SDR

Consensus

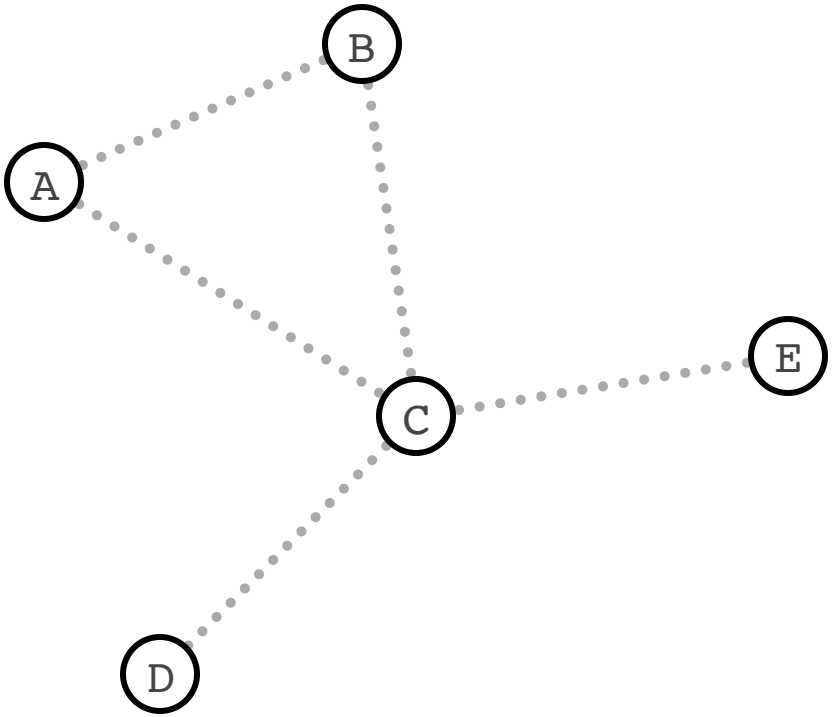
Consensus naïf

Olivier Lemer

↓ Consensus ?

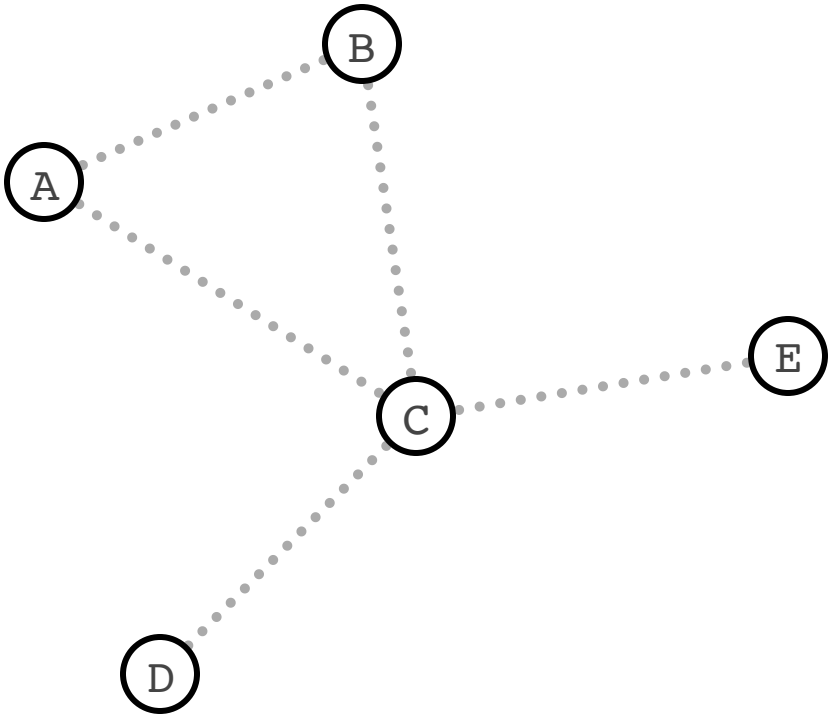
Consensus

Qu'est-ce que c'est ?



Consensus

Qu'est-ce que c'est ?



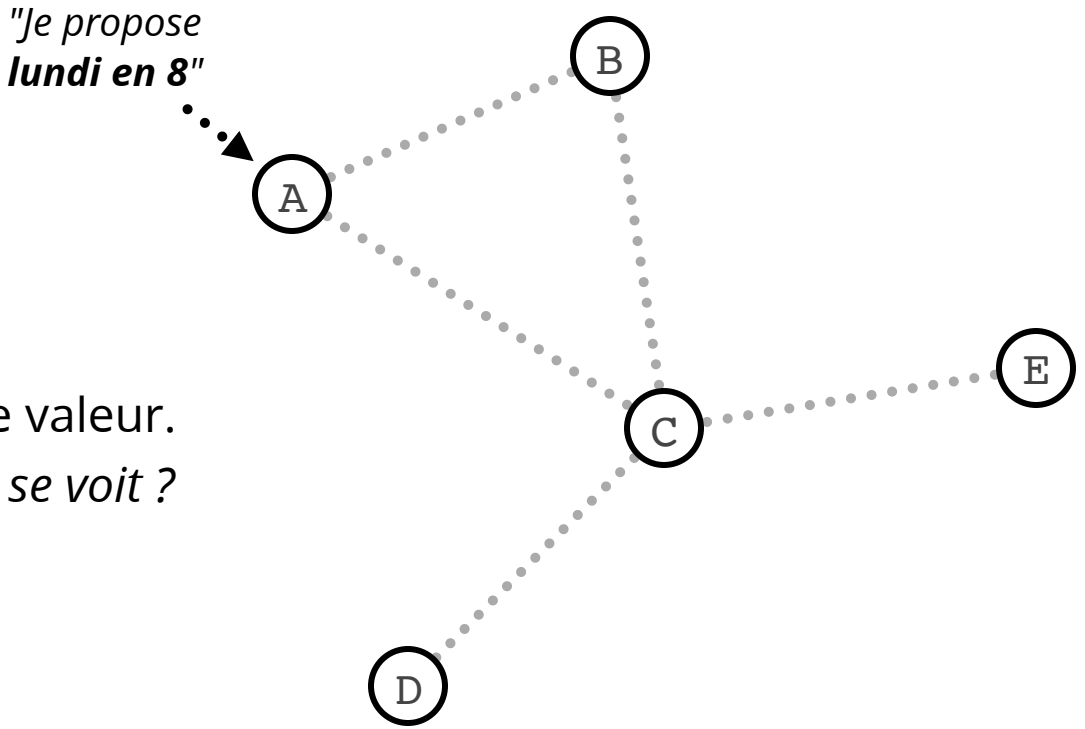
Au début

Chaque processus **propose** une valeur.

Par exemple : quand est-ce qu'on se voit ?

Consensus

Qu'est-ce que c'est ?



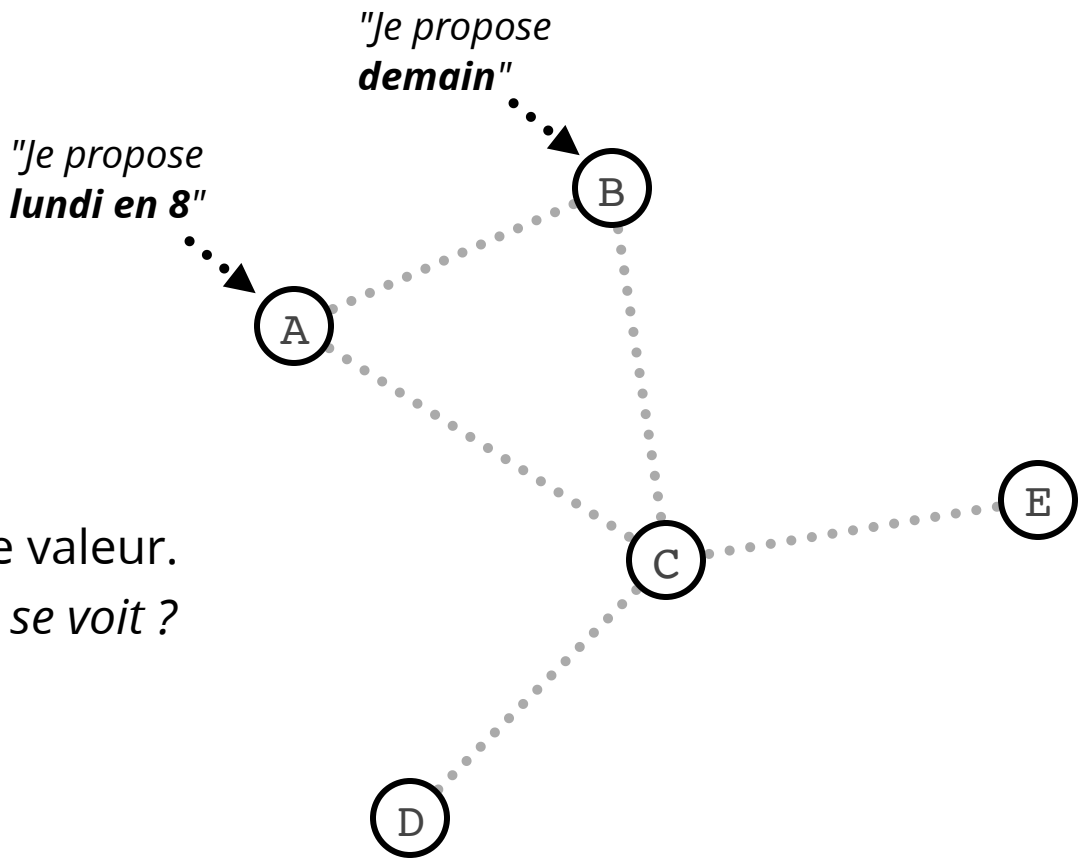
Au début

Chaque processus **propose** une valeur.

Par exemple : quand est-ce qu'on se voit ?

Consensus

Qu'est-ce que c'est ?



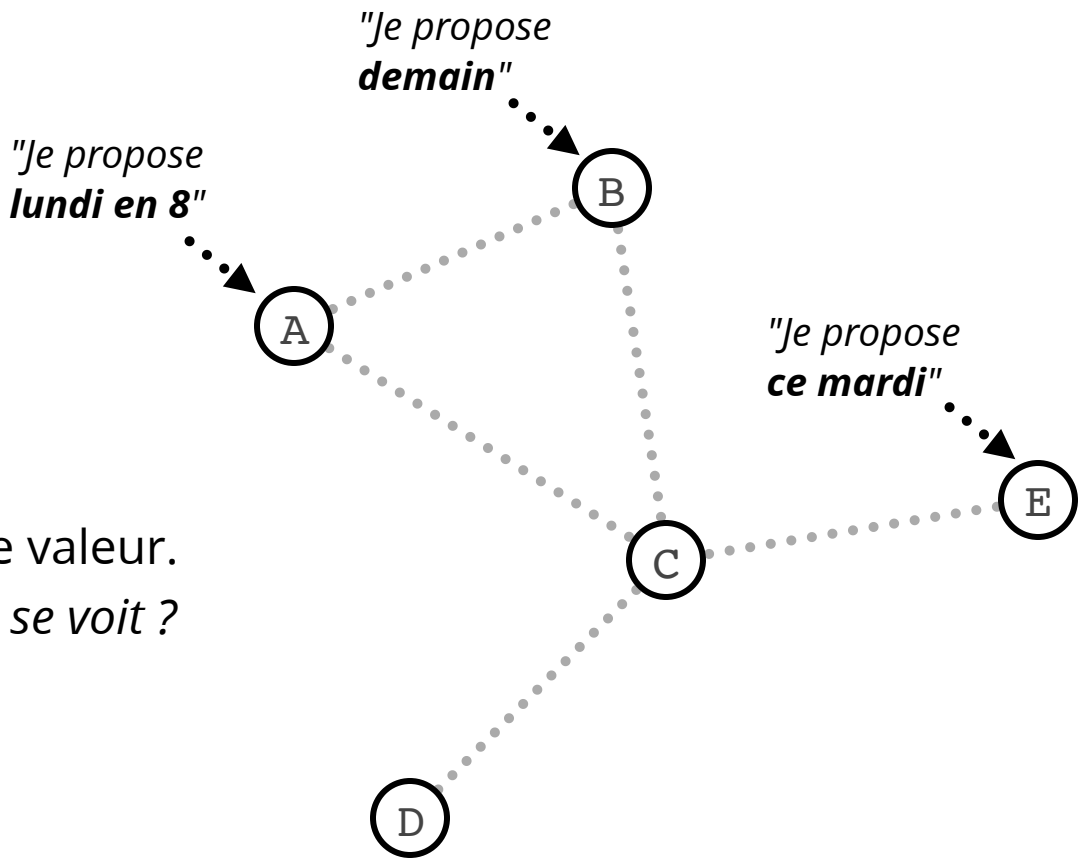
Au début

Chaque processus **propose** une valeur.

Par exemple : quand est-ce qu'on se voit ?

Consensus

Qu'est-ce que c'est ?



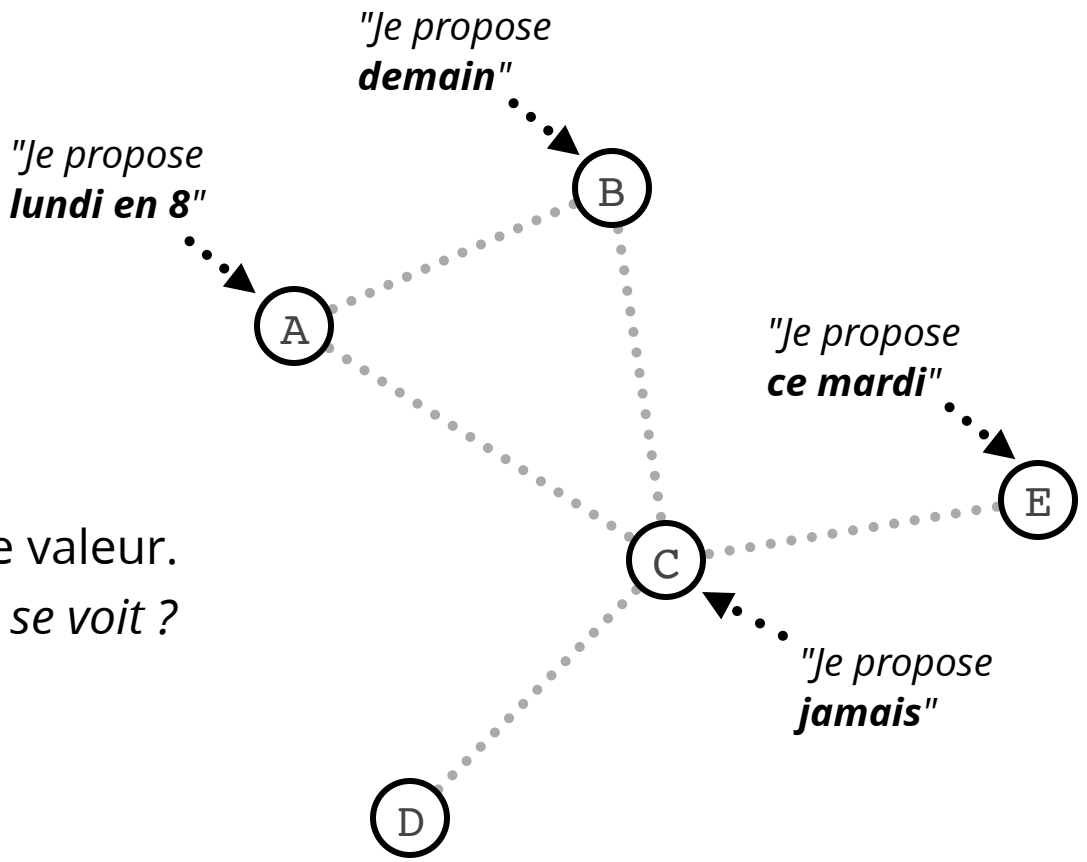
Au début

Chaque processus **propose** une valeur.

Par exemple : quand est-ce qu'on se voit ?

Consensus

Qu'est-ce que c'est ?



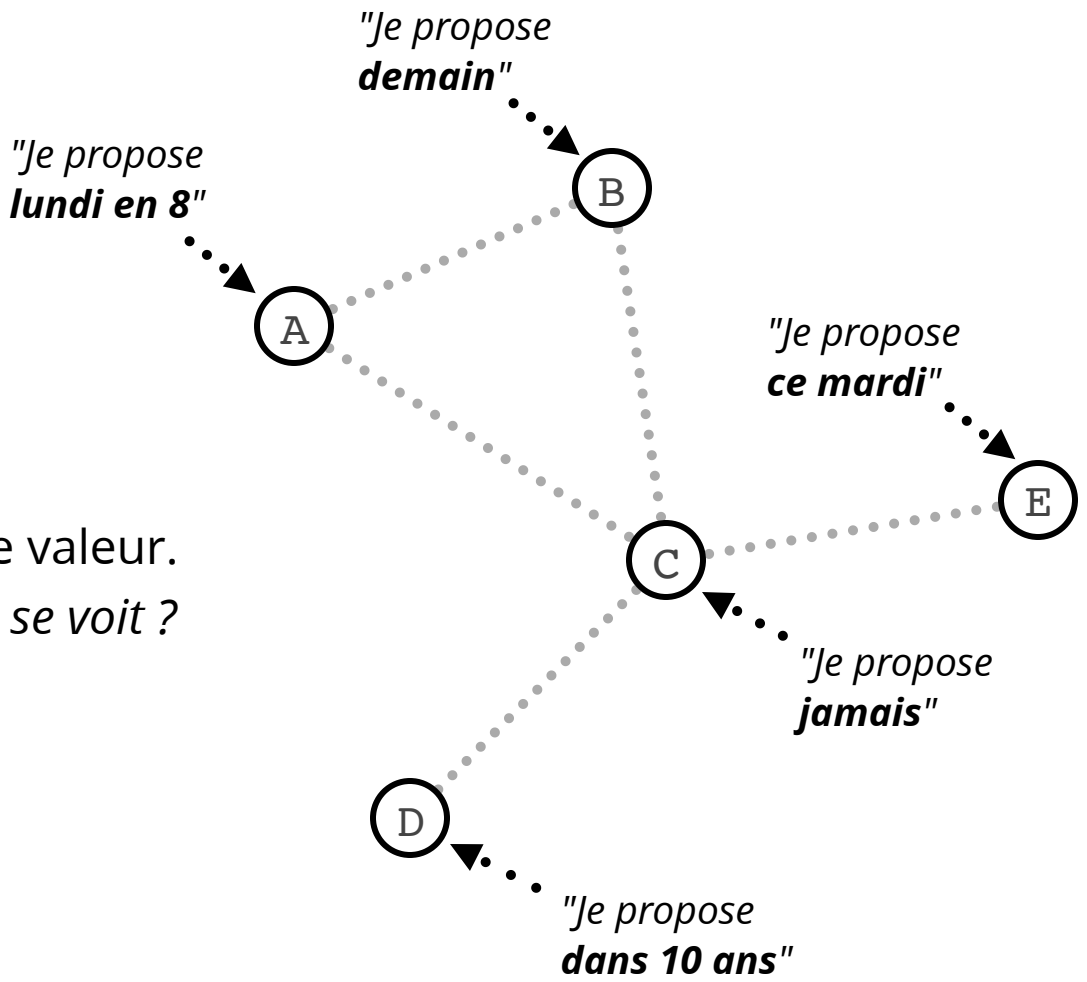
Au début

Chaque processus **propose** une valeur.

Par exemple : quand est-ce qu'on se voit ?

Consensus

Qu'est-ce que c'est ?



Au début

Chaque processus **propose** une valeur.

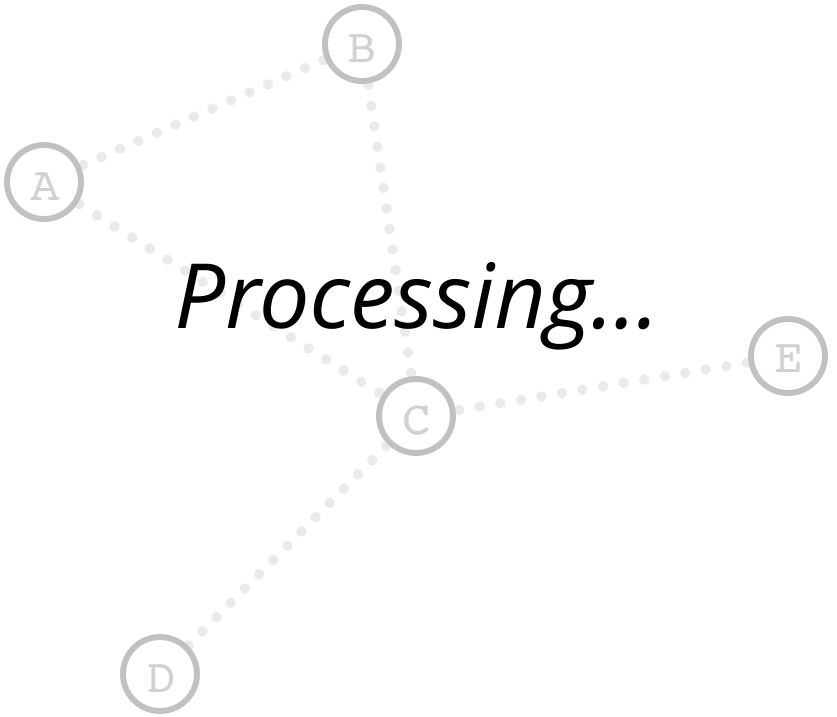
Par exemple : quand est-ce qu'on se voit ?

Consensus

Qu'est-ce que c'est ?

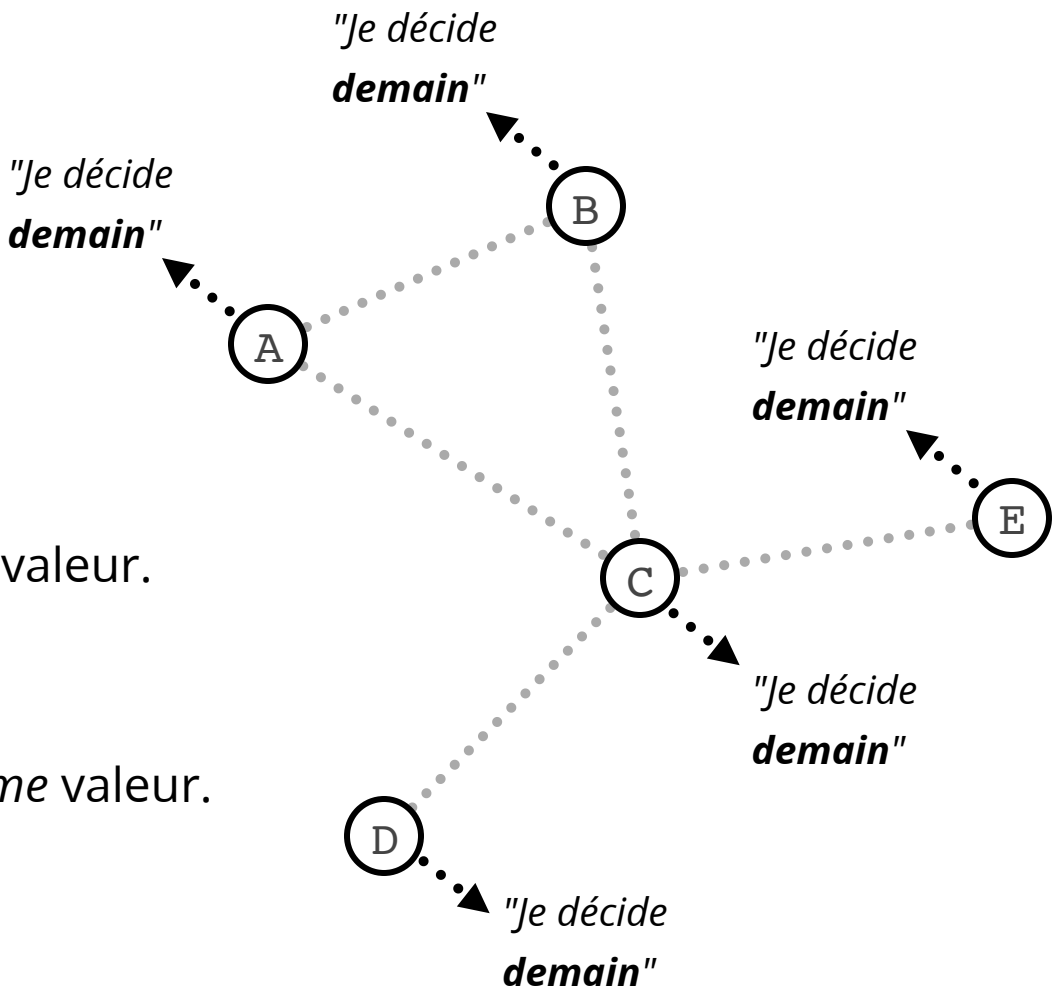
Au début

Chaque processus **propose** une valeur.



