**JMS api:**

**Connection:**Astrazione della connessione TCP/IP.  
Dopo il costoso handshake client e server collegano tra loro in modo simmetrico i loro socket (stream bufferizzati io) e tale collegamento rimane attivo fino a che non viene interrotto (close di uno dei due).  
 **Session: (thread io su una connection)**Su una connection e’ possibile creare N Session.   
Almeno una e’ sempre necessaria per creare un Producer o un Consumer.  
Per migliorare le performances dell io e’ possibile creare N Session su un’ unica Connection.  
(mentre una Session aspetta dati , un’altra Session puo’ spedire/ricevere)

**Producer:**

**Consumer:**

**PERFORMANCES:**

**Quante Connessioni/quanto tempo aperte?**Connessione aperta ed inattiva spreca risorse sul server. (Monitorare traffico su sockets e buffers)  
Creare una nuova connessione e’ costoso per un client perche’ richiede N cicli di request-response per lhandshake iniziale. Necessita di riutilizzo.  
Bilanciare le due esigenze con un pool di connessioni, da misurare sull effettivo traffico atteso.

**Transazioni lente?**La transazionalita di una Session permette di rispedire messaggi persi nella trasmissione.  
Non ha un overhead superiore all ack manuale.  
Quello che rallenta e’ invece la persistenza dei messaggi.   
Per cui accoppiando transazionalita e persistenza si ottiene massima reliability con un prezzo in performances.  
In tal caso e’ possibile studiare delle ottimizzazioni con una commit manuale sulla Session cercando il chunk size ottimale per scrivere sul disco del MOM in modalita batch. (da studiare e verificare caso per caso per MOM vendor)

**Ack e transazioni:**Entrambi lavorano sulla conferma di ricezione del messaggio (lato client o MOM).  
La transazione permette con la commit di autorizzare la cancellazione del (dei) messaggio da parte del MOM. Ed ha anche la possibilita di andare in Rollback.

L’Ack fa la stessa cosa, ma non esiste la possibilita di ’UnAck’ un messaggio o una serie di messaggi.  
  
In caso siano presenti entrambe ha priorita **chi?**

**Pub/Sus vs P2P:**Pub Sub e’ indicato quando lo stesso messaggio deve raggiungere N destinatari. (Che possono aggiungersi in qualsiasi momento). E’ piu difficile da gestire con load balancing.  
  
Queue indicato per gestione 1 Producer 1 Consumer (logico, che puo’ essere diviso in N per questioni di performances). Permette ottima scalabilita attraverso load balancing.

**Performances check list:**

* Persistence? (=slower)
* Message selectors? (=slower)
* Concurrency? (quale dimensione ottimale per traffico)
* Durable subscribers? (=slower)
* Request/reply, "synchronously" with temporary queues (= overhead = slower)
* Queue prefetching (=impacts performance in some aspects)
* Caching (faster on client vs resource consuming on server for inactive)