A Decision Tree-based Approach to Dynamic Pointcut Evaluation

Robert Dyer and Hridesh Rajan

Department of Computer Science
Iowa State University
{rdyer,hridesh}@cs.iastate.edu

October 19, 2008

- Motivation: Dynamic PCD Evaluation
- Approach: Decision-tree based Matching
- Technical Contributions:
  - Formalization of the PCD Evaluation problem
  - Algorithms using Decision-tree structures for faster matching
  - Use of implication relationships for partial evaluation of type predicates

$$\begin{array}{lll} \boldsymbol{a} & \in & \mathcal{A}\text{, the set of attributes} \\ \boldsymbol{o} & \in & \mathcal{O}\text{, the set of operators} \\ \boldsymbol{v} & \in & \mathcal{V}\text{, the set of values} \end{array}$$

$$
\begin{aligned}
\boldsymbol{a} &\in \mathcal{A}, \text{ the set of attributes} \\
\boldsymbol{o} &\in \mathcal{O}, \text{ the set of operators} \\
\boldsymbol{v} &\in \mathcal{V}, \text{ the set of values}
\end{aligned}
$$

$$
\begin{aligned}
\textit{pred} &::= (\ \boldsymbol{a}\ ,\ \boldsymbol{o}\ ,\ \boldsymbol{v}\ ) \\
\textit{fact} &::= (\ \boldsymbol{a}\ ,\ \boldsymbol{v}\ )
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{a} &\in \mathcal{A}, \text{ the set of attributes} \\
\boldsymbol{o} &\in \mathcal{O}, \text{ the set of operators} \\
\boldsymbol{v} &\in \mathcal{V}, \text{ the set of values}
\end{aligned}
$$

$$
\begin{aligned}
\textit{pred} &::= (\ \boldsymbol{a}\ ,\ \boldsymbol{o}\ ,\ \boldsymbol{v}\ ) \\
\textit{fact} &::= (\ \boldsymbol{a}\ ,\ \boldsymbol{v}\ )
\end{aligned}
$$

$$
\begin{aligned}
\textit{PCD} &::= \textit{pred} \\
&\ \ \mid (\ \textit{PCD}\ ) \\
&\ \ \mid \textit{pred}\ \&\&\ \textit{PCD} \\
&\ \ \mid \textit{pred}\ ||\ \textit{PCD} \\
\textit{join point} &::= \textit{fact} \\
&\ \ \mid \textit{fact}\ \&\&\ \textit{join point}
\end{aligned}
$$

$\mathcal{A} ::= \{\textbf{\textit{modifier}}, \textbf{\textit{type}}, \textbf{\textit{name}}\}$
$\mathcal{V} ::= \{\textbf{\textit{v}} : \textbf{\textit{v}}$ is a modifier, type or name in the program$\}$
$\mathcal{O} ::= \{==, !=\}$

$\mathcal{A}$ ::= {***modifier***, ***type***, ***name***}
$\mathcal{V}$ ::= {***v*** : ***v*** is a modifier, type or name in the program}
$\mathcal{O}$ ::= {==, !=}

Example PCD

```
(modifier, ==, public) && (type, !=, void) &&
(name, ==, "Set")
```

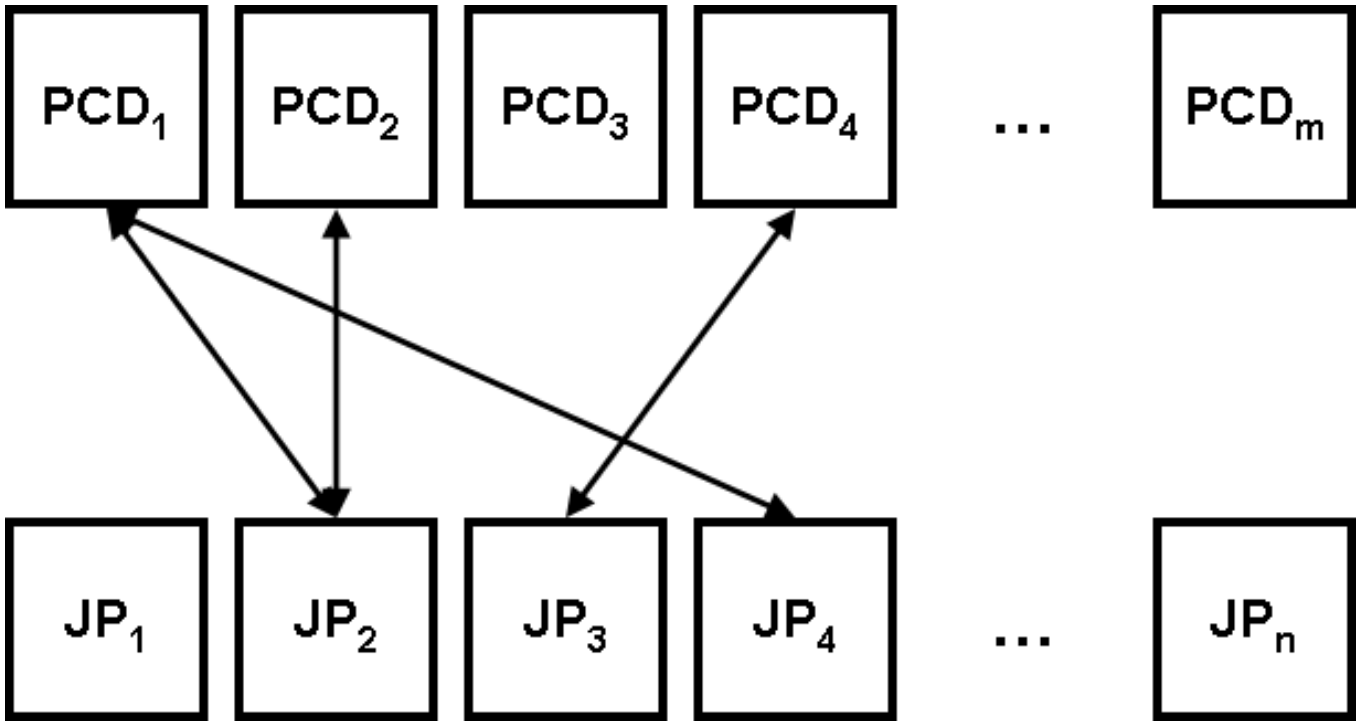
$\mathcal{A}$ ::= {***modifier***, ***type***, ***name***}
$\mathcal{V}$ ::= {***v*** : ***v*** is a modifier, type or name in the program}
$\mathcal{O}$ ::= {==, !=}

Example PCD

```
(modifier, ==, public) && (type, !=, void) &&
(name, ==, "Set")
```

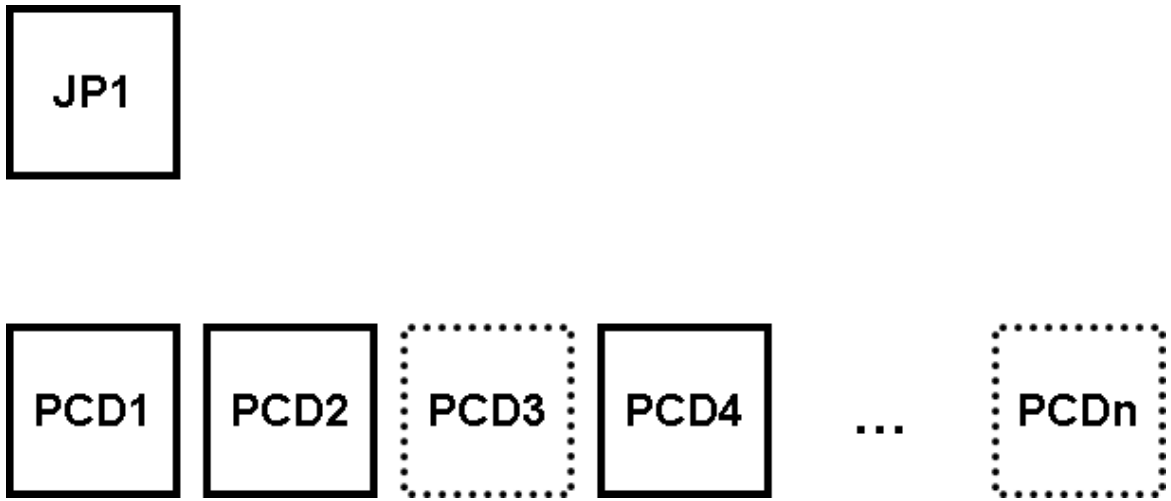
Example join point

```
(modifier, public) && (type, FElement) &&
(name, "Set")
```


PCD1
PCD2
PCD3
PCD4
...
PCDm
JP1
JP2
JP3
JP4
...
JPn

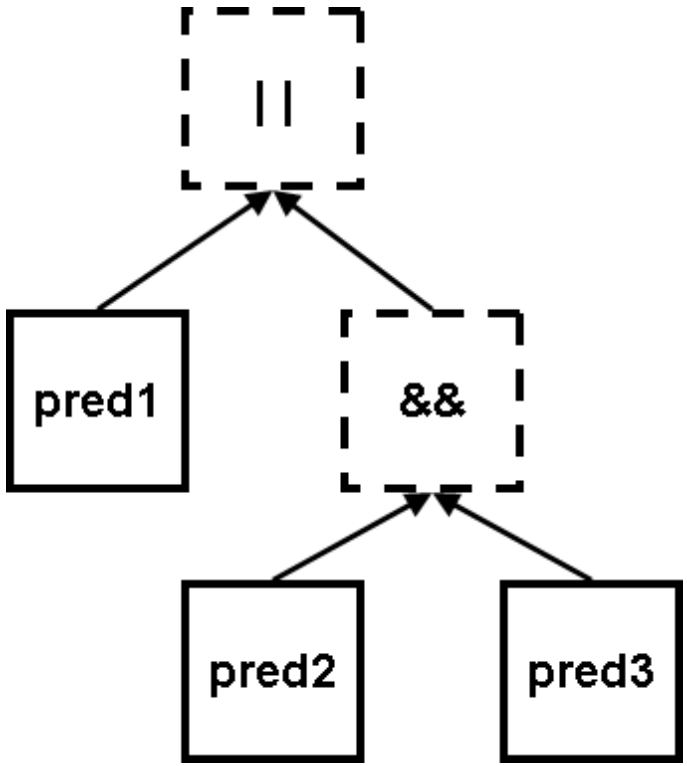
- 2 ways of viewing the problem
  - **PCDEval'**

