Ateliers R³

Session 3 - R, du code performant

Au menu aujourd'hui:

La performance (allégorie)

Les languages de programmation

"haut niveau" ⟵ ⟶ "bas niveau"

ASM

```
PALINDRO CSECT
         USING  PALINDRO,R13
         B      72(R15)
         DC     17F'0'
         STM    R14,R12,12(R13)
         ST     R13,4(R15)
         ST     R15,8(R13)
         LR     R13,R15
         LA     R8,BB
         LA     R9,AA+L'AA-1
         LA     R6,1
LOOPI    C      R6,=A(L'AA)
         BH     ELOOPI
         MVC    0(1,R8),0(R9)
         LA     R8,1(R8)
         BCTR   R9,0
[...] (+33 lignes)
```

Les languages de programmation

"haut niveau" ⟵ ⟶ "bas niveau"

C

```c
int palindrome(const char *s)
{
   const char *t;
   for (t = s; *t != '\0'; t++) ; t--;
   while (s < t)
   {
     if ( *s++ != *t-- ) return 0;
   }
   return 1;
}
```

ASM

```
PALINDRO CSECT
         USING  PALINDRO,R13
         B      72(R15)
         DC     17F'0'
         STM    R14,R12,12(R13)
         ST     R13,4(R15)
         ST     R15,8(R13)
         LR     R13,R15
         LA     R8,BB
         LA     R9,AA+L'AA-1
         LA     R6,1
LOOPI    C      R6,=A(L'AA)
         BH     ELOOPI
         MVC    0(1,R8),0(R9)
         LA     R8,1(R8)
         BCTR   R9,0
[...] (+33 lignes)
```

Les languages de programmation

"haut niveau" ⟵ ⟶ "bas niveau"

R

```
is_palin <- function(str) {
  split <- unlist(strsplit(str, ""))
  all( split == rev(split) )
}
```

C

```
int palindrome(const char *s)
{
   const char *t;
   for (t = s; *t != '\0'; t++) ; t--;
   while (s < t)
   {
     if ( *s++ != *t-- ) return 0;
   }
   return 1;
}
```

ASM

```
PALINDRO CSECT
         USING  PALINDRO,R13
         B      72(R15)
         DC     17F'0'
         STM    R14,R12,12(R13)
         ST     R13,4(R15)
         ST     R15,8(R13)
         LR     R13,R15
         LA     R8,BB
         LA     R9,AA+L'AA-1
         LA     R6,1
LOOPI    C      R6,=A(L'AA)
         BH     ELOOPI
         MVC    0(1,R8),0(R9)
         LA     R8,1(R8)
         BCTR   R9,0
[...] (+33 lignes)
```

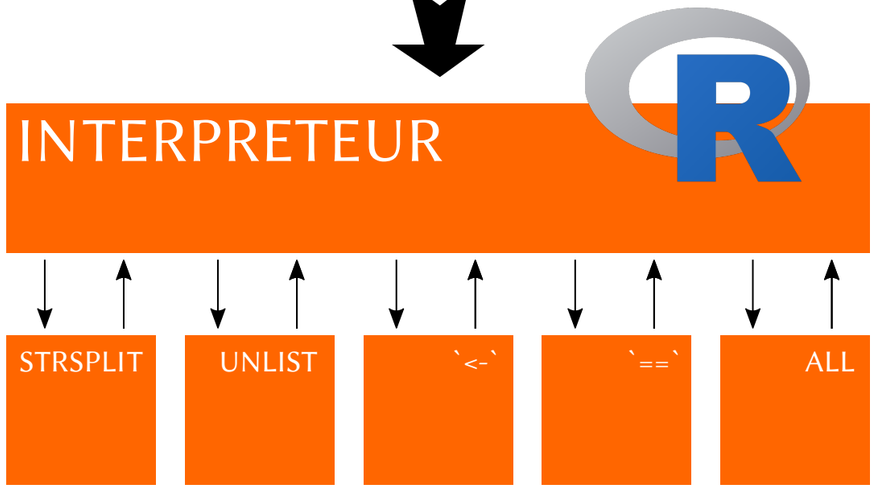
Les languages de programmation

"haut niveau" ⟵ ⟶ "bas niveau"

