JavaScript Bindings @BlinkOn 1.0

haraken@chromium.org

I am haraken

from Tokyo

here

Nice to meet you!

Topics

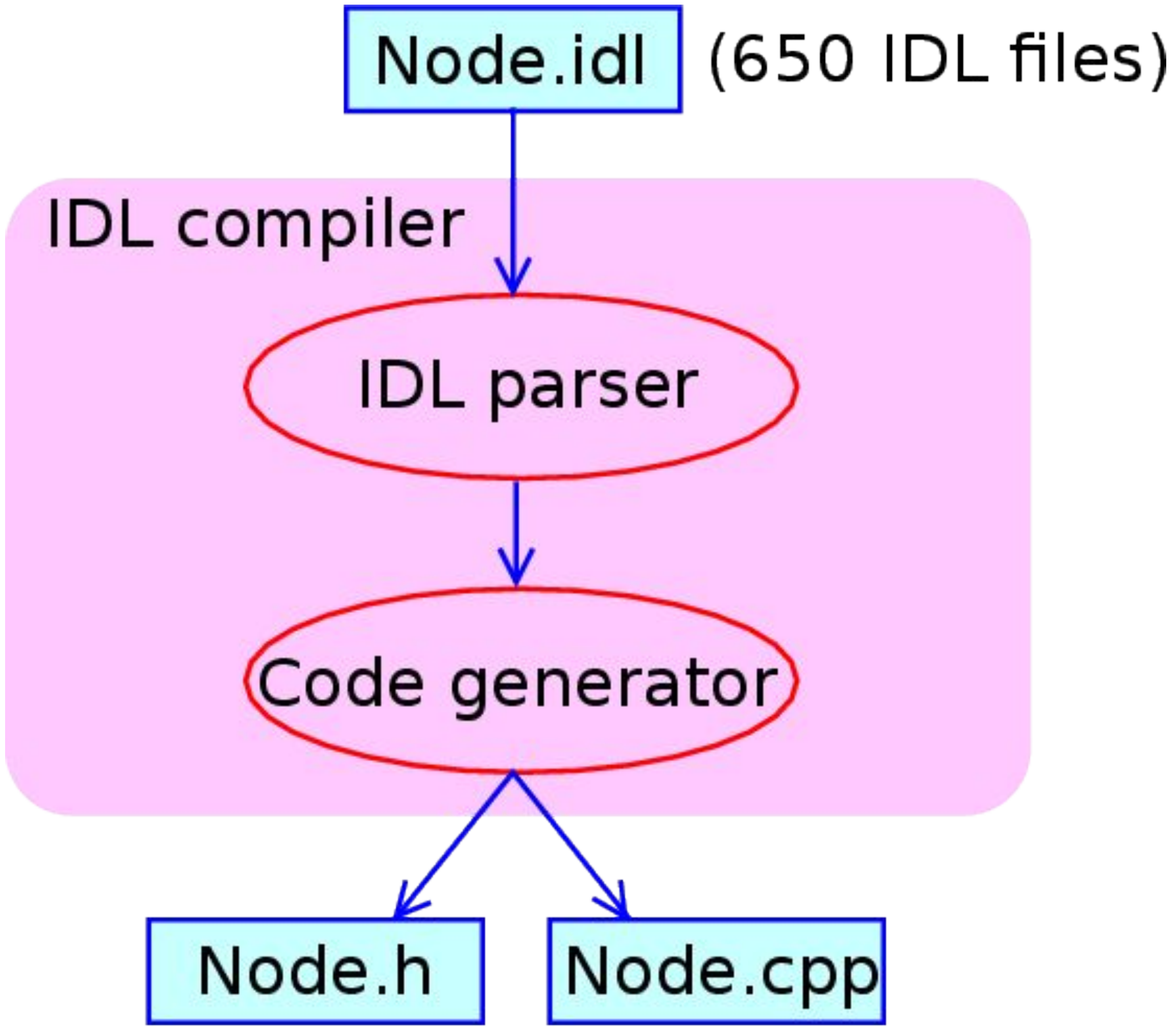
- Code quality
- Memory
- Performance
- Security

Special thanks!

