TATA CONSULTANCY SERVICES

TATA

Ab Initio

Basics Training Course

Experience certainty.

IT Services
Business Solutions
Outsourcing

Course Content

- Ab Initio Architecture
- Overview of Graph
- Ab Initio functions
- Basic components
- Partitioning and De-partitioning
- Case Studies

Course Objective

To understand the fundamentals of Ab Initio ETL.

Ab Initio Architecture

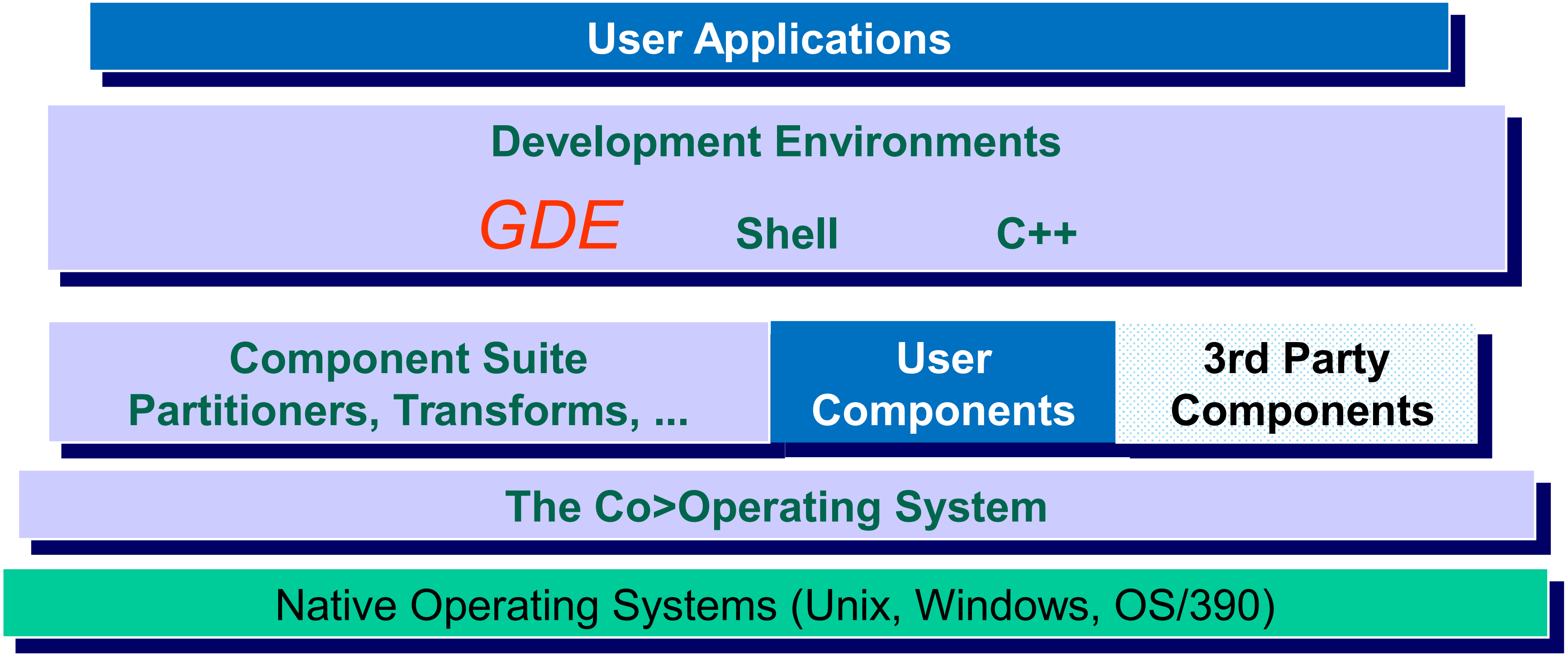
TATA CONSULTANCY SERVICES

July 6, 2010

Introduction

- Data processing tool from Ab Initio software corporation (http://www.abinitio.com)
- Latin for “from the beginning”
- Designed to support largest and most complex business applications
- Ab Initio software is a **general-purpose** data processing platform for **enterprise class**, **mission-critical** applications such as:
  - Data warehousing
  - Batch processing
  - Click-stream analysis
  - Data movement
  - Data transformation
- Graphical, intuitive, and “fits the way your business works” text

Ab Initio Product Architecture :



User Applications

Development Environments

GDE

Shell

C++

Component Suite
Partitioners, Transforms, ...

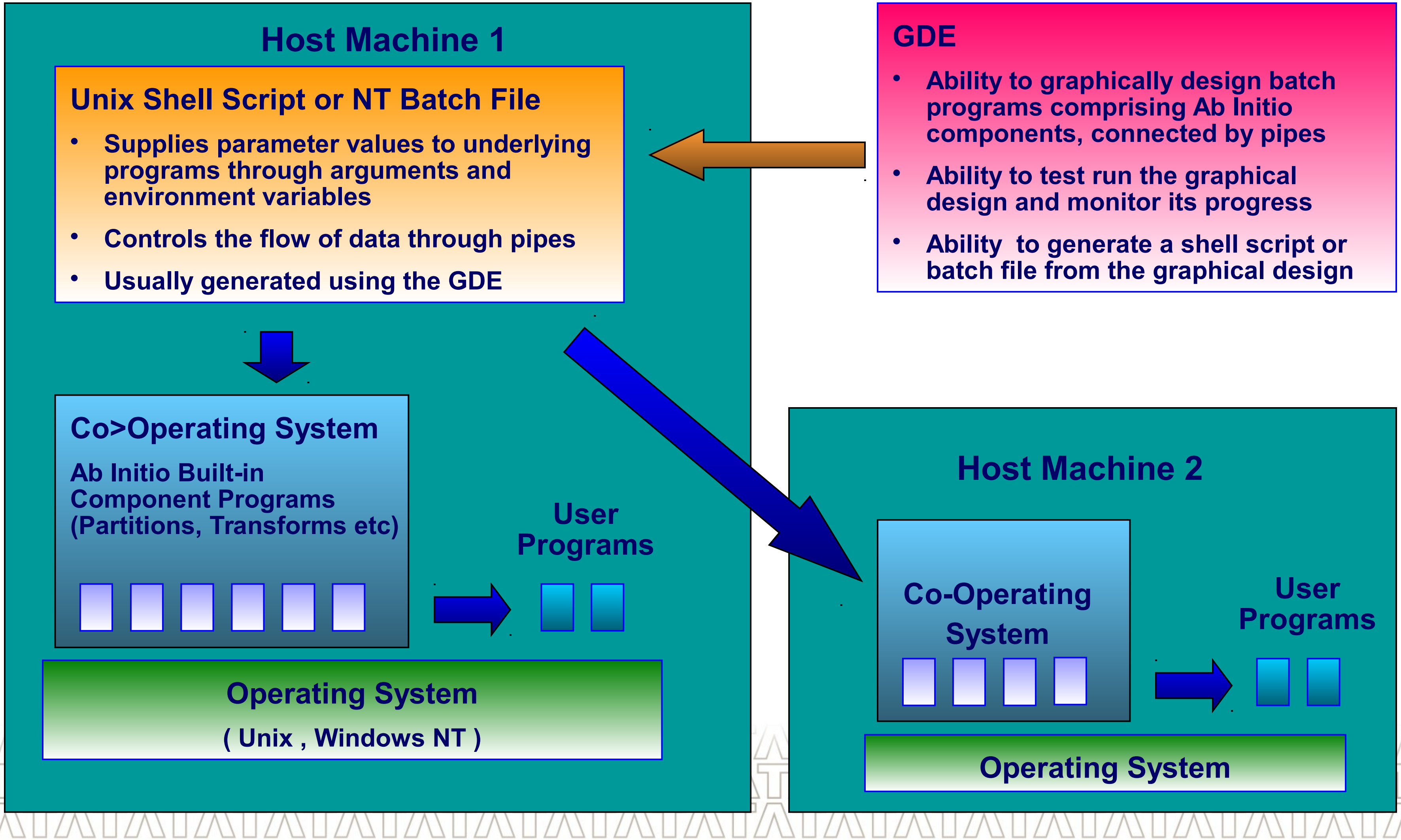
User
Components

3rd Party
Components

The Co>Operating System

Native Operating Systems (Unix, Windows, OS/390)

Client Server Communication



Host Machine 1

Unix Shell Script or NT Batch File

- Supplies parameter values to underlying programs through arguments and environment variables
- Controls the flow of data through pipes
- Usually generated using the GDE

Co>Operating System

Ab Initio Built-in Component Programs (Partitions, Transforms etc)

User Programs

Operating System

(Unix , Windows NT)

GDE

- Ability to graphically design batch programs comprising Ab Initio components, connected by pipes
- Ability to test run the graphical design and monitor its progress
- Ability to generate a shell script or batch file from the graphical design

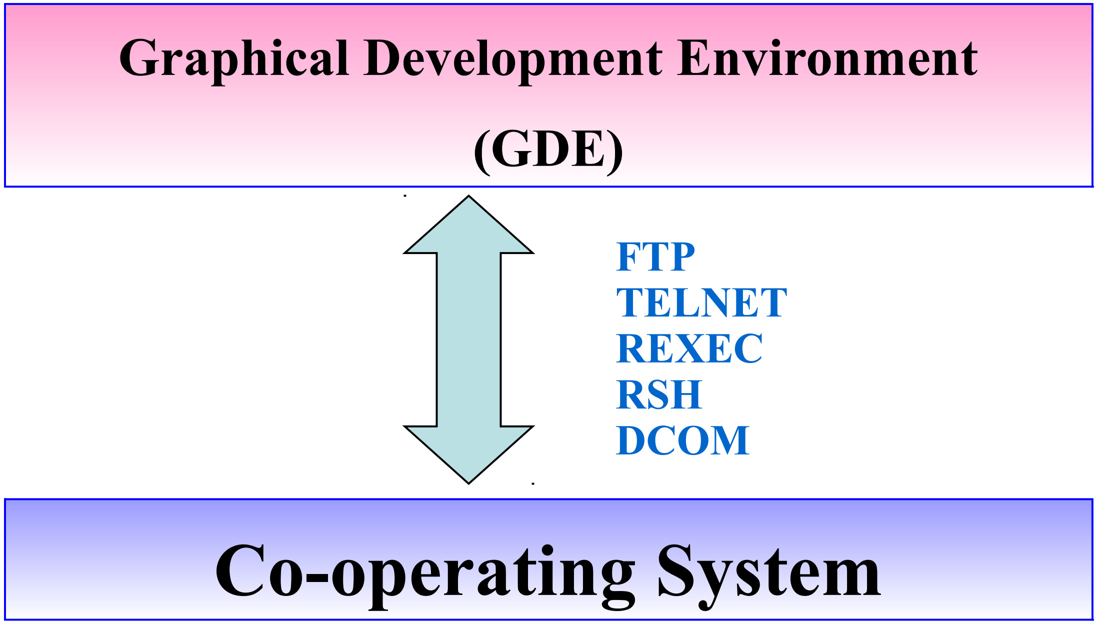
Host Machine 2

Co-Operating System

User Programs

Operating System

Ab Initio – Process Flow



On a typical installation, the Co-operating system is installed on a Unix or Windows NT server while the GDE is installed on a Pentium PC.

CO>Operating System

- Layered on the top of the operating system
- Unites a network of computing resources into a data-processing system with scalable performance
- Co>Operating system runs on …
  - Sun Solaris 2.6, 7, and 8 (SPARC)
  - IBM AIX 4.2, and 4.3
  - Hewlett-Packard HP-UX 10.20, 11.00, and 11.11
  - Siemens Pyramid Reliant UNIX Release 5.43
  - IBM DYNIX/ptx 4.4.6, 4.4.8, 4.5.1, and 4.5.2
  - Silicon Graphics IRIX 6.5
  - Red Hat Linux 6.2 and 7.0 (x86)
  - Windows NT 4.0 (x86) with SP 4, 5 or 6
  - Windows NT 2000 (x86) with no service pack or SP1
  - Digital UNIX V4.0D (Rev. 878) and 4.0E (Rev. 1091)
  - Compaq Tru64 UNIX Versions 4.0F (Rev 1229) and 5.1 (Rev 732)
  - IBM OS/390 Version 2.8, 2.9, and 2.10
  - NCR MP-RAS 3.02

Graphical Development Environment

The GDE …

- can talk to the Co-operating system using several protocols like Telnet, Ab Initio / Rexec and FTP
- GUI for building applications
- Co-operating system and GDE have independent release mechanisms
- Co-operating system upgrade is possible without change in the GDE release

Note: During deployment, GDE sets AB_COMPATIBILITY to the Co>Operating System version number. So, a change in the Co>Operating System release requires a re-deployment

Overview of Graph

The Graph Model

A Graph

- Logical modular unit of an application.
- Consists of several components that forms the building blocks of an Ab Initio application

A Component

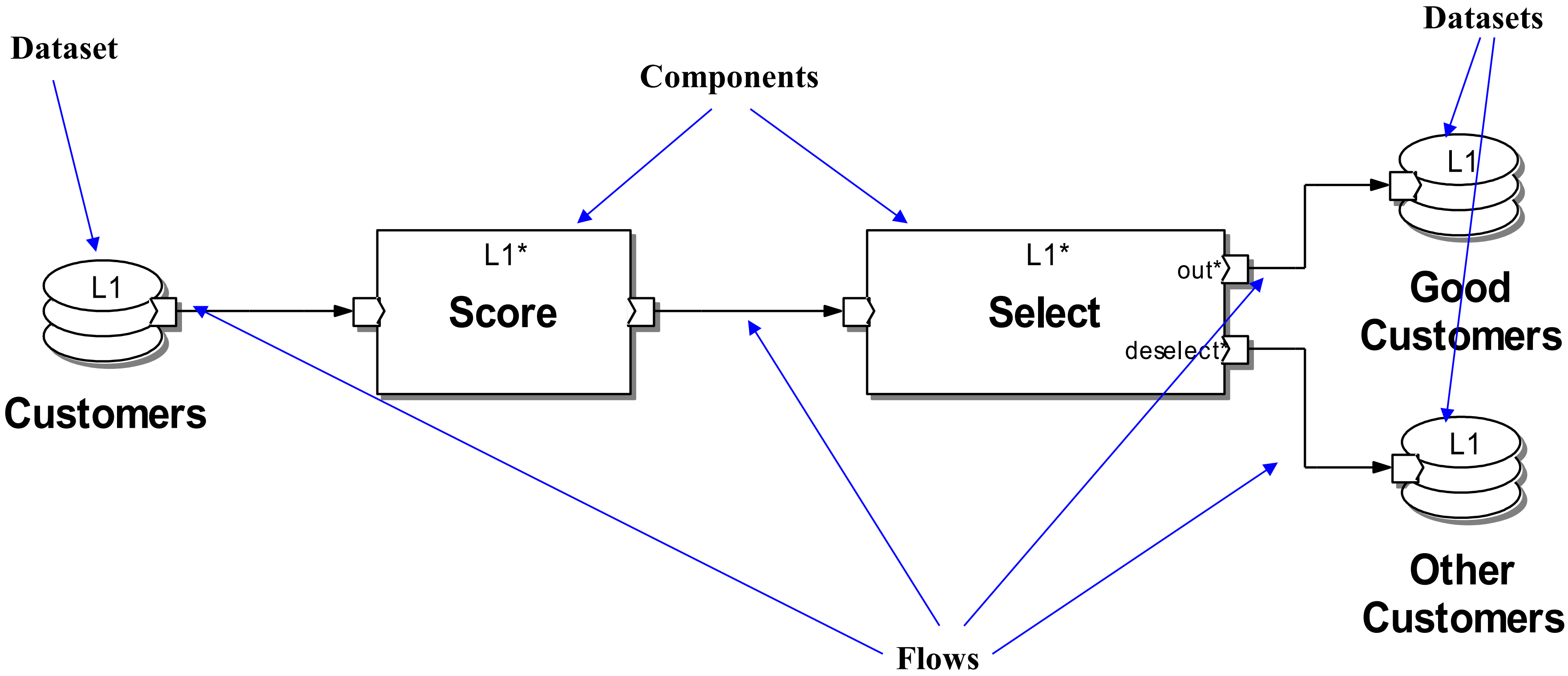
- A program that does a specific type of job controlled by its parameter settings

A Component Organizer

- Groups all components under different functional categories

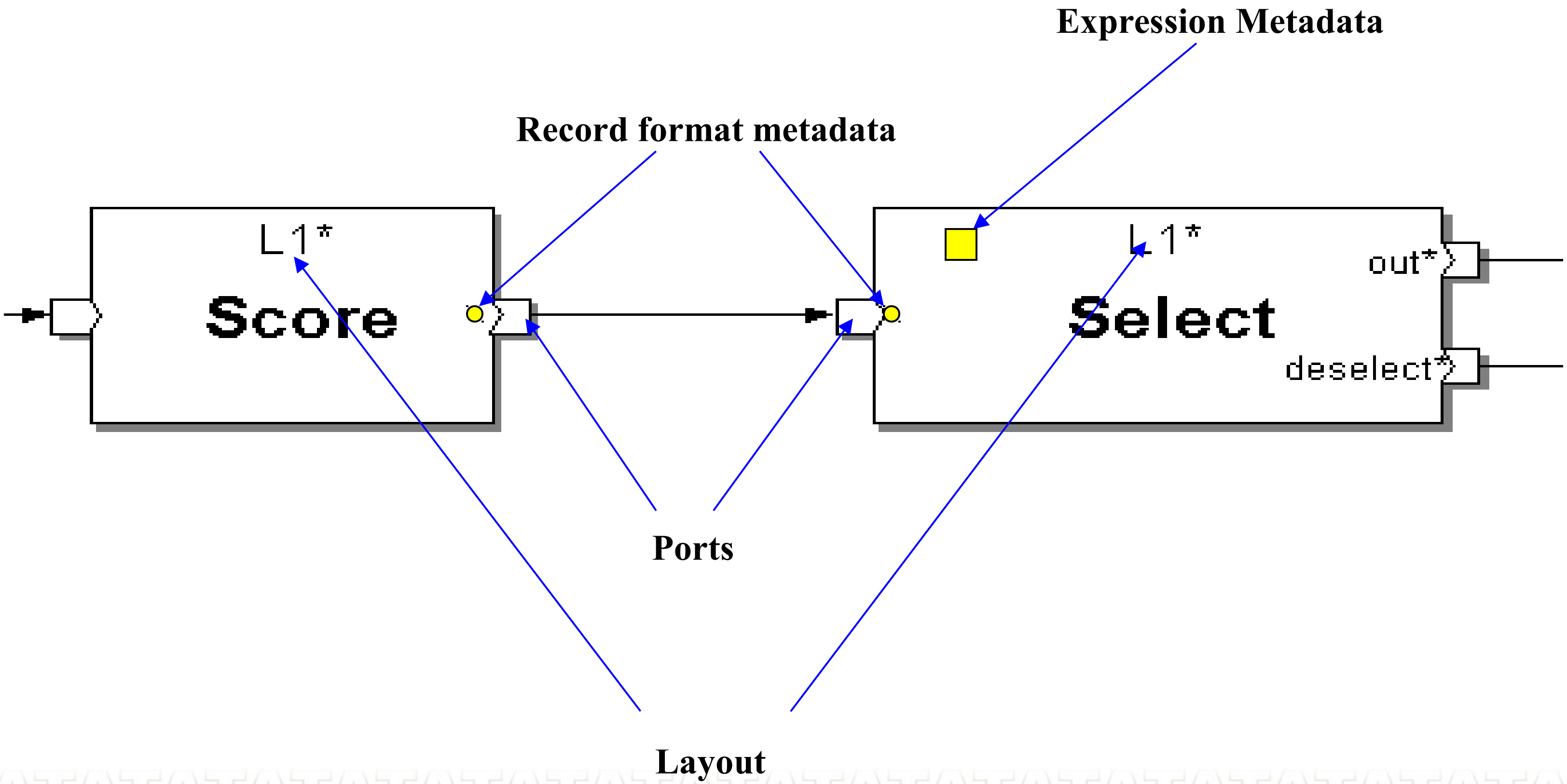
The Graph Model: Naming the pieces

A Sample Graph …



The Graph Model: A closer look

A Sample Graph …



Parts of typical graph

- **Datasets –** A table or a file which holds input or output data.
- **Meta Data –** Data about data.
- **Components –** Building blocks of a graph.
- **Flows –** Connectors by which 2 components are joined**.**
- **Layouts –** Defines which component will run where.
- **Start script –** A script which gets executed before the graph execution starts.
- **End script –** This script runs after the graph has completed running.
- **Host Profile –** A file containing values of the connection parameters with the host.

Types of Datasets

Datasets can be of following types:

- **Input Datasets**
  - itable – Input Table is used to unload/read data directly from a database table to the Abinitio graph as input
  - Input File – A data file acting as input to the Abinitio graph. Supports formats such as Flat files and XML files. These files can be serial or multi-file
- **Output Datasets**
  - otable – Output Table is used to load data directly into a database table
  - Output File – A data file acting as output of the Abinitio graph. Supports formats such as Flat files and XML files. These files can be serial or multi-file

- Databases connected as direct input/output are oracle, teradata, netezza, DB2, MS SQL, Red Brick, Sybase etc

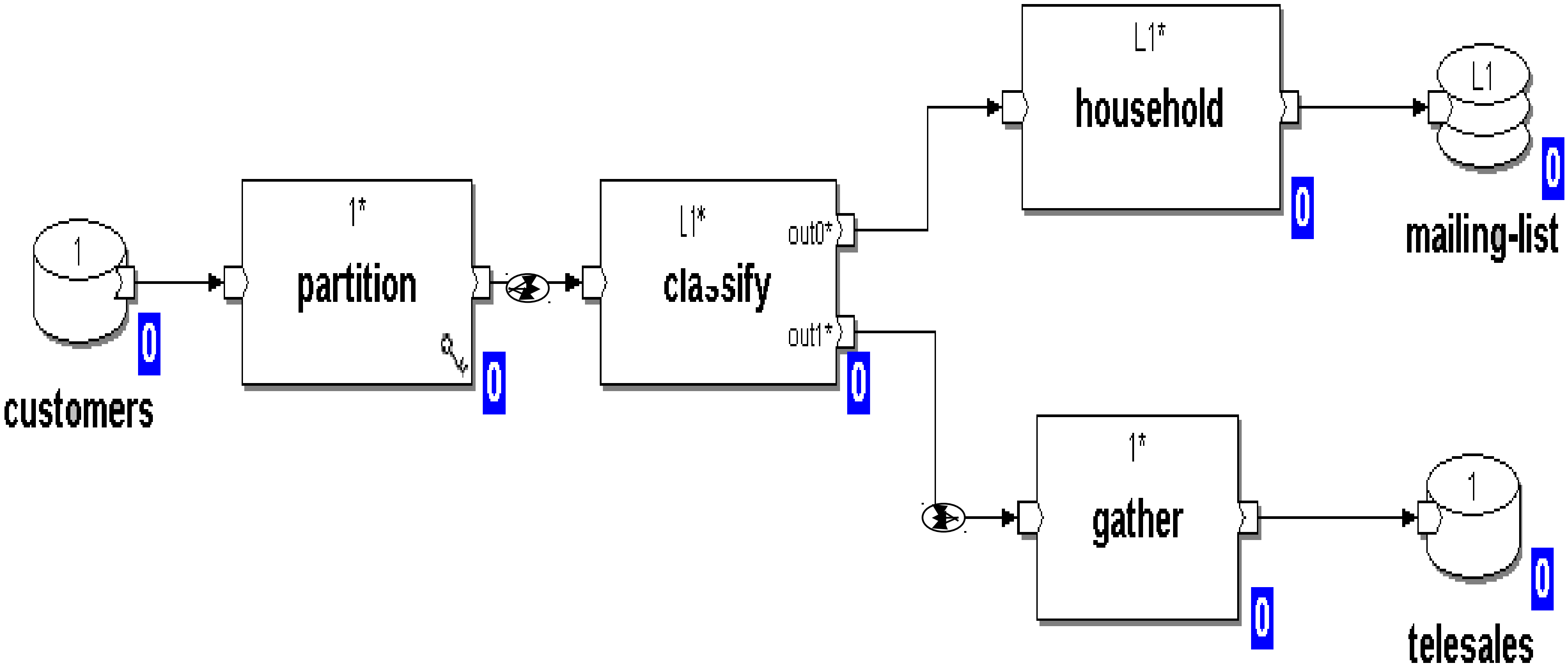
Structural Components of a Graph

- Start Script
  - Local to the Graph
- Setup Command
  - Ab Initio Host (AIH) file
  - Builds up the environment to run a graph
- Graph
- End Script
  - Local to the Graph

Runtime Environment

- The graph execution can be done from the GDE itself or from the back-end as well
- A graph can be deployed to the back-end server as a Unix shell script or Windows NT batch file.
- The deployed shell or the batch file can be executed at the back-end

A sample graph



Layout

1.Layout determines the location of a resource.

2.A layout is either serial or parallel.

3.A serial layout specifies one node and one directory.

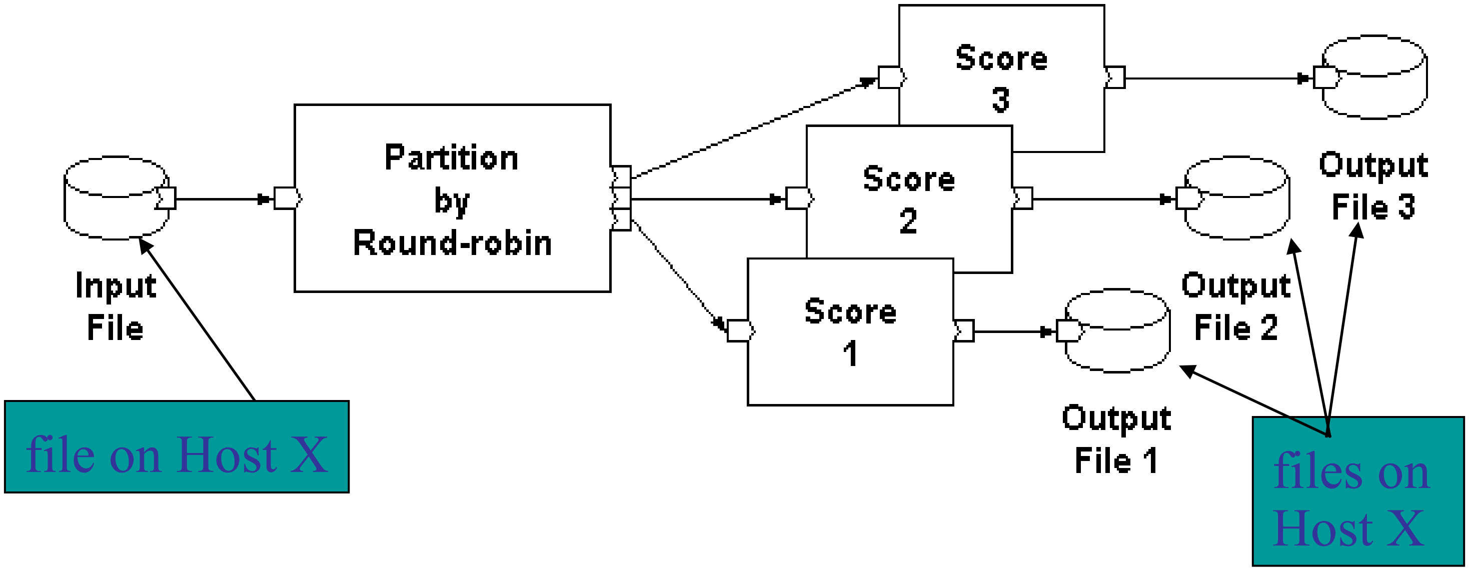
4.A parallel layout specifies multiple nodes and multiple directories. It is permissible for the same node to be repeated.

5.The location of a Dataset is one or more places on one or more disks.

6.The location of a computing component is one or more directories on one or more nodes. By default, the node and directory is unknown.

7.Computing components propagate their layouts from neighbors, unless specifically given a layout by the user.

