SIMD.js Moving towards Stage 3

John McCutchan (Google)
Peter Jensen (Intel)
Dan Gohman (Mozilla)
Abhijith Chatra (Microsoft)
Daniel Ehrenberg (Google)

Agenda

- Developments Since Stage 2 Approval
- Notable design decisions
- Questions for the committee
- Implementation Status
- Specification Status

Developments Since Stage 2 Approval

Boolean vectors

- **Previously**, the result of SIMD.Float32x4.greaterThan was a SIMD. Int32x4 vector, with -1/0 for true/false
- **Now**, new boolean vector types, e.g. SIMD.Bool32x4, represent boolean results and values
- Used for select and logic operations
- More efficiently implementable on some architectures
- Not simply Boolx4 because the registers in implementations may be represented differently based on width.

Unsigned operations

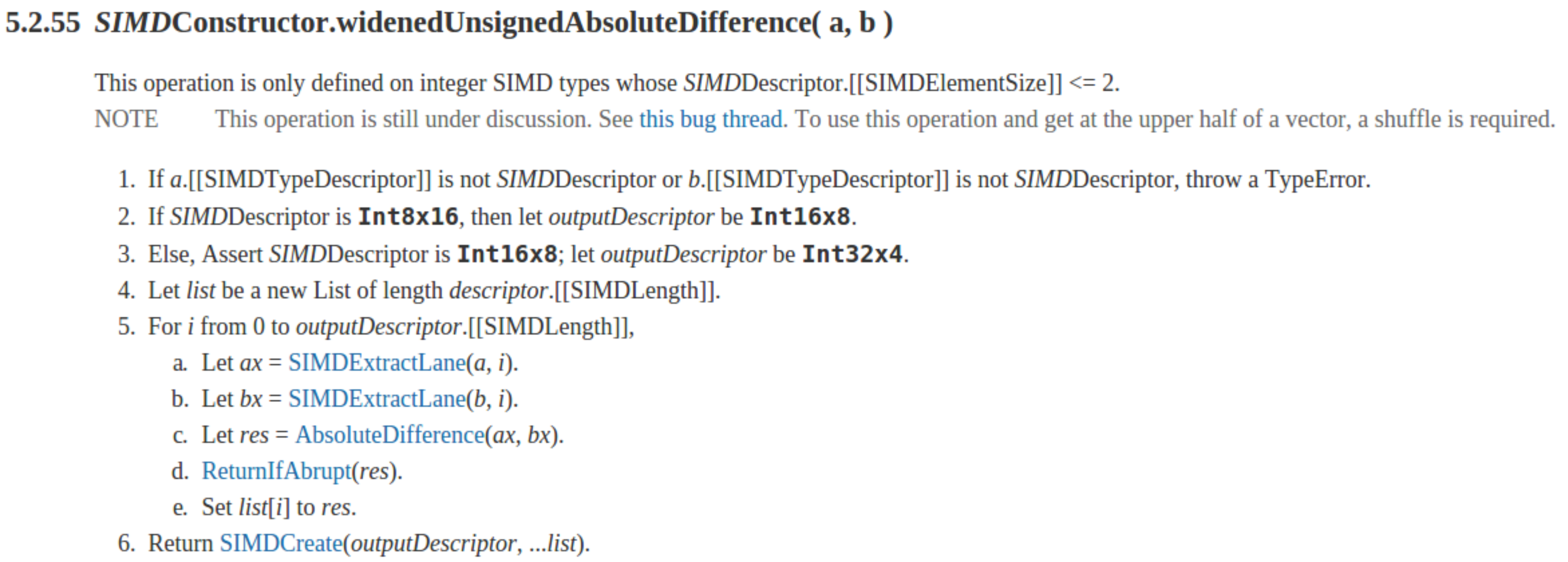
- Unsigned comparisons
- Unsigned saturating add/sub
- Unsigned extractLane
- No changes needed for constructor, replaceLane because coercion will wrap unsigned values to the appropriate signed value
- No separate unsigned type

Postponed features

- Float64x2--we couldn’t find an important use case with improved performance
- Int64x2--Not needed due to boolean vectors, and really not needed because Float64x2 is out
- selectBits--minimal utility due to select, and efficiently implementable in terms of other boolean operations

sumOfAbsoluteDifferences replacement

- widenedAbsoluteDifference, unsignedHorizontalSum, absoluteDifference
- Seeking implementation feedback: applications and benchmarks
- Replaces sumOfAbsoluteDifferences (slow on ARM)



5.2.55 *SIMD*Constructor.widenedUnsignedAbsoluteDifference(a, b)

This operation is only defined on integer SIMD types whose *SIMD*Descriptor.[[SIMDElementSize]] <= 2.

