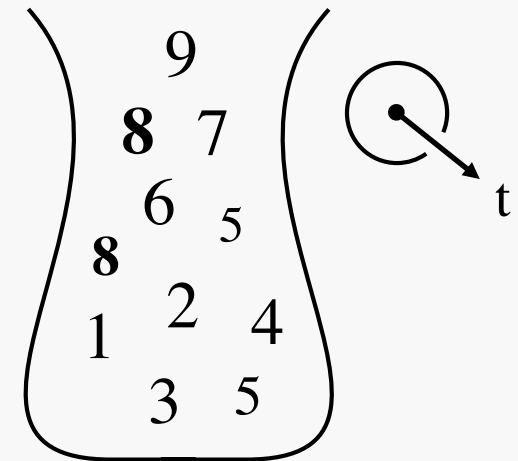


Couches logicielles
Architecture
Micro-architecture
Logique/Arithmétique
Circuit logique
Circuit analogique
Dispositif
Physique



# ACCÉLÉRATION DU CALCUL

ES102 / CM8



# CRITÈRES DE PERFORMANCE

- Encombrement (A)
- Energie (E)
- Temps (T)

- en ES102 :

- A mesuré

- par le nombre de transistors mis en jeu (silicon Area)

- E : à peine mentionné

- dépensée en cycles de charge/décharge de capacités  $\rightarrow CV^2$

- T examiné à bas niveau :

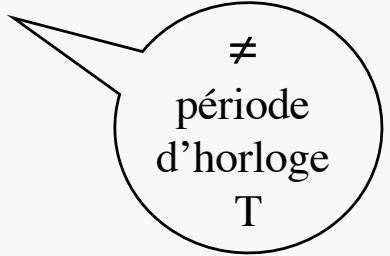
- délais combinatoires = temps de (dé)chargement de capacités MOS (en entrée de portes) par des courants de drain (en sortie de portes)  $\rightarrow PC5$

- minimisés par le choix de formules structurelles de type FDM en logique CMOS  $\rightarrow CM4$

- les délais s'ajoutent par mise en série de portes

- notion de chemin critique (chemin le plus long en temps)

$\rightarrow$  nouvelles perspectives désormais, architecturales et séquentielles

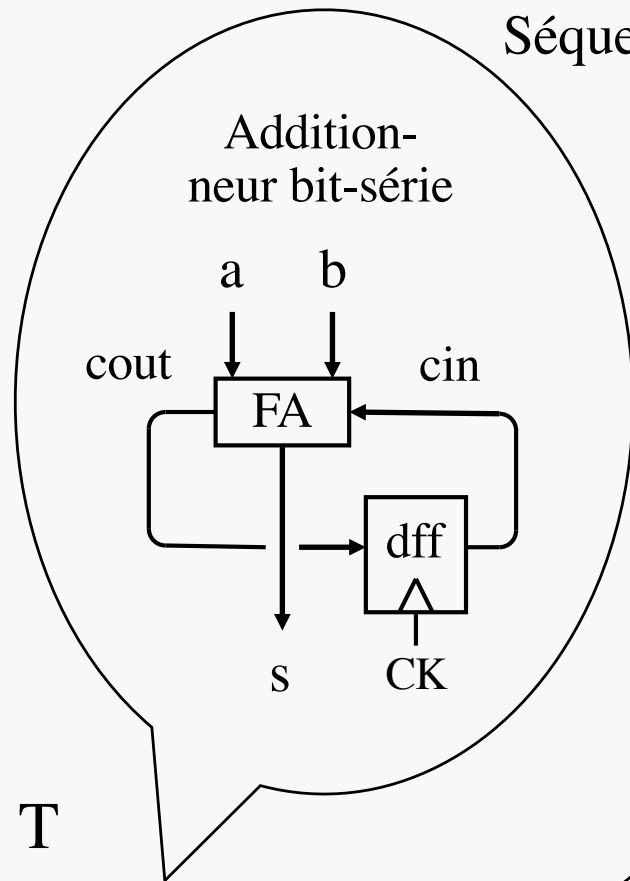


≠  
période  
d'horloge  
T



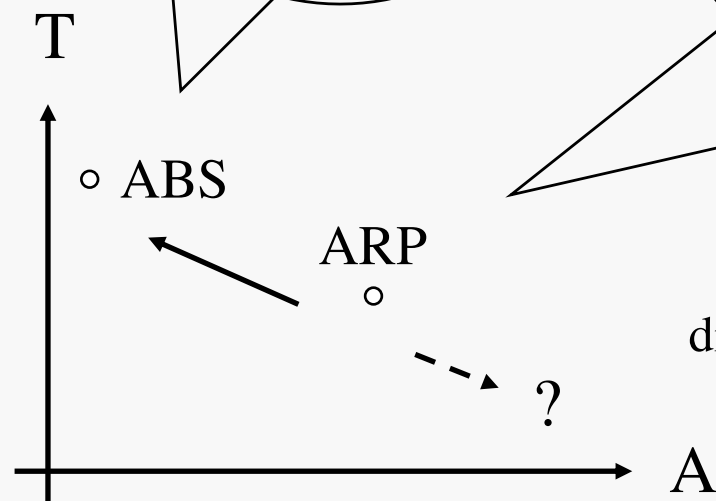
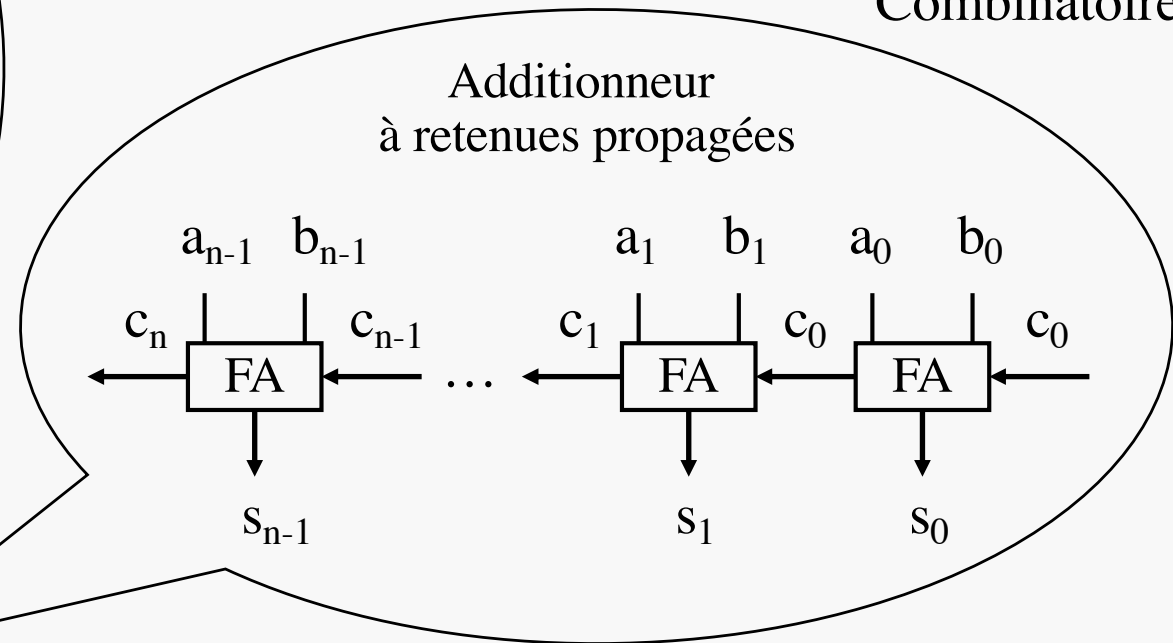
# T VERSUS A

Séquentiel



Motivation initiale  
pour la séquentialisation :  
réduire A, quitte à augmenter T

Combinatoire



Motivation séquentielle plus récente :  
disposer d'un jeu polyvalent de primitives combinables,  
pour pouvoir exécuter des algorithmes variés

