

# ***Systemes Distribués***

***Licence Informatique 3<sup>ème</sup> année***

## ***Introduction générale***

Eric Cariou

*Université de Pau et des Pays de l'Adour  
UFR Sciences Pau – Département Informatique*

Eric.Cariou@univ-pau.fr

# *Plan général du cours*

- ◆ Introduction aux systèmes distribués
  - ◆ Problématique, concepts généraux
- ◆ Partie « programmation des systèmes distribués »
  - ◆ Techniques de communication à distance
    - ◆ Sockets TCP/UDP (en C et Java)
    - ◆ Middleware : Java RMI
  - ◆ En complément : programmation Java
    - ◆ Entrées/sorties (flux) et concurrence en Java (threads)
- ◆ Partie « introduction à l'algorithmique distribuée »
  - ◆ Gestion du temps et d'états distribués

# *Systemes distribués*

- ◆ Système distribué en opposition à système centralisé
- ◆ Système centralisé : tout est localisé sur la même machine et accessible par le programme
  - ◆ Système logiciel s'exécutant sur une seule machine
  - ◆ Accédant localement aux ressources nécessaires (données, code, périphériques, mémoire ...)
- ◆ Système distribué : une définition parmi d'autres (Andrew Tannenbaum)
  - ◆ Ensemble d'ordinateurs indépendants connectés en réseau et communiquant via ce réseau
  - ◆ Cet ensemble apparaît du point de vue de l'utilisateur comme une unique entité

# *Systemes distribués*

- ◆ Vision matérielle d'un système distribué : architecture matérielle
  - ◆ Machine multi-processeurs avec mémoire partagée, CPU multicore
  - ◆ Cluster d'ordinateurs dédiés au calcul/traitement massif parallèle
  - ◆ Ordinateurs standards connectés en réseau
- ◆ Vision logicielle d'un système distribué
  - ◆ Système logiciel composé de plusieurs entités logicielles s'exécutant indépendamment et en parallèle sur un ensemble d'ordinateurs connectés en réseau
- ◆ Dans ce cours
  - ◆ Conception logicielle des systèmes distribués
  - ◆ Par défaut sur une architecture matérielle de type ordinateurs connectés en réseau

# Systemes distribués

- ◆ Système distribué
  - ◆ Réalisation d'une certaine tâche globale par un ensemble d'entités logicielles distribuées
  - ◆ L'utilisateur a un point d'accès à ce système
  - ◆ L'utilisateur et/ou des entités n'ont pas besoin de connaître le détail de l'architecture du système (transparence)
  - ◆ « Un système distribué est un système qui m'empêche de travailler quand une machine dont je n'ai jamais entendu parler tombe en panne » *Leslie Lamport*
- ◆ Exemples de systèmes distribués
  - ◆ Serveur de fichier
  - ◆ Web
  - ◆ Calculs distribués

# *Exemples de systèmes distribués*

- ◆ Serveur de fichiers
  - ◆ Accès aux fichiers de l'utilisateur quelque soit la machine utilisée
  - ◆ Machines du département informatique
    - ◆ Clients : scinfeXXX
    - ◆ Un serveur de fichier
    - ◆ Sur toutes les machines : `/home/durand` est le « home directory » de l'utilisateur *durand*
    - ◆ Physiquement : fichiers se trouvent uniquement sur le serveur
    - ◆ Virtuellement : accès à ces fichiers à partir de n'importe quelle machine cliente en faisant « croire » que ces fichiers sont stockés localement
    - ◆ Arborescence de fichiers Unix : arborescence unique avec
      - ◆ Répertoires physiquement locaux
      - ◆ Répertoires distants montés via le protocole NFS (Network File System)

# *Exemples de systèmes distribués*

- ◆ Serveur de fichier (suite)
  - ◆ Intérêts
    - ◆ Accès aux fichiers à partir de n'importe quelle machine
    - ◆ Système de sauvegarde associé à ce serveur
    - ◆ Transparent pour l'utilisateur
  - ◆ Inconvénients
    - ◆ Si réseau ou le serveur plante : plus d'accès aux fichiers pour personne

# *Exemples de systèmes distribués*

- ◆ Autre exemple de système distribué : Web
  - ◆ Un serveur web auquel se connecte un nombre quelconque de navigateurs web (clients)
  - ◆ Accès à distance à de l'information
    - ◆ Accès simple
      - ◆ Serveur renvoie une page HTML statique qu'il stocke localement
    - ◆ Traitement plus complexe
      - ◆ Serveur interroge une base de données pour générer dynamiquement le contenu de la page
  - ◆ Transparent pour l'utilisateur : les informations s'affichent dans son navigateur quelque soit la façon dont le serveur les génère

# *Exemples de systèmes distribués*

## ◆ Calculs scientifiques

### ◆ Plusieurs architectures matérielles généralement utilisées

- ◆ Ensemble de machines identiques reliées entre elles par un réseau dédié et très rapide (cluster)
- ◆ Ensemble de machines hétérogènes connectées dans un réseau local ou bien encore par Internet (grille)

### ◆ Principe général

- ◆ Un (ou des) serveur distribue des calculs aux machines clients
- ◆ Un client exécute son calcul puis renvoie le résultat au serveur

### ◆ Avantage

- ◆ Utilisation d'un maximum de ressources de calcul

### ◆ Inconvénient

- ◆ Si réseau ou serveur plante, arrête le système

# *Intérêts des systèmes distribués*

- ◆ Utiliser et partager des ressources distantes
  - ◆ Système de fichiers : utiliser ses fichiers à partir de n'importe quelle machine
  - ◆ Imprimante : partagée entre toutes les machines
- ◆ Optimiser l'utilisation des ressources disponibles
  - ◆ Calculs scientifiques distribués sur un ensemble de machines
- ◆ Système plus robuste
  - ◆ Duplication pour fiabilité : deux serveurs de fichiers dupliqués, avec sauvegarde
  - ◆ Plusieurs éléments identiques pour résister à la montée en charge ...

