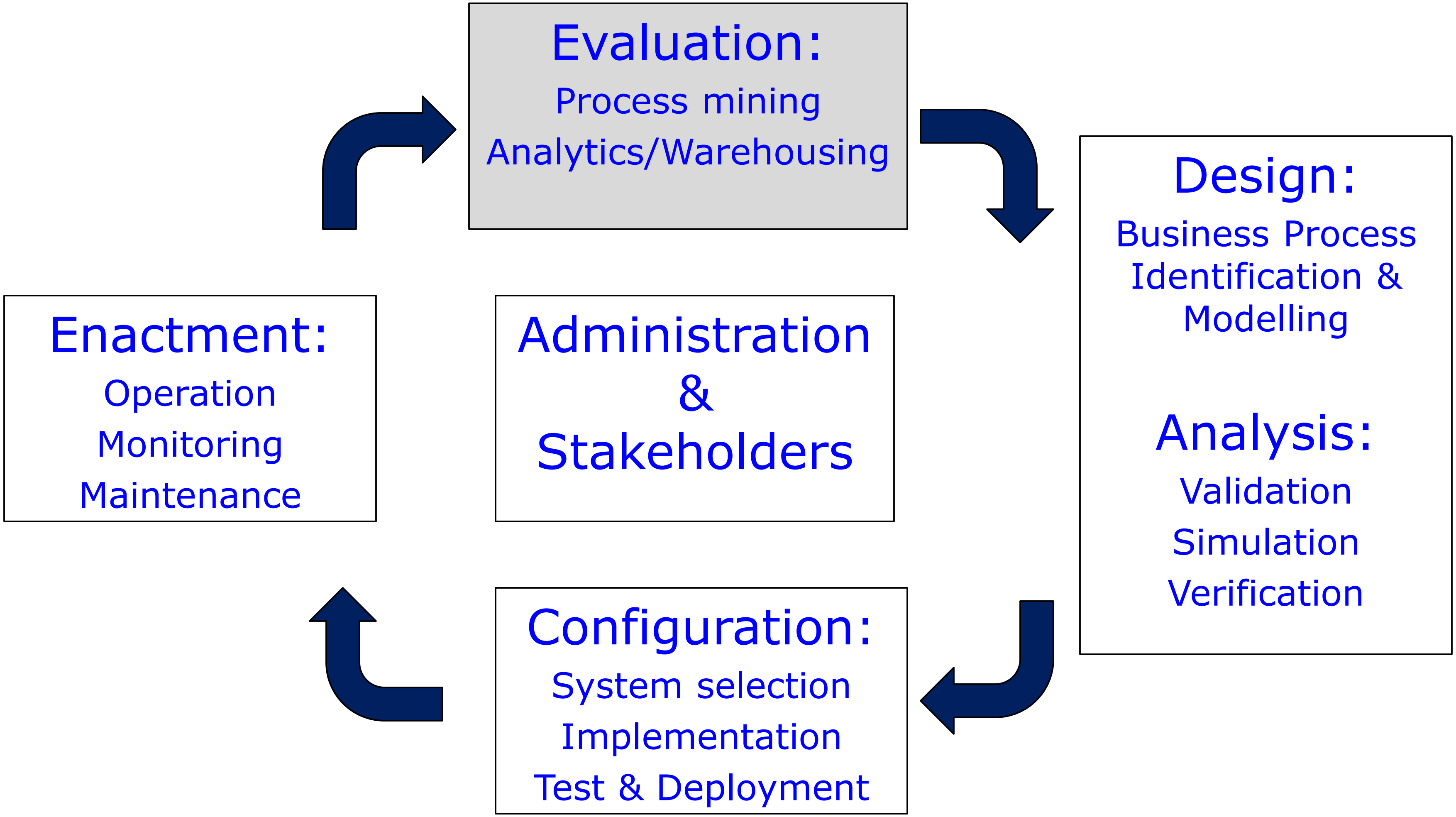
CS565 - Business Process Management Systems

Process Flexibility

BUSINESS PROCESS LIFECYCLE



Evaluation:
Process mining
Analytics/Warehousing

Design:
Business Process Identification & Modelling

Analysis:
Validation
Simulation
Verification

Configuration:
System selection
Implementation
Test & Deployment

Enactment:
Operation
Monitoring
Maintenance

Administration & Stakeholders

BUSINESS PROCESS FLEXIBILITY

- BP Flexibility is the ability of a BP to address changes in the context or operating environment.
- Changes can be:
  - Foreseen
  - Unforeseen
- Addressing includes:
  - Varying or adapting those BPs that are affected by changes
  - Retaining those parts not affected

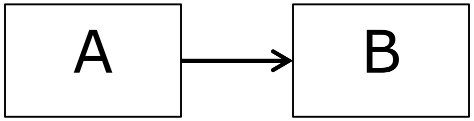
FLEXIBILITY TYPES

- 4 types or approaches:
  - By design: design-time specification of strategies to address foreseen changes
  - By deviation: small deviation from "as-is" BP to handle occasional unforeseen changes
  - By underspecification: similar to 1st but the addressing is handled at run-time (strategy not known or not generally applicable)
  - By change: actual process adaptation & evolution to handle both occasional and permanent unforeseen changes

BP SPECIFICATION FOR FLEXIBILITY

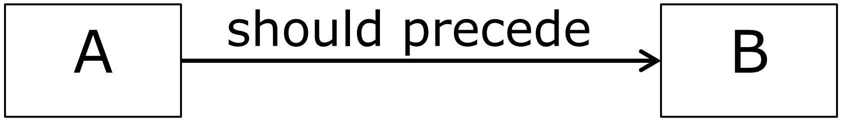
- Imperative/procedural approach: precise definition of how BP tasks must be executed
  - Flexibility achieved by adding execution paths
- Declarative approach: focuses on what must be done and not how
  - By default, all execution paths are allowed
  - The more constraints are provided, the more execution paths are filtered
  - Constraints are relations between tasks
  - Both mandatory & optional constraints are allowed
  - Flexibility achieved by removing or weakening constraints

