Two-Pronged Plan

Ross Tate

The Two Prongs

JS API for Async/Await

- No change to wasm
- Only changes JS API
- Solely supports interop with JS's async/await paradigm
- Ships first and soon

Wasm for First-Class Stacks

- Changes wasm
- No change to JS API???
- Supports language/runtime features utilizing stacks
- Ships second

Preliminary Design

JS API for Async/Await

- interface Stacklet {
  constructor();
  }
- namespace WebAssembly {…
  Promise<Instance> instantiateAsync(
  Module moduleObject,
  optional object asyncImportObject,
  optional Stacklet stackletObject,
  optional object syncImportObject);
  }

Async Imports and Exports

- instance = instantiateAsync(module, asyncImports);
- succeeds whenever instantiate(module, asyncImports) would
- when an exported function of instance is called
  - it might call an (asynchronous) imported function
  - but that function can return a promise (even if its return type was i32…)
  - in which case the instance awaits the promise
    and the exported function returns a promise

Stacklets and Implementation

- By default, instantiateAwait creates a new Stacklet
- An export call first switches to the instance's stacklet
  - instance's stacklet remembers the caller's stacklet
- An asynchronous import first switches back to the caller's stacklet
- If the call returns a promise, then export awaits the promise
- If not, switches back to the instance's stacklet
  - Instance's stacklet remembers the caller/callee's stacklet
- Completed export call switches to caller's stacklet
- Traps during export call switch to caller's stacklet

Synchronous Imports and Exports

- instance = instantiateAsync(module, asyncImps, stacklet, syncImps);
- succeeds when syncImps are all exports of async modules w/ same stacklet
- calls to imports taken from syncImp do not stack switch

- particularly useful for lazy module loading

Motivation

Why JS API First

Business Reason

Async/await JS interop is an urgently needed high-priority feature

Design Reason

Fewer design variables makes for faster consensus

Engineering Reason

Engines can focus substantial effort on solely multi-stack challenges

Why Better for Wasm Design

More balanced consideration of tradeoffs

More informed accounting of engineering constraints and benefits

More room to develop comprehensive design