Quatrième partie

Intergiciels à message
Message Oriented Middleware

Plan

La communication asynchrone par message

Objectif : exploiter les possibilités d'une communication asynchrone

Avantages

- Évite le blocage de l'appelant inhérent à l'appel procédural
- Découple l'envoi de la réception
- Récepteur(s) anonyme(s)
- Étend le simple protocole point-à-point
- Autorise une communication de type publication/abonnement (publish/subscribe) : protocole $m \rightarrow n$

Difficulté

- Programmation délicate car asynchrone :
  Approche événementielle : Événement $\rightarrow$ Réaction

Interaction par messages

Modèle élémentaire

Send(message, destination)
Receive(message, source)

Synchrone/asynchrone

- Communication synchrone : rendez-vous entre émission et réception bloquantes
- Communication asynchrone :
  - Émission non bloquante
  - Réception bloquante, non déterministe

Paramètres

Désignation

- Communication point à point entre activités (canaux)
- Communication indirecte : passage par une boite à lettres
  - attachée à un processus ($n \rightarrow 1$: port, porte)
  - partagée par plusieurs processus ($n \rightarrow m$: file de messages)
- Statique (ex : IPv4) ou dynamique

Propriétés du service de communication

- Fiabilité (perte)
- Intégrité
- Qualité de service (débit, latence, gigue)
- Ordonnancement relatif des réceptions par rapport aux émissions (p.e. fifo)

Communication asynchrone par messages

- Envoi asynchrone : l'émetteur ne se bloque pas
- Réception sélective à la demande du récepteur
- Diffusion possible d'un message à plusieurs récepteurs
- Couplage minimal entre émetteur et récepteur :
  - L'émetteur ne connaît pas le(s) récepteur(s) : il publie
  - Un récepteur doit explicitement souscrire pour recevoir
  - Un récepteur contrôle à quel moment il accepte
- Le récepteur peut ne pas être actif (présent) lorsque le message est envoyé

Usage : Architectures logicielles à composants

Exemple : supervision d'un réseau

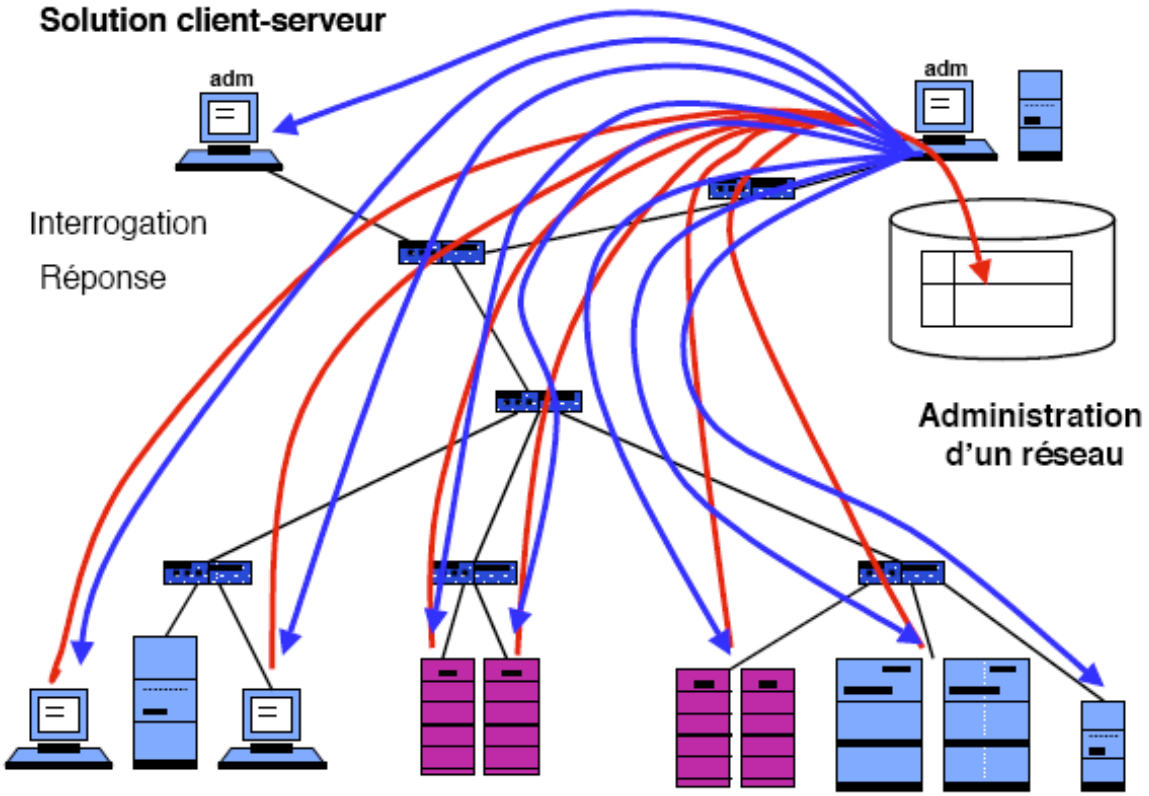
Contexte

- Surveillance de l'état de machines, systèmes et applications dans un environnement distribué
- Flot permanent de données en provenance de sources diverses sur le réseau
- Possibilité permanente d'évolution (ajout, suppression, déplacement des équipements)
- Possibilité d'accès des administrateurs depuis n'importe quel poste de travail

Objectifs

- suivi des changements de configuration dynamiques
- émission de messages signalant les changements d'état et les mises à jour
- statistiques, journal de fonctionnement