BP SPECIFICATION FOR FLEXIBILITY



Imperative approach

{[A,B]}



Declarative approach

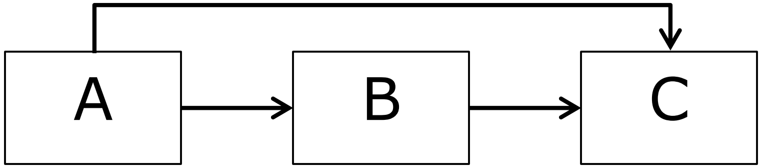
{[A], [A,A], [A,B,A], [A,B,B], …}

FLEXIBILITY BY DESIGN

- Model alternative execution paths at design-time to anticipate for foreseen changes at runtime
- Each foreseen change maps to selecting one alternative path
- Realisation options (most common):
  - parallelism,
  - choice,
  - iteration,
  - Interleaving (execute tasks in any order but not concurrently),
  - multiple instances (of a task),
  - cancellation (of a task now or in the near future)

FLEXIBILITY BY DESIGN

- Workflow patterns cover all possible options
- Realisation options thoroughly studied at imperative approach
- Equally applicable to declarative approach through the use of less constructs/constraints
- Realisation options can be differently implemented
  - E.g., deferred vs. exclusive choice



- Drawbacks:
  - Model complexity increases
  - Impossible to model unlimited or unknown alternative cases

FLEXIBILITY BY DEVIATION

- Temporally deviate from prescribed execution sequence to accommodate changes in operating environment at runtime
  - Swap task order between "register patient" and "perform triage" in a clinical emergency situation
- Actual process definition is not altered or the tasks included in it
- Just execution sequence of particular instance is modified
- Realisation options:
  - Vary actual tasks to be executed (from those enabled)
    - Imperative: apply deviation operations
    - Declarative: just violate optional constraints

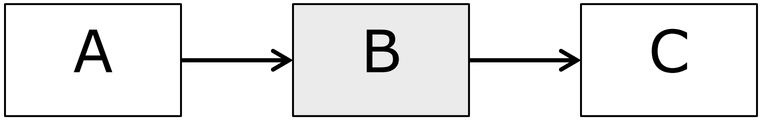
FLEXIBILITY BY DEVIATION

- Deviation operations:
  - Undo task A: shift control at moment before execution of A. Does not always imply that task actions are undone or reversed.
  - Redo task A: re-execute task A without shifting control. Small example: re-enter data that have been wrongly provided
  - Skip task A: pass control to the next task from A. Skipped task is not compensated. Quite useful in emergency situations with the skipping of non-critical tasks
  - Create additional instance of A: to run in parallel with process instances created on the moment of task instantiation. Flexibility can be controlled by limiting the number of concurrent task instances running in parallel. Example: do a separate reservation for a set of people in trip arrangement

FLEXIBILITY BY DEVIATION

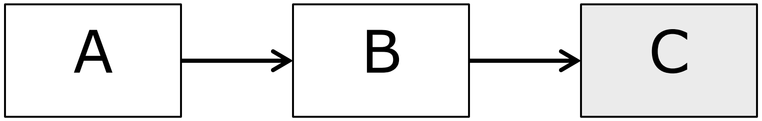
- Deviation operations:
  - Invoke task A: initiate task not enabled & executed in current execution. The thread of control is not altered. Example: an additional task, not foreseen, should be executed to check if provided data are fraudulent in a reviewing insurance claim process instance. Then, next task in order normally takes place.
- Deviation operations can be differently implemented
- Additional requirements for each operation could be provided
  - E.g., A can be undone only when A has been previously executed
- Should identify which operations have been performed in the execution trace. Different ways can be used to perform this
  - Undo operations can be logged either explicitly or by just removing affected task from execution trace

FLEXIBILITY BY DEVIATION



Before skipping B

Trace: [A]



After skipping B

Trace: [A, “skip B”]

FLEXIBILITY BY DEVIATION

- Drawbacks:
  - Who decides about performing which deviation operations & what knowledge does he/she have available for this?
  - Not all operations may have realizations
  - Some tasks might be difficult to undo their effects
  - Not suitable for cases where more drastic changes must occur at process structure & process replanning is actually needed

FLEXIBILITY BY UNDESPECIFICATION

- When all execution paths cannot be defined in advance, it is required to dynamically add them as process fragments at runtime, thus executing incomplete process definitions
- BP model is not modified at runtime but just missing information is filled in for undefined parts
- More suitable for BPs where it is known beforehand which points need to be adjusted. Content for these points is not yet known. Also suitable when overall BP structure is fixed but different parts are designed and controlled by different work groups.
- Incomplete process definition contains underspecified placeholders. Their content is specified when executed.

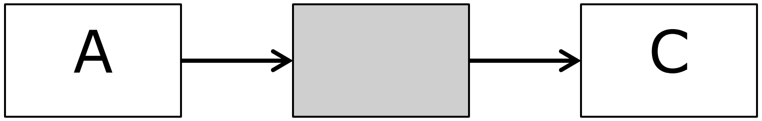
FLEXIBILITY BY UNDERSPECIFICATION

- Two types of placeholder enactment exist:
  - Late binding: select one process fragment from the candidate ones to realize placeholder. Candidate list is fixed.
  - Late modelling: An existing process fragment can be selected or a new one can be specified. Subsumes former type.
- Process fragments are stored in a repository
- Two moments for realization can be exploited:
  - Before placeholder execution: either when process instance is commenced or before the placeholder is executed the first time
  - At placeholder execution

