Trustworthy Global Computing 2014

An Information Flow Monitor For the DOM API

Introducing References and Live Primitives

José Fragoso Santos
Tamara Rezk
Inria

Ana Almeida Matos
Instituto Superior Técnico

Goals

- Enforce **Secure Information Flow** in a **DOM**-like API

Why?

JavaScript programs can encode illegal flows using the **DOM** API

Contributions

- Information flow monitor for a DOM-like API with **references**
- Simple language to reason about the DOM in which **nodes** are **first-class** values
- Enforcement of secure information flow even in the presence of **live collections**

Special kind of the data structure in the DOM API

Core DOM

$e ::= \mathbf{new}(e) \mid \mathbf{insert}(e_1, e_2, e_3) \mid \mathbf{remove}(e_1, e_2) \mid$

$\mid \mathbf{length}(e) \mid \mathbf{value}(e) \mid \mathbf{store}(e_1, e_2) \mid$

$\mid \mathbf{move}{\downarrow}(e_1, e_2) \mid \mathbf{move}{\uparrow}(e_2) \mid$

$\mid \mathbf{x} \mid e_1; e_2 \mid \mathbf{if}(e_0)\ \{\ e_1\ \}\ \mathbf{else}\ \{\ e_2\ \} \mid$

$\mid \mathbf{while}(e_0)\ \{\ e_1\ \}$

Core DOM

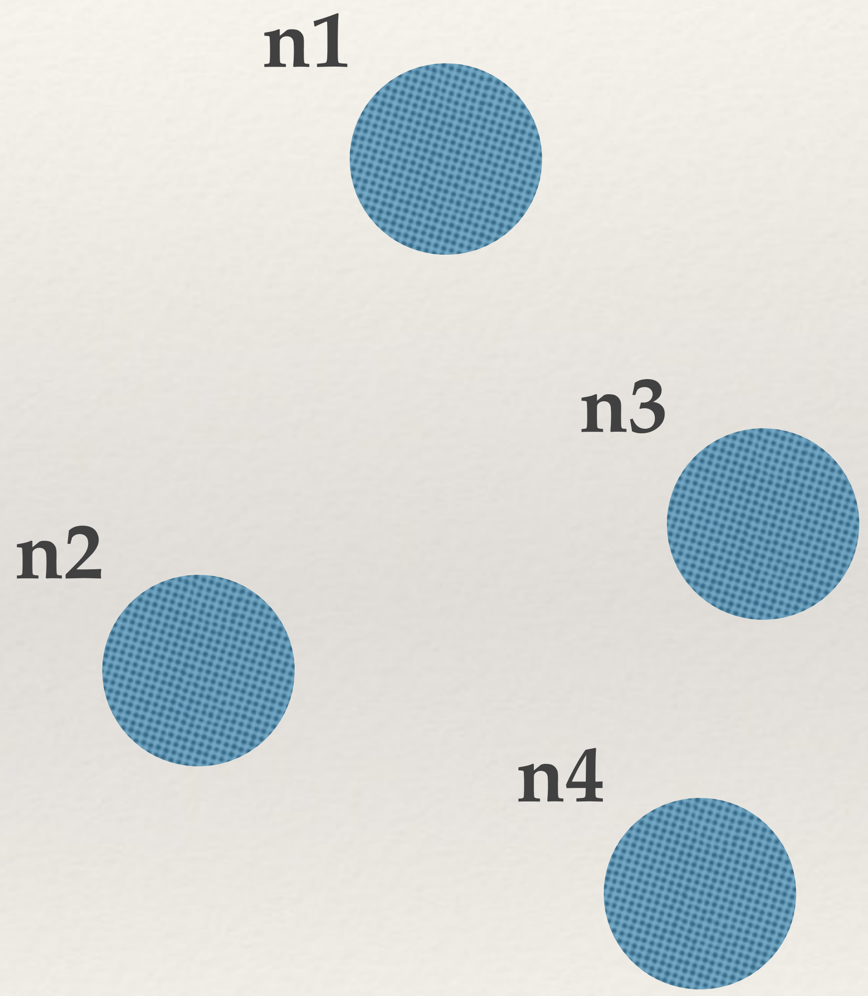
Create **New** Nodes

n1 := **new**("DIV");

n2 := **new**("DIV");

n3 := **new**("DIV");

n4 := **new**("DIV")



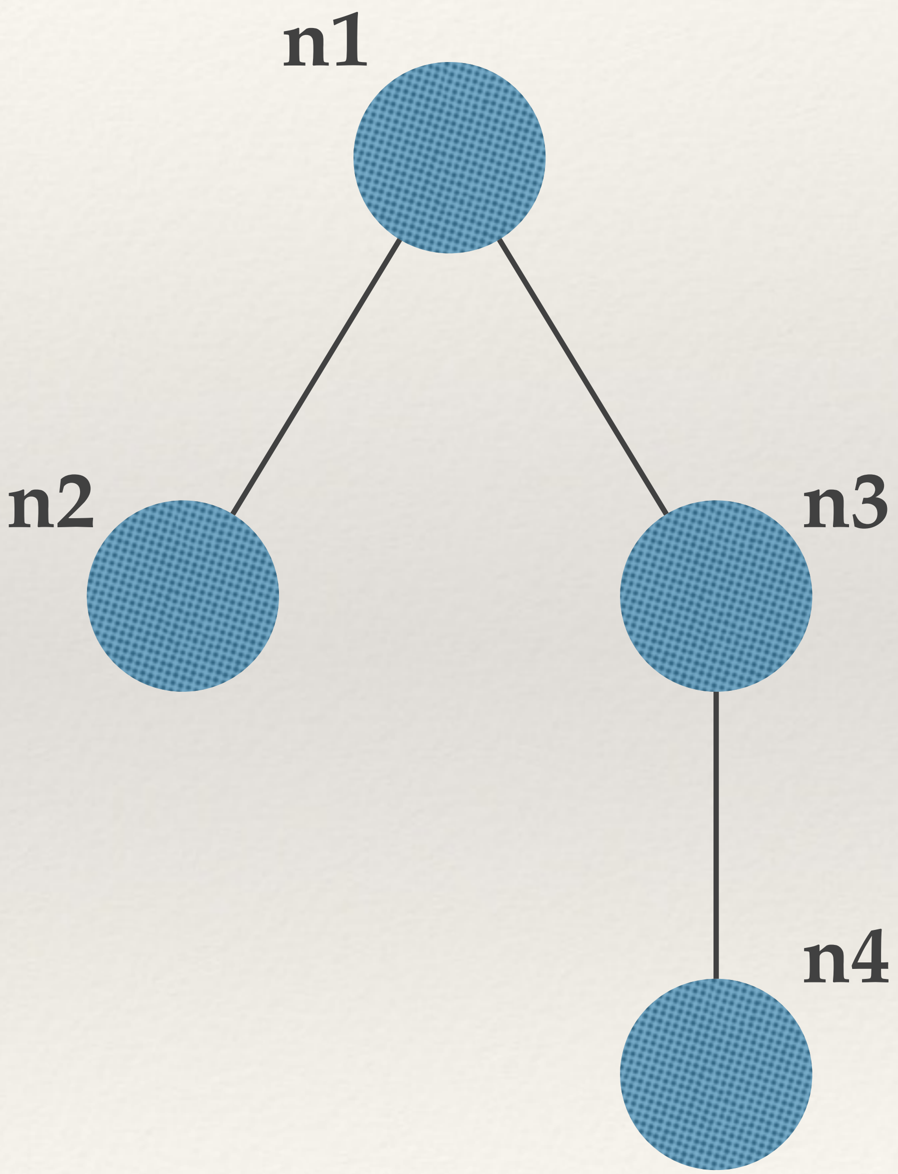
Core DOM

Inserting Nodes in Trees

insert(n1, n2, 0);

insert(n1, n3, 1);

