On Compiling CNFs into Structured Deterministic DNNFs

Friedrich Slivovsky

joint work with Simone Bova, Florent Capelli, and Stefan Mengel

Model Counting (#SAT)

Instance: A propositional formula F in CNF

Problem: Count the satisfying assignments of F

Model Counting (#SAT)

Instance: A propositional formula F in CNF

Problem: Count the satisfying assignments of F

#P-complete for HORN, monotone 2CNF

Model Counting (#SAT)

Instance: A propositional formula F in CNF

Problem: Count the satisfying assignments of F

#P-complete for HORN, monotone 2CNF

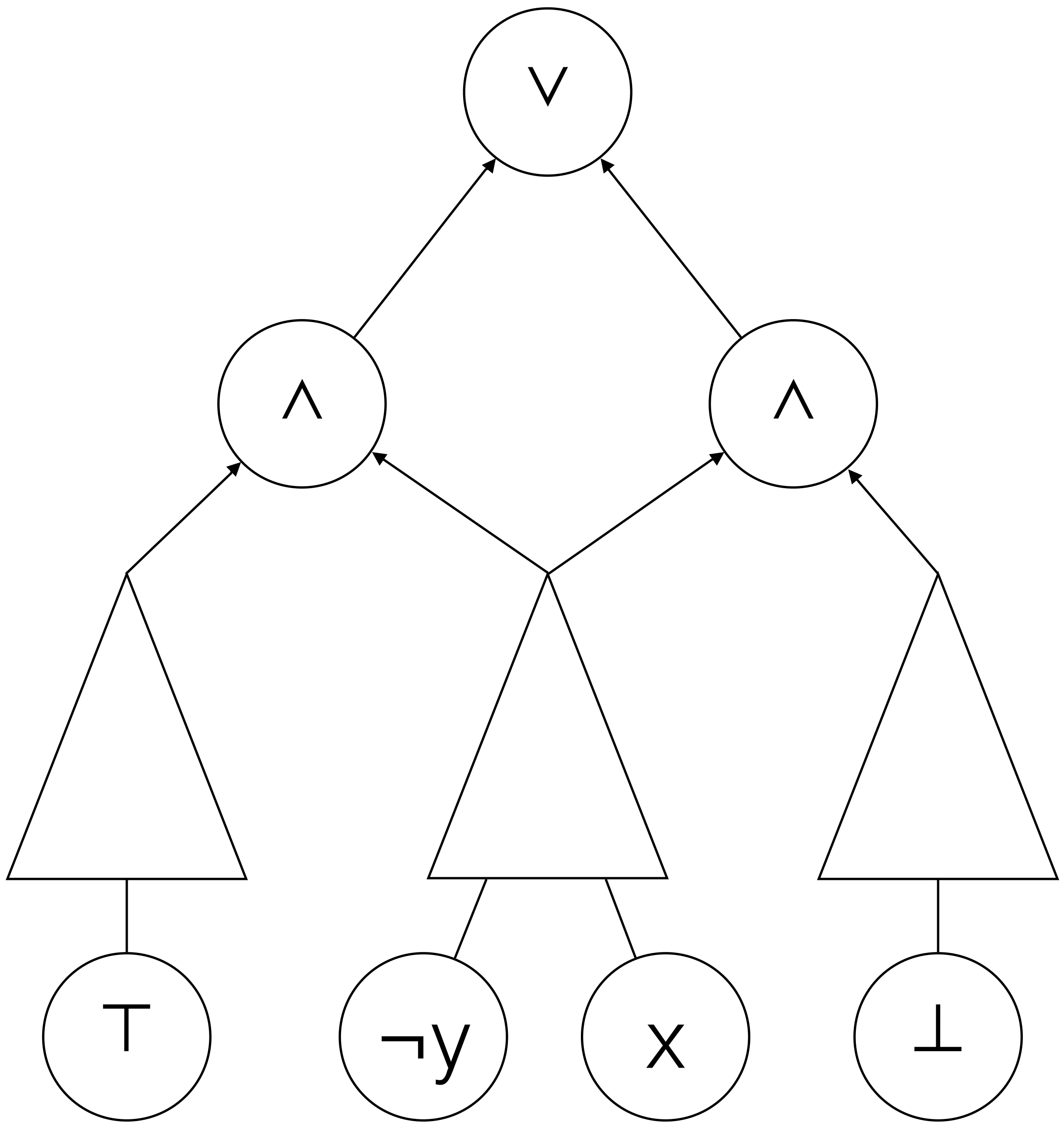
structural restrictions often yield tractability

Previous talk:
exact model counters implicitly compile CNFs into **decision** DNNFs

Previous talk:
exact model counters implicitly compile CNFs into **decision** DNNFs

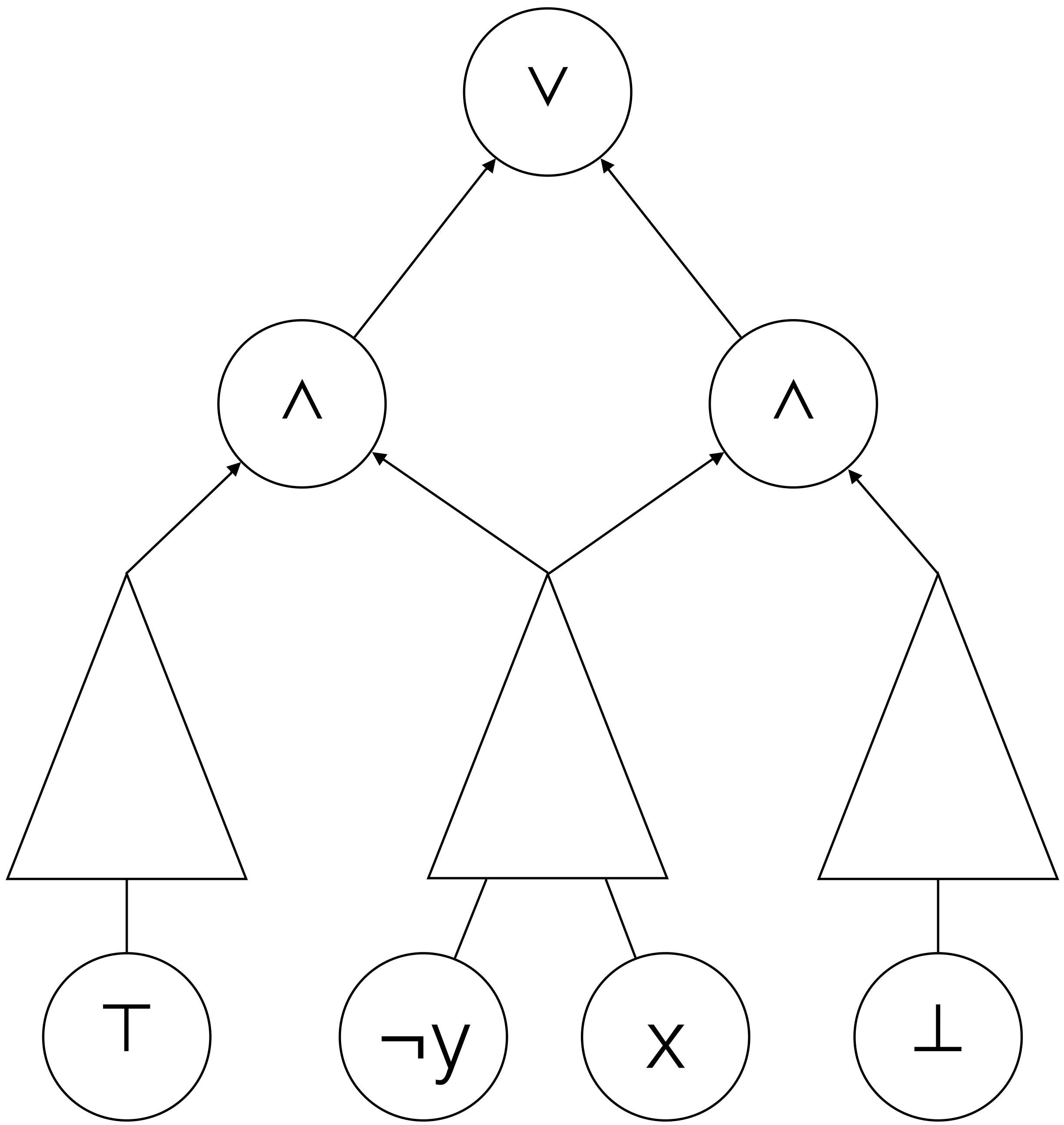
This talk:
compilation of CNFs into **structured deterministic** DNNFs based on new model counting algorithms

