future.mirai: Use the Mirai Parallelization Framework in Futureverse - Easy!

Henrik Bengtsson

University of California, San Francisco

R Foundation

R Consortium

@HenrikBengtsson

useR! 2024, Salzburg, Austria (2024-07-09)

Futureverse - A Friendly, Unifying Parallelization Framework in R

- Package **future** provides fundamental building blocks for evaluating R code in parallel
    - `future()`, `value()`, and `resolved()`
    - `%<-%` (future-assignment operator) on top of `future()` & `value()`

```
> x <- 1:100
> y <- slow_sum(x)                    # ~1 min ... waiting!
> y
[1] 5050
```

Total time: **1 minute**

Futureverse - A Friendly, Unifying Parallelization Framework in R

- Package **future** provides fundamental building blocks for evaluating R code in parallel
  - `future()`, `value()`, and `resolved()`
  - `%<-%` (future-assignment operator) on top of `future()` & `value()`

```
> x <- 1:100
> a <- slow_sum(x[ 1:50 ])      # ~30 sec
> b <- slow_sum(x[51:100])      # ~30 sec
> y <- a + b
> y
[1] 5050
```

Total time: **1 minute**

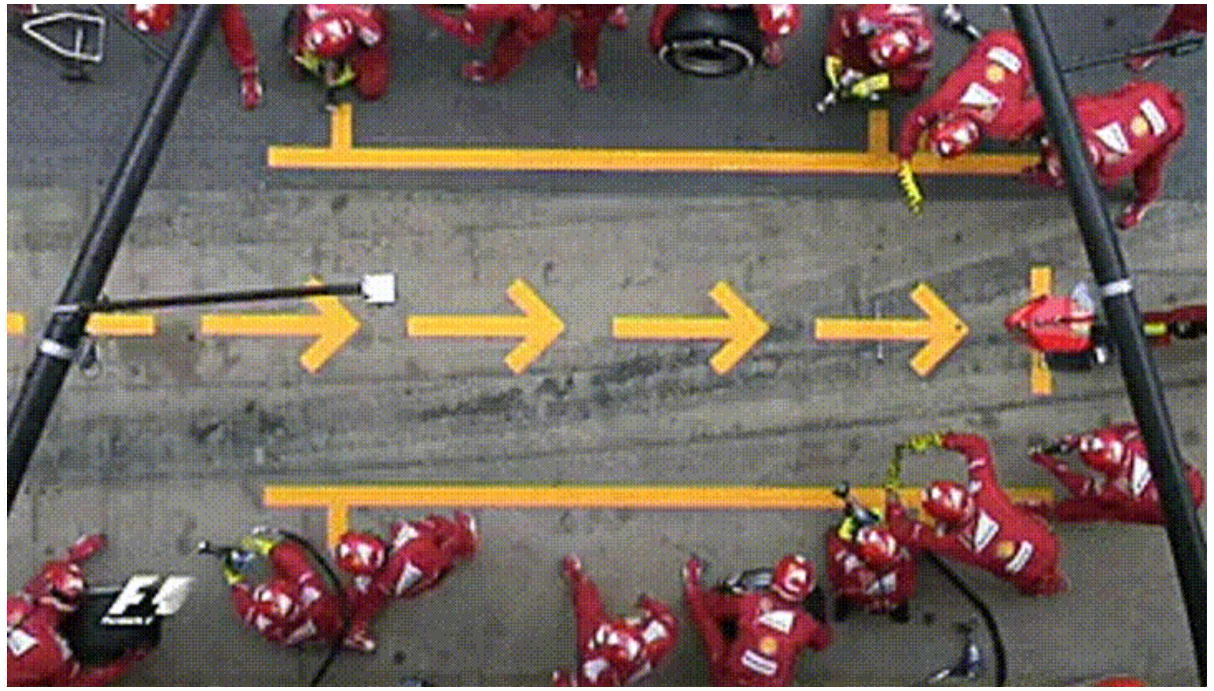
Evaluate R expressions in the background

```
> library(future)
> plan(multisession)              # parallel
                                  # local ma
> x <- 1:100
> a %<-% slow_sum(x[ 1:50 ])      # ~0 sec
> b %<-% slow_sum(x[51:100])      # ~0 sec

> 1 + 2
[1] 3

> y <- a + b                      # get resu
> y
[1] 5050
```

