Instituto Superior Técnico, 1st December 2014

Securing Client Side Web Applications

José Fragoso Santos
PhD Candidate
Inria
Team **INDES**

Problem

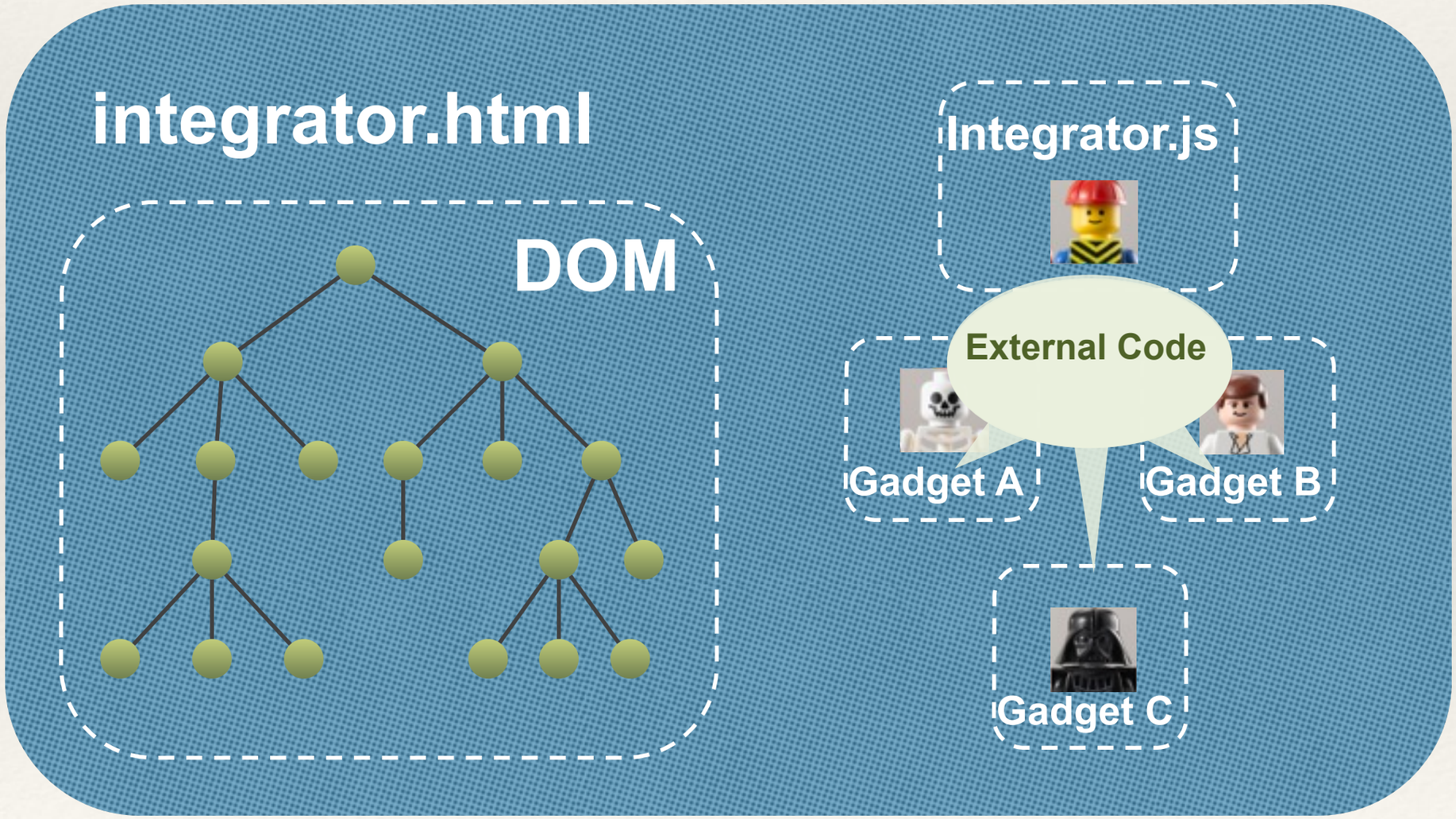
- **Client Side Web Applications:**
  - Combine data/code from **different origins** to create a new services

Problem

- **Client Side Web Applications:**
  - Combine data/code from **different origins** to create a new services
- Are implemented in **JavaScript**

