Deciding the Physical Implementation of ETL Workflows

Vasiliki Tziovara

Panos Vassiliadis

Univ. of Ioannina

Alkis Simitsis

Almaden Research Center

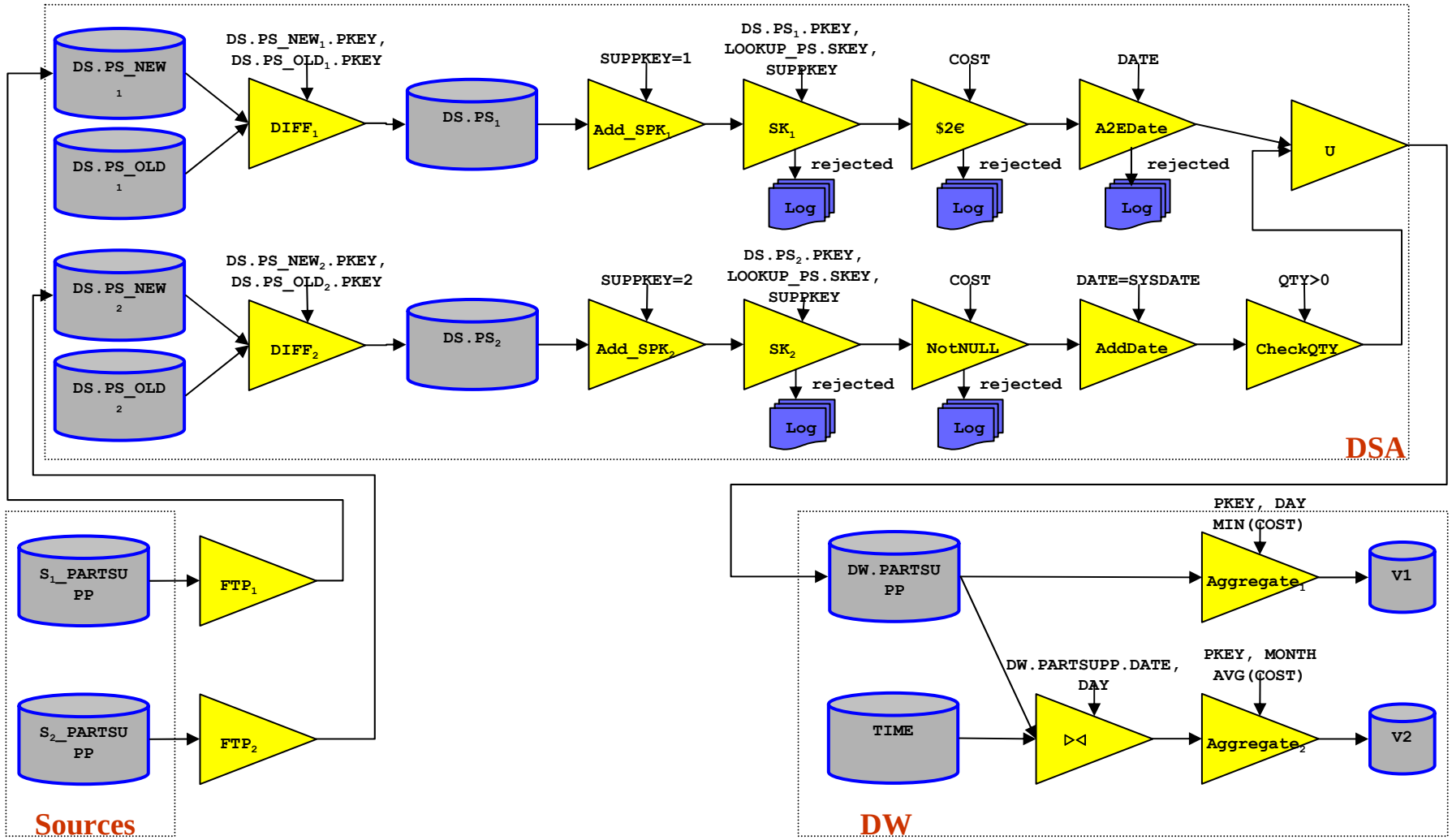
Roadmap

- Background & problem formulation
- General solutions & improvements
- Experiments & results
- Conclusions & future work

Roadmap

- Background & problem formulation
- General solutions & improvements
- Experiments & results
- Conclusions & future work

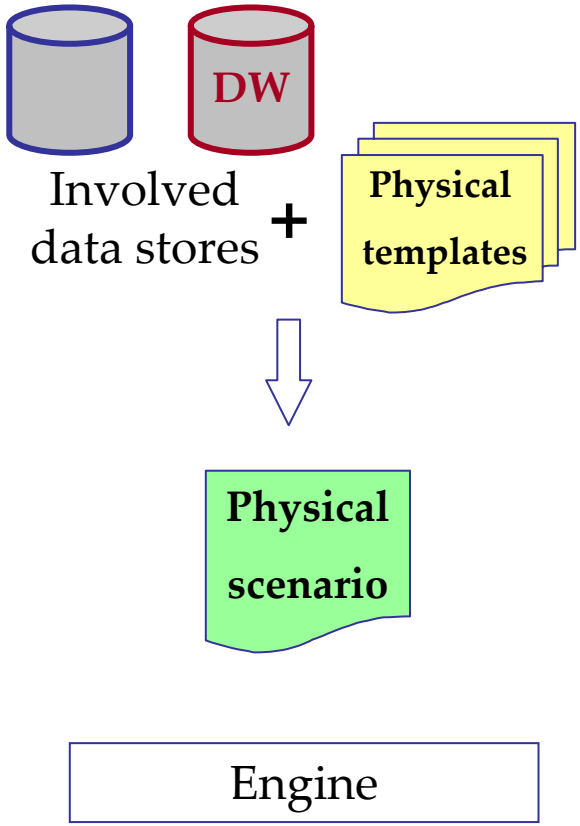
ETL workflows



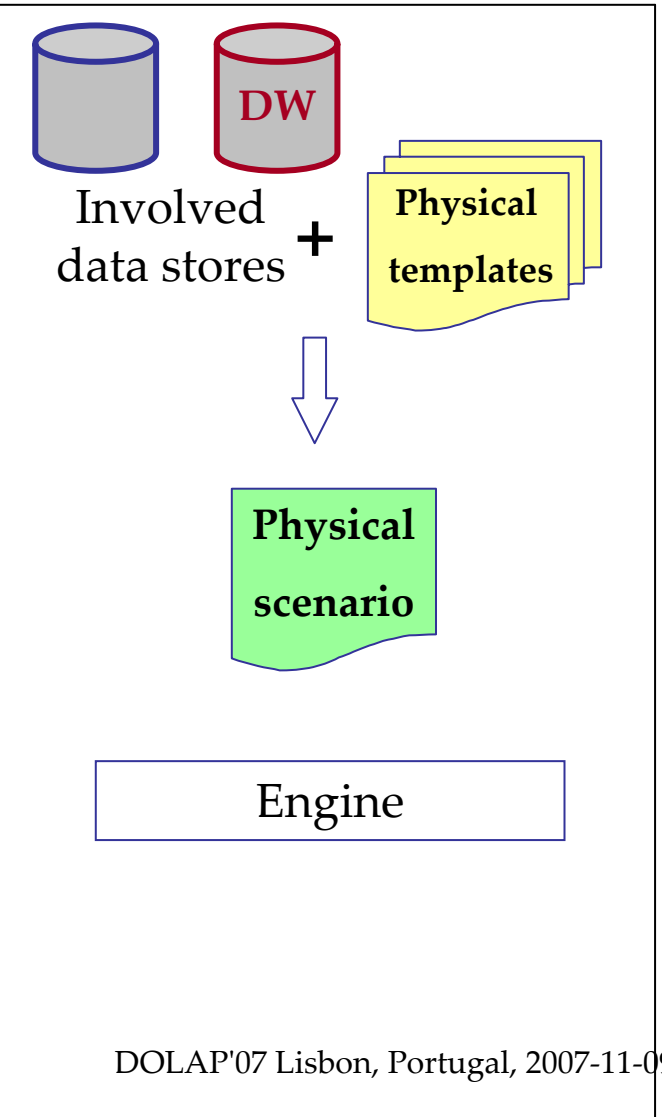
Fundamental research question

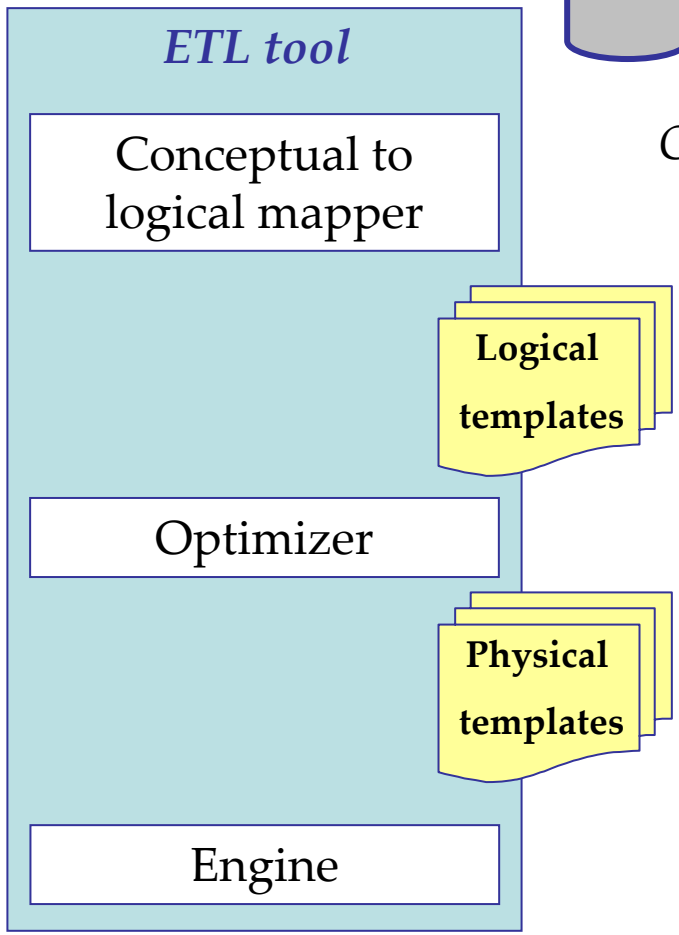
- Now: currently, ETL designers work directly at the physical level (typically, via libraries of physical-level templates)
- Challenge: can we design ETL flows as declaratively as possible?
- Detail independence:
  - no care for the algorithmic choices
  - no care about the order of the transformations
  - (hopefully) no care for the details of the inter-attribute mappings

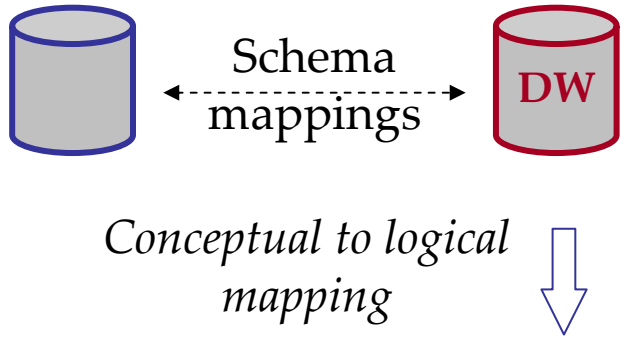
Now:


DW
Involved data stores
+
Physical templates
Physical scenario
Engine

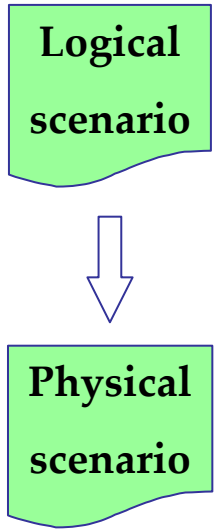
Vision:







Conceptual to logical mapping



Detail independence

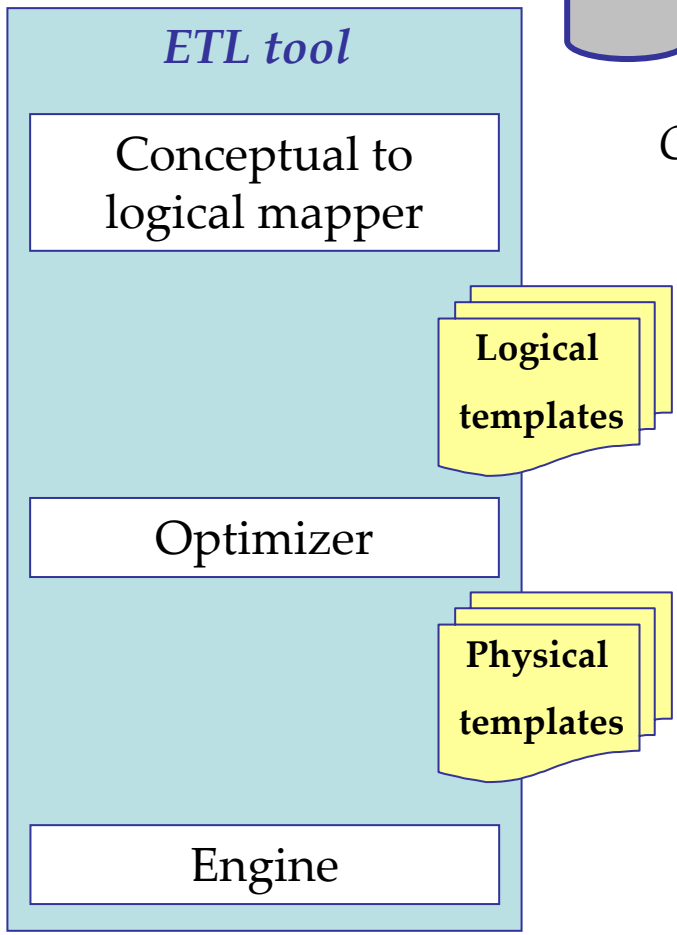
Automate

(as much as possible)

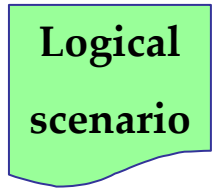
Conceptual: the details of the inter-attribute mappings

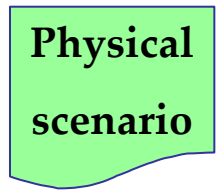
Logical: the order of the transformations

