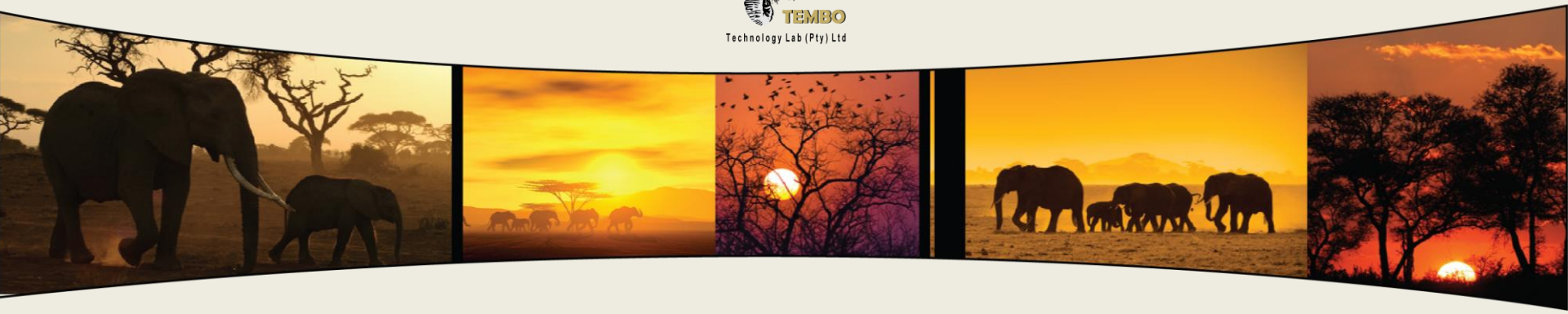
TEMBO
Technology Lab (Pty) Ltd



AO Advanced Modernization Workshop

Tommy Atkins
Chief Development Officer (CDO)

AO
Adsero Optima™

TEMBO Technology Lab (Pty) Ltd

1969: Trainee Operator

LEO III

1972: Cobol Programmer

ICL 450

1974: RPG Programmer

IBM System 3/10

1975: RPG II Programmer

IBM System/32

System/34

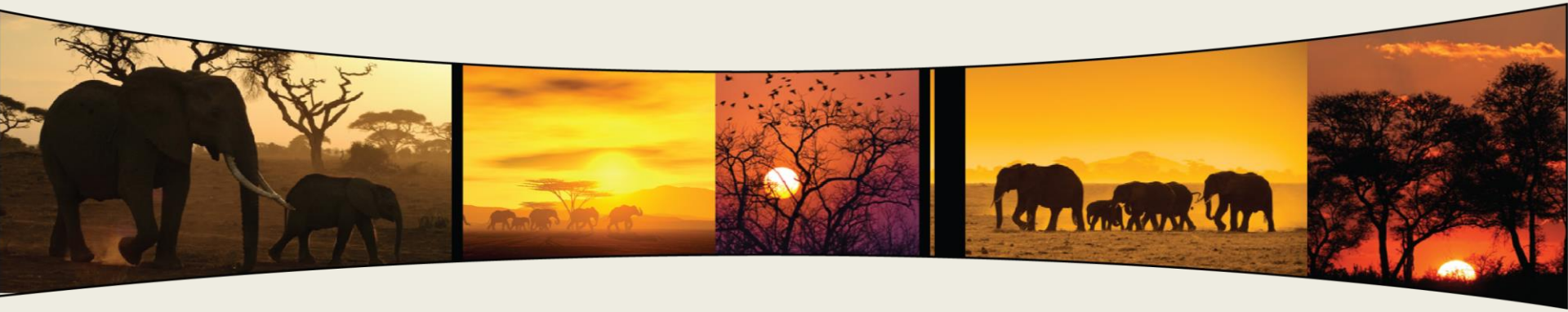
System/36

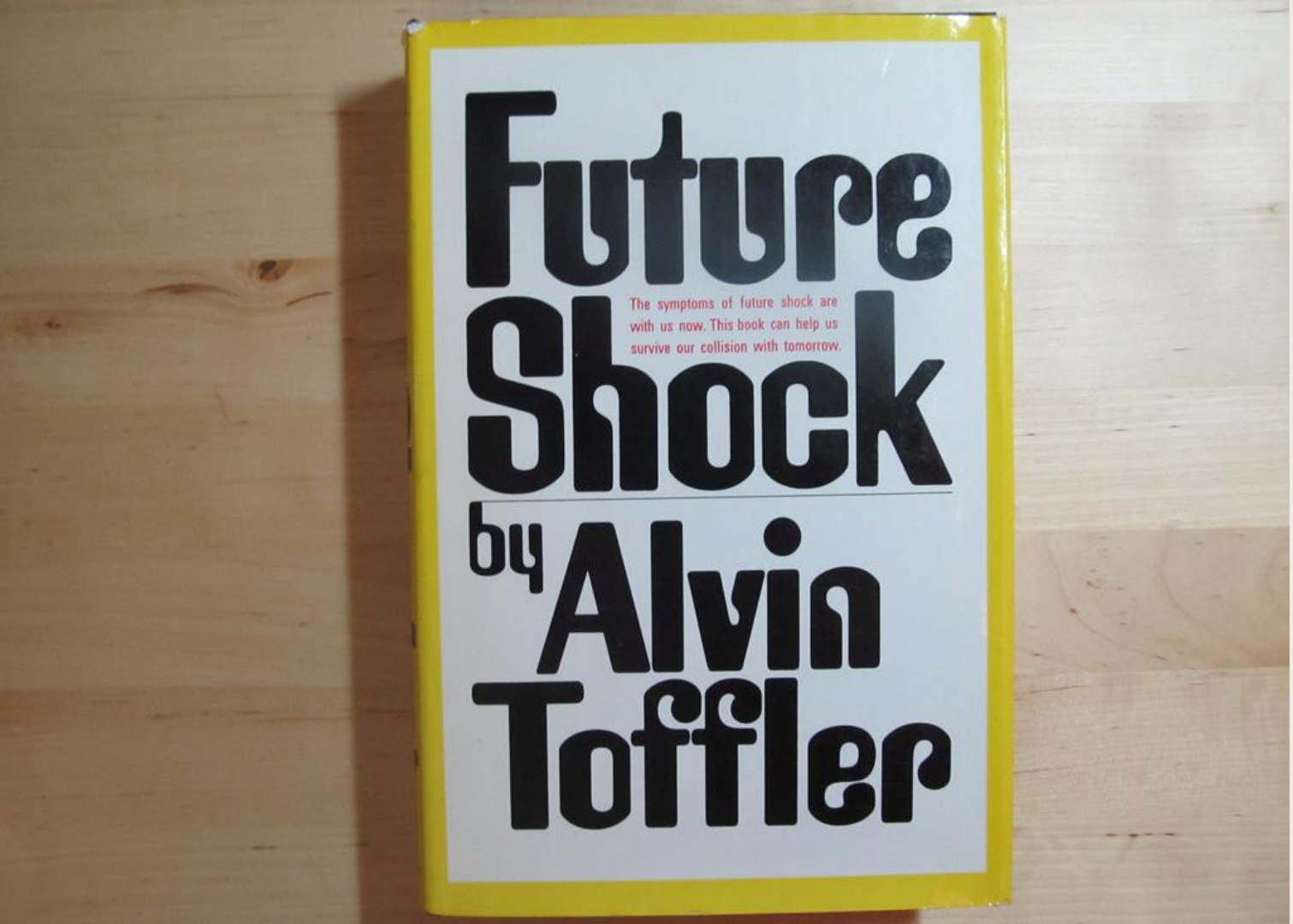
System/38

Terminal 5250

AS/400

Database & Application - Design & Development





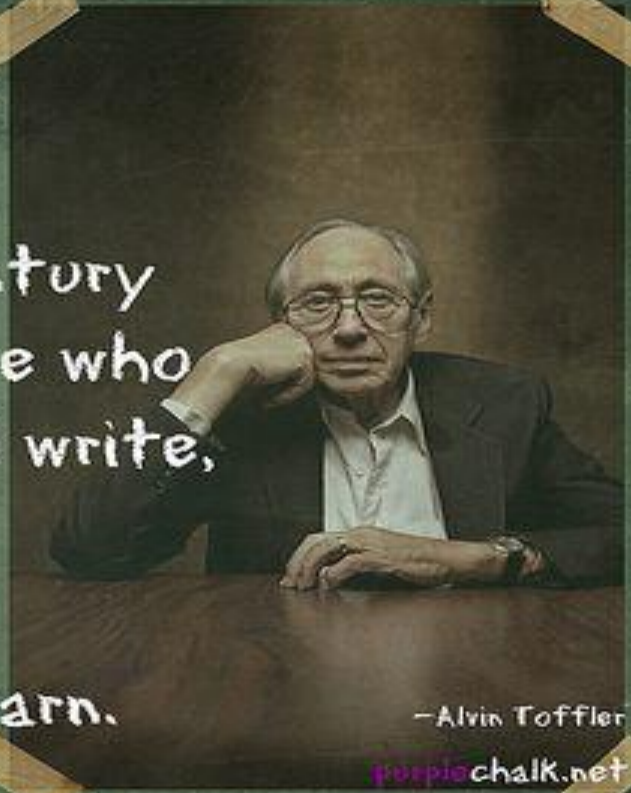
Written 1970

Sold 6 million copies

Future shock is the shattering stress and disorientation that we induce in individuals by subjecting them to too much change in too short a time.

(Alvin Toffler)

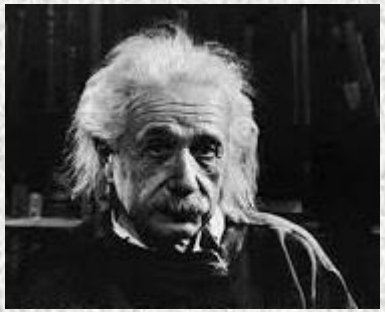
The illiterate
of the 21st century
will not be those who
cannot read and write,
but those who
cannot learn,
unlearn, and relearn.



–Alvin Toffler

purpleChalk.net

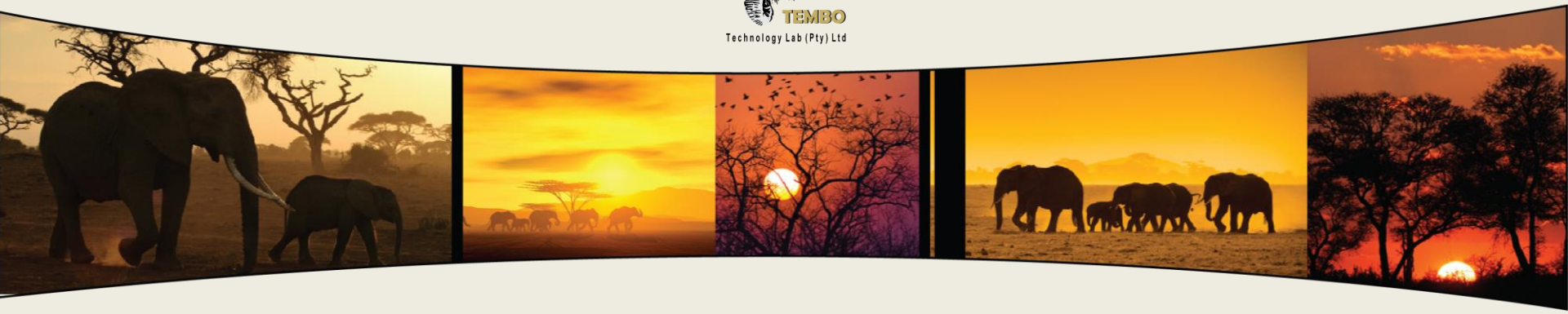
Ph.: ostrevv/flickr, cc by-sa 2.0



**“Computers are incredibly fast, accurate and stupid;
Humans are incredibly slow, inaccurate and brilliant;
Together they are powerful beyond imagination.”**

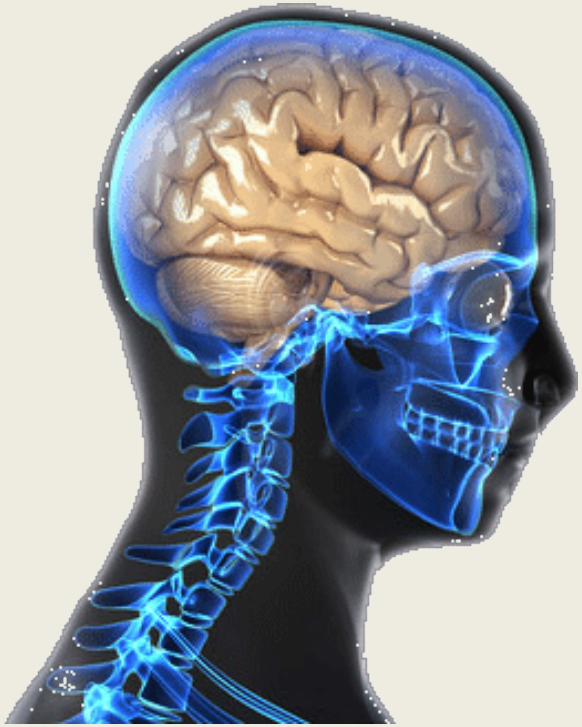
Albert Einstein

TEMBO
Technology Lab (Pty) Ltd

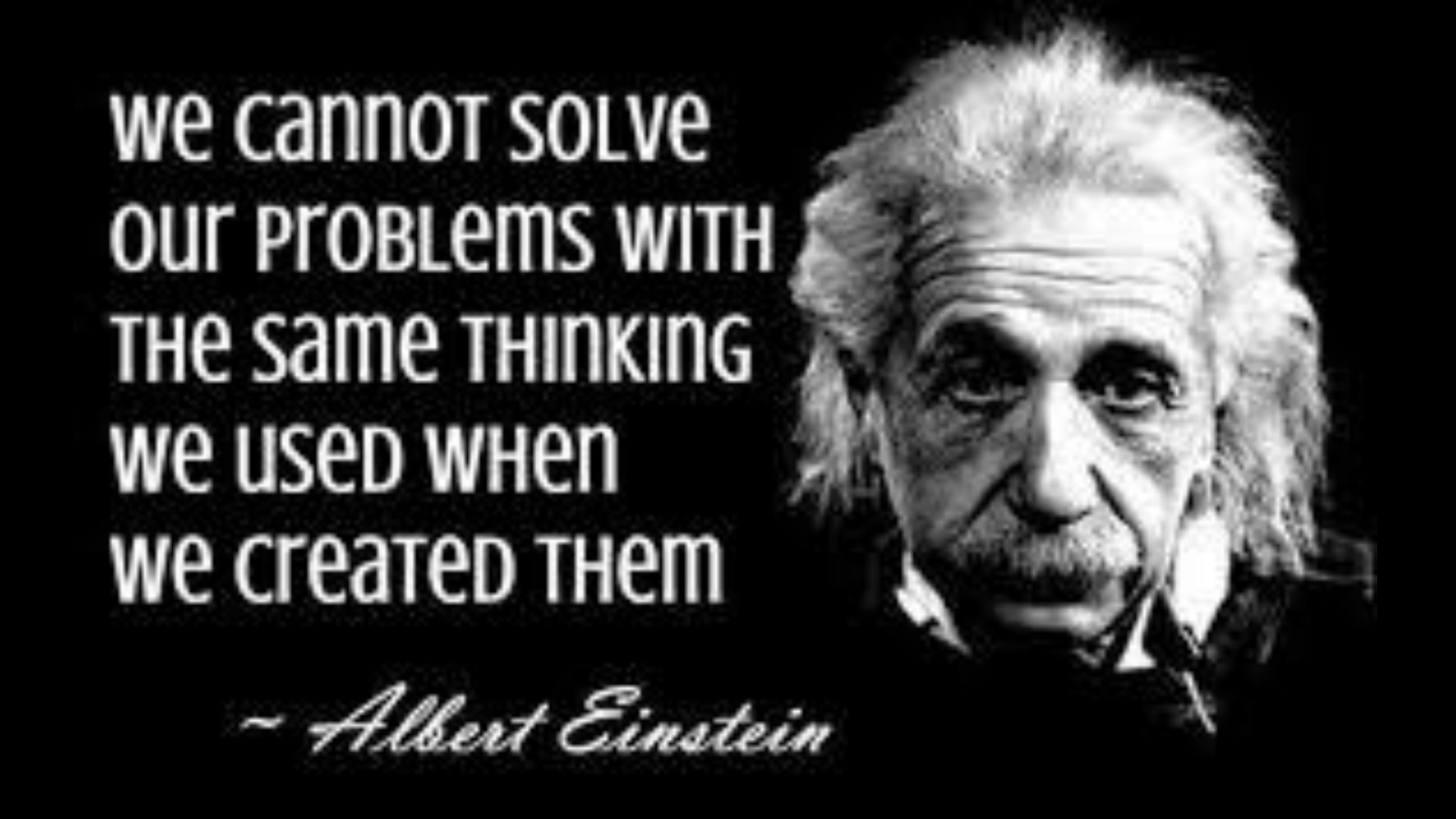


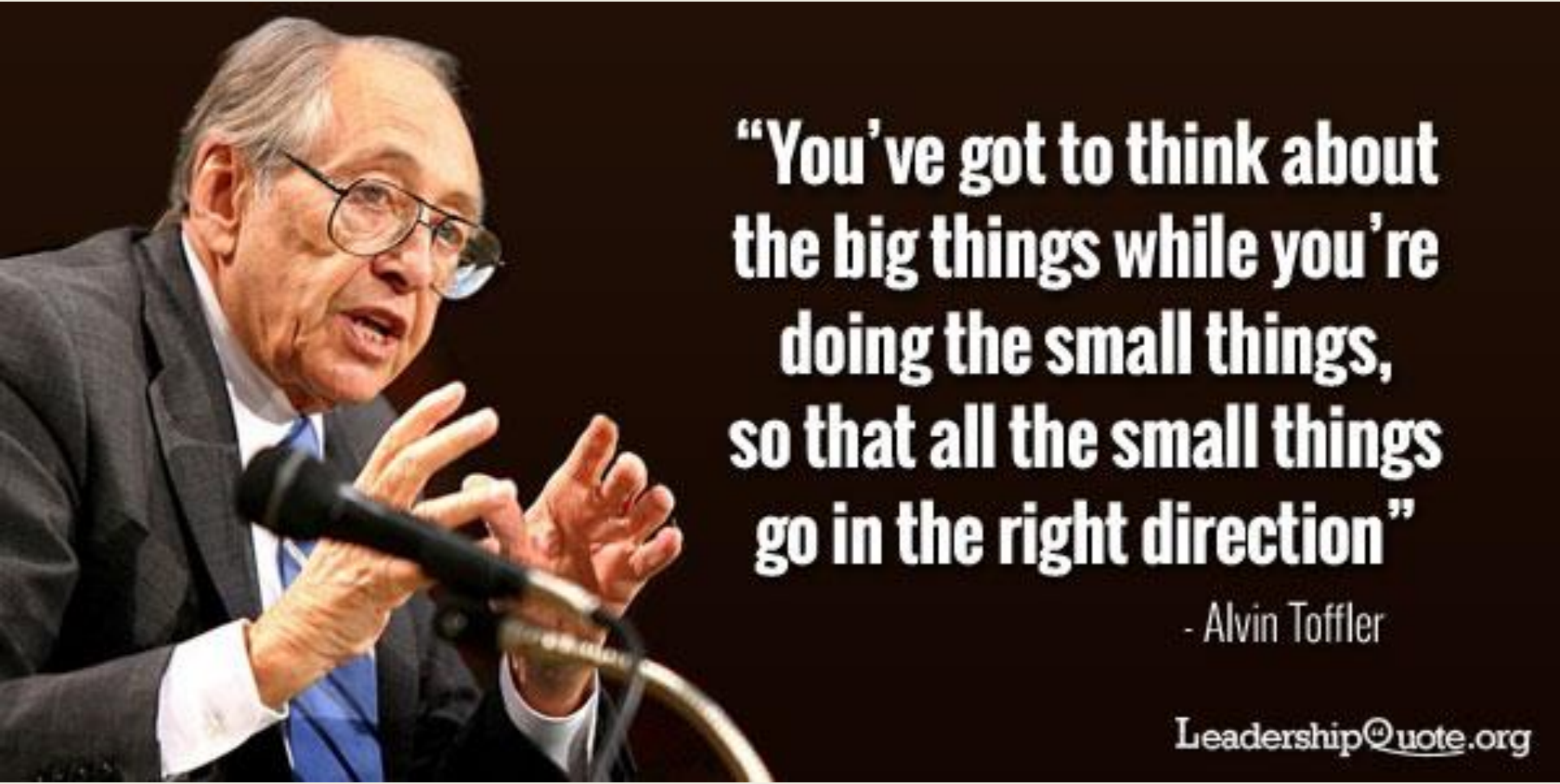
What is the Best Way to Approach Application Modernization?

AO
Adsero Optima™



Think Modern


We cannot solve
our problems with
the same thinking
we used when
we created them
~ Albert Einstein

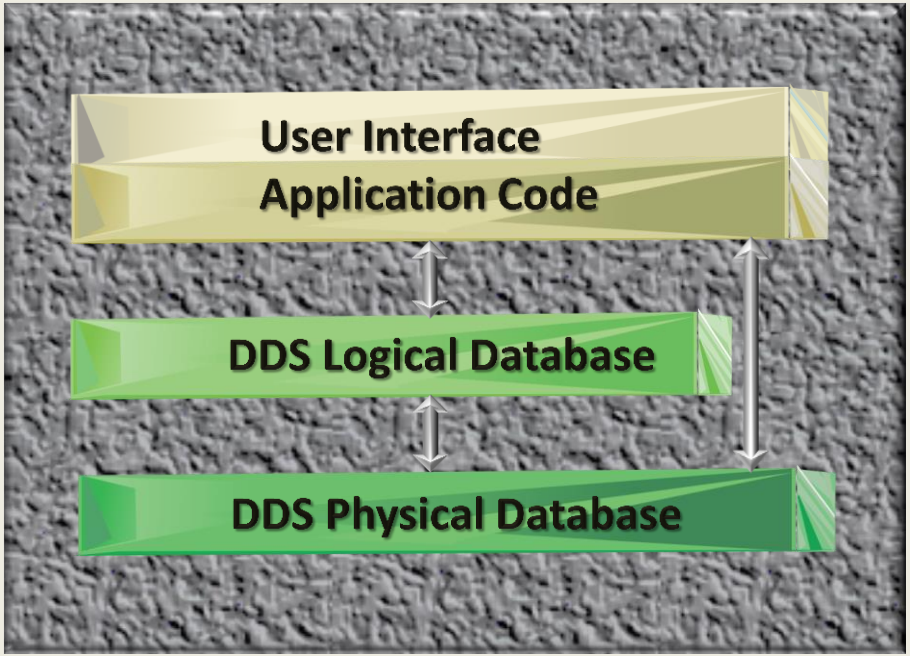

"You've got to think about the big things while you're doing the small things, so that all the small things go in the right direction"
- Alvin Toffler
LeadershipQuote.org

Legacy

Modernization

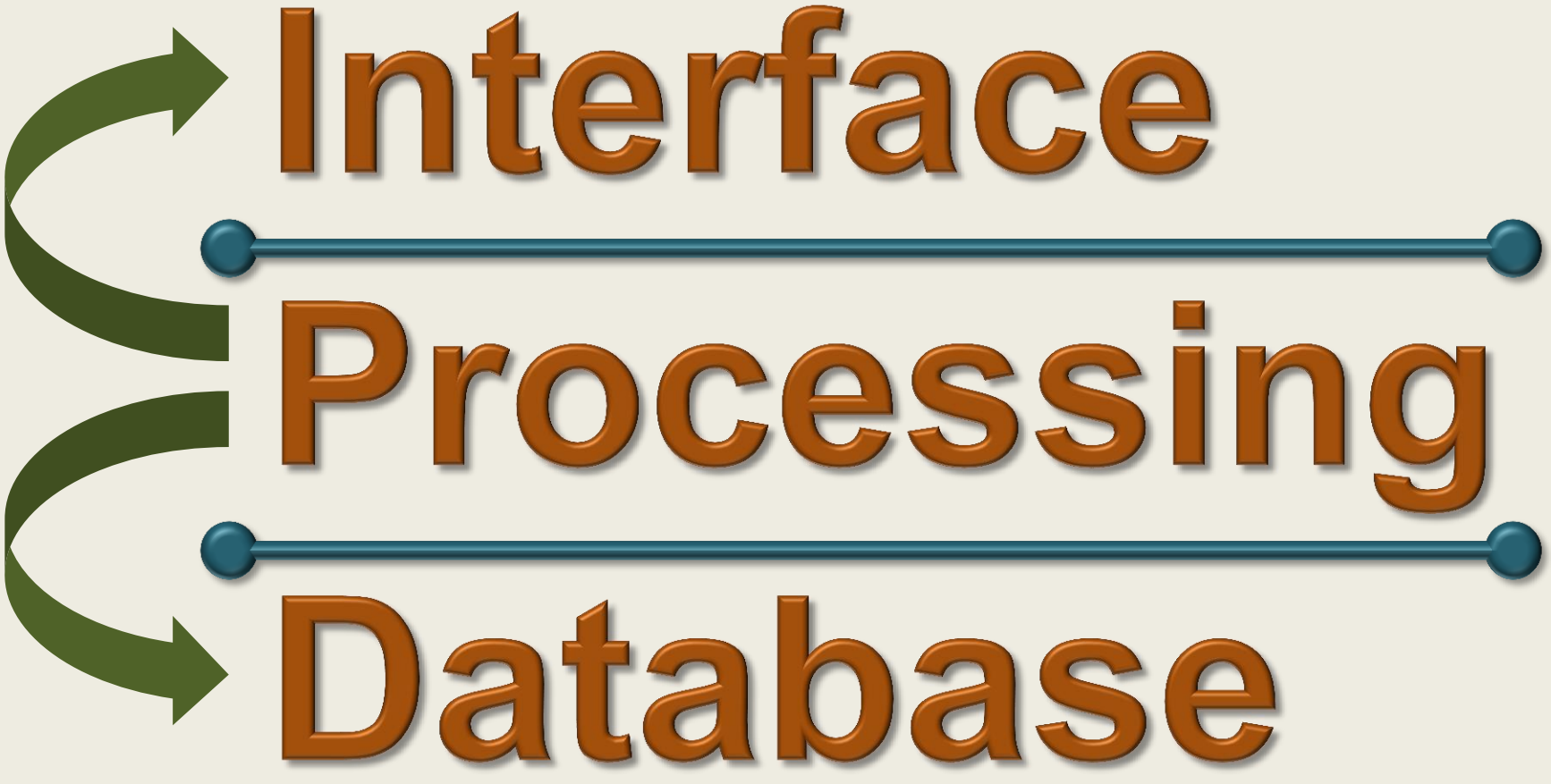
Modern

Legacy

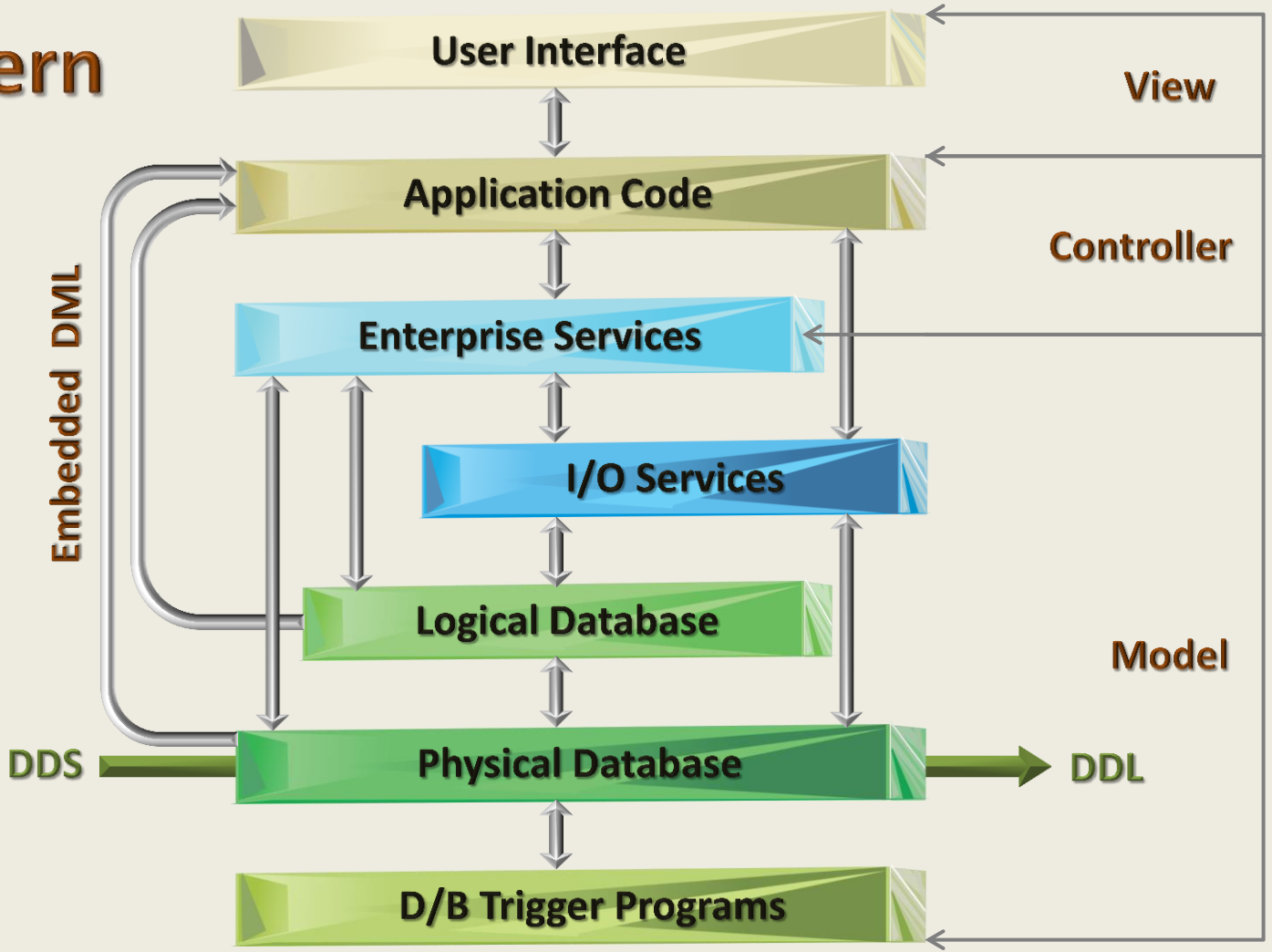

User Interface
Application Code
DDS Logical Database
DDS Physical Database

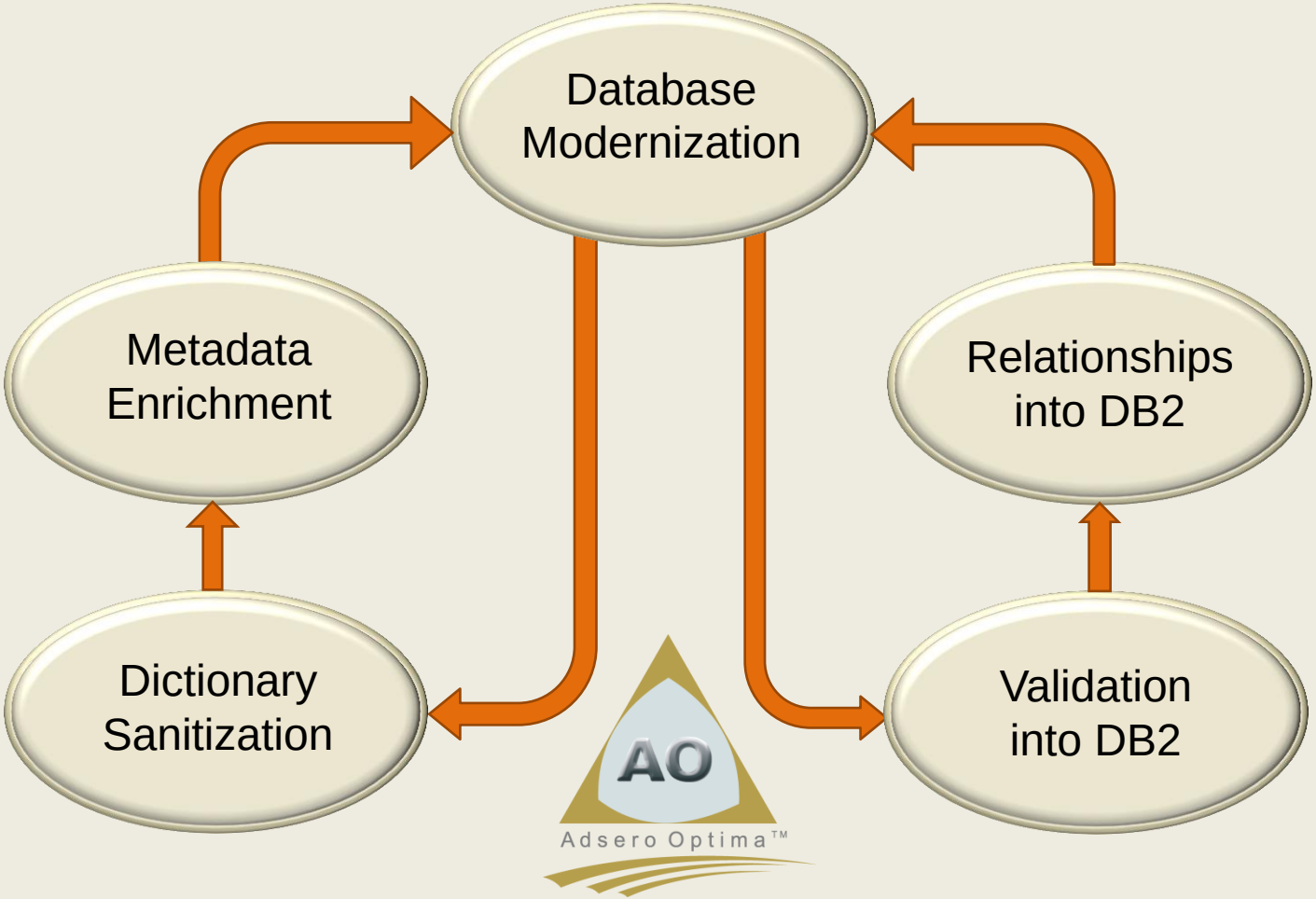
Model
View
Controller

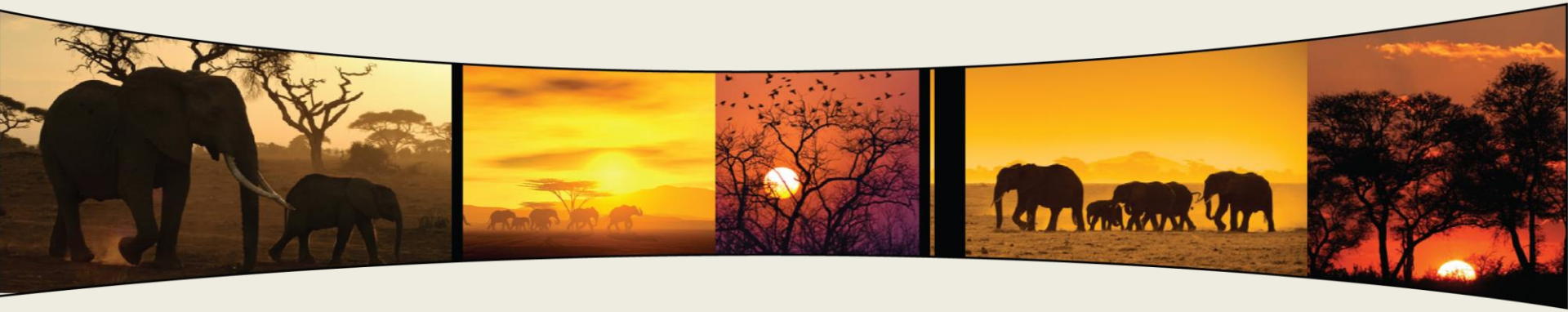
View
Controller
Model


Interface
Processing
Database

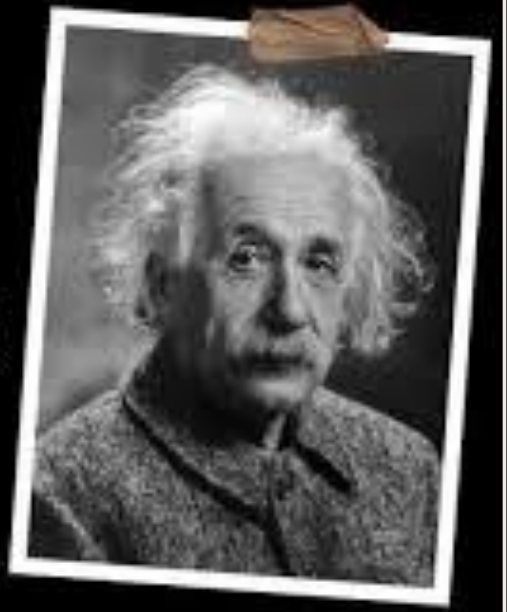
Modern

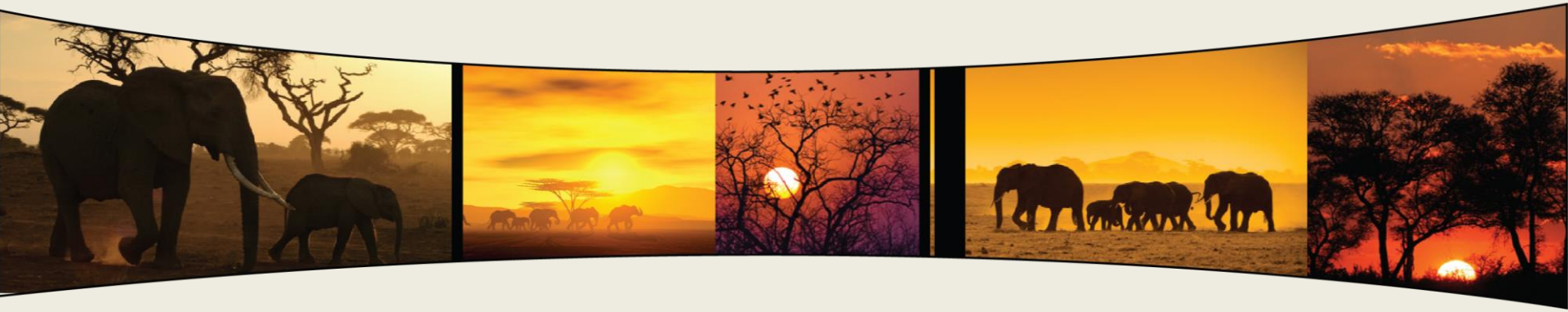

User Interface
View
Application Code
Controller
Embedded DML
Enterprise Services
I/O Services
Logical Database
Model
DDS
Physical Database
DDL
D/B Trigger Programs


