SDR

Communiquer avec des pannes

Speaker notes

Slides by Olivier Lemer

Based on
Cachin, Christian, Rachid Guerraoui, and Luís Rodrigues. Introduction to reliable and secure distributed programming. Springer Science & Business Media, 2011.

Aujourd'hui

1. Types de pannes

2. Protocoles de fiabilité

3. Broadcast

4. Détection de pannes

Types de pannes

Panne permanente

Panne dont on ne reviendra jamais

Speaker notes

La caractéristique "correcte en terme de pannes permanentes" d'un processus n'a de sens et n'est utile que dans un monde théorique. Elle nous servira à dire des choses comme "cet algorithme a besoin que tous les processus soient corrects en terme de pannes permanentes pour pouvoir fonctionner correctement" (ce qui est simplement une manière plus formelle de dire "cet algorithme a besoin qu'aucun processus ne tombe jamais en panne pour fonctionner correctement").

Dans ce cas-ci, ça parait un peu superflus d'introduire ce genre de formulation, mais pour d'autres types de panne (e.g. le suivant), ce sera très utile pour parler clairement et brièvement.

Panne permanente

Panne dont on ne reviendra jamais

On dit qu'un processus est **correct en terme de panne permanente** quand il **ne tombera jamais en panne permanente.**

Speaker notes

La caractéristique "correcte en terme de pannes permanentes" d'un processus n'a de sens et n'est utile que dans un monde théorique. Elle nous servira à dire des choses comme "cet algorithme a besoin que tous les processus soient corrects en terme de pannes permanentes pour pouvoir fonctionner correctement" (ce qui est simplement une manière plus formelle de dire "cet algorithme a besoin qu'aucun processus ne tombe jamais en panne pour fonctionner correctement").

Dans ce cas-ci, ça parait un peu superflus d'introduire ce genre de formulation, mais pour d'autres types de panne (e.g. le suivant), ce sera très utile pour parler clairement et brièvement.

Panne permanente

Panne dont on ne reviendra jamais

On dit qu'un processus est **correct en terme de panne permanente** quand il **ne tombera jamais en panne permanente.**

Speaker notes

La caractéristique "correcte en terme de pannes permanentes" d'un processus n'a de sens et n'est utile que dans un monde théorique. Elle nous servira à dire des choses comme "cet algorithme a besoin que tous les processus soient corrects en terme de pannes permanentes pour pouvoir fonctionner correctement" (ce qui est simplement une manière plus formelle de dire "cet algorithme a besoin qu'aucun processus ne tombe jamais en panne pour fonctionner correctement").

Dans ce cas-ci, ça parait un peu superflus d'introduire ce genre de formulation, mais pour d'autres types de panne (e.g. le suivant), ce sera très utile pour parler clairement et brièvement.

Panne permanente

Panne dont on ne reviendra jamais

On dit qu'un processus est **correct en terme de panne permanente** quand il **ne tombera jamais en panne permanente.**

Speaker notes

La caractéristique "correcte en terme de pannes permanentes" d'un processus n'a de sens et n'est utile que dans un monde théorique. Elle nous servira à dire des choses comme "cet algorithme a besoin que tous les processus soient corrects en terme de pannes permanentes pour pouvoir fonctionner correctement" (ce qui est simplement une manière plus formelle de dire "cet algorithme a besoin qu'aucun processus ne tombe jamais en panne pour fonctionner correctement").

Dans ce cas-ci, ça parait un peu superflus d'introduire ce genre de formulation, mais pour d'autres types de panne (e.g. le suivant), ce sera très utile pour parler clairement et brièvement.

Panne récupérable

Panne dont on peut revenir, *parfois en ayant perdu des informations ("amnésie").*

Un processus qui revient d'une panne en est conscient.

Speaker notes

Comme pour les autres types de panne, le terme "correct" n'a de sens que dans un monde théorique, puisqu'il est impossible dans la vraie vie de prévoire le futur.

L'intéret de ce terme devient ici plus clair ; il est plus simple de dire "cet algorithme nécessite que tous les processus soient corrects en terme de panne récupérable" que de dire "cet algorithme nécessite que tous les processus ne tombe jamais en panne à partir d'un certain instant T inconnu mais fini, bien qu'ils puissent tomber en panne autant qu'ils veulent avant cet instant T du moment qu'ils en reviennent avant T".

Ça nous permettra aussi de classifier plus clairement les algorithmes que nous verrons, typiquement : "l'algorithme A fait la supposition que tous les processus sont corrects en terme de pnane permanente, alors que l'algorithme B fait la supposition qu'ils sont corrects en terme de panne récupérable, donc l'algorithme B est plus résiliant que le A".

Panne récupérable

Panne dont on peut revenir, *parfois en ayant perdu des informations ("amnésie").*

Un processus qui revient d'une panne en est conscient.

Speaker notes

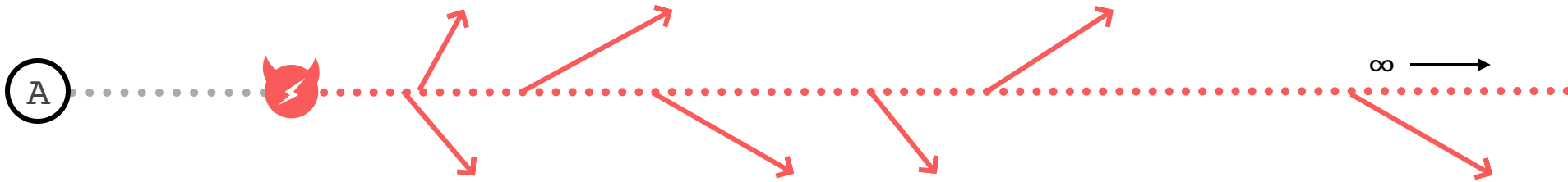
Comme pour les autres types de panne, le terme "correct" n'a de sens que dans un monde théorique, puisqu'il est impossible dans la vraie vie de prévoire le futur.

L'intéret de ce terme devient ici plus clair ; il est plus simple de dire "cet algorithme nécessite que tous les processus soient corrects en terme de panne récupérable" que de dire "cet algorithme nécessite que tous les processus ne tombe jamais en panne à partir d'un certain instant T inconnu mais fini, bien qu'ils puissent tomber en panne autant qu'ils veulent avant cet instant T du moment qu'ils en reviennent avant T".

Ça nous permettra aussi de classifier plus clairement les algorithmes que nous verrons, typiquement : "l'algorithme A fait la supposition que tous les processus sont corrects en terme de pnane permanente, alors que l'algorithme B fait la supposition qu'ils sont corrects en terme de panne récupérable, donc l'algorithme B est plus résiliant que le A".

Panne arbitraire
a.k.a. byzantine

Panne suite à laquelle tout est possible...



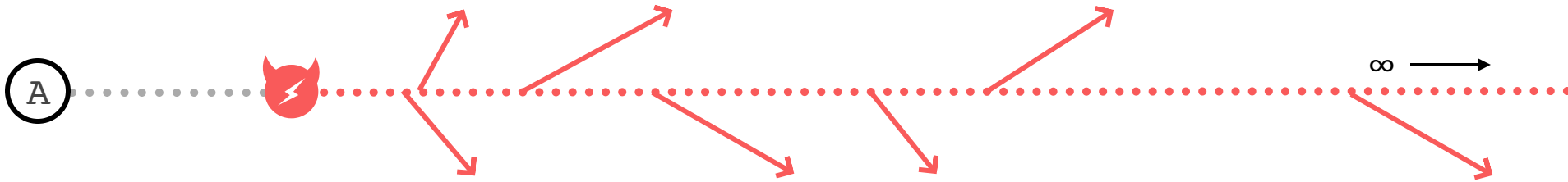
Peut être due à un bug, ou bien à une corruption par un agent malicieux.

Speaker notes

Il sera probablement clair que ce genre de panne est extrèmement difficile à tolérer. En d'autres termes, il est extrèmement difficile de concevoir un algorithme distribué capable de continuer de fonctionner correctement même quand l'un des processus du système commence à agir de manière malicieuse. On ne peut plus faire confiance à personne dans le système.

Panne arbitraire
a.k.a. byzantine

Panne suite à laquelle tout est possible...



Peut être due à un bug, ou bien à une corruption par un agent malicieux.

On dit qu'un processus est **correct en terme de panne arbitraire** quand il **suivra toujours l'algorithme attendu.**

Speaker notes

Il sera probablement clair que ce genre de panne est extrèmement difficile à tolérer. En d'autres termes, il est extrèmement difficile de concevoir un algorithme distribué capable de continuer de fonctionner correctement même quand l'un des processus du système commence à agir de manière malicieuse. On ne peut plus faire confiance à personne dans le système.

and more...

Panne d'omission

Lorsqu'un message devant être envoyé ne l'est pas.

Panne d'eavesdropping

Lorsqu'un message peut être lu par une entité extérieure au système.

Garanties à fournir

Garanties du réseau *Supposées, à ne pas casser.*

fair-lossy pertes de message autorisées,
mais après assez de renvois, le message finit par arriver.

Garanties du transport layer

Speaker notes

Aucune de ces trois garanties n'est donnée par le réseau : elles sont construites par les protocoles de fiabilité. Réémission sur timeout contre la perte, identifiant de requête contre la duplication, un seul message en vol contre le réordonnancement.

Garanties à fournir

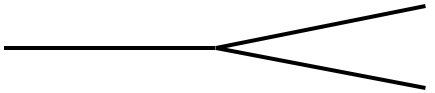
Garanties du réseau *Supposées, à ne pas casser.*

fair-lossy pertes de message autorisées,
mais après assez de renvois, le message finit par arriver.

Garanties du transport layer

Pas de perte

Un message envoyé est reçu.



Pas de duplication

Un message envoyé n'est reçu qu'une fois.

Pas de changement d'ordre

L'ordre de réception est celui d'envoi.

Speaker notes

Aucune de ces trois garanties n'est donnée par le réseau : elles sont construites par les protocoles de fiabilité. Réémission sur timeout contre la perte, identifiant de requête contre la duplication, un seul message en vol contre le réordonnancement.

Garanties à fournir

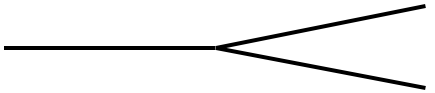
Garanties du réseau *Supposées, à ne pas casser.*

fair-lossy pertes de message autorisées,
mais après assez de renvois, le message finit par arriver.

Garanties du transport layer

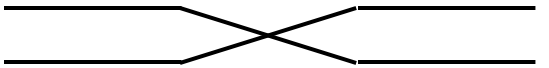
Pas de perte

Un message envoyé est reçu.



Pas de duplication

Un message envoyé n'est reçu qu'une fois.



Pas de changement d'ordre

L'ordre de réception est celui d'envoi.

Garanties du système

Speaker notes

Aucune de ces trois garanties n'est donnée par le réseau : elles sont construites par les protocoles de fiabilité. Réémission sur timeout contre la perte, identifiant de requête contre la duplication, un seul message en vol contre le réordonnancement.

RPC

Remote Procedure Call

Interface identique à un appel de fonction classique,
mais exécuté sur une autre machine.

Speaker notes

L'interface d'un RPC ressemble à un appel de fonction, mais les modes d'échec sont ceux du réseau. C'est cette tension qui justifie quatre protocoles au lieu d'un.

RPC

Remote Procedure Call

Interface identique à un appel de fonction classique,
mais exécuté sur une autre machine.

Problème : L'autre machine peut tomber en panne.

Speaker notes

L'interface d'un RPC ressemble à un appel de fonction, mais les modes d'échec sont ceux du réseau. C'est cette tension qui justifie quatre protocoles au lieu d'un.

RPC

Remote Procedure Call

Interface identique à un appel de fonction classique,
mais exécuté sur une autre machine.

Problème : L'autre machine peut tomber en panne.

Résolu par 4 protocoles de fiabilité.

Speaker notes

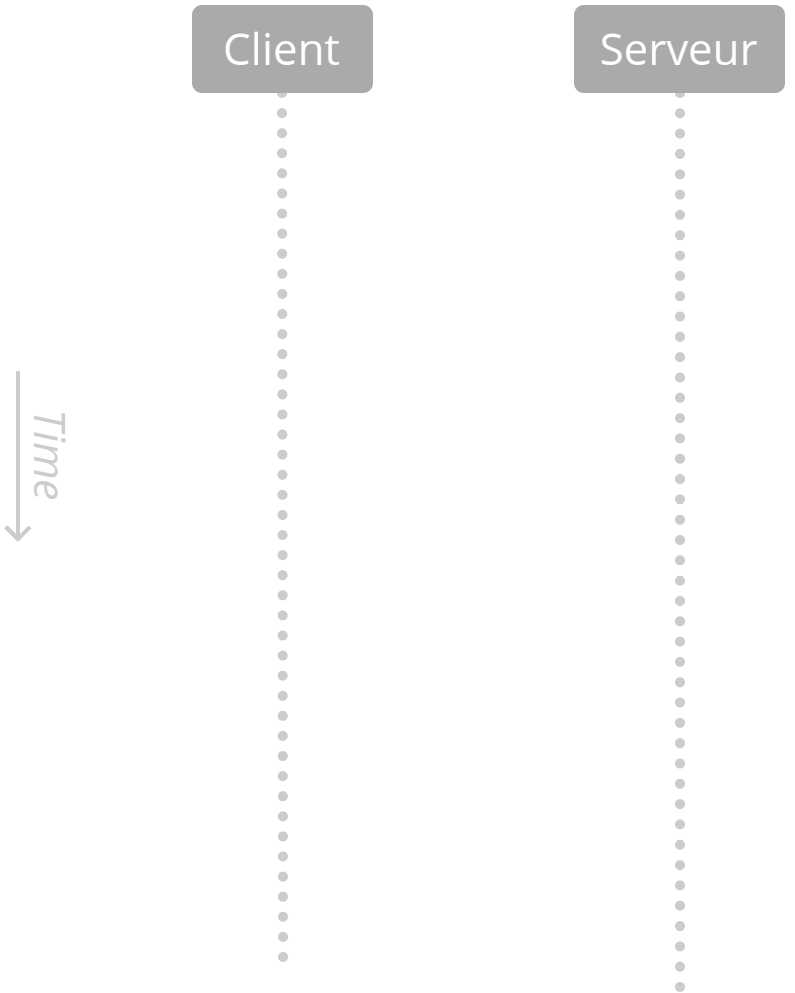
L'interface d'un RPC ressemble à un appel de fonction, mais les modes d'échec sont ceux du réseau. C'est cette tension qui justifie quatre protocoles au lieu d'un.

Protocole Request

"peut-être" (a.k.a. *maybe*)

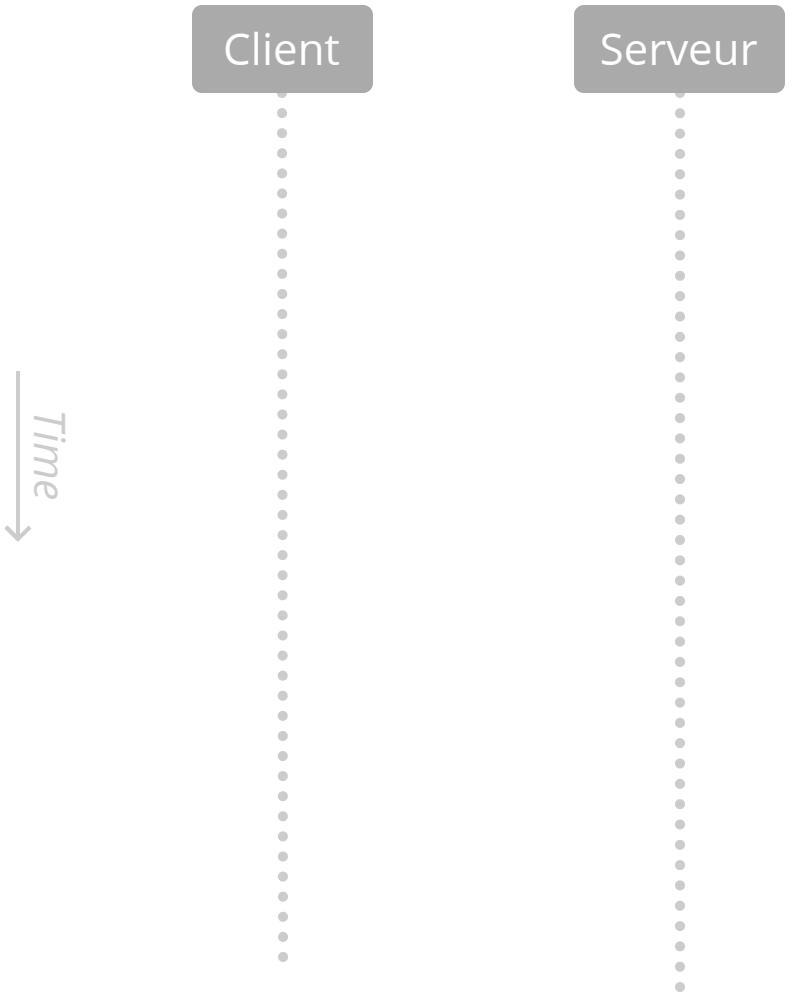
Protocoles de fiabilité

Protocole **R** (Request)



Protocoles de fiabilité

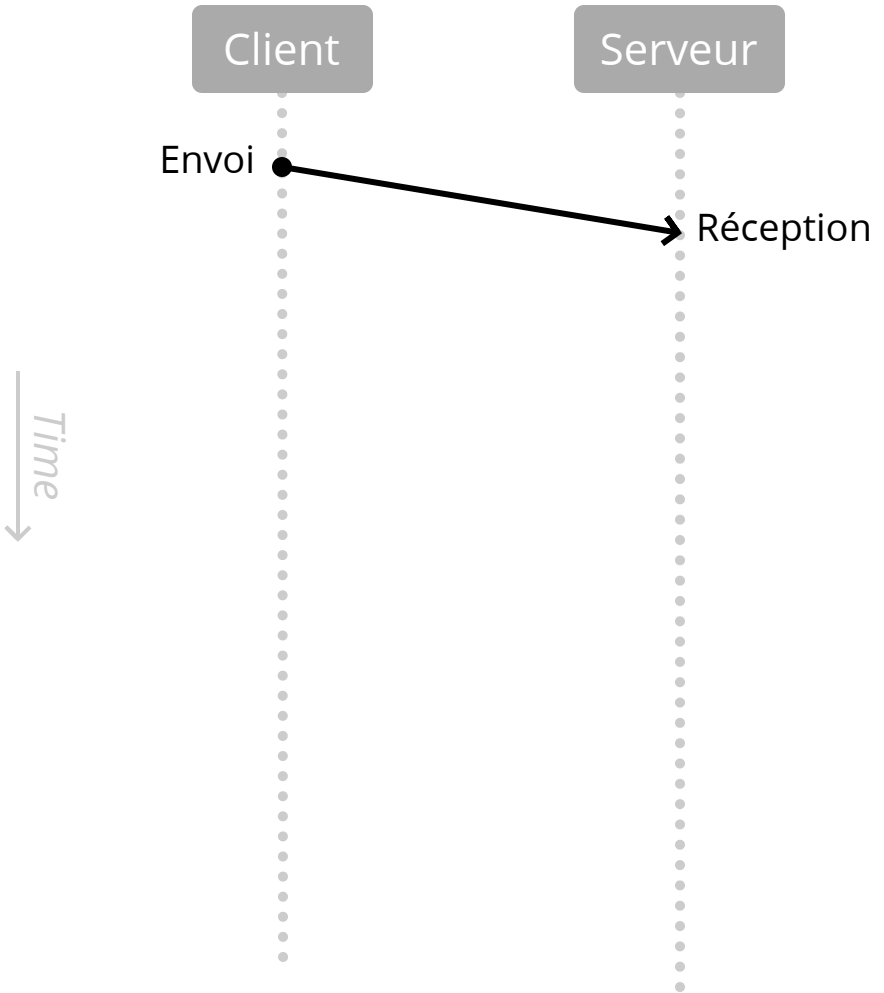
Protocole **R** (Request)



Uniquement une requête,
pas de réponse du serveur.

Protocoles de fiabilité

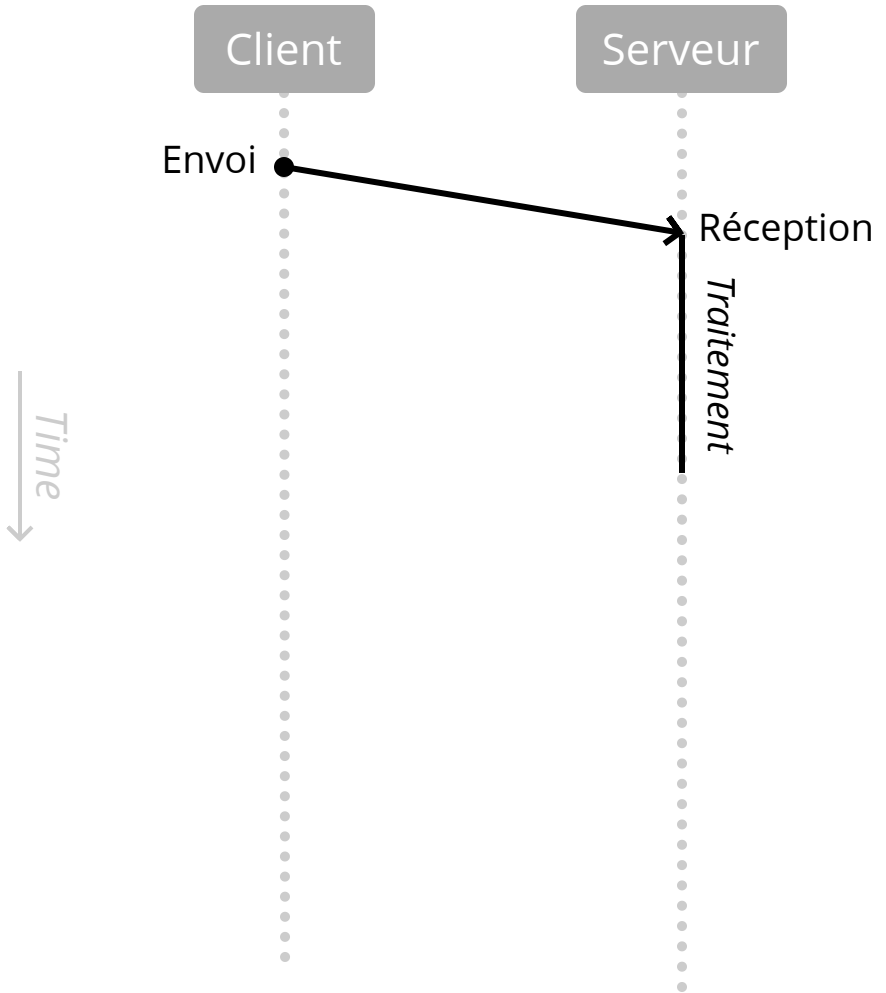
Protocole **R** (Request)



Uniquement une requête,
pas de réponse du serveur.

Protocoles de fiabilité

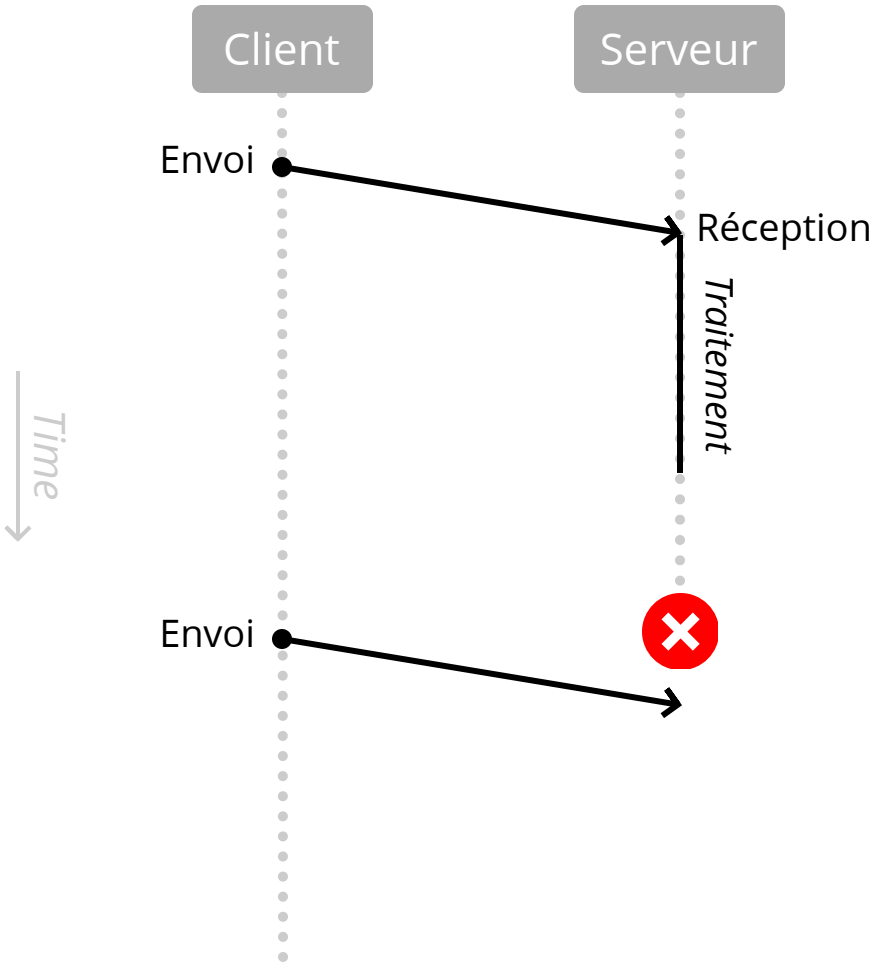
Protocole **R** (Request)



Uniquement une requête,
pas de réponse du serveur.

Protocoles de fiabilité

Protocole **R** (Request)

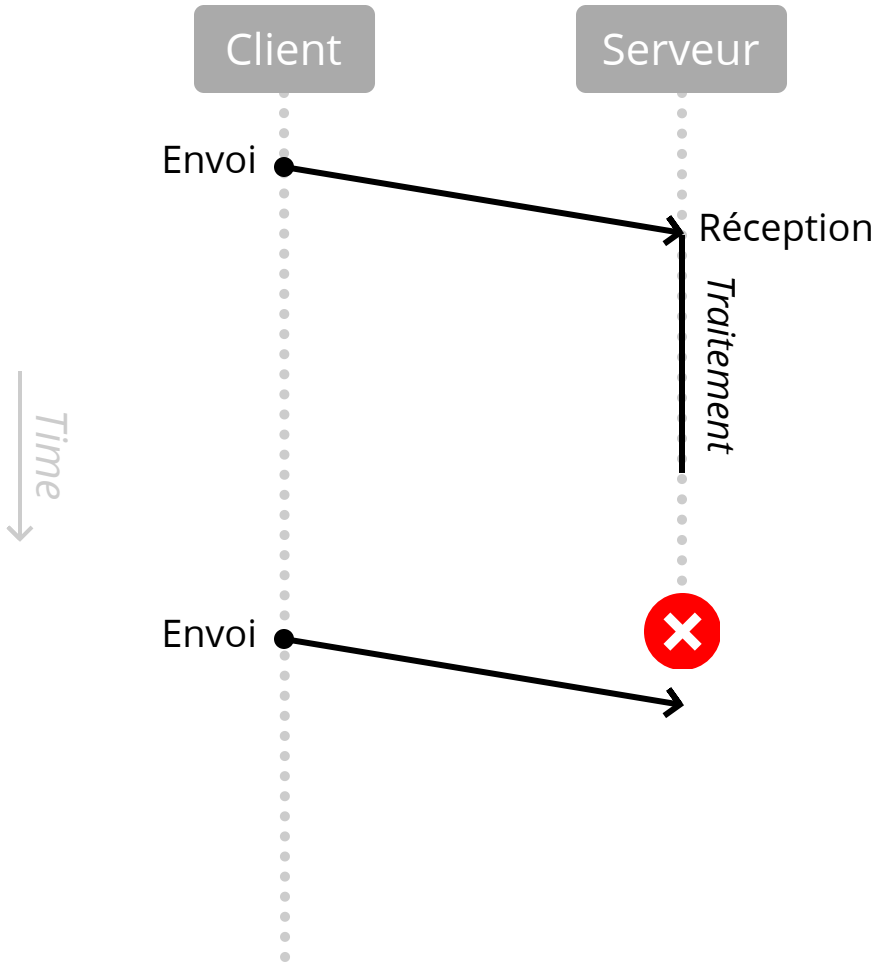


Uniquement une requête,
pas de réponse du serveur.

Panne Serveur ?

Protocoles de fiabilité

Protocole **R** (Request) *(aka "Peut-être")*

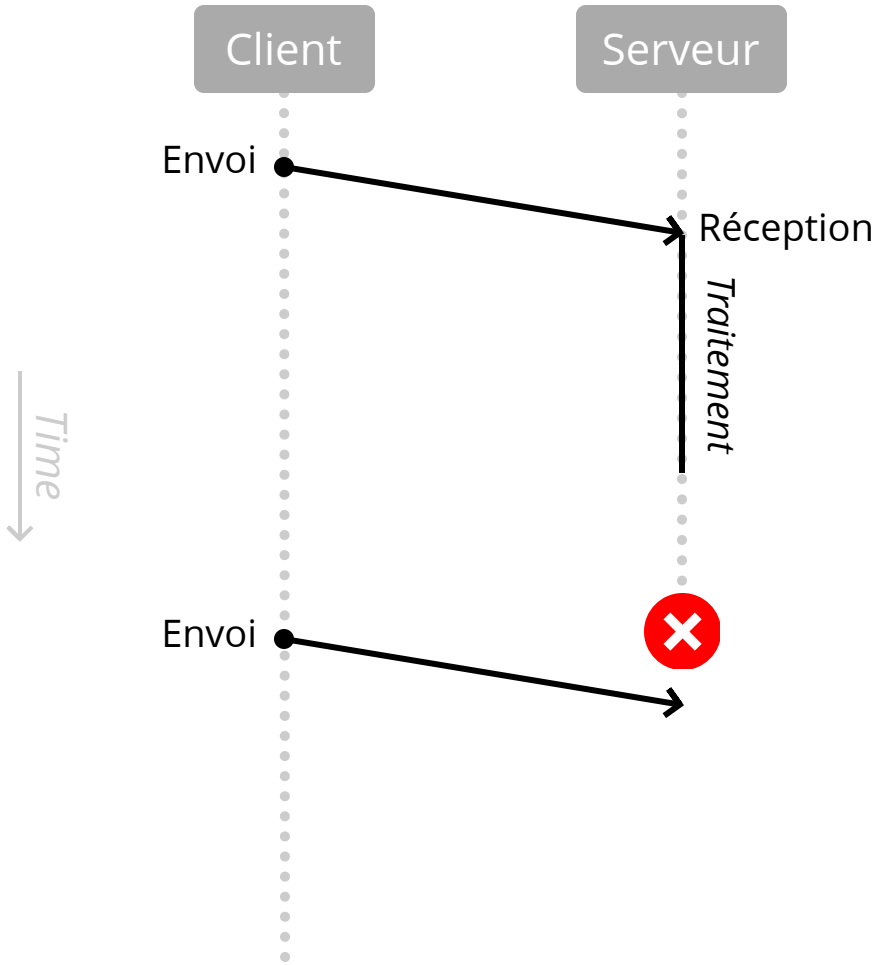


Uniquement une requête,
pas de réponse du serveur.

Panne Serveur ?
Requête perdue, client l'ignore.

Protocoles de fiabilité

Protocole **R** (Request) *(aka "Peut-être")*



Uniquement une requête,
pas de réponse du serveur.

Panne Serveur ?
Requête perdue, client l'ignore.

