Informatics Department
Aristotle University of Thessaloniki

Workflow Optimization for Big Data Analytics

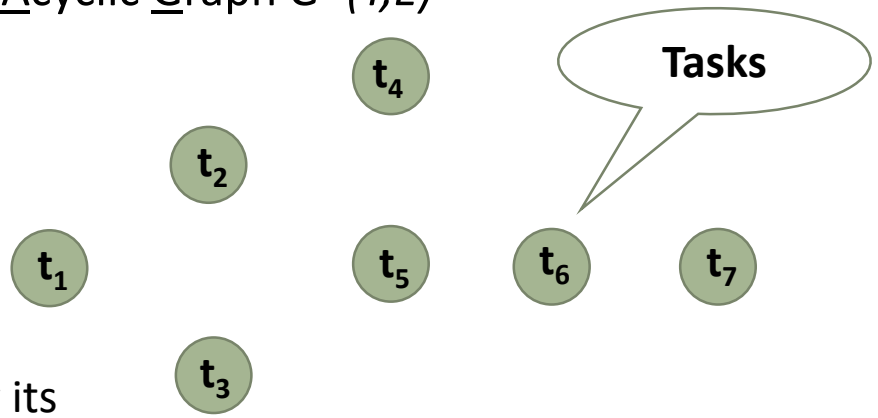
An overview of the Data Flow Optimization problem and solutions

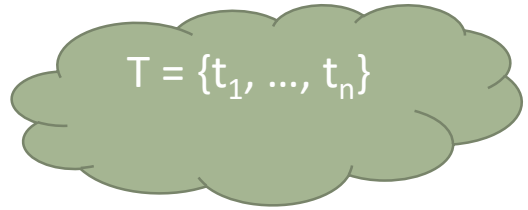
Dr. Georgia Kougka
Post-doc

September 2018

Data-Centric Flows (1/2)

- **Workflow definition:** a sequence of tasks to be executed to solve a problem or realize a process.
- **Flow Representation:** Directed Acyclic Graph *G=(T,E)*
- **Vertices** → flow **tasks**

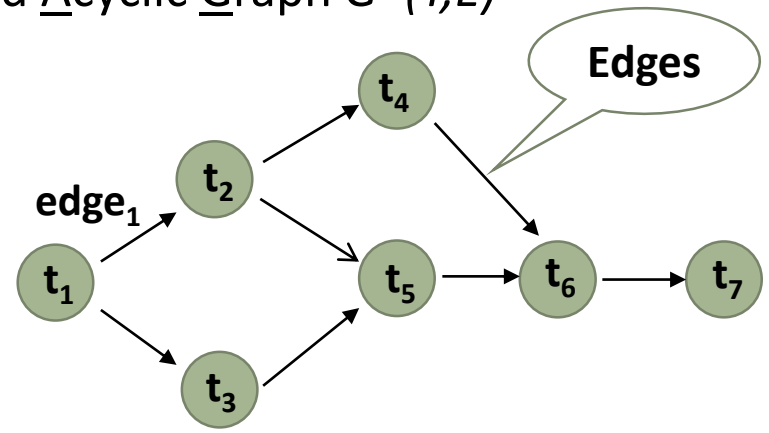


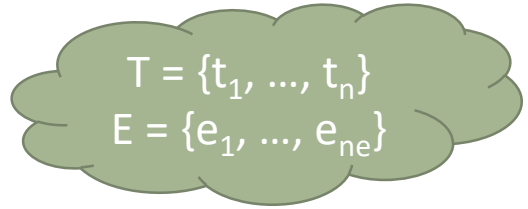


- Each workflow is characterized by its
- **control flow**, and
- **data flow** aspects/properties.
- Control flow properties: define the co-ordination of tasks, how they are triggered, etc.
- Workflow for data analysis or simply a ***data flow*** is a type of workflow, where the tasks retrieve, manipulate, process or output data.

Data-Centric Flows (2/2)

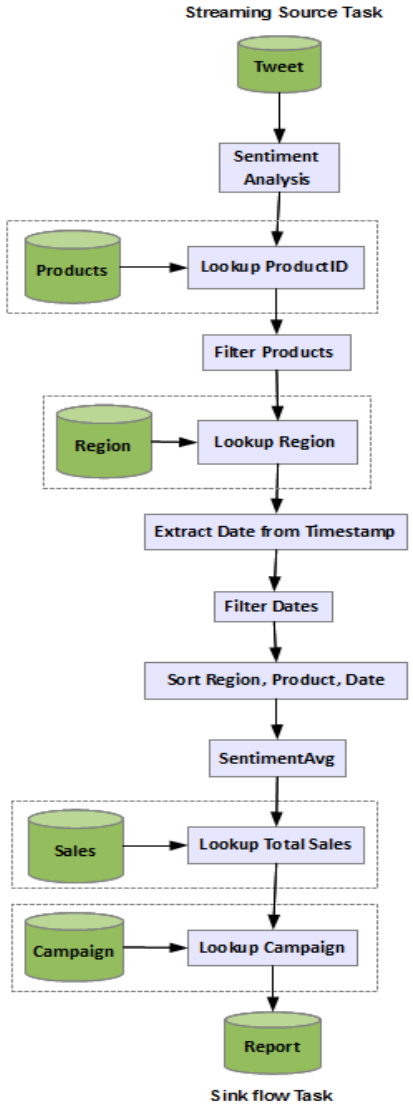
- **Workflow definition:** a sequence of tasks to be executed to solve a problem or realize a process.
- **Flow Representation:** Directed Acyclic Graph *G=(T,E)*
- **Vertices** → flow **tasks**





- Workflow for data analysis or simply a ***data flow*** is a type of data flow, where the tasks retrieve, manipulate, process or output data.
- Graph **Edges** in data flows → **transfer** data between the tasks.
- Each task may be executed when the total size of data or only a subset of the data is available.

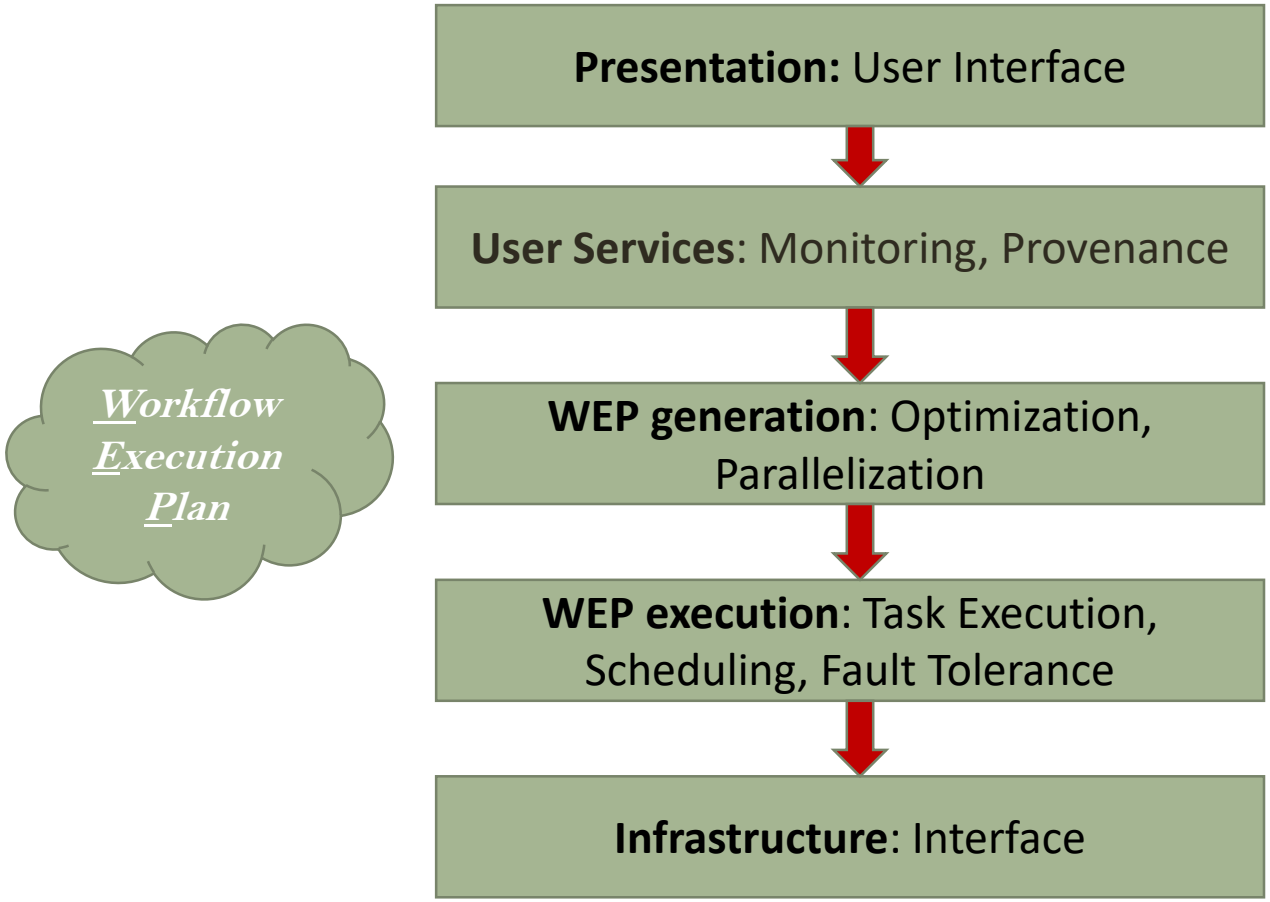
Data Flow Example



- Encountered in both business and scientific scenarios.
- Extend traditional Extract – Transform – Load (ETL) flows.
- Part of Big Data Analysis.
- May include complex analysis tasks.
- Modern data flows:
  - are complex
  - operate in highly dynamic environment
  - their execution order and other execution details are defined manually
- Consequently, manually designed flows are very prone to suboptimality

Data Flow Optimization (1/2)

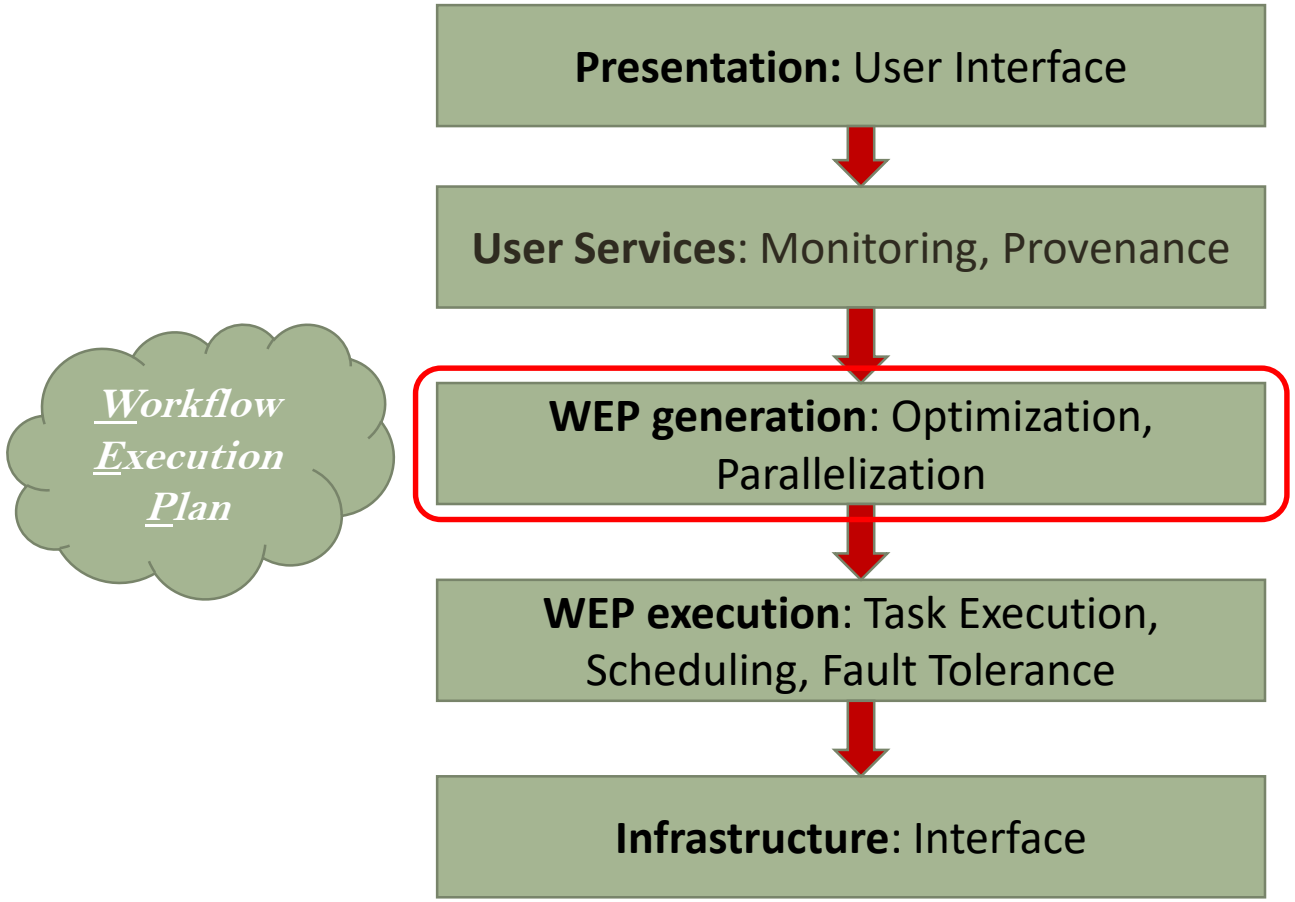
- Architecture of Workflow Management System (WfMS):



- Query Optimization **VS** Flow Optimization:
  - i. Possible arbitrary dependencies among their execution order,
  - ii. Tasks do not belong necessarily to an algebra with clear semantics→ black box, e.g. UDFs, and
  - iii. Other optimization criteria than performance, such as reliability.

Data Flow Optimization (2/2)

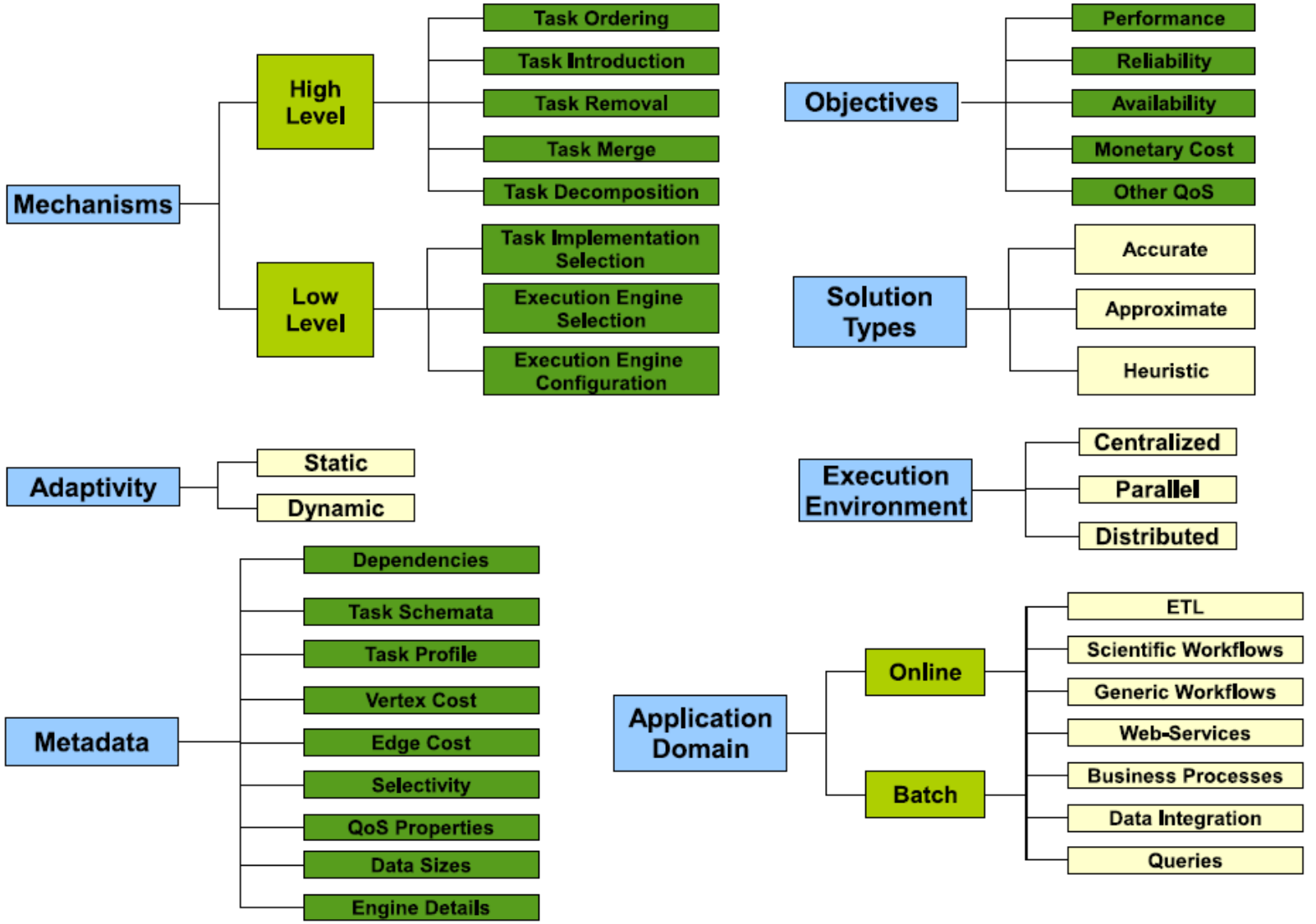
- Architecture of Workflow Management System (WfMS):



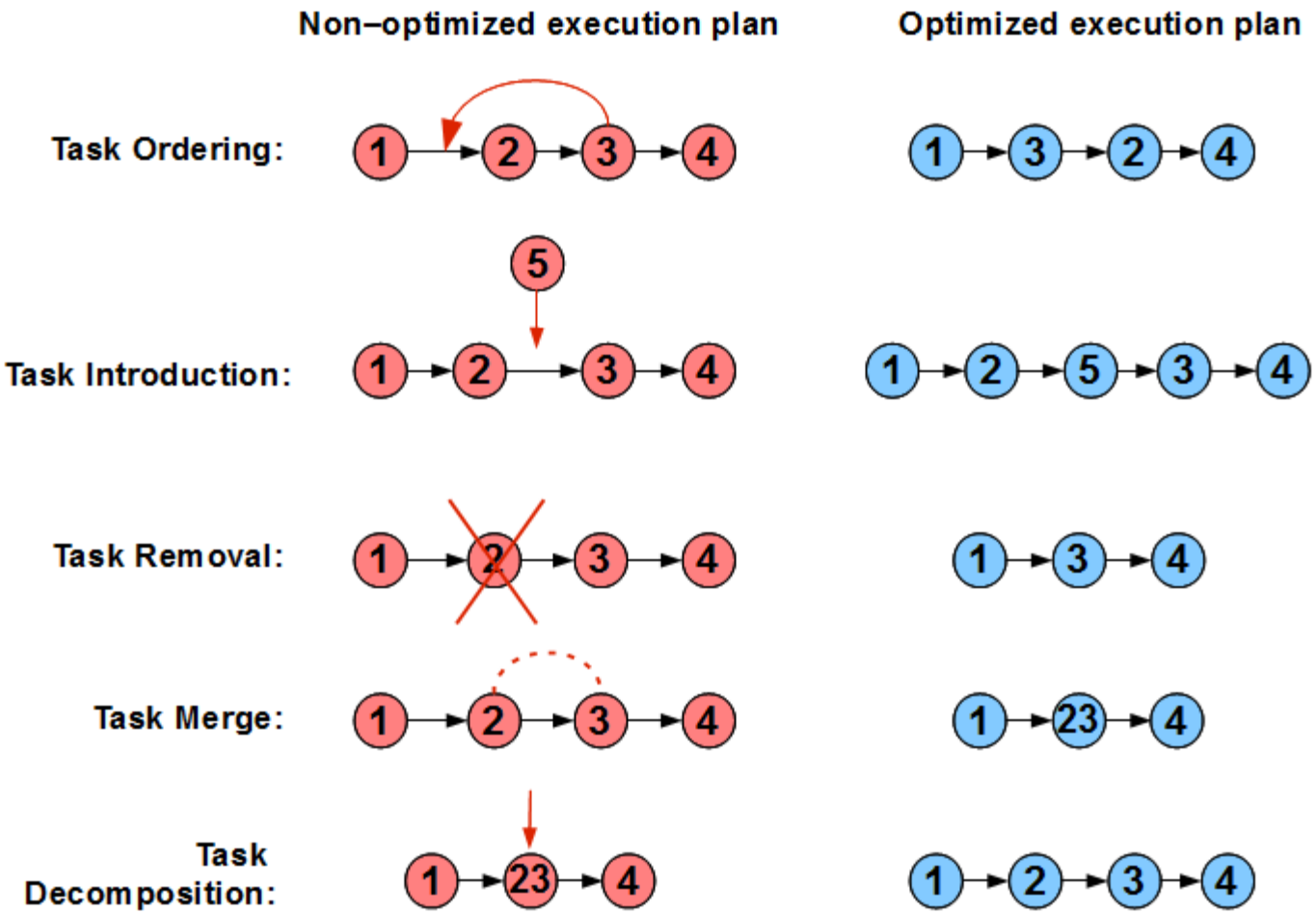
- Query Optimization **VS** Flow Optimization:
  - i. Possible arbitrary dependencies among their execution order,
  - ii. Tasks do not belong necessarily to an algebra with clear semantics→ black box, e.g. UDFs, and
  - iii. Other optimization criteria than performance, such as reliability.