- 2 ways of viewing the problem
  - **PCDEval**

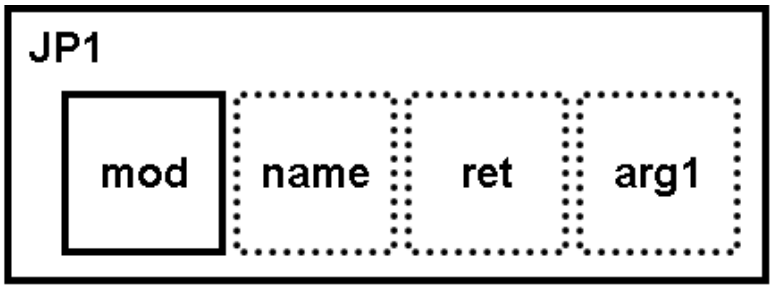


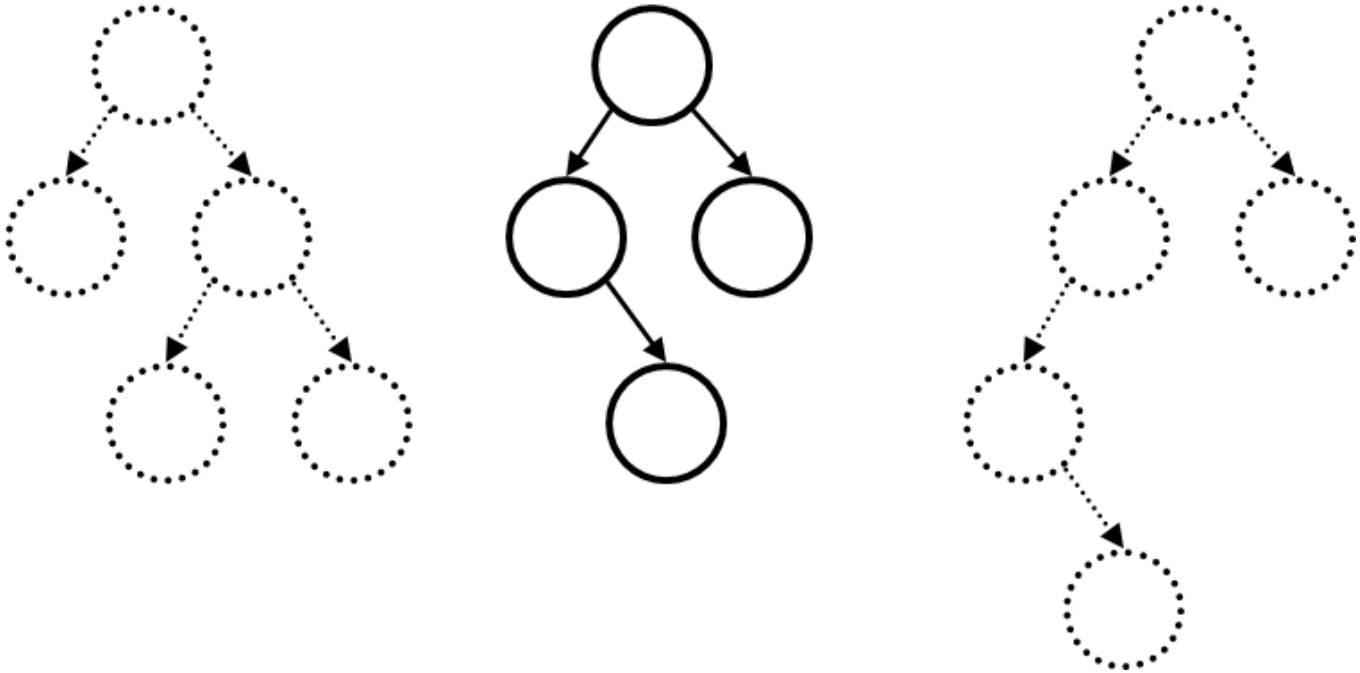
- Evaluation Algorithm overview
  - Order predicates for efficiency
  - Create PCD evaluation tree(s)
  - Add predicates to decision trees
  - Create links to parents

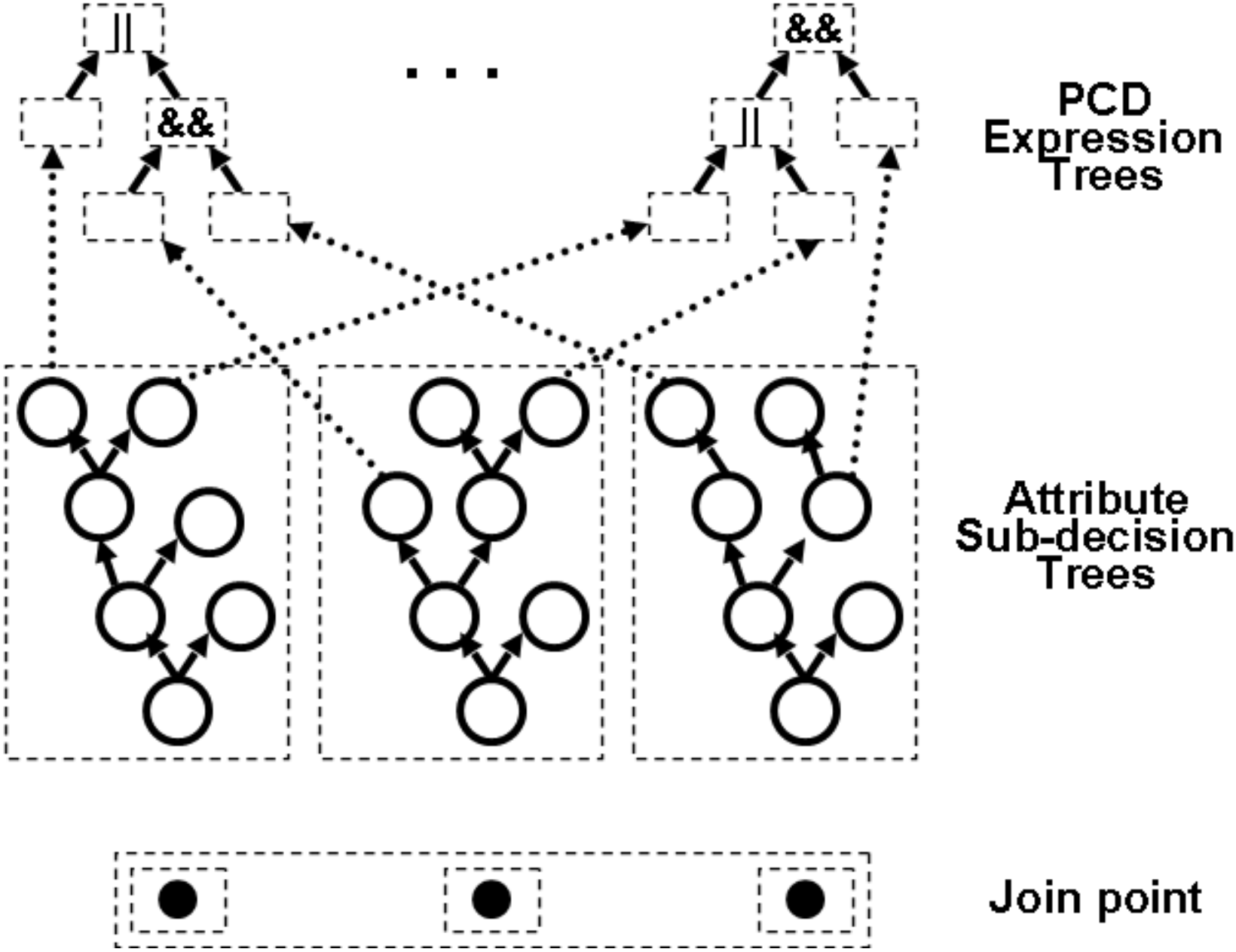
Consider the following PCD: ***Pred*1||(*Pred*2&&*Pred*3)**

