Desarrollo de Software para Sistemas Distribuidos

Curso 2015

Fundamentos de Cliente/Servidor en 3 capas

- Las llamadas aplicaciones Cliente/Servidor de misión crítica dejaron de ser un elemento teórico. Cada día, millones de transacciones de comercio electrónico se inician desde las PC's y se ejecutan en servidores distribuidos. Con Cliente/Servidor se mezcla el procesamiento local y remoto en una aplicación única. Este es de alguna manera el fundamento de Internet, intranets y extranets.
- Sin embargo el Cliente/Servidor como lo conocemos hasta ahora está lejos de continuar siendo exitoso. La elección de la arquitectura de las aplicaciones es lo que puede hacer o no exitoso un proyecto Cliente/Servidor.

Por qué Cliente/Servidor en 3 capas?

- Hacia los 2000, el Cliente/Servidor se transformó en la arquitectura de aplicaciones más elegida del mercado. El Cliente/Servidor se aplicó sobre las aplicaciones monolíticas de los mainframes como una manera de dividir la carga de procesamiento.

Para qué Cliente/Servidor en 3 capas?

- Luego, la computación Cliente/Servidor típica en dos capas se transformó en una arquitectura en tres capas y el impacto de dicho cambio asegura ser mayor que el que tuvo el Cliente/Servidor sobre las aplicaciones de mainframes.
- Si bien el concepto de tres capas fue inicialmente dirigido por la necesidad de escalar las aplicaciones en dos capas, el mercado actual de servicios y componentes lo están tomando como su paradigma.

Dónde están las 3 capas?

- Cada capa describe una partición lógica de las aplicaciones entre clientes y servidores. Las capas nos permiten describir la elección de la arquitectura básica:
  - **2-tier divide la carga de procesamiento en dos. La mayoría de la lógica de la** aplicación corre en el cliente, que envía requerimientos SQL a una base de datos.
  - **3-tier divide la carga de procesamiento en: clientes corriendo una GUI,** aplicaciones servidoras corriendo lógica de negocios y una base de datos y/o aplicaciones delegadas.

Beneficios y limitaciones de la arquitectura de 2 capas?

- En los sistemas Cliente/Servidor de dos capas la lógica están dividida bien en la interface de usuario sobre el cliente o bien sobre la base de datos o bien en ambas.
- Para acceder a los datos, los clientes deben conocer la manera en que están organizados y dónde residen. El concepto de procedimientos almacenados provisto por Sybase en 1986 fue una primera aproximación para independizar al cliente de la estructura y ubicación de los datos.

Beneficios y limitaciones de la arquitectura de 2 capas? (cont)

- La simplicidad fue el mayor factor que dirigió la popularidad de las 2 capas. Es adecuado para aplicaciones simples y pequeñas (nivel departamental) pero carecen de una propiedad: escalabilidad.
- La carencia de esta propiedad en la actualidad justifica la evolución de las 2 capas hacia una arquitectura capaz de manejar un mundo más complejo donde las aplicaciones se dividen y distribuyen a través de múltiples procesadores, plataformas y ubicaciones geográficas.

Comparación entre 2 y 3 capas

Características de la Aplicación	Departamental	Intergaláctica
Cantidad de clientes por app.	Menos de 100	Millones
Cantidad de servidores por app.	1 o 2 servidores homogeneos	Mas de 100.000 servidores realizando distintas tareas
Geografía	Campus	Global
Interacción Server to Server	NO	SI
Middleware	SQL y Stored Procedures	Componentes
Arquitectura	2 capas	3 capas (o N capas)

Características de 3 capas

- El cliente provee la GUI e interactúa con el servidor a través de servicios remotos y métodos de invocación.
- · La lógica de la aplicación vive en el nivel medio y se distribuye y administra en forma separada de la interface de usuario y de la base de datos.
- · Las aplicaciones en 3 capas minimizan los intercambios en la red creando niveles de servicios abstractos.

Características de 3 capas

- · Las aplicaciones en 3 capas sustituyen unas pocas invocaciones al servidor por varias consultas y actualizaciones SQL.
- · Proveen mayor seguridad al no exponer la estructura física de los datos.

