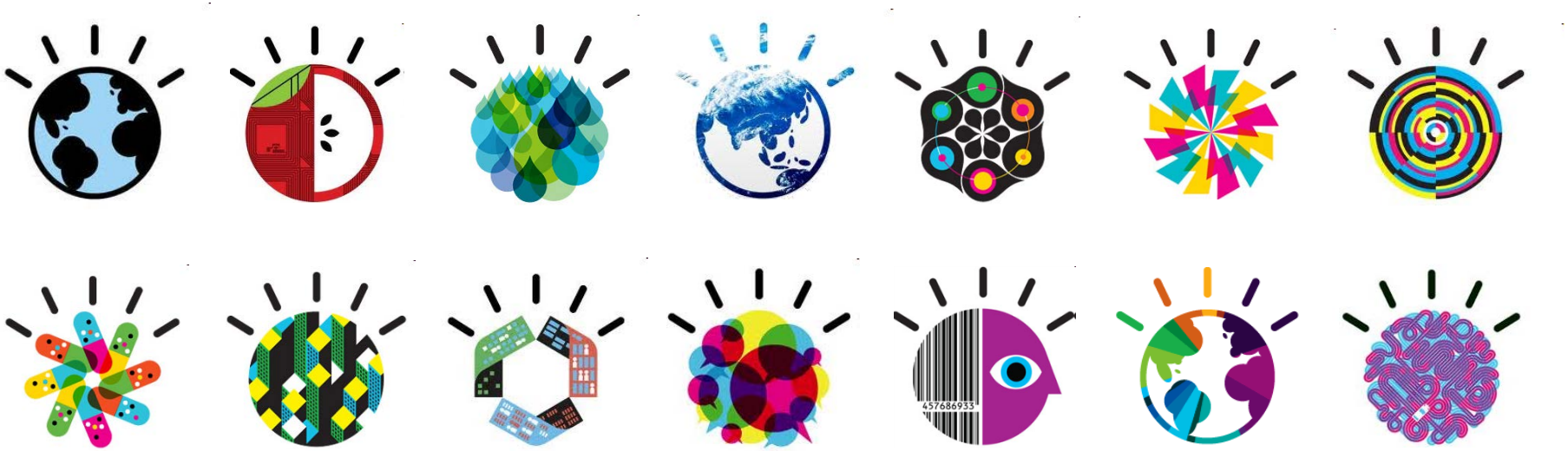
infocura
We care for your information

Java Application Performance
On DB2 for z/OS

<Ludovic Janssens, Consultant>



Premier Business Partner IBM

Presentation Overview

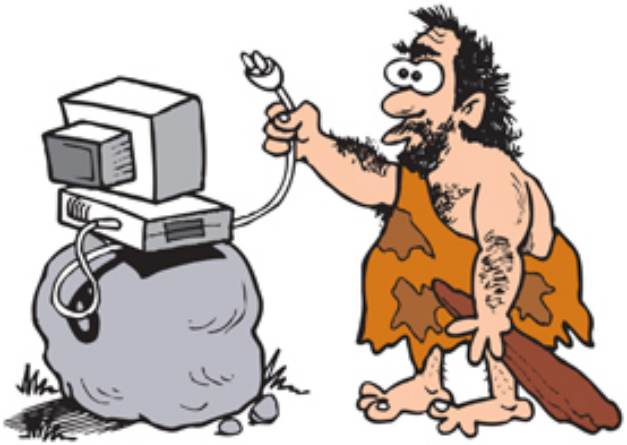
What is this presentation about?

- Why this presentation?
- DB2 Connectivity
  - Type 4 connectivity architecture
  - Connection Pools
  - Relevant Configurations
- Getting a grip on your Java SQL
  - Object Relational Mapping Strategies
    - Java Standards
    - ORM software features
  - Online Java vs Batch Java
  - Not all Java workload is Java coding
    - Groovy and Gorm
    - Scala
- End-to-End Monitoring and Tuning
  - Importance
  - Identification throughout the enterprise
  - Explain facilities

Why this presentation?

