# Rdsm: Distributed (Quasi-)Threads Programming in R

, Gaithersburg, MD
July 21, 2010

Norm Matloff
Department of Computer Science
University of California at Davis
Davis, CA 95616 USA
matloff@cs.ucdavis.edu

# Parallel R

- Many excellent packages are available.

# Parallel R

- Many excellent packages are available.
- But most use message-passing paradigm or variants, e.g. Rmpi, snow.

# Parallel R

- Many excellent packages are available.
- But most use message-passing paradigm or variants, e.g. Rmpi, snow.
- True shared-memory choices very limited.

# Parallel R

- Many excellent packages are available.
- But most use message-passing paradigm or variants, e.g. Rmpi, snow.
- True shared-memory choices very limited.
  - bigmemory
  - attached C (OpenMP, CUDA)

# ¡*Arriba* sharing!

# ¡*Arriba* sharing!

- Many in the parallel processing community consider shared-memory paradigm to be clearer, more concise, e.g. Chandra (2001), Hess (2002).

# ¡*Arriba* sharing!

- Many in the parallel processing community consider shared-memory paradigm to be clearer, more concise, e.g. Chandra (2001), Hess (2002).
- Conversion from sequential code easier than in message-passing case.

# Why Threads?

My definition here: Concurrent processes, communicating through shared memory.

# Why Threads?

My definition here: Concurrent processes, communicating through shared memory.

- Enable parallel computation.

# Why Threads?

My definition here: Concurrent processes, communicating through shared memory.

- Enable parallel computation.
  - Standard approach for speedup on shared-memory machines.

# Why Threads?

My definition here: Concurrent processes, communicating through shared memory.

- Enable parallel computation.
  - Standard approach for speedup on shared-memory machines.
- Enable parallel **I/O**!

# Why Threads?

My definition here: Concurrent processes, communicating through shared memory.

- Enable parallel computation.
  - Standard approach for speedup on shared-memory machines.
- Enable parallel **I/O**!
  - Perhaps less well-known, more commonly used.

# Why Threads?

My definition here: Concurrent processes, communicating through shared memory.

- Enable parallel computation.
  - Standard approach for speedup on shared-memory machines.
- Enable parallel **I/O**!
  - Perhaps less well-known, more commonly used.
  - E.g. Web servers.

# Rdsm: History and Motivation

# Rdsm: History and Motivation

- Goals:

# Rdsm: History and Motivation

- Goals:
  - Shared-memory vehicle for R, providing threads-like environment.

# Rdsm: History and Motivation

- Goals:
  - Shared-memory vehicle for R, providing threads-like environment.
  - Distributed computing capability, e.g. for collaborative tools.

# Rdsm: History and Motivation

- Goals:
  - Shared-memory vehicle for R, providing threads-like environment.
  - Distributed computing capability, e.g. for collaborative tools.
- Easy to build on my previous product, PerlDSM (Matloff, 2002).

# What Is Rdsm?

# What Is Rdsm?

- Provides R programmers with a threads-like programming environment:

# What Is Rdsm?

- Provides R programmers with a threads-like programming environment:
  - Multiple R processes.

# What Is Rdsm?

- Provides R programmers with a threads-like programming environment:
  - Multiple R processes.
  - Read/write shared variables, accessed through ordinary R syntax.

# What Is Rdsm?

- Provides R programmers with a threads-like programming environment:
  - Multiple R processes.
  - Read/write shared variables, accessed through ordinary R syntax.
  - Locks, barriers, wait/signal, etc.

# What Is Rdsm?

- Provides R programmers with a threads-like programming environment:
  - Multiple R processes.
  - Read/write shared variables, accessed through ordinary R syntax.
  - Locks, barriers, wait/signal, etc.
- Platforms:

Processes can be on the same mulicore machine or on distributed, geographically disperse machines.

# Applications of Rdsm

# Applications of Rdsm

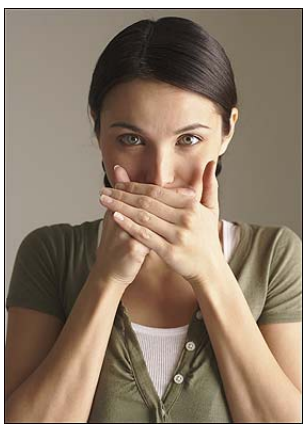
- Performance programming, in “embarrassingly parallel” (EP) settings.

