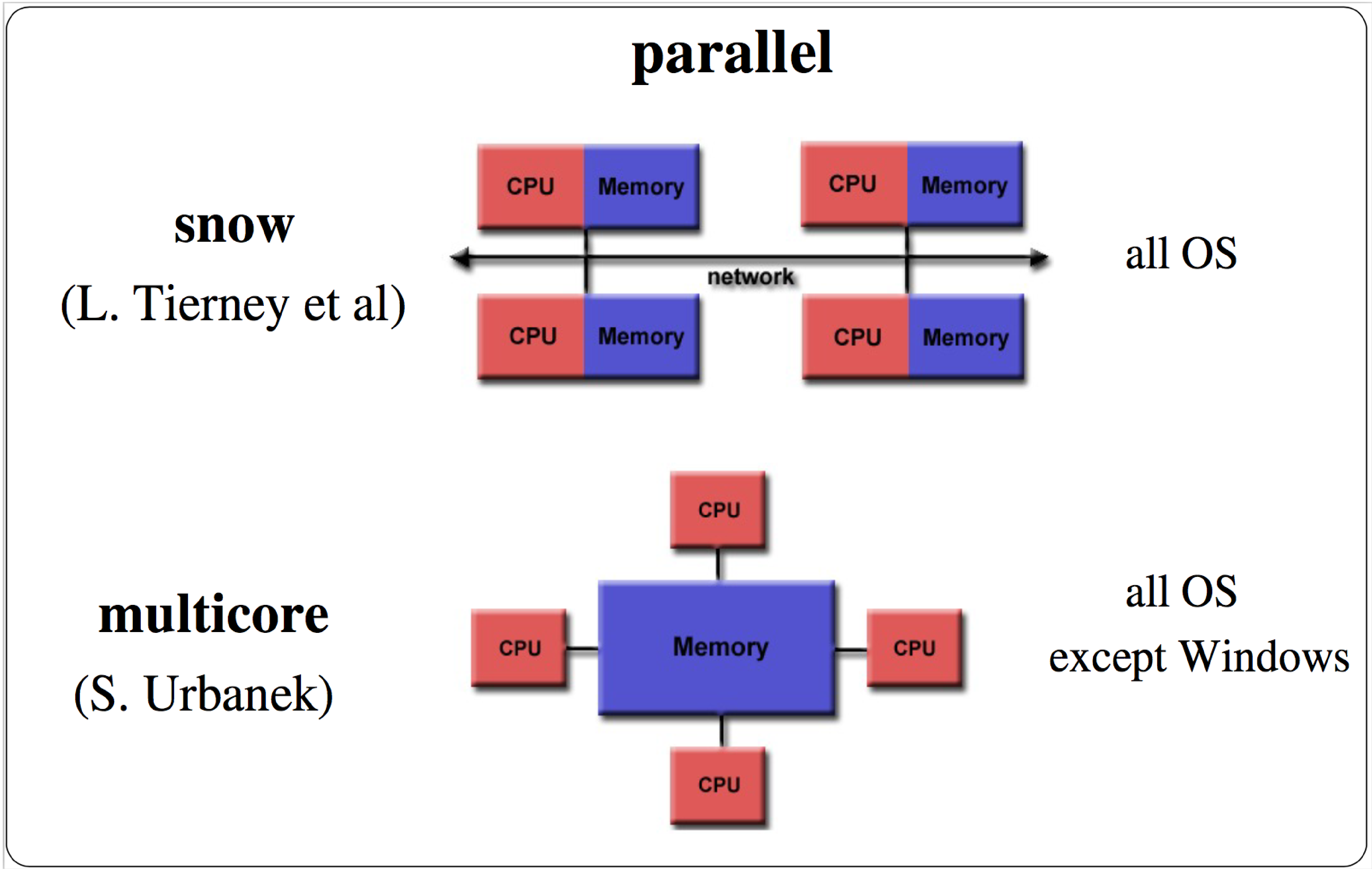
PARALLEL PROGRAMMING IN R

Cluster Basics

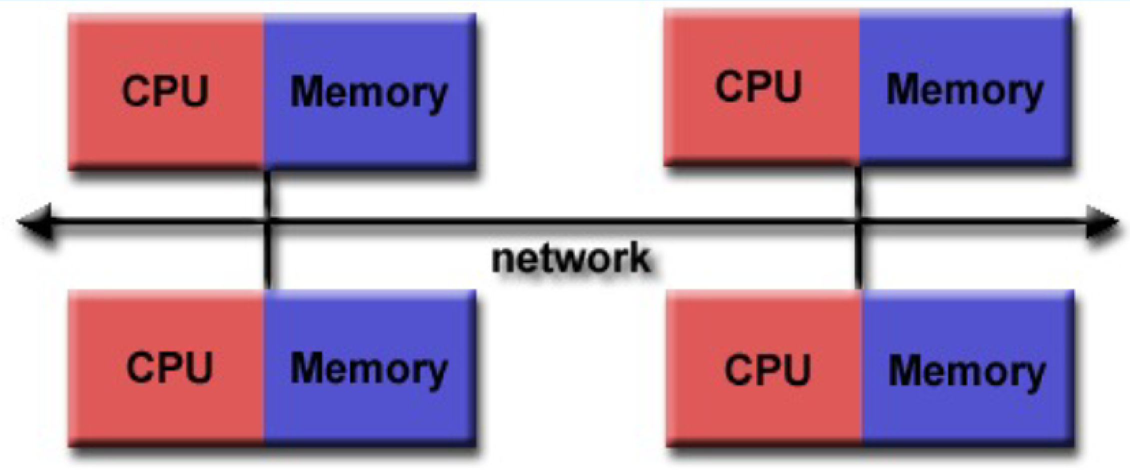
Hana Sevcikova
University of Washington


parallel
snow
(L. Tierney et al)
CPU