Why bother listening?

- Many Java developers think about the mainframe as the big black box from the stone age
- Many mainframe specialists think of Java as something that is that much automated, that it cannot work properly on the mainframe and DB2 for z/OS dba's see (generated) dynamic SQL as a heresy



Maybe it’s time to make peace...

... And to boldly go where no man has gone before

DB2 Connectivity

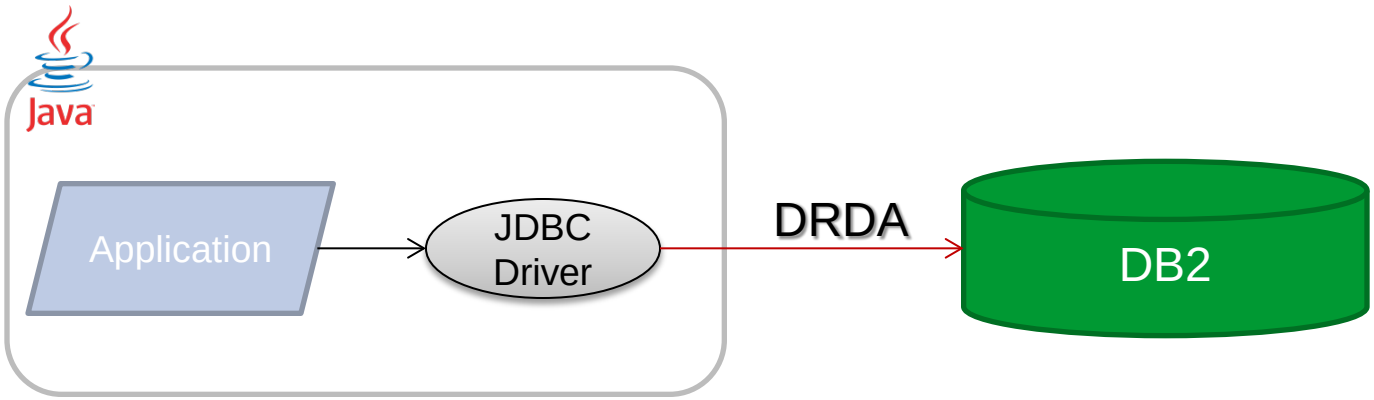
Any fact becomes important when it's connected to another

(Umberto Eco, Foucault's Pendulum)

