PG306
Programmation des applications réparties

Alexandre Denis – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest
France

Préambule

- Le prof
  - N'hésitez pas pour toute question
  - `Alexandre.Denis@inria.fr`
- L'équipe
  - Équipe Inria **Tadaam**
  - HPC, MPI, placement, etc.
  - Opportunités de poste ingénieur, stage, thèse, projet, etc.

Organisation du cours

- 6 séances de 4h de cours intégré
- En pratique :
  - Cours général
  - Présentation d'une technologie
  - TP sur la technologie présentée
- Au programme :
  - Java RMI, CORBA, Web Services, REST, Hadoop
  - Autres abordés dans les exposés

Approche du cours

- Grande **diversité** de technologies, d'implémentations
  - Chacun réinvente la roue sans se préoccuper de ce qui se fait :
    - chez les concurrents (Sun/Oracle, Microsoft, IBM, Redhat, etc.)
    - dans les domaines voisins (HPC, big data, finance, gestion, stockage, web, etc.)
  - Beaucoup de startups & de projets open-source +/- viables
- Domaine qui change très vite
  - Surtout en technos Web
- On apprend les **concept généraux** et les fondamentaux (RMI, CORBA, REST, ...)
- On doit savoir **s'adapter** rapidement à une techno proche

Évaluation

- Projet + exposé (50%-50%)
  - pas d'examen écrit, rattrapage à l'oral
- **Projet** en binôme (ou individuel)
  - Projet CORBA + REST
  - Rendre code + rapport
- **Exposé** en binôme (ou individuel)
  - **Présentation** de 20 minutes d'une technologie
  - **Démonstration** de 5 minutes sur un exemple **développé par vos soins**
  - **Choix des sujets au plus tard le 30 octobre 2017**
  - Exposés le 27 novembre

PG306
Introduction aux applications réparties

Alexandre Denis – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest
France

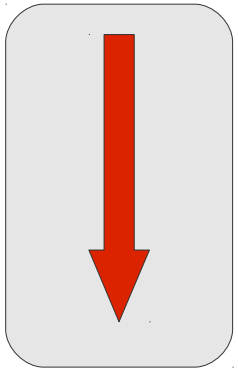
À l'origine...

- Joseph Licklider, département *Command and Control Research*, DARPA
  - Imagine de connecter des machines
  - à l'origine d'ARPANET (1969)
- « *La promesse des ordinateurs en tant que medium de communication entre les gens rendrait insignifiante les origines historiques en tant que machines de calcul* » (1962)
- Deux grandes familles qui s'ignorent
  - Parallélisme, HPC
  - Applications réparties

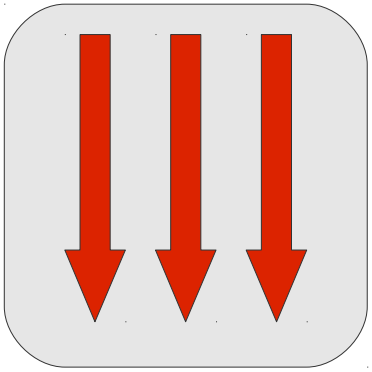
Définitions

Définitions

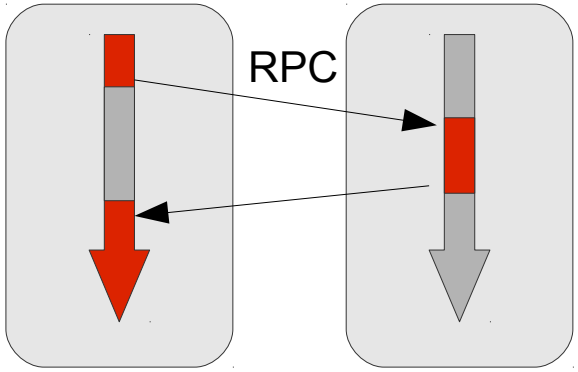
- **Fil d'exécution** :
  - Suite logique d'actions résultant de l'exécution d'un programme
  - i.e. **séquence** d'évènements, qui peut se déplacer au cours du temps
  - Exemple : processus (mono-thread), pthread



Processus mono-thread



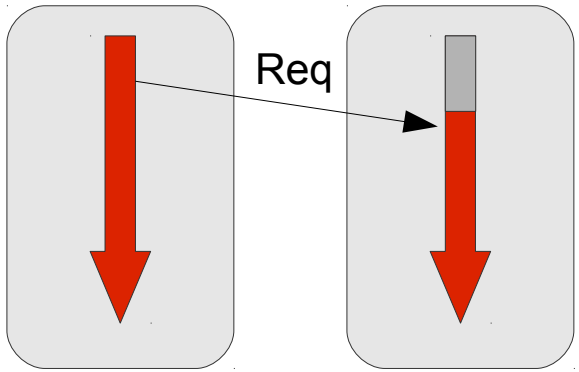
Processus multi-thread :
chaque thread = fil d'exécution



Fil d'exécution qui se déplace

Définitions

- **Système concurrent** :
  - Ensemble, éventuellement variable dans le temps, de fils d'exécution qui interagissent pour parvenir à un but commun
  - i.e. plusieurs choses se déroulent **en même temps**
  - Exemples : machines multi-processeur, machines parallèles, grappes de PC, client(s) + serveur
- **Synchronisation**



