Simulation à événements discrets

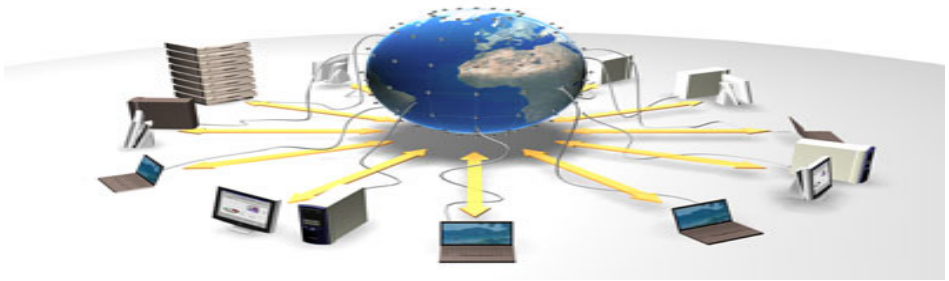
Jonathan Lejeune

Sorbonne Université/LIP6-INRIA

ARA – 2021/2022

Inspiré des supports de cours de Sébastien Monnet

Système distribué : c'est quoi ?

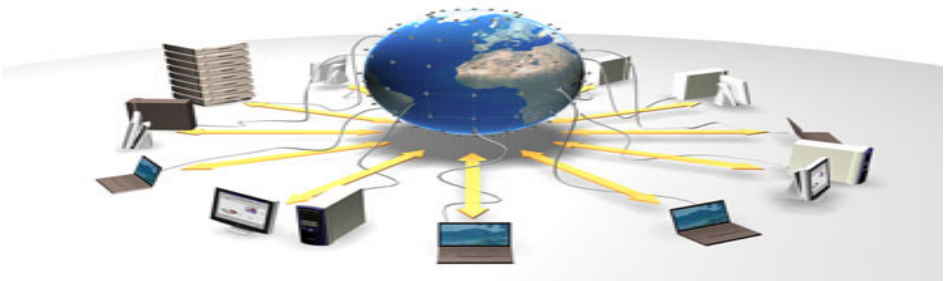


Définition

Une collection d'unités de calcul :

- qui ne partagent pas de mémoire commune et d'horloge globale
- qui communiquent par envois de messages sur un réseau de communication
- où chacune possède sa propre mémoire et exécute son propre OS
- qui sont semi-autonomes et faiblement couplées
- qui coopèrent pour traiter un problème collectivement

Système distribué : c'est quoi ?



Point de vue de *Leslie Lamport, 1987*

"Vous savez que vous êtes dans un système distribué lorsque la panne d'une machine dont vous ignoriez l'existence vous empêche de continuer votre travail."

Point de vue de *A. Tanenbaum, 2003*

"Une collection d'ordinateurs indépendants qui apparaît à l'utilisateur du système comme un seul et unique ordinateur cohérent."

Système distribué : A quoi ça sert ?

Objectif principal

Fournir aux utilisateurs un service informatique fiable et en permanence disponible.

Sous-objectif 1 : Passage à l'échelle

- Parallélisation et distribution de tâches
- Partage de ressources de calcul et de stockage distantes
- Répartition de charge

Sous-objectif 2 : Tolérer les pannes

- Géo-réplication de données
- Réplication de service
- Contrôle des résultats

Problèmes fondamentaux

- Diffusion fiable
- Checkpointing
- Réplication
- Consensus
- Exclusion mutuelle
- Élection de leader

} **Besoin d'algorithmes répartis**

Comment tester les algorithmes/systèmes répartis ?

Problèmes de tests sur des conditions réelles

- Nombre de sites généralement limités
  - $\Rightarrow$ Coût élevé
  - $\Rightarrow$ Réservation de ressources parmi plusieurs utilisateurs
  - $\Rightarrow$ Difficulté de tester le passage à l'échelle
- Indéterminisme
  - $\Rightarrow$ Résultats/bugs non reproductibles
- Fautes, latences
  - $\Rightarrow$ Environnement difficilement contrôlable
- Vision globale impossible
  - $\Rightarrow$ Monitoring difficile

Comment tester les algorithmes/systèmes répartis ?

Solution

Concevoir des plates-formes d'évaluation pour :

- Émuler/simuler un grand nombre de nœuds
- Injecter des fautes pour observer la réaction du système
- Vérifier des propriétés algorithmiques
- Évaluer le comportement/les performances du système

Simulation vs. Émulation

Émulation

Reproduction exacte par un logiciel du comportement d'un modèle où toutes les variables sont connues.

Simulation

Imitation d'un comportement physique réel et complexe en suivant un modèle abstrait qui extrapole des variables inconnues.

