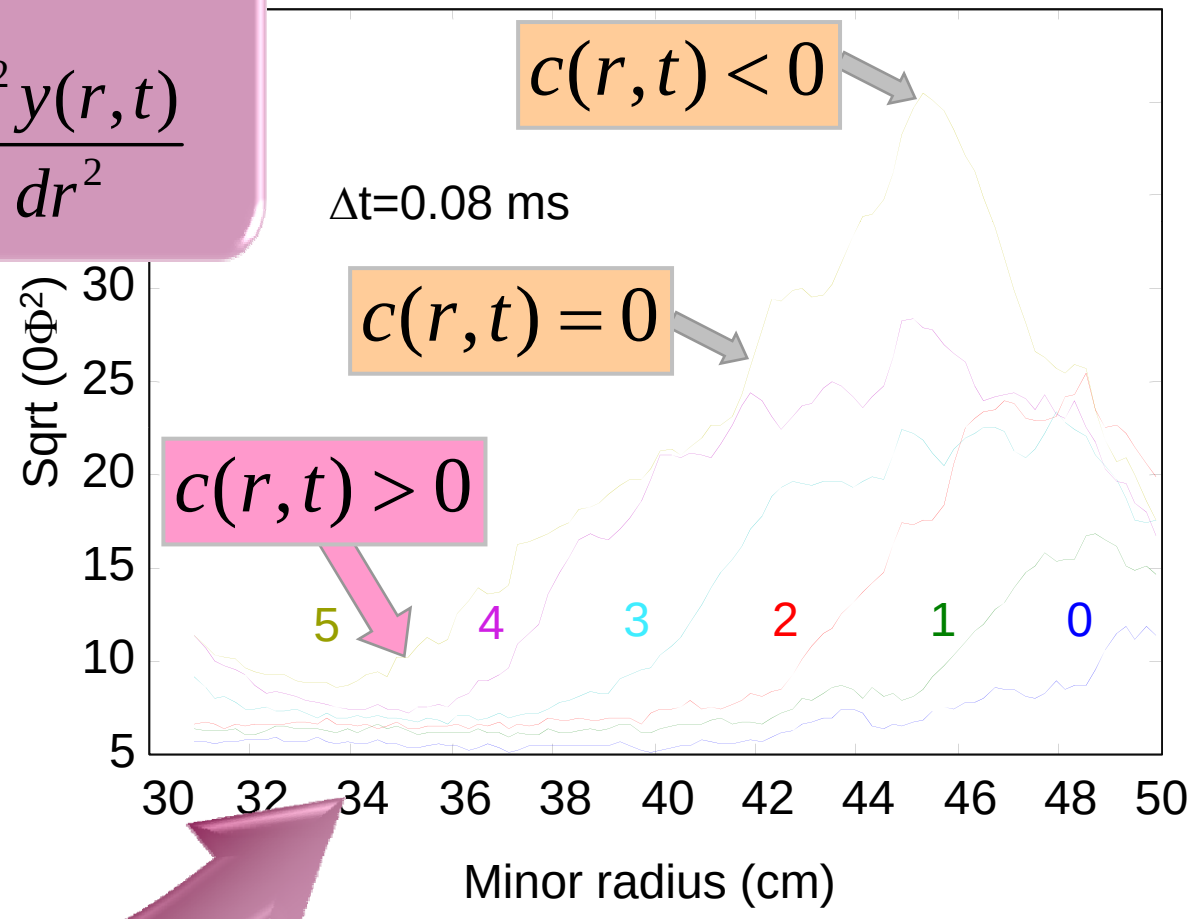


How to Define Fronts Mathematically?
Maximum Positive Curvature Points

$$r_{front}(t) = \arg\max_{r:\, c(r,t)>0} c(r,t)$$

$$= \arg\max_{r:\, c(r,t)>0} \frac{d^2 y(r,t)}{dr^2}$$

$$y(r,t) = \delta\phi^2(r,t)$$



Radial Front

$$r_{front}(t=5) = 35$$

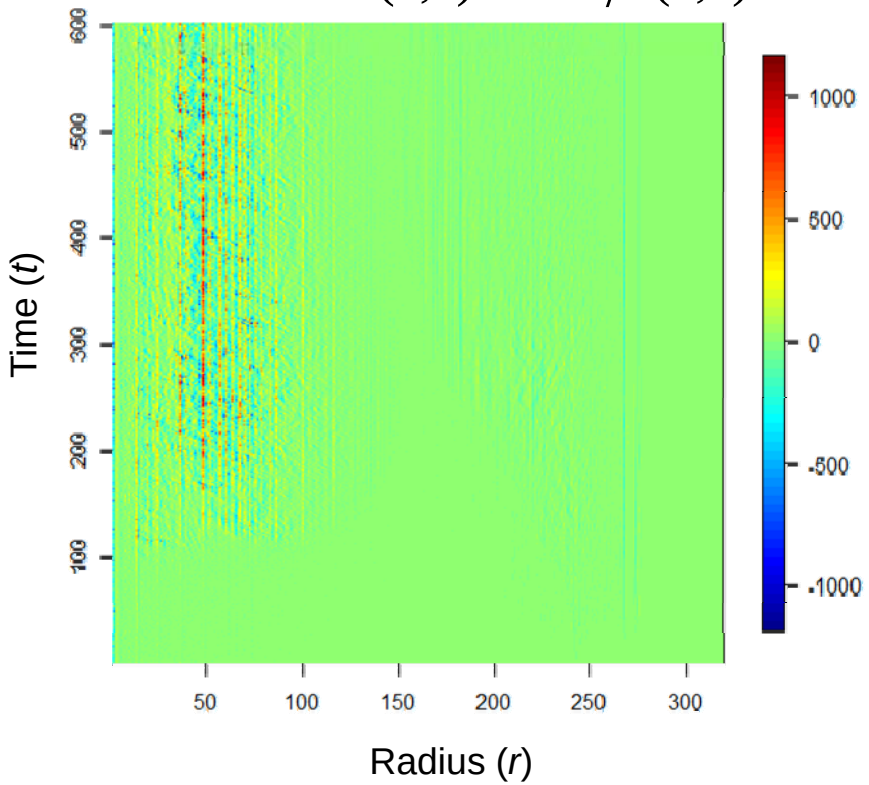
The Problem
Finding the Maxima in Noisy Data

Curvature function calculations from numerical simulation output data produced noisy patterns that challenged finding the maxima.

$$c(r,t) \sim \frac{d^2 y}{dr^2}$$

$$y(r,t) = \delta\phi^2(r,t)$$

Curvature $c(r,t)$ for $\delta\phi^2(r,t)$



Our Innovation

Strategy to Discover Fronts in Noisy Data

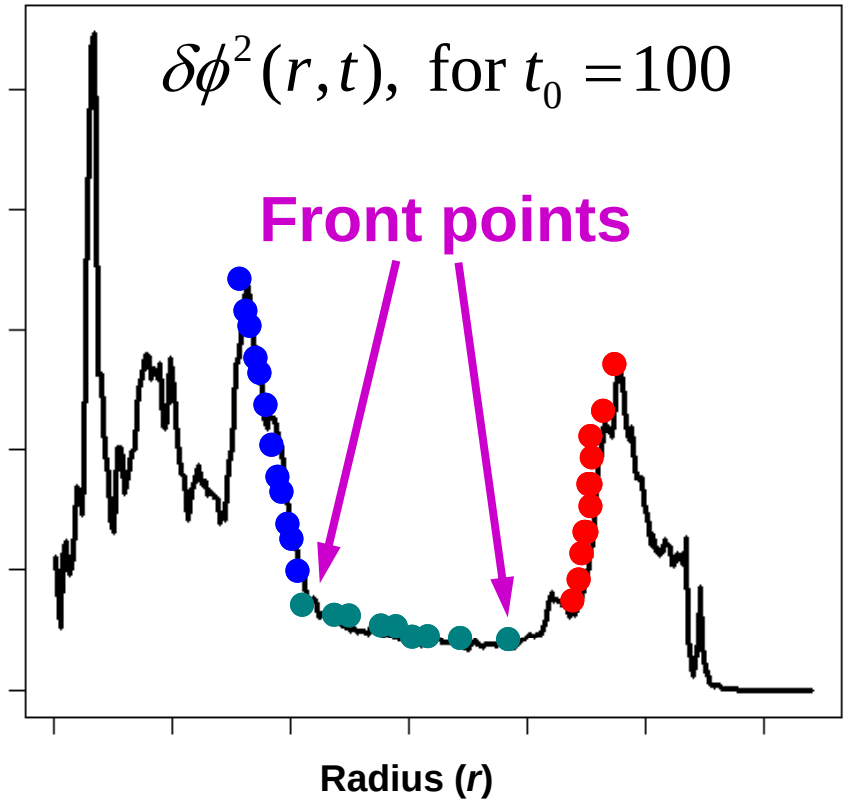
Intuition: The **front** points are where the segments change their ***slope*** from the direction **almost parallel to the x-axis** to the direction **almost parallel to the y-axis**.

Linear Approximation:

For a fixed time-step, t_0, consider the approximation of $\delta\phi^2\ (r,t_0)$ with line segments in a small $(r - \Delta r,\ r + \Delta r)$ spatial region around the point of interest.

