Information (Computational) Systems in the Big-Data Era
Eindhoven University of Technology, 14 May 2012

Process Mining in Complex Domains

joint work with Antonella Guzzo and Luigi Pontieri

Gianluigi Greco
University of Calabria

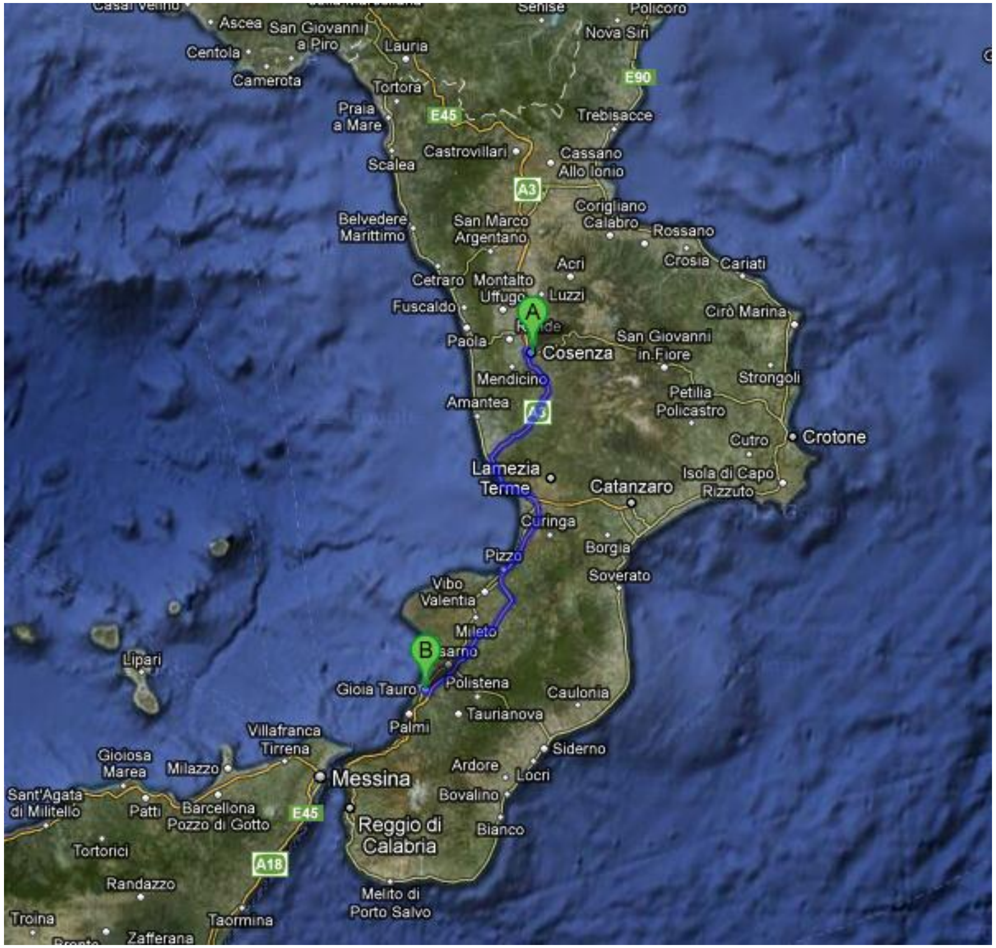
An Application Domain

An Application Domain

An Application Domain

Gioia Tauro

- Italian harbor acting as a maritime freight hub (about 4 millions of containers per year).
  - Berth planning
  - Routing
  - …
  - Yard planning



Yard Planning

- The mission is to offer high quality of service to the navigation lines, while reducing the overall cost of internal logistic processes.
- Critical performance measures are
  - *the latency time* elapsed when serving a ship (where, typically, a number of containers are both discharged off and charged on), and
  - *the overall costs* of moving the containers around the yard.
- A key factor impacting on both these measures is the number of "*house-keeping*" moves that are applied to the containers.

Yard Planning

- The mission is to offer high quality of service to the navigation lines, while reducing the overall cost of internal logistic processes.
- Critical performance measures are
  - *the latency time* elapsed when serving a ship (where, typically, a number of containers are both discharged off and charged on), and
  - *the overall costs* of moving the containers around the yard.
- A key factor impacting on both these measures is the number of "*house-keeping*" moves that are applied to the containers.

Minimize house-keeping moves

Yard Planning

- The mission is to offer high quality of service to the navigation lines, while reducing the overall cost of internal logistic processes.
- Critical performance measures are
  - *the latency time* elapsed when serving a ship (where, typically, a number of containers are both discharged off and charged on), and
  - *the overall costs* of moving the containers around the yard.
- A key factor impacting on both these measures is the number of "*house-keeping*" moves that are applied to the containers.

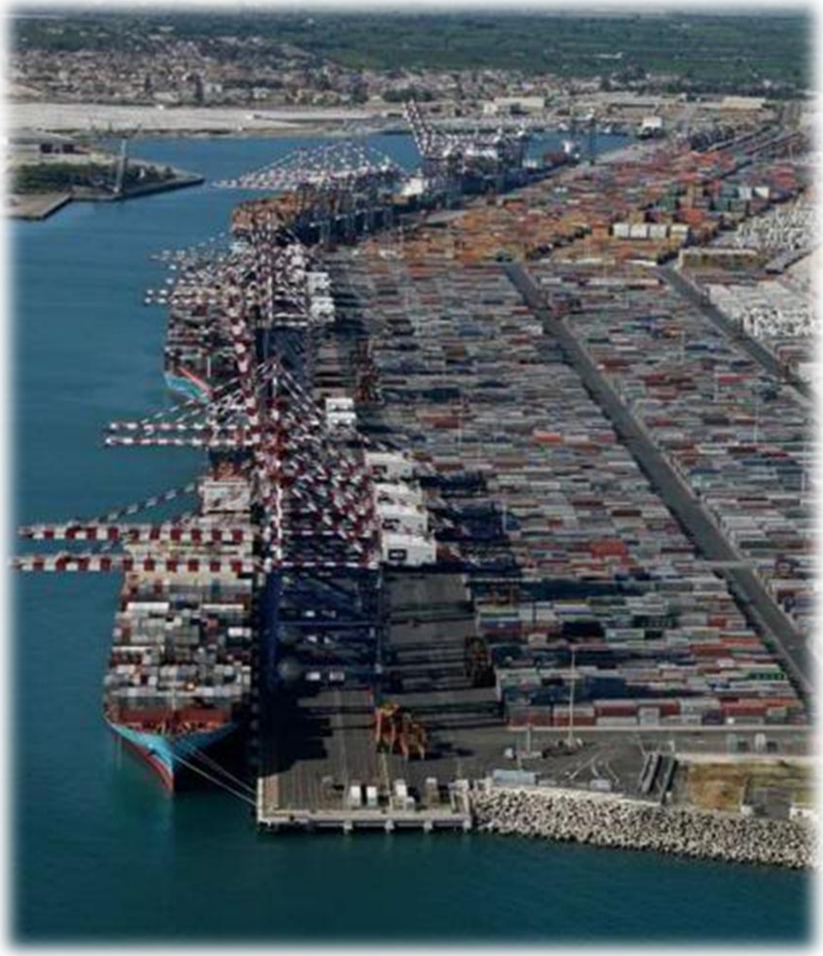
Minimize house-keeping moves

↓

Understand the process, first!

