WEBASSEMBLY illustrated

exploring some mental models and implementations

Takenobu T.

WIP

Rev. 0.00.0

NOTE

- Please refer to the official documents in detail.
- This information is based on "WebAssembly Specification Release 1.0 (Draft, last updated Oct 31, 2018)".
- This information is current as of Nov, 2018. Still work in progress.

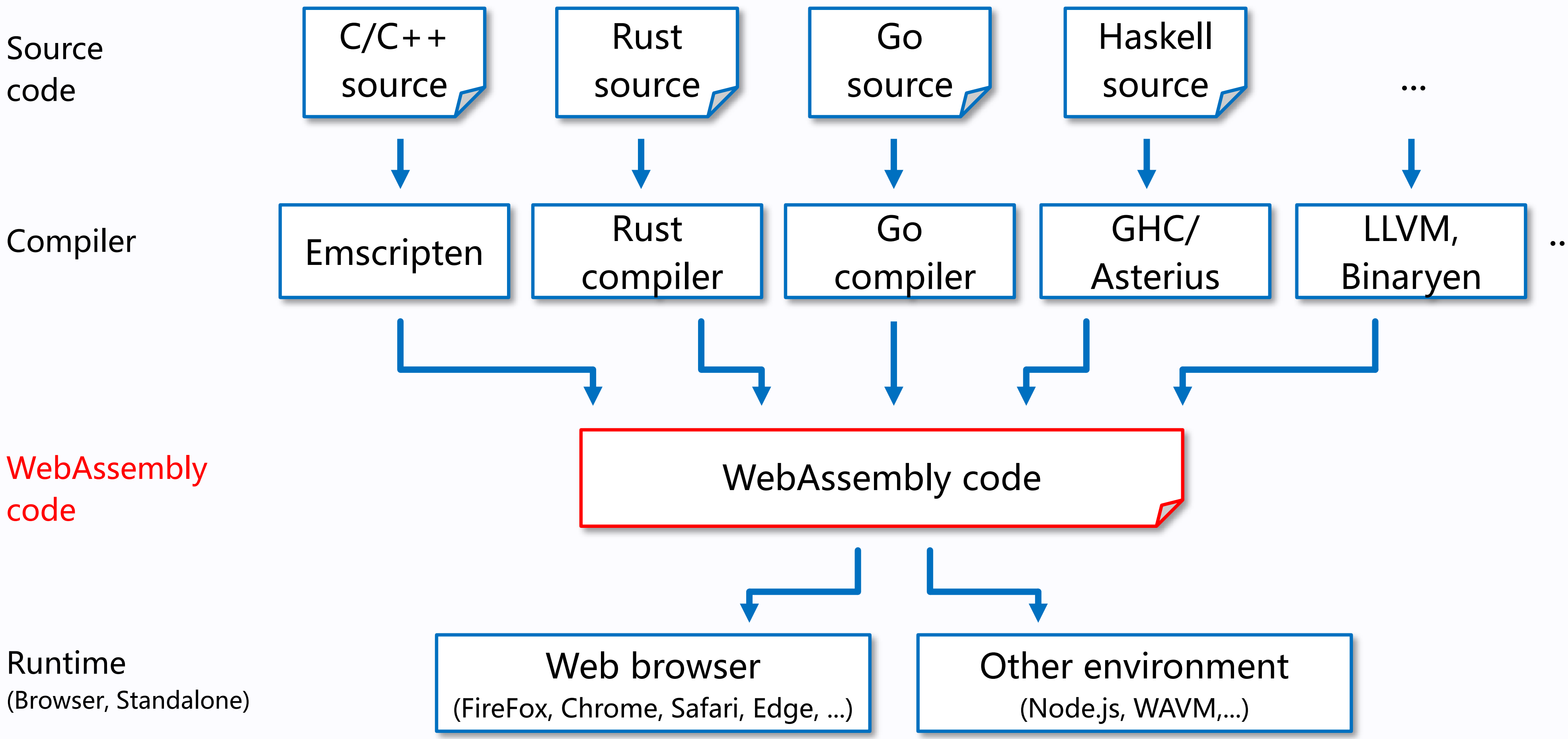
Contents

1. Introduction

1. Introduction

Basic concept

WebAssembly is a code format



[spec, 1.1]

WebAssembly is a safe, portable, low-level code format.

References : [1] Ch.1.1

WebAssembly code

Text format (S-expression)

```
(module
  (func (export "add2")
    (param $x i64)
    (param $y i64)
    (result i64)
    (i64.add
      (get_local $x)
      (get_local $y))))
```

Bynary format

```
0x0061736d010000 ...
```

[spec, 1.2.1]

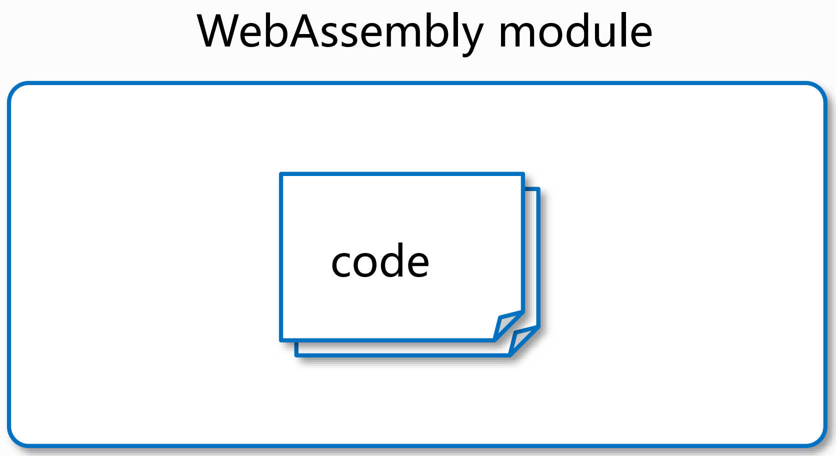
WebAssembly encodes a low-level, assembly-like programming language.

[spec, 2.1]

WebAssembly is a programming language that has multiple concrete representations (its binary format and the text format). Both map to a common structure.

References : [1] Ch.1.1

WebAssembly module

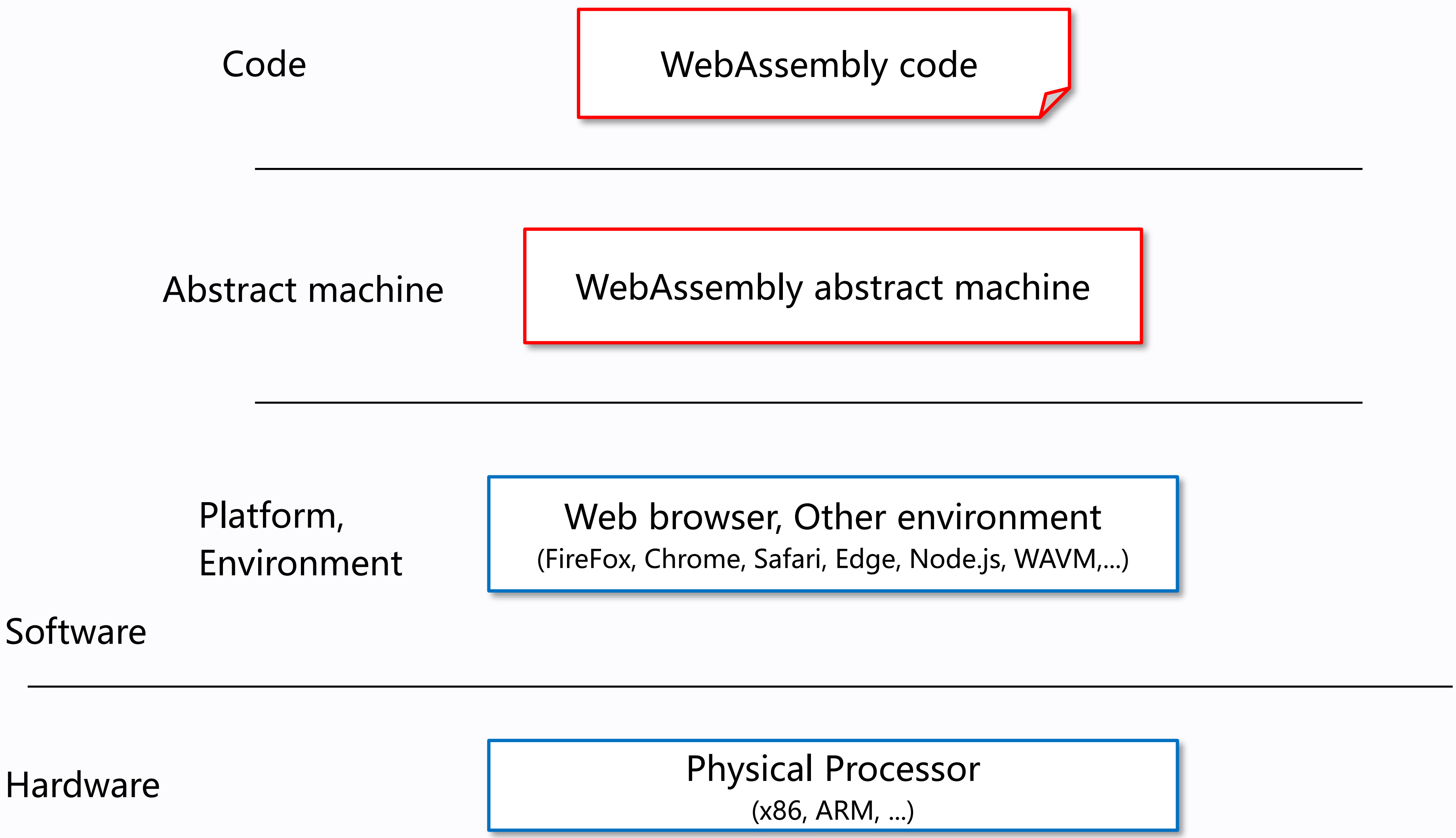


[module]

The distributable, loadable, and executable unit of code in WebAssembly is called a module.