# *Inconvénients/points faibles*

- ◆ Si problème au niveau du réseau
  - ◆ Le système marche mal ou plus du tout
- ◆ Bien souvent, un élément est central au fonctionnement du système : serveur
  - ◆ Si serveur plante : plus rien ne fonctionne
  - ◆ Goulet potentiel d'étranglement si débit d'information très important
- ◆ Sans élément central
  - ◆ Gestion du système totalement décentralisée et distribuée
  - ◆ Nécessite la mise en place d'algorithmes +/- complexes

# *Particularités des systèmes distribués*

- ◆ Système distribué = éclaté
  - ◆ Connaissance des éléments formant le système : besoin d'identification et de localisation
  - ◆ Gestion du déploiement et de la présence d'éléments essentiels
- ◆ Communication à distance est centrale
  - ◆ Techniques et protocoles de communication
  - ◆ Contraintes du réseau : fiabilité (perte de données) et temps de propagation (dépendant du type de réseau et de sa charge)
- ◆ Naturellement concurrent et parallèle
  - ◆ Chaque élément sur chaque machine est autonome
  - ◆ Besoin de synchronisation, coordination entre éléments distants et pour l'accès aux ressources (exclusion mutuelle ...)

# *Particularités des systèmes distribués*

## ◆ Hétérogénéité

- ◆ Des machines utilisées (puissance, architecture matérielle...)
- ◆ Des systèmes d'exploitation tournant sur ces machines
- ◆ Des langages de programmation des éléments logiciels formant le système
- ◆ Des réseaux utilisés : impact sur performances, débit, disponibilité ...
  - ◆ Réseau local rapide
  - ◆ Internet
  - ◆ Réseaux sans fil

# Particularités des systèmes distribués

- ◆ Exemple hétérogénéité des données : codage des entiers
  - ◆ Entier sur 32 bits (4 octets)
    - ◆ Ex :  $(010AD3F2)_{16} = 17486834$
  - ◆ A partir de l'adresse de l'entier en mémoire, les 4 octets ne sont pas toujours placés dans le même ordre
    - ◆ *Big Endian* (ex : x86) : octet de poids fort d'abord  
| 01 | 0A | D3 | F2 |
    - ◆ *Little Endian* (ex : SPARC) : octet de poids faible d'abord  
| F2 | D3 | 0A | 01 |
    - ◆ Si un ordinateur à processeur x86 envoie un entier à un ordinateur à processeur SPARC, les nombres seront différents
    - ◆  $(010AD3F2)_{16}$  est interprété comme  $(F2D30A01)_{16}$  soit la valeur (en non signé) de 4073916929

# Particularités des systèmes distribués

- ◆ Exemple hétérogénéité des données : codage des chaînes de caractères
  - ◆ Principe courant : un tableau de char (octet) avec information sur la taille ou la fin de chaîne
    - ◆ En C : code ASCII de valeur 0 pour marquer la fin de la chaîne  
`|B|o|n|j|o|u|r|\0|`
    - ◆ En Pascal : le premier caractère est un nombre (codé via un code ASCII) précisant la longueur de la chaîne  
`|\7|B|o|n|j|o|u|r|`
  - ◆ Les deux tableaux ont la même taille (8 octets) mais problème en cas d'échange
    - ◆ De C vers Pascal : le code ASCII du premier caractère 'B' est considéré comme la taille de la chaîne soit 66 caractères au lieu de 7
    - ◆ De Pascal vers C : le premier caractère '\7' sera considéré comme faisant partie de la chaîne (ici le caractère de contrôle BELL) et la chaîne se terminera jusqu'à qu'un '\0' soit trouvé
    - ◆ Dans les deux cas, on considérera qu'une zone mémoire débordant du tableau de la chaîne en fait partie

# *Fiabilité des systèmes distribués*

- ◆ Nombreux points de pannes ou de problèmes potentiels
  - ◆ Réseau
    - ◆ Une partie du réseau peut-être inaccessible
    - ◆ Les temps de communication peuvent varier considérablement selon la charge du réseau
    - ◆ Le réseau peut perdre des données transmises
  - ◆ Machine
    - ◆ Une ou plusieurs machines peut planter, engendrant une paralysie partielle ou totale du système
- ◆ Peut augmenter la fiabilité par redondance, duplication de certains éléments
  - ◆ Mais rend plus complexe la gestion du système
- ◆ Tolérance aux fautes
  - ◆ Capacité d'un système à gérer et résister à un ensemble de problèmes

