SDR

Sondes et échos

Synchronisation

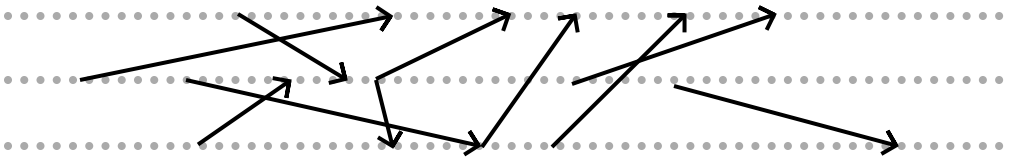
Olivier Lemer

↓ Algorithme "synchrone" ?

Définitions

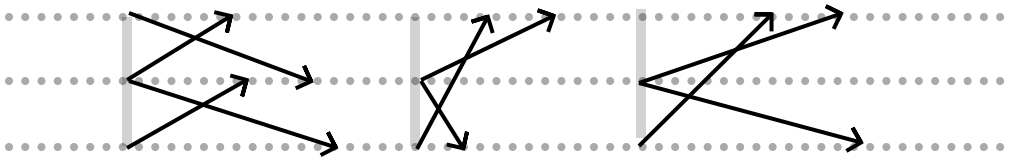
Jusqu'ici, tous nos algorithmes étaient **asynchrones**.

"Chaque processus peut envoyer ou recevoir un message à tout moment."



Il existe aussi des algorithmes **synchrones**.

"Tous les processus avancent à l'unisson."

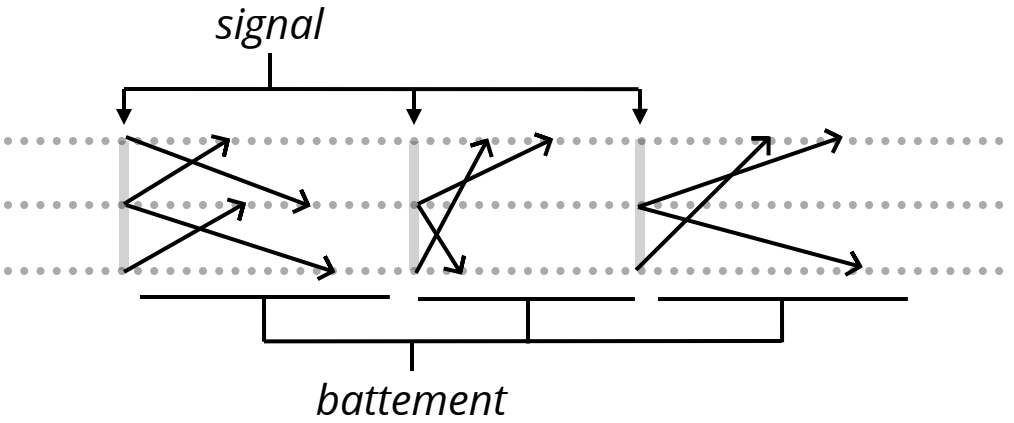


Attention : rien à voir avec un *système* synchrone

Requirements d'un algorithme synchrone

Un **signal** est reçu pour déclencher un nouveau **battement**.

(pulse)



Dans chaque battement, un processus peut

- Faire des calculs *en fonction des battements précédents*.
- Envoyer jusqu'à un message par voisin, *fonction des battements précédents*.
- Recevoir jusqu'à un message par voisin, *fonction des battements précédents.*

Un processus est **prêt** quand tous *ses* messages ont été reçus.

Remarques

Remarques, donc :

- Tout événement d'un battement k ne dépend que des battements < k.
- Il est possible de n'envoyer aucun message durant un battement.
- Il est *impossible* d'envoyer *plusieurs* messages au même voisin durant un même battement.

L'ordre des événements est évident :

E a eu lieu avant F

⇳

Le battement de E est plus ancien que celui de F.

C'est un **ordre partiel** : *deux événements du même battement n'ont pas d'ordre.*

Aujourd'hui

Algorithmes de synchronisation ***Baruch Awerbuch***

Synchroniseur alpha

Sur graphe arbitraire

Synchroniseur beta

Sur arbre

Synchroniseur alpha

Comment synchroniser des processus ?

Deux approches de synchronisation

Centralisée

*Un **contrôleur** est responsable d'envoyer les signaux à tous les processus au bon moment.*

Décentralisée

Les processus doivent communiquer pour savoir quand ils peuvent déclencher un signal localement.