- Adam Barth, Adam Klein, Joshua Bell
- Christophe Dumez
- Jochen Eisinger, Dan Carney, Marja Holtta
- Kentaro Hara, Nils Barth, Kouhei Ueno, Koji Hara
- Thomas Sepez
- Nat Duca, Dan Sinclair, Scott Graham
- Mads Ager, Erik Corry, Vyacheslav Egorov, Ian Zerny, Gustav Wibling

Code quality

1. Rewriting the IDL compiler in Python



1. Rewriting the IDL compiler in Python

- Currently, the IDL compiler is written in 8000 lines of Perl
    - Very messy

- We're rewriting it in Python
    - Much more readable using jinja template engine
    - Faster

2. Cleaning up IDL attributes

- IDL attributes control generated code
    - e.g., [EnableAtRuntime], [Constructor], …

- Too many IDL attributes have embarrassed Blink developers

- Since Blink started, we've decreased # of IDL attributes from 115 to 61

- Well documented here

3. Removing custom bindings

- Ideally: 100% binding code should be auto-generated

- In reality: the IDL compiler is not mature enough, and thus a lot of binding code are hand-written
    - Problem: they are buggy (especially for edge cases)

3. Removing custom bindings

- We've removed 1000 lines of custom bindings since Blink started

- However, we still have 11000 lines of custom bindings

- We should remove more!

Summary about code quality

- We’ve been improving the IDL compiler, but need to improve more

- I’ve seen a lot of Blink developers having trouble in writing binding code
    - Bindings shouldn’t block Blink developers from implementing new Web features

Summary about code quality

- Volunteers are welcome!

Good news:

- There are a lot of low hanging fruits
- We have active reviewers covering all time zones
    - christophe (East US), abarth (West US), haraken (Japan), jochen (Germany)

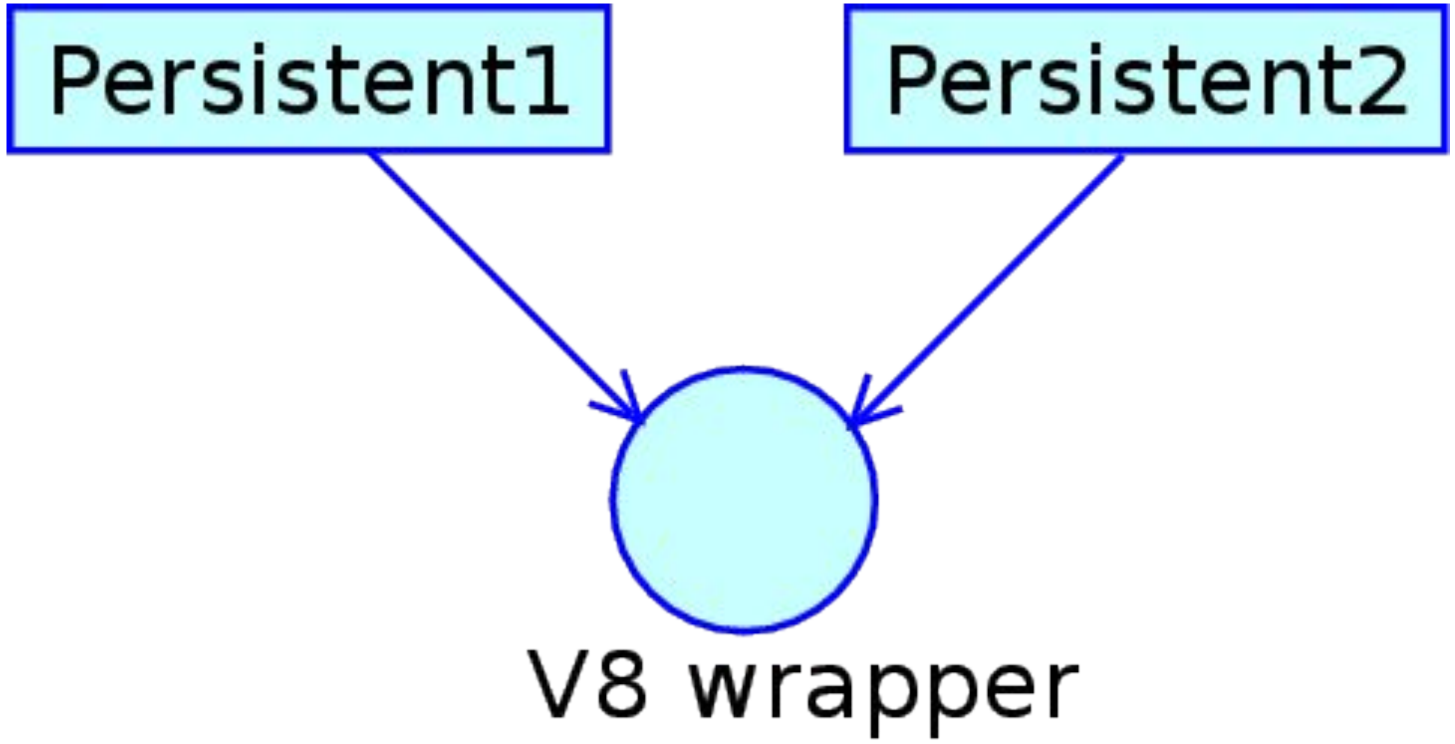
Memory

1. Making V8 handles safer

- There are two kinds of V8 handles
  - Handle (on-stack handle):
    - Automatically deallocated when the current HandleScope exits
  - Persistent (on-heap handle):
    - Not deallocated until you explicitly call Dispose()
    - Persistents can have weak callbacks

