SDR

Communiquer avec des pannes

Olivier Lemer

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



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



L'aspect "correct" est une caractéristique *théorique*.

- Elle servira à analyser les algorithmes du cours.
  (e.g. "cet algorithme fonctionne ssi tous les processus sont corrects en terme de panne permanente", équivalent à dire "cet algorithme fonctionne ssi aucun processus ne tombe en panne.")
- Elle n'a aucun sens dans la vraie vie.
  (Impossible de savoir si un processus tombera un jour en panne ou non)

Panne récupérable

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



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

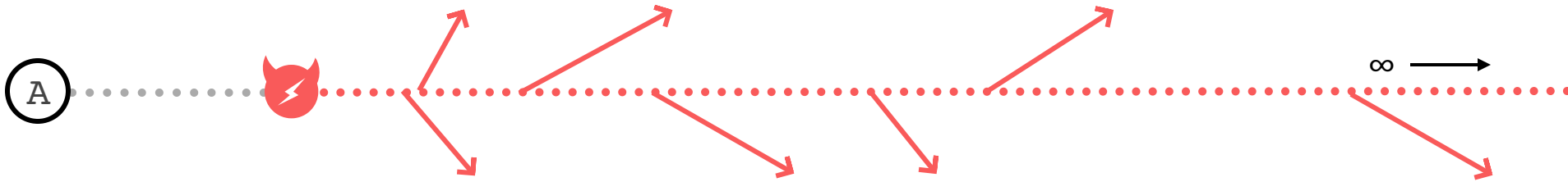
On dit qu'un processus est **correct en terme de panne récupérable** quand **il existe un instant T après lequel il ne tombera plus en panne.**



Donc, un processus qui tombe constamment en panne, ou qui ne revient jamais d'une panne n'est *pas correct*.

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.**



Évidemment, le type de panne le plus couteux à supporter.

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.*

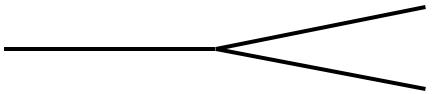
fair-lossy pertes de message autorisées,
mais après assez de renvois, le message finit par arriver.

Garanties du transport layer



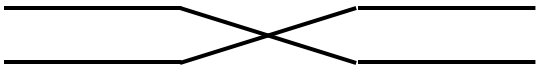
Pas de perte

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

Tolérer les pannes

En cas de panne, maintenir le **progrès** (l'algorithme avance)
maintenir la **validité** (l'algorithme reste correcte)

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.

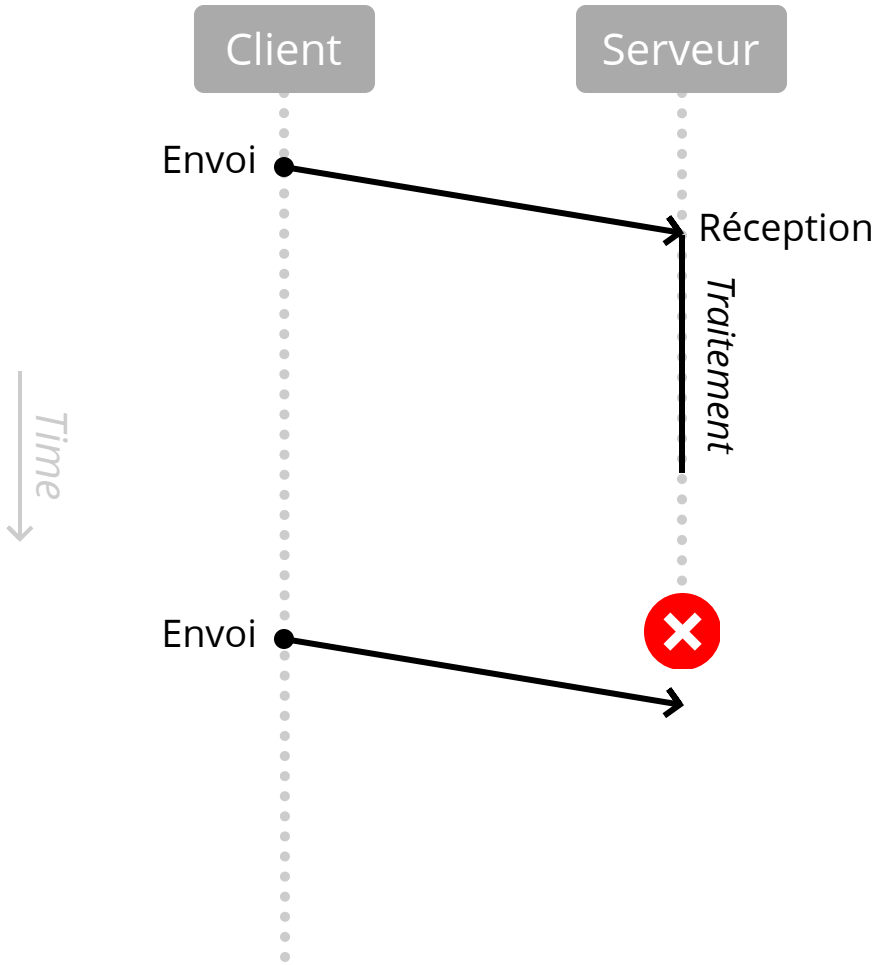


Protocole Request

"peut-être" (a.k.a. *maybe*)

Protocoles de fiabilité

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



Uniquement une requête,
pas de réponse du serveur.

Panne Serveur ?
Requête perdue, client l'ignore.

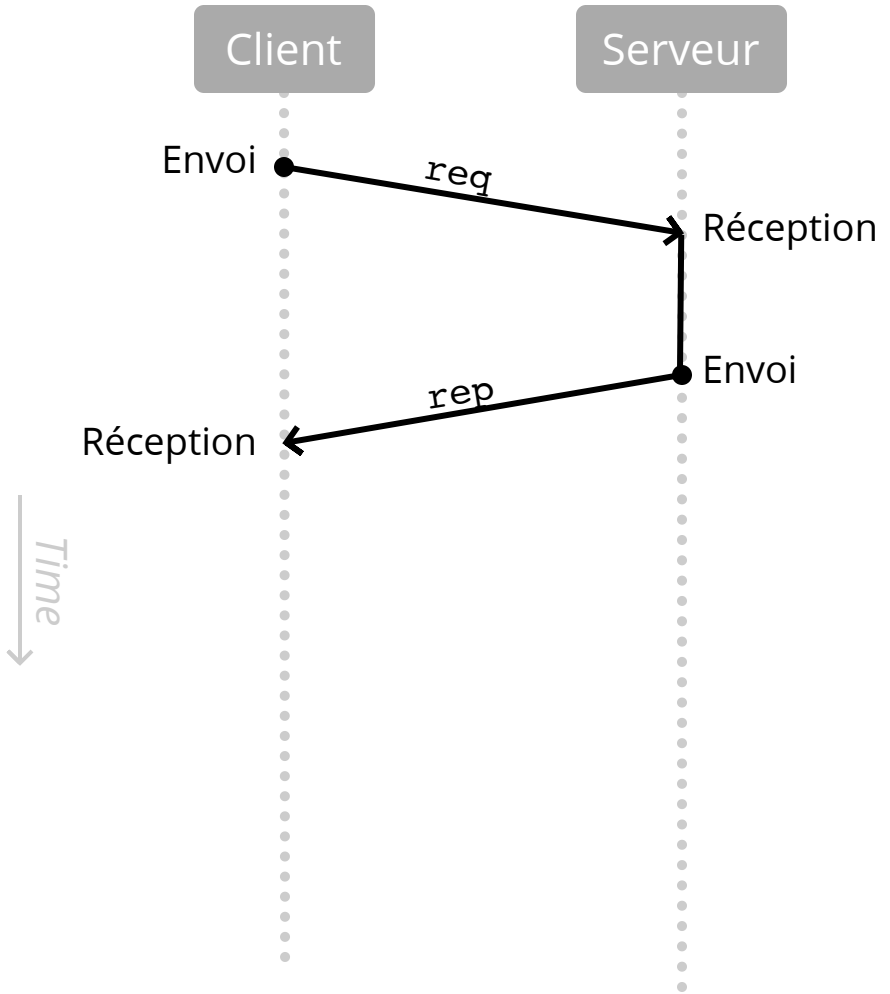
Le message sera **peut-être** traité.

