Holistic Specifications for Robust Code

Sophia Drossopoulou
Imperial College London

based on prior work with James Noble (VU Wellington),
Toby Murray (Uni Melbourne) , Mark Miller (Agorics),
and Susan Eisenbach, Shupeng Loh and Emil Klasan (Imperial)

Today

- Traditional Specifications do not adequately address Robustness
- Holistic Specifications — Summary and by Example
- Holistic Specification Semantics

Today

- **Traditional Specifications do not adequately address Robustness**
- Holistic Specifications — Summary and Examples
- Holistic Specification Semantics

Traditional Specification Languages do not adequately address robustness considerations

Traditional Specification Languages do not adequately address robustness considerations

Traditional Specs

Traditional Specification Languages do not adequately address robustness considerations

Traditional Specs

- designed for *closed* world
- pre- and post condition per function; *sufficient* conditions for some action/effect
- *explicit* about each individual function, and *implicit* about emergent behaviour

Traditional Specification Languages do not adequately address robustness considerations

Traditional Specs

Robustness considerations

- designed for *closed* world
- pre- and post condition per function; *sufficient* conditions for some action/effect
- *explicit* about each individual function, and *implicit* about emergent behaviour

Traditional Specification Languages do not adequately address robustness considerations

Traditional Specs

- designed for *closed* world
- pre- and post condition per function; *sufficient* conditions for some action/effect
- *explicit* about each individual function, and *implicit* about emergent behaviour

Robustness considerations

- concerned with *open* world
- *necessary* conditions for some action/effect
- *explicit* about emergent behaviour

Today

- Traditional Specifications do not adequately address Robustness
- **Holistic Specifications — Summary Examples**
- Holistic Specification Semantics

Holistic Assertions — summary

Holistic Assertions — summary

e ::= this | x | e.fld | …

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

Holistic Assertions — summary

$$
\begin{array}{lll}
e & ::= & \text{this} \mid x \mid e.\text{fld} \mid \ldots \\
A & ::= & e > e \mid e = e \mid \ldots \\
 & & \mid A \rightarrow A \mid A \wedge A \mid \exists x.\, A \mid \ldots
\end{array}
$$

Holistic Assertions — summary

e ::= this | x | e.fld | …
A ::= e>e | e=e | …
| A → A | A ∧ A | ∃x. A | …
| **Access**(e,e’)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …
| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

| A **in** S

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

| A **in** S

| x.**Calls**(y,m,z1,..zn)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

| A **in** S

| x.**Calls**(y,m,z1,..zn)

| x **obeys** A

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …
| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)	permission
| **Changes**(e)	authority
| **Will**(A) | **Was**(A)	time
| A **in** S	space
| x.**Calls**(y,m,z1,..zn)	control
| x **obeys** A	trust

Holistic Assertions — examples

- ERC20
- DAO
- DOM attenuation
- Bank & Account
- Escrow

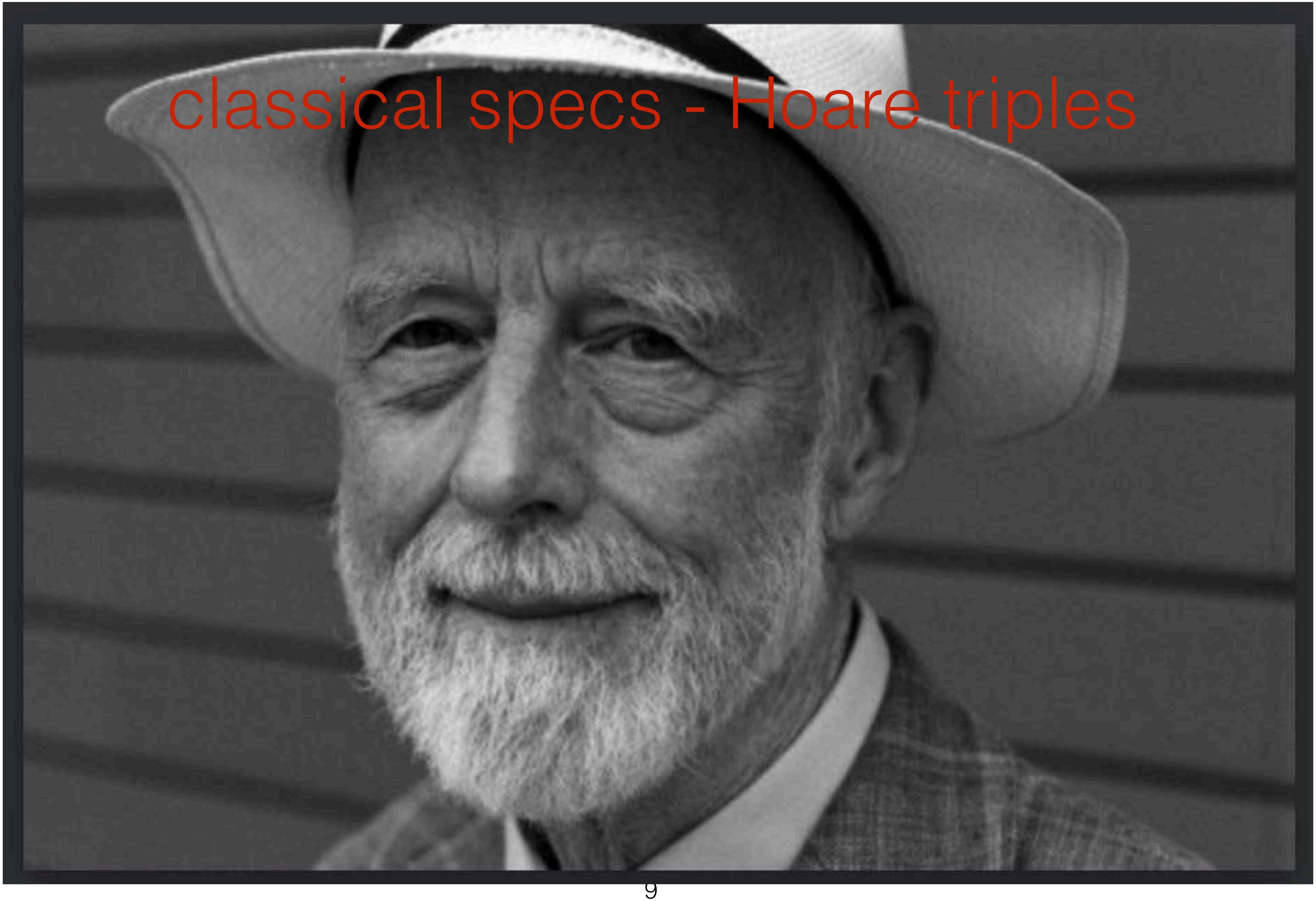
Example1: ERC20

a popular standard for initial coin offerings. (https://theethereum.wiki/w/index.php/ERC20_Token_Standard); allows clients to buy and transfer tokens, and to designate other clients to transfer on their behalf.

In particular, a client may call
- `transfer`: transfer some of her tokens to another clients,
- `approve`: authorise another client to transfer some of her tokens on her behalf.
- `transferFrom`: cause another client's tokens to be transferred

Moreover, ERC20 keeps for each client
- `balance` the number of tokens she owns

classical specs - Hoare triples



classical specs - Hoare triples

classical specs - Hoare triples

A

classical specs - Hoare triples

A { `x` **calls** `y.f(args)` }

classical specs - Hoare triples

A { `x` **calls** `y.f(args)` } **A'**

classical specs - Hoare triples

A { `x` **calls** `y.f(args)` } **A**’

If **A** hods, and `x` calls `y.f(args)`, then, **A’** holds after the call

classical specs - Hoare triples

A { `x` **calls** `y.f(args)` } **A’**

If **A** hods, and `x` calls `y.f(args)`, then, **A’** holds after the call

classical specs - Hoare triples

A { `x` **calls** `y.f(args)` } **A'**

If **A** hods, and `x` calls `y.f(args)`, then, **A'** holds after the call

precondition

postcondition

ERC20 classical spec - transfer

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1's balance decreases by m, and c2's balance increases by m.

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1's balance decreases by m, and c2's balance increases by m.

$$e{:}ERC20 \quad \wedge \quad this = c1 \neq c2 \quad \wedge \quad e.balance(c1) > m$$

$\{$ `e.transfer(c2,m)` $\}$

$$e.balance(c1) = e.balance(c1)_{pre} - m \quad \wedge \quad e.balance(c2) = e.balance(c2)_{pre} + m$$

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1's balance decreases by m, and c2's balance increases by m.

$$e{:}ERC20 \quad \wedge \quad this = c1 \neq c2 \quad \wedge \quad e.balance(c1) > m$$

{ `e.transfer(c2,m)` }

$$e.balance(c1) = e.balance(c1)_{pre} - m \quad \wedge \quad e.balance(c2) = e.balance(c2)_{pre} + m$$

precondition

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1's balance decreases by m, and c2's balance increases by m.

$\text{e:ERC20} \land \text{this} = \text{c1} \neq \text{c2} \land \text{e.balance(c1)} > \text{m}$

{ `e.transfer(c2,m)` }

$\text{e.balance(c1)} = \text{e.balance(c1)}_{\text{pre}} - \text{m} \land \text{e.balance(c2)} = \text{e.balance(c2)}_{\text{pre}} + \text{m}$

precondition

postcondition

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1's balance decreases by m, and c2's balance increases by m.

$$e{:}ERC20 \quad \wedge \quad this = c1 \neq c2 \quad \wedge \quad e.balance(c1) > m$$

$$\{ \quad \texttt{e.transfer(c2,m)} \quad \}$$

$$e.balance(c1) = e.balance(c1)_{pre} - m \quad \wedge \quad e.balance(c2) = e.balance(c2)_{pre} + m$$

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1’s balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1’s balance decreases by m, and c2’s balance increases by m.

$$e{:}ERC20 \quad \wedge \quad this = c1 \neq c2 \quad \wedge \quad e.balance(c1) > m$$

$$\{ \quad \texttt{e.transfer(c2,m)} \quad \}$$

$$e.balance(c1) = e.balance(c1)_{pre} - m \quad \wedge \quad e.balance(c2) = e.balance(c2)_{pre} + m$$

effect

ERC20 classical spec - transfer

For any ERC20 contract e, and different clients c1, c2.

c1's balance is larger than m.

{ c1 calls `e.transfer(c2,m)` }

c1's balance decreases by m, and c2's balance increases by m.

$$e{:}ERC20 \;\wedge\; this = c1 \neq c2 \;\wedge\; e.balance(c1) > m$$

$$\{\; \texttt{e.transfer(c2,m)} \;\}$$

$$e.balance(c1) = e.balance(c1)_{pre} - m \;\wedge\; e.balance(c2) = e.balance(c2)_{pre} + m$$

sufficient condition

effect

ERC20 classical spec - transfer - 2

What if c1’s balance not large enough?

ERC20 classical spec - transfer - 2

What if c1’s balance not large enough?

e:ERC20 $\wedge$ this = c $\wedge$ e,balance(c1) < m
{ `e.transfer(c2,m)` }
$\forall$ c. e,balance(c) = e,balance(c)$_{\text{pre}}$

ERC20 classic spec - authorised transfer

ERC20 classic spec - authorised transfer

For any ERC20 contract e, and different clients c1, c2, c3.

ERC20 classic spec - authorised transfer

For any ERC20 contract e, and different clients c1, c2, c3.

c1 is authorised to spend at least m on c2's behalf and

ERC20 classic spec - authorised transfer

For any ERC20 contract e, and different clients c1, c2, c3.

c1 is authorised to spend at least m on c2's behalf and

c2's balance is at least m

ERC20 classic spec - authorised transfer

For any ERC20 contract e, and different clients c1, c2, c3.

c1 is authorised to spend at least m on c2's behalf and

c2's balance is at least m

{ c1 calls `e.transferFrom(c2,c3,m)` }

ERC20 classic spec - authorised transfer

For any ERC20 contract e, and different clients c1, c2, c3.

c1 is authorised to spend at least m on c2's behalf and

c2's balance is at least m

{ c1 calls `e.transferFrom(c2,c3,m)` }

c2's balance decreases by m, and c3's balance increases by m.

ERC20 classic spec - authorised transfer

For any ERC20 contract e, and different clients c1, c2, c3.

c1 is authorised to spend at least m on c2’s behalf and

c2’s balance is at least m

{ c1 calls `e.transferFrom(c2,c3,m)` }

c2’s balance decreases by m, and c3’s balance increases by m.

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m’) ∧ m’≧ m
e.balance(c1) ≧m
{ e.transferFrom(c2,c3,m) }
e.balance(c1) = e.balance(c1)$_{\text{pre}}$ -m ∧
e.balance(c2) = e.balance(c2)$_{\text{pre}}$ +m ∧
e.Authorized(c1,c2,m’-m)

ERC20 classic spec - authorised transfer - 2

ERC20 classic spec - authorised transfer - 2

What if c1 is not authorised, or c1’’s authorisation is insufficient, or c2 ihas insufficient tokens?

ERC20 classic spec - authorised transfer - 2

What if c1 is not authorised, or c1’’s authorisation is insufficient, or c2 ihas insufficient tokens?

$$
\begin{array}{l}
\texttt{e}:\texttt{ERC20} \ \wedge \ \ \text{this} = c1 \neq c2 \neq c3 \neq c1 \ \wedge \\
(\ \neg \, \texttt{e.Authorized(c1,c2,m)} \\
\quad \vee \ \texttt{e.Authorized(c1,c2,m')} \ \wedge \ \ \texttt{m'<m} \\
\quad \vee \ \texttt{e.balance(c1)} < \texttt{m} \ \ ) \\
\qquad \{ \ \ \texttt{e.transferFrom(cl',m)} \ \ \} \\
\forall\, c. \ \ \texttt{e.balance(c)} = \texttt{e.balance(c)}_{\text{pre}} \ \ \wedge \\
\forall\, c,m. \, [\ \texttt{e.Authorized(c1,c2,m)} \longleftrightarrow \texttt{e.Authorized(c1,c2,m)} \,]
\end{array}
$$

ERC20 classic spec - authorising

ERC20 classic spec - authorising

```
e:ERC20  ∧ this  = c1
     { e.approve(c2,m)  }
e.Authorized(c1,c2,m)
```

ERC20 classical spec

ERC20 classical spec

$$e{:}ERC20 \quad \wedge \quad this = c1 \neq c2 \ \wedge \quad e.balance(c1) > m$$

{ `e.transfer(c2,m)` }

$$e.balance(c1) = e.balance(c1)_{pre} - m \ \wedge \ e.balance(c2) = e.balance(c2)_{pre} + m$$

ERC20 classical spec

$e{:}ERC20 \;\wedge\; this = c1 \neq c2 \;\wedge\; e.balance(c1) > m$

{ `e.transfer(c2,m)` }

$e.balance(c1) = e.balance(c1)_{pre} - m \;\wedge\; e.balance(c2) = e.balance(c2)_{pre} + m$

$e{:}ERC20 \;\wedge\; this = c1 \;\wedge\; e,balance(c1) < m$

{ `e.transfer(c2,m)` }

$\forall\, c.\; e,balance(c) = e,balance(c)_{pre}$

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m
{ `e.transfer(c2,m)` }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧ e.balance(c2) = e.balance(c2)$_{pre}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m
{ `e.transfer(c2,m)` }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m’) ∧ m’≧ m
∧ e.balance(c1) ≧m
{ `e.transferFrom(c2,c3,m)` }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧
e.balance(c2) = e.balance(c2)$_{pre}$ +m ∧ e.Authorized(c1,c2,m’-m)

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m

{ e.transfer(c2,m) }

e.balance(c1) = e.balance(c1)$_{\text{pre}}$ -m ∧ e.balance(c2) = e.balance(c2)$_{\text{pre}}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m

{ e.transfer(c2,m) }

∀ c. e,balance(c) = e,balance(c)$_{\text{pre}}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧

e.Authorized(c1,c2,m’) ∧ m’≧ m

∧ e.balance(c1) ≧m

{ e.transferFrom(c2,c3,m) }

e.balance(c1) = e.balance(c1)$_{\text{pre}}$ -m ∧

e.balance(c2) = e.balance(c2)$_{\text{pre}}$ +m ∧ e.Authorized(c1,c2,m’-m)

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧

(¬ e.Authorized(c1,c2,m) ∨ e.Authorized(c1,c2,m’) ∧ m’<m

∨ e.balance(c1) < m)

{ e.transferFrom(c2,c3,m) }

∀ c. e.balance(c) = e.balance(c)$_{\text{pre}}$ ∧

∀ c,m. [e.Authorized(c1,c2,m)⟷e.Authorized(c1,c2,m)]

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m
{ e.transfer(c2,m) }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧ e.balance(c2) = e.balance(c2)$_{pre}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m
{ e.transfer(c2,m) }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m’) ∧ m’≧ m
∧ e.balance(c1) ≧m
{ e.transferFrom(c2,c3,m) }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧
e.balance(c2) = e.balance(c2)$_{pre}$ +m ∧ e.Authorized(c1,c2,m’-m)

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
(¬ e.Authorized(c1,c2,m) ∨ e.Authorized(c1,c2,m’) ∧ m’<m
∨ e.balance(c1) < m)
{ e.transferFrom(c2,c3,m) }
∀ c. e.balance(c) = e.balance(c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m)⟷e.Authorized(c1,c2,m)]

e:ERC20 ∧ this = c1 { e.aprove(c2,m) } e.Authorized(c1,c2,m)

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m
{ `e.transfer(c2,m)` }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧ e.balance(c2) = e.balance(c2)$_{pre}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m
{ `e.transfer(c2,m)` }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m') ∧ m'≧ m
∧ e.balance(c1) ≧m
{ `e.transferFrom(c2,c3,m)` }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧
e.balance(c2) = e.balance(c2)$_{pre}$ +m ∧ e.Authorized(c1,c2,m'-m)

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
(¬ e.Authorized(c1,c2,m) ∨ e.Authorized(c1,c2,m') ∧ m'<m
∨ e.balance(c1) < m)
{ `e.transferFrom(c2,c3,m)` }
∀ c. e.balance(c) = e.balance(c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m)⟷e.Authorized(c1,c2,m)]

e:ERC2

… { `e.balanceOf(c)` } ….

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m
{ e.transfer(c2,m) }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧ e.balance(c2) = e.balance(c2)$_{pre}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m
{ e.transfer(c2,m) }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m’) ∧ m’≧ m
∧ e.balance(c1) ≧m
{ e.transferFrom(c2,c3,m) }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧
e.balance(c2) = e.balance(c2)$_{pre}$ +m ∧ e.Authorized(c1,c2,m’-m)

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
(¬ e.Authorized(c1,c2,m) ∨ e.Authorized(c1,c2,m’) ∧ m’<m
∨ e.balance(c1) < m)
{ e.transferFrom(c2,c3,m) }
∀ c. e.balance(c) = e.balance(c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m)⟷e.Authorized(c1,c2,m)]

… { e.totalSupply() } ….

… { e.balanceOf(c) } ….

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m
{ `e.transfer(c2,m)` }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧ e.balance(c2) = e.balance(c2)$_{pre}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m
{ `e.transfer(c2,m)` }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m’) ∧ m’≧ m
∧ e.balance(c1) ≧m
{ `e.transferFrom(c2,c3,m)` }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧
e.balance(c2) = e.balance(c2)$_{pre}$ +m ∧ e.Authorized(c1,c2,m’-m)

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
(¬ e.Authorized(c1,c2,m) ∨ e.Authorized(c1,c2,m’) ∧ m’<m
∨ e.balance(c1) < m)
{ `e.transferFrom(c2,c3,m)` }
∀ c. e.balance(c) = e.balance(c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m)⟷e.Authorized(c1,c2,m)]

… { `e.totalSupply()` } ….

… { `e.allowanceOf(c2)` } ….

… { `e.balanceOf(c)` } ….

ERC20 classical spec

e:ERC20 ∧ this = c1≠c2 ∧ e.balance(c1) >m
{ e.transfer(c2,m) }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧ e.balance(c2) = e.balance(c2)$_{pre}$ +m

e:ERC20 ∧ this = c1 ∧ e,balance(c1) < m
{ e.transfer(c2,m) }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
e.Authorized(c1,c2,m’) ∧ m’≧ m
∧ e.balance(c1) ≧m
{ e.transferFrom(c2,c3,m) }
e.balance(c1) = e.balance(c1)$_{pre}$ -m ∧
e.balance(c2) = e.balance(c2)$_{pre}$ +m ∧ e.Autho

sufficient conditions
for change of balance

e:ERC20 ∧ this = c1≠c2≠c3≠c1 ∧
(¬ e.Authorized(c1,c2,m) ∨ e.Authorized(c1,c2,m’) ∧ m’<m
∨ e.balance(c1) < m)
{ e.transferFrom(c2,c3,m) }
∀ c. e.balance(c) = e.balance(c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m)⟷e.Authorized(c1,c2,m)]

… { e.totalSupply() } ….

… { e.allowanceOf(c2) } ….

… { e.balanceOf(c) } ….

ERC20   classical spec

$e{:}ERC20 \;\land\; this = c1 \neq c2 \;\land\; e.balance(c1) > m$

{ `e.transfer(c2,m)` }

$e.balance(c1) = e.balance(c1)_{pre} - m \;\land\; e.balance(c2) = e.balance(c2)_{pre} + m$

$e{:}ERC20 \;\land\; this = c1 \;\land\; e,balance(c1) < m$

{ `e.transfer(c2,m)` }

$\forall c.\ e,balance(c) = e,balance(c)_{pre}$

$e{:}ERC20 \;\land\; this = c1 \neq c2 \neq c3 \neq c1 \;\land$

$e.Authorized(c1,c2,m') \;\land\; m' \geqq m$

$\land\; e.balance(c1) \geqq m$

{ `e.transferFrom(c2,c3,m)` }

$e.balance(c1) = e.balance(c1)_{pre} - m \;\land$

$e.balance(c2) = e.balance(c2)_{pre} + m \;\land\; e.Authorized(c1,c2,m'-m)$

$e{:}ERC20 \;\land\; this = c1 \neq c2 \neq c3 \neq c1 \;\land$

$(\;\neg\, e.Authorized(c1,c2,m) \;\lor\; e.Authorized(c1,c2,m') \;\land\; m' < m$

$\lor\; e.balance(c1) < m\;\;)$

{ `e.transferFrom(c2,c3,m)` }

$\forall c.\ e.balance(c) = e.balance(c)_{pre} \;\land$

$\forall c,m.\ [\; e.Authorized(c1,c2,m) \longleftrightarrow e.Authorized(c1,c2,m) \;]$

… { `e.totalSupply()` } ….

… { `e.allowanceOf(c2)` } ….

… { `e.balanceOf(c)` } ….

ERC20 classical spec

∧ e.balance(c1) >m
r(c2,m) }
ance(c2) = e.balance(c2)$_{pre}$ +m

RC20 ∧ this = c1 ∧ e,balance(c1) < m
{ e.transfer(c2,m) }
∀ c. e,balance(c) = e,balance(c)$_{pre}$

Is that robust?

e.A
∧ e.balance(c1) ≧m
{ e.transferFrom m) }
e.balance(c1) = e.bala pre -m ∧
e.balance(c2) = e.balanc re +m ∧ e.Authorized(c1,c2,m'-m)

e:ERC20 ∧ this = c1≠c2 1 ∧
(¬ e.Authorized(c1,c2 ∨ e.Authorized(c1,c2,m') ∧ m'<m
∨ e.balance(c1) < m)
{ e.transferFrom(c2, m) }
∀ c. e.balance(c) = e.balanc (c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m → e.Authorized(c1,c2,m)]

… { e.totalSupply() } ….
… { e.allowanceOf(c2) } ….
… { e.balanceOf(c) } ….

ERC20 classical spec

∧ e.balance(c1) >m
r(c2,m) }
ance(c2) = e.balance(c2)$_{pre}$ +m

RC20 ∧ this = c1 ∧ e,balance(c1) < m
{ e.transf
∀ c. e,balance(c)

e.
∧ e.balance(c1) ≥m
{ e.transferFrom m) }
e.balance(c1) = e.bala $_{pre}$ -m ∧
e.balance(c2) = e.balanc $_{pre}$ +m ∧ e.Authorized(c1,c2, -m)

e:ERC20 ∧ this = c1≠c2 1 ∧
(¬ e.Authorized(c1,c2 ∨ e.Authorized(c1,c2,m’) ∧ m’<m
∨ e.balance(c1) < m)
{ e.transferFrom(c2, m) }
∀ c. e.balance(c) = e.balanc (c)$_{pre}$ ∧
∀ c,m. [e.Authorized(c1,c2,m) →e.Authorized(c1,c2,m)]

… { e.totalSupply() } ….
… { e.allowanceOf(c2) } ….
… { e.balanceOf(c) } ….

Is that robust?

a function that takes 0.5% from each account?

ERC20 classical spec

∧ e.balance(c1) >m
r(c2,m) }
ance(c2) = e.balance(c2)$_{pre}$ +m

RC20 ∧ this = c1 ∧ e,balance(c1) < m
{ e.transf
∀ c. e,balance(c)

e.:
∧ e.balance(c1) ≥m
{ e.transferFrom m) }
e.balance(c1) = e.bala $_{pre}$ -m ∧
e.balance(c2) = e.balanc re +m ∧ e.Authorized(c1,c2, -m)

e:ERC20 ∧ this = c1≠c2 1 ∧
(¬ e.Authorized(c1,c2 ∨ e.Authorized(c
∨ e.balance(c1) < m)
{ e.transferFrom(c2, m) }
∀c. e.balance(c) = e.balanc (c)$_{pre}$ ∧
∀c,m. [e.Authorized(c1,c2,m) → e.Authorized(c1,c2,m)]

… { e.totalSupply() } ….

… { e.allowanceOf(c2) } ….

… { e.balanceOf(c) } ….

Is that robust?

a function that takes 0.5% from each account?

can authority increase?

ERC20 classical spec

∧
r
ance

RC20 ∧ thi = c1 ∧ e,balance(c1) < m
{ .transf
∀ c. e,ba ance(c)

e.
∧ e.balance(c1) ≧m
{ e.transferFrom m) }
e.balance(c1) = e.bala $_{pre}$ -m ∧
e.balance(c2) = e.balanc $_{re}$ +m ∧ e.Autho ized(c1,c2, -m)

e:ERC20 ∧ this = c1≠c2 1 ∧
(¬ e.Authorized(c1,c2, ∨ e.Authorized(c
∨ e.balance(c1) < m)
{ e.transferFrom(c2, m) }
∀c. e.balance(c) = e.balanc (c)$_{pre}$ ∧
∀c,m. [e.Authorized(c1,c2,m) → e.Authori ed(c1,c2,m)]

… { e.totalSupply() } ….
… { e.allowanceOf(c2) } ….
… { e.balanceOf(c) } ….

Is that robust?

a “super-cleint,” authorised on all?

a function that takes 0.5% from each account?

can authority increase?

ERC20 classical spec

∧

r

ance

RC20 ∧ thi = c1 ∧ e,balance(c1) < m

{ .transf

∀ c. e,ba ance(c)

e.

∧ e.balance(c1) ≧m

{ e.transferFrom m) }

e.balance(c1) = e.bala $_{pre}$ -m ∧

e.balance(c2) = e.balanc re +m ∧ e.Autho ized(c1,c2, -m)

1 ∧

∨ e.Authorized(c

2, m) }

anc (c) $_{pre}$ ∧

c2,m, →e.Authori ed(c1,c2,m)]

.totalSupply() } ….

… { e.allowanceOf(c2) } ….

… { e.balanceOf(c) } ….

Is that robust?

a “super-cleint,”
authorised on all?

a function that
takes 0.5% from
each account?

can authority increase?

ERC20 classical spec

Is that robust?

a “super-cleint,” authorised on all?

a function that takes 0.5% from each account?

can authority increase?

∧

r

ance

RC20 ∧ thi = c1 ∧ e,balance(c1) < m

{ .transf

∀ c. e,ba ance(c)

e./

∧ e.balance(c1) ≥m

{ e.transferFrom m) }

e.balance(c1) = e.bala $_{pre}$ -m ∧

e.balance(c2) = e.balanc $_{re}$ +m ∧ e.Autho ized(c1,c2, -m)

I am worried about who/what can reduce my balance

zed(c

}

anc (c) $_{pre}$ ∧

c2,m) →e.Authori ed(c1,c2,m)]

.totalSupply() } ….

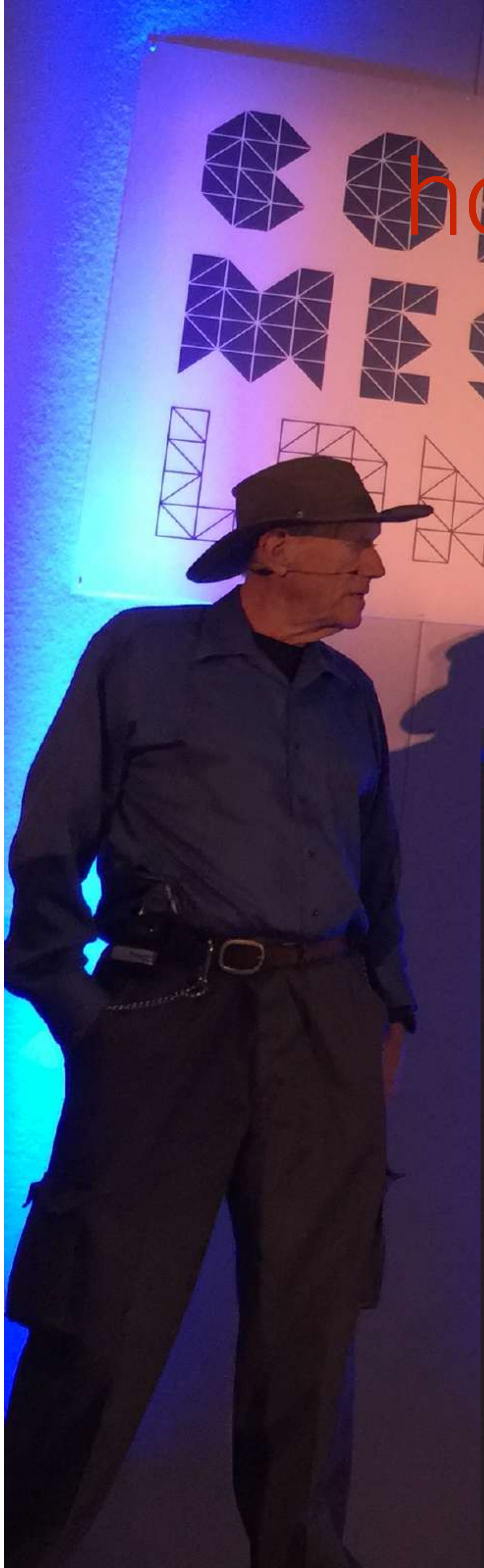
… { e.allowanceOf(c2) } ….

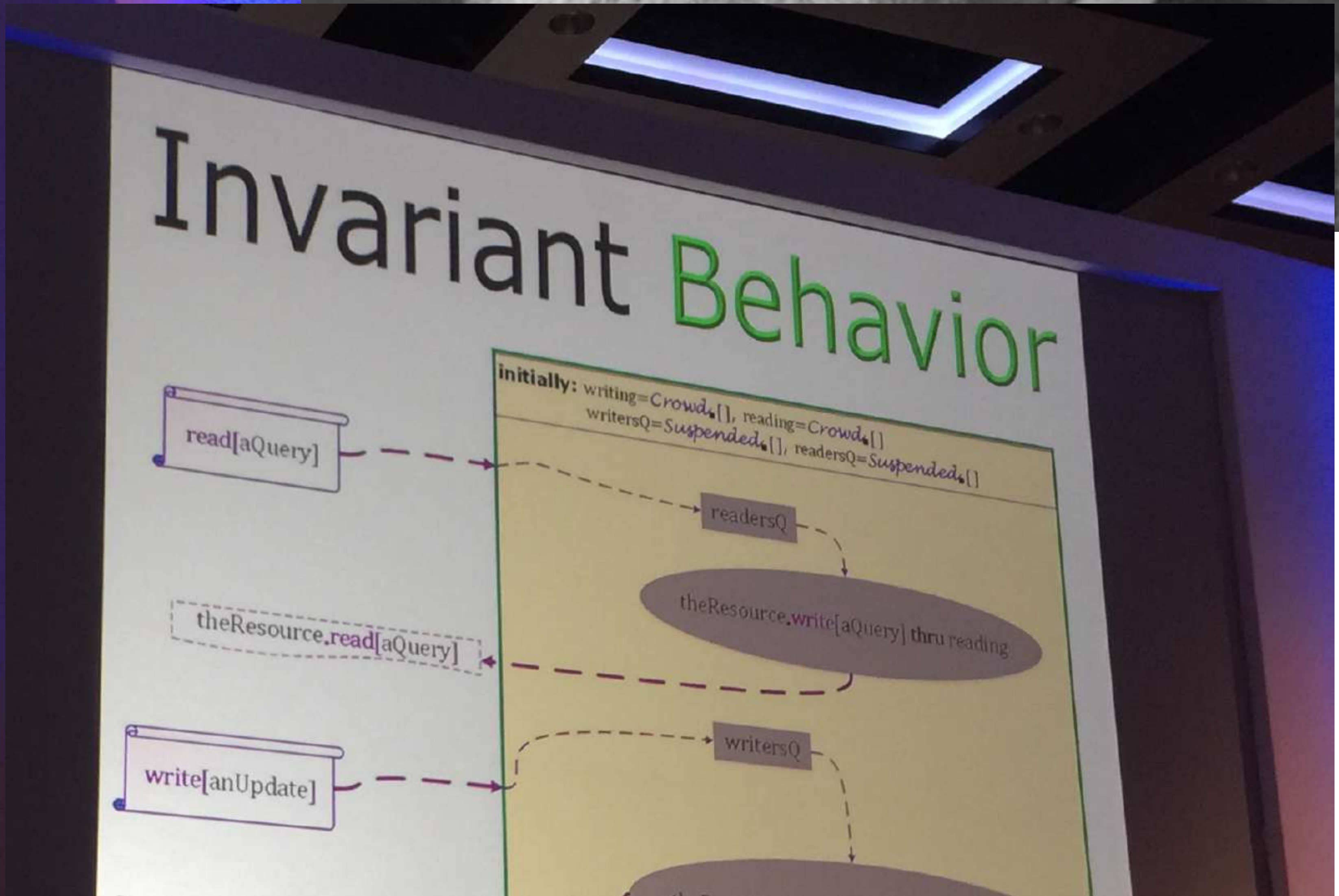
… { e.balanceOf(c) } ….

holistic specs - invariants

holistic specs - invariants

holistic specs - invariants





Invariant Behavior

initially: writing=Crowd•[], reading=Crowd•[]
writersQ=Suspended•[], readersQ=Suspended•[]

read[aQuery]

readersQ

theResource.write[aQuery] thru reading

theResource.read[aQuery]

writersQ

write[anUpdate]

holistic specs - invariants

- state

holistic specs - invariants ++

- state
- **time**
- **space**
- **control**
- **permission**
- **authority**
- **(when/where do they hold)**

holistic spec - reduce balance

holistic spec - reduce balance

∀ `e`:ERC20. ∀ `c1`: Client. ∀ m: Nat.

[`e.balance(c1) =` **Was**`(e.balance(c1))-m`

holistic spec - reduce balance

∀ `e`:ERC20. ∀ `c1`: Client. ∀ m: Nat.

[`e.balance(c1) =` **Was**(`e.balance(c1)-m`

⟶

holistic spec - reduce balance

∀ e:ERC20. ∀ c1: Client. ∀ m: Nat.

[e.balance(c1) = **Was**(e.balance(c1))-m

⟶

(∃c2,c3: Client.

Was (c1.**Calls**(e,transfer,c2,m)))

holistic spec - reduce balance

∀ e:ERC20. ∀ c1: Client. ∀ m: Nat.

[e.balance(c1) = **Was**(e.balance(c1)) - m

⟶

(∃c2,c3: Client.