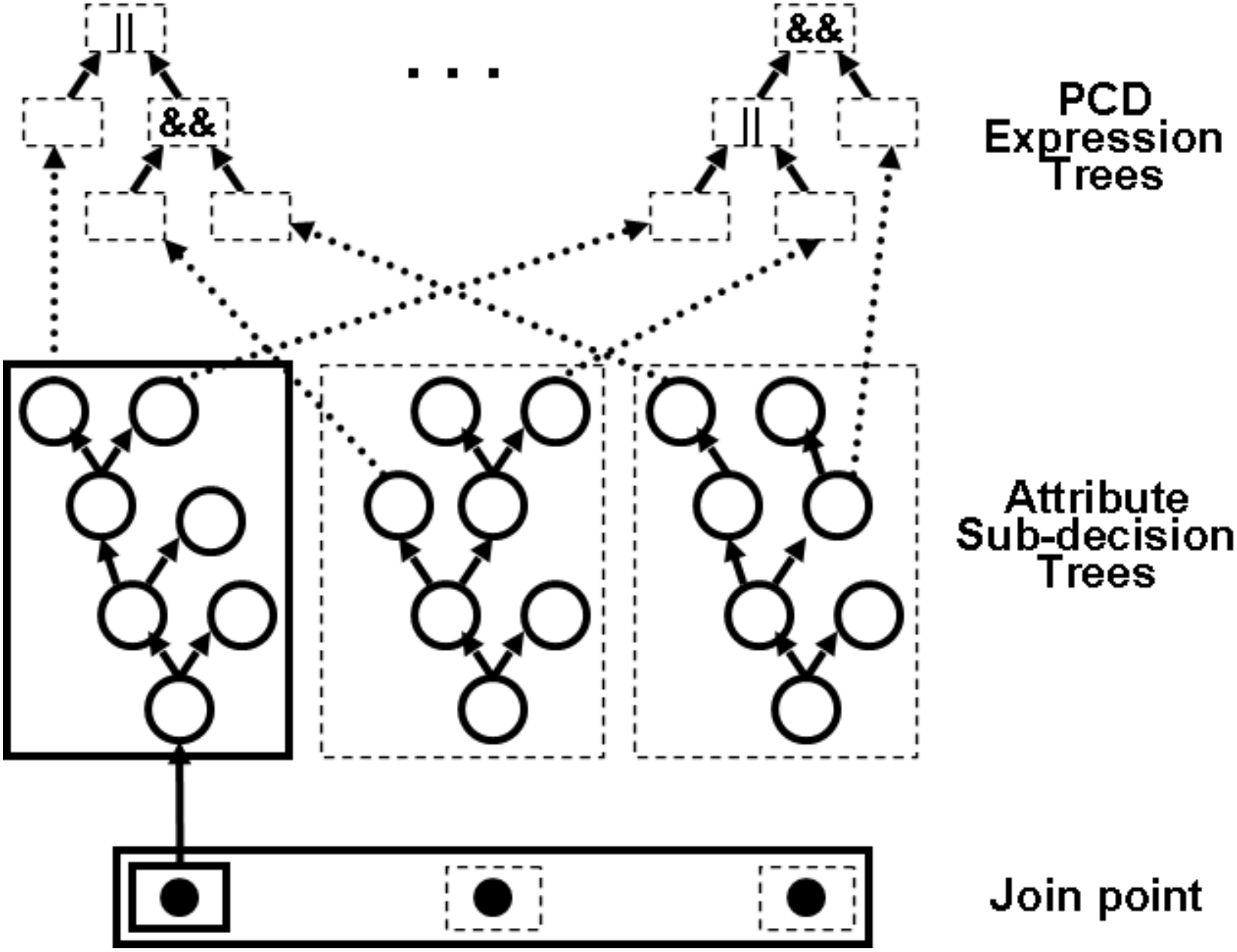


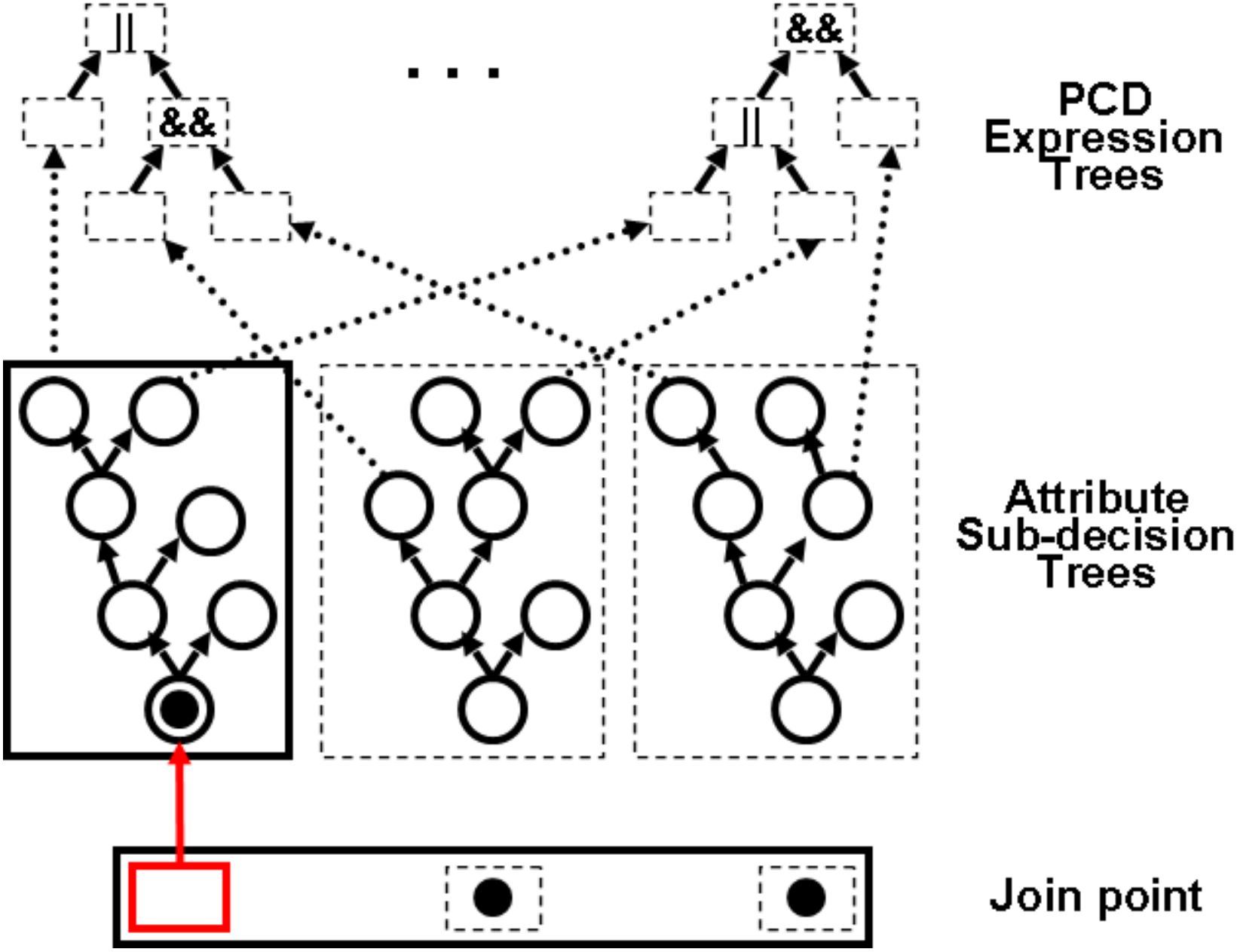
- Order predicates for efficiency
  - Modifiers are simple to match
  - Makes other decision-trees disjoint (smaller)