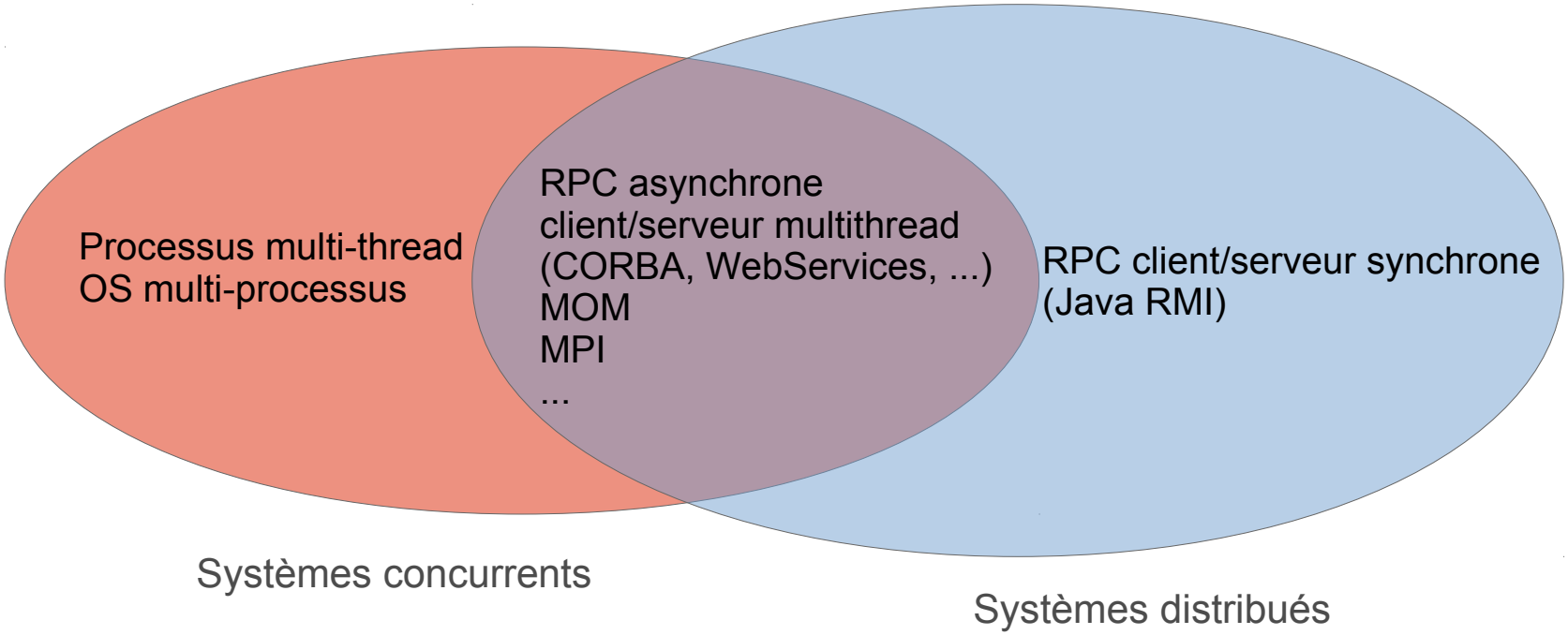
Client + serveur concurrents

Définitions

- **Système distribué** :
  - Plusieurs ressources de calcul sans **mémoire commune**
  - Pas de notion d'échelle ni de répartition géographique
  - Exemple : grappe de PC, client+serveur (navigateur web + serveur http), PC+GPU
- **Communications**

Systèmes concurrents / distribués

•



Définitions

- **Système réparti** : système ***essentiellement*** distribué, par opposition au parallélisme ou l'aspect distribué n'est qu'un artefact
  - C'est un ensemble de processus communicants, répartis sur un réseau de machines, le plus souvent hétérogènes, et coopérant à la résolution d'un problème commun
  - Plusieurs composantes réparties sur plusieurs machines et interconnectées grâce à un intergiciel (wikipedia)
  - A collection of independant computers that appear to the user as a single computer (A. Tanenbaum)
- Répartition sur des mémoires **distribuées**, **communications**, **synchronisation**, **hétérogénéité**

Historique

Historique

- Systèmes intrinsèquement hétérogènes
- Compatibilité attendue entre machines de générations différentes
- Maintenance des applications sur le long terme
- Nécessité d'avoir quelques notions des systèmes et logiciels anciens
- Perspective historique pour comprendre l'écosystème actuel

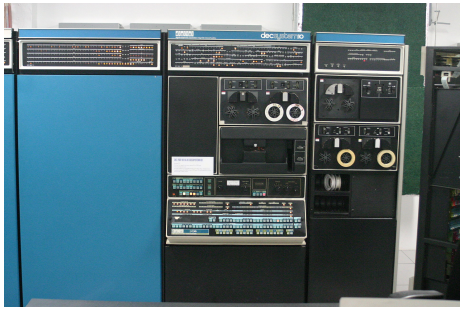
Historique des réseaux

- 1969 : ARPANET, 4 noeuds (56 kb/s)
- 1973 : Ethernet (3 Mb/s)
- 1978 : Compuserve
- 1980 : Ethernet 10 Mb/s
- 1983 : naissance d'Internet (TCP/IP, smtp, ftp, telnet)
- 1990 : World Wide Web (www)
- 1998 : Gigabit Ethernet
- 1999 : ADSL (11 Mb/s)
- 2002 : Ethernet 10 Gb/s
- 2010 : InfiniBand 40 Gb/s
- 2014 : Ethernet 100 Gb/s
- 2017 : InfiniBand 200 Gb/s

Historique des réseaux, que retenir ?

