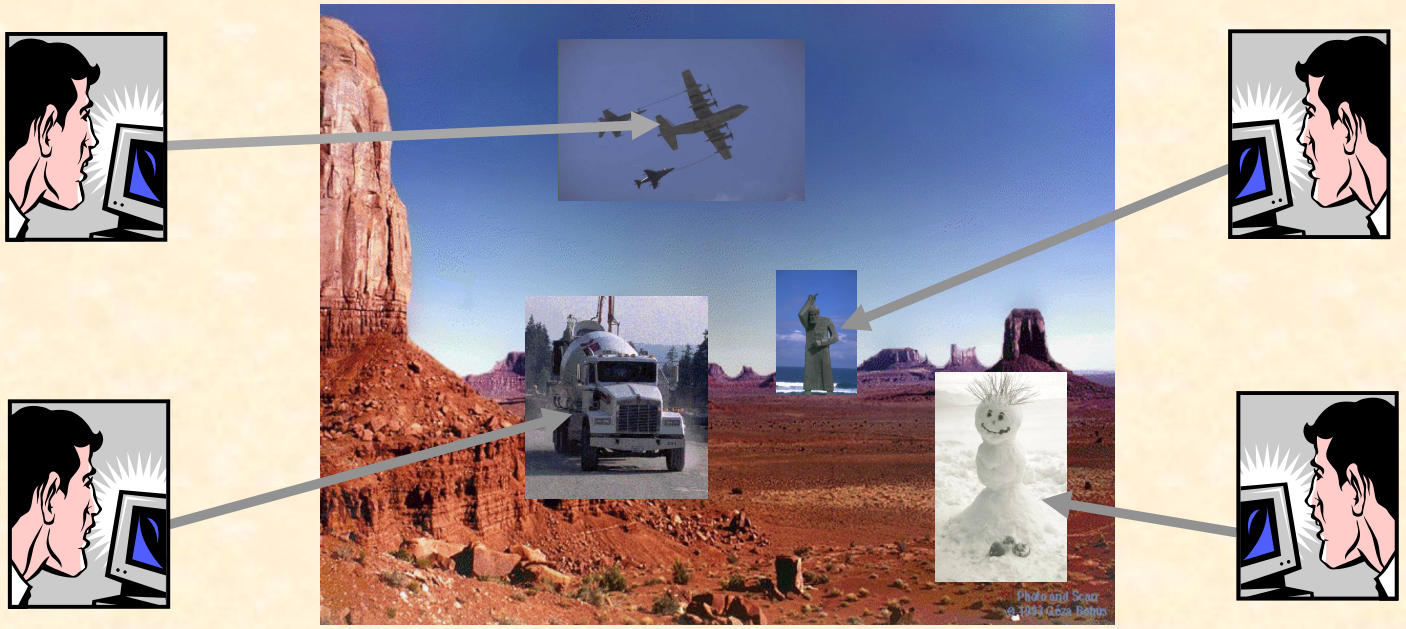
# Réalité Virtuelle Distribuée
# Historique & Architectures

***Patrice Torguet***

***IRIT Equipe VORTEX***

# *Introduction*

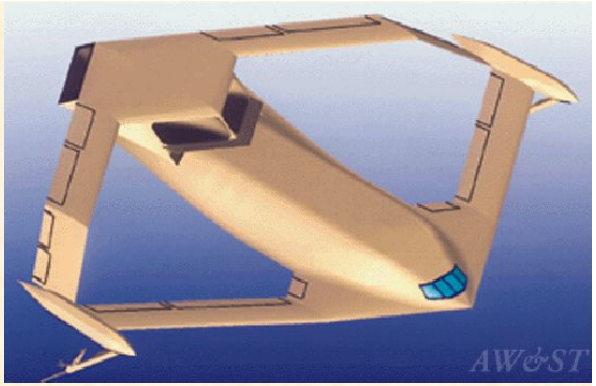
- Définitions
  - Réalité Virtuelle (RV) : **techniques** et **outils** qui ont pour but de **faire croire** à une personne qu'elle est réellement présente dans un **monde synthétique**
  - Réalité Virtuelle Distribuée (RVD) : **plusieurs** personnes connectées en réseau **partagent** virtuellement le même monde synthétique



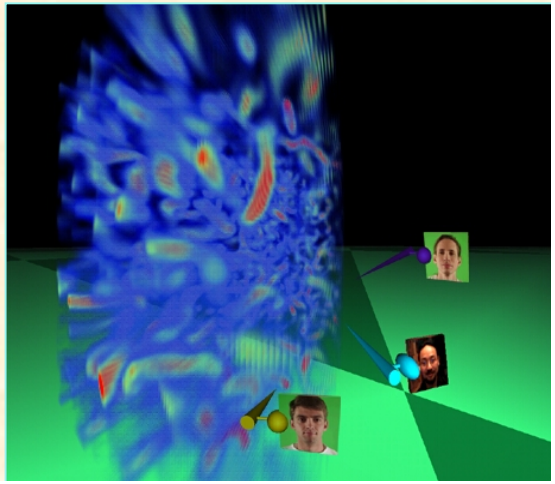
# *Introduction*

- Synonymes :
  - Réalité Virtuelle Distribuée (DVR)
  - Environnements Virtuels Collaboratifs (CVE)
  - Environnements Virtuels en Réseau (NVE)
  - Mondes Virtuels en Réseau (NVW)
  - Mondes Virtuels Collaboratifs (CVW)
  - Mondes Virtuels Partagés (SVW)

# *Applications sérieuses*

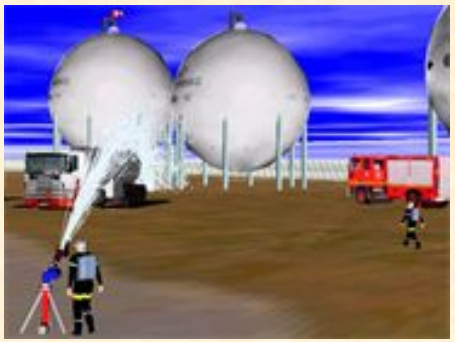


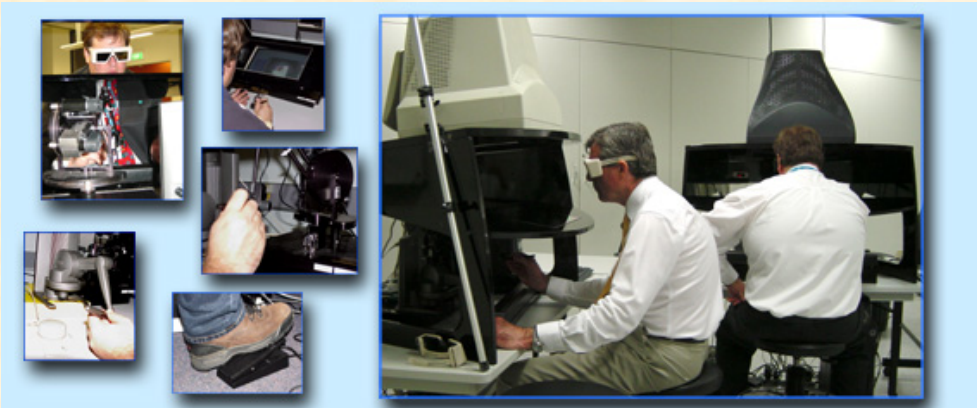

- Prototypage virtuel collaboratif
  - Permettre a plusieurs ingénieurs de travailler sur une même maquette virtuelle
  - Exemples :
    - ✓Utilisé par les participants du CSA (Common Support Aircraft) pour l'Armée Américaine
    - ✓Utilisé lors de la conception de l'Airbus A380
- Visualisation scientifique collaborative
  - Travail collaboratif sur de la visualisation de données scientifiques
  - Exemples :
    - ✓Visualisation collaborative de protéines et autres molécules (Université de Buffalo)
    - ✓Visualisation collaborative de réservoirs de pétrole (Université du Nord de l'Arizona)
    - ✓Visualisation de données volumiques (Université de l'Illinois à Chicago)



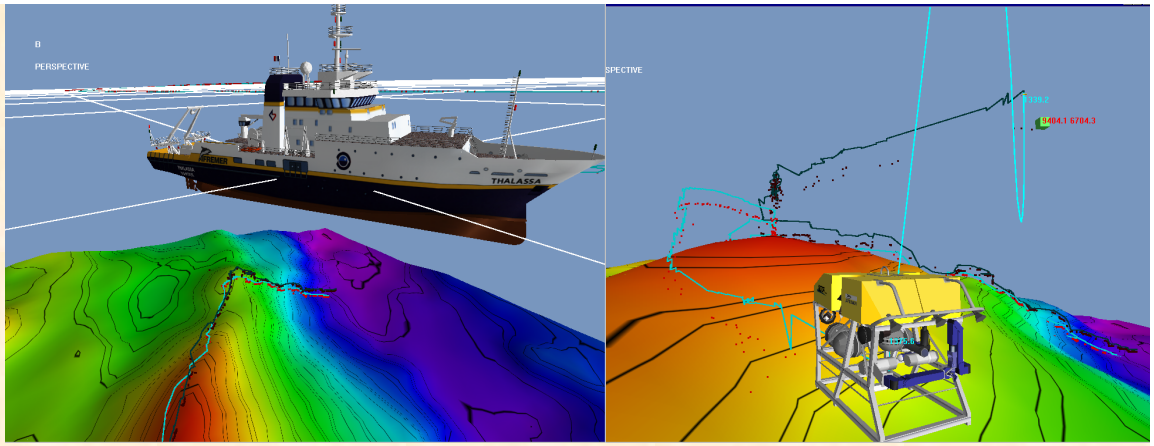
# *Applications sérieuses*


- Simulation militaire
  - Entraînement des militaires
  - Exemples :
    - ✓Simulateurs de vols en formation (Aéronavale)
    - ✓Entraînement multi-arme (CCTT)
- Simulation civile
  - Entraînement des civils
  - Exemples :
    - ✓Entraînement de pompiers (ENSTB)
    - ✓Entraînement de chirurgiens (ICT Australie)

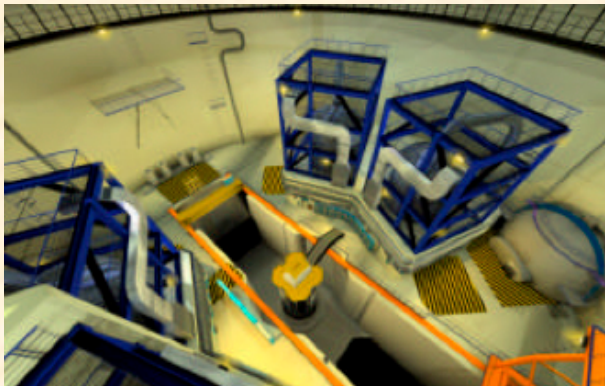




# *Applications sérieuses*




- Téléopération
  - Pilotage de robots dans des endroits distants et/ou risqués
  - Exemples :
    - ✓pilotage de sous-marin automatique au fond de la mer (Ifremer)
    - ✓pilotage d'un robot nettoyeur dans une centrale nucléaire (EDF)
    - ✓pilotage semi-automatique d'un robot sur Mars (NASA)
- Téléprésence
  - Amener l'expertise d'un être humain à un autre endroit
  - Exemple :
    - ✓expertise médicale lors d'une opération chirurgicale ou une intervention urgentiste (British Telecom)



# *Applications ludiques*

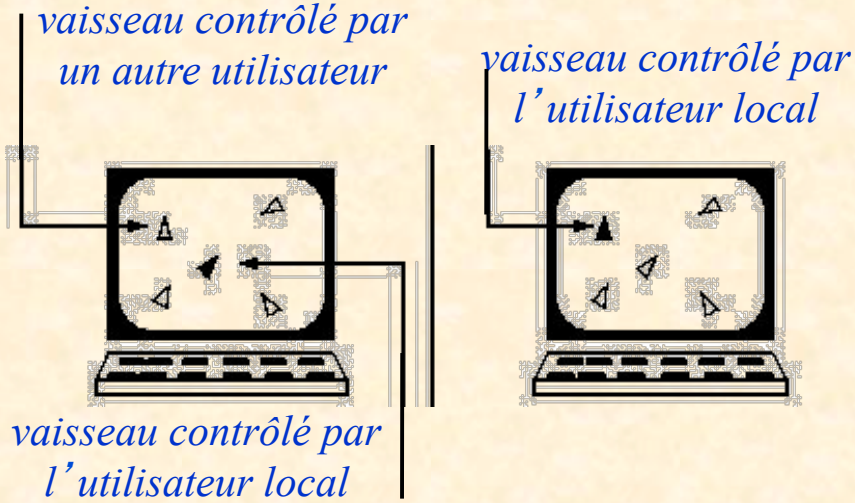
- Jeux multi-joueurs
  - Jeux de tirs à la première personne (FPS)
    - ✓Doom, Quake, Unreal Tournament...
  - Jeux de stratégie temps-réel (RTS)
    - ✓Warcraft, Starcraft, Age of Empires...
  - Jeux de rôle (RPG)
    - ✓Neverwinter Nights, Final Fantasy...
  - Jeux de simulations sportives
    - ✓Pro Evolution Soccer, Need for Speed...
  - Jeux massivement multi-joueurs
    - ✓Ultima Online, Everquest, World of Warcraft...
  - ...

*Plan du cours*

- Architectures
  - Architectures réparties
  - Gestion de la cohérence
  - Optimisation par gestion de l'intérêt
- Spécificités des jeux en réseau
  - FPS
  - RTS
  - MMORPG

## *Un petit exemple pour commencer*

- Exemple simple (2D) pour illustrer le cours
- Chaque utilisateur contrôle un petit vaisseau (représenté par un triangle)
- Visualisation des vaisseaux contrôlés par les autres utilisateurs




- Ecrit en Java pour la simplicité

- MUVE (multi user virtual environment)

- Disponible à l'URL suivante :

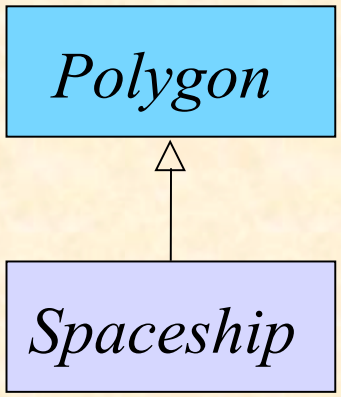
http://torguet.net/cours/

## *Un petit exemple pour commencer*

- Utilise les Sockets BSD avec le protocole de transport UDP multicast
  - Format des paquets
    - ✓un identificateur unique de vaisseau
    - ✓la position et l'orientation du vaisseau
  - Sémantique des paquets
    - ✓A chaque fois qu'un vaisseau bouge dans une application on envoie un paquet à toutes les autres applications
    - ✓Lors de la réception du premier paquet d'un vaisseau on crée le triangle qui va représenter ce vaisseau
    - ✓Lors de la réception des paquets suivants d'un même vaisseau on met à jour la position et l'orientation de ce vaisseau
  - Pas de gestion d'erreur

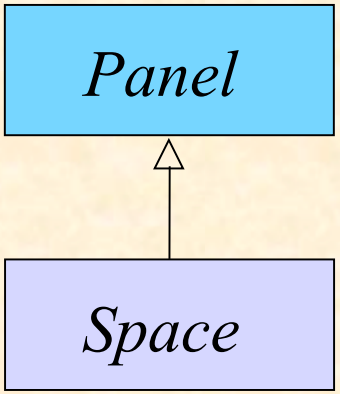
# *Un petit exemple pour commencer*

- Les différentes classes



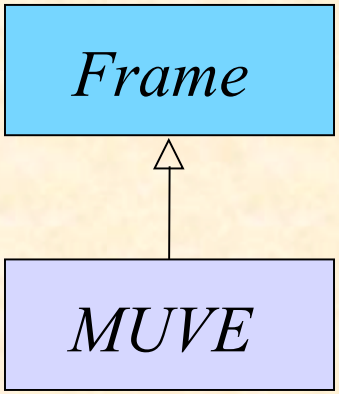

***Attributs :***
*id*
*x, y, velocity, currentAngle*
*oldX, oldY, oldAngle*

***Méthodes :***
*Spaceship(id)*
*translate(x,y)*
*rotate(angle)*
*update(time)*
***sendNetworkUpdate()***




***Attributs :***
*local_ship, spaceships*
*lastTime*
*socket, internetAddress (ia)*

***Méthodes :***
*Space(id)*
***connectToNetwork()***
*updateLocalShip()*
*paintComponent()*
***run()***




***Attributs :***
*space*

***Méthodes :***
*MUVE(id)*
*run()*
*initComponents()*
*myKeyPressed()*
*exitForm()*
*main()*

## *Un petit exemple pour commencer*

- La classe Spaceship
  - méthode void sendNetworkUpdate()

```
// si rien n'a été modifié on n'envoie rien (Sémantique des paquets)
    if ((oldAngle == currentAngle) && (oldX == x) && (oldY == y))
      return;
    // mise à jour des valeurs précédentes
    oldAngle = currentAngle; oldX = x; oldY = y;

// on place les valeurs à envoyer dans un tampon (Format des paquets)
    ByteArrayOutputStream stream = new ByteArrayOutputStream(40);
    DataOutputStream      dos    = new DataOutputStream(stream);
    dos.writeInt(id); dos.writeInt(x); dos.writeInt(y); dos.writeDouble(currentAngle);
// on crée un paquet contenant ce tampon
    DatagramPacket dp = new DatagramPacket(stream.toByteArray(),
        stream.toByteArray().length, Space.ia, 2000);
// on envoie le paquet sur le réseau
    Space.socket.send(dp);
```

## *Un petit exemple pour commencer*

- La classe Space
  - méthode void connectToNetwork()

```
// on utilise ici le multicast
// crée un socket sur un port choisi
      socket = new MulticastSocket(2000);

// s'abonne à un groupe multicast choisi
      ia = InetAddress.getByName("224.11.4.2");
      socket.joinGroup(ia);

// crée et démarre le thread qui recevra les messages réseaux
      Thread networkListener = new Thread(this);
      networkListener.start();
```

## *Un petit exemple pour commencer*

- La classe Space
  - méthode void run()

```
// le tampon qui recevra les mesages
    byte buffer[] = new byte[256];
    DatagramPacket dp = new DatagramPacket(buffer, buffer.length);

while (true) {
   // attends un paquet
      socket.receive(dp);
  // décortique le paquet (Format des paquets)
      ByteArrayInputStream  stream = new ByteArrayInputStream(buffer);
      DataInputStream       dis    = new DataInputStream(stream);
  // on récupére d'abord l'id du vaisseau
      int id = dis.readInt(); Integer theId = new Integer(id);
```

# *Un petit exemple pour commencer*

- La classe Space
  - méthode void run()  (suite)

```
// (Sémantique des paquets)
// on cherche le vaisseau dans l'ensemble des vaisseaux connus
      Spaceship sp = (Spaceship) spaceships.get(theId);

// si le vaisseau n'existe pas et que ce n'est pas le vaisseau local on le crée localement
      if ((sp == null) && (id != local_ship.id)) {
        sp = new Spaceship(id);
        spaceships.put(theId, sp);
          }
// si le vaisseau existe on le met à jour
      if (sp != null) {
        // on met à jour les valeurs (Format des paquets)
        sp.x = dis.readInt();
        sp.y = dis.readInt();
        sp.currentAngle = dis.readDouble();
        // on appelle rotate qui va positionner correctement le vaisseau
        sp.rotate(0);
      }
} // fin du tantque
```