- Optimization Dimensions:
  - ➢ *High Level or Logical* flow plan execution
  - ➢ *Low Level or Physical* flow plan execution

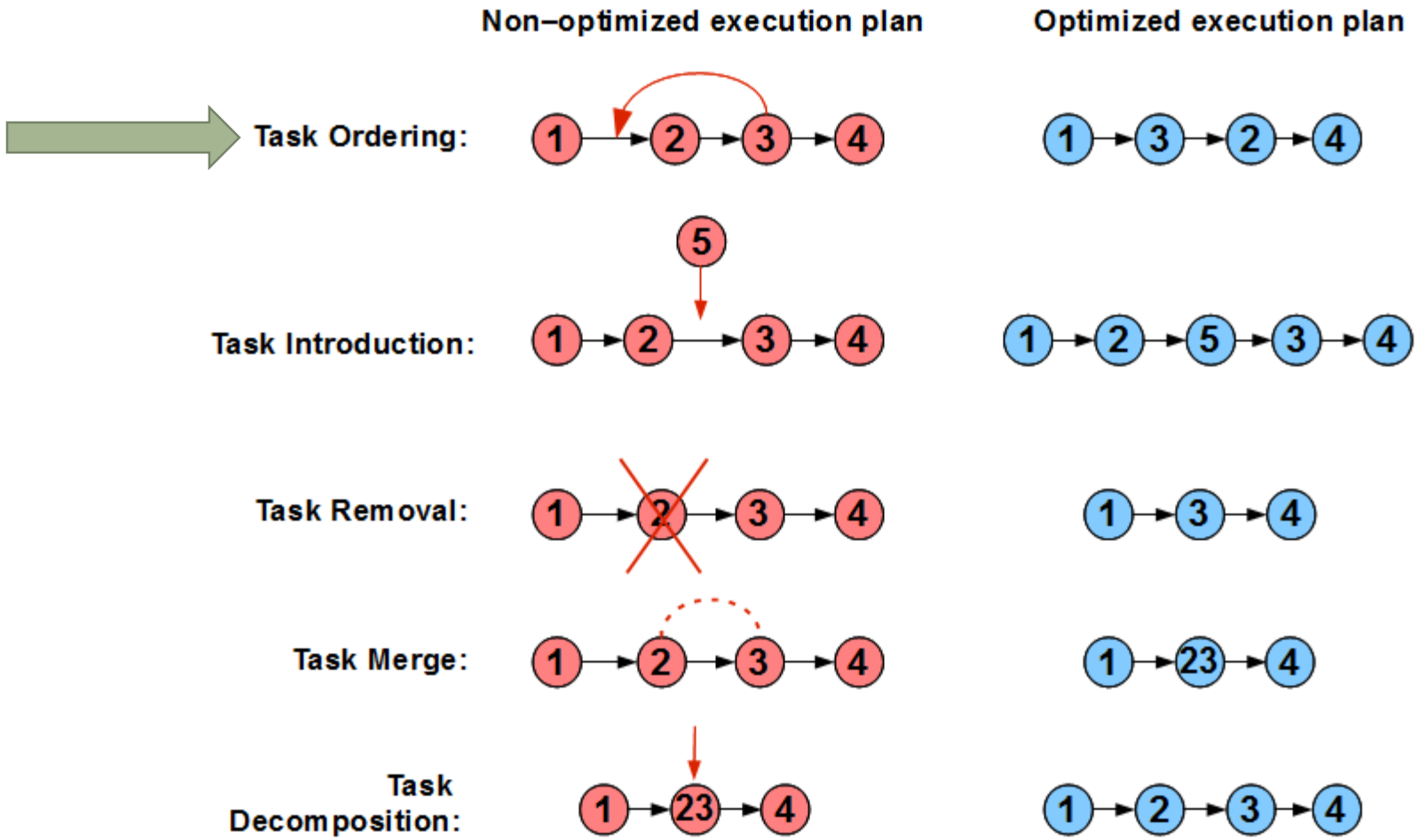
A Taxonomy of Data-centric Flow Optimization



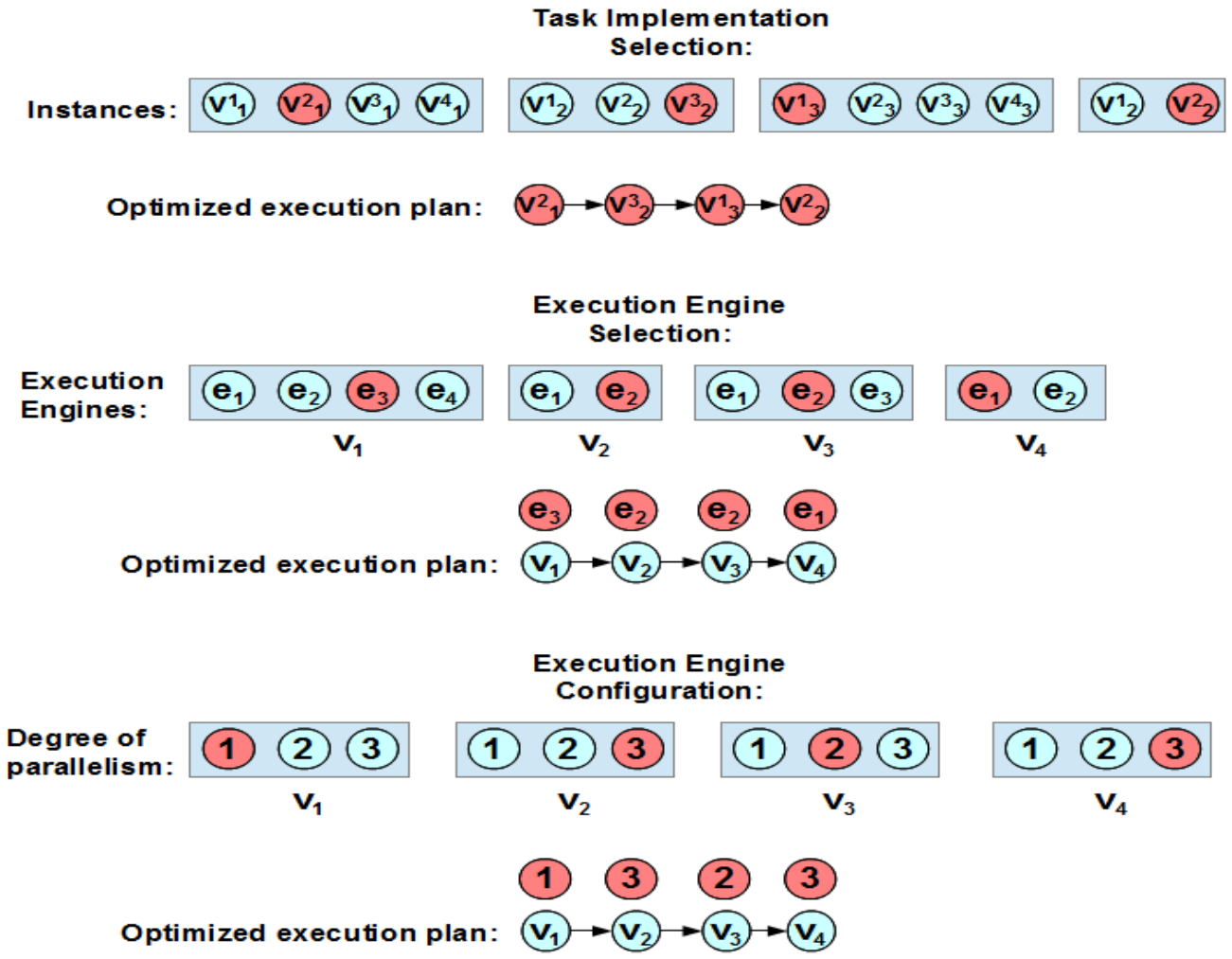
Logical Flow Optimization Mechanisms (1/2)



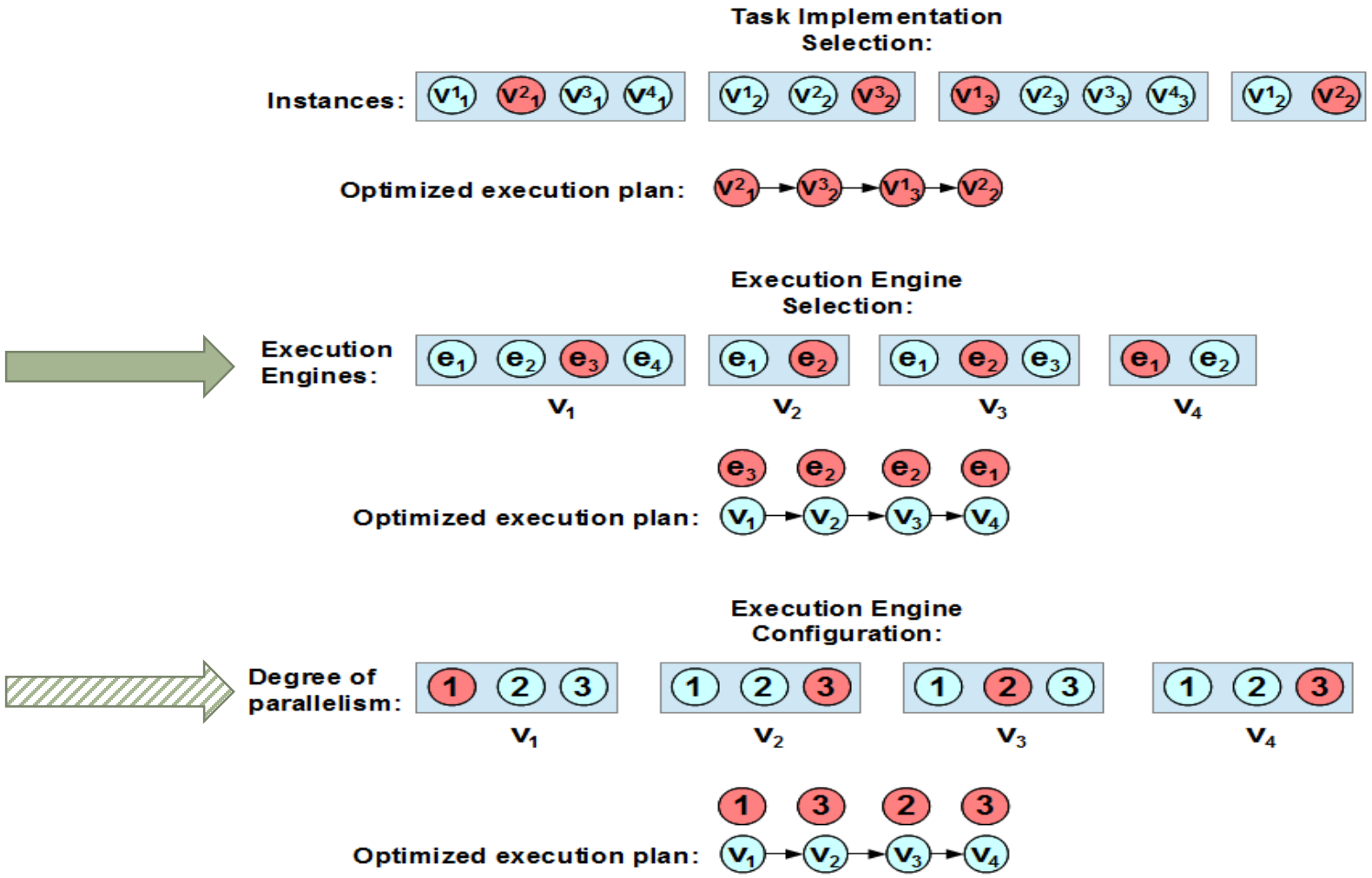
Logical Flow Optimization Mechanisms (2/2)



Physical Flow Optimization Mechanisms (1/2)

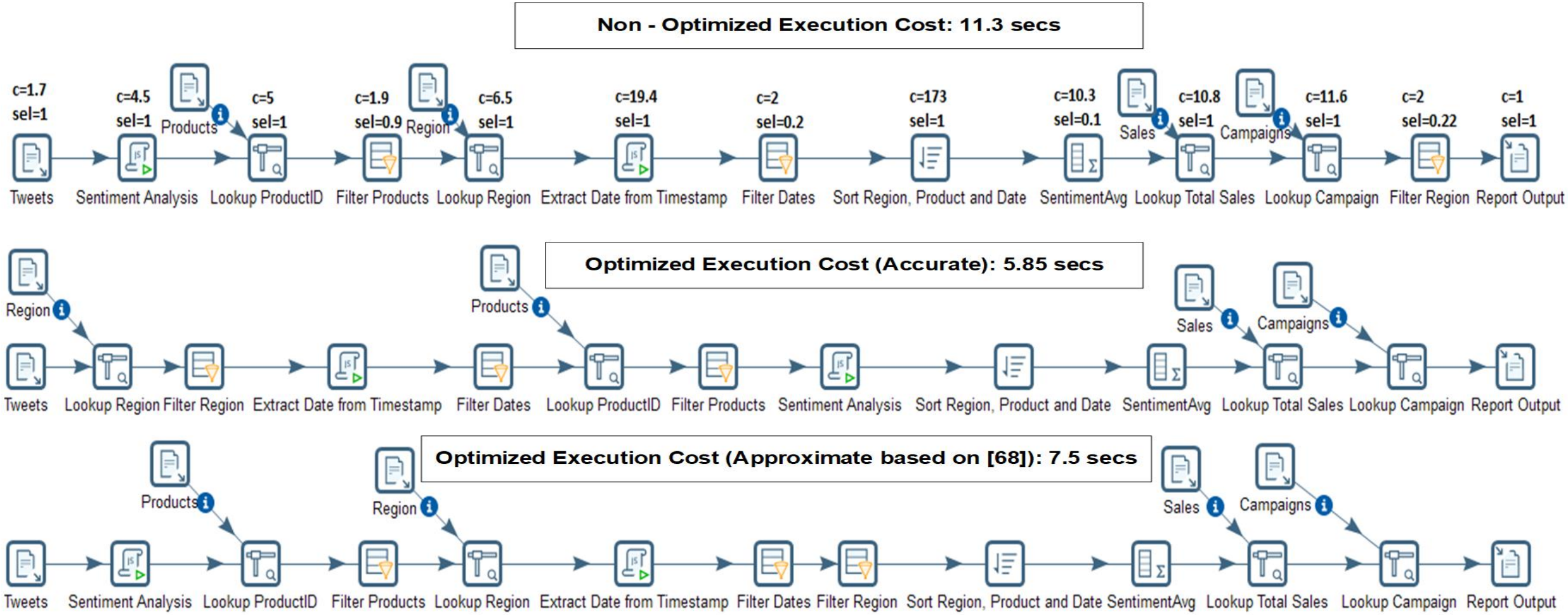


Physical Flow Optimization Mechanisms (2/2)



Task Ordering Mechanism

- Why task re-ordering matters?