Consensus

Qu'est-ce que c'est ?



Au début

Chaque processus **propose** une valeur.

À la fin

Chaque processus **décide** la *même* valeur.

(Laquelle ? C'est arbitraire.)

↓ Suppositions du jour

Suppositions du jour

Pas de perte, de duplication, ou de réordonnancement des messages.

Toute panne est **permanente**.

Il existe une borne supérieure constante **`T`**
sur la durée de transit de tout message.
On a donc un système "synchrone".
*(À ne pas confondre avec un **algorithme** synchrone)*

Les durées de traitement sont négligeables.

Le réseau est une clique (tous connectés à tous).

Suppositions du jour

Pas de perte, de duplication, ou de réordonnancement des messages.

Toute panne est **permanente**.

Il existe une borne supérieure constante $\mathbf{T}$ sur la durée de transit de tout message.
On a donc un système "synchrone".
*(À ne pas confondre avec un **algorithme** synchrone)*

Les durées de traitement sont négligeables.

*Comprendre : on pourra utiliser un détecteur de pannes **parfait***

Le réseau est une clique (tous connectés à tous).

Pourquoi cette supposition sur `T` ?

"C'est une supposition irréaliste d'avoir une borne sur la durée de transit des messages"

Et pourtant on l'utilise quand même ! Pourquoi ?

Speaker notes

Fischer, M., N. Lynch, and M. Paterson (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382.

Pourquoi cette supposition sur `T` ?

"C'est une supposition irréaliste d'avoir une borne sur la durée de transit des messages"

Et pourtant on l'utilise quand même ! Pourquoi ?

"Sans garantie sur la durée de transit des messages,
***il est impossible d'atteindre un consensus**,*
dès que même un seul processus peut tomber en panne."

Fischer, M., N. Lynch, and M. Paterson (1985).

Speaker notes

Fischer, M., N. Lynch, and M. Paterson (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382.

Pourquoi cette supposition sur `T` ?

> *"C'est une supposition irréaliste d'avoir une borne sur la durée de transit des messages"*

Et pourtant on l'utilise quand même ! Pourquoi ?

> *"Sans garantie sur la durée de transit des messages,*
> ***il est impossible d'atteindre un consensus**,*
> *dès que même un seul processus peut tomber en panne."*
>
> *Fischer, M., N. Lynch, and M. Paterson (1985).*

On a donc besoin d'un système au moins *partiellement synchrone*.

Speaker notes

Fischer, M., N. Lynch, and M. Paterson (1985). Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382.

Propriétés

Ce qui définit le consensus

Propriétés

Propriétés

Terminaison

Tout processus correct **décide** d'une valeur, *un jour*.

(on finira par décider)

Propriétés

Terminaison

Tout processus correct **décide** d'une valeur, *un jour*.

(on finira par décider)

Validité

Si un processus décide $\mathbf{v}$, alors un processus a proposé $\mathbf{v}$.

(on n'invente pas de valeur)

Propriétés

Terminaison

Tout processus correct **décide** d'une valeur, *un jour*.

(on finira par décider)

Validité

Si un processus décide $\mathtt{v}$, alors un processus a proposé $\mathtt{v}$.

(on n'invente pas de valeur)

Unicité

Aucun processus ne décide deux fois.

(une seule décision par consensus)

Propriétés

Terminaison

Tout processus correct **décide** d'une valeur, *un jour*.

(on finira par décider)

Validité

Si un processus décide $\mathtt{v}$, alors un processus a proposé $\mathtt{v}$.

(on n'invente pas de valeur)

Unicité

Aucun processus ne décide deux fois.

(une seule décision par consensus)

Accord

Tous deux processus décident la même valeur

(on décide tous pareil)

↓ Solution naïve

Solution naïve

Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

A

B

C

D

Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

Solution naïve

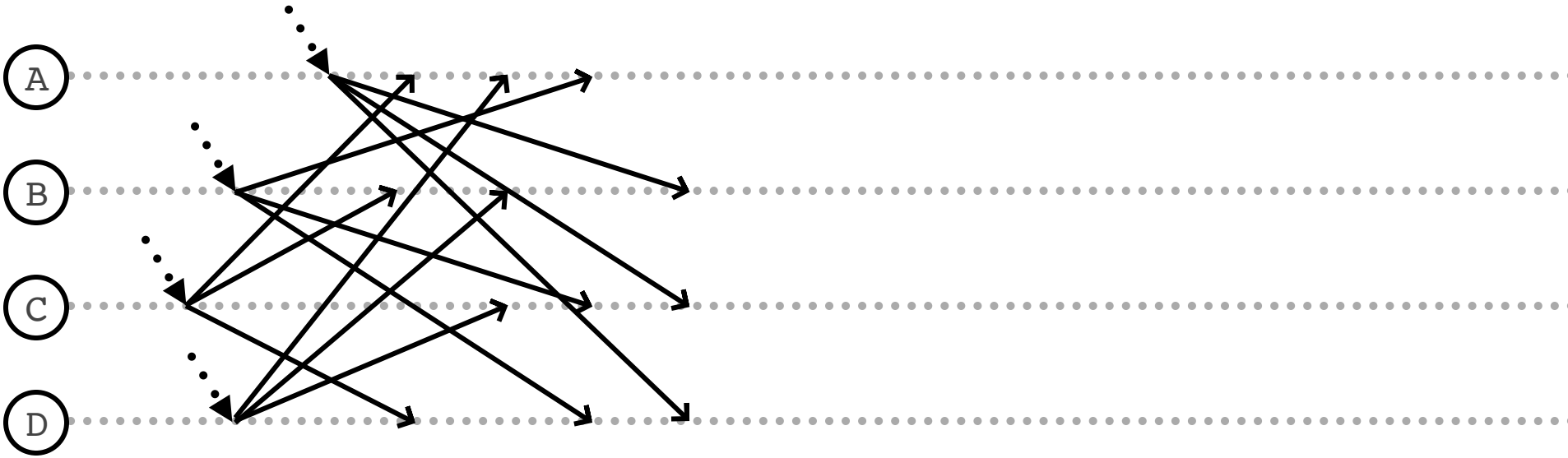
Quand je propose

Envoyer ma proposition à tout le monde.

Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.



Solution naïve

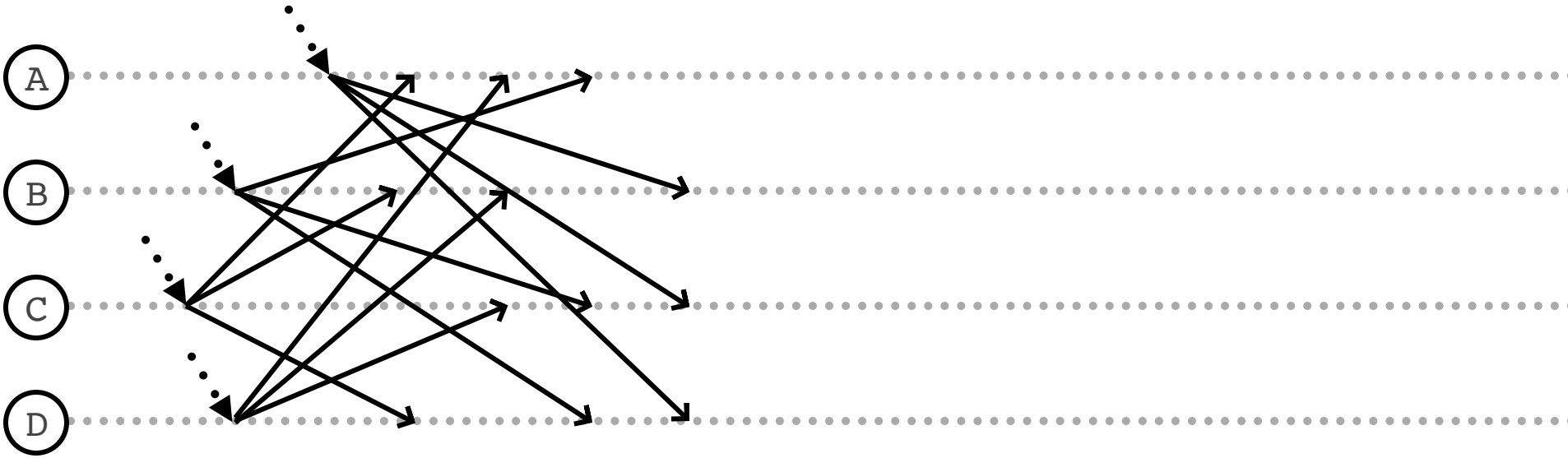
Quand je propose

Envoyer ma proposition à tout le monde.

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

(i.e. en suivant tous le même algorithme)



Solution naïve

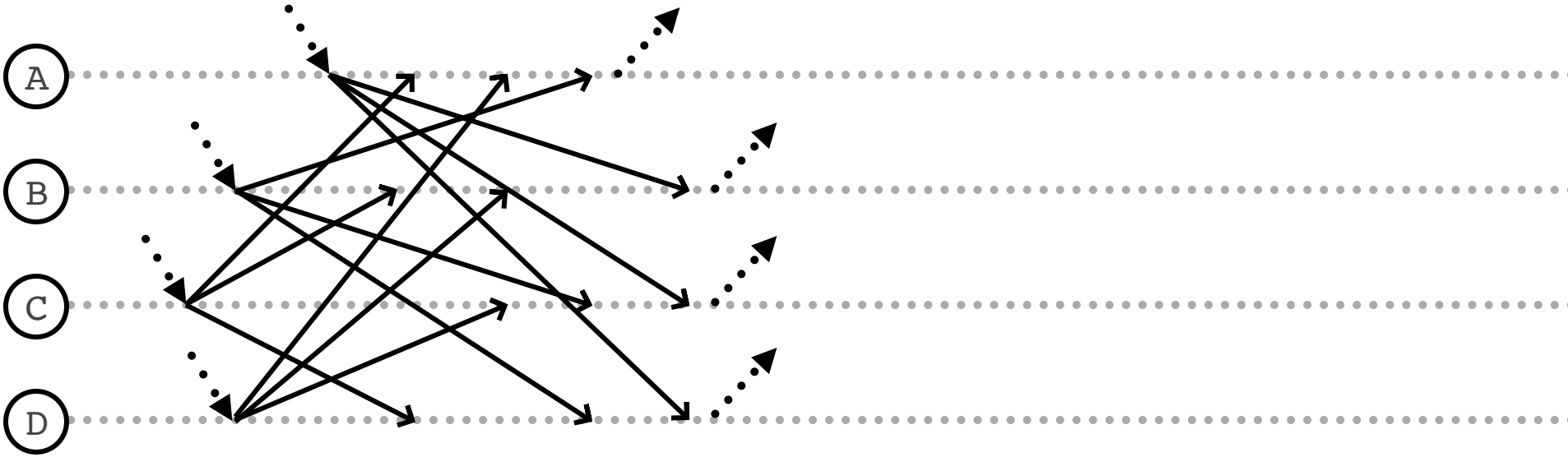
Quand je propose

Envoyer ma proposition à tout le monde.

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

(i.e. en suivant tous le même algorithme)



Solution naïve

Quand je propose

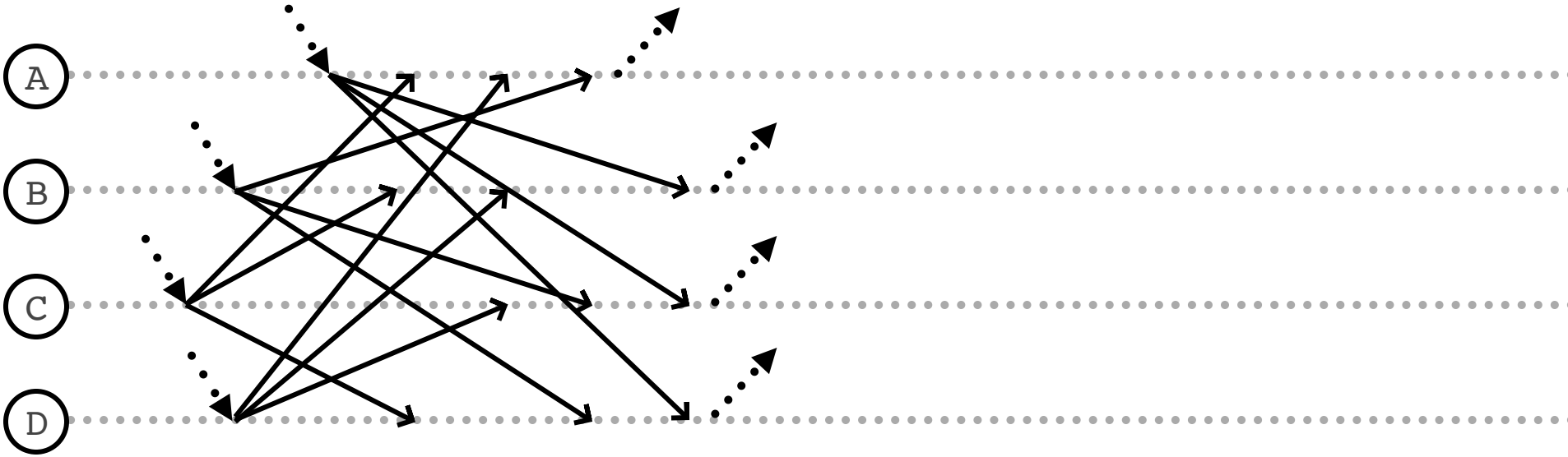
Envoyer ma proposition à tout le monde.

Problème ?

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

(i.e. en suivant tous le même algorithme)



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

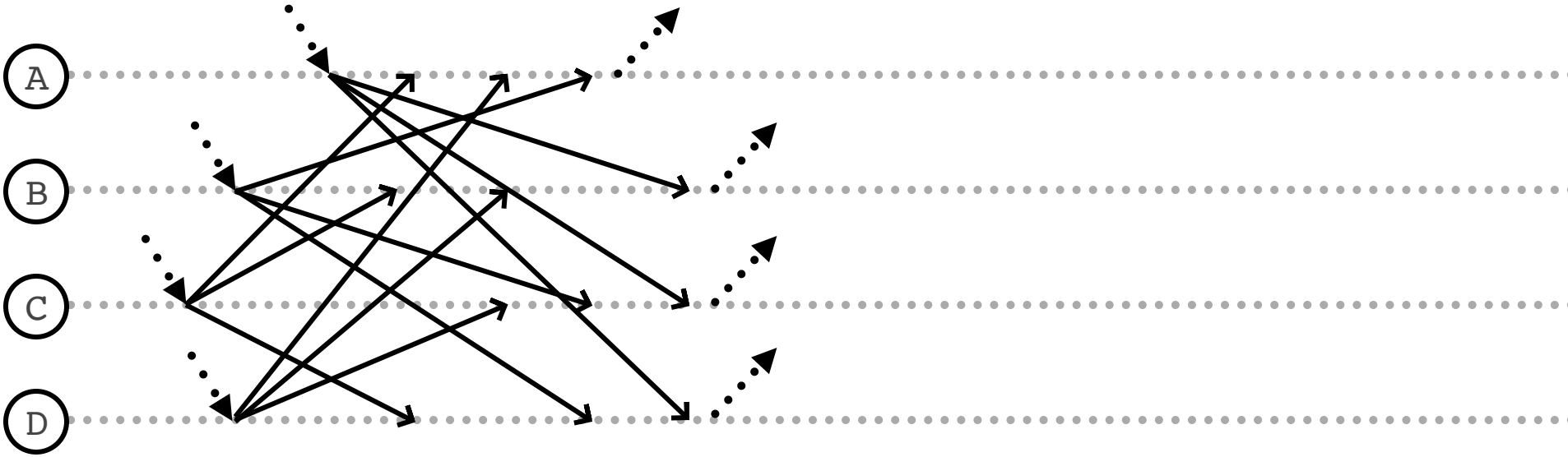
Problème ?

Si un processus tombe en panne ?

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

(i.e. en suivant tous le même algorithme)



Solution naïve

Quand je propose

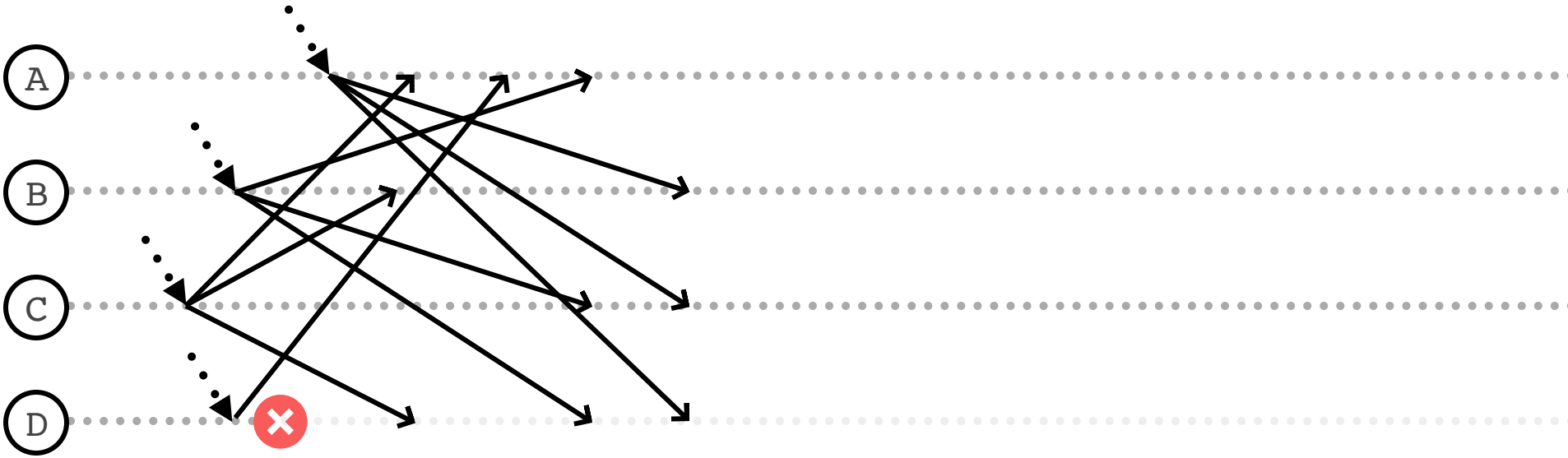
Envoyer ma proposition à tout le monde.

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?



Solution naïve

Quand je propose

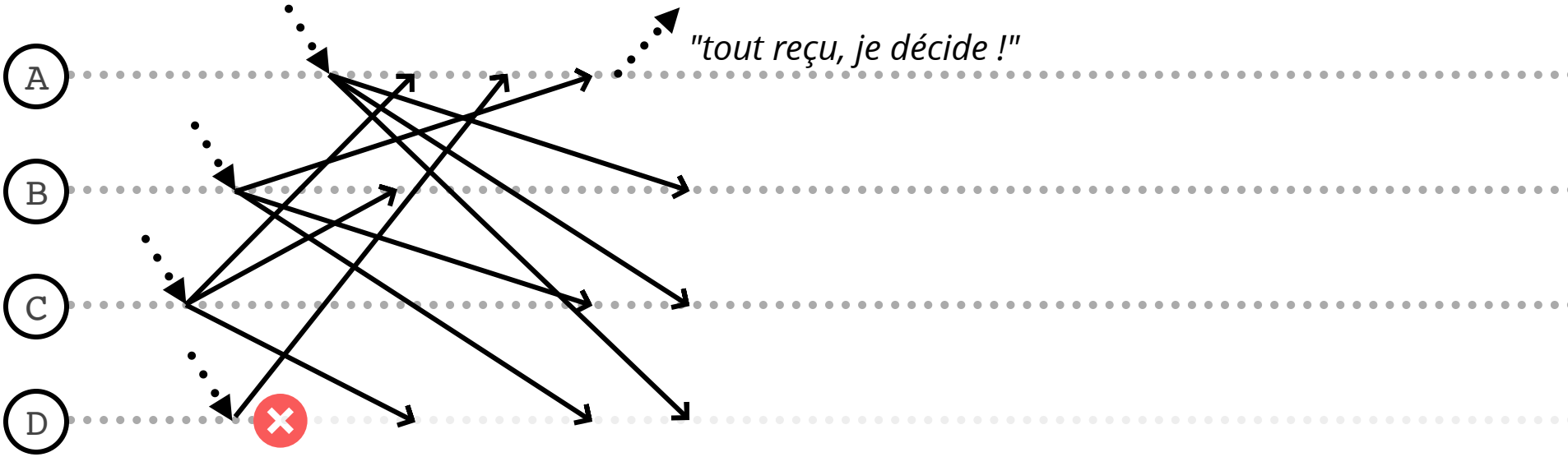
Envoyer ma proposition à tout le monde.

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?