Layout

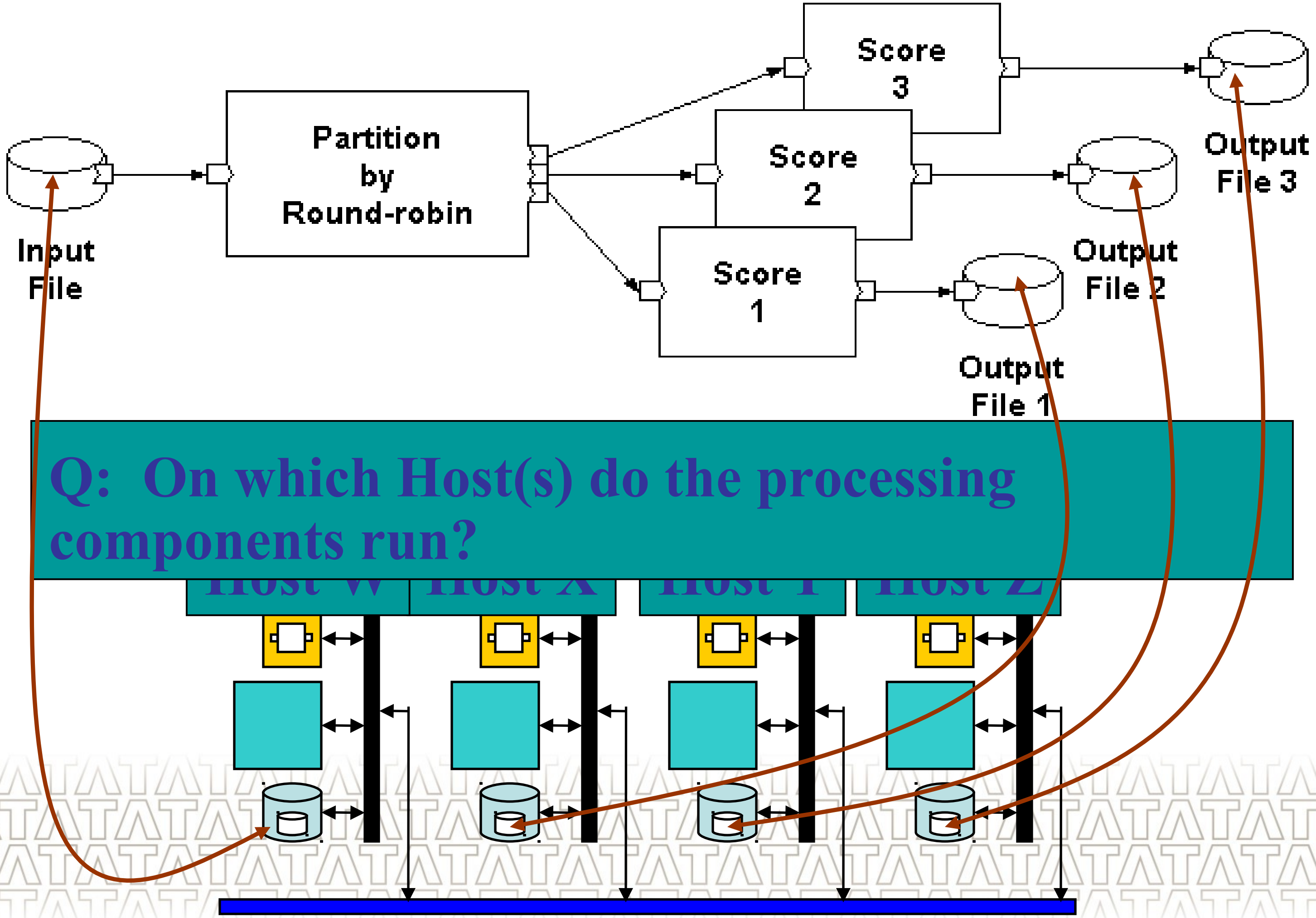


file on Host X

files on Host X

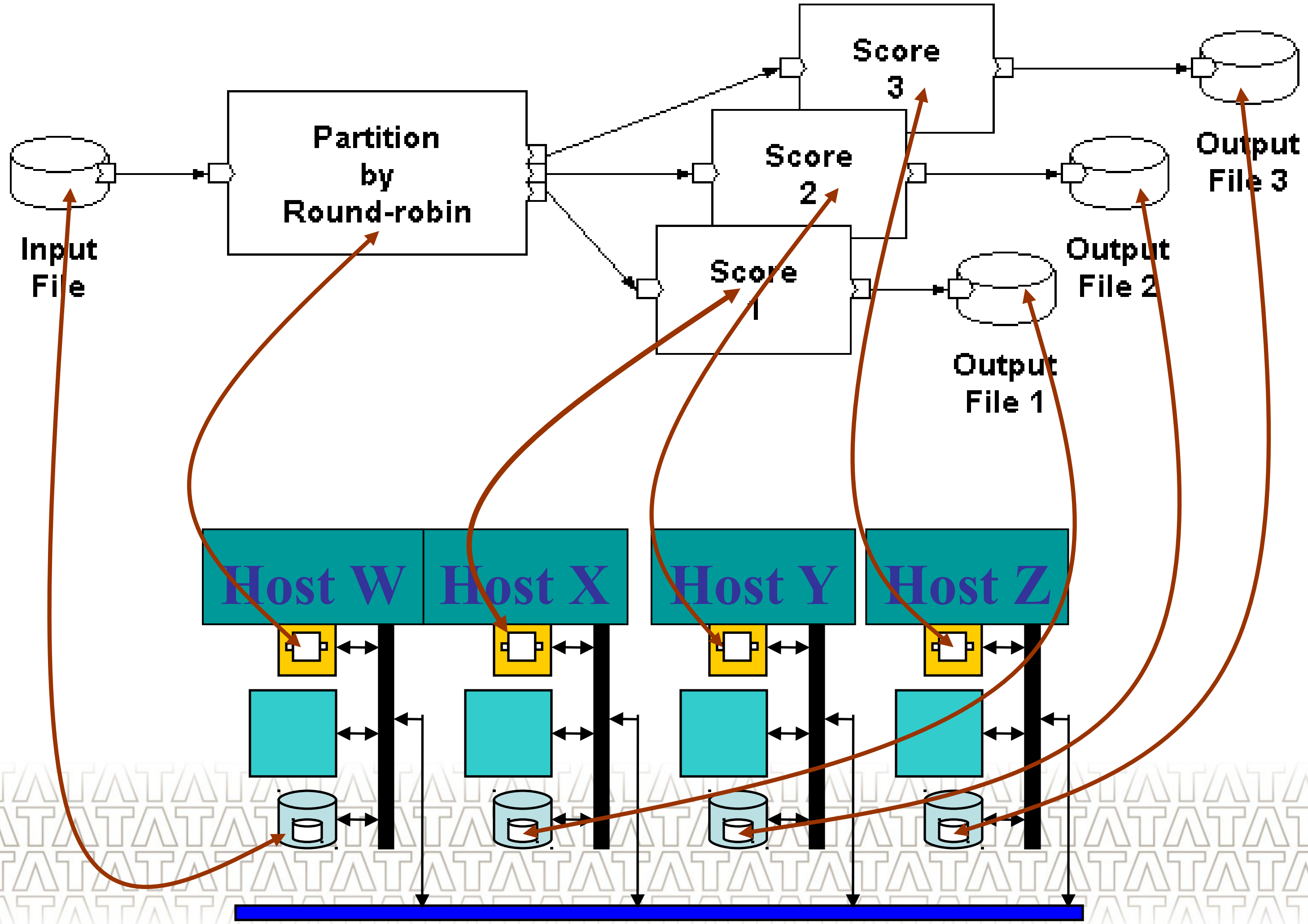
Q: On which host do the components run?
A: On Host X.

Layout Determines What Runs Where



Q: On which Host(s) do the processing components run?

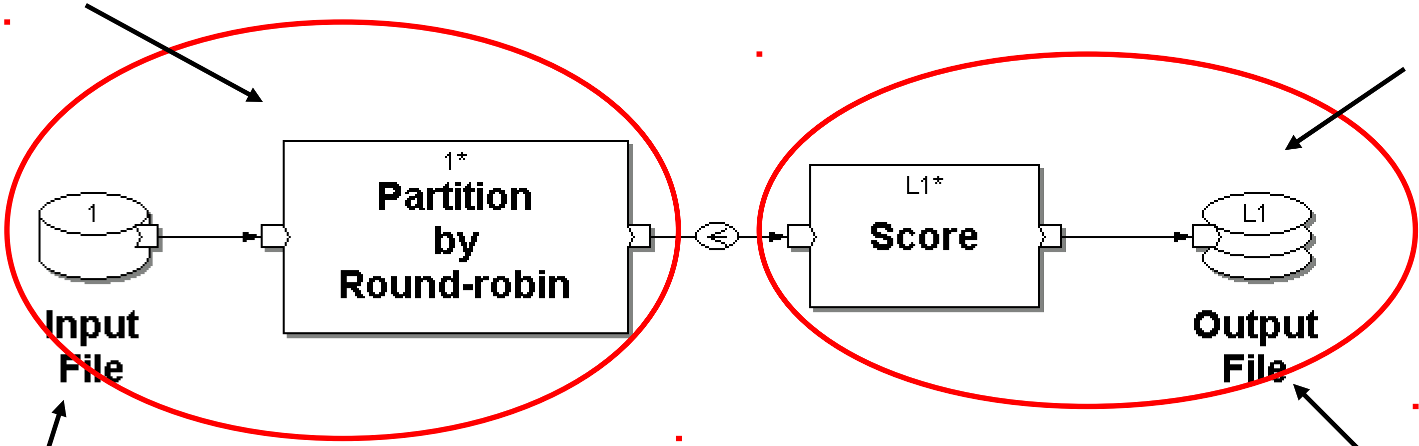
Layout Determines What Runs Where



Layout Determines What Runs Where

Serial

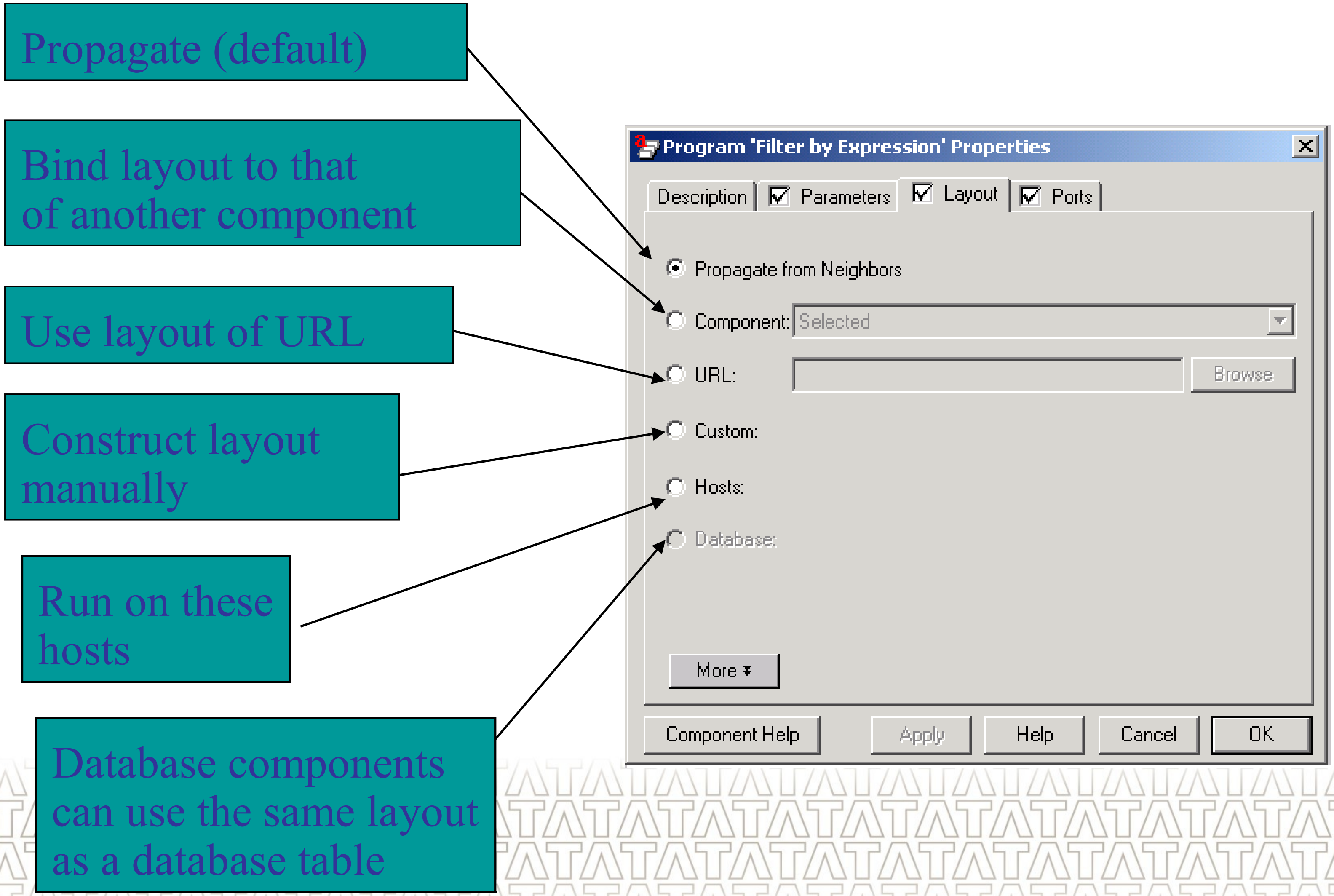
Parallel



file on Host W

3-way multifile on Hosts X,Y,Z

Controlling Layout



Propagate (default)

Bind layout to that of another component

Use layout of URL

Construct layout manually

Run on these hosts

Database components can use the same layout as a database table

Program 'Filter by Expression' Properties

Description | ☑ Parameters | ☑ Layout | ☑ Ports

● Propagate from Neighbors

○ Component: Selected

○ URL:

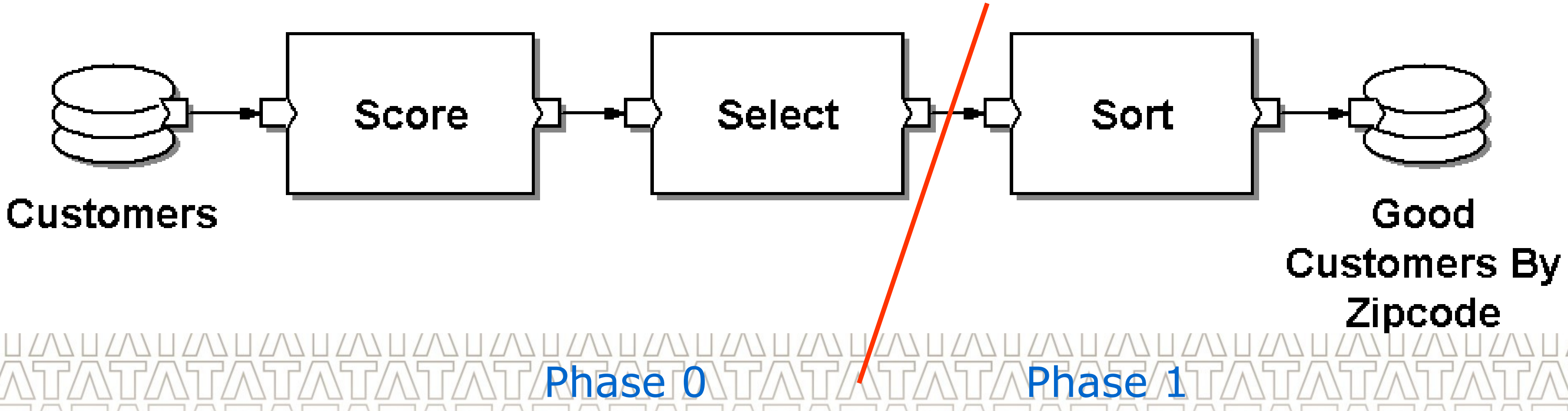
○ Custom:

○ Hosts:

○ Database:

Phase of a Graph

- Phases are used to break up a graph into blocks for performance tuning.
- Breaking an application into phases limits the contention for :
  - Main memory
  - Processors
- Breaking an application into phases costs: Disk Space
- The temporary files created by phasing are deleted at the end of the phase, regardless of whether the run was successful.



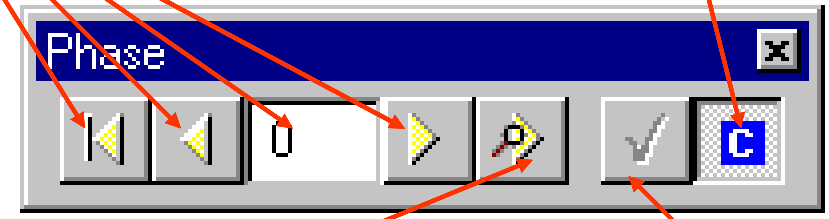
Checkpoint & Recovery

- A checkpoint is a point at which the Co>Operating System saves all the information it would need to restore a job to its state at that point. In case of failure, you can recover completed phases of a job up to the last completed checkpoint.

- Only as each new checkpoint is completed successfully are the temporary files corresponding to the previous checkpoint deleted.

- Any Phase Break can be a checkpoint.

The Phase Toolbar



A Toggle between:
Phase (P), and Checkpoint After Phase (C)

Select Phase Number

View Phase

Set Phase

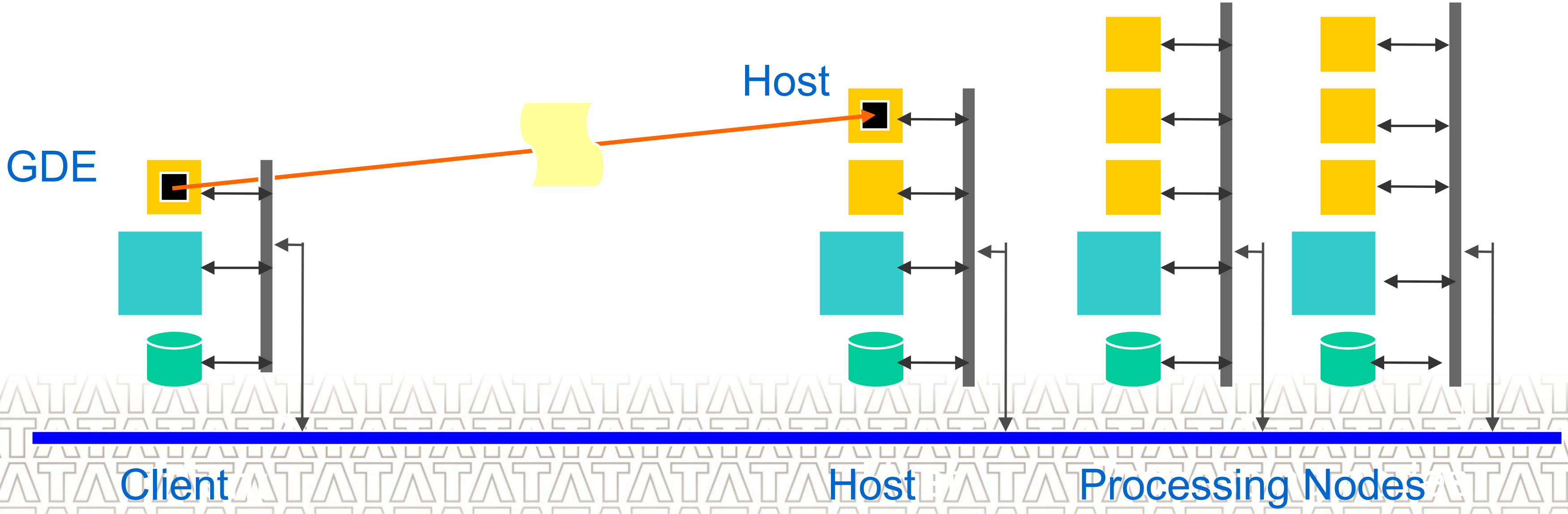
Anatomy of a Running Job

What happens when you push the "Run" button?

- Your graph is translated into a script that can be executed in the Shell Development Environment.
- This script and any metadata files stored on the GDE client machine are shipped (via FTP) to the server.
  - The script is invoked (via REXEC or TELNET) on the server.
  - The script creates and runs a job that may run across many nodes.
  - Monitoring information is sent back to the GDE client.

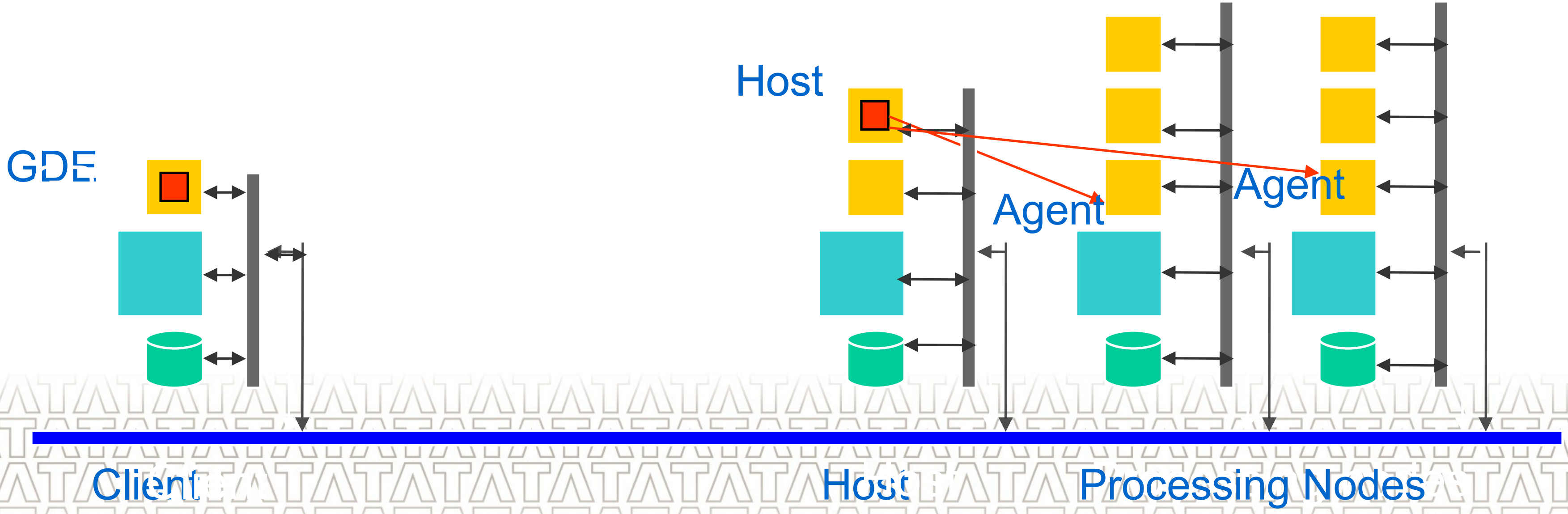
Anatomy of a Running Job

- **Host Process Creation**
  - Pushing "Run" button generates script.
  - Script is transmitted to Host node.
  - Script is invoked, creating Host process.



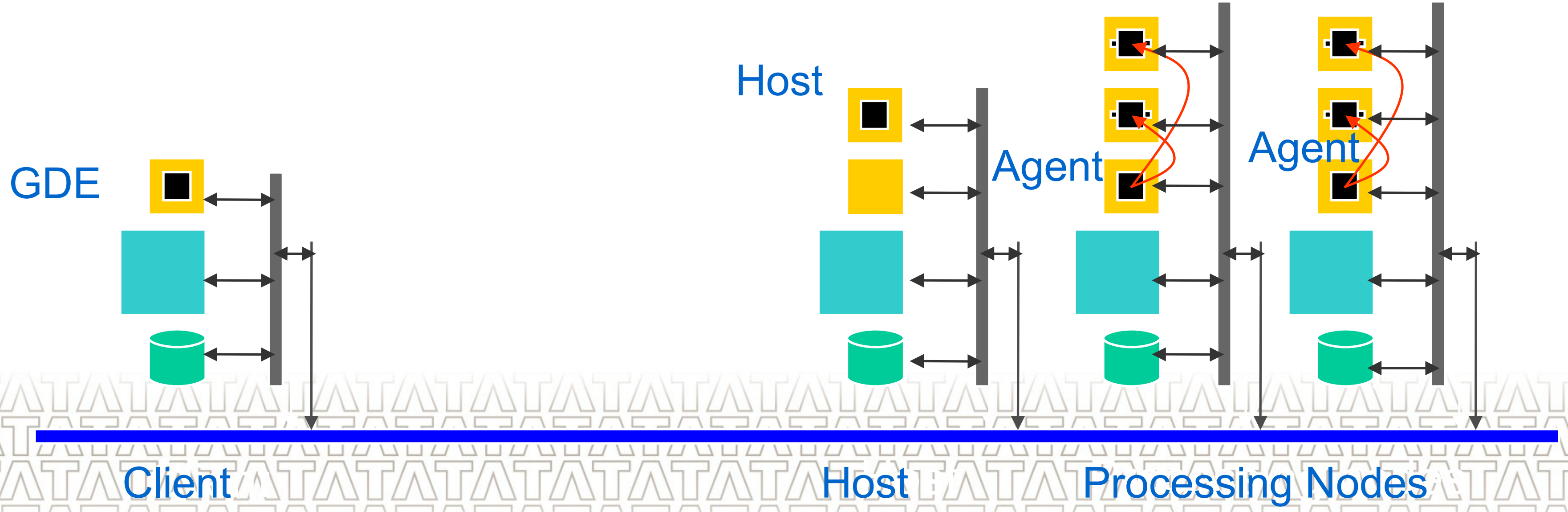
Anatomy of a Running Job

- **Agent Process Creation**
  - Host process spawns Agent processes.



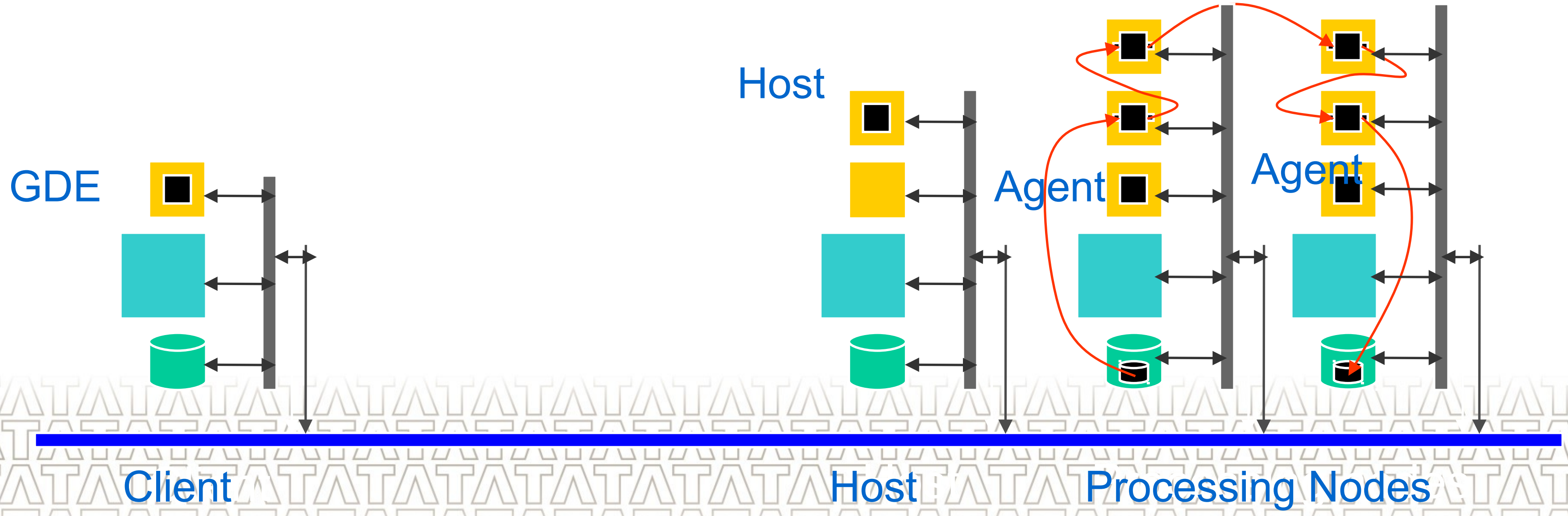
Anatomy of a Running Job

- **Component Process Creation**
  - Agent processes create Component processes on each processing node.



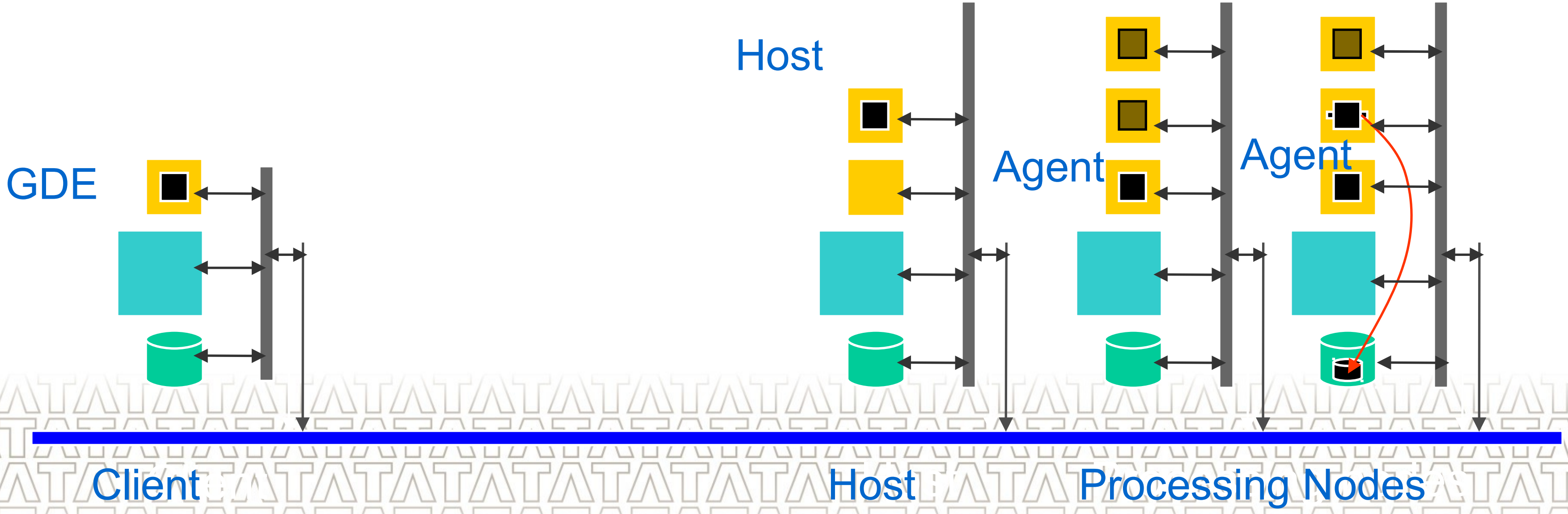
Anatomy of a Running Job

- **Component Execution**
  - Component processes do their jobs.
  - Component processes communicate directly with datasets and each other to move data around.