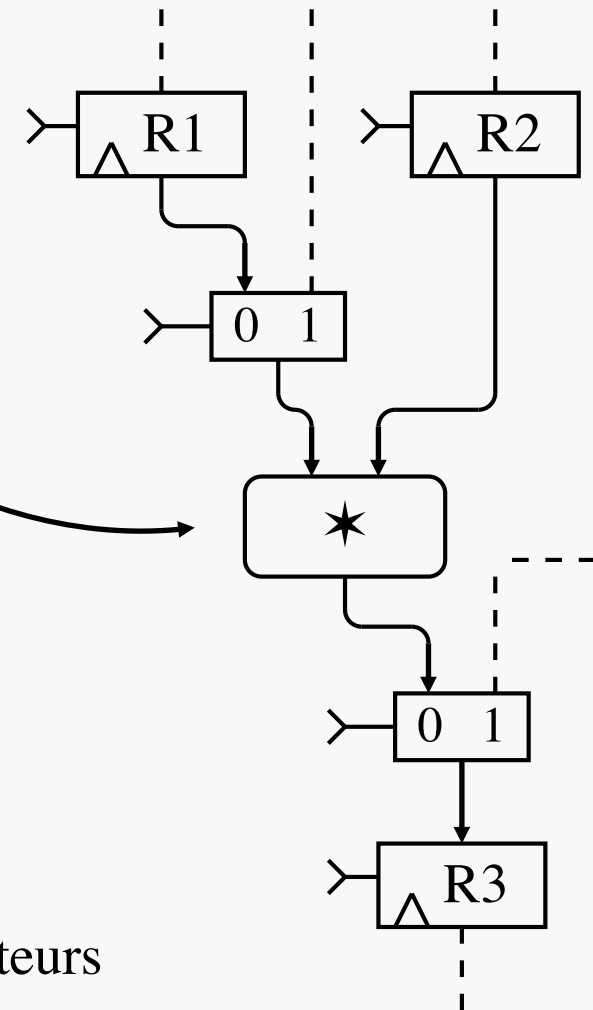
Recherche de puissance de calcul désormais...

# ACCÉLÉRATION DU CALCUL

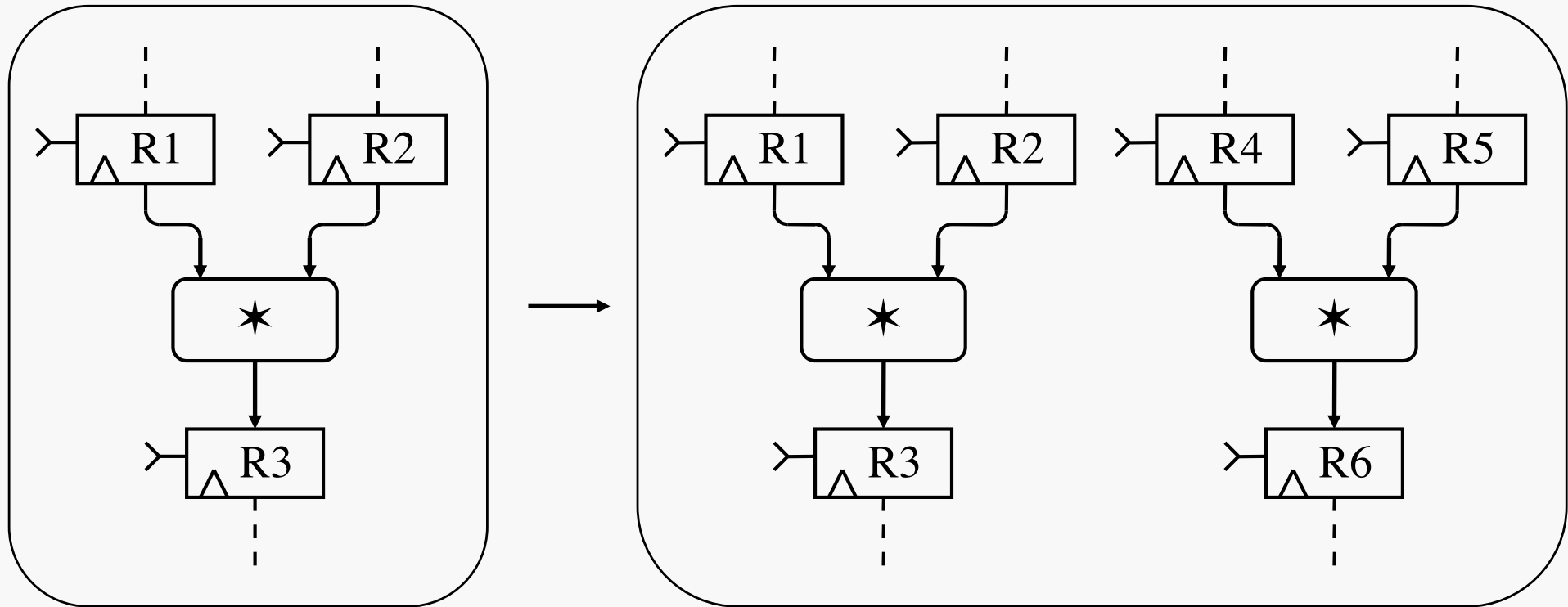
- Chemin de données (CD)
  - notre nouveau standard, séquentiel, de calcul numérique
  - $f_{CK}$  max. déterminée par le maillon le plus lent
- Que faire si  $\star$  trop lent ou surutilisé ?  
 Introduire du *parallélisme* sur  $\star$   
 $\Rightarrow$  augmenter  $A_\star$  pour diminuer  $T_\star$

## → Programme de la séance :

- techniques architecturales générales
  - du(/multi)plication d'unités fonctionnelles
  - pipeline : cadence accrue par travail à la chaîne
- techniques spécifiques : arithmétique des ordinateurs
  - accélération combinatoire des opérations les plus courantes, addition et multiplication
    - par des décompositions fonctionnelles astucieuses



