Systèmes distribués et virtualisation de ressources

Tanguy RISSET
(Transparents : Antoine Fraboulet)
tanguy.risset@insa-lyon.fr

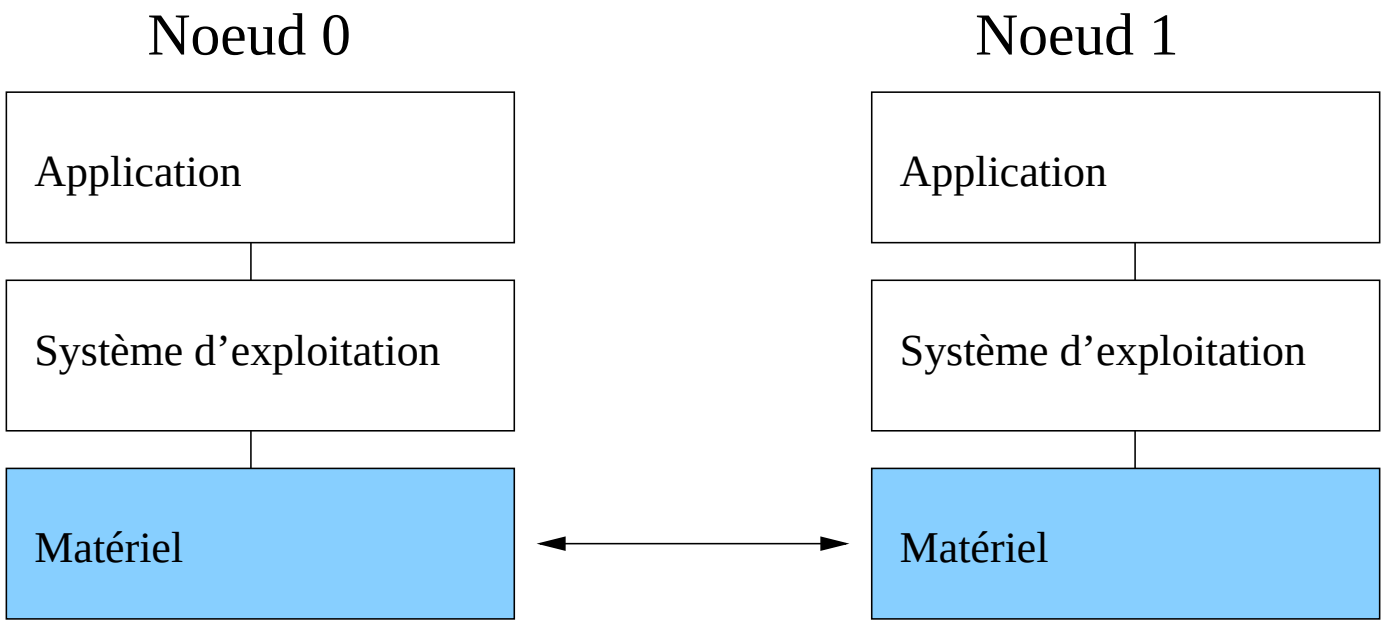
Plan

1 Distribution de ressources

1. Distribution de ressources
   - Distribution matérielle
   - Distribution au niveau du système d'exploitation
   - Distribution applicative
2. Virtualisation de données
   - Distribution au niveau des disques
   - Distribution au niveau du système d'exploitation
3. Étude de cas
   - Serveur de vidéo à la demande
4. Systèmes de fichiers distribués

Distribution de ressources

Distribution matérielle

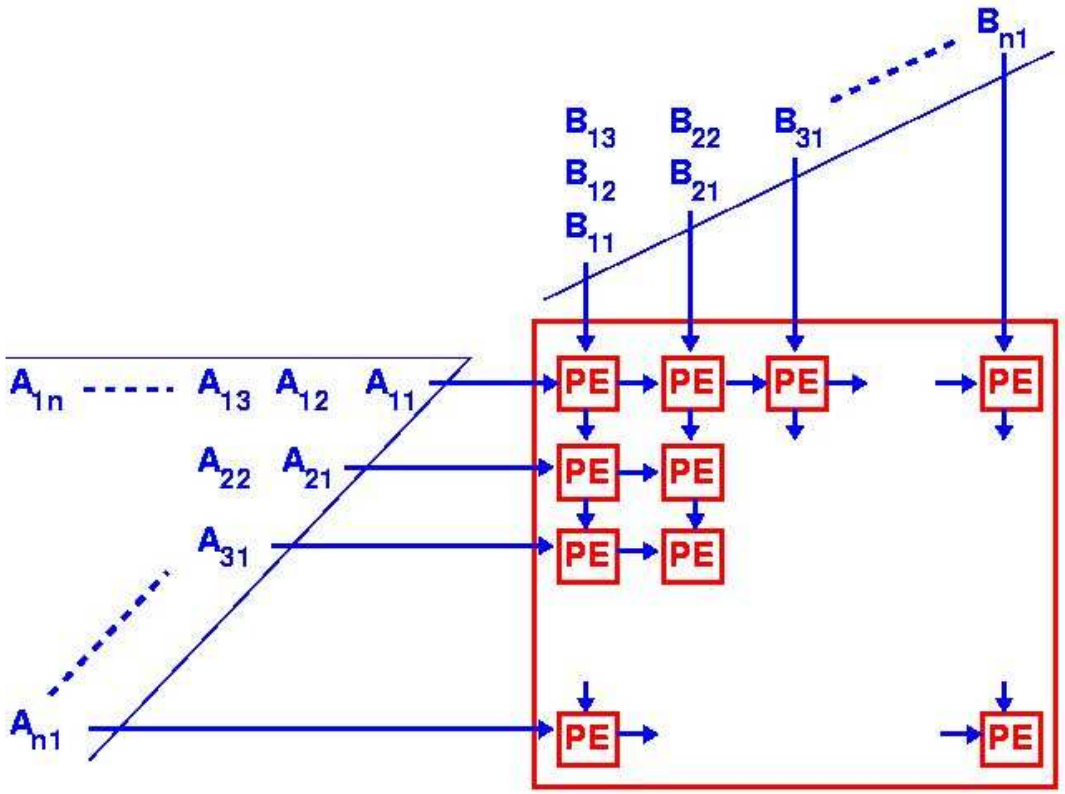


Motivations

- Répartition de charge
  - temps de calcul
    - simulation (météo, CAO, . . .)
    - programmation parallèle
  - taille des problèmes
    - mémoire partagée
    - communications entre machines
- Modularité
  - facilité d'intégration
    - construction par briques (modules)
  - développements idépendants
    - définitions d'interfaces

Super calculateurs

- Machines vectorielles : SIMD
  - une instruction est exécutée sur plusieurs données
  - réseaux d'unités de calcul



Super calculateurs (2)

- Machines parallèles : MIMD
  - réseaux de processeurs avec mémoire locale
  - Augmentation des ressources (mémoires et caches)
  - Topologies de communications complexes

Super calculateurs (3)

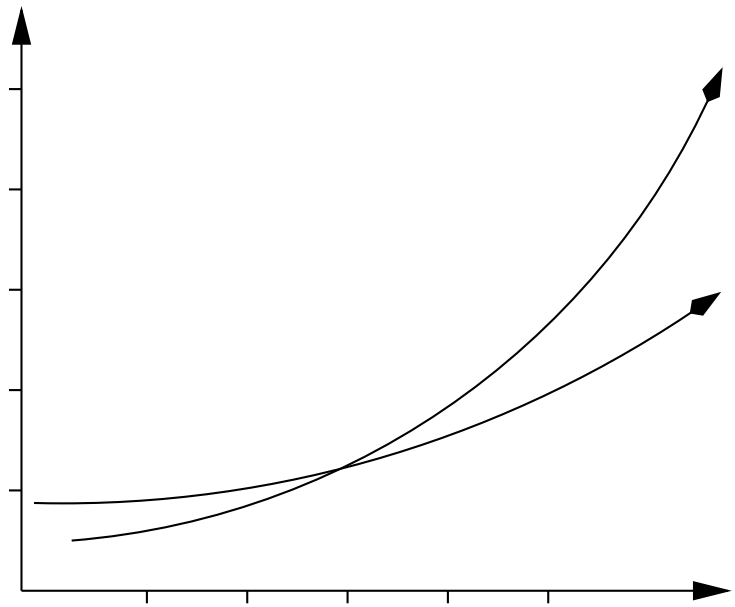
- Canaux de communications internes aux machines
- Topologies
  - point à point, anneaux, étoiles, tores, hypercubes

    $\Rightarrow$ routage
- Communications
  - commutation de circuits
  - commutation de paquets (store and forward, whormhole)
  - problèmes de débits et temps de latence

    $\Rightarrow$ protocoles

Intégration des réseaux dans les machines

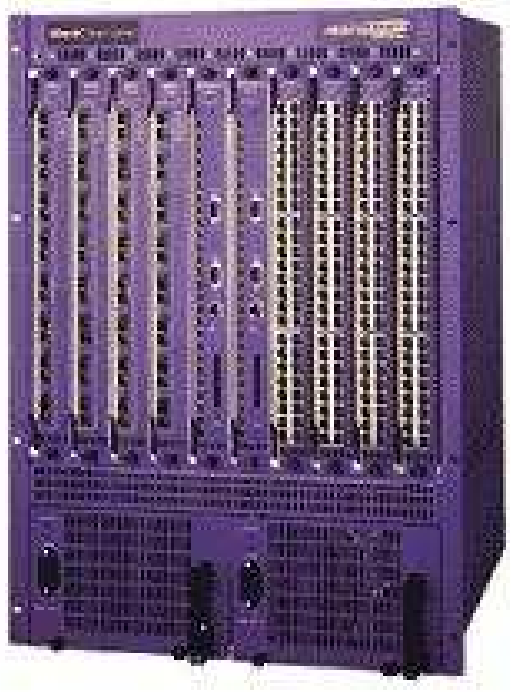
- Évolution des vitesses réseaux et machines



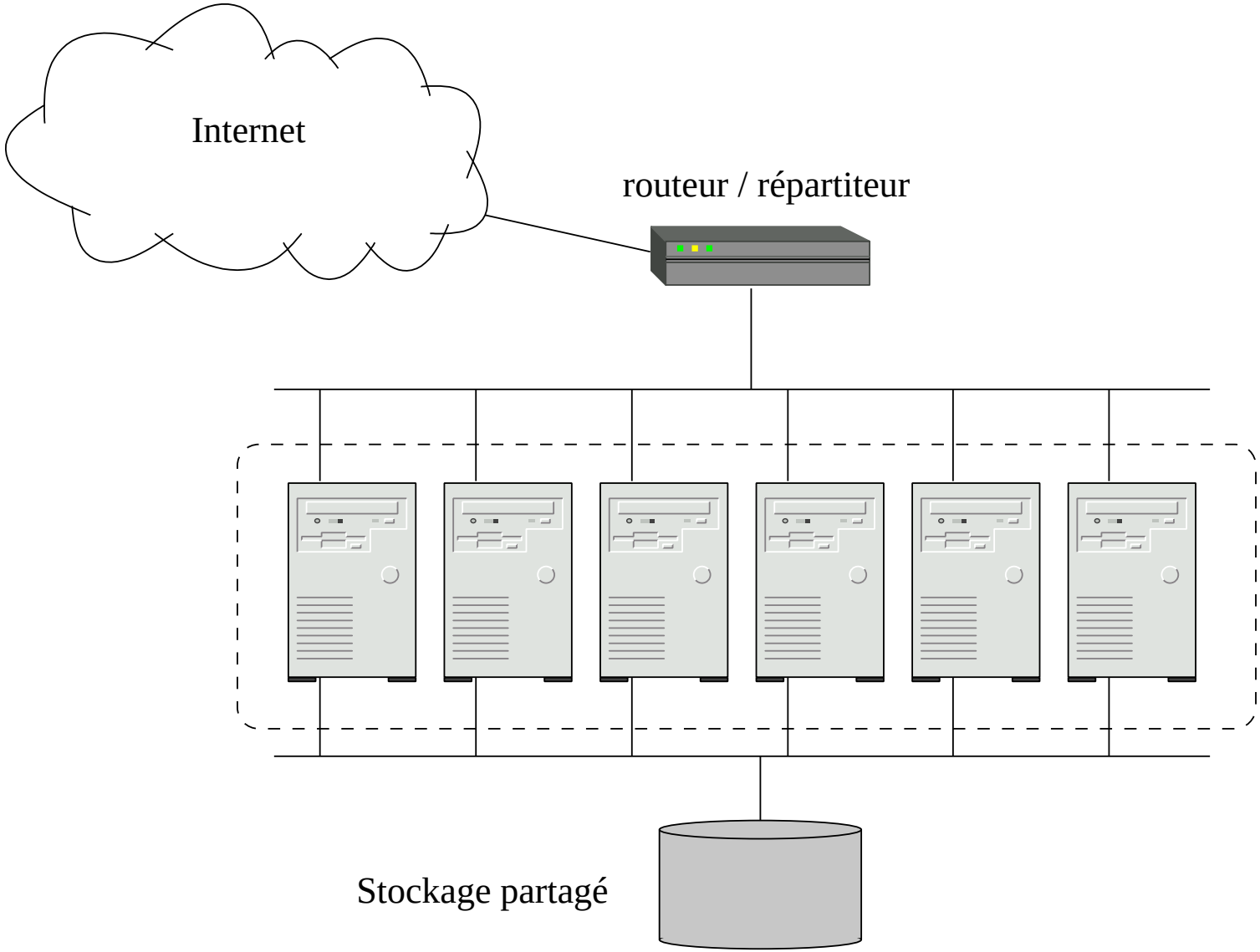
- Actuellement : point de croisement entre débits de l'accès mémoire et débits des réseaux (réseaux très haut débit).

Distribution matérielle

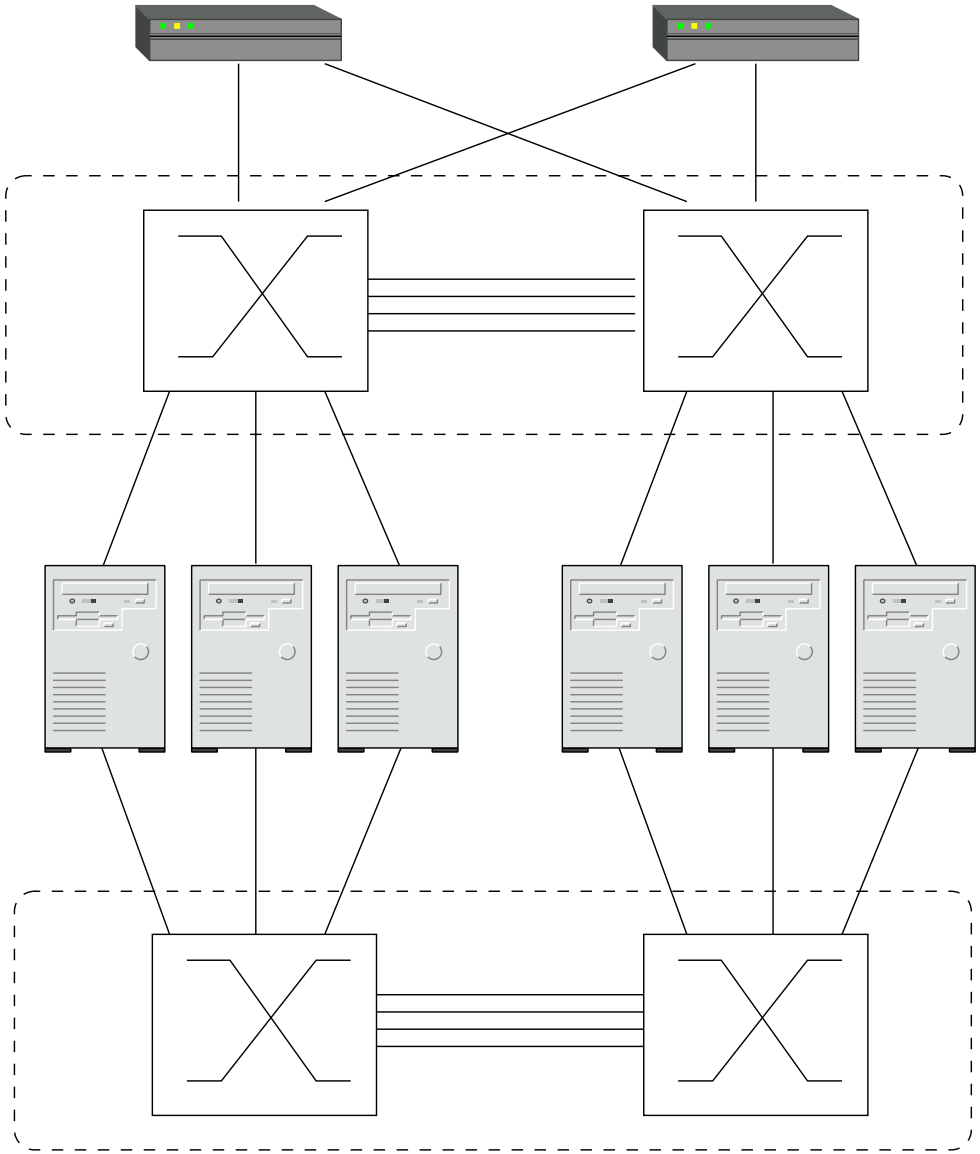
- Répartition de charge à l'arrivée des requêtes
- Matériel spécialisé
- Très haute performance
  - commutateurs niveau 3 (IP)
  - gestion de protocoles évolués (HTTP, FTP, ...)
  - processeurs et architectures dédiés