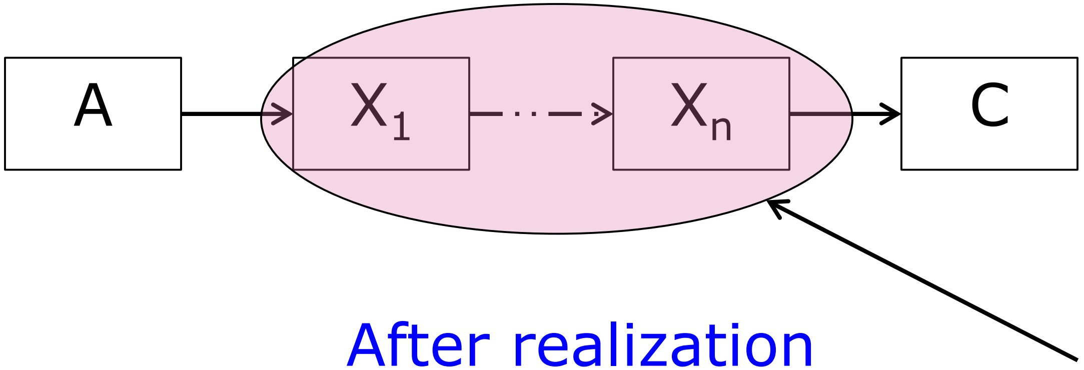
FLEXIBILITY BY UNDERSPECIFICATION

- Placeholders can be realized once or multiple times. Two realisation types exist:
  - Static realisation: initial placeholder realization is used for all subsequent placeholder executions
  - Dynamic realisation: placeholder is realized for each execution
- Drawbacks:
  - When should a new fragment must be created instead of using available, candidate ones?
  - Manual or automatic construction of new fragment?
  - Cannot perform adaptation for BP points not foreseen

FLEXIBILITY BY UNDERSPECIFICATION



Before realization



After realization

Process fragment

FLEXIBILITY BY CHANGE

- Events cannot always be addressed by small temporal deviations from prescribed process definition
- Process replanning must be performed
- Either some process instances are modified or even the process model (process evolution)
- One or more currently executed process instances must be migrated to new process definition

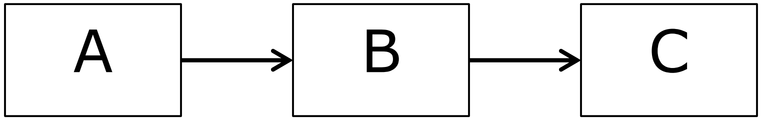
FLEXIBILITY BY CHANGE

- Variation points:
  - Effect/Impact of change: changes are performed at the instance or model level.
    - Momentary change (instance level)
    - Evolutionary change (process model level)
  - Moment of allowed change at instance or model level:
    - Entry time: changes performed only when instance is created. For evolutionary changes, only new instances are affected (old stay with previous model)
    - On-the-fly: changes performed at any point in process execution. Momentary changes map to customizing modified instance. Evolutionary changes are propagated to both new and old instances. Old instances must be migrated.

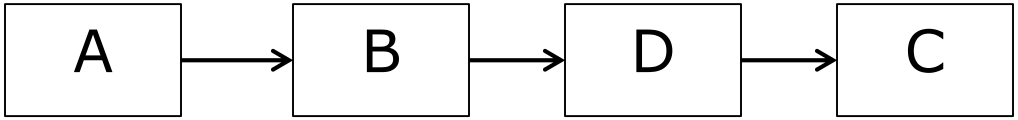
FLEXIBILITY BY CHANGE

- Variation points:
  - Migration strategy: indicates the handling of instances impacted by evolutionary change
    - Backward recovery: instances are aborted
    - Forward recovery: instances are aborted, possibly compensated and restarted
    - Proceed: changes are ignored by old process instances
    - Transfer: instances are transferred to respective state in new process definition
- Drawback:
  - Can always old instances be migrated?
  - Which migration strategy to choose from?
  - When should we move to an evolutionary change?
  - More time consuming than performing small deviations