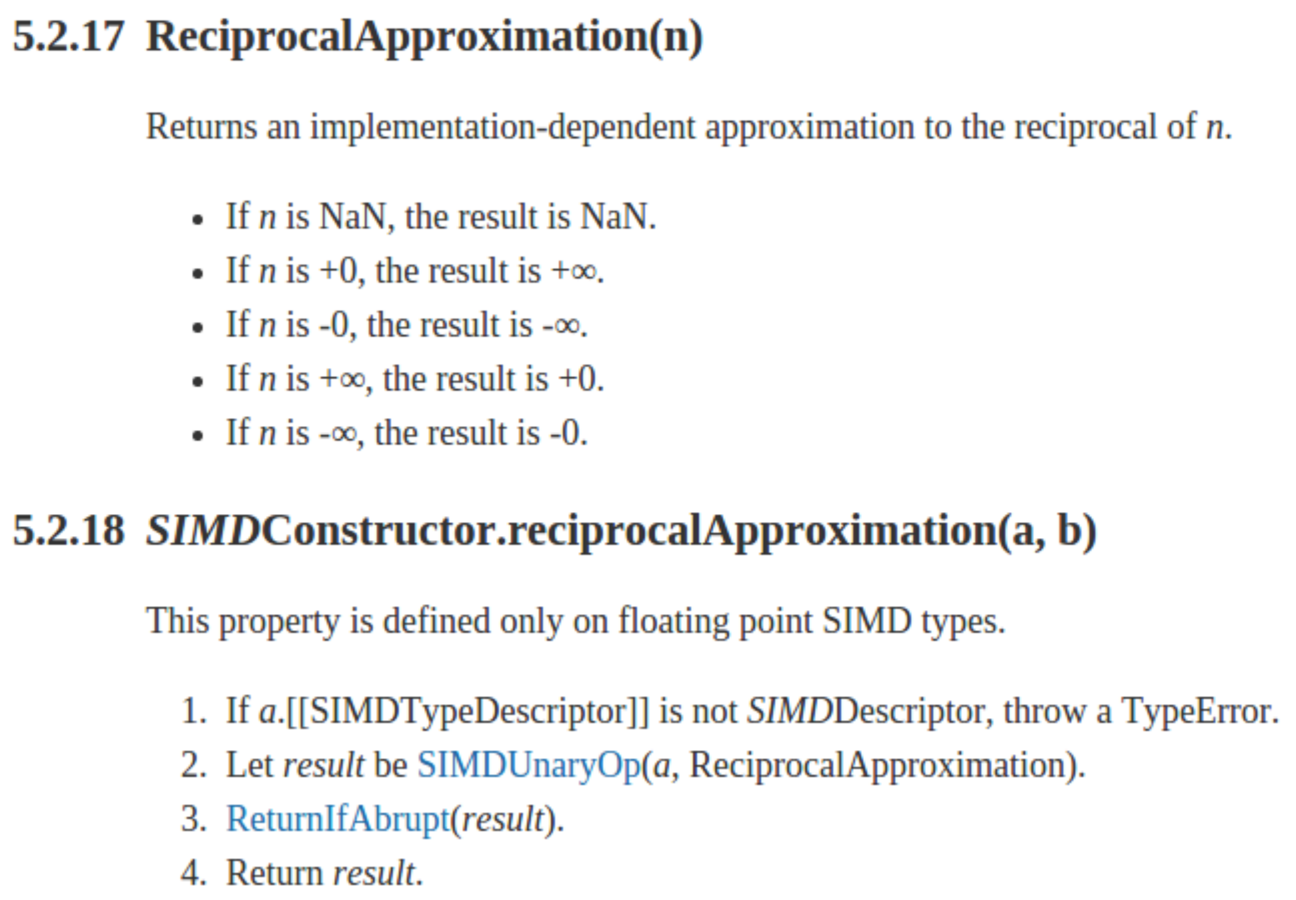
NOTE This operation is still under discussion. See this bug thread. To use this operation and get at the upper half of a vector, a shuffle is required.