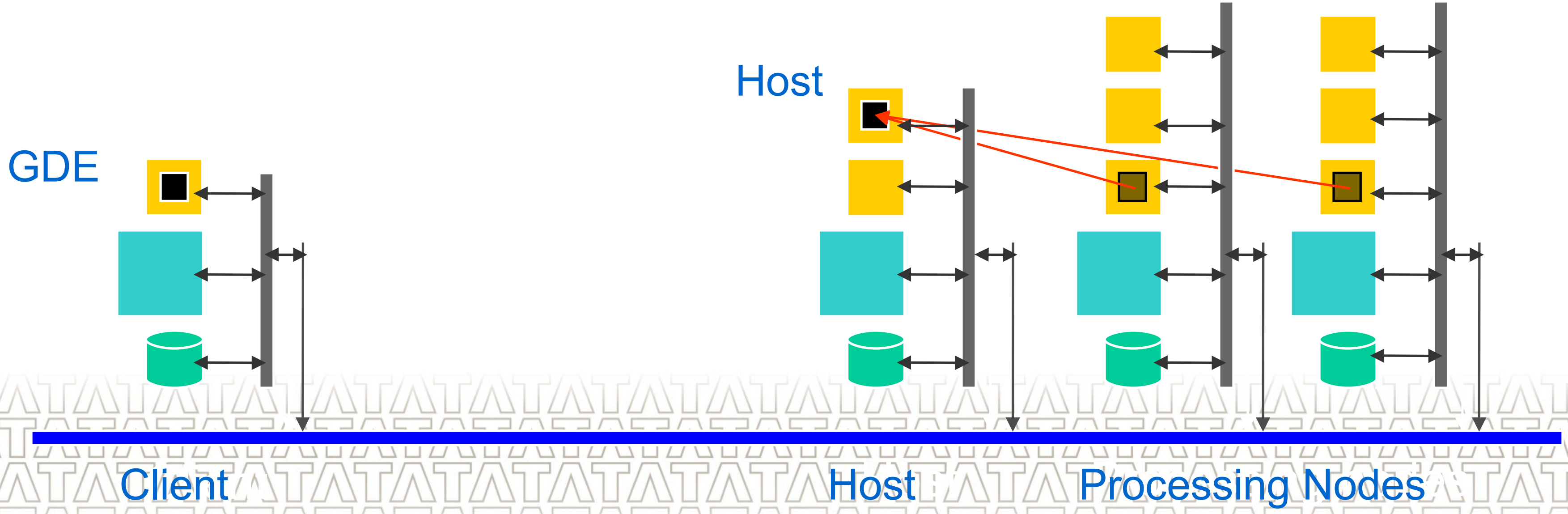
Anatomy of a Running Job

- **Successful Component Termination**
  - As each Component process finishes with its data, it exits with success status.



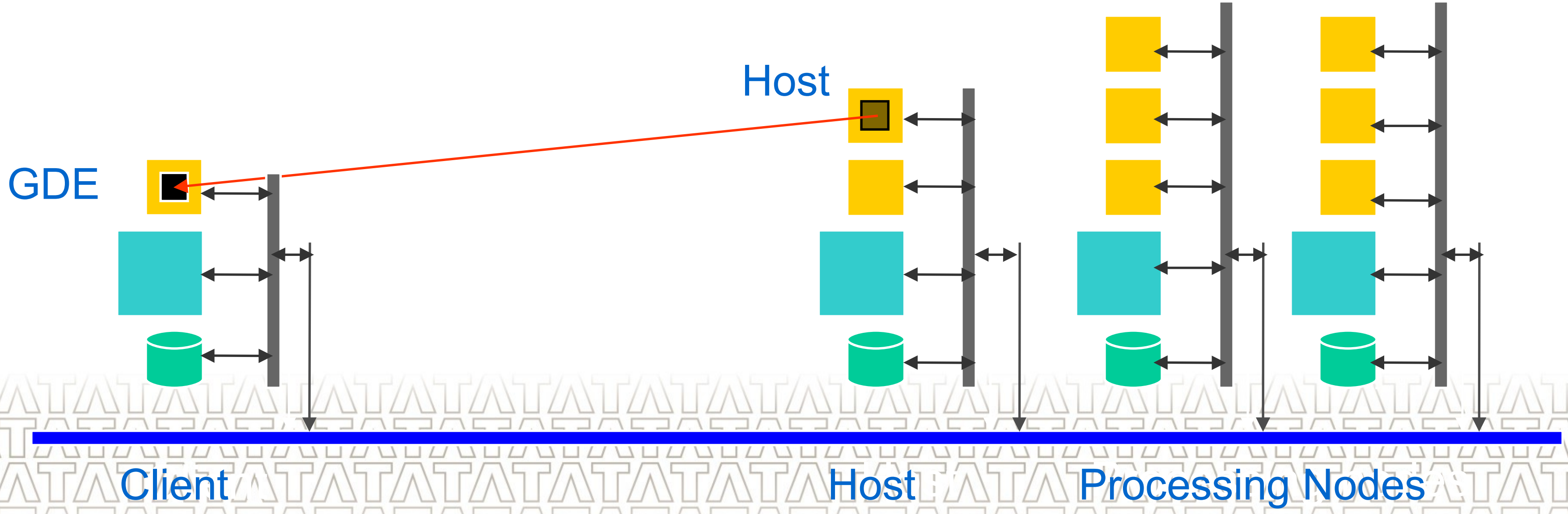
Anatomy of a Running Job

- **Agent Termination**
  - When all of an Agent’s Component processes exit, the Agent informs the Host process that those components are finished.
  - The Agent process then exits.



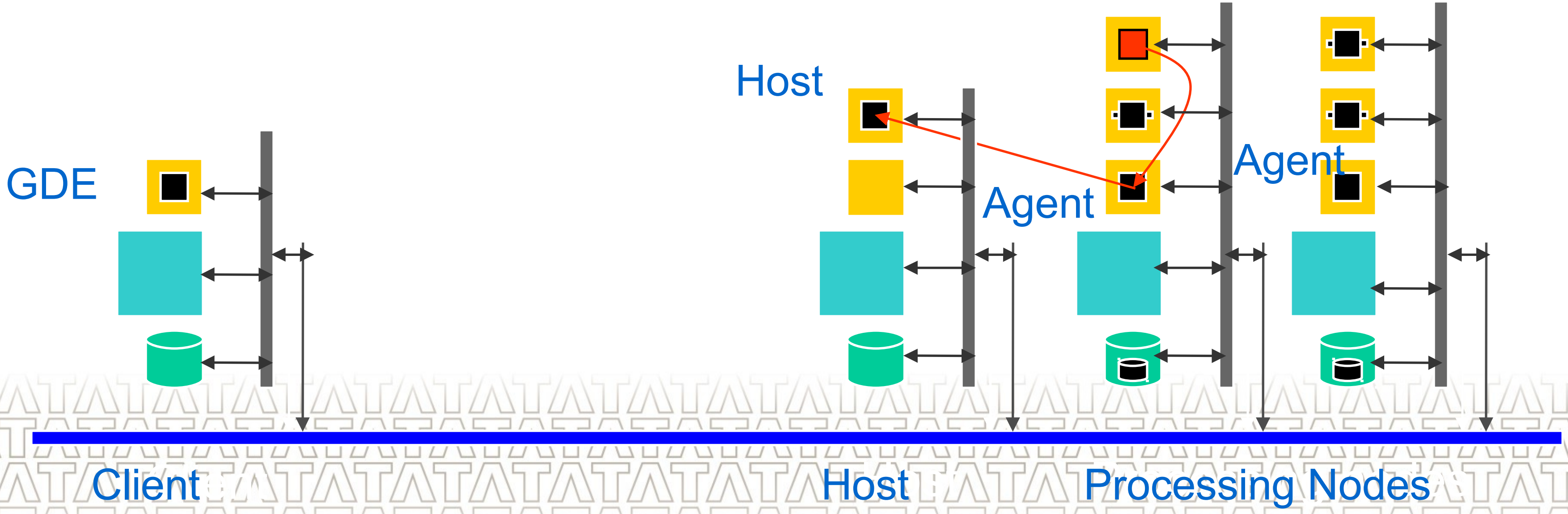
Anatomy of a Running Job

- **Host Termination**
  - When all Agents have exited, the Host process informs the GDE that the job is complete.
  - The Host process then exits.



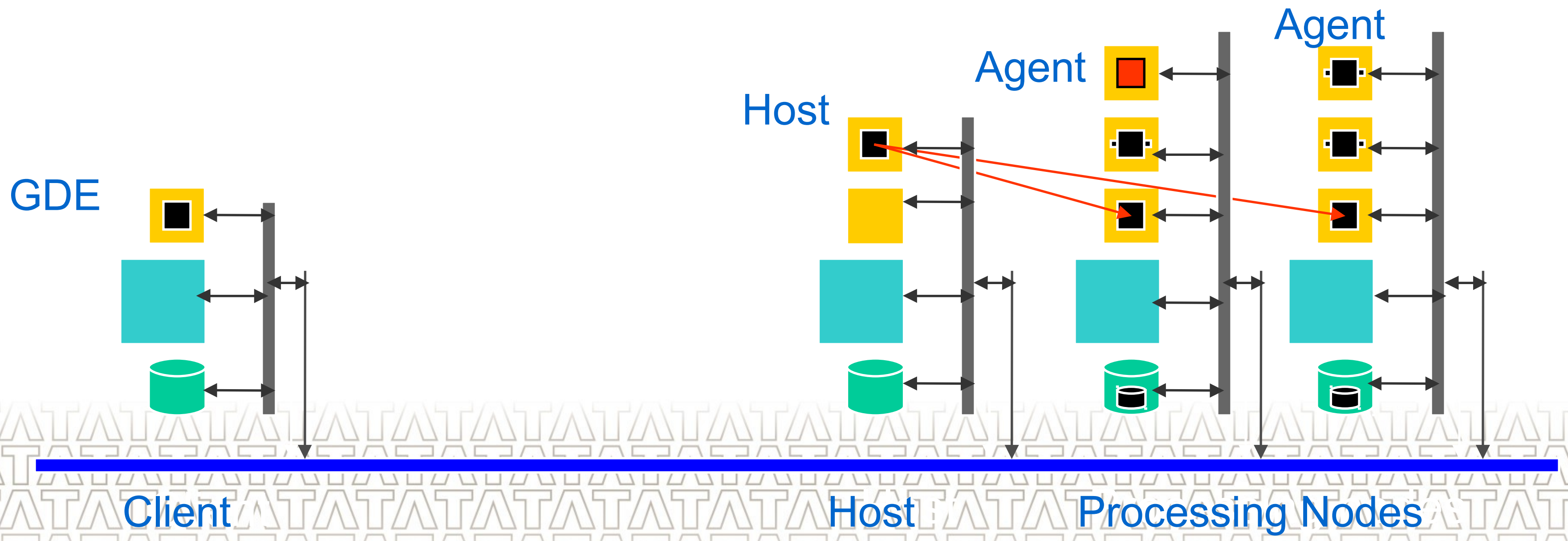
Anatomy of a Running Job

- **Abnormal Component Termination**
  - When an error occurs in a Component process, it exits with error status.
  - The Agent then informs the Host.



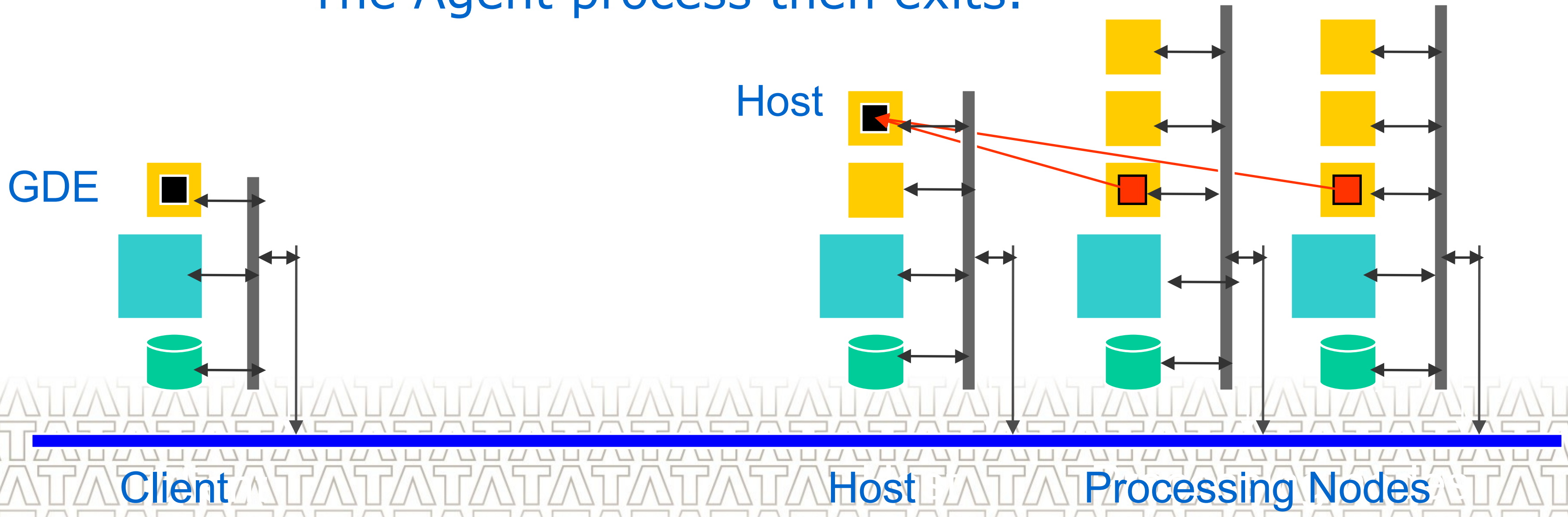
Anatomy of a Running Job

- **Abnormal Component Termination**
  - The Host tells each Agent to kill its Component processes.



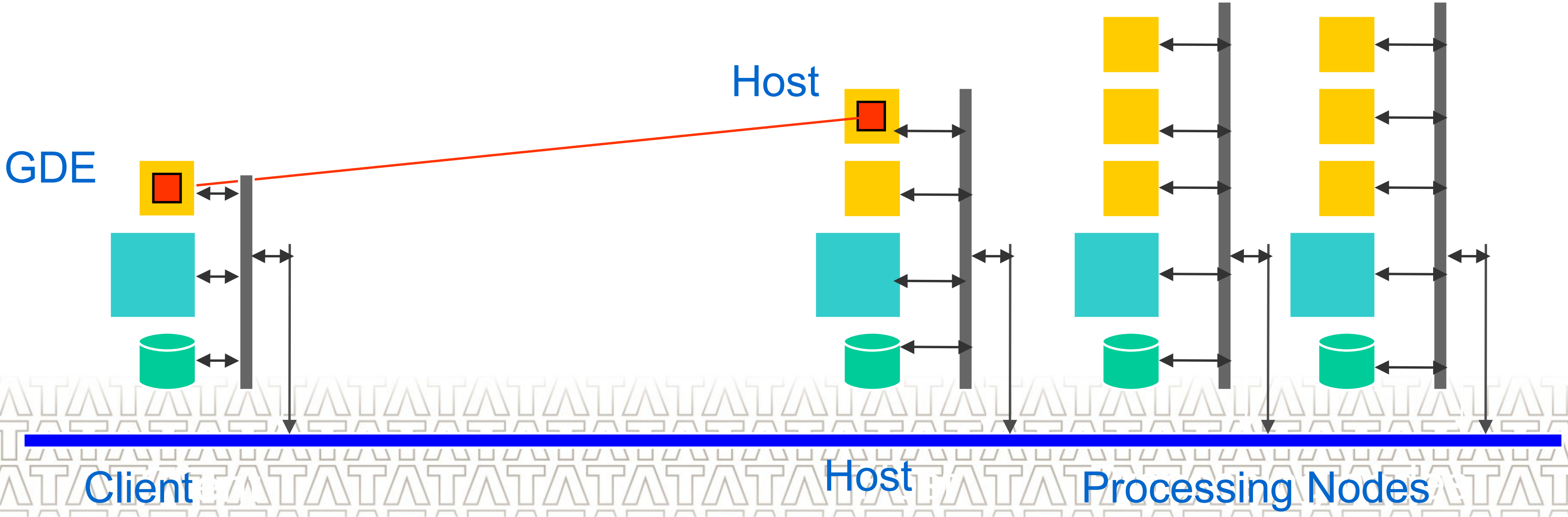
Anatomy of a Running Job

- **Agent Termination**
  - When every Component process of an Agent have been killed, the Agent informs the Host process that those components are finished.
  - The Agent process then exits.



Anatomy of a Running Job

- **Host Termination**
  - When all Agents have exited, the Host process informs the GDE that the job failed.
  - The Host process then exits.



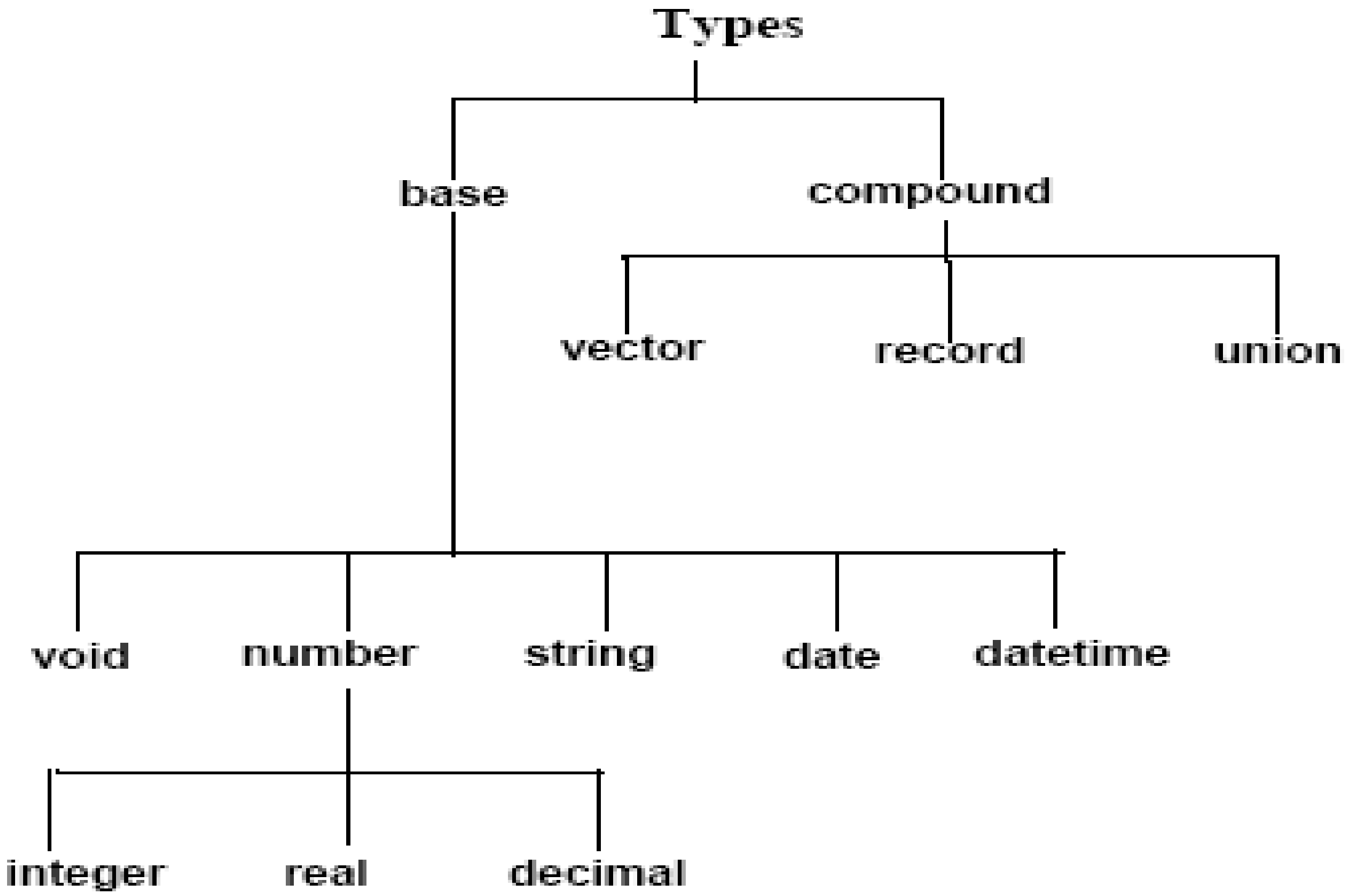
Ab Initio Functions

TATA CONSULTANCY SERVICES

July 6, 2010

DML(Data Manipulation Language)

- DML provides different set of data types including Base,Compound as well as User-defined data types



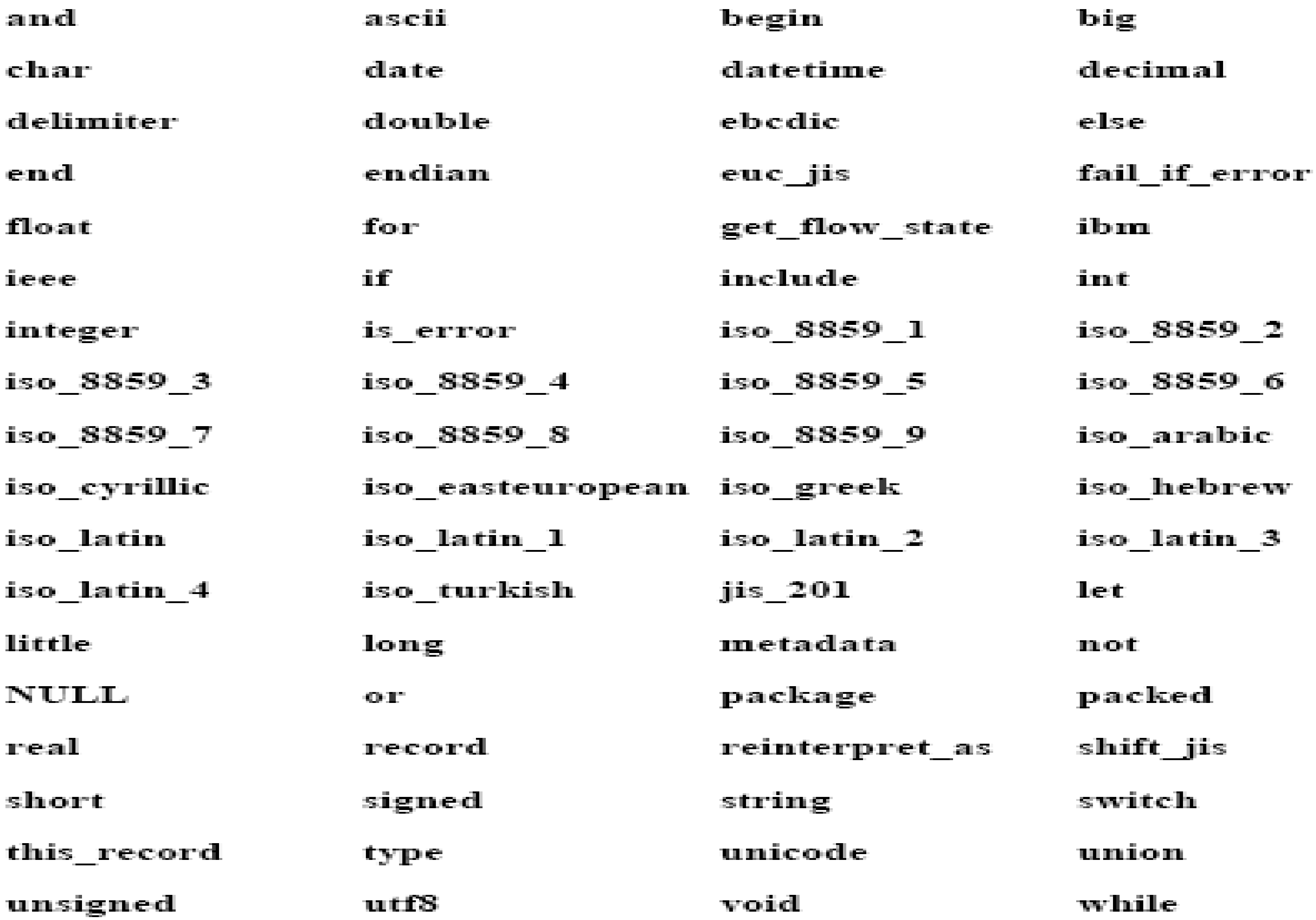
Data Manipulation Language or DML

DML Syntax :

- Record types begin with **record** and end with **end**
- Fields are declared: data_type(length) field_name; (fixed length DML)

  or data_type(delimiter) field_name; (delimited DML)
- Field names consist of **letters(a...z,A...Z), digits(0...9), underscores(_)** and are **Case sensitive**
- **Keywords/Reserved words** cannot be used as field names.

