Cross-Engine Contributions at Scale: How newcomers accelerated Temporal and Upsert in SpiderMonkey, V8, and Boa

Jonas Haukenes (University of Bergen)
Mikhail Barash (University of Bergen)
Shane F. Carr (Google)

Brage Hogstad
Henrik Tennebekk
Idris Elmi
Johannes Helleve
Jonas Haukenes
Lauritz Angeltveit
Magnus Fjeldstad
Mathias Ness
Sebastian Matthews
Sune Lianes
Vetle Larsen

Daniel Minor
Kevin Ness
Mikhail Barash
Philip Chimento
Shane F. Carr

Mikhail Barash

Associate Professor of Programming Languages
Co-Convener, Ecma TC39-TG5
Member, Ecma Executive Committee

Academic supervisor of the project

How to enable more contributions from newcomers?

University of Bergen (Norway)

- Students who are eager to learn
- Without prior experience in contributing to open-source projects

Project Setting and Motivations

- Specialized class tailored for 5-7 participating students

- Real-world impact
- Deep technical learning
- Involvement in open-source and standards work
- Bridging academia & industry
- Accelerated prototyping & implementation
- Web engines can offload some educational work to academia

2 proposals
3 case studies

during Fall 2024 – Spring 2025

Case Study 1
`Map.prototype.upsert` in **SpiderMonkey**
with a mentor's support

Case Study 2
`Map.prototype.upsert` in **V8**
without a mentor's support

Case Study 3
Temporal in **Boa**
and **indirectly** in **V8**
with mentors' support

`Map.prototype.upsert` in SpiderMonkey & V8

Case Studies 1 and 2

Jonas Haukenes

Master Student in Algorithms & Programming Theory
Delegate, Ecma TC39

Participant of case studies 1 & 2

Other participants of case study 1

Sune Lianes

Mathias Ness

Lauritz Angeltveit

Vetle Larsen

Map.prototype.upsert

```
// Currently
let prefs = new getUserPrefs();
if (!prefs.has("useDarkmode")) {
  prefs.set("useDarkmode", true); // default to true
}
```

```
// Using getOrInsert
let prefs = new getUserPrefs();
prefs.getOrInsert("useDarkmode", true); // default to true
```

Looking at the spec

1 Map.prototype.getOrInsert (*key*, *value*)

When the getOrInsert method is called the following steps are taken:

1. Let *M* be the **this** value.
2. Perform ? RequireInternalSlot(*M*, [[MapData]]).
3. Set *key* to CanonicalizeKeyedCollectionKey(*key*).
4. For each Record { [[Key]], [[Value]] } *p* of *M*.[[MapData]], do
    a. If *p*.[[Key]] is not EMPTY and SameValue(*p*.[[Key]], *key*) is **true**, return *p*.[[Value]].
5. Let *p* be the Record { [[Key]]: *key*, [[Value]]: *value* }.
6. Append *p* to *M*.[[MapData]].
7. Return *value*.

SpiderMonkey

- Self-hosted JavaScript ⇒ *easy to start with an implementation*
  - Similar to the ordinary JavaScript
  - Learn-on-the-go by looking at implementation of similar proposals

- Let's walk through the good, the bad and the ugly of implementing this!

Implementation in SpiderMonkey

- ✓ Navigating the codebase → *SearchFox*
- ✓ Defining the function in self-hosted JavaScript

```
function MapGetOrInsert(key, value) {
  // Step 1.  Let M be the this value.
  var M = this;

  // Step 2.  Perform ? RequireInternalSlot(M, [[MapData]]).
  if (!IsObject(M) || (M = GuardToMapObject(M)) === null) {
    return callFunction(
      CallMapMethodIfWrapped,
      this,
      key,
      value,
      "MapGetOrInsert"
    );
  }
```