1. If *a*.[[SIMDTypeDescriptor]] is not *SIMD*Descriptor or *b*.[[SIMDTypeDescriptor]] is not *SIMD*Descriptor, throw a TypeError.
2. If *SIMD*Descriptor is **`Int8x16`**, then let *outputDescriptor* be **`Int16x8`**.
3. Else, Assert *SIMD*Descriptor is **`Int16x8`**; let *outputDescriptor* be **`Int32x4`**.
4. Let *list* be a new List of length *descriptor*.[[SIMDLength]].
5. For *i* from 0 to *outputDescriptor*.[[SIMDLength]],
    a. Let *ax* = SIMDExtractLane(*a*, *i*).
    b. Let *bx* = SIMDExtractLane(*b*, *i*).
    c. Let *res* = AbsoluteDifference(*ax*, *bx*).
    d. ReturnIfAbrupt(*res*).
    e. Set *list*[*i*] to *res*.
6. Return SIMDCreate(*outputDescriptor*, ...*list*).

Other spec changes

- Homoiconic toString()
  - SIMD.Float32x4(1, 2, 3, 4).toString() => “SIMD.Float32x4(1, 2, 3, 4)”
- Shift operations max out at 0/-1, rather than wrapping around
- Ops like reciprocalApproximation are loosely specified, like Math.sin
- Removed operations on DataView--TypedArray ops suffice
- Operations on subnormals may flush to 0, unlike ES scalars
- Various minor spec bug fixes

New reciprocalApproximation definition



5.2.17 ReciprocalApproximation(n)

Returns an implementation-dependent approximation to the reciprocal of *n*.

- If *n* is NaN, the result is NaN.
- If *n* is +0, the result is +∞.
- If *n* is -0, the result is -∞.
- If *n* is +∞, the result is +0.
- If *n* is -∞, the result is -0.

5.2.18 *SIMD*Constructor.reciprocalApproximation(a, b)

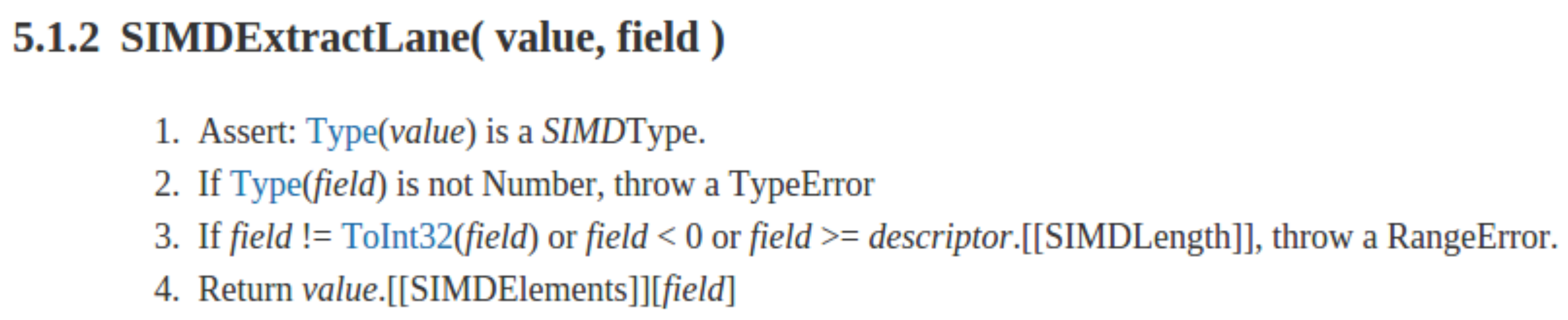
This property is defined only on floating point SIMD types.

1. If *a*.[[SIMDTypeDescriptor]] is not *SIMD*Descriptor, throw a TypeError.
2. Let *result* be SIMDUnaryOp(*a*, ReciprocalApproximation).
3. ReturnIfAbrupt(*result*).
4. Return *result*.