# *Sécurité des systèmes distribués*

- ◆ Nature d'un système distribué fait qu'il est beaucoup plus sujet à des attaques
  - ◆ Communication à travers le réseau peuvent être interceptées
  - ◆ On ne connaît pas toujours bien un élément distant avec qui on communique
- ◆ Solutions
  - ◆ Connexion sécurisée par authentification avec les éléments distants
  - ◆ Cryptage des messages circulant sur le réseau

# *Transparences*

## ◆ Transparence

- ◆ Fait pour une fonctionnalité, un élément d'être invisible ou caché à l'utilisateur ou un autre élément formant le système distribué
  - ◆ Devrait plutôt parler d'opacité dans certains cas ...
- ◆ But est de cacher l'architecture, le fonctionnement de l'application ou du système distribué pour apparaître à l'utilisateur comme une application unique cohérente
- ◆ L'ISO définit plusieurs transparences (norme RM-ODP)
  - ◆ Accès, localisation, concurrence, réplication, mobilité, panne, performance, échelle

# *Transparences*

- ◆ **Transparence d'accès**
  - ◆ Accès à des ressources distantes aussi facilement que localement
  - ◆ Accès aux données indépendamment de leur format de représentation
- ◆ **Transparence de localisation**
  - ◆ Accès aux éléments/ressources indépendamment de leur localisation
- ◆ **Transparence de concurrence**
  - ◆ Exécution possible de plusieurs processus en parallèle avec utilisation de ressources partagées
- ◆ **Transparence de réplication**
  - ◆ Possibilité de dupliquer certains éléments/ressources pour augmenter la fiabilité

# *Transparences*

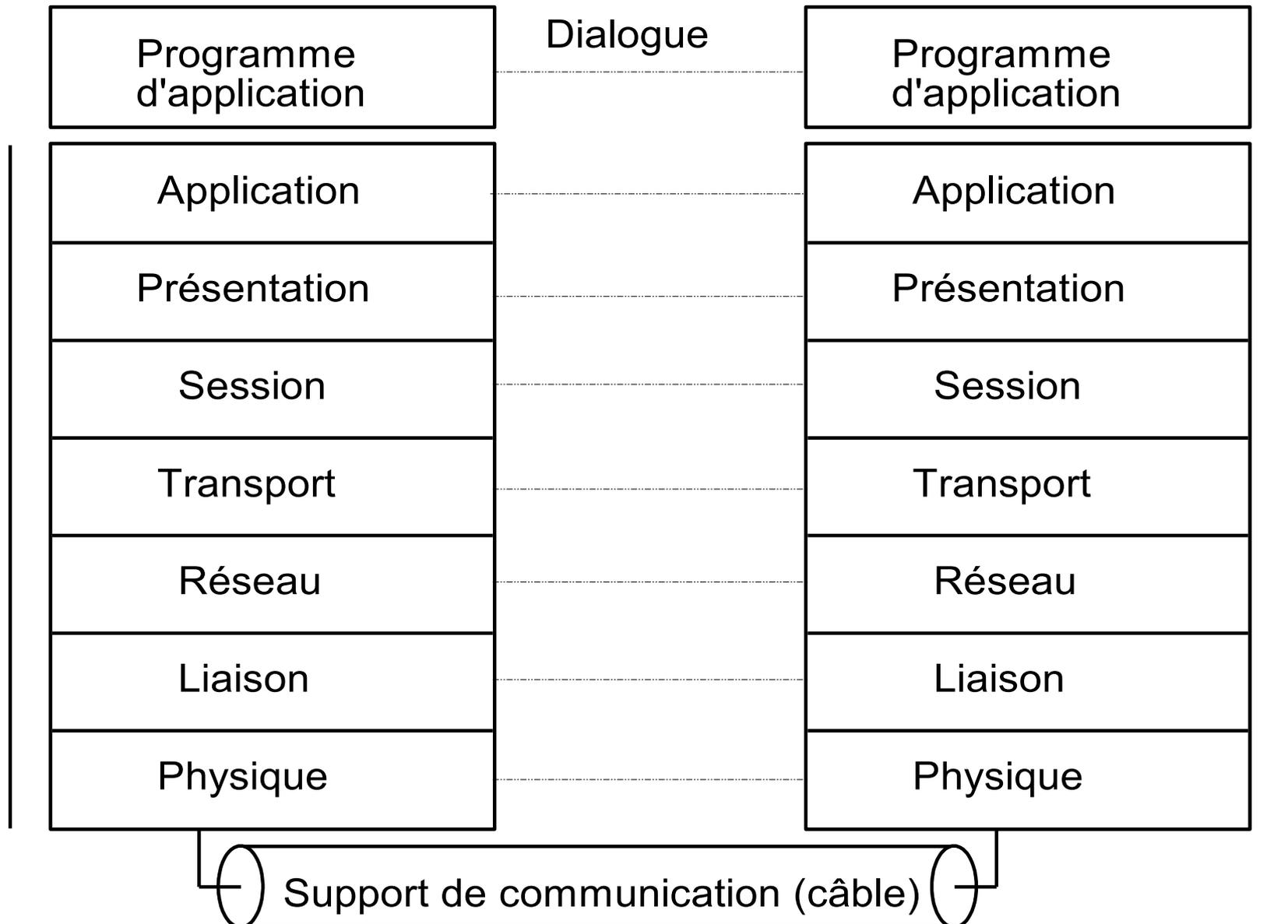
- ◆ Transparence de mobilité
  - ◆ Possibilité de déplacer des éléments/ressources
- ◆ Transparence de panne
  - ◆ Doit supporter qu'un ou plusieurs éléments tombe en panne
- ◆ Transparence de performance
  - ◆ Possibilité de reconfigurer le système pour en augmenter les performances
- ◆ Transparence d'échelle
  - ◆ Doit supporter l'augmentation de la taille du système (nombre d'éléments, de ressources ...)