Solution naïve

Quand je propose

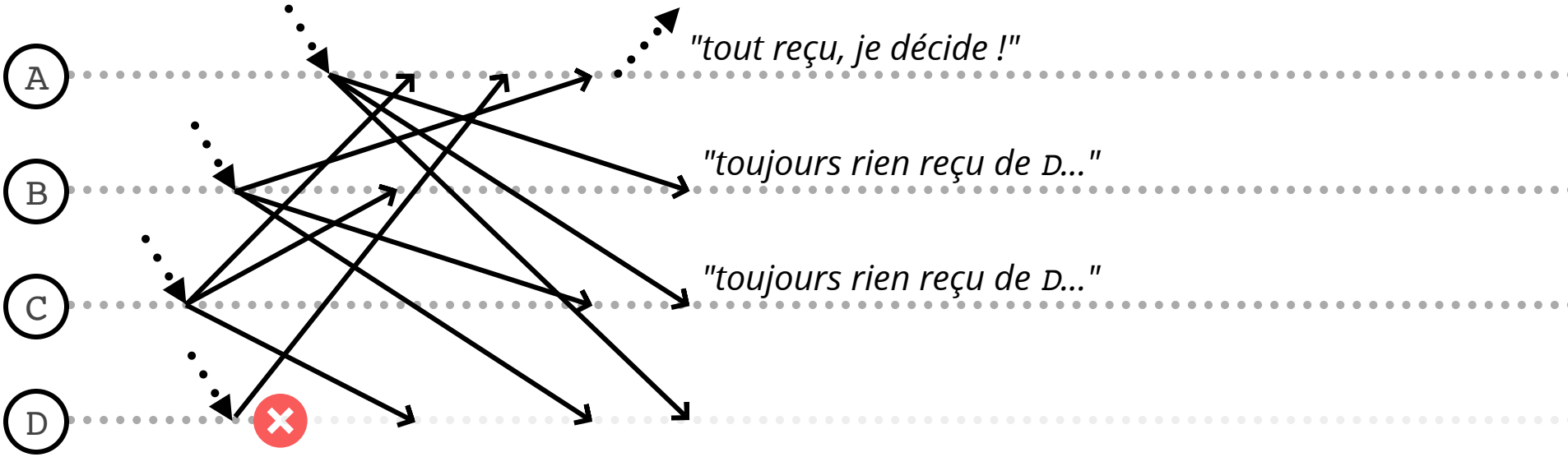
Envoyer ma proposition à tout le monde.

Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

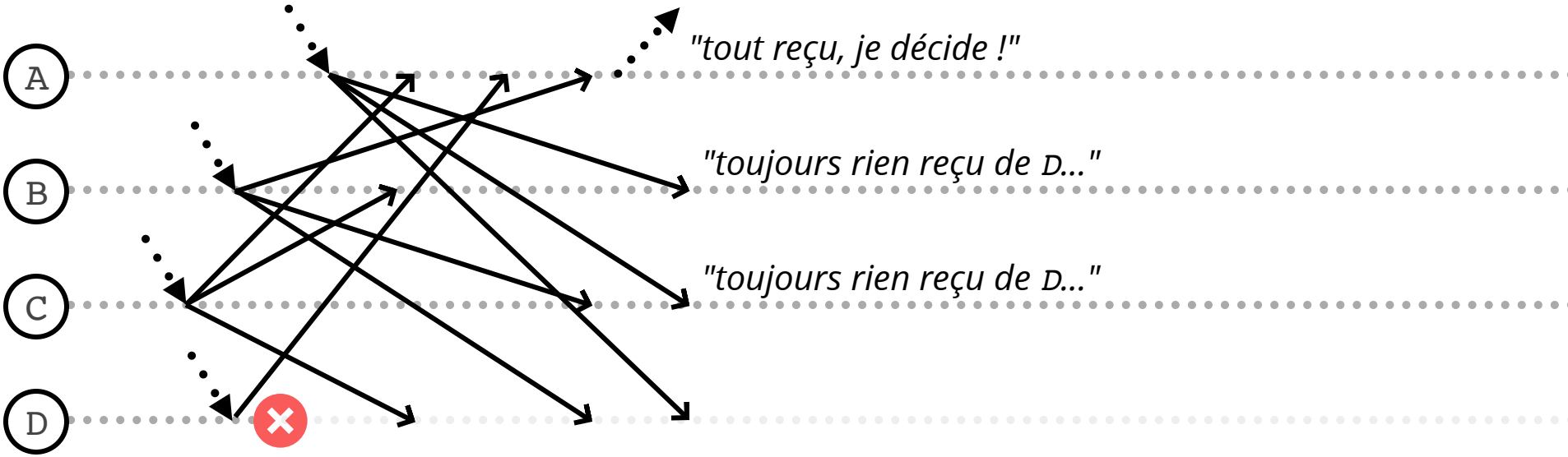
Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?

*Risque de perte de **terminaison**.*



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

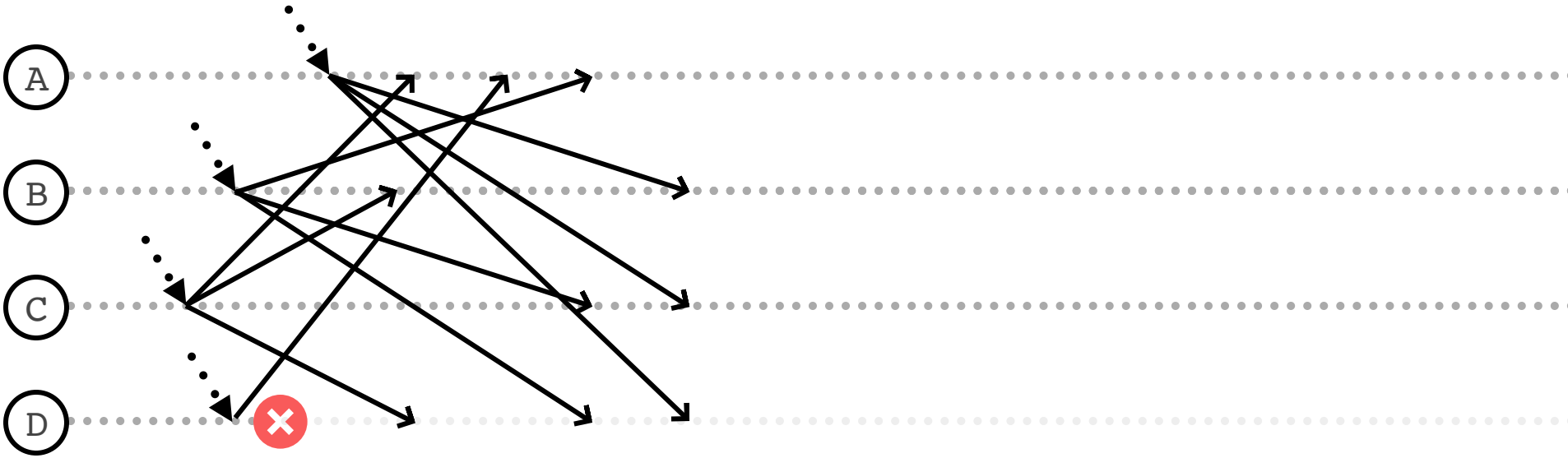
Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?

*Risque de perte de **terminaison**,*



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

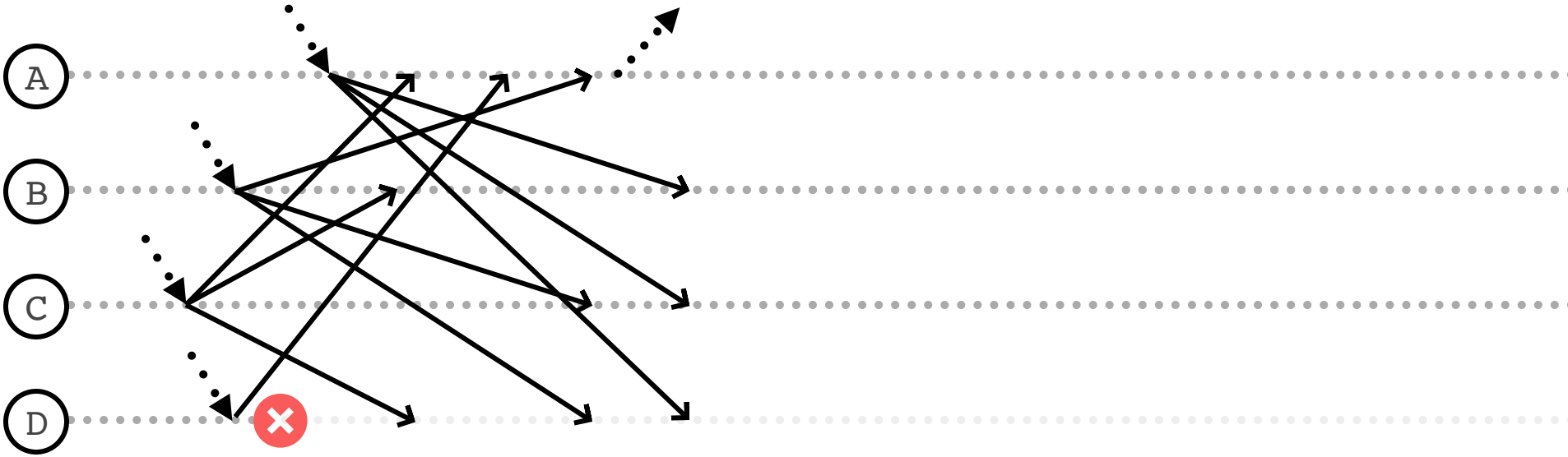
Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?

*Risque de perte de **terminaison**,*



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

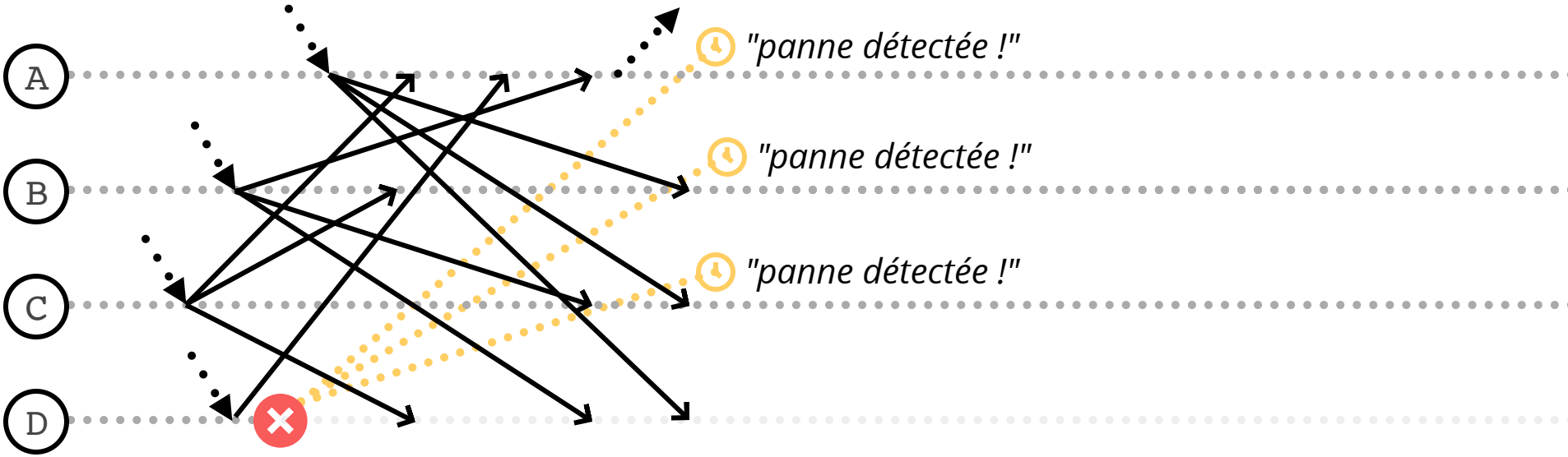
Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?

*Risque de perte de **terminaison**,*



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

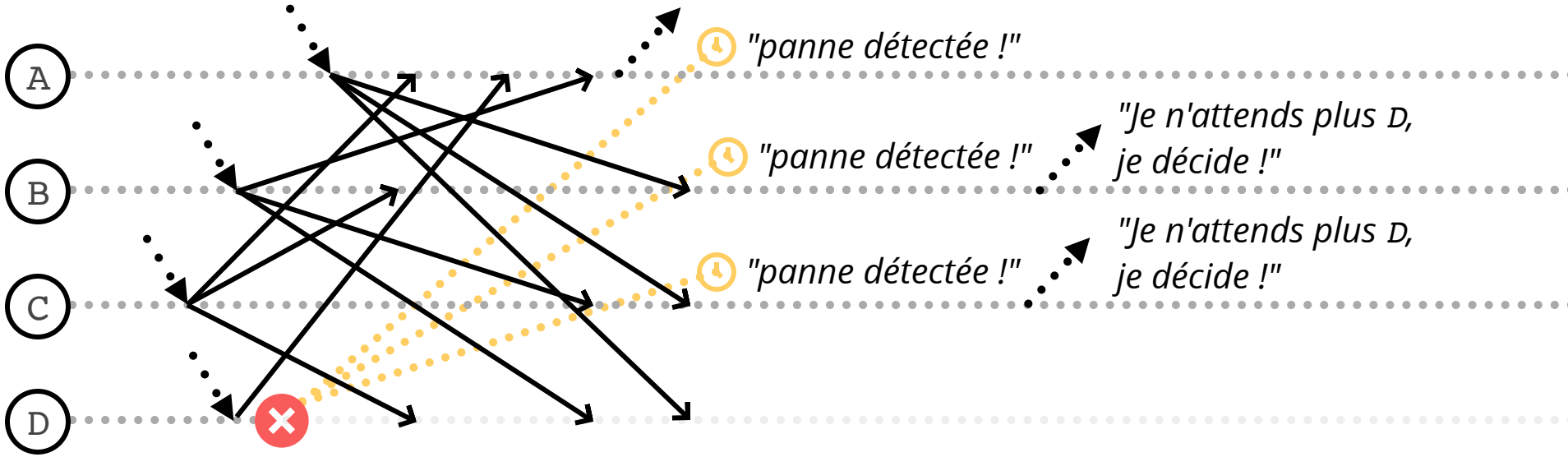
Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?

*Risque de perte de **terminaison**,*



Solution naïve

Quand je propose

Envoyer ma proposition à tout le monde.

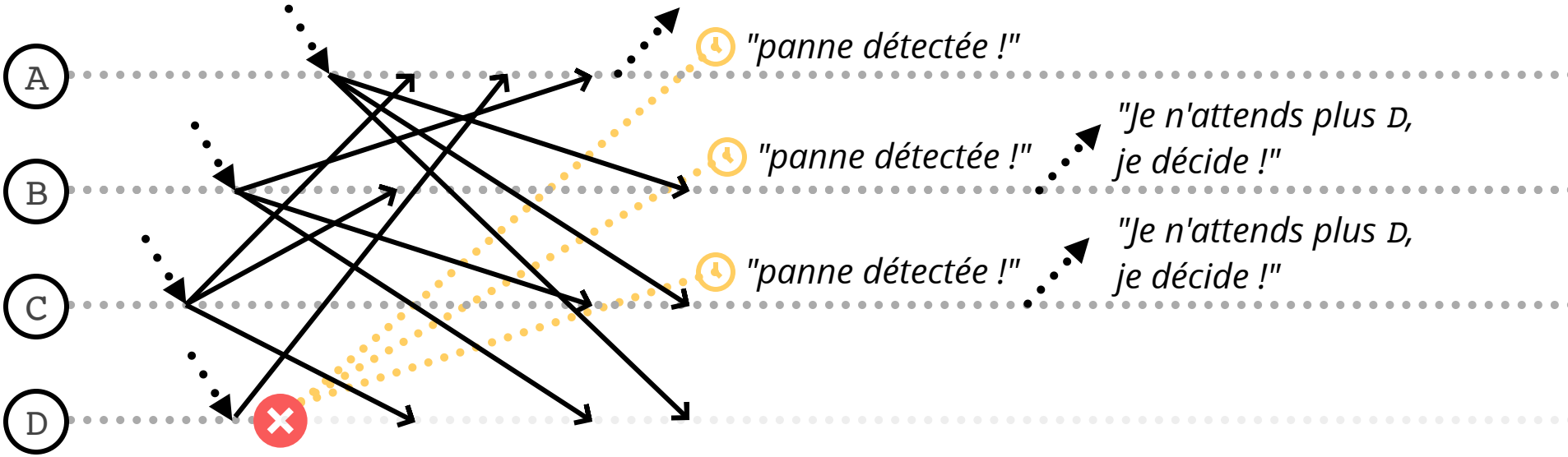
Quand j'ai reçu toutes les propositions

J'en choisis une de manière déterministe.

Problème ?

Si un processus tombe en panne ?
Durant un envoi ?

*Risque de perte de **terminaison**,*
*ou d'**accord** (si détection de pannes).*

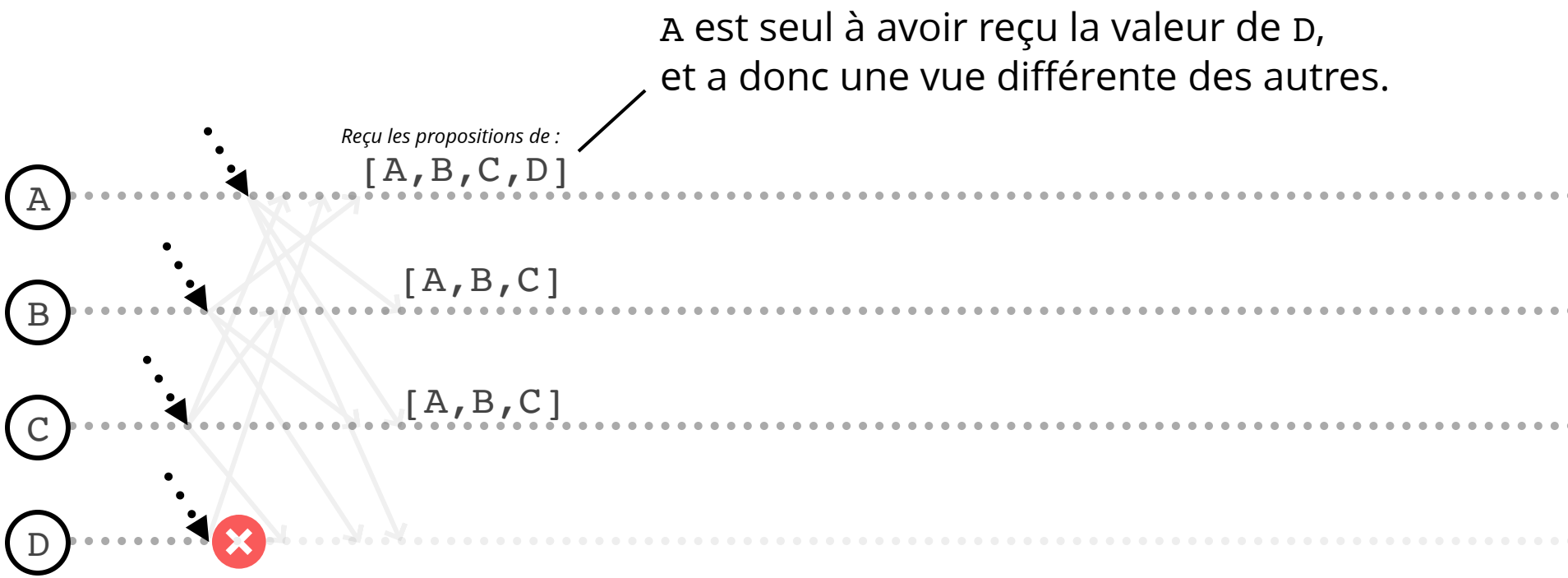


Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.



Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ?

Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ? Qu'est-ce qu'on veut ? On garde la proposition de `D` ou pas ?

Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ? Qu'est-ce qu'on veut ? On garde la proposition de `D` ou pas ?
Souvenez-vous de la propriété de "Validité" :
"Si un processus décide v, alors un processus a proposé v."

Solution naïve

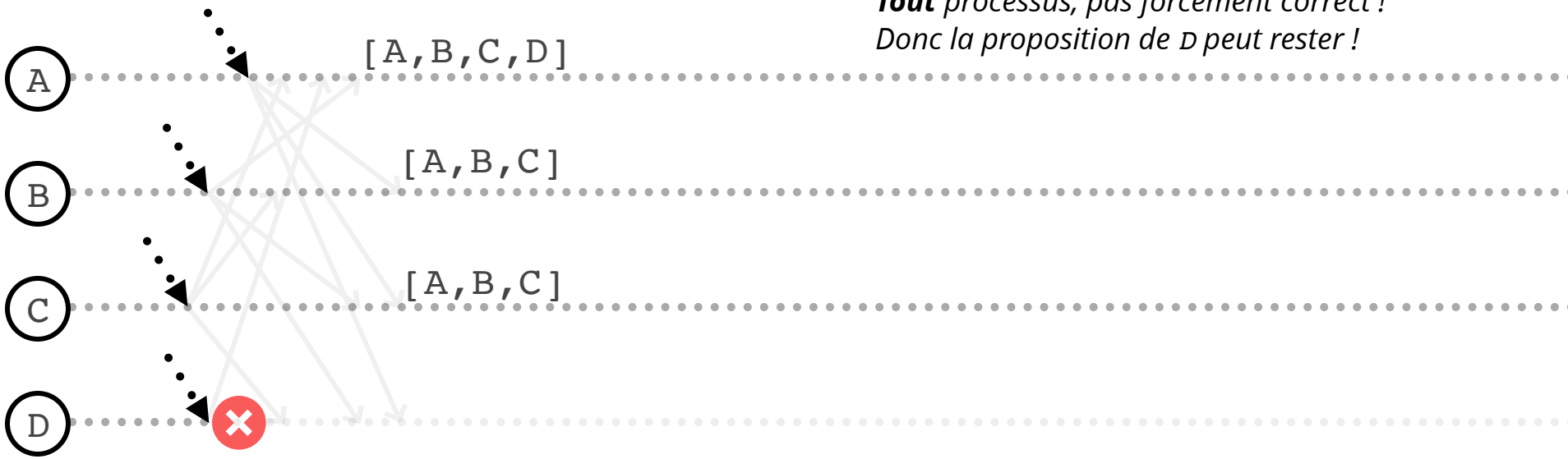
Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ? Qu'est-ce qu'on veut ? On garde la proposition de `D` ou pas ?
Souvenez-vous de la propriété de "Validité" :
"Si un processus décide `v`, alors un processus a proposé `v`."

***Tout** processus, pas forcément correct !*
Donc la proposition de `D` peut rester !



Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ? Il faut propager ce qu'on sait aux autres !

Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ? Il faut propager ce qu'on sait aux autres !
Quand je n'attends plus de proposition, je lance un nouveau round.

Solution naïve

Le problème

Si un processus tombe en panne

Possible que tout le monde n'ait pas toutes les valeurs.

Solution ? Il faut propager ce qu'on sait aux autres !

Quand je n'attends plus de proposition, je lance un nouveau round.

Pour chaque processus, soit j'ai reçu sa proposition, soit j'ai détecté sa panne.

Plusieurs rounds

Comme solution à notre problème

Deuxième round

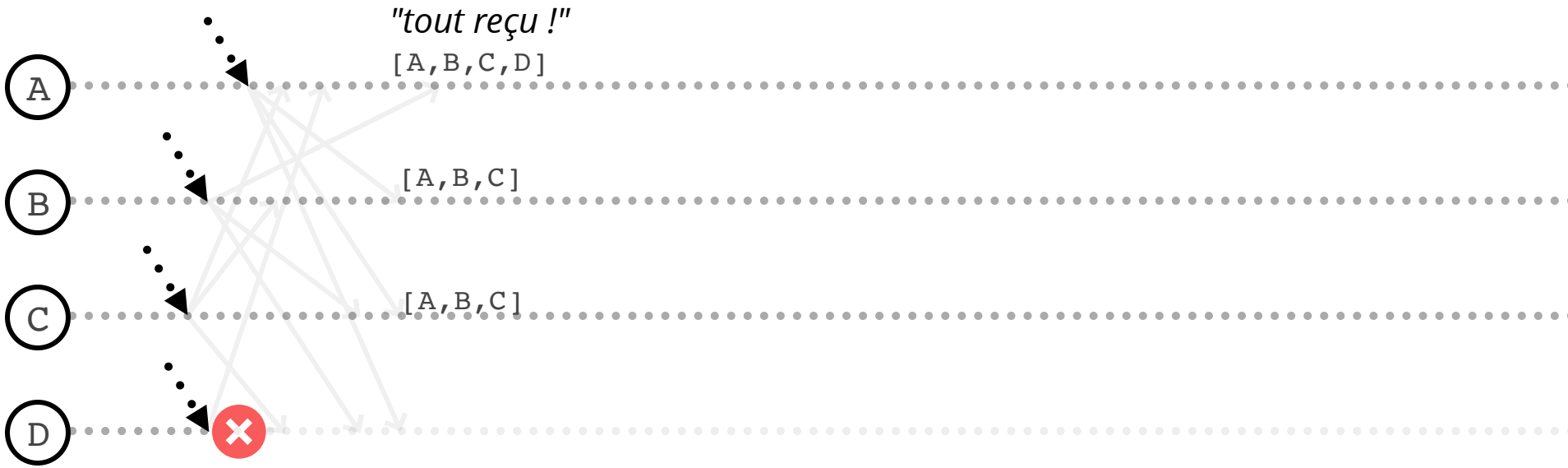
Lorsque je n'ai plus rien à recevoir

Je lance un deuxième round, avec toutes les valeurs connues.