Notable design decisions

Strong type check on lanes

- Lanes are required to be Int32s and not implicitly coerced

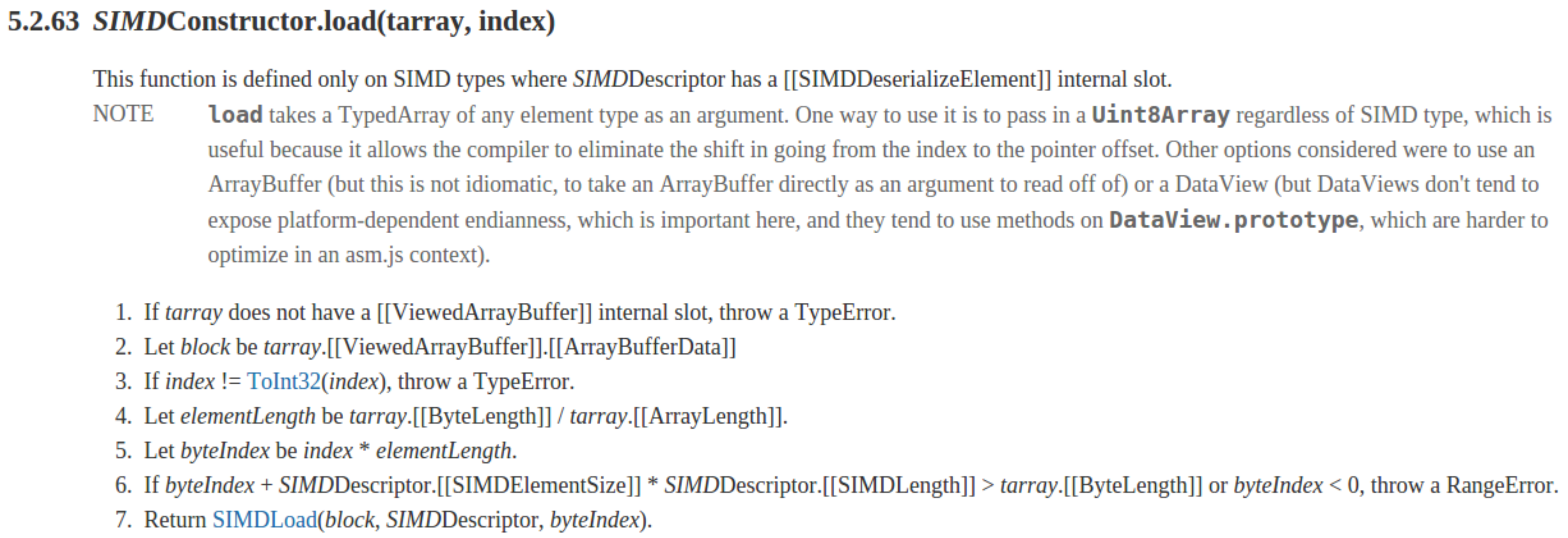


5.1.2 SIMDExtractLane(value, field)

1. Assert: Type(*value*) is a *SIMD*Type.
2. If Type(*field*) is not Number, throw a TypeError
3. If *field* != ToInt32(*field*) or *field* < 0 or *field* >= *descriptor*.[[SIMDLength]], throw a RangeError.
4. Return *value*.[[SIMDElements]][*field*]

load and store operating on TypedArrays

- load and store take TypedArrays as arguments and permit array element type mismatch with SIMD type



5.2.63 *SIMD*Constructor.load(tarray, index)

This function is defined only on SIMD types where *SIMD*Descriptor has a [[SIMDDeserializeElement]] internal slot.