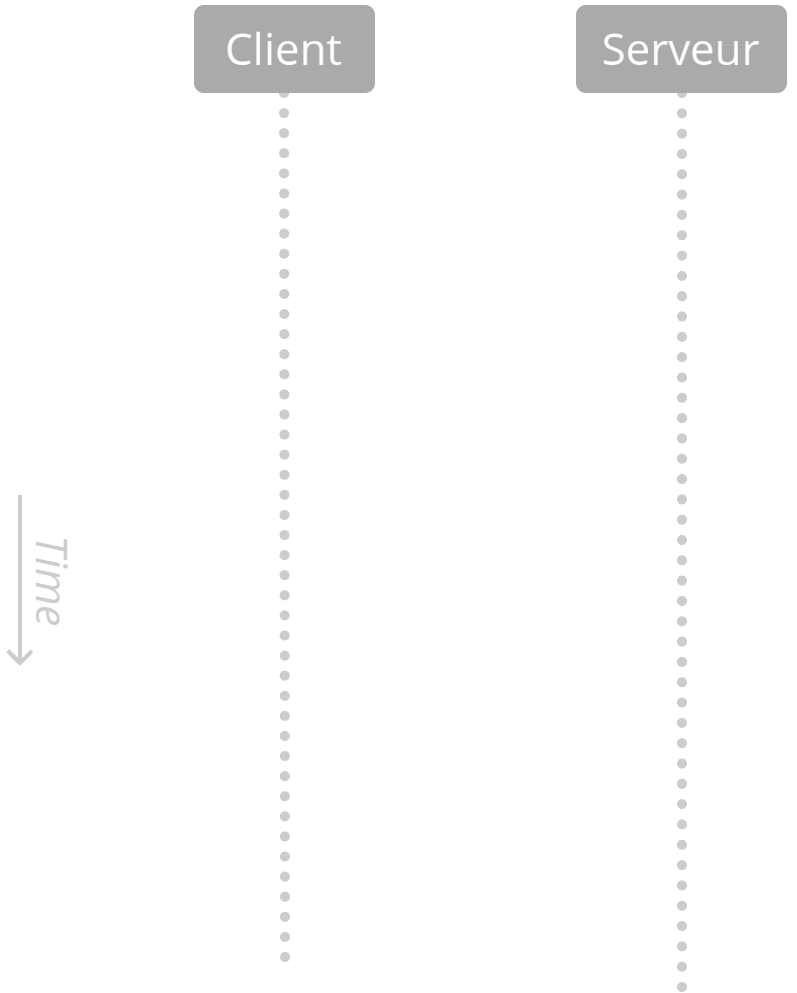
Le message sera **peut-être** traité.

Protocole Request-Reply

"au moins une fois" (a.k.a. *at-least-once*)

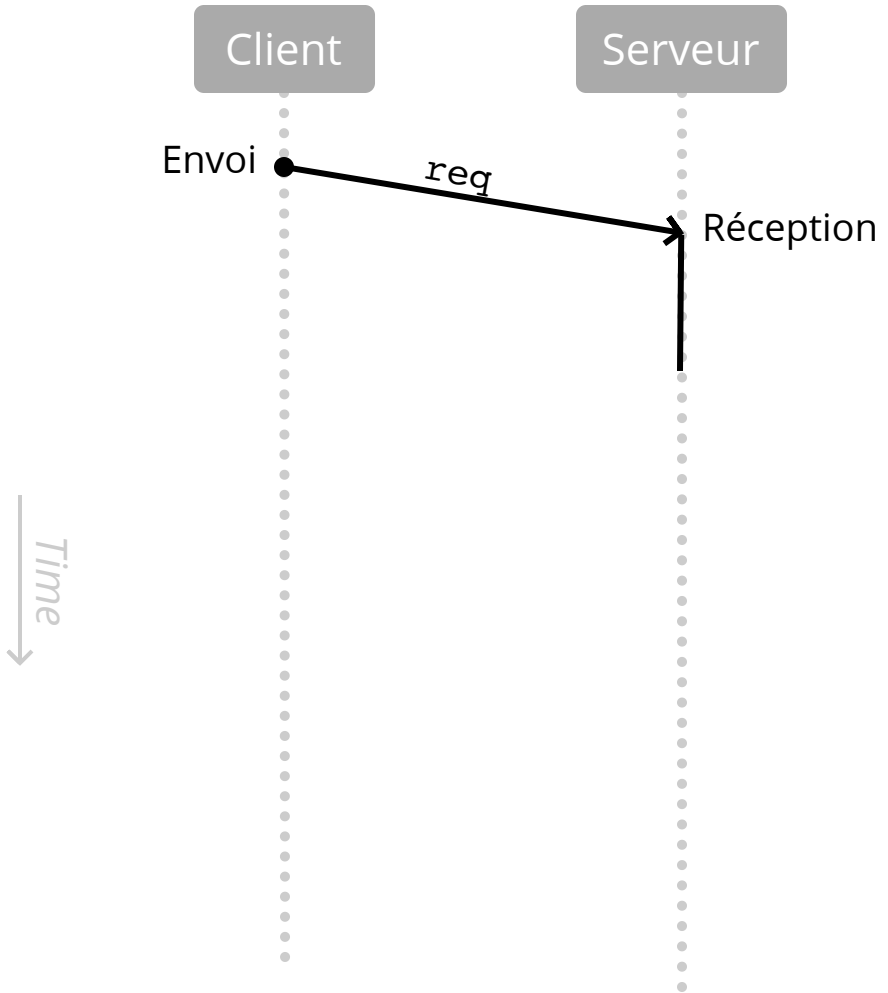
Protocoles de fiabilité

Protocole **RR** (Request-Reply)



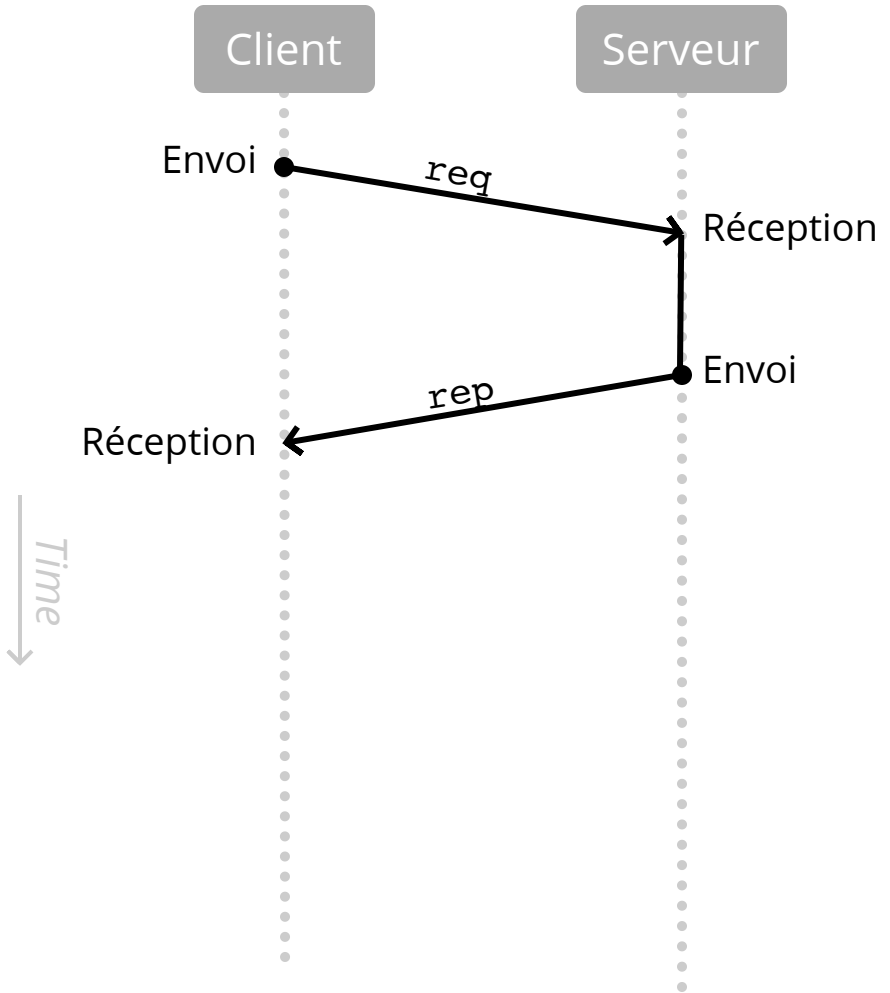
Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Protocoles de fiabilité

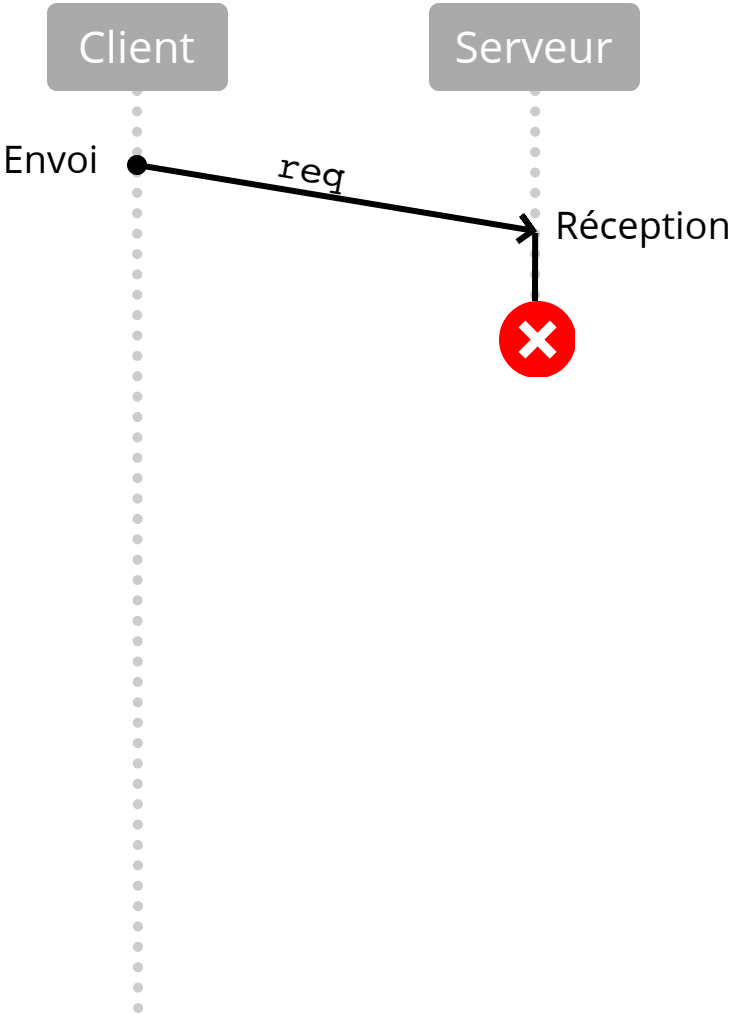
Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Protocoles de fiabilité

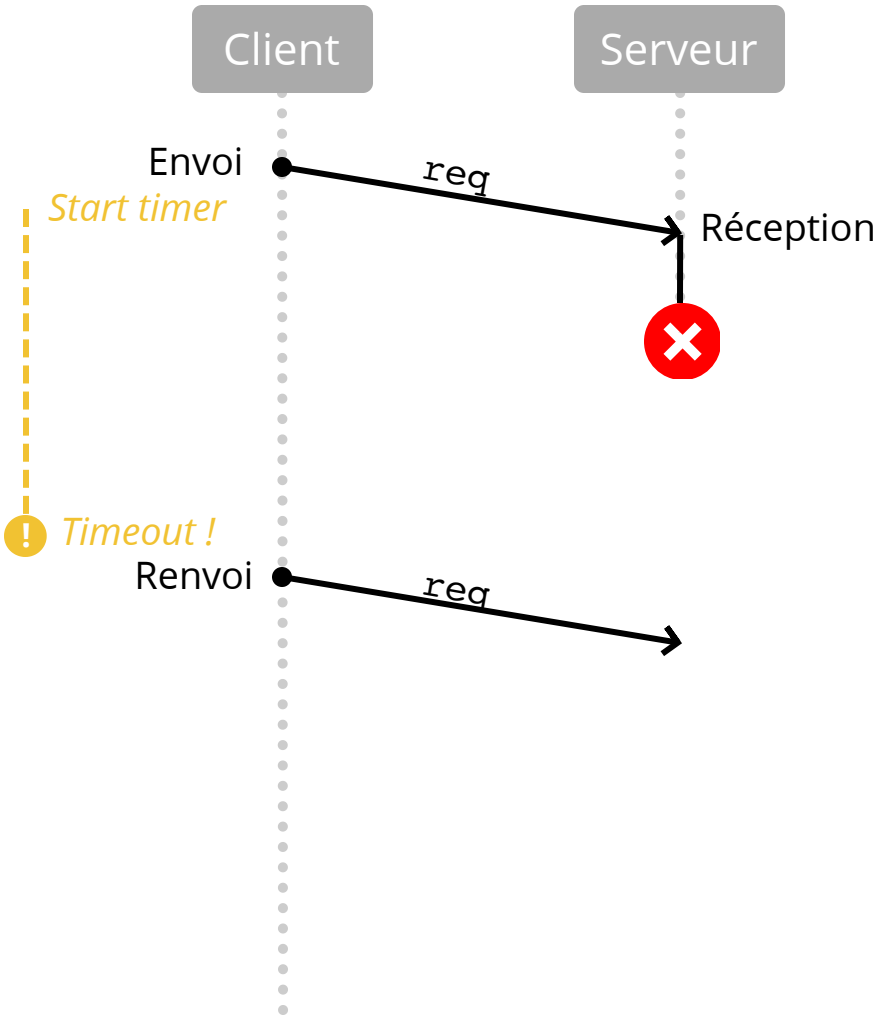
Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



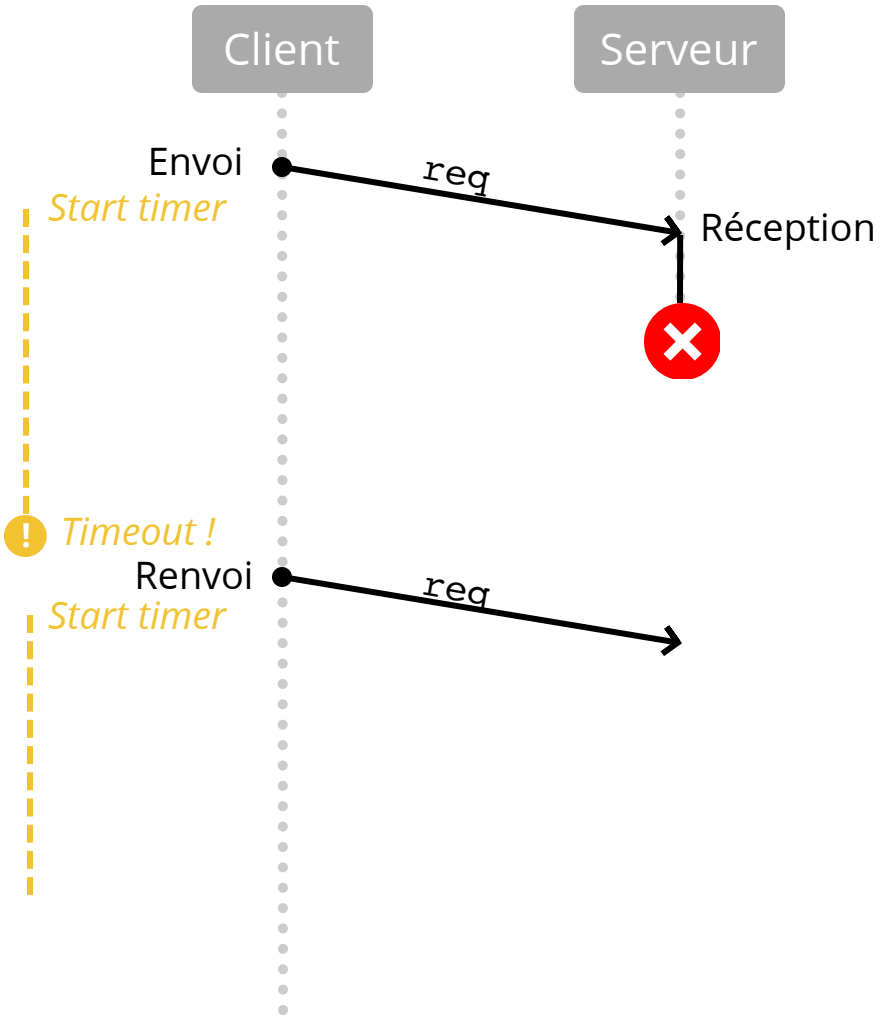
Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



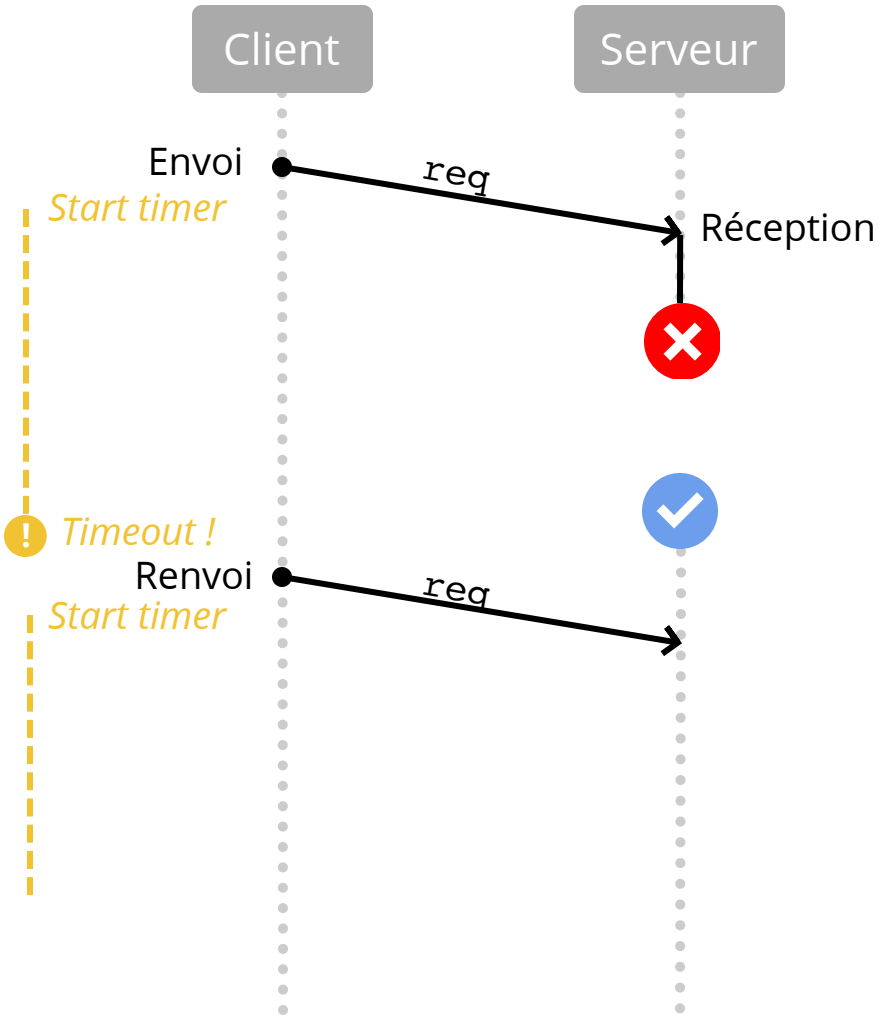
Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



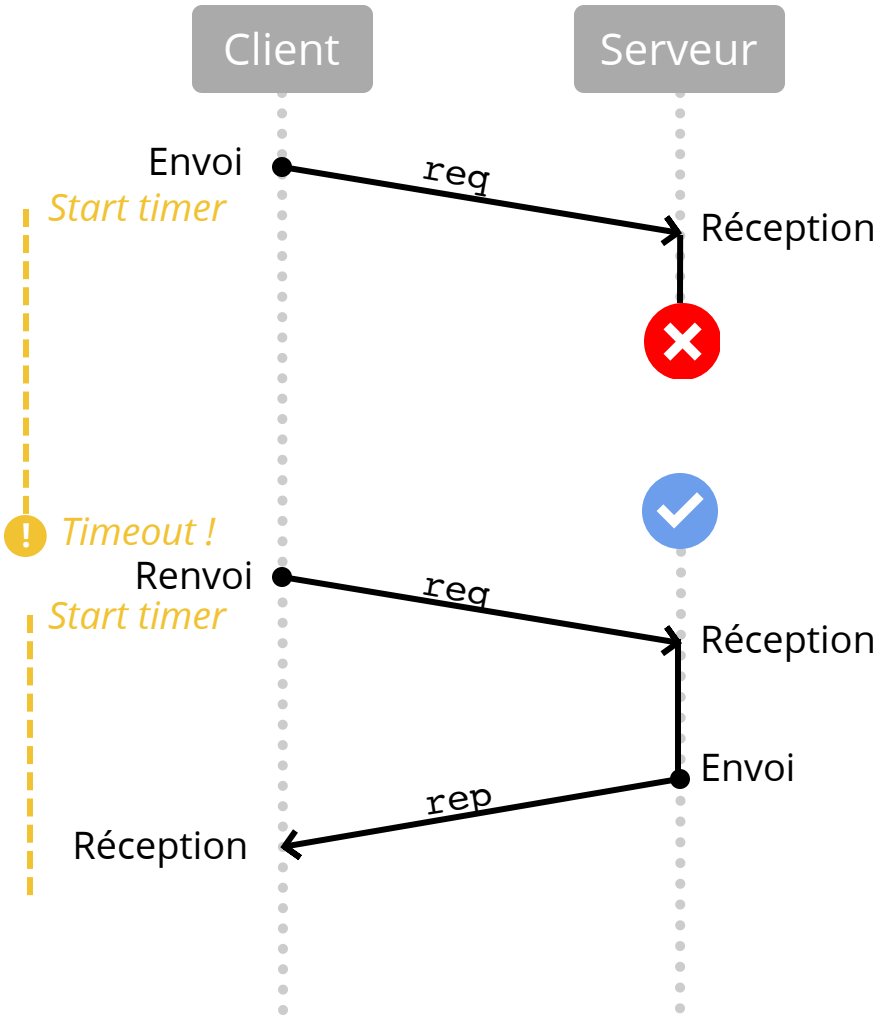
Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



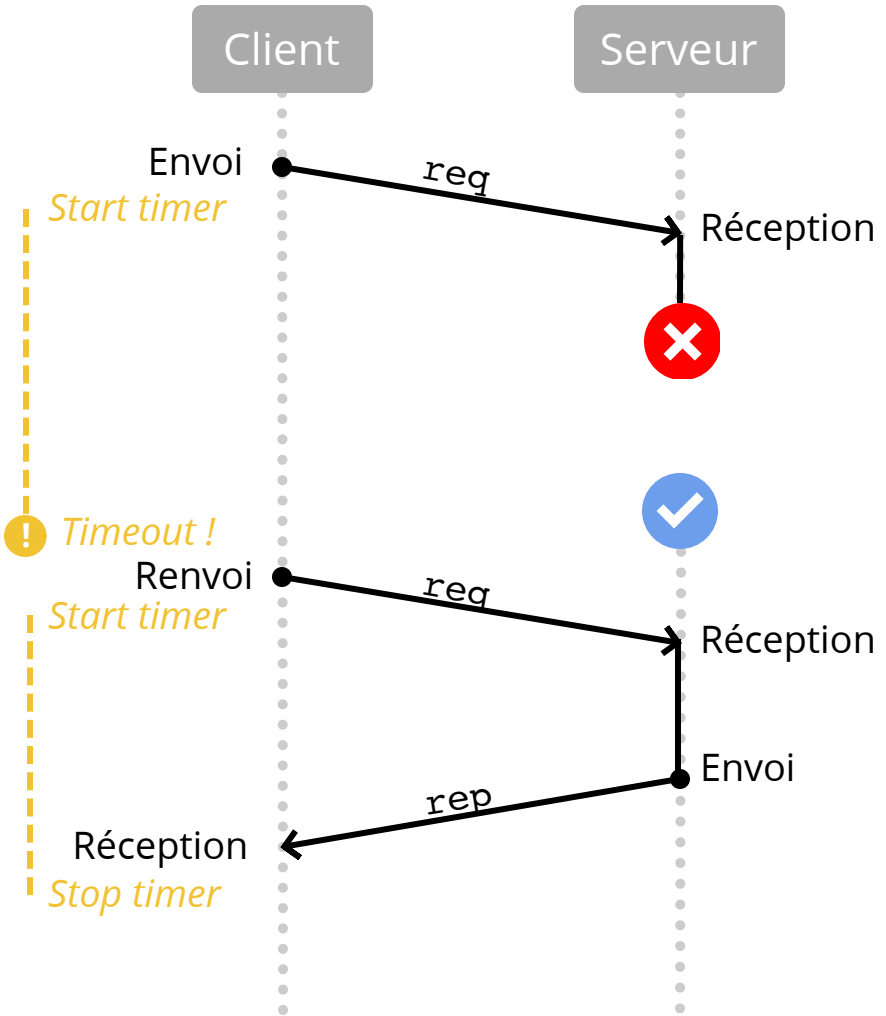
Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



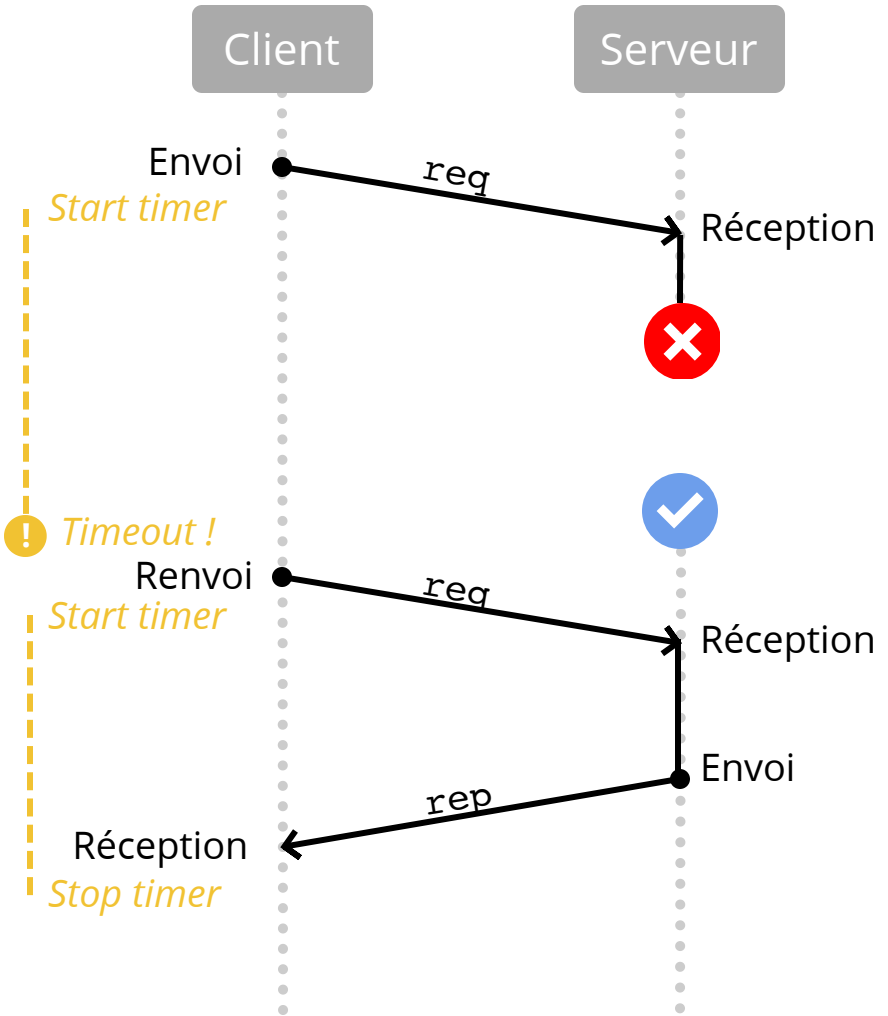
Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

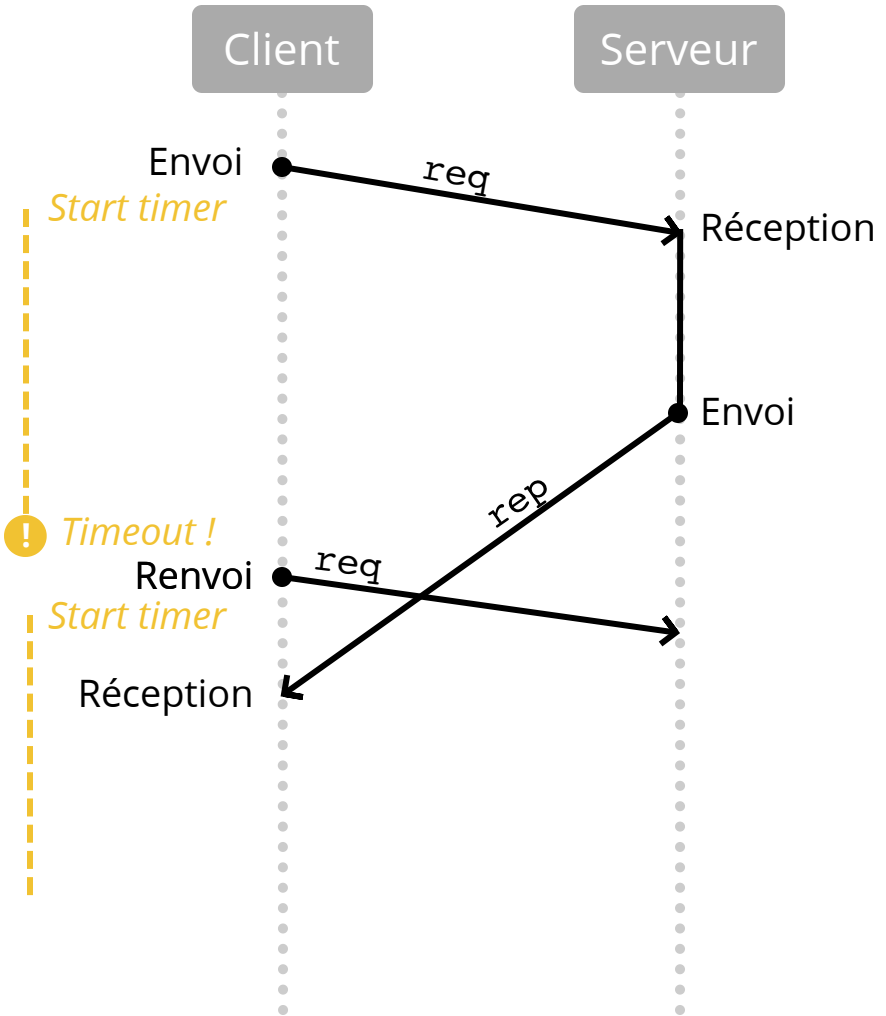
Timeout

déclenche un renvoi.
est annulé à la réception.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

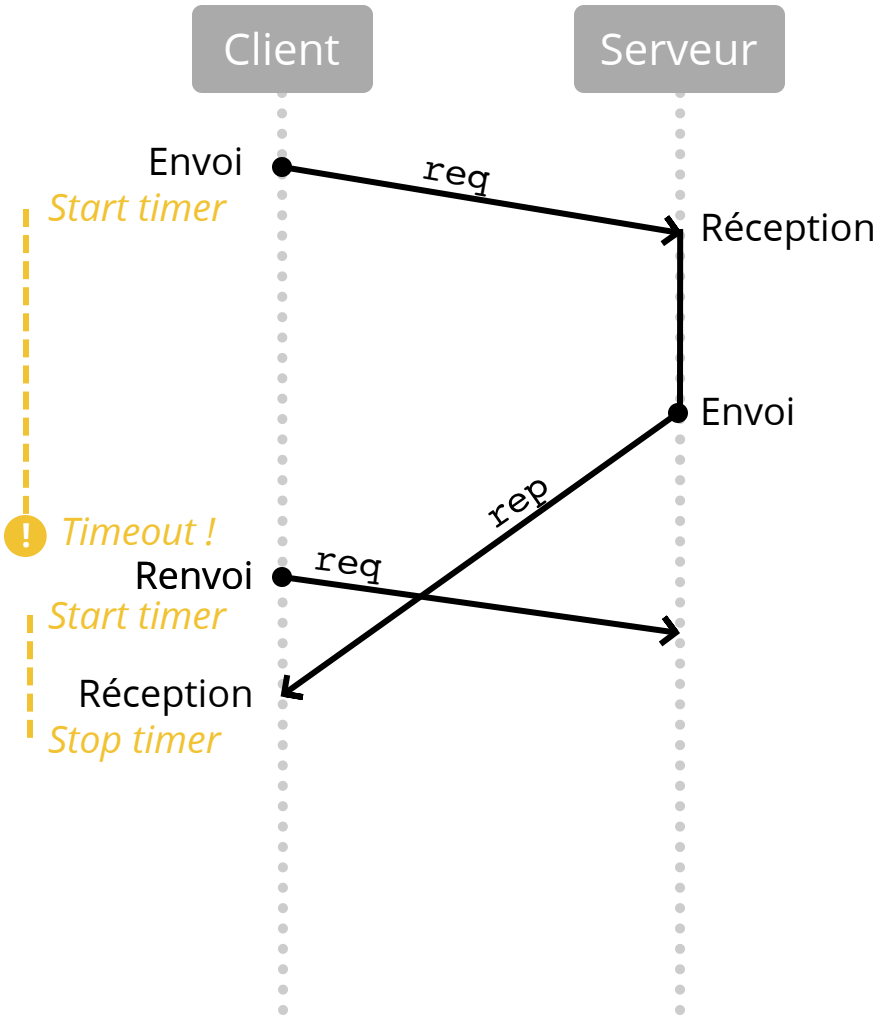
Timeout

déclenche un renvoi.
est annulé à la réception.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