Was (c1.**Calls**(e,transfer,c2,m)))

∨

Was (e.Authorized(c1,c2,m) ∧

c2.**Calls**(e,transferFrom,c1,c3,m)))   ]

holistic spec - reduce balance

∀ e:ERC20. ∀ c1: Client. ∀ m: Nat.
[e.balance(c1) = **Was**(e.balance(c1)) - m

⟶

(∃c2,c3: Client.
Was (c1.**Calls**(e,transfer,c2,m)))
∨
Was (e.Authorized(c1,c2,m) ∧
c2.**Calls**(e,transferFrom,c1,c3,m)))]

This says: A client's balance decreases *only* if that client, or somebody authorised by that client, made a payment.

holistic spec - reduce balance

∀ e:ERC20. ∀ c1: Client. ∀ m: Nat.
[e.balance(c1) = **Was**(e.balance(c1))-m

⟶

(∃c2,c3: Client.
Was (c1.**Calls**(e,transfer,c2,m)))
∨
Was (e.Authorized(c1,c2,m) ∧
c2.**Calls**(e,transferFrom,c1,c3,m)))]

effect

This says: A client's balance decreases *only* if that client, or somebody authorised by that client, made a payment.

holistic spec - reduce balance

∀ e:ERC20. ∀ c1: Client. ∀ m: Nat.

[e.balance(c1) = **Was**(e.balance(c1)) - m

⟶

(∃c2,c3: Client.

Was (c1.**Calls**(e,transfer,c2,m)))

∨

Was (e.Authorized(c1,c2,m) ∧
c2.**Calls**(e,transferFrom,c1,c3,m)))   ]

effect

necessary
condition

This says: A client's balance decreases *only* if that client, or somebody authorised by that client, made a payment.

holistic spec - authority

holistic spec - authority

`e.Authorized(c1,c2,m)` ≐

`c2` is authorised by `c1` for `m` iff

holistic spec - authority

`e.`Authorized(`c1,c2`,`m`) ≐

Was(`c1`.**Calls**(`e,approve,`c2,`m`))

`c2` is authorised by `c1` for `m` iff

in previous step `c1` informed `e` that it authorised `c2` for `m`

holistic spec - authority

$$e.\mathrm{Authorized}(c1,c2,m) \stackrel{\cdot}{\equiv}$$

$$\textbf{Was}(\,c1.\textbf{Calls}(e,\mathtt{approve},c2,m)\,)$$

$$\vee$$

$$\textbf{Was}(\,e.\mathrm{Authorized}(c1,c2,m+m') \wedge c2.\textbf{Calls}(e,\mathtt{transferFrom},c1,\_,m')\,)$$

c2 is authorised by c1 for m iff

in previous step c1 informed e that it authorised c2 for m

or

in previous step c2 was authorised for m+m’ and spent m’ for c1

holistic spec - authority

$$e.\text{Authorized}(c1,c2,m) \stackrel{\cdot}{\equiv}$$

$$\textbf{Was}(\ c1.\textbf{Calls}(e,\text{approve},c2,m)\ )$$

$$\vee$$

$$\textbf{Was}(\ e.\text{Authorized}(c1,c2,m+m')\ \wedge c2.\textbf{Calls}(e,\text{transferFrom},c1,\_,m')\ )$$

$$\vee$$

$$\textbf{Was}(\ e.\text{Authorized}(c1,c2,m)\ \wedge \neg\, c2.\textbf{Calls}(e,\text{transferFrom},c1,\_,\_)\ )$$

c2 is authorised by c1 for m iff

in previous step c1 informed e that it authorised c2 for m

or

in previous step c2 was authorised for m+m’ and spent m’ for c1

or

in previous step c2 was authorised for m and did not spend c1

holistic spec - authority

effect

$$\boxed{\texttt{e.Authorized(c1,c2,m)}} \doteq$$

$$\textbf{Was}(\ \texttt{c1.}\textbf{Calls}(\texttt{e,approve,c2,m})\ )$$

$$\vee$$

$$\textbf{Was}(\ \texttt{e.Authorized(c1,c2,m+m')}\ \wedge\ \texttt{c2.}\textbf{Calls}(\texttt{e,transferFrom,c1,\_,m'})\ )$$

$$\vee$$

$$\textbf{Was}(\ \texttt{e.Authorized(c1,c2,m)}\ \wedge\ \neg\ \texttt{c2.}\textbf{Calls}(\texttt{e,transferFrom,c1,\_,\_})\ )$$

c2 is authorised by c1 for m iff

in previous step c1 informed e that it authorised c2 for m

or

in previous step c2 was authorised for m+m’ and spent m’ for c1

or

in previous step c2 was authorised for m and did not spend c1

holistic spec - authority

effect

e.Authorized(c1,c2,m) ≐

Was(c1.**Calls**(e,approve,c2,m))

∨

Was(e.Authorized(c1,c2,m+m') ∧ c2.**Calls**(e,transferFrom,c1,_,m'))

∨

Was(e.Authorized(c1,c2,m) ∧ ¬ c2.**Calls**(e,transferFrom,c1,_,_))

necessary
conditions

c2 is authorised by c1 for m iff

in previous step c1 informed e that it authorised c2 for m

or

in previous step c2 was authorised for m+m' and spent m' for c1

or

in previous step c2 was authorised for m and did not spend c1

classical vs holistic

classical vs holistic

$$e{:}\mathrm{ERC20} \wedge e.\mathrm{balance}(cl) > m \wedge e.\mathrm{balance}(cl') = m' \wedge cl \neq cl'$$
$$\{\ \texttt{e.transfer}(cl', m) \wedge \mathrm{Caller} = cl \}$$
$$e.\mathrm{balance}(cl) = e.\mathrm{balance}(cl)_{pre} - m \wedge e.\mathrm{balance}(cl')_{pre} = m' + m$$

$$e{:}\mathrm{ERC20} \wedge e.\mathrm{balance}(cl) > m \wedge e.\mathrm{balance}(cl') = m' \wedge cl \neq cl'$$
$$\wedge \mathrm{Authorized}(e, cl, cl'')$$
$$\{\ \texttt{e.transferFrom}(cl', m) \wedge \mathrm{Caller} = cl'' \}$$
$$e.\mathrm{balance}(cl) = e.\mathrm{balance}(cl)_{pre} - m \wedge e.\mathrm{balance}(cl')_{pre} = m' + m$$

$$e{:}\mathrm{ERC20} \wedge e.\mathrm{balance}(cl) > m \wedge e.\mathrm{balance}(cl') = m'$$
$$\{\ \texttt{e.allow}(cl') \wedge \mathrm{Caller} = cl \ \}$$
$$\mathrm{Authorized}(e, cl, cl'')$$

Classical

- per function; :*sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

classical vs holistic

$e{:}ERC20 \wedge\ e.balance(cl) > m\ \wedge\ e.balance(cl') = m'\ \wedge cl \neq cl'$

$\{\ \texttt{e.transfer}(cl',m)\ \wedge Caller{=}cl\}$

$e.balance(cl) = e.balance(cl)_{pre} - m\ \wedge\ e.balance(cl')_{pre} = m'+m$

$e{:}ERC20 \wedge\ e.balance(cl) > m\ \wedge\ e.balance(cl') = m'\ \wedge cl \neq cl'$

$\wedge\ \ Authorized\,(e, cl, cl'')$

$\{\ \texttt{e.transferFrom}(cl',m)\ \wedge Caller{=}cl''\}$

$e.balance(cl) = e.balance(cl)_{pre} - m\ \wedge\ e.balance(cl')_{pre} = m'+m$

$e{:}ERC20 \wedge\ e.balance(cl) > m\ \wedge\ e.balance(cl') = m'$

$\{\ \texttt{e.allow}(cl')\ \wedge Caller{=}cl\ \}$

$Authorized\,(e, cl, cl'')$

… … another 7 specs … ….

Classical

- per function; :*sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

classical vs holistic

e:ERC20 ∧ e.balance(cl) >m ∧ e.balance(cl’) = m’ ∧ cl≠cl’
{ `e.transfer`(cl’,m) ∧ Caller=cl}
e.balance(cl) = e.balance(cl)$_{pre}$ -m ∧ e.balance(cl’)$_{pre}$ = m’+m

e:ERC20 ∧ e.balance(cl) >m ∧ e.balance(cl’) = m’ ∧ cl≠cl’
∧ Authorized(e, cl, cl’’)
{ `e.transferFrom`(cl’,m) ∧ Caller=cl’’}
e.balance(cl) = e.balance(cl)$_{pre}$ -m ∧ e.balance(cl’)$_{pre}$ = m’+m

e:ERC20 ∧ e.balance(cl) >m ∧ e.balance(cl’) = m’
{ `e.allow`(cl’) ∧ Caller=cl }
Authorized(e, cl, cl’’)

… … another 7 specs … ….

Classical

- per function; :*sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

Holistic

- *necessary* conditions for some action/effect
- *explicit* about emergent behaviour

∀ e:ERC20. ∀ cl: Client.
[e.balance(cl) = **Was**(e.balance(cl)) - m)
→
(∃cl’,cl’’: Client.
Was (cl.**Calls**(`e.transfer`(cl’,m)))
∨
Was (Authorized(e,cl, cl’’) ∧ cl’’.**Calls**(`e.transferFrom`(cl,cl’,m)))]

Authorized(cl, cl’) ≐ ∃m: Nat. **Was***(cl.**Calls**(`e.approve`(cl’,m)))

classical vs holistic

$e{:}ERC20 \wedge e.balance(cl) > m \wedge e.balance(cl') = m' \wedge cl \neq cl'$

$\{\ \texttt{e.transfer}(cl', m) \wedge Caller = cl \}$

$e.balance(cl) = e.balance(cl)_{pre} - m \wedge e.balance(cl')_{pre} = m' + m$

$e{:}ERC20 \wedge e.balance(cl) > m \wedge e.balance(cl') = m' \wedge cl \neq cl'$

$\wedge\ Authorized(e, cl, cl'')$

$\{\ \texttt{e.transferFrom}(cl', m) \wedge Caller = cl'' \}$

$e.balance(cl) = e.balance(cl)_{pre} - m \wedge e.balance(cl')_{pre} = m' + m$

$e{:}ERC20 \wedge e.balance(cl) > m \wedge e.balance(cl') = m'$

$\{\ \texttt{e.allow}(cl') \wedge Caller = cl \ \}$

$Authorized(e, cl, cl'')$

… … another 7 specs … ….

Classical

- per function; :*sufficient* conditions for some action/ effect
- *explicit* about individual function, and *implicit* about emergent behaviour

Holistic

- *necessary* conditions for some action/effect
- *explicit* about emergent behaviour

$\forall\ e{:}ERC20.\ \forall\ cl{:}\ Client.$

$[\ e.balance(cl) = \textbf{Was}(e.balance(cl)) - m\ )$

$\rightarrow$

$(\ \exists cl', cl''{:}\ Client.$

$\textbf{Was}(\ cl.\textbf{Calls}(\texttt{e.transfer}(cl', m))\ )$

$\vee$

$\textbf{Was}(\ Authorized(e, cl, cl'') \wedge cl''.\textbf{Calls}(\texttt{e.transferFrom}(cl, cl', m))\ )\ ]$

$Authorized(cl, cl') \doteq \exists m{:}\ Nat.\ \textbf{Was*}(\ cl.\textbf{Calls}(\texttt{e.approve}(cl', m))\ )$

Example2: DAO simplified

DAO, a “hub that disperses funds”; (https://www.ethereum.org/dao).
… clients may contribute and retrieve funds :

- `payIn(m)` pays into DAO m on behalf of client
- `repay()` withdraws all moneys from DAO

Vulnerability: Through a buggy version of `repay()`, a client could re-enter the call and deplete all funds of the DAO.

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO.\quad d.ether = \sum_{\texttt{cl} \in dom(\texttt{d.directory})} \texttt{d.directory(cl)}$

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO. \quad d.ether = \sum_{\mathtt{cl} \in dom(\mathtt{d.directory})} \mathtt{d.directory(cl)}$

R2: $d{:}DAO \ \wedge \ n{:}Nat \ \wedge \ d.directory(cl)=n>0 \ \wedge \ this=cl$

$\{$ `d.repay()` $\}$

$d.directory(cl)=0 \ \wedge d.\mathbf{Calls}(cl,send,\mathtt{n})$

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO. \quad d.ether = \sum_{\mathtt{cl} \in dom(\mathtt{d.directory})} \mathtt{d.directory(cl)}$

R2: $d{:}DAO \ \wedge \ n{:}Nat \ \wedge \ d.directory(cl)=n>0 \ \wedge \ this=cl$

{ `d.repay()` }

$d.directory(cl)=0 \ \wedge d.\mathbf{Calls}(cl,send,n)$

$d{:}DAO \ \wedge \ n{:}Nat \ \wedge \ d.directory(cl)=0 \ \wedge \ this=cl$

{ `d.repay()` }

“nothing changes”

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO.\quad d.ether = \sum_{\mathtt{cl} \in dom(\mathtt{d.directory})} \mathtt{d.directory(cl)}$

R2: $d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl)=n>0 \ \wedge\ this=cl$

$\{\ \mathtt{d.repay()}\ \}$

$d.directory(cl)=0 \ \wedge\ d.\mathbf{Calls}(cl,send,n)$

$d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl)=0 \ \wedge\ this=cl$

$\{\ \mathtt{d.repay()}\ \}$

“nothing changes”

This spec avoids the vulnerability, 😅

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO.\quad d.ether = \sum_{\mathtt{cl} \in dom(\mathtt{d.directory})} \mathtt{d.directory(cl)}$

R2: $d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl)=n>0 \ \wedge\ this=cl$

$\{\ $ `d.repay()` $\ \}$

$d.directory(cl)=0 \ \wedge d.\mathbf{Calls}(cl,send,\mathtt{n})$

$d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl)=0 \ \wedge\ this=cl$

$\{\ $ `d.repay()` $\ \}$

“nothing changes”

This spec avoids the vulnerability, 😅

provided the attack goes through the function repay. 😢

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO.\quad d.ether = \sum_{\mathtt{cl} \in dom(\mathtt{d.directory})} \mathtt{d.directory(cl)}$

R2: $d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl)=n>0 \ \wedge\ this=cl$

$\{\ \mathtt{d.repay()}\ \}$

$d.directory(cl)=0 \ \wedge\ d.\mathbf{Calls}(cl,send,\mathtt{n})$

$d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl)=0 \ \wedge\ this=cl$

$\{\ \mathtt{d.repay()}\ \}$

“nothing changes”

This spec avoids the vulnerability, 😅

provided the attack goes through the function repay. 😢

To avoid the vulnerability in general, we need to inspect the specification of *all* the functions in the DAO.

classical spec

Assuming DAO keeps a directory of contributions, and require:

R1: directory is compatible with the amount of ether kept in the DAO, and

R2: that withdraw reduces the ether by that amount.

R1: $\forall d{:}DAO.\quad d.ether = \sum_{\texttt{cl} \in dom(\texttt{d.directory})} \texttt{d.directory(cl)}$

R2: $d{:}DAO \;\wedge\; n{:}Nat \;\wedge\; d.directory(cl){=}n{>}0 \;\wedge\; this{=}cl$

{ `d.repay()` }

$d.directory(cl){=}0 \;\wedge\; d.$**Calls**$(cl,send,n)$

$d{:}DAO \;\wedge\; n{:}Nat \;\wedge\; d.directory(cl){=}0 \;\wedge\; this{=}cl$

{ `d.repay()` }

“nothing changes”

This spec avoids the vulnerability, 😅

provided the attack goes through the function repay. 😢

To avoid the vulnerability in general, we need to inspect the specification of *all* the functions in the DAO. DAO - interface has *nineteen* functions. 🥵

holistic

holistic

∀ cl:External. **∀**d:DAO. **∀**.n:Nat.
[cl.**Calls**(`d.repay()`) ∧ d.Balance(cl) = n
→
d.ether≧n ∧ **Will**(d.**Calls**(cl.send(n))]

holistic

$$
\begin{array}{l}
\forall\ \mathrm{cl{:}External}.\ \forall \mathrm{d{:}DAO}.\ \forall.\mathrm{n{:}Nat}. \\
[\ \ \mathrm{cl}.\textbf{Calls}(\texttt{d.repay()}\ )\ \wedge\ \mathrm{d.Balance(cl)} = \mathrm{n} \\
\quad \rightarrow \\
\quad \mathrm{d.ether} \geqq \texttt{n} \wedge\ \textbf{Will}(\mathrm{d}.\textbf{Calls}(\mathrm{cl.send}(\texttt{n}\ ))\ \ ]
\end{array}
$$

This specification avoids the vulnerability, regardless of which function introduces it:
The DAO will always be able to repay all its customers.

holistic

$$
\begin{aligned}
&\forall\ \text{cl:External. } \forall\text{d:DAO. } \forall\text{.n:Nat.}\\
&[\ \ \text{cl.}\mathbf{Calls}(\texttt{d.repay()}\ )\ \wedge\ \text{d.Balance(cl)} = \text{n}\\
&\quad \rightarrow\\
&\quad \text{d.ether}\geqq\texttt{n}\ \wedge\ \mathbf{Will}(\text{d.}\mathbf{Calls}(\text{cl.send(}\texttt{n}\text{))}\ \ ]
\end{aligned}
$$

This specification avoids the vulnerability, regardless of which function introduces it:
The DAO will always be able to repay all its customers.

$\text{d.Balance(cl)} \equiv$

holistic

$$\begin{array}{l} \forall\ cl{:}External.\ \forall d{:}DAO.\ \forall.n{:}Nat. \\ [\ \ cl.\mathbf{Calls}(\texttt{d.repay()}\ )\ \wedge\ d.Balance(cl) = n \\ \quad \rightarrow \\ \quad d.ether \geqq \texttt{n}\ \wedge\ \mathbf{Will}(d.\mathbf{Calls}(cl.send(\texttt{n}\ ))\ \ ] \end{array}$$

This specification avoids the vulnerability, regardless of which function introduces it:
The DAO will always be able to repay all its customers.

$$d.Balance(cl) \equiv \quad 0 \qquad \text{if}\ \ cl.\mathbf{Calls}(d,initialize()\ )$$

holistic

∀ cl:External. **∀**d:DAO. **∀**.n:Nat.
[cl.**Calls**(`d.repay()`) ∧ d.Balance(cl) = n
→
d.ether≥`n` ∧ **Will**(d.**Calls**(cl.send(`n`))]

This specification avoids the vulnerability, regardless of which function introduces it:
The DAO will always be able to repay all its customers.

d.Balance(cl) ≡ 0 if cl.**Calls**(d,initialize())
m+m' if **Was**(d.Balance(cl),m) ∧ cl.**Calls**(d.payIn(m'))

holistic

$$
\begin{array}{l}
\forall\ cl{:}External.\ \forall d{:}DAO.\ \forall .n{:}Nat. \\
[\ \ cl.\mathbf{Calls}(\texttt{d.repay()}\ )\ \wedge\ d.Balance(cl) = n \\
\quad \rightarrow \\
\quad d.ether \geqq \texttt{n}\ \wedge\ \mathbf{Will}(d.\mathbf{Calls}(cl.send(\texttt{n}\ ))\ \ ]
\end{array}
$$

This specification avoids the vulnerability, regardless of which function introduces it:
The DAO will always be able to repay all its customers.

$$
d.Balance(cl) \equiv
\begin{array}{ll}
0 & \text{if}\ \ cl.\mathbf{Calls}(d,initialize()\ ) \\
m+m' & \text{if}\ \ \mathbf{Was}(d.Balance(cl),m)\ \ \wedge\ cl.\mathbf{Calls}(d.payIn(m')\ \ ) \\
0 & \text{if}\ \ \mathbf{Was}(cl.\mathbf{Calls}(d.repayIn()\ \ )
\end{array}
$$

holistic

$$\forall\ cl{:}External.\ \forall d{:}DAO.\ \forall.n{:}Nat.$$

$$[\ \ cl.\mathbf{Calls}(\texttt{d.repay()}\ )\ \wedge\ d.Balance(cl) = n$$

$$\rightarrow$$

$$d.ether \geqq \texttt{n} \wedge\ \mathbf{Will}(d.\mathbf{Calls}(cl.send(\texttt{n}\ ))\ \ ]$$

This specification avoids the vulnerability, regardless of which function introduces it:
The DAO will always be able to repay all its customers.

$$d.Balance(cl) \equiv \begin{cases} 0 & \text{if } cl.\mathbf{Calls}(d,initialize()\ ) \\ m+m' & \text{if } \mathbf{Was}(d.Balance(cl),m)\ \wedge\ cl.\mathbf{Calls}(d.payIn(m')\ \ ) \\ 0 & \text{if } \mathbf{Was}(cl.\mathbf{Calls}(d.repayIn()\ \ ) \\ \mathbf{Was}(d.Balance(cl)) & \text{otherwise} \end{cases}$$

classical vs holistic

$\forall d{:}DAO.\quad d.ether = \sum_{\texttt{cl} \in dom(\texttt{d.directory})} \texttt{d.directory(cl)}$

$d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl) = n > 0 \wedge\ \texttt{this} = cl$
{ `d.repay()` }
$d.directory(cl) = 0 \ \wedge d.Calls(cl.send(\texttt{n}))$

$d{:}DAO \ \wedge\ n{:}Nat \ \wedge\ d.directory(cl) = 0 \ \wedge\ this = cl$
{ `d.repay()` }
“nothing changes”

classical vs holistic

$$\forall d{:}DAO.\quad d.ether = \sum_{\texttt{cl} \,\in\, dom(\texttt{d.directory})} \texttt{d.directory(cl)}$$

$d{:}DAO \;\wedge\; n{:}Nat \;\wedge\; d.directory(cl)=n>0 \;\wedge\; \texttt{this}=cl$

{ `d.repay()` }

$d.directory(cl)=0 \;\wedge\; d.Calls(cl.send(\texttt{n}))$

$d{:}DAO \;\wedge\; n{:}Nat \;\wedge\; d.directory(cl)=0 \;\wedge\; this=cl$

{ `d.repay()` }

"nothing changes"

Classical

- per function; *sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

classical vs holistic

$\forall$d:DAO.   d.ether = $\sum_{\texttt{cl} \in \text{dom}(\texttt{d.directory})}$ `d.directory(cl)`

d:DAO ∧ n:Nat ∧ d.directory(cl)=n>0 ∧ `this`=cl
{ `d.repay()` }
d.directory(cl)=0 ∧ d.Calls(cl.send(n))

d:DAO ∧ n:Nat ∧ d.directory(cl)=0 ∧ this=cl
{ `d.repay()` }
“nothing changes”

… … specs for another 19 functions ….

Classical

- per function; *sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

classical vs holistic

$\forall$d:DAO. d.ether = $\sum_{\texttt{cl} \in \mathrm{dom}(\texttt{d.directory})}$ `d.directory(cl)`

d:DAO ∧ n:Nat ∧ d.directory(cl)=n>0 ∧ `this`=cl
{ `d.repay()` }
d.directory(cl)=0 ∧ d.Calls(cl.send(`n`))

d:DAO ∧ n:Nat ∧ d.directory(cl)=0 ∧ this=cl
{ `d.repay()` }
“nothing changes”

… … specs for another 19 functions ….

Classical

- per function; *sufficient* conditions for some action/effect
- *explicit* about individual function, and *implicit* about emergent behaviour

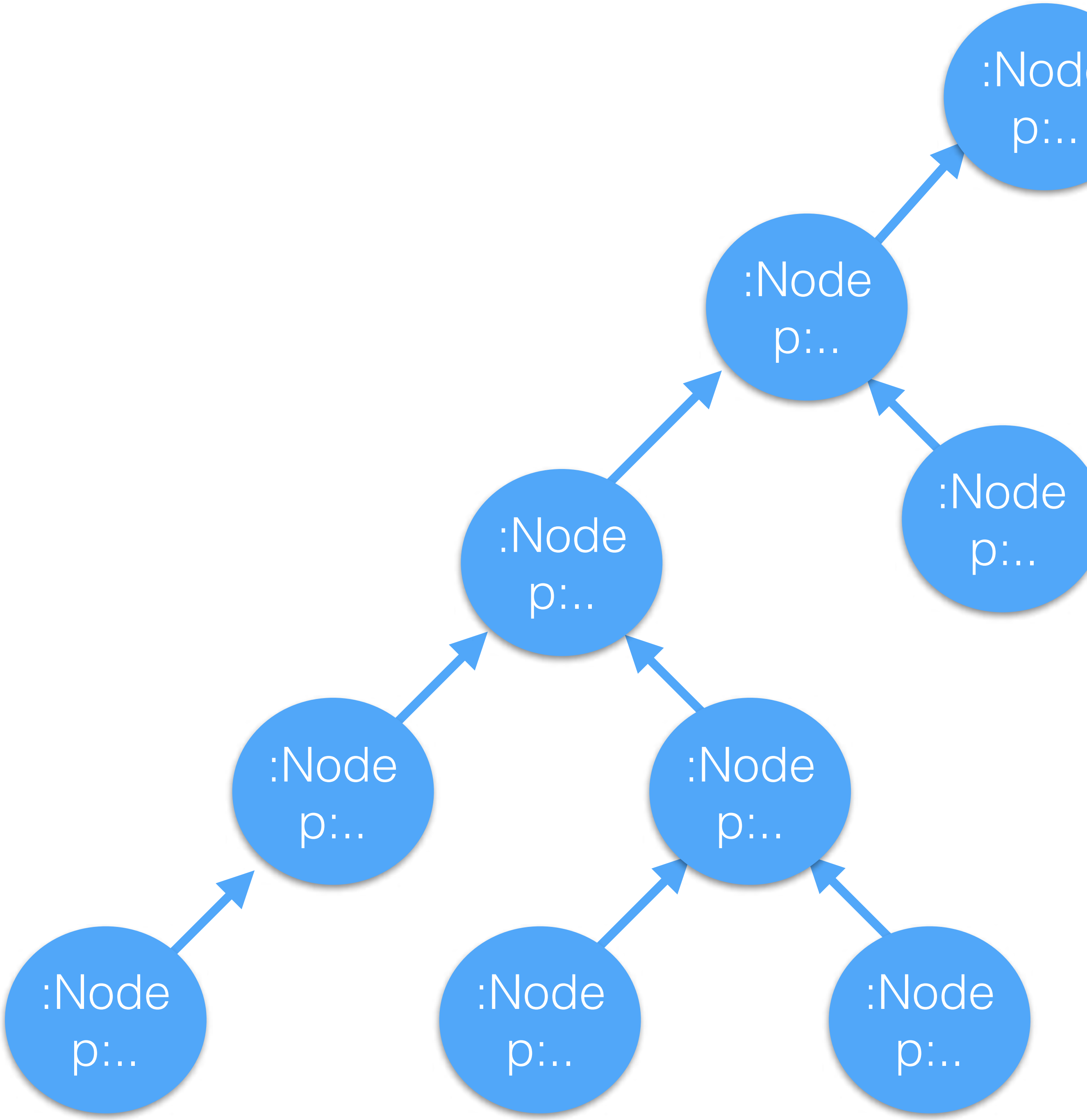
Holistic

- *necessary* conditions for some action/effect
- *explicit* about emergent behaviour

∀ cl:External. ∀d:DAO. ∀.n:Nat.
[cl.**Calls**(`d.repay()`) ∧ d.Balance(cl) = n
→
d.ether≥`n` ∧ **Will**(d.**Calls**(cl.send(`n`))]

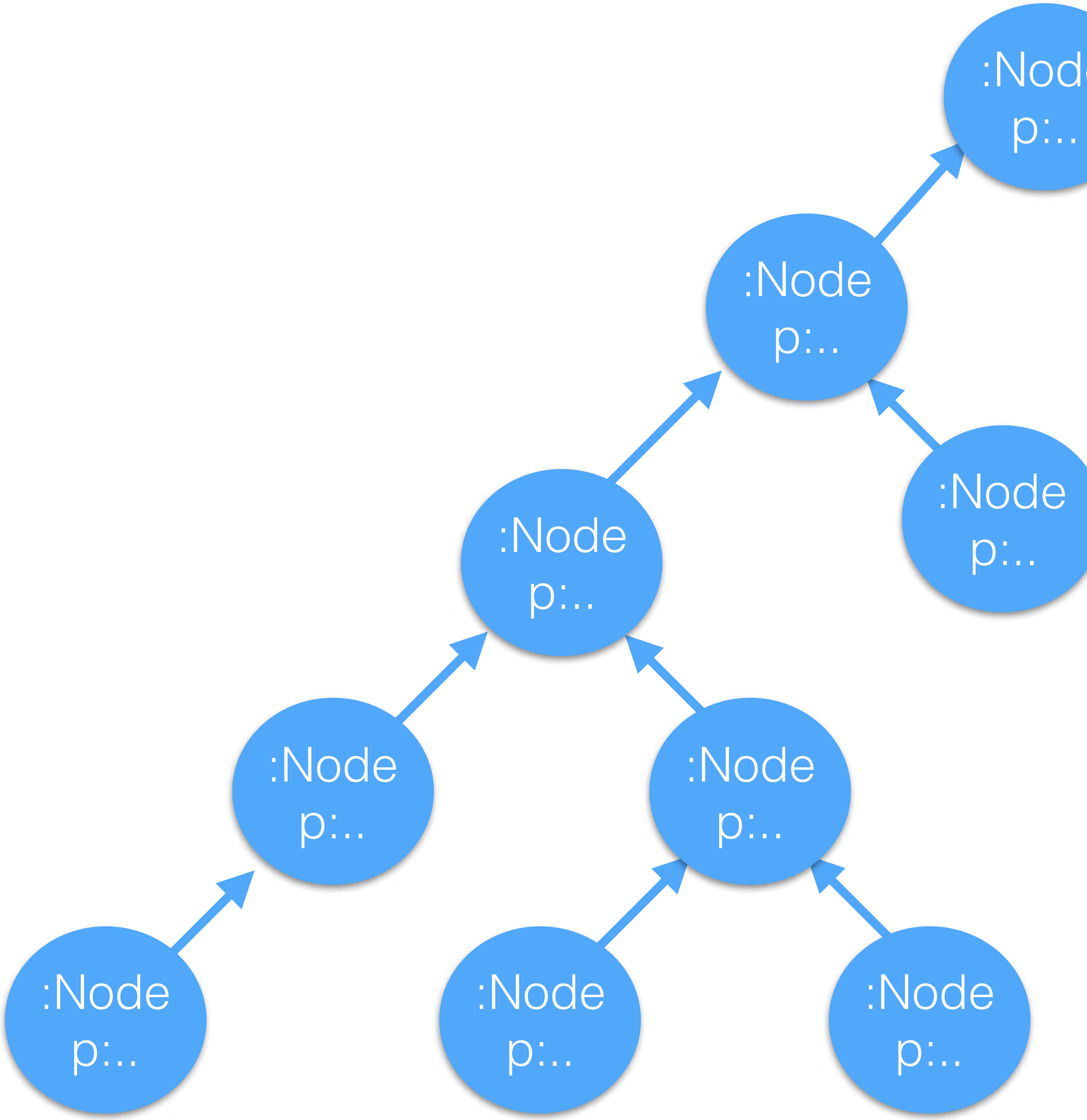
d.Balance(cl) ≡	m	if cl.**Calls**(d,initialize,m)
	m+m’	if **Was**(d.Balance(cl),m) ∧ **Was**(cl.**Calls**(d.payIn(m’))
	0	if **Was**(cl.**Calls**(d.repayIn())