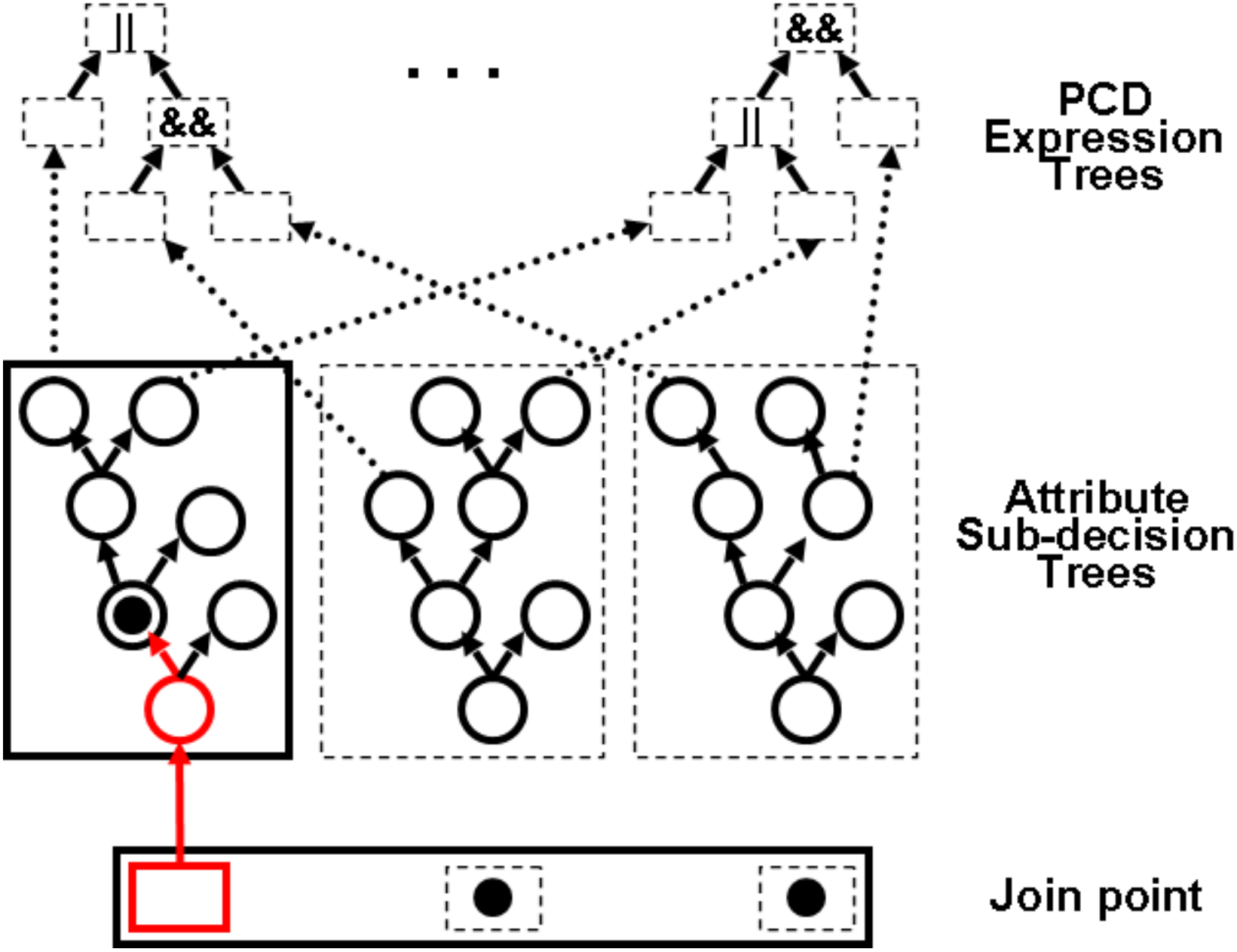

JP1
mod
name
ret
arg1

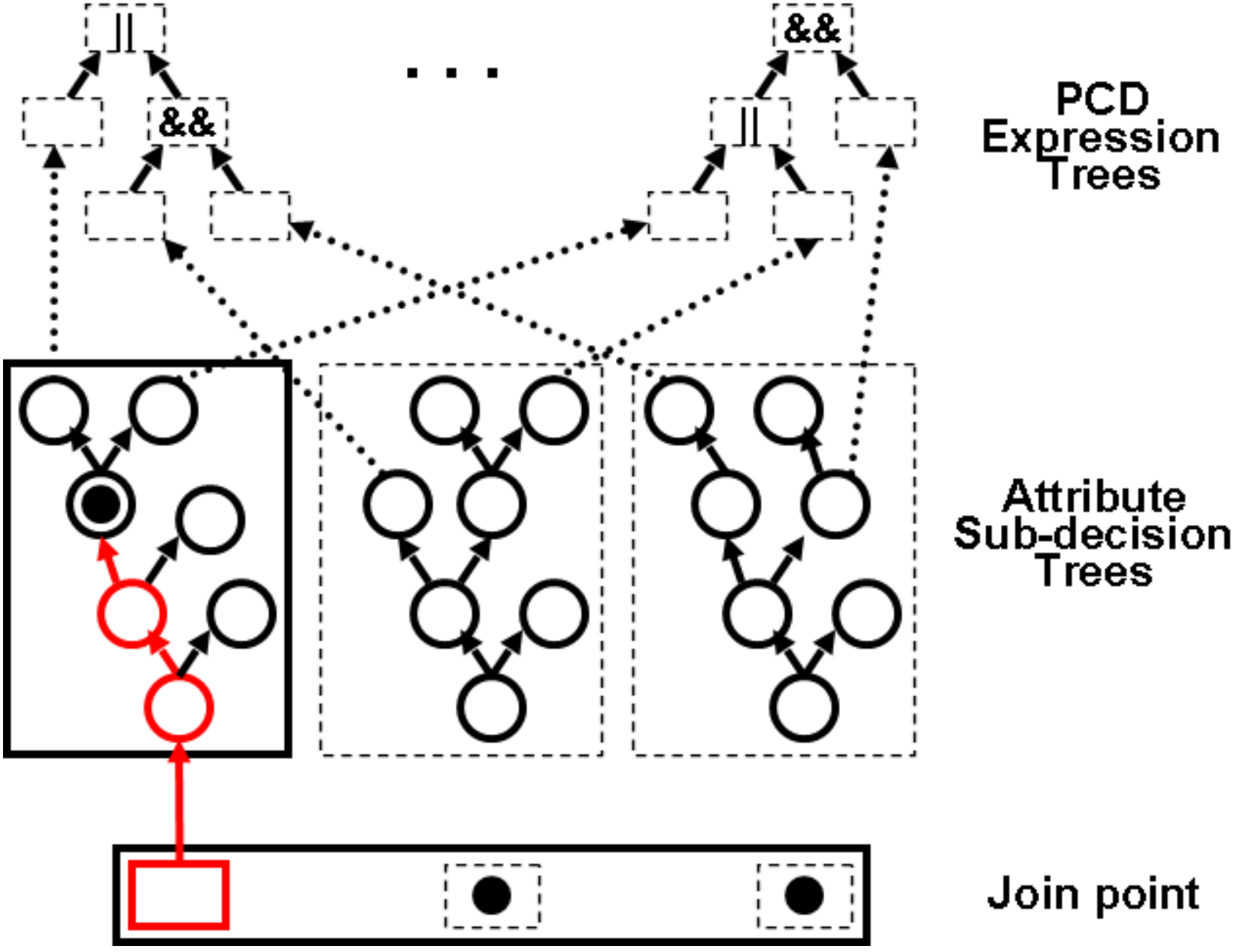


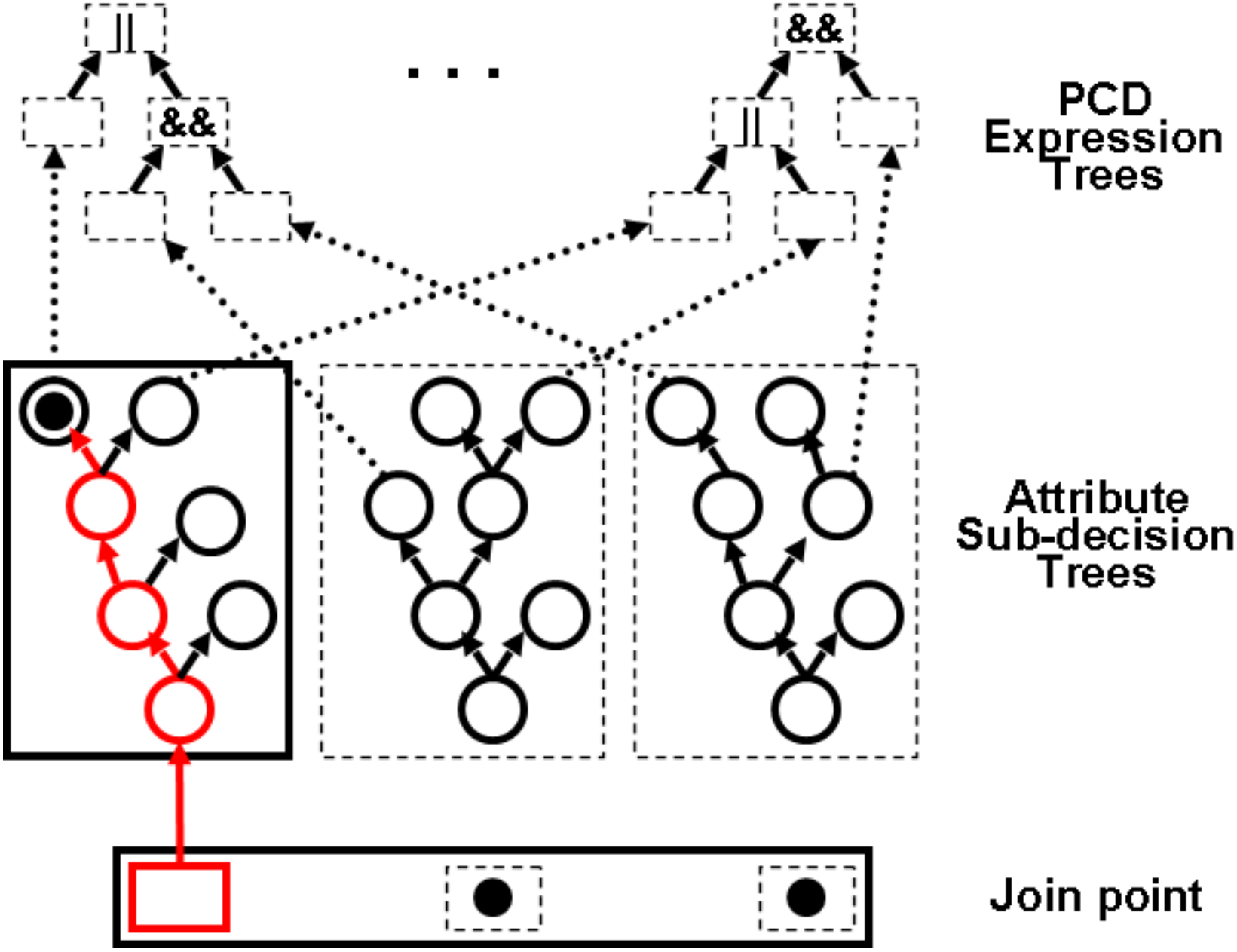

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

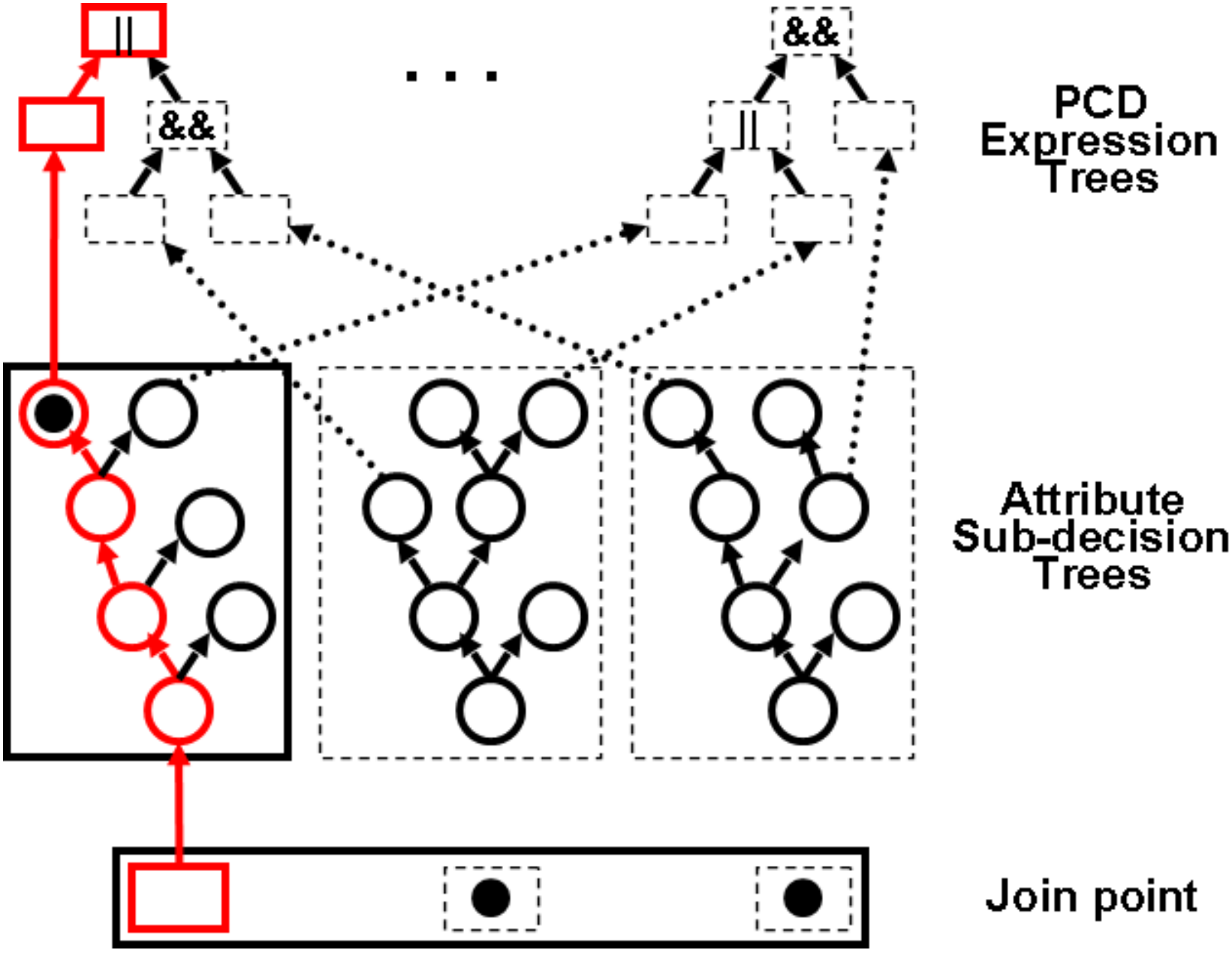

||
&&
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

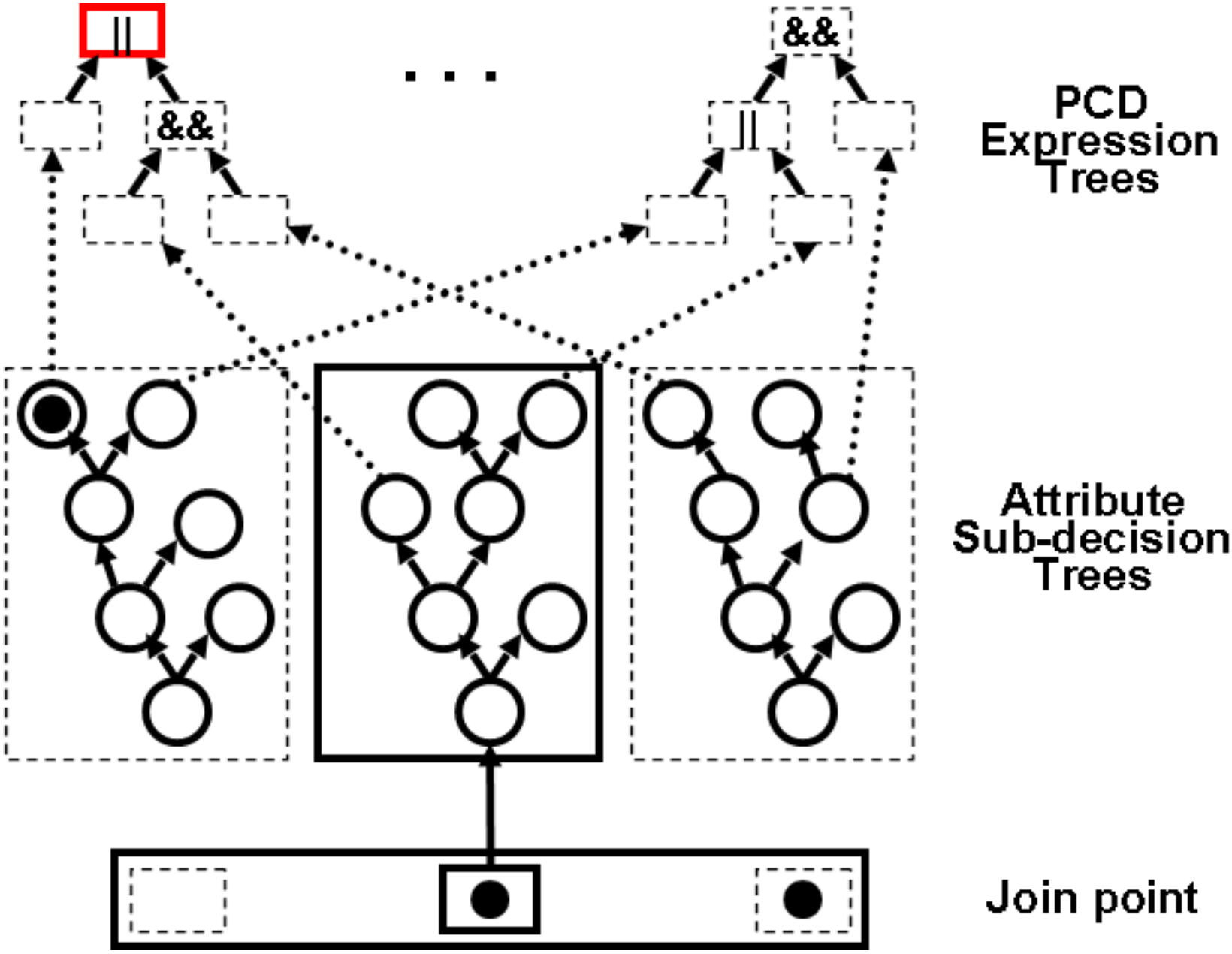

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

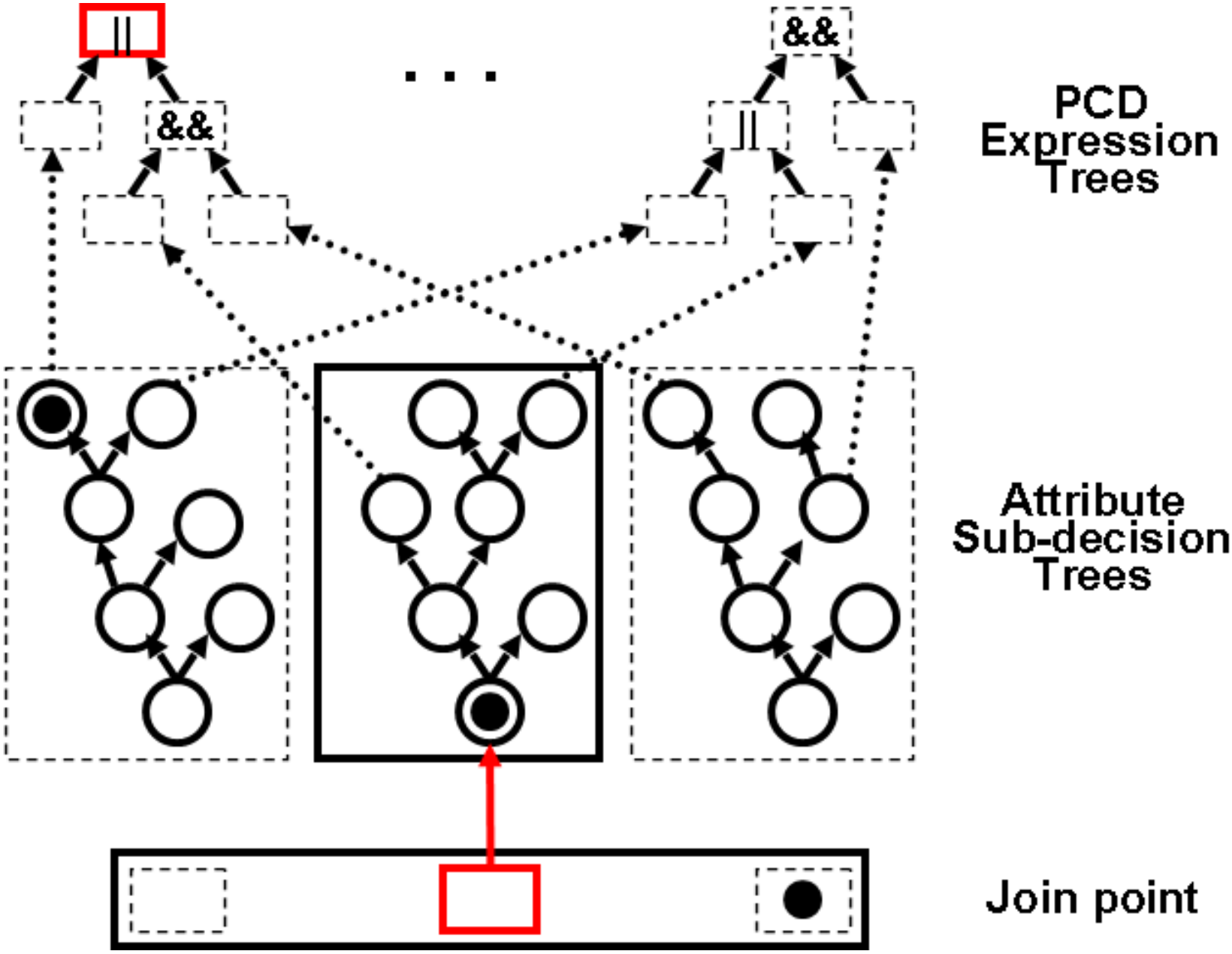