# DU(/MULTI)PLICATION D'UNITÉS

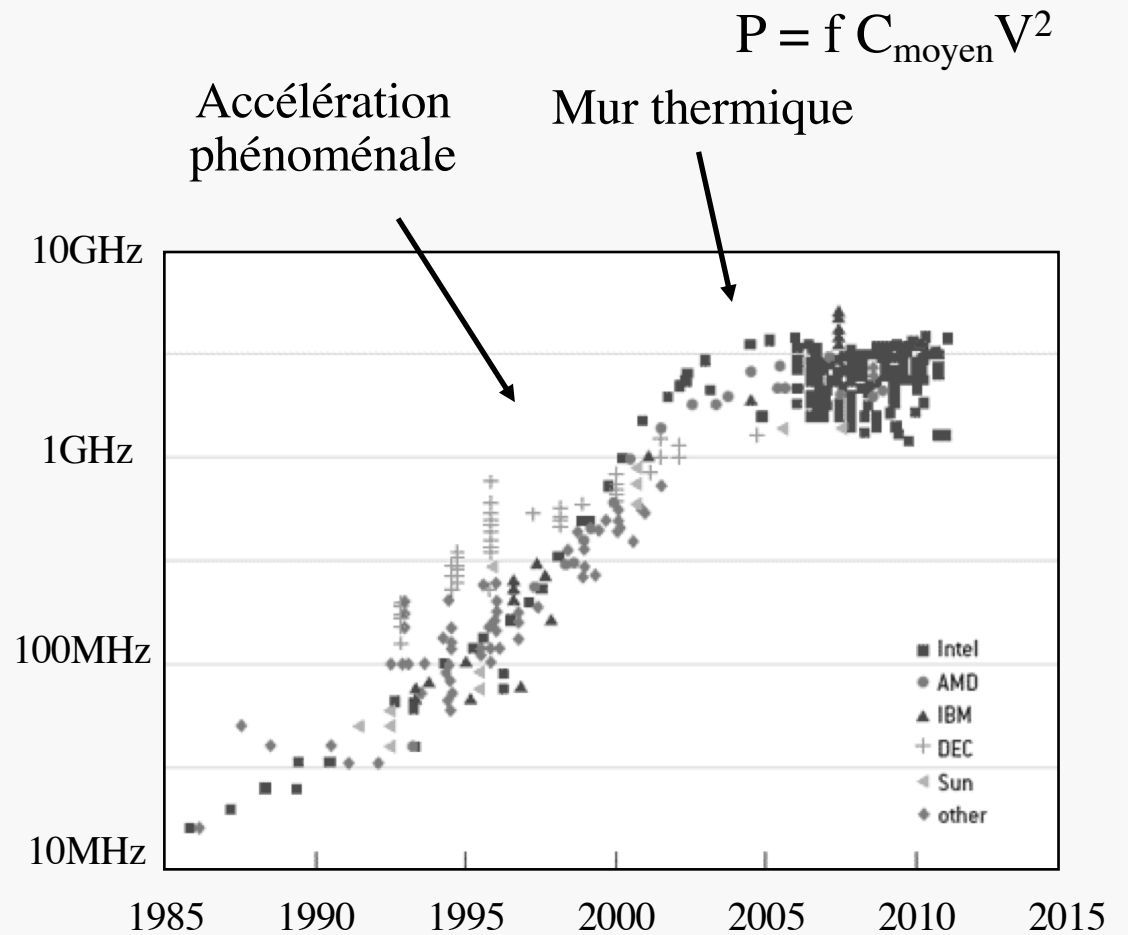


- Suppose des opérations  $\star$  indépendantes les unes des autres
  - pour pouvoir être exécutées simultanément
  - suppose aussi des moyens de distribution du travail
- Multiples déclinaisons : multi-cœur, superscalaire, VLIW, ...
- Exemples :  $\geq 2$  voies entières au sein d'un même cœur ARM ( $\rightarrow$  smartphones)  
5376 cœurs sur Nvidia GV100 ( $\rightarrow$  supercalculateur & GPU)

# PIPELINE, ALIAS TRAVAIL À LA CHAÎNE



*Les temps modernes 1936*



Fréquence d'horloge de différents microprocesseurs au cours du temps

alias travail  
à la chaîne

On suppose  
la fonction  $\star$   
décomposable  
en  $\nwarrow$  et  $\nearrow$  :

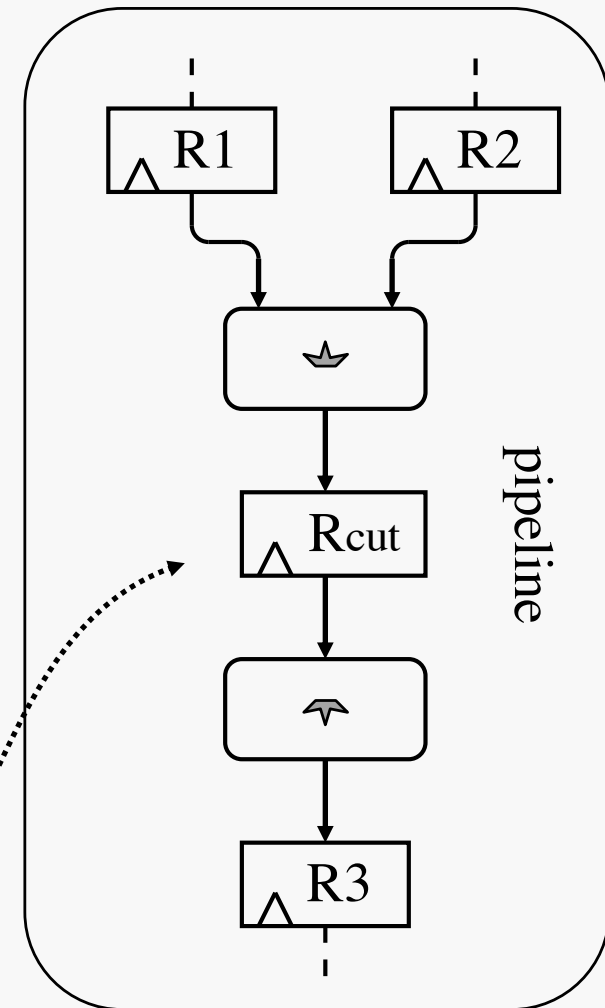
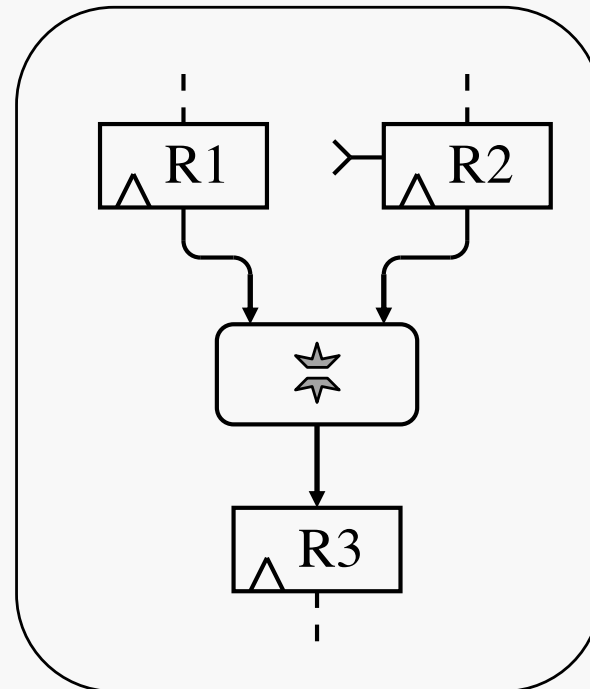
$$\star = \nearrow \circ \nwarrow$$

Idéalement :

$$\tau_{\nwarrow} \approx \frac{1}{2} \tau_{\star}$$

$$\tau_{\nearrow} \approx \frac{1}{2} \tau_{\star}$$

## PIPELINE



- Bascule D multi-bit insérée pour couper  $\star$  en 2
  - sa largeur : le nombre de fils qui relient  $\nwarrow$  à  $\nearrow$
- Possible alors de doubler quasiment la fréquence d'horloge maximale, donc la puissance de calcul
  - attention : à un instant donné, il y a 2 calculs successifs en cours dans le pipeline → PC8/Exo2
  - la durée d'un calcul  $\star$  complet, appelée *latence*, demeure quasi-identique
- Technique remarquable pour accélérer les maillons faibles
  - poussée à ses limites par Intel *circa* 2005 (record : Prescott avec 31 étages, cadencé à 3,6GHz)



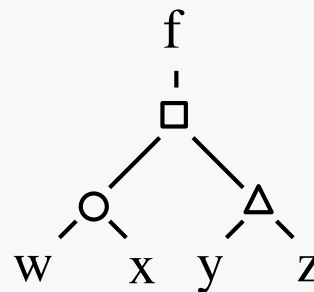
# PARALLÉLISME COMBINATOIRE

- Techniques spécifiques d'accélération pour opérations courantes
  - principe : rendre les ressources utilisées simultanément productives  $\Rightarrow T \searrow$
  - idéalement sans trop en augmenter le nombre :  $A \rightarrow$

- mais limité par les dépendances

- addition :  $c_{i+1}$  dépend de  $c_i$ 
  - comment paralléliser dans ces conditions ?
- fonction parité  $p$  : facile car  $\oplus$  associatif
  - structure idéale : l'arbre équilibré
  - $\rightarrow$  temps de calcul logarithmique
- fonction booléenne  $f$  quelconque :

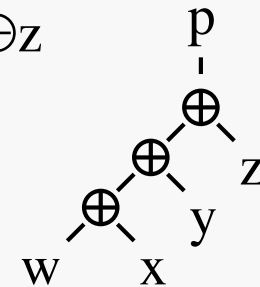
- de  $n$  variables (effectives)
- Théorème de Winograd* : impossible de calculer  $f$  plus vite qu'en  $\Theta(\log(n))$
- $\rightarrow$  borne inférieure logarithmique



$$[(w \oplus x) \oplus y] \oplus z$$

$$T = \Theta(n)$$

$$A = \Theta(n)$$

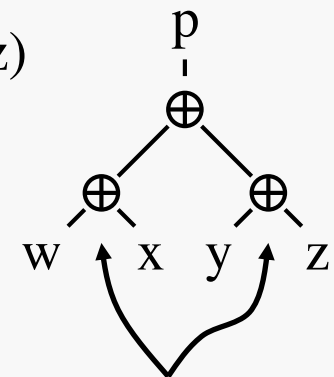


$t \uparrow$

$$(w \oplus x) \oplus (y \oplus z)$$

$$T = \Theta(\log n)$$

$$A = \Theta(n)$$

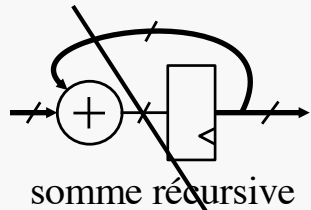


$t \uparrow$

calculs indépendants,  
donc exécutables  
simultanément

Pour une fonction à 4 variables, avec des portes à 2 entrées, on ne peut espérer plus rapide qu'un arbre de profondeur 2