- Augmentation constante des débits des réseaux
- Fossé permanent entre :
  - Les réseaux d'accès (ADSL, VDSL, FTTH)
  - Les réseaux locaux (Ethernet)
  - Les réseaux haute-performance des clusters (Infiniband, ...)
- Les protocoles fondamentaux ont évolué rapidement, puis se sont stabilisés
- Convergence vers TCP / Ethernet

Historique machines et OS (serveurs)



- 1964 : système Multics, langage PL/1
- 1966 : mainframe IBM 360, puis 370, S/390
  - Ancêtre de l'actuelle série Z sous zOS
- 1968 : DEC PDP-10 (1000kB) sous TOPS-10
- 1969 : Unix, système multi-tâche, multi-utilisateur, langage C
- 1984 : Sun SPARC
- 1988 : miniordinateur IBM AS/400 (Application Server)
  - Virtualisation, hyperviseur, code intermédiaire, longue durée de vie des applications
- 1991 : naissance de Linux
- 1995 : Sun ultraSPARC, Solaris
- 1995 : Intel Pentium Pro
- 2005 : AMD Opteron (x86_64)
- 2016 : Intel Xeon Phi (KNL)

Hitorique machines et OS

- Grande diversité de machines & OS
- Convergence vers x86_64 & POSIX
  - Il subsiste des mainframes avec processeurs et OS différent
- Hétérogénéité quand même :
  - Générations différentes
  - Hétérogénéité entre client et serveur

Historique systèmes répartis

- RPC : Sun RPC (1999), RPC v2 (1995)
- OMG : CORBA (1991), **CORBA 2** avec IIOP (1995), **CORBA 3 CCM** (2002), *DDS* (2004)
  - CORBA un peu ancien, mais très utilisé en finance, pilotage de machines industrielles, contrôle aérien, etc.
- Sun Java : JDK 1.0 (1996), **Java RMI** (1997), *J2EE/EJB* (1998), *Spring* (2002), JMS (2001), *JMS 2.0* (2013)
- SGBD : SQL (IBM, ISO en 1996), ODBC (Microsoft, 1992), JDBC (Sun, 1997), *Cassandra* (Apache, 2010), Hbase (2009), MongoDB (2010)
- Microsoft : COM (1993), DCOM (1996), *.NET* (2002)
- Web Services : XML-RPC (1998), **SOAP** (2003), **REST** (~2000), **JAX-RS** (2008)
- Big Data : Google MapReduce & GoogleFS (2004), **Hadoop** (2009), *Apache Spark* (2014)

Intergiciels & logiciels pour applications réparties

Code métier

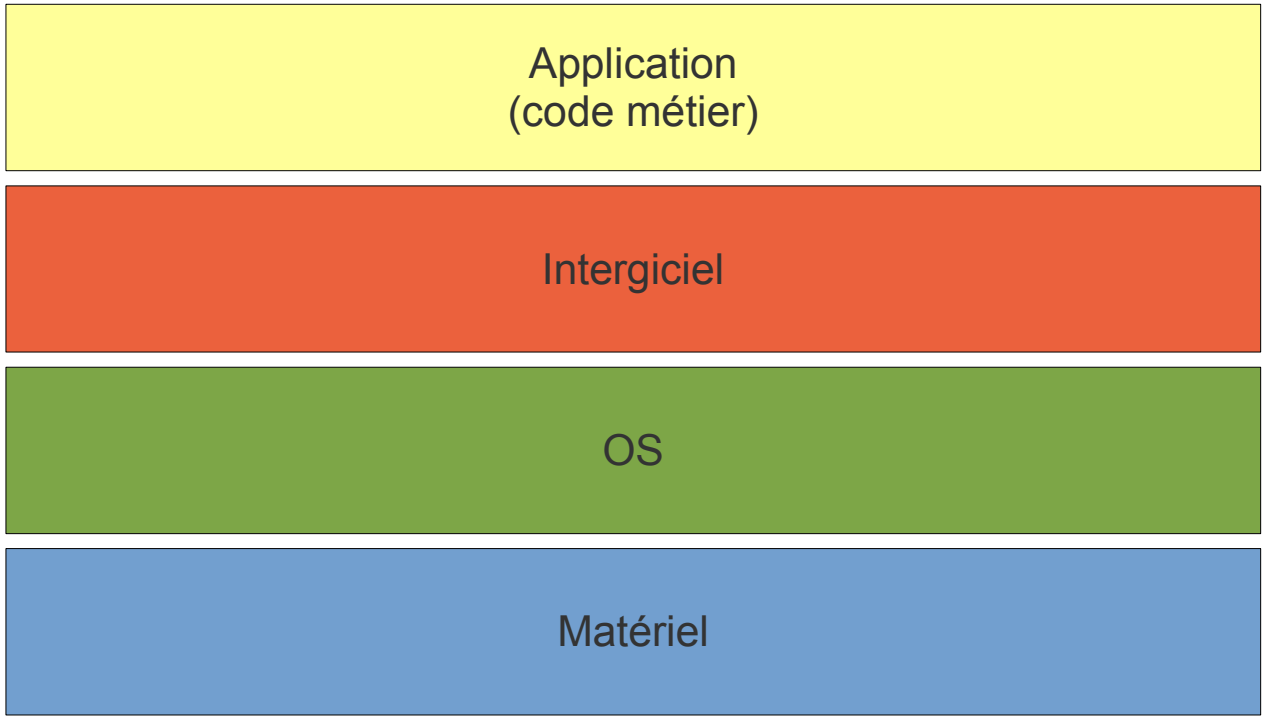
- **Code métier** : réalise le coeur de métier de l'application
  - Calcul, manipulation de données, etc.
- **Propriétés non-fonctionnelles** : code non-spécifique à l'application
  - Communications
  - Conversions de données
  - Cycle de vie
  - Persistance
  - Sécurité
  - Annuaire