Peux-t-on simuler ou émuler

- un téléphone ? : émulation et/ou simulation
- un cyclone ? : simulation
- une chute d'un corps ? : simulation
- une machine ? : émulation et/ou simulation

Un comportement émulable est simulable. La réciproque est fausse

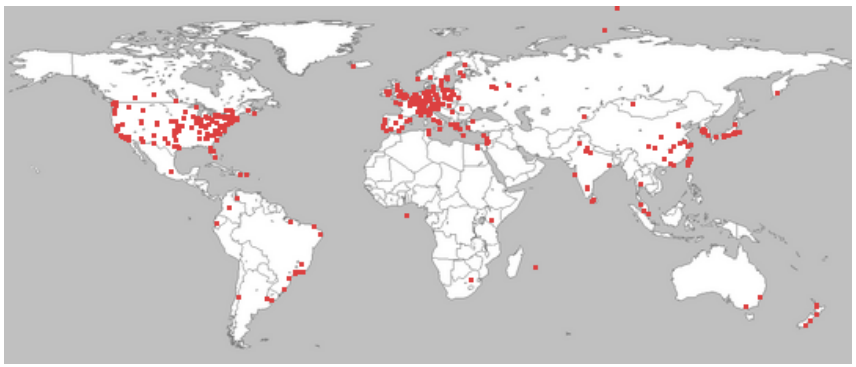
Plates-formes distribuées

Caractéristiques

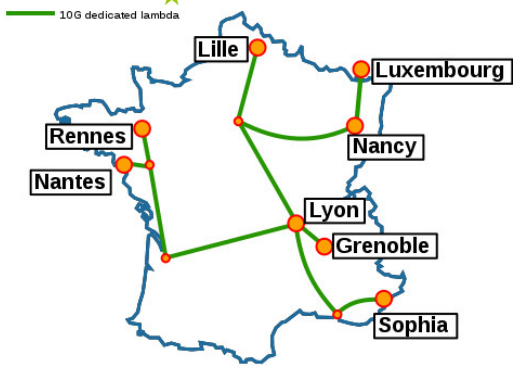
- Un ensemble de sites géographiquement éloignés et reliés par un réseau
- Chaque machine peut émuler plusieurs nœuds virtuels (selon sa capacité)
- Au sein d'une même machine, les communications peuvent être ralenties pour émuler le réseau

⇒ Possibilité d'avoir très grand nombre de nœuds (virtuels)

PLANETLAB
An open platform for developing, deploying, and accessing planetary-scale services



Grid'5000



Avantages/Inconvénients des plates-formes réparties

Avantages

- Bonne montée en charge
- On peut faire tourner la vraie application sur les nœuds (virtuels)
- L'appli est réellement distribuée entre des sites physiquement distants

$\Rightarrow$ Proche d'un essai grandeur nature

Inconvénients

- Ressources précieuses
  - Il faut avoir accès à de telles plateformes
  - Il faut réserver les nœuds par avance et pour un temps limité
- Difficile à prendre en main
  - Des systèmes de déploiement complexes
  - Développement/débugage difficile !

Plates-formes centralisées : les simulateurs

Idée

- Mettre au point un modèle simplifié du système original
- Le simulateur s'exécute (en général) sur une seule machine qui simule l'ensemble des nœuds.
- Simplifications possibles : Application, OS, couches transport/réseau

$\Rightarrow$ mémoire d'une machine est généralement suffisante pour simuler des (centaines de) milliers de nœuds simplifiés.

Plates-formes centralisées : les simulateurs

Objectif

Tester **l'interaction** entre les nœuds du système :

- Les nœuds de l'application sont considérés comme des modules qui échangent des messages
- Il n'est pas toujours nécessaire :
  - de simuler le fonctionnement interne de chaque nœud
  - de représenter chaque donnée

Network Simulator (ns-{1,2,3})

Avantages

- Open Source (en C++)
- Simule précisément les protocoles réseau (TCP, WiFi, etc.)
- Très répandu dans la communauté réseau

Inconvénients

- Passage à l'échelle ?
- Très orienté "réseau"

OMNeT++

Avantages :

- Très générique : simule des modules qui échangent des messages
  $\Rightarrow$ Peut simuler des réseaux, des architectures multi-processeurs, des applications multi-threadées, etc.
- Permet de générer :
  - des graphes de séquence
  - une représentation graphique de la topologie
  - ...

Inconvénient :

Trop générique ?

SimGrid

Avantages :

- Très performant
- Modélisation du réseau réaliste
- Interfaces MSG, pseudo-posix et MPI
- Fonctionne sur le modèle d'un OS
- Écrit en C, et propose de nombreux bindings (Java, Ruby, ...)

Inconvénients :

- Prise en main difficile ...

Peersim

Avantages :

- Moins de 20000 lignes de code Java (Javadoc comprise)
- API très simple d'utilisation (comparé aux autres simulateurs)
- Peut se résumer à un moteur de simulation a événements discrets

Inconvénients :

- Parfois un peu trop simpliste (pas de simulation fine des protocoles réseau, ni même de gestion de bande passante)
- Gestion de topologie dynamique non intégré

En résumé :