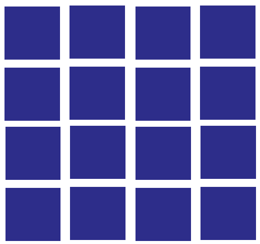
The yard

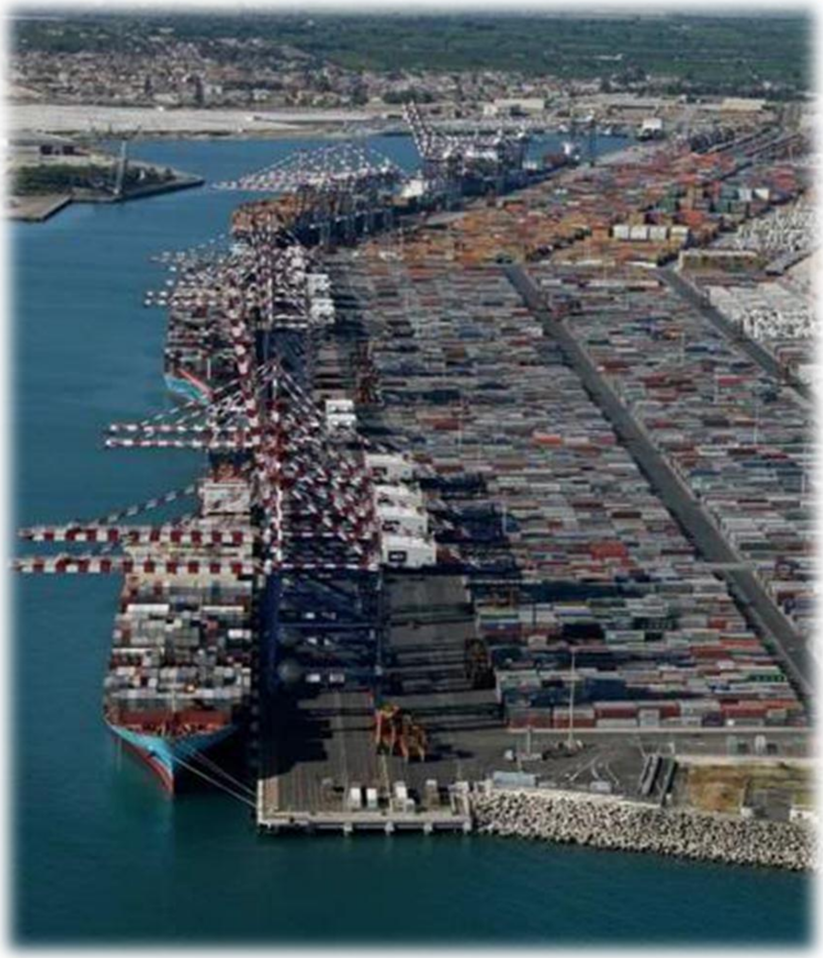
slot



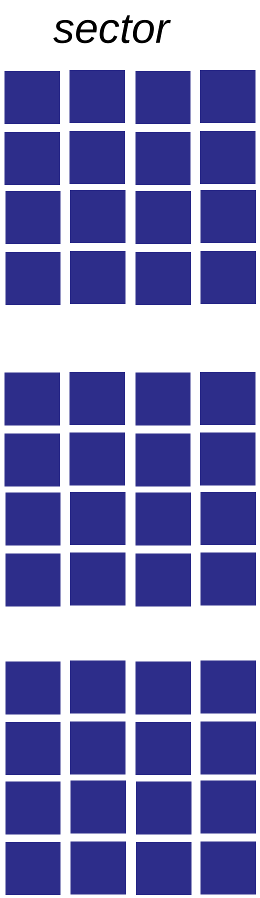
The yard

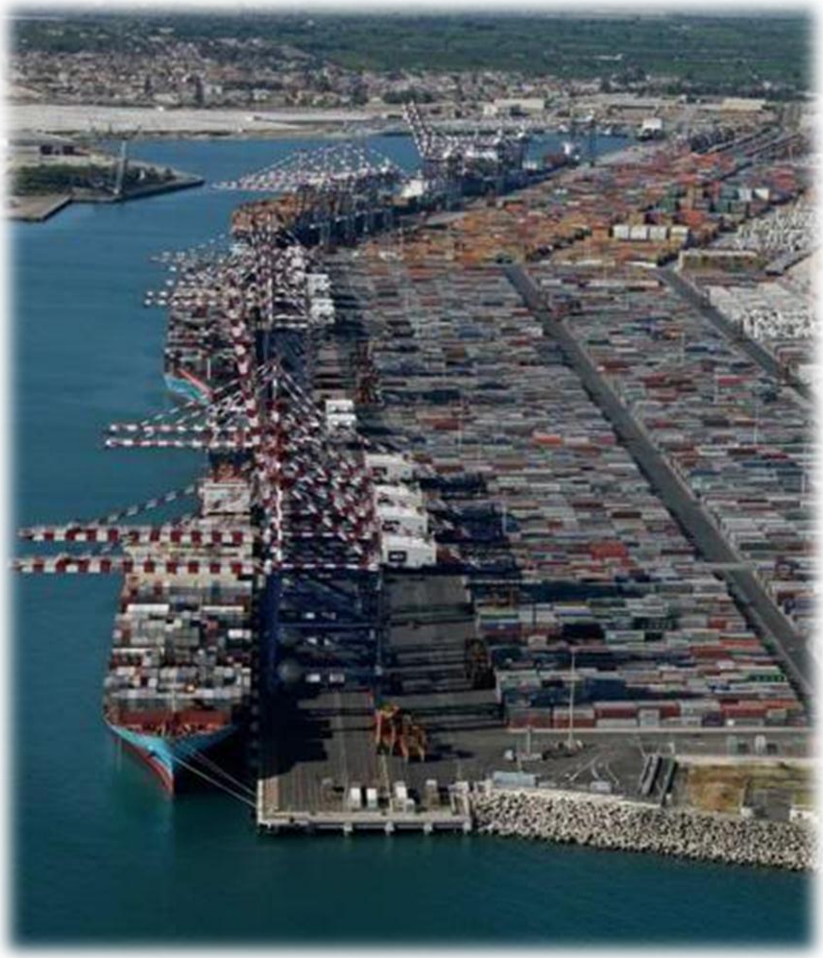
block



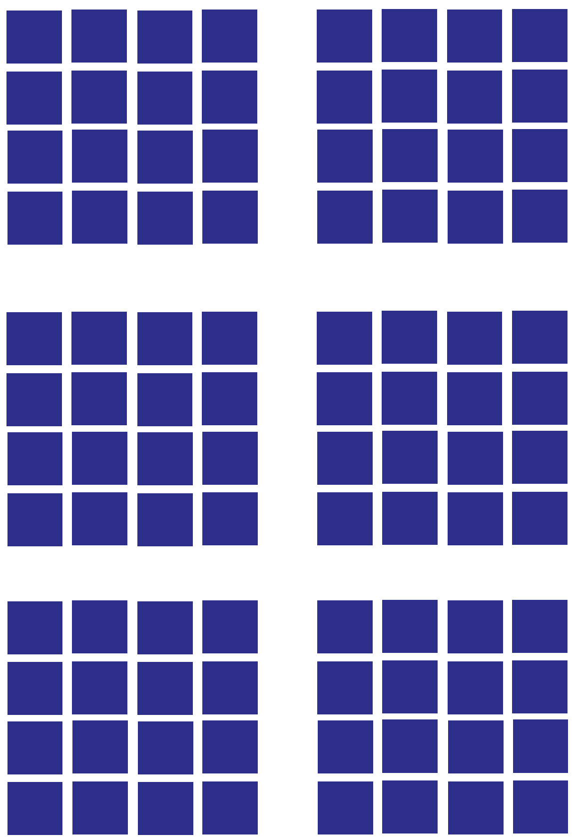


The yard





The yard

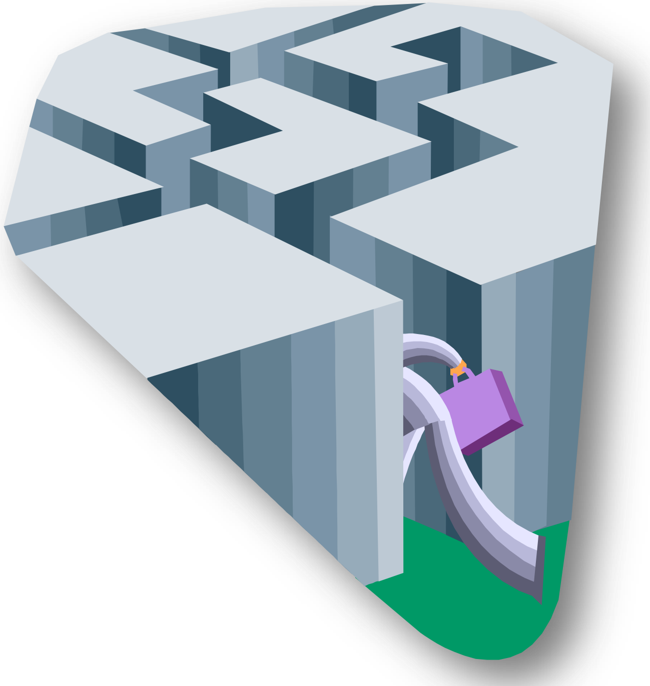


Life Cycle

- The container is initially unloaded from the ship, with the help of a crane
- It is first stocked within a zone near to the dock
- It is carried to some slot of the yard, via
  - cranes
  - straddle-carriers (a vehicle capable of picking and carrying a container, by possibly lifting it up)
  - multi-trailers (a sort of train-like vehicle that can transport many containers
- At boarding time, the container is first placed in a yard area close to the dock
- Finally, it is loaded on the cargo by means of a crane

Challenges

- Logs from transactional systems
- Logs mix different usage scenarios
- Traces are stored at different level of details
- Noise
- Huge volume of data



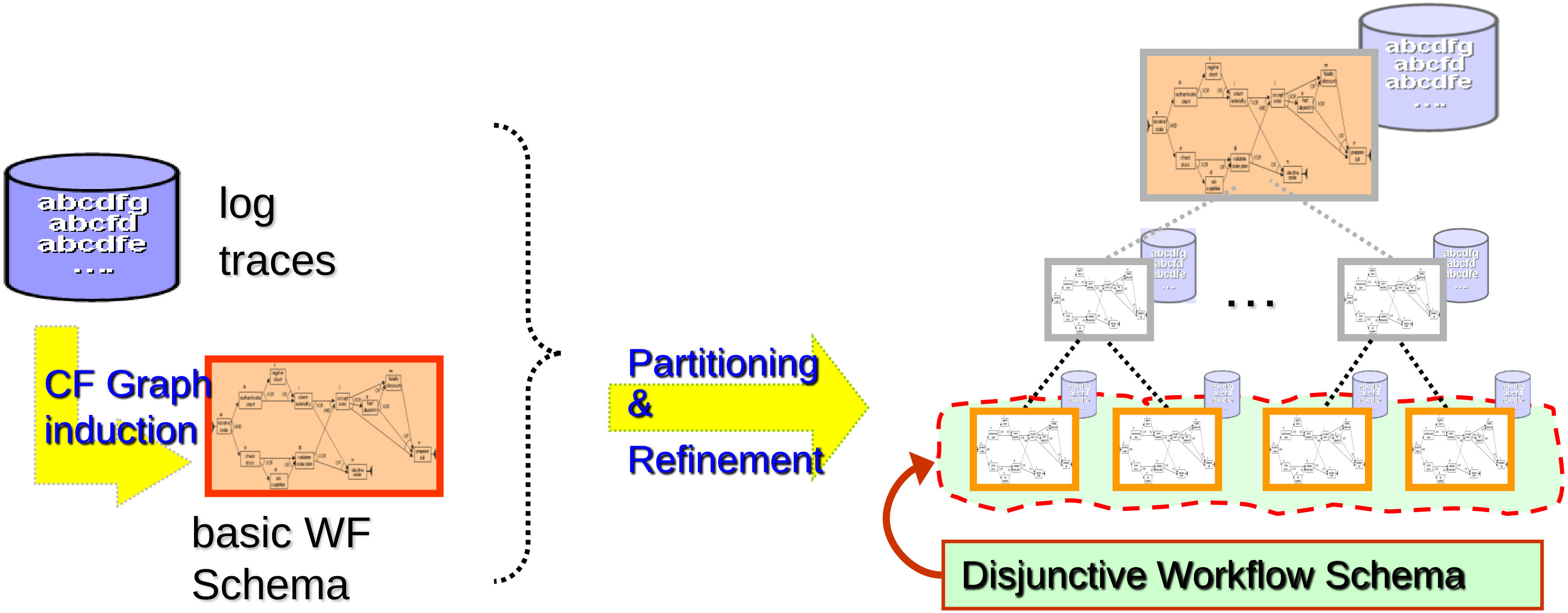
Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Framework
- Implementation Issues

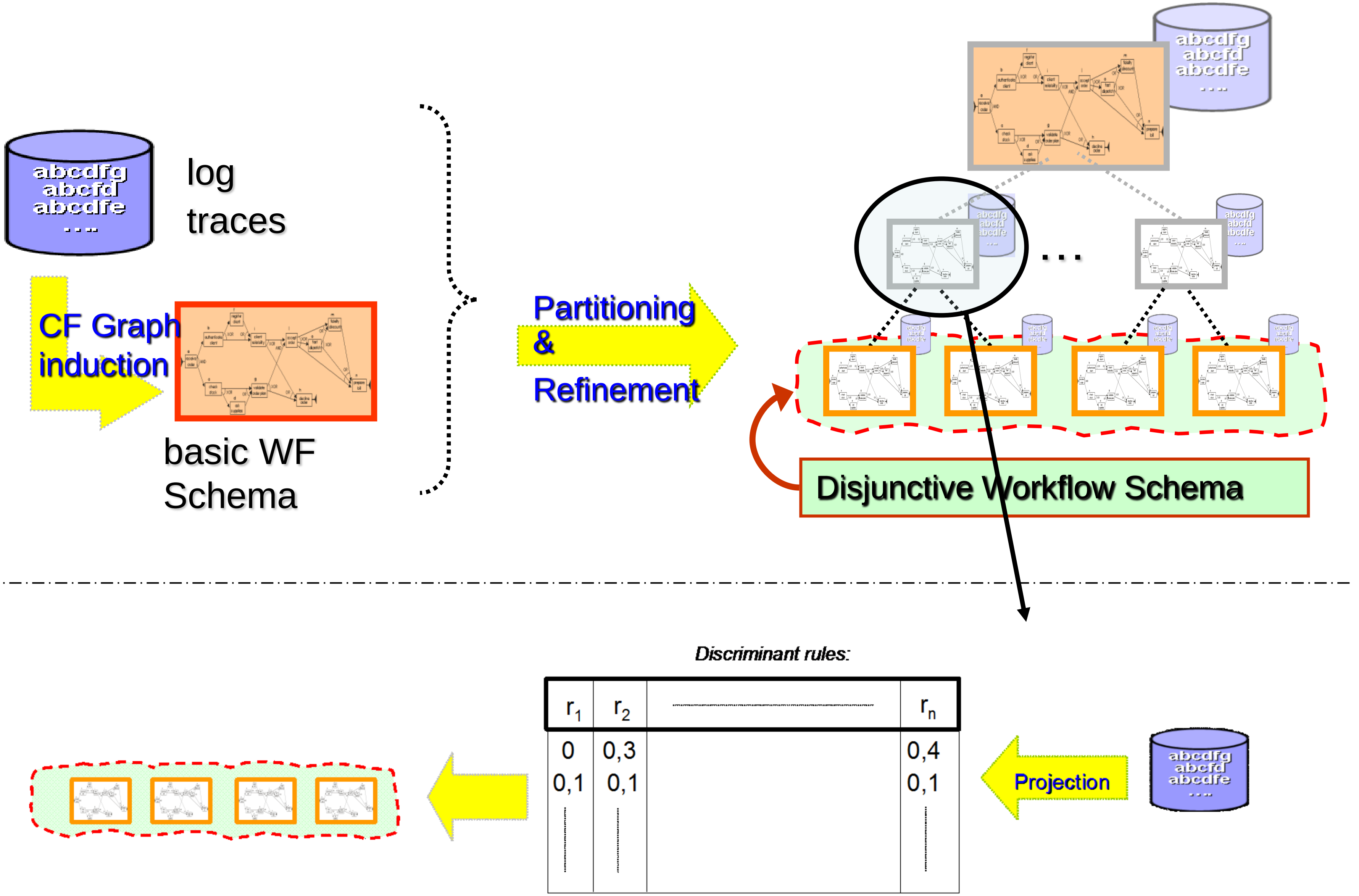
Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Framework
- Implementation Issues