Service web



Serveur transactionnel



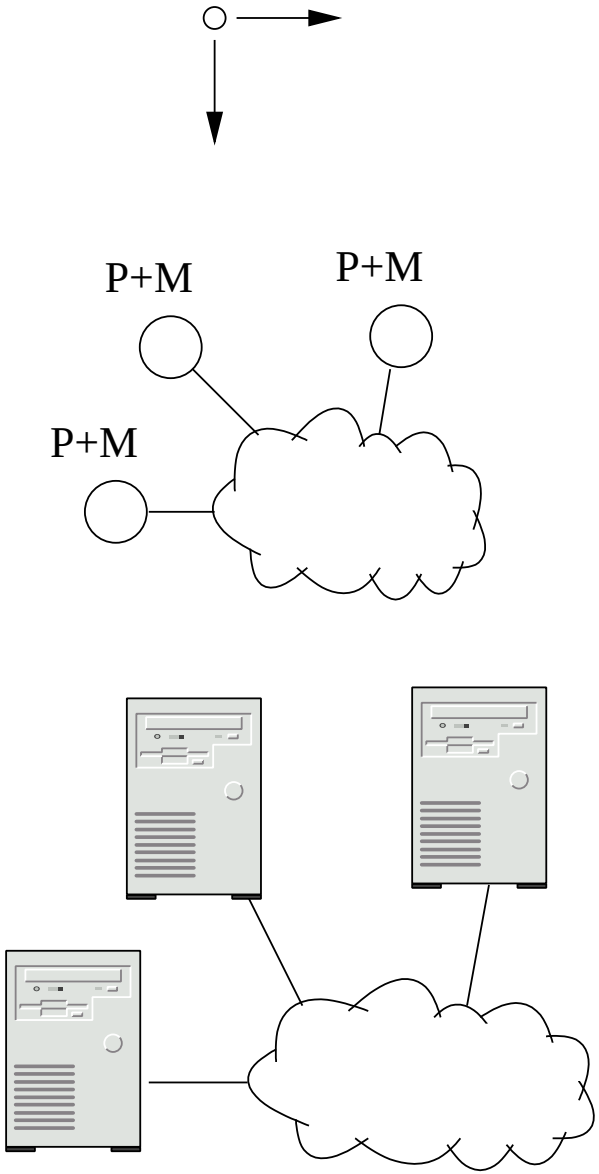
Routeurs WAN

Frontal réseaux
distribution des requêtes

Serveurs d'applications

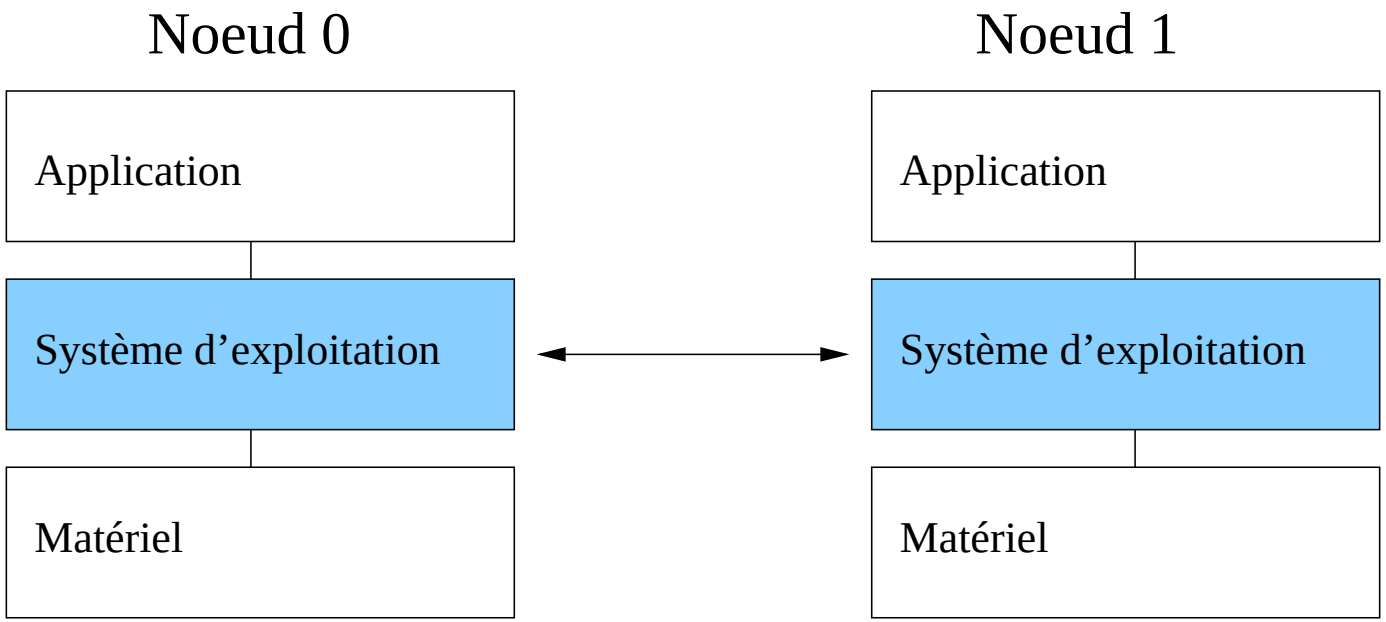
Répartition interne
Accès bases SQL

Répartition matérielle



Distribution de ressources

Distribution au niveau du système d'exploitation



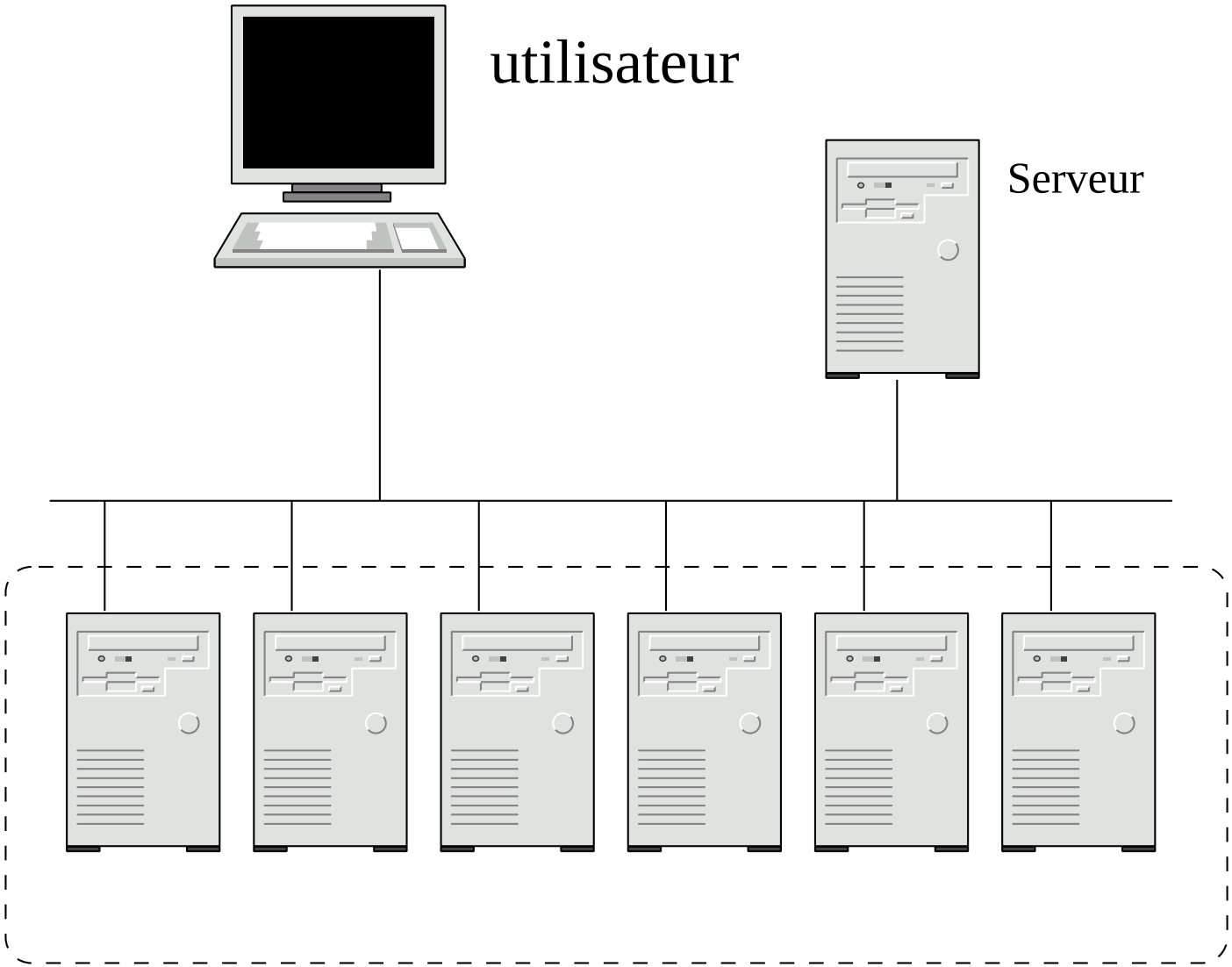
Fermes de machines

- But : réaliser une machine multi-processeurs à moindre coût
  - utilisation de machines standard
  - interconnexion par un réseau rapide
- Premiers VAXCluster en 1980 (DIGITAL)
- Fermes/Grappes/Clusters sont les mêmes choses

Fermes

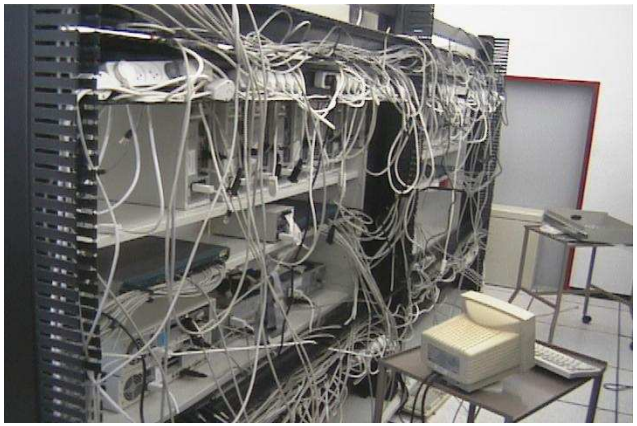
- Système parallèle à mémoire distribuée et à liens lâches (loosely coupled system)
  - Possibilité d'avoir des machines hétérogènes
  - Possibilité d'avoir machines SMP (tightly coupled system)
  - Possibilité d'avoir des réseaux rapides dédiés

Fermes



Clusters

- L'emballage peux varier
- L'intérieur est toujours un réseau !



Utilisation des fermes

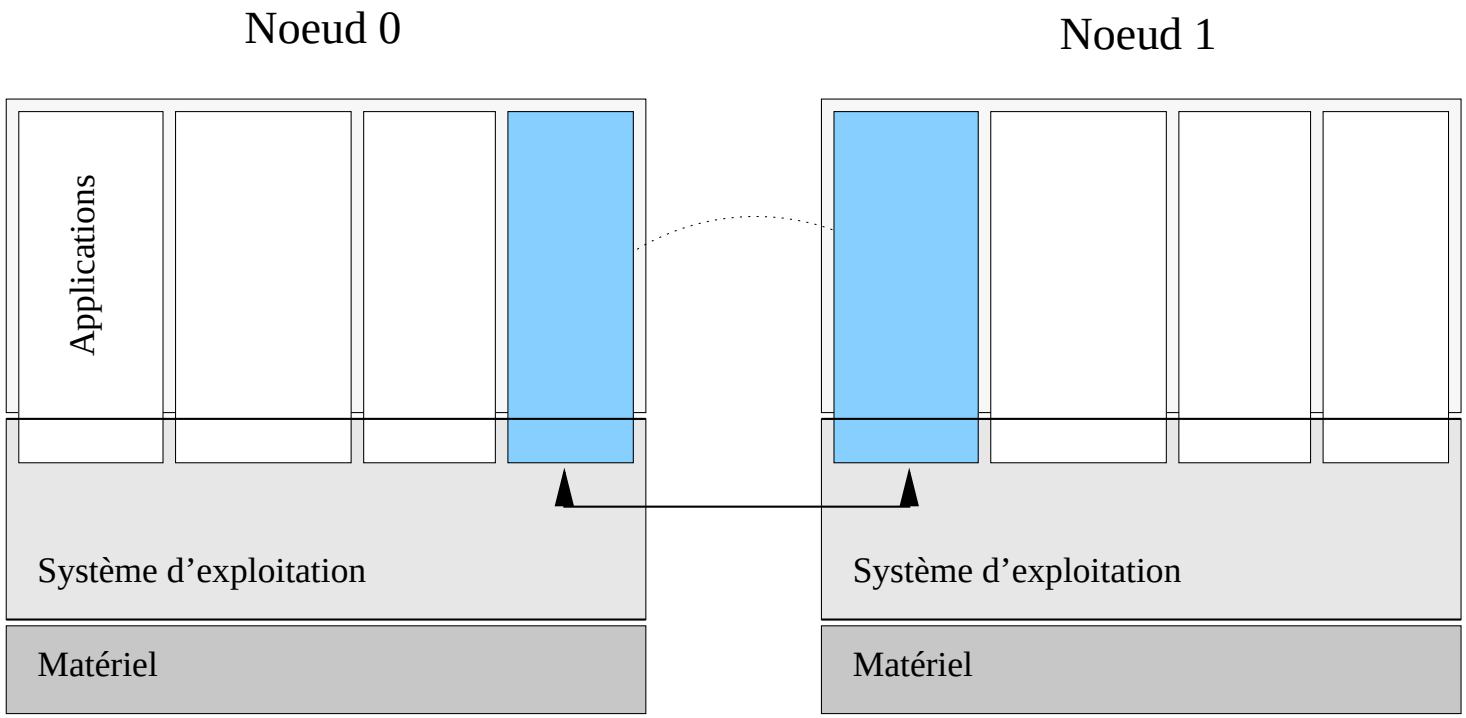
- Performance (nombre de requêtes)
- Disponibilité (incidents et pannes)
- Répartition de charge

- Augmentation du nombre de requêtes sur les serveurs
  - aucun serveur unique n’est capable de tenir la charge
  - un service centralisé est sensible aux pannes

Utilisation d'applications standards : Mosix

- Pas de modification des applications mais modification du système d'exploitation (FreeBSD ou Linux) (`www.mosix.org`)
- Utilisation d'un cluster comme une très grosse machine
- Répartition et synchronisation transparentes (SMP)
- Répartition de charge dynamique (migration automatique)

Mosix



- Migration transparente
  - mémoire
  - contexte
  - communications

Conclusion

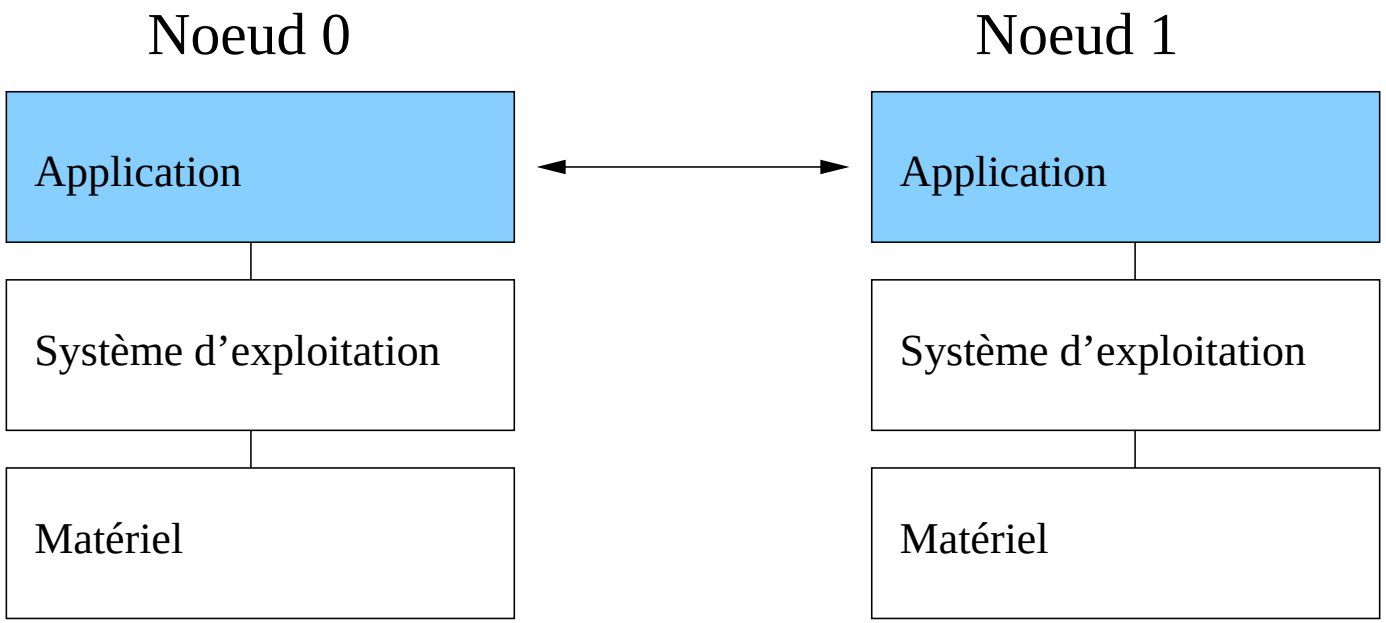
- Matériel très performant et modulaire
- Chaque type d'application a son type de cluster
- Mise au point et calibrage difficile

- Virtualisation des ressources
- Distribution la plus transparente possible

- Nécessite un système d'exploitation dédié.

Distribution de ressources

Distribution applicative



Passage de messages

- Les applications gèrent la communication de façon explicite
- Une bibliothèque fournit des primitives d'envois de message
- Modèle de communication
  - appels bloquants
- Abstraction de la topologie du réseau
  - Commuications collectives évoluées

Exemple : MPI

- Communauté de processus appelée `communicator`
- Le `communicator` par défaut est `MPI_COMM_WORLD`
- Chaque processus a un numéro appelé `rank` $[0, N-1]$
- Initialisation : `MPI_Init()`
- Fin de processus : `MPI_Finalize()`
- Un processus peut connaître son rang avec `MPI_Comm_rank()`
- La taille du `communicator` est donnée par `MPI_Comm_size()`
- Communications
  - `MPI_Send()` envois
  - `MPI_Recv()` réceptions
  - Les appels sont bloquants

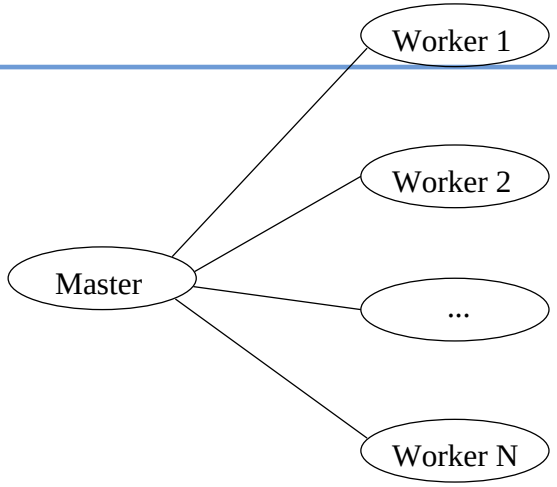
MPI (2)

- Un `tag` permet de filtrer les messages reçus
  - `MPI_ANY_TAG` prend tous les messages
- Nécessiter de connaître le type de données envoyées
  - `MPI_CHAR`
  - `MPI_INT`, `MPI_LONG` ...
- Code de retour détaillé des fonctions disponible

MPI (3)

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#define BUF_SIZE 80
int main(int argc, char* argv[])
{
  int id,i;
  int n_processes;
  char buffer[BUF_SIZE];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&id);
  MPI_Comm_size(MPI_COMM_WORLD,&n_processes);

  if (id==0) {
    for(i=1;i<n_processes;i++) {
      MPI_Recv(buffer,BUF_SIZE,MPI_CHAR,MPI_ANY_SOURCE,MPI_ANY_TAG,
               MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      printf("%s",buffer);
  } else {
      sprintf(buffer,"Hello, I'm process %d\n",id);
      MPI_Send(buffer,strlen(buffer)+1,MPI_CHAR,0,0,MPI_COMM_WORLD);
  }

  MPI_Finalize();
  return 0;
}
```

Exemple d'utilisation : mode maître / esclaves




```
while (1) {
   msg=get_msg_worker();
   if (msg.type == RESULT)
      save_result(msg.result);
   if (there_is_work()) {
      job = generate_new_job();
      send_msg_worker(job);
   } else {
      send_msg_worker(QUIT);
   }
   if (all_workers_done)
      terminate();
}
```