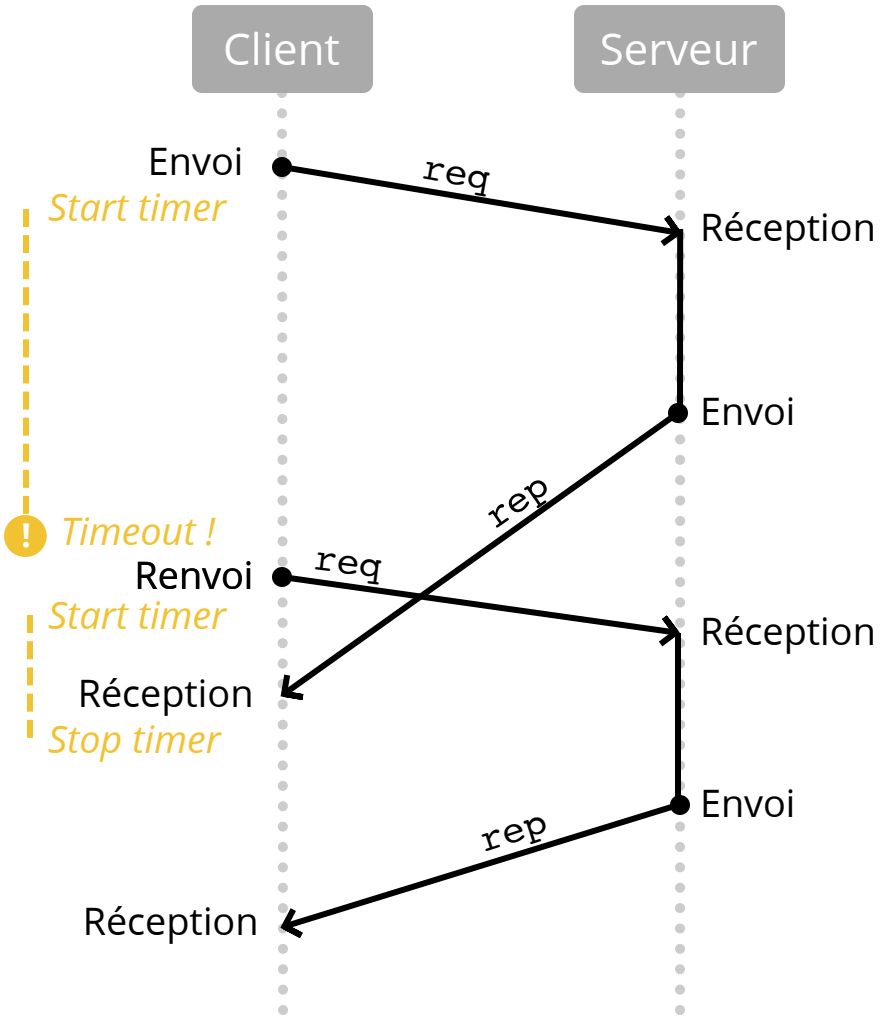
Timeout

déclenche un renvoi.
est annulé à la réception.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

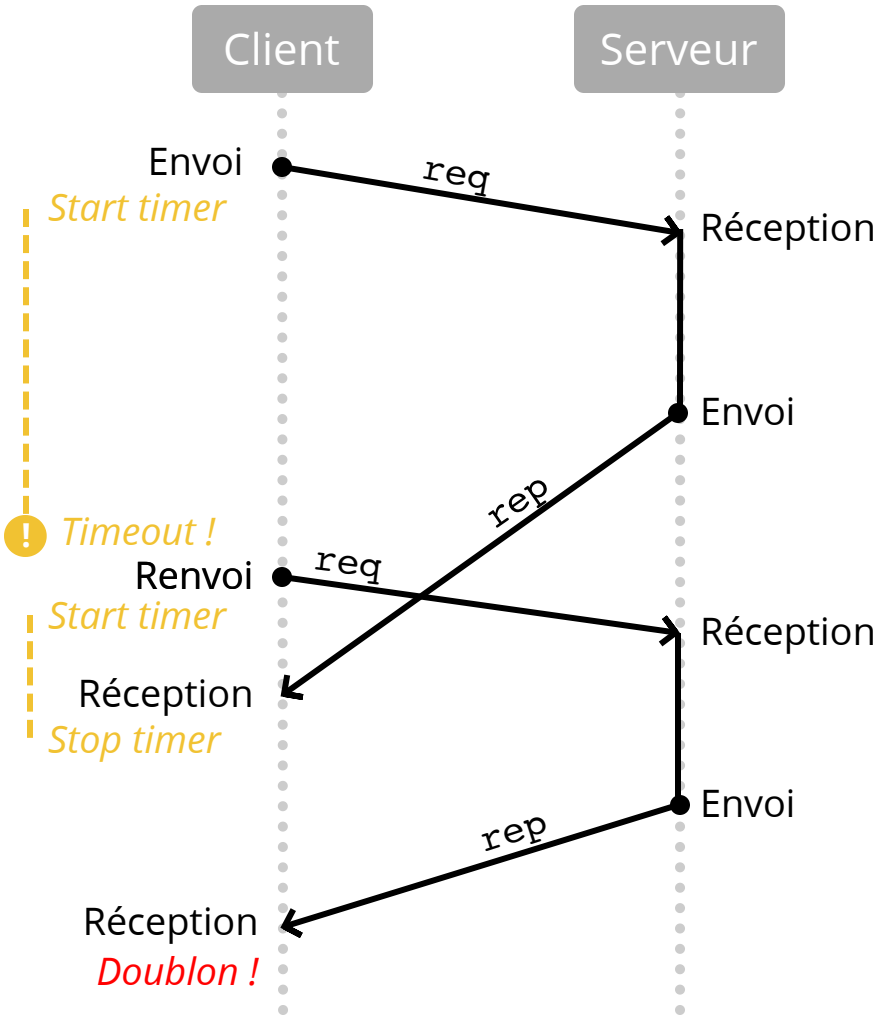
Timeout

déclenche un renvoi.
est annulé à la réception.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

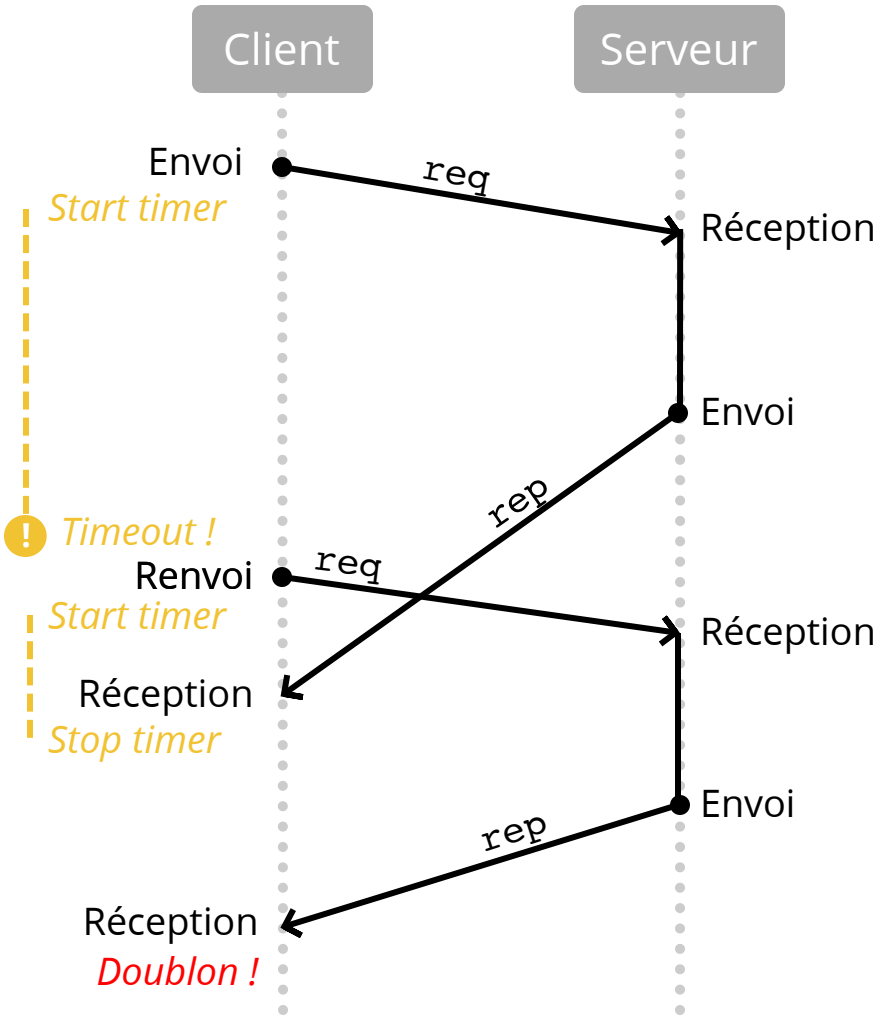
déclenche un renvoi.
est annulé à la réception.

Problème ?

Risque de reception double.

Protocoles de fiabilité

Protocole **RR** (Request-Reply) *(version "Au moins une fois")*



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Problème ?

Risque de reception double.

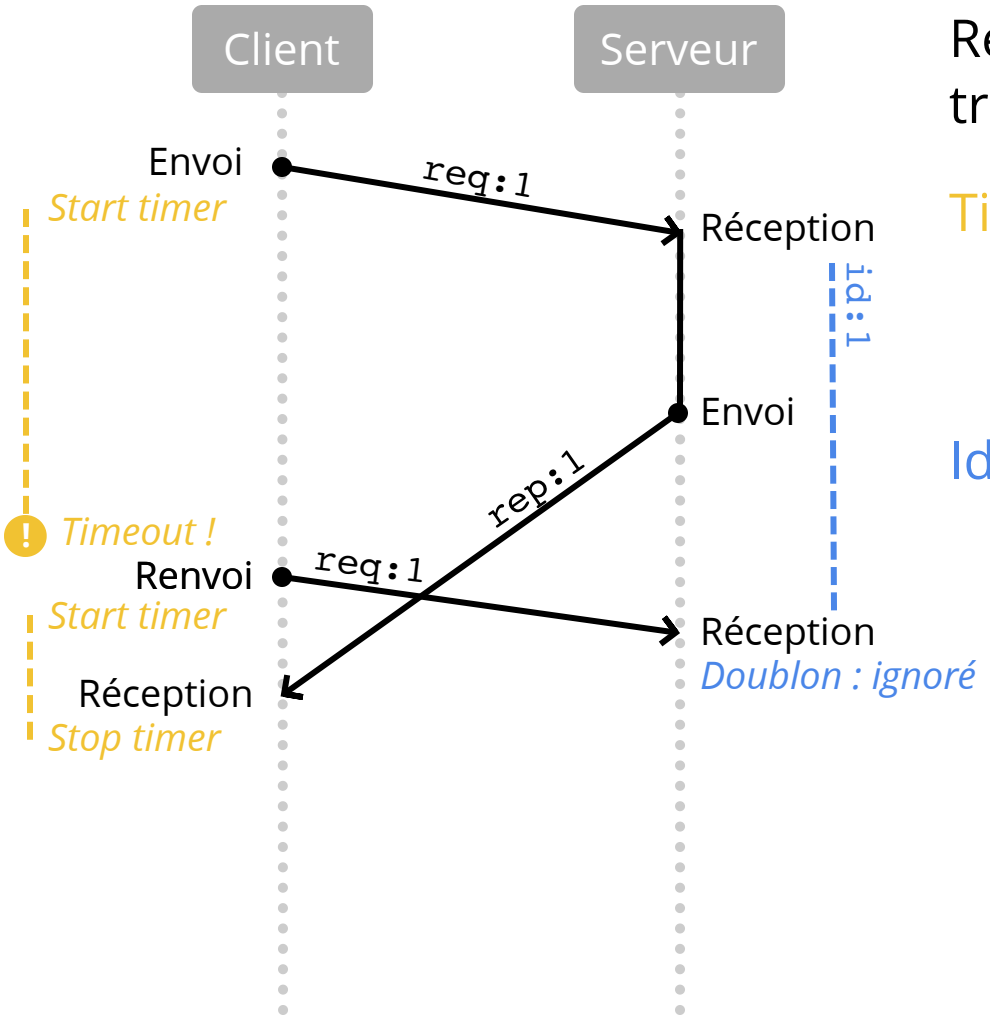
Le message sera traité
au moins une fois

Protocole Request-Reply

"au plus une fois" (a.k.a. *at-most-once*)

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

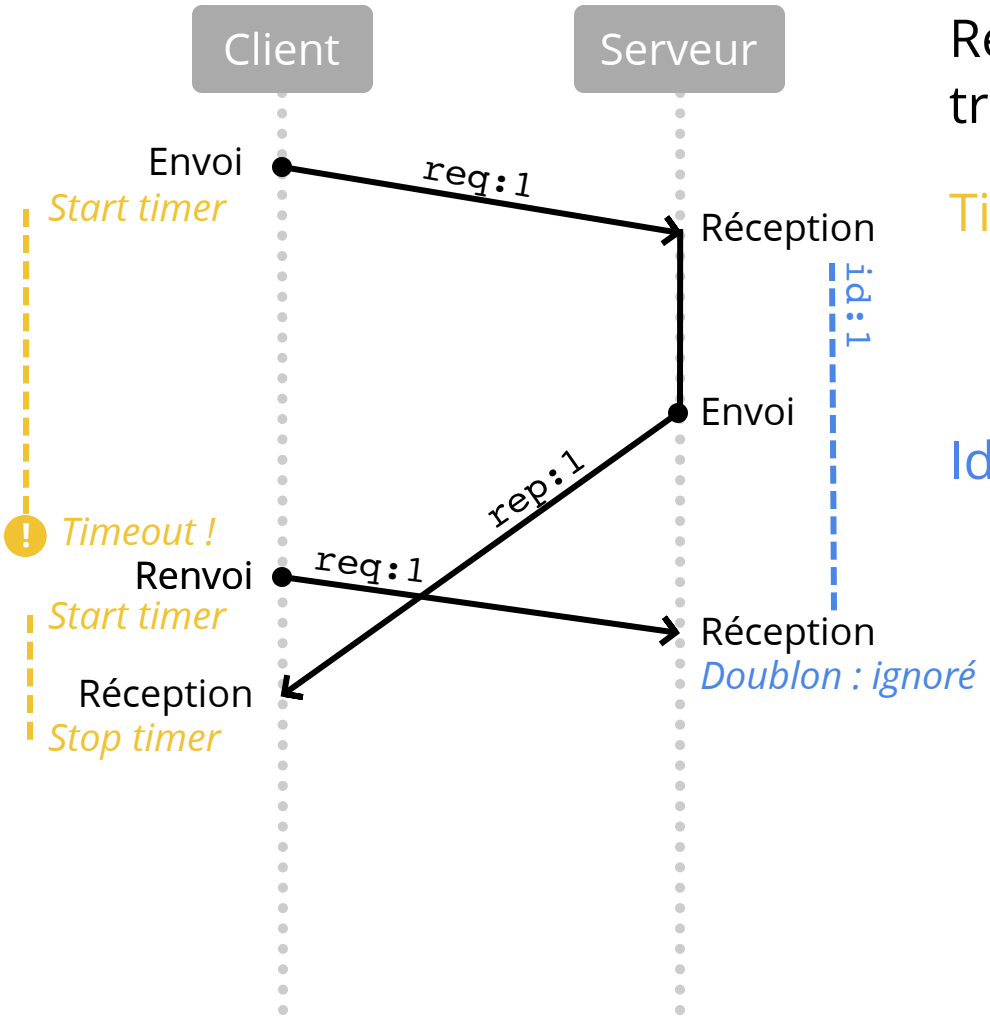
déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

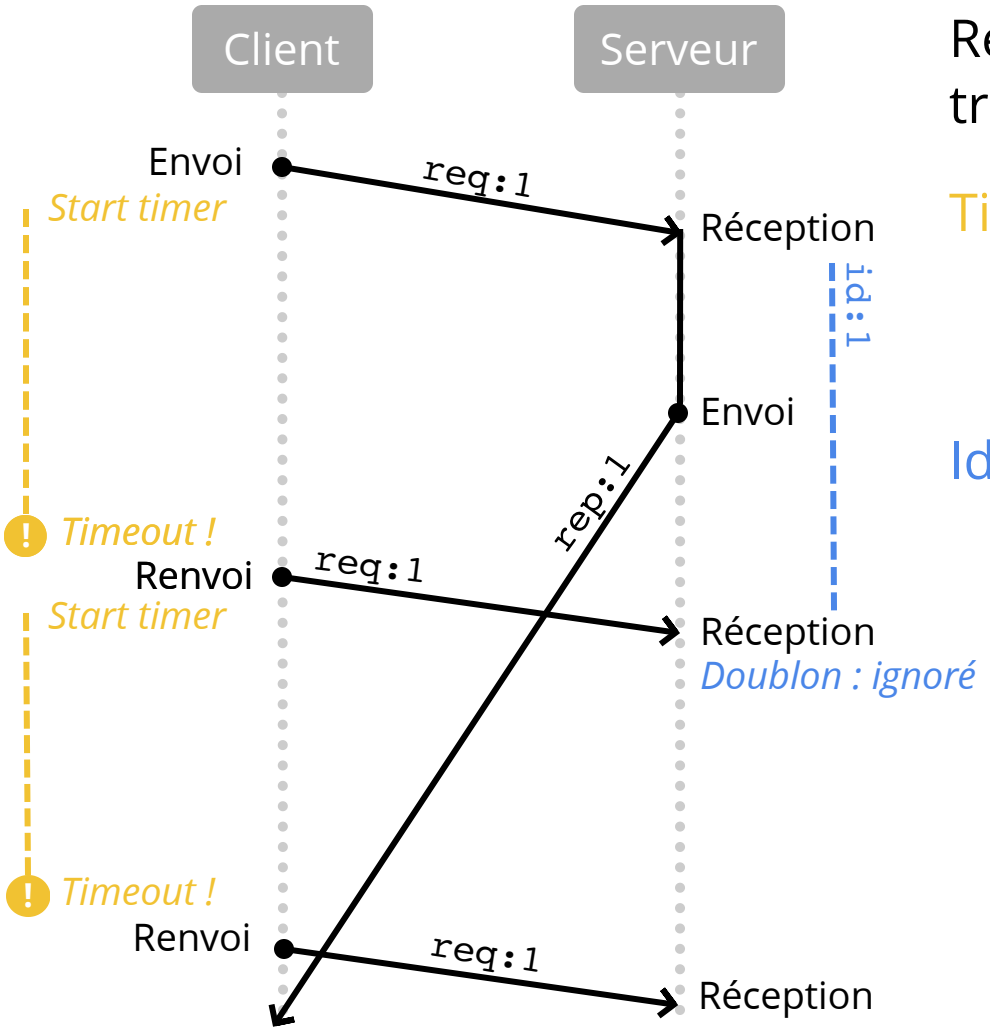
déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké **indéfiniment** ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

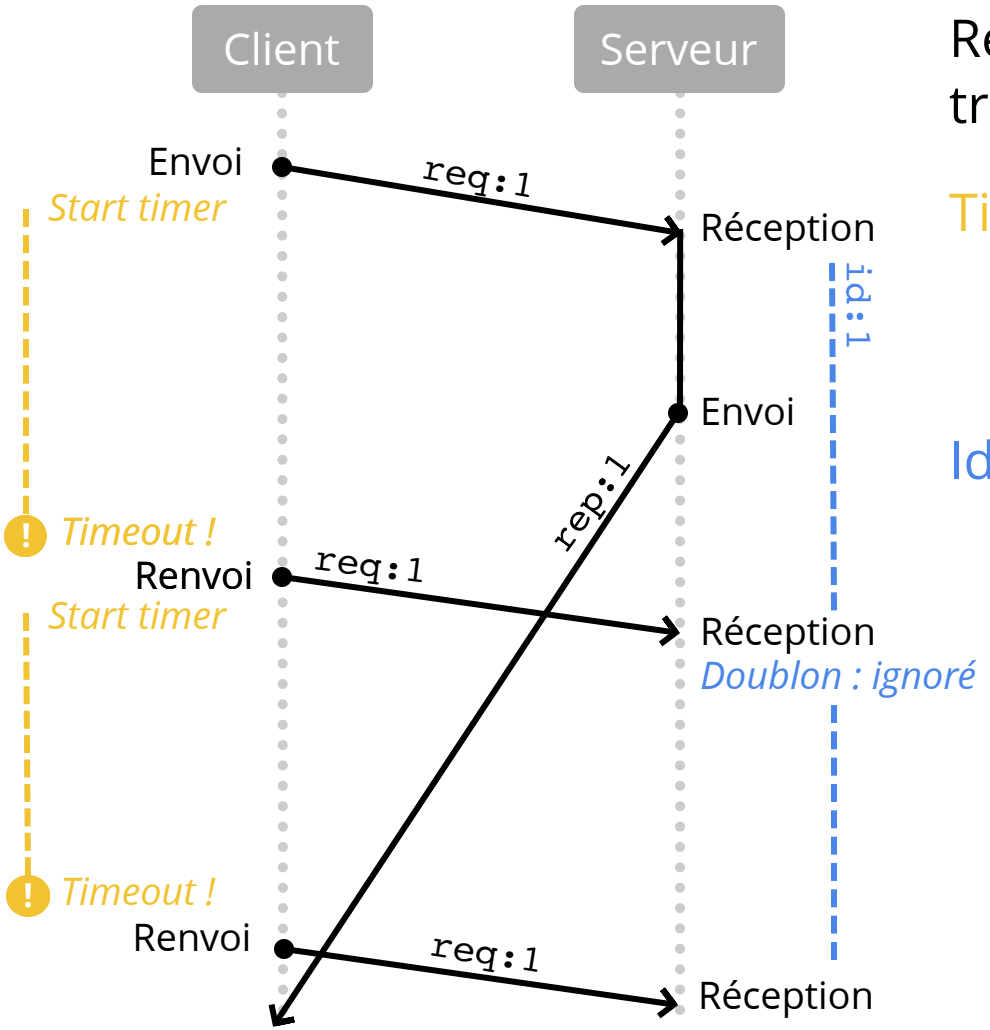
déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké **indéfiniment** ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

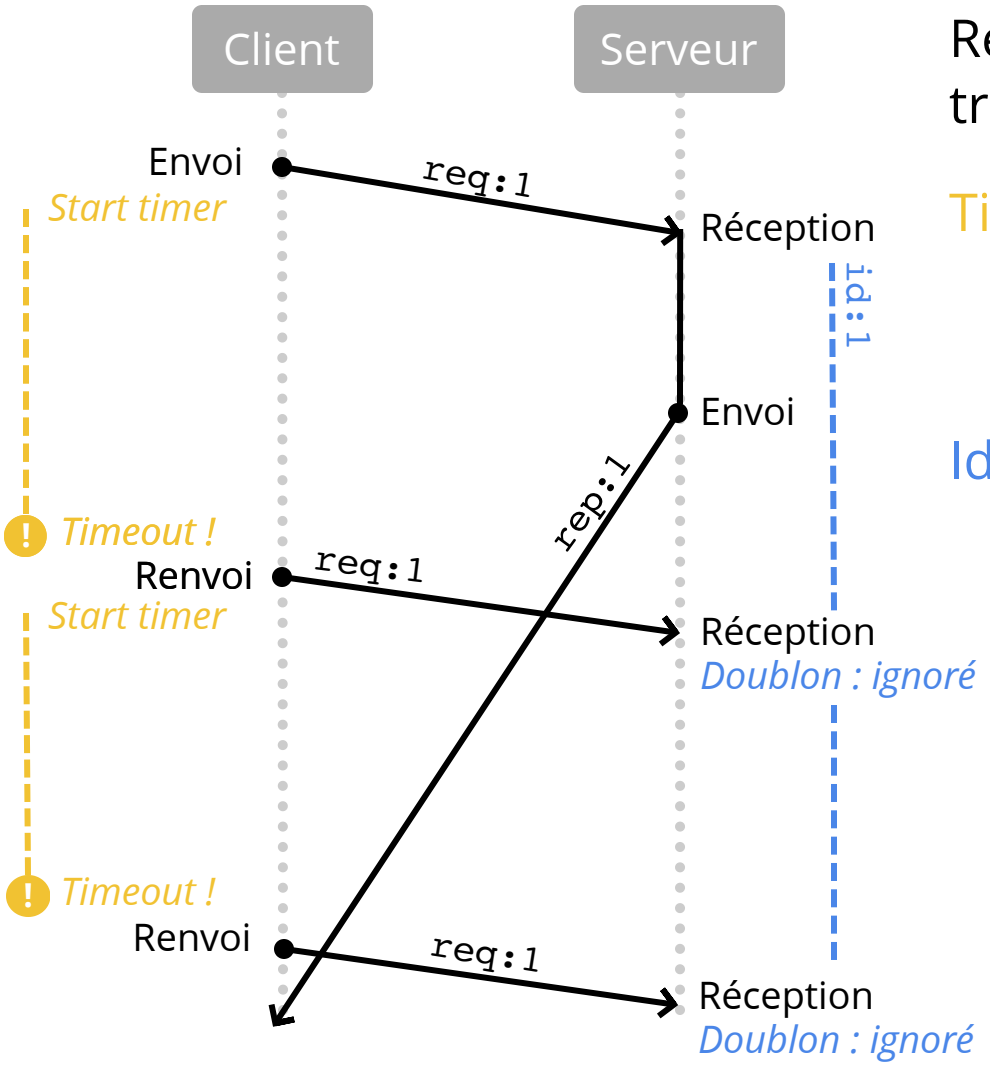
déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké **indéfiniment** ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

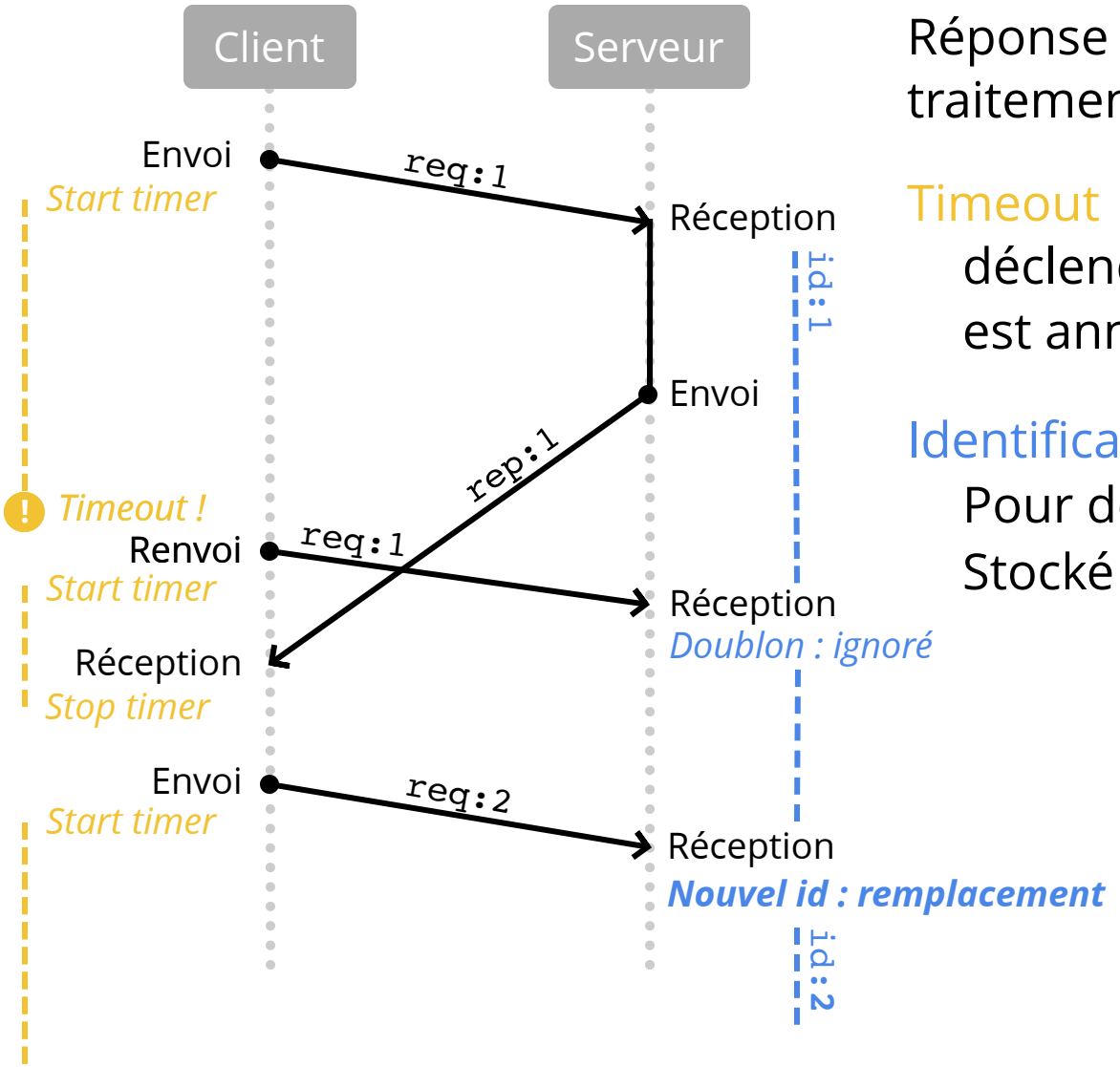
déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké **indéfiniment** ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

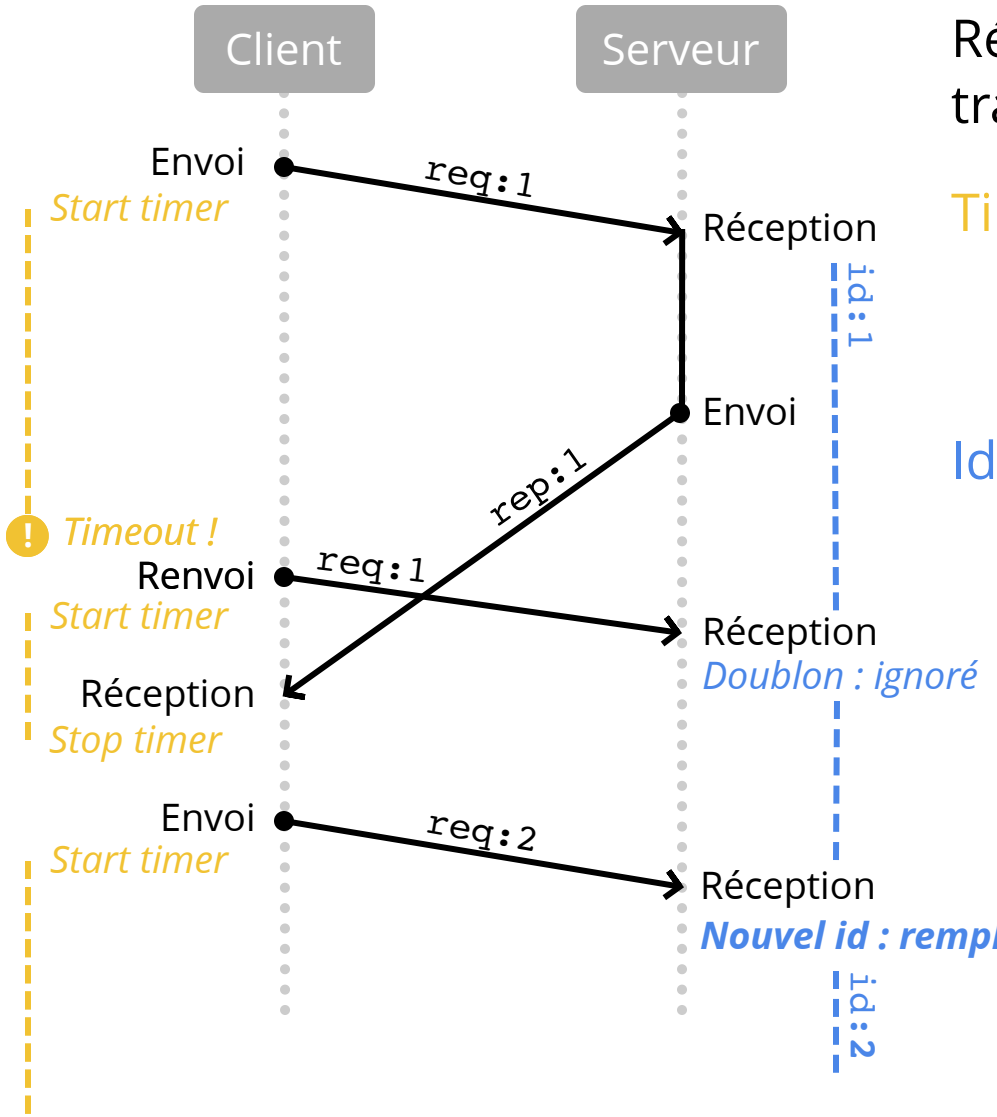
déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

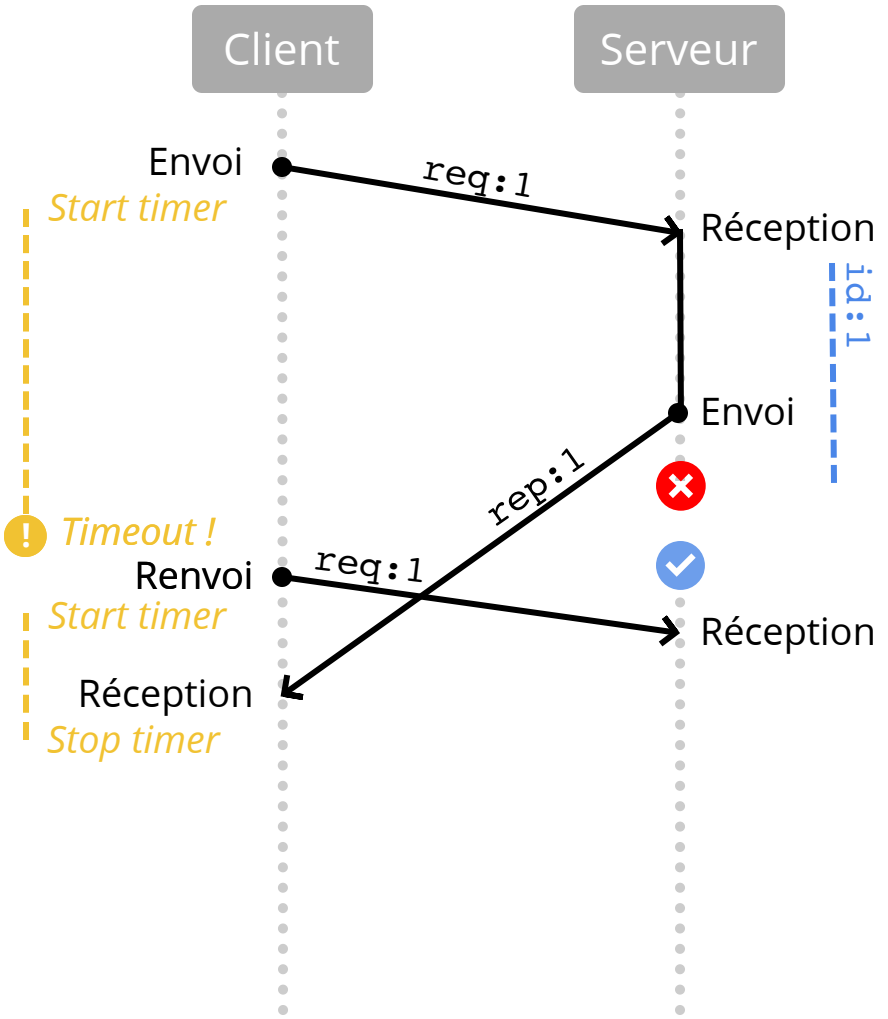
Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

- déclenche un renvoi.
- est annulé à la réception.