Synchronisation décentralisée

Problèmes à résoudre

Comment savoir quand je suis prêt ?

(i.e. tous mes messages de ce battement ont été reçus)

Demander un ACK pour chaque message envoyé.

Quand tous mes messages sont acquittés, je suis prêt.

Comment savoir quand je peux lancer le battement suivant ?

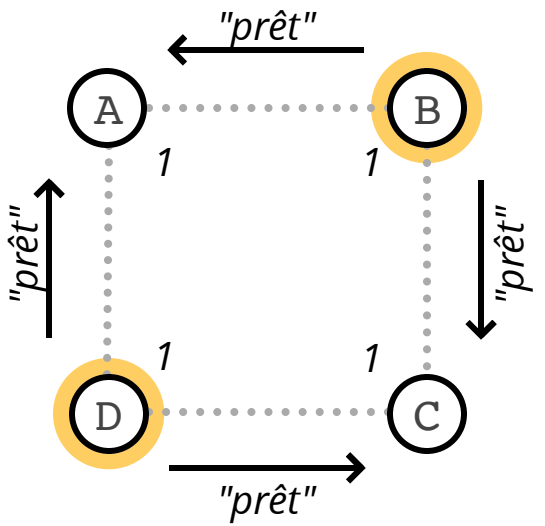
(i.e. le battement actuel est officiellement terminé)
(i.e. j'ai reçu tous les messages qui m'étaient destinés)

Informer mes voisins quand je suis prêt.

Quand tous mes voisins sont prêts, je peux lancer le battement suivant.

Synchronisation décentralisée

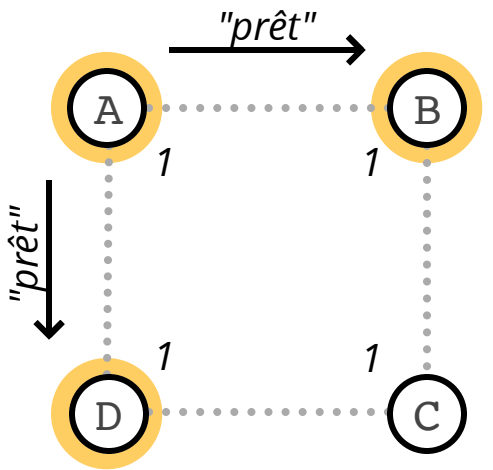
Autre problème ?



Prêt

Synchronisation décentralisée

Autre problème ?

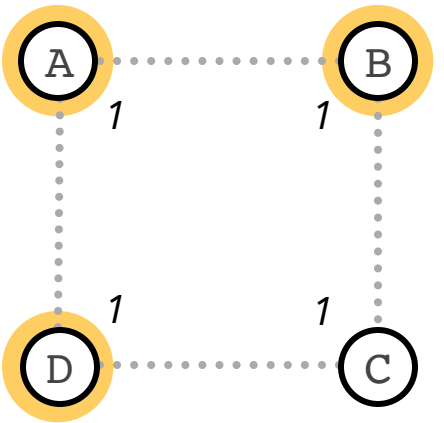


Prêt

Synchronisation décentralisée

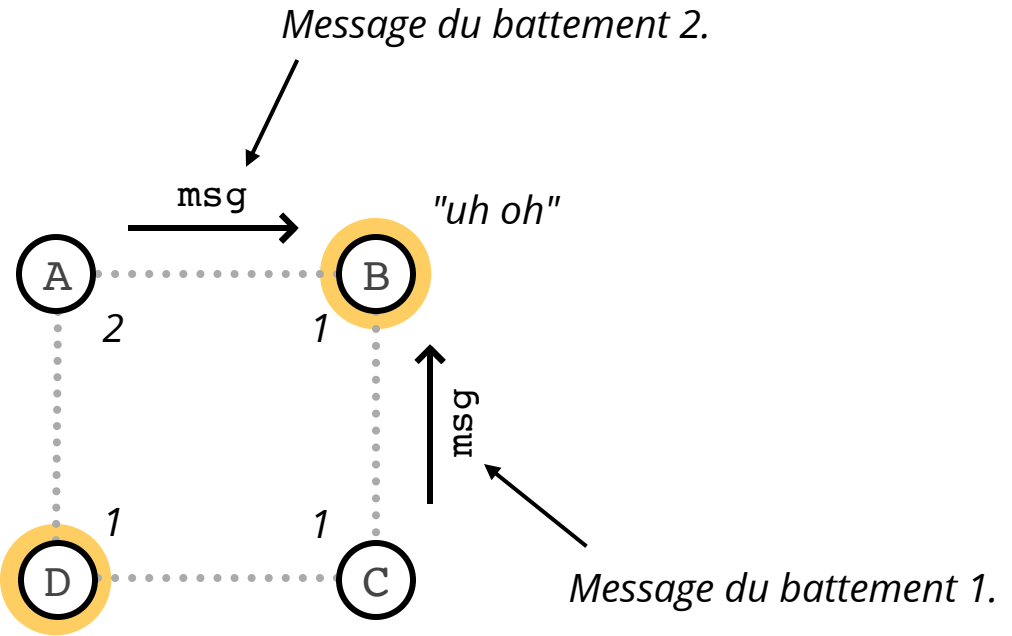
Autre problème ?