deterministic DNNF

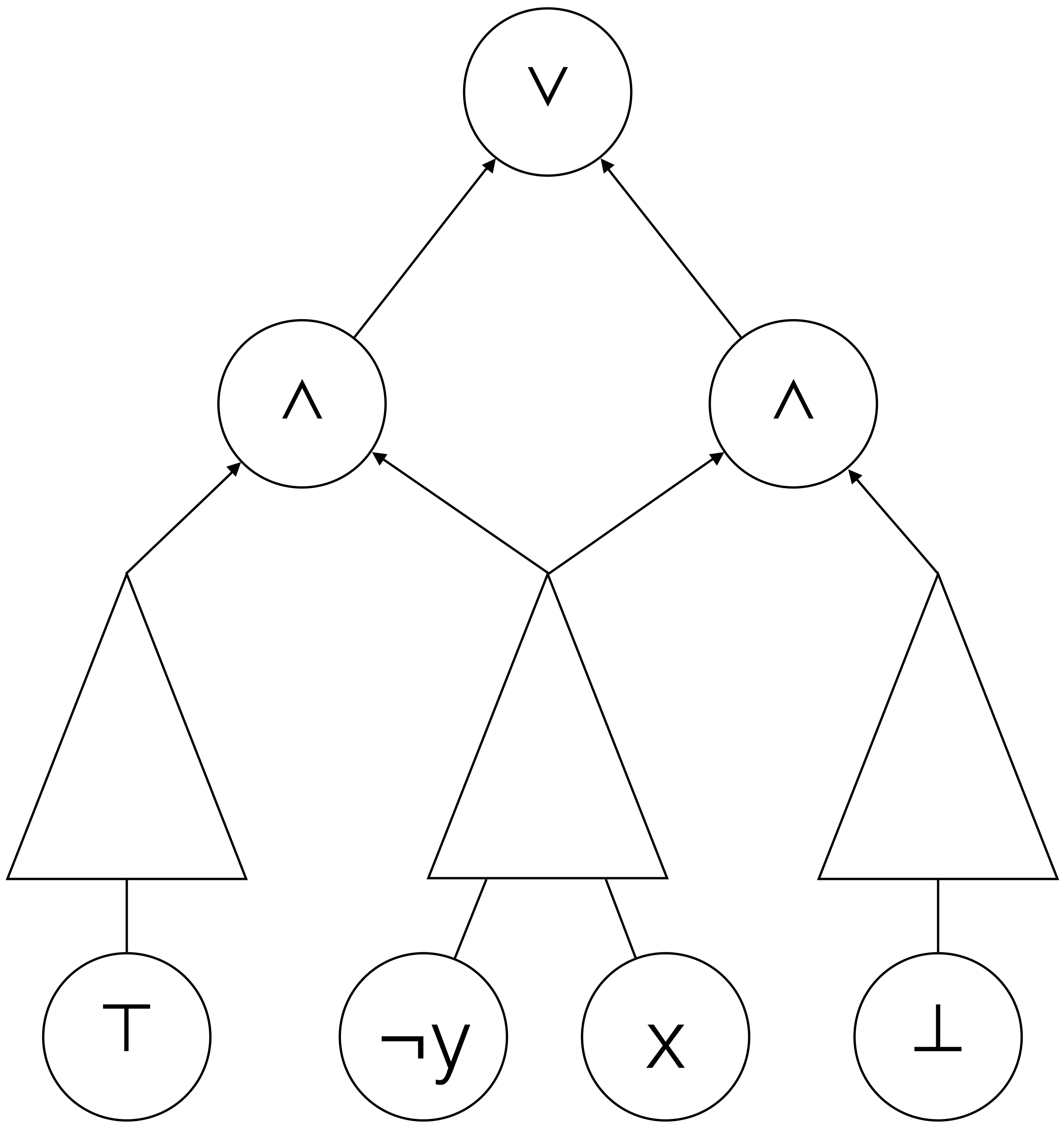


deterministic DNNF

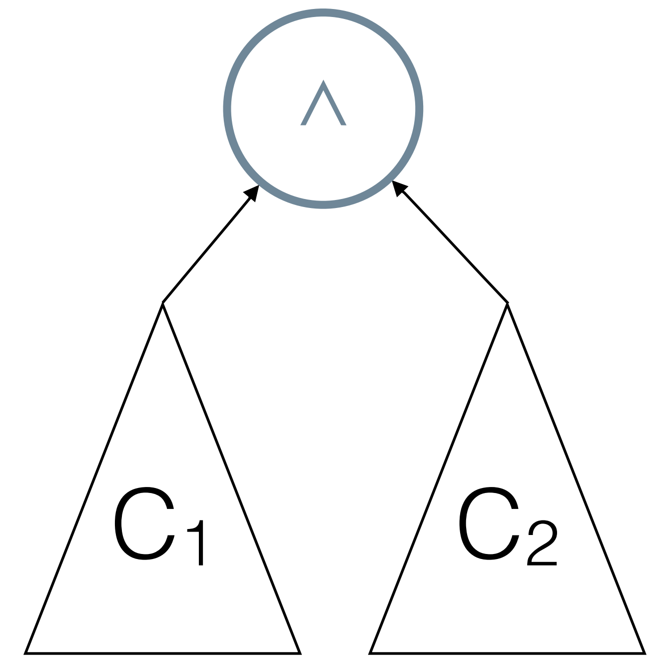
decomposable



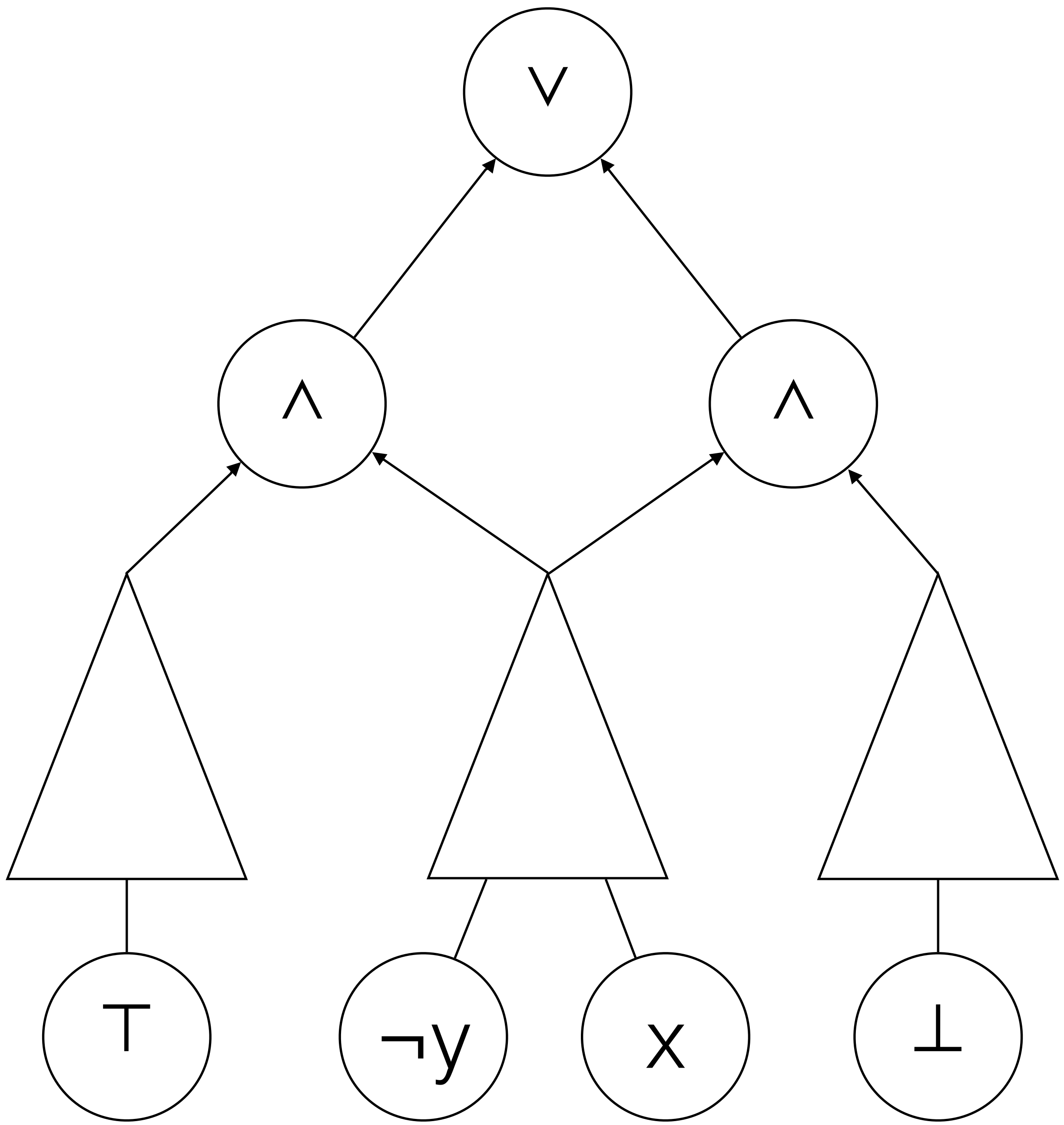
deterministic DNNF



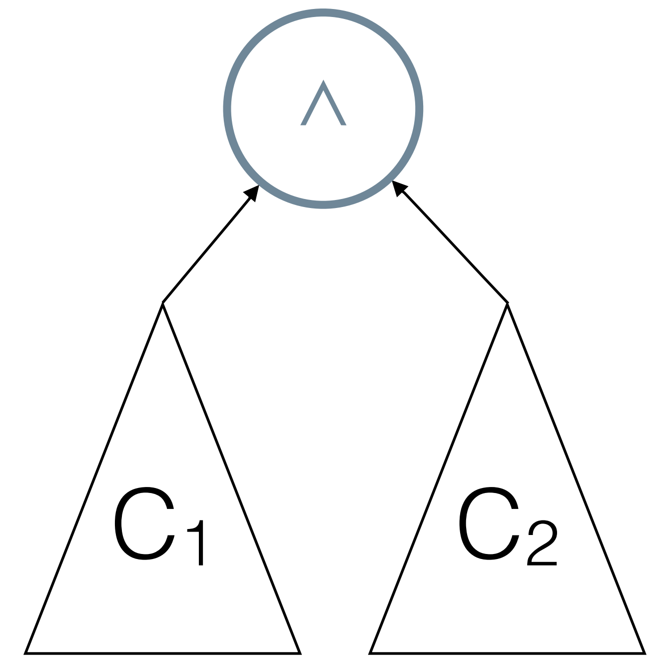
decomposable



deterministic DNNF

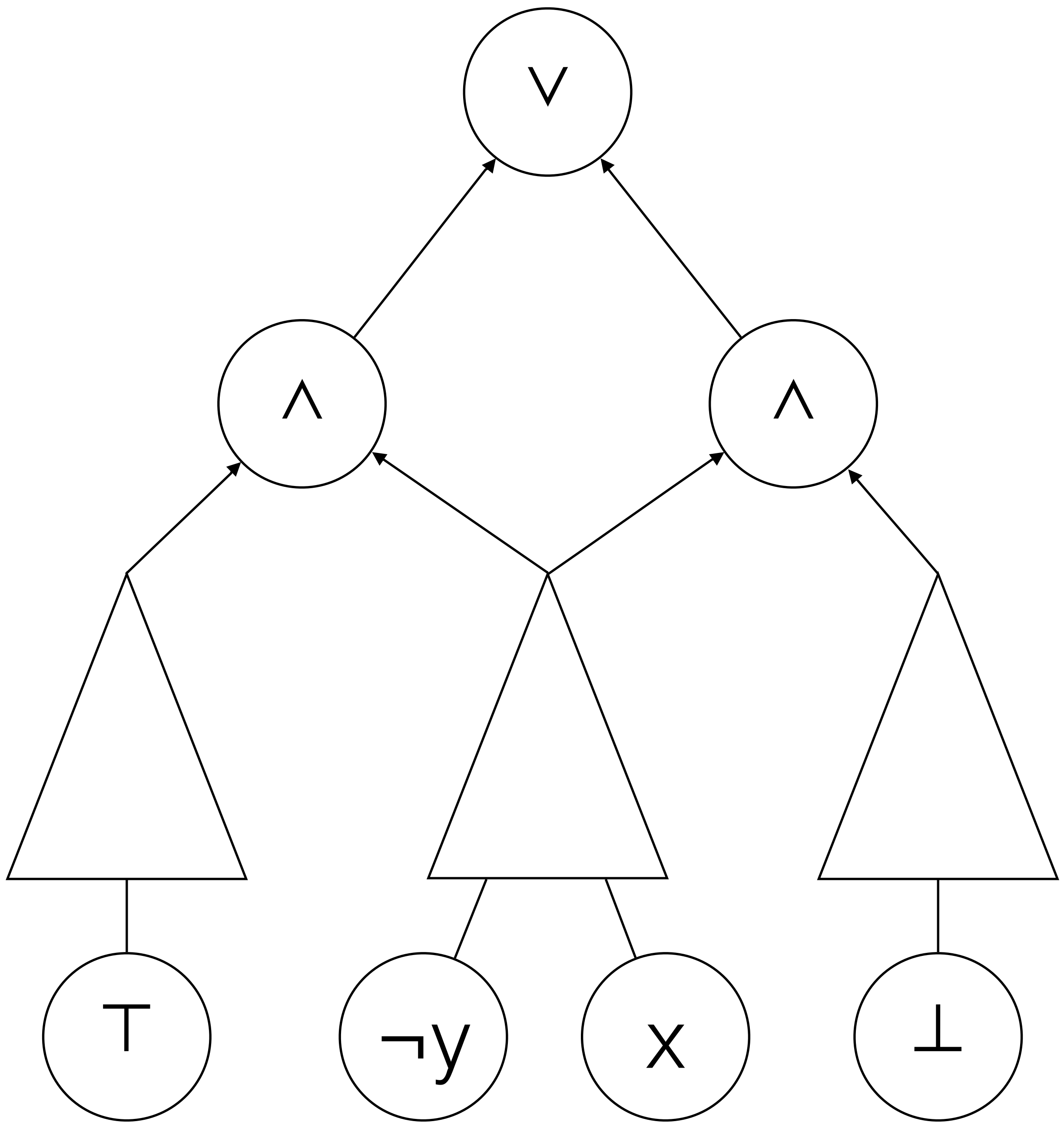


decomposable



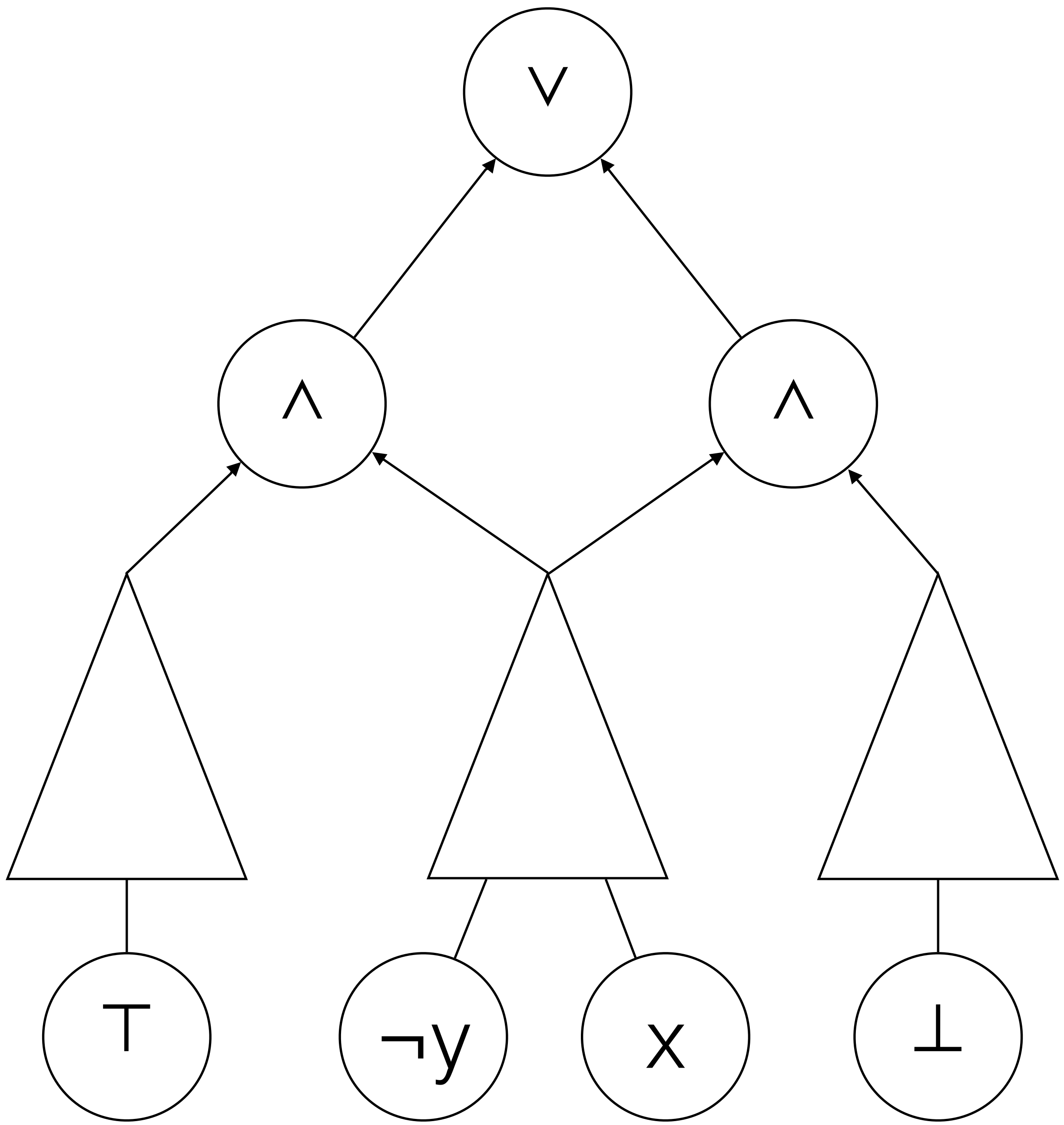
$var(C_1) \cap var(C_2) = \varnothing$

deterministic DNNF

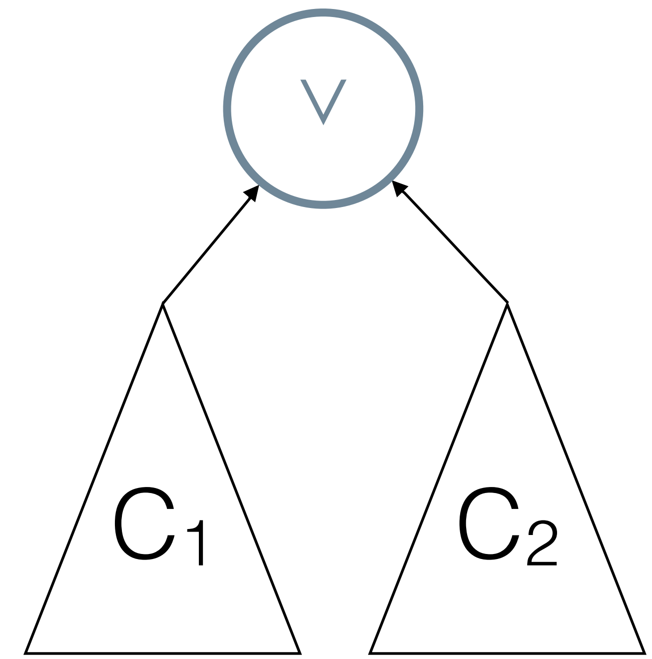


deterministic

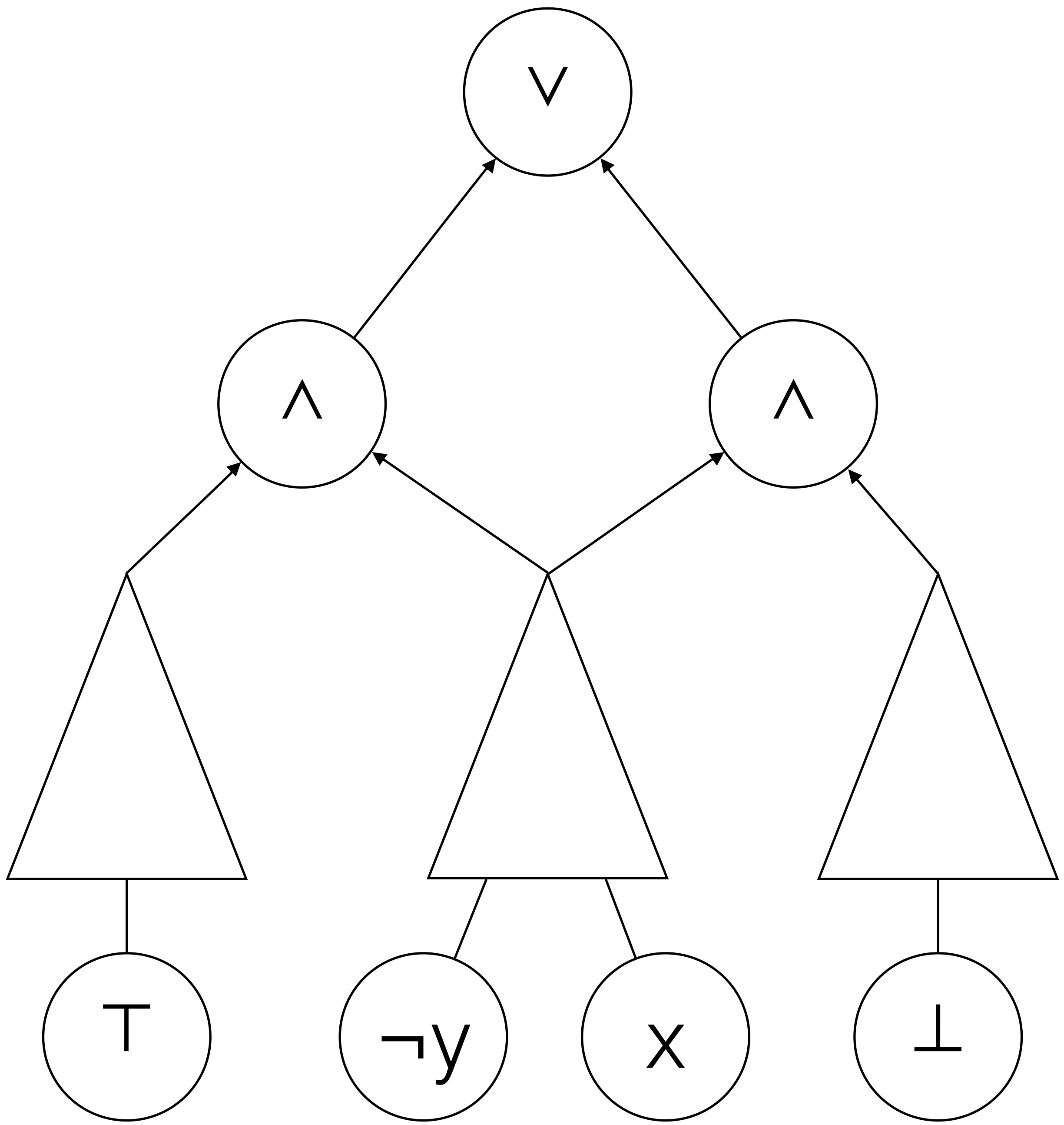
deterministic DNNF



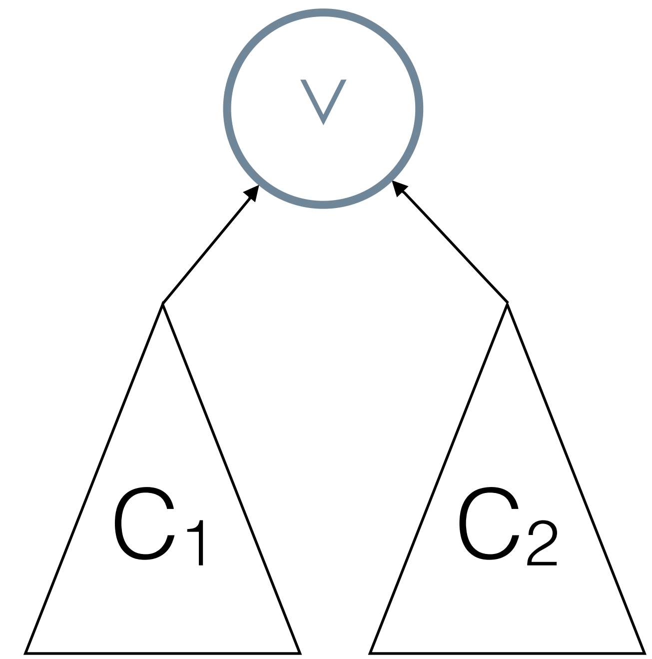
deterministic



deterministic DNNF



deterministic



$\text{models}(C_1) \cap \text{models}(C_2) = \varnothing$

CNFs are not **polysize compilable** into DNNFs

CNFs are not **polysize compilable** into DNNFs

unless PH collapses
(Selman & Kautz 1996)