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

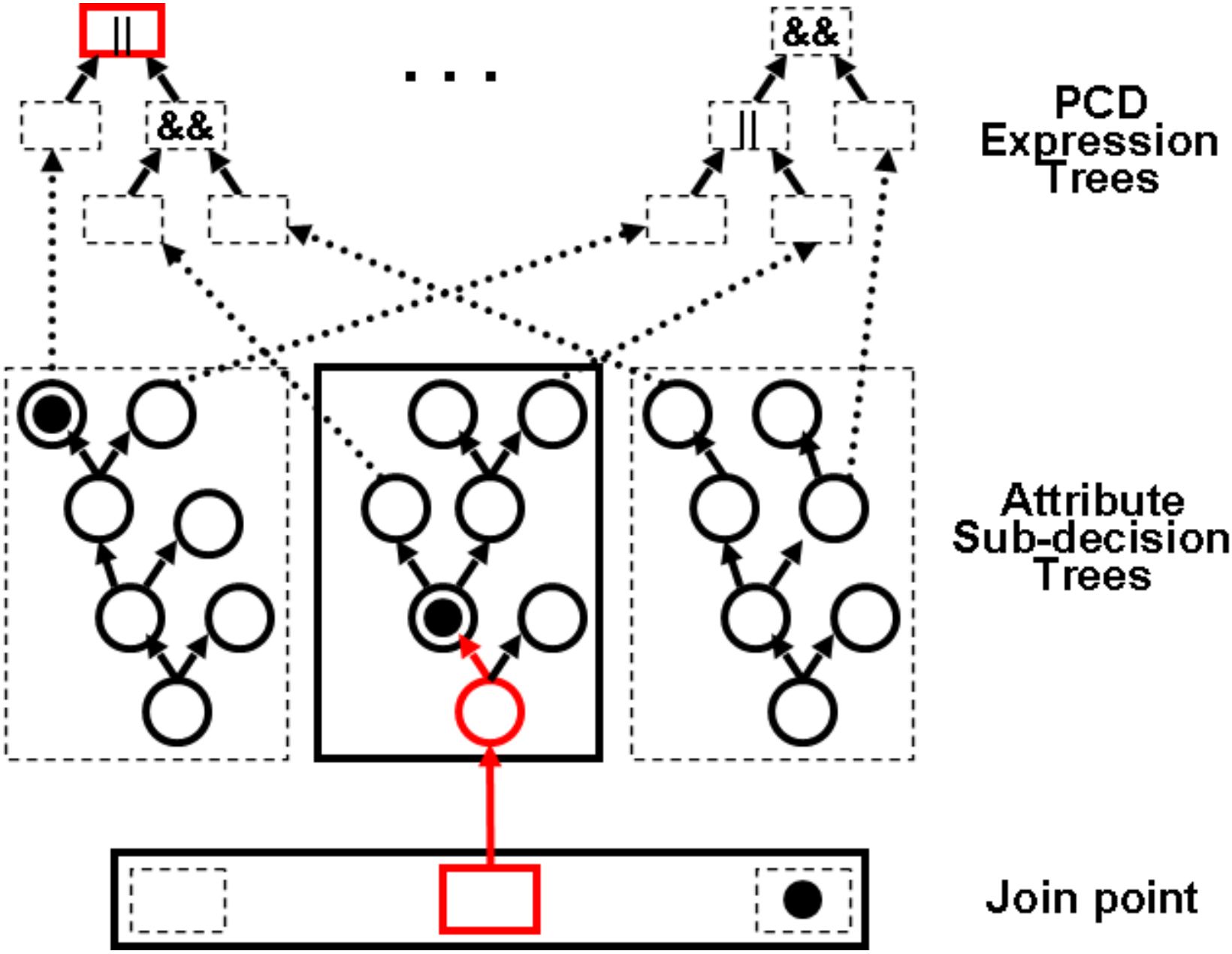

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

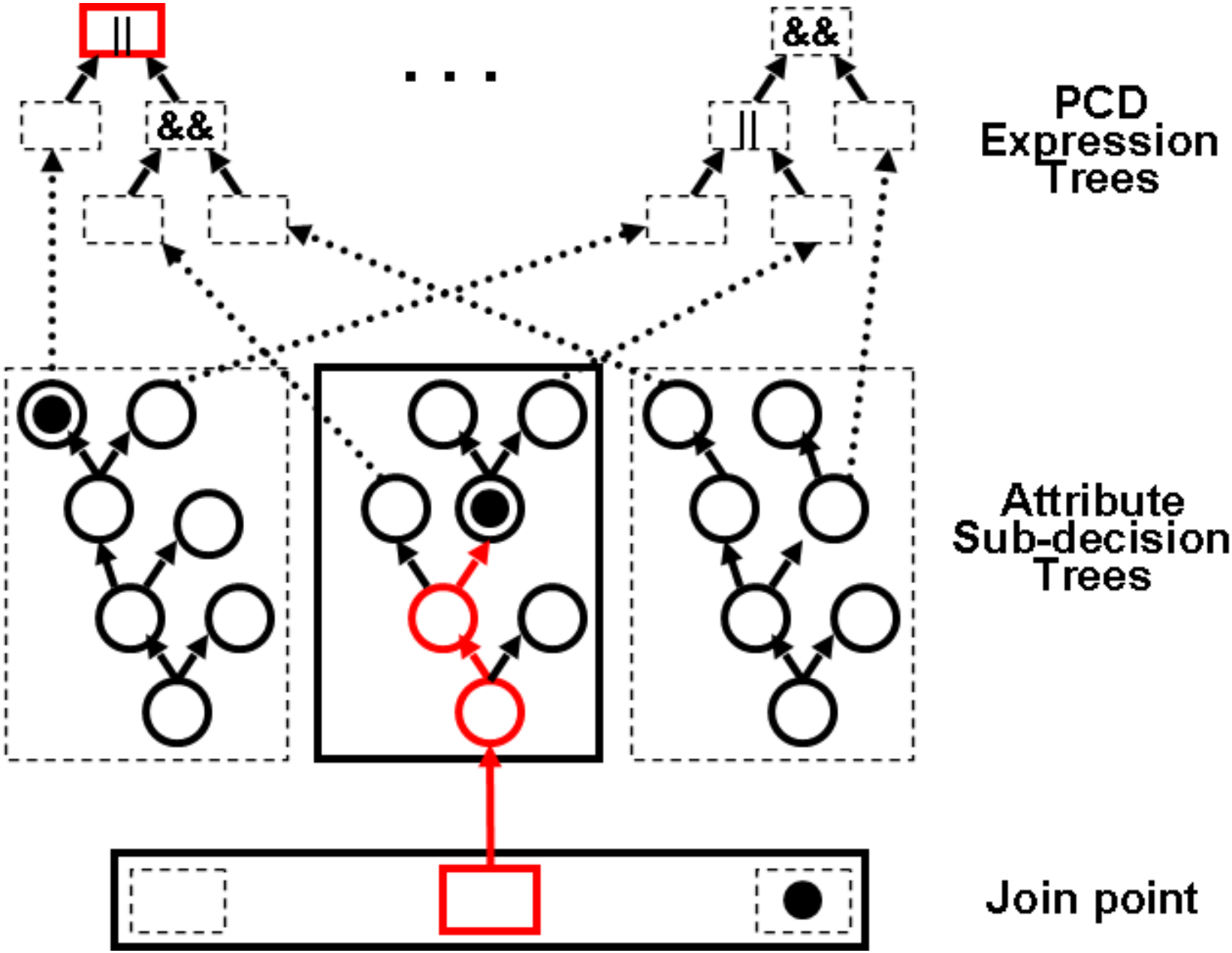

||
&&
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

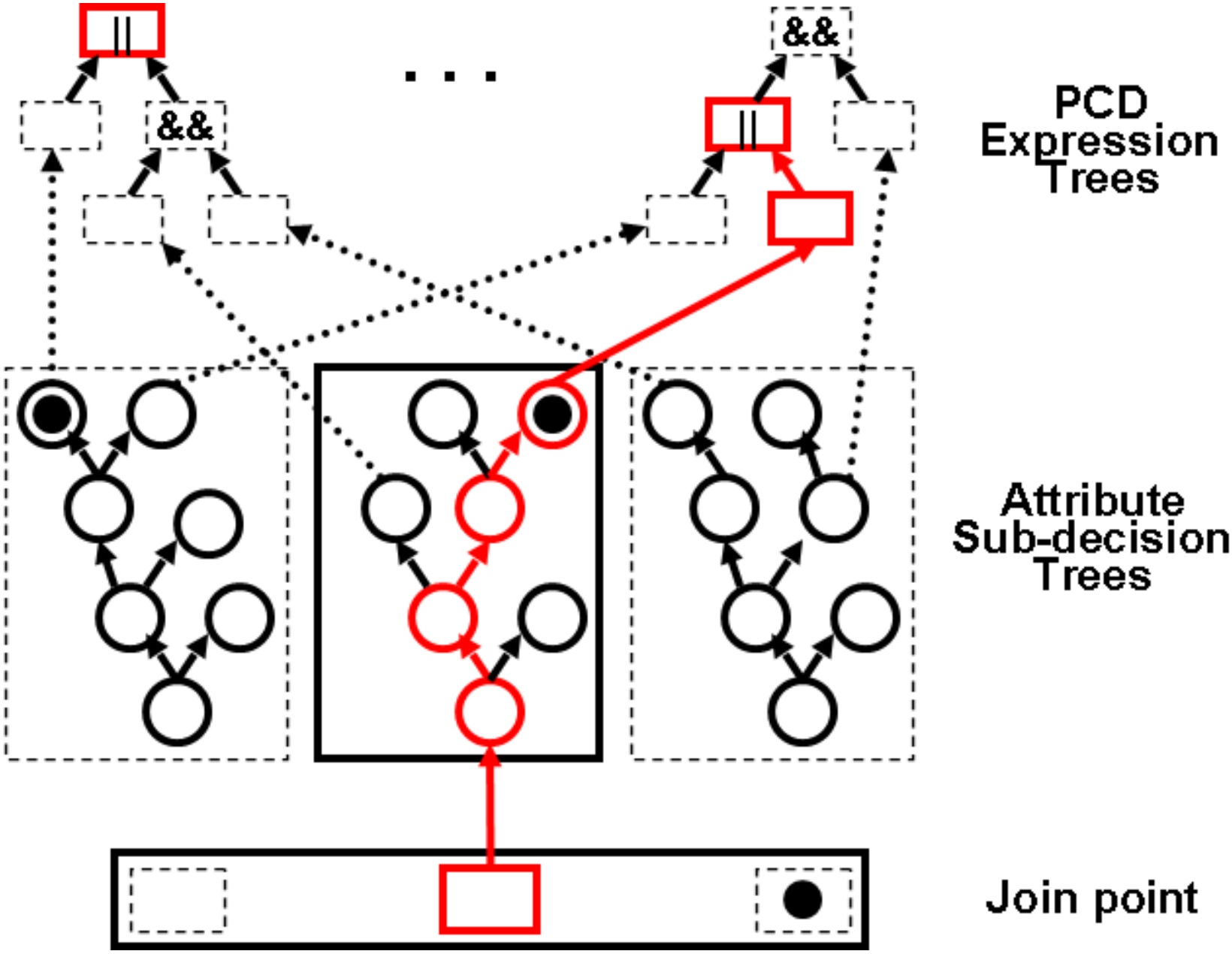

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

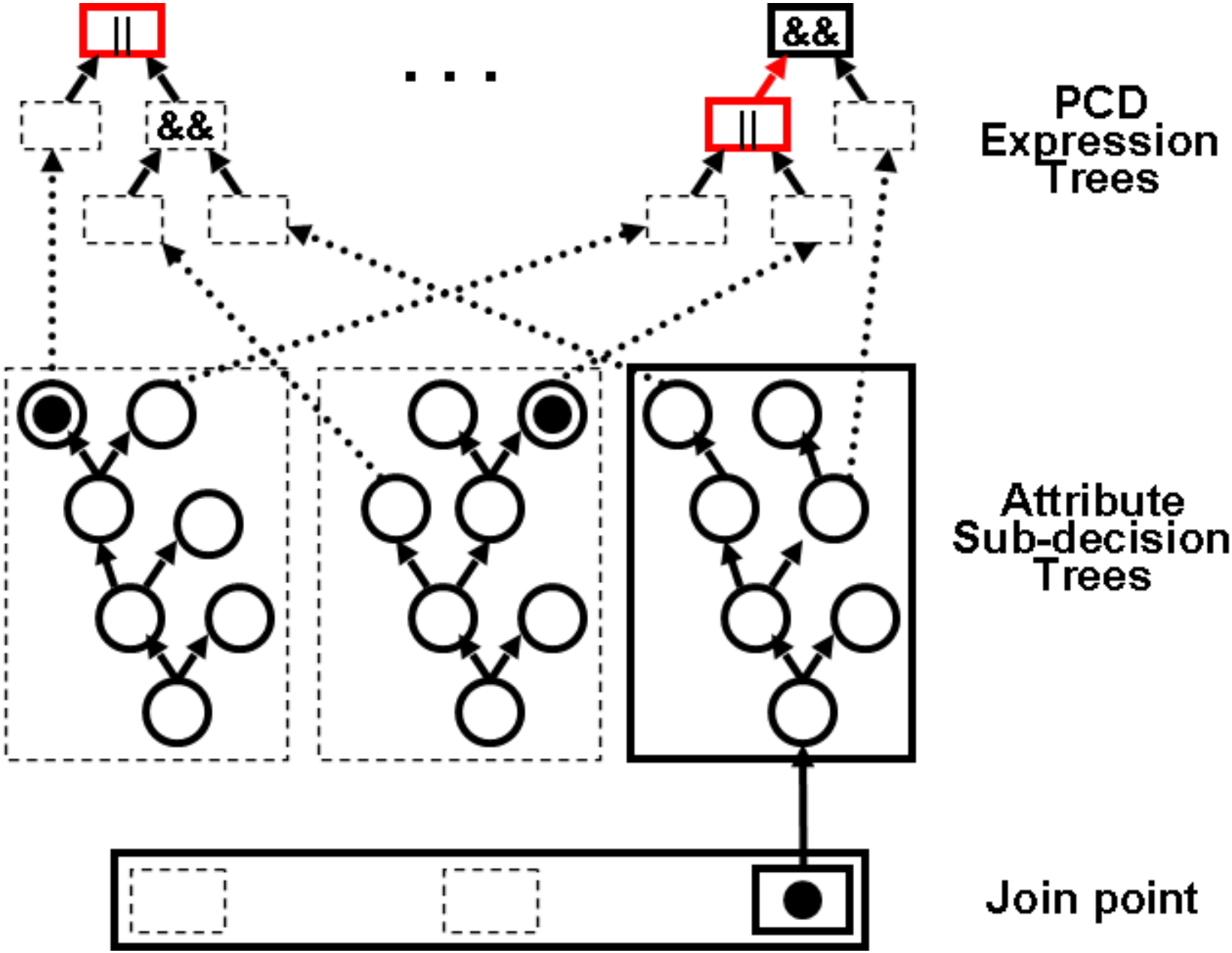

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

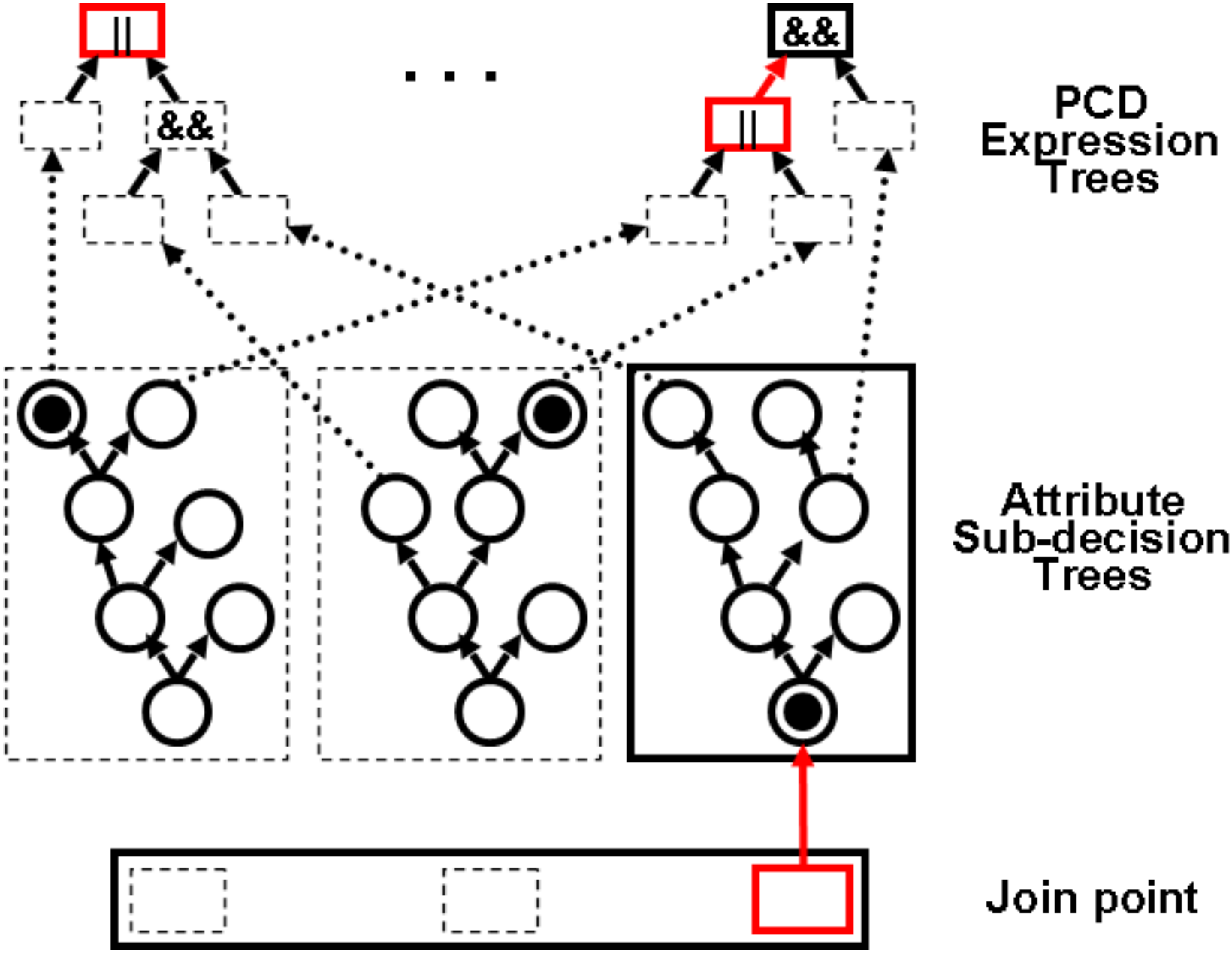