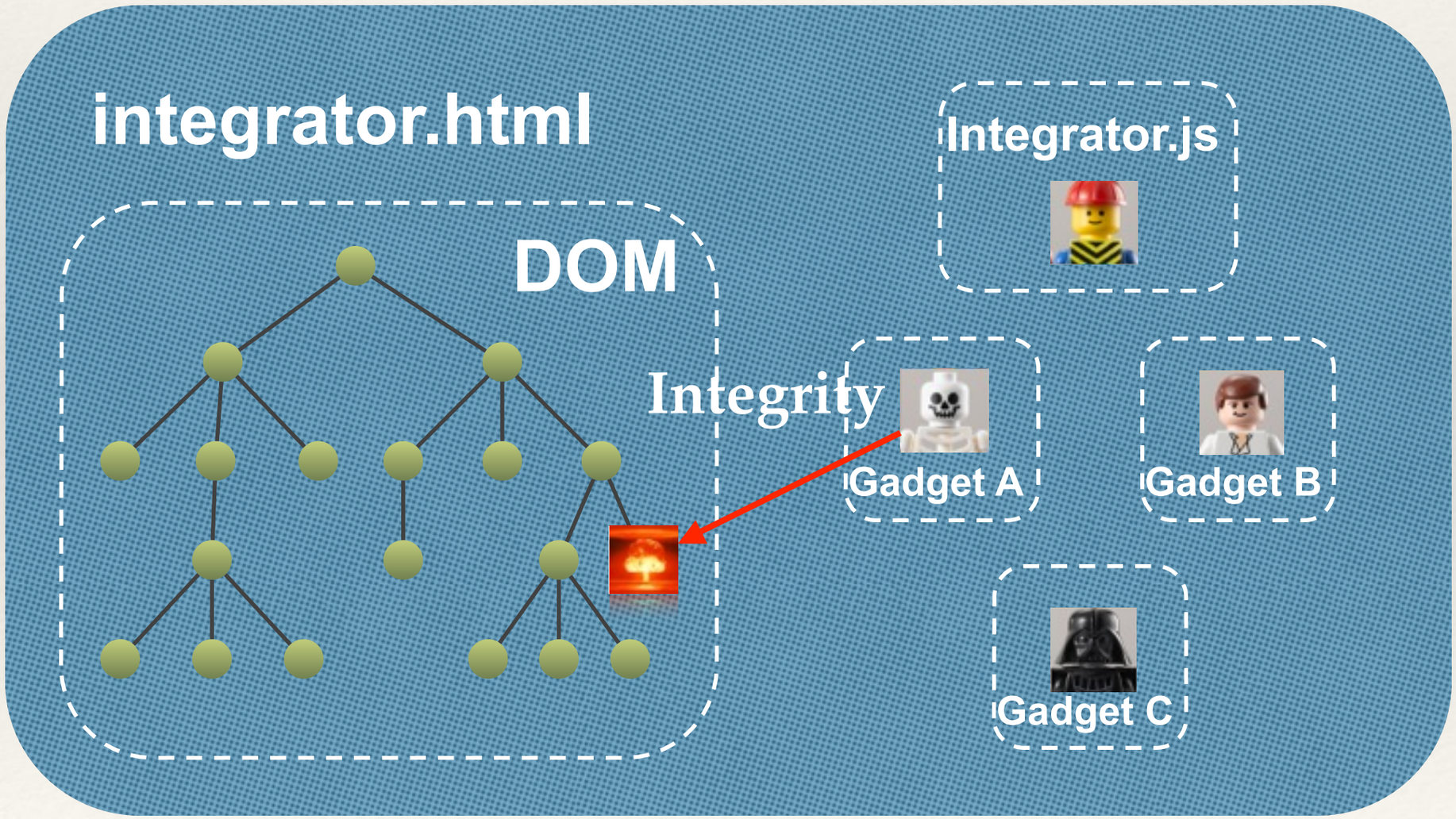
Problem

Data and **code** from **different origins**



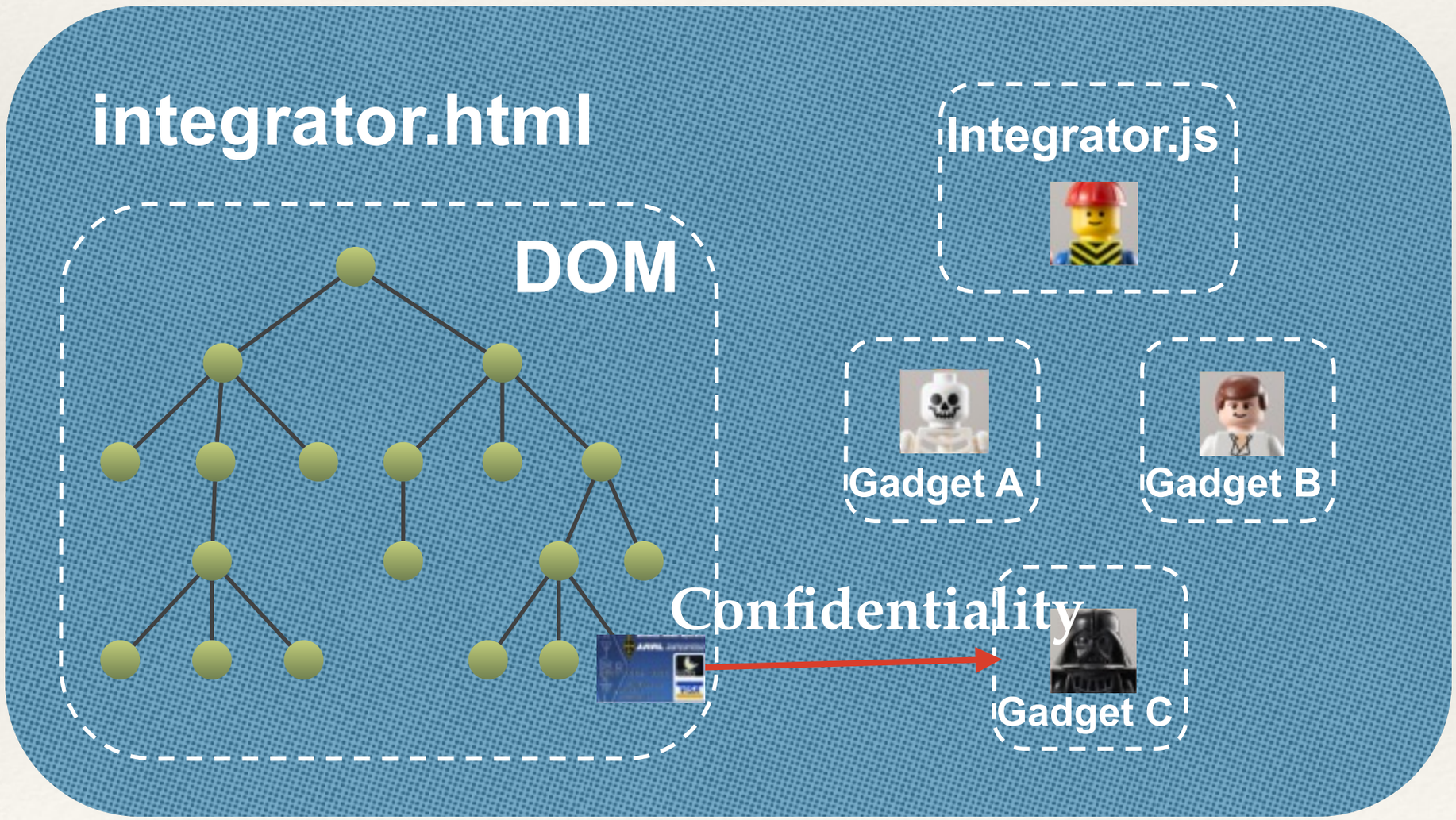
Problem

Untrusted scripts can modify **high integrity** content



Problem

Untrusted scripts can read **high confidentiality** content



Concrete Threats in the Browser

Common Threats in the Browser

- **Cookie Stealing**: External scripts have unlimited access to the `cookie` property of the `window` object
- **Location Hijacking:** External scripts can either influence the document's location directly or the string variables that are forming a URL
- **History Sniffing:** Malicious scripts can check whether the user has ever visited a specific URL
- **Behaviour Tracking:** Malicious scripts can gather precise information about the user's mouse clicks and movements, scrolling behaviour

Cookie Stealing

- External scripts have unlimited access to the `cookie` property of the `window` object
- External scripts can subsequently **transmit** the value of the cookies to an arbitrary **remote website**

```
document.cookie = “cookieval=" +
   encodeURIComponent(val) + “;”;
```

Location Hijacking

- External scripts can either influence the document's location directly or the string variables that are forming a URL
- External scripts can navigate the page to a malicious website without the knowledge of the user.

```
document.location // document.URL
window.history
```

history.back()
history.forward()
history.go()

History Sniffing

- Malicious scripts can check whether the user has ever visited a specific URL
- JavaScript code can create an invisible link to the target URL and then use the browser's interface to check how this link is displayed. TRICK: the browser displays visited and unvisited links in different colours

```
$("#link1").css("color");
```

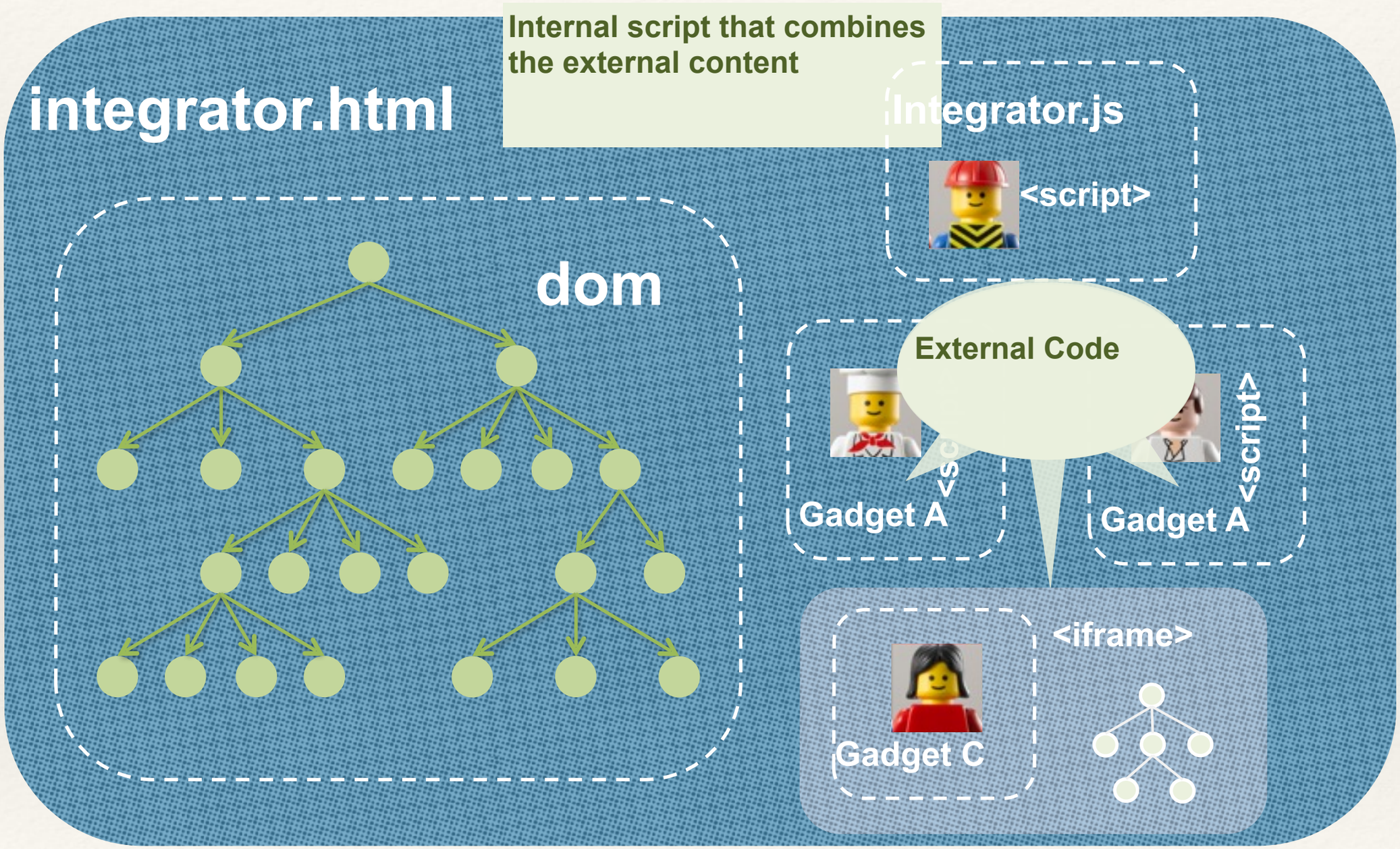
Behaviour Tracking

- Malicious scripts can gather precise information about the user's mouse clicks and movements, scrolling behaviour