References : [1] Ch.1.1

Architecture layer



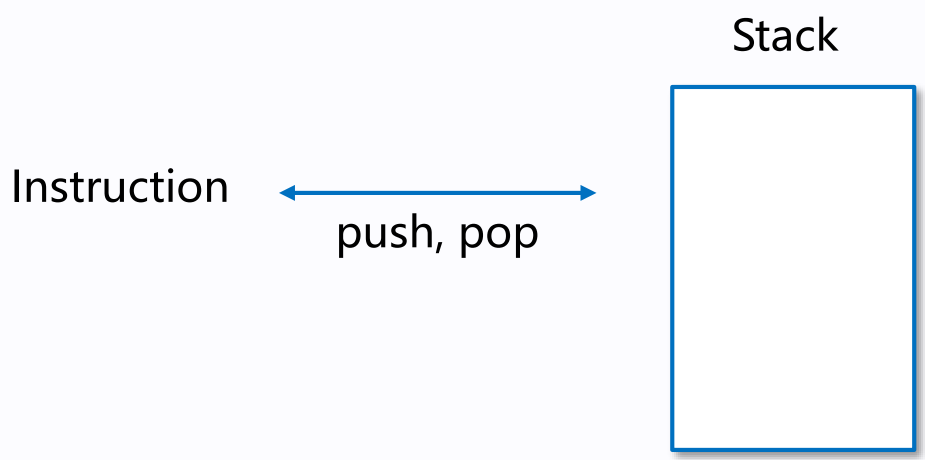
[spec, 1.1.2]

WebAssembly is a virtual instruction set architecture (virtual ISA).

References : [1] Ch.1.1

Computational model

The computational model of WebAssembly is based on a stack machine.
Code ha meirei no sequence. inorder.
Anmokuno operand stack manipulate.
Stack, anonymous register



[rational, why]

[spec, 1.2.1]

The computational model of WebAssembly is based on a stack machine.

[spec, 2.4]

WebAssembly code consists of sequences of instructions.
Its computational model is based on a stack machine in that instructions manipulate values on an implicit operand stack, consuming (popping) argument values and producing or returning (pushing) result values.

References : [1] Ch.1.2

Value types

Integers	32bit
Integers	64bit
Floating-point numbers	32bit
Floating-point numbers	64bit

[spec, 1.2.1]

WebAssembly provides only four basic value types.

singed / unsiged shikibetsu nashi.
Boolean, memory address, Integer 32bit.

References : [1] Ch.1.2

Instructions

[spec, 2.4]

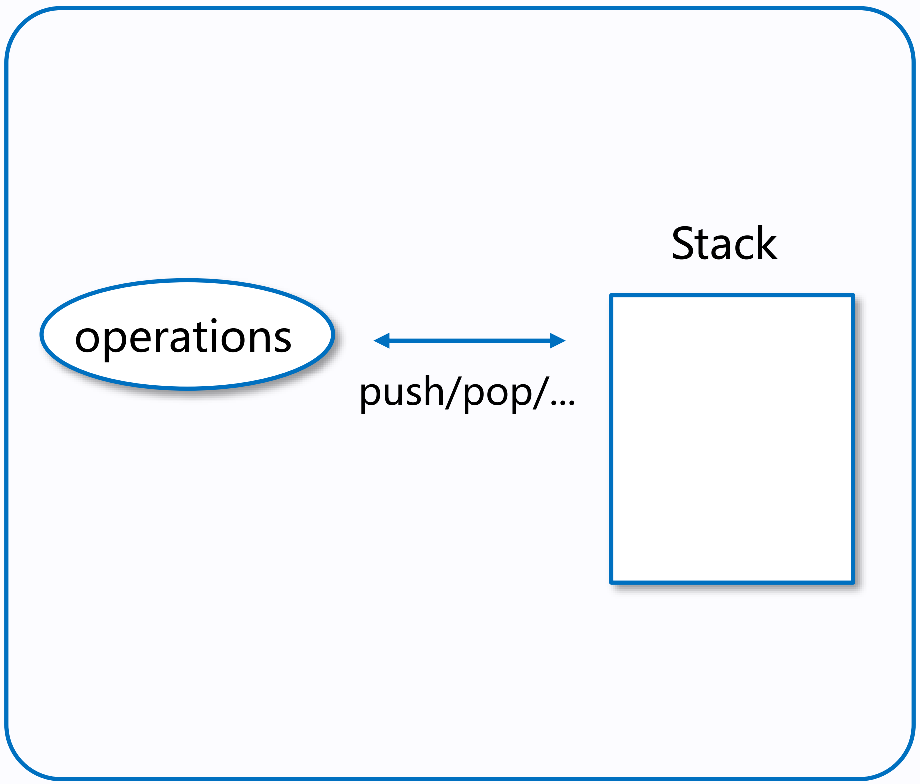
@@@ see tegaki, No.7

WebAssembly code consists of sequences of instructions. Its computational model is based on a stack machine in that instructions manipulate values on an implicit operand stack, consuming (popping) argument values and producing or returning (pushing) result values.

References : [1] Ch.1.2

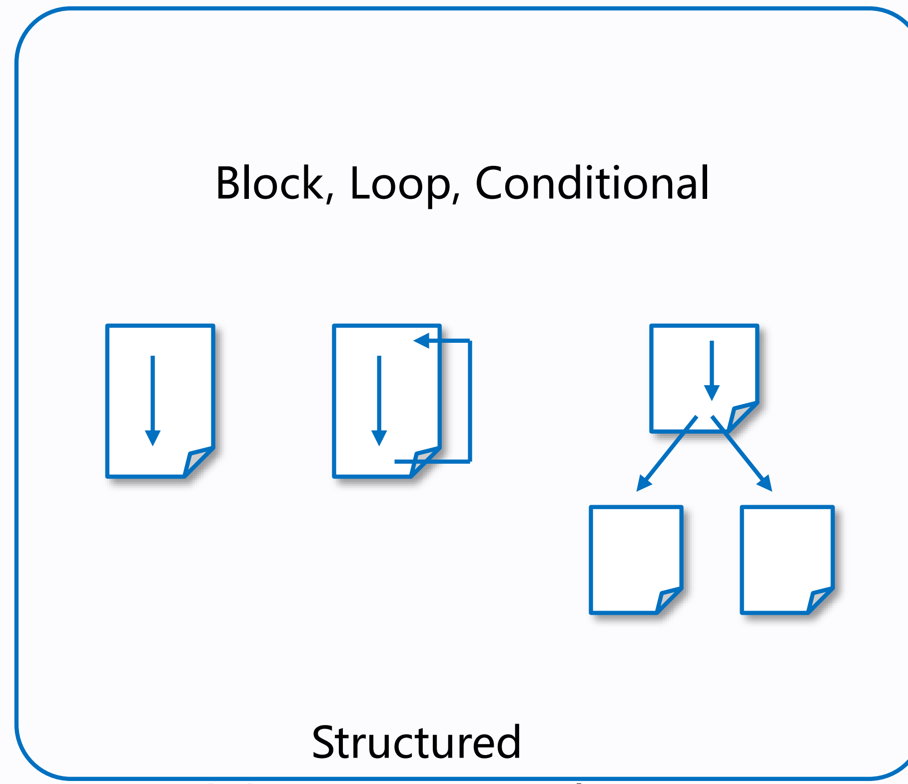
Instructions

Simple instructions



2shurui

Control instructions



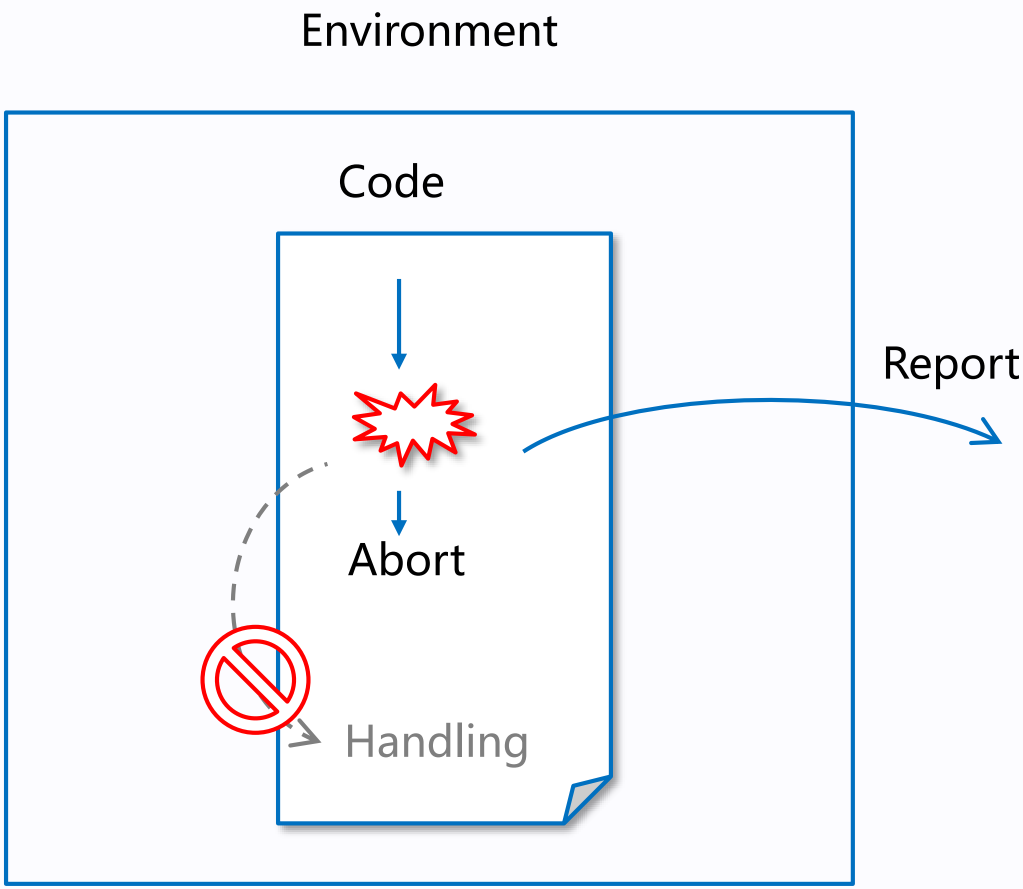
Branch ha, construct only target

Instructions fall into two main categories.
Simple instructions perform basic operations on data.
Control instructions alter control flow.