# *Transparences*

- ◆ Un système donné va offrir un certain nombre de transparences
  - ◆ Souvent au minimum transparences de localisation, d'accès et de concurrence
- ◆ Système distribué ouvert
  - ◆ Peut être étendu en nombre d'éléments matériels le constituant
  - ◆ Possibilité d'ajouts de nouveaux services ou de ré-implémentation de services existants au niveau logiciel
  - ◆ Fonctionnement se base sur des interfaces d'interactions clairement définies

# Rappel sur les réseaux

- ◆ Norme OSI de l'ISO : architecture en 7 couches

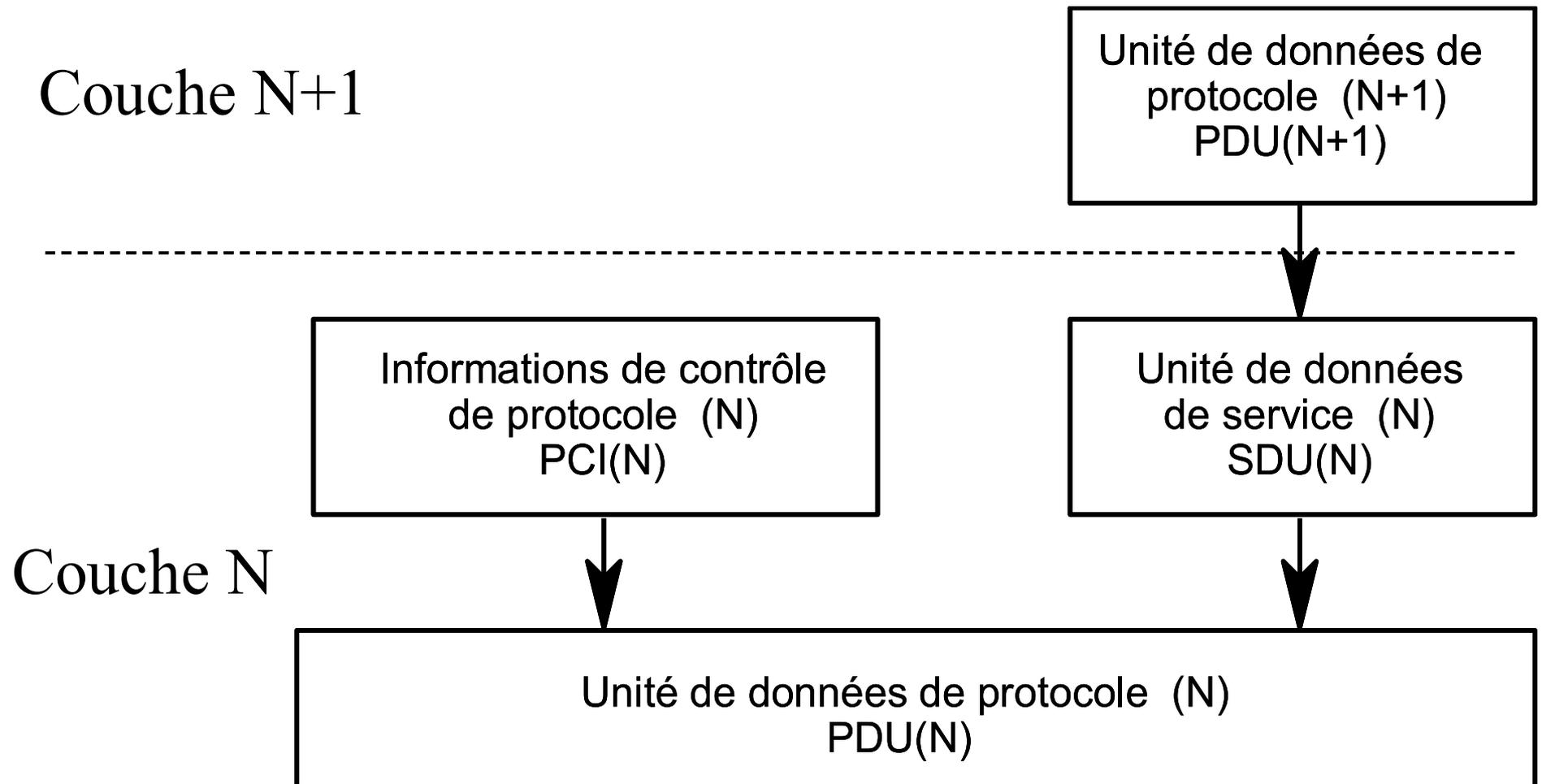


# *Rappel sur les réseaux*

- ◆ Modèle de communication en couche
  - ◆ Une couche à un rôle particulier
  - ◆ Une couche d'une entité communique avec une couche de même niveau d'une autre entité en respectant un certain protocole de communication
  - ◆ Pour communiquer avec une autre entité, une couche utilise les services de sa couche locale inférieure
  - ◆ Données échangées entre 2 couches : trames ou paquets
    - ◆ Données structurées
    - ◆ Taille bornée
    - ◆ Deux parties
      - ◆ Données de la couche supérieure à transmettre
      - ◆ Données de contrôle de la communication entre couches