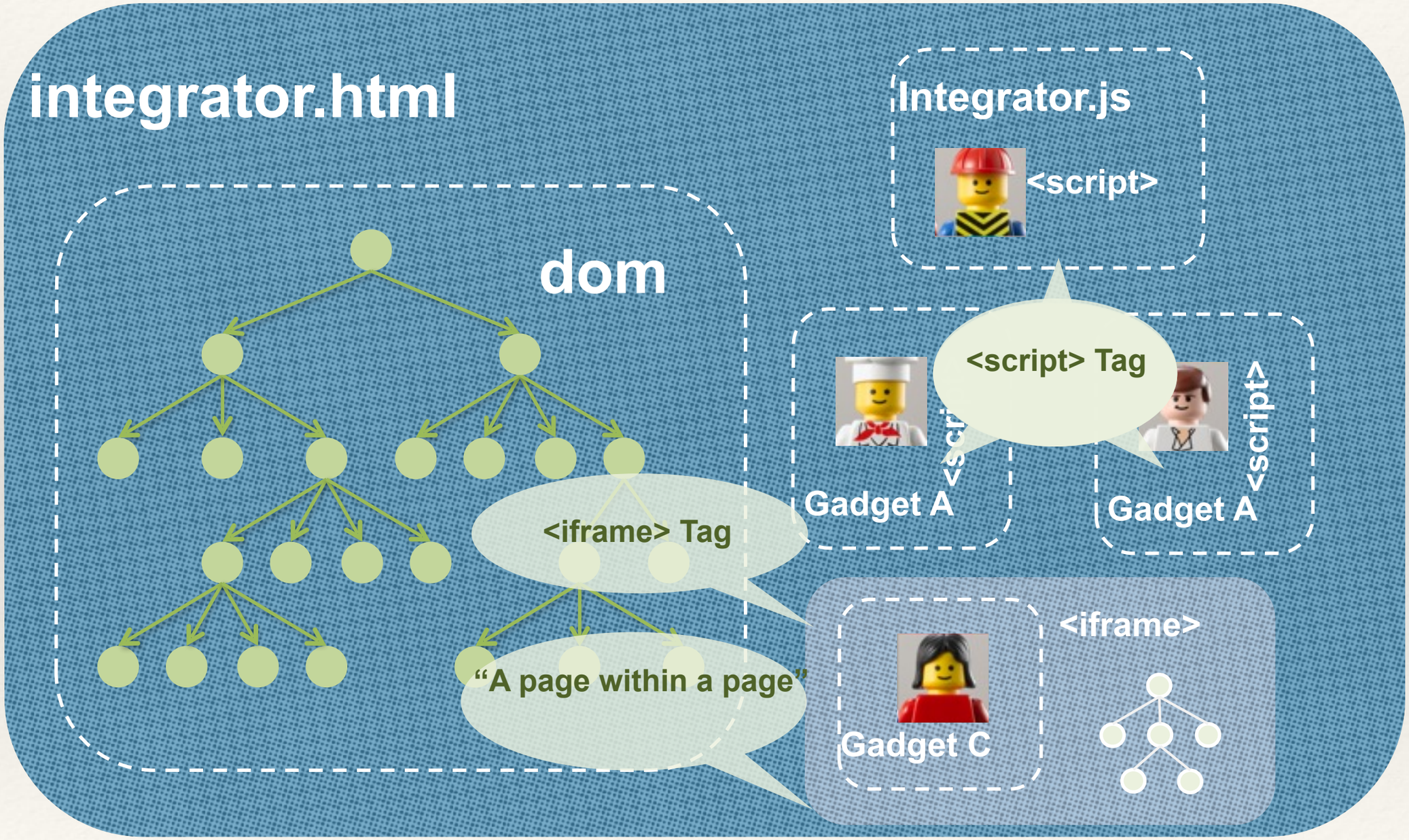
```
mydiv.addEventListener(“mousedown",
    handleMouseDown, true);
```