Memory
CPU
Memory
network
CPU
Memory
CPU
Memory
all OS
multicore
(S. Urbanek)
CPU
CPU
Memory
CPU
CPU
all OS
except Windows

parallel

snow

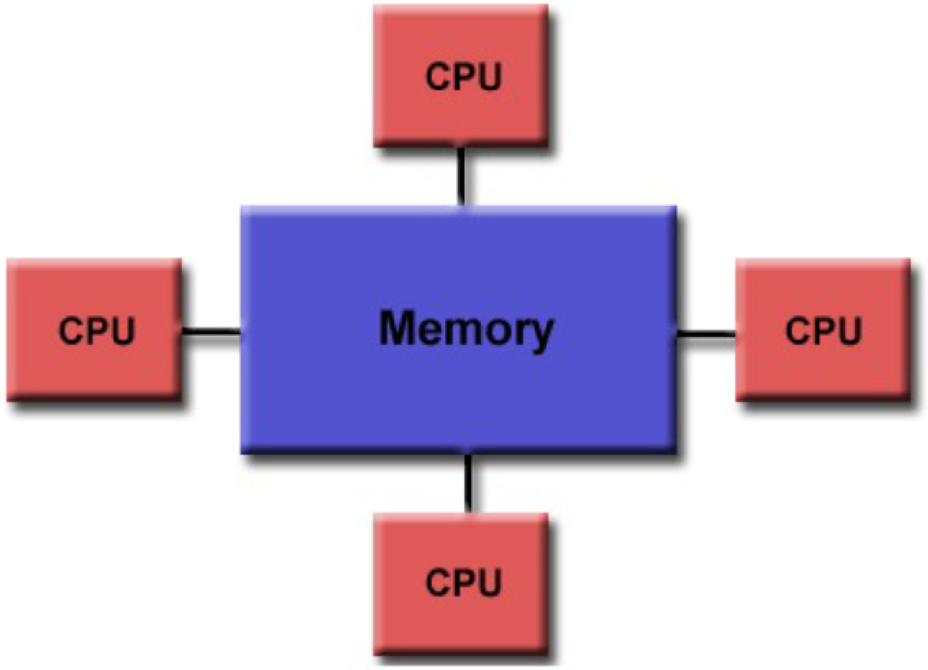
(L. Tierney et al)