Total time: **30 seconds**

Splitting up into more chunks to speed it up further

```
> library(future)
> plan(multisession)

> x <- 1:100
> a %<-% slow_sum(x[ 1:25 ])    # ~0 sec
> b %<-% slow_sum(x[26:50 ])    # ~0 sec
> c %<-% slow_sum(x[51:75 ])    # ~0 sec
> d %<-% slow_sum(x[76:100])    # ~0 sec

> y <- a + b + c + d            # get resu
> y
[1] 5050
```



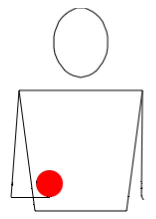
Total time: **15 seconds**

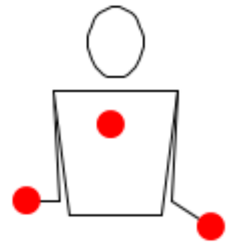
End-user can choose from many parallel backends

```r
plan(sequential)
plan(multisession)          # uses {parallel}'s "snow" machinery
plan(multicore)             # uses {parallel}'s "multicore" machinery

plan(cluster, workers = c("n1", "n1", "n1", "n2", "n3"))
plan(cluster, workers = c("n1", "m2.uni.edu", "vm.cloud.org"))
```

These are internally based on the **parallel** package.

Higher-level parallelization from `future()` and `value()`





```r
y <- lapply(X, slow_sum)
```

```r
plan(multisession, workers = 4
y <- future_lapply(X, slow_sum
```

Easily implemented via `future()` and `value()`, e.g.

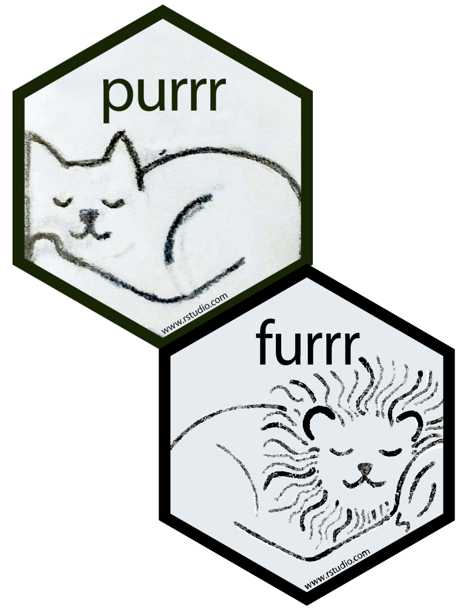
```r
future_lapply <- function(X, FUN, ...) {
  fs <- lapply(X, function(x) future(FUN(x, ...)))
  lapply(fs, value)
}
```

User-friendly, higher-level functions

- The concept of "futures" was invented in 1975 (sic!)
- `future()`, `value()`, and `resolved()` are easy to understand, powerful constructs

These building blocks lay the foundation for higher-level functions:

- **future.apply**, e.g. `future_lapply()` and `future_replicate()`
- **furrr**, e.g. `future_map()` and `future_map_dbl()`
- **doFuture**, e.g. `foreach() %dofuture% { ... }`
- ...
- *- Maybe your package will be next?*



Futureverse allows you to stick with your favorite coding style

Parallel alternatives to traditional, sequential functions:

```r
y <- lapply(x, some_fcn)                     ## base R
y <- future_lapply(x, some_fcn)              ## {future.apply}
```

```r
y <- map(x, some_fcn)                        ## {purrr}
y <- future_map(x, some_fcn)                 ## {furrr}
```

```r
y <- foreach(z = x) %do% some_fcn(x)         ## {foreach}
y <- foreach(z = x) %dofuture% some_fcn(z)   ## {foreach} + {doFuture}
```

Yes, we can of course use base-R or **magrittr** pipes where we want to, e.g.

```r
y <- x |> future_map(some_fcn)
```

```r
y <- x %>% future_map(some_fcn)
```

Anyone can develop additional parallel backends

From the very beginning in 2015, the plan and hope has been that additional R backends would become available in the future (pun intended!)