The Browser Security Model

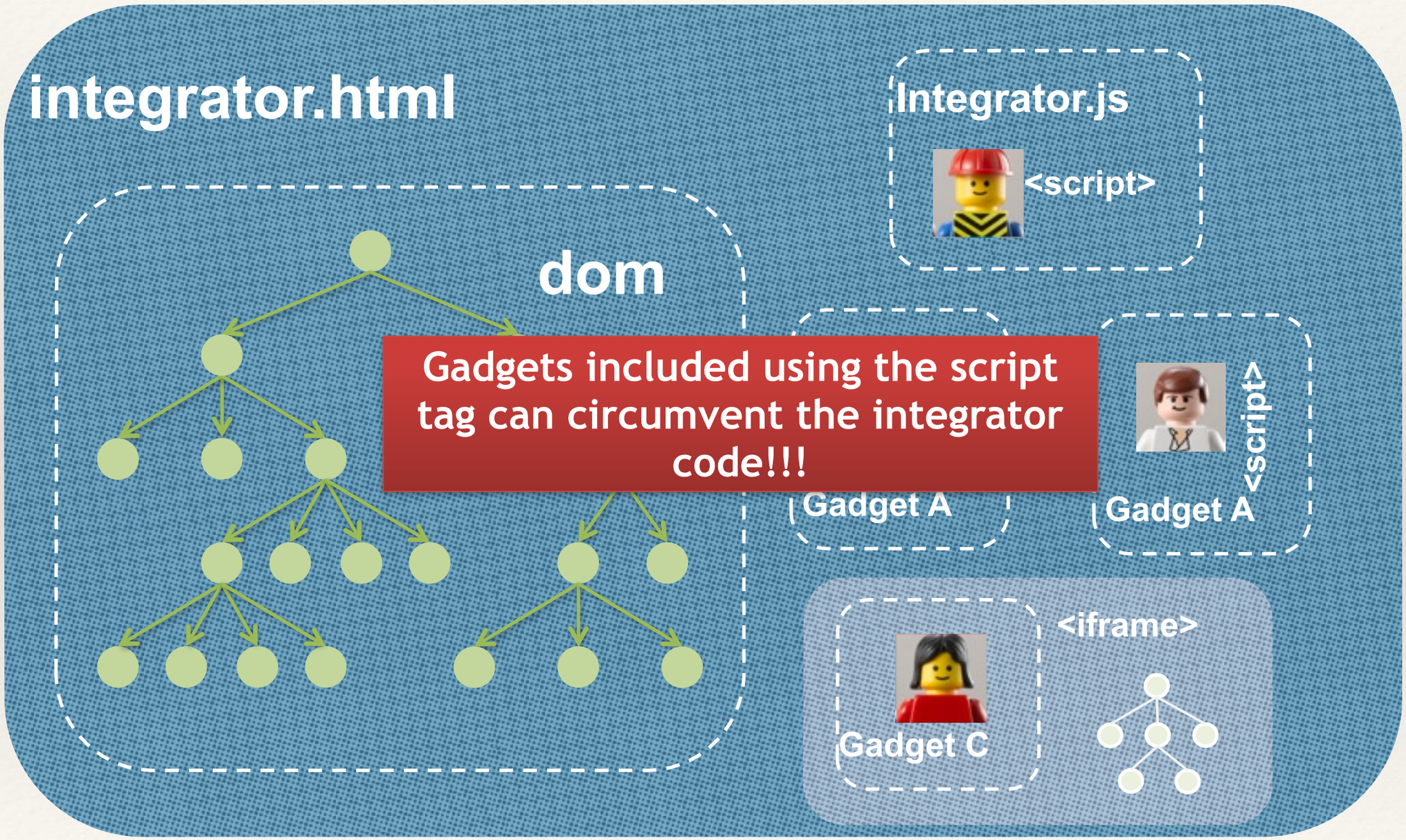
Including External Gadgets



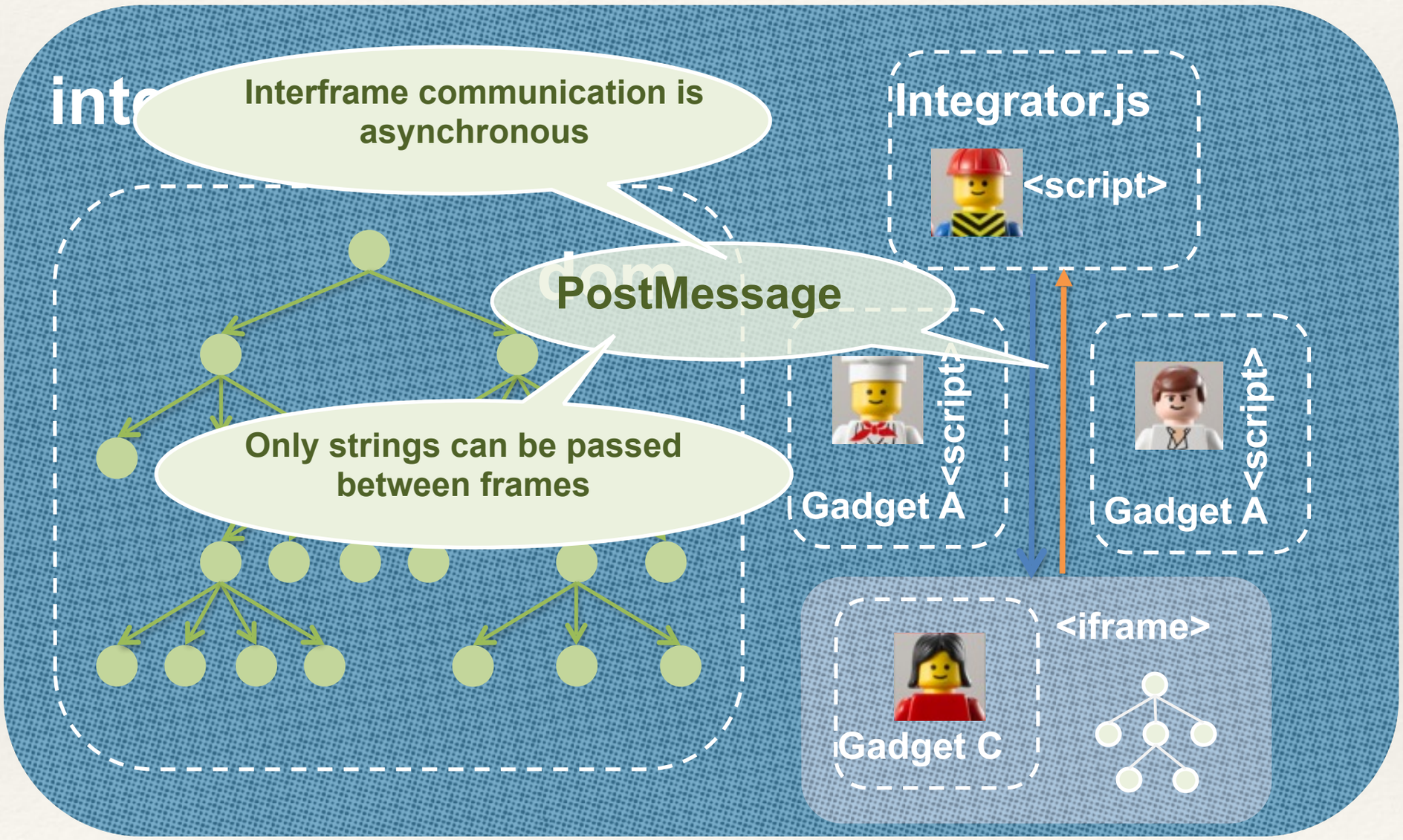
Including External Gadgets



Including External Gadgets



<iframe> and the PostMessage API



<script> versus <iframe>

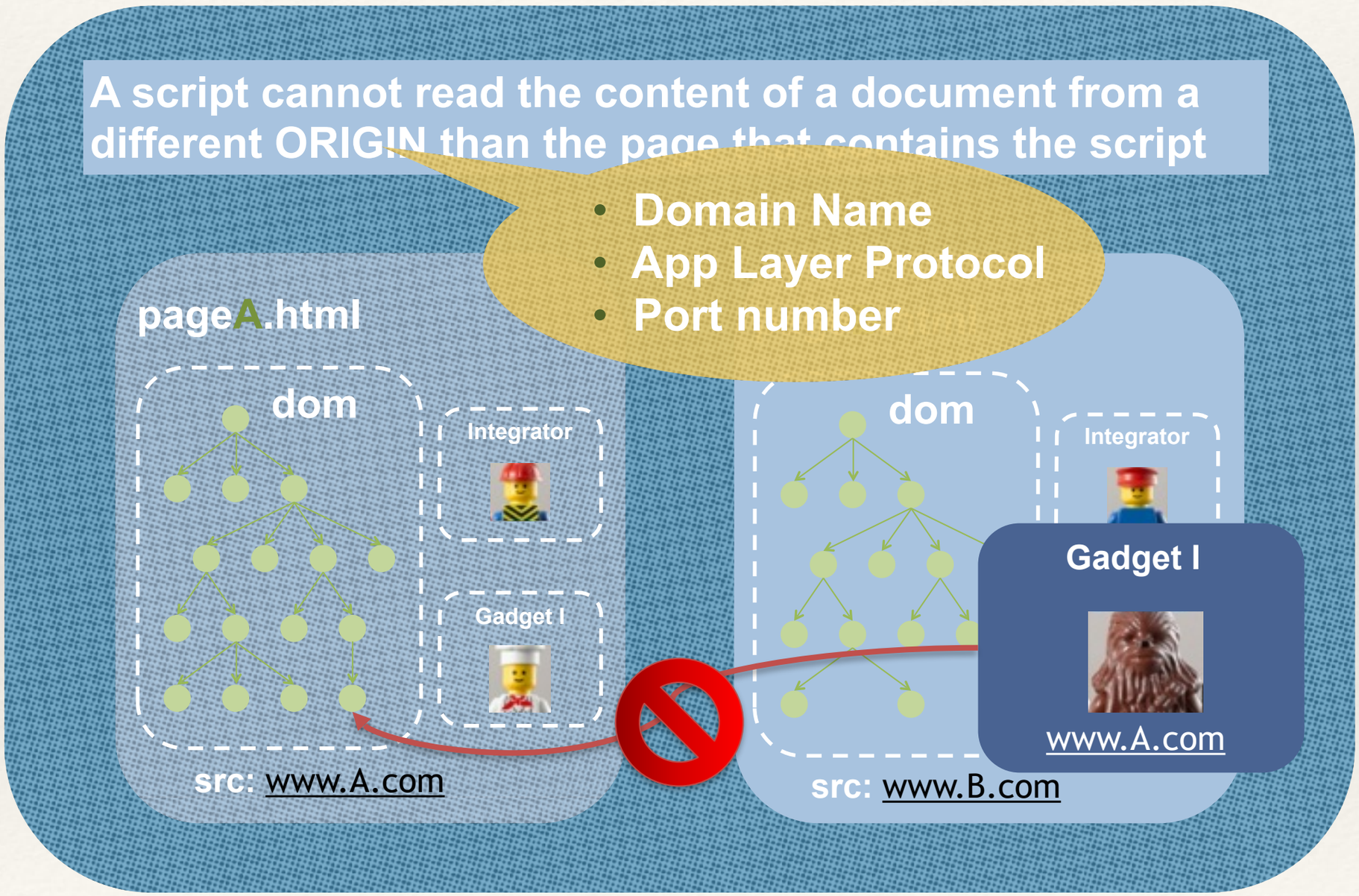
Gadgets with the script tag

- Security Issues
- Communication

Gadgets with the iframe tag

- Communication
- Security

The Same Origin Policy (SOP)



Simulating Private Fields

Simulating Private Fields

- JavaScript does not allow the specification of accessors for object properties

```
Person = function (name, id) {
    this.name = name;
    this.id = id;
}
```

```
p = new Person("John Doe", 1);
p.id
```