Intergiciel

- L'intergiciel (*middleware*)
  - Factorise le code transversal
  - Masque/ gère la complexité de l'infrastructure sous-jacente
    - Gère l'hétérogénéité (conversions de données, indépendance p/r OS, portabilité code)
    - Gère les communications
  - Fournit les services de base (annuaire, localisation, transactions, etc.)
- Dans la pile logicielle, le code métier (application) est souvent le plus petit/simple
  - ex.: ORB 200k sloc, MPI 500k sloc, JDK 3M sloc

Pile logicielle

•

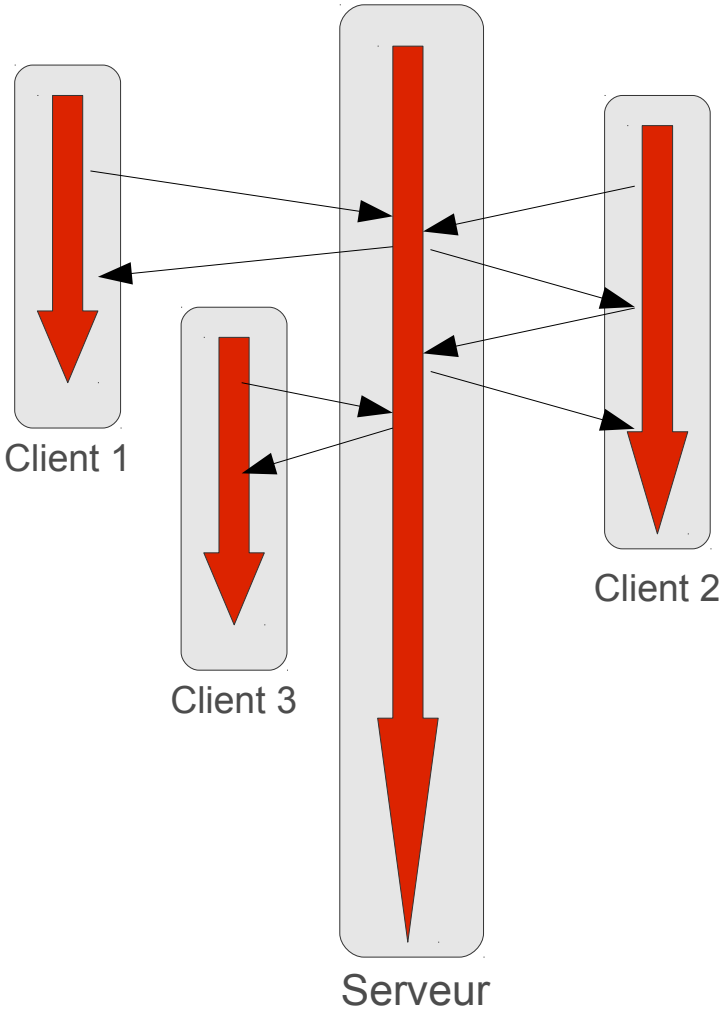


Approches

- **Protocole brut** : l'interface est le protocole
  - Implémentation à la main dans l'application, ou utilisation de bibliothèques tierces
  - Exemple : REST
- **Intergiciel**
  - Bibliothèque utilisée par le code applicatif
  - Exemple : CORBA
- **Annotations**
  - Ajout de pragma dans le code, le compilateur génère le code
  - Exemples : JAX-RS, OpenMP
- **Framework** (cadriciel, canevas)
  - Inversion du contrôle
  - Génération de code, remplissage de canevas à trous, héritage de classes
  - Exemple : Hadoop

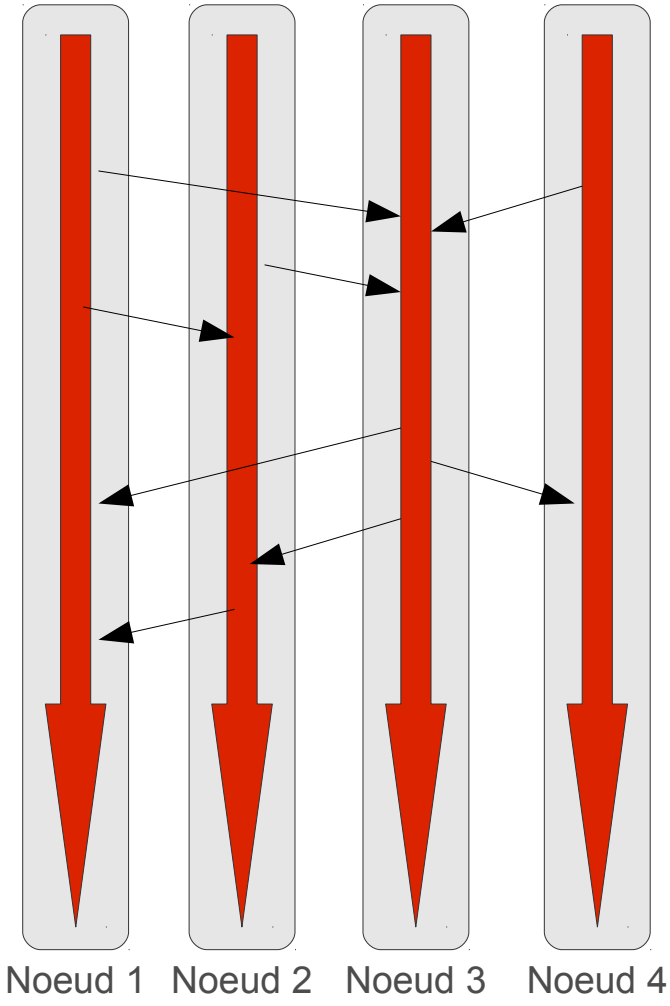
Topologies des systèmes répartis

- **Client / serveur**
  - Serveur statique
  - Clients dynamiques
  - Exemples : web, Web Services, CORBA