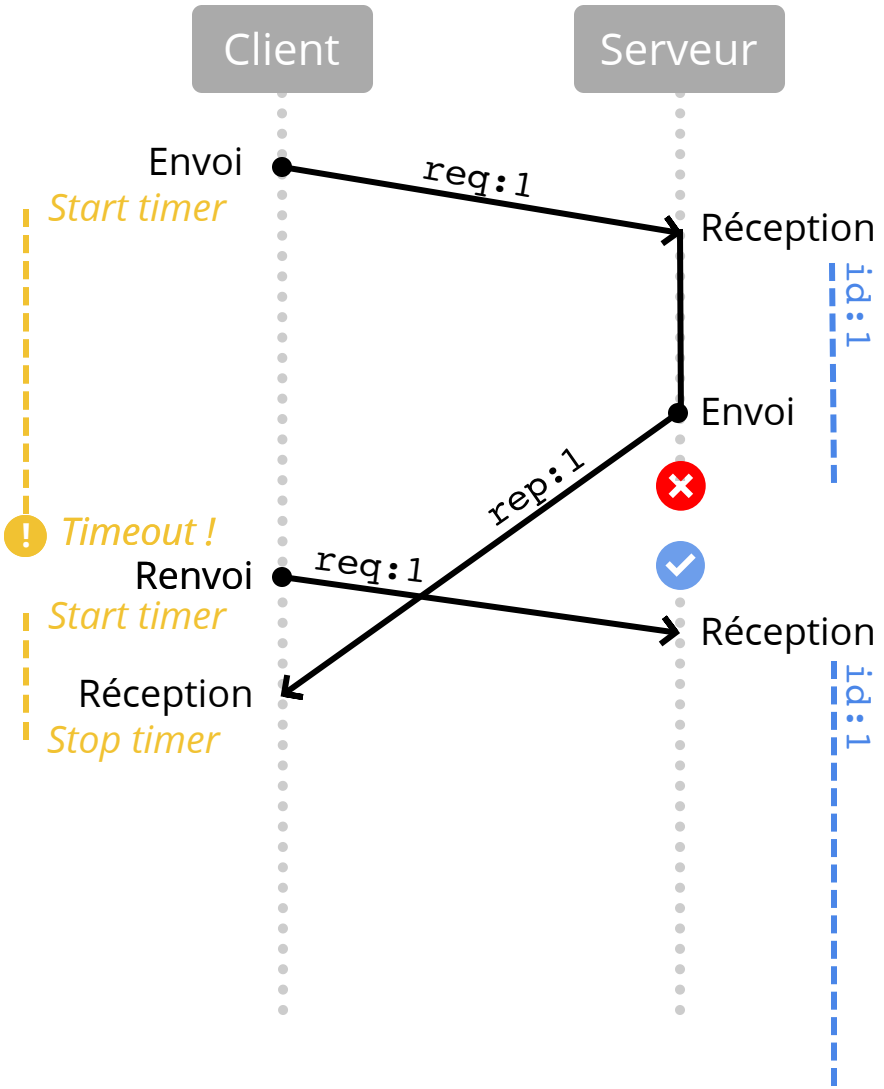
Identificateur

- Pour détecter les doublons.
- Stocké jusqu'à nouveau message.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

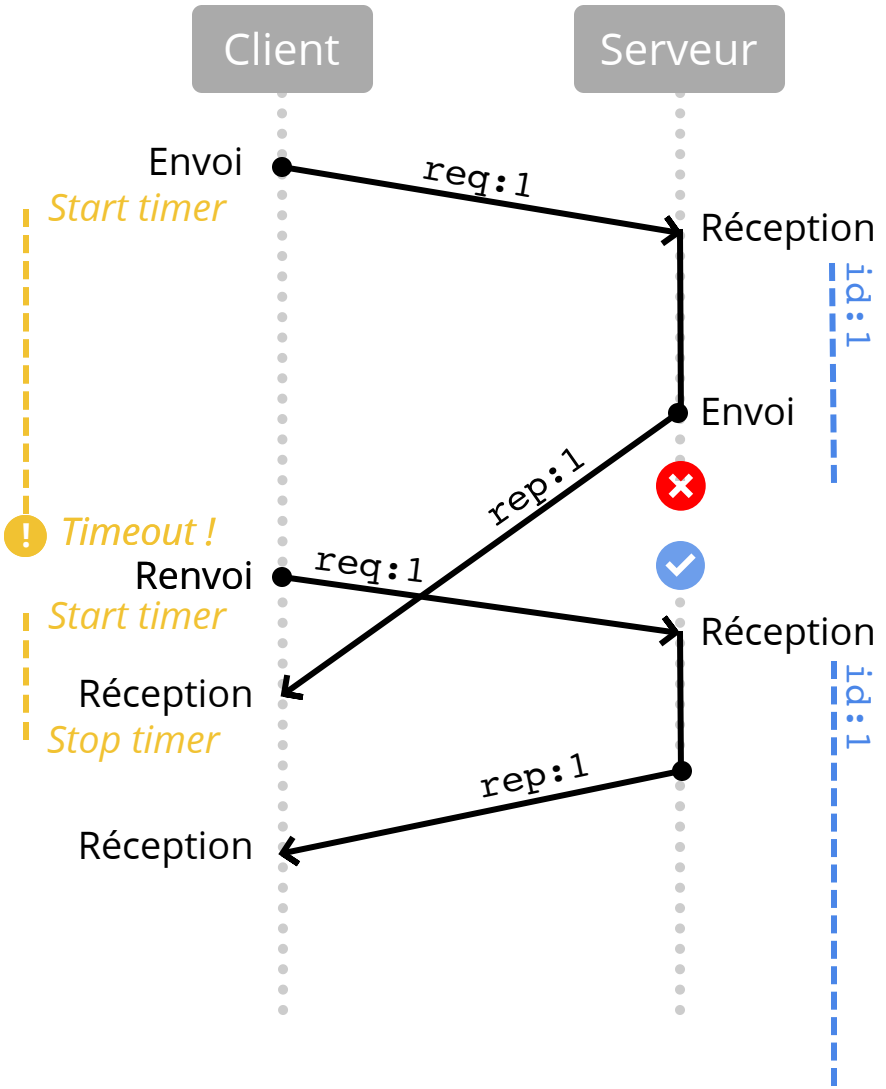
Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

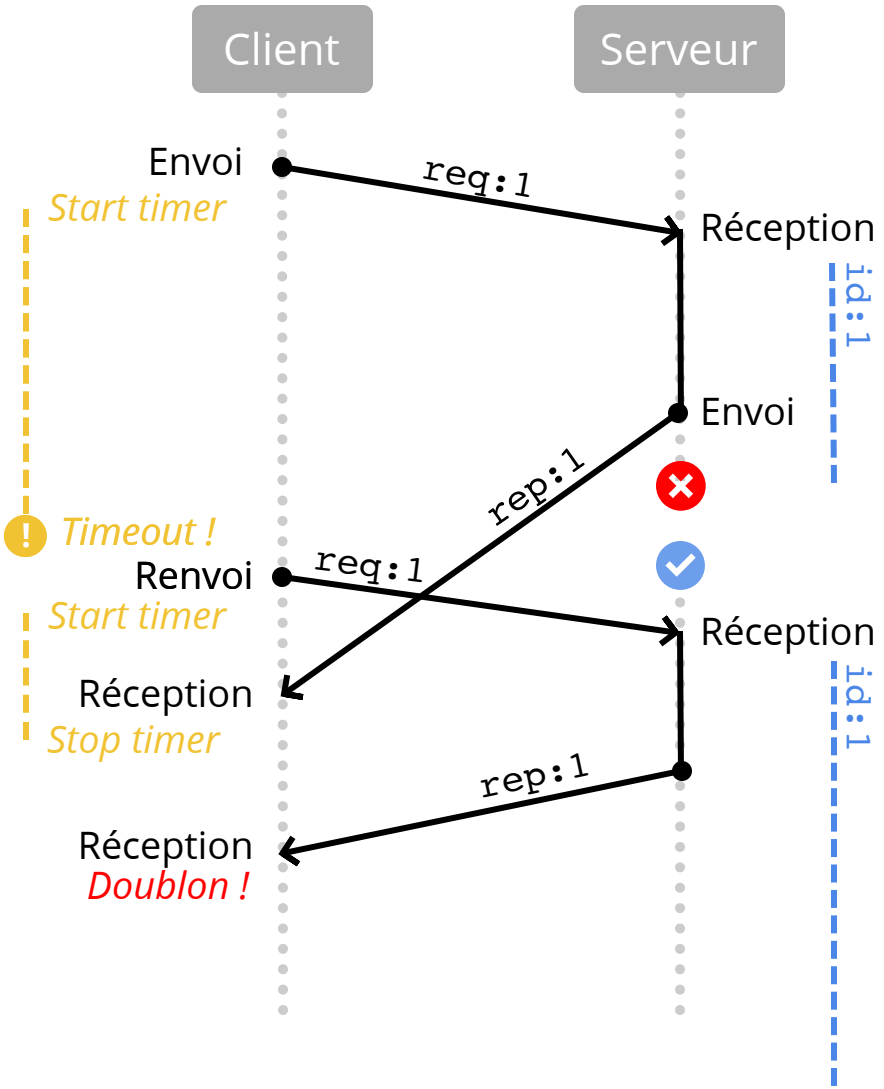
Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

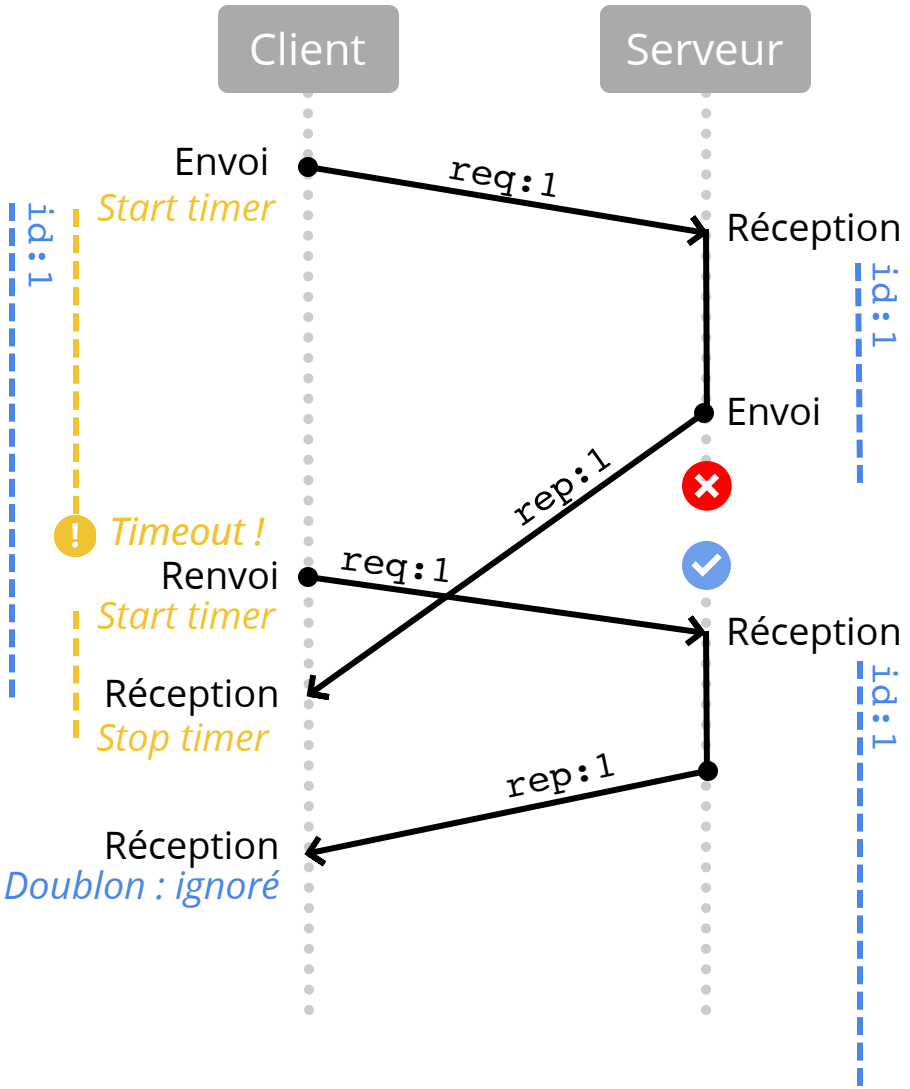
Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message.

Problème ?
Risque de reception double.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.

est annulé à la réception.

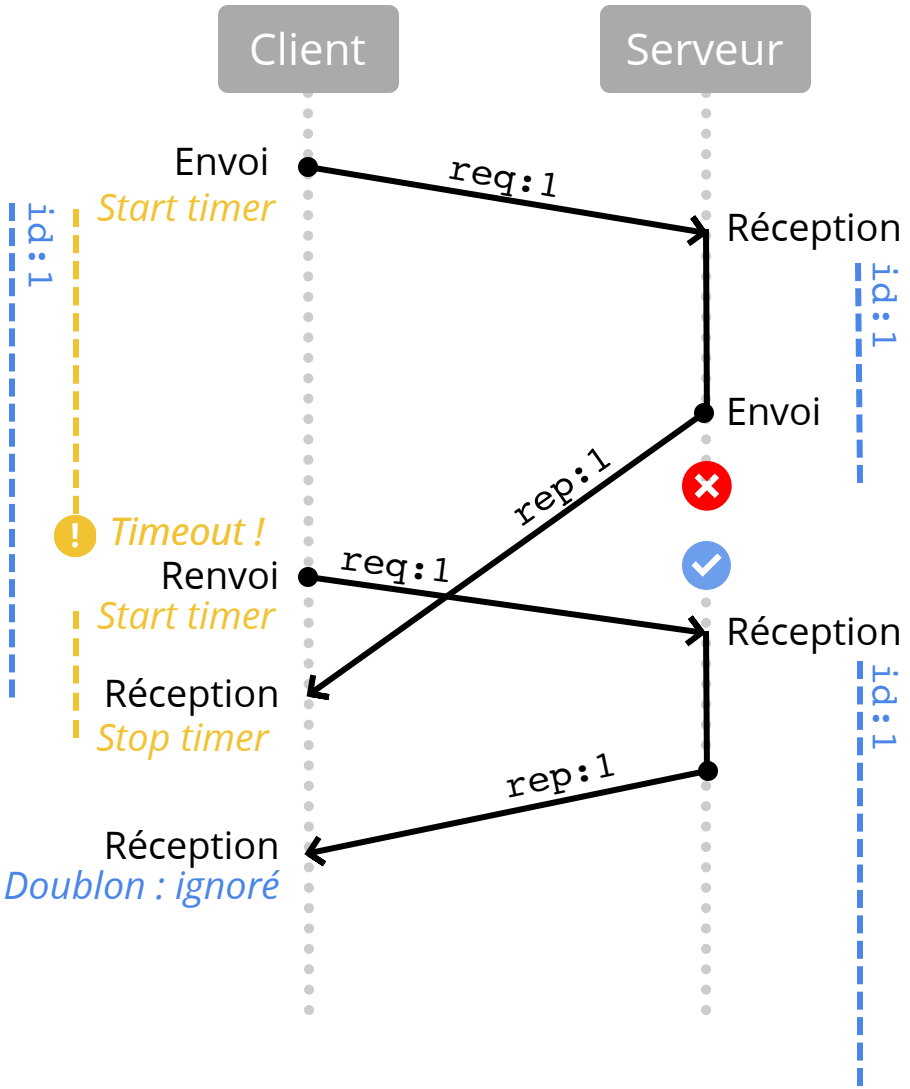
Identificateur

Pour détecter les doublons.

Stocké jusqu'à nouveau message dans serveur *et* client.

Protocoles de fiabilité

Protocole **RR** (Request-Reply) *(version "Au plus une fois")*



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.

est annulé à la réception.

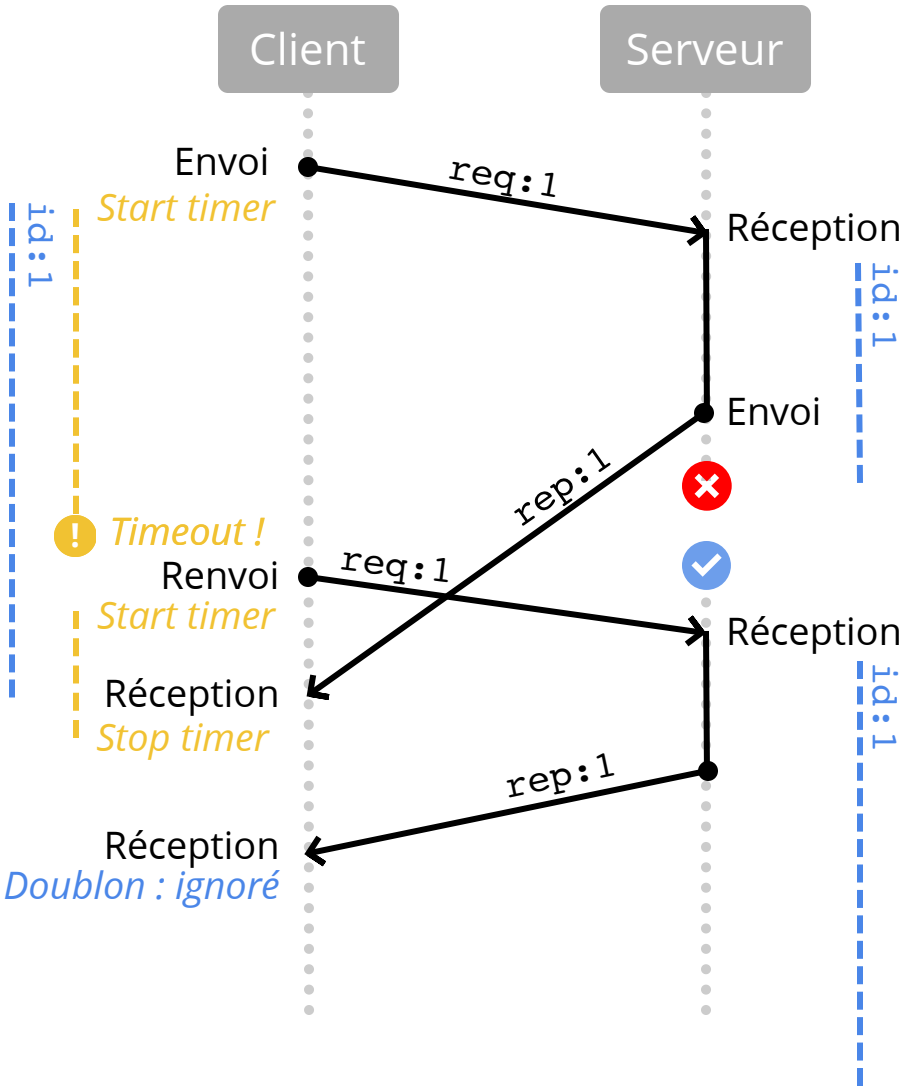
Identificateur

Pour détecter les doublons.

Stocké jusqu'à nouveau message dans serveur *et* client.

Protocoles de fiabilité

Protocole **RR** (Request-Reply) *(version "Au plus une fois")*



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

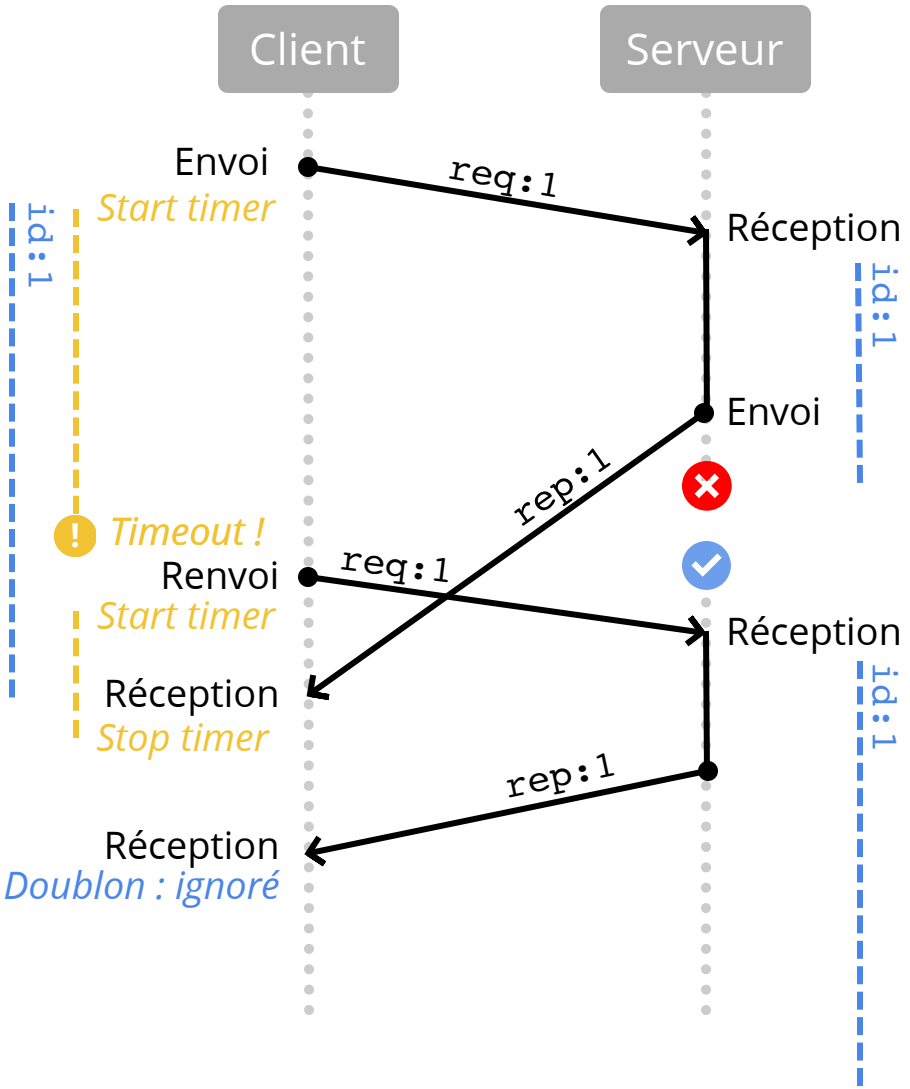
Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message dans serveur *et* client.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply) *(version "Au plus une fois")*



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message dans serveur *et* client.

Problème ?

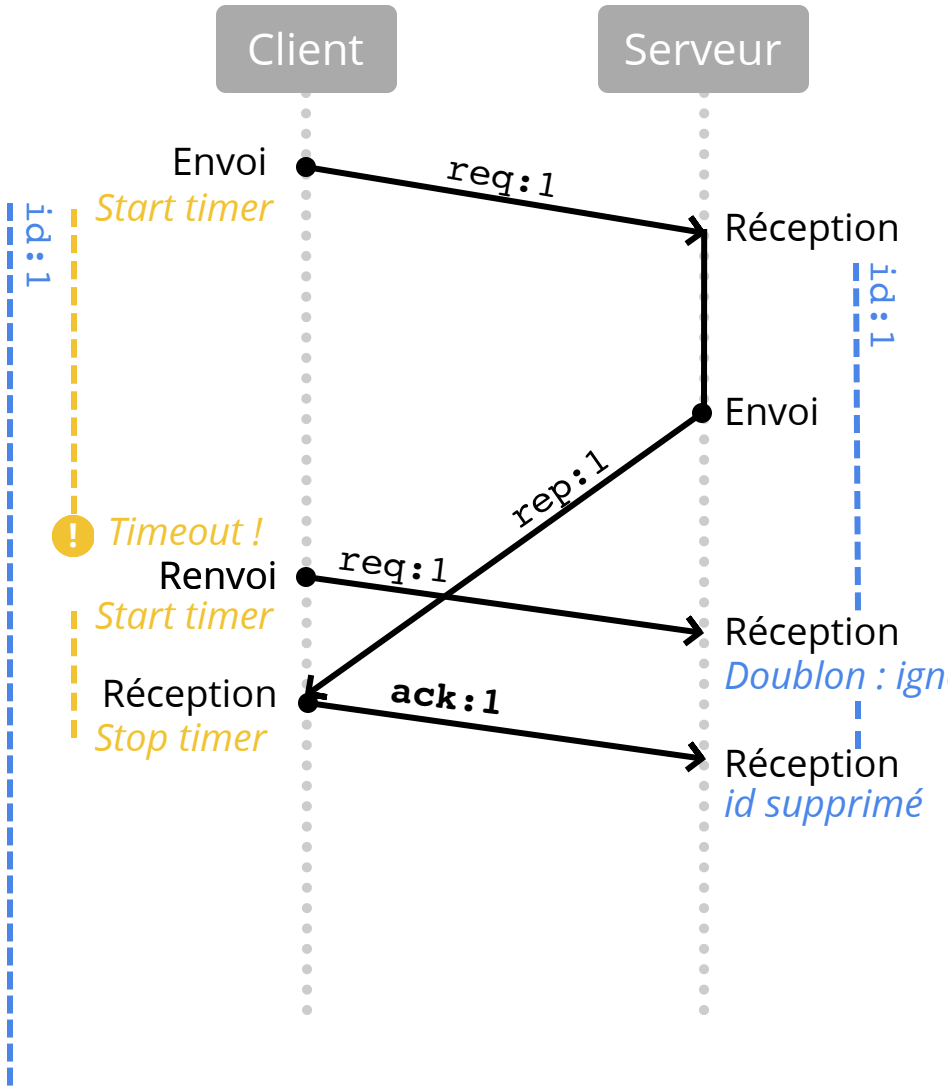
Si requêtes rares mais beaucoup de clients :
trop d'identificateurs à gérer.

