Fiabilité algorithmique dans les systèmes distribués

Arnaud Labourel

Section 1

Fiabilité dans les Systèmes distribués

Rappels Fiabilité vs Sécurité

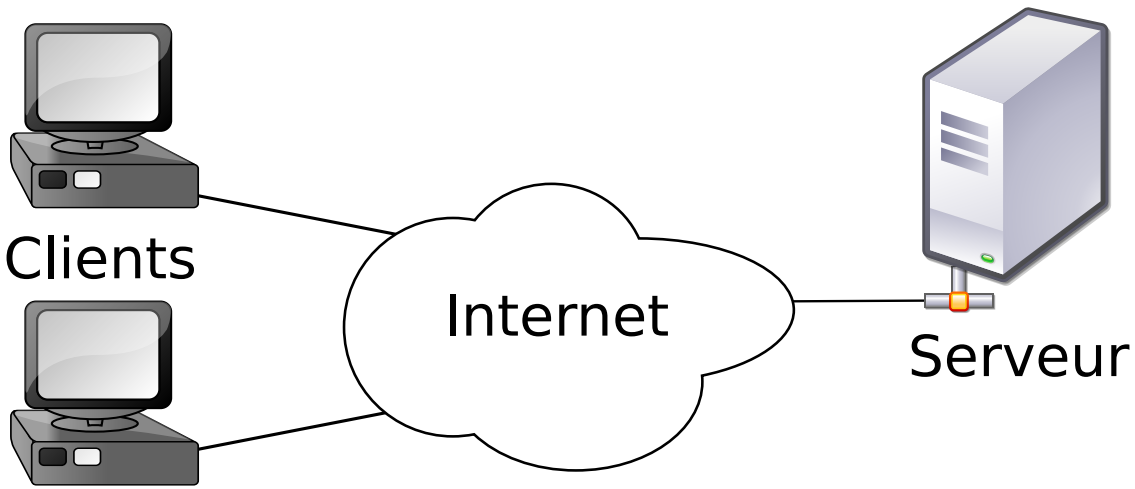
Deux problématiques distinctes :

- **Fiabilité** : le système se comporte toujours comme attendu (spécification) dans le cadre/environnement prévu :
  - qualité logicielle
  - test, simulation, débogage
  - validation, certification
- **Sécurité** : le système résiste à des attaques malveillantes (contexte non prévu)
  - virus, *malware*, *phishing*
  - détection d'intrusion, pot de miel
  - authentification, confidentialité, secret, anonymat, politique de sécurité

Cadre distribué

Toutes les infrastructures actuelles sont distribuées (interactions entre plusieurs machines) :

- paradigme client-serveur
- serveurs (2, 3, n-tiers)
- cloud



Une solution, de nouveaux problèmes ?

La réplication/mobilité résout la problématique des crashs et autres risques de pertes.

$\Rightarrow$ les données étant répliquées, la **haute disponibilité** nécessite de considérer le problème de leur synchronisation.