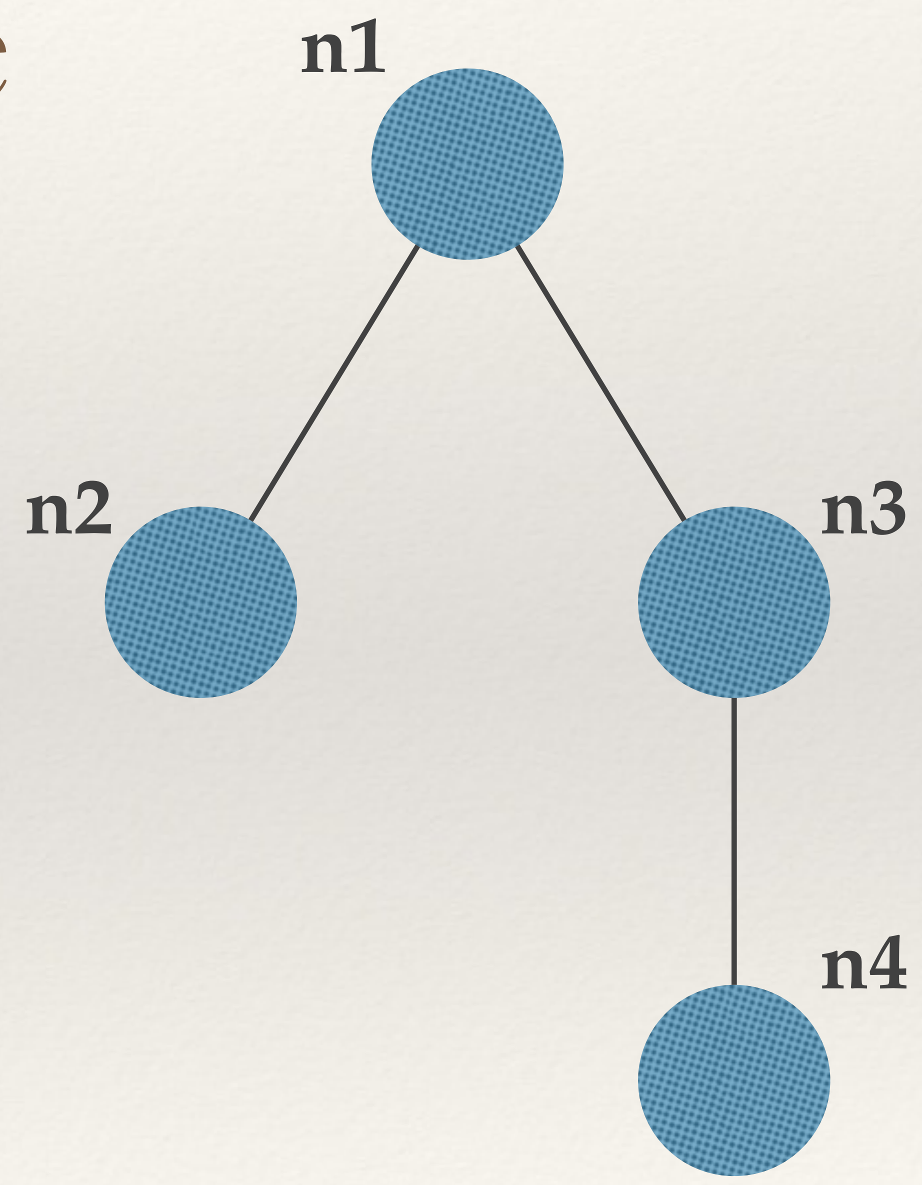
insert(n3, n4, 0);



Core DOM

Remove a node from a tree

remove(n1, n2);

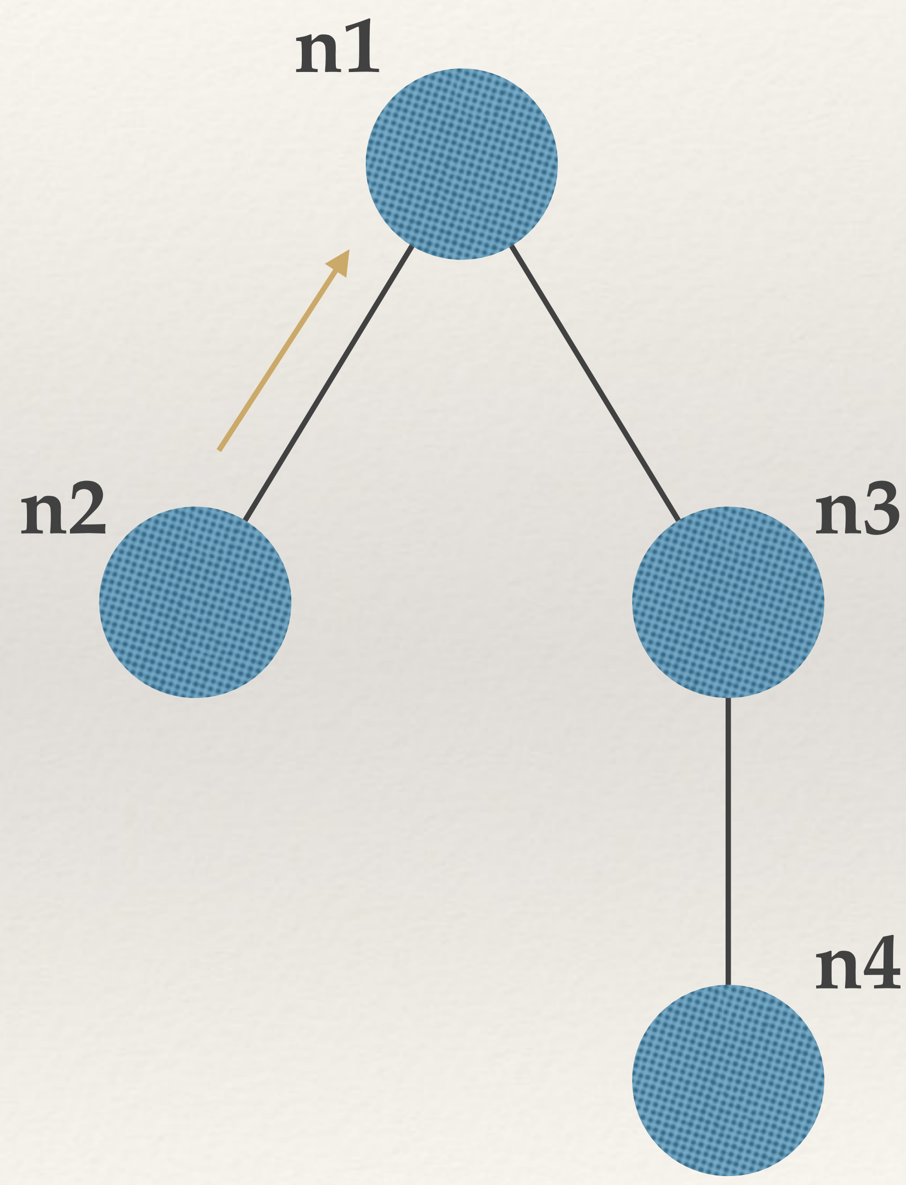


Core DOM

Get the **parent** of a node

x := **move**↑(n2);

$\boxed{x = n1}$

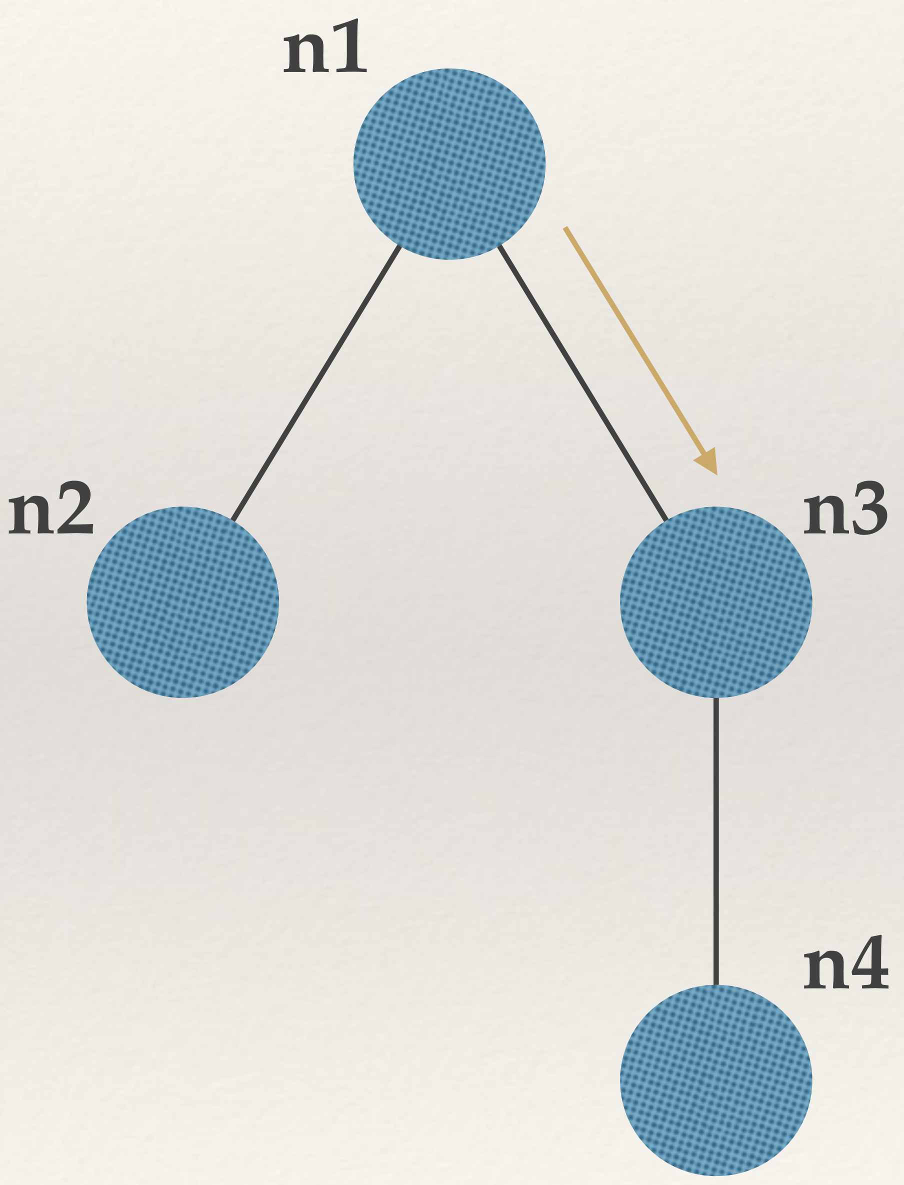


Core DOM

Get the i^{th} **child** of a node

x := **move**↓(n1, 1);

