A Novel Graph Based Approach for Automatic Composition of High Quality Grid Workflows

Jun Qin, Thomas Fahringer, Radu Prodan

University of Innsbruck, Austria

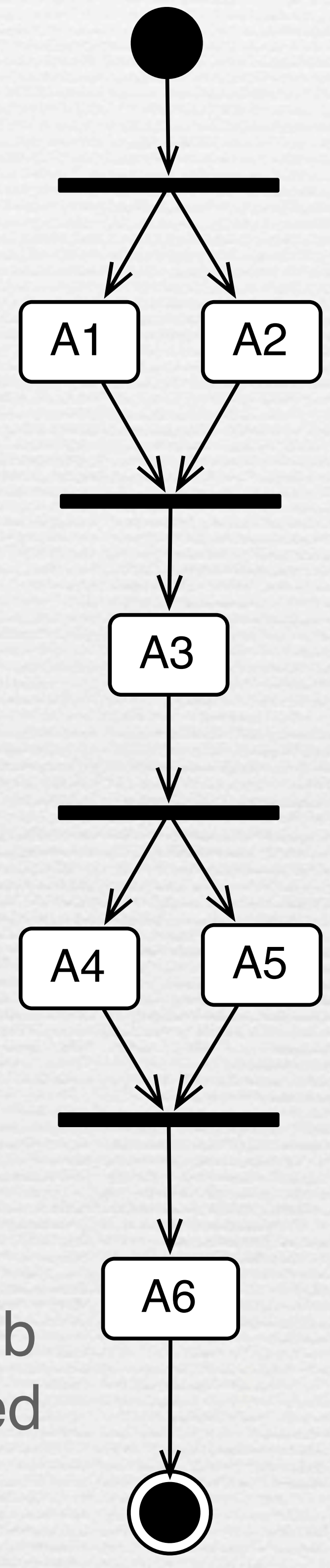
HPDC 2009 *Munich, Germany* *June 11-13, 2009*

Outline

- Introduction: Grid workflow & composition
- Comparison with existing work
- Background: ASKALON, AGWL
- Formal definition of the Grid workflow composition problem
- Grid workflow composition algorithm
    - ADD graph and its creation
    - Workflow extraction
    - Workflow optimization
    - Complexity analysis
    - Composition of Grid workflows with branches and loops
- Experimental results
- Summary and Future work

Introduction

- Grid workflow is an important programming model for the Grid
  - a Grid workflow consists of a set of activities, and
  - a set of control flow/data flow dependences
- Grid workflow composition
  - selection of workflow activities
  - specification of control flow and data flow dependences
  - time consuming and error prone process, optimization takes longer time
- Abstract Grid workflow
  - using abstract activities reduces the user effort to select activities
  - the selection among hundreds abstract activities is still challenging
- Automatic Grid workflow composition
  - different from that in Business Process Management, Semantic Web Services, and requires high quality: portable, fault tolerant, optimized



Introduction

- Grid workflow is an important programming model for the Grid
  - a Grid workflow consists of a set of activities, and
  - a set of control flow/data flow dependences
- Grid workflow composition
  - selection of workflow activities

A novel ADD graph based approach for automatic composition of high quality Grid workflows

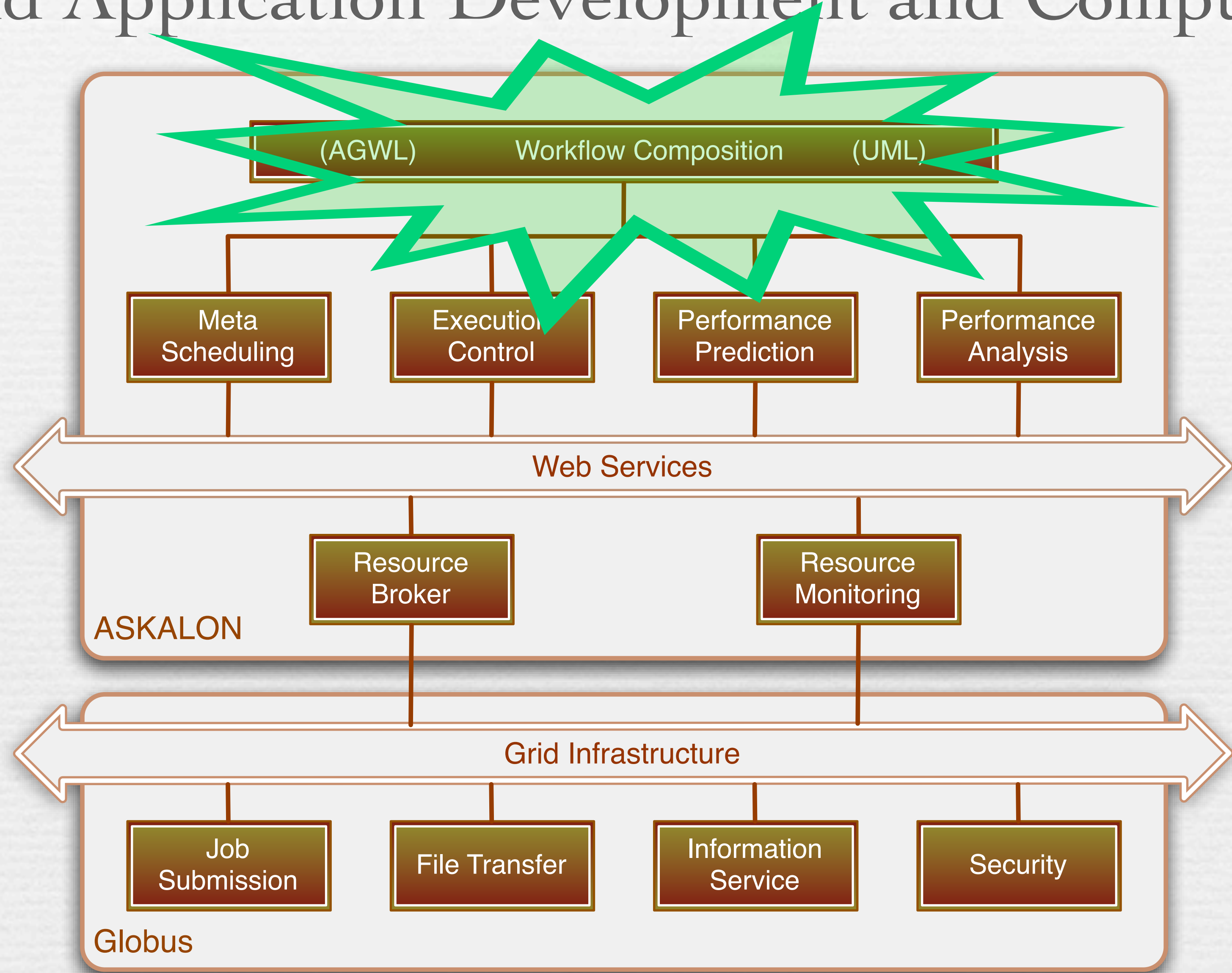
- Abstract Grid workflow
  - using abstract activities reduces the user effort to select activities
  - the selection among hundreds abstract activities is still challenging
- Automatic Grid workflow composition
  - different from that in Business Process Management, Semantic Web Services, and requires high quality: portable, fault tolerant, optimized

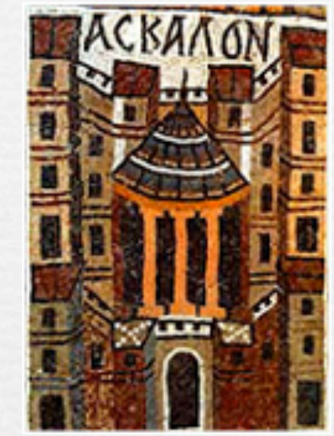
Existing Work

- Existing work suffers from one of the following drawbacks:
  - Limiting to specific workflow notation systems such as Petri Nets
  - Focusing only on simple constructs like DAG
  - Cannot handle or do not consider alternative control flows
  - No workflow optimization
  - Only generating workflow instances from workflow templates
  - Assuming workflow tasks has ranks
- Our approach goes beyond existing work:
  - A general solution
  - Generation of alternative workflows, thus support fault tolerance
  - Considering workflow optimization
  - Generation of workflows with branches and parallel/sequential loops

ASKALON

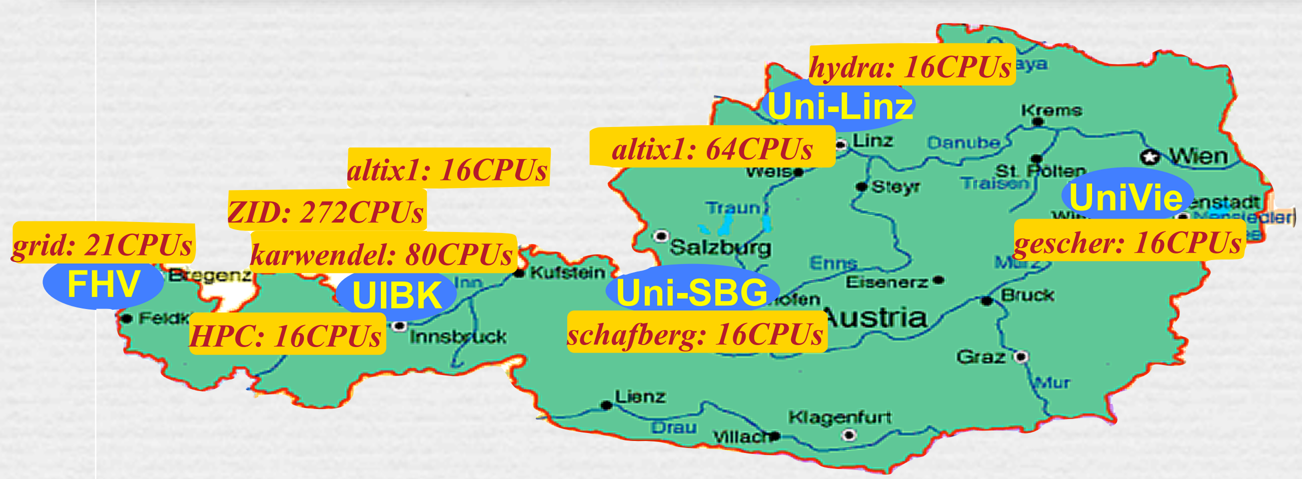
Grid Application Development and Computing Environment





Developed by the Univ. of Innsbruck, Austria

www.askalon.org

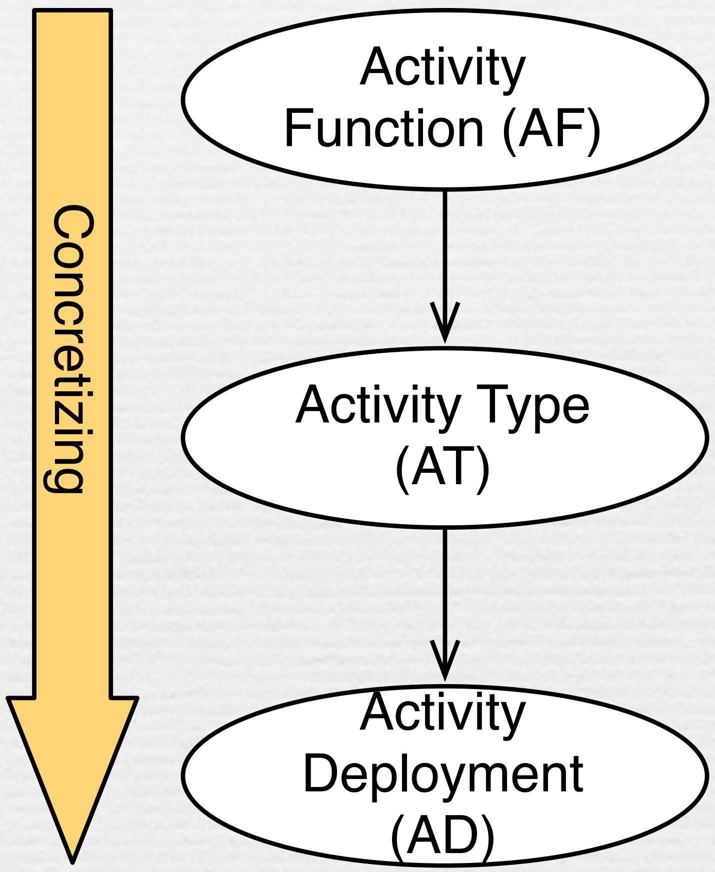


AUSTRIAN GRID

Abstract Grid Workflow Language (AGWL)

- XML-based language for describing scientific Grid workflows at a high level of abstraction
- A rich set of control flow constructs
  - *sequence, parallel, if, switch, while, doWhile, for, forEach, parallelFor, parallelForEach, dag, alternative*
- data flow links: source data port ➔ sink data port
  - in case of multiple sink data ports, each receives a data copy
  - data collection
- properties and constraints
- the main interface to ASKALON

Abstraction in AGWL



abstract:
semantic description

abstract:
syntactic description

concrete:
full description required for execution

data semantics

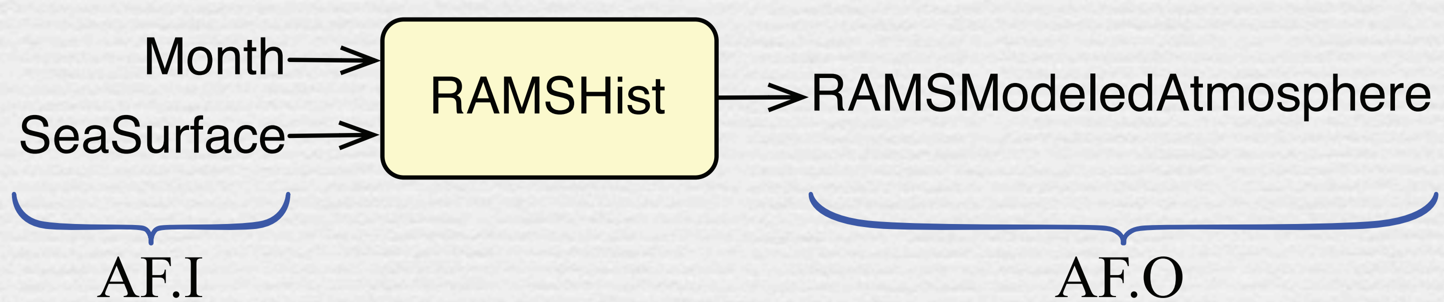
the meaning of the data, referring to a concept/class in an ontology

Example:
Month, EvenMonth,
ModelParameter, RAMSModelParameter

data representation

storage related info of the data, such as storage type, content type, etc.

Example:
File in FileSystem,
integer in Memory,
string in Memory



Jun Qin, Thomas Fahringer, A Novel Domain Oriented Approach for Scientific Grid Workflow Composition, SC'08.

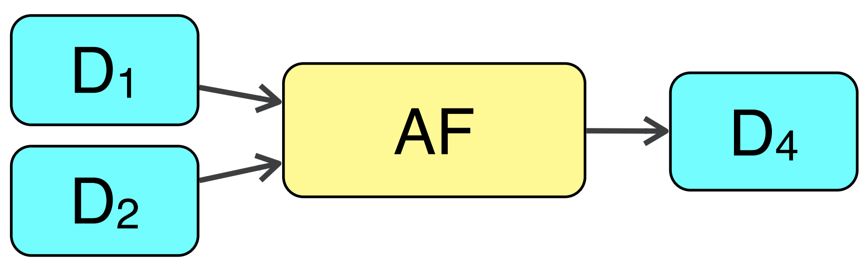
Definition of the Grid workflow composition problem

- STRIPS (STanford Research Institute Problem Solver)
  - initial state, goal state, actions
- Apply STRIPS into Grid workflow composition
  - state: a set of Data Classes, indicating the availability of data
    - initial state: user provided data which can be consumed by activities
    - goal state: user required data which must be produced by the composed workflow
  - action: Activity Function

Definition of the Grid workflow composition problem

state$_1$: D_1, $D_{2.1}$, D_3, $D_{2.1}$ is *subclass* of D_2 according to ontology

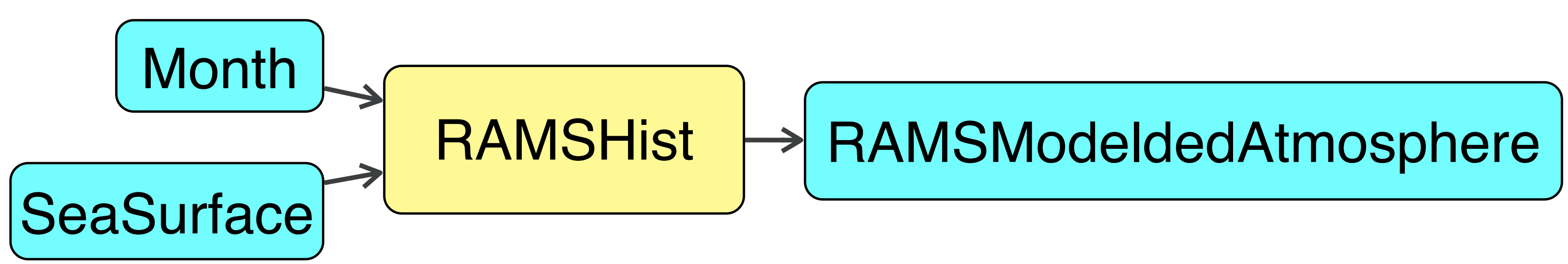
state$_1$ ⊨AF.I
(entails)



state$_2$: D_1, $D_{2.1}$, D_3, D_4

state$_1$: Month, SeaSurface

state$_1$ ⊨ AF.I



state$_2$: Month, SeaSurface, RAMSModeldedAtmosphere

Definition of the Grid workflow composition problem

- The Grid workflow composition problem can be defined by the function:

$$f : (s_{init}, s_{goal}, \mathcal{AF}) \rightarrow w$$

  - s_{init}: the initial state
  - s_{goal}: the goal state
  - $\mathcal{AF}$: the set of AFs among which some AFs will be selected for the composition of the Grid workflow w
  - w is a DAG of AFs connected by control flow edges
    1) $s_{init} \models$ AF.I for any AF which has no incoming control flow edges
    2) $s_{init} \cup (\bigcup_{AF' \in \mathcal{AF}'}$ AF.O$) \models$ AF.I for any AF which has predecessors
    3) $s_{init} \cup (\bigcup$AF.O$) \models s_{goal}$

A Simulated Domain

- Data Classes (DC)

D_0 D_1 D_2 D_3 D_4 D_5

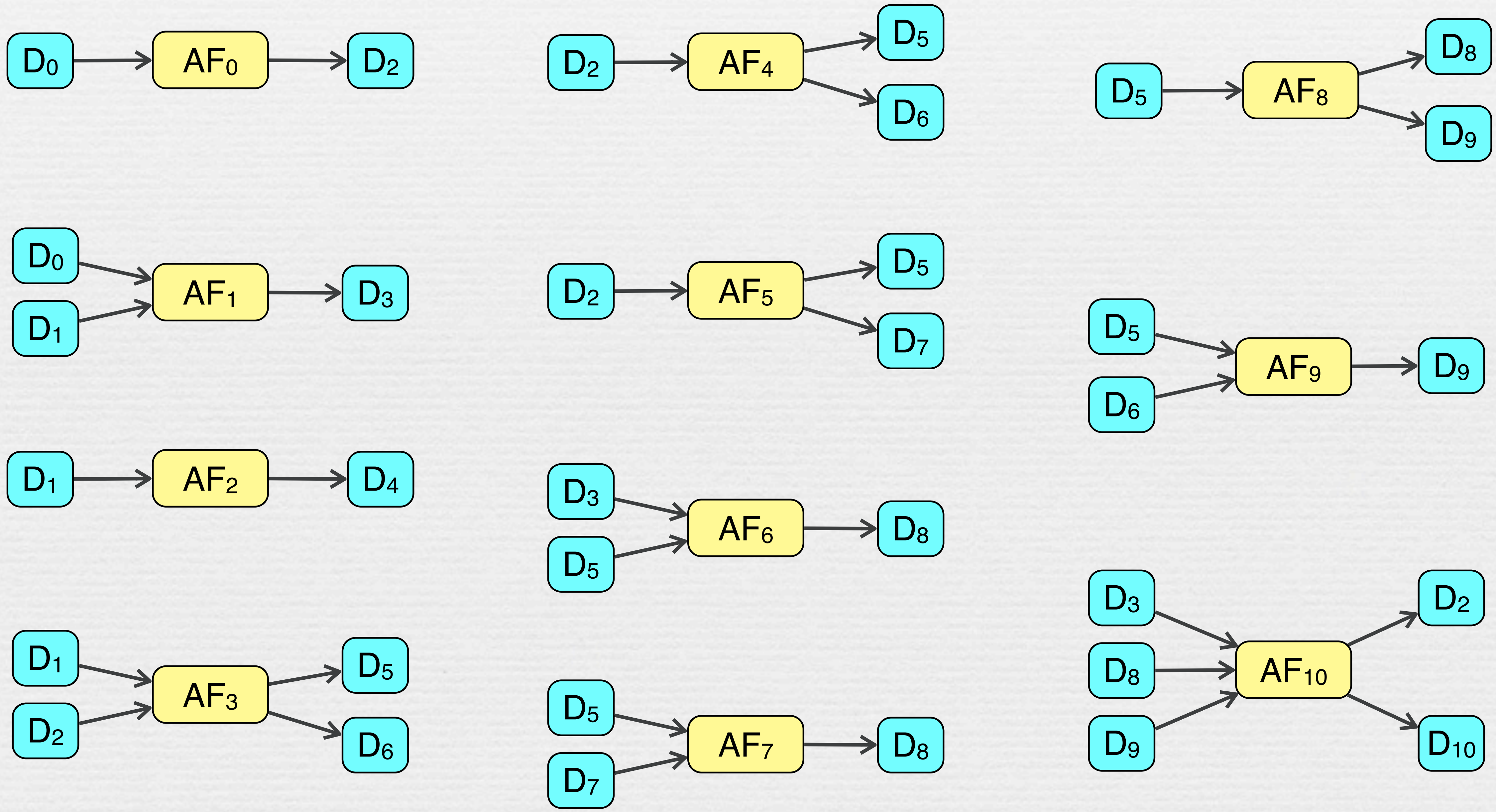
D_6 D_7 D_8 D_9 D_{10}

- Activity Functions (AF)