Approche client-serveur



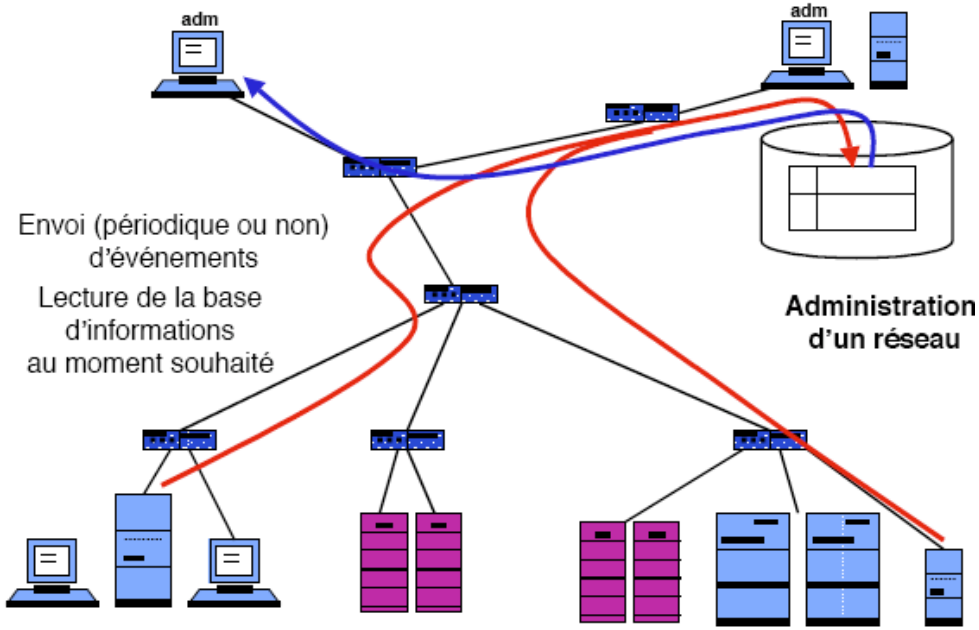
- Interactions synchrones
- Communication essentiellement 1 vers 1 (ou n vers 1)
- Entités (clients, serveurs) désignées explicitement
- Organisation de l'application plutôt statique

Approche client-serveur par inversion de contrôle

Client-serveur avec inversion du contrôle

- Le service d'administration s'abonne auprès des clients sur les événements qui l'intéressent
- Les clients contactent le service d'administration en cas de tel événement
- $\Rightarrow$ mécanisme de « callback »
- Entités (clients, serveurs) désignées explicitement
- Organisation de l'application plutôt statique
- Découverte de nouveaux équipements ?

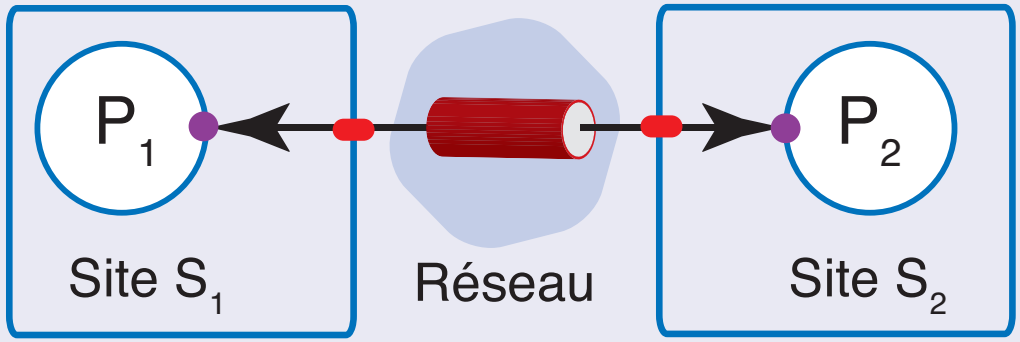
Approche MOM



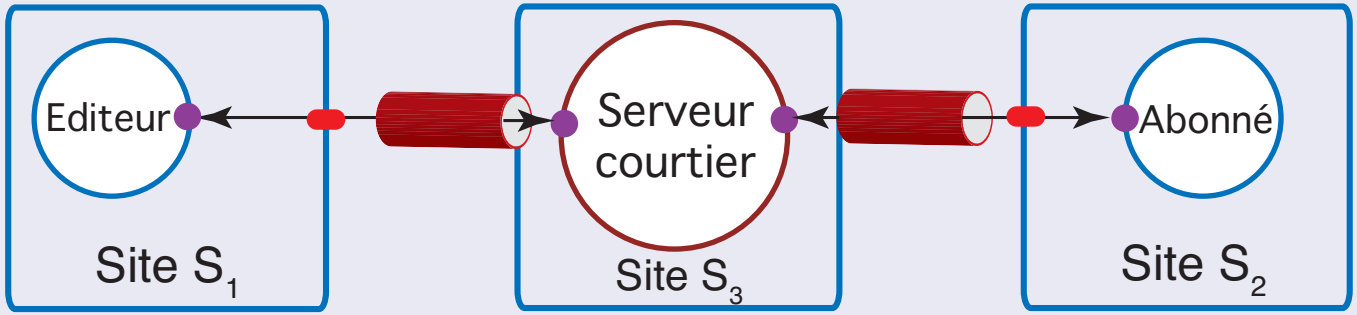
- Les composants administrés émettent des messages :
  - changements d'état et de configuration, d'alertes
  - horloge (relevé périodique, statistiques)
- Des processus cycliques (démons) mettent à jour l'état du système à partir des notifications reçues
- Inversion des rôles des producteurs et des consommateurs de données

Communication indirecte

Communication directe



Communication indirecte



Queue vs Sujet (Topic)