Data Flow Metadata

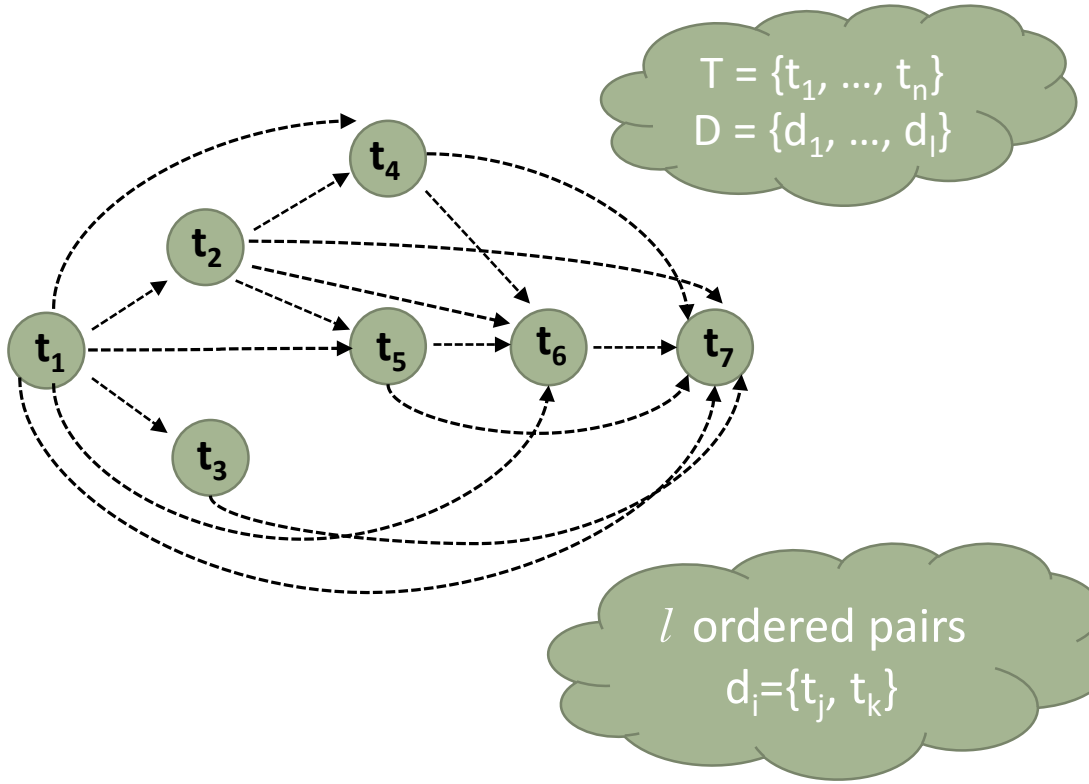
- *Qualitative Metadata*
  - ***Dependencies***

- *Quantitative Metadata*
  - ***Vertex Cost***: time cost (c_i) that a task requires to process one data tuple
  - ***Edge Cost***: cost associated with edges, such as data transmission cost between tasks.
  - ***Selectivity***: the (average) ratio of the output to the input data size of a task (sel_i)

Dependency Graph (1/3)

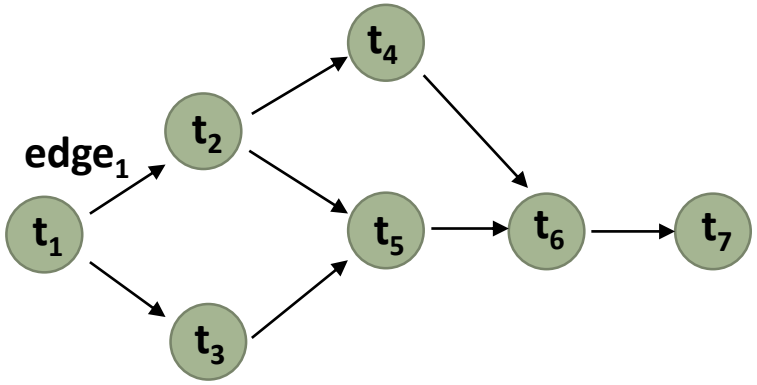
- ***Precedence Constraints*** or ***Dependencies***
- Dependencies on the valid edges of graph *G*
- Graphs G and PC are not semantically equivalent
- Non-executable flow representation
- Partial order representation of the tasks, not the exact execution order

- Dependency Graph (DAG): *PC (T,D)*



Dependency Graph (2/3)

- Execution Plan (DAG): *G=(T,E)*



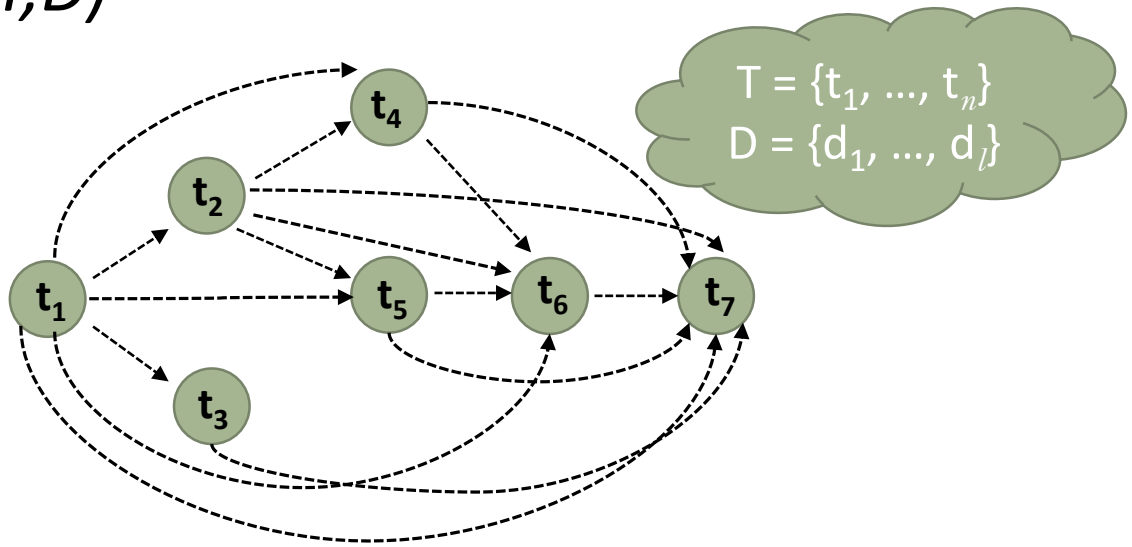
$n = 7$
$l = 16$

DoF = $1 - \frac{2*l}{n(n-1)} = 1 - \frac{2*16}{7(7-1)} = 1 - 0{,}76 = 0{,}24$

Degree of Freedom

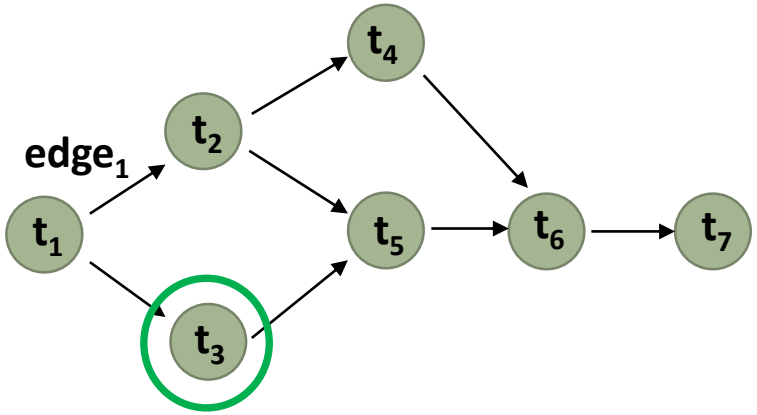
0,24 DoF
or
76% Dependency Constraints

- Dependency Graph (DAG): *PC (T,D)*



Dependency Graph (3/3)

- Execution Plan (DAG): *G=(T,E)*



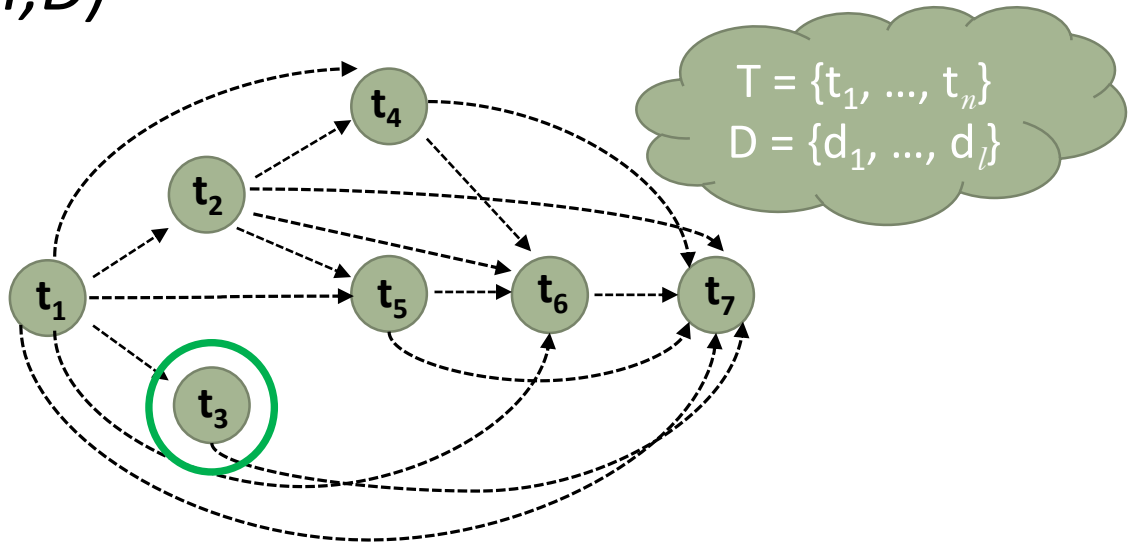
$n = 7$
$l = 16$

DoF = $1 - \frac{2*l}{n(n-1)} = 1 - \frac{2*16}{7(7-1)} = 1 - 0{,}76 = 0{,}24$

Degree of Freedom

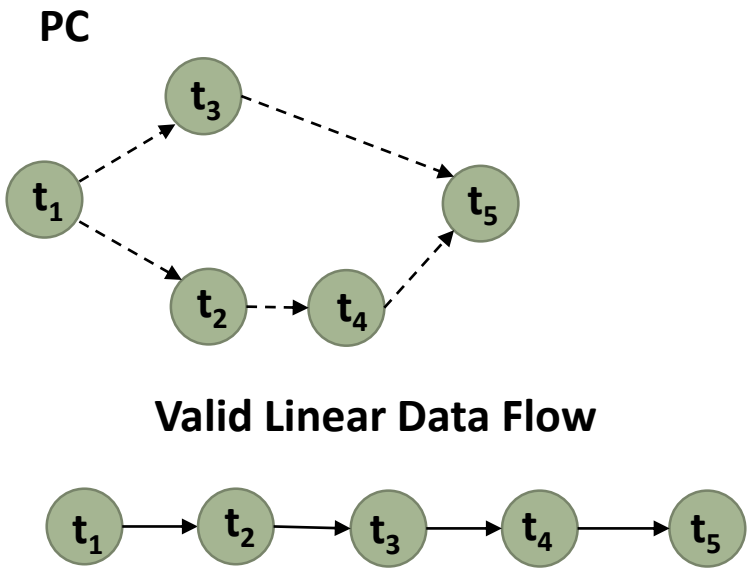
0,24 DoF
or
76% Dependency Constraints

- Dependency Graph (DAG): *PC (T,D)*



Logical Flow Optimization: Task Ordering (1/4)

- Decompose workflows in linear sub-flows/segments
- Optimization of linear sub-flows:
  - Preserving the dependency constraints, is a necessary and sufficient condition for generating a valid execution plan.

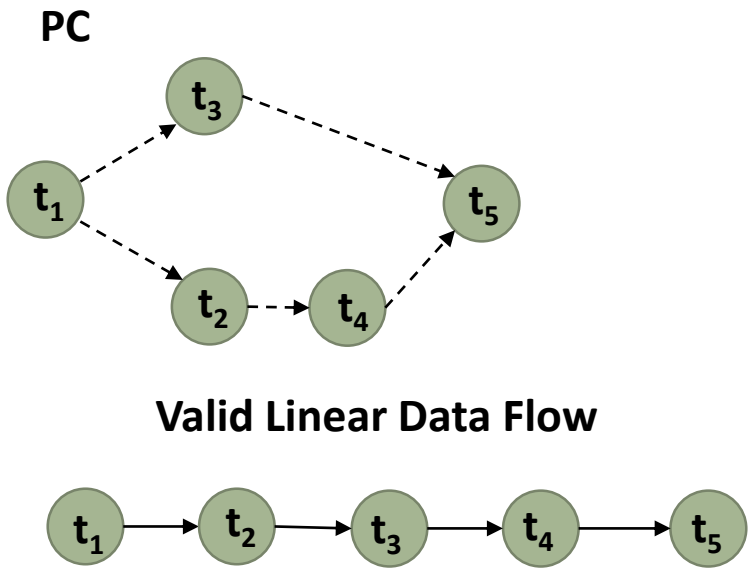


- **Problem definition (Objective function)**:

Minimize the sum of the costs of all tasks (flow execution cost) of a data flow (Sum Cost Metric Minimization)

Logical Flow Optimization: Task Ordering (2/4)

- Decompose workflows in linear sub-flows/segments
- Optimization of linear sub-flows:
  - Preserving the dependency raints, is a necessary and nt condition for generating a execution plan.

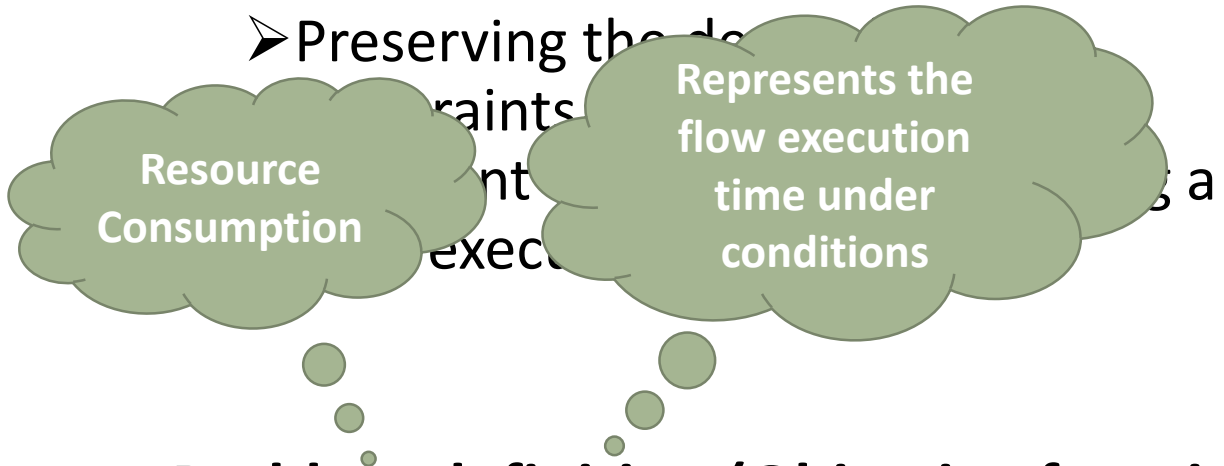


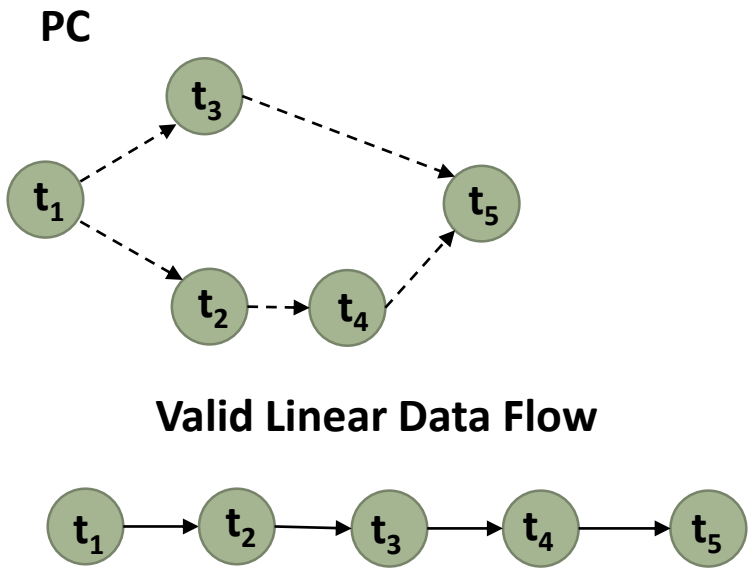
- **Problem definition (Objective function)**:

Minimize the sum of the costs of all tasks (flow execution cost) of a data flow (Sum Cost Metric Minimization)

Logical Flow Optimization: Task Ordering (3/4)

- Decompose workflows in linear sub-flows/segments
- Optimization of linear sub-flows:
  - Preserving the de
    raints
    nt
    exec
    a



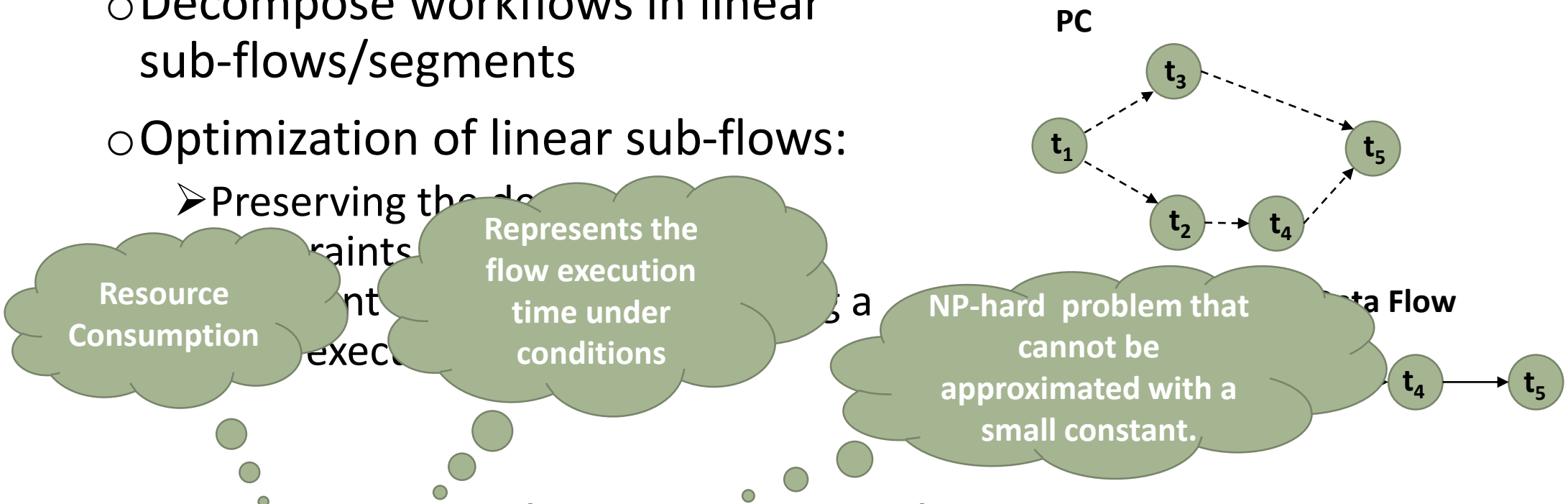


- **Problem definition (Objective function)**:

Minimize the sum of the costs of all tasks (flow execution cost) of a data flow (Sum Cost Metric Minimization)

Logical Flow Optimization: Task Ordering (4/4)

- Decompose workflows in linear sub-flows/segments
- Optimization of linear sub-flows:
  - Preserving the de
    raints
    nt
    exec
    a



- **Problem definition (Objective function)**:

Minimize the sum of the costs of all tasks (flow execution cost) of a data flow (Sum Cost Metric Minimization)