$\boxed{x = n3}$

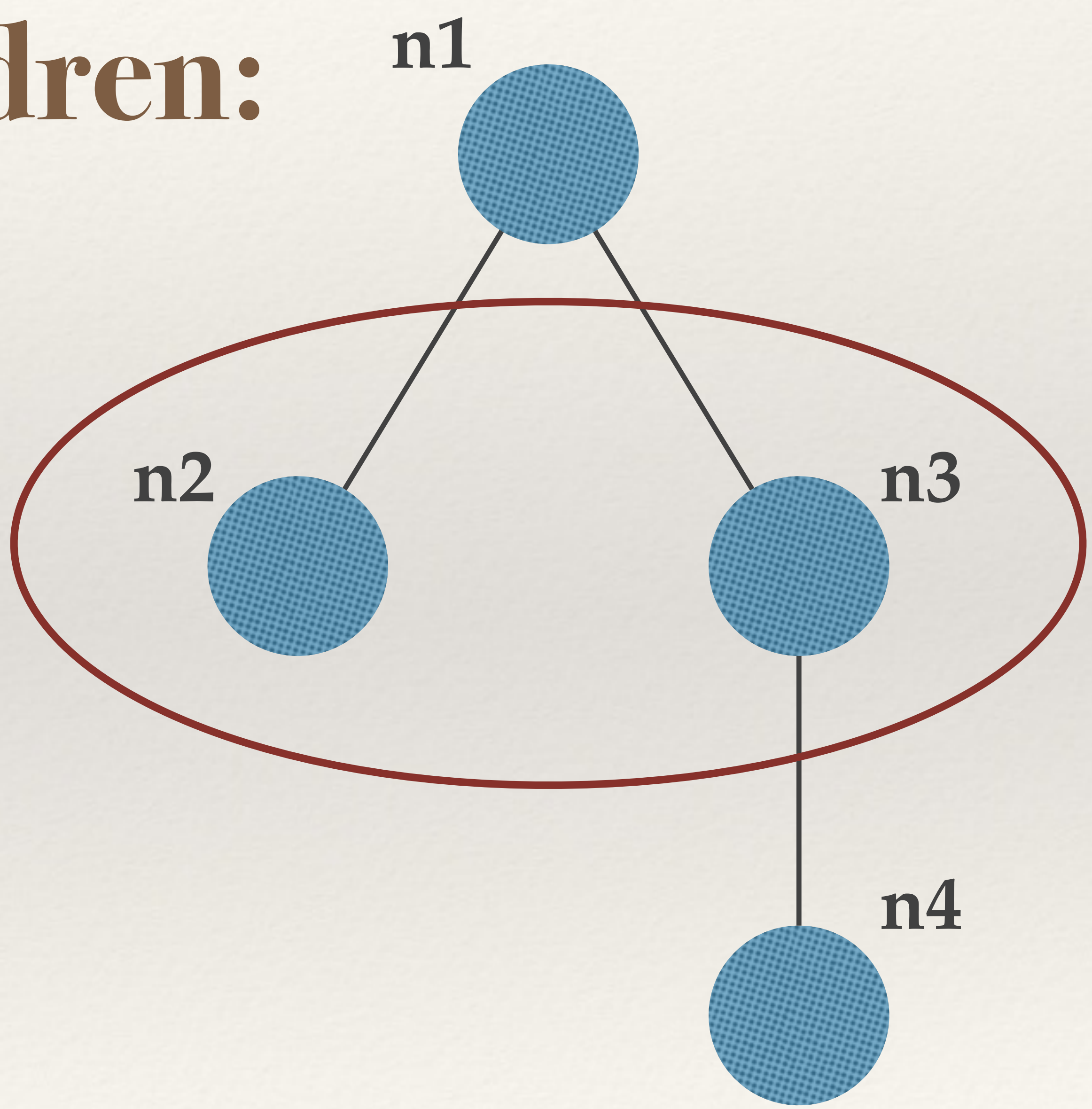


Core DOM

A node's **number of children:**

x := **length**(n1);

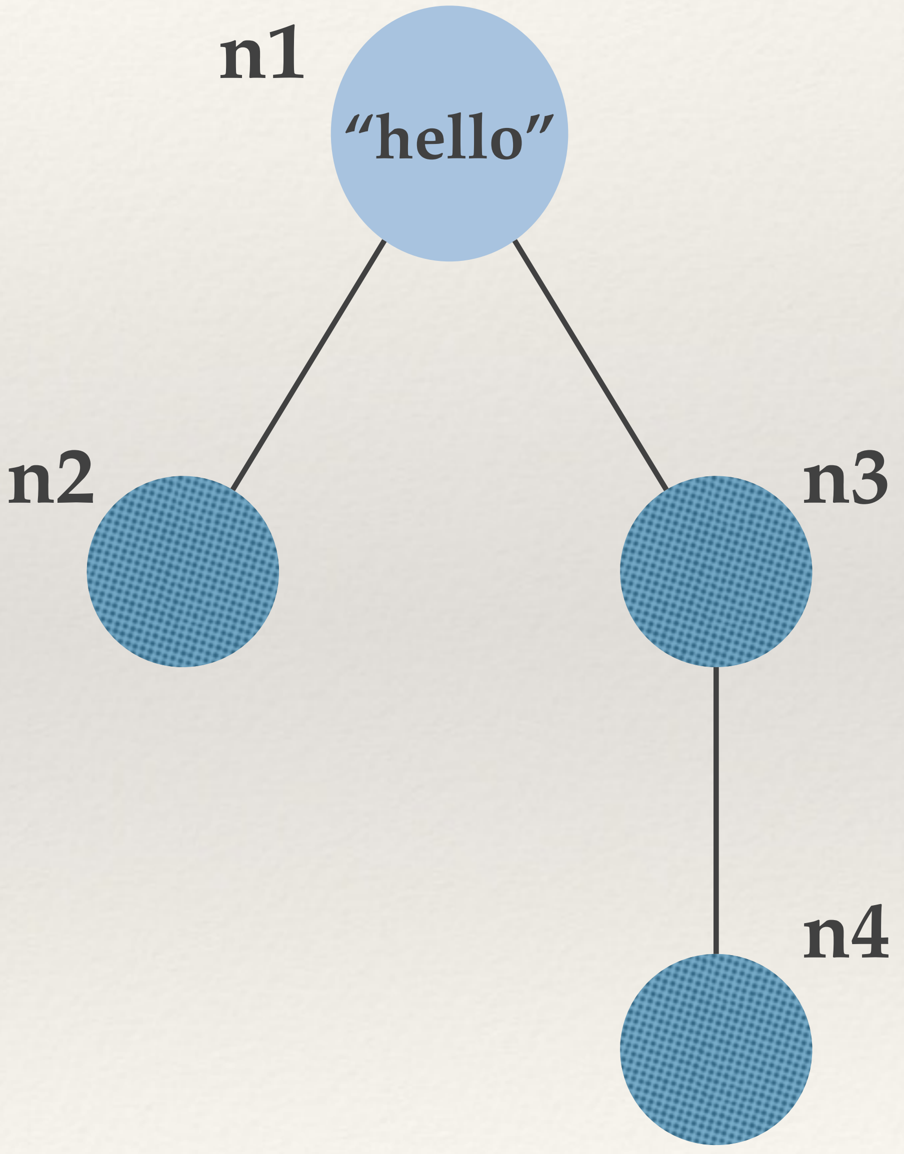
$\boxed{x = 2}$



Core DOM

Store a value in a node:

x := **store**(n1, "hello");