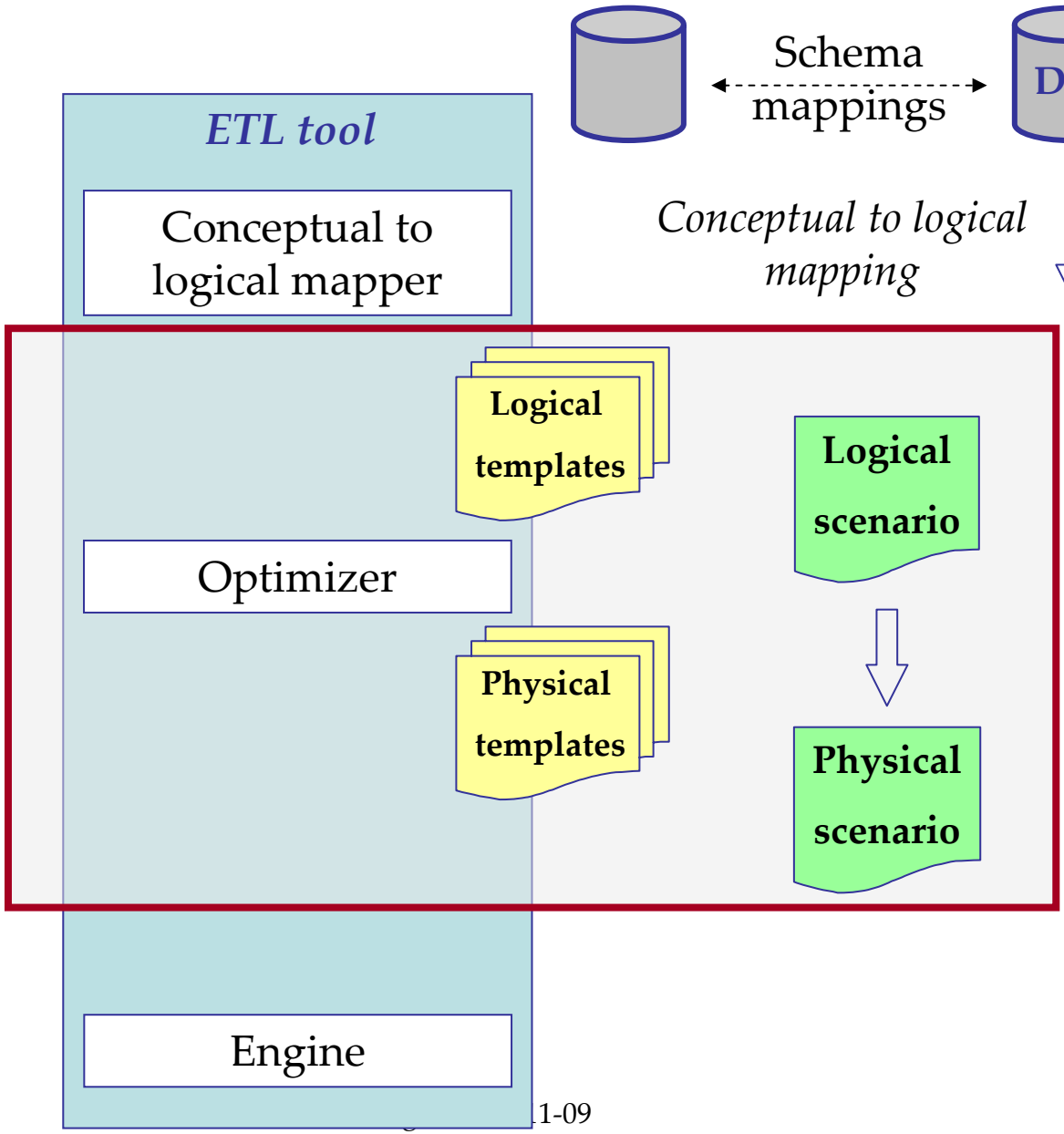
Physical: the algorithmic choices



Conceptual to logical mapping







"identify the best possible physical implementation for a given logical ETL workflow"

Problem formulation

- Given a logical-level ETL workflow G^L
- Compute a physical-level ETL workflow G^P
- Such that
  - the semantics of the workflow do not change
  - all constraints are met
  - the cost is minimal

Problem formulation

- Given a logical-level ETL workflow G^L
- Compute a physical-level ETL workflow G^P
- Such that
  - the semantics of the workflow do not change
  - all constraints are met
  - the cost is minimal

ETL workflows

- We model an ETL workflow as a directed acyclic graph **G(V,E)**.
  - Each node $v \in \mathbf{V}$ is either an activity a or a recordset r.
  - An edge $(a,b) \in \mathbf{E}$ denotes that b receives data from node a for further processing.

Templates

- Logical & physical **templates** of activities, aid the designer specify the scenario faster
- 1:N mapping of logical to physical mappings

LOGICAL – LEVEL **TEMPLATE**
1. Semantics (abstract): $\sigma_{\$1 > \$2}$
LOGICAL – LEVEL INSTANCE
1. Semantics (concrete): $\sigma_{A>50}$
PHYSICAL – LEVEL **TEMPLATE**
A. Order-aware implementation
Precondition (abstract): {$1 desc}
B. Order-free implementation
Precondition (abstract): {}
PHYSICAL – LEVEL INSTANCE
1. Semantics (concrete): $\sigma_{A>50}$
2. Precondition (concrete): {A desc}

Problem formulation

- Given a logical-level ETL workflow G^L
- Compute a physical-level ETL workflow G^P
- Such that
  - the semantics of the workflow do not change
  - all constraints are met
  - the cost is minimal

Semantics and constraints

- All recordsets, activities and provider links are mapped to their physical representations
  - Templates act as intermediaries here
- All preconditions are met
  - E.g., the input to a physical activity requiring a certain ordering of the incoming tuples, must obey the necessary ordering

Problem formulation

- Given a logical-level ETL workflow G^L
- Compute a physical-level ETL workflow G^P
- Such that
  - the semantics of the workflow do not change
  - all constraints are met
  - the cost is minimal

Cost model

- We employ a simple cost model

$$Cost(G) = \sum_{i=1}^{n} cost(a_i^P)$$

- For black-box activities, obtain cost per tuple via micro-benchmarks

$$cost_C(i) = m_i \times cpt(i)$$

Solution

- We model the problem of finding the physical implementation of an ETL process as a state-space search problem.
- **States**. A state is a graph $\mathbf{G^P}$ that represents a physical-level ETL workflow.
  - The initial state $\mathbf{G_0^P}$ is produced after the random assignment of physical implementations to logical activities w.r.t. preconditions and constraints.
- **Transitions**. Given a state $\mathbf{G^P}$, a new state $\mathbf{G^{P'}}$ is generated by replacing the implementation of a physical activity a^P of $\mathbf{G^P}$ with another valid implementation for the same activity.
  - Extension: **introduction of a sorter activity** (at the physical-level) as a new node in the graph.

Roadmap

- Background & problem formulation
- General solutions & improvements
- Experiments & results
- Conclusions & future work

Algorithmic alternatives

- Exhaustive approach to the simple variant problem (without any sorters introduced)
  - Straightforward
- Sorter introduction
  - Intentionally introduce sorters to reduce execution & resumption costs
- Not covered in this paper:
  - Heuristics
  - Failures as part of the problem

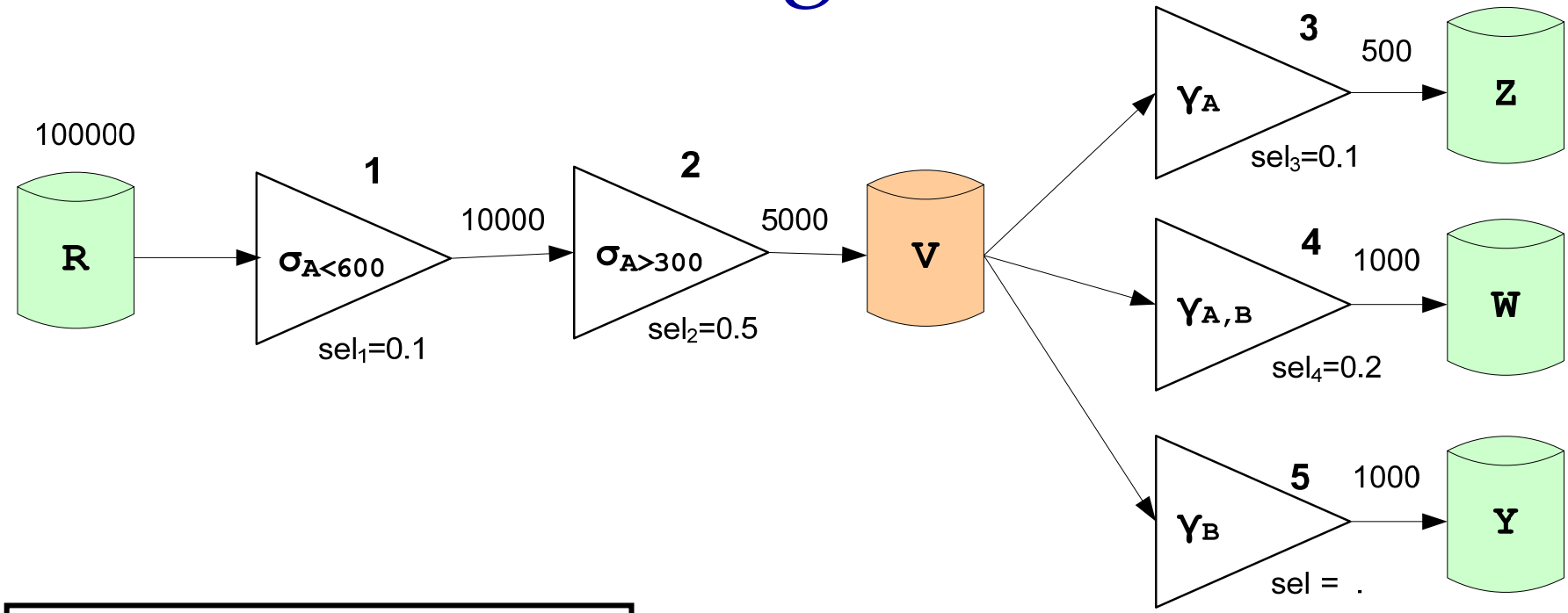
Sorters: impact

- *We intentionally introduce orderings, (via appropriate physical-level sorter activities) towards obtaining physical plans of lower cost.*

- Semantics: unaffected

- Price to pay:
  - cost of sorting the stream of processed data
- Gain:
  - it is possible to employ order-aware algorithms that significantly reduce processing cost
  - It is possible to amortize the cost over activities that utilize common useful orderings

Sorter gains



- Without order
  - $cost(\sigma_i) = n$
  - $cost_{SO}(\gamma) = n*\log_2(n)+n$
- With appropriate order
  - $cost(\sigma_i) = sel_i * n$
  - $cost_{SO}(\gamma) = n$

Cost(G) = 100.000+10.000
+3*[5.000*$\log_2$(5.000)+5.000] =
309.316

If sorter $S_{A,B}$ is added to V:

Cost(G') = 100.000+10.000
+2*5.000+[5.000*$\log_2$(5.000)+5.000]
= 247.877

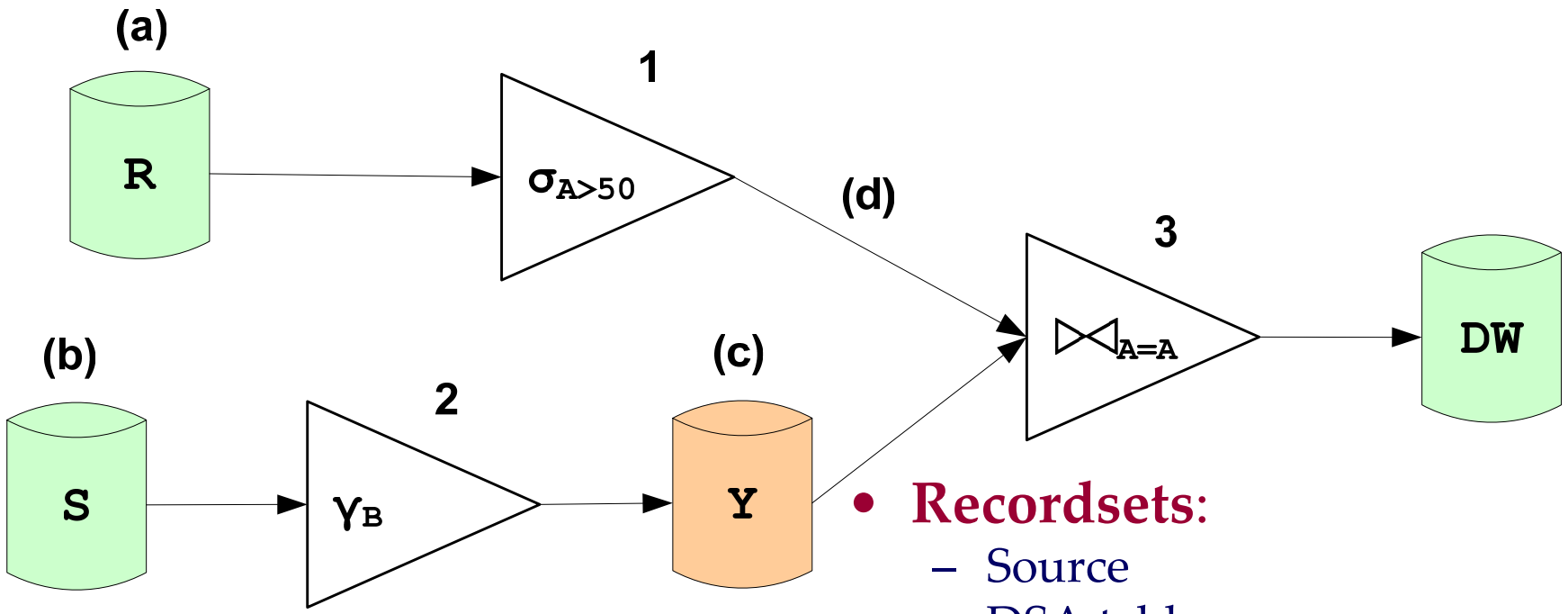
Sorters: the issues

- 3 main issues:
  - Candidate positions for introducing sorters?
  - Over which attributes should we order?
  - Ascending or descending order?

Candidate positions for sorters

- 3 possible positions
  - Source recordsets
  - DSA recordsets
  - Edges between activities

Candidate positions for sorters



- **Recordsets:**
  - Source
  - DSA table
- **Edges among activities**

  Here: positions (a)-(d)

Over which attributes should we order the data?

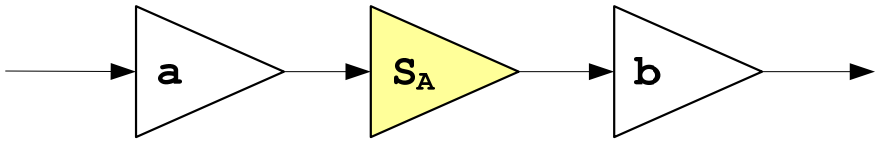
- Interesting order:
  - Traditionally: a set of attributes present in the join, grouping, and ordering conditions of a query.
  - Here: a set of attributes such that, an ordering of the data over them can lead to a cheaper evaluation plan for a query.

- Automatic derivation of interesting orders is possible with a little extra help at the template level ☺
  - For each order-aware template, we need to define a set of (parameterized) input attributes which act as a precondition for the template to be used.
  - In other words: if the incoming stream is sorted over these attributes, then the implementation can be used
  - The interesting order is this list of attributes

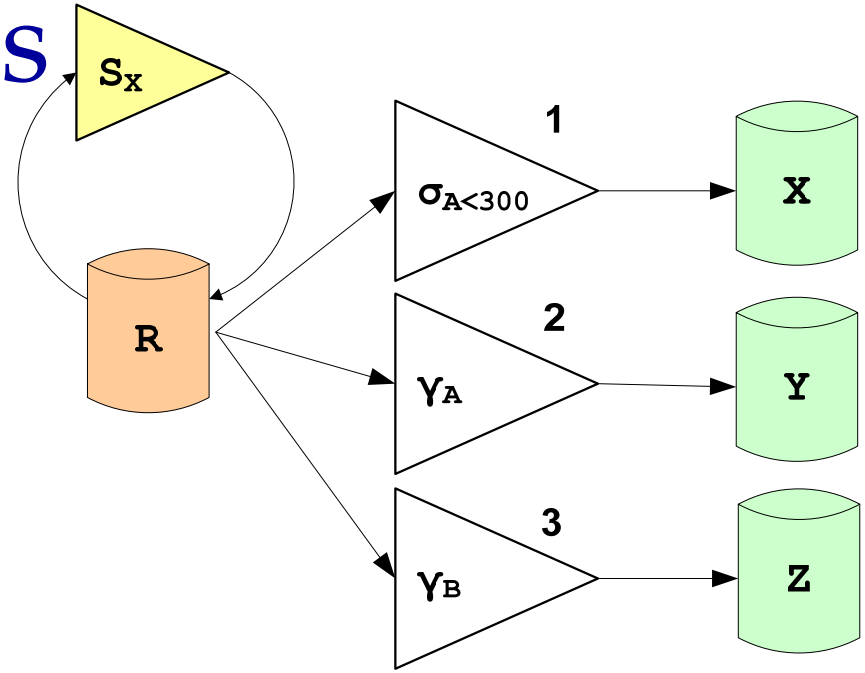
Interesting orders

- Interesting orders for sorters placed between subsequent activities a and b
  - the interesting orders of the implementations of b determine the ordering X imposed by the sorter S_X.
- Interesting orders for sorters placed over relations
  - Discover the interesting orders of the activities that receive data from the relation.
  - Combine all i.o.'s into a single set with each interesting order considered once in the set.