AF_0 AF_1 AF_2 AF_3 AF_4 AF_5

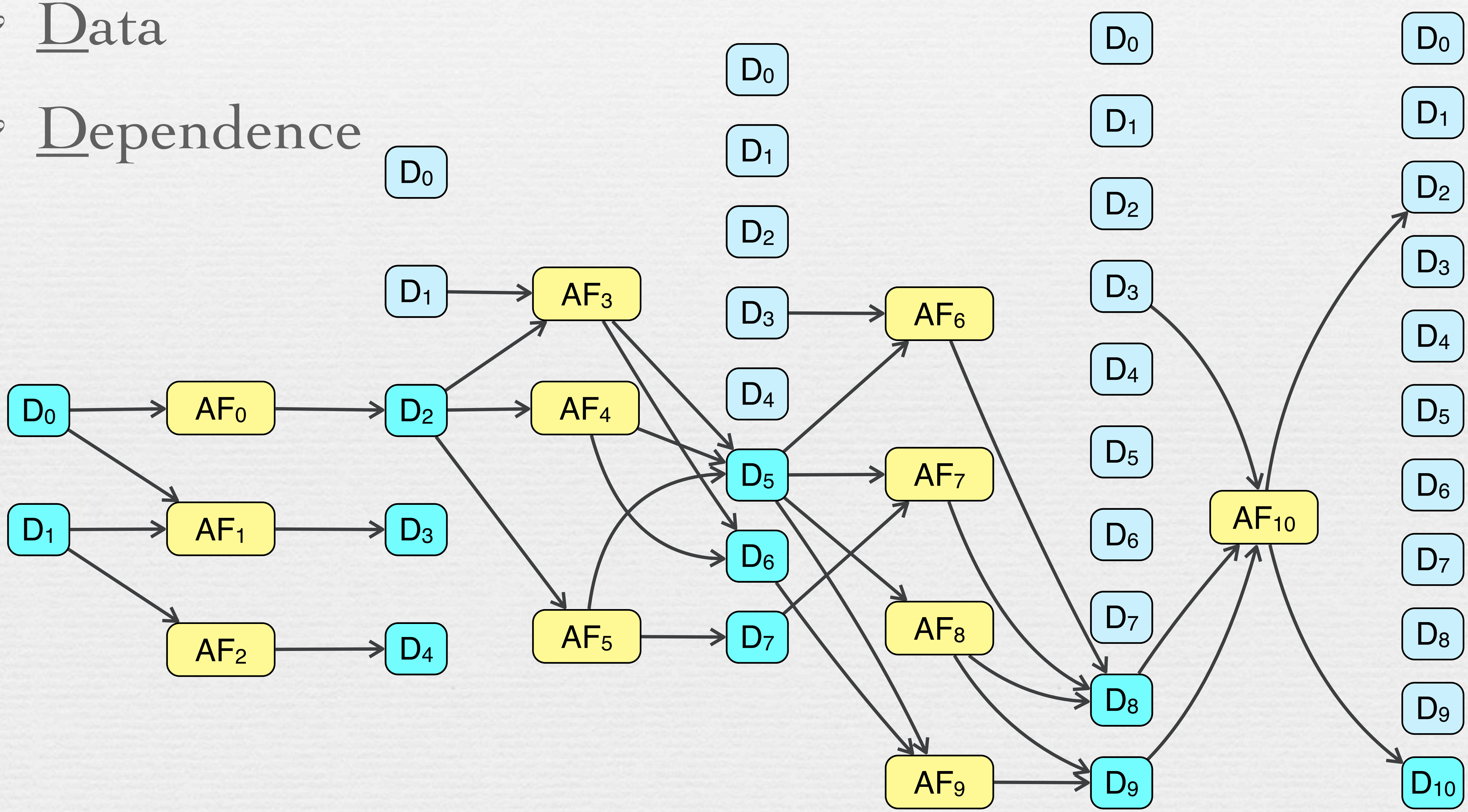
AF_6 AF_7 AF_8 AF_9 AF_{10}

A Simulated Domain



ADD Graph

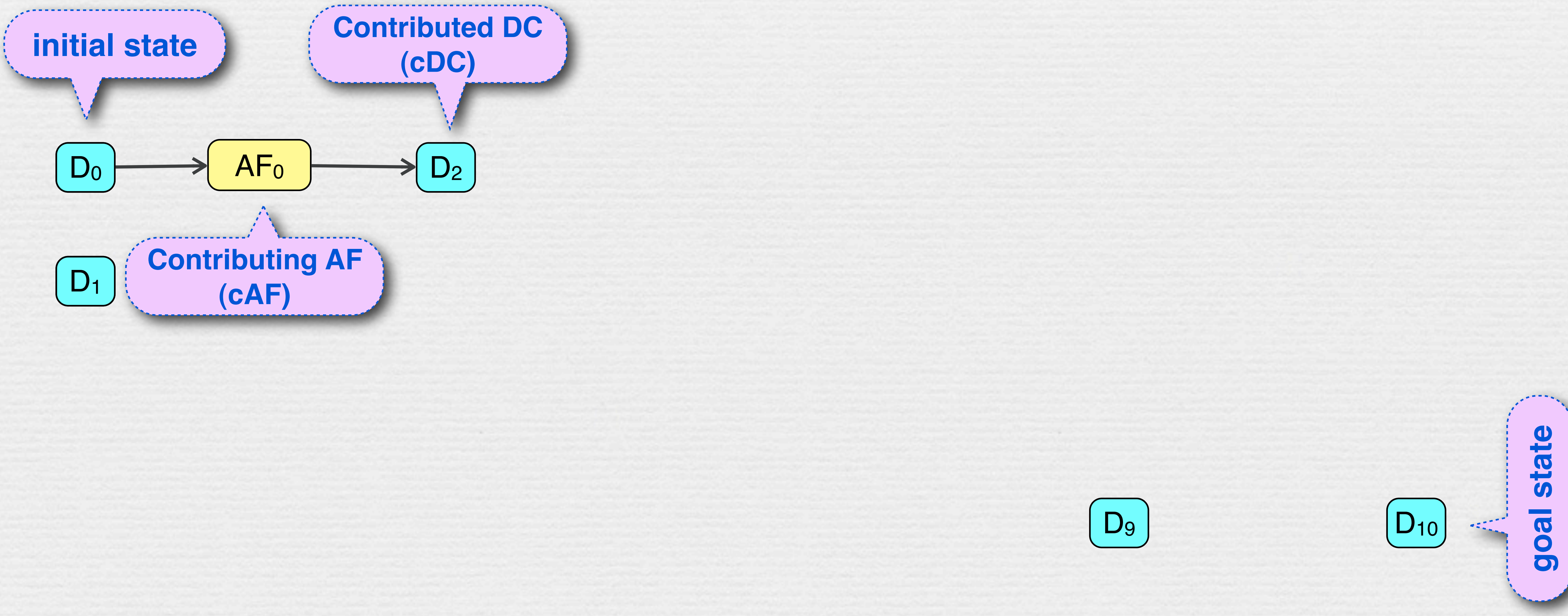
- Activity Function
- Data
- Dependence



ADD Graph Creation

$s_{init} = \{D_0, D_1\}$

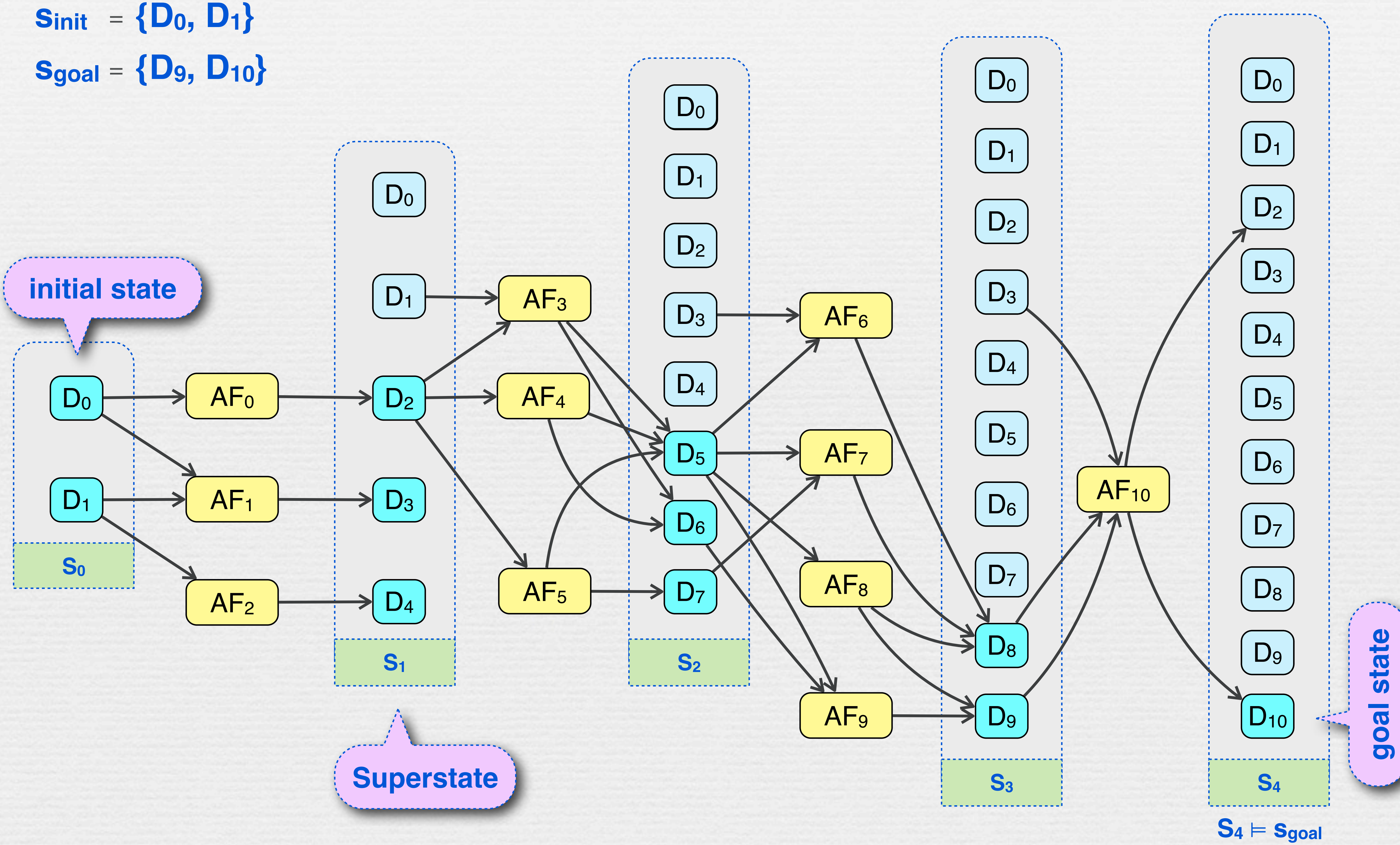
$s_{goal} = \{D_9, D_{10}\}$



ADD Graph Creation

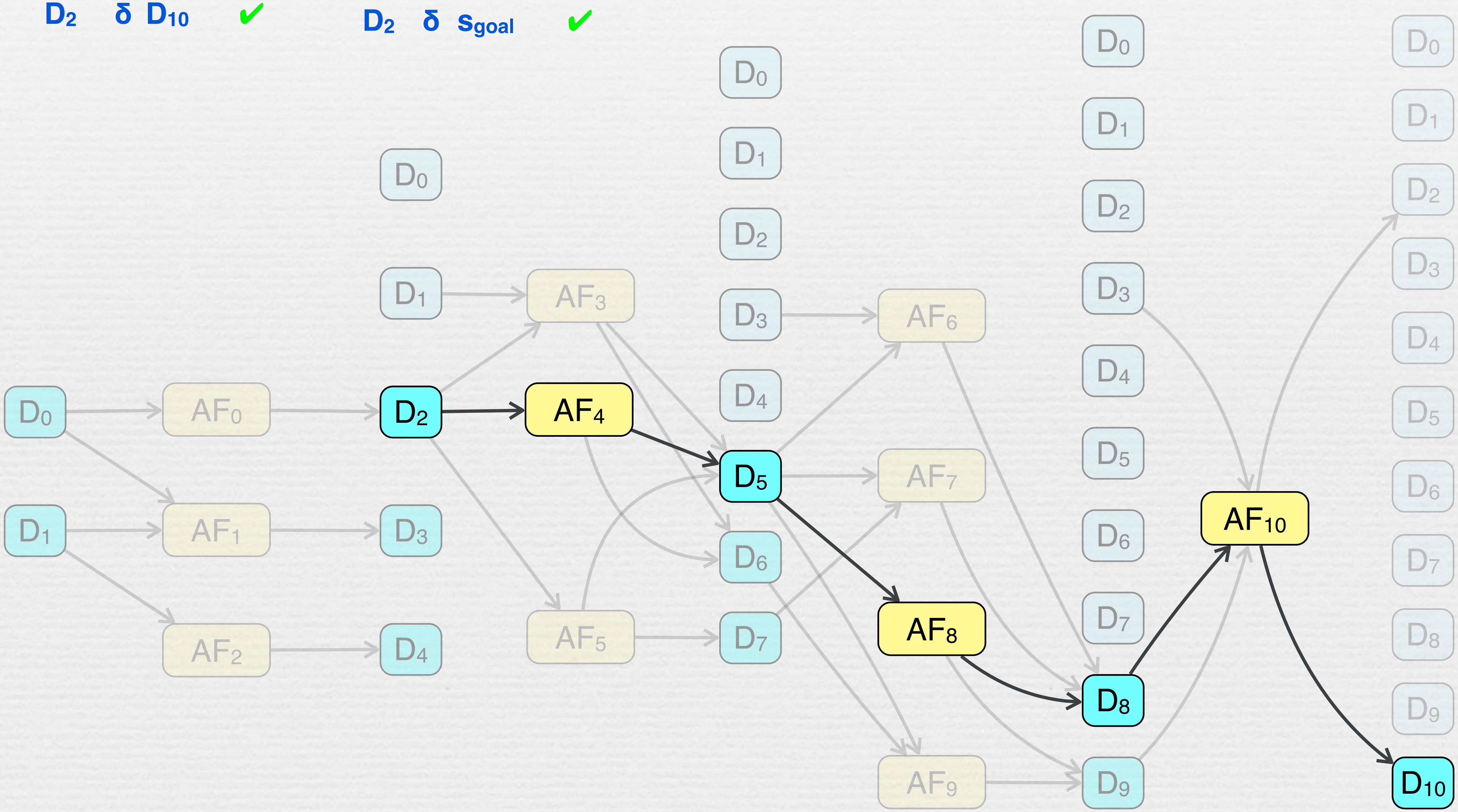
$s_{init} = \{D_0, D_1\}$

$s_{goal} = \{D_9, D_{10}\}$



Notation: dependence

$D_2 \;\delta\; D_{10}$ ✔ $D_2 \;\delta\; s_{goal}$ ✔



Dependence: $N_i \;\delta\; N_j$

Notation: dependence

$D_2 \ \delta \ D_{10}$ ✔

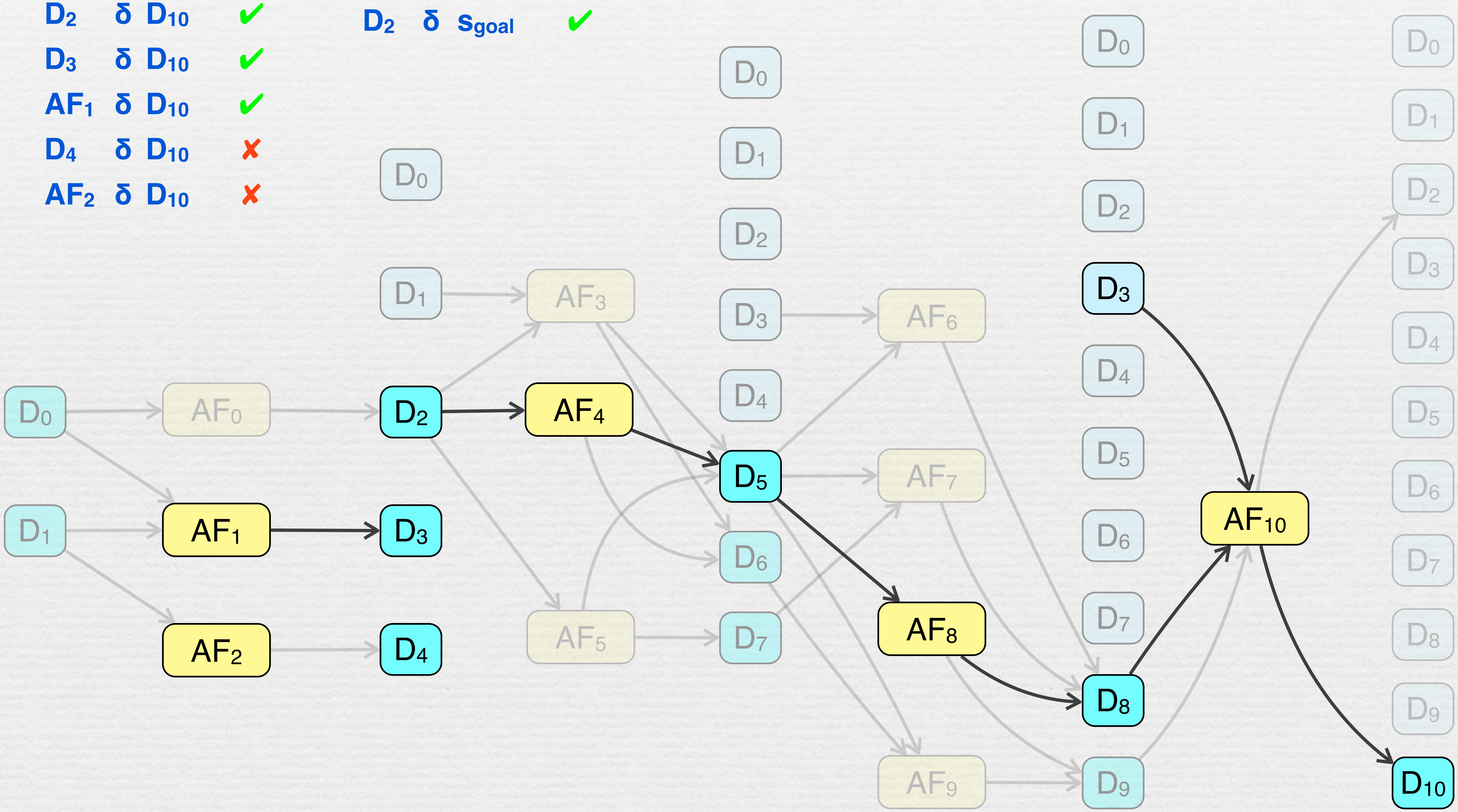
$D_3 \ \delta \ D_{10}$ ✔

$AF_1 \ \delta \ D_{10}$ ✔

$D_4 \ \delta \ D_{10}$ ✘

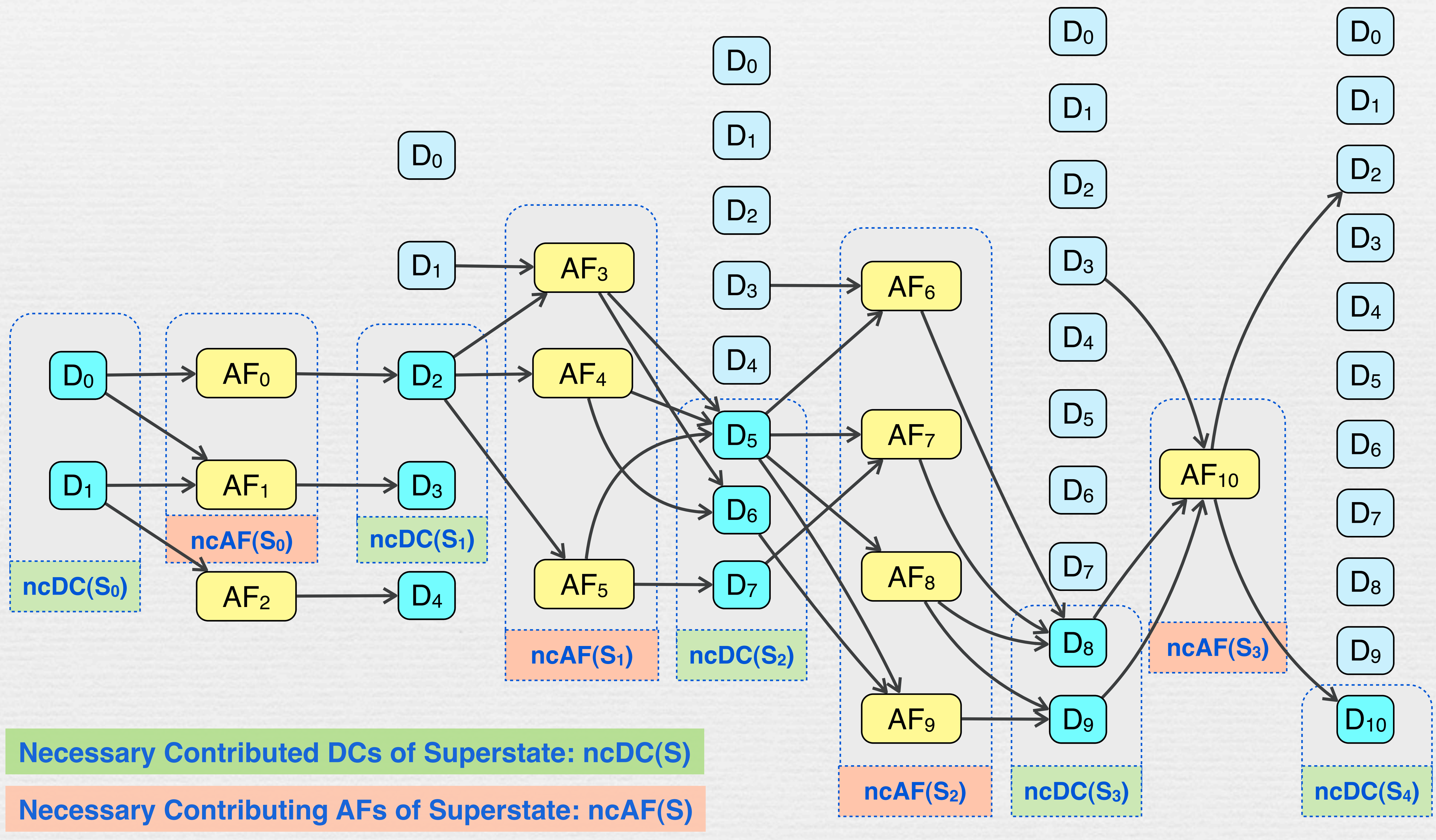
$AF_2 \ \delta \ D_{10}$ ✘

$D_2 \ \delta \ s_{goal}$ ✔



Dependence: $N_i \ \delta \ N_j$

Notation: ncDC(S), ncAF(S)