CNFs are not **polysize compilable** into DNNFs

unless PH collapses
(Selman & Kautz 1996)

this can proved **unconditionally**
(Bova, Capelli, Mengel, S. 2014)

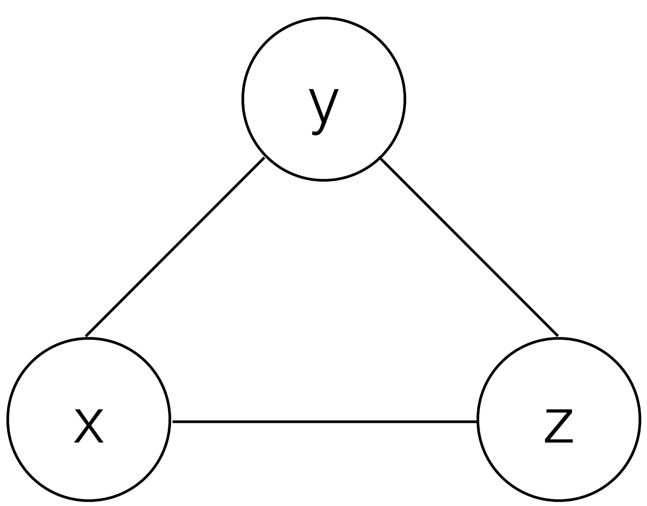
Structural Parameters

$(x \vee \neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (y \vee z)$

Structural Parameters

$(x \lor \neg y \lor z) \land (\neg x \lor \neg z) \land (y \lor z)$

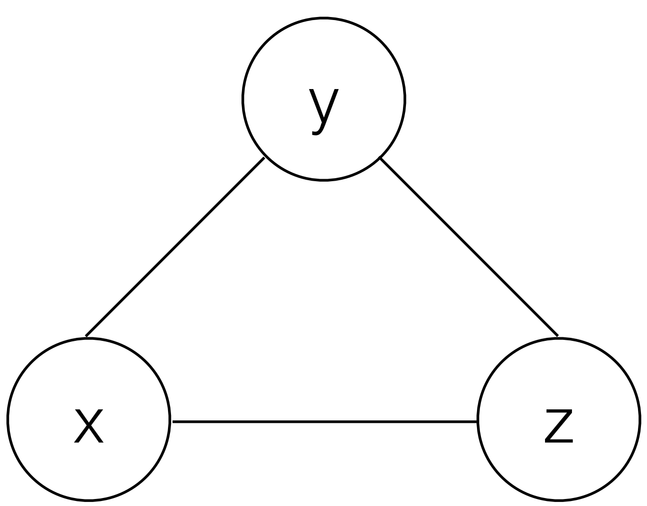
primal graph



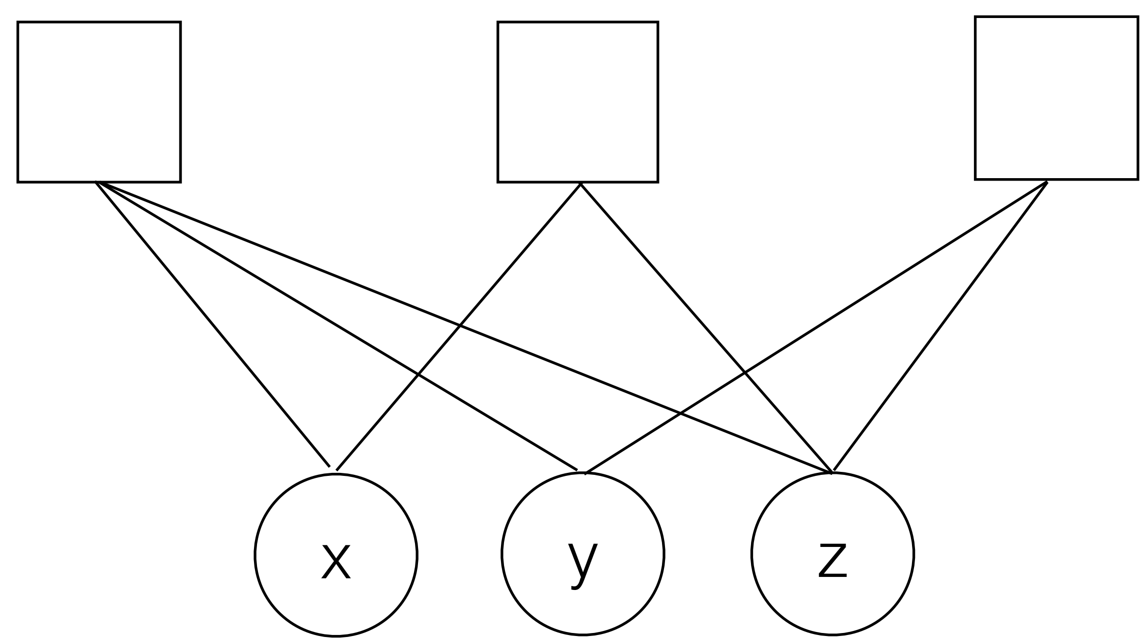
Structural Parameters

$(x \vee \neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (y \vee z)$

primal graph



incidence graph



incidence treewidth

primal treewidth

incidence treewidth

primal treewidth

decision

$2^k n$