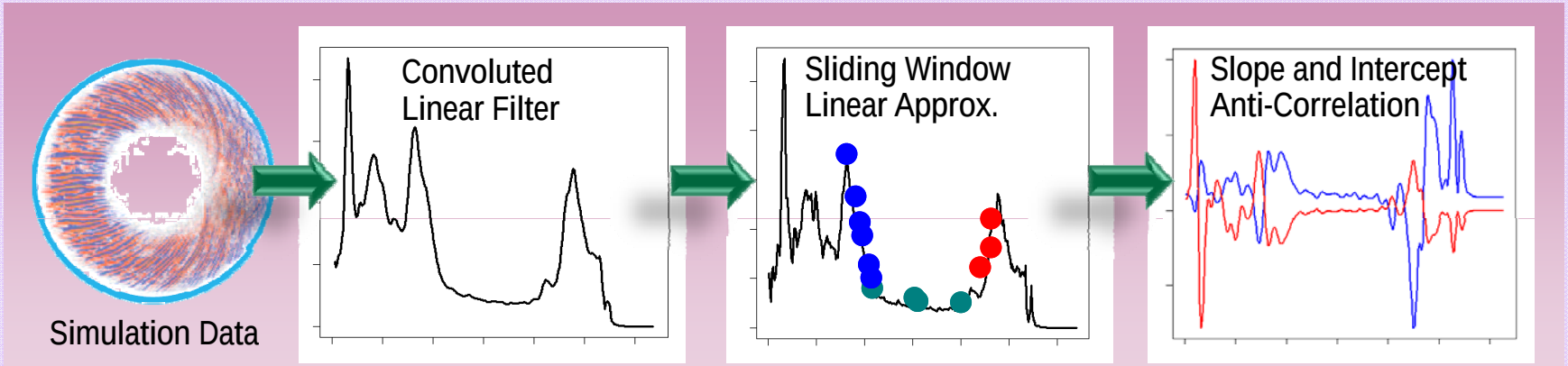
$$l(r) = a \cdot r + b$$

slope **intercept**

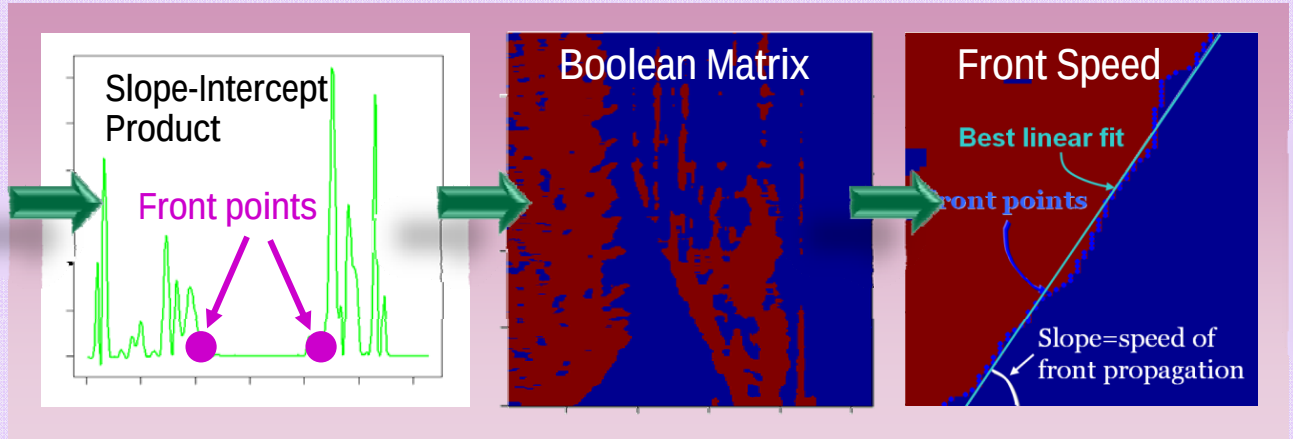


Turbulent Front Detection and Tracking
A Multi-Step Knowledge Discovery Process

Data Pre-Processing Steps



Front Detection and Tracking Steps

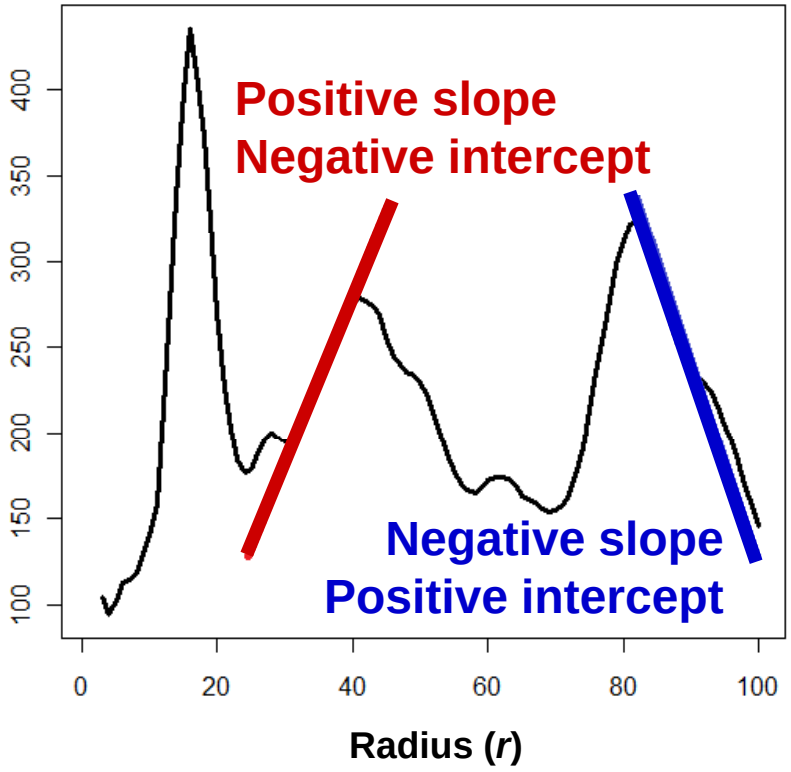


Efficient run is performed by our pRapply parallel system

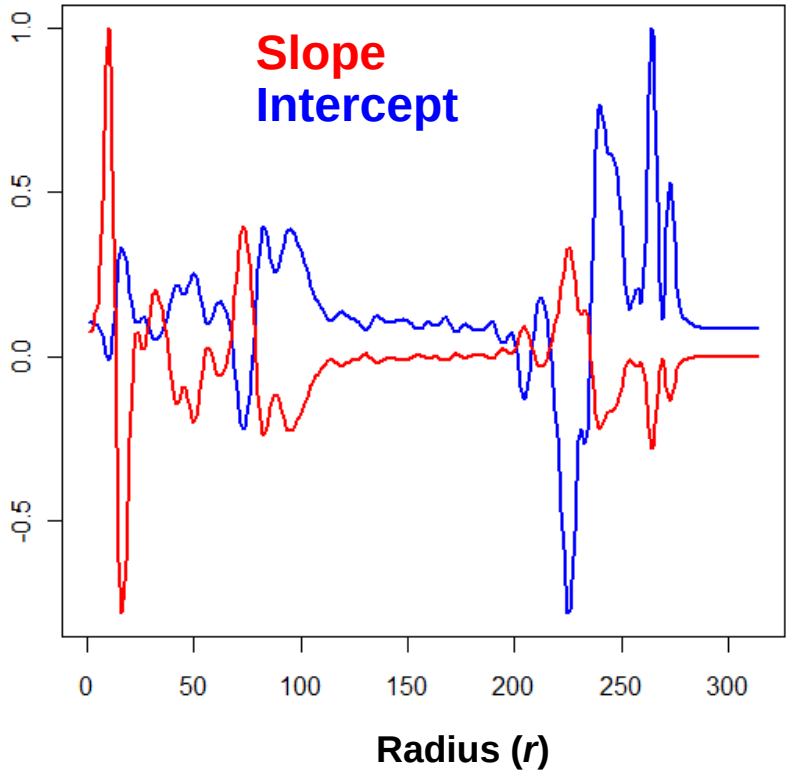
Slope-and-Intercept Anti-Correlation Step
Reversed Positive-Negative Signs

$$l(r) = a \cdot r + b$$

slope intercept



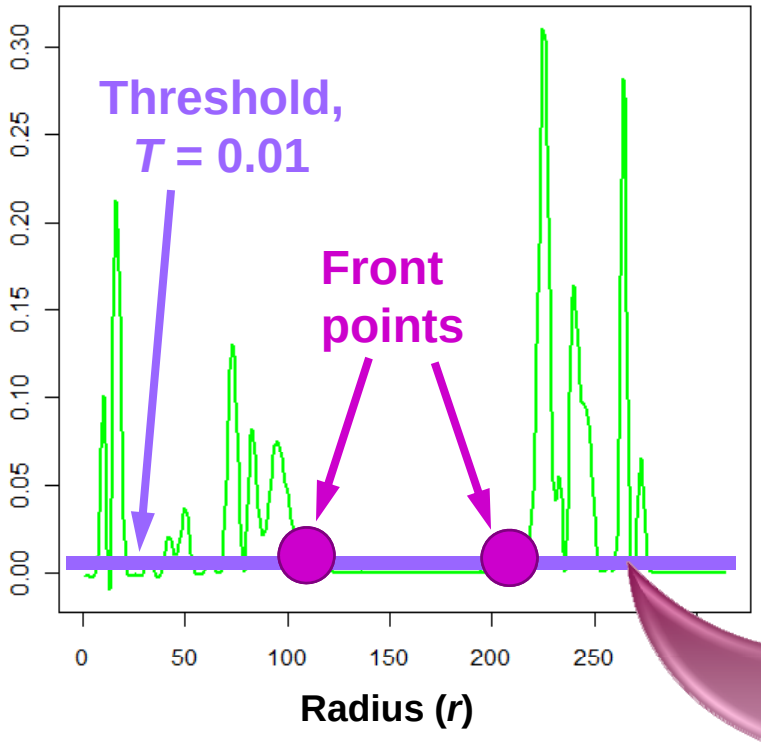
(-1,1)-Normalized Slope and Intercept



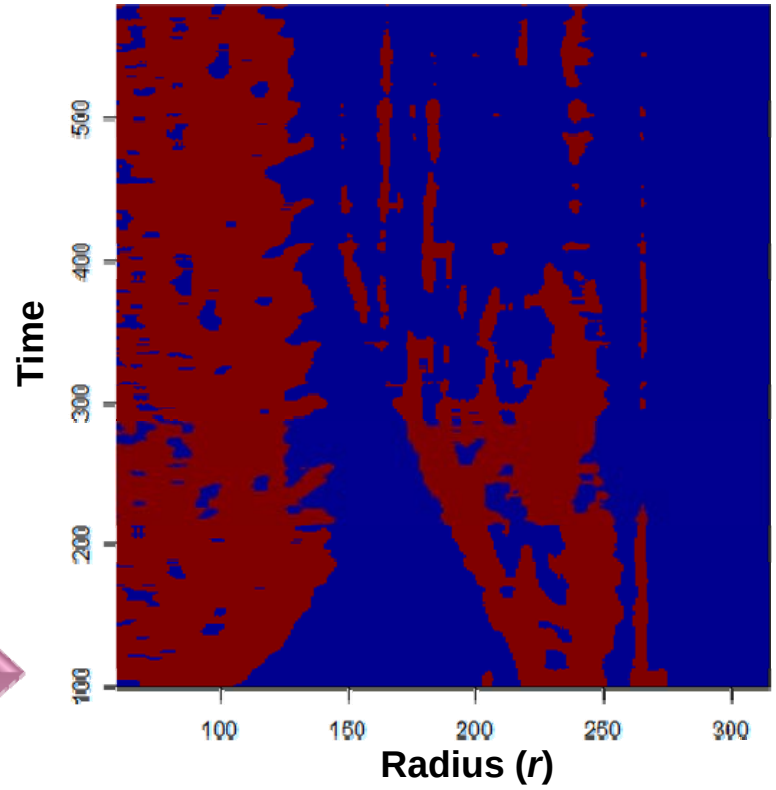
Slope-Intercept Product and Boolean Matrix Steps
With the Clearly Discerned Front Points

Multiplying the normalized slope and the intercept amplified the signal to clearly discern the front points.

Slope-intercept-product "signal"