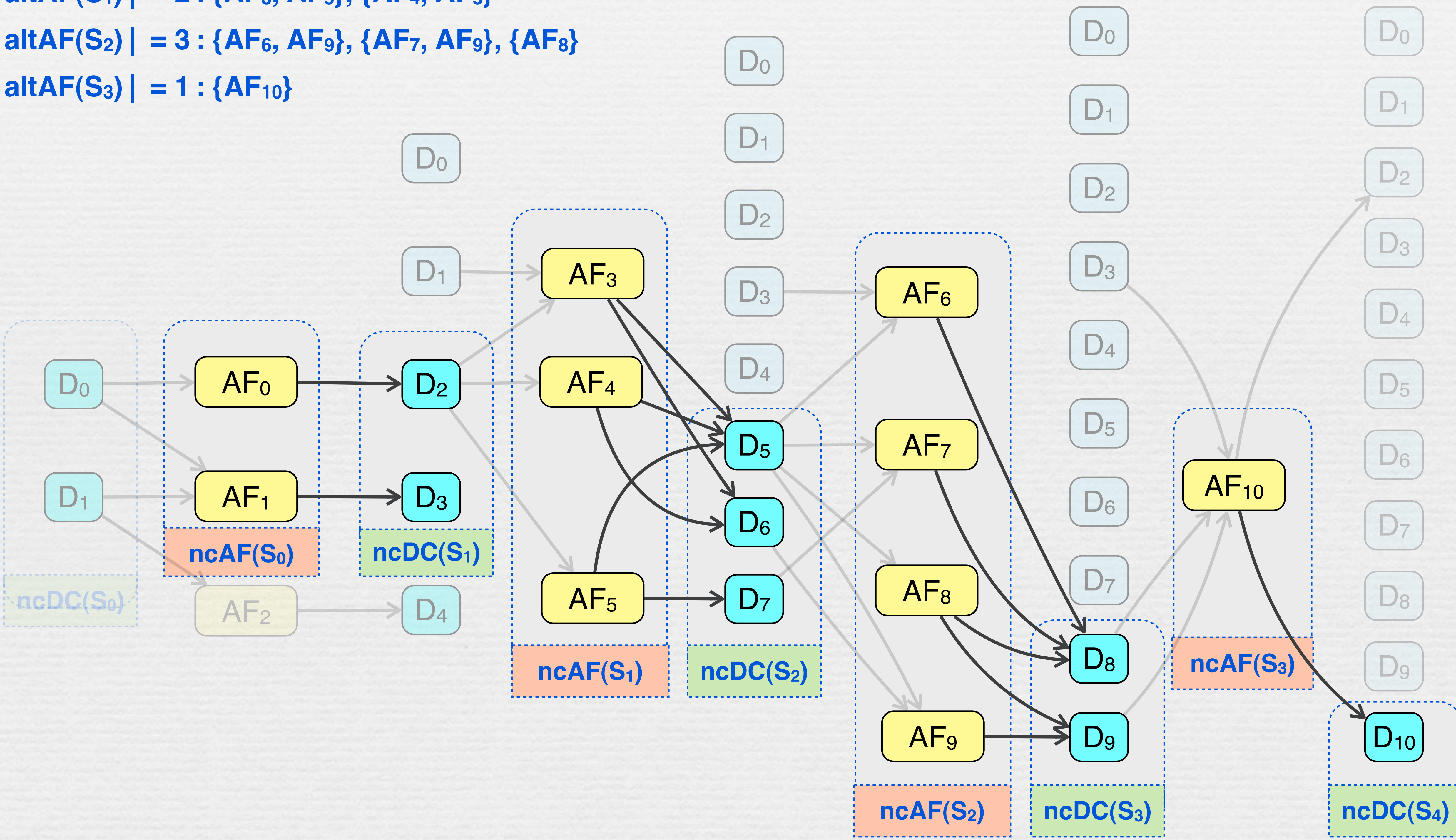
Notation: altAF(S)

| altAF(S_0) | = 1 : {AF_0, AF_1}

| altAF(S_1) | = 2 : {AF_3, AF_5}, {AF_4, AF_5}

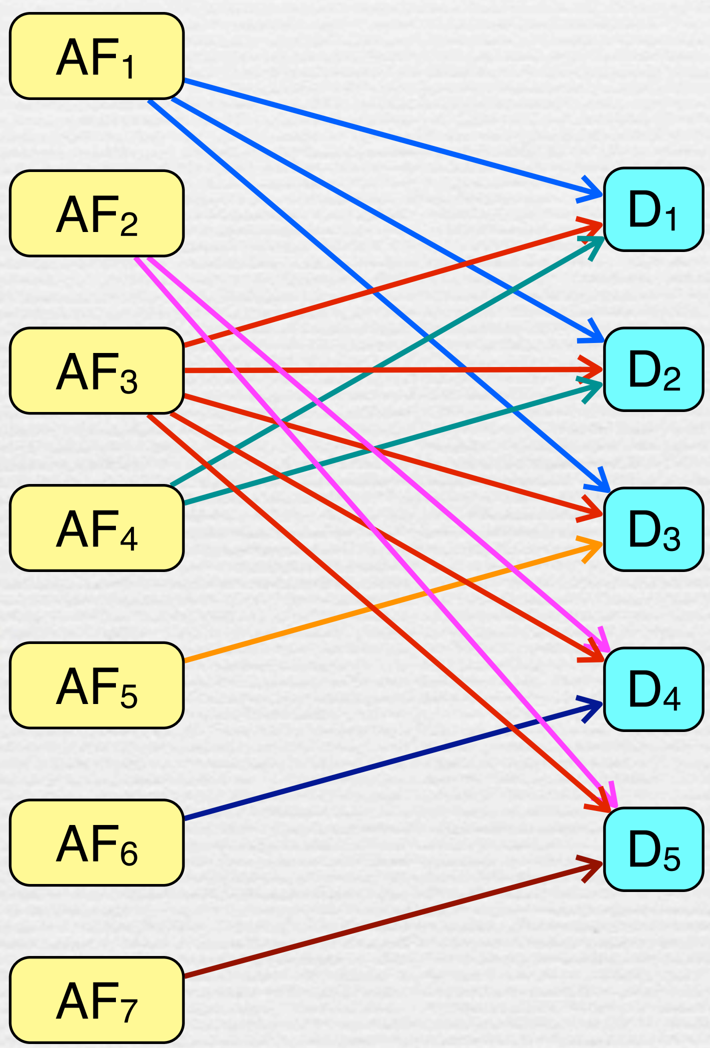
| altAF(S_2) | = 3 : {AF_6, AF_9}, {AF_7, AF_9}, {AF_8}

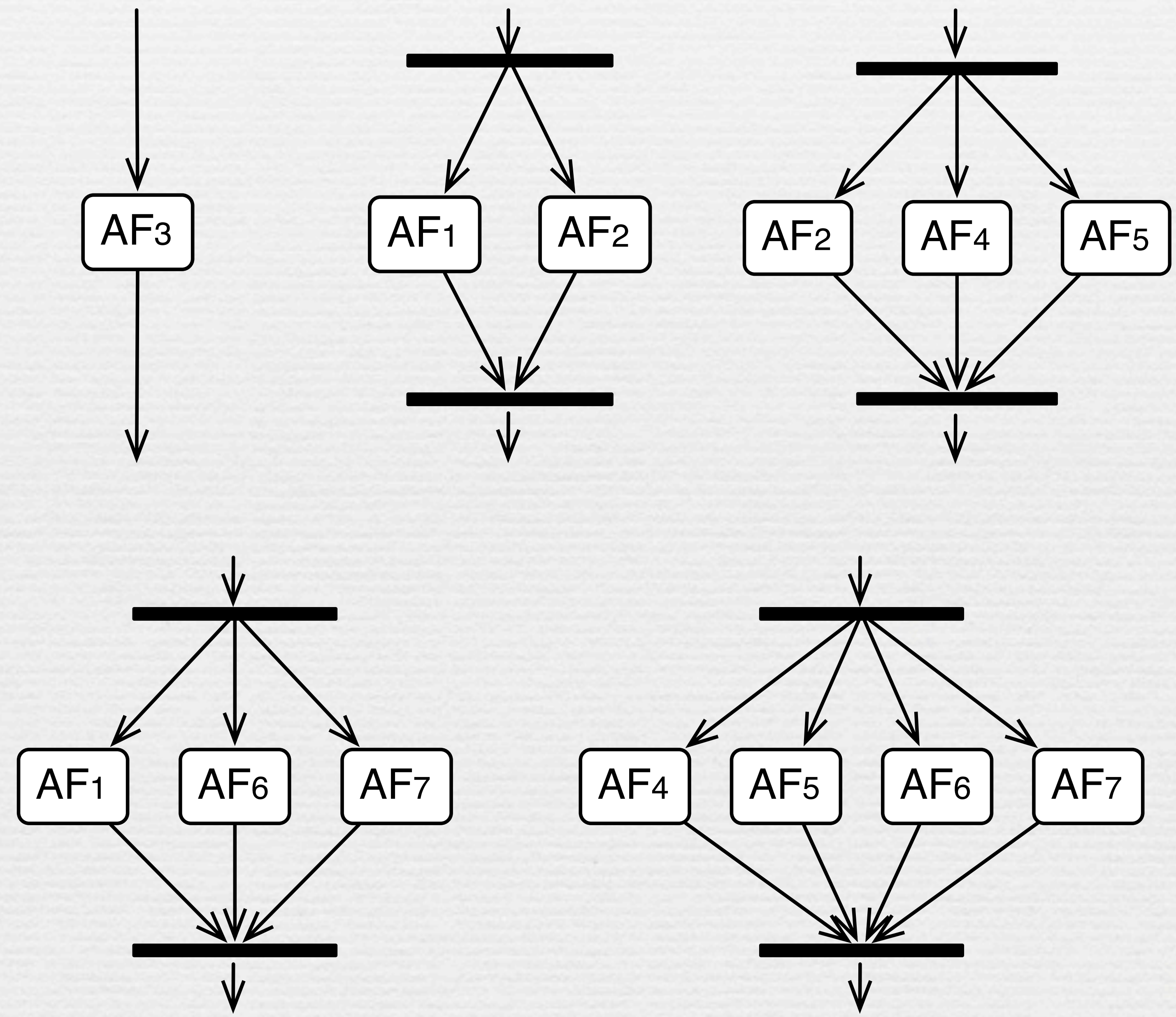
| altAF(S_3) | = 1 : {AF_{10}}



Alternative AF Combination of Superstate: altAF(S)

Calculation of altAF(S)





| altAF(S) | = 5 : {AF_3}, {AF_1, AF_2}, {AF_2, AF_4, AF_5}, {AF_1, AF_6, AF_7}, {AF_4, AF_5, AF_6, AF_7}

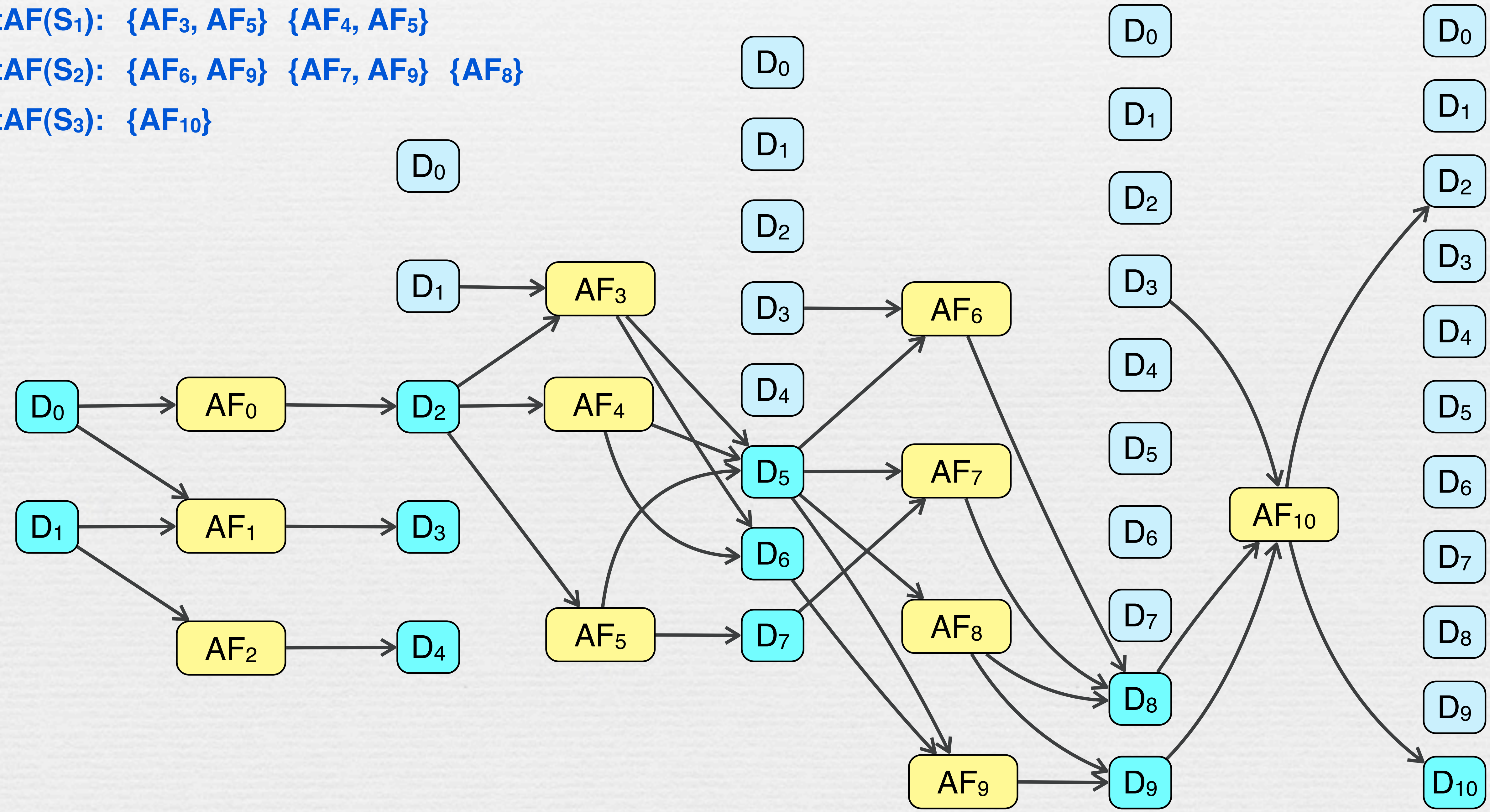
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} {AF_4, AF_5}

altAF(S_2): {AF_6, AF_9} {AF_7, AF_9} {AF_8}

altAF(S_3): {AF_{10}}



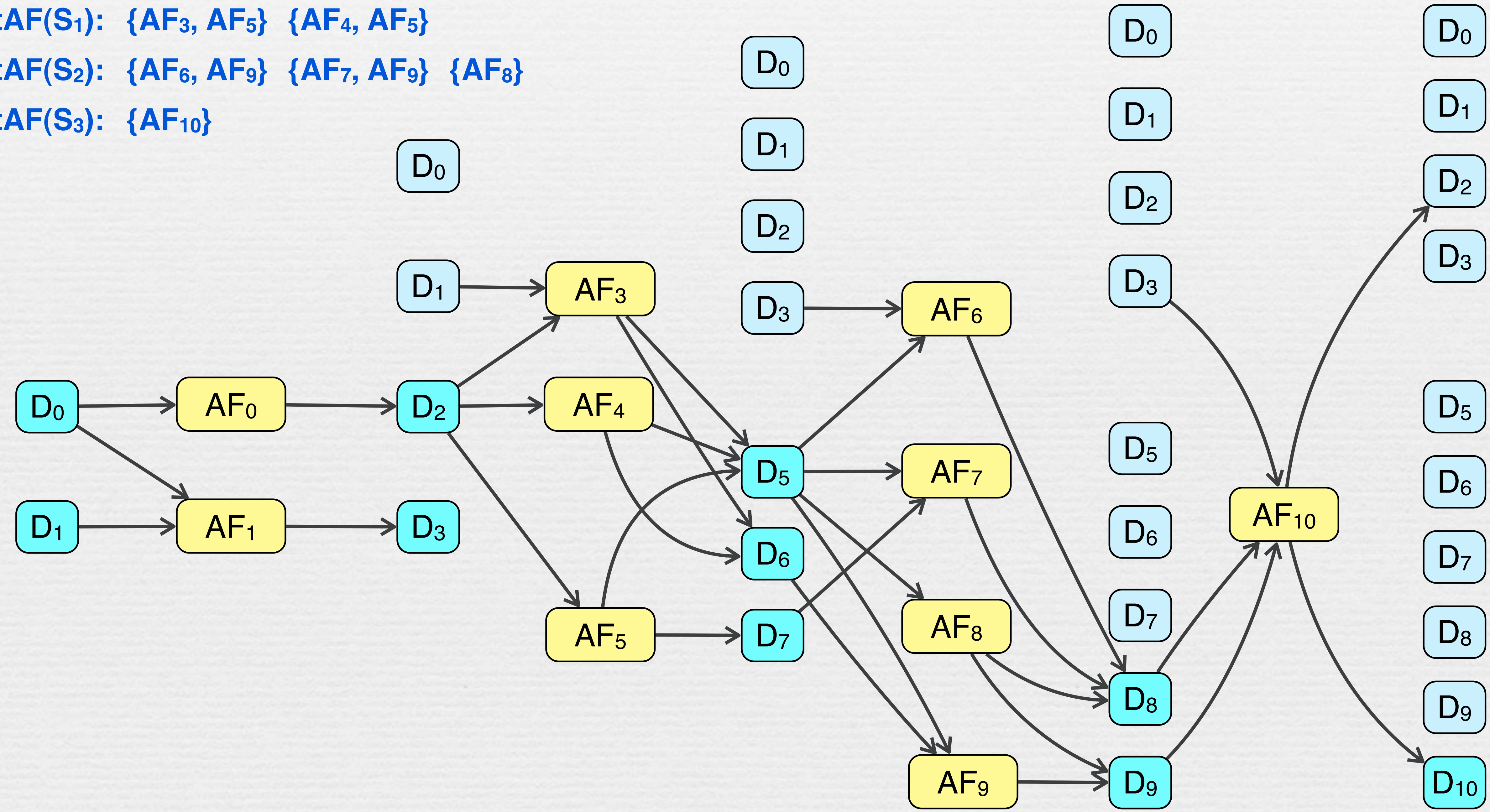
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} {AF_4, AF_5}