incidence treewidth

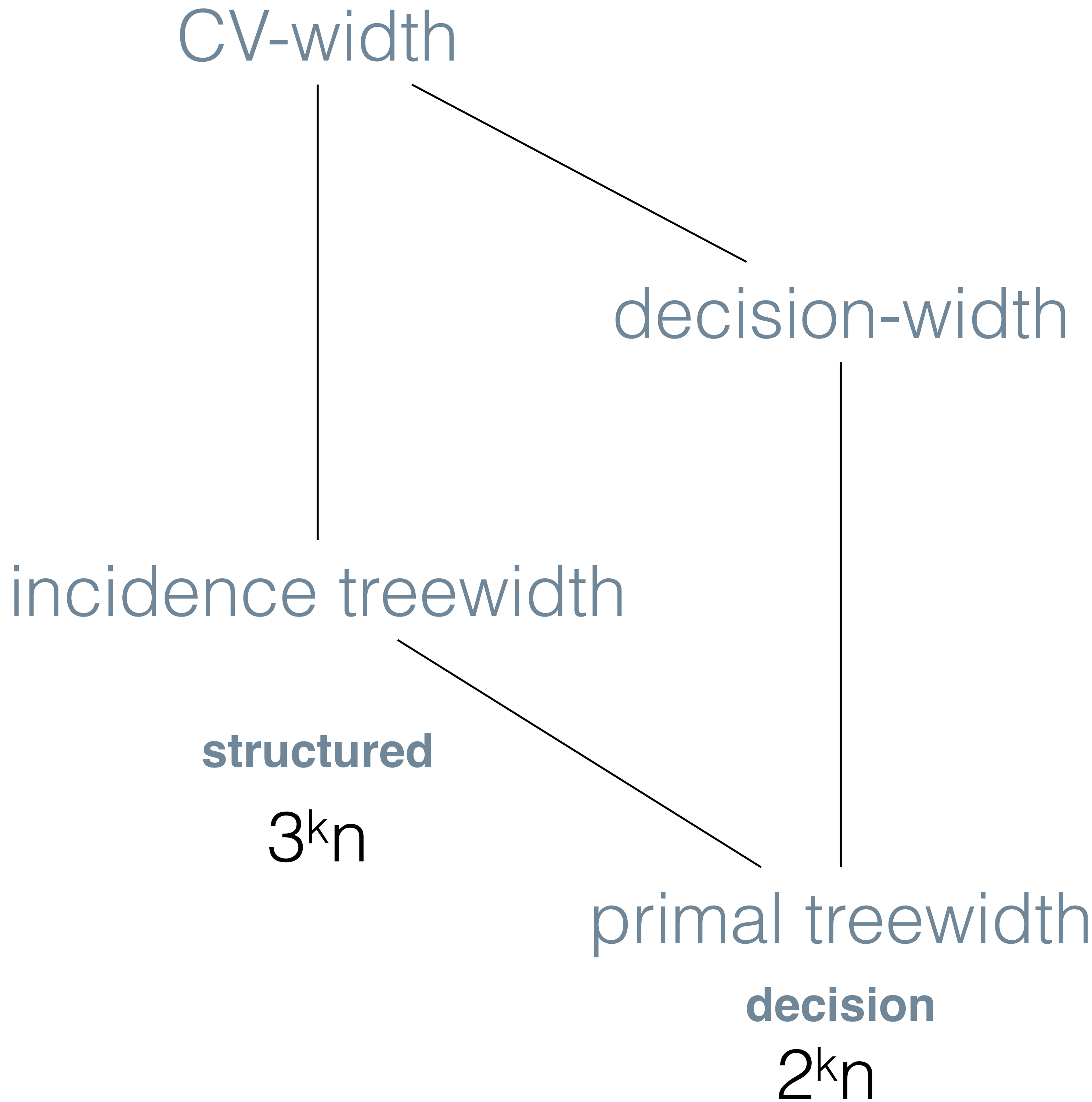
structured

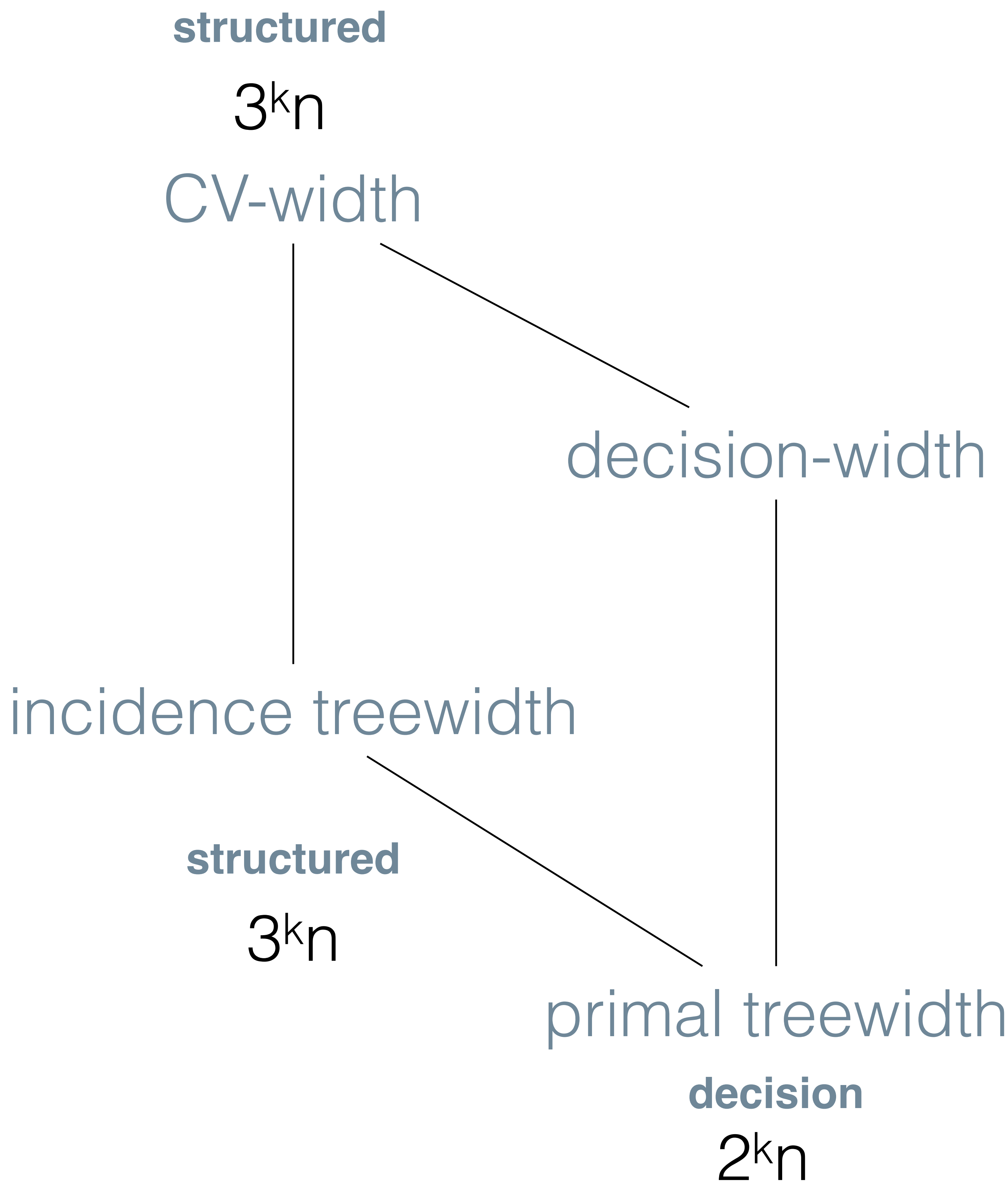
$3^k n$

primal treewidth

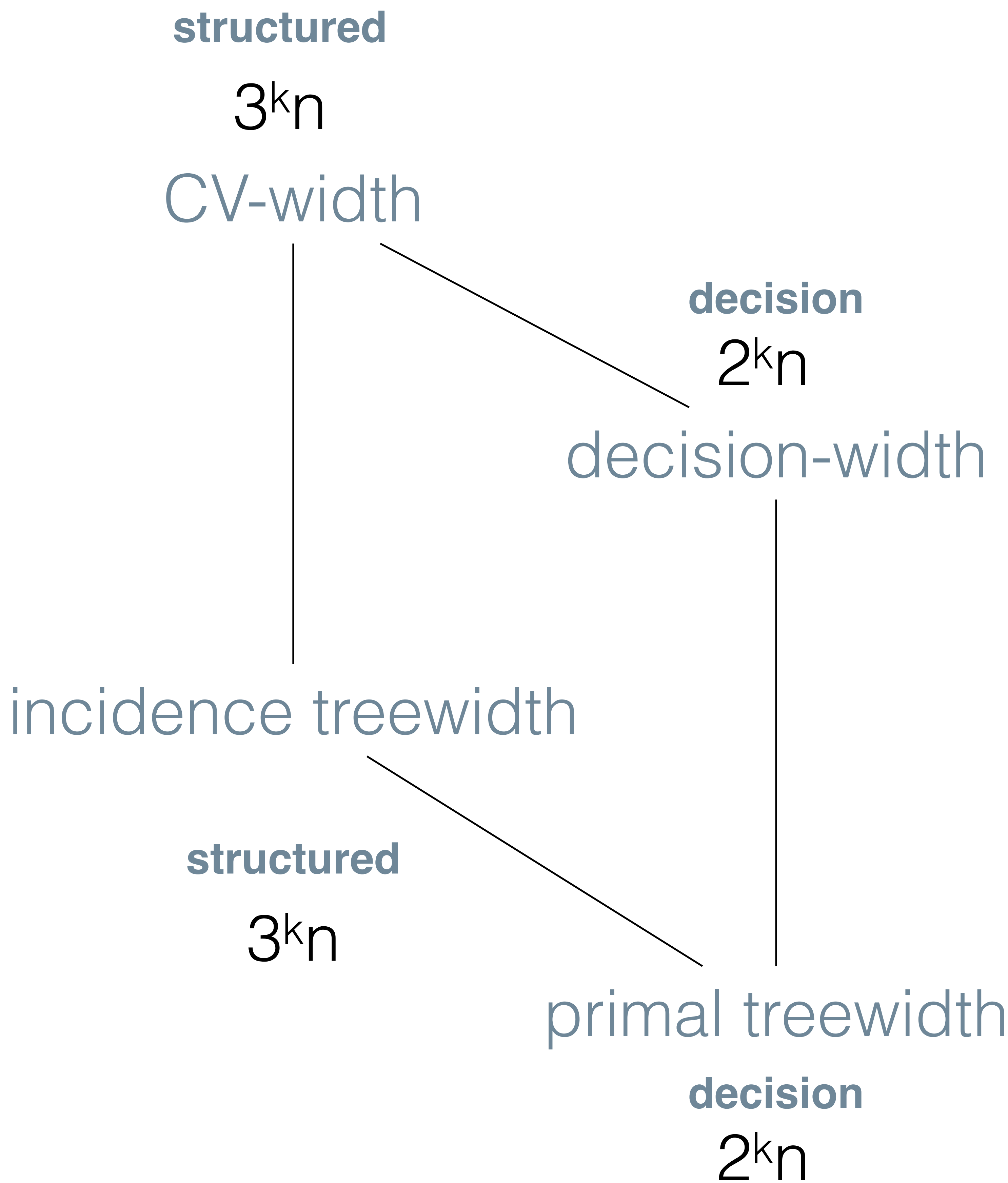
decision

$2^k n$

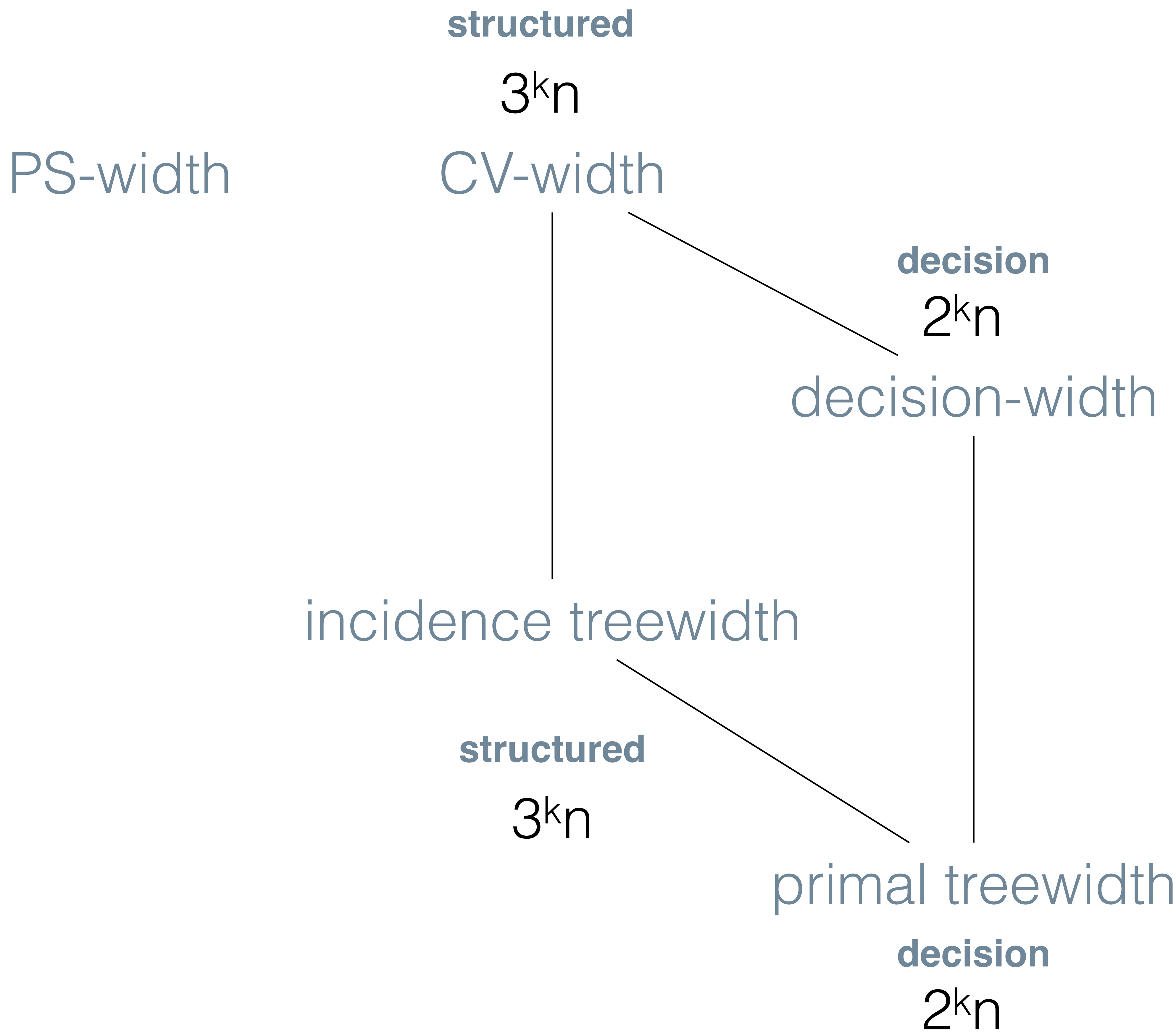

CV-width
decision-width
incidence treewidth
structured
3^k n
primal treewidth
decision
2^k n

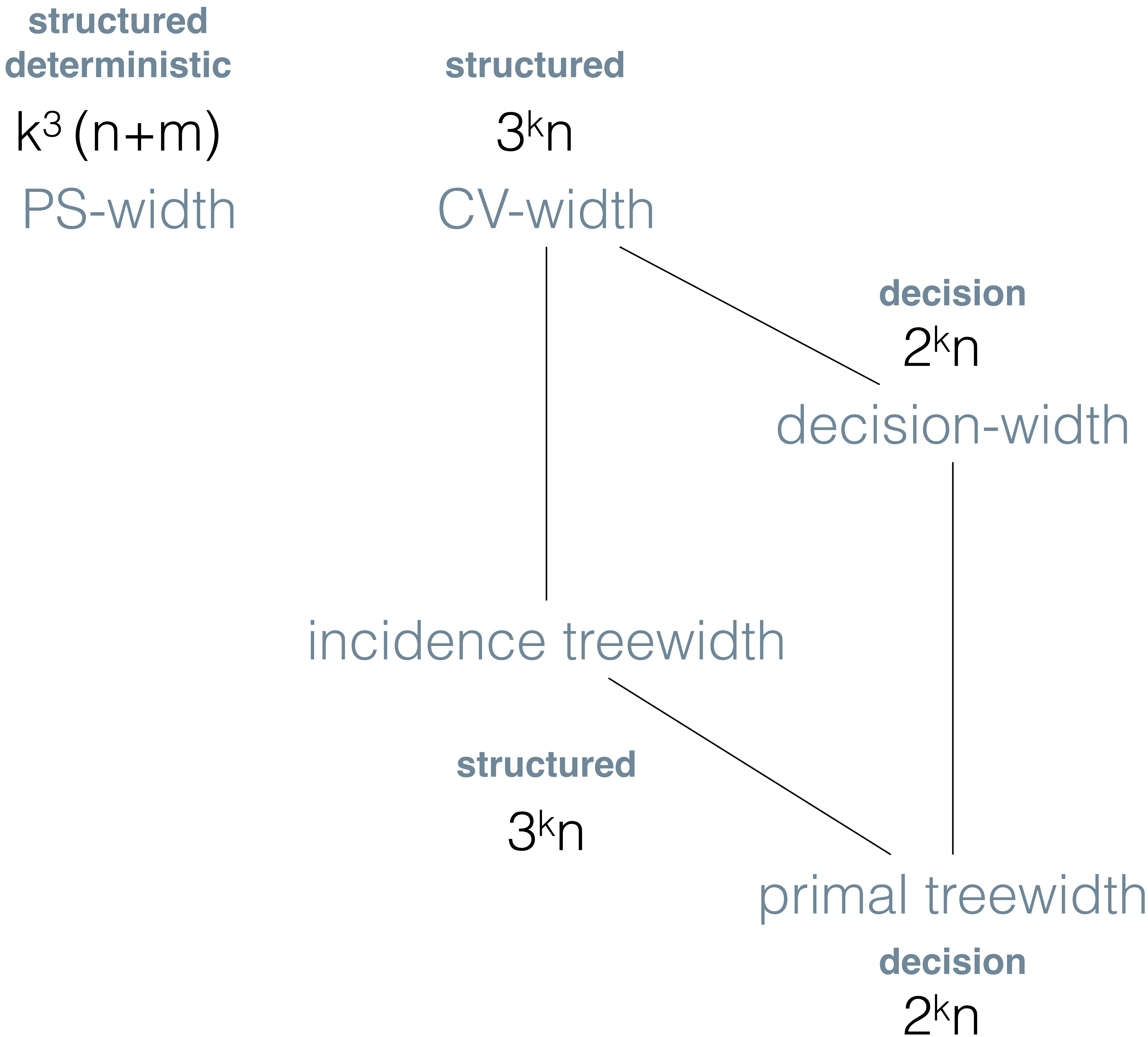


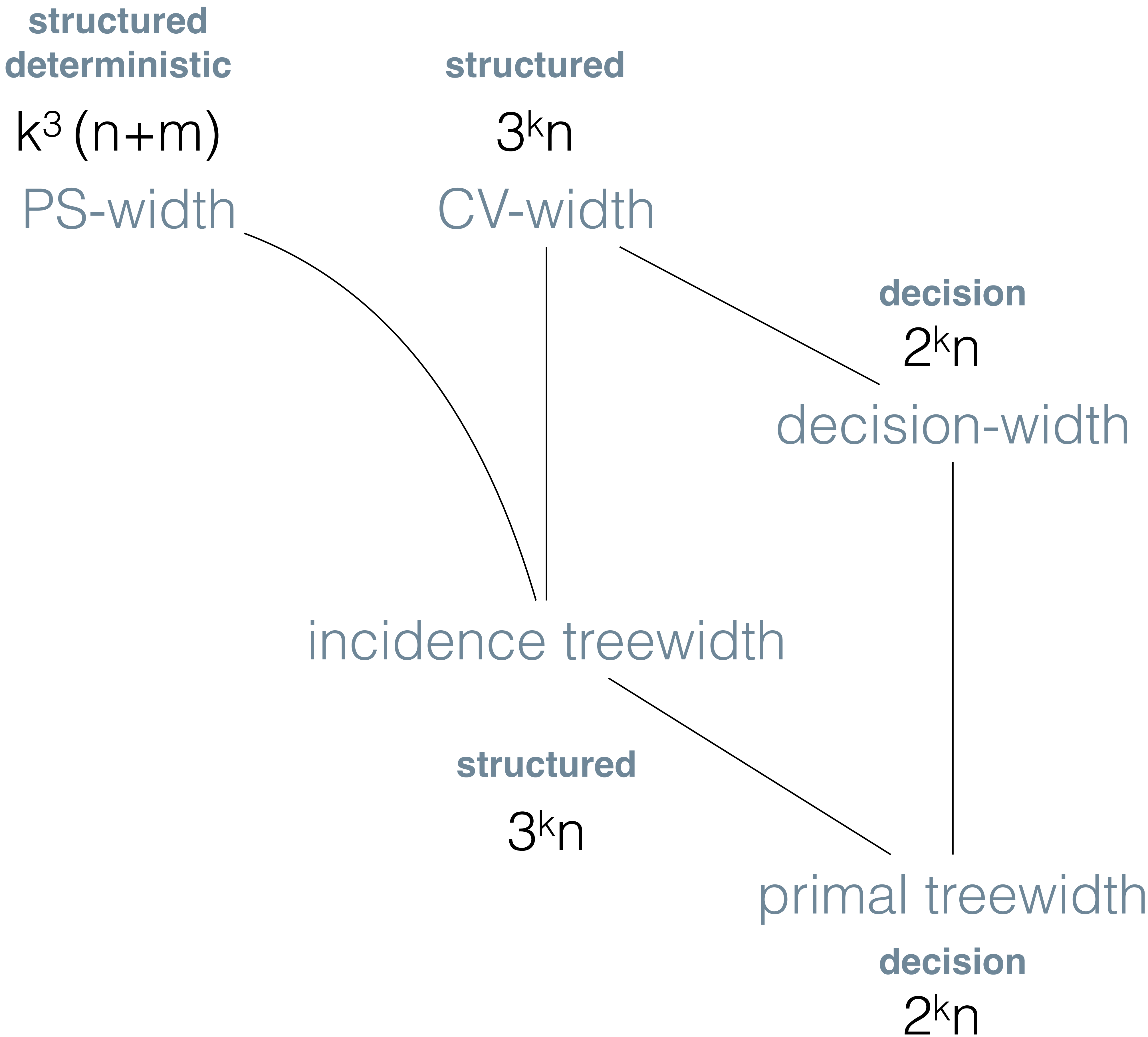
structured
3^k n
CV-width
decision-width
incidence treewidth
structured
3^k n
primal treewidth
decision
2^k n

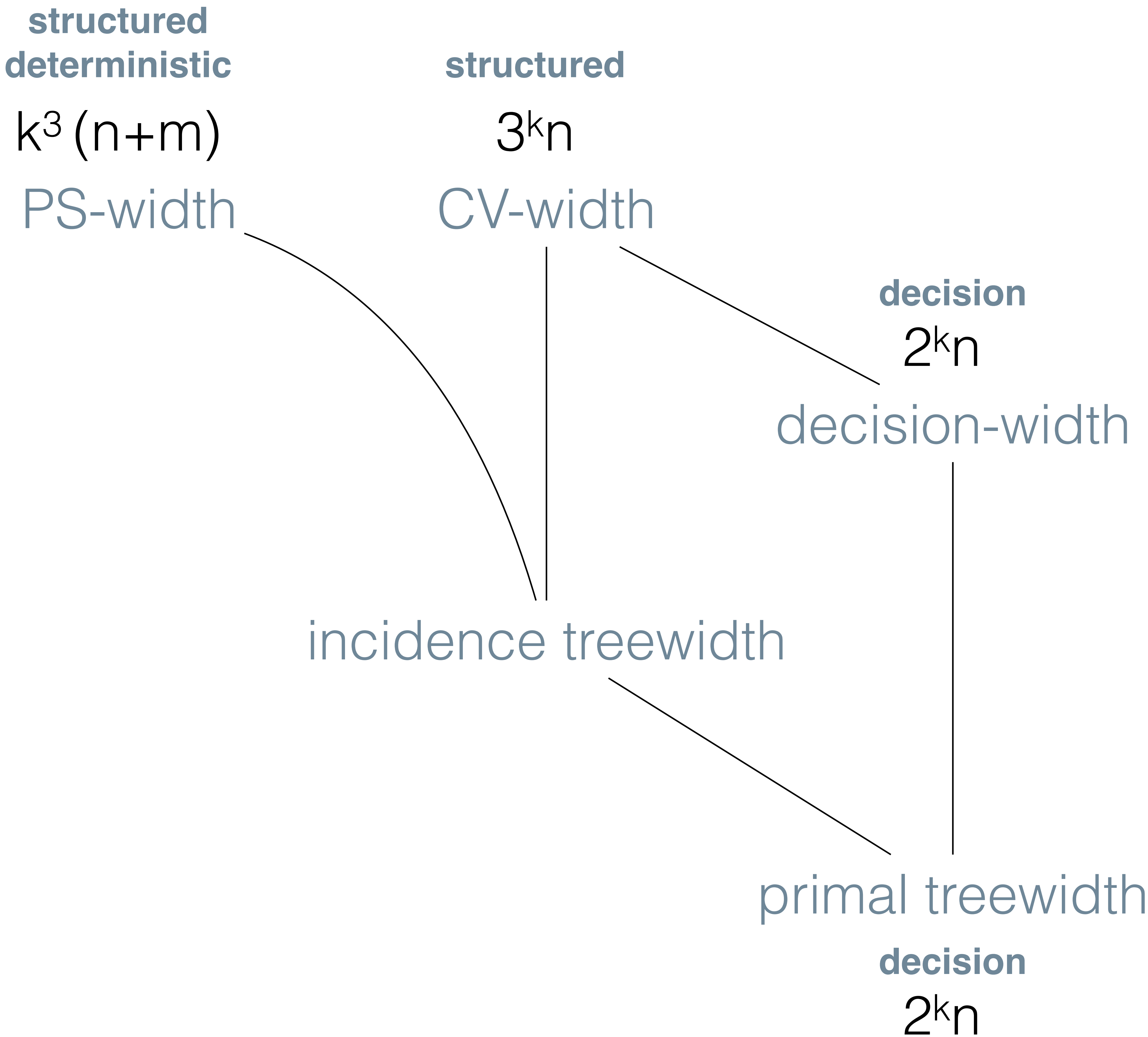


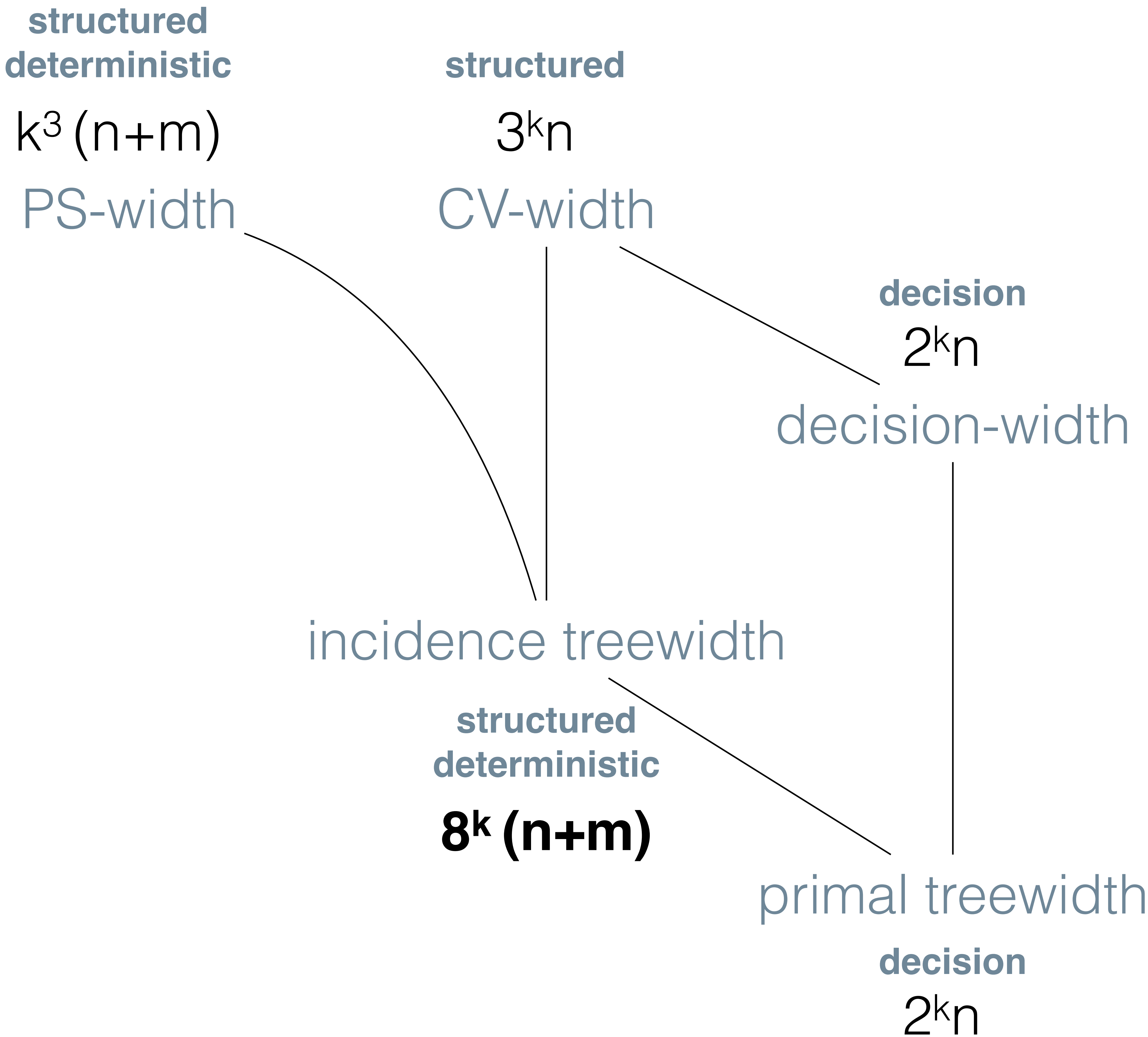
structured
3kn
CV-width
decision
2kn
decision-width
incidence treewidth
structured
3kn
primal treewidth
decision
2kn


