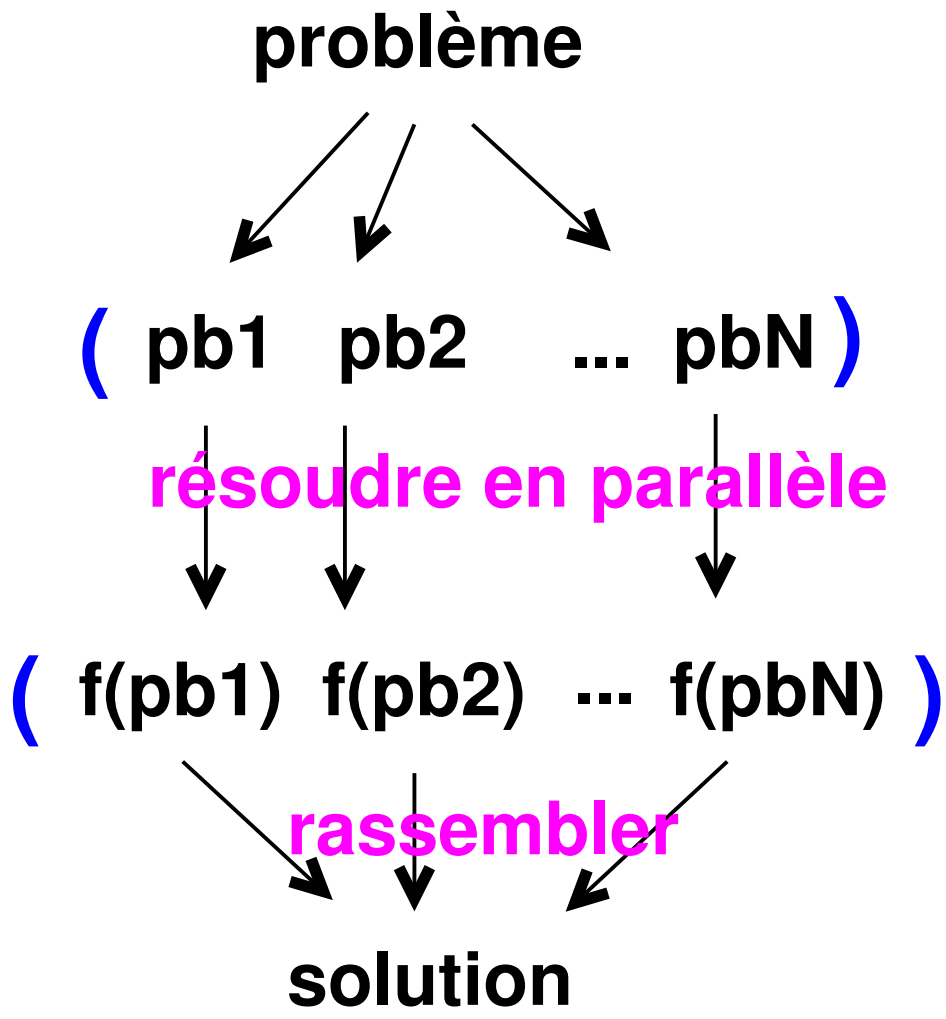
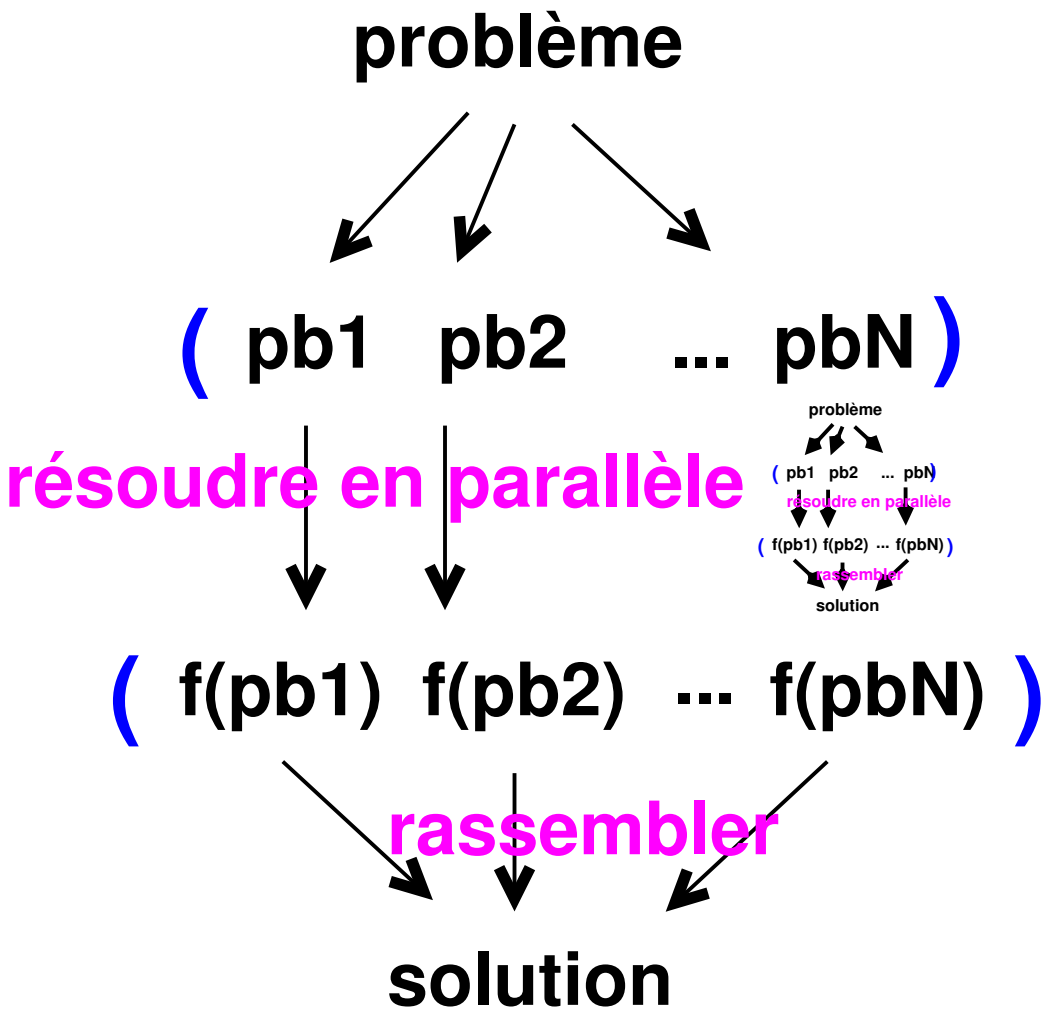
(Séquence 5.3