Example 3: DOM attenuation



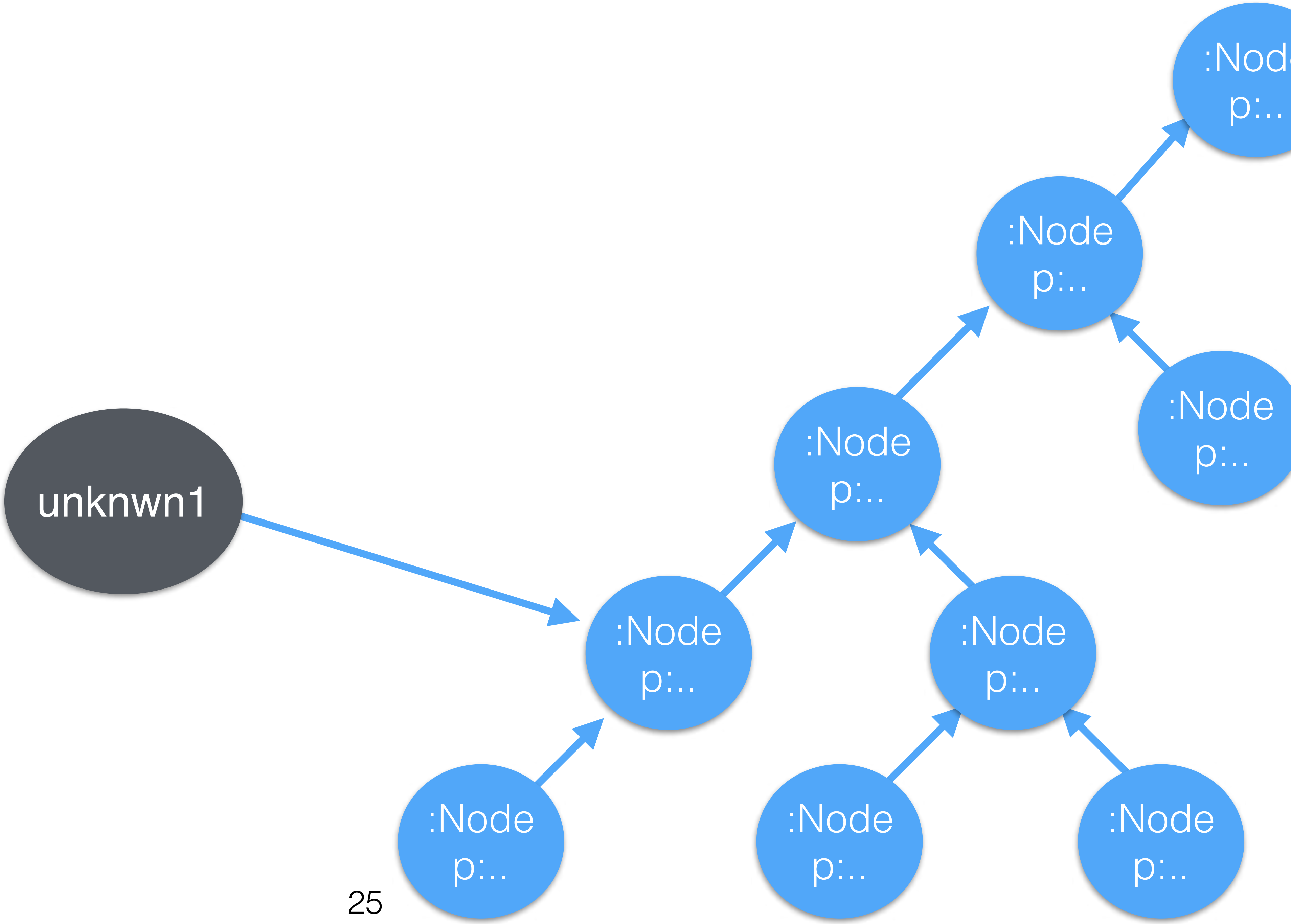
Example 3: DOM attenuation

Access to any `Node` gives access to ***complete*** tree



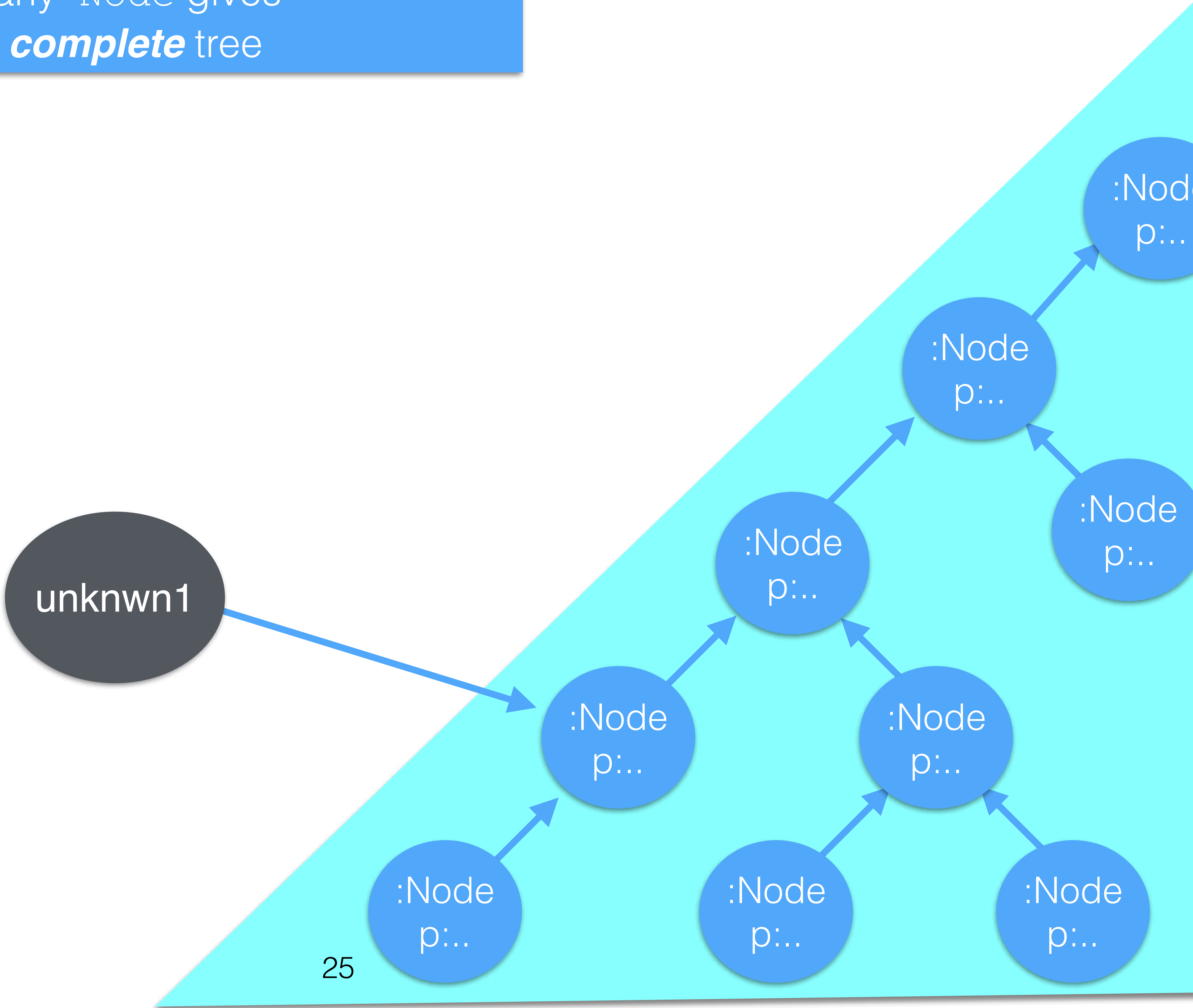
Example 3: DOM attenuation

Access to any `Node` gives access to ***complete*** tree



Example 3: DOM attenuation

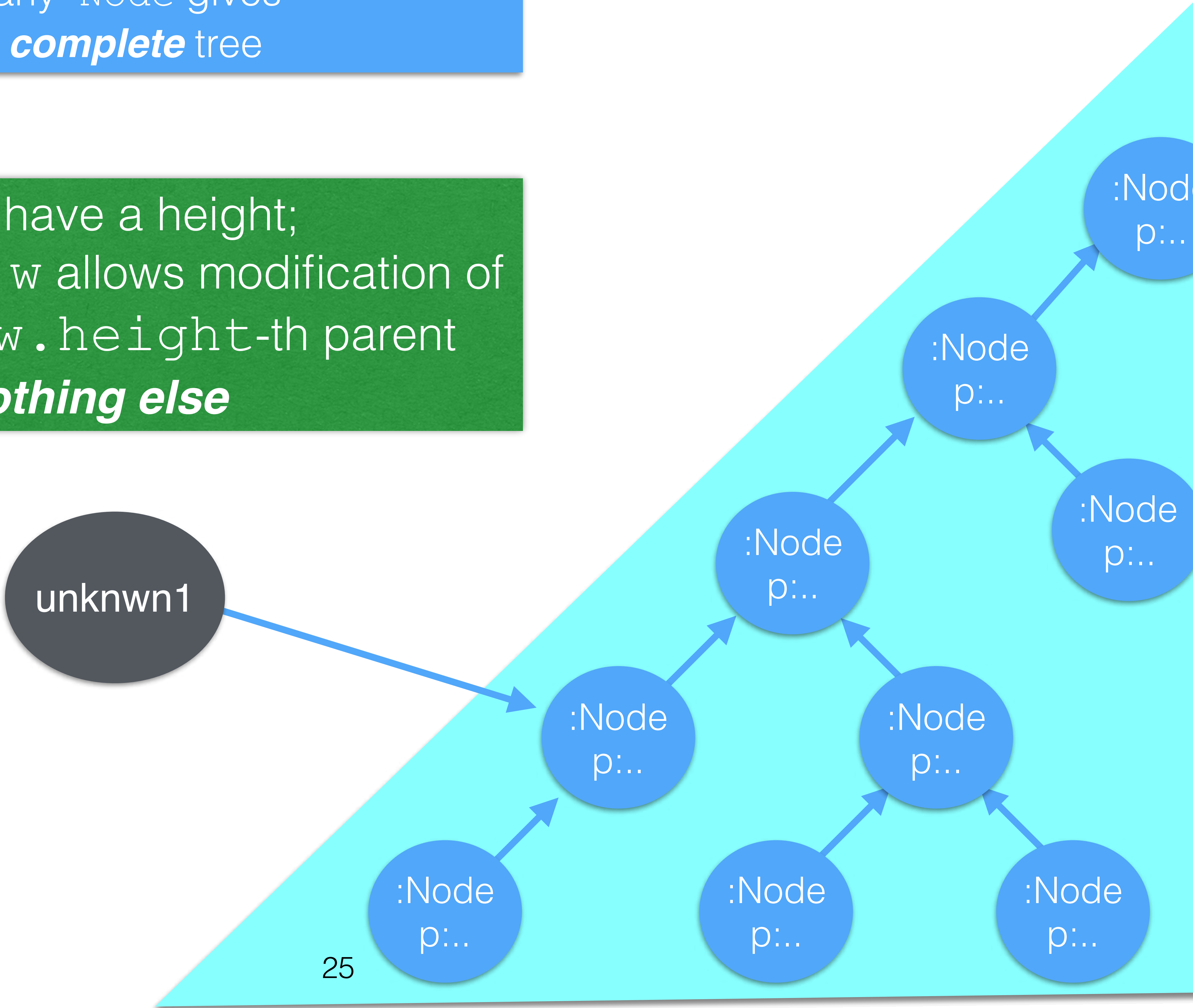
Access to any `Node` gives access to ***complete*** tree



Example 3: DOM attenuation

Access to any `Node` gives access to ***complete*** tree

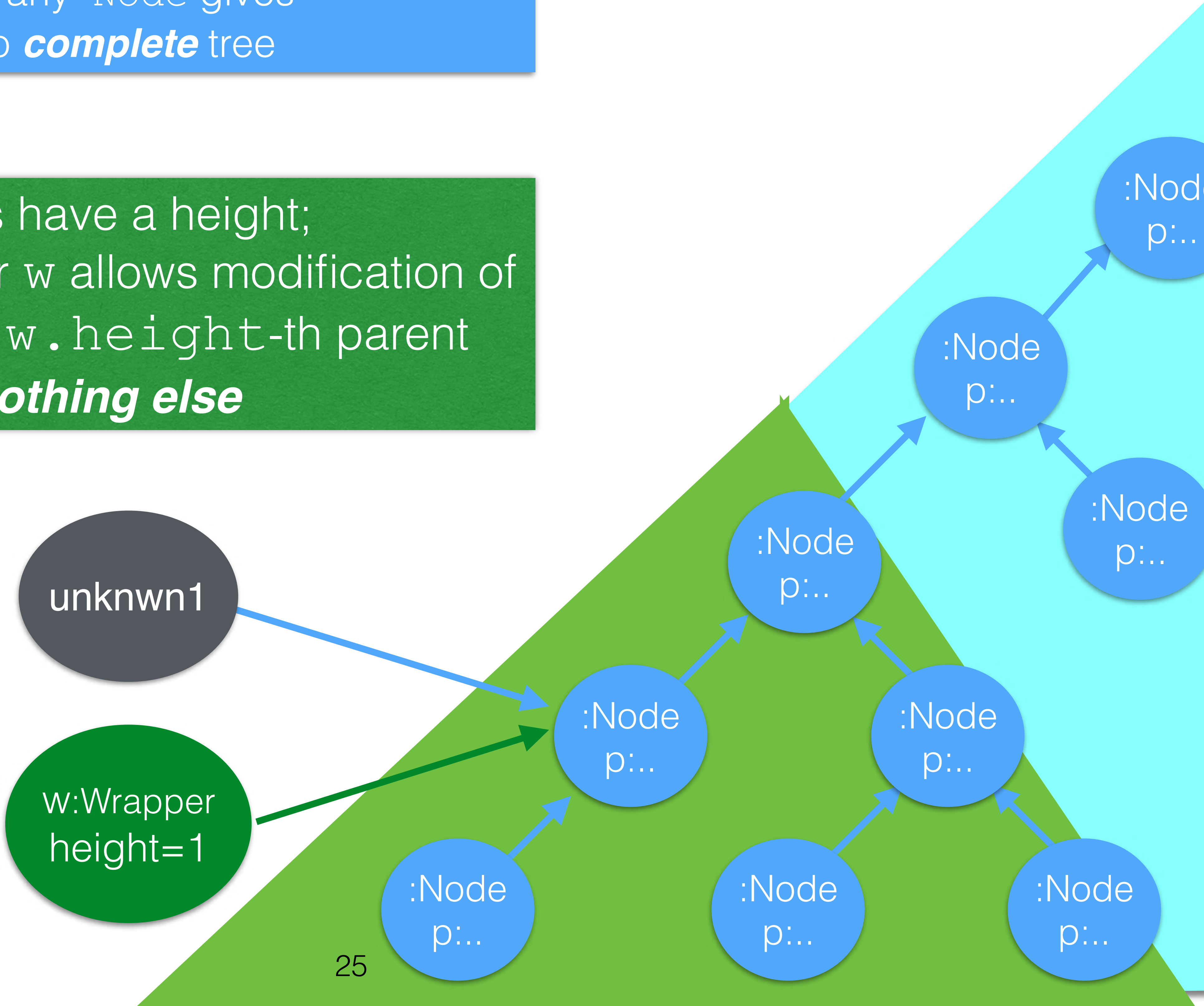
Wrappers have a height;
Access to Wrapper `w` allows modification of `Node`s under the `w.height`-th parent
and nothing else



Example 3: DOM attenuation

Access to any `Node` gives access to ***complete*** tree

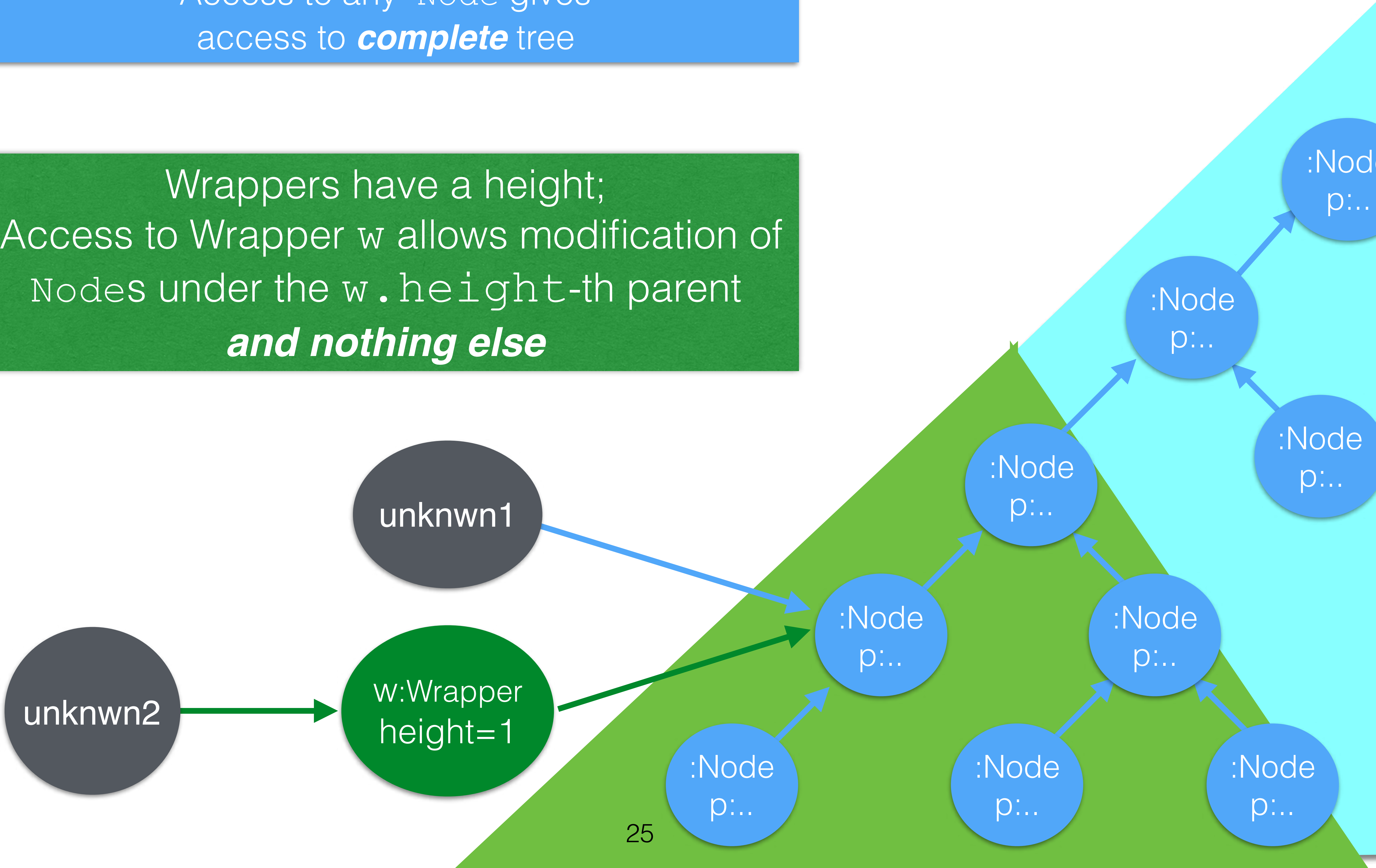
Wrappers have a height;
Access to Wrapper `w` allows modification of `Node`s under the `w.height`-th parent
and nothing else



Example 3: DOM attenuation

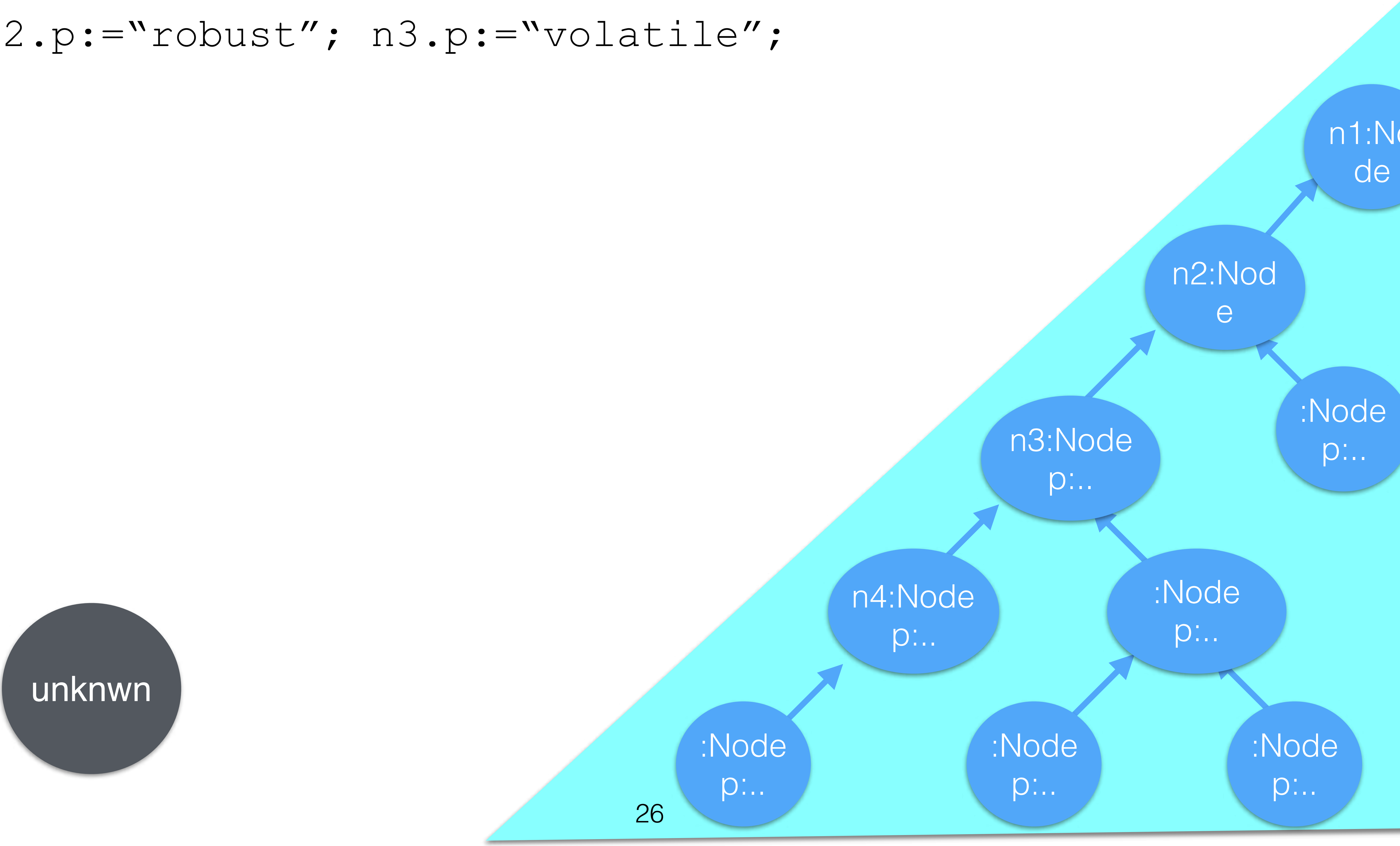
Access to any `Node` gives
access to ***complete*** tree

Wrappers have a height;
Access to Wrapper `w` allows modification of
`Node`s under the `w.height`-th parent
and nothing else



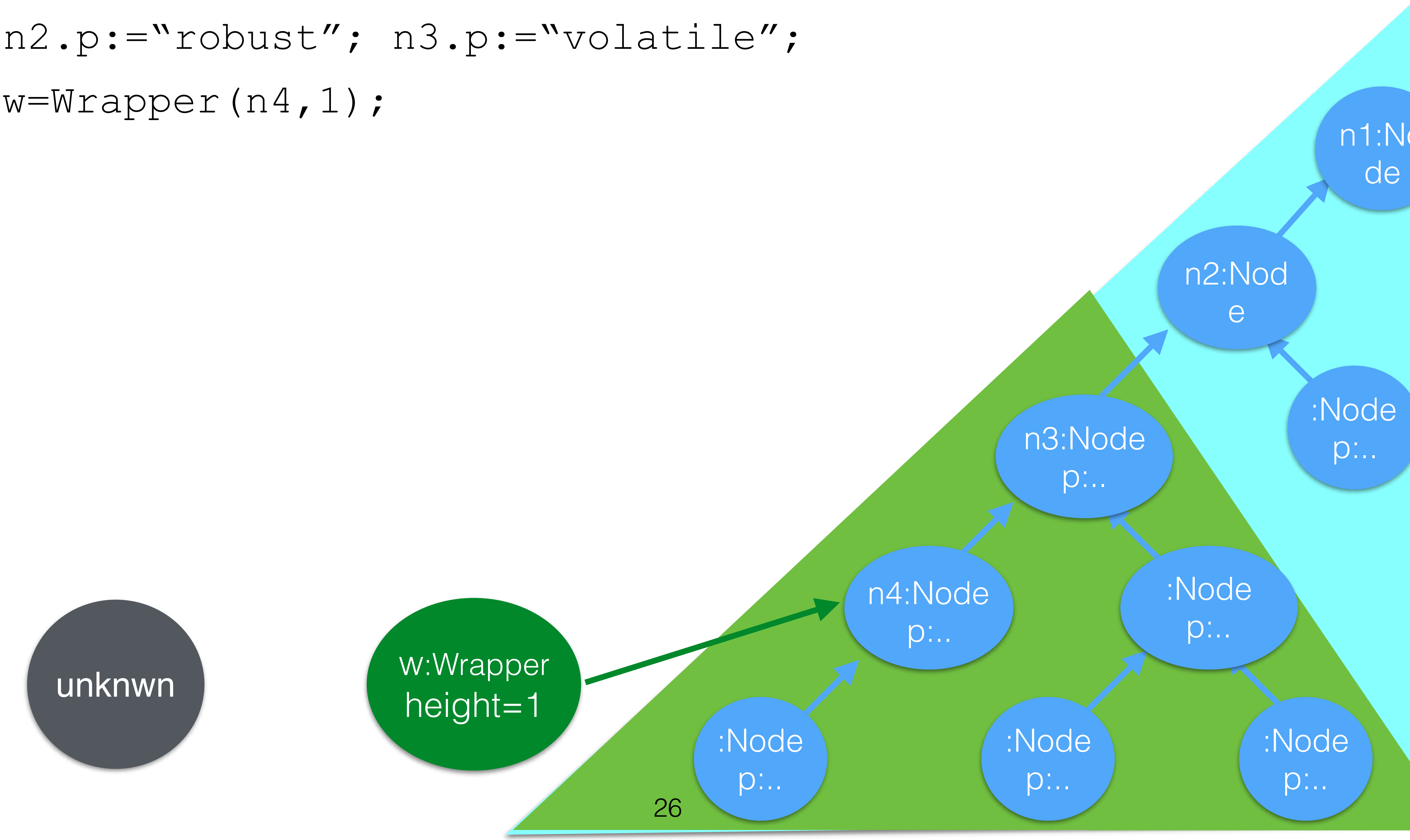
DOM attenuation

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
```



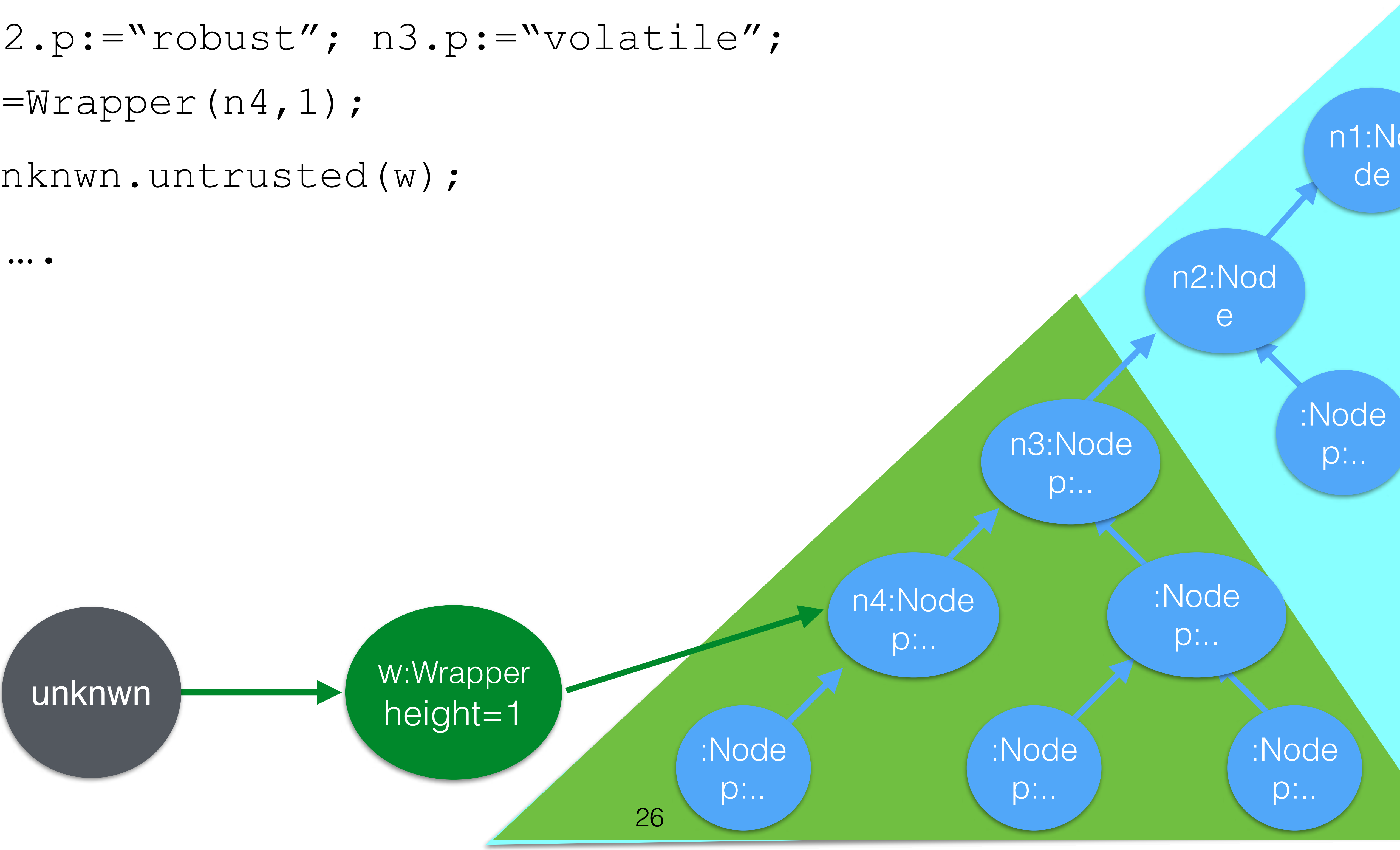
DOM attenuation

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
```



DOM attenuation

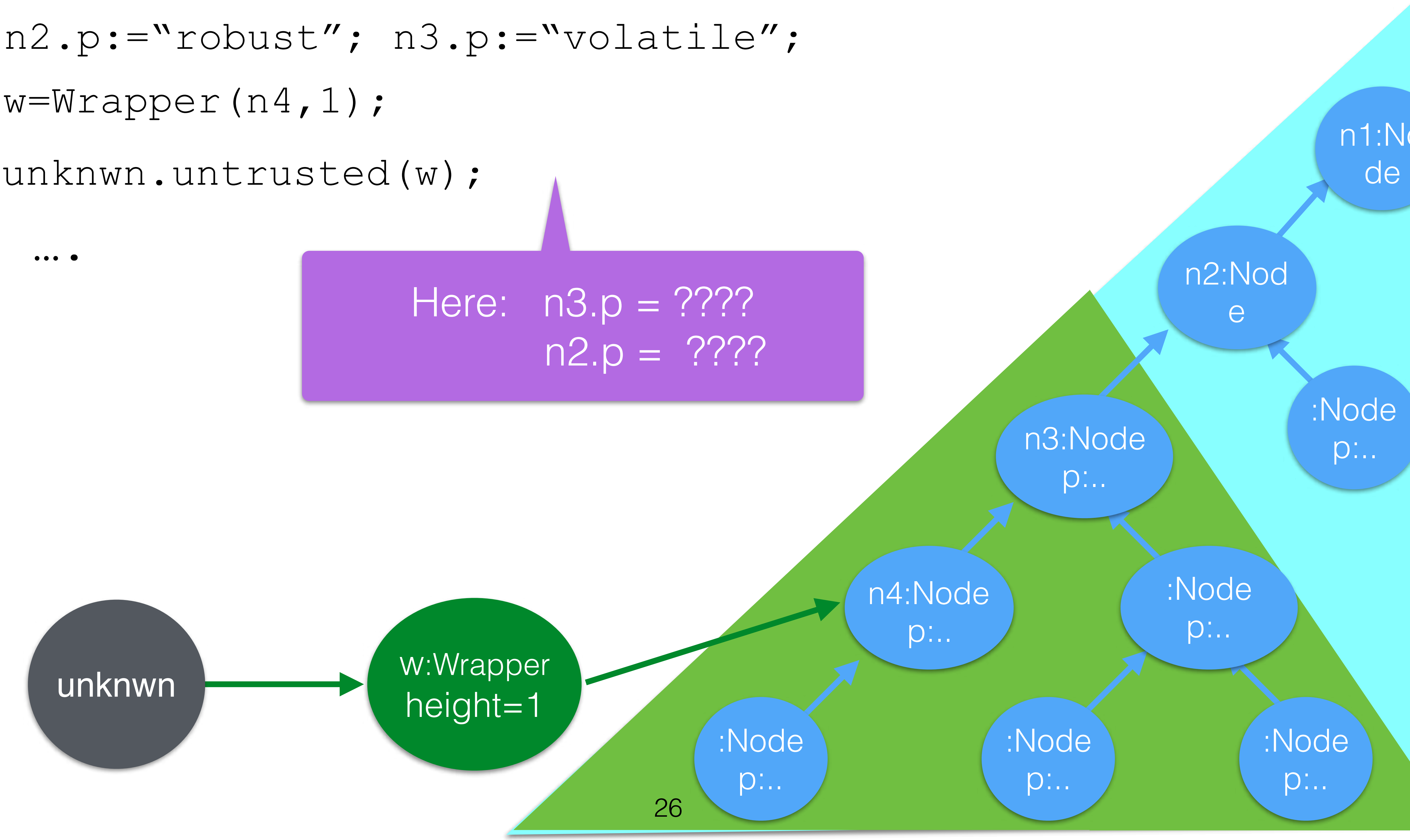
```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```



DOM attenuation

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

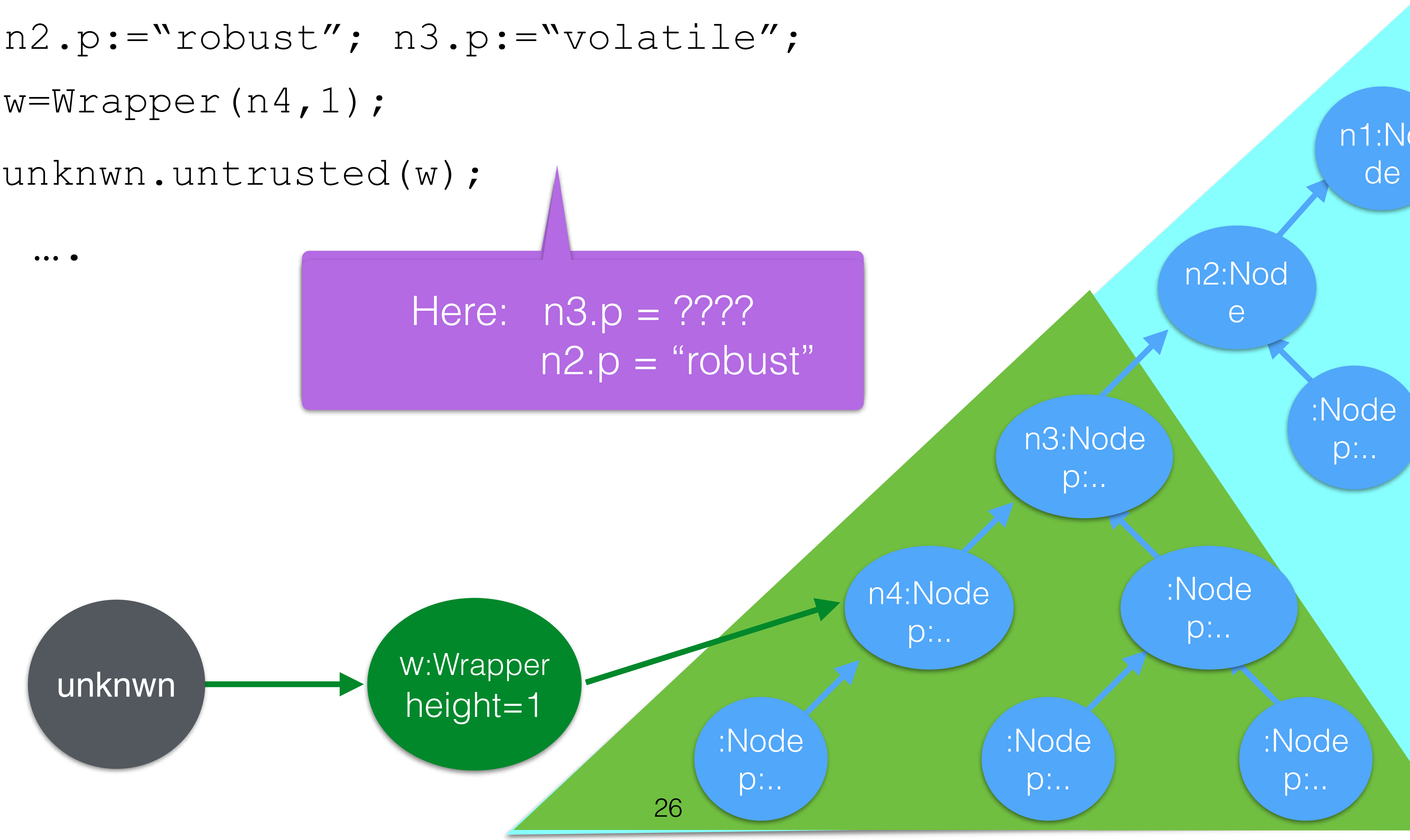
Here: n3.p = ????
n2.p = ????



DOM attenuation

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

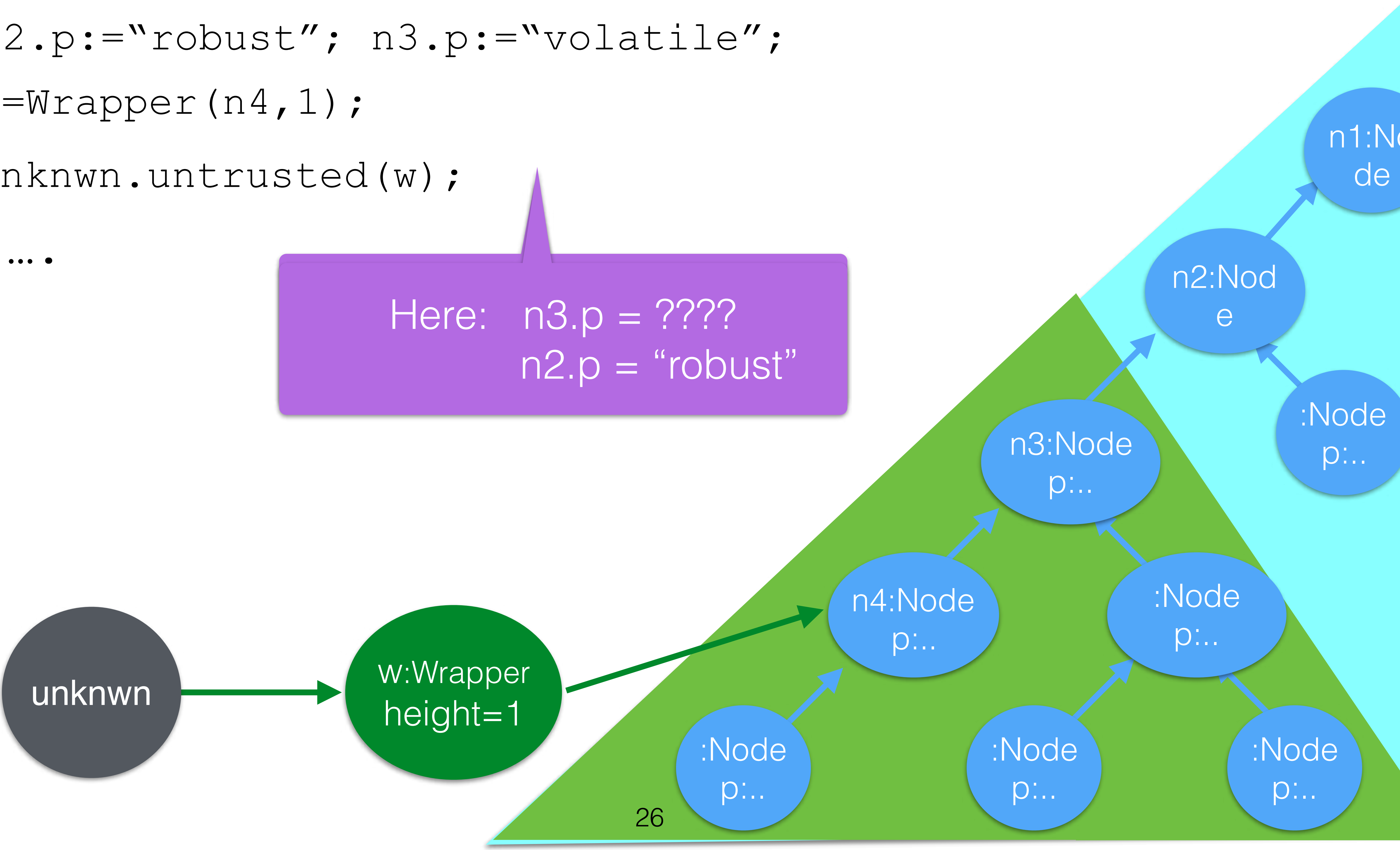
Here: n3.p = ????
n2.p = “robust”



DOM attenuation

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

Here: n3.p = ????
n2.p = “robust”

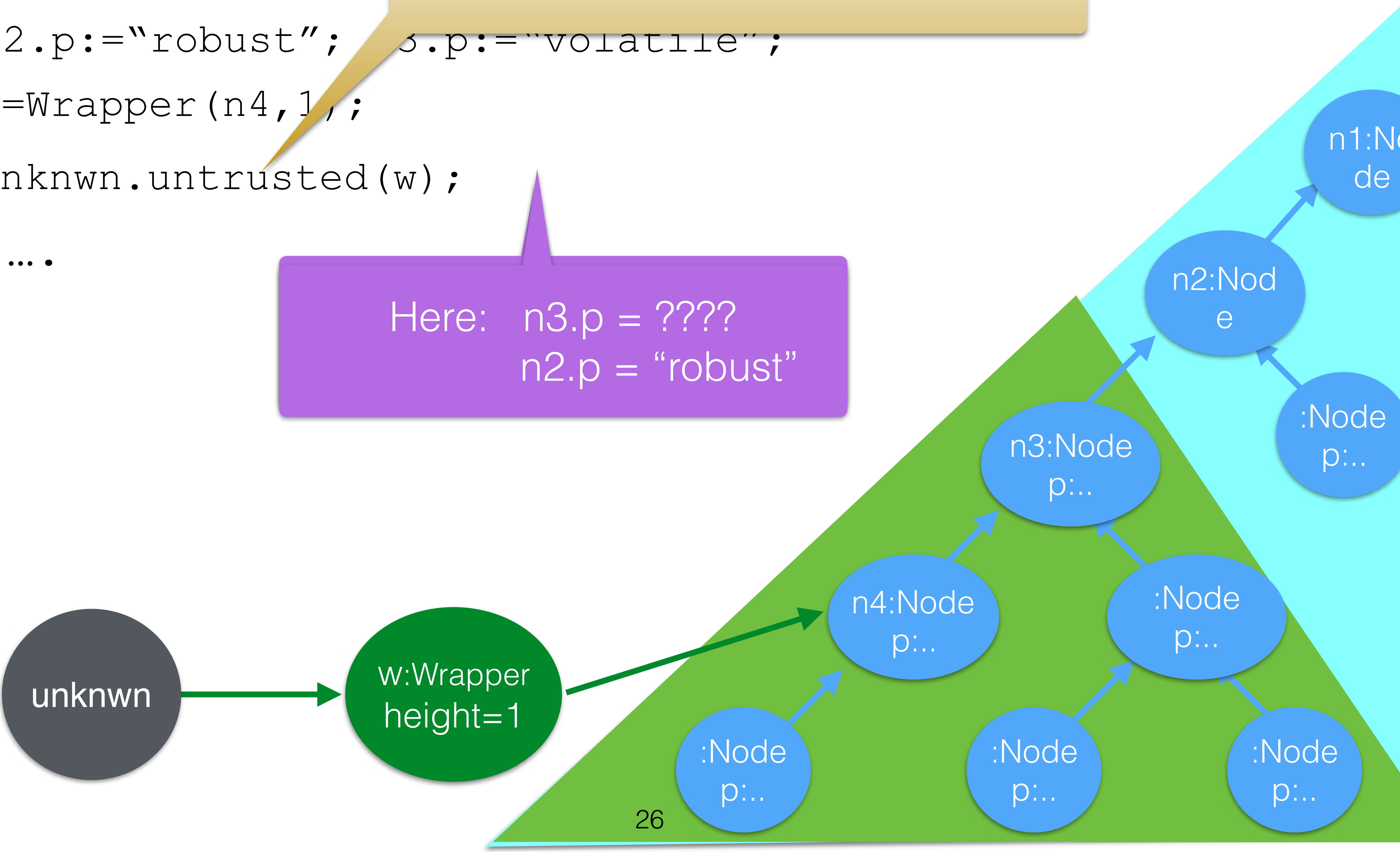


DOM attenuation