The **future.callr** package wraps the **callr** package that is an alternative to the built-in **parallel**-based `multisession` backend:

```
plan(future.callr::callr)                       # locally using callr
```

The **future.batchtools** package wraps the **batchtools** package that can run tasks on high-performance compute (HPC) clusters, e.g.

```
plan(future.batchtools::batchtools_slurm)       # on a Slurm job scheduler
plan(future.batchtools::batchtools_sge)         # on a SGE job scheduler
```

And, now also **mirai**-based backends:

```
plan(future.mirai::mirai_multisession)          # locally using mirai
plan(future.mirai::mirai_cluster)               # using mirai cluster
```

mirai - Minimalist Async Evaluation Framework for R

- The **mirai** R package by Charlie Gao (anno 2022)
- *mirai* is Japanese for “future”

Mirai API		Future API
`m <- mirai(expr)`	create a future	`f <- future(expr)`
`r <- !unresolved(m)`	check if done	`r <- resolved(f)`
`v <- m[]`	wait & get result	`v <- value(f)`

- Somewhat lower-level interface than the **future** package
- Minimum *overhead* through highly optimized implementation

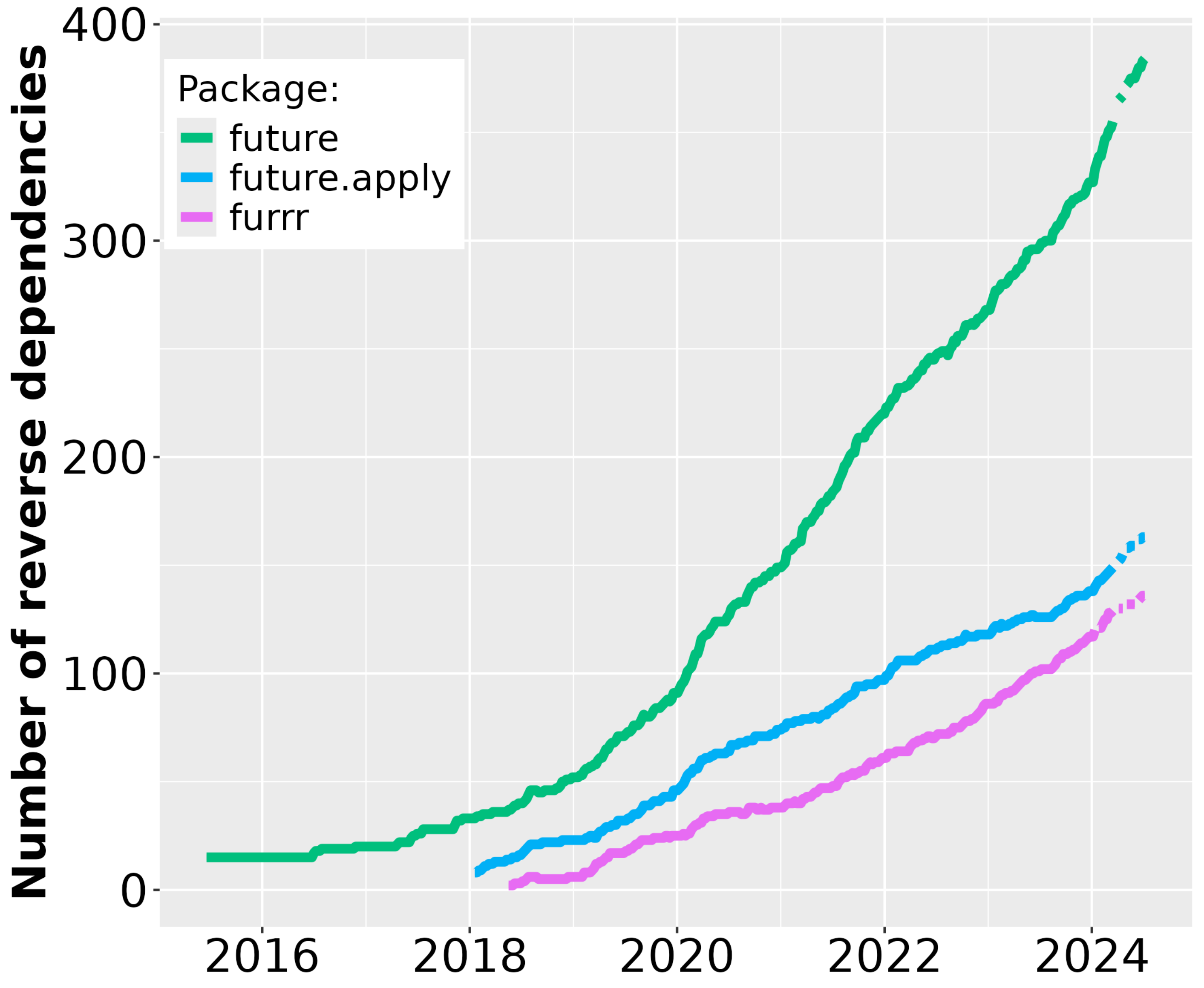
- **Provides a powerful queueing-mechanism for processing tasks in parallel**