Protocole Request-Reply-Acknowledge

(a.k.a. *at-most-once*, avec libération de l'état serveur)

Protocoles de fiabilité

Protocole **RRA** (Request-Reply-Acknowledge)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

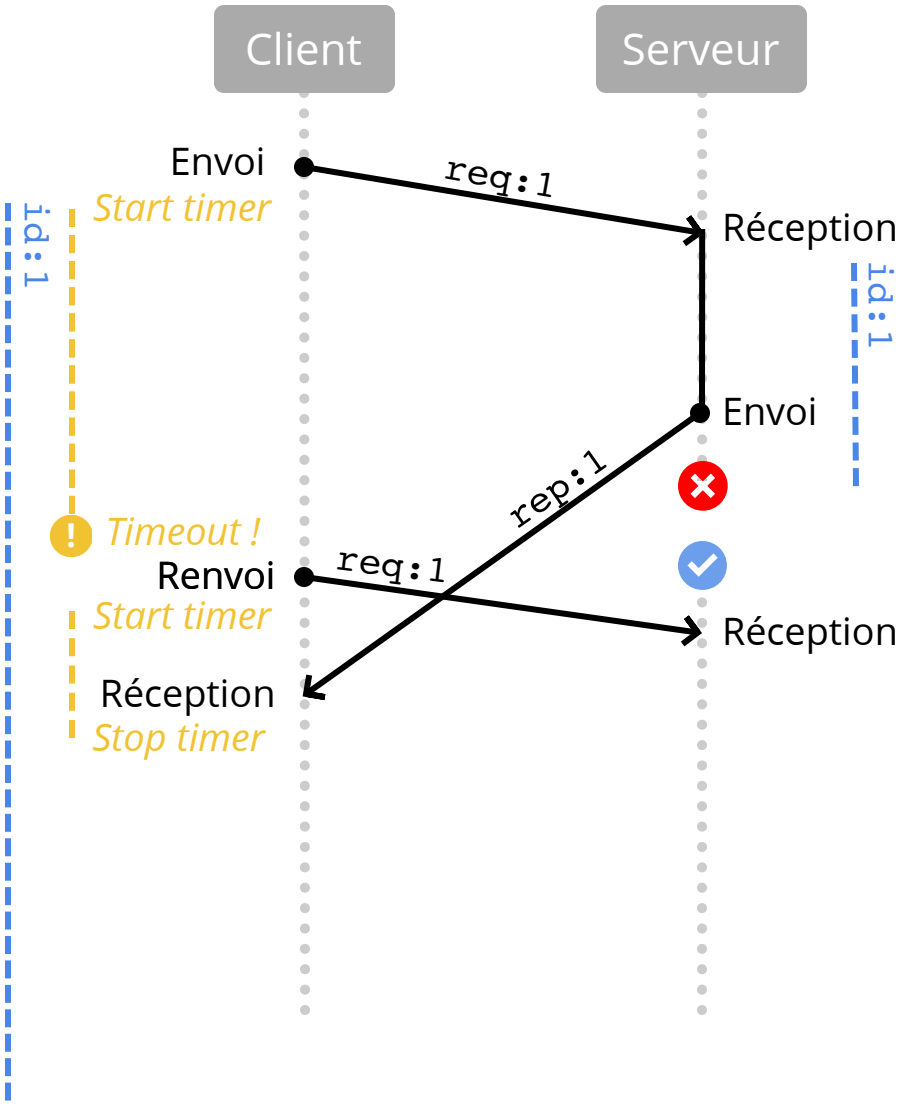
Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

Acknowledgment

Notifie le serveur
de la réception d'une réponse

Protocoles de fiabilité

Protocole **RRA** (Request-Reply-Acknowledge)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

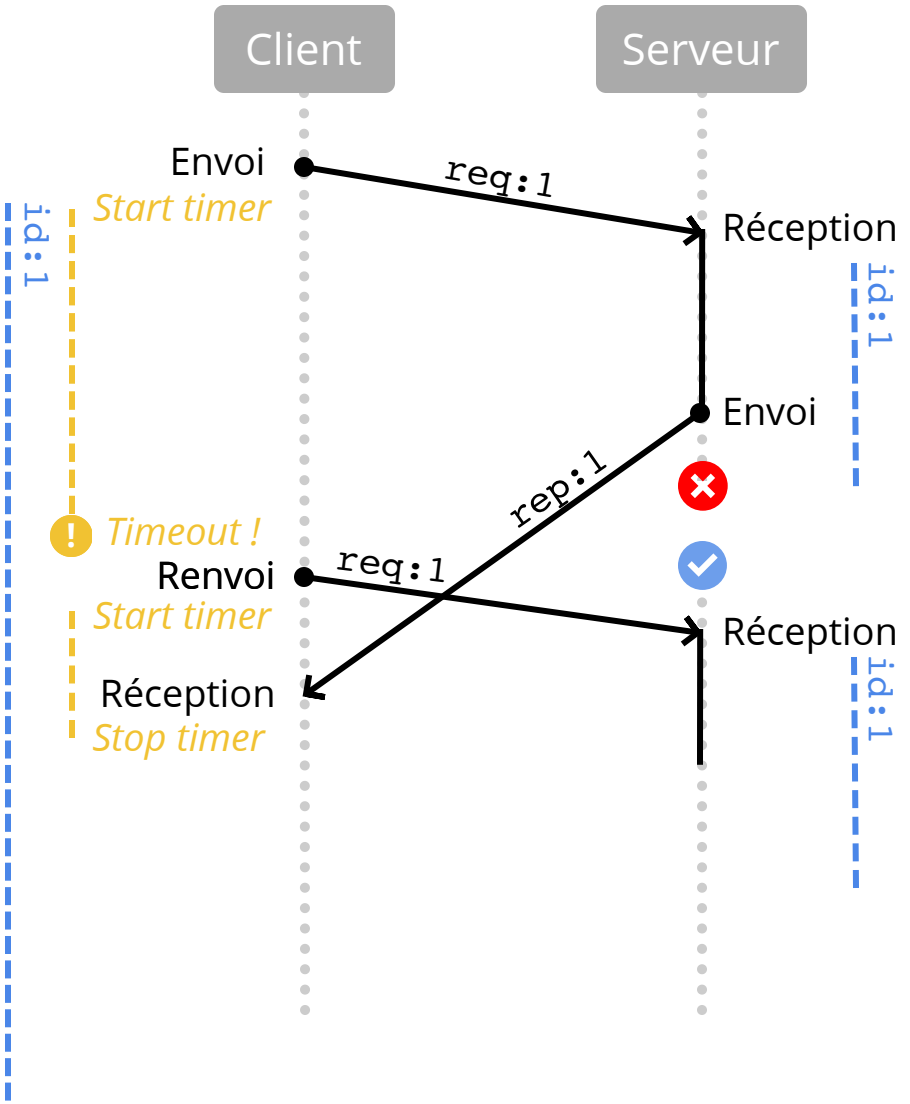
Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

Acknowledgment

Notifie le serveur
de la réception d'une réponse

Protocoles de fiabilité

Protocole **RRA** (Request-Reply-Acknowledge)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

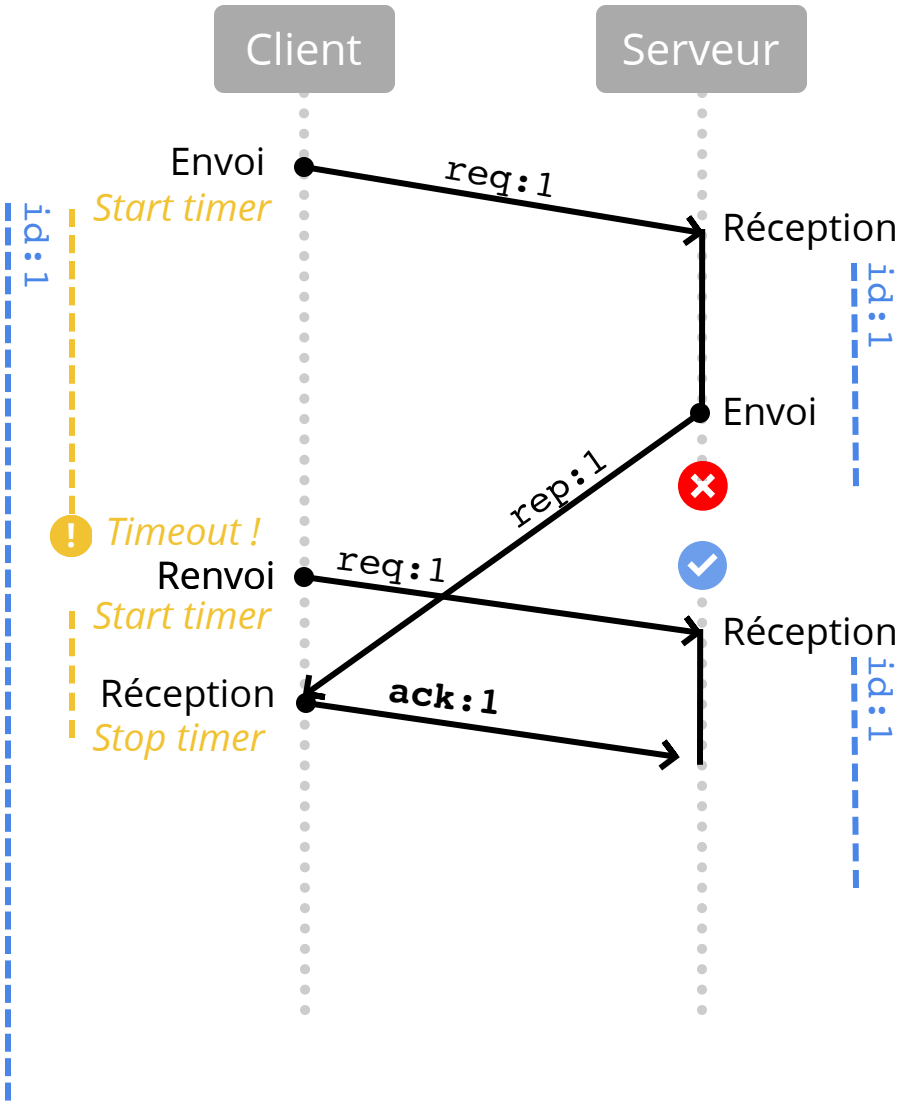
Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

Acknowledgment

Notifie le serveur
de la réception d'une réponse

Protocoles de fiabilité

Protocole **RRA** (Request-Reply-Acknowledge)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

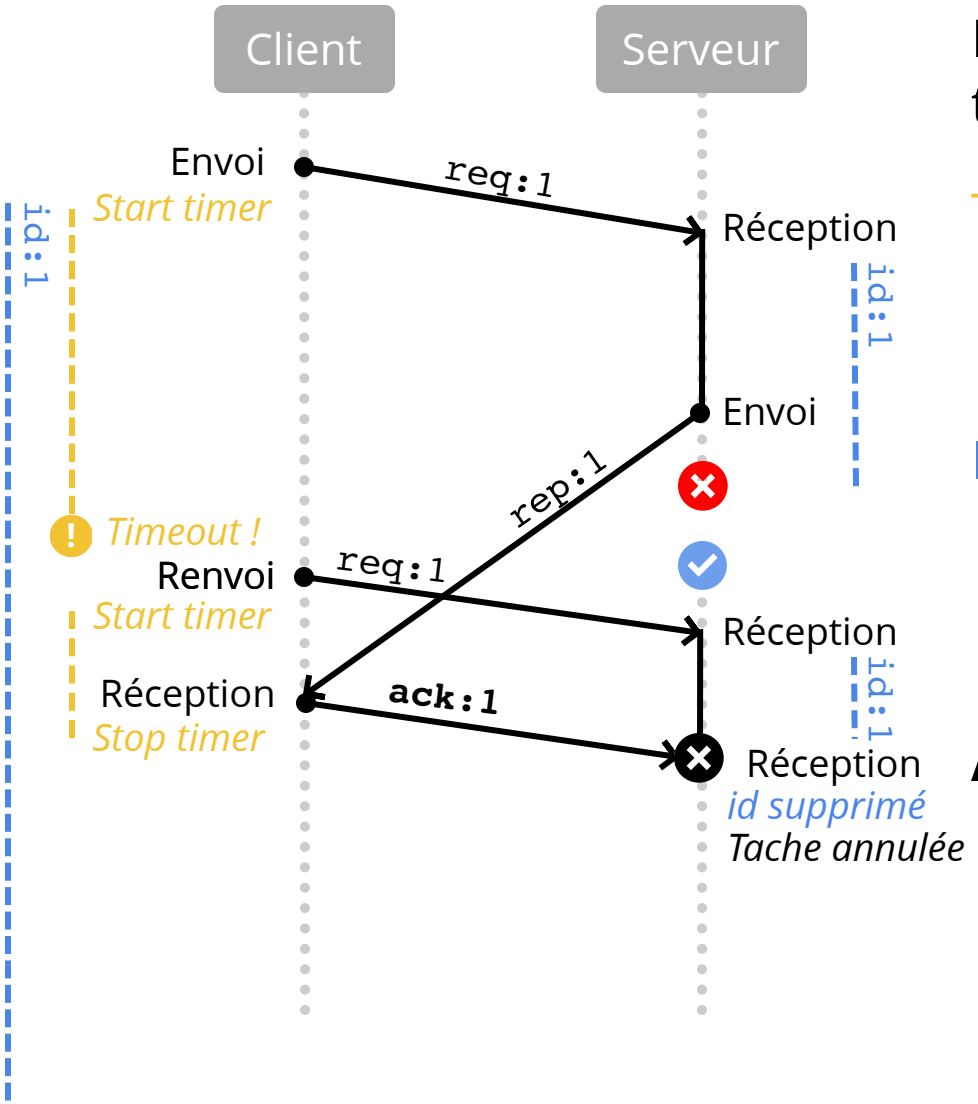
Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

Acknowledgment

Notifie le serveur
de la réception d'une réponse

Protocoles de fiabilité

Protocole **RRA** (Request-Reply-Acknowledge)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

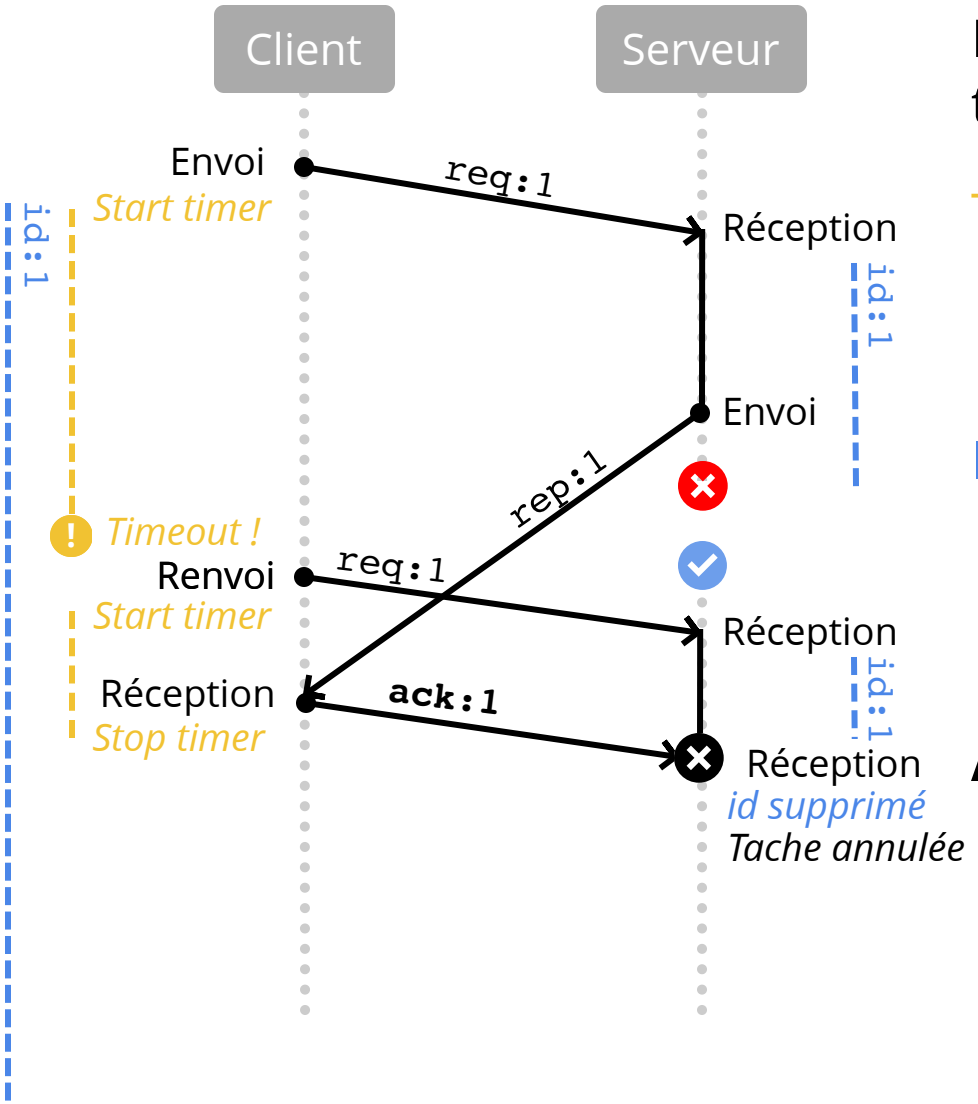
Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

Acknowledgment

Notifie le serveur
de la réception d'une réponse

Protocoles de fiabilité

Protocole **RRA** (Request-Reply-Acknowledge)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

Identificateur

Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

Acknowledgment

Notifie le serveur
de la réception d'une réponse, *et*
de messages oubliés.

Exercice

Question 1

Supposez que le réseau ne nous garantit plus l'absence de perte de message. Quelles sont alors les garanties données par les différentes classes de fiabilité ? Pour répondre, notez pour chacun des 4 protocoles, s'il y a ou non risque de

- absence de traitement d'une requête
- duplication de traitement d'une requête
- absence de confirmation d'une requête
- duplication de confirmation de traitement

et si oui, un exemple de scénario qui peut le causer.

Si de nouveaux problèmes surgissent pour RR avec id et RRA, quelle solution proposez-vous ?

Question 2

Que se passe-t-il dans RR version "exactement une fois" si le traitement est très long ?

Question 3

Que se passe-t-il dans RR version "exactement une fois" si le client tombe en panne temporairement avant d'avoir reçu une réponse ? Quelle solution proposer ?

Protocoles de fiabilité

Résumé

R maybe

"peut-être"

RR at-least-once

"au moins une fois"

RR + id at-most-once

"exactement une fois"

RRA at-most-once

"au moins une fois" + libère l'état serveur

Speaker notes

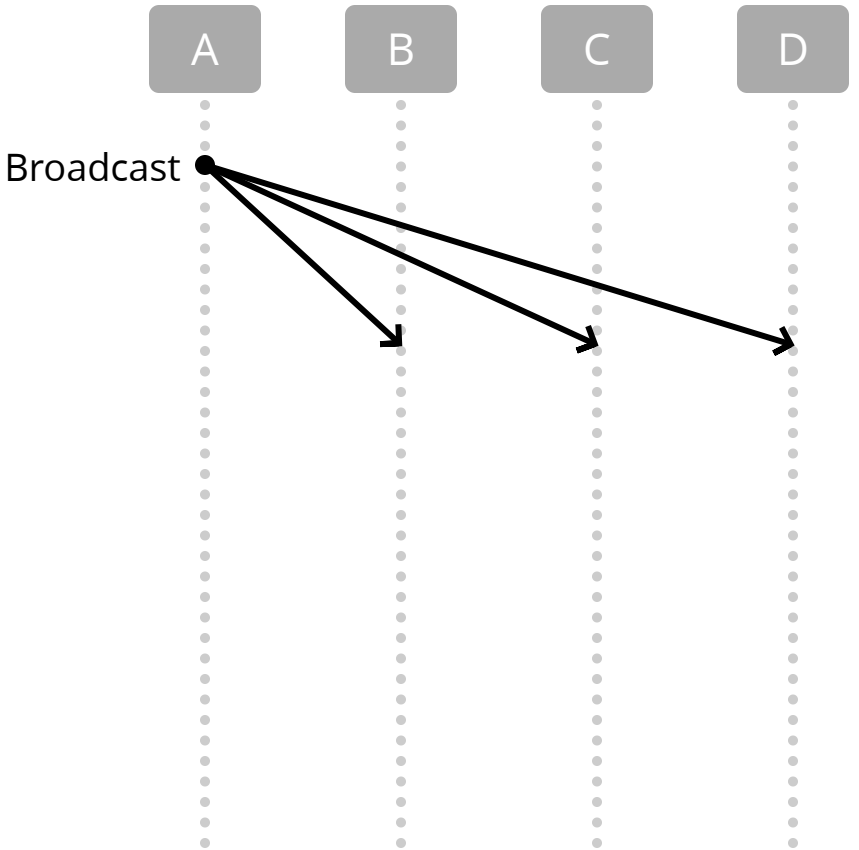
La discordance de vocabulaire : ce que le cours appelle « exactement une fois » est at-most-once dans la littérature, parce que si le client tombe en panne avant de recevoir la réponse, l'appel peut n'avoir jamais été exécuté. Et l'ordre : aucun des quatre protocoles ne transporte de numéro de séquence.

Reliable Broadcast

Une diffusion fiable

Broadcast

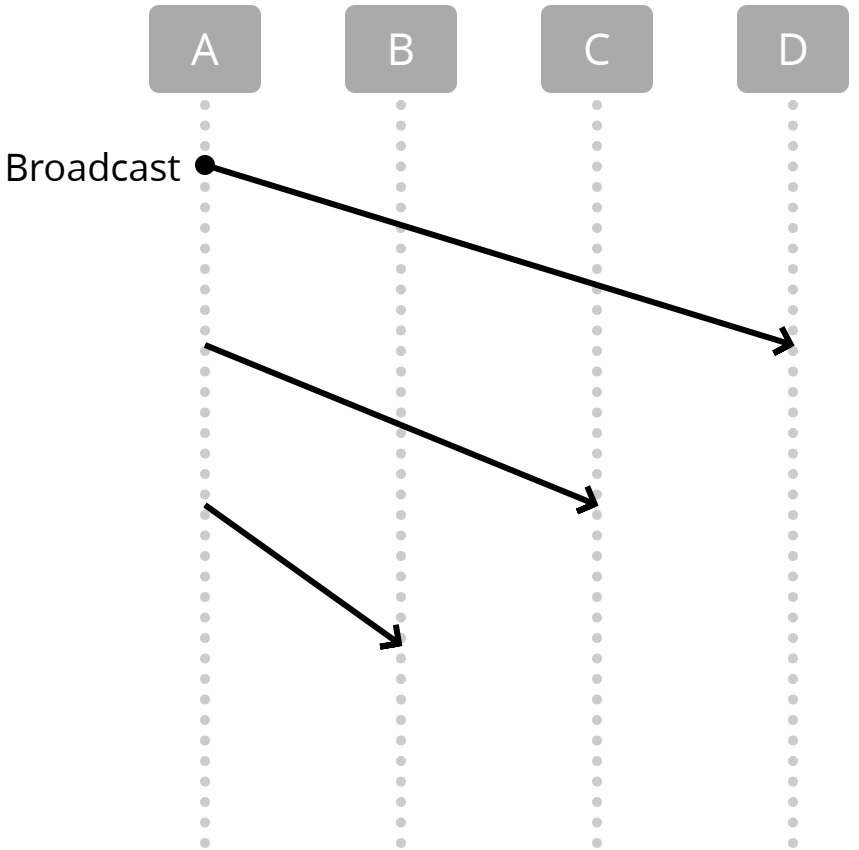
Le plus simple : Best effort



Envoyer à tous les processus.

Broadcast

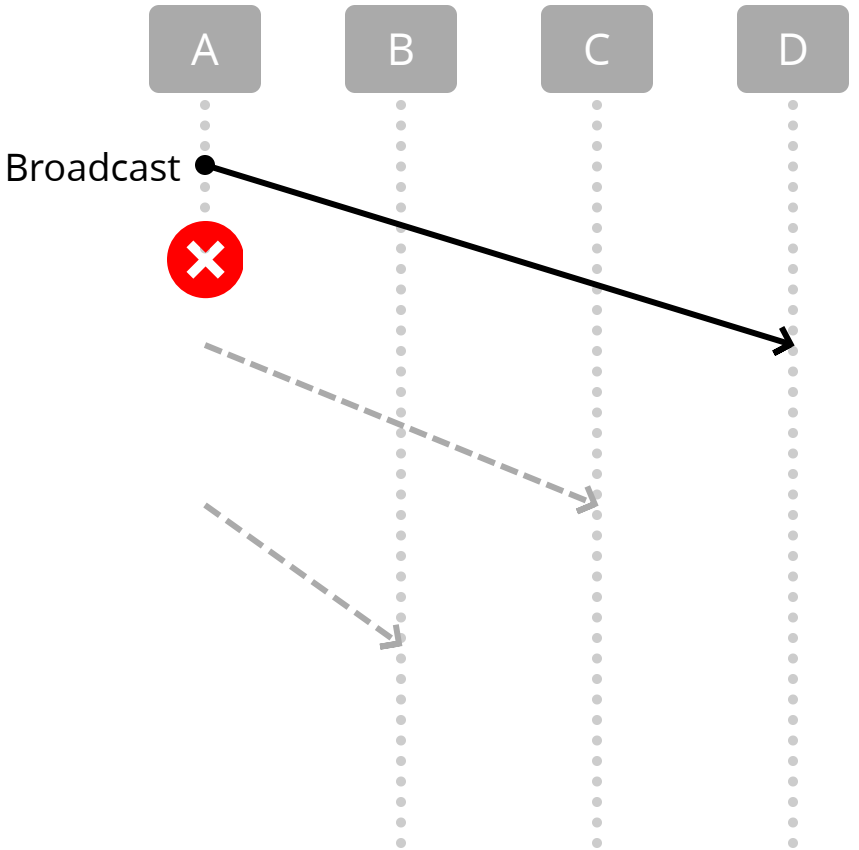
Le plus simple : Best effort



Envoyer à tous les processus.

Broadcast

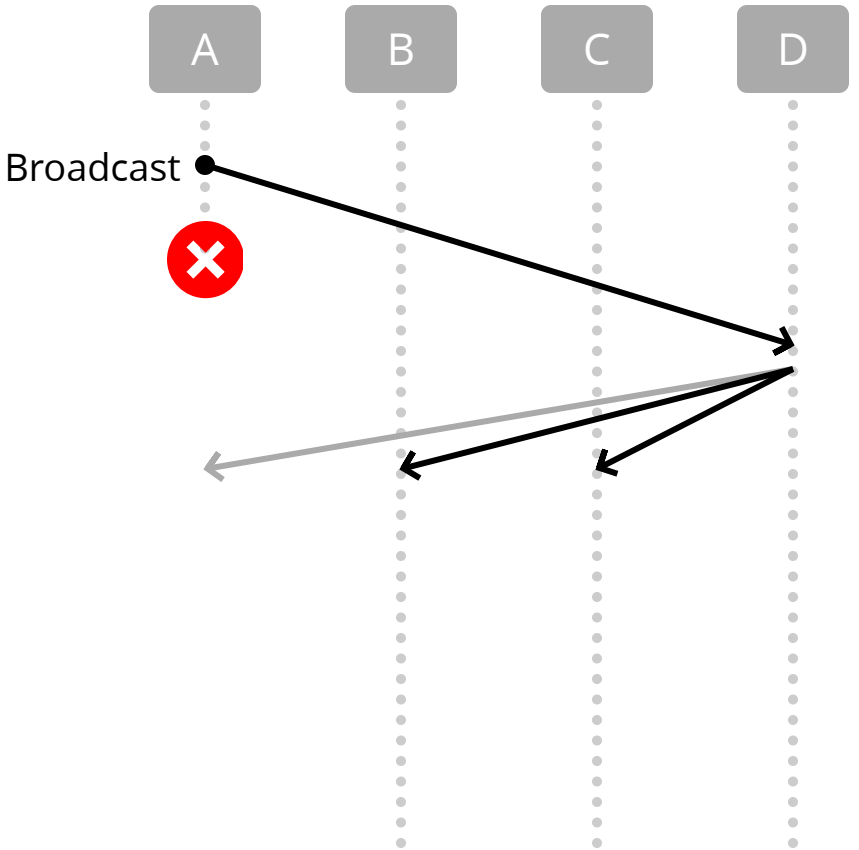
Le plus simple : Best effort



Envoyer à tous les processus.

Broadcast

Tolérance aux pannes : Reliable Broadcast

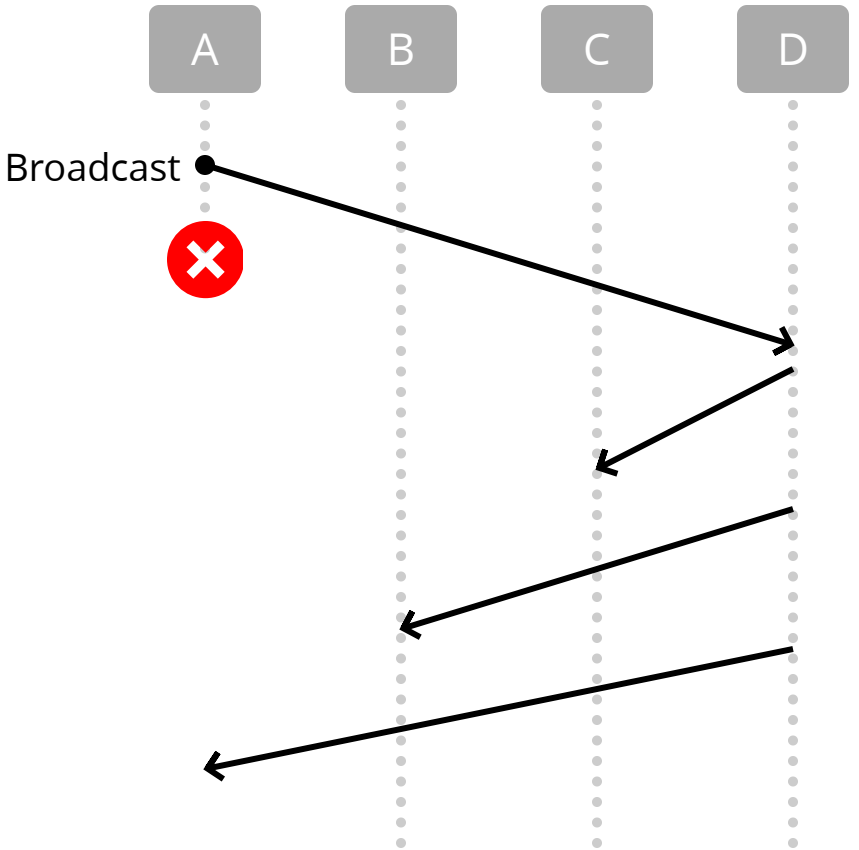


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,

Broadcast

Tolérance aux pannes : Reliable Broadcast

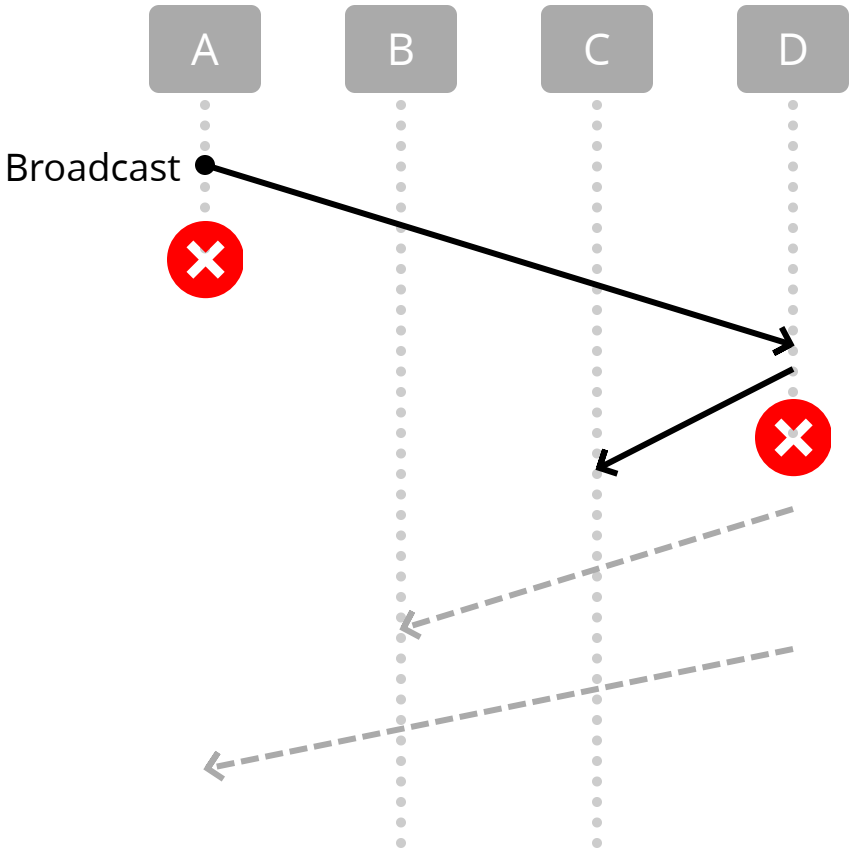


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,

Broadcast

Tolérance aux pannes : Reliable Broadcast

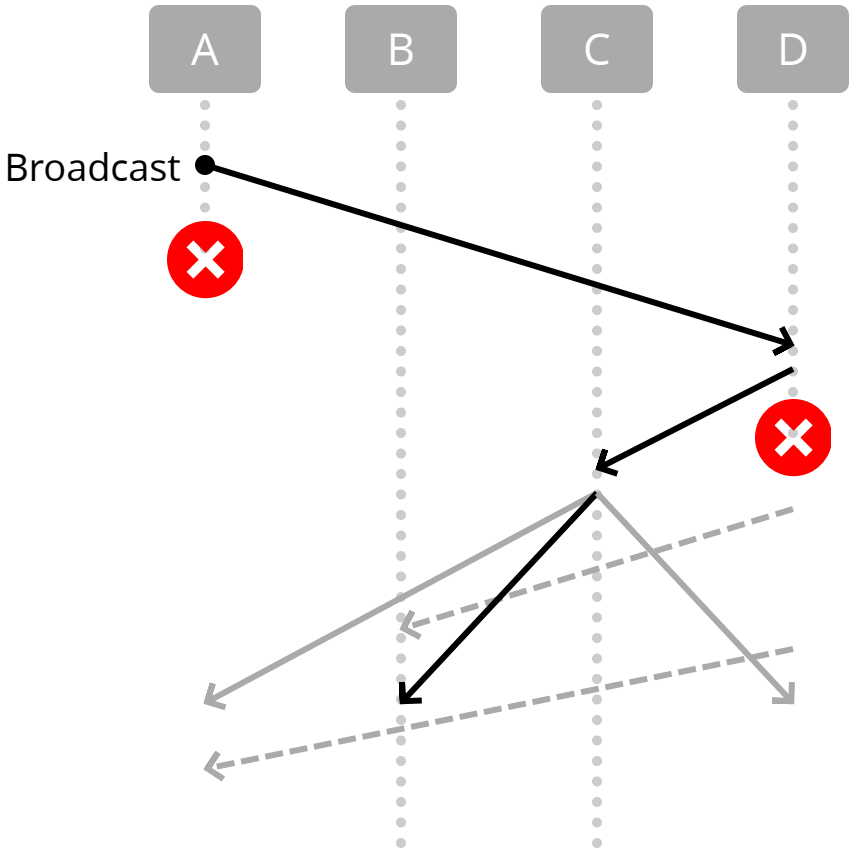


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,

Broadcast

Tolérance aux pannes : Reliable Broadcast

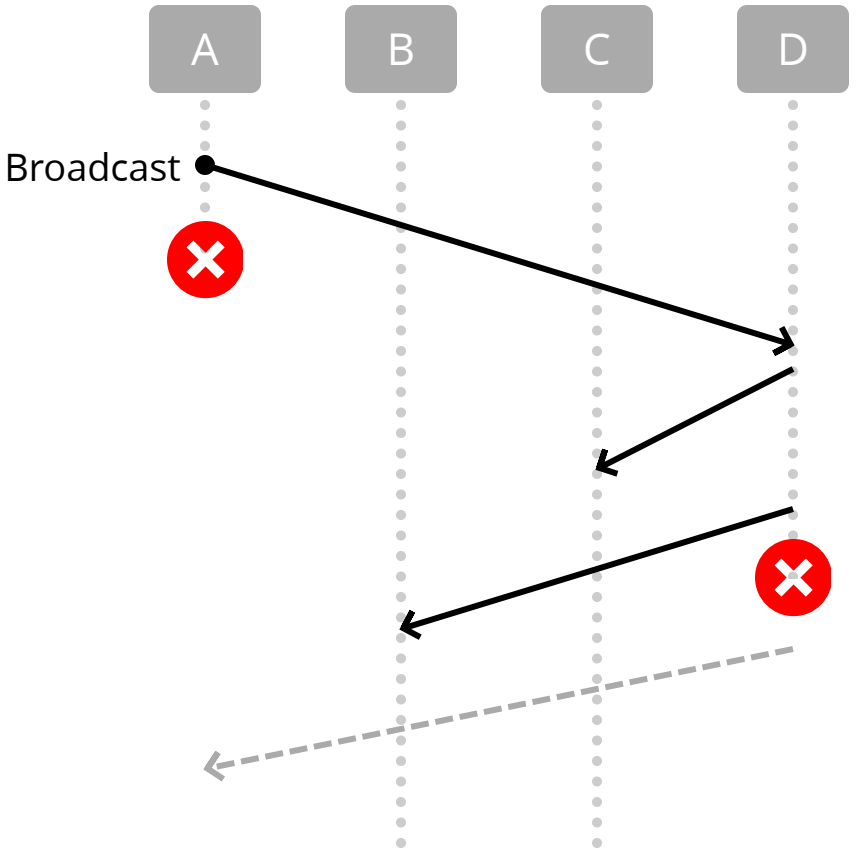


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,
peu importe d'où il vient,