# INITIATION AU CALCUL « PRÉFIXE » ET À SA PARALLÉLISATION



# calcul de serie

suite = range(16) # exemple

serie = [] # liste vide

cumul = 0

for valeur in suite:

cumul = cumul + valeur

serie.append(cumul)

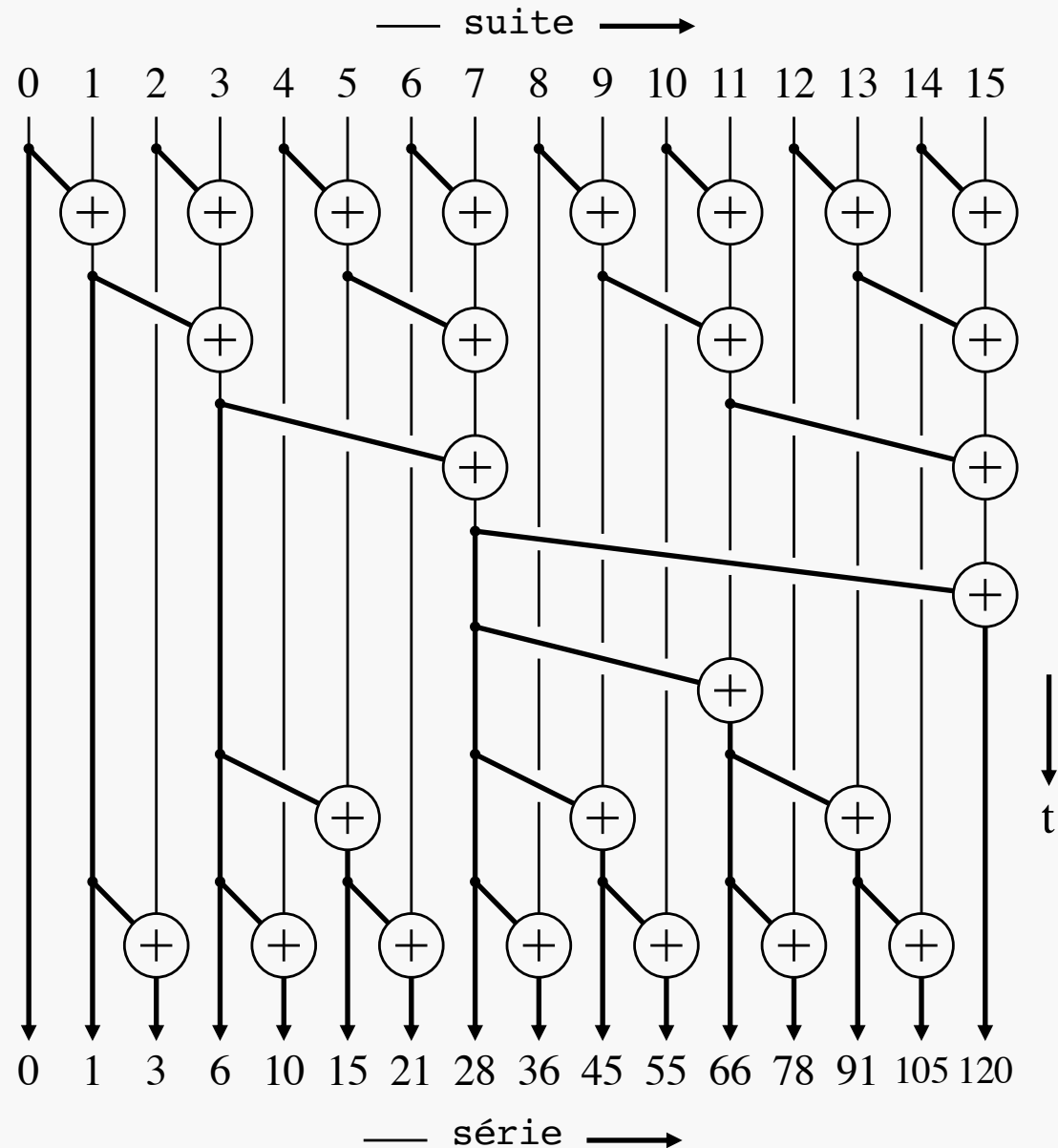
print(suite)

print(serie)

En Informatique, la transformation de la liste `signal` en la liste `serie` est appelée *somme préfixe* (ou addition préfixe)

La structure ci-contre *parallélise* l'opérateur *somme-préfixe*, en temps logarithmique

Exemple illustratif seulement. Exploitation du principe au niveau binaire ci-après ↙



# DÉCODAGE GRAY = XOR PRÉFIXE

L'organisation précédente peut être appliquée à tout opérateur associatif, tel le OU exclusif

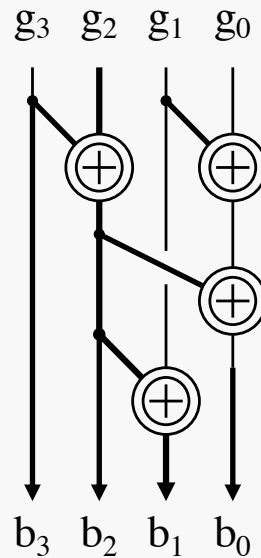
→ XOR préfixe

$$b_3 = g_3$$

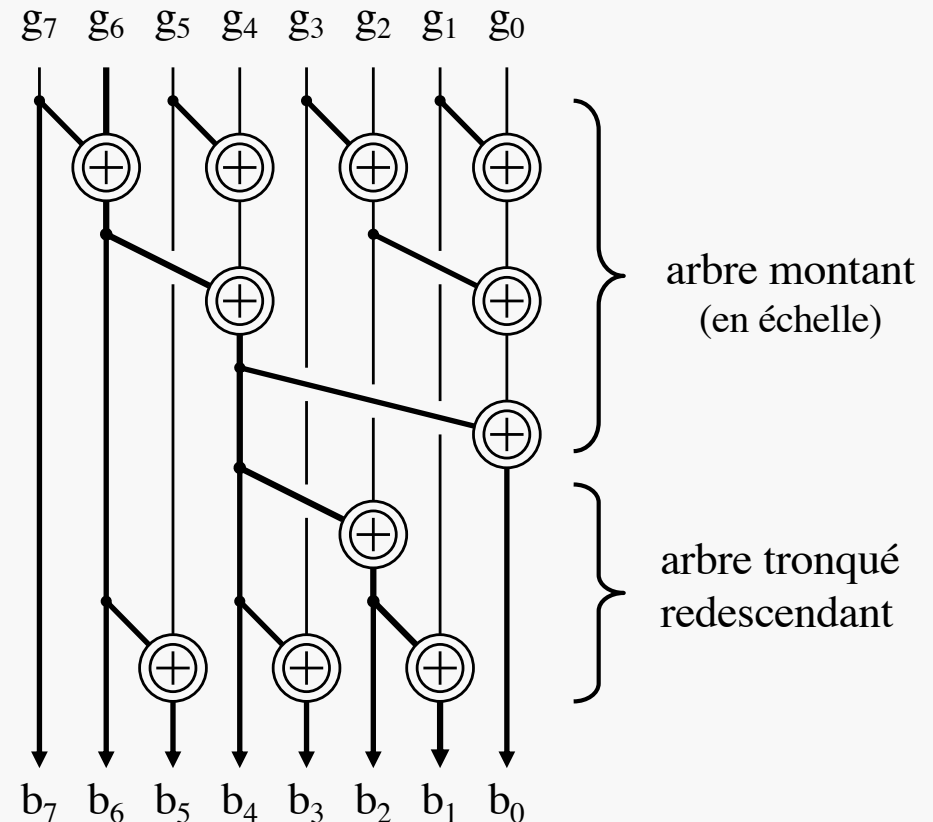
$$b_2 = g_3 \oplus g_2$$

$$b_1 = g_3 \oplus g_2 \oplus g_1$$

$$b_0 = g_3 \oplus g_2 \oplus g_1 \oplus g_0$$



n=4  
PC1/Exo3



n=8

Longueur du **chemin critique** =  $2[\log_2(n)-1] \Rightarrow T = \Theta(\log(n))$