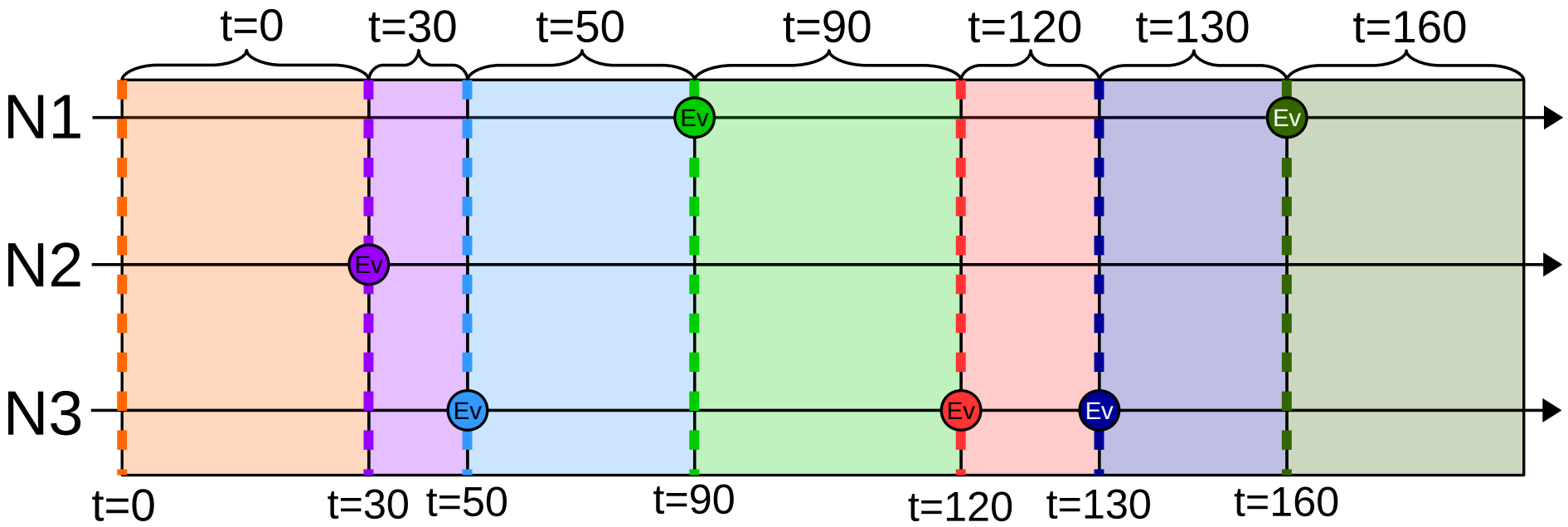
- Soit on teste le vrai système sur une plate-forme répartie
- Soit on conçoit un modèle simplifié et on teste dans un simulateur

La deuxième solution peut être satisfaisante dans de très nombreux cas.

Simulation à événements discrets

Principes généraux

- Deux entités : les nœuds et les événements
- Le temps évolue seulement lorsqu'un événement survient sur un nœud
  $\Rightarrow$ **Discrétisation du temps**
- On teste la réaction du système aux événements
- On ne se préoccupe pas du comportement du système entre les év.



Les événements

Attributs

1) Une estampille temporelle $\Rightarrow$ **sa date de délivrance**.
2) Un nœud $\Rightarrow$ **son destinataire**.
3) Un objet $\Rightarrow$ **son type et ses données**.

Exemples d'événement

- Réception d'un message
- Panne
- Appel d'une primitive applicative

Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

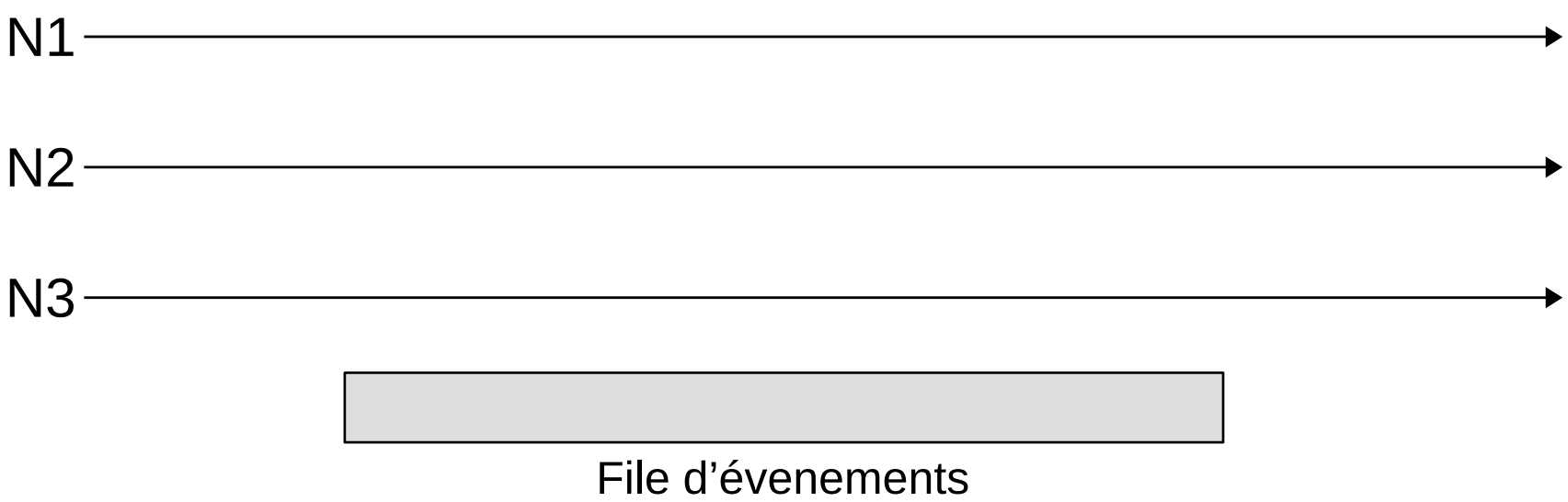
  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