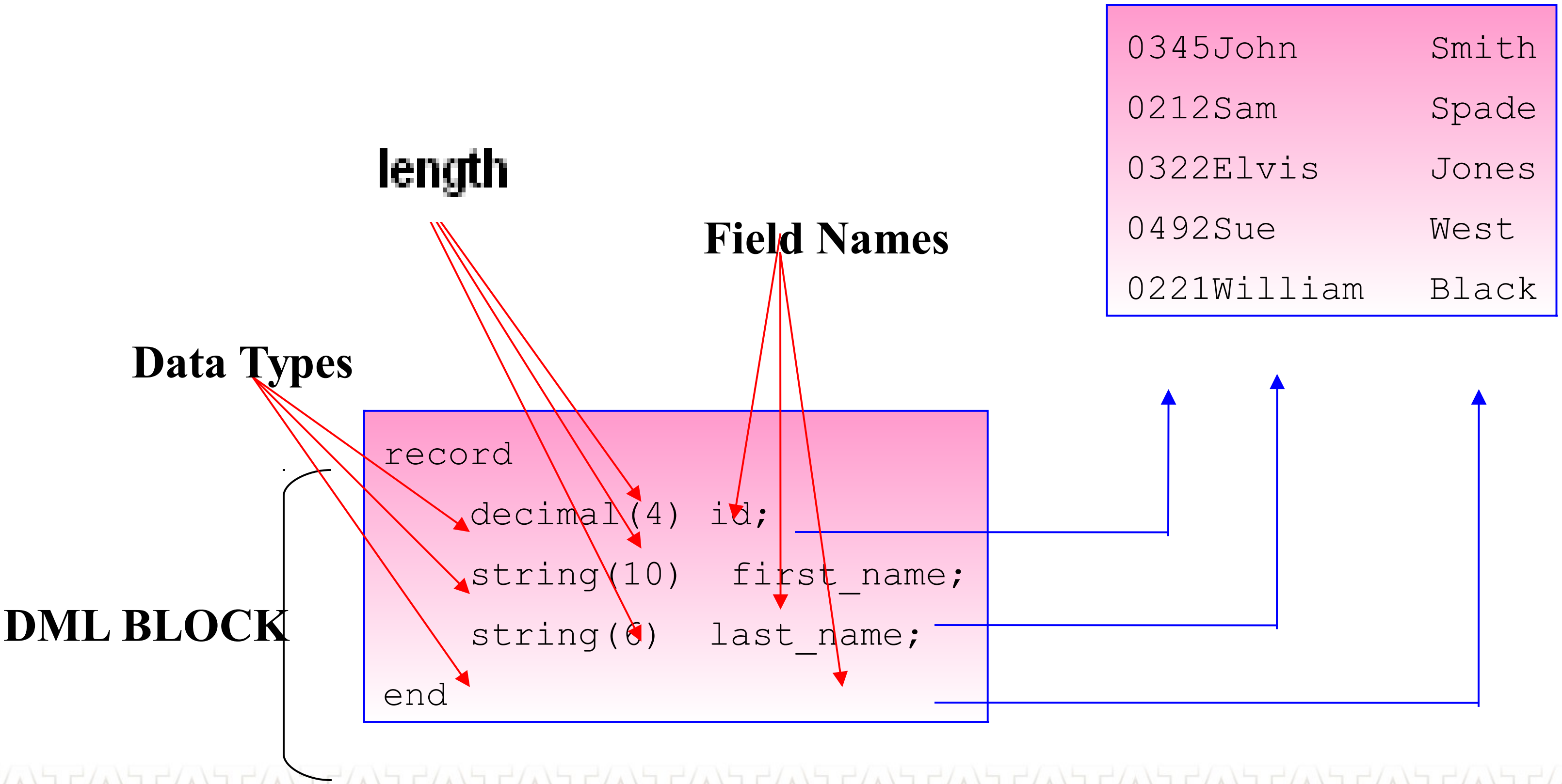
Keywords/Reserved Words



and
ascii
begin
big
char
date
datetime
decimal
delimiter
double
ebcdic
else
end
endian
euc_jis
fail_if_error
float
for
get_flow_state
ibm
ieee
if
include
int
integer
is_error
iso_8859_1
iso_8859_2
iso_8859_3
iso_8859_4
iso_8859_5
iso_8859_6
iso_8859_7
iso_8859_8
iso_8859_9
iso_arabic
iso_cyrillic
iso_easteuropean
iso_greek
iso_hebrew
iso_latin
iso_latin_1
iso_latin_2
iso_latin_3
iso_latin_4
iso_turkish
jis_201
let
little
long
metadata
not
NULL
or
package
packed
real
record
reinterpret_as
shift_jis
short
signed
string
switch
this_record
type
unicode
union
unsigned
utf8
void
while

Data Manipulation Language or DML

Record Format Metadata in DML



length

Field Names

Data Types

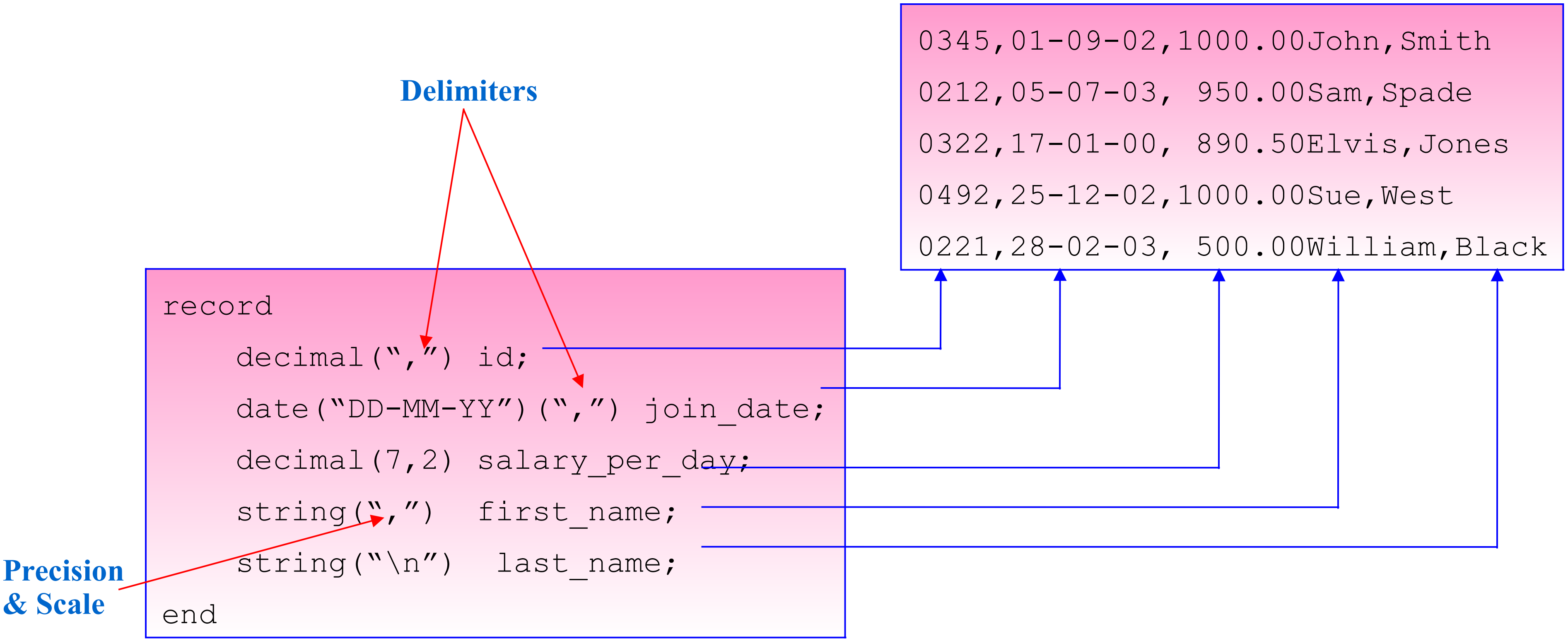
DML BLOCK

```
record
    decimal(4)  id;
    string(10)   first_name;
    string(6)   last_name;
end
```

```
0345John      Smith
0212Sam       Spade
0322Elvis     Jones
0492Sue       West
0221William   Black
```

Data Manipulation Language or DML

More DML Types



Delimiters

```
0345,01-09-02,1000.00John,Smith
0212,05-07-03, 950.00Sam,Spade
0322,17-01-00, 890.50Elvis,Jones
0492,25-12-02,1000.00Sue,West
0221,28-02-03, 500.00William,Black
```

```
record
    decimal(“,”) id;
    date(“DD-MM-YY”)(“,”) join_date;
    decimal(7,2) salary_per_day;
    string(“,”)  first_name;
    string(“\n”)  last_name;
end
```

Precision & Scale

Built-in Functions

Ab Initio built-in functions are DML expressions that

- can manipulate strings, dates, and numbers
- access system properties

Function categories

- Date functions
- Inquiry and error functions
- Lookup functions
- Math functions
- Miscellaneous functions
- String functions

Date Functions

- date_day
- date_day_of_month
- date_day_of_week
- date_day_of_year
- date_month
- date_month_end
- date_to_int
- date_year
- datetime_add
- datetime_day
- datetime_day_of_month
- datetime_day_of_week
- datetime_day_of_year
- datetime_difference
- datetime_hour
- datetime_minute
- datetime_second
- datetime_microsecond
- datetime_month
- datetime_year

Inquiry and Error Functions

- fail_if_error
- force_error
- is_error
- is_null
- length_of
- write_to_log
- first_defined
- is_defined
- is_failure
- is_valid
- size_of
- write_to_log_file

Lookup Functions

- lookup
- lookup_count
- lookup_local
- lookup_count_local
- lookup_match
- lookup_next
- lookup_next_local

Math Functions

- Ceiling
- decimal_round
- decimal_round_down
- decimal_round_up
- Floor
- decimal_truncate
- math_abs
- math_acos
- math_asin
- math_atan
- math_cos
- math_cosh
- math_exp
- math_finite
- math_log
- math_log10
- math_tan
- math_pow
- math_sin
- math_sinh
- math_sqrt
- math_tanh

Miscellaneous Functions

- allocate
- ddl_name_to_dml_name
- ddl_to_dml
- hash_value
- next_in_sequence
- number_of_partitions
- printf
- Random
- raw_data_concat
- raw_data_substring
- scanf_float
- scanf_int
- scanf_string
- sleep_for_microseconds
- this_partition
- translate_bytes
- unpack_nibbles

String Functions

- char_string
- decimal_lpad
- decimal_lrepad
- decimal_strip
- is_blank
- is_bzero
- re_index
- re_replace
- string_char
- string_compare
- string_concat
- string_downcase
- string_filter
- string_lpad
- string_length
- string_upcase
- string_trim
- string_substring
- re_replace_first
- string_replace_first
- string_pad
- string_ltrim
- string_lrtrim

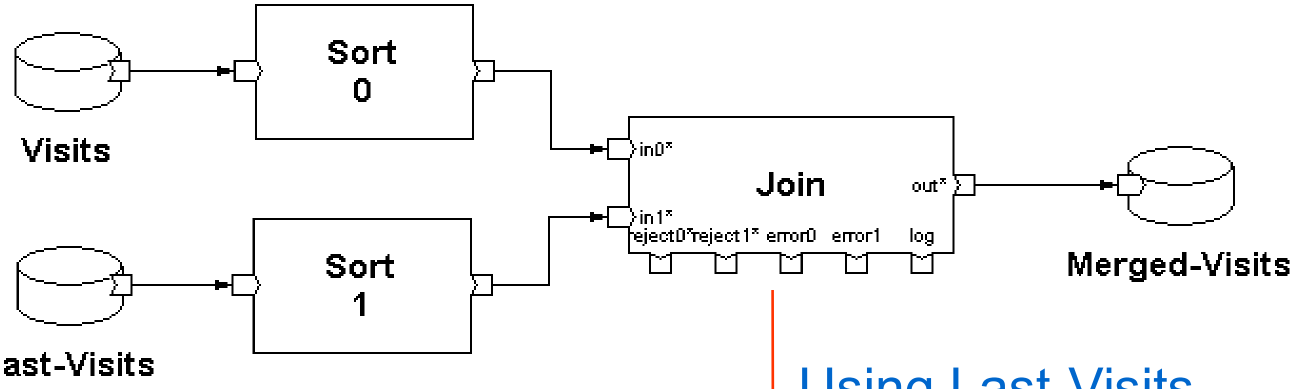
Lookup File

- Represents one or more Serial or Multifile
- The file you want to use as a Lookup must fit into main memory
- This allows a transform function to retrieve records much more quickly than it could retrieve them if they were stored on disk
- Lookup File associates key values with corresponding data values to index records and retrieve them
- **Lookup parameters:**
  - **Key:** Name of the key fields against which Lookup File matches its arguments
  - **Record Format:** The record format you want Lookup File to use when returning data records
- We use **Lookup functions** to call Lookup Files where the first argument to these lookup functions is the “name of the Lookup File”. The remaining arguments are values to be matched against the fields named by the **key parameter**.

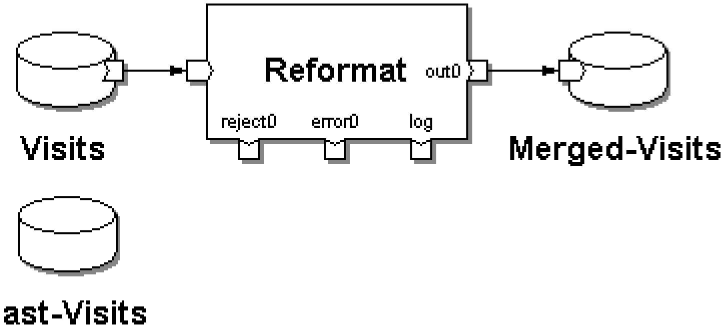
`lookup`(*“file-name” , key-expression*)

- The Lookup functions returns a record that matches the key values and has the format given by the **Record Format parameter.**

Using Lookup File instead of Join



Using Last-Visits
as a lookup file



Lookup File

- **Storage Methods**
  - **Serial lookup :** lookup()
    - whole file replicated to each partition
  - **Parallel lookup :** lookup_local()
    - file partitions held separately
- **Lookup Functions**

Name	Arguments	Purpose
lookup()	File Label and Expression.	Returns a data record from a Lookup File which matches with the values of the expression argument
lookup_count()	- do -	Returns the number of matching data records in a Lookup File.
lookup_next()	File Label	Returns successive data records from a Lookup File.
lookup_local	File Label and Expression.	Returns a data record from a partition of a Lookup File.
lookup_count_local()	- do -	Same as lookup_count but for a single partition
lookup_next_local()	File Label	Same as lookup_count but for a single partition

NOTE: Data needs to be partitioned on same key before using lookup local functions

Transform Functions : XFRs

- Transform functions direct the behavior of transform component
- It is named,parameterized sequence of local variable definition, statements & rules that computes the expression from input values & variables,and assigns results to the output object.
- **Syntax**

```
  output-var[,output-var….]::xform-name(input-var[,input-var…])=
begin
       local-variable-declaration-list
           Variable-list
           Rule-list
    end;
```

- **A transform function definition consists of:**
  1. A list of output variables followed by a double colon(::)
  2. A name for the transform function
  3. A list of input variables
  4. An optional list of local variable definition
  5. An optional list of local statements
  6. A series of rules

The list of local variable definitions, if any, must precede the list of statements. The list of statements, if any, must appear before the list of rules

Example:

```
1.   temp::trans1(in) =
    begin
      temp.sum :: 0;.............Local variable declaration with field sum
    end;
2.  out.temp::trans2(temp, in) =
    begin
      temp.sum :: temp.sum + in. amount;
      out. city :: in. city;
      out.sum :: temp.sum;
```

Basic Components

TATA CONSULTANCY SERVICES

Basic Components

- Filter by Expression
- Reformat
- Redefine Format
- Replicate
- Join
- Sort
- Rollup
- Aggregate
- Dedup Sorted

Reformat

1. Reads record from **in port**
2. Changes the record format by dropping fields, or by using DML expressions to add fields, combine fields, or transform the data in the records.
3. Records written to **out ports**, if the function returns a success status
4. Records written to **reject ports** with descriptive message to **error port**, if the function returns NULL

Ports of Reformat Component

- **IN**
  - Records enter into the component from this port
- **OUT**
  - Success records are written to this port

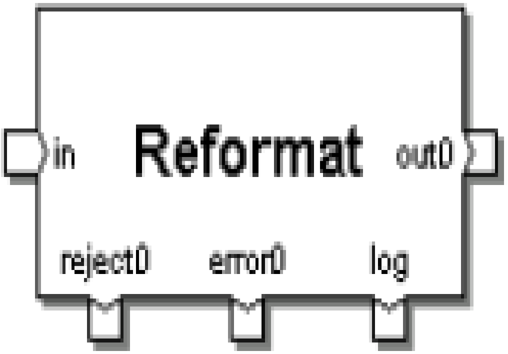
Diagnostic Ports :

- **REJECT**
  - Input records that caused error are sent to this port
- **ERROR**
  - Associated error message is written to this port
- **LOG**
  - Logging records are sent to this port

Note: Every transform component has got diagnostic ports

Reformat

Parameters of Reformat Component



- **Count :** The integer from 1 to 20 that sets the number of each of the following.

1.out ports
2.error ports
3.reject ports
4.transform parameters

The default value is 1

- **transform*n*:** Either the name of file, or a transform string, containing a transform function corresponding to an out port n.
- **Reject-Threshold :** The components tolerance for reject event
  - **Abort on first reject:** The component stops the execution of graph at the first reject event it generates.
  - **Never Abort:** The component does not stops execution of the graph, no matter how many reject events it generates
  - **Use Limit/Ramp:** The component uses the settings in the ramp & limit parameters to determine how many reject events to allow before it stops the execution of graph.
- **Limit:** contains an integer that represents a number of reject events
- **Ramp:** contains a real number that represents a rate of reject events in the number of records processed.

Tolerance value=limit + ramp*total number of records read

Reformat

Typical Limit and Ramp settings . .

- **Limit =** 0 **Ramp =** 0.0 → **Abort on any error**
- **Limit =** 50 **Ramp =** 0.0 → **Abort after 50 errors**
- **Limit =** 1 **Ramp =** 0.01 → **Abort if more than 2 in 100 records causes error**
- **Limit =** 1 **Ramp =** 1 → **Never Abort**

- **Logging:** specifies whether or not you want the component to generate log records for certain events. The values of logging parameter is True or False.

The default value is False.

- **log_input:** indicates how often you want the component to send an input record to its log port.

For example: If you select 100,then the component sends every 100th input record to its log port

- **log_output:** indicates how often you want the component to send an output record

to its log port.

For example: If you select 100,then the component sends every 100th output record to its log port

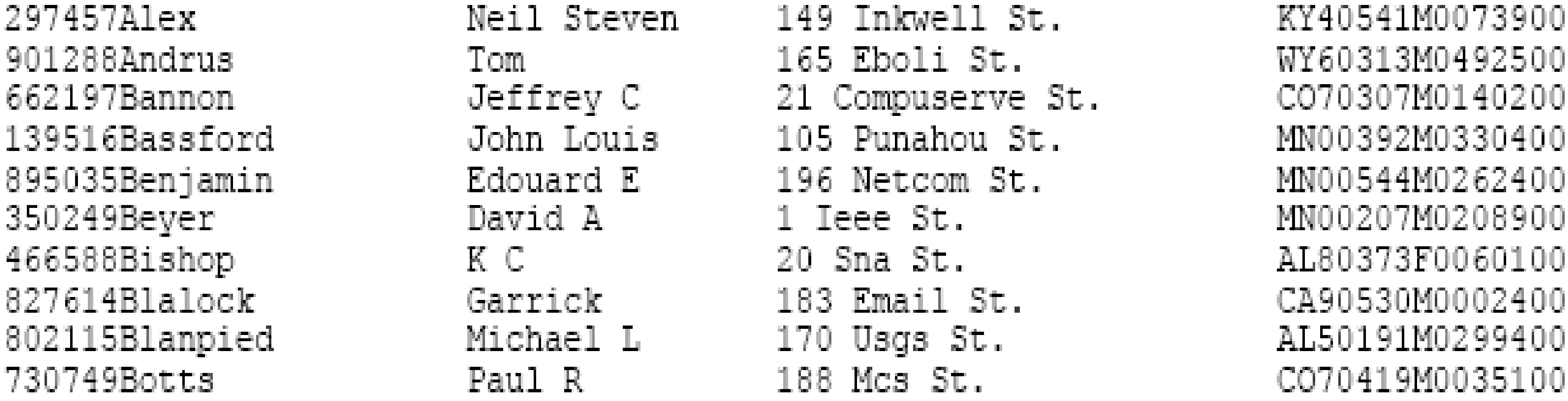
- **log_reject:**indicates how often you want the component to send an reject record

to its log port.

For example: If you select 100,then the component sends every 100th reject record to its log port

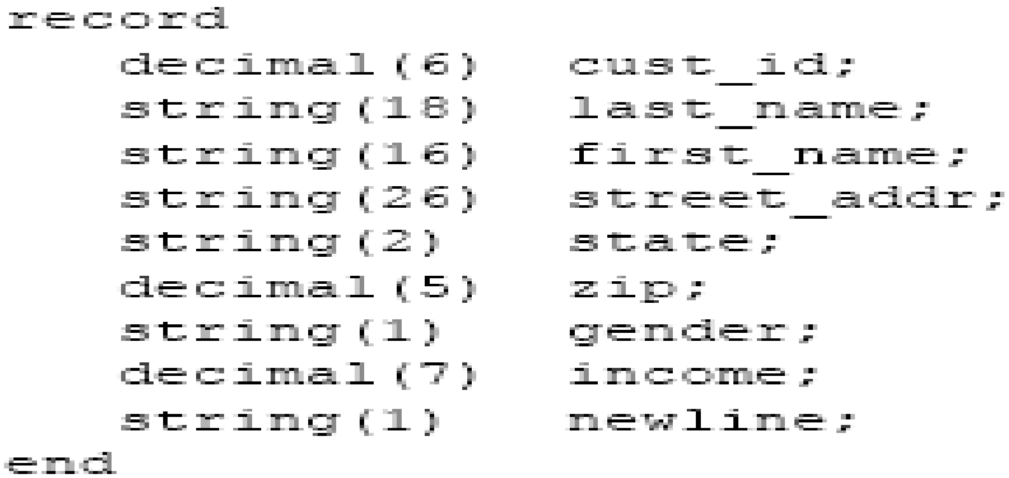
Example of Reformat

The following is the data of the **Input file :**



```
297457Alex              Neil Steven     149 Inkwell St.           KY40541M0073900
901288Andrus            Tom             165 Eboli St.             WY60313M0492500
662197Bannon            Jeffrey C       21 Compuserve St.         CO70307M0140200
139516Bassford          John Louis      105 Punahou St.           MN00392M0330400
895035Benjamin          Edouard E       196 Netcom St.            MN00544M0262400
350249Beyer             David A         1 Ieee St.                MN00207M0208900
466588Bishop            K C             20 Sna St.                AL80373F0060100
827614Blalock           Garrick         183 Email St.             CA90530M0002400
802115Blanpied          Michael L       170 Usgs St.              AL50191M0299400
730749Botts             Paul R          188 Mcs St.               CO70419M0035100
```

The following is the record format of the **Input file :**

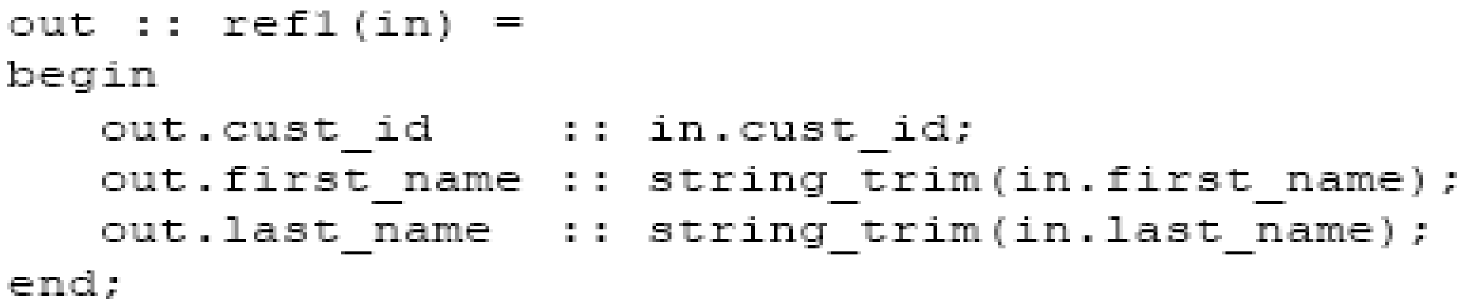


```
record
    decimal(6)   cust_id;
    string(18)   last_name;
    string(16)   first_name;
    string(26)   street_addr;
    string(2)    state;
    decimal(5)   zip;
    string(1)    gender;
    decimal(7)   income;
    string(1)    newline;
end
```

Example of Reformat

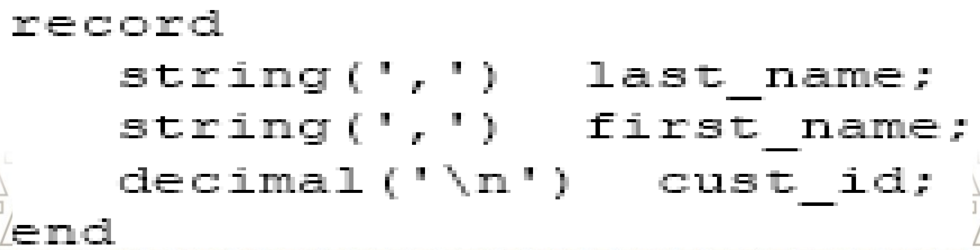
- In this example Reformat has the two transform functions, each of which writes output to an out port

- Reformat uses the following transform function to write output to out port **out0:**



```
out :: ref1(in) =
begin
   out.cust_id      :: in.cust_id;
   out.first_name   :: string_trim(in.first_name);
   out.last_name    :: string_trim(in.last_name);
end;
```

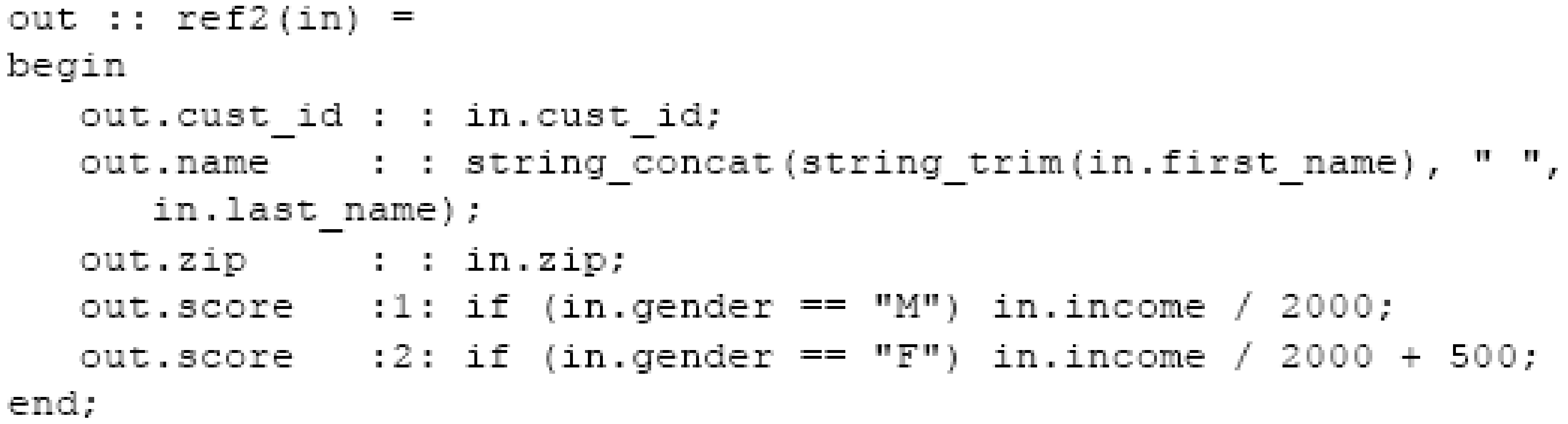
- The following is the record format of **out0:**



```
record
    string(',')   last_name;
    string(',')   first_name;
    decimal('\n')   cust_id;
end
```

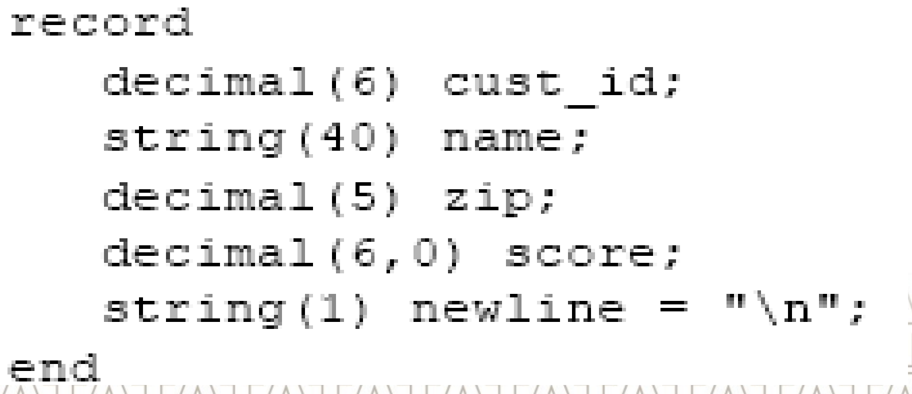
Example of Reformat

- Reformat uses the following transform function to write output to out port **out1:**



```
out :: ref2(in) =
begin
   out.cust_id : : in.cust_id;
   out.name    : : string_concat(string_trim(in.first_name), " ",
      in.last_name);
   out.zip     : : in.zip;
   out.score   :1: if (in.gender == "M") in.income / 2000;
   out.score   :2: if (in.gender == "F") in.income / 2000 + 500;
end;
```

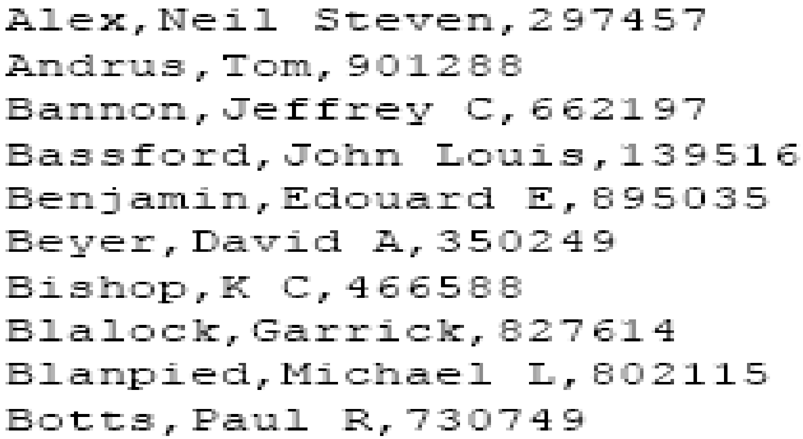
The following is the record format of **out1:**



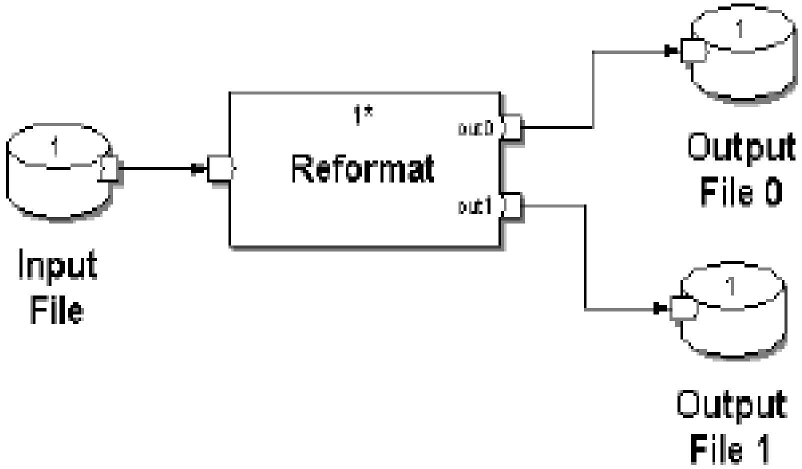
```
record
   decimal(6) cust_id;
   string(40) name;
   decimal(5) zip;
   decimal(6,0) score;
   string(1) newline = "\n";
end
```

