Reference Types for Wasm

Proposal

Andreas Rossberg

Dfinity

Motivation

Need to reference host objects inside Wasm

…e.g. JS, DOM objects (Web)
…e.g. actors, capabilities (Dfinity)

Currently, requires ref-int bijection at boundary

…slow, brittle, may leak memory

Host binding proposal tried to abstract over this, becomes very messy

Proposal

Add opaque reference type

New form of value type

…usable for locals, globals, parameters, results

Can be passed from/to embedder

Can be put into tables

Can not be constructed or accessed in Wasm!

Splits off minimum part from GC proposal

Main Insight

Does not imply GC in Wasm!

GC support is only necessary if embedder's host references are GC'ed

Other embeddings may have non-GCed pointers as host references

Types

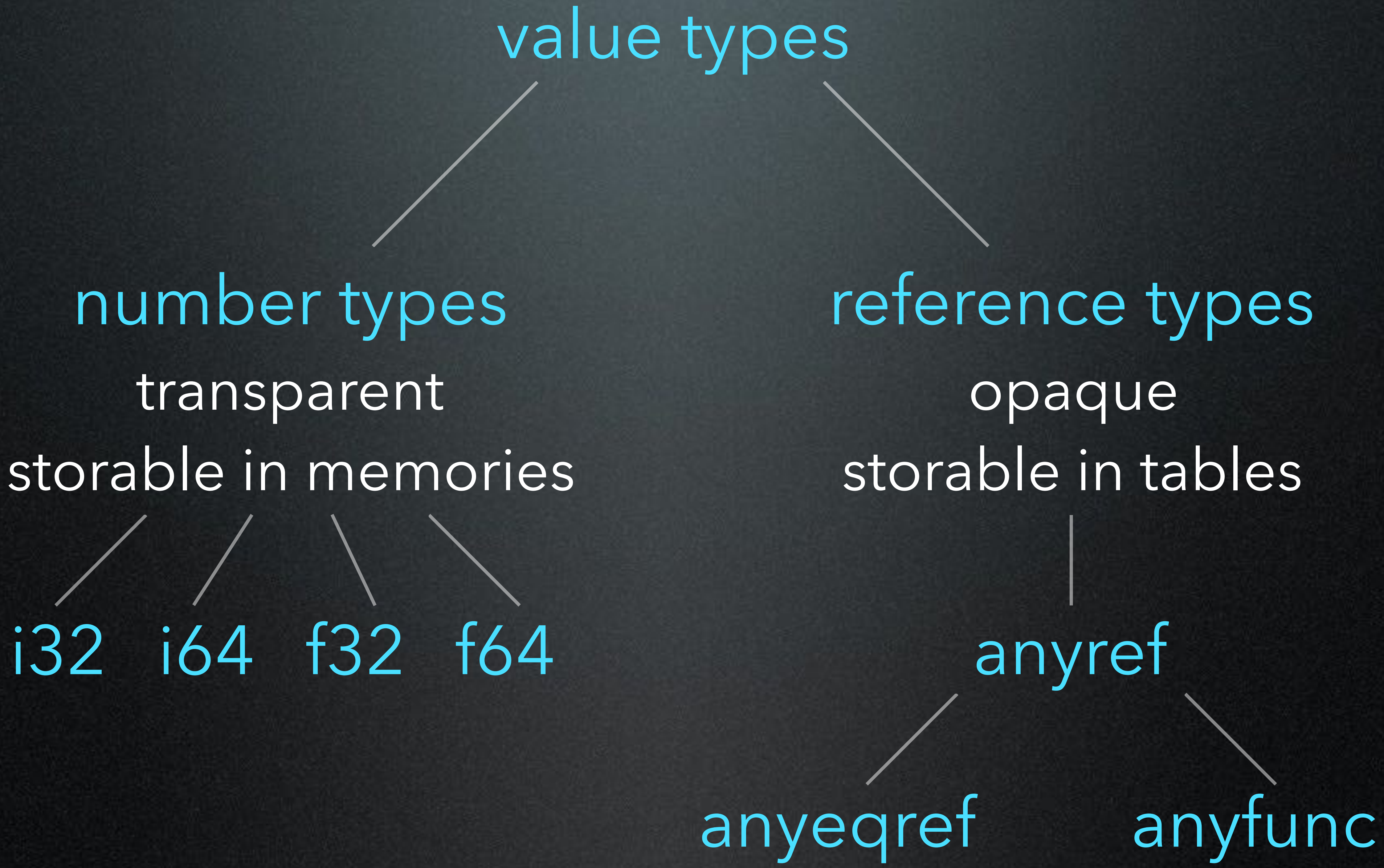
Introduce a new type anyref

…both a value type and an element type

Vice versa, anyfunc becomes value type, too

…also is a reference type (subtype of anyref)

…element type = reference type


value types
number types
transparent
storable in memories
i32
i64
f32
f64
reference types
opaque
storable in tables
anyref
anyeqref
anyfunc

Instructions

ref.null – creates a null value

ref.isnull – check for null

ref.eq – compare references (anyeqref only)

table.get – load reference from table

table.set – store reference into table

Comparisons

Not all references should be comparable, e.g.,

…JS strings are reference types in Wasm,
but reference equality would reveal
implementation details of the engine

…functions with reference equality make
various optimisations hard or impossible

Solution: distinguish subtype anyeqref

Tables

Element type can now be either anyfunc, anyref, or anyeqref

Only useful with multiple tables, so allow those

Table instructions (including call_indirect) take table index immediate

(Should we rename to table.call?)

Typing

ref.null : [] → [nullref]   (coming back to this)

ref.isnull : [anyref] → [i32]

ref.eq : [anyeqref, anyeqref] → [i32]

table.get $x : [i32] → [t] (iff $x : table of t)

table.set $x : [i32, t] → [] (iff $x : table of t)

Subtyping

Subtyping is applicable everywhere (and is non-coercive)

…anyfunc ≤ anyref
…anyeqref ≤ anyref

```
(func (param $x anyeqref) (result anyref) (get_local $x))
```

```
(table $t 10 anyref)
(func (local $f anyfunc) (table.set $t (i32.const 1) (get_local $f))
```

Easy extension to validation algorithm

…pop checks for subtype instead of type equality
…select returns lub

Typing **ref.null**

Two options:

…utilise subtyping will nullref type

…or require type annotation

Typing **ref.null**

ref.null : [] → [nullref]

…where nullref ≤ any.*

Subtyping does the rest naturally

Similar to handling in C++, Java, Scala, etc.

Typing **ref.null**

ref.null <reftype> : [] → [reftype]

Require (redundant) type annotation

Involves ad-hoc predicate check once we have non-nullable ref types

Generated value is always the same

JS API

Only exported Wasm functions and null match anyfunc

JS objects, functions, symbols, null match anyeqref

JS objects, functions, strings, symbols, null match anyref

Open Question: Allow all JS values for anyref?

Proposal Status

prose spec :	✓
formal spec :	✓
interpreter :	✓
tests :	✓
JS API :	(✓)

(ready for stage 2 or 3?)

Remaining steps

Resolve null typing

Resolve JS values allowed for anyref

Finalise opcode assignment

Implementations

JS API tests

Future extensions

Typed function references

…ref <*functype*> ≤ anyfunc

…**call_ref** : [t1*, ref (func t1* t2*)] → [t2*]

…**ref.func** $f : [] → [ref <*functype*>]

Type imports/exports

…to distinguish different host types