altAF(S_2): {AF_6, AF_9} {AF_7, AF_9} {AF_8}

altAF(S_3): {AF_{10}}



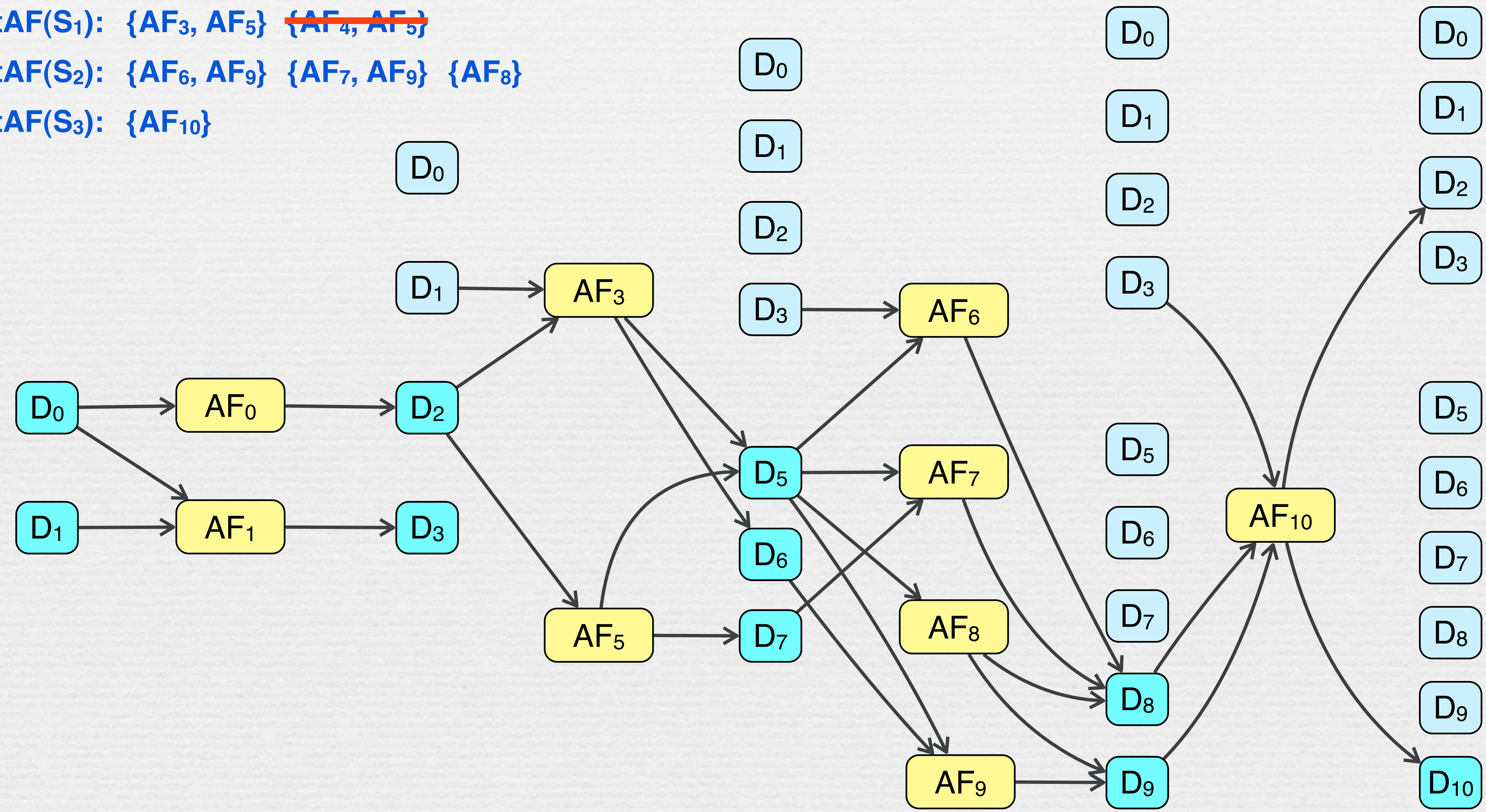
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} ~~{AF_4, AF_5}~~

altAF(S_2): {AF_6, AF_9} {AF_7, AF_9} {AF_8}

altAF(S_3): {AF_{10}}



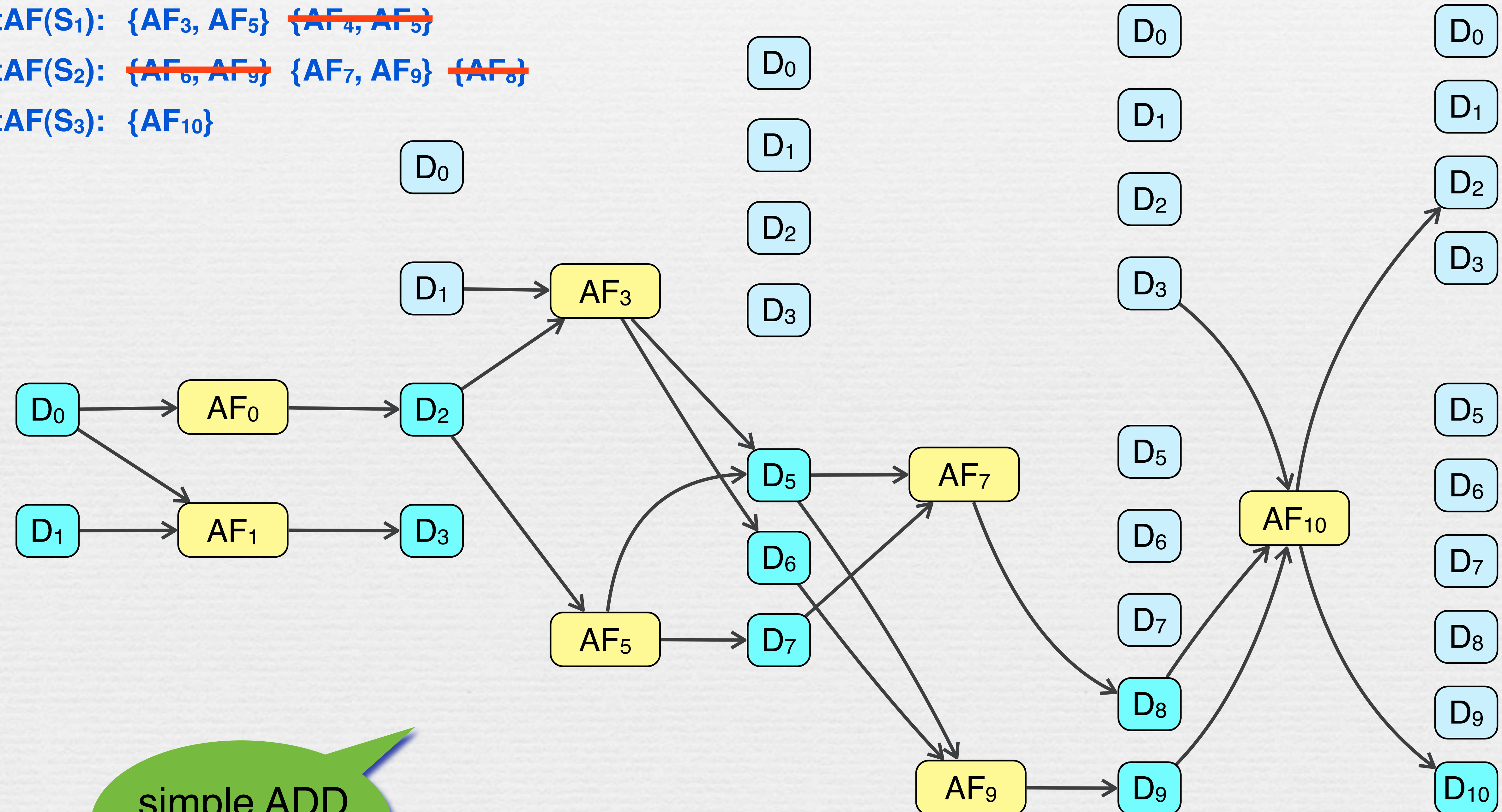
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} ~~{AF_4, AF_5}~~

altAF(S_2): ~~{AF_6, AF_9}~~ {AF_7, AF_9} ~~{AF_8}~~

altAF(S_3): {AF_{10}}



simple ADD Graph

(1) for all i ∈ [0,n): | altAF(S_i) | = 1

(2) all AFs/DCs are necessary

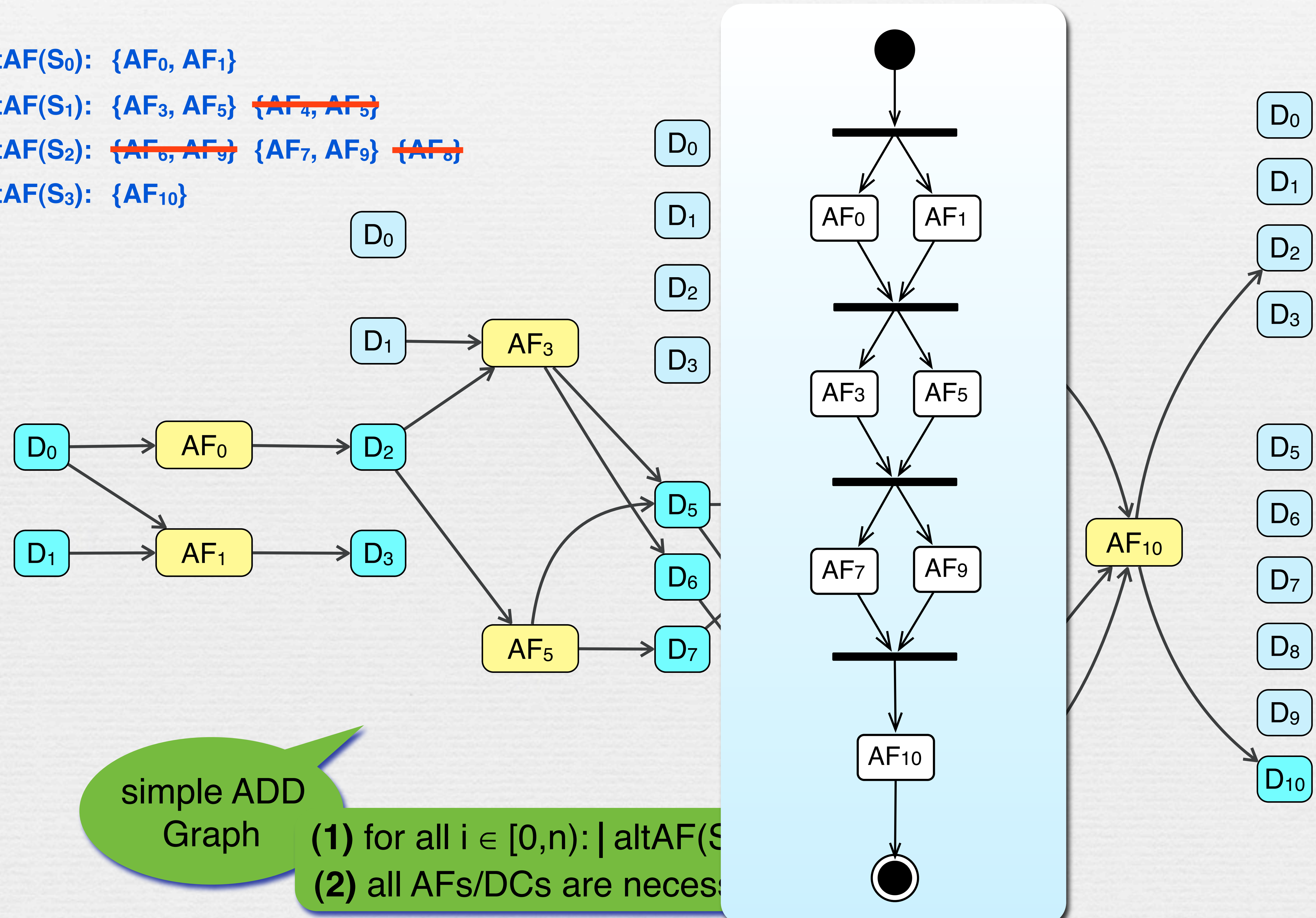
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} ~~{AF_4, AF_5}~~

altAF(S_2): ~~{AF_6, AF_9}~~ {AF_7, AF_9} ~~{AF_8}~~

altAF(S_3): {AF_{10}}



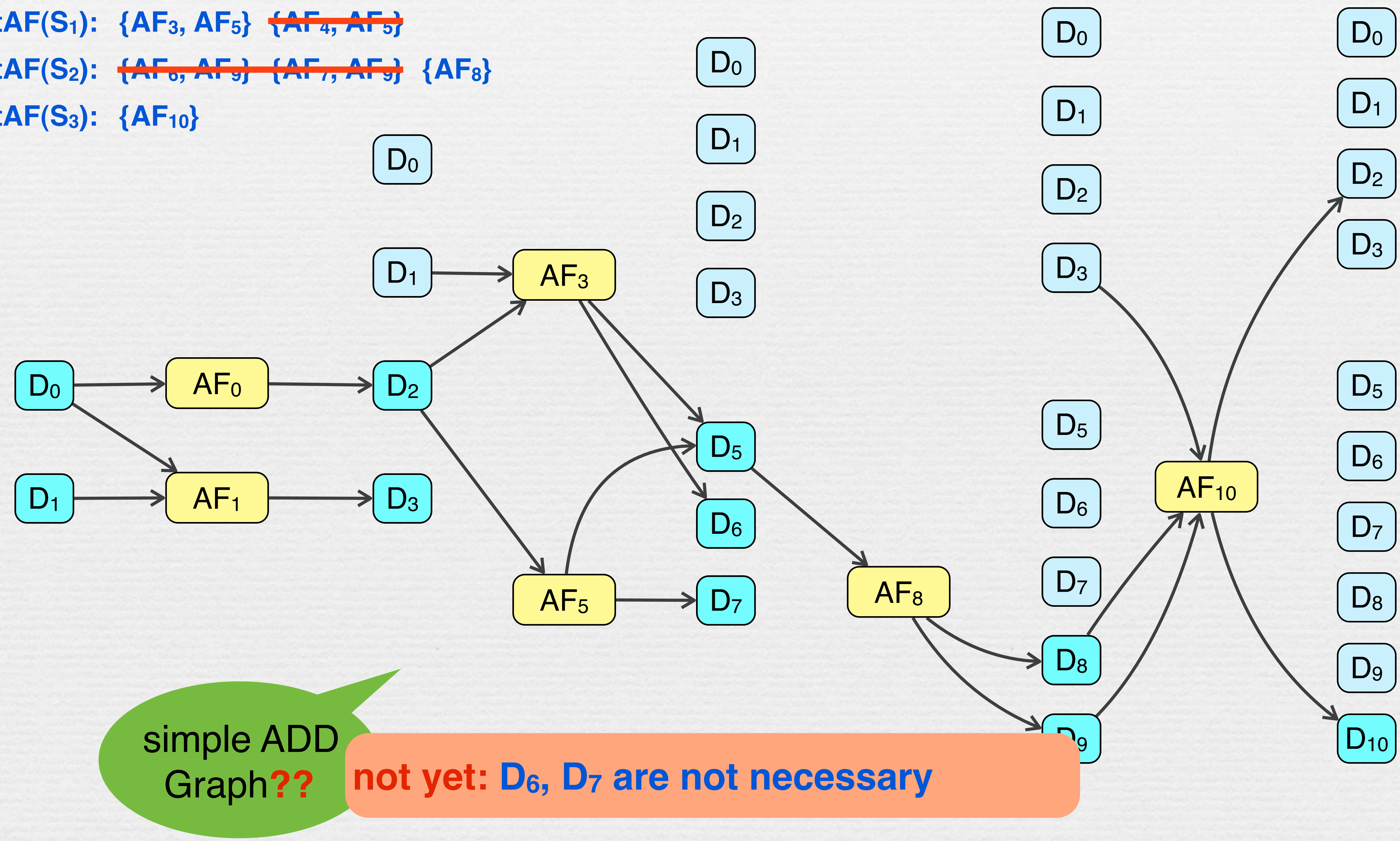
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} ~~{AF_4, AF_5}~~

altAF(S_2): ~~{AF_6, AF_9}~~ ~~{AF_7, AF_9}~~ {AF_8}

altAF(S_3): {AF_{10}}



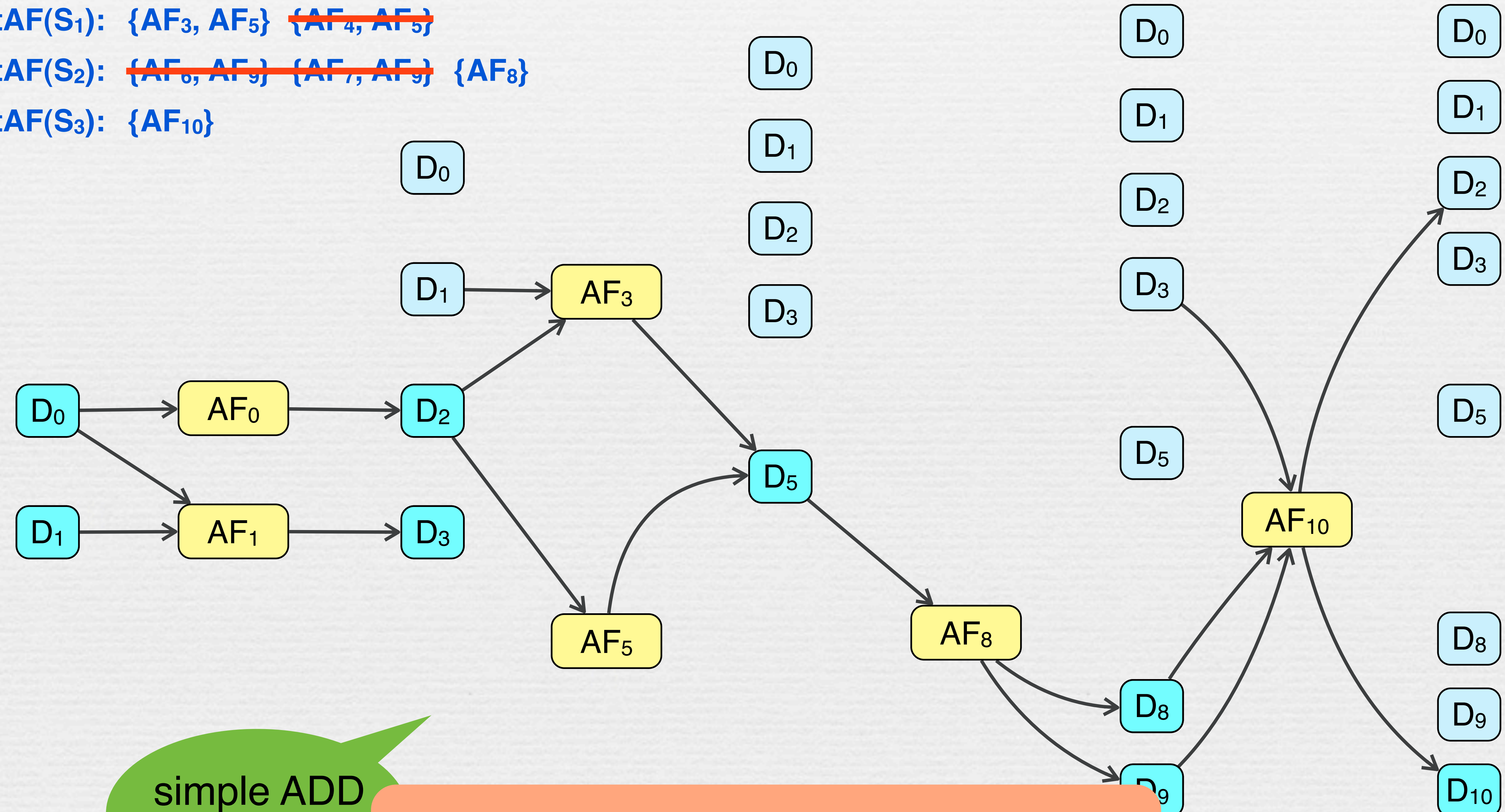
Workflow Extraction

altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} ~~{AF_4, AF_5}~~

altAF(S_2): ~~{AF_6, AF_9} {AF_7, AF_9}~~ {AF_8}

altAF(S_3): {AF_{10}}



simple ADD Graph??