"Tous mes voisins sont prêts,
je passe au suivant."



Synchronisation décentralisée

Autre problème ?

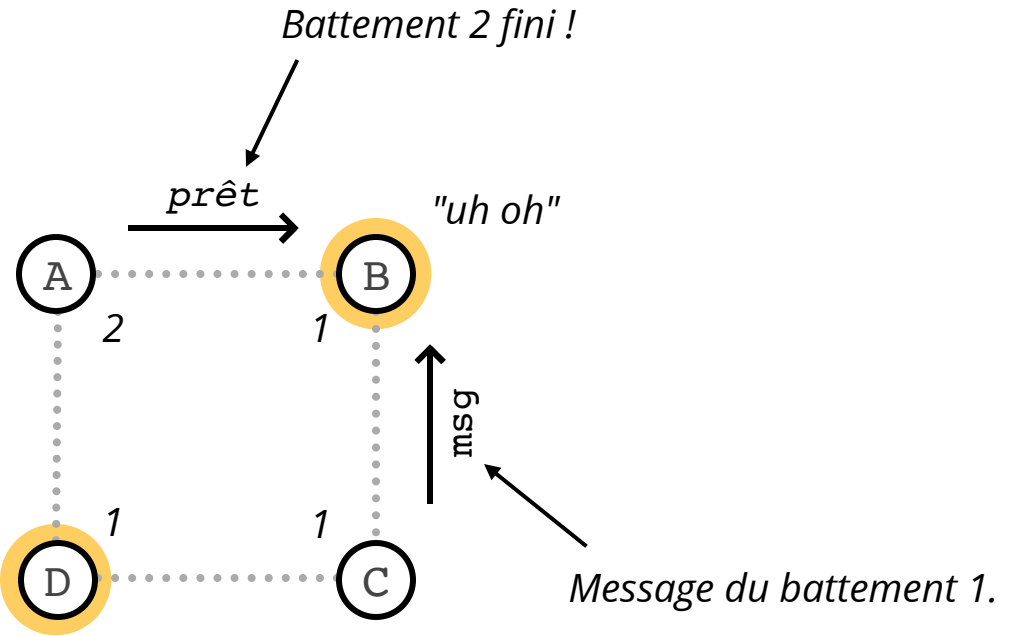


Que doit faire `B` ?

Si un processus me dit `prêt`, tous ses messages suivants appartiendront au prochain battement.
Si je ne suis pas encore au battement suivant, je les buffer et les traiterai au moment venu.

Synchronisation décentralisée

Autre problème ?



Que doit faire B ?

*Si un processus me dit `prêt`, **tous ses messages** suivants appartiendront au prochain battement.*
Si je ne suis pas encore au battement suivant, je les buffer et les traiterai au moment venu.

Synchroniseur alpha

Résumé et complexités

Résumé

Réception d'un message de `i`

Si j'ai noté que `i` est prêt
    Je mets le message de coté
Sinon
    Je stocke le message
    Je réponds `ACK`

Réception d'un `ACK`

Je note la réception du message

Un `ACK` reçu pour chaque envoi

J'envoie *prêt* à tous mes voisins

Besoin d'un numéro de battement sur les messages ?

Non ! Si j'ai reçu un prêt d'un voisin, je sais que ses prochains messages sont du battement suivant.

Réception d'un *prêt* de `i`

Si j'ai noté que `i` est prêt
    Je mets le message de coté
Sinon
    Je note que `i` est prêt
    Si mes voisins et moi sommes prêts
        Je lance un nouveau battement

Début de battement

Je réinitialise tous les statuts "prêt"
Pour tout message mis de coté
    Je traite sa réception
Je fais mes calculs
J'envoie mes messages

Complexité

Communication par battement *(en ignorant les messages de l'algorithme synchronisé)*

Sur chaque lien, on a deux `prêt` par battement, max.