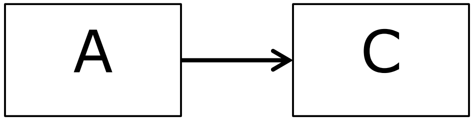
FLEXIBILITY BY CHANGE



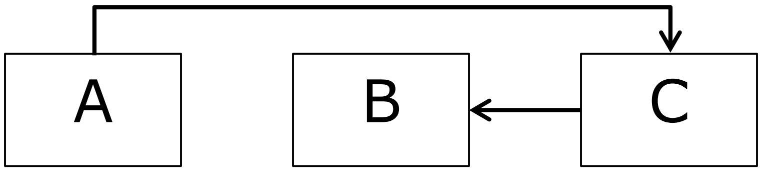
Initial Model



Extend

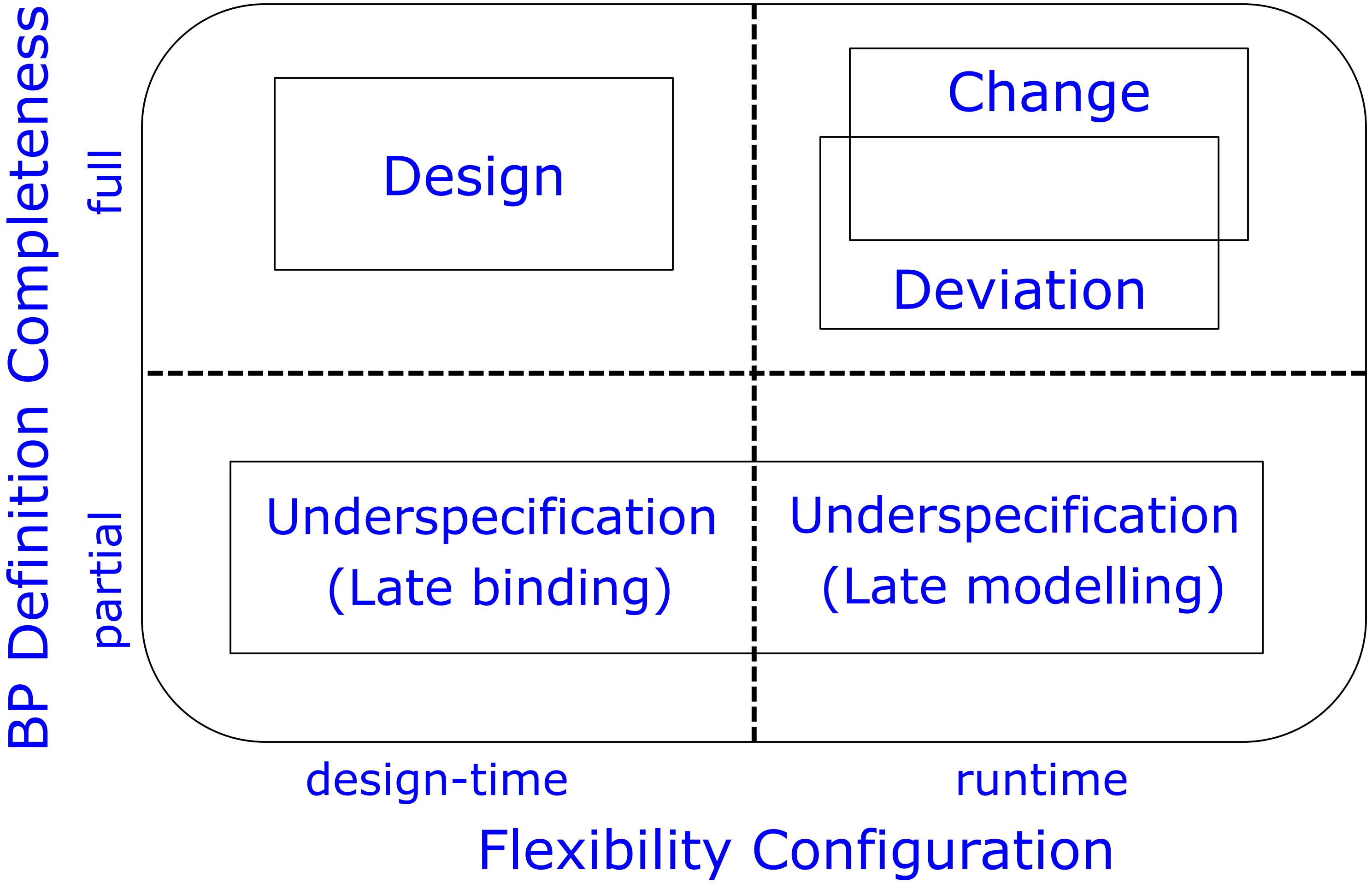


Reduce



Relink

FLEXIBILITY SPECTRUM



FLEXIBILITY WORK COMPARISON

	ADEPT	YAWL	FLOWer	DECLARE
Parallelism	+	+	+	+
Choice	+	+	+	+
Iteration	+	+	+	+
Interleaving		+	+/-	+
Multiple Instances		+	+	+
Cancellation		+		+
Undo			+	▨
Redo			+	▨
Skip			+	▨
Create additional instance				▨
Invoke task			+	▨
Violation of optional constraints	▨	▨	▨	+

FLEXIBILITY WORK COMPARISON

	ADEPT	YAWL	FLOWer	DECLARE
Late binding		+		
Late modelling		+		
Static, before placeholder				
Dynamic, before placeholder				
Static, at placeholder				
Dynamic, at placeholder		+		
Momentary change	+			+
Evolutionary change				+
Entry time	+			+
On-the-fly	+			+
Forward recovery				
Backward recovery				
Proceed				+
Transfer				+

CHALLENGES

- Support for all types of flexibility
- Accommodate for additional perspectives:
  - Organisational
  - Information
  - Application
- Use process mining to discover adaptation logic in system supporting deviation and/or change operations

CHANGE OPERATIONS

- Change patterns on control-flow of a BP have been proposed
  - Focus on high-level BP adaptation
  - Are associated to pre- & post-conditions to guarantee soundness of resulting model
- Change support features were also proposed
  - Guarantee that changes are performed in a correct and consistent way, change traceability is enabled and process changes facilitate users
- Both can be used for evaluating approaches in BP adaptation
- Not all BP aspects have been covered (data flow, resources)

CHANGE PATTERNS

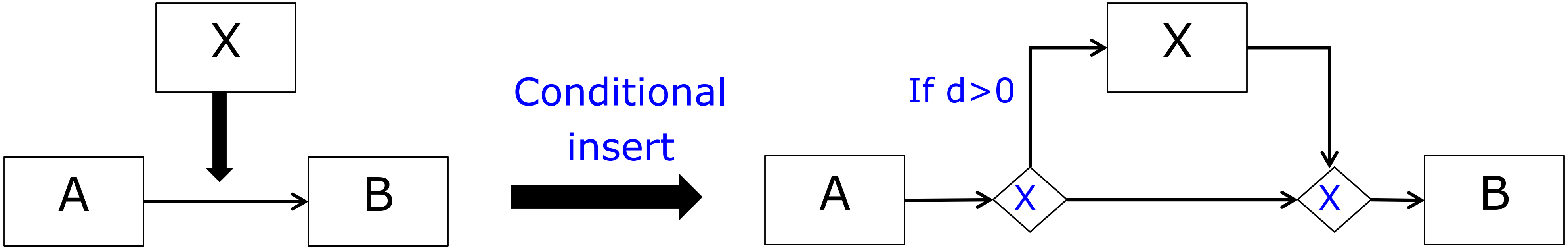
- Two main categories:
  - Adaptation patterns: modify BP at model or instance level by applying high-level operations (e.g., activity insertion); can be applied at the whole BP schema. Low-level operations are not considered due to lack of abstraction & not guaranteeing model soundness.
  - Patterns for changes in predefined regions: allow participants to complete information for unspecified BP parts during runtime.

	Adaptation Pattern	Patterns in changes to predefined regions
Structural process change	YES	NO
Anticipation of change	NO	YES
Change restricted to specific regions	NO	YES
Application area	Unanticipated exceptions, unforeseen situations	Address uncertainty by deferring decisions at runtime