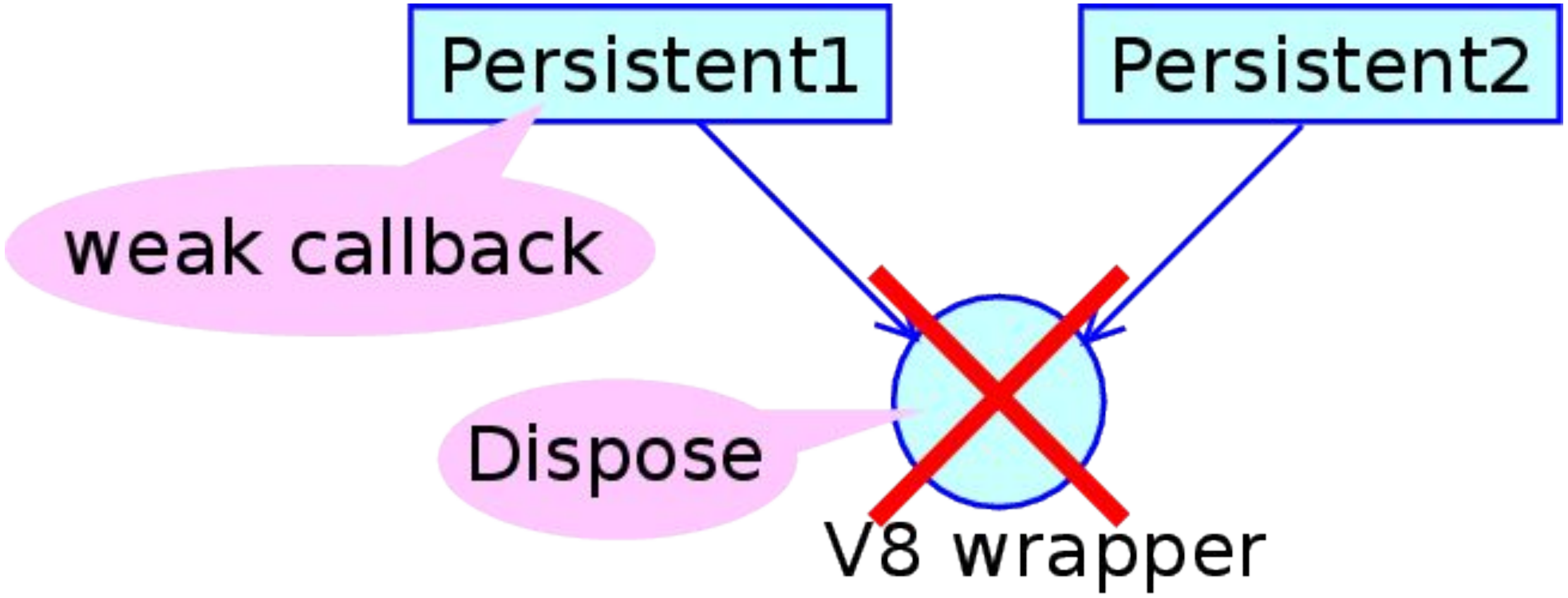
1. Making V8 handles safer

- A problem happens when two Persistents point to the same V8 wrapper



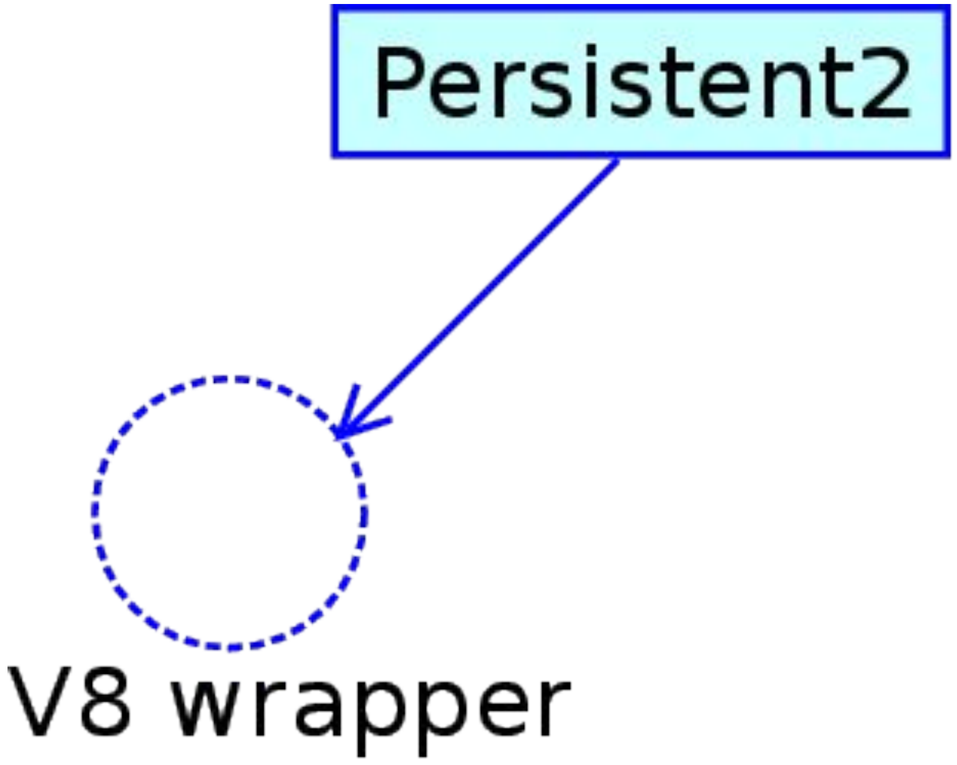
1. Making V8 handles safer

- Imagine that a weak callback of Persistent1 is triggered and calls Dispose()



1. Making V8 handles safer

- ...then Persistent2 becomes a dangling pointer



1. Making V8 handles safer

Solution: Disallow copy constructors of Persistents

- to make sure that the problematic situation (i.e., two Persistents point to the same wrapper) never happens

- We've removed all Persitent copying from Blink, and deprecated all fragile V8 APIs