Broadcast

Tolérance aux pannes : Reliable Broadcast

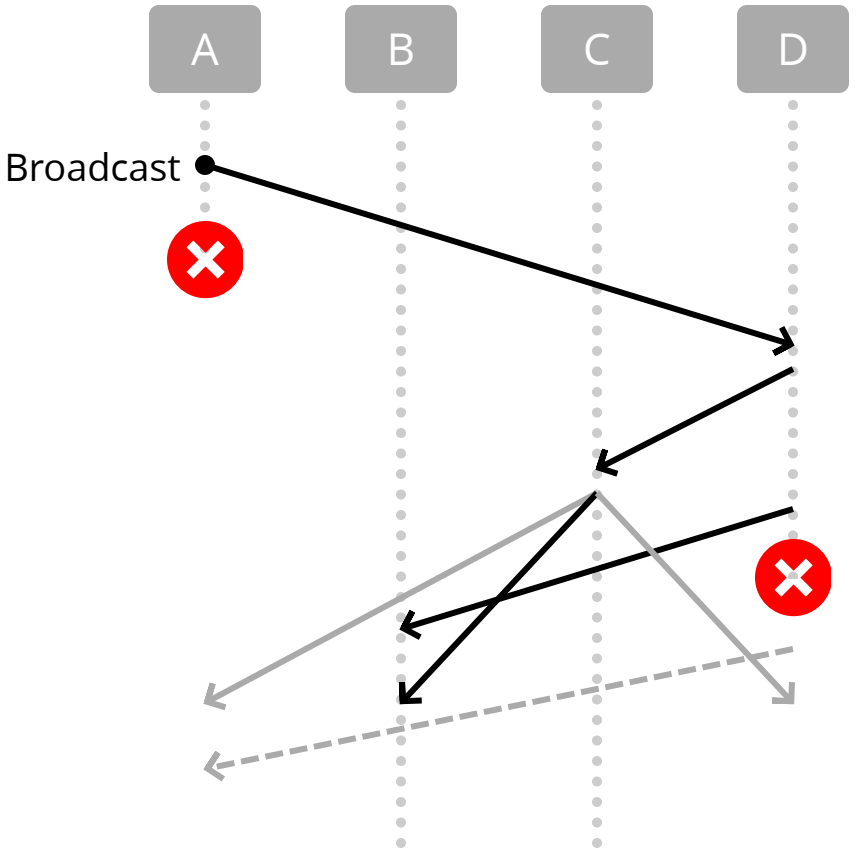


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,
peu importe d'où il vient,

Broadcast

Tolérance aux pannes : Reliable Broadcast

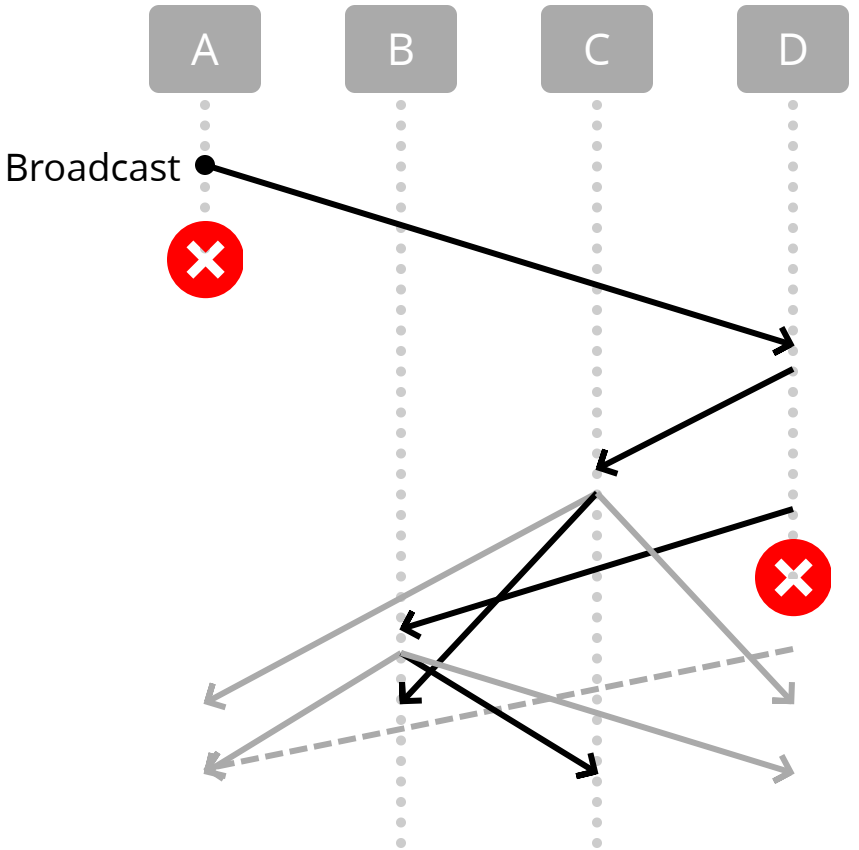


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,
peu importe d'où il vient,

Broadcast

Tolérance aux pannes : Reliable Broadcast

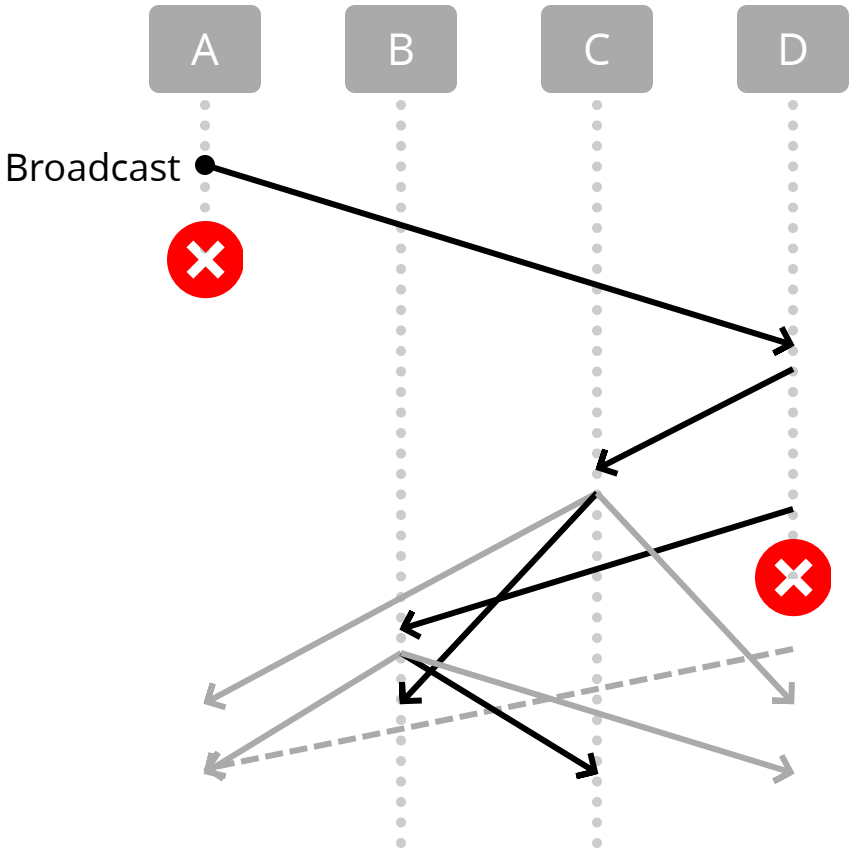


Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,
peu importe d'où il vient,

Broadcast

Tolérance aux pannes : Reliable Broadcast



Envoyer à tous les processus.

Lorsqu'on reçoit un message,
le renvoyer à tous,
peu importe d'où il vient,
sauf si déjà reçu.

Reliable Broadcast

Propriétés garanties

Reliable Broadcast

Propriétés garanties

Validité Si un processus correct envoie m, alors il finit par délivrer m

Reliable Broadcast

Propriétés garanties

Validité Si un processus correct envoie `m`,
alors il finit par délivrer `m`

Accord Si un processus correct délivre `m`,
alors tous les processus corrects délivrent `m`

Reliable Broadcast

Propriétés garanties

Validité Si un processus correct envoie `m`,
alors il finit par délivrer `m`

Accord Si un processus correct délivre `m`,
alors tous les processus corrects délivrent `m`

Non-création Un processus ne délivre `m` que si `m` a été envoyé

Reliable Broadcast

Propriétés garanties

Validité	Si un processus correct envoie `m`, alors il finit par délivrer `m`
Accord	Si un processus correct délivre `m`, alors tous les processus corrects délivrent `m`
Non-création	Un processus ne délivre `m` que si `m` a été envoyé
Non-duplication	Un processus délivre `m` au plus une fois

↓ Détecteur de panne parfait

Requirements

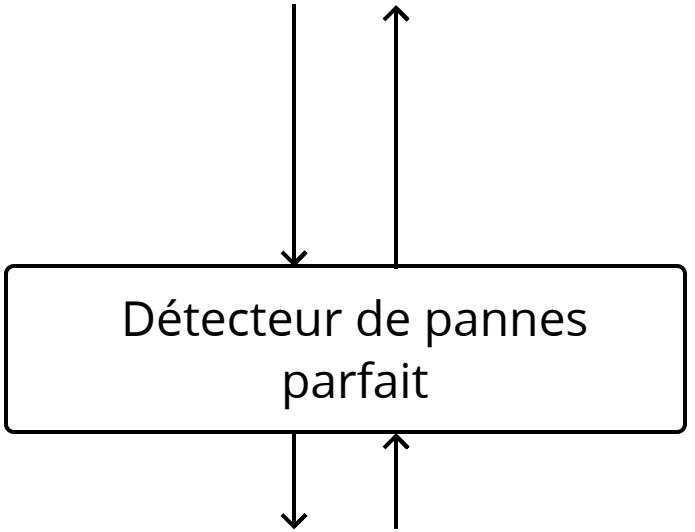
Supposons pour l'instant des pannes permanentes uniquement.

Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.

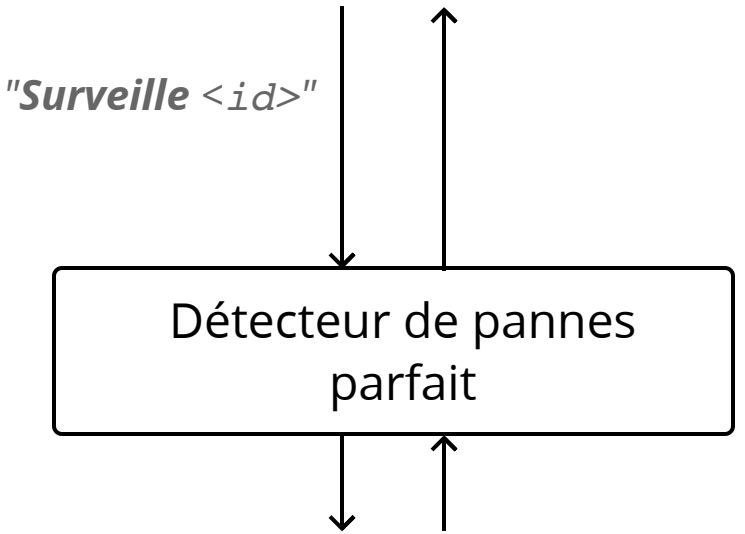


Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.

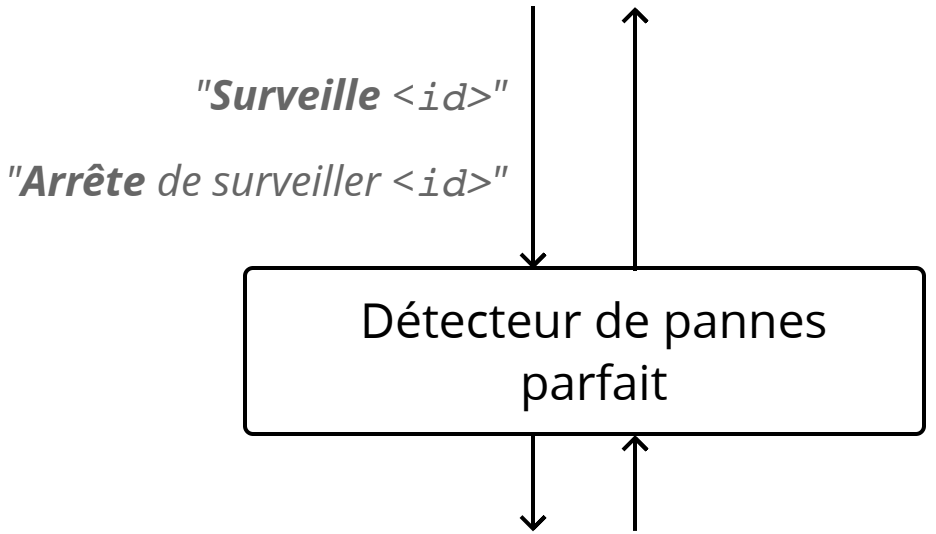


Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.

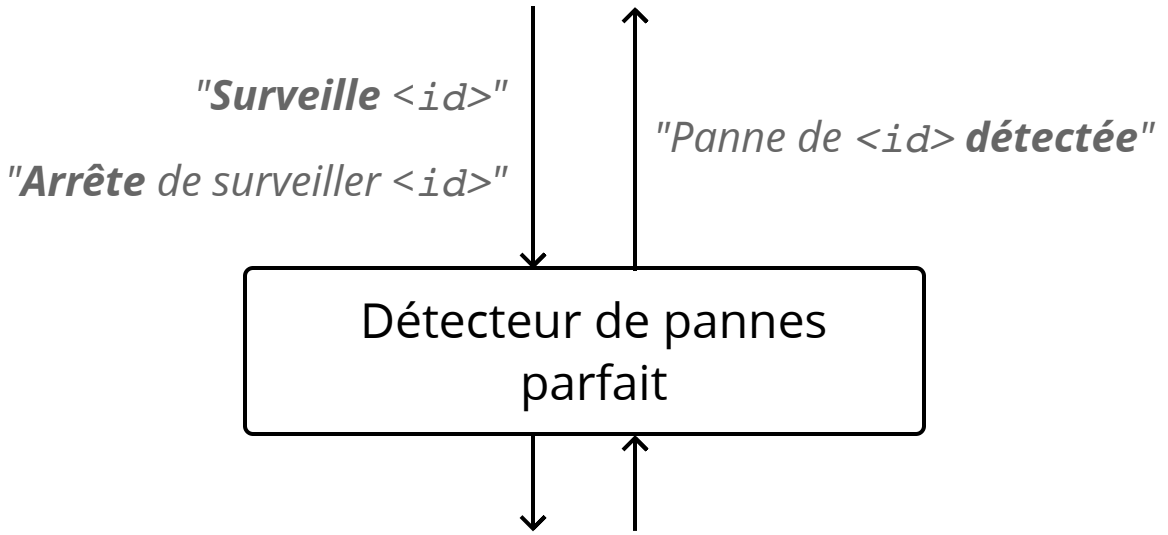


Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.

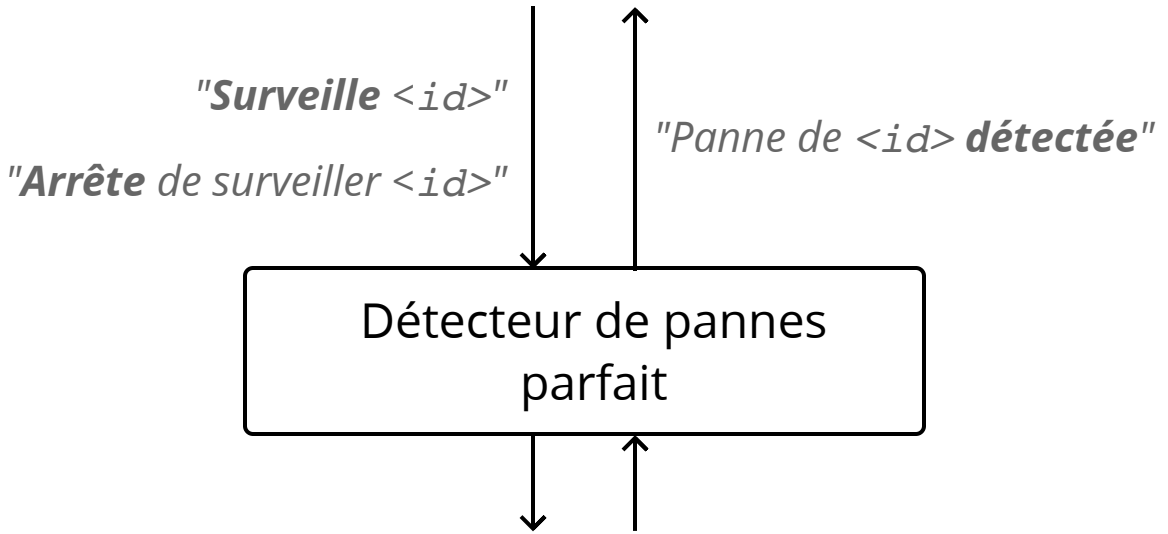


Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.



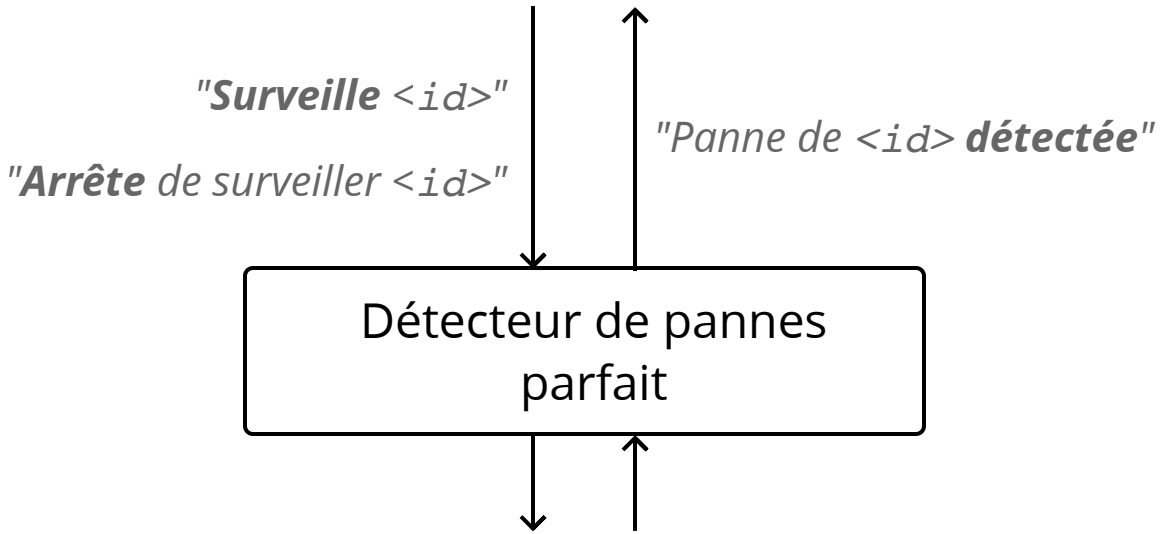
Propriétés

Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.



Propriétés

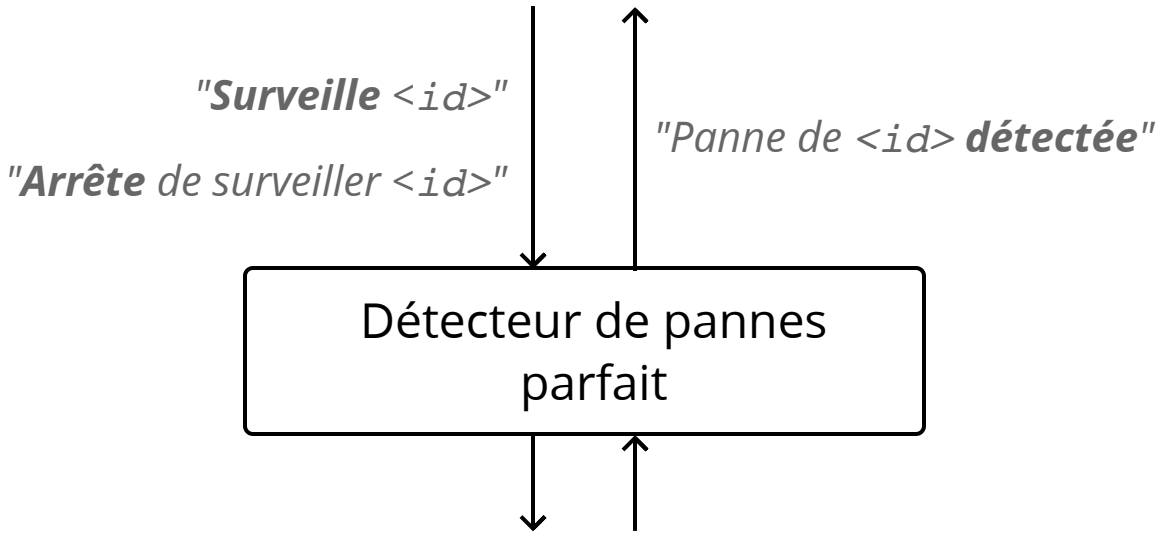
- *Complétude* : Un jour, tout processus en panne sera détecté par tout processus correct.

Speaker notes

Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Requirements

Supposons pour l'instant des pannes permanentes uniquement.



Propriétés

- *Complétude* : Un jour, tout processus en panne sera détecté par tout processus correct.

Speaker notes

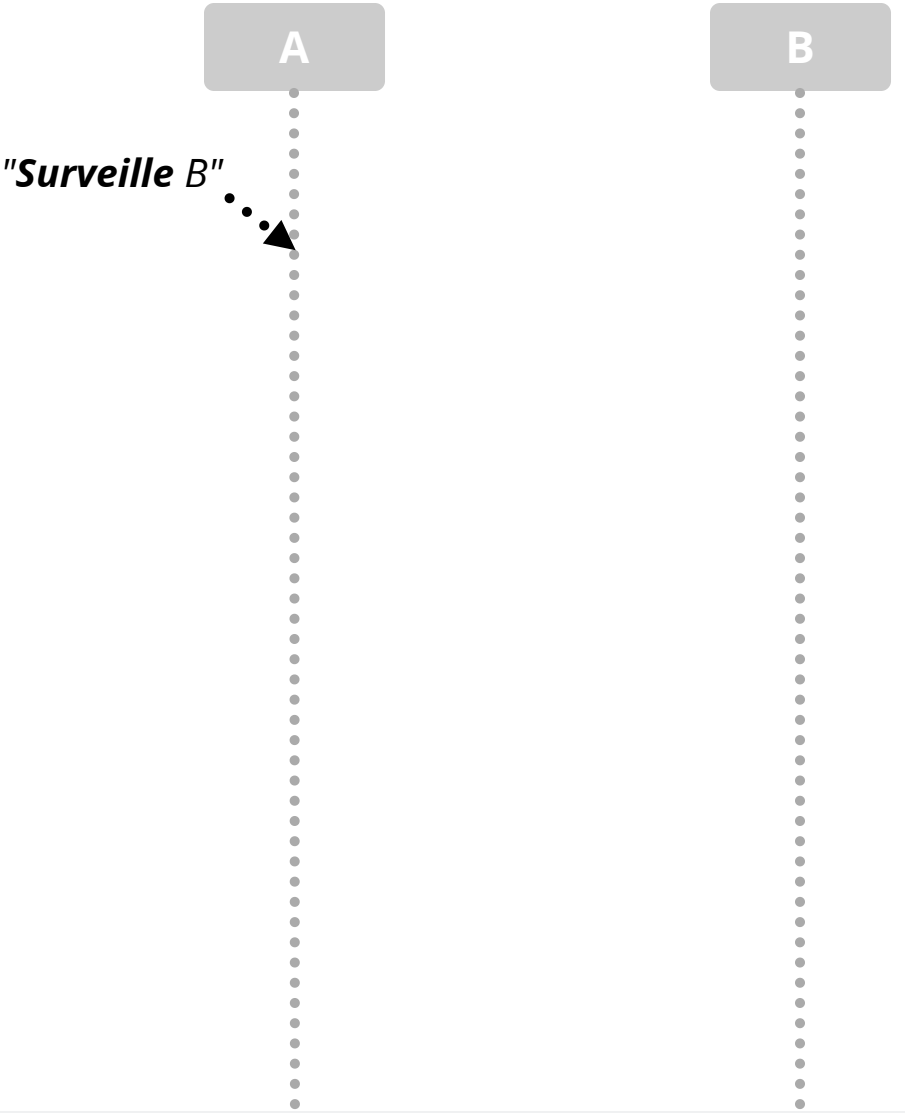
Quand nous parlons ici de propriétés, on entend les propriétés que l'on veut que le détecteur de pannes parfait possède. S'il ne vérifie pas ces propriétés, on ne l'appellera pas "détecteur de pannes parfait".

Suppositions

- Offert par le réseau et maintenu par les protocoles de fiabilité : pas de perte, de duplication, ni de réordonnancement.
- Toute panne est permanente.
- Il existe une borne supérieure constante **`T`** sur la durée de transit de tout message.

  On parle de système distribué "synchrone".
- Les durées de traitement sont négligeables.

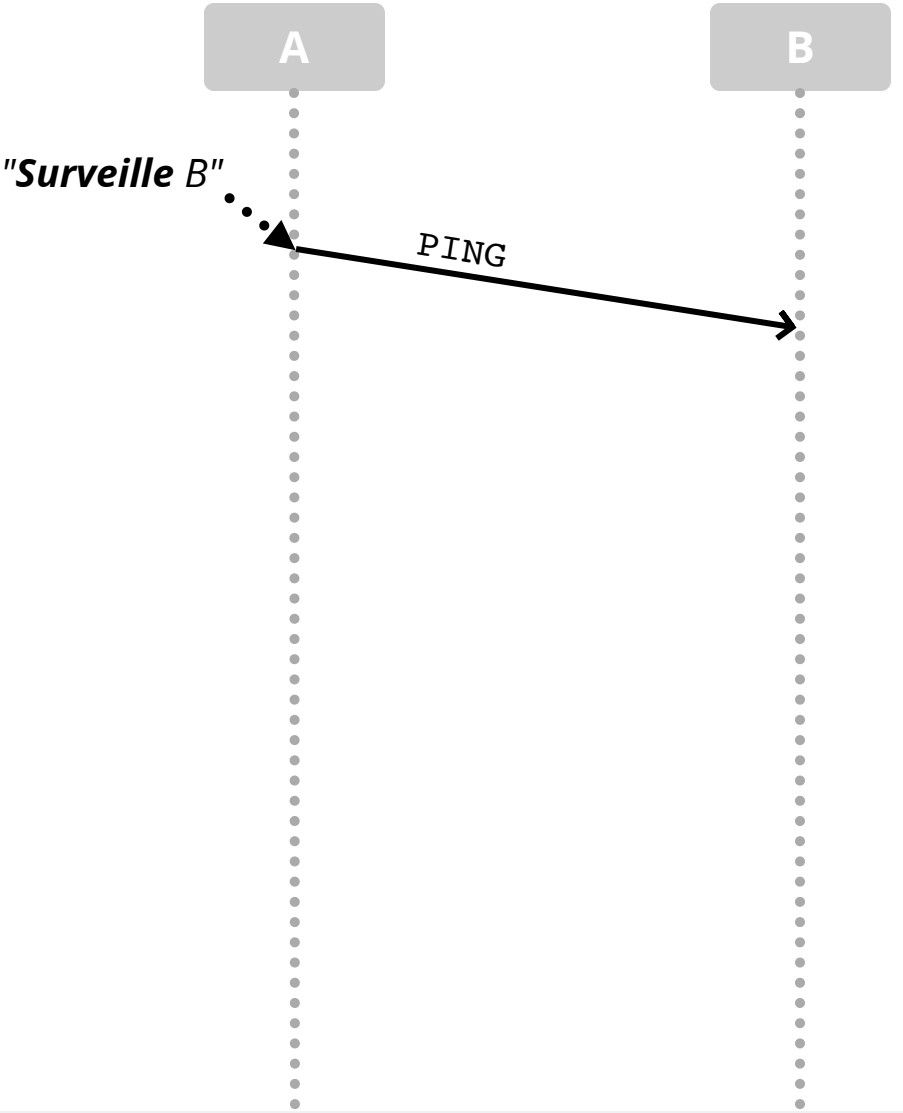
Heartbeat



Heartbeat

Périodiquement, le surveillant

envoie un `ping`, et

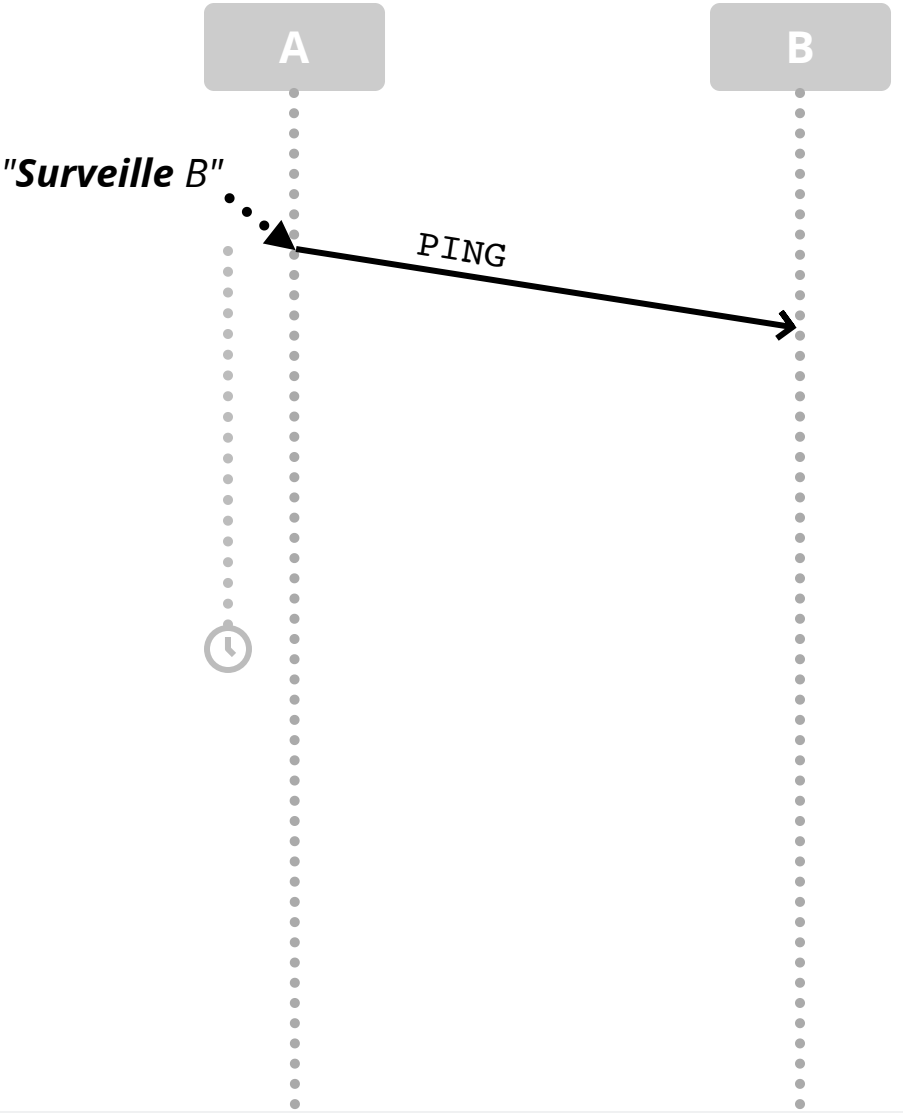


Heartbeat

Périodiquement, le surveillant

envoie un `ping`, et

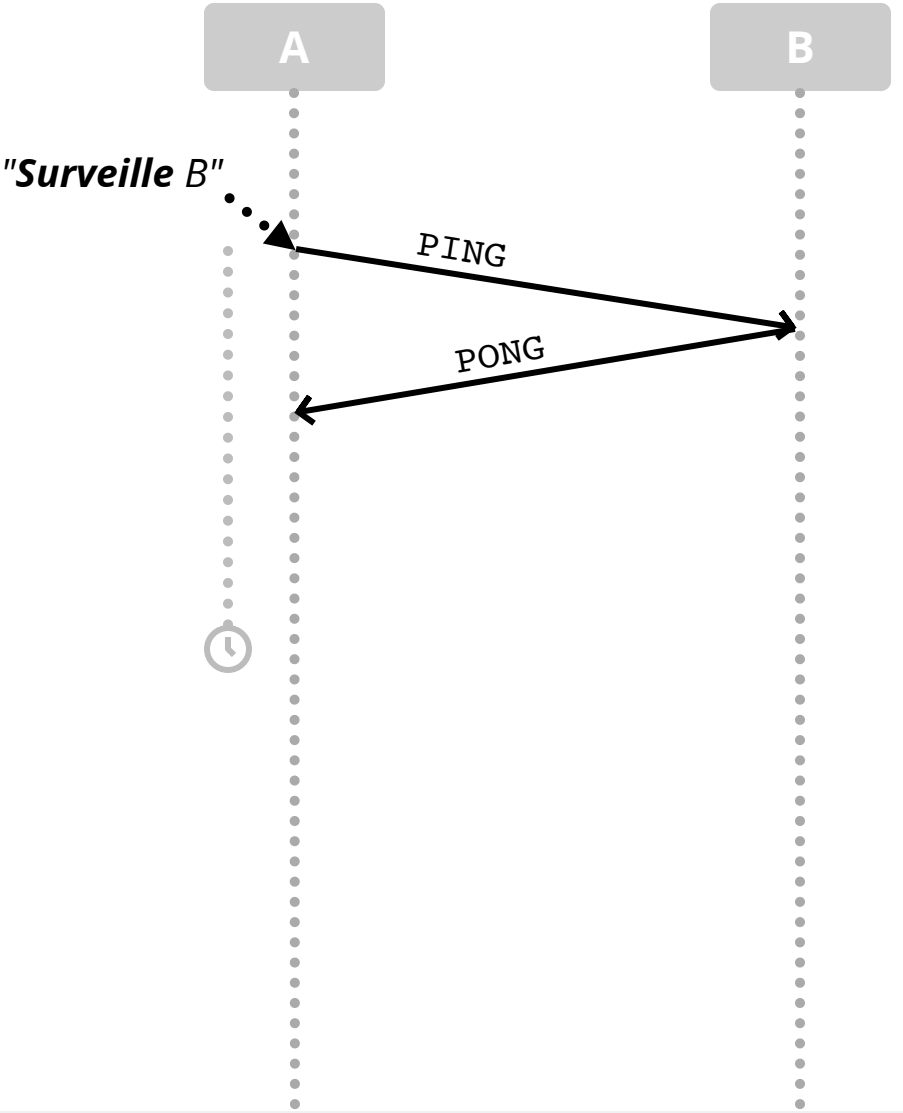
lance un timer de `2T`.