structured
3ᵏn
PS-width
CV-width
decision
2ᵏn
decision-width
incidence treewidth
structured
3ᵏn
primal treewidth
decision
2ᵏn

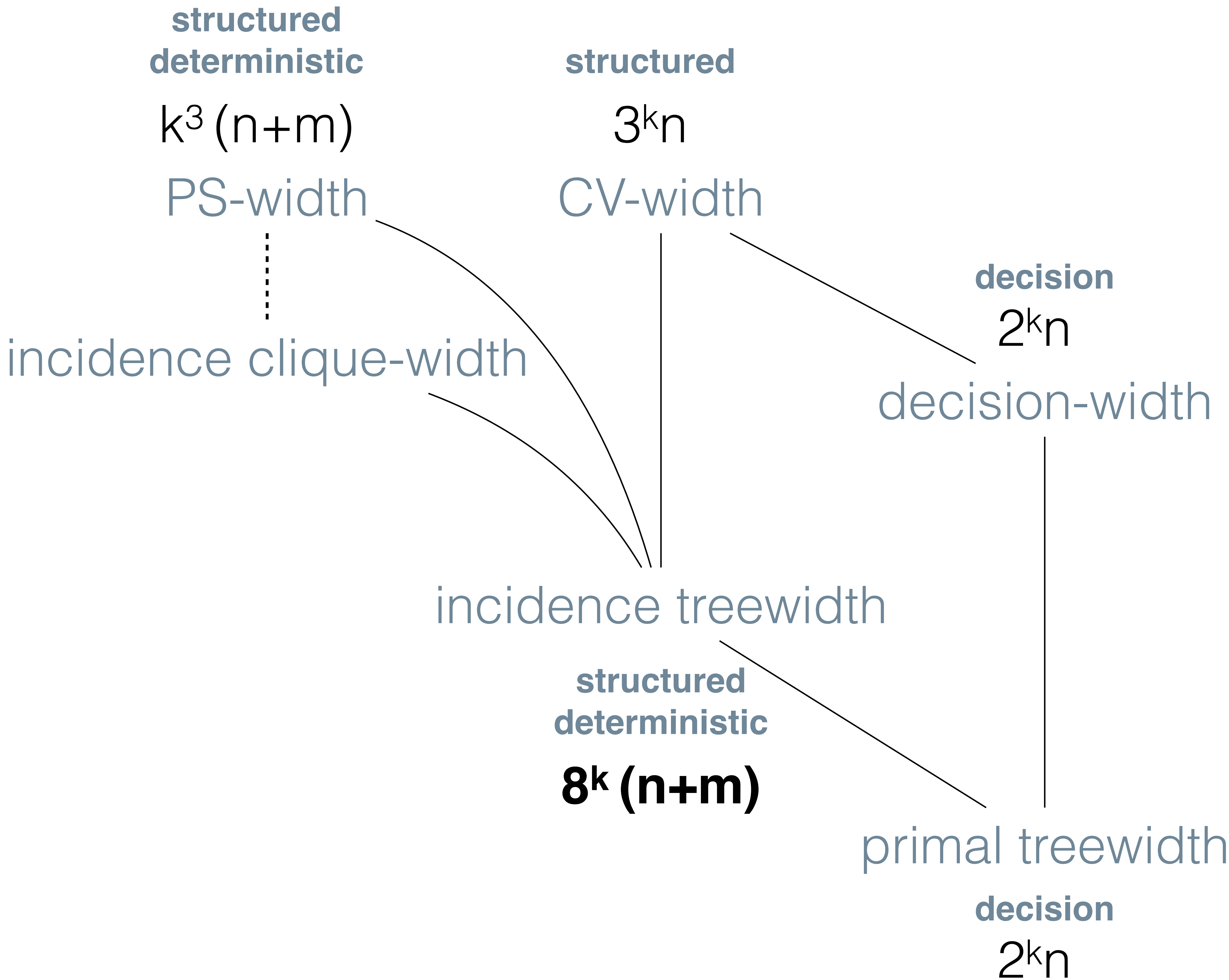

structured deterministic
k3 (n+m)
PS-width
structured
3kn
CV-width
decision
2kn
decision-width
incidence treewidth
structured
3kn
primal treewidth
decision
2kn


structured
deterministic
k3 (n+m)
PS-width
structured
3kn
CV-width
decision
2kn
decision-width
incidence treewidth
structured
3kn
primal treewidth
decision
2kn

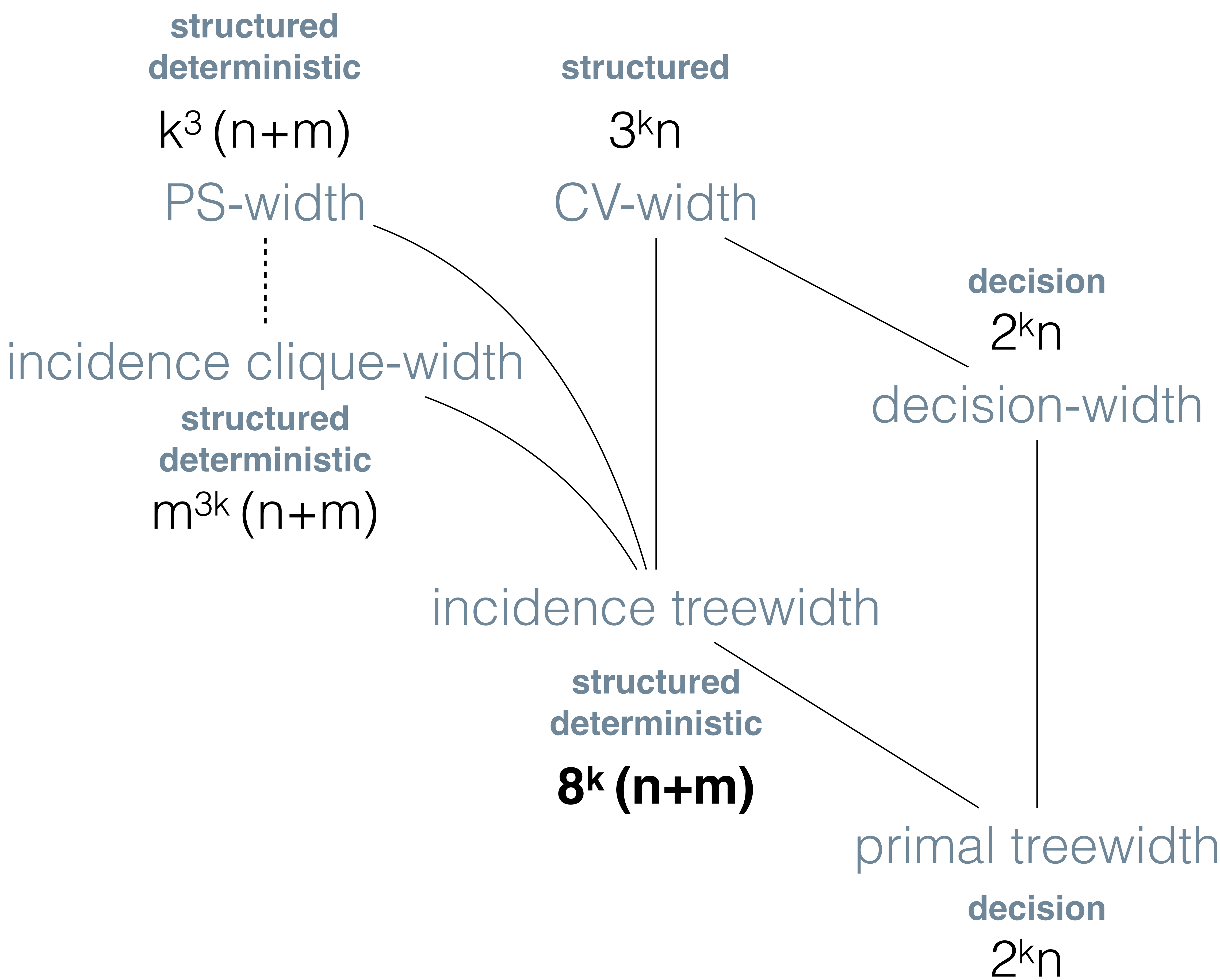

structured deterministic
$k^3 (n+m)$
PS-width
structured
$3^k n$
CV-width
decision
$2^k n$
decision-width
incidence treewidth
primal treewidth
decision
$2^k n$



structured
deterministic
k3 (n+m)
PS-width
structured
3kn
CV-width
decision
2kn
decision-width
incidence treewidth
structured
deterministic
8k (n+m)
primal treewidth
decision
2kn



structured
deterministic
k3 (n+m)
structured
3kn
PS-width
CV-width
decision
2kn
incidence clique-width
decision-width
incidence treewidth
structured
deterministic
8k (n+m)
primal treewidth
decision
2kn


structured
deterministic
k3 (n+m)
PS-width
structured
3kn
CV-width
decision
2kn
decision-width
incidence clique-width
structured
deterministic
m3k (n+m)
incidence treewidth
structured
deterministic
8k (n+m)
primal treewidth
decision
2kn

The Compilation Algorithm

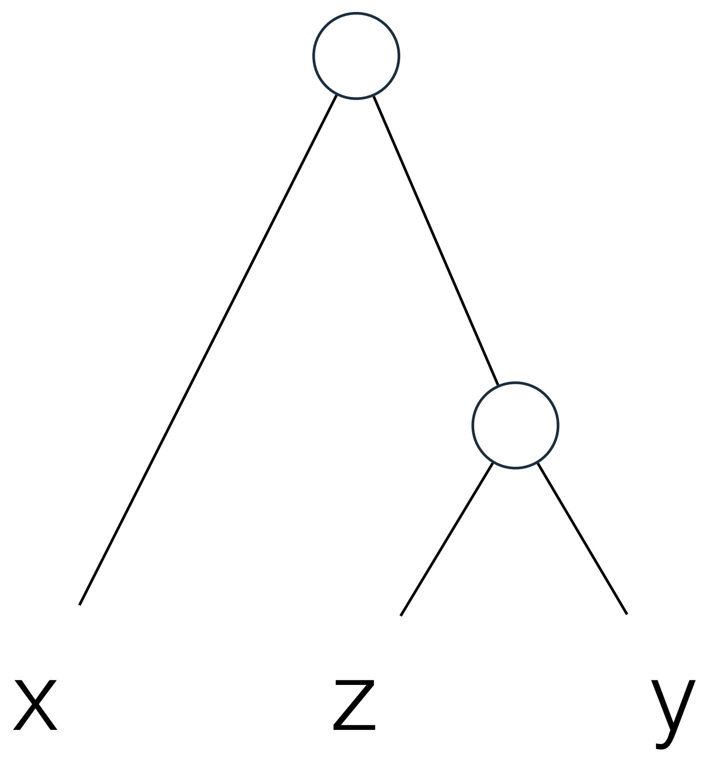
$$(x \lor \neg y \lor z) \land (\neg x \lor \neg z) \land (y \lor z)$$

$$(x \lor \neg y \lor z) \land (\neg x \lor \neg z) \land (y \lor z)$$

vtree

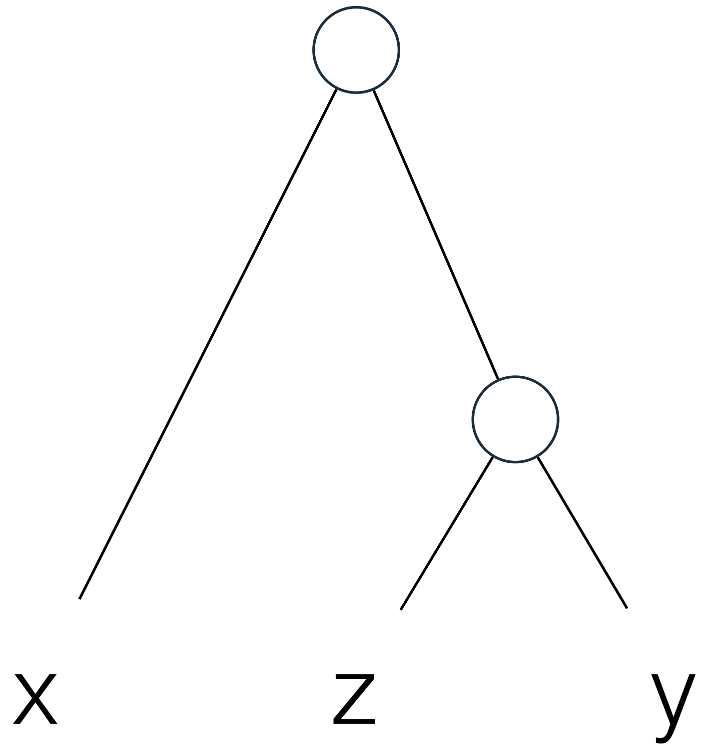
$$(x \vee \neg y \vee z) \wedge (\neg x \vee \neg z) \wedge (y \vee z)$$