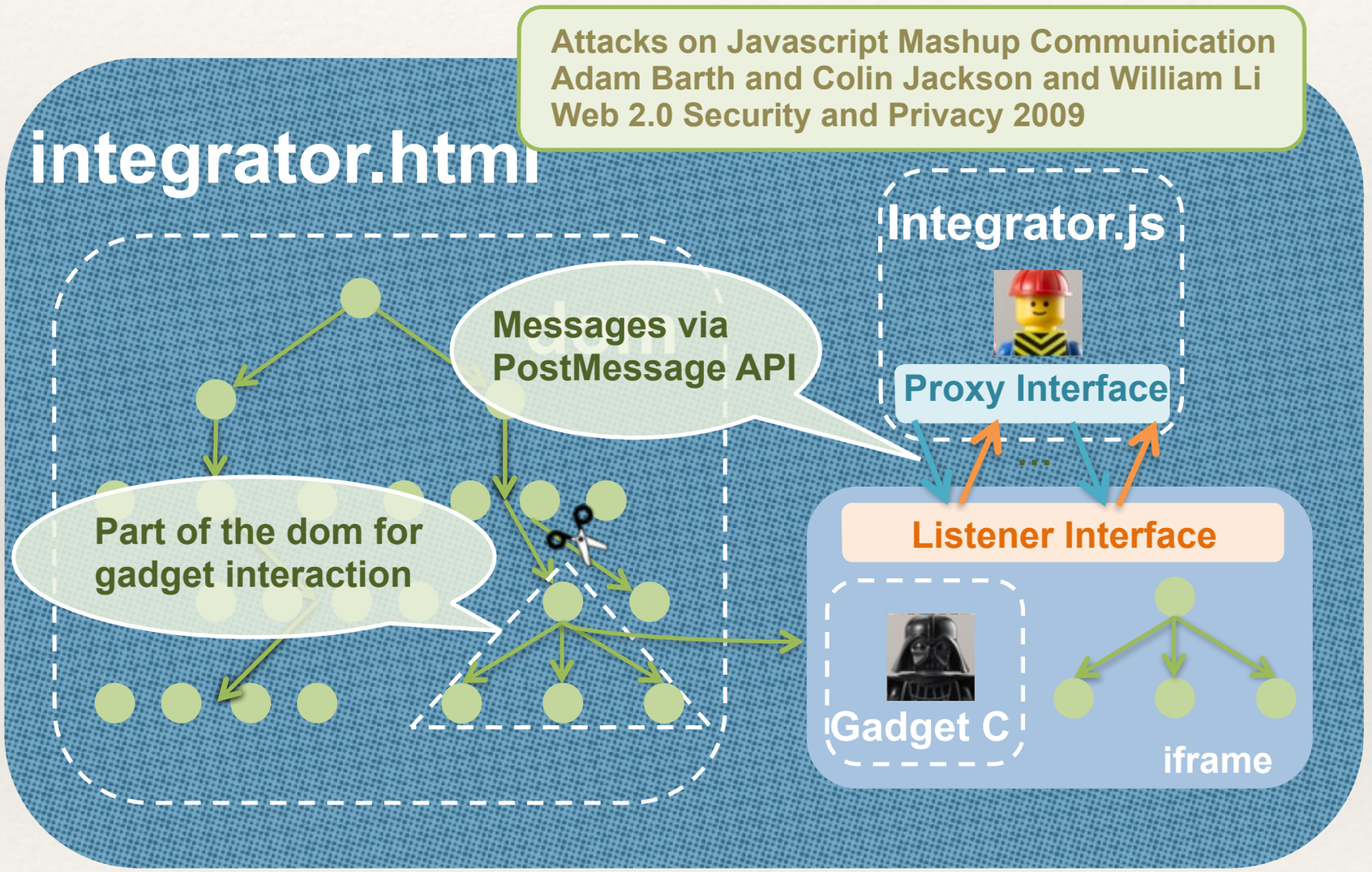
Simulating Private Fields

- JavaScript does not allow the specification of accessors for object properties

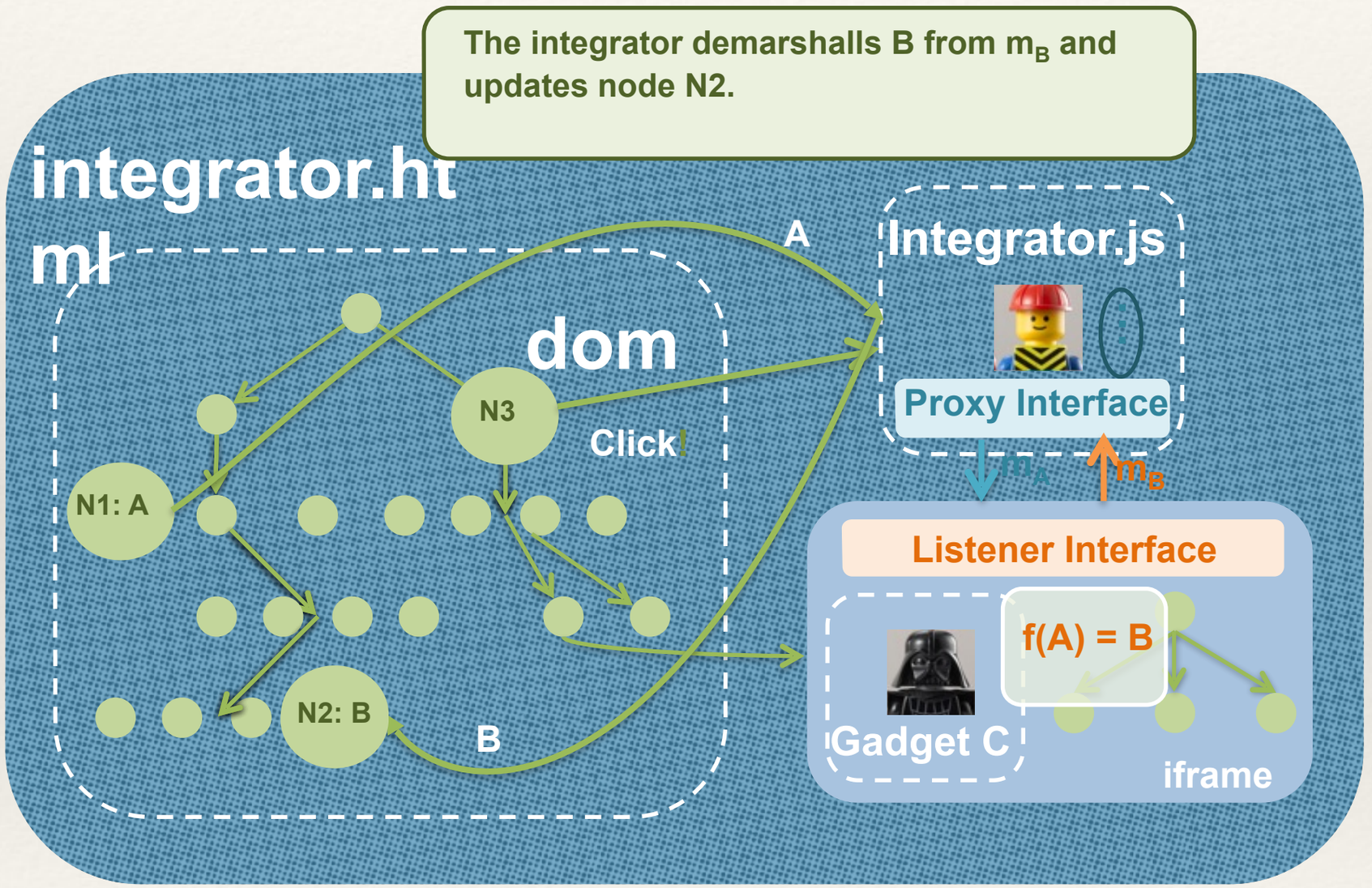
```
Person = function (name, id) {
    var internal_id = id;
    this.name = name;
    this.getId = function (key) {
        // verify the key
        return internal_id;
    }
}
```

Automatic Sandboxing of External Code

Sandboxing External Scripts: A recipe



Sandboxing External Scripts: A recipe



Security By Compilation

Rewrite a program **M** as **M′** so that:

- The external scripts in **M′** are sandboxed in frames
- The semantics of **M′** is contained in the semantics of **M**