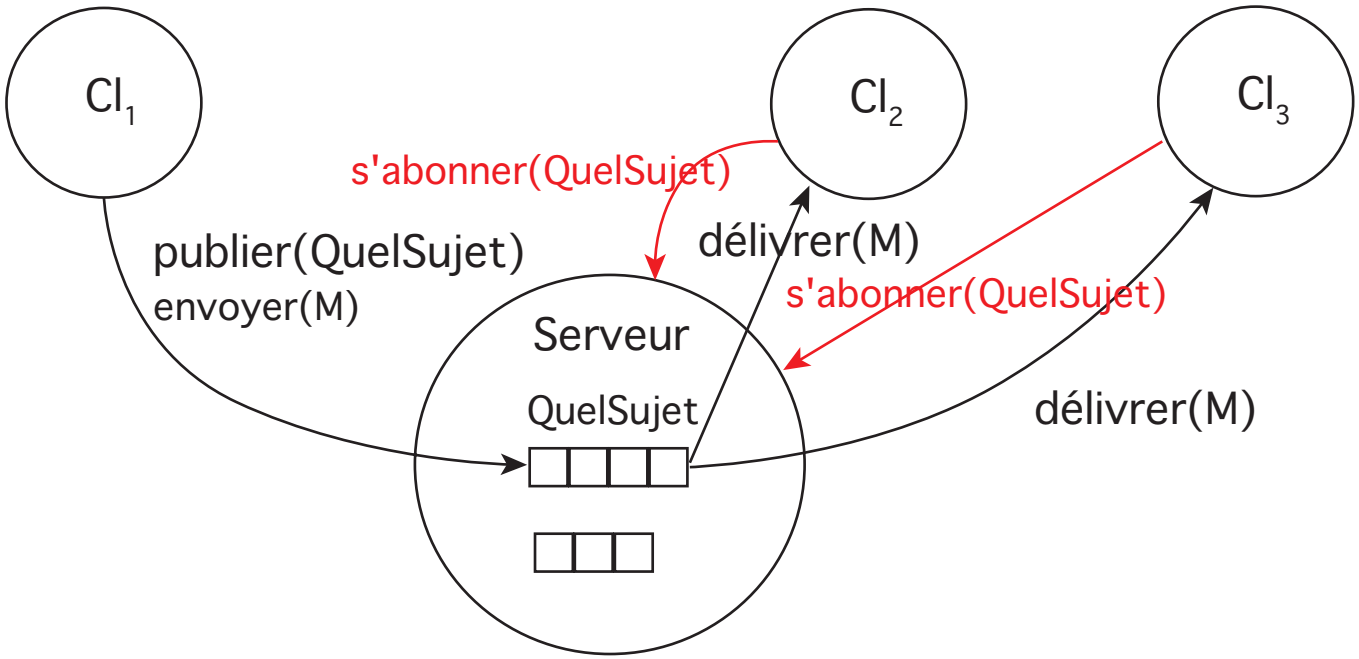
File de messages / Message Queue

- Interface : ajouter dans la file, retirer dans la file
- Plusieurs producteurs, plusieurs consommateurs
- Retrait destructif
- Persistance
- Découplage temporel production / consommation

Publication – abonnement / Publish – subscribe

- Interface : envoyer un message, obtenir un message
- Plusieurs producteurs, plusieurs consommateurs
- Retrait non destructif, mais chaque consommateur n'obtient qu'au plus une fois chaque message
- Abonnement pour délivrance asynchrone

L'échange par publication/abonnement (publish/subscribe)



- Précédence temporelle : on ne peut recevoir que ce qui a été publié après s'être abonné (contrairement à une queue)

Modèle requête/réponse

Modèle client-serveur avec des messages :

- Une file de requêtes par serveur
- Une file de réponse par client
- Une requête identifie la file de réponse à utiliser
- Client :
  1. envoyer message de requête sur la file de requête
  2. attendre message de reponse sur sa file de réponse
- Serveur :
  1. attendre message de requête sur la file de requête
  2. traiter la requête
  3. envoyer message de reponse sur la file de réponse identifiée dans la requête

Domaine d'application : systèmes faiblement couplés

- Découplage temporel : interactions asynchrones / systèmes autonomes communicants
  - Communication en mode « push » : découverte des évolutions de l'environnement
  - Fonctionnement en mode déconnecté : site absent ou utilisateur mobile
- Découplage spatial : systèmes à grande échelle
  - Fonctionnement en mode partitionné / déconnecté
  - Communication « anonyme »
  - Communication *n*–*m*
- Découplage sémantique : systèmes hétérogènes
  Modèle d'interaction minimal → possibilité d'intégrer des environnements (systèmes, réseaux) / applications hétérogènes

Plan

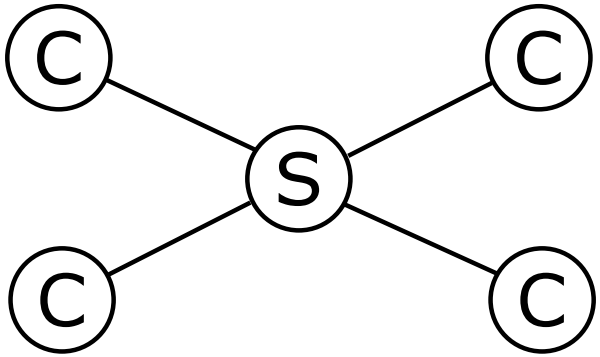
Les intergiciels à messages
MOM : Message-Oriented Middleware

Architecture et fonctionnalités de base

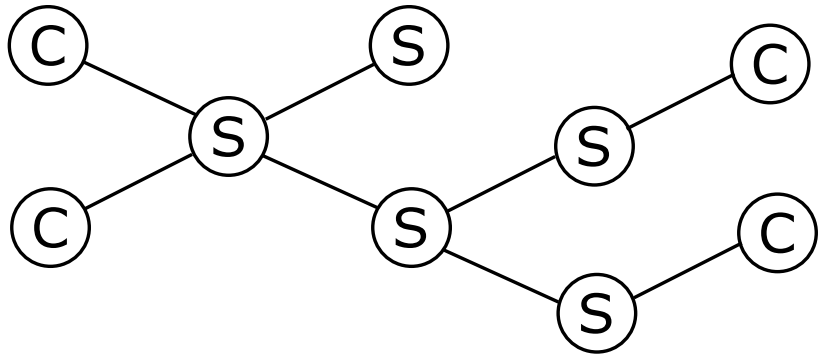
- Les clients (applicatifs) s'adressent à un serveur
- Ils envoient/recoivent leurs messages au(x) serveur(s)
- Un service courtier (broker ou provider) sert d'intermédiaire pour stocker et router les messages vers leurs destinataires
- Le protocole d'échange peut être de type publier/s'abonner (publish/subscribe)
- Critère de réception par le contenu, par le sujet (topic)
- Gestion de la persistance des messages
- Communication point-à-point possible