Implementation in SpiderMonkey

- ✓ Navigating the codebase → *SearchFox*
- ✓ Defining the function in self-hosted JavaScript
  - ✘ How to hook it to the engine? → *mentor's support was needed*

```
// This code is from: /js/src/builtin/MapObject.cpp
// ...
const JSFunctionSpec MapObject::methods[] = {
  // ...
  JS_FN("set", set, 2, 0),
  JS_FN("delete", delete_, 1, 0),
  JS_FN("keys", keys, 0, 0),
  JS_FN("values", values, 0, 0),
  JS_FN("clear", clear, 0, 0),
  JS_SELF_HOSTED_FN("forEach", "MapForEach", 2, 0),
  JS_FN("entries", entries, 0, 0),
  // ...
};
```

```
// ...
JS_SELF_HOSTED_FN("getOrInsert", "MapGetOrInsert", 2,0),
// ...
```

Implementation in SpiderMonkey

✓ Copy and adapt existing code to implement the body of the function
  - `MapGroupBy(items, callbackfn)` is similar

- Copying `MapGroupBy` gives a spec-accurate implementation, but web engines prioritize matching observable behavior, allowing room for optimization

Implementation in SpiderMonkey

4. For each Record { [[Key]], [[Value]] } *p* of *M*.[[MapData]], do
    a. If *p*.[[Key]] is not EMPTY and SameValue(*p*.[[Key]], *key*) is **true**, return *p*.[[Value]].

- We want to avoid implementing a full `Map` iteration
- For that, we can use `std_map_has` for key lookup
- `std_map_has` is defined in C++, but it was not exposed to the self-hosted JavaScript
    - Change the visibility of `std_map_has` from **`private`** to **`public`**
    - We were able to figure this out ourselves, but we were unsure whether it was OK to do so → *mentor support was needed*

Implementation in SpiderMonkey

2 Map.prototype.getOrInsertComputed (*key*, *callbackfn*)

When the getOrInsertComputed method is called the following steps are taken:

1. Let *M* be the **this** value.
2. Perform ? RequireInternalSlot(*M*, [[MapData]]).
3. If IsCallable(*callbackfn*) is **false**, throw a **TypeError** exception.
4. Set *key* to CanonicalizeKeyedCollectionKey(*key*).
5. For each Record { [[Key]], [[Value]] } *p* of *M*.[[MapData]], do
   a. If *p*.[[Key]] is not EMPTY and SameValue(*p*.[[Key]], *key*) is **true**, return *p*.[[Value]].
6. Let *value* be ? Call(*callbackfn*, **undefined**, « *key* »).
7. NOTE: The Map may have been modified during execution of *callbackfn*.
8. For each Record { [[Key]], [[Value]] } *p* of *M*.[[MapData]], do
   a. If *p*.[[Key]] is not EMPTY and SameValue(*p*.[[Key]], *key*) is **true**, then
      i. Set *p*.[[Value]] to *value*.
      ii. Return *value*.
9. Let *p* be the Record { [[Key]]: *key*, [[Value]]: *value* }.
10. Append *p* to *M*.[[MapData]].
11. Return *value*.

✓ **Implemented in self-hosted JS**

Implementation in SpiderMonkey: performance

✓ For performance, we rewrote both implementations in C++

- Review: To optimize performance, keep `getOrInsertComputed` self-hosted (faster JS-to-JS calls), but move `getOrInsert` to C++.

jandem added a comment. Edited · Feb 14 2025, 4:54 PM

It's almost end of day for me but a question before I review the patch in detail: have you looked at some micro-benchmarks for the self-hosted vs native implementations? I'm asking because `getOrInsertComputed` performs a call into JS code when the key doesn't exist in the `Map`, and calling JS code is a lot faster from self-hosted code than from C++. It's one of the main reasons for implementing builtins in self-hosted code.