```
while (1) {
   send_master(REQUEST_WORK);
   msg = get_msg_master();
   if (msg.type == JOB) {
      result = do_work(msg.job);
      send_master(result);
   } else if (msg.type == QUIT) {
      terminate();
   }
}
```

MPI et les autres

MPI, PVM, BIP, RPC, . . .

- Bibliothèques de passages de messages
  - Programmation spécifique
  - Efficace, portable
  - Très dur à *bien* utiliser

- Besoin de développer des méthodes plus évoluées
  - abstraction des communications
  - intégrer au mieux la distribution dans le logiciel

Programmation et utilisation des systèmes distribués

- Programmation d'applications spécifiques
  - Bibliothèques de passages de messages bas niveau
    - MPI (Message Passing Interface)
  - Appel de procédure à distance
    - RPC (Remote Procedure Call)
    - RMI
- Utilisation d'applications standards
  - Mosix (modification du système d'exploitation)

Conclusion

- Solutions très performantes mises en avant
- Chaque type d'application a son type de cluster
- Mise au point et calibrage difficile

- Virtualisation des ressources
- Distribution la plus transparente possible

- $\Rightarrow$ Cohérence globale
- $\Rightarrow$ Tolérance aux pannes

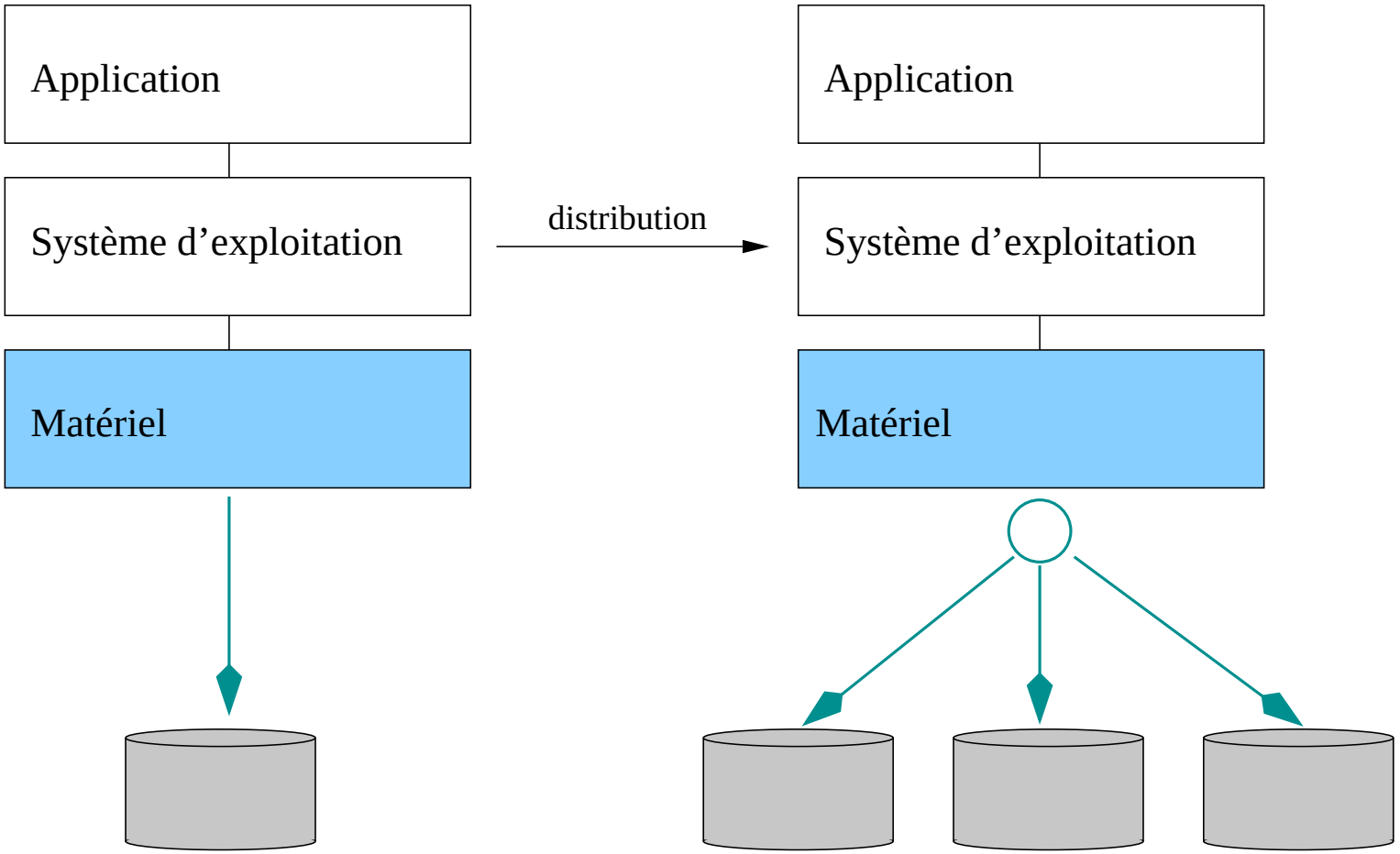
Plan

2 Virtualisation de données

1. Distribution de ressources
   - Distribution matérielle
   - Distribution au niveau du système d'exploitation
   - Distribution applicative
2. Virtualisation de données
   - Distribution au niveau des disques
   - Distribution au niveau du système d'exploitation
3. Étude de cas
   - Serveur de vidéo à la demande
4. Systèmes de fichiers distribués

Virtualisation du stockage

Distribution au niveau des disques



Distribution du stockage: pourquoi

- Augmentation exponentielle des performances
  - Puissance (Joy) :

$$\text{MIPS} = 2^{\text{année}-1984}$$

  - Densité (Moore) :

$$\text{Transistors par puce} = 2^{\text{année}-1964}$$

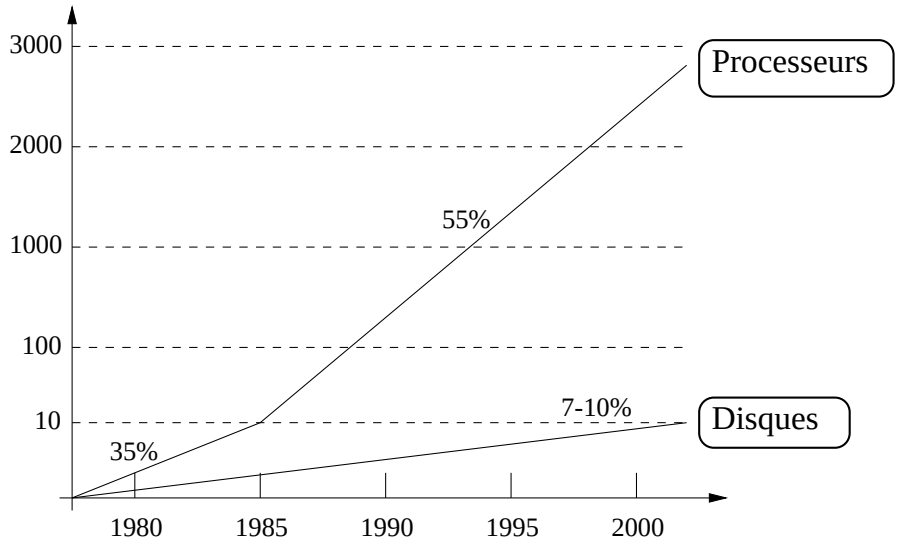
  - Densité des supports magnétiques «Maximal Areal Density» (Frank):

$$\text{MAD} = 10^{\frac{\text{année}-1971}{10}}$$

- “Stagnation” des performances pour la rapidité d’accès aux disques

Problèmes des disques

- Performances faibles
  - Limités par le temps de recherche
  - Limités par la vitesse de rotation des plateaux (débit)
  - Performance des disques : augmentation de 7% par an



- Perte des données en cas de panne

Loi d'Amdhal

- Pour un programme donné avec une technique permettant d'accelerer des portions du traitement par un facteur k:
  - Fraction du temps passé dans les calculs "accelérables": f
  - Accélération du temps de calcul pour les calculs accélerables: k
- Après accélération: $T_{New} = T_{Old} * (1 - f + f/k)$
- Accélération effective : "speedup" S

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

Soit pour une application avec $f = 0,9$ (10% du temps passé en E/S): si on a une accélération $k = 10$, le speedup réél S n'est que de 5 (pour $k = 100$ on a $S = 10$).

Augmenter les performances

- On peut utiliser des systèmes de cache logiciel mais il reste de nombreuses limitations
  - Les données sont volatiles en mémoire
  - Requêtes aléatoires de petites tailles (transactions)
  - Requêtes peu fréquentes de grandes tailles (simulation/modélisation)
  - Le problème des pannes n'est pas résolu
- Il faut mettre en place de nouvelle techniques pour assurer une évolution plus robuste.