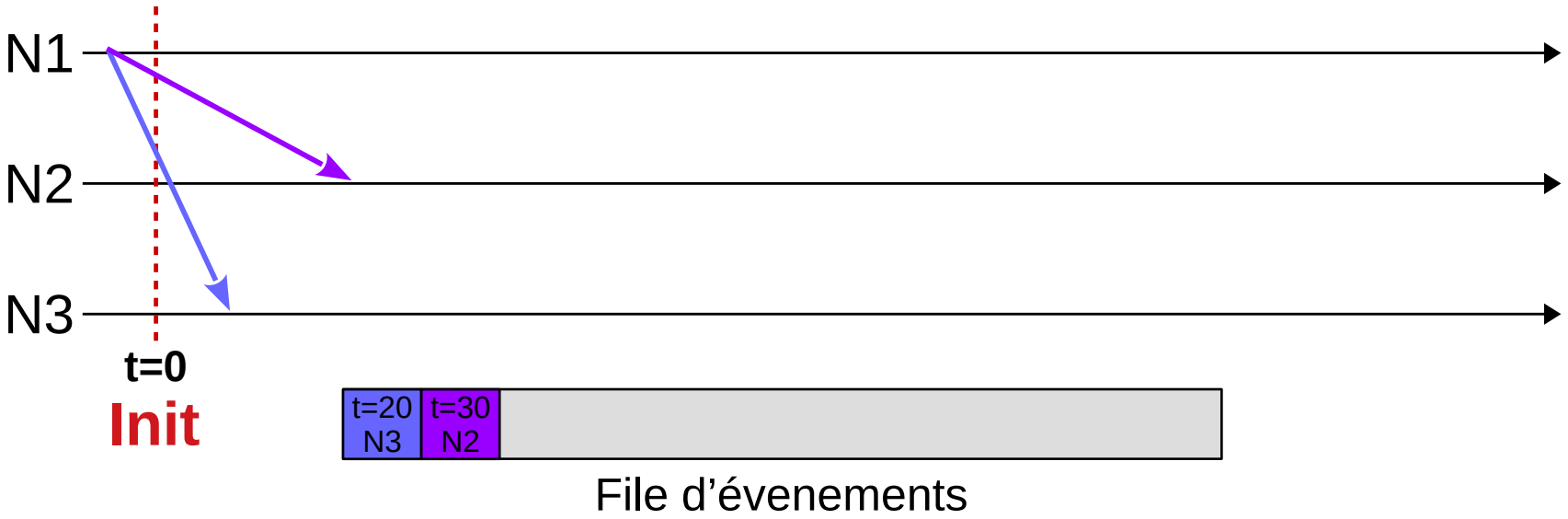


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

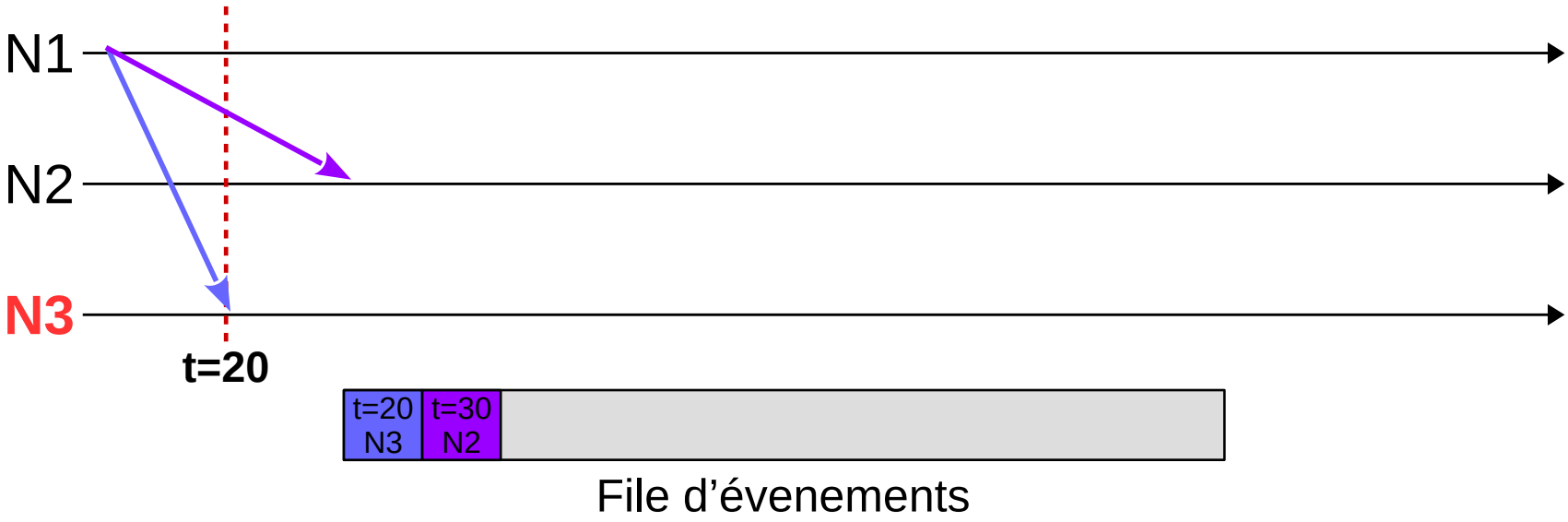


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

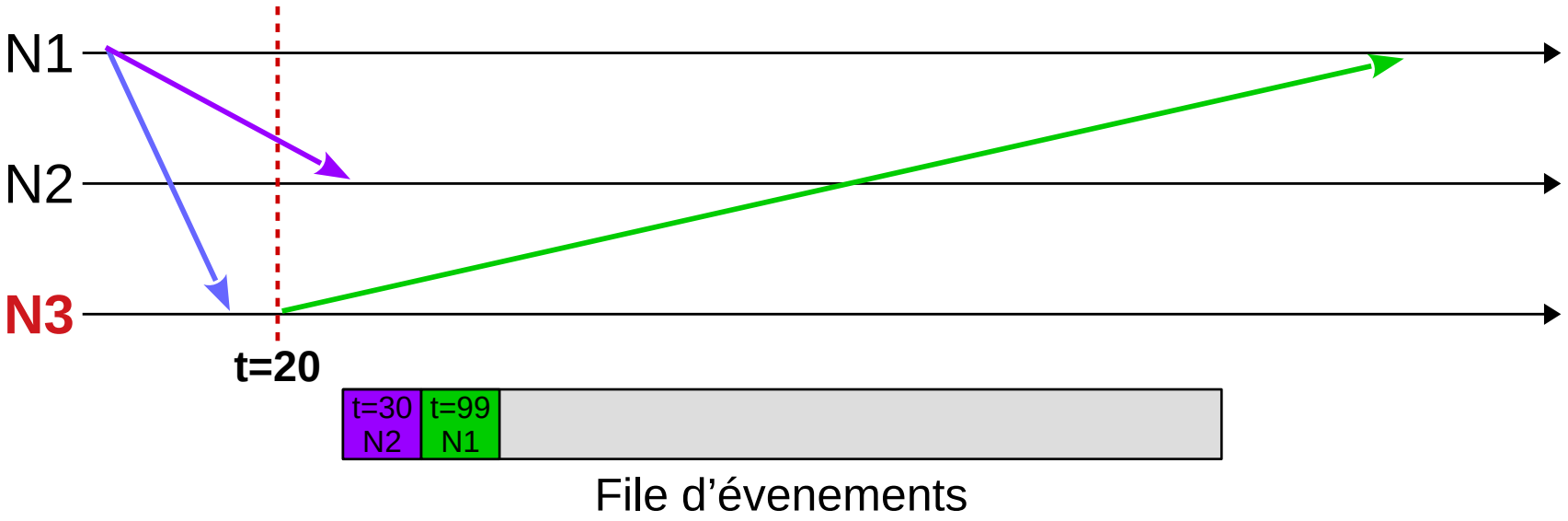


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

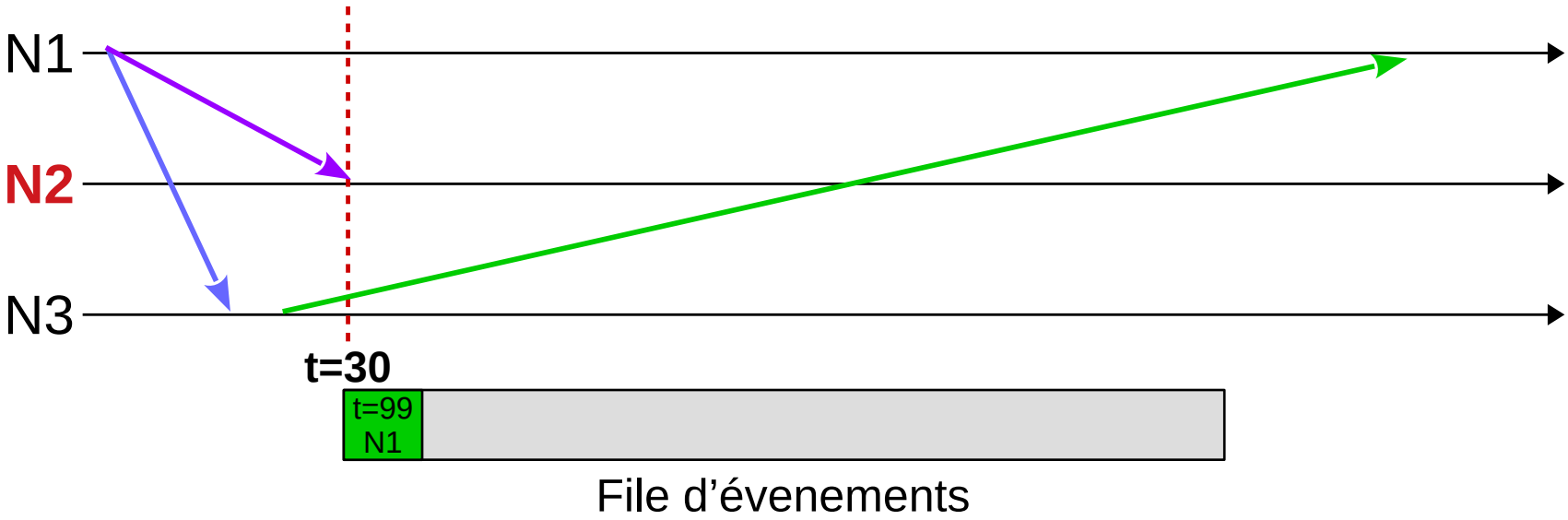


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