[spec, 1.2.1]

References : [1] Ch.1.2

Trap



[spec, 1.2.1]

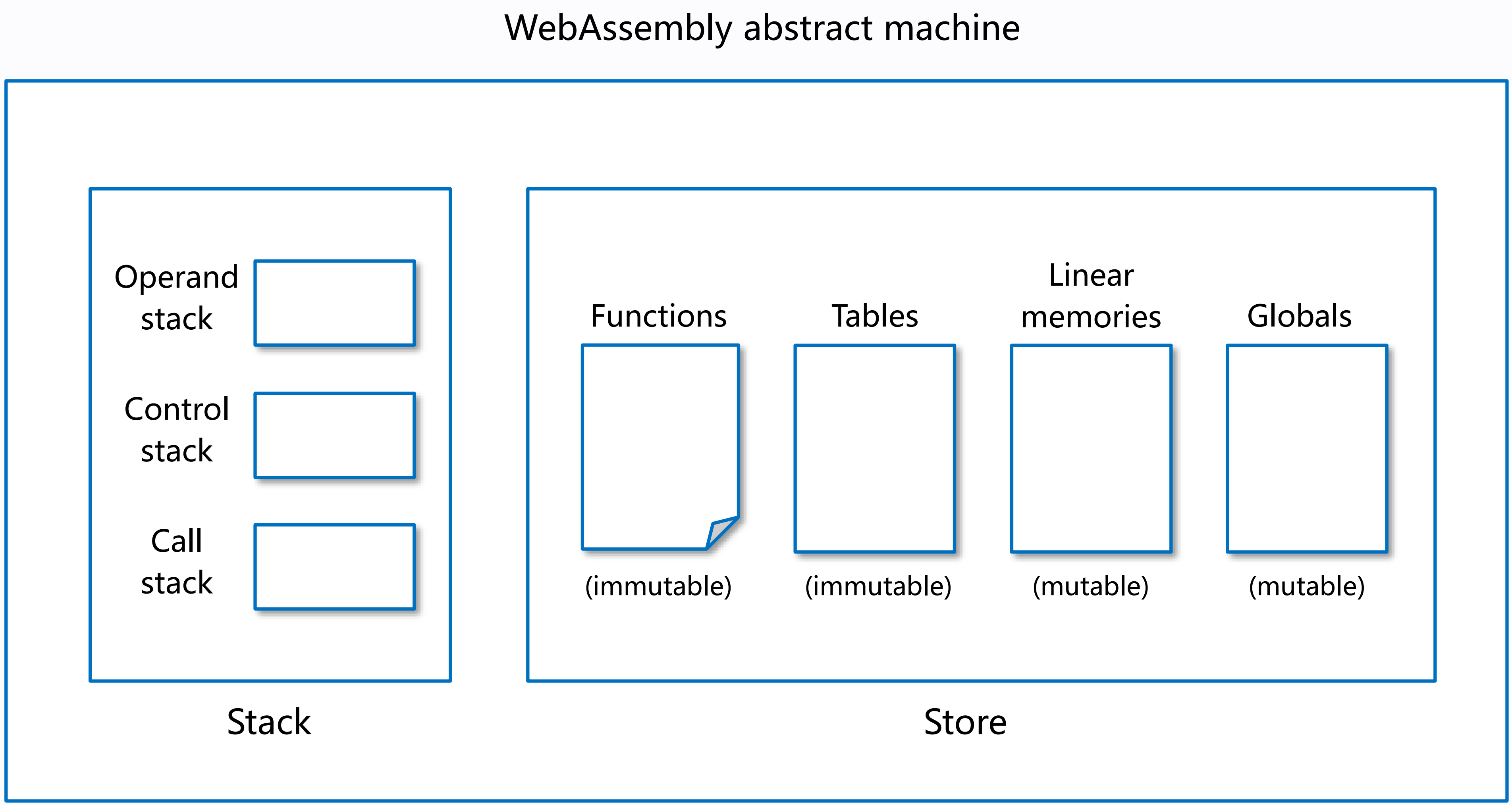
Under some conditions, certain instructions may produce a trap, which immediately aborts execution.
Traps cannot be handled by WebAssembly code, but are reported to the outside environment, where they typically can be caught.

References : [1] Ch.1.2

2. WebAssembly abstract machine

Abstract machine

WebAssembly abstract machine



[spec, 4.1.2 (stack)] tegaki No.12

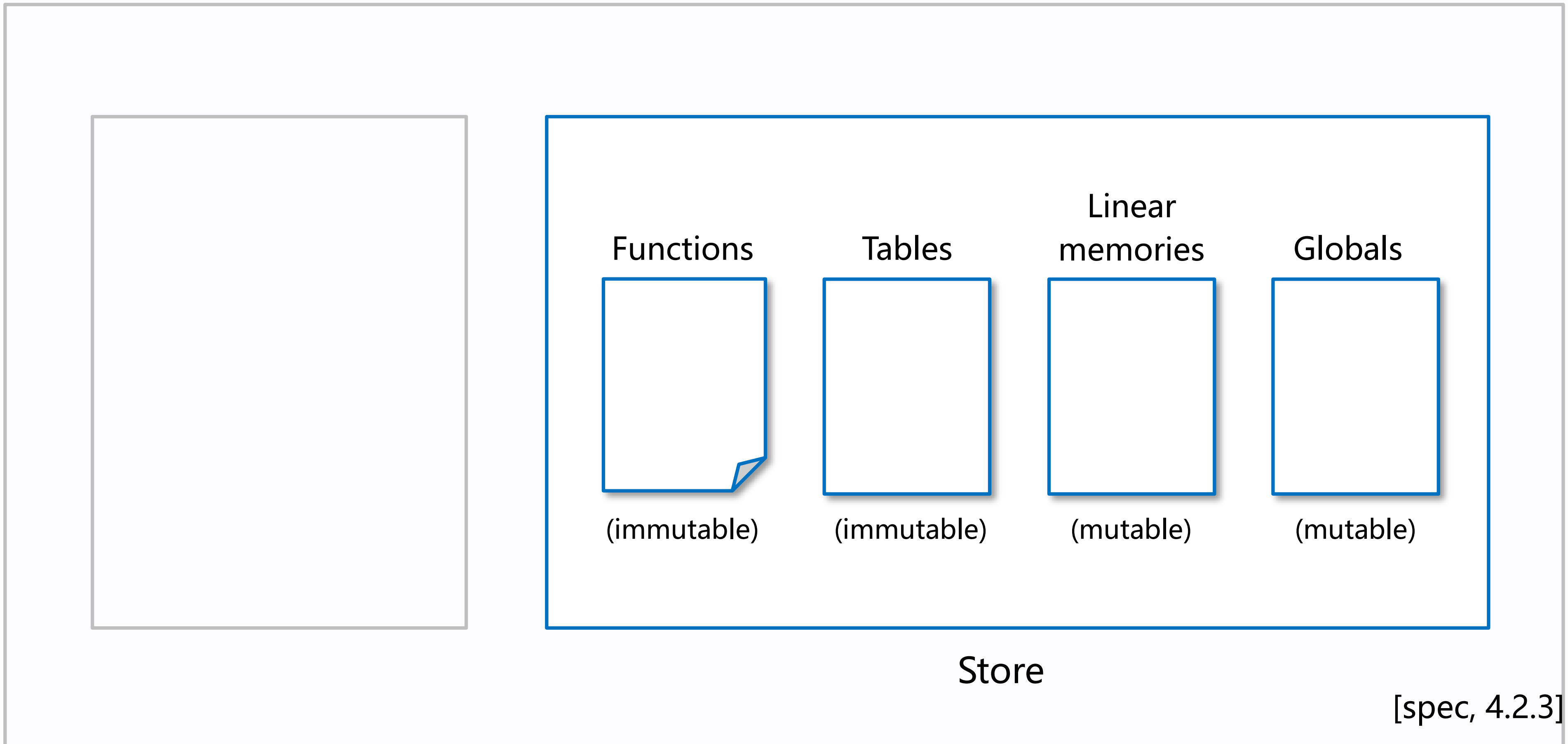
[spec, 4.2.3 Store]

[spec, 4.2]

Store, stack, and other runtime structure forming the WebAssembly abstract machine.

References : [1] Ch.1.2, Ch1.4

Store



The store represents all global state that can be manipulated byWebAssembly programs.
It consists of the runtime representation of all instances of functions, tables, memories, and globals that have been allocated during the life time of the abstract machine.