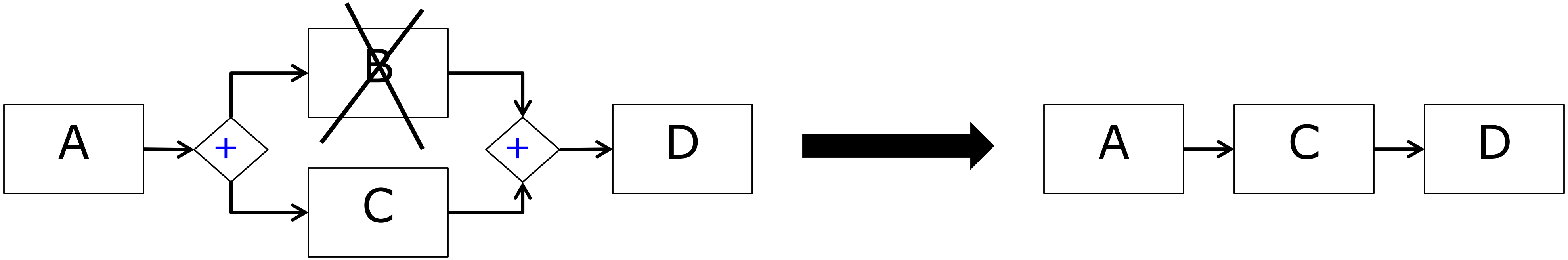
ADAPTATION PATTERNS

- Insert Process Fragment
  - Design choice: where to embed the fragment
    - serial inclusion between succeeding activities by some approaches
    - others allow insertion of fragment between two activity sets that meet certain constraints (e.g., parallel or conditional insert)
- Delete Process Fragment
  - Straightforward (single design choice)
  - Different realizations:
    - Physically delete the fragment from the model
    - Replace fragment with silent/empty activity
    - Fragment embedded in conditional branch with condition equal to FALSE

EXAMPLES



Insert process fragment

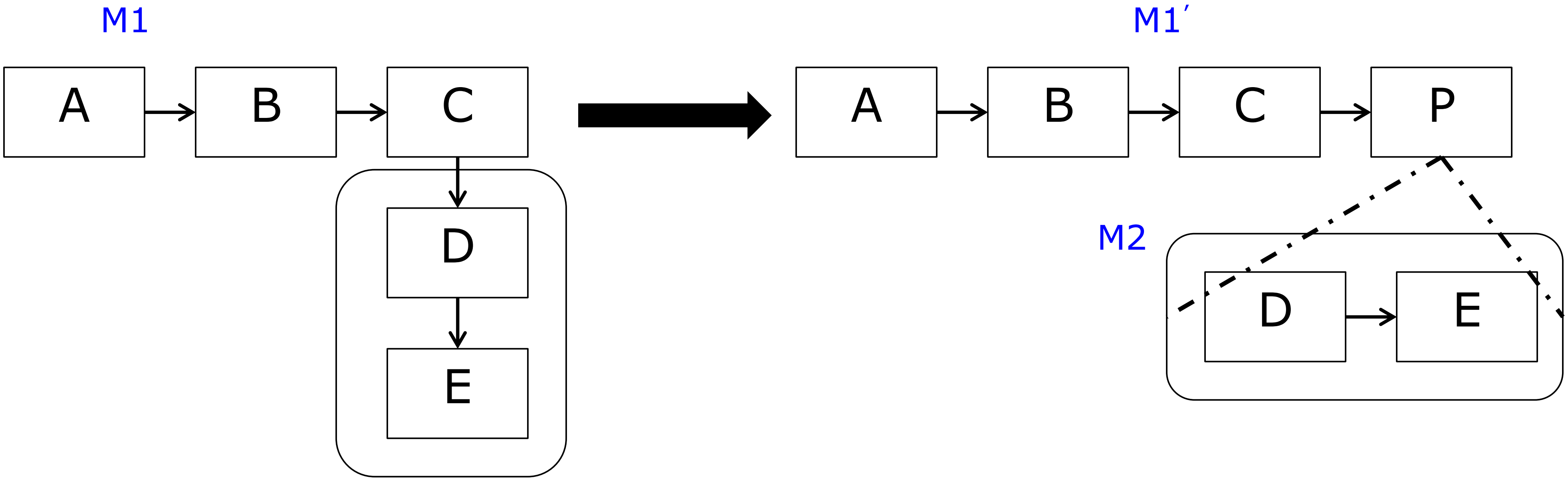


Delete process fragment

ADAPTATION PATTERNS

- Move Process Fragment
  - Design choice: where to embed fragment
  - Can be realized as a sequence of delete & insert
- Replace Process Fragment
  - Can be realized as a sequence of delete & insert
- Swap Process Fragment
  - Can be realized as above patterns or with two moves
- Extract SubProcess
  - Extract process fragment from one model and encapsulate in a separate sub-schema/model
  - Add hierarchy level to simplify schema or hide information from users
  - Implemented through graph aggregation techniques