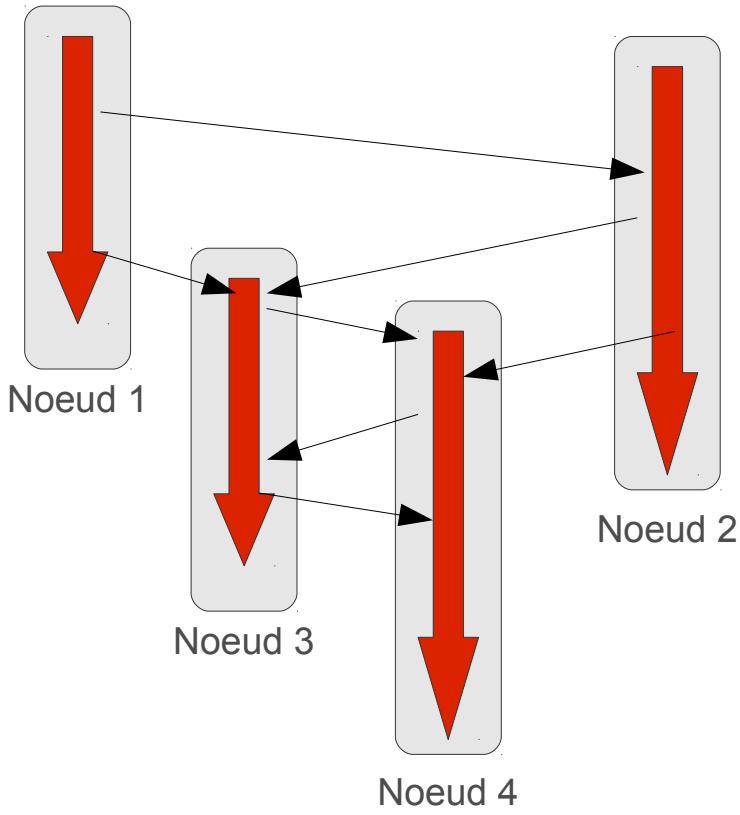
Topologies des systèmes répartis

- Client / serveur
- **Fédération (cluster)**
  - Ensemble ~statique
  - Monde fermé
  - Exemples : Hadoop, MPI



Topologies des systèmes répartis

- Client / serveur
- Fédération (cluster)
- **Pair à pair**
  - Entièrement dynamique
  - Tout le monde peut être client et serveur
    - Tous les noeuds ne le sont pas forcément (architectures hierarchiques)
  - Connexion directe entre pairs qui communiquent
  - Pas de connexion de tout le monde vers tout le monde
  - ~ Connexion paresseuse



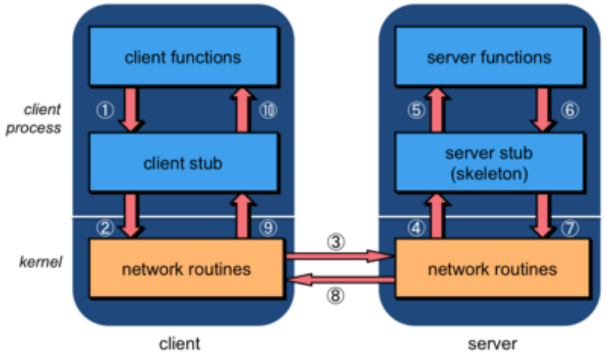
Paradigmes de communication

Paradigme de communication

- Façon d'intégrer les communications dans le fil d'exécution
- Différents critères :
  - Synchrone (ex.: RMI) / asynchrone (MOM)
    - Notion de point de synchronisation (= barrière)
  - Couplage fort / couplage faible
  - Architecture générale des applications
  - "Conteneur" de ce qui est partagé
    - = entité de distribution
    - ex.: fonction, objet, composant, service, etc.

Paradigme RPC

- RPC = Remote Procedure Call
  - Entité de distribution = programme (processus)
  - Entité de désignation = procédure (fonction)
  - ex.: Sun RPC, DCE RPC, XML-RPC
- Transparence assurée par des souches (stubs)
  - Interception locale de l'appel de procédure par la souche
  - Sérialisation (*marshalling*) des arguments et transport vers le serveur



Sun RPC

- Représentation des données portable à travers le réseau
  - XDR : eXternal Data Representation
- Accès aux services via le portmapper
- Description d'un programme en langage RPCL (fichier `.x`)
  - Syntaxe proche du C
  - Un programme fournit plusieurs procédures
  - Un client effectue à distance un appel **synchrone** & **bloquant** d'une procédure d'un programme
- Contraintes
  - **Numéro** de programme, **numéro** de procédure, **un seul argument** par procédure, ...
- Ancêtre de tous les systèmes de RPC, aujourd'hui tombé en désuétude

Paradigme RMI

- RMI = Remote Method Invocation
  - Entité de désignation et de distribution = objet
  - ~ RPC orienté objet (encapsulation, héritage)
  - ex.: Smalltalk, Java RMI, CORBA
- Architecture générale
  - Serveurs, hébergeant des **objets distribués**
  - Distinction entre objet distribué (abstrait) et objet de l'implémentation (servant)
  - **Bus logiciel** (*broker*) routant les requêtes
  - Clients invoquant des méthodes sur les objets distants
- Cf. présentations Java RMI et CORBA