References : [1] Ch.1.2, Ch1.4

Stack

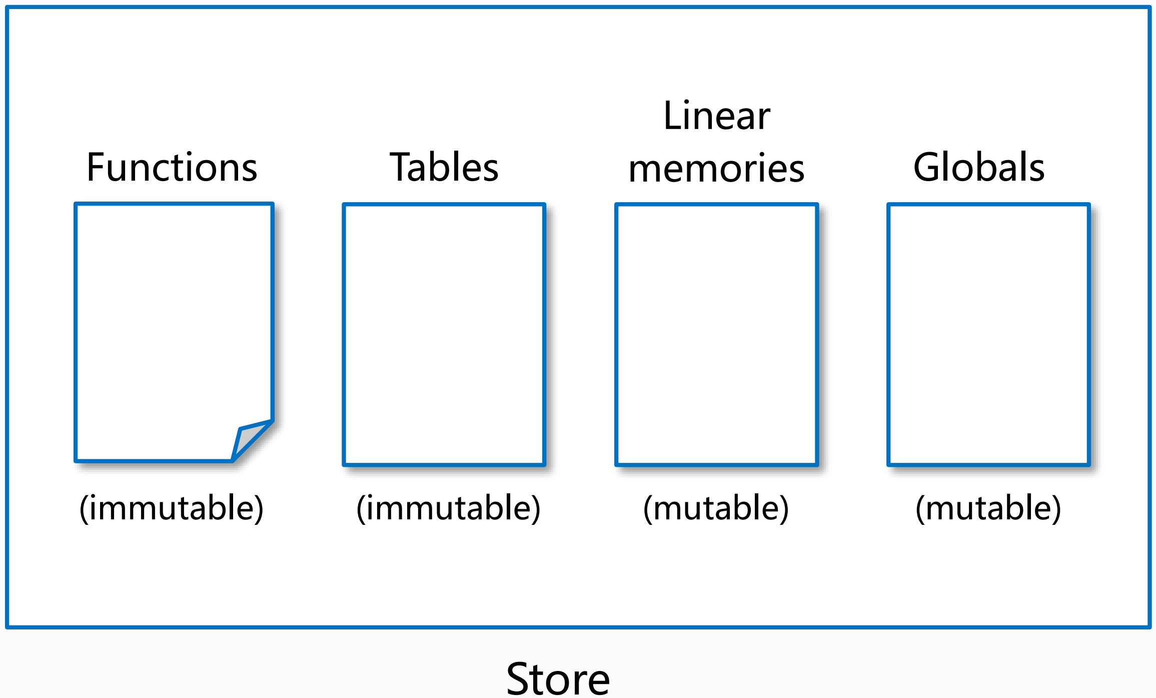
[spec, 4.2.12]

Besides the store, most instructions interact with an implicit stack. The stack contains three kinds of entries:

- Values: the operands of instructions.
- Labels: active structured control instructions that can be targeted by branches.
- Activations: the call frames of active function calls.

References : [1] Ch.1.2, Ch1.4

Index spaces



[spec, 2.5.1]

@@@ see tegaki, No.10

References : [1] Ch.1.2, Ch1.4

Linear memory

[spec, 4.2.8]

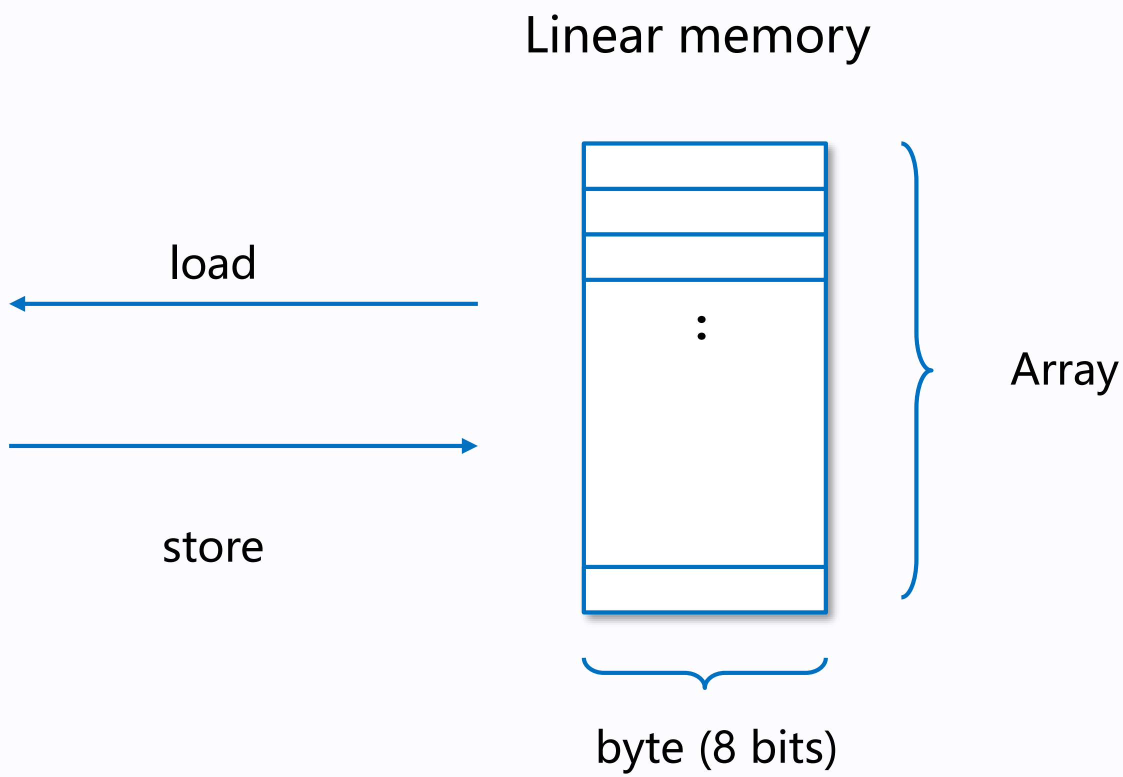
The len
which i

65536 -
memor

this pag

The byt
data se

provide



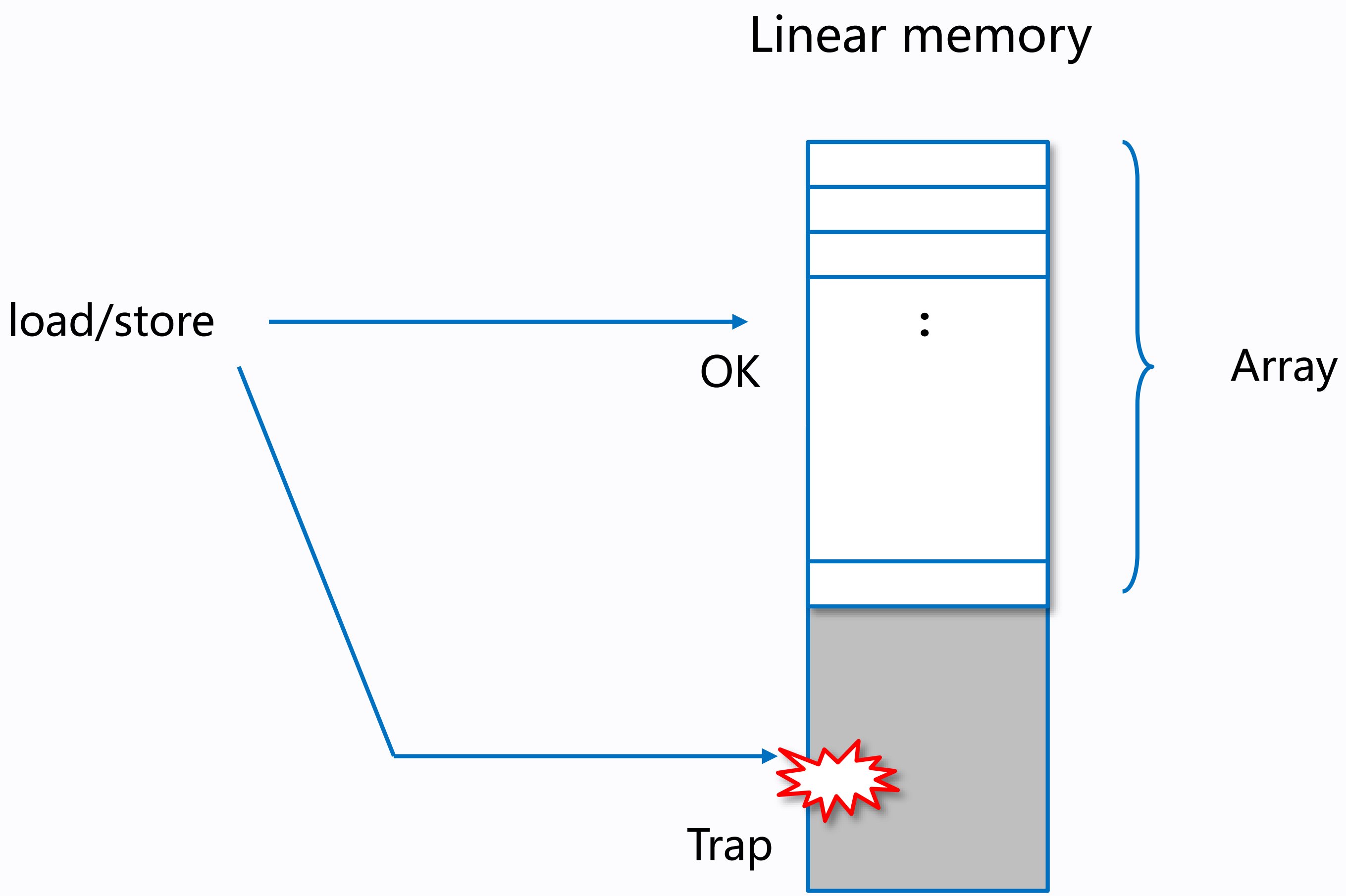
[spec, 1.2.1]

A linear memory is a contiguous, mutable array of raw bytes.

A program can load and store values from/to a linear memory at any byte address (including unaligned).
Integer loads and stores can specify a storage size which is smaller than the size of the respective value type.