Database Modernization
Metadata Enrichment
Relationships into DB2
Dictionary Sanitization
Validation into DB2
AO
Adsero Optima™

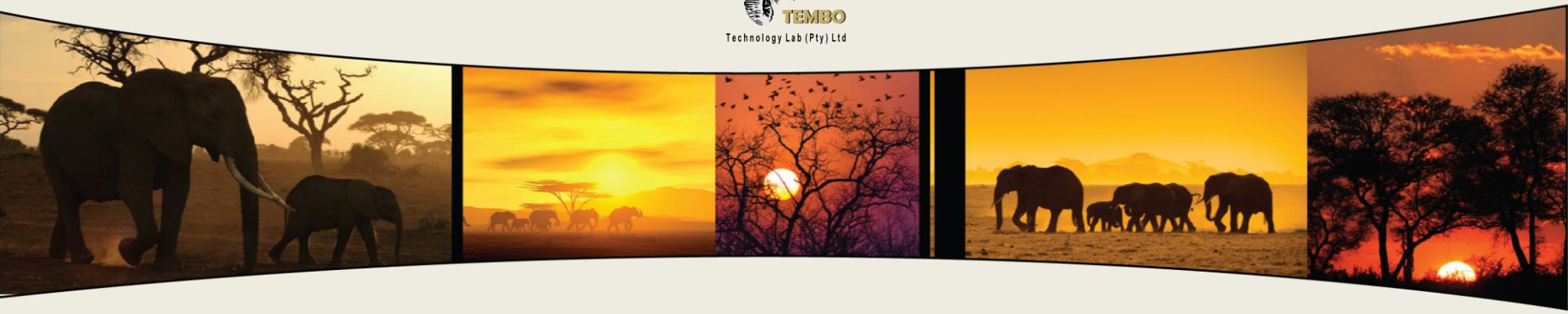


"Education is not the learning of facts, but the training of the mind to think."
-Albert Einstein





TEMBO
Technology Lab (Pty) Ltd

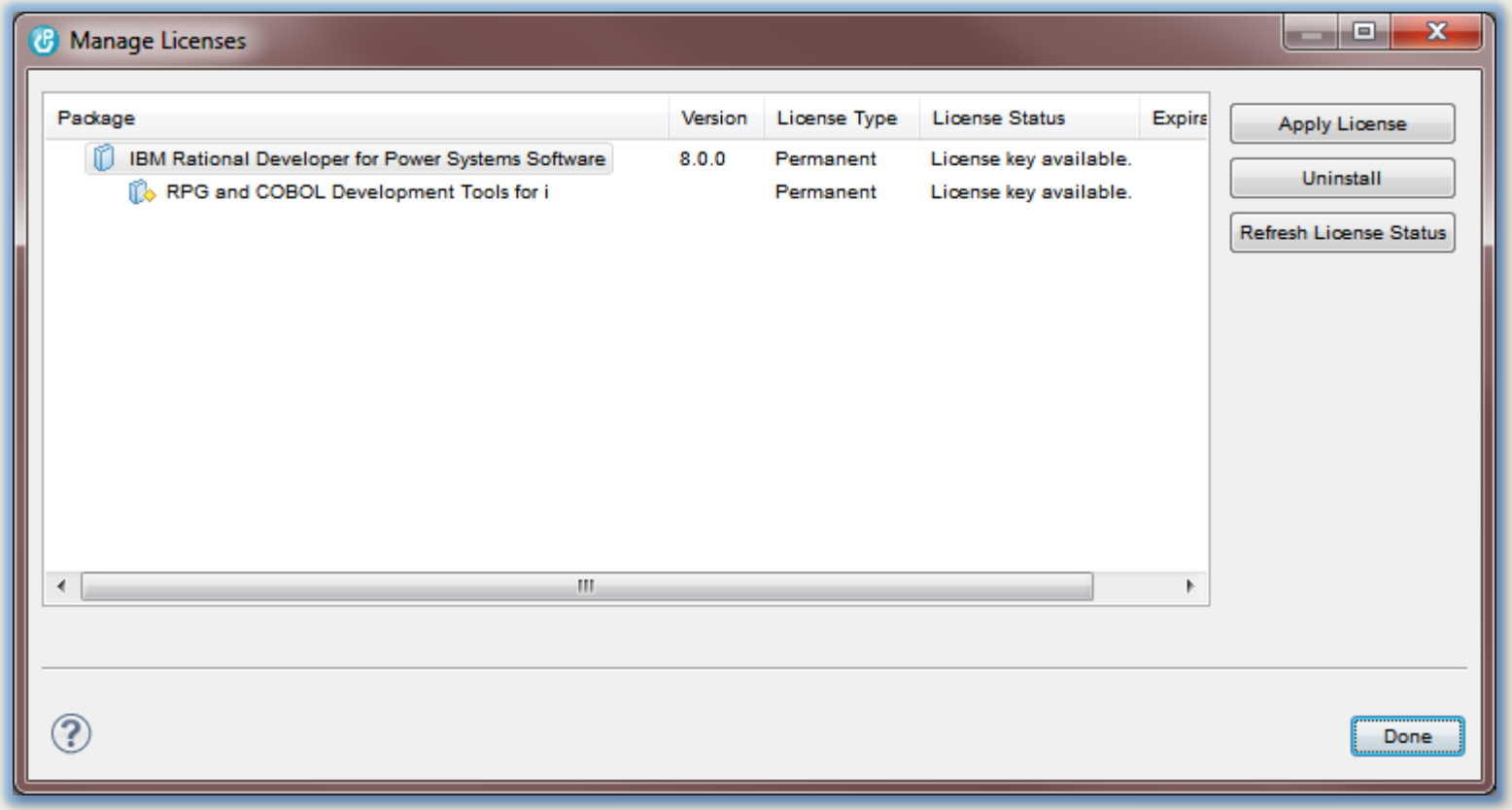


IBM Rational Developer for Power Systems (WDSC/RDP/RDi)

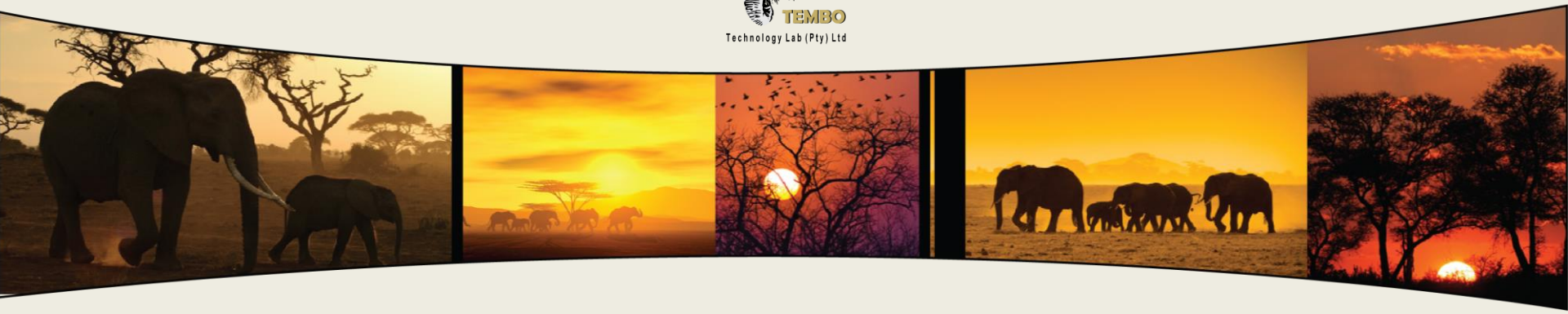
IBM Rational Developer for Power Systems

- RDi is a highly functional and flexible system management and code development interface (IDE) for, but not limited to, the IBM i.
- As a result of this capability and RDi's ability to allow the user to personalize and customize the interface to suite individual preferences, it is not possible to provide a step-by-step "how to" for the product.
- This "Walk-About" session will introduce you to the multiple capabilities of RDP and provide guidelines for the use and customization of the features and functions.
- RDi will be used exclusively throughout the following sessions of this course and will therefore provide many opportunities to consolidate understanding and skill with the product.

IBM Rational Developer for Power Systems



TEMBO
Technology Lab (Pty) Ltd



DB2 Relational Database
Physical Data Store

DB2 Relational Database (Physical Data Store)

- The DB2 physical database definition and it's data form the life-blood of a company.

- Systems development becomes "DATA-CENTRIC".
  - Collects, Manages and Presents data in the form of information that the company requires to operate in a profitable manner.

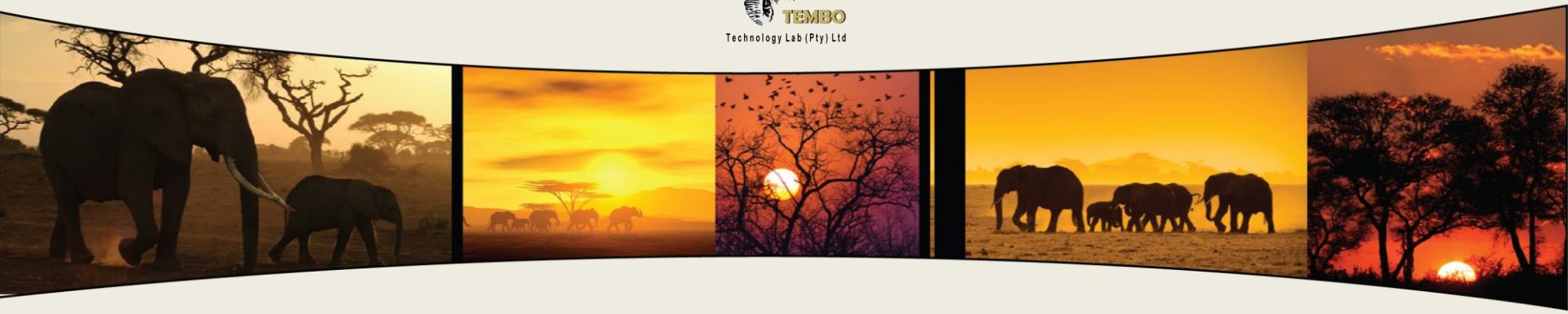
DB2 Relational Database (Physical Data Store)

- The Database built into the IBMi is a “Relational Database”.
- To ensure maximum flexibility, productivity and efficiency from a relational database, the data should be normalized to at least 3NF (3rd Normal Form).

DB2 Relational Database (Physical Data Store)

- The modern MVC concept separates the Database (Model) from the 2 other components (View and Controller).
- This requires the Database to become truly “SELF-AWARE”
- Using mainly “CONSTRAINTS” and “TRIGGERS”

TEMBO
Technology Lab (Pty) Ltd



DBMS - Codd's 12 Rules

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell.

Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value).

No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment.

This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalogue, known as data dictionary, which can be accessed by authorized users.

Users can use the same query language to access the catalogue which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations.

This language can be used directly or by means of some application.

If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion.

This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database.

Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application).

Any change in logical data must not affect the applications using it.

For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application.

This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it.

All its integrity constraints can be independently modified without the need of any change in the application.

This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

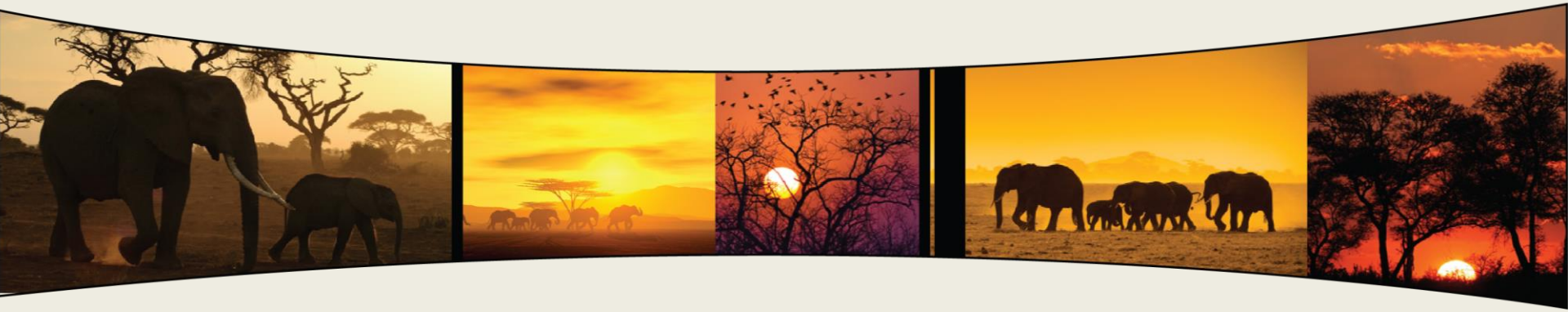
The end-user must not be able to see that the data is distributed over various locations.

Users should always get the impression that the data is located at one site only.

This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.



Normalizing Data: Theory

In the design of a relational database management system (RDBMS), the process of organizing data to minimize redundancy is called normalization.

In the design of a relational database management system (RDBMS), the process of organizing data to minimize redundancy is called **normalization**. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.
Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the First Normal Form (1NF) in 1970.[11] Codd went on to define the Second Normal Form (2NF) and Third Normal Form (3NF) in 1971,[12] and Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974.[13] Higher normal forms were defined by other theorists in subsequent years, the most recent being the Sixth Normal Form (6NF) introduced by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.[14]
Informally, a relational database table (the computerized representation of a relation) is often described as "normalized" if it is in the Third Normal Form.[15] Most 3NF tables are free of insertion, update, and deletion anomalies, i.e. in most cases 3NF tables adhere to BCNF, 4NF, and 5NF (but typically not 6NF).
A standard piece of database design guidance is that the designer should create a fully normalized design; selective denormalization can subsequently be performed for performance reasons.[16] However, some modeling disciplines, such as the dimensional modeling approach to data warehouse design, explicitly recommend non-normalized designs, i.e. designs that in large part do not adhere to 3NF.[17]
Free the database of modification anomalies
An **update anomaly**. Employee 519 is shown as having different addresses on different records.
An **insertion anomaly**. Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded.
A **deletion anomaly**. All information about Dr. Giddens is lost when he temporarily ceases to be assigned to any courses.
When an attempt is made to modify (update, insert into, or delete from) a table, undesired side-effects may follow. Not all tables can suffer from these side-effects; rather, the side-effects can only arise in tables that have not been sufficiently normalized. An insufficiently normalized table might have one or more of the following characteristics:
The same information can be expressed on multiple rows; therefore updates to the table may result in logical inconsistencies. For example, each record in an "Employees' Skills" table might contain an Employee ID, Employee Address, and Skill; thus a change of address for a particular employee will potentially need to be applied to multiple records (one for each of his skills). If the update is not carried through successfully—if, that is, the employee's address is updated on some records but not others—then the table is left in an inconsistent state. Specifically, the table provides conflicting answers to the question of what this particular employee's address is. This phenomenon is known as an **update anomaly**.
There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "Faculty and Their Courses" table might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code—thus we can record the details of any faculty member who teaches at least one course, but we cannot record the details of a newly-hired faculty member who has not yet been assigned to teach any courses except by setting the Course Code to null. This phenomenon is known as an **insertion anomaly**.
There are circumstances in which the deletion of data representing certain facts necessitates the deletion of data representing completely different facts. The "Faculty and Their Courses" table described in the previous example suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member. This phenomenon is known as a **deletion anomaly**.
Minimize redesign when extending the database structure
When a fully normalized database structure is extended to allow it to accommodate new types of data, the pre-existing aspects of the database structure can remain largely or entirely unchanged. As a result, applications interacting with the database are minimally affected.
[edit] Make the data model more informative to users
Normalized tables, and the relationship between one normalized table and another, mirror real-world concepts and their interrelationships.
[edit] Avoid bias towards any particular pattern of querying
Normalized tables are suitable for general-purpose querying. This means any queries against these tables, including future queries whose details cannot be anticipated, are supported. In contrast, tables that are not normalized lend themselves to some types of queries, but not others.
For example, consider an online bookseller whose customers maintain wishlists of books they'd like to have. For the obvious, anticipated query—what books does this customer want?—it's enough to store the customer's wishlist in the table as, say, a homogeneous string of authors and titles.
With this design, though, the database can answer only that one single query. It cannot by itself answer interesting but unanticipated queries: What is the most-wished-for book? Which customers are interested in WWII espionage? How does Lord Byron stack up against his contemporary poets? Answers to these questions must come from special adaptive tools completely separate from the database. One tool might be software written especially to handle such queries. This special adaptive software has just one single purpose: in effect to normalize the non-normalized field.
Unforeseen queries can be answered trivially, and entirely within the database framework, with a normalized table.
[edit] Example
Querying and manipulating the data within an unnormalized data structure, such as the following non-1NF representation of customers' credit card transactions, involves more complexity than is really necessary
To each customer there corresponds a *repeating group* of transactions. The automated evaluation of any query relating to customers' transactions therefore would broadly involve two stages:
Unpacking one or more customers' groups of transactions allowing the individual transactions in a group to be examined, and
Deriving a query result based on the results of the first stage
For example, in order to find out the monetary sum of all transactions that occurred in October 2003 for all customers, the system would have to know that it must first unpack the *Transactions* group of each customer, then sum the *Amounts* of all transactions thus obtained where the *Date* of the transaction falls in October 2003.
One of Codd's important insights was that this structural complexity could always be removed completely, leading to much greater power and flexibility in the way queries could be formulated (by users and applications) and evaluated (by the DBMS). The normalized equivalent of the structure above would look like this:
Functional dependency In a given table, an attribute *Y* is said to have a functional dependency on a set of attributes *X* (written *X* → *Y*) if and only if each *X* value is associated with precisely one *Y* value. For example, in an "Employee" table that includes the attributes "Employee ID" and "Employee Date of Birth", the functional dependency {Employee ID} → {Employee Date of Birth} would hold. It follows from the previous two sentences that each {Employee ID} is associated with precisely one {Employee Date of Birth}.**Trivial functional dependency**A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.Full functional dependencyAn attribute is fully functionally dependent on a set of attributes X if it isfunctionally dependent on X, and
not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a *full* functional dependency, because it is also dependent on {Employee ID}.
Transitive dependencyA transitive dependency is an indirect functional dependency, one in which *X*→*Z* only by virtue of *X*→*Y* and *Y*→*Z*.Multivalued dependencyA multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.Join dependencyA table *T* is subject to a join dependency if *T* can always be recreated by joining multiple tables each having a subset of the attributes of *T*.SuperkeyA superkey is a combination of attributes that can be used to uniquely identify a database record. A table might have many superkeys.Candidate keyA candidate key is a special subset of superkeys that do not have any extraneous information in them: it is a minimal superkey.Examples: Imagine a table with the fields <Name>, <Age>, <SSN> and <Phone Extension>. This table has many possible superkeys. Three of these are <SSN>, <Phone Extension, Name> and <SSN, Name>. Of those listed, only <SSN> is a candidate key, as the others contain information not necessary to uniquely identify records ('SSN' here refers to Social Security Number, which is unique to each person).
Non-prime attributeA non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.Prime attributeA prime attribute, conversely, is an attribute that does occur in any candidate key.Primary keyMost DBMSs require a table to be defined as having a single unique key, rather than a number of possible unique keys. A primary key is a key which the database designer has designated for this purpose.**[edit] Normal forms**
The **normal forms** (abbrev. **NF**) of relational database theory provide criteria for determining a table's degree of vulnerability to logical inconsistencies and anomalies. The higher the normal form applicable to a table, the less vulnerable it is to inconsistencies and anomalies. Each table has a "**highest normal form**" (**HNF**): by definition, a table always meets the requirements of its HNF and of all normal forms lower than its HNF; also by definition, a table fails to meet the requirements of any normal form higher than its HNF.
The normal forms are applicable to individual tables; to say that an entire database is in normal form *n* is to say that all of its tables are in normal form *n*.
Newcomers to database design sometimes suppose that normalization proceeds in an iterative fashion, i.e. a 1NF design is first normalized to 2NF, then to 3NF, and so on. This is not an accurate description of how normalization typically works. A sensibly designed table is likely to be in 3NF on the first attempt; furthermore, if it is 3NF, it is overwhelmingly likely to have an HNF of 5NF. Achieving the "higher" normal forms (above 3NF) does not usually require an extra expenditure of effort on the part of the designer, because 3NF tables usually need no modification to meet the requirements of these higher normal forms.
The main normal forms are summarized below.
. **Denormalization**
Main article: Denormalization
Databases intended for online transaction processing (OLTP) are typically more normalized than databases intended for online analytical processing (OLAP). OLTP applications are characterized by a high volume of small transactions such as updating a sales record at a supermarket checkout counter. The expectation is that each transaction will leave the database in a consistent state. By contrast, databases intended for OLAP operations are primarily "read mostly" databases. OLAP applications tend to extract historical data that has accumulated over a long period of time. For such databases, redundant or "denormalized" data may facilitate business intelligence applications. Specifically, dimensional tables in a star schema often contain denormalized data. The denormalized or redundant data must be carefully controlled during extract, transform, load (ETL) processing, and users should not be permitted to see the data until it is in a consistent state. The normalized alternative to the star schema is the snowflake schema. In many cases, the need for denormalization has waned as computers and RDBMS software have become more powerful, but since data volumes have generally increased along with hardware and software performance, OLAP databases often still use denormalized schemas.
Denormalization is also used to improve performance on smaller computers as in computerized cash-registers and mobile devices, since these may use the data for look-up only (e.g. price lookups). Denormalization may also be used when no RDBMS exists for a platform (such as Palm), or no changes are to be made to the data and a swift response is crucial.
[edit] Non-first normal form (NF² or N1NF)
In recognition that denormalization can be deliberate and useful, the non-first normal form is a definition of database designs which do not conform to first normal form, by allowing "sets and sets of sets to be attribute domains" (Schek 1982). The languages used to query and manipulate data in the model must be extended accordingly to support such values.
One way of looking at this is to consider such structured values as being specialized types of values (domains), with their own domain-specific languages. However, what is usually meant by non-1NF models is the approach in which the relational model and the languages used to query it are extended with a general mechanism for such structure; for instance, the nested relational model supports the use of relations as domain values, by adding two additional operators (*nest* and *unnest*) to the relational algebra that can create and flatten nested relations, respectively.
Consider the following table:

Normalizing Data: Theory

- First Normal Form (1NF)

(**1NF or Minimal Form**) is a normal form used in database normalization. A relational database table that adheres to 1NF is one that meets a certain minimum set of criteria. These criteria are basically concerned with ensuring that the table is a faithful representation of a **relation** and that it is free of **repeating groups**.

According to Date's definition of 1NF, a table is in 1NF, **if and only if,** it is "isomorphic to some relation", which means, specifically, that it satisfies the following five conditions:

1. There's no top-to-bottom ordering to the rows.
2. There's no left-to-right ordering to the columns.
3. There are no duplicate rows.
4. Every row-and-column intersection contains exactly one value from the applicable domain (and nothing else).
5. All columns are regular [i.e. rows have no hidden components such as row IDs, object IDs, or hidden timestamps].

Violation of any of these conditions would mean that the table is not strictly relational, and therefore that it is not in first normal form.

Normalizing Data: Theory

- First Normal Form (1NF)

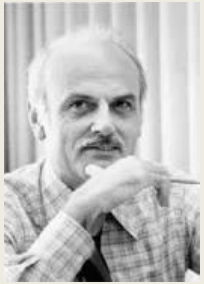
Examples of tables (or views) that would not meet this definition of first normal form are:

- A table that lacks a unique key. Such a table would be able to accommodate duplicate rows, in violation of condition 3.
- A view whose definition mandates that results be returned in a particular order, so that the row-ordering is an intrinsic and meaningful aspect of the view. This violates condition 1. The tuples in true relations are not ordered with respect to each other.
- A table with at least one nullable attribute. A nullable attribute would be in violation of condition 4, which requires every field to contain exactly one value from its column's domain. It should be noted, however, that this aspect of condition 4 is controversial. It marks an important departure from Codd's later vision of the relational model, which made explicit provision for nulls.

Normalizing Data: Theory

- First Normal Form (Summary)
  - Each Physical File/Table must have a unique key.
  - A record/row may not contain recurring values.

Normalizing Data: Theory



- Second Normal Form (2NF)

  Originally defined by E.F. Codd in 1971.

  A table that is in first normal form (1NF) must meet one additional criteria if it is to qualify for second normal form.

  A 1NF table is 2NF, **if and only if,** all its non-prime attributes are functionally dependent on the whole of every candidate key.

  A non-prime attribute is one that does not belong to any candidate key.

  Note that when a 1NF table has no composite candidate keys (candidate keys consisting of more than one attribute), the table is automatically in 2NF.

Normalizing Data: Theory

- Third Normal Form (3NF)

  Generally the normal form used in database normalization

  3NF was originally defined by E.F. Codd in 1971.

  It offers the best balance between flexibility and performance for the development of applications

  - Codd's definition states that a table is in 3NF, **if and only if,** both of the following conditions hold:

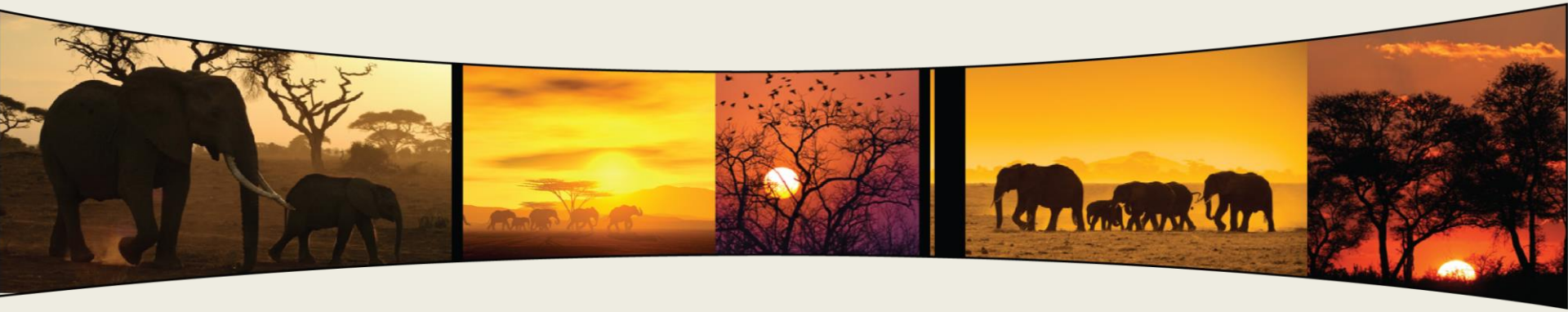
    – The Table (T) is in second normal form (2NF)

    – Every non-prime attribute of T is non-transitively dependent (i.e. directly dependent) on every candidate key of T.

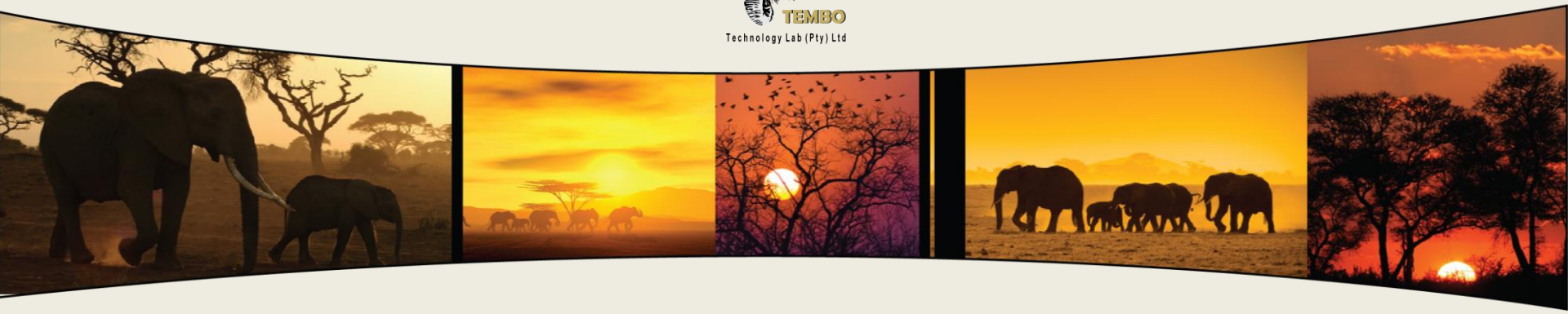
Normalizing Data: Summary

- Requiring existence of "the key" ensures that the table is in 1NF
- Requiring that non-key attributes be dependent on "the whole key" ensures 2NF
- Further requiring that non-key attributes be dependent on "nothing but the key" ensures 3NF.

“The KEY, the whole KEY and nothing but the KEY”



TEMBO
Technology Lab (Pty) Ltd



Data Dictionary

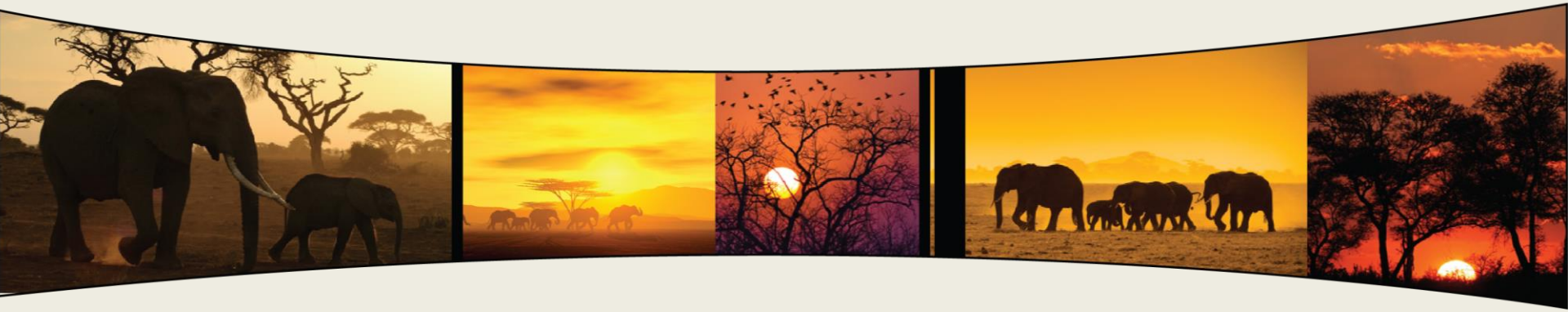
Data Dictionary

A **data dictionary**, or Metadata Repository, as defined in the *IBM Dictionary of Computing*, is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format.

- Many things to many people
- Definition of terms
- One “thing” –
  one term and one definition

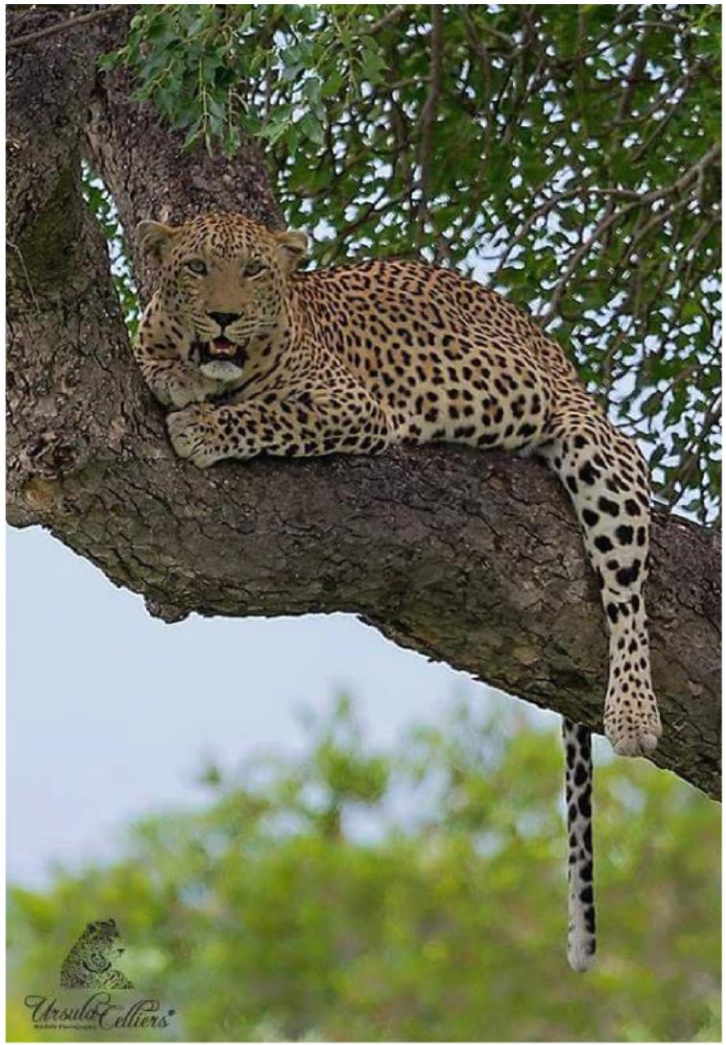
Data Dictionary - Benefits

- Consistent naming and definition
  - Facilitates “Data-Centric” systems development
  - Assists in the development of “Data Analytics”
  - Cognitive Computing a.k.a. Rule based Systems (Watson and AI techniques)

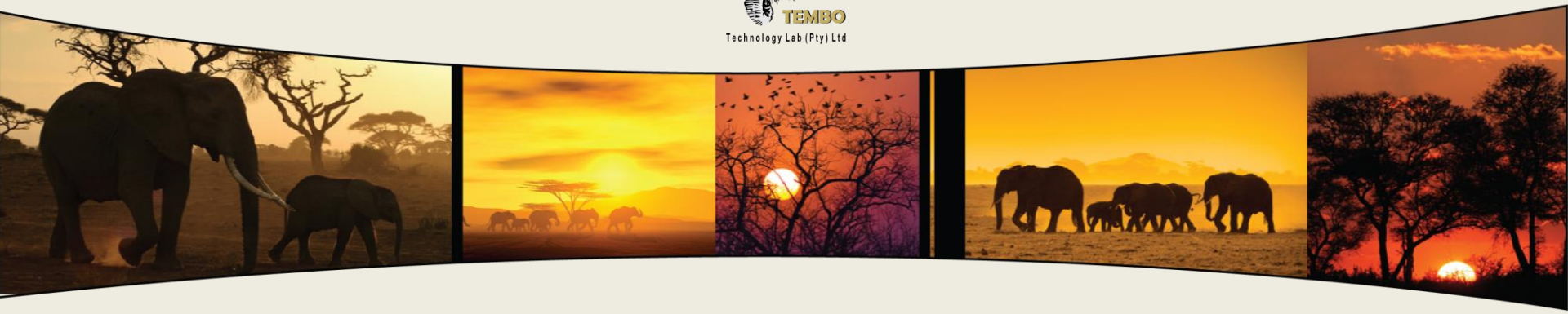


AO Advanced Modernization Workshop

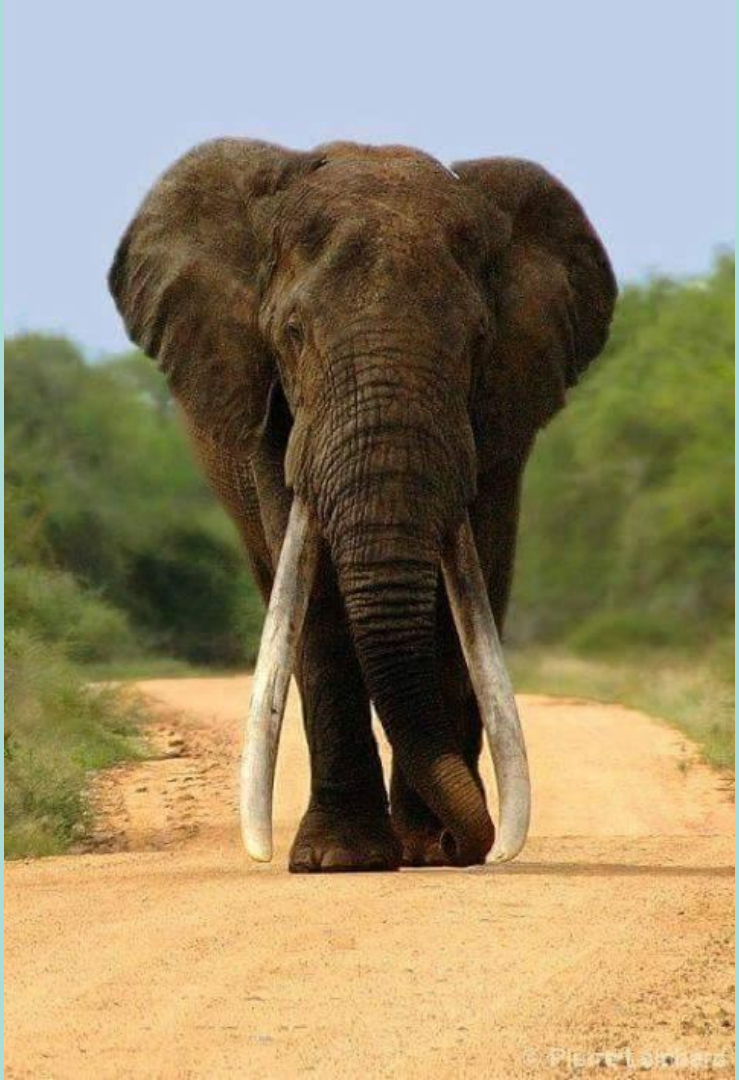
Leopards



TEMBO
Technology Lab (Pty) Ltd



DB2 Relational Database Components



Kruger National Park's
Biggest Tusker

"Duke"

Died in 2001 of natural causes


No thanks!
We are too busy

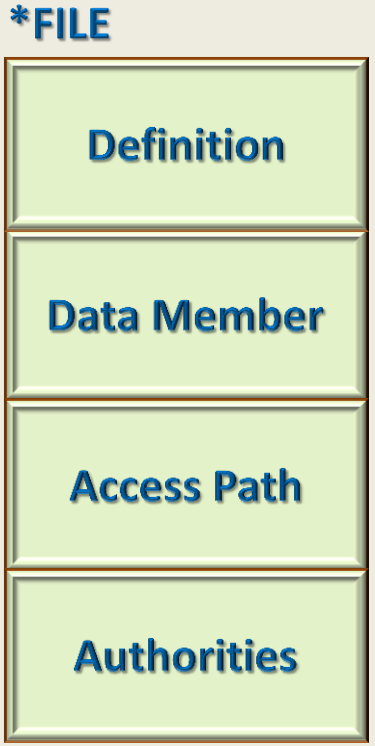
DB2 Components

- Physical Database
- Constraints
  - Primary Keys
  - Unique Keys
  - Check Constraints
  - Referential Constraints
- Journals & Receivers
- Commitment Control
- Trigger Programs
- Input/Output Servers
- Enterprise Servers

Physical Database

DDS Physical Files

DDL Tables



Physical Definitions

*FILE Definition

- Constraints
  - Primary Key
  - Unique Key
  - Check
  - Referential
- Journals
- Triggers
  - Before/Insert
  - Before/Delete
  - Before/Update
  - After/Insert
  - After/Delete
  - After/Update
  - After/Read

Maximum of 300/file

Maximum of 300/file

RDBMS

The *PAG Object

Must be in memory for the job to execute

*PAG

- Job Attributes
- File Definition Table x n
- File Buffers x n
- File Record Pointer x n
- Job Variable Values
- … Other Job Specific Info.