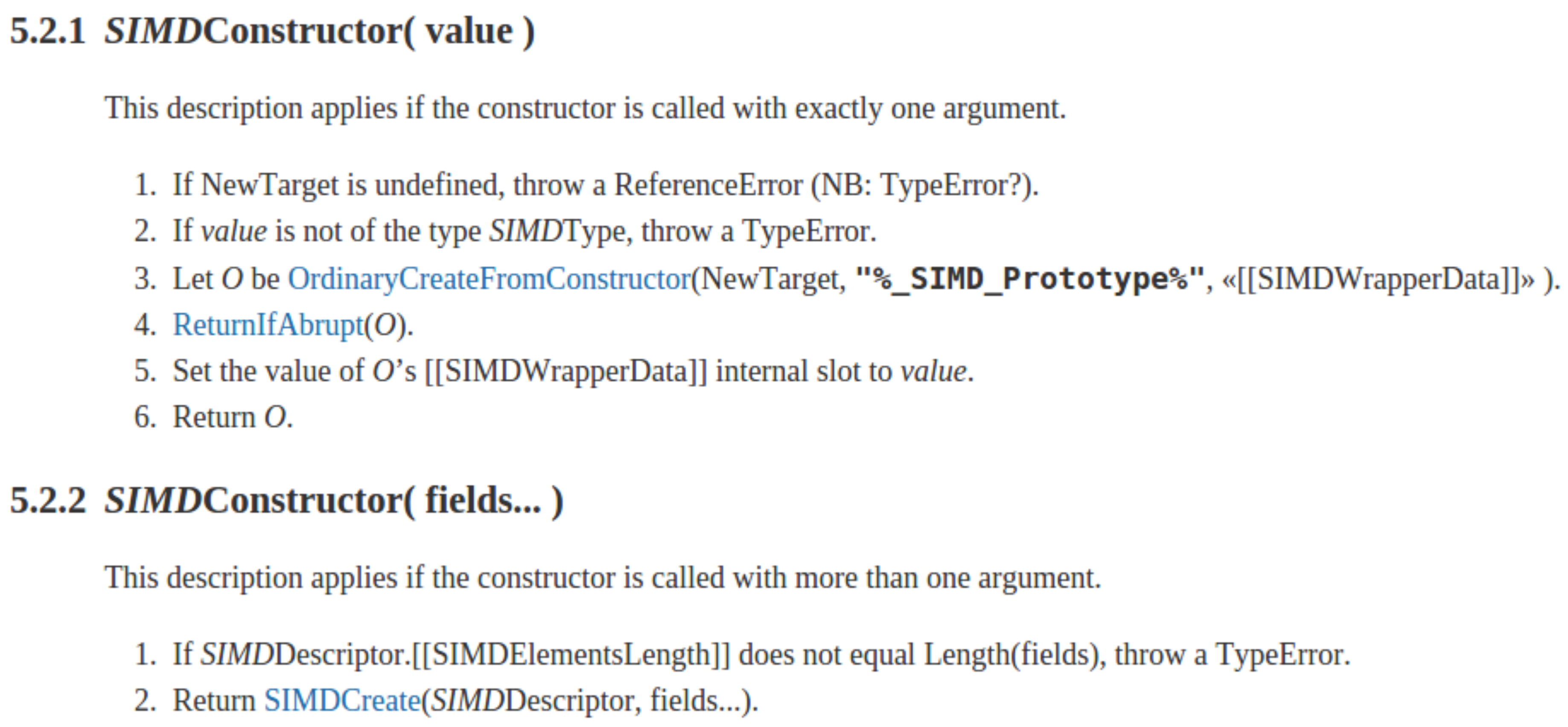
NOTE `load` takes a TypedArray of any element type as an argument. One way to use it is to pass in a `Uint8Array` regardless of SIMD type, which is useful because it allows the compiler to eliminate the shift in going from the index to the pointer offset. Other options considered were to use an ArrayBuffer (but this is not idiomatic, to take an ArrayBuffer directly as an argument to read off of) or a DataView (but DataViews don't tend to expose platform-dependent endianness, which is important here, and they tend to use methods on `DataView.prototype`, which are harder to optimize in an asm.js context).

1. If *tarray* does not have a [[ViewedArrayBuffer]] internal slot, throw a TypeError.
2. Let *block* be *tarray*.[[ViewedArrayBuffer]].[[ArrayBufferData]]
3. If *index* != ToInt32(*index*), throw a TypeError.
4. Let *elementLength* be *tarray*.[[ByteLength]] / *tarray*.[[ArrayLength]].
5. Let *byteIndex* be *index* * *elementLength*.
6. If *byteIndex* + *SIMD*Descriptor.[[SIMDElementSize]] * *SIMD*Descriptor.[[SIMDLength]] > *tarray*.[[ByteLength]] or *byteIndex* < 0, throw a RangeError.
7. Return SIMDLoad(*block*, *SIMD*Descriptor, *byteIndex*).

Questions for the committee

Wrapper constructors

- Should wrapper constructors be explicitly [[Construct]]-able, like Number, or not, like Symbol?



5.2.1 *SIMD*Constructor(value)

This description applies if the constructor is called with exactly one argument.

1. If NewTarget is undefined, throw a ReferenceError (NB: TypeError?).
2. If *value* is not of the type *SIMD*Type, throw a TypeError.
3. Let *O* be OrdinaryCreateFromConstructor(NewTarget, **`"%_SIMD_Prototype%"`**, «[[SIMDWrapperData]]»).
4. ReturnIfAbrupt(*O*).
5. Set the value of *O*'s [[SIMDWrapperData]] internal slot to *value*.
6. Return *O*.