References : [1] Ch.1.2

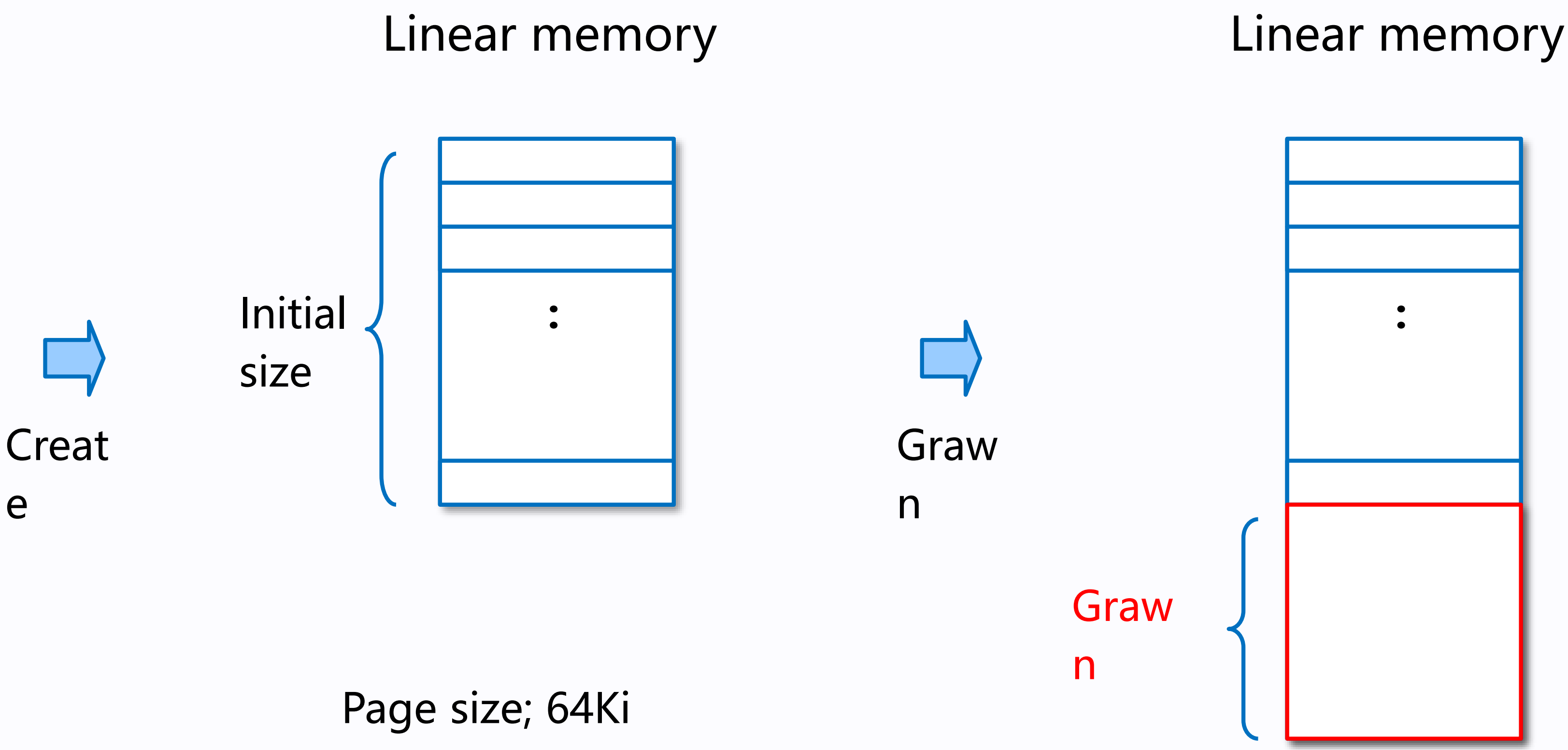
Linear memory



[spec, 1.2.1]

A trap occurs if an access is not within the bounds of the current memory size.

Linear memory

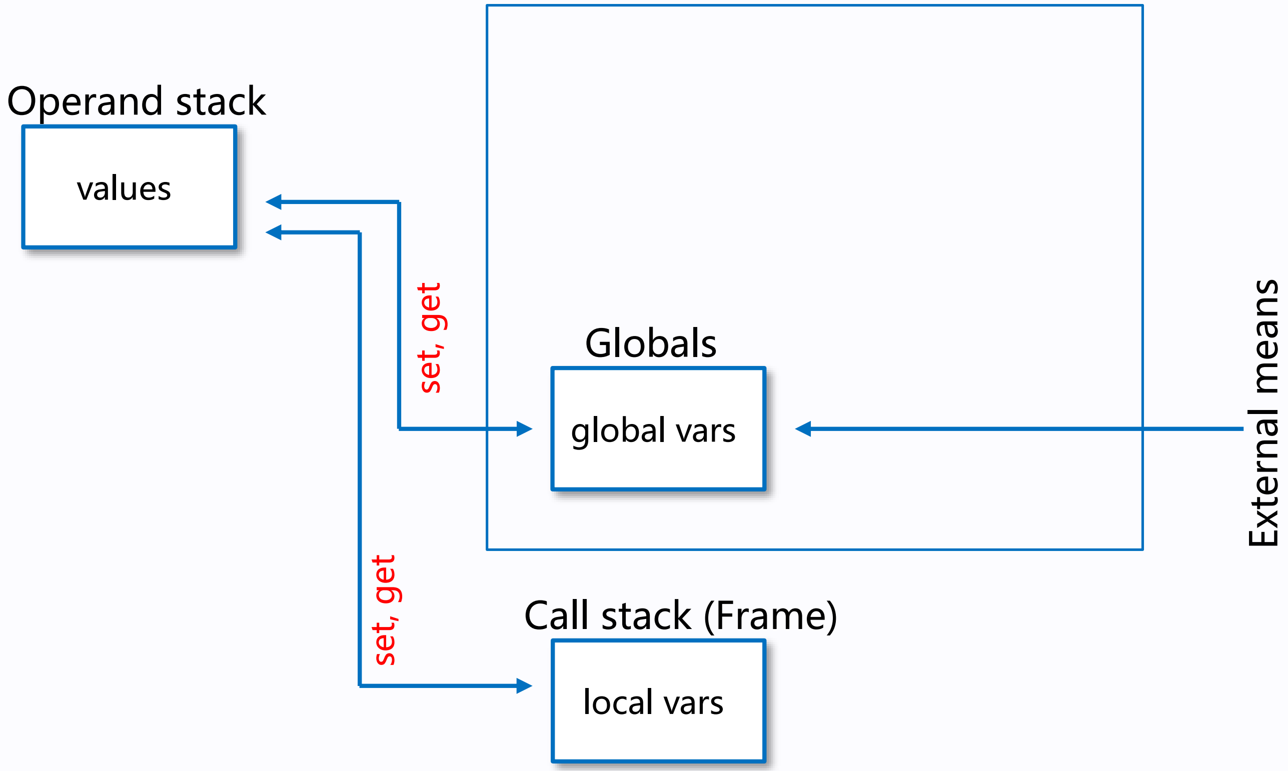


[spec, 1.2.1]

A memory is created with an initial size but can be grown dynamically.

Global and local variables

Variables



[spec, 2.4.3]

[spec, 2.5.3] local

[spec, 2.5.6] global

References : [1] Ch.1.2, Ch1.4

Global

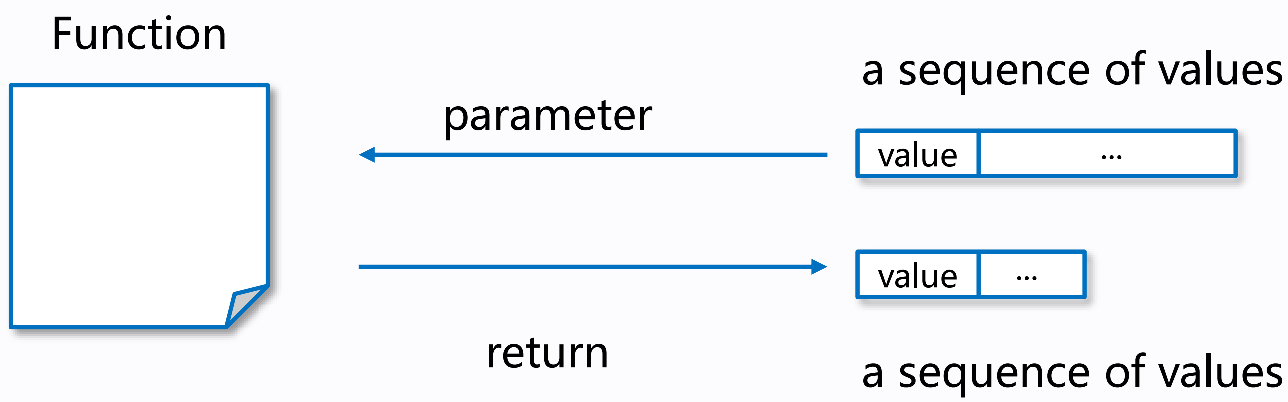
[spec, 4.2.9]

The value of mutable globals can be mutated through variable instructions or by external means provided by the embedder.

References : [1] Ch.1.2

Function and table

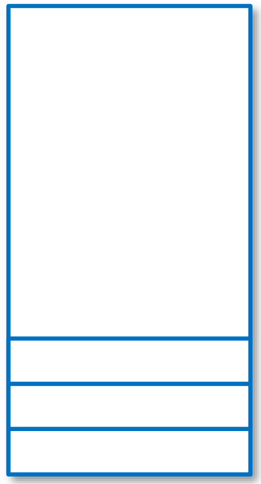
Functions



[spec, 1.2.1]

Code is organized into separate functions. Each function takes a sequence of values as parameters and
returns a sequence of values as results.