EXAMPLES

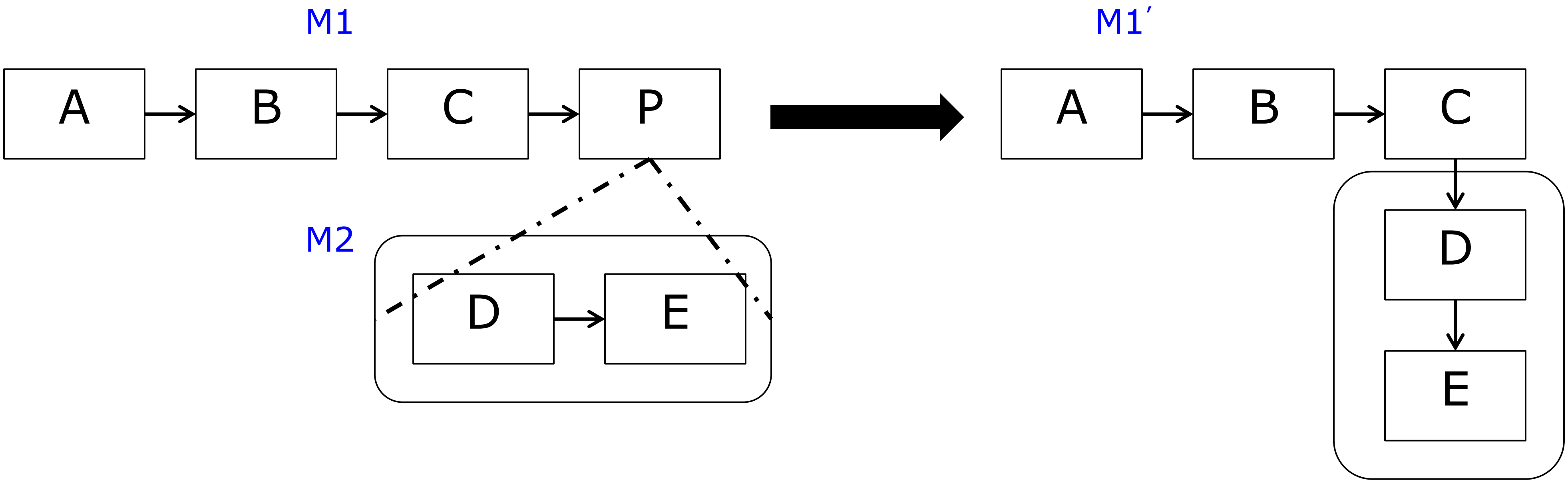


Extract SubProcess

ADAPTATION PATTERNS

- Inline SubProcess
  - Opposite to Extract
  - Flattens hierarchy of process (to e.g. reduce levels)
- Embed process fragment in a loop
  - Could be realized through combining patterns for adding process fragment, inserting & deleting control dependency
- Parallelize Process Fragments
  - Parallelize fragments which were sequentially executed
  - Realized through either inserting & deleting control dependency or moving process fragment patterns

EXAMPLES

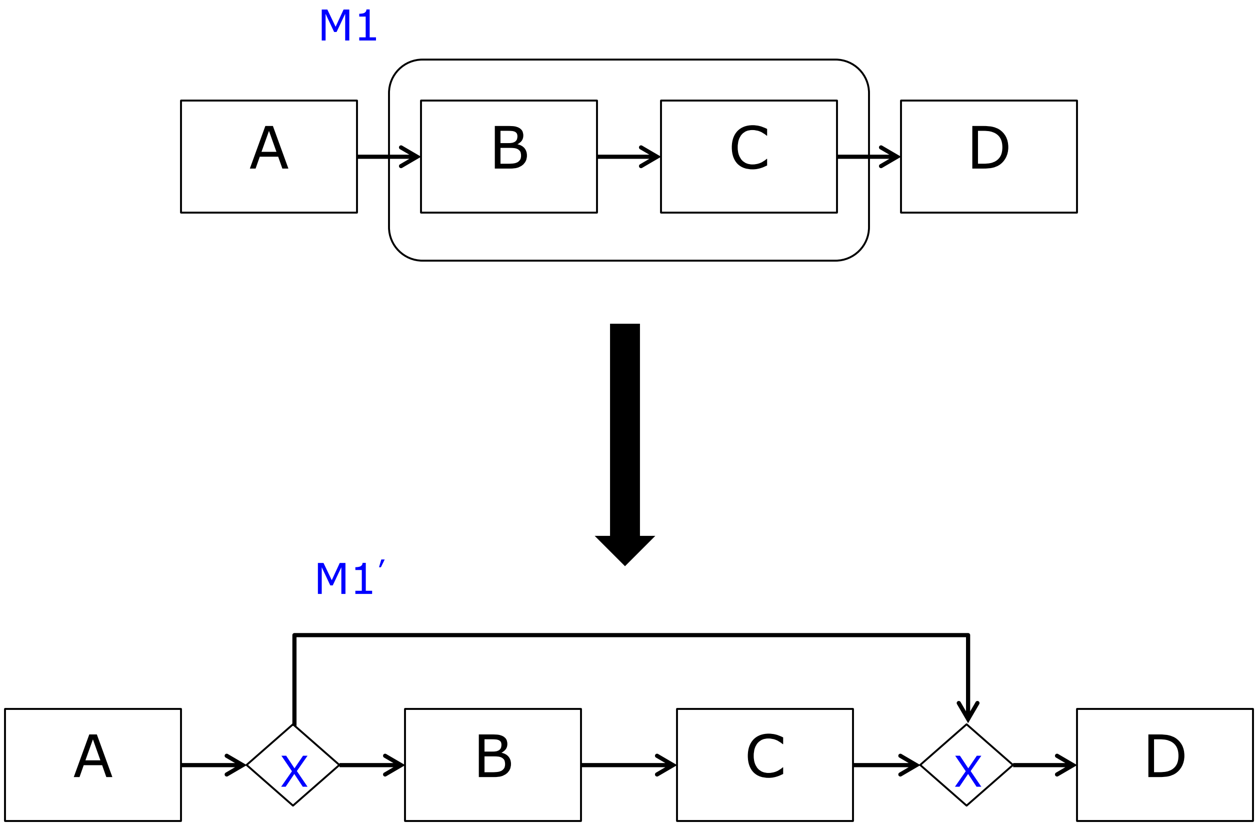


Inline SubProcess

ADAPTATION PATTERNS

- Embed process fragment in conditional branch
  - Can be realized via inserting process fragment and adding & deleting dependency control
- Add control dependency
  - A control edge is added to the model
  - Ensure that use of this pattern meets certain pre- & post-conditions
  - Can be associated to attributes (e.g., transition conditions)
  - Additional parameters might be needed for different types of controls (loop backs, synchronization of parallel activities)