R

```
is_palin <- function(str) {
  split <- unlist(strsplit(str, ""))
  all( split == rev(split) )
}
```

script (texte)

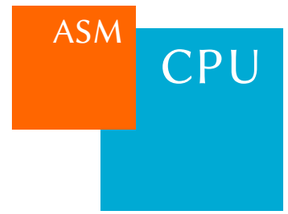


C

```
int palindrome(const char *s)
{
   const char *t;
   for (t = s; *t != '\0'; t++) ; t--;
   while (s < t)
   {
     if ( *s++ != *t-- ) return 0;
   }
   return 1;
}
```

ASM

```
PALINDRO CSECT
         USING  PALINDRO,R13
         B      72(R15)
         DC     17F'0'
         STM    R14,R12,12(R13)
         ST     R13,4(R15)
         ST     R15,8(R13)
         LR     R13,R15
         LA     R8,BB
         LA     R9,AA+L'AA-1
         LA     R6,1
LOOPI    C      R6,=A(L'AA)
         BH     ELOOPI
         MVC    0(1,R8),0(R9)
         LA     R8,1(R8)
         BCTR   R9,0
[...] (+33 lignes)
```

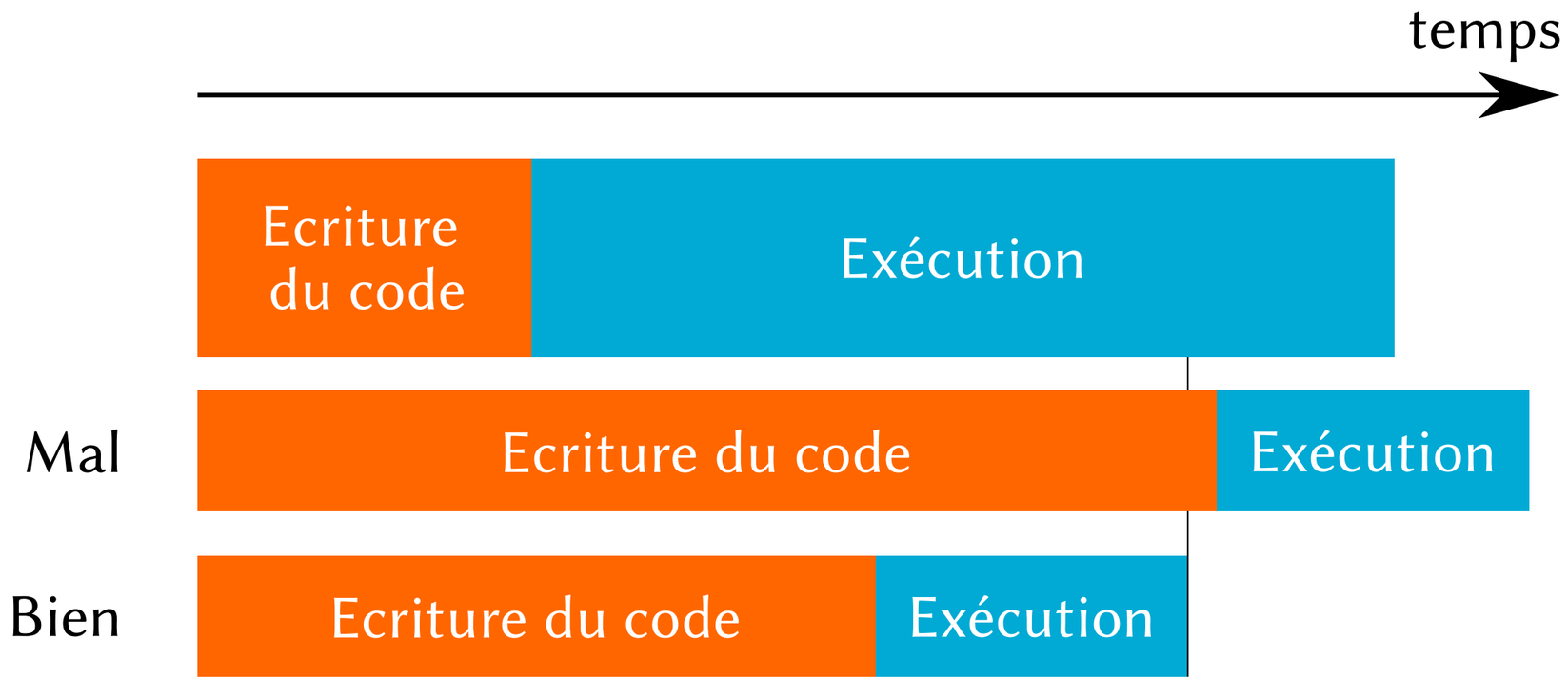


Optimiser son script

Aujourd'hui:

1. Vectorisation
2. Fonctions et structures de données spécialisées
3. Exécution en parallèle

Quand optimiser son script ?

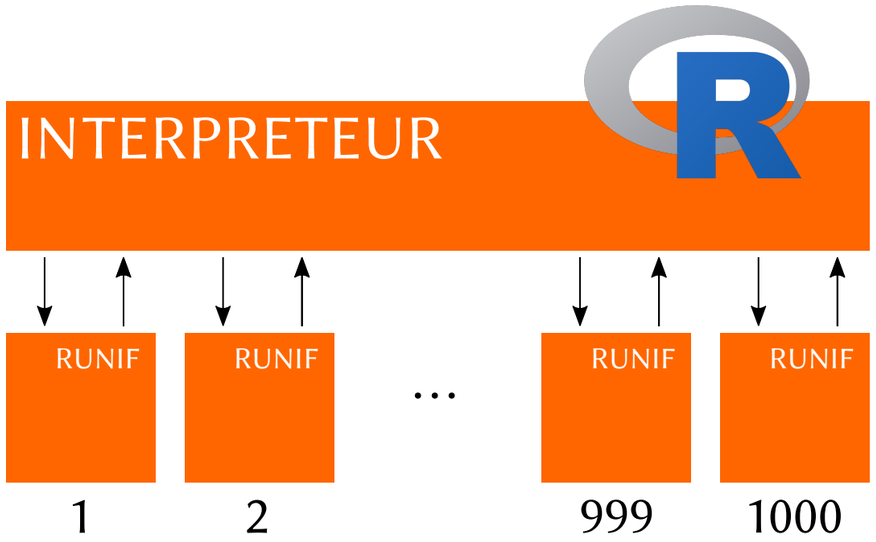


Une stratégie: la vectorisation

Exemple: tirer 1000 nombres aléatoires entre 0 et 1

Pas vectorisé

```
numbers <- numeric(1000)
for ( i in seq(1, 1000) ) {
  numbers[i] <- runif(1, 0, 1)
}
```