Donc 2E.

Donc O(E).

Donc O(V^2). ← Pas génial…

Performances, en fonction de T

Séquence de 3 messages par battement, max.

Donc 3T.

Donc O(T).

Avec V = nombre de processus ; E = nombre de liens ; T = durée de transit maximale

Synchroniseur beta

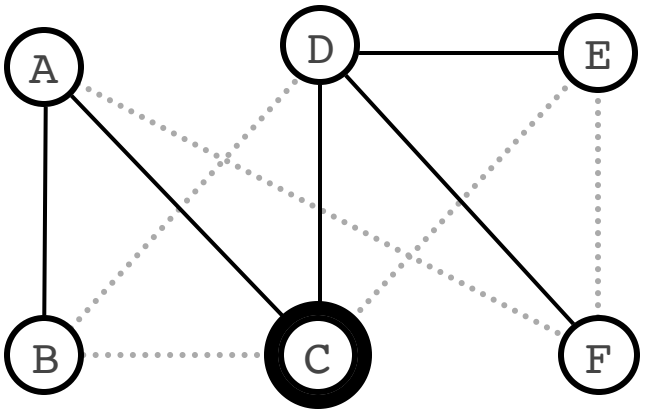
Une meilleure complexité…

Changement de structure

On suppose un arbre intégré sur le réseau.

(a.k.a. un "spanning tree")

On élit un **leader**, qui devient la racine de l'arbre.



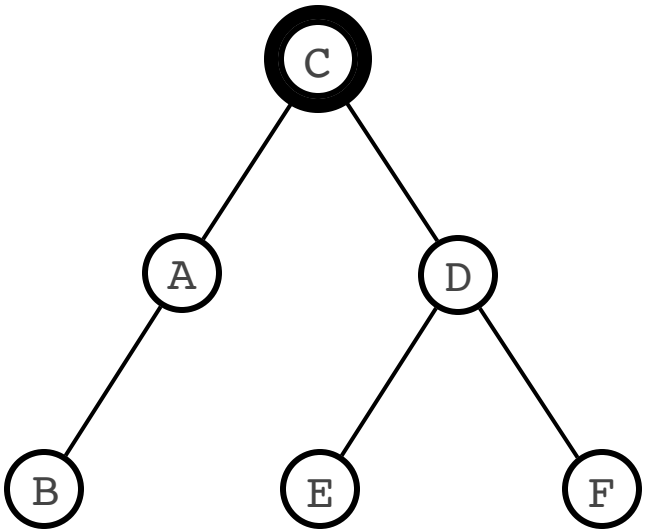
Changement de structure

On suppose un arbre intégré sur le réseau.

(a.k.a. un "spanning tree")

On élit un **leader**, qui devient la racine de l'arbre.

Chaque noeud **représente** lui *et ses enfants*.



Quand suis-je prêt ?

Tous mes messages *et ceux de mes enfants* ont été reçus.

Quand lancer le prochain battement ?

Quand la racine est prête.

(elle représente l'arbre entier ; si elle est prête, tout l'arbre l'est)

Comment lancer le prochain battement ?

La racine envoie le signal à travers l'arbre.