# Rappel sur les réseaux

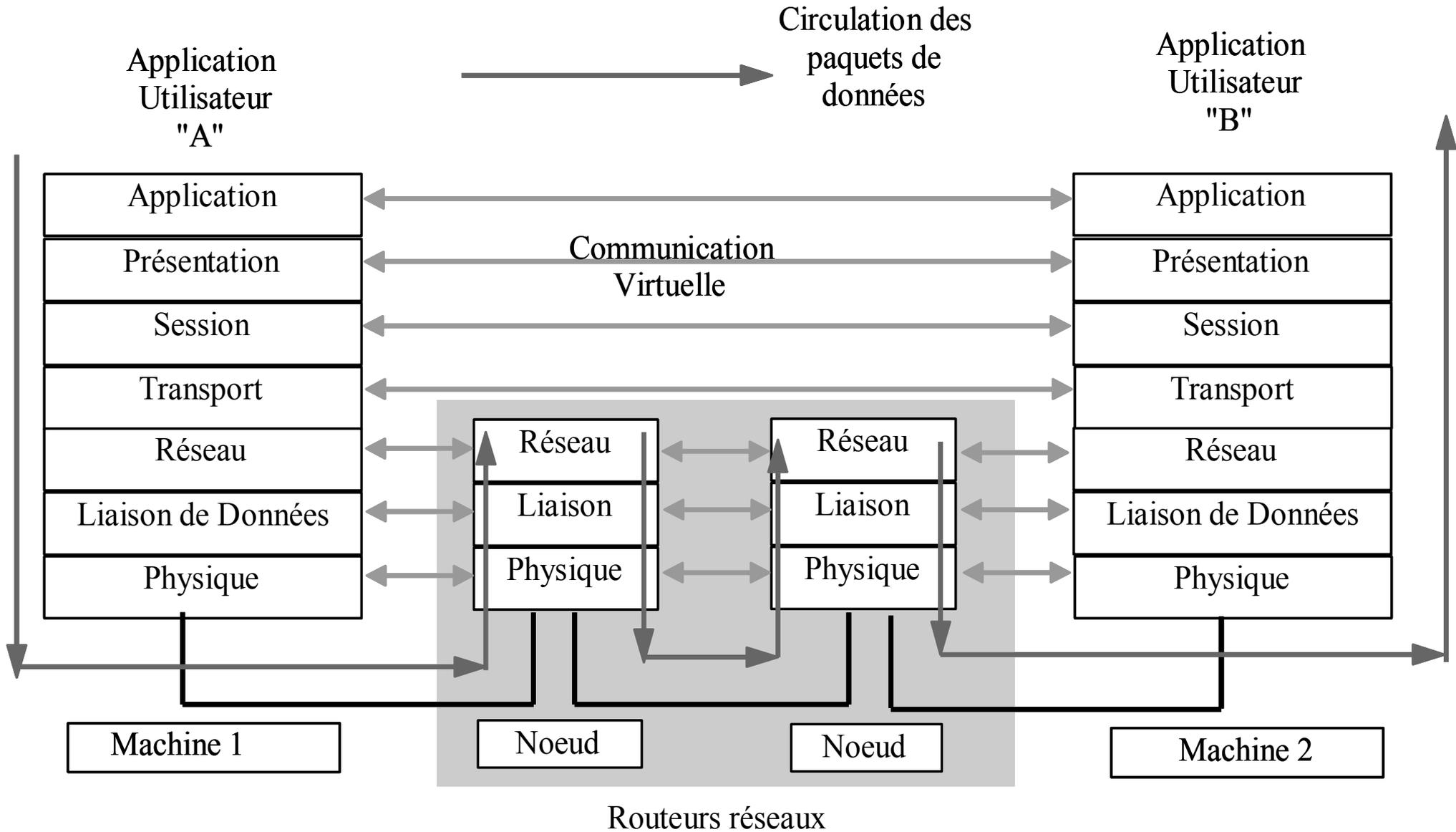
- ◆ Organisation des trames/paquets entre couches N et N+1



# *Rappel sur les réseaux*

- ◆ Norme OSI de l'ISO : architecture en 7 couches
  - ◆ Physique : transmission des données binaires sur un support physique
  - ◆ Liaison : gestion d'accès au support physique, assure que les données envoyées sur le support physique sont bien reçues par le destinataire
  - ◆ Réseau : transmission de données sur le réseau, trouve les routes à travers un réseau pour accéder à une machine distante
  - ◆ Transport : transmission (fiable) entre 2 applications
  - ◆ Session : synchronisation entre applications, reprises sur pannes
  - ◆ Présentation : indépendance des formats de représentation des données (entiers, chaînes de caractères...)
  - ◆ Application : protocoles applicatifs (HTTP, FTP, SMTP ...)