Coût matériel :  $[2(n-1) - \log_2(n)]$  nœuds  $\Rightarrow A = \Theta(n)$

calcul combinatoire  
parallélisé en *temps*  
*logarithmique* alors que  
solution itérative en  
*temps linéaire*

# ANTICIPER LE CALCUL DES RETENUES...

$$a_i \oplus b_i$$

$$a_i + b_i$$

$$a_i \cdot b_i$$

*p pour propagation*  
*g pour génération*

$$c_1 = p_0 \cdot c_0 + g_0$$

$$c_2 = p_1 c_1 + g_1 = p_1 p_0 \cdot c_0 + (p_1 g_0 + g_1)$$

$$c_{i+1} = p_i \cdot c_i + g_i$$

relation  
pseudo-affine  
entre  $c_{i+1}$  et  $c_i$

Soit  $\alpha : \mathbb{B}^2 \times \mathbb{B}^2 \rightarrow \mathbb{B}^2$  t.q.

$$(p_1, g_1) \alpha (p_0, g_0) = (p_1 p_0, p_1 g_0 + g_1)$$

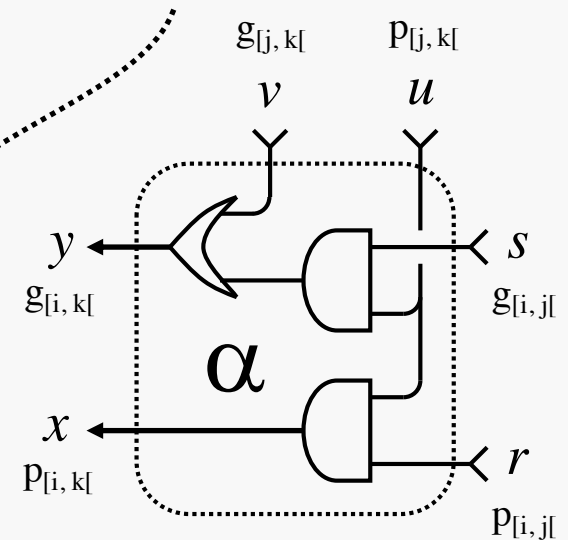
$$\begin{matrix} u & v & r & s & x & y \end{matrix}$$

$$(0, c_1) = (p_0, g_0) \alpha (0, c_0)$$

On vérifie

$$(0, c_2) = (p_1, g_1) \alpha [(p_0, g_0) \alpha (0, c_0)]$$

$$= [(p_1, g_1) \alpha (p_0, g_0)] \alpha (0, c_0)$$



$\alpha$  associatif ? Oui !  $\Rightarrow$  Parenthèses inutiles ! – Finalement, on passe ainsi de 0 à  $i+1$  :

$$\rightarrow (0, c_{i+1}) = (p_i, g_i) \alpha (p_{i-1}, g_{i-1}) \alpha \cdots \alpha (p_1, g_1) \alpha (p_0, g_0) \alpha (0, c_0)$$

$(p_{[0, i]}, g_{[0, i]})$  : pour l'intervalle  $[0, i]$

La suite des  $(p_{[0, i]}, g_{[0, i]})$  est l' $\alpha$ -préfixe de la suite des  $(p_i, g_i)$ .

Une fois cette suite calculée, tous les  $c_i$  et  $s_i$  s'obtiennent en temps constant.

$$c_0 = 0 \Rightarrow c_{i+1} = g_{[0, i]}$$

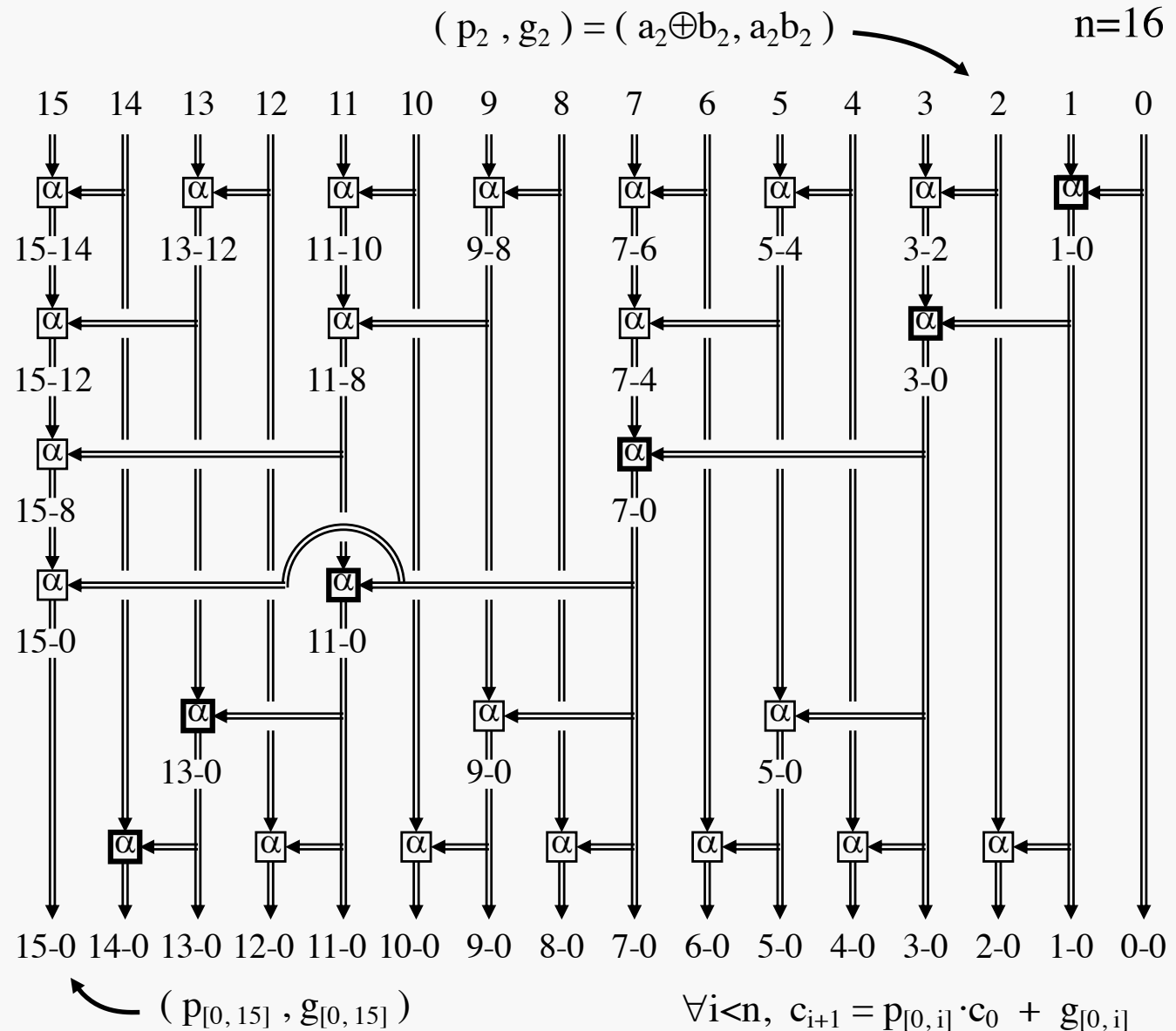
# $\alpha$ -PRÉFIXE $\rightarrow$ ADDITIONNEUR À RETENUES ANTICIPÉES

Le graphe ci-contre :