vtree



$$(x \lor \neg y \lor z) \land (\neg x \lor \neg z) \land (y \lor z)$$

vtree

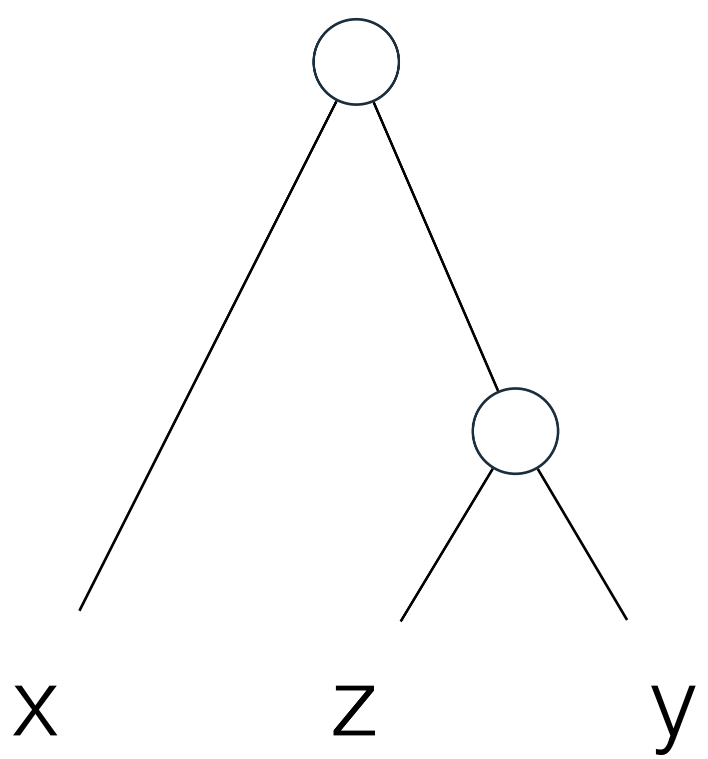


branch decomposition

C_1 C_2 C_3

$$(x \lor \neg y \lor z) \land (\neg x \lor \neg z) \land (y \lor z)$$

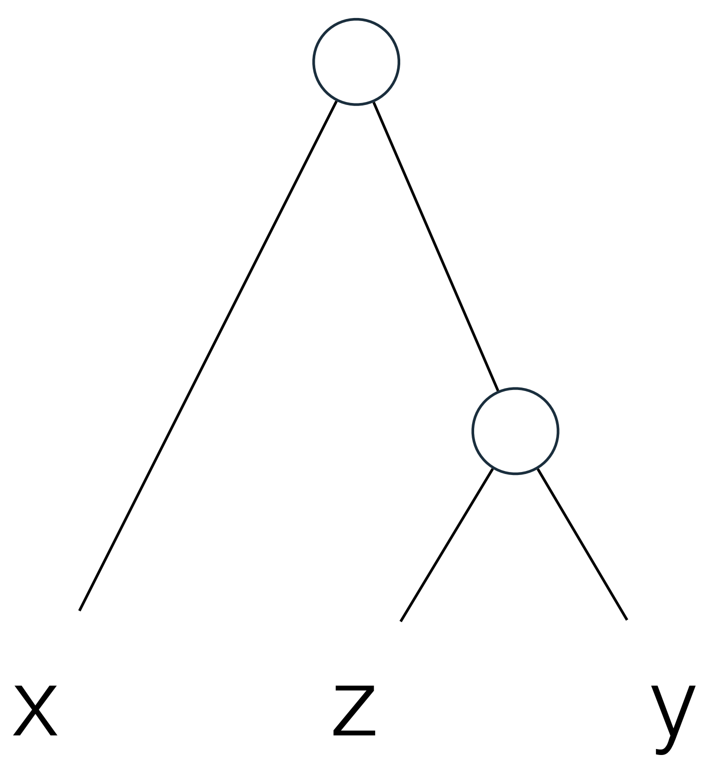
vtree



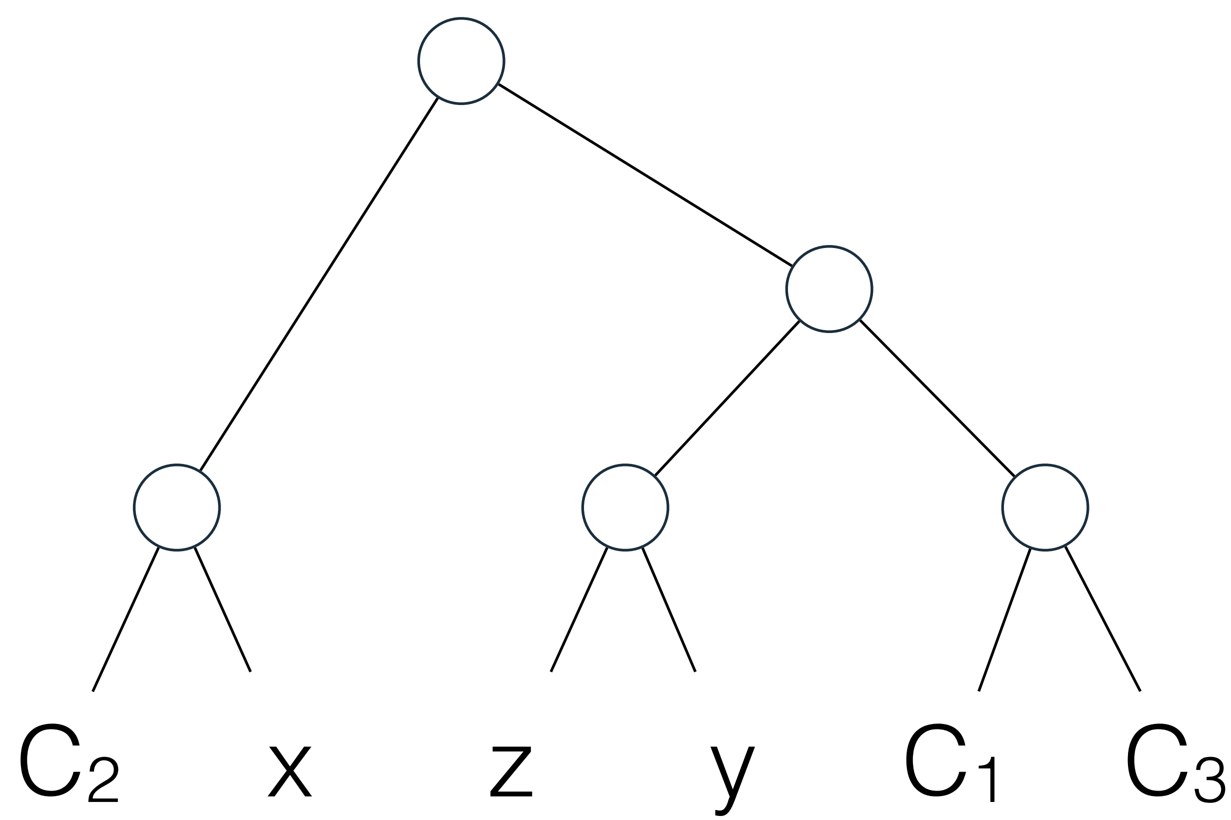
branch decomposition

$$\overset{C_1}{(x \lor \neg y \lor z)} \land \overset{C_2}{(\neg x \lor \neg z)} \land \overset{C_3}{(y \lor z)}$$

vtree



branch decomposition



Projections

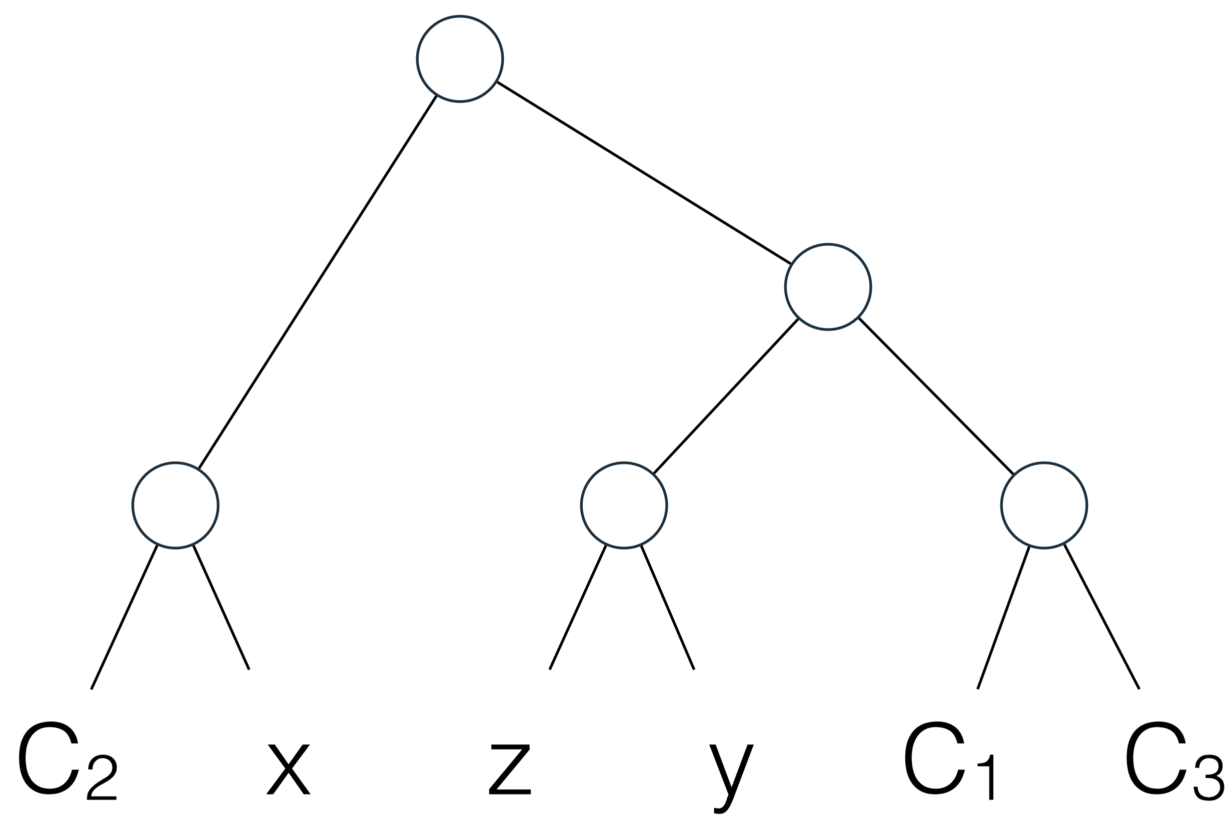
The **projection** of F under assignment $\boldsymbol{\tau}$ is the set **F(τ)** of clauses of F satisfied by $\boldsymbol{\tau}$.

Projections

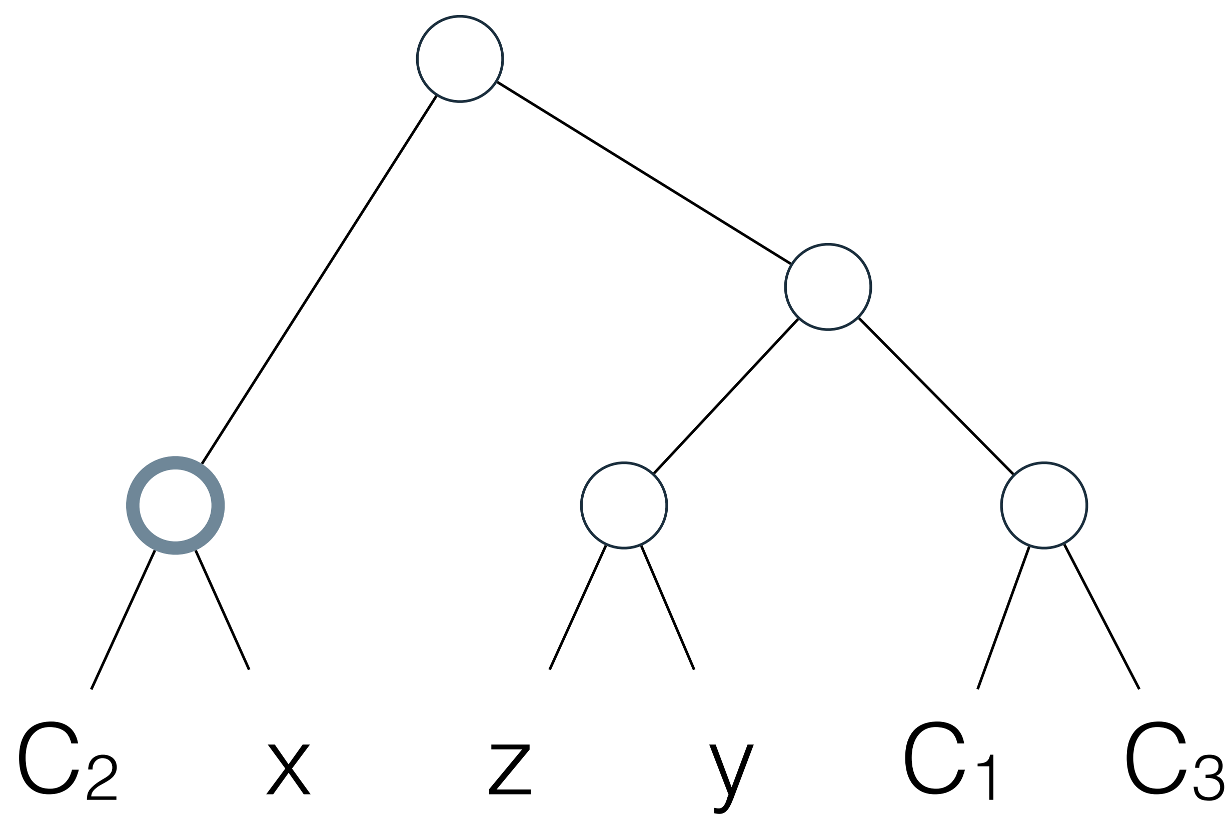
The **projection** of F under assignment τ is the set **F(τ)** of clauses of F satisfied by τ.

proj(F, X) the set of projections of F under assignments to X.