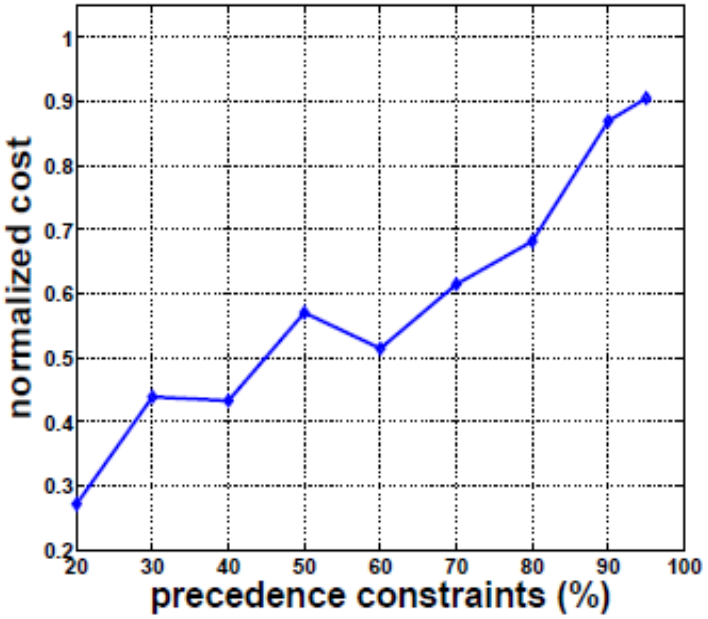
Benefits of Task Ordering Mechanism

- Up to 3 times cost improvement.
- 90%-95% dependencies→ 10-15% cost improvement

Flow examples with a small number of tasks and high percentage of dependencies

- TPC-DI benchmark, where linear flows with <20 tasks and 85-95+% constraints

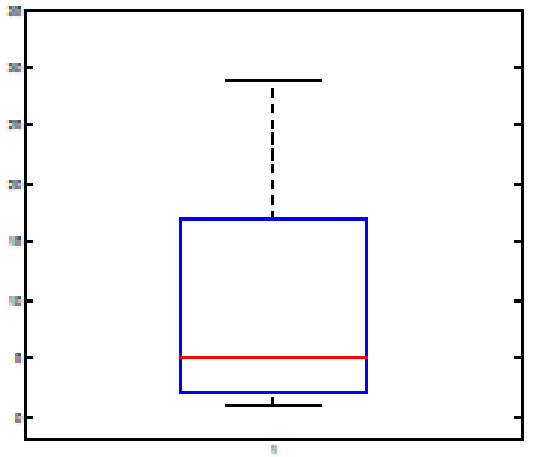
- In such scenarios, accurate algorithms can still be applicable:
  - *Dynamic Programming (DP)*
  - *Finding Optimal Topological Sorting (TopSort)*



100 random flows

n=15
$c_i \in [1,100]$
$sel_i = \in (0,2]$

20-95% dependency constraints



Dynamic Programming Algorithm

- ✓**Accurate for linear data flows**
- ✓Creates an execution plan based on optimal subsets of a flow
- ✓Calculates the cost of task subsets of size n based on subsets of size n-1
- ✓e.g. subsets $\{t_1\}$, $\{t_2\}$, $\{t_3\}$, $\{t_1, t_2\}$, …., $\{t_1 , t_2 , \dots , t_n\}$
- ✓**Time Complexity**: $O(n^2 2^n)$

Dynamic Programming Algorithm

	Costs	Sel	partialPlan
1:	5	0,5	1
2:	10	0,2	2
3:	10	0,1	1 2
4:	15	0,3	…
	…	…	…
8:	12	0,2	4
	…	…	…
16:	25	0,5	5
	…	…	…
24:	17	0,1	4 5
	…	…	…

Subsets of different sizes (2-n)

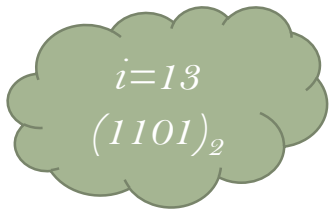
4	5
3	5
…	…

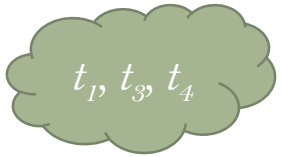
3	4	5
2	4	5
…	…	…

…	…	…	…
…	…	…	…
…	…	…	…

- **E.g. for subset size 2, subset {4 5}:**
  4 – 5 valid order
  5 – 4 invalid order because of dependencies

1. **Position estimation of subsets {4} and {5}:**
   $pos(4)=2^{4-1}=2^3=8$
   $pos(5)=2^{5-1}=2^4=16$
   pos(4-5)= pos(4)+ pos(5)=8+16=24
2. **Cost and selectivity estimation of {4 5}**
   Costs(24)= Costs(8)+ Sel(8)* Costs(16)
   = 12+0,2*25=17
   Sel(24)=Sel(8)*Sel(16)=0,2*0,5=0,1





partialPlan: 1 4 3

Topological Sorting Algorithm

- ✓**Exhaustive production of all the topological sortings.**
- ✓**Time Complexity**: $O(n!)$. Despite the complexity, in practice is applicable for many scenarios.
- ✓Produces the execution plan from the previous one, ensuring the dependency constraints.
- ✓Each plan is based on the previous one, with the minimal amount of changes.
- ✓Two main operations: *Rotation or Swap*

Evaluation (1/2)

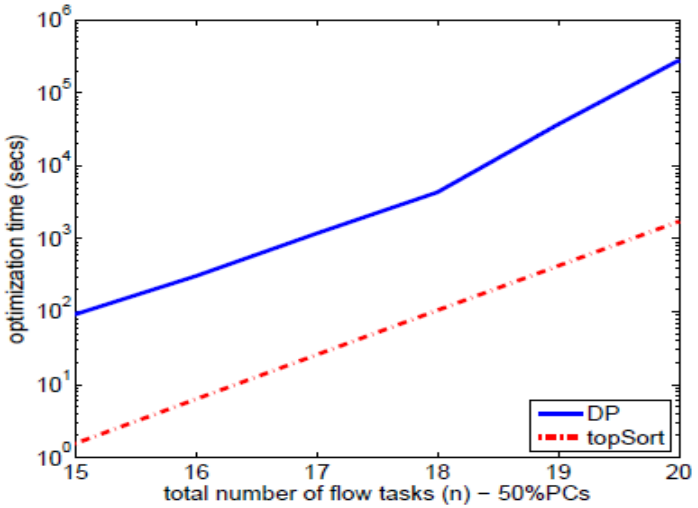
- Experimental Environment
  - Synthetic Data Flows

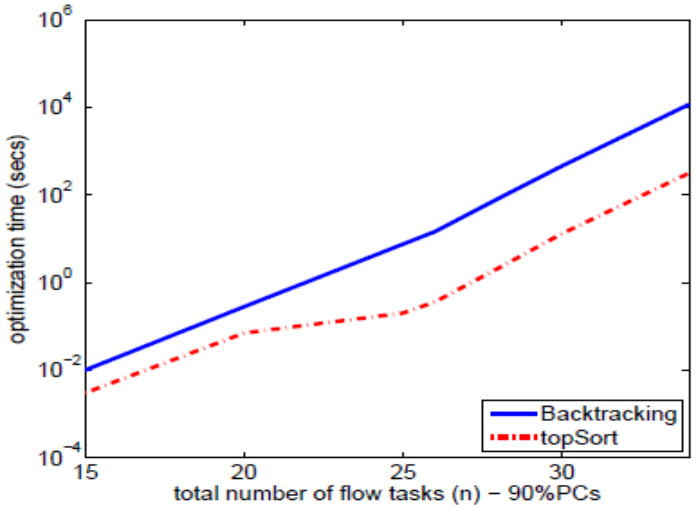
- Algorithm Comparison
  - *Backtracking* (best of State of the Art)→ cannot scale for flows with high percentage of dependencies. Worst case: O(n!)
  - *DP* → cannot scale for execution plans with high number of operators.
  - Compare *TopSort to* each algorithm individually.

Evaluation (2/2)

DP >3 days for n=20

TopSort 50 times faster



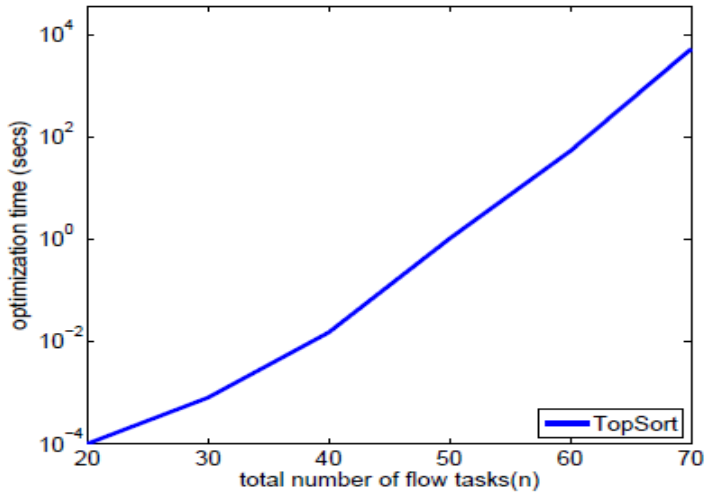


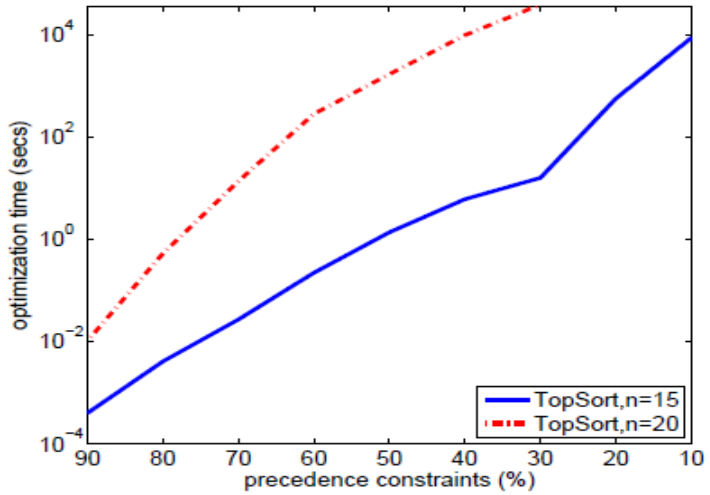
Backtracking orders of magnitudes slower

n=30 Backtracking 459 secs TopSort 13 secs

PC=98% e.g. TPC-DI

n=60 1 min





Conclusions

- *TopSort* algorithm has better performance for linear flows.
- Topological sorting on a previous sorting, with the minimal amount of changes.
- Exploiting dependency constraints
- Applicability:
  - TopSort: 60 tasks and 98% constraints → improvement in 1 minute
  - TopSort: 15 tasks and 30% constraints → improvement in < 1 minute
  - Best of State-of-the-Art: 35 times slower, in scenarios that is applicable
  - Best of State-of-the-Art: inapplicable for low percentage of dependency constraints
- Despite the performance improvements, the accurate techniques cannot scale.

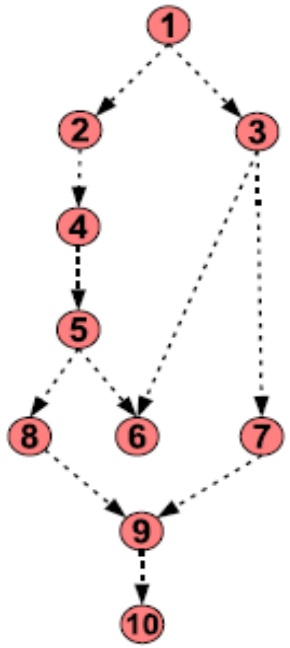
Rank Ordering Algorithm

- Demanding need to adopt innovative algorithms that scale for linear flows.

- Rank Ordering algorithm in abstract level
  1. **Pre-processing phase**: Modify PC graph
  2. **Apply the modified *KBZ* algorithm**: Generation of the execution plan
  3. **Post-process phase**: Improve the G graph

RO-I Algorithm (1/2)

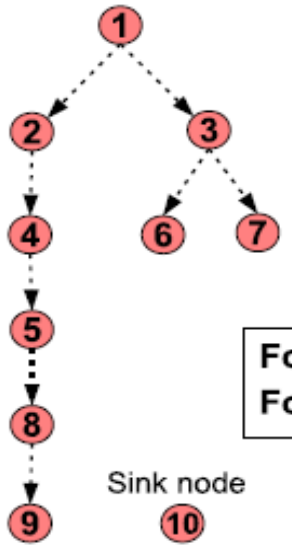
Initial pruned PC graph



Label	Rank
1	0
2	-0.0071
3	0.1167
4	-0.0152
5	-0.0021
6	0.0101
7	-0.0029
8	0.0176
9	0.0021
10	0

Initial Cost : 317.3132

Pre-processing phase

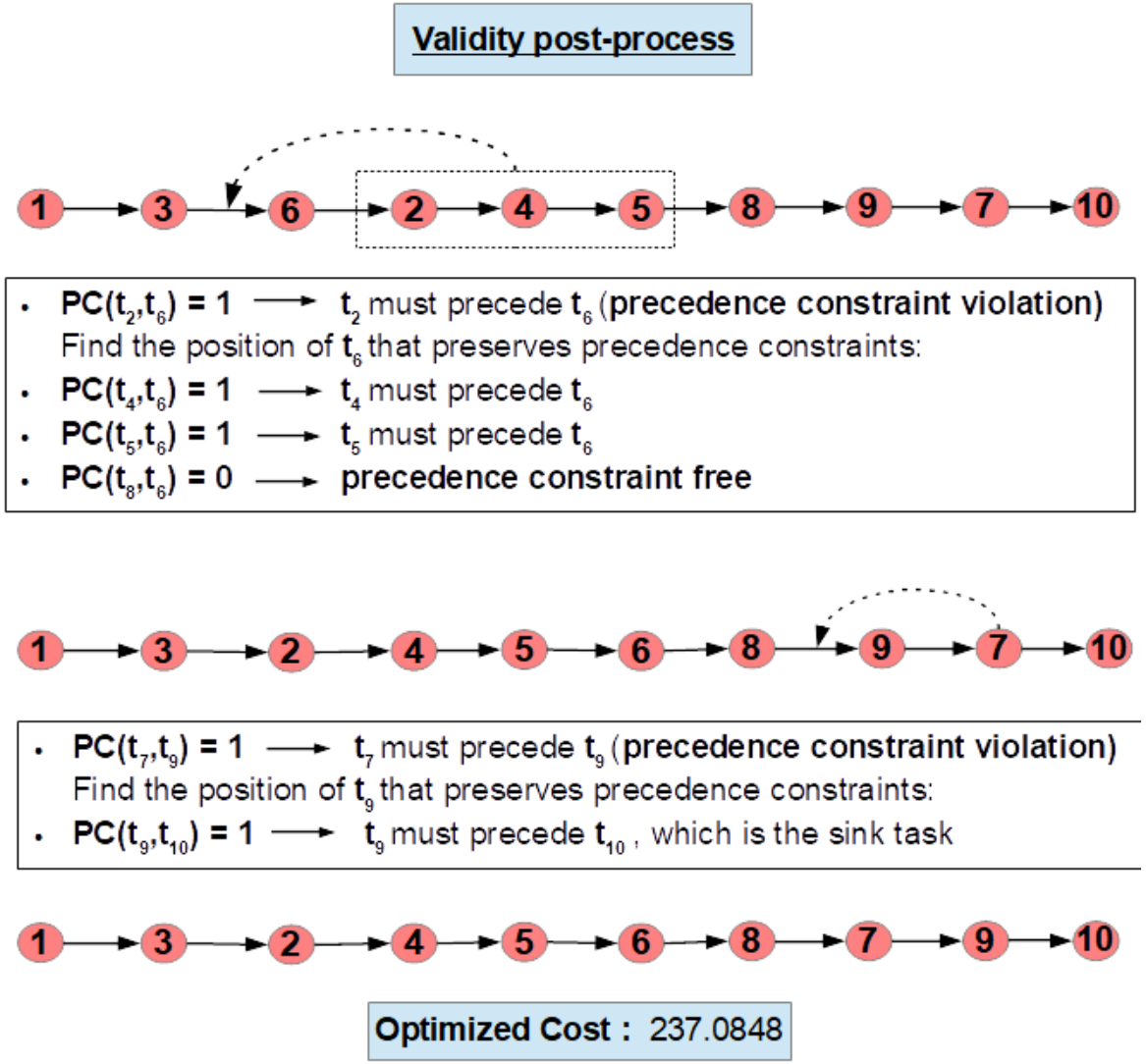


For t_6: $rank_3 > rank_5$ ⟶ $PC(t_5,t_6)=0$
For t_9: $rank_8 > rank_7$ ⟶ $PC(t_7,t_9)=0$

- Pre-processing phase: convert the PC graph to a tree
- **rank=$(1-sel_i)/c_i$**

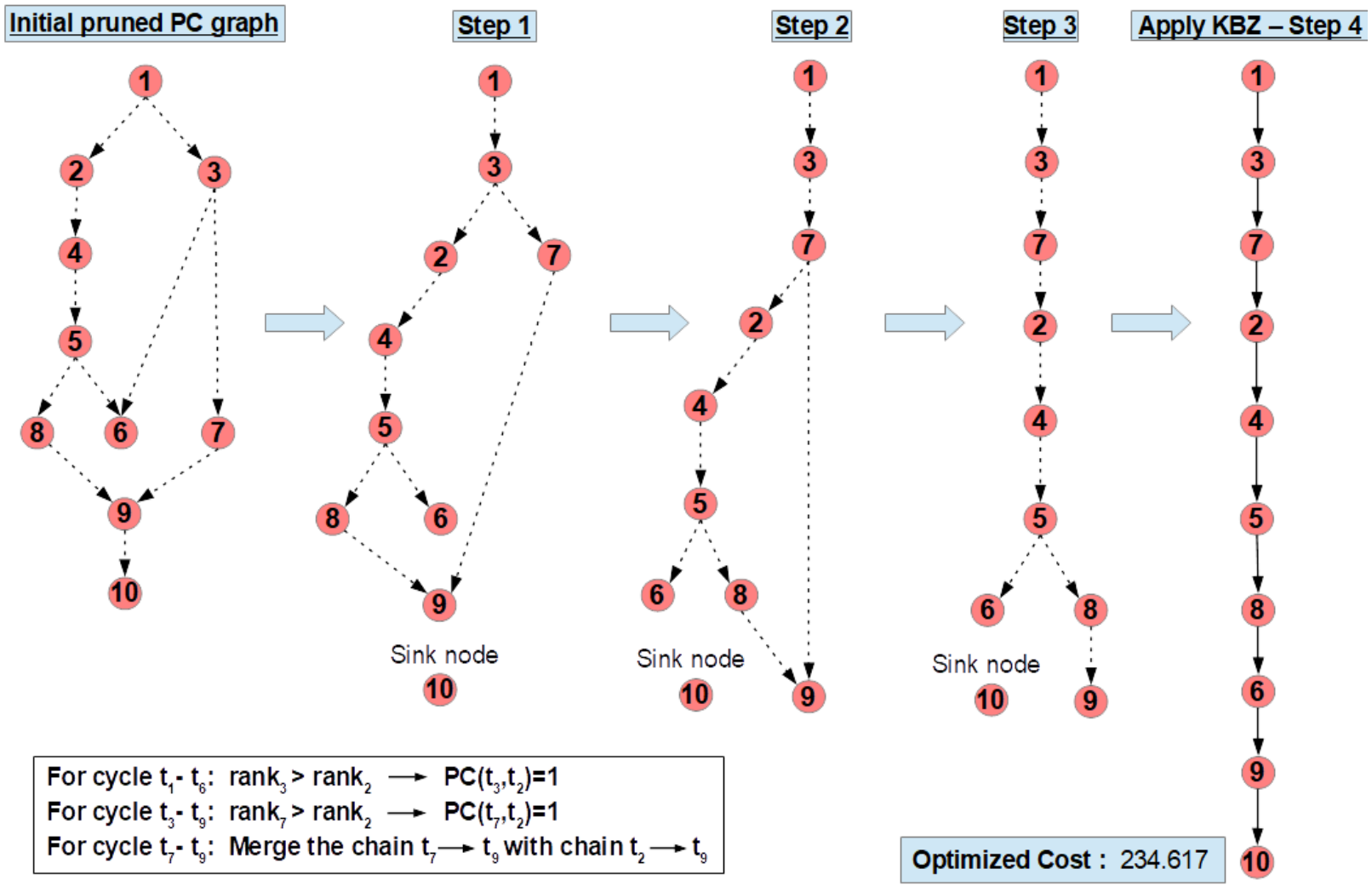
Label	1	2	3	4	5	6	7	8	9	10
c	1	98	6	46	94	79	34	51	96	1
sel	1	1.7	0.3	1.7	1.2	0.2	1.1	0.1	0.8	1