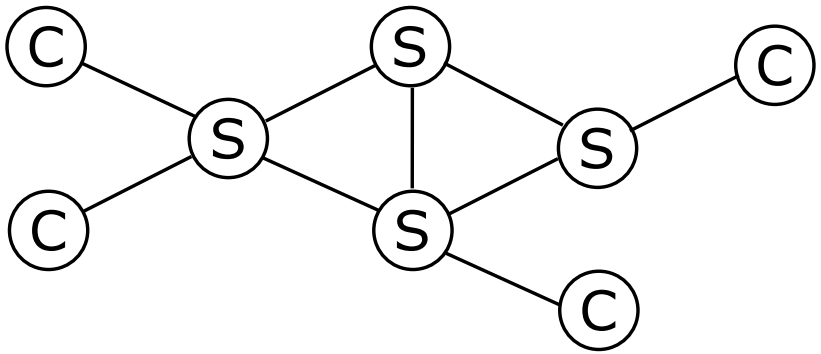
Architectures du service courtier

Centralisée

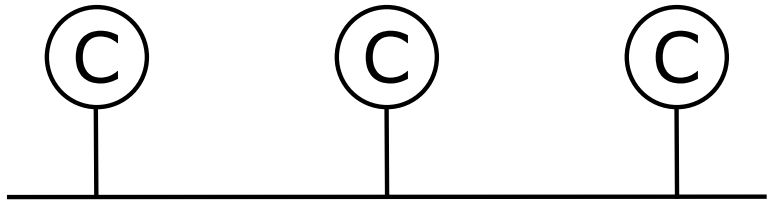


Snowflake





Distribuée



Bus logiciel

Intergiciel à messages (suite)

Les éléments d'un intergiciel à message

- Service de gestion des messages implanté par un ou plusieurs serveurs (providers)
- Une API client
- Les objets messages pour communiquer

La notion de message

- l'en-tête ou header qui contient les informations de gestion :
  - file destinataire, identifiant du message
  - dates de disponibilité, d'échéance, ...
- les propriétés : suite de couples (clé,valeur) précisant le contenu du message
- les données applicatives (charge utile ou payload)

Intergiciel à messages (suite)

Fonctionnalités complémentaires

- Définition de priorités entre messages
- Compression des données utiles du messsage
- Échéance maximale pour recevoir un message
- Publication à date minimale fixée
- Routage des messages d'un serveur à l'autre (forward)
- Lancement d'applications lorsque des messages sont disponibles pour elles
- Possibilité d'alertes sur critères :
  - Présence de messages dans une file donnée
  - Nombre de messages présents

La standardisation et les produits MOM

- API standard pour Java : JMS (Java Message Service)
- Quelques MOM :
  - Open Message Queue (intégré dans GlassFish, implantation de référence)
  - ActiveMQ de Apache
  - Joram de l'INRIA (intégré dans Jonas)
  - OpenJMS
- Autres standards :
  - HLA (High-Level Architecture) pour interconnecter des simulateurs
  - XMPP (eXtensible Messaging and Presence Protocol)
  - AMQP (Advanced Message Queuing Protocol)
  - Microsoft Message Queuing (MSMQ)

Apache Kafka

A distributed streaming platform

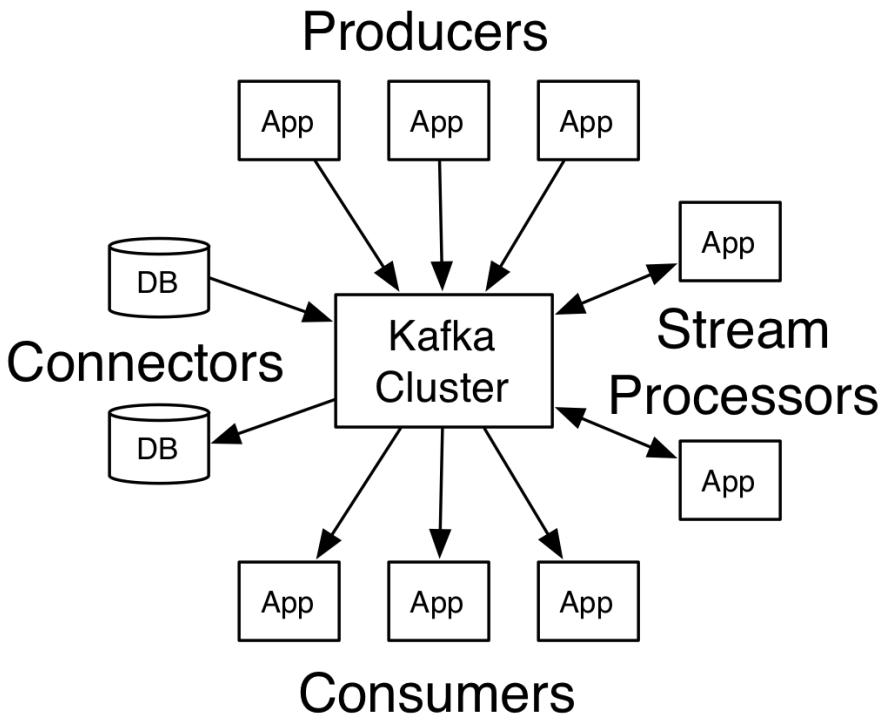
- Distribution, stockage et traitement de flux de données
- Points forts : haut débit, latence faible, tolérant aux fautes
  $\rightarrow$ traitement de gros flux de données temps réel

Les données

- Un flux est identifié par un sujet (*topic*)
- Un flux est constitué d'enregistrements (*record*)
- Un enregistrement contient une clef, une valeur et une date (*timestamp*)