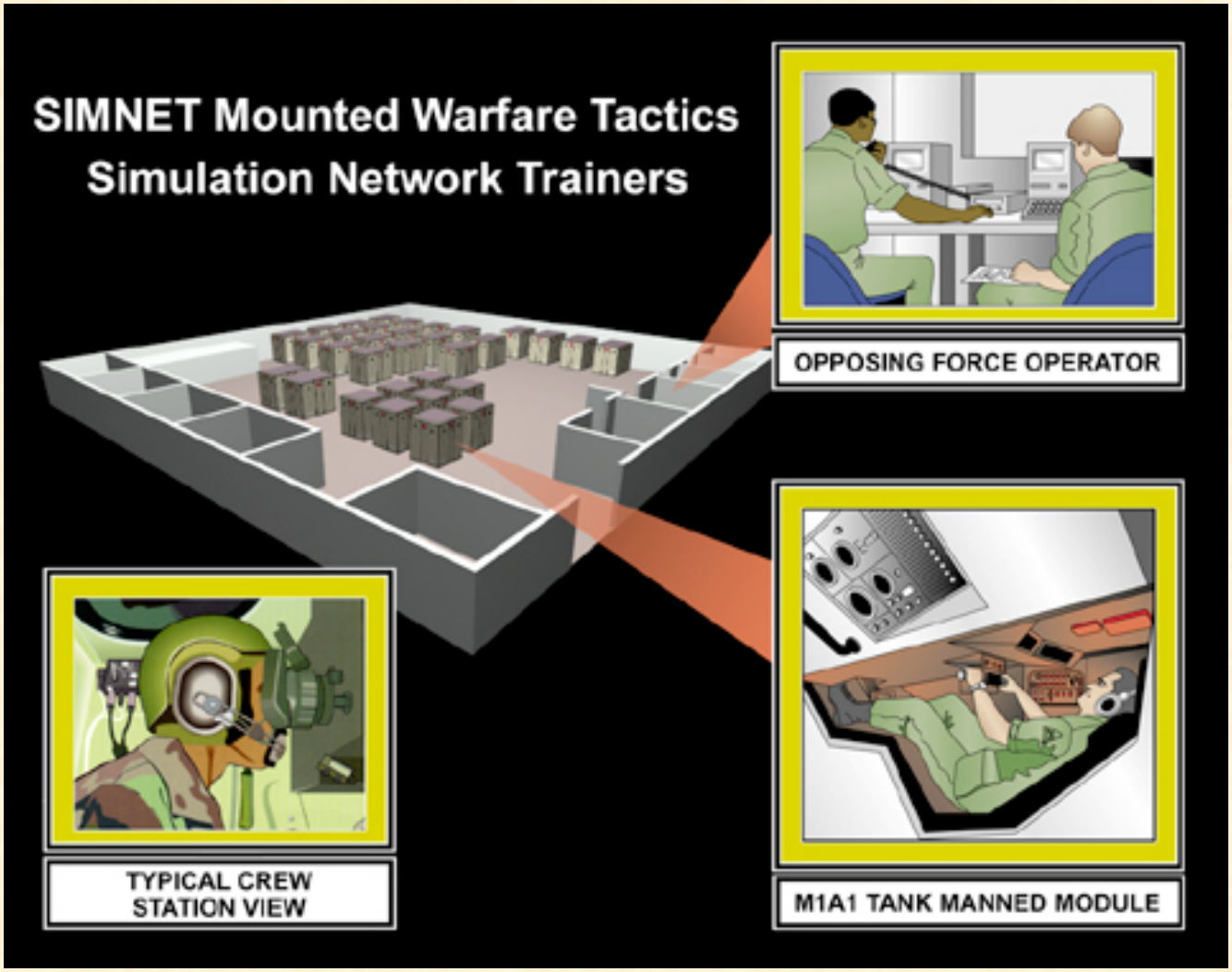
# *Historique*

- Applications militaires
  - SIMNET
  - DIS
- Applications ludiques
  - MUD1 et les MUDs/MOOs/MUSHs
  - SGI Flight & Dogfight
  - Lucasfilm Habitat
  - Doom
  - Warcraft
  - Ultima Online

# *SIMNET (simulator networking)*

- SIMNET est un environnement virtuel réparti développé pour DARPA par BBN (Bolt, Beranek and Newman), Perceptronics et Delta Graphics.
- Les travaux ont débuté en 1983 et le système fut opérationnel pour l'armée à partir de fin mars 1990.
- Spécifications
  - Simuler de 100 à 100 000 entités (simulateurs hardware).
  - Multi-sites (géographiques).
  - Simulations hétérogènes (simulateurs différents).
  - Peu coûteux / coût d'une simulation grandeur nature.
  - Répartition totale (pas de site central).
  - Fonctionne en temps réel.

# *SIMNET (simulator networking)*



## *SIMNET (simulator networking)*

- Problèmes clefs du projet
  - Fabriquer des simulateurs de haute qualité à faible coût
  - Les relier à travers un réseau pour créer un champ de bataille virtuel cohérent.

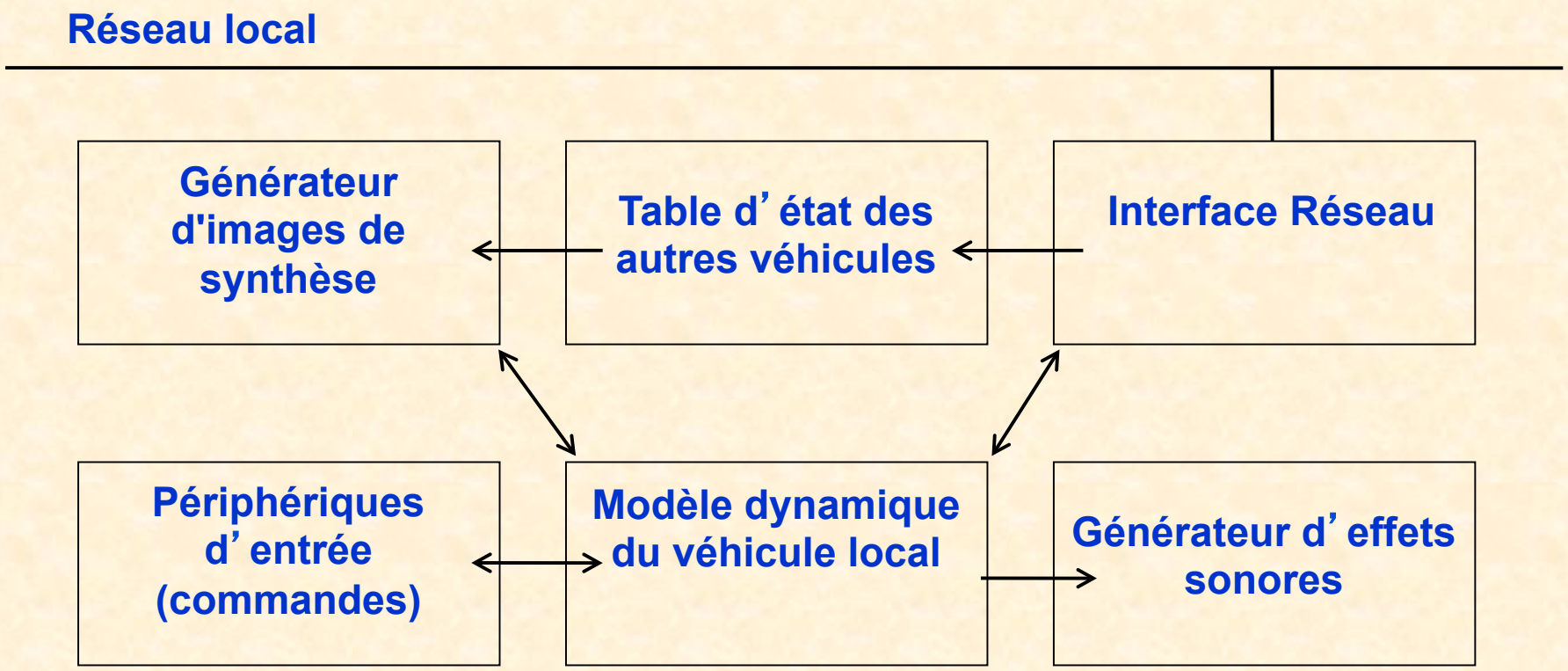
- Pour étudier ces problèmes, le projet SIMNET a mis en place un ensemble de test comprenant 11 sites avec entre 50 et 100 simulateurs sur chaque site.

## *SIMNET (simulator networking)*

- SIMNET pouvait être rejoint depuis n'importe quel endroit du réseau en utilisant un simulateur comme un portail vers le monde synthétique. Une fois dans ce monde synthétique, l'utilisateur pouvait interagir avec les autres utilisateurs présents dans le champ de bataille virtuel.

- L'interaction dans cet environnement était totalement libre (non scriptée) dans les limites de la chaîne de commandement imposée aux participants de la simulation (ordres donnés aux militaires).

# *Architecture de SIMNET*



# *SIMNET (simulator networking)*

- L'architecture logicielle est basée sur trois composants :
  - Une architecture objet-événement,
  - La notion de nœud de simulation autonome,
  - Un ensemble d'algorithmes prédictifs appelés "dead reckoning"
- L'architecture objet-événement modélise le monde par une collection d'objets dont les interactions sont représentées par une collection d'événements.
  - Les objets sont les véhicules et les systèmes d'armes qui interagissent à travers le réseau.
  - Les événements sont des messages issus à travers le réseau pour indiquer un changement de l'état du monde ou d'un des objets.

## *SIMNET (simulator networking)*

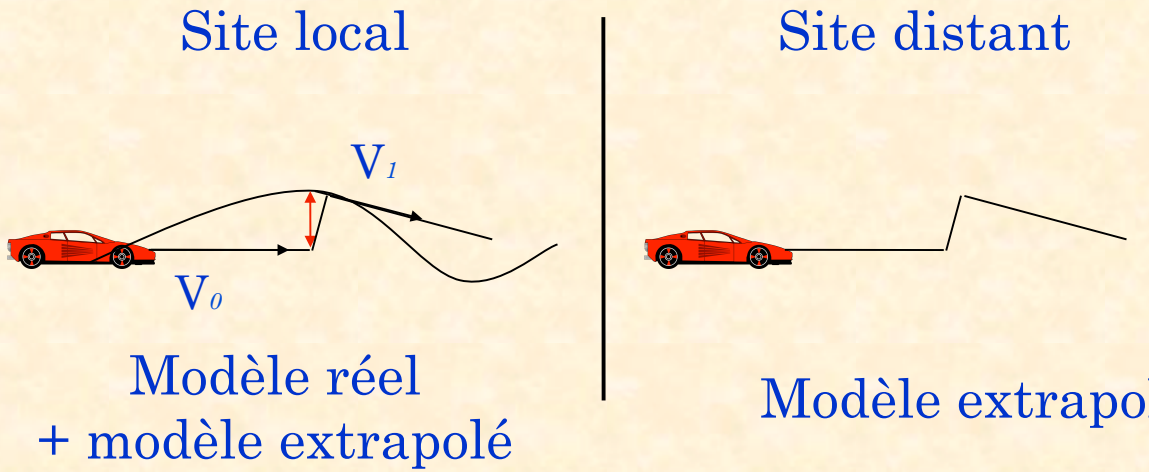
- La notion de nœud de simulation autonome de SIMNET signifie que les participants (véhicules et systèmes d'armes) doivent envoyer des messages réseau pour représenter de façon précise leur état courant (Ils ne mentent pas sur leur état).
- Les récepteurs de tels messages doivent recevoir les informations de changement d'état et les utiliser pour mettre à jour le modèle local du monde.
- Un nœud doit envoyer des messages de mise à jour à chaque fois que les objets changent (mouvements…).

## *SIMNET (simulator networking)*

- Le système ne disposant pas de serveur central, il est plus résistant aux pannes. Si un nœud de simulation s'arrête de fonctionner la simulation continue avec les autres nœuds.
- Les participants peuvent arriver et partir à n'importe quel moment.
- Les nœuds doivent émettre régulièrement des messages appelés « heartbeats » (toutes les 5 secondes en général) de façon à informer les autres participants que l'objet local existe encore dans le système (et doit donc être affiché à l'écran)

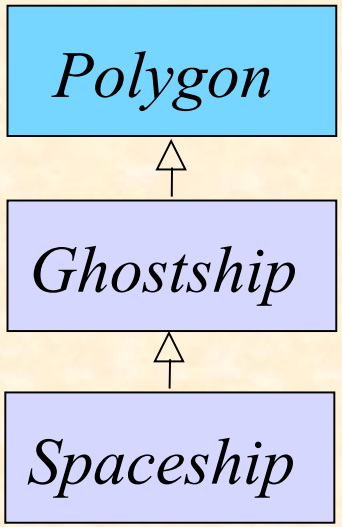
# *SIMNET (simulator networking)*

- Le « Dead-Reckoning »
  - Paradigme objet-fantôme : les objets ne doivent envoyer des messages que lorsque les autres simulateurs du réseau ne peuvent plus prédire fidèlement leur état.
  - Tous les simulateurs gèrent des copies fantômes de tous les objets de façon à pouvoir prédire leur état courant grâce à un ensemble d'information (vitesse et position par exemple).



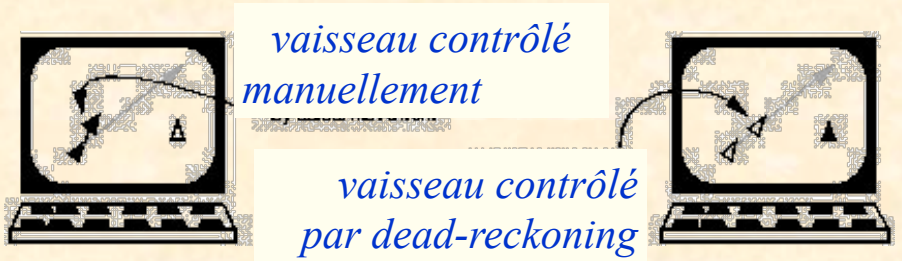
# *Retour sur l'exemple*

- Ajout du dead-reckoning




***Attributs :***
*id, velocity,*
*x, y, currentAngle*

***Méthodes :***
*Ghostship(id)*
*translate(x,y)*
*rotate(angle)*
*update(time)*




***Attributs :***
*ghost, maxDistance2*

***Méthodes :***
*Spaceship(id)*
***update****(time)*
***sendNetworkUpdate()***

# *Retour sur l'exemple*

- La classe Spaceship
  - méthode void update(double time)

```
// déplacer le vaisseau
…
// déplacer son fantôme
ghost.update(time);
```

méthode void sendNetworkUpdate()

```
// on compare la distance entre le fantôme et le vaisseau à la distance max. acceptée
    if ((x - ghost.x)*(x - ghost.x) + (y - ghost.y)*(y - ghost.y) < maxDistance2)
      return;
    // mise à jour du fantôme
        ghost.x = x; ghost.y = y; ghost.currentAngle = currentAngle; ghost.velocity = velocity;
// on place les valeurs à envoyer dans un tampon
    ByteArrayOutputStream stream = new ByteArrayOutputStream(40);
    DataOutputStream      dos    = new DataOutputStream(stream);
    dos.writeInt(id); dos.writeInt(x); dos.writeInt(y);
    dos.writeDouble(currentAngle); dos.writeInt(velocity);
// on crée un paquet contenant ce tampon
    DatagramPacket dp = new DatagramPacket(stream.toByteArray(),
        stream.toByteArray().length, Space.ia, 2000);
// on envoie le paquet sur le réseau
    Space.socket.send(dp);
```

## *Retour sur l'exemple*

- La classe Space
  - méthode synchronized void paintComponent(Graphics g)

```
// affichage du vaisseau local
    …
// on affiche les autres vaisseaux après les avoir mis à jour (avec le DR)
    Enumeration ships = ghostships.elements();
    while(ships.hasMoreElements()) {
          Ghostship sp = (Ghostship)(ships.nextElement());
     sp.update(time - lastTime);
     g.drawPolygon(sp);
    }
    méthode void run()

 ...
  // on cherche le vaisseau dans l'ensemble des vaisseaux connus
       Ghostship sp = (Ghostship) ghostships.get(theId);
  // si le vaisseau n'existe pas et que ce n'est pas le vaisseau local on crée un fantôme
       …
  // si le vaisseau existe on le met à jour
       if (sp != null) {
         // on met à jour les valeurs
         sp.x = dis.readInt(); sp.y = dis.readInt();
         sp.currentAngle = dis.readDouble(); sp.velocity = dis. readDouble();
         ...
```

## *SIMNET (simulator networking)*

- SIMNET et l'extensibilité
  - SIMNET a été capable de gérer 850 objets répartis sur 5 sites lors d'un exercice en Mars 1990 (la plupart des objets étaient des SAF - forces semi-automatisées).
  - Les objets envoyaient en moyenne un paquet de 156 octets toutes les secondes en atteignant une bande passante maximale de 1.06 Mbits/seconde (juste au dessous de celle des liens T-1 utilisés).

## *DIS (Distributed Interactive Simulation)*

- DIS est le successeur de SIMNET mais il est plus générique
- Il possède les 3 mêmes composants de base :
  - une architecture objet-événement
  - nœuds de simulations totalement distribués
  - ensemble d'algorithmes prédictifs pour le « dead reckoning »
- Le cœur de l'architecture logicielle est le PDU (protocol data unit - unité de donnée de protocole).
- La clef de l'architecture est de déterminer quand un véhicule (nœud) doit envoyer un PDU.

## *DIS (Distributed Interactive Simulation)*

- Le standard DIS (IEEE 1278) défini 27 PDU différents dont seulement 4 (Etat de l'entité, Feu, Détonation et Collision) sont utilisés par les véhicules pour interagir avec leur environnement virtuel.
- En fait, la plupart des simulations dites compatibles DIS, ne gèrent que ces 4 PDUs et soit rejettent silencieusement les 23 autres PDUs soit émettent un bref message d'erreur indiquant qu'un PDU non géré a été reçu.

## *DIS (Distributed Interactive Simulation)*

- Lors d'une démonstration à la conférence I/ITSEC'93 (Interservice/Industry Training and Education Conference) on a calculé que 96% du trafic total de DIS correspondait à l'échange de PDU Etat de l'entité (ESPDU).
- Les 4% restant se répartissent en : Transmetteur (50%), Emission (39%), Feu (4%) et Détonation (4%).
- La simulation comprenait 79 participants.
- En moyenne, les aéronefs ont envoyé 1 ESPDU/s et les véhicule terrestre 0,17 ESPDU/s. Certains participants ont émis jusqu'à 20 ESPDU/s.