```
function mm(unknwn)
  n1:=Node(…); n2:=                                :=Node(n3,…);
  n2.p:=“robust”;  3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

open world

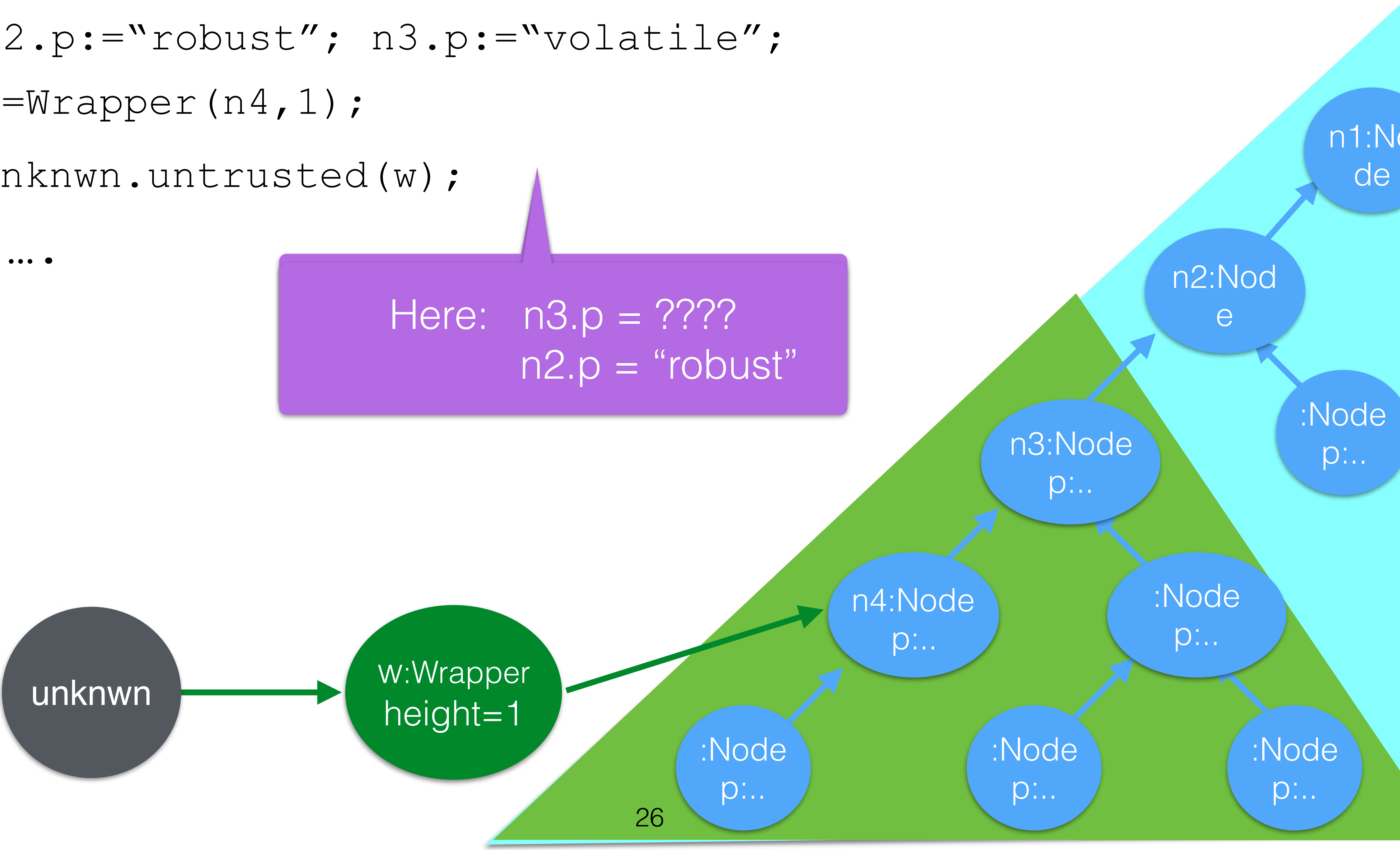
Here: n3.p = ????
       n2.p = “robust”