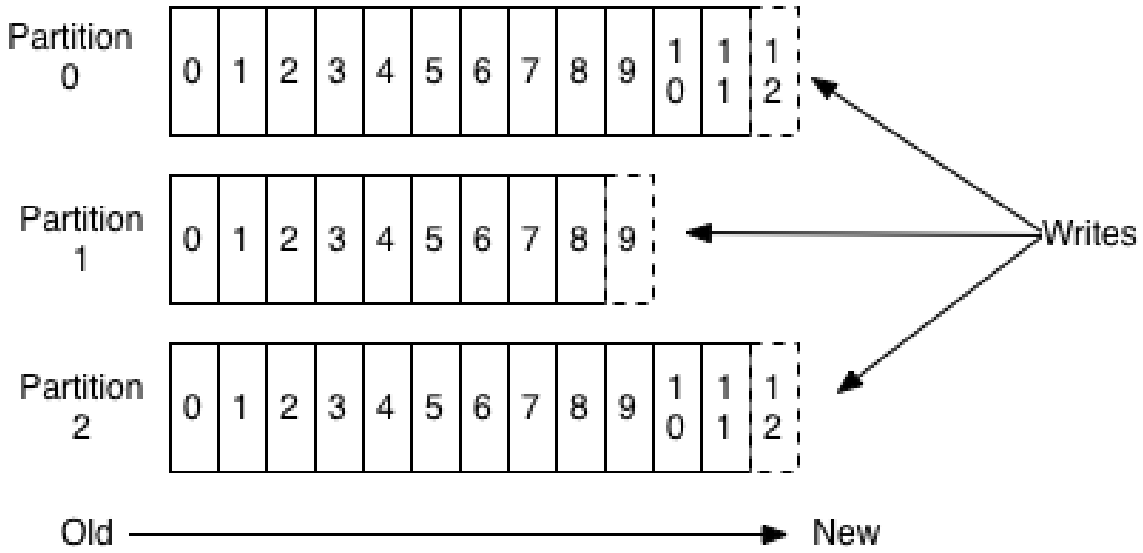
Les APIs

- Producer : publication d'enregistrements
- Consumer : abonnement à des topics et obtention d'enregistrements
- Streams : transformation de flux, un ou des flux en entrée, un ou des flux en sortie
- Connector : vision flux de BD



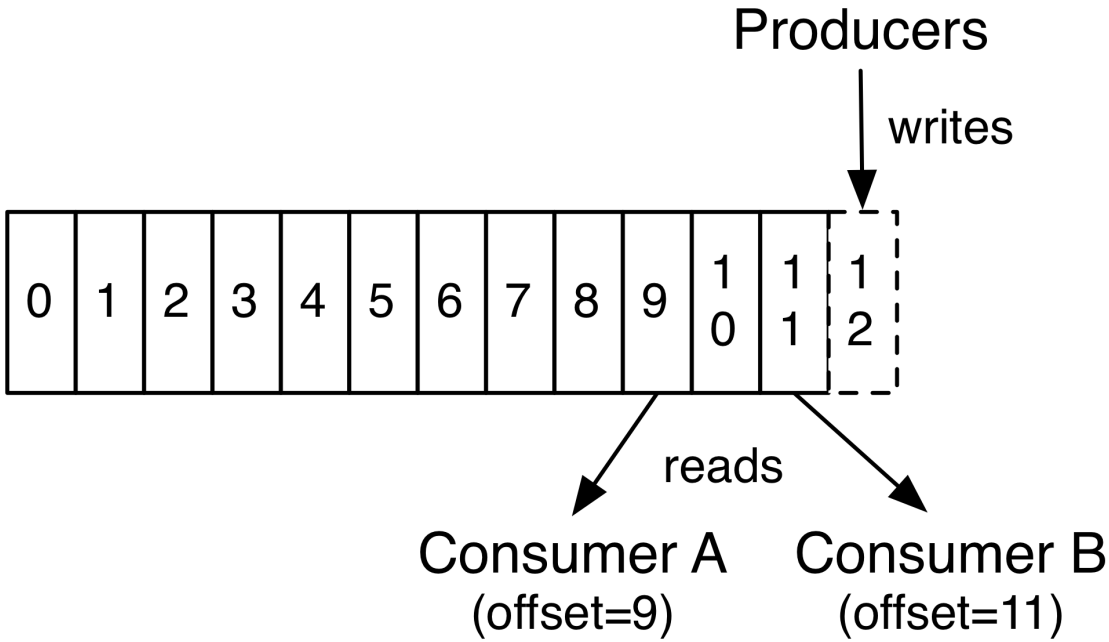
source : https://kafka.apache.org

Topic et partitions



- Un topic est multi-producteurs, multi-consommateurs
- Un topic est décomposé en partitions → parallélisation
- Chaque partition contient une séquence ordonnée et immuable d'enregistrements, strictement croissante
- Immuable : pas d'effacement à la consommation (mais possibilité de délai de rétention avec oubli)