## *DIS (Distributed Interactive Simulation)*

- DIS permet à n'importe quel type d'ordinateur pourvu d'une carte réseau et qui peut lire et écrire des PDU DIS et gérer l'état de ces PDU correctement de faire partie intégrante de la simulation.
- Une telle architecture hétérogène implique que des simulateurs, des stations de travail et même des PC peuvent s'opposer dans une même simulation.
- Il y a eu plusieurs utilisations de DIS avec plus de 300 à 500 participants (ce qui est prévu par DIS).
- Néanmoins, dans la plupart des cas la norme est en fait modifiée pour arriver à créer des simulations efficaces.

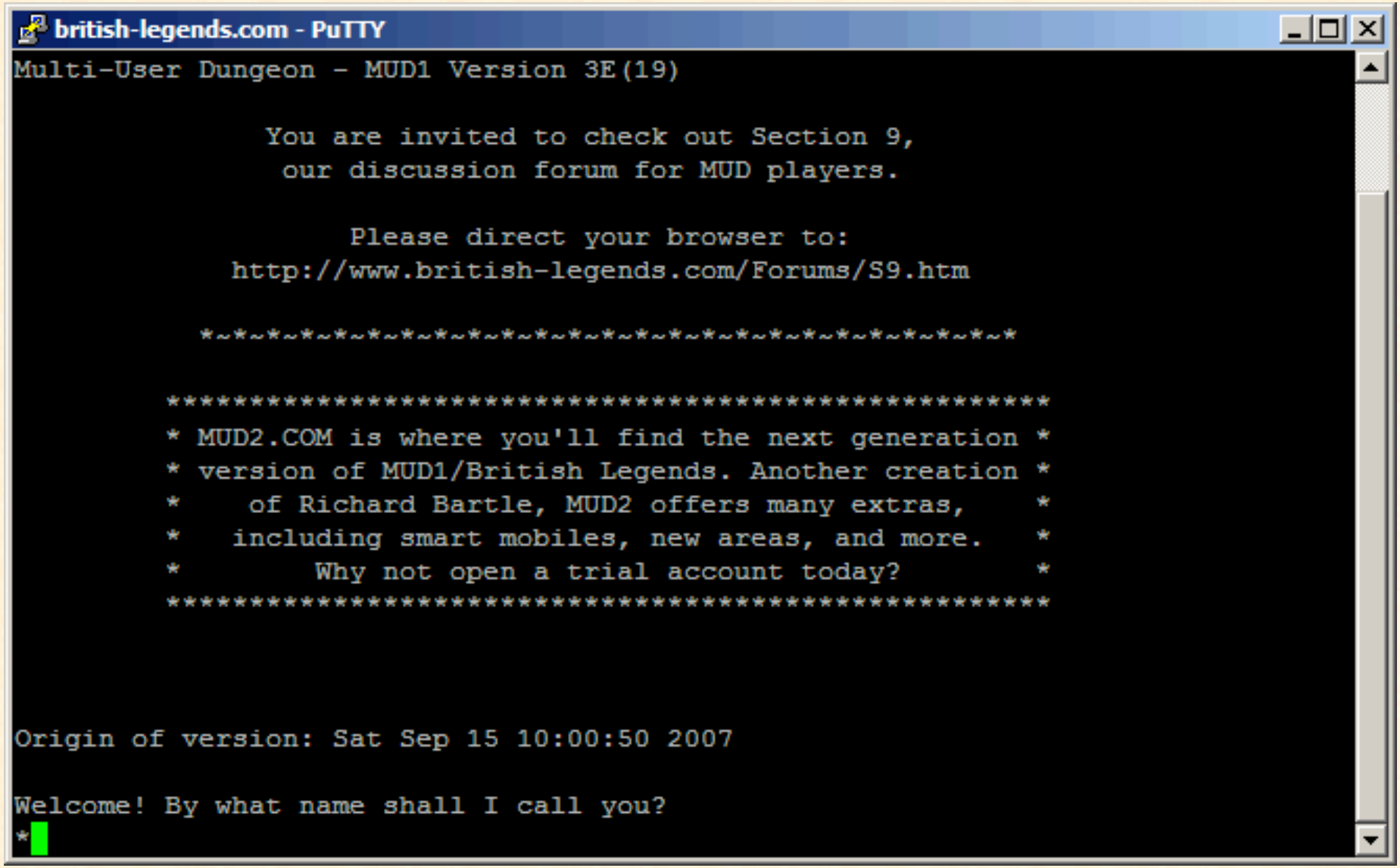
## *Exemple d'utilisation de DIS : le CCTT*

- Le centre d'entraînement tactique au combat rapproché (Close Combat Tactical Trainer) de l'armée américaine est l'un des environnements virtuels les plus peuplés.

## *MUD1 et les MUDs*

- En 1978, Roy Trubshaw et Richard Bartle programment, en assembleur MACRO-10 sur un super ordinateur DecSystem-10 de l'université d'Essex, un jeu en mode texte multi-utilisateur appelé MUD (Multi User Dungeon)
- C'est un jeu en mode texte qui permet de se déplacer dans un monde virtuel, de ramasser des objets, de tuer des monstres et les personnages des autres utilisateurs (et aussi de discuter avec eux...)
- De nombreux autres jeux du même type ont été développés entre 1978 et aujourd'hui

# *MUD1 et les MUDs*



## *SGI Flight & Dogfight*

- Gary Tarolli de la société Silicon Graphics est probablement le premier créateur d'environnement virtuel distribué.
- Il a programmé, en 1983, la première version de Flight, démo d'un simulateur de vol.
- Les aspects répartis ont été ajouté petit à petit à partir de 1984.
- La première version utilisait un câble série entre 2 stations de travail SGI. Il fonctionnait à environ 7 images/s sur un 68000 (avec à peu près 500 polygones par seconde)

## *SGI Flight & Dogfight*

- Flight fut amélioré en utilisant le multicasting XNS sur Ethernet pour le SIGGRAPH de 1984. Elle fut ensuite livrée avec toutes les SGI.
- Au début de 1985, les ingénieurs de SGI ont ensuite transformé Flight en Dogfight en permettant aux joueurs de se tirer les uns sur les autres.
- Ils ont ainsi probablement créé le premier monde virtuel en réseau interactif.
- Problème : des paquets étaient (sont ?) émis à chaque tour de boucle => encombrement du réseau.

## *Lucasfilm Habitat*

- Premier monde virtuel commercial en réseau (1986)
- Chatworld (monde où l'on bavarde) en 2D (MUDs, MOOs, MUSHs)
- Développé pour Lucasfilm Games et Quantum Computer Services (AOL)
- Fonctionnait au début sur des Commodore 64
- Les utilisateurs en phase de test payaient 0.08 $/minute (1 personne a dépensé jusqu'à 1000 $ en un mois ~ 3,5 jours « online »)
- 15 000 utilisateurs sur une base de 100 000 (utilisateurs de BBS)

# *DOOM*

- Le 10 décembre 1993, Id Software a sorti la version shareware de DOOM
- Ce jeu est probablement à l'origine de tous les jeux en réseau actuels
- Tout comme flight et dogfight il utilisait une bande passante réseau énorme en envoyant des messages à la vitesse de rendu
- On estime que la version shareware de Doom a été récupérée 15 million de fois sur Internet et/ou passée de personne à personne

## *Warcraft*

- Dune II (1992) est le premier jeu de stratégie temps-réel (RTS) contrôlé à la souris.


- Warcraft (1994) est le premier RTS a introduire un mode multi-joueur (modem, connexion directe RS-232 ou réseau local IPX).

# *Ultima Online et les MMORPG*


- Ultima Online est un descendant direct des jeux de type MUD ou Habitat.
- En 1997, il est le premier a les avoir fait passer à une échelle réellement massive en drainant 100 000 joueurs payants ($9.95 par mois) en un an (soit un revenu de 12 million de dollars par an en comptant les ventes des boîtes du jeu).

# *World of Warcraft*

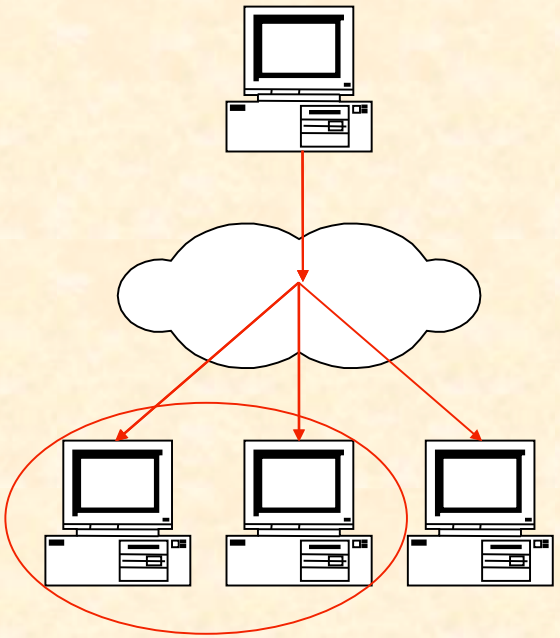
- World of Warcraft, sorti en 2004 (10 ans après Warcraft 1), est devenu le MMO le plus joué au monde avec plus de 10 millions de joueurs réguliers (22/01/08).
- Cela dit, chaque serveur (en fait groupe de serveurs) ne gère que ~5000 joueurs simultanément.

# *EVE Online*

- EVE Online est actuellement le MMO le plus massif car il gère plus de 41000 utilisateurs simultanément sur le même serveur (en fait groupe de serveur)
- L'interactivité est cependant assez limitée.

# *Architecture*

- Modèle de communication
  - diffusion
  - point à point
  - groupes de communication (multicasting)



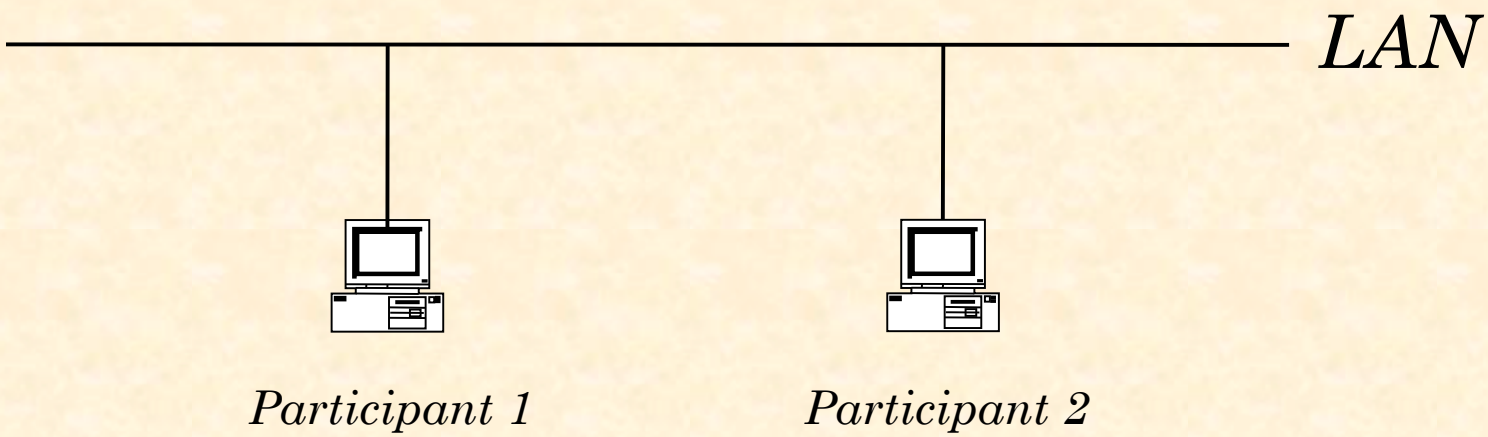
# *Architecture*

- Architecture répartie
  - Systèmes sans serveur
    - ✓Egal à Egal (Peer to Peer P2P)
    - ✓Groupes de communication (Multicast)
  - Systèmes centralisés (Client/Serveur C/S)
  - Systèmes à plusieurs serveurs
  - Systèmes à plusieurs serveurs coordonnés

- Choix de l'architecture

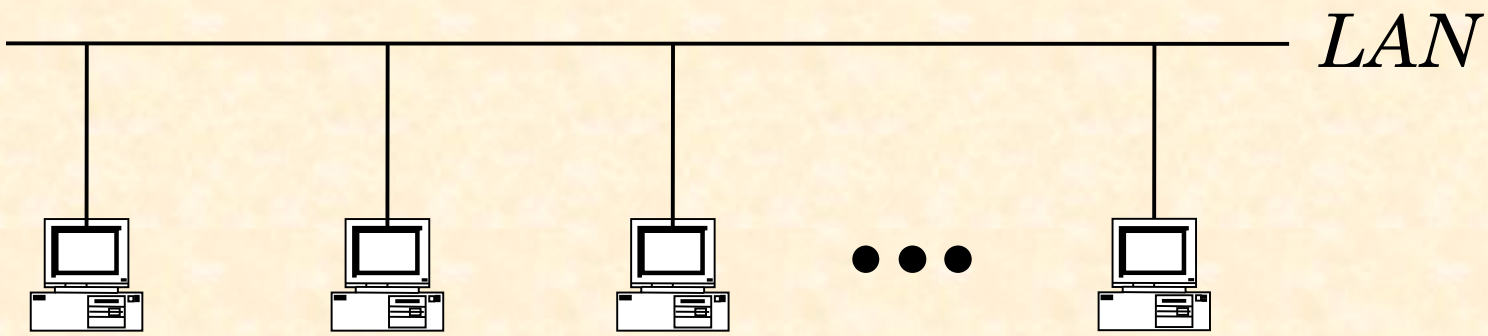
## *Systèmes sans serveur : Egal à Egal*

- Cas simple : 2 participants sur un réseau local
  - Chaque participant communique directement son état à l'autre participant



## *Systèmes sans serveur : Egal à Egal*

- Cas général : Chaque participant peut communiquer avec tous les autres grâce à une simple diffusion



## *Systèmes sans serveur : Egal à Egal*

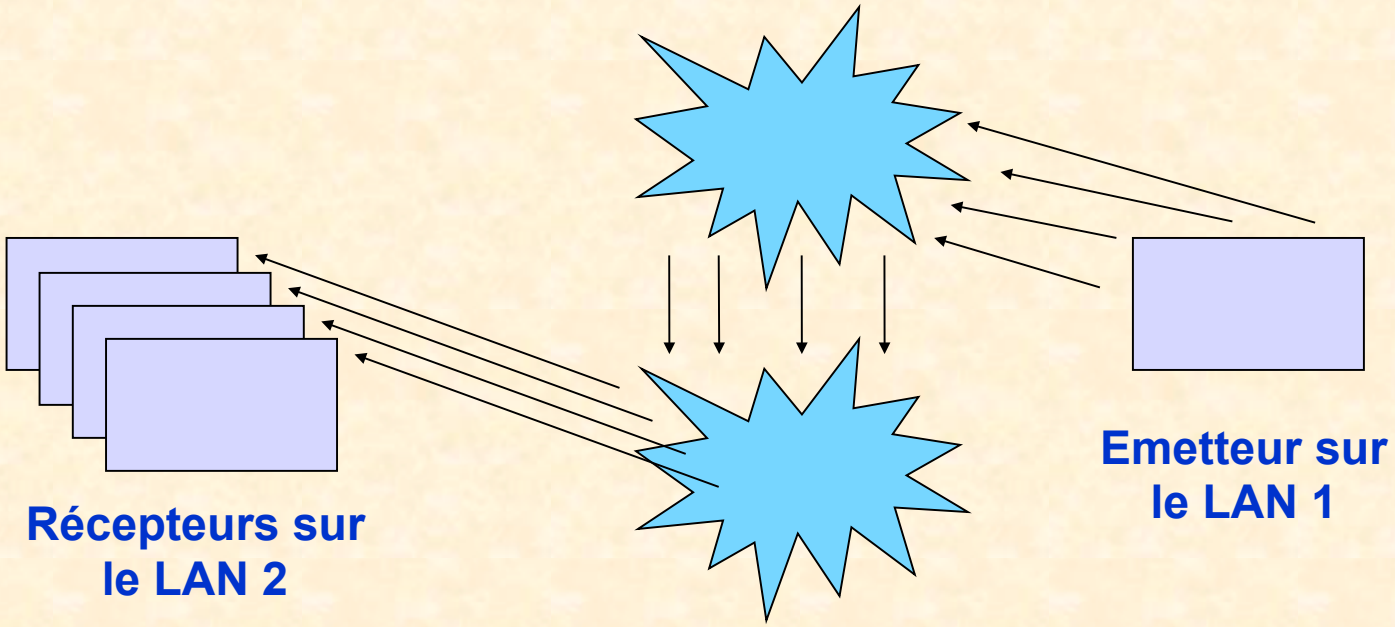
- Exemple Réseau Local Ethernet 10 Mbps (100 Mbps)
  - Bande passante disponible : ~ 7 Mbps (80 Mbps)
  - Ethernet sature pour une utilisation de plus de 70%
- Exemple de paquet standard le ESPDU de DIS
  - 144 octets de long (1152 bits)
- Cas le plus défavorable
  - la limite est de 6000 PDU / seconde pour 7 Mbps (70000)
  - si on suppose un taux par participant de 30 PDU / seconde
  - limite de 200 participants sur un réseau Ethernet 10 Mbps (2300)

## *Systèmes sans serveur : Egal à Egal*

- Mais, le réseau local n'est qu'un cas particulier simple
- Un réseau longue distance oblige l'envoi individuel de chaque message
- Donc, 200 participants sur un LAN 10 Mbps correspondent à :
  - = 1 participant envoyant à 200 participants
  - = 200 participants envoyant à 1 participant
  - = 14 participants communiquant simultanément
  - = 1 participant gérant 14 véhicules envoyant à 14 participants

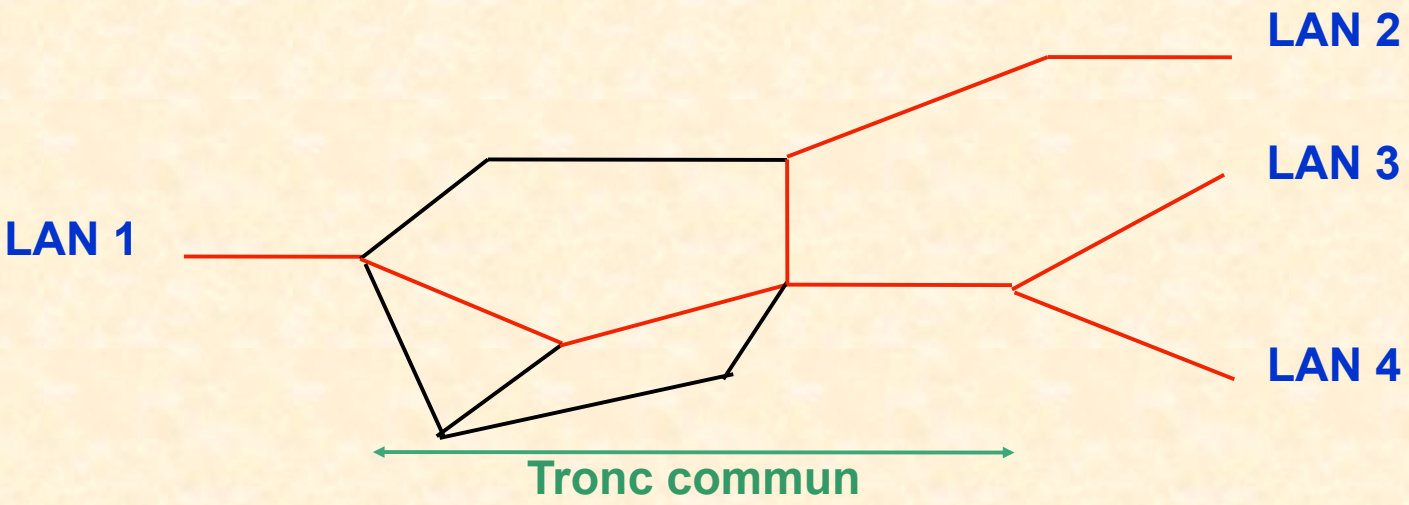
## *Systèmes sans serveur : Multicasting*