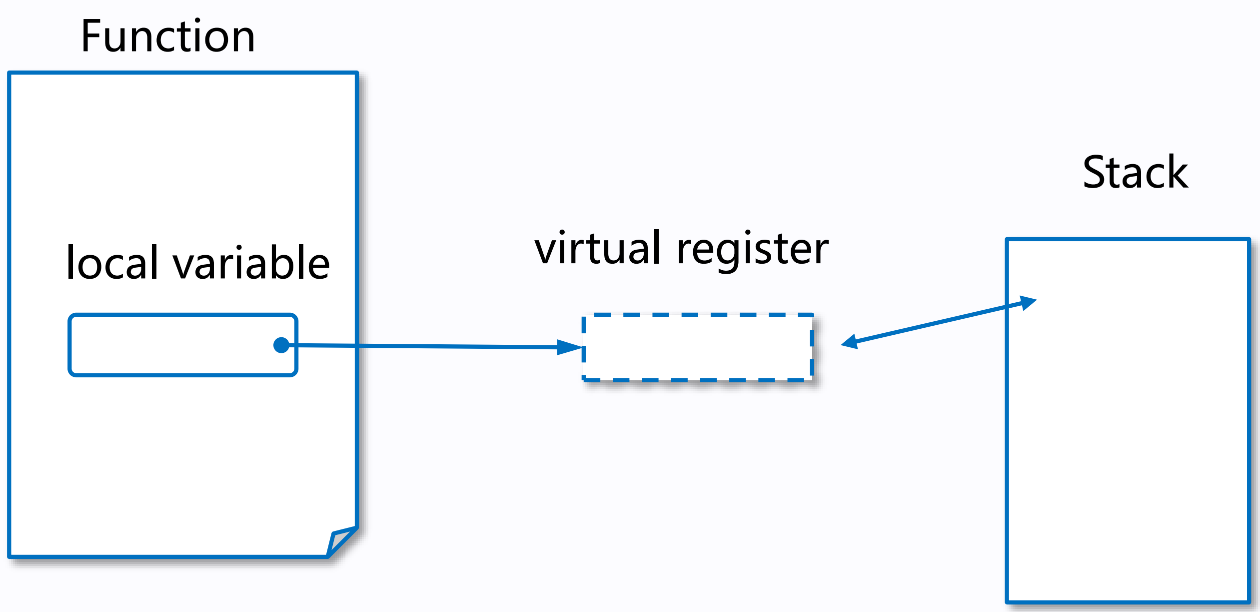
Functions



[spec, 1.2.1]

Functions can call each other, including recursively, resulting in an implicit call stack that cannot be accessed directly.

Functions

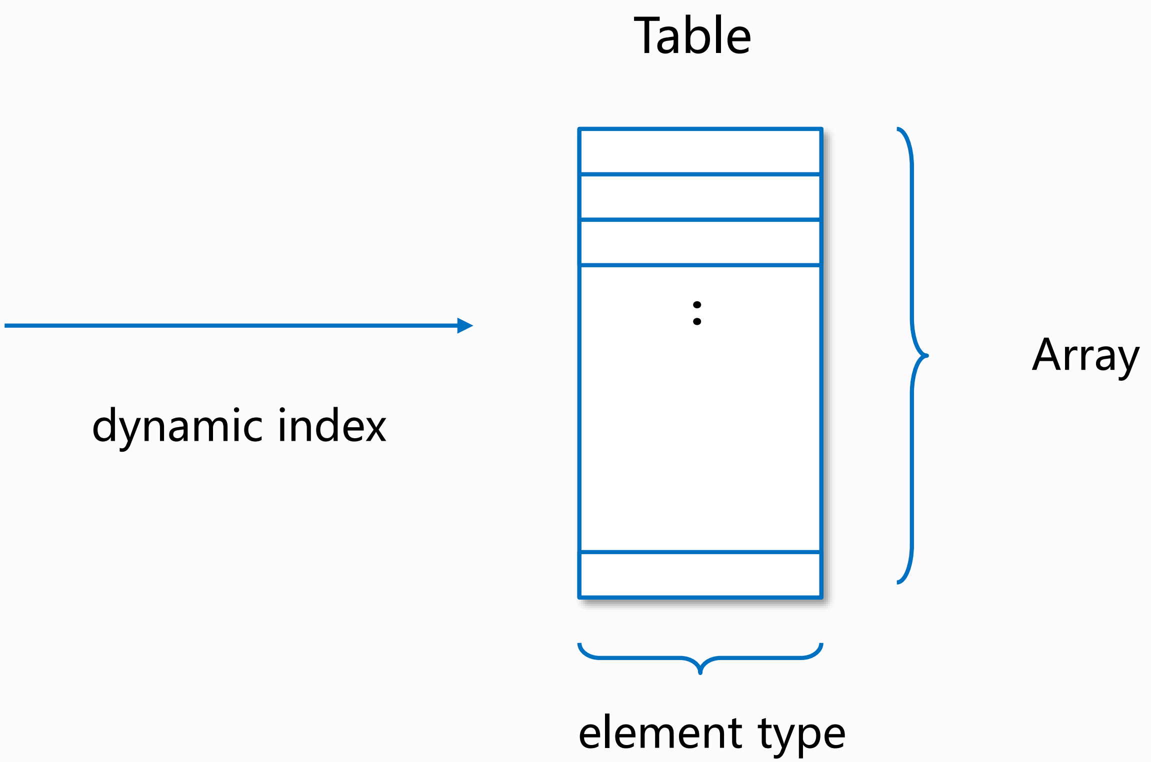


[spec, 1.2.1]

Functions may also declare mutable local variables that are usable as virtual registers.

References : [1] Ch.1.2

Tables



[spec, 1.2.1]

A table is an array of opaque values of a particular element type.

This allows emulating function pointers by way of table indices.

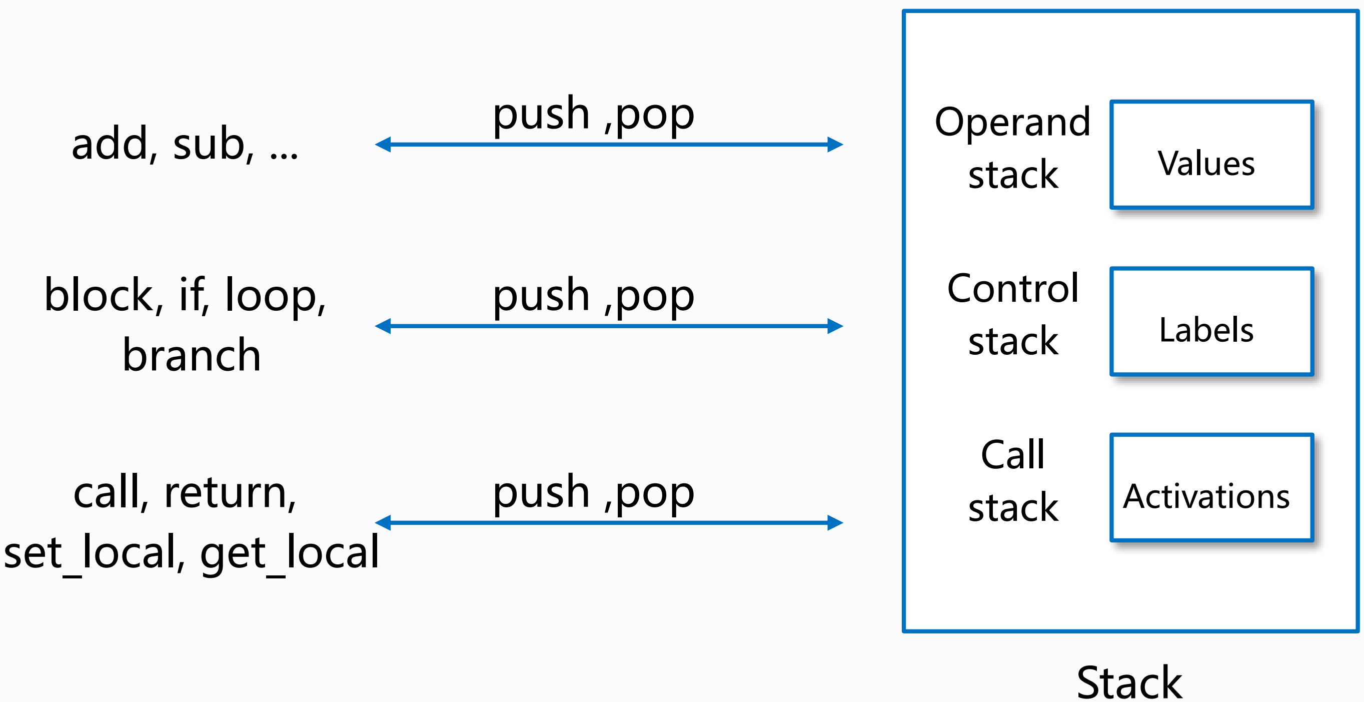
References : [1] Ch.1.2

2. WebAssembly abstract machine

Stack

Stack

[spec, 4.2.12]



structured stack
[rational, why]

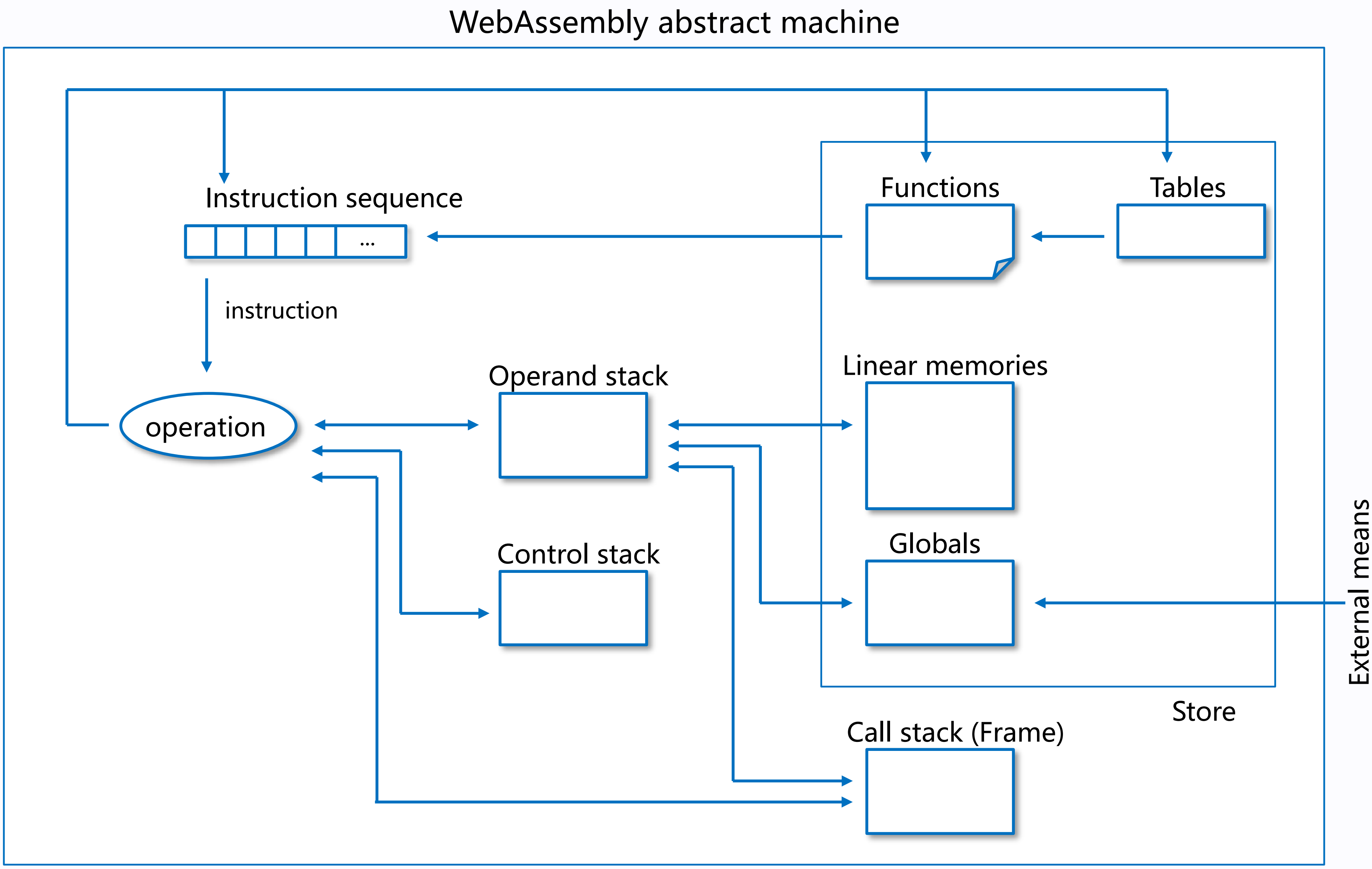
[spec, 4.2.12]