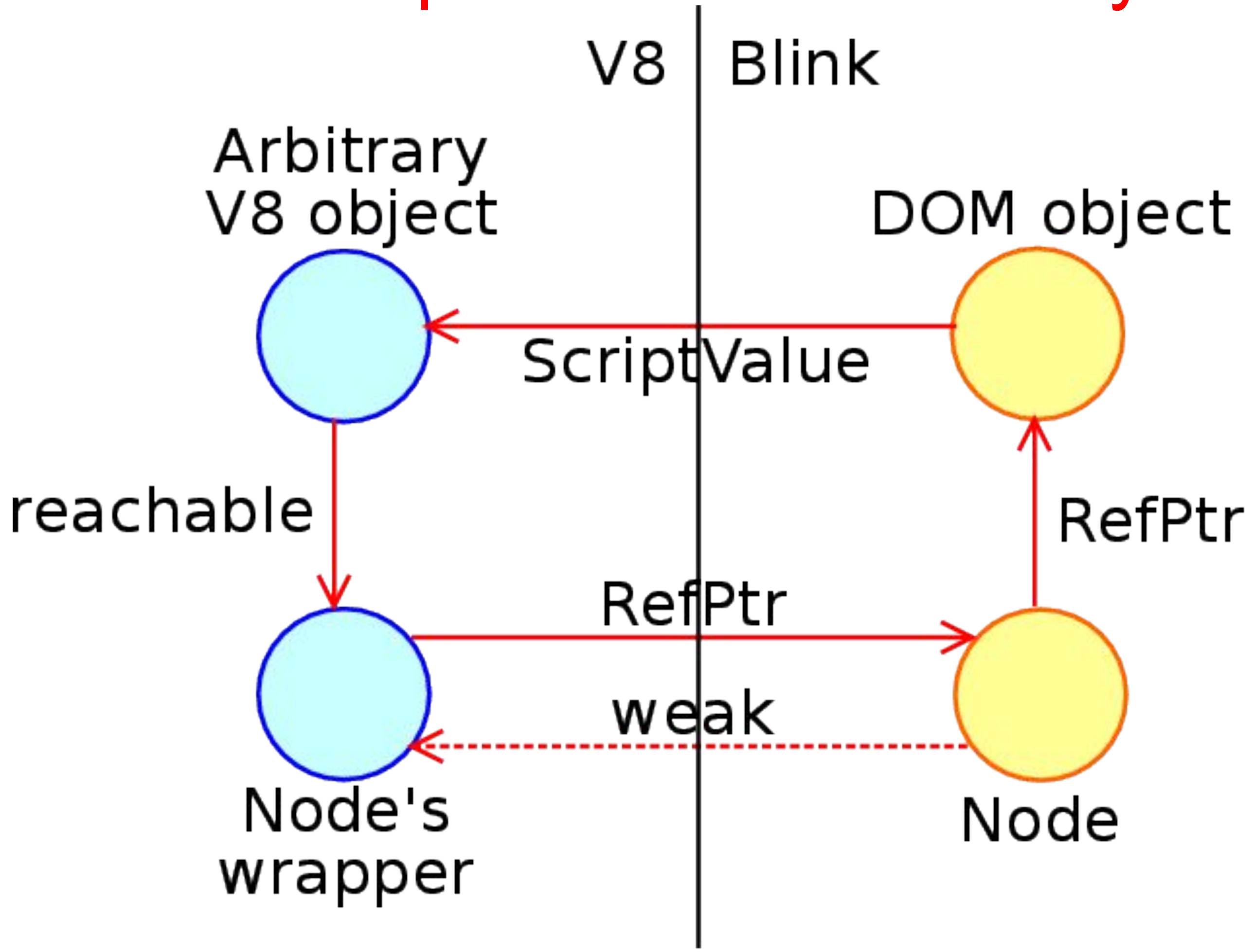
2. Removing ScriptValue

- ScriptValue is a class to hold a V8 wrapper in DOM objects
    - Internally, ScriptValue is realized by holding a Persistent to the V8 wrapper

```
class DOMObjectInCore {
  …;