- L'envoi de messages sur un WAN est très coûteux
  - Les messages échangés entre LAN sont envoyés individuellement
  - Le même message peut voyager plusieurs fois entre deux LAN



## *Systèmes sans serveur : Multicasting*

- Le multicasting utilise le « trunk sharing » pour économiser la bande passante
- Après s'être abonné à une adresse multicast, les participants reçoivent tous les messages envoyés sur ce canal
- Permet à des groupes de tailles quelconques de communiquer grâce à une seule transmission
- Souvent donné comme « La Solution » aux problèmes d'extensibilité (scalability)



## *Le Multicasting à l'heure actuelle*

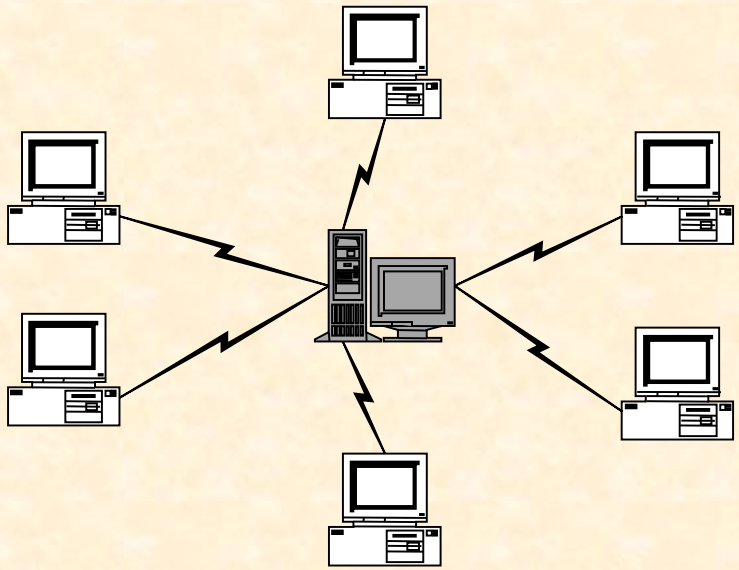
- Le Multicasting est une technologie disponible depuis plusieurs années mais le MBone n'est pas encore confondu avec Internet
  - Les routeurs Multicast ne sont pas trop répandus ou alors ils ne sont pas configurés correctement
  - Cela dit c'est le meilleur choix ...
  - … donc si un faible pourcentage des participants n'a pas accès au multicasting on peut utiliser des proxy
    - ✓ Les proxy s'abonnent à l'adresse Multicast
    - ✓ Les clients se connectent directement (via UDP) au proxy
  - Autre solution : overlay multicast ou multicast applicatif

## *Systèmes sans serveur*

- Avantages
  - Pas de goulot d'étranglement central, pas de point sensible
  - Le Multicasting conserve la bande passante du réseau
  - L'utilisation de plusieurs adresses Multicast peut servir à filtrer les messages
- Inconvénients
  - Difficile à gérer et à mettre en place
  - La consommation en bande passante (sans Mcast) est en $O(\text{participants})^2$
  - Tous les paquets diffusés sur un réseau doivent être examinés par toutes les machines du réseau (Mcast et diffusion)

## *Systèmes centralisés*

- Schéma utilisé par la plupart des FPS sur Internet
  - En général il gèrent de 8 à 32 joueurs simultanément
  - Ce nombre dépendant de la latence acceptable et de la complexité

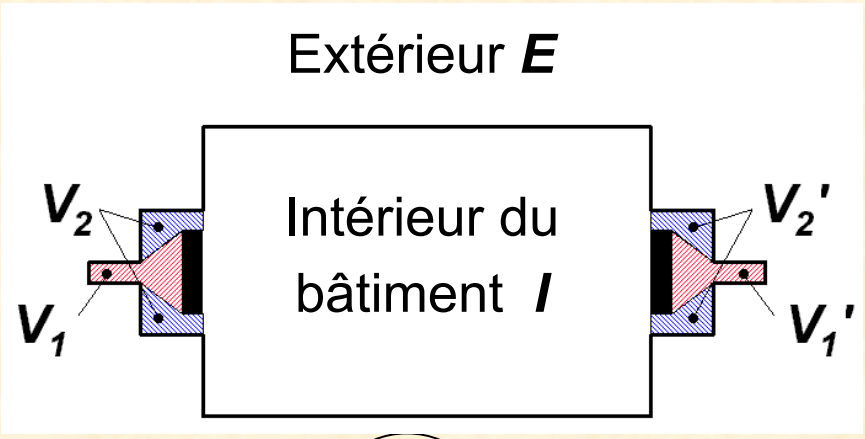


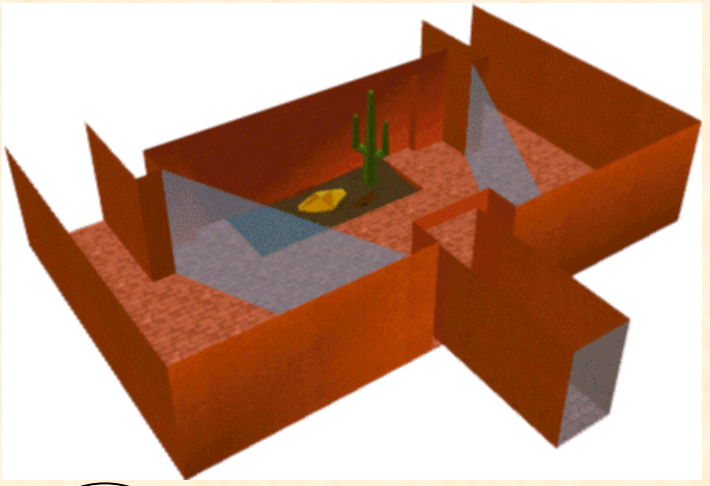
## *Systèmes centralisés*

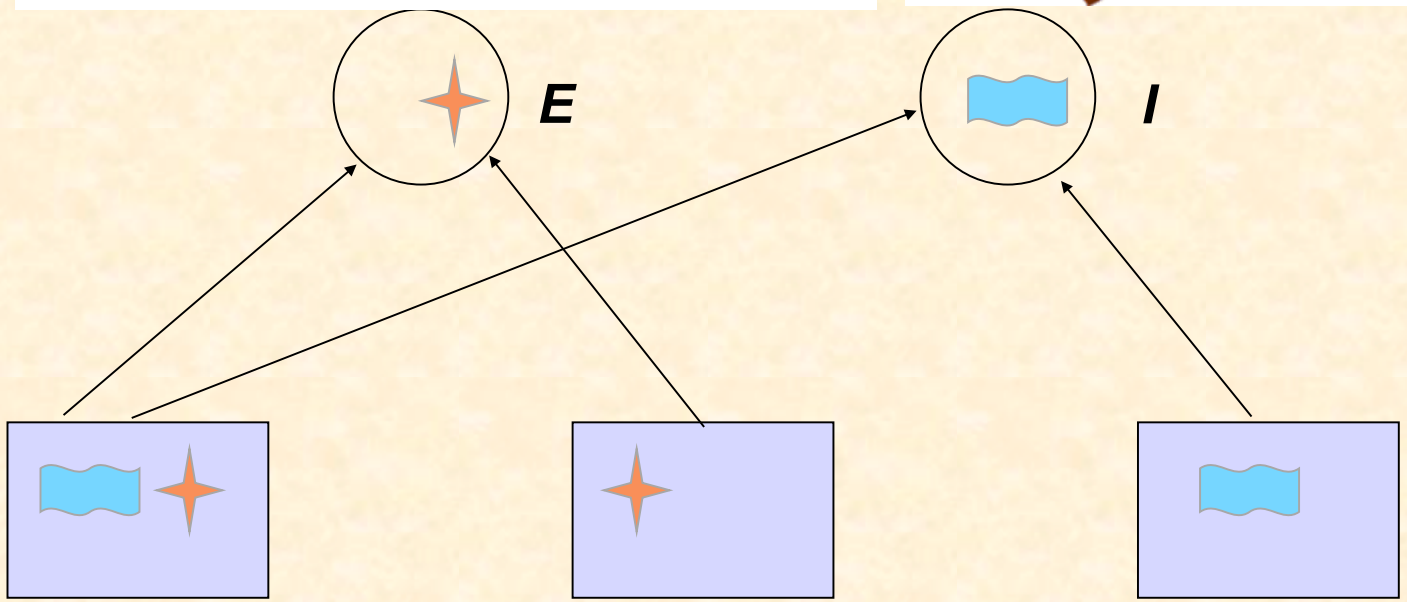
- Inconvénients
  - le serveur central est un goulot d'étranglement
  - problèmes de fiabilité (quand le serveur tombe en panne)
  - augmentation de la latence
- Avantages
  - les goulots d'étranglement peuvent être utiles
  - possibilité d'authentifier/de facturer les participants
  - le serveur peut filtrer les messages
  - le serveur peut compresser plusieurs messages en un seul

## *Systèmes à plusieurs serveurs*

- Les serveurs peuvent gérer des parties différentes du monde virtuel



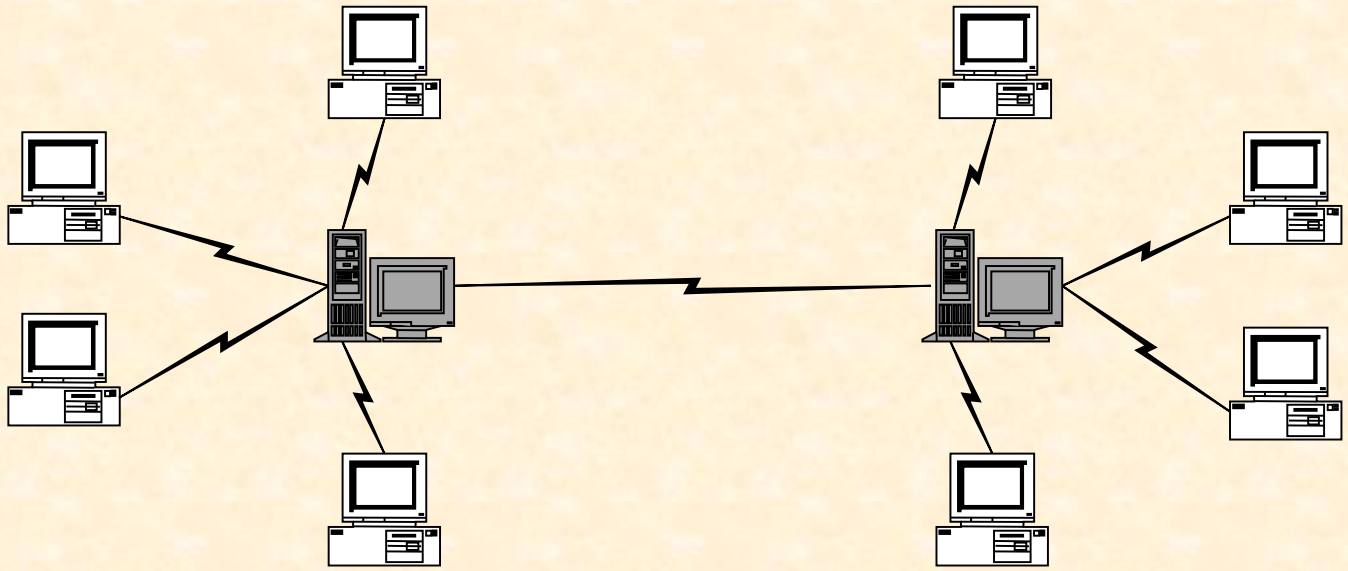


## *Systèmes à plusieurs serveurs*

- Avantages
  - Fiabilité : utiliser les serveurs pour gérer une redondance
  - Extensibilité : les tâches du serveur central sont divisées
    - ✓ en groupant les clients
    - ✓ en divisant le monde en régions
- Inconvénients
  - Les objets partagés par plusieurs serveurs ne propagent pas leurs changements
  - Nous avons toujours un point sensible
    - ✓ en fait plusieurs points sensibles
  - C'est le modèle du World Wide Web
    - ✓ qui est parfait non ?

## *Systèmes à plusieurs serveurs coordonnés*

- Les serveurs communiquent
  - pour gérer les modifications d'objets partagés
  - pour équilibrer les charges



## *Systèmes à plusieurs serveurs coordonnés*

- Avantages
  - une hiérarchie de serveurs peut permettre un filtrage efficace
  - équilibrage des charges dynamique
  - ils peuvent partager un même modèle de monde
  - les serveurs peuvent communiquer en multicast (solution hybride P2P et C/S)
- Inconvénients
  - la coordination est une tâche délicate
  - si il y a une hiérarchie, les niveaux d'indirections peuvent augmenter la latence

*En résumé*

- Le choix n'est pas facile

- Si le plus important c'est

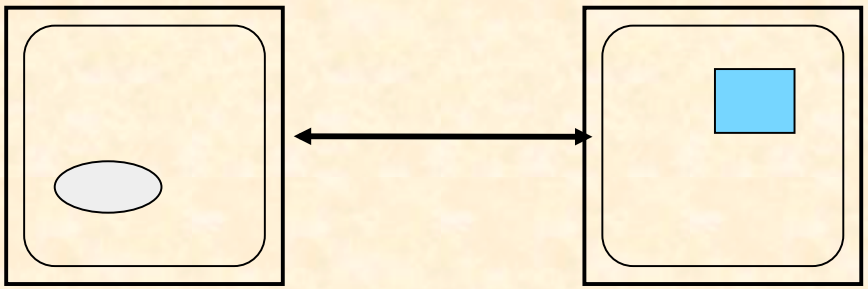
- l'extensibilité : Sans serveur / Serveurs multiples non coord.
- la fiabilité : Serveurs multiples coordonnés
- la simplicité : Centralisé
- l'interactivité : Sans serveur

## *Gestion dynamique d'un état partagé*

- Définition
- Quel est le problème
- Le choix entre Cohérence et Débit
- Les différentes approches
  - Entrepôts partagés
  - Diffusions fréquentes
  - Dead-reckoning

## *Gestion dynamique d'un état partagé*

- Un système de RVD est utilisé par plusieurs participants
- … répartis sur plusieurs machines
- … modifiant tous en permanence leur état (position, etc.)
- … et essayant tous de se voir bouger en temps-réel



- Problème : Comment être certain qu'ils voient tous la même chose ?

## *Quel est l'intérêt de partager un même état ?*

- Un état partagé rends un environnement virtuel plus réaliste
  - On peut voir la position et l'orientation de tous les autres objets de l'environnement (y compris les avatars)
  - On peut percevoir les comportements des autres participants et des objets autonomes
  - Ainsi, on peut déterminer comment et quand interagir (se rapprocher, tirer, parler…)
  - On peut partager les mêmes effets de l'environnement (météorologie, terrain, visibilité, radiations…)

## *Problématique répartie*

- Le but : transmettre l'information à tous les sites en temps-réel
  - Mais, la traversée du réseau prends du temps (latence)
  - Chaque participant a une latence différente
  - Le flux d'information est limité par la bande passante du réseau
  - Il peut y avoir des pertes de paquets

- Le problème : s'assurer que tout le monde a reçu l'information prends du temps

## *Le choix entre Cohérence et Débit*

- Il est impossible de permettre un changement fréquent de l'état partagé et de garantir que tous les sites ont accès à des versions identiques de cet état

- Il faut choisir entre
  - un monde avec de nombreux changements dans son état
  - … et ne pas attendre la résolution des problèmes d'incohérence dus à la transmission

  - et un monde moins dynamique
  - … et prendre le temps de résoudre toutes les incohérences

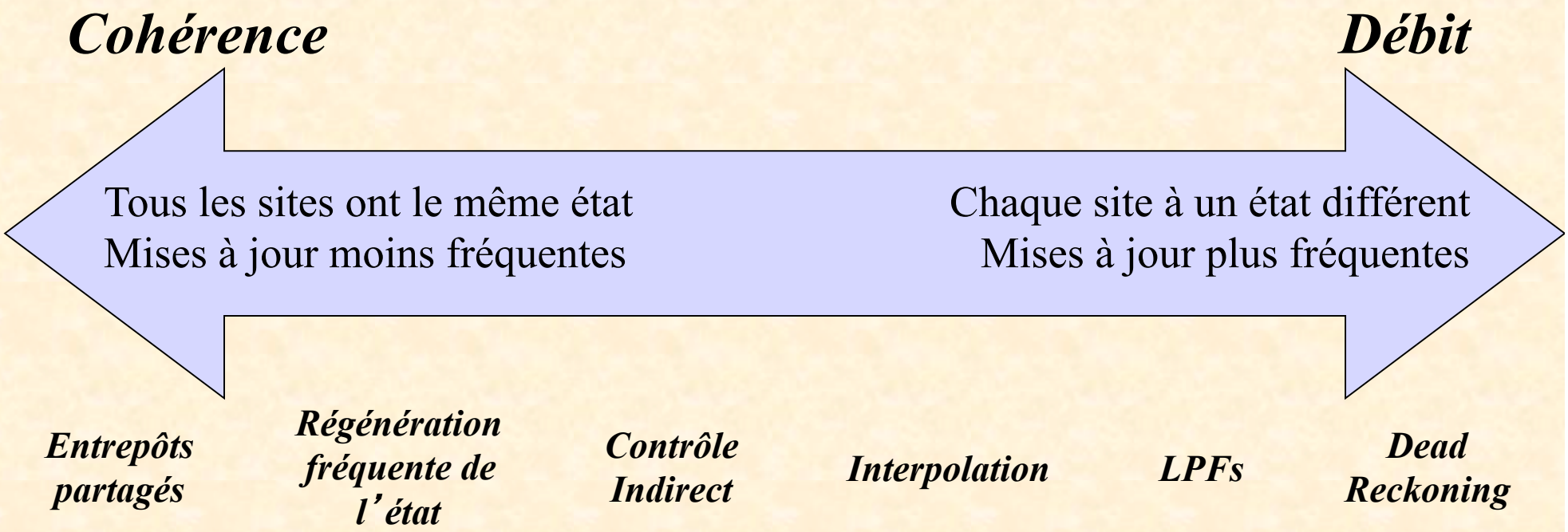
## *Autre formulation du choix*

- La bande passante réseau disponible doit être répartie entre les messages servant à mettre à jour l'état partagé et les messages servant à maintenir une vue cohérente pour tous les participants de ce même état partagé