Deuxième round

Lorsque je n'ai plus rien à recevoir

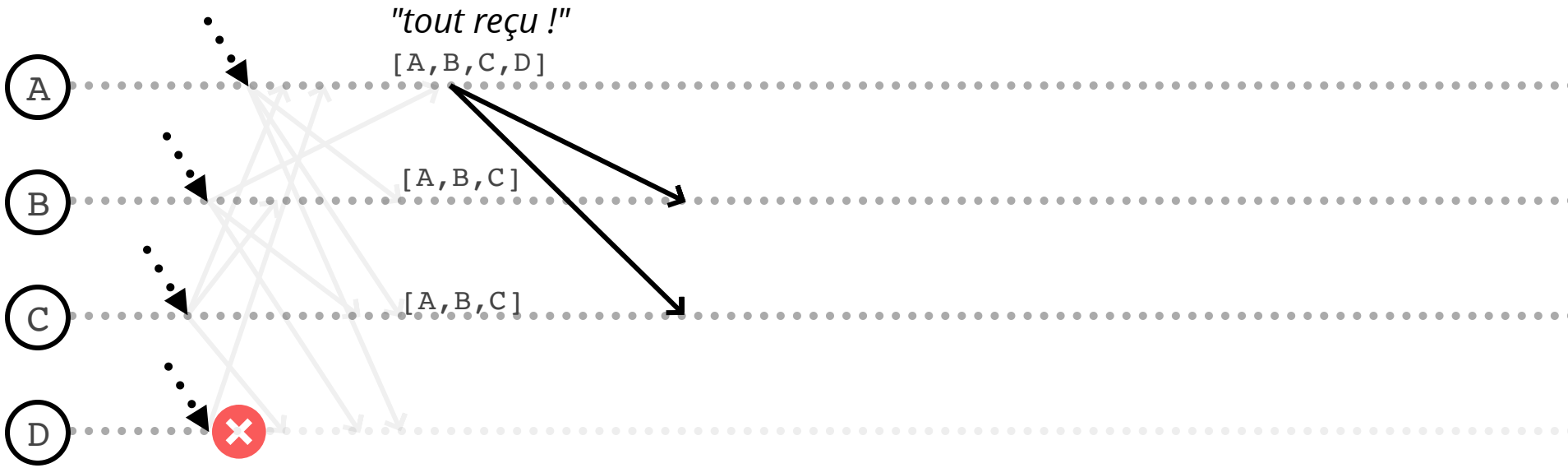
Je lance un deuxième round, avec toutes les valeurs connues.



Deuxième round

Lorsque je n'ai plus rien à recevoir

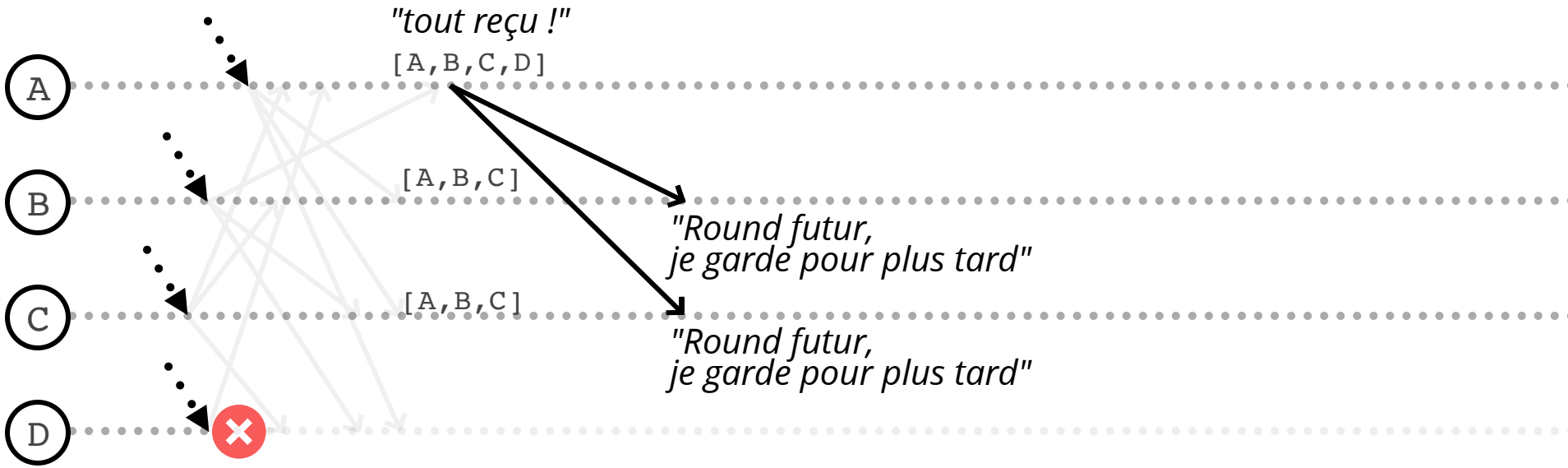
Je lance un deuxième round, avec toutes les valeurs connues.



Deuxième round

Lorsque je n'ai plus rien à recevoir

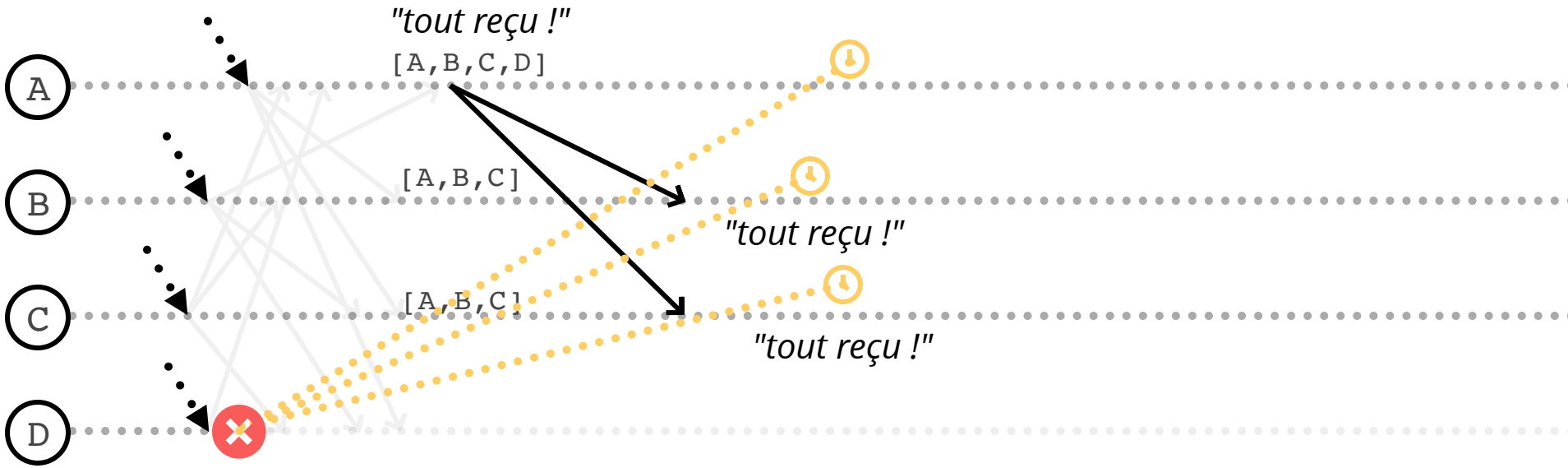
Je lance un deuxième round, avec toutes les valeurs connues.



Deuxième round

Lorsque je n'ai plus rien à recevoir

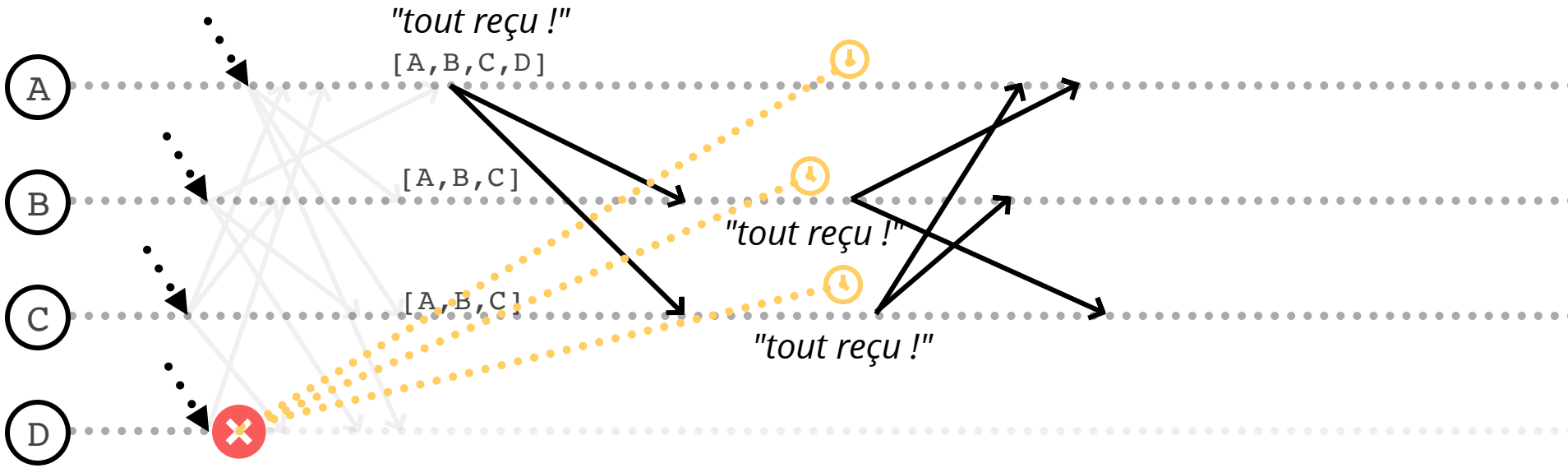
Je lance un deuxième round, avec toutes les valeurs connues.



Deuxième round

Lorsque je n'ai plus rien à recevoir

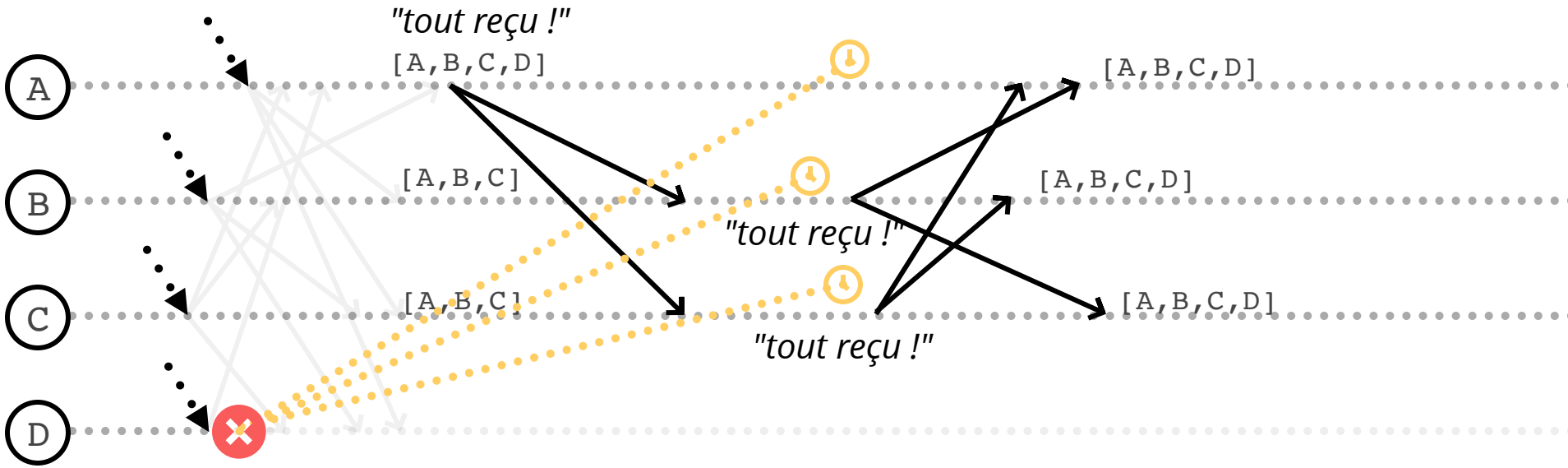
Je lance un deuxième round, avec toutes les valeurs connues.



Deuxième round

Lorsque je n'ai plus rien à recevoir

Je lance un deuxième round, avec toutes les valeurs connues.

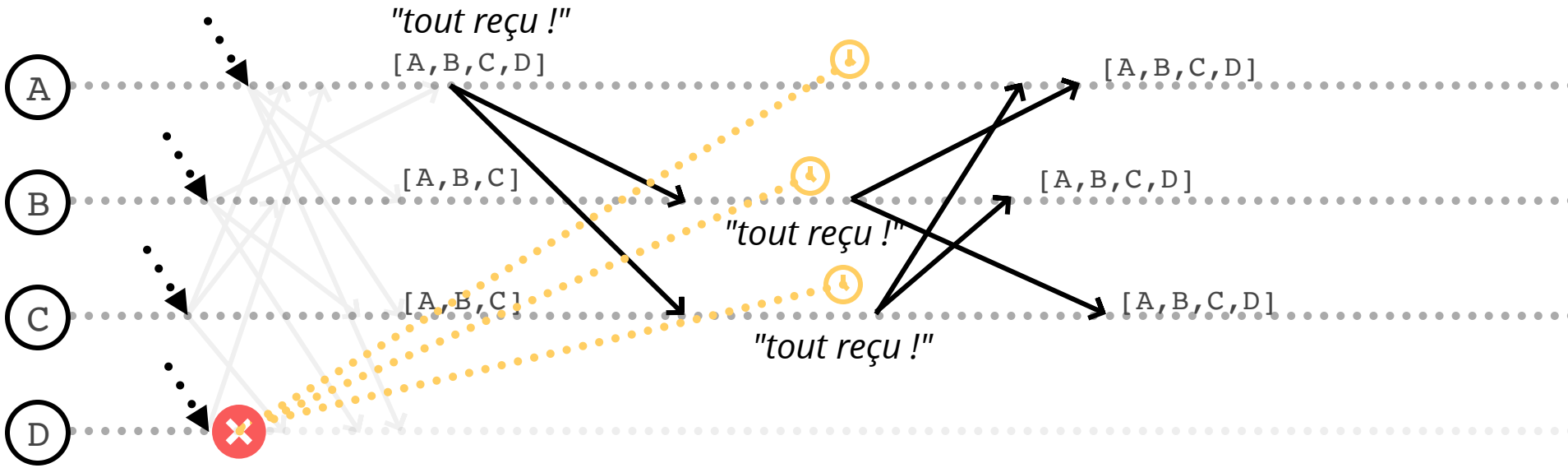


Deuxième round

Lorsque je n'ai plus rien à recevoir

Je lance un deuxième round, avec toutes les valeurs connues.

On est bon ?



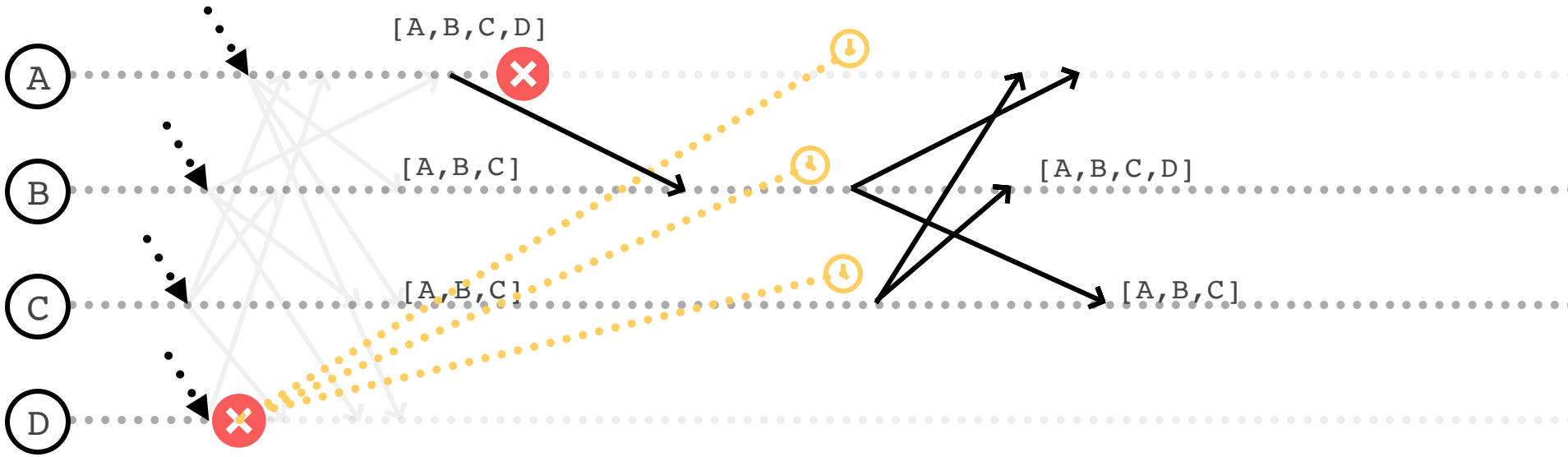
Deuxième round

Lorsque je n'ai plus rien à recevoir

Je lance un deuxième round, avec toutes les valeurs connues.

On est bon ?

Non ! Même problème !



Deuxième round

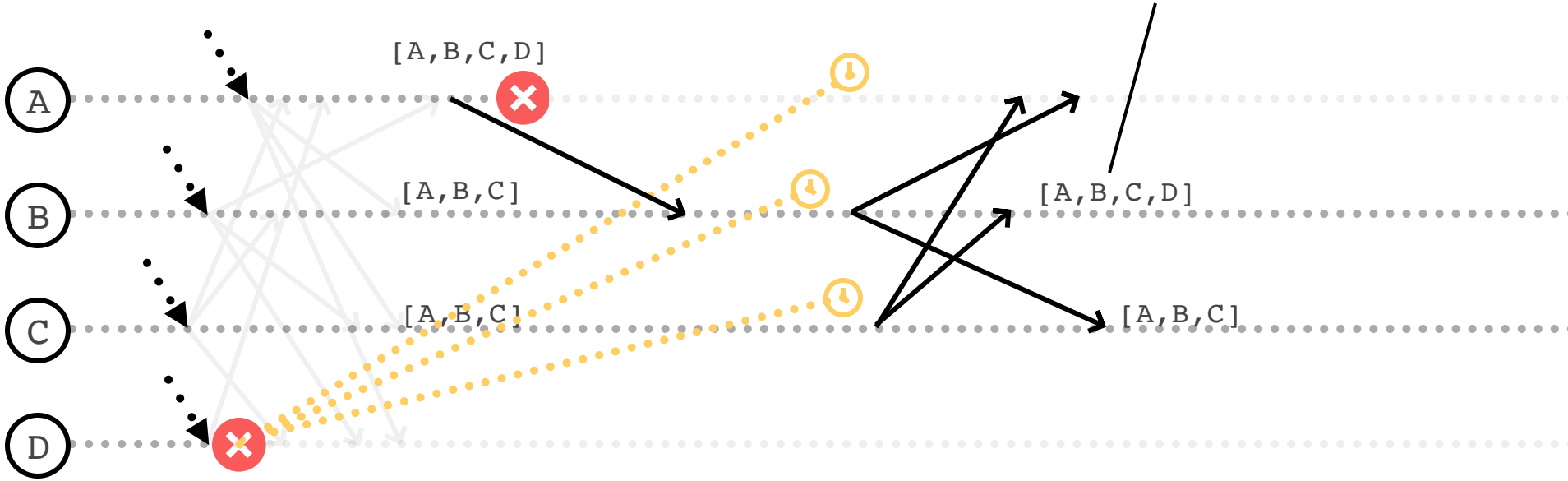
Lorsque je n'ai plus rien à recevoir

Je lance un deuxième round, avec toutes les valeurs connues.

On est bon ?

Non ! Même problème !

Seul `B` a reçu les données de `A`,
et donc celles de `D`.
`C` a encore une vision incomplète.



Deuxième round

Lorsque je n'ai plus rien à recevoir

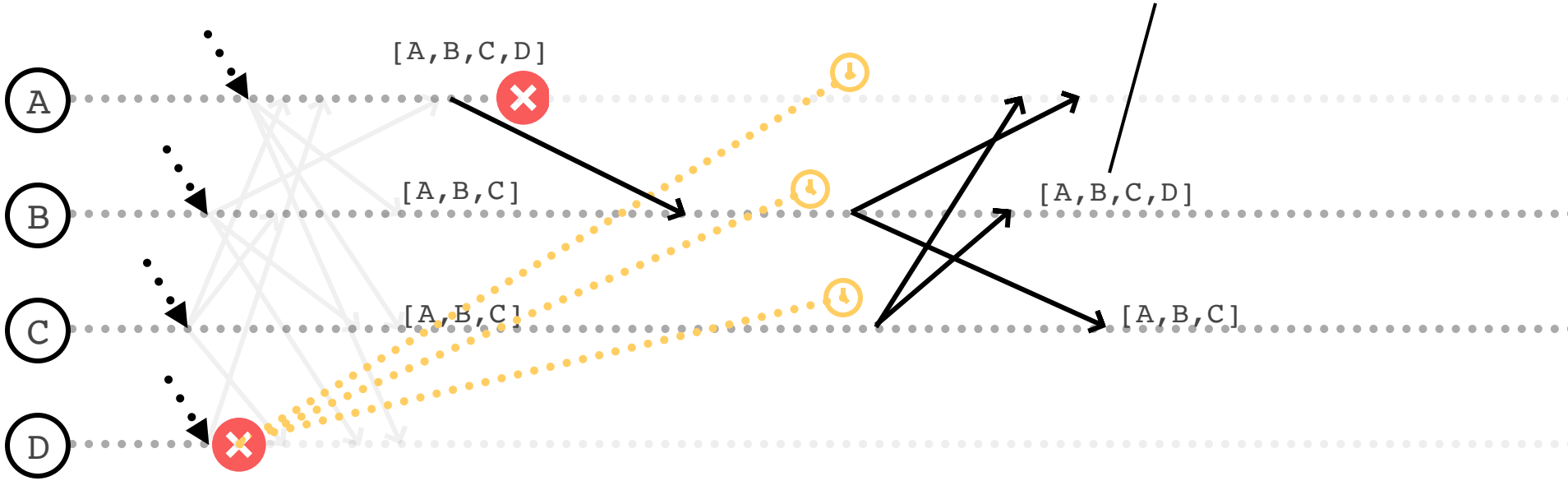
Je lance un deuxième round, avec toutes les valeurs connues.