Évaluation des systèmes de stockages

- Critère d'évaluation des systèmes de stockage:
  - Performances
  - sureté de fonctionnement (reliability)
  - Coût
- Patterson et al. montrent qu'il est plus intéressant d'utiliser plusieurs petits disques peu cher plutôt qu'un gros disque cher.

Extension des systèmes existants ?

- Augmenter le nombre de disque:
  - permet de répartir la bande passante
  - augmente la capacité à moindre frais
  - permet d'utiliser des disques du marché
- Mais cela fragilise aussi le système
  - Si on considère un temps moyen de bon fonctionnement constant pour un disque et un réseau de disques indépendants, on obtient:

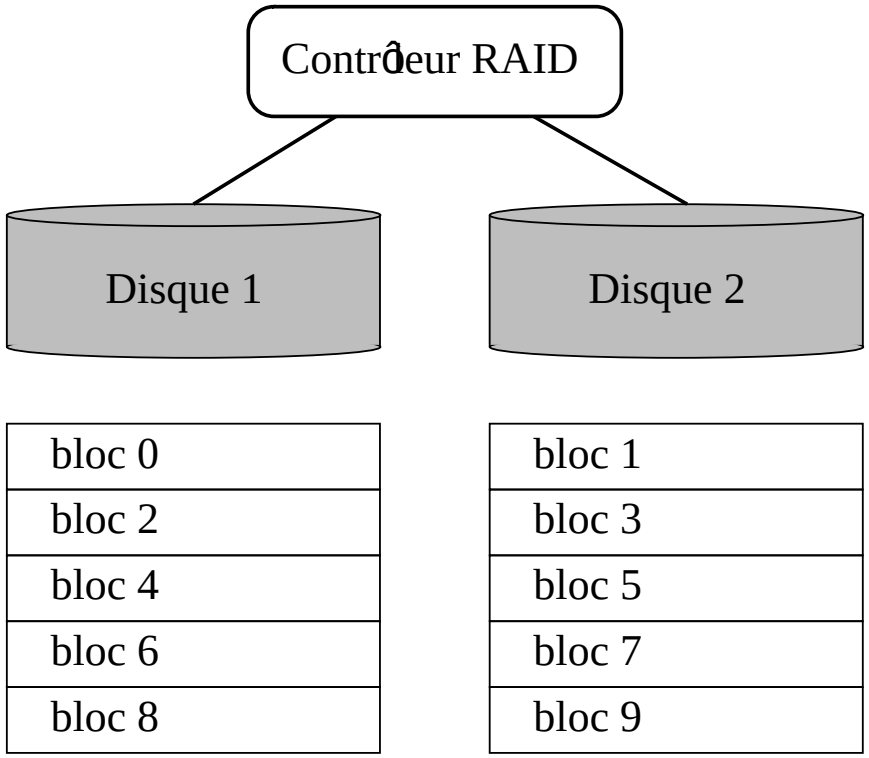
    $$\text{MTTF}_{\text{système}} = \frac{\text{MTTF}_{\text{disque}}}{\text{Nombre de disques}}$$

    - MTTF: Mean Time To Failure
    - Pour un MTTF de 30 000 heures (3,5 ans)
      - réseau de 1000 disques : une panne toutes les 30 heures !
      - réseau de 100 disque: une panne toute les deux semaines

Partage et réplication

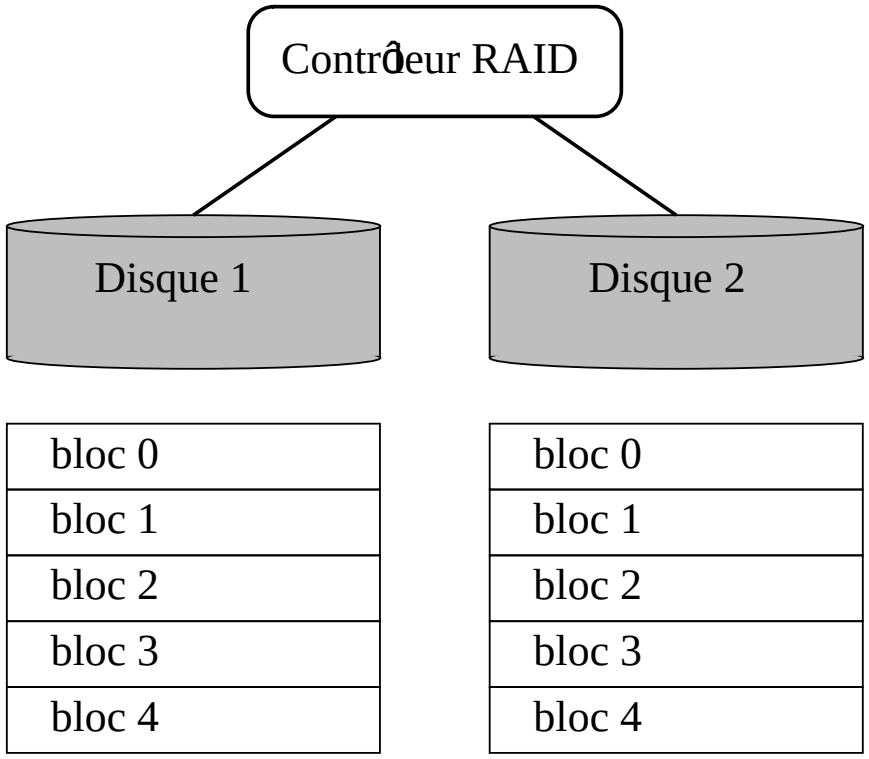
- RAID = Redondant Array of Inexpensive Disks
- Les disques sont répartis en groupes de *fiabilité*
- Chaque groupe possède des disques supplémentaires contenant de l'information redondante
- Si un disque tombe en panne, son information est reconstruite grâce à la redondance
- Le nombre de disque par groupe et le taux de redondance permet d'avoir des systèmes avec des caractéristiques différentes
- Patterson et al. classent les RAID par niveau (*level*)

RAID 0 (*Striping*): pas de redondance



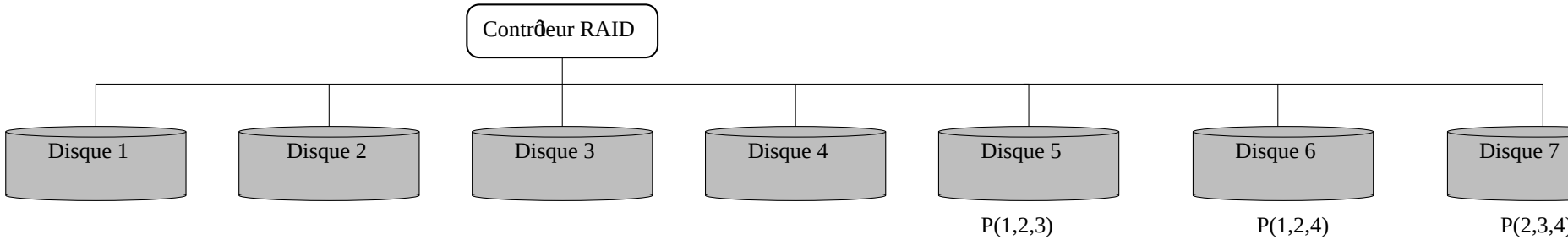
- Segmentation de données
  - pas de redondance
  - très bonne performances en lecture et écriture
  - problèmes de sureté de fonctionnement (panne=données perdues).
- Utilisation : 100%

RAID 1 (*Mirroring*)



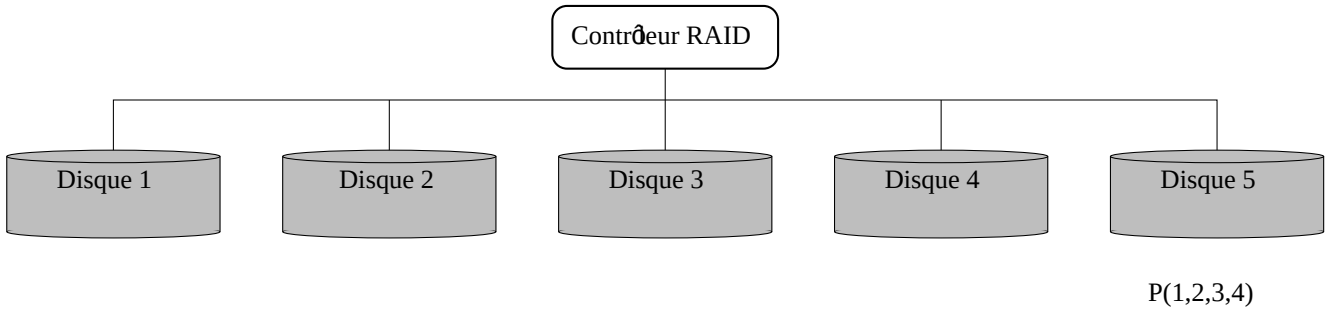
- Duplication des disques
  - redondance totale
  - accès en parallèle en lecture
  - écriture synchrone sur les 2 disques
- Sur-coût : 100%
- Utilisation : 50%

RAID 2 (*group parity check*)



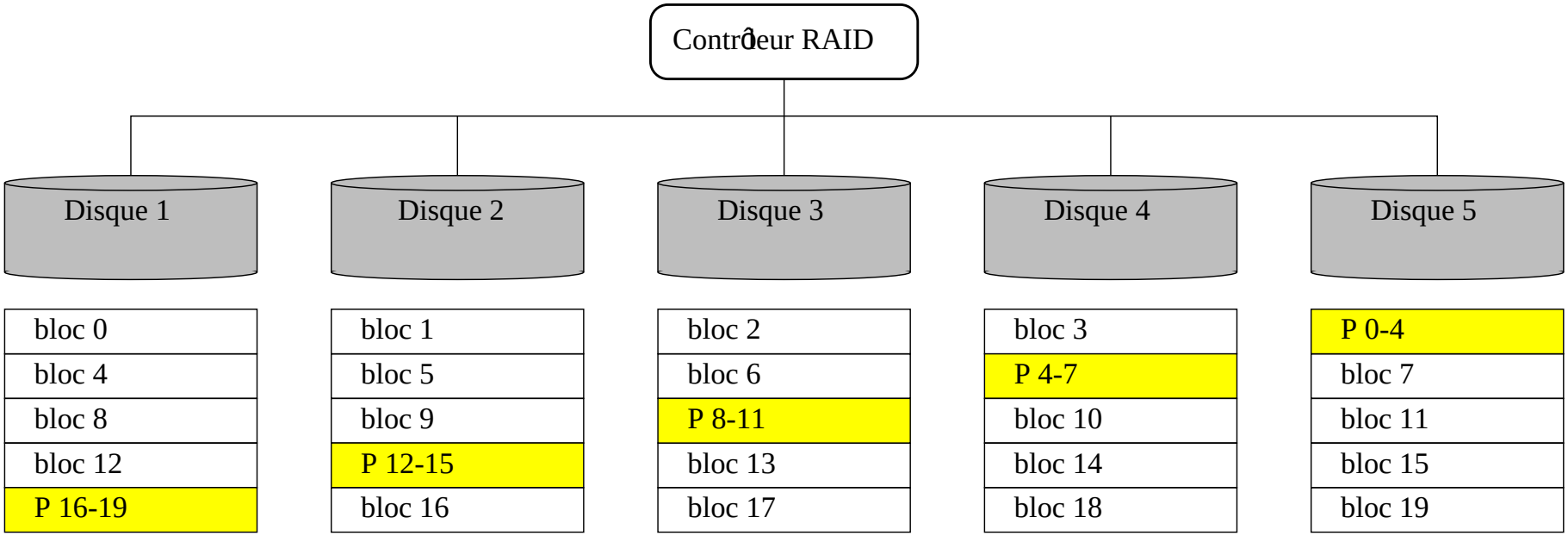
- Techniques de codage par parité, héritée des codages des mémoires.
  - Pour 4 disques de données: 3 disques supplémentaires qui contiennent la parité de chaque bit des disques (Log(N) disques supplementaires).
  - Supposons qu'un erreur détruise le bit N du disque 1
  - les parités des bits N des disques 5 et 6 seront fausses
  - La valeur de la parité permet de restaurer la bonne valeur
- Sur-coût : Log(N)%

RAID 3 (*bit interleaved parity check*)



- Dans le cas d'une problème de disque on peut identifier le disque fautif à partir du contrôleur de disque,
- il suffit donc d'un seul disque supplémentaire pour stocker la parité.
- Les données sont entrelacées niveau bit sur les N disques: bit 0 sur disque 1, bit 1 sur disque 2 etc.
- Une lecture utilise donc tous les disques (sauf le disque de parité), une seule requète est traitée simultanément.
- Sur-coût : 1

RAID 5: *Block interleaved distributed parity*



RAID 5

- Le plus utilisé pour les gros volumes
  - Les données de correction sont réparties sur l'ensemble des disques
  - Accès parallèles en lecture et écriture
  - S'emploie à partir de 3 disques
- Sur-coût : 33% à 4%
- Utilisation : 66% à 96%

Disponibilité : temps de réparation

- P : probabilité d'une autre erreur dans le groupe avant la réparation en cours

