{{progressr}}

An Inclusive, Unifying API for Progress Updates

@HenrikBengtsson

University of California San Francisco, R Foundation, R Consortium

Applying a slow function to a vector

```
slow <- function(x) {
  Sys.sleep(1.0)
  sqrt(x)
}

x <- 1:50
y <- lapply(x, function(z) {
  slow(z)
})
```

=> 50 seconds to complete

With slow functions we want to know ...

- Is it still running?

  `/ Processing ...`

- How much longer? (secs, mins, hours, days)

  `[=========>-------------]  40% ETA  8s`

Progress updates in R

utils::txtProgressBar() - built-in & basic

```
x <- 1:50
pb <- txtProgressBar(max=length(x))
y <- lapply(x, function(z) {
  setTxtProgressBar(pb, getTxtProgressBar(pb)+1)
  slow(z)
})
```

```
|=============                                         |  42%
```

progress: beautiful progress bars

```
x <- 1:50
pb <- progress::progress_bar$new(total=length(x))
y <- lapply(x, function(z) {
  pb$tick()
  slow(z)
})
```

```
[============>-----------------]  42%
```

PROGRESS

by Gábor Csárdi

Things we need to be aware of

Let user control progress updates

```
snail <- function(x, progress = FALSE) {
  if (progress) pb <- progress::progress_bar$new(total=length(x))

  lapply(x, function(z) {
    if (progress) pb$tick()
    slow(z)
  })
}

> x <- 1:50
> y <- snail(x, progress=TRUE)
```

We must be careful with output

```
snail <- function(x, progress = FALSE) {
  if (progress) pb <- progress::progress_bar$new(total=length(x))

  lapply(x, function(z) {
    if (!progress) message("z=", z)
    if (progress) pb$tick()
    slow(z)
  })
}
```

```
z=1
[=>---------]  10%z=2
[==>--------]  20%
```

```
> y <- snail(x)                  # with messages
> y <- snail(x, progress=TRUE)   # no messages
```

It doesn't work with parallel processing

```
library(parallel)
cl <- makeCluster(4)
x <- 1:50
pb <- progress::progress_bar$new(total=length(x))
clusterExport(cl, c("slow", "pb"))
y <- parLapply(cl, x, function(z) {
  pb$tick()
  slow(z)
})
```

=> *no progress bar (output from workers is dropped)*

We can do better with R's condition framework

progressr - Unifying API for Inclusive Progress Updates

Can be used with for loops, while loops, lapply, purrr, foreach, ...

API for Developers:

```
p <- progressor(along=x)
p()
```

API for Users:

```
with_progress({ expr })
```

<u>Developer decides:</u>

where in the code progress updates should be signaled

<u>User decides:</u>

if, when, and how progress updates are presented

CRAN 0.6.0

It's all about signalling progress

```
snail <- function(x) {
  p <- progressr::progressor(along=x)
  lapply(x, function(z) {
    p()                ⇐ signal a progress condition
    slow(z)
  })
}

# Just a regular function
> x <- 1:50
> y <- snail(x)
>
```

User decides how progress is presented

```
# without progress updates
> x <- 1:50
> y <- snail(x)


> with_progress(y <- snail(x))
[======>-----------]  40%




> handlers("progress", "beepr")
> with_progress(y <- snail(x))
[======>-----------]  40%
```

♬ ♪ ♪ ♪ … ♬

beepr by Rasmus Bååth

Worry free use of cat() and message()

```
snail <- function(x) {
  p <- progressr::progressor(along=x)
  lapply(x, function(z) {
    message("z=", z)        ⇐ No worries!
    p()
    slow(z)
  })
}

> with_progress(y <- snail(1:10))
z=1
z=2
 [======>---------------]  20%
```

Parallelization: progressr + future = ❤️

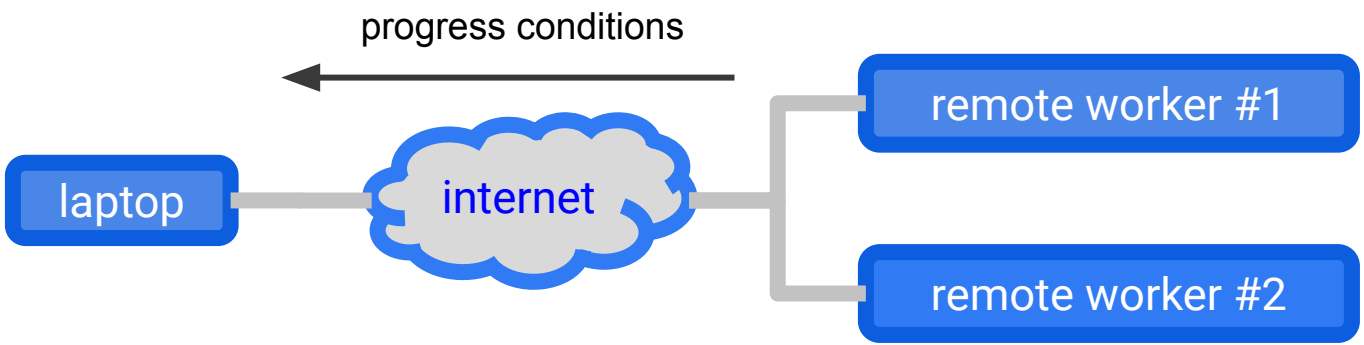
Works with any future API:

- future.apply
- furrr
- foreach /w doFuture
- BiocParallel

```
snail <- function(x) {
  p <- progressor(along=x)
  future_lapply(x, function(z) {
    p()
    slow(z)
  })
}
```

```
> plan(multisession)
> with_progress(y <- snail(x))
[=========>---------------]  40%
```

future + progressr: also distributed processes




```
library(future)
plan(cluster, workers = c("w1.remote.org", "w2.remote.org"))

with_progress({
  y <- snail(1:1000)   ⇐ parallelization via futures internally
})
[======>-----------]  40%
```

Take-home messages

- Developer decides *what* progress to report on

  ```
  p <- progressor(...), p()
  ```

- End-user decide *when and how* progress is reported

  ```
  with_progress(...), handler(...)
  ```

- Output doesn’t clash with existing progress information

- Works with parallel processing using futures

Create new progress handlers for end-users

Existing:

- utils::txtProgressBar() [default]
- progress::progress_bar()
- beepr::beep()
- Shiny, ...

I encourage you to build handlers for:

- Pushbullet, Twitter, Telegram, SMS
- Email notifications
- Change color on a smart light bulb (ZigBee)
- ...

There's no limit to what you can do

```
> handler("ransid_image")
> with_progress(y <- snail(1:1000))
```

Thank you! 💜

@HenrikBengtsson

HenrikBengtsson/progressr
install.packages("progressr")

ransid by Mike FC