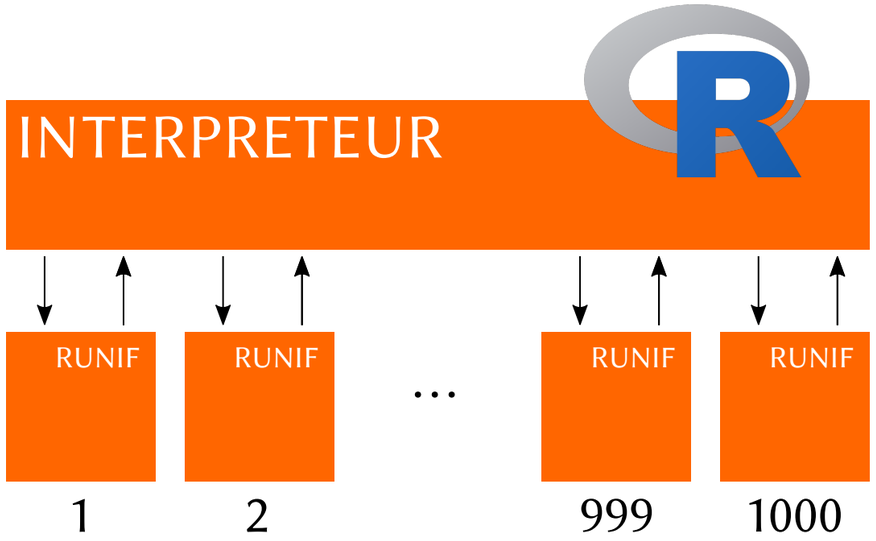


Une stratégie: la vectorisation

Exemple: tirer 1000 nombres aléatoires entre 0 et 1

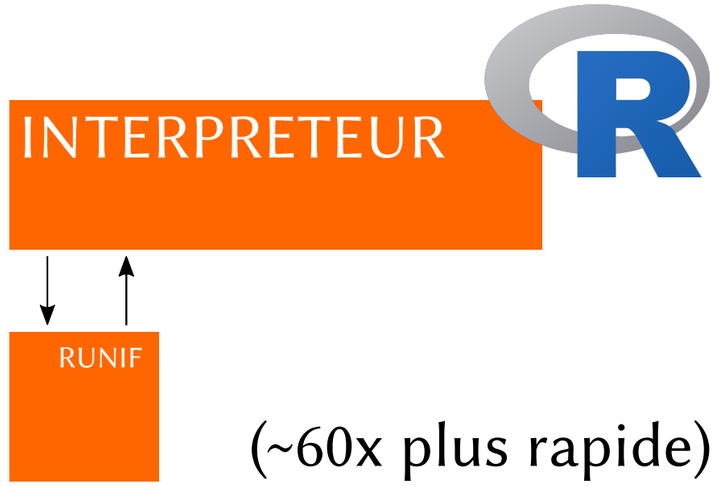
Pas vectorisé

```
numbers <- numeric(1000)
for ( i in seq(1, 1000) ) {
  numbers[i] <- runif(1, 0, 1)
}
```



Vectorisé

```
numbers <- runif(1000, 0, 1)
```



(~60x plus rapide)

Une stratégie: la vectorisation

```
lettres <- c("a", "b", "c", "d")
for ( i in lettres ) {
  lettres[i] <- toupper(lettres[i])
}
```

```
lettres <- c("a", "b", "c", "d")
lettres <- toupper(lettres)
```

```
x <- c(0, 1, 5, 10)
a <- 2.1
b <- 3.1
y <- numeric(4)
for ( i in seq(1, length(x)) ) {
  y[i] <- a + b * x[i]
}
```

```
x <- c(0, 1, 5, 10)
a <- 2.1
b <- 3.1
y <- a + b * x
```

Une stratégie: la vectorisation. Moyennes par colonnes d'un tableau de données

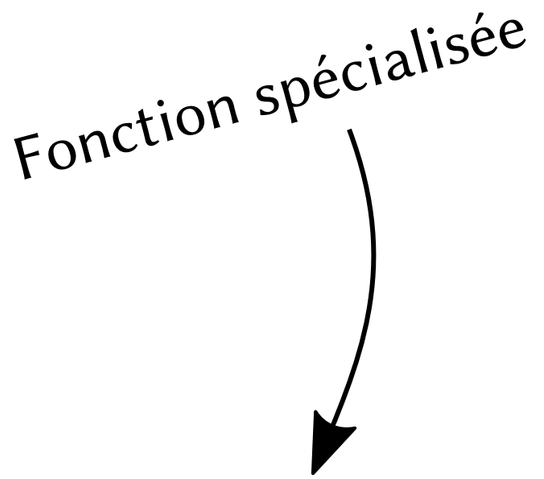
```
abundances
     espece_1 espece_2 espece_3 ...
[1,]      1.1      4.4      8.9
[2,]      2.2      4.8      9.0
...       ...      ...      ...
```

```r
means <- numeric(ncol(abundances))
for (i in seq(1, ncol(abundances)) ) {
  means[i] <- mean(abundances[ ,i])
}
```