Without any numbers to back this up, my suggestion would be to keep `getOrInsertComputed` as a self-hosted function but implement `getOrInsert` in C++. What do you think?

js/src/builtin/MapObject.cpp

959 Nit: just `"getOrInsert"`

Implementation in SpiderMonkey: self-hosted → C++

```
function MapGetOrInsert(key, value) {
  // Step 1.  Let M be the this value.
  var M = this;

  // Step 2.  Perform ? RequireInternalSlot(M, [[MapData]]).
  if (!IsObject(M) || (M = GuardToMapObject(M)) === null) {
    return callFunction(
      CallMapMethodIfWrapped,
      this,
      key,
      value,
      "MapGetOrInsert"
    );
  }

  // Step 3.  Set key to CanonicalizeKeyedCollectionKey(key).
  // Step 4.  For each Record { [[Key]], [[Value]] } p of M.[[MapData]], do
  for (var p of allowContentIter(callFunction(std_Map_entries, M))) {
  // Step 4.a.  If p.[[Key]] is not empty and SameValue(p.[[Key]], key) is true, return p.[[Value]].
    if (SameValueZero(p[0], key)) {
      return p[1];
    }
  }


  // Step 5.  Let p be the Record { [[Key]]: key, [[Value]]: value }.
  // Step 6.  Append p to M.[[MapData]].
  callFunction(std_Map_set, M, key, value);

  // Step 7.  Return value.
  return value;
}
```

```
bool MapObject::getOrInsert(JSContext* cx, const Value& key, const Value& val,
                            MutableHandleValue rval) {
  HashableValue k;
  if (!k.setValue(cx, key)) {
    return false;
  }

  bool needsPostBarriers = isTenured();
  if (needsPostBarriers) {
    if (!PostWriteBarrier(this, k)) {
      ReportOutOfMemory(cx);
      return false;
    }
    // Use the Table representation which has post barriers.
    if (const Table::Entry* p = Table(this).getOrAdd(cx, k, val)) {
      rval.set(p->value);
    } else {
      return false;
    }
  } else {
    // Use the PreBarrieredTable representation which does not.
    if (const PreBarrieredTable::Entry* p =
            PreBarrieredTable(this).getOrAdd(cx, k, val)) {
      rval.set(p->value);
    } else {
      return false;
    }
  }
  return true;
}
```

Implementation in SpiderMonkey

- We didn't know before the review that we need to register `getOrInsert` and `getOrInsertComputed` as common property names for efficient access and reuse in the engine

```
MACRO_(getOrInsert, "getOrInsert")

MACRO_(getOrInsertComputed, "getOrInsertComputed")
```