EXAMPLES

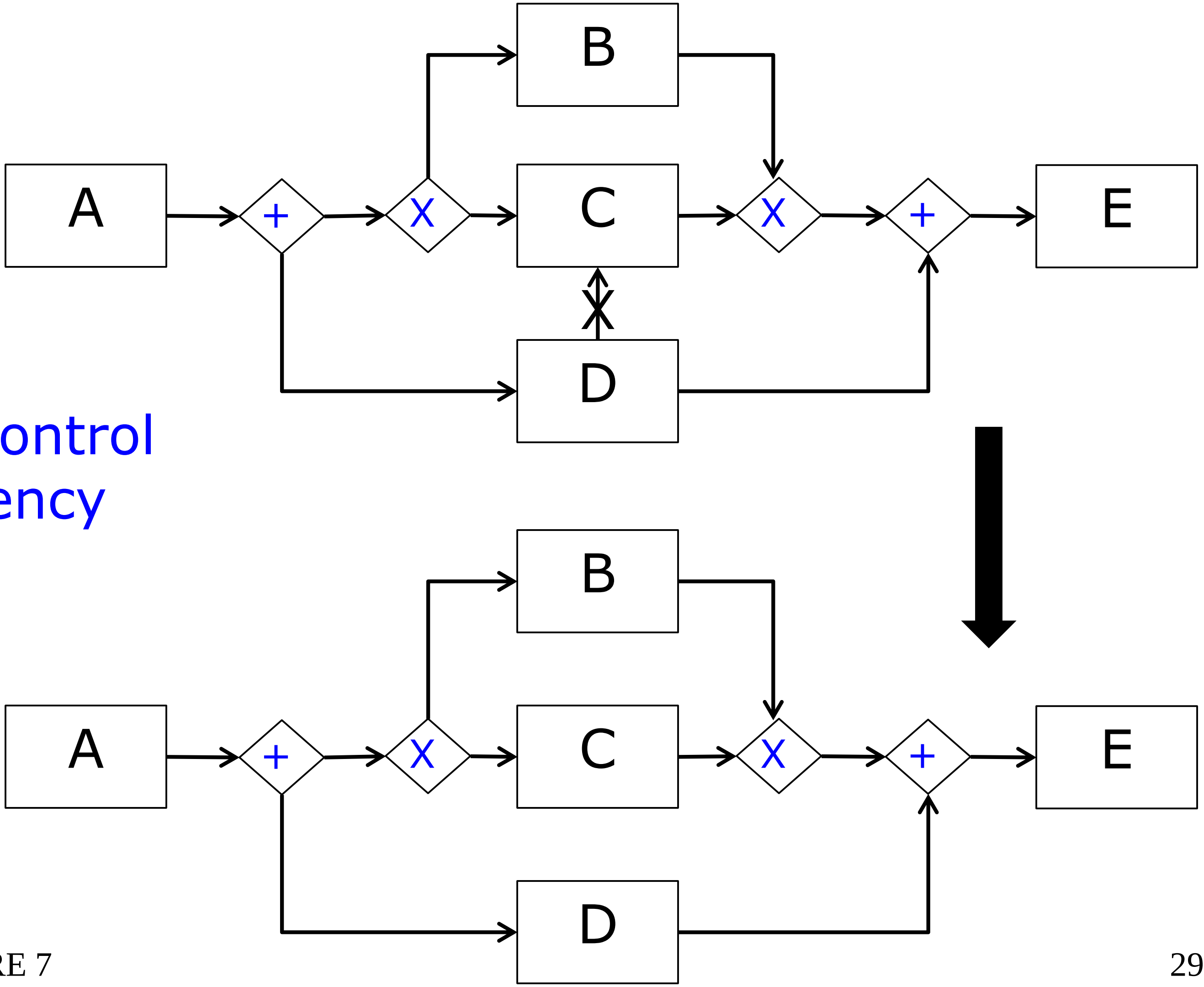


Embed process fragment
in conditional branch

ADAPTATION PATTERNS

- Remove Control Dependency
- Update Condition
  - Update transition conditions
  - Correctness of condition must be checked (e.g., all workflow relevant data elements are present in the process model)
- Copy Process Fragment