Une stratégie: la vectorisation. Moyennes par colonnes d'un tableau de données

```
abundances
     espece_1 espece_2 espece_3 ...
[1,]      1.1      4.4      8.9
[2,]      2.2      4.8      9.0
...       ...      ...      ...
```

```
means <- numeric(ncol(abundances))
for (i in seq(1, ncol(abundances)) ) {
  means[i] <- mean(abundances[ ,i])
}
```




```
colMeans(abundances)
```

Une stratégie: la vectorisation.

Remplacer les boucles for par des applications de fonctions

```
ex <- c("coude", "pied", "nez")
str <- rep("", length(ex))
for ( part in ex ) {
  str[part] <- paste("On les plante avec le", part,
                     "à la mode de chez nous")
}
```

Une stratégie: la vectorisation.

Remplacer les boucles for par des applications de fonctions

```
ex <- c("coude", "pied", "nez")
str <- rep("", length(ex))
for ( part in ex ) {
  str[part] <- paste("On les plante avec le", part,
                     "à la mode de chez nous")
}
```

Avec un vecteur: sapply()

```
ex <- c("coude", "pied", "nez")
les_choux <- function(part) {
  str <- paste("On les plante avec le", part,
               "à la mode de chez nous")
  return(str)
}
sapply(ex, les_choux)
```

Une stratégie: la vectorisation.

Remplacer les boucles for par des applications de fonctions

```
ex <- c("coude", "pied", "nez")
str <- rep("", length(ex))
for ( part in ex ) {
  str[part] <- paste("On les plante avec le", part,
                     "à la mode de chez nous")
}
```

Avec un vecteur: sapply()

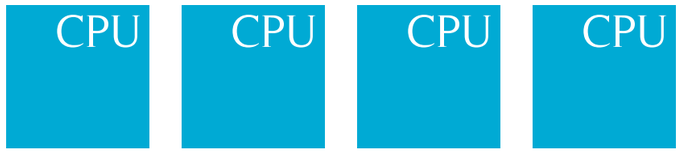
```
ex <- c("coude", "pied", "nez")
les_choux <- function(part) {
  str <- paste("On les plante avec le", part,
               "à la mode de chez nous")
  return(str)
}
sapply(ex, les_choux)
```

Avec une liste: lapply()

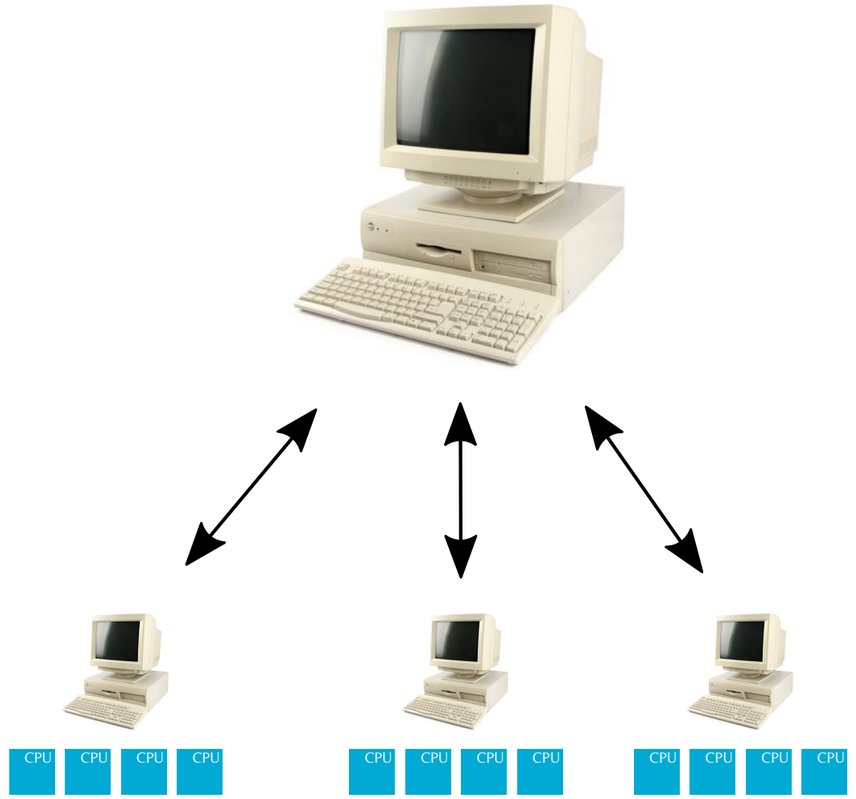
```
ex <- list("coude", "pied", "nez")
les_choux <- function(part) {
  str <- paste("On les plante avec le", part,
               "à la mode de chez nous")
  return(str)
}
lapply(ex, les_choux)
```