RO-I Algorithm (2/2)



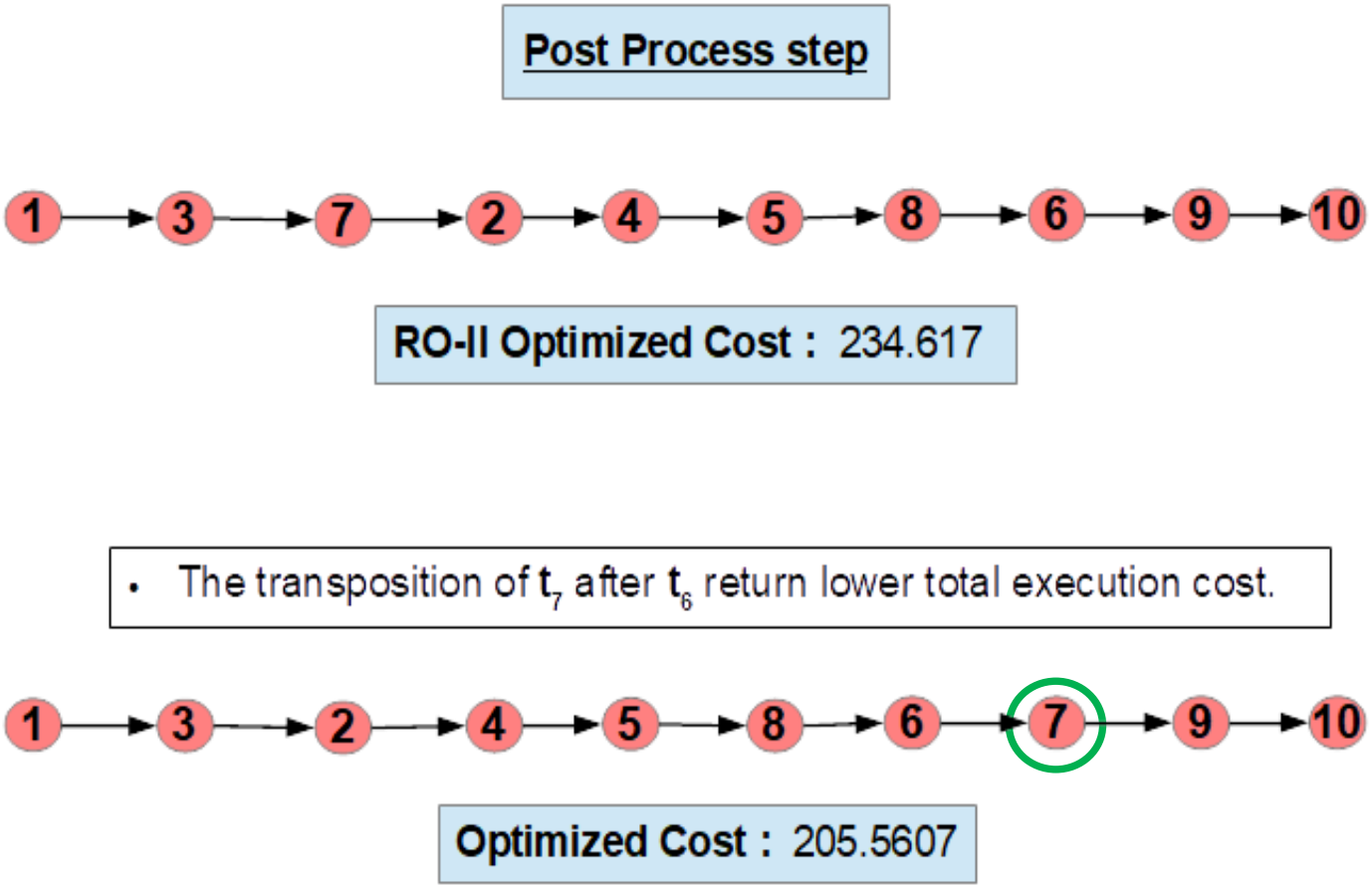
- Generation of temporarily non-valid plans
- Corrections with post-process phase
- Time complexity: $O(n^2)$

RO-II Algorithm



- Valid plan generation, but with more restrictions
- Post-process procedure is not necessary
- Time complexity: $O(n^2)$

RO-III Algorithm



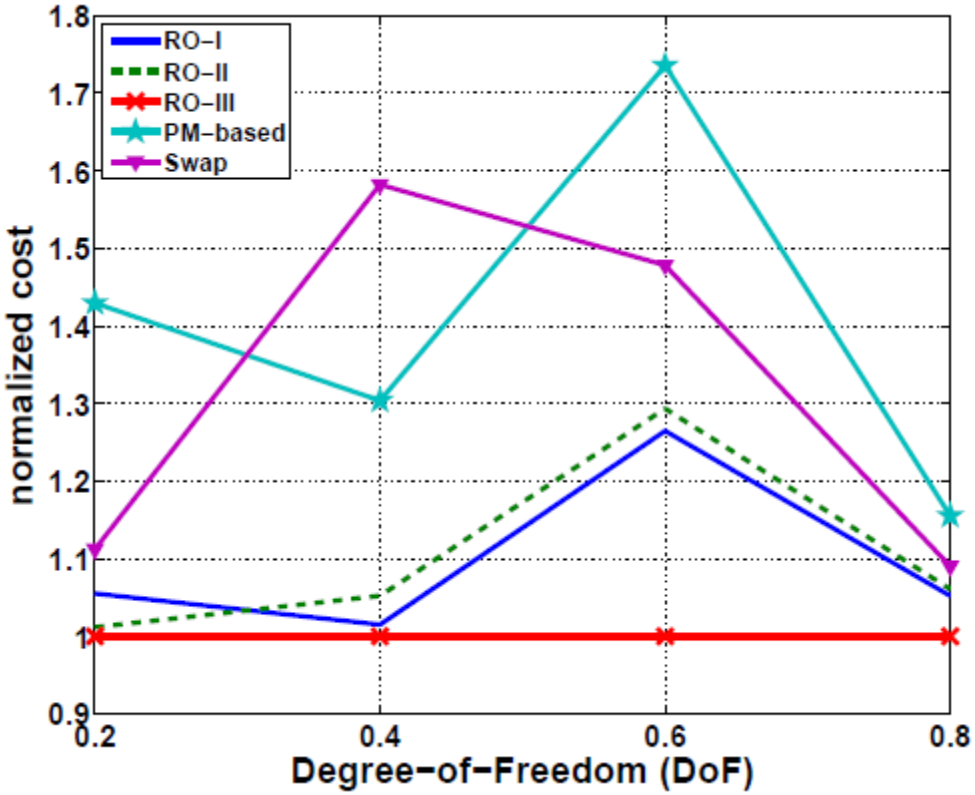
- *RO-I and RO-II* deviates from the optimal solution
- Valid plan generation, but with more restrictions
- Post-process procedure for further improvement
- Checks all the possible re-orderings of each sub-flow of size from 1 to k from the left to the right
- Time complexity of the phase: $O(kn^2)$

Important Observations

- The *RO-I* and *RO-II* outweigh the techniques of the state-of-the-art.
- There is no one superior.
- The RO-III is better than RO-II, because it extends it.

Experimental Evaluation (1/2)

- Experiments in a real execution environment Pentaho Data Integration (PDI) for 100.000 records:



$c_i \in [1, 100]$
$sel_i = \in (0, 2]$

n = 30

Speed-up $= \frac{SCM(B)}{SCM(A)}$

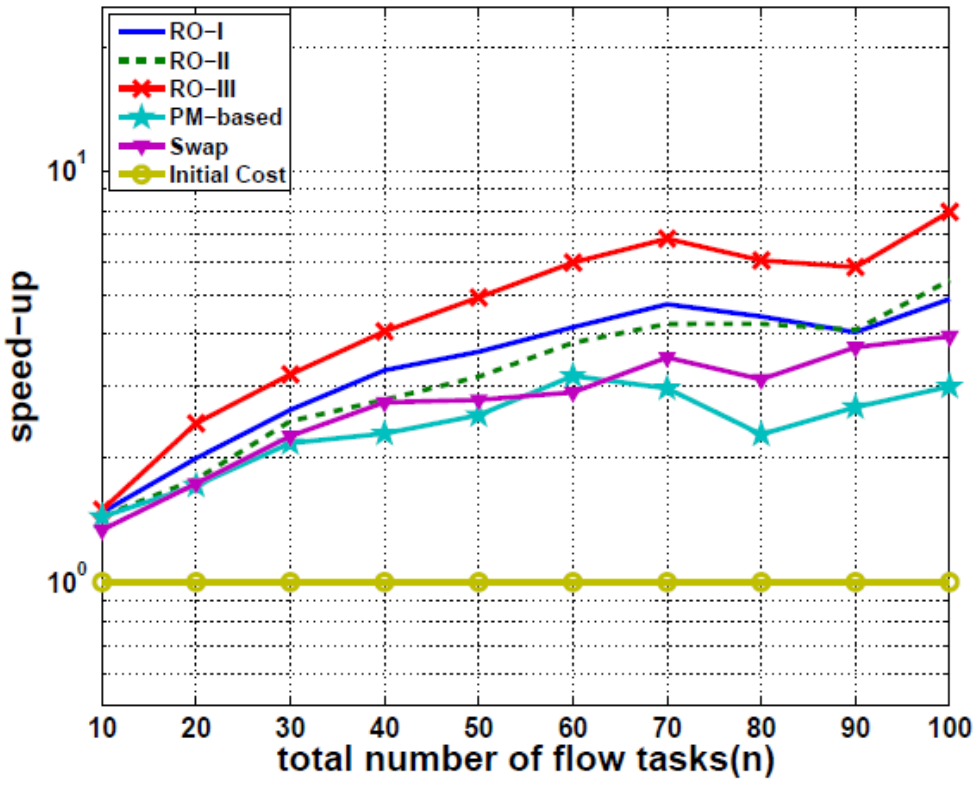
Maximum speed-up of *RO-III*

alg\DoF	0,2	0,4	0,6	0,8
PM-based	1,84	2,41	**4,77**	2,06
Swap	1,84	3,30	**3,62**	1,39

Performance improvement (average value)
DoF=0,4 → 30,3%
DoF=0,6 → 47,8%

Experimental Evaluation (2/2)

- Experiments with synthetic data flows



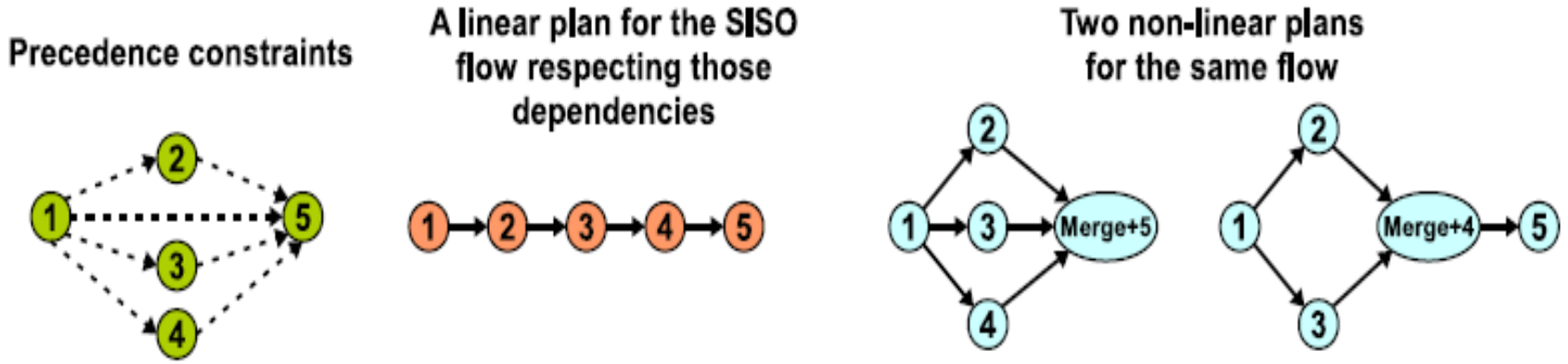
Degree-Of-Freedom (DoF): 0,4

RO-III 3 times better for n= 100 in average

DoF=0,4							
	RO-III better			same	RO-III worse		
n	♯times	avg	median	♯times	♯times	avg	median
10	47	1,1993	1,1081	53	0	-	-
20	96	1,4283	1,2477	1	3	1,2489	1,0817
40	100	1,8940	1,3474	0	0	-	-
60	99	2,0278	1,4952	0	1	1,0909	1,0909
80	100	2,2472	1,5798	0	0	-	-
100	99	3,5996	1,5130	0	1	1,0793	1,0793

Non-linear Data Flows

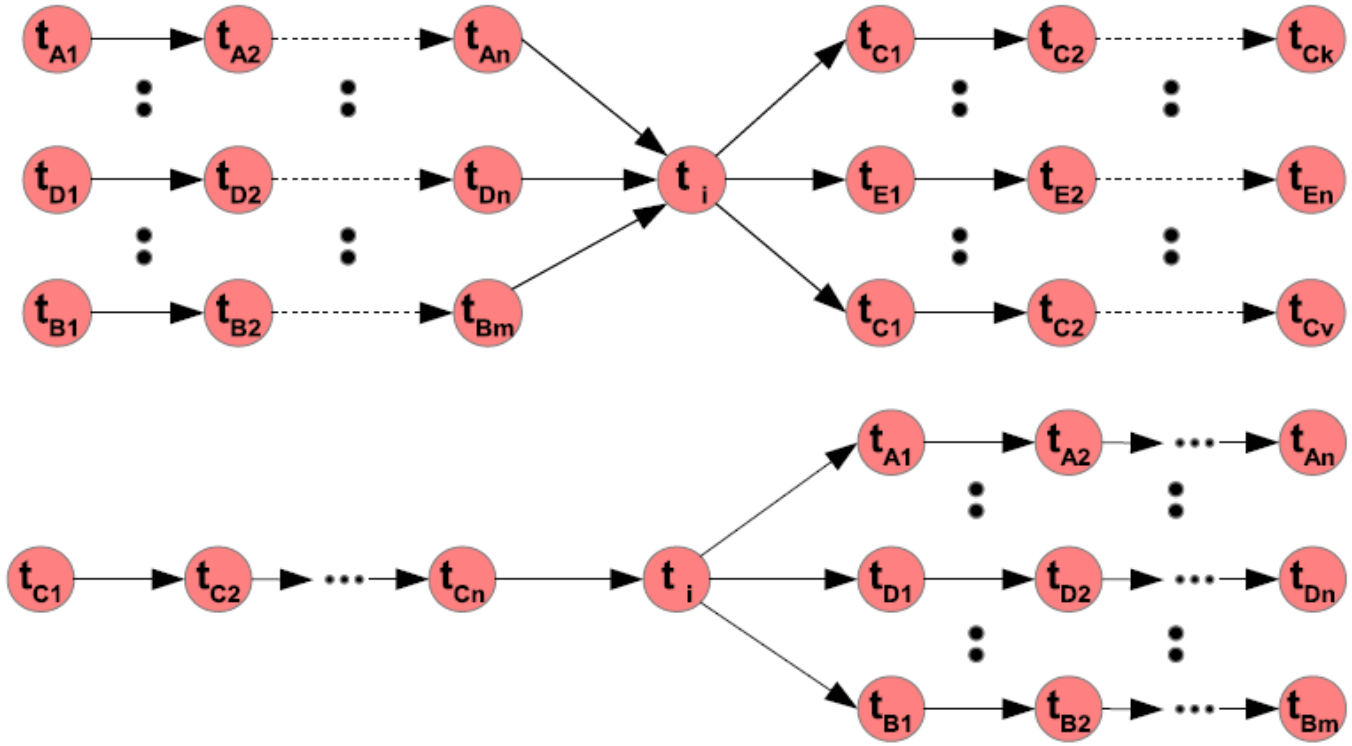
- Non-linear flow plan generation for optimization
- Merge cost of the input data: *merge cost (mc)*
- Process phase of a linear flow plan:
  - Execution of successive tasks with sel>1, while the dependency constraints are preserved



- Time Complexity: $O(n^2)$

MIMO Workflows

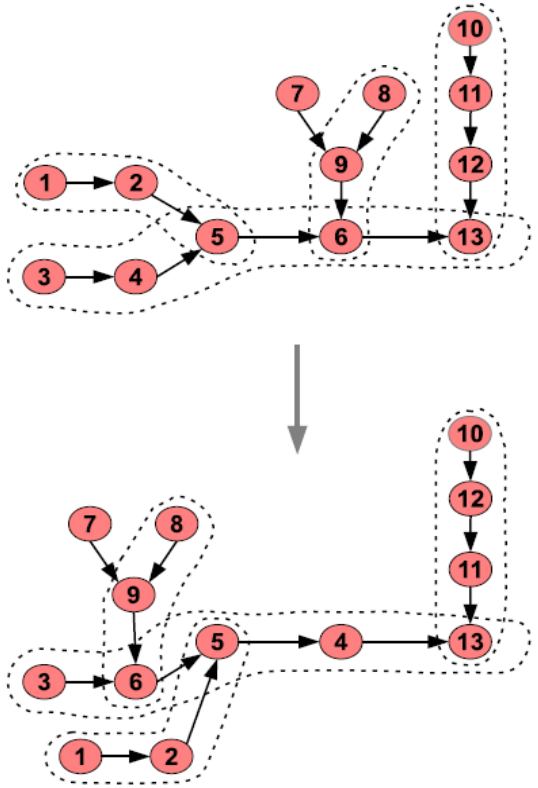
- *Multiple Input Multiple Output (MIMO)*