  ScriptValue m_value;
};
```

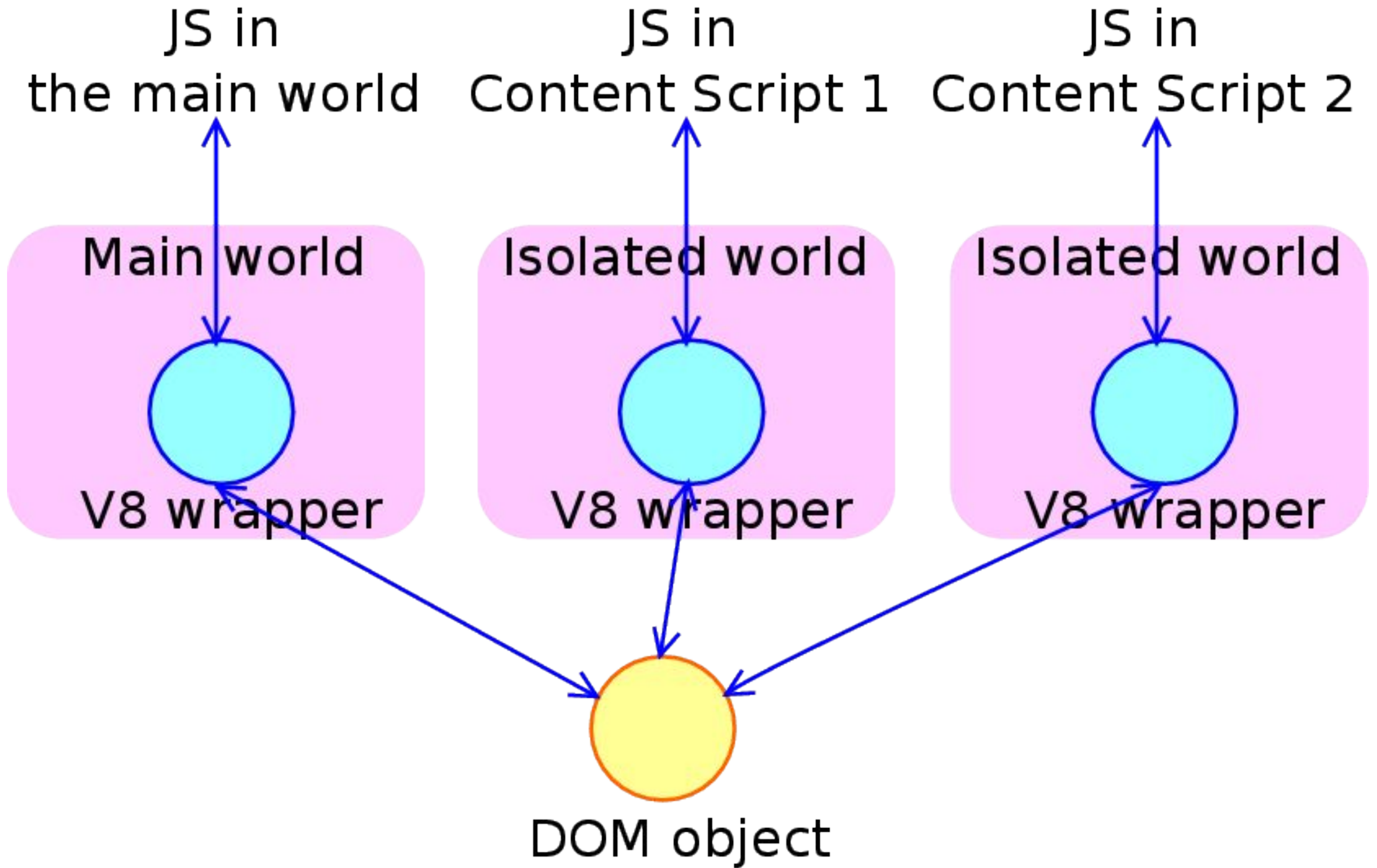
2. Removing ScriptValue

Problem 1: It can produce reference cycles



2. Removing ScriptValue

Problem 2: It can leak V8 wrappers between isolated worlds (i.e., Content Scripts)