La parallélisation

Sur votre ordinateur



Sur d'autres ordinateurs



La parallélisation: en pratique

De nombreux paquets disponibles: parallel, foreach, snow, rmpi, etc.

Sur votre machine: future + future.apply

La parallélisation: en pratique

Avec une liste

```
ex <- list("coude", "pied", "nez")
les_choux <- function(part) {
  ligne <- paste("On les plante avec le", part,
                 "à la mode de chez nous")
  return(ligne)
}
lapply(ex, les_choux)
```

Exécution parallèle: lapply => future_lapply

```
ex <- list("coude", "pied", "nez")
plan(multisession, workers = 3)
future_lapply(ex, les_choux)
```

c'est tout

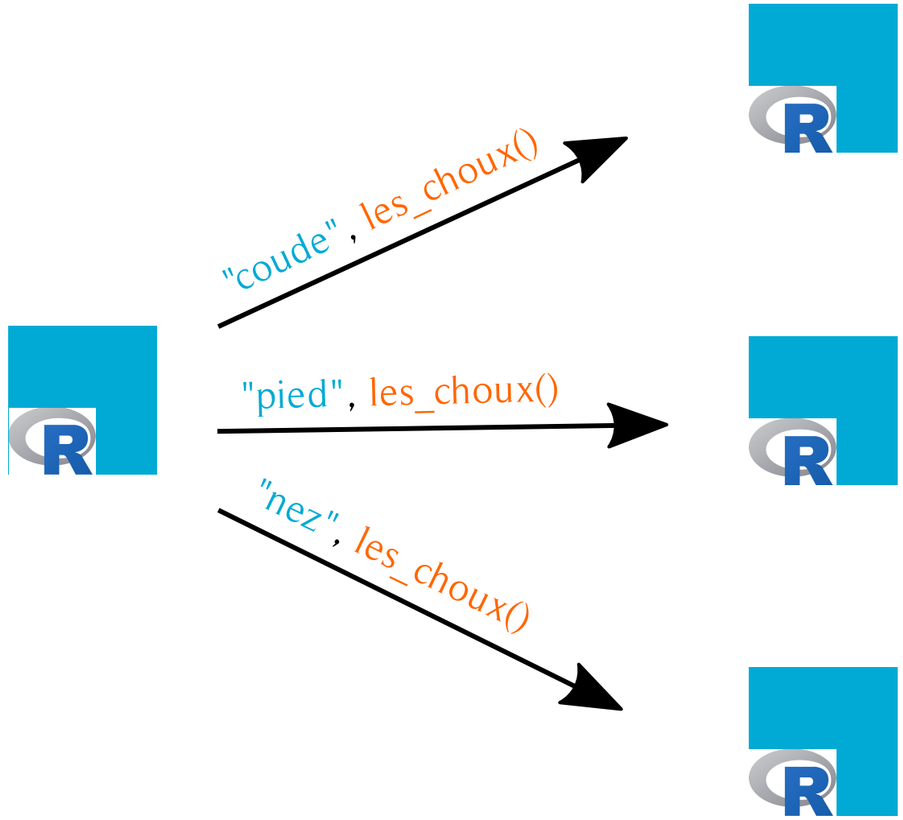
La parallélisation: en pratique

```
plan(multisession, workers = 3)
```

```
ex <- list("coude", "pied", "nez")
plan(multisession, workers = 3)
future_lapply(ex, les_choux)
```