Start from an initial valid plan:
Step 1: Extract linear sub-flows
Step 2: Optimization of the linear sub-flows based on RO-III
Step 3: Re-ordering checking between binary and unary operators
Step 4: Repeat the above steps until no changes

Data Flows SIMO and MISO (1/2)

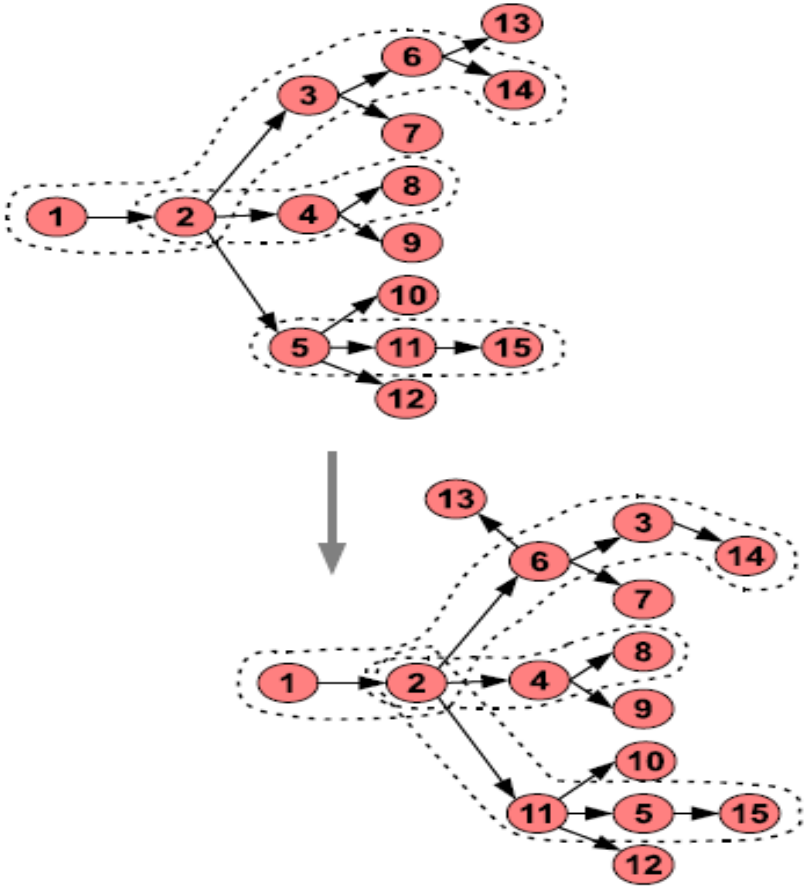
- *Multiple Input Single Output (MISO) Case*



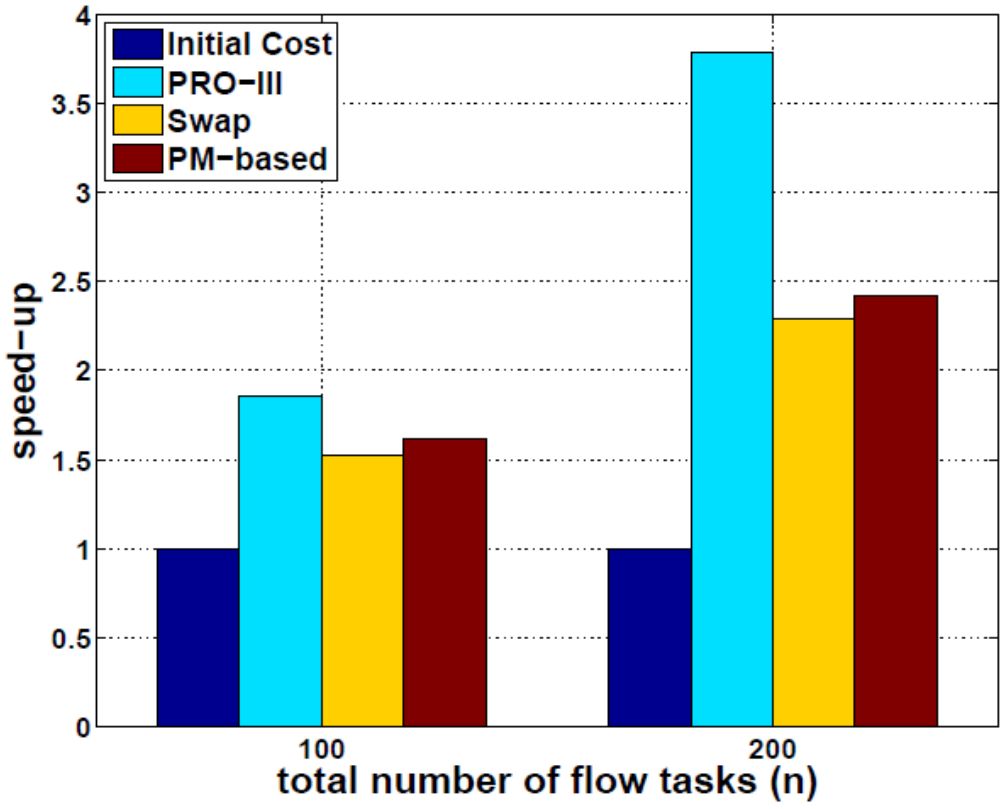
- L-SISO with maximum path length (branch):
  L-SISO segment : 1 – 2 – 5 – 6 – 13 (1)
  L-SISO segment : 3 – 4 – 5 – 6 – 13 (2)
- L-SISO with minimum sum of ranks:
  rank(1) > rank(2)
- Optimize L-SISO (2):
  L-SISO segment : 3 – 4 – 5 – 6 – 13
  Optimized L-SISO: 3 – 6 – 5 – 4 – 13
- L-SISO with maximum path length:
  L-SISO segment : 10 – 11 – 12 – 13 (3)
  Optimized L-SISO: 10 – 12 – 11 – 13
- L-SISO with maximum path length (branch):
  L-SISO segment : 7 – 9 – 6 (4)
  L-SISO segment : 8 – 9 – 6 (5)
- L-SISO with minimum sum of ranks:
  rank(4) > rank(5)
- Optimize L-SISO (5):
  L-SISO segment : 8 – 9 – 6
  Optimized L-SISO: 8 – 9 – 6
- L-SISO with maximum path length:
  L-SISO segment : 1 – 2 – 5 (6)
  Optimized L-SISO: 1 – 2 – 5

Data Flows SIMO and MISO (2/2)

- *Single Input Multiple Output (SIMO) Case*



Experimental Evaluation

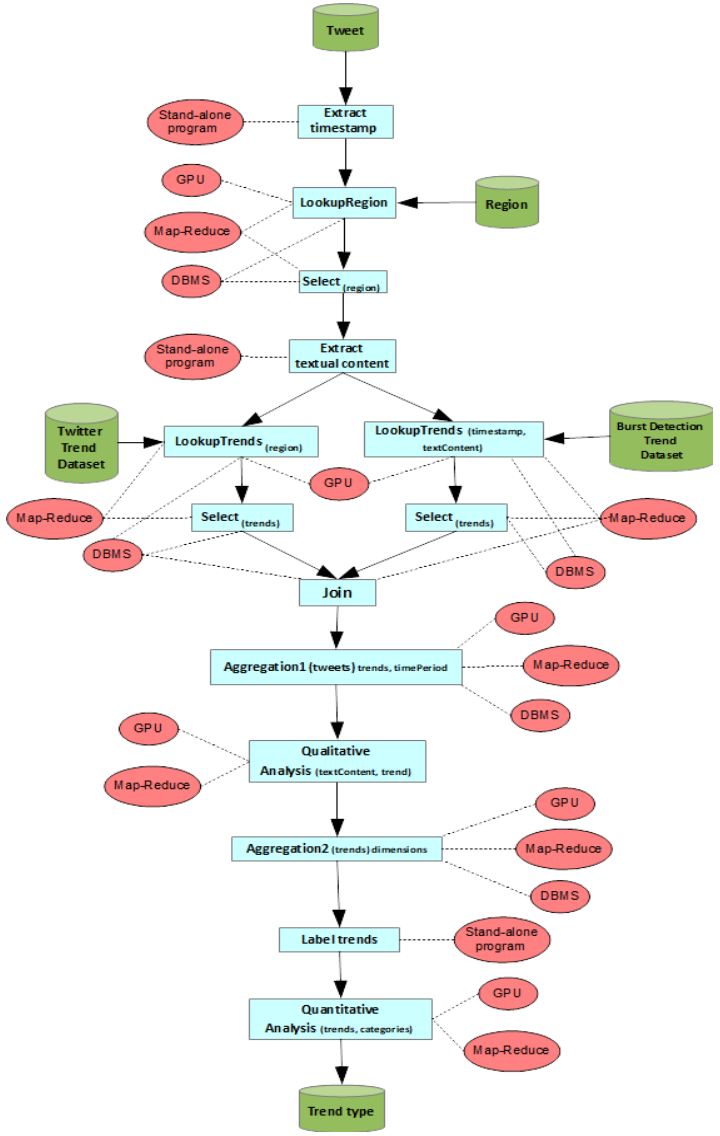


- 10 sub-flows with:
  - ➢ 10 tasks (flow with 100 tasks)
  - ➢ 20 tasks (flow with 200 tasks)
- *DoF* linear subset 0.6
- For subset of size 10:
  - ➢ Up to 86% improvement (PRO-III)
- For subset of size 20:
  - ➢ 3.8 times improvement (PRO-III)

➢ 62% PRO-III better than Swap

➢ 55% PRO-III better than PM-based

Engine Selection Mechanism (1/3)



- Flow with 14 tasks
- Possible allocation of tasks to engines:
  - $2^6 3^5 = 15.552$
  - Increases exponentially in the number of available candidate engines
  - m^n : possible allocations, where the m is the number of the execution engines
- Data transfer/ switching between different engines → possible extra cost
- Execution engine constraints

Engine Selection Mechanism (2/3)

- **Problem**: *Allocating flow activities to specific heterogeneous and independent execution engines*

- Problem challenges:
  - The problem is NP-hard.
  - The number of flow nodes and candidate execution engines may be large.
  - Execution engines are heterogeneous.
  - Shipping data and engine switching incurs extra cost (this makes the problem NP-hard).

Engine Selection Mechanism (3/3)

- **Optimization Objective**→ *minimize the sum of all the costs of each task on the engines that has been allocated (flow execution cost) + the cost of data shipping between different engines and engine switching*

- In general, flow task allocation to execution engines in order to:
  i. minimize the total execution cost,
  ii. each task is allocated to a single execution engine and
  iii. the allocations algorithms are executed in a couple of seconds

Metadata

- Valid logical execution plan
- Set of execution engines that activities can be allocated to (*m size*)
- Execution cost of a task when mapped to an execution engine e_j ($c_{i,j}$):
  - possibility of different task implementations for each execution engine
  - time units
  - different for each execution engine or different engine configurations
- Graph/Flow edge cost ($ce_{e_{i\to j}}^{t_k}$) or *inter-engine cost*:
  i. cost of engine switching from an engine e_i to e_j
  ii. data transfer from the output of task t_k to a subsequent task
  iii. no cost for data transfer between tasks allocated to the same execution engine
- Execution engine constraints

Simple Heuristics

Algorithm input

C

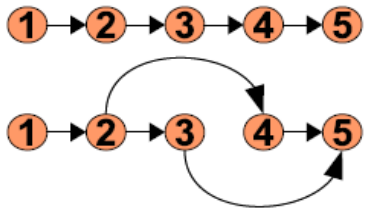
6	8	2
5	8	3
9	2	5
5	9	2
4	7	4

CE

0	2	7
6	0	2
8	3	0

CONSTR

0	1	1
1	1	1
1	1	1
0	0	1
1	1	0



linear flow

n = 5
m = 3

non-linear flow

H1 and H2

	H1			Allocation
	6	8	2	3 :f(1)
	5	8	3	3 :f(2)
	9	2	5	3 :f(3)
	5	9	2	3 :f(4)
	4	7	4	1 :f(5)
average	5.8	6.8	3.2	

H1 allocation plan: ***f*** = 3 3 3 3 1
H1 non-linear allocation cost: 32
H1 linear allocation cost: 24

H2			Allocation
6	8	2	3 :f(1)
5	8	3	3 :f(2)
9	2	5	2 :f(3)
5	9	2	3 :f(4)
4	7	4	1 :f(5)

H2 allocation plan: ***f*** = 3 3 2 3 1
H2 non-linear allocation cost: 30
H2 linear allocation cost: 26

- ***H1***:
  - rank all engines based on their average execution cost for all flow activities in increasing order
  - allocate each task to the engine with the lowest cost that is capable of executing that activity
- ***H2***:
  - rank all engines based on their execution cost for each flow activity separately
  - allocate each task to the engine with the lowest cost that is capable of executing that activity

Anytime Algorithms (1/3)

- *Branch and Bound-Iteration Capping (**BB-IC**) Algorithm*
  - *calculates the flow execution cost after each task gets allocated*
  - *if the cost exceeds the current minimum cost up to this time (between H_1 and H_2) or violates the engine selection constraints*
  - *this intermediate allocation plan is abandoned*
  - *e.g. if the 3rd task allocation to 4th engine is invalid*, then we do not examine the allocations *based on this mapping*
  - *exponential complexity* ☹
  - *cap the number of the allowed iterations (allows only a prespecified number of iterations)* ☺
  - *investigate other allocations for the parts of the flow that contribute the most to the total cost* ☺

Anytime Algorithms (2/3)

- Random-walk (***RWR-b***):
  - *starting from the allocation derived from the best performing heuristic between H_1 and H_2*
  - *random perturbations for a pre-specified number of times (r)*
  - *re-starting point is the best performing allocation detected thus far*

- ***RWR-b*** and ***BB-IC*** :
  - explore only a very small part of the search space
  - e.g. ***BB-IC*** for 10,000 number of iterations and n=m=100 checks only the possible change of 2 tasks
  - propose algorithms in order to prune the search space: *set-cover algorithms*

Anytime Algorithms (3/3)

- Set Cover (***SC1*** και ***SC2***) Algorithms
  - **SC1**: (i) reduces the total number of candidate execution engines *ENG*
    (ii) selects the engine that is capable of processing the most activities
  - **SC2**: (i) 1st task allocation, similar to SC1
    (ii) chooses the engine with the lowest average inter-engine cost with respect to the last added engine across all activities
  - Apply both *BB-IC* and *RWR-b* to the reduced engine set
- The maximum number of iterations in the pre-processing phase: *m*
- The number of iterations or restarts of *BB-IC* and *RWR-b* define the actual execution time of SC flavors

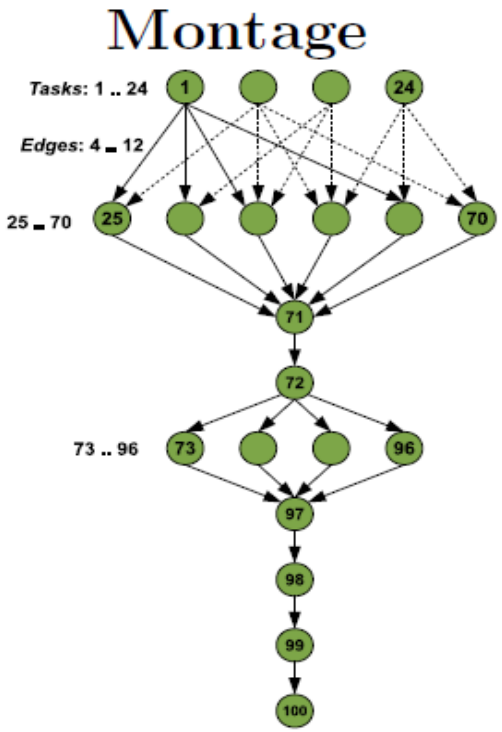
Hybrid Solutions

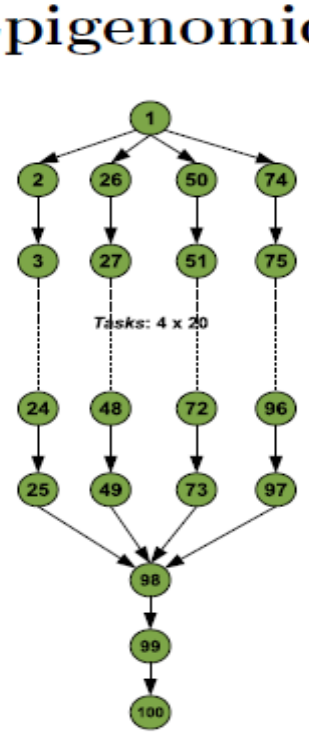
- A hybrid solution (***BEST***): metaheuristic algorithm that chooses the allocation plan with the lowest execution cost among *BB-IC*, *RWR-b*, *SC1* and *SC2*.