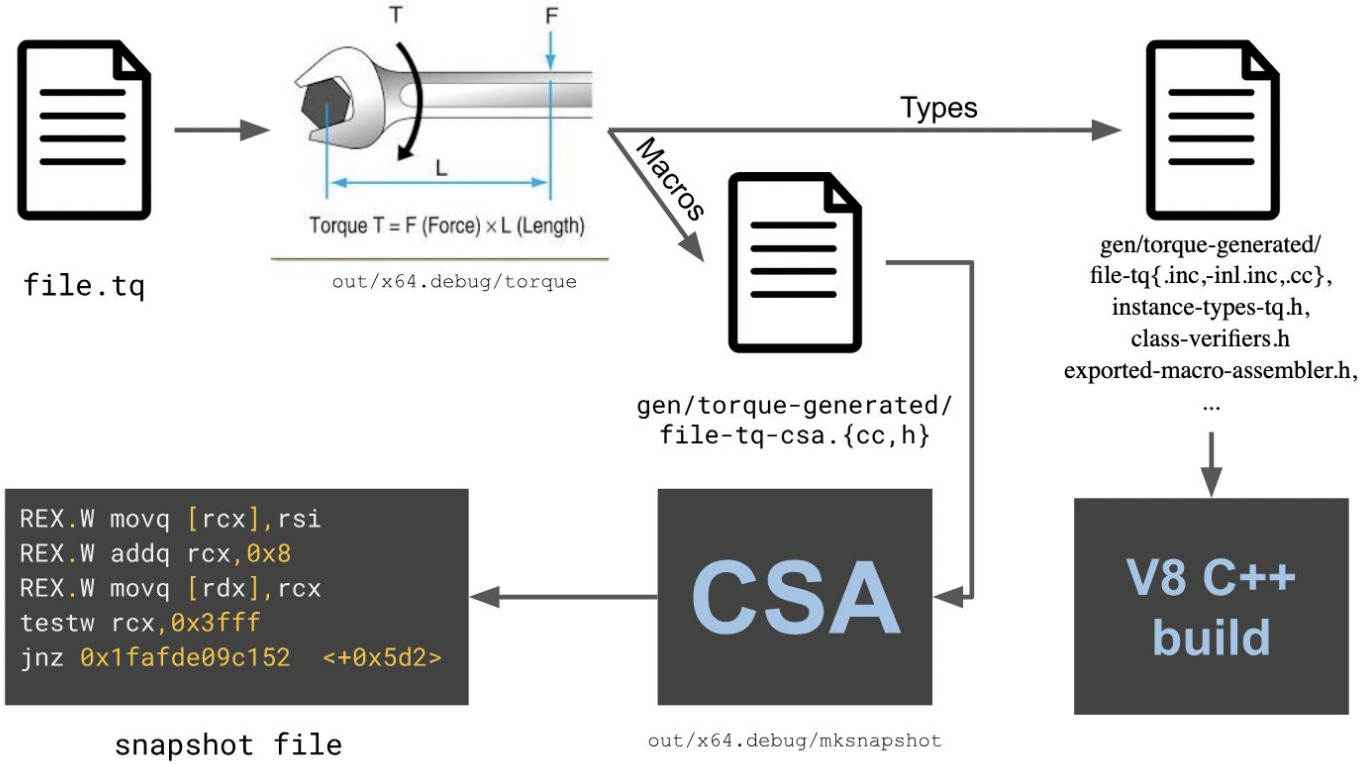
- **Now we have a working implementation → *Ready to ship in Firefox Nightly!***
  - After handling some Nightly-specific code → *mentor's support was needed*

Implementation in V8

- Torque ⇒ a high level abstraction to write built-in functions in
  - TypeScript-like syntax
  - A lot of built-in functions to model by
  - A guide from V8 developers team to get started
  - VSCode extension (syntax highlighting, error messages)

- So, is it easier than self-hosted JavaScript in SpiderMonkey?
  - Not really...

Torque is transpiled to CSA/C++



https://v8.dev/docs/torque#how-torque-generates-code

Implementation in V8

- `MapGroupBy(items: JSANy, callback: JSAny)` was the only `Map`-related built-in implemented in Torque
  - We intended to use that code as a reference, but it was difficult to draw inspiration from.
  - Instead: we looked at the implementation of built-ins in `Set`
  - This enabled us to implement the first steps of the spec

1. Let *M* be the **this** value.
2. Perform ? RequireInternalSlot(*M*, [[MapData]]).
3. Set *key* to CanonicalizeKeyedCollectionKey(*key*).
4. For each Record { [[Key]], [[Value]] } *p* of *M*.[[MapData]], do

Implementation in V8

- Variable names in Torque
  - Unused variables need to be prefixed with an underscore → *difficult to figure out without mentor*
  - This got fixed, and the transpiler now shows better error messages

- Difficult-to-understand error messages after Torque code is transpiled to CSA/C++

Implementation in V8

4. For each Record { [[Key]], [[Value]] } *p* of *M*.[[MapData]], do
    a. If *p*.[[Key]] is not EMPTY and SameValue(*p*.[[Key]], *key*) is **true**, return *p*.[[Value]].

- How to implement `SameValue(p.[[Key]], key)` is **true**?
- Need to handle comparisons → *challenging without a mentor's support*

Implementation in V8

- How to call built-in functions written in CSA/C++ from Torque?
  - How to call

    `TF_BUILTIN(MapPrototypeSet, CollectionsBuiltinAssembler)`

    from Torque?
  - Our solution: redeclare

    `TNode<JSMap> CollectionsBuiltinAssembler::MapSet`

    with the same body
- *Without a mentor's support, it was challenging to understand whether and what we should expose ourselves vs. what has already been exposed*

Implementation in V8

- We have figured out how to write functions in CSA/C++ and expose them to Torque
- Optimizing the implementation → *straightforward*

Mentor support on SpiderMonkey

Special thanks to Daniel Minor who mentored the case study in SpiderMonkey!