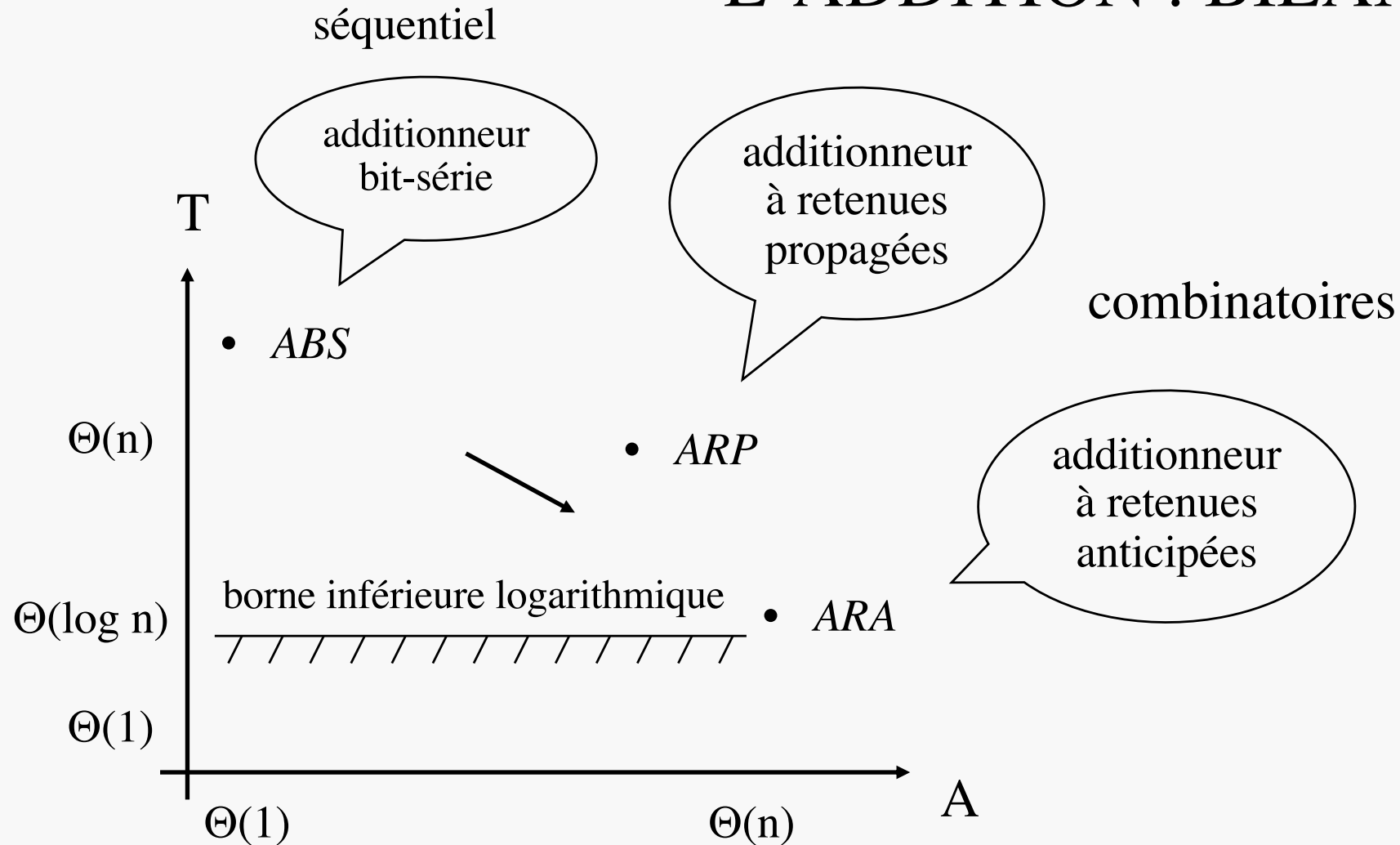
- calcule l' $\alpha$  préfixe en  $T=\Theta(\log(n))$ .
- permet l'*addition en temps logarithmique* !  
 $\approx$  borne de Winograd
- en *anticipant* le calcul des retenues
- pour un coût matériel linéaire (certes 2-3 fois plus cher que l'ARP)

$\rightarrow$  Additionneur dit de Brent et Kung

Doubles traits partout car couples  $(p, g)$



# ACCÉLÉRATION DE L'ADDITION : BILAN



# REPRÉSENTATION REDONDANTE DES NOMBRES

$$15 \times 19 = 15 \times \underbrace{21}_{20-1} = 300 - 15 = 285$$

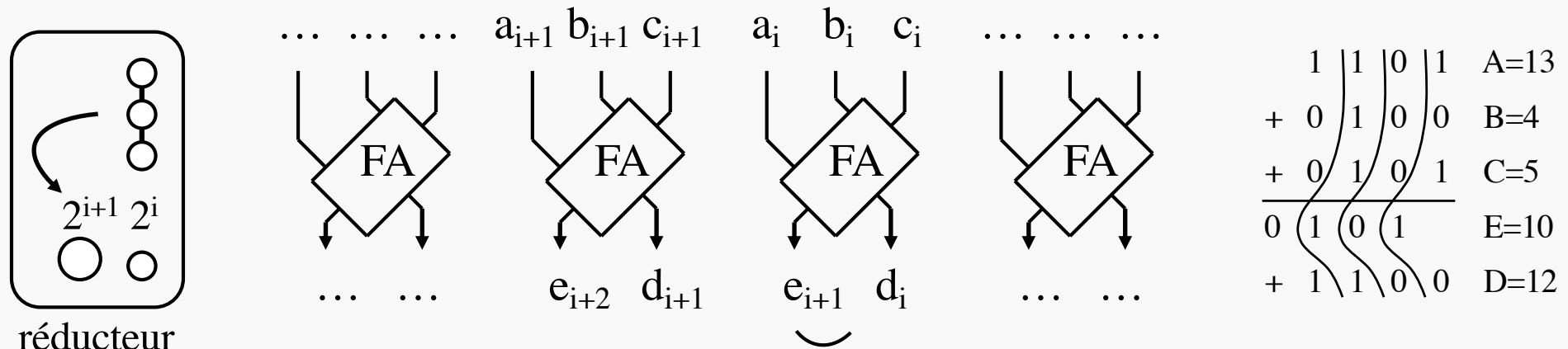
Comptant en base 10, nous utilisons parfois aussi -1 comme chiffre...

Utiliser plus de chiffres que la base permet d'avoir plusieurs représentations possibles pour un même nombre, et de choisir la plus commode pour un calcul donné

→ Notations dites *redondantes*, ou *molles*

*Planche suivante encore plus molle ; -)*

# ADDITIONNEUR À RETENUES CONSERVÉES



*L'ARC transforme 3 nombres en 2,  
à somme constante*

En considérant que  $A+B$  et  $E+D$  représentent chacun un entier (notation très *molle*), on dispose alors d'un additionneur en temps constant :

$$\begin{array}{l}
 \text{nombre classique} \\
 + \text{ nombre mou} \\
 \hline
 = \text{nombre mou}
 \end{array}$$

$$\begin{array}{l}
 A = \Theta(n) \\
 T = \Theta(1) !
 \end{array}$$

$$\begin{aligned}
 \forall i, & \ll a_i + b_i + c_i = d_i + 2e_{i+1} \gg \\
 \Rightarrow & \sum a_i \cdot 2^i + \sum b_i \cdot 2^i + \sum c_i \cdot 2^i \\
 & = \sum d_i \cdot 2^i + 2 \cdot \sum e_{i+1} \cdot 2^i \\
 \Rightarrow & A + B + C = D + E \\
 \Rightarrow & (A + B) + C = (D + E)
 \end{aligned}$$