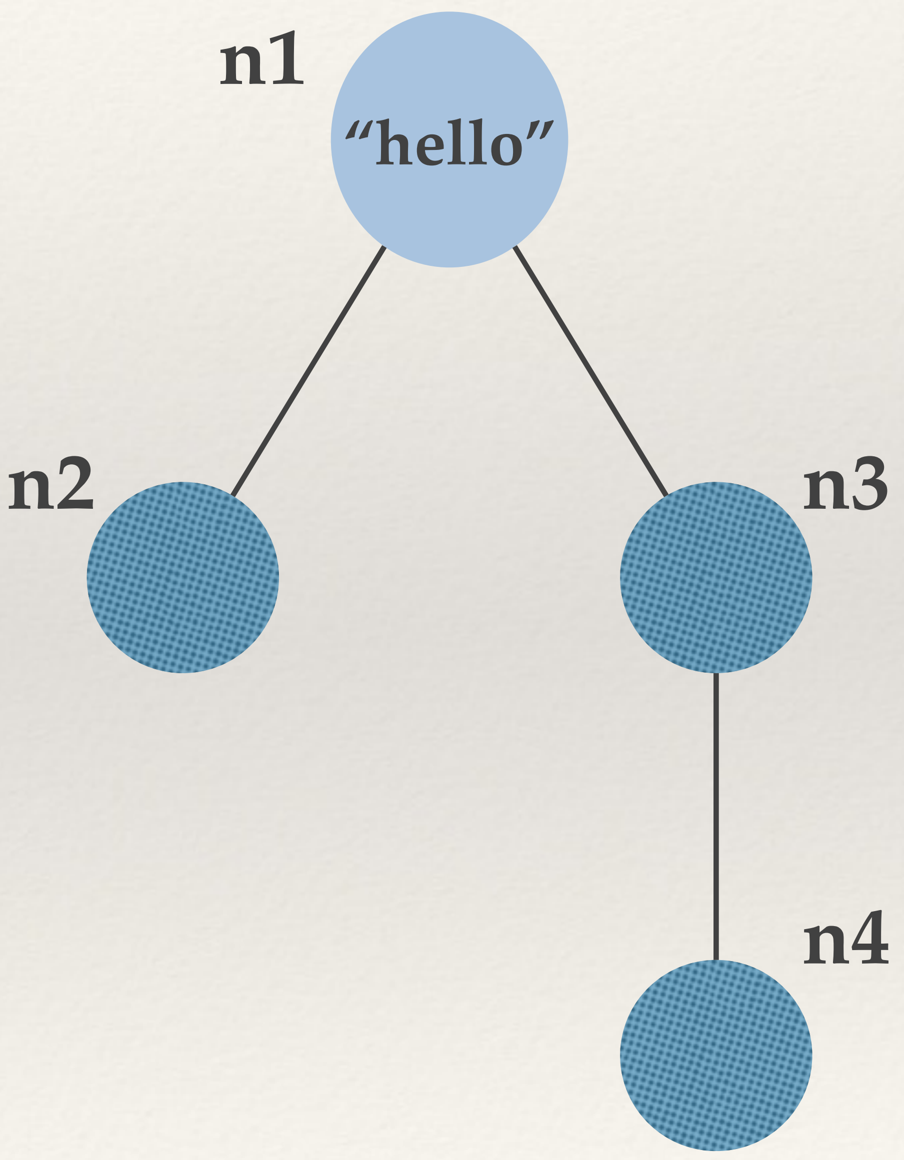


Core DOM

Read a **value** from a node:

x := **value**(n1);

x = "hello"



A Formal Semantics for Core DOM

$$\langle f, e \rangle \longrightarrow \langle f', e' \rangle$$

- **f** - initial Core DOM **forest**
- **e** - **expression** to evaluate
- **f′** - **forest** after the step
- **e′** - **expression** after the step

A Formal Semantics for Core DOM

A Core DOM forest:

$$\mathbf{f : Ref \longrightarrow Nodes}$$

$$f(r) = \langle tag, value, r, children \rangle$$

A Formal Semantics for Core DOM

A Core DOM forest:

$$\mathbf{f : Ref \longrightarrow Nodes}$$

f(r) = <**tag**, value, r, children>

tag name of the node,
e.g. DIV, SPAN, etc

A Formal Semantics for Core DOM

A Core DOM forest:

$$\mathbf{f : Ref \longrightarrow Nodes}$$

f(r) = <tag, **value**, r, children>

the **value** stored inside the node

A Formal Semantics for Core DOM

A Core DOM forest:

f : Ref ⟶ Nodes

f(r) = <tag, **value**, **r,** children>

a **reference** pointing to the node's **parent**

A Formal Semantics for Core DOM

A Core DOM forest:

$$\mathbf{f : Ref \longrightarrow Nodes}$$

$$f(r) = \langle tag, value, \mathbf{r}, \mathbf{children} \rangle$$

a list of references pointing to the children of the node

A Formal Semantics for Core DOM

Rule INSERT

r_1 is not an ancestor of r_0 $\quad\quad$ $|\, f(r_0.\text{children}) \,| \geq i$

r_1 is an orphan node $\Leftrightarrow f(r_1.\textbf{parent}) = \text{null}$

$f' = f\,[\; r_1.\textbf{parent} \mapsto r_0\;,\; r_0.\textbf{children} \mapsto \text{Shift}_R(r_0.\textbf{children}, i, r_1)\;]$

$\langle f, \textbf{insert}(r_0, r_1, i)\rangle \longrightarrow \langle f', r_1\rangle$

Information Components of a Node

- What can we **know** about a node?
- How can we **use the language** to learn it?

Existence	**Structure**
Value	**Position**

Information Components of a Node

Existence	Structure
Value	Position

Check whether a node **exists:**

```
l := 0;
n1 := null;
if (h) {
      n1 := new("DIV")
}
if (n1) {
      l := 1
}
```

$$\mathbf{h = 0} \;\longrightarrow\; \mathbf{l = 0}$$

$$\mathbf{h = 1} \;\longrightarrow\; \mathbf{l = 0}$$

Information Components of a Node

Existence	Structure
Value	**Position**

Check "who" is the **parent** of a given node:

```
if (h) {
    insert(n1, n3, 0)
}
if (n1) {
    insert(n2, n3, 0)
}
l := move↑(n3);
```

$$\mathbf{h = 0} \;\longrightarrow\; \mathbf{l = n1}$$

$$\mathbf{h = 1} \;\longrightarrow\; \mathbf{l = n2}$$

Information Components of a Node

Existence	Structure
Value	**Position**

Check "who" is the **i^{th} child** of a given node:

```
insert(n1, n2, 0)
if (h) {
    insert(n1, n3, 0)
}
l := move↓(n1, 0)
```

h = 0 → l = **n2**

h = 1 → l = **n3**

Information Components of a Node

Existence	Structure
Value	Position

Check the **value** stored in a node:

```
if (h) {
    store(n1, 1)
} else {
    store(n1, 0)
}
l := value(n1)
```

$$\mathbf{h = 0} \longrightarrow \mathbf{l = 0}$$

$$\mathbf{h = 1} \longrightarrow \mathbf{l = 1}$$

Information Components of a Node

Existence	Structure
Value	Position

Check the **number of children** of a node:

```
if (h) {
    store(n1, n2)
}
l := length(n1)
```

$$\mathbf{h = 0} \longrightarrow l = 0$$

$$\mathbf{h = 1} \longrightarrow l = 1$$

Information Components of a Node

Existence	Structure
Value	**Position**

Position = Index + Parent

Index:

Position a node a node occupies in the list of child of children of its parent

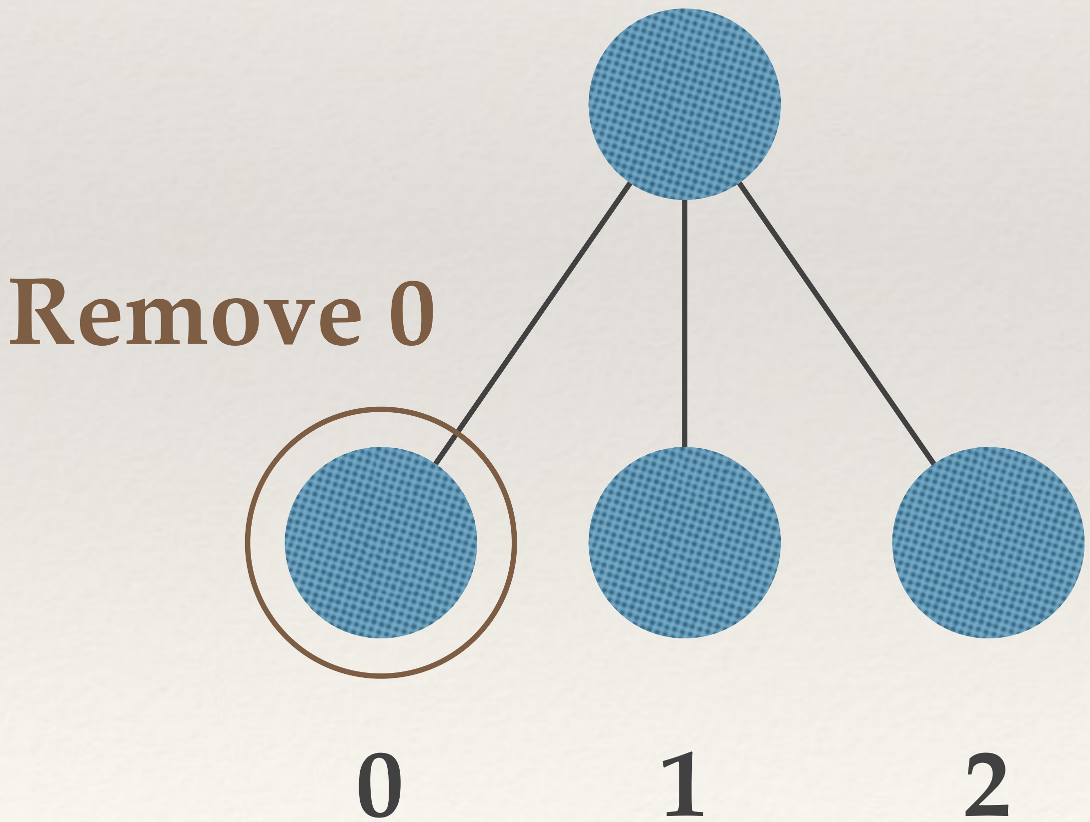
Order Leaks

- New kind of implicit flow
- Cannot be directly expressed in previous models

Changing the position of a node changes the position of its right siblings

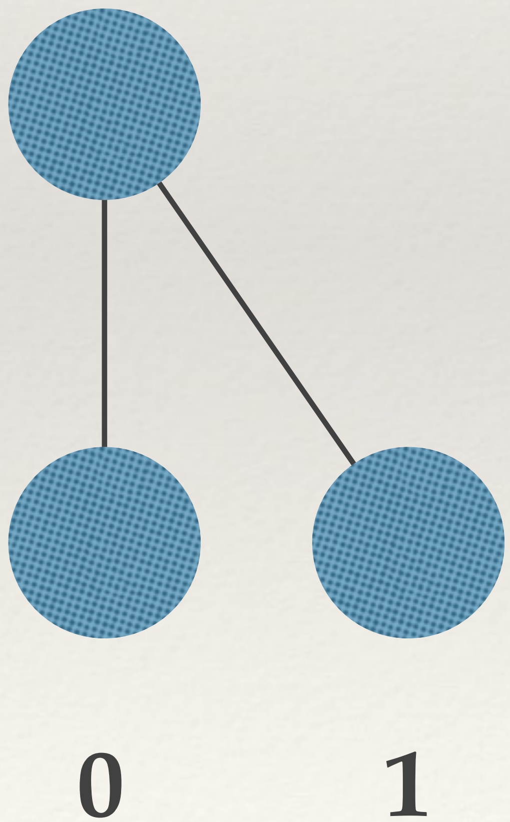
Order Leaks

Changing the position of a node changes the position of its right siblings



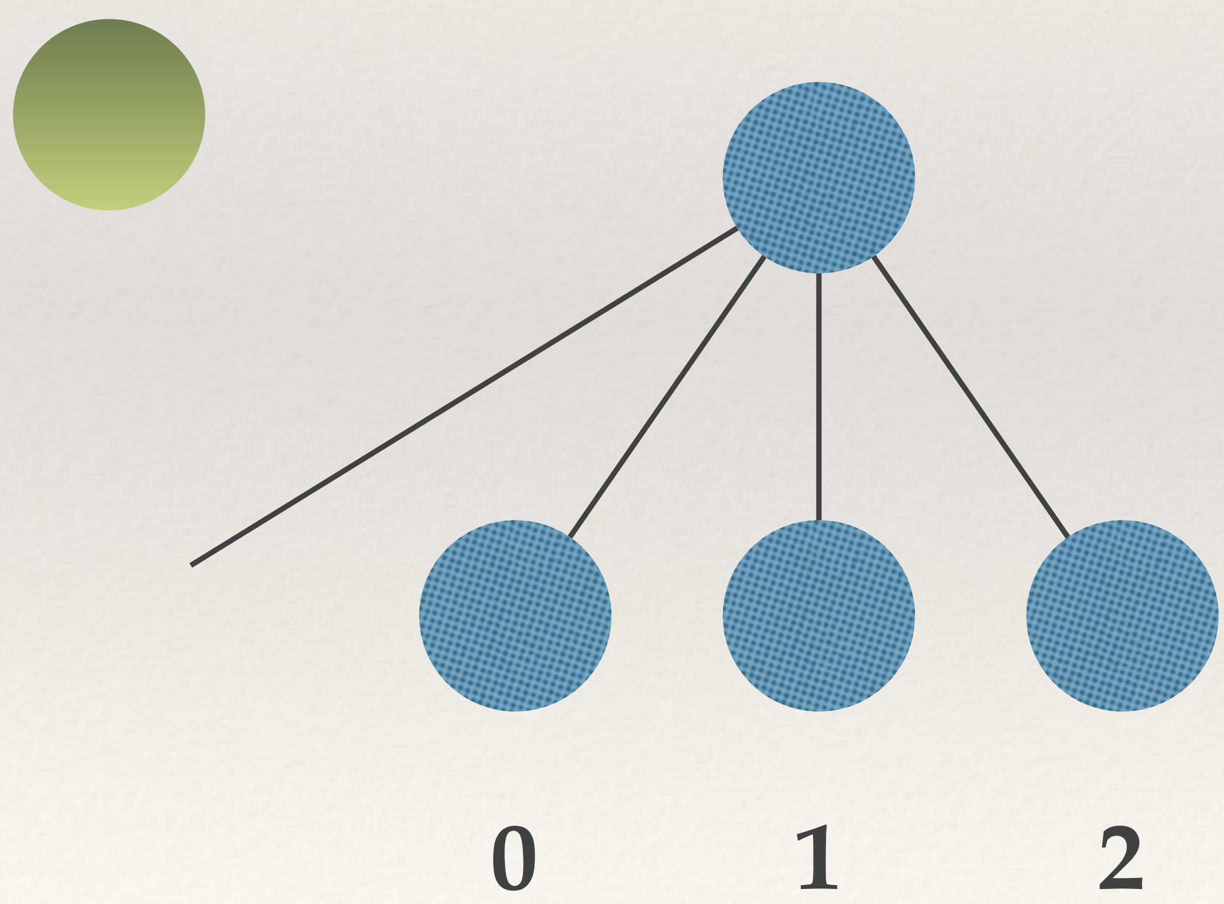
Order Leaks

Changing the position of a node changes the position of its right siblings



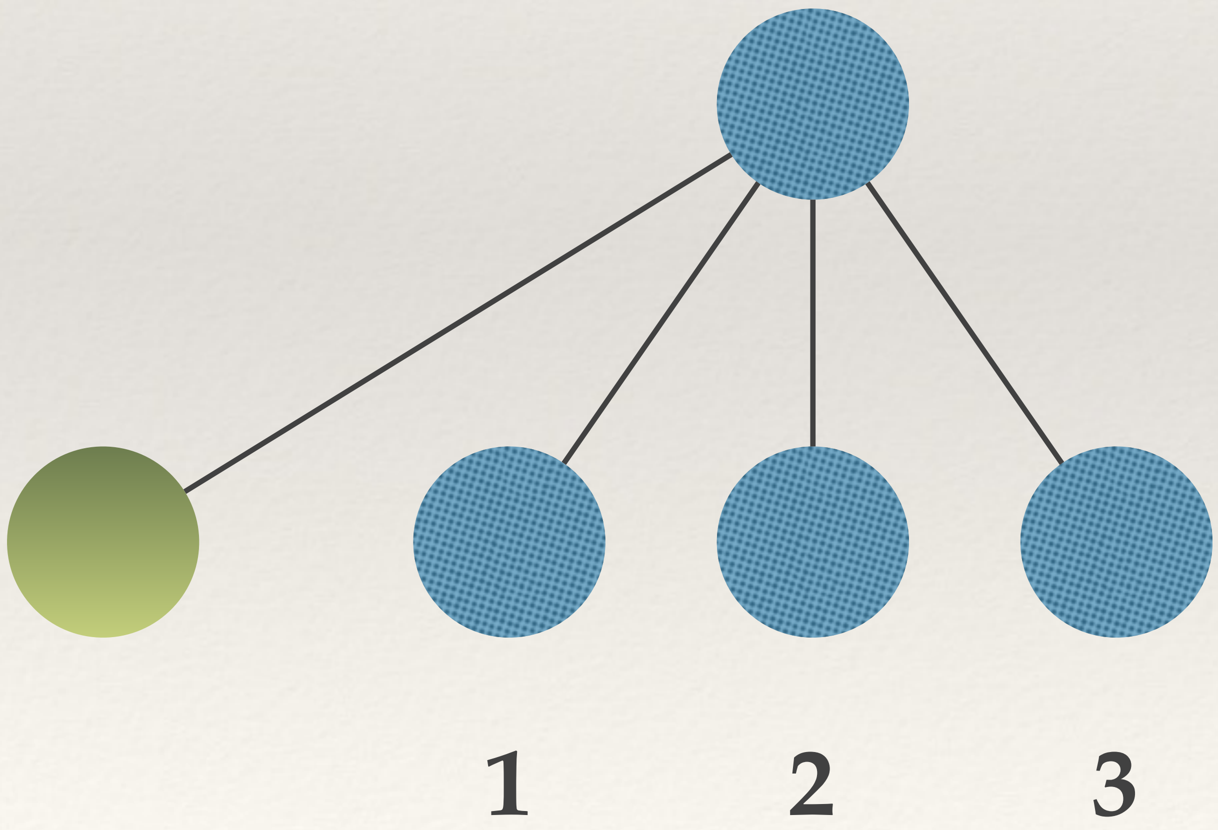
Order Leaks

Changing the position of a node changes the position of its right siblings



Order Leaks

Changing the position of a node changes the position of its right siblings



Labeling DOM Forests

For each DOM node:

- Node Level
- Position Level
- Structure Security Level
- Value Level

$$\Sigma : \textbf{Ref} \longrightarrow \textbf{L}^4$$

$$\Sigma(\mathbf{r}) = \langle \sigma_0, \sigma_1, \sigma_2, \sigma_3 \rangle$$

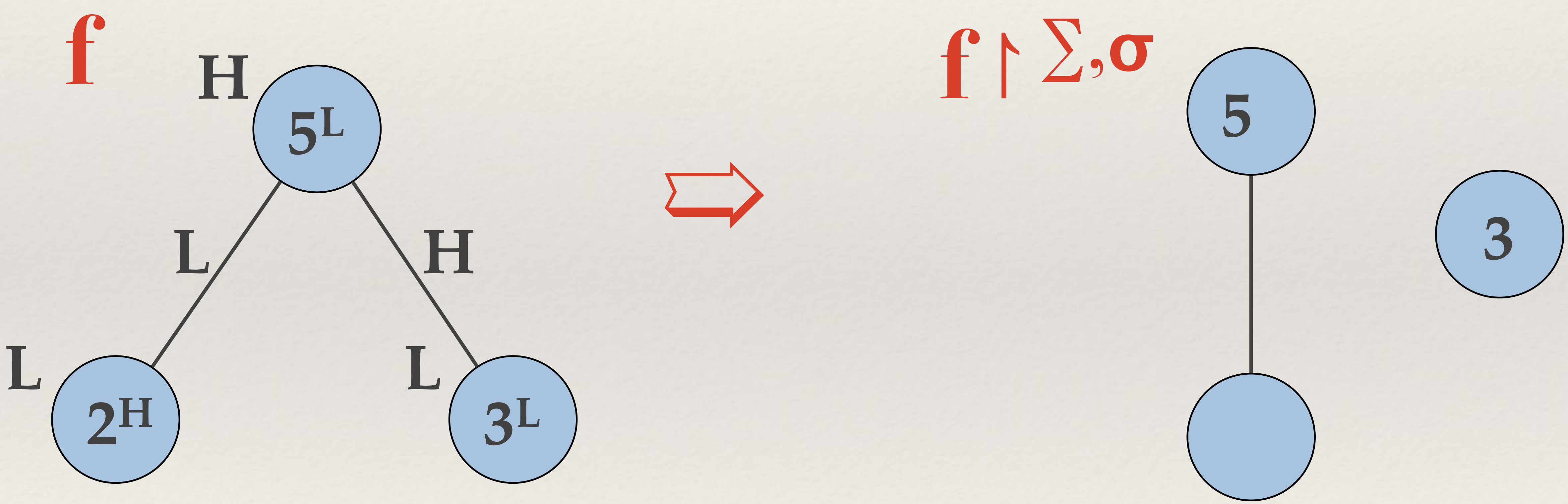
Indistinguishable DOM Forests

What can an attacker see at level σ?

- The existence of nodes whose existence levels are ≦ **σ**
- The positions of nodes whose position levels are ≦ **σ**
- The number of children of nodes whose structure security level are ≦ **σ**
- The values stored in nodes whose value levels are ≦ **σ**

Indistinguishable DOM Forests

What can an attacker see at level σ?



Monitoring Information Flow

- **Flow-Sensitive**
- **Purely Dynamic**
- **No-sensitive-upgrade Discipline**

No-sensitive Upgrages

Implicit Flows are **BLOCKED** by the monitor

Position Level

Structure Security Level

Flow-Insensitive

Value Level

Flow Sensitive

Monitor Transitions

α

$\langle f, e \rangle$

$$\langle \Sigma, \boldsymbol{\sigma}_0 :: \ldots :: \boldsymbol{\sigma}_n \rangle \xrightarrow{\alpha} \langle \Sigma, \boldsymbol{\sigma} \rangle$$

- Σ **and** Σ' - initial and final **labelings**
- $\sigma_0, \ldots, \sigma_n$ - levels of the **subexpressions**
- σ - **reading effect**

Monitor Transitions

Rule INSERT

$\sum(r_1).pos \geq$ Position Level of New Left Sibling

$$\sigma_0 \sqcup \sigma_1 \sqcup \sigma_2 \;\leq\; \sum(r_1).pos$$

$$\sigma_0 \sqcup \sigma_1 \sqcup \sigma_2 \;\leq\; \sum(r_0).struct$$

$$<\sum, \sigma_0 :: \sigma_1 :: \sigma_2> \quad \xrightarrow{<r_0,\, r_1>} \quad <\sum, \sigma_2>

Constraints - Summary

A Program CANNOT:

Change the position of nodes with visible positions in invisible contexts

Change the number of children of a node with a visible number of children in an invisible context

Noninterference

$$
\begin{tikzcd}
f_0, \Sigma_0 \arrow[d, "\mathbf{Prog}"] & \sim_\sigma & f_1, \Sigma_1 \arrow[d, "\mathbf{Prog}"] \\
f_0', \Sigma_0' & \sim_\sigma & f_1', \Sigma_1'
\end{tikzcd}
$$

Live Collections

A special kind of DATA STRUCTURE that automatically reflects modifications to the document

```
divs = document.getElementsByTagName("DIV");
i = 0;
while(i <= divs.length){
  document.appendChild(document.createElement("DIV"));
  i++;
}
```