Primary Key Constraints

One or more fields make up the primary key.

The defined order provides the index sequence.

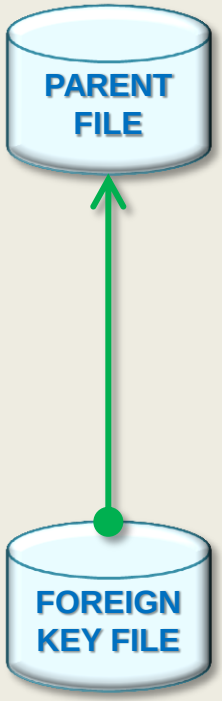
Only the primary is available for RPG keyed access.

Unique Constraints

One or more fields making up a unique key.

Used by RDBMS. Not available for direct client usage.

Referential Constraints

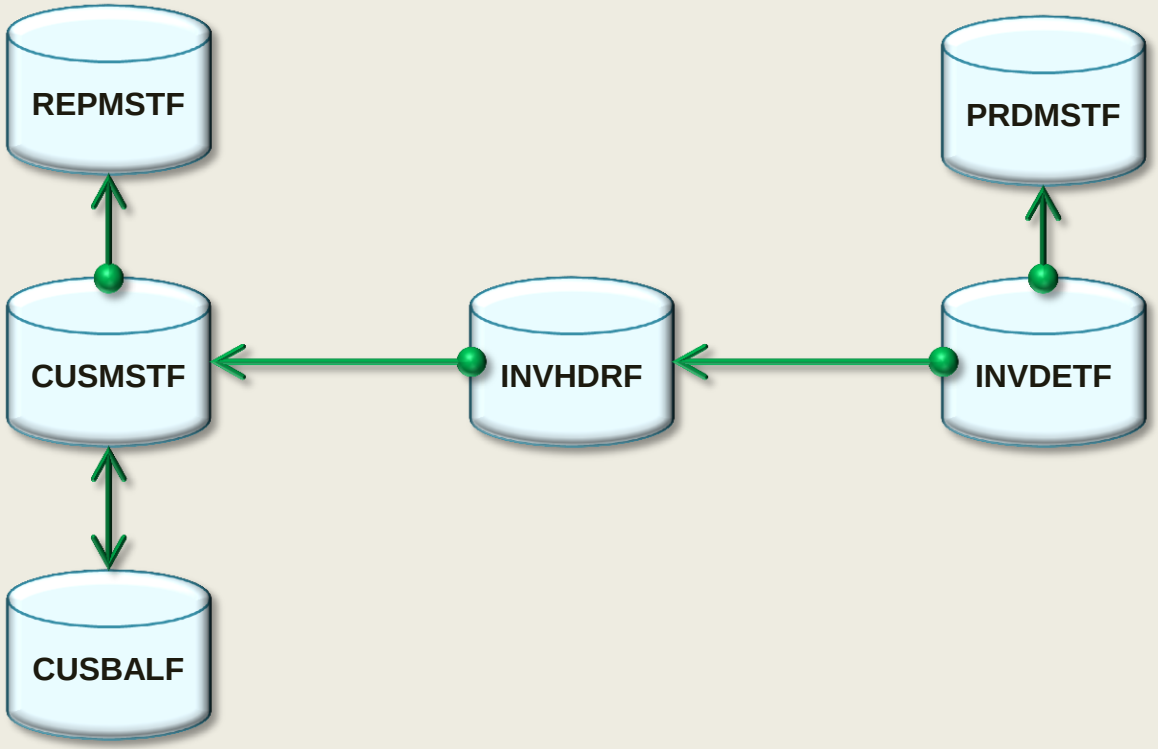


Requires a Primary or Unique Key.

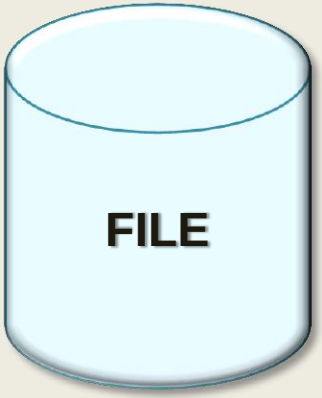
Constraint is defined here

Requires Fields Matching the Parent Primary or a Unique Key Exactly.

Referential Constraints

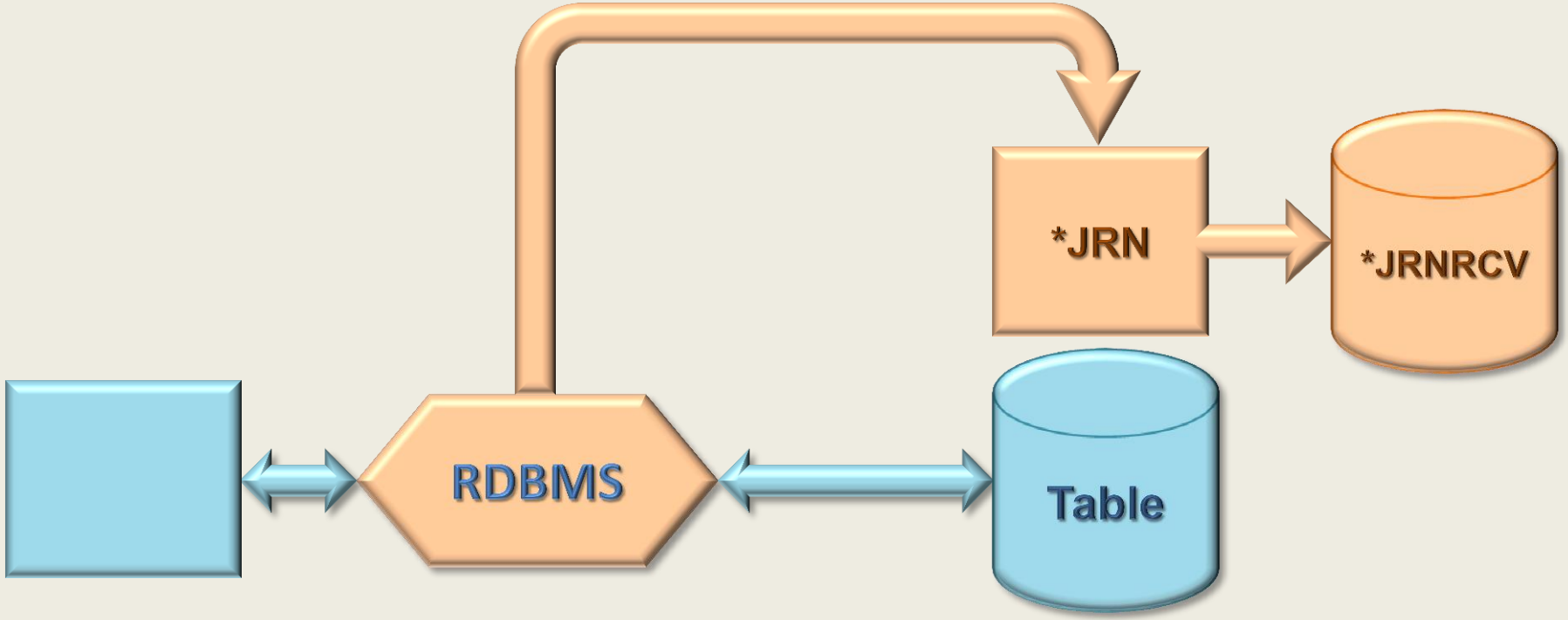


Check Constraints

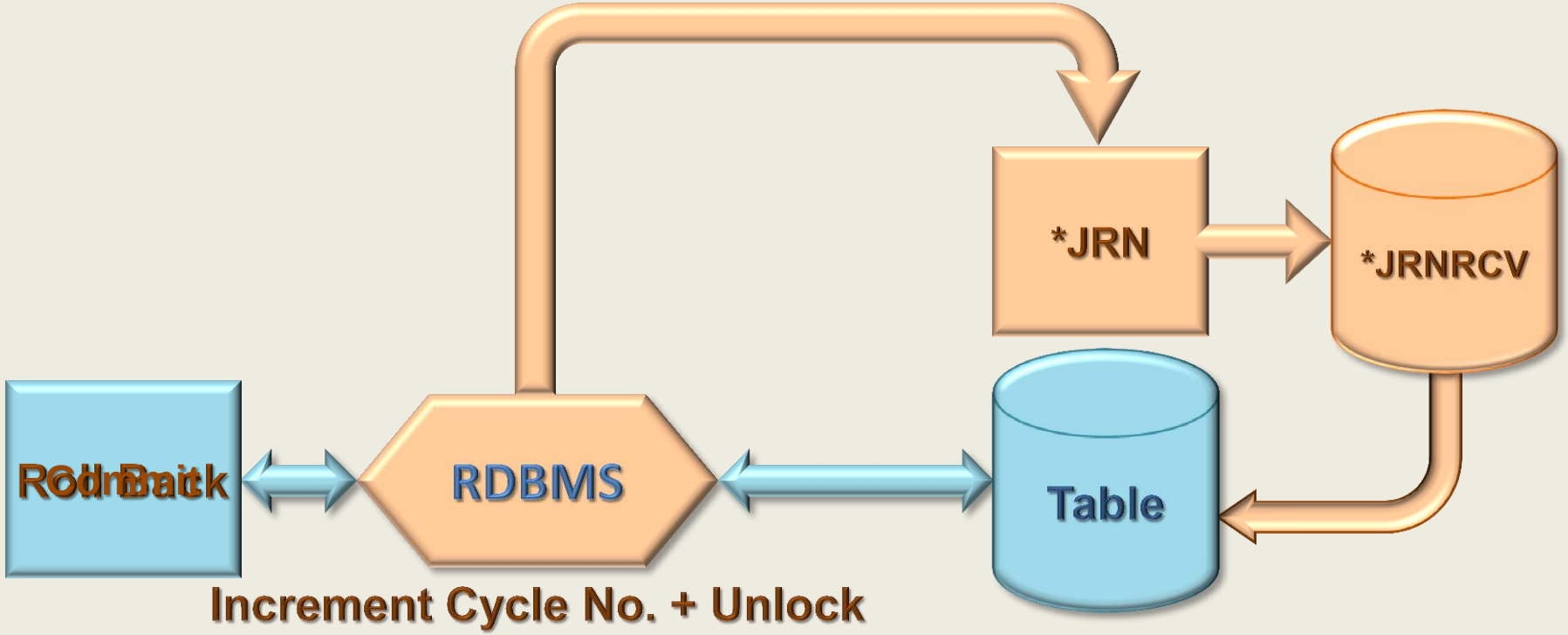


A SQL “WHERE” clause, without the WHERE defining conditions to be checked as ‘TRUE’

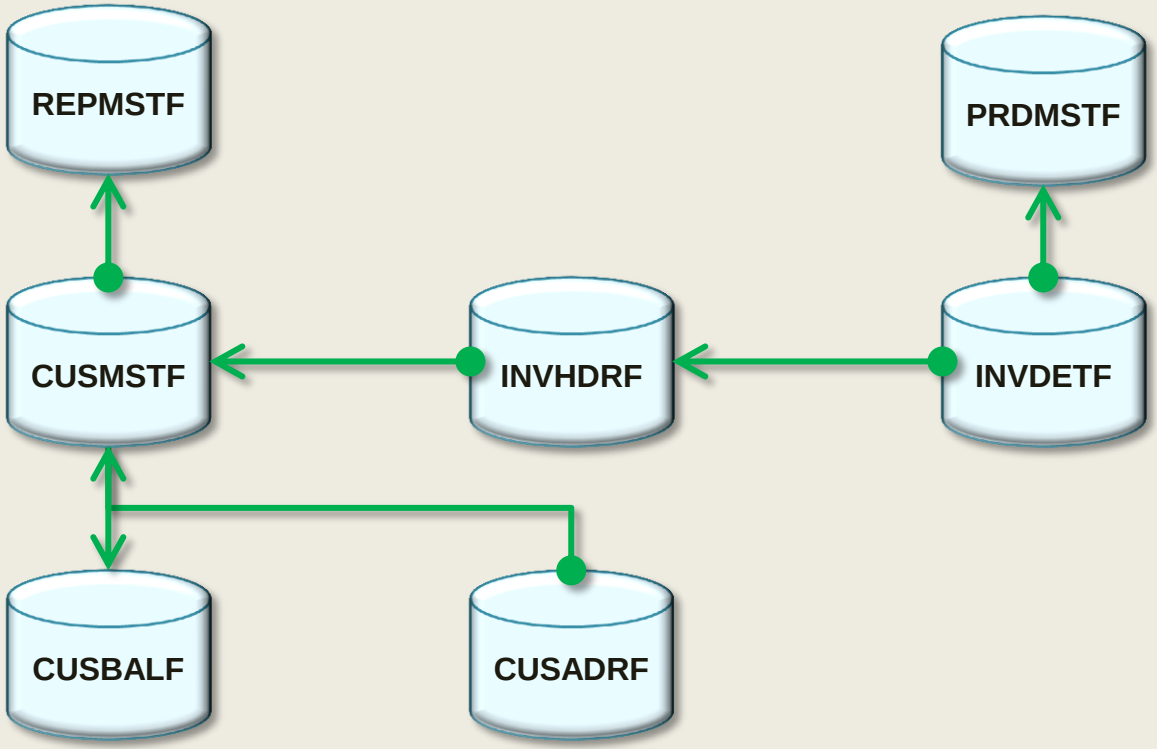
Journals & Receivers

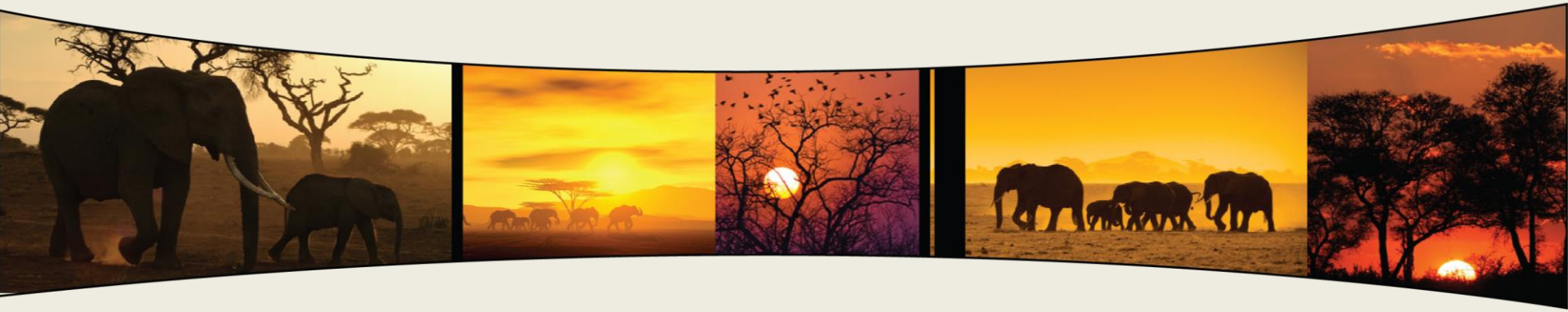


Commitment Control

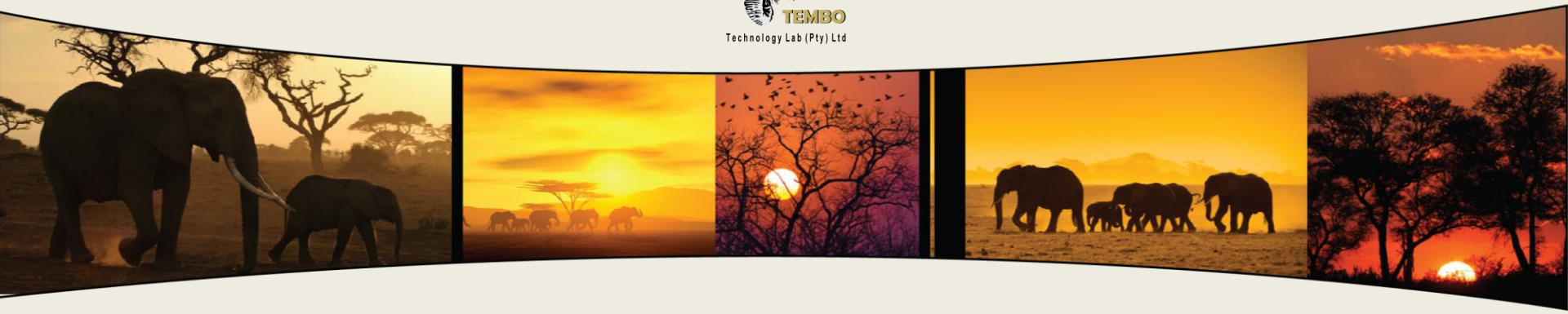


Referential Constraints





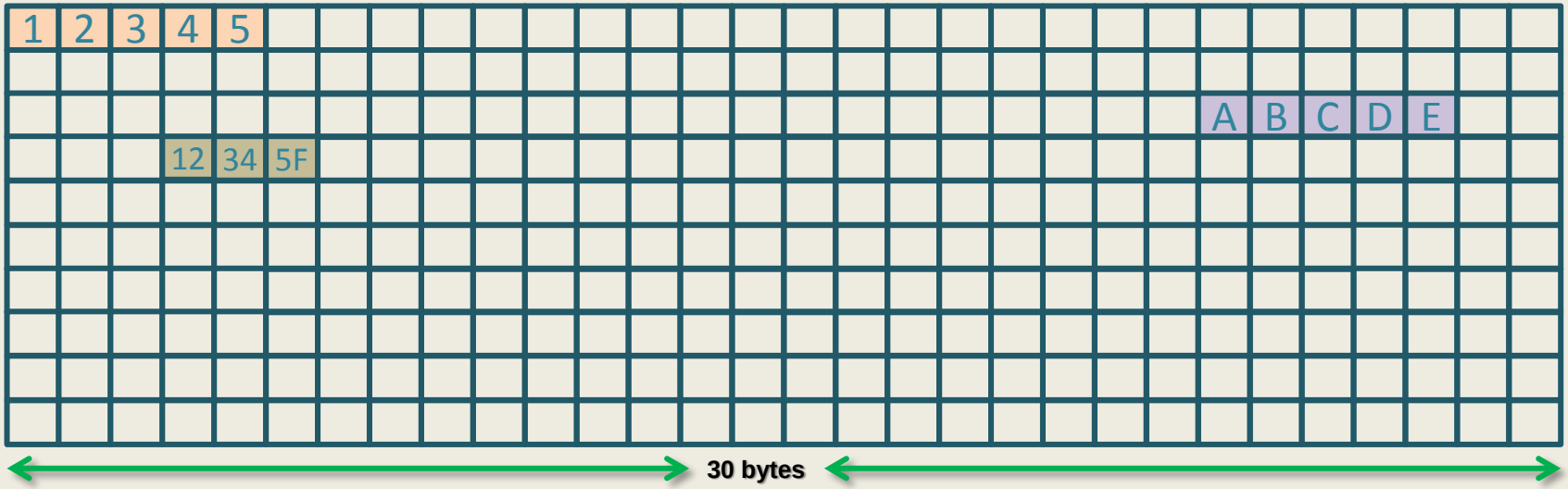
TEMBO
Technology Lab (Pty) Ltd



Pointers

Pointers

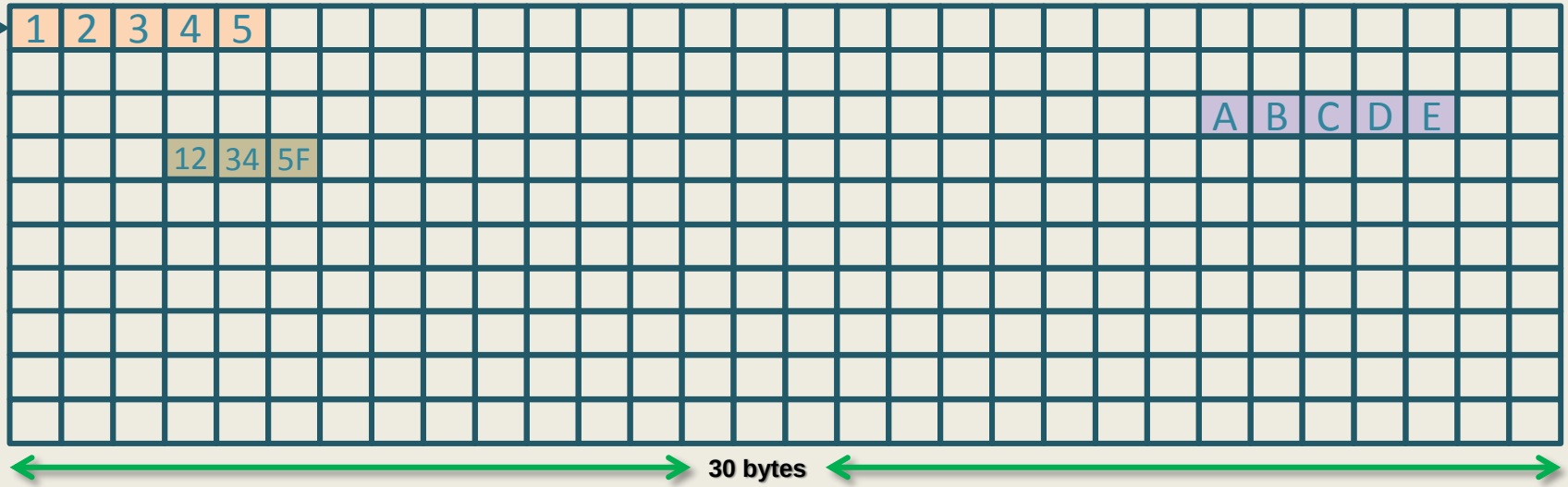
```
SPP = 1C5736254E001000  d SPP      s         *
                        d Offset   s       10U 0

                        d Field1   s         5       based(SPP)
                        d Field2   s         5p 0    based(SPP)
                        d Field3   s         5s 0    based(SPP)
```



Pointers

```
SPP = 1C5736254E001000
Offset = 83
```

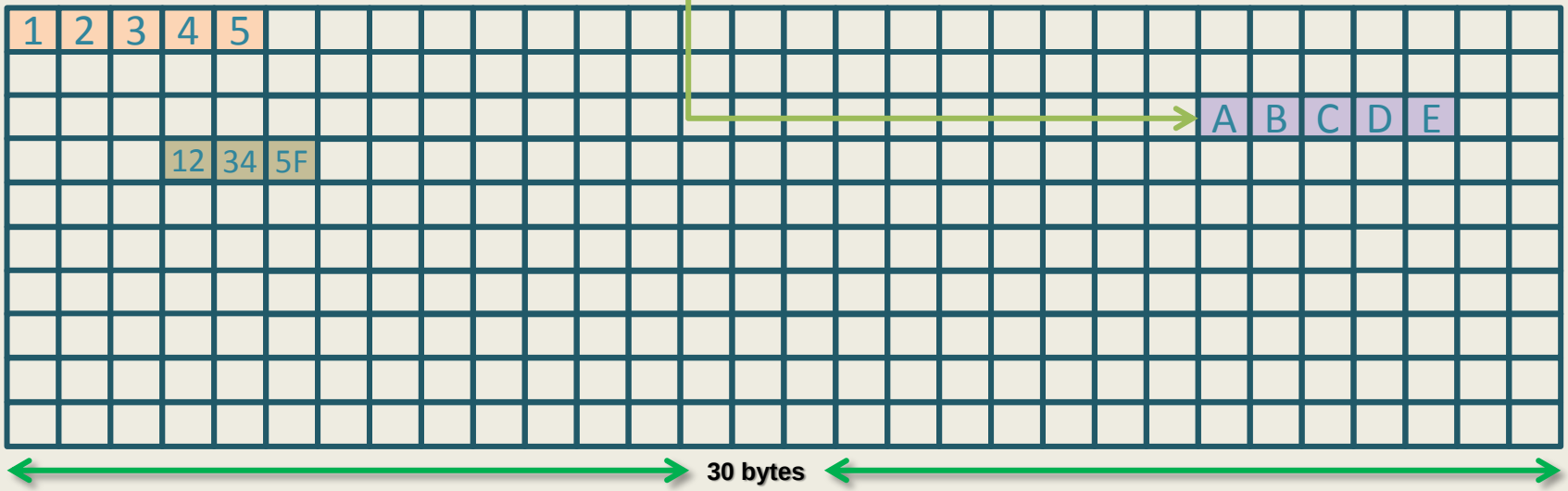
```
d SPP      s        *
d Offset   s      10U 0

d Field1   s        5      based(SPP)
d Field2   s        5p 0   based(SPP)
d Field3   s        5s 0   based(SPP)
```



Pointers

```
SPP = 1C5736254E001000
Offset = 83

SPP += Offset
```

```
d SPP      s        *
d Offset   s      10U 0

d Field1   s       5       based(SPP)
d Field2   s       5p 0    based(SPP)
d Field3   s       5s 0    based(SPP)
```

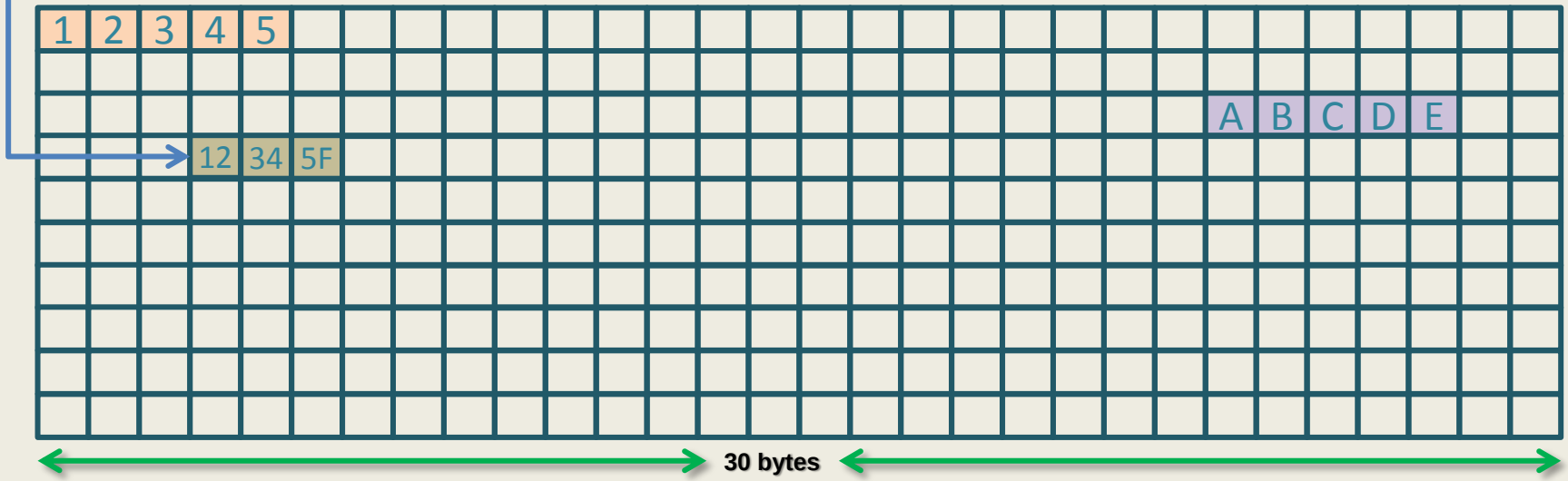


Pointers

```
SPP = 1C5736254E001000
Offset = 83
```

```
d SPP      s        *
d Offset   s      10U 0

d Field1   s       5      based(SPP)
d Field2   s       5p 0   based(SPP)
d Field3   s       5s 0   based(SPP)
```

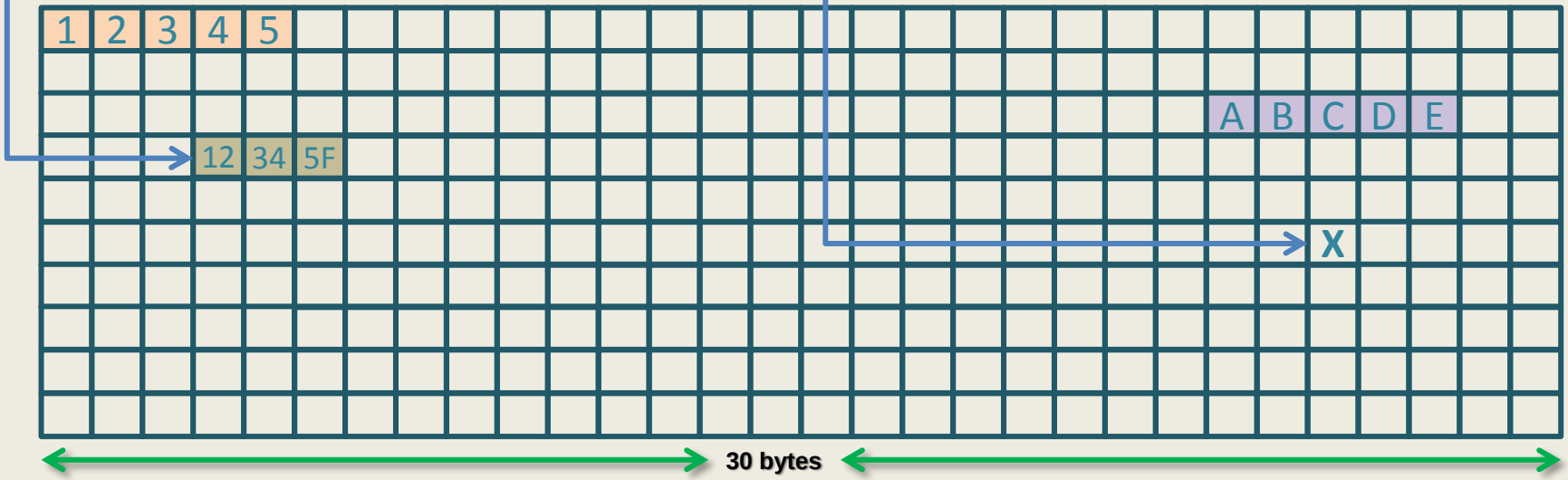
```
SPP += Offset

SPP += (Offset + 10)
```



Pointers

```
SPP = 1C5736254E001000
Offset = 83

SPP += Offset

SPP += (Offset + 10)
```

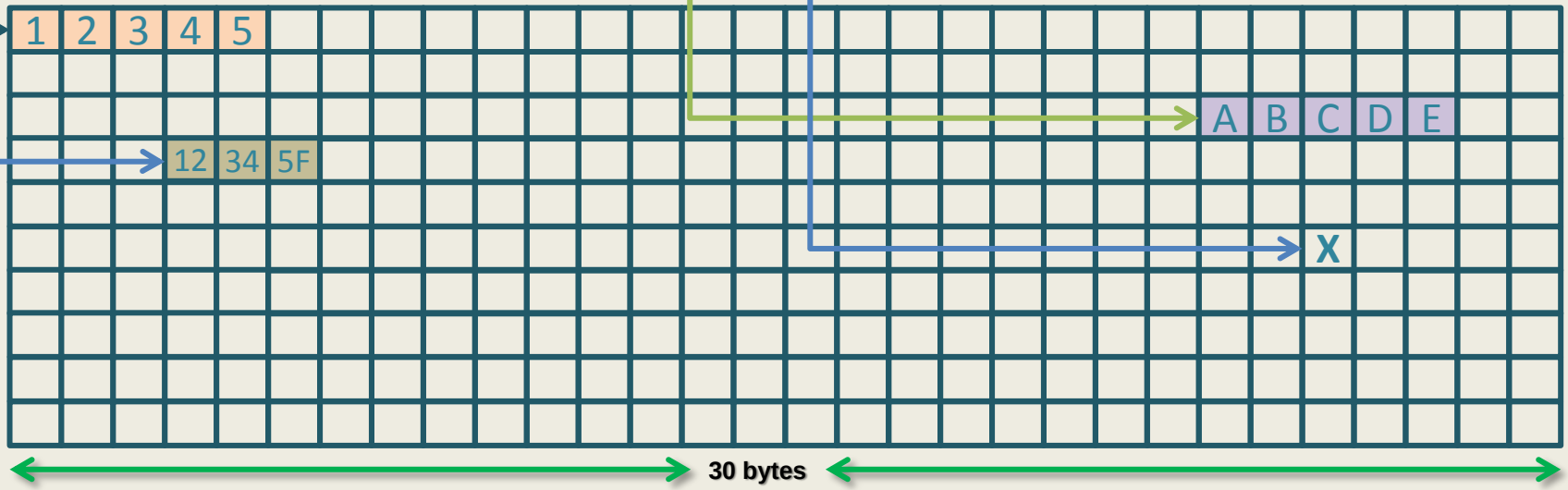
```
d SPP      s        *
d Offset   s      10U 0

d Field1   s       5       based(SPP)
d Field2   s       5p 0    based(SPP)
d Field3   s       5s 0    based(SPP)
```

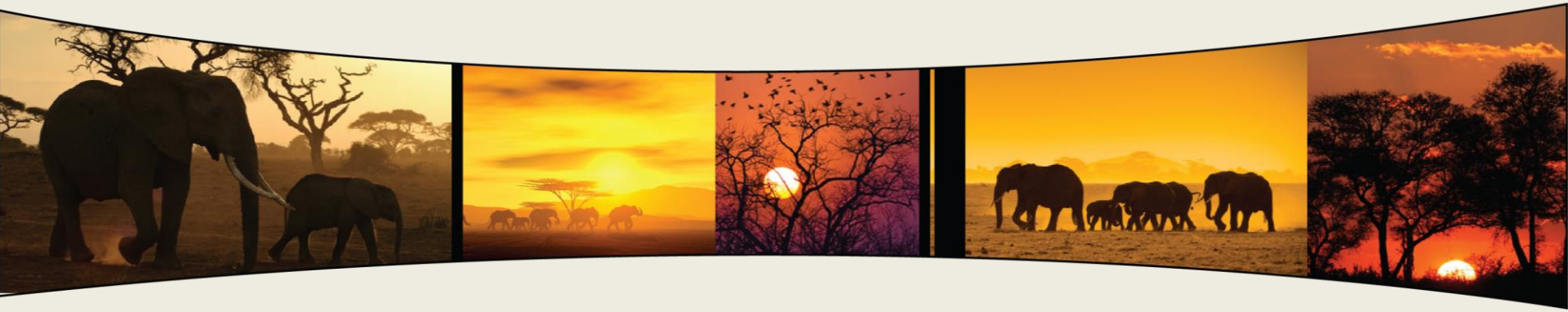


Pointers

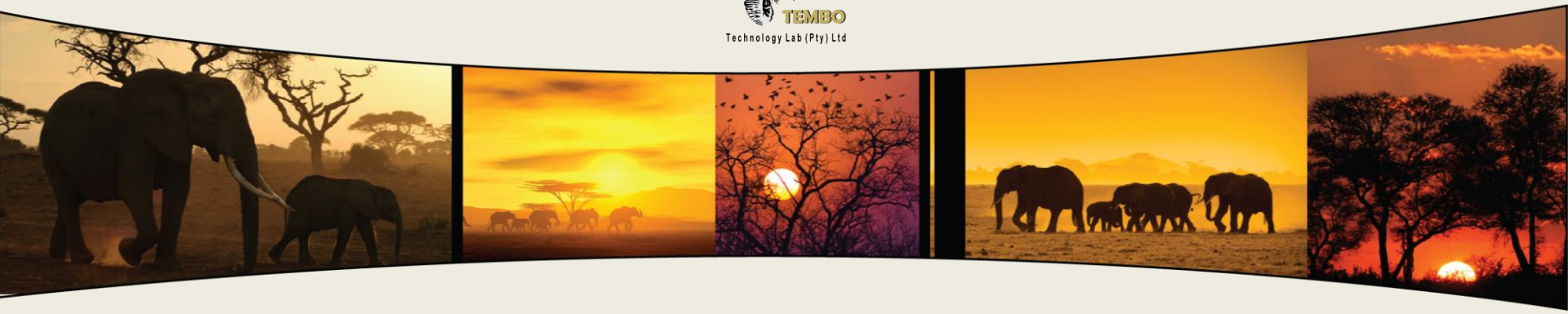
```
SPP = 1C5736254E001000
Offset = 83

SPP += Offset

SPP += (Offset + 10)
```

```
d SPP      s        *
d Offset   s      10U 0

d Field1   s       5       based(SPP)
d Field2   s       5p 0    based(SPP)
d Field3   s       5s 0    based(SPP)
```



