# Profiling and efficiency test in R


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# Packages and functions for memory profiling

- profvis package

- Memory: Memory allocated or deallocated (for negative numbers) for a given call stack. This is represented in megabytes and **aggregated** over all the call stacks over the code in the given row.
- Time: Time spent in milliseconds. This field is also **aggregated** over all the call stacks executed over the code in the given row
- More details in the previous meeting slides (page 12-14) made by Núria
- Tip: Sourcing the function file (instead of calling function from package) can show the profiling of each line.

# Packages and functions for memory profiling

- peakRAM package

  Small package with one function to tell you what's the peak RAM in a given chunk of code.

- memuse package
  - Nice user guide
  - Useful functions: Sys.filesize, Sys.meminfo, Sys,procmem, memuse

```
> memuse::Sys.filesize("/esarchive/exp/ecmwf/system5c3s/monthly_mean/tas_f6h/tas_19810101.nc")
26.647 MiB
> memuse::Sys.meminfo()
Totalram:  15.383 GiB   # Nord3-standard node: 32Gb; medmem node: 64Gb
Freeram:    6.946 GiB
> memuse::Sys.procmem()
Size:  180.852 MiB
Peak:  180.852 MiB
> memuse(res, unit = 'best')
8.922 KiB
```

# Some comparisons

(1) RAM used

- **memuse::Sys.procmem** shows the amount of ram used by the current R process
- **pryr::mem_used** shows how much memory is currently used by R. Sum-up of gc()

```
> pryr::mem_used()
31.2 MB
> memuse::Sys.procmem()
Size:  66.734 MiB
Peak:  66.734 MiB
```

(2) peak RAM

- **peakRAM::peakRAM** monitors the total and peak RAM used by any number of R expressions or functions
- **memuse::Sys.procmem** shows the amount of ram used by the current R process

```
> peakRAM::peakRAM({d <- func(10000)})

     Function_Call Elapsed_Time_sec Total_RAM_Used_MiB Peak_RAM_Used_MiB
1 {d <- func(10000)}           0.001                 0.1                0.2
```

# Some comparisons

(3) Data size

- **utils::object.size**
- **pryr::object_size** is more accurate than object.size
- **memuse::memuse**

```
> object.size(data)
1600784 bytes
> format(object.size(data), unit = 'auto')
[1] "1.5 Mb"
> pryr::object_size(data)
1,600,784 B
> pryr::compare_size(data)
   base    pryr
1600784 1600784
> memuse::memuse(data)
1.527 MiB
```

# Why do you need profiling?

**Spend some time on profiling = save more time in the long term!**

→ Check your script to find the efficiency bottleneck memory- or time-wise. If it happens in some functions, report in the corresponding GitLab.

→ Your tests would be more practical and meaningful than what we do.

→ Remember that multiApply could be heavy for light operation; try to use more cores and larger data to see if the performance makes sense.