EXAMPLES



Remove control dependency

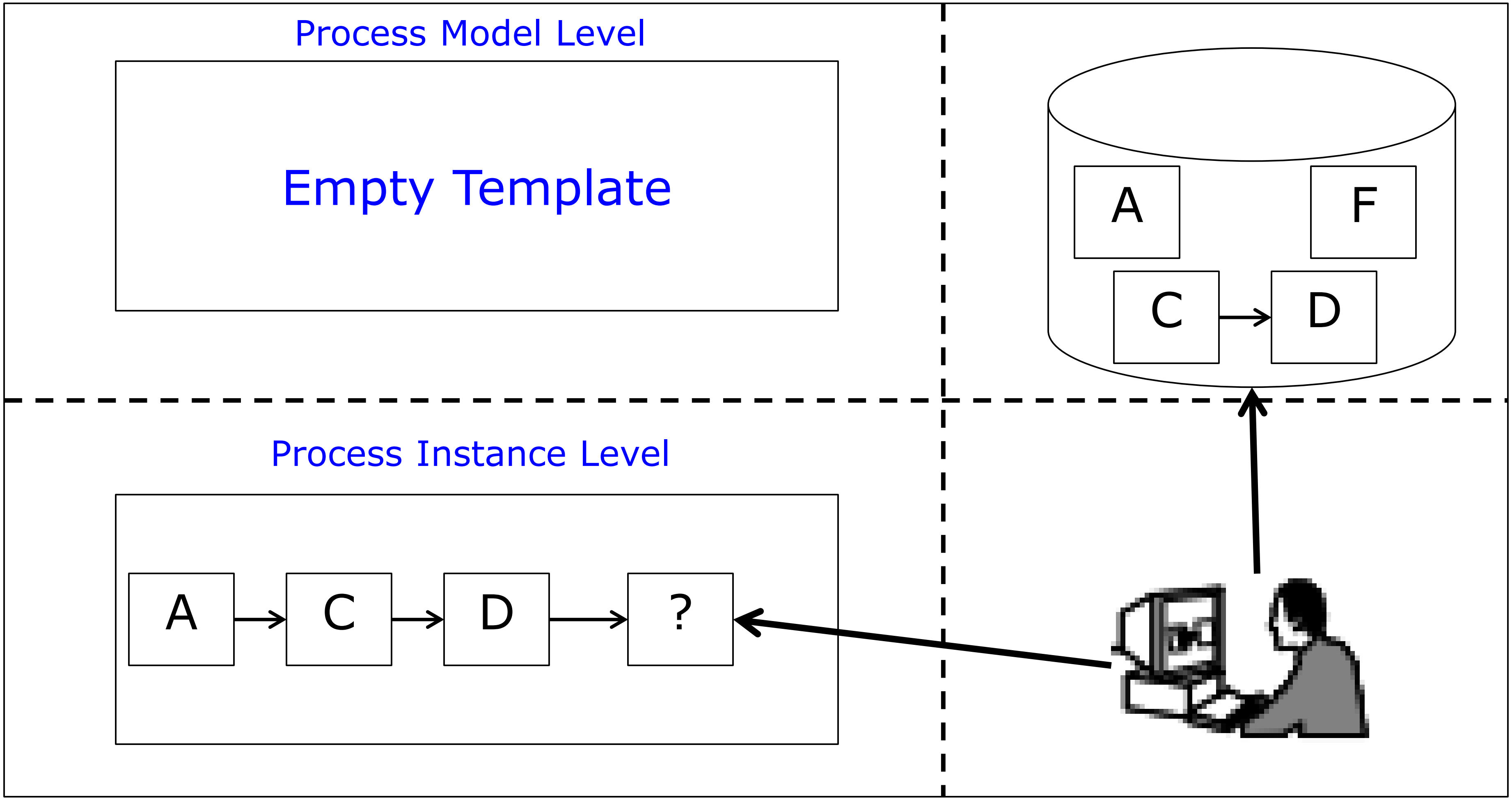
PATTERNS FOR CHANGE IN PREDEFINED REGIONS

- Late Selection of Process Fragments
  - Realize a particular activity based on predefined rules or user decisions or a combination of the first two options
- Late Modelling of Process Fragments
  - Model selected parts of process model at runtime
  - Design choice for selecting process fragments from repository, from specific set or by newly defined activities or process fragments
  - Design choice for applying the same modelling constructs at build-time or by further considering more restrictions
  - Design choice for performing realization: at process instantiation, placeholder activity enablement or certain process state is reached
  - Design choice for working with an empty template or adapting an existing predefined template

PATTERNS FOR CHANGE IN PREDEFINED REGIONS

- Late Composition of Process Fragments
  - On-the-fly composition of fragments from repository by also inserting appropriate control dependencies
  - Interleaved routing workflow pattern as a special case
  - Exact decisions about control flow are deferred at runtime
  - An activity might be executed multiple times (vs special case)
- Multi-instance Activity (also workflow pattern)
  - Decision about how many instances to create can rely on design- or run-time knowledge. Latter should be available before activity execution or when activity is enabled

EXAMPLE – LATE MODELLING



Process Model Level

Empty Template

Process Instance Level

F1.VERSION CONTROL & INSTANCE MIGRATION

- No Version Control:
  - Manual copy of model generated to be modified
  - Current model modification:
    - Running instances are either withdrawn or
    - Remain associated to modified model
  - Can lead to inconsistent states, deadlocks or runtime errors
- Version Control:
  - Running instances remain associated to old model, new instances are mapped to new model
  - Alternative: controlled migration of selected instances to new model
  - Alternative: uncontrolled migration -> inconsistencies & errors

F2. SUPPORT FOR INSTANCE-SPECIFIC CHANGES

- Unplanned changes at instance level are addressed through high-level patterns or low-level primitives
- Uncertainty handled by keeping process parts unspecified until runtime
- Instance changes are permanent or temporary (valid for a certain time period – e.g., current iteration of a loop)

F3. CORRECTNESS OF CHANGE

- To avoid runtime errors, different criteria are introduced for moving instances to new model to reassure compliance
- Additionally, formal constraints depending of the respective formalism must be considered

F4. TRACEABILITY & ANALYSIS OF CHANGES

- Change patterns or primitives must be entered into a change log
- Change analysis & mining become easier when high-level information is stored -> continuous process improvement
- An execution log is enough for traceability concerning changes in particular process regions
- Logs can be enriched with semantic information covering the reasons & context of changes

F5. ACCESS CONTROL FOR CHANGES

- To avoid security issues wrt. the misuse of the change capabilities, authorization should be enabled:
  - Only particular users can change process model or instances
  - More coarse-grained granularity maps to authorizing single change patterns.
  - Authorizations can also depend on the object to change (e.g., object type – model vs instance or sets of process models attributed to particular process designers)

F6. CHANGE REUSE

- Change reuse must be exploited to avoid spending time in finding the same solution to the same problem
- This can be supported by annotating changes with contextual information and storing them in logs
  - Contextual information maps to matching similar situations
    - User is presented only with solutions to those situations
  - For predefined region changes, historical cases must be presented to users & frequent, re-occurring pattern realizations must be stored as templates

F7. CHANGE CONCURRENCY CONTROL

- Concurrent changes at instance level concerning process structure & state must be dealt with
  - Can lead to errors or inconsistencies (e.g. violating state constraints) if performed in an uncontrolled manner
- Ways to address:
  - Forbid concurrent changes (strict due to long-term locks required)
  - Allow concurrent changes on structure or state
    - Pessimistically or optimistically
- Could also need to address concurrent changes both at model & instance level

CHANGE SUPPORT FEATURES

Change Support Features			
Change Support Feature	Scope	Change Support Feature	Scope
F1. Schema Evolution, Version Control & Instance Migration	M	F3. Correctness of Changes	M+I
		F4. Traceability & Analysis	M+I
No version control – old model is overwritten		1. Traceability of Changes	
1. Running instances cancelled		2. Annotation of Changes	
2. Running instances remain		3. Change Mining	
Version control		F5. Access Control for Changes	M+I
3. Co-existence of old/new instances, no instance migration		1. Changes are restricted to authorized users	
4. Uncontrolled migration of all instances		2. Application of single change patterns is restricted	
5. Controlled migration of compliant instances		3. Authorizations depend on object to change	

CHANGE SUPPORT FEATURES

Change Support Features			
Change Support Feature	Scope	Change Support Feature	Scope
F2. Support for Instance-Specific Changes	I	F6. Change Reuse	I
		F7. Change Concurrency Control	M+I

1. Unplanned changes
   a. Temporary
   b. Permanent
2. Preplanned changes
   a. Temporary
   b. Permanent

1. Uncontrolled concurrent changes
2. Concurrent changes prohibited
3. Concurrent changes of an instance’s structure & state
4. Concurrent Changes at instance & model level

RECOMMENDED READING

- M.H. Schonenberg, R.S. Mans, N.C. Russell, N.A. Mulyar and W.M.P. van der Aalst. Towards a Taxonomy of Process Flexibility (Extended Version). Available at: http://bpmcenter.org/wp-content/uploads/reports/2007/BPM-07-11.pdf
- Barbara Weber, Stefanie Rinderle-Ma and Manfred Reichert. Change Support in Process-Aware Information Systems – A Pattern-Based Analysis. Available at: http://eprints.eemcs.utwente.nl/11331/01/main.pdf
- http://theprocessconsultant.com/process-improvement-flexibility/