# Applications of Rdsm

- Performance programming, in "embarrassingly parallel" (EP) settings.
- EP is possibly the limit for any parallel R, but there are lots of EP apps.

# Applications of Rdsm

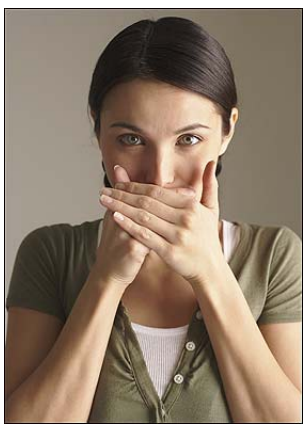
- Performance programming, in "embarrassingly parallel" (EP) settings.
- EP is possibly the limit for any parallel R, but there are lots of EP apps.



Nothing to be embarrassed about. :-)

# Applications of Rdsm

- Performance programming, in "embarrassingly parallel" (EP) settings.
- EP is possibly the limit for any parallel R, but there are lots of EP apps.

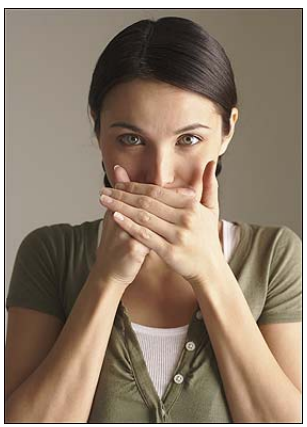


Nothing to be embarrassed about. :-)

- Parallel I/O applications, e.g. parallel collection of Web data and its concurrent statistical analysis.

# Applications of Rdsm

- Performance programming, in "embarrassingly parallel" (EP) settings.
- EP is possibly the limit for any parallel R, but there are lots of EP apps.



Nothing to be embarrassed about. :-)

- Parallel I/O applications, e.g. parallel collection of Web data and its concurrent statistical analysis.
- Collaborative tools.

# Applications of Rdsm

- Performance programming, in "embarrassingly parallel" (EP) settings.
- EP is possibly the limit for any parallel R, but there are lots of EP apps.

Nothing to be embarrassed about. :-)

- Parallel I/O applications, e.g. parallel collection of Web data and its concurrent statistical analysis.
- Collaborative tools.
- Even games!

# What Does Rdsm Code Look Like?

# What Does Rdsm Code Look Like?

Answer: Except for initialization, it looks just like—and IS—ordinary R code.

# What Does Rdsm Code Look Like?

Answer: Except for initialization, it looks just like—and IS—ordinary R code.
For example, to replace the 5th column of a shared matrix **m** by a vector of all 1s:

```
m[,5] <- 1  # use recycling
```

# What Does Rdsm Code Look Like?

Answer: Except for initialization, it looks just like—and IS—ordinary R code.
For example, to replace the 5th column of a shared matrix **m** by a vector of all 1s:

```
m[,5] <- 1  # use recycling
```

This is ordinary, garden-variety R code.

# What Does Rdsm Code Look Like?

Answer: Except for initialization, it looks just like—and IS—ordinary R code.
For example, to replace the 5th column of a shared matrix **m** by a vector of all 1s:

```
m[,5] <- 1  # use recycling
```

This is ordinary, garden-variety R code.
And it IS shared: If process 3 executes the above and then process 8 does

```
x <- m[2,5]
```

then **x** will be 1 at process 8.

# What Does Rdsm Code Look Like? (cont'd.)

The only difference is in creating the variable:

# What Does Rdsm Code Look Like? (cont'd.)

The only difference is in creating the variable:

```
# create shared 6x6 matrix
newdsm("m","dsmm","double",size=c(6,6))
```

Note the special **"dsmm"** class for shared matrices.

# What Does Rdsm Code Look Like? (cont'd.)

The only difference is in creating the variable:

```
# create shared 6x6 matrix
newdsm("m","dsmm","double",size=c(6,6))
```

Note the special **"dsmm"** class for shared matrices. (Also have classes for shared vectors and lists.)