Interesting orders

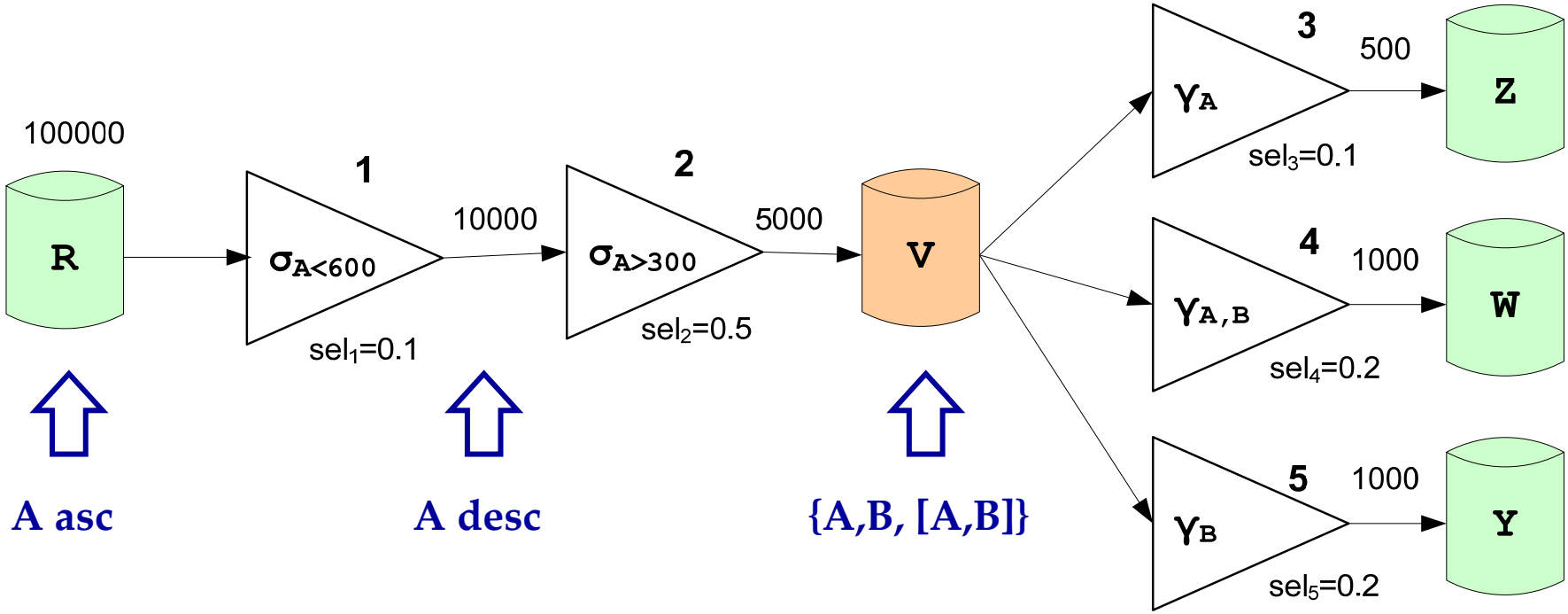


- **A** is defined by the interesting orders of b



- Decide interesting orders of activities receiving data from R
- Union $\{A_{asc}, A, B\}$
- Result: $\{A_{asc}, B\}$

Interesting orders



Interesting orders for ETL activities

- Can be defined at the template level!
- Customizable per scenario
  - See on the right
- Examples
  - Filters: selection condition attributes
  - Join variants: join attributes
  - Aggregation variants: aggregation attributes
  - Function: important parameters that can be defined at the template level

PHYSICAL – LEVEL TEMPLATE
1. Semantics (abstract): $\sigma_{\$1 > \$2}$ 2. Interesting Orders (abstract): {$1 desc}
PHYSICAL – LEVEL INSTANCE
1. Semantics (concrete): $\sigma_{A>50}$ 2. Interesting Orders (concrete): {A desc}

Ascending or descending orders?

- Depends on the semantics of the activity
  - $\sigma_{age > 40}$ requires a descending order
  - $\sigma_{age < 40}$ requires a ascending order

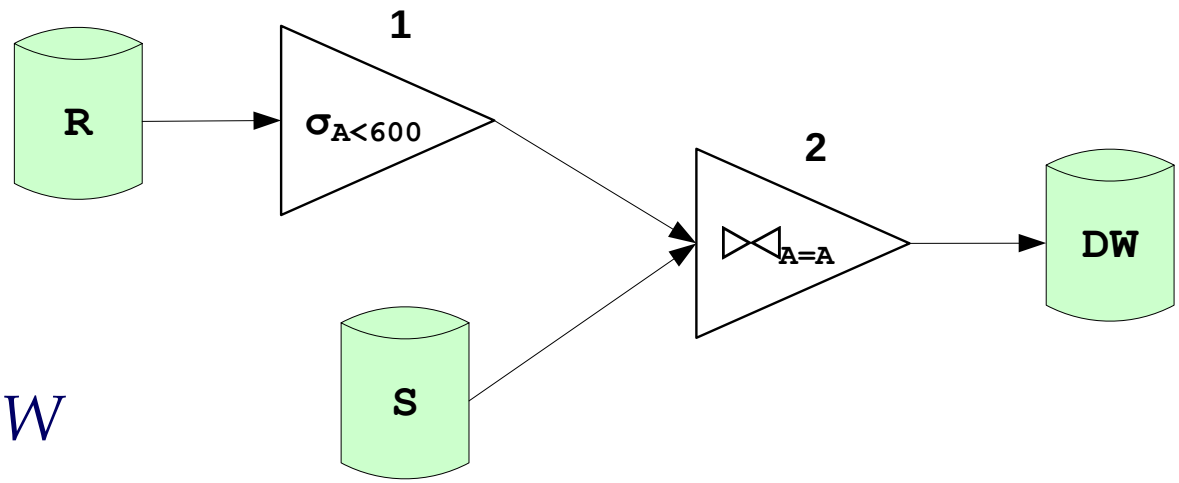
- Can be defined at the template level ☺

Algorithmic issues

- We have implemented a simple optimizer
  - Based on an exhaustive algorithm
  - Deals with all the aforementioned issues
  - Tries to save memory, via a compact representation of ETL scenarios: signatures

Signatures

- Strings that act as short representations of scenarios (for memory savings)
  - Consecutive nodes connect with “.”
  - Parallel paths connect with “//”. Each path is enclosed in parentheses.



((R.1)//(S)).2.DW

Exhaustive Ordering (EO)

- Input: a logical level graph G(V,E)
- Output: the signature with the minimum cost
  - $S_0 \leftarrow EGS(S)$;
  - $S_{MIN} = S_0$;
  - ∀ combination *c* of places for sorters
    - ∀ place *p*
      - ∀ possible order *o* in *p*
        - generate a new signature S';
        - Cost(S') ← Compute_cost(S');
        - If (Cost(S')<Cost(S_{MIN})) S_{MIN} = S';
  - return S_{MIN};

By EGS

By CPC

By GPO

By GPS

Roadmap

- Background & problem formulation
- General solutions & improvements
- Experiments & results
- Conclusions & future work

Problem in experimental setup

- When experimenting with ETL workflows what **test suites** should we use?

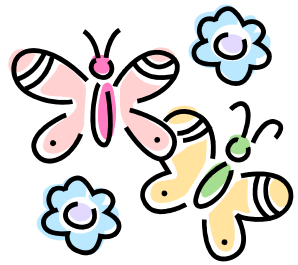
- We have faced the problem before:
  - Logical optimization of the ETL process (transposition of activities to speed up the workflow – ICDE05, TKDE05)

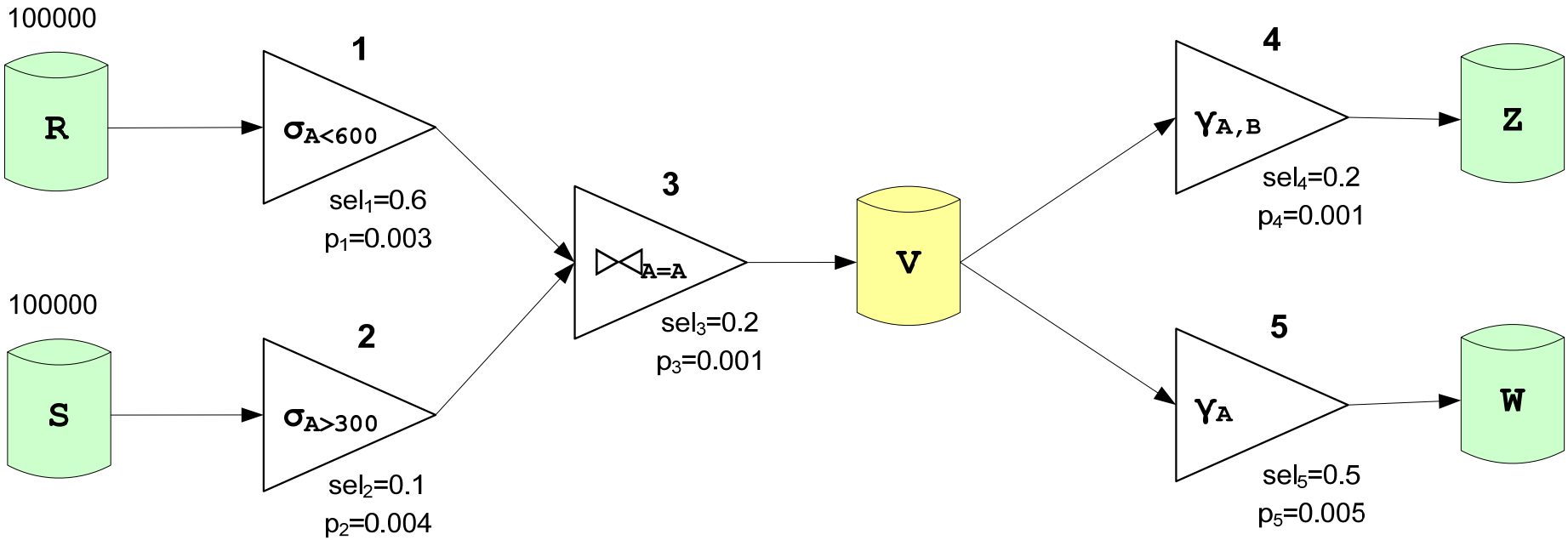
Problem in experimental setup

- Existing standards are insufficient
  - TPC-H
  - TPC-DS
- Practical cases are not publishable

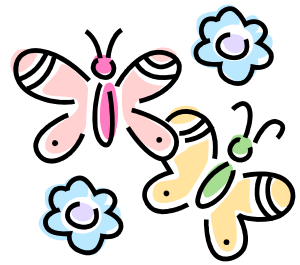
- We resort in devising our own ad-hoc test scenarios either through a specific set of scenarios which obey a common structural pattern

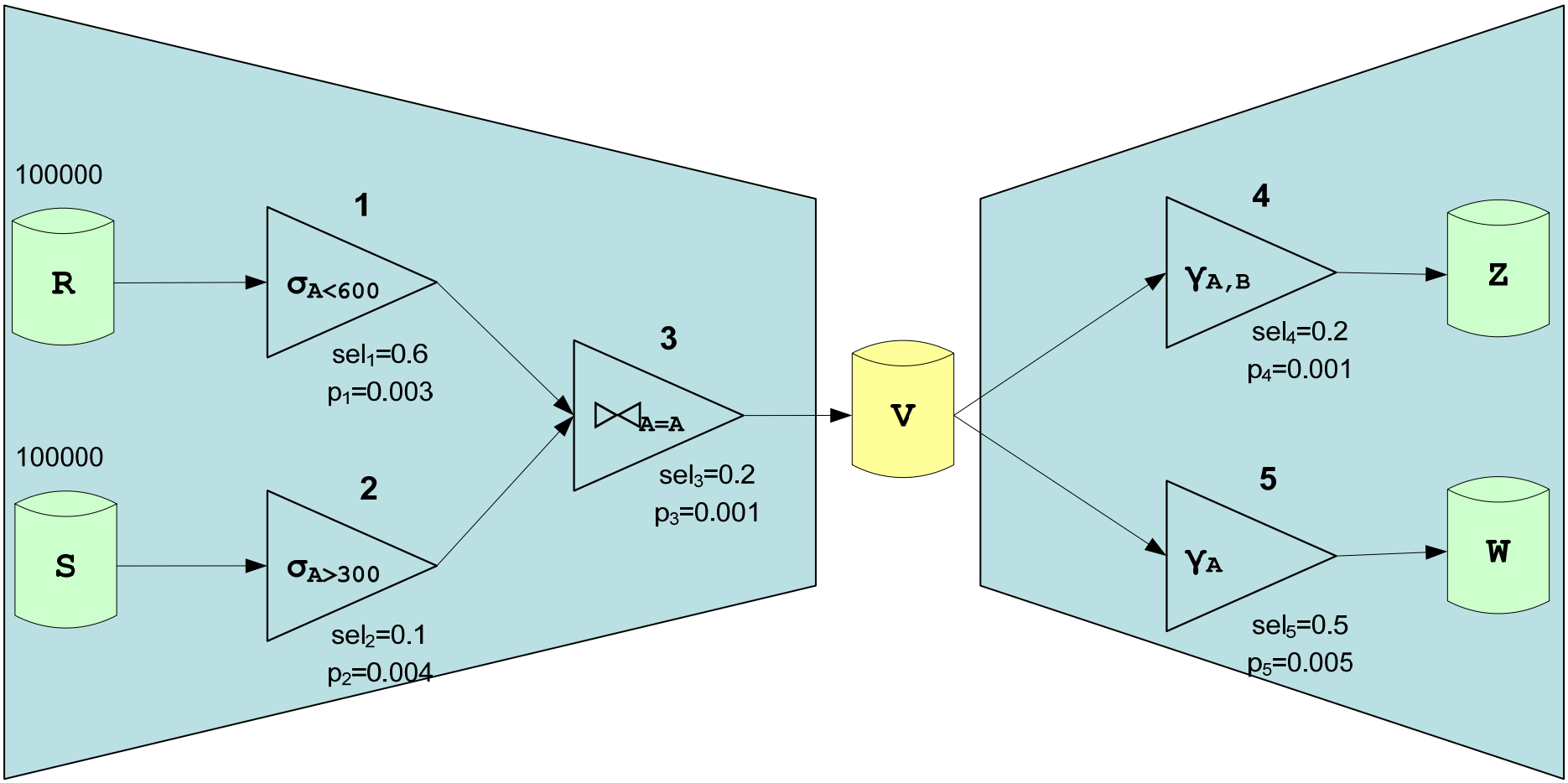
Butterflies to the rescue!



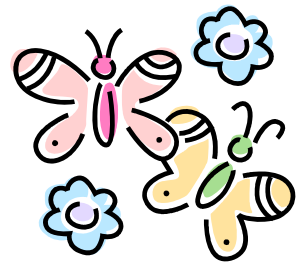


Butterflies to the rescue!





Butterflies to the rescue!