True-False Boolean Matrix of Thresholded Values

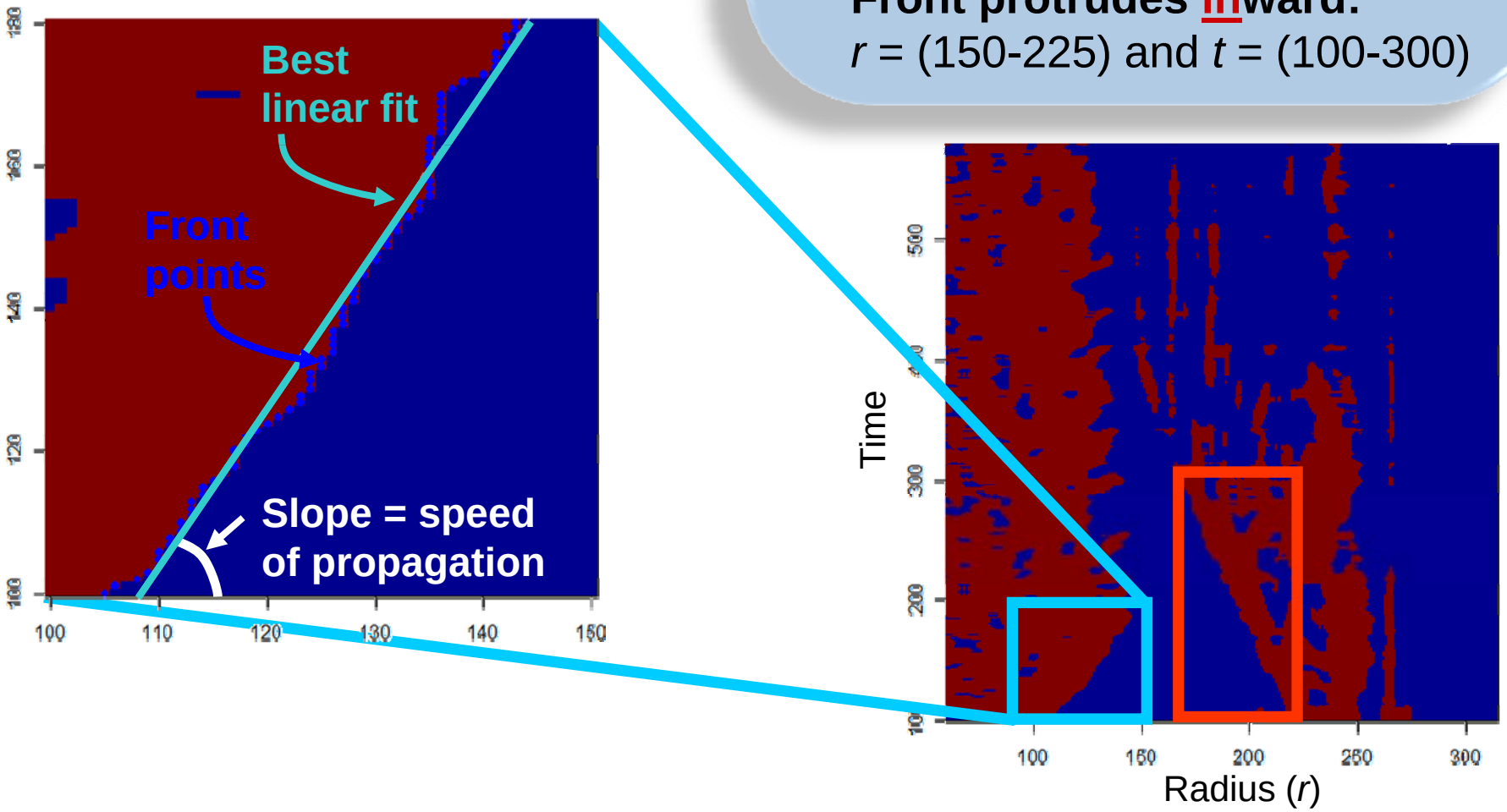


Front Propagation in Space and Time

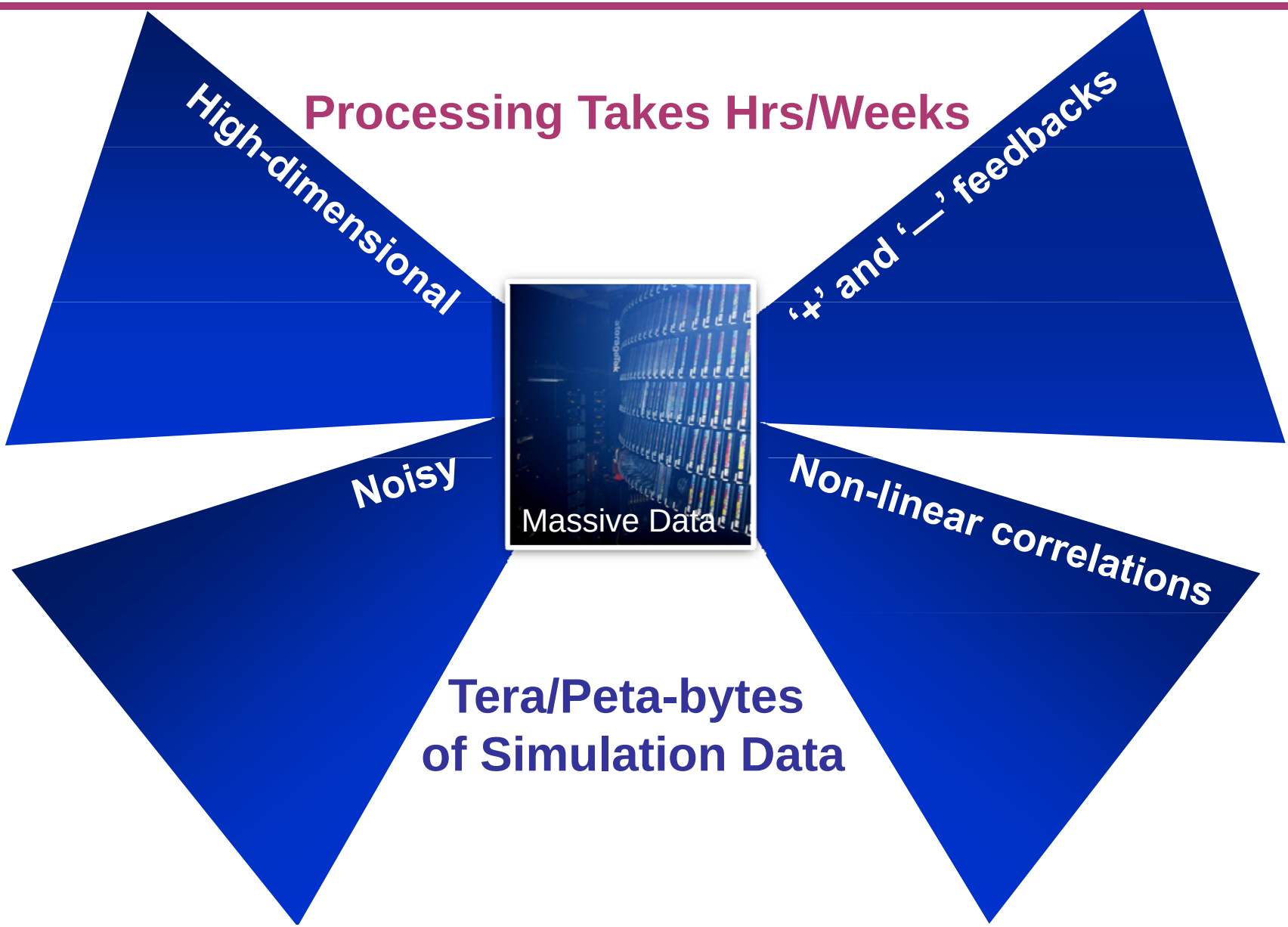
Quantifying the Speed

Discovery:

Front protrudes outward:
r = (100-150) and t = (100-180)
Front protrudes inward:
r = (150-225) and t = (100-300)



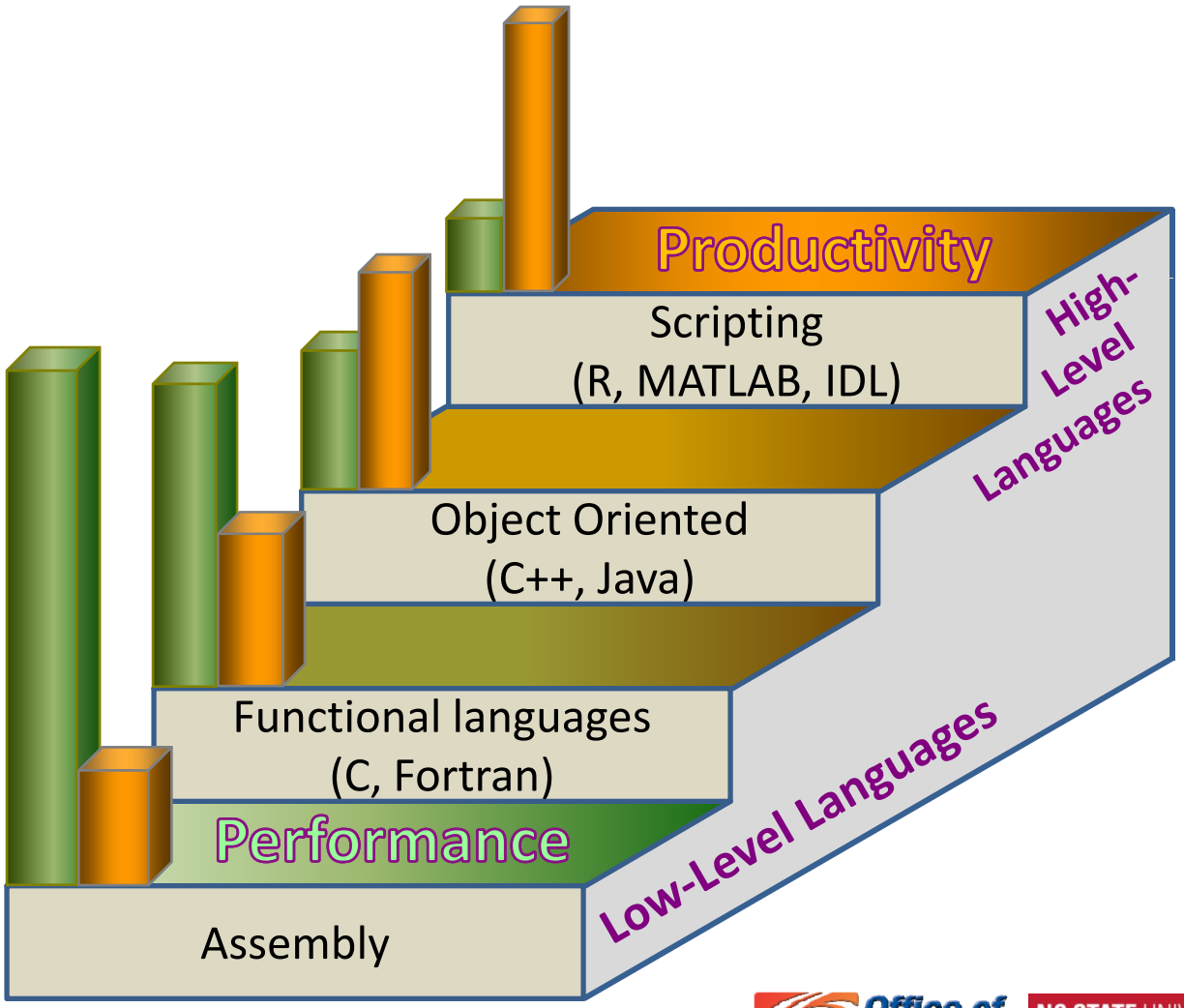
It Was Not Just the Complexity
But the Massive Size of the Data



Outline

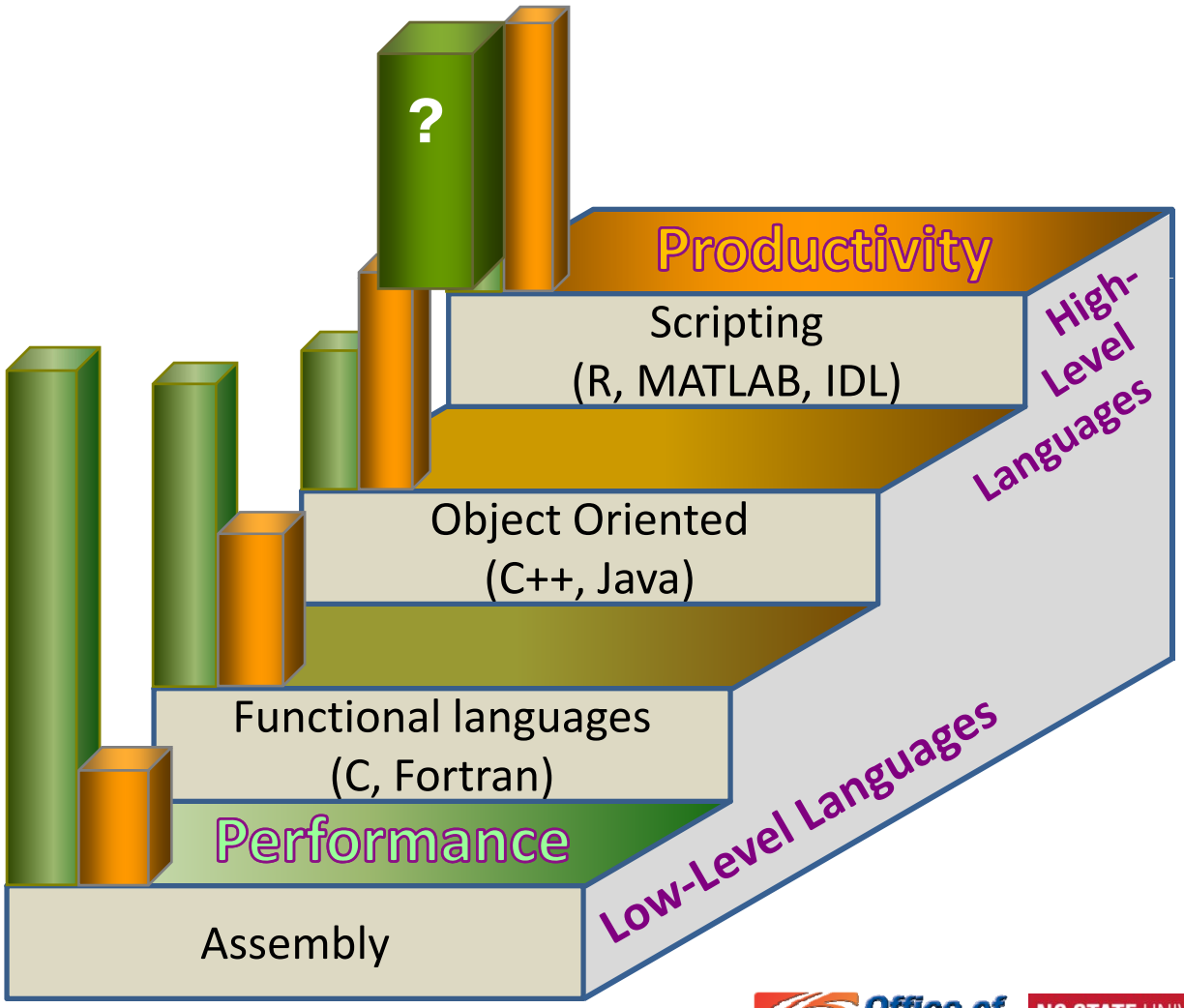
- Application Driver
- **Introduction to Parallel R**
- Analytics-Aware Parallel I/O
- In situ Analytics in Staging

The Programmer's Dilemma



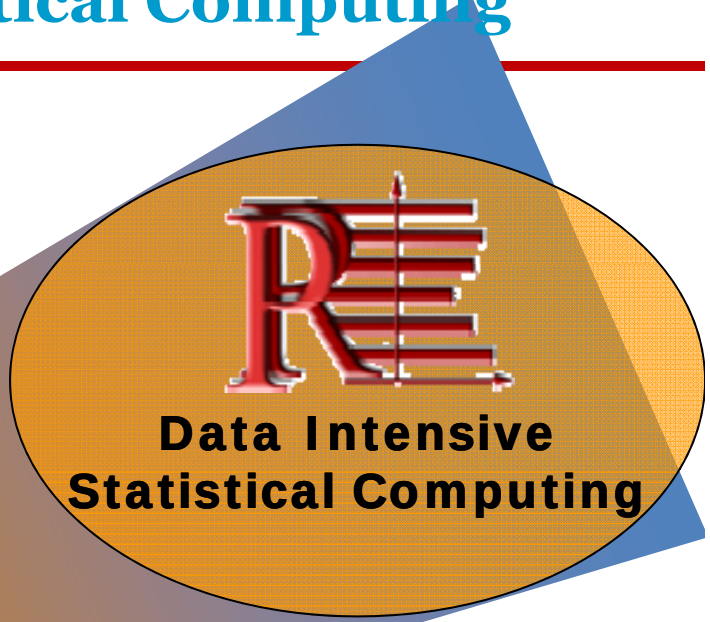
Towards High-Performance High-Level Languages