$$\text{MTTF}_{\text{groupe}} = \frac{\text{MTTF}_{\text{disque}}}{D + C} * \frac{1}{P}$$

- Remplacement des disques défectueux
  - Commutation automatique : «spare disks»
  - Remplacement manuel à chaud : disques «hot plug»

Disponibilité

- Le temps d'intervention en cas de panne rentre en compte
- MTTR : Mean Time to Repair

$$\text{MTTF}_{\text{RAID}} = \frac{(\text{MTTF}_{\text{disque}})^2}{(D + C * n_G) * (G + C - 1) * \text{MTTR}}$$

- Les formules sont les même pour tous les niveaux de RAID. On peut, par exemple, prendre D=100, G=9, C=1 et MTTR=1h pour obtenir un MTTF de 90 millions d'heures !!

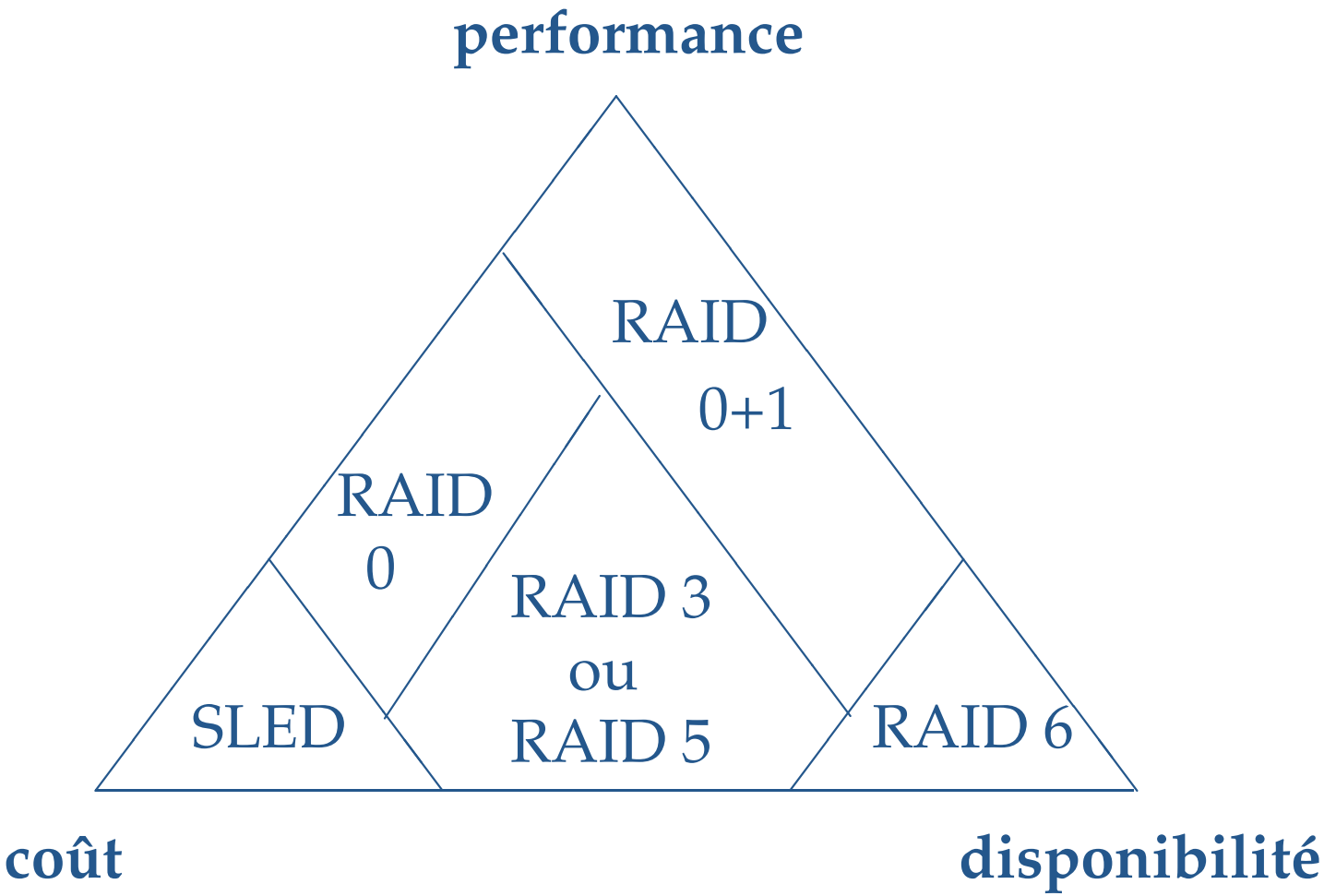
Performance(s)

- Organisation des disques suivant les besoins
  - Modèle transactionnel:
    - transferts de petites tailles
    - nombre de lecture/écriture indépendantes par seconde
    - besoin d'un taux d'E/S important
  - Modèle de simulation (supercomputing):
    - transferts de grandes tailles
    - besoin d'un *flux* important

Organisations RAID

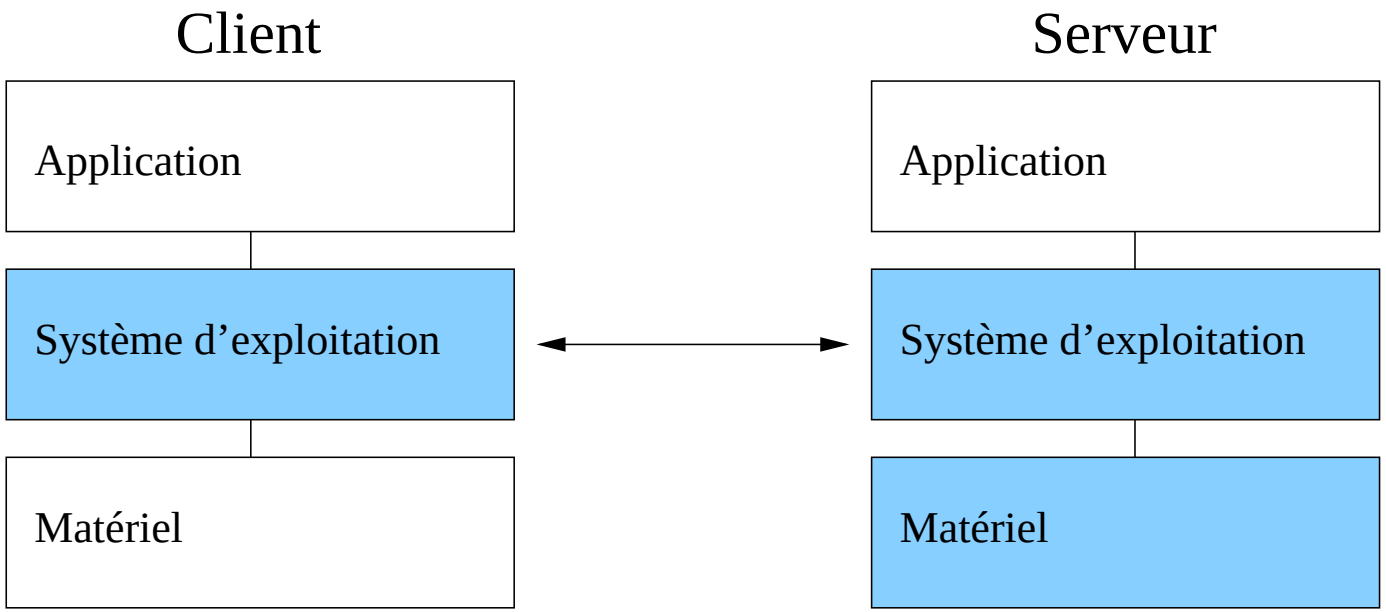
- RAID 0: Segmentation des données («striping»)
- RAID 1: Disques de donnés («Mirroring»,«Shadowing»)
- RAID 2: Réseau de disques avec correction d'erreur utilisant le code de Hamming (obsolète)
- RAID 3: Réseau utilisant un disque de parité. Les données sont segmentées par bits, octets ou par mots.
- RAID 4: Réseau utilisant un disque de parité. Les données sont segmentées par secteur ou par groupe de secteurs
- RAID 5: Réseau de disques avec contrôle de la parité distribué sur l'ensemble des disques
- RAID 6: Réseau de disques avec double contrôle de parité sur l'ensemble des disques (C=2)

Organisations RAID



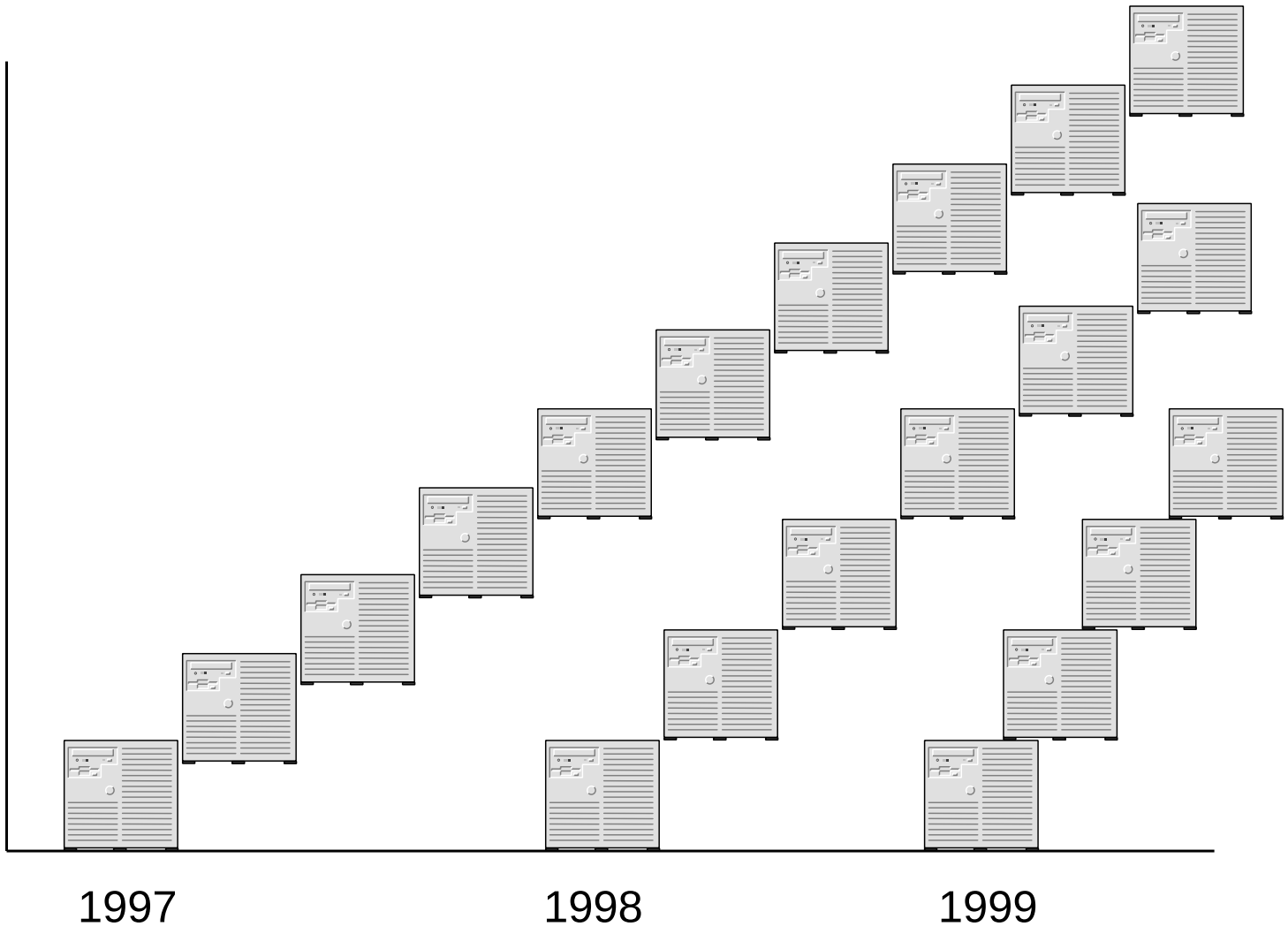
Virtualisation des données

Distribution au niveau du système d'exploitation



Stockage en réseaux

Croissance des besoins en stockage



Stockage sur un réseau et stockage réparti

- NAS : Network Attached Storage

  Stockage accessible depuis le réseau local en utilisant des systèmes de fichiers tels NFS ou CIFS (SMB)

- SAN : Storage Area Network

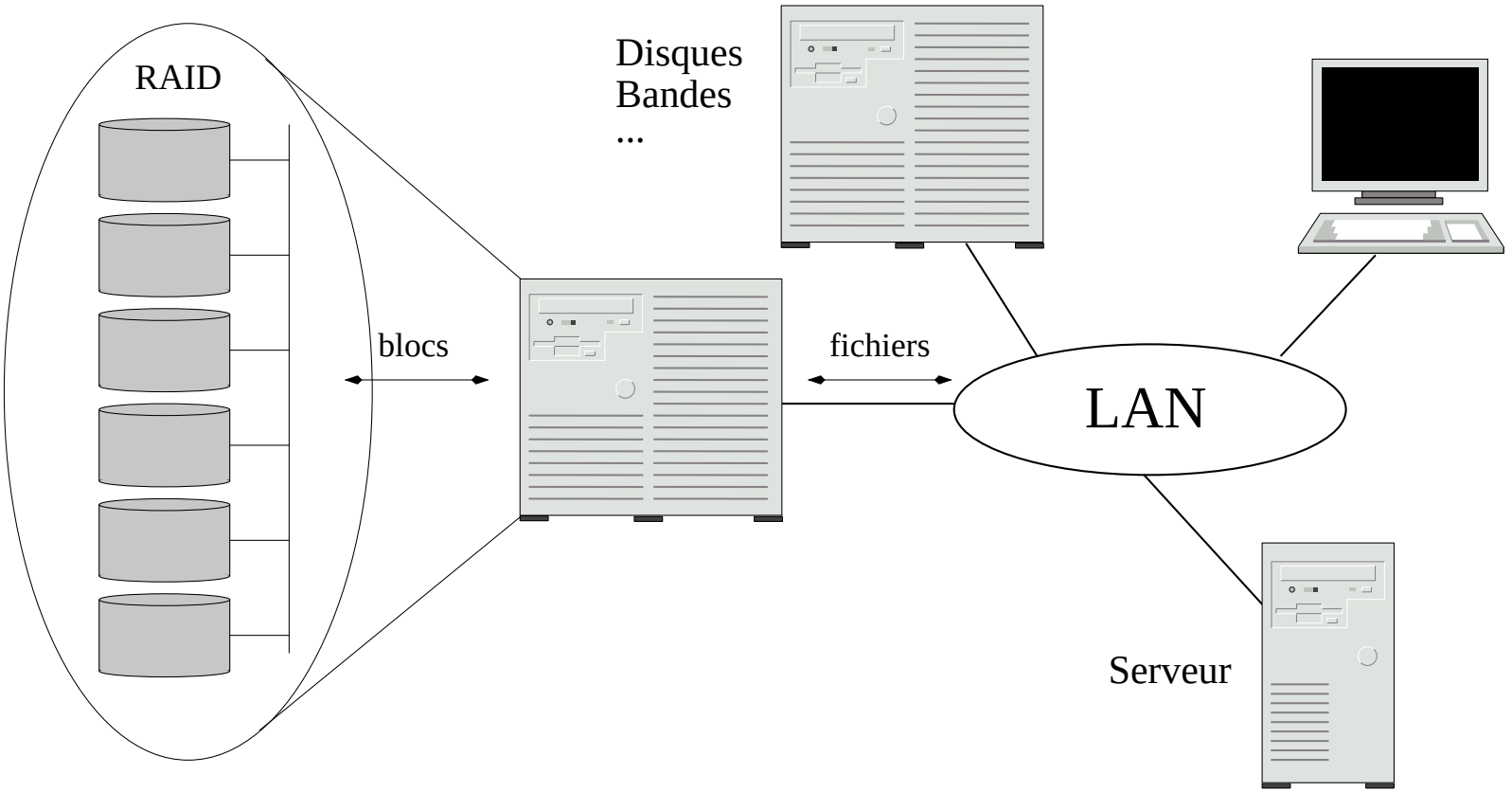
  Réseau indépendant de périphériques de stockage

- SSP : Storage Services Providers

  Location d'un espace disque chez un hébergeur

NAS : Network Attached Storage

- Essentiellement une boîte prête à l'emplois avec un SE embarqué conçu pour optimiser les transactions entre les services réseaux (NFS, SMB, FTP, ...) et les disques

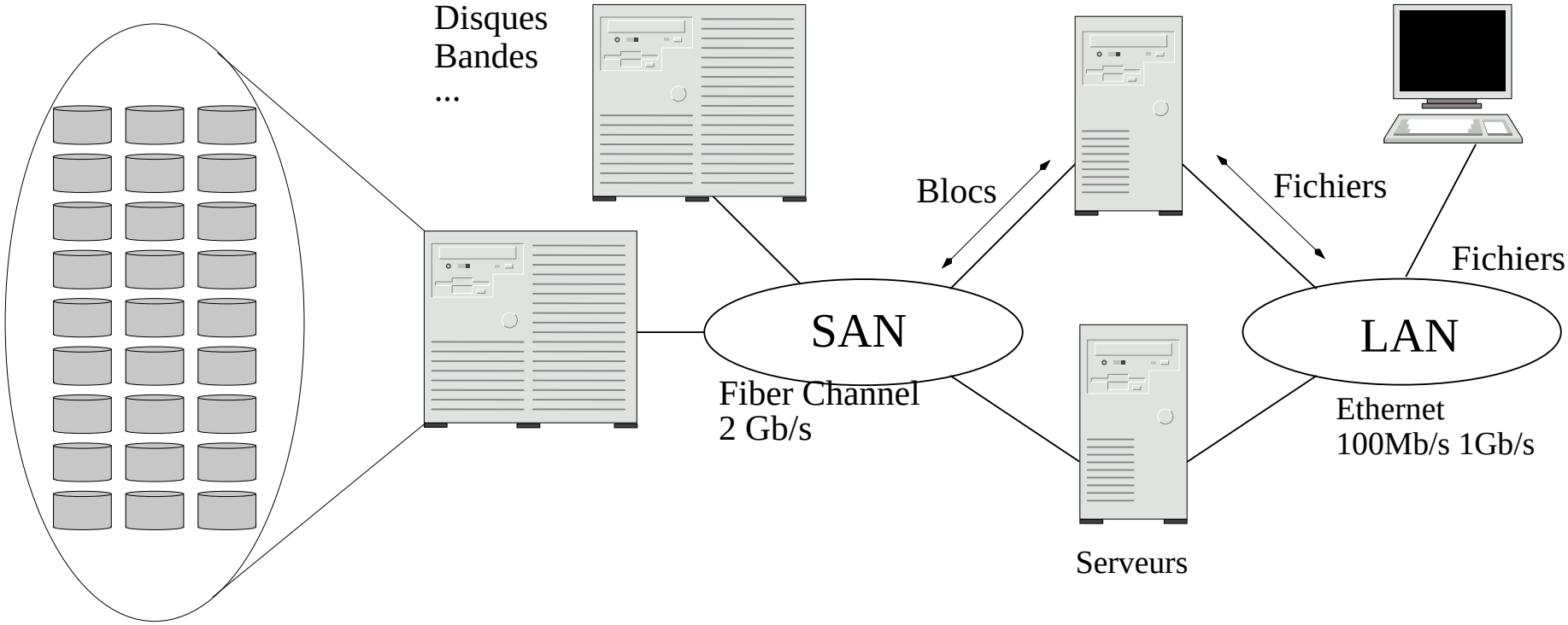


NAS : Network Attached Storage

- Systèmes efficaces (serveurs dédiés)
- Disques organisés en RAID 0/1/5
- Implémente un système de fichier évolué
  - journalisé pour éviter les problèmes d'inconsistance
  - capable de faire des images pour les sauvegardes

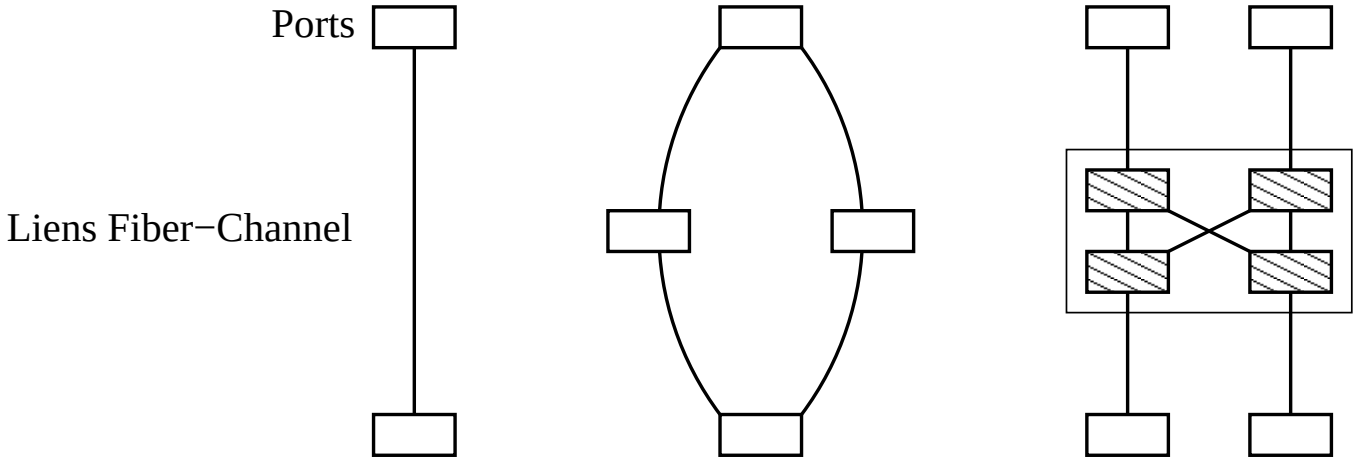
- Les échanges entre les serveurs de fichiers et les serveurs d'applications ou les clients passent par le LAN
- Problème de temps de sauvegarde trop importants

SAN : architecture



SAN : fonctionnement

- Sous-réseau rapide de composants de stockage partagés (connexions Fibre-Channel multi Gigabit)
- Un SAN met à disposition de tous les serveurs sur un LAN (WAN) les composants de stockage
- Un composant de stockage est une boîte contenant des disques et rien d'autre
- La panne d'un serveur ne bloque pas les données



SAN

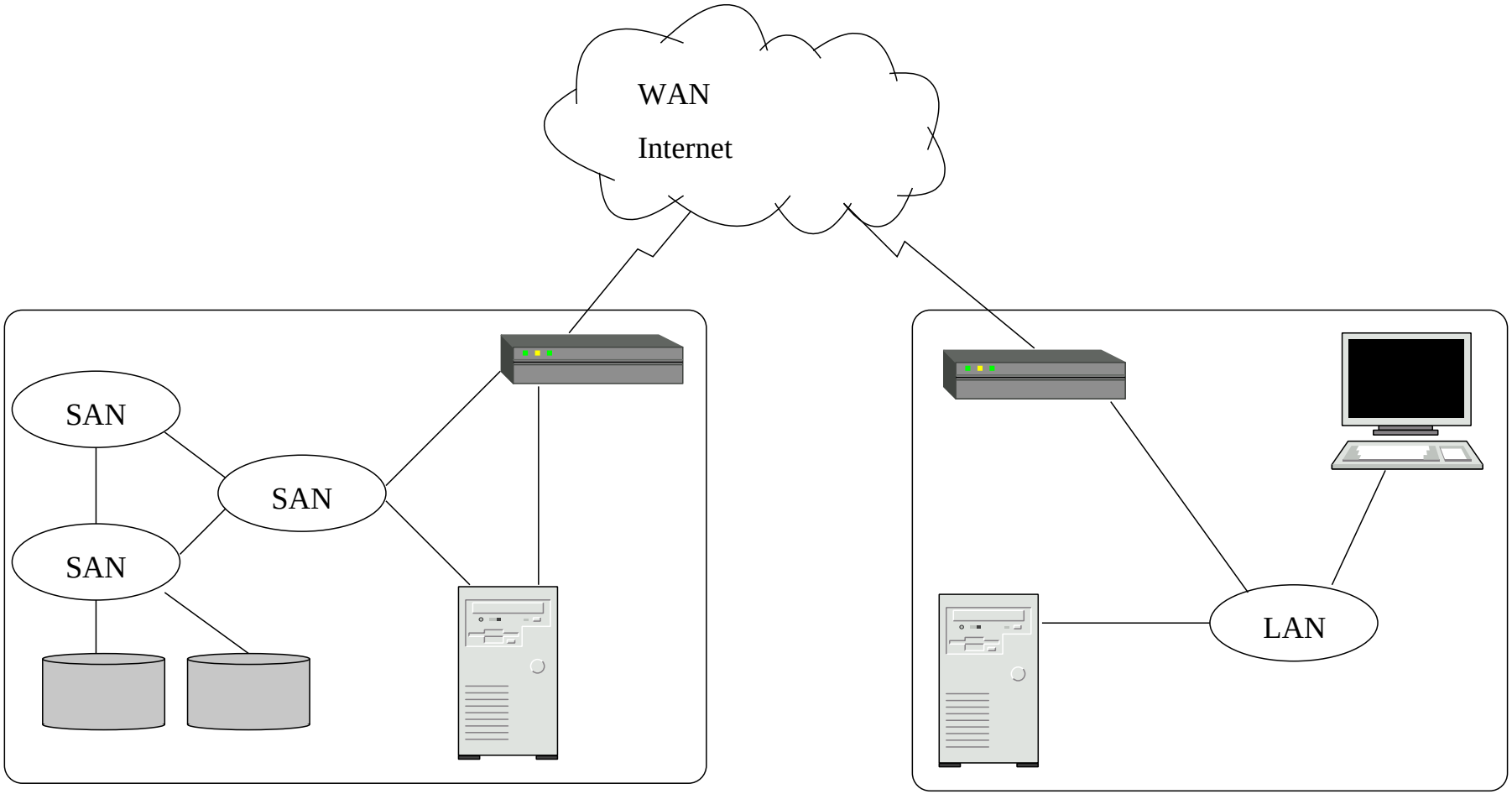
- Virtualisation du stockage
- Extensibilité d'un LAN