Temporal in Boa and V8

Case Study 3

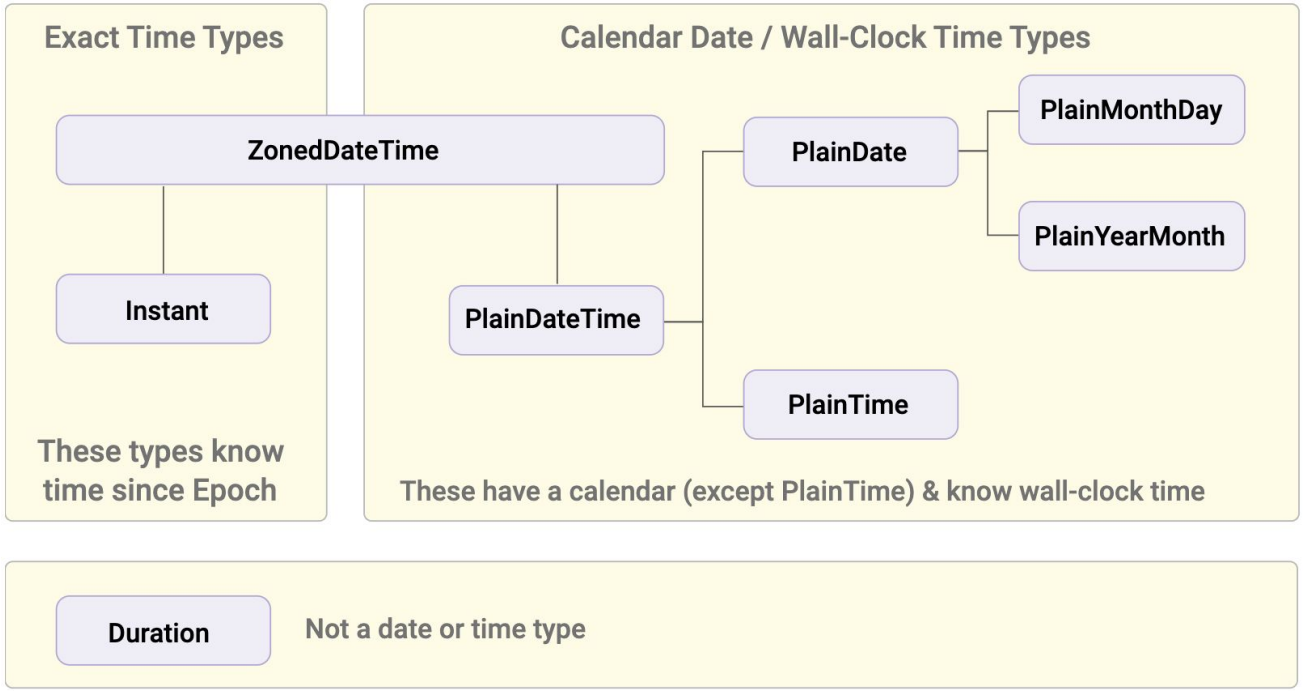
Shane Carr

Convener, ECMA TC39-TG2
Chair, Unicode ICU4X-TC
GitHub: @sffc

Mentor, January-May 2025

What is the Temporal proposal?