Patrones de comparación

Característica	2 capas	3 capas
Administración de sistema	Compleja (más lógica en el cliente)	Menos compleja (se centralizan las aplicaciones)
Seguridad	Baja (a nivel de datos)	Alta (a nivel de servicios y métodos)
Encapsulamiento de datos	Baja (tablas expuestas)	Alta (el cliente invoca servicios y métodos)
Performance	Pobre (muchas sentencias SQL viajan por la red)	Buena (se intercambian sólo preguntas y respuestas)
Escalabilidad	Pobre	Excelente
Reuso de aplicaciones	Pobre (aplicaciones monolíticas en el cliente)	Excelente (se reusan servicios y objetos)
Soporte de Internet	Pobre (limitaciones por ancho de banda)	Excelente (los “thin” clientes son fáciles de descargar)
Soporte de BD heterogéneas	NO	SI
Elección de mecanismo de comunicación	NO (sólo sincrónico y orientado a la conexión)	SI (mensajes, colas, broacasting)

Componentes: de 3 a N capas

- El nivel medio en muchas aplicaciones en 3 capas no es implementado como un programa monolítico sino que es una colección de componentes utilizadas por varias transacciones.
- Cada componente automatiza una función de negocio relativamente pequeña. Los clientes suelen combinar varias componentes del nivel medio dentro de unatransacción única.

Tipos de Componentes Servidoras

Hay dos tipos de componentes en el nivel medio:

- **Servicios:** implementan una función de negocio. Son procesos sin estado. Reciben un pedido, lo ejecutan y terminan.
- **Objetos:** exponen un conjunto de procesos o métodos no sólo un proceso. ORBs provee una infraestructura para objetos distribuidos que se comunican a través de lenguajes, sistemas operativos y plataformas.

Objetos y sus estados

Dependiendo de la implementación estos objetos pueden tener o no estado

- · **Objetos sin estado (Stateless) no tienen un estado único. El objeto cuenta** bancaria no tiene estado, quien tiene estado es la cuenta bancaria de Juan Perez.
- · **Objetos con estado (Stateful) permite a los clientes requerir servicios de un** objeto específico usando un identificador del objeto. El nivel medio debe proveer la manera de encontrar dicho objeto y también encargarse de almacenar el estado (commit)

Comunicación entre componentes

Las alternativas de comunicación son:

- · **Conversacional: es sincrónico o orientado a la conexión tipo TCP/IP.**
- · **Pregunta-Respuesta: interacción sólo entre cliente y servidor tipo RPC**
- · **Colas: interacción desacoplada y asincrónica.**
- · **Publicar-Suscribir: permite que las componentes registren su interés en ciertos** mensajes junto con un administrador de eventos.
- · **Broadcast y datagramas: permite una comunicación en un solo sentido a uno o** más clientes.

Transacciones en entornos distribuidos

La computación C/S se sustenta en el concepto de compartir datos y todo la cooperación que existe se centra en ello, pero los programas que operan sobre los datos son igualmente importantes. Para conseguir esta armonía se necesita algo similar a un director de orquesta que orden una sinfonía con su batuta.

Propiedades ACID

ACID significa Atomicidad, Consistencia, Aislación (Isolation) y Durabilidad y constituyen las cuatro propiedades básicas que deben cumplir las transacciones:

- · **Atomicidad: una transacción es un unidad indivisible de trabajo. Todas las** acciones tienen éxito o fallan cumpliendo la proposición del "todo o nada". La atomicidad se define desde el punto de vista del consumidor de la transacción.
- · **Consistencia: una transacción que se ejecuta deja el sistema en un estado** correcto y si no aborta.

Propiedades ACID (cont)

- · **Aislación: el comportamiento de una transacción no es afectado por otras** transacciones concurrentes. La transacción serializa los accesos a recursos compartidos.
- · **Durabilidad: los efectos de la transacción son permanentes después del commit,** los cambios son persistentes.