Type 4 connectivity architecture

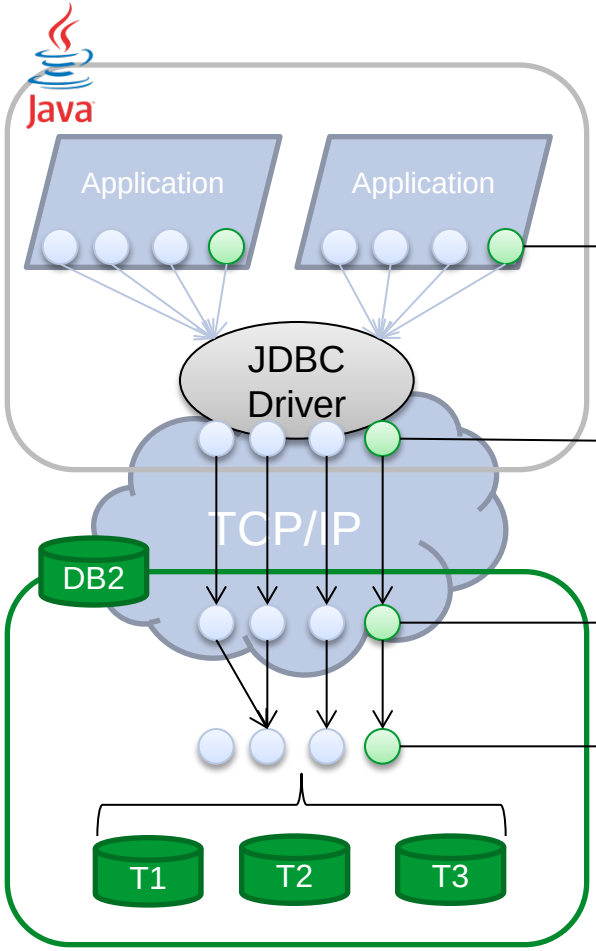
The Joy of DRDA

- JDBC is able to do direct database calls thanks to DRDA. The driver is pure java code
  - With the DB2 Connect Universal Edition, the license is on the mainframe and Java can connect directly to DB2 for z/OS
  - With a gateway, you use two type 4 connections: 1 to the gateway and 1 to the database
- DB2 11 provides an improvement: with DRDACBF the requester opens a secondary connection to the DB2 server for each read-only cursor



Connection Pools

Connection Pooling Architecture



Application Connection Pool: DBCP, C3P0, BoneCP, WAS
Active + Idle Connections (Logical Connections)

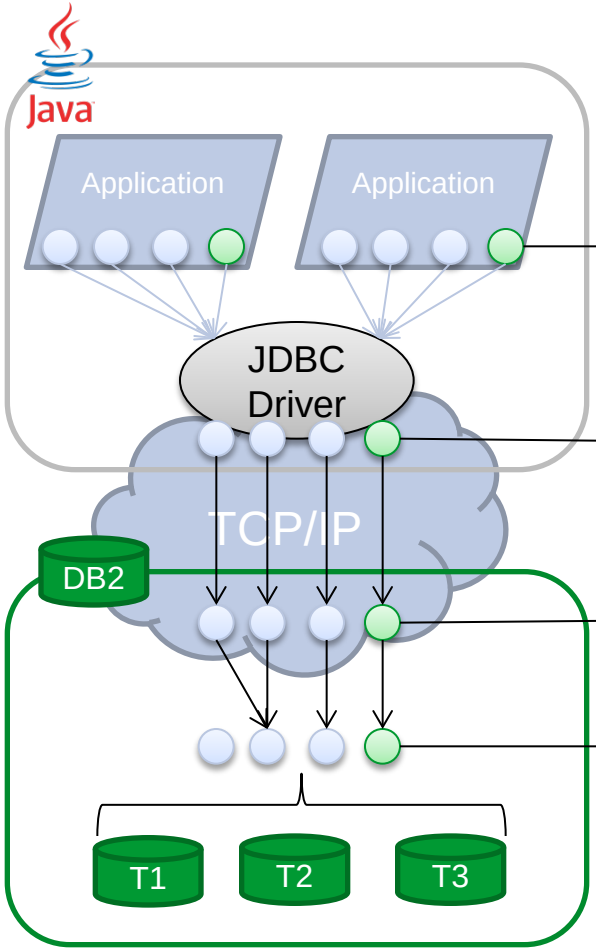
JDBC Global Transport Pool:
Active + Idle TCP/IP Threads (Physical Connections)

DB2 Inbound TCP/IP Connection Pool:
CONDBAT/CMSTAT/MAXTYPE1 (Active/Inactive)
(Logical Database Threads)

DB2 Inbound Database Connection Pool:
MAXDBAT (Active/Inactive) (Physical Database Threads)

Connection Pools

Connection Pooling Architecture



Set enableConnectionConcentrator to true!
(false by default)

Application Connection Pool: More logical connections possible per TCP/IP thread (Caveat: DBCP is single threaded!!)

JDBC Driver Connection Pool: The TCP/IP threads are heterogeneous

DB2 Inbound TCP/IP Connection Pool: Logical Database Threads are heterogeneous and can connect to more physical threads

DB2 Inbound Database Connection Pool: Physical Database Threads are heterogeneous and can accept more logical connections

Relevant Configurations

zParms and driver properties to be aligned

- **Pool Sizes**:
  - zParms:
    - CONDBAT, MAXDBAT, MAXTYPE1, MAXCONQN
  - Driver Configuration:
    - Application: depends on the connection pool in use
    - Global Transport Pool:
      - db2.jcc.maxTransportObjects, db2.jcc.minTransportObjects
- **TimeOut configuration**: TCP/IP Keep Alive, Idle Thread Time Out, Pool Thread Time Out, Resource Time Out ...
  - zParms:
    - TCPKALV, IDTHTOIN, POOLINAC, IRLMWT, MAXCONQW
  - Driver Configuration:
    - Application: depends on the connection pool in use
    - Global Transport Pool:
      - db2.jcc.maxTransportObjectIdleTime, db2.jcc.maxTransportObjectWaitTime

Getting a grip on distributed SQL

To understand your fear is the beginning of really seeing (Bruce lee)

Object-Relational Mapping Strategies

Java is standardized, but what are the Java standards?