How do we get there? – Parallelization



One Hat Does NOT Fit All

Parallel R for Data Intensive Statistical Computing



•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

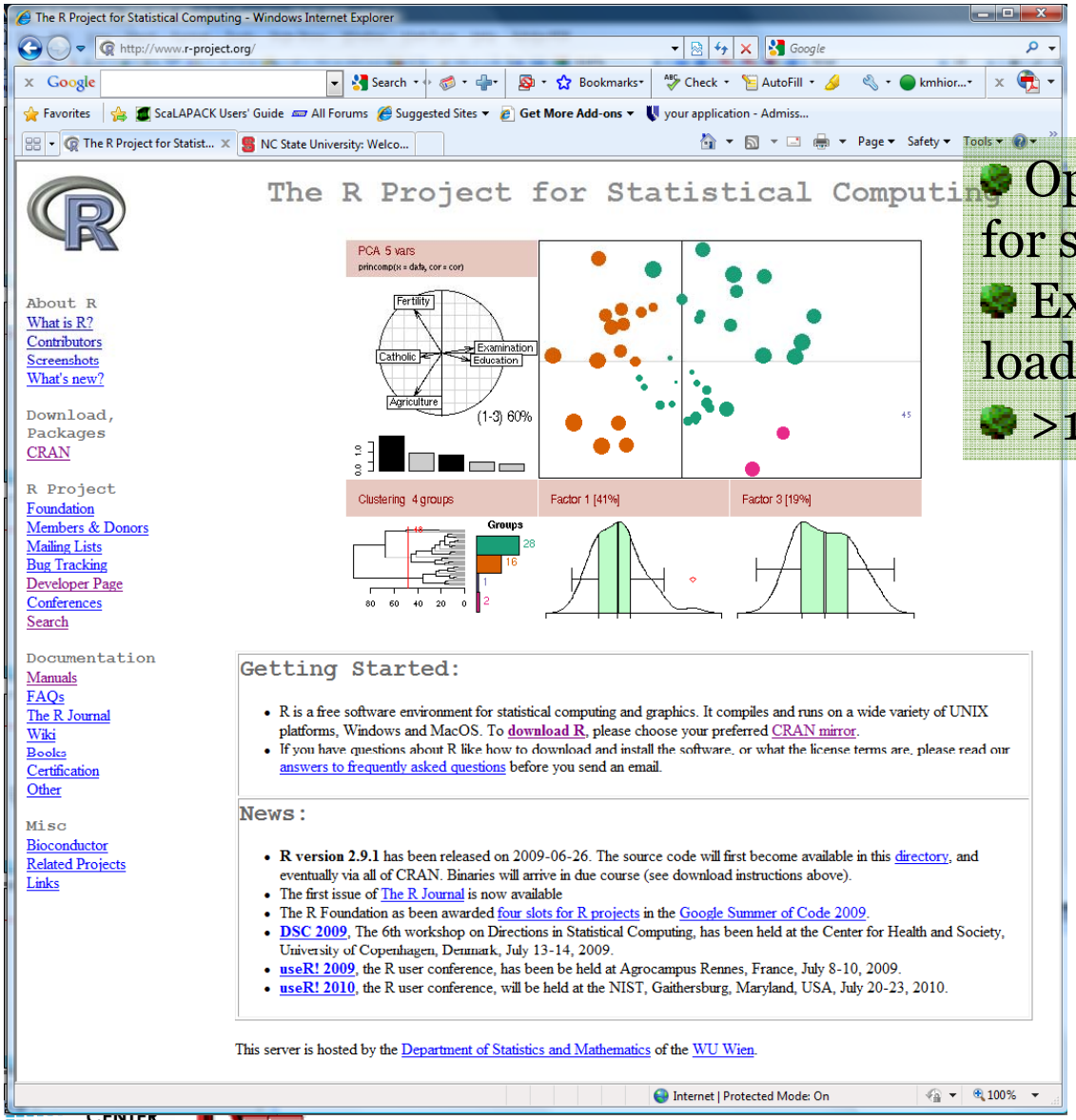
SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