not yet: D_6, D_7 are not necessary

not yet: | altAF(S_1) | = 2 : {AF_3}, {AF_5}

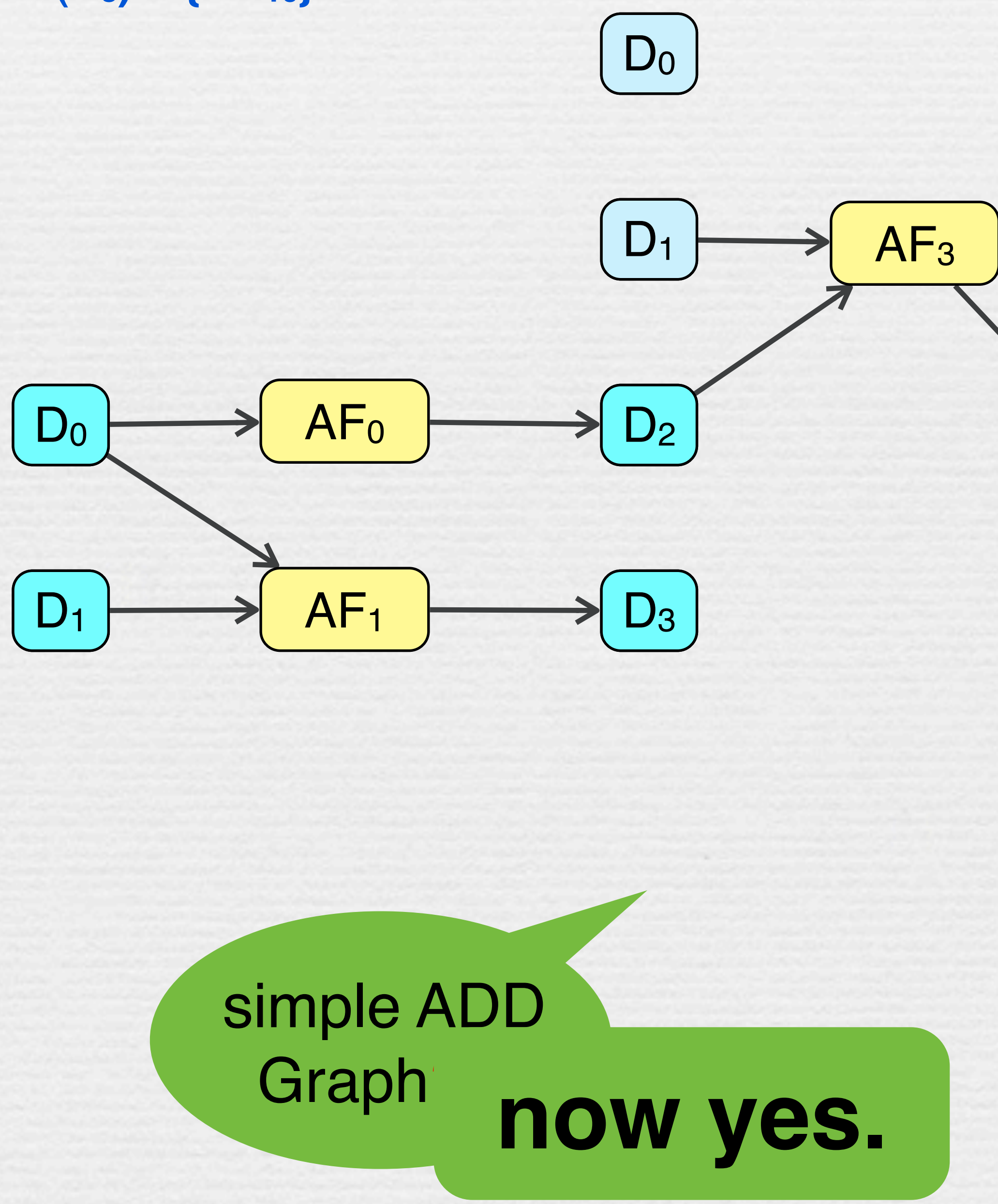
Workflow Extraction

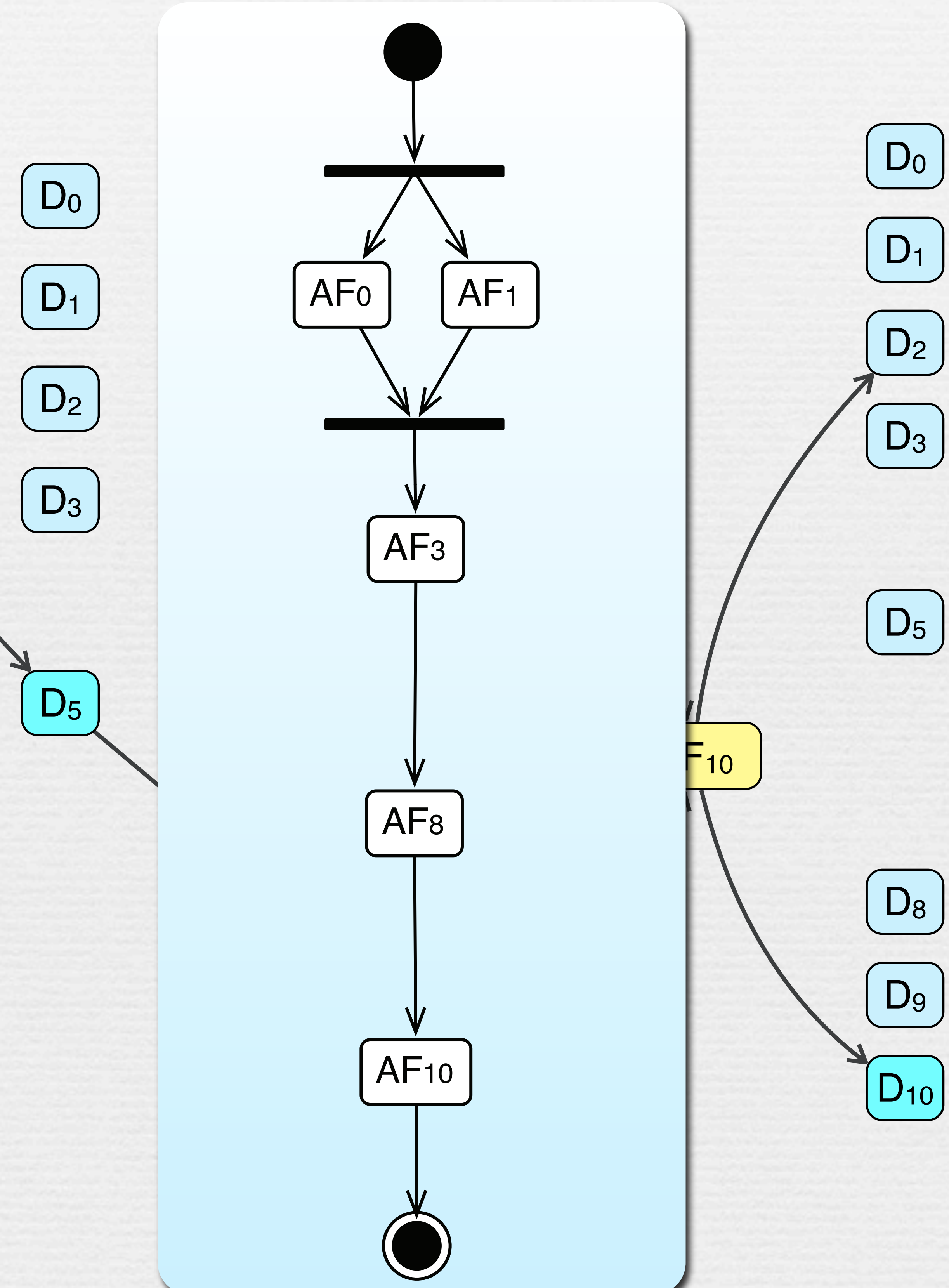
altAF(S_0): {AF_0, AF_1}

altAF(S_1): {AF_3, AF_5} ~~{AF_4, AF_5}~~

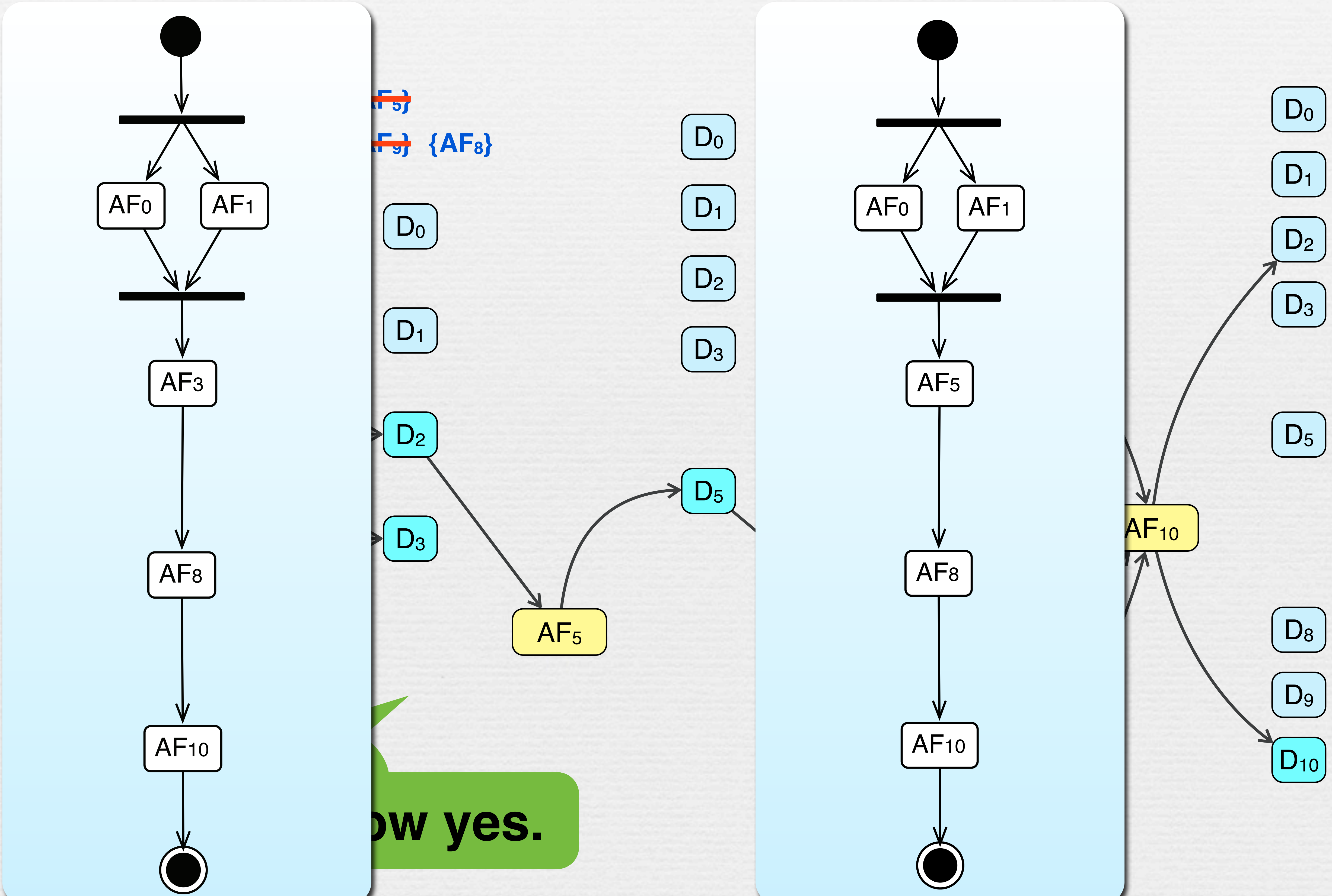
altAF(S_2): ~~{AF_6, AF_9}~~ ~~{AF_7, AF_9}~~ {AF_8}

altAF(S_3): {AF_{10}}

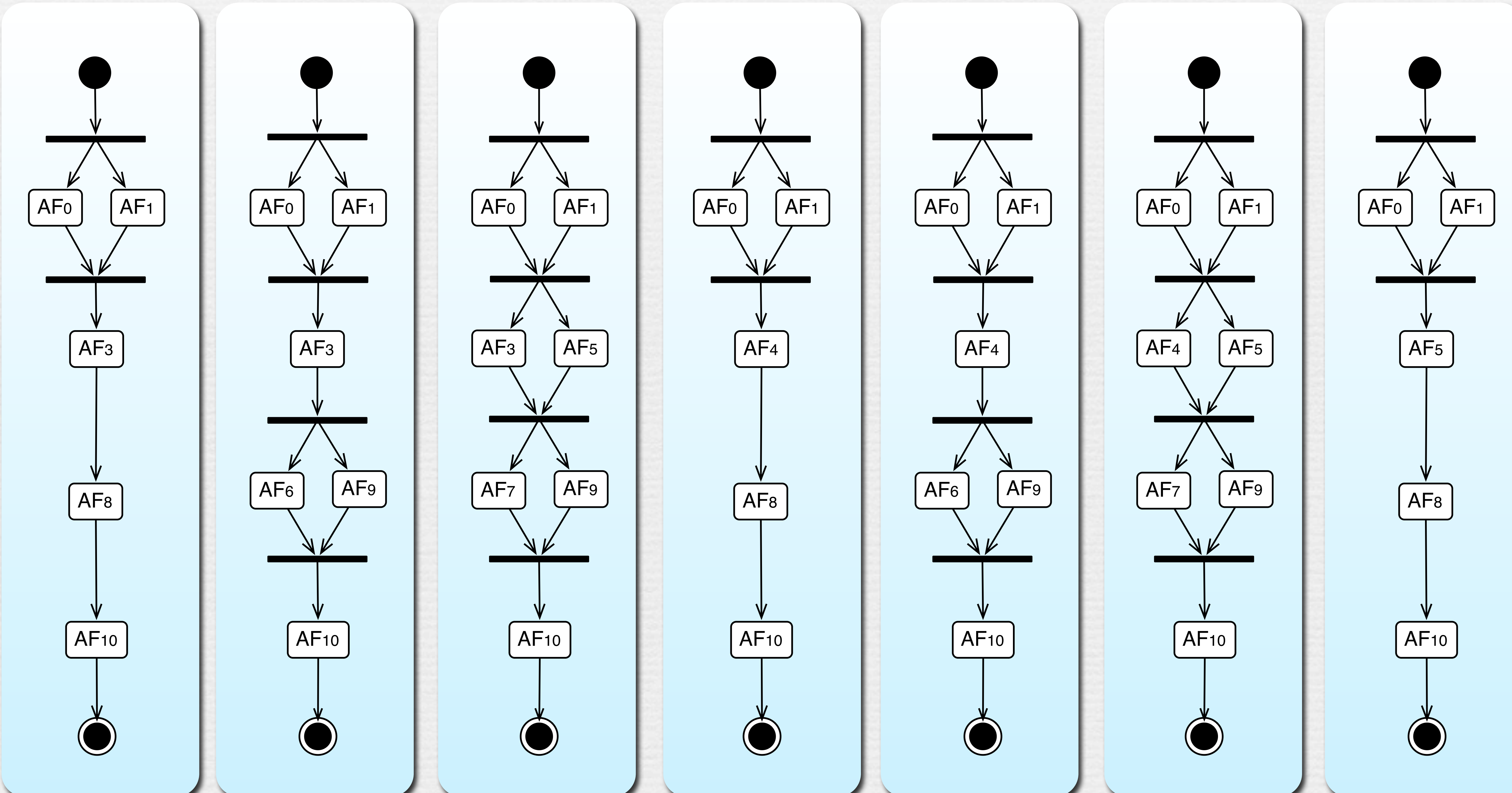




Workflow Extraction



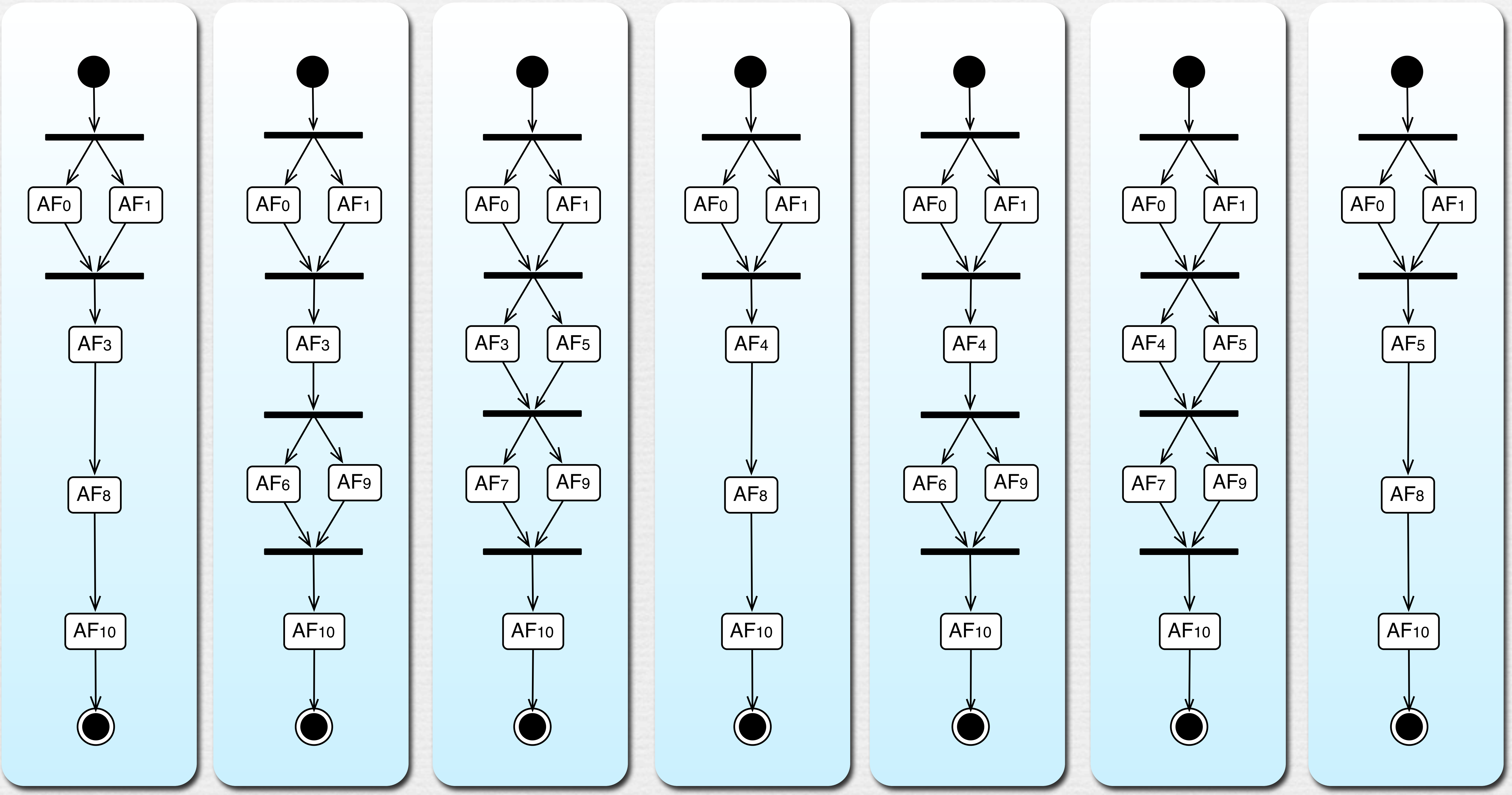
Workflow Extraction



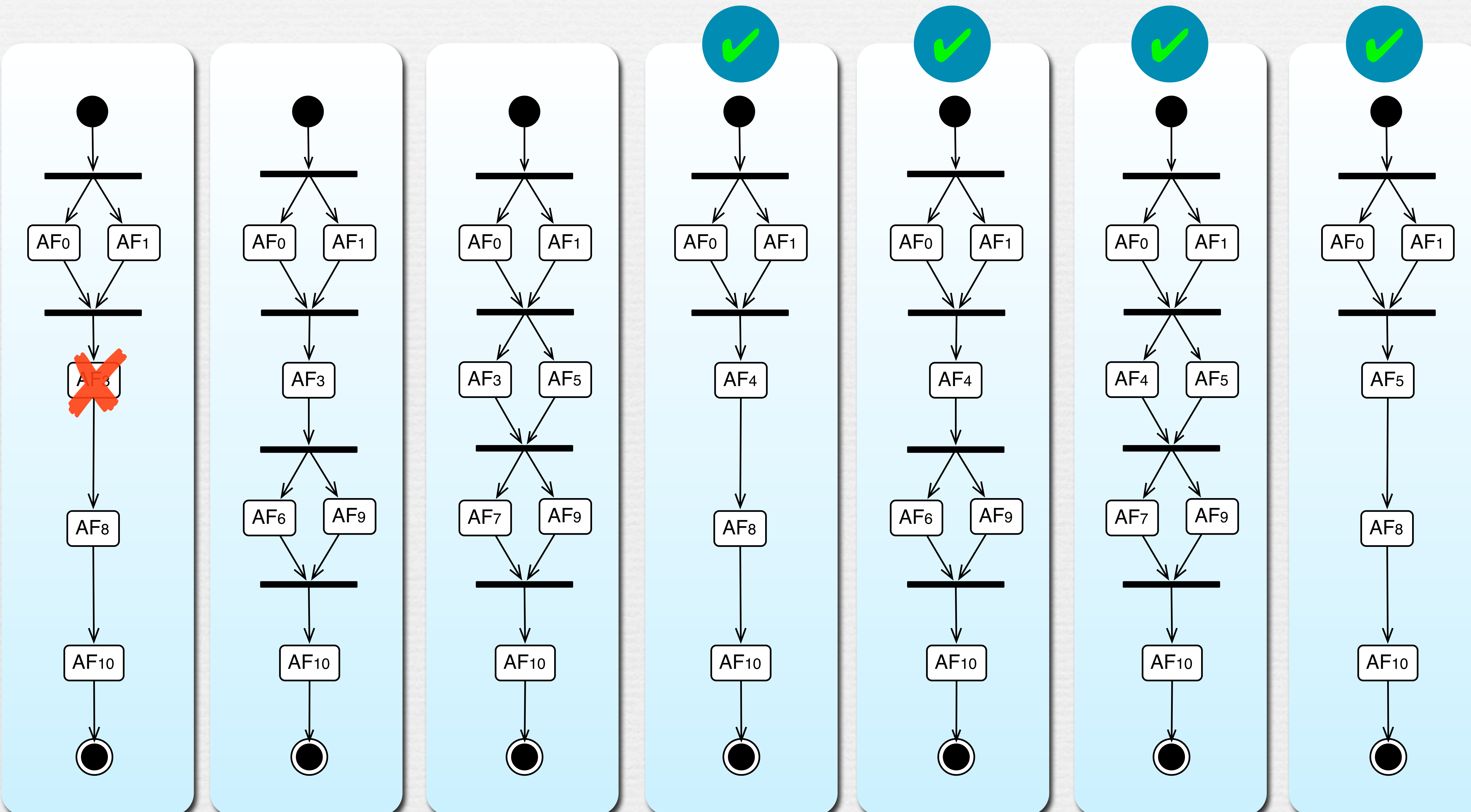
Why Alternative Workflows

- To support fault tolerance
- Why fault tolerance?
  - services and computers of distributed systems may fail unexpectedly
  - services may be registered or unregistered at any time without intimation, may happen even during workflow execution
  - Especially important for the Grid due to its dynamic nature.
- Alternative workflows is helpful
  - service associated with alternative activities may still available
  - alternative activities may run on different computers
- Automatic generation of alternative workflows makes ASKALON very different from other systems.