- SQL Connectivity
  - JDBC (Java Database Connectivity)
    - Standard for connecting dynamic SQL to the database
    - On its own a rather complex and verbose standard API
  - SQLJ (SQL for Java)
    - Standard for connecting static SQL to the database
    - Resembles classic (COBOL, PL/I, ...) embedded SQL implementations
    - Its database centric approach does not easily fit in the object-oriented paradigm
- In order to facilitate JDBC usage, we can map hierarchical Java Object Model with the Relational Data Model in DB2 (and resolve the '*object-relational impedance mismatch*')
  - This requires **Object Relational Mapping Software (ORM)**: Hibernate, iBatis (myBatis), OpenJPA ...
  - Current standards involve
    - JDO (Java Data Objects): standard for any data store access
    - JPA (Java Persistence API): standard for ORM implementations, subset of JDO

Object-Relational Mapping Strategies

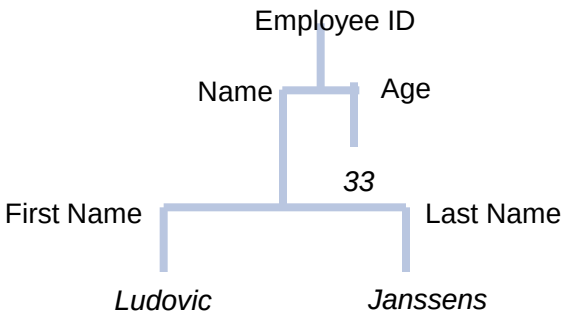
The ORM Software Types

- Type 1: Domain based ‘persistence layers’ (eg. Hibernate, OpenJPA, EclipseLink, …):
  - The object model is the basis for the SQL generation, using a variety of mapping techniques (see next slide)
  - A structured domain (query) language us used (eg. HQL, JPQL, JDOQL)
  - An underlying meta-model reflects the database
- Type 2: SQL based persistence layer (eg. iBatis/myBatis):
  - Java methods are linked to SQL clauses and statements
  - The SQL is built following the application logic
- Both strategies are valid, but in the scope of the SOA principles (loose coupling and service autonomy), the domain approach, that does not assume a relational model underneath, is the preferred one.

Object-Relational Mapping Strategies

Domain based mapping strategies

- Lazy loading
  - From the hierarchical object model parent nodes are fetched without their children until the data of the children is requested by the application
  - Less SQL when you only need parent data, more SQL when you need the child detail
- Eager loading
  - From the hierarchical object model parent nodes are fetch with their children in order to load all details up to a certain degree
  - Less SQL when needing alls detail throughout your application



Lazy:

```
[Need EmployeeID]
Select EmployeeID
From Emptab
Where Name = 'Ludovic Janssens';
[Need First and Last Name]
Select FirstName, LastName
From NameTab
Where FirstName = 'Ludovic';
[Need Age]
Select Age
From PersInfoTab
Where Name = 'Ludovic Janssens';
```

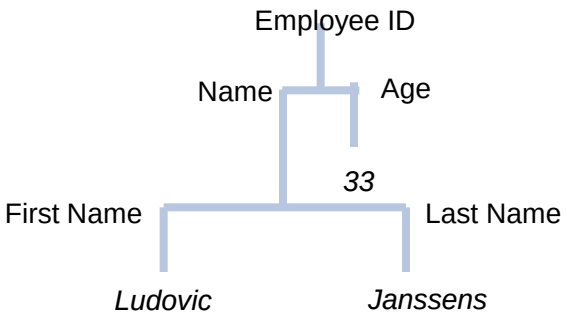
Eager:

```
[Need EmployeeID]
Select EmployeeID
From Emptab
Where Name = 'Ludovic Janssens';
Select FirstName, LastName
From NameTab
Where FirstName = 'Ludovic';
Select Age
From PersInfoTab
Where Name = 'Ludovic Janssens';
[Need First and Last Name] → get from cache
[Need Age] → get from cache
```