reduce

map-reduce


problème
(pb1 pb2 ... pbN)
résoudre en parallèle
(f(pb1) f(pb2) ... f(pbN))
rassembler
solution

map-reduce


problème
(pb1 pb2 ... pbN)
résoudre en parallèle
problème
(pb1 pb2 ... pbN)
résoudre en parallèle
(f(pb1) f(pb2) ... f(pbN))
rassembler
solution
(f(pb1) f(pb2) ... f(pbN))
rassembler
solution

Combiner les éléments d'une liste

Combiner entre eux les éléments d'une liste, à l'aide d'une fonction binaire *fn*

- soit *L1* une liste d'éléments de type *alpha*,
- et *fn* une fonction : $alpha * beta \rightarrow beta$;
- on applique *fn* sur chaque élément de la liste *L1* (en démarrant avec un élément de base de type *beta*),
- pour obtenir un élément de type *beta*.

```
;;; reduce: (alpha * beta -> beta)
;;;         * beta * LISTE[alpha] -> beta
;;; (reduce fn base L) rend le résultat de
;;;       fn(e1, fn(e2, ... fn(en, base) ...))
```

La fonctionnelle `reduce`

Passer en paramètres : l'opérateur binaire et l'élément de base

```
;;; reduce: (alpha * beta -> beta)
;;;         * beta * LISTE[alpha] -> beta
;;; (reduce fn base L) rend
;;;   fn(e1, fn(e2, ... fn(en, base) ...))
(define (reduce fn base L)
  (if (pair? L)
      (fn (car L)
          (reduce fn base (cdr L)))
      base ) )
```