Heartbeat

Périodiquement, le surveillant
envoie un `ping`, et
lance un timer de `2T`.

Le surveillé
répond à `ping` par `pong`.

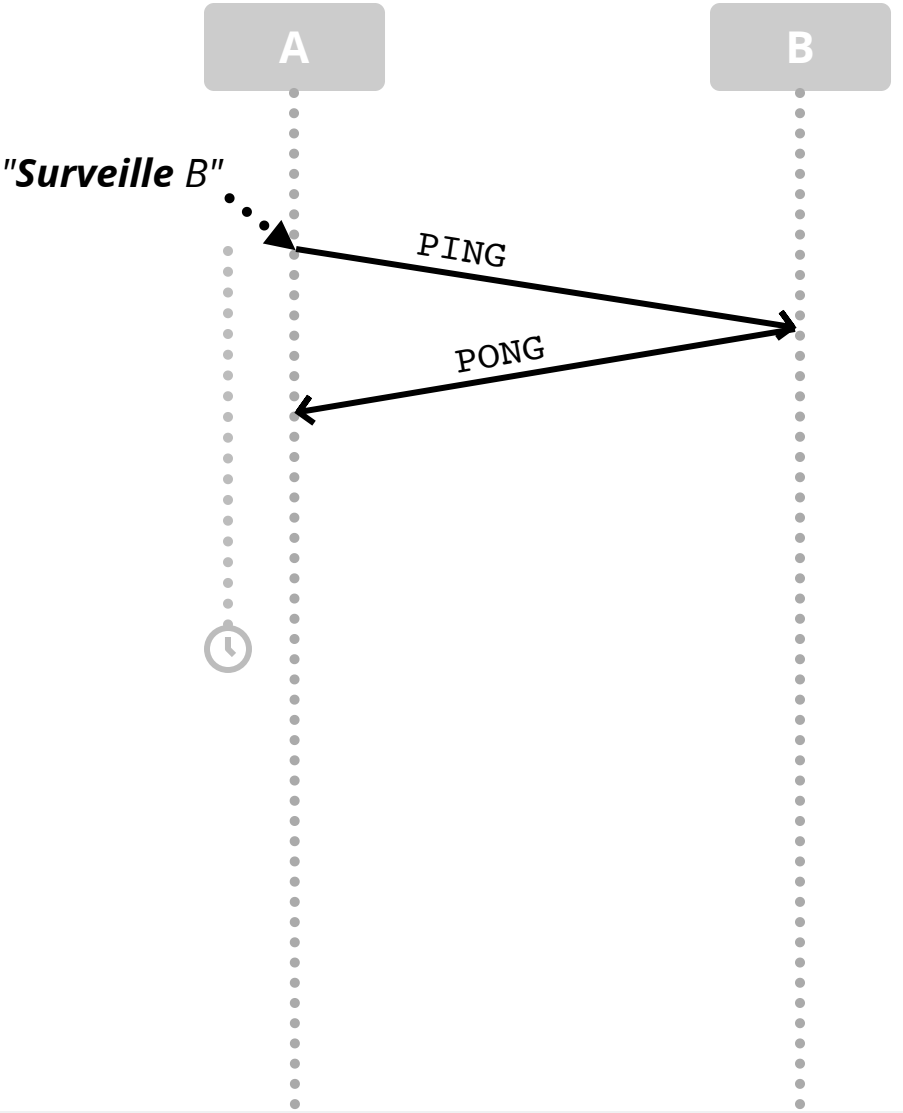


Heartbeat

Périodiquement, le surveillant
    envoie un `ping`, et
    lance un timer de `2T`.

À la fin du timer,
    Si le pong a été reçu,
        Un nouveau ping est envoyé
        Un nouveau timer est lancé

Le surveillé
    répond à `ping` par `pong`.

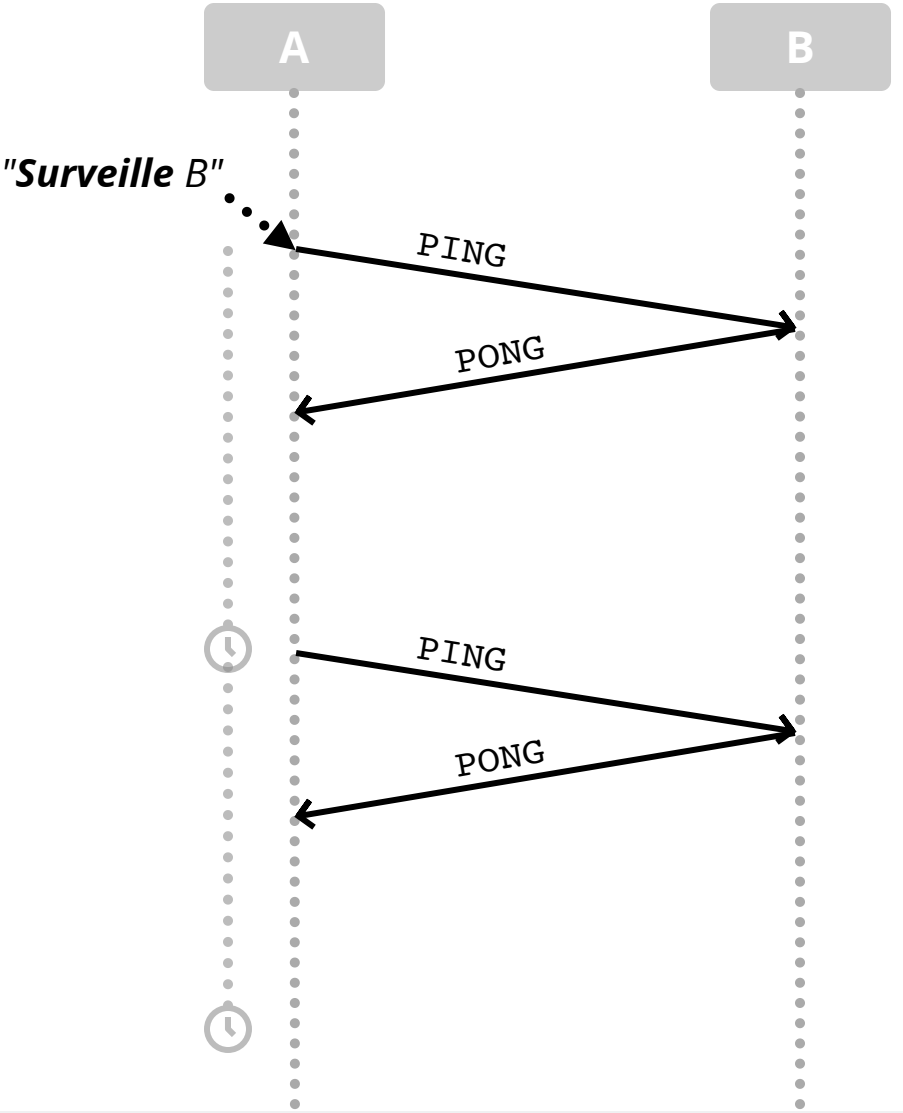


Heartbeat

Périodiquement, le surveillant
    envoie un `ping`, et
    lance un timer de `2T`.

À la fin du timer,
    Si le pong a été reçu,
        Un nouveau ping est envoyé
        Un nouveau timer est lancé

Le surveillé
    répond à `ping` par `pong`.

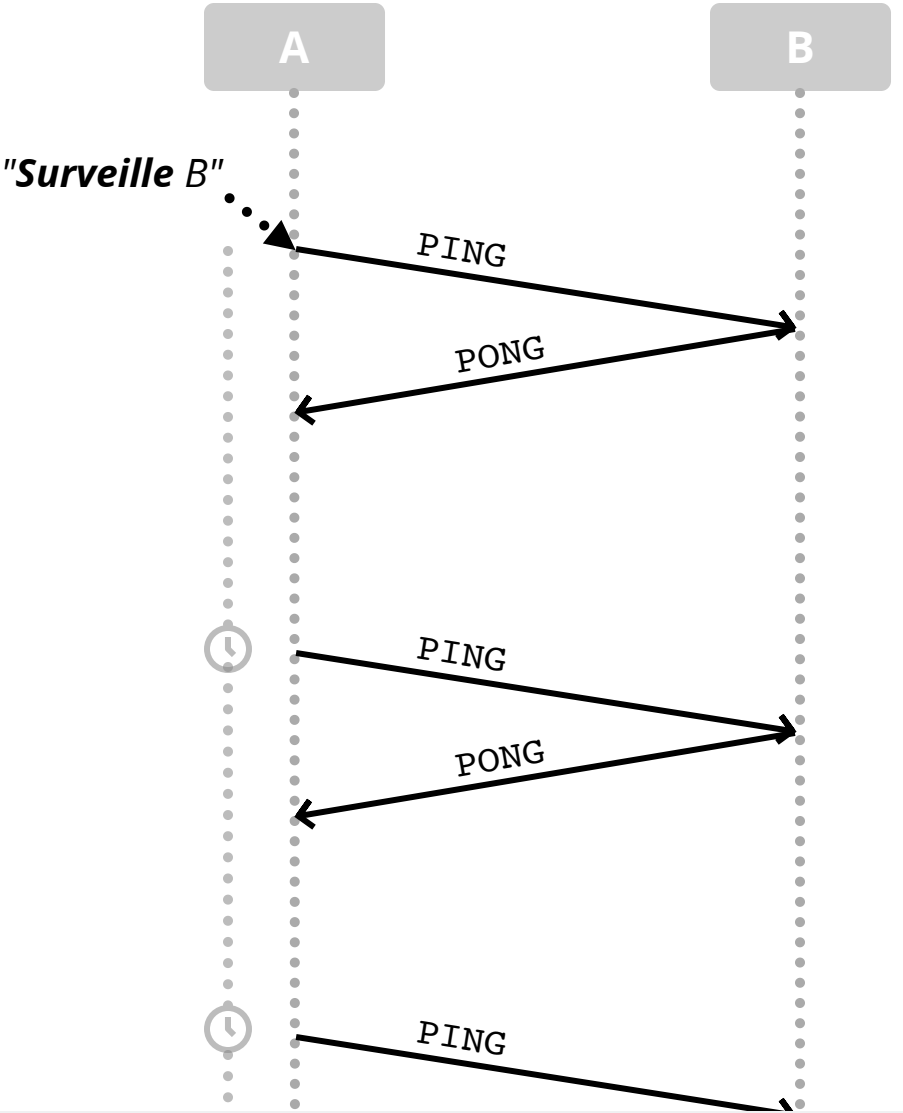


Heartbeat

Périodiquement, le surveillant
    envoie un `ping`, et
    lance un timer de `2T`.

À la fin du timer,
    Si le pong a été reçu,
        Un nouveau ping est envoyé
        Un nouveau timer est lancé

Le surveillé
    répond à `ping` par `pong`.

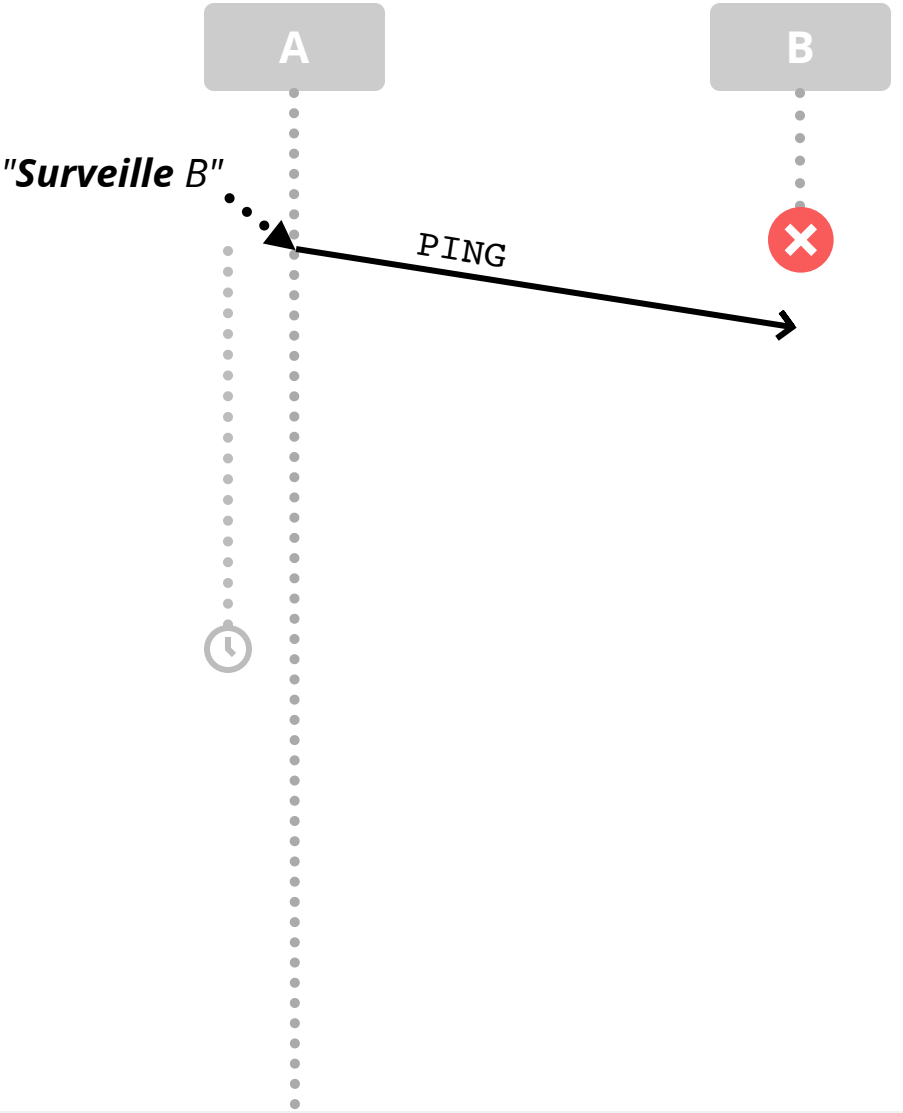


Heartbeat

Périodiquement, le surveillant
- envoie un `ping`, et
- lance un timer de `2T`.

À la fin du timer,
- Si le pong a été reçu,
  - Un nouveau ping est envoyé
  - Un nouveau timer est lancé

Le surveillé
- répond à `ping` par `pong`.

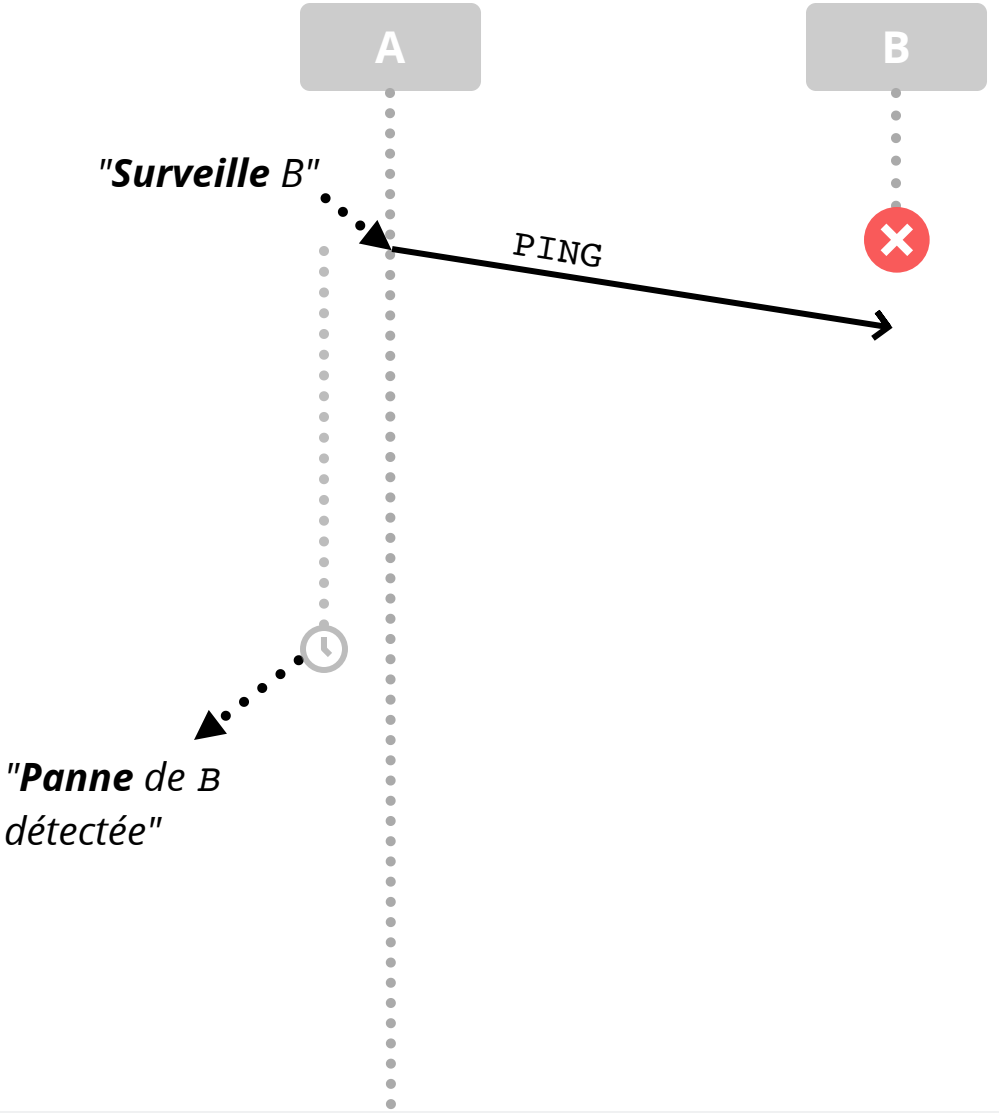


Heartbeat

Périodiquement, le surveillant
- envoie un `ping`, et
- lance un timer de `2T`.

À la fin du timer,
- Si le pong a été reçu,
  - Un nouveau ping est envoyé
  - Un nouveau timer est lancé
- Sinon,
  - Une panne est signalée

Le surveillé
- répond à `ping` par `pong`.



Heartbeat

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne sera détecté par tout processus correct."

Précision ?

"Si un processus p est détecté par un quelconque processus, alors p est en panne."

Heartbeat

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne sera détecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après $2T$, cette panne sera découverte.

Précision ?

"Si un processus p est détecté par un quelconque processus, alors p est en panne."

Heartbeat

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne sera détecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après $2T$, cette panne sera découverte.

Précision ?

"Si un processus p est détecté par un quelconque processus, alors p est en panne."

Oui, puisqu'un message ne prend jamais plus de T unités de temps pour transiter, je dois avoir reçu un pong après $2T$ si le processus est correct.

Heartbeat

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne sera détecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après $2T$, cette panne sera découverte.

Précision ?

"Si un processus p est détecté par un quelconque processus, alors p est en panne."

Oui, puisqu'un message ne prend jamais plus de T unités de temps pour transiter, je dois avoir reçu un pong après $2T$ si le processus est correct.

Problème ?

Heartbeat

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne sera détecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après $2T$, cette panne sera découverte.

Précision ?

"Si un processus p est détecté par un quelconque processus, alors p est en panne."

Oui, puisqu'un message ne prend jamais plus de T unités de temps pour transiter, je dois avoir reçu un pong après $2T$ si le processus est correct.

Problème ?

La supposition d'un système synchrone est irréaliste. Les vrais réseaux ne nous donnent pas de garantie sur la durée de transit d'un message.

Pseudocode

Variables

`surveillés`	*ensemble d'entiers*	processus que je surveille
`pongs_reçus`	*ensemble d'entiers*	ont répondu au dernier ping
`T`	*constante*	borne sur la durée de transit

Pseudocode

Initialisation

Écouter infiniment les événements suivants :

- Demande de surveillance de `i`
- Demande d'arrêt de surveillance de `i`
- Réception d'un ping de `i`
- Réception d'un pong de `i`
- Fin du timer de `i`

Pseudocode

Demande de surveillance de `i`

```
Ajouter i à surveillés
Envoyer un ping à i
Lancer un timer de 2T pour i
```

Demande d'arrêt de surveillance de `i`

```
Enlever i de surveillés
```

Réception d'un ping de `i`

```
Envoyer un pong à i
```

Pseudocode

Réception d'un pong de `i`

```
Ajouter i à pongs_reçus
```

Fin du timer de `i`

```
Si i est dans pongs_reçus
    Enlever i de pongs_reçus
    Envoyer un ping à i
    Lancer un timer de 2T pour i
Sinon
    Signaler « Panne de i détectée »
    Enlever i de surveillés
```

Détecteur de panne parfait un jour

Requirements

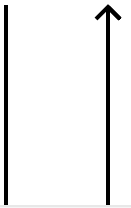
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



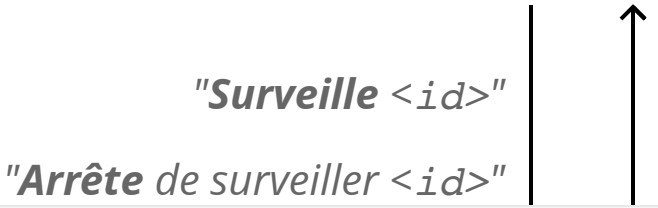
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



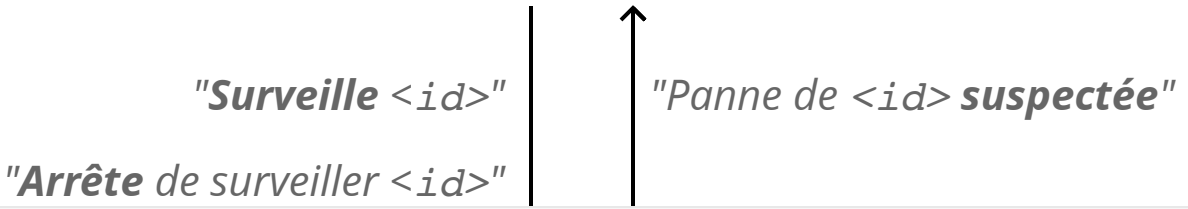
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



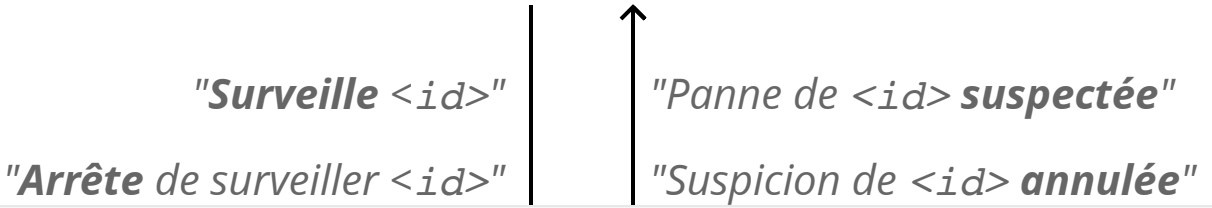
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



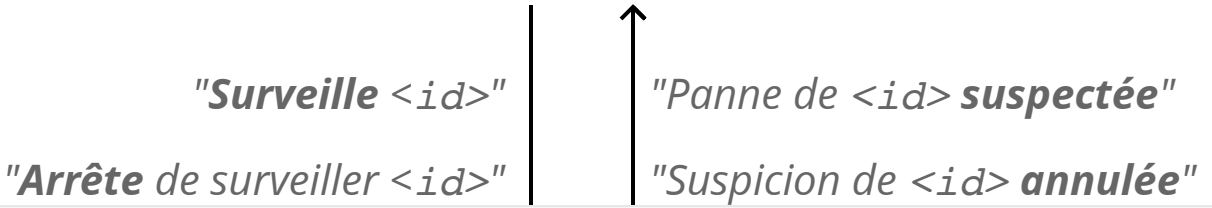
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



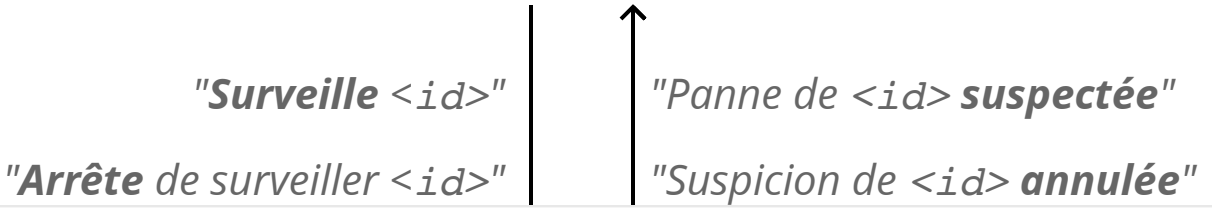
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



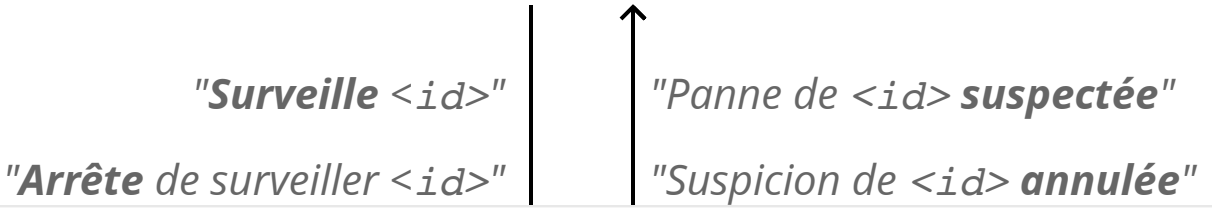
Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Requirements



Speaker notes

Notez : la propriété "précision un jour" implique qu'à partir d'un certain instant, toute suspicion faite par un processus correct sera d'un processus qui est soit en panne permanente, soit en panne récupérable sans arrêt (c'est à dire qu'il récupère de ses pannes, mais y tombe à nouveau plus tard).

Notez aussi : la propriété "complétude" ne parle que de panne permanente. En effet, ce détecteur n'est pas "complet" lorsqu'il s'agit de pannes récupérables : un processus en panne récupérable peut être incorrect mais répondre quand même à tous les pings (souvenez-vous qu'un processus est incorrect en terme de pannes récupérables s'il tombe sans arrêt en panne, même s'il s'en remet à chaque fois, il pourrait donc tomber en panne et s'en remettre suffisamment rapidement pour ne pas avoir laissé passer de pings, et donc sans que ce ne soit remarqué).

En général, une panne récupérable est le problème de la couche applicative, qui se rend compte de la panne, et peut la traiter correctement. Cette panne peut ainsi être traitée comme une panne permanente à laquelle s'est suivie l'introduction d'un nouveau processus au système. Un détecteur de panne comme celui décrit ici fonctionnerait alors, puisqu'il détecterait bien qu'une panne a eu lieu, puis la couche applicative gérerait l'introduction du nouveau processus dans le système (en réalité, sa réintroduction).

Suppositions

- **Construit** par les protocoles de fiabilité : pas de perte, de duplication, ni de réordonnancement.
- Toute panne est permanente ***ou récupérable.***
- Il existe une borne supérieure constante **`T` inconnue** sur la durée de transit de tout message.

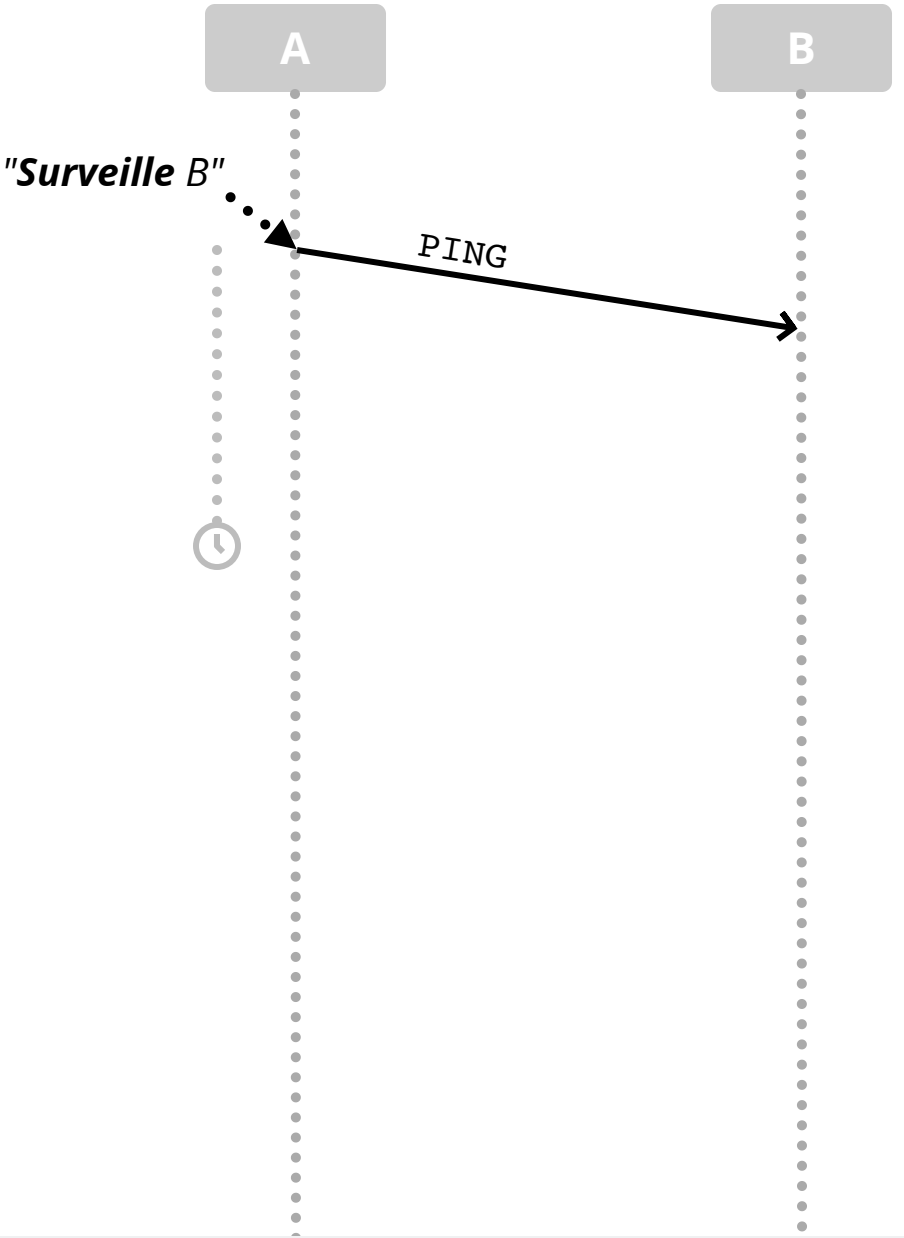
  *On parlera ici de système distribué "**partiellement** synchrone".*
- Les durées de traitement sont négligeables.