Consommation

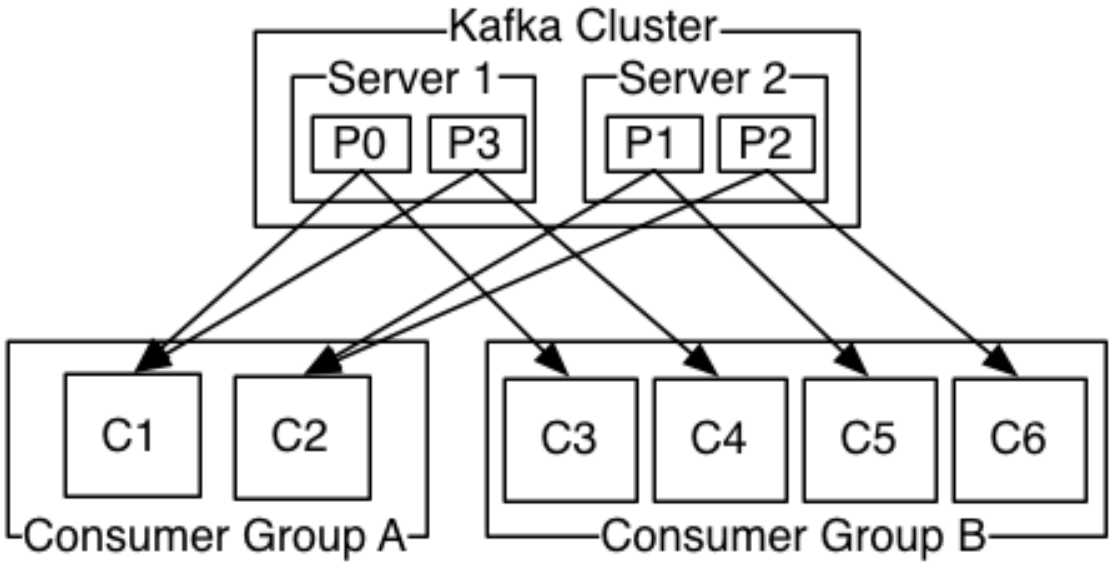


- Les producteurs d'une partition ajoutent en queue
- Chaque consommateur a un offset, qu'il peut changer arbitrairement (pour retourner en arrière ou pour sauter aux plus récents enregistrements)

Distribution & tolérance aux fautes

- Un *Kafka cluster* est un ensemble de serveurs sur un ensemble de machines
- Chaque partition est répliquée sur plusieurs serveurs
- Chaque partition possède un serveur leader et des serveurs de secours
- Le leader gère toutes les lectures et écritures, les secours suivent passivement
- Un algorithme d'élection choisit un nouveau leader en cas de défaillance (cf cours *Systèmes répartis*)
- En pratique, un serveur leader d'une partition est secours pour d'autres partitions du même ou d'autres topics

Architecture globale



- Un producteur publie dans une partition d'un topic
- Les consommateurs forment des groupes.
- Un groupe reçoit depuis toutes les partitions, mais un enregistrement n'est délivré qu'une fois par groupe (à un seul consommateur)
  $\rightarrow$ partage de charge

Traitement de flux

- Séparation (*branch*) : scission d'un flux en plusieurs flux
- Filtrage : flux limité aux enregistrements vérifiant un prédicat
- *map* : applique une transformation à chaque enregistrement
- *flatmap* : applique une transformation à chaque enregistrement, pour produire 0, 1 ou plusieurs enregistrements
- Agrégation des enregistrements ayant la même clef en un unique enregistrement
- Comptage des enregistrements d'une même clef
- Jointure de plusieurs flux

```
builder.stream("clients-topic")
    .map((nom, date) -> KeyValue.pair(nom.toLowerCase(),date))
    .groupByKey()
    .count()
    .filter((nom, count) -> count > 100)
    .to("clients-fidèles-topic");
```

Plusieurs points de vue

Kafka est ...

- un intergiciel à messages, unifiant queue et abonnement, avec objectif de performance (de Kafka et des applications l'utilisant) $\rightarrow$ parallélisation
- un système de stockage d'informations incrémentales, fiable (réplication) et performant
- un système de traitement de flux

Plan

Le standard JMS (Java Message Service)

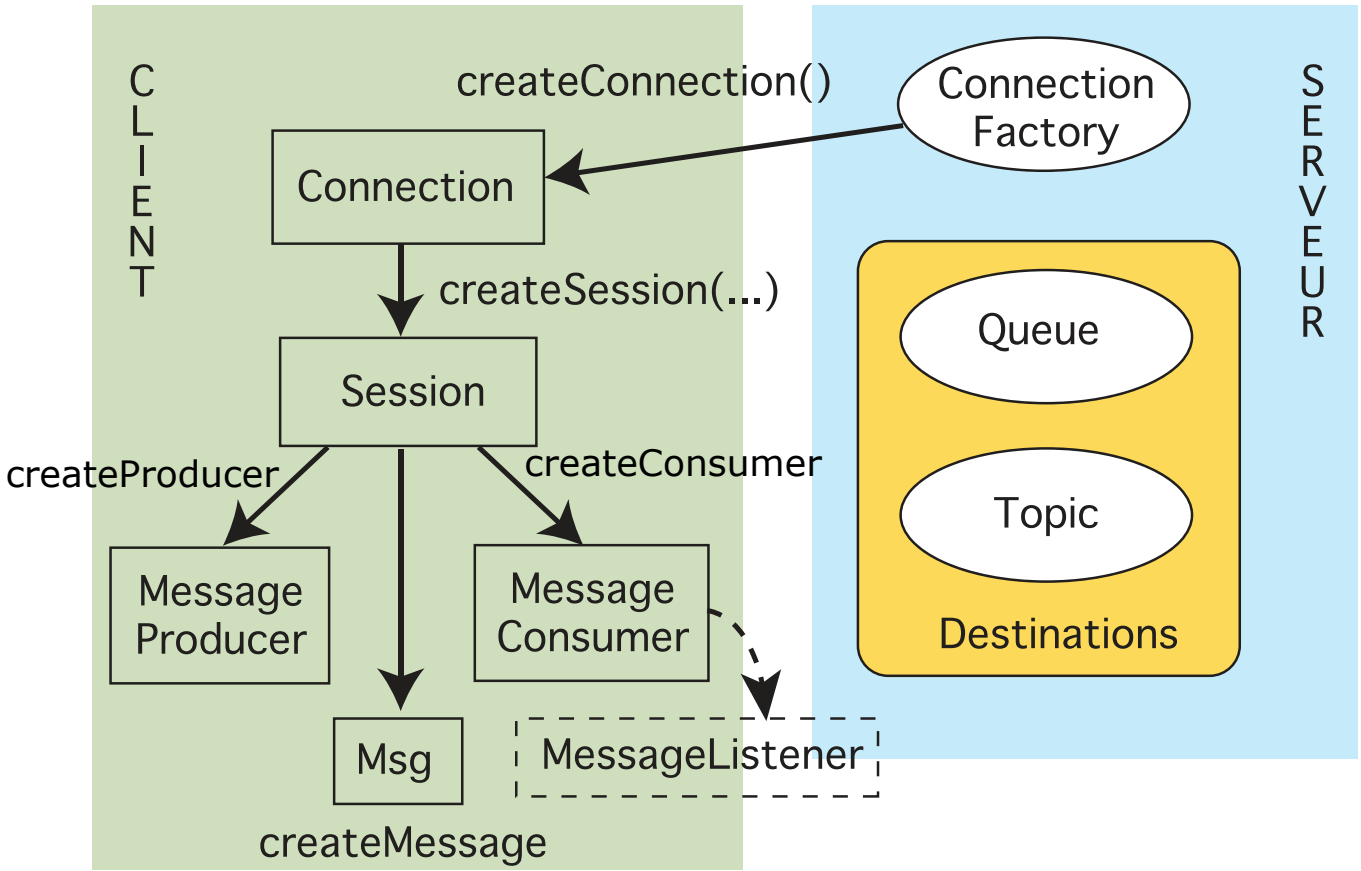
Les objets globaux (administrés) accessibles à distance