# Rappel sur les réseaux

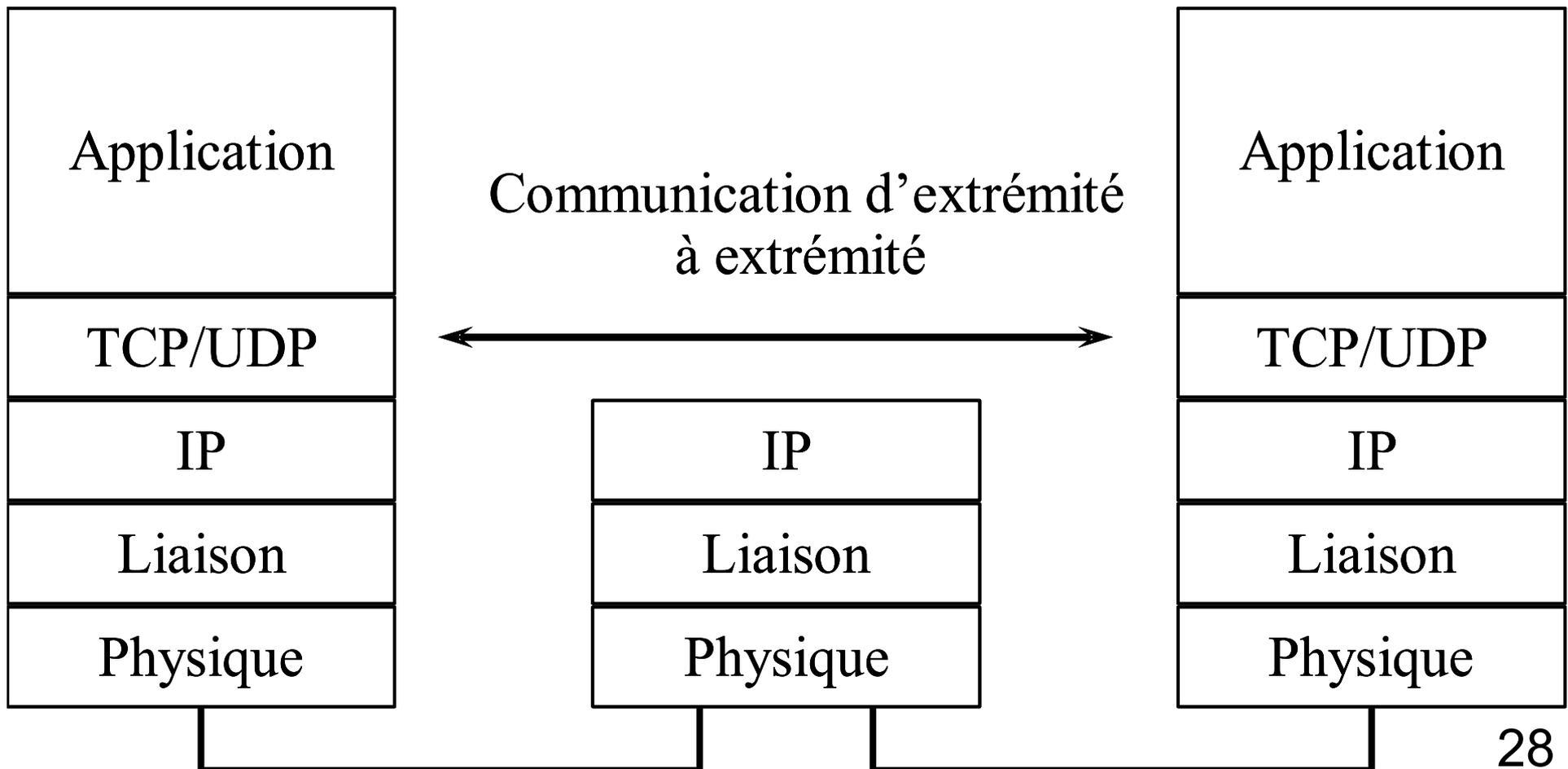


# *Rappel sur les réseaux*

- ◆ Réseaux TCP/IP
  - ◆ Réseaux locaux, internet ...
  - ◆ Couche réseau : IP (Internet Protocol)
    - ◆ Gestion des communications et connexions entre les machines à travers le réseau
    - ◆ Recherche des routes à travers le réseau pour accéder à une machine
  - ◆ Couche transport
    - ◆ TCP : connexion virtuelle directe et fiable entre 2 applications
    - ◆ UDP : mode datagramme
      - ◆ Envoi de paquets de données
      - ◆ Pas de gestion de l'ordre d'arrivée, pas de gestion des paquets perdus

# Rappel sur les réseaux

- ◆ TCP ou UDP
  - ◆ Communication entre systèmes aux extrémités
  - ◆ Pas de visibilité des systèmes intermédiaires