Note: It is possible to model the WebAssembly semantics using separate stacks for operands, control constructs, and calls. However, because the stacks are interdependent, additional book keeping about associated stack heights would be required. For the purpose of this specification, an interleaved representation is simpler.

References : [1] Ch.1.2, Ch1.4

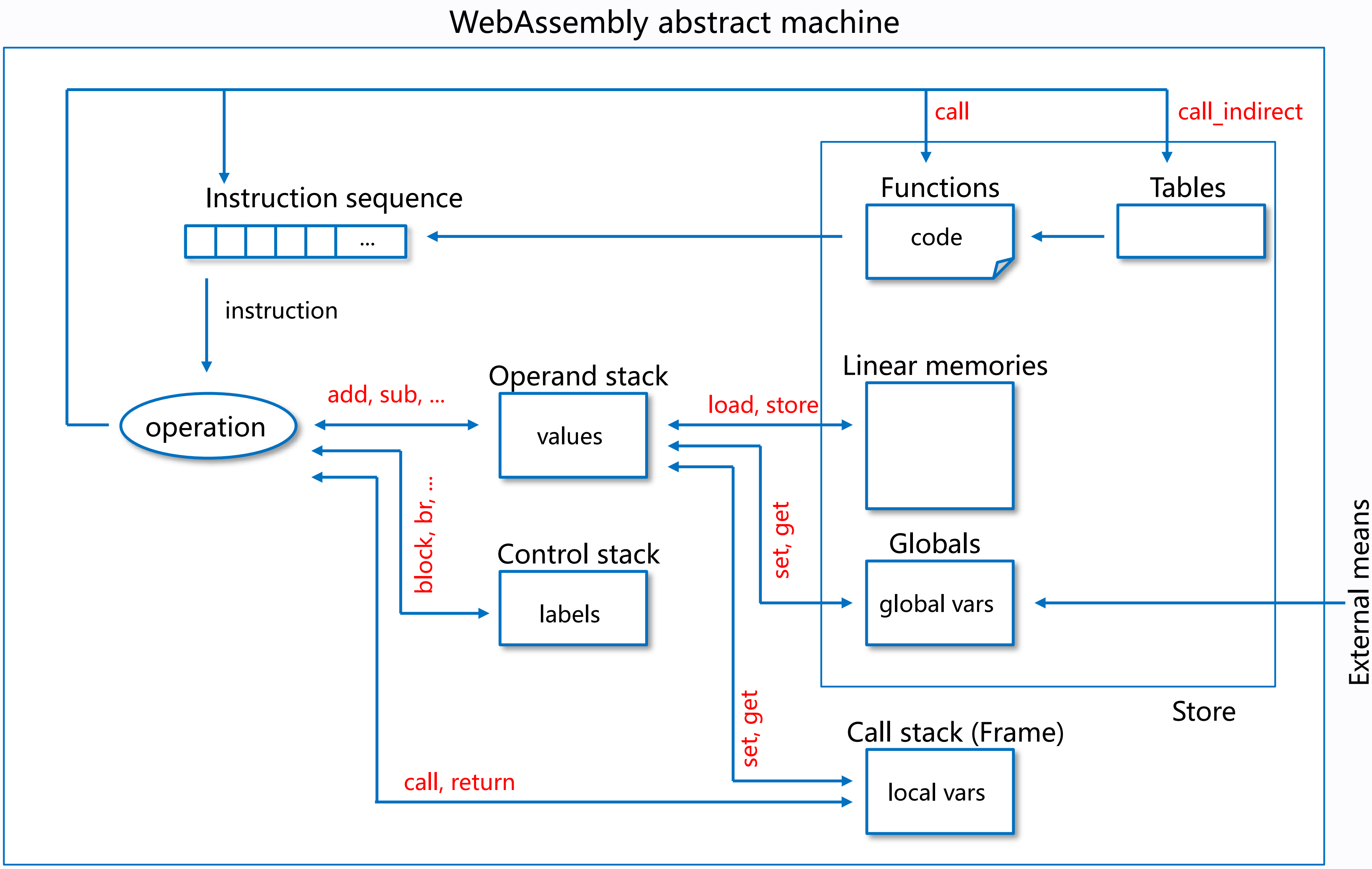
Computational model

Computational model



References : [1] Ch.1.2, Ch1.4

Computational model



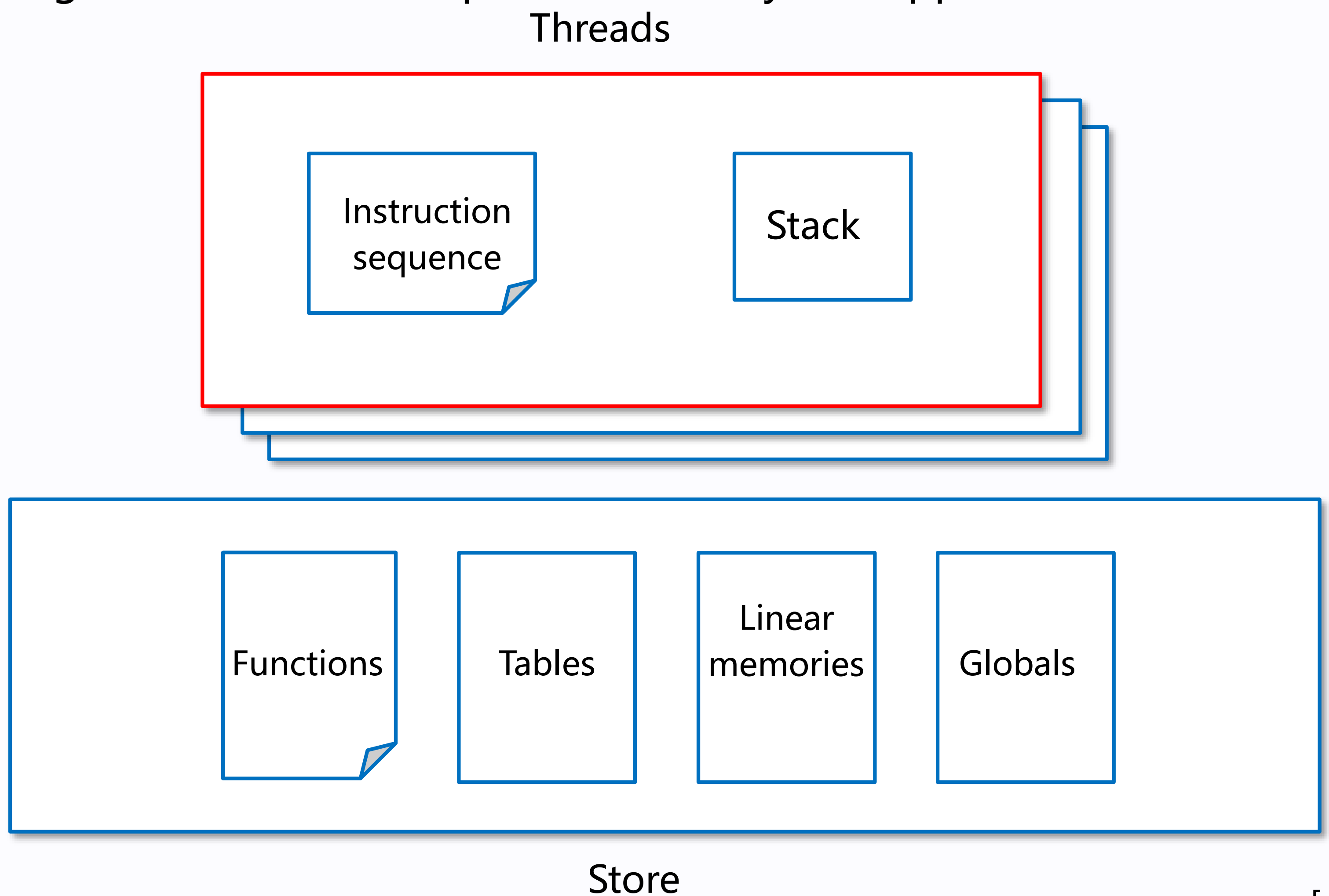
References : [1] Ch.1.2, Ch1.4

2. WebAssembly abstract machine

Thread

Thread

Note: The current version of WebAssembly is single-threaded, but configurations with multiple threads may be supported in the future.