all OS

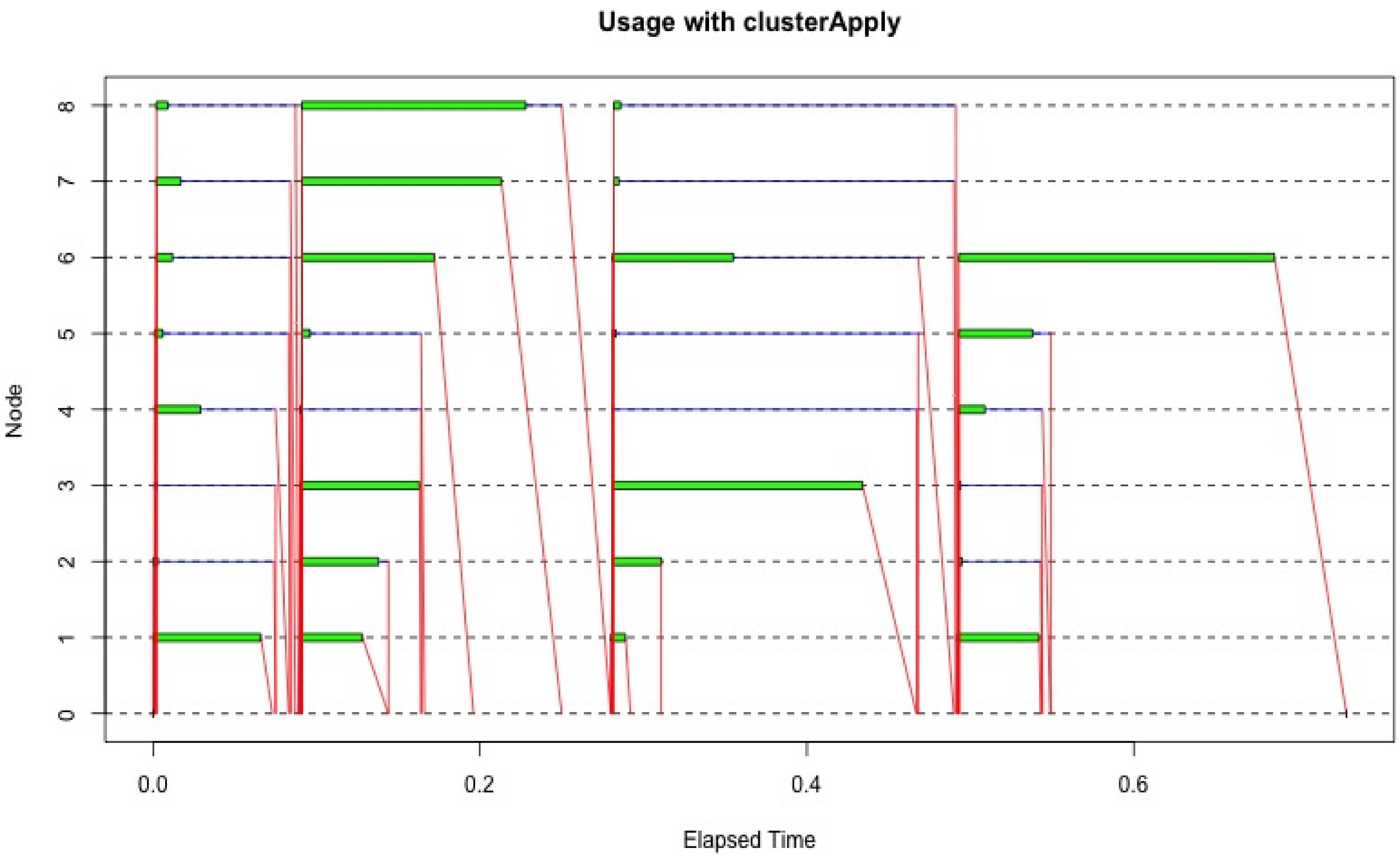
multicore

(S. Urbanek)



all OS
except Windows

mclapply, mcmapply,
mcMap


Usage with clusterApply
Node
Elapsed Time
0
1
2
3
4
5
6
7
8
0.0
0.2
0.4
0.6

Supported backends

Socket communication (default, all OS platforms)

```
cl <- makeCluster(ncores, type = "PSOCK")
```

- Workers start with an empty environment (i.e. new R process).

Supported backends

Forking (not available for Windows)

```
cl <- makeCluster(ncores, type = "FORK")
```

- Workers are complete copies of the master process.

Supported backends

Using the **MPI** library (uses Rmpi)

```
cl <- makeCluster(ncores, type = "MPI")
```

PARALLEL PROGRAMMING IN R

Let's practice!