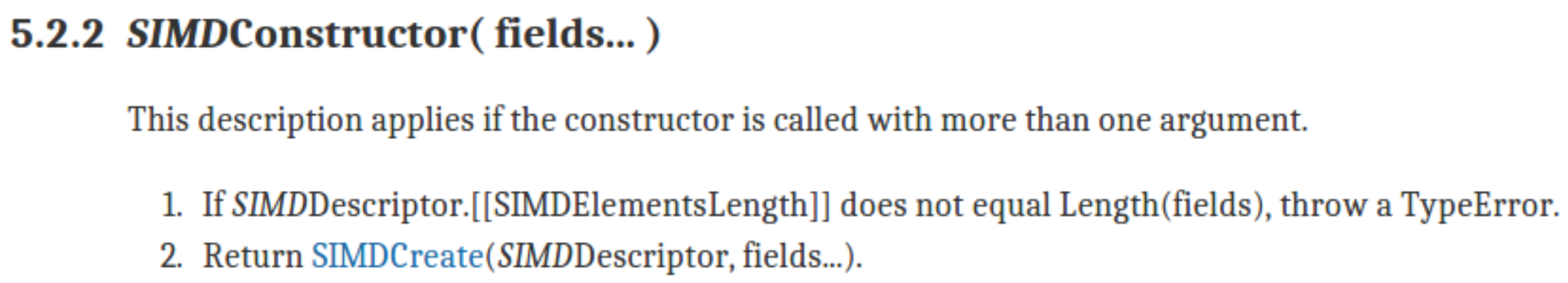
5.2.2 *SIMD*Constructor(fields...)

This description applies if the constructor is called with more than one argument.

1. If *SIMD*Descriptor.[[SIMDElementsLength]] does not equal Length(fields), throw a TypeError.
2. Return SIMDCreate(*SIMD*Descriptor, fields...).

Spec language "innovation" acceptable?

- Rest parameters
- *SIMD* as a spec meta-variable



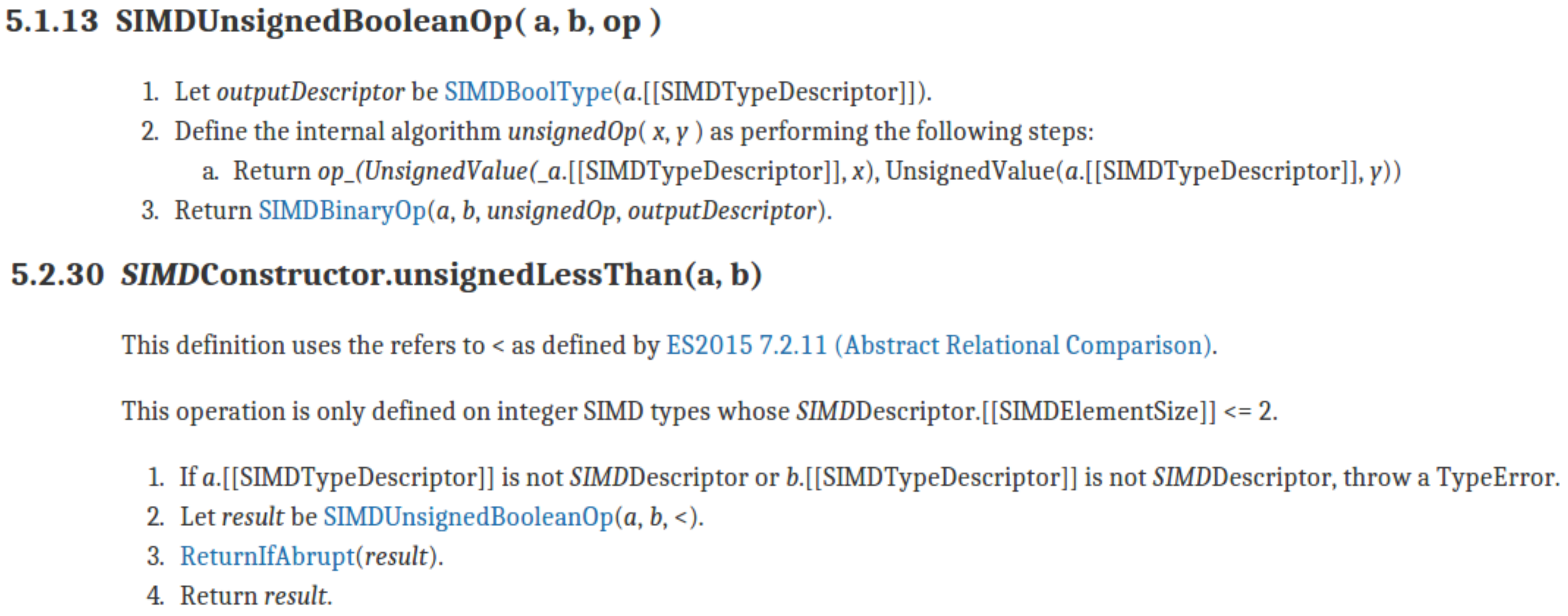
5.2.2 *SIMD*Constructor(fields…)

This description applies if the constructor is called with more than one argument.