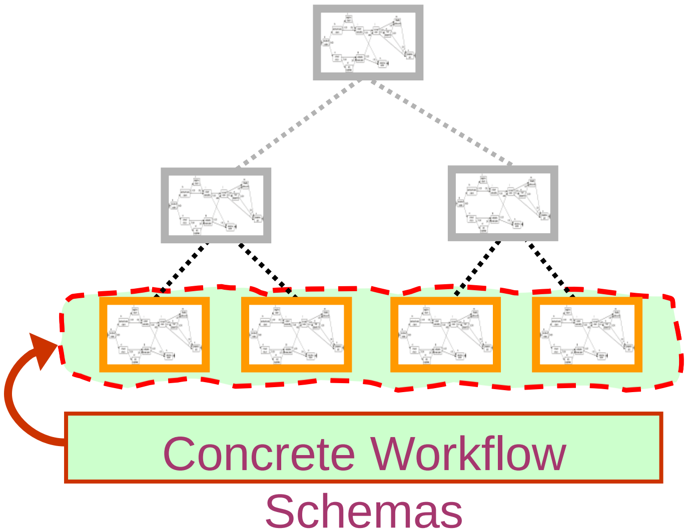
(1) Clustering



(1) Clustering



(2) Abstraction

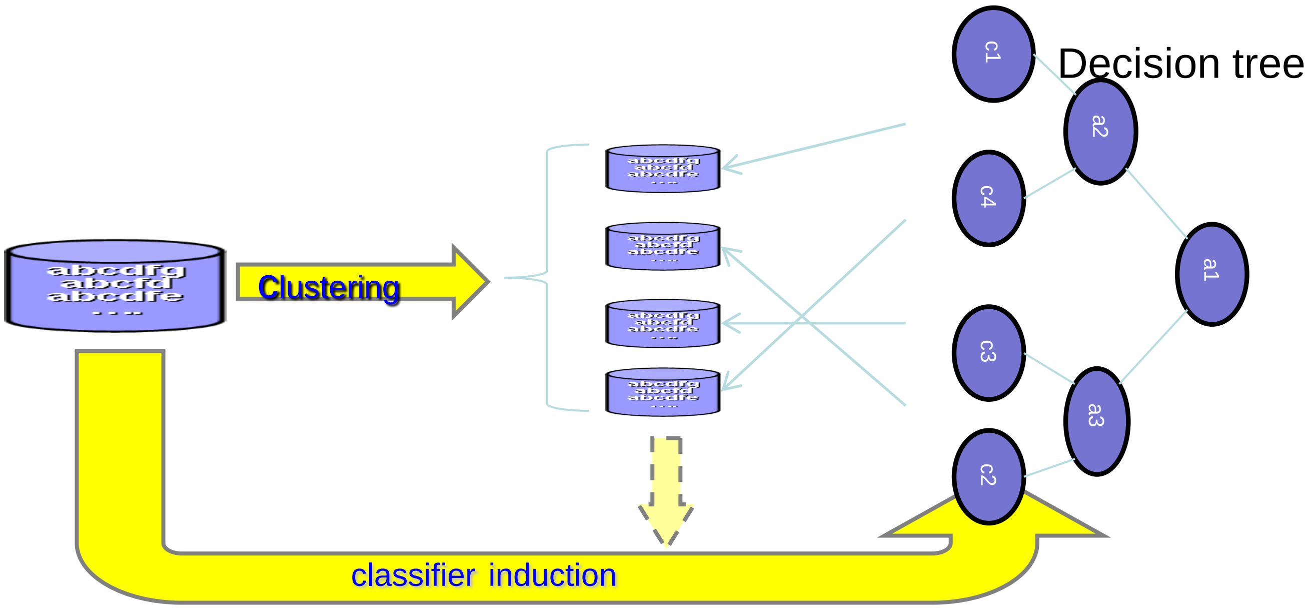


- The tree describes the process behavior at different level of details
- At the highest level of detail (leaves of the tree), the schemas could be used to support the design of concrete workflow models
- At lower levels, the schemas are abstract views over heterogeneous behaviors, which could support analysis and monitoring tasks

- Basic Idea:
  1) The hierarchy is restructured bottom-up at different levels
  2) Produce an abstraction dictionary

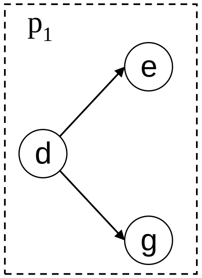
(3) Classification

- Basic idea:
  - find a comprehensive representation for the process, describing both structural and non-structural aspects
  - A rule-based classifier is induced to discriminate among given structural clusters, based on process/task data
  - help interpreting/predicting the different ways of executing the process, based on properties of process/task instances

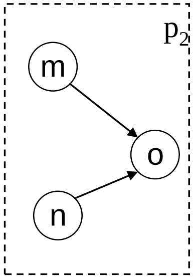


(4) Outlier Detection

- Structural patterns are identified



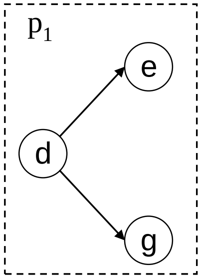
fork s-pattern



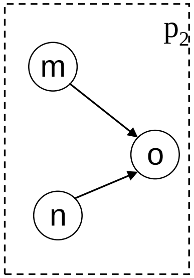
join s-pattern

(4) Outlier Detection

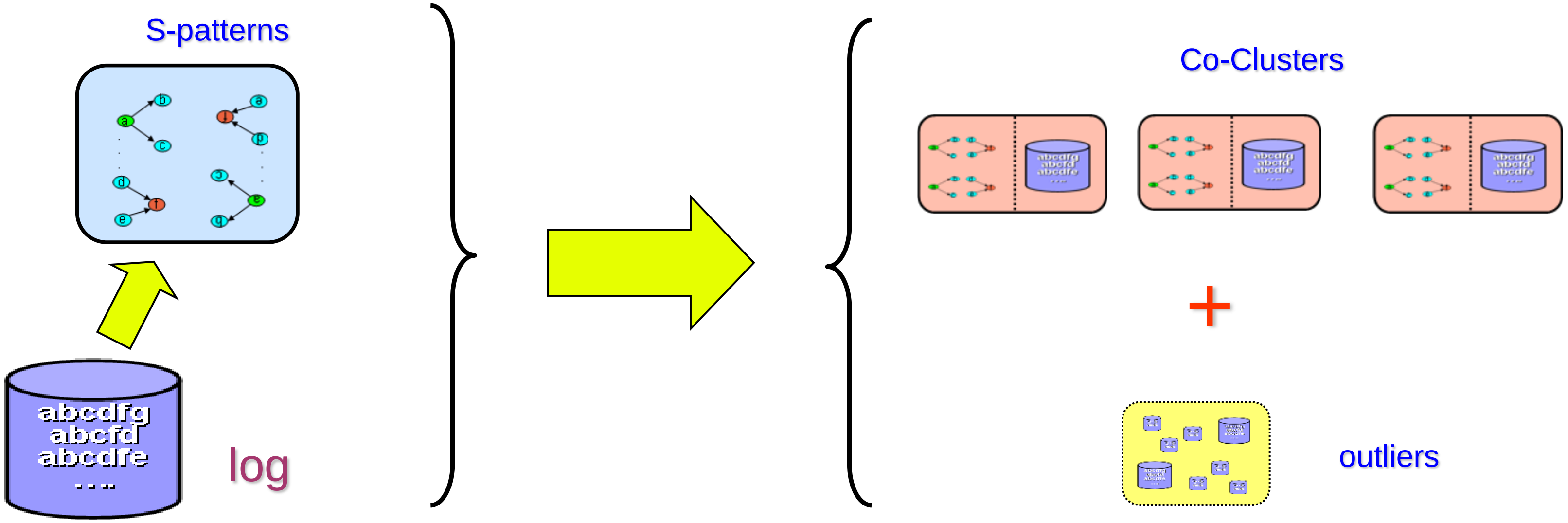
- Structural patterns are identified
- They are co-clustered with the traces, based on what an extent these latter support them
- Mark as outlier each trace *t* such that either
  - *t* has not been assigned to any cluster
  - *t* belongs to a cluster whose cardinality is “appreciably smaller” than the average cluster size



fork s-pattern



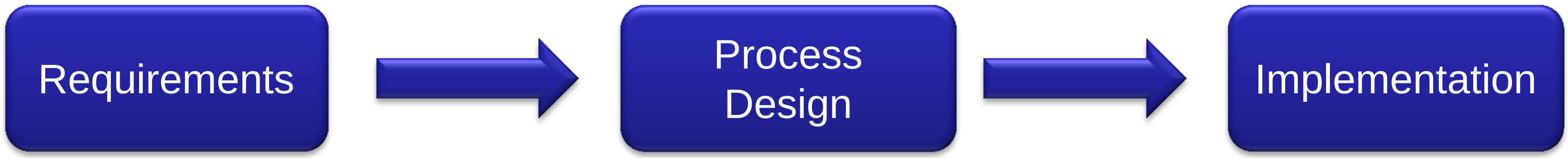
join s-pattern



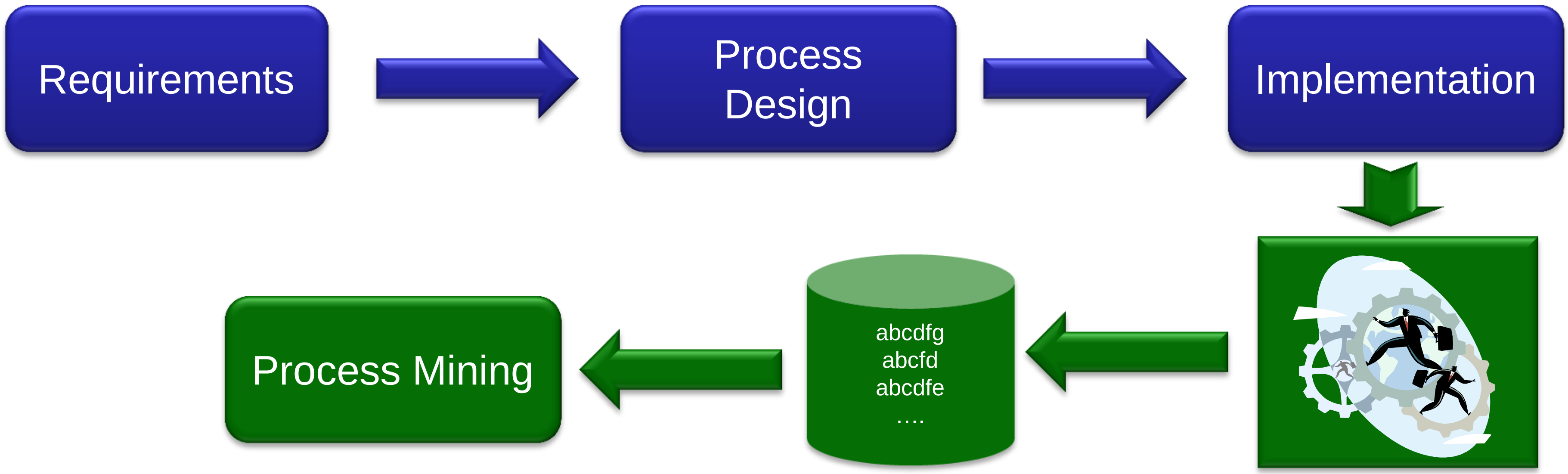
Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Framework
- Implementation Issues