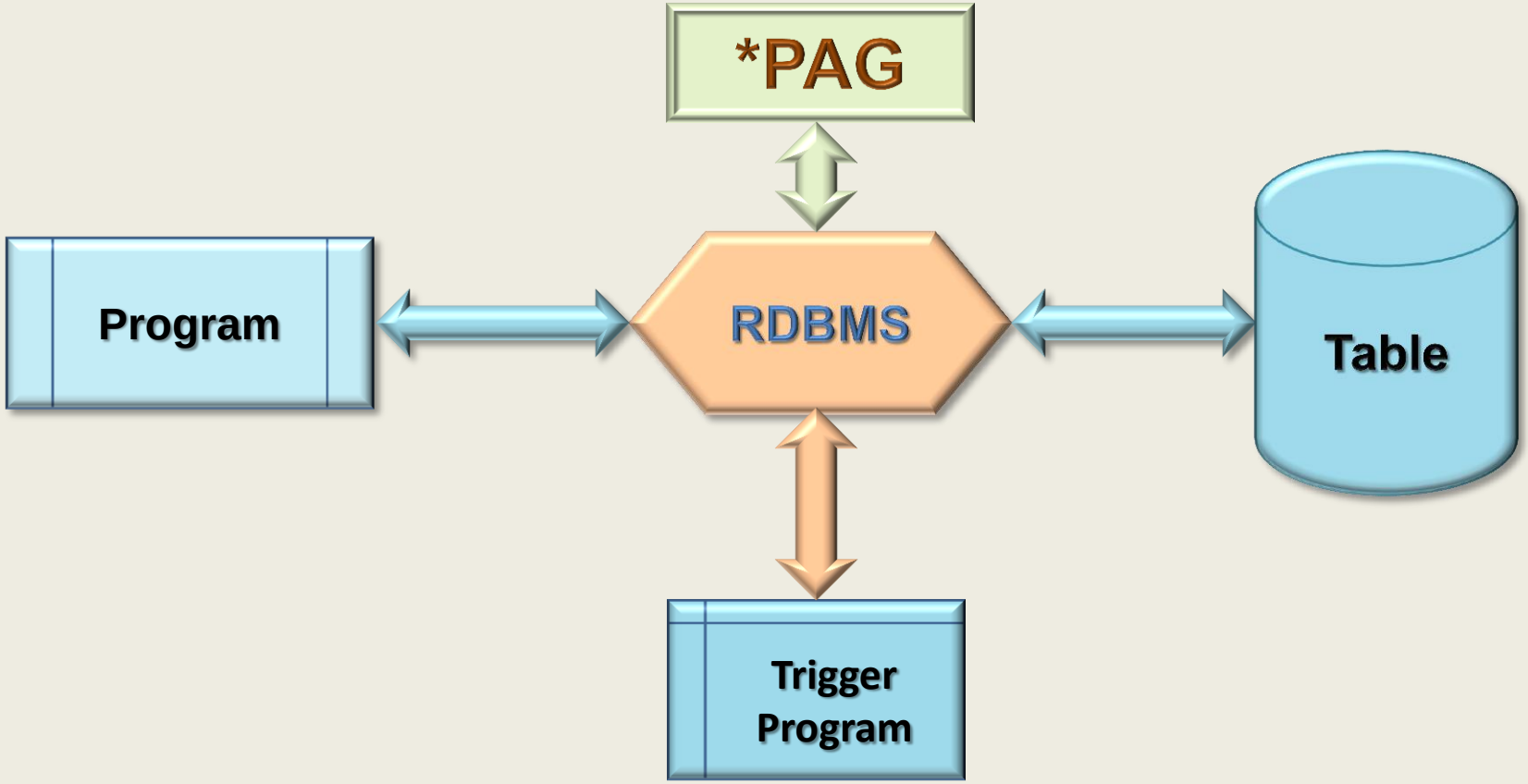


TEMBO
Technology Lab (Pty) Ltd

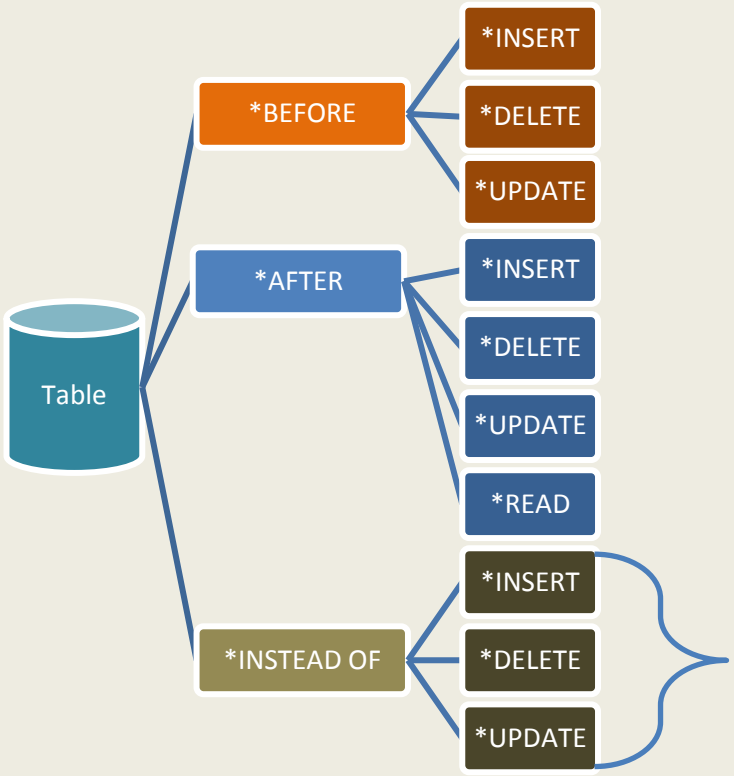


DB2 Relational Database Trigger Programs

Trigger Programs



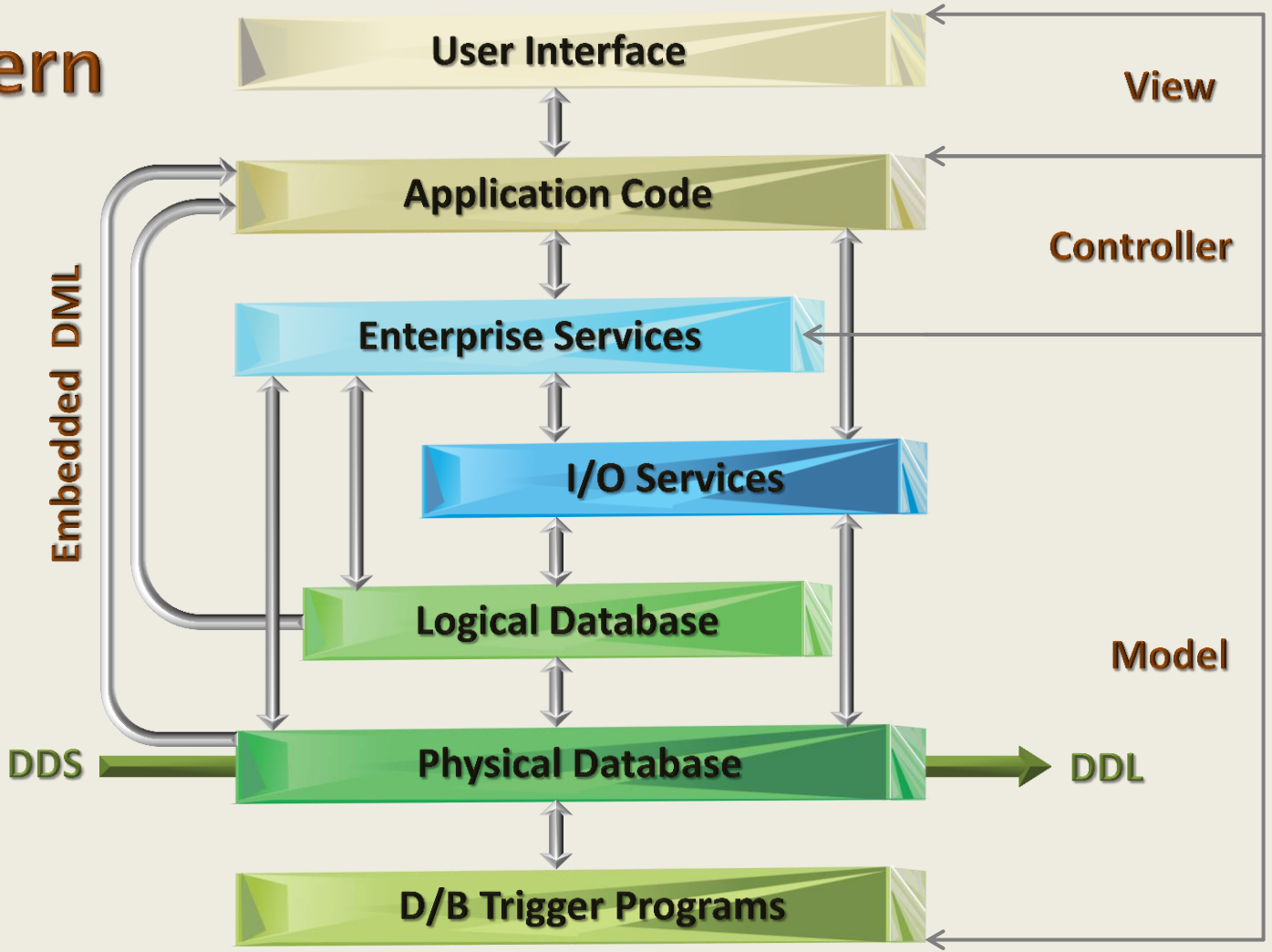
Trigger Events



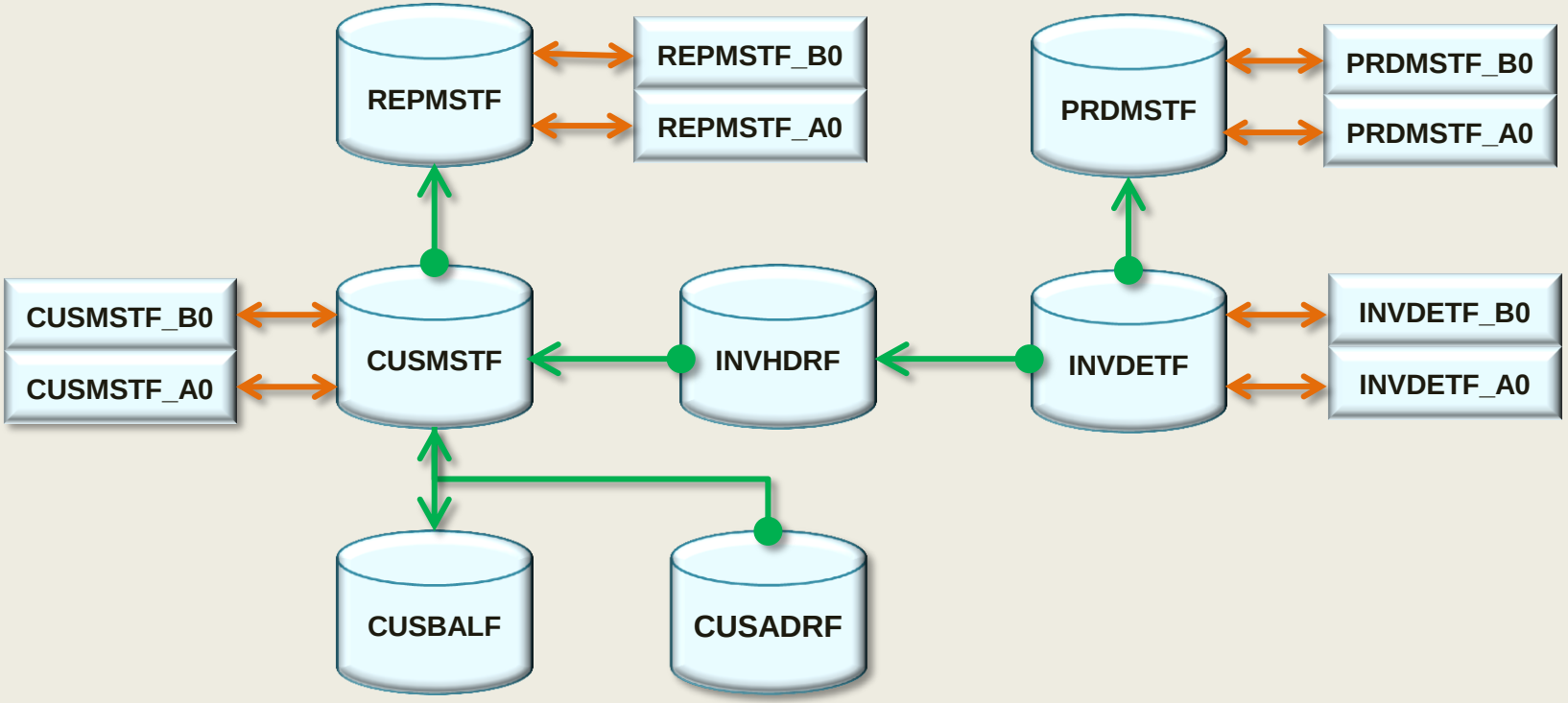
A Maximum of 300 Trigger Programs are allowed

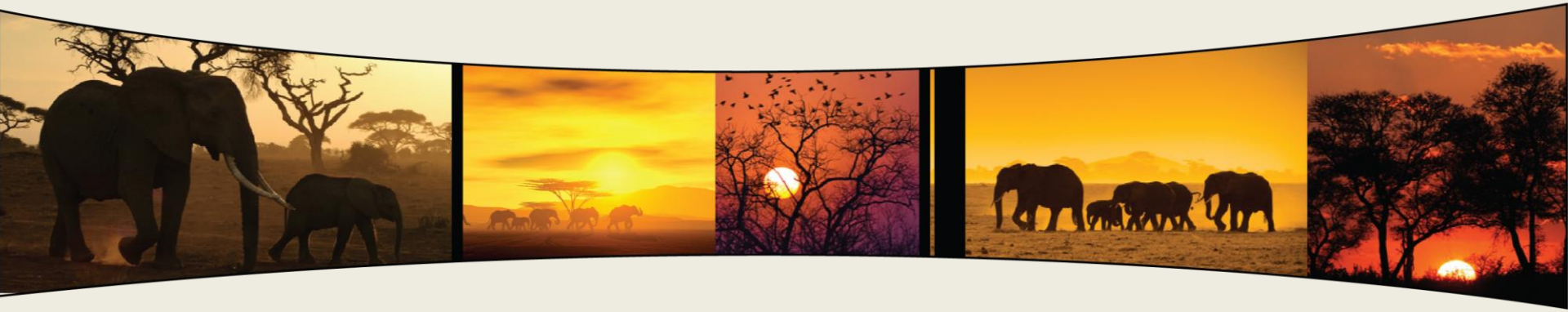
SQL Logicals Only

Modern


User Interface
View
Application Code
Controller
Embedded DML
Enterprise Services
I/O Services
Logical Database
Model
DDS
Physical Database
DDL
D/B Trigger Programs

Constraints + Triggers





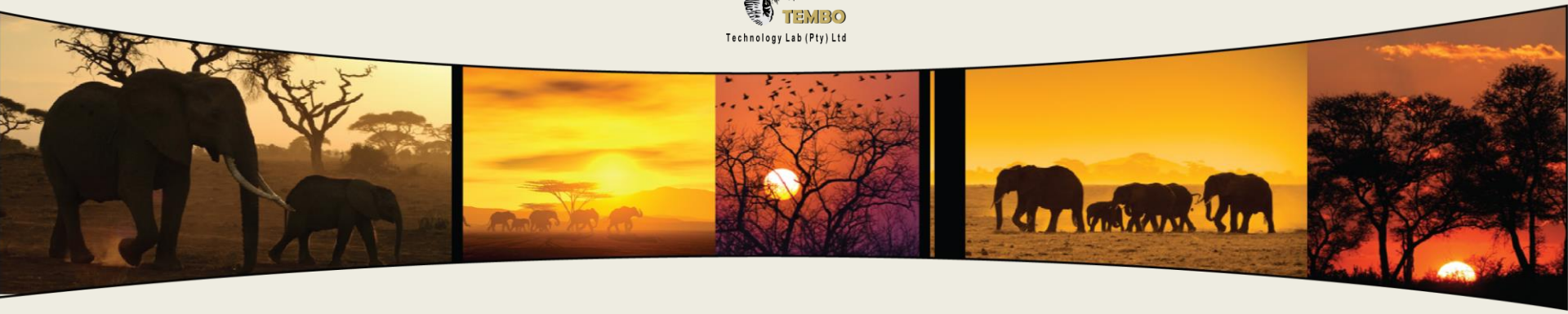
Kgalagadi Transfrontier Park

Gemsbok

Kalahari Lion

Cell: +27 83 678 9883
USA Office: +1-651-348-2468
Fax: +27 86 558 3626
Email: tommya@tembotechlab.com
Skype: tommya.atkins
Professional Resume:
http://www.linkedin.com/in/tommyatkins
RECLAIM YOUR HERITAGE!!! : www.adsero-optima.com

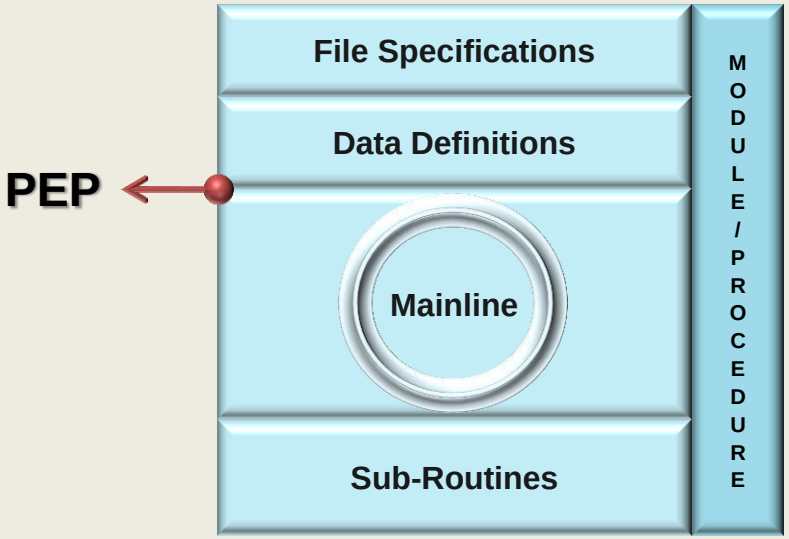
TEMBO
Technology Lab (Pty) Ltd



ILE Program Construction

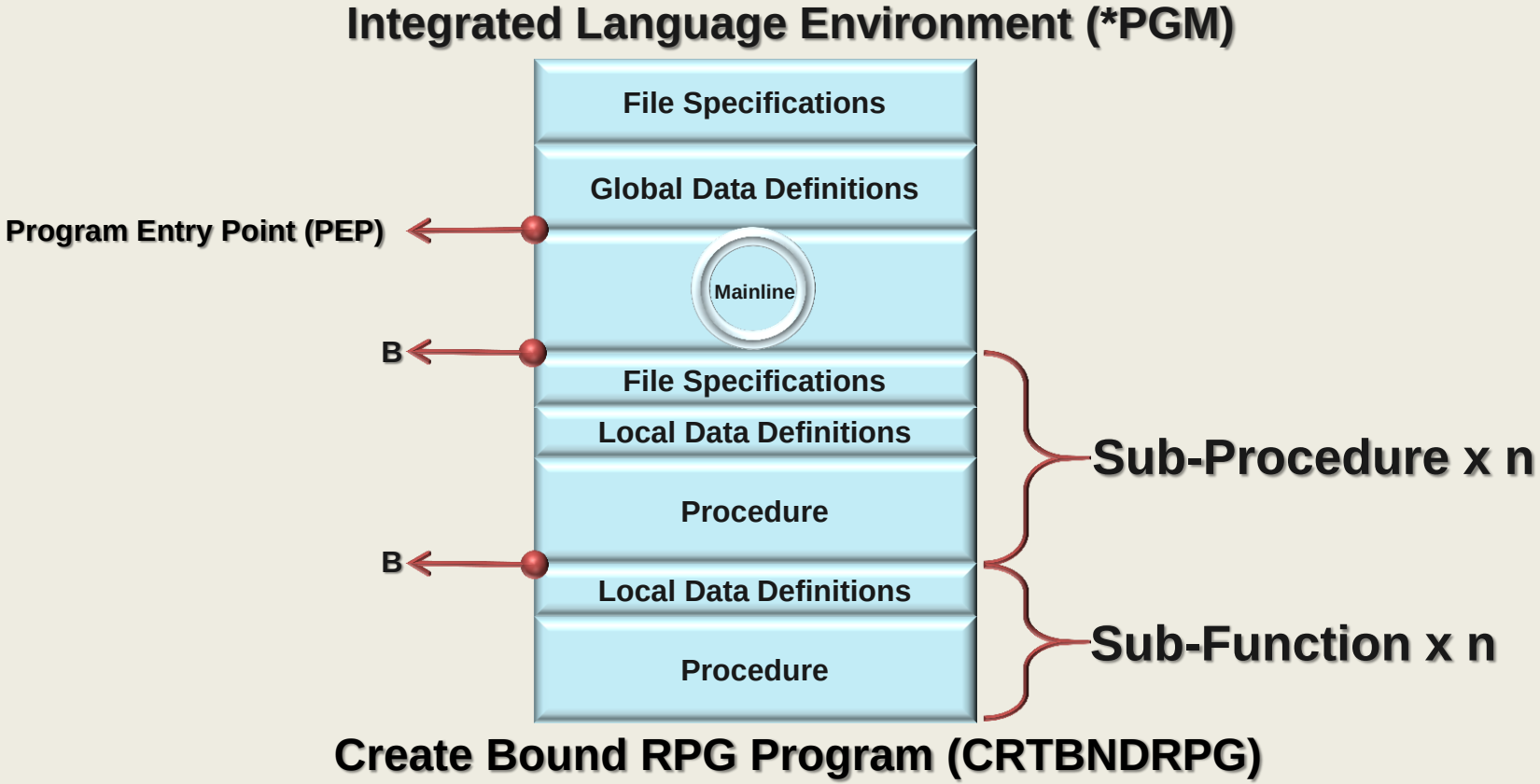
OPM Programs

Original Program Model (*PGM)

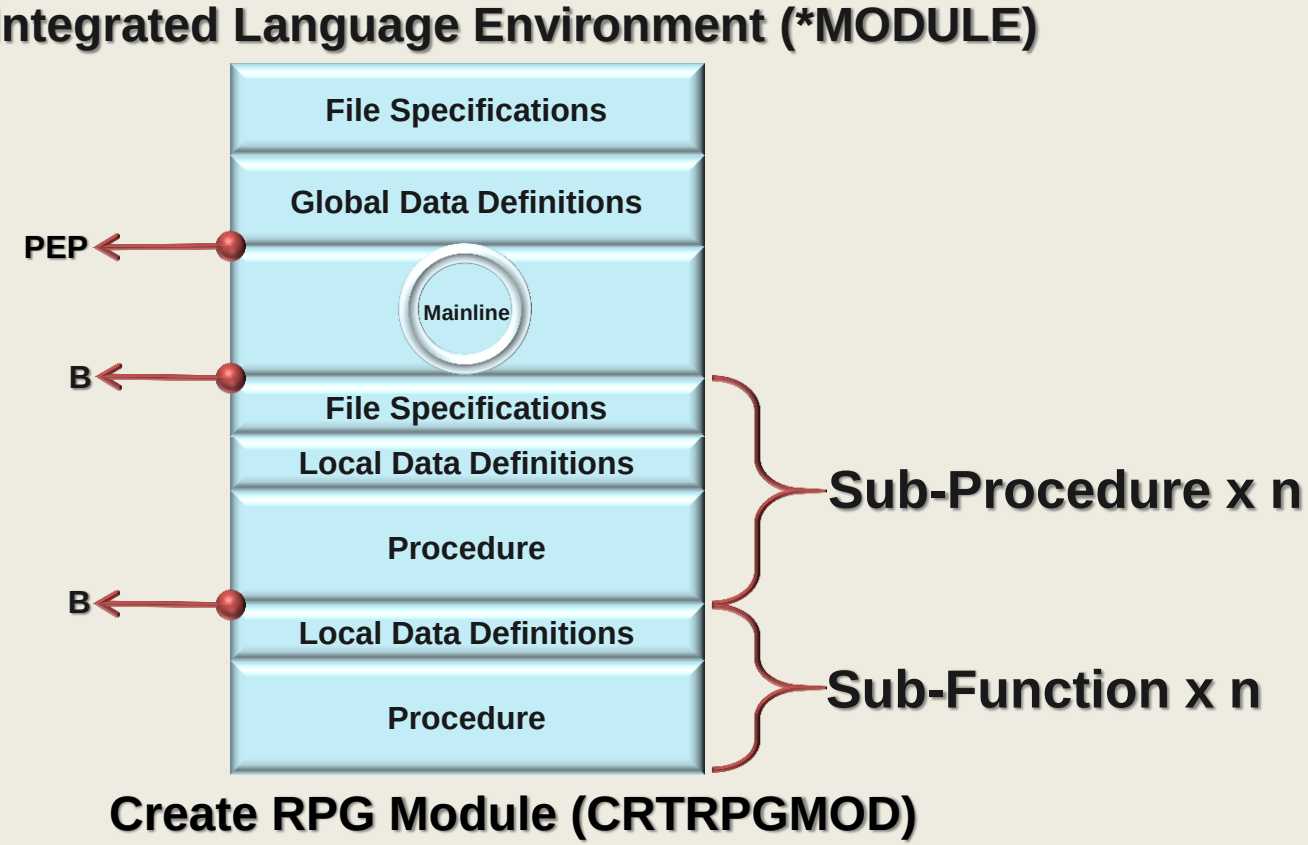


Create RPG/400 Program (CRTRPGPGM)

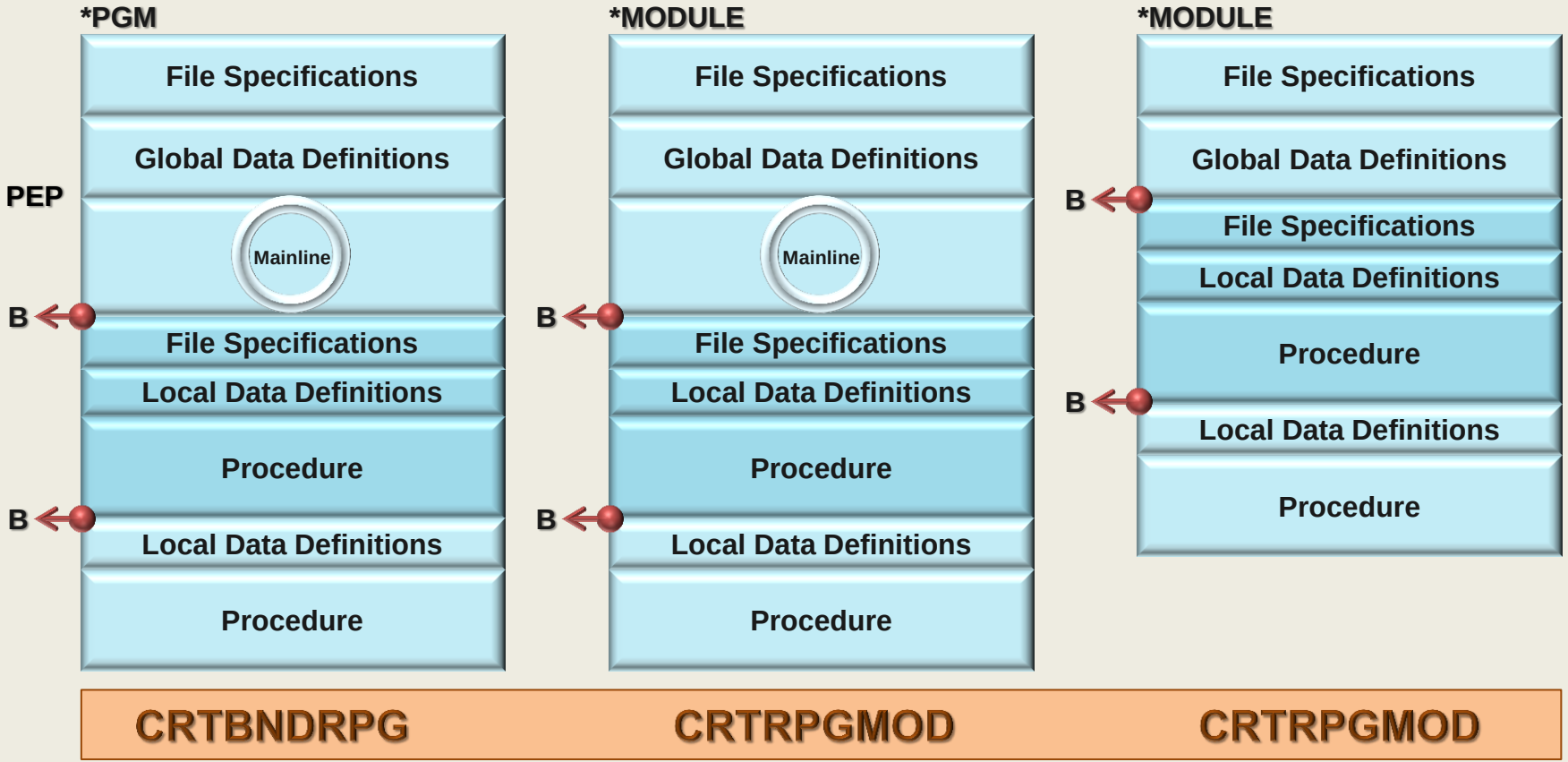
ILE Programs



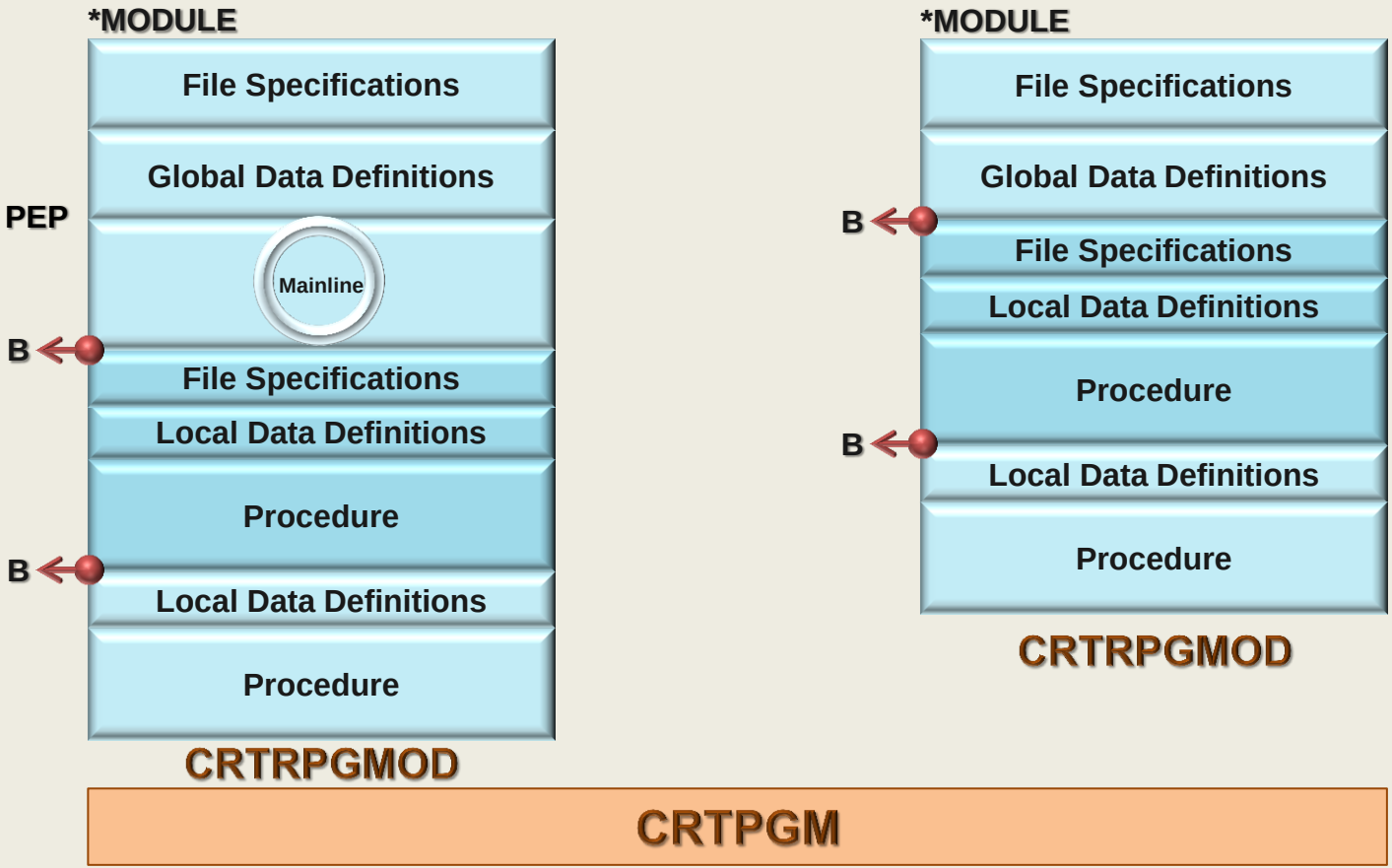
ILE Programs



Programs from *MODULES

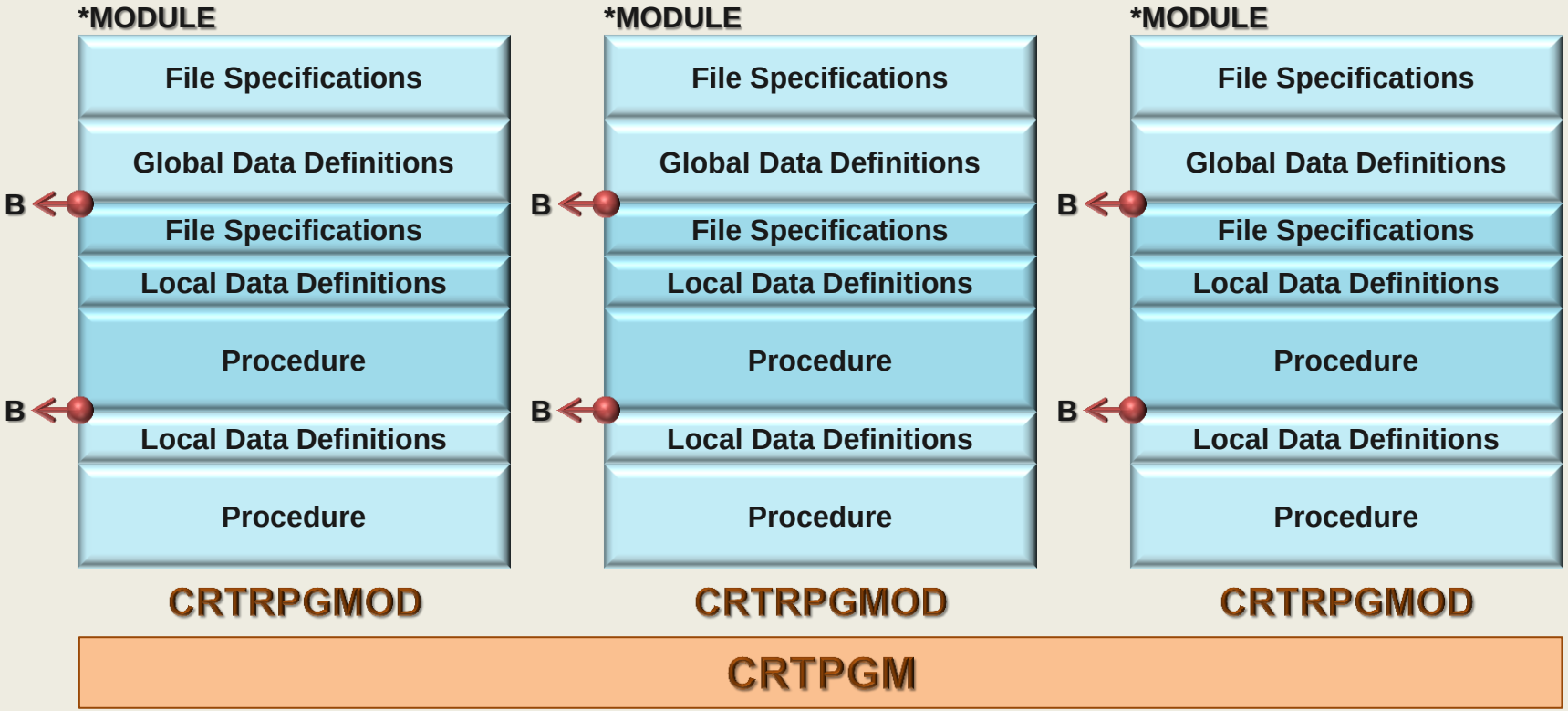


Programs from *MODULES

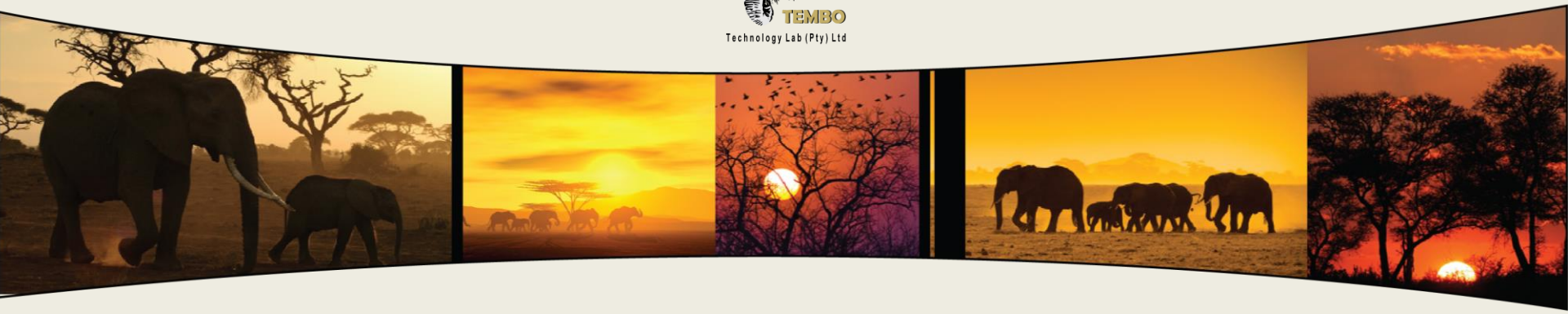


Programs from *MODULES

MAIN(procedure_name) or NOMAIN



TEMBO
Technology Lab (Pty) Ltd

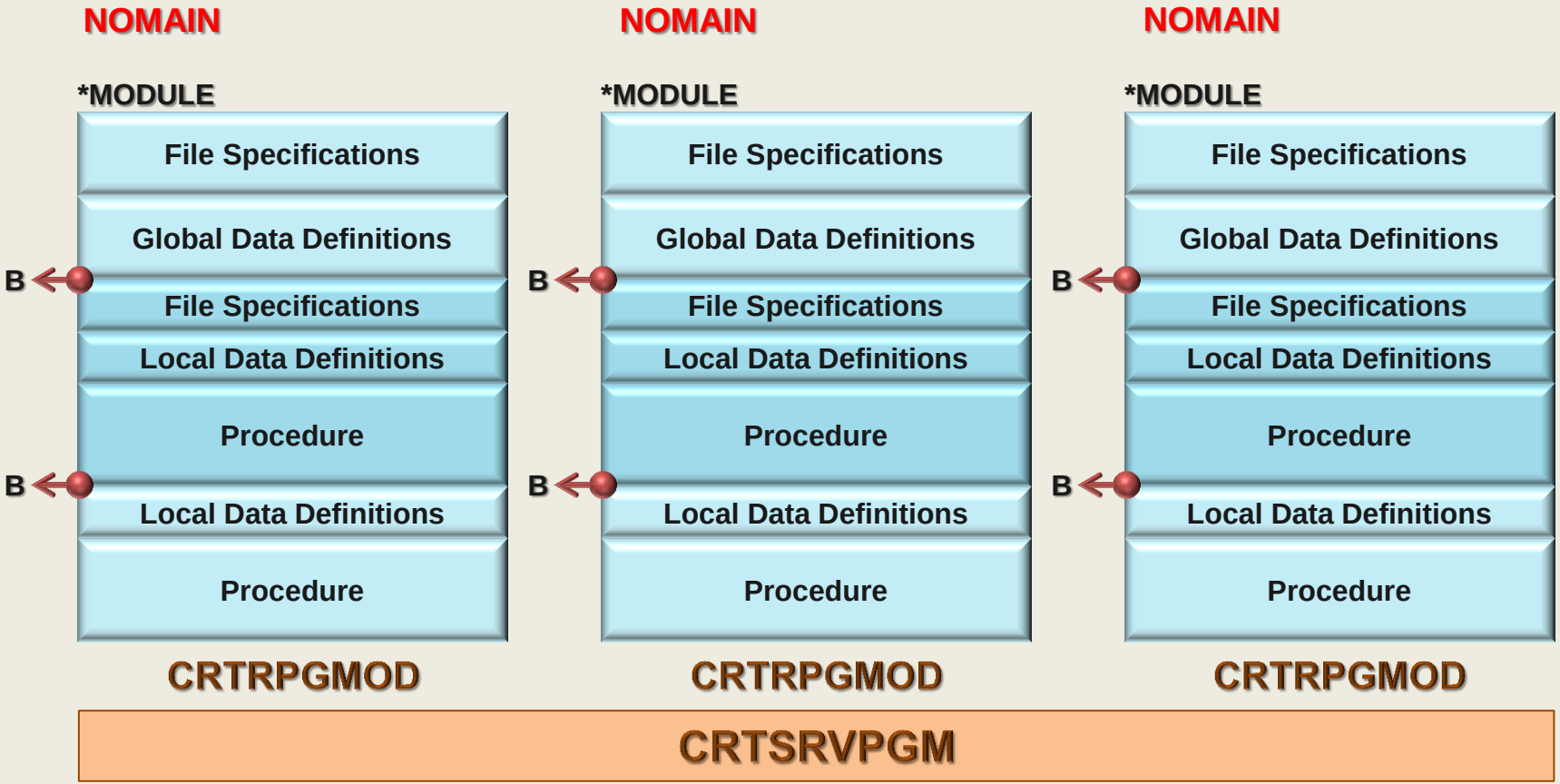


Service Programs

Service Programs

A service program (*SRVPGM) is a method of binding *MODULES and their included procedures together into a single object, within which selected procedures are individually callable from other application code, including other service program procedures.