On est bon ?

Non ! Même problème !

Solution ?

Seul `B` a reçu les données de `A`,
et donc celles de `D`.
`C` a encore une vision incomplète.



Deuxième round

Lorsque je n'ai plus rien à recevoir

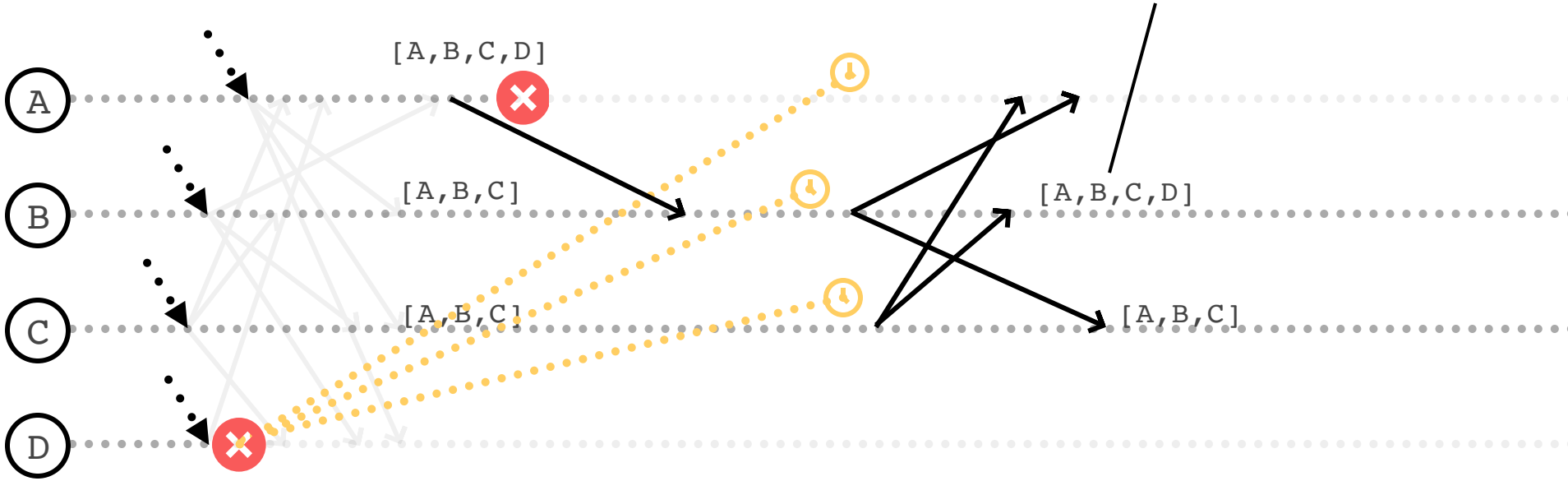
Je lance un deuxième round, avec toutes les valeurs connues.

On est bon ?

Non ! Même problème !

Solution ? *On refait pareil !*

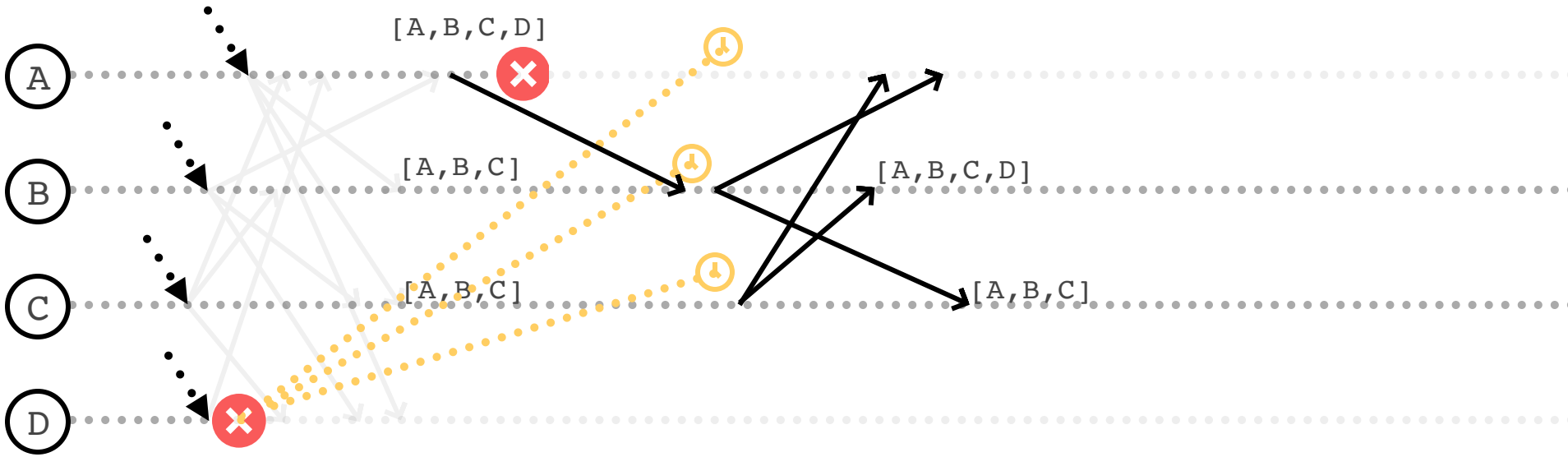
Seul `B` a reçu les données de `A`,
et donc celles de `D`.
`C` a encore une vision incomplète.



Plusieurs rounds

Quand j'ai la proposition de chaque processus non détecté

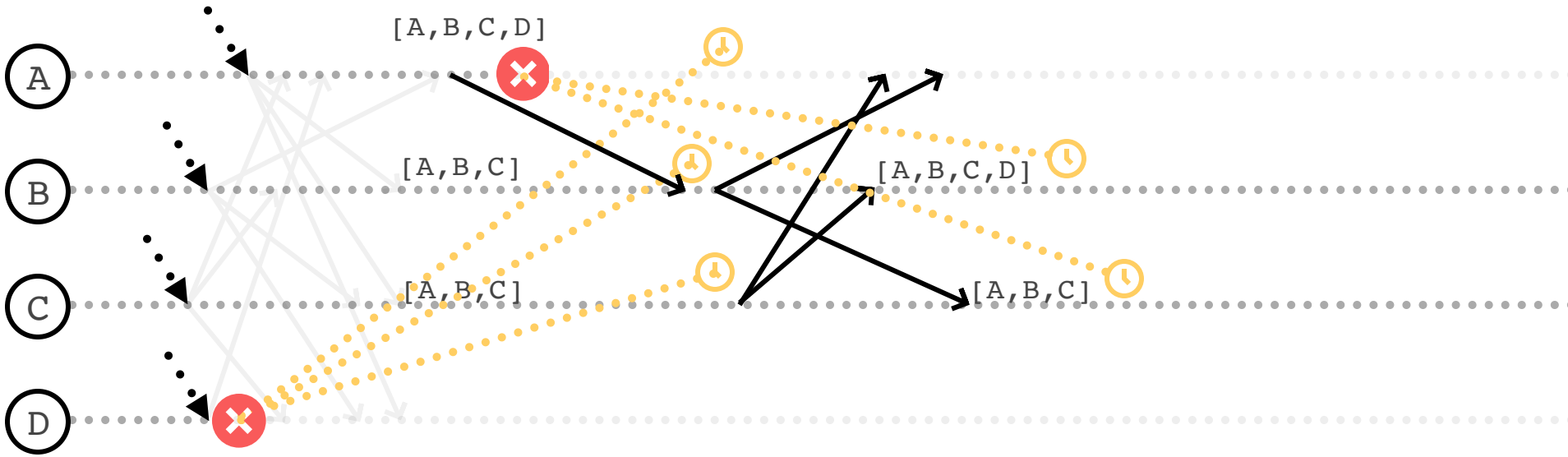
Je lance un nouveau round, avec toutes les valeurs connues.



Plusieurs rounds

Quand j'ai la proposition de chaque processus non détecté

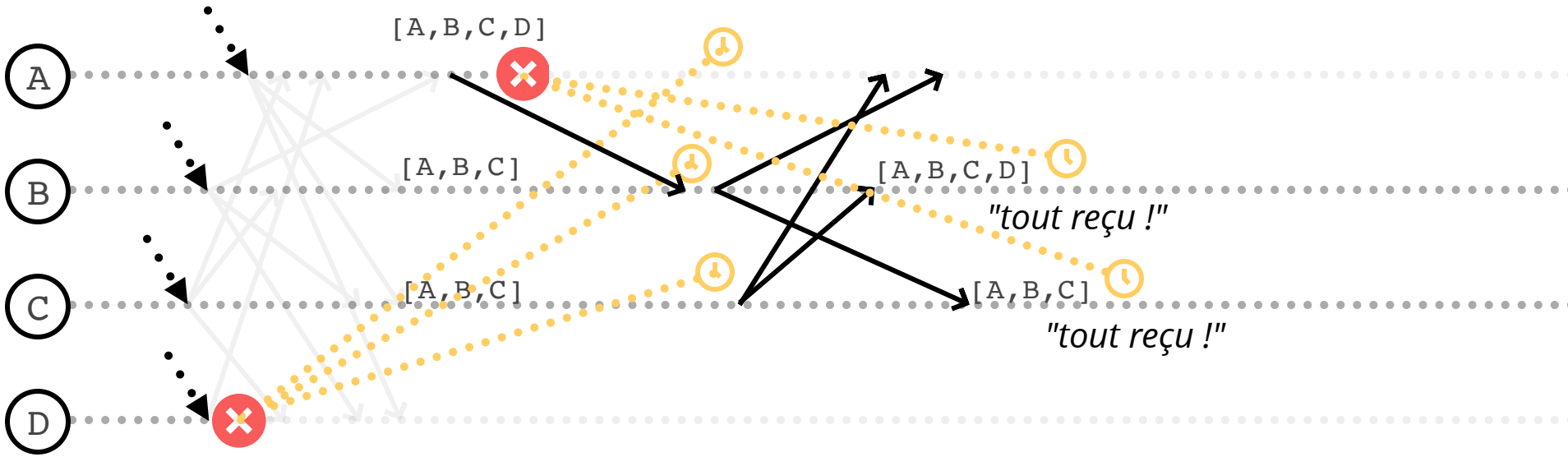
Je lance un nouveau round, avec toutes les valeurs connues.



Plusieurs rounds

Quand j'ai la proposition de chaque processus non détecté

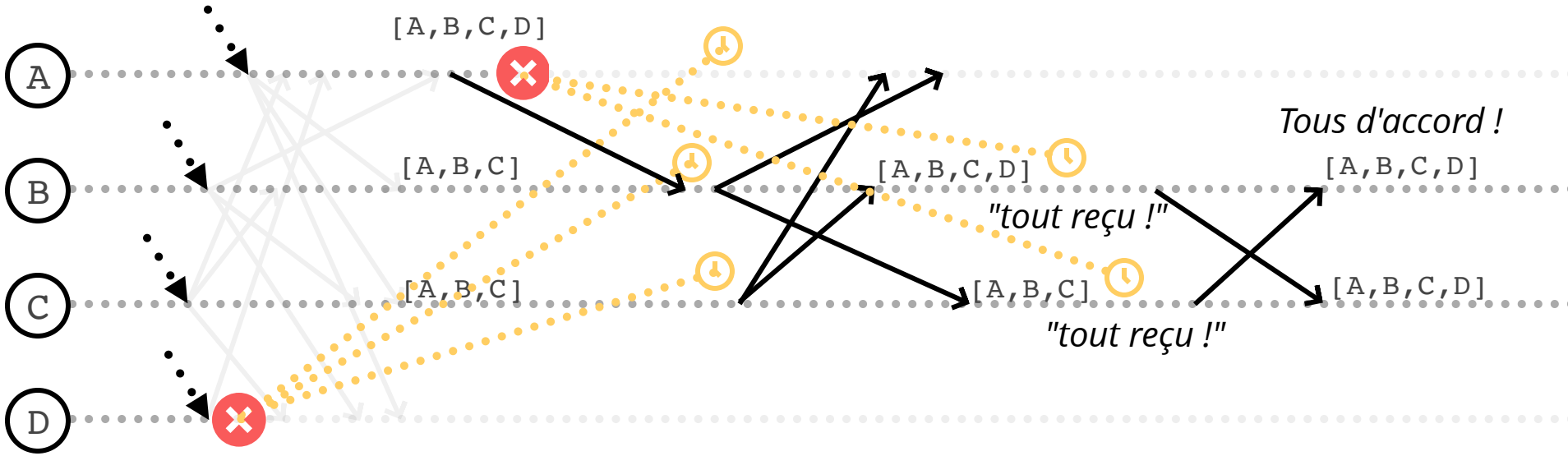
Je lance un nouveau round, avec toutes les valeurs connues.



Plusieurs rounds

Quand j'ai la proposition de chaque processus non détecté

Je lance un nouveau round, avec toutes les valeurs connues.

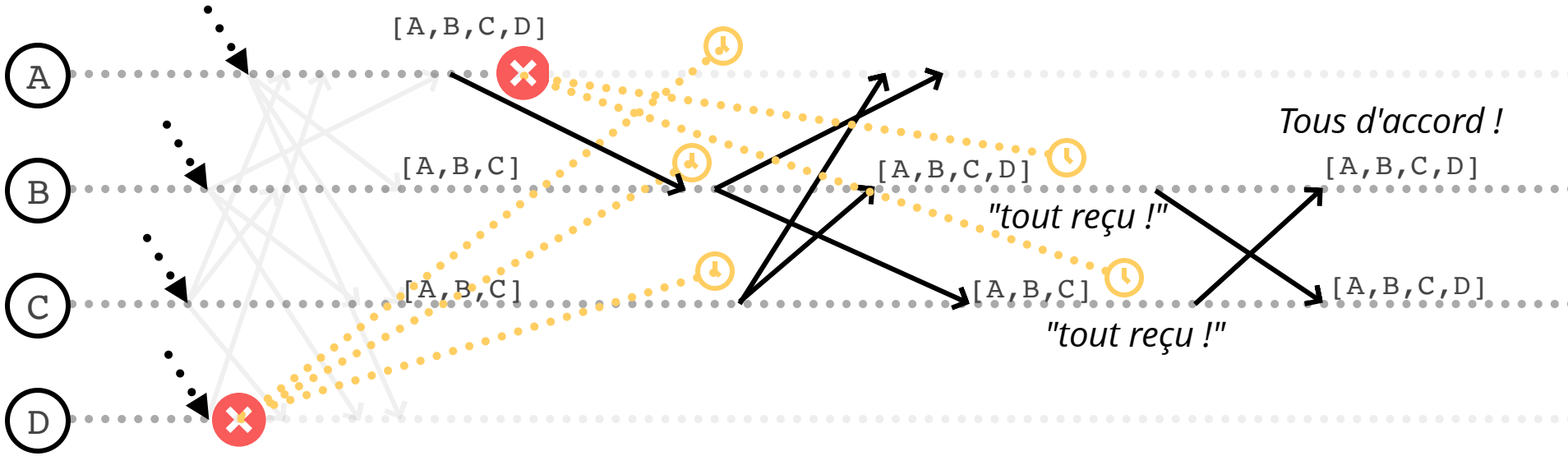


Plusieurs rounds

Quand j'ai la proposition de chaque processus non détecté

Je lance un nouveau round, avec toutes les valeurs connues.

Et maintenant, on est bon ?



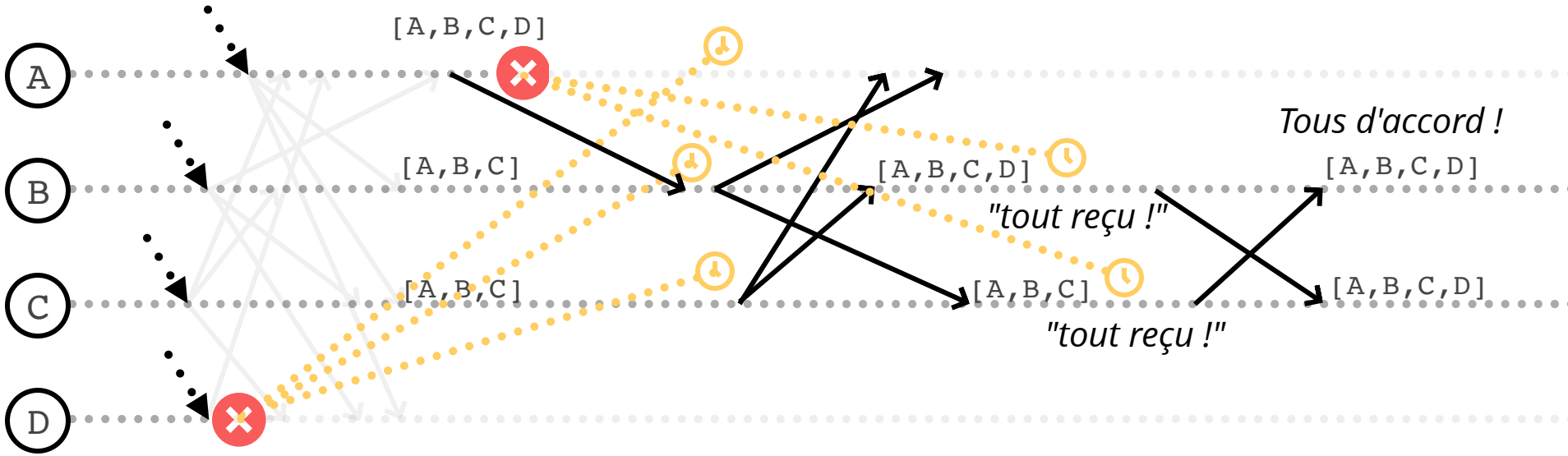
Plusieurs rounds

Quand j'ai la proposition de chaque processus non détecté

Je lance un nouveau round, avec toutes les valeurs connues.

Et maintenant, on est bon ?

Peut-être… Mais quand s'arrêter ?



Combien de rounds ?

Ou "quand s'arrêter" ?

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

Il peut prendre 2T à détecter ; chaque round devrait donc durer 2T.

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

Il peut prendre 2T à détecter ; chaque round devrait donc durer 2T.

Comment, alors ?

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

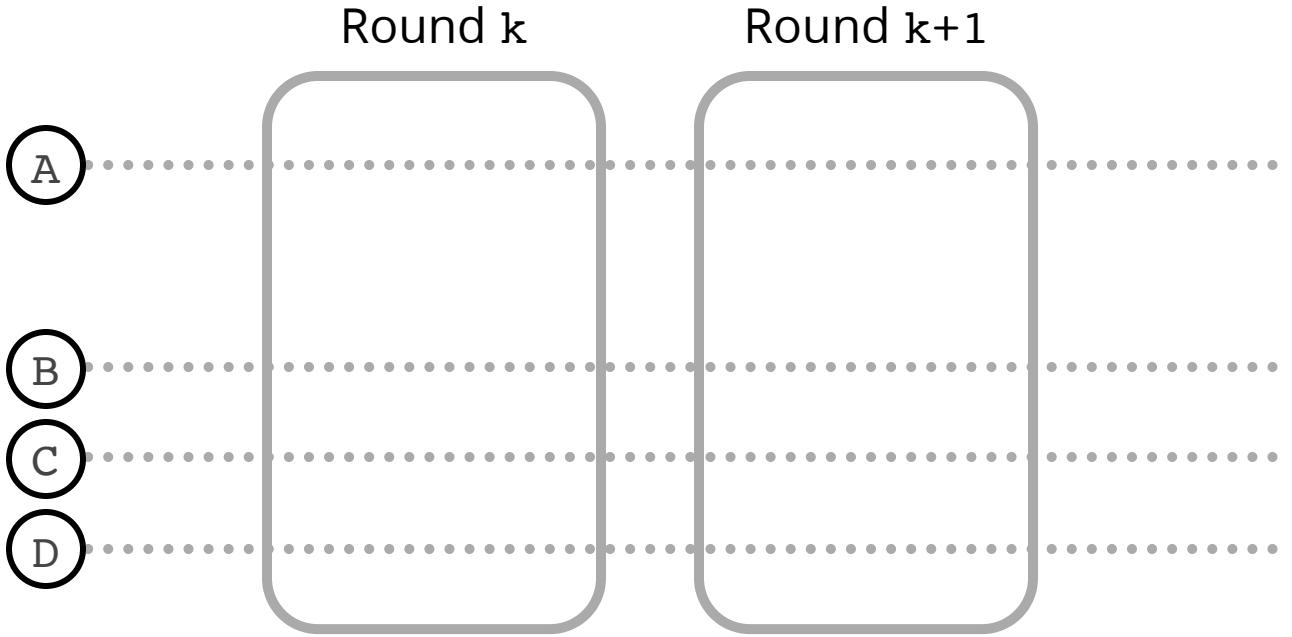
Il peut prendre 2T à détecter ; chaque round devrait donc durer 2T.

Comment, alors ?

> *Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.*

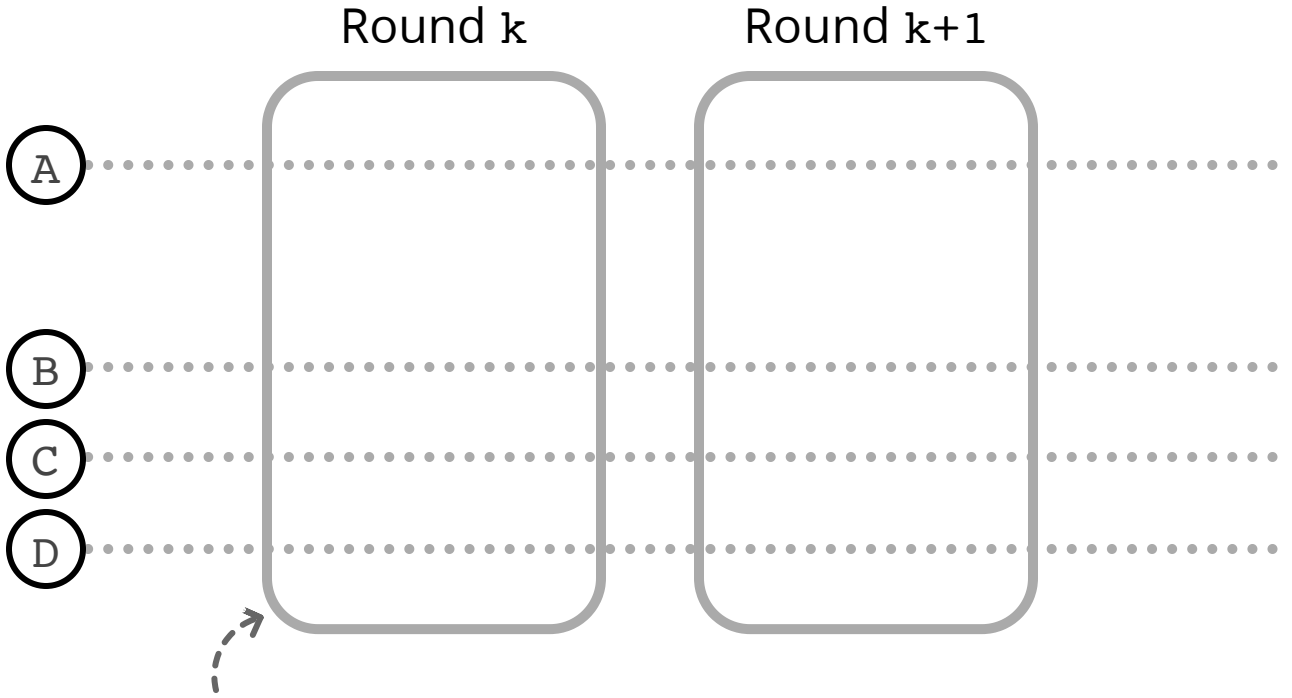
Quand s'arrêter

Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.