Protocole Request-Reply

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

Protocoles de fiabilité

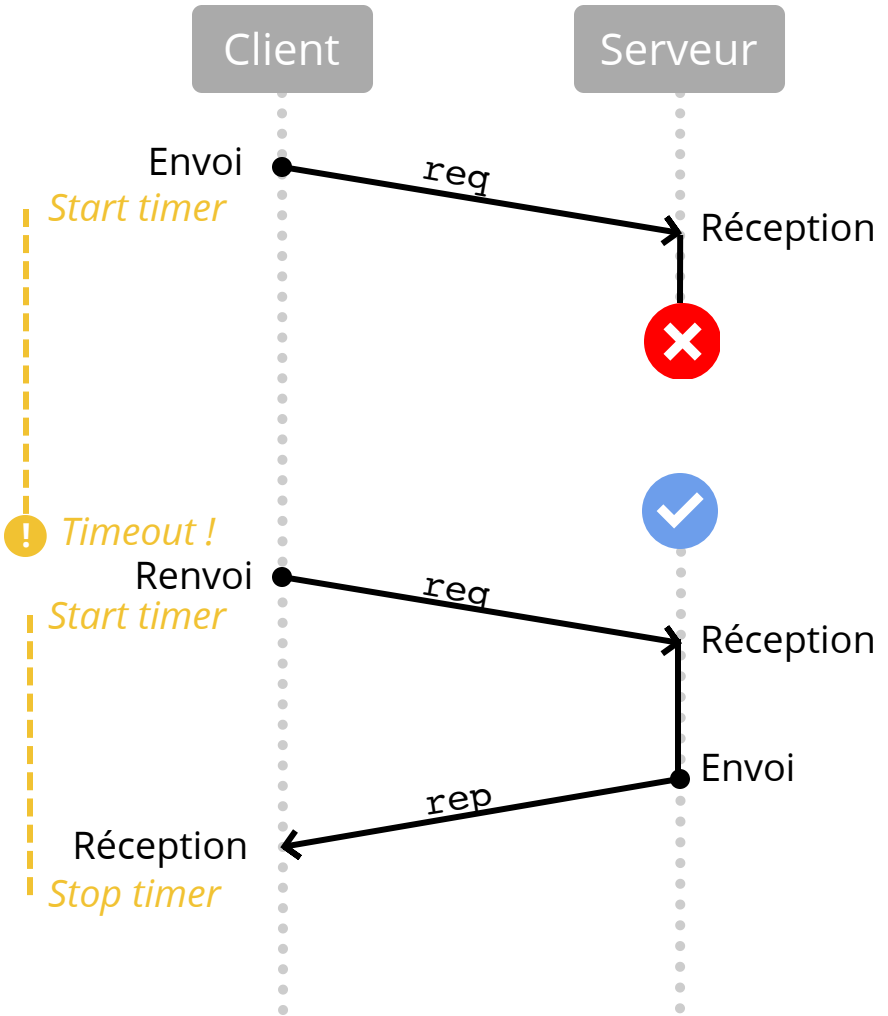
Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

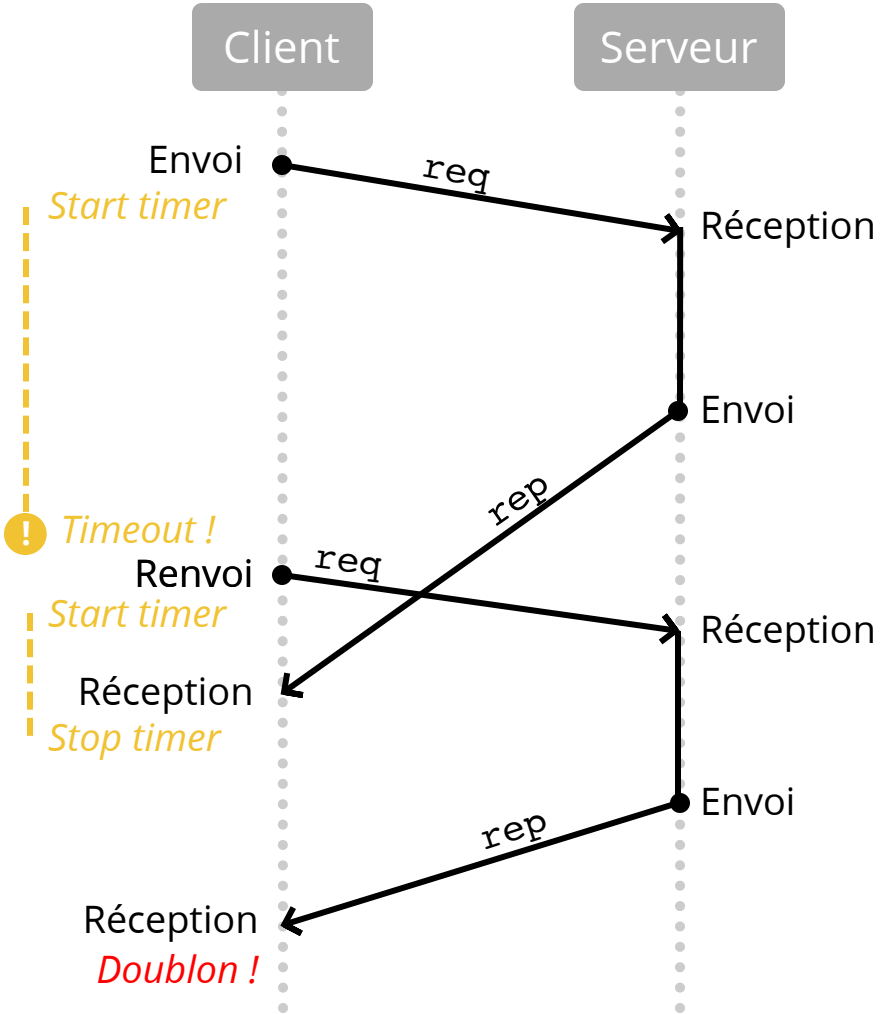
Timeout

déclenche un renvoi.
est annulé à la réception.

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply) *(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.

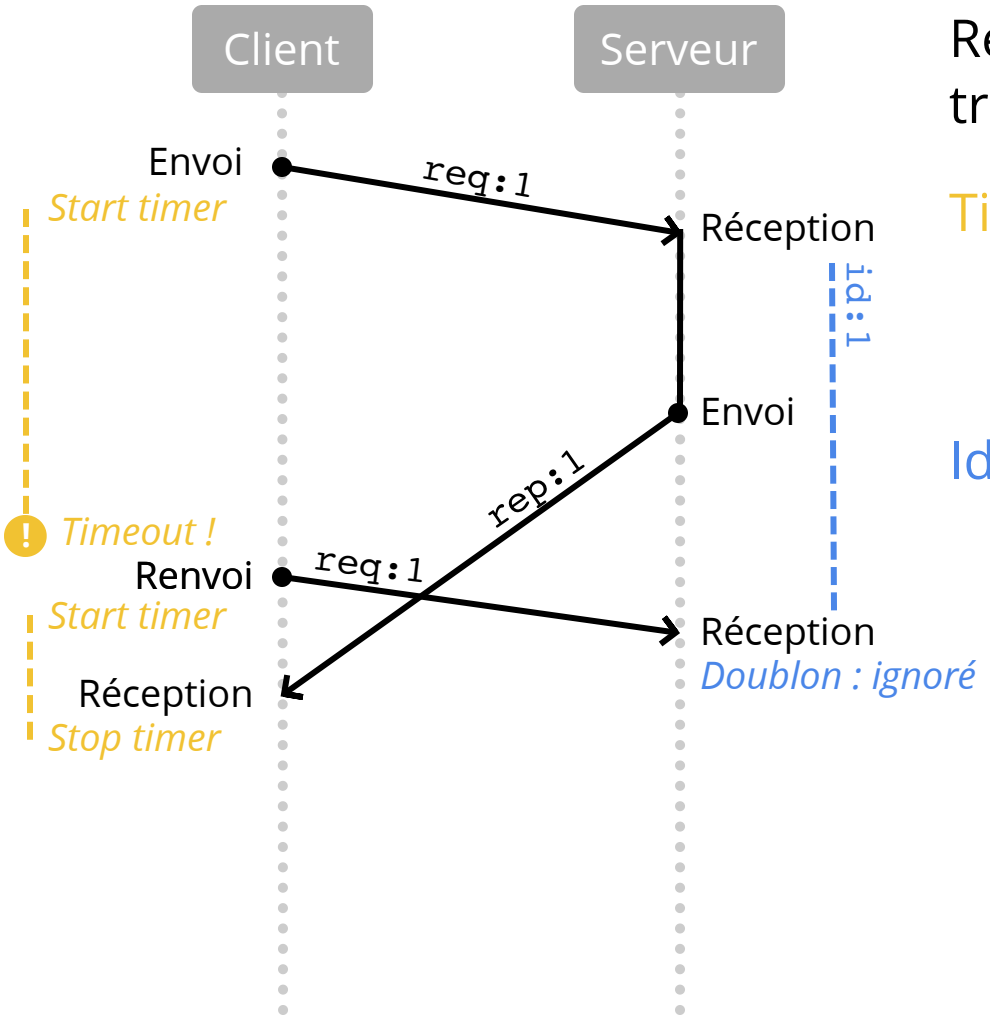
Le message sera traité
au moins une fois