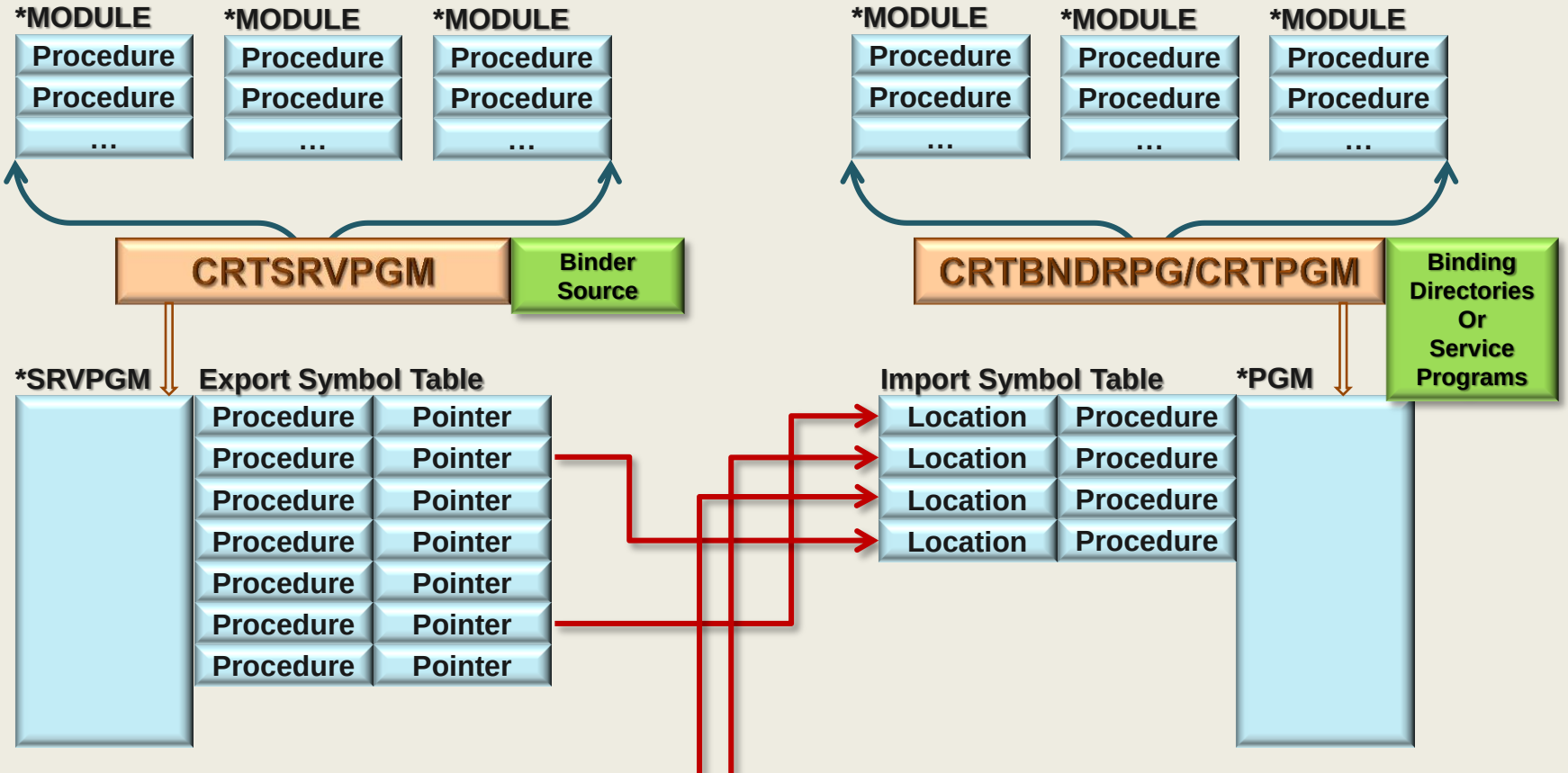
Service Programs from *MODULES



*SRVPGM's: Do's & Don'ts

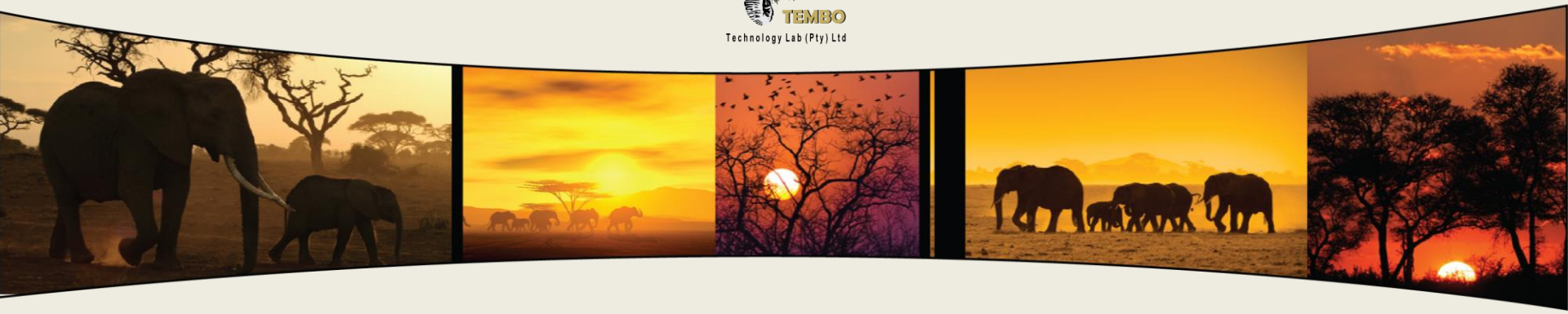
- **Always use Binder Source when binding service programs.**
  - **Controls Signature**
  - **Controls Export Symbol Table**
- **Write A CLLE compiler.**
  - **Eliminates Compilation Mistakes**
- **Don't use Binding Directories**
  - **List the Additional Service Programs required**

Binding to Service Programs



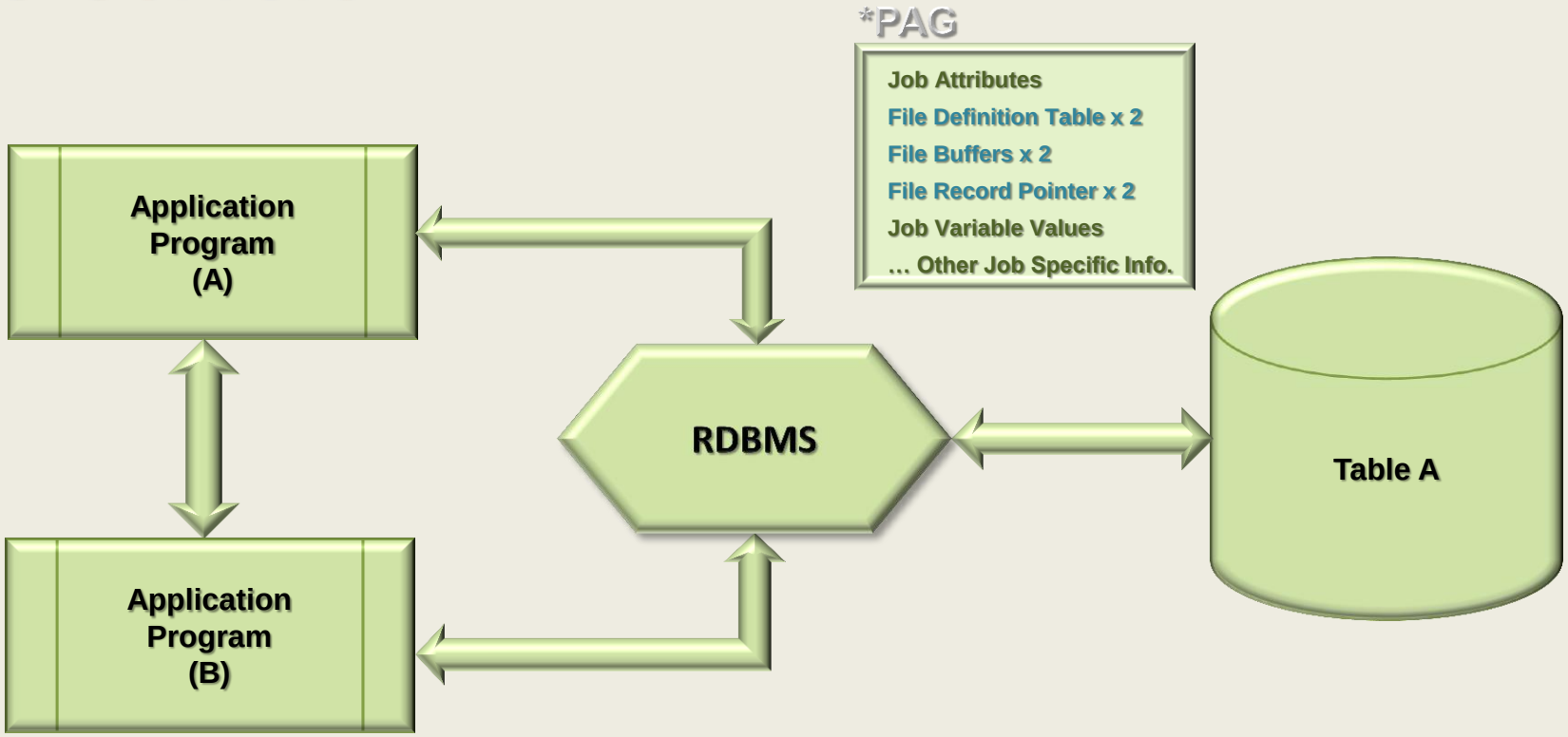

No…you go down
and talk to them!

TEMBO
Technology Lab (Pty) Ltd

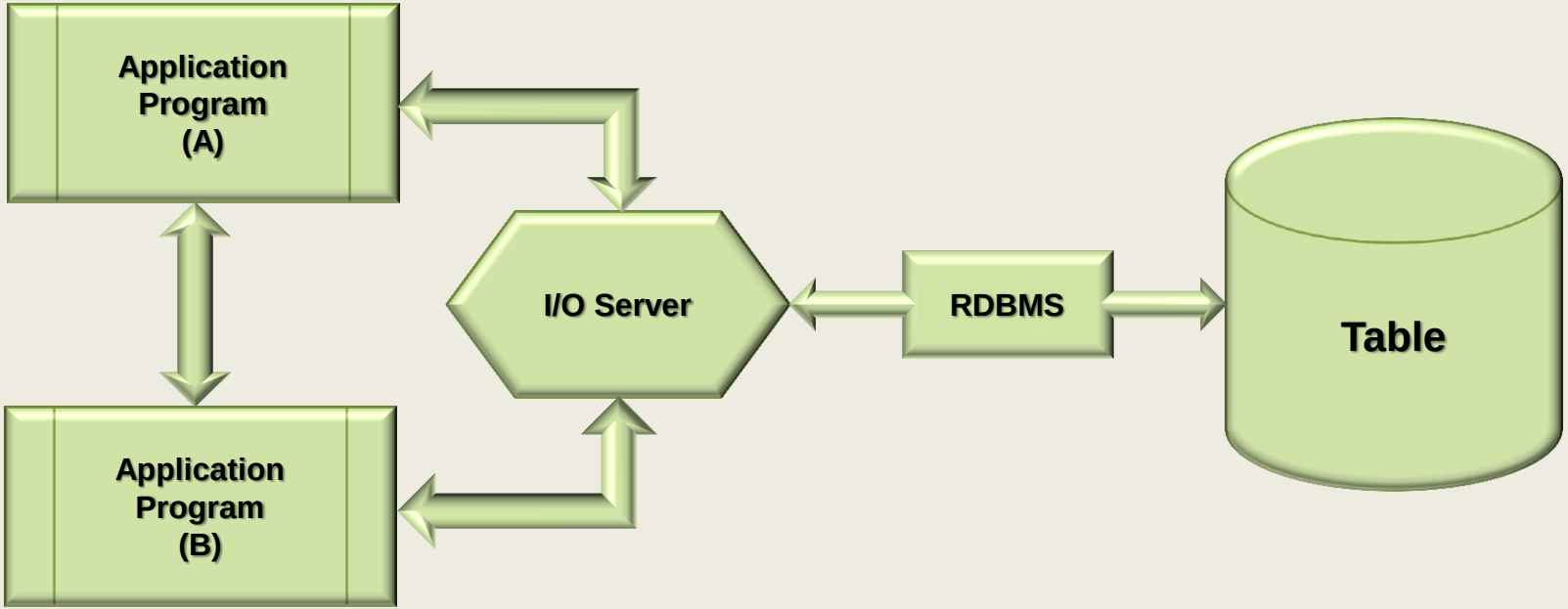


DB2 Relational Database
I/O Servers

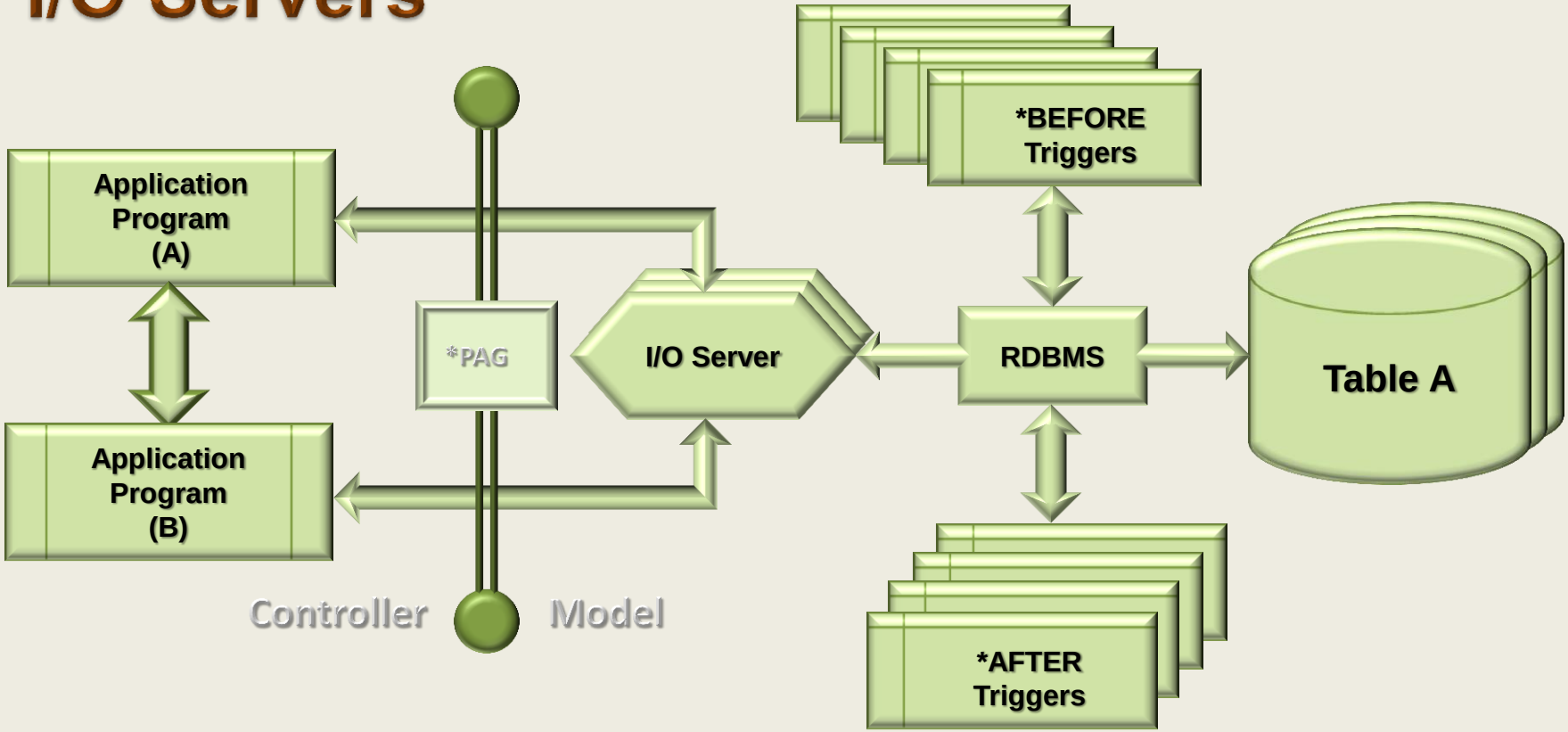
I/O Servers

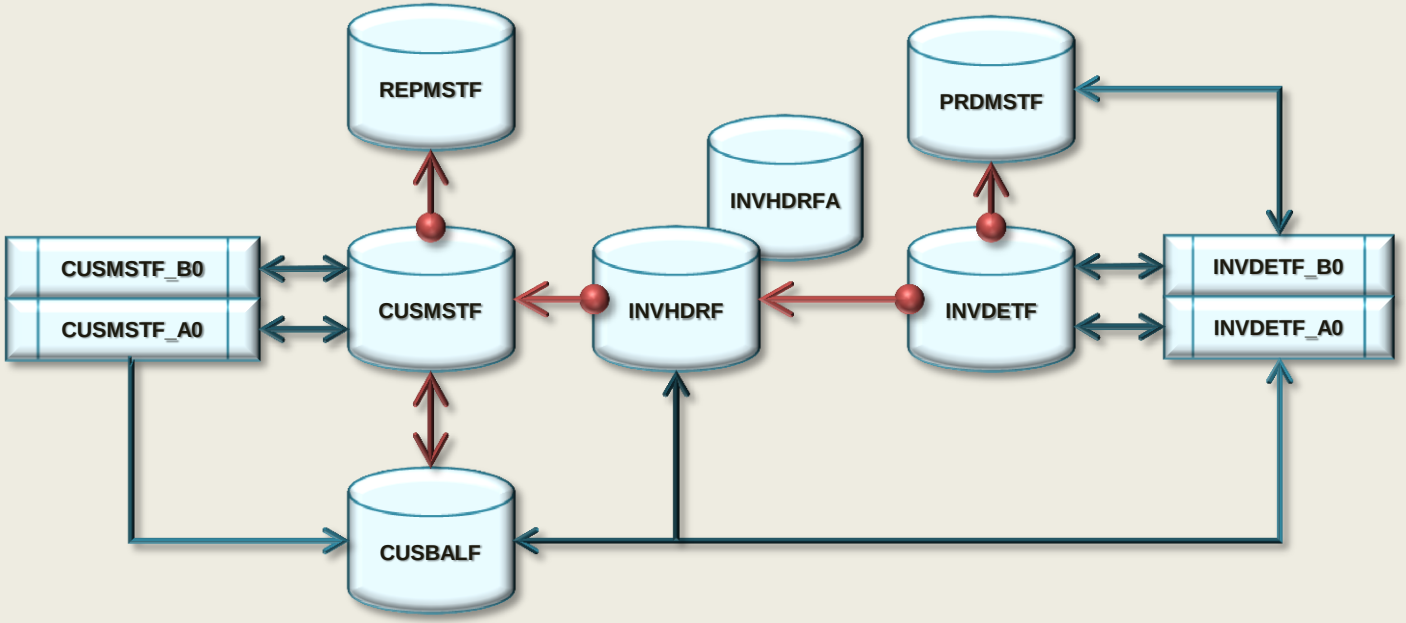

*PAG
Job Attributes
File Definition Table x 2
File Buffers x 2
File Record Pointer x 2
Job Variable Values
... Other Job Specific Info.
Application Program (A)
Application Program (B)
RDBMS
Table A

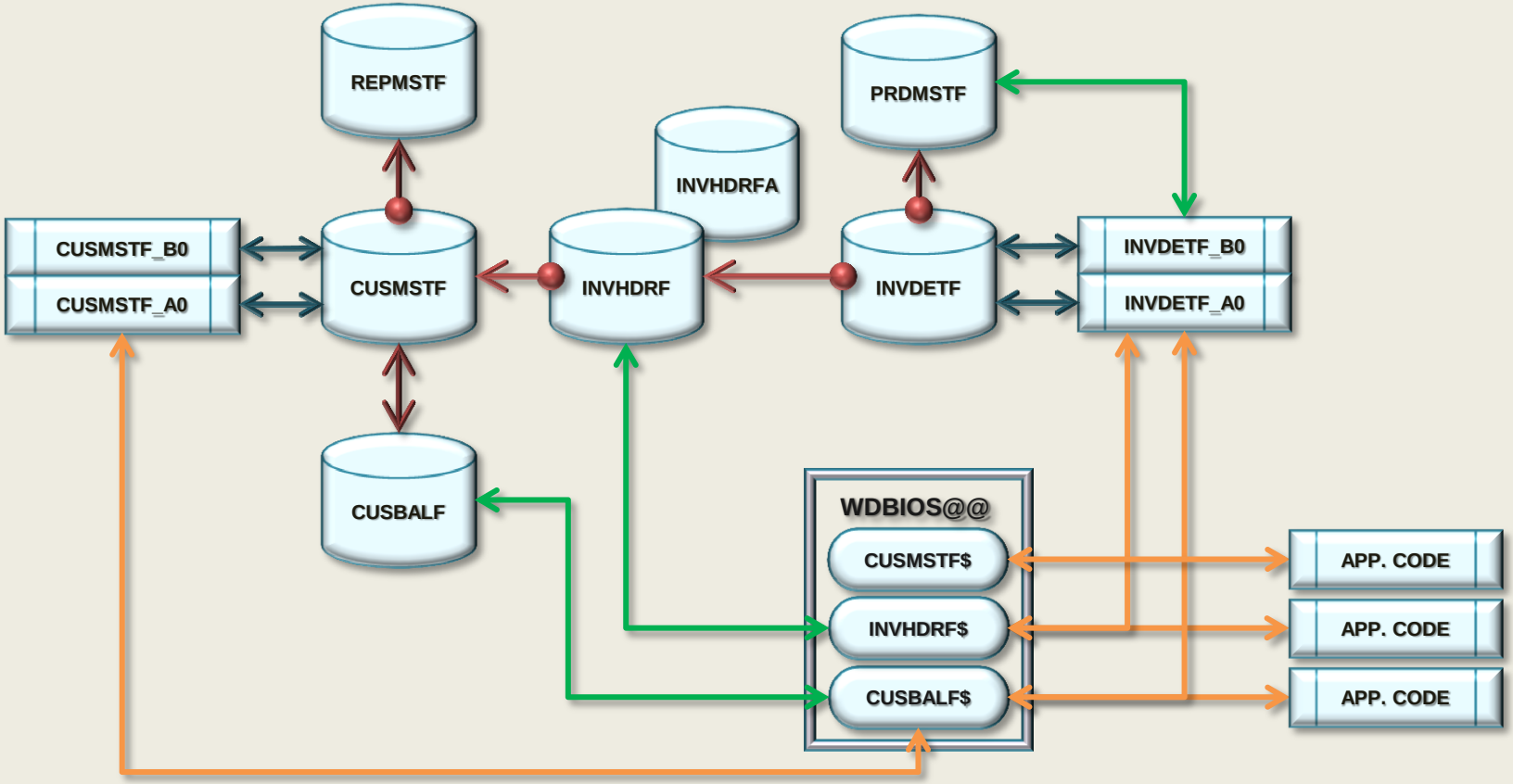
I/O Servers


Application Program (A)
Application Program (B)
I/O Server
RDBMS
Table

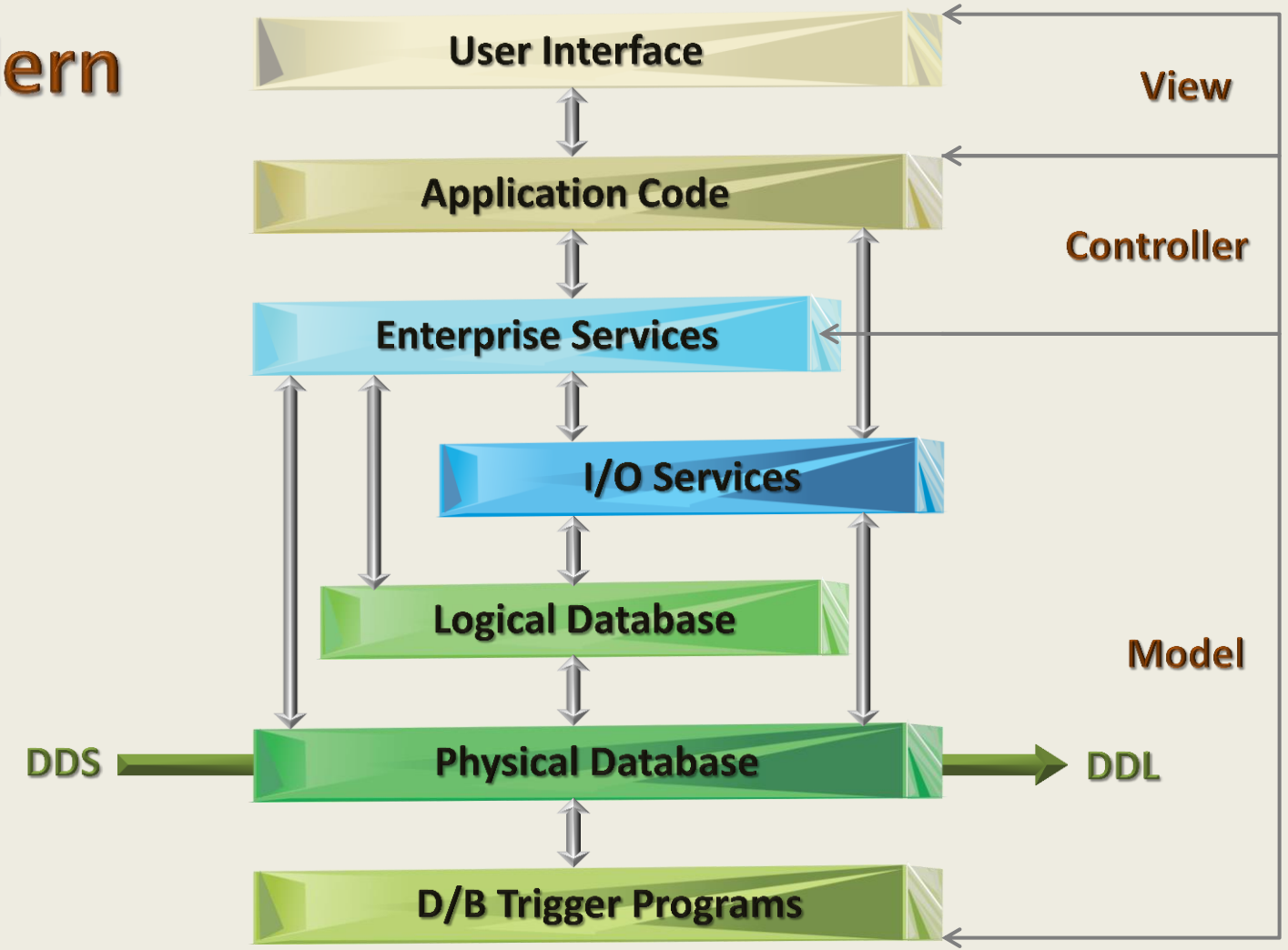
I/O Servers


Application Program (A)
Application Program (B)
*PAG
I/O Server
*BEFORE Triggers
RDBMS
Table A
*AFTER Triggers
Controller
Model

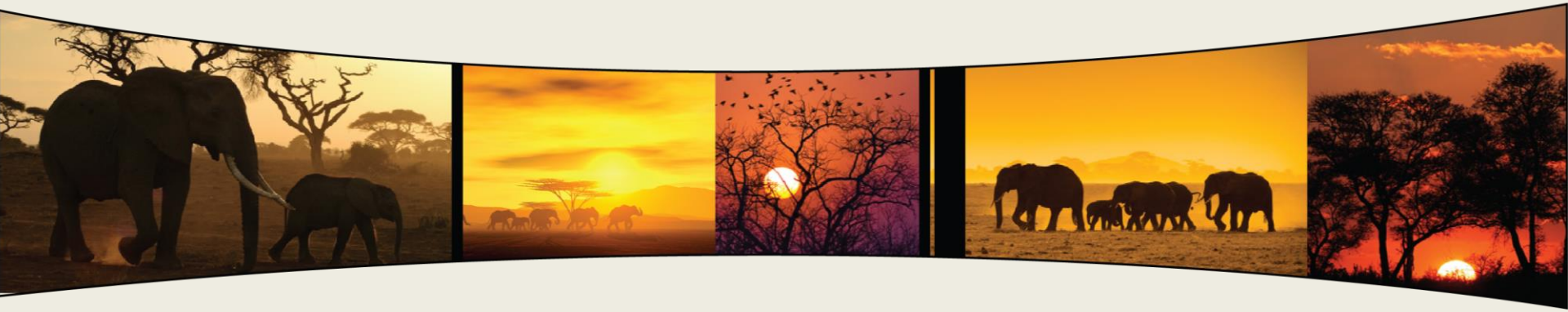

REPMSTF
PRDMSTF
INVHDRFA
CUSMSTF_B0
CUSMSTF_A0
CUSMSTF
INVHDRF
INVDETF
INVDETF_B0
INVDETF_A0
CUSBALF


REPMSTF
PRDMSTF
INVHDRFA
CUSMSTF_B0
CUSMSTF_A0
CUSMSTF
INVHDRF
INVDETF
INVDETF_B0
INVDETF_A0
CUSBALF
WDBIOS@@
CUSMSTF$
INVHDRF$
CUSBALF$
APP. CODE
APP. CODE
APP. CODE

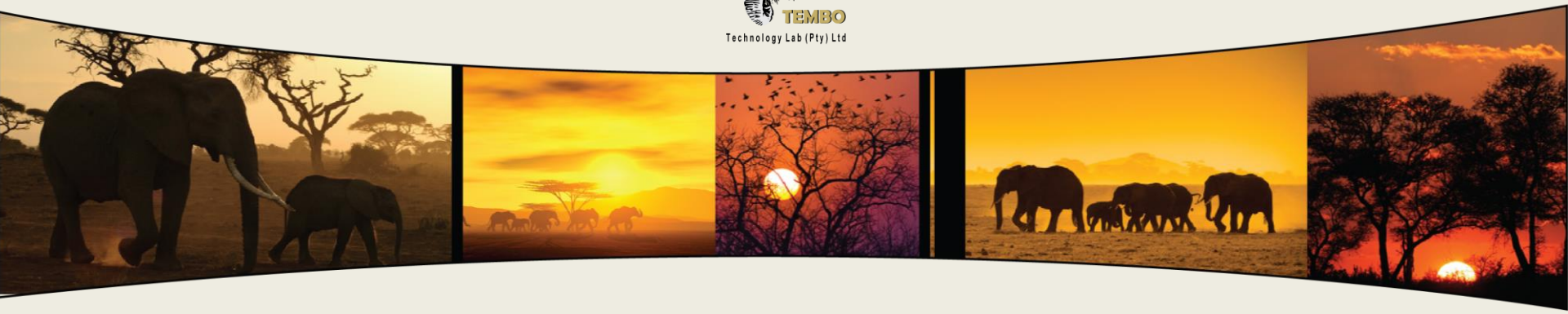
Modern


User Interface
View
Application Code
Controller
Enterprise Services
I/O Services
Logical Database
Model
DDS
Physical Database
DDL
D/B Trigger Programs


© Ursula Celliers - Wildlife Photography
Nyala



TEMBO
Technology Lab (Pty) Ltd



DB2 Relational Database
Viewing the “Data Store”

Physical Database

DDS Physical Files

DDL Tables

Logical Database

DDS View
DDS Joins
DDS Multi-Format

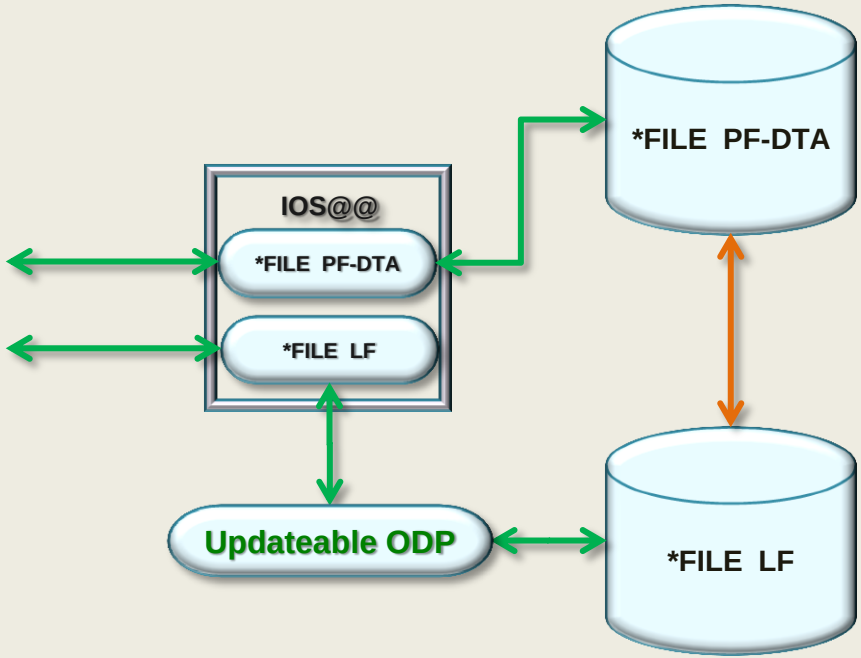
DDL EV Index
DDL BR Index
DDL View

Logical Database

DDS View
DDS Joins
DDS Multi-Format

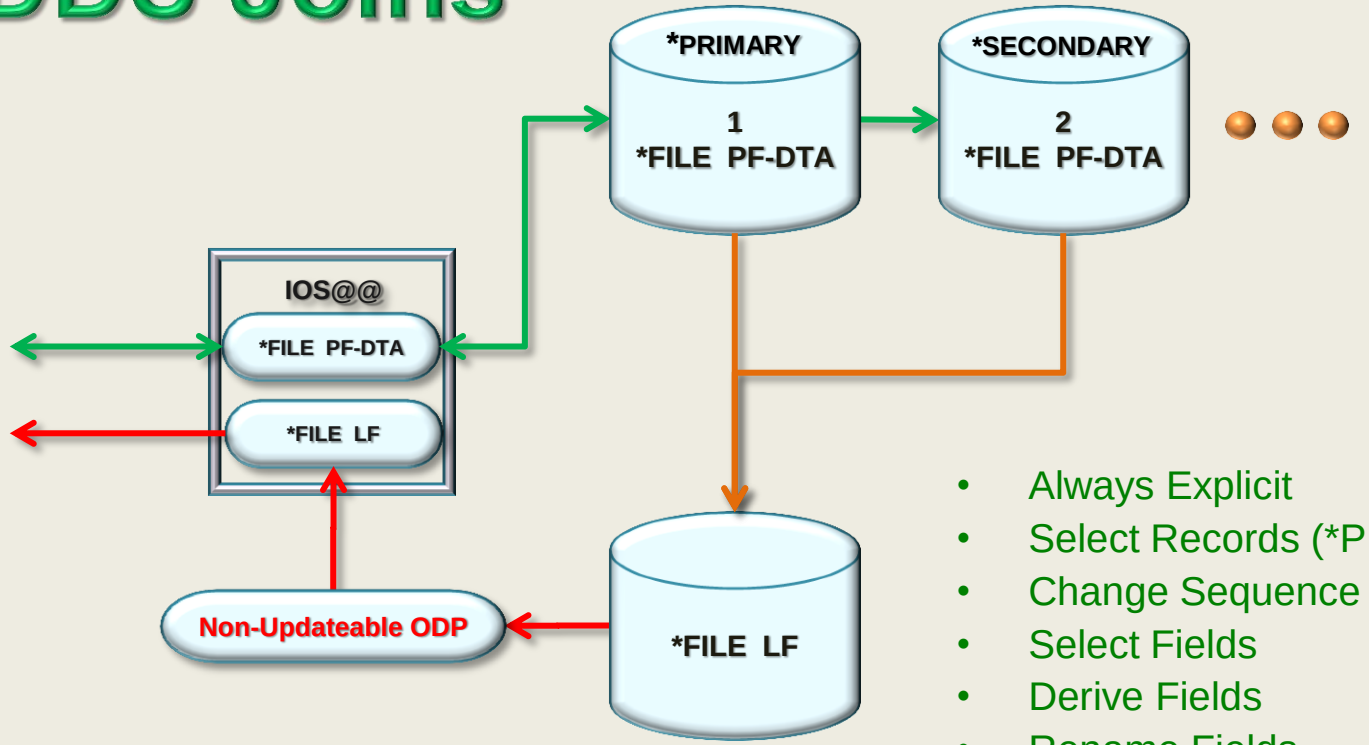
DDL EV Index
DDL BR Index
DDL View

DDS View



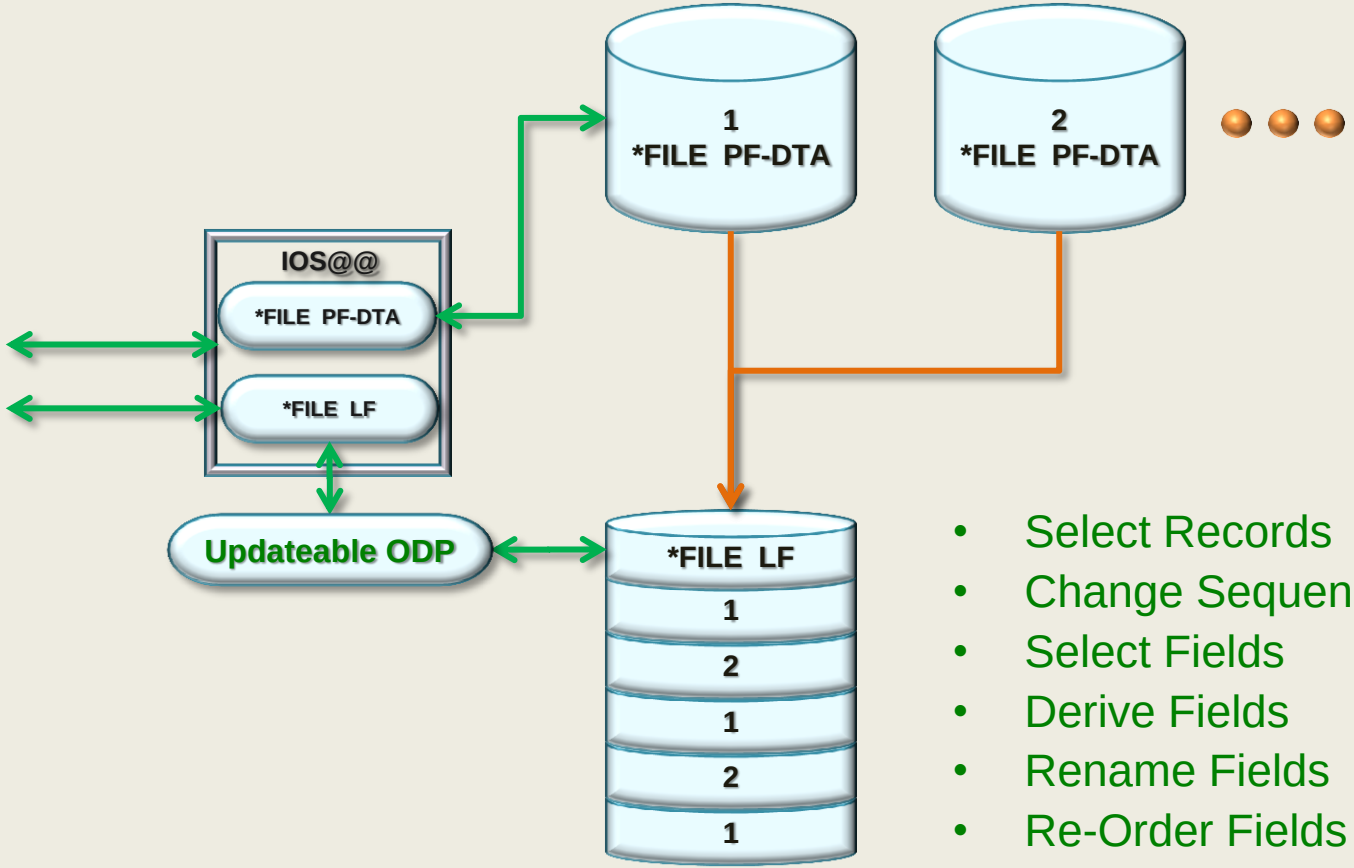
- Select Records
- Change Sequence
- Select Fields (Explicit)
- Derive Fields (Explicit)
- Rename Fields (Explicit)
- Re-Order Fields (Explicit)