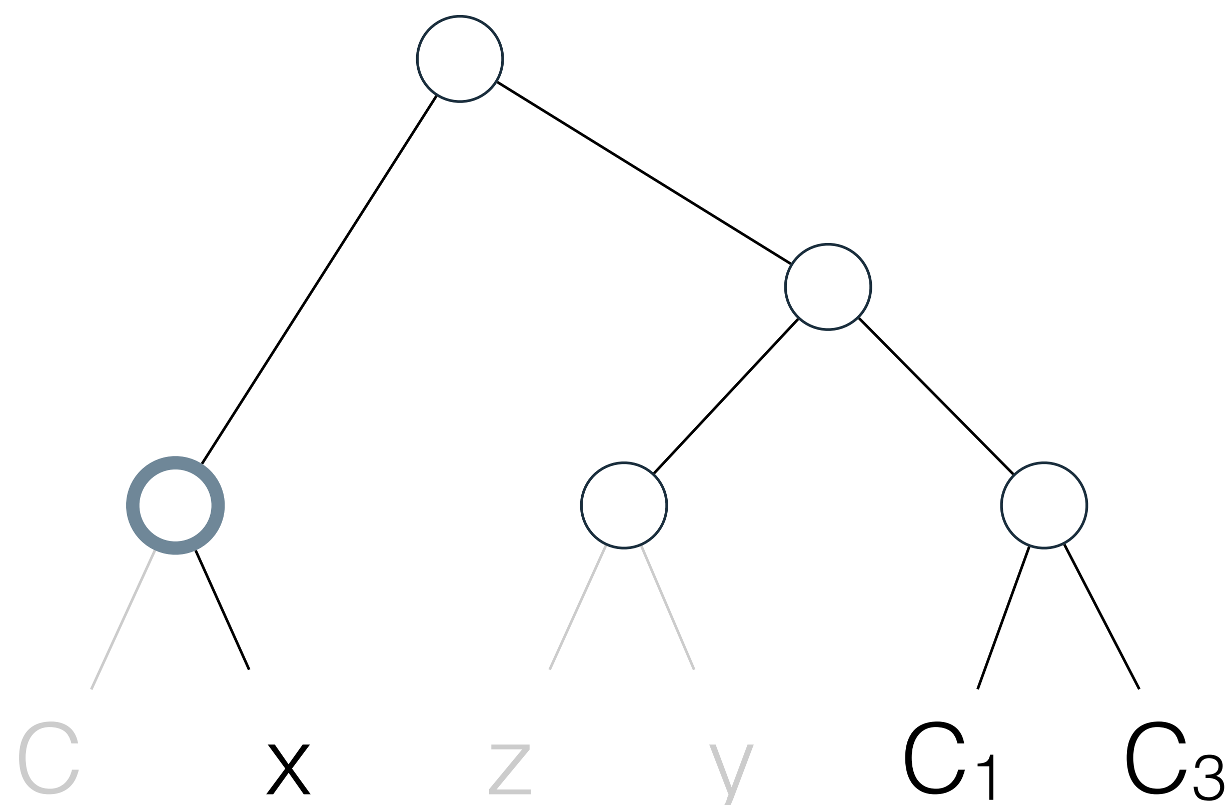
PS-width



PS-width

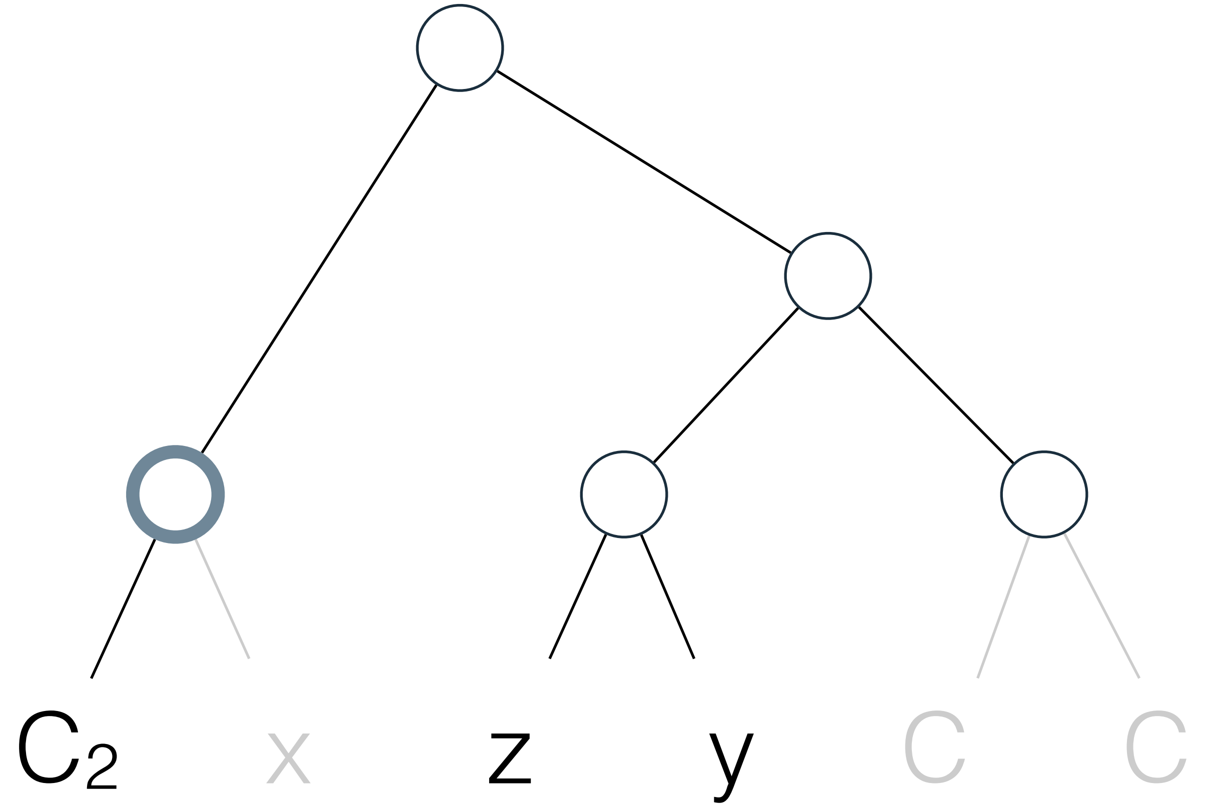


PS-width



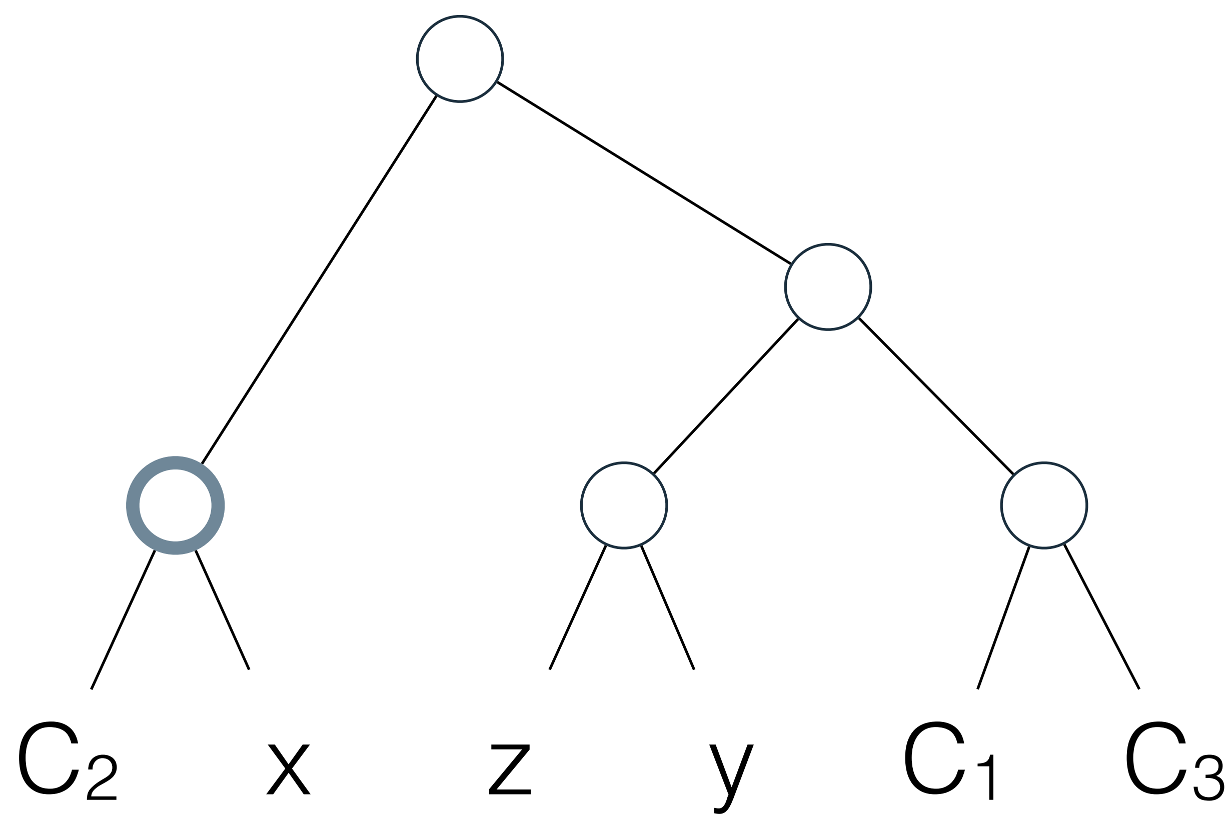
proj({C_1, C_3}, {x})

PS-width



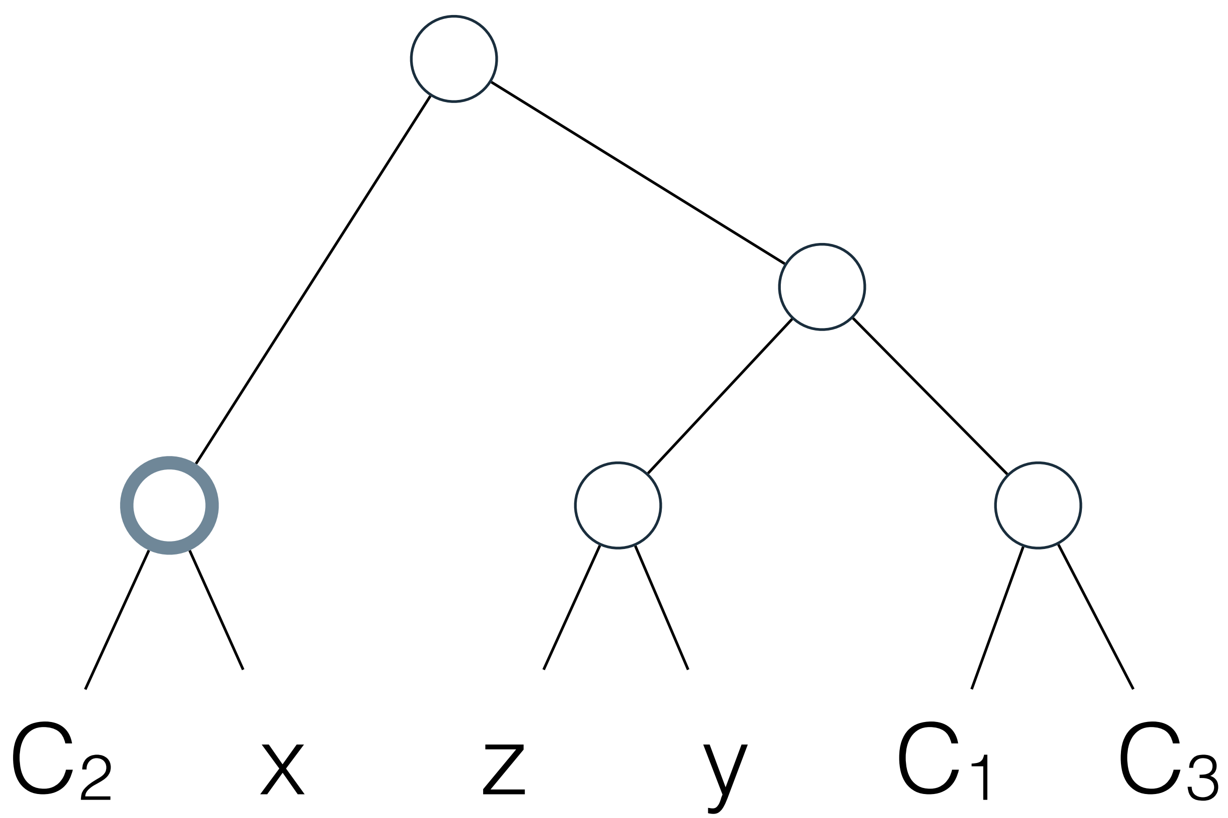
proj({C1, C3}, {x}) **proj({C2}, {z,y})**

PS-width



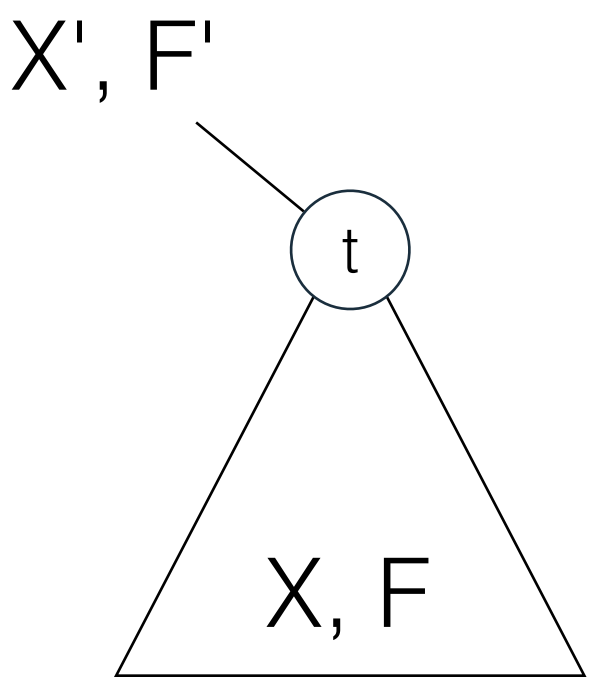
|proj({C1, C3}, {x})| |proj({C2}, {z,y})|

PS-width

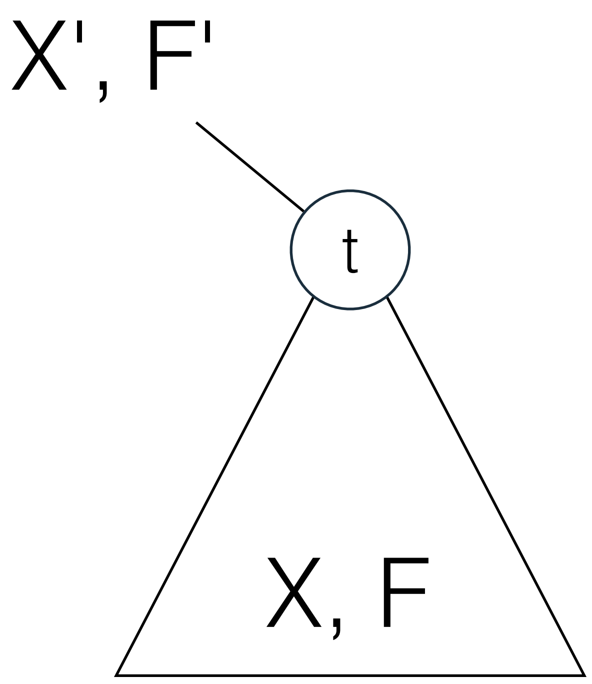


max($|proj(\{C_1, C_3\}, \{x\})|$ **,** $|proj(\{C_2\}, \{z,y\})|$ **)**

Shapes

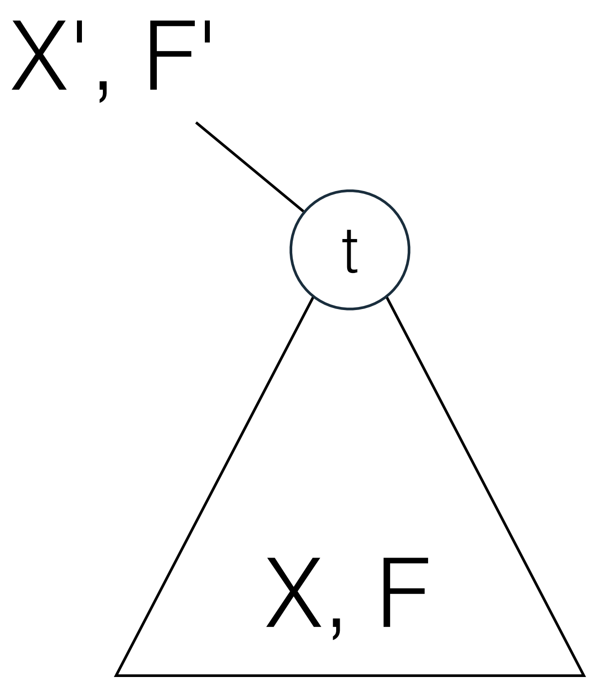


Shapes



A **shape** for t is a pair **(S, S’)** with S ∊ proj(F’, X) and S’ ∊ proj(F, X’).

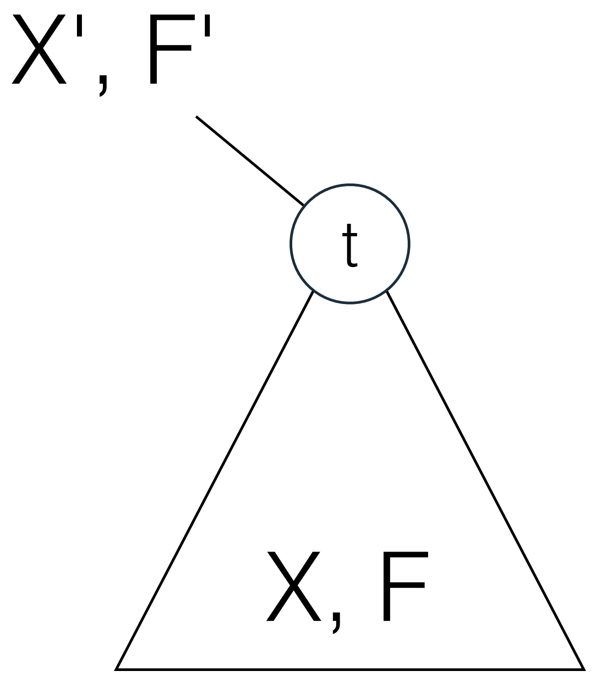
Shapes



A **shape** for t is a pair **(S, S')** with S ∈ proj(F', X) and S' ∈ proj(F, X').

An assignment $\boldsymbol{\tau}$: X → {0,1} **has shape** (S, S') if

Shapes

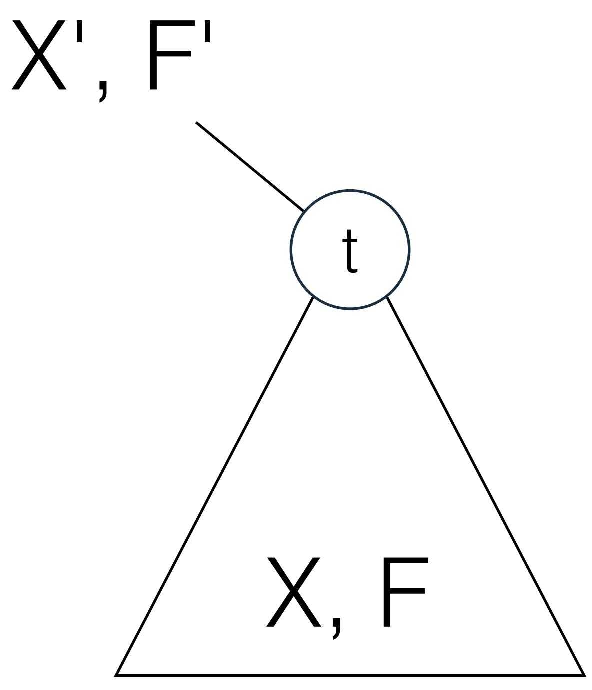


A **shape** for t is a pair **(S, S')** with S ∈ proj(F', X) and S' ∈ proj(F, X').