# What Does Rdsm Code Look Like? (cont'd.)

The only difference is in creating the variable:

```
# create shared 6x6 matrix
newdsm("m","dsmm","double",size=c(6,6))
```

Note the special **"dsmm"** class for shared matrices. (Also have classes for shared vectors and lists.)
Otherwise, it's ordinary R syntax, with threads.

# Embarrassingly Parallel Example: Find Best k in k-NN Regression

Rdsm provides the familiar threads shared-memory environment.

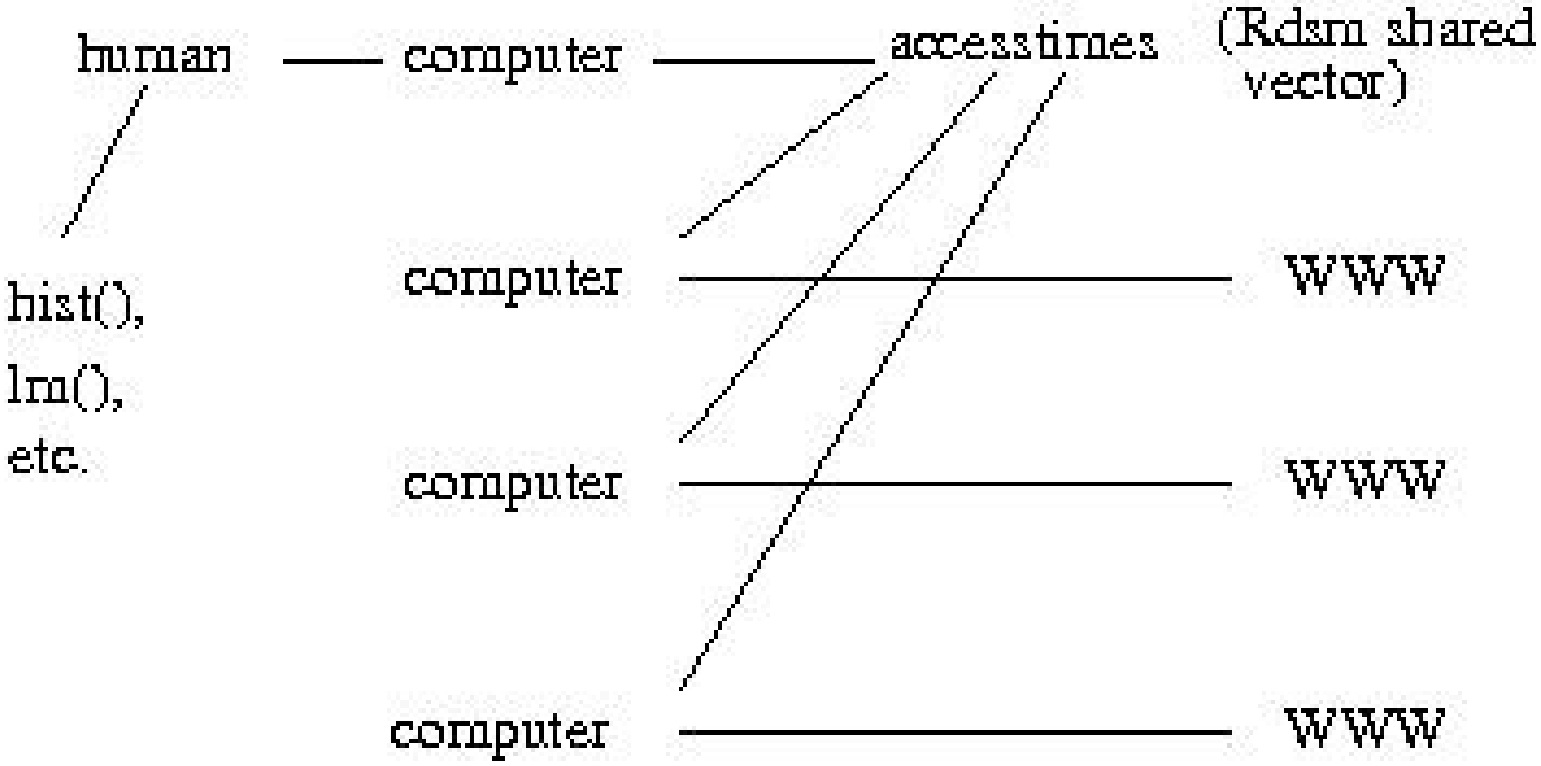
```
# have SHARED vars minmse, mink best found so far
# each process executes the following
rng <- findrange()  # range of k for this process
for (k in rng$mystart:rng$myend) {
    mse <- crossvalmse(x,y,k)
    lock("minlock")
    if (mse < minmse) {
        minmse <- mse
        mink <- k
    }
    unlock("minlock")
}
```

# Parallel I/O Example: Web Speed Monitor

Goal: Continually measure Web speed while concurrently allowing stat analysis on the collected data.