PARALLEL PROGRAMMING IN R

The core of parallel

Hana Sevcikova
University of Washington

Core Functions

Main processing functions:

- clusterApply
- clusterApplyLB

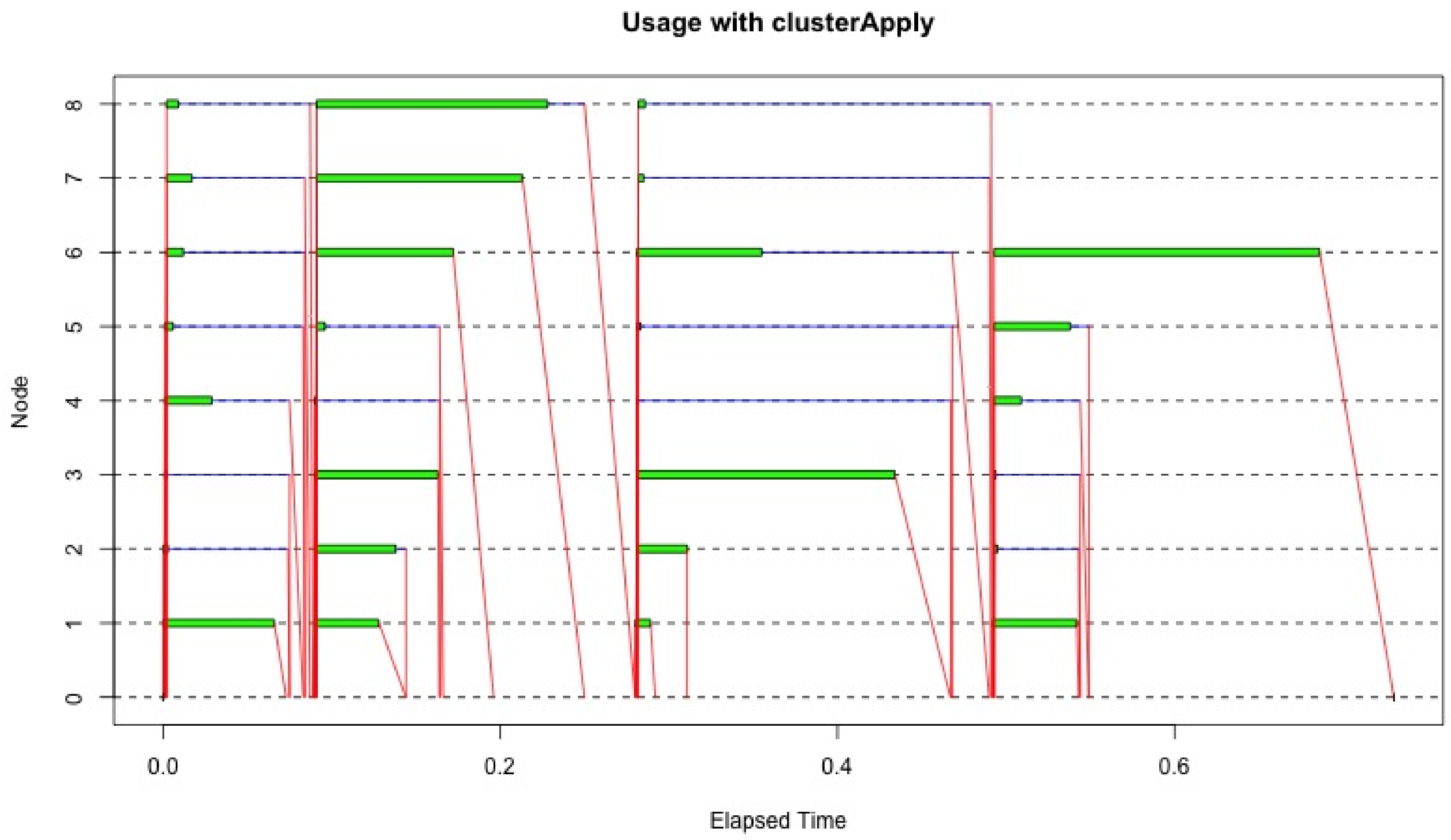
Wrappers:

- parApply, parLapply, parSapply
- parRapply, parCapply
- parLapplyLB, parSapplyLB

clusterApply: Number of tasks

```
clusterApply(cl, x = arg.sequence, fun = myfunc)
```

`length(arg.sequence)` = number of tasks (green bars)



Parallel vs. Sequential

Not all embarrassingly parallel aplications are suited for parallel processing.