A **butterfly** is an ETL workflow that consists of three distinct components:

- **Body**: a central, detailed point of persistence (*fact* or *dimension* table) that is populated with the data produced by the left wing.
- **Left wing**: sources, activities and intermediate results. Performs extraction, cleaning and transformation + loads the data to the body.
- **Right wing**: materialized views, reports, spreadsheets, as well as the activities that populate them, to support reporting and analysis

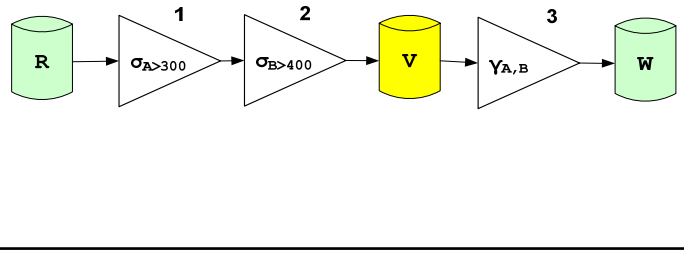
Butterfly classes

- Butterflies constitute a fundamental pattern of reference. Sub-components:
  - Line
  - Combinators
- Left-winged variants (heavy on the ETL part)
  - Primary flow
  - Wishbone
  - Tree
- Right winged variants (heavy on the "reporting" part)
  - Fork
- Irregular variants

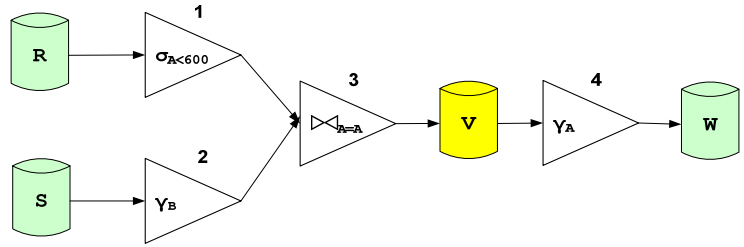
Details at Vassiliadis et al @ QDB 2007

(in conj. with VLDB 2007)

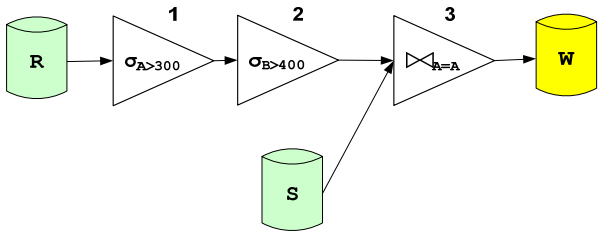
Butterfly classes



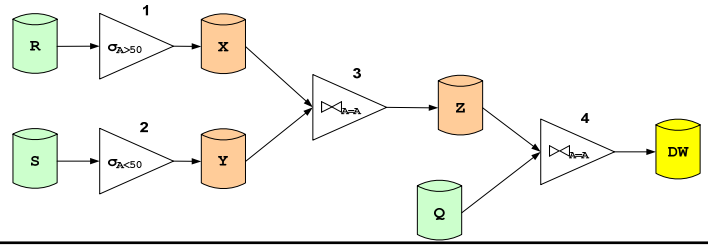
(a) Line



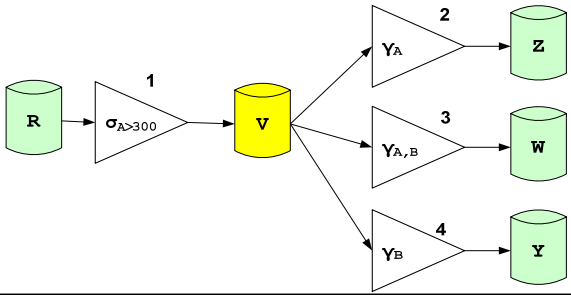
(b) Wishbone



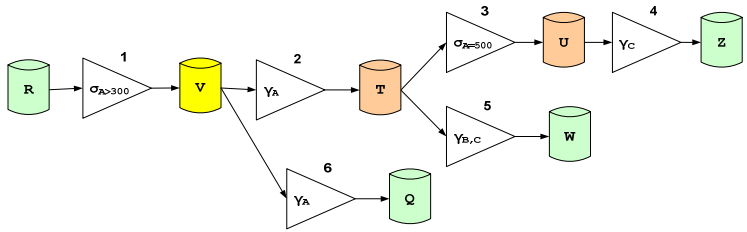
(c) Primary Flow



(d) Tree



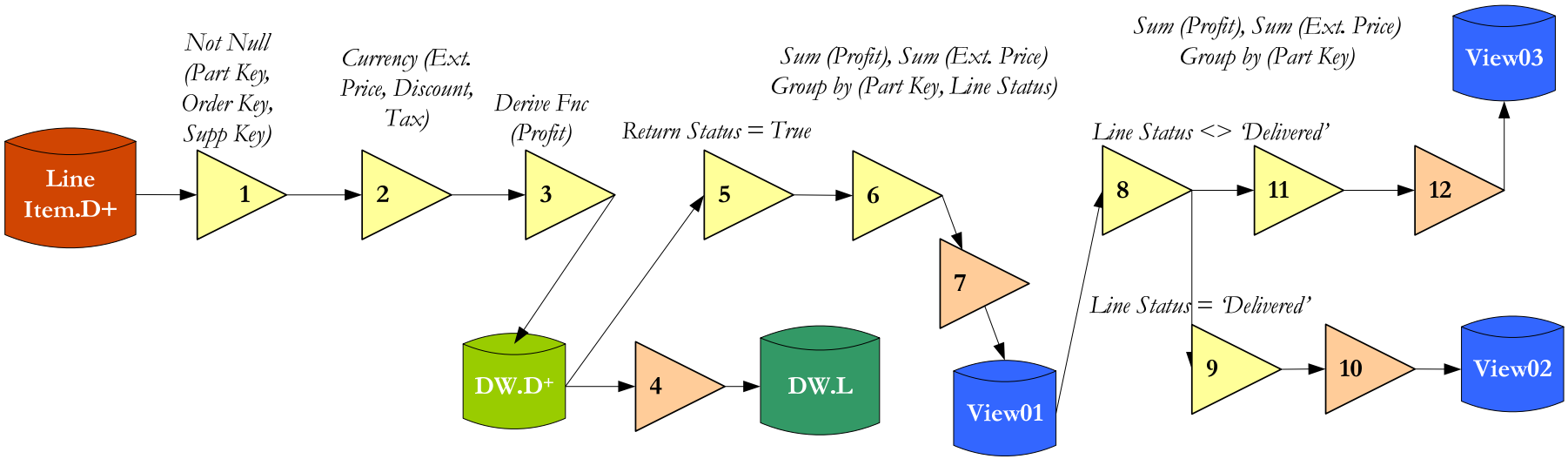
(e) Flat Hierarchy - Fork



(f) Deep Hierarchy

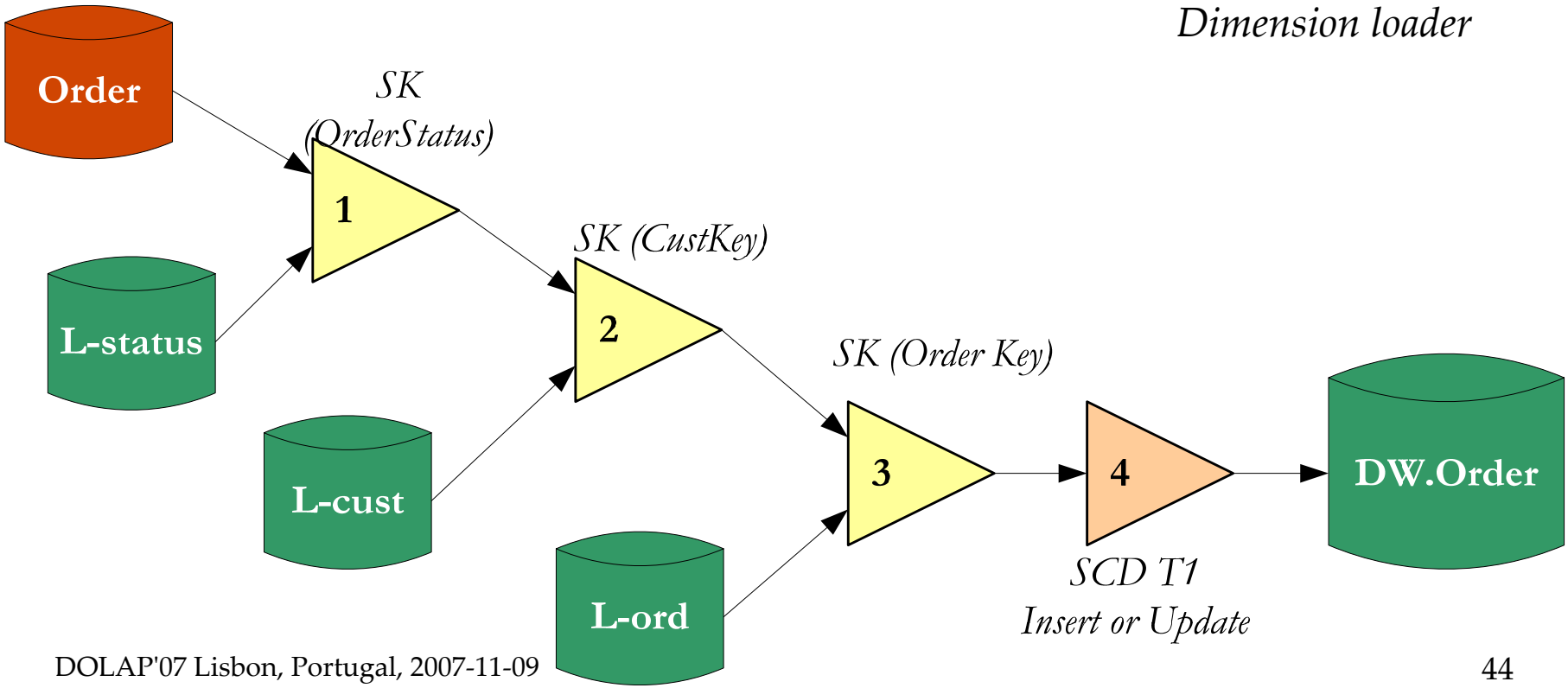
Line

- *Simplest pattern*
- *Observe the router #8*



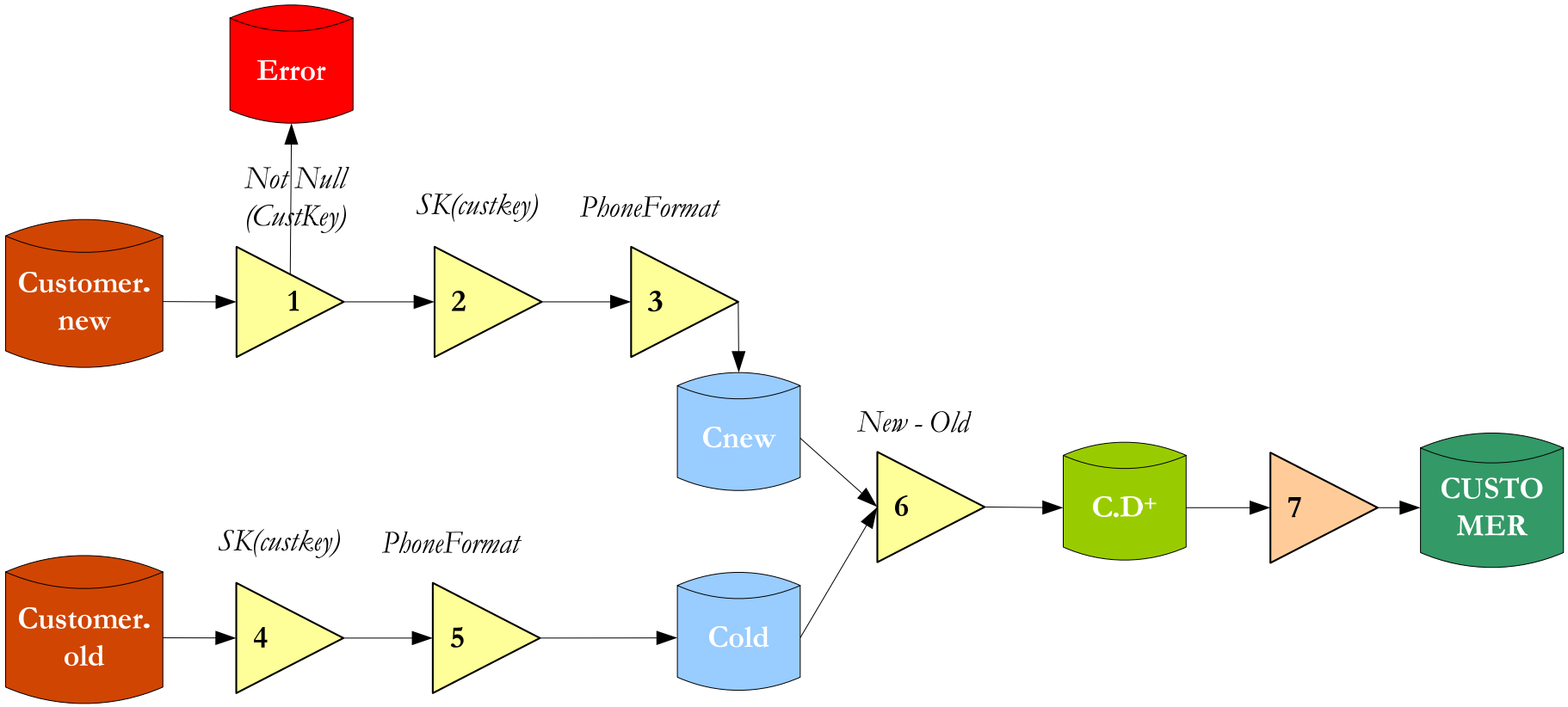
Primary Flow

- *Typical for assigning surrogate keys to factual records*
- *TPC-DS only pattern*
- *Observe the Slowly Changing Dimension loader*



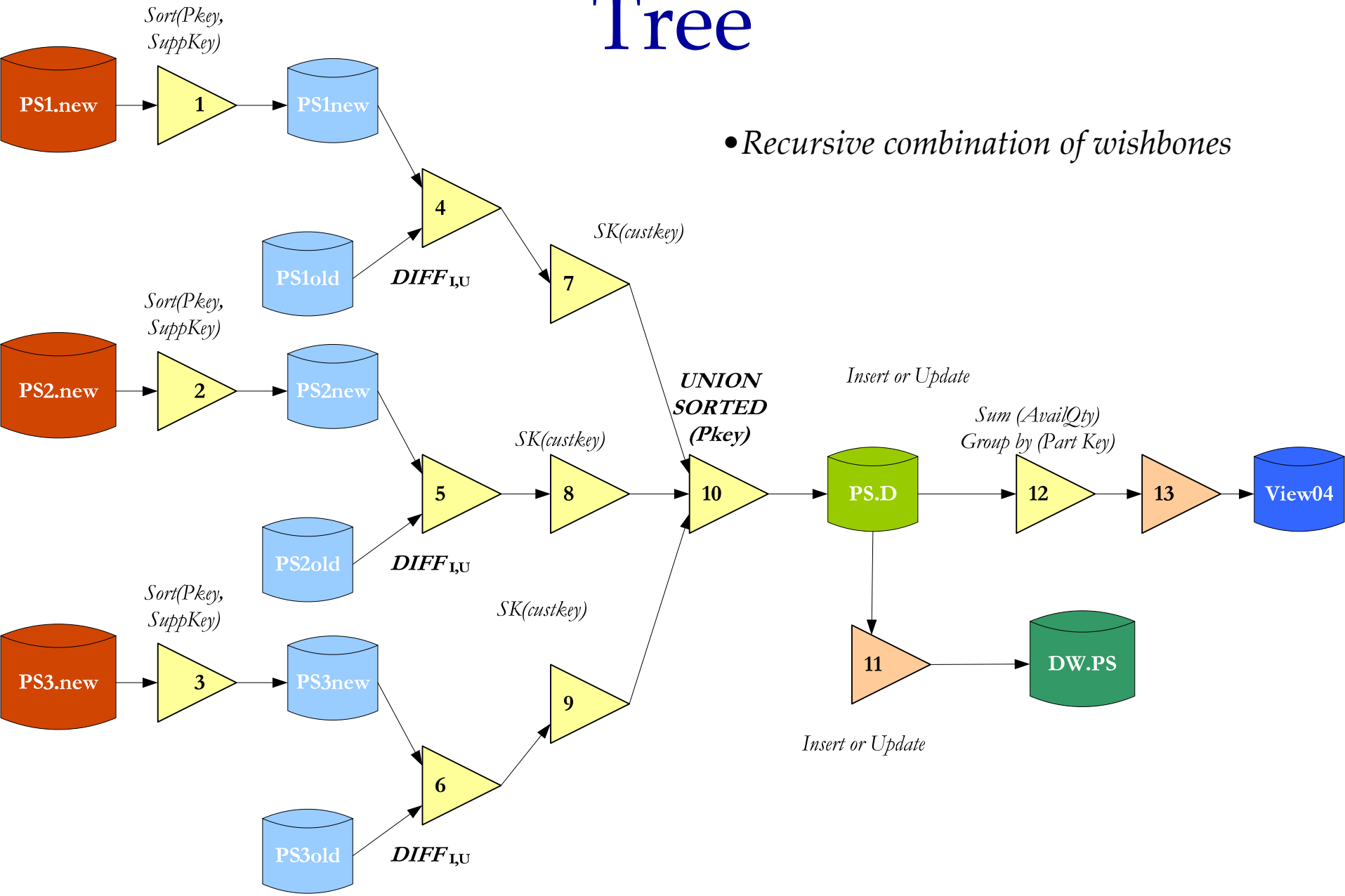
Wishbone

- *2-3 Small lines, combined via a join variant*
- *Observe the quarantine for errors (#1)*