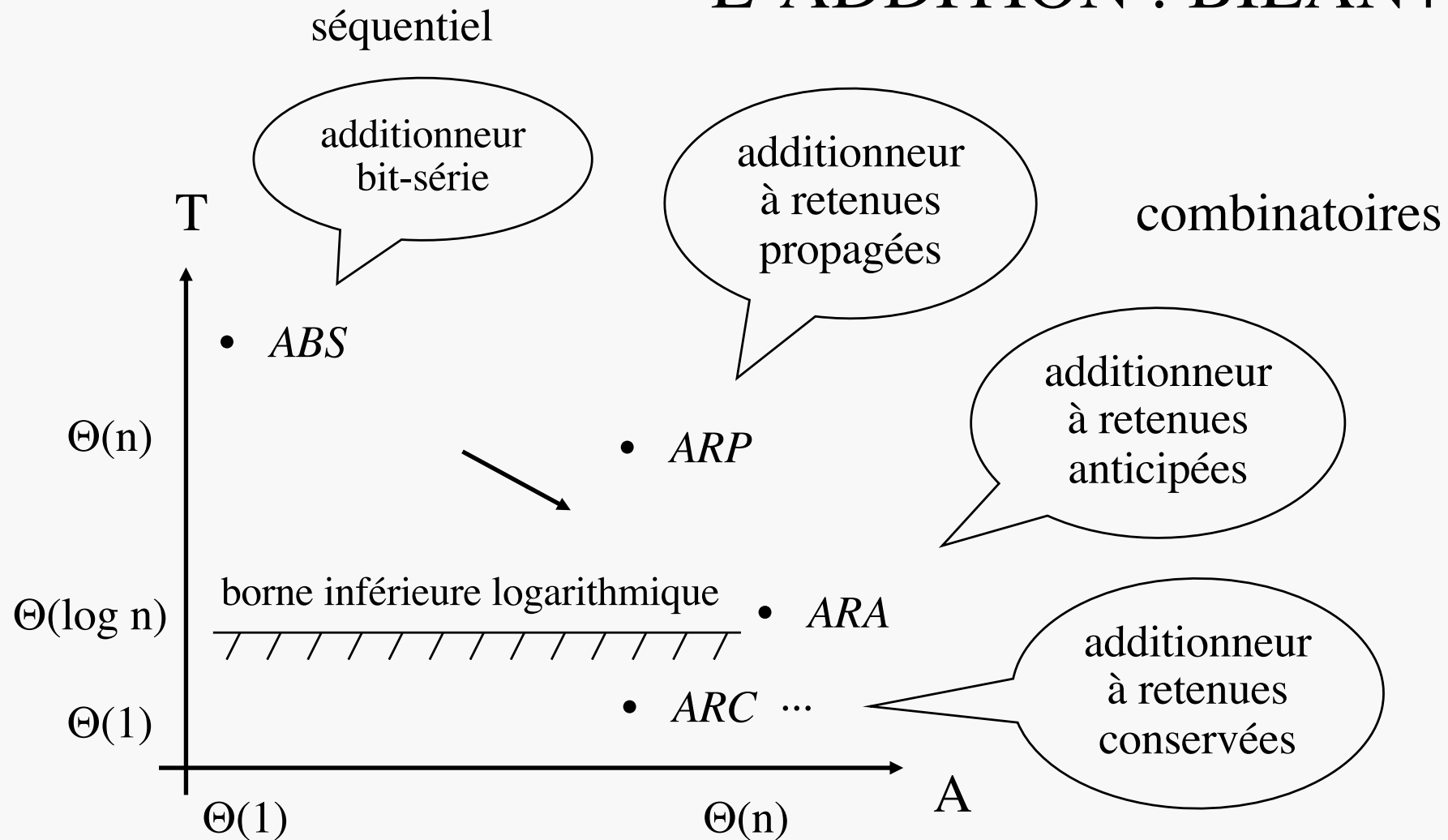
Utilisation spécifique

→ parfait pour une somme  
récursive ultra-rapide

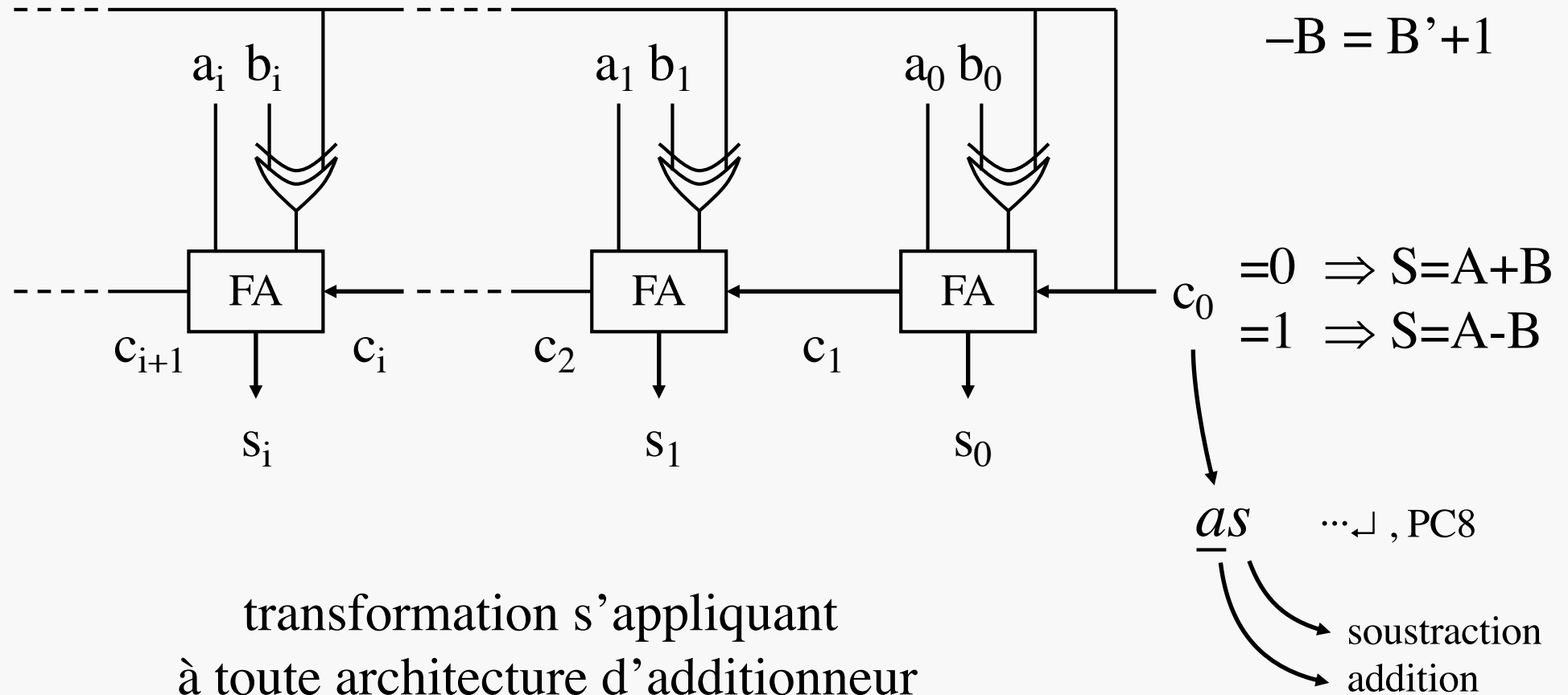
mais besoin de repasser en notation  
classique à un moment donné...