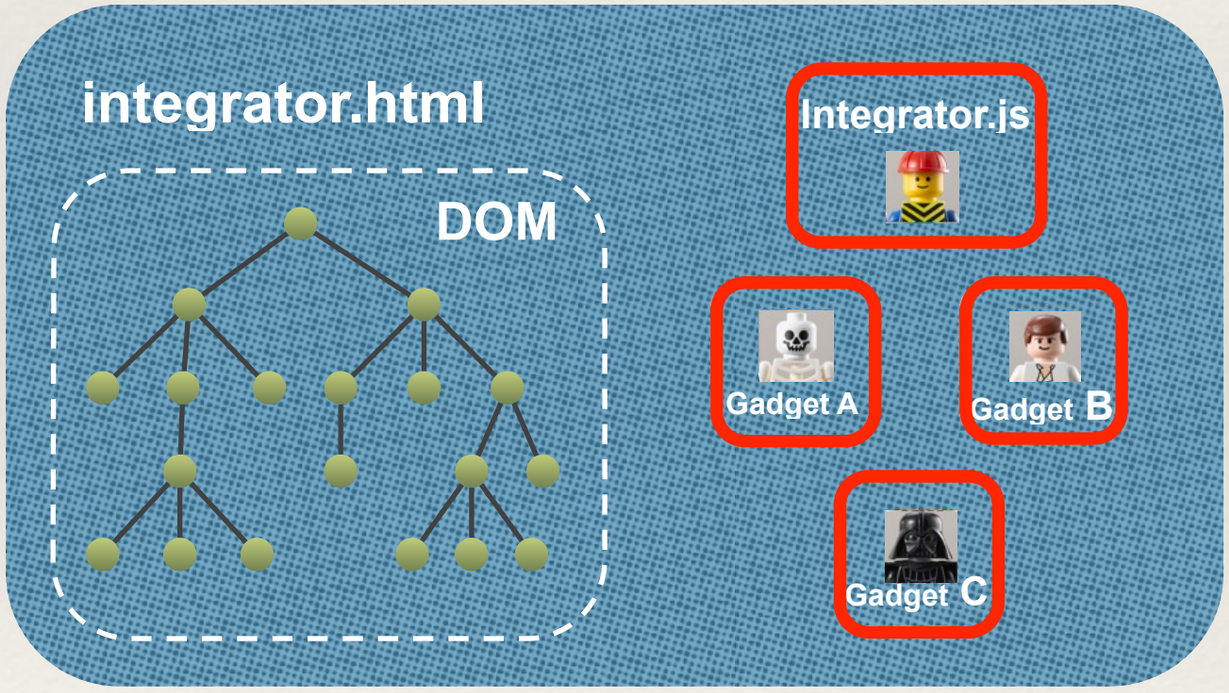
Security By Compilation

Rewrite a program **M** as **M′** so that:

- Use a CPS transformation
- Use Opaque Object Handlers

Monitoring Secure Information Flow in Core JavaScript

Monitoring Secure Information Flow in Core JavaScript

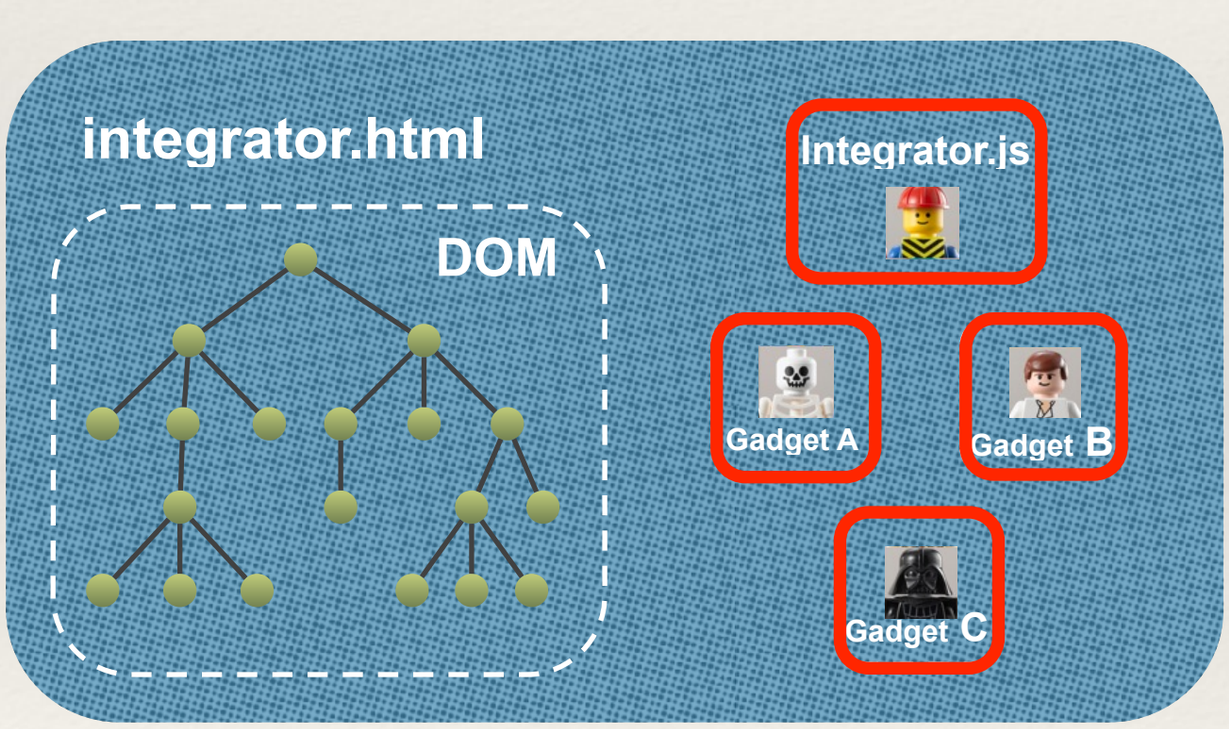


Information flow Monitor

+

Inlining Transformation

Monitoring Secure Information Flow in Core JavaScript



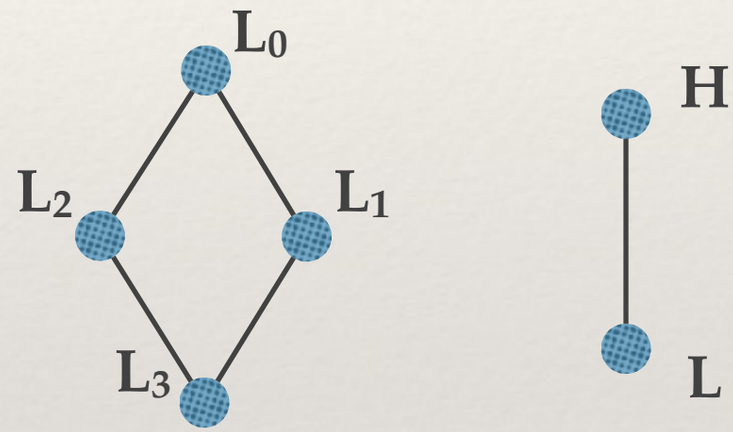
Information flow Monitor and Inlining Transformation

- Purely Dynamic
- Flow Sensitive
- No-Sensitive Upgrade

Labeling Program Resources

Public Outputs (**Low**) may **NOT** depend on **Private** Inputs (**High**)

- Establish a **lattice of security levels**
- Label **resources** with security levels



Σ : **Resources → Levels**

Labeling JavaScript Objects

What do we mean by RESOURCES?

How should we label these RESOURCES?

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

Resources: Object Properties and Variables

Labeling JavaScript Objects

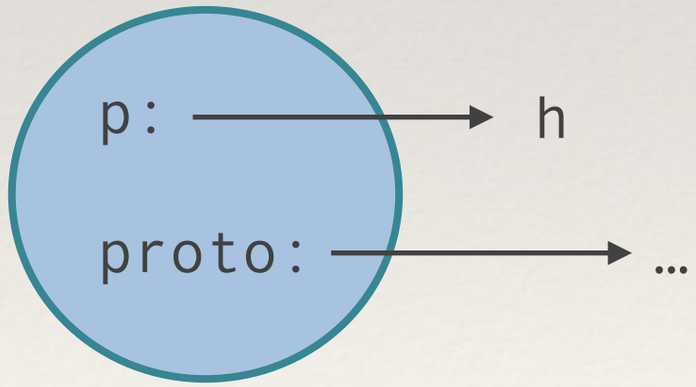
Check the **value** of a property:

```
o = {};
o.p = h;
l = o.p
```

Check whether a given property **exists**:

```
o = {};
if (h) {
  o.p = h
}
l = “p” in o
```

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

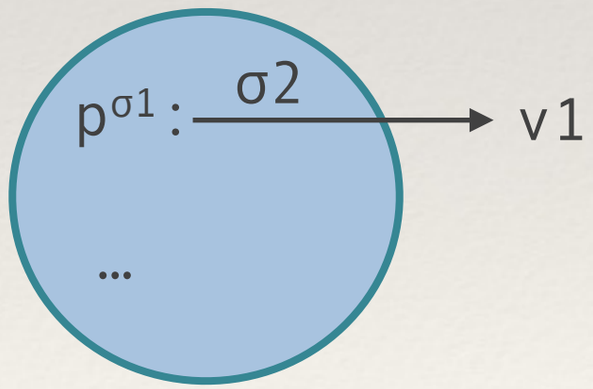


Labeling JavaScript Objects

Associate each **property** with:

- A **value level** corresponding to its value
- An **existence level** corresponding to its existence