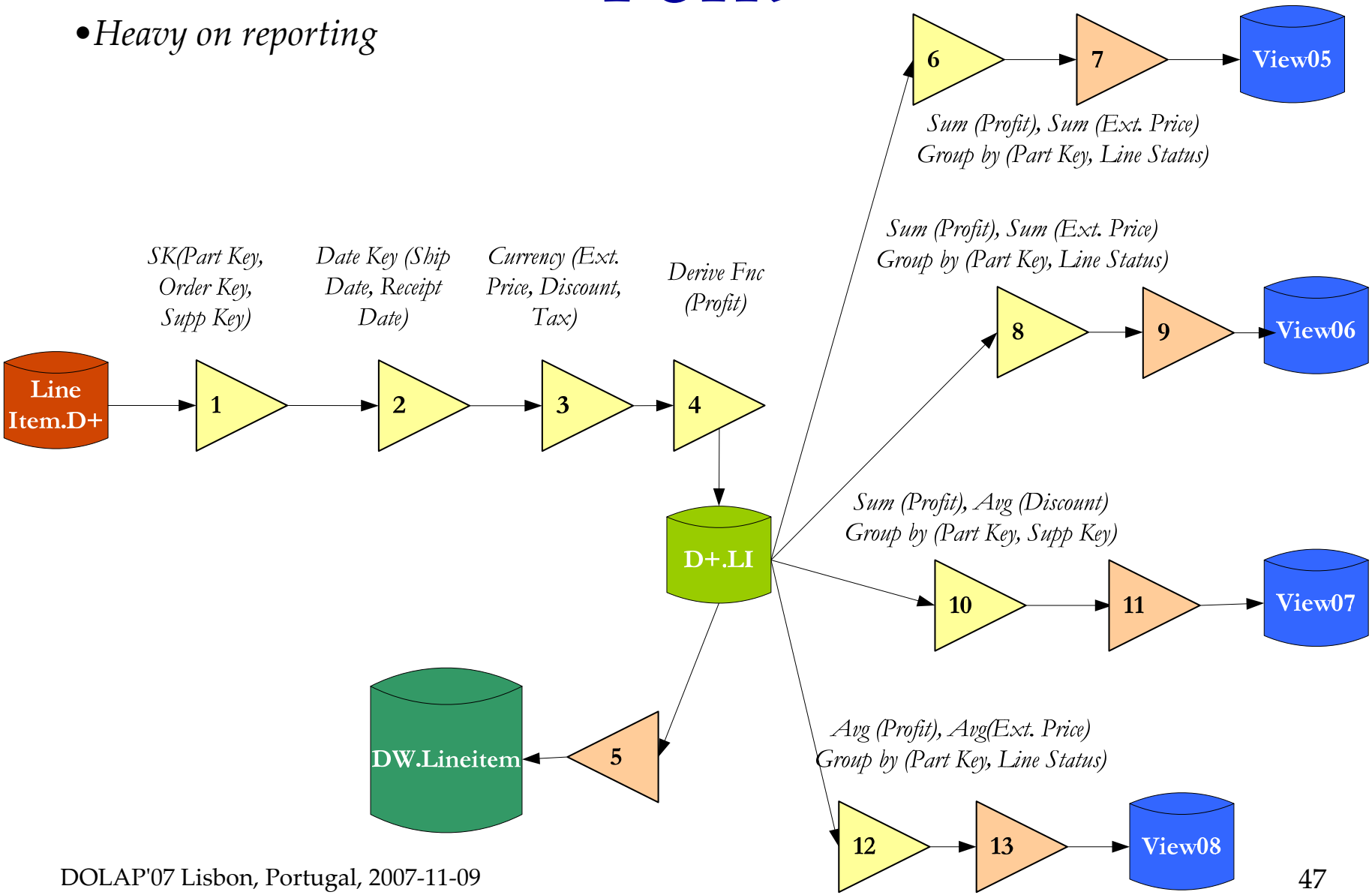
Tree

- *Recursive combination of wishbones*

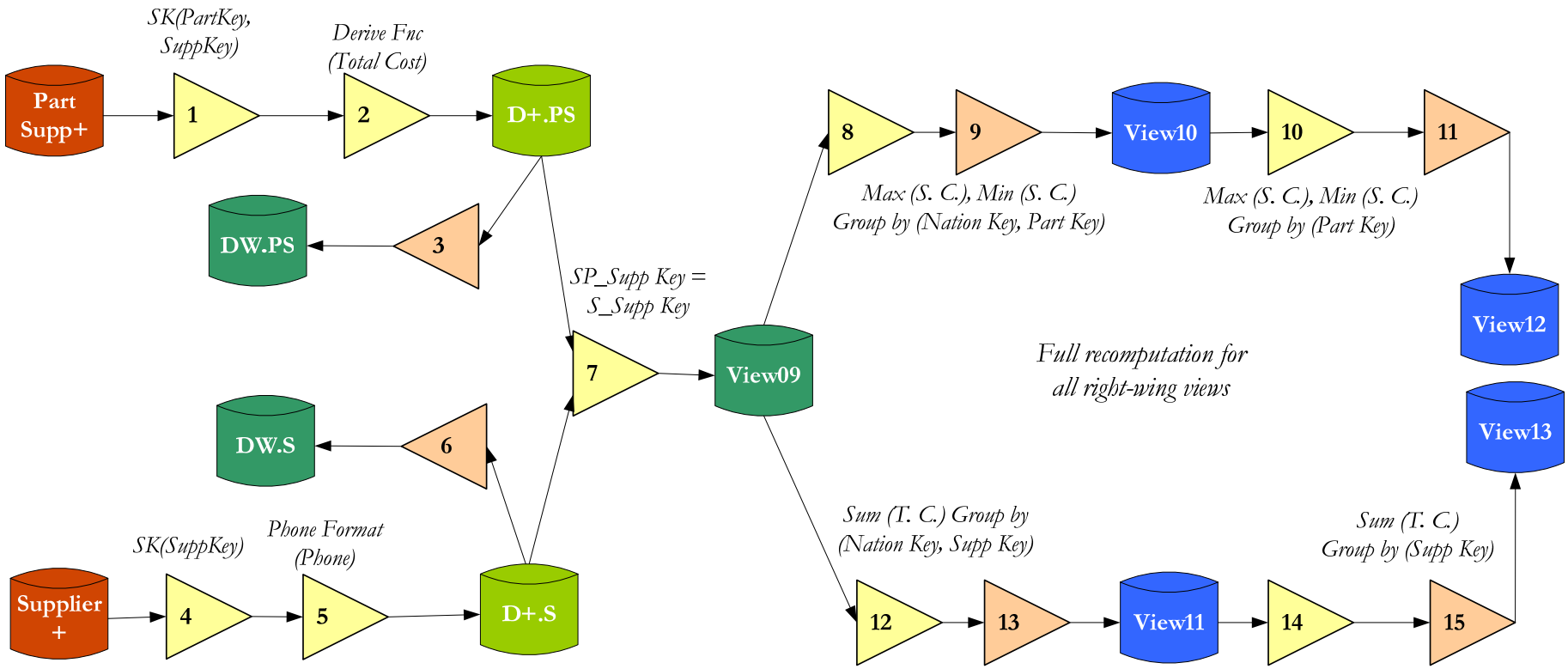


Fork

- *Heavy on reporting*



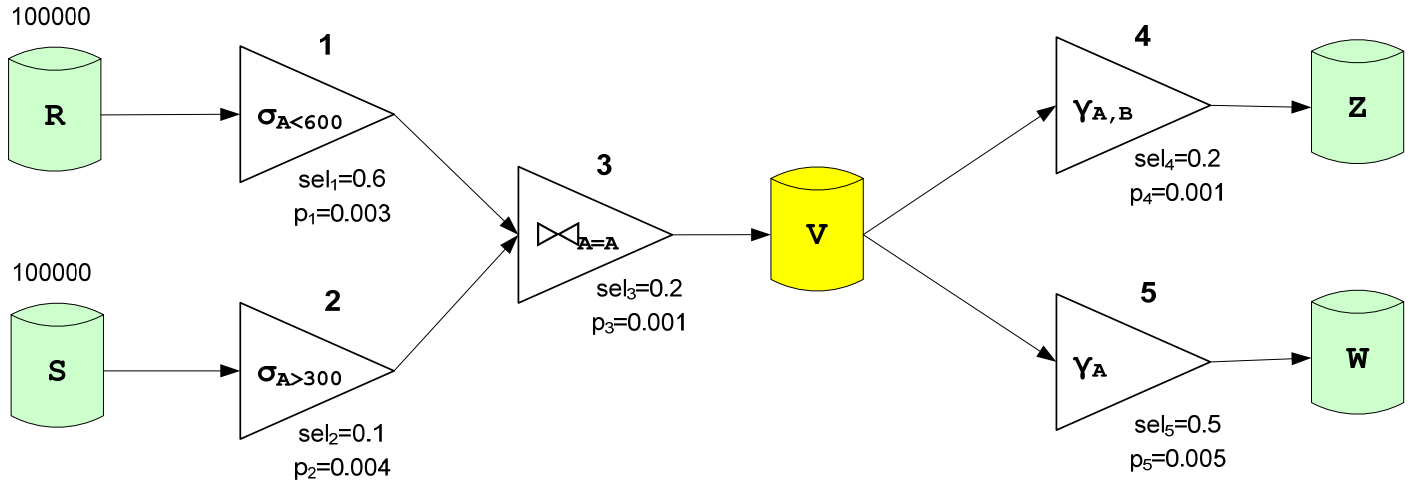
Balanced Butterfly



Experimental configuration

- Cost measures
  - Estimated execution time for scenarios
  - Number of produced scenarios
  - Computation time
  - Estimated resumption cost
  - #sorters, sorters' cost, pct of sorters' cost over total cost
- Parameters
  - amount of data arriving at the DW (by controlling the internal butterfly selectivity)
- All the experiments were conducted on an Intel(R) Pentium(R) running at 1,86 GHz with 1GB RAM and the machine has been otherwise unloaded during experiments.

Results



Number of nodes: 10
Execution time: 28 sec
Number of generated signatures: 181

S_id	Top-10 Signatures	Cost
56	((R.1.1_3(A))//(S.S!(A).2@SO.P)).3@MJ.V.((4@SO.Z)//(5@SO.W))	2.560.021
23	((R.1.1_3(A))//(S.2@SO.P)).3@MJ.V.((4@SO.Z)//(5@SO.W))	2.560.021
50	((R.1)//(S.S!(A).2@SO.P)).3@HJ.V.V!(A).((4@SO.Z)//(5@SO.W))	2.943.325
53	((R.1)//(S.S!(A).2@SO.P)).3@HJ.V.V!(B).((4@SO.Z)//(5@SO.W))	2.943.325
11	((R.1)//(S.S!(A).2@SO.P)).3@HJ.V.((4@SO.Z)//(5@SO.W))	2.943.325
17	((R.1)//(S.2@SO.P)).3@HJ.V.V!(A).((4@SO.Z)//(5@SO.W))	2.943.325
20	((R.1)//(S.2@SO.P)).3@HJ.V.V!(B).((4@SO.Z)//(5@SO.W))	2.943.325
4	((R.1)//(S.2@SO.P)).3@HJ.V.((4@SO.Z)//(5@SO.W))	2.943.325
10	((R.1)//(S.S!(A).2@SO.P)).3@SMJ.V.((4@SO.Z)//(5@SO.W))	2.996.202
3	((R.1)//(S.2@SO.P)).3@SMJ.V.((4@SO.Z)//(5@SO.W))	2.996.202

Results

S_id	Number of Sorters	Cost of Sorters	Percentage of Sorter Cost
56	2	1.803.841	70%
23	1	142.877	6%
50	2	2.107.144	72%
53	2	2.107.144	72%
11	1	1.660.964	56%
17	1	446.180	15%
20	1	446.180	15%
4	0	0	0%
10	1	1.660.964	55%
3	0	0	0%

Observations

- **Butterflies with no right wing**
  - **Not particularly improved when sorters are involved** (esp., Wishbones and Trees)
  - Certain cases, in trees, where sorters might help – if the data pushed through the involved branch has a small size or a large number of activities share the same interesting order.
- Sorters at the sources: too costly!
- Sorters are beneficial when a significant right wing is present, e.g., in Forks or Deep-hierarchy butterflies

Observations

- **Balanced butterflies**
  - Many candidate positions for sorters.
  - **The body of the butterfly is a good candidate to place a sorter**, especially when the left wing is highly selective.
  - Overall, the introduction of sorters appears to benefit the overall cost.
- **Butterflies with a right-deep hierarchy**
  - Similar to BB
  - The size of the right wing is the major determinant of the overall completion cost (large no. of candidate positions for sorters).
- **Forks**
  - **Sorters are highly beneficial for forks**.
  - The body of the butterfly is typically a good candidate for a sorter.

Observations

- The best scenario is typically found early – within the first 50-60 signatures

Roadmap

- Background & problem formulation
- General solutions & improvements
- Experiments & results
- Conclusions & future work

Conclusions

- We have dealt with the problem of determining the best possible physical implementation of an ETL workflow, given its logical-level description and an appropriate cost model as inputs.
- We have experimented with artificially introducing sorters in the physical representation of the workflow
- Not covered: failures, heuristics
- Long version: Vasiliki Tziovara's MSc
  http://charon.cs.uoi.gr/~tech_report/public.php (MT 2006-13)

Message to you

- *There is a vast, **unexplored area of research** on the **optimization of ETL scenarios***
  - *Many open issues: order of activities, treatment of indexes, active warehousing, …*
- *We need a **commonly agreed benchmark** that realistically reflects real-world ETL scenarios*

Butterflies to the rescue !!