1. If *SIMD*Descriptor.[[SIMDElementsLength]] does not equal Length(fields), throw a TypeError.
2. Return SIMDCreate(*SIMD*Descriptor, fields…).

Spec language “innovation” acceptable?

- Higher-order internal algorithms, including closures and infix ops



5.1.13 SIMDUnsignedBooleanOp(a, b, op)

```
1. Let outputDescriptor be SIMDBoolType(a.[[SIMDTypeDescriptor]]).
2. Define the internal algorithm unsignedOp( x, y ) as performing the following steps:
    a. Return op_(UnsignedValue(_a.[[SIMDTypeDescriptor]], x), UnsignedValue(a.[[SIMDTypeDescriptor]], y))
3. Return SIMDBinaryOp(a, b, unsignedOp, outputDescriptor).
```

5.2.30 *SIMD*Constructor.unsignedLessThan(a, b)

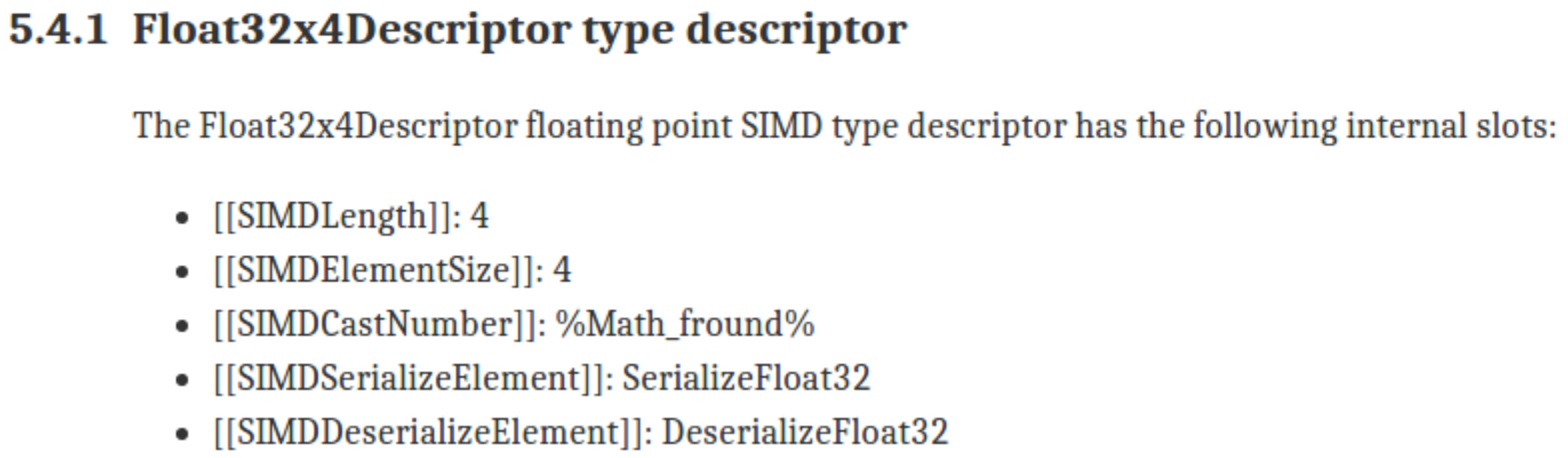
This definition uses the refers to < as defined by ES2015 7.2.11 (Abstract Relational Comparison).

This operation is only defined on integer SIMD types whose *SIMD*Descriptor.[[SIMDElementSize]] <= 2.

```
1. If a.[[SIMDTypeDescriptor]] is not SIMDDescriptor or b.[[SIMDTypeDescriptor]] is not SIMDDescriptor, throw a TypeError.
2. Let result be SIMDUnsignedBooleanOp(a, b, <).
3. ReturnIfAbrupt(result).
4. Return result.
```

Spec language “innovation” acceptable?

- Refactor TypedArray spec language like SIMD.js numerical types, or the reverse?



5.4.1 Float32x4Descriptor type descriptor

The Float32x4Descriptor floating point SIMD type descriptor has the following internal slots:

- [[SIMDLength]]: 4
- [[SIMDElementSize]]: 4
- [[SIMDCastNumber]]: %Math_fround%
- [[SIMDSerializeElement]]: SerializeFloat32
- [[SIMDDeserializeElement]]: DeserializeFloat32

Separate spec?

Implementation Status

Firefox implementation status

- In Firefox nightly:
- Float32x4 and Int32x4 entirely implemented and optimized in JavaScript (regular and asm.js) on x86 and x64.
- Missing boolean vectors
- Other SIMD types (Int16x8, Int8x16, Float64x2) partially implemented in the interpreter only (ergo not optimized). The newer APIs (SAD) haven't been implemented yet.
- All SIMD types are implemented as value objects, at the moment.