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

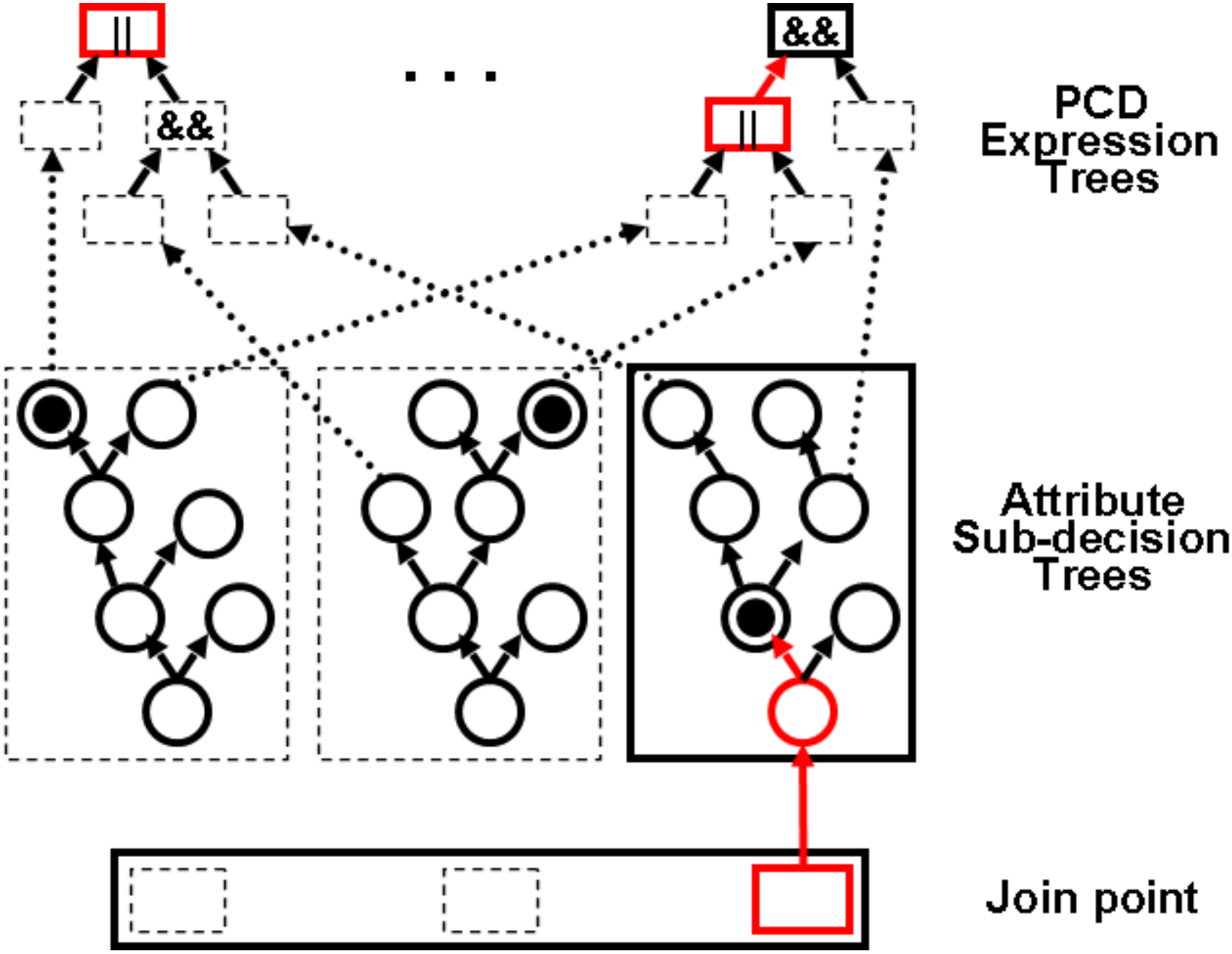

||
&&
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

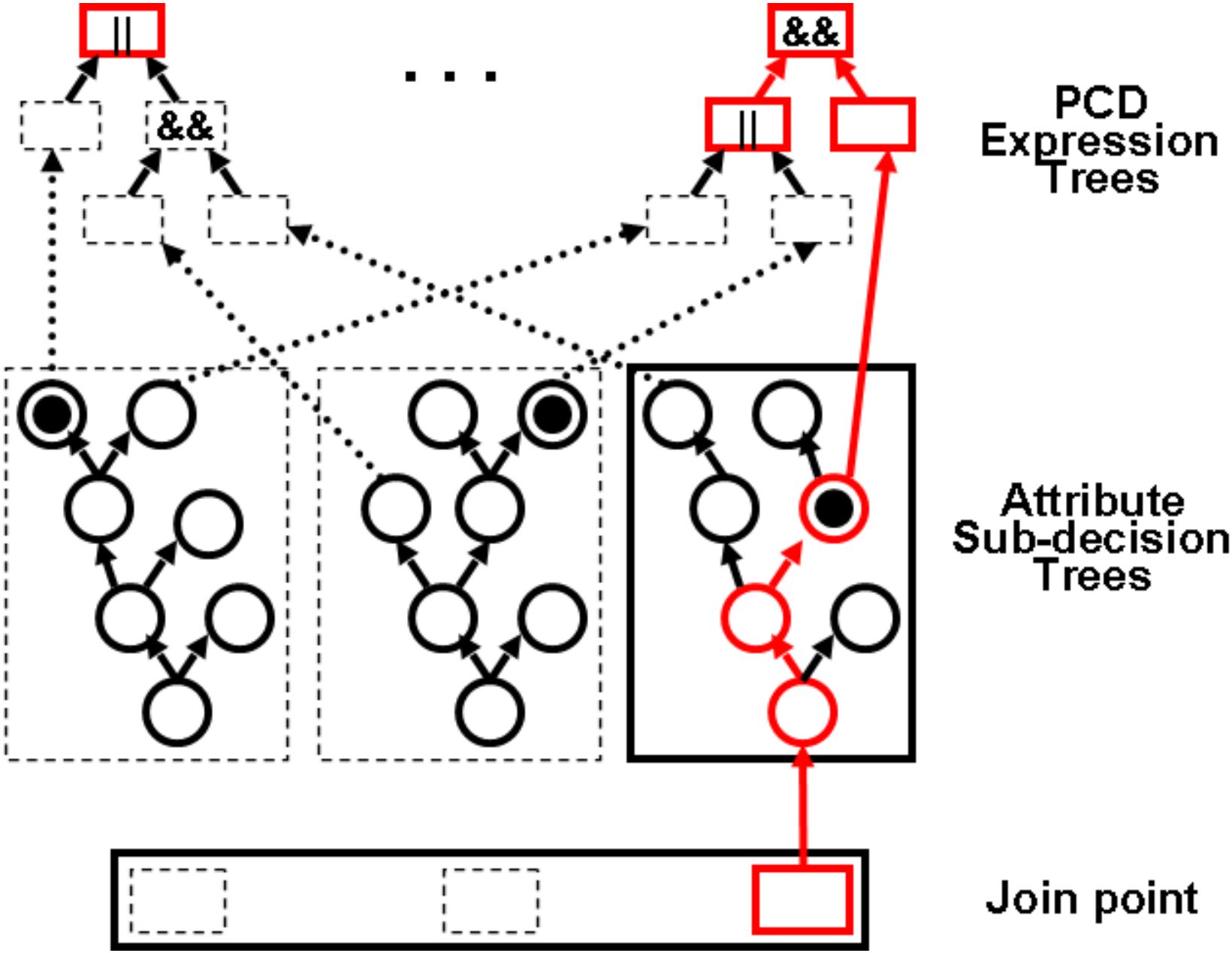

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

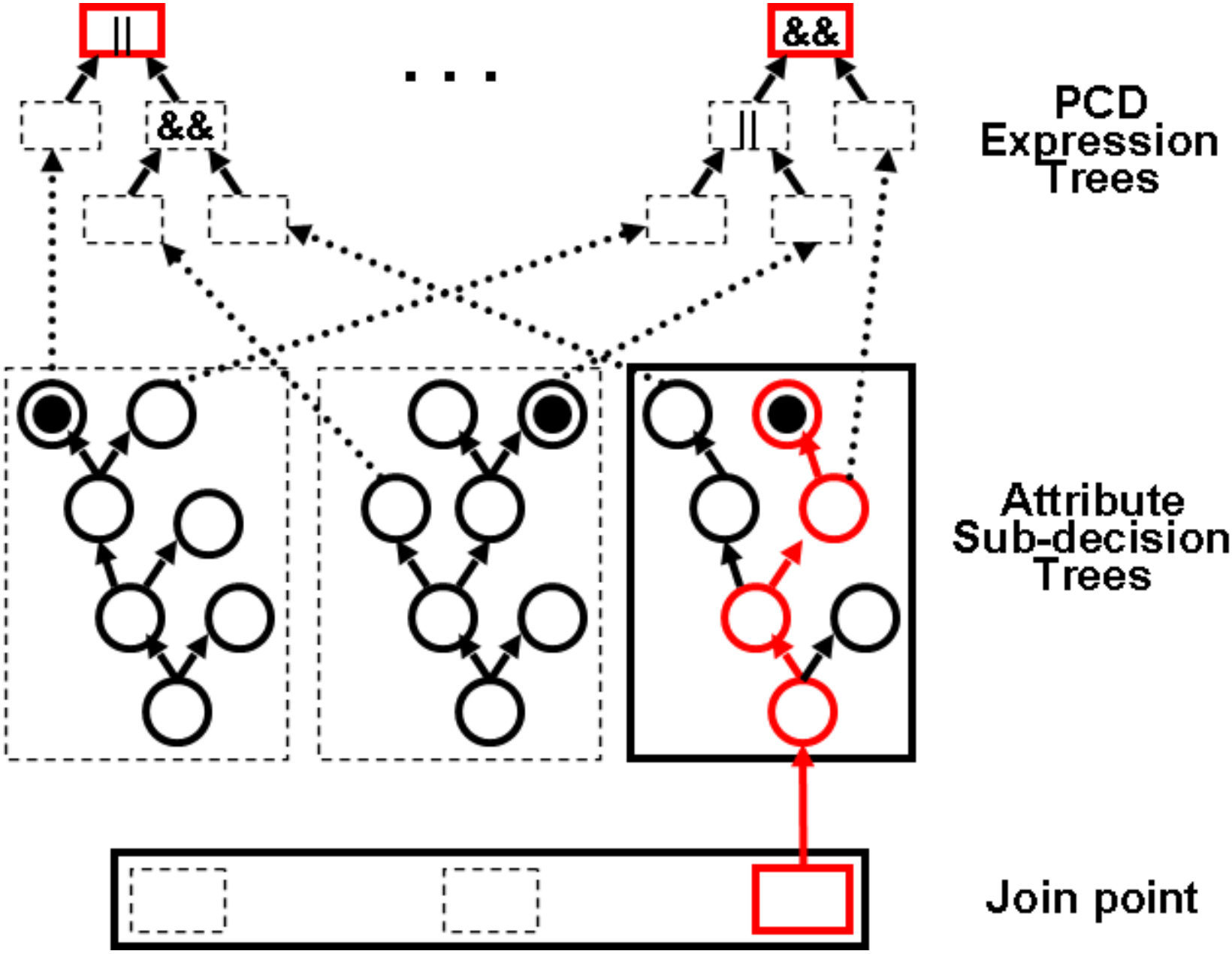

||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point


||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point


||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point


||
&&
. . .
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point


||
&&
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point


||
&&
&&
||
PCD
Expression
Trees
Attribute
Sub-decision
Trees
Join point

- Goal: Reduce size of decision-trees
- Idea: Partially evaluate predicates

- Known: $\boldsymbol{B} \lessdot \boldsymbol{C}$
- Evaluate: $\boldsymbol{A} \lessdot \boldsymbol{B}$, $\boldsymbol{A} \lessdot \boldsymbol{C}$