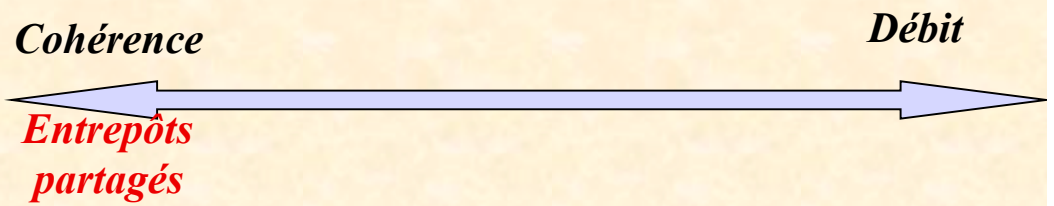
- Soit on envoie plus de messages de mise à jour
- Soit on envoie plus de messages pour s'assurer de la cohérence
- … mais on ne peut pas faire les deux

*Implication sur la conception de l'application*

- Il faut choisir la technique de gestion de l'état partagé en fonction des caractéristiques de l'environnement à simuler

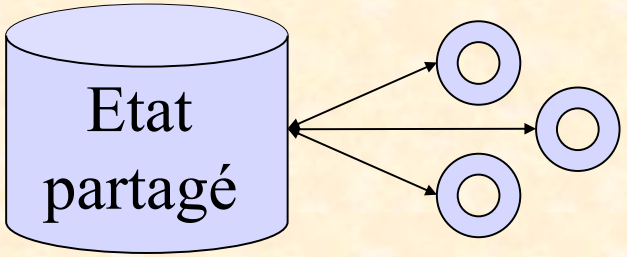


# *Les entrepôts partagés*



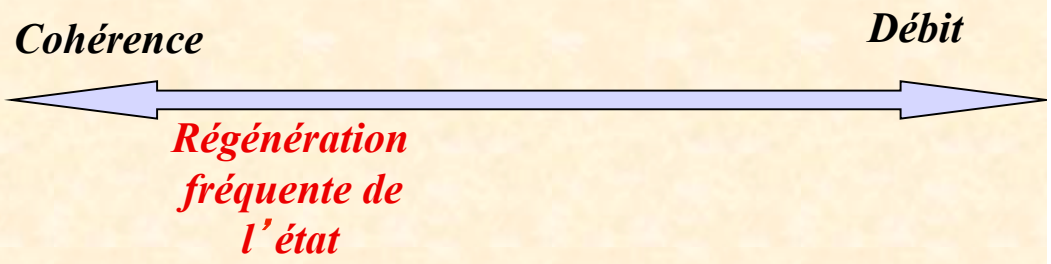

- Conserver l'état dans un lieu de stockage commun
  - Les mises à jour sont faites dans ce lieu de stockage
  - Toutes les lectures de l'état sont faites depuis ce lieu (pas de cache)
  - Exemple : placer toute l'information sur un système de fichier NFS ou AFS
  - Meilleures approches : entrepôts « virtuels »
    - ✓Un serveur envoie les mises à jour de façon synchrone vers chaque cache client
    - ✓Protocoles de cohérence répartie (ex. ISIS)

## *Les entrepôts partagés*



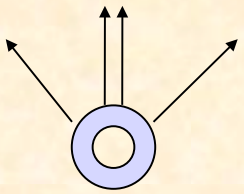

- Avantages
  - Modèle de programmation simple (indépendant du site)
  - Cohérence d'état absolue
  - Les données appartiennent à tout le monde -- Il suffit de rajouter des verrous ou des sémaphores à l'état si nécessaire
- Inconvénients
  - Souvent cela crée un point névralgique
  - Souvent cela crée un goulot d'étranglement central
  - L'éfficacité est difficilement prévisible
  - Surcoût en communications
- A réserver aux petits systèmes sur LAN ou à des systèmes nécessitant une forte cohérence

## *Régénération fréquente*

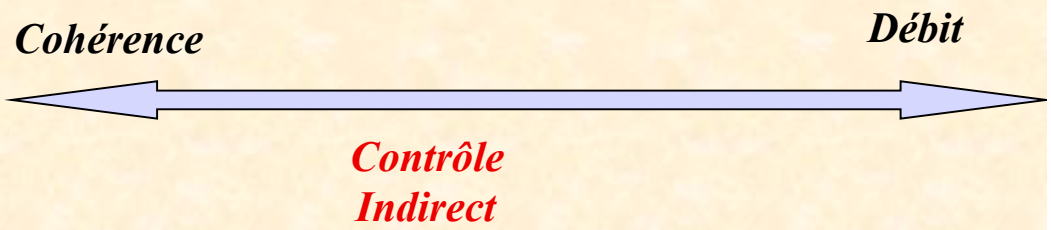



- Envoyer fréquemment l'état courant
  - Utilise généralement des diffusions sur des groupes de communication en utilisant le filtrage pour réduire l'utilisation de la bande passante
  - A rajouter dans la boucle d'événements ou sur un « timer »
  - Ignore les problèmes de fiabilité du réseau : les mises à jours sont suffisamment fréquentes pour que les incohérences ne durent pas longtemps
  - Peut être intégré avec un serveur gérant la propriété des objets ou un système de verrous répartis

## *Régénération fréquente*



- Avantages
  - Assez simple à mettre en œuvre
  - Ne nécessite pas de serveur ou d'autre infrastructure
  - Meilleur débit que les entrepôts partagés
- Inconvénients
  - Consomme une bande passante considérable (à réserver aux réseaux locaux)
  - L'utilisateur perçoit la latence réseau (« lag ») et la gigue (variation de la latence)
  - Pas transparent puisque l'utilisateur perçoit la différence de taux de mise à jour
- Très utilisé, bon pour des systèmes moyens sur LAN, intéressant si vous devez adapter un système mono-utilisateur

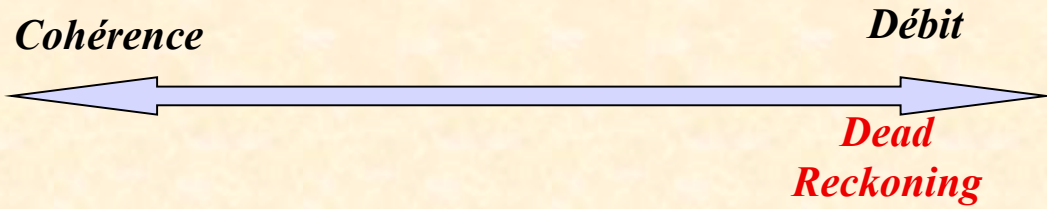



# *Contrôle indirect*

- Préciser uniquement le point d'arrivée et l'estampille temporelle de fin de déplacement
  - L'objet X (0,10,0) doit être en (10,10,0) à T0+10s
  - En local on déplace X avec comme vitesse 1m/s sur l'axe X
  - Lorsque la machine M reçoit le message il s'est écoulé 2s
    - ✓On va donc déplacer X avec comme vitesse 1,25m/s sur l'axe X
- On aura donc des incohérences pendant le déplacement mais pas à la fin.

## *Contrôle indirect*

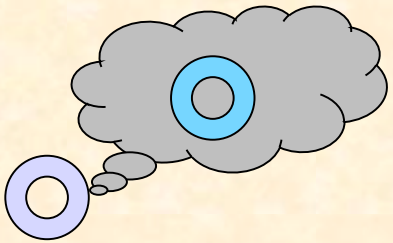
- Avantages
  - On peut gérer des latences importantes
  - La cohérence est parfaite à certains moments précis
- Inconvénients
  - Pas de cohérence en cours de mouvement
  - Il faut gérer les obstacles éventuels de façon identique sur toutes les machines
  - Moins interactif (on ne contrôle pas finement les avatars)
- A réserver aux systèmes sur WAN qui gèrent un grand nombre d'entités (exemple RTS)




# *Dead-reckoning*

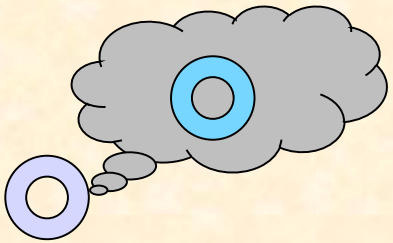
- Prédiction de l'état des objets distants basé sur des mises à jour
  - Les sites prédisent les comportements grâce aux mises à jour reçues précédemment
  - Lorsqu'une mise à jour est reçue, on fait converger l'état (mal) prédit et on met à jour les algos de prédictions
  - Les mises à jour sont envoyées moins fréquemment (dès que la prédiction deviens trop fausse)
  - La possession de l'état d'un objet doit être explicite

## *Dead-reckoning*



- Types d'algorithme de prédiction
  - prédiction linéaire
  - prédiction quadratique
  - prédiction utilisant des Splines
  - techniques hybrides
- Méthodes de convergence
  - convergence instantanée
  - convergence linéaire
  - convergence quadratique
  - convergence utilisant des Splines
  - techniques hybrides

## *Dead-reckoning*

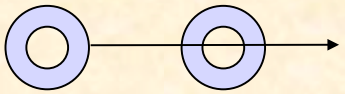


- Avantages
  - Chaque objet peut générer ses messages de mise à jour de façon totalement autonome
  - Insensible à la latence du réseau
  - Diminue la fréquence des mises à jour et donc l'utilisation de la bande passante
- Inconvénients
  - Plus complexe à mettre en œuvre
  - Le modèle de prédiction dépends du type d'objet
  - Les erreurs de prédiction peuvent être importantes si le réseau est mauvais (en fiabilité et/ou latence)
- Intéressant pour des gros systèmes fonctionnant sur des WAN et se satisfaisant de quelques incohérences

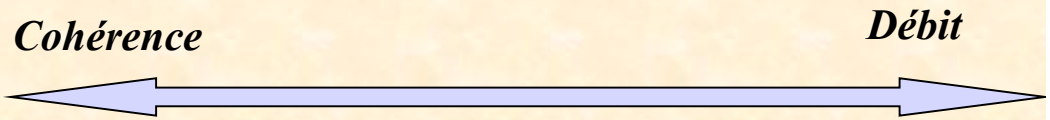

## *Interpolation*

- Le principal problème du dead-reckoning est que lors des extrapolations on est quasiment toujours hors de la trajectoire réelle de l'objet
- Une solution consiste à envoyer régulièrement la position des entités (par exemple toutes les 100 ms) et interpoler les entités dans le passé en n'utilisant que les positions connues
- Donc on ne sort de la trajectoire qu'en cas de perte de message (si on extrapole dans ce cas, autre solution on immobilise l'entité)

## *Interpolation*



- Avantages
  - On est toujours sur la trajectoire connue de l'entité
  - On a une utilisation constante de la bande passante (pratique pour calculer le nombre max d'entité qu'on peut gérer)
- Inconvénients
  - Problème de perte de message. Que faire ?
  - Si la fréquence des messages de mise à jour est faible on peut modifier la trajectoire des entités (pb de sous échantillonnage)
- Intéressant pour des systèmes moyens fonctionnant sur des WAN qui nécessitent une bonne cohérence spatiale et se satisfaisant d'incohérences temporelles

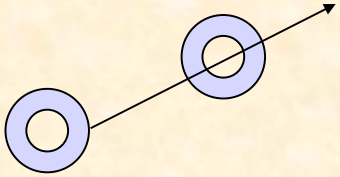



## *Filtres de Perception Locale*

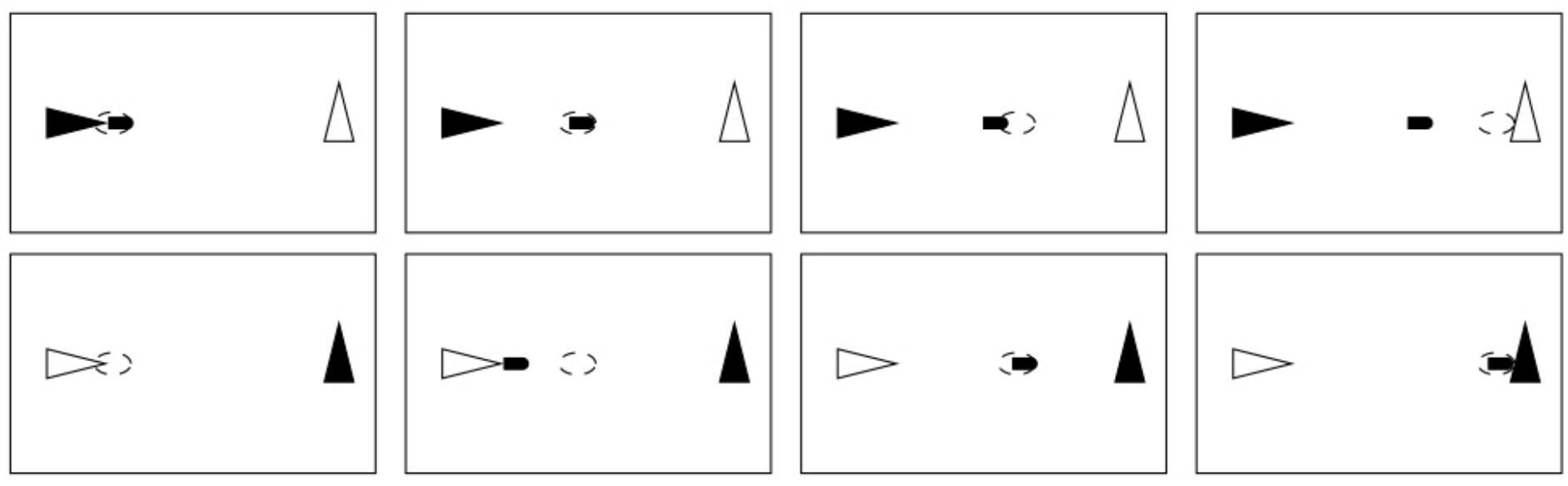
***LPFs***

- Lorsqu'il n'y a pas d'interaction directe entre les entités
- Les LPFs modifient l'écoulement du temps pour cacher la latence
- Quand un joueur J1 (connecté à la machine N) lance une balle à un instant T0
- Le message indiquant le lancement arrive à T1 au bout de 100 ms sur la machine M (utilisé par J2)
- Sur la machine N on va ralentir le mouvement de la balle au fur à mesure qu'elle s'éloigne de façon à laisser le temps aux autres joueurs d'interagir avec la balle
- Sur la machine M on va l'accélérer pour rattraper les 100 ms de retard initial

*Filtres de Perception Locale*

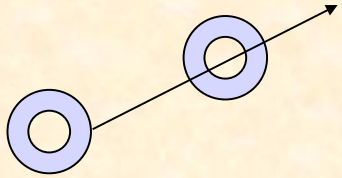


Sur la machine N

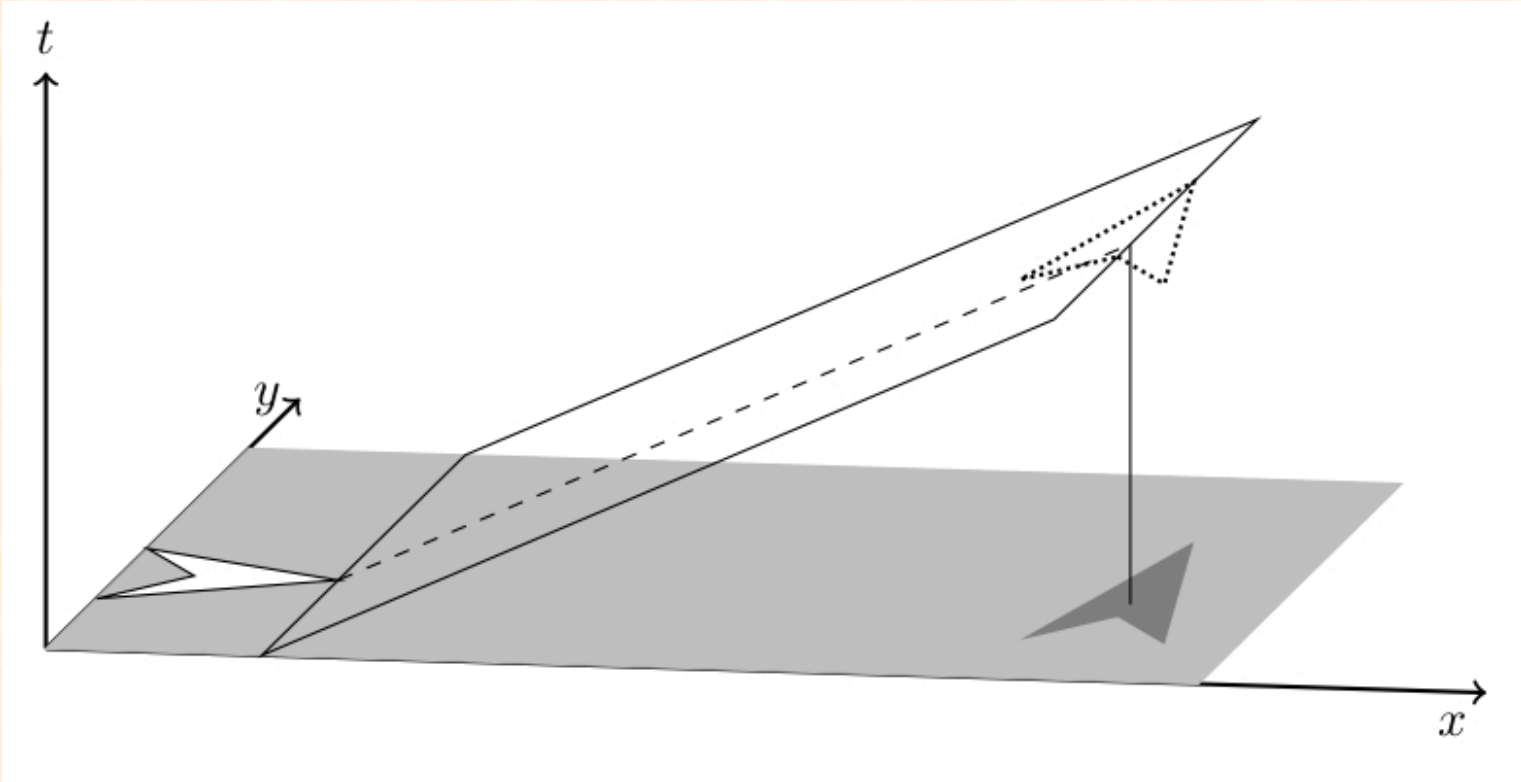


Sur la machine M

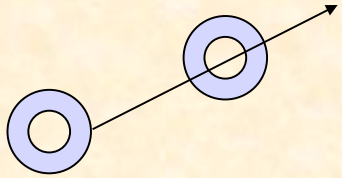
## *Filtres de Perception Locale*



- Contour Temporel sur la machine N :