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” )
```

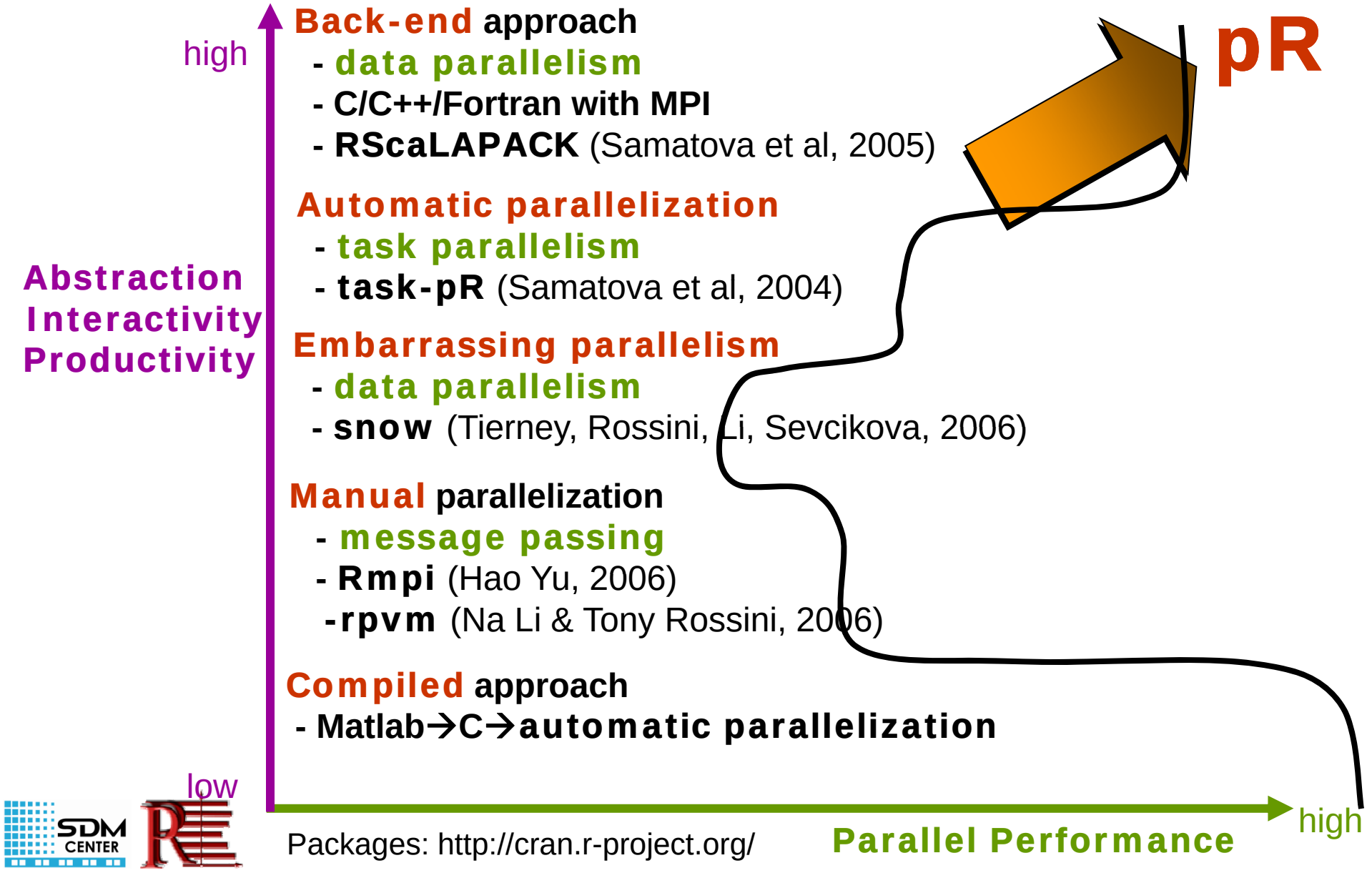
Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

Lessons Learned from R/Matlab Parallelization

Interactivity and High-Level: Curse & Blessing



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

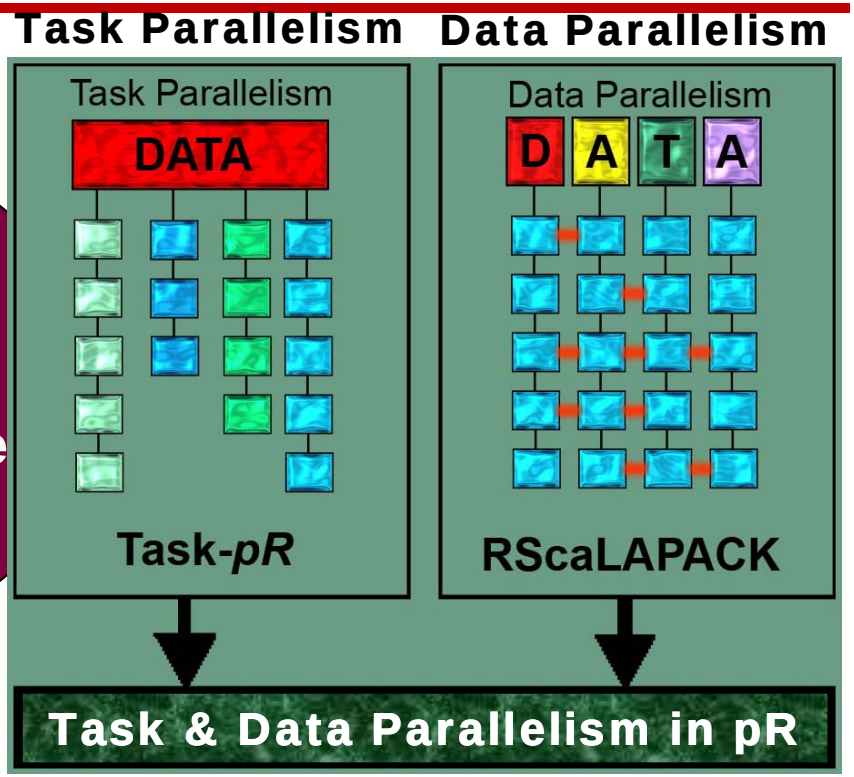
Task and Data Parallelism in pR

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

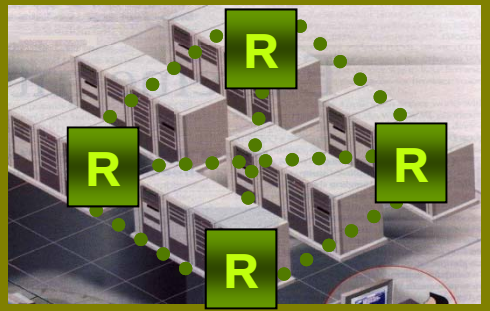
OAK RIDGE National Laboratory

p R Multi-tiered Architecture

Interactive R Client

Loosely Coupled

R Servers



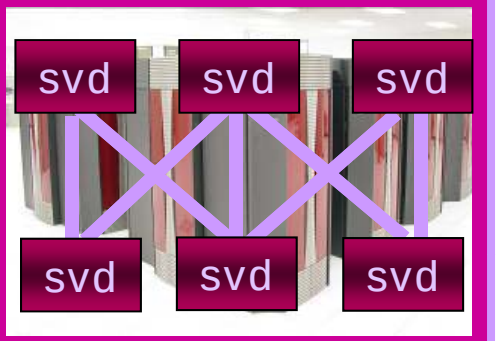
Task / Embarrassingly parallel jobs

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)

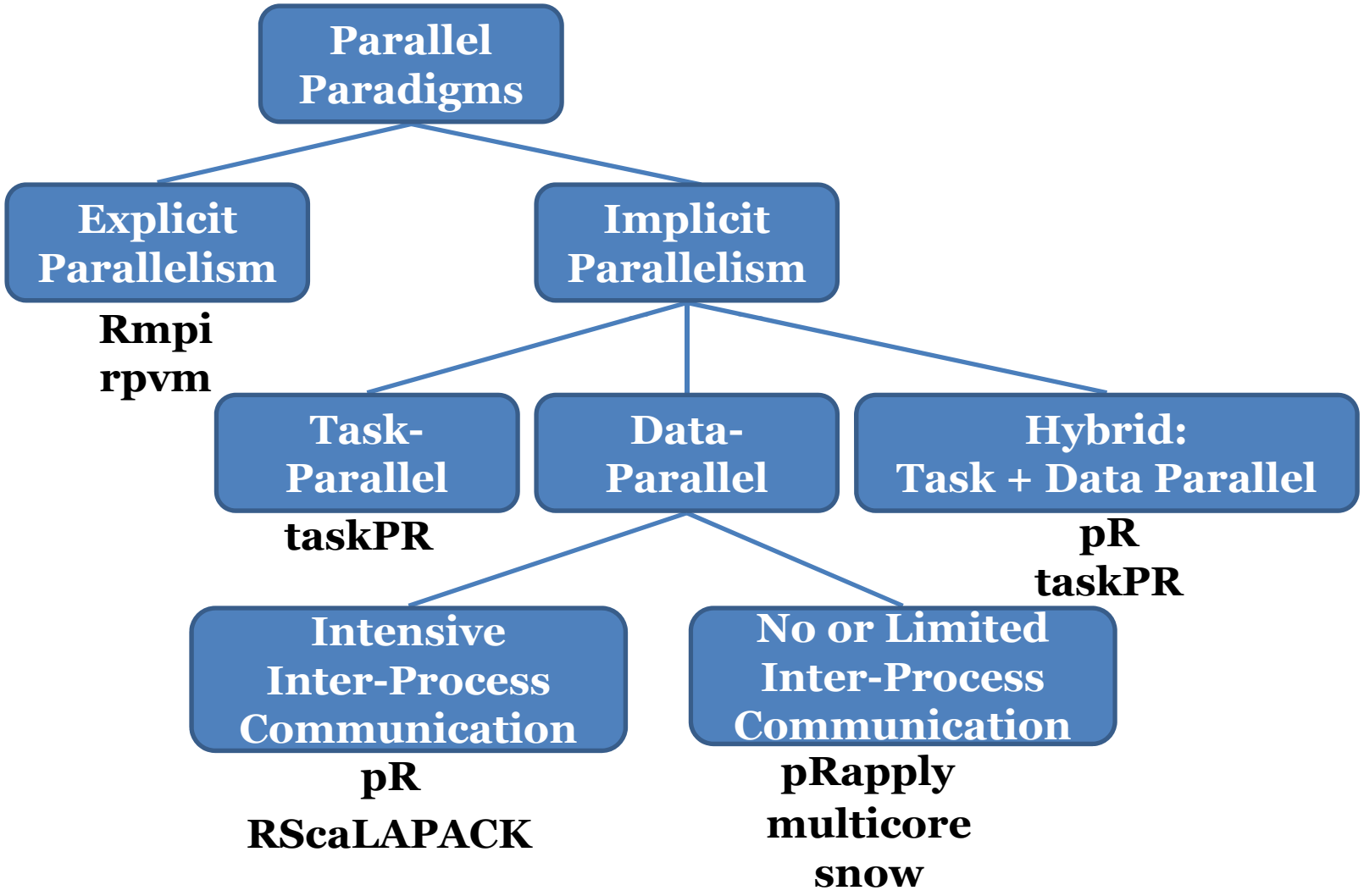
Memory & I/O Management

U.S. DEPARTMENT OF ENERGY

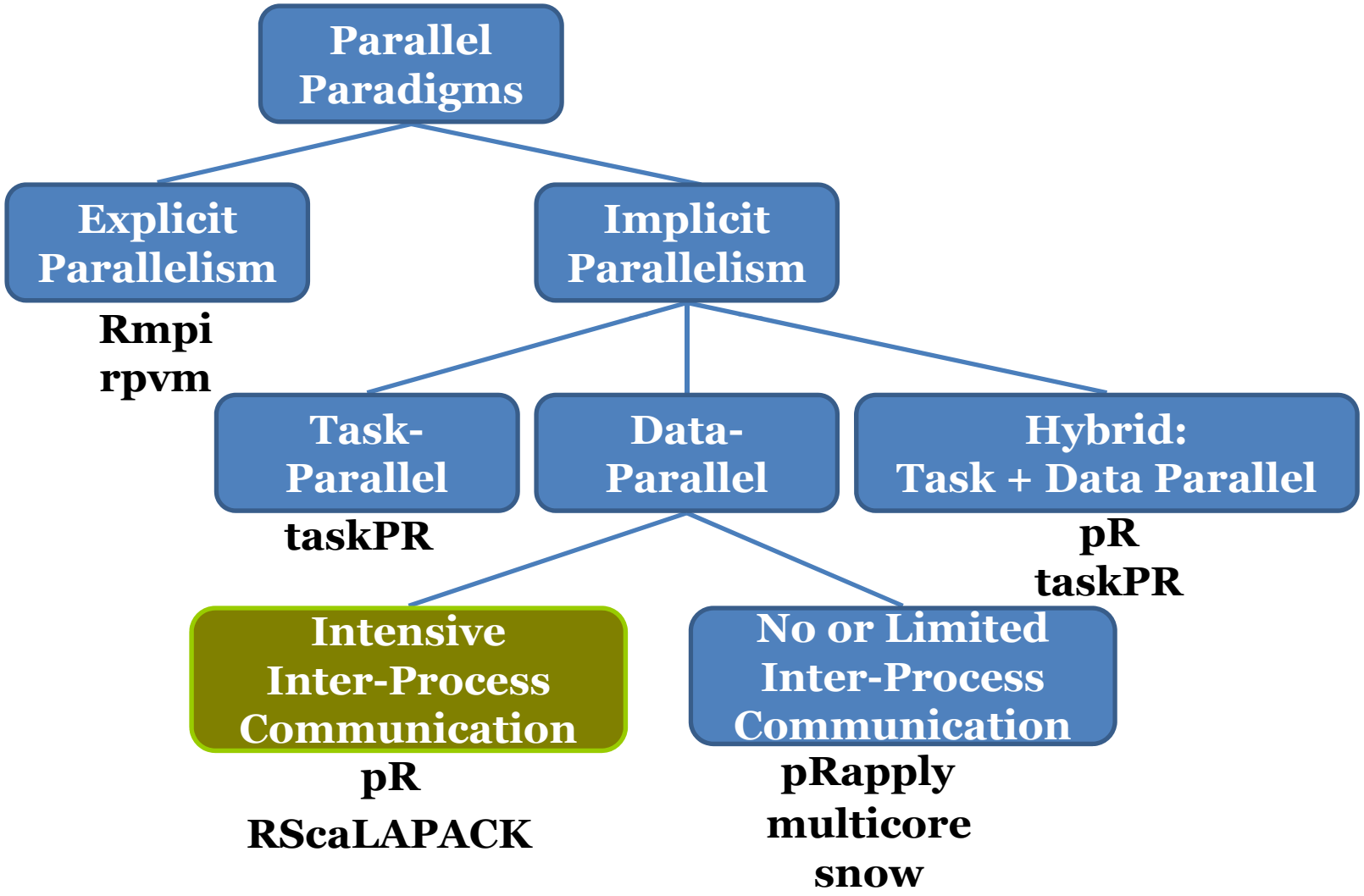
CENTER

OAK RIDGE National Laboratory

Parallel Paradigm Hierarchy

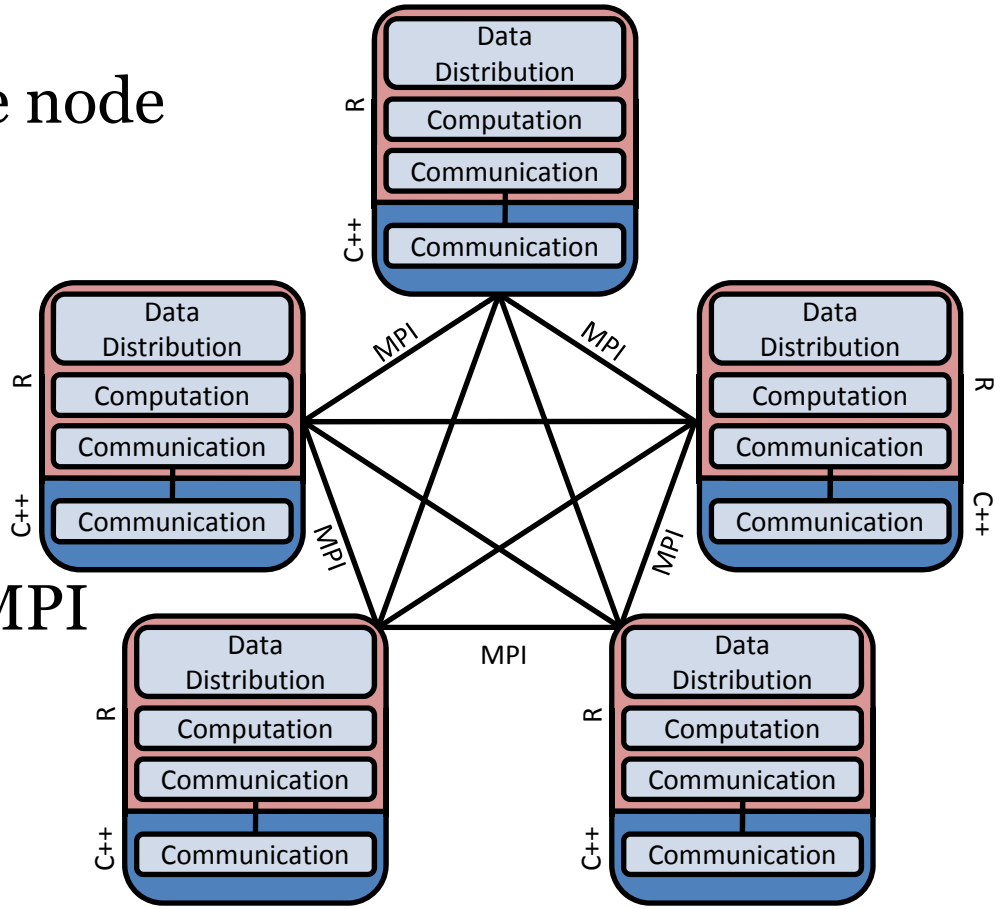


Parallel Paradigm Hierarchy



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 Example: Do it Yourself

R End-User

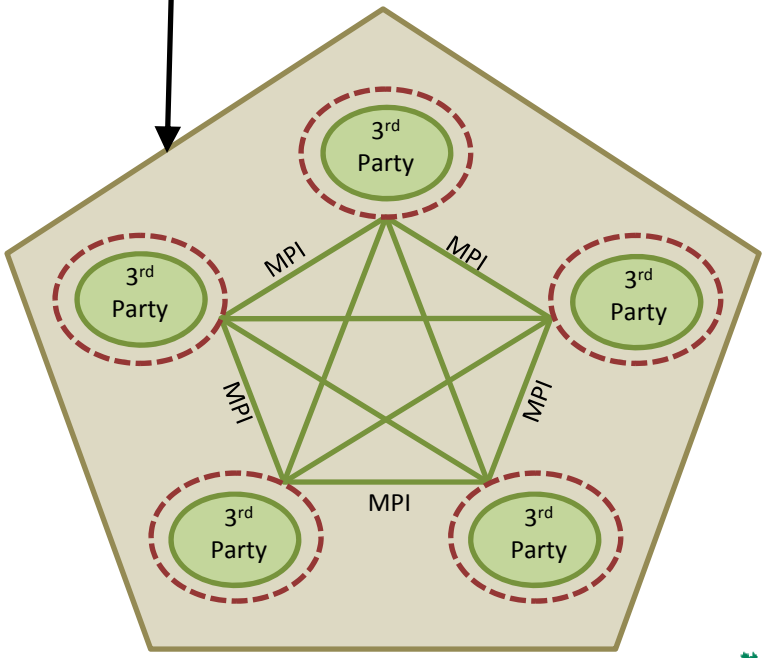
```
A = matrix(c(1:256),16,16)
B = matrix(c(1:256),16,16)
C = mm(A, B)
```

pR glue code

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