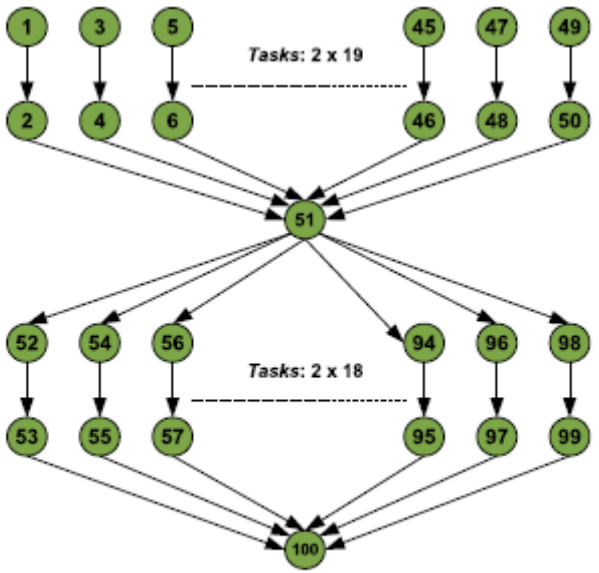
- Performance increase up to 10%

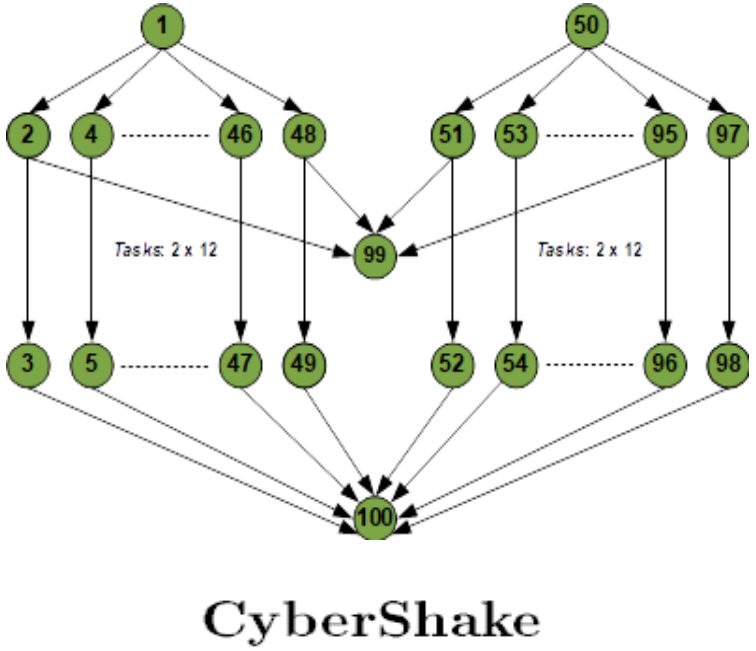
- Dynamic Programming Algorithm
  - Optimal Solution for linear flows
  - Approximate solution for general flow plans

Experimental Evaluation (1/3)









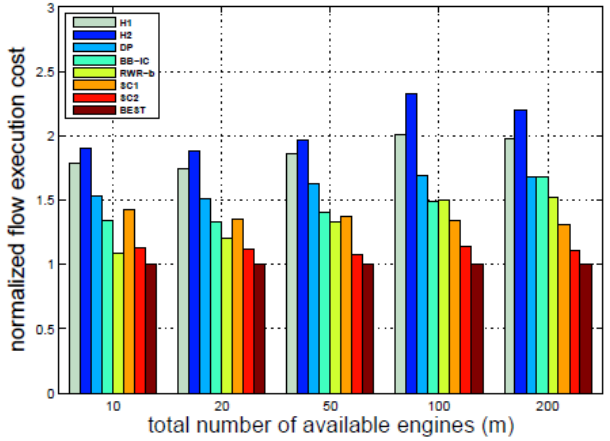
n = m = 100

Table 1: Normalized performance for real-world flows with 50% engine constraints

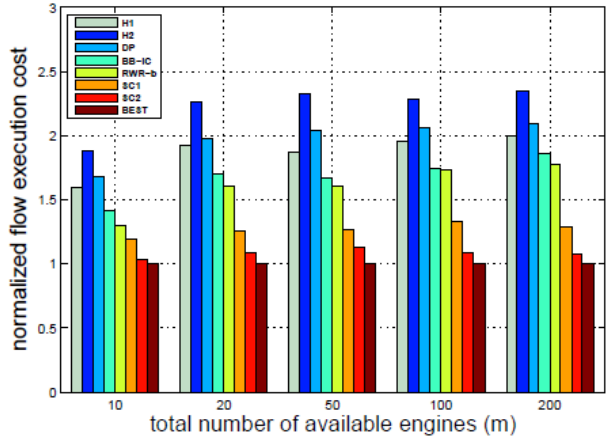
ACCURATE STATISTICS								
flow/alg	*H1*	*H2*	*DP*	*BB-IC*	*RWR-b*	*SC1*	*SC2*	*BEST*
Montage	1.3355	1.4083	1.4043	1.2555	1.2362	1.1578	1.0815	**1**
Epigenomics	1.5147	1.0282	**0.3208**	1.0057	1.0118	1.3652	1.1720	1
LIGO	1.3559	1.0512	**0.7601**	1.0245	1.0358	1.2604	1.0843	1
CyberShake	1.2858	1.1806	**0.9751**	1.1267	1.1304	1.1577	1.0489	1

Experimental Evaluation (2/3)

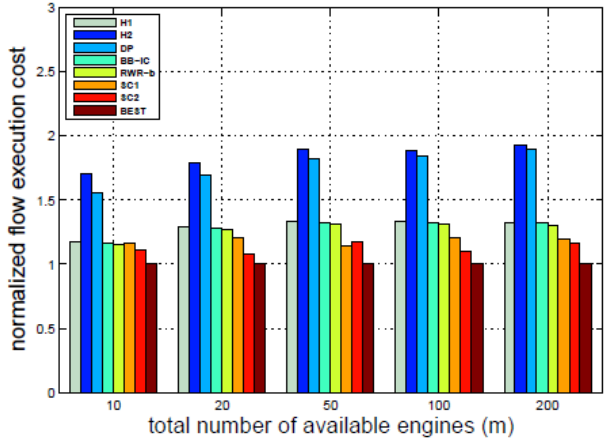
n=10



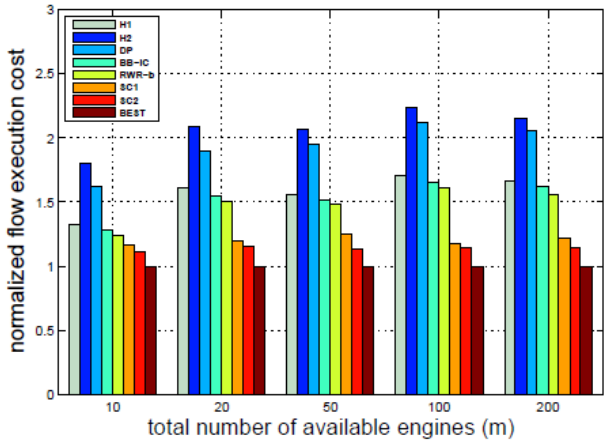
n=20



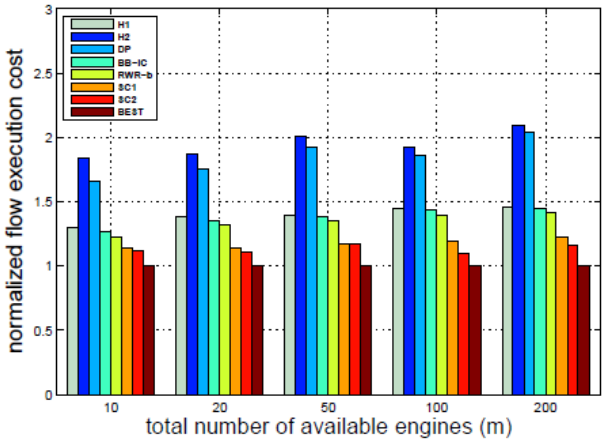
n=200



n=50



n=100



Dense flows

n=50: BEST is 70% better
n=100: BEST is 45% better
n=200: BEST is 33% better

Execution Engine Configuration (1/3)

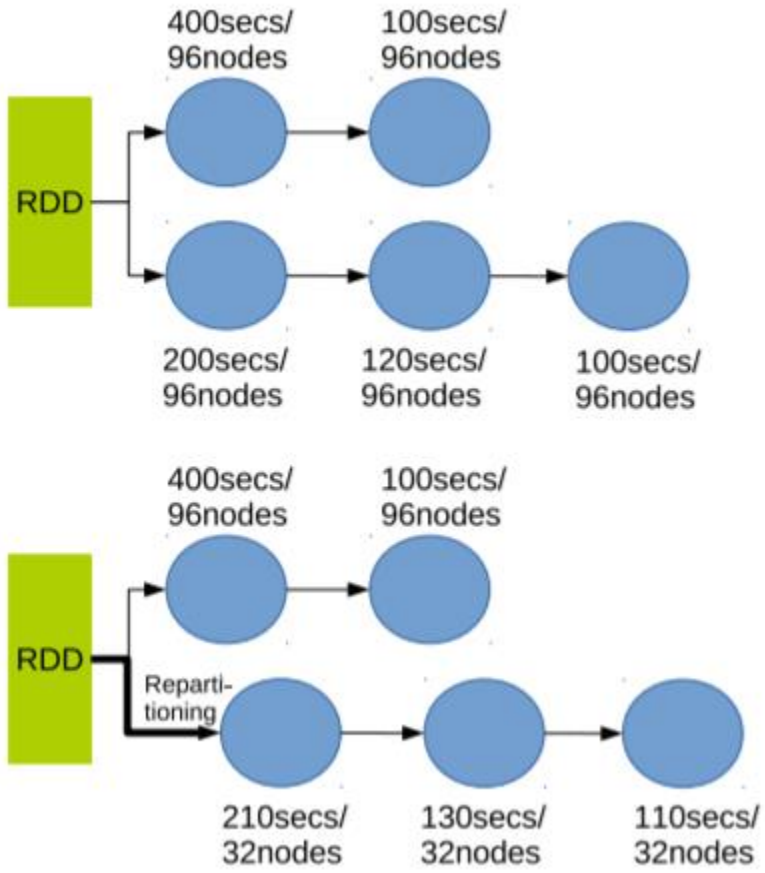
- Running a Spark application on all the available processors:
  - ➢does not necessarily imply lower running time
  - ➢may entail waste of resources
  - ➢leads to lower performance due to sub-optimal partitioning

- Propose novel algorithms for configuring dynamic partitioning

- ***Optimization aim***: minimize the resource consumption under the constraint that running time does not increase more than a user-defined threshold

Motivation Example

***rc* =** 920 × 96 = 88320



***rc* =** 500 × 96 + 450 × 32 = **62400**

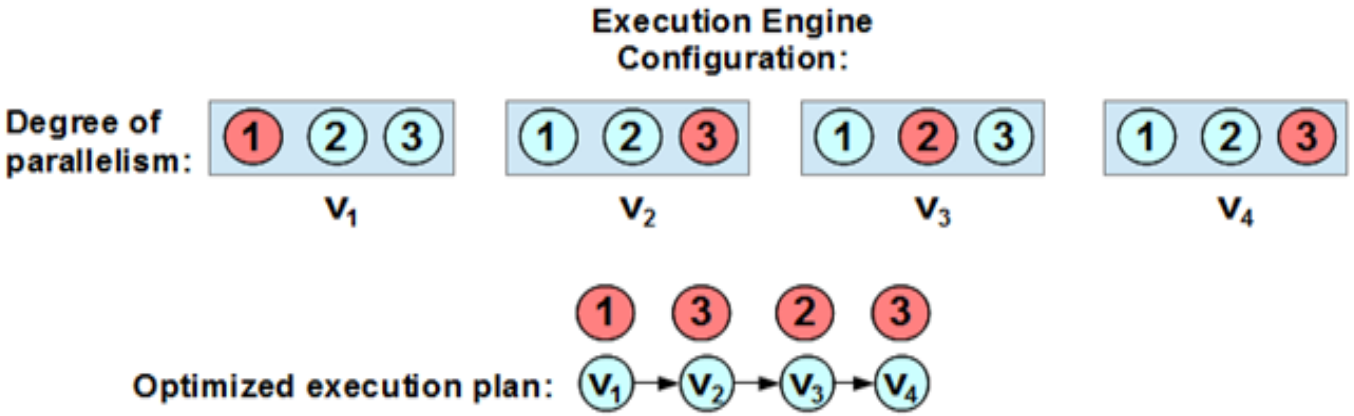
- RDD initially partitioned across 96 nodes
- The degree of partitioning/parallelism (DoP) in each stage group remains the same.
- Fixed DoP at 96 nodes → occupied for 920 secs
- After repartitioning → 950 secs

!!!BUT for 450 secs only 32 machines occupied

- ***rc* drops by 29.9%**

Execution Engine Configuration (2/3)

- Select the optimal degree of parallelism for each task
  - Considering both the resource consumption and running time
- Massive Parallel Execution Environments, such as SPARK
- Degree of Parallelism → execution engine
- Modifying the degree of parallelism → extra cost



Execution Engine Configuration (3/3)

- Find the optimal degree of parallelism in linear SPARK flows
- Optimization Objective: Minimization of response time resource consumption
- *DP*:
  - 5% response time improvement compared to the response time has degree of parallelism one
  - 25% resource consumption minimization

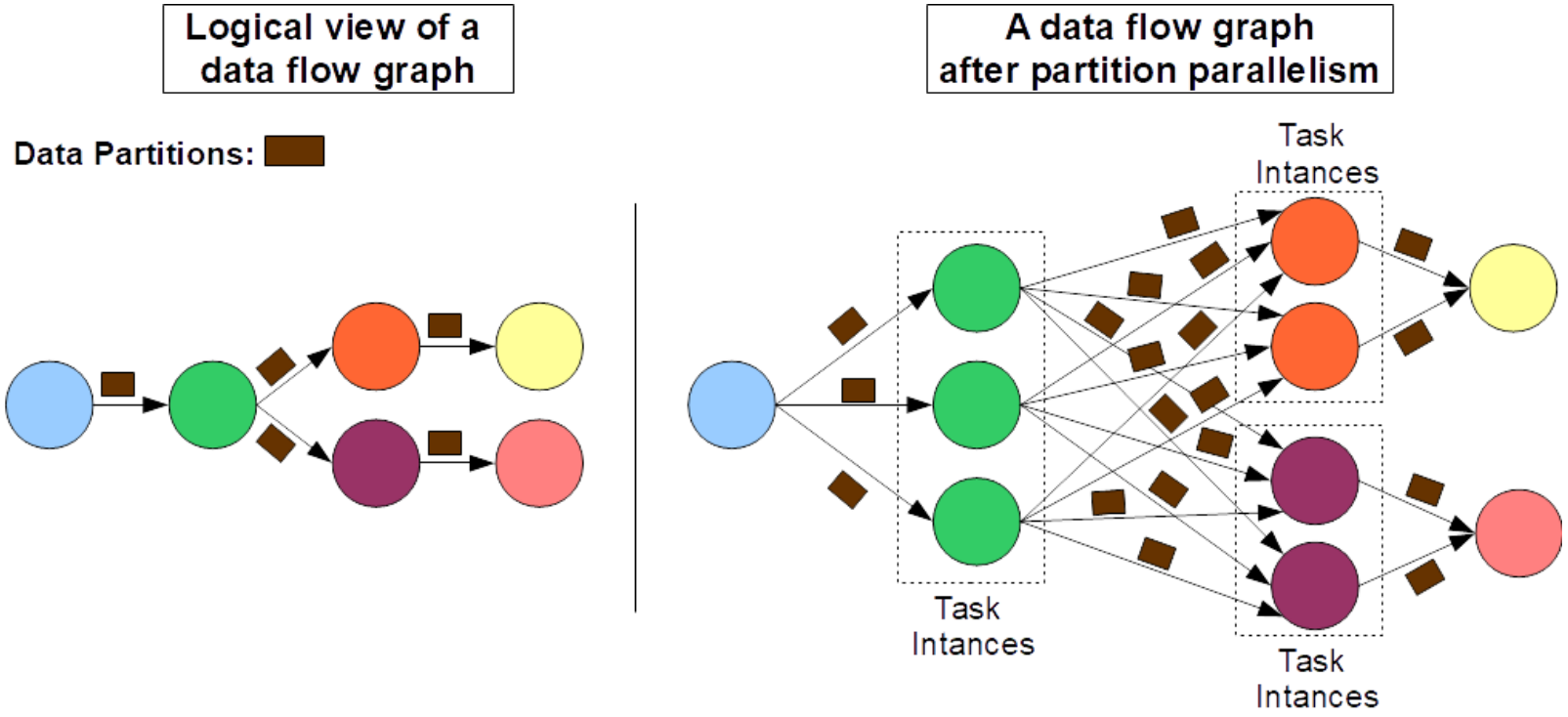
Summary

- *DP* outweighs for flows with linear flow subsets.
- *DP* is optimal for linear flows
- Optimization of data flows with very high number of tasks
- Cost improvement up to 7 times for linear (*DP*), 4.68 times for dense and 3.5 for sparse (*BEST*) flows.
- Optimization in less than 1 second.

Data Flow Execution (1/2)

- Many tasks are executed in parallel
  - ➢Employ the three main forms of parallelism:
    1. Partitioned
    2. Pipelined
    3. Independent
  - ➢Time overlapping and interplay between the constituent tasks in a flow.
  - ➢Execution time overlaps → Not all the task executions contribute to the overall running time
- Multiple tasks share the same computation resources
- Concurrent execution of the tasks using same resource pool has impact on the task execution cost.
- Existing cost models may not capture the real execution time.

Data Flow Execution (2/2)



Existing Cost Models

- Up to date no cost model estimates the response time (wallclock time) accurately in modern systems.
- Cost-models are important: they are embedded in cost-based optimizers.
- Optimization solutions employ simple cost models, e.g. cost-based task ordering. I.e., bad models → bad optimizations
- Rely on the metadata of each task
- Example of inadequacy:
  - When summing the task costs, the cost model considers the flow resource consumption, which is different from flow running time

- Main observations:
  - Execution cost computation **deviates** from real execution time
  - Optimizations may not be reflected on response time.
- Need for advanced cost model!!!

Cost Models and Metrics

- Cost metrics for which good optimization solutions already exist:

❑**Sum Cost Metric of Full plan (SCM-F)**

❑**Sum Cost Metric of Critical Path (SCM-CP)**

❑**Bottleneck**

❑Throughput (cannot capture response time)

- How these metrics can define response time?

❑**SCM-F**→ tasks of a flow executed sequentially or when tasks are pipelined **but** on the same processor

❑**SCM-CP**→ flow branches executed independently and tasks of each branch executed sequentially

❑**Bottleneck**→ tasks executed in a pipelined manner on different processors

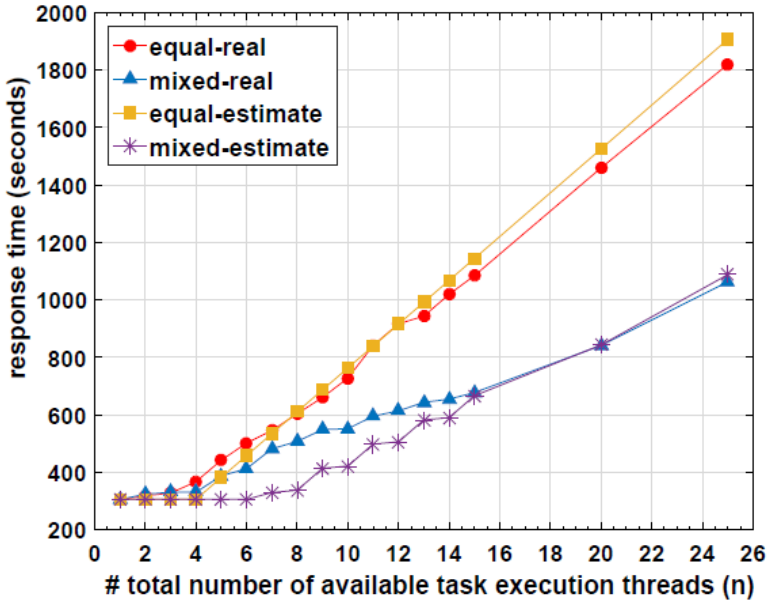
Why do we need a new cost model?