Modelos de transacciones

- Transacciones planas
  - Son las mas frecuentes en los sistemas transaccionales.
  - Cuentan con primitivas para iniciar una transacción (*begin transaction*) que puede terminar exitosamente con *commit* o fallar con *rollback*.
  - Cumplen fácilmente las propiedades ACID.
  - Son generalmente cortas.
  - Puede ser distribuida (two-phase commit)

Limitaciones de transacciones planas

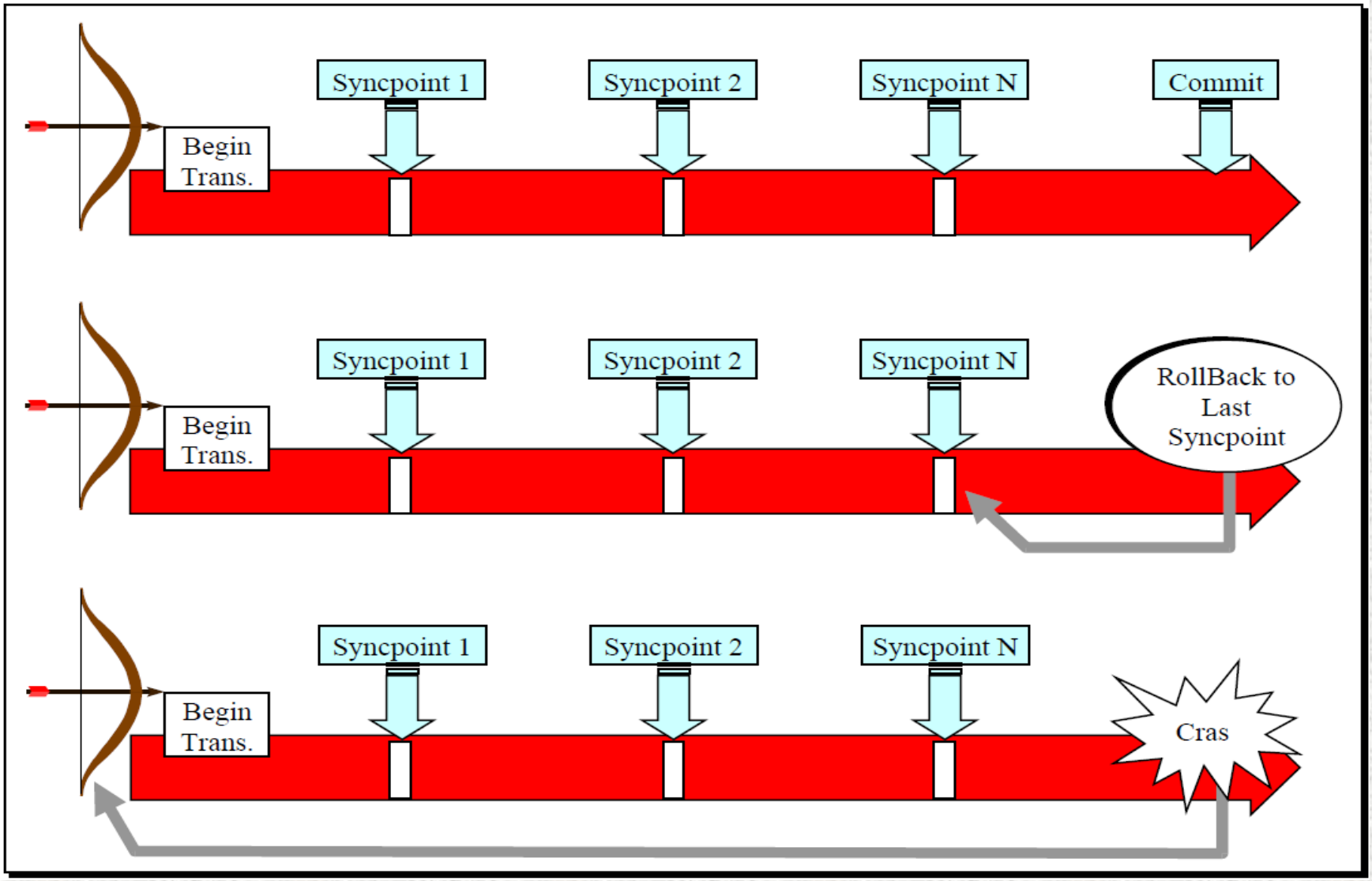
- Transacciones compuestas que necesitan rollback parciales (viajes, reservaciones de hotel y renta de autos pueden cancelarse parcialmente)
- · Transacciones con humanos en el medio (alguien deja la transacción abierta y se va a almorzar)
- · Transacciones que se extienden por un largo periodo de tiempo
- · Transacciones que involucran un gran volumen de datos (se actualizan 1 millón de registros juntos y se produce un rollback en el 999.999)
- · Transacciones que se trasmiten por Internet (constituye un problema político)

Transacciones encadenadas

Una transacción encadenada introduce un concepto de control lineal para secuenciar transacciones.