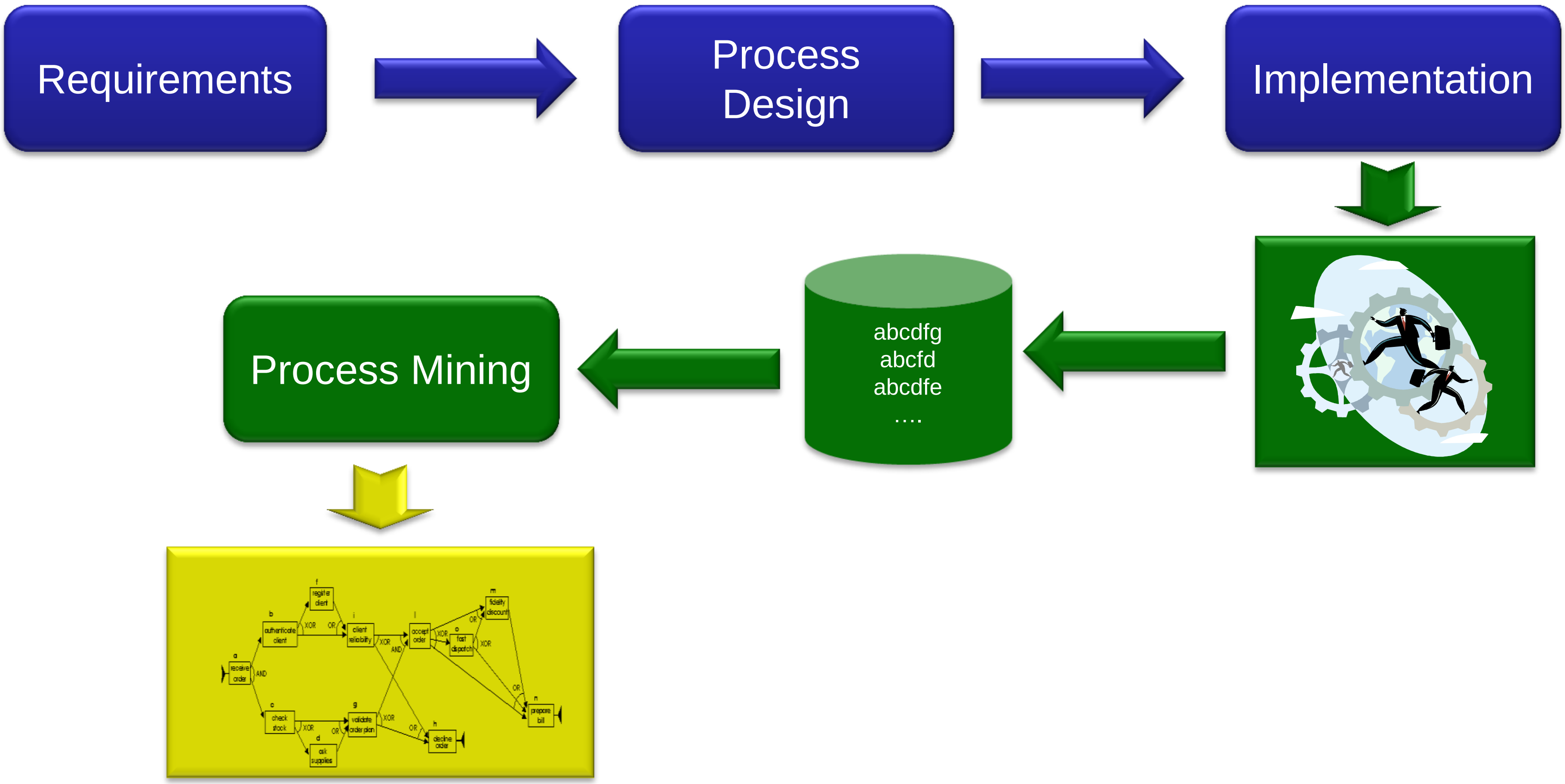
Process Mining



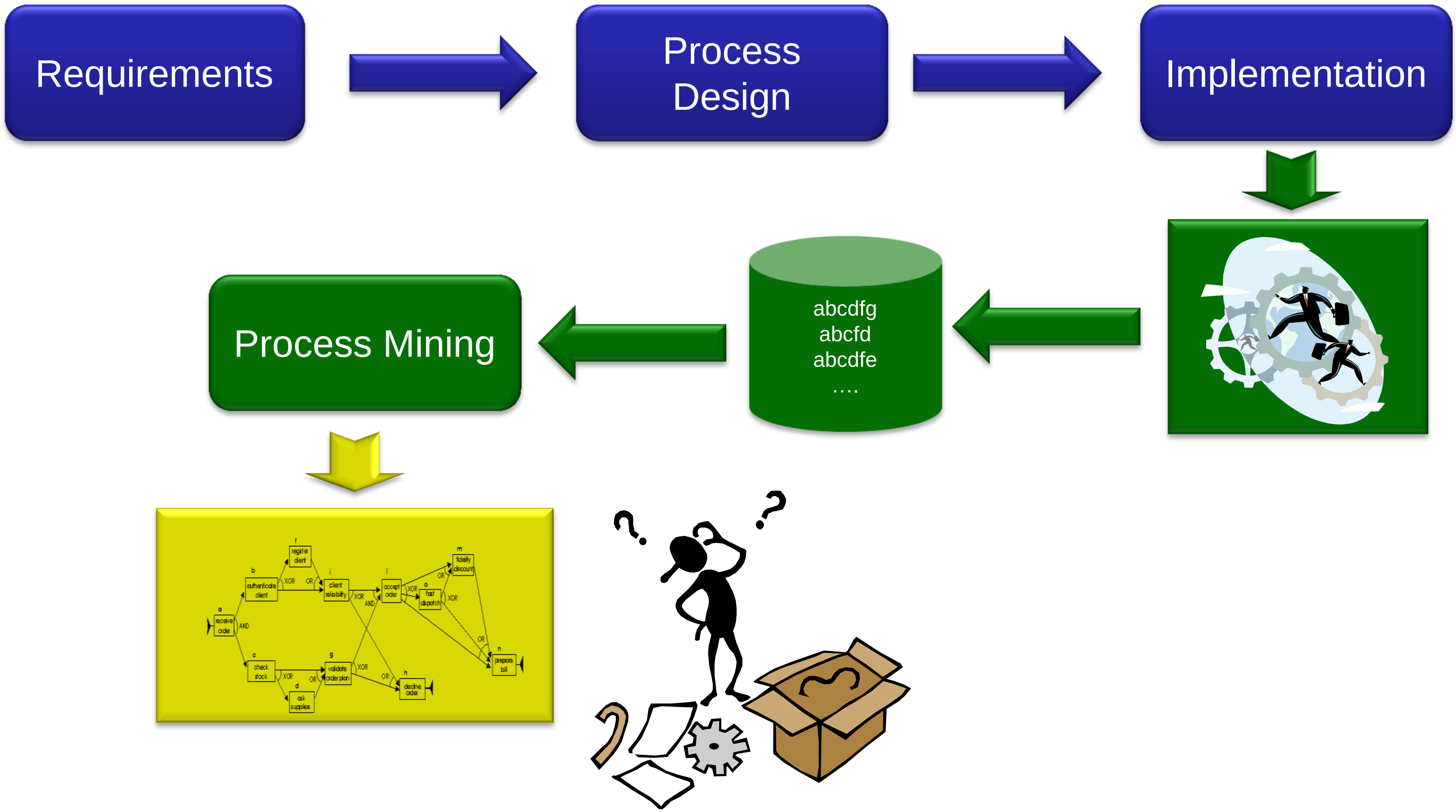
Process Mining



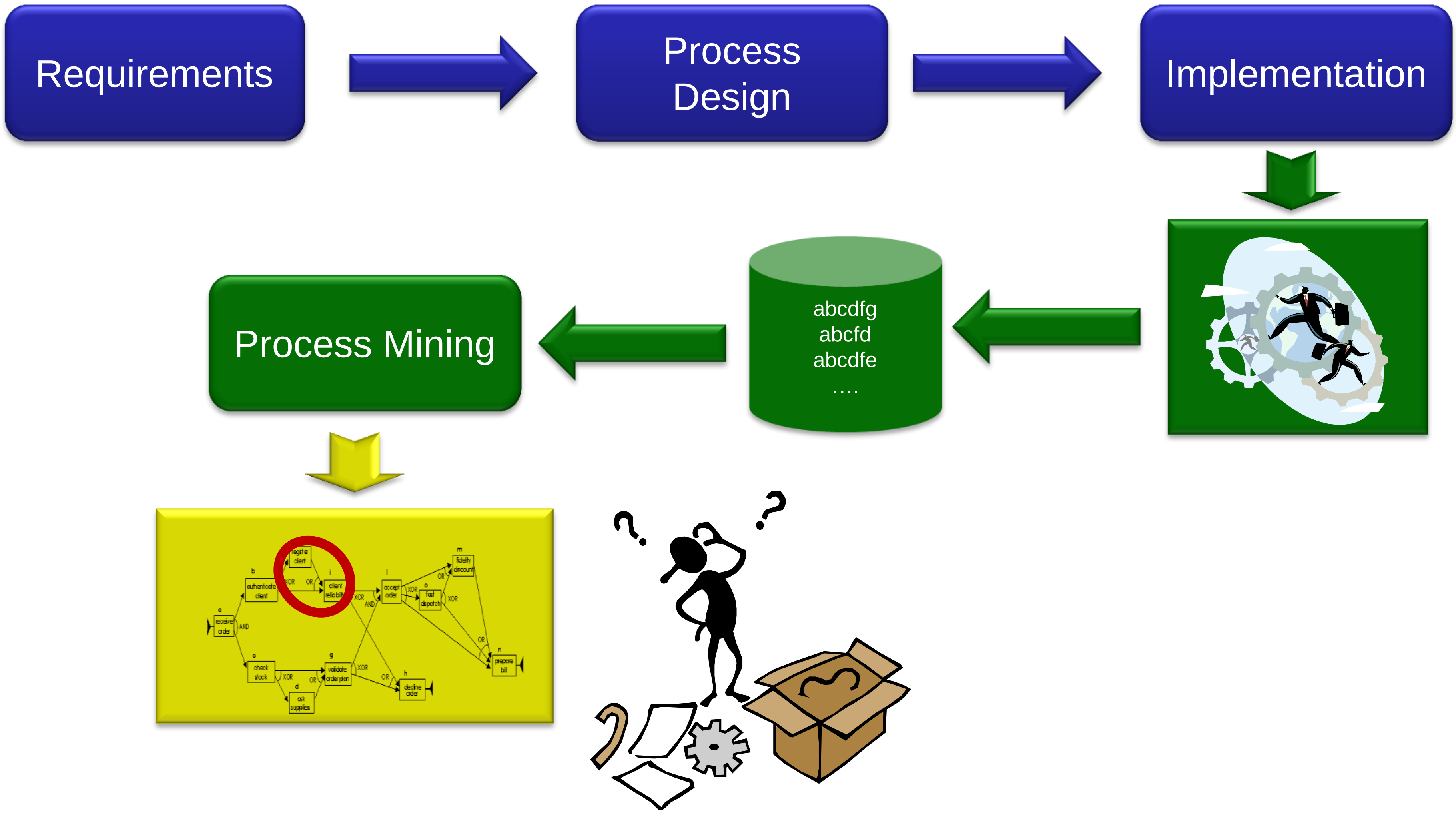
Process Mining



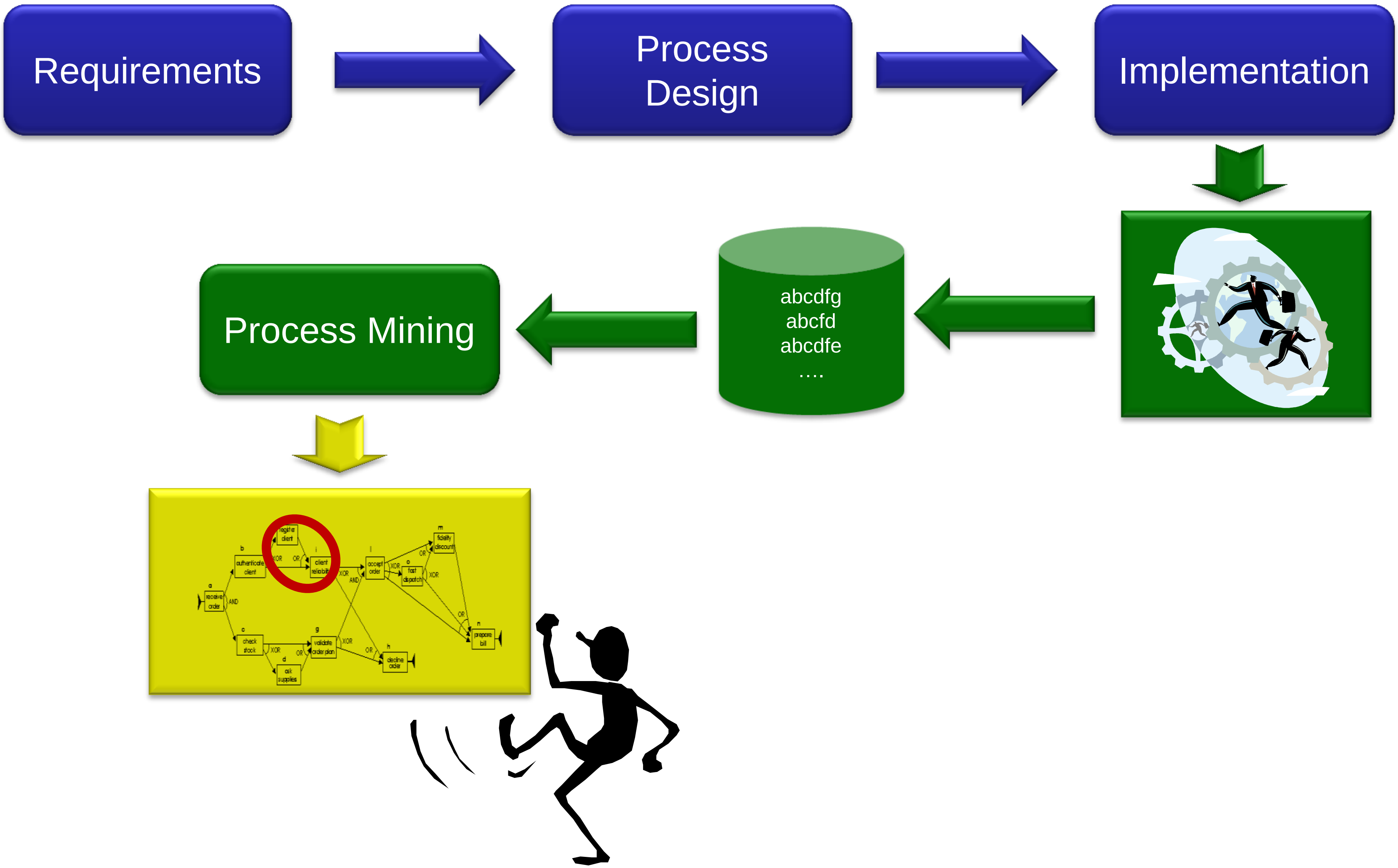
Process Mining



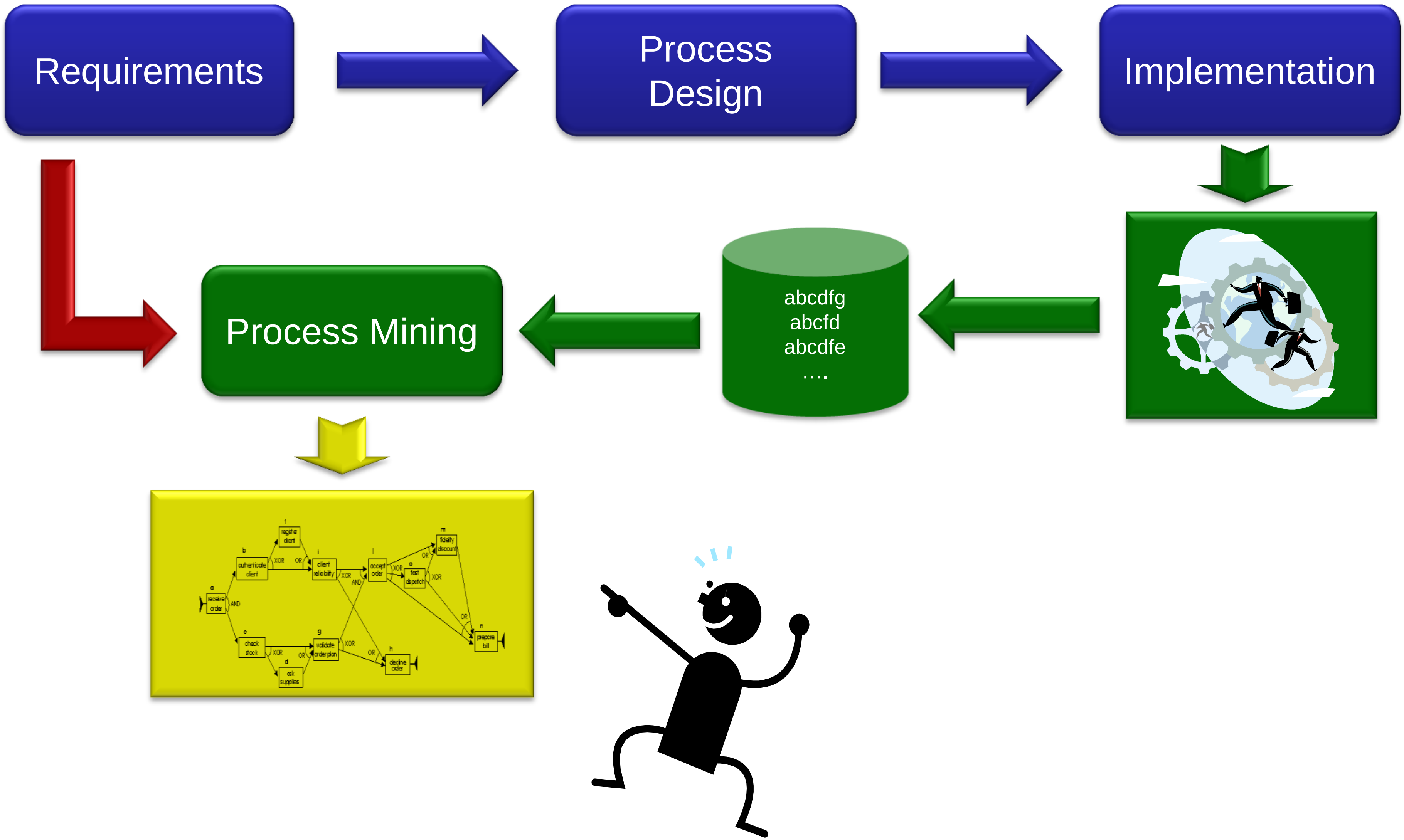
Process Mining



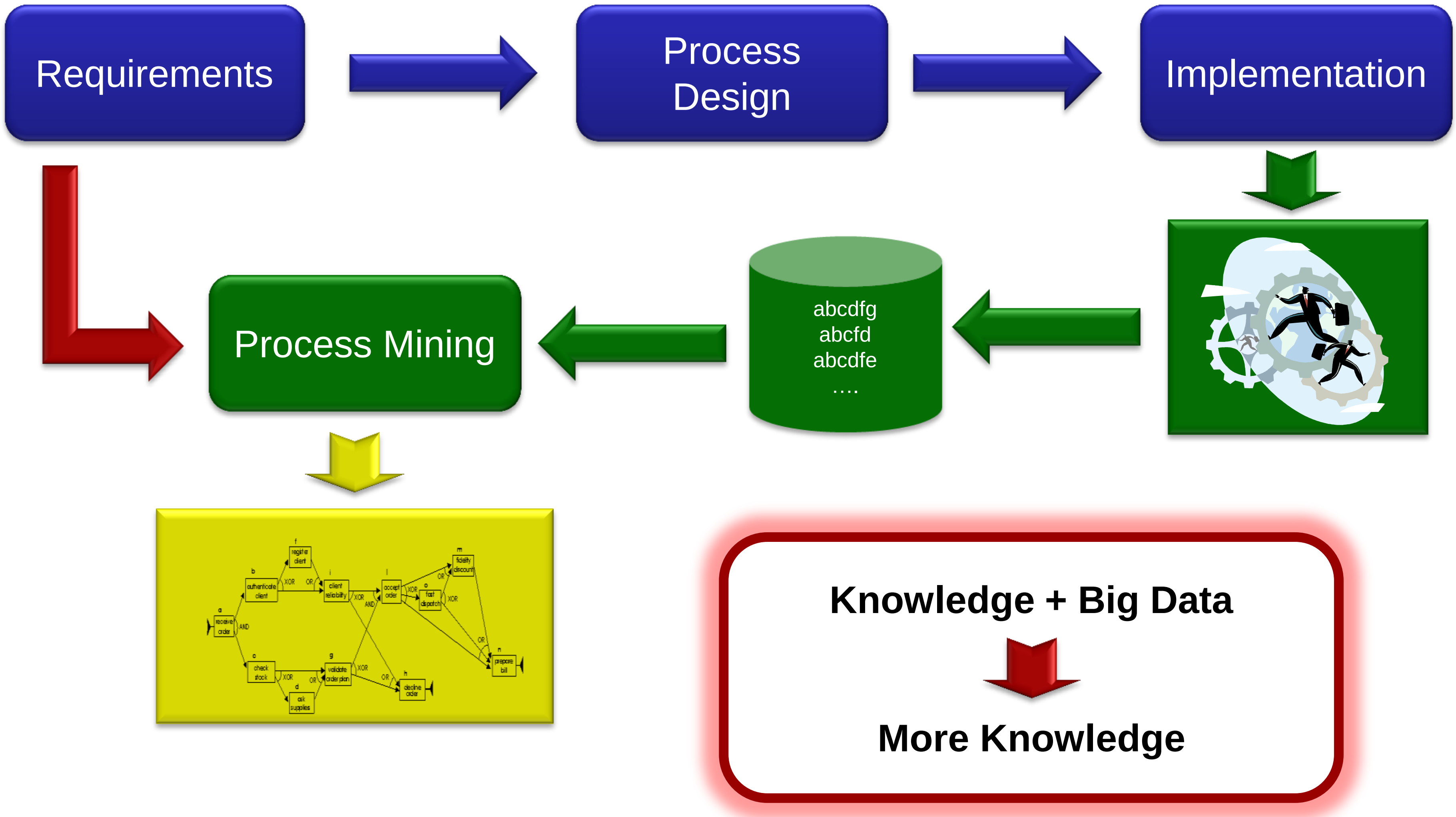
Process Mining



Process Mining + Background Knowledge



Process Mining + Background Knowledge



Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Framework
- Implementation Issues

Process Models

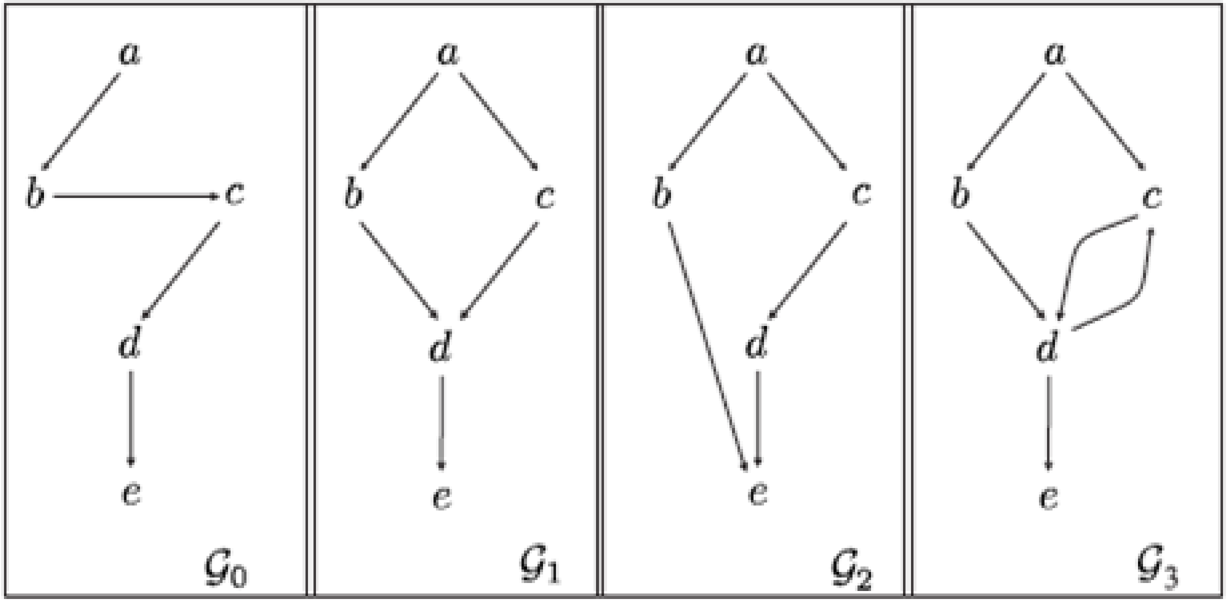
Dependency Graph: directed graphs whose nodes one-to-one correspond with the activities and such that an edge from an activity **a** to an activity **b** means that, in some enactment we expect that an actual flow of information can occur from **a** to **b**.