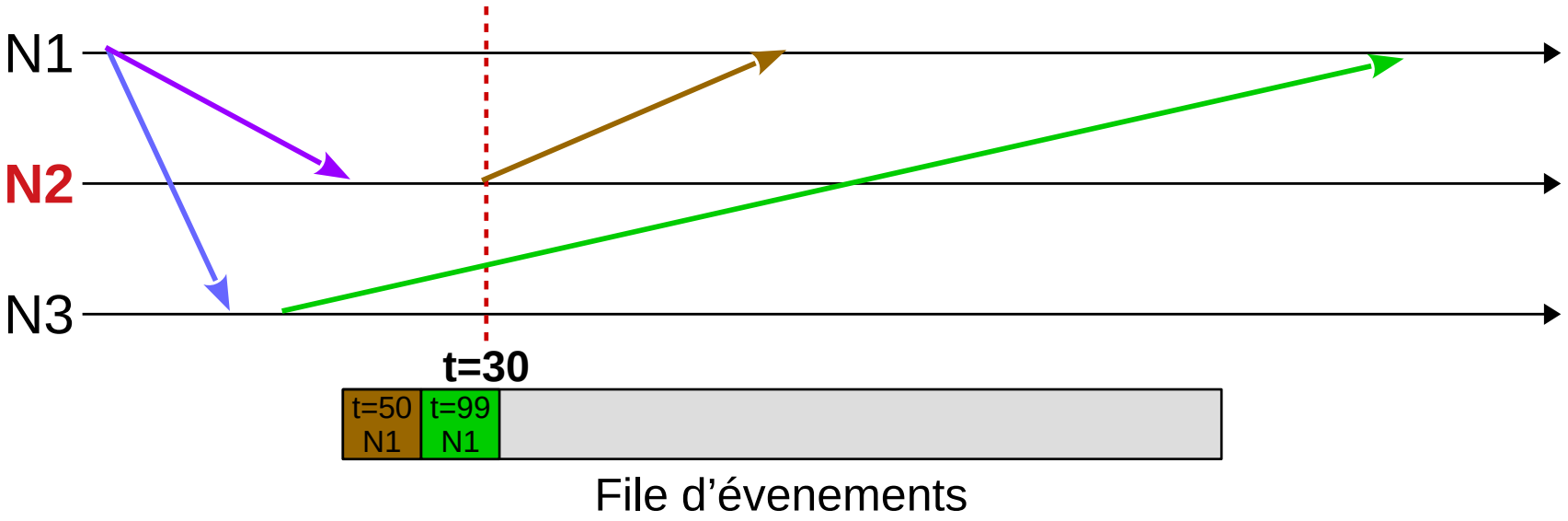


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

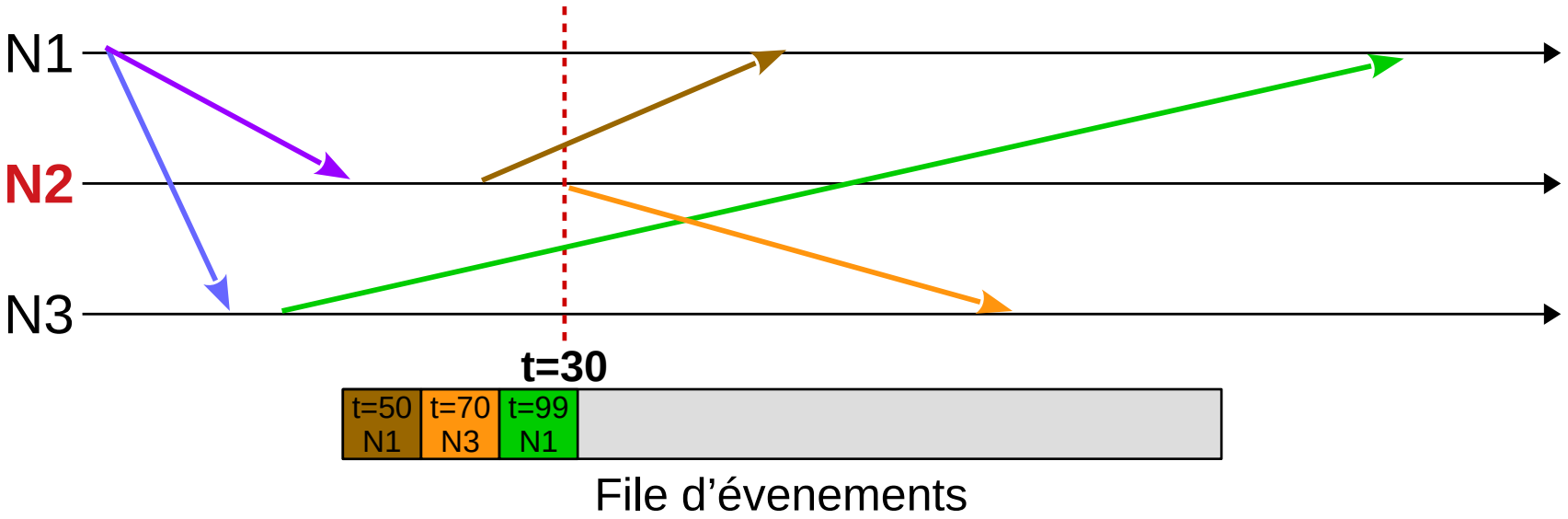


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

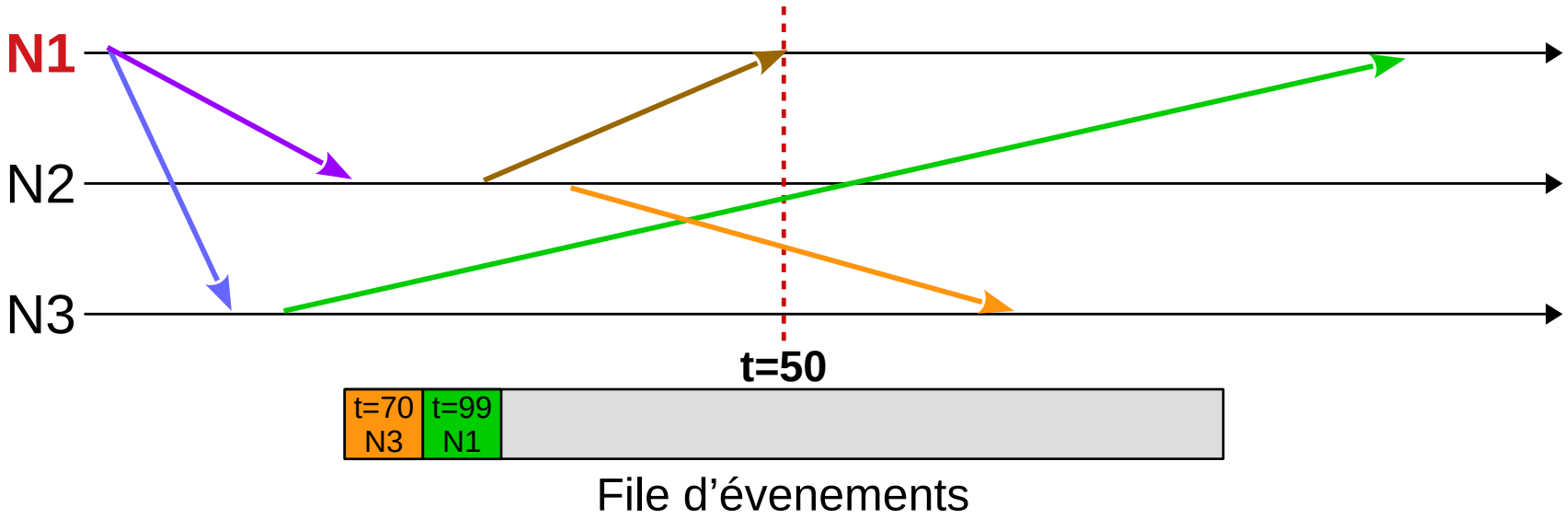


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

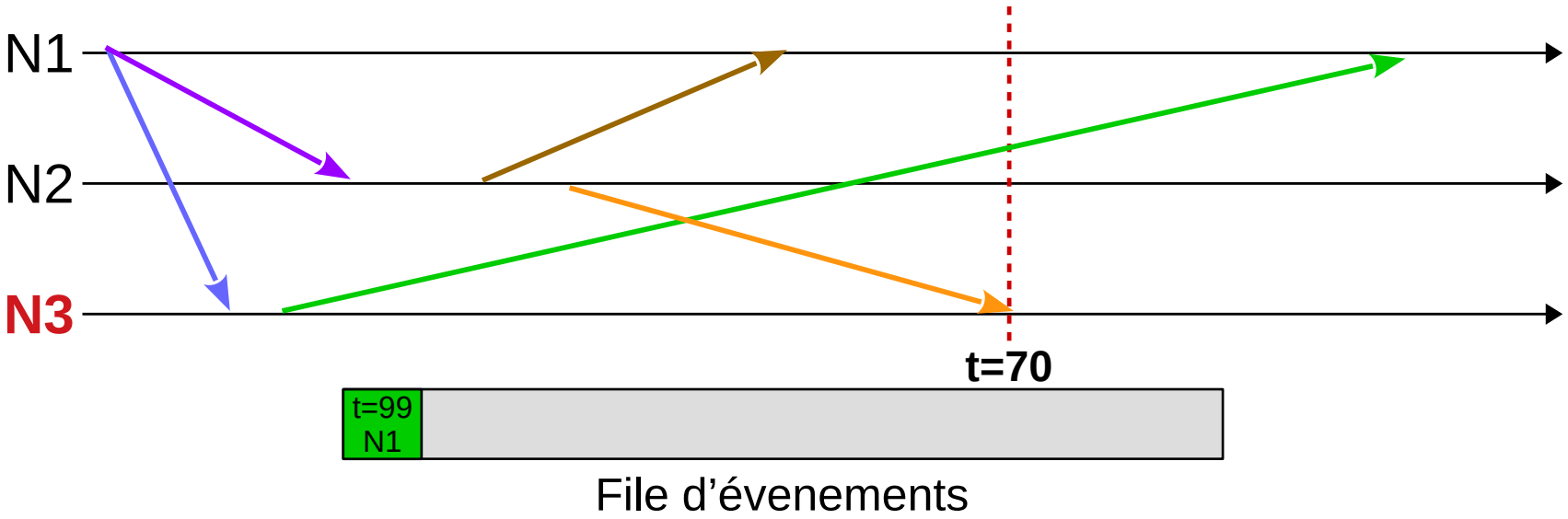