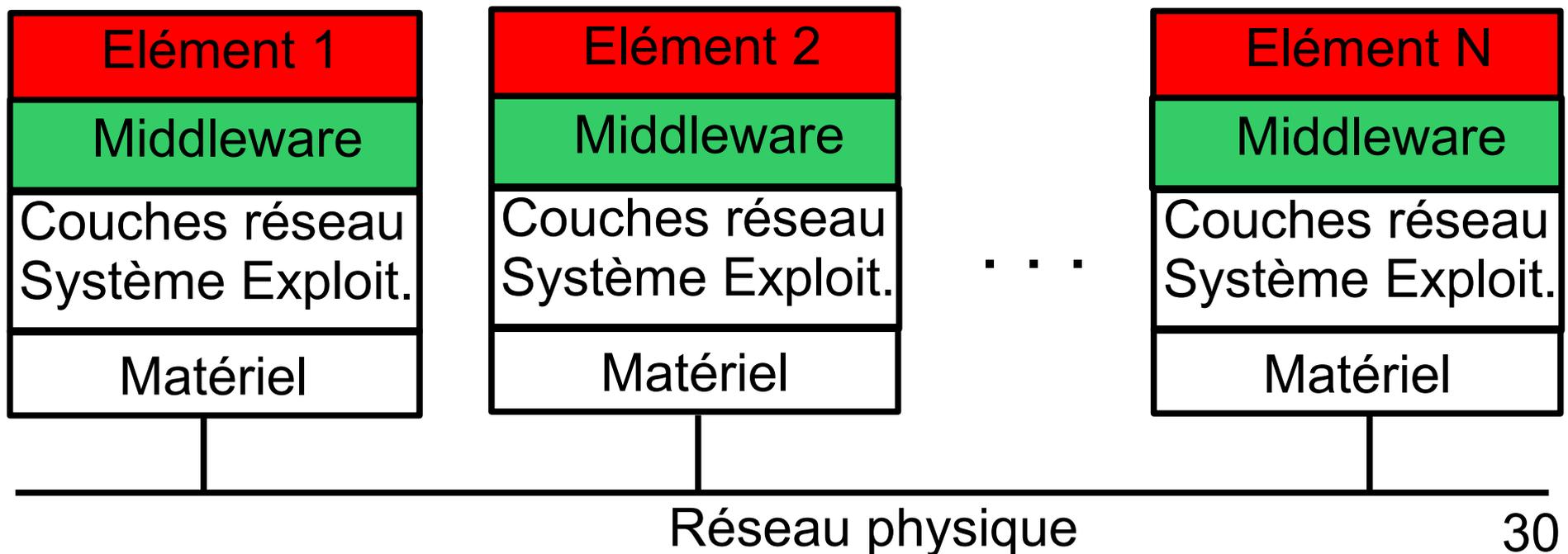


# *Communication*

- ◆ Système distribué
  - ◆ Ensemble d'entités logicielles communiquant entre-elles
- ◆ Entités logicielles s'exécutent sur des machines reliées entre elles par un réseau
- ◆ Communication entre entités logicielles
  - ◆ Le plus basique : directement en appelant les services des couches TCP ou UDP
  - ◆ Plus haut niveau : définition de couches offrant des services plus complexes
    - ◆ Couche réalisée en s'appuyant sur les couches TCP/UDP
    - ◆ Exemple de service : appel d'une procédure chez une entité distante
    - ◆ Notion de middleware (intergiciel)

# Communication

- ◆ Middleware ou intergiciel : couche logiciel
  - ◆ S'intercale entre le système d'exploitation/réseau et les éléments de l'application distribuée
  - ◆ Offre un ou plusieurs services de communication entre les éléments formant l'application ou le système distribué



# *Communication*

- ◆ But et fonctionnalités d'un middleware
  - ◆ Gestion de l'hétérogénéité
    - ◆ Langage de programmation, systèmes d'exploitation utilisés ...
  - ◆ Offrir des abstractions de communication de plus haut niveau
    - ◆ Appel d'une procédure à distance sur un élément
    - ◆ Communication via une mémoire partagée
    - ◆ Diffusion d'événements
    - ◆ ...
  - ◆ Offrir des services de configuration et de gestion du système
    - ◆ Service d'annuaire pour connaître les éléments présents
    - ◆ Services de persistance, de temps, de transaction, de sécurité ...

# *Communication*

- ◆ Protocole de communication
  - ◆ Ensemble de règles et de contraintes gérant une communication entre plusieurs entités
- ◆ But du protocole
  - ◆ Se mettre d'accord sur la façon de communiquer pour bien se comprendre
  - ◆ S'assurer que les données envoyées sont bien reçues
- ◆ Plusieurs types d'informations circulent entre les entités
  - ◆ Les données à échanger
  - ◆ Les données de contrôle et de gestion du protocole

# *Communication*

- ◆ Exemple basique de protocole
  - ◆ Une entité envoie des données à une deuxième entité
  - ◆ La deuxième entité envoie un acquittement pour prévenir qu'elle a bien reçue les données
  - ◆ Mais si utilise un réseau non fiable ou aux temps de transmission non bornés
    - ◆ Comment gérer la perte d'un paquet de données ?
    - ◆ Comment gérer la perte d'un acquittement ?
    - ◆ Comment gérer qu'un message peut arriver avant un autre alors qu'il a été émis après ?