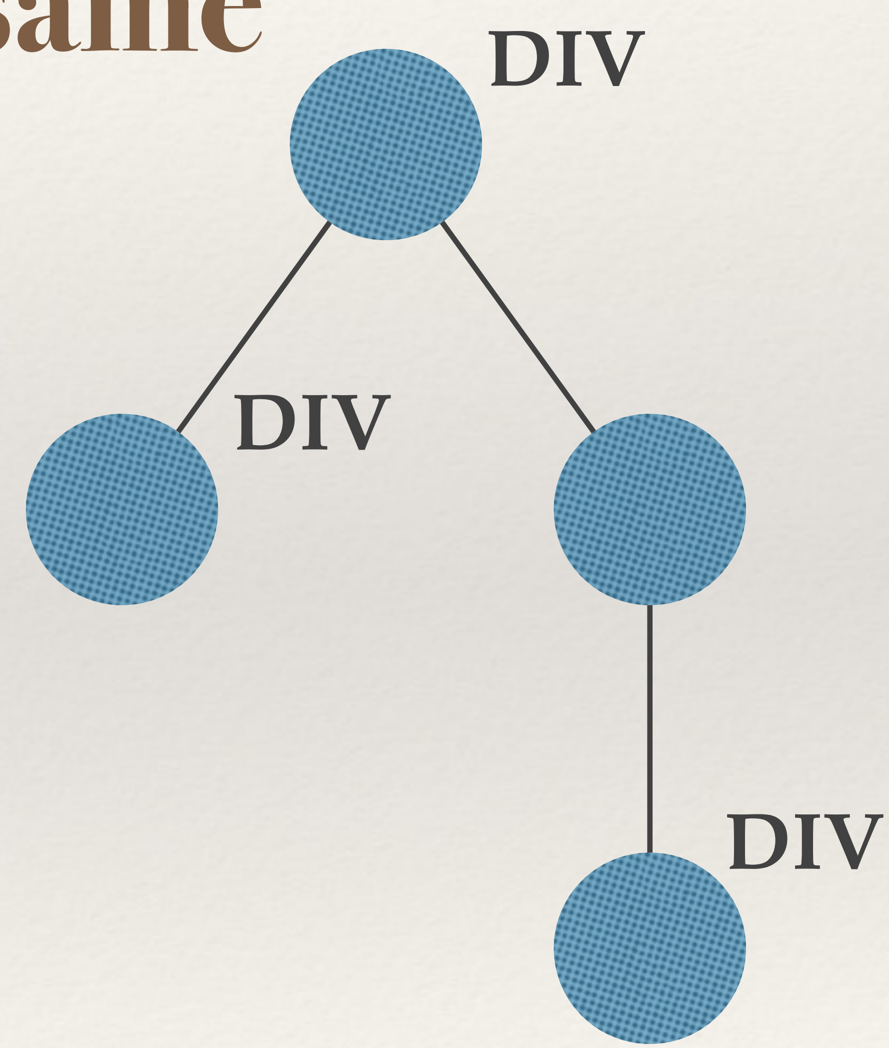
Infinite Loop

Modeling live collections

Number of nodes with the **same tag** in the **same tree**

$x := \mathbf{length}_{\lightning}(n1, \text{“DIV”});$

$\boxed{x = 3}$

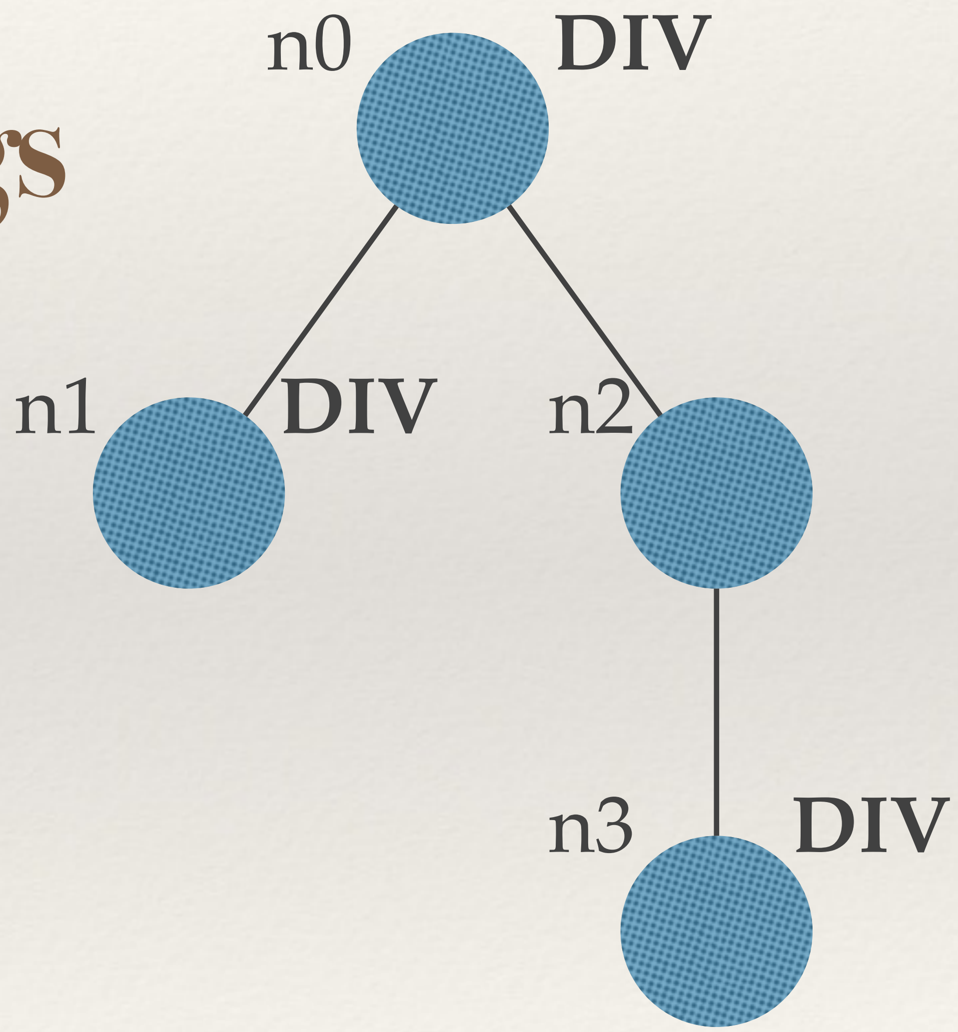


Modeling live collections

Nodes with the **same tag**
in the **same tree** as **siblings**

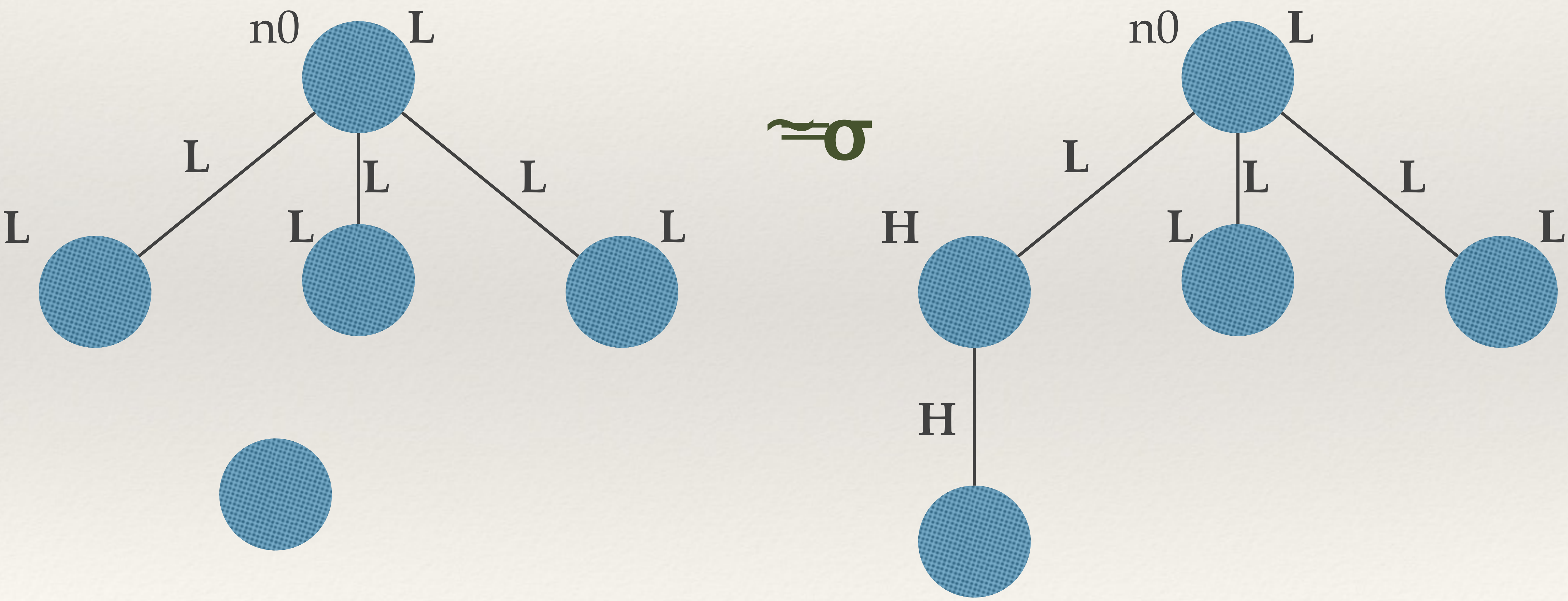
$x := \mathbf{move}_{\lightning}(n0, 2);$