Object-Relational Mapping Strategies

Domain based mapping strategies

- Join Fetching
  - Join the tables that are fetched
  - Reduces the amount of SQL statements issued, but can generate too big joins
- Batch Fetching
  - If several queries are fetching data that is related to each other, batch fetch will first fetch the base object and then issue simple join queries to fetch the details (eg. Fetch employees, fetch those who work part time, fetch their telephone number)
  - Will generate more SQL, but more efficient SQL



Batch:

```
Select EmployeeID
From Emptab
Where Name = 'Ludovic Janssens';
Select FirstName, LastName
From NameTab
Where FirstName = 'Ludovic';
Select Age
From PersInfoTab
Where Name = 'Ludovic Janssens';
```

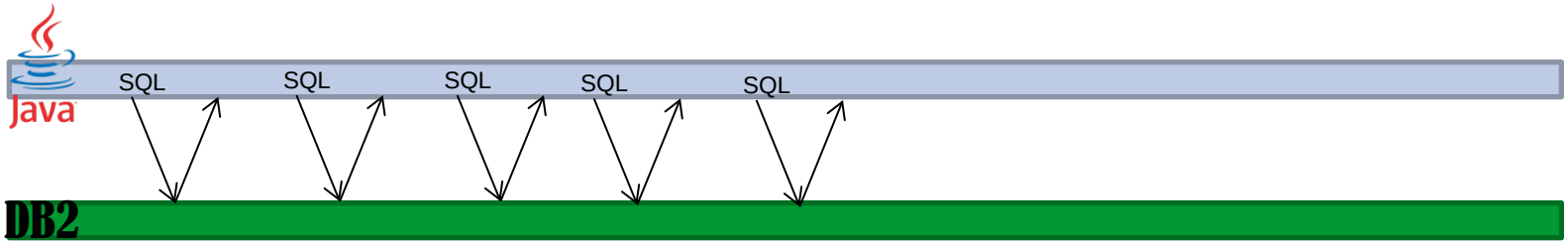
Join:

```
Select EmployeeID, FirstName, LastName, Age
From EmpTab A
    inner join NameTab B on A.Name = B.FirstName
    inner join PersInfoTab C on A.Name = C.Name
Where B.FirstName = 'Ludovic'
  and B.LastName = 'Janssens';
```

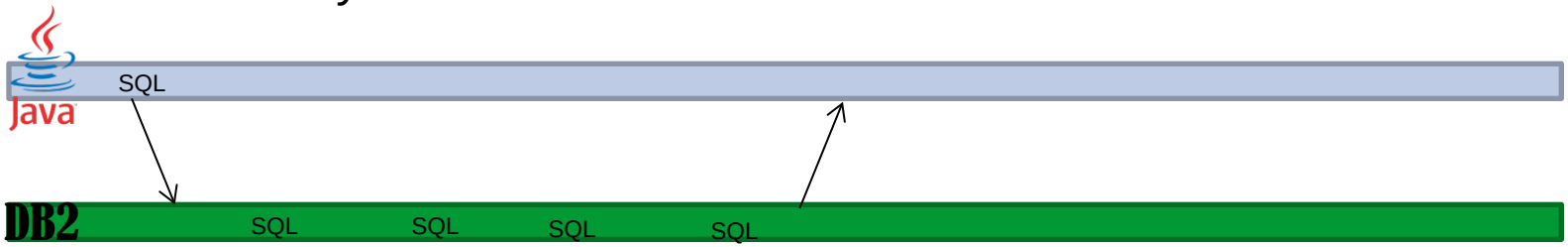
Java Online vs. Java Batch

Save network time

- Online Java
  - Connects on the ‘getConnection’ call and closes the connection when returning the query result