Section 2

Exemple de systèmes distribués

De la Pratique à la Théorie

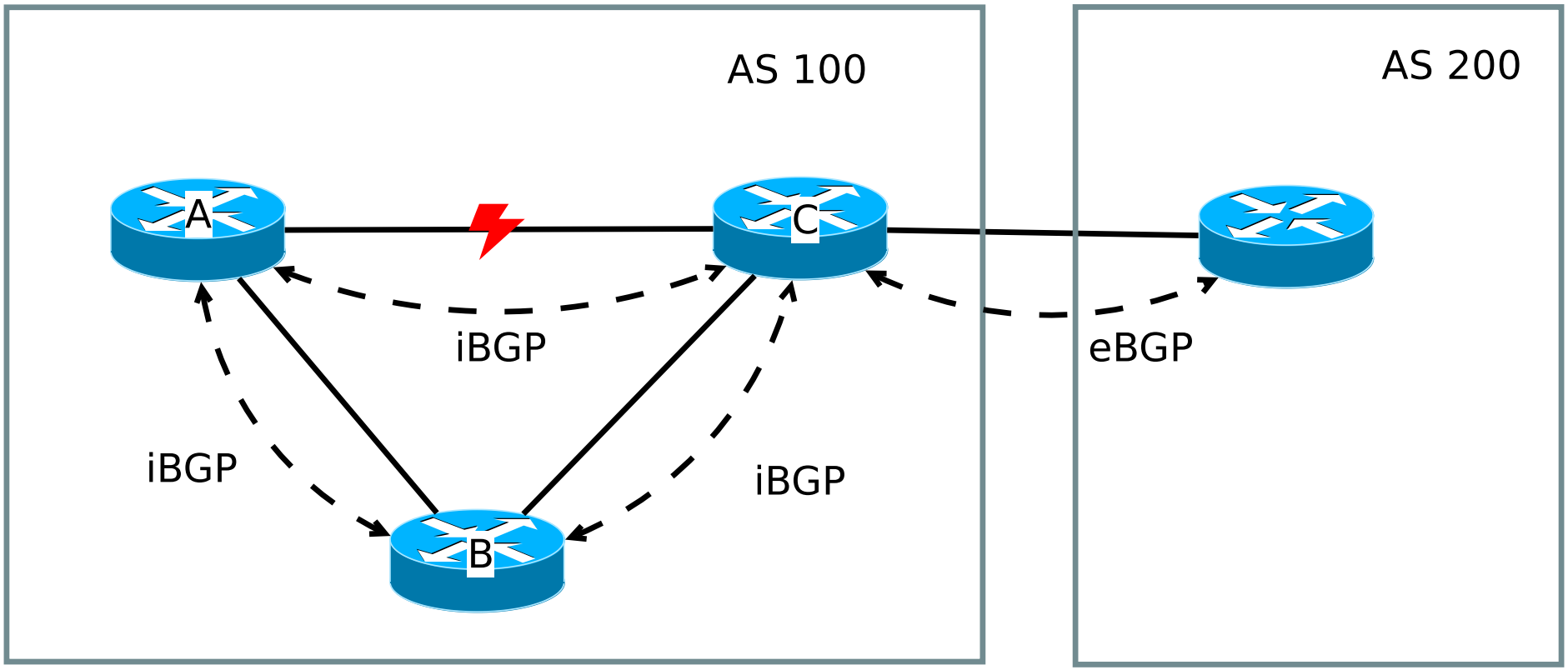
Objectif de cette partie

Présenter les concepts et algorithmes de l'**algorithmique distribuée** (très influencés par la pratique).

Exemples de systèmes distribués

- internet
- applications (git, mastodon, ...)
- *Internet of Things*
- grilles de calcul
- *cluster cloud*
- *blockchain*
- ...

Internet



Border Gateway Protocol

Connecte entre eux des *Autonomous System*.

Internet

Protocole IP : Routage et transmission de paquets

Définition : algorithme distribué ayant pour objectif d'acheminer des données depuis une source jusqu'à une destination.

Difficultés:

- le réseau évolue, son graphe est dynamique
- impossible de maintenir localement une connaissance **complète**

et **en temps réel** de la topologie globale.

$\Rightarrow$ deux problématiques

1. acheminement à l'aide d'une information **locale**, présente immédiatement sur le routeur.
2. maintenance (en parallèle) de ces informations locales par **échange global** d'informations

Applications Client/Serveur

- paradigme client/serveur
- architecture centralisée ou non
- protocole :
  - à état : enregistre l'état d'une session entre 2 requêtes (exemples : TCP, FTP)
  - sans état : n'enregistre pas l'état d'une session de communication entre deux requêtes successives (exemples : IP, HTTP)
- n-tiers, micro-services, ...
- modularité et passage à l'échelle