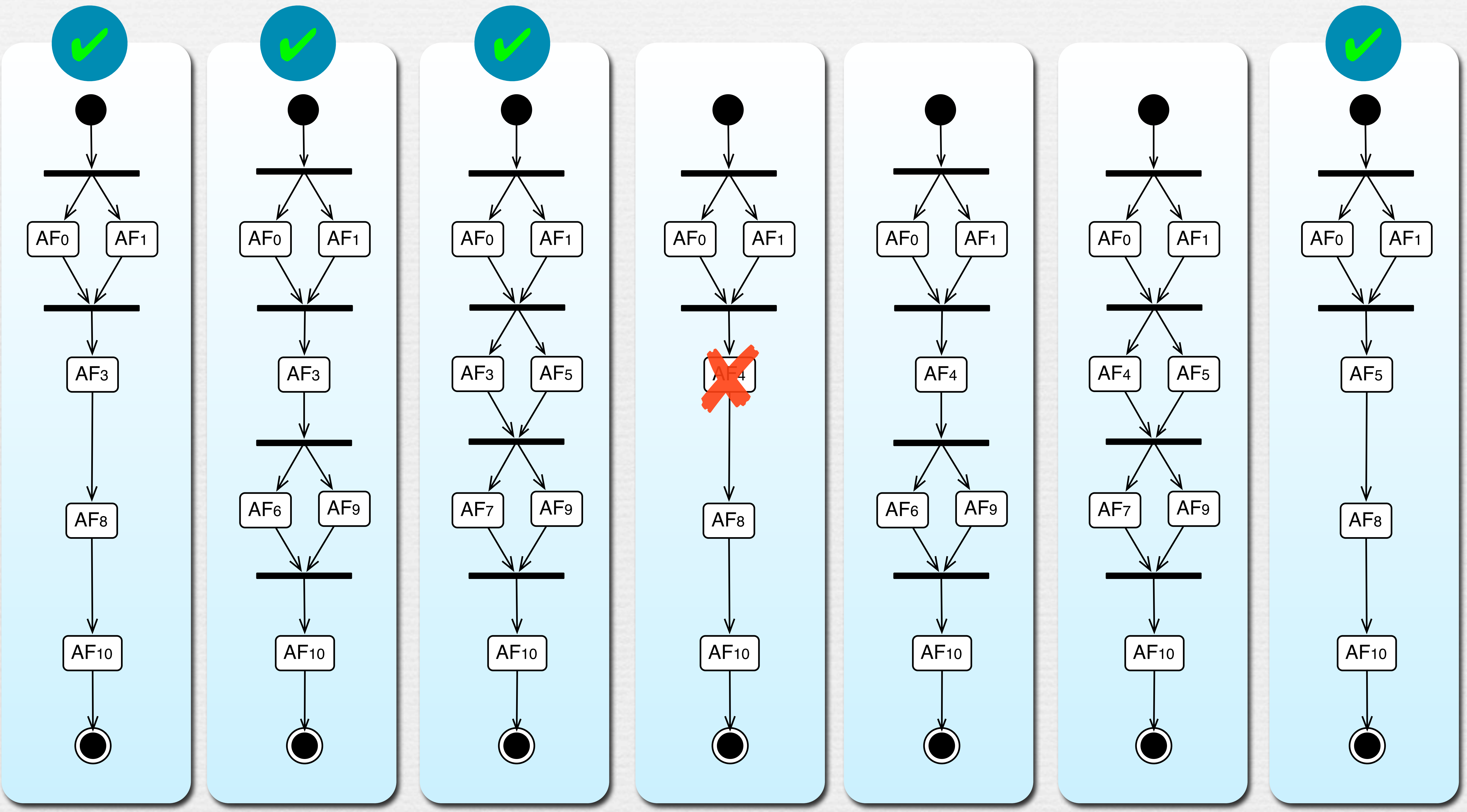
Fault Tolerance Support



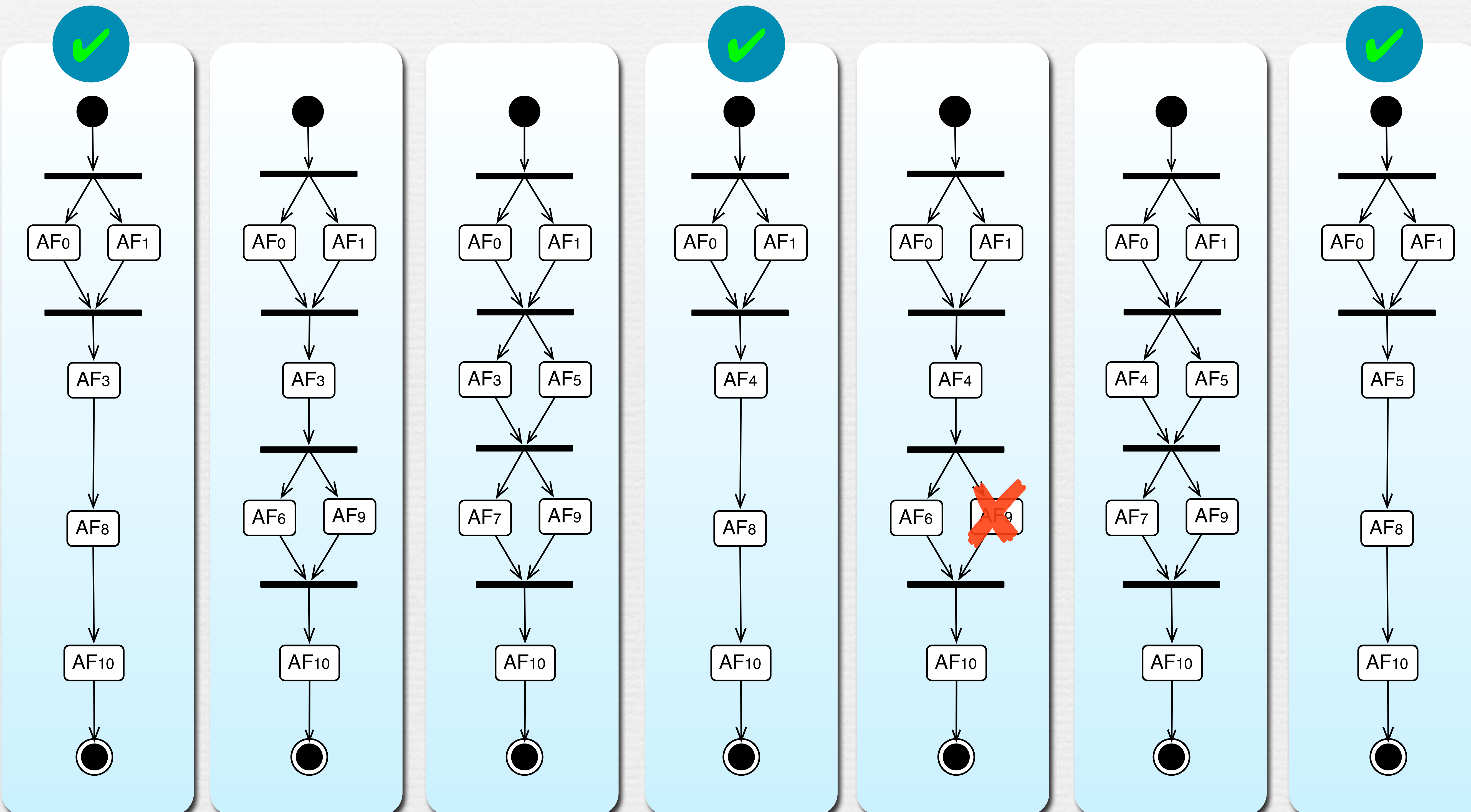
Fault Tolerance Support

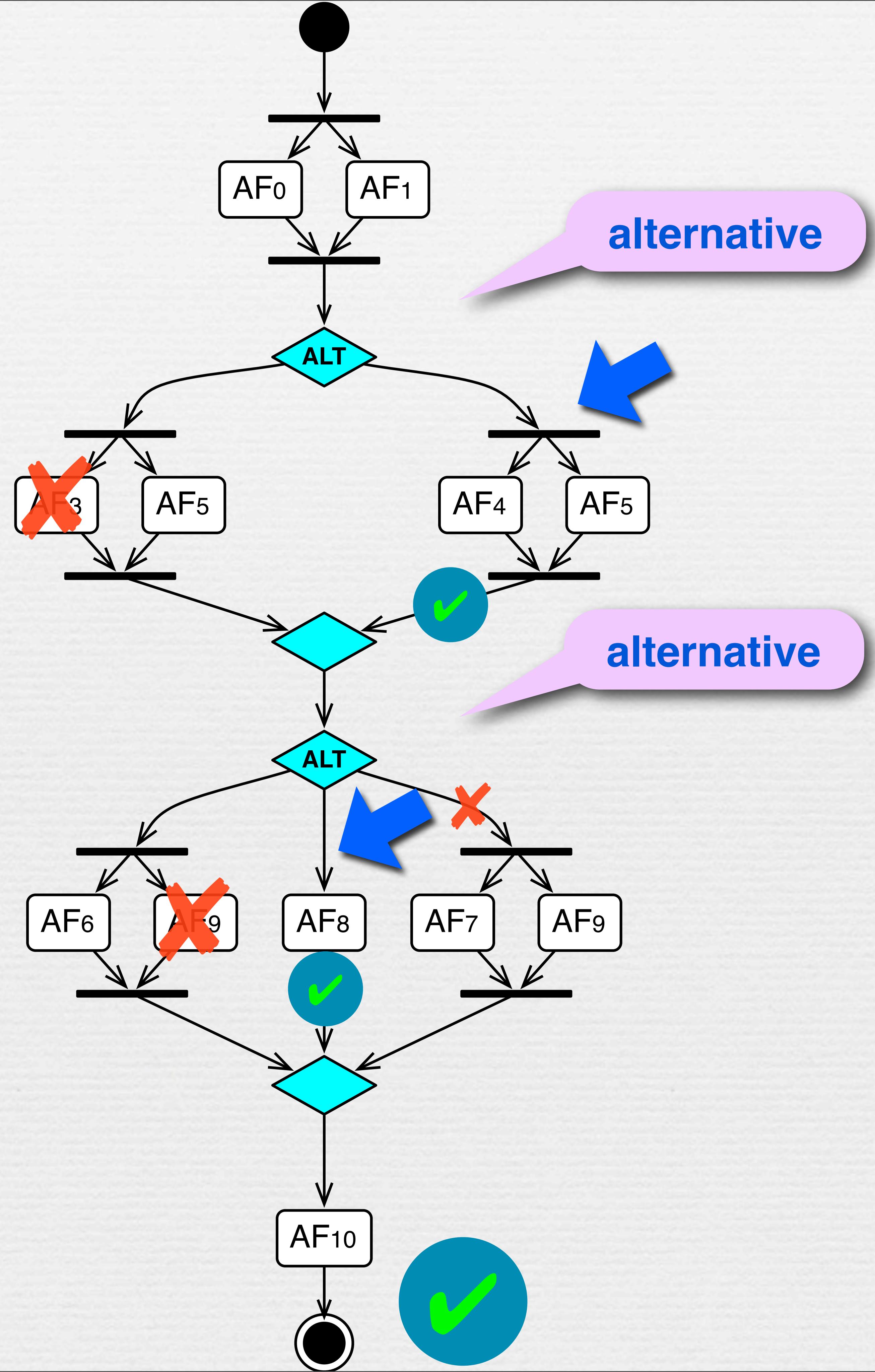


Fault Tolerance Support



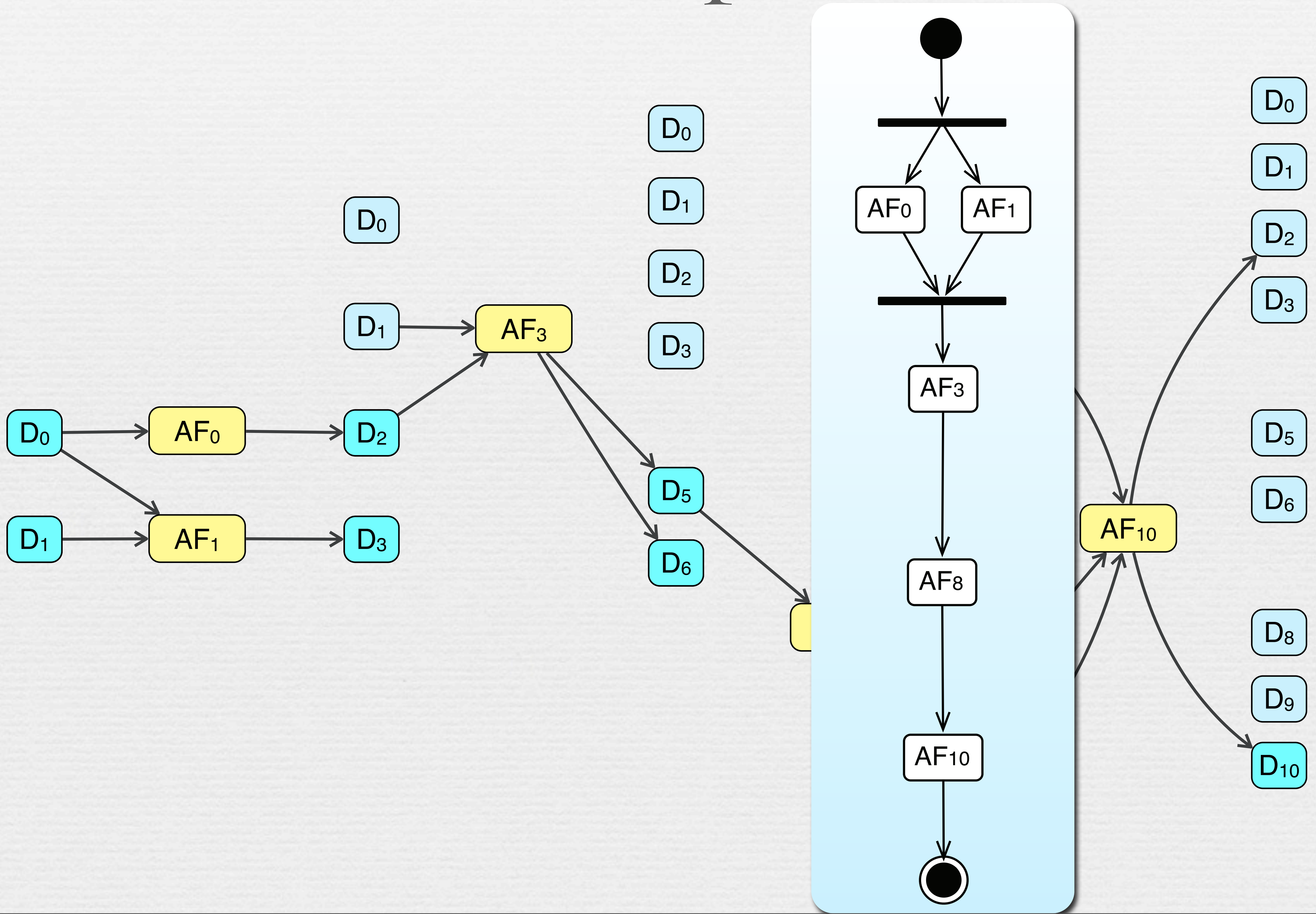
Fault Tolerance Support



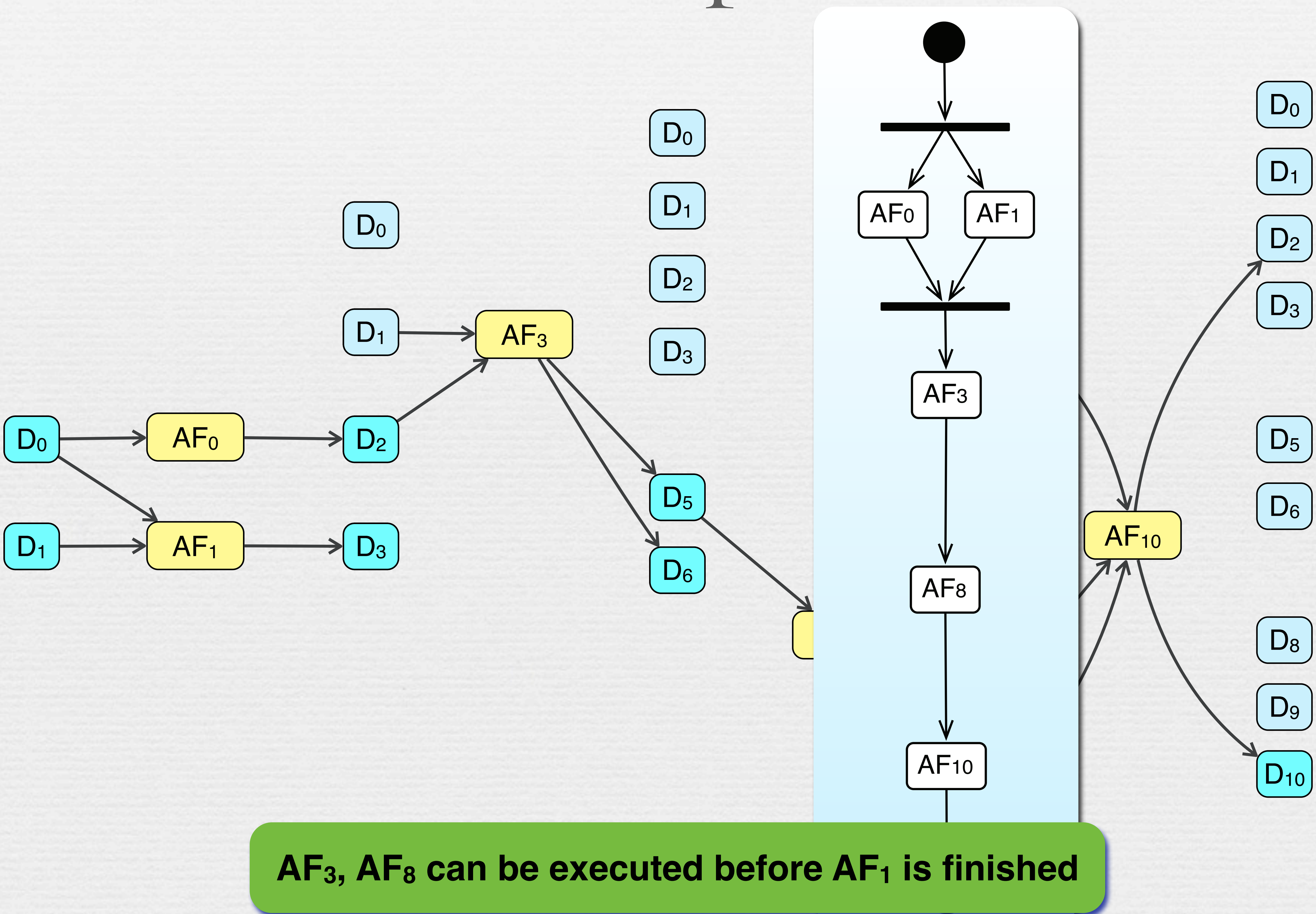


alternative
ALT
AF0
AF1
AF3
AF5
AF4
AF5
alternative
ALT
AF6
AF9
AF8
AF7
AF9
AF10

Workflow Optimization

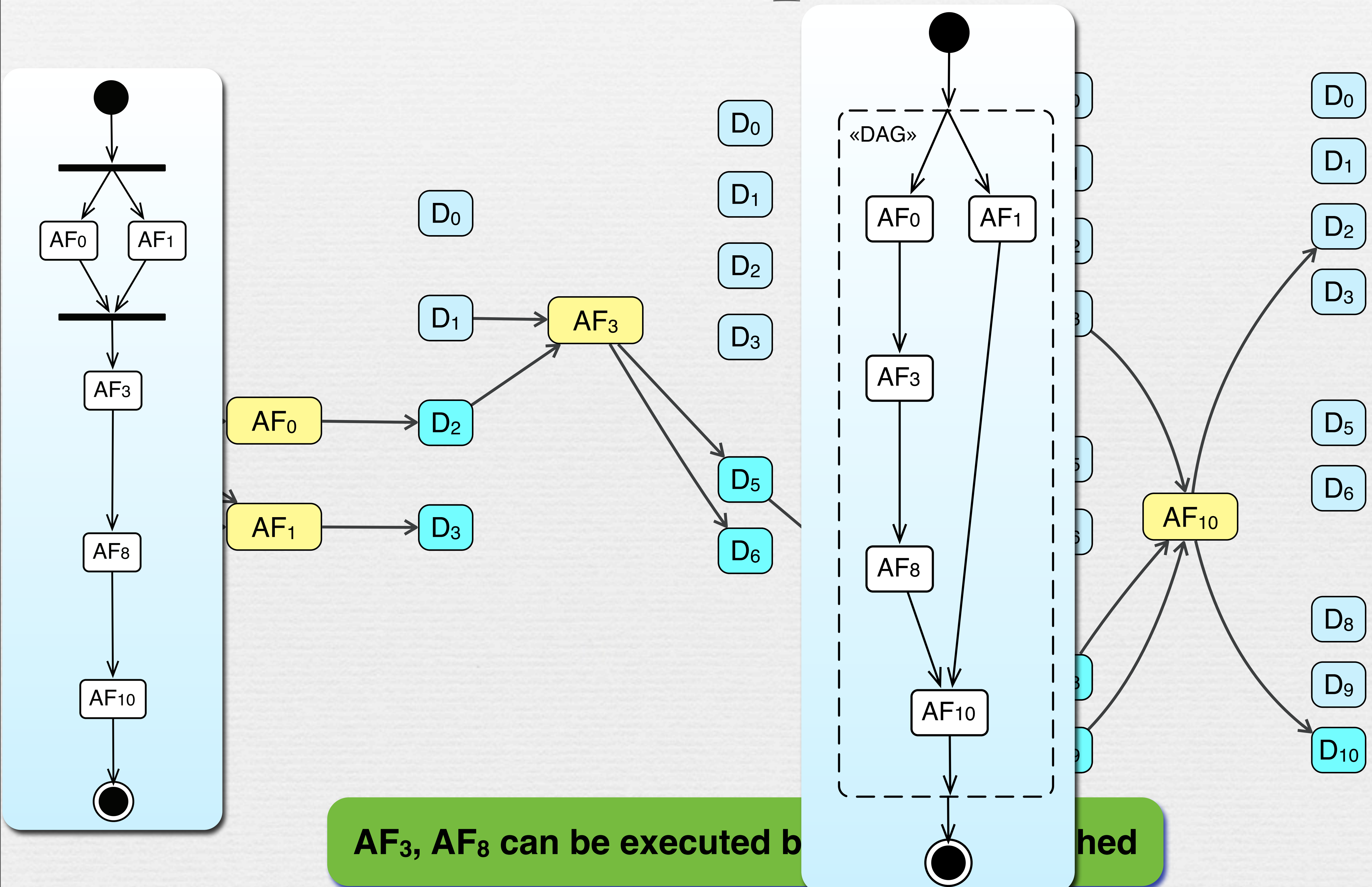


Workflow Optimization



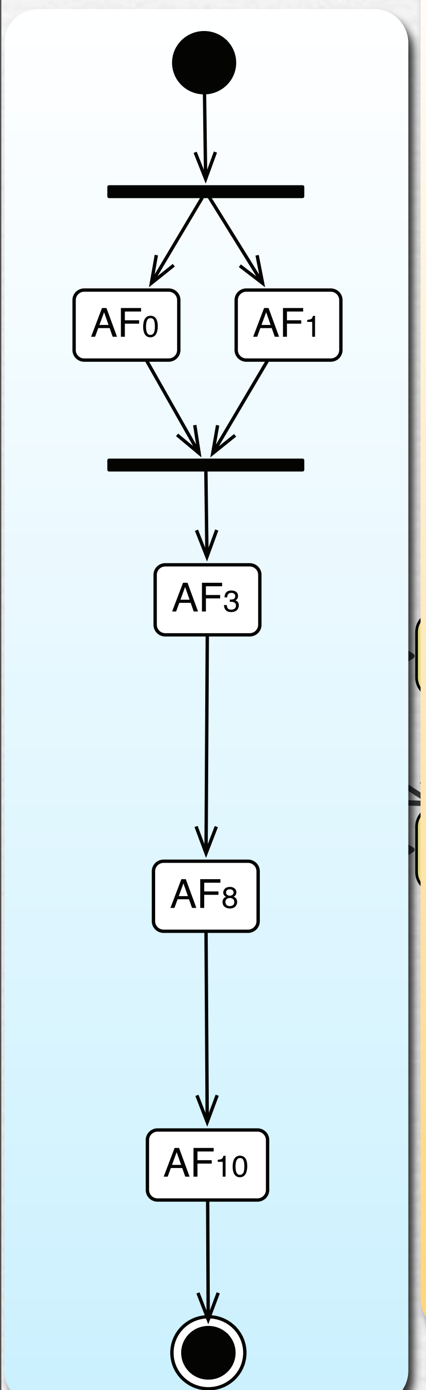
AF_3, AF_8 can be executed before AF_1 is finished

Workflow Optimization

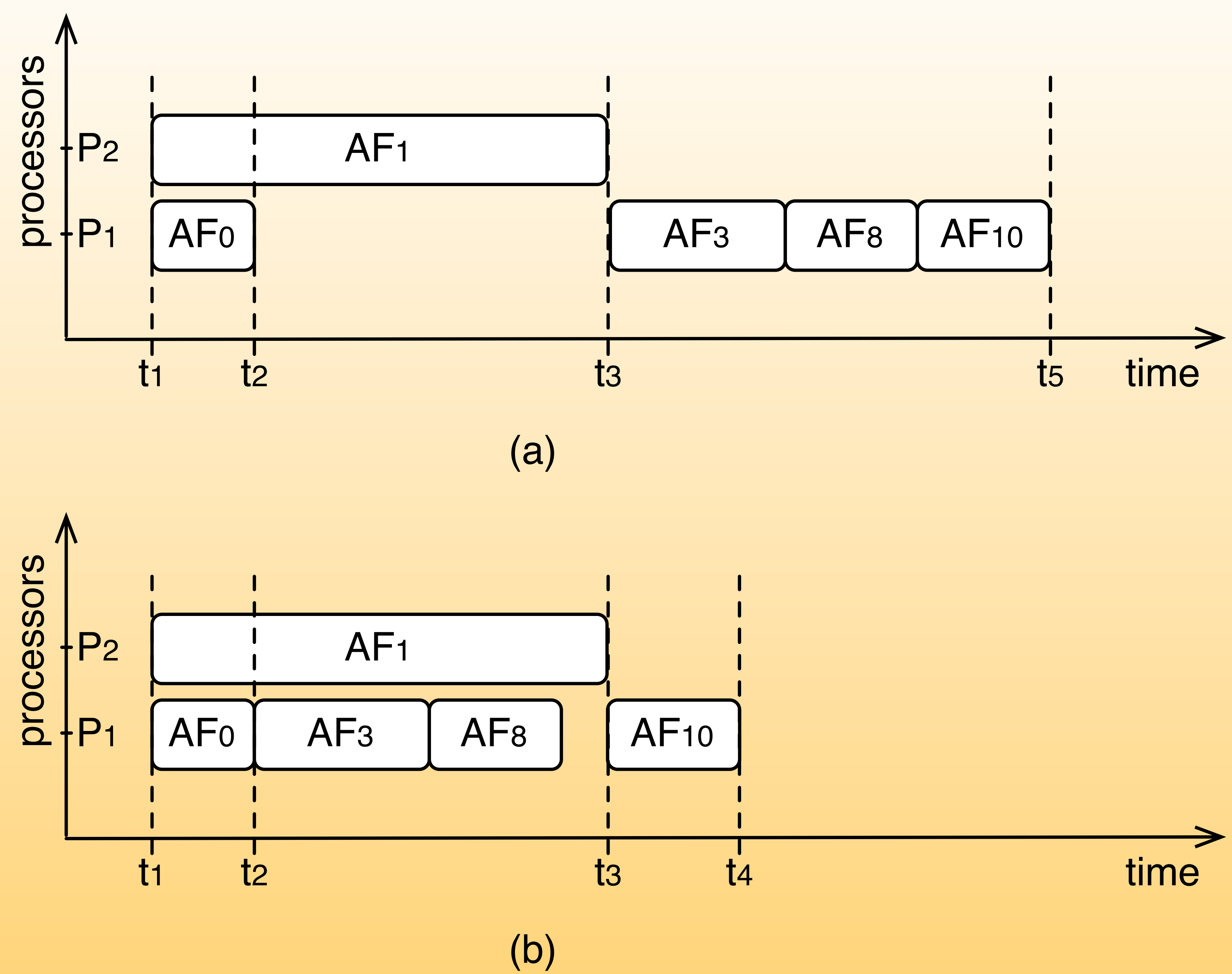


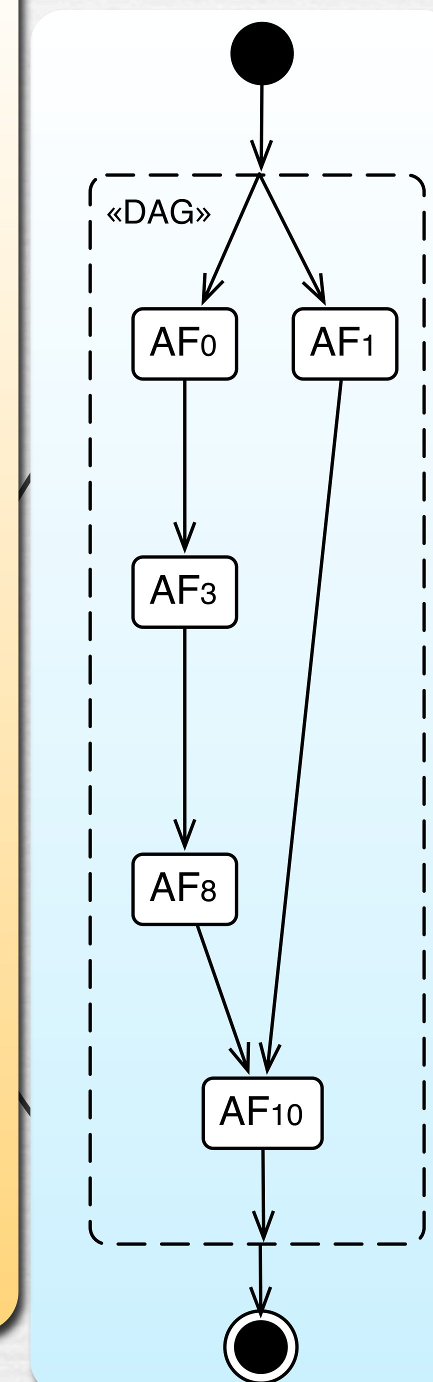
AF3, AF8 can be executed b… …hed

Workflow Optimization



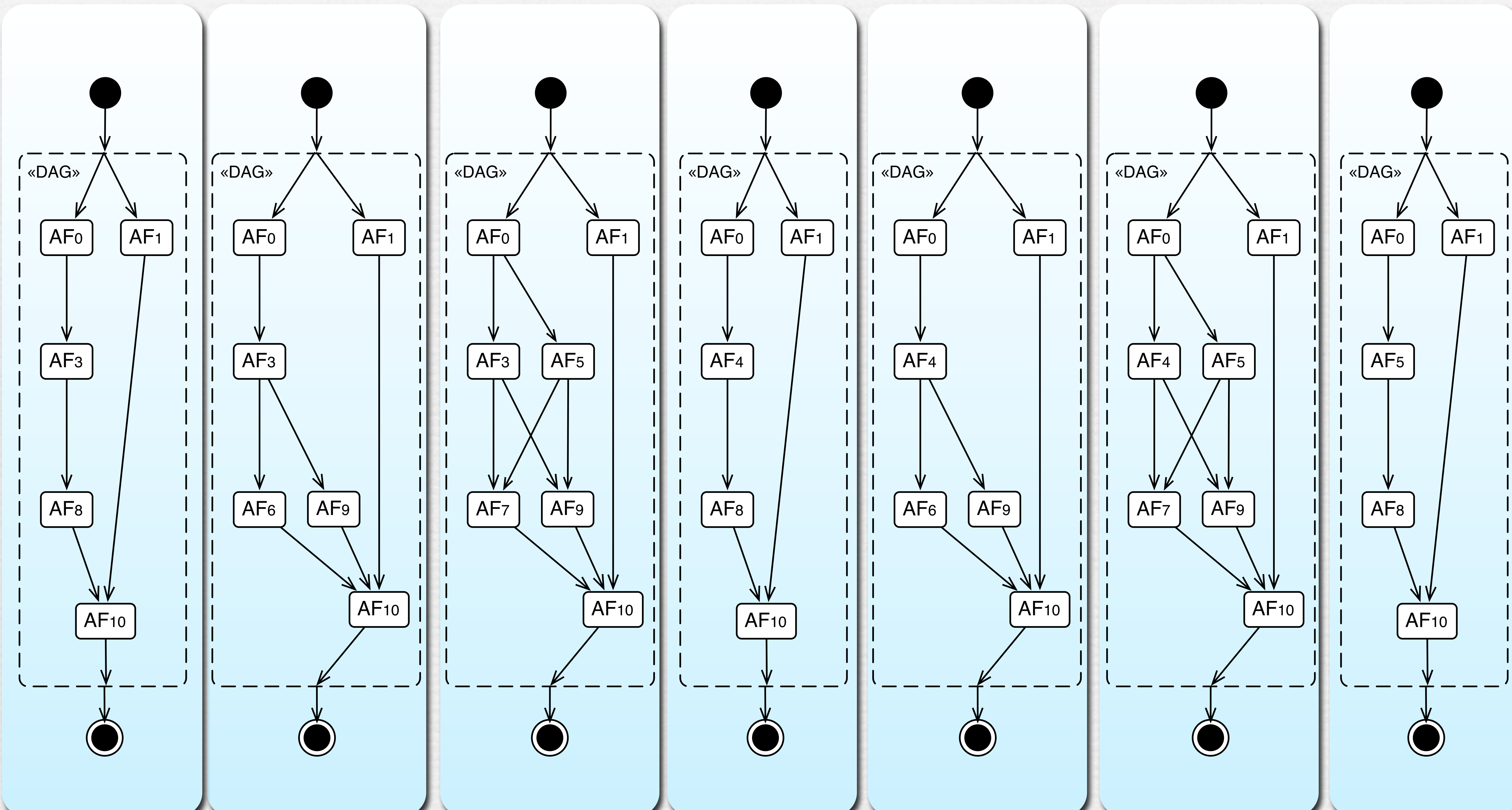