La forma mas fácil de implementarlas es con puntos de guarda dentro de una transacción plana que permite salvar cambios hasta el commit definitivo. Si se produce un rollback se hace hasta el ultimo punto de guarda pero si el sistema se cae (crash) se pierden todos los puntos de guarda, es decir no cumplen la propiedad de Durabilidad.

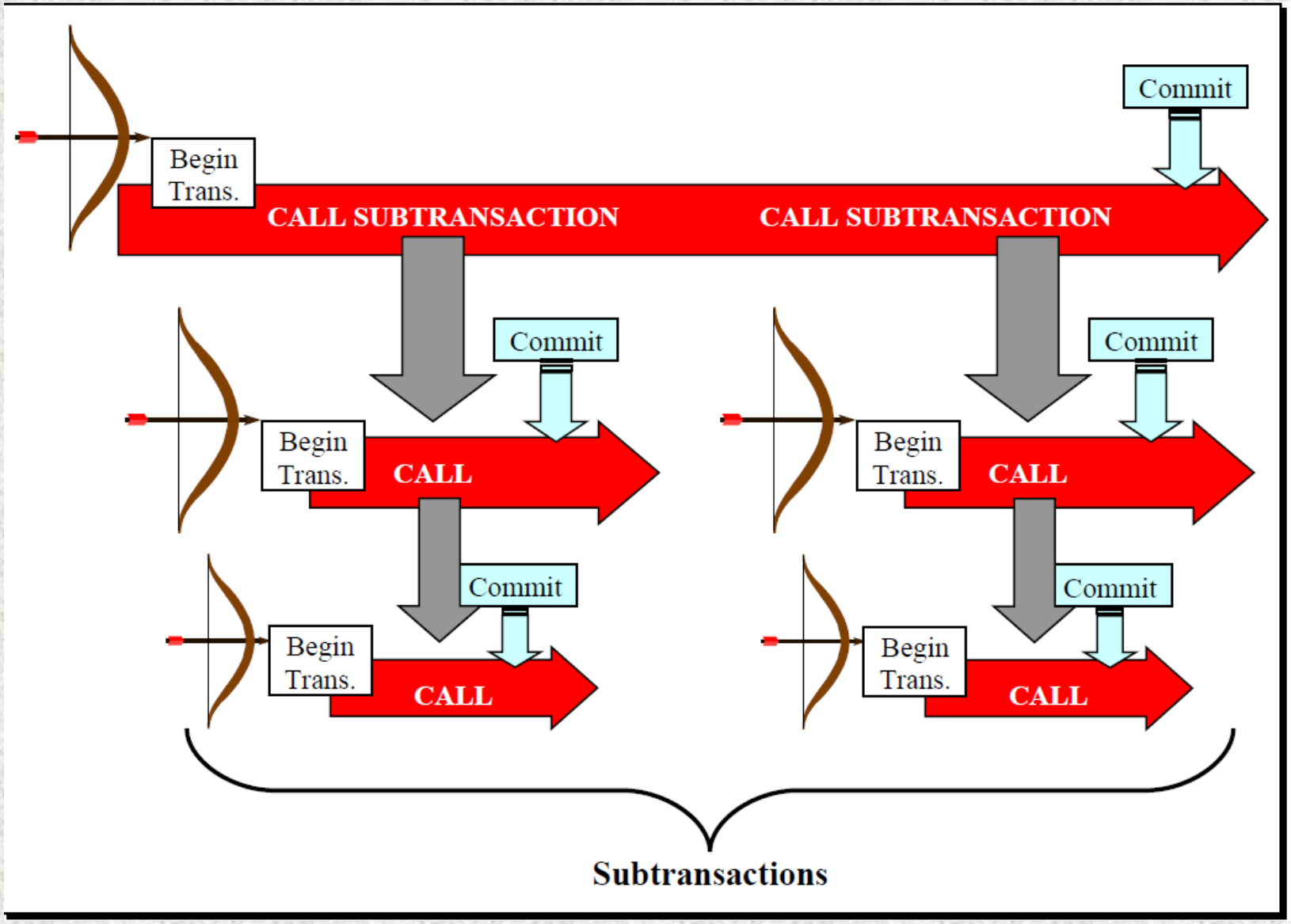
Transacciones encadenadas



Transacciones anidadas

- Definen transacciones dentro de transacciones.
- Conservan la misma semántica que los procedimientos de un lenguaje de programación, estableciendo una relación jerárquica de subtransacciones.
- La transacción principal comienza las subtransacciones y cada una de ellas puede hacer commit y rollback de sus piezas de trabajo locales.
- Cuando una subtransacción termina, los resultados están sólo disponibles para el padre.
- El commit de una subtransacción se hace permanente después del commit local y el de todos sus ancestros.

Transacciones anidadas



Monitores de transacciones

- Los MT son un concepto que apareció ya en los viejos mainframes brindando un ambiente de ejecución robusto y permitieron soportar aplicaciones OLTP (On Line Transaction Processing) a gran escala.
- Con la migración de las OLTP a plataformas C/S surgió la necesidad de introducir nuevamente la idea de MT.
- Administran las transacciones desde su punto de origen a través de uno o mas servidores. Cuando la transacción termina se garantiza un estado consistente del sistema y además de ejecutar transacciones los MT las rutean a través de la red, balancean la carga de su ejecución y las recuperan después de fallas.

Monitores de transacciones

- La funcionalidad específica es:
  - Administración del procesamiento(monitoreo y balanceo de carga)
  - Administración de transacciones (garantiza ACID a todos los programas bajo su protección)

TP Lite – TP Heavy

- TP-Lite es simplemente la integración de funciones de un MT en los motores de base de datos (a través de Stored Procedure y Triggers).

- TP-Heavy son los MT que realizan la administración, balanceo de carga, sincronización y monitoreo de las transacciones. (Ejemplos: CICS, Tuxedo y Jaguar CTS).