An assignment $\boldsymbol{\tau}$: X → {0,1} **has shape** (S, S') if

1. F'($\boldsymbol{\tau}$) = S

Shapes

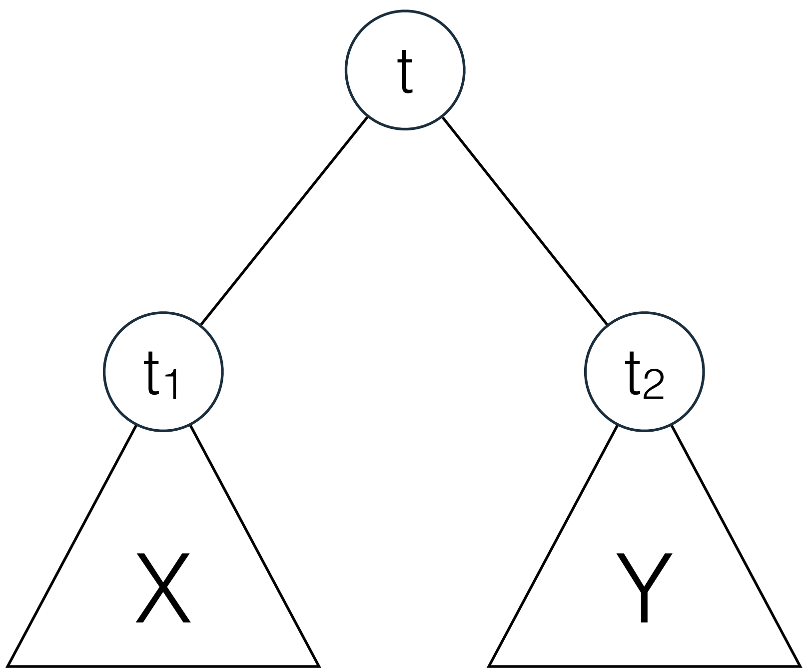


A **shape** for t is a pair **(S, S')** with $S \in \text{proj}(F', X)$ and $S' \in \text{proj}(F, X')$.

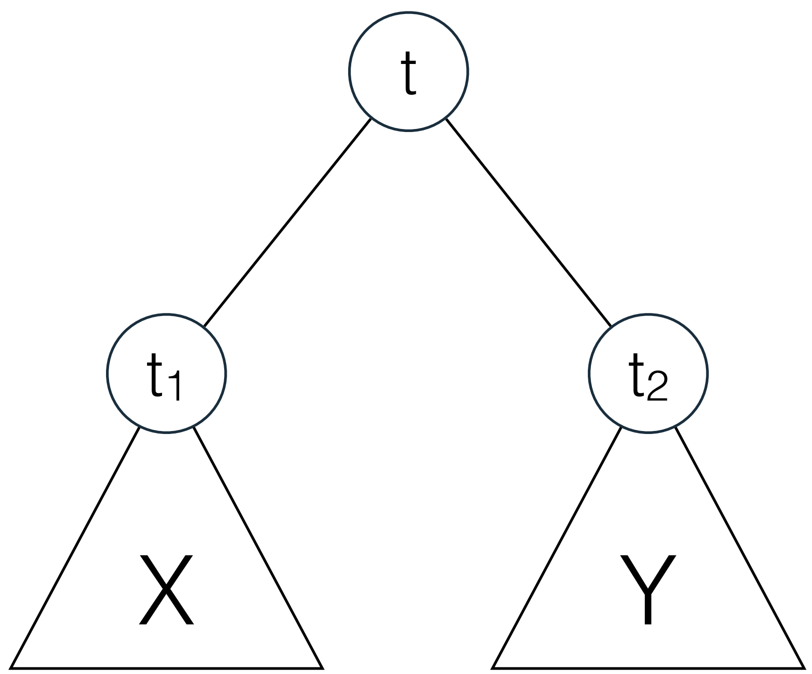
An assignment $\boldsymbol{\tau}: X \to \{0,1\}$ **has shape** (S, S') if

1. $F'(\boldsymbol{\tau}) = S$
2. $F(\boldsymbol{\tau}) \cup S' = F$

Decomposing Shapes

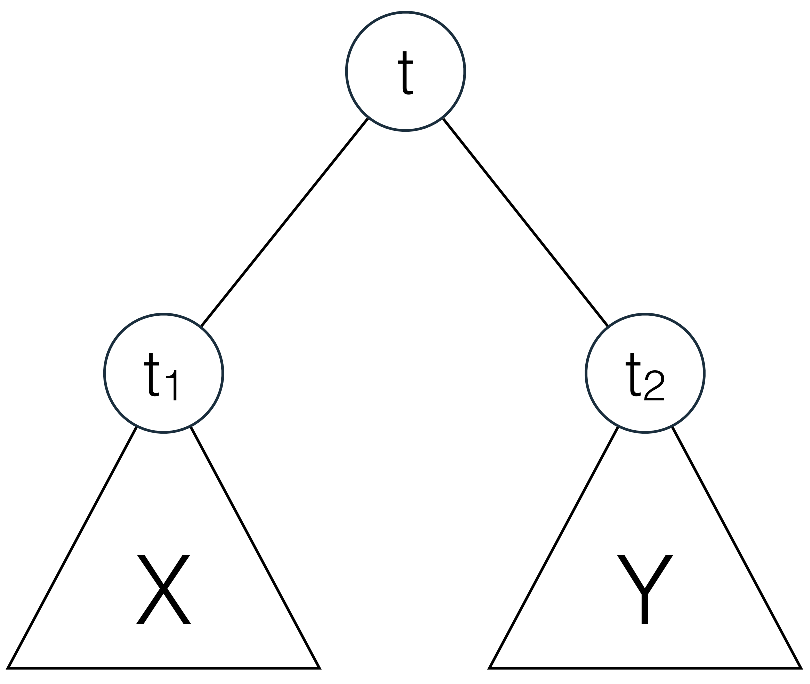


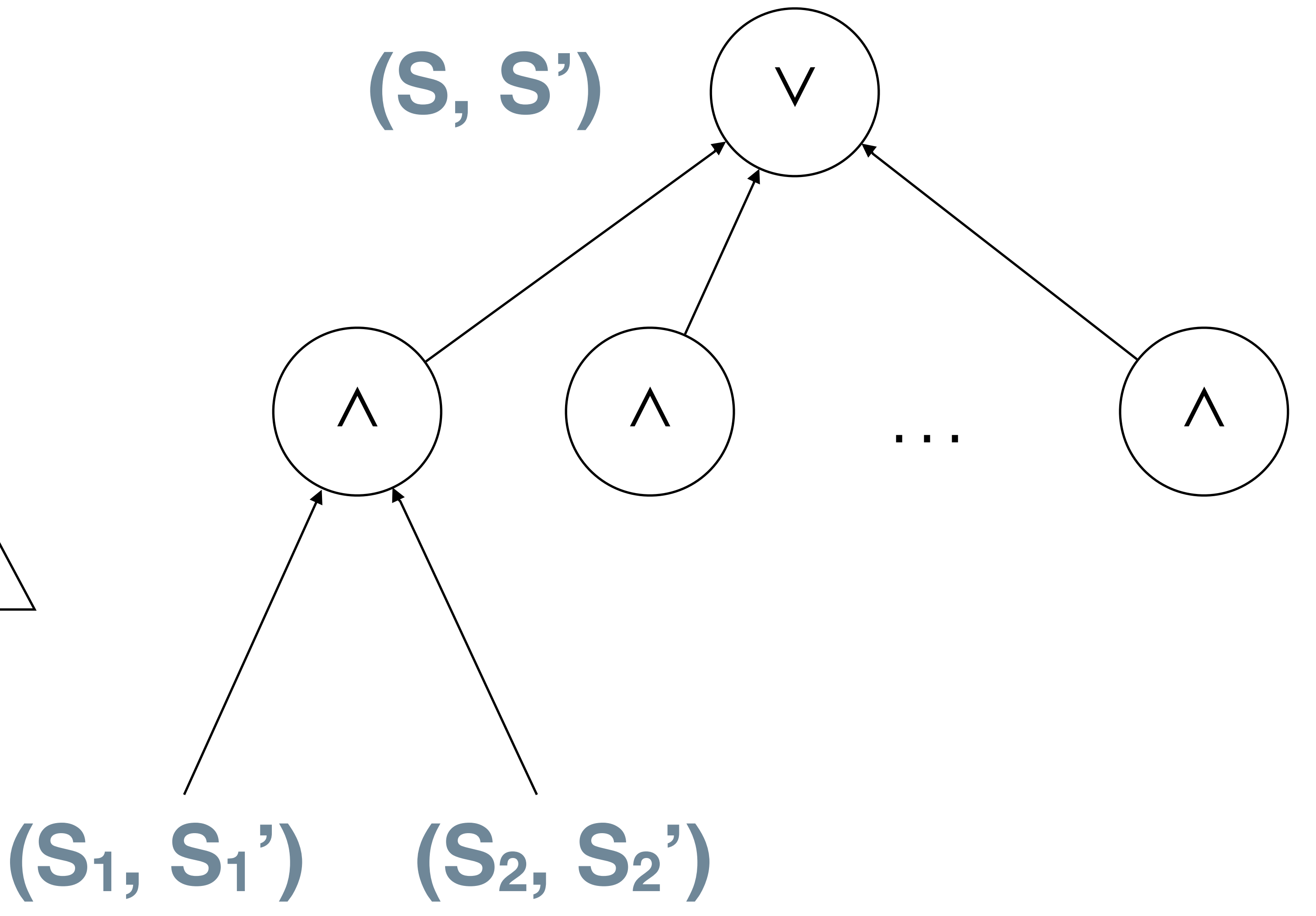
Decomposing Shapes



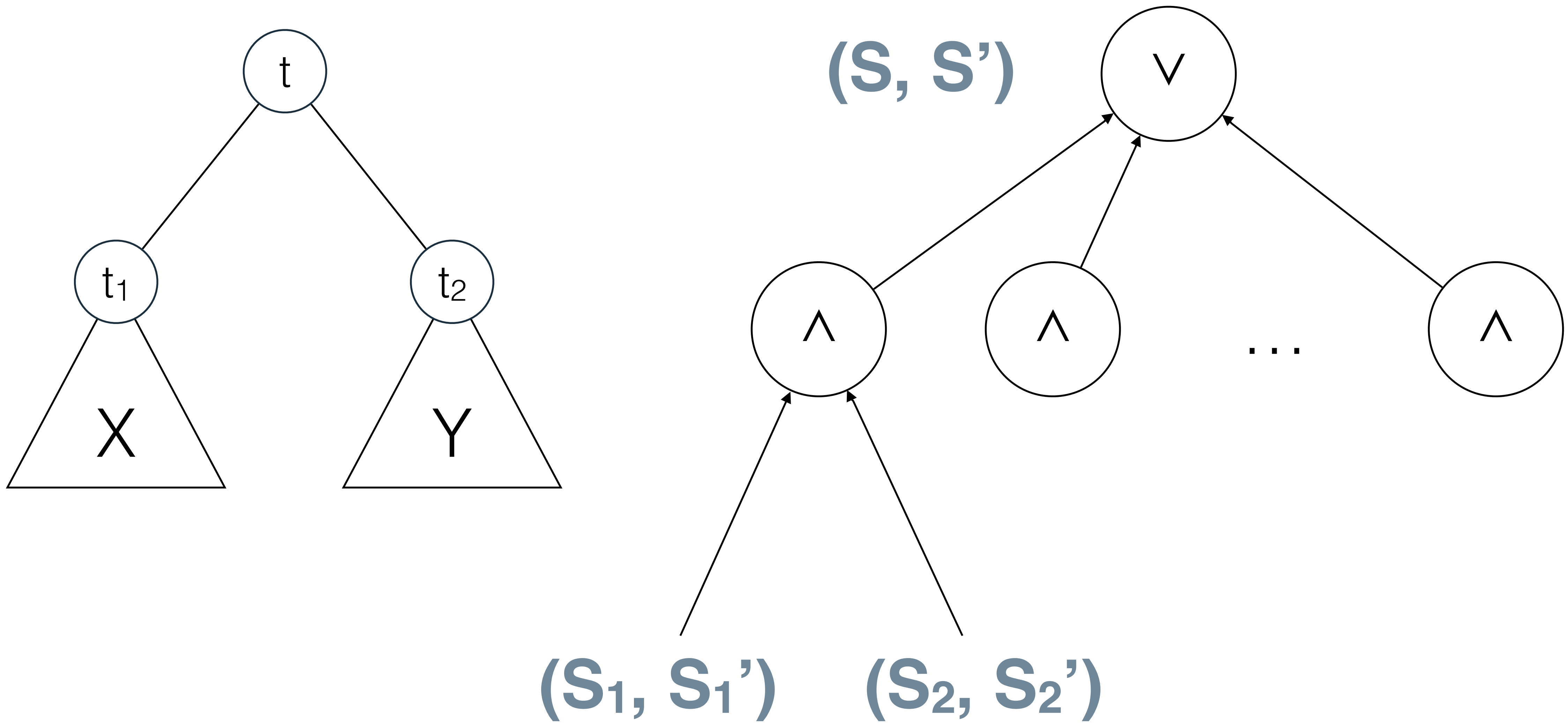
(S, S')

Decomposing Shapes



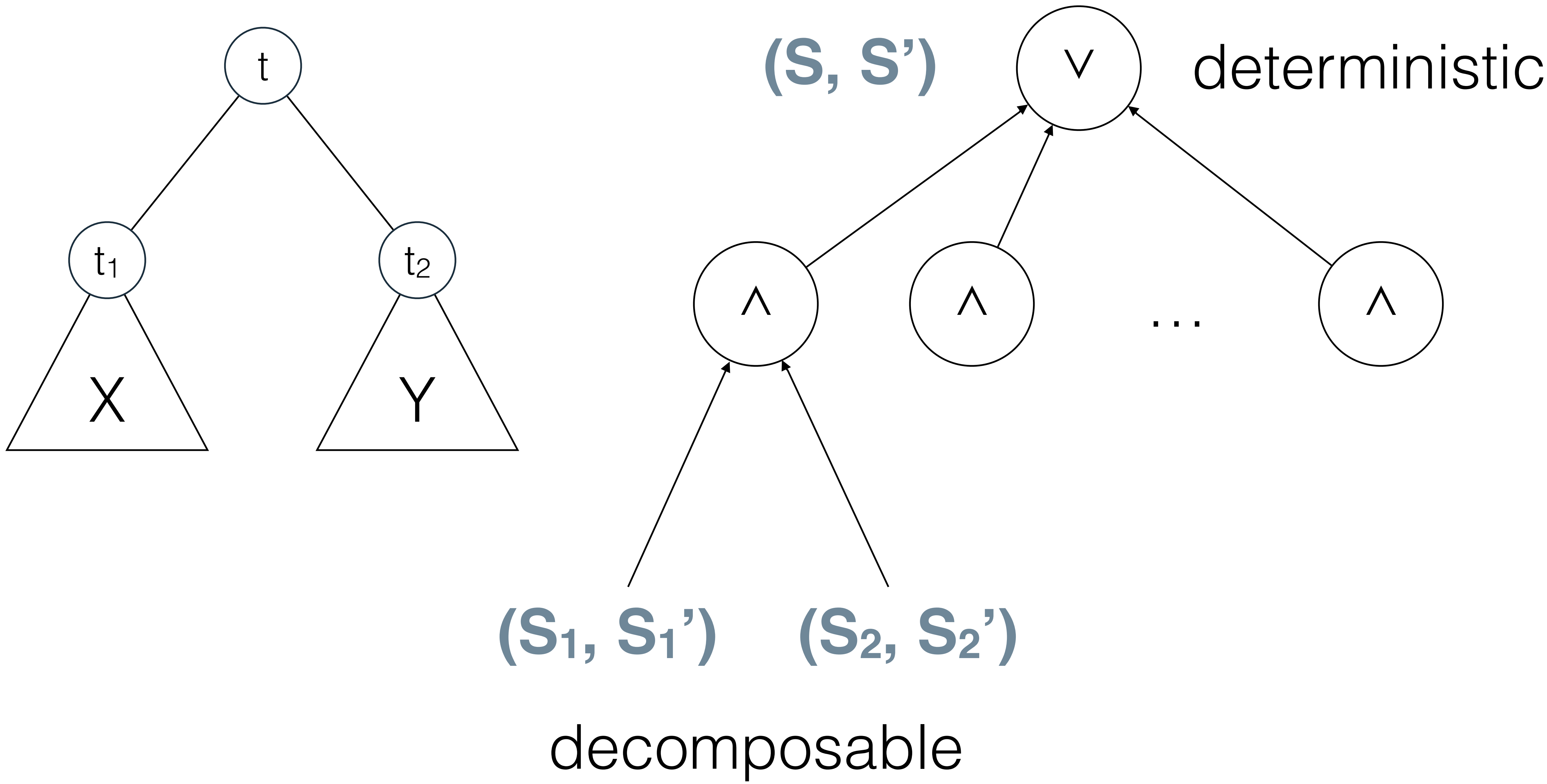


Decomposing Shapes



decomposable

Decomposing Shapes



Open Questions

Open Questions

Can we compile into more restrictive languages?

decision DNNFs, SDDs

Open Questions

Can we compile into more restrictive languages?

decision DNNFs, SDDs

What is the relation between PS-width and CV-width?

Open Questions

Can we compile into more restrictive languages?

decision DNNFs, SDDs

What is the relation between PS-width and CV-width?

Can decompositions of small PS-width be computed efficiently?