Exemple d'architecture 3-tiers

Presentation tier

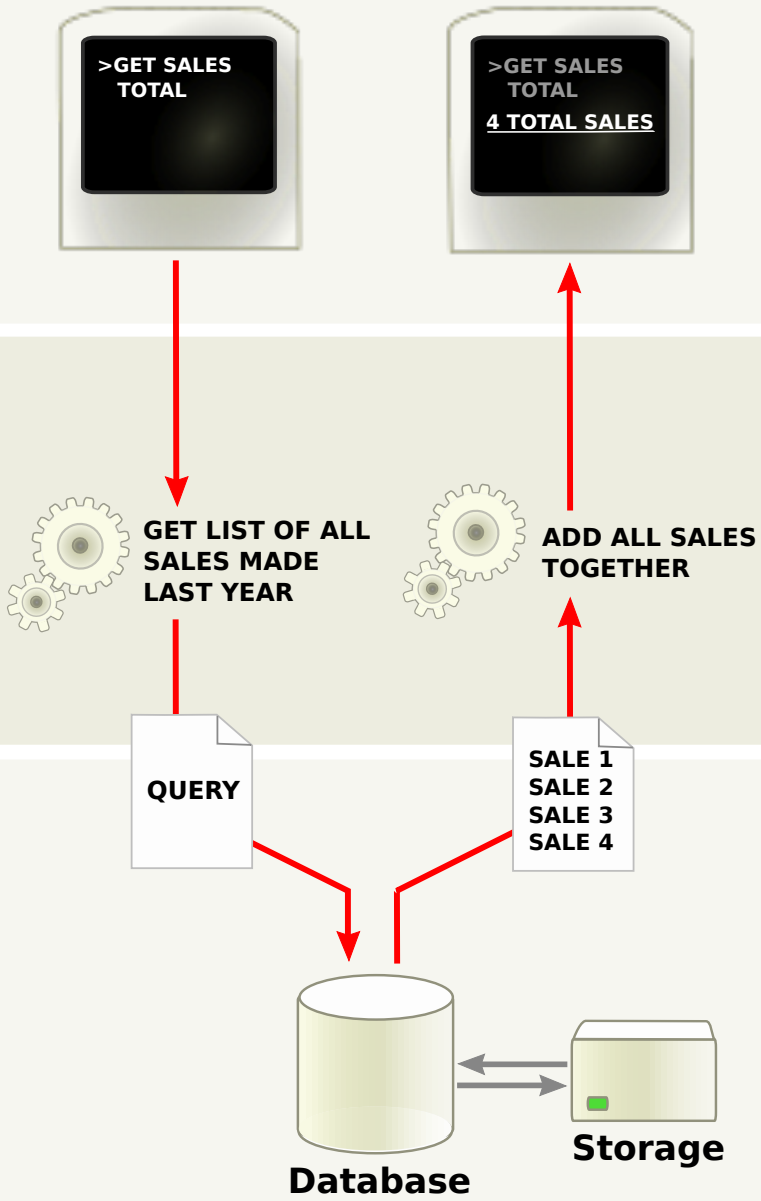
The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

Logic tier

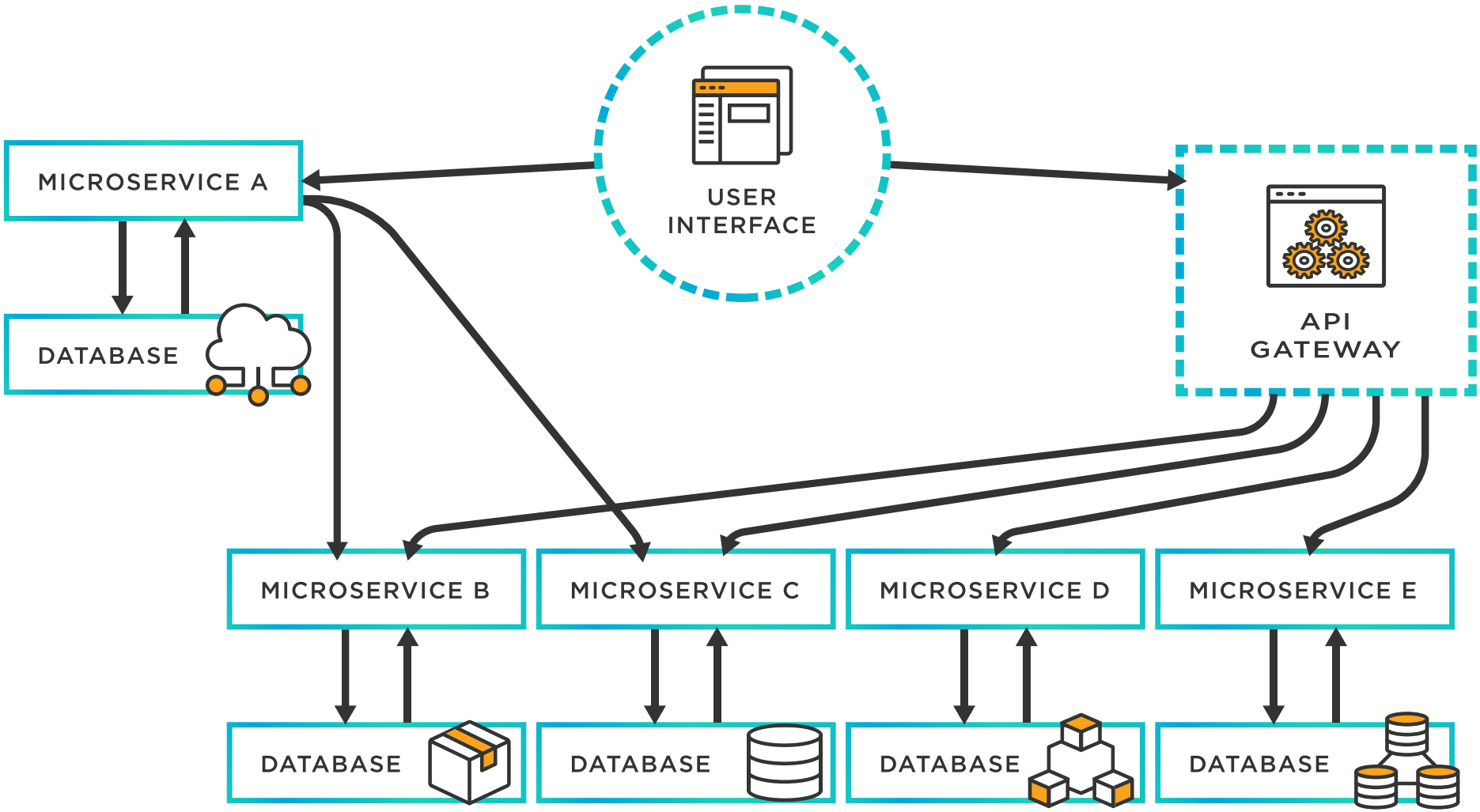
This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