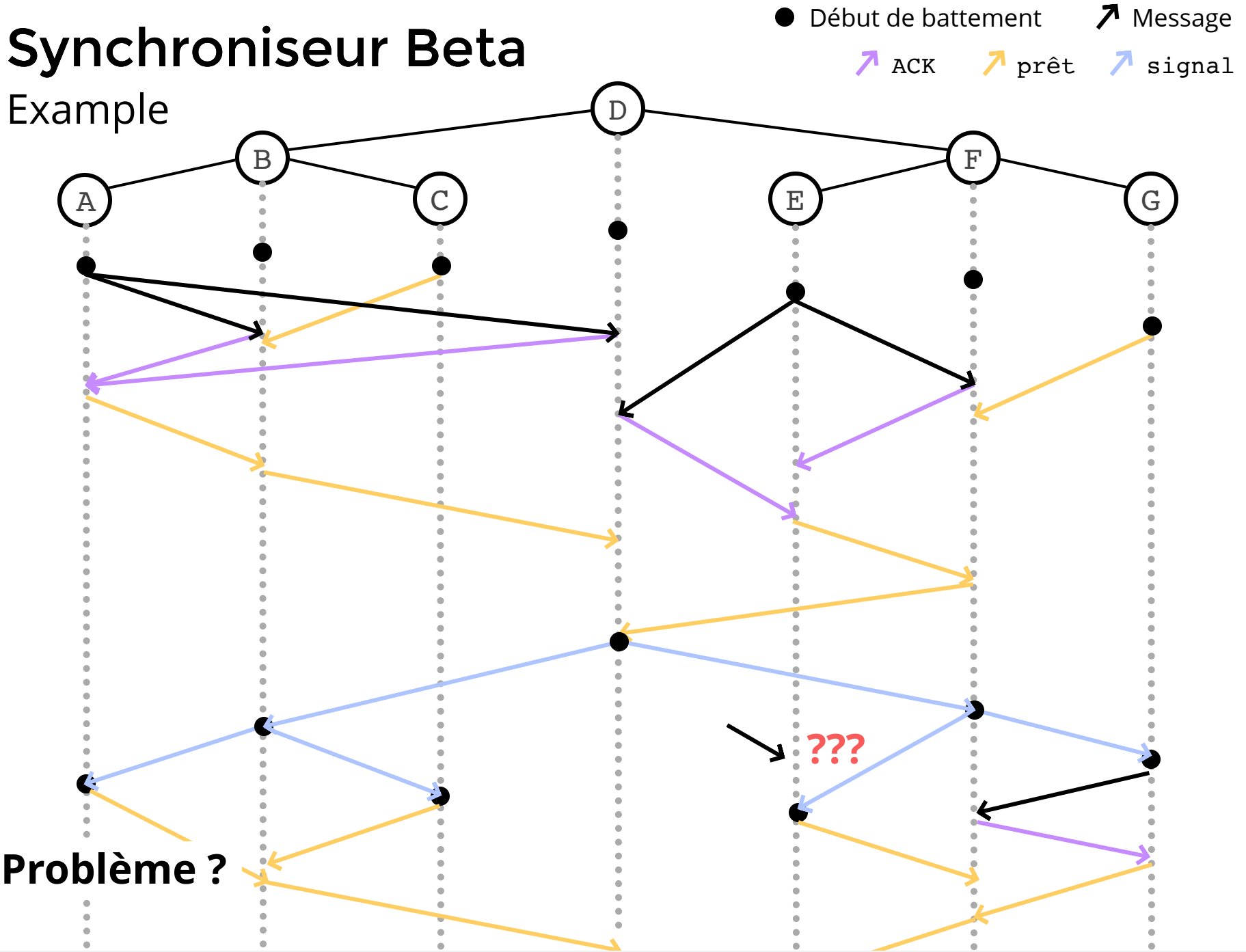


Synchroniseur beta

Exemple & Remarques