- Les serveurs sont utilisés pour les applications
- Bande passante du LAN laissée aux utilisateurs
- Fédération des équipements (sauvegarde, baie de disques)
- Autonomie du stockage vis à vis des réseaux
- Notions de droits (routage / zonage) pour les accès

- Mais : interopérabilité difficile et grosse charge d'administration

SSP : Storage Services Providers

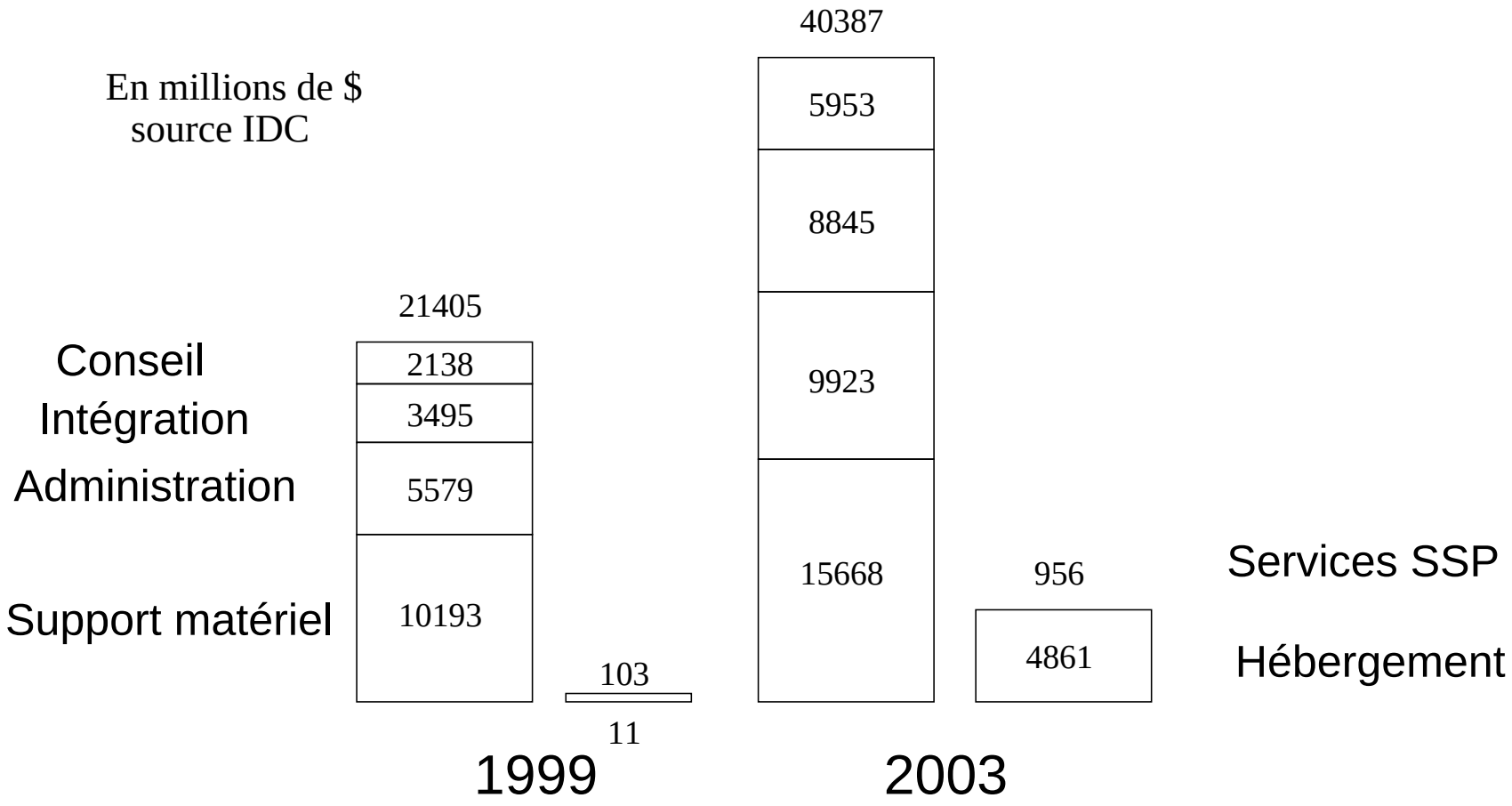
- Un SSP est une compagnie qui propose un espace de stockage et des services de gestion
  - sauvegarde
  - archivage
  - partage de données entre plusieurs sites
- Avantages : réduction des coûts de possession
  - évolutivité
  - maintenance
  - assurance

SSP : architecture



Évolution du marché

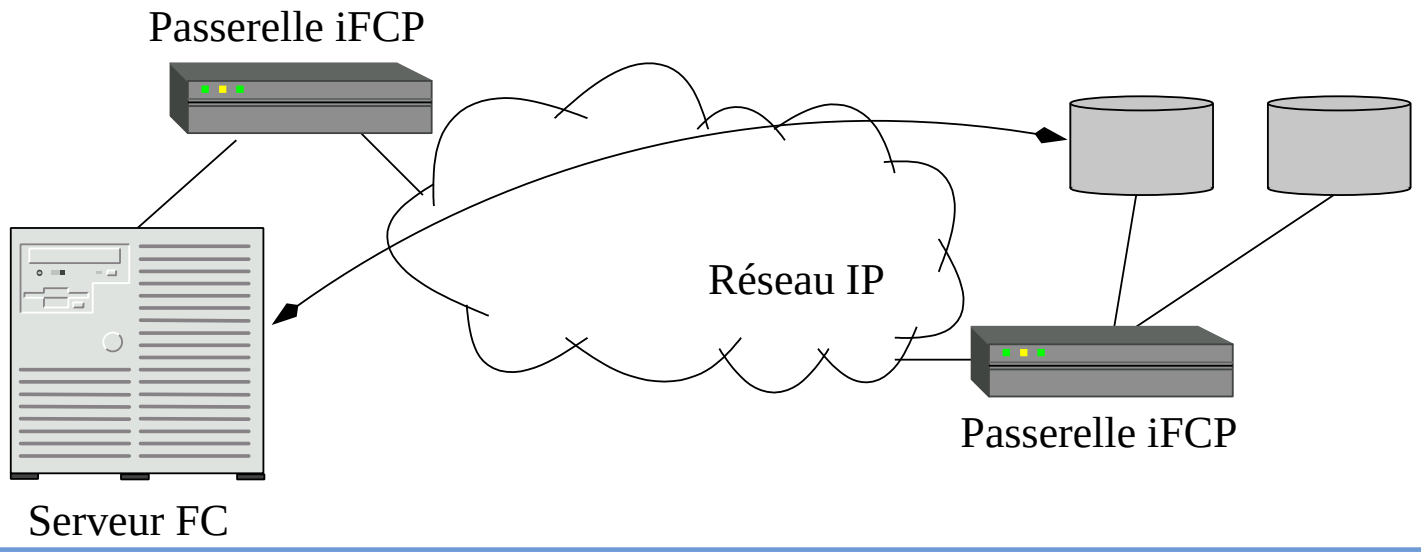
- Dépenses en service de stockage (États Unis)



Évolution des techniques

- Virtualisation de l'architecture
- iSCSI: Internet SCSI
  - Protocole SCSI encapsulé dans IP
- iFCP: Internet Fiber Channel Protocol
  - Définition d'un protocole de passerelle à passerelle
  - Permet le rattachement de produits FC à des réseaux IP

SCSI
iSCSI
TCP
IP
Liaison



Conclusion

- Les données sont devenues la principale richesse des entreprises
- Elles ont maintenant leur place dans les technologies des systèmes distribués
- L'administration et l'évolutivité nécessitent de rendre prendre de la distance entre le stockage physique et son utilisation

- Virtualisation du stockage des données

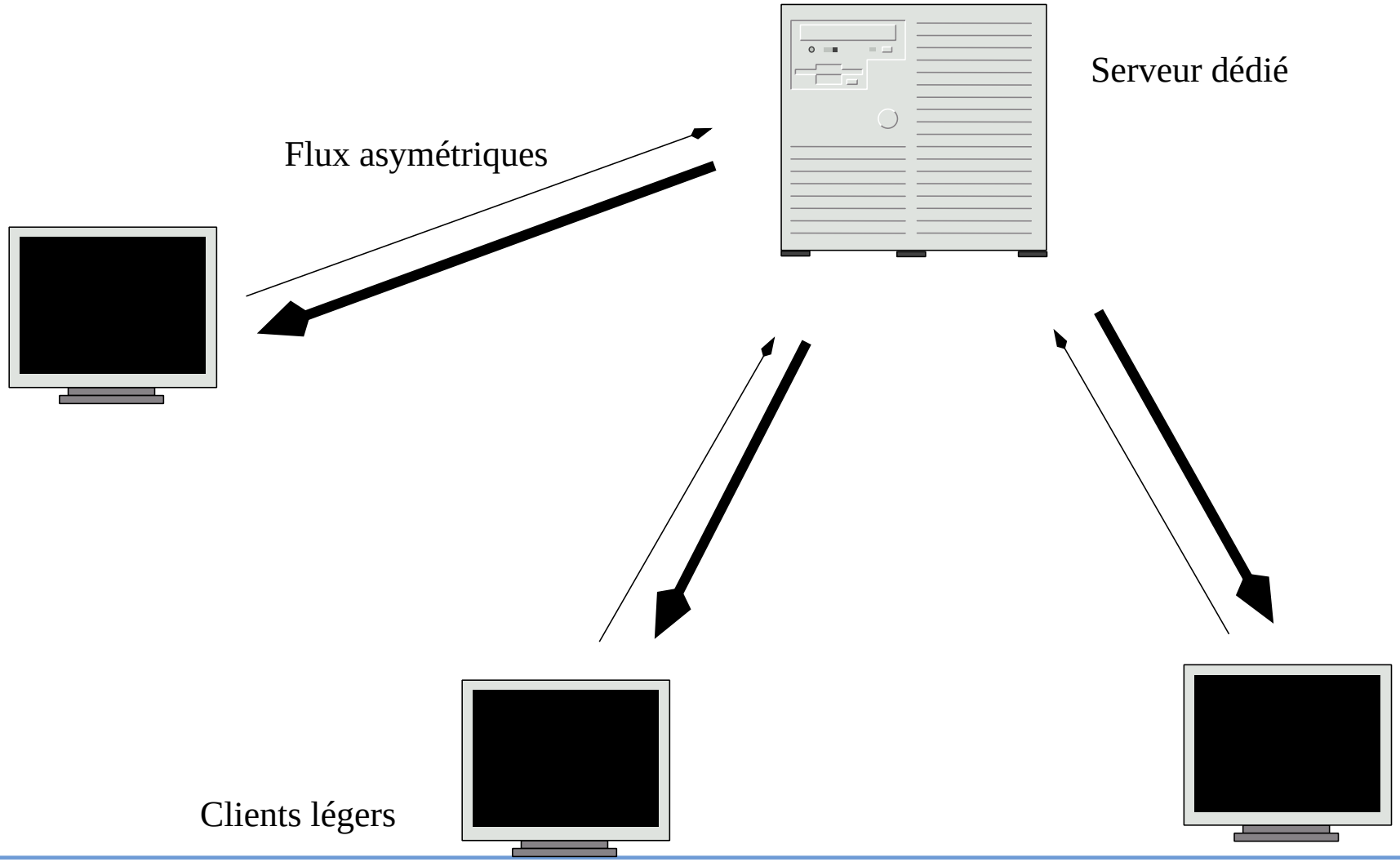
Plan

3 Étude de cas

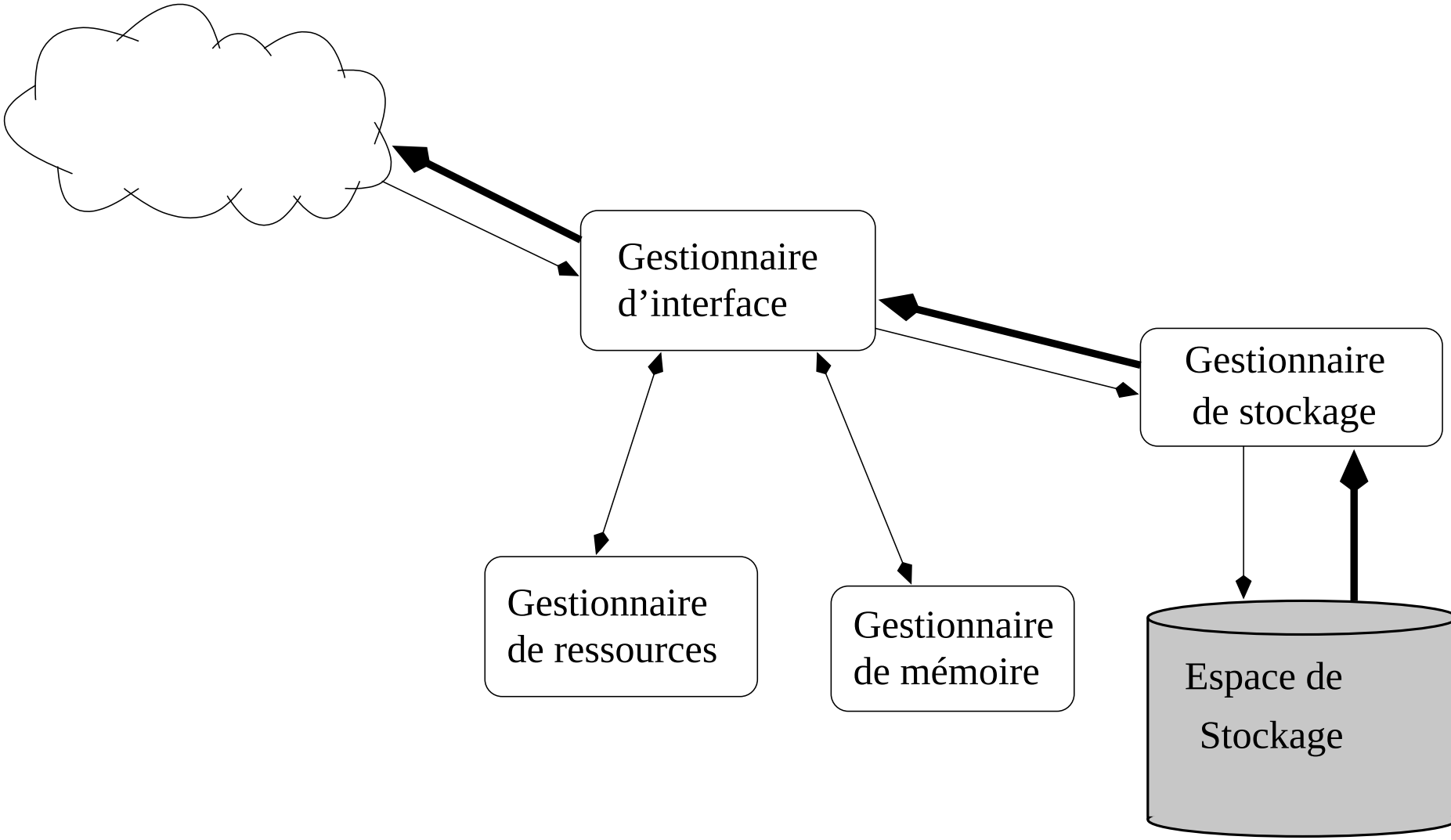
1. Distribution de ressources
   - Distribution matérielle
   - Distribution au niveau du système d'exploitation
   - Distribution applicative
2. Virtualisation de données
   - Distribution au niveau des disques
   - Distribution au niveau du système d'exploitation
3. Étude de cas
   - Serveur de vidéo à la demande
4. Systèmes de fichiers distribués

Étude de cas

Serveur de vidéo à la demande



Serveur Vidéo



Serveur Vidéo existant

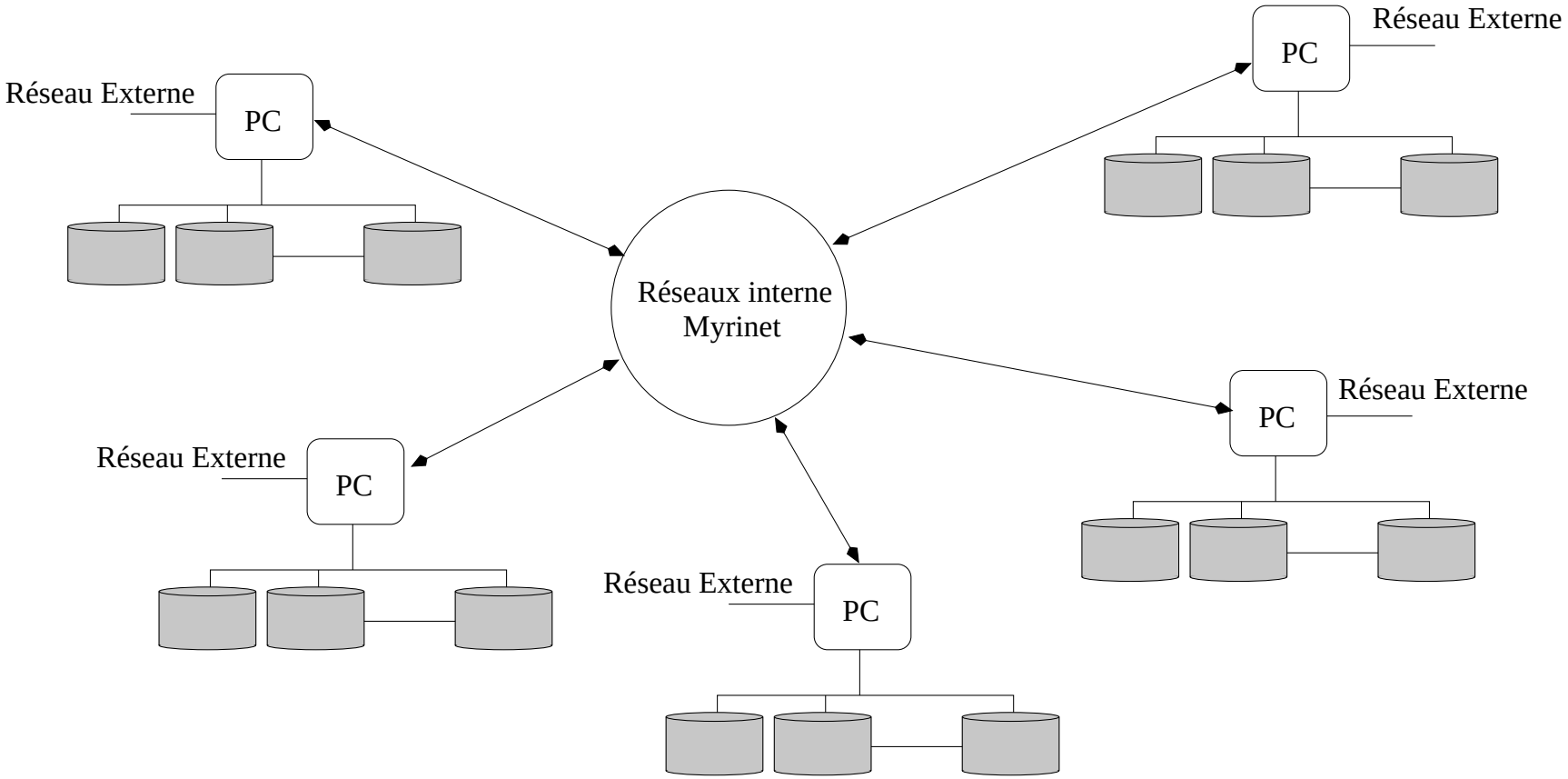
- Serveurs commerciaux
  - Tiger (Microsoft94à, serveur dédié, solution totalement distribuée sur un ensemble de PC).
  - Fellini (At&T), Machine à mémoire partagée

	Tiger	Fellini
Architecture	distribuée	partagée (SMP)
Placement des données	Bloc de taille variable allocation variable	bloc de taille fixe allocation cyclique
Type de codage	CBR	CBR & données statiques
Accès interactif	non	oui
Points forts	pas de synchronisation	deux API (Temps réèl ou pas)
Points faibles	un seul débit	mémoire partagée

Objectifs souhaités

- Grand nombre de clients supportés et faible coût
  - développement réduit
  - performance garanties
- Transparence de la gestion du système vis-à-vis des clients
  - Système perçu comme une machine unique
  - Communication avec le client simple
- Système fiable
  - tolérance aux pannes des composants
  - maintenir la disponibilité des données et les performances
  - reconstruire des données après une panne

Performance à faible coût



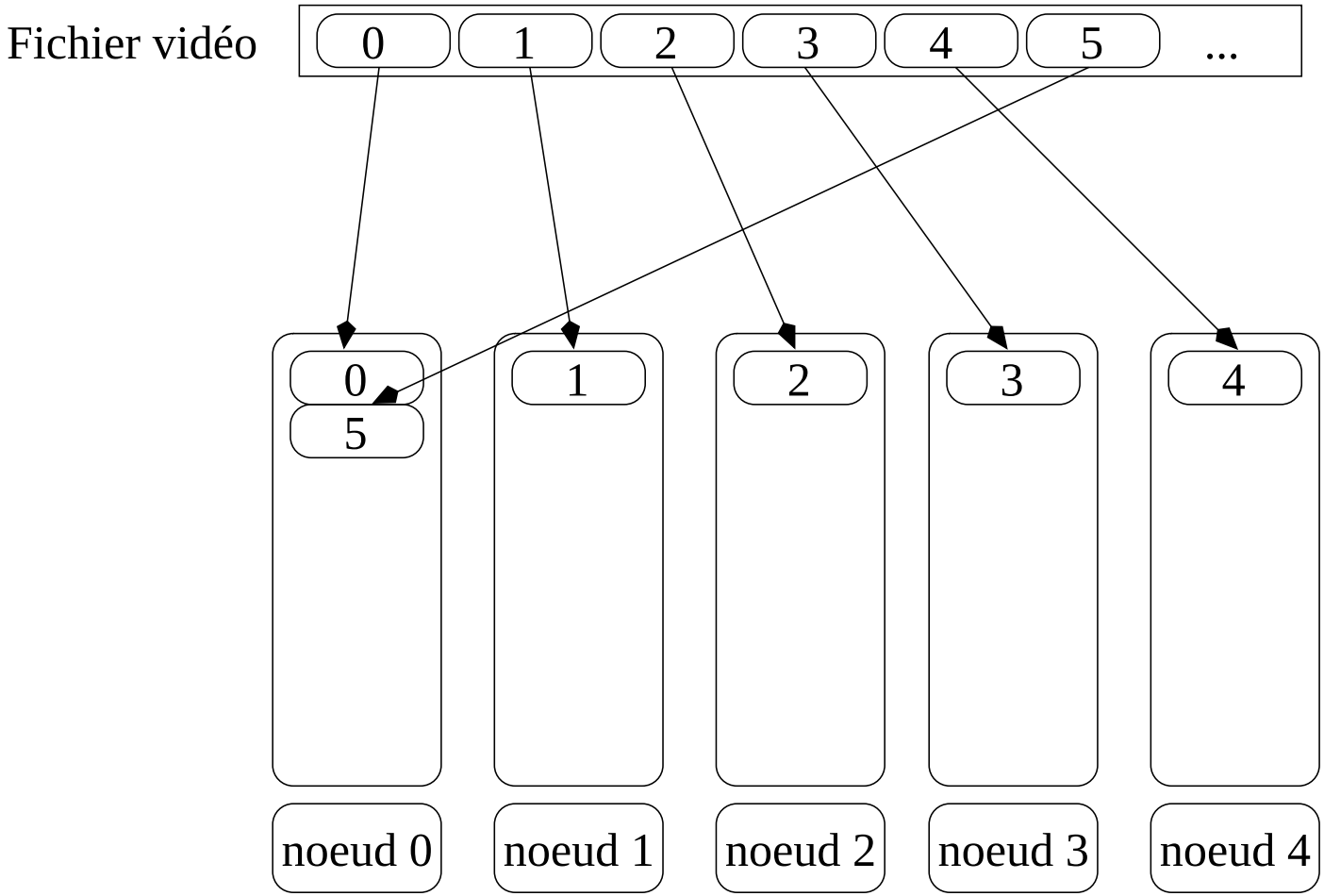
- Utilisation d'une grappe de PC
- Distribution des données sur les nœuds
  - équilibrage de charge

Tolérance aux pannes

- Utilisation des données de redondance
  - faible espace de stockage supplémentaire
- Stratégie *Streaming RAID*
  - pas d'utilisation de bande passante en cas de panne
  - modification pour éliminer le délai de recouvrement de panne

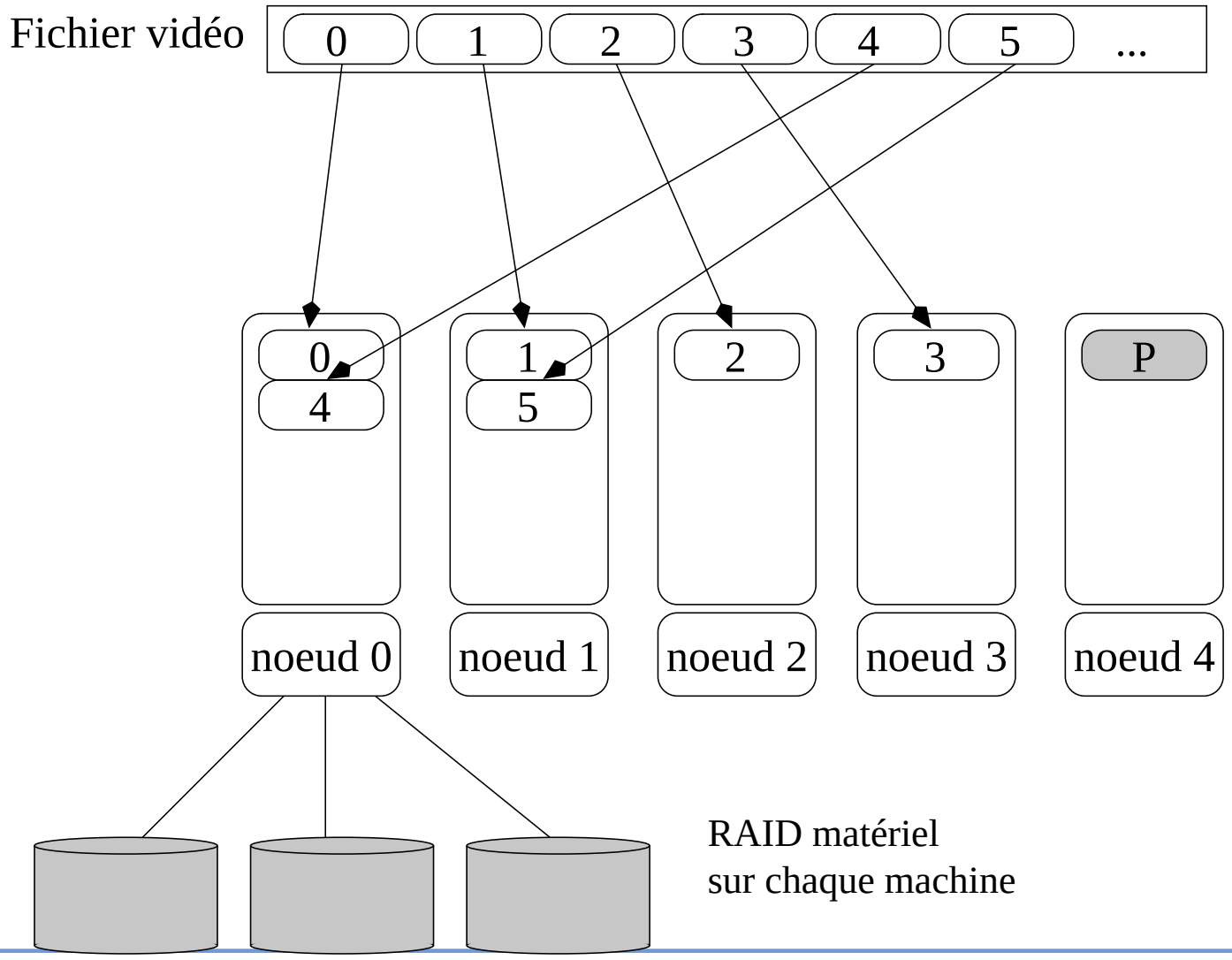