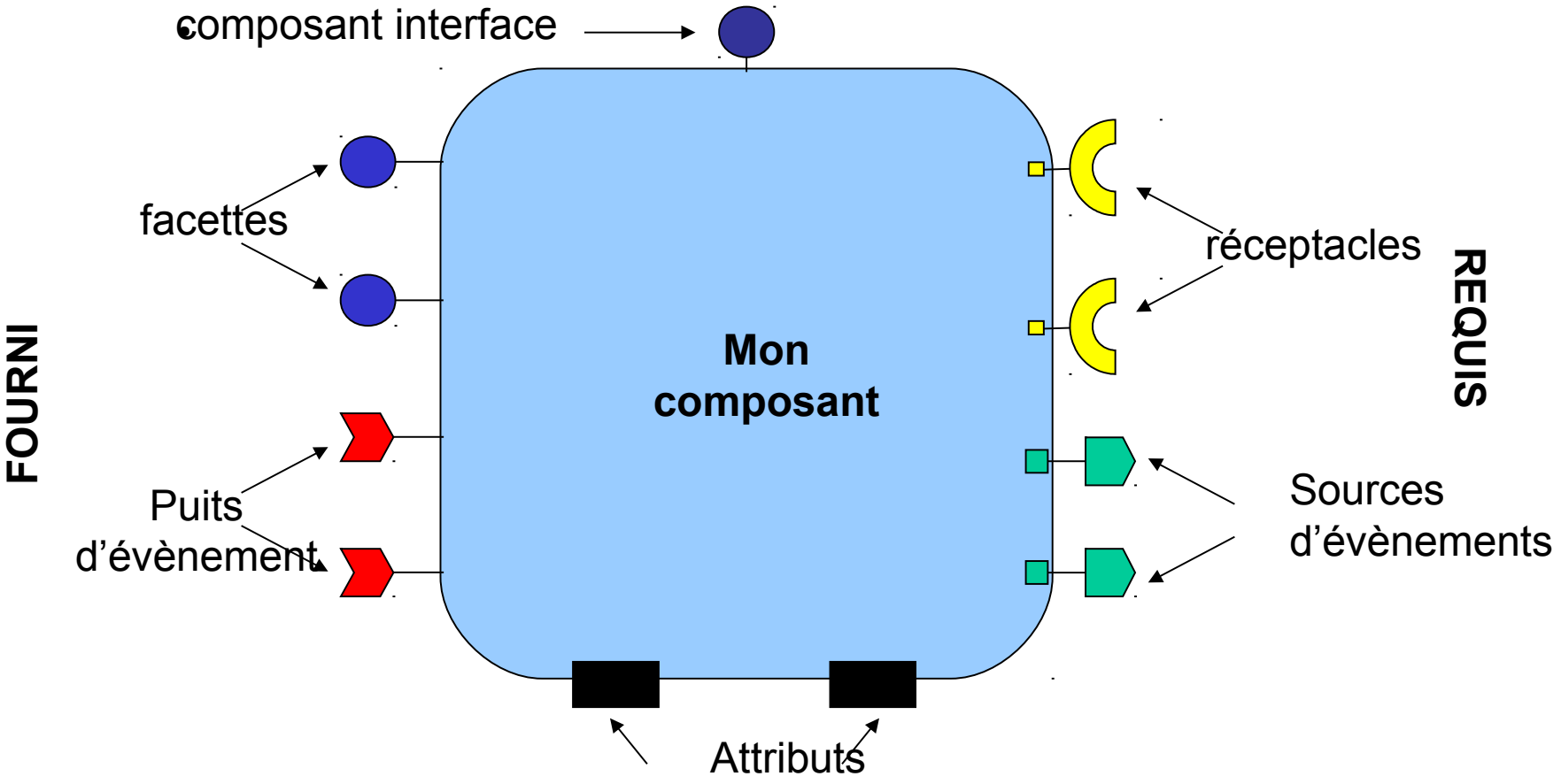
Paradigme Composants logiciels

- Motivation : limites du modèle à objets
  - Pas d'expression de ce qui est requis
  - Pas de vue globale, pas de gestion du déploiement
- Définition : « *Un composant logiciel est une **unité de composition** dotée d'**interfaces spécifiées**. Un composant logiciel peut être déployé indépendamment et sujet à une **composition par une tierce entité*** » (Szyperski & Pfister, 1997)
- Ex.: EJB (Entreprise Java Beans), CCM (CORBA 3), DCOM (Microsoft), .NET