$\boxed{x = n3}$



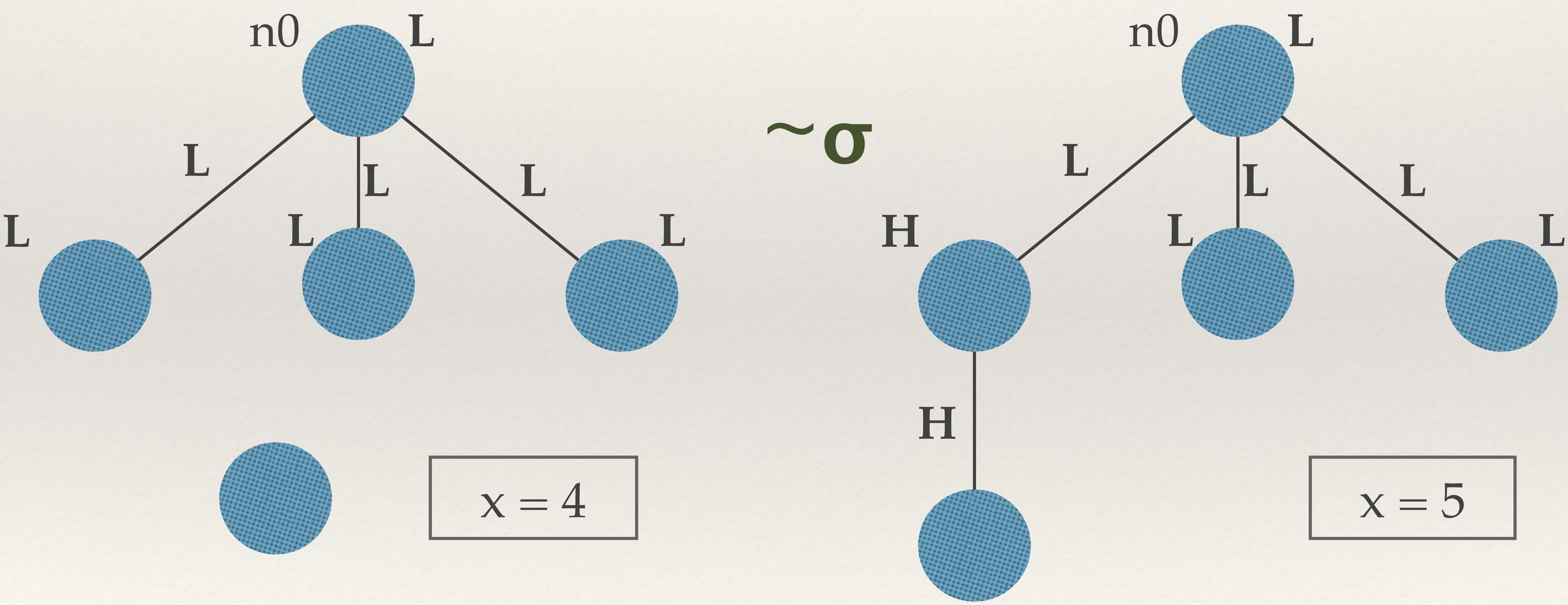
New Leaks

All empty divs



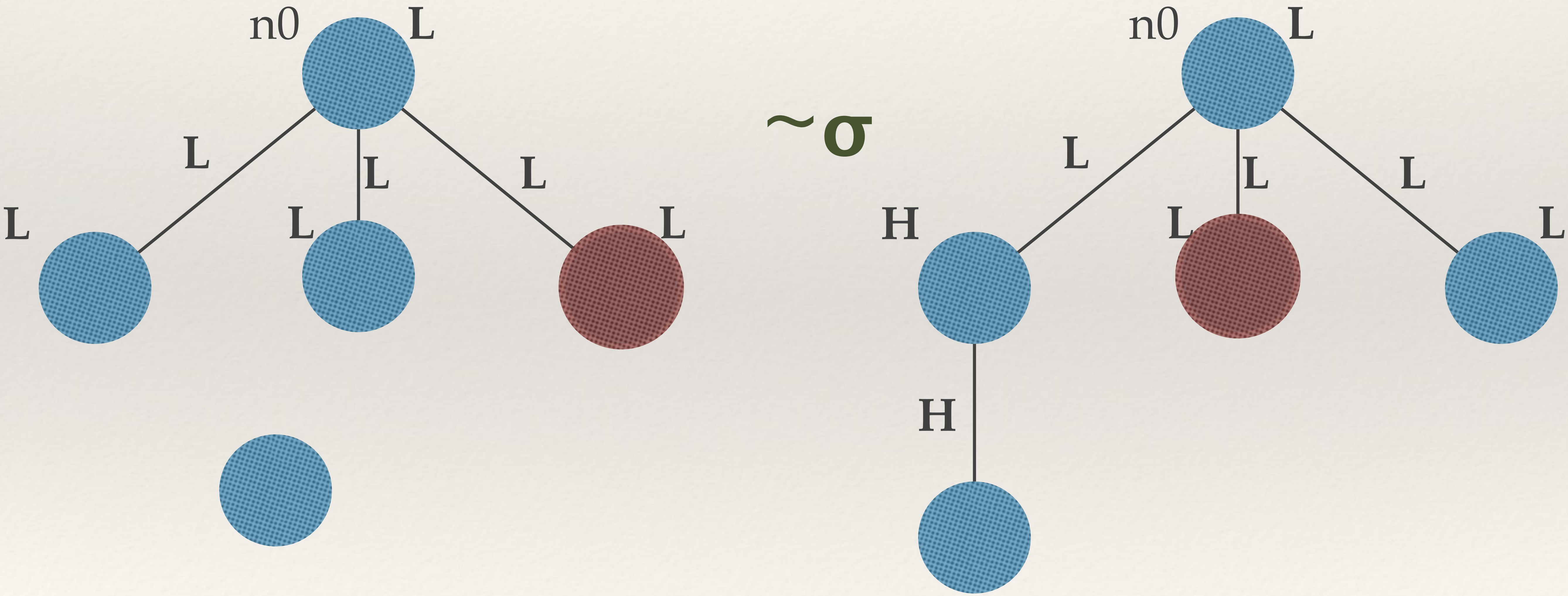
New Leaks

x := **length**$_{\lightning}$(n0, “DIV”);



New Leaks

x := **move**$_{\lightning}$(n0, “DIV”, 3);



New Leaks

Live collections **increase** the **observational power** of an attacker

The **low-equality** $\sim_\sigma$ does **not** work any more

Indistinguishable DOM Forests

What can an attacker see at level σ when using live collections?

- **Live Index of a node** - the position that the node occupies in the list containing all the other nodes in the tree with the same tag in **document order**
- **Positon = Parent + Index + Live Index**

Indistinguishable DOM Forests

What can an attacker see at level σ when using live collections?

- Global Position Level = Tag Level => upper bound on the levels of the contexts in which one can change the position of a node with tag TAG

Indistinguishable DOM Forests

What can an attacker see at level σ when using live collections?

- **Live Indexes of the nodes with position level ≦ σ**
- **The number of descendants of every node with tag TAG, provided that the tag level is ≦ σ**

Well-Labeled Forests

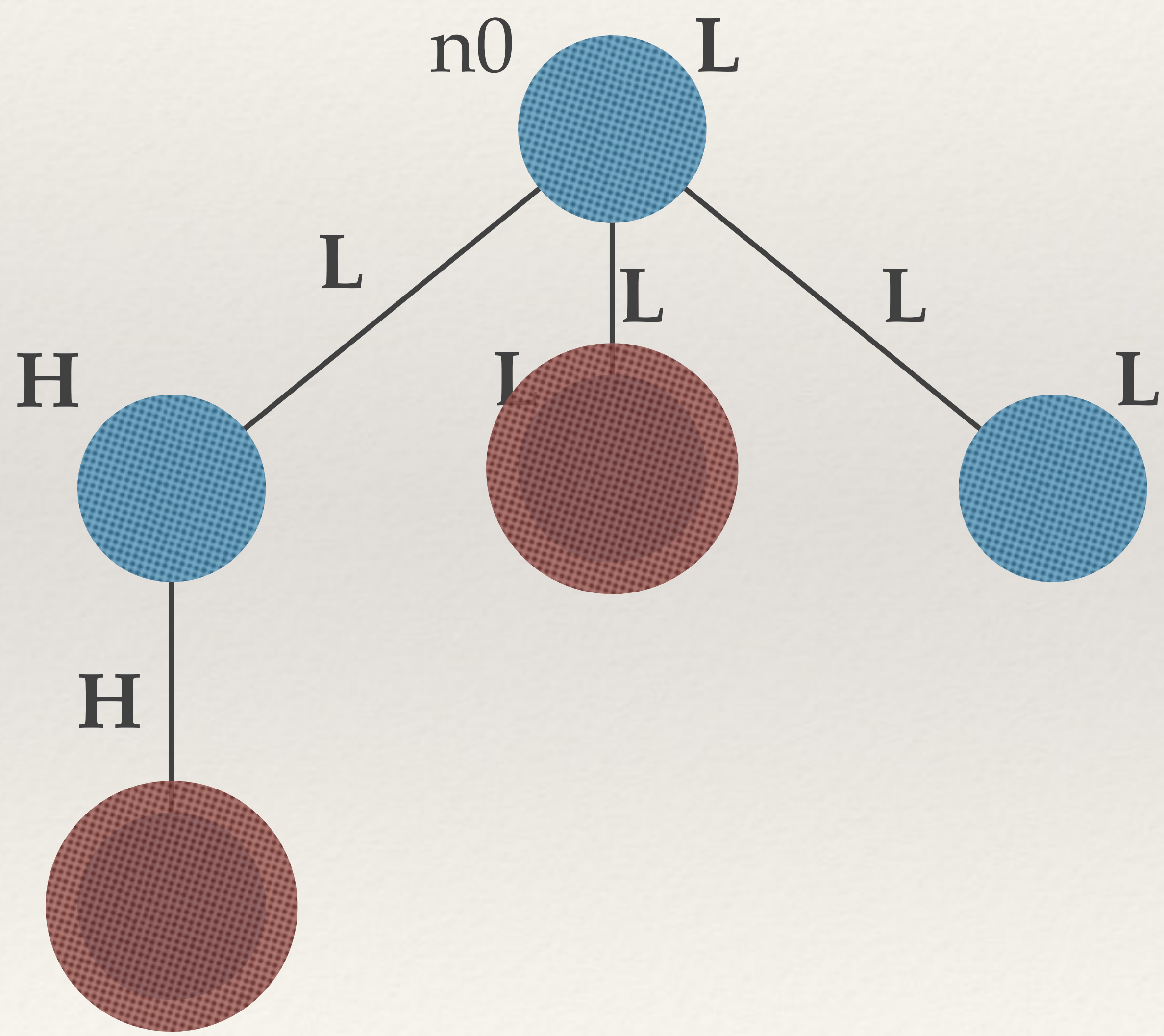
$$\mathrm{WL}_{\lightning}(f, \Sigma)$$

- Position levels are increasing in document order
- The position level of every node is ≦ the tag level of its tag

Well-Labeled Forests

All empty divs

$\neg \mathrm{WL}_{\lightning}(f, \Sigma)$



Enforcement

Block-on-read instead of **Block-on-Write**

Block the execution when trying to use a live construct and the forest is **not well-labeled**

Summary

A **flow-sensitive** monitor for securing information in a DOM-like language

- References => Nodes as values
- Live Collections

Main References

Russo and Sabelfed. **Tracking Information Flow in Dynamic Tree Structures.** ESORICS 2009.

Gardner and Smith and Wheelhouse and Zarfaty. **DOM Towards a Formal Specification.** Plan-X 2008.

Thank you

Questions…