Process Models

Dependency Graph: directed graphs whose nodes one-to-one correspond with the activities and such that an edge from an activity **a** to an activity **b** means that, in some enactment we expect that an actual flow of information can occur from **a** to **b**.

Example

abcde
acbde



Process Models

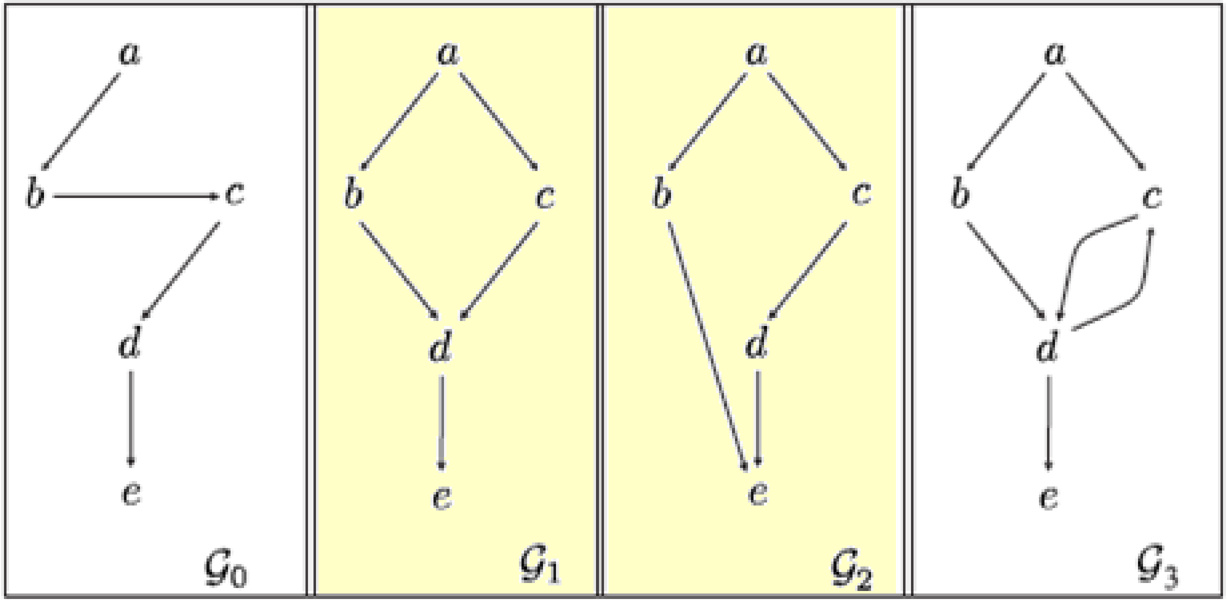
Dependency Graph: directed graphs whose nodes one-to-one correspond with the activities and such that an edge from an activity **a** to an activity **b** means that, in some enactment we expect that an actual flow of information can occur from **a** to **b**.

Example

abcde
acbde

$\mathcal{G}_1 \vdash \{abcde, acbde\}$

$\mathcal{G}_2 \vdash \{abcde, acbde\}$



Process Models

Dependency Graph: directed graphs whose nodes one-to-one correspond with the activities and such that an edge from an activity **a** to an activity **b** means that, in some enactment we expect that an actual flow of information can occur from **a** to **b**.

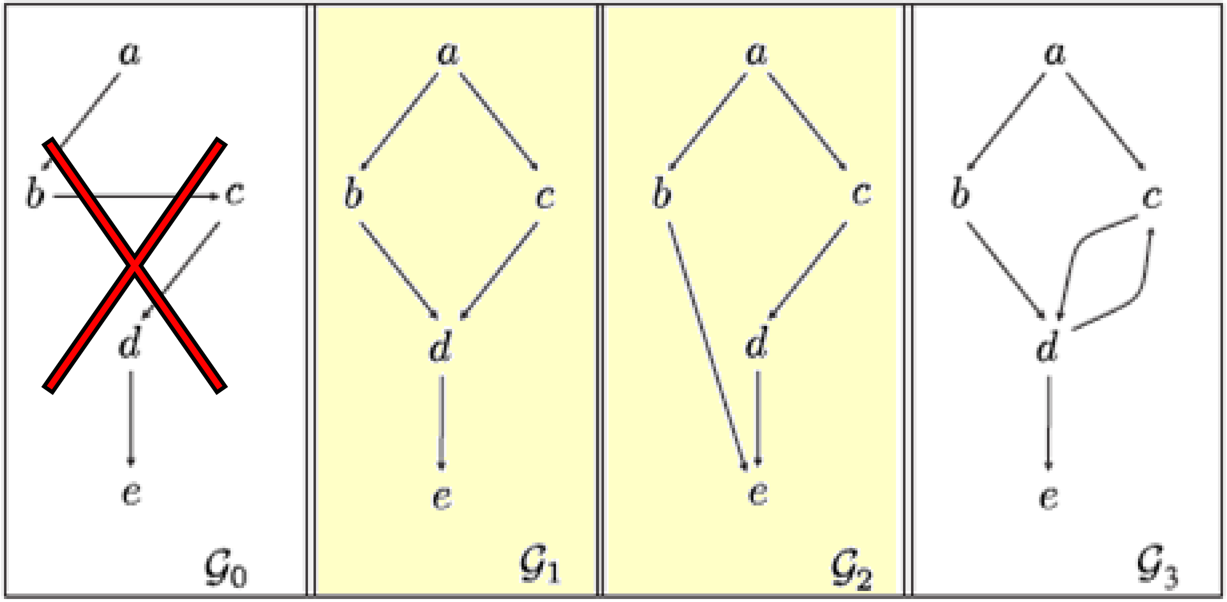
Example

abcde
acbde

$\mathcal{G}_0 \cancel{\vdash} \{abcde, acbde\}$

$\mathcal{G}_1 \vdash \{abcde, acbde\}$

$\mathcal{G}_2 \vdash \{abcde, acbde\}$



Precedence Constarints

- *Two kinds of positive constraints* π*:*
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$

  where $S \subseteq \mathcal{A}$*, with* $|S| \geq 1$*, is a non-empty set of activities and* $a \in \mathcal{A} \setminus S$ *is an activity.*

- *For a positive constraint* π*,* $\neg\pi$ *is a negative precedence constraint.*

Precedence Constarints

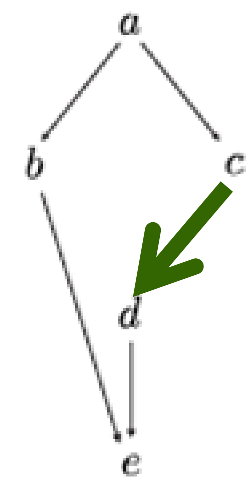
- Syntax
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*

- Semantics is interpreted over *directed graphs*

Precedence Constarints

- Syntax
  - *edge constraint* $S \to a$
  - *path constraint* $S \leadsto a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*
- Semantics is interpreted over *directed graphs*

- $\{a,c\} \longrightarrow d$

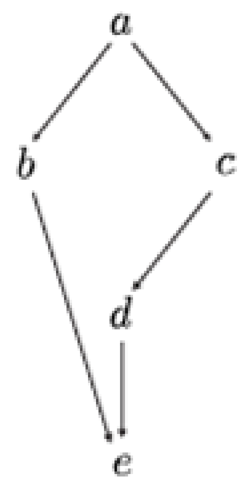


Precedence Constarints

- Syntax
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*

- Semantics is interpreted over *directed graphs*

- $\{a,c\} \longrightarrow d$
- $\{b\} \longrightarrow c$

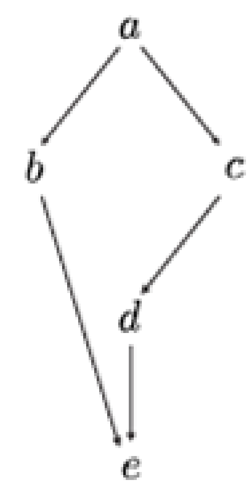


Precedence Constarints

- Syntax
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*

- Semantics is interpreted over *directed graphs*

- $\{a,c\} \longrightarrow d$
- $\cancel{\{b\} \longrightarrow c}$

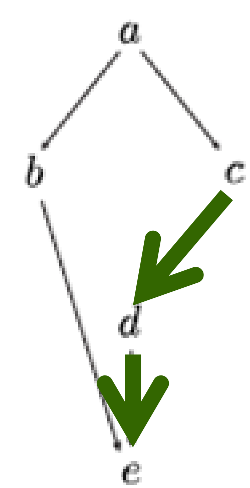


Precedence Constarints

- Syntax
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*

- Semantics is interpreted over *directed graphs*

- $\{a,c\} \longrightarrow d$
- $\cancel{\{b\} \longrightarrow c}$
- $\{c,b\} \rightsquigarrow e$

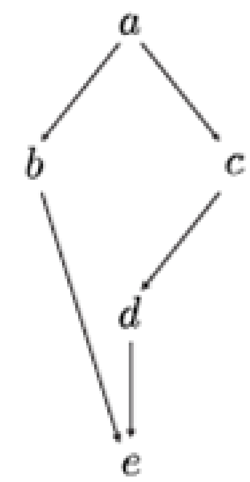


Precedence Constarints

- Syntax
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*

- Semantics is interpreted over *directed graphs*

- $\{a,c\} \longrightarrow d$
- $\cancel{\{b\} \rightarrow c}$
- $\{c,b\} \rightsquigarrow e$
- $\{c\} \rightsquigarrow b$

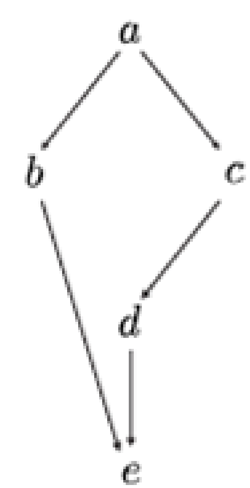


Precedence Constarints

- Syntax
  - *edge constraint* $S \rightarrow a$
  - *path constraint* $S \rightsquigarrow a$
  - *For a positive constraint* π, $\neg\pi$ *is a negative precedence constraint.*

- Semantics is interpreted over *directed graphs*

- $\{a,c\} \longrightarrow d$
- $\cancel{\{b\} \longrightarrow c}$
- $\{c,b\} \rightsquigarrow e$
- $\cancel{\{c\} \rightsquigarrow b}$



Revisiting Process Discovery

DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute a dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

Revisiting Process Discovery

DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute a dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