# Parallel I/O Example: Web Speed Monitor

Goal: Continually measure Web speed while concurrently allowing stat analysis on the collected data.
Rdsm solution:



# Web Speed Monitor (cont'd.)

What's in the picture:

# Web Speed Monitor (cont'd.)

What's in the picture:

- multiple Rdsm threads, 4 here

# Web Speed Monitor (cont'd.)

What's in the picture:

- multiple Rdsm threads, 4 here
- 3 of the threads gather data, by continually probing the Web

# Web Speed Monitor (cont'd.)

What's in the picture:

- multiple Rdsm threads, 4 here
- 3 of the threads gather data, by continually probing the Web
- those 3 threads write access times to the shared vector **accesstimes**

# Web Speed Monitor (cont'd.)

What's in the picture:

- multiple Rdsm threads, 4 here
- 3 of the threads gather data, by continually probing the Web
- those 3 threads write access times to the shared vector **accesstimes**
- in 4th thread, human gives R commands, reading the shared vector **accesstimes**

# Web Speed Monitor (cont'd.)

What's in the picture:

- multiple Rdsm threads, 4 here
- 3 of the threads gather data, by continually probing the Web
- those 3 threads write access times to the shared vector **accesstimes**
- in 4th thread, human gives R commands, reading the shared vector **accesstimes**
- the human applies R's myriad statistical operations to the data at his/her whim—concurrently with the data collection

# Example of Collaborative Structures: Online Auction

# Example of Collaborative Structures: Online Auction

- asynchronous—anyone can bid at any time, no turns

# Example of Collaborative Structures: Online Auction

- asynchronous—anyone can bid at any time, no turns
- shared variables:

# Example of Collaborative Structures: Online Auction

- asynchronous—anyone can bid at any time, no turns
- shared variables:
  - **latestbid**

# Example of Collaborative Structures: Online Auction

- asynchronous—anyone can bid at any time, no turns
- shared variables:
  - **latestbid**
  - **nbidders**, number who haven't dropped out of the bidding yet

# Example of Collaborative Structures: Online Auction

- asynchronous—anyone can bid at any time, no turns
- shared variables:
  - **latestbid**
  - **nbidders**, number who haven't dropped out of the bidding yet
- if n participants, then 2n Rdsm threads

# Example of Collaborative Structures: Online Auction

- asynchronous—anyone can bid at any time, no turns
- shared variables:
  - **latestbid**
  - **nbidders**, number who haven't dropped out of the bidding yet
- if n participants, then 2n Rdsm threads
- for a participant, one thread watches **latestbid**, the other submits bids

# Auction (cont'd.)

# Auction (cont'd.)

Built-in Rdsm functions used:

# Auction (cont'd.)

Built-in Rdsm functions used:

- **wait()**, **signal()**: Watcher threads call **wait()**, bidder threads call **signal()**.

# Auction (cont'd.)

Built-in Rdsm functions used:

- **wait()**, **signal()**: Watcher threads call **wait()**, bidder threads call **signal()**.
- **lock()**, **unlock()**: Usual need for lock, but with check for need to cancel bid.

# Auction (cont'd.)

Built-in Rdsm functions used:

- **wait()**, **signal()**: Watcher threads call **wait()**, bidder threads call **signal()**.
- **lock()**, **unlock()**: Usual need for lock, but with check for need to cancel bid.
- **fa()**: Fetch-and-add, to atomically decrement **nbidders** when someone drops out.