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

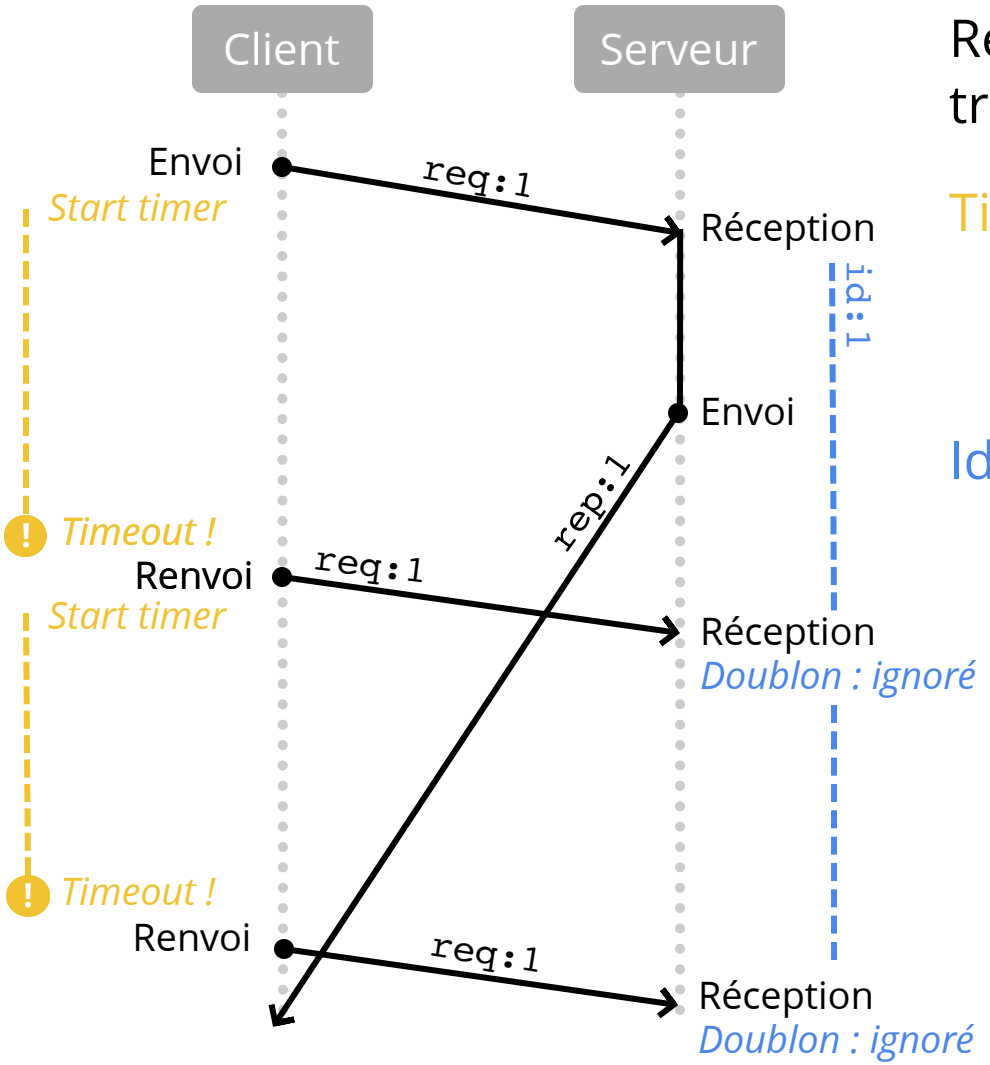
déclenche un renvoi.
est annulé à la réception.

Identificateur

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

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

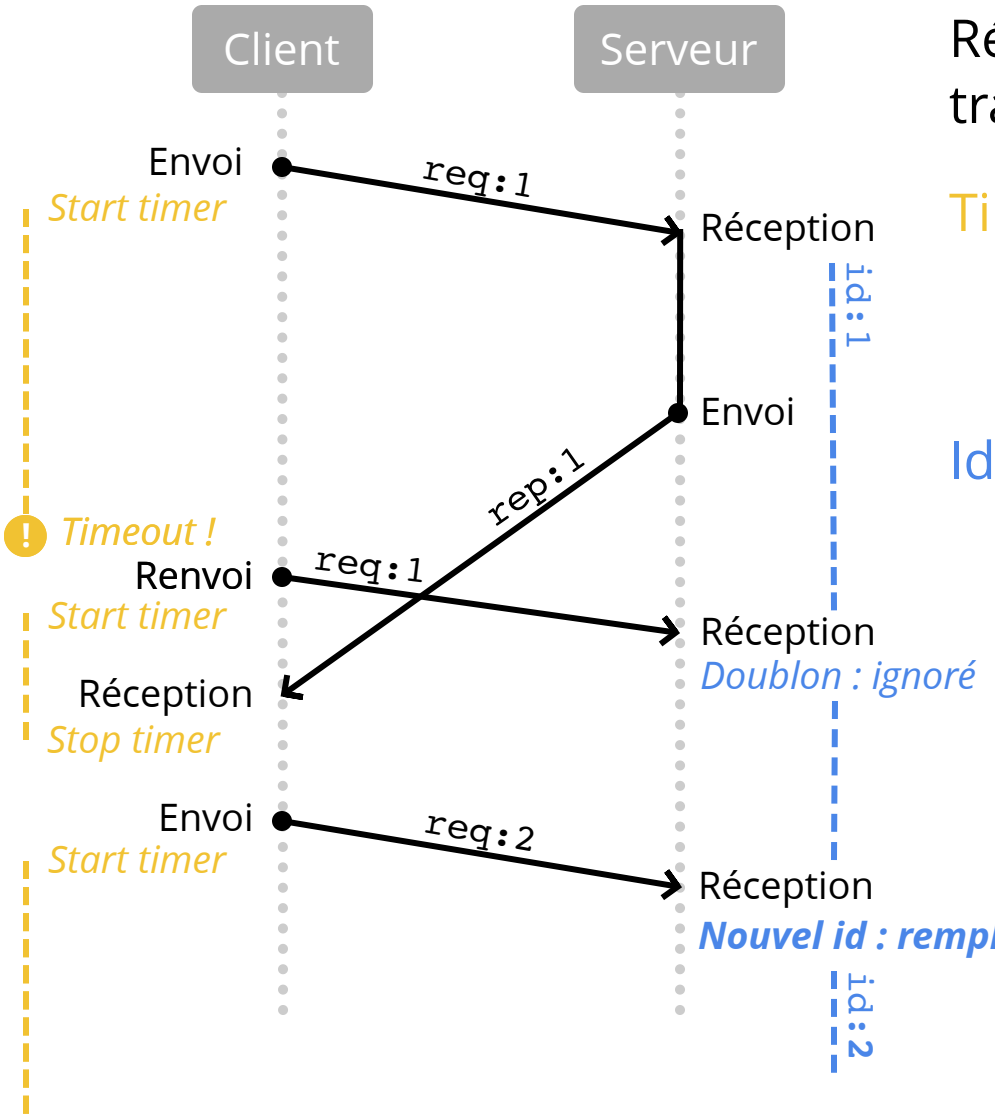
déclenche un renvoi.
est annulé à la réception.

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.