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

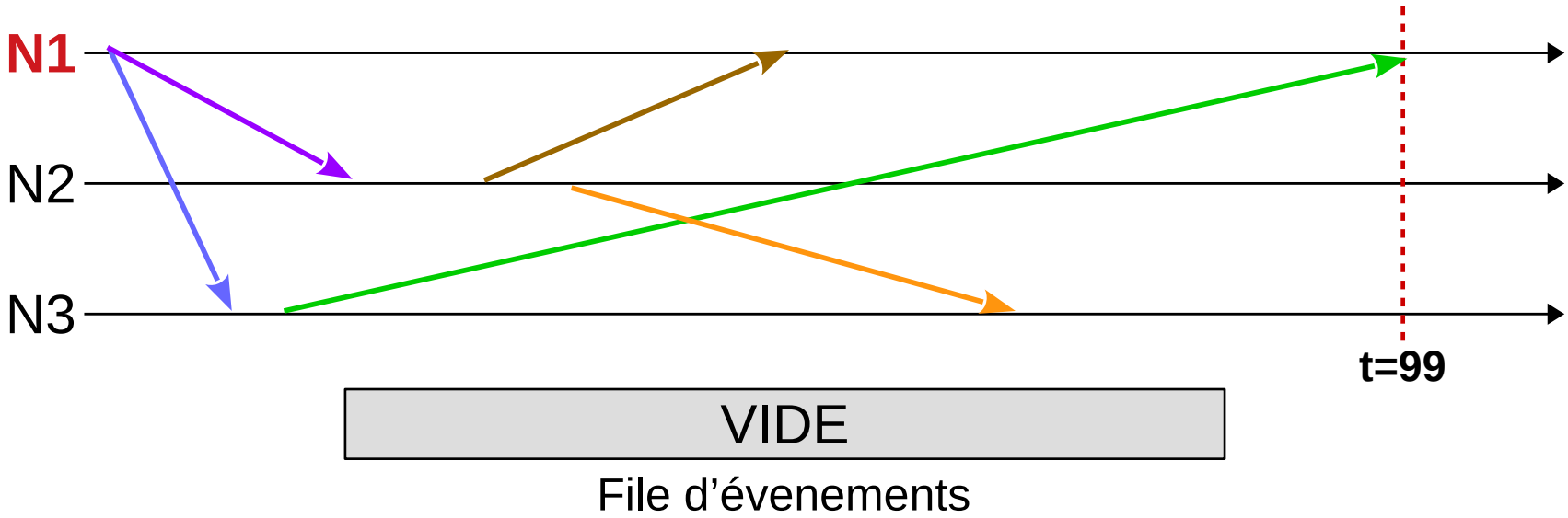


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire

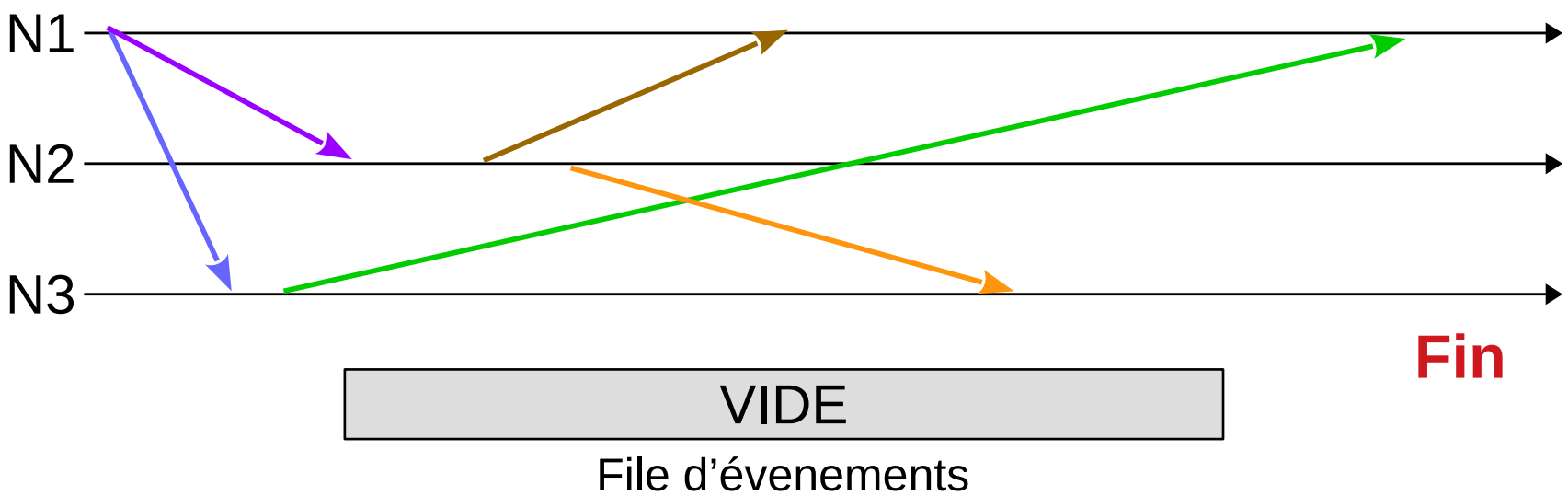


Algorithme général

Principes

- Chaque événement créé est inséré dans une file triée par estampille croissante
- Tant que file non vide et temps courant $<$ temps max de simulation :

  récupérer l'événement *ev* en tête de file
  temps courant $\leftarrow$ estampille de *ev*
  délivrer *ev* au nœud destinataire



Avantages/Inconvénients

Avantages

- Le comportement du système entre les événements n'est pas simulé
  $\Rightarrow$ charge de calcul allégée
- Simulation reproductible
  $\Rightarrow$ le même bug peut être rejouer jusqu'à sa résolution
- Vision globale du système (message en transit + date de délivrance)
  $\Rightarrow$ debug facilité
- Tout se fait dans la même machine
  $\Rightarrow$ Possibilité de tricher

Inconvénients

- Trouver le bon compromis de finesse entre :
  - simulation précise au risque d'être ralentie par trop d'événements
  - simulation simpliste au risque de fausser le résultat