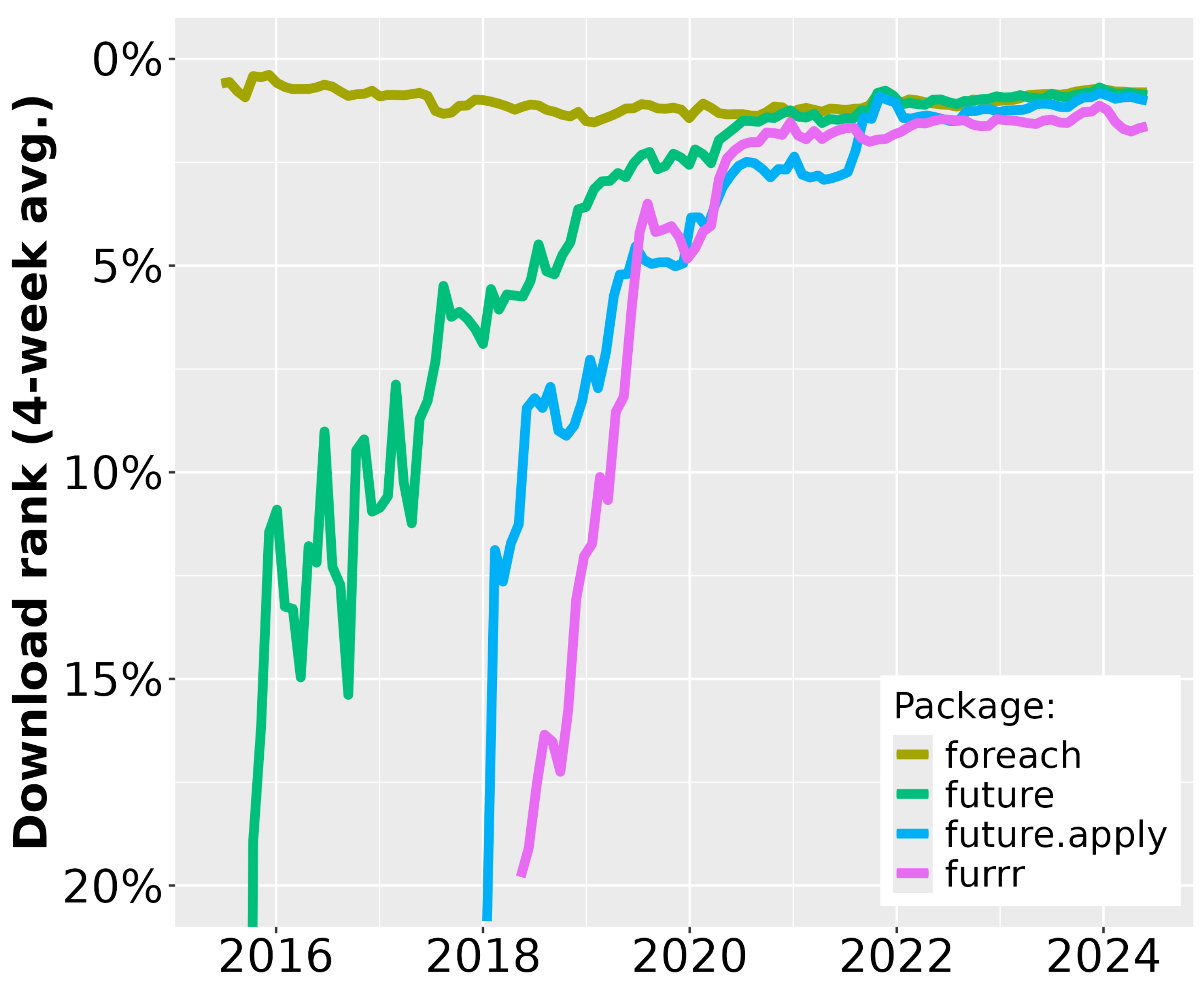
future.mirai - A mirai-based parallel backend for Futureverse