c'est tout

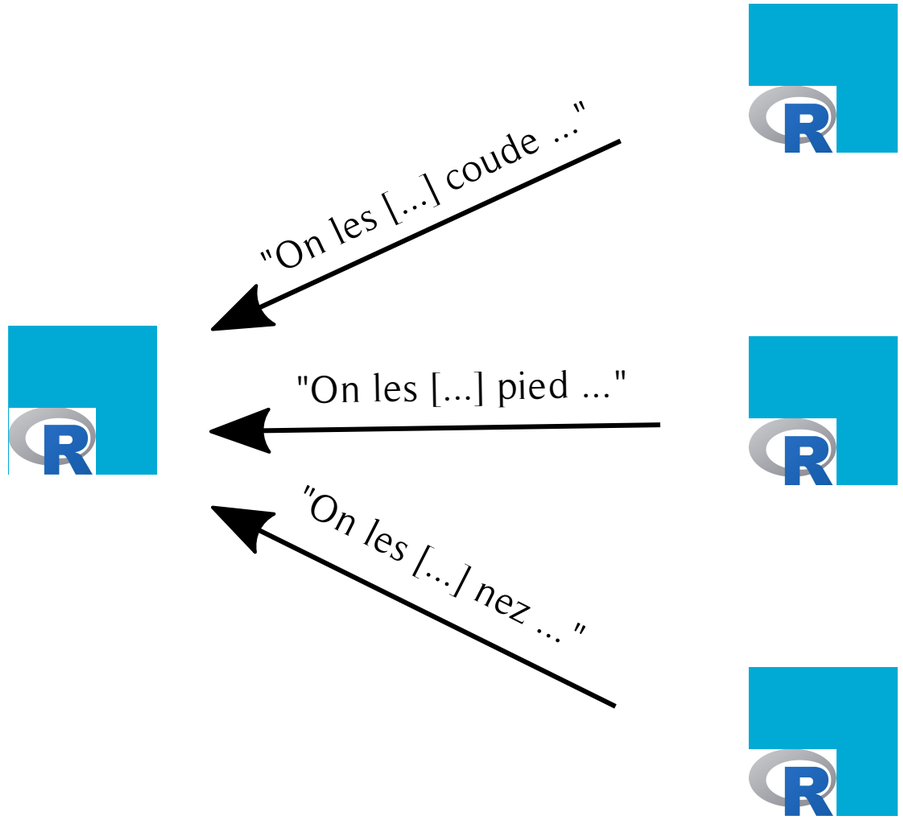
La parallélisation: en pratique




```
ex <- list("coude", "pied", "nez")
plan(multisession, workers = 3)
future_lapply(ex, les_choux)
```