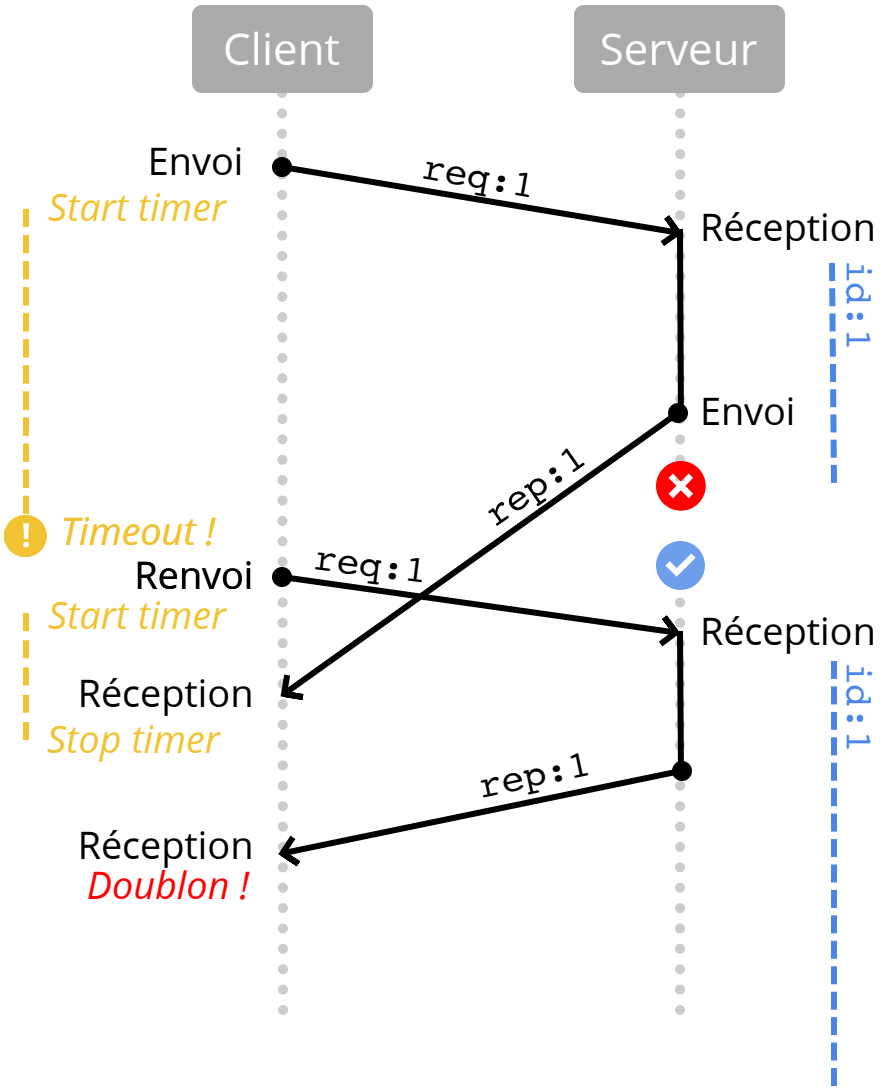
Identificateur

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

Problème ?

Protocoles de fiabilité

Protocole **RR** (Request-Reply)



Réponse envoyée en fin de traitement.

Timeout

déclenche un renvoi.
est annulé à la réception.

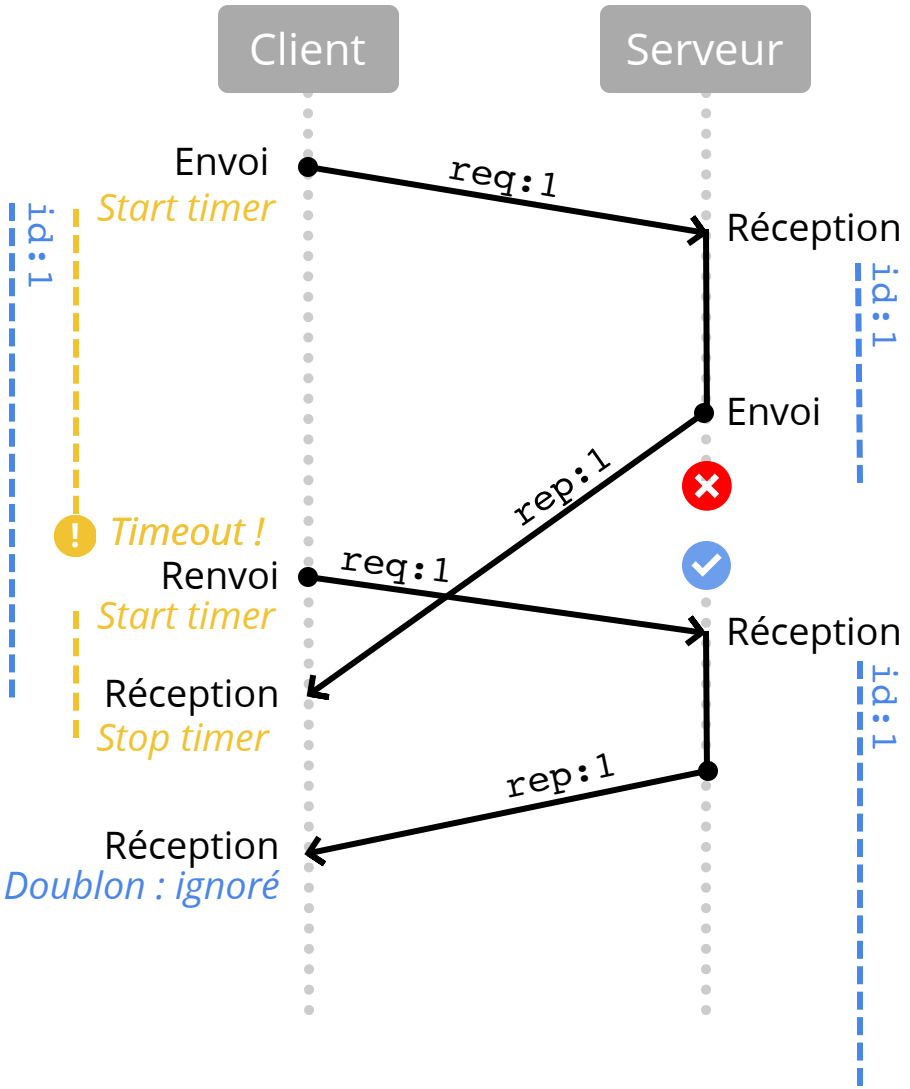
Identificateur

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

Problème ?
Risque de reception double.

Protocoles de fiabilité

Protocole **RR** (Request-Reply) *(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 ?

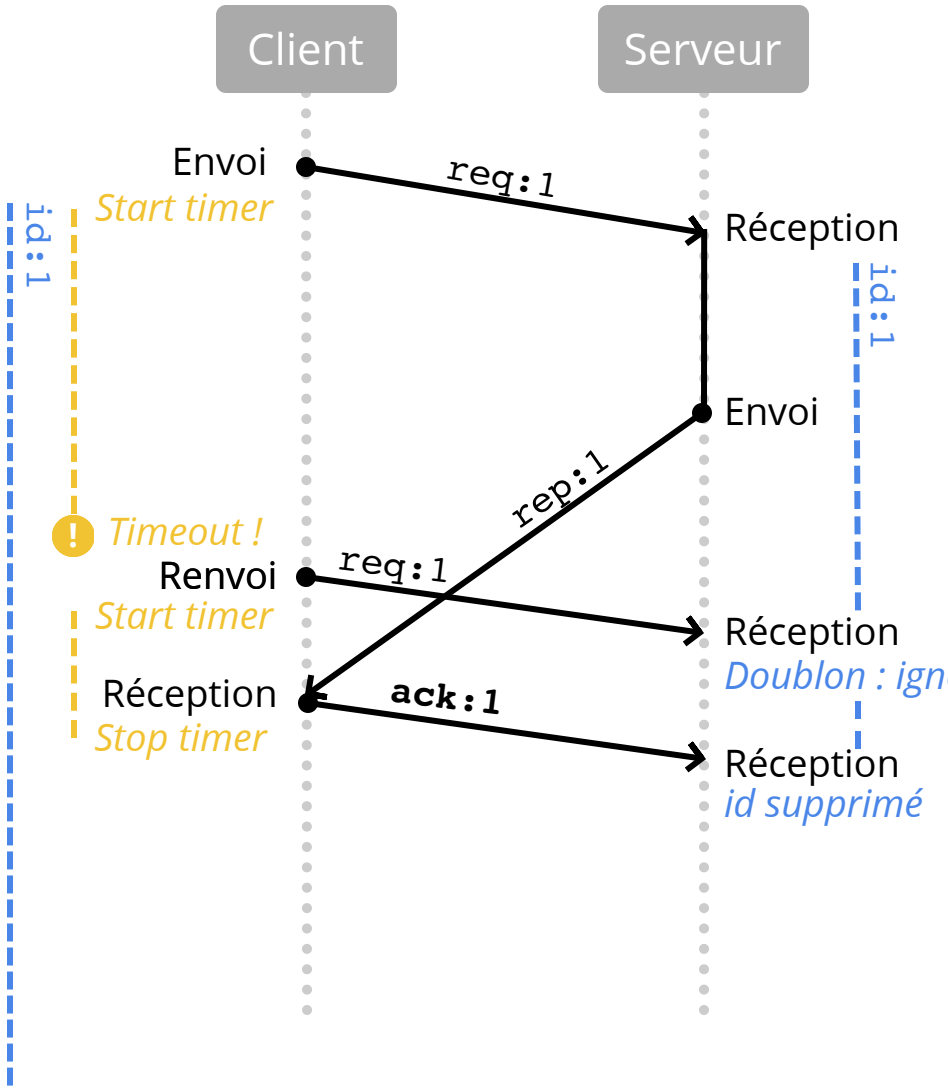
Si requêtes rares mais beaucoup de clients :
trop d'identificateurs à gérer.