Quand s'arrêter

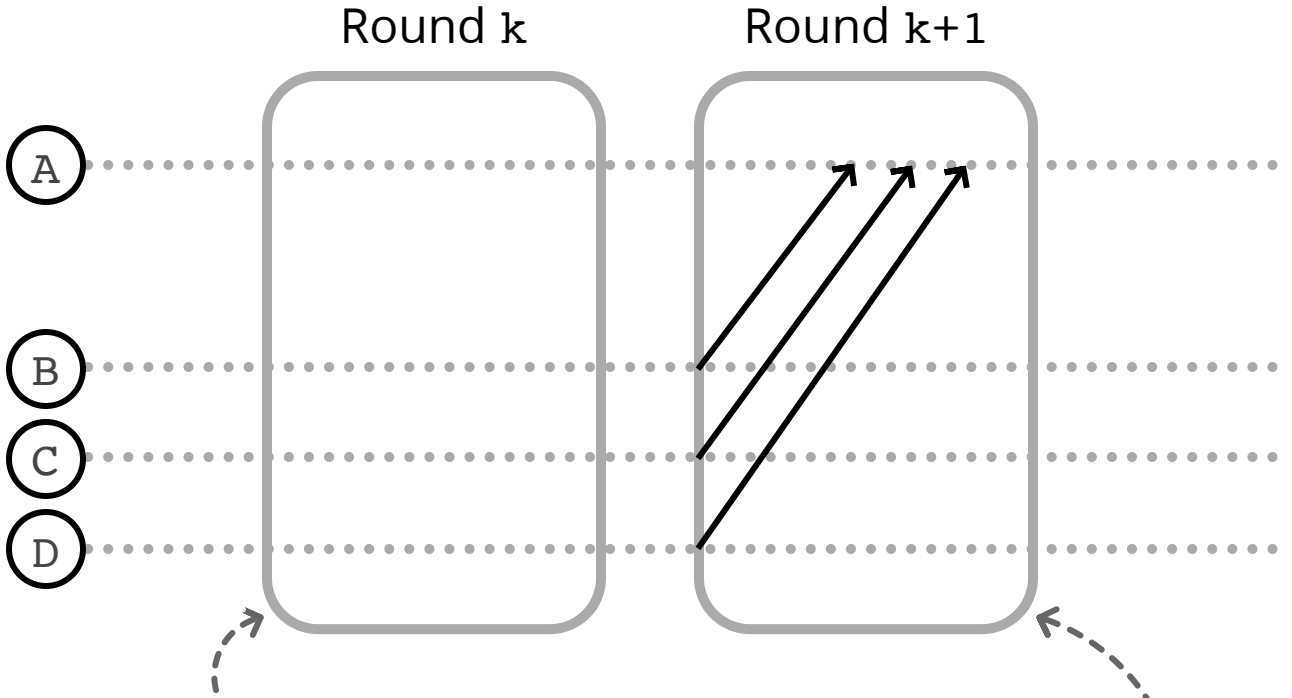
Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.



À la fin du round k, l'union de ce que tout le monde sait correspond à ce que je veux savoir

Quand s'arrêter

Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.



À la fin du round k, l'union de ce que tout le monde sait correspond à ce que je veux savoir, et est garantie de m'arriver durant le round k+1.

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

Il peut prendre 2T à détecter ; chaque round devrait donc durer 2T.

Comment, alors ?

> *Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.*

Donc je suis sûr d'avoir toutes les propositions.

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

Il peut prendre 2T à détecter ; chaque round devrait donc durer 2T.

Comment, alors ?

> *Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.*

Donc je suis sûr d'avoir toutes les propositions.

Est-ce que je peux m'arrêter, alors ?

Quand s'arrêter

Quand suis-je sûr d'avoir toutes les propositions ?

Quand il n'y a pas eu de pannes pendant le round précédent.

Utiliser le détecteur de pannes ?

Il peut prendre 2T à détecter ; chaque round devrait donc durer 2T.

Comment, alors ?

> *Si je reçois de la part des mêmes processus deux fois de suite, alors aucune panne n'a eu lieu entre temps.*

Donc je suis sûr d'avoir toutes les propositions.

Est-ce que je peux m'arrêter, alors ?

Pas encore, je suis peut-être le seul dans cette situation !

La raison pour laquelle un processus attend d'avoir deux rounds de suite à recevoir de la part des mêmes processus :

Regardons d'abord C :
- À la fin du round 1, C sait que A et B sont up, mais ne sait pas s'il y a un désaccord entre eux ; c'est possible. (et par désaccord j'entends que A et B ne possèdent pas le même ensemble de propositions).
- À la fin du round 2, C n'a pas reçu les propositions de A car il est tombé en panne. C ne peut donc pas être certain d'avoir toutes les propositions, car A avait peut-être des propositions différentes, et a peut-être eu le temps de les partager avec d'autres avant de tomber en panne. Il ne peut donc pas prendre de décision.

En revanche, regardons B :
- À la fin du round 1, comme pour C, B sait que A et C sont up, mais qu'ils sont peut-être en désaccord.
- À la fin du round 2, B a reçu les valeurs de A et de C ; ce désaccord potentiel entre A et C est donc résolu du point de vue de B, puisque B voit les deux ensembles de propositions.

B peut donc maintenant prétendre avoir une vision globale/complète des propositions "en lice" :
- S'il existe une propositions que B ne connait pas en fin de round 2, mais qu'un autre processus la connait, cela implique que cette proposition était connue par au moins un processus en fin de round 1 (celui qui l'a partagée durant le round 2).
- Pour que ce processus qui avait cette proposition en fin de round 1 ne l'ait pas envoyée à tout le monde durant le round 2, il faut qu'il soit tombé en panne durant l'envoi du round 2. Cela implique aussi qu'il était up durant le round 1, nécessairement.
- S'il était up durant le round 1, alors B a reçu ses messages durant le round 1. Or il n'a pas reçu son message durant le round 2, ce qui signifie que B n'a pas reçu les messages des mêmes envoyeurs durant les rounds 1 et 2, et qu'il ne peut donc pas prétendre avoir une vision complete.

C'est une preuve par contradiction : Si B a reçu des mêmes envoyeurs deux fois de suite, mais qu'il n'a pas une vision complete, alors B n'a pas reçu des mêmes envoyeurs deux fois de suite. L'erreur est forcément dans la partie "il n'a pas une vision complete" : donc si B a reçu des mêmes envoyeurs deux fois de suite, il a une vision complète.

La raison pour laquelle un processus attend d'avoir deux rounds de suite à recevoir de la part des mêmes processus :

Regardons d'abord C :
- À la fin du round 1, C sait que A et B sont up, mais ne sait pas s'il y a un désaccord entre eux ; c'est possible. (et par désaccord j'entends que A et B ne possèdent pas le même ensemble de propositions).
- À la fin du round 2, C n'a pas reçu les propositions de A car il est tombé en panne. C ne peut donc pas être certain d'avoir toutes les propositions, car A avait peut-être des propositions différentes, et a peut-être eu le temps de les partager avec d'autres avant de tomber en panne. Il ne peut donc pas prendre de décision.

En revanche, regardons B :
- À la fin du round 1, comme pour C, B sait que A et C sont up, mais qu'ils sont peut-être en désaccord.
- À la fin du round 2, B a reçu les valeurs de A et de C ; ce désaccord potentiel entre A et C est donc résolu du point de vue de B, puisque B voit les deux ensembles de propositions.

B peut donc maintenant prétendre avoir une vision globale/complète des propositions "en lice" :
- S'il existe une propositions que B ne connait pas en fin de round 2, mais qu'un autre processus la connait, cela implique que cette proposition était connue par au moins un processus en fin de round 1 (celui qui l'a partagée durant le round 2).
- Pour que ce processus qui avait cette proposition en fin de round 1 ne l'ait pas envoyée à tout le monde durant le round 2, il faut qu'il soit tombé en panne durant l'envoi du round 2. Cela implique aussi qu'il était up durant le round 1, nécessairement.
- S'il était up durant le round 1, alors B a reçu ses messages durant le round 1. Or il n'a pas reçu son message durant le round 2, ce qui signifie que B n'a pas reçu les messages des mêmes envoyeurs durant les rounds 1 et 2, et qu'il ne peut donc pas prétendre avoir une vision complete.

C'est une preuve par contradiction : Si B a reçu des mêmes envoyeurs deux fois de suite, mais qu'il n'a pas une vision complete, alors B n'a pas reçu des mêmes envoyeurs deux fois de suite. L'erreur est forcément dans la partie "il n'a pas une vision complete" : donc si B a reçu des mêmes envoyeurs deux fois de suite, il a une vision complète.

La raison pour laquelle un processus attend d'avoir deux rounds de suite à recevoir de la part des mêmes processus :

Regardons d'abord C :
- À la fin du round 1, C sait que A et B sont up, mais ne sait pas s'il y a un désaccord entre eux ; c'est possible. (et par désaccord j'entends que A et B ne possèdent pas le même ensemble de propositions).
- À la fin du round 2, C n'a pas reçu les propositions de A car il est tombé en panne. C ne peut donc pas être certain d'avoir toutes les propositions, car A avait peut-être des propositions différentes, et a peut-être eu le temps de les partager avec d'autres avant de tomber en panne. Il ne peut donc pas prendre de décision.

En revanche, regardons B :
- À la fin du round 1, comme pour C, B sait que A et C sont up, mais qu'ils sont peut-être en désaccord.
- À la fin du round 2, B a reçu les valeurs de A et de C ; ce désaccord potentiel entre A et C est donc résolu du point de vue de B, puisque B voit les deux ensembles de propositions.

B peut donc maintenant prétendre avoir une vision globale/complète des propositions "en lice" :
- S'il existe une propositions que B ne connait pas en fin de round 2, mais qu'un autre processus la connait, cela implique que cette proposition était connue par au moins un processus en fin de round 1 (celui qui l'a partagée durant le round 2).
- Pour que ce processus qui avait cette proposition en fin de round 1 ne l'ait pas envoyée à tout le monde durant le round 2, il faut qu'il soit tombé en panne durant l'envoi du round 2. Cela implique aussi qu'il était up durant le round 1, nécessairement.
- S'il était up durant le round 1, alors B a reçu ses messages durant le round 1. Or il n'a pas reçu son message durant le round 2, ce qui signifie que B n'a pas reçu les messages des mêmes envoyeurs durant les rounds 1 et 2, et qu'il ne peut donc pas prétendre avoir une vision complete.

C'est une preuve par contradiction : Si B a reçu des mêmes envoyeurs deux fois de suite, mais qu'il n'a pas une vision complete, alors B n'a pas reçu des mêmes envoyeurs deux fois de suite. L'erreur est forcément dans la partie "il n'a pas une vision complete" : donc si B a reçu des mêmes envoyeurs deux fois de suite, il a une vision complète.

La raison pour laquelle un processus attend d'avoir deux rounds de suite à recevoir de la part des mêmes processus :

Regardons d'abord C :
- À la fin du round 1, C sait que A et B sont up, mais ne sait pas s'il y a un désaccord entre eux ; c'est possible. (et par désaccord j'entends que A et B ne possèdent pas le même ensemble de propositions).
- À la fin du round 2, C n'a pas reçu les propositions de A car il est tombé en panne. C ne peut donc pas être certain d'avoir toutes les propositions, car A avait peut-être des propositions différentes, et a peut-être eu le temps de les partager avec d'autres avant de tomber en panne. Il ne peut donc pas prendre de décision.

En revanche, regardons B :
- À la fin du round 1, comme pour C, B sait que A et C sont up, mais qu'ils sont peut-être en désaccord.
- À la fin du round 2, B a reçu les valeurs de A et de C ; ce désaccord potentiel entre A et C est donc résolu du point de vue de B, puisque B voit les deux ensembles de propositions.

B peut donc maintenant prétendre avoir une vision globale/complète des propositions "en lice" :
- S'il existe une propositions que B ne connait pas en fin de round 2, mais qu'un autre processus la connait, cela implique que cette proposition était connue par au moins un processus en fin de round 1 (celui qui l'a partagée durant le round 2).
- Pour que ce processus qui avait cette proposition en fin de round 1 ne l'ait pas envoyée à tout le monde durant le round 2, il faut qu'il soit tombé en panne durant l'envoi du round 2. Cela implique aussi qu'il était up durant le round 1, nécessairement.
- S'il était up durant le round 1, alors B a reçu ses messages durant le round 1. Or il n'a pas reçu son message durant le round 2, ce qui signifie que B n'a pas reçu les messages des mêmes envoyeurs durant les rounds 1 et 2, et qu'il ne peut donc pas prétendre avoir une vision complete.

C'est une preuve par contradiction : Si B a reçu des mêmes envoyeurs deux fois de suite, mais qu'il n'a pas une vision complete, alors B n'a pas reçu des mêmes envoyeurs deux fois de suite. L'erreur est forcément dans la partie "il n'a pas une vision complete" : donc si B a reçu des mêmes envoyeurs deux fois de suite, il a une vision complète.

La raison pour laquelle un processus attend d'avoir deux rounds de suite à recevoir de la part des mêmes processus :

Regardons d'abord C :
- À la fin du round 1, C sait que A et B sont up, mais ne sait pas s'il y a un désaccord entre eux ; c'est possible. (et par désaccord j'entends que A et B ne possèdent pas le même ensemble de propositions).
- À la fin du round 2, C n'a pas reçu les propositions de A car il est tombé en panne. C ne peut donc pas être certain d'avoir toutes les propositions, car A avait peut-être des propositions différentes, et a peut-être eu le temps de les partager avec d'autres avant de tomber en panne. Il ne peut donc pas prendre de décision.

En revanche, regardons B :
- À la fin du round 1, comme pour C, B sait que A et C sont up, mais qu'ils sont peut-être en désaccord.
- À la fin du round 2, B a reçu les valeurs de A et de C ; ce désaccord potentiel entre A et C est donc résolu du point de vue de B, puisque B voit les deux ensembles de propositions.

B peut donc maintenant prétendre avoir une vision globale/complète des propositions "en lice" :
- S'il existe une propositions que B ne connait pas en fin de round 2, mais qu'un autre processus la connait, cela implique que cette proposition était connue par au moins un processus en fin de round 1 (celui qui l'a partagée durant le round 2).
- Pour que ce processus qui avait cette proposition en fin de round 1 ne l'ait pas envoyée à tout le monde durant le round 2, il faut qu'il soit tombé en panne durant l'envoi du round 2. Cela implique aussi qu'il était up durant le round 1, nécessairement.
- S'il était up durant le round 1, alors B a reçu ses messages durant le round 1. Or il n'a pas reçu son message durant le round 2, ce qui signifie que B n'a pas reçu les messages des mêmes envoyeurs durant les rounds 1 et 2, et qu'il ne peut donc pas prétendre avoir une vision complete.

C'est une preuve par contradiction : Si B a reçu des mêmes envoyeurs deux fois de suite, mais qu'il n'a pas une vision complete, alors B n'a pas reçu des mêmes envoyeurs deux fois de suite. L'erreur est forcément dans la partie "il n'a pas une vision complete" : donc si B a reçu des mêmes envoyeurs deux fois de suite, il a une vision complète.

La raison pour laquelle un processus attend d'avoir deux rounds de suite à recevoir de la part des mêmes processus :

Regardons d'abord C :
- À la fin du round 1, C sait que A et B sont up, mais ne sait pas s'il y a un désaccord entre eux ; c'est possible. (et par désaccord j'entends que A et B ne possèdent pas le même ensemble de propositions).
- À la fin du round 2, C n'a pas reçu les propositions de A car il est tombé en panne. C ne peut donc pas être certain d'avoir toutes les propositions, car A avait peut-être des propositions différentes, et a peut-être eu le temps de les partager avec d'autres avant de tomber en panne. Il ne peut donc pas prendre de décision.

En revanche, regardons B :
- À la fin du round 1, comme pour C, B sait que A et C sont up, mais qu'ils sont peut-être en désaccord.
- À la fin du round 2, B a reçu les valeurs de A et de C ; ce désaccord potentiel entre A et C est donc résolu du point de vue de B, puisque B voit les deux ensembles de propositions.

B peut donc maintenant prétendre avoir une vision globale/complète des propositions "en lice" :
- S'il existe une propositions que B ne connait pas en fin de round 2, mais qu'un autre processus la connait, cela implique que cette proposition était connue par au moins un processus en fin de round 1 (celui qui l'a partagée durant le round 2).
- Pour que ce processus qui avait cette proposition en fin de round 1 ne l'ait pas envoyée à tout le monde durant le round 2, il faut qu'il soit tombé en panne durant l'envoi du round 2. Cela implique aussi qu'il était up durant le round 1, nécessairement.
- S'il était up durant le round 1, alors B a reçu ses messages durant le round 1. Or il n'a pas reçu son message durant le round 2, ce qui signifie que B n'a pas reçu les messages des mêmes envoyeurs durant les rounds 1 et 2, et qu'il ne peut donc pas prétendre avoir une vision complete.

C'est une preuve par contradiction : Si B a reçu des mêmes envoyeurs deux fois de suite, mais qu'il n'a pas une vision complete, alors B n'a pas reçu des mêmes envoyeurs deux fois de suite. L'erreur est forcément dans la partie "il n'a pas une vision complete" : donc si B a reçu des mêmes envoyeurs deux fois de suite, il a une vision complète.

Résumé intermédiaire

Jusqu'au "partage de décision"

Résumé intermédiaire

Résumé intermédiaire

Initialement

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.

J'entre en round 1.

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.
Sinon, je mets à jour ma liste de propositions.

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.

J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.

Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.

J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.

Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Sinon,

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.
Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,
Je décide parmi mes propositions *(de manière déterministe).*

Sinon,

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.
Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,
    Je décide parmi mes propositions *(de manière déterministe).*
    J'envoie ma décision à tous les processus en ligne, et je m'arrête.
Sinon,

Résumé intermédiaire

Initialement

Ma liste de propositions ne contient que la mienne.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.
Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,
    Je décide parmi mes propositions *(de manière déterministe).*
    J'envoie ma décision à tous les processus en ligne, et je m'arrête.
Sinon,
    Je passe au round suivant.

Phase de décision

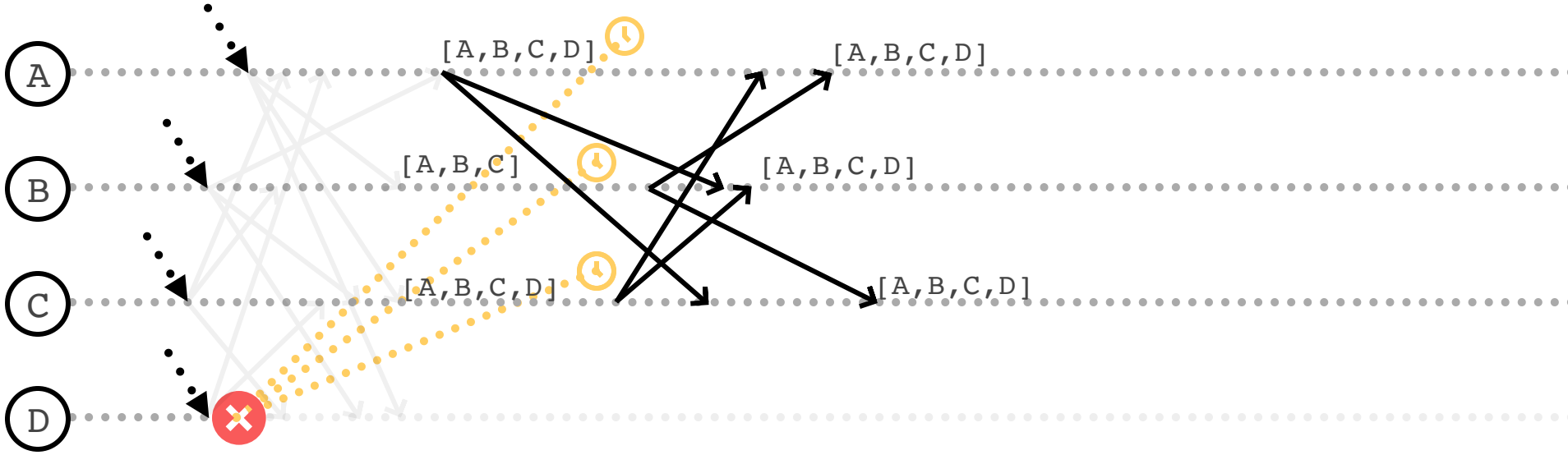
Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.



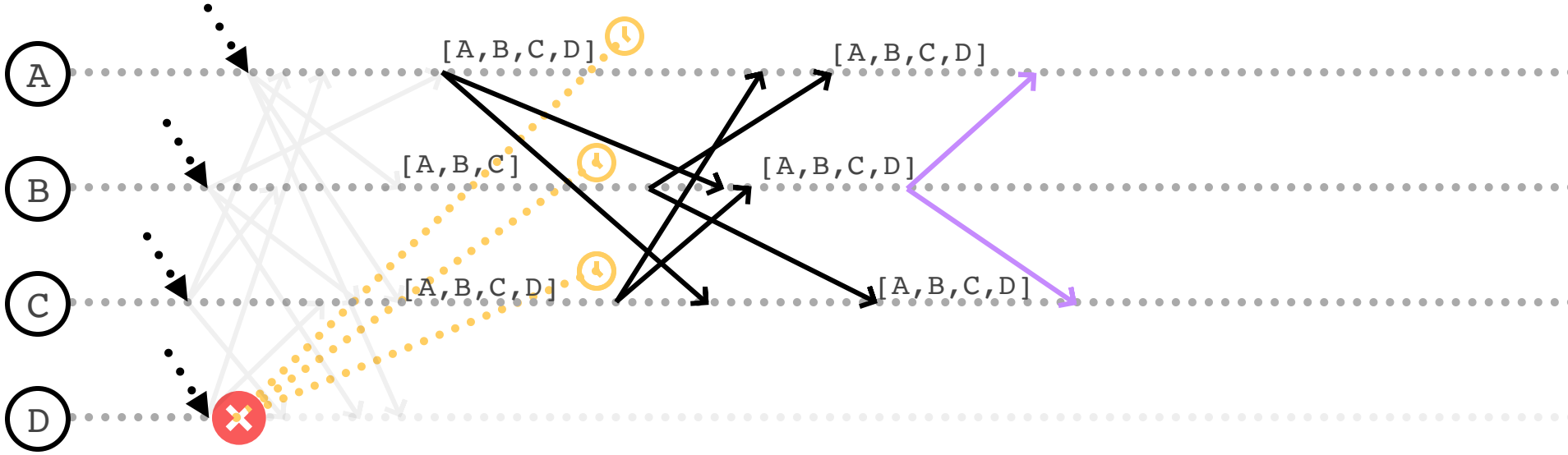
Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.