DDS Joins



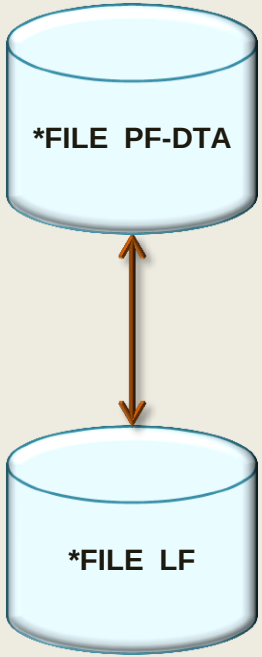
- Always Explicit
- Select Records (*PRIMARY)
- Change Sequence (*PRIMARY)
- Select Fields
- Derive Fields
- Rename Fields
- Re-Order Fields

DDS Multi-Format



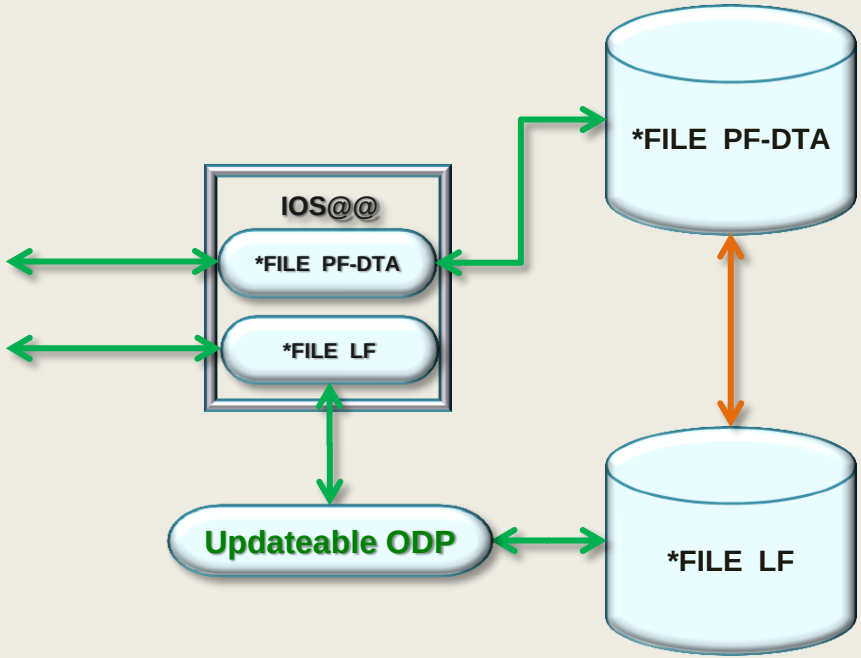
- Select Records
- Change Sequence
- Select Fields
- Derive Fields
- Rename Fields
- Re-Order Fields

DDL Encoded Vector Index (EV)



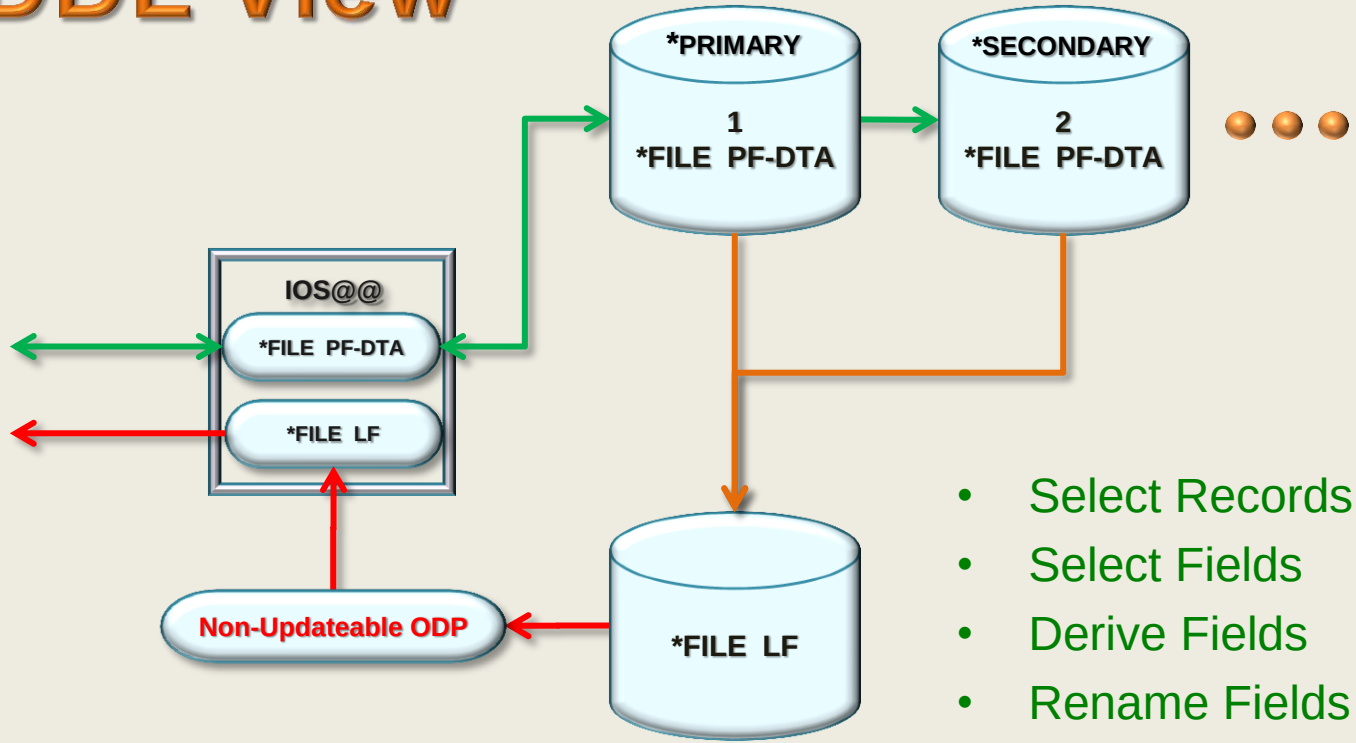
- Select Records
- Change Sequence
- Select Fields (Explicit)
- Derive Fields (Explicit)
- Rename Fields (Explicit)
- Re-Order Fields (Explicit)

DDL Binary Radix Index (BR)

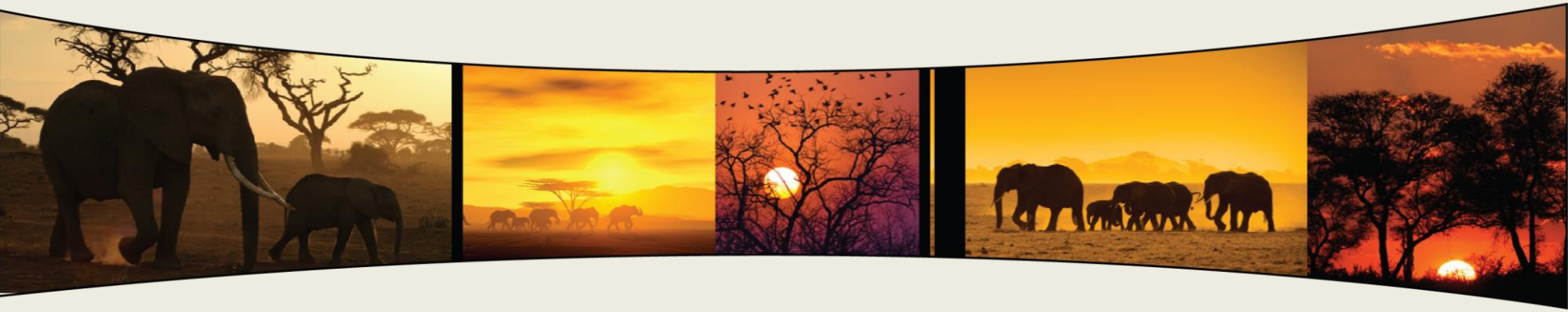


- Select Records
- Change Sequence
- Select Fields (Explicit)
- Derive Fields (Explicit)
- Rename Fields (Explicit)
- Re-Order Fields (Explicit)

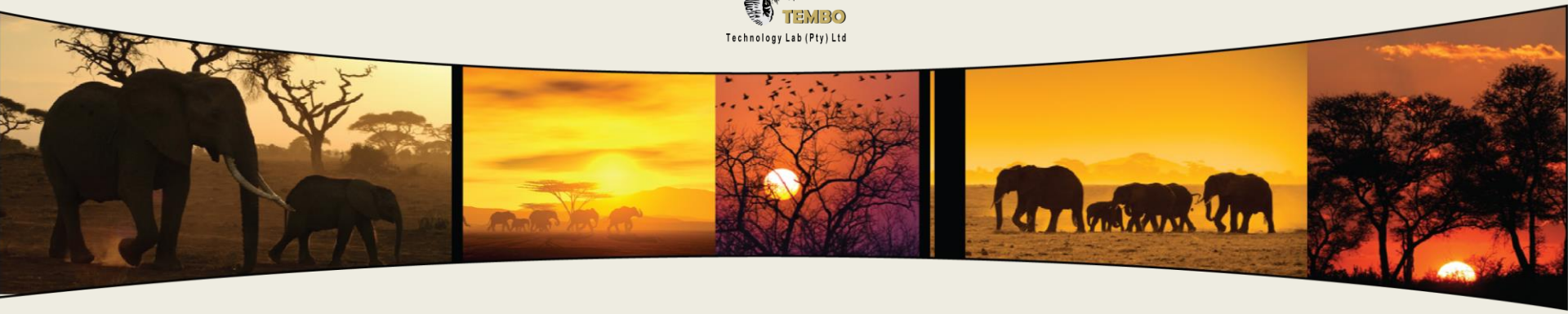
DDL View



- Select Records
- Select Fields
- Derive Fields
- Rename Fields
- Re-Order Fields



TEMBO
Technology Lab (Pty) Ltd



RPG IV for Integrated Language Environment

Integrated Language Environment

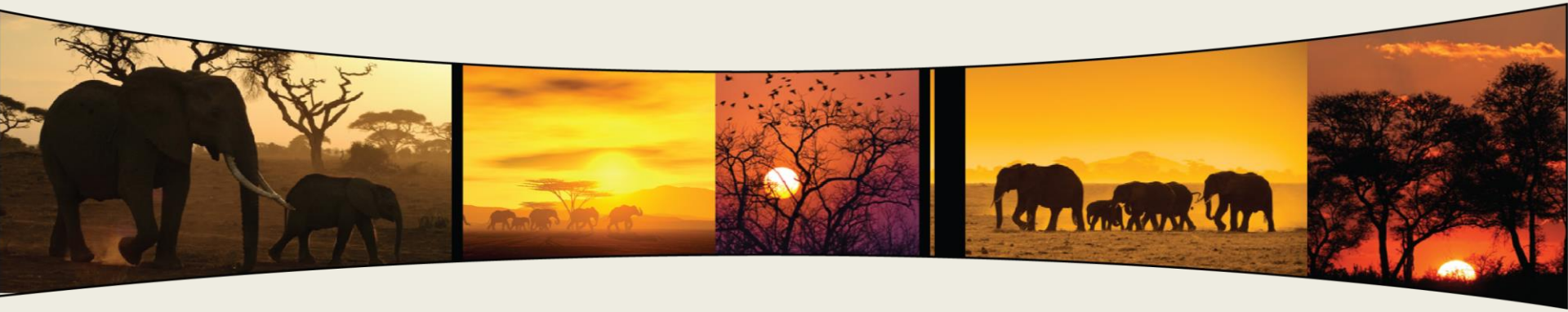
- **RPG IV**
- **Partially Free**
- **Completely Free Form**

Integrated Language Environment

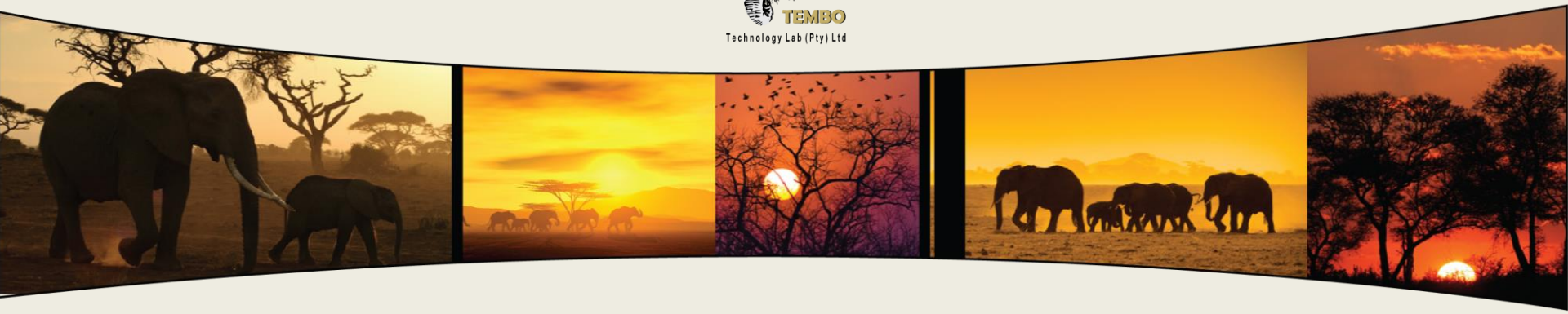
- **ILE Components**
  - **Modules**
  - **Procedures**
  - **Functions**

ILE Compilers

- Create RPG Module (CRTRPGMOD)
  - Create non-executable *MODULE's
- Create Program (CRTPGM)
  - Bind *MODULE's to create a *PGM executable
- Create Bound RPG Program (CRTBNDRPG)
  - Bind *MODULE's to create a *PGM executable
- Create Service Program (CRTSRVPGM)
  - Bind *MODULE's to create a *SRVPGM containing callable procedures



TEMBO
Technology Lab (Pty) Ltd



RPG IV for Integrated Language Environment

Activation Groups

ILE in a Box!

- Activation Groups
  - What
  - Types
  - Why
    - Resource Management
    - Commit Cycles

ILE-Speak

- ILE
  - Programs & Service Programs
    - Modules
      - Cycle Module
      - Linear Module
    - Procedures and Functions
      - Cycle-Main Procedure
      - Linear Main Procedure
    - Sub-Procedures & Sub-Functions

Activation Groups

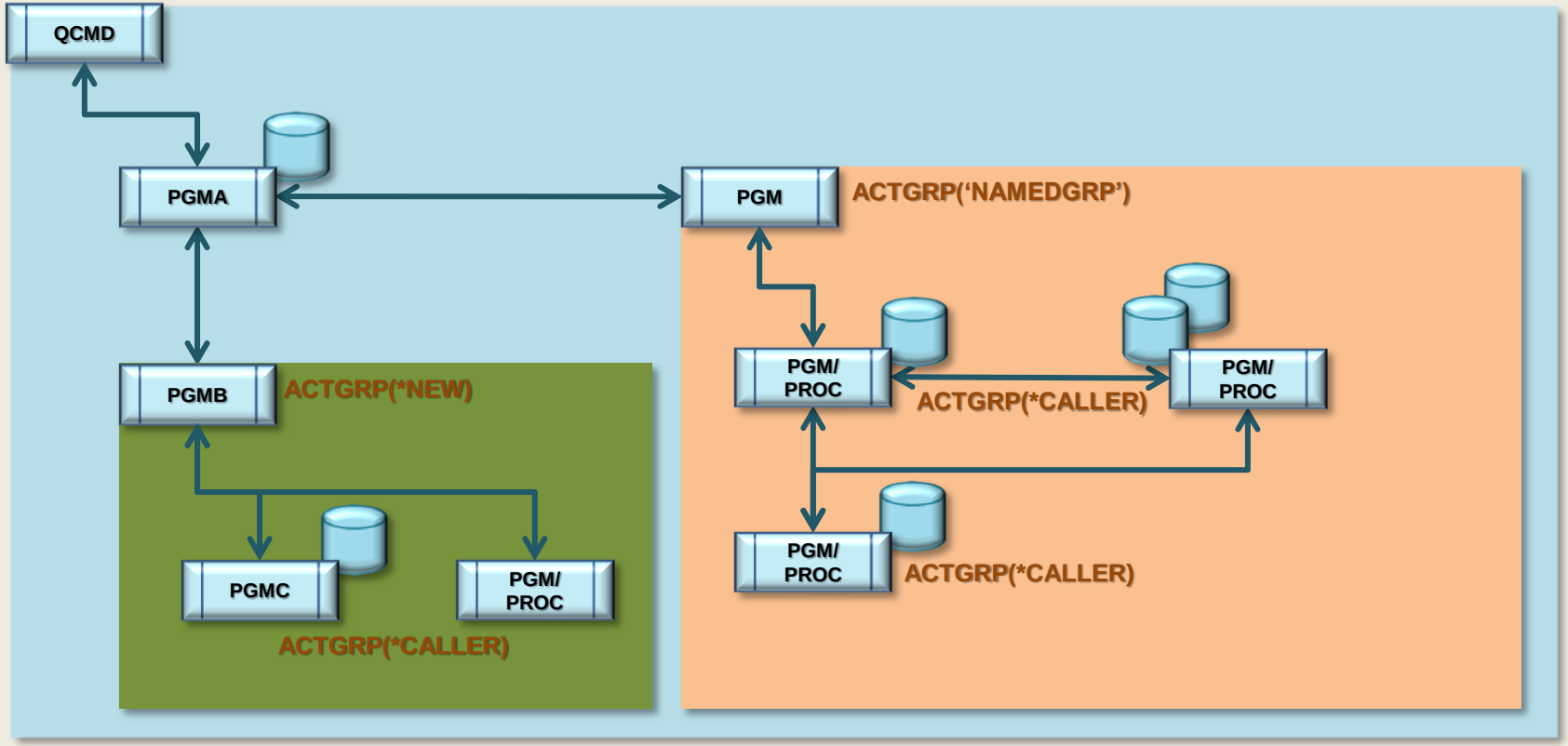
***DFTACTGRP**

QCMD

- **When a job is started on the IBMi, a *DFTACTGRP is created, and it cannot be terminated except by ending the job. *DFTACTGRP is where all original OPM program objects run as well as all OS functions.**
- **In addition, limited-function RPG IV programs can run in *DFTACTGRP. Limited function is defined as programs that don't contain any procedures, don't call any procedures, and use no contemporary built-in functions (BIFs) as well as any features that require procedures not supported by *DFTACTGRP.**

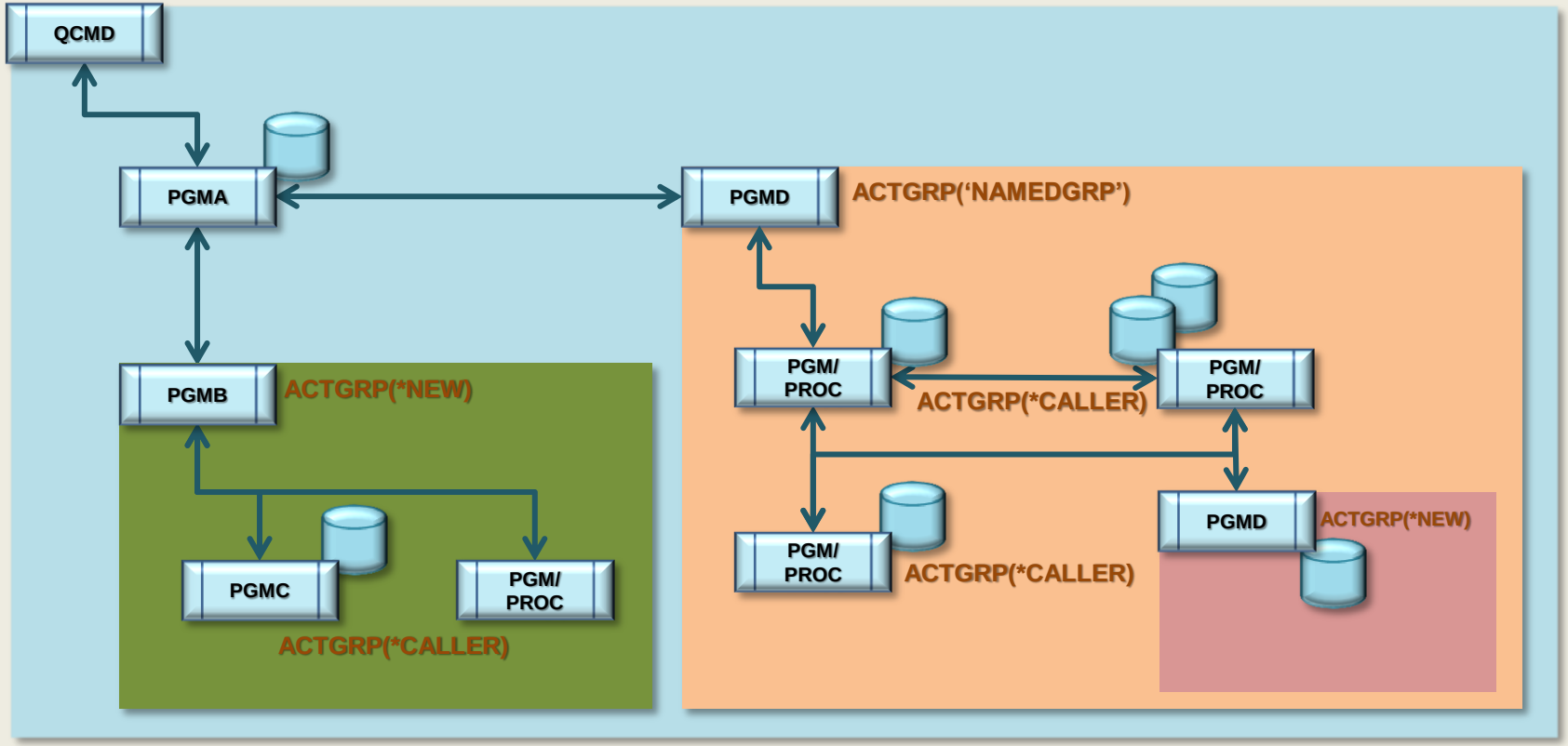
Activation Groups

***DFTACTGRP**

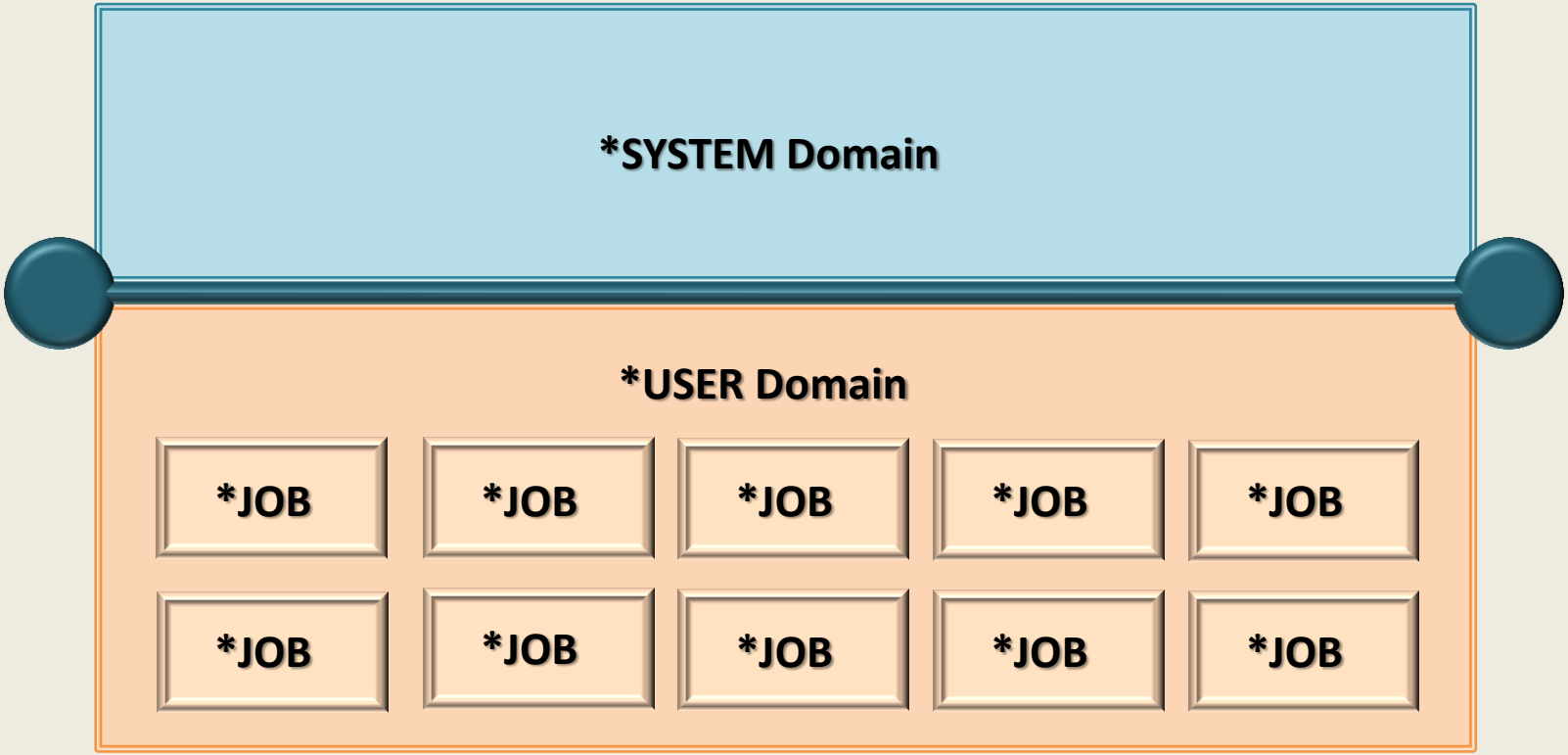


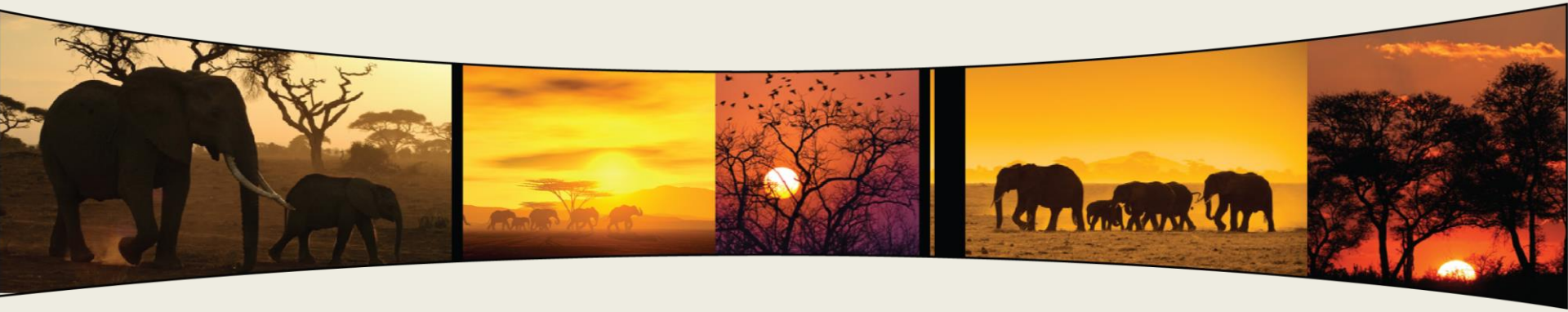
Activation Groups

***DFTACTGRP**

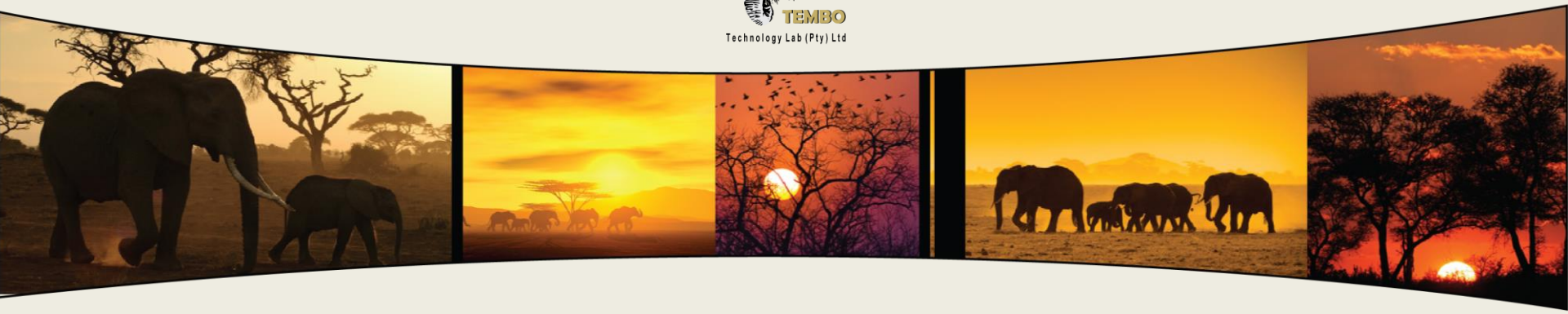


The Bigger Picture





TEMBO
Technology Lab (Pty) Ltd



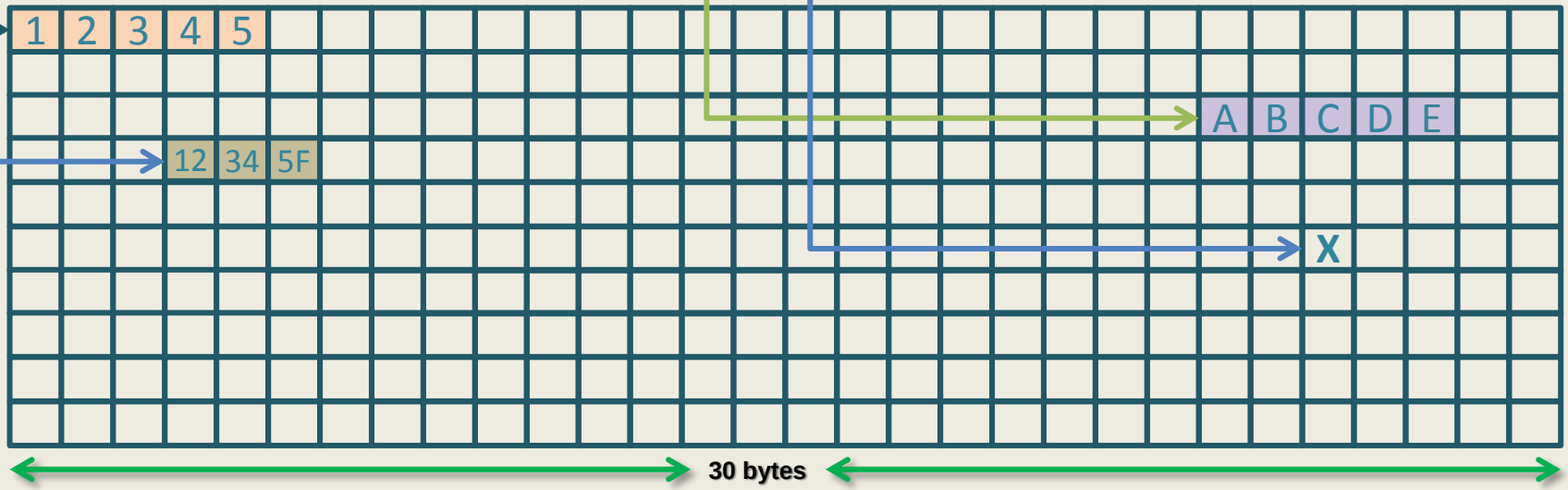
RPG IV for Integrated Language Environment

Pointers and Data Exports

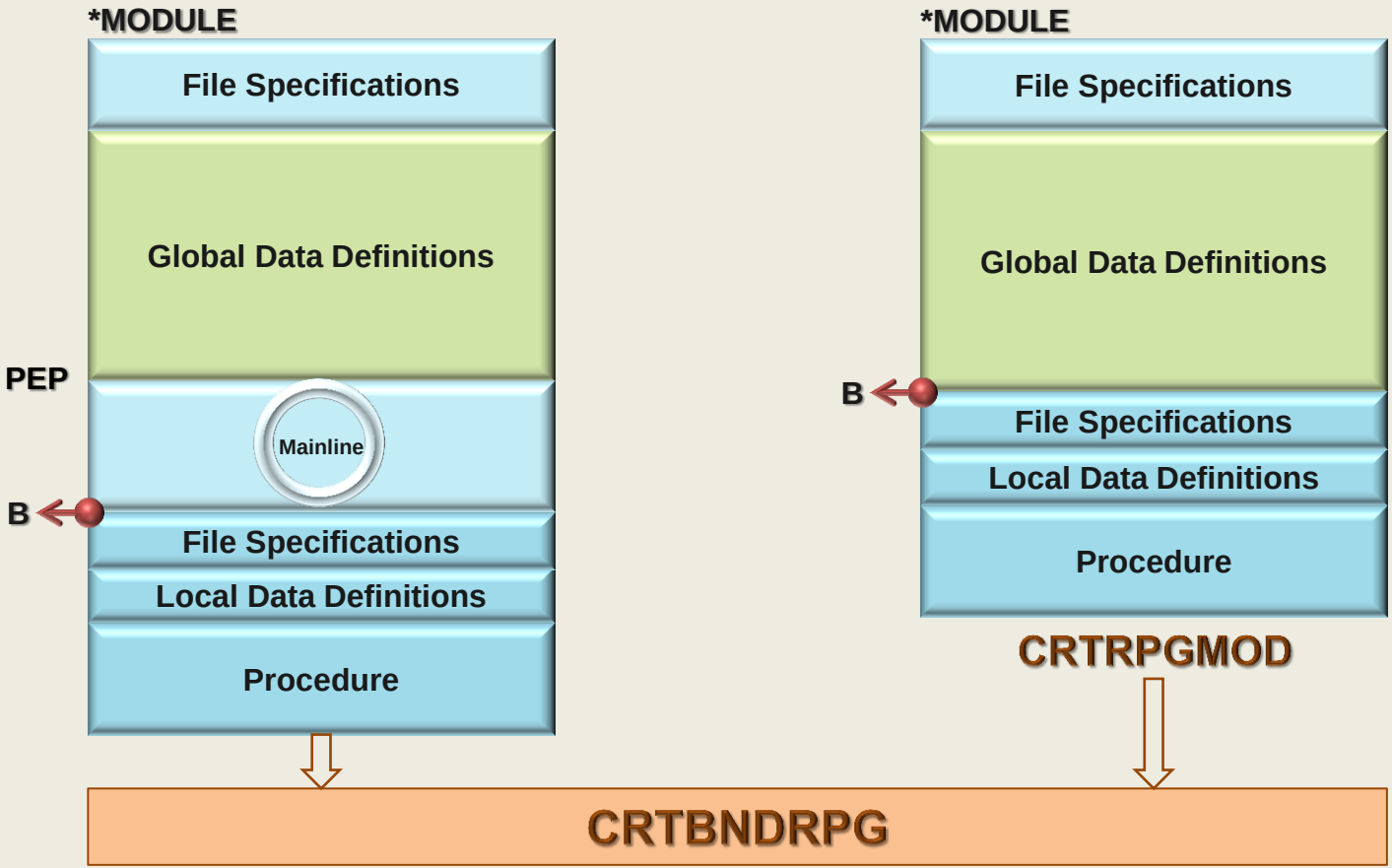
Pointers

```
SPP = 1C5736254E001000
Offset = 83

SPP += Offset

SPP += (Offset + 10)
```