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

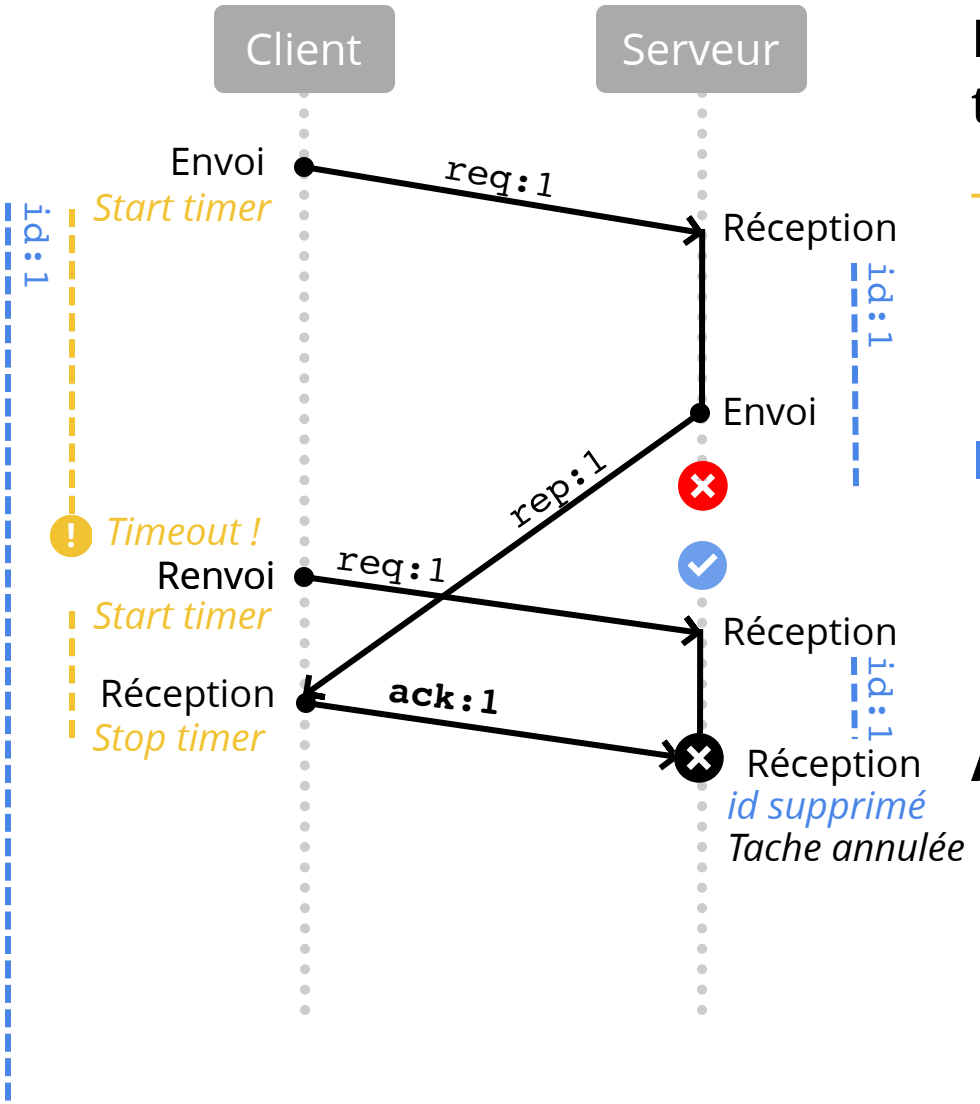
Pour détecter les doublons.
Stocké jusqu'à nouveau message ou **acknowledge**.

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, *et*
de messages oubliés.

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

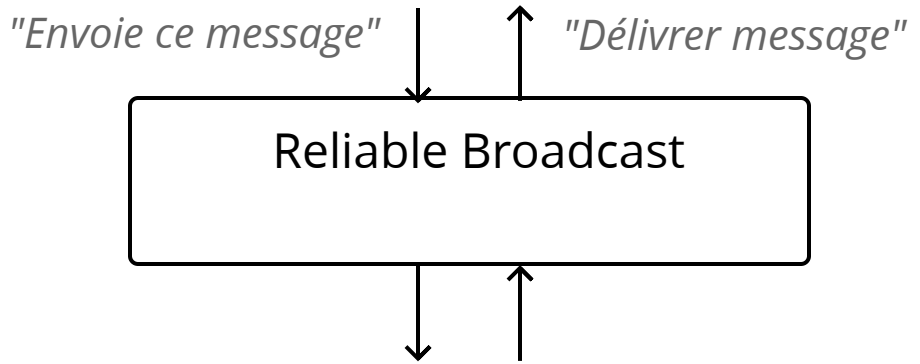
On pourra supposer à partir de maintenant un envoi fiable, type RRA.

Reliable Broadcast

Une diffusion fiable

Reliable Broadcast

Objectifs



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

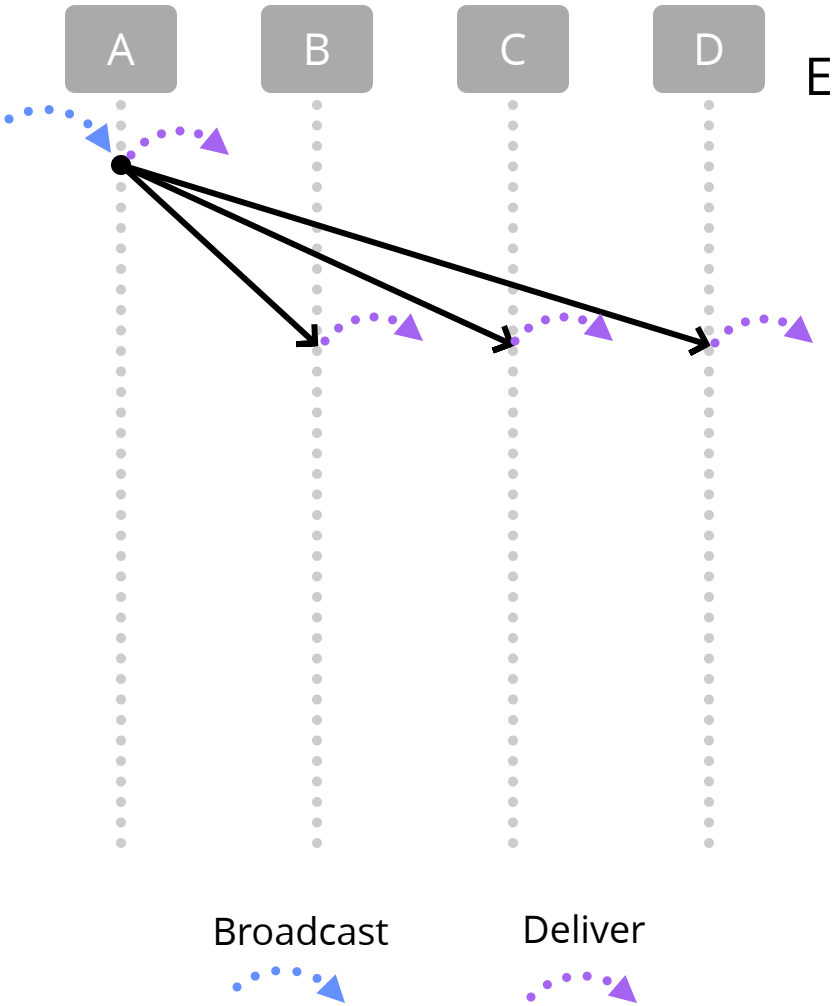
Accord Si un processus correct délivre `m`,
alors tous les processus corrects délivrent `m`