2. Removing ScriptValue

Problem 2: It can leak V8 wrappers between isolated worlds

```
class DOMObject {
  ScriptValue value() {
    // Isolated-world-unaware...
    return m_value;
  }
  ScriptValue m_value;
};
```

2. Removing ScriptValue

- In summary, ScriptValues are dangerous

- We’re removing ScriptValues (with a substantial amount of whiteboard work :-)

3. Oilpan

- Oilpan is a project that aims to replace reference counting in Blink with a general garbage collection

- In short: GC for Blink

3. Oilpan

Goal 1: Simpler programming model

- You no longer need to worry about reference cycles
- Thus, you no longer need raw pointers and worry about their lifetime

```
class A { RefPtr<B> m_b; }
class B { A* m_a; } // back pointer
```

3. Oilpan

Goal 2: Better security

- Reference cycles don't have to be broken by holding raw pointers which you can easily forget to clear out
- No use-after-free

```
class A { RefPtr<B> m_b; }
class B { A* m_a; } // back pointer
```

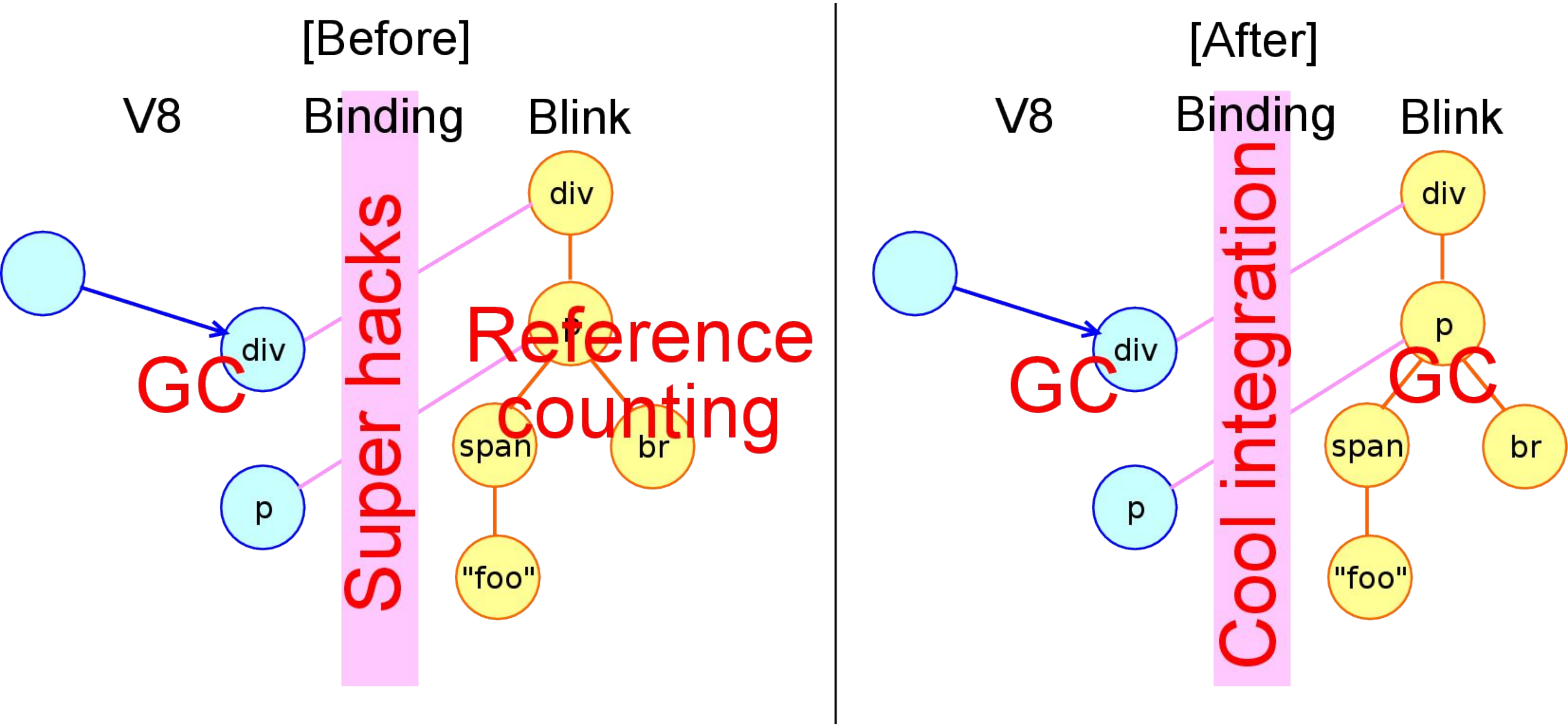
3. Oilpan

Goal 3: DOM becomes traceable

- With Oilpan, we can completely understand reachability of all objects in JavaScript and DOM
    - We can create an excellent devtool to diagnose memory leaks
- In long-term: DOM snapshotting

3. Oilpan

Goal 4: Better GC integration between V8 and Blink



3. Oilpan

Current status:

- We’re developing in a public, experimental branch