- Brings modern date/time types to ECMAScript
- High demand from developers



Ten reasons why implementing Temporal is challenging

1. Giant spec: bigger than all of ECMA-402
2. Spec volatility creates a moving target
3. Many many edge cases
4. Integration with time zone database
5. Needs calendrical calculations
6. Creates a large codebase to maintain
7. Hard (but not impossible) for a single person to implement
8. Behavior needs to match Intl behavior
9. Everything needs to be reasonably efficient
10. And all of this needs to be interoperable across engines

Paths toward implementing Temporal in an engine

Implementation Directly in the Engine

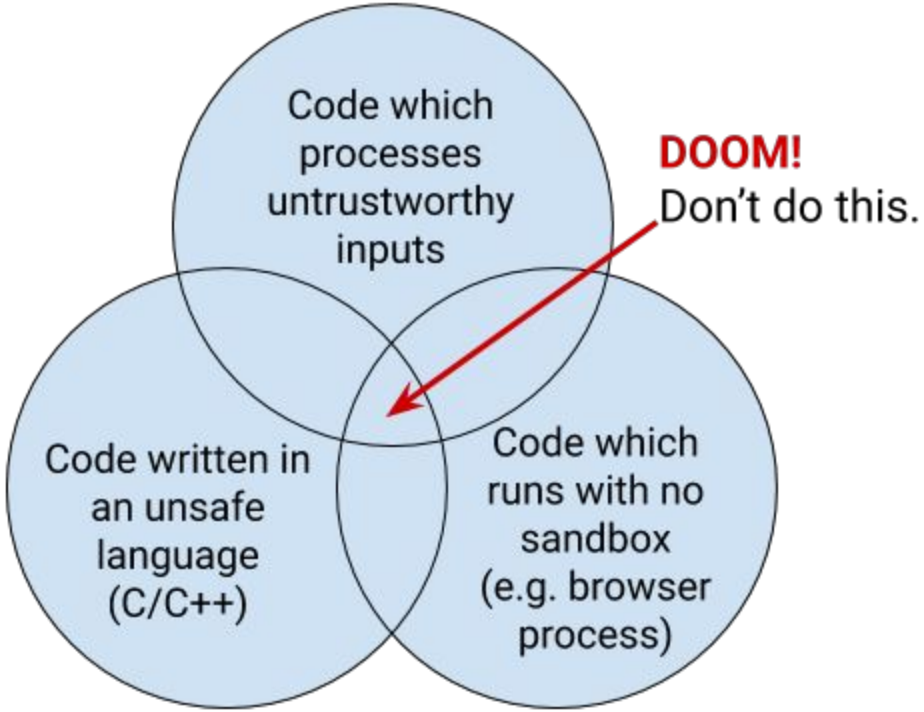
Implementation with a Third-Party Library

Advantages of using a third-party library

Leveraging a library limits engine-owned code to the JS bindings layer.

- Lower upfront review cost and maintenance cost for the engine
- Bigger ecosystem to maintain and improve the library
- Success stories: Intl, Regular Expressions, JSON, Zlib, ...

"Rule of Two"


Code which processes untrustworthy inputs
Code written in an unsafe language (C/C++)
Code which runs with no sandbox (e.g. browser process)
DOOM!
Don't do this.

Using Rust

From the Google Security Blog:

Why We Chose to Bring Rust into Chromium

Our goal in bringing Rust into Chromium is to **provide a simpler** (no IPC) and **safer** (less complex C++ overall, no memory safety bugs in a sandbox either) **way to satisfy the rule of two, in order to speed up development** (less code to write, less design docs, less security review) **and improve the security** (increasing the number of lines of code without memory safety bugs, decreasing the bug density of code) **of Chrome**. And we believe that we can use third-party Rust libraries to work toward this goal.