Microsoft Edge implementation status

- Majority of SIMD.*.* APIS supported.
- Some of the new APIS need to be implemented such as ExtractLane and ReplaceLane, and unsigned operations
- Asm.js optimization is complete (minus new api support).
- Non-asm.js optimization we plan to start soon.

V8 implementation status

- Intel object implementation will not be used for V8 and work has started to implement value types from scratch. Intel code generation may be rewritten to the new model.
- Bill Budge has added a Float32x4 type with correct value semantics and basic operations, without acceleration, behind a flag.

Specification Status

Specification Status

- SIMD.js Specification v0.7.2
- Updated polyfill and tests validate all operations, basic value semantics
- SIMD.js is ready for reviewers and and editor comments/signoff
- Hope to move to Stage 3 in the September meeting

Questions!

References

Spec, polyfill, tests and benchmarks repository
https://github.com/tc39/ecmascript_simd

Published Paper on Dart + JS prototype implementations
John McCutchan, Haitao Feng, Nicholas Matsakis, Zachary Anderson, Peter Jensen (2014) A SIMD Programming Model for Dart, JavaScript, and Other Dynamically Typed Scripting Languages, Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing
https://sites.google.com/site/wpmvp2014/paper_18.pdf

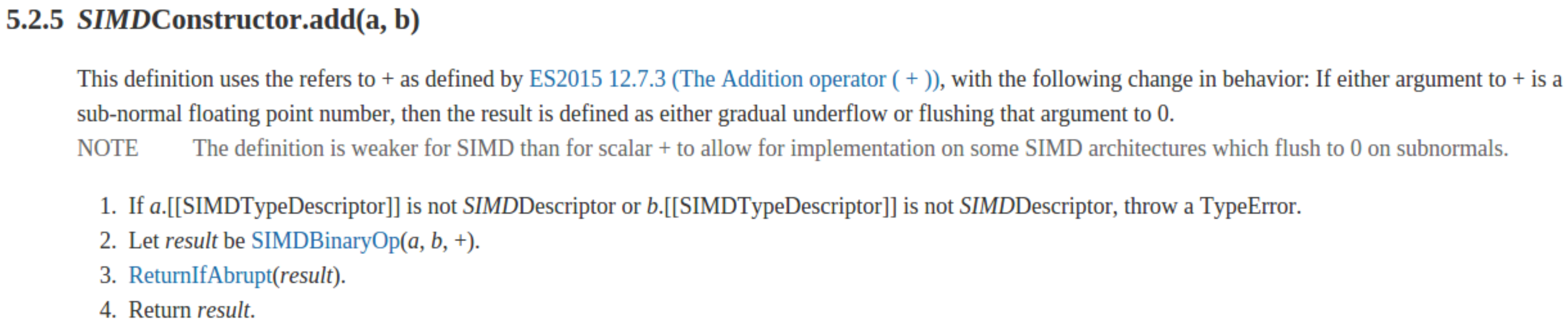
Published Paper: SIMD in JavaScript via C++ and Emscripten
Peter Jensen, Ivan Jibaja, Ningxin Hu, Dan Gohman, John McCutchan (2015)
Workshop on Programming Models for SIMD/Vector Processing - WPMVP'15
https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx3cG12cDIwMTV8Z3g6NTkzYWE2OGNlNDAyMTRjOQ

HTML5 Developer Conference Presentation (May 2014)
http://peterjensen.github.io/html5-simd/html5-simd.html#/

SIMD documentation on MDN
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SIMD

Wikipedia
http://en.wikipedia.org/wiki/SIMD

New subnormal behavior



5.2.5 *SIMD*Constructor.add(a, b)

This definition uses the refers to + as defined by ES2015 12.7.3 (The Addition operator (+)), with the following change in behavior: If either argument to + is a sub-normal floating point number, then the result is defined as either gradual underflow or flushing that argument to 0.

NOTE The definition is weaker for SIMD than for scalar + to allow for implementation on some SIMD architectures which flush to 0 on subnormals.

1. If *a*.[[SIMDTypeDescriptor]] is not *SIMD*Descriptor or *b*.[[SIMDTypeDescriptor]] is not *SIMD*Descriptor, throw a TypeError.
2. Let *result* be SIMDBinaryOp(*a*, *b*, +).
3. ReturnIfAbrupt(*result*).
4. Return *result*.