Thank you!
All pictures are imported from MS Clipart and MSDN

Auxiliary slides

Goal of this work

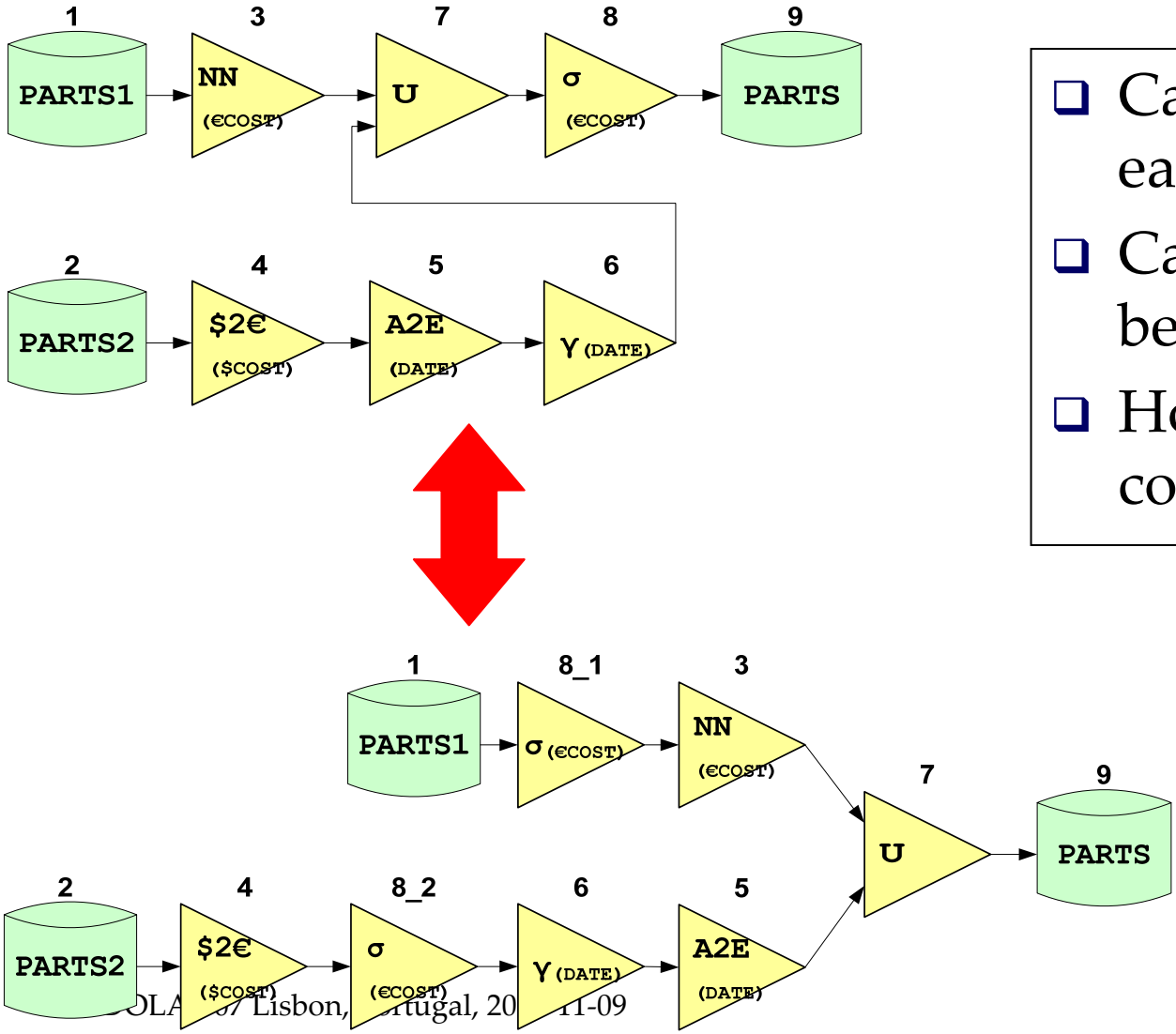
- *The objective of this work is to identify the best possible physical implementation for a given logical ETL workflow*

ETL workflows are not big queries

- It is not possible to express all ETL operations in terms of relational algebra and then optimize the resulting expression as usual. In addition, the cases of functions with unknown semantics -`black-box' operations- or with `locked' functionality -e.g., an external call to a DLL library- are quite common.
- Failures are a critical danger for an ETL workflow. The staging of intermediate results is often imposed by the need to resume a failed workflow as quickly as possible.
- ETL workflows may involve processes running in separate environments, usually not simultaneously and under time constraints; thus their cost estimation in typical relational optimization terms is probably too simplistic.

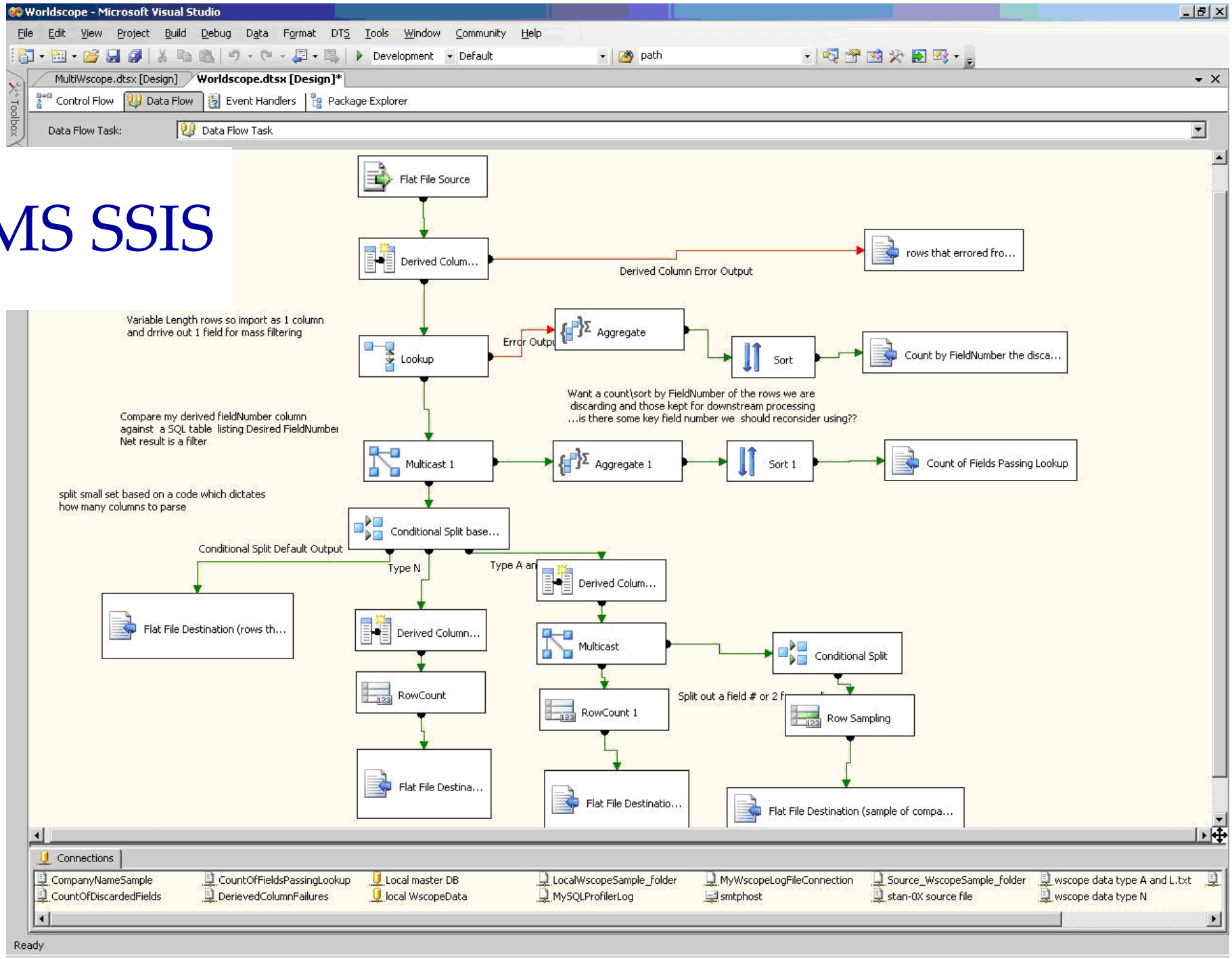
- All the aforementioned reasons can be summarized by mentioning that neither the semantics of the workflow can always be specified, nor its structure can be determined solely on these semantics; at the same time, the research community has not come-up with an accurate cost model so far.

Logical Optimization



- Can we push selection early enough?
- Can we aggregate before \$2€ takes place?
- How about naming conflicts?

MS SSIS


Worldscope - Microsoft Visual Studio
File Edit View Project Build Debug Data Format DTS Tools Window Community Help
Development Default path
MultiWscope.dtsx [Design] Worldscope.dtsx [Design]*
Control Flow Data Flow Event Handlers Package Explorer
Data Flow Task: Data Flow Task
Flat File Source
Derived Colum...
Derived Column Error Output
rows that errored fro...
Variable Length rows so import as 1 column
and drrive out 1 field for mass filtering
Lookup
Error Outpu
Aggregate
Sort
Count by FieldNumber the disca...
Want a count\sort by FieldNumber of the rows we are
discarding and those kept for downstream processing
...is there some key field number we should reconsider using??
Compare my derived fieldNumber column
against a SQL table listing Desired FieldNumbe
Net result is a filter
Multicast 1
Aggregate 1
Sort 1
Count of Fields Passing Lookup
split small set based on a code which dictates
how many columns to parse
Conditional Split base...
Conditional Split Default Output
Type N
Type A ar
Derived Colum...
Flat File Destination (rows th...
Derived Column...
Multicast
Conditional Split
RowCount
RowCount 1
Split out a field # or 2 f
Row Sampling
Flat File Destina...
Flat File Destinatio...
Flat File Destination (sample of compa...
Connections
CompanyNameSample
CountOfDiscardedFields
CountOfFieldsPassingLookup
DerievedColumnFailures
Local master DB
local WscopeData
LocalWscopeSample_folder
MySQLProfilerLog
MyWscopeLogFileConnection
smtphost
Source_WscopeSample_folder
stan-0X source file
wscope data type A and L.txt
wscope data type N
Ready

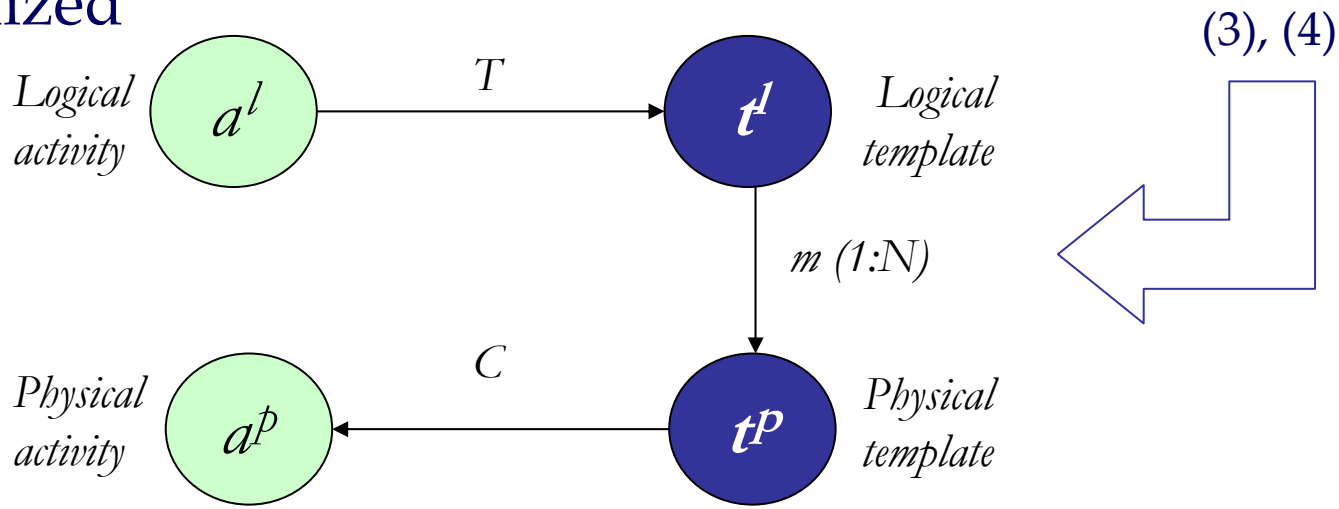
Problem Formulation

Constraints

- The data consumer of a recordset cannot be another recordset. Still, more than one consumer is allowed for recordsets.
- Each activity must have at least one provider, either another activity or a recordset. When an activity has more than one data providers, these providers can be other activities or activities combined with recordsets.
- Each activity must have exactly one consumer, either another activity or a recordset.
- Feedback of data is not allowed; i.e., the data consumer of an activity cannot be the same activity.

Template customization

1. The designer selects a logical level template
2. The designer customizes the template with the appropriate schemata and parameters
3. The optimizer (or the designer) chooses one of the available implementations of the logical template (i.e., a physical template)
4. The physical template is then appropriately customized



Problem formulation

Formal definition of the problem. Given a logical-level ETL workflow $\mathbf{G^L(V^L, E^L)}$, where $\mathbf{V^L = A^L \cup R}$, $\mathbf{A^L}$ is the set of logical activities and $\mathbf{R}$ is the set of recordsets, determine the physical-level graph $\mathbf{G^P(V^P, E^P)}$, where $\mathbf{V^P = A^P \cup R}$, $\mathbf{A^P}$ is the set of physical activities, such that:

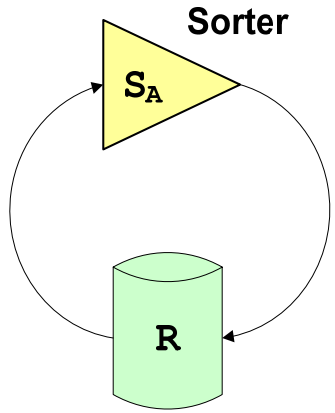
- $a_i^P = C(\mathcal{M}(T(a_i^L))), a_i^P \in \mathbf{A^P}, a_i^L \in \mathbf{A^L}$
- $\forall e^L = (x^L, y^L), e^L \in \mathbf{E^L}, x^L, y^L \in \mathbf{V^L}$, introduce e^P to $\mathbf{E^P}$ where $e^P = (x^P, y^P), x^P, y^P \in \mathbf{V^P}$
- $\wedge_i constr(a_i^P) = true, a_i^P \in \mathbf{A^P}$
- $\sum_i cost(a_i^P) = minimal, a_i^P \in \mathbf{A^P}$

Interesting Orders

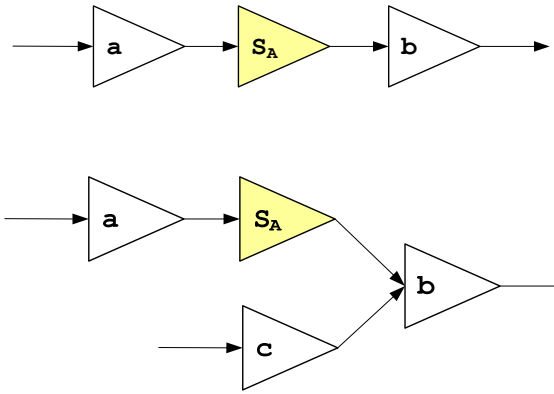
- Traditional view of interesting orders
  - Sorting specification useful for determining best left-deep query plan in traditional query processing (Selinger et al.)
- ETL workflows
  - A list of attributes over which the input of an activity can be sorted
  - Similarly, a list of attributes for sorting recordsets

Transitions in the state-space problem due to sorters

- ASR(v, R): Add Sorter on RecordSet
  - G(V,E) →G′(V′,E′), s.t.
    - V′ = V ∪ {v}
    - e′=(R, v), e′′=(v, R), E′ = E ∪e′∪e′′



- ASE(v, a, b): Add Sorter on Edge
  - G(V,E) →G′(V′,E′), s.t.
    - V′ = V ∪ {v}
    - Remove (a, b)
    - ∀ e ∈ E με e=(a, b) insert e′=(a,v) and e′′=(v, b). I.e., E′ = E ∪e′∪e′′- e.