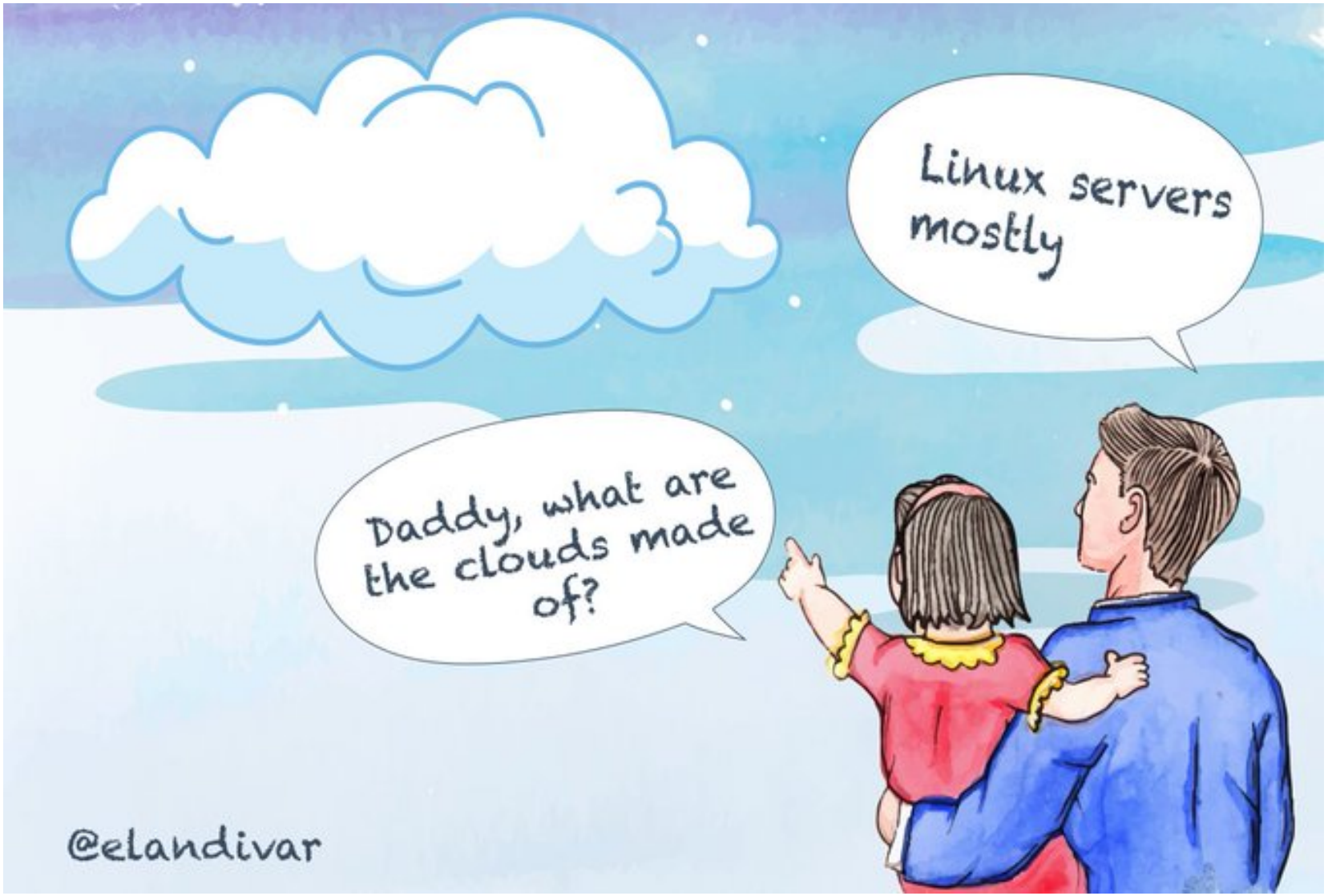
Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.