- Makes mirai ecosystem available to Futureverse
- Existing Futureverse code can use it without modification

```r
library(future.mirai)
plan(mirai_multisession)    # parallelize via
                            # mirai framework
x <- rnorm(100)
a %<-% sum(x[ 1:50 ])
b %<-% sum(x[51:100])
y <- a + b

z <- future.apply::future_sapply(x, slow)

z <- x |> furrr::future_map_dbl(slow)

z <- foreach::foreach(.x = x) %dofuture% { slo
```

Futureverse is very well tested thanks to lots of real-world use





~400 CRAN packages depend directly on the **future** package - it grows 3× faster than **foreach** at 1200 reverse dependencies

The **future** packages is among **the 1% most downloaded** CRAN packages

As a Futureverse user, you can help mirai!

If you use one of the 100’s of CRAN packages that parallelize via Futureverse, by setting:

```
plan(future.mirai::mirai_multisession)
```

you will parallelize via **mirai**, with all its benefits.

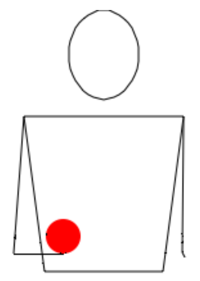
Importantly, by doing so, you will also:

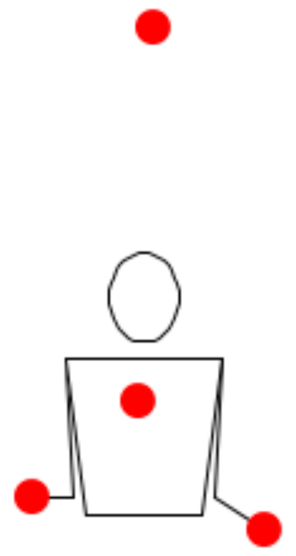
- increase the real-world test coverage of **mirai**
- help increase the stability and quality of **mirai**