Boa and Temporal_rs

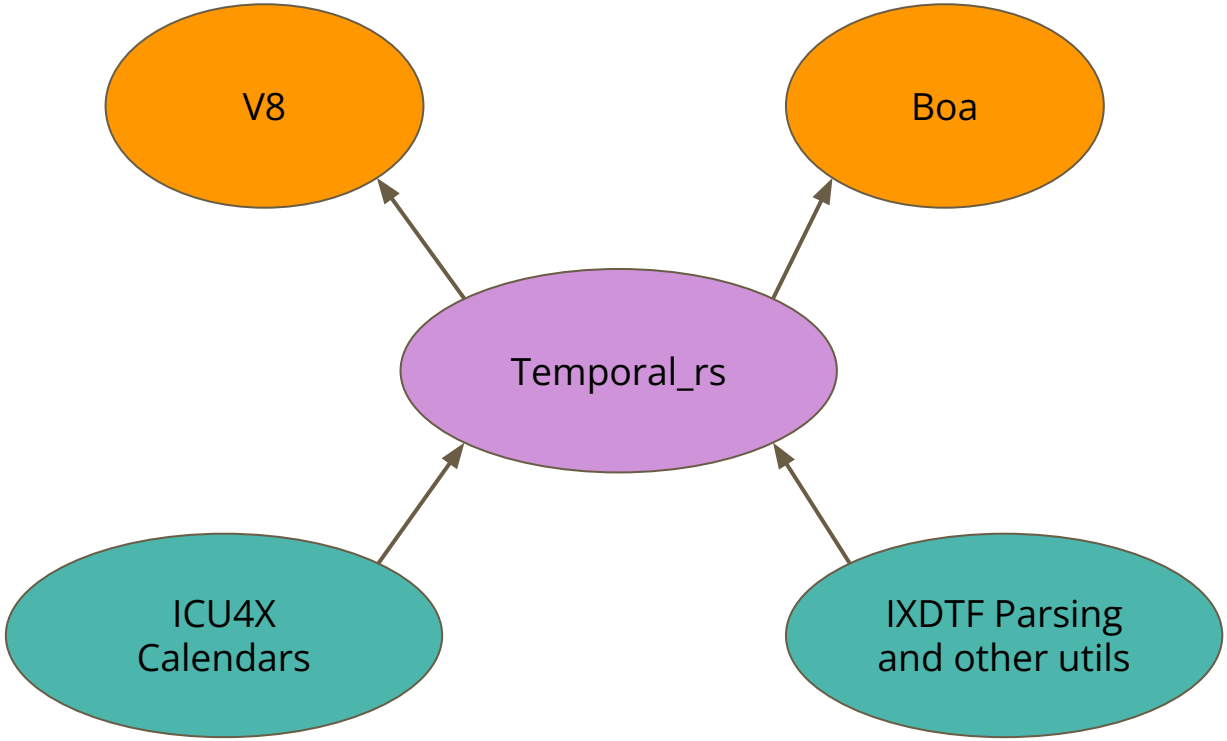
Boa: Javascript lexer, parser and interpreter written in Rust

Temporal_rs: a standalone Rust project used in Boa

Opportunities for V8:

- Library aims to be aligned with the Temporal spec
- Lead developer of Boa is a Temporal proposal champion (Jason Williams)
- Being written in Rust solves the "Rule of Two" in V8
- Gives an onramp for more Rust projects like ICU4X

Architecture of Temporal in V8 and Boa



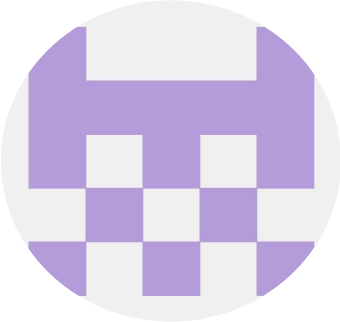
Working with students from UiB

4-month project to bring Temporal_rs to spec compliance.

Shane mentored 6 students in the form of a weekly one-hour meeting, plus a Rust mini-course at the start of the semester.

Students contributed primarily to Temporal_rs as well as Test262 and Boa.

Special thanks to Kevin Ness and Philip Chimento for responding quickly to PRs.