Example of Reformat

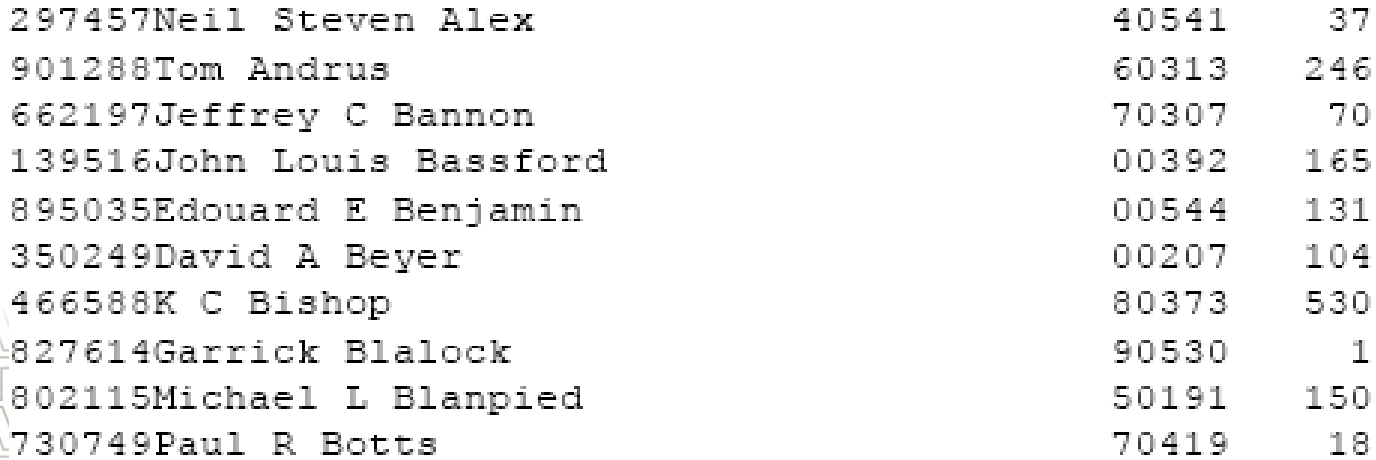
- The graph produces **Output File 0** with the following output :



```
Alex,Neil Steven,297457
Andrus,Tom,901288
Bannon,Jeffrey C,662197
Bassford,John Louis,139516
Benjamin,Edouard E,895035
Beyer,David A,350249
Bishop,K C,466588
Blalock,Garrick,827614
Blanpied,Michael L,802115
Botts,Paul R,730749
```



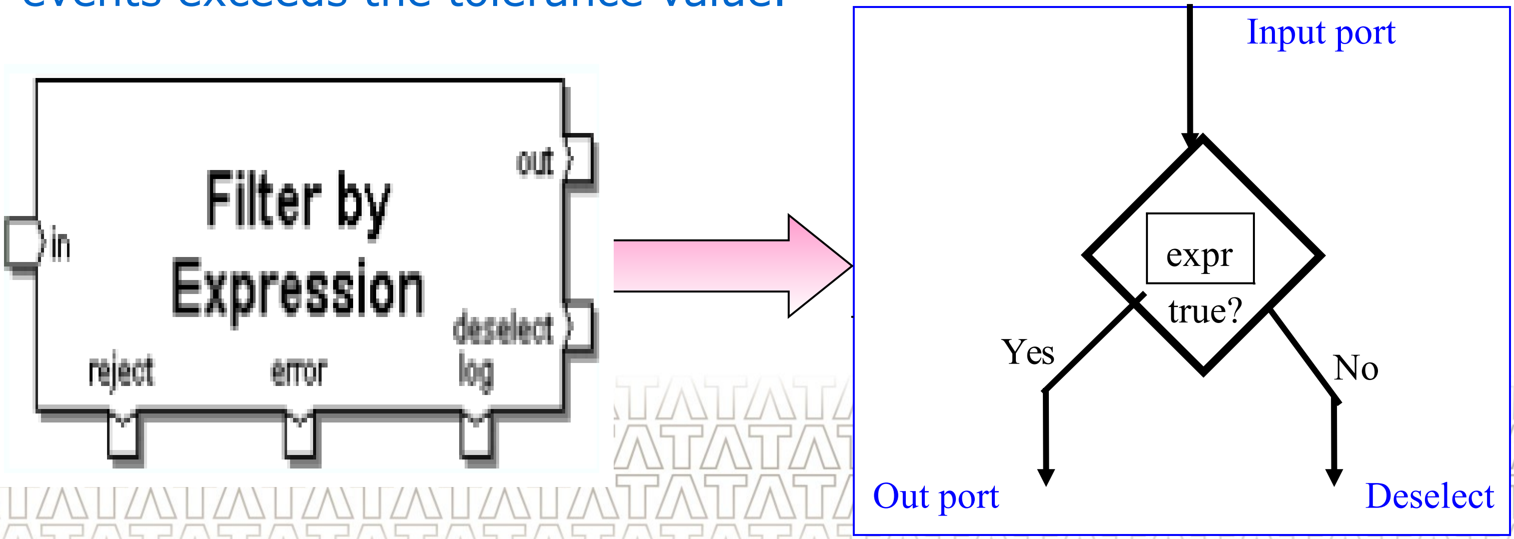
- The graph produces **Output File 1** with the following output :



```
297457Neil Steven Alex                        40541     37
901288Tom Andrus                              60313    246
662197Jeffrey C Bannon                        70307     70
139516John Louis Bassford                     00392    165
895035Edouard E Benjamin                      00544    131
350249David A Beyer                           00207    104
466588K C Bishop                              80373    530
827614Garrick Blalock                         90530      1
802115Michael L Blanpied                      50191    150
730749Paul R Botts                            70419     18
```

Filter by Expression

1. Reads record from the in port
2. Applies the expression in the **select_expr** parameter to each record. If the expression returns
   - **Non-0 Value :**it writes the record to the **out port**
   - **0 :**it writes the record to **deselect port** & if you do not connect deselect port, discards the record.
   - **NULL :**it writes the record to the **reject port** and a descriptive error message to the **error port.**
3. Filter by Expression stops the execution of graph when the number of reject events exceeds the tolerance value.



Ports of Filter by Expression

- **IN**
  - Records enter into the component through this port
- **DESELECT**
  - Records returning 0 after applying expression are written to this port
- **OUT**
  - Success records are written to this port

Diagnostic Ports :

- **REJECT**
  - Input records that caused error are sent to this port
- **ERROR**
  - Associated error message is written to this port
- **LOG**
  - Logging records are sent to this port

Filter by Expression

Parameters of Filter by Expression Component :

- **select_expr :** filter condition for input data records
- **Reject-Threshold :** The components tolerance for reject event
  - **Abort on first reject:** The component stops the execution of graph at the first reject event it generates.
  - **Never Abort:** The component does not stops execution of the graph, no matter how many reject events it generates
  - **Use Limit/Ramp:** The component uses the settings in the ramp & limit parameters to determine how many reject events to allow before it stops the execution of graph.
- **Limit:** contains an integer that represents a number of reject events
- **Ramp:** contains a real number that represents a rate of reject events in the number of records processed.

Tolerance value=limit + ramp*total number of records read

Typical Limit and Ramp settings . .

- **Limit =** 0 **Ramp =** 0.0 → **Abort on any error**
- **Limit =** 50 **Ramp =** 0.0 → **Abort after 50 errors**
- **Limit =** 1 **Ramp =** 0.01 → **Abort if more than 2 in 100 records causes error**
- **Limit =** 1 **Ramp =** 1 → **Never Abort**

Filter by Expression

- **Logging:** specifies whether or not you want the component to generate log records for certain events. The values of logging parameter is True or False.

The default value is False.

  - **log_input:** indicates how often you want the component to send an input record to its log port.

For example: If you select 100,then the component sends every 100th input record to its log port

  - **log_output:** indicates how often you want the component to send an output record

to its log port.

For example: If you select 100,then the component sends every 100th output record to its log port

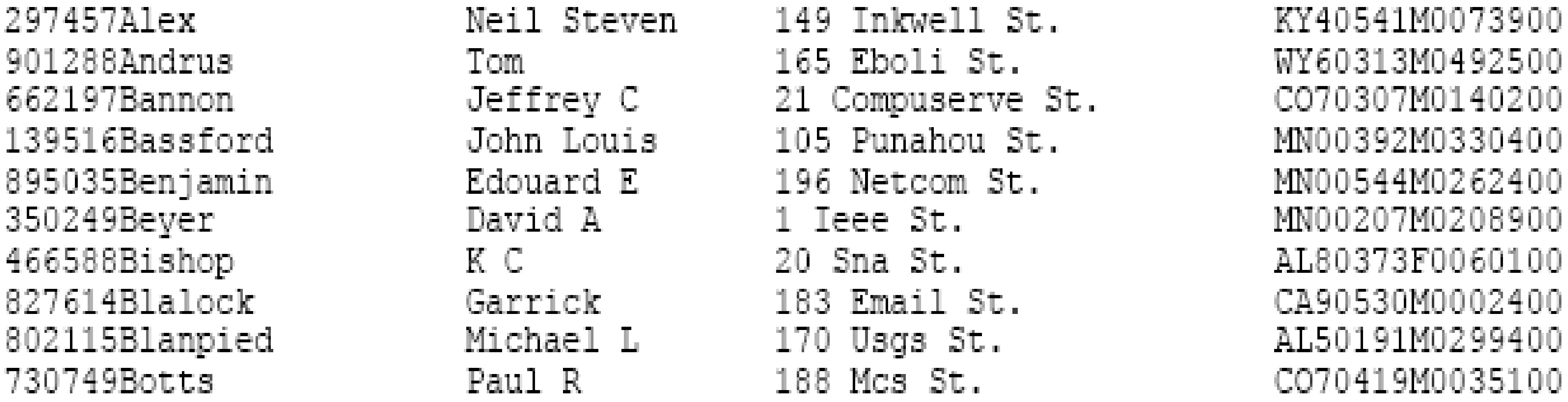
  - **log_reject:**indicates how often you want the component to send an reject record

to its log port.

For example: If you select 100,then the component sends every 100th reject record to its log port

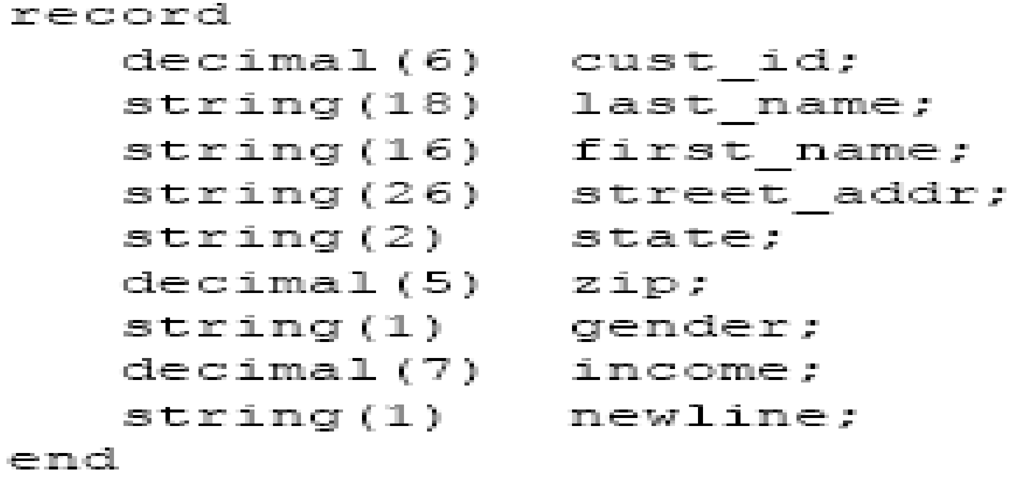
Example of Filter by Expression

The following is the data of the **Input file :**



```
297457Alex                 Neil Steven      149 Inkwell St.           KY40541M0073900
901288Andrus               Tom              165 Eboli St.             WY60313M0492500
662197Bannon               Jeffrey C        21 Compuserve St.         CO70307M0140200
139516Bassford             John Louis       105 Punahou St.           MN00392M0330400
895035Benjamin             Edouard E        196 Netcom St.            MN00544M0262400
350249Beyer                David A          1 Ieee St.                MN00207M0208900
466588Bishop               K C              20 Sna St.                AL80373F0060100
827614Blalock              Garrick          183 Email St.             CA90530M0002400
802115Blanpied             Michael L        170 Usgs St.              AL50191M0299400
730749Botts                Paul R           188 Mcs St.               CO70419M0035100
```

The following is the record format of the **Input file :**

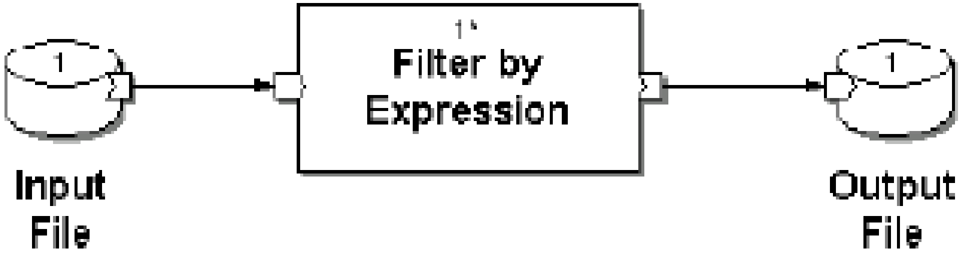


```
record
     decimal(6)   cust_id;
     string(18)   last_name;
     string(16)   first_name;
     string(26)   street_addr;
     string(2)    state;
     decimal(5)   zip;
     string(1)    gender;
     decimal(7)   income;
     string(1)    newline;
end
```

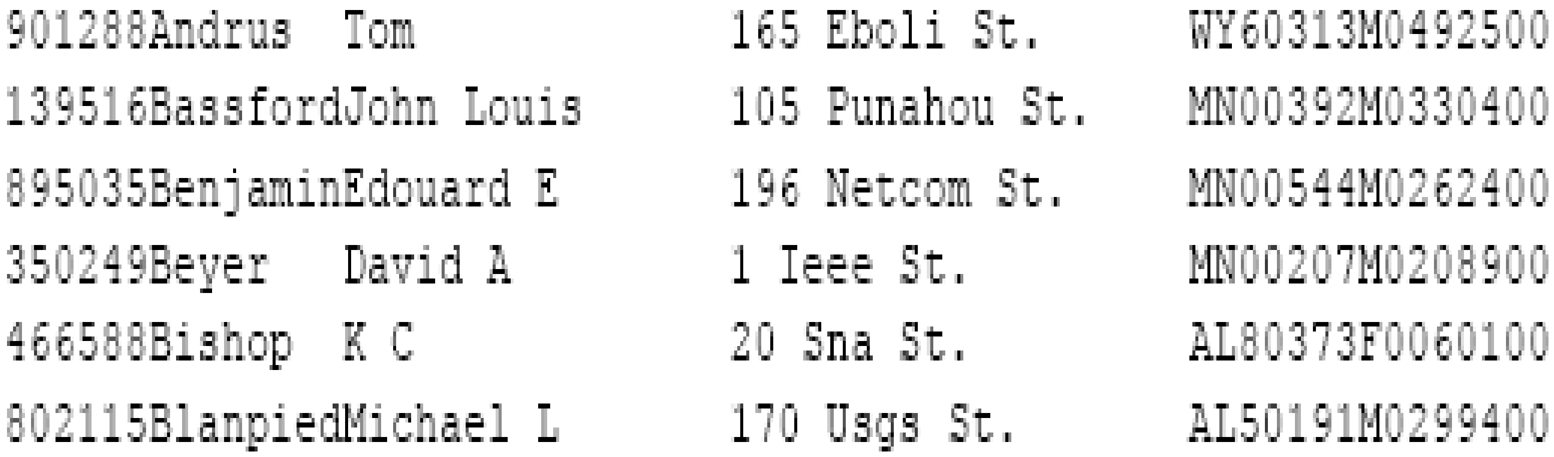
Example of Filter by Expression

- Let Filter by Expression uses the following filter expression.

Gender = = “F” || income> 200000



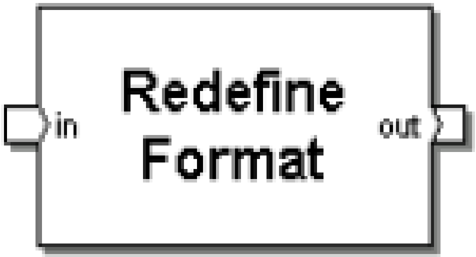
- The graph produces the output file with following data :



```
901288Andrus   Tom             165 Eboli St.      WY60313M0492500
139516BassfordJohn Louis       105 Punahou St.    MN00392M0330400
895035BenjaminEdouard E        196 Netcom St.     MN00544M0262400
350249Beyer    David A         1 Ieee St.         MN00207M0208900
466588Bishop   K C             20 Sna St.         AL80373F0060100
802115BlanpiedMichael L        170 Usgs St.       AL50191M0299400
```

Redefine Format

1. Redefine format copies data records from its input to its output without changing the values in the data records.



2. Reads records from in port.

3. writes the data records to the out port with the fields renamed according to the record format of the out port.

Parameters: **None**

Example of Redefine format

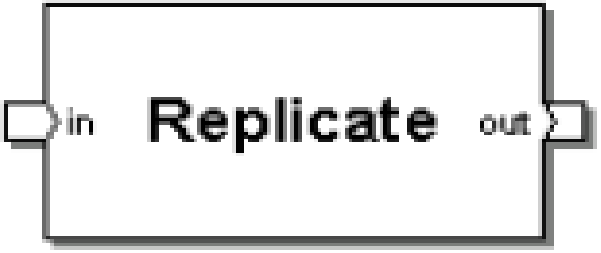
Suppose the input record format is:

```
record
    String(10)      first_name;
    String(10)      last_name;
    String(30)      address;
    Decimal(6)      postal_code;
    Decimal(8.2)    salary;
end
```

You can reduce the number of fields by specifying the output record format as :

```
record
   String(56)       personal_info;
   Decimal(8.2)     salary;
end
```

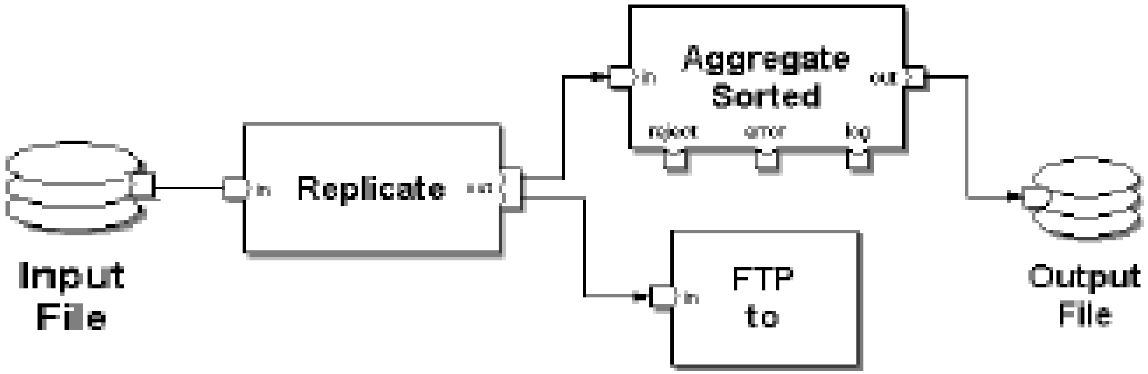
Replicate



- Arbitrarily combines all the data records it receives into a single flow
- Writes the copy of that flow to each of the output flows connected to the out port

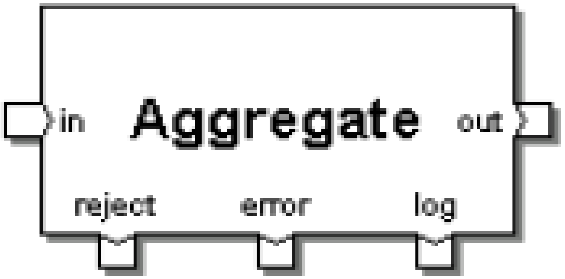
Example of Replicate

Suppose you want to aggregate the flow of records and also send them to the another computer, you can accomplish this by using Replicate component.



Aggregate

- Reads record from the in port
- If you have defined the **select** parameter, it applies the expression in the **select** parameter to each record. If the expression returns
  - **Non-0 Value :**it processes the record
  - **0 :**it does not process that record
  - **NULL :** writes a descriptive error message to the **error port** & stops the execution of the graph**.**
- If you do not supply an expression for the **select** parameter, Aggregate processes all the records on the in port.
- Uses the transform function to aggregate information about groups of records.
- Writes output data records to **out** port that contain aggregated information



Ports of Aggregate Component

- **IN**
  - Records are read from this port
- **OUT**
  - aggregated records are written to this port

Diagnostic Ports :

- **REJECT**
  - Input records that caused error are written to this port
- **ERROR**
  - Associated error message is written to this port
- **LOG**
  - Logging records are written to this port

Aggregate

Parameters of Aggregate component :

- **Sorted-input :**
  - **Input must be sorted or grouped: Aggregate** requires grouped input, and **max-core** parameter is not available
  - **In memory: Input need not be sorted :**Aggregate requires ungrouped input, and requires the use of max-core parameter.

  Default is **Input must be sorted or grouped.**
- **Max-core :** maximum memory usage in bytes
- **Key:** name of the key field Aggregate uses to group the data records
- **Transform :** either name of the file containing the transform function, or the transform string.
- **Select:** filter for data records before aggregation
- **Reject-Threshold :** The components tolerance for reject event
  - **Abort on first reject:** The component stops the execution of graph at the first reject event it generates.
  - **Never Abort:** The component does not stops execution of the graph, no matter how many reject events it generates
  - **Use Limit/Ramp:** The component uses the settings in the ramp & limit parameters to determine how many reject events to allow before it stops the execution of graph.

Aggregate

- **Limit:** contains an integer that represents a number of reject events
- **Ramp:** contains a real number that represents a rate of reject events in the number of records processed.
- **Logging:** specifies whether or not you want the component to generate log records for certain events. The values of logging parameter is True or False.

The default value is False.

- **log_input:** indicates how often you want the component to send an input record to its log port.

For example: If you select 100,then the component sends every 100th input record to its log port

- **log_output:** indicates how often you want the component to send an output record

to its log port.

For example: If you select 100,then the component sends every 100th output record to its log port

- **log_reject:**indicates how often you want the component to send an reject record

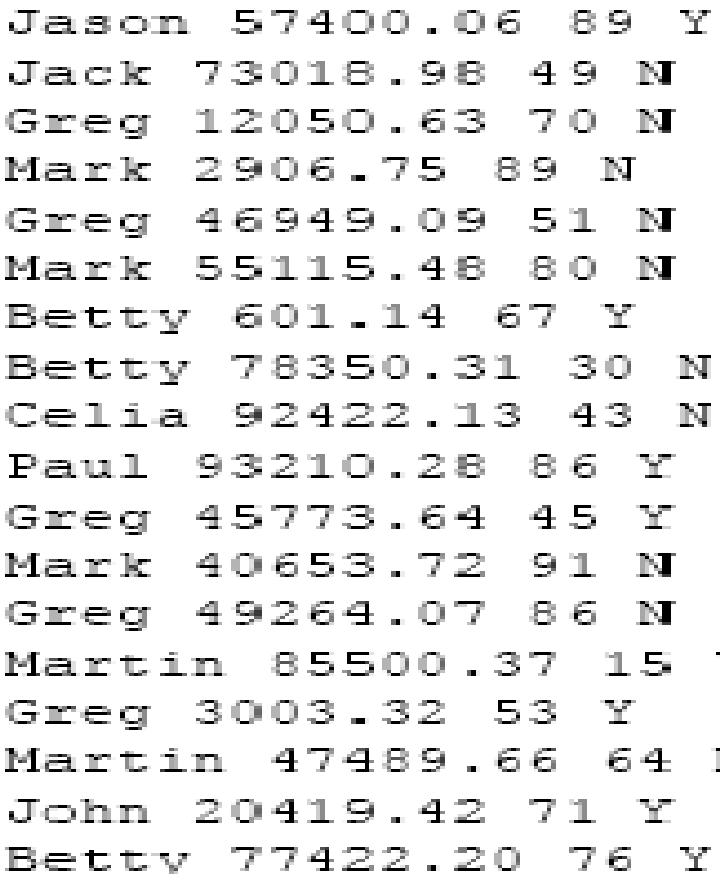
to its log port.

For example: If you select 100,then the component sends every 100th reject record to its log port

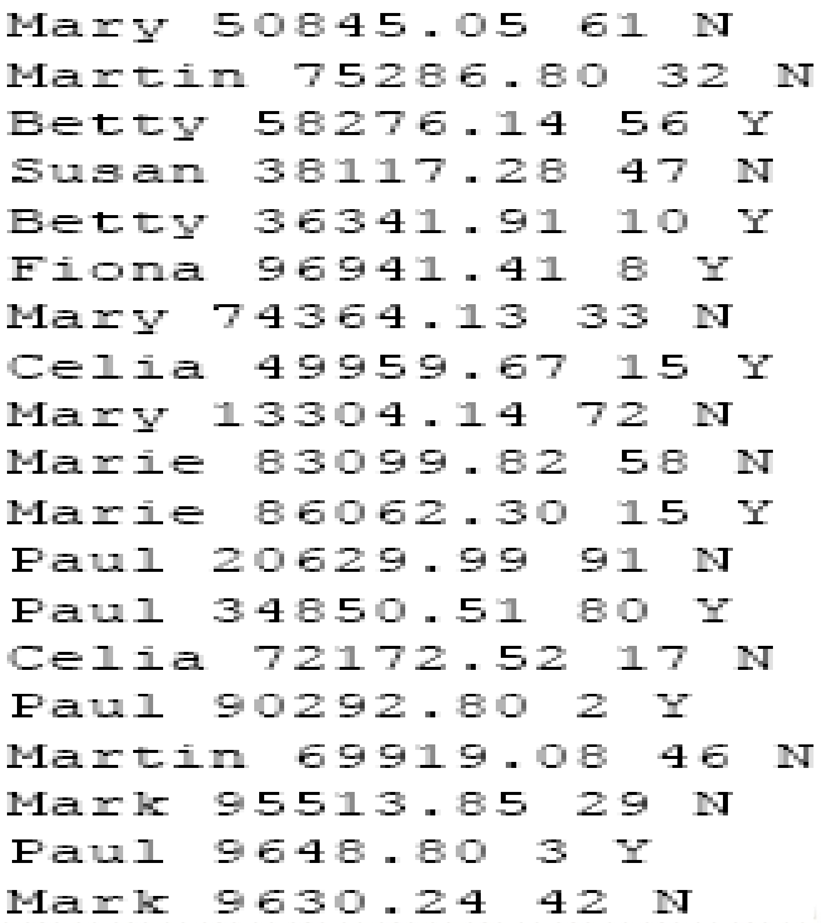
- **log_intermediate:** indicates how often you want the component to send an intermediate record to its log port

Example of Aggregate

- The following is the data of the **Input File** :



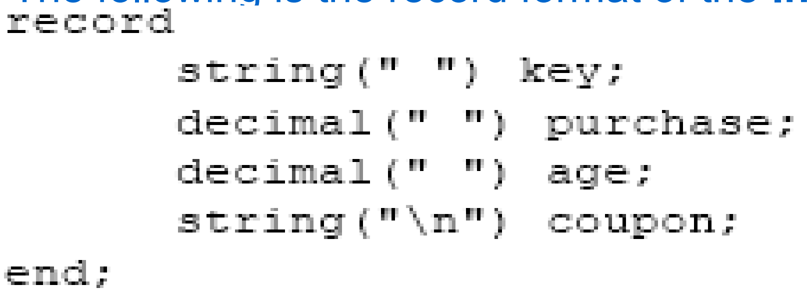
```
Jason 57400.06 89 Y
Jack 73018.98 49 N
Greg 12050.63 70 N
Mark 2906.75 89 N
Greg 46949.09 51 N
Mark 55115.48 80 N
Betty 601.14 67 Y
Betty 78350.31 30 N
Celia 92422.13 43 N
Paul 93210.28 86 Y
Greg 45773.64 45 Y
Mark 40653.72 91 N
Greg 49264.07 86 N
Martin 85500.37 15
Greg 3003.32 53 Y
Martin 47489.66 64
John 20419.42 71 Y
Betty 77422.20 76 Y
```



```
Mary 50845.05 61 N
Martin 75286.80 32 N
Betty 58276.14 56 Y
Susan 38117.28 47 N
Betty 36341.91 10 Y
Fiona 96941.41 8 Y
Mary 74364.13 33 N
Celia 49959.67 15 Y
Mary 13304.14 72 N
Marie 83099.82 58 N
Marie 86062.30 15 Y
Paul 20629.99 91 N
Paul 34850.51 80 Y
Celia 72172.52 17 N
Paul 90292.80 2 Y
Martin 69919.08 46 N
Mark 95513.85 29 N
Paul 9648.80 3 Y
Mark 9630.24 42 N
```

Example of Aggregate

- The following is the record format of the **Input file**:

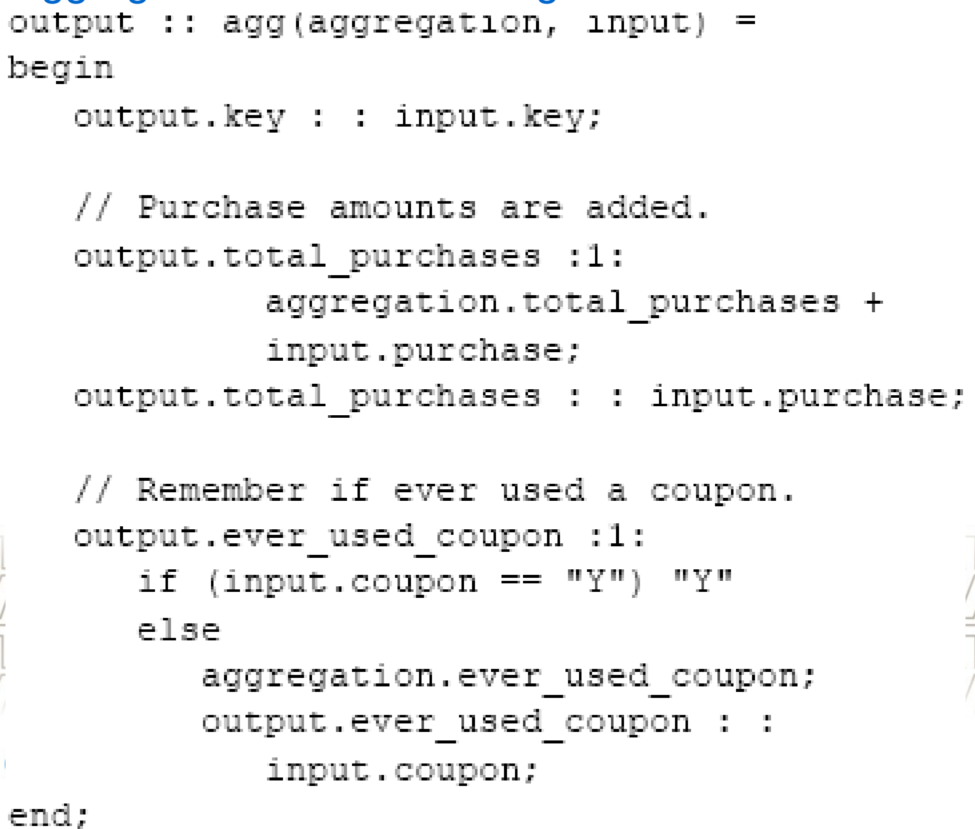


```
record
        string(" ") key;
        decimal(" ") purchase;
        decimal(" ") age;
        string("\n") coupon;
end;
```

- The Aggregate uses the following key specifier to sort the data.