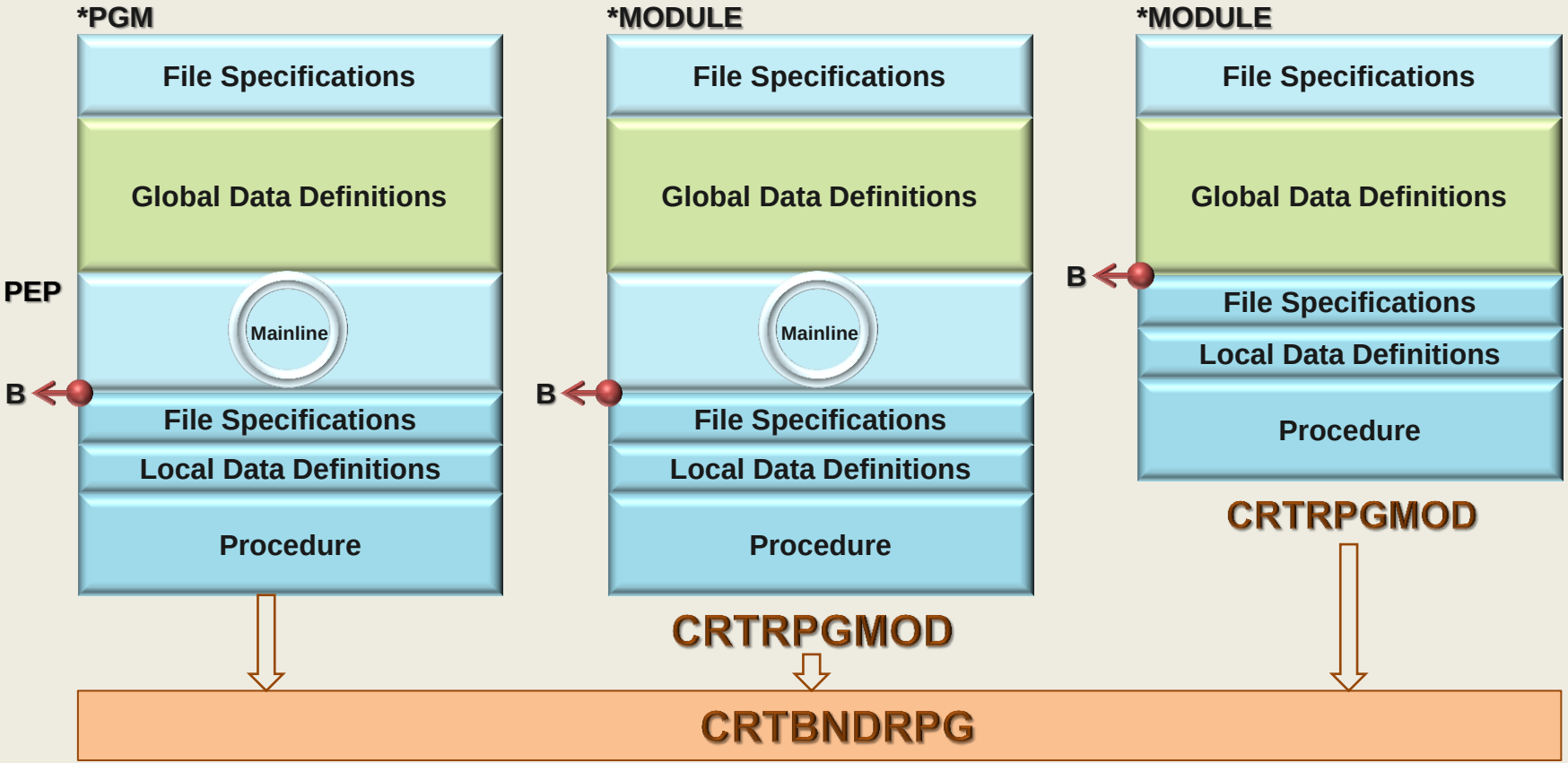
```
d SPP      s        *
d Offset   s      10U 0

d Field1   s       5       based(SPP)
d Field2   s       5p 0    based(SPP)
d Field3   s       5s 0    based(SPP)
```



Programs from *MODULES

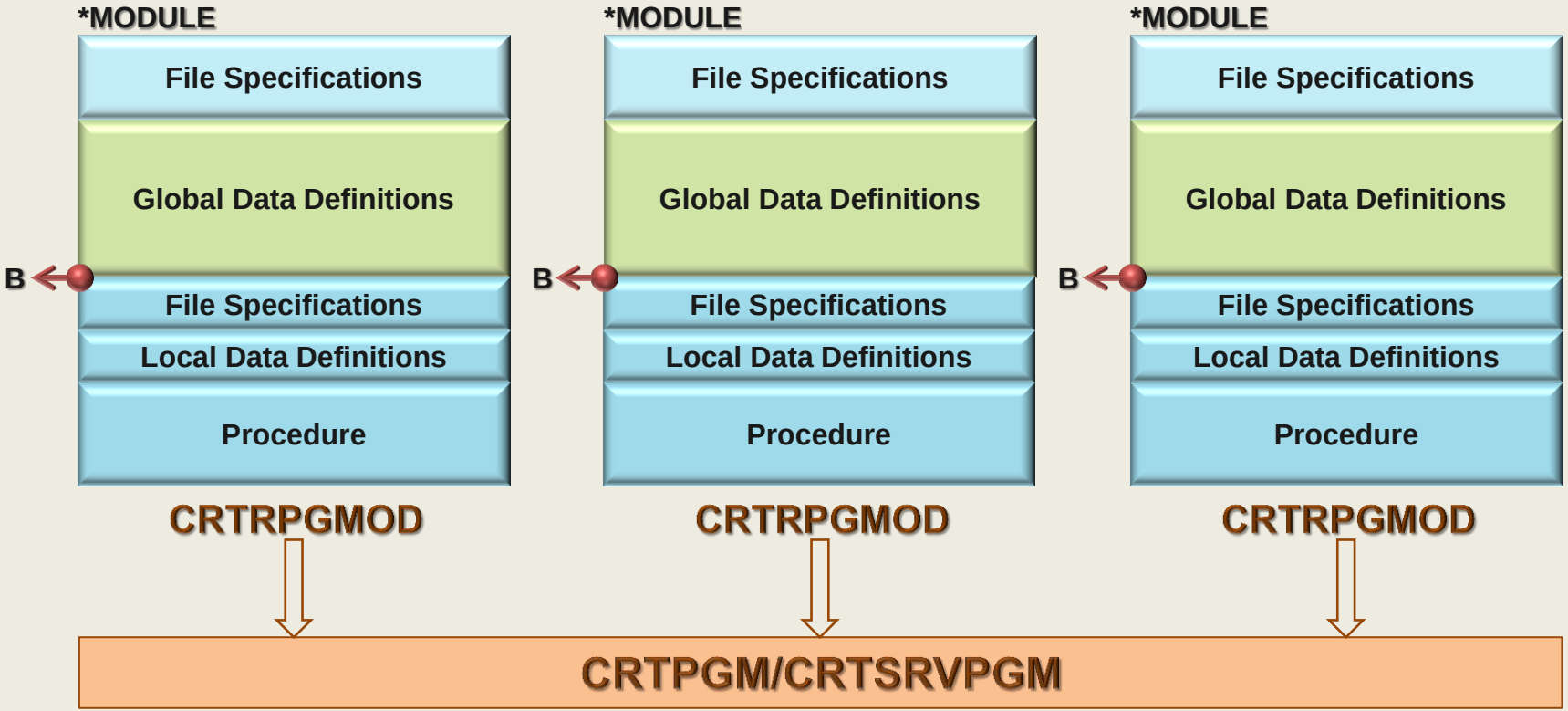


Programs from *MODULES



Programs from *MODULES

MAIN(procedure_name) or NOMAIN



Data Exports/Imports

Module01

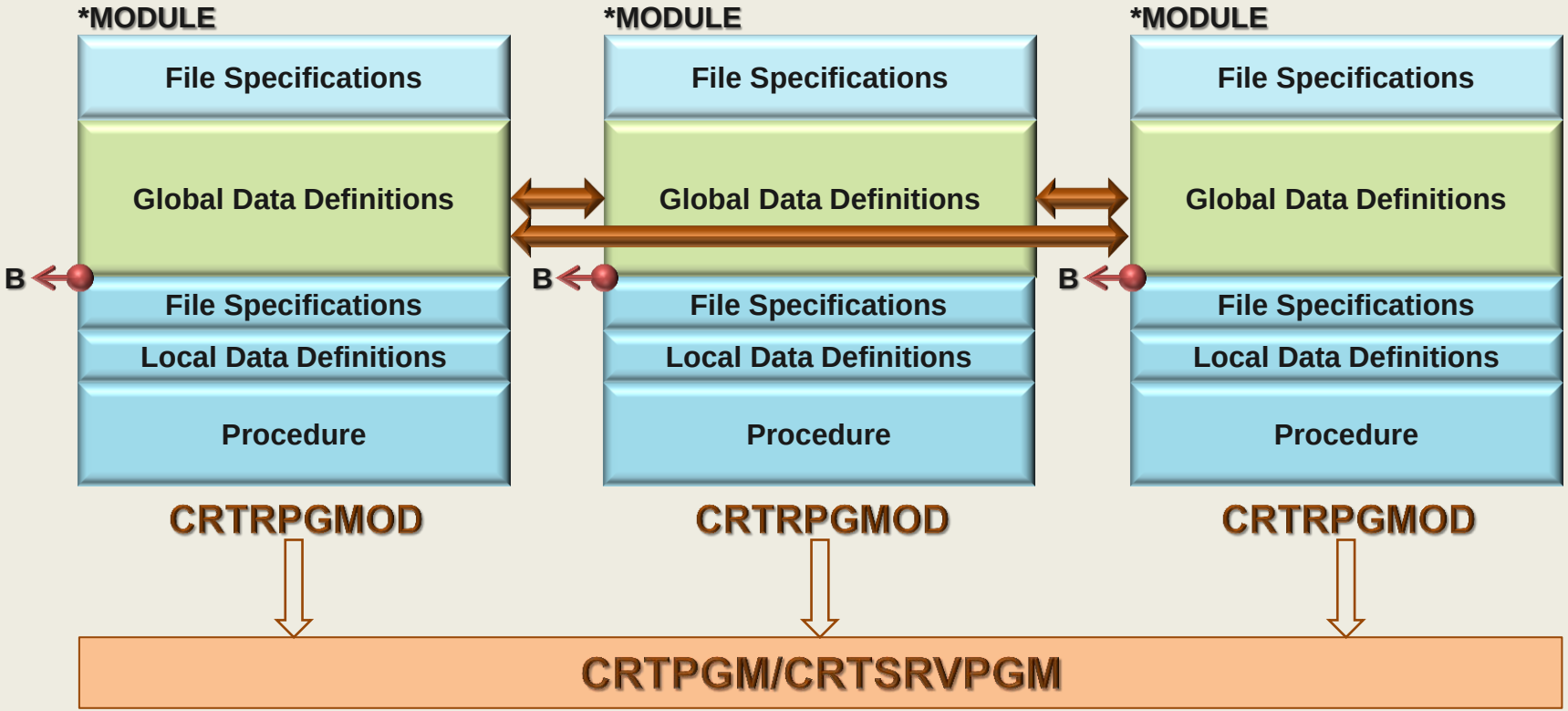
```
d UIN             s                10    export inz('EXECDMI')
d UIKL            s                10i 0 export inz(%len(UIK))
d UIP             s                  *   export inz(%addr(UI))
d UIEP            s                  *   export
 *
d DS_FILEA      e ds                       export extname('FILEA')
```

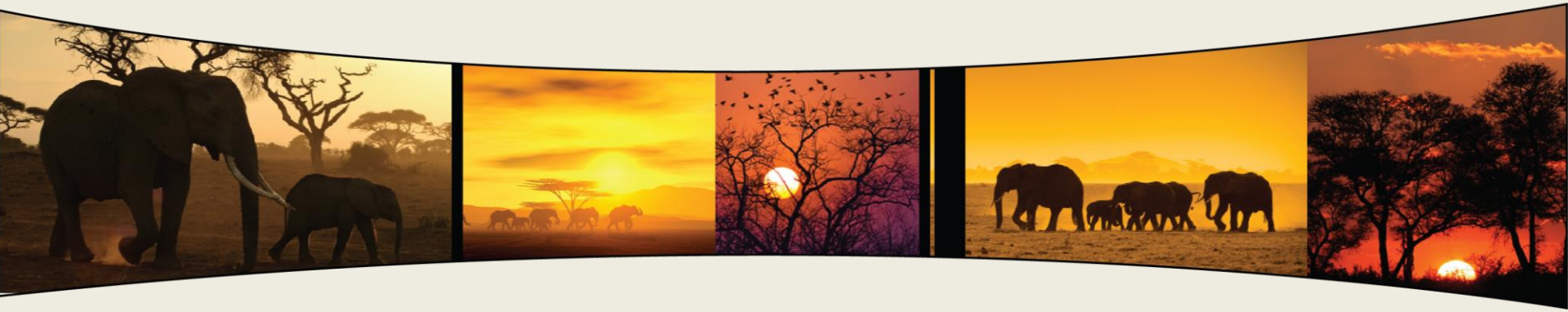
Module02

```
d UIN             s                10    import
d UIKL            s                10i 0 import
d UIP             s                  *   import
d UIEX            s                  *   import(UIEP)
 *
d DS_FILEA      e ds                       import extname('FILEA')
```

Data Exports/Imports

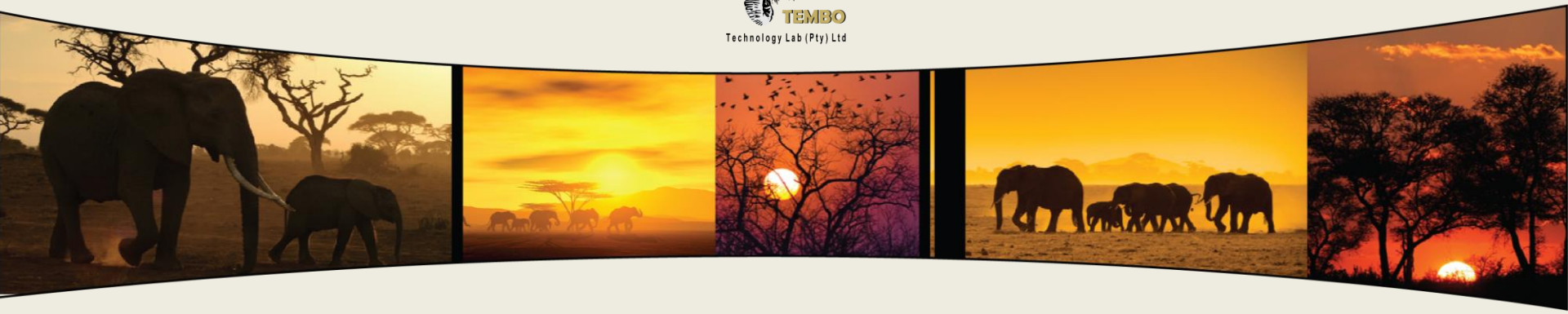
MAIN(procedure_name) or NOMAIN



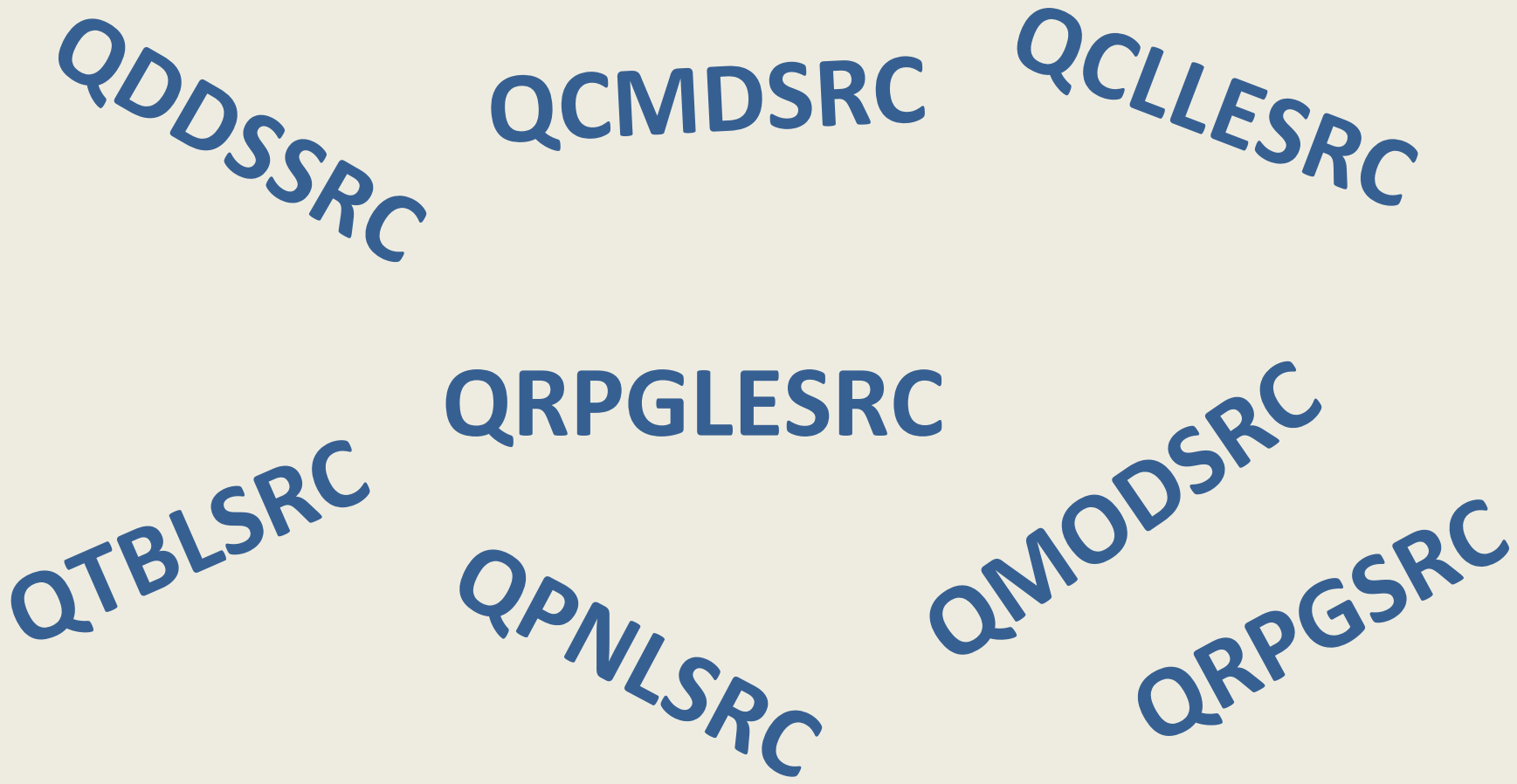


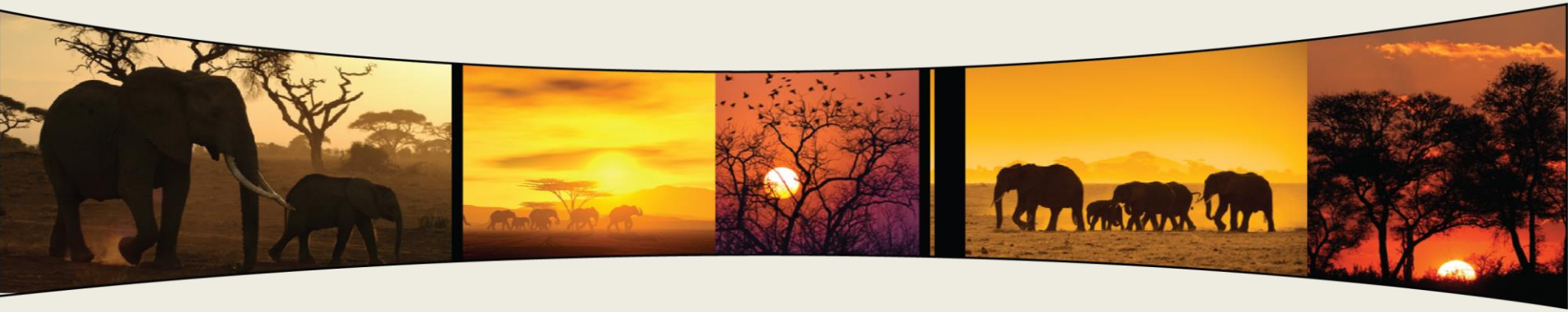
African Pygmy Kingfisher

TEMBO
Technology Lab (Pty) Ltd

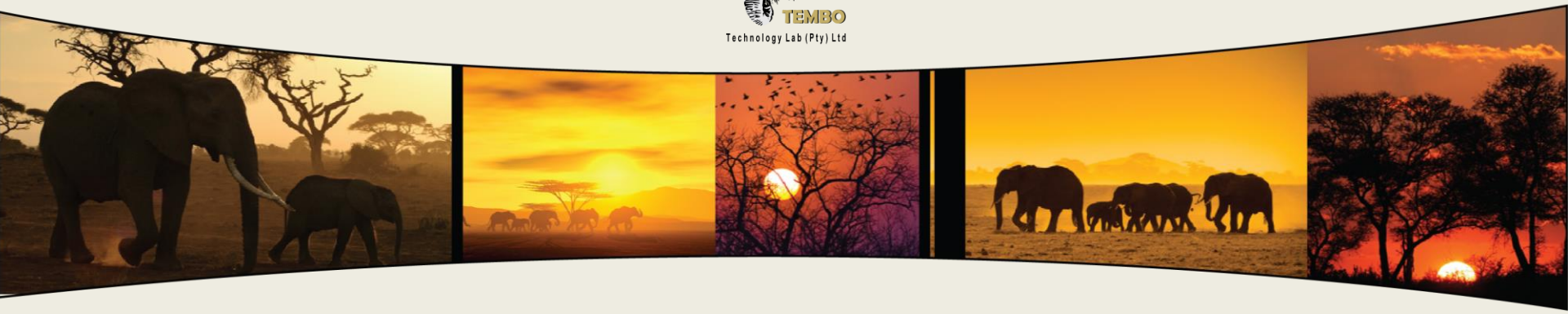


! Naming & Other Standards !


QDDSSRC
QCMDSRC
QCLLESRC
QRPGLESRC
QTBLSRC
QMODSRC
QPNLSRC
QRPGSRC



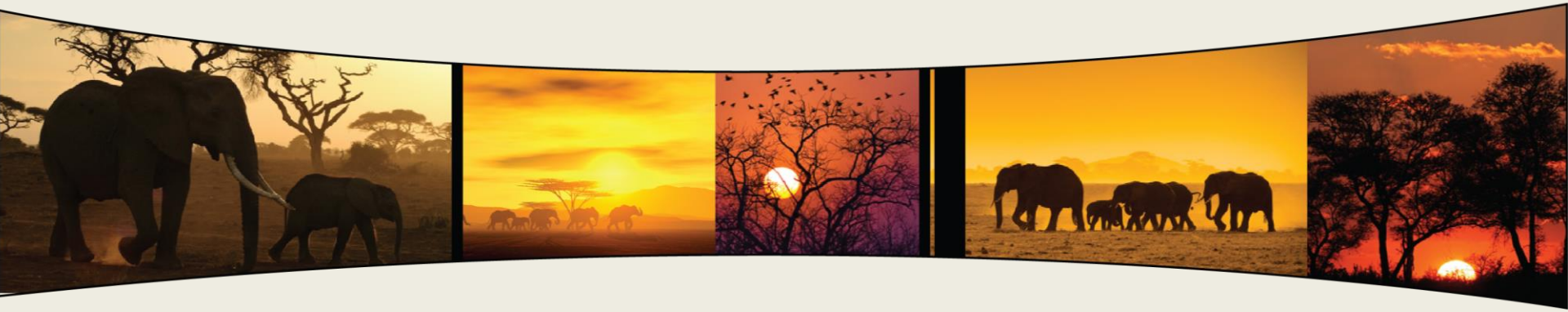
TEMBO
Technology Lab (Pty) Ltd



RPG IV for Integrated Language Environment

Imbedded SQL

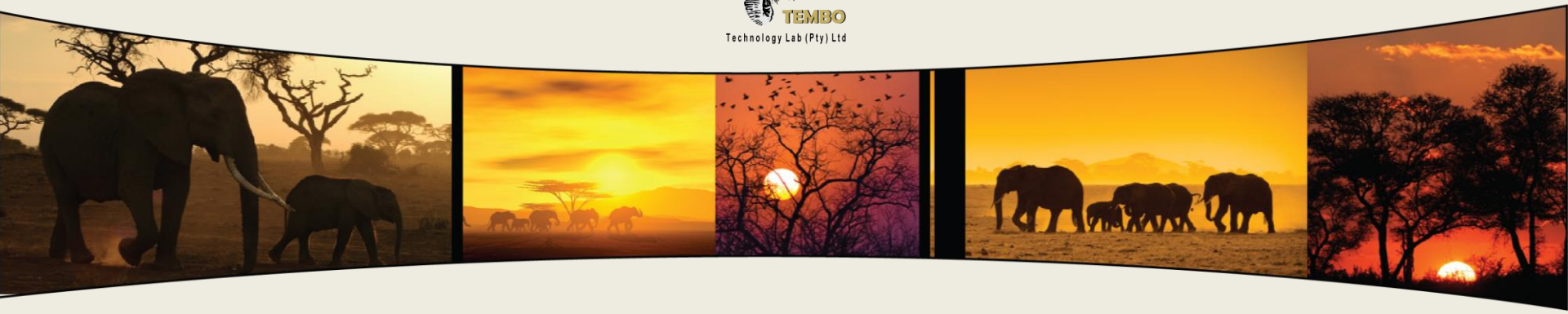
```
//=============================================================
SQLEXE = 'SELECT * FROM AOF130F WHERE '+%trim(SQL1) +
         ' ORDER BY '+ORDR+' FOR READ ONLY';
//=============================================================
exec sql prepare AOF130F_S1 from :SQLEXE;
exec sql declare AOF130F_C1 cursor for AOF130F_S1;
exec sql open    AOF130F_C1;
//=============================================================
dow 0=0;
    //=========================================================
    exec sql fetch AOF130F_C1 into :RECIN;
    //=========================================================
    if ERH50(CAPtr);
       SETLSTALL(P.APPH);
       leave;
    endif;
    //=========================================================
    if FMTSEQ = 1;
       EEPP = '1101';
    else;
       EEPP = '0111';
       DEFNTP = ' ';
    endif;
    //=========================================================
    eval-corr BufferDS = AOF130FR;
    BUFFER = BufferDS;
    ADDLSTLE(P.APPH:AOF130FP:AOF130FL:'AOF130FR');
    //=========================================================
enddo;
//=============================================================
exec sql close AOF130F_C1;
*inlr=*on;
return;
//=============================================================
```




Secretary Bird


Kudu Bull

TEMBO
Technology Lab (Pty) Ltd



RPG IV for Integrated Language Environment

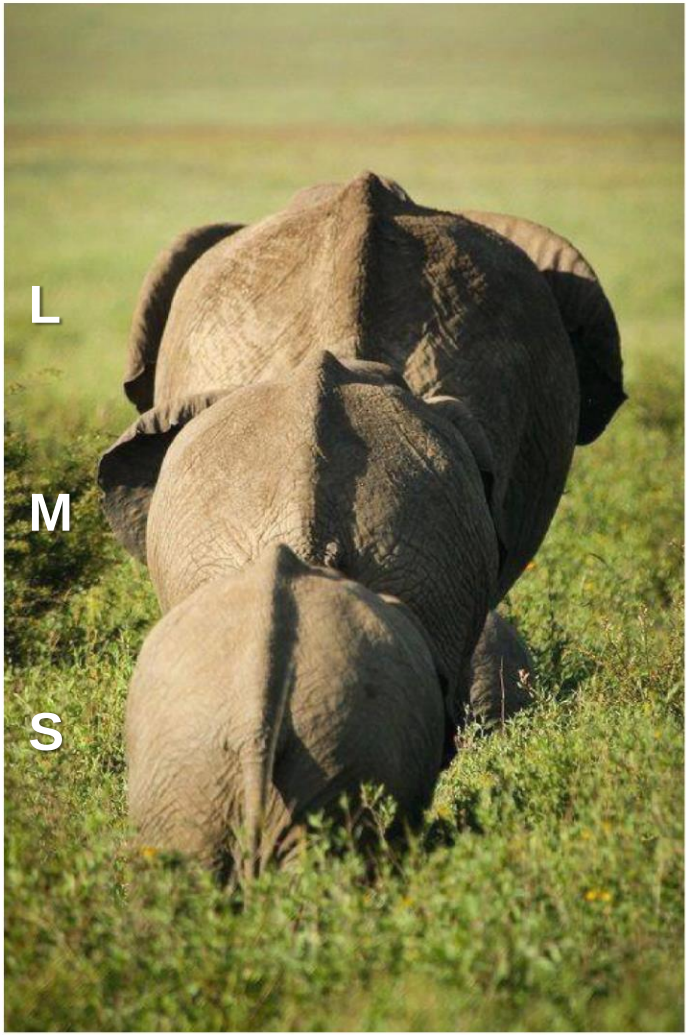
User Spaces & User Indexes

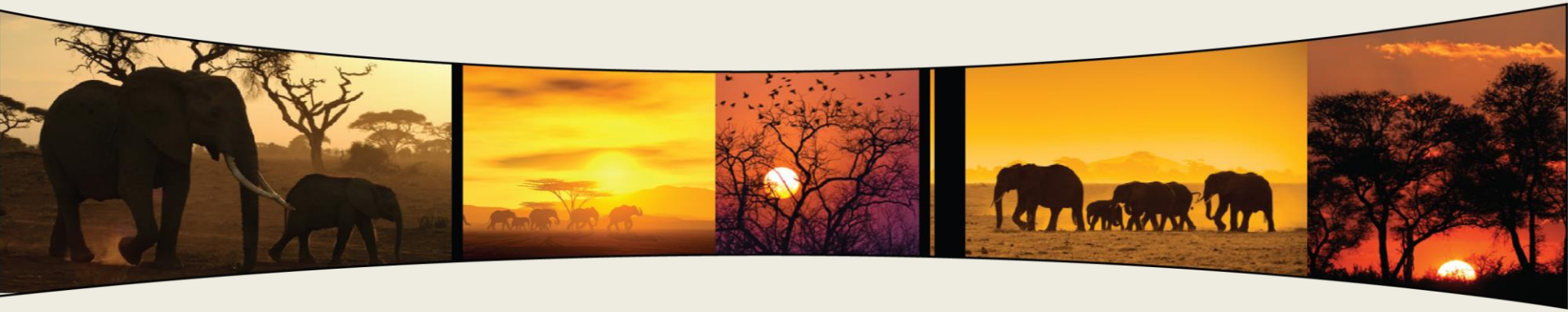
User Space API's

- Create User Space (QUSCRTUS) API
- Change User Space Attributes (QUSCUSAT) API
- Retrieve Pointer to User Space (QUSPTRUS) API
- Delete User Space (QUSDLTUS) API
- Change User Space (QUSCHGUS) API
- Retrieve User Space (QUSRTVUS) API
- Retrieve User Space Attributes (QUSRUSAT) API

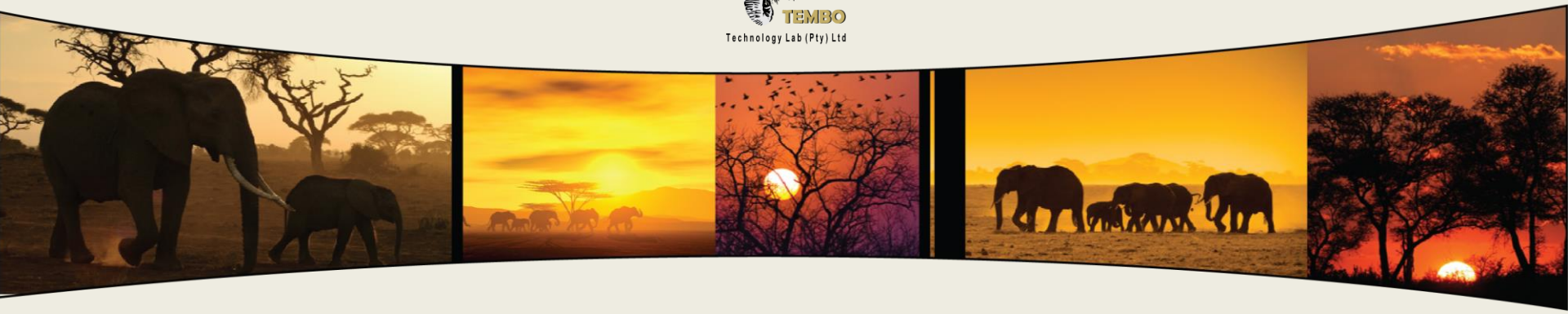
User Index API's

- Create User Index (QUSCRTUI) API
- Add User Index Entries (QUSADDUI) API
- Retrieve User Index Entries (QUSRTVUI) API
- Remove User Index Entries (QUSRMVUI) API
- Delete User Index (QUSDLTUI) API
- Retrieve User Index Attributes (QUSRUIAT) API


L
M
S



TEMBO
Technology Lab (Pty) Ltd



RPG IV for Integrated Language Environment

Application Programming Interfaces

API Overview and Concepts

i5/OS® application programming interfaces (APIs) allow your application program written in a high-level language to use specific data or functions of the IBM® i5/OS operating system.

IBM intends that the APIs will continue to work as they originally worked, and any existing applications that use the APIs will continue to work without any incompatible changes in future releases. Significant architectural changes, however, might necessitate incompatible changes.

Additionally, some API definitions, such as the UNIX® type of API definitions, are established by industry standards organizations, where the degree of compatibility is determined by the organizations.

API Overview and Concepts

An application programming interface (API) is a functional interface supplied by the operating system or a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

- **Some APIs provide the same functions as control language (CL) commands and output file support.**
- **Some APIs provide functions that CL commands do not.**
- **Most APIs work more quickly and use less system overhead than the CL commands.**
- **Some APIs have no equivalent CL commands.**

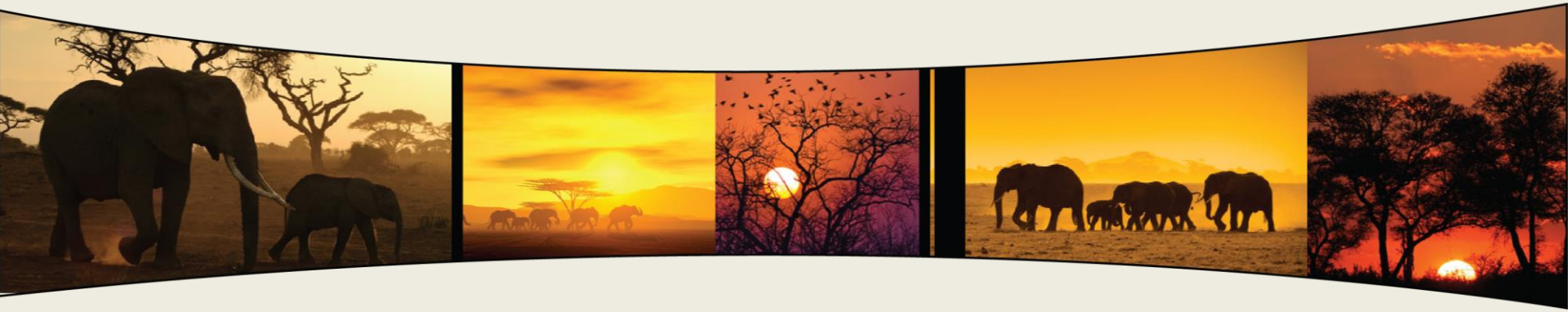
Advantages of API's

- APIs provide better performance when getting system information or when using system functions that are provided by CL commands or output file support.
- APIs provide system information and functions that are not available through CL commands.
- You can use calls from high-level languages to APIs.
- You can access system functions at a lower level than what was initially provided on the system.
- Data is often easier to work with when returned by an API.

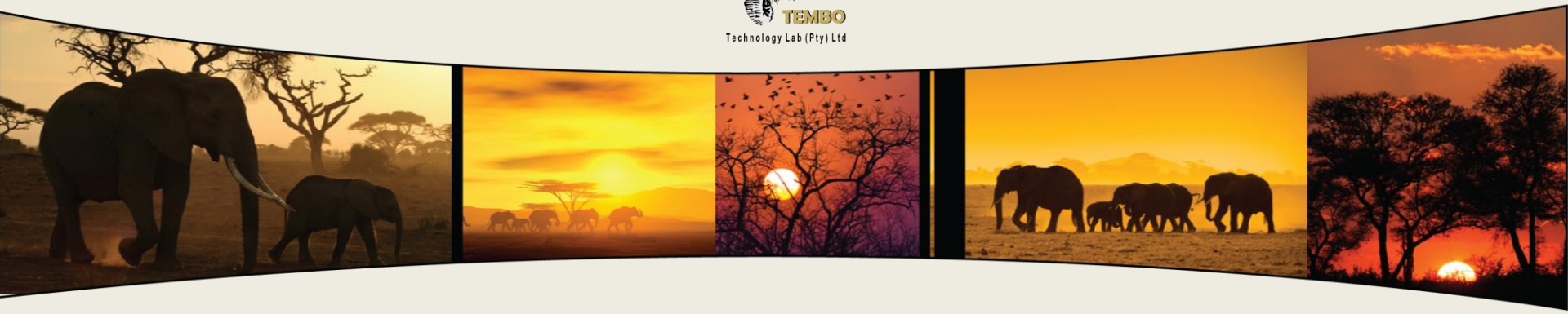
API Terminology

Before using the i5/OS APIs, you need to understand several terms that refer to i5/OS objects.