DOM attenuation

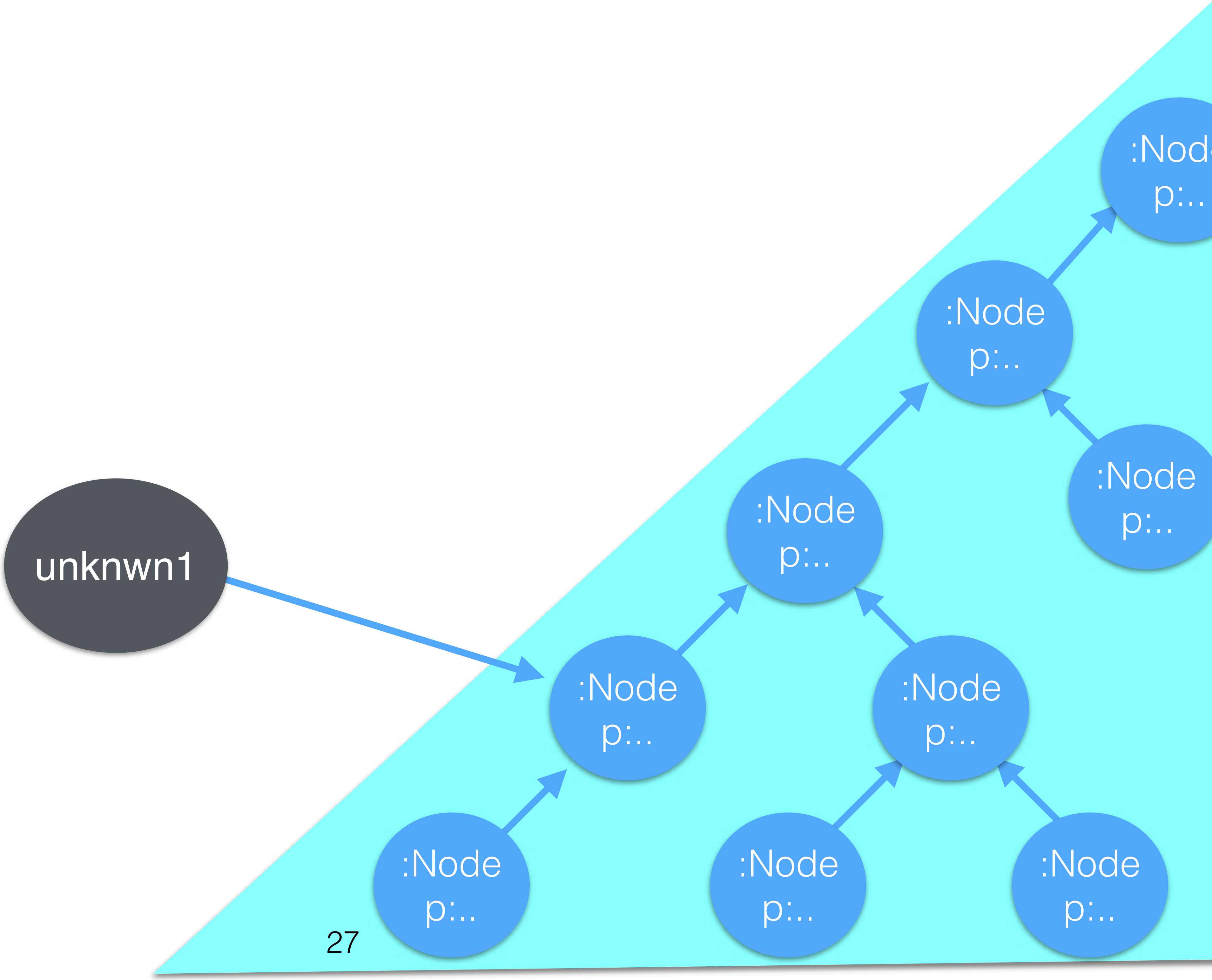
```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

Here: n3.p = ????
n2.p = “robust”



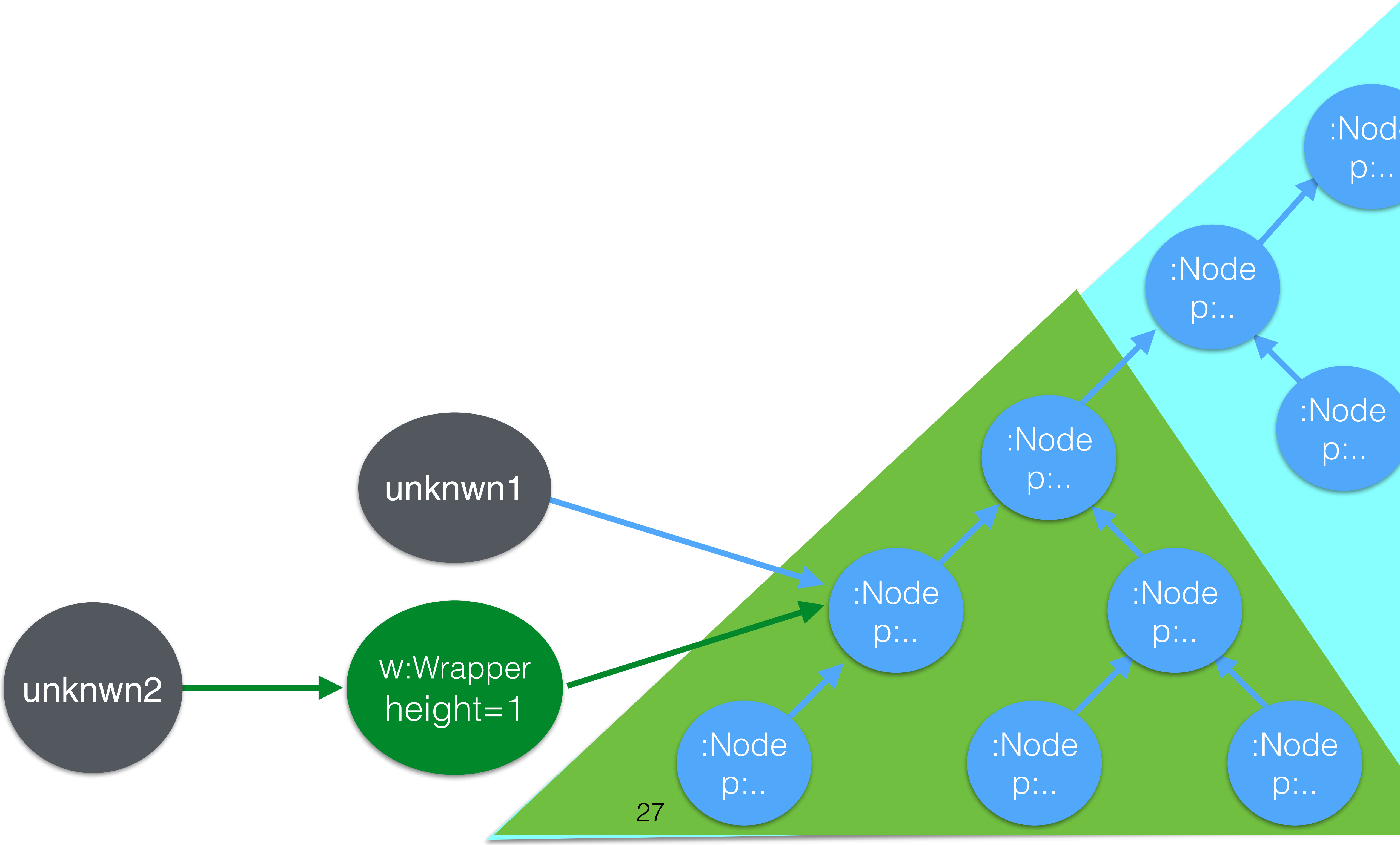
classical

Access to Wrapper `w` allows modification of `Node`s under the `w.height`-th parent and nothing else



classical

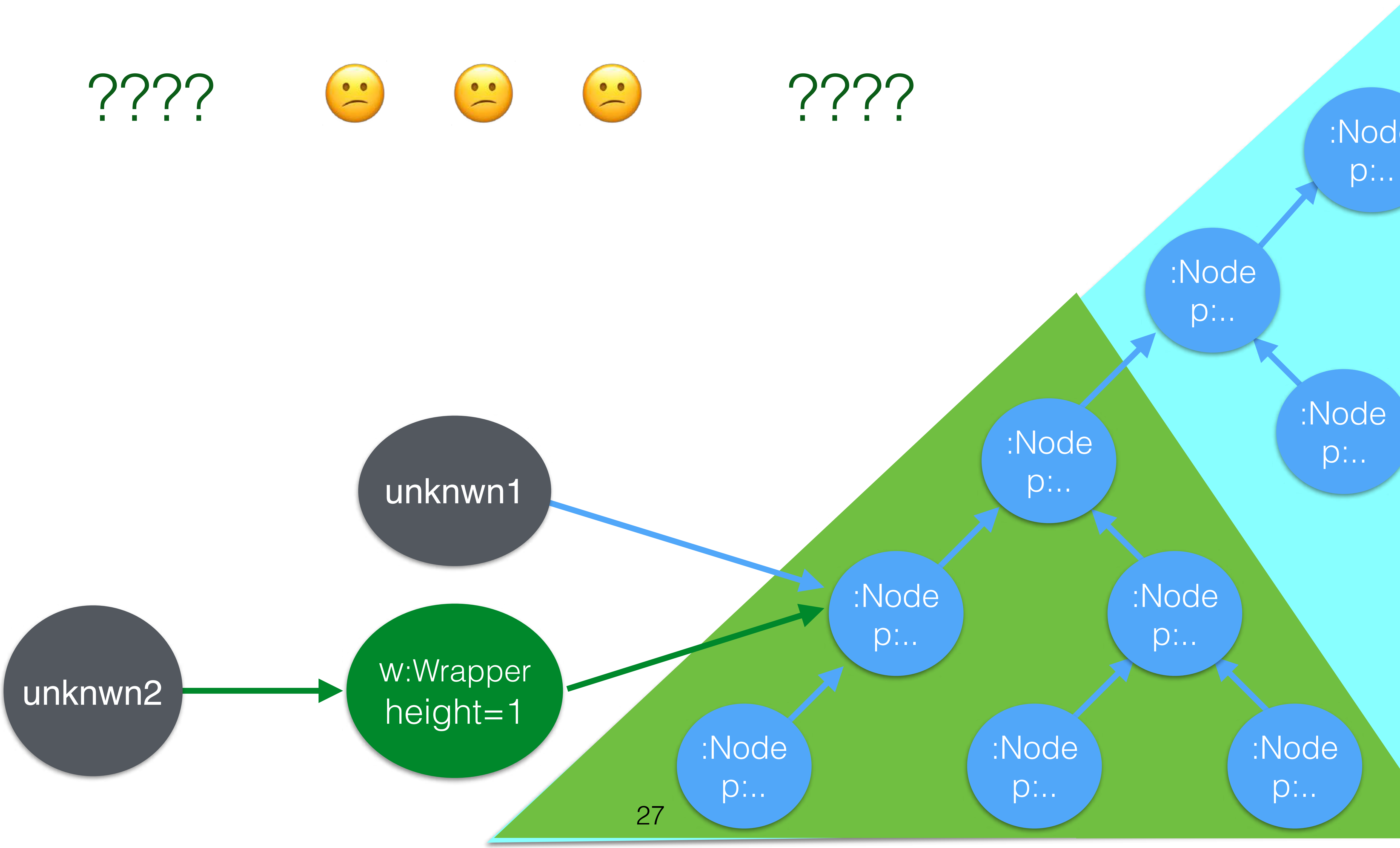
Access to Wrapper `w` allows modification of `Node`s under the `w.height`-th parent and nothing else



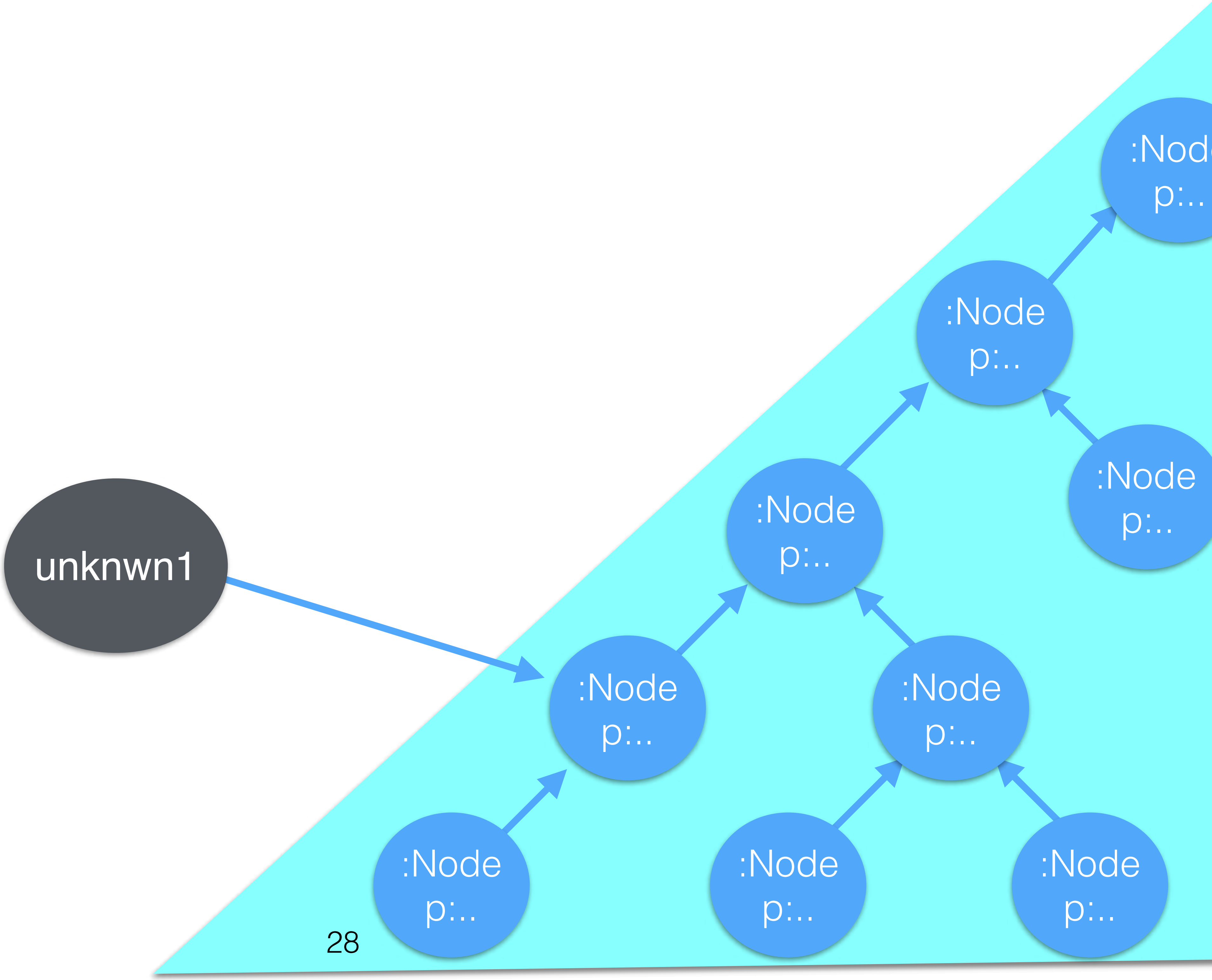
classical

Access to Wrapper `w` allows modification of `Node`s under the `w.height`-th parent and nothing else

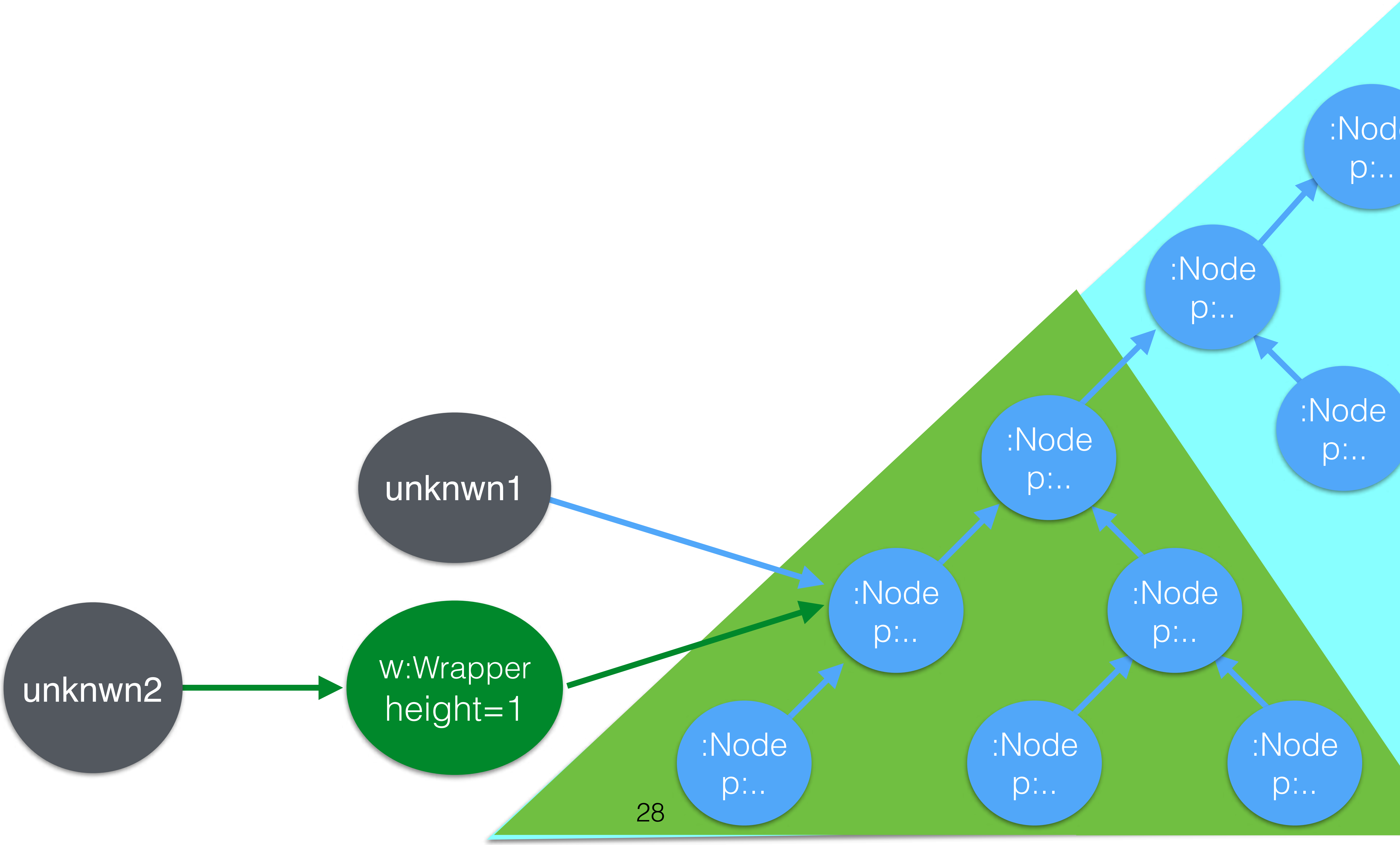
???? 😕 😕 😕 ????



holistic


unknwn1
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..

holistic


:Nod
p:..
:Node
p:..
:Node
p:..
:Node
p:..
unknwn1
:Node
p:..
:Node
p:..
unknwn2
w:Wrapper
height=1
:Node
p:..
:Node
p:..
:Node
p:..

holistic

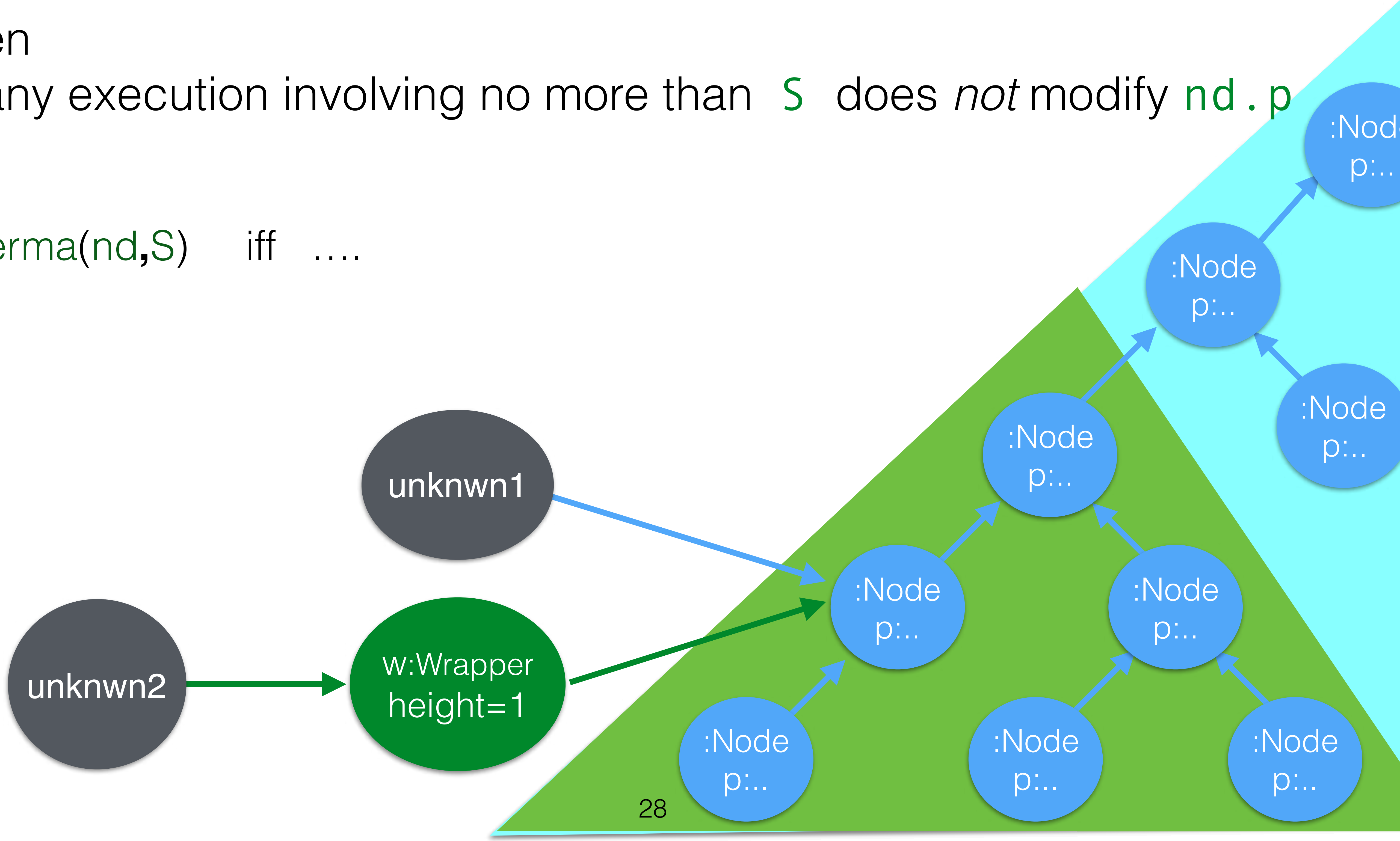
If

a node `nd` is external to a set `S`

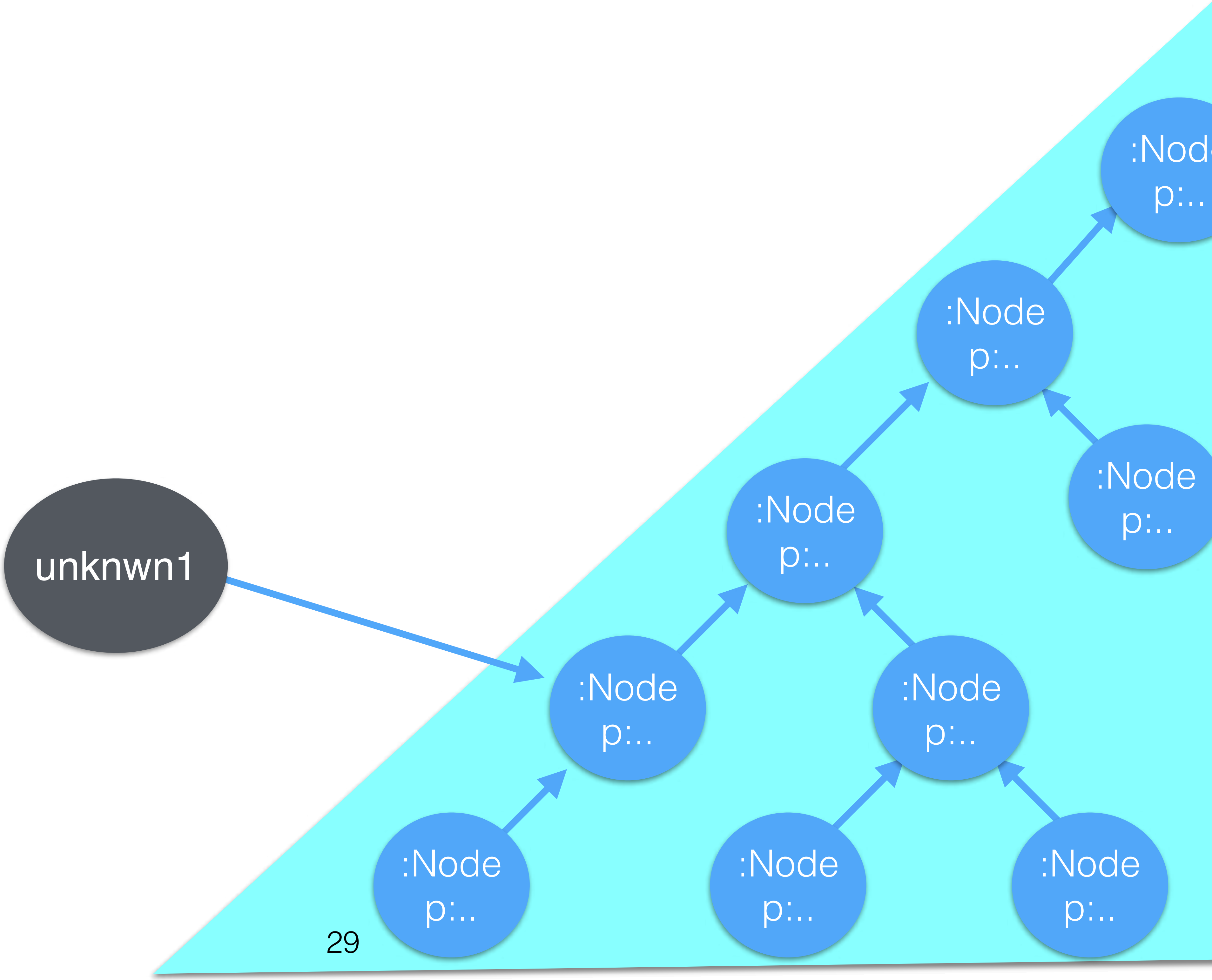
then

any execution involving no more than `S` does *not* modify `nd.p`

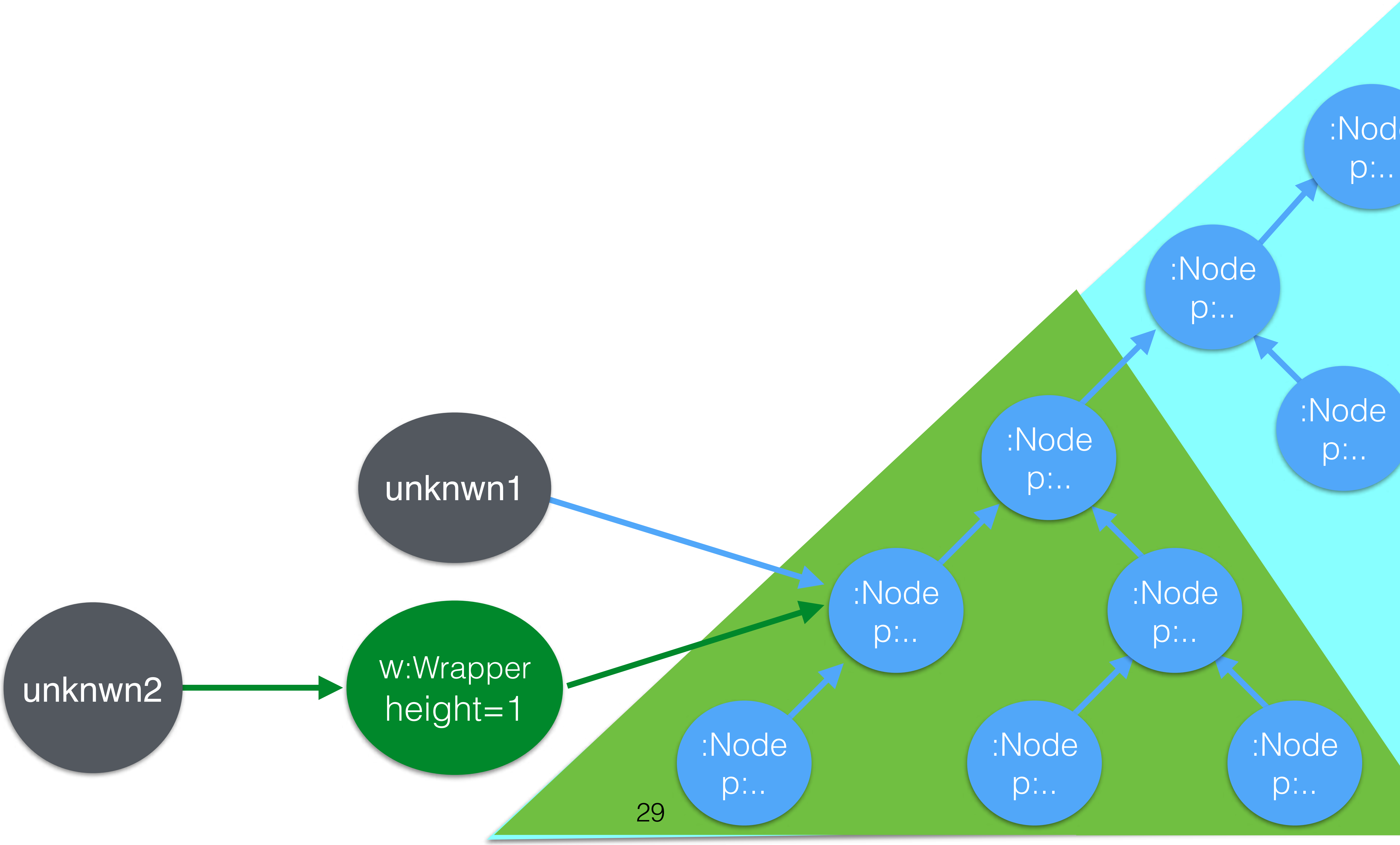
Exterma(nd**,**S) iff ….



holistic


unknwn1
:Node p:..
:Node p:..
:Node p:..
:Node p:..
:Node p:..
:Node p:..
:Node p:..
:Node p:..
:Node p:..

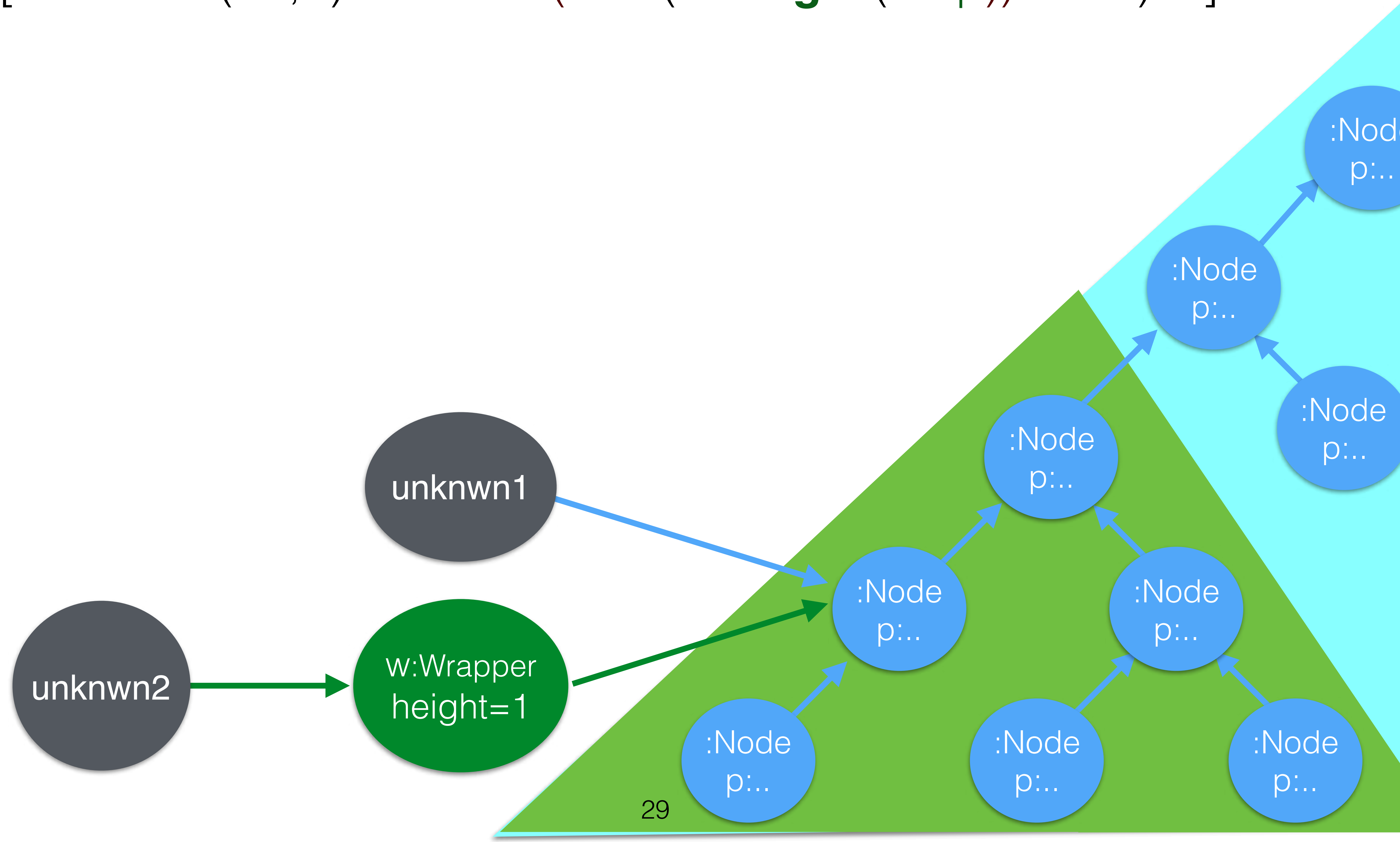
holistic


:Nod
p:..
:Node
p:..
:Node
p:..
:Node
p:..
unknwn1
:Node
p:..
:Node
p:..
unknwn2
w:Wrapper
height=1
:Node
p:..
:Node
p:..
:Node
p:..

holistic

∀S:Set. ∀nd:Node.

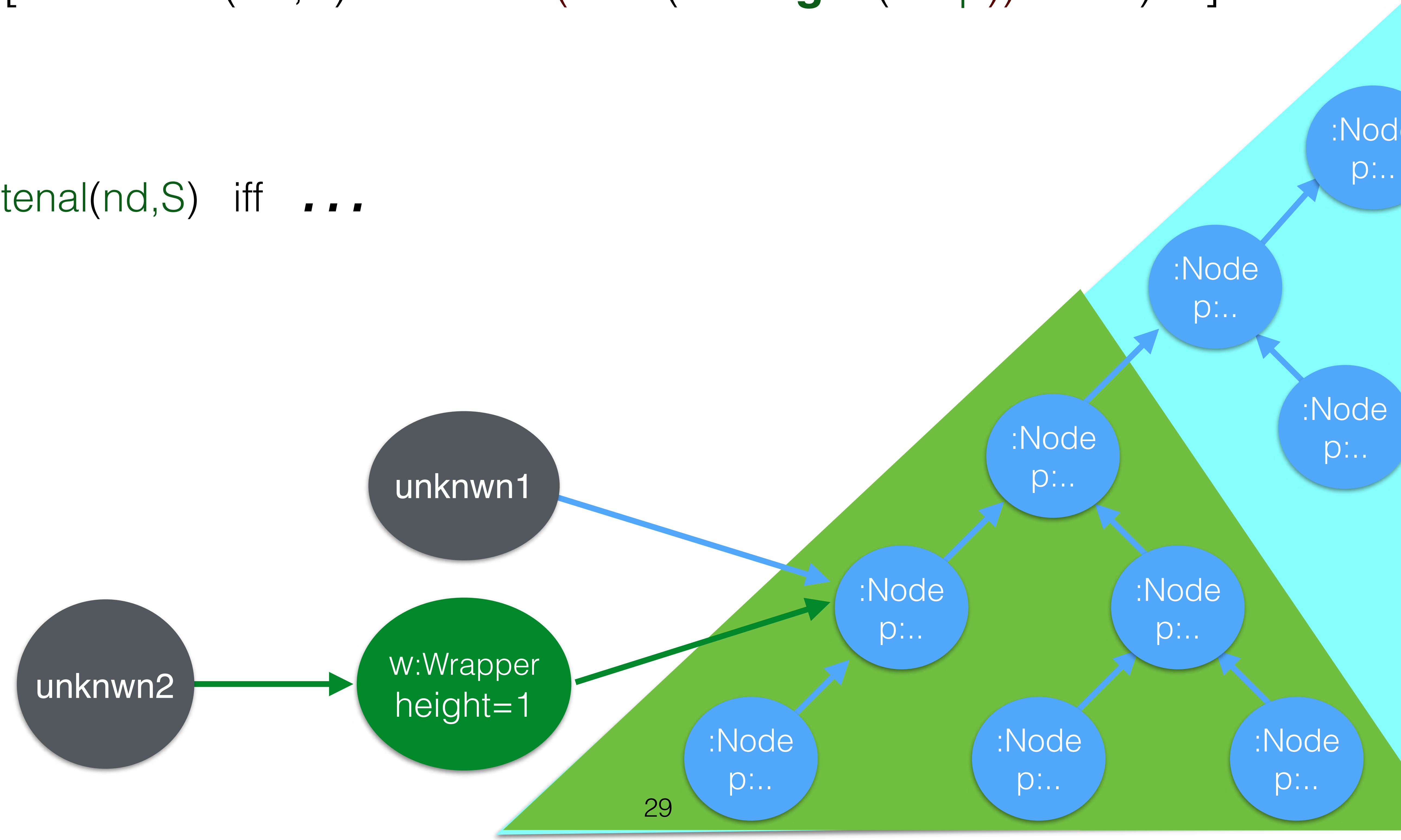
[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]



holistic

∀S:Set. ∀nd:Node.
[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]

Extenal(nd,S) iff **. . .**



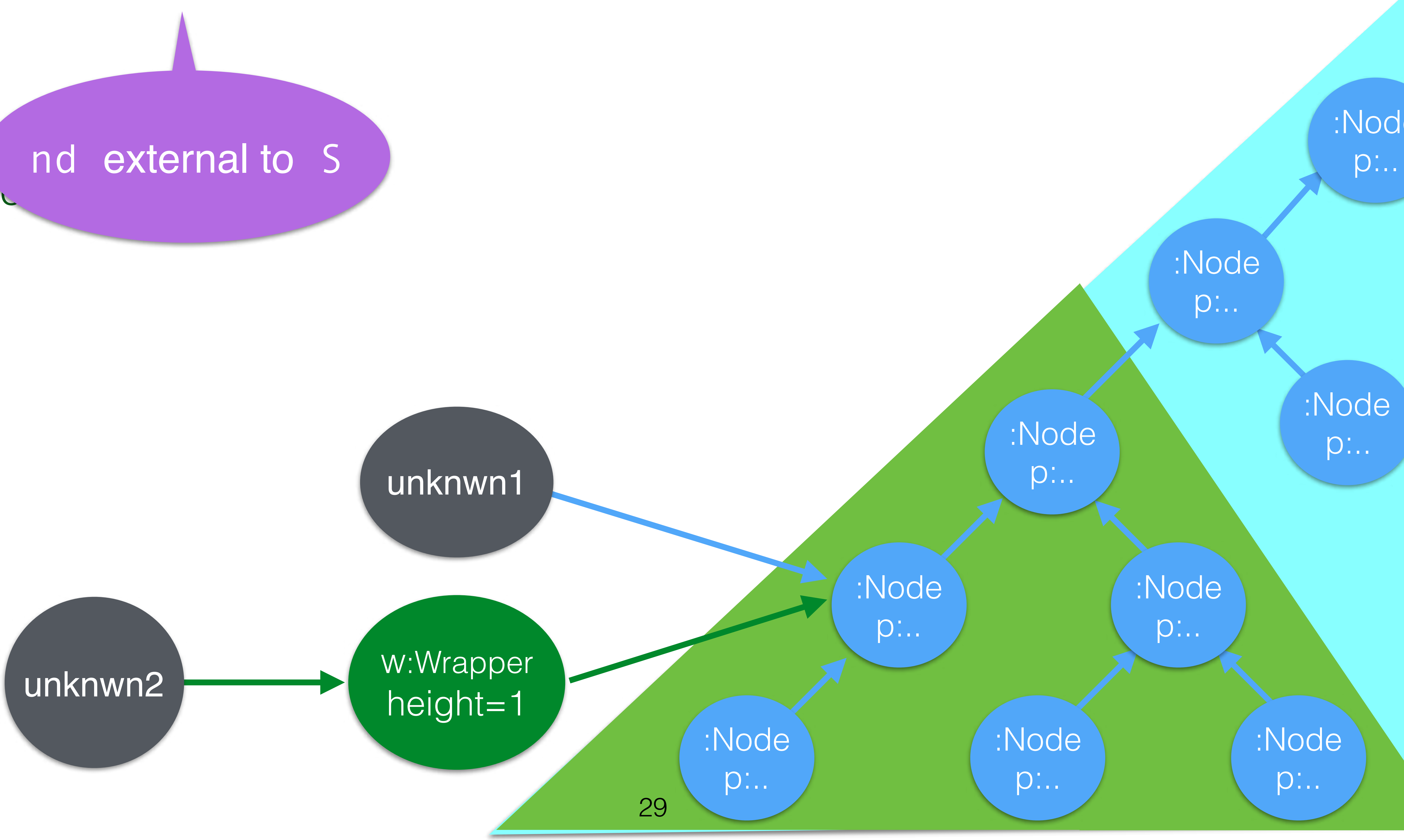
holistic

$\forall$S:Set. $\forall$nd:Node.

[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]

nd external to S

Ext



holistic

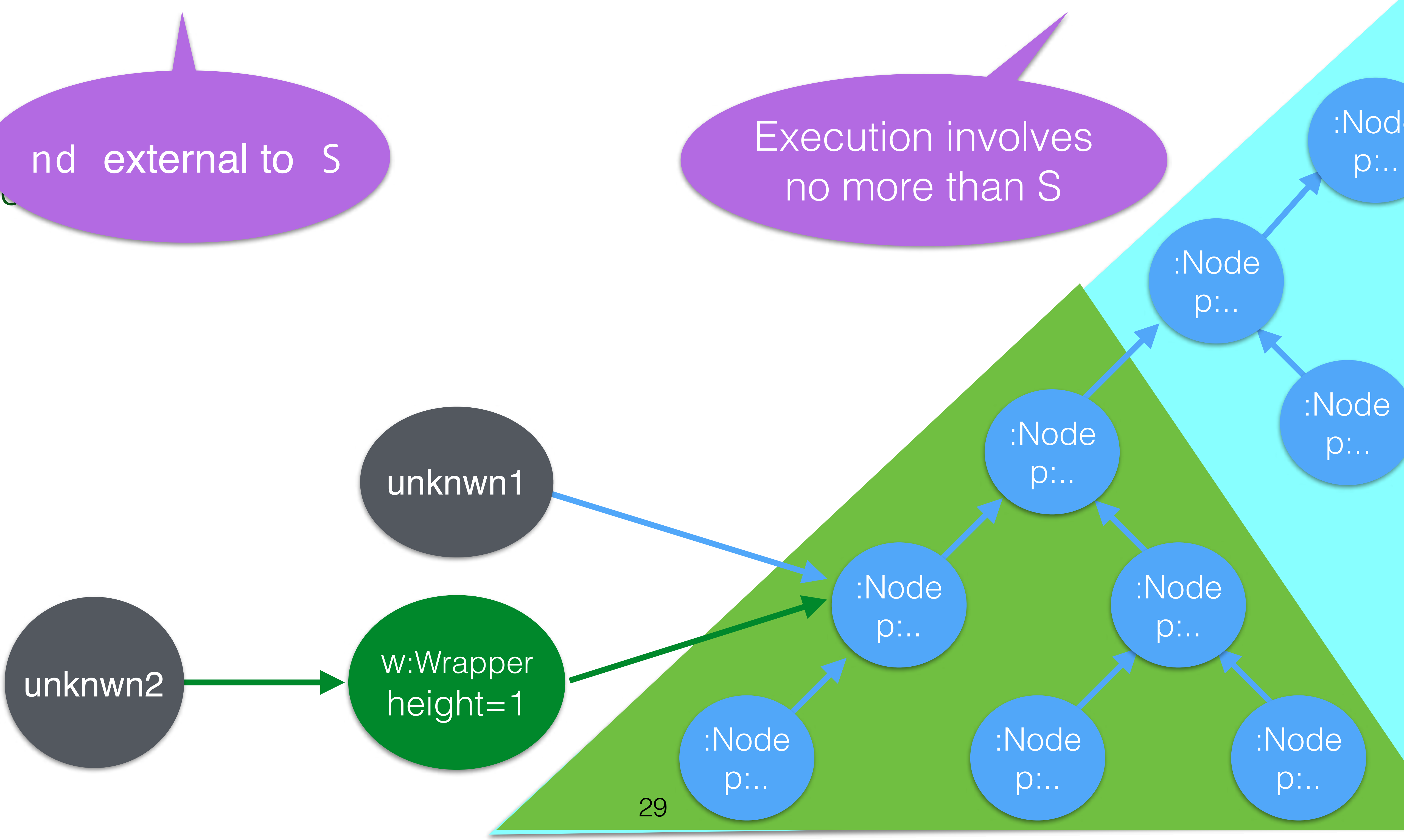
∀S:Set. ∀nd:Node.

[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]

nd external to S

Execution involves no more than S

Ext



holistic

∀S:Set. ∀nd:Node.

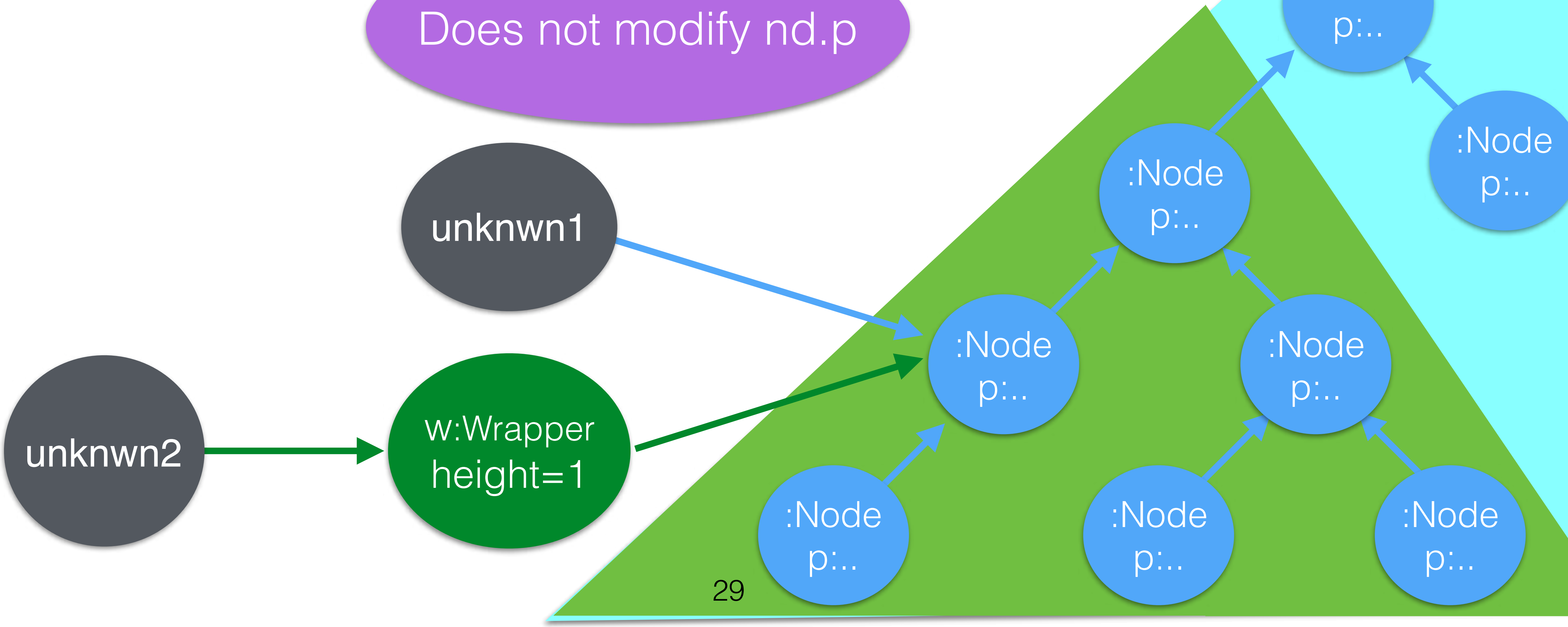
[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]

nd external to S

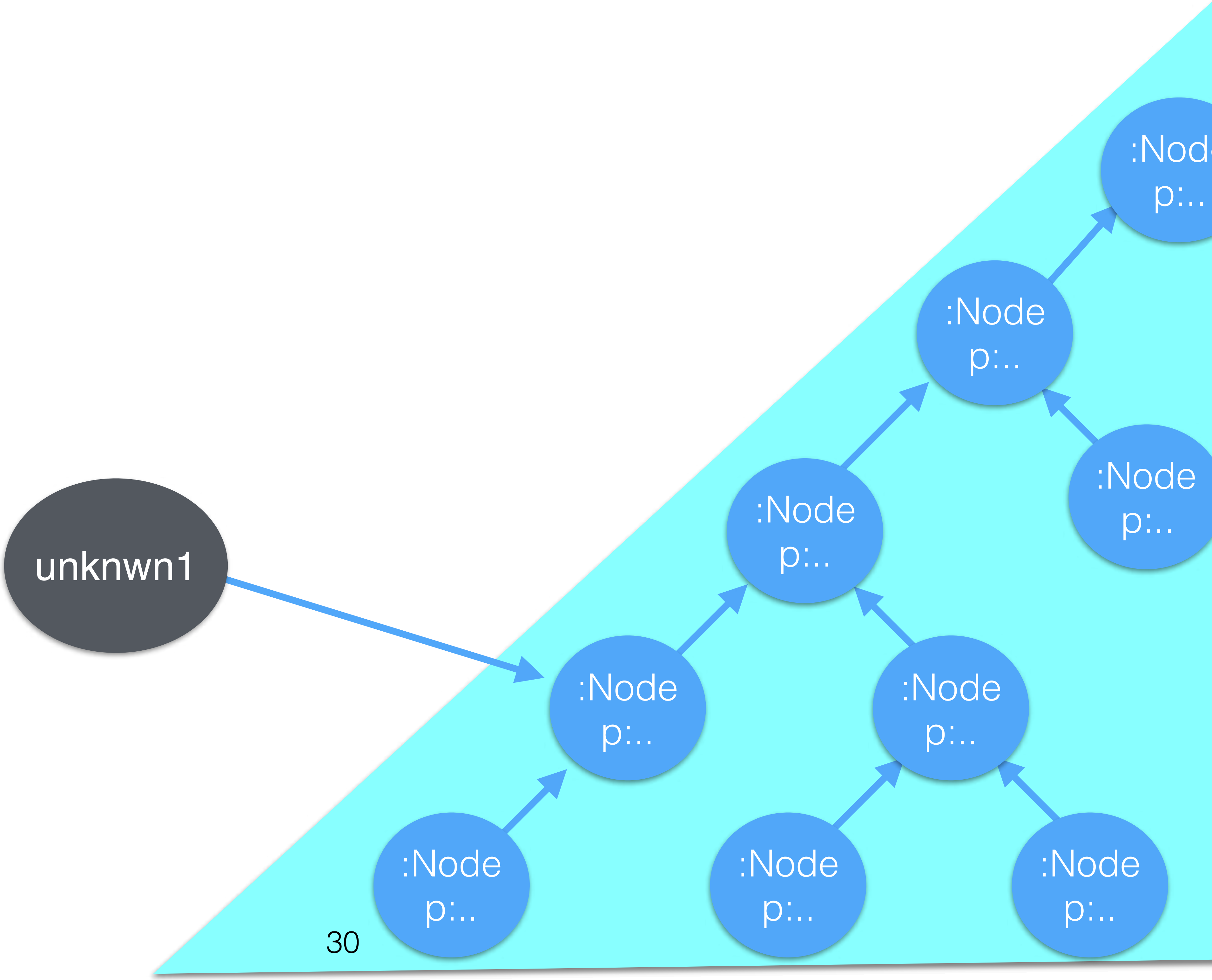
Execution involves no more than S

Ext

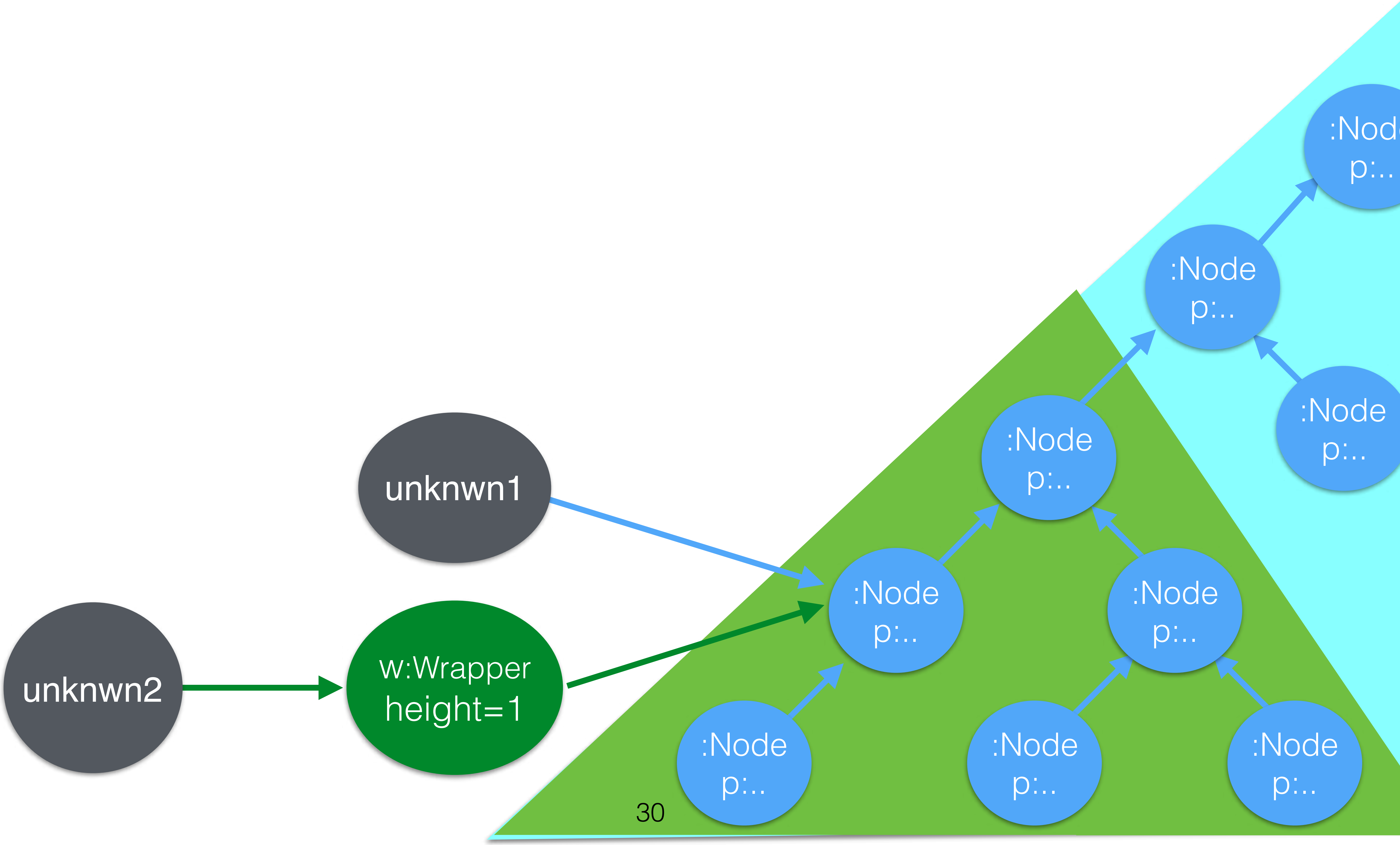
Does not modify nd.p



holistic


unknwn1
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Node
p:..
:Nod
p:..

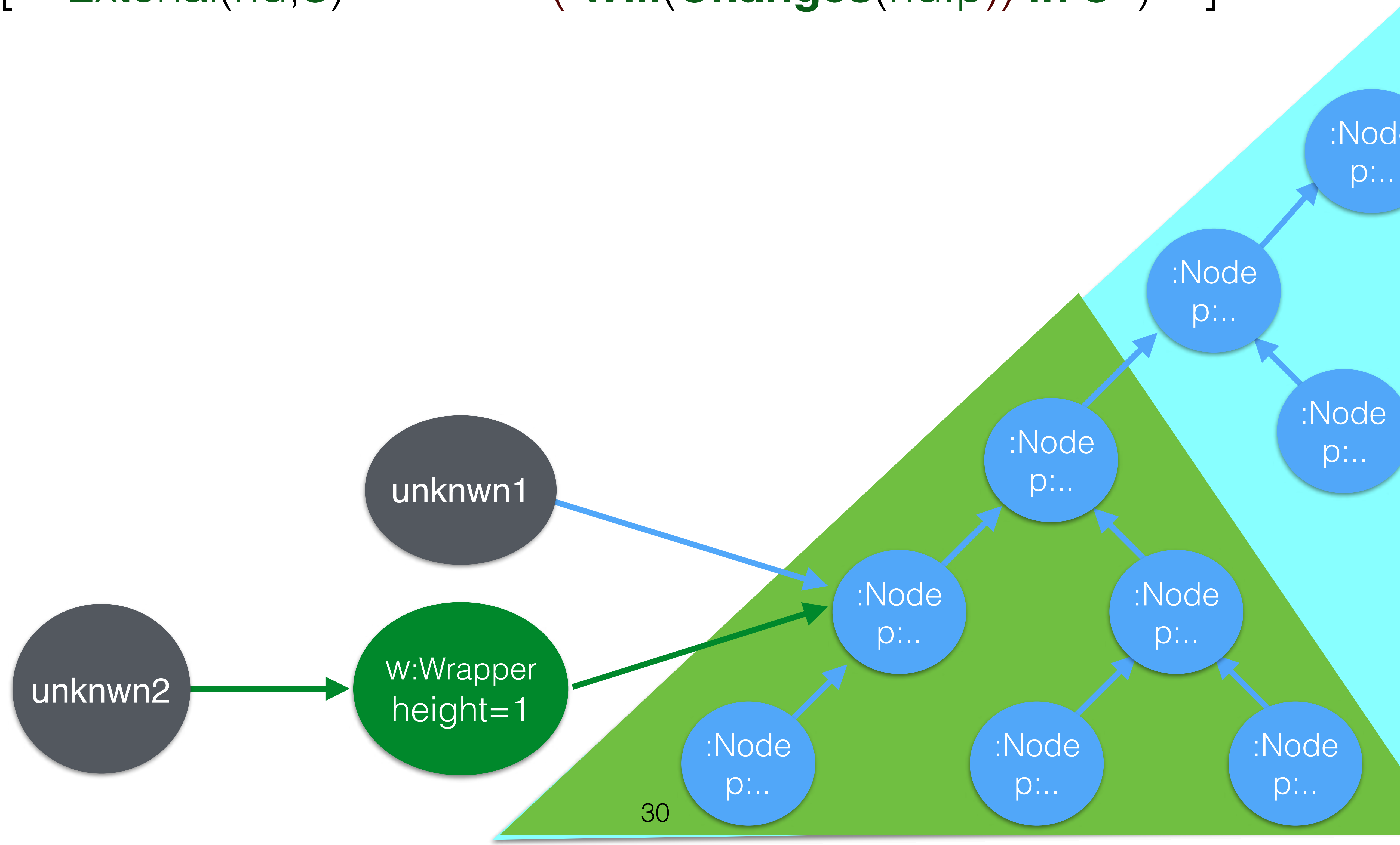
holistic


:Nod
p:..
:Node
p:..
:Node
p:..
:Node
p:..
unknwn1
:Node
p:..
:Node
p:..
unknwn2
w:Wrapper
height=1
:Node
p:..
:Node
p:..
:Node
p:..

holistic

∀S:Set. ∀nd:Node.

[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]

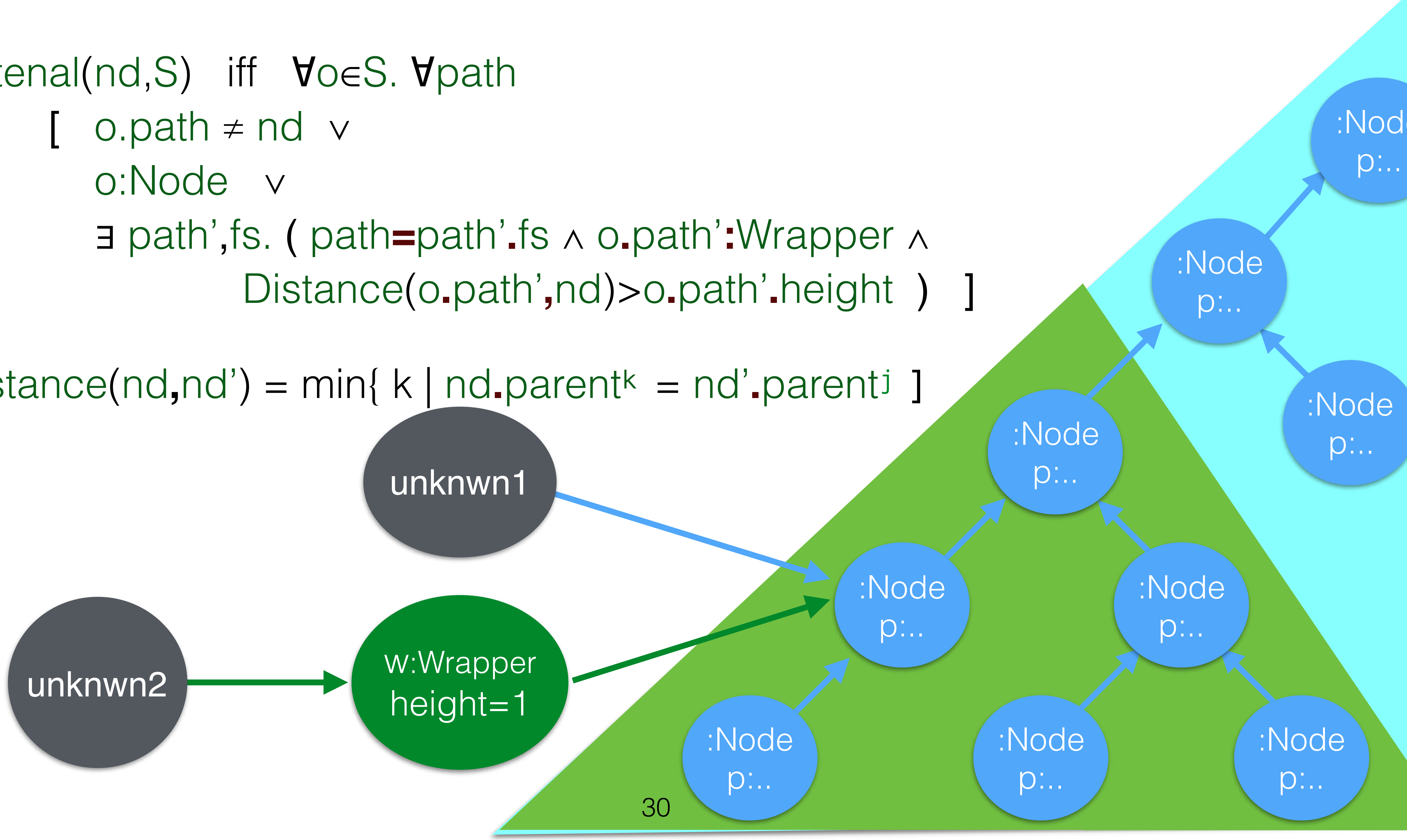


holistic

∀S:Set. ∀nd:Node.
[Extenal(nd,S) → ¬ (**Will**(**Changes**(nd.p)) **in** S)]

Extenal(nd,S) iff ∀o∊S. ∀path
[o.path ≠ nd ∨
o:Node ∨
∃ path’,fs. (path=path’.fs ∧ o.path’:Wrapper ∧
Distance(o.path’,nd)>o.path’.height)]

Distance(nd,nd’) = min{ k | nd.parent^k = nd’.parent^j]



external

Extenal(nd,S)   iff   ∀o∈S. ∀path

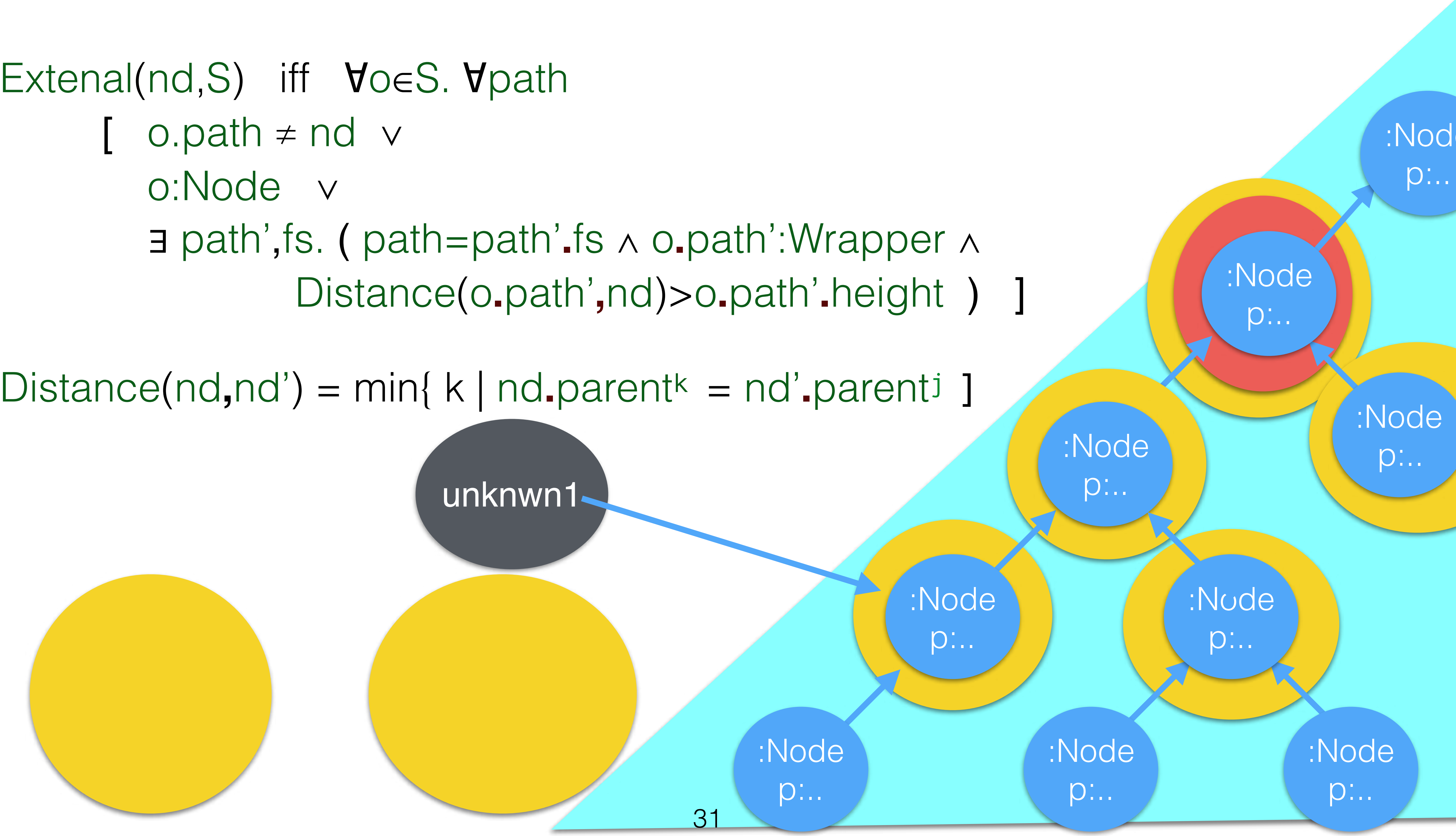
[o.path ≠ nd ∨

o:Node   ∨

∃ path',fs. (path=path'.fs ∧ o.path':Wrapper ∧

Distance(o.path',nd)>o.path'.height)   ]

Distance(nd,nd') = min{ k | nd.parent^k = nd'.parent^j]



external

Extenal(nd,S) iff $\forall o \in S.\ \forall path$

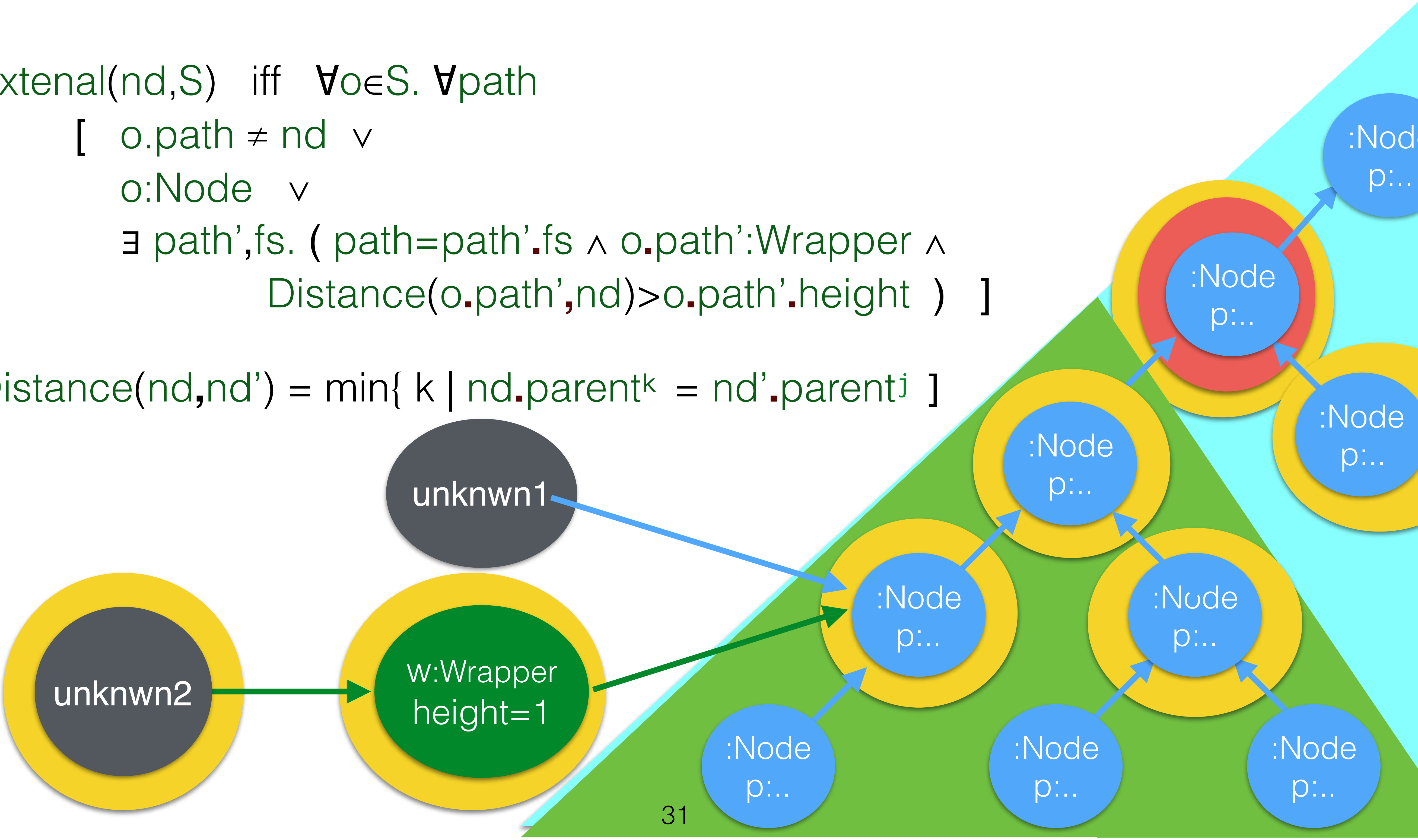
$[\ o.path \neq nd\ \vee$

$o{:}Node\ \vee$

$\exists\ path',fs.\ (\ path=path'.fs \wedge o.path'{:}Wrapper \wedge$

$Distance(o.path',nd) > o.path'.height\ )\ ]$

$Distance(nd,nd') = min\{\ k \mid nd.parent^k = nd'.parent^j\ ]$



external

Extenal(**RedNode**, **YellowSet**)

$\text{Extenal}(nd,S) \quad \text{iff} \quad \forall o \in S.\ \forall path$

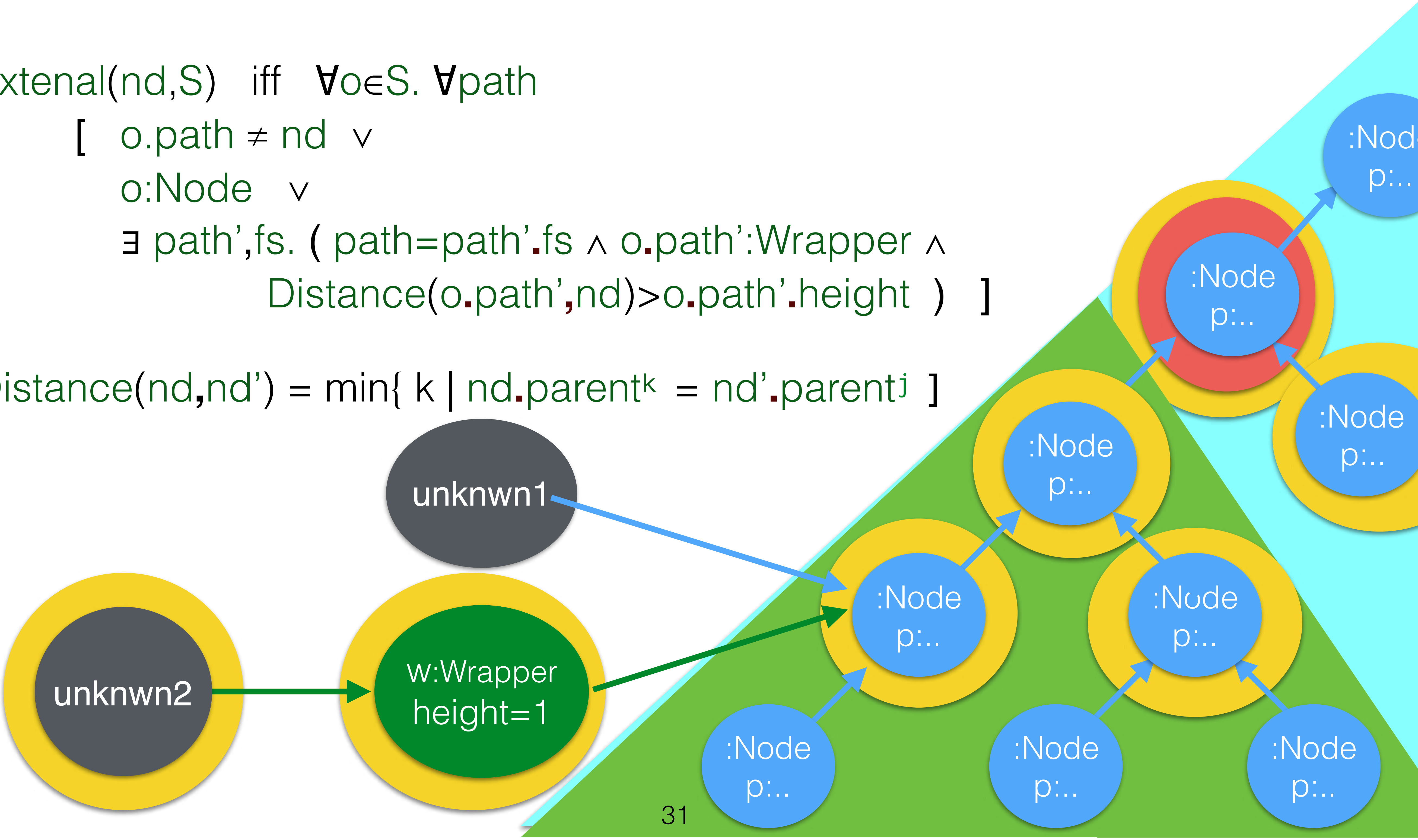
$\quad [\quad o.path \neq nd \quad \vee$

$\quad\quad o{:}\text{Node} \quad \vee$

$\quad\quad \exists\ path',fs.\ (\ path = path'.fs \wedge o.path'{:}\text{Wrapper} \wedge$

$\quad\quad\quad \text{Distance}(o.path',nd) > o.path'.height\ \ )\ \ ]$

$\text{Distance}(nd,nd') = \min\{\ k \mid nd.parent^{k} = nd'.parent^{j}\ \ ]$



external

$\text{Extenal}(nd,S)$ iff $\forall o \in S.\ \forall path$

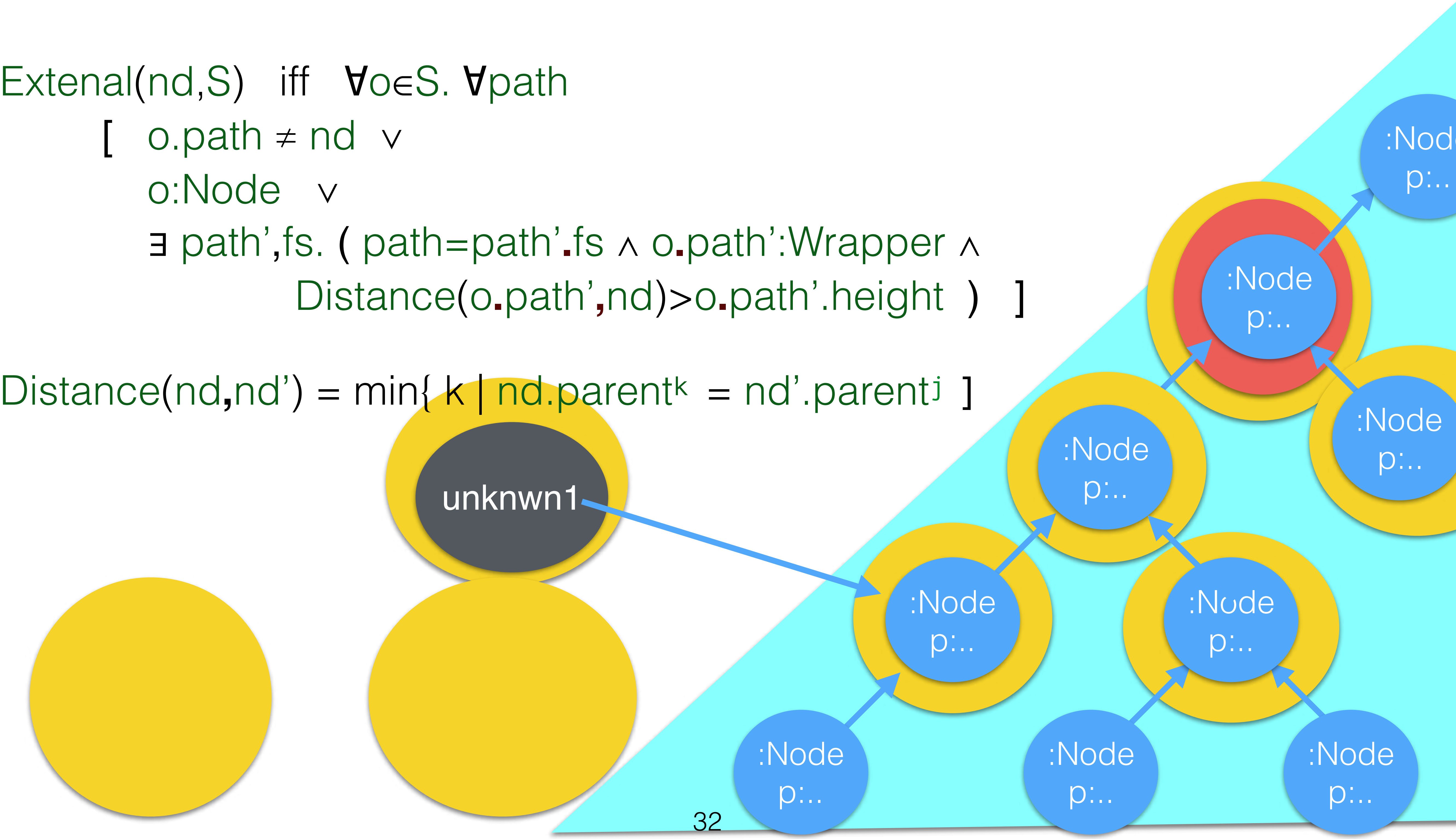
$[\ \ o.path \neq nd\ \ \vee$

$o{:}Node\ \ \vee$

$\exists\ path',fs.\ (\ path=path'.fs \wedge o.path'{:}Wrapper \wedge$

$Distance(o.path',nd) > o.path'.height\ \ )\ \ ]$

$Distance(nd,nd') = \min\{\ k \mid nd.parent^{k}\ = nd'.parent^{j}\ \ ]$



external

Extenal(nd,S)   iff   ∀o∈S. ∀path

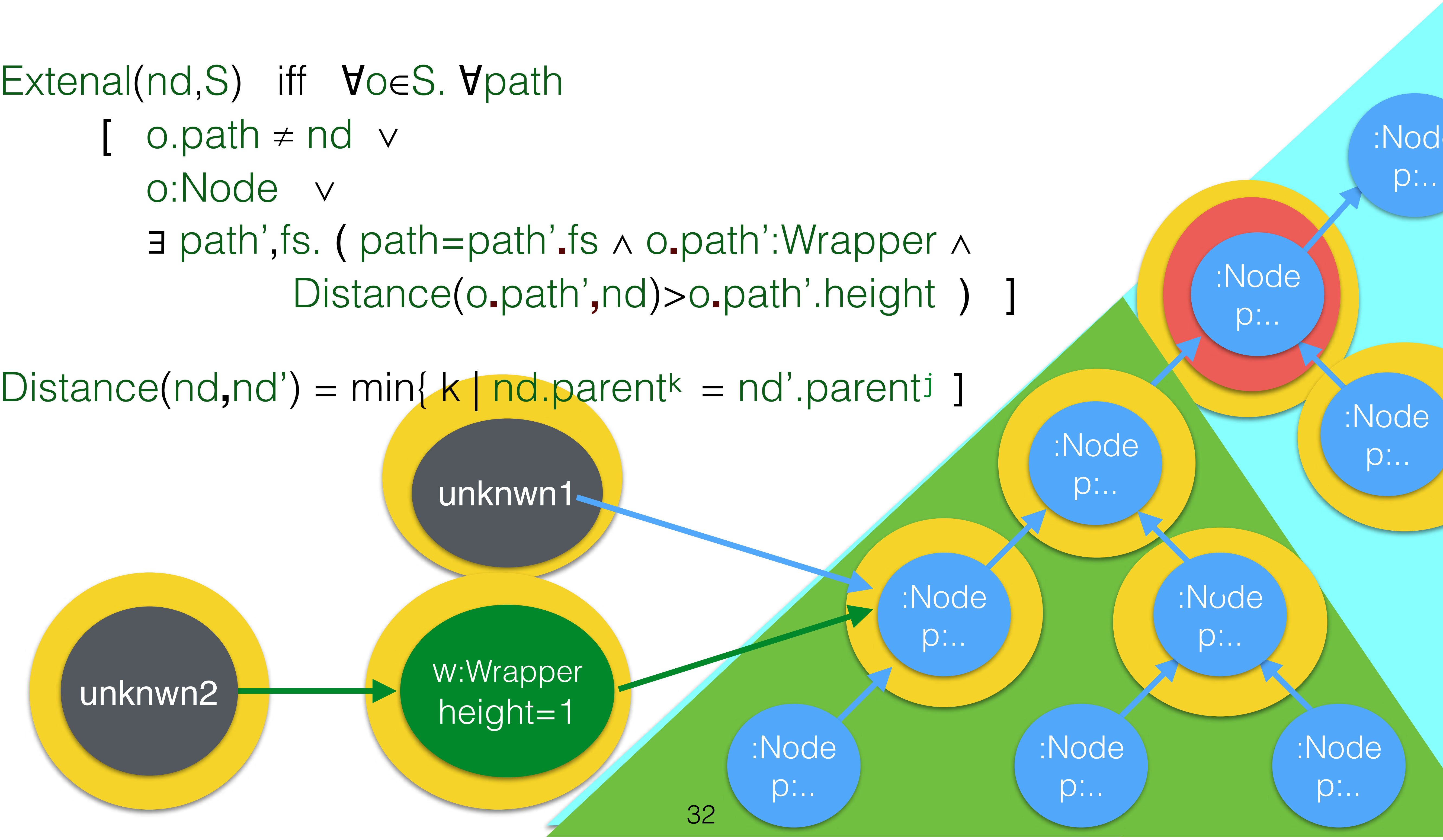
[o.path ≠ nd ∨

o:Node ∨

∃ path',fs. (path=path'.fs ∧ o.path':Wrapper ∧

Distance(o.path',nd)>o.path'.height)]

Distance(nd,nd') = min{ k | nd.parent^k = nd'.parent^j]



external

Extenal(**RedNode**, **YellowSet**)

Extenal(nd,S) iff $\forall o \in S.\ \forall path$

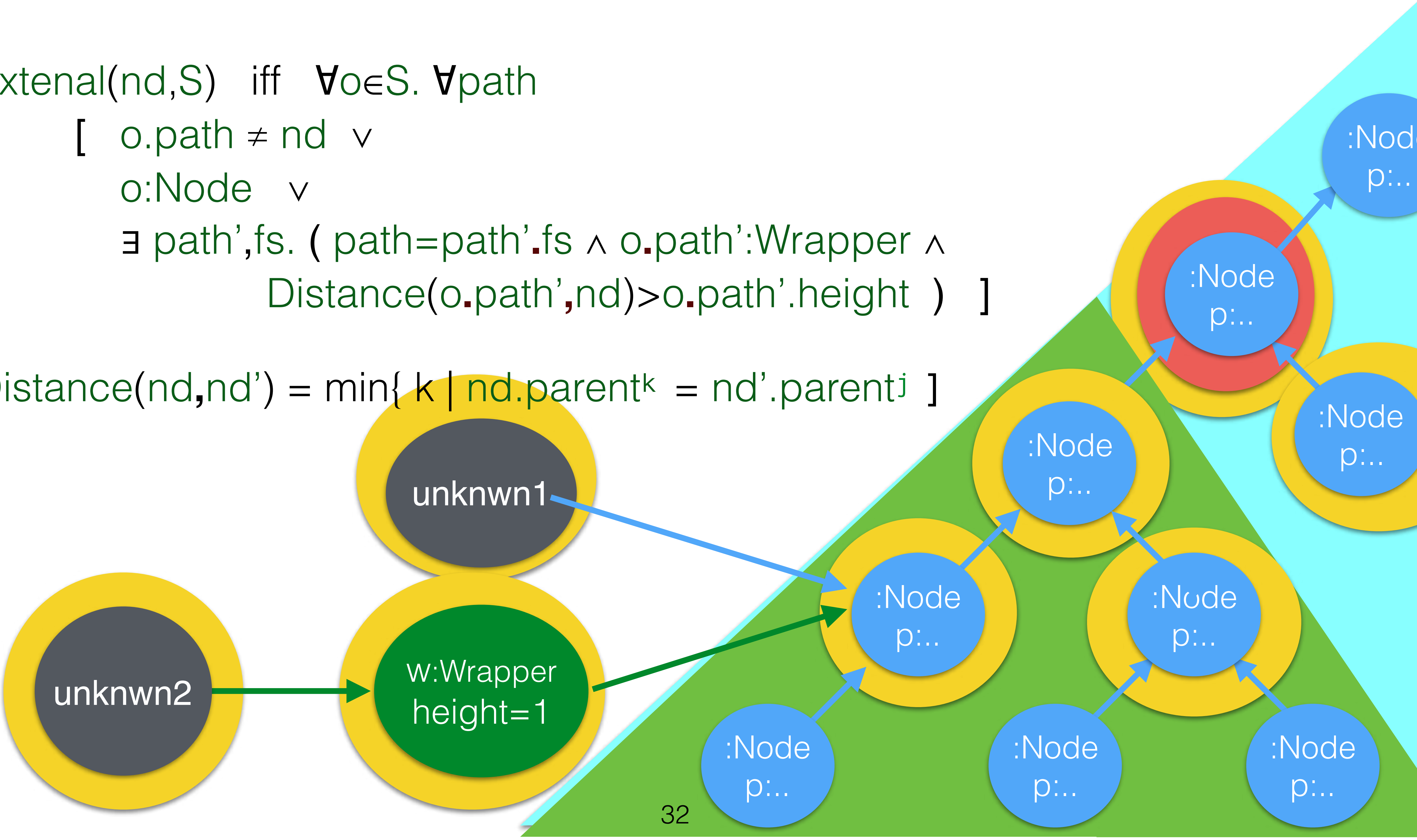
$[\ \ o.path \neq nd \ \ \vee$

$o{:}Node \ \ \vee$

$\exists\ path',fs.\ (\ path=path'.fs \wedge o.path'{:}Wrapper \wedge$

$Distance(o.path',nd) > o.path'.height\ \ )\ \ ]$

$Distance(nd,nd') = \min\{\ k \mid nd.parent^k\ = nd'.parent^j\ \ ]$

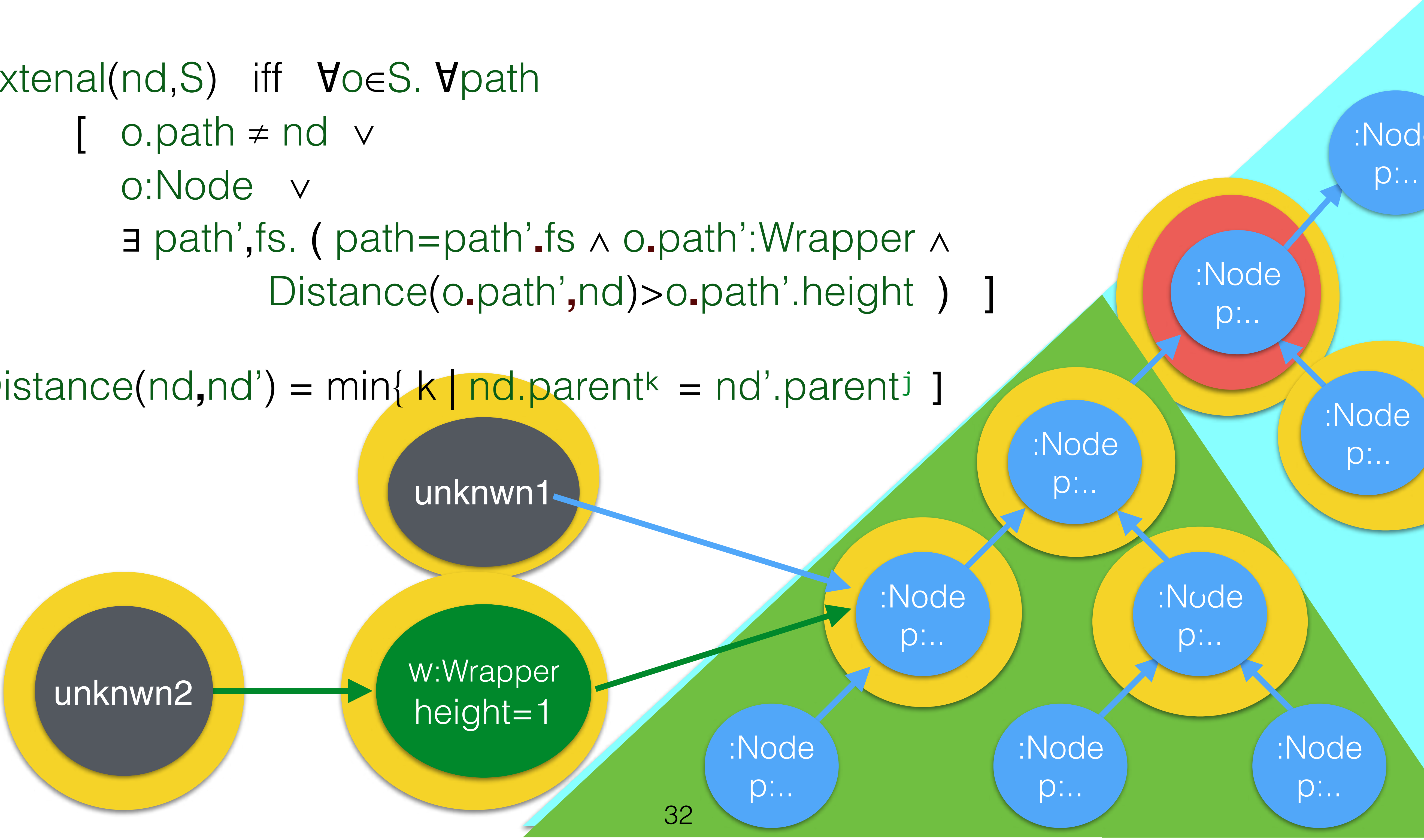


external

$$\neg\ \text{Extenal}(\textbf{RedNode}, \textbf{YellowSet})$$

$$\begin{aligned}
&\text{Extenal}(nd,S)\quad \text{iff}\quad \forall o\in S.\ \forall path\\
&\qquad [\ \ o.path \neq nd\ \ \vee\\
&\qquad\ \ \ o{:}Node\ \ \ \vee\\
&\qquad\ \ \ \exists\ path',fs.\ (\ path=path'.fs \wedge o.path'{:}Wrapper \wedge\\
&\qquad\qquad\qquad Distance(o.path',nd)>o.path'.height\ \ )\ \ ]
\end{aligned}$$

$$Distance(nd,nd') = \min\{\ k \mid nd.parent^k\ = nd'.parent^j\ \ \}$$



external

Extenal(nd,S) iff ∀o∈S. ∀path

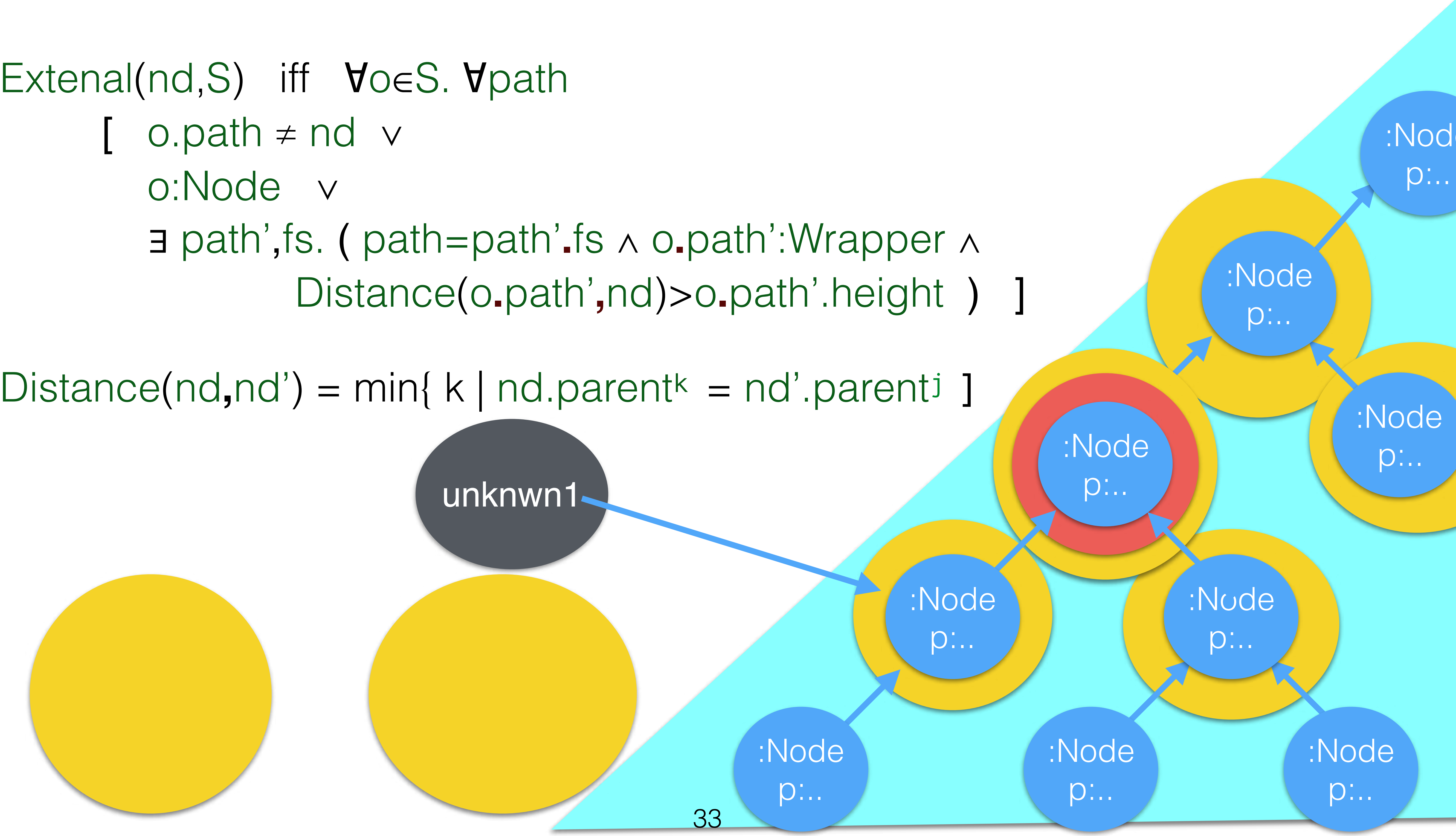
[o.path ≠ nd ∨

o:Node ∨

∃ path',fs. (path=path'.fs ∧ o.path':Wrapper ∧

Distance(o.path',nd)>o.path'.height)]

Distance(nd,nd') = min{ k | nd.parent^{k} = nd'.parent^{j}]



external

Extenal(nd,S) iff ∀o∈S. ∀path

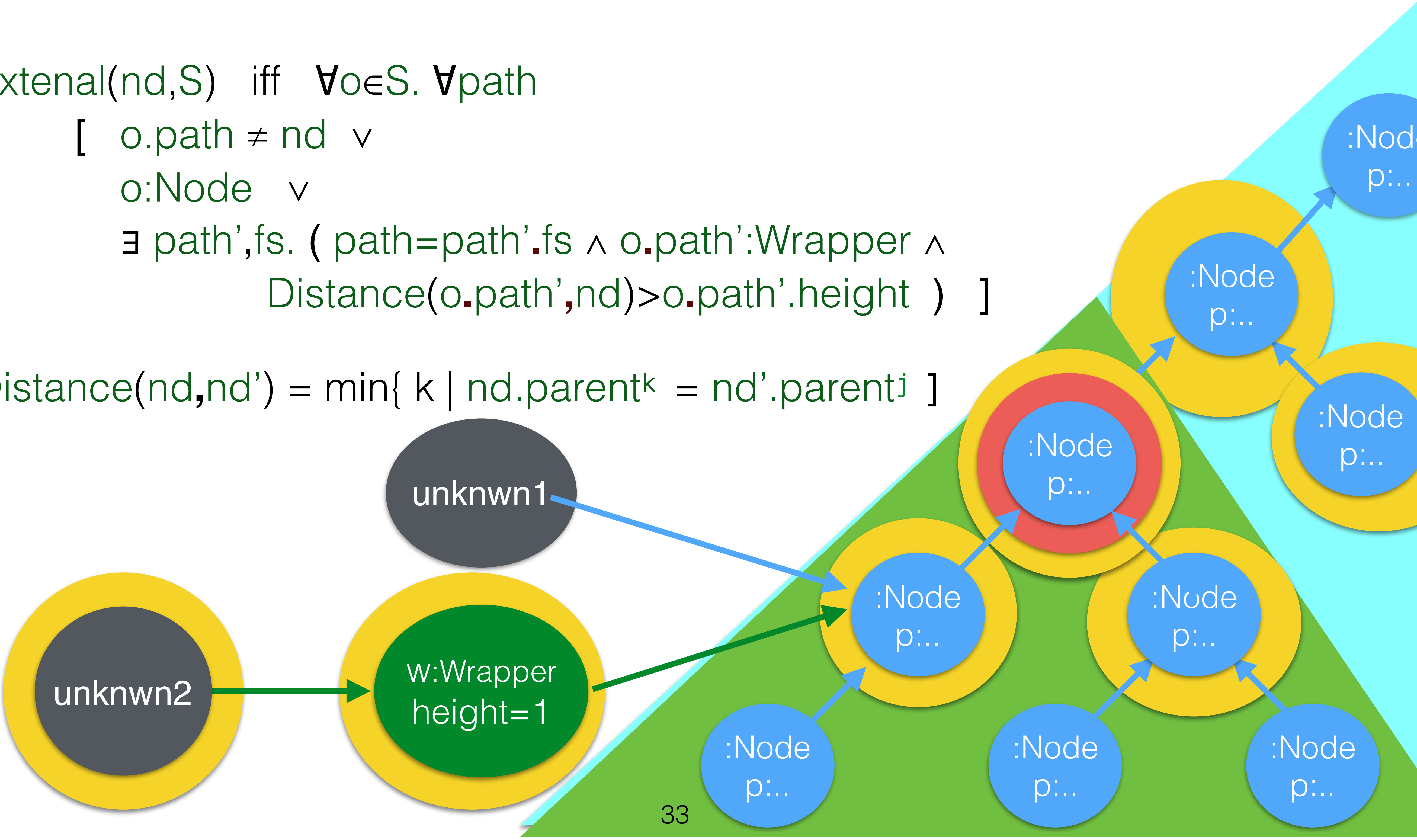
[o.path ≠ nd ∨

o:Node ∨

∃ path',fs. (path=path'.fs ∧ o.path':Wrapper ∧

Distance(o.path',nd)>o.path'.height)]

Distance(nd,nd') = min{ k | nd.parent^k = nd'.parent^j]

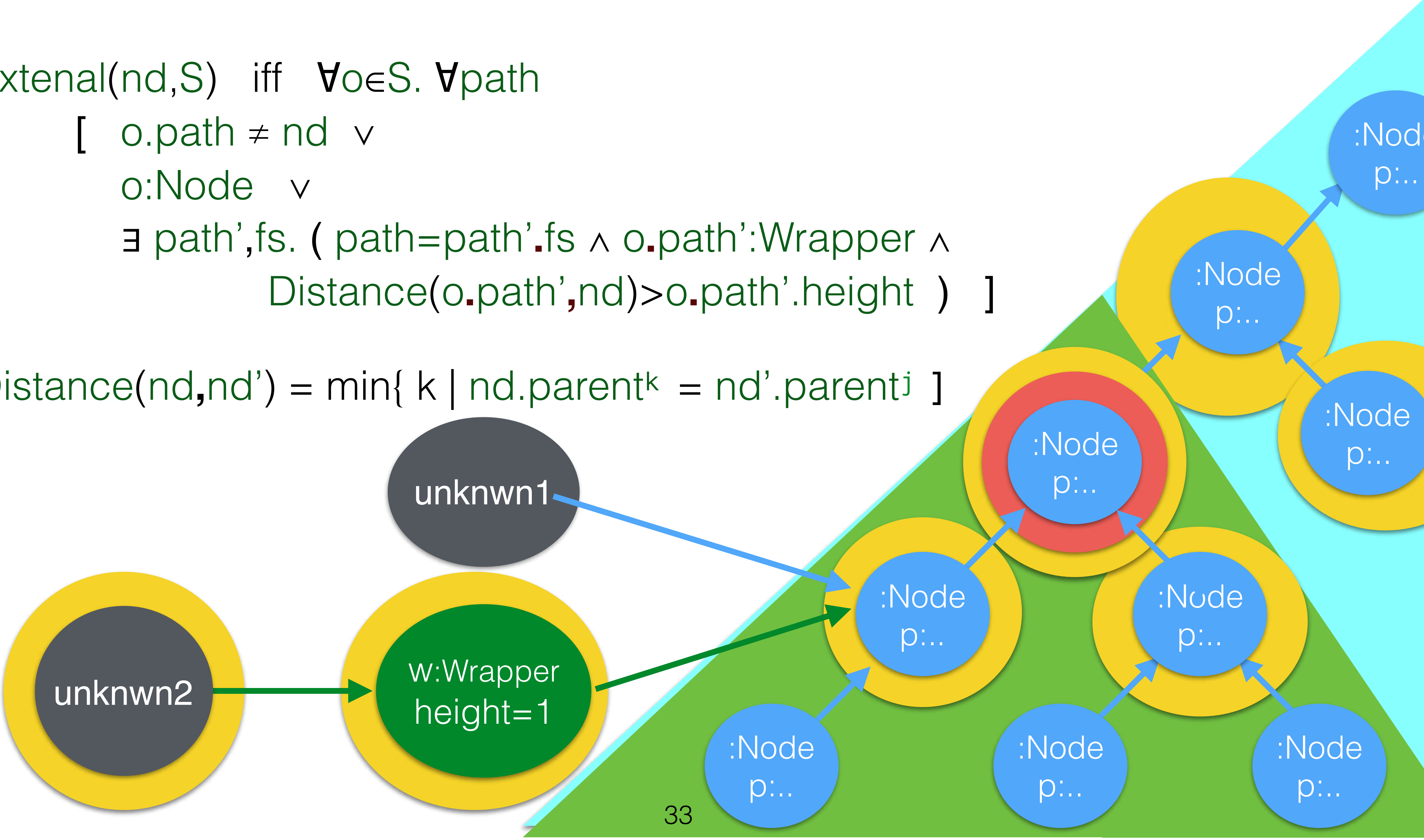


external

Extenal(**RedNode**, **YellowSet**)

Extenal(nd,S) iff ∀o∈S. ∀path
[o.path ≠ nd ∨
o:Node ∨
∃ path’,fs. (path=path’.fs ∧ o.path’:Wrapper ∧
Distance(o.path’,nd)>o.path’.height)]

Distance(nd,nd’) = min{ k | nd.parent^k = nd’.parent^j]

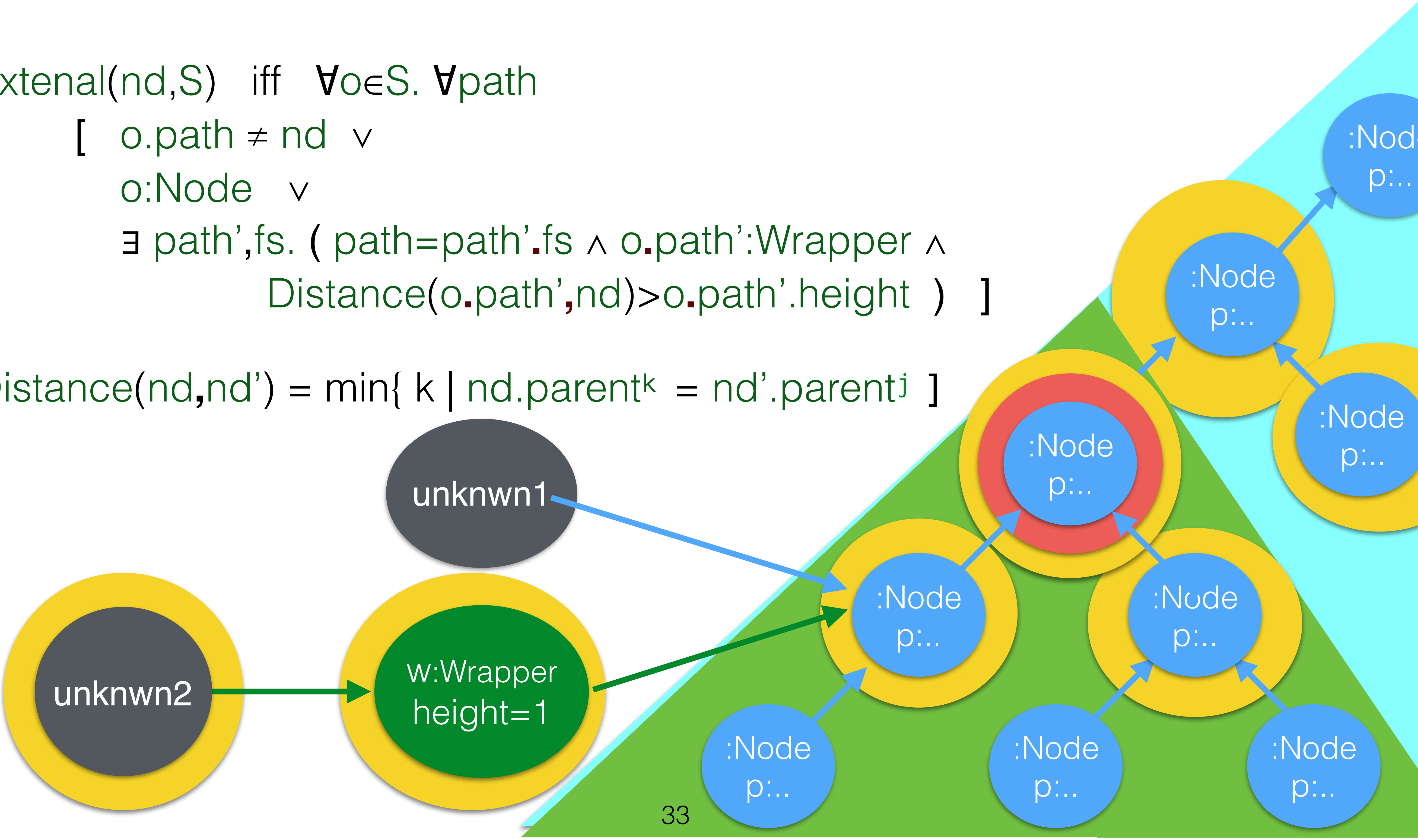


external

$\neg$ Extenal(**RedNode**, **YellowSet**)

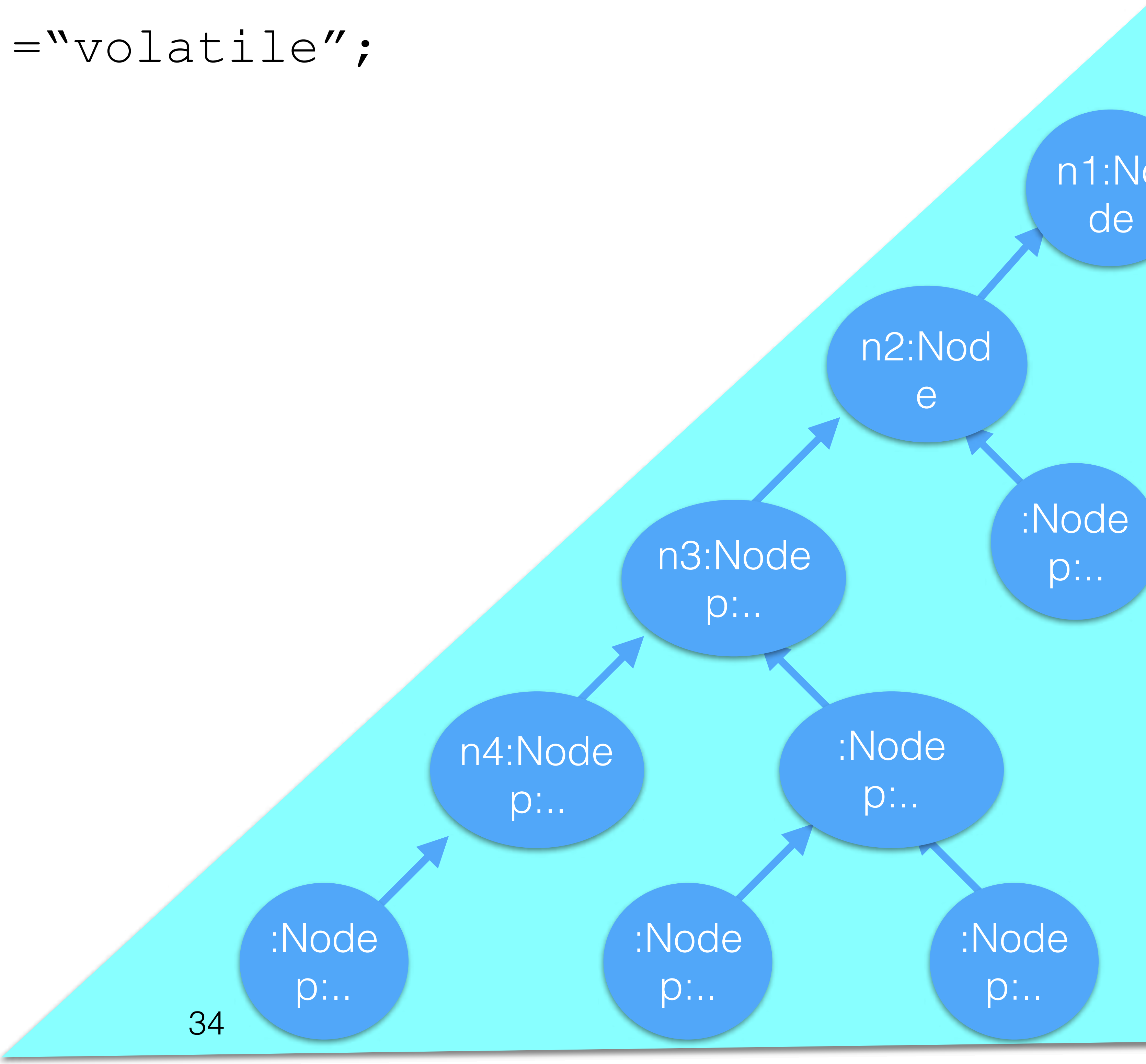
$$
\begin{array}{l}
\text{Extenal(nd,S)} \quad \text{iff} \quad \forall o \in S.\ \forall \text{path} \\
\qquad [\ \ o.\text{path} \neq \text{nd} \ \ \vee \\
\qquad\quad o:\text{Node} \quad \vee \\
\qquad\quad \exists\ \text{path}',\text{fs}.\ (\ \text{path}=\text{path}'.\text{fs} \wedge o.\text{path}':\text{Wrapper} \wedge \\
\qquad\qquad\qquad \text{Distance}(o.\text{path}',\text{nd})>o.\text{path}'.\text{height}\ \ )\ \ ]
\end{array}
$$

$$\text{Distance(nd,nd')} = \min\{\ k \mid \text{nd.parent}^{k} = \text{nd'.parent}^{j}\ \ ]$$



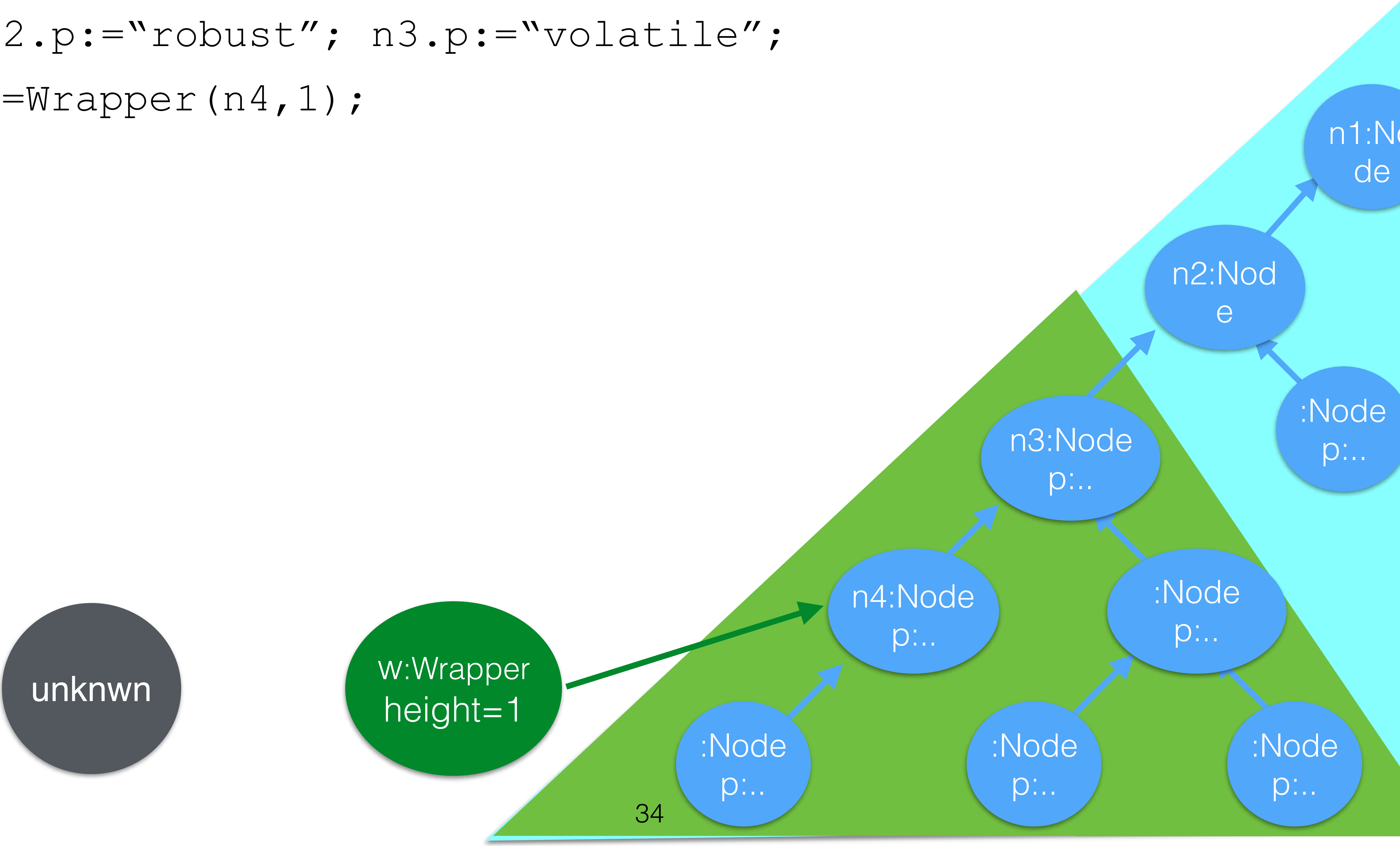
using holistic spec

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
```