Synchroniseur Beta

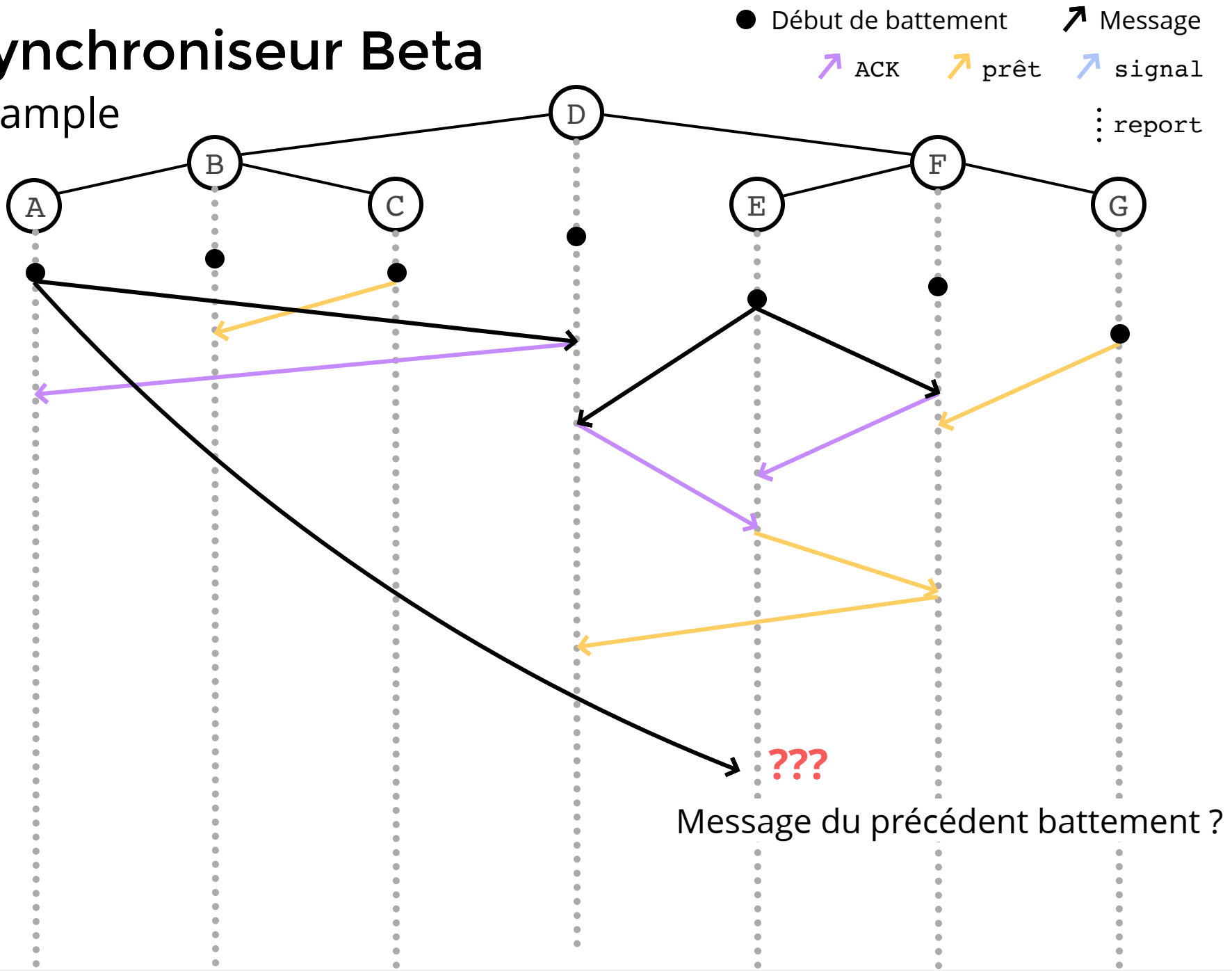
Example



Problème ?