- Désignation par JNDI (Java Naming and Directory Interface)
- Les fabriques de connexions (connection factories)
- Les destinations réparties en 2 domaines de désignation : files (queues) et sujets (topics)
- Ces objets sont créés dans le(s) serveur(s) courtier(s) implantant JMS

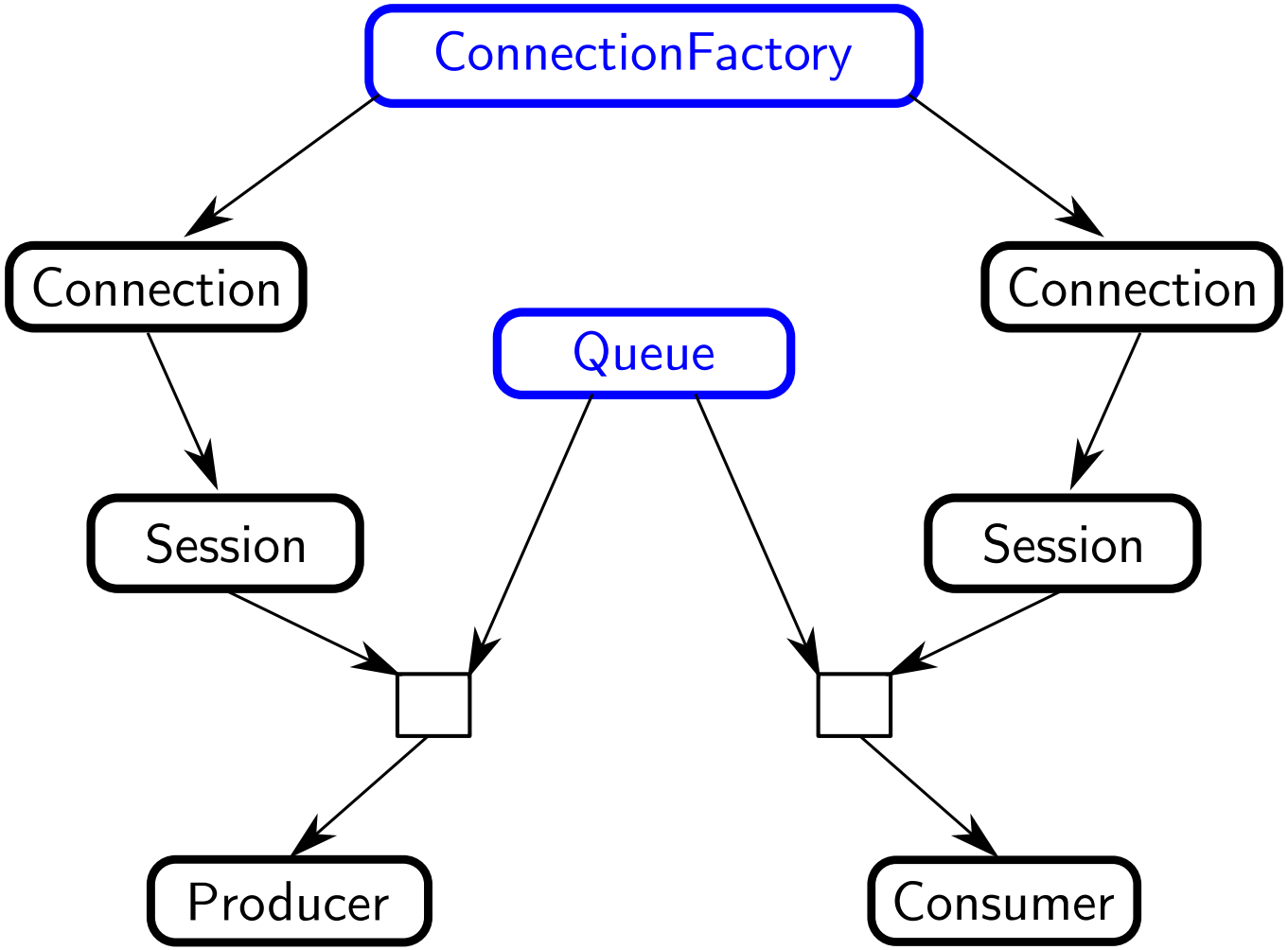
Les objets clients

- Les connexions permettent de se connecter à un serveur JMS
- Les sessions gèrent les échanges via une file ou un sujet
- Les producteurs/consommateurs de messages pour l'envoi/la réception de messages dans le cadre d'une session