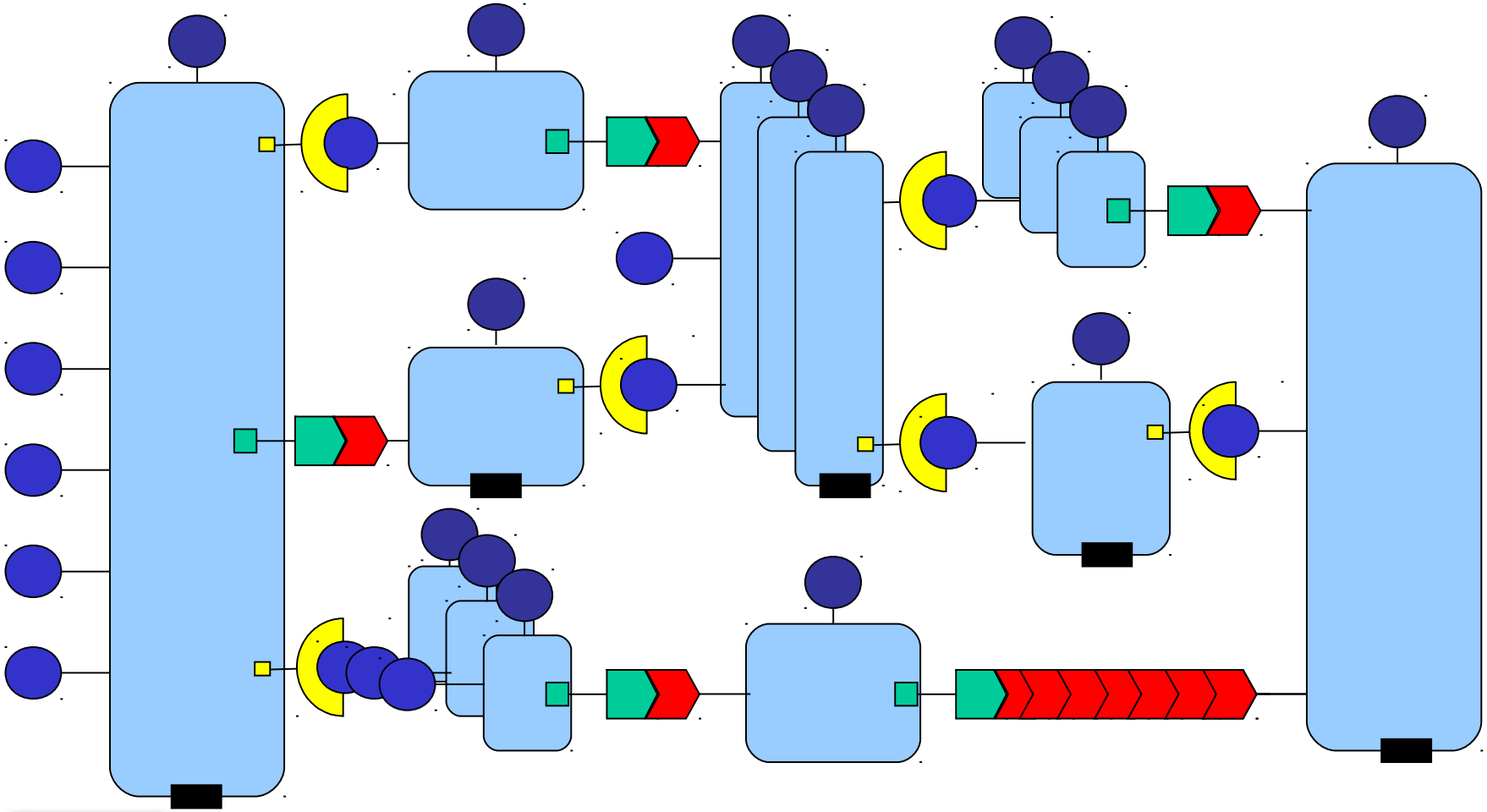
Description d'un composant

- Un modèle de composant est défini par :
  - Ce qu'offre un composant aux autres composants (***provide*** = *facette*)
  - Ce qu'il demande des autres composants (***use*** = *réceptacle*)
  - Le modèle de collaboration utilisé entre composant
    - **Synchrone** via les invocations d'opération
    - **Asynchrone** via la notification d'évènement
  - Quelles propriétés du composant sont configurables (***attributs***)
  - Quelles sont les opérations du cycle de vie (***home***)

Un composant logiciel



Construction d'une application = assemblage



Propriétés des composants

- Assemblage de composants
  - Application -> déploiement
  - Composite (= composant)
- Par rapport à un modèle objet
  - ++ évolutivité, réutilisabilité
    - briques réellement interchangeables grâce aux réceptacles et attributs explicites
  - ++ vraie encapsulation
  - -- surcoût en performance (indirections)
  - -- complexité de mise en oeuvre
    - Outil de déploiement, conception de code réellement encapsulé, code chargeable dynamiquement, ...

Paradigme MOM

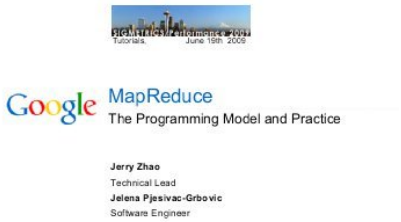
- MOM = Message Oriented Middleware
  - Échange de messages
  - send/recv asynchrone, file de messages
  - Modèles d'évènements : pusblish/subscribe, push/pull
  - **Couplage faible**
- Ex.:
  - Standard AMQP (JPMorgan)
    - Apache Qpid, RabbitMQ, zeroMQ
  - Sun JMS (Java Message Service)
  - Microsoft MSMQ
  - IBM MQSeries (WebSphere)
  - Jboss HornetQ
  - CORBA Event Service
  - OMG DDS

Paradigmes du Web

- Pas un paradigme de communication en tant que tel
- Interfaces et protocoles de type web
- ex.: Web Services (SOAP, XML-RPC), REST

Big Data

- Traiter de gros volumes de données
- Opérations « simples »
- Les données ne bougent pas, on amène le traitement aux données
- Domaines d'application : fouille de données, recherche, statistiques, indexation, etc.
  - Google, Yahoo, Facebook, Amazon, etc.
- Ex.: MapReduce, Hadoop, Spark



Problématiques transverses

Transparence réseau

- Transparence = masquer la répartition des données, des traitements
  - Donner l'illusion d'invocations locales
  - Approche transparente : stubs (RPC, RMI)
  - Approche à communications explicites : REST, MOM
- Le réparti doit gérer des problématiques non-exprimables en local
  - Transparence totale impossible
  - Ex.: localisation des objets, exceptions causées par le réseau, etc.