Phase de décision

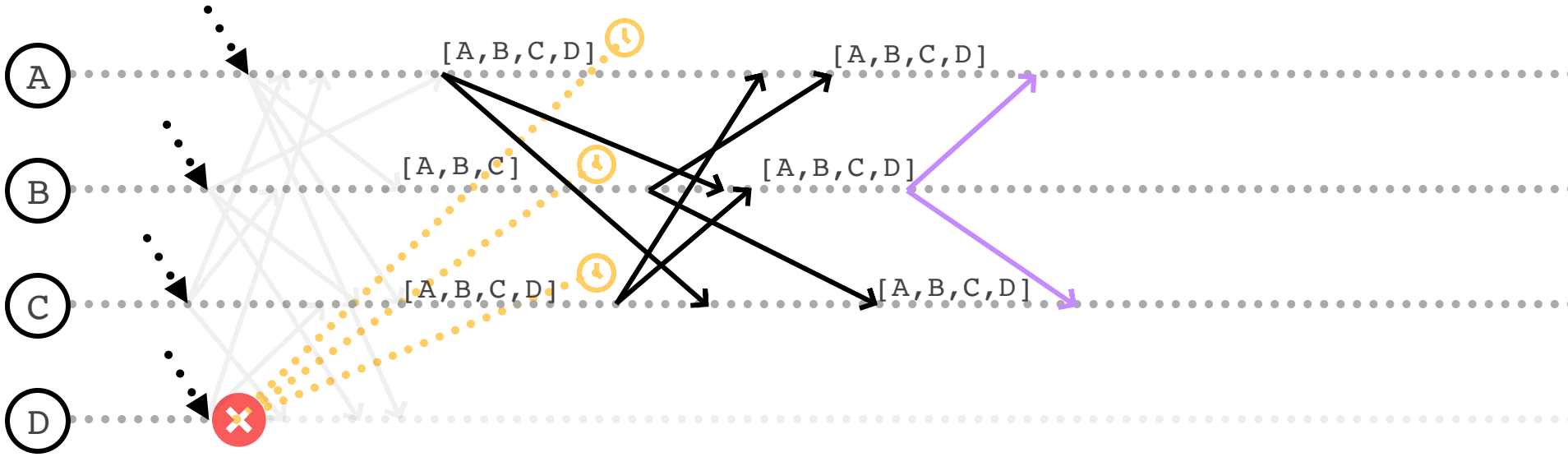
Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision



Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

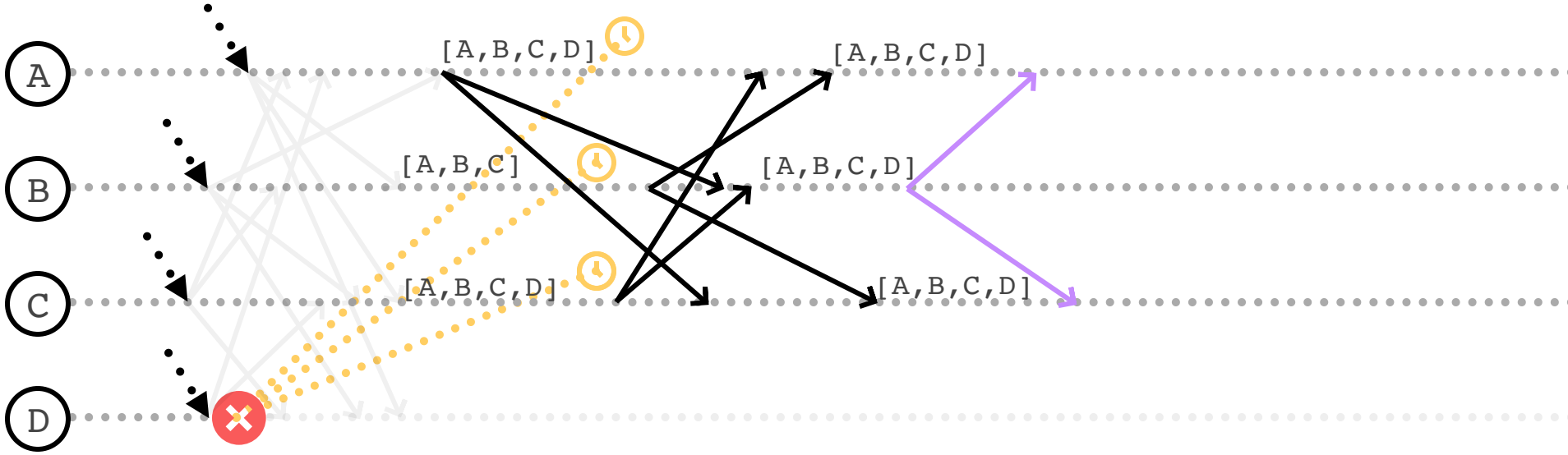
Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision

Je note et je m'arrête ?



Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

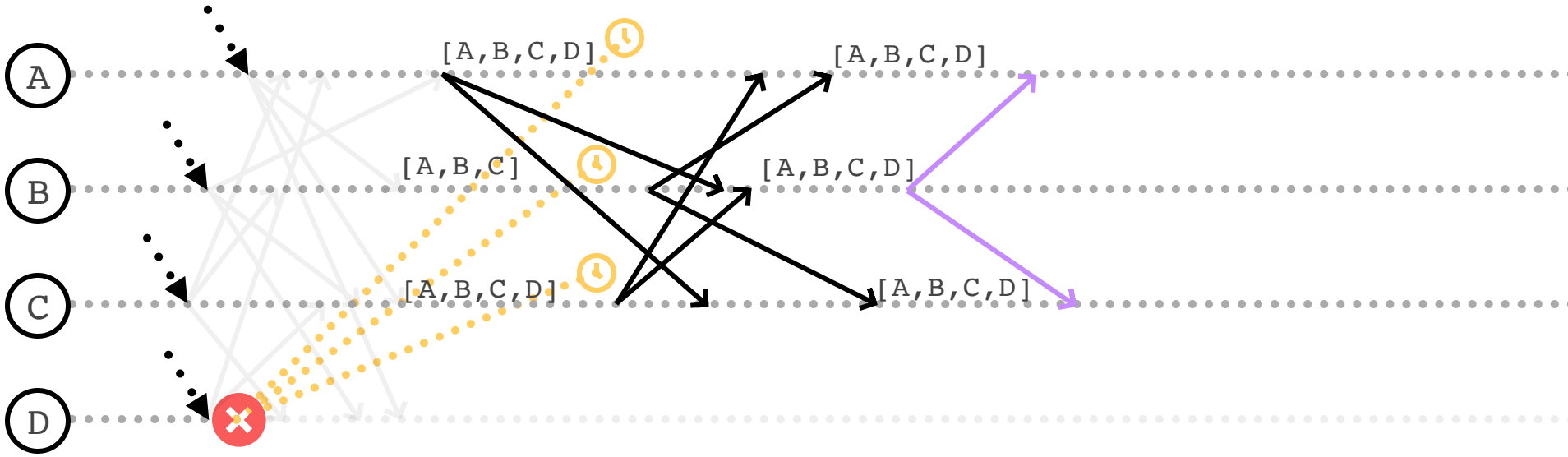
Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision

Je note et je m'arrête ? *Non ! Même risque qu'avant*



Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

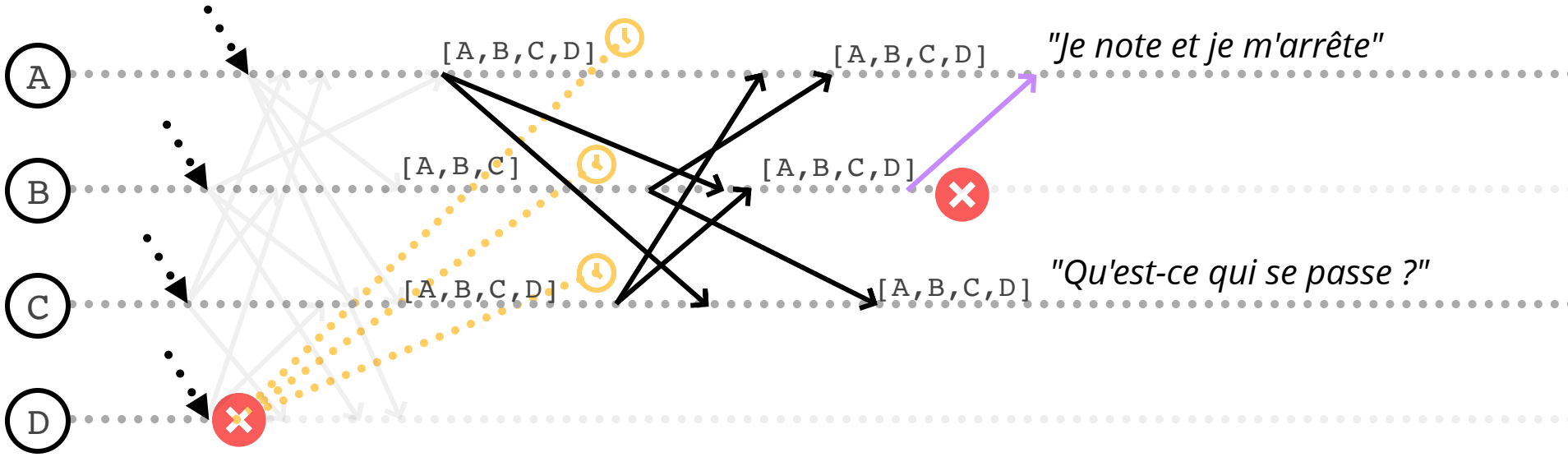
Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision

Je note et je m'arrête ? *Non ! Même risque qu'avant*



Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

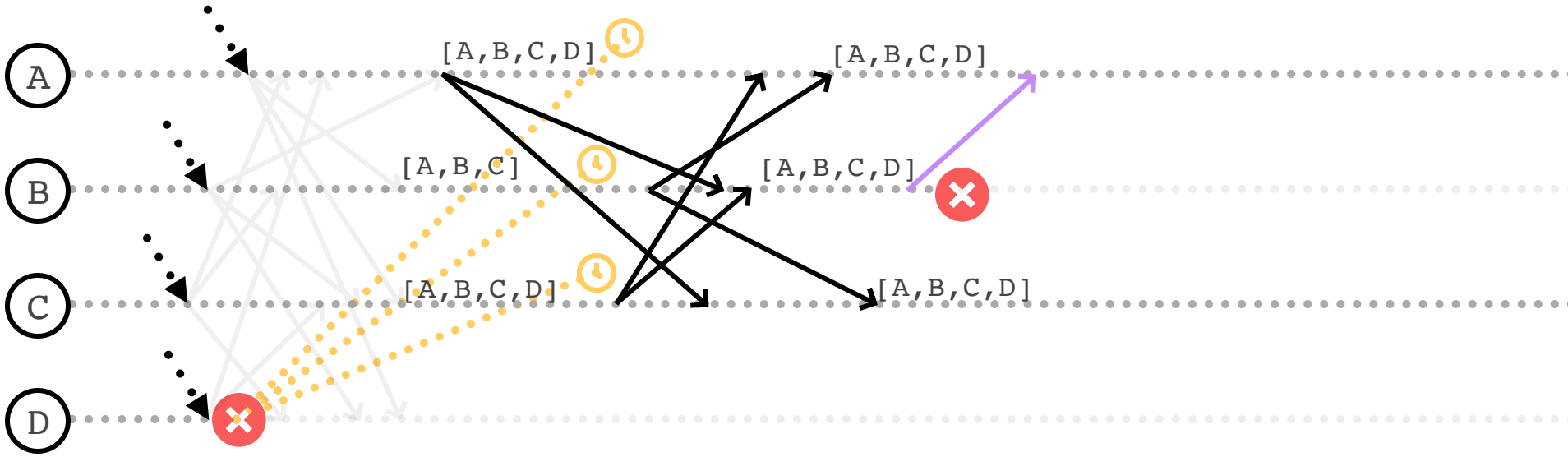
Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision

Je *propage*, et je m'arrête.



Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

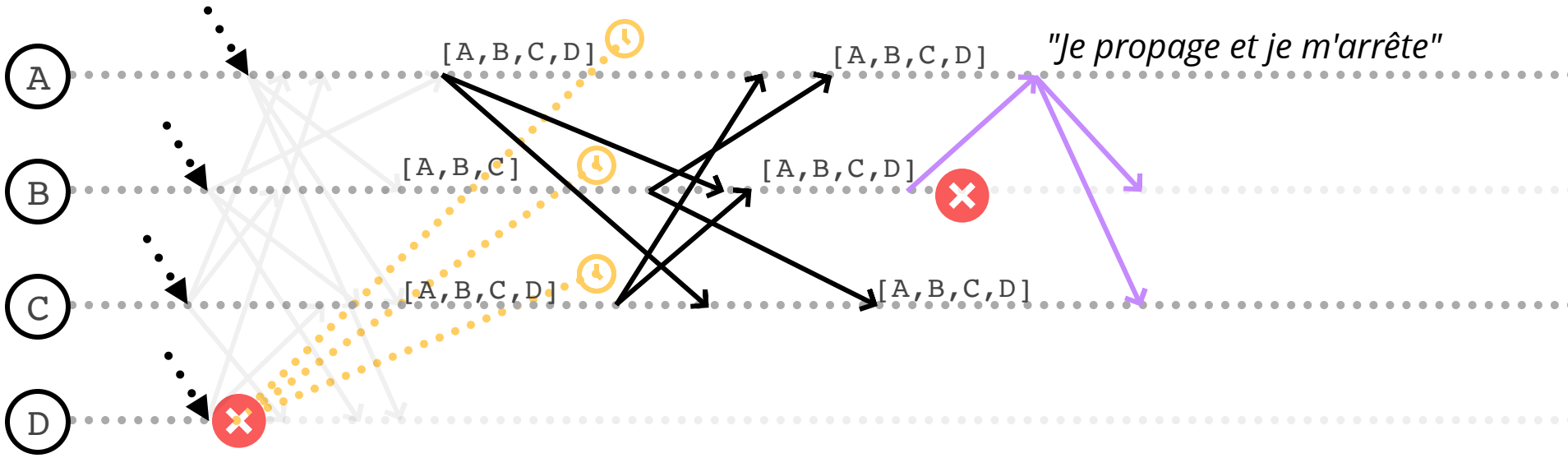
Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision

Je *propage*, et je m'arrête.



Phase de décision

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

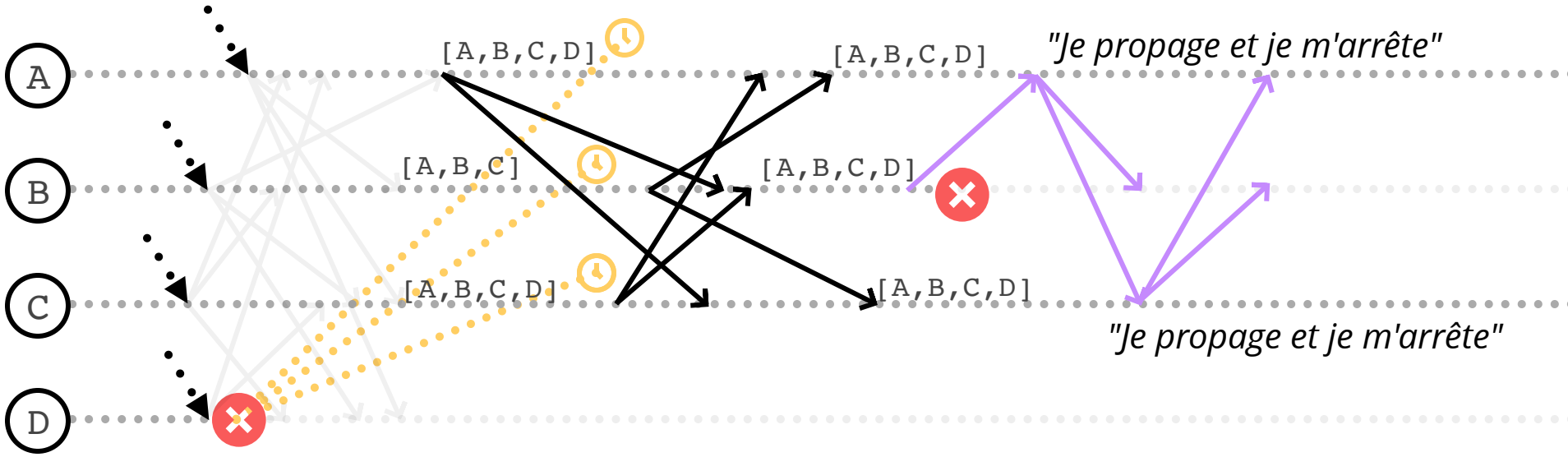
Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,

Je décide parmi mes propositions *(de manière déterministe).*

J'envoie ma décision à tous les processus en ligne, et je m'arrête.

Quand je reçois une décision

Je *propage*, et je m'arrête.



Phase de décision

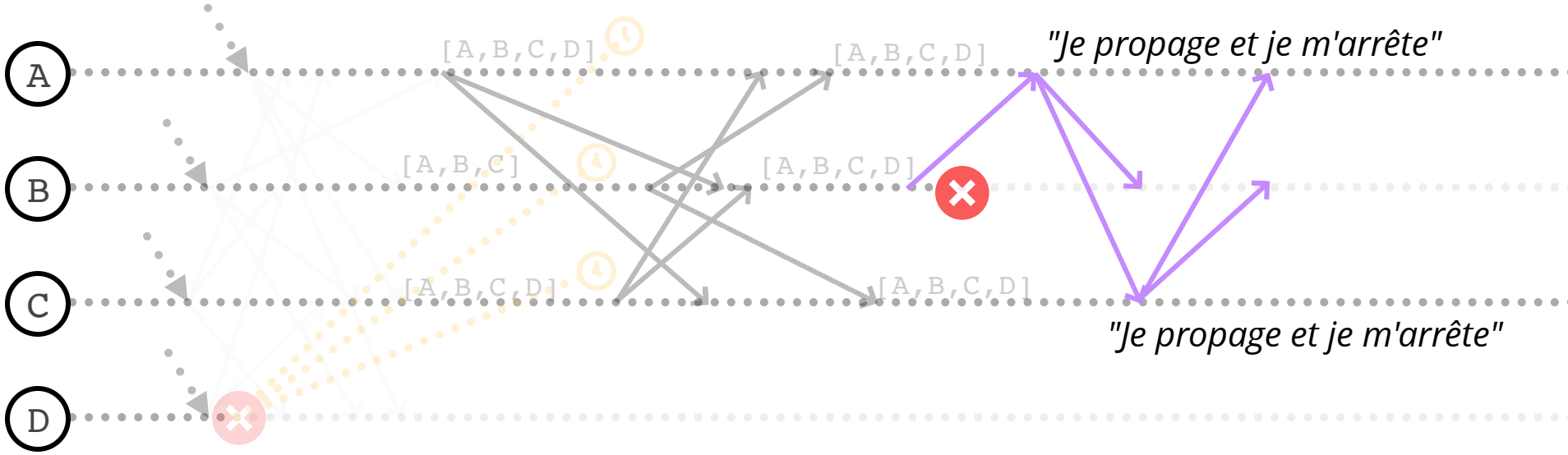
Remarque

L'approche utilisée ici,

"À la réception d'un broadcast, je broadcast à mon tour"

n'est pas spécifique au consensus.

Elle permet un broadcast sûr dans tout contexte de broadcast.



Résumé complet

et analyse de tolérance aux pannes

Résumé

Initialement

Ma liste ne contient que ma proposition.
J'entre en round 1.

Quand je reçois une décision

Je *propage*, et je m'arrête.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.
Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,
    Je décide parmi mes propositions *(de manière déterministe).*
    J'envoie ma décision à tous les processus en ligne, et je m'arrête.
Sinon,
    Je passe au round suivant.

Résumé

Initialement

Ma liste ne contient que ma proposition.
J'entre en round 1.

Quand j'entre en round `x`

J'envoie ma liste de propositions avec l'id de round.

Quand je reçois un message pour le round `x`

Si je ne suis pas encore en round `x`, je repousse sa réception.
Sinon, je mets à jour ma liste de propositions.

Quand j'ai reçu un message de tous ceux dont je n'ai pas détecté la panne

Si j'ai reçu, ce round, de la part des mêmes qu'au round précédent,
    Je décide parmi mes propositions *(de manière déterministe).*
    J'envoie ma décision à tous les processus en ligne, et je m'arrête.
Sinon,
    Je passe au round suivant.

Quand je reçois une décision

Je *propage*, et je m'arrête.

Tolérance aux pannes

Combien de pannes puis-je tolérer ?

Tolérance aux pannes

Combien de pannes puis-je tolérer ?

Jusqu'à N-1 (!)

C'est excellent. En général, difficile de tolérer plus qu'une constante, ou N/2-1.

Consensus IRL

Un problème central

Un problème central

Tout type d'information peut être proposé

Un problème central

Tout type d'information peut être proposé

"Qui de nous sera l'élu ?"

Un problème central

Tout type d'information peut être proposé

"Qui de nous sera l'élu ?"
"Qui de nous aura la SC ?"

Un problème central

Tout type d'information peut être proposé

"Qui de nous sera l'élu ?"
"Qui de nous aura la SC ?"

Voire même
"Quelle modification fait-on ensuite de l'état global"

Un problème central

Tout type d'information peut être proposé

"Qui de nous sera l'élu ?"
"Qui de nous aura la SC ?"