# ACCÉLÉRATION DE L'ADDITION : BILAN+



# ADDITIONNER OU SOUSTRAIRE, AU CHOIX



# ACCÉLÉRATION DE LA MULTIPLICATION

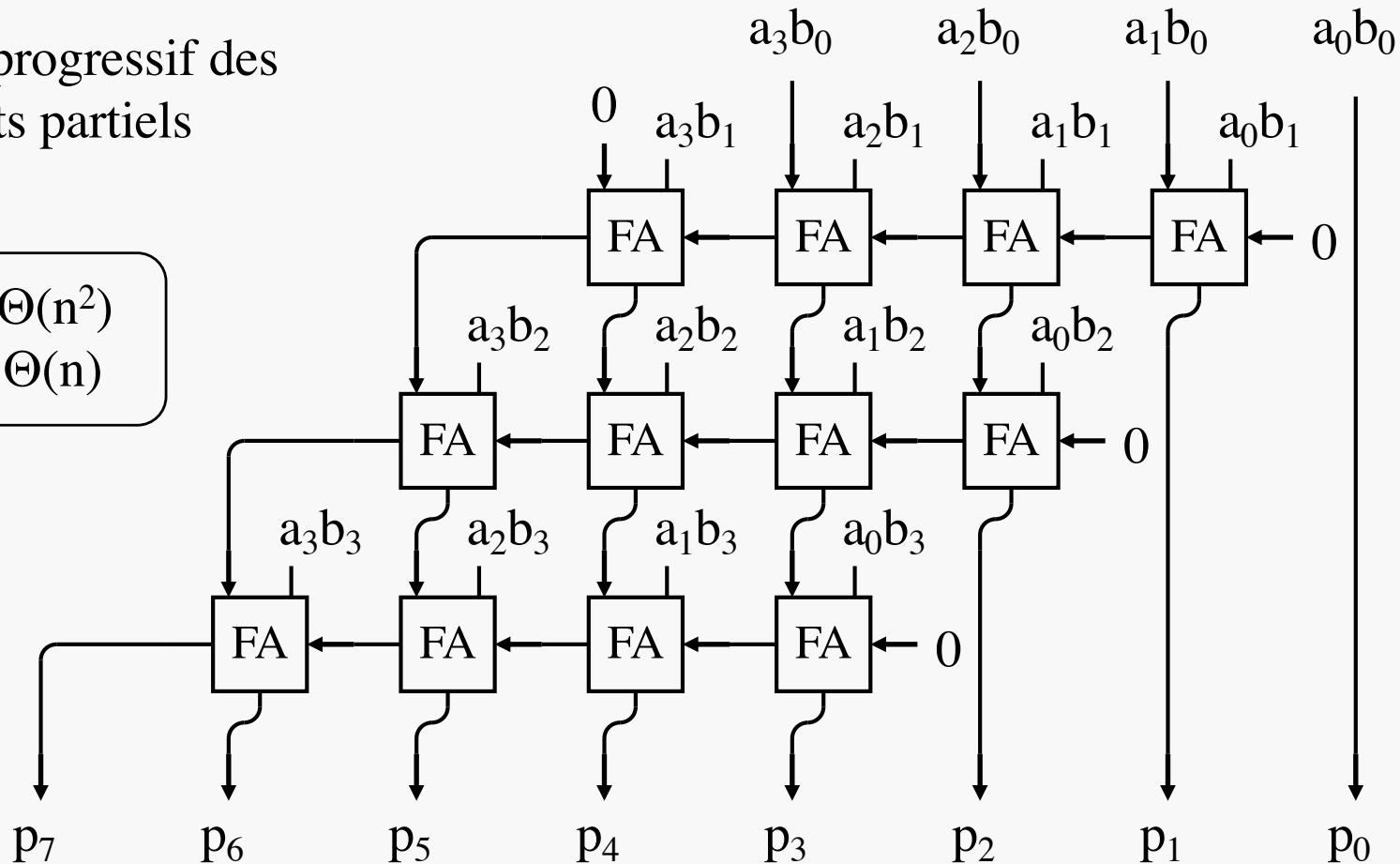


# MULTIPLIEUR À RETENUES PROPAGÉES

Ajout progressif des  
produits partiels

$$A = \Theta(n^2)$$

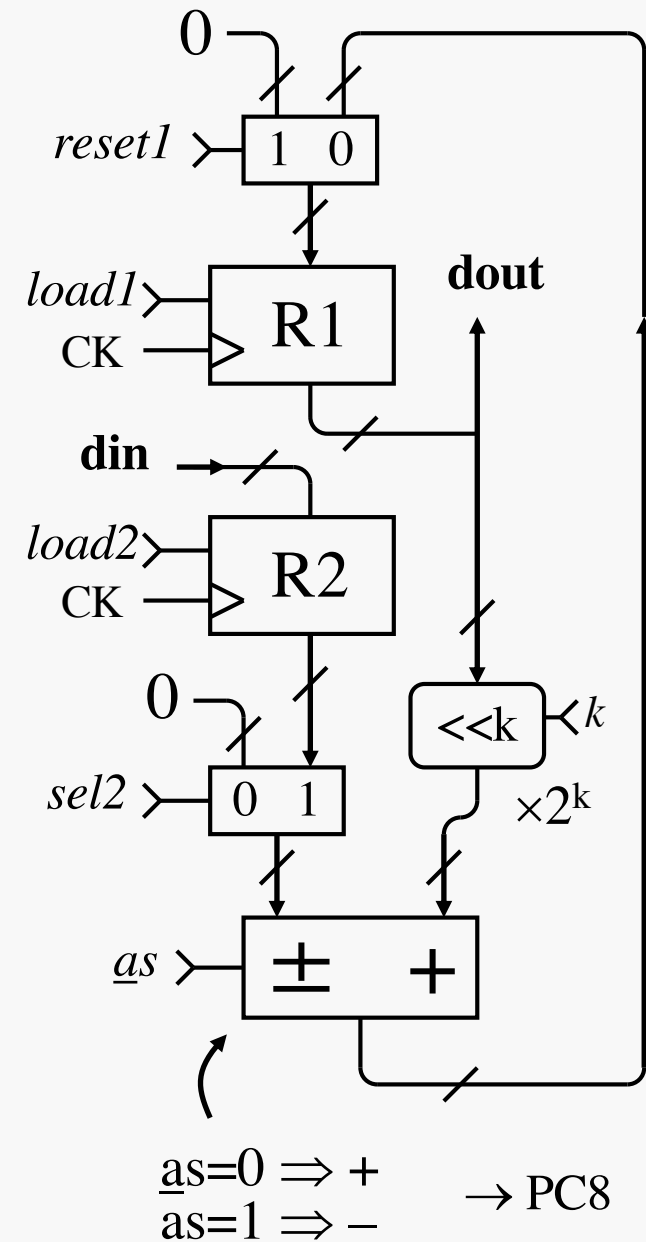
$$T = \Theta(n)$$



# RECODAGE DE BOOTH

→ PC9

- Pour processeur du pauvre
  - façon « calculatrice primitive »
  - mais avec décaleur à portée variable  $k$ 
    - pour sauter par-dessus les 0 du multiplieur (le facteur)
  - et additionneur-soustracteur...
- Idée : recoder le multiplieur en exploitant le *chiffre*  $-1$  (noté 1), pour faire apparaître encore plus de 0
  - appelée *recodage de Booth*
  - Règle 1 :  $01 - 11 \rightarrow 10 - 01$ 
    - ex. :  $01111 \rightarrow 10001$   $\left. \begin{array}{l} 8+4+2+1 \rightarrow 16-1 \end{array} \right\} \text{nécessite seulement une addition + une soustraction}$
    - bien, mais :  $0110111 \rightarrow 1011001$  (et un décalage de portée 4)
  - Règle 2 :  $11 \rightarrow 01$  (appliquée après la règle 1)
  - met un maximum de bits à 0, au moins la moitié  
⇒ gain d'un facteur 2 en temps sur le pire cas
- Exemple :  $01011100110111001111011$  normal  
 $10100101001001010000101$  Booth



# MULTIPLICATION EN TEMPS LOGARITHMIQUE

- FA : réduit 3 bits de même poids en 2, dont 1 deux fois plus lourd
  - Soit la suite  $u_{k+1} = \lfloor 3/2 \cdot u_k \rfloor$ 
    - avec  $u_0 = 2$  asymptotiquement géométrique
    - $(u_k) = (2, 3, 4, 6, 9, 13, 19, 28 \dots)$
  - Soit un « tas » de bits de hauteur  $n$ 
    - soit  $m$  le plus petit entier t.q.  $u_m \geq n$
    - moyennant quelques FAs travaillant en parallèle, on peut abaisser la hauteur du tas de  $u_m$  à  $u_{m-1}$  en un  $\tau_{FA}$ 
      - avec un FA par barreau ci-contre
        - le moins de barreaux possible
        - parfois seulement 2 bits sur un barreau
    - et ainsi de suite...
- $\Rightarrow$  hauteur 2 atteinte en  $\approx \log_{3/2}(n) \cdot \tau_{FA}$
- pour finir, addition en temps logarithmique