TP Lite vs. TP Heavy

- **Alcance del Commit:** en TP-Lite los SP son definidos en un dialecto de SQL y almacenados como un objeto dentro de la BD, constituyendo una unidad transaccional pero no pudiendo participar con otras unidades transaccionales respecto de una transacción global.

  Si A llama a B, B es otro unidad transaccional y luego A muere, los commit hechos por B no pueden ser deshechos luego de la muerte de A, violando el "todo o nada". Por lo tanto la única solución es que A y B estén en la misma transacción generando una unidad demasiado grande.

  Los procesos TP-Heavy son escritos usando lenguajes procedurales estándar (no SQL) permitiendo lograr el "todo o nada" y el concepto de transacción global.

TP Lite vs. TP Heavy

- **Administración de recursos heterogéneos**: un SP en TP-Lite solo puede "cometer" transacciones que afecten a recursos propios del vendedor de la BD.

  Los procedimientos TP-Heavy pueden manejar actualizaciones ACID sobre múltiples recursos de distinto tipo dentro del alcance de una transacción.

TP Lite vs. TP Heavy

- **Administración de procesos:** un SP en TP-Lite es invocado, se ejecuta bajo la protección ACID y puede quedar en cache para reusos futuros.

  Un proceso TP-Heavy es manejado como una clase de servidor pudiendo manejar prioridades, poseen firewalls para que otros procesos no los interfieran y si la clase muere, la transacción puede ser reasignada a otra clase.

TP Lite vs. TP Heavy

- **Invocaciones C/S**: una invocación a un SP es totalmente no estándar, se realiza a través del mecanismo de RPC provisto por el vendedor.

  En TP-Heavy el mecanismo de interacción es cualquiera (RPC o mensajes) y entran a un server con two-phase commit.

TP Lite Vs. TP Heavy

- **Performance:** ya sabemos que los SP son mucho mejores en cuanto a la performance medida en trafico de red respecto del SQL estático o embebido pero son incapaces de mejorar la performance en otros aspectos como balanceo de carga, además de ser (los SP) muchas veces interpretados línea a línea dentro del motor

  TP-Heavy que tiene código precompilado. Además con TP-Heavy se pueden multiplexar los requerimientos del cliente.