Execution Time Comparison





AF_3, AF_8 can be executed before AF_1 is finished

Workflow Optimization



Algorithm Analysis

W *is the set of all possible workflows, given a set of* ***x*** *AFs, an initial state* s_{init} *and a goal state* s_{goal}

Proposition 1. *The* ***worst case execution time*** *taken by our algorithm to find an element of* ***W*** *is* ***a quadratic in x*** *if* $W \neq \emptyset$. *If such an element is not found by our algorithm, then necessarily* $W = \emptyset$.

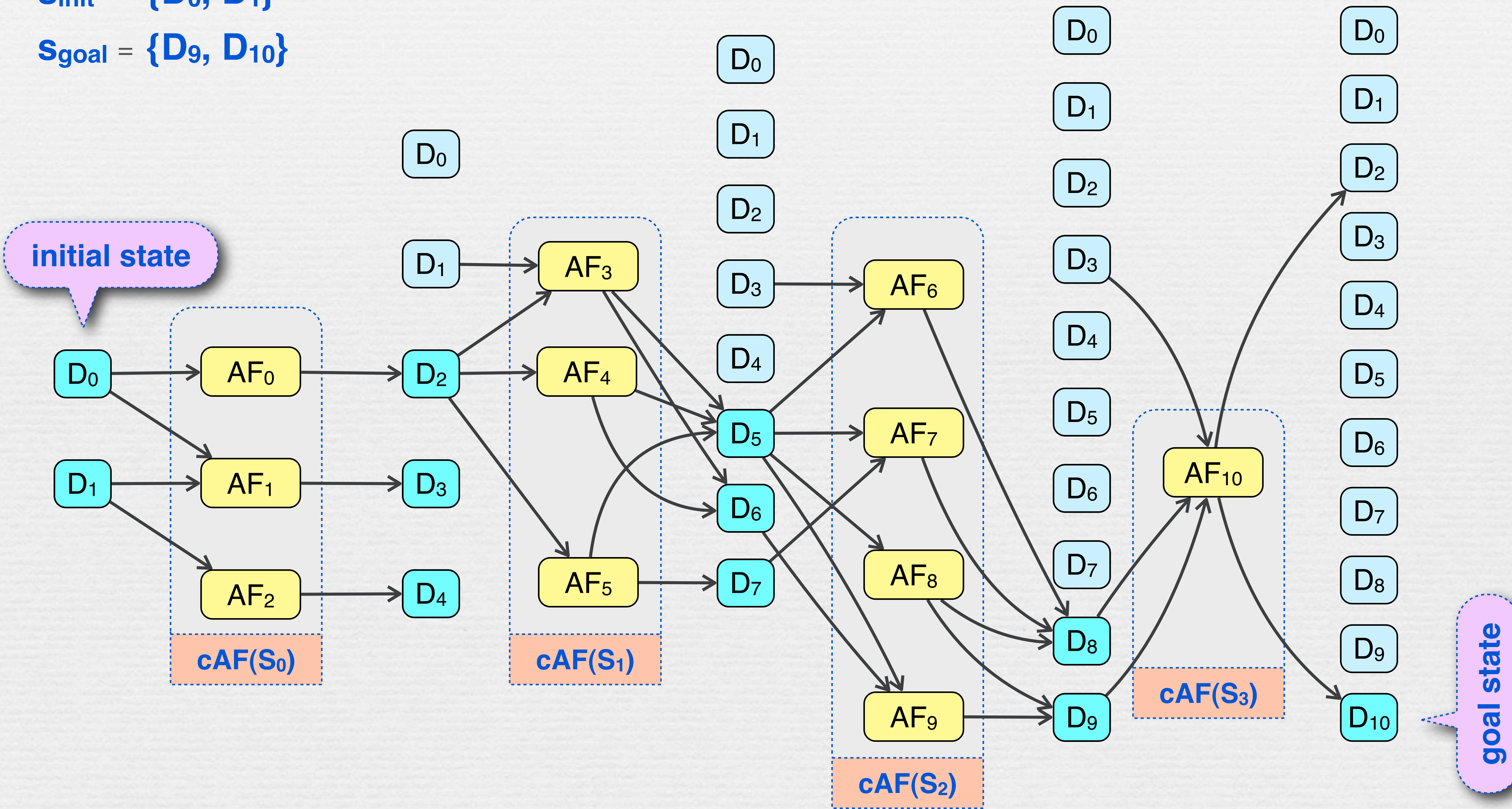
$$x + (x-1) + (x-2) + \ldots + 1 = \frac{x^2 + x}{2}$$

Proposition 2. *If an element of W is found by our algorithm, the number of the superstates of the ADD graph is minimum, which also means that* ***the length of the DAG workflow is minimum****.*