Key

- Aggregate uses the following transform function to write output.



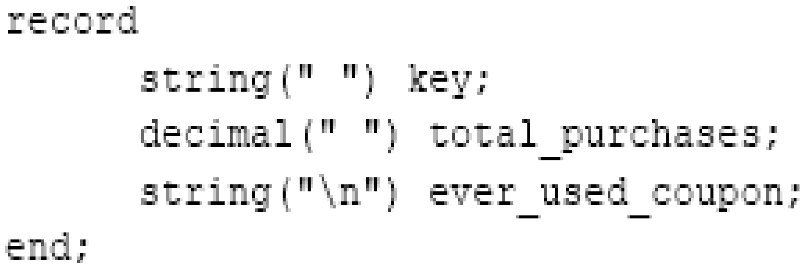
```
output :: agg(aggregation, input) =
begin
   output.key : : input.key;

   // Purchase amounts are added.
   output.total_purchases :1:
            aggregation.total_purchases +
            input.purchase;
   output.total_purchases : : input.purchase;

   // Remember if ever used a coupon.
   output.ever_used_coupon :1:
      if (input.coupon == "Y") "Y"
      else
         aggregation.ever_used_coupon;
         output.ever_used_coupon : :
            input.coupon;
end;
```

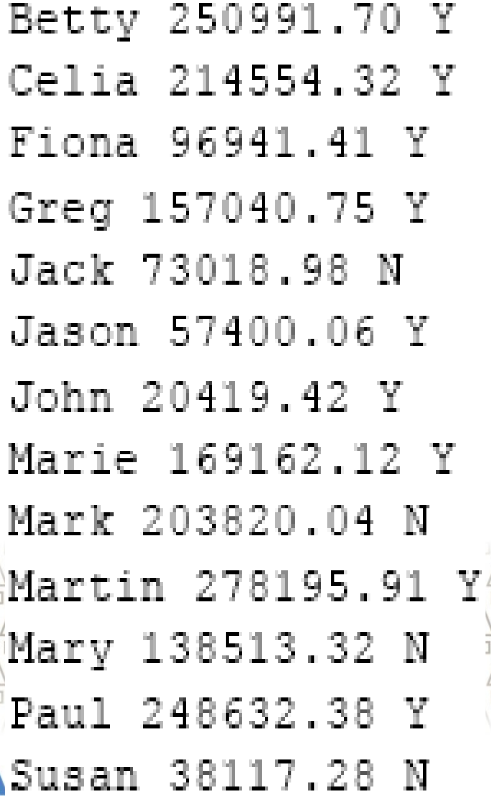
Example of Aggregate

- The following is the record format of the **out port of Aggregate**

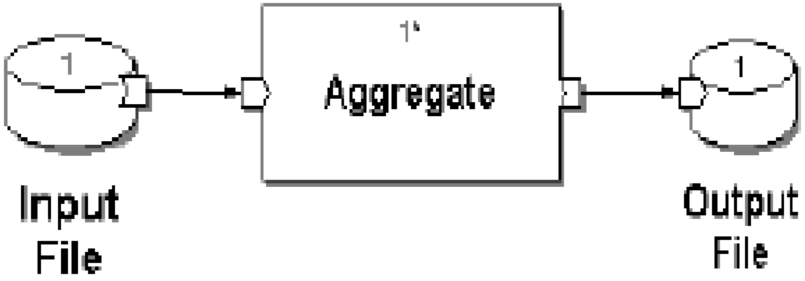


```
record
      string(" ") key;
      decimal(" ") total_purchases;
      string("\n") ever_used_coupon;
end;
```

- After the processing the graph produces the following **Output File :**

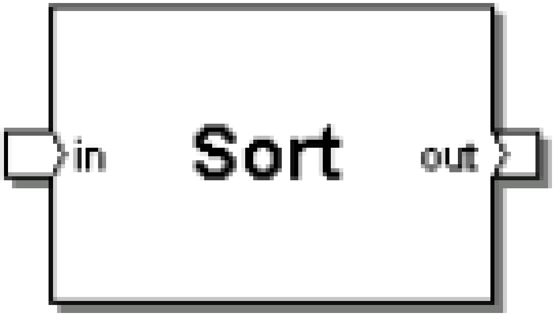


```
Betty 250991.70 Y
Celia 214554.32 Y
Fiona 96941.41 Y
Greg 157040.75 Y
Jack 73018.98 N
Jason 57400.06 Y
John 20419.42 Y
Marie 169162.12 Y
Mark 203820.04 N
Martin 278195.91 Y
Mary 138513.32 N
Paul 248632.38 Y
Susan 38117.28 N
```



Sort

- Sort component sorts and merges the data records.
- The sort component :
  - Reads the records from all the flows connected to the in port until it reaches the number of bytes specified in the **max-core** parameter
  - Sorts the records and writes the results to a temporary file on disk
  - Repeat this procedure until it has read all the records
  - Merges all the temporary files, maintaining the sort order
  - Writes the result to the out port



<u>Ports:</u>

1.IN:records are read from this port

2.OUT:records after sorting are written to this port

Sort

- **Parameters of Sort component :**

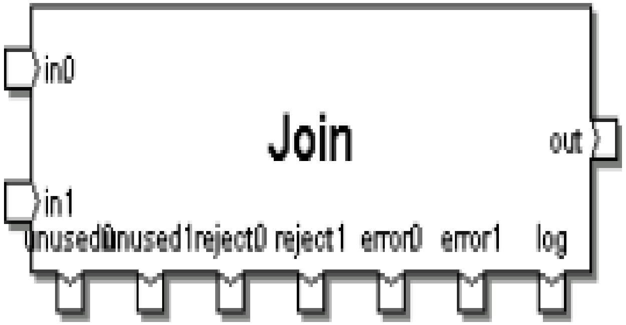
i. **Key:**name of the key fields and sequence specifier,you want sort to use when it orders data records

ii. **Max-core:** maximum memory usage in bytes.

When sort reaches the number of bytes specified in the max-core parameter, it sorts the records it has read and writes a temporary file to disk.

Join

1. Reads records from multiple input ports
2. Operates on records with matching keys using a multi-input transform function
3. Writes result to the output ports



Parameters of Join:

- **Count:** An integer from 2 to 20 specifying number of following ports and parameters. **Default is 2.**
  - **In** ports
  - **Unused** ports
  - **Reject** ports
  - **Error** ports
  - **Record-required** parameter
  - **Dedup** parameter
  - **Select** parameter
  1. **Override-key** parameter

1. **Key:** Name of the fields in the input record that must have matching values for Join to call transform function

Join

- **Sorted-input:**
  - **Input must be sorted: Join** accepts unsorted input, and permits the use of **maintain-order** parameter
  - **In memory: Input need not be sorted : Join** requires sorted input, and **maintain-order** parameter is not available.

Default is **Input must be sorted**

- **Logging:** specifies whether or not you want the component to generate log records for certain events. The values of logging parameter is True or False.

The default value is False.

- **log_input:** indicates how often you want the component to send an input record to its log port.

For example: If you select 100,then the component sends every 100th input record to its log port

- **log_output:** indicates how often you want the component to send an output record

to its log port.

For example: If you select 100,then the component sends every 100th output record to its log port

- **log_reject:**indicates how often you want the component to send an reject record

to its log port.

For example: If you select 100,then the component sends every 100th reject record to its log port

- **log_intermediate:** indicates how often you want the component to send an intermediate record to its log port

Join

- **Max-core :** maximum memory usage in bytes
- **Transform :** either name of the file containing the transform function, or the transform string.
- **Select*n*:** filter for data records before aggregation. One per **in***n* port.
- **Reject-Threshold :** The components tolerance for reject event
  - **Abort on first reject:** The component stops the execution of graph at the first reject event it generates.
  - **Never Abort:** The component does not stops execution of the graph, no matter how many reject events it generates
  - **Use Limit/Ramp:** The component uses the settings in the ramp & limit parameters to determine how many reject events to allow before it stops the execution of graph.
- **Limit:** contains an integer that represents a number of reject events
- **Ramp:** contains a real number that represents a rate of reject events in the number of records processed.
- **Driving: number** of the port to which you connect the driving input. The driving input is the

  largest input. All the other inputs are read into memory.

The driving parameter is only available when the **sorted-input** parameter is set to

In memory: Input need not be sorted. Specify the port number as the value of the driving parameter. The Join reads all other inputs into memory

Default is 0

- **Max-memory:** maximum memory usage in bytes before Join writes temporary files to disk. Only available when the **sorted-input parameter** is set to **Inputs must be sorted.**

Join

- **Maintain-order:** set to **True** to ensure that records remain in the original order of the driving input. Only available when the **sorted-input** parameter is set to **In memory:Input need not be sorted**.

Default is False.

- **Override-key*n:*** alternative names for the key fields for a particular **in*n*** port.

Default value is 0.0

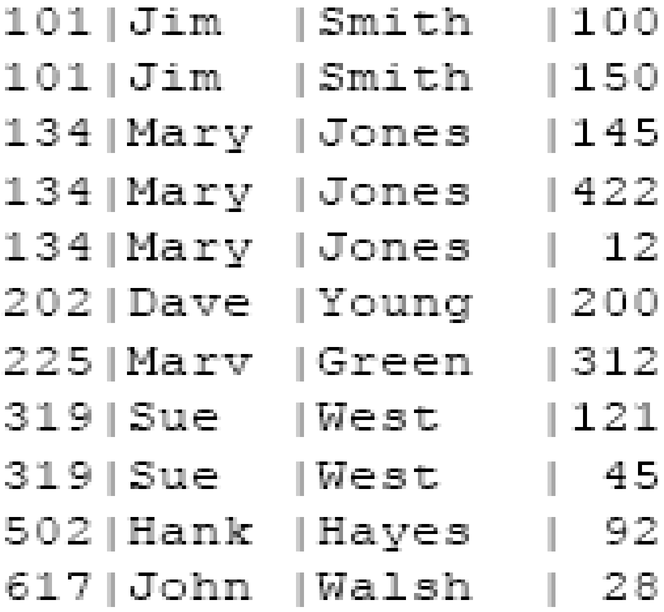
- **Dedup*n*:** set the **dedup*n*** parameter to **True** to remove duplicates from the corresponding inn port before joining.

Default is False, which does not remove duplicates.

- **join-type:** choose from the following
  - **Inner join: sets** the **record-required*n*** parameter for all ports to True. Inner join is the default.
  - **Outer join:** sets the **record-required*n*** parameters for all ports to **False.**
  - **Explicit:** allows you to set the **record-required*n*** parameter for each port individually.
- **record-required*n*:**This parameter is available only when the **join-type** parameter is set to **Explicit.** There is one **record-required*n*** parameter per **in*n*** port.
  - ➔ When there are **2 inputs**, set **record-requiredn** to **True** for the input port for which you want to call the transform for every record regardless of whether there is a matching record on the other input port.
  - ➔ When there are **more than 2 inputs, set record-requiredn** to **True** when you want to call the transform only when there are records with matching keys on all input ports for which **record-requiredn** is **True**.

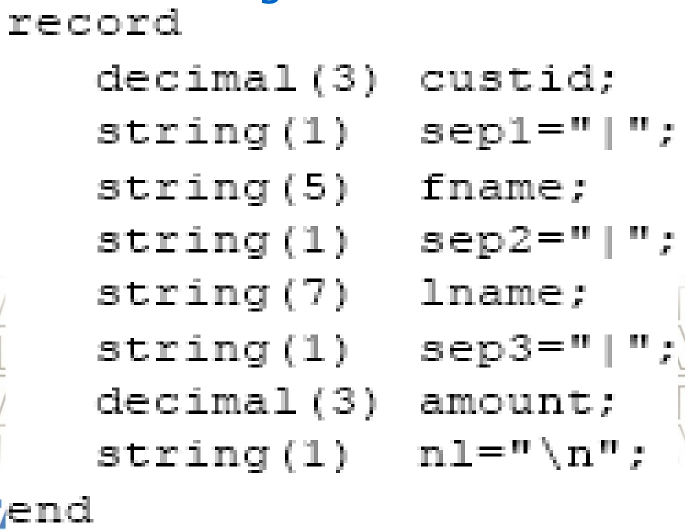
Example of Join

- The following is the data of the **Input File 0 :**



```
101|Jim  |Smith  |100
101|Jim  |Smith  |150
134|Mary |Jones  |145
134|Mary |Jones  |422
134|Mary |Jones  | 12
202|Dave |Young  |200
225|Marv |Green  |312
319|Sue  |West   |121
319|Sue  |West   | 45
502|Hank |Hayes  | 92
617|John |Walsh  | 28
```

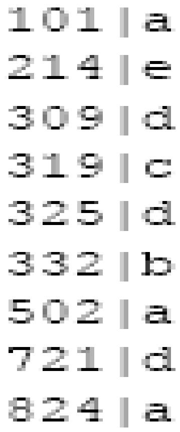
- The following is the record format of the **Input File 0:**



```
record
   decimal(3) custid;
   string(1)  sep1="|";
   string(5)  fname;
   string(1)  sep2="|";
   string(7)  lname;
   string(1)  sep3="|";
   decimal(3) amount;
   string(1)  nl="\n";
end
```

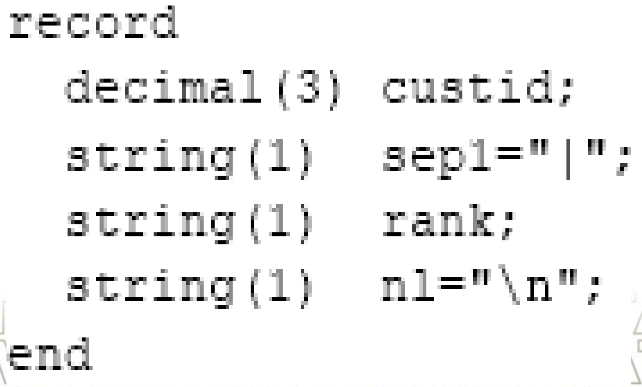
Example of Join

- The following is the data of the **Input File 1:**



```
101|a
214|e
309|d
319|c
325|d
332|b
502|a
721|d
824|a
```

- The following is the record format of the **Input File 1:**



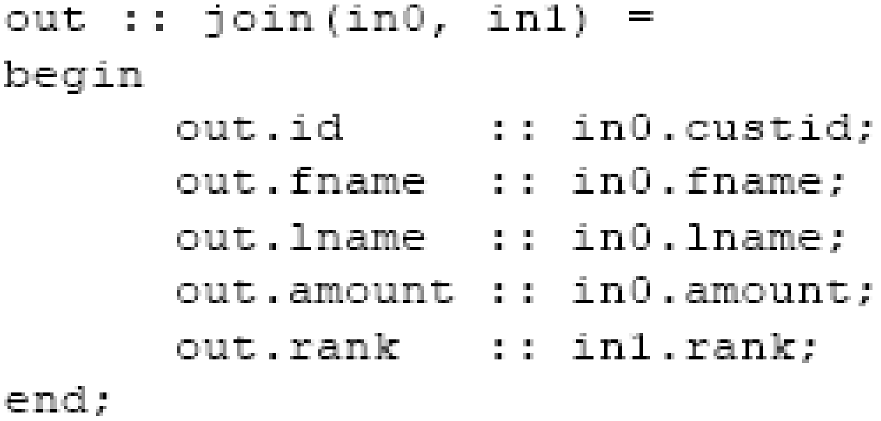
```
record
  decimal(3) custid;
  string(1)  sep1="|";
  string(1)  rank;
  string(1)  nl="\n";
end
```

Example of Join

- The sort component uses the following key to sort the data .

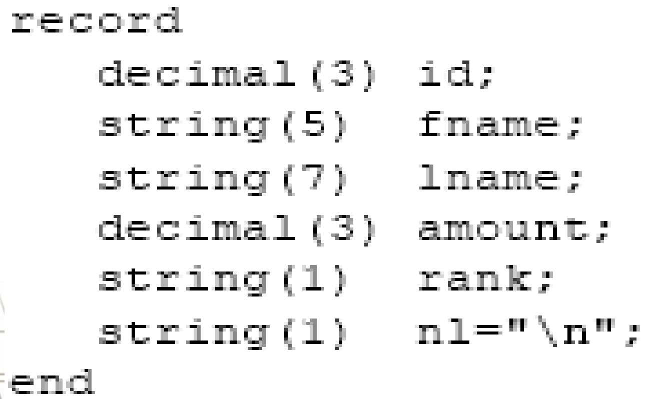
  Custid

- Join uses the following transform function to write output.



```
out :: join(in0, in1) =
begin
      out.id      :: in0.custid;
      out.fname   :: in0.fname;
      out.lname   :: in0.lname;
      out.amount  :: in0.amount;
      out.rank    :: in1.rank;
end;
```

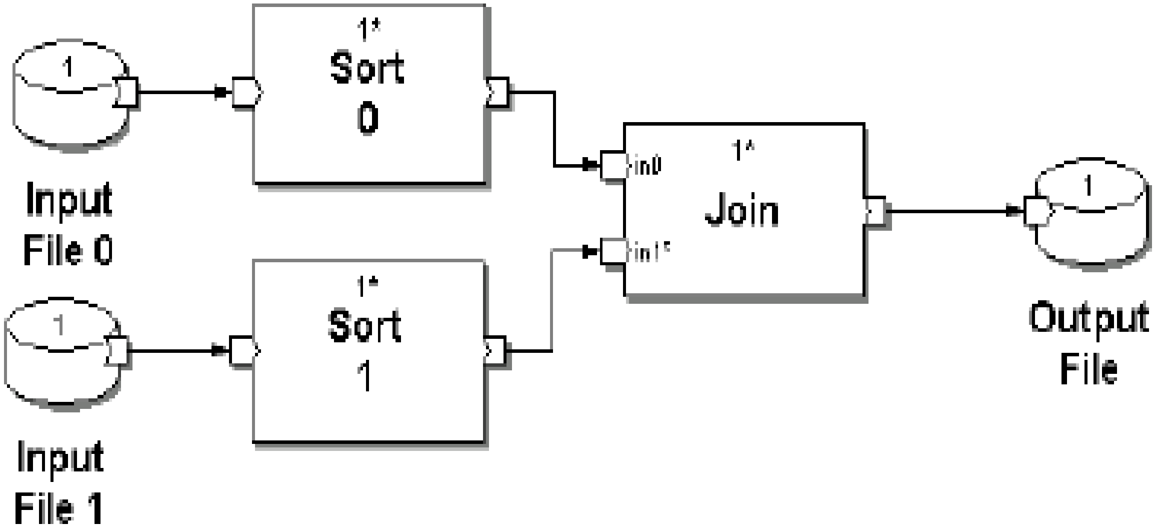
- The following is the record format of the out port of Join.



```
record
   decimal(3) id;
   string(5)  fname;
   string(7)  lname;
   decimal(3) amount;
   string(1)  rank;
   string(1)  nl="\n";
end
```

- Join uses the default value, Inner **join, for** the **join-type** parameter.

Example of Join

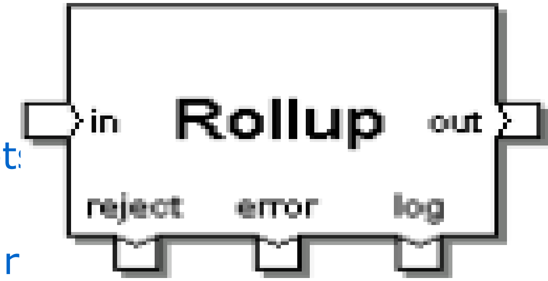


- Given the preceding data, record formats, parameter, and transform function, the graph produces **Output File** with the following data.

```
101Jim  Smith  100a
101Jim  Smith  150a
319Sue  West   121c
319Sue  West    45c
502Hank Hayes   92a
```

Rollup

- Rollup performs a general aggregation of data i.e. it reduces the group of records to a single output record



Parameters of Rollup Component:

- **Sorted-input:**
  - **Input must be sorted or grouped: Rollup** accept

grouped input and **max-core** parameter is not available.

  - **In memory: Input need not be sorted : Rollup** r

ungrouped input, and requires use of the **max-core** parameter.

Default is **Input must be sorted or grouped.**

- **Key-method:** the method by which the component groups the records.
  - **Use key-specifier:** the component uses the key specifier.
  - **Use key_change function:** the component uses the key_change transform function.
- **Key:** names of the key fields Rollup can use to group or to define groups of data records.

If the value of the **key-method parameter** is **Use key-specifier** ,you must specify the value for the **key parameter**.

- **Max-core :** maximum memory usage in bytes
- **Transform :** either name of the file containing the type and transform function, or the transform string.
- **check-sort:** indicates whether or not to abort execution on the first input record that is out of sorted order. **The Default is True.**

This parameter is available only when key-method parameter is Use key-specifier

- **Limit:** contains an integer that represents a number of reject events

Rollup

- **Ramp:** contains a real number that represents a rate of reject events in the number of records processed.
- **Logging:** specifies whether or not you want the component to generate log records for certain events. The values of logging parameter is True or False.

The default value is False.

- **log_input:** indicates how often you want the component to send an input record to its log port.

For example: If you select 100,then the component sends every 100th input record to its log port

- **log_output:** indicates how often you want the component to send an output record to its log port.

For example: If you select 100,then the component sends every 100th output record to its log port

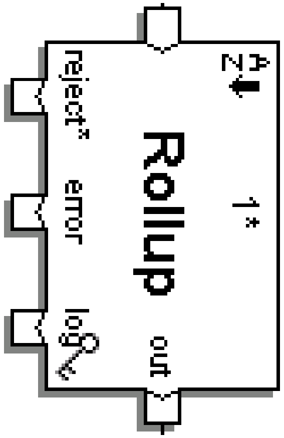
- **log_reject:**indicates how often you want the component to send an reject record to its log port.

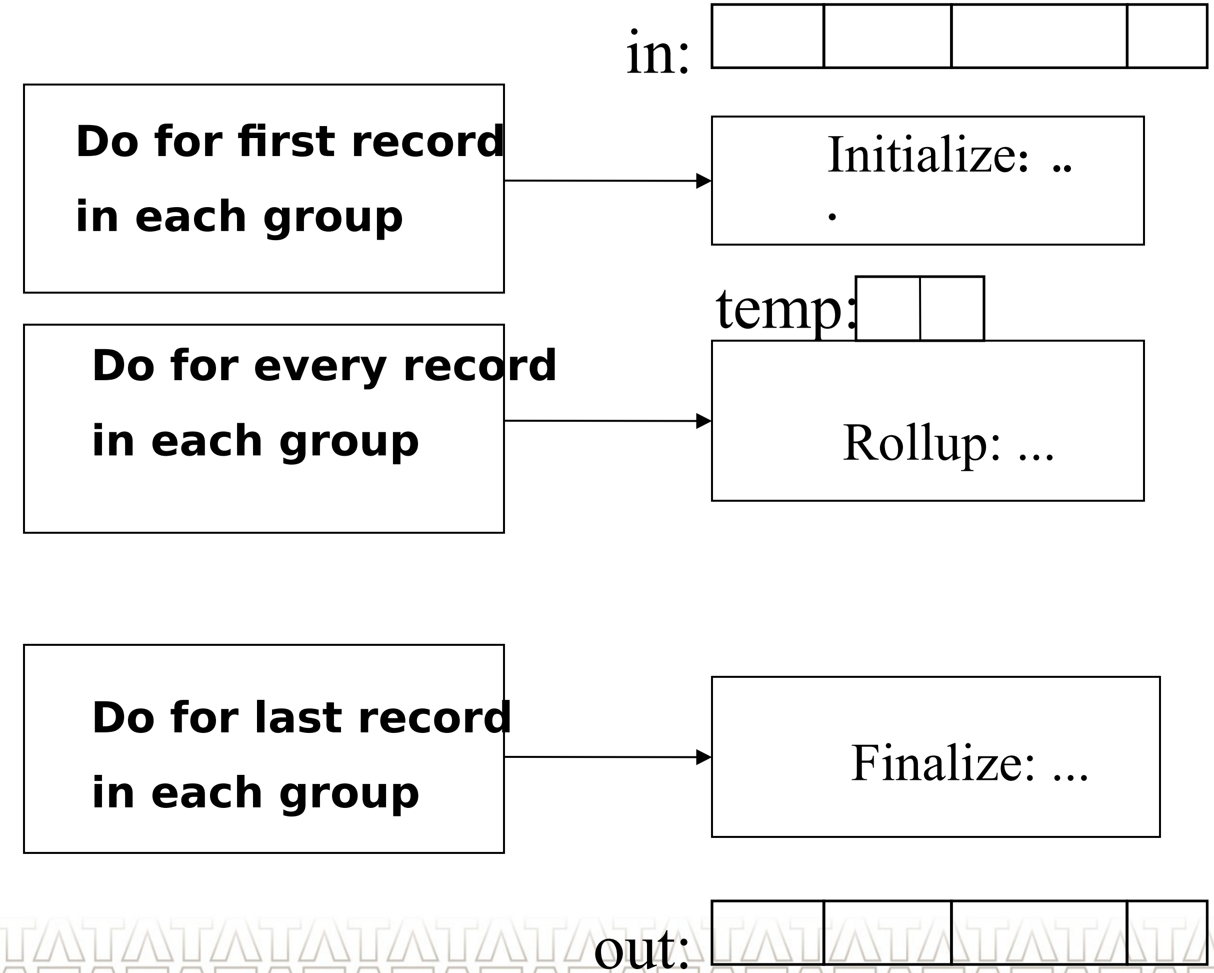
For example: If you select 100,then the component sends every 100th reject record to its log port

- **log_intermediate:** indicates how often you want the component to send an intermediate record to its log port

- **Reject-Threshold :** The components tolerance for reject event
  - **Abort on first reject:** The component stops the execution of graph at the first reject event it generates.
  - **Never Abort:** The component does not stops execution of the graph, no matter how many reject events it generates
  - **Use Limit/Ramp:** The component uses the settings in the ramp & limit parameters to determine how many reject events to allow before it stops the execution of graph.