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

Non-duplication Un processus délivre `m` au plus une fois

Broadcast

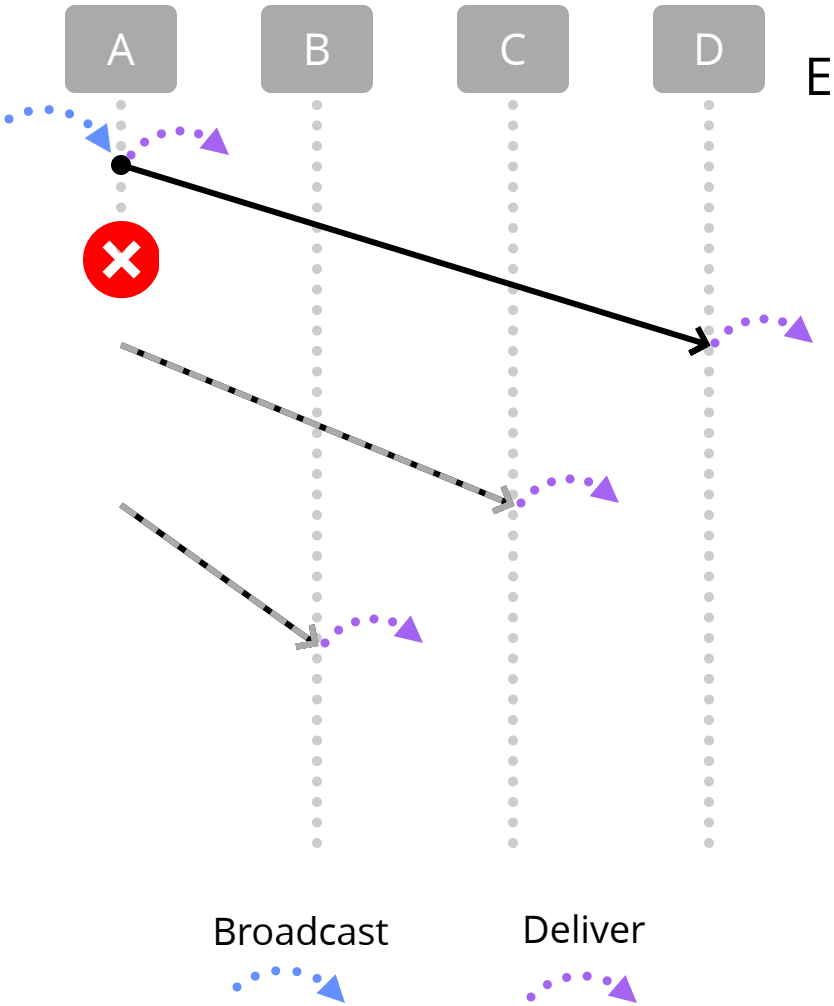
Le plus simple : Best effort



Envoyer à tous les processus.

Broadcast

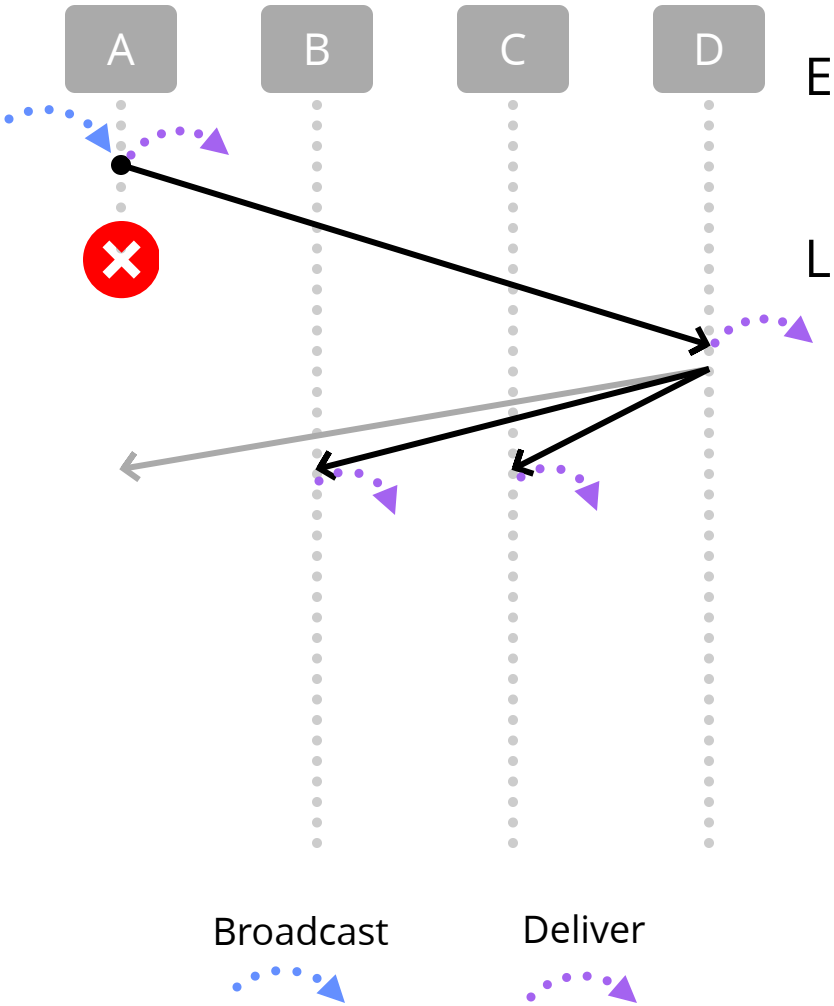
Le plus simple : Best effort



Envoyer à tous les processus.

Broadcast

Tolérance aux pannes : Reliable Broadcast

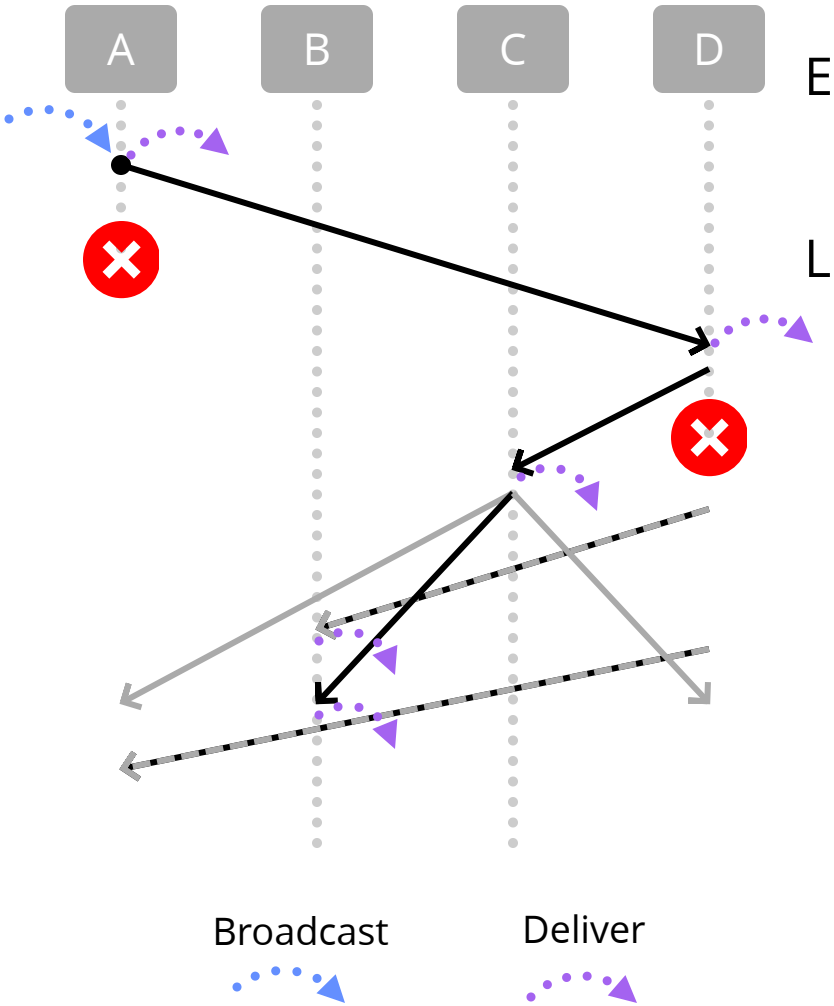


Envoyer à tous les processus.