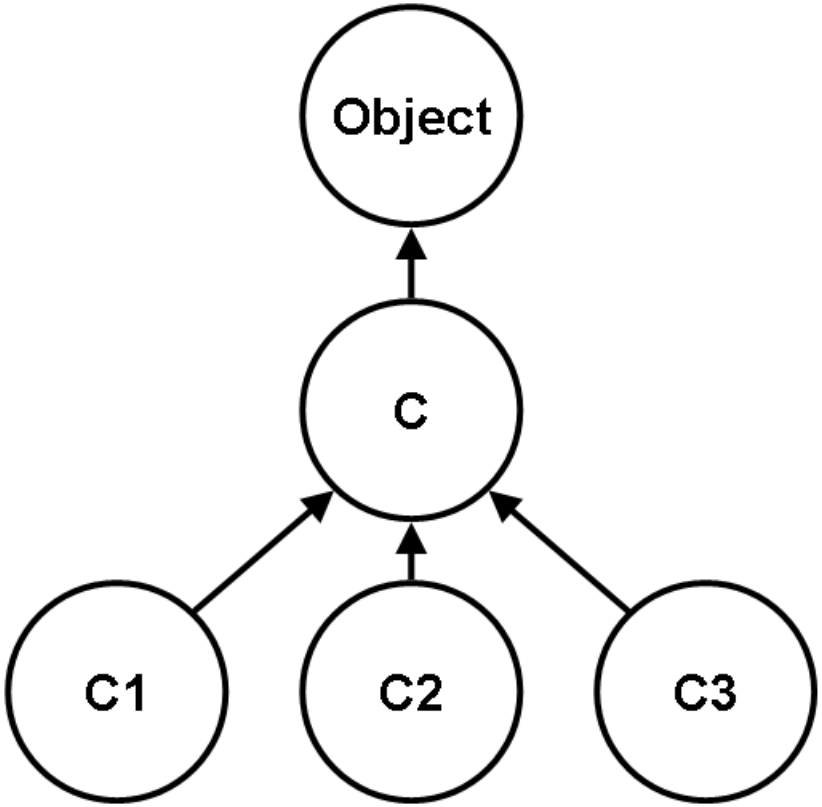
- Known: $B \lessdot C$
- Evaluate: $A \lessdot B$, $A \lessdot C$
- $A \lessdot B \wedge B \lessdot C$

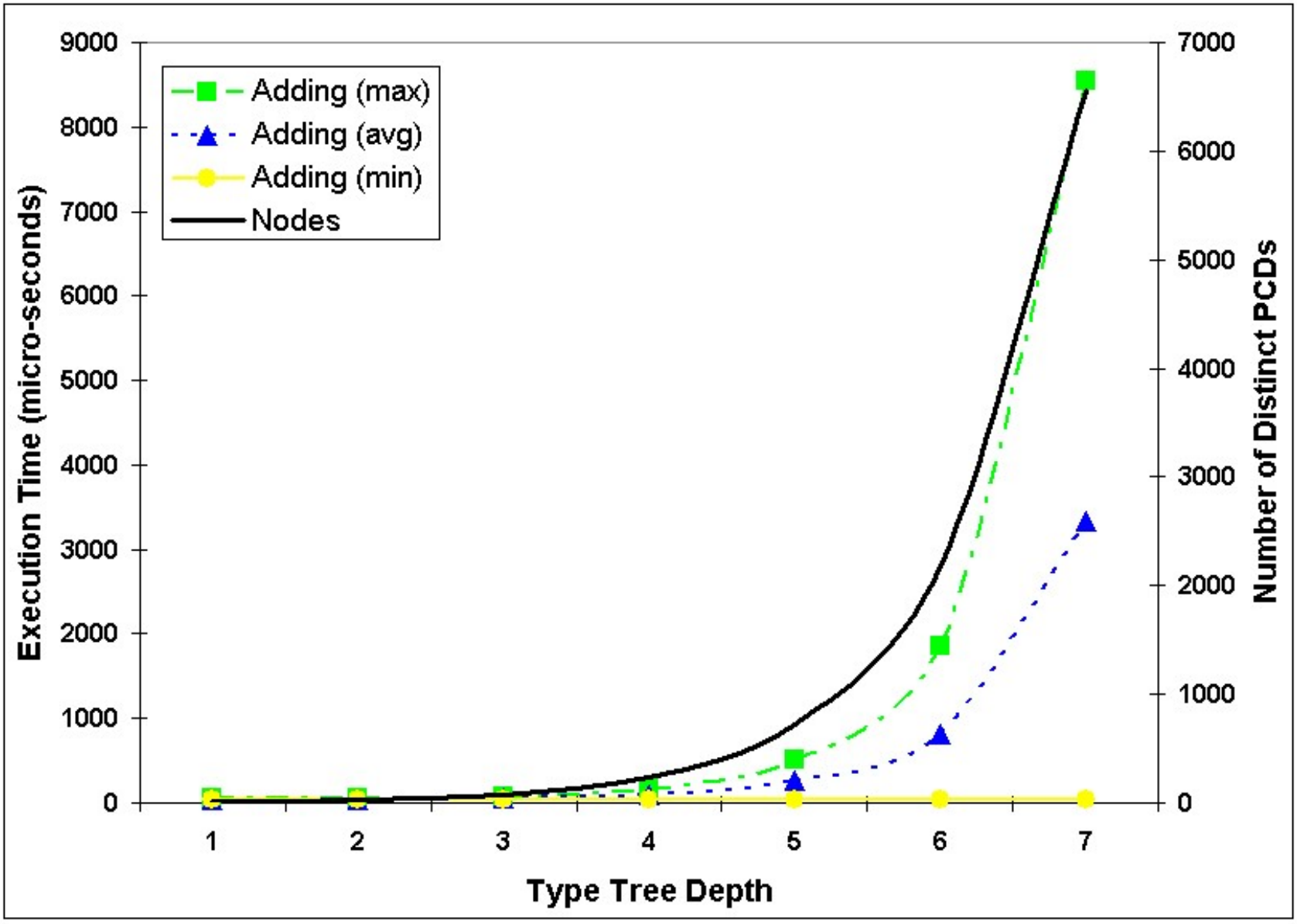
- Known: $\boldsymbol{B} \lessdot \boldsymbol{C}$
- Evaluate: $\boldsymbol{A} \lessdot \boldsymbol{B}$, $\boldsymbol{A} \lessdot \boldsymbol{C}$
- $\boldsymbol{A} \lessdot \boldsymbol{B} \wedge \boldsymbol{B} \lessdot \boldsymbol{C} \longrightarrow \boldsymbol{A} \lessdot \boldsymbol{C}$

- Known: $B \lessdot C$
- Evaluate: $A \lessdot B$, $A \lessdot C$

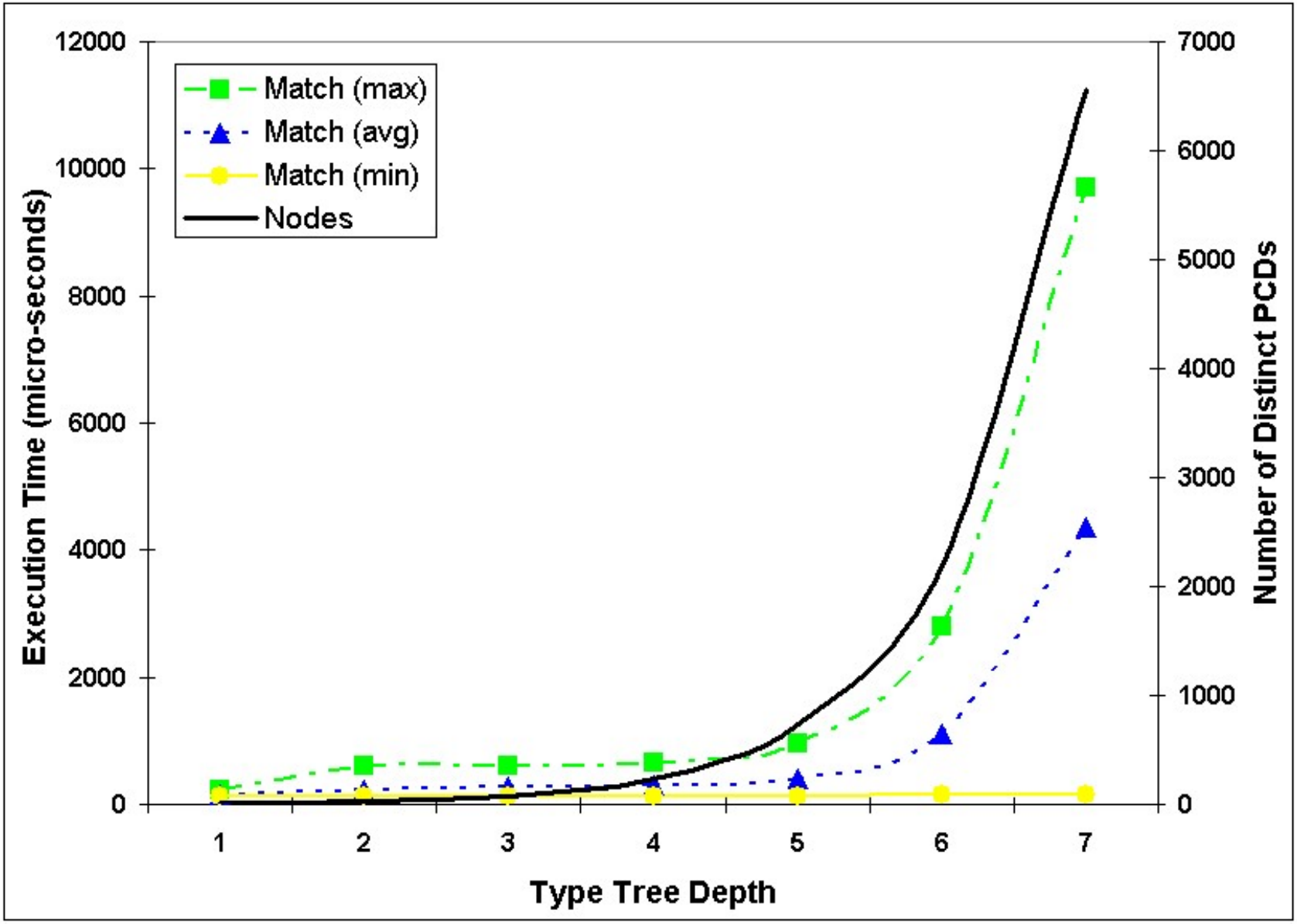
- $A \lessdot B \wedge B \lessdot C \longrightarrow A \lessdot C$
- Partially Evaluate: $A \lessdot B$

- Created implementation in Nu virtual machine
- Bind and Remove primitives for deploying/un-deploying advice
- Synthetic micro-benchmark
  - Measures time to Bind (add to trees) and match
  - Varies type hierarchy depth


Object
C
C1
C2
C3



Old matching code - ($\sim$40μs constant)



Old matching code - average case 3-50x slower
worst case 3-88x slower

Related Work

- Efficient Matching Techniques
- Dynamic Residue Evaluation
- Partial Evaluation Techniques

Future Work

- Example Implementation(s)
- Real-world Evaluations

- Motivation: Dynamic PCD Evaluation
  - PCDs arrive dynamically
  - PCDs might be removed later
  - Matching the whole (loaded) system against a PCD is too slow
- Approach: Decision-tree based Matching
  - Order evaluations based on cost
  - Partially evaluate wherever possible
- Technical Contributions:
  - Formalization of the PCD Evaluation problem
  - Algorithms using Decision-tree structures for faster matching
  - Use of implication relationships for partial evaluation of type predicates

Questions?

http://www.cs.iastate.edu/~nu/

```
C.run
C1.run
C2.run
C3.run
..

class C {
  public static void run() {
    measure { Bind.. //to methods returning C1 }
    measure { Bind.. //to methods returning C2 }
    measure { Bind.. //to methods returning C3 }

    measure { C1.testMethod }
    measure { C2.testMethod }
    measure { C3.testMethod }
  }
  public C testMethod() { return NULL }
}
```