Exemple d'architecture microservive



Cloud



Source : Edgar Landivar (@elandivar)

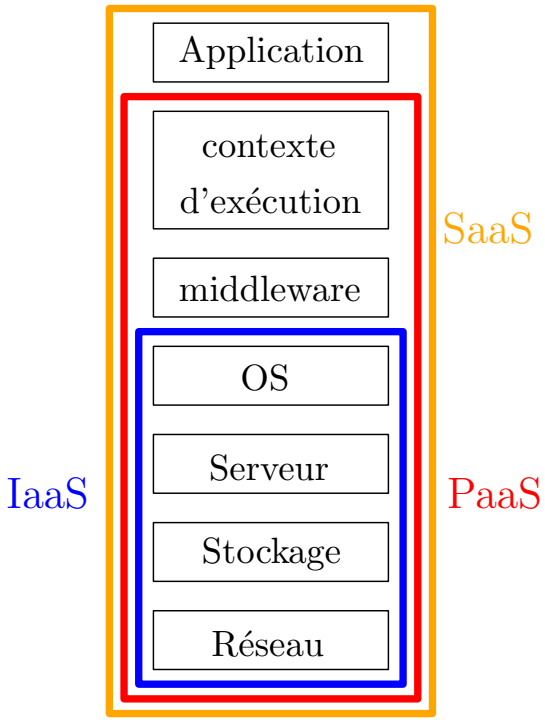
Cloud

À partir du modèle client-serveur,

- les données sont réparties/dupliquées sur plusieurs data-center
- les machines virtuelles sont réparties également

Architecture centralisée : une seule entité contrôle le matériel et la pile logicielle jusqu'à un certain niveau

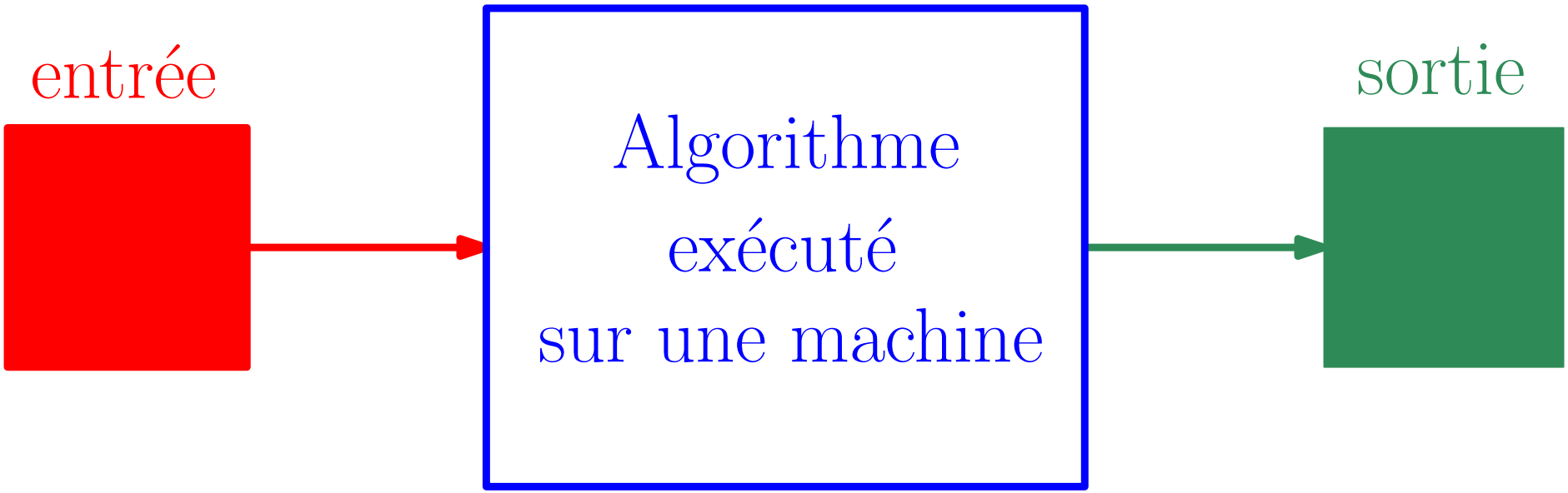
- IaaS : Infrastructure as a Service (accès uniquement à une infrastructure)
- SaaS : Software as a Service (accès à des services en ligne)
- PaaS : Platform as a Service (accès à une plateforme applicative)