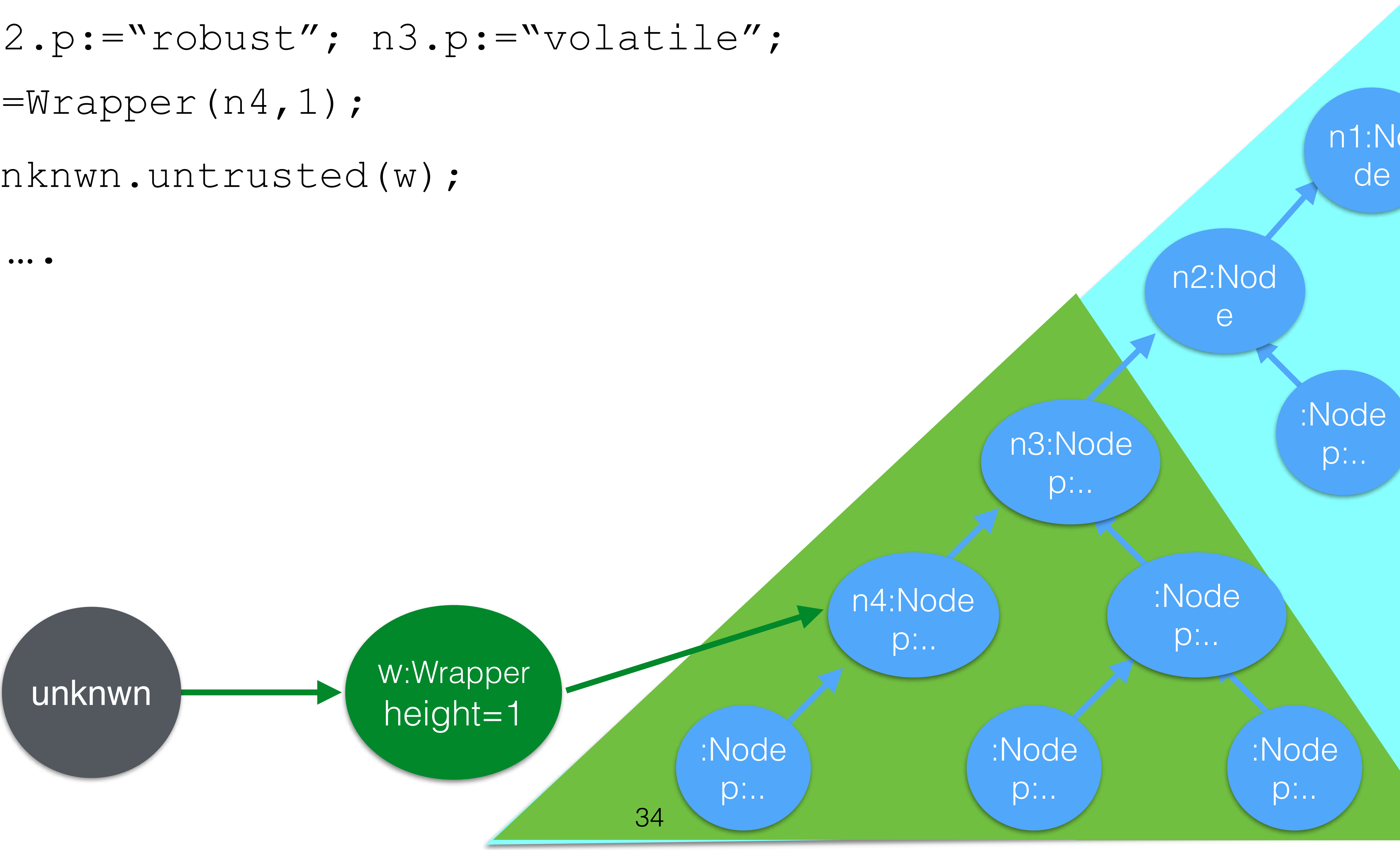
using holistic spec

```
function mm(unknwn){
   n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
   n2.p:="robust"; n3.p:="volatile";
   w=Wrapper(n4,1);
```