## *Filtres de Perception Locale*



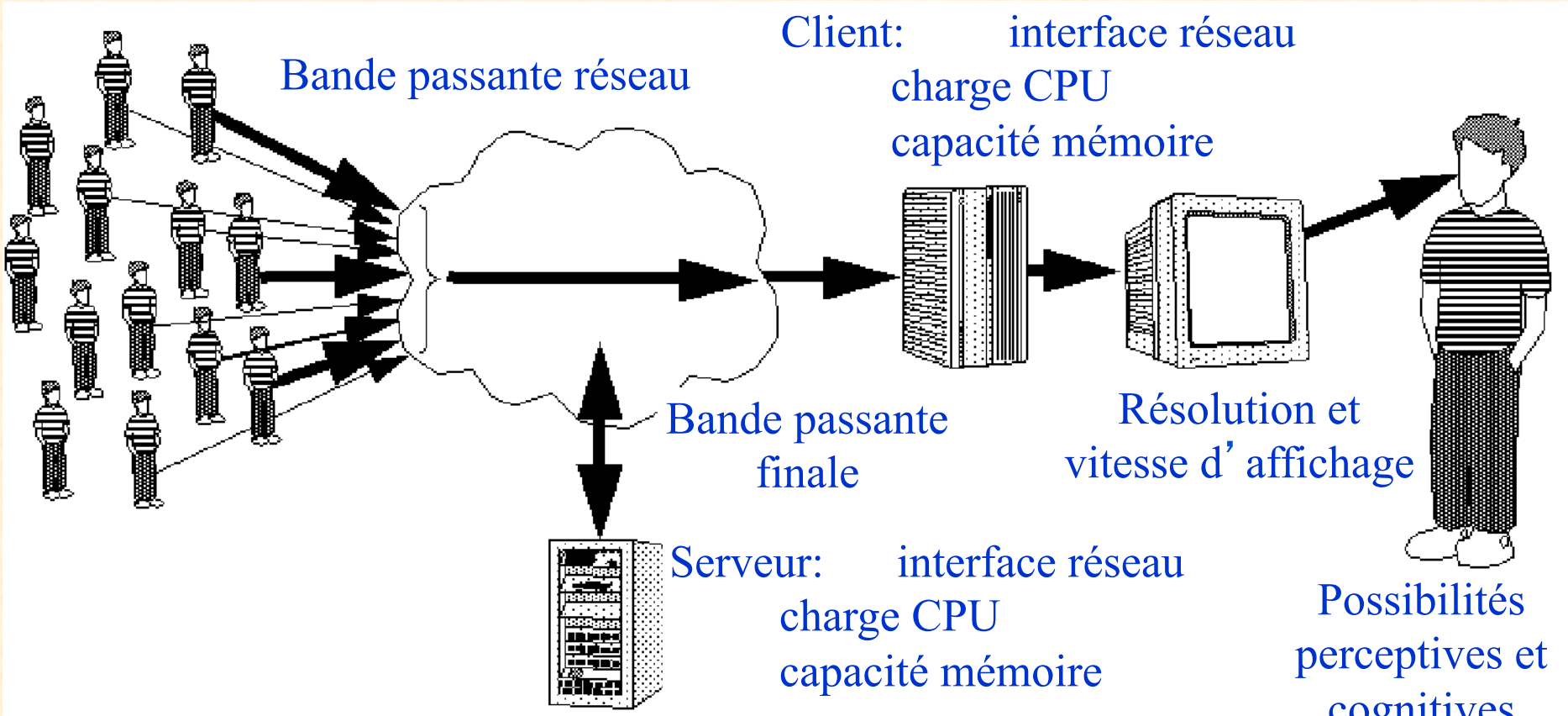
- Avantages
  - On cache la latence
  - On évite tous les problèmes de causalité
  - On peut gérer le « bullet time »
- Inconvénients
  - Ne gère pas les interactions directes
  - Problème de perte de message. Que faire ?
- Intéressant pour des systèmes moyens fonctionnant sur des WAN fiables qui ne nécessitent pas d'interaction directe

## *Filtrage par zone d'intérêt*

- Présentation du problème
- Définition de l'intérêt
- Exemples
  - Le modèle spatial d'interaction de MASSIVE
  - La gestion prédictive d'intérêt
  - Les sphères étendues
  - Les espaces de routage de HLA
  - Filtrage spatial du NPSNET
  - Filtrage spatial de RING/Spline
  - Structure d'Espace Echelle
  - La gestion d'intérêt en 3 étapes

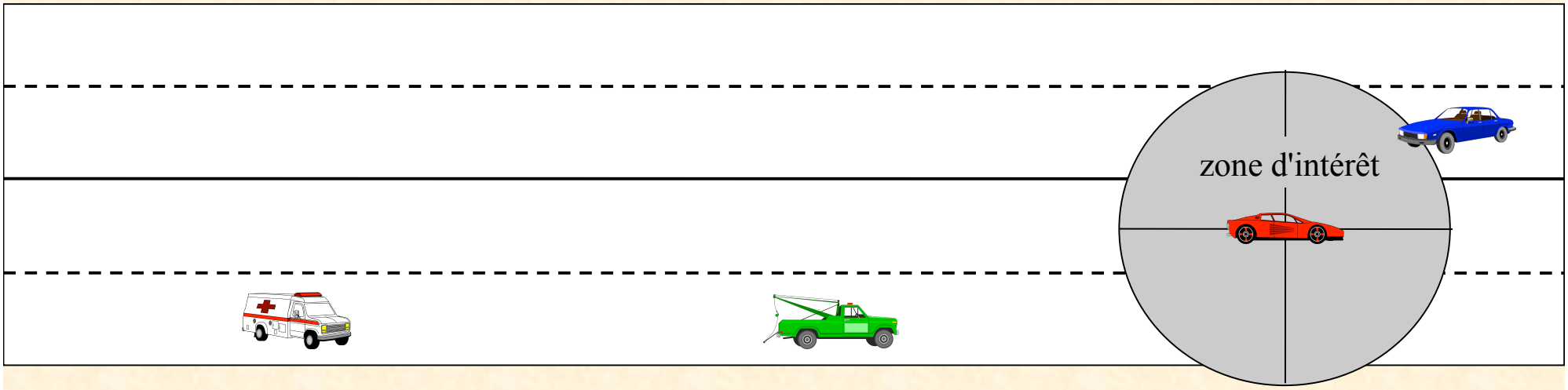
*Présentation du problème*

- Goulots d'étranglement possibles



# *Présentation du problème*

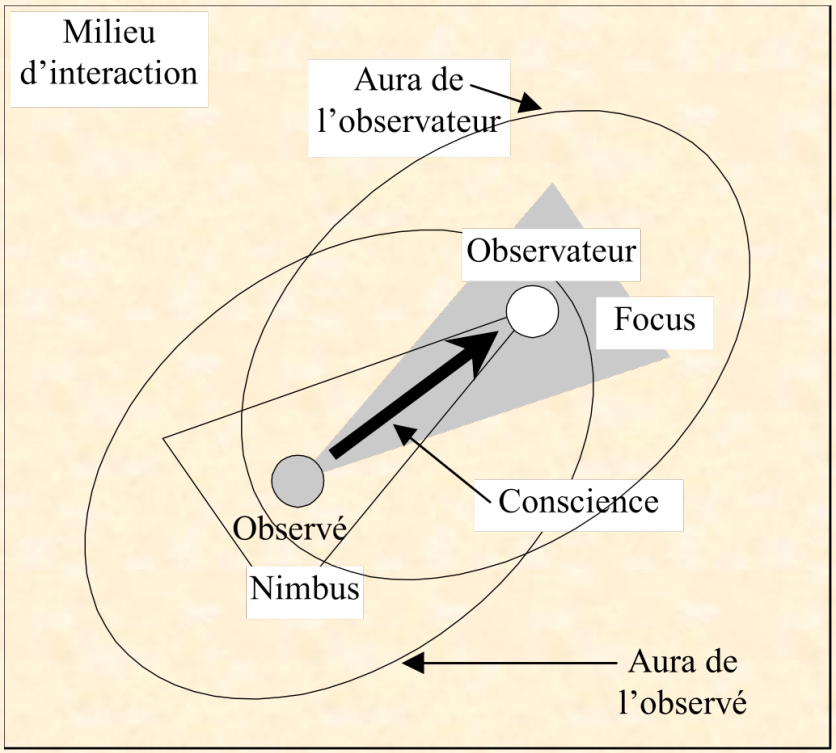
- But : voir et entendre « suffisamment », mais pas plus ...
  - Extensibilité
  - Efficacité
- Où « suffisamment » est défini par :
  - L'intérêt, la perception, la visibilité...
  - Les contraintes systèmes (goulots d'étranglement)



## *Définition de l'intérêt*

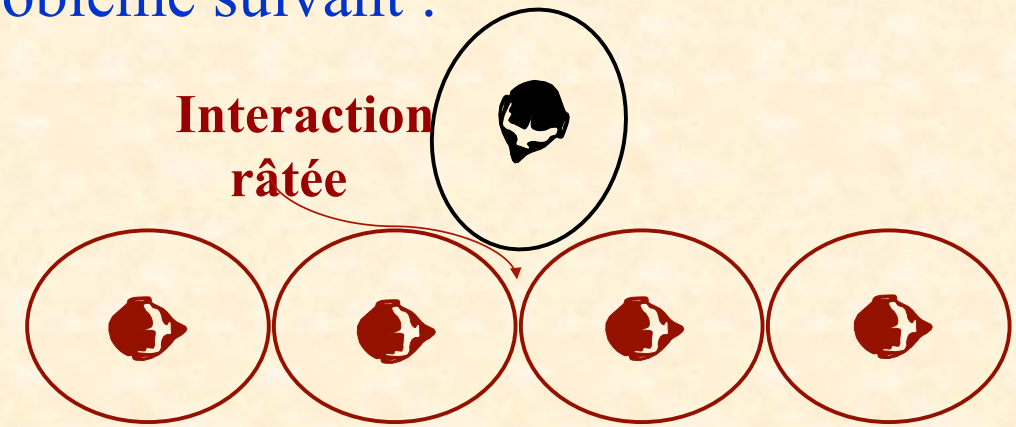
- Intérêt = perception
  - Exploitation des contraintes du « monde réel » (ex. visibilité)
- Modèles explicites de l'intérêt (principalement spatiaux)
  - Appartenance disjointe (ex. monde, pièce, classe)
  - Zone d'intérêt, « Aura »
  - Modélisation mathématique de la conscience
  - Distance topologique

# *Modèle spatial d'interaction de MASSIVE*

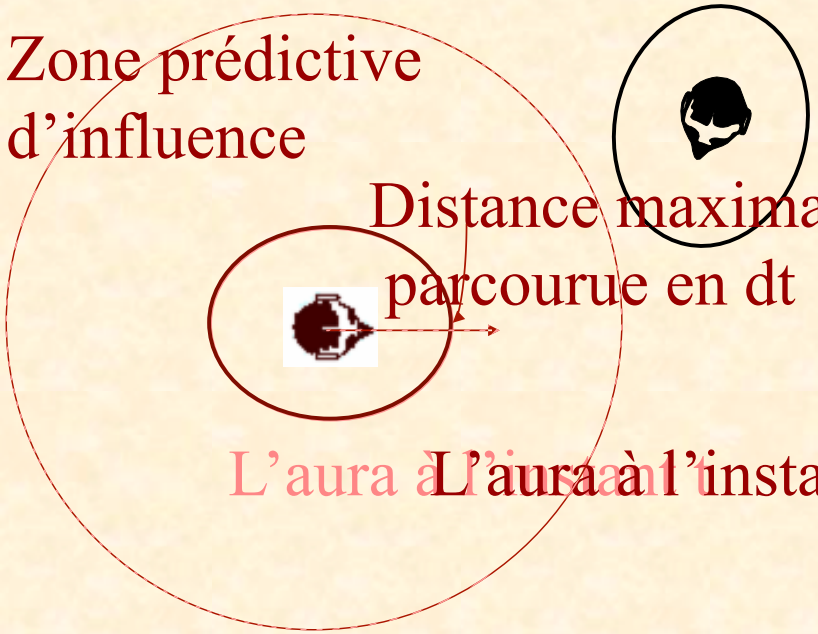


## *Gestion prédictive de l'intérêt*

- Basé sur le modèle spatial d'interaction
- Gère le problème suivant :

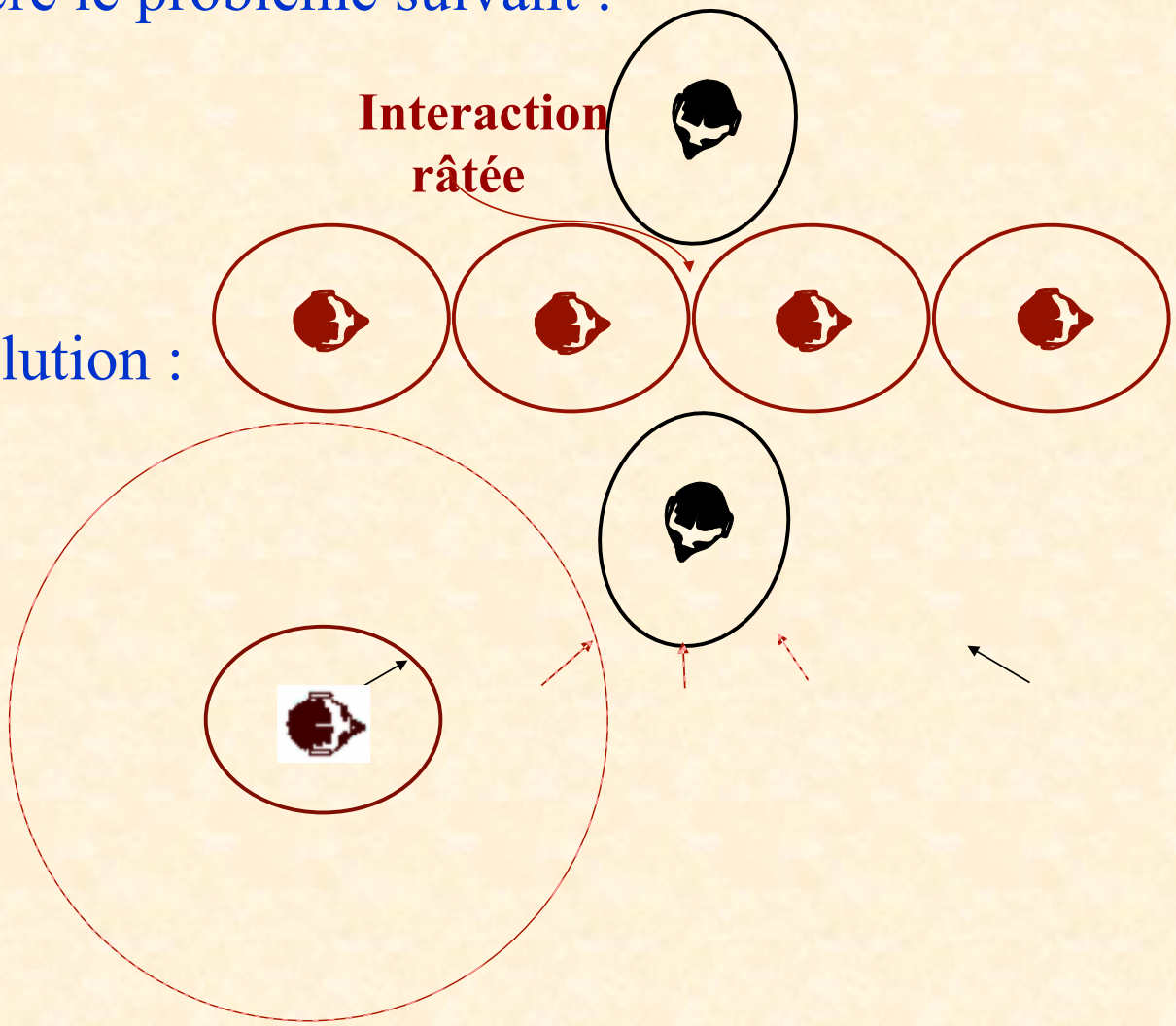



- Solution :



# *Gestion prédictive de l'intérêt*

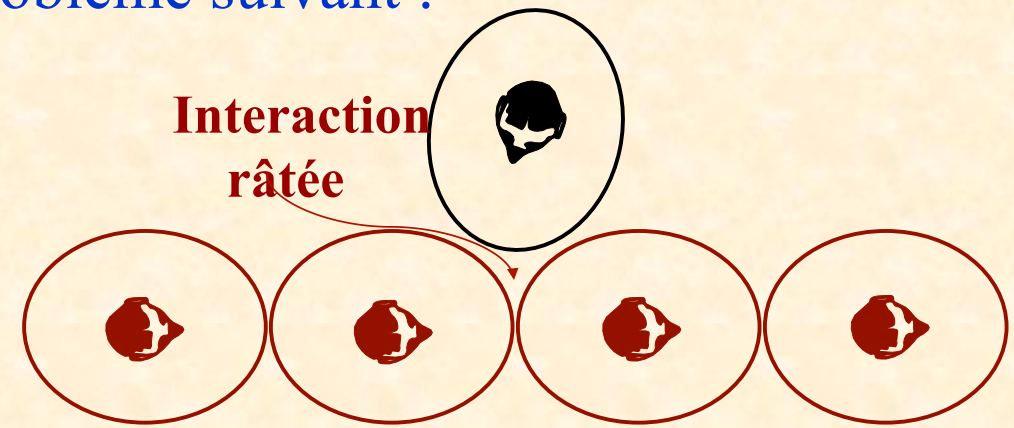
- Basé sur le modèle spatial d'interaction
- Gère le problème suivant :



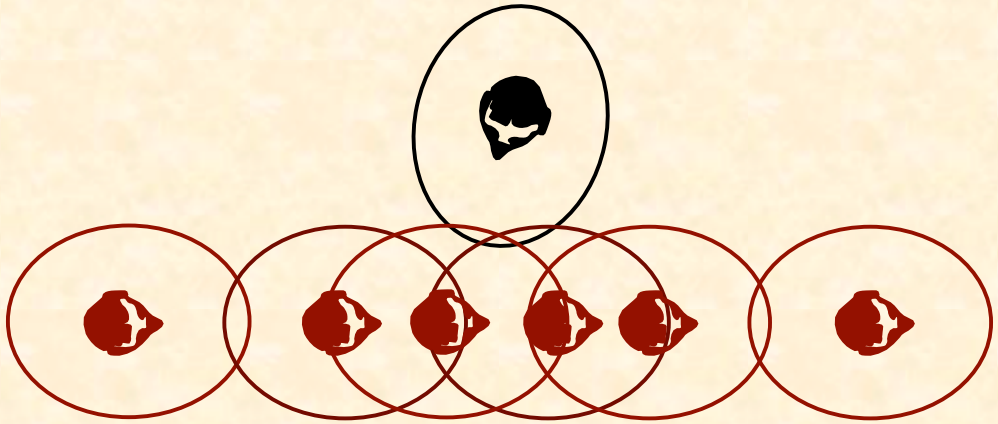

- Solution :

## *Gestion prédictive de l'intérêt*

- Basé sur le modèle spatial d'interaction
- Gère le problème suivant :