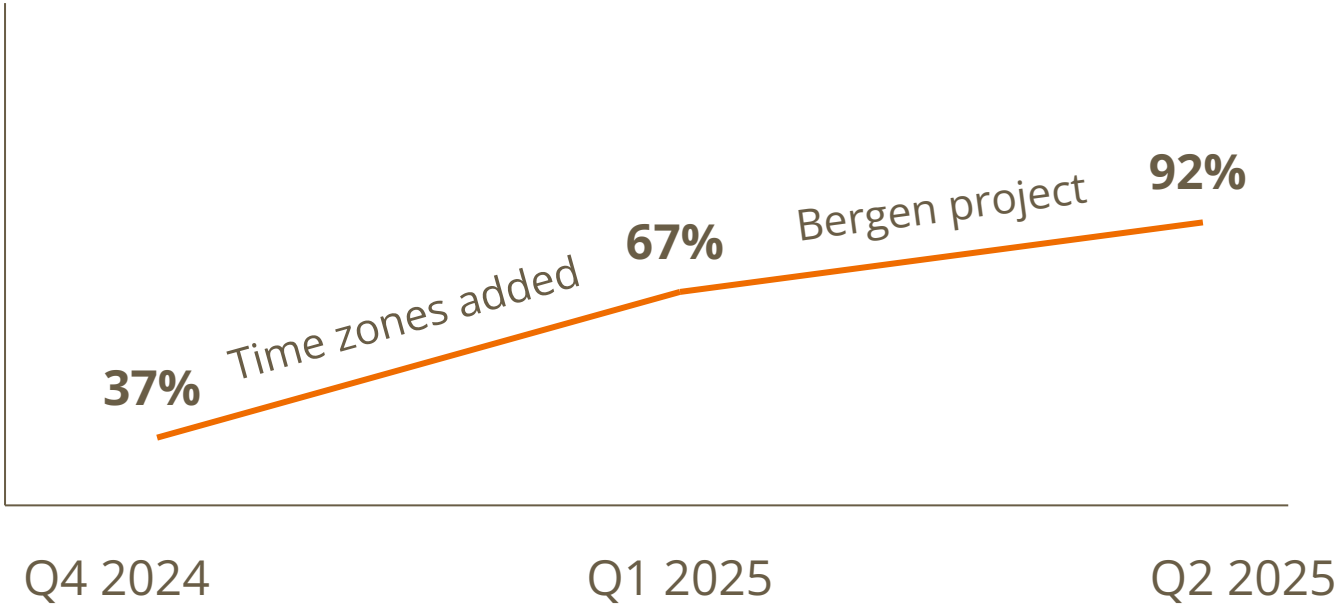
Temporal_rs Code Samples

```rust
// Temporal_rs Code:

impl PlainMonthDay {
    pub fn to_plain_date(
        &self,
        year: Option<PartialDate>
    ) -> TemporalResult<PlainDate> {
        // … spec logic …
    }
}
```

```rust
// Boa Code:

impl PlainYearMonth {
   fn to_plain_date(this: &JsValue,
        args: &[JsValue],
        context: &mut Context
    ) -> JsResult<JsValue> {
        let ym = this
            .as_object()
            .and_then(JsObject::downcast_ref)?;
        //… field validation …
        let result = ym.inner.to_plain_date()?;
        create_temporal_date(result, …)
            .map(Into::into)
    }
}
```

Code that Henrik wrote in [33a4ab96cd40ef7538a1d64461c503fb2bb06869]

Boa / Temporal_rs Test262 Coverage



V8 is working, too!

Temporal.Instant on Temporal_rs landed in CL 6508228! (May 8, 2025)

```
d8> first = new Temporal.Instant(1000000000000000000n)
2001-09-09T01:46:40Z

d8> second = new Temporal.Instant(2000000000000000000n)
2033-05-18T03:33:20Z

d8> second.since(first, {largestUnit: "hours"})
PT277777H46M40S
```

Lessons from Rust library-based Temporal in Boa/V8

Successes:

1. Overall, using a library reduces the barrier to entry.
2. The library also is more amenable to long-term maintenance.
3. Using Rust results in higher quality code that is easier to review.

Lessons learned / areas for more discussion:

1. Library is separated from Test262; longer feedback cycle.
2. Harder to write Rust, requiring extra work ahead of time.
3. Handling of slots and option validation works differently.
4. Risk of too many implementations that aren't actually different.

★ Tutorial on how to start contributing to SpiderMonkey and V8

Leveraging standalone Rust code:

- ★ Lower barrier to entry
- ★ Modularity and reuse

Come to our breakout session on **Wednesday** at **12:00**