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

Broadcast

Tolérance aux pannes : Reliable Broadcast

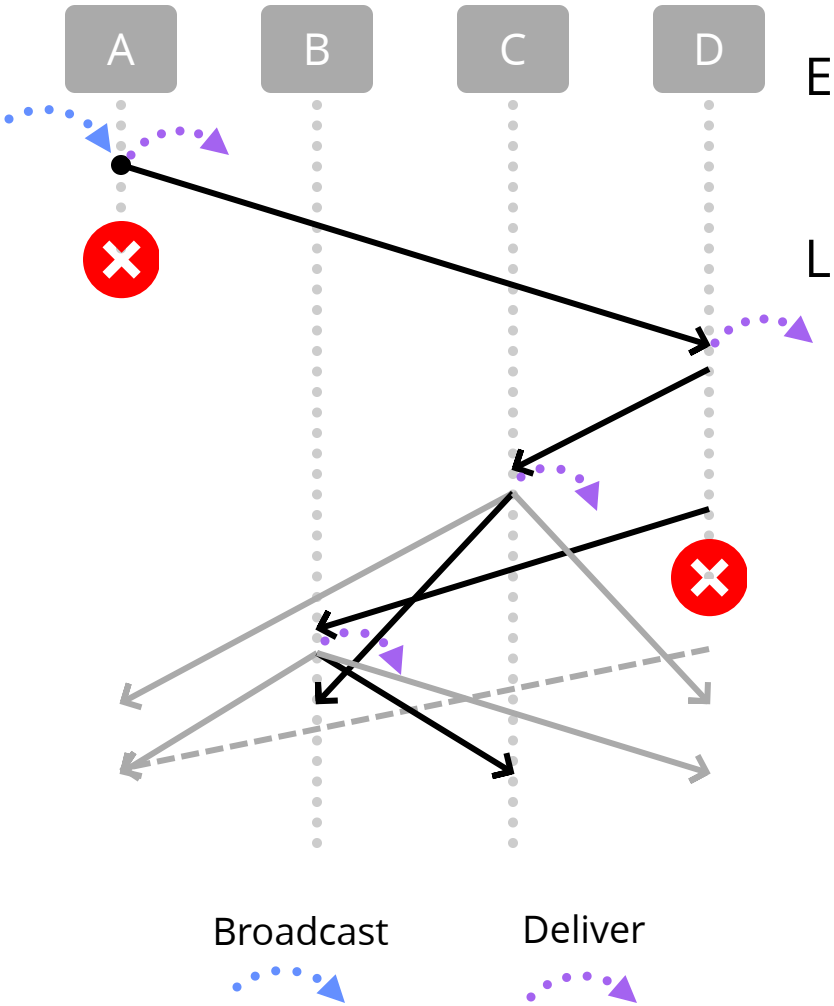


Envoyer à tous les processus.

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

Broadcast

Tolérance aux pannes : Reliable Broadcast



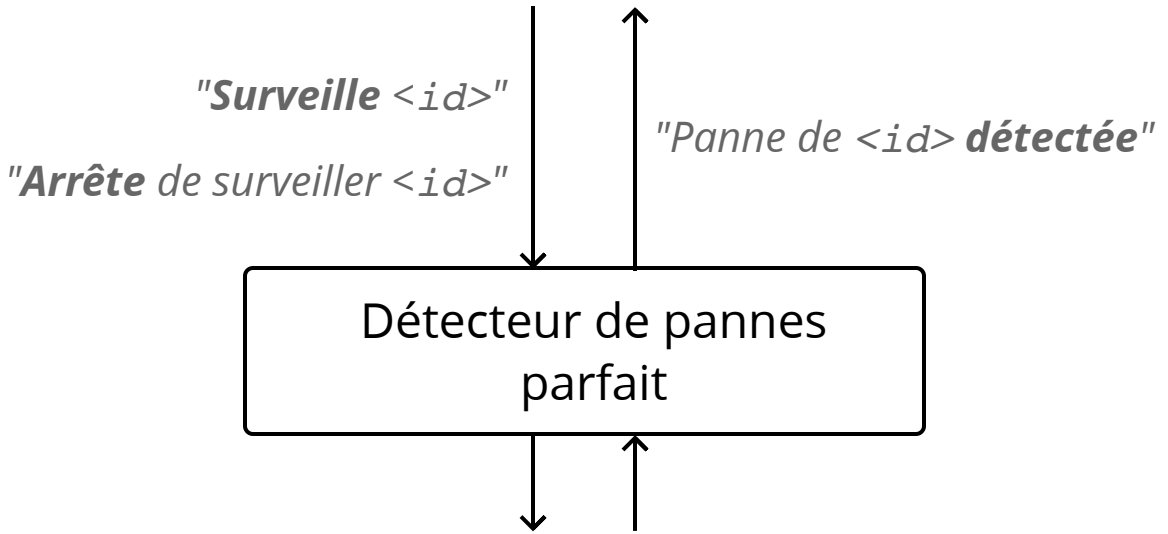
Envoyer à tous les processus.

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

↓ Détecteur de panne parfait

Requirements

Supposons pour l'instant des pannes permanentes uniquement.



Propriétés

- *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.

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.

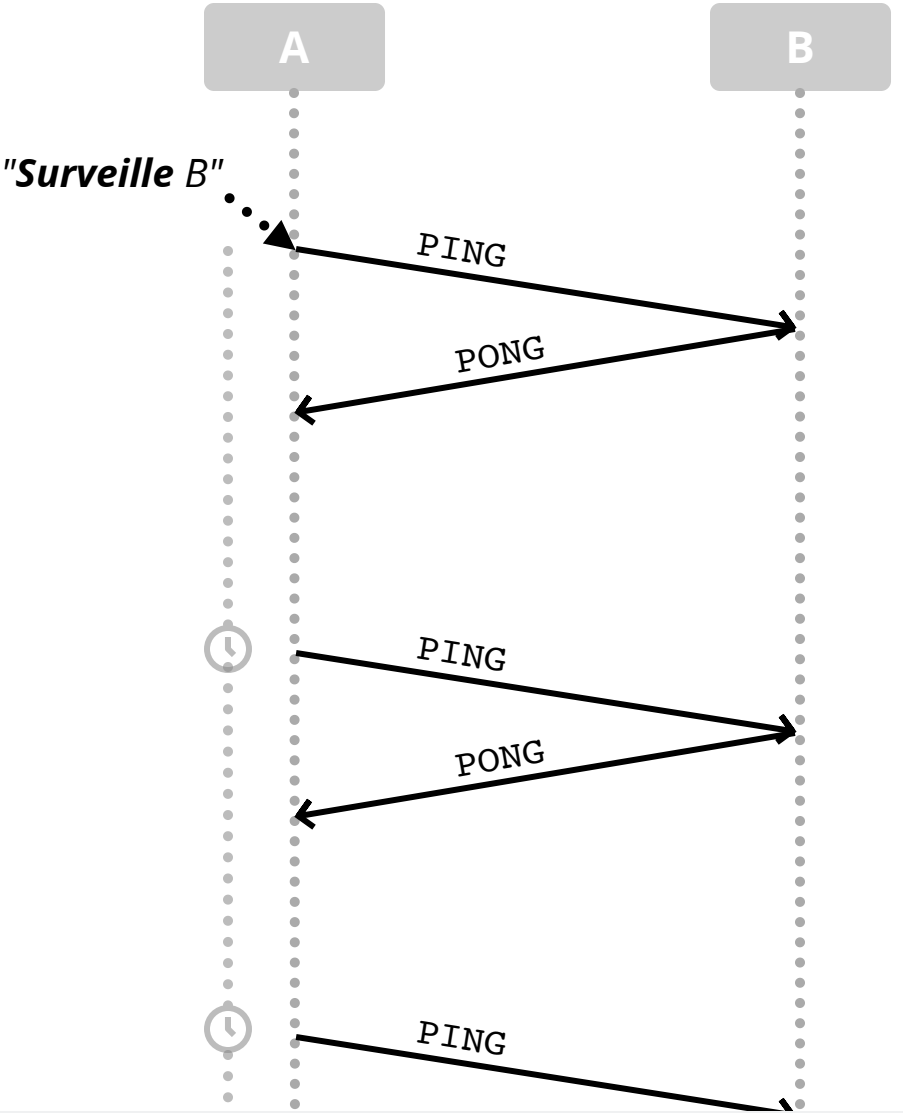
  On parle de système distribué "synchrone".
- Les durées de traitement sont négligeables.

Heartbeat

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`.