Rollup





in:

Do for first record
in each group

Initialize: ..
.

temp:

Do for every record
in each group

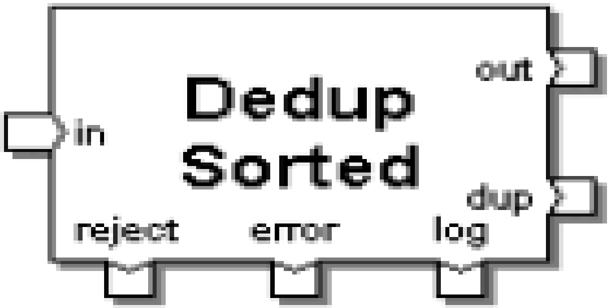
Rollup: ...

Do for last record
in each group

Finalize: ...

out:

Dedup Sorted



- Separates one specified record in each group of records from the rest of the records in that group
- Requires grouped input.
- Reads grouped flow of records from the in port.
- If your records are not already grouped, use Sort Component to group them
- It applies the expression in the **select** parameter to each record. If the expression returns
  - **Non-0 Value :**it processes the record
  - **0 :** it does not process that record
  - **NULL :** writes the record to the reject port & a descriptive error message to the **error port.**
- If you do not supply an expression for the **select** parameter, Dedup Sorted processes all the records on the in port.
- Dedup sorted considers any consecutive records with the same key value to be in the same group.
  - If a group consists of one record, Dedup sorted writes that record to the out port.
  - If a group consists of more than one record, Dedup sorted uses the value of keep parameter to determine:
    - Which record to write to the out port.
    - Which record or records to write to dup port

Ports of Dedup Sorted Component

- **IN**
  - Records enter into the component from this port
- **OUT**
  - Output records are written to this port
- **DUP**
  - Duplicate records are written to this port

Diagnostic Ports :

- **REJECT**
  - Input records that caused error are written to this port
- **ERROR**
  - Associated error message is written to this port
- **LOG**
  - Logging records are written to this port

Dedup Sorted

Parameters of Dedup Sorted Component :

- **Key:** name of the key field, you want Dedup sorted to use when determining group of data records.
- **select: filter** for records before Dedup sorted separates duplicates.
- **keep: determines** which record Dedup sorted keeps to write to the out port
  - **first: keeps** first record of the group. **This is the default.**
  - **last: keeps** the last record of the group.
  - **unique- only: keeps** only records with unique key values.

Dedup sorted writes the remaining records of the each group to the dup port

- **Reject- threshold: The** components tolerance for reject events
  - **Abort on first reject:** The component stops the execution of graph at the first reject event it generates.
  - **Never Abort:** The component does not stops execution of the graph, no matter how many reject events it generates
  - **Use Limit/Ramp:** The component uses the settings in the ramp & limit parameters to determine how many reject events to allow before it stops the execution of graph.
- **Limit:** contains an integer that represents a number of reject events
- **Ramp:** contains a real number that represents a rate of reject events in the number of records processed.
- **Check- sort:** indicates whether you want processing to abort on the first record that is out of sorted order.

Dedup Sorted

- **Logging:** specifies whether or not you want the component to generate log records for certain events. The values of logging parameter is True or False.

The default value is False.

- **log_input:** indicates how often you want the component to send an input record to its log port.

For example: If you select 100,then the component sends every 100th input record to its log port

- **log_output:** indicates how often you want the component to send an output record

to its log port.

For example: If you select 100,then the component sends every 100th output record to its log port

- **log_reject:**indicates how often you want the component to send an reject record

to its log port.

For example: If you select 100,then the component sends every 100th reject record to its log port

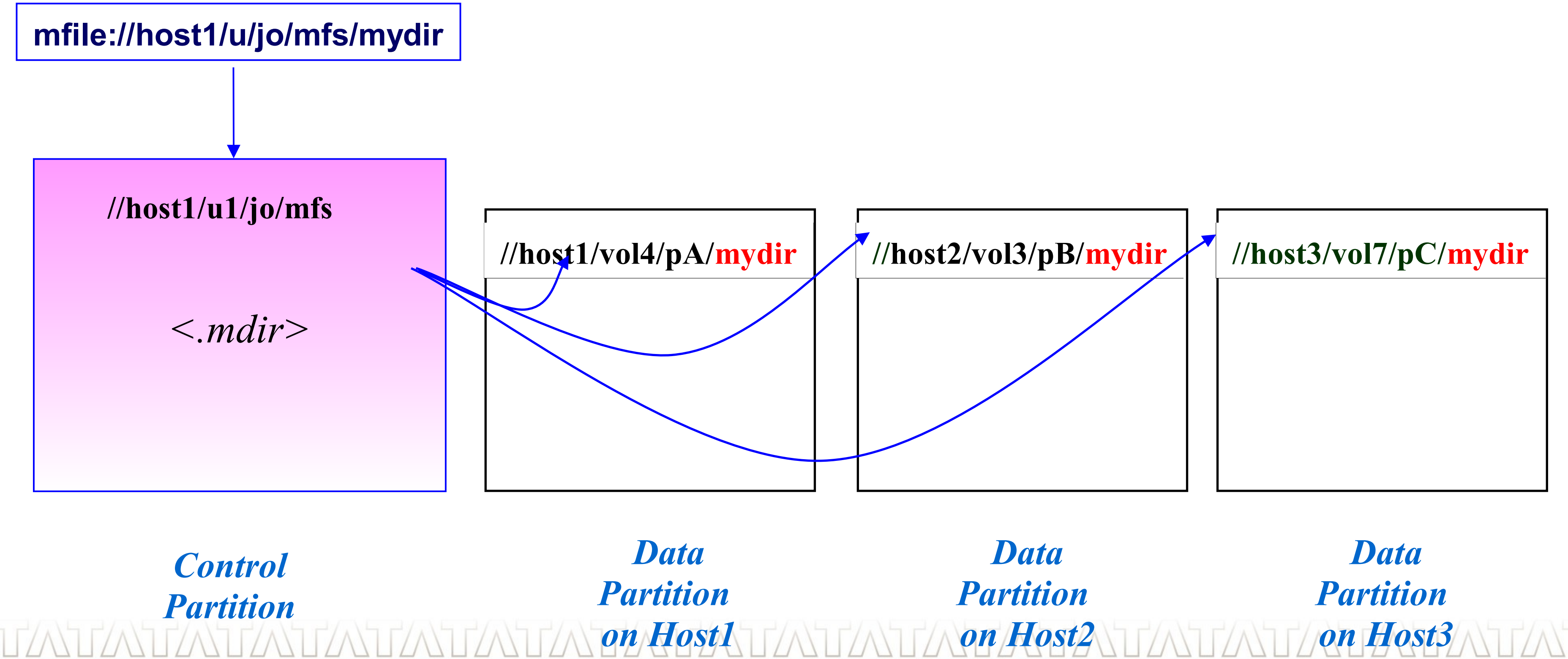
Partitioning and De-partitioning

Multifiles

- **A global view of a set of ordinary files called *partitions* usually located on different disks or systems**
- **Ab Initio provides shell level utilities called *"m_ commands"* for handling multifiles (copy, delete, move etc.)**
- **Multifiles reside on Multidirectories**
- **Each is represented using URL notation with *"mfile"* as the protocol part:**
  - mfile://pluto.us.com/usr/ed/mfs1/new.dat

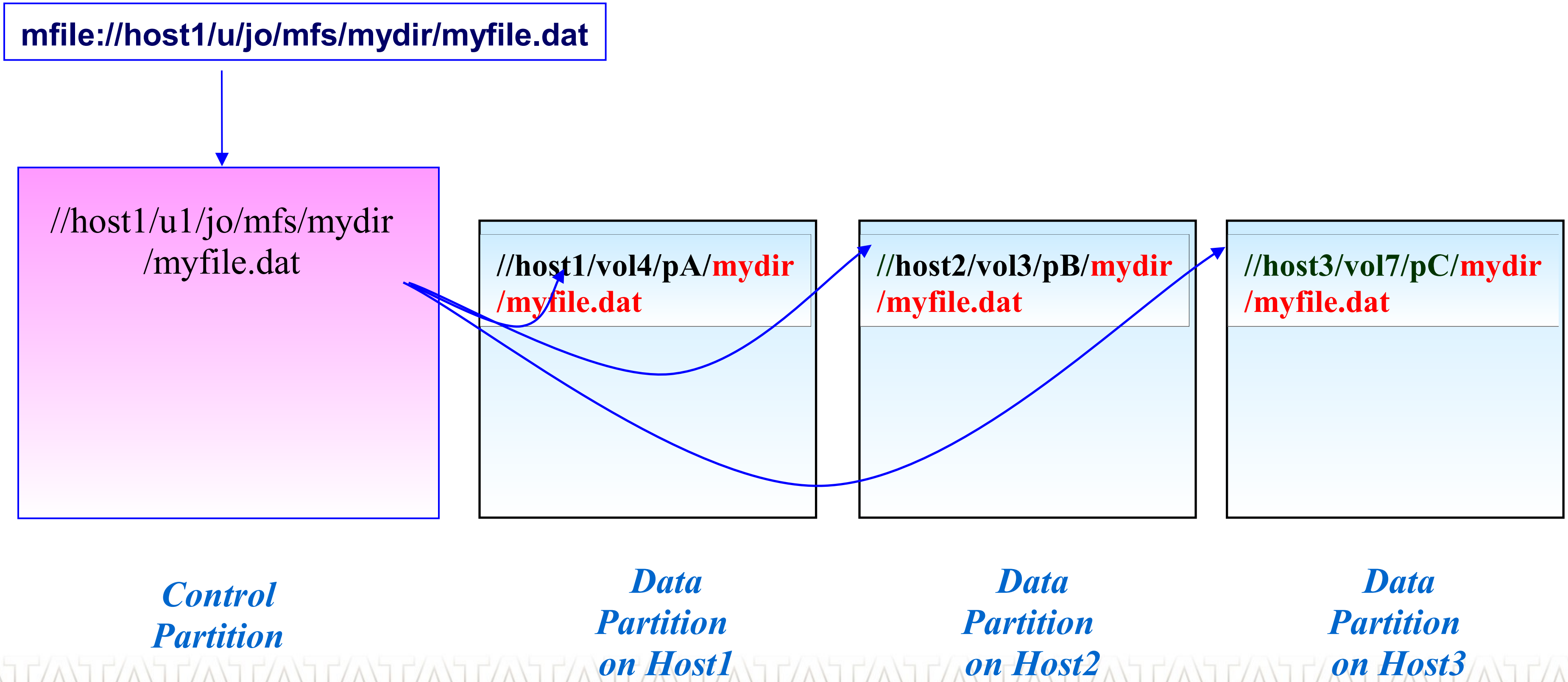
A Multidirectory

A directory spanning across partitions on different hosts



A Multifile

A file spanning across partitions on different hosts



mfile://host1/u/jo/mfs/mydir/myfile.dat

//host1/u1/jo/mfs/mydir
/myfile.dat

**//host1/vol4/pA/mydir
/myfile.dat**

**//host2/vol3/pB/mydir
/myfile.dat**

**//host3/vol7/pC/mydir
/myfile.dat**

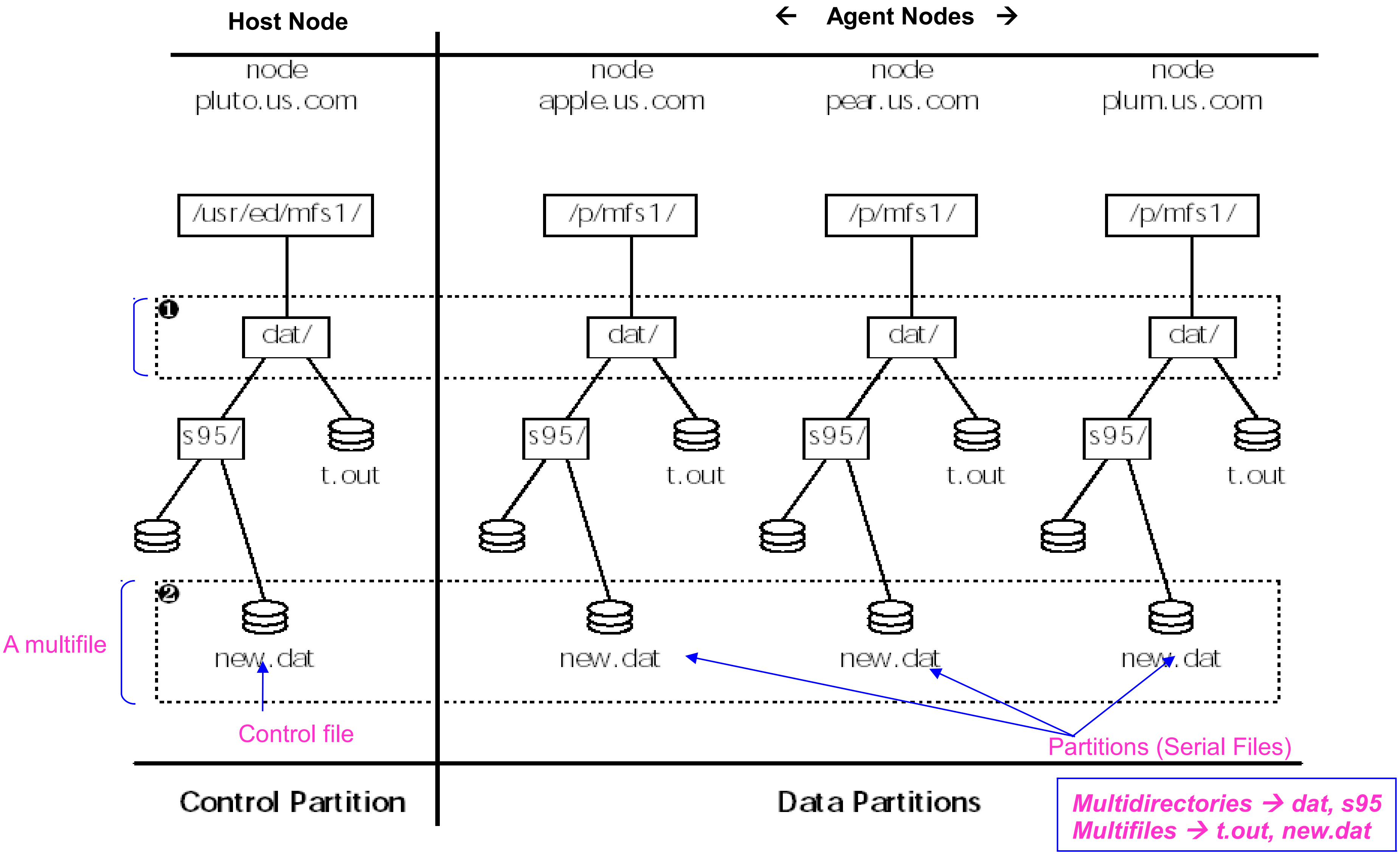
Control Partition

Data Partition on Host1

Data Partition on Host2

Data Partition on Host3

A Sample multifile system



Multidirectories → dat, s95
Multifiles → t.out, new.dat

Parallelism

Parallel Runtime Environment

Where some or all of the components of an application – datasets and processing modules are replicated into a number of partitions, each spawning a process.

Ab Initio can process data in parallel runtime environment

Forms of Parallelism

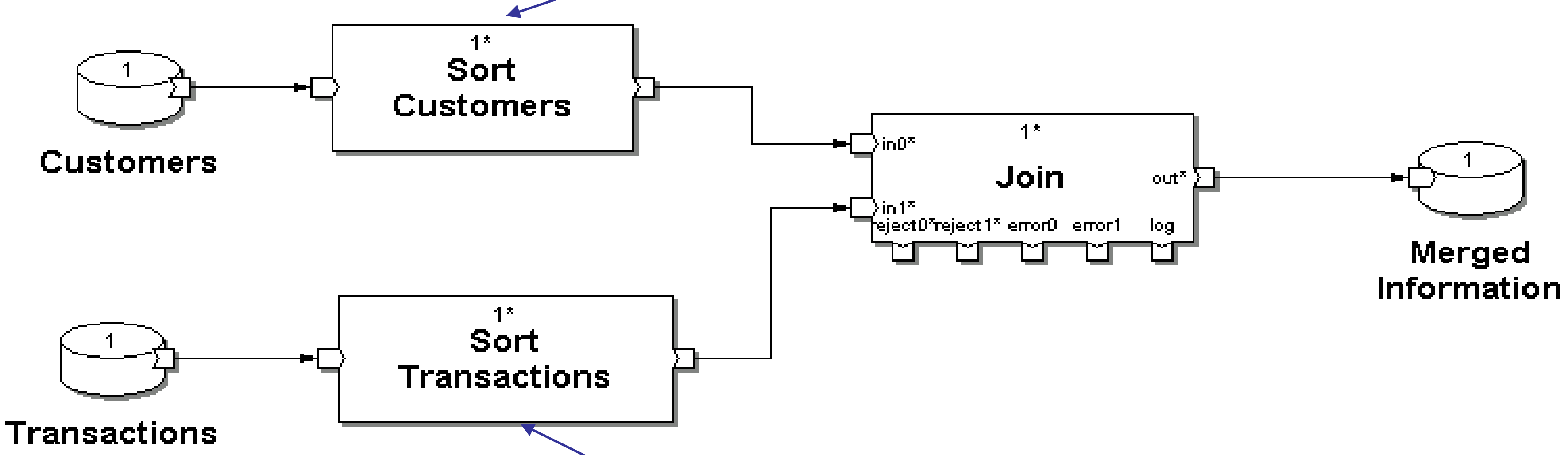
- Component Parallelism
- Pipeline Parallelism
- Data Parallelism

Inherent in Ab Initio

Component Parallelism

When different instances of same component run on separate data sets

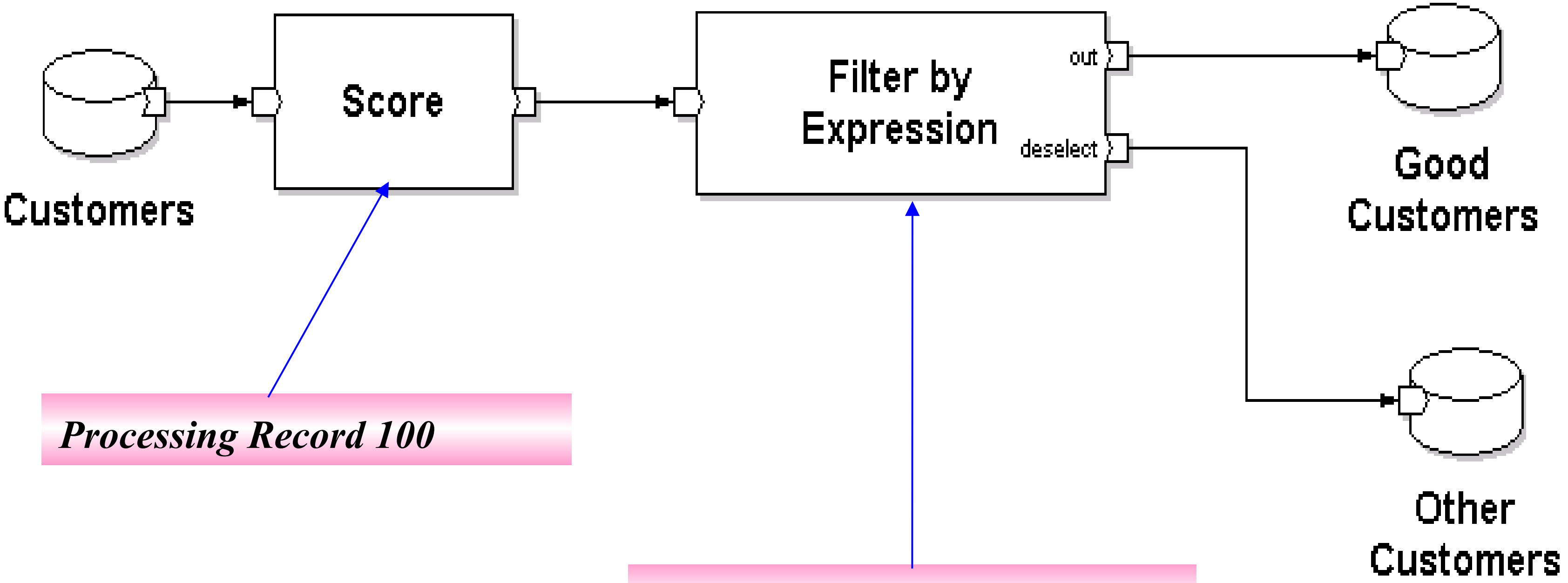
Sorting Customers



Sorting Transactions

Pipeline Parallelism

When multiple components run on same data set



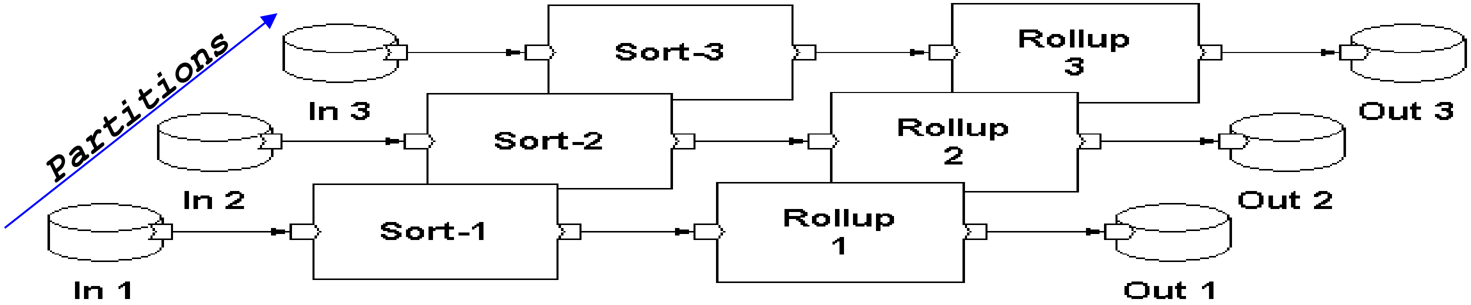
Processing Record 100

Processing Record 99

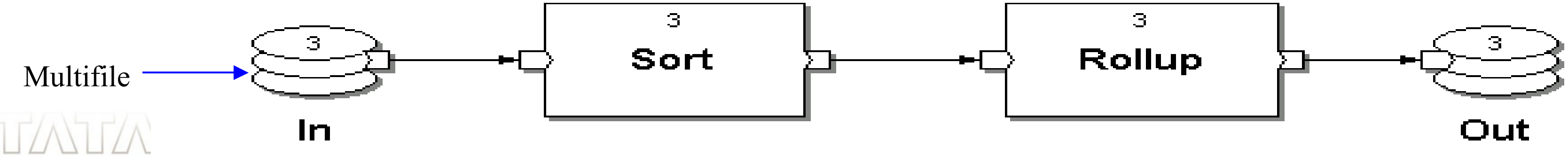
Data Parallelism

When data is divided into segments or *partitions* and processes run simultaneously on each *partition*

Expanded View



Global View



NOTE : # of processes per component = # of partitions

Data parallelism features

- Data parallelism scales with data and requires data partitioning
- Data can be partitioned using different partitioning methods.
- The actual way of working in a parallel runtime environment is transparent to the application developer.
- It can be decided at runtime whether to work in serial or in parallel, as well as to determine the degree of parallelism

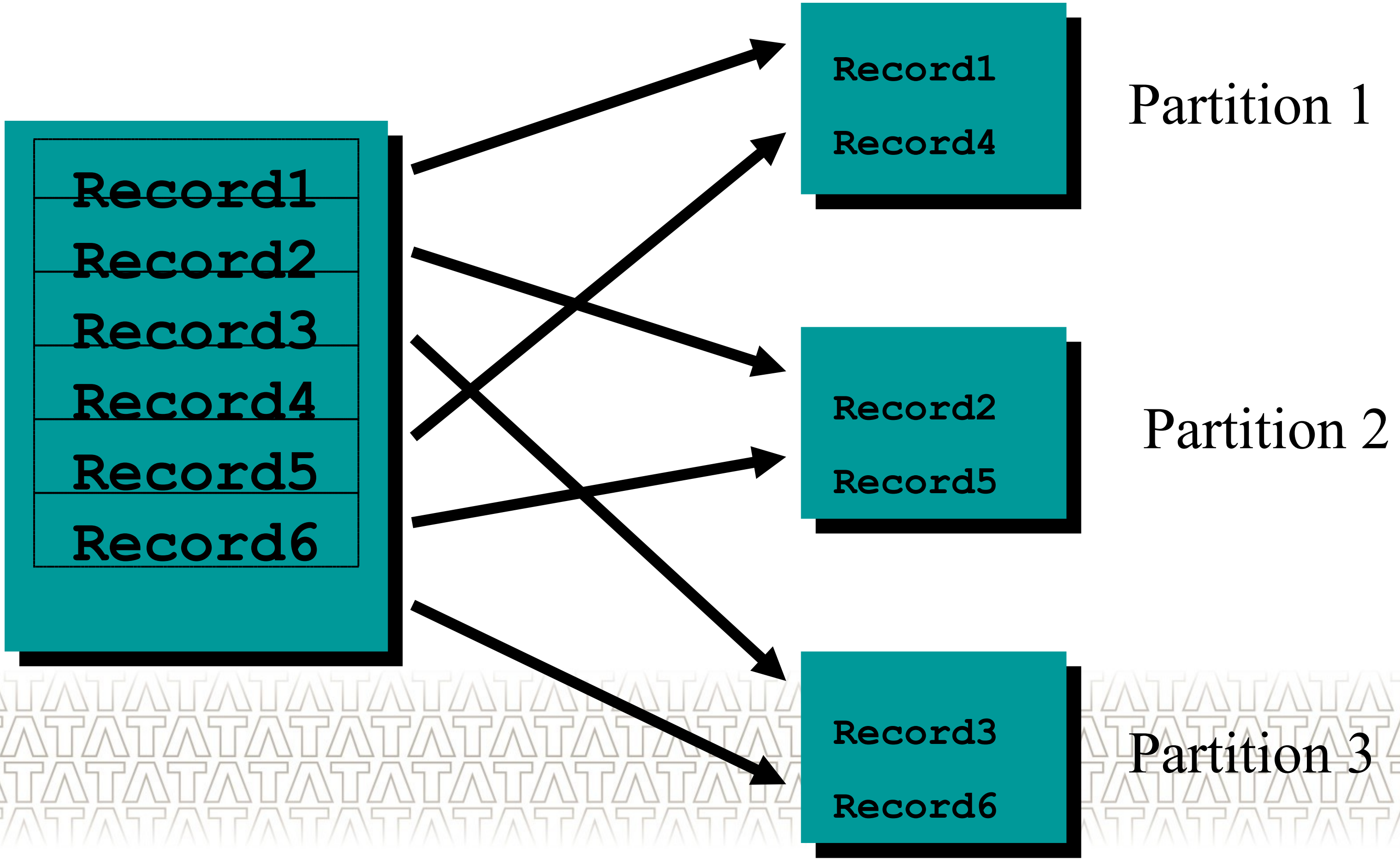
Data Partitioning Components

Data can be partitioned using

- Partition by Round-robin
- Partition by Key
- Partition by Expression
- Partition by Range
- Partition by Percentage
- Broadcast
- Partition by Load Balance

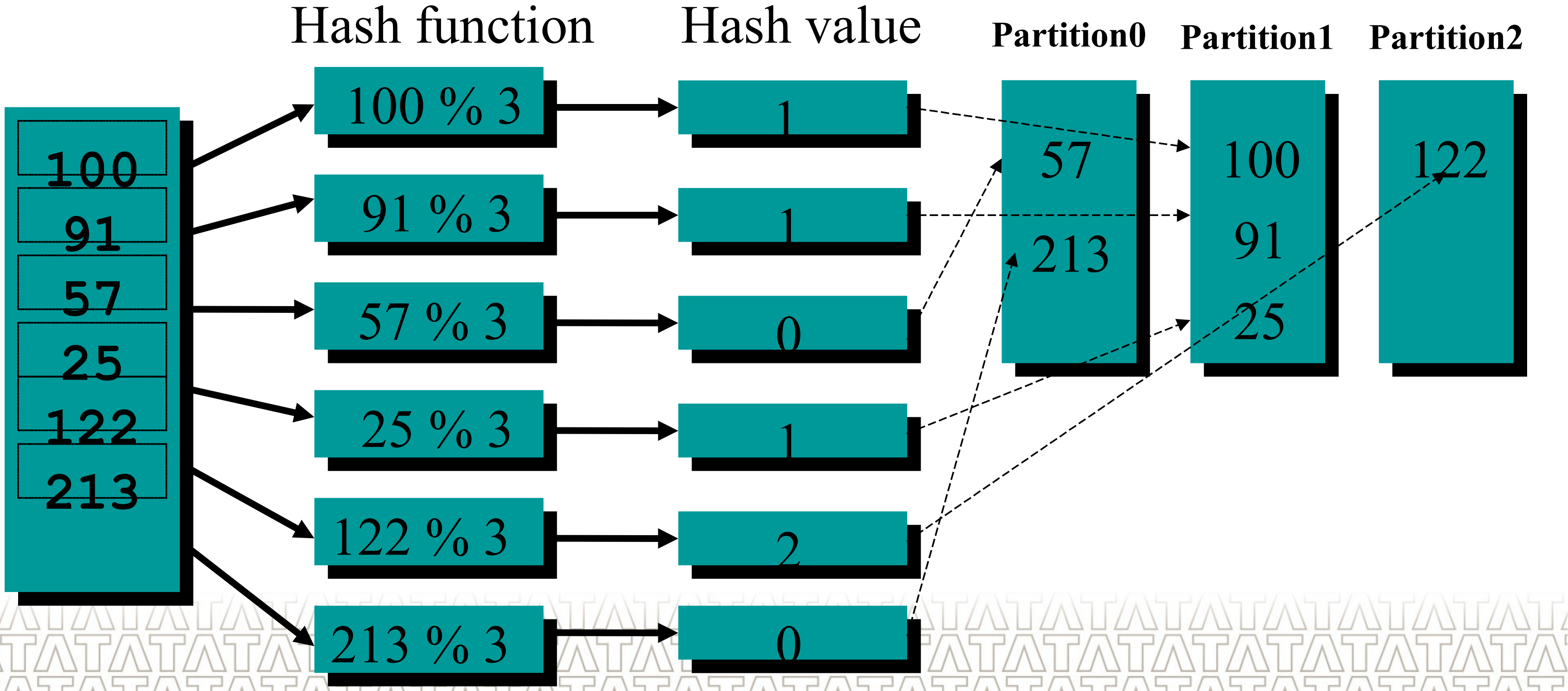
Partition by Round-robin

- Writes records to each partition evenly
- Block-size records go into one partition before moving on to the next.