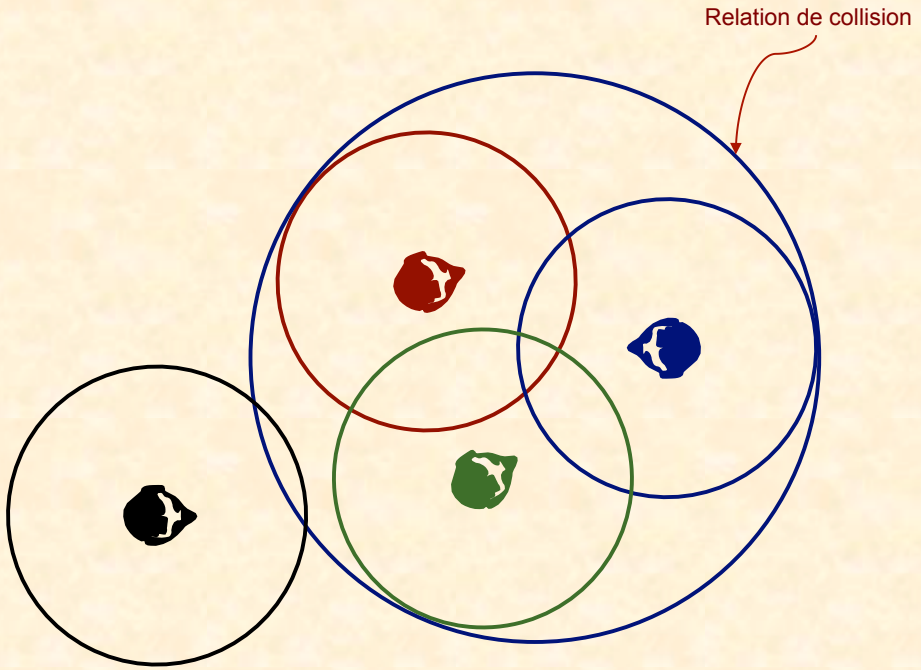



- Solution :

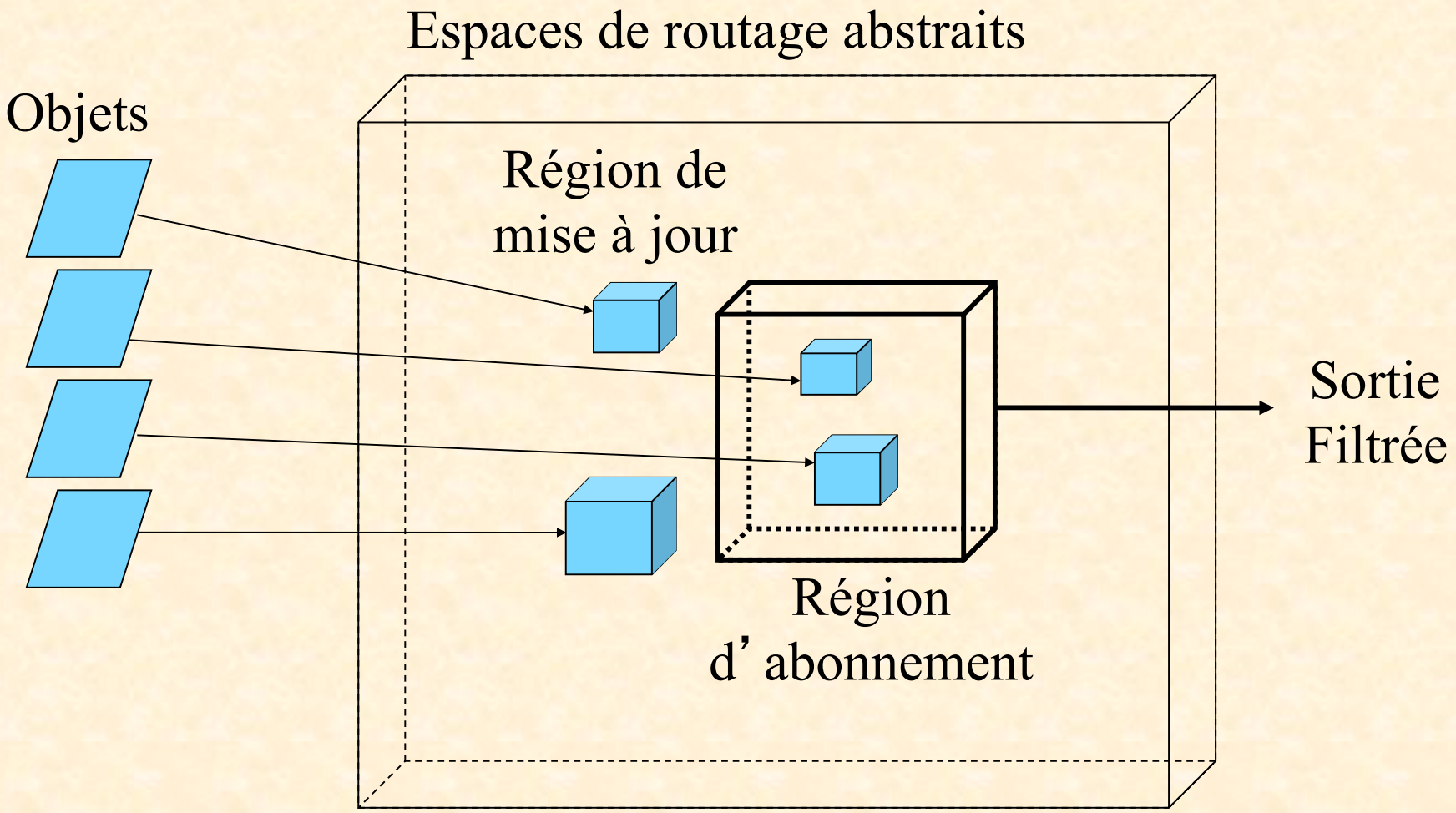


# *Les sphères étendues*

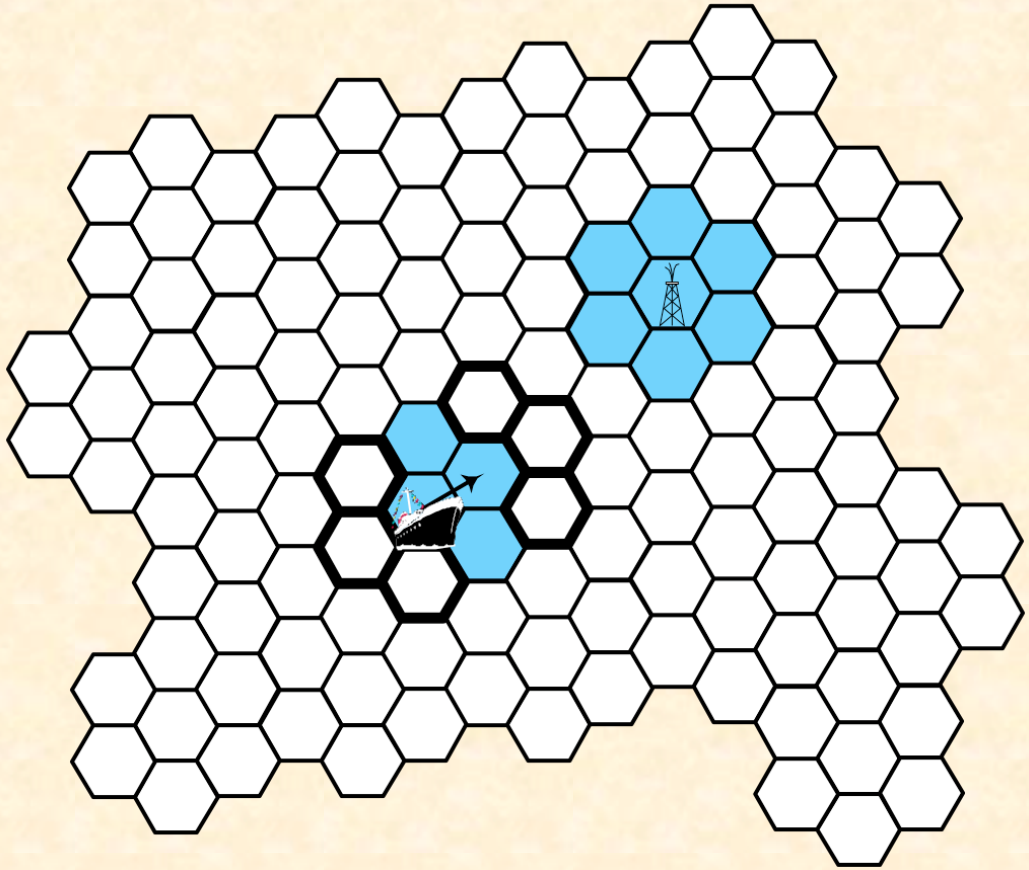
- Basé sur le modèle spatial d'interaction
- Ajoute une hiérarchie pour limiter le nombre de collisions à gérer



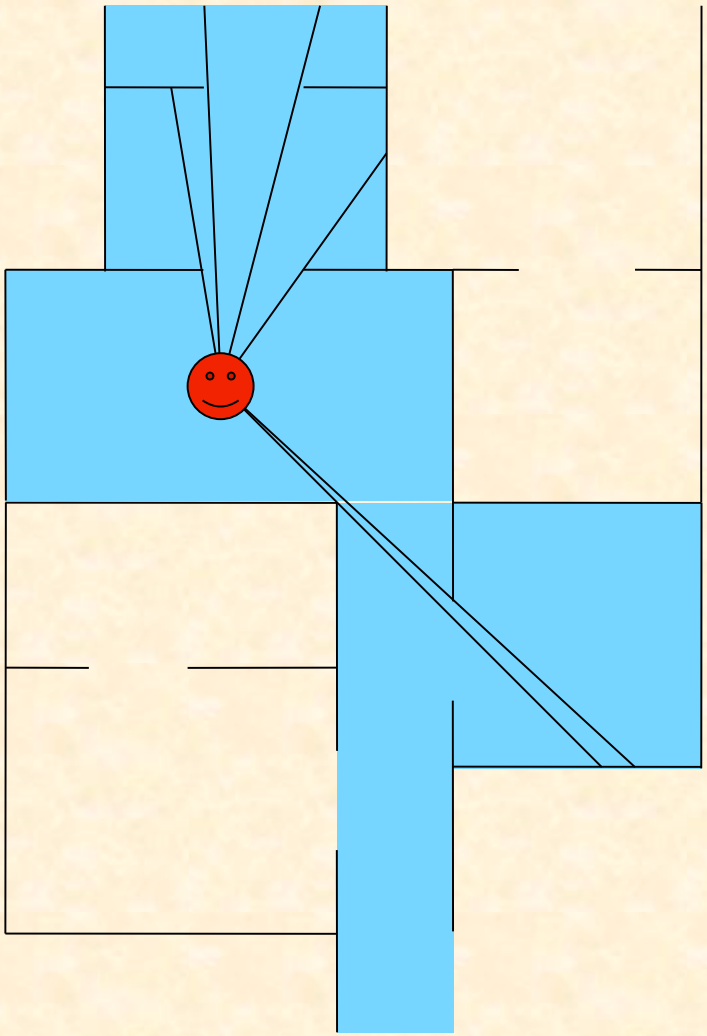
## *Les espaces de routage de HLA*



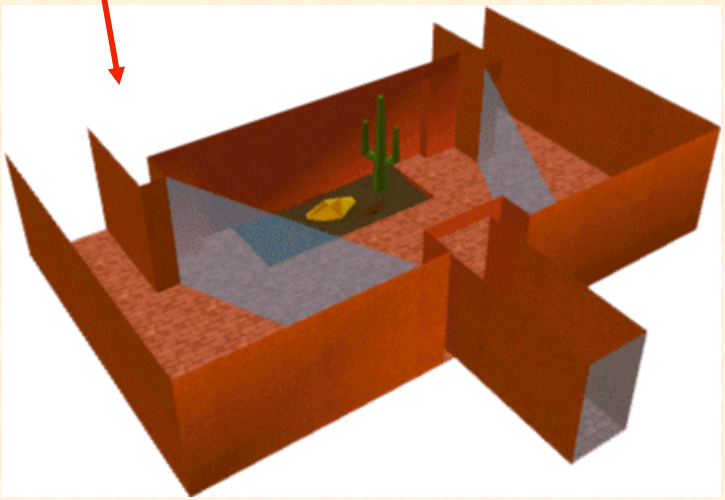
# *Filtrage spatial du NPSNET*

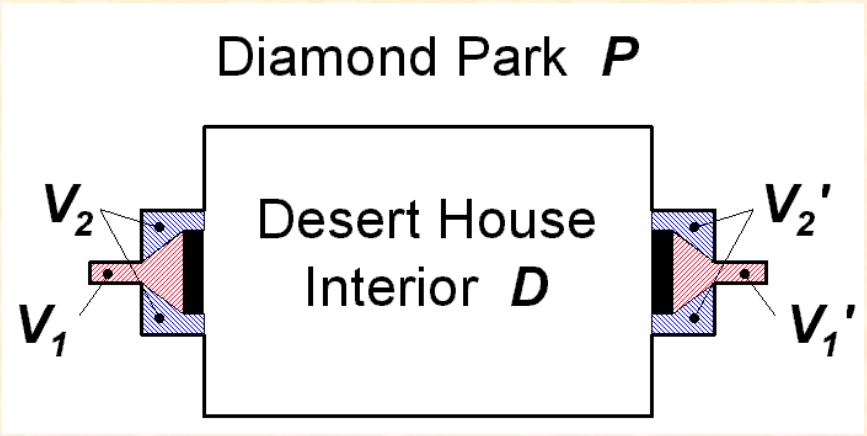


# *Filtrage spatial de RING*

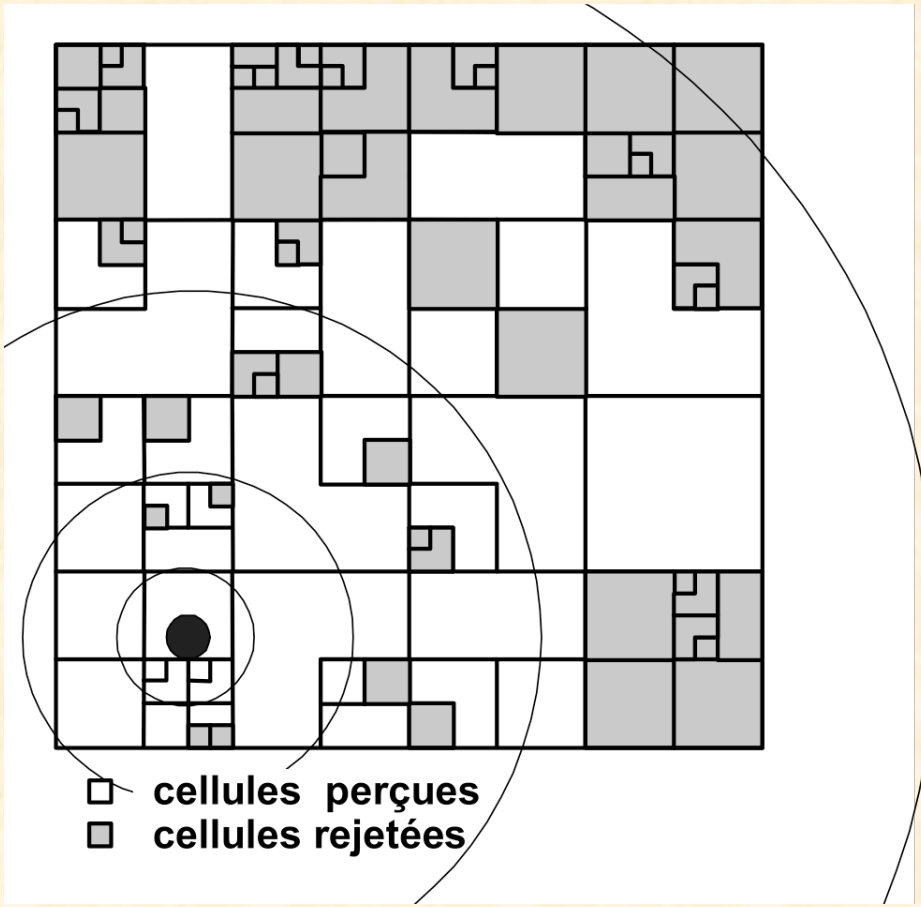


# *Filtrage spatial de Spline*





# *Structure d'Espace Echelle*

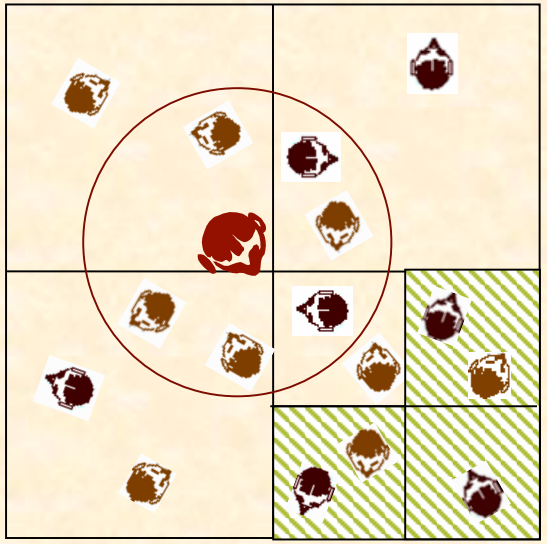


# *Gestion d'intérêt en trois étapes*

- Première étape
  - Utilisation d'un Quadtree

- Deuxième étape
  - Utilisation d' Auras

- Troisième étape
  - Filtrage fonctionnel



# *Spécificités des jeux en réseau*

- Les jeux de stratégie temps-réel (RTS)
  - Modèle synchrone
  - Modèle asynchrone
- Les jeux à la première personne (FPS)
  - Architecture standard
- Les jeux de rôle massivement multi joueurs (MMORPG)
  - Les “shards”
  - Les zones
  - Les instances
  - Les proxies
  - Architecture sans zones (seamless)

# *RTS*

- Le joueur commande une armée et une population
- Il construit des bâtiments et des unités
- Il donne des ordres à ses unités
- Il ne connaît que ce que ses unités voient (“Fog of war”)
- Le jeu a souvent une interactivité limitée
- Plusieurs phases typiques :
  - Construction
  - Exploration du monde
  - Combat

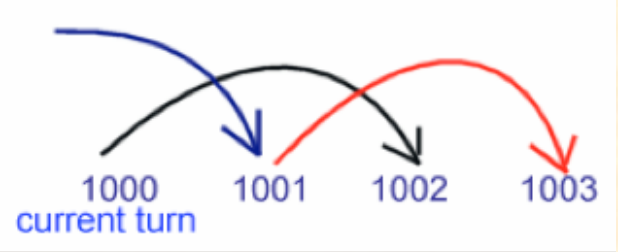
## *RTS synchrone*

- Le problème : comment faire passer les positions de milliers d'unités avec une connexion modem 56K
  - Considérons une information de position en 2D (x,y sur 16 bits), une orientation (8 bits) et un indentifiant de l'unité (16 bits) => 5*8 = 40 bits
  - Limite : 56000 / 40 = 1400 unités se déplaçant en même temps 1 fois par seconde
- Solution : simuler le monde de manière identique sur chaque machine et ne faire passer que les ordres des utilisateurs (exemple : clic souris en position X,Y écran, appui sur une touche du clavier, choix de l'option I d'un menu, ordre O donné à un groupe d'unité...)