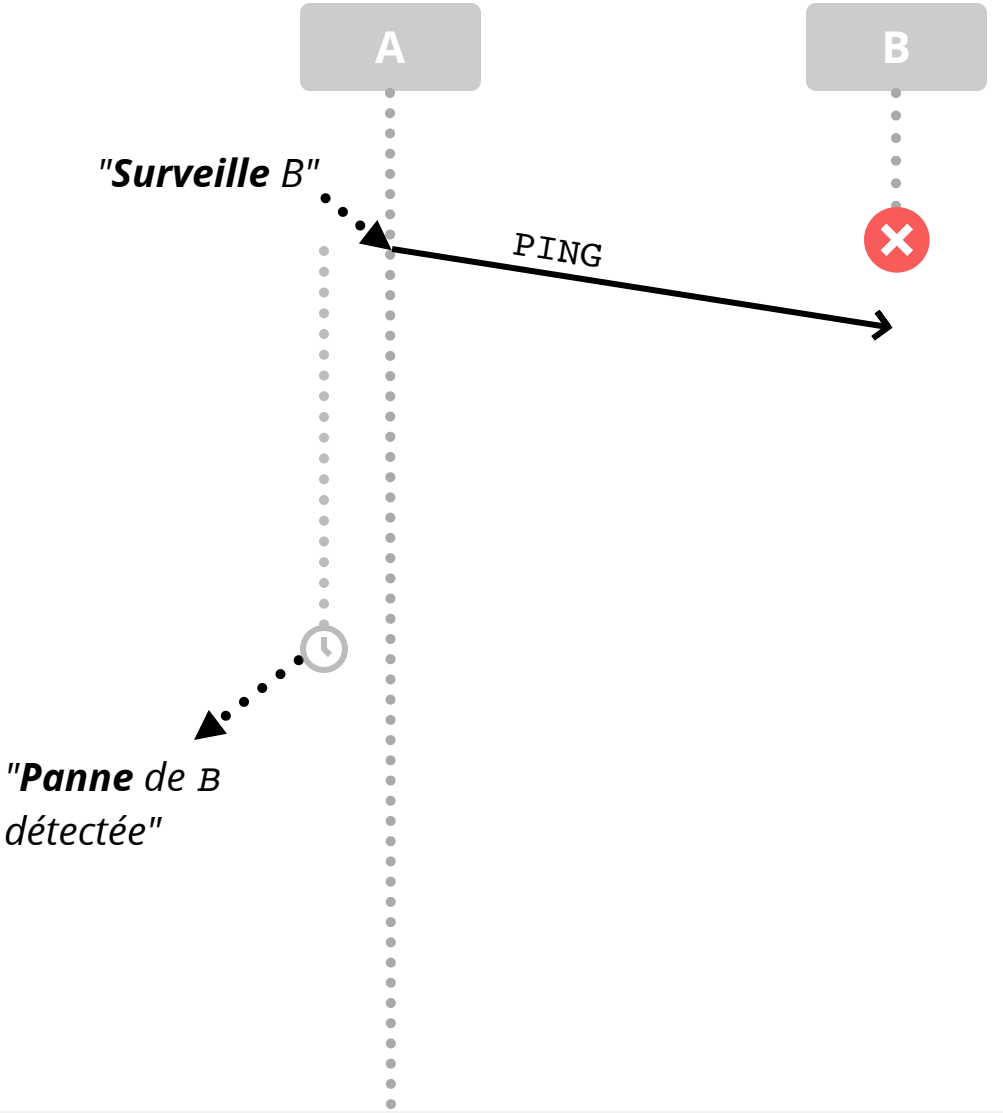


Heartbeat

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."

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`

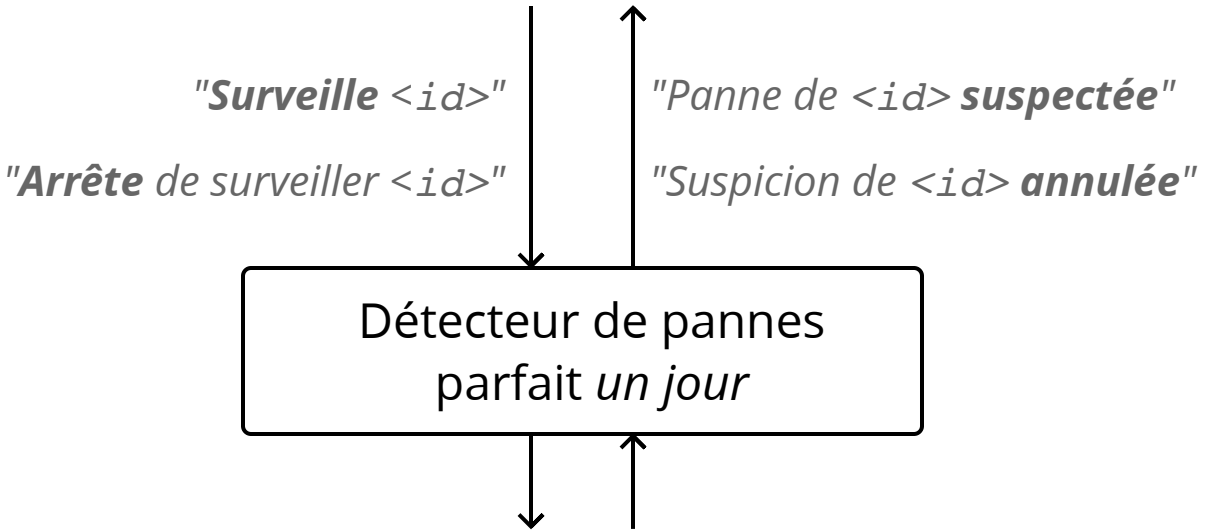
```
Ajouter i à pongs_reçus
```

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



Propriétés

- *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.

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.

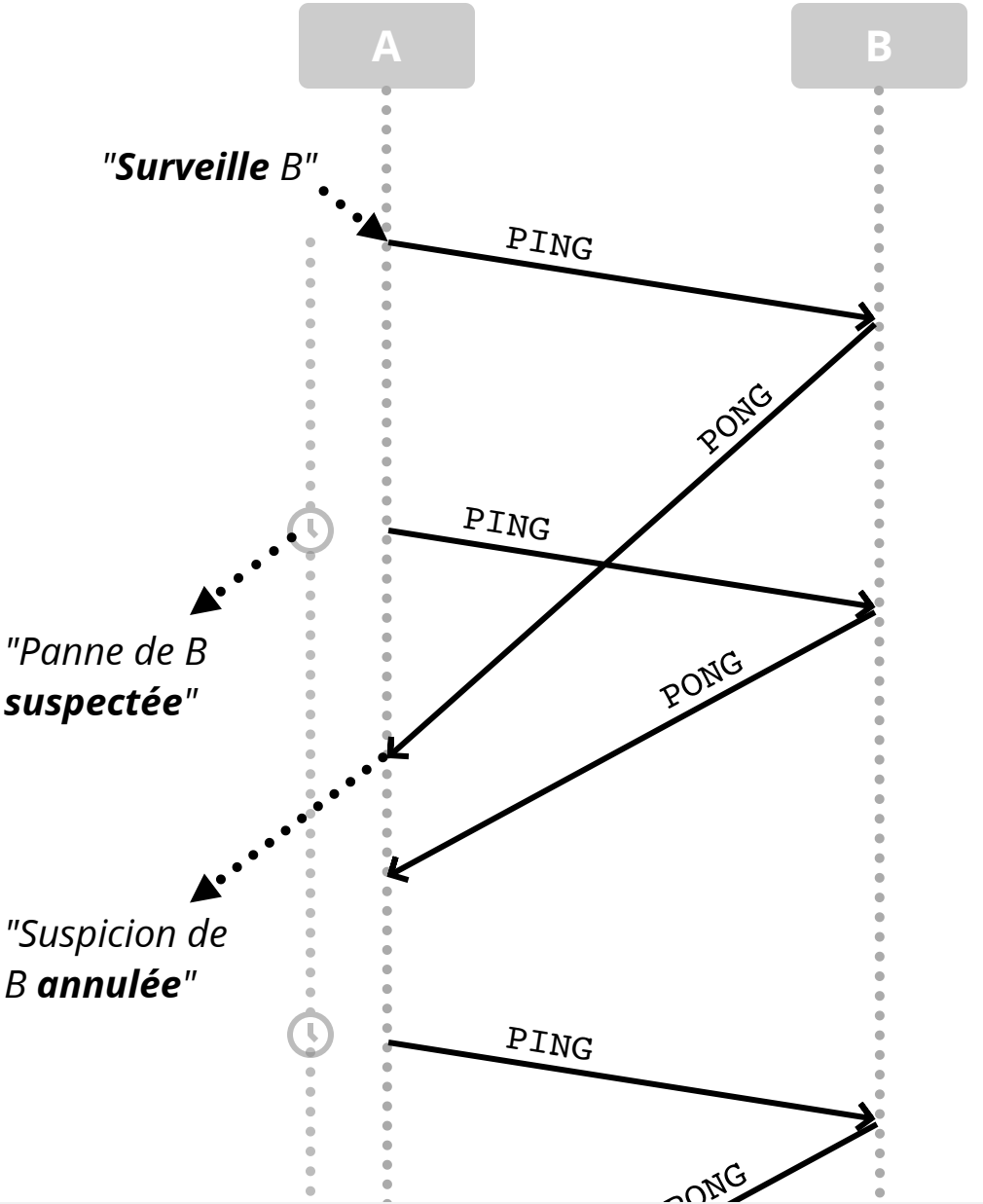
Timeout dynamique

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

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."

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.

Remarque : Δ ne décroît jamais.

Un réseau redevenu rapide garde un détecteur arbitrairement lent.

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.