Voire même
"Quelle modification fait-on ensuite de l'état global"

Ou de manière plus abstraite
*"Quelle transition applique-t-on sur **une machine d'état partagée**"*

Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

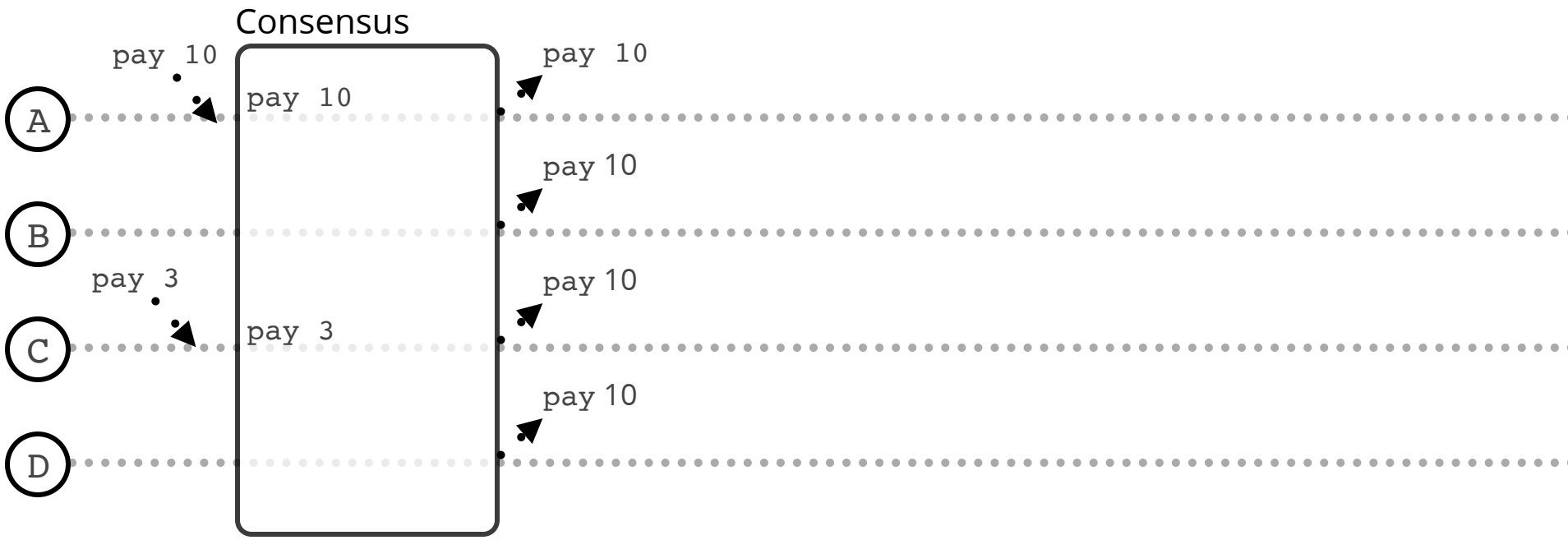
Si je souhaite modifier l'état, je lance un consensus et propose la modification

Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

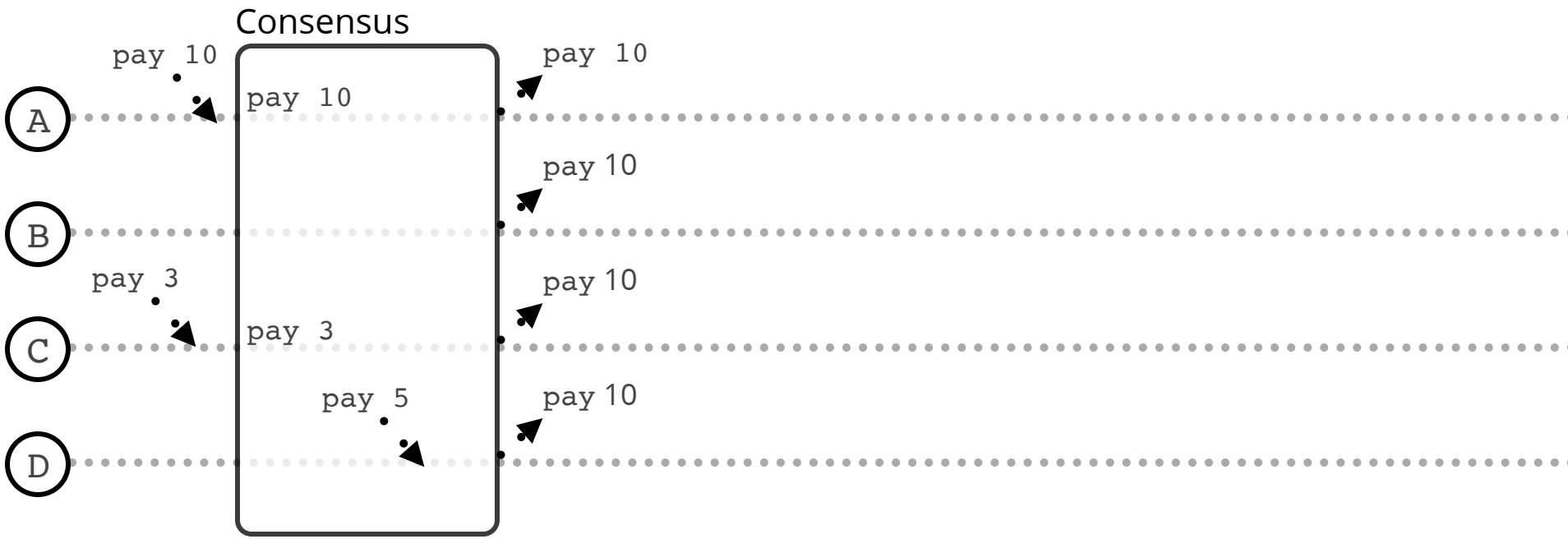


Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification



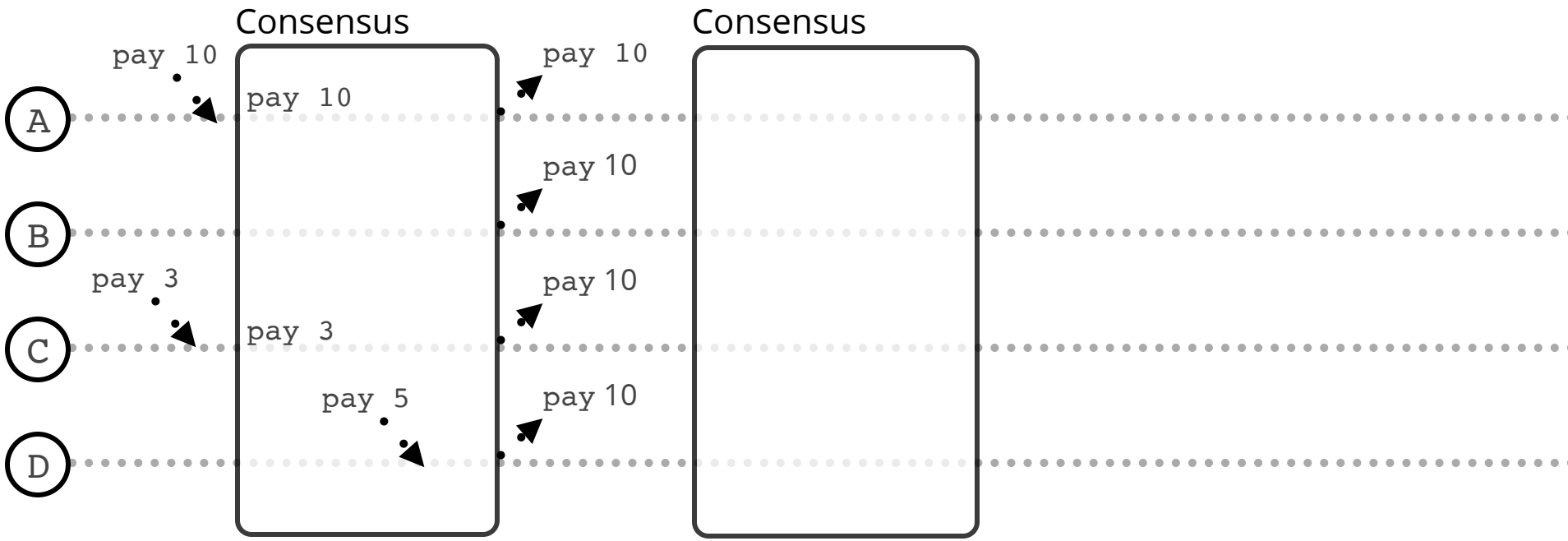
Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Tant que ma proposition n'a pas été acceptée, je recommence.



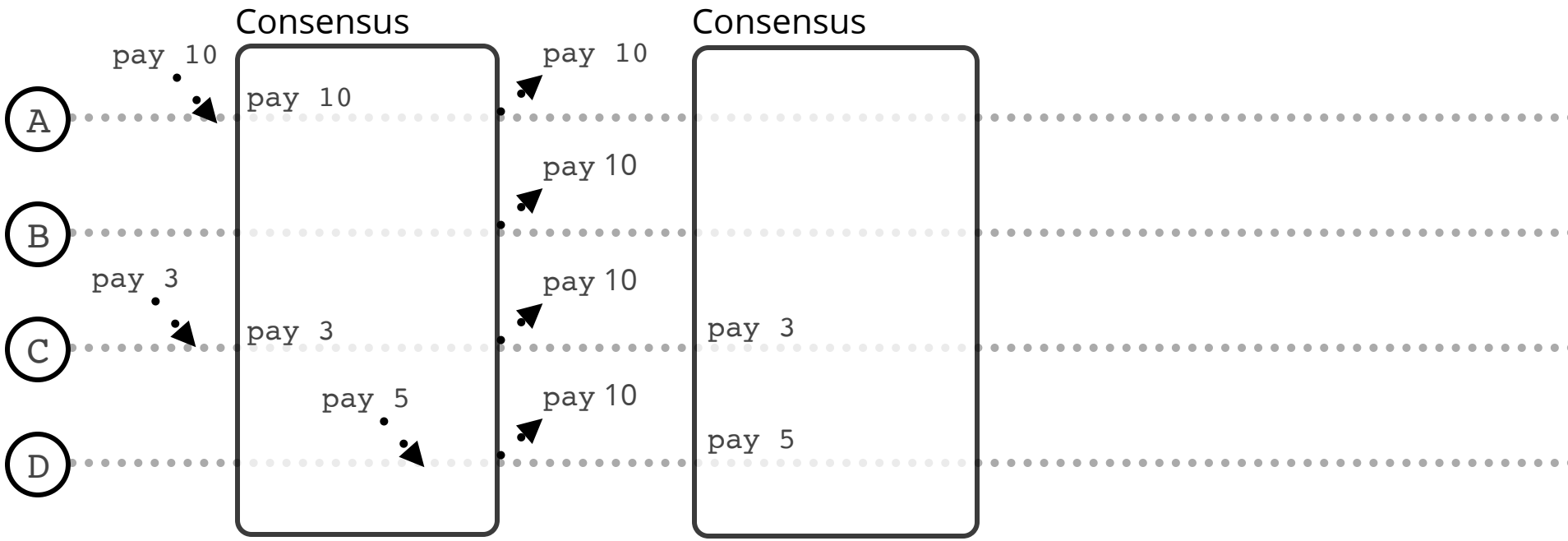
Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Tant que ma proposition n'a pas été acceptée, je recommence.



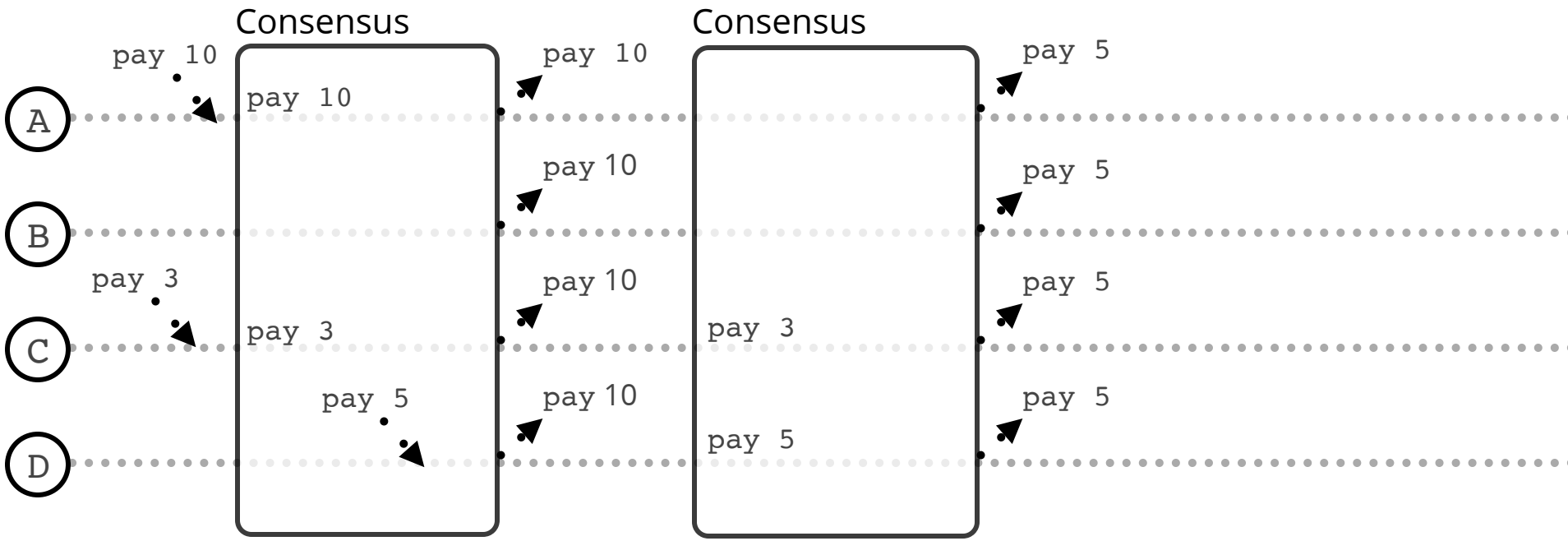
Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Tant que ma proposition n'a pas été acceptée, je recommence.



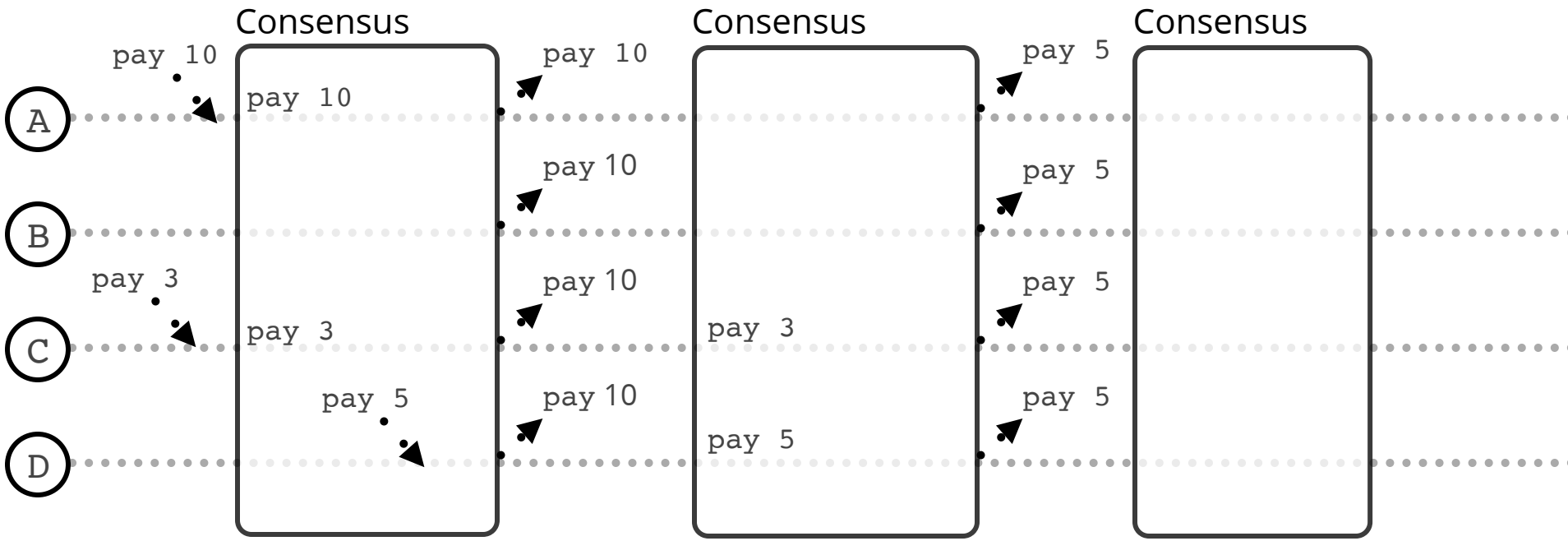
Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Tant que ma proposition n'a pas été acceptée, je recommence.



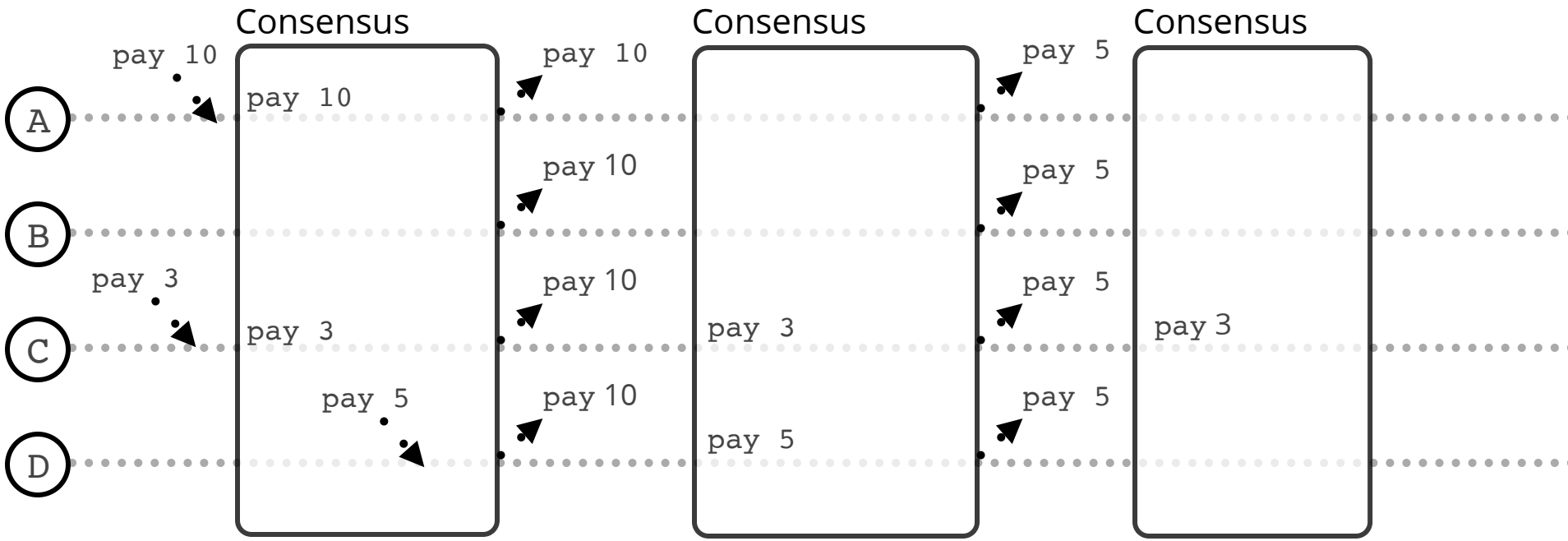
Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Tant que ma proposition n'a pas été acceptée, je recommence.



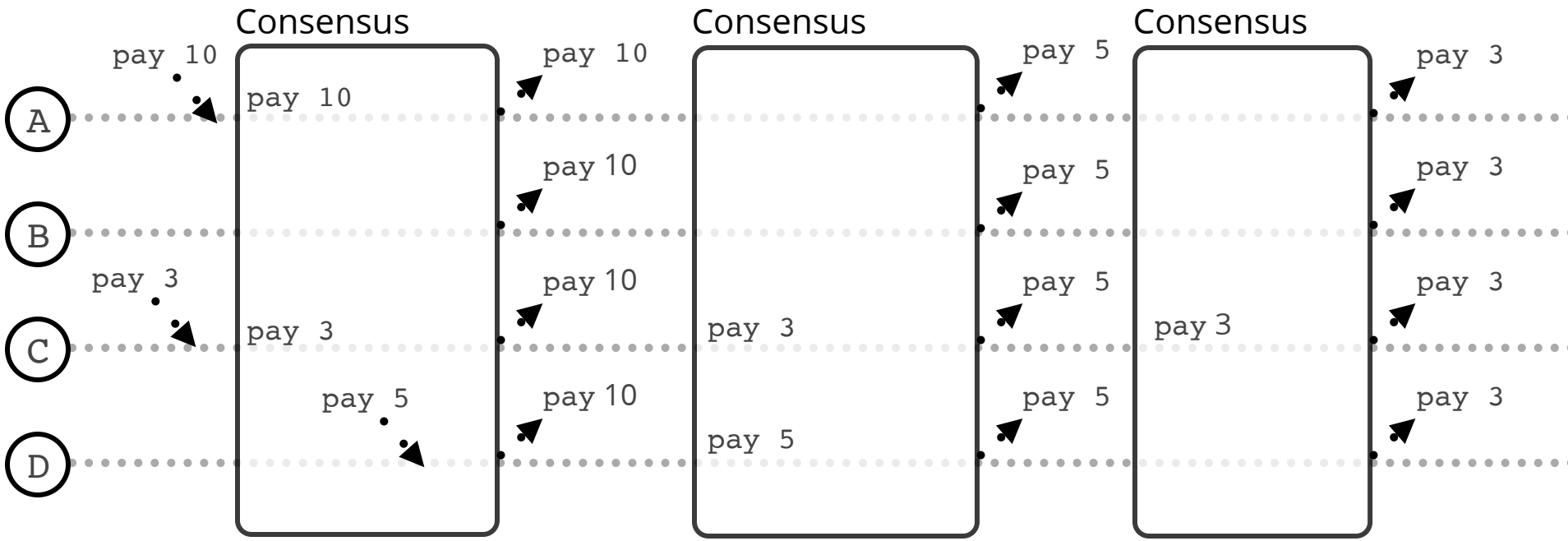
Machine d'état partagée

Tous les processus maintiennent une copie de la même machine d'état :

- Des états *(par exemple, un ensemble de comptes bancaires)*
- Des transitions entre états *(par exemple, des commandes `pay`)*

Si je souhaite modifier l'état, je lance un consensus et propose la modification

Tant que ma proposition n'a pas été acceptée, je recommence.



Machine d'état partagée

Exemples

États	**Transitions**
Graphe de dettes	Ajouts de payements
Qui a le jeton de la SC	Passation du jeton à un autre
Qui est l'élu	Passation du rôle à un autre
Comptes en banque	Transaction d'un compte à un autre

Machine d'état partagée

Exemples

États	**Transitions**
Graphe de dettes	Ajouts de payements
Qui a le jeton de la SC	Passation du jeton à un autre
Qui est l'élu	Passation du rôle à un autre
Comptes en banque	Transaction d'un compte à un autre

La solution à tout ?

Machine d'état partagée

Exemples

États	**Transitions**
Graphe de dettes	Ajouts de payements
Qui a le jeton de la SC	Passation du jeton à un autre
Qui est l'élu	Passation du rôle à un autre
Comptes en banque	Transaction d'un compte à un autre

La solution à tout ?

Tout ce qui peut être modélisé par une machine d'état, donc oui.

Machine d'état partagée

Exemples

États	**Transitions**
Graphe de dettes	Ajouts de payements
Qui a le jeton de la SC	Passation du jeton à un autre
Qui est l'élu	Passation du rôle à un autre
Comptes en banque	Transaction d'un compte à un autre

La solution à tout ?

Tout ce qui peut être modélisé par une machine d'état, donc oui.

Mais, bien plus lourd.

Consensus IRL

Suppositions

Jusqu'à N/2 pannes récupérables
Pas de panne arbitraire
Système partiellement asynchrone
Sur les messages : intégrité uniquement

Paxos
Lamport (1989)
Encore lui, bien sûr

Raft *(prochain cours)*
Diego Ongaro and John Ousterhout (2014)
Améliorations de Paxos, spécialisé dans la simulation de machines d'état

Et d'autres bien sûr...

↓ Exercices

Exercice 1

Proposer une implémentation de l'exclusion mutuelle utilisant l'abstraction de consensus vue dans ce cours (sous forme de boite noire).

Garantissez notamment

- que toute demande sera un jour satisfaite, *même si un autre processus demande la SC sans cesse* ;
- qu'aucune entrée en SC ne sera autorisée avant la fin de celle en cours.

Exercice 2

Nous souhaitons avoir un algorithme de consensus permettant de s'accorder non plus sur une proposition, mais sur **un ordonnancement de propositions**, contenant toutes les propositions des processus valides, et autorisant les propositions de processus en panne.

Proposer une première version supposant un réseau synchrone.

Proposer ensuite une version ne faisant pas de supposition sur le réseau, utilisant (sous forme de boite noire) un algorithme de consensus tolérant un réseau partiellement synchrone.

Comment un tel algorithme pourrait-il être utilisé dans votre projet ChatsApp, afin d'offrir la garantie d'ordre global de réception des messages ?