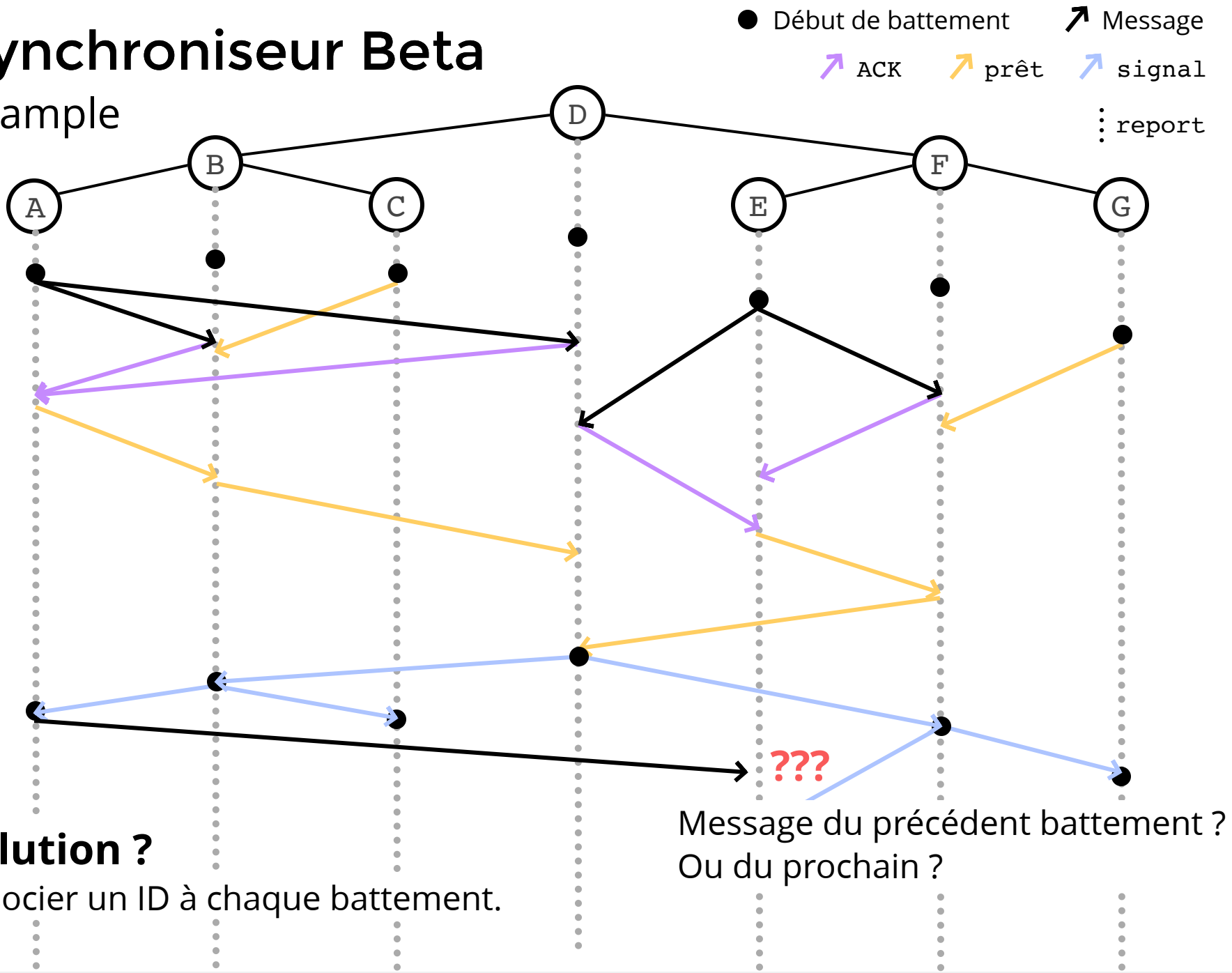
Synchroniseur Beta

Example



Synchroniseur Beta

Example



Solution ?

Associer un ID à chaque battement.

Message du précédent battement ?
Ou du prochain ?

Synchroniseur Beta

Remarques

Associer un ID à chaque battement :

Repousser le traitement d'un message destiné au battement suivant.

Le nouveau battement ne commence pas au même moment partout.

L'arbre ne sert qu'à la logique de l'algorithme. Il peut y avoir d'autres connexions utilisées par l'algorithme synchrone lui-même.

Un processus qui n'a pas de messages à envoyer est prêt dès la fin de ses calculs locaux.

En quoi est-ce mieux qu'un système centralisé ?

C'est un système centralisé !

Mais le "centre" peut changer dynamiquement si besoin.

Il faut alors reconstruire le spanning tree, mais c'est possible.

Synchroniseur Beta

Un algorithme par sondes et échos ?

Ce synchroniseur utilise une approche par sondes et échos sur arbre !

Réception d'une `sonde` *("signal")*

Je la propage à mes enfants

Je fais des calculs (optionnel) ← *Lancement d'un battement*
(et attente de fin de battement local si je n'ai pas d'enfants)

Si je n'ai pas d'enfants

  Je réponds avec `écho`

Réception d'un `écho` *("prêt")*

Si j'ai tous les `échos`

  Je fais des calculs (optionnel) ← *Attente de fin de battement local*

  Je réponds `écho` à mon parent

  Si je suis la source

    J'ai fini



Synchroniseur beta

Résumé & Complexités

Résumé

Début de battement

Pour tout message mis de coté
    Je traite sa réception.
Je fais mes calculs.
J'envoie mes messages.

Réception d'un message de `i`

S'il vient du prochain battement
    Je le mets de coté.
Sinon
    Je réponds `ACK`.
    Je stocke le message.

Réception d'un *prêt* de `i`

Je note que `i` est prêt.

Réception d'un `ACK`

Je note la réception du message.
Si j'ai un `ACK` pour tous mes envois
    Je me note *prêt*.

Tous mes enfants et moi sommes prêts

Si je suis la racine
    Je fais semblant de recevoir un signal.
Sinon
    J'envoie *prêt* à mon parent.

Réception d'un signal

Je réinitialise tous les statuts "prêt".
J'envoie un signal à tous mes enfants.
Je déclenche un nouveau battement.

Pourquoi pas besoin de check si doublon (`i` déjà prêt) ?
Impossible ; j'envoie un signal entre tous 2 "prêt"

Complexité

Communication par battement *(en ignorant les messages de l'algorithme synchronisé)*

Sur chaque lien *de l'arbre*, un `prêt`, et un signal par battement.

Donc 2 fois le nombre de liens *dans l'arbre*.

Donc 2 fois V-1

Donc O(V). ← Mieux !