c'est tout

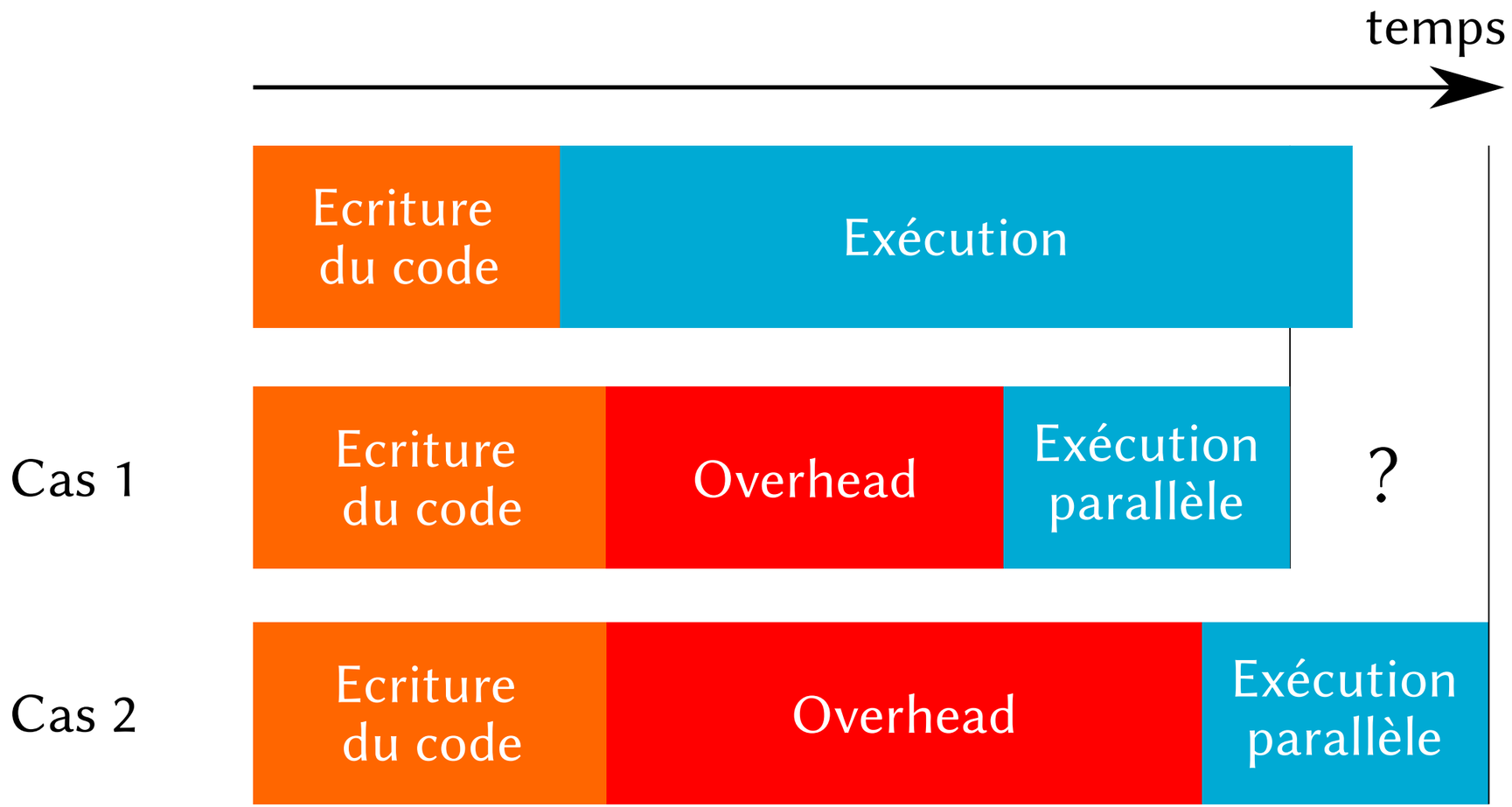
La parallélisation: en pratique




```
ex <- list("coude", "pied", "nez")
plan(multisession, workers = 3)
future_lapply(ex, les_choux)
```

c'est tout

Optimiser son script



En pratique...

Recap:

- **Pourquoi R est relativement lent**
- **Quelques stratégies pour améliorer la vitesse d'exécution**
  - **Vectorisation**
  - **Exécution parallèle**
- (pas vu) **Mesurer la vitesse d'exécution**

Atelier: jeudi 22 octobre, 14h !

Tous les exercices et infos sur https://rrr.mbb.cnrs.fr