Section 3

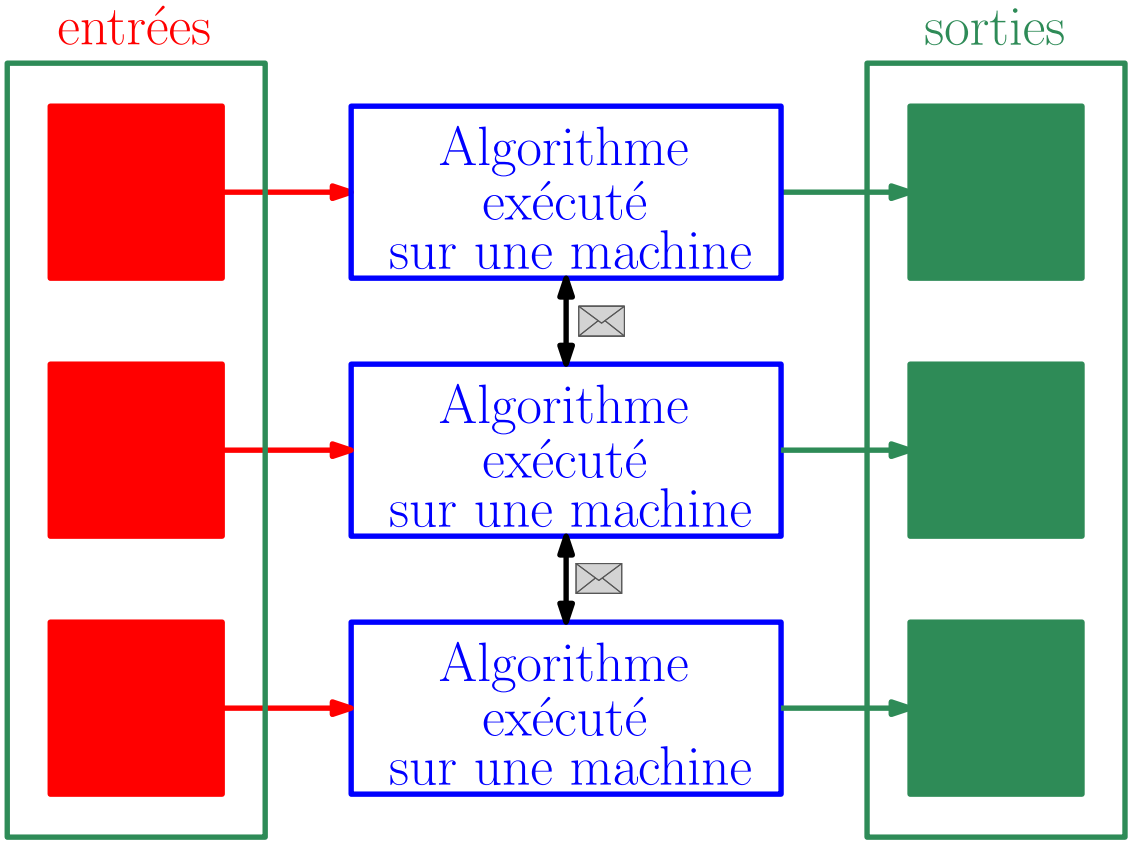
Modèle distribué

Algorithmique “classique”



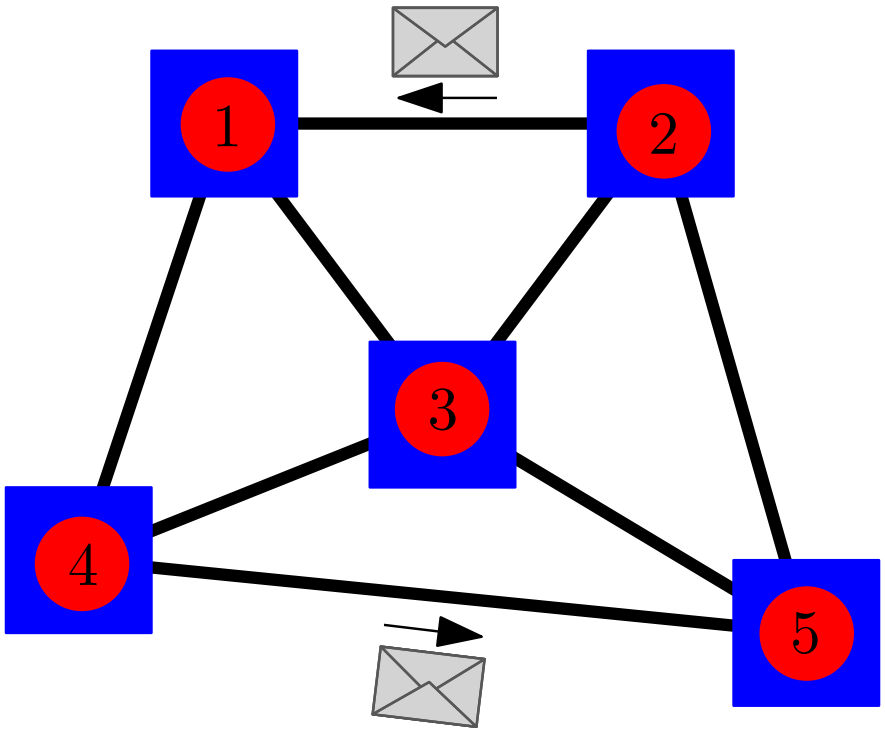
- Un processus qui exécute un algorithme
- Une donnée en entrée
- Une donnée en sortie
- Complexité : nombre d’opérations/instructions et/ou mémoire