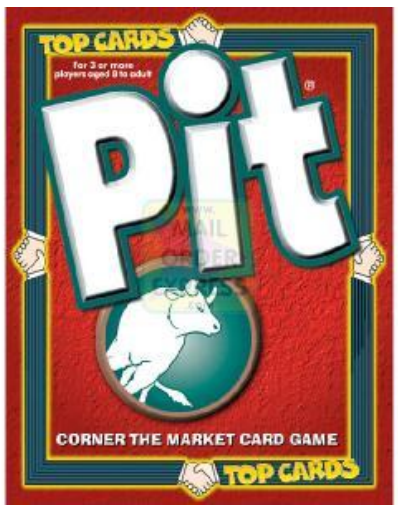
# R...As a GAME Platform????

# R...As a GAME Platform????

Well, just for fun...

# R...As a GAME Platform????

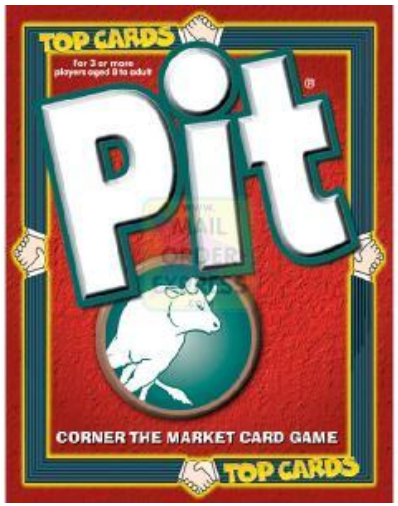
Well, just for fun...




- Project for my parallel programming students: Use Rdsm to implement the card game, Pit.

# R...As a GAME Platform????

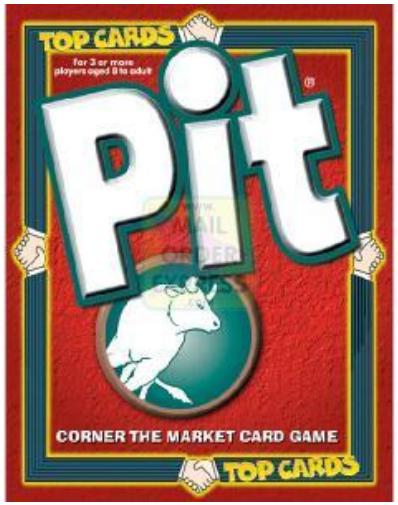
Well, just for fun...




- Project for my parallel programming students: Use Rdsm to implement the card game, Pit.
- Asynchronous—no turns! Like Auction.R.

# R...As a GAME Platform????

Well, just for fun...




- Project for my parallel programming students: Use Rdsm to implement the card game, Pit.
- Asynchronous—no turns! Like Auction.R.
- Transaction coding tricky; when is a trade "official"?

# How Rdsm Works

# How Rdsm Works

Same scheme as in PerIDSM (Matloff, 2002):

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.
- Physical storage of shared variables at server.

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.
- Physical storage of shared variables at server.
- Rdsm shared-variable classes:

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.
- Physical storage of shared variables at server.
- Rdsm shared-variable classes:
  - **dsmv**: shared vector
  - **dsmm**: shared matrix
  - **dsml**: shared list

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.
- Physical storage of shared variables at server.
- Rdsm shared-variable classes:
  - **dsmv**: shared vector
  - **dsmm**: shared matrix
  - **dsml**: shared list
- Redefine indexing functions, e.g. **"[.dsmv"**, **"[<-.dsmv"**.

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.
- Physical storage of shared variables at server.
- Rdsm shared-variable classes:
  - **dsmv**: shared vector
  - **dsmm**: shared matrix
  - **dsml**: shared list
- Redefine indexing functions, e.g. **"[.dsmv"**, **"[<-.dsmv"**.
- New indexing functions communicate with server.

# How Rdsm Works

Same scheme as in PerlDSM (Matloff, 2002):

- R processes run on clients.
- Physical storage of shared variables at server.
- Rdsm shared-variable classes:
  - **dsmv**: shared vector
  - **dsmm**: shared matrix
  - **dsml**: shared list
- Redefine indexing functions, e.g. **"[.dsmv"**, **"[<-.dsmv"**.
- New indexing functions communicate with server.
- But all is transparent to programmer.