# *RTS synchrone*

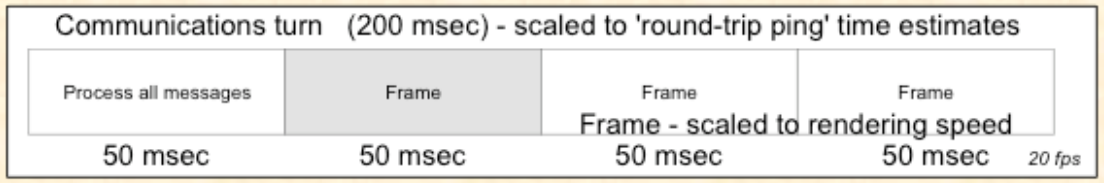
- Fonctionnement (exemple de Age of Empire 1/2)
  - On calcule la longueur d'un tour de jeu en fonction de la latence calculée (aller-retour / 2)
  - 1 Tour de jeu est plus long qu'un pas de simulation
  - Dans chaque tour de jeu
    - ✓ A chaque commande utilisateur, on envoie un message
    - ✓ On reçoit les commandes des autres utilisateurs
      - Toutes les commandes sont stockées
    - ✓ Lorsque le tour est terminé, on envoi un message de fin de tour avec des informations de temps pour éventuellement re-régler la longueur d'un tour
    - ✓ Puis on exécute les commandes dans le même ordre sur chaque machine (joeur 1 en premier, puis joueur 2, joueur 3...)
    - ✓ Puis on exécute X pas de simulation (dépends de la longeur d'un tour)

# *RTS synchrone*

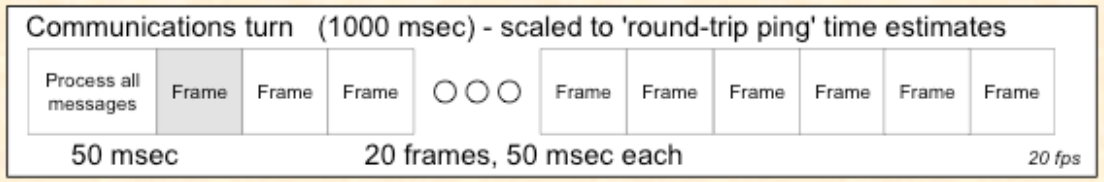



- Fonctionnement (exemple de Age of Empire 1/2)
  - Pour gérer les problèmes de latence élevée on envoie des commandes qui seront exécutées dans 1 ou 2 tours de jeu
  - La taille d'un tour de jeu dépends de la latence

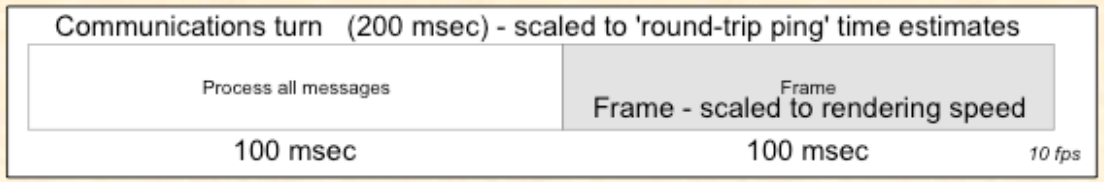
*A Single Communication Turn*




*High Internet Latency with normal machine performance*




*Poor machine performance with normal latency*



## *RTS Synchrone*

- Avantages
  - Quelle que soit la vitesse du réseau on peut gérer un nombre quasi infini d'unités
  - Exemple de bande passante utilisée : Warcraft III : 7 kbps max
- Inconvénients
  - Il faut attendre que tous les messages des joueurs dans un pas de temps soient envoyés avant de les exécuter dans le même ordre.
    Si on a une grande latence le jeu peut bloquer.
    Ce n'est pas bon pour le passage à l'échelle (on attends trop)
  - On doit avoir une simulation totalement déterministe. Si on utilise un générateur pseudo aléatoire il faut qu'il soit synchronisé et faire le même nombre d'appels sur chaque machine. Très dur à programmer.
  - Impossible de rejoindre le jeu en cours de partie
  - Il y a de nombreux moyens de tricher : révélation de carte...

## *RTS Asynchrone*

- Idée : un serveur gère la simulation complète et n'envoie que les informations correspondant aux zones visibles de la carte à chaque joueur
- Avantage :
  - Beaucoup plus facile à programmer (s'apparente au filtrage)
  - Plus de problème de triche de type "révélation de carte"
  - On peut rajouter des joueurs en cours de jeu
- Inconvénient :
  - La bande passante utilisée augmente énormément avec la durée de la partie. Plus on explore le monde avec un grand nombre d'unité plus on voit d'unités (en plus dans certains jeux on peut utiliser des radars...)
  - Les tours de jeux peuvent augmenter énormément pour attendre la réception de toutes les mises à jour.

## *FPS*

- Le joueur contrôle un personnage
- Il collecte des armes et des bonus
- Il élimine les adversaires (autres joueurs et “bots”)
- Le jeu est très interactif

## *FPS : architecture standard*

- Un serveur simule le monde
- Il reçoit les ordres de mouvement/tirs de chaque joueur/"bot"
- Il calcule les actions sur le monde
- Il redistribue les résultats à tous les clients
- Pour éviter que les clients attendent trop on utilise une variante du "dead reckoning"
- Avantage :
  - Très interactif (grâce au dead-reckoning)
- Inconvénient :
  - Passage à l'échelle difficile (un MMOFPS nécessite énormément de serveurs et doit avoir une latence assez faible avec l'ensemble des clients)

## *MMORPG*

- Le joueur contrôle un personnage
- Il se déplace dans le monde
- Il discute avec d'autres joueurs (soit proches - dans la même zone -, soit ayant le même intérêt - guildes)
- Lors des combats, le joueur active des compétences/sorts/ coups spéciaux qui ont une durée de récupération plus ou moins longue. Le reste du temps son personnage effectue des attaques standards.
- Le personnage gagne de nouvelles compétences au fur et à mesure
- L'interactivité est plus faible que pour un FPS mais plus forte que pour un RTS
- Le gros attrait : plusieurs milliers de joueurs partagent le même monde

## *MMORPG : les "shards"*

- Il est souvent impossible de permettre à tous les utilisateurs d'un MMO de se connecter sur le même serveur.
- Pour résoudre le problème, Ultima Online a introduit le concept de "shard".
- Chaque "shard" est une copie identique (au démarrage) de l'univers de jeu qui évolue indépendamment.
- Un joueur doit choisir sur quel "shard" il veut jouer lors de la création de son personnage (il existe aussi des services payants pour transférer le personnage d'un shard à un autre shard).
- Le terme vient du fait que dans l'univers des jeux Ultima, l'univers de jeu est supposé être prisonnier de plusieurs éclats (shards) d'un cristal magique.

## *MMORPG : les "shards"*

- Avantage
  - On peut rajouter des "shards" quand le nombre d'utilisateurs augmente et en enlever (en transférant les personnages) quand il diminue.
  - On peut placer des "shards" a plusieurs endroits dans le monde de façon à diminuer la latence avec les clients.
- Inconvénient
  - Les joueurs ne sont pas tous dans le même monde partagé ce qui limite les interactions avec d'autres joueurs

- Note : certains jeux comme EVE online n'utilisent pas de "shards" mais limitent en général l'interactivité de l'application pour permettre cela.

## *MMORPG : les zones*

- Pour pouvoir gérer plus d'utilisateurs dans le même "shard", Everquest a introduit le concept de zones en 1999.
- Chaque zone est connectée à une ou plusieurs autres zones par des "téléporteurs" qui font transiter le personnage.
- Un joueur ne peut pas voir ni interagir avec les joueurs situés dans d'autres zones.
- En général, chaque zone est gérée par un processus différent.
- L'ensemble des processus est ensuite réparti de façon statique ou dynamique sur un ensemble d'ordinateurs réunis dans un même cluster.
- Des optimisations permettent de gérer toutes les zones d'un même ordinateur dans le même processus (pour éviter les changements de contexte).

# *MMORPG : les zones*

- Avantage
  - Les zones permettent de répartir la charge du monde virtuel sur plusieurs machines.
- Inconvénients
  - Elles créent des divisions artificielles du monde virtuel.
  - Une zone peut devenir très populaire et donc saturer un ordinateur.
  - Il est souvent assez difficile de partager le monde virtuel en zones a priori pour éviter des déséquilibres de charge.

## *MMORPG : les instances*

- Le concept d'instance ou de zone instanciée a été introduit par Everquest en 2003.
- Le principe est de réserver une zone à un petit groupe de joueur et de la dupliquer pour chaque groupe (ainsi il y a plusieurs instances de la zone).
- Les instances sont gérées de façon similaire aux zones mais requièrent beaucoup moins de ressources système.
- Un ordinateur peut donc gérer un nombre très important d'instances.
- De plus, comme les instances ont une durée limitée (au plus quelques heures), le système dans son ensemble peut équilibrer les charges avec des instances. Un ordinateur gérant peu de monde sera choisi pour gérer une instance lors de sa création.

# *MMORPG : les instances*

- Avantage
  - Les instances permettent de répartir finement la charge du monde virtuel sur plusieurs machines.
  - On peut prévoir facilement la charge requise pour gérer une instance compte-tenu de la limite sur la taille des groupes.
- Inconvénients
  - Pas vraiment d'inconvénients si ce n'est que les duplication d'une même zones ne sont pas très naturelles
- Note : on peut aussi avoir des instances qui durent dans le temps (appelées des raids) mais il n'y a pas de grande différences côté gestion de ces instances par le système si ce n'est côté SGBD.

## *MMORPG : les proxies*

- Pour éviter qu'un client ne voie directement tout les ordinateurs d'un serveur de MMORPG on utilise des serveurs proxies.
- Lors de sa connexion au système (et après une étape d'authentification sur un serveur spécifique) un proxy est attribué au client et le restera pendant toute la session de jeu (jusqu'à la déconnexion du système).

- Avantage
  - Les proxies cachent les autres serveurs qui sont donc moins sujets à des tentatives de piratage
- Inconvénient
  - La traversée des proxies rajoute un peu de latence

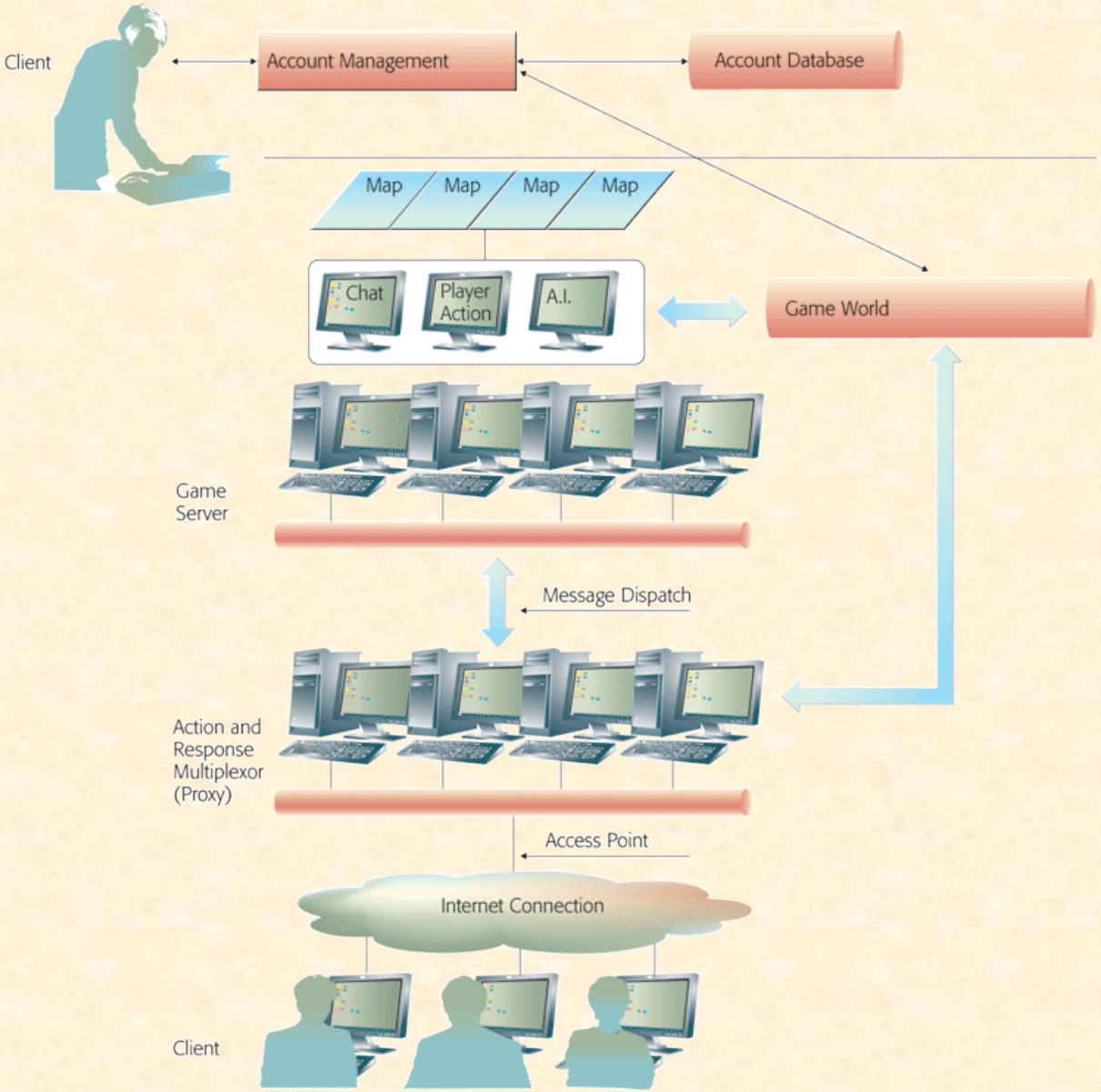
## *MMORPG : architecture sans zones*

- L'architecture sans zones (seamless : sans coutures) a été introduite par World of Warcraft en 2004.
- Le principe est de gérer tout (ou une grande partie) le monde virtuel sans zones (comme pour Ultima Online) mais avec plusieurs ordinateurs.
- Le monde virtuel est découpé en régions qui sont allouées dynamiquement à plusieurs machines.
- Si une région est trop peuplée elle est re-découpée et les parties sont attribuées à des machines différentes

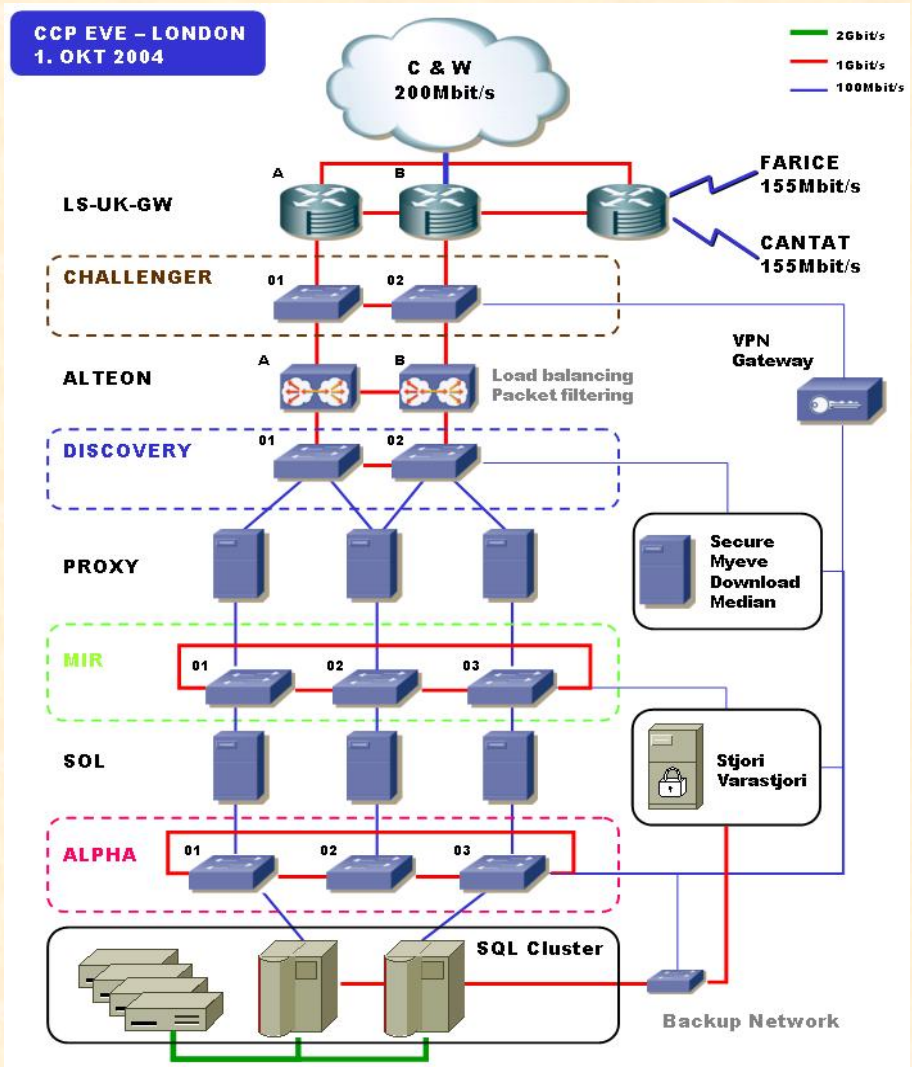
# *MMORPG : architecture sans zones*

- Avantage
  - Le monde parait plus réaliste puisqu'il n'y a plus de découpages artifiels
- Inconvénients
  - Cette architecture est beaucoup plus difficile à gérer et à coder
  - Les transferts d'objets entre personnages placés dans des régions différentes sont très compliqués à gérer pour éviter des duplications d'objets si l'une ou l'autre des machines crashent.

# *Exemple d'architecture de MMO*



# *Exemple d'architecture de MMO : EVE online*




- 400 GHz CPU / 200 Gb RAM
- 2 Routeurs (CISCO Alteon)
- 14 serveurs Proxy (IBM Blade)
- 55 serveurs Sol (IBM x335)
- 2 serveurs BD (en cluster, IBM Brick x445)
- Windows 2000, MS SQL Server
- Mise à jour :
  - AMD x64 IBM Blade

## *Remerciements*


- Ce cours est basé en partie sur :
  - Le cours de Michael Zyda (Gamepipe laboratory)
  - L'ouvrage "Networked Virtual Environments - Design and Implementation", Sandeep Singhal & Michael Zyda, ACM Press 1999.
  - Les ouvrages "Massively Multiplayer Game Development" tome 1 et 2, Alexander Thor (éditeur), Charles River Media, 2003 et 2005
  - L'ouvrage "Algorithms and Networking for Computer Games", Jouni Smed & Harri Hakonen, Wiley, 2006
  - Le cours sur la réalité virtuelle distribuée du SIGGRAPH'99
  - La soutenance de thèse de Nicolas Farcet (LCR de Thomson CSF)