Algorithmique distribuée



- n données en entrée et en sortie
- n processus qui exécutent le même algorithme et peuvent envoyer des messages à leurs voisins
- complexité : nombre de messages (calculs locaux ignorés)

Réseau de communication



Donnée

Processus

Chaque processus connaît l'information locale (pas d'information globale). Ils doivent communiquer pour produire leurs sorties.

Communication par passage de message

Primitives

- `envoyer(v)` : envoyer un message via le réseau
- `m = recevoir()` : attendre la réception d'un message

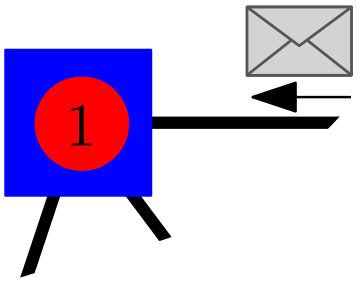
Remarques

- la topologie (graphe de communication entre les nœuds) est fréquemment *complète*.
- variantes suivant les hypothèses (connaissance du voisin ayant envoyé le message ou pas, ...)

Difficultés du distribué

Chaque processus n'a qu'une vision locale du réseau.

$\Rightarrow$ il est souvent impossible ou difficile de stocker la carte complète du réseau sur un seul nœud (exemple internet)



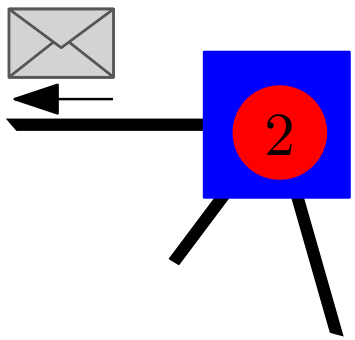
Donnée

Processus

Difficultés du distribué

Chaque processus n'a qu'une vision locale du réseau.

$\Rightarrow$ il est souvent impossible ou difficile de stocker la carte du réseau sur un seul nœud (exemple internet)



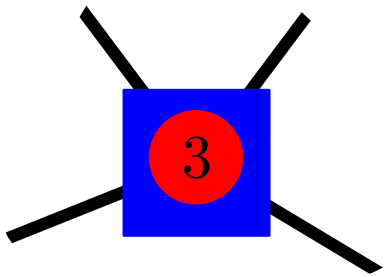
Donnée

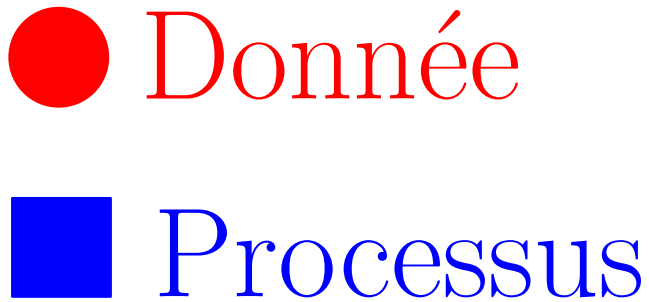
Processus

Difficultés du distribué

Chaque processus n'a qu'une vision locale du réseau.

$\Rightarrow$ il est souvent impossible ou difficile de stocker la carte du réseau sur un seul nœud (exemple internet)

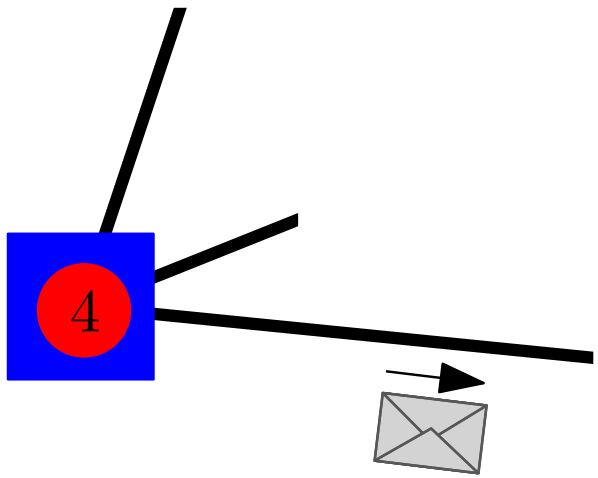




Difficultés du distribué

Chaque processus n'a qu'une vision locale du réseau.