Processing overhead:

- Starting/stopping cluster
- Number of messages sent between nodes and master
- Size of messages (sending big data is expensive)

Things to consider:

- How big is a single task (green bar)
- How much data need to be sent
- How much gain is there by running it in parallel $\longrightarrow$ **benchmark**

PARALLEL PROGRAMMING IN R

Let's practice!

PARALLEL PROGRAMMING IN R

Initialization of Nodes

Hana Sevcikova
University of Washington

Why to initialize?

- Each cluster node starts with an empty environment (no libraries loaded).
- Repeated communication with the master is expensive.

- Example:

```
clusterApply(cl, rep(1000, n), rnorm, sd = 1:1000)
```

  - Master sends a vector of 1:1000 to all `n` tasks (`n` can be very large).

- Good practice: Master initializes workers at the beginning with everything that stays constant or/and is time consuming. Examples:
  - sending static data
  - loading libraries
  - evaluating global functions

clusterCall

- Evaluates the same function with the same arguments on all nodes.

Example:

```
cl <- makeCluster(2)
clusterCall(cl, function() library(janeaustenr))
```

```
clusterCall(cl, function(i) emma[i], 20)
```

```
[[1]]
[1] "She was the youngest of the two daughters of a most affectionate,"

[[2]]
[1] "She was the youngest of the two daughters of a most affectionate,"
```

clusterEvalQ

- Evaluates a literal expression on all nodes (equivalent to `evalq()`)

Example:

```
cl <- makeCluster(2)
clusterEvalQ(cl, {
    library(janeaustenr)
    library(stringr)
    get_books <- function() austen_books()$book %>% unique %>% as.character
})
```

```
clusterCall(cl, function(i) get_books()[i], 1:3)
```

```
[[1]]
[1] "Sense & Sensibility" "Pride & Prejudice"   "Mansfield Park"

[[2]]
[1] "Sense & Sensibility" "Pride & Prejudice"   "Mansfield Park"
```

clusterExport

- Exports given objects from master to workers.

Example:

```
books <- get_books()
cl <- makeCluster(2)
clusterExport(cl, "books")
```

```
clusterCall(cl, function() print(books))
```

```
[[1]]
[1] "Sense & Sensibility" "Pride & Prejudice"   "Mansfield Park"
[4] "Emma"                "Northanger Abbey"    "Persuasion"

[[2]]
[1] "Sense & Sensibility" "Pride & Prejudice"   "Mansfield Park"
[4] "Emma"                "Northanger Abbey"    "Persuasion"
```

PARALLEL PROGRAMMING IN R

Let's practice!

PARALLEL PROGRAMMING IN R

Subsetting data

Hana Sevcikova
University of Washington

Data chunks

- Each task applied to different data (data chunk)
- Data chunks are passed to workers as follows:

  1. Random numbers generated on the fly
  2. Passing chunks of data as argument
  3. Chunking on workers' side

Data chunk as random numbers

```
myfunc <- function(n, ...) mean(rnorm(n, ...))

clusterApply(cl, rep(1000, 20), myfunc, sd = 6)
```

Data chunk as argument

- Dataset is chunked into several blocks on master
- Each block passed to worker via an argument
- Incorporated into higher level functions (`parApply()` etc)

```
cl <- makeCluster(4)
mat <- matrix(rnorm(12), ncol=4)
```

```
           [,1]      [,2]      [,3]       [,4]
[1,]  1.1540263 -2.180922 0.5322614  0.5578128
[2,] -1.8763588 -1.625226 0.4058091 -0.5532732
[3,] -0.1685597 -1.089104 0.1770636  0.5483025
```

Sum of columns (`colSums(mat)`):

```
parCapply(cl, mat, sum)
unlist(clusterApply(cl, as.data.frame(mat), sum))
```

- Sends each worker a column of `mat`

Chunking on workers' end

Example of matrix multiplication $M \times M$:

```
n <- 100
M <- matrix(rnorm(n * n), ncol = n)
clusterExport(cl, "M")
```

```
mult_row <- function(id) apply(M, 2, function(col) sum(M[id,] * col))
```

```
clusterApply(cl, 1:n, mult_row) %>% do.call(rbind, .)
```

PARALLEL PROGRAMMING IN R

Let's practice!