using holistic spec

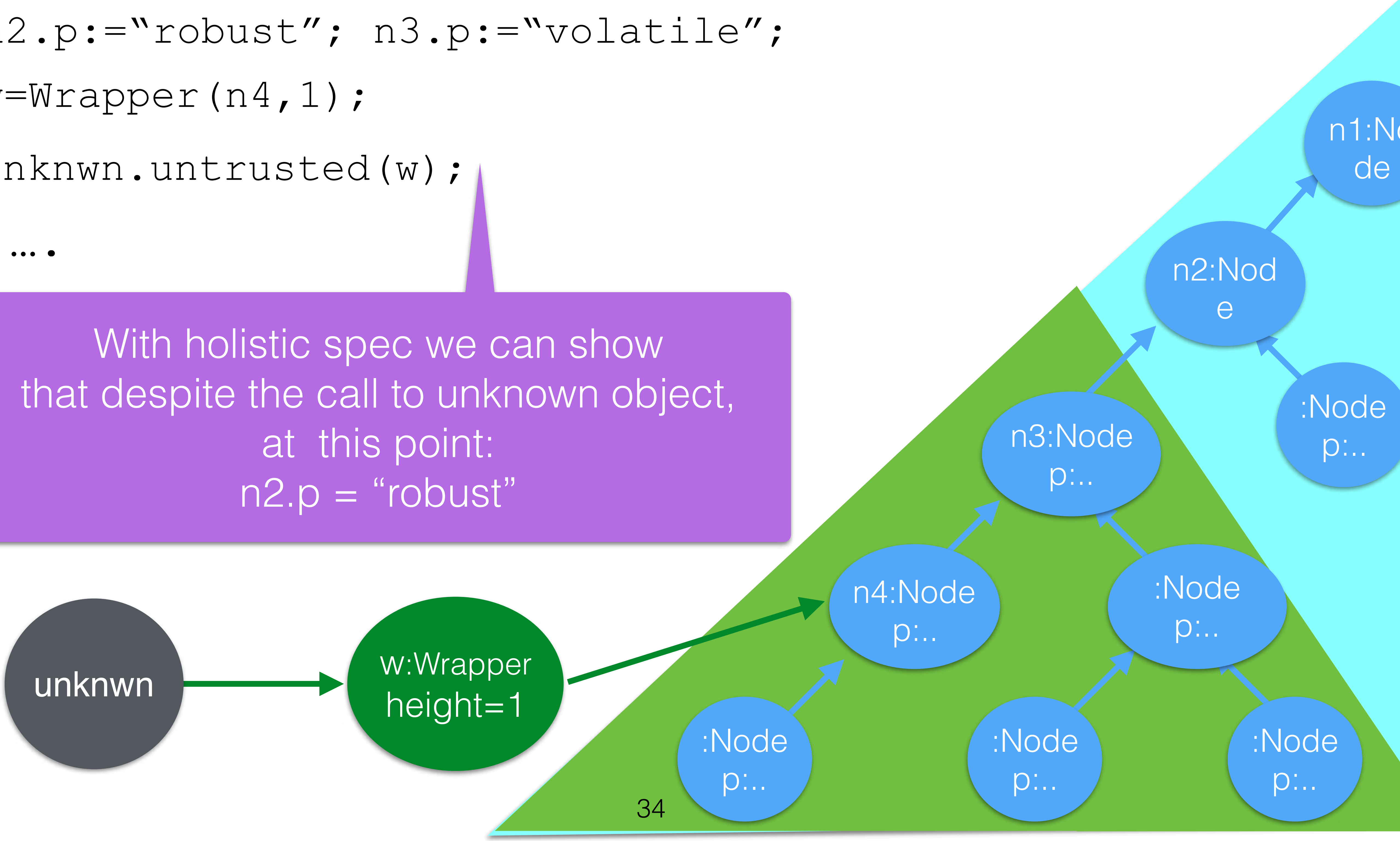
```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“volatile”;
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```



using holistic spec

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:="robust"; n3.p:="volatile";
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

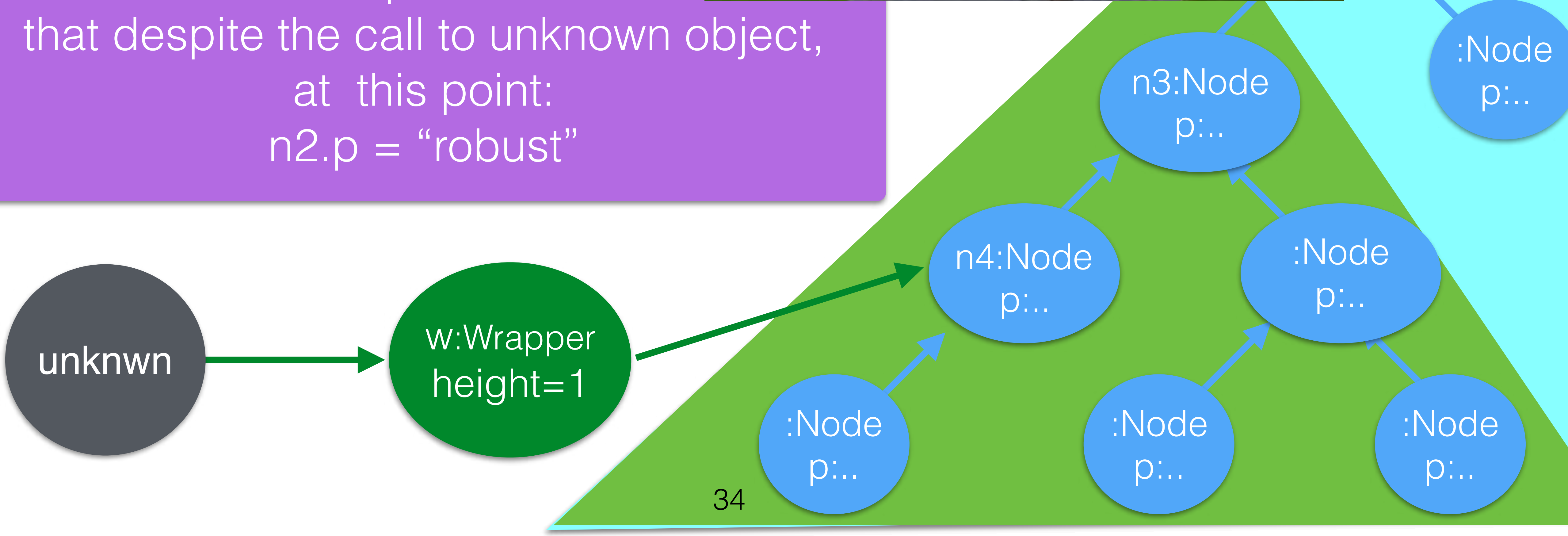
With holistic spec we can show
that despite the call to unknown object,
at this point:
n2.p = “robust”



using holistic spec