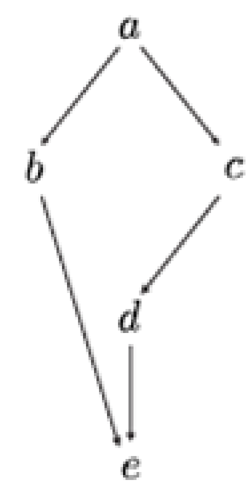
ACYCLIC-DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute an acyclic dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

Revisiting Process Discovery

DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute a dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

ACYCLIC-DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute an acyclic dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

$$\begin{cases} \Pi_0 = \{ \neg(\{b\} \rightsquigarrow d), \neg(\{d\} \rightsquigarrow b)\} \\ t_0 \;=\; abcde \end{cases}$$



$\mathcal{G}_2$

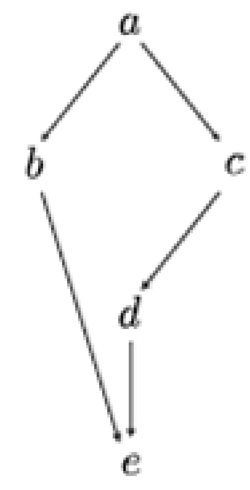
Revisiting Process Discovery

DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute a dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

ACYCLIC-DG-MINING: Given a log L and a set Π of precedence constraints over $\mathcal{A}(L)$, compute an acyclic dependency graph $\mathcal{G}$ for L with $\mathcal{G} \models \Pi$.

$$\begin{cases} \Pi_0 = \{ \neg(\{b\} \rightsquigarrow d), \neg(\{d\} \rightsquigarrow b)\} \\ t_0 = abcde \end{cases}$$

$\mathcal{G}_2$ is a solution to DG-MINING (on input $\{t_0\}$ and Π_0)



$\mathcal{G}_2$

A Closer Look

- As a result of our formulation, process discovery is conceptually carried out via:
  - a *learning task* (i.e., building all possible dependency graphs for a given input log), followed by
  - a *reasoning task* (i.e., to filter out those graphs that do not satisfy the precedence constraints defined by the analyst)

A Closer Look

- As a result of our formulation, process discovery is conceptually carried out via:
  - a *learning task* (i.e., building all possible dependency graphs for a given input log), followed by
  - a *reasoning task* (i.e., to filter out those graphs that do not satisfy the precedence constraints defined by the analyst)

exponentially many dependency graphs might be built in the learning phase

A Closer Look

- As a result of our formulation, process discovery is conceptually carried out via:
  - a *learning task* (i.e., building all possible dependency graphs for a given input log), followed by
  - a *reasoning task* (i.e., to filter out those graphs that do not satisfy the precedence constraints defined by the analyst)

exponentially many dependency graphs might be built in the learning phase

↓

A two-phase approach is unfeasible

«Compiling» Logs into Constraints

- Let L be a log. For each trace $t[1]...t[n] \in L$,

$$\boxed{\pi(t) = \{\, \{t[1], ..., t[i-1]\} \to t[i] \;\mid 1 < i \leq n\,\}}$$

Moreover, let $\pi(L) = \bigcup_{t \in L} \pi(t)$.

«Compiling» Logs into Constraints

- Let L be a log. For each trace $t[1]...t[n] \in L$,

$$\boxed{\pi(t) = \{\ \{t[1], ..., t[i{-}1]\} \to t[i]\ \mid 1 < i \leq n\,\}}$$

Moreover, let $\pi(L) = \bigcup_{t \in L} \pi(t)$.

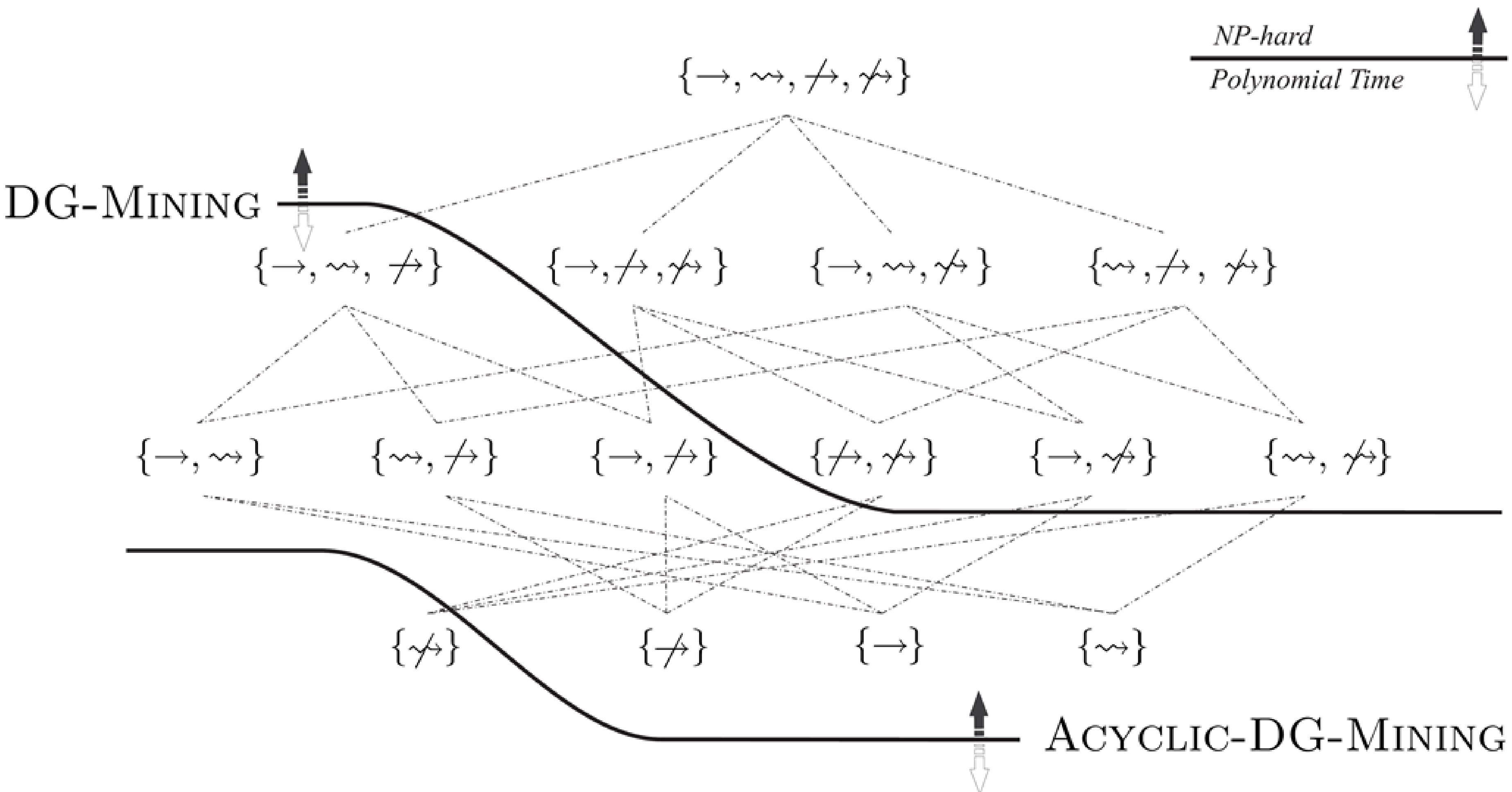
Theorem

$\mathcal{G}$ is a solution to DG-MINING *(resp.,* ACYCLIC-DG-MINING*) on input L and Π*

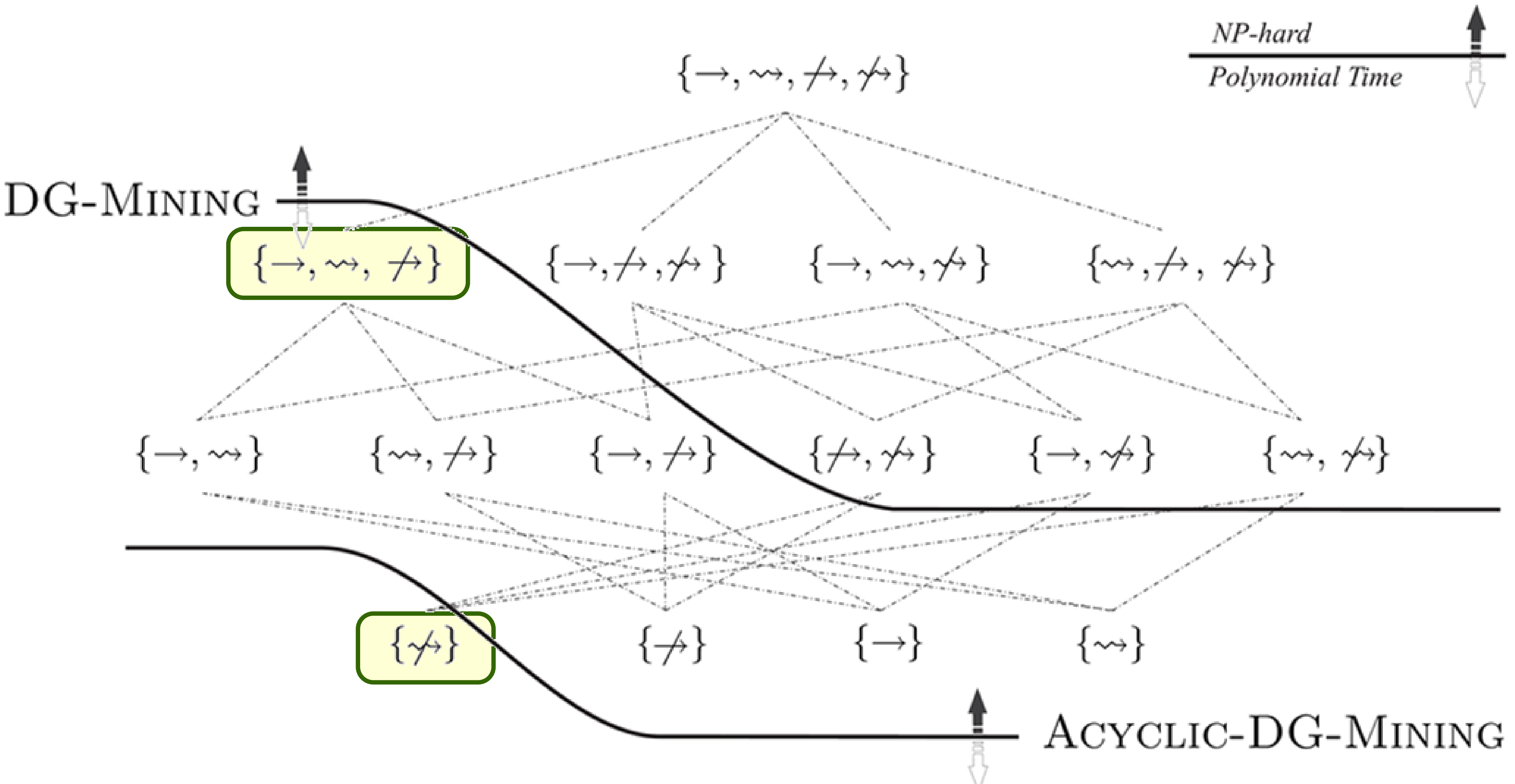
$$\Updownarrow$$

$$\mathcal{G} \models \pi(L) \cup \Pi.$$

Complexity Analysis



Complexity Analysis



Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Framework
- **Implementation Issues**

CP encoding: preliminaries

- The mining of a dependency graph based on precedence constraints is turned into a constraints satisfaction problem (CSP) or a constraint satisfaction optimization problem (CSOP)
  - existing constraint programming platforms can bereused to efficiently compute models