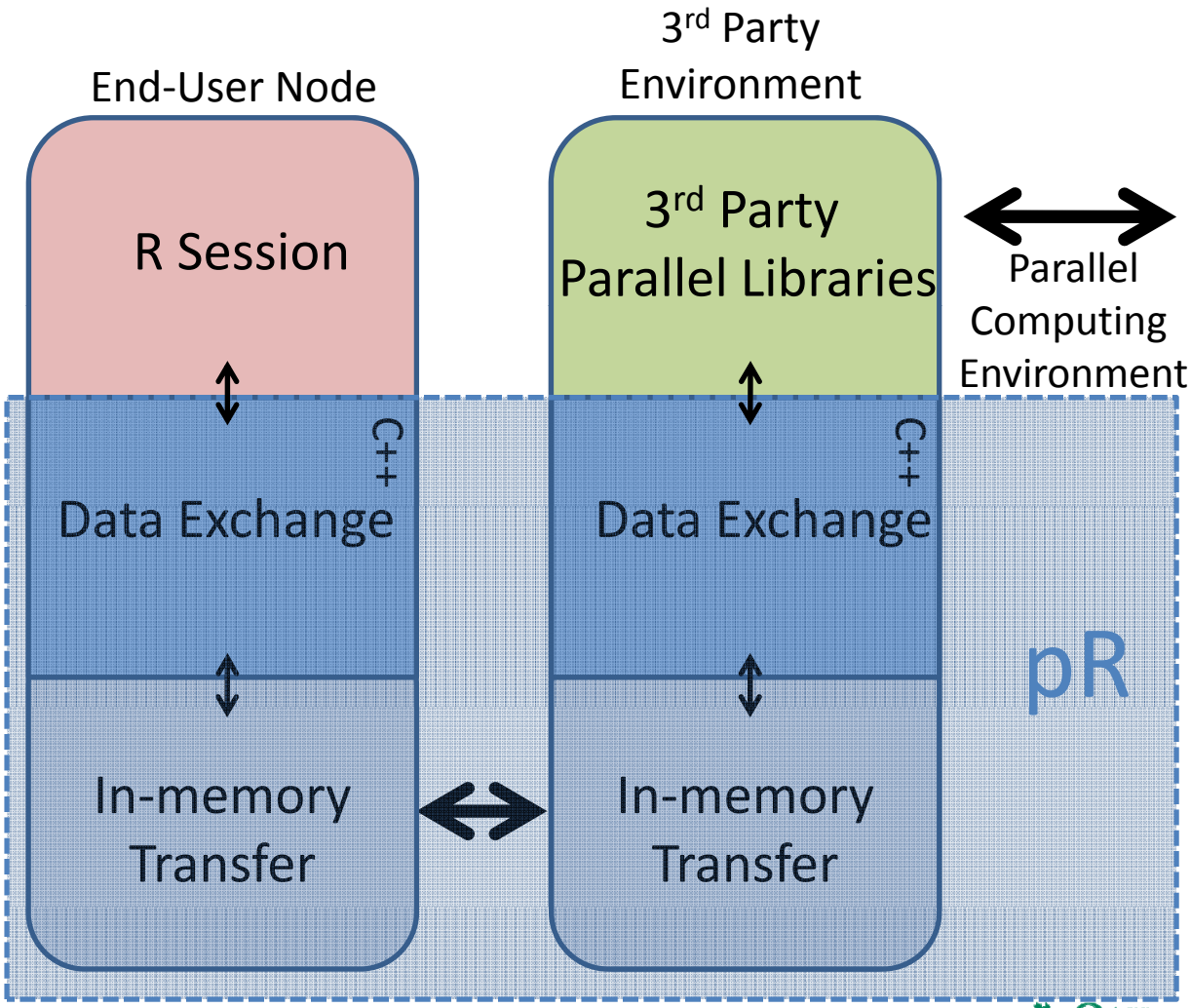
3rd Party Code: HPC MPI code

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

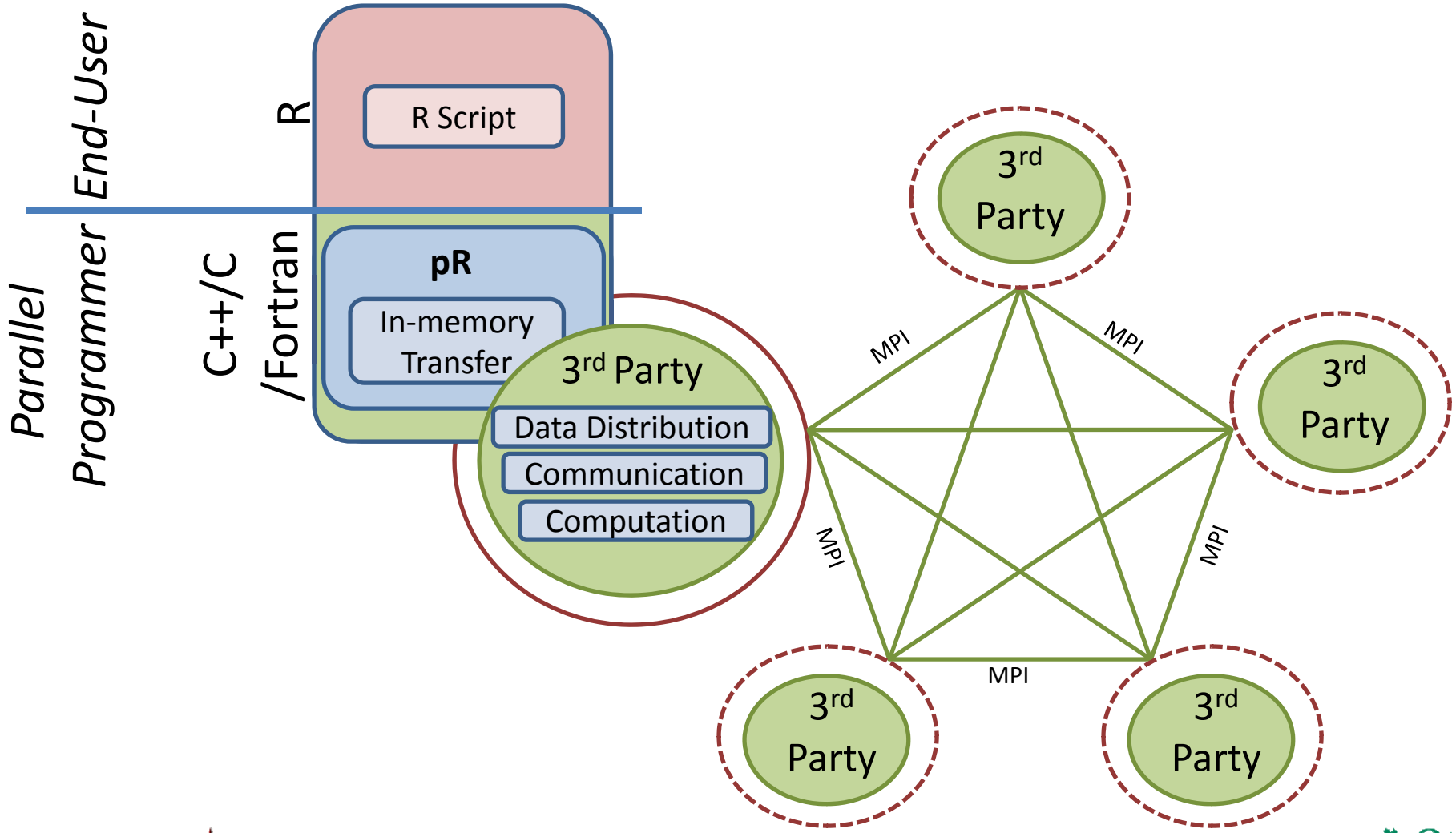


pR Parallel Plugin
Goals & Software Stack

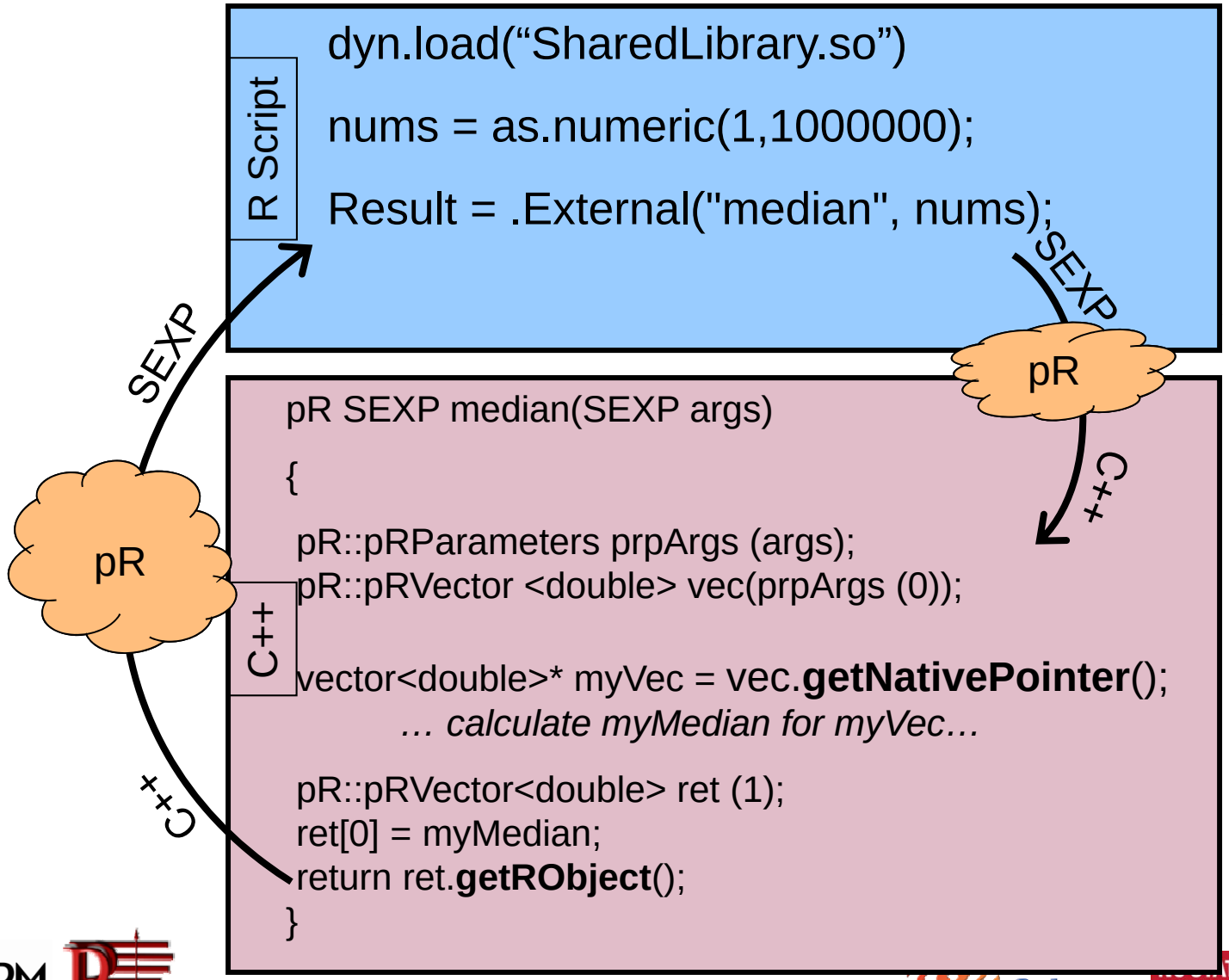
- Couple general parallel computation capabilities within R
- Shield end-users from the complexities of parallel computing
- Enable execution of compiled and scripting codes together



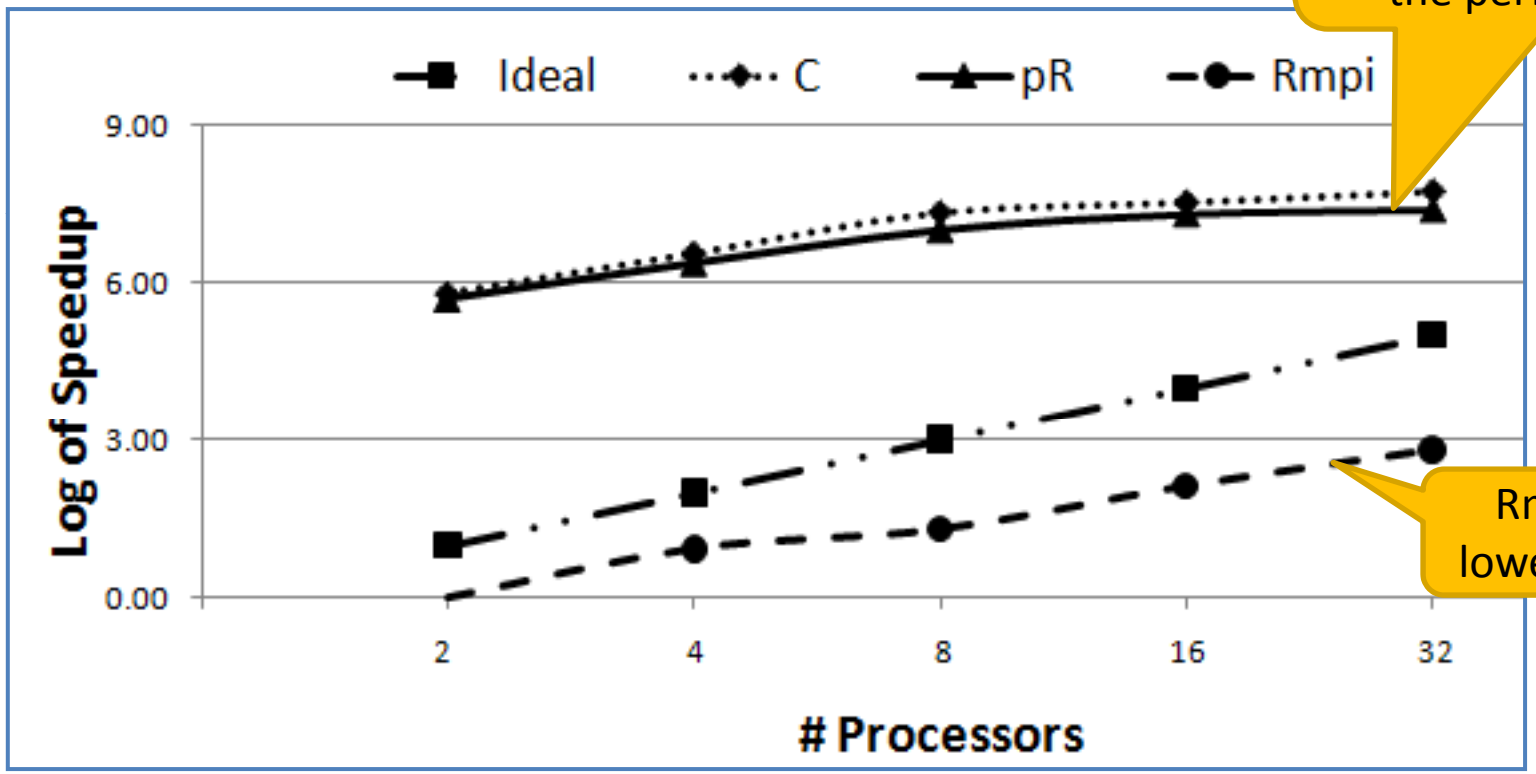
pR Overview


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



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



pR introduces minimal overhead & closely mirrors the performance of C

Rmpi speedup is lower than the ideal

Using a matrix-multiplication testcase on 4096 x 4096 matrices,

and comparing against a serial R implementation.

SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

Ex: RScaLAPACK Examples

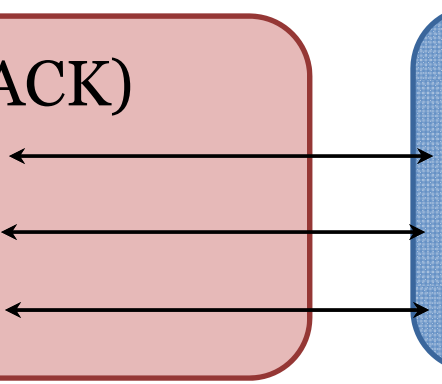
```
A = matrix(rnorm(256),16,16)
b = as.vector(rnorm(16))
```

Using RScaLAPACK:

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

Using R:

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



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

RScaLAPACK Functions

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