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



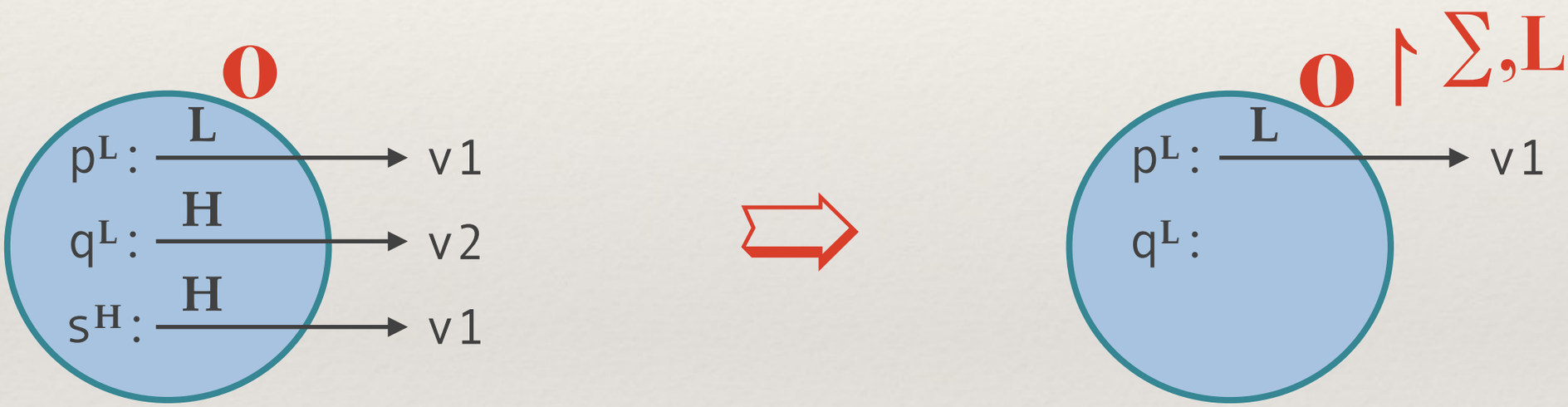
Indistinguishable JavaScript Memories

What can an attacker see at level σ?

- The existence of properties whose existence levels are ≦ **σ**
- The values associated with properties whose position levels are ≦ **σ**
- The values associated with variables whose security levels are ≦ **σ**

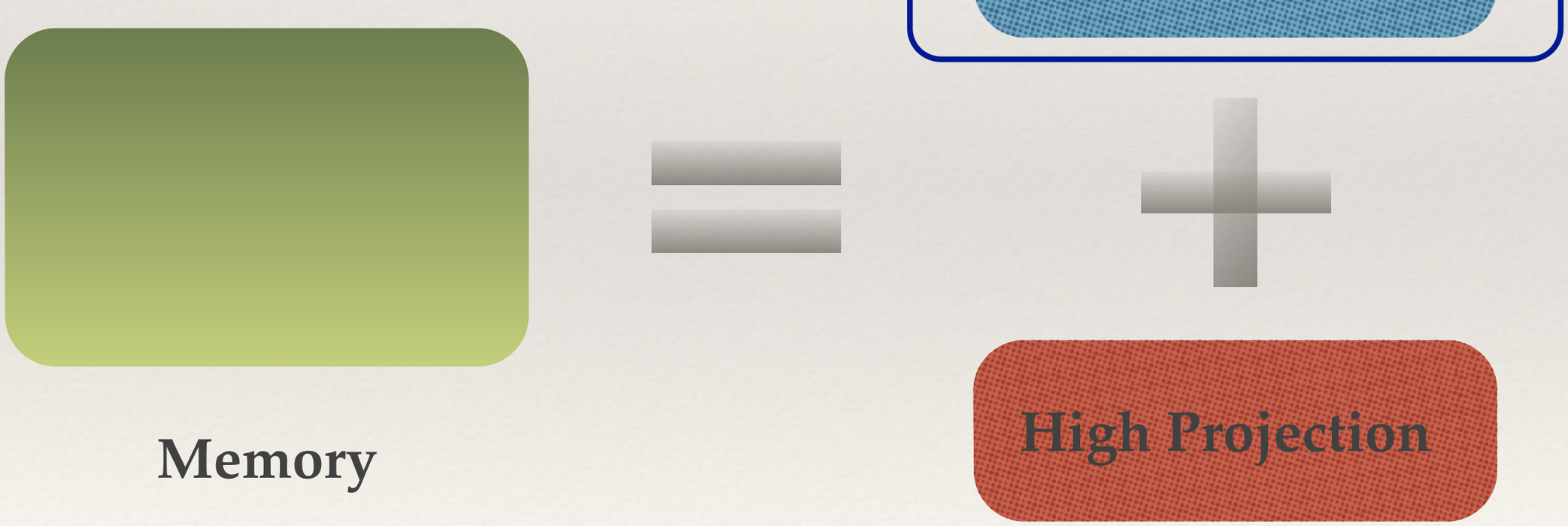
Indistinguishable JavaScript Memories

What can an attacker see at level L?



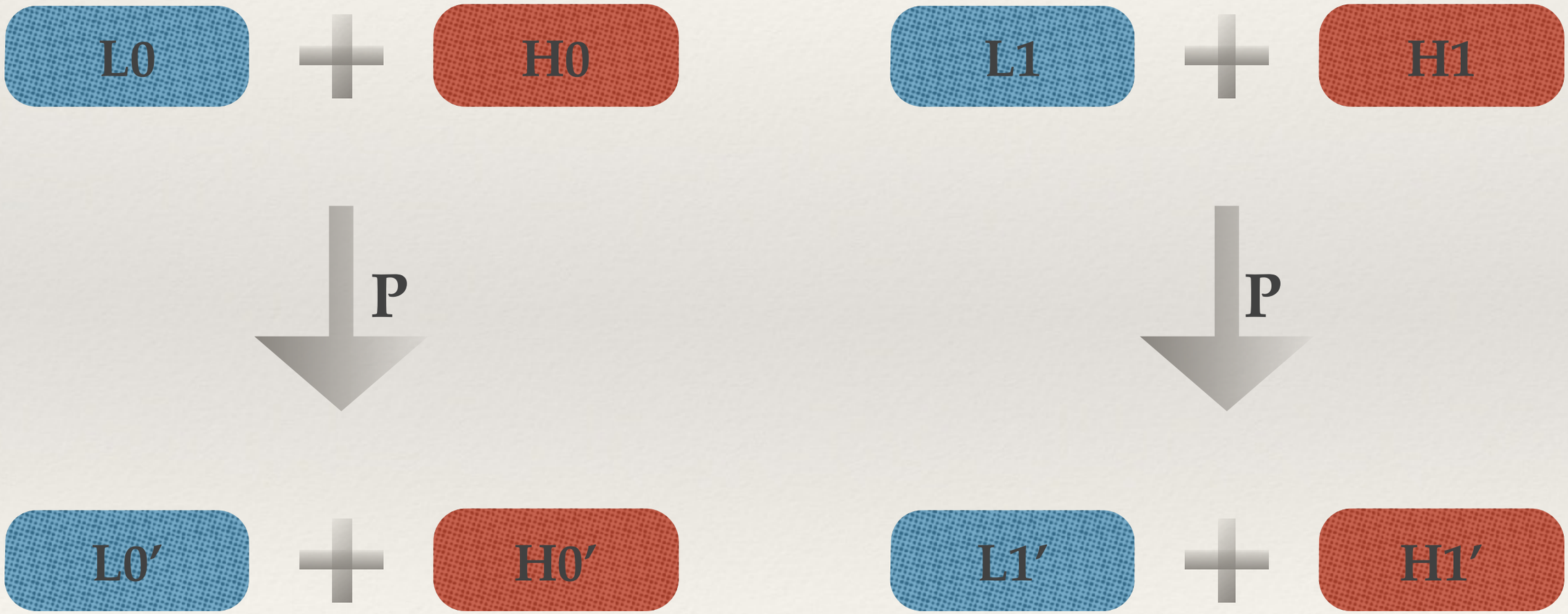
Indistinguishable JavaScript Memories

What can an **attacker observe** at a given level **σ**?