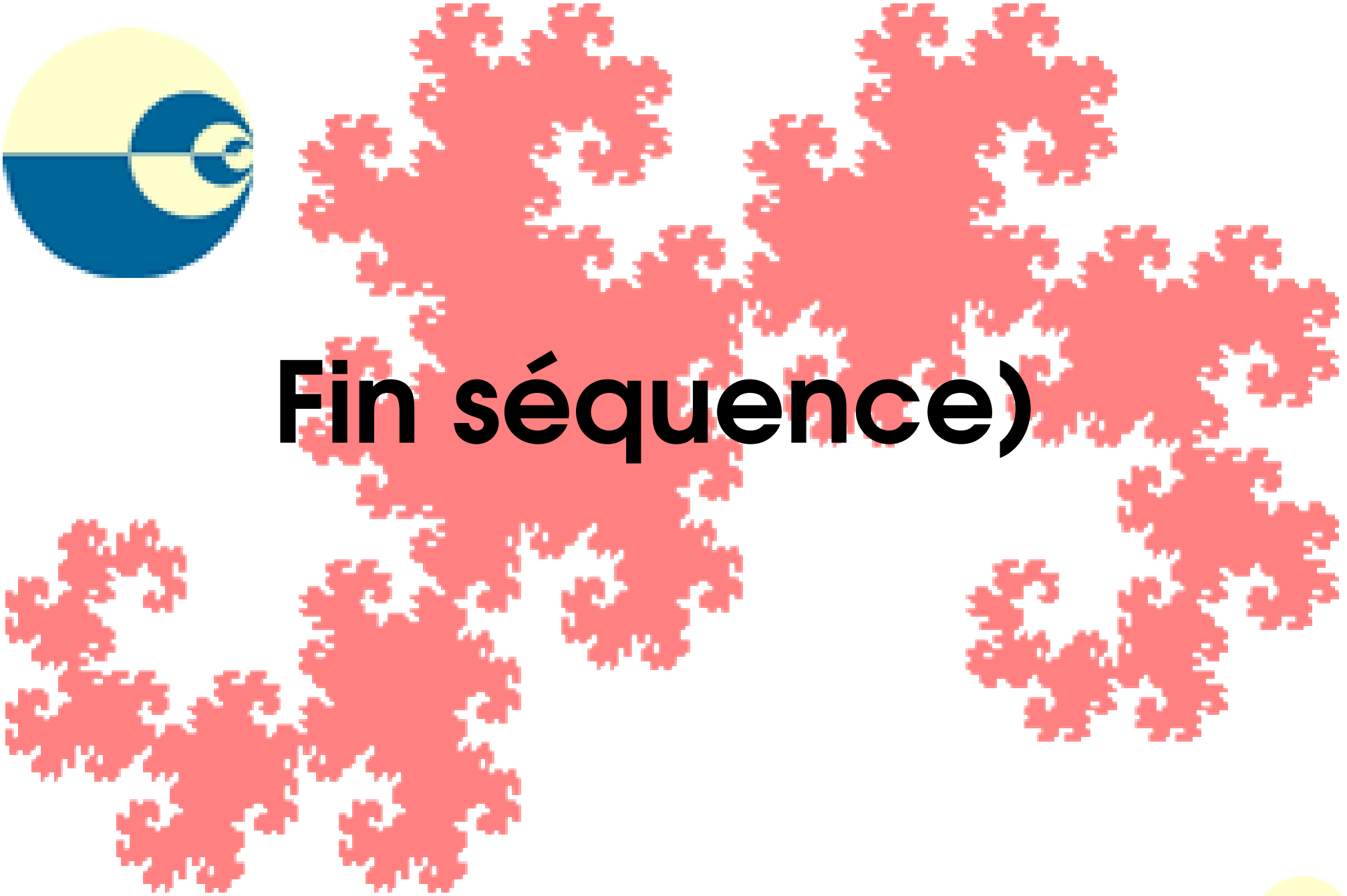
Applications `reduce`

```
(somme (list 1 2 3 4 5)) ≡
  (reduce + 0 (list 1 2 3 4 5))   → 15
  (+ 1 (+ 2 (+ 3 (+ 4 (+ 5 0)))))

(factorielle 5)  ≡
  (reduce * 1 (list 1 2 3 4 5))   → 120
  (* 1 (* 2 (* 3 (* 4 (* 5 1)))))

(somme-carres (list 1 2 3 4 5)) ≡
  (reduce + 0 (map carre (list 1 2 3 4 5))) → 55
  (+ (carre 1) (+ (carre 2) ... (+ (carre 5) 0) ...))
```

Associativité et `reduce`

Attention à l'ordre des opérations : $a - (b - c)$ n'est pas $(a - b) - c$!

```
(reduce - 0 (list 30 20 10 5))      →15

(reduce - 0 (list 30 20 10 5))      ≡
  (- 30 (- 20 (- 10 (- 5 0))))
```



Fin séquence)