Algorithmic Issues

Exhaustive Ordering (EO)

Algorithm: Exhaustive Ordering (**EO**)

Input: An logical graph $\mathbf{G^L} = (\mathbf{V^L}, \mathbf{E^L})$ with n nodes
Output: A signature S_{MIN} having minimal cost

```
1  Begin
2      S_0 ← Compute_Signature
3      Cost(S_0) ← Compute_Cost(S_0), S_MIN = S_0
4      Let D be a dictionary that contains signatures and
       respective costs, add S_0 and Cost(S_0) to D
5      Let Γ = {γ_1, γ_2, ..., γ_m} be the set of all possible
       combinations of candidate positions for sorters
6      Given a γ∈Γ, let P_γ ={p_γ1 ,p_γ2 ,..., p_γk } be the set of
       possible positions of combination γ
7      Given a position p_γ, let O_cs = {o_1, o_2, ..., o_n} be the set
       of candidate sorters over p_γ (including no sorter)
8      Foreach γ ∈ Γ {
9          Foreach p_γ ∈ P_γ {
10             Foreach o ∈ O_cs {
11                 generate a new signature S'
12                 If (S' ∉ D) {
13                     Cost(S') ← Compute_Cost(S')
14                     Store S' and Cost(S') to D
15                     If (Cost(S') < Cost(S_MIN)) {
16                         S_MIN = S'
17                 }}
18     }}}
19     Return S_MIN
20 End
```

Signature Generation

- Algo GSign by [SiVS04]
- Extension of GSign -> EGSign with:
  - many targets
  - Usage of DSA tables
  - Incorporation of the physical-level layers
    - e.g., *R.1.V.((2@NL.W)//(3.Z))*
  - Incorporation of sorters
    - e.g., *R.1. 1_2(A,B).2.V.((3.W)//(4.Z))*
    - e.g., *R.1.V.V!(A,B).((2.W)//(3.Z))*

GSign vs EGSign

Algorithm Get Signature (GSign)

1. **Input:** A state S, i.e., a graph **G=(V,E)**, a state S′ with edges in the reverse direction than the ones of S, and a node v that is a target node for the state S
2. **Output:** The signature sign of the state S

```
3. Begin
4. Id ← find the Id of v;
5. sign = "." + Id + sign;
6. if (outdeg(v)==2) {
7.    v1 ← next of v with the lowest Id;
8.    GSign(S,v1,s1);
9.    v2 ← next of v with the highest Id;
10.   GSign(S,v2,s2);
11.   sign="(("+s1+")//("+s2+"))"+ sign;
12. }
13. else if (outdeg(v)==1) {
14.    v ← next of v;
15.    GSign(S,v,sign);
16. }
17. sign = sign.replace_all("(.","(");
18. End.
```

Algorithm GSign ([SiVS04])

Algorithm Extended GSign (EGS)

1. **Input:** A state S, i.e., a graph **G=(V,E)**, a state S′ with edges in the reverse direction than the ones of S, and a set T with the target nodes for the state S
2. **Output:** The signature sign of the state S

```
3.  Begin
4.  n = 0;
5.  for each v in T {
6.    GSign(S,v,C[n]);
7.    n++;
8.  }
9.  if (n==1) {
10.   sign = C[0];
11. }
12. else {
13.  for i = 1 to n {
14.    V = FindRecordset(C[i],C[0]);
15.    str0=first part of C[0] until V;
16.    str1=rest of C[0], after V;
17.    str2=rest of C[i], after V;
18.    C[0]=str0+"(("+str1+")//("+str2+"))";
19.    C[0]= C[0].replace_all("(.","(");
20.  }
21. }
22. sign = C[0];
23. End.
```

Algorithm Extended GSign

Employed Algorithms

- Generate Possible Orders (GPO)
  - Takes as input a set of attributes and produces all the possible combinations of them
    - e.g. interesting orders: {A,B}→{A},{B},{A,B},{B,A}
- Compute Place Combinations (CPC)
  - Input: a set of possible places
  - Output: all their possible combinations
    - e.g. positions {R, (1-2)}→{R},{(1-2)},{R,(1-2)}

Generate Possible Orders (GPO)

Algorithm Generate Possible Orders (GPO)

1. **Input**: A set `LookupSet` with n items
2. **Output:** A set `ResultSet` that contains all possible orders.
3. **Begin**
4. Let LookupSet = {I_1, I_2, … , I_n} be a set with n items
5. TempSet = {};
6. Add to ResultSet all items of LookupSet as different sets, i.e., ResultSet = {{ I_1}, {I_2}, … , {I_n}};
7. do
8. {
9. TempSet = {};
10. **for each** set s in ResultSet {
11. **for each** item i in LookupSet {
12. **if** (**i** ∉ **s**)
13. {
14. TempSet = TempSet ∪ {s ∪ {i}};
15. }
16. }
17. ResultSet = ResultSet ∪ TempSet;
18. }
19. while (TempSet ≠ ∅);
20. **return** ResultSet;
21. **End.**

Compute Place Combinations (CPC)

Algorithm Compute Place Combinations (CPC)

1. **Input**: A set `LookupSet` with n items
2. **Output:** A set `ResultSet` that contains all possible orders.
3. **Begin**
4. Let LookupSet = {I_1, I_2, … , I_n} be a set with n items
5. TempSet = {};
6. Add to ResultSet all items of LookupSet as different sets, i.e., ResultSet = {{ I_1}, {I_2}, … , {I_n}};
7. do
8. {
9. TempSet = {};
10. **for each** set s in ResultSet {
11. **for each** item i in LookupSet {
12. **if** (**i** ∉ **s**)
13. {
14. z = s ∪ {i};
15. if Not(ExistsIn(TempSet,z)){
16. TempSet = TempSet ∪ {z};
17. }
18. }
19. }
20. ResultSet = ResultSet ∪ TempSet;
21. }
22. while (TempSet ≠ ∅);
23. **return** ResultSet;
24. **End.**

Employed Algorithms

- Generate Possible Signatures (GPS)
  - Input: a signature
  - Output: all possible signatures with sorters
  - uses CPC και GPO
  - AppendOrder(S,o,p):
    - Append order o in place p of signature S
    - If p=(a,b) replace in S the string a.b with a.a_b(o).b
    - If p=V then replace in S the string V with the string V.V!(o)

Generate Possible Signatures (GPS)

Algorithm Generate Possible Signatures (GPS)

1. **Input**: A signature `S` of a graph **`G=(V,E)`** with n nodes
2. **Output:** **`C`** is a collection with all signatures that contain possible sorters.

```
3.  Begin
4.  for each place p in G {
5.    ResultSet = CPC(p);   //combinations of places
6.  }
7.  for each combination c in ResultSet {  //for each combination of places
8.      for each place p in combination c {  //for each place
9.          CandidateSet = GPO(p);   //candidate orderings for place p
10.         for each order o in CandidateSet{
11.            tempSignature = AppendOrder(S,o,p); //append order o in place p of S
12.            C = C ∪ {tempSignature};
13.         }
14.     }
15. }
16. return C;
17. End.
```

Observations

- **Balanced butterflies**. The general case of butterflies is characterized by many candidate positions for sorters. Overall, the introduction of sorters appears to benefit the overall cost. The body of the butterfly is a good candidate to place a sorter, especially when the left wing is highly selective.
- **Butterflies with a right-deep hierarchy**. These butterflies behave similarly to the general case of balanced butterflies. The size of the right wing is the major determinant of the overall completion cost of our algorithms due to the large number of candidate positions for sorters.
- **Lines**. The generated space of alternative physical representations of a linear scenario is linear to the size of the workflow (without addition of sorters). In our experiments we have observed that due to the selectivities involved, the left wing might eventually determine the overall cost (and therefore, placing filters as early as possible is beneficial, as one would typically expect).
- **Butterflies with no right wing**. In principle, the butterflies that comprise just a left wing are not particularly improved when sorters are involved. In particular, the introduction of sorters in Wishbones and Trees does not lead to the reduction of the total cost of the workflow. However, there are certain cases, in trees, where sorters might help - provided that the data pushed through the involved branch has a small size or a large number of activities share the same interesting order.
- **Forks**. Sorters are highly beneficial for forks. This is clearly anticipated since a fork involves a high reusability of the butterfly's body. Therefore, the body of the butterfly is typically a good candidate for a sorter.

Related Work

Related Work

[Arkt05] ARKTOS II http://www.cs.uoi.gr/~pvassil/projects/arktos_II/index.html

[ChSh99] S. Chaudhuri, K. Shim. Optimization of Queries with User-Defined Predicates. *In the ACM Transactions on Database Systems, Volume 24(2)*, pp. 177-228, 1999.

[CuWi03] Y. Cui, J. Widom. Lineage tracing for general data warehouse transformations. *In the VLDB Journal Volume 12 (1)*, pp. 41-58, May 2003.

[Hell98] J. M. Hellerstein. Optimization Techniques for Queries with Expensive Methods. *In the ACM Transactions on Database Systems, Volume 23(2)*, pp. 113-157, June 1998.

[Inmo02] W. Inmon, Building the Data Warehouse, John Wiley & Sons, Inc. 2002.

[LWGG00] W. Labio, J.L. Wiener, H. Garcia-Molina, V. Gorelik. Efficient Resumption of Interrupted Warehouse Loads. *In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000)*, pp. 46-57, Dallas, Texas, USA, 2000.

Related Work

[MoSi79] C. L. Monma and J. Sidney. Sequencing with series-parallel precedence constraints. *In Math. Oper. Res. 4*, pp. 215-224, 1979.

[NeMo04] T. Neumann, G. Moerkotte. An Efficient Framework for Order Optimization. *In Proceedings of the 30th VLDB Conference (VLDB 2004)*, pp. 461-472, Toronto, Canada, 2004.

[PPDT06] Programmar Parser Development Toolkit version 1.20a. NorKen Technologies. Available at http:// www.programmar.com, 2006.

[SAC+79] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In Philip A. Bernstein, editor, *In Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, Boston, Massachusetts, pp. 23-34, May 30 - June 1, 1979.

[SiSM96] D. Simmen, E. Shekita, T. Malkenus. Fundamental Techniques for Order Optimization. *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, Montreal, Quebec, Canada, June 1996.

[SiVS04] A. Simitsis, P. Vassiliadis, T. K. Sellis: Optimizing ETL Processes in Data Warehouse Environments, 2004.

Related Work

[SiVS05] A. Simitsis, P. Vassiliadis, T. K. Sellis. Optimizing ETL Processes in Data Warehouses. *In Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, pp. 564-575, Tokyo, Japan, April 2005.

[Ullm88] J. D. Ullman, Principles of Database and Knowledge-base Systems, Volume I, Computer Science Press, 1988.

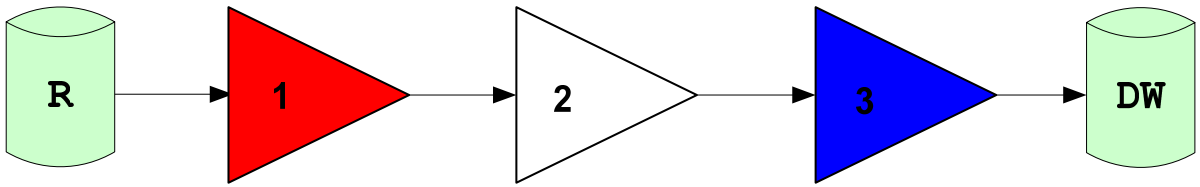
[VaSS02] P. Vassiliadis, A. Simitsis, S. Skiadopoulos. Modeling ETL Activities as Graphs. *In Proceedings of the 4th International Workshop on the Design and Management of Data Warehouses (DMDW'2002) in conjunction with CAiSE'02*, pp. 52-61, Toronto, Canada, May 27, 2002.

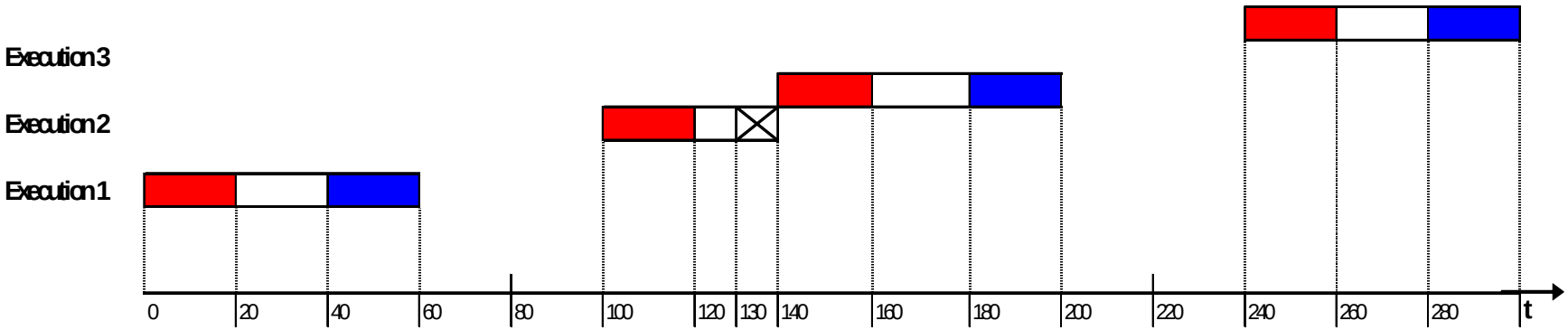
[VSGT03] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis. A Framework for the Design of ETL Scenarios. *In the 15th Conference on Advanced Information Systems Engineering (CAiSE '03)*, Klagenfurt/Austria, 16 - 20 June 2003.

[WaCh03] X. Wang, M. Cherniack. Avoiding sorting and grouping in processing queries. *In Proceedings of 29th VLDB Conference (VLDB 2003)*, Berlin, Germany, September 9-12, 2003.

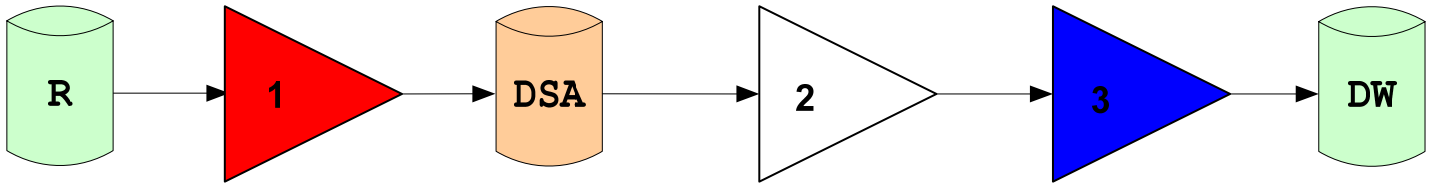
Resumption

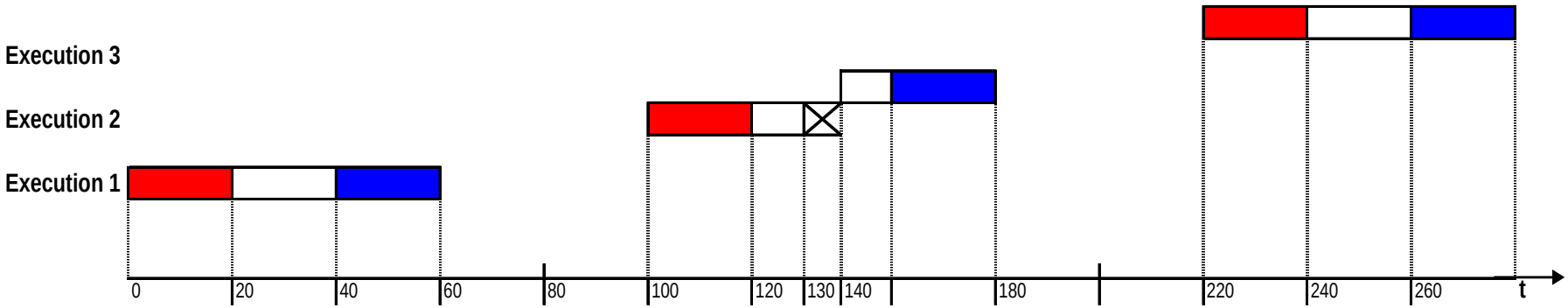
Refreshment Failures





Refreshment Failures

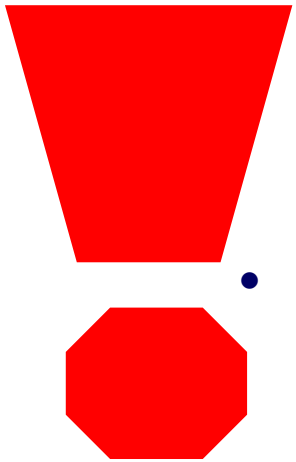




Resumption

- Each DSA table is considered a savepoint
  - In the absence of DSA tables, resumptions starts from scratch
  - Otherwise, after a failure, each activity part refers to the closest savepoint to resume work
- In the latter case, ordering pays off, since each savepoint can (a) give rescued data to subsequent activities and (b) detect which subset of the sorted incoming data have to be requested from its providers