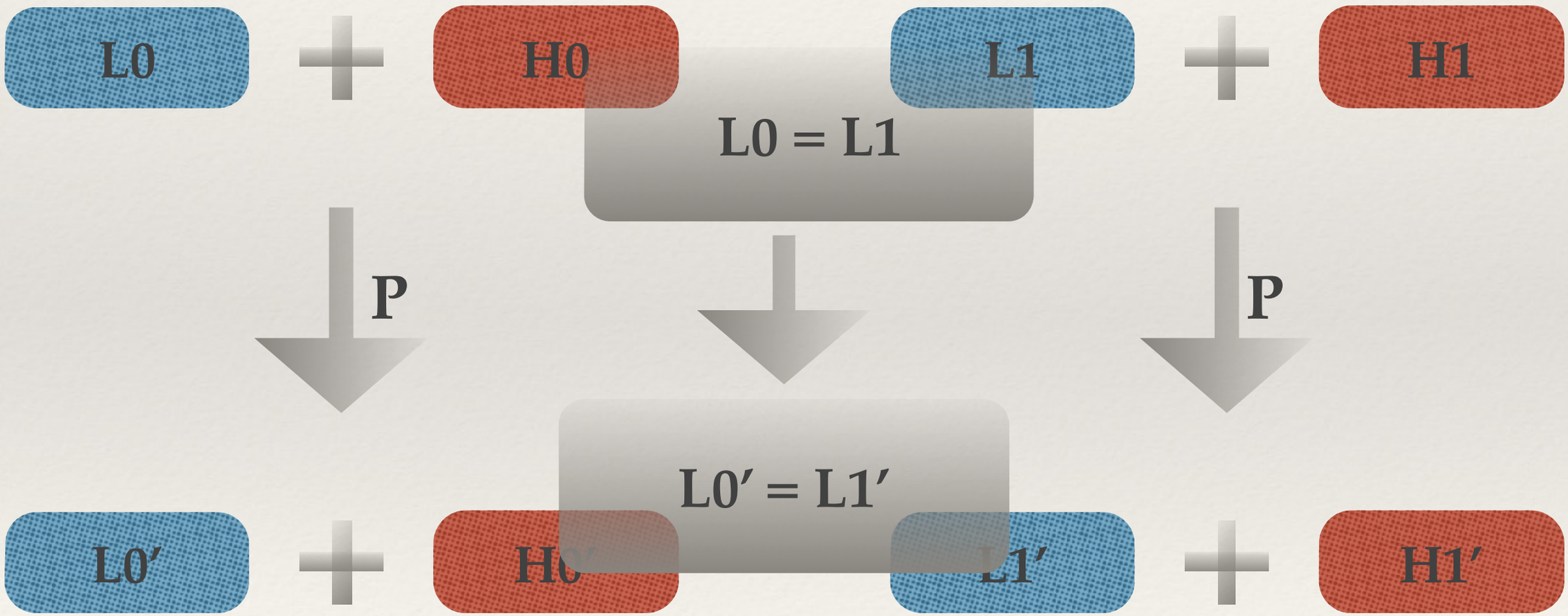
Noninterference

When should a problem **P** allowed to execute?



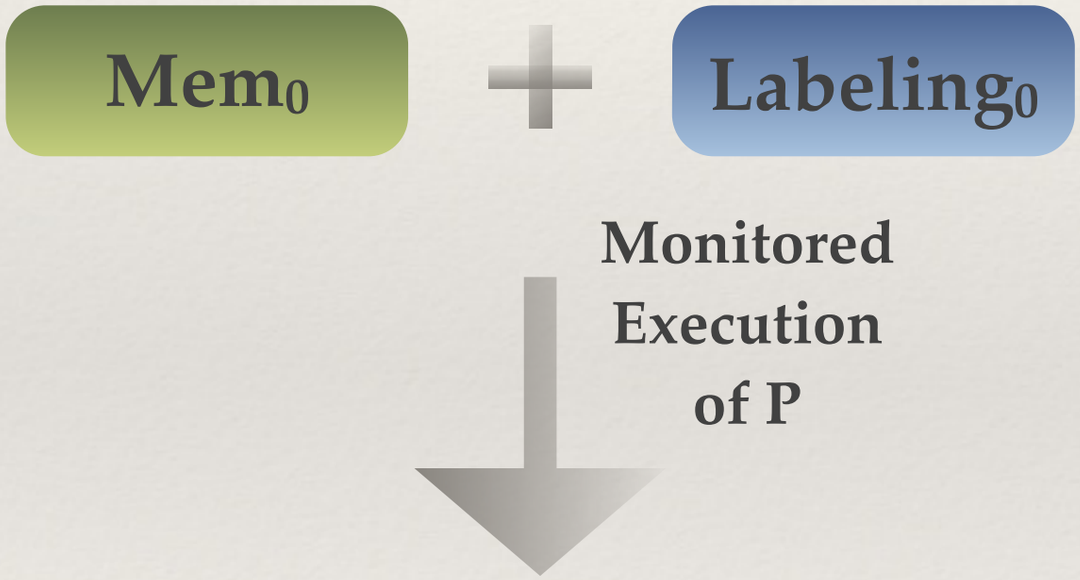
Noninterference

When should a problem **P** allowed to execute?



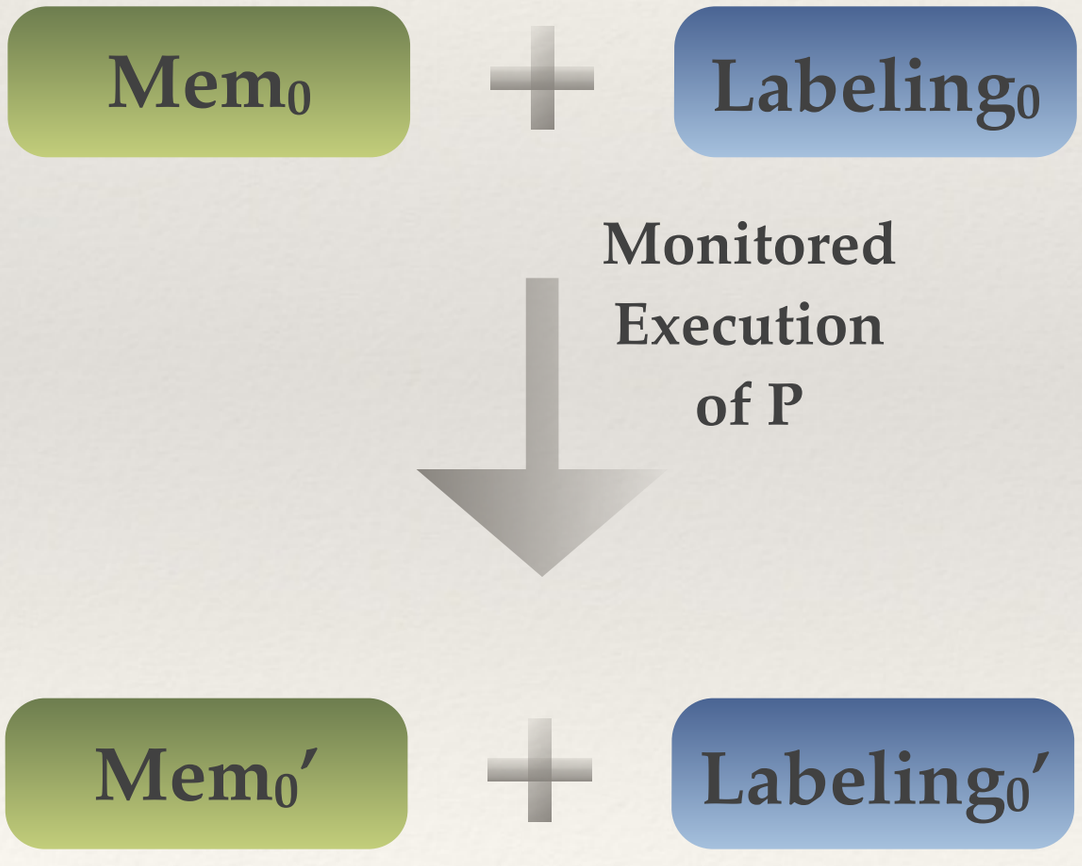
Information Flow Monitoring

Standard Execution + **Abstract Execution on the Labeling + Constraints**



Information Flow Monitoring

Standard Execution + **Abstract Execution on the Labeling + Constraints**



Constraints:

- **No-sensitive upgrades**
- **Visible resources cannot be upgraded in invisible contexts**
- **Example: One cannot update a property with a visible value level in an invisible context**

Security By Compilation

Rewrite a program **P** as **P′** so that:

- **P′** only executes if the execution of **P** is secure
- The semantics of **P′** is contained in the semantics of **P**

Security By Compilation

A **recipe** for designing **inlining transformations**:

- Specify a monitored semantics
- Prove the monitored semantics secure
- Specify an program transformation that inlines the monitor
- Prove that the inlining compiler is correct

Security By Compilation

Encode the Security Labeling in Memory

Security By Compilation

$$\text{Comp}\langle \mathbf{P} \rangle = \mathbf{P}'$$

PhD Grants at INRIA Sophia Antipolis

starting date between March and October 2015 (3 years)

- Formal Analysis and Implementations for JavaScript Security and Privacy:
  - **Tamara Rezk**: tamara.rezk@inria.fr
  - **Nataliia Bielova**: nataliia.bielova@inria.fr
- Reactive Web Programming:
  - **Manuel Serrano**: manuel.serrano@inria.fr

Thank You