Timeout dynamique

Initialement,

envoie un `ping`, et

lance un timer de durée **Δ**



Timeout dynamique

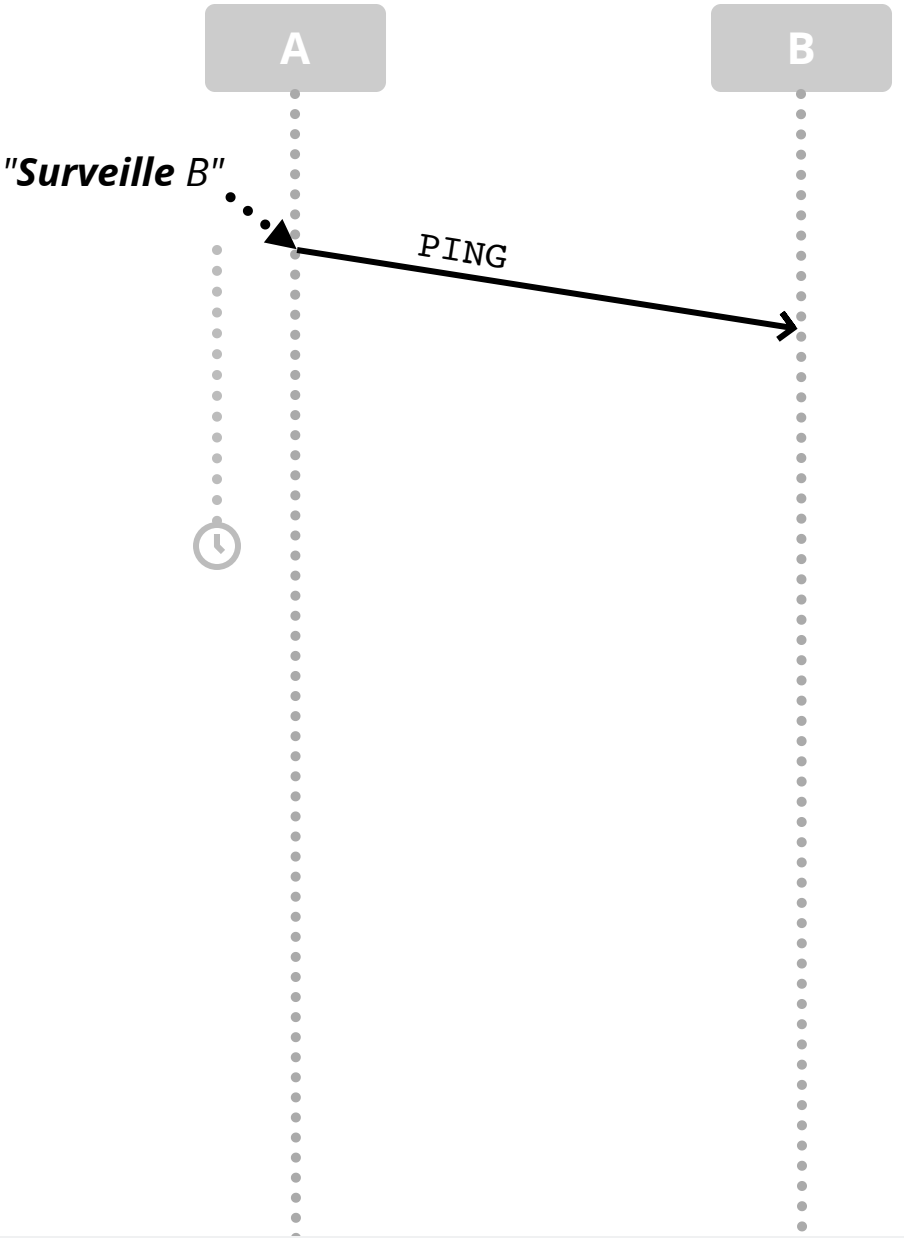
Initialement,

envoie un `ping`, et

lance un timer de durée **Δ**

À la fin du timer,

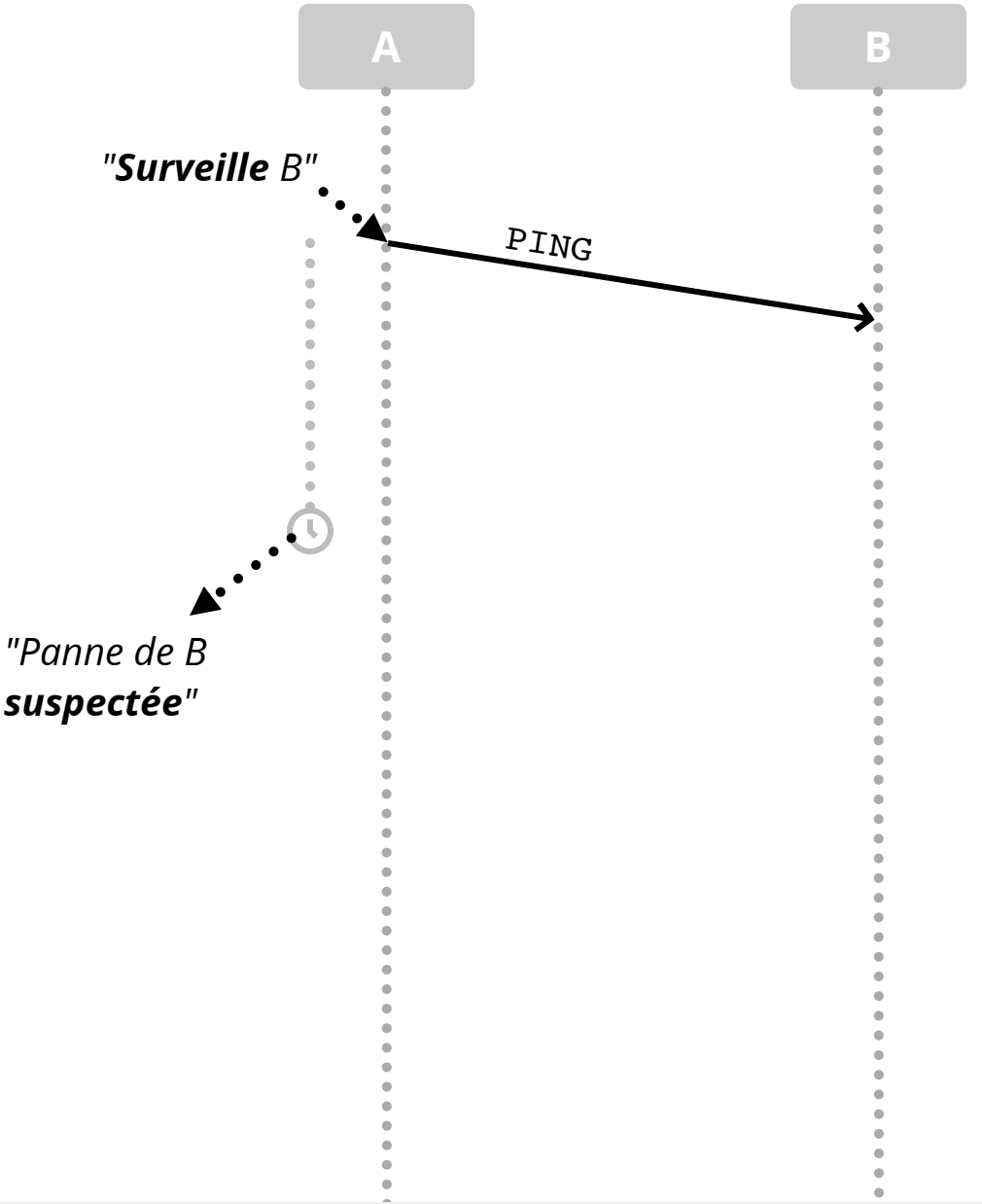
Si le pong n'a pas été reçu,



Timeout dynamique

Initialement,
  envoie un `ping`, et
  lance un timer de durée **Δ**

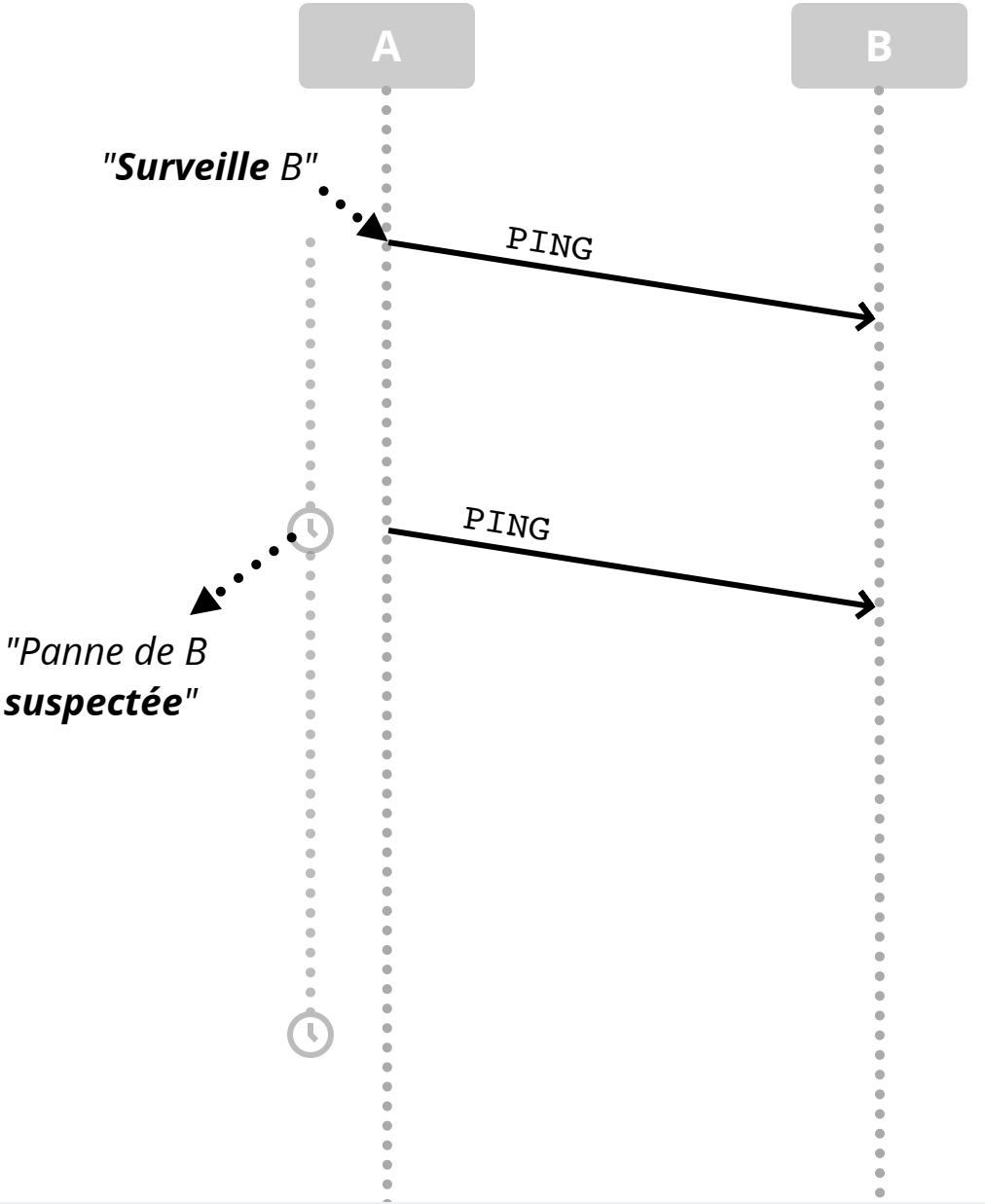
À la fin du timer,
  Si le pong n'a pas été reçu,
    Δ est incrémenté d'une constante
    Une panne est suspectée



Timeout dynamique

Initialement,
  envoie un `ping`, et
  lance un timer de durée **Δ**

À la fin du timer,
  Si le pong n'a pas été reçu,
    Δ est incrémenté d'une constante
    Une panne est suspectée
  Envoie un `ping`, et
  lance un timer de durée **Δ**

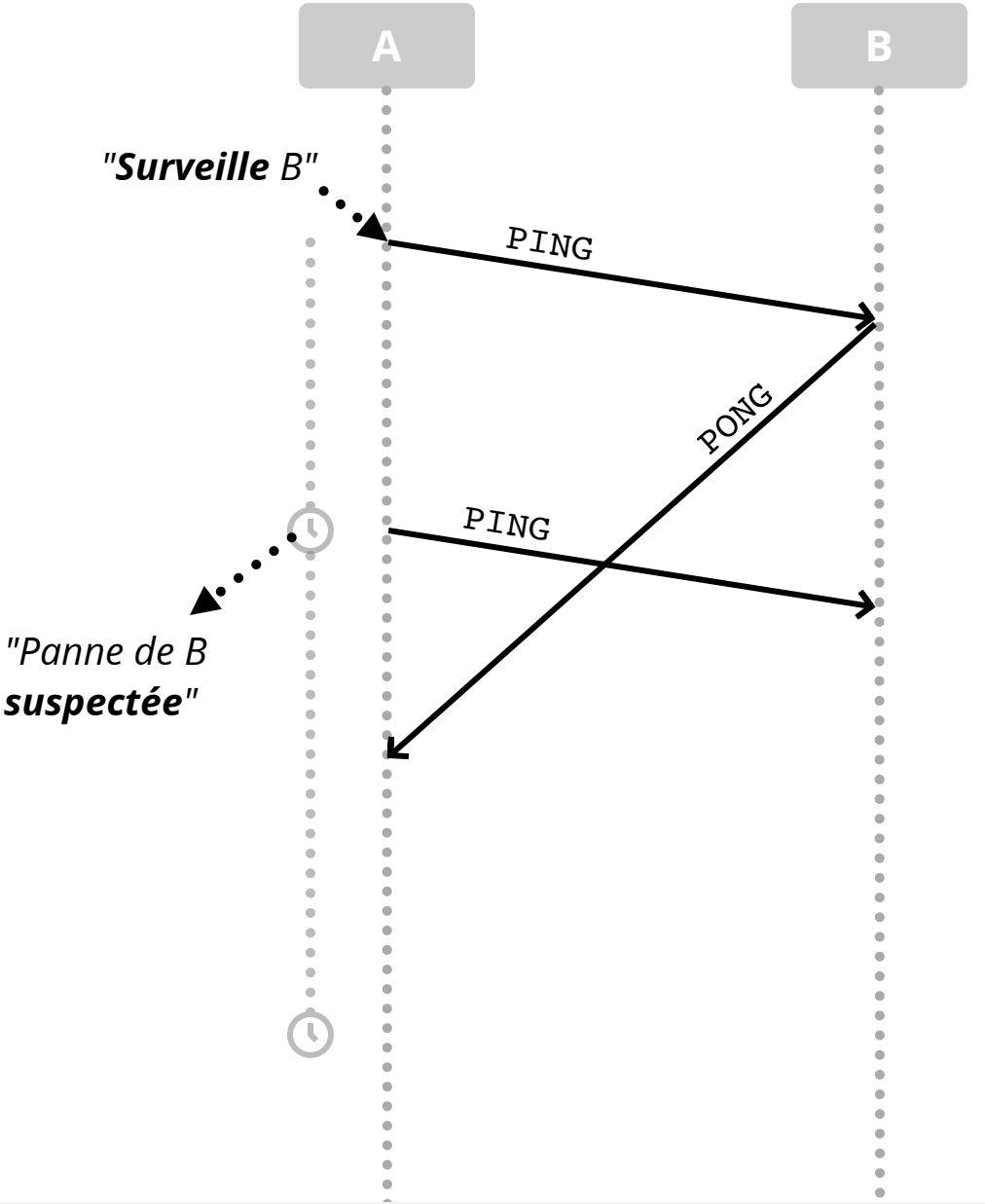


Timeout dynamique

Initialement,
  envoie un `ping`, et
  lance un timer de durée **Δ**

À la fin du timer,
  Si le pong n'a pas été reçu,
    Δ est incrémenté d'une constante
    Une panne est suspectée
  Envoie un `ping`, et
  lance un timer de durée **Δ**

Le surveillé
  répond à `ping` par `pong`.



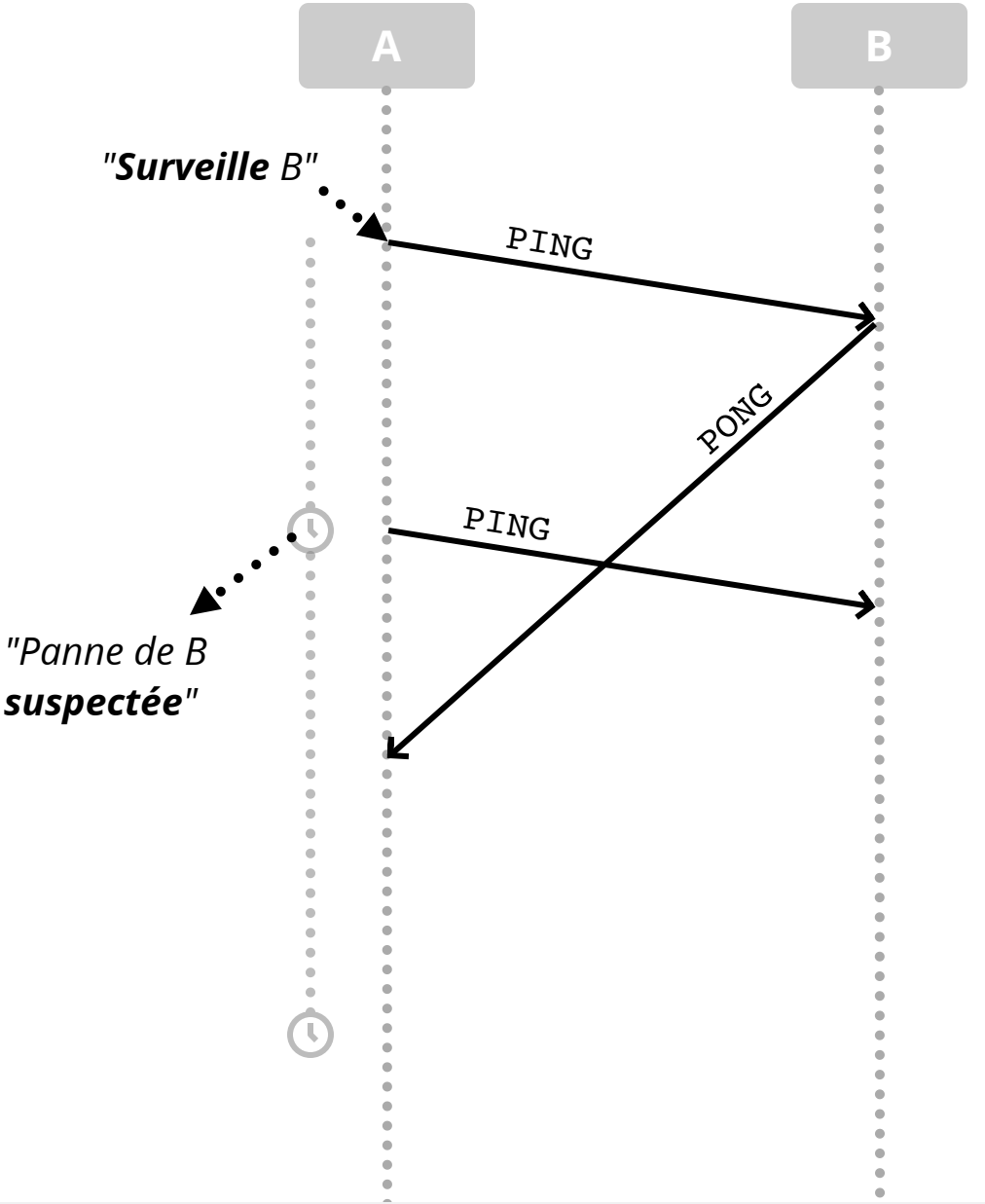
Timeout dynamique

Initialement,
  envoie un `ping`, et
  lance un timer de durée **Δ**

À la fin du timer,
  Si le pong n'a pas été reçu,
    Δ est incrémenté d'une constante
    Une panne est suspectée
  Envoie un `ping`, et
  lance un timer de durée **Δ**

Si un pong est reçu d'un suspecté

Le surveillé
  répond à `ping` par `pong`.



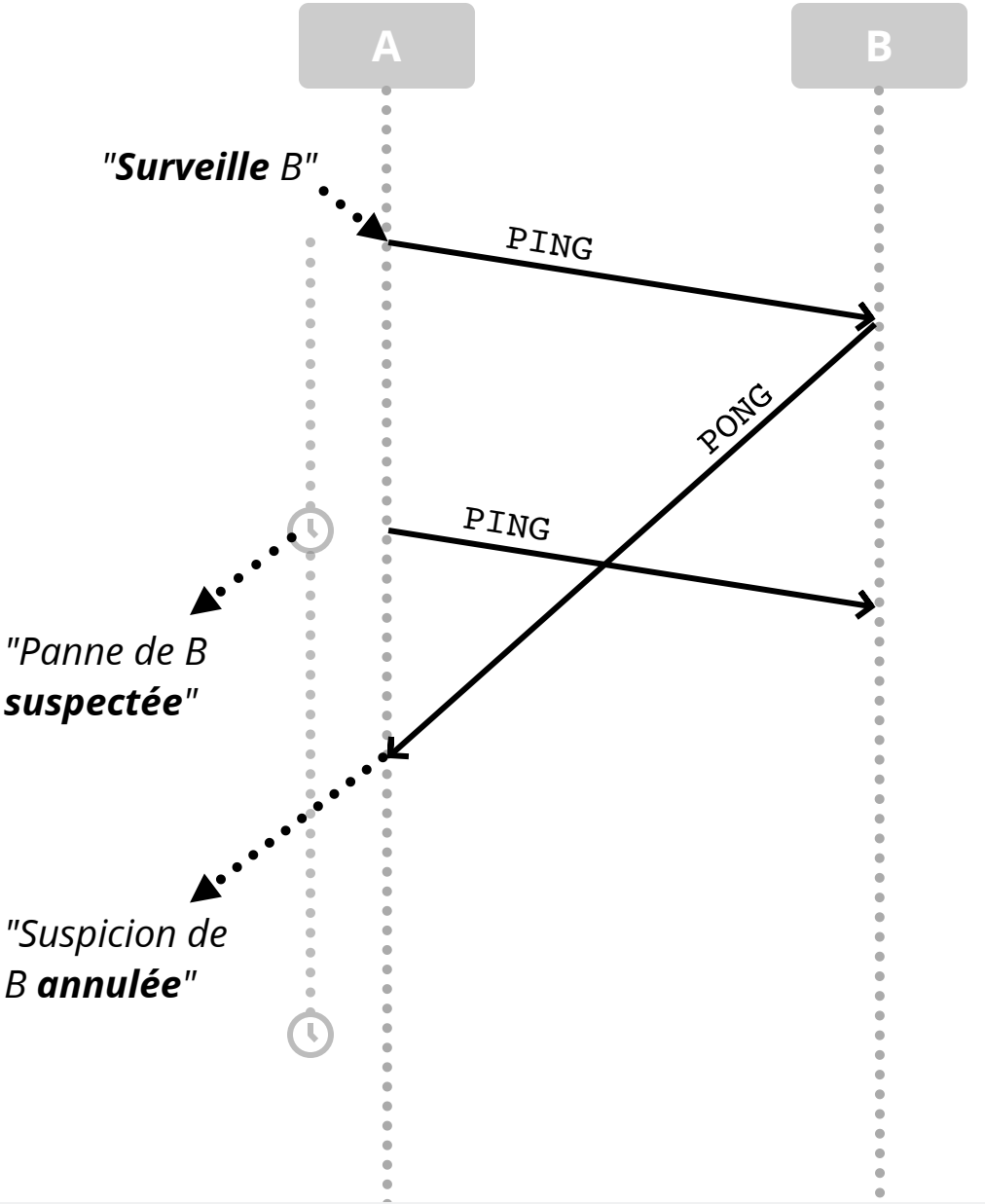
Timeout dynamique

Initialement,
  envoie un `ping`, et
  lance un timer de durée **Δ**

À la fin du timer,
  Si le pong n'a pas été reçu,
    Δ est incrémenté d'une constante
    Une panne est suspectée
  Envoie un `ping`, et
  lance un timer de durée **Δ**

Si un pong est reçu d'un suspecté
  La suspicion est annulée.

Le surveillé
  répond à `ping` par `pong`.



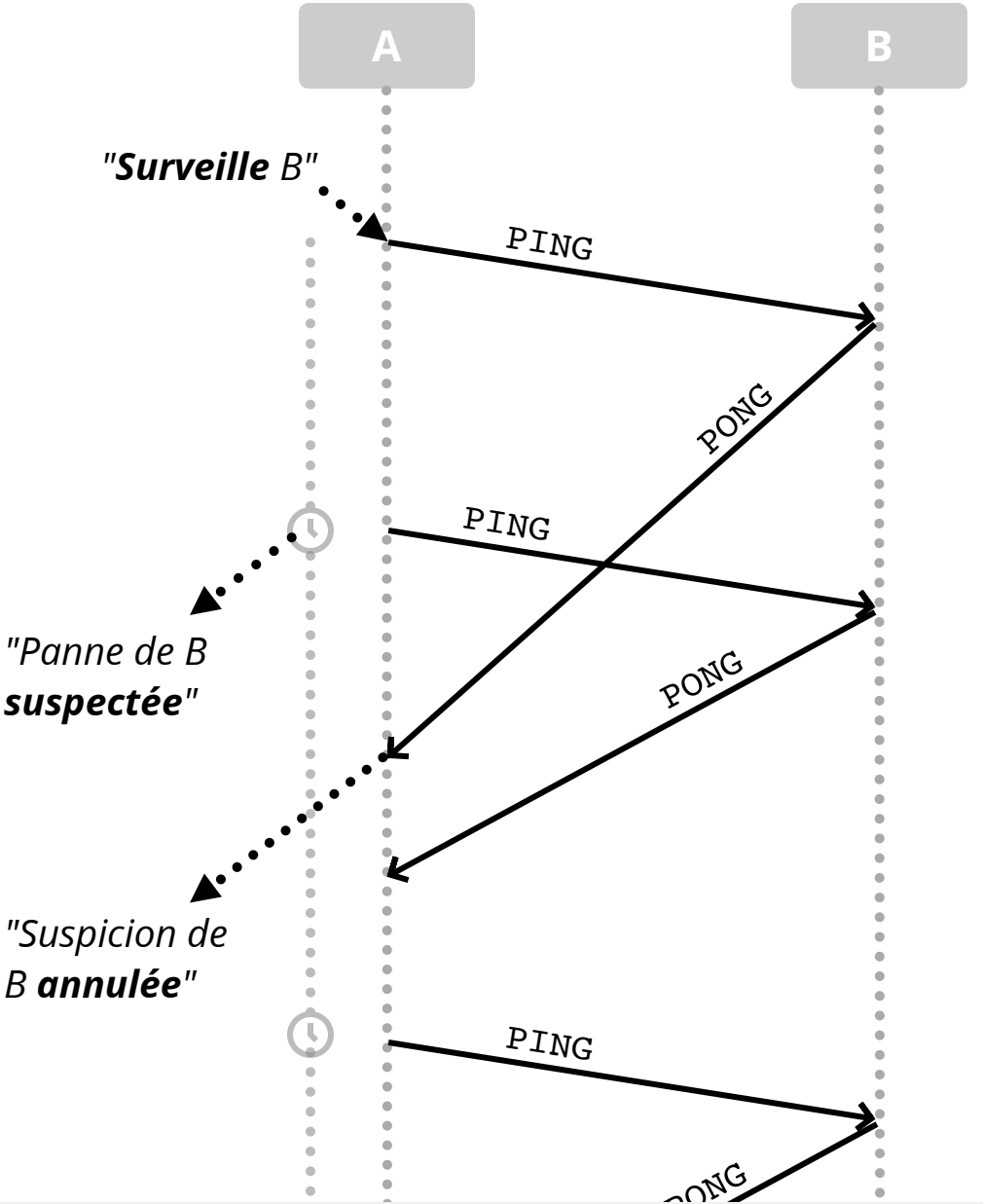
Timeout dynamique

Initialement,
  envoie un `ping`, et
  lance un timer de durée **Δ**

À la fin du timer,
  Si le pong n'a pas été reçu,
    Δ est incrémenté d'une constante
    Une panne est suspectée
  Envoie un `ping`, et
  lance un timer de durée **Δ**

Si un pong est reçu d'un suspecté
  La suspicion est annulée.

Le surveillé
  répond à `ping` par `pong`.



Timeout dynamique

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne permanente sera suspecté par tout processus correct."

Précision un jour ?

"Un jour, aucun processus correct ne sera suspecté par un processus correct."

Timeout dynamique

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne permanente sera suspecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après Δ, cette panne sera découverte.

Précision un jour ?

"Un jour, aucun processus correct ne sera suspecté par un processus correct."

Timeout dynamique

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne permanente sera suspecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après Δ, cette panne sera découverte.

Précision un jour ?

"Un jour, aucun processus correct ne sera suspecté par un processus correct."

Oui, un processus correct (en terme de panne permanente comme récupérable) répondra aux pings en un temps fini ; quand Δ sera assez grand, plus de panne ne sera suspectée : tout processus sera soit "en panne" et suspecté pour toujours, ou "correct" et plus jamais suspecté.

Timeout dynamique

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne permanente sera suspecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après Δ, cette panne sera découverte.

Précision un jour ?

"Un jour, aucun processus correct ne sera suspecté par un processus correct."

Oui, un processus correct (en terme de panne permanente comme récupérable) répondra aux pings en un temps fini ; quand Δ sera assez grand, plus de panne ne sera suspectée : tout processus sera soit "en panne" et suspecté pour toujours, ou "correct" et plus jamais suspecté.

Était-il nécessaire d'avoir un timeout dynamique ?

Timeout dynamique

Est-ce que ça vérifie

Complétude ?

"Un jour, tout processus en panne permanente sera suspecté par tout processus correct."

Oui, un processus en panne permanente ne répondra plus aux pings, et après Δ, cette panne sera découverte.

Précision un jour ?

"Un jour, aucun processus correct ne sera suspecté par un processus correct."

Oui, un processus correct (en terme de panne permanente comme récupérable) répondra aux pings en un temps fini ; quand Δ sera assez grand, plus de panne ne sera suspectée : tout processus sera soit "en panne" et suspecté pour toujours, ou "correct" et plus jamais suspecté.

Était-il nécessaire d'avoir un timeout dynamique ?

Oui, sinon on risquerait de constamment suspecter un processus qui est simplement lent mais bien correct.

Timeout dynamique

Δ ne décroît jamais.

Un réseau redevenu rapide garde un détecteur arbitrairement lent.

Speaker notes

Le timeout ne redescend jamais : si le réseau redevient rapide, le détecteur reste lent. Le faire décroître échangerait de la réactivité contre de la précision — on risquerait de resuspecter un processus simplement lent. Ce n'est pas le compromis retenu ici.

Pseudocode

Variables

`surveillés`	*ensemble d'entiers*	processus que je surveille
`suspectés`	*ensemble d'entiers*	ceux que je suspecte
`pongs_reçus`	*ensemble d'entiers*	ont répondu au dernier ping
Δ	*entier, init* Δ_0	timeout courant, commun à tous

Pseudocode

Demande de surveillance de `i`

```
Ajouter i à surveillés
Envoyer un ping à i
Lancer un timer de Δ pour i
```

Réception d'un pong de `i`

```
Ajouter i à pongs_reçus
Si i est dans suspectés
        Enlever i de suspectés
        Signaler « Suspicion de i annulée »
```

Pseudocode

Fin du timer de `i`

Si `i` n'est pas dans `pongs_reçus`
    Δ est incrémenté d'une constante
    Ajouter `i` à `suspectés`
    Signaler « Panne de `i` suspectée »
Enlever `i` de `pongs_reçus`
Envoyer un ping à `i`
Lancer un timer de Δ pour `i`

Les deux autres événements sont ceux du détecteur parfait.

Exercice

Le détecteur parfait suppose une borne T connue et des pannes permanentes.

Que se passe-t-il si un processus surveillé tombe en panne **puis récupère** ?

Reprenez le pseudocode du détecteur parfait et dites, pour chacune des deux propriétés, si elle tient encore, et pourquoi.