$\Rightarrow$ il est souvent impossible ou difficile de stocker la carte du réseau sur un seul nœud (exemple internet)



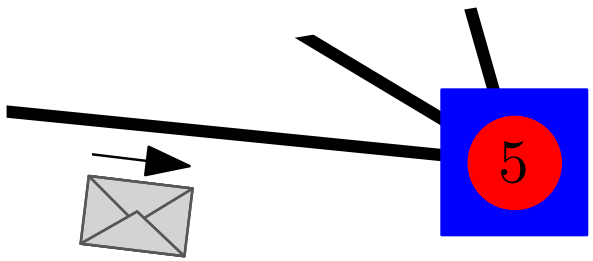
Donnée

Processus

Difficultés du distribué

Chaque processus n'a qu'une vision locale du réseau.

$\Rightarrow$ il est souvent impossible ou difficile de stocker la carte du réseau sur un seul nœud (exemple internet)



Donnée

Processus

Pannes et défaillances

Dans un système distribué avec énormément de nœuds, il est probable :

- qu'il y ait un ou plusieurs nœuds en panne à tout moment (crash, lenteur, ...) ;
- qu'il y ait un ou plusieurs liens de communication en panne (coupure, lenteur, ...).

$\Rightarrow$ La plupart des algorithmes distribués prennent en compte la possibilité de panne.

Dans certains cas (hypothèses de pannes), certains problèmes sont impossibles à résoudre.

Défaillances possibles

crash le processus s'arrête, c'est-à-dire il n'effectue plus aucune opération $\iff$ il n'est plus choisi par l'ordonnanceur.

messages perdus, *partionnement* les messages ne parviennent pas à destination (géré par TCP...) ou bien le réseau est partitionné

erreur byzantine le comportement du processus est arbitraire

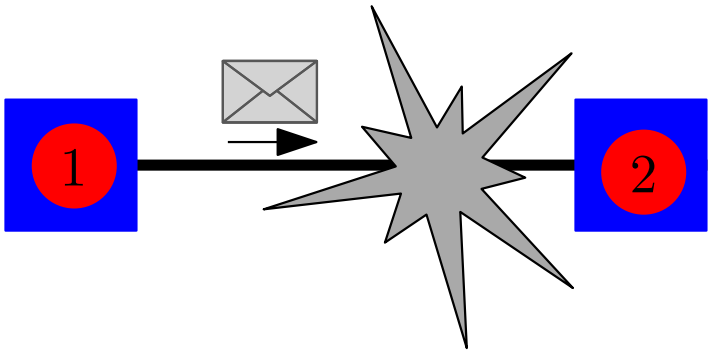
$\Rightarrow$ le nœud est compromis et contrôlé par un adversaire. On suppose généralement qu'il n'a pas la puissance de déchiffrer des messages chiffrés sans la clé privée.

honnête mais curieux suit l'algorithme, mais les données non chiffrées seront possiblement lues ou décodées

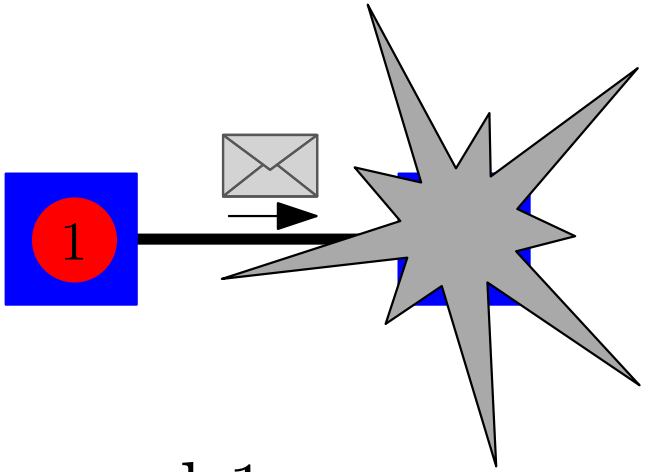
Difficultés des pannes

Une des grandes difficultés en algorithme distribuée est de savoir dans une situation donnée quelle panne a eu lieu.

message perdu



nœud crashé



dans les deux cas, le nœud 1
n'aura pas de réponse du nœud 2

Généralement impossible de distinguer entre :

- des messages perdus vers un nœud
- un nœud qui a crashé