- We’ve moved some DOM objects in modules/, the CSS hierarchy and the Node hierarchy to Oilpan’s heap
- We’re starting performance work to make the regression down to 0

3. Oilpan

Future plan:

- Once performance & programmability problems are resolved, we will start upstreaming

Summary about memory

- In short-term: we're killing use-after-frees and reference cycles
    - Making V8 handles safer
    - Removing ScriptValues
    - ...

- In long-term: Oilpan will solve a ton of memory problems we currently have

Performance

1. Dromaeo results

	Firefox 22 on Win	Chrome 30 on Win
DOM Attributes	1316 runs/s	615 runs/s
DOM Modification	294 runs/s	351 runs/s
DOM Query	13657 runs/s	12243 runs/s
DOM Traversal	369 runs/s	343 runs/s
Total	1640 runs/s	1361 runs/s

1. Dromaeo results

- In Windows, Chrome is 20% slower than Firefox

    c.f., In Linux, Chrome is 10% faster than Firefox

- Indeed Dromaeo is a micro benchmark,
 but this is a problem

- We should make it faster!

2. Is Dromaeo a good benchmark?

- Good question!

2. Is Dromaeo a good benchmark?

- Dromaeo is just a micro-benchmark
- However, Dromaeo is testing very common call paths in web applications

IMHO:

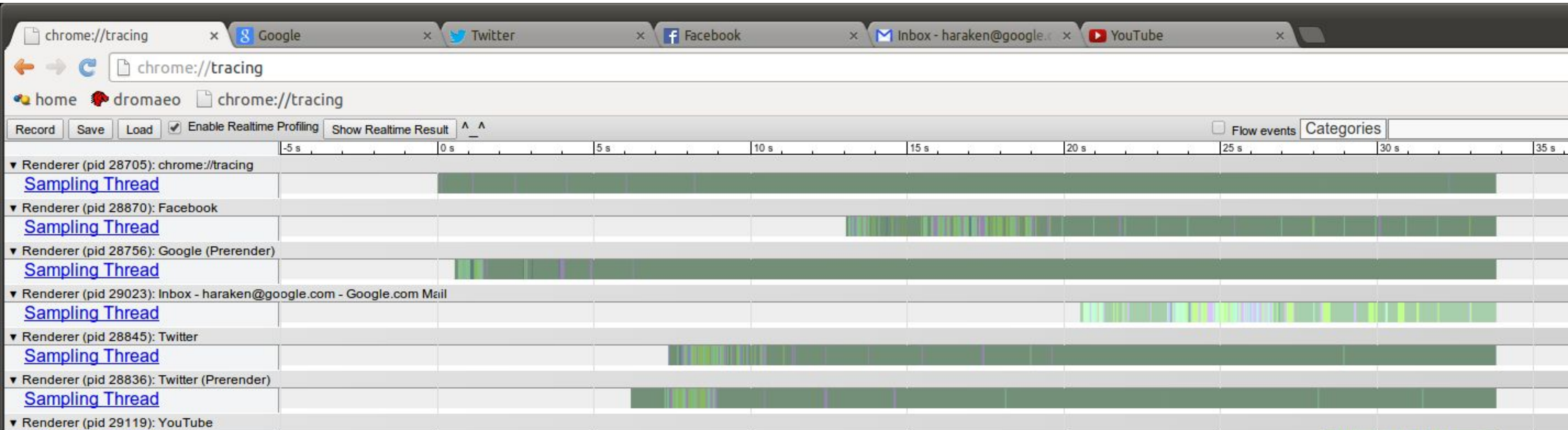
- It's worth watching Dromaeo performance (i.e., we shouldn't regress Dromaeo or lose in Dromaeo)
- However, when doing optimization work, we should profile real-world apps, not Dromaeo