```
function mm(unknwn){
  n1:=Node(…); n2:=Node(n1,…);n3:=Node(n2,…);n4:=Node(n3,…);
  n2.p:=“robust”; n3.p:=“vola
  w=Wrapper(n4,1);
  unknwn.untrusted(w);
   ….
```

With holistic spec we can sho
that despite the call to unknown object,
at this point:
n2.p = “robust”



Bank and Account

- Banks and Accounts
- Accounts hold money
- Money can be transferred between Accounts
- A banks' currency = sum of balances of accounts held by bank

[Miller et al, Financial Crypto 2000]

Bank/Account - 2

classical

robustness

- **Pol_1**: With two accounts of same bank one can transfer money between them.
- **Pol_2**: Only someone with the Bank of a given currency can violate conservation of that currency
- **Pol_3**: The bank can only inflate its own currency
- **Pol_4**: No one can affect the balance of an account they do not have.
- **Pol_5**: Balances are always non-negative.
- **Pol_6**: A reported successful deposit can be trusted as much as one trusts the account one is depositing to.

[Miller et al, Financial Crypto 2000]

Pol_4 — holistic

- **Pol_4**: No-one can affect the balance of an account they do not have

Pol_4 — holistic

- **Pol_4**: No-one can affect the balance of an account they do not have

a:Account ∧ **Will** (**Changes**(a.balance)) **in** S)

→

Pol_4 — holistic

- **Pol_4**: No-one can affect the balance of an account they do not have

$$a{:}\mathrm{Account} \quad \wedge \quad \mathbf{Will}\ (\ \mathbf{Changes}(a.\mathrm{balance})\ )\ \mathbf{in}\ S\ \ )$$

$$\rightarrow$$

$$\exists o{\in}S.\ \mathrm{Access}(o,a)$$

Pol_4 — holistic

- **Pol_4**: No-one can affect the balance of an account
  they do not have

$$\mathrm{a{:}Account} \quad \wedge \quad \mathbf{Will}\ (\ \mathbf{Changes}(\mathrm{a.balance})\ )\ \mathbf{in}\ \mathrm{S}\ \ )$$

$$\rightarrow$$

$$\exists \mathrm{o} \in \mathrm{S}.\ \mathrm{Access}(\mathrm{o},\mathrm{a})$$

This says: If some execution starts now and involves at most the objects from S, and modifies a.balance at some future time, then at least one of the objects in S can access a directly now.

Pol_4 — holistic

- **Pol_4**: No-one can affect the balance of an account they do not have

$$a{:}\mathrm{Account} \ \wedge \ \textbf{Will}\ (\ \textbf{Changes}(\mathrm{a.balance})\ )\ \textbf{in}\ S\ \ )$$

$$\rightarrow$$

necessary condition

$$\exists o \in S.\ \mathrm{Access}(o,a)$$

This says: If some execution starts now and involves at most the objects from S, and modifies a.balance at some future time, then at least one of the objects in S can access a directly now.

Pol_4 — classical

- **Pol_4**: No-one can affect the balance of an account they do not have

Pol_4 — classical

- **Pol_4**: No-one can affect the balance of an account they do not have

???? 😕 😕 😕 ????

Today

- Traditional Specifications do not adequately address Robustness
- Holistic Specifications — Summary and by Example
- **Holistic Specification Semantics**

Giving meaning to holistic Assertions

Giving meaning to holistic Assertions

We define in a“conventional” way (omit from slides):

module	$M : \text{Ident} \longrightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef}$
configuration	$\sigma : \text{Heap} \times \text{Stack} \times \text{Code}$
execution	$M, \sigma \rightsquigarrow \sigma'$

Giving meaning to holistic Assertions

We define in a“conventional” way (omit from slides):

module	$M : \mathit{Ident} \longrightarrow \mathit{ClassDef} \cup \mathit{PredicateDef} \cup \mathit{FunctionDef}$
configuration	$\sigma : \mathit{Heap} \times \mathit{Stack} \times \mathit{Code}$
execution	$M, \sigma \rightsquigarrow \sigma'$

Define module concatenation * so that

$M * M'$ undefined, iff $dom(M) \cap dom(M') \neq \emptyset$

otherwise

$(M * M')(id) = M(id)$ if $M'(id)$ undefined, else $M'(id)$

Giving meaning to holistic Assertions

We define in a“conventional” way (omit from slides):

module	M : Ident —-> ClassDef ∪ PredicateDef ∪ FunctionDef
configuration	σ : Heap × Stack × Code
execution	M, σ ⇝ σ’

Define module concatenation * so that

M*M’ undefined, iff dom(M)∩dom(M’) ≠∅

otherwise

(M*M’)(id) = M(id) if M’(id) undefined, else M’(id)

Lemma

- M*M’ = M’*M
- (M1*M2)*M3 = M1*(M2*M3)
- M, σ ⇝ σ’ ∧ M*M’ defined ⟶ M*M’, σ ⇝ σ’

Giving meaning to holistic Assertions

We define in a“conventional” way (omit from slides):

module	$M : \text{Ident} \longrightarrow \text{ClassDef} \cup \text{PredicateDef} \cup \text{FunctionDef}$
configuration	$\sigma : \text{Heap} \times \text{Stack} \times \text{Code}$
execution	$M, \sigma \rightsquigarrow \sigma'$

Define module concatenation * so that

$M{*}M'$ undefined, iff $dom(M) \cap dom(M') \neq \emptyset$

otherwise

$(M{*}M')(id) = M(id)$ if $M'(id)$ undefined, else $M'(id)$

We will define

$M, \sigma \models A$

$Initial(\sigma)$ and $Arising(M)$

$M \models A$

Giving meaning to holistic Assertions

Giving meaning to holistic Assertions

We define in a“conventional” way (omit from slides):

module	M : Ident —-> ClassDef ∪ PredicateDef ∪ FunctionDef
configuration	σ : Heap × Stack × Code
execution	M, $\sigma \rightarrow \sigma'$

Define module concatenation * so that

M*M’ undefined, iff dom(M)∩dom(M’) ≠∅

otherwise

(M*M’)(id) = M(id) if M’(id) undefined, else M’(id)

Giving meaning to holistic Assertions

We define in a“conventional” way (omit from slides):

module	M : Ident —-> ClassDef ∪ PredicateDef ∪ FunctionDef
configuration	σ : Heap × Stack × Code
execution	$M, \sigma \to \sigma'$

Define module concatenation * so that

$M*M'$ undefined, iff $dom(M) \cap dom(M') \neq \emptyset$

otherwise

$(M*M')(id) = M(id)$ if $M'(id)$ undefined, else $M'(id)$

We will define $M, \sigma \models A$

Initial(σ) and Arising(M)

$M \models A$

Holistic Assertions — summary

Holistic Assertions — summary

e ::= this | x | e.fld | …

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

Holistic Assertions — summary

$$
\begin{array}{lll}
e & ::= & \mathrm{this} \mid x \mid e.\mathrm{fld} \mid \ldots \\
A & ::= & e>e \mid e=e \mid \ldots \\
 & \mid & A \to A \mid A \land A \mid \exists x.\ A \mid \ldots
\end{array}
$$

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …
| A → A | A ∧ A | ∃x. A | …
| **Access**(e,e’)
| **Changes**(e)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

| A **in** S

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …

| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’)

| **Changes**(e)

| **Will**(A) | **Was**(A)

| A **in** S

| x.**Call**(y,m,z1,..zn)

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …
| A → A | A ∧ A | ∃x. A | …
| **Access**(e,e’)
| **Changes**(e)
| **Will**(A) | **Was**(A)
| A **in** S
| x.**Call**(y,m,z1,..zn)
| x **obeys** A

Holistic Assertions — summary

e ::= this | x | e.fld | …

A ::= e>e | e=e | …
| A → A | A ∧ A | ∃x. A | …

| **Access**(e,e’) permission

| **Changes**(e) authority

| **Will**(A) | **Was**(A) time

| A **in** S space

| x.**Call**(y,m,z1,..zn) control

| x **obeys** A trust

Semantics of Expressions

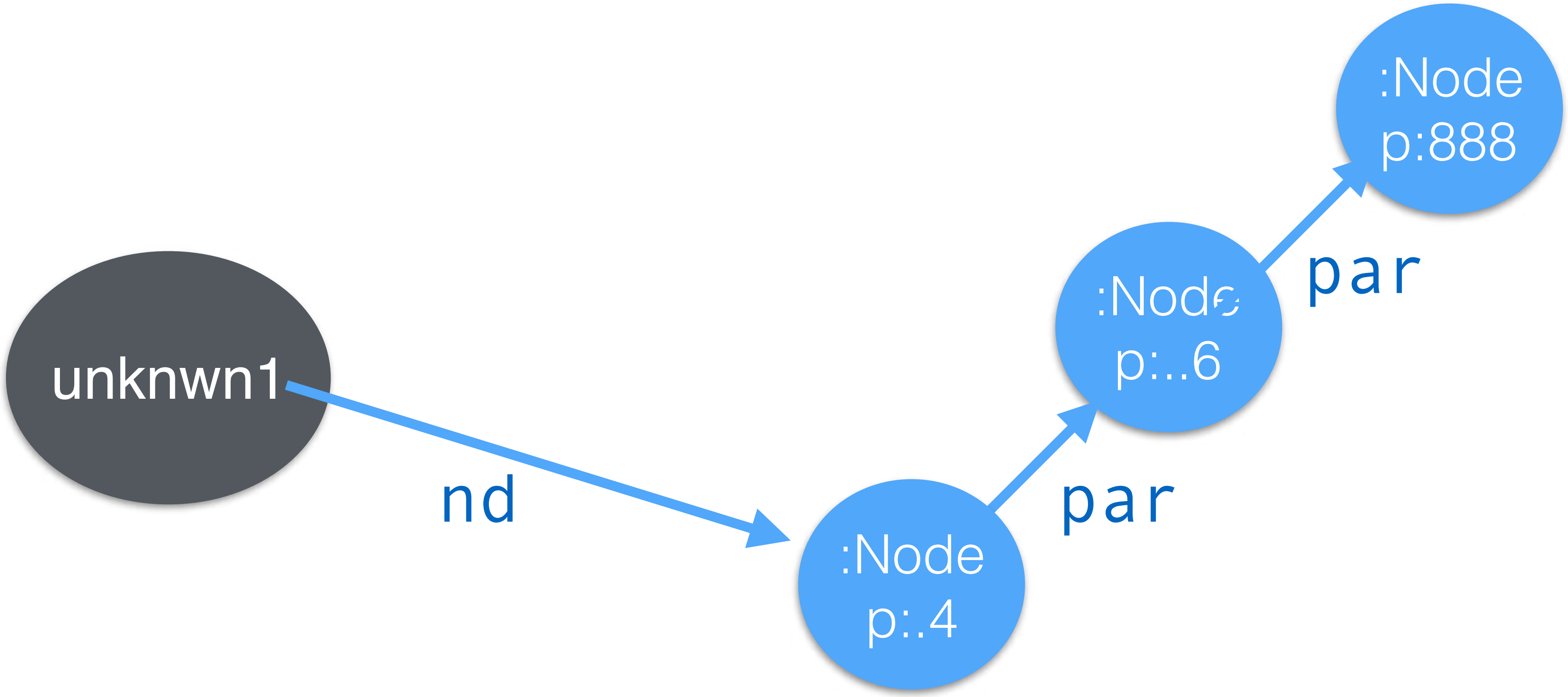
e ::= this | x | e.fld | func(e1,…en) | …

Define $\lfloor e \rfloor_{M,\sigma}$ as expected

Semantics of Expressions

e ::= this | x | e.fld | func(e1,…en) | …

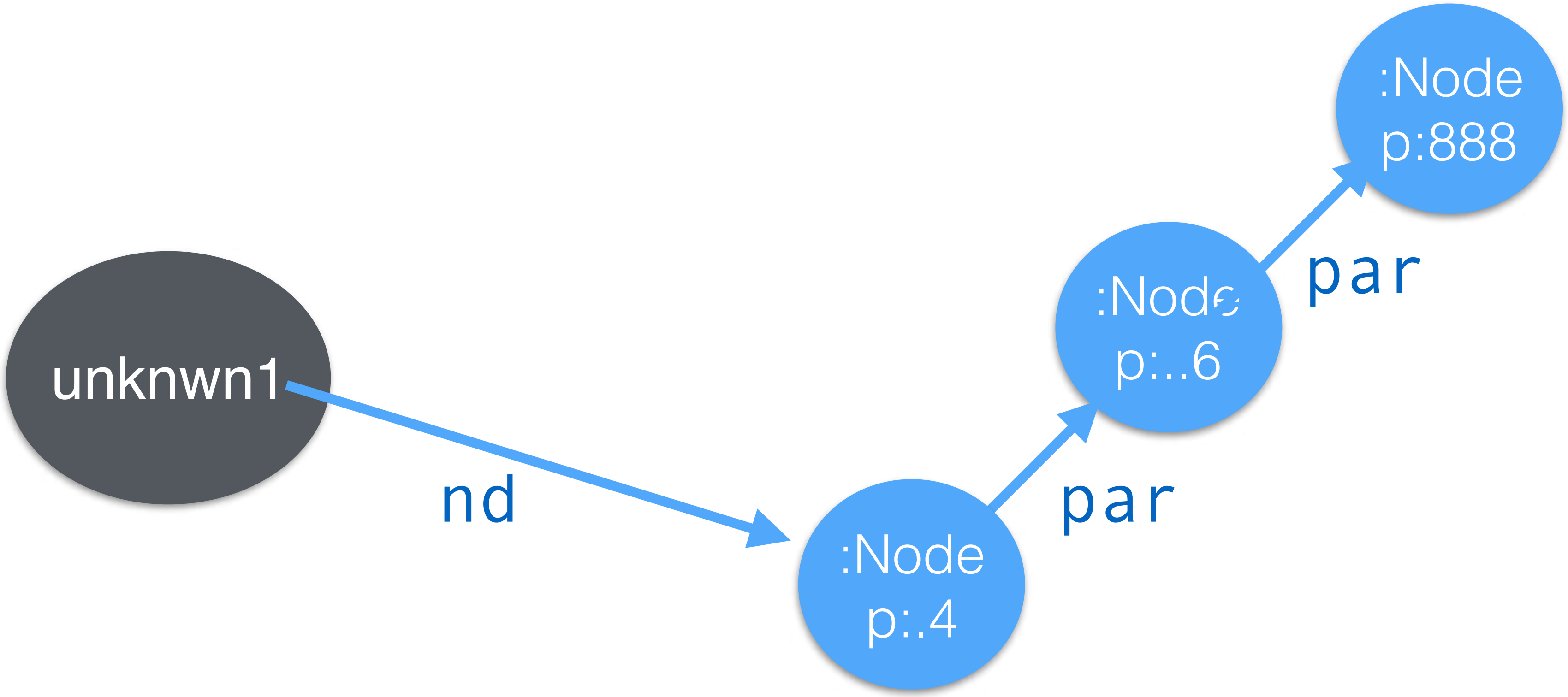
Define $\lfloor e \rfloor_{M,\sigma}$ as expected



Semantics of Expressions

e ::= this | x | e.fld | func(e1,…en) | …

Define $\lfloor e \rfloor_{M,\sigma}$ as expected



Eg, $\lfloor \text{unknwn1.nd.par.par.p} \rfloor_{M,\sigma} = 888$

Semantics of holistic Assertions

“Conventional part”

$A \ ::= \ e>e \ \mid \ A \rightarrow A \ \mid \ \exists x.A \mid \ \ldots$

Semantics of holistic Assertions

“Conventional part”

$A \ ::= \ e>e \ \ | \ \ A \rightarrow A \ \ | \ \ \exists x.A \ | \ \ldots$

We define $M, \sigma \models A$

Semantics of holistic Assertions

“Conventional part”

$A \ ::= \ e>e \ \mid \ A \rightarrow A \ \mid \ \exists x.A \mid \ \dots$

We define $M, \sigma \models A$

$M,\sigma \models e>e'$ iff $\lfloor e \rfloor_{M,\sigma} > \lfloor e' \rfloor_{M,\sigma}$

$M,\sigma \models A \rightarrow A'$ iff $M,\sigma \models A$ implies $M,\sigma \models A'$

$M,\sigma \models \exists x.A$ iff $M,\sigma[z \mapsto \iota] \models A[x \mapsto z]$
for some $\iota \in dom(\sigma.heap)$, and z free in A

Semantics of holistic Assertions

“Unconventional part”

A ::= **Access**(x,x') | **Changes**(e) | **Will**(A) | A **in** S | x.**Calls**(y,m,z_1,..z_n)

Semantics of holistic Assertions

“Unconventional part”

$A ::= \mathbf{Access}(x,x') \;|\; \mathbf{Changes}(e) \;|\; \mathbf{Will}(A) \;|\; A\ \mathbf{in}\ S \;|\; x.\mathbf{Calls}(y,m,z_1,..z_n)$

$M,\sigma \models \mathbf{Access}(x,x')$ iff $\lfloor x \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma} \quad \vee$

$\lfloor x.fld \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ for some field $fld \quad \vee$

$\lfloor this \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \wedge \lfloor y \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$

$\wedge$ y is formal parameter of current function

Semantics of holistic Assertions

“Unconventional part”

$A ::= \textbf{Access}(x,x') \mid \textbf{Changes}(e) \mid \textbf{Will}(A) \mid A\ \textbf{in}\ S \mid x.\textbf{Calls}(y,m,z_1,..z_n)$

$M,\sigma \vDash \textbf{Access}(x,x')$ iff $\lfloor x \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma} \quad \vee$

$\lfloor x.fld \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ for some field $fld \quad \vee$

$\lfloor this \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \wedge \lfloor y \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$

$\wedge$ y is formal parameter of current function

$M,\sigma \vDash \textbf{Changes}(e)$ iff $M, \sigma \rightsquigarrow \sigma' \wedge \lfloor e \rfloor_{M,\sigma} \neq \lfloor e \rfloor_{M,\sigma'}$

Semantics of holistic Assertions

"Unconventional part"

$A \ ::= \ \textbf{Access}(x,x') \ | \ \textbf{Changes}(e) \ | \ \textbf{Will}(A) \ | \ A \ \textbf{in} \ S \ | \ x.\textbf{Calls}(y,m,z_1,..z_n)$

$M,\sigma \vDash \textbf{Access}(x,x')$ iff $\lfloor x \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma} \quad \lor$

$\lfloor x.fld \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ for some field $fld \quad \lor$

$\lfloor this \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \land \lfloor y \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$

$\land$ y is formal parameter of current function

$M,\sigma \vDash \textbf{Changes}(e)$ iff $M, \sigma \rightsquigarrow \sigma' \land \lfloor e \rfloor_{M,\sigma} \neq \lfloor e \rfloor_{M,\sigma'}$

$M,\sigma \vDash \textbf{Will}(A)$ iff $\exists \sigma'.[\ M,\sigma \rightsquigarrow^* \sigma' \land M, \sigma' \vDash A\ ]$

Semantics of holistic Assertions

“Unconventional part”

$A \ ::= \ \textbf{Access}(x,x') \ \mid \ \textbf{Changes}(e) \ \mid \ \textbf{Will}(A) \ \mid \ A \ \textbf{in} \ S \mid \ x.\textbf{Calls}(y,m,z_1,..z_n)$

$M,\sigma \models \textbf{Access}(x,x')$ iff $\lfloor x \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma} \quad \vee$

$\lfloor x.fld \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ for some field fld $\quad \vee$

$\lfloor this \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \wedge \lfloor y \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$

$\wedge$ y is formal parameter of current function

$M,\sigma \models \textbf{Changes}(e)$ iff $M, \sigma \to \sigma' \ \wedge \ \lfloor e \rfloor_{M,\sigma} \neq \lfloor e \rfloor_{M,\sigma'}$

$M,\sigma \models \textbf{Will}(A)$ iff $\exists \sigma'.[\ M,\sigma \to^{*} \sigma' \ \wedge \ M, \sigma' \models A\ ]$

$M,\sigma \models A \ \textbf{in} \ S$ iff $M,\sigma@_{Os} \models A$ where $Os = \lfloor S \rfloor_{M,\sigma}$

Semantics of holistic Assertions

“Unconventional part”

$$A \;::=\; \textbf{Access}(x,x') \;\mid\; \textbf{Changes}(e) \;\mid\; \textbf{Will}(A) \;\mid\; A \textbf{ in } S \mid\; x.\textbf{Calls}(y,m,z_1,..z_n)$$

$M,\sigma \models \textbf{Access}(x,x')$ iff $\lfloor x \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma} \quad \vee$

$\lfloor x.fld \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$ for some field $fld \quad \vee$

$\lfloor this \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \wedge \lfloor y \rfloor_{M,\sigma} = \lfloor x' \rfloor_{M,\sigma}$

$\wedge$ y is formal parameter of current function

$M,\sigma \models \textbf{Changes}(e)$ iff $M,\, \sigma \rightarrow \sigma' \;\wedge\; \lfloor e \rfloor_{M,\sigma} \neq \lfloor e \rfloor_{M,\sigma'}$

$M,\sigma \models \textbf{Will}(A)$ iff $\exists\sigma'.[\, M,\sigma \rightarrow^{*} \sigma' \;\wedge\; M,\, \sigma' \models A \,]$

$M,\sigma \models A \textbf{ in } S$ iff $M,\sigma@_{Os} \models A$ where $Os = \lfloor S \rfloor_{M,\sigma}$

$M,\sigma \models x.\textbf{Calls}(y,m,z_1,..z_n)$ iff $\lfloor this \rfloor_{M,\sigma} = \lfloor x \rfloor_{M,\sigma} \;\wedge\; \sigma.code{=}y'.m(z_1'..z_n') \;\wedge$ ..

Semantics of holistic Assertions
- the full truth -

$M,\sigma \models \textbf{Access}(e,e')$ iff … as before …

$M,\sigma \models \textbf{Changes}(e)$ iff $M, \sigma \rightsquigarrow \sigma' \ \wedge\ \lfloor e \rfloor_{M,\sigma} \neq \lfloor e[z\mapsto y] \rfloor_{M,\sigma'[y\mapsto\sigma(z)]}$

where $\{z\}=\text{Free}(e) \wedge y$ fresh in e, σ,σ'

$M,\sigma \models \textbf{Will}(A)$ iff $\exists\sigma',\sigma'',\phi.[\ \sigma = \sigma'.\phi \wedge\ M,\phi \rightsquigarrow^{*}\sigma' \ \wedge$

$M, \sigma'[y\mapsto\sigma(z)] \models\ A[z\mapsto y]\ ]$

where $\{z\}=\text{Free}(A) \wedge y$ fresh in A, σ,σ'

$M,\sigma \models A\ \textbf{In}\ S$ iff $M,\sigma@_{Os} \models A$ where $Os = \lfloor S \rfloor_{M,\sigma}$

$M,\sigma \models x.\textbf{Calls}(y,m,z_1,..z_n)$ iff … as before …

Semantics of holistic Assertions

- the full truth -

$M,\sigma \models$ **Access**(e,e') iff … as before …

$M,\sigma \models$ **Changes**(e) iff $M, \sigma \rightarrow \sigma' \ \wedge\ \lfloor e \rfloor_{M,\sigma} \neq \lfloor e[z \mapsto y] \rfloor_{M,\sigma'[y \mapsto \sigma(z)]}$

where $\{z\}=Free(e) \wedge y$ fresh in e, σ,σ'

$M,\sigma \models$ **Will**(A) iff $\exists \sigma',\sigma'',\phi.[\ \sigma = \sigma'.\phi \ \wedge\ M,\phi \rightarrow^* \sigma' \ \wedge$

$M, \sigma'[y \mapsto \sigma(z)] \models A[z \mapsto y]\ ]$

where $\{z\}=Free(A) \wedge y$ fresh in A, σ,σ'

$M,\sigma \models A$ **In** S iff $M,\sigma@_{Os} \models A$ where $Os = \lfloor S \rfloor_{M,\sigma}$

$M,\sigma \models x.$**Calls**$(y,m,z_1,..z_n)$ iff … as before …

Arising Configurations

A runtime configuration is *initial* iff
1) The heap contains only one object, of class Object
2) The stack consists of just one frame, where this points to that object.

The code can be arbitrary

$$\text{Initial}(\sigma) \quad \text{iff} \quad \sigma.\text{heap}=(1\mapsto(\text{Object},\ldots)\,) \;\wedge\; \sigma.\text{stack}=(\text{this}\mapsto 1).[\,]$$

A runtime configuration σ *arises* from a module M if there is some initial configuration σ_0 whose execution M in reaches σ in a finite number of steps.

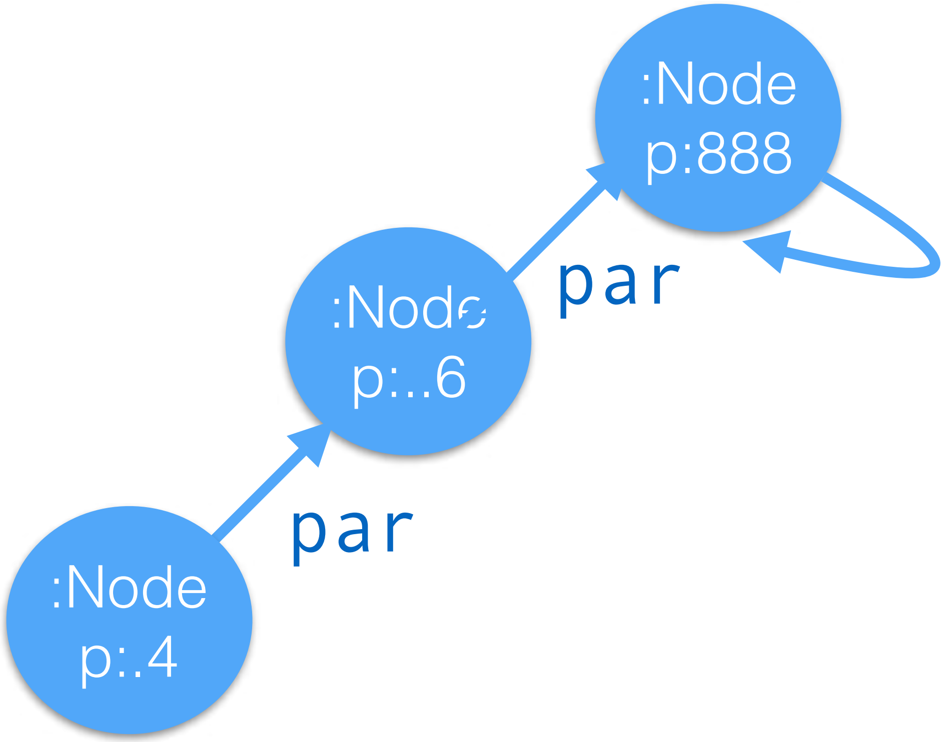
$$\text{Arising}(M) = \{\ \sigma \mid \exists\, \sigma_0.\ \text{Initial}(\sigma_0) \;\wedge\; M, \sigma_0 \rightarrow^* \sigma \ \}$$

Arising expresses “defensiveness”

Assume a Tree-module, $\mathtt{M}_{\mathtt{tree}}$.

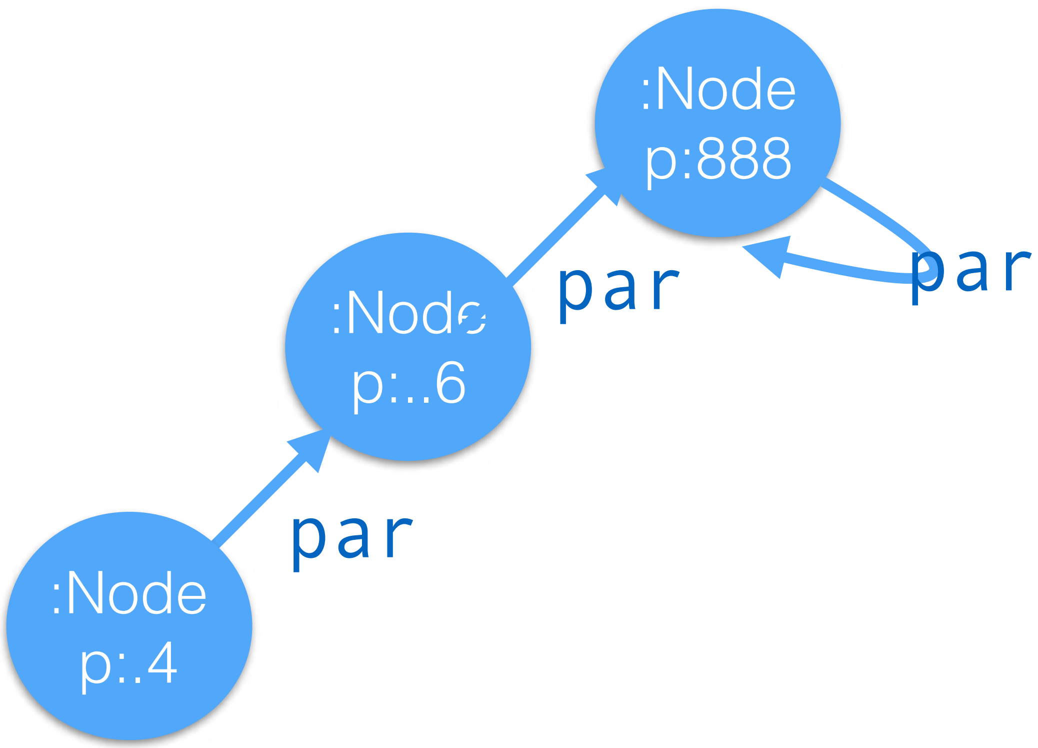
Arising expresses “defensiveness”

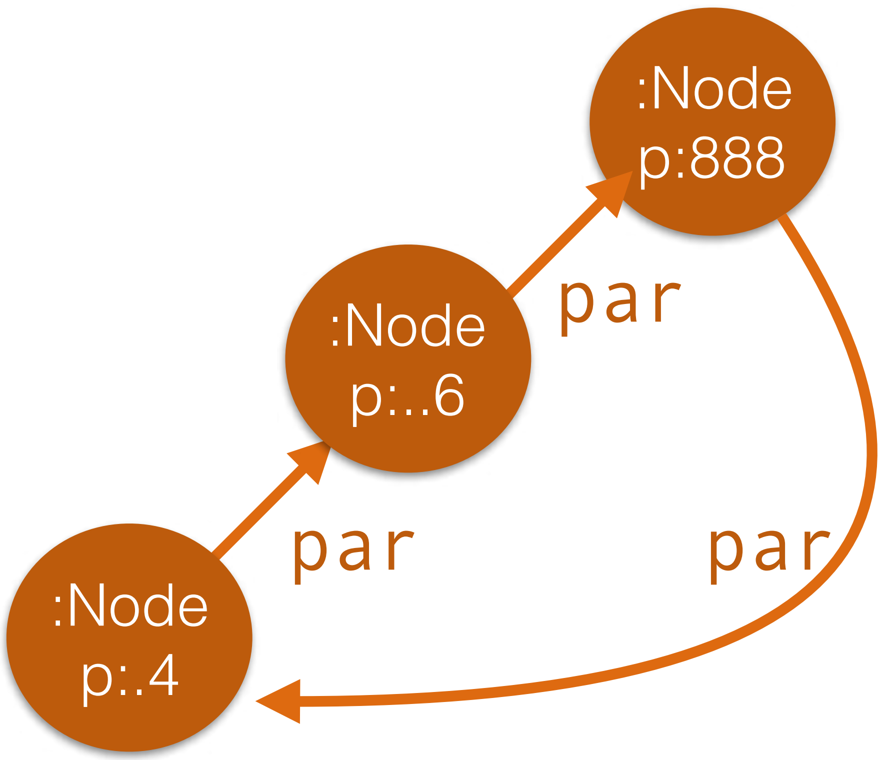
Assume a Tree-module, $\mathtt{M}_{\mathtt{tree}}$.



Arising expresses “defensiveness”

Assume a Tree-module, $\mathtt{M}_{\mathtt{tree}}$.

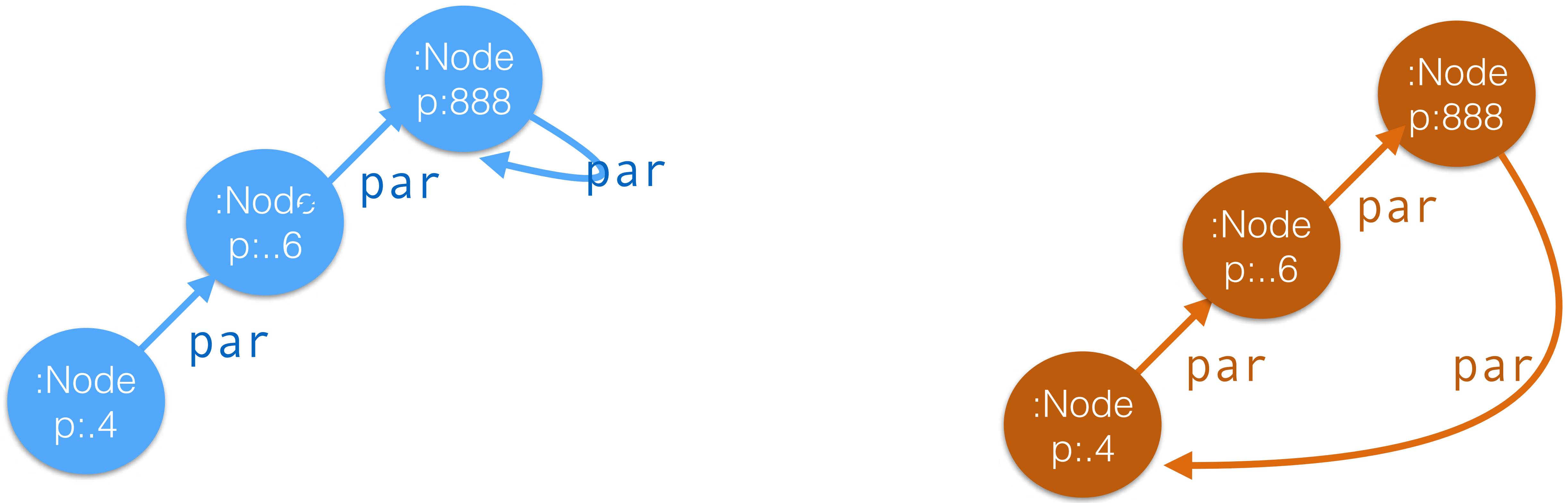




Arising expresses “defensiveness”

Assume a Tree-module, M_{tree}.

`blue configuration` *arises*
from $M_{tree}*M'$ for some module M'

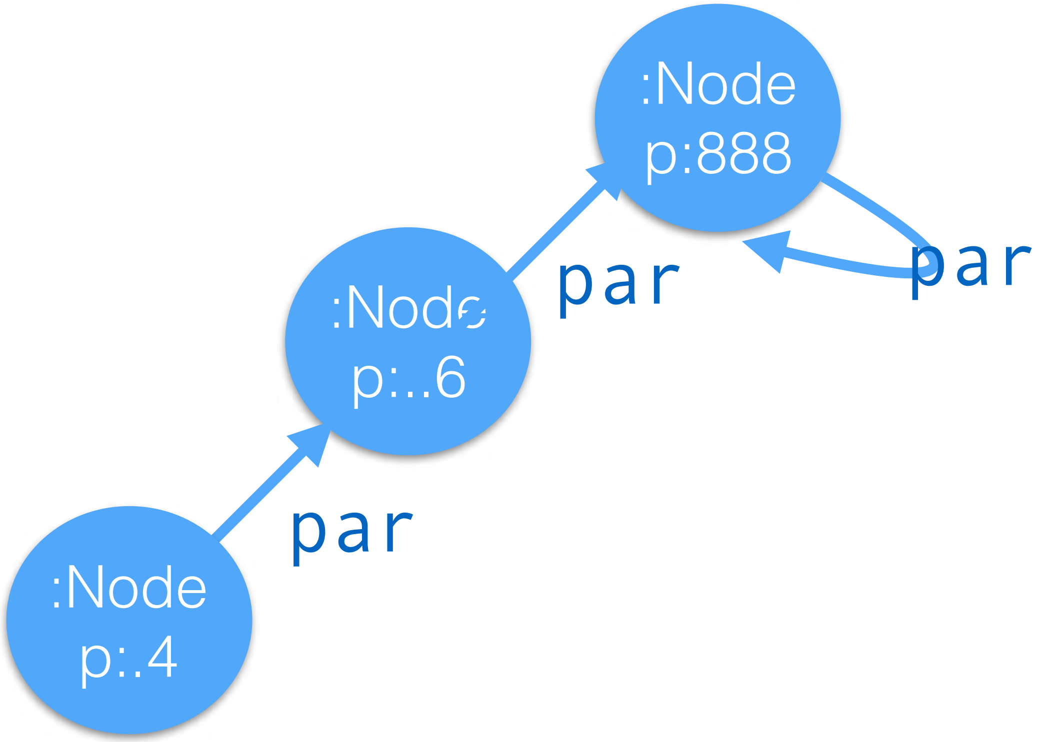


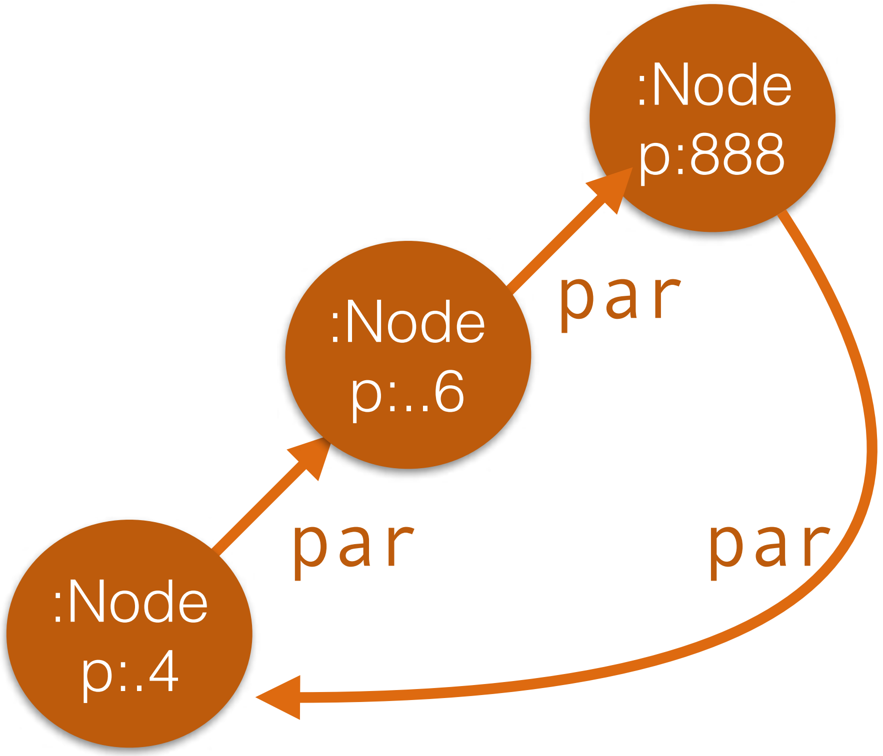
Arising expresses “defensiveness”

Assume a Tree-module, M_{tree}.

blue configuration *arises* from M_{tree}*M’ for some module M’

brown configuration *does not arise* from M_{tree}*M’ for *any* module M’





Giving meaning to Assertions

$M \models A \quad \text{iff} \quad \forall M'.\forall \sigma \in \text{Arising}(M^*M').\ \ M^*M', \sigma \models A$

A module M satisfies an assertion A if all runtime configurations σ which arise from execution of code from M*M’ (for any module M’), satisfy A.

Giving meaning to Assertions

$$M \models A \quad \text{iff} \quad \forall M'.\forall \sigma \in Arising(M{*}M').\ \ M{*}M', \sigma \models A$$

A module M satisfies an assertion A if all runtime configurations σ which arise from execution of code from M*M’ (for any module M’), satisfy A.

Giving meaning to Assertions

$M \models A \quad \text{iff} \quad \boxed{\forall M'}.\forall \sigma \in \mathit{Arising}(M*\boxed{M'}).\ \ M*M', \sigma \models A$

A module M satisfies an assertion A if all runtime configurations σ which arise from execution of code from M*M' (for any module M'), satisfy A.

open world

Summary of our Proposal

A ::= e>e | e=e | f(e1,..en) | …
| A → A | A ∧ A | ∃x. A | …

| **Access**(x,y)	permission
| **Changes**(e)	authority
| **Will**(A) | **Was**(A)	time
| A **in** S	space
| x.**Calls**(y,m,z1,..zn)	call

$M, \sigma \models A$

Arising(M)

$M \models A$

Classical Specification vs Holistic Specification

- fine-grained
- per function

- ADT as a whole
- emergent behaviour

Classical Specification vs Holistic Specification

- fine-grained
- per function

- ADT as a whole
- emergent behaviour

Classical Specification vs Holistic Specification

- fine-grained
- per function

- ADT as a whole
- emergent behaviour

Which is “stronger”?
“Closed” ADT with classical spec implies holistic spec.
(closed: no functions can be added, all functions have classical specs, ghost state has known representation)

Classical Specification vs Holistic Specification

Classical Specification

- fine-grained
- per function

Holistic Specification

- ADT as a whole
- emergent behaviour

Which is "stronger"?
"Closed" ADT with classical spec implies holistic spec.
(closed: no functions can be added, all functions have classical specs, ghost state has known representation)

Why do we need holistic specs?
* "closed ADT" is sometimes too strong a requirement.
* Holistic aspect is cross-cutting (eg no payment without authorization)
* Allows reasoning in open world (eg DOM wrappers)

Thank you