ADD Graph Creation

s_{init} = $\{D_0, D_1\}$

s_{goal} = $\{D_9, D_{10}\}$



all cAF(S) are disjoint

Branches & Loops

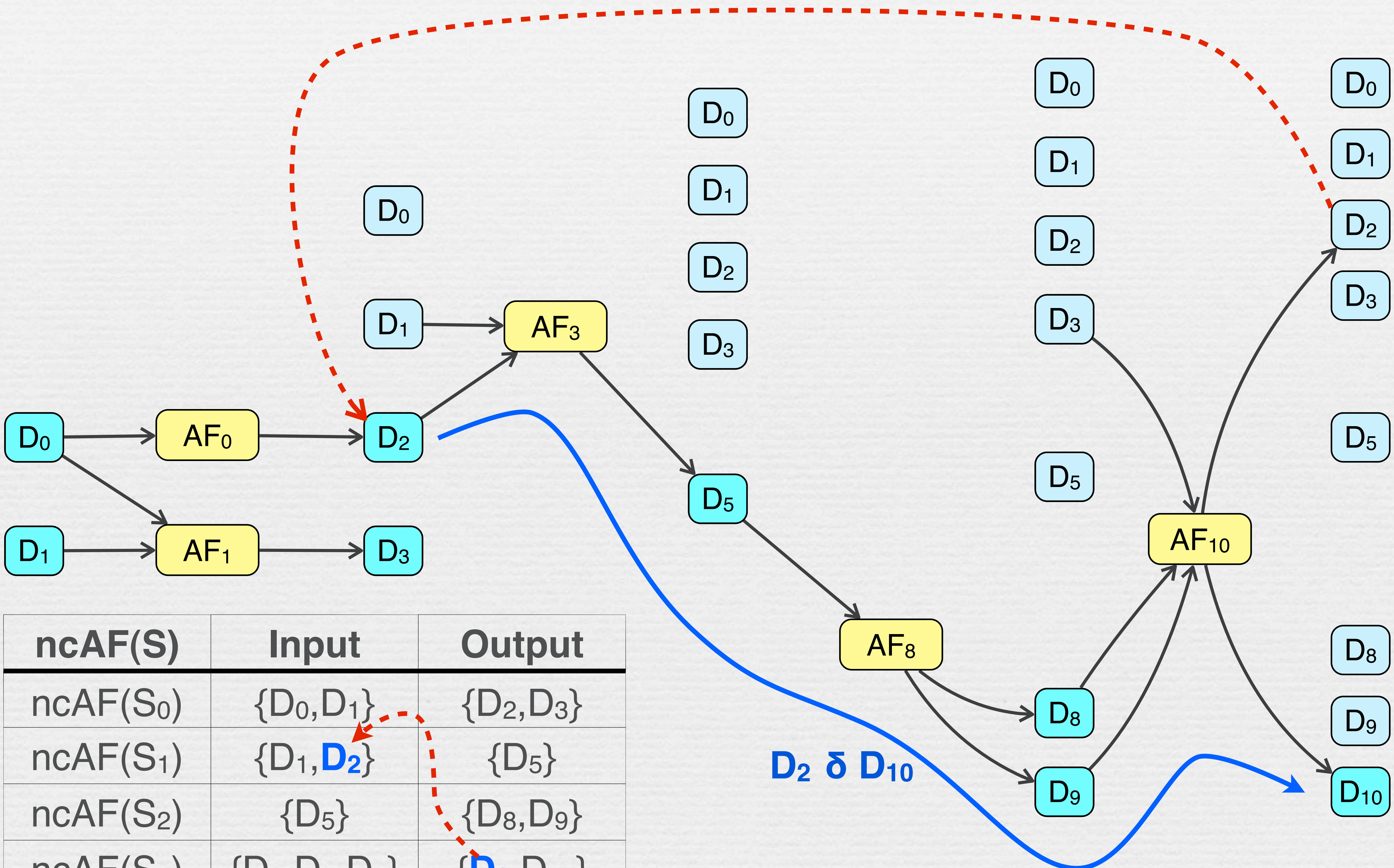
- Branches
  - s_{goal}={D9, D10(*agwl:precondition*=”*D6=true*”)}
    - STEP 1: s_{init1}=S_{init} , s_{goal1}={D9, D6}, to obtain ADD graph with $S_n \vDash s_{goal1}$, thereby get workflow W_1
    - STEP 2: s_{init2}=S_n, s_{goal2}={D10}, get workflow W_2
    - STEP 3: the workflow is W_1 , followed by an `if` construct where W_2 is the `then` branch, `else` branch is empty
- Parallel Loops
  - S_{init}={D1(*agwl:cardinality*=”*multiple*”, *agwl:access-order*=”*parallel*”), D2,}
    - STEP 1: s_{init}={D1, D2}, get workflow W
    - STEP 2: put W in a `parallelForEach` construct, which iterates over data collection D_1

Sequential Loops

- Sequential Loops
  - s_{goal}={D9, D10(*agwl:postcondition*="*D10<0.1*")}
    - STEP 1: s_{goal}={D9, D10}
    - STEP 2: find the start/stop point of the sequential loop
    - STEP 3: insert `doWhile` loop

Find Seq. Loop in a simple ADD Graph

s_{goal}={D_9, D_{10}(*agwl:postcondition*=”$D_{10}<0.1$”)}

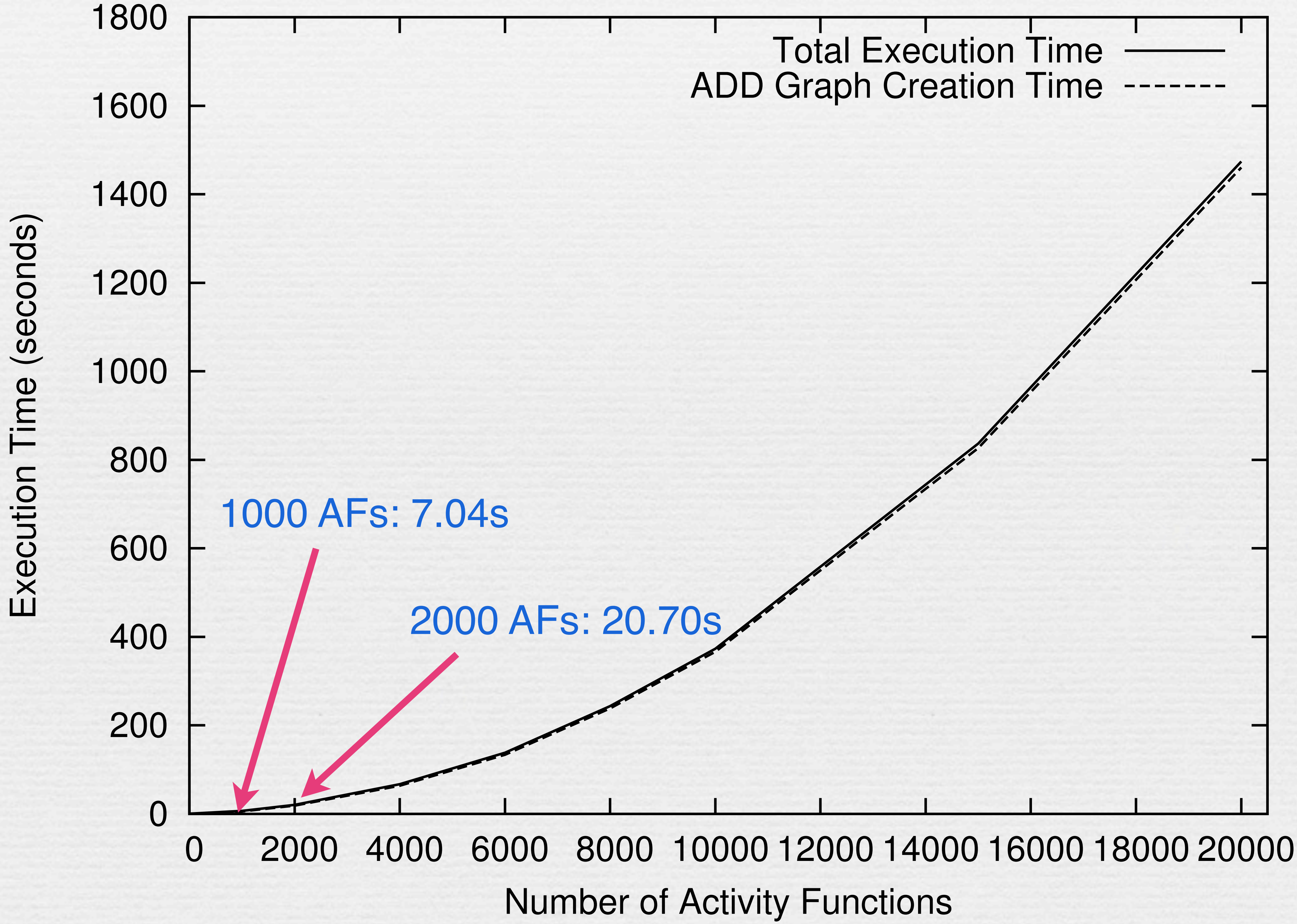


ncAF(S)	Input	Output
ncAF(S_0)	{D_0,D_1}	{D_2,D_3}
ncAF(S_1)	{D_1,**D_2**}	{D_5}
ncAF(S_2)	{D_5}	{D_8,D_9}
ncAF(S_3)	{D_3,D_8,D_9}	{**D_2**,D_{10}}

Experimental Results

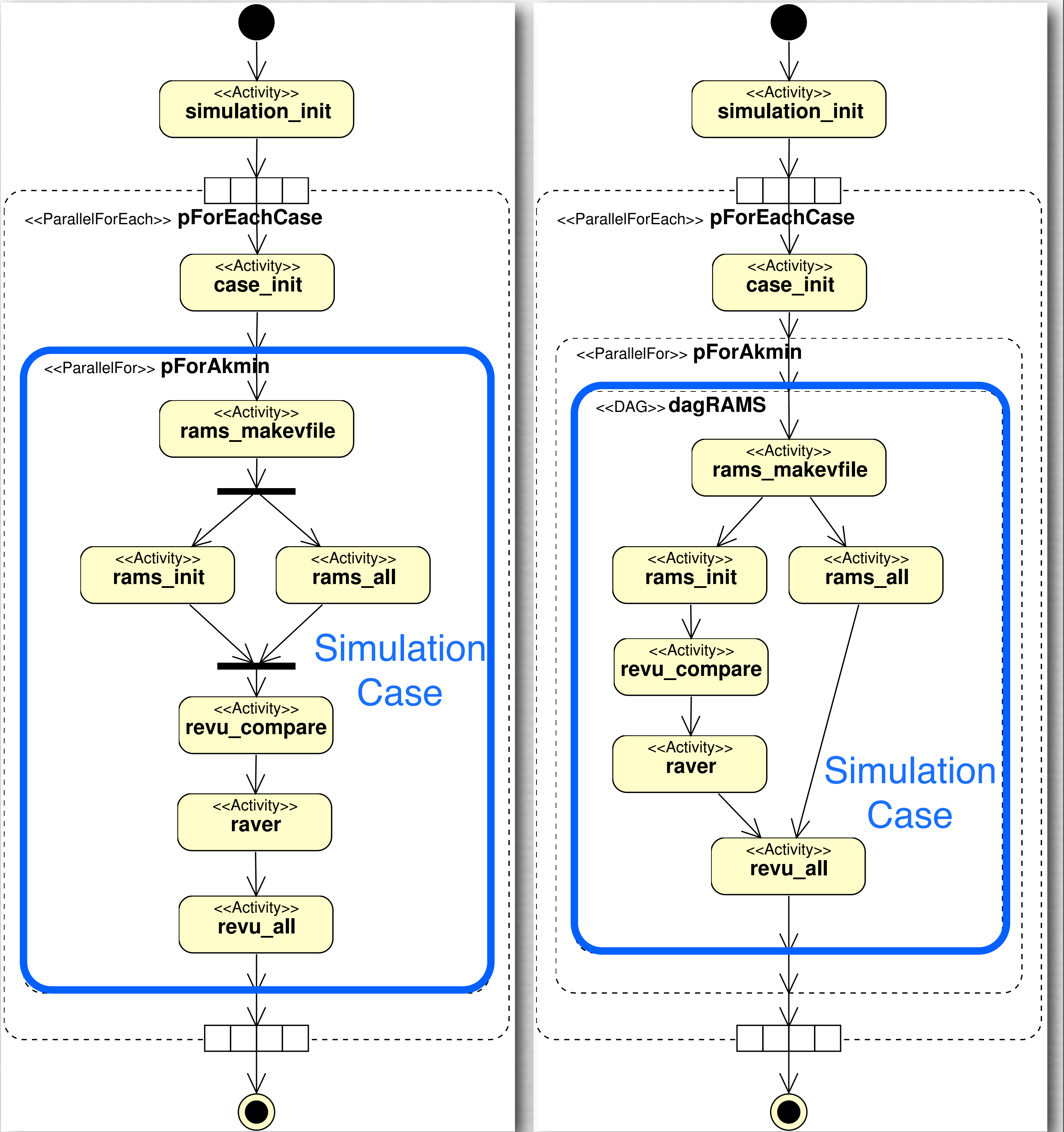
- Experiment 1
  - the composition of a Grid workflow in a simulated domain,
- Hardware and software environment
  - 2GB Memory, 2.4 GHz Intel Core 2 Duo CPU, JRE 1.5.0_16
- Ontology Setup
  - thousands of AFs and DCs: AF_0, AF_1, ...
  - thousands of DCs: D_0, D_1, ...
  - each AF has random number (between 1 and 10) of input and output DCs
    - AF_0: (D_0) ➔ ($D_{1.1}$)
    - AF_1: (D_0, D_1) ➔ (D_2)
    - AF_2: (D_2) ➔ (D_1, $D_{3.2}$)
    - AF_3: (D_0, D_1, D_3) ➔ ($D_{4.1}$)
    - ...
    - AF_{100}: (D_3, D_{12}, D_{35}, D_{82}, D_{90}, D_{93}, D_{100}) ➔ (D_5, D_{23}, D_{52}, D_{73}, $D_{101.8}$)
    - ...
    - AF_{20000}: ...

Algorithm Execution Time



Experimental Results

- Experiment 2
  - the composition of a real world Grid workflow MeteoAG in the meteorology domain
  - 19 AFs in the ontology
  - algorithm execution time:
    - 0.54 seconds for non-optimized version
    - 0.64 seconds for optimized version

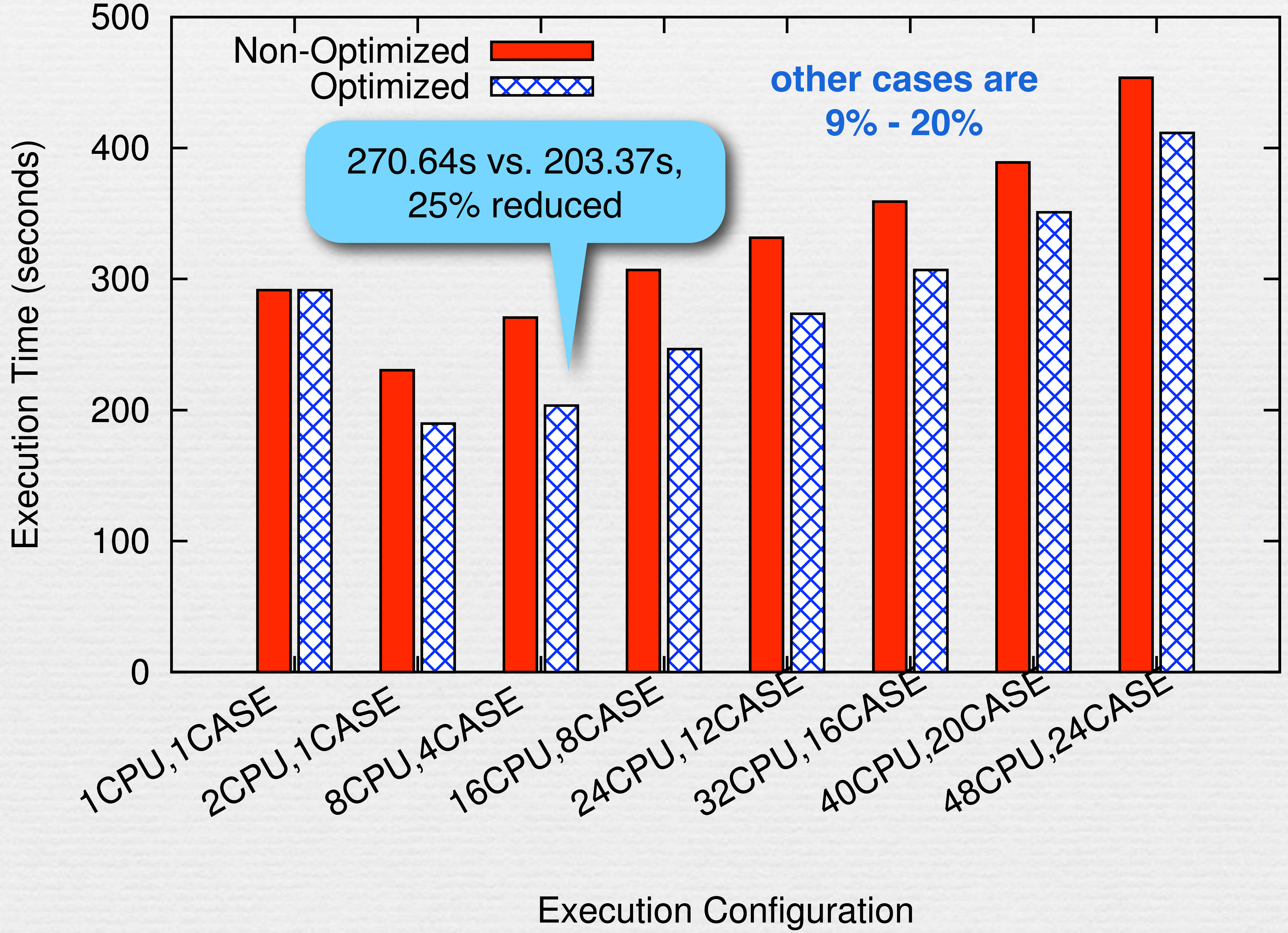


Experimental Results

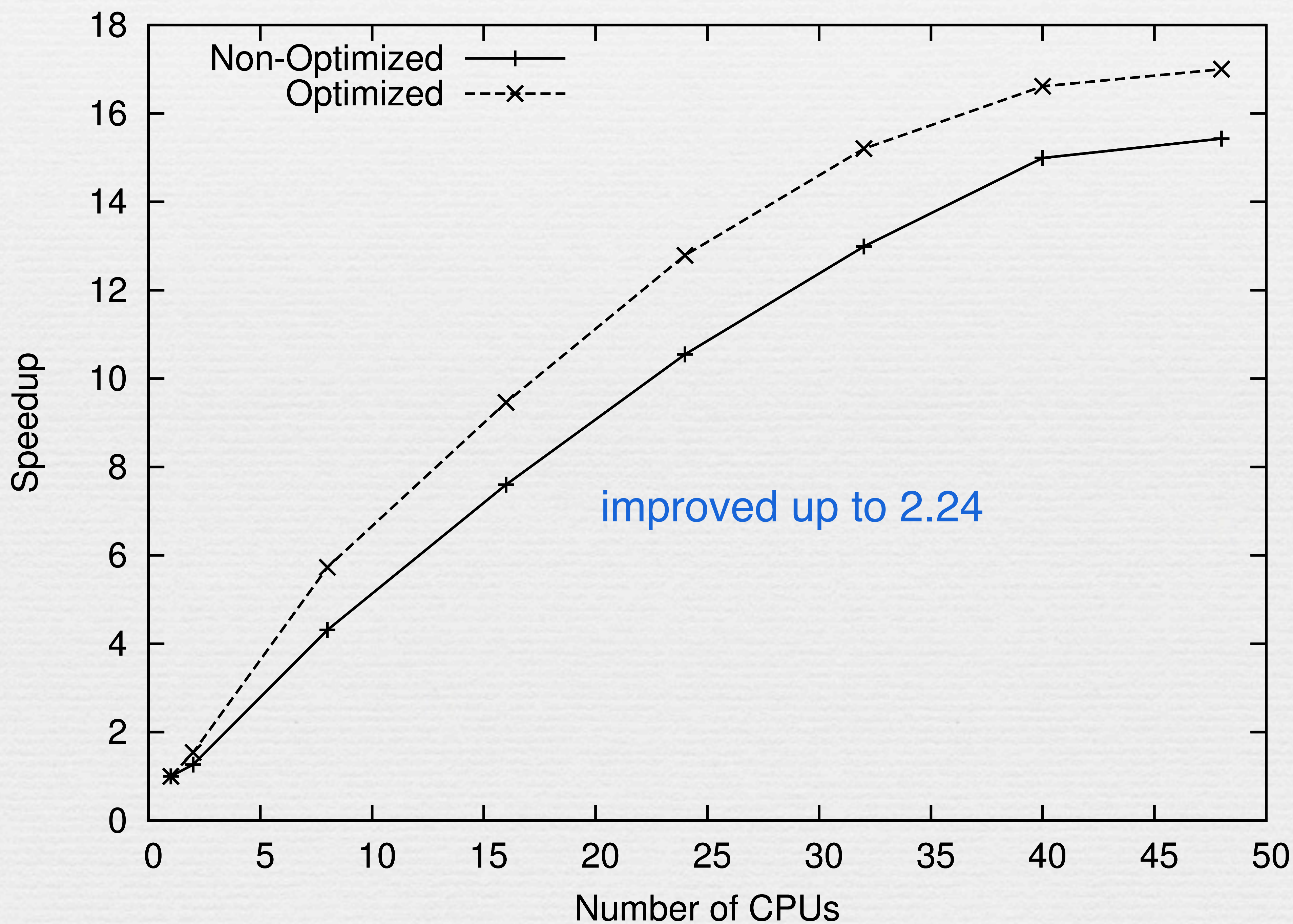
- Experiment 3
  - the comparison of the execution time of the optimized and non-optimized MeteoAG
  - Austrian Grid testbed

Grid Site	*CPU*	*#*	*GHz*	*LRM*	*Location*
karwendel	Dual Core AMD Opteron	8	2.4	SGE	Innsbruck
altix1	Itanium 2	8	1.4	PBS	Innsbruck
schafberg	Itanium 2	8	1.4	PBS	Salzburg
altix1jku	Itanium 2	8	1.4	PBS	Linz
c703-pc1801	Pentium 4	8	2.8	Torque	Innsbruck
c703-pc2601	Pentium 4	8	2.8	Torque	Innsbruck

Workflow Execution Time



Speedup



Non-Optimized
Optimized
improved up to 2.24
Speedup
Number of CPUs
0
2
4
6
8
10
12
14
16
18
0
5
10
15
20
25
30
35
40
45
50

Summary

- We formalized the Grid workflow composition problem based on the STRIPS language
- We presented a novel ADD graph based algorithm for automatic composition of high quality Grid workflows: portable, fault tolerant support, optimized.
- Our approach
  - provides a general solution for automatic Grid workflow composition
  - can generate alternative workflows automatically
  - considers workflow optimization
  - can compose workflows with branches and loops
- To the best of our knowledge, ASKALON provides the only widely used workflow systems with a general solution for automatic workflow composition
  - others are built for demonstration of concepts
- Future work
  - partially known initial state

Thank you

- For more information:
    - ASKALON: www.askalon.org
    - AGWL: www.askalon.org/agwl