3. Profiling real-world apps

- We’re moving our focus from micro benchmarks to real-world web apps

3. Profiling real-world apps

- Implementing a sampling profiler in about:tracing
  - which can visualize what percentages of the main thread executions are charged on what performance factors (layout, style recalculation, major GC, DOM attribute getters, etc)



Renderer (pid 28705): chrome://tracing
Sampling Thread
Renderer (pid 28870): Facebook
Sampling Thread
Renderer (pid 28756): Google (Prerender)
Sampling Thread
Renderer (pid 29023): Inbox - haraken@google.com - Google.com Mail
Sampling Thread
Renderer (pid 28845): Twitter
Sampling Thread
Renderer (pid 28836): Twitter (Prerender)
Sampling Thread
Renderer (pid 29119): YouTube

4. Build performance

- The IDL compiler is slow

- If you touch any IDL file, all 650 IDL files are rebuilt
    - This implies that all dependent .h/.cpp files are also rebuilt
    - This is a serious problem in Mac/Windows/Android builds

4. Build performance

- We have an optimization plan
    - which guarantees that only necessary IDL files are rebuilt

Summary about performance

- We’re moving our focus from micro benchmarks to real-world web apps

- However, it’s still problematic that Dromaeo is slow in Blink -- let’s make it faster!

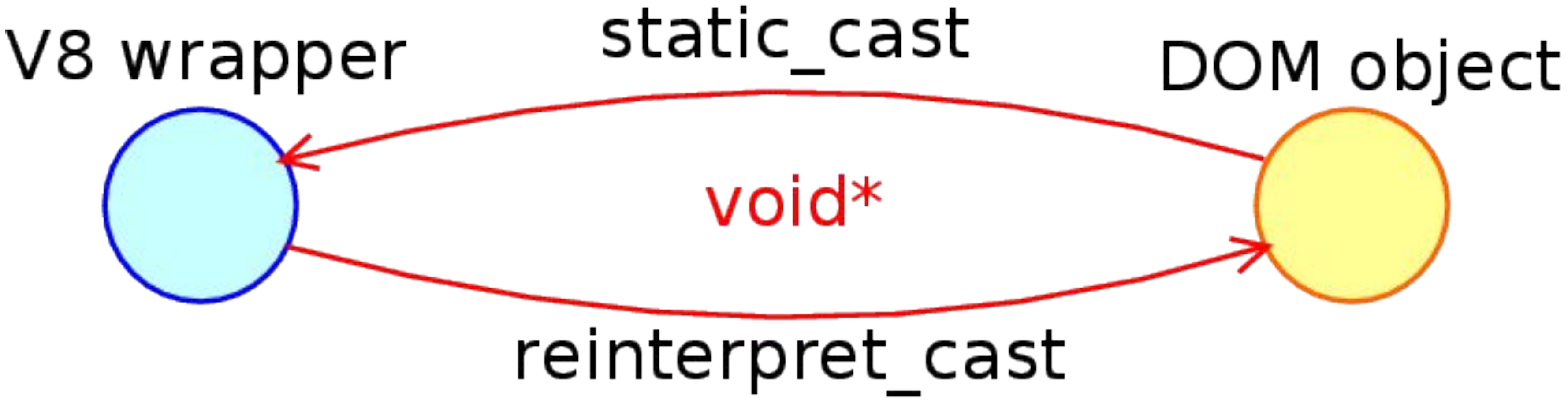
Good news:

  - No one complains about optimization work
    - Volunteers are welcome!

Security

Binding integrity

- The binding layer is the best target for security exploits
    - because type information is lost when converting DOM objects <=> V8 wrappers



Binding integrity

Solution: We’re adding security checks to prevent the security exploits that abuse types and use-after-frees

- In a nutshell, whenever Blink returns V8 wrappers to V8, we check that the wrappers have correct types

Binding integrity

(1) When we create a DOM object:

- Store the object type into the DOM object

(2) When we wrap the DOM object:

- Check that the stored type is still correct
- Store the C++ pointer of the DOM object into the wrapper

(3) When we return the wrapper to V8:

- Check that the C++ pointer stored in the wrapper is identical to the C++ pointer of the DOM object

Summary about security

- It’s complicated

- The point is that the security checks guarantee that DOM objects are always wrapped to V8 wrappers of correct types

Thanks for listening!

- Code quality

- Memory

- Performance

- Security

V8 bindings welcome contributors!

Active reviewers are waiting for your patches.

Questions?

Code quality:

- Rewriting the IDL compiler in Python, Removing custom bindings

Memory:

- Making V8 handles safer, Removing ScriptValues, Oilpan

Performance:

- Dromaeo is slow, Profiling real-world apps, Build performance

Security:

- Binding integrity