Reconfiguration dynamique

- Connexions & déconnexions dynamiques
- Tolérance aux fautes, aux défaillances
  - À grande échelle, faute = événement habituel
  - Gestion des défaillances obligatoire
  - checkpoint/restart, réplication
  - Réaction aux exceptions

Connectivité réseau

- IP fut un grand pas en avant pour l'interconnexion des réseaux...
  - jusqu'à l'invention des firewalls !
  - Firewall, firewall à état
  - Réseaux privés, adresses non-routables
- Solution grâce à IPv6 ?... firewalls quand même...
- Routage au niveau utilisateur, **réseau d'overlay**
- Utilisation de proxy (http, port 80)
  - À l'origine de la popularité des services web
  - Proxy « intelligent » interprétant le flux de données
  - -> fuite en avant...

Hétérogénéité

- Hétérogénéité intrinsèque en réparti
  - **Interopérabilité** : machines différentes connectées
  - **Portabilité** : même code sur différentes machines
- Importance des **interfaces** (= *contrat*)
  - IDL, WSDL, etc.
  - Interface = typage mais aussi **sémantique**
- Représentation des données (XDR, CDR, XML, JSON, etc.)
  - Compromis : efficacité, lisibilité
- Standards **ouverts**, interfaces **normalisées**
  - Seule garantie pour l'interopérabilité et la **pérennité**

Déploiement et cycle de vie

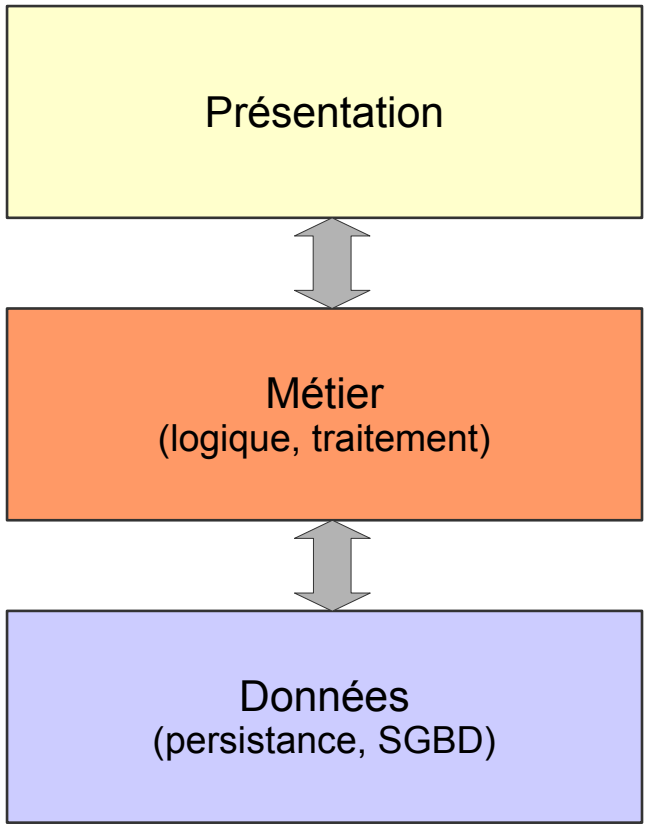
- Déploiement des applications
  - **Sélection** des ressources
  - **Configuration**, adaptation aux ressources
  - **Copie** des applications & données
  - **Lancement** des binaires
- **Intégré** aux plate-formes de composants
- **Implicite** pour les services (serveurs d'application, déploiement codé en dur)
- **Ad-hoc** en objets répartis
- Localisation, nommage, annuaire, résolution de noms
  - RMI, CORBA, WS-* ont un système d'annuaire

Intégration et codes patrimoniaux

- Intégration de codes écrits par des équipes différentes
  - Ex.: plate-forme d'intégration *SALOME* (CORBA)
- Encapsulation de codes pré-existants (patrimoniaux)
- Écriture de code réutilisables
  - Ex.: composants logiciels (CCM, EJB, etc.)
- Technologies du réparti utilisées pour encapsuler, même parfois en non-réparti
  - Ex.: CORBA pour interfacer des codes C++, Ada, COBOL, Java, Python, VB, etc.

Encapsulation : architecture 3 tiers

- Architecture 3 tiers
  - **Séparation** des couches
  - **Interfaces** bien définies
  - Meilleure maintenance
  - Poste client léger



Sécurité des applications réparties

- Aspects de la sécurité :
  - **Confidentialité** : non-interception des données
  - **Authentification** : savoir à qui on parle
  - **Non-intrusion** : prise de contrôle d'une machine à distance
- Problématiques liées
  - Interception de mot de passe/certificat -> intrusion
  - Pas d'authentification d'un client RPC -> exécution de code arbitraire

Solutions de sécurité

- Authentification/confidentialité
  - Certificats et chiffrement SSL/TLS
  - Géré nativement dans l'intergiciel
- Contrôle d'exécution (qui peut exécuter quoi ?)
  - Solution ad-hoc, de niveau applicatif
  - **Risque élevé de faille**
- **Délégation** d'authentification
  - Autoriser un tiers à agir en notre nom
  - Ex.: autoriser Facebook à accéder à nos photos Instagram
  - Oauth (RFC IETF) : délégation d'authentification par token
    - Sans donner login/mot de passe
    - Révocable

Organisations Virtuelles (VO)

- Pour gérer une fédération d'infrastructures mises en commun
- Autorité locale (chacun est root sur ses propres machines)
- Utilisateurs identifiés par un certificat (X.509)
- VO = groupe, projet, équipe, consortium, etc.
  - Est composée d'utilisateur
  - Accès aux ressources autorisé en fonction des VO
  - Un utilisateur peut être dans plusieurs VO
- Pas de « chef unique »
  - Délégation d'autorité
  - Confiance mutuelle entre les sites d'une VO
  - Chaque site garde son autonomie
  - Un site peut être dans plusieurs VO
- Ex.: Globus

http://dept-info.labri.fr/~denis/