Performance à faible coût

Distribution des données



Tolérance aux pannes

Redondance des données

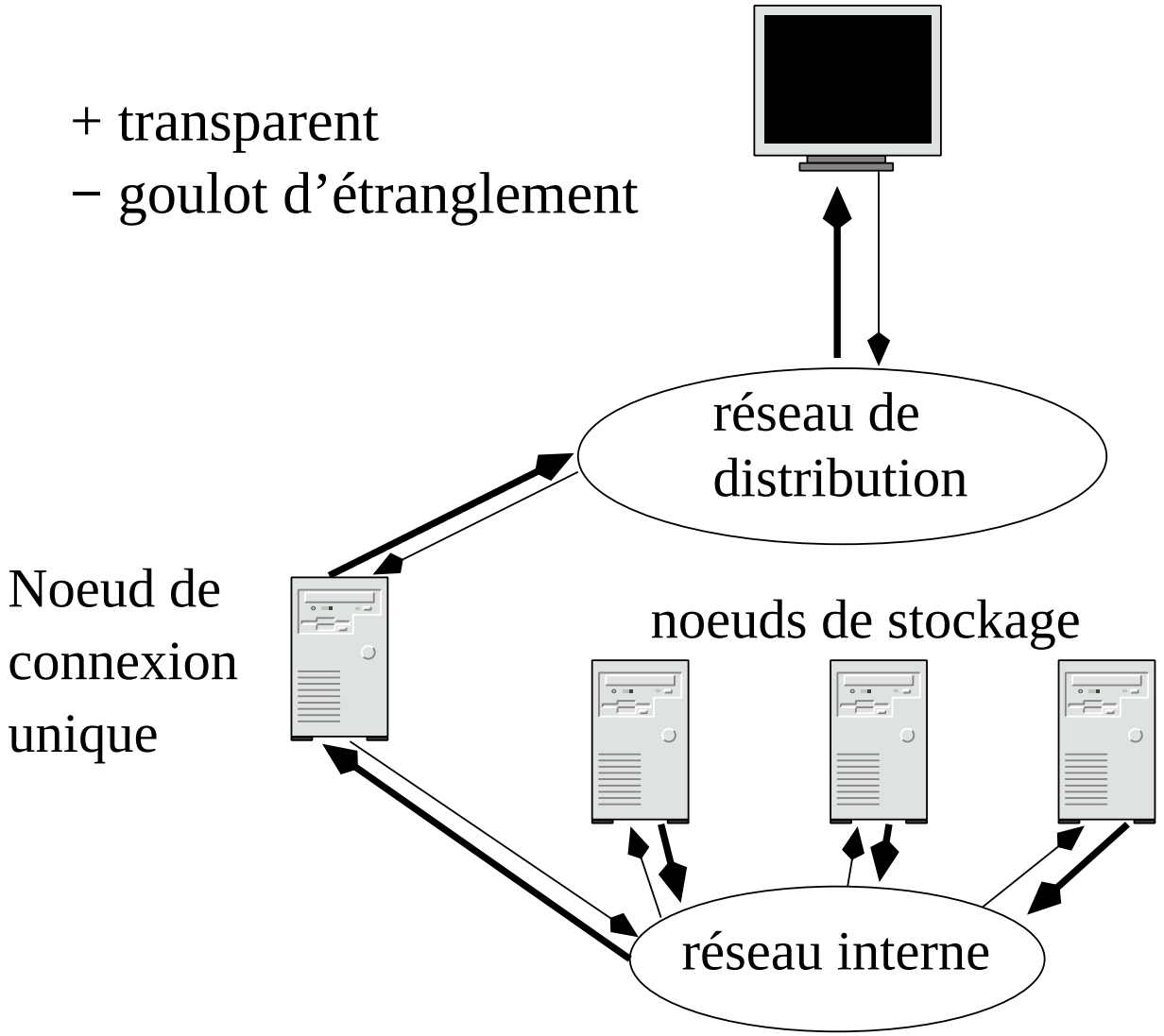


Transparence

- Le client ne dois pas apercevoir
  - la gestion distribuée des données
  - la gestion de la tolérance aux pannes
- 3 approches possibles pour gérer la grappe de PC
  - centralisé
  - semi-centralisé
  - distribué

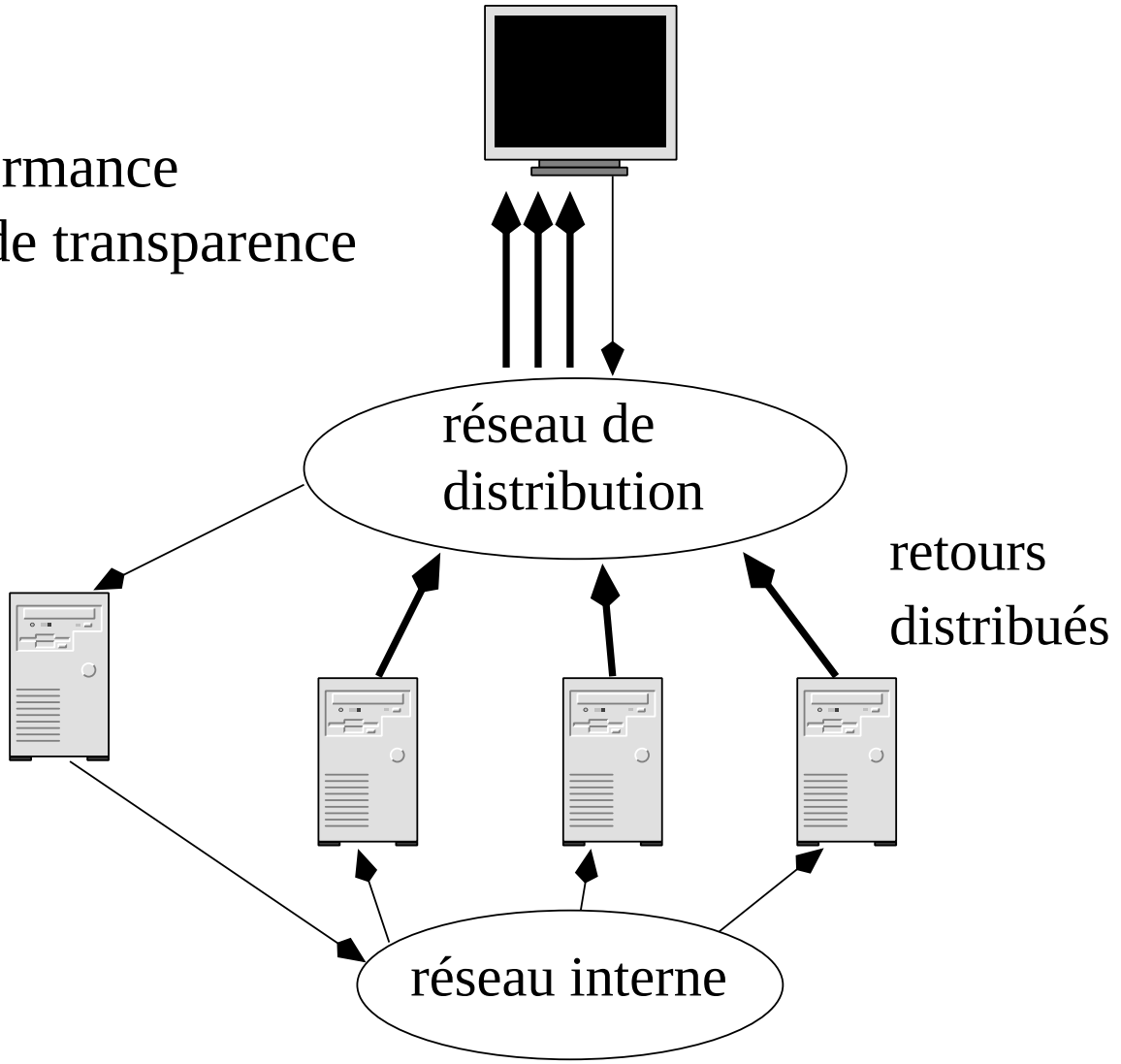
Transparence : approche centralisée

+ transparent
− goulot d'étranglement



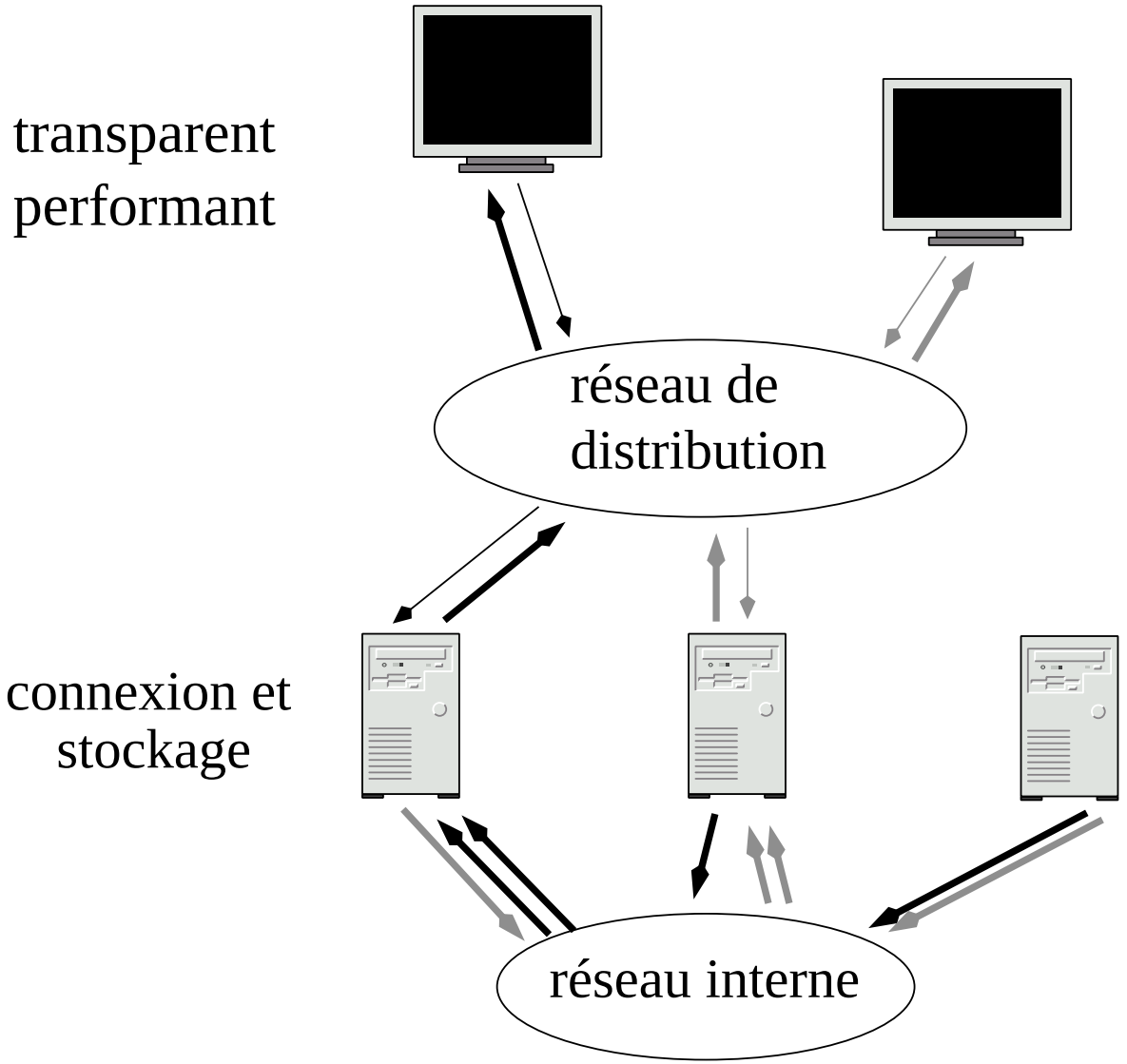
Transparence : semi-centralisé

+ performance
− pas de transparence

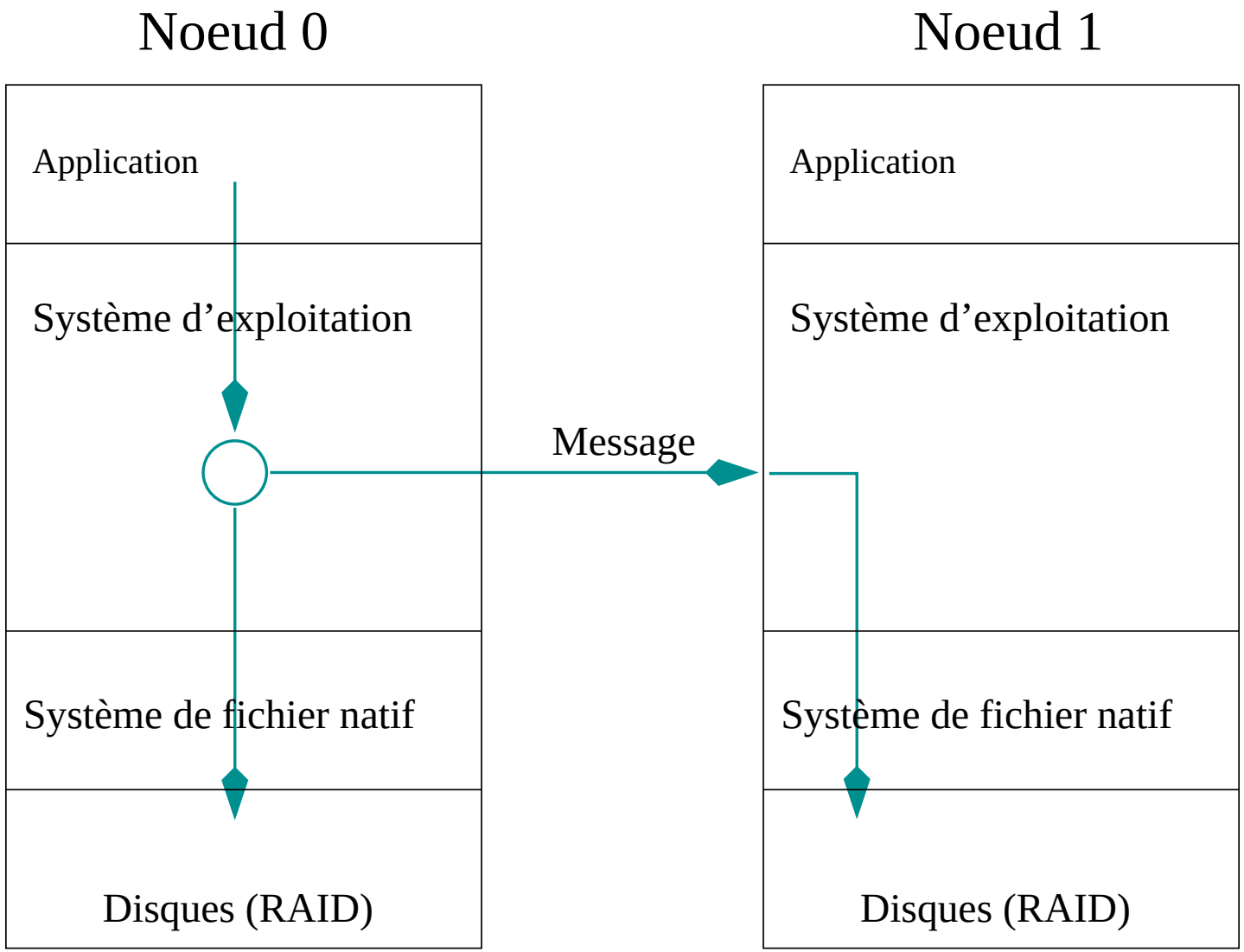


Transparence : distribué

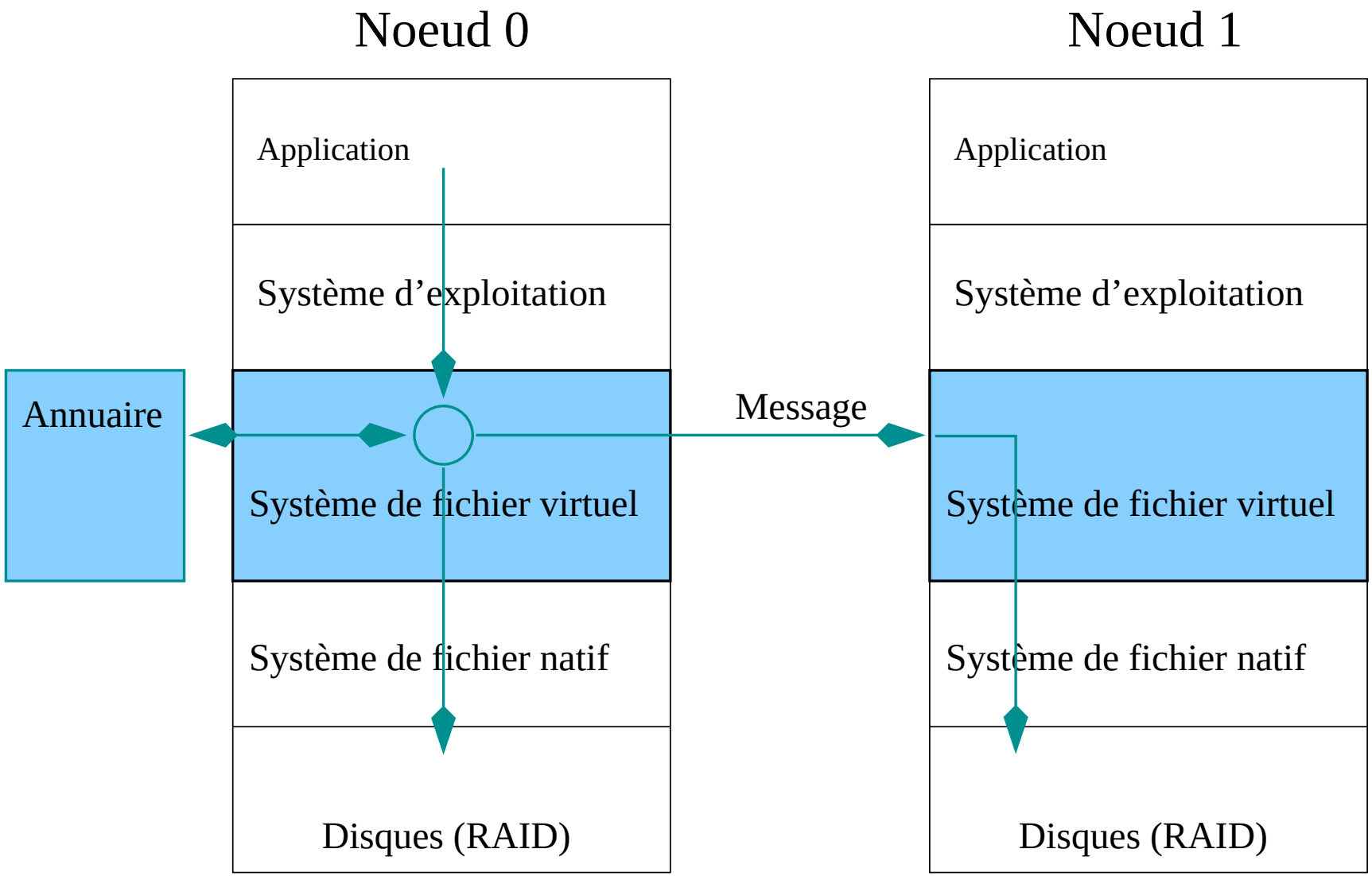
+ transparent
+ performant



Réalisation de la distribution



Réalisation de la distribution



Réalisation de la distribution

- Table des fichiers distribués
  - nom du fichier
  - type d'accès
  - compteur d'accès
  - méta-informations : volume distribué, taille, date de dernière modification
- Table des fichiers répliquée sur tous les nœuds
  - Contrôle de la cohérence
  - Exclusion pour modifications (granularité)
  - Propagation des modifications (granularité)
- Procédure de démarrage et d'arrêt
- Procédure de reconstruction après panne

Plan

4 Systèmes de fichiers distribués

1. Distribution de ressources
   - Distribution matérielle
   - Distribution au niveau du système d'exploitation
   - Distribution applicative
2. Virtualisation de données
   - Distribution au niveau des disques
   - Distribution au niveau du système d'exploitation
3. Étude de cas
   - Serveur de vidéo à la demande
4. Systèmes de fichiers distribués

Distribution des fichiers

- Service de fichiers
  - interface proposée pour la manipulation de fichiers
  - création / modification / contrôle d'accès
- Serveur de fichiers
  - machine proposant la réalisation du service
  - une machine peut proposer plusieurs services (NFS, SMB, . . .)
- Exemples : Andrew File System, Sun Network File system (NFS), Bayou, Coda

Système de fichier distribué

- Système de fichier classique: facilité d'interfaçage avec le stockage sur disque.
- Un système de fichier distribué doit être transparent au niveau de l'utilisateur (i.e. semblable à un système de fichier classique):
  - performance,
  - API,
  - tolérance aux pannes (panne réseau, panne de serveurs).
- Parmi les premier systèmes distribués réalisés (recherche en 1970, NFS début des années 80).

Rappel: fichier

- Un fichier contient des données et des attributs (ou meta-données), attributs typiques:

Taille fichier
Date de création
Date de lecture
Date d'écriture
nombre de références
Propriétaire
type de fichier
Liste de contrôle d'accès

- Répertoire: type particulier de fichiers.

Architecture système de fichier classique

- Un système de fichier contient toujours plus ou moins les même modules:
- Les fichiers sont manipulés par le système de fichier par des identificateurs interne (ID, exemple: i-node).

Module de Répertoire	Relie les noms de fichiers aux IDs
Module de fichier	Relie les IDs aux fichiers
Module de con-tôle d'accès	vérifie les permissions
Module d'accès aux fichiers	écriture ou lecture (données/attributs)
Module de blocs	alloue/désalloue les blocs sur le disqu
Module d'E/S	E/S sur le disque et tampon

Exemple d'unix

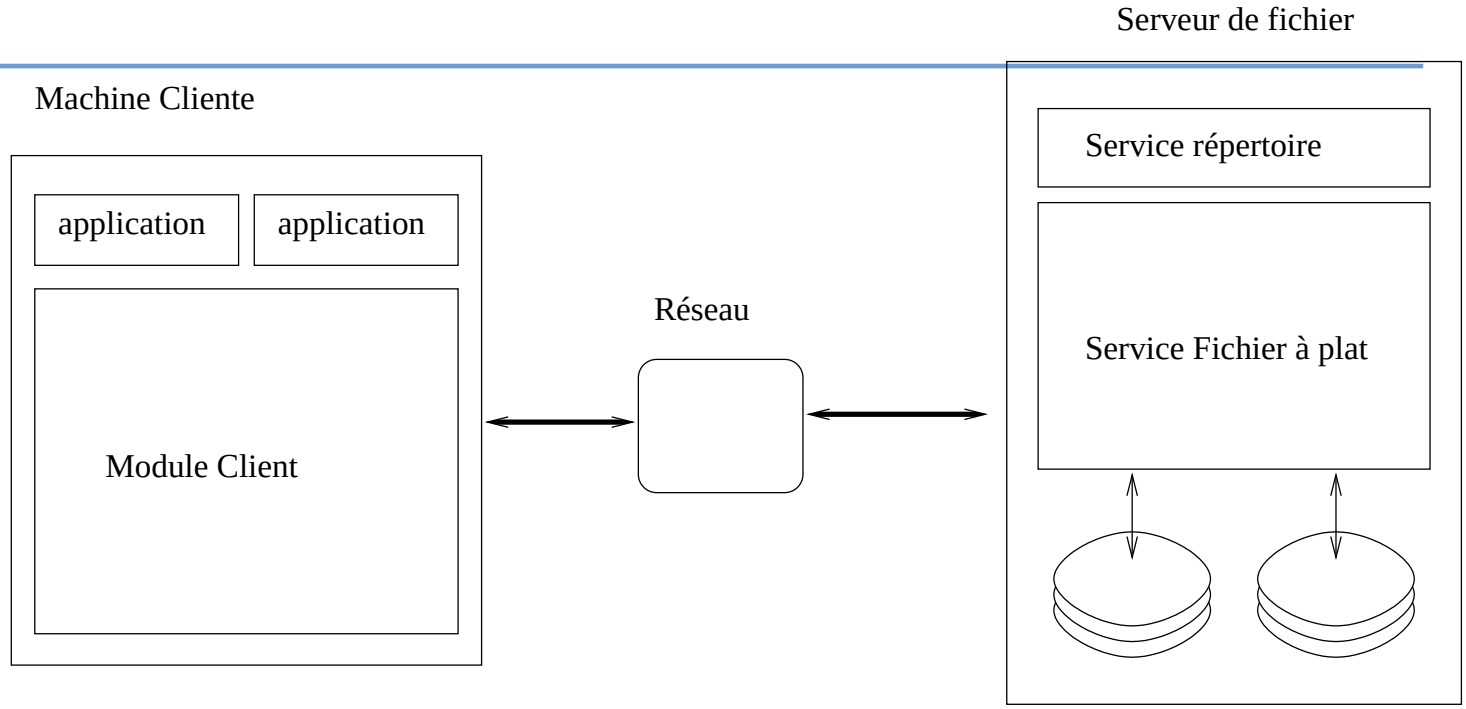
- API du système de fichier:

```
filedes=open(name,mode)
filedes=creat(name,mode)
status=close(filedes)
count=read(filedes,buffer,n)
count=write(filedes,buffer,n)
pos=lseek(filedes,offset,whence)
status=unlink(name)
status=link(name1,name2)
status=stat(name,buffer)
```

Système de fichiers distribués

- Transparence:
  - Même type d'accès pour tous les fichiers (local/distant)
  - Même type de noms (*uniform file name space*)
  - Même type de performances
  - Déplacement physique de fichiers distants.
  - Passage à l'échelle
- Mise à jour concurrente
- Hétérogénéité matérielle.
- Tolérance aux fautes
  - Communications (opérations idempotentes)
  - Pannes des serveurs (modules "sans état")
- Sécurité
- Efficacité

Architecture d'un système de fichier distribué



- Service de fichier à plat (flat file service): UFID, Read, Write, Create, Set/Get attribute
- Service de répertoire: correspondance nom–UFID
- Client: RPC, mécanisme de cache

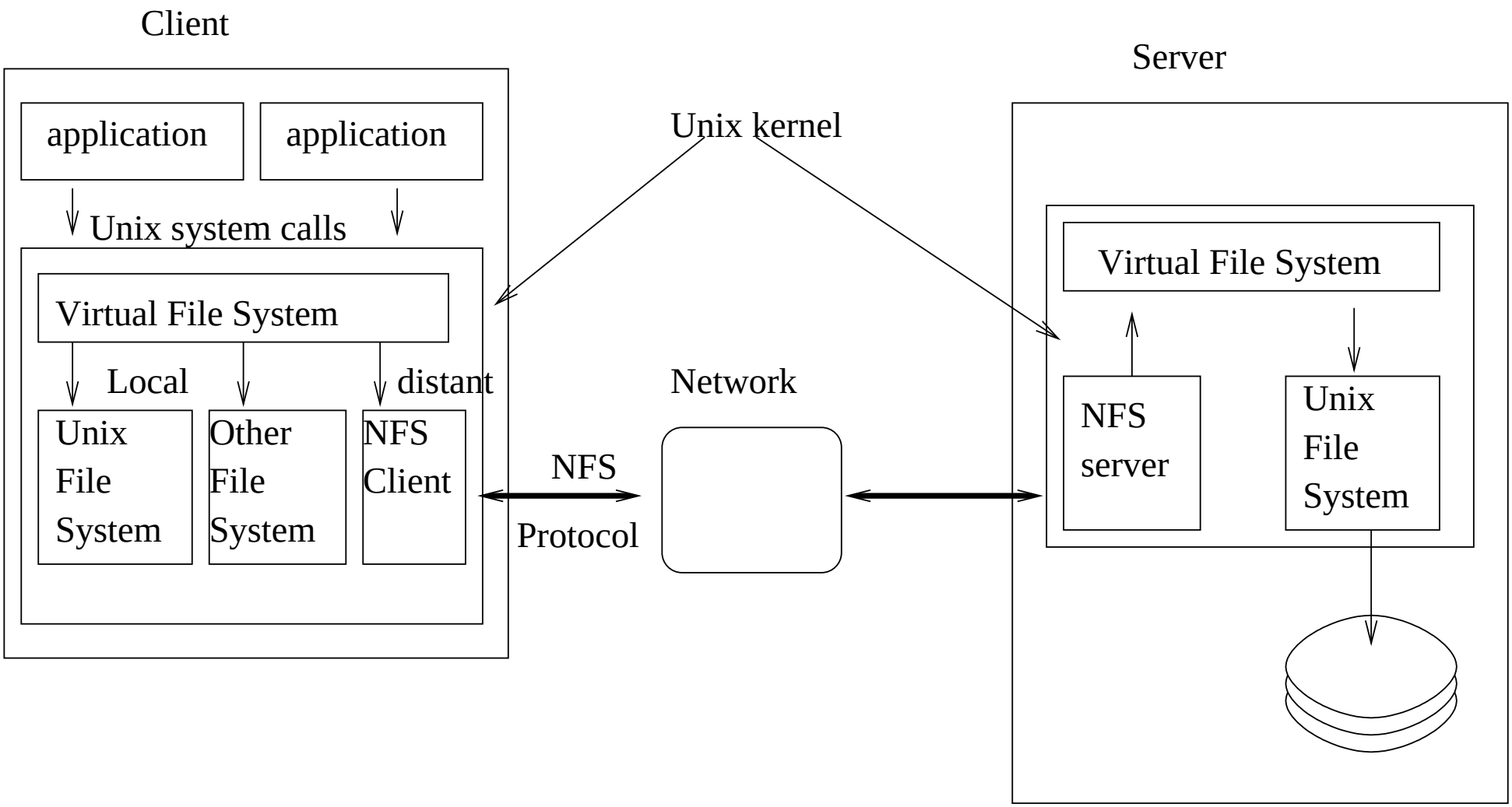
Comparaison avec Unix

- Fonctionnellement équivalent mais:
  - Pas d'opération `open` et `close`
  - Opérations d'E/S repetable (idempotente)
  - Pas d'état de fichier stocké dans le système (tolérance aux pannes)
  - Contrôle d'accès à chaque accès

Sun NFS

- Sun Network File System: spécification indépendante de l'OS hôte.
- Sur Unix: chaque processeur possédant des fichiers partagés possède un serveur NFS intégré au noyau unix.
- Chaque client possède un client NFS intégré au noyau unix.

Sun NFS


Client
application
application
Unix system calls
Virtual File System
Local
distant
Unix
File
System
Other
File
System
NFS
Client
Unix kernel
Network
NFS
Protocol
Server
Virtual File System
NFS
server
Unix
File
System

Virtual File System (VFS)

- Homogénéiser les appels systèmes et les noms de fichiers.
- Identificateurs de fichiers utilisé par VFS (*file handle*):

Filesystem identifier	i-node number of file	i-node generation number

- chaque fichier possède un v-node dans VFS: soit un i-node (fichier local) soit un *handle* (fichier distant)

Cache du serveur

- Les systèmes de fichier conventionnels utilisent un buffer d'E/S comme un cache avec les caractéristiques suivante:
  - *read-ahead* (accès séquentiel)
  - *delayed-write* (sync toutes les 30 secondes)
- Système de fichier distribués
  - write-through: mémoire cache *et* écriture disque avant acquittement.
  - write-back: écriture sur disque uniquement lors des *commit*: plus performant, moins tolérant aux pannes.

Cache du client

- Le client stocke en cache les résultats de requête (read, write, getAttribute, lookup, readdir)
- *poling* pour vérifier la cohérence des données dans le cache.
- Deux dates étiquettent les blocs dans le cache:
  - T_c dernière validation de l'entrée du cache
  - T_m dernière modification du bloc sur le serveur
- Un bloc du cache à la date T est valide si $T - T_c < t$ (t: intervalle de rafraîchissement) ou si la date T_m enregistrée sur le client est la même que celle présente sur le serveur.
- La valeur de t est choisie pour faire un compromis entre l'efficacité et la consistance. Pour Sun Solaris: $3s \leq t \leq 30s$ pour les fichiers $30s \leq t \leq 60s$

Cache du client

- Pour optimiser le polling:
  - L'arrivée d'un nouveau T_m est appliqué à tous les blocs du fichier dans le cache
  - Les attributs d'un fichier sont envoyé avec les résultats de toutes les requêtes (*piggybacked*)
  - La valeur de t est adaptée dynamiquement
- Pour la consistance du cache lors de l'écriture:
  - Les blocs sont marqués comme *dirty*, il seront écrits dans le fichier distant de manière asynchrone (fermeture du fichier ou `sync`).
  - Le *read-ahead* et *delayed write* peuvent être améliorés à l'aide d'un *bio-deamon*.

Andrew file system

- Conçu pour être plus résistant au passage à l'échelle (plus de 1000 machines).
- Points clés pour la performance:
  - Service de fichiers entiers (< 64kB)
  - Cache de fichier entier (cache jusqu'à 100 Mb)
  - Fichiers peu fréquemment mis à jour dupliquées localement (librairies unix)
  - Hypothèses sur les tailles moyennes des fichiers accédés
    - petits fichiers (moins de 10kB)
    - Plus de `read` que de `write`
    - accès séquentiel
    - fichiers utilisés par un seul utilisateur
  - mécanisme de `promise` et `callback` pour la cohérence

Principe du *promise* et *call back*

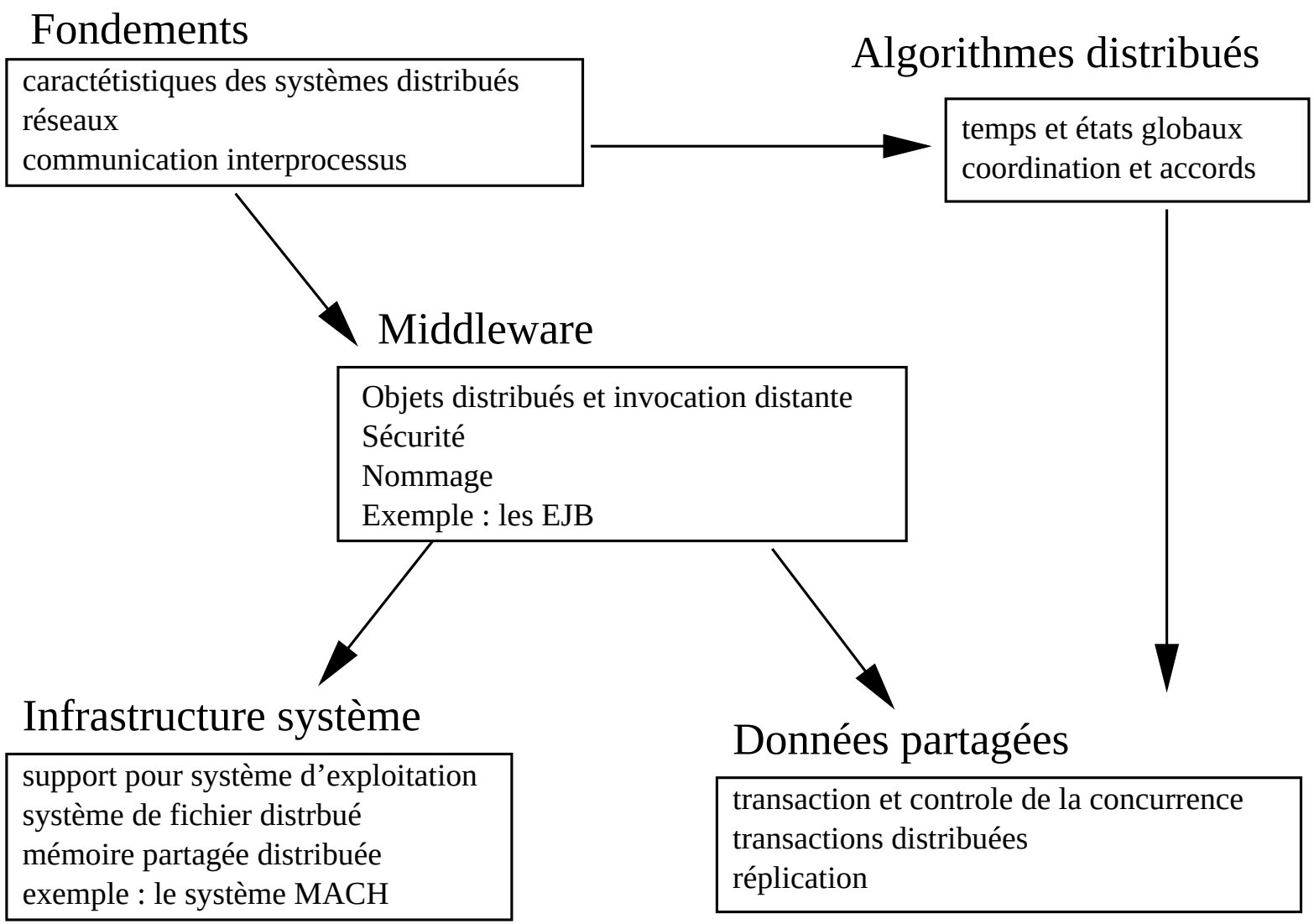
- Lorsque le serveur envoi un fichier au client, il envoi un $callback - promise$ il avertira lorsqu'un autre client modifiera le fichier.
- Lorsque le fichier est modifié, le serveur envoi un *callback* à toutes les copies du fichier
  $\Rightarrow callback - promise \leftarrow cancelled$
- Lorsque le client veut ouvrir le fichier il vérifie que le $callback - promise$ est valide.
- Après un crash du client, le serveur peut renvoyer les fichiers correspondant aux tags $callback - promise$ valides du client.
- Plus extensible qu'un mécanisme basé sur les time-stamp
- Approximation de la *one copy update semantic*

Conclusion générale

- Virtualisation du calcul
  - indépendance du réseau
  - indépendance du système
- Virtualisation du stockage
  - indépendance du stockage
    - protocoles iSCSI et iFCP
  - indépendance des protocoles
    - systèmes de fichiers distribués
- Développement de services “MiddleWare” pour abstraire la distribution sans modifier les systèmes de façon trop importante

  $\Rightarrow$ suite du cours

Plan


Fondements
caractétistiques des systèmes distribués
réseaux
communication interprocessus
Algorithmes distribués
temps et états globaux
coordination et accords
Middleware
Objets distribués et invocation distante
Sécurité
Nommage
Exemple : les EJB
Infrastructure système
support pour système d'exploitation
système de fichier distrbué
mémoire partagée distribuée
exemple : le système MACH
Données partagées
transaction et controle de la concurrence
transactions distribuées
réplication