Partition by Key

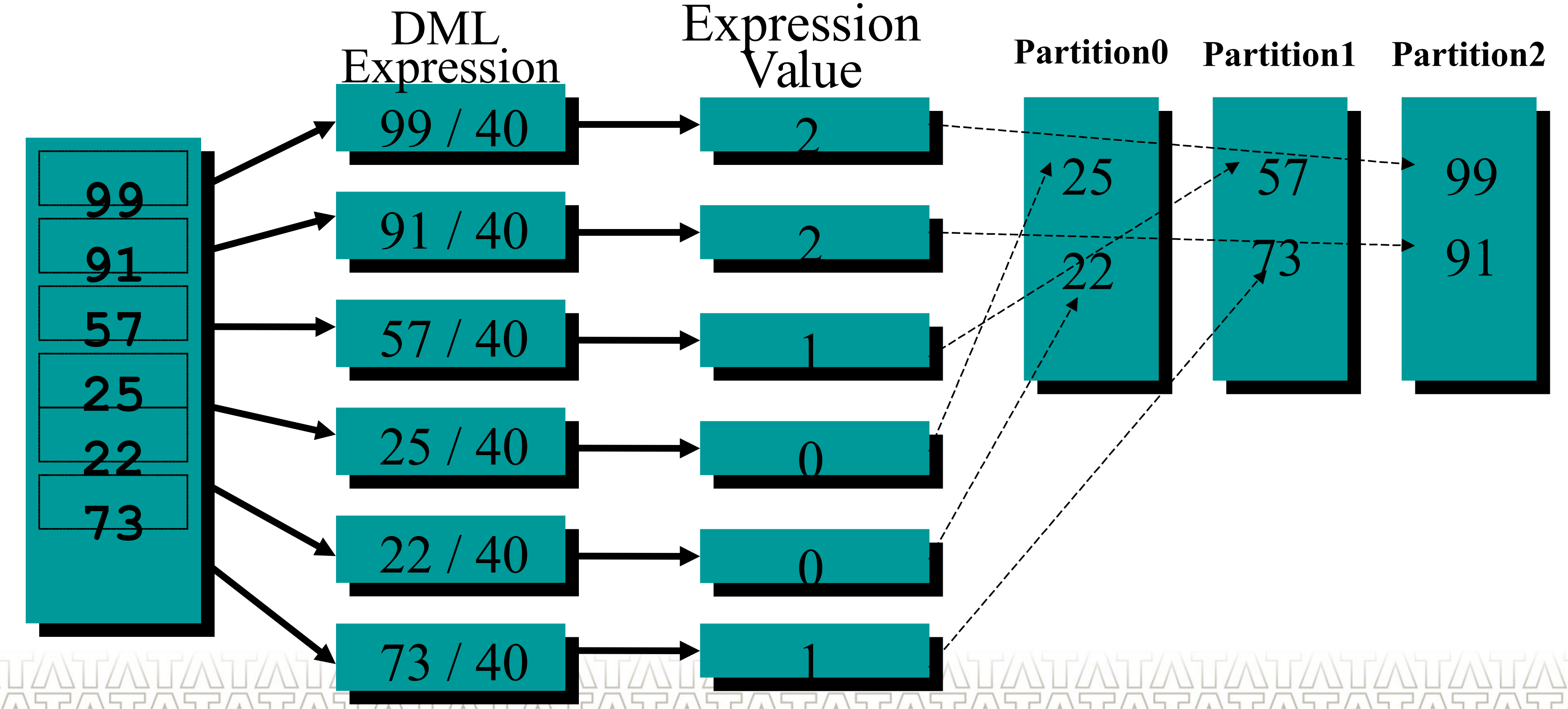
- Distributes data records to its output flow partitions according to key values



- Data may not be evenly distributed across partitions

Partition by Expression

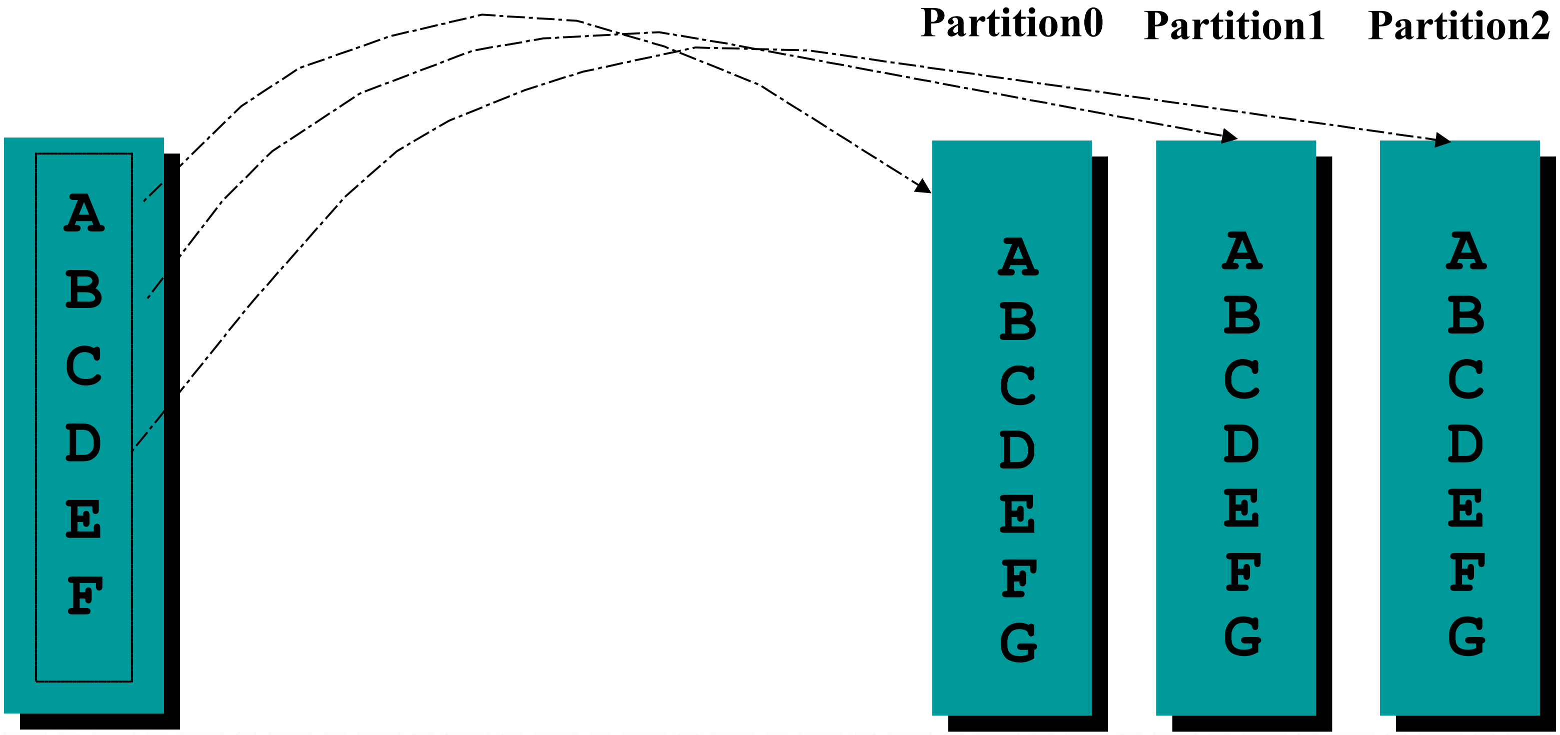
- Distributes data records to partitions according to DML expression values



- Does not guarantee even distribution across partitions
- Cascaded Filter by Expressions can be avoided

Broadcast

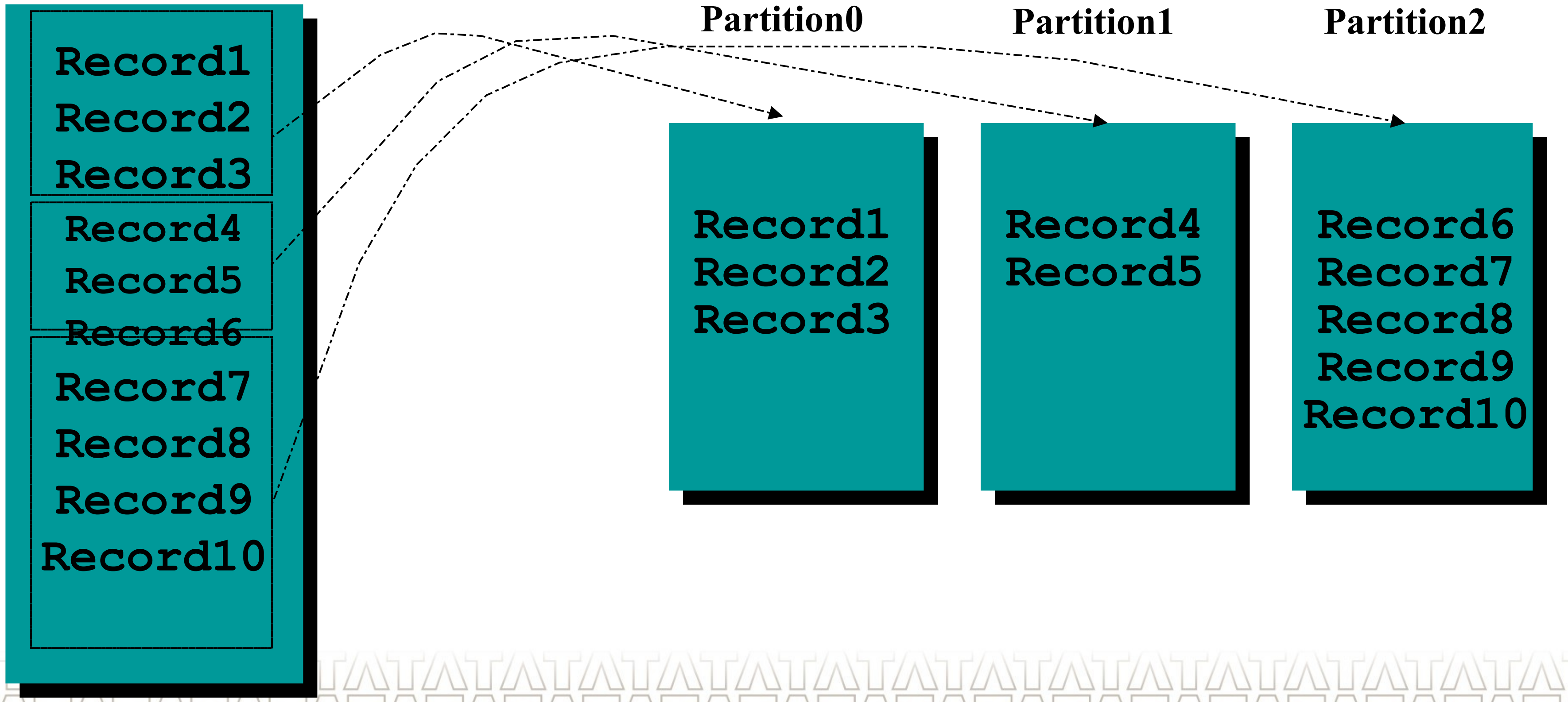
- Combines all data records it receives into a single flow
- Writes copy of that flow into each output data partition



- Increases data parallelism when connected single fan-out flow to out port

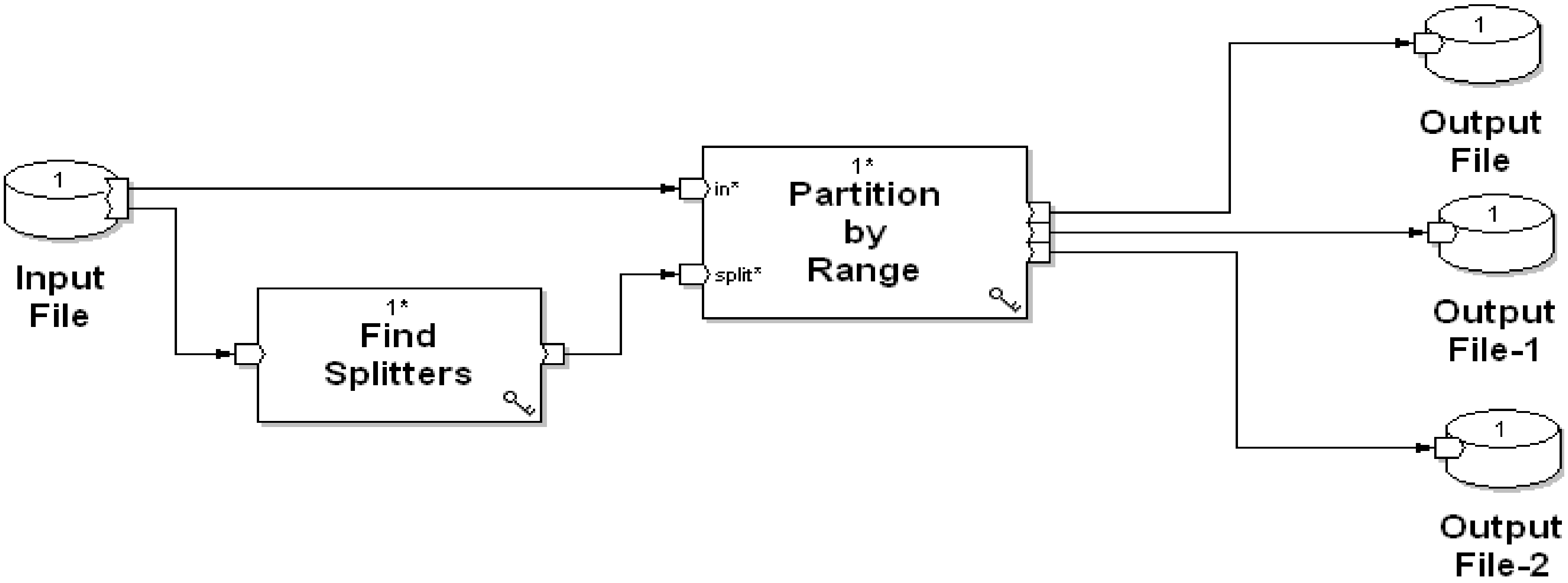
Partition by Percentage

- Distributes a specified percentage of the total number of input data records to each output flow



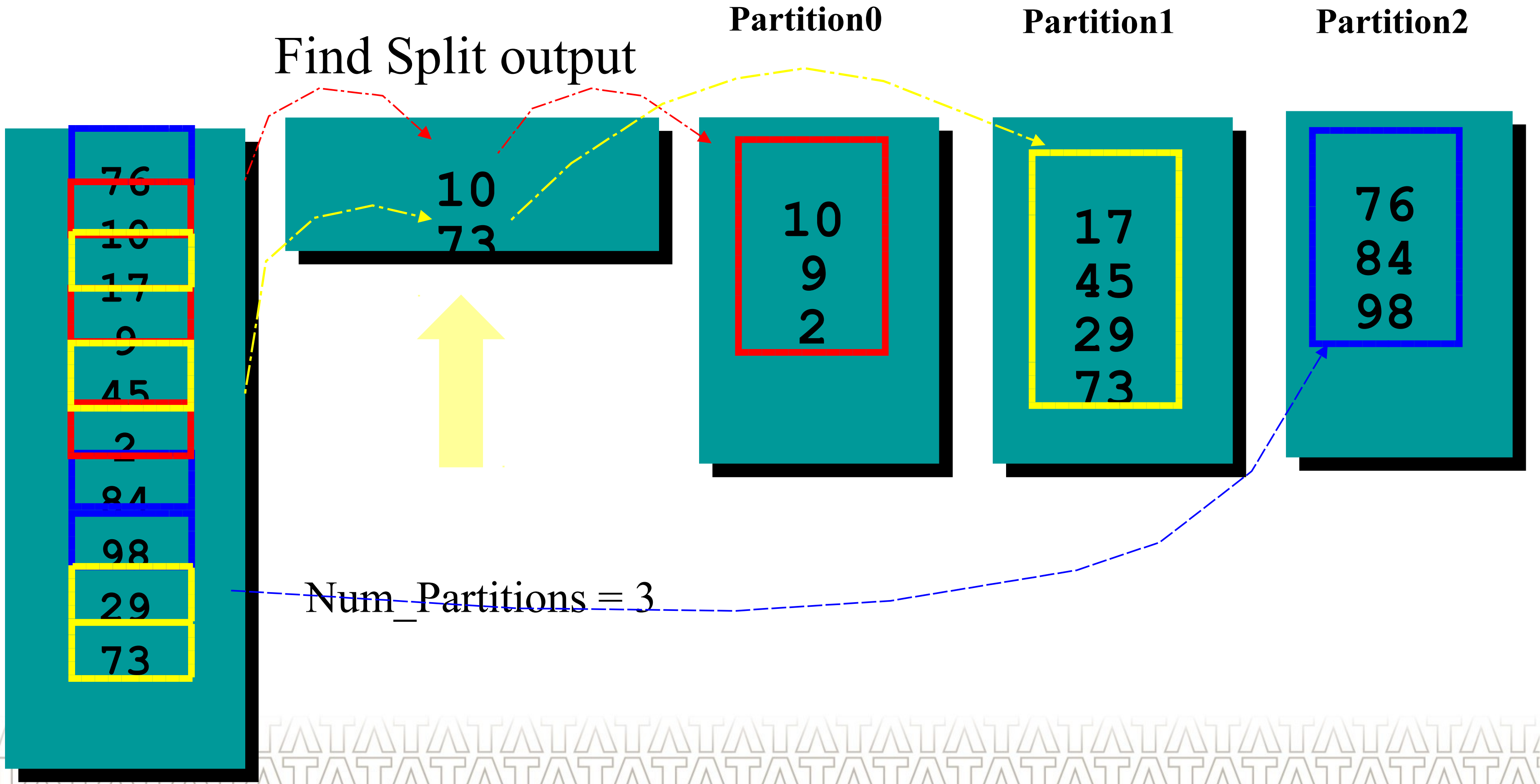
Partition by Range

- Distributes data records to its output flow partitions according to the ranges of key values specified for each partition.
- Typically used in conjunction with Find Splitter component for better load balancing



- Key range is passed to the partitioning component through its split port

Partition by Range



- Key values greater than 73 go to partition 2

Summary of Partitioning Methods

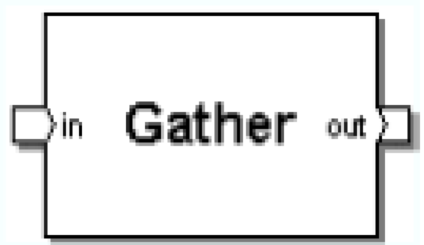
Method	Key-Based	Balancing	Uses
Round-robin	No	Even	Record-independent parallelism
Partition by Key	Yes	Depends on the key value	Key-dependent parallelism
Partition by Expression	Yes	Depends on data and expression	Application specific
Broadcast	No	Even	Record-independent parallelism
Partition by Percentage	No	Depends on the percentage specified	Application specific
Partition by Range	Yes	Depends on splitters	Key-dependent parallelism, Global Ordering

Departitioning Components

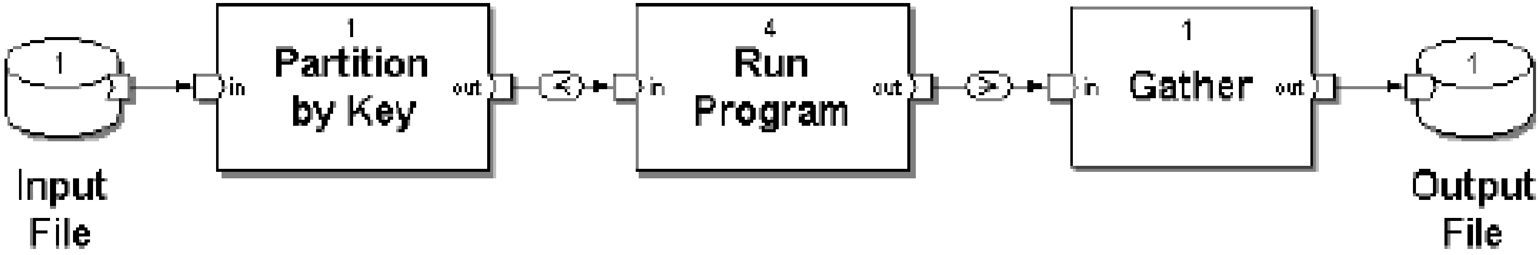
- **Gather**
- **Concatenate**
- **Merge**
- **Interleave**

Departitioning Components

- **Gather**



  - Reads data records from the flows connected to the input port
  - Combines the records arbitrarily and writes to the output



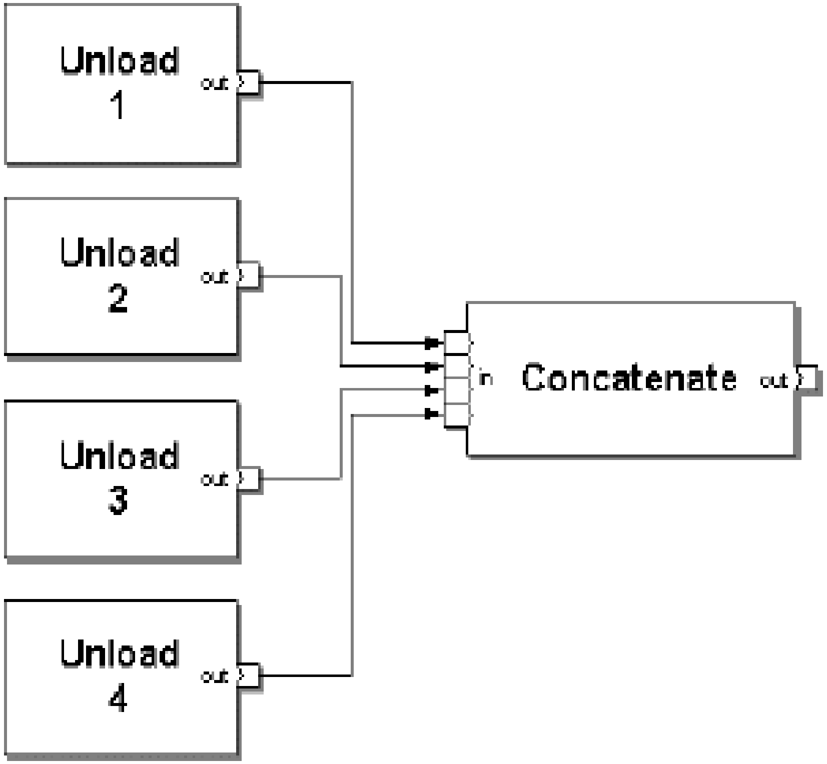
  - Combines data records from multiple flow partitions that have been sorted on a key
  - Maintains the sort order

Concatenate

in
Concatenate
out

Concatenate appends multiple flow partitions of data records one after another

Concatenate



- Reads the flows in the order in which you connect to them to in port
- In above Graph, concatenate reads first Unload 1, then Unload 2 and so on
- **Parameters:** None

Merge

in
Merge
out

- Combines data records from multiple flow partitions that have been sorted on a key

- Maintains the sort order

- **Parameters of Merge Component:**
  - **key :** Name of he key fields and the sequence specifier you want Merge to use to maintain the order of data records while merging them

Interleave

in
Interleave
out

- Combines blocks of records from multiple flow partitions in round-robin fashion
- Reads number of records specified in blocksize from first flow then from second flow and so on
- Writes the records to the out port

- **Parameters of Interleave Component :**
  - **Blocksize:** number of data records Interleave reads from each flow before reading the same number of data records from the next flow.

Departitioning Components

- **Summary of Departitioning Methods**

Method	Key-Based	Ordering	Uses
Concatenate	No	Global	Creating serial flows from partitioned data
Interleave	No	Inverse of Round Robin partition	Creating serial flows from partitioned data
Merge	Yes	Sorted	Creating ordered serial flows
Gather	No	Arbitrary	Unordered departitioning

Case Studies

Case Study 1

In a shop, the customer file, contains the following fields:

Field Name	Data Type	Length/Delimiter	Format/Mask
Cust_id	Decimal	“| ” (pipe)	None
amount	Decimal	“\n”(newline)	None

Here are some sample data for customer file:

Cust_id	amount
215657	1000
462310	1500
462310	2000
215657	2500
462310	5500
215657	4500

Develop the AbInitio Graph, which will do the following:

It takes the first three records of each Cust_id and sum the amounts, the output file is as follows -

Field Name	Data Type	Length/Delimiter	Format/Mask
Cust_id	Decimal	"|"(pipe)	None
Total_amount	Decimal	"\n"(newline)	None

Where total_amount is the sum of first three records for each Cust_id.

Case Study 2

Consider the following BP_PRODUCT file , containing the following fields :

Field Name	Data Type	Length/Delimiter	Format/Mask
product_id	Decimal	“|”(pipe)	None
product_code	String	“|”(pipe)	None
plan_details_id	Decimal	“|”(pipe)	None
plan_id	Decimal	“|”(pipe)	None

Here are some sample data for the BP_PRODUCT file :

product_id	product_code	plan_details_id	plan_id
147	OPS	11111	111
154	NULL	12121	222
324	VB	12312	111
148	PCAT	23412	999
476	VB	34212	666

Develop the AbInitio Graph, which will do the following:

Firstly filtered out those records where product_code is NULL.

Then save the data in three output file, where
First output file contains records having product_code OPS, second having PCAT, third having VB.

Case Study 3

In a retail shop, the customer_master file, contains the details of all the existing customers. It consists of the following fields:

Field Name	Data Type	Length/Delimiter	Format/Mask
Cust_id	String	“|”(pipe)	None
Cust_name	String	“|”(pipe)	None
cust_address	String	“|”(pipe)	None
newline	None	“\n”(newline)	None

Sample data of customer_master file:

Cust_id	Cust_name	Cust_address
215657	S Chakraborty	Saltlake
462310	J Nath	Kolkata
124343	D Banerjee	Kolkata
347492	A Bose	Kolkata
560124	C Tarafdar	Kolkata
439684	W Ganguly	Durgapur

An input file is received on daily basis detailing all the transactions of that day. The file contains the following fields:

Field Name	Data Type	Length/Delimiter	Format/Mask
Cust_id	String	"|"(pipe)	None
Cust_name	String	"|"(pipe)	None
cust_address	String	"|"(pipe)	None
purchase_date	Date	"|"(pipe)	"YYYYMMDD"
product_name	String	"|"(pipe)	None
quantity	number	4	None
amount	number	8	None
new_line	none	"\n"(newline)	none

- Sample data of the file :

Cust_id	Cust_name	Cust_address	Purchase_date	Product _name	quantity	amount
215657	Chakraborty	Nagerbazar	20060626	P1	1	1000
462310	J Nath	Kolkata	20060626	P3	2	5000
124343	D Banerjee	Kolkata	20060626	P43	3	2123

Develop an ab initio graph that will accept the input transaction details file and do the following:

1) If it is a new customer record, then insert the details in the output file.
2) If it is an existing customer record and Cust_address has not been changed, then do nothing
3) If it is an existing customer record and the Cust_address has been changed, then update it in the output file

The output file will contain the following fields:

Field Name	Data Type	Length/Delimiter	Format/Mask
Cust_id	String	“|”(pipe)	None
Cust_name	String	“|”(pipe)	None
cust_address	String	“|”(pipe)	None
Purchase_date	number	“|”(pipe)	“YYYYMMDD”
product_name	String	“|”(pipe)	None
Total_sales	number	“|”(pipe)	none
newline	None	“\n”(newline)	None

Where total_sales = Quantity * Amount ;

Queries???

TATA CONSULTANCY SERVICES

TATA

Thank You!!!

bipm.services@tcs.com

Experience certainty.

IT Services
Business Solutions
Outsourcing