# *Modèles d'interaction*

- ◆ Les éléments distribués interagissent, communiquent entre eux selon plusieurs modèles possibles
  - ◆ Client/serveur
  - ◆ Diffusion de messages
  - ◆ Mémoire partagée
  - ◆ Pair à pair
  - ◆ ...
- ◆ Abstraction/primitive de communication basique
  - ◆ Envoi de message d'un élément vers un autre élément
  - ◆ A partir d'envois de messages, peut construire les protocoles de communication correspondant à un modèle d'interaction

# *Modèles d'interaction*

- ◆ Rôle des messages
  - ◆ Données échangées entre les éléments
    - ◆ Demande de requête
    - ◆ Résultat d'une requête
    - ◆ Donnée de toute nature
    - ◆ ...
  - ◆ Gestion, contrôle des protocoles
    - ◆ Acquiescement : message bien reçu
    - ◆ Synchronisation, coordination ...

# *Modèle client/serveur*

- ◆ Deux rôles distincts
  - ◆ Client : demande que des requêtes ou des services lui soient rendus
  - ◆ Serveur : répond aux requêtes des clients
- ◆ Interaction
  - ◆ Message du client vers le serveur pour faire une requête
  - ◆ Exécution d'un traitement par le serveur pour répondre à la requête
  - ◆ Message du serveur vers le client avec le résultat de la requête
- ◆ Exemple : serveur Web
  - ◆ Client : navigateur Web de l'utilisateur
  - ◆ Requêtes : récupérer le contenu d'une page HTML gérée ou générée par le serveur

# *Modèle client/serveur*

- ◆ Modèle le plus répandu
  - ◆ Fonctionnement simple
  - ◆ Abstraction de l'appel d'un service : proche de l'appel d'une opération sur un élément logiciel
    - ◆ Interaction de base en programmation
- ◆ Particularités du modèle
  - ◆ Liens forts entre le client et le serveur
  - ◆ Un client peut aussi jouer le rôle de serveur (et vice-versa) dans une autre interaction
  - ◆ Nécessité généralement pour le client de connaître précisément le serveur (sa localisation)
    - ◆ Ex : URL du site Web
  - ◆ Interaction de type « 1 vers 1 »
    - ◆ 1 client communique avec 1 serveur à un moment donné

# *Diffusion de messages*

## ◆ Deux rôles distincts

- ◆ Émetteur : envoie des messages (ou événements) à destination de plusieurs récepteurs
  - ◆ Diffusion (broadcast) : à tous ceux qui sont présents
  - ◆ A un sous-ensemble de récepteurs : multicast
- ◆ Récepteurs : reçoivent les messages envoyés
- ◆ Peut être à la fois émetteur et récepteur

## ◆ Interaction

- ◆ Émetteur envoie un message
- ◆ Le middleware s'occupe de transmettre ce message à chaque récepteur

# *Diffusion de messages*

- ◆ Deux modes de réception
  - ◆ Le récepteur va vérifier lui-même qu'il a reçu un message (pull)
    - ◆ Boîte aux lettres
  - ◆ Le récepteur est prévenu que le message est disponible et il lui est transmis (push)
    - ◆ Le facteur sonne à la porte pour remettre en main propre le courrier
- ◆ Particularités du modèle
  - ◆ Dépendance plus faible entre les participants
    - ◆ Pas besoin pour l'émetteur d'être directement connecté aux récepteurs ni même de savoir combien ils sont
  - ◆ Interaction de type « 1 vers N »

# *Mémoire partagée*

- ◆ Les éléments communiquent via une mémoire partagée à l'aide d'une interface d'accès à la mémoire
  - ◆ Ajout d'une donnée à la mémoire
  - ◆ Lecture d'une donnée dans la mémoire
  - ◆ Retrait d'une donnée de la mémoire
- ◆ Le middleware gère l'accès à la mémoire pour chacun des participants
- ◆ Particularité du modèle
  - ◆ Aucun lien, aucune interaction directe entre les participants

# *Mémoire partagée*

- ◆ Complexité du modèle : dans la gestion de la mémoire
  - ◆ On est dans un système distribué
  - ◆ Comment gérer une mémoire dans ce contexte ?
  - ◆ Plusieurs solutions
    - ◆ Déployer toute la mémoire sur un seul site
      - ◆ Accès simple mais goulot potentiel d'étranglement et fiabilité faible
    - ◆ Éclater la mémoire sur plusieurs sites
      - ◆ Avec ou sans duplication des données
      - ◆ Il faut mettre en place des algorithmes +/- complexes de gestion de mémoire distribuée

# *Modèle pair à pair (peer to peer)*

- ◆ Un seul rôle : pas de distinction entre les participants
  - ◆ Chaque participant est connecté avec tous les participants d'un groupe et tout le monde effectue les mêmes types d'actions
  - ◆ Pour partager des données, effectuer un calcul commun ...
- ◆ Exemples
  - ◆ Modèles d'échanges de fichiers (bittorrent...)
    - ◆ Avec parfois un mode hybride client/serveur – P2P
    - ◆ Serveur sert à connaître qui possède un fichier ou faire des recherches
    - ◆ Le mode P2P est utilisé ensuite pour les transferts
      - ◆ Chacun envoie une partie du fichier à d'autres participants
  - ◆ Algorithme de consensus : choix d'une valeur parmi plusieurs
    - ◆ Chacun mesure une valeur (la même en théorie) puis l'envoie aux autres
    - ◆ Une fois reçues les valeurs de chacun, localement, chacun exécute le même algorithme sur ces valeurs pour élire la bonne valeur