- Bath Java
  - Connects on the first ‘getConnection’ call and closes when all queries have been completed
  - Restriction: only similar DML can be issued in one batch



Not all Java Load is Java coding

On Groovy and Scala

- Groovy
  - According to the makers: ‘an agile and dynamic language for the Java Virtual Machine’
  - Makes use of the Java Virtual Machine (JVM), but includes features from scripting languages
  - GORM is part of the GRAILS framework and covers Hibernate 3
    - Typical feature is *SQL flushing*: build queries and release them when they are really required (is actually a hibernate feature, but very often used in this context)
      → Interesting for Java batch and for KeepDynamic
- Scala
  - Scala is a ‘scalable’ language, it is concise and makes an optimized usage of the JVM
  - Scala is Object-Oriented and Functional, its relational mapping is based on functions rather than objects
    - SQL can be the basis for Scala Queries (no DSL required)
    - Native Scala can match with SQL structures and make any database API obsolete
  - Ideal for Web Applications with massive amounts of queries (KeepDynamic)

End-to-End Monitoring and Tuning

Bond: “In my business, you prepare for the unexpected.”

Franz Sanchez: “And what business is that?”

Bond: “I help people with problems.”

Franz Sanchez: “Problems solver.”

Bond: “More of a problem eliminator.”

(From James Bond - *License to Kill*)

The Importance of End-to-End Monitoring

Where do we have to look?

- Without End-to-End Monitoring
  - Local Database Monitoring
    - Gives you a view from the moment the TCP/IP thread arrives on the mainframe until it leaves.
    - DB2 time, z/OS times
  - Local Application Monitoring
    - Gives you a view of the application thread until it leaves the server and a new view when it comes back
- With End-to-End Monitoring
  - Thread view from the issueing of the getConnection until the retrieval of the data
  - Clear view on where time is spent and whether the network causes latency
    - Base for decisions on Java batch implementation or choosing for Stored Procedures, ...
  - JDBC profiling gives a view on the SQL that is executed

Identification throughout the enterprise

Fill out the properties!

- clientUser, clientApplicationInformation, clientAccountingInformation, clientWorkstationName, clientProgramName, clientProgramId, clientCorrelationToken (DB2 11)
  - JDBC properties or WLM_SET_CLIENT_INFO
  - DB2 registers
- Values can be used
  - Monitoring (profiles, trace records, ...)
  - Security (trusted contexts secondary authId ...)
  - WLM (profiles)
  - RLF (profiles)
  - ...

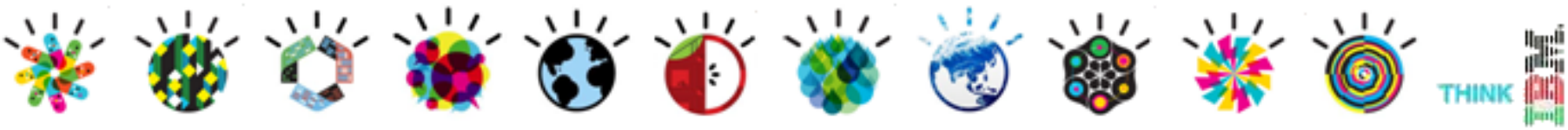
Explain Facilities

currentExplainMode

- currentExplainMode = NO → Default
  - No explains are done
  - Application is ran normally
- currentExplainMode = YES
  - Explains are run next to the application
  - Explains are done in the explain tables set set by the currentSchema and currentSQLID
- currentExplainMode = ONLY
  - Explains are run, but the application does not execute its statements on the database
  - Ideal for production tuning

Questions?

Thank You !



Short Word on Infocura

Who are we?

- INFOCURA
  - Is specialized in
    - the
      distribution, consulting, installation, migration, tuning, auditing and education
      of
  - IBM Information Management Products
    - both on
  - Distributed and Mainframe platforms
- INFO – CURA means: I care for your information

- Please visit http://www.infocura.com/