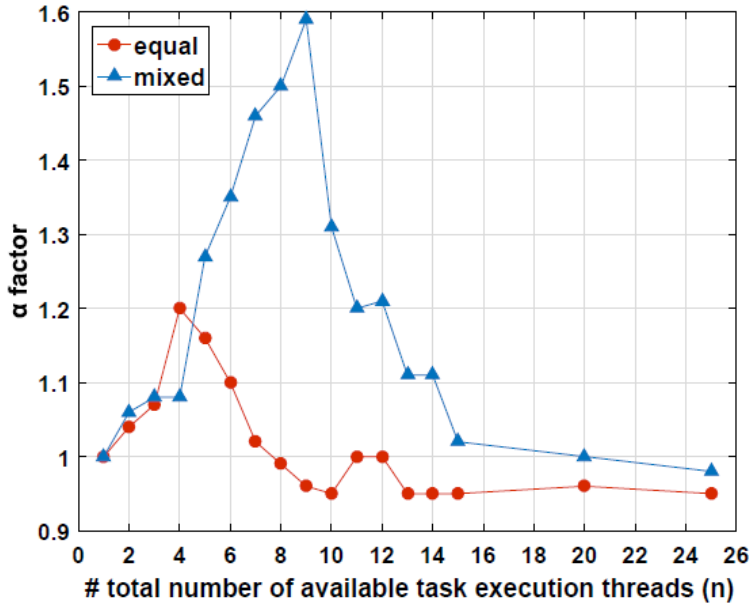
- Modern platforms (e.g. Kettle, SPARK) employ pipelined parallelism on multiple processors.
  - **SCM-F** and **SCM-CP** cannot capture response time
  - **Bottleneck** is not appropriate→ pipelined tasks are executed on the same processor + blocking tasks (e.g. sort operators)

- A new cost model that considers:
  - all the types of parallelism
  - the corresponding overlaps during task execution
  - the resource contention and benefits due to sharing resources

Response Time Modeling

- Extra execution cost because of multithreading
- Multithreading overhead factors:
  - context switching between threads (concurrent execution)
  - locks for restricting the access of tasks to a specific engine or core
  - **contention** for using shared resources (α factor)





Up to 60% deviation of response time estimation, if we won't consider (through a variable) the above factors.
=> Considering time overlaps is not enough!

What is Business Intelligence?

- The process of transforming the business data into ***knowledge*** using techniques enabling the users to take effective fact based decisions.

Information/Knowledge

Business Intelligence

- Businesses need to manage data assets, as they represent the physical world.
- BI can manage enormous structured/unstructured data to identify or create business opportunities.
- BI is characterized by three aspects:
  i. Precise and concise interpretation of large volume data
  ii. Identifying new opportunities
  iii. Implementing an effective solution to be competitive.

Why do we need BI?

- Nowadays global competition is growing and businesses need to answer challenging questions:
  - ✓How can I improve my business performance?
  - ✓How can I ensure Quality of Service?
  - ✓What would be my insightful decision based on ocean of data?
  - ✓How quick can I take decisions based on that enormous data? (End-to-End BI Solution)
  - ✓How can I integrate heterogeneous data and import to a common framework to analyze it further?

Pentaho Data Integration

- PDI (Community Edition) began as an open source project, named KETTLE (Kettle Extraction Transformation Transport Load Environment).
- PDI Architecture → Spoon
- Spoon is the graphical design interface for designing ETL jobs and transformations.
- Create complex ETL jobs without writing much of code.
- Edit, execute or debug a transformation or job.
- Set up remote PDI servers to coordinate jobs across a collection of clustered machines and execute jobs within a cluster of Carte cluster nodes.

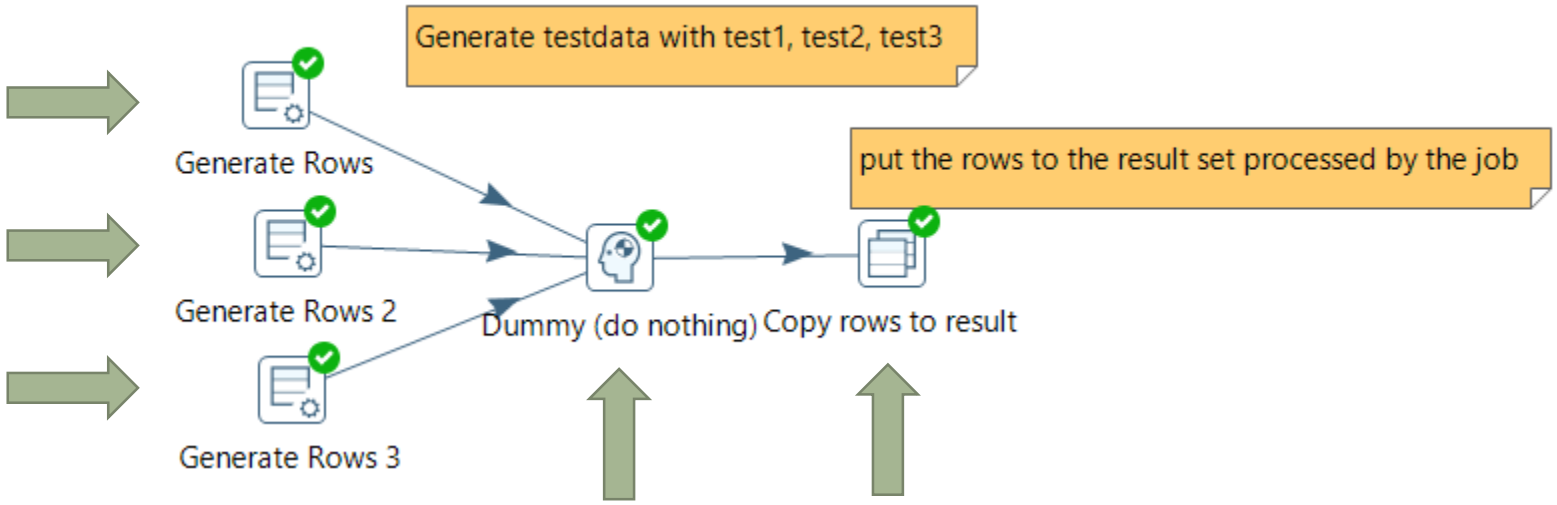
Jobs vs Transformations

- Transformations:
  - ✓ moving and transforming rows from source to target
  - ✓Steps can be executed in parallel

- Jobs:
  - ✓ consider the high level flow control: executing transformations, sending mails on failure, etc.
  - ✓Steps executed only in order (single-thread execution)

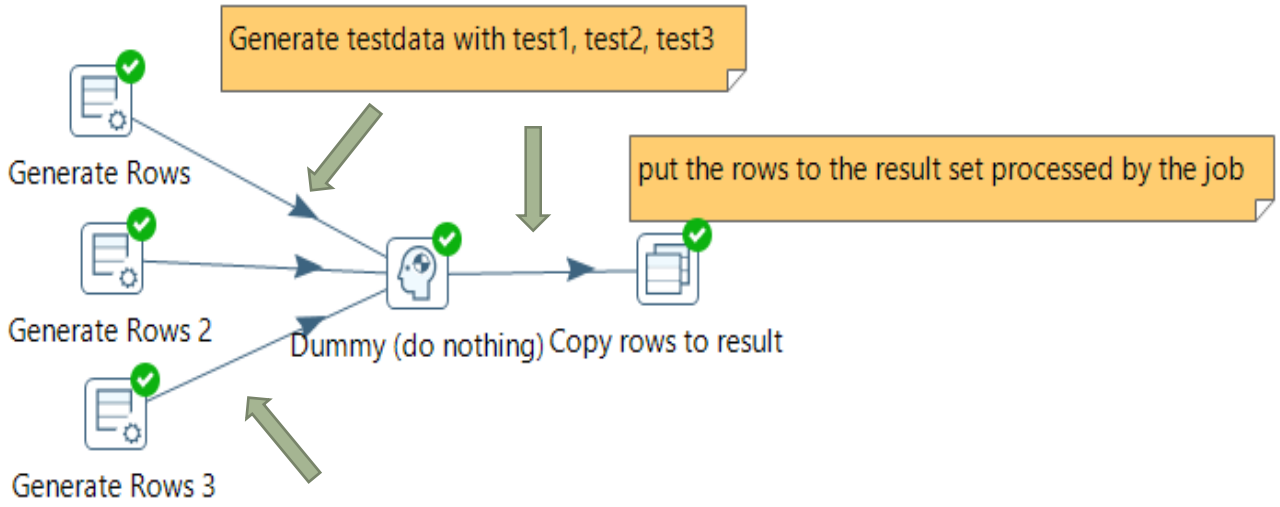
Steps

- Building blocks of a transformation.
- Example: A text file input or output.
- Designed to perform a specific task, such as reading data, filtering rows and logging to a database.
- Configured to perform the tasks we aim to design.



Hops

- Connect the steps and allow schema metadata to pass from one step to another.
- Do not necessarily express a sequential execution.
- Determine the flow of the data through the steps, not necessarily the sequence in which they are executed.
- When a transformation is running, each step instantiates its own thread, pushes and passes data.
- All steps can be executed in parallel, so the initialization is not predictable.



Pentaho Examples (demo)

- How can we create a transformation/job?
- Add a sequence – Basic example
- Twitter – Product campaign

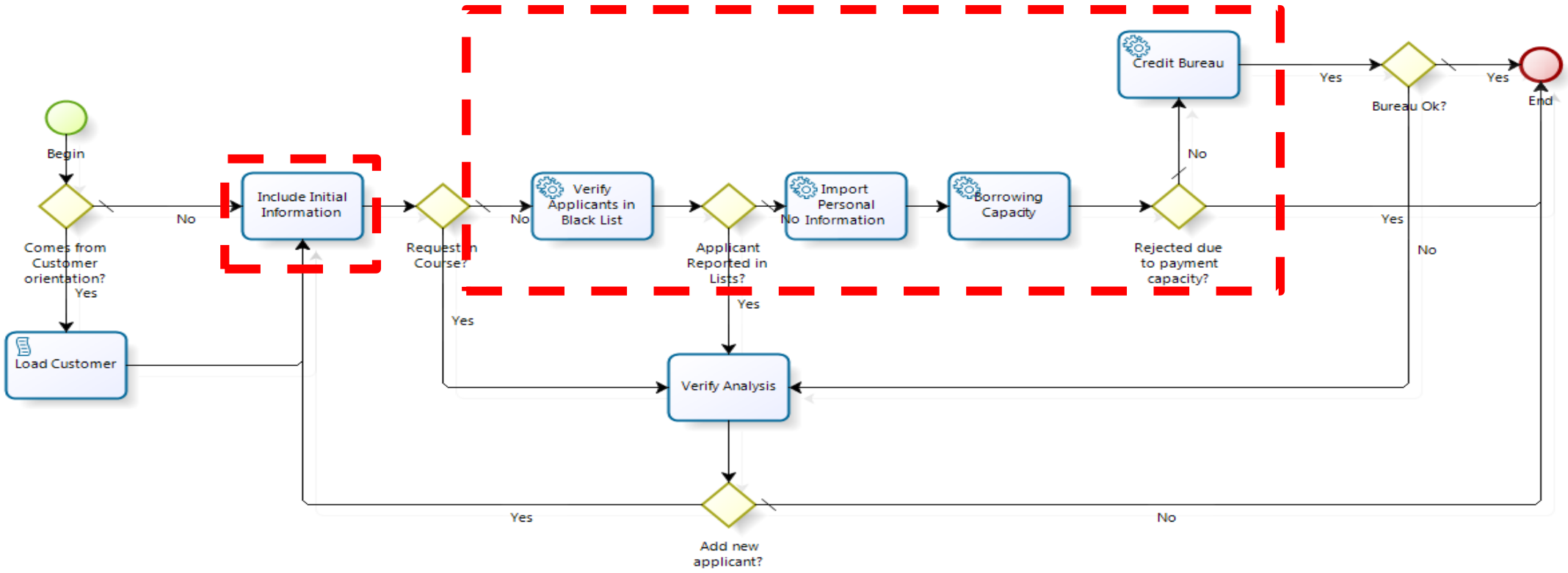
Business Process Optimization

- Borrow lessons from the data management community for ***Business Process Optimization*** (BPO).

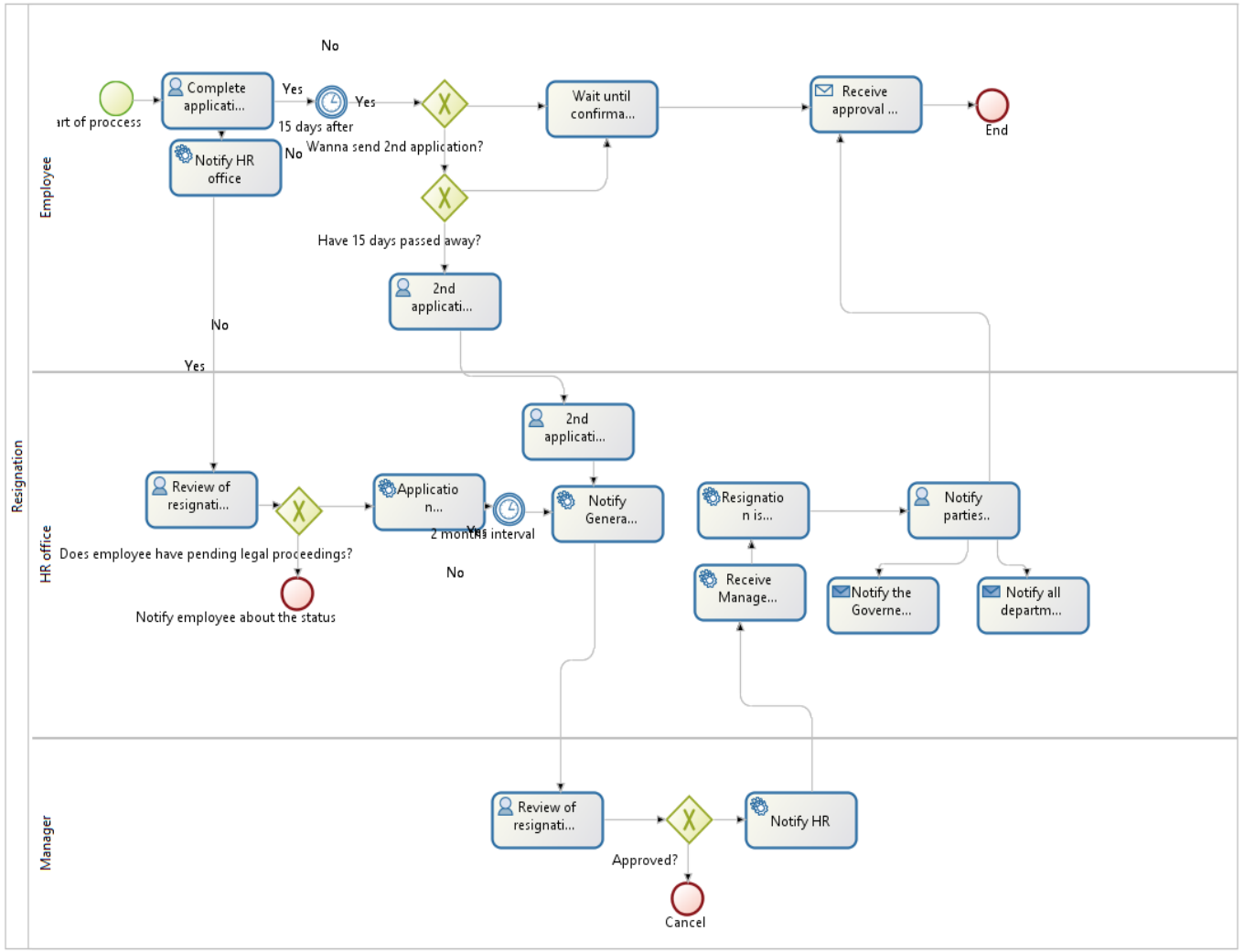
- Automated cost-based optimization of performance of BPMN processes:

1. BPMN challenge to automate production of executable workflow/process execution.
2. Less burden for workflow designers.
3. Flexibility and resilience.
4. Performance benefits.

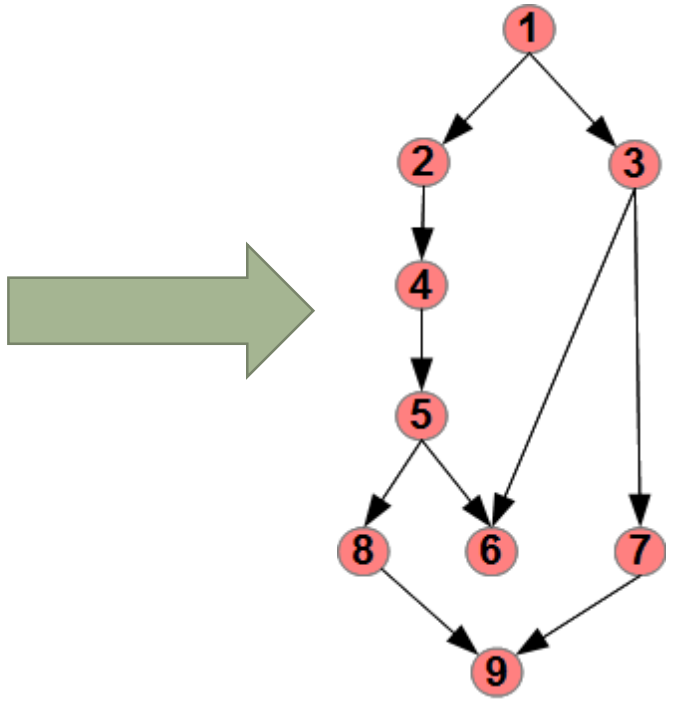
A BPMN example



Mapping BPs to DAGs



Map BPMN diagrams to DAGs amenable to cost-based optimizations.



Review of the Workflow Optimization research area

- Need for *holistic solutions*
- Common evaluation approach
- Embedding optimization algorithms in WfMS
- Only a few proposals considering the edge cost
- Optimization of multiple flows simultaneously
- Statistical metadata collection (early stage of research)
- Existing model costs fail to reflect the response time of the parallel task execution
- End-to-End Business Process Optimization Solutions

Related Work (1/2)

- Georgia Kougka, and Anastasios Gounaris: "Cost-based optimization of data flows based on task re-ordering", Transactions on Large-Scale Data- and Knowledge-Centered Systems (TLDKS), 2017.
- Georgia Kougka, and Anastasios Gounaris: " Optimal Task Ordering in Chain Data flows: Exploring the Practicality of non-Scalable Solutions'', Proceedings 19th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), 2017.
- Georgia Kougka, and Anastasios Gounaris: "Optimization of Data-intensive Flows: Is it Needed? Is it Solved?", Proceedings of the 17th International Workshop on Data Warehousing and OLAP (DOLAP), 2014.
- Georgia Kougka, and Anastasios Gounaris: " Declarative expression and optimization of data-intensive flows", Proceedings 19th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), 2013.

Related Work (2/2)

- Georgia Kougka, Anastasios Gounaris, and Kostas Tsichlas: "Practical algorithms for execution engine selection in data flows", Future Generation Computer Systems (Elsevier), 2015.
- Anastasios Gounaris, Georgia Kougka, Ruben Tous, IEEE Transactions on Parallel and Distributed Systems (TPDS), 2017.
- Georgia Kougka, Anastasios Gounaris, and Ulf Leser: ''Modeling Data Flow Execution in a Parallel Environment'', Proceedings 19th International Conference on Data Warehousing and Knowledge Discovery (DaWaK), 2017.
- Georgia Kougka, Anastasios Gounaris, and Alkis Simitsis: ''The Many Faces of Data-centric Workflow Optimization: A Survey‘’, (*to appear*).

Thank you!!!