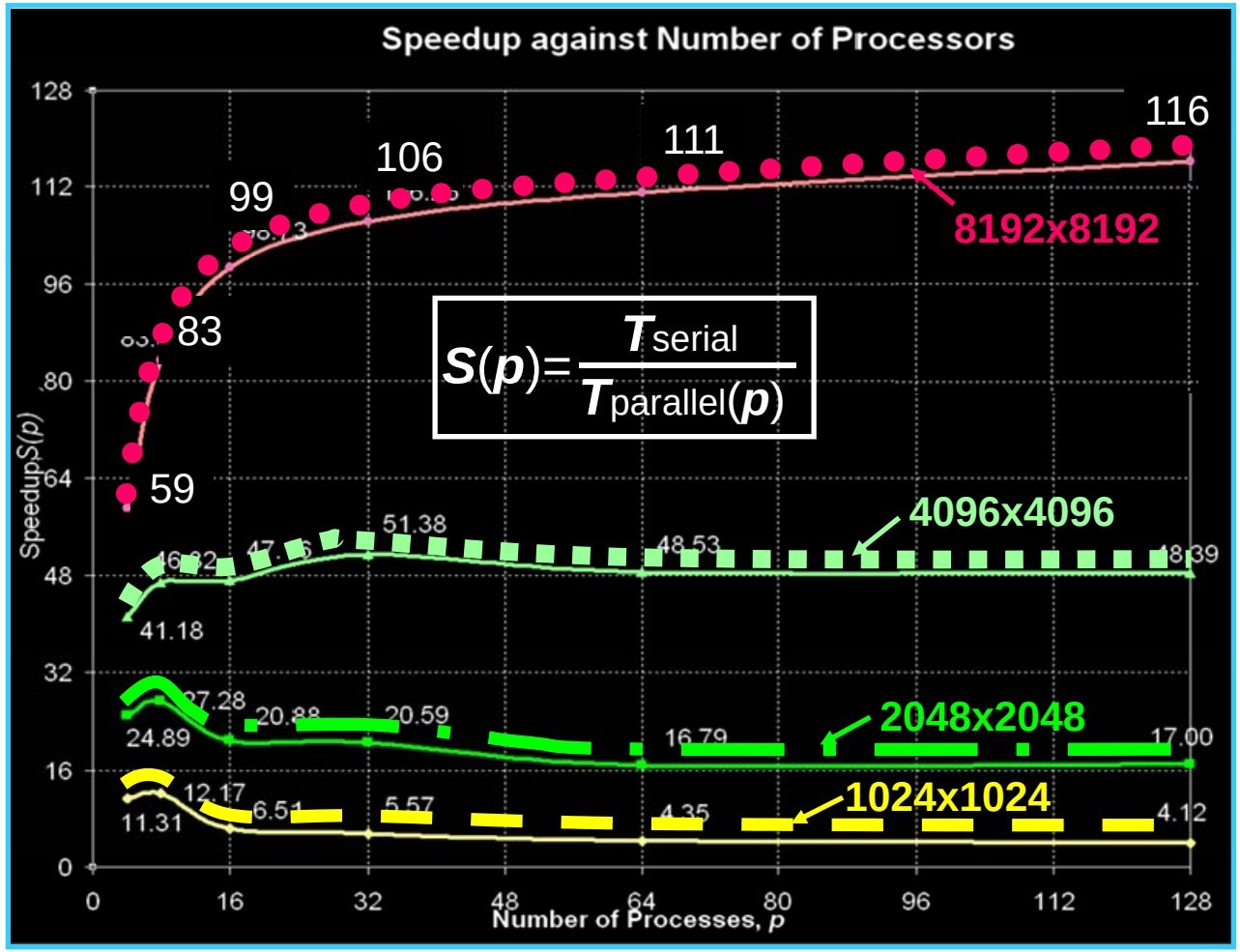
Currently Supported Functions

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

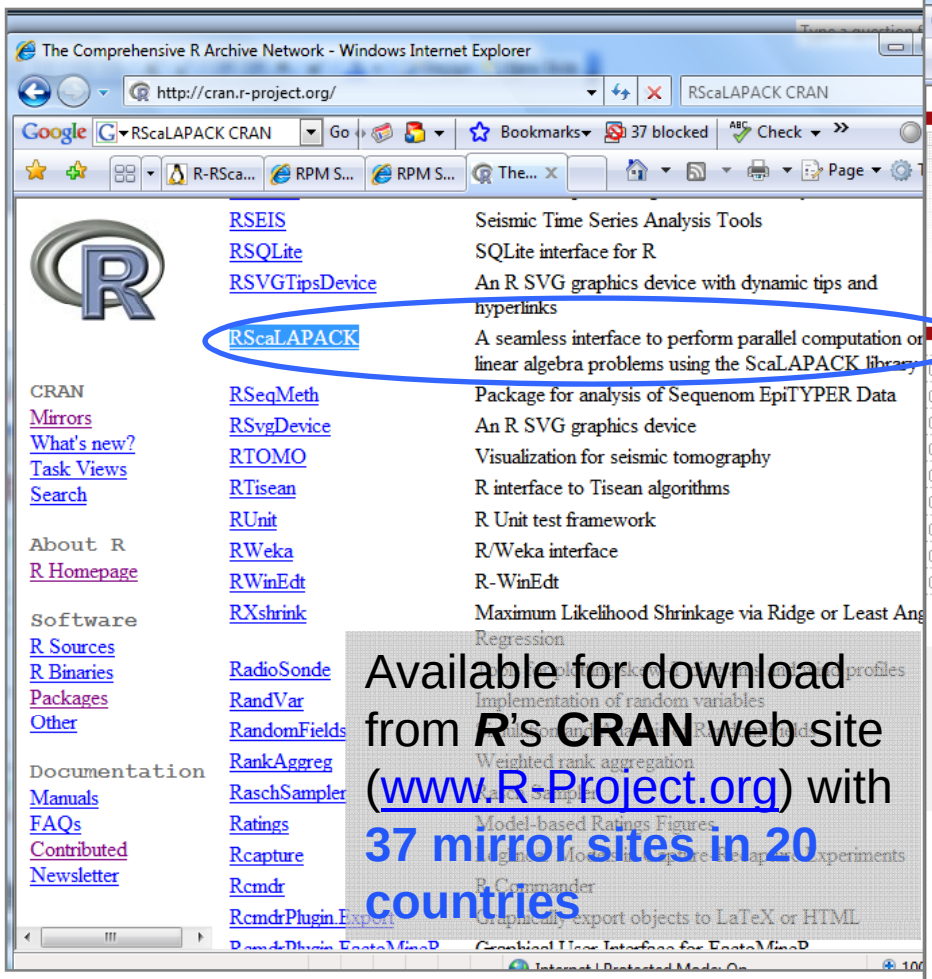
```
library (RScaLAPACK)
A = matrix(rnorm(128*128),128,128)
?sla.gridInit
```

Changing processor grid:

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

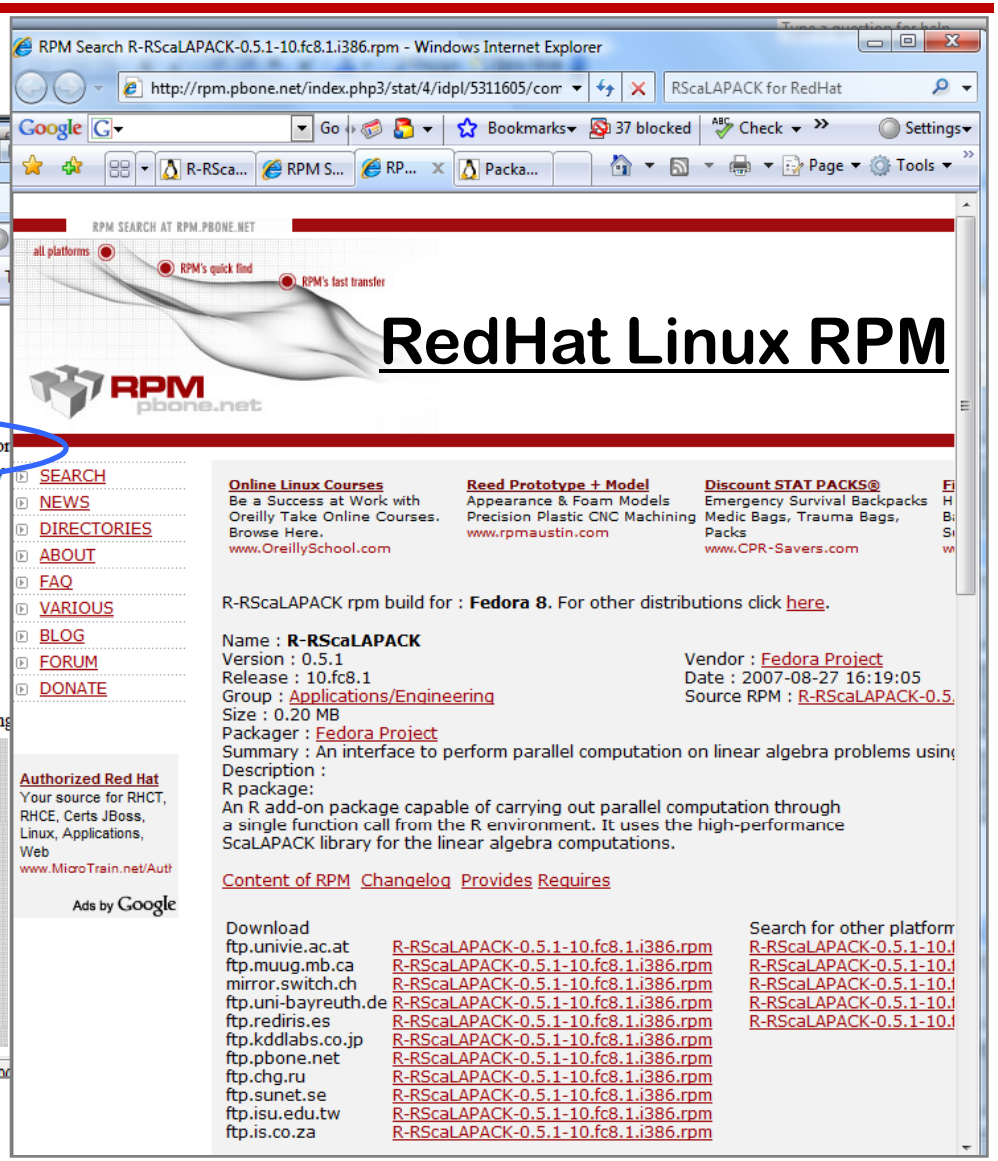
RedHat and CRAN Distribution

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



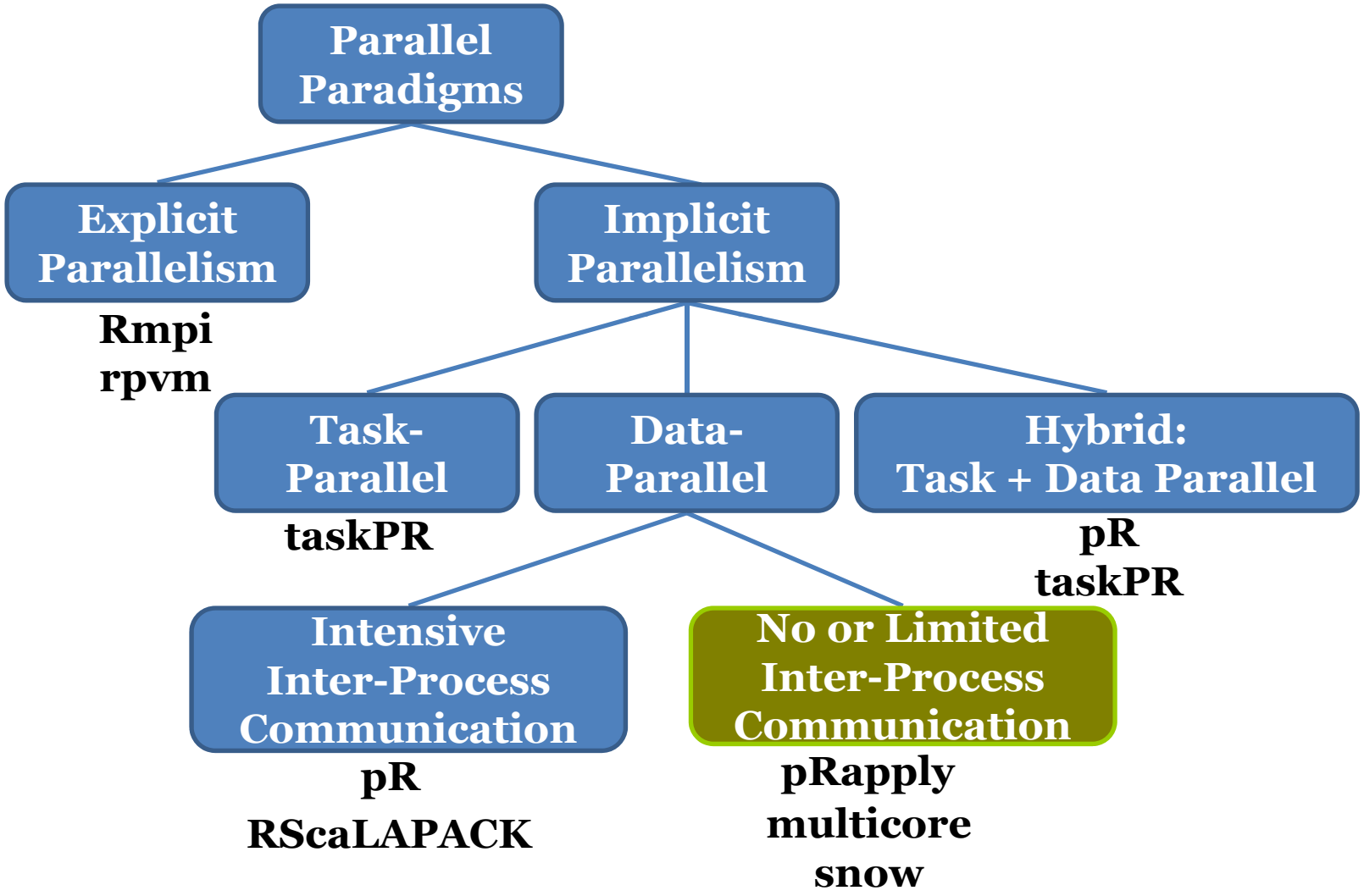
http://rpmfind.net/linux/RPM/RByName.html

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

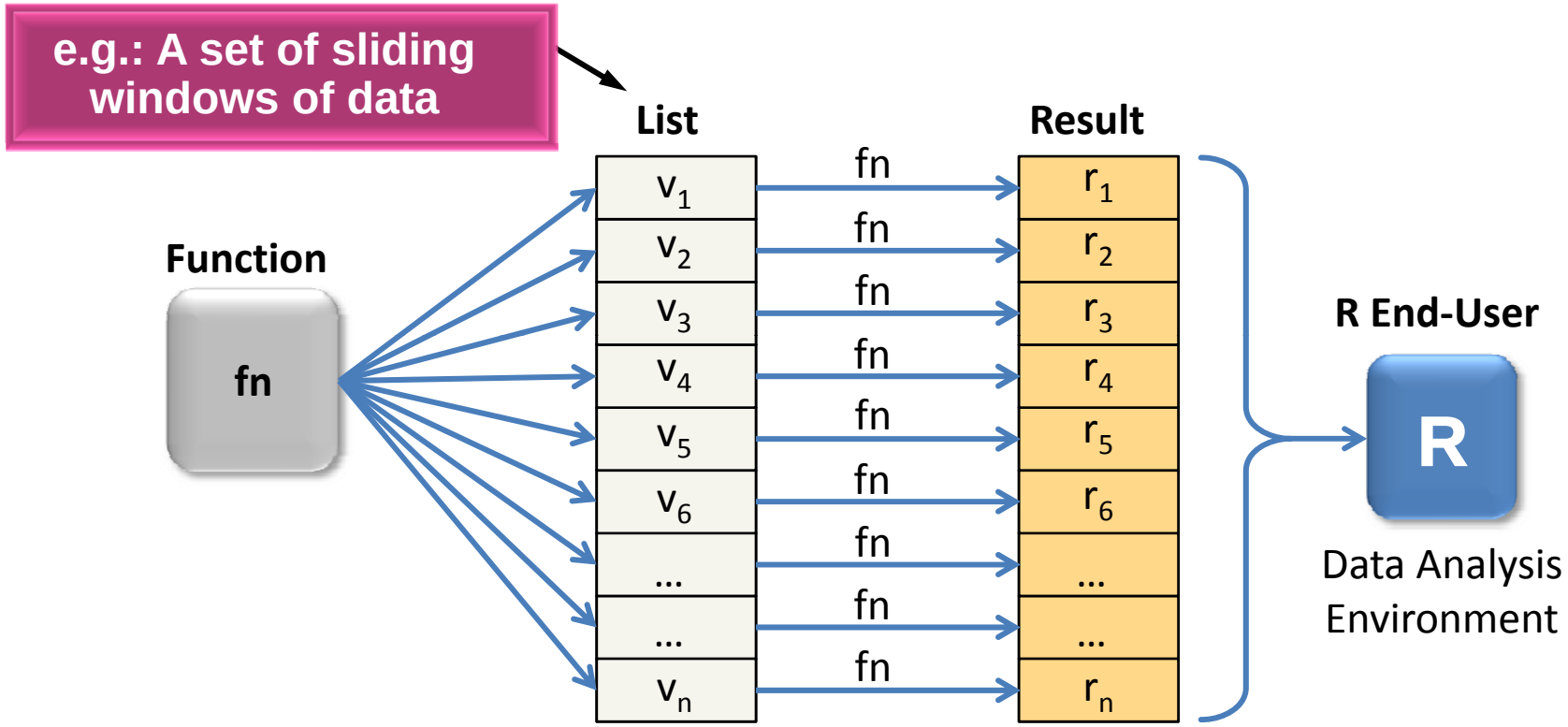
    export LD_LIBRARY_PATH=<path2deps>/lib:$LD_LIBRARY_PATH
- **Install RScaLAPACK**:
  - R CMD INSTALL *--configure-args="--with-f77*
    --with-mpi=<MPI install home directory>
    --with-blacs=<blacs build>/lib
    --with-blas=<blas build>/lib
    --with-lapack=<lapack build>/lib
    --with-scalapack=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

Parallel Paradigm Hierarchy



R's *lapply* Method

Natural Candidate for Parallel Computation



Example Functions:

- **Sliding Window-based Processing**
- **Smoothing in Space or Time**
- **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)

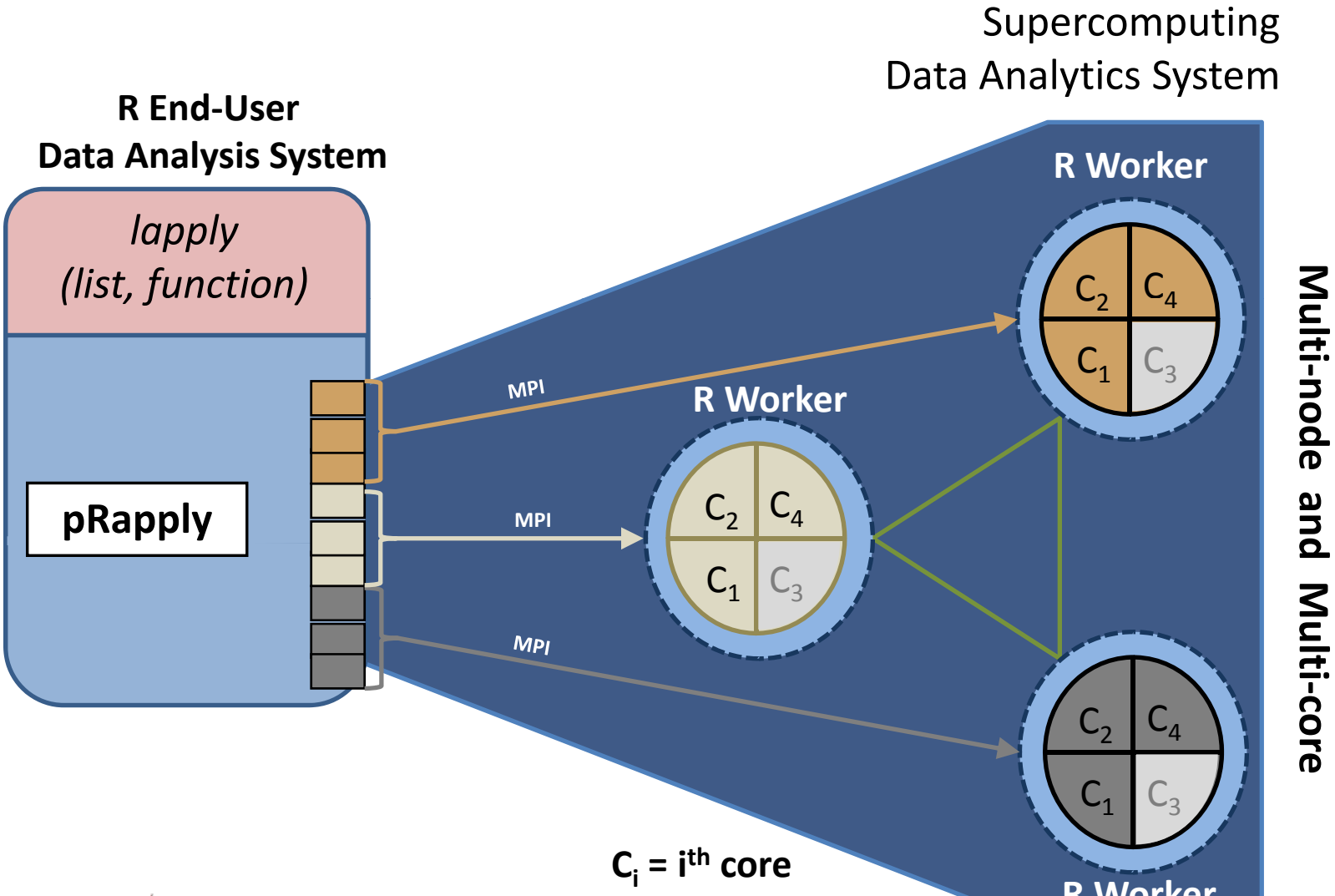
Using pRapply:

```
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)
```

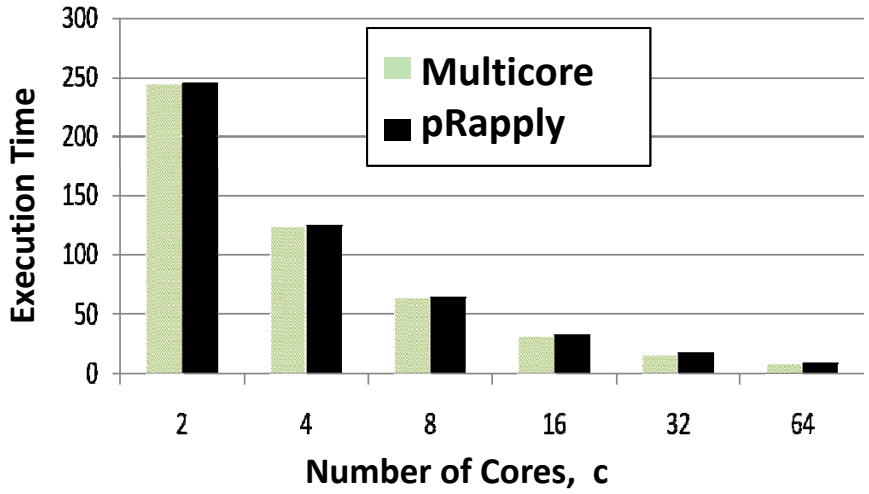
pRapply
Automatic Parallelization of lapply()



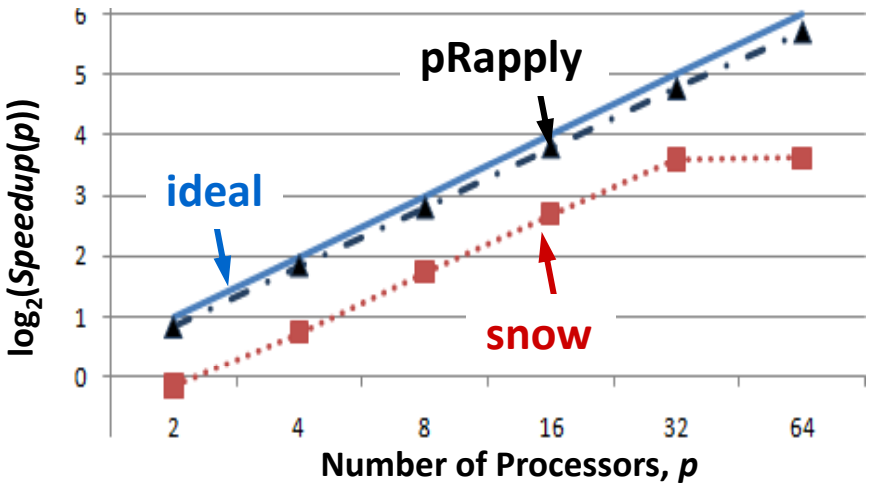
pRapply Performance

pRapply **is like** ***multicore*****, but supports hybrid multi-node, multi-core execution and scales linearly with the number of processors.**

Multi-Core Performance



Multi-Node Speedup



$$Speedup(p) = \frac{Time_{serial}}{Time_{parallel}(p)}$$

SDM CENTER

Office of Science U.S. DEPARTMENT OF ENERGY

NC STATE UNIVERSITY Department of Computer Science

OAK RIDGE National Laboratory

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 parallel C: Negligible Overhead Induced
- Enables automatic parallelization of data-parallel codes in hybrid multi-core and multi-node environments

Outline

- Application Driver
- Introduction to Parallel R
- Analytics-Aware Parallel I/O
- In situ Analytics in Staging

Analytics-Aware Parallel I/O
Embedding Data Analytics into Parallel I/O

- **Communication-free statistics for ADIOS writes:**
  - Rationale: power efficiency, ε-steal simulation time, etc.
  - E.g., average = (sum, count) pair per writer per time step
  - XML-defined variables and functions to compute
  - Supported functions: min, max, average, std, histograms, etc.
- **Time series analysis for ADIOS reads (e.g., bpls):**
  - E.g., average = Total_sum / Total_count per time step
  - **Local statistics**: per time step, like a movie, across all writers
  - **Global statistics**: across all time steps, across all writers
  - Within the user-specified **time window**: local and global statistics
  - Example supported functions:
    - Pearson correlation or covariance of two time series
    - For each variable: min, max, avg, std, histogram
  - Creates histogram files for viewing via gnuplot (bpls –p)

Write Performance: Histogram Calculation

Histogram Time (sec)			
	Default Stats	10 break pts	20 break pts
10MB	0.03	0.07	0.08
100MB	0.25	0.71	0.79
1GB	2.39	6.68	7.62

- **Test configuration:**
  - Example: genarray
  - No of Writers: Does not matter (b/s communication free)
  - 3D Variables: 9
  - Data size: 10-fold the size at each experiment
  - Default statistics: min, max, avg, std (no histograms)
  - Histogram statistics:
    - Double the number of break points
    - Includes default stats

Read Performance: bpls -lap

- Test configuration:
  - File System: Lustre
  - Processors: 10K (number of writers)
  - Variables: 100
  - Bytes per variable: 160
  - Command: time bpls -lap g_625x16x1_5x2x2.bp
- Timing results:
  - real 0m3.064s
  - user 0m2.476s
  - sys 0m0.576s

Challenges for In situ Analytics in Staging

- Enable statistics-driven ADIOS writes
  - *Ad hoc* decision to write every k-th time step (k=10 for XGC)
  - What to write and when to write; how to represent the discarded data?
  - Rationale: to reduce data stored yet to increase information content
  - Implications on BP file format
- Build the analytical kernels for *in situ* analytics
  - Spatial data analytics
  - Time-window based data analytics
  - Global-context data analytics (approx. streamline vs. exact out-of-core)
- Build cost-efficient analytical pipelines
  - Different objective/cost functions and constraints
  - Hybrid: multi-core, multi-node, communication-free, in-node vs. in-staging, on GPUs, in-writes vs. in-reads, index-based analysis
- Couple analysis environment with Data Staging Middleware
  - Programming model
  - How to plug-in 3rd party analysis environments (e.g., parallel R) into staging

Selected 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.