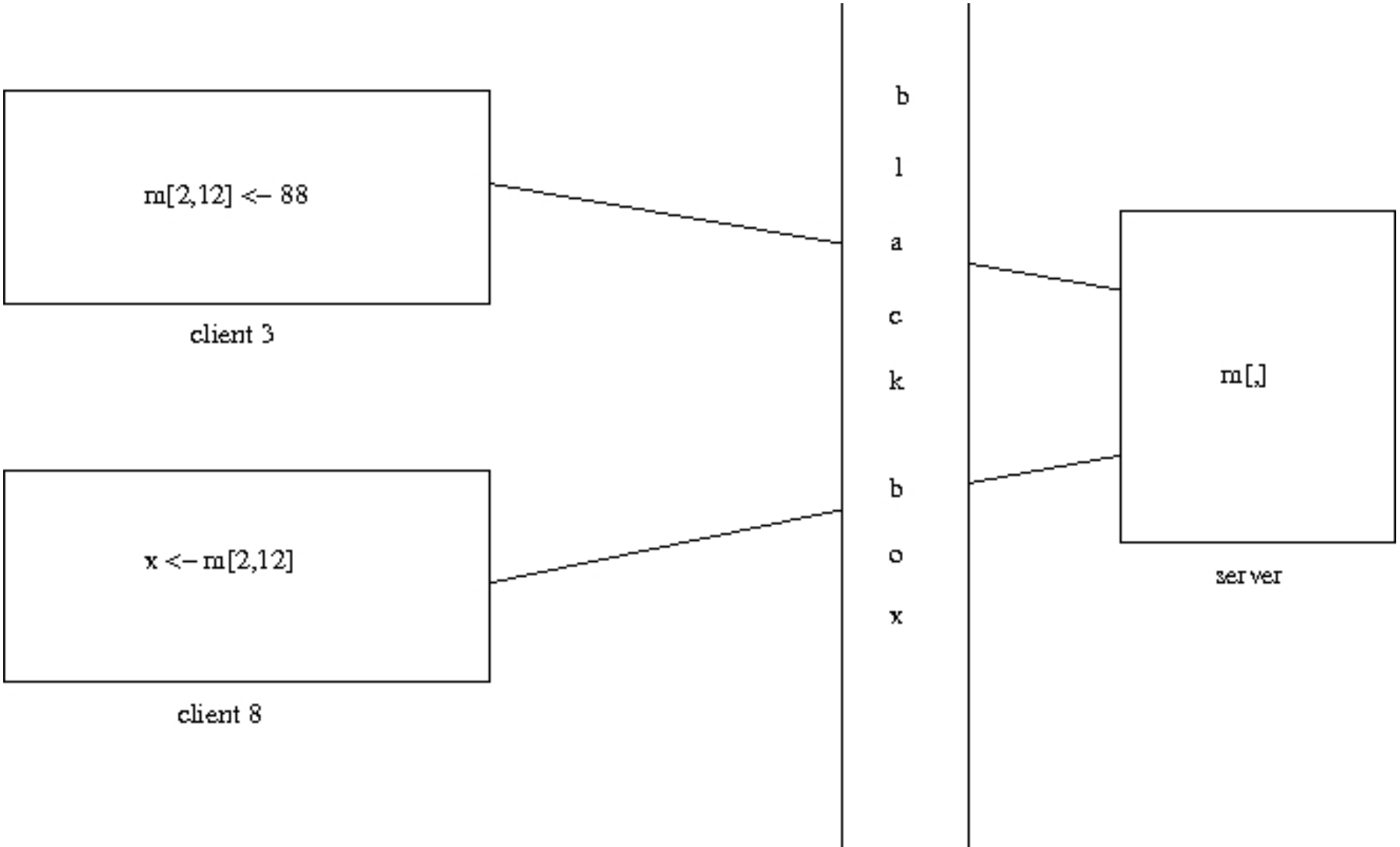
# Rdsm Internals, cont'd.

# Rdsm Internals, cont'd.

E.g. client 3 writes to **m[2,12]**, then client 8 reads it:

# Rdsm Internals, cont'd.

E.g. client 3 writes to **m[2,12]**, then client 8 reads it:



# Comparison to bigmemory

- Rdsm has functions for threads infrastructure[1]
- Rdsm is usable across fully independent machines[2]
- bigmemory may be faster on embarrassingly parallel apps

[1] I've written an incomplete set for bigmemory.

[2] but could try bigmemory with NFS files