- Binding Directory (*BNDDIR)
  - An object that contains a list of names of modules and service programs.
- Data Queue (*DTAQ)
  - An object that is used to communicate and store data used by several programs in a job or between jobs.
- Module (*MODULE)
  - An object that is made up of the output of the compiler.
- Program (*PGM)
  - A sequence of instructions that a computer can interpret and run. A program can contain one or more modules.
- Service Program (*SRVPGM)
  - An object that packages externally supported callable routines into a separate object.
- User Index (*USRIDX)
  - An object that provides a specific order for byte data according to the value of the data.
- User Queue (*USRQ)
  - An object consisting of a list of messages that communicate information to other application programs. Only programming languages that can use machine interface (MI) instructions can access *USRQ objects.
- User Space (*USRSPC)
  - An object consisting of a collection of bytes used for storing any user-defined information.

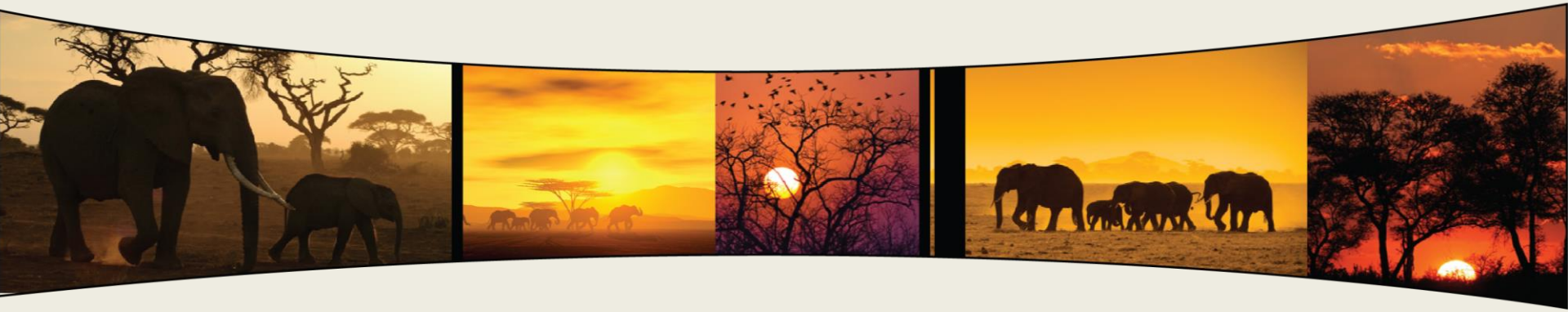


TEMBO
Technology Lab (Pty) Ltd

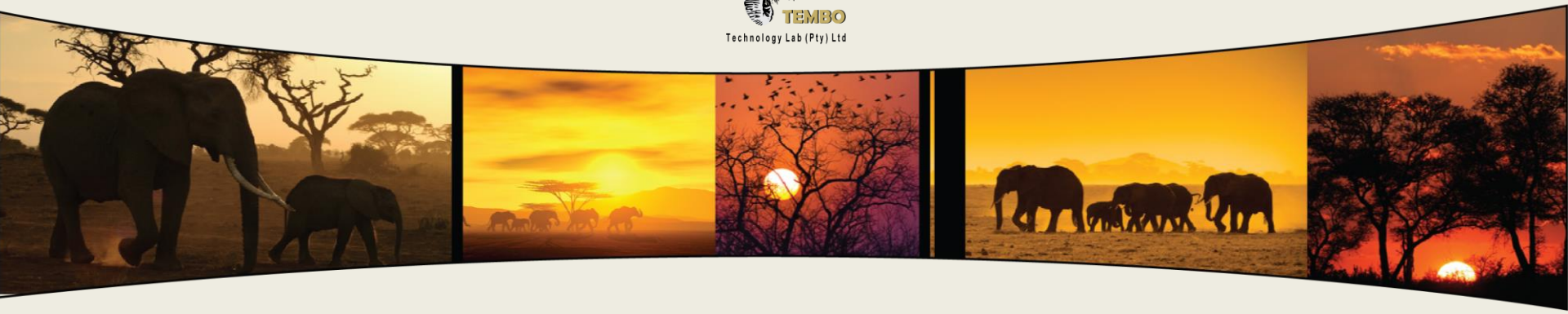


RPG IV for Integrated Language Environment

Built in Functions (BIF's)

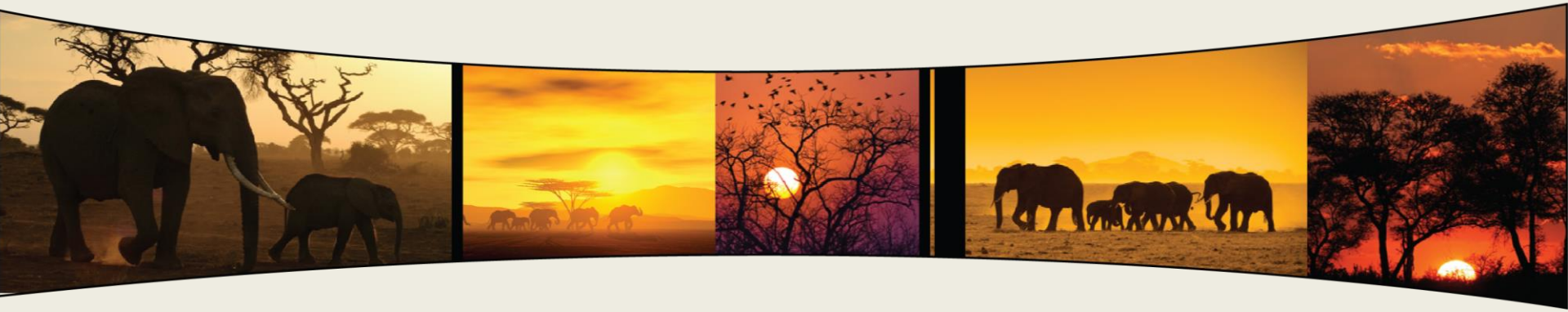


TEMBO
Technology Lab (Pty) Ltd

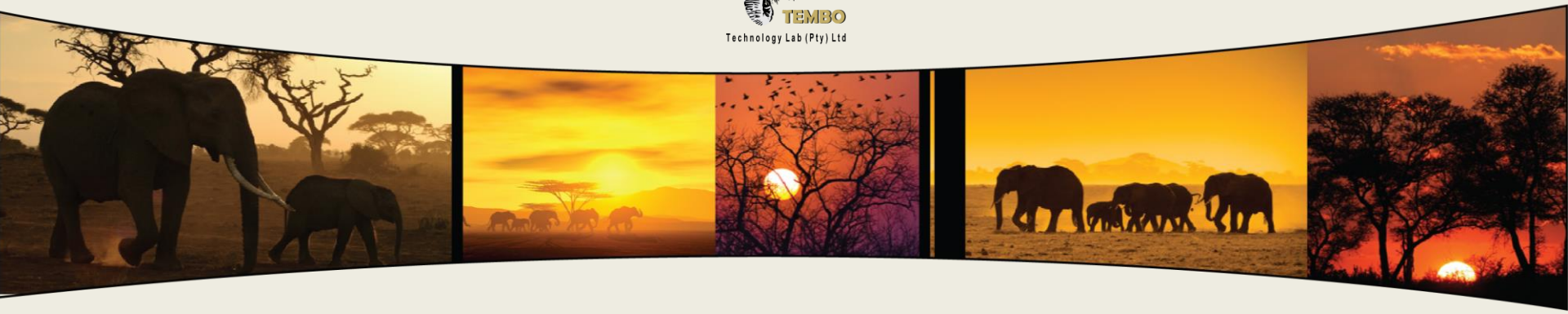


RPG IV for Integrated Language Environment

User Defined Commands

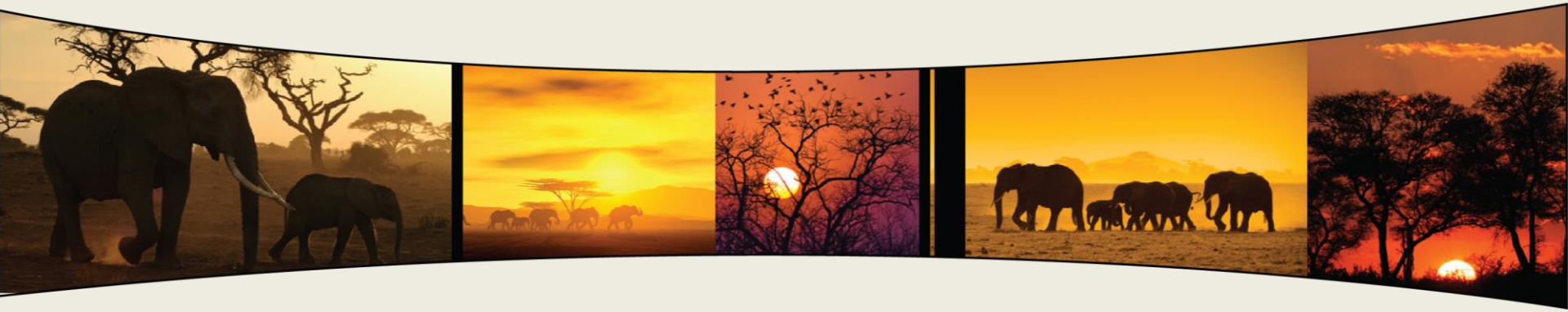


TEMBO
Technology Lab (Pty) Ltd

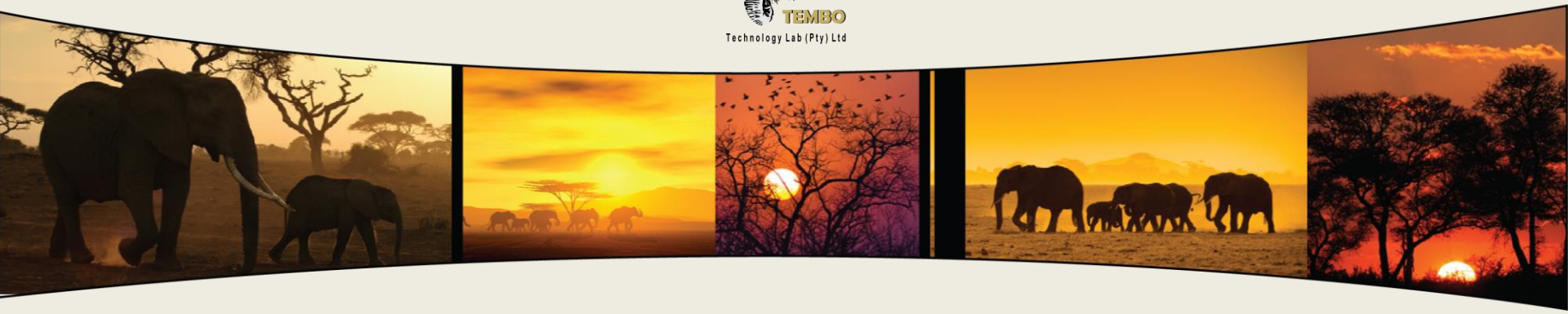


RPG IV for Integrated Language Environment

Stored Procedures



TEMBO
Technology Lab (Pty) Ltd



RPG IV for Integrated Language Environment

Rational Open Access: RPG Edition

New HANDLER keyword

Other than the HANDLER keyword, there is no new RPG syntax related to using an Open Access file

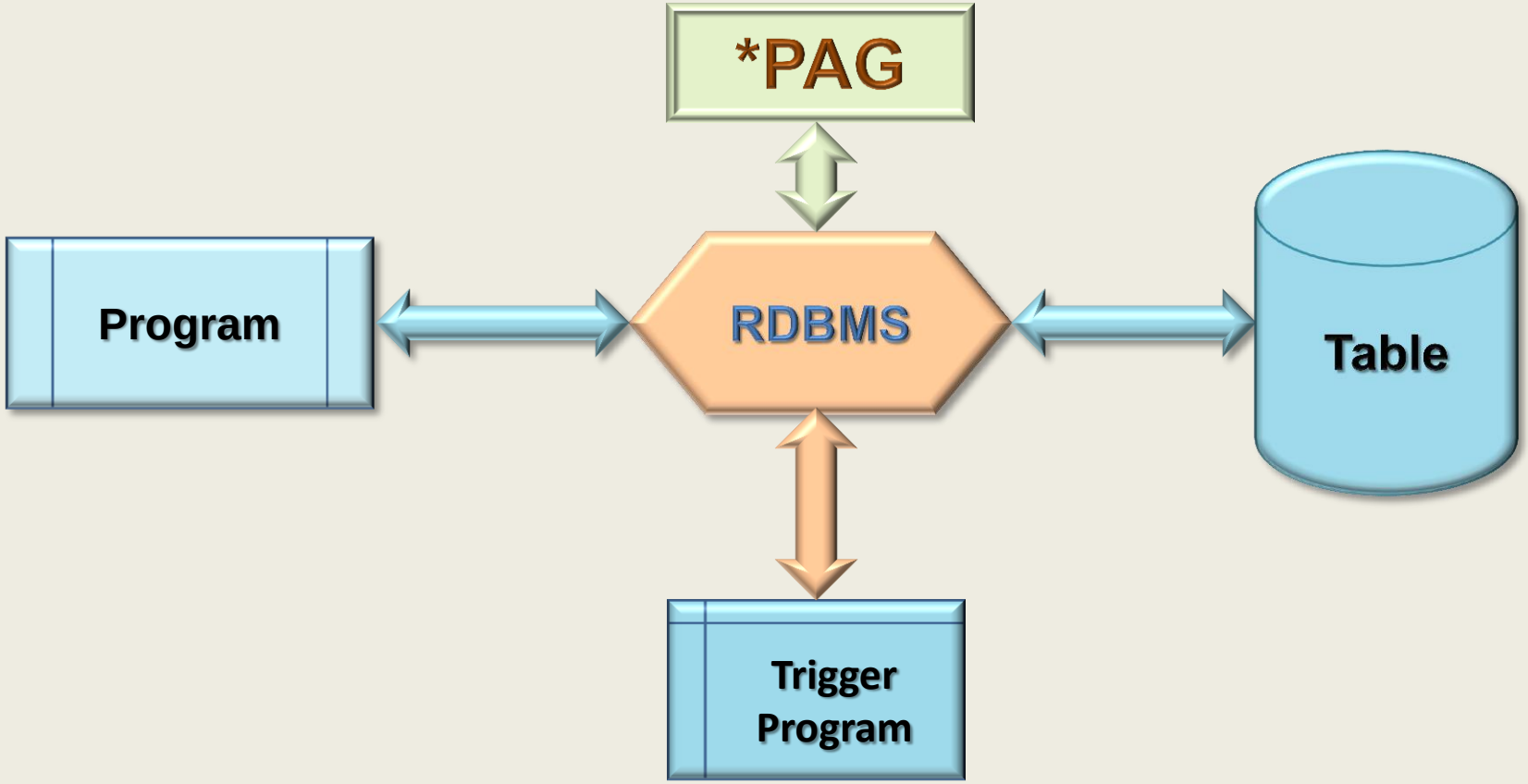
```
Fmyfile CF E             WORKSTN HANDLER('MYLIB/MYSRV(hdlMyfile)'[:myds])
```

- **A character literal or variable identifying a procedure in a service program in the form 'LIBRARY/SRVPGM(procedure)'.**
- **A character literal or variable identifying a program in the form 'LIBRARY/PGM'.**
- **A prototype for a bound procedure.**
- **A procedure pointer literal (%PADDR) or variable.**
- **All names are case-sensitive.**

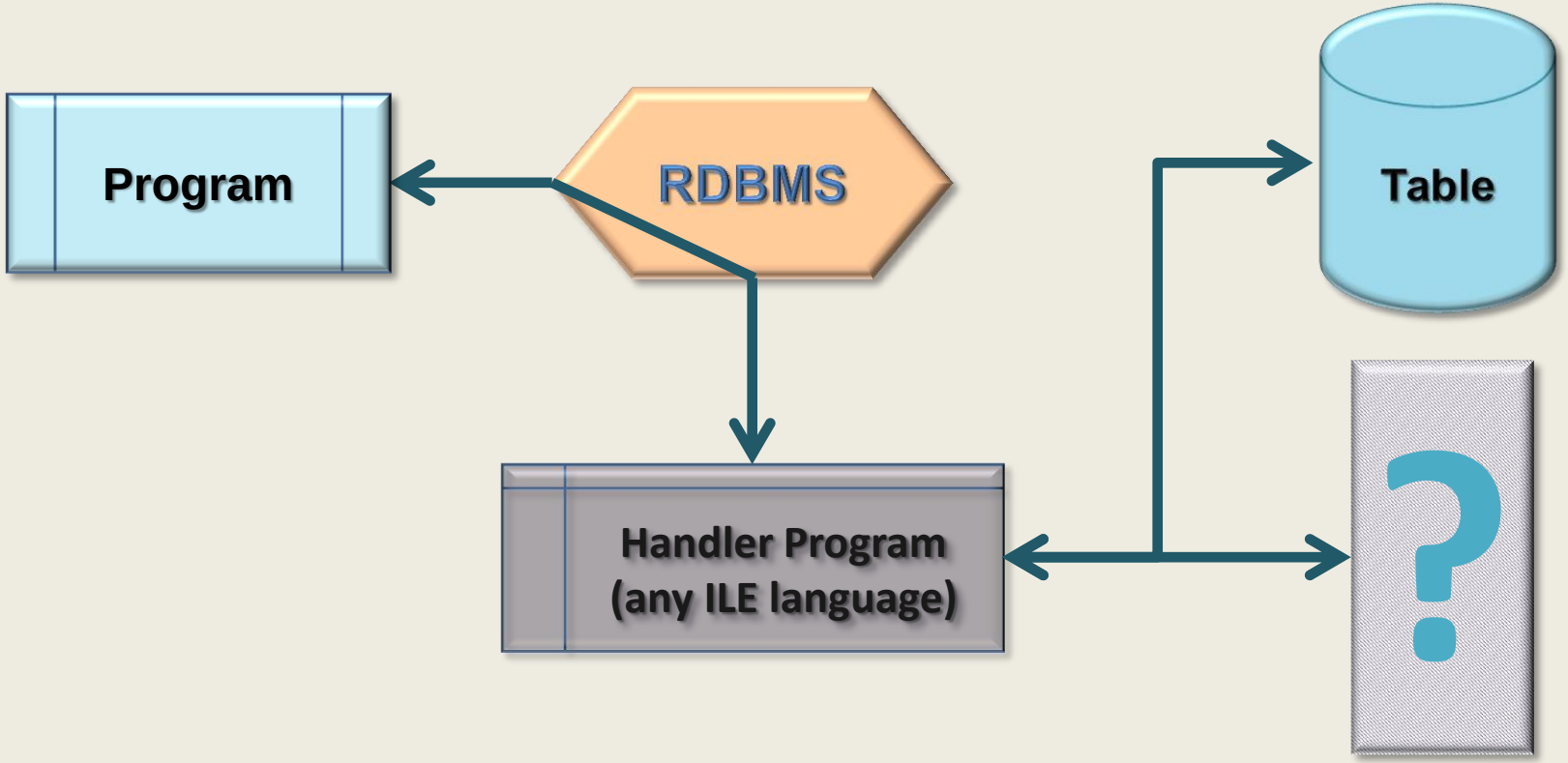
Rational® Open Access: RPG Edition

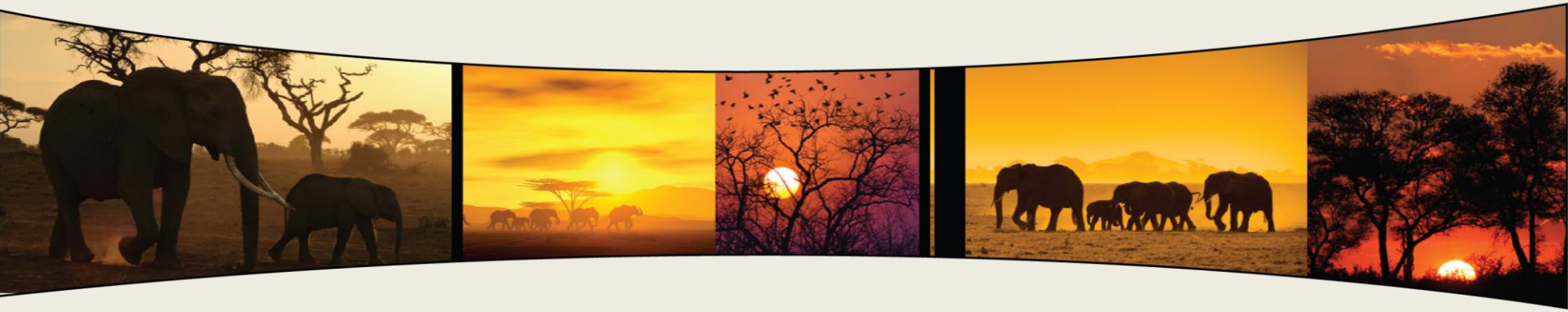
- **Provides a way for RPG programmers to use the simple and well-understood RPG I/O model to access resources and devices that are not directly supported by RPG.**
- **Open Access opens up RPG's file I/O capabilities, allowing anyone to write innovative I/O handlers to access other devices and resources such as:**
  - **Browsers**
  - **Mobile devices**
  - **Cloud computing resources**
  - **Web services**
  - **External databases**
  - **XML files**
  - **Spreadsheets**
  - **And more**

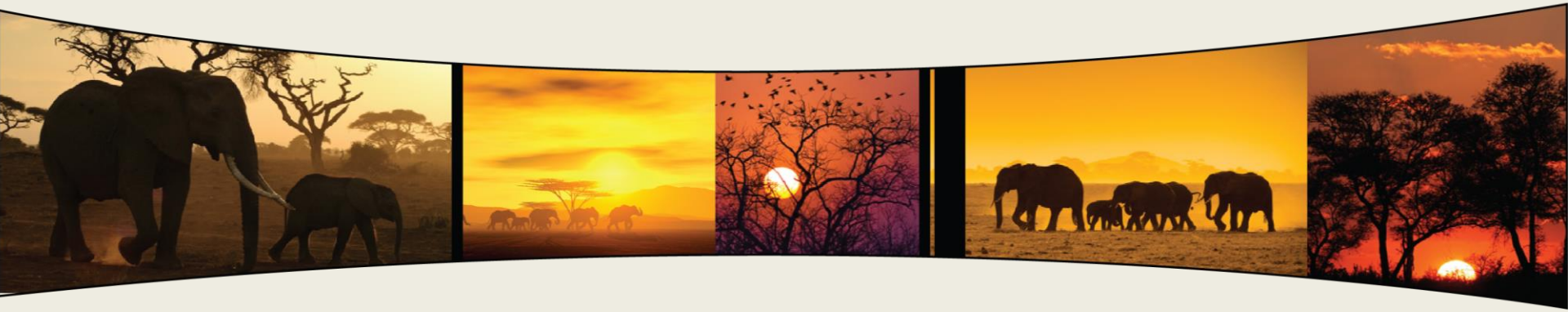
ROA : RPG Edition



ROA : RPG Edition



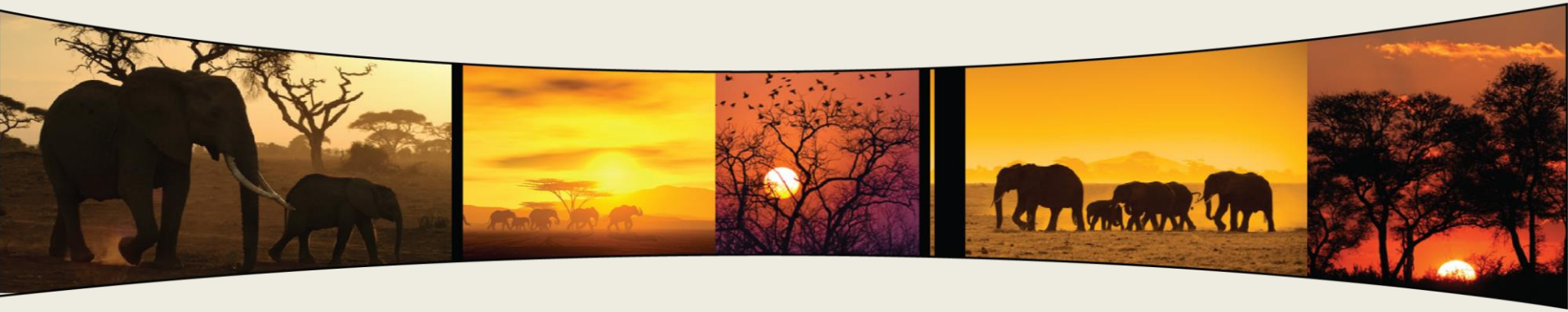




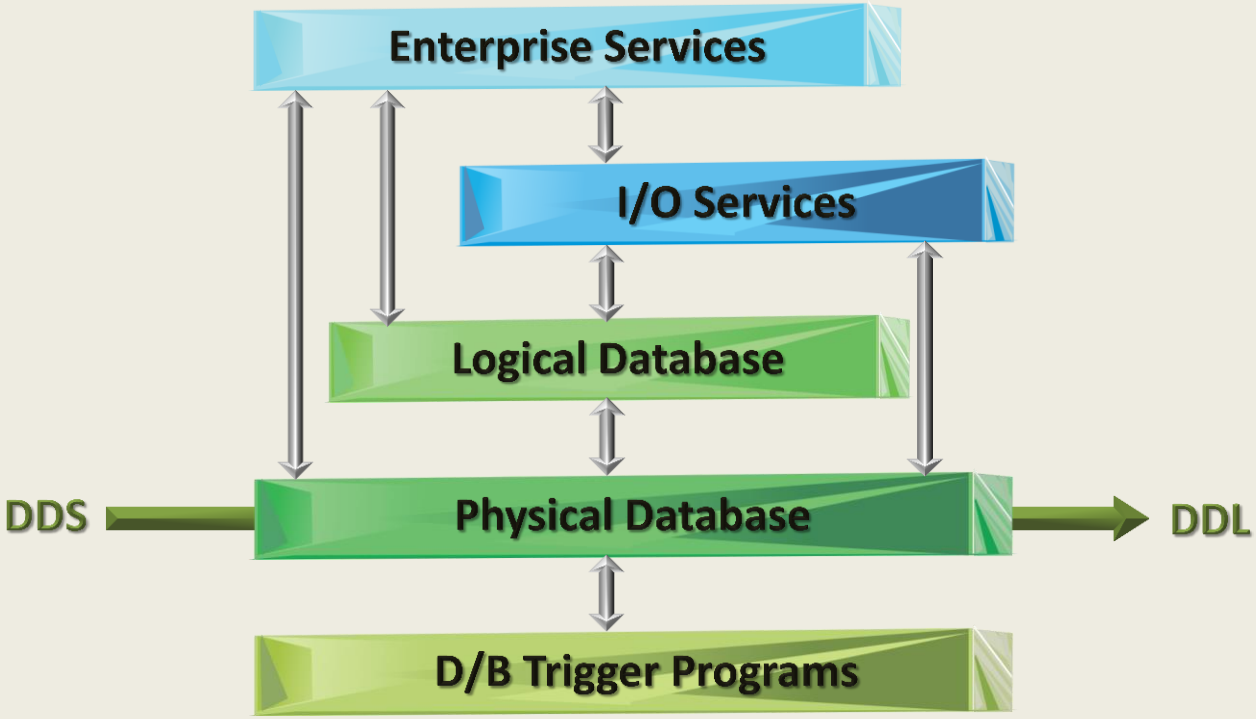
The End


© Pete Bette
Photography


Mashtulele

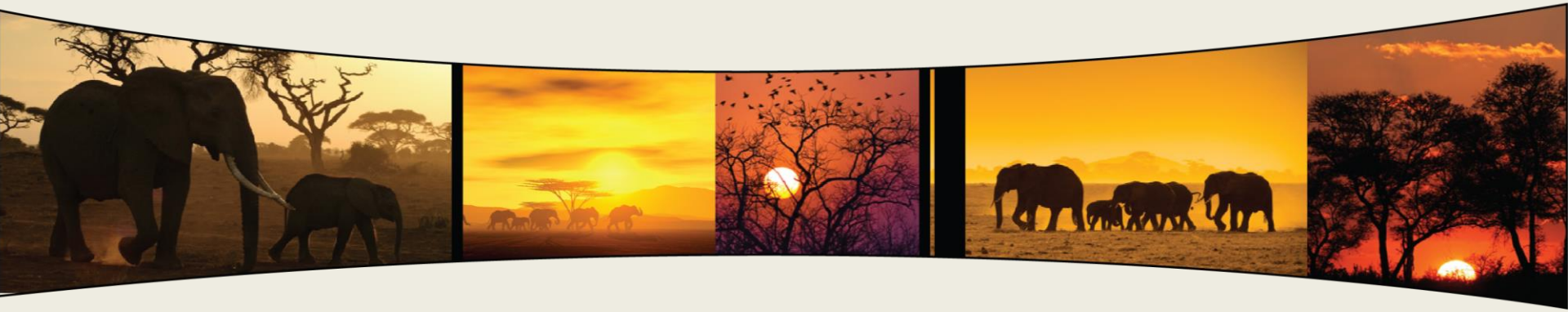


Where and How do we begin ?

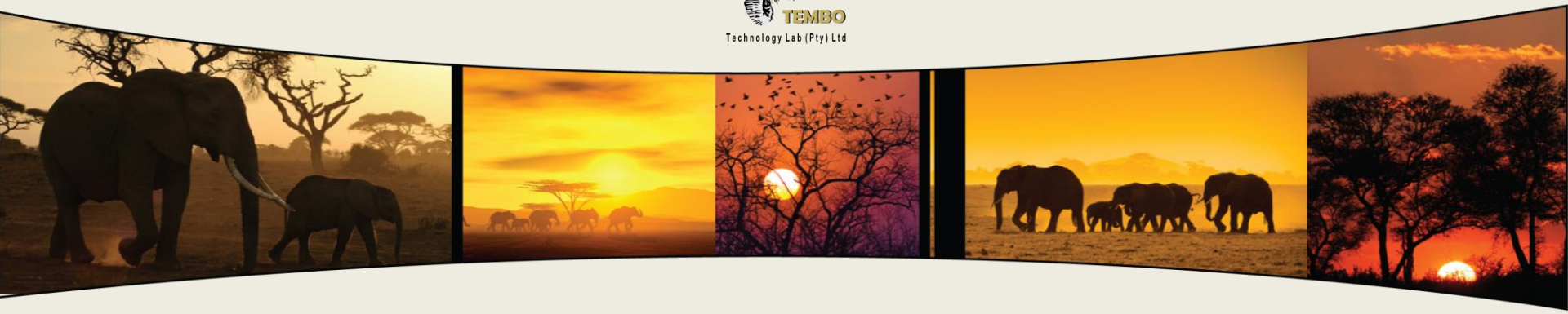


Where and How do we begin ?

- Unique Keys
- Migrate DDS to DDL
- Add Constraints
- Database Normalization
- Dictionary Sanitization
- Logical Database
- Add Event Triggers
- Create I/O Servers
- Enterprise Services



TEMBO
Technology Lab (Pty) Ltd



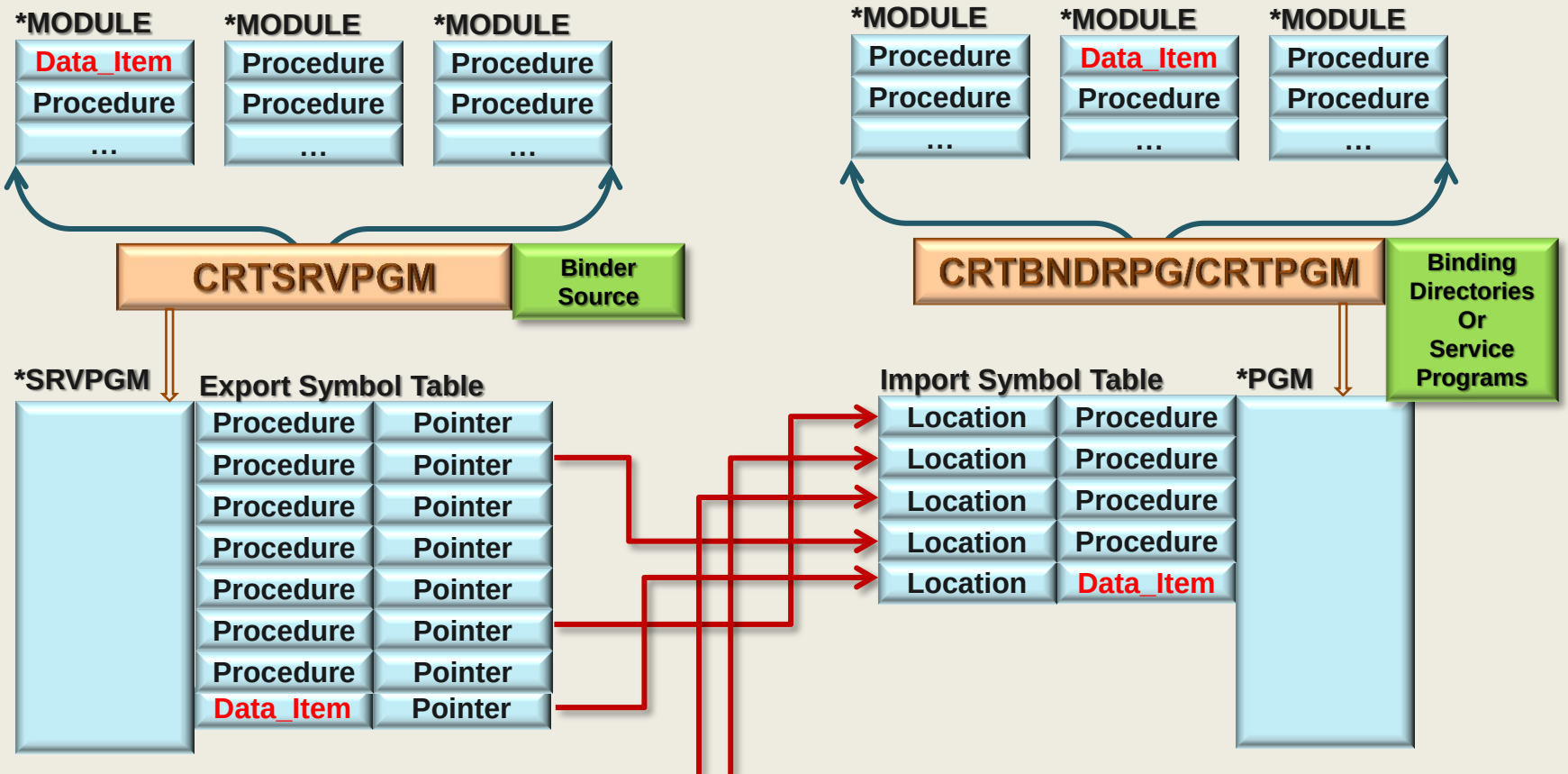
A Different Perspective on Service Programs

An activation group is a substructure of a job in which Integrated Language Environment (ILE) programs and service programs are activated.

This substructure contains the resources necessary to run the program.

These resources include: static and global program variables, dynamic storage, temporary data management resources, certain types of exception handlers and ending procedures.

Binding to Service Programs



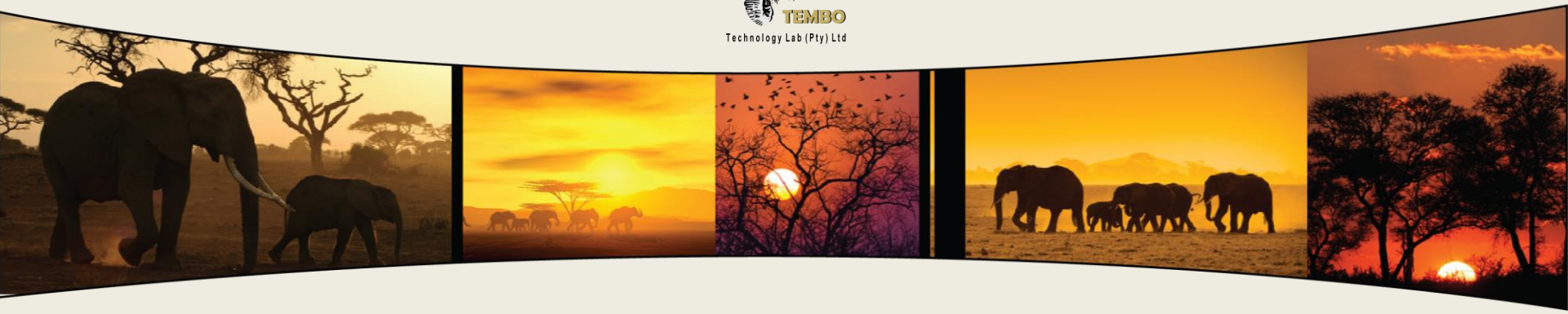

Andre Liebenberg Photography


JUST HANGIN' AROUND


© Ursula Celliers - Wildlife Photography


© Ursula Cedlers - Wildlife Photogra

TEMBO
Technology Lab (Pty) Ltd



Final Session

?? Q & A ??

- How do you pass a multiple key in the one field?
- If you are doing a dow loop on the pointers, do you use a returning indicator to signal eof?
- Can you go back through the DS used for the pointers and why?

- Is there a limit of open data paths that a job can have and is it tied to the *PAG and amount of available resources for that job.
- What happens if the job reaches that limit? Will the job end abnormally?

- In your experience, have you seen an average number of ODP's where it starts creating issues?
- Is there a way to check for these so that we know we have an issue either in the program or via another system job? I have read one article on using a custom QAQQINI file to cap the open data paths . I much rather we control properly via activation groups and in the programs.

- How to handle file Record Locks.
- Will record locks occur in the I/O Server Programs or Trigger Programs?

- Cascade Deletes.

  Can the deleted cascaded records be saved to a backup file?

  Can a trigger program be written to do this?

- Should we use CLLE programs now instead of CLP programs?
  - What is the difference?

- Can a parallel DDL can have a unique key while the DDS does not, and maintain the correct Level ID?

- How do we promote the Warehouse Dictionary after sanitizing it, to the main Averitt Dictionary?

- Is there a way to import 1 file to a schema that is already in AO?
- Is there a way to convert 1 file to ddl from dds in a schema without converting all the others with the build schema.

- How to send a message to the user in the application program that the "Record has Changed" when a record is updated thru the I/O Server Program?

- I am pretty sure we could do it from the I/O Server Program to the Application Program. A prototype parameter could be used. Other messages could be; "Record has been added' and 'Record has been Deleted'.

- Why are we getting the message that Commitment Control has to be started for a file that is journaled, when calling an I/O Server Program from the Command Line?

- Can you show us what naming conventions you use for your programs and files..(not for the session database you are using for the classes).