CP encoding: preliminaries

- The mining of a dependency graph based on precedence constraints is turned into a constraints satisfaction problem (CSP) or a constraint satisfaction optimization problem (CSOP)
  - existing constraint programming platforms can bereused to efficiently compute models
- Formally, a **CSP instance** is a triple (Var,U,C), where
  - $Var = \{X_1,\ldots,X_m\}$ is a finite set of variables
  - U is a function mapping each variable $X_i \in Var$ to a domain $U(X_i)$ of values
  - C is a finite set of constraints, i.e., boolean functions over $\{X_{i1},\ldots,X_{ik}\}$, such as:
    - summation constraints, of the form: $\Sigma\, w_j \times X_j \geq \gamma$
    - reified (summation) constraints, of the form: $\Sigma\, w_j \times X_j \geq \gamma \leftrightarrow X$

… where X is a boolean variable, while γ and all w_j are real numbers

CP encoding: preliminaries

- The mining of a dependency graph based on precedence constraints is turned into a constraints satisfaction problem (CSP) or a constraint satisfaction optimization problem (CSOP)
  - existing constraint programming platforms can bereused to efficiently compute models
- Formally, a **CSP instance** is a triple (Var,U,C), where
  - $Var = \{X_1,\dots,X_m\}$ is a finite set of variables
  - U is a function mapping each variable $X_i \in Var$ to a domain $U(X_i)$ of values
  - C is a finite set of constraints, i.e., boolean functions over $\{X_{i1},\dots,X_{ik}\}$, such as:
    - summation constraints, of the form: $\Sigma\, w_j \times X_j \geq \gamma$
    - reified (summation) constraints, of the form: $\Sigma\, w_j \times X_j \geq \gamma \leftrightarrow X$

    … where X is a boolean variable, while γ and all w_j are real numbers
- An **assignment θ** for the CSP instance (Var,U,C) is a function mapping each variable $X_i \in Var$ to an element of its associated domain $U(X_i)$
  - θ is a **solution** to (Var,U,C) if it satisfies all the constraints in C

CP encoding: preliminaries

- The mining of a dependency graph based on precedence constraints is turned into a constraints satisfaction problem (CSP) or a constraint satisfaction optimization problem (CSOP)
  - existing constraint programming platforms can bereused to efficiently compute models
- Formally, a **CSP instance** is a triple (Var,U,C), where
  - $Var = \{X_1,\ldots,X_m\}$ is a finite set of variables
  - U is a function mapping each variable $X_i \in Var$ to a domain $U(X_i)$ of values
  - C is a finite set of constraints, i.e., boolean functions over $\{X_{i1},\ldots,X_{ik}\}$, such as:
    - summation constraints, of the form: $\Sigma\, w_j \times X_j \geq \gamma$
    - reified (summation) constraints, of the form: $\Sigma\, w_j \times X_j \geq \gamma \leftrightarrow X$

    … where X is a boolean variable, while γ and all w_j are real numbers
- An **assignment θ** for the CSP instance (Var,U,C) is a function mapping each variable $X_i \in Var$ to an element of its associated domain $U(X_i)$
  - θ is a **solution** to (Var,U,C) if it satisfies all the constraints in C
- In addition to constraints, in a **CSOP** instance, an optimal solution is searched, minimizing a linear cost function of the form

$$f(\theta) = \sum_{i=1}^{n} w_i \times \theta(X_i)$$

Basic encoding algorithm *PCtoCSP*

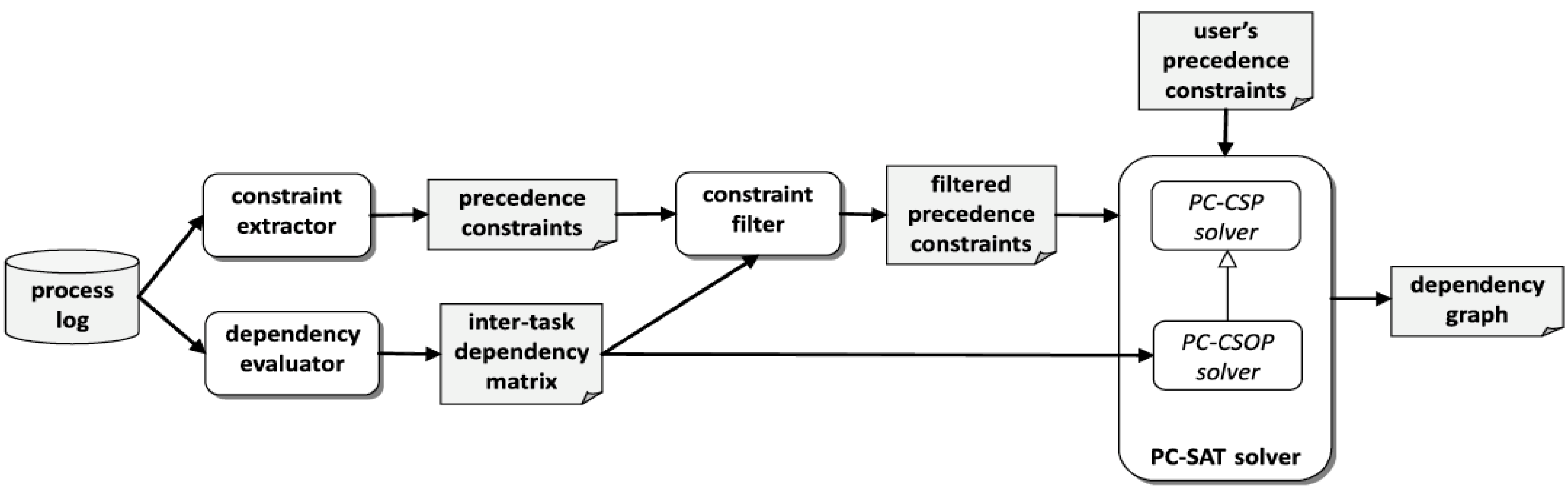
A given set of precedence constraints over activities $\{a_1, \ldots, a_n\}$ is encoded into a CSP instance, containing a series of **variables for each pair a_i and a_j** of activities:

- an "edge" variables $e[a_i,a_j]$,
- "path" variables $p[a_i,a_j]^l$ and "path-through" variables $p[a_i,a_k,a_j]^l$, for $k,l = 1..n$, where l denotes the maximum number of edges in the respective path

Basic encoding algorithm *PCtoCSP*

A given set of precedence constraints over activities $\{a_1, \ldots, a_n\}$ is encoded into a CSP instance, containing a series of **variables for each pair a_i and a_j** of activities:

- an "edge" variables $e[a_i,a_j]$,
- "path" variables $p[a_i,a_j]^l$ and "path-through" variables $p[a_i,a_k,a_j]^l$, for $k,l = 1..n$, where l denotes the maximum number of edges in the respective path



CP solving algorithms

- The computation of the dependency graph for a given set of precedence constraints is carried out by *PC-SAT solver* module, based on Gecode

CP solving algorithms

- The computation of the dependency graph for a given set of precedence constraints is carried out by *PC-SAT solver* module, based on Gecode
- As a basic solution scheme, PC-CSP uses backtracking, while PC-CSOP uses a branch-and-bound approach.

CP solving algorithms

- The computation of the dependency graph for a given set of precedence constraints is carried out by *PC-SAT solver* module, based on Gecode
- As a basic solution scheme, PC-CSP uses backtracking, while PC-CSOP uses a branch-and-bound approach.
- During the exploration, all solution algorithms alternate two kinds of steps:
  - *branching*, where a value is assigned to some variables as in standard search methods, and
  - *constraint propagation*, where different constraints can be iteratively applied as to shrink the space of the possible dependency graphs and propagate the consequences of choices made in the previous steps

CP solving algorithms

- The computation of the dependency graph for a given set of precedence constraints is carried out by *PC-SAT solver* module, based on Gecode
- As a basic solution scheme, PC-CSP uses backtracking, while PC-CSOP uses a branch-and-bound approach.
- During the exploration, all solution algorithms alternate two kinds of steps:
  - *branching*, where a value is assigned to some variables as in standard search methods, and
  - *constraint propagation*, where different constraints can be iteratively applied as to shrink the space of the possible dependency graphs and propagate the consequences of choices made in the previous steps
- The PC-SAT solver module exploits standard Gecode's propagators for all kinds of constraints required in our framework.

CP solving algorithms

- The computation of the dependency graph for a given set of precedence constraints is carried out by *PC-SAT solver* module, based on Gecode
- As a basic solution scheme, PC-CSP uses backtracking, while PC-CSOP uses a branch-and-bound approach.
- During the exploration, all solution algorithms alternate two kinds of steps:
  - *branching*, where a value is assigned to some variables as in standard search methods, and
  - *constraint propagation*, where different constraints can be iteratively applied as to shrink the space of the possible dependency graphs and propagate the consequences of choices made in the previous steps
- The PC-SAT solver module exploits standard Gecode's propagators for all kinds of constraints required in our framework.
- Ad-hoc branching policies have been implemented:
  - we first branch on the edge variables before considering path variables;
  - all variables $p[a_i,a_j]^l$ are always considered before those of the form $p[a_i,a_k,a_j]^l$

Heuristics

In order to pragmatically reduce the size of the search space and speed-up the computation, three types of heuristics can be used:

1. **Redundancy Reduction**. Two policies, relying on two different notions of constraint subsumption:
   - A constraint $S \longrightarrow a$ is filtered out if there is another precedence constraint $S' \longrightarrow a$ such that $S' \supset S$
   - A constraint $S \longrightarrow a$ is filtered out if there exist another constraint $S'' \longrightarrow a$ such that $S'' \subset S$
     - this (weaker) notion allows for recognizing skip-like control flow structures, where some synchronizing (i.e. join) activity a can be activated by an activity in $S \setminus S''$ or, optionally, by an activity in $S \cap S''$.

Heuristics

In order to pragmatically reduce the size of the search space and speed-up the computation, three types of heuristics can be used:

1. **Redundancy Reduction**. Two policies, relying on two different notions of constraint subsumption:
   - A constraint $S \rightarrow a$ is filtered out if there is another precedence constraint $S' \rightarrow a$ such that $S' \supset S$
   - A constraint $S \rightarrow a$ is filtered out if there exist another constraint $S'' \rightarrow a$ such that $S'' \subset S$
     - this (weaker) notion allows for recognizing skip-like control flow structures, where some synchronizing (i.e. join) activity a can be activated by an activity in $S \setminus S''$ or, optionally, by an activity in $S \cap S''$.

2. **Closed World Assumption (CWA)**. In order to reduce the size of the search space, further constraints are introduced as follows:
   - an edge (*x*, *y*) is not permitted to appear in the model if activity *y* never follows activity *x*, (directly or ndirectly), in any trace of the log.
   - In the case of unfolding, CWA constraints are expressed over real activities, rather than on their unfolded versions.

Heuristics: Constraint Size reduction

- The **nr. of elements in constraint bodies** is a key factor for scalability.
- Let $\{ t[1], ..., t[i-1] \} \rightarrow t[i]$ be a constraint in the set $\pi(t)$ of precedence constraints derived from a given trace $t[1], ..., t[n]$.
- Three strategies for shrinking the size of the body (i.e., left hand part):

a) *Maximal horizon H over past activities*:
  - remove each $t[j]$ s.t. $j < i - H$, from $\{ t[1], ..., t[i-1] \}$

b) *Two kinds of lower thresholds* for edge weights: $\boldsymbol{\sigma}_{\mathbf{abs}}$ ("absolute"), and $\boldsymbol{\sigma}_{\mathbf{r2b}}$ ("relative to best predecessor", like in *Heuristics Miner*)
  - remove any t[j] such that *weight*(t[j],t[i]) < σ_{abs}
  - remove any t[j] s.t. *weight*(t[j],t[i]) < $\sigma_{r2b} \times \text{argmax}_{1 \le k < I}$ { *weight*(t[k], t[i]) }

c) *Maximum number $\boldsymbol{K_{top}}$ of activities that can occur in the body:*
  - at most K_{top} elements are kept, with top dependency scores (w.r.t. t[j])

Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Framework
- Implementation Issues

Outline

- Application Domain
- Process Mining Approaches @UniCAL
- Another Challenge in Process Mining
- Formal Frame
- Implementation Issues

Thank you!