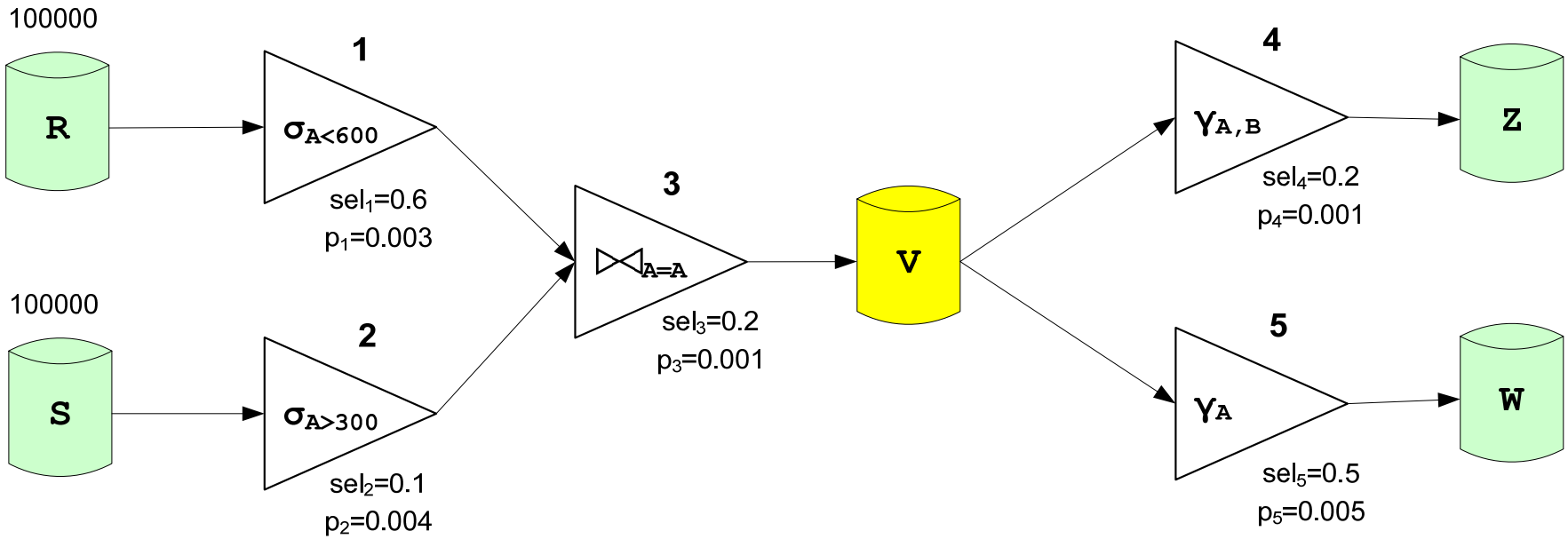
Cost models and resumption

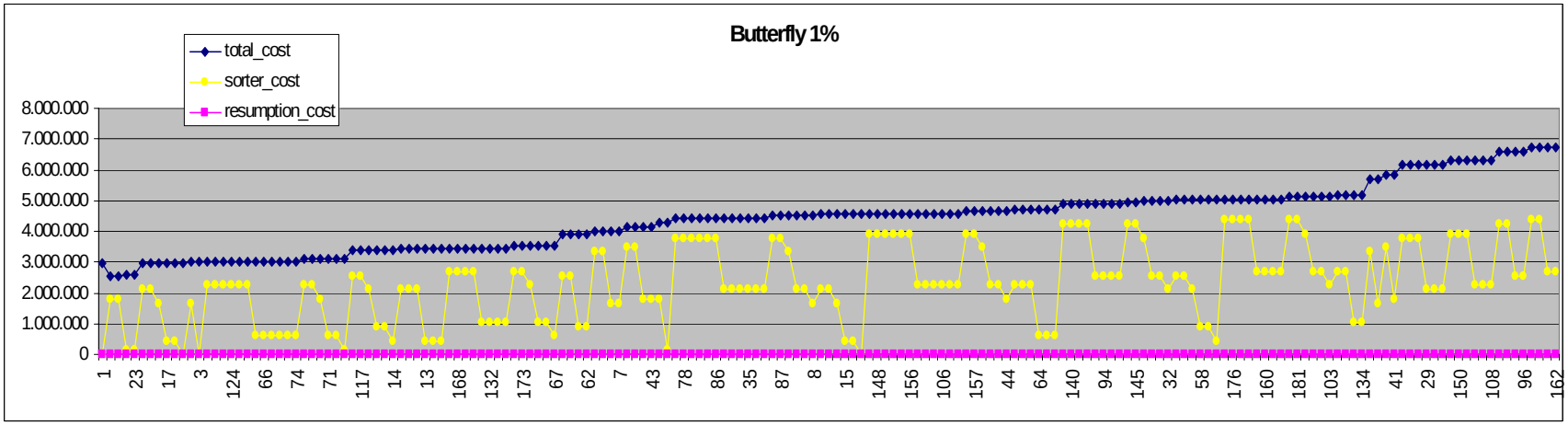


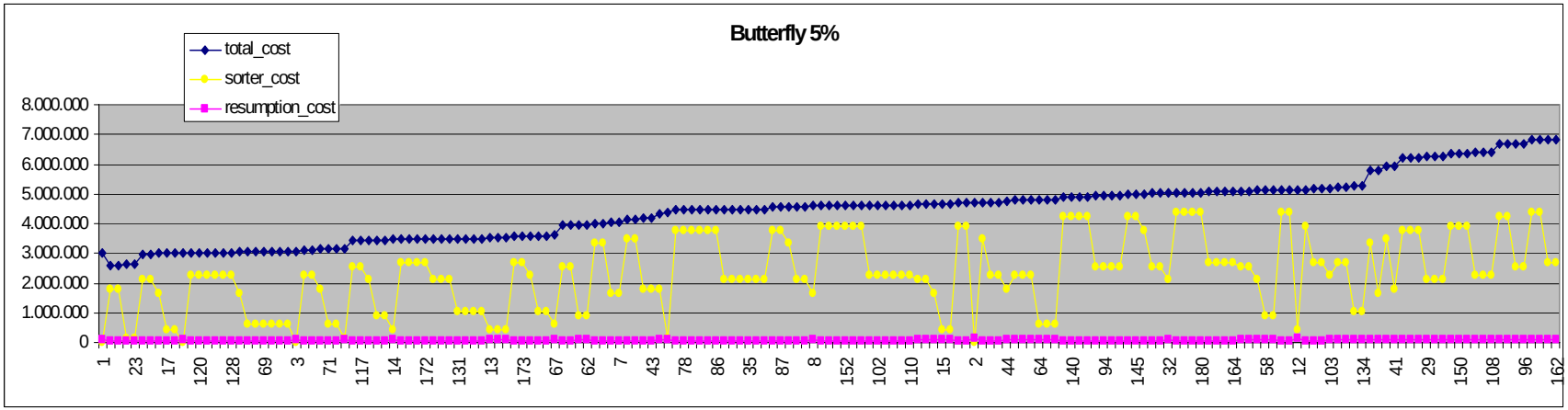
- Κανονική λειτουργία
  - Αριθμό πλειάδων που επεξεργάζεται η activity
  - Για *m* πλειάδες εισόδου: *computational_cost(i)* is a function of *m*
  - Για black-box: *computational_cost(i) =m* cost_per_tuple(i)*

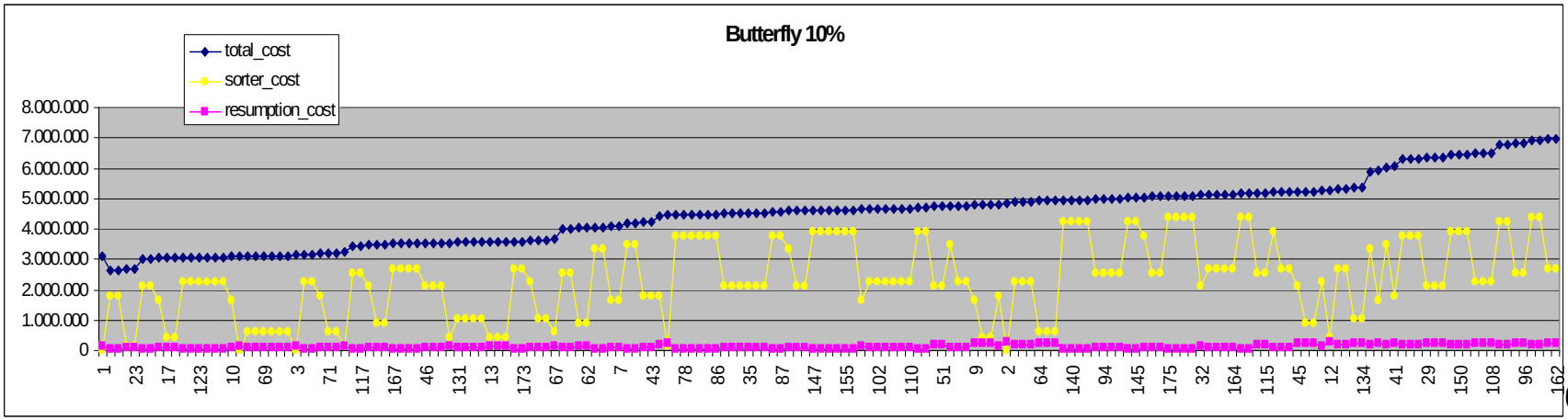
  όπου *cost_per_tuple:* κόστος επεξεργασίας μίας μόνο πλειάδας
  - συνολικά *Computational_cost(G)* = $\sum_{i=1}^{n} computational\_cost(i)$
- Κανονική λειτουργία με ανάκαμψη λόγω αποτυχιών
  - k activities στο μονοπάτι από την activity i ως το τελευταίο savepoint
  - m activities στο μονοπάτι από την activity i ως το DW
  - p_i: η πιθανότητα αποτυχίας της activity i σε μια εκτέλεση
  - *cost_until_crash(i) = 50% * computational_cost(i)*
  - *resumption_cost(i) = cost_until_crash(i) +* $\sum_{j=1}^{k} computational\_cost(j)$ *computational_cost(i) +* $\sum_{j=1}^{m} computational\_cost(j)$
  - *Resumption_cost(G) =* $\sum_{i=1}^{n} p_i * resumption\_cost(i)$

Variable failure rates p_i

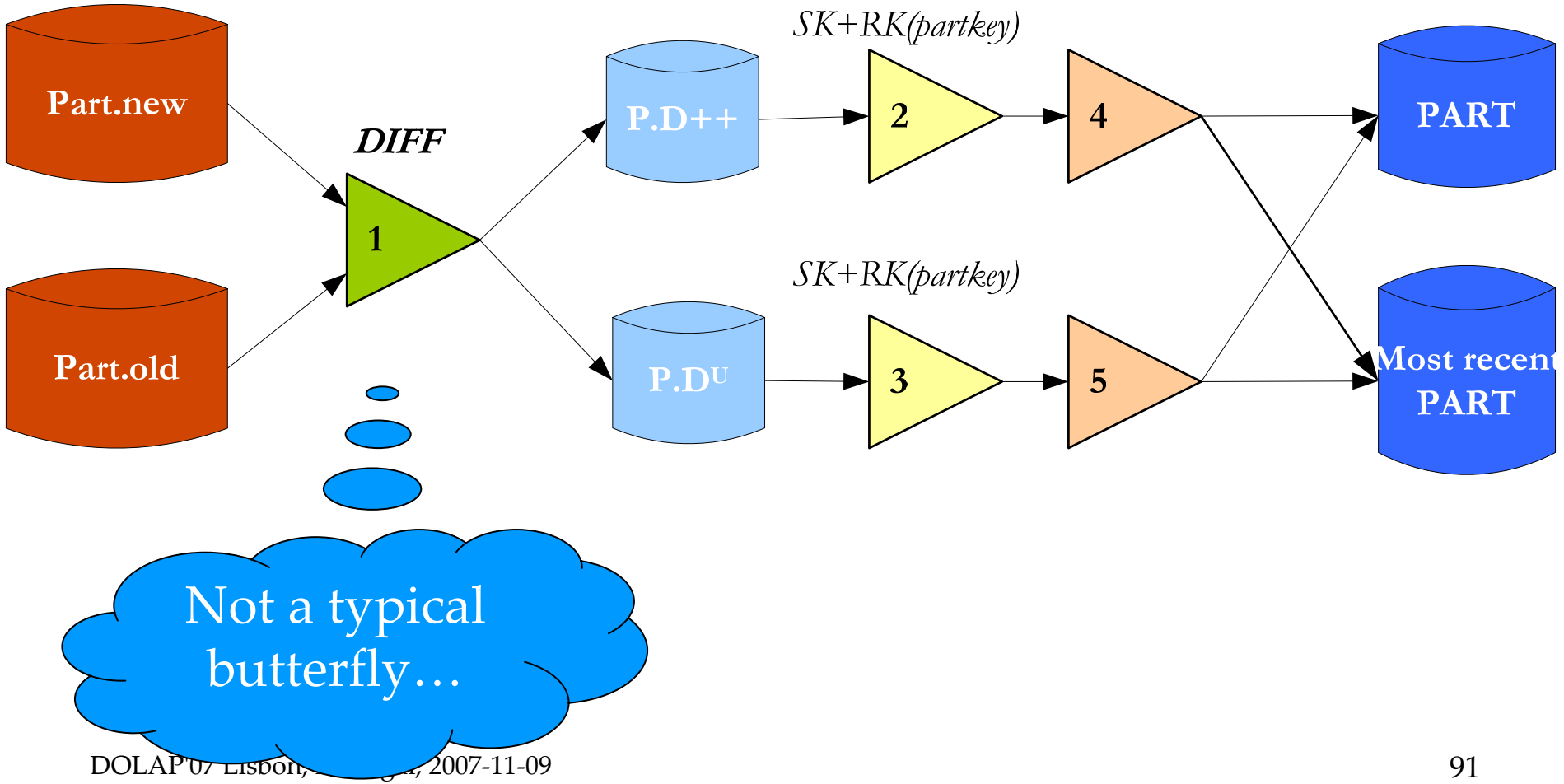



Butterfly 1%
total_cost
sorter_cost
resumption_cost
8.000.000
7.000.000
6.000.000
5.000.000
4.000.000
3.000.000
2.000.000
1.000.000
0
1 23 17 3 124 66 74 71 117 14 13 168 132 173 67 62 7 43 78 86 35 87 8 15 148 156 106 157 44 64 140 94 145 32 58 176 160 181 103 134 41 29 150 108 96 162


Butterfly 5%
total_cost
sorter_cost
resumption_cost
8.000.000
7.000.000
6.000.000
5.000.000
4.000.000
3.000.000
2.000.000
1.000.000
0
1 23 17 120 128 69 3 71 117 14 172 131 13 173 67 62 7 43 78 86 35 87 8 152 102 110 15 2 44 64 140 94 145 32 180 164 58 12 103 134 41 29 150 108 96 162


Butterfly 10%
total_cost
sorter_cost
resumption_cost
8.000.000
7.000.000
6.000.000
5.000.000
4.000.000
3.000.000
2.000.000
1.000.000
0
1 23 17 123 10 69 3 71 117 167 46 131 13 173 67 62 7 43 78 86 35 87 147 155 102 110 51 9 2 64 140 94 145 175 32 164 115 45 12 134 41 29 150 108 96 162

Balanced Butterfly
Slowly Changing Dimension of Type II



On-going/Future Work

- This work is part of the ARKTOS II project

http://www.cs.uoi.gr/~pvassil/projects/arktos_II