Performances, en fonction de T

Les messages et leur `ACK`, puis la chaine de *prêt* et du signal.

Donc environ 2T + 2TV dans le pire des cas

2T + 2Tlog(V) si l'arbre est parfaitement équilibré.

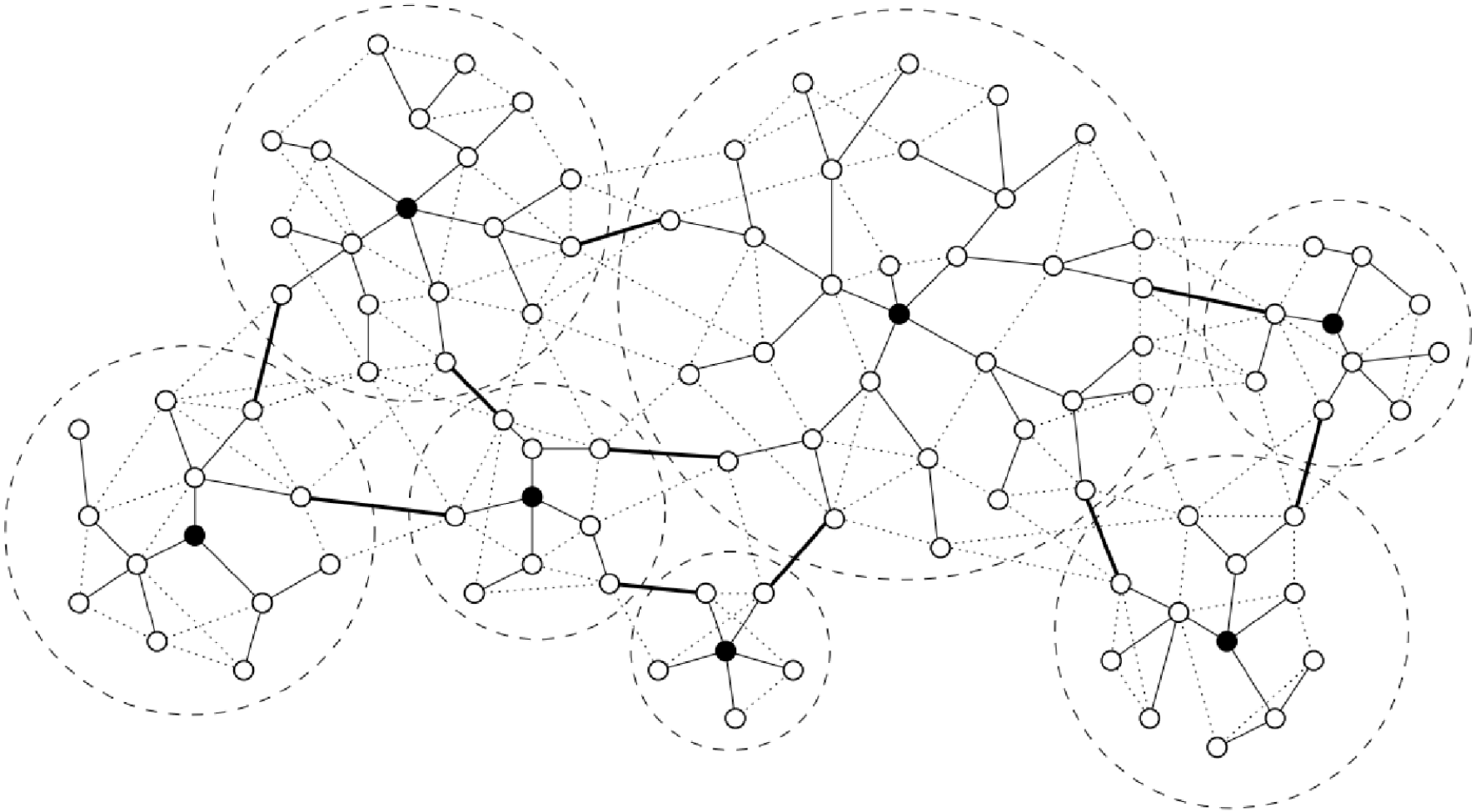
Donc O(TV), ou O(Tlog(V)) si l'arbre est équilibré. ← Moins bien...

Avec V = nombre de processus ; E = nombre de liens ; T = durée de transit maximale

↓ Synchroniseur Gamma ?

Le meilleur des deux mondes

FYI



https://www.iitg.ac.in/psm/GIAN-DNA/material/5-sync.pdf