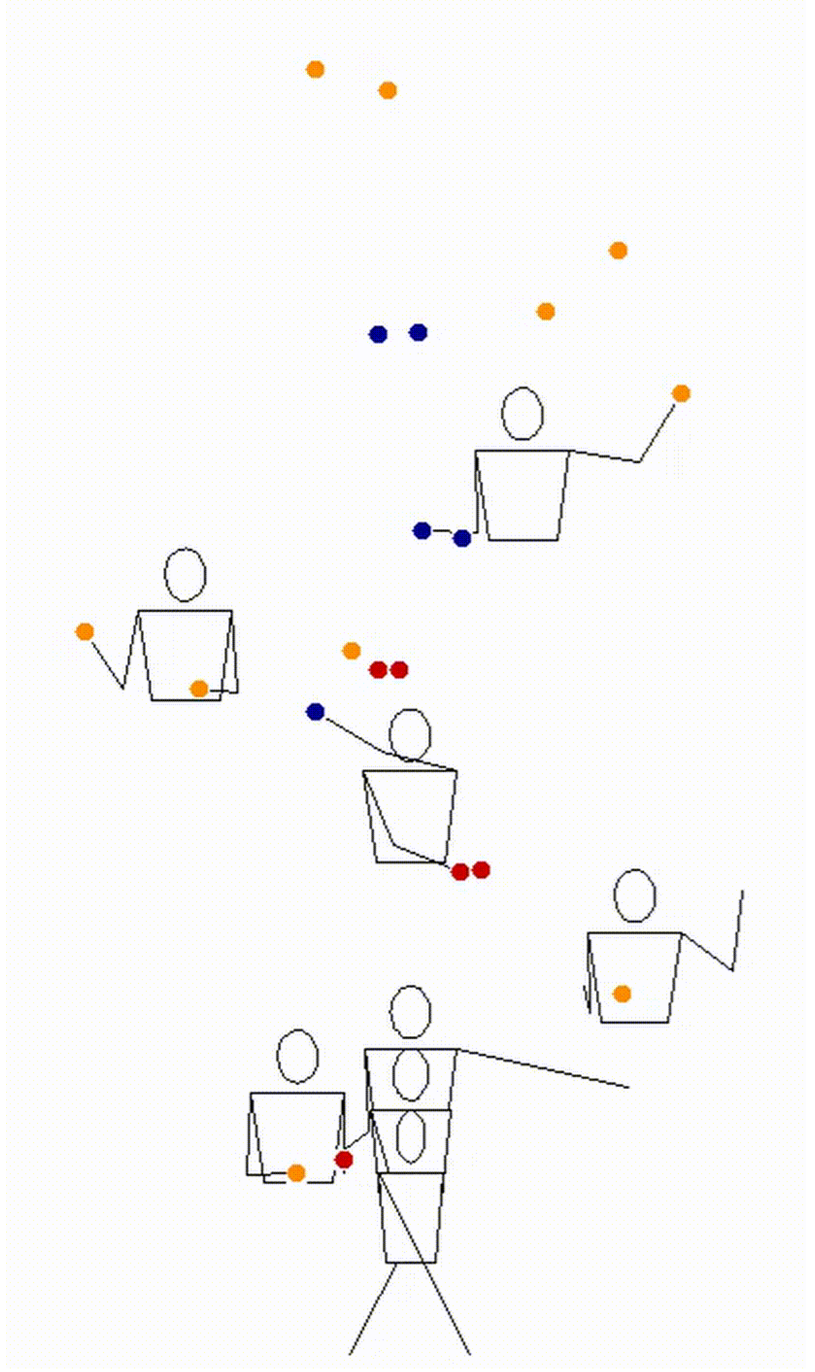
Please give feedback and reach out if you run into issues 🙏

Nested parallelization with some care

- Futureverse protects against over parallelization
- **future.mirai** opens up for more nested parallelization


Sequential processing


Parallelization with 4 workers


Nested parallelization with 5 workers each running 3 parallel tasks

Not a competition: Should I use Future API or Mirai API?

- Futureverse is well established and well tested
  - **future.apply**, **furrr**, **foreach** with **doFuture**, …
  - automatically relays output, messages, warnings, errors, etc.
  - real-time progress updates
  - 100's of packages already parallelize via futures
    ⇒they can all use `plan(mirai_multisession)` immediately
- **mirai** is is self-contained implementation
  - optimized for minimum overhead
  - undergoes stunning development

There is a promising future for parallelization in R

- It's easy to get started - just try it
- Support: https://github.com/HenrikBengtsson/future/discussions
- Tutorials: https://www.futureverse.org/tutorials.html
- Blog posts: https://www.futureverse.org/blog.html
- More features on the roadmap
- I love feedback and ideas