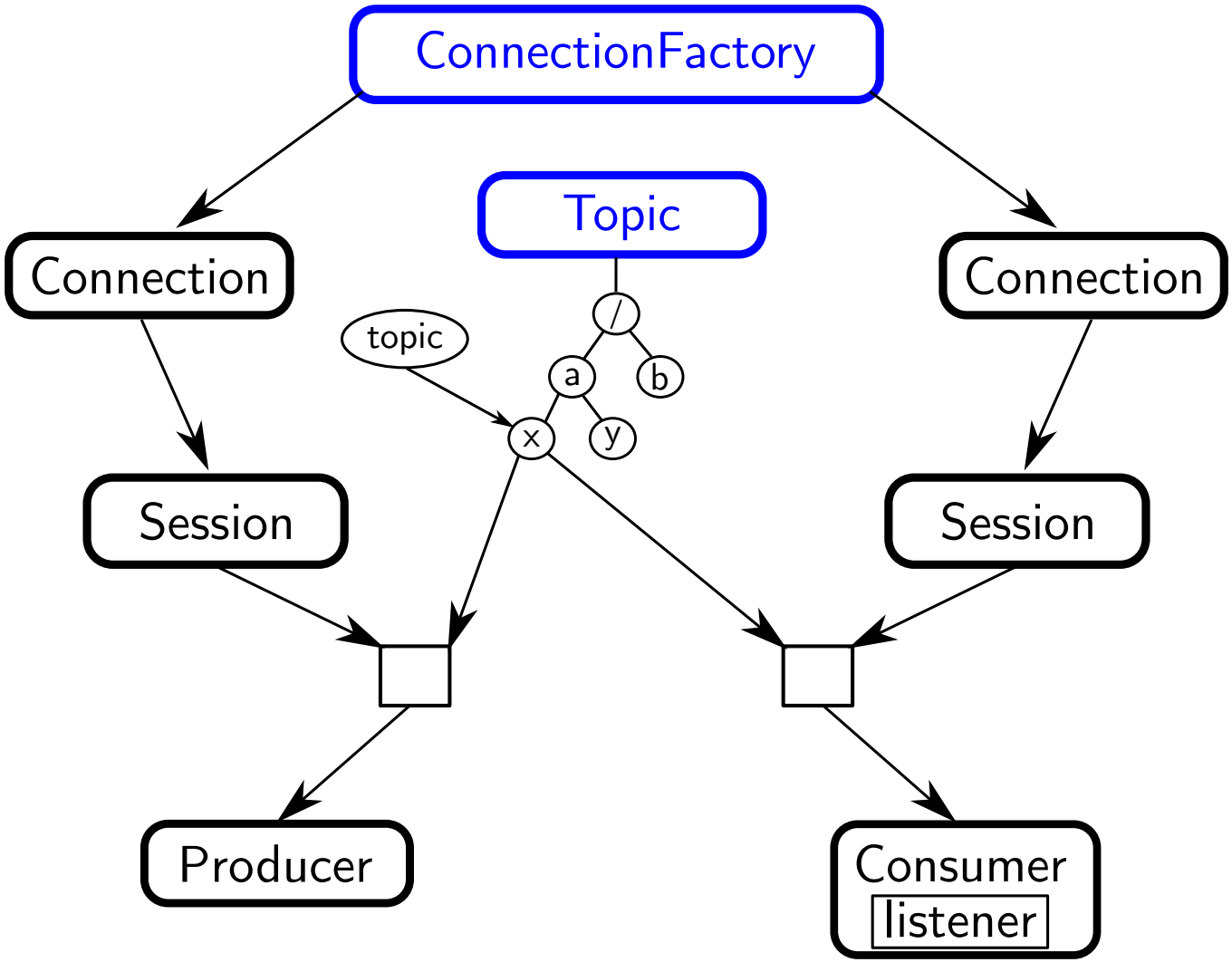
Les objets de communication



Les objets – Queue



Les objets – Topic



Un exemple de publication/abonnement

Le client éditeur : nom de la destination fourni dans args[0]

```
import javax.jms.* ;   import javax.naming.* ;
public class Editeur {
 public static void main(String[] args) {
 try {
  InitialContext jndiCtx = new InitialContext();
  ConnectionFactory scf = (C...) jndiCtx.lookup("MaConnFactory")
  Destination dest = (Destination) jndiCtx.lookup(args[0]);
  Connection conn = scf.createConnection();
  conn.start();
  Session session =
       conn.createSession(false,Session.AUTO_ACKNOWLEDGE);
  MessageProducer editeur = session.createProducer(dest);
  TextMessage m = session.createTextMessage();
  m.setText("publication exemple");
  editeur.send(m); ...
}
```

Un exemple de publication/abonnement

Un client abonné : nom de la source fourni dans args[0]

```
import javax.jms.* ;   import javax.naming.* ;
public class Abonne {
 public static void main(String[] args) {
  try {
  InitialContext jndiCtx = new InitialContext();
  ConnectionFactory scf = (C...) jndiCtx.lookup("MaConnFactory")
  Destination src = (Destination) jndiCtx.lookup(args[0]);
  Connection conn = scf.createConnection();
  Session session =
       conn.createSession(false,Session.AUTO_ACKNOWLEDGE);
  MessageConsumer abonné = session.createConsumer(src);
  abonné.setMessageListener(new MonMsgListener());
  conn.start();
  ...
```

Un exemple de publication/abonnement

Le traitant activé sur occurrence d'un message

```
import javax.jms.* ;
public class MonMsgListener implements MessageListener {
  public void onMessage(Message m) {
   try {
    if (m instanceof TextMessage) {
     TextMessage msg = (TextMessage) m ;
     System.out.println(msg.getText()) ;
     }
   } catch (JMSException je) {...}
  }
}
```

Un exemple de publication/abonnement

La création des objets globaux

```
import org.objectweb.joram.client.jms.tcp.TcpConnectionFactory;
public class CreateDestination {
  public static void main(String args[]) throws Exception {
    // Creating the JMS administered objects:
    javax.jms.ConnectionFactory connFactory
             = TcpConnectionFactory.create("localhost", 16010);

    Destination destination = Topic.create(0);
    // Destination destination = Queue.create(0);

    // Binding objects in JNDI
    javax.naming.Context jndiCtx = new InitialContext();
    jndiCtx.bind("MaConnFactory", connFactory);
    jndiCtx.bind("MonTopic", destination);
    }
}
```

Conclusion

- Mode de communication adaptée à la circulation des informations dans le système informatique des entreprises
- Intégration dans des intergiciels à base de composants
- Mécanisme de base dans les bus de services