[spec, 4.2.13]

A thread is a computation over instructions that operates relative to a current frame referring to the home module instance that the computation runs in.

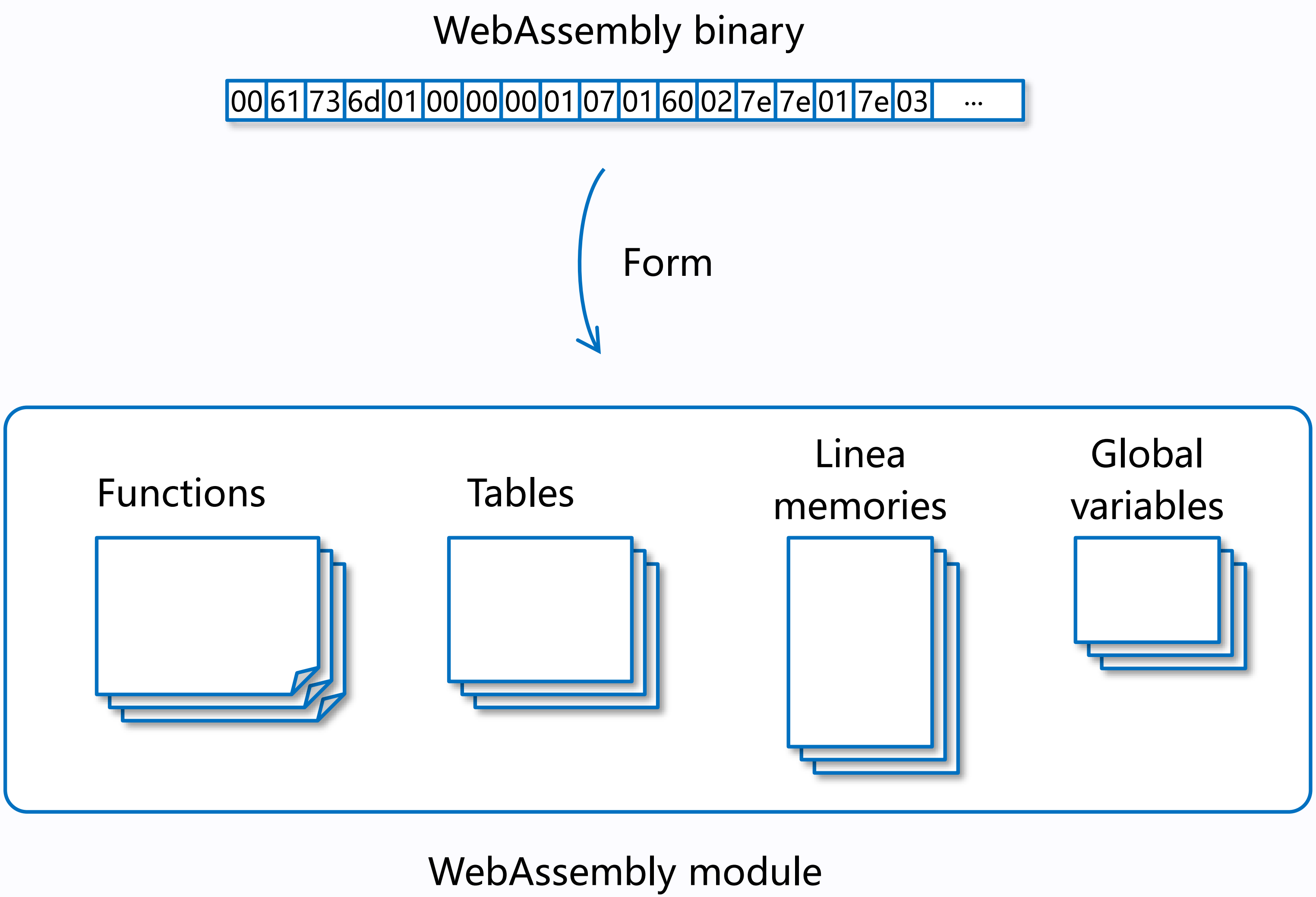
References : [1] Ch.1.2, Ch1.4

3. WebAssembly module

Module

Modules

WebAssembly modules are distributed in a binary format. [spec, 1.2.2]

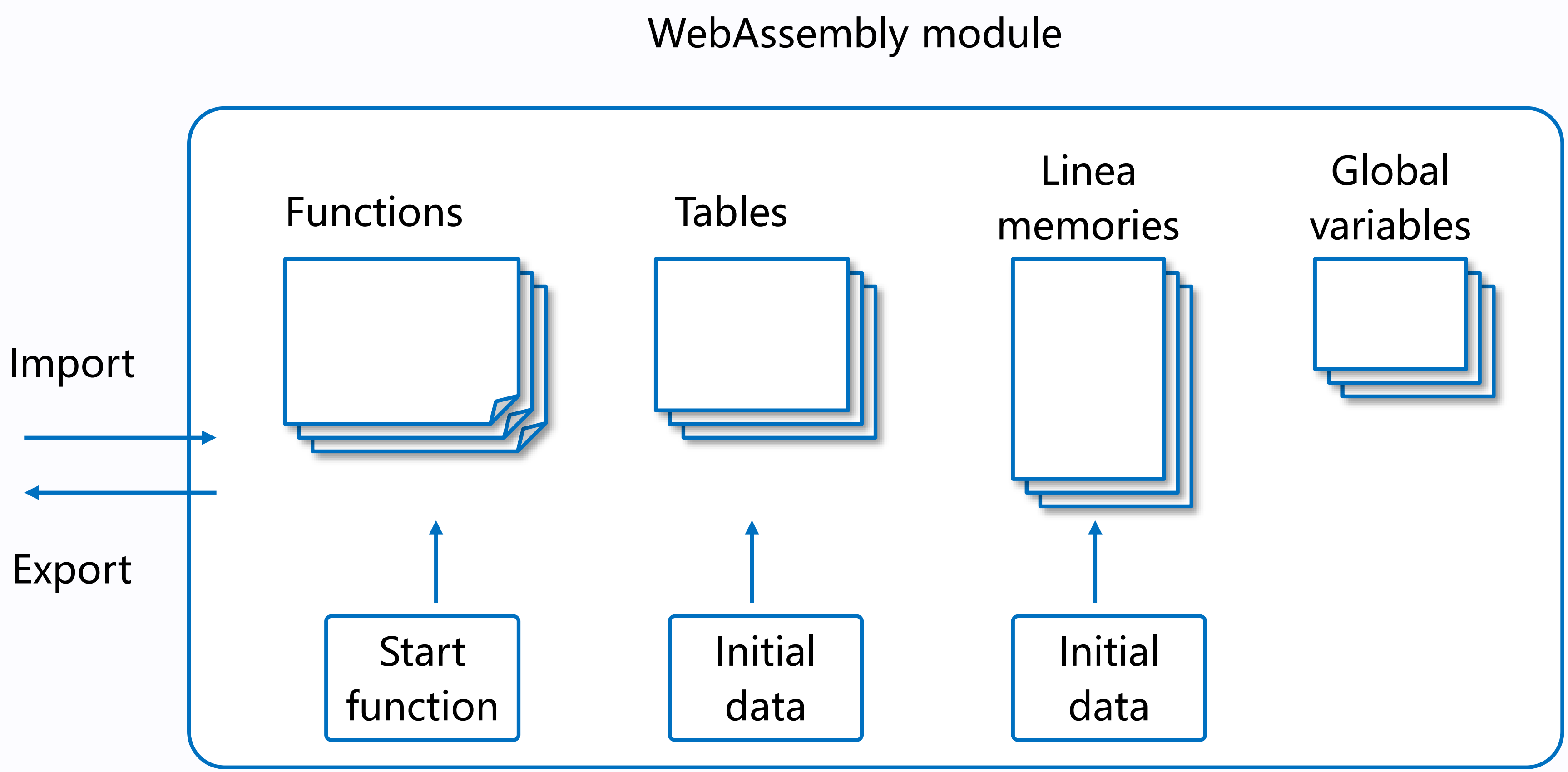


[spec, 1.2.1]

A WebAssembly binary takes the form of a module that contains definitions for functions, tables, and linear memories, as well as mutable or immutable global variables.

References : [1] Ch.1.2

Modules



References : [1] Ch.1.2

Modules

[spec, 2.5]

@@@ see tegaki, No.10

WebAssembly programs are organized into modules, which are the unit of deployment, loading, and compilation.
A module collects definitions for types, functions, tables, memories, and globals.
In addition, it can declare imports and exports and provide initialization logic in the form of data and element segments or a start function.

References : [1] Ch.1.2

Modules

[spec, 4.2.5]

@@@ see tegaki, 1026-2

A module instance is the runtime representation of a module.
It is created by instantiating a module, and collects runtime representations of all entities that are imported, defined, or exported by the module.

References : [1] Ch.1.2

Validation

[spec, 1.2.2]

A decoded module has to be valid.
Validation checks a number of well-formedness conditions to guarantee that the module is meaningful and safe.
In particular, it performs type checking of functions and the instruction sequences in their bodies, ensuring for example that the operand stack is used consistently.

[spec, 3.1]

Validation checks that a WebAssembly module is well-formed. Only valid modules can be instantiated.

References : [1] Ch.1.2

Types

[spec, 2.3]

Various entities in WebAssembly are classified by types.
Types are checked during validation, instantiation, and possibly execution.

References : [1] Ch.2.3

Instantiation and invocation

[spec, 1.2.2]

Instantiation. A module instance is the dynamic representation of a module, complete with its own state and execution stack. Instantiation executes the module body itself, given definitions for all its imports. It initializes globals, memories and tables and invokes the module’s start function if defined. It returns the instances of the module’s exports.

Invocation. Once instantiated, furtherWebAssembly computations can be initiated by invoking an exported function on a module instance. Given the required arguments, that executes the respective function and returns its results.

Instantiation and invocation are operations within the embedding environment.

References : [1] Ch.1.2

Instantiation and invocation

[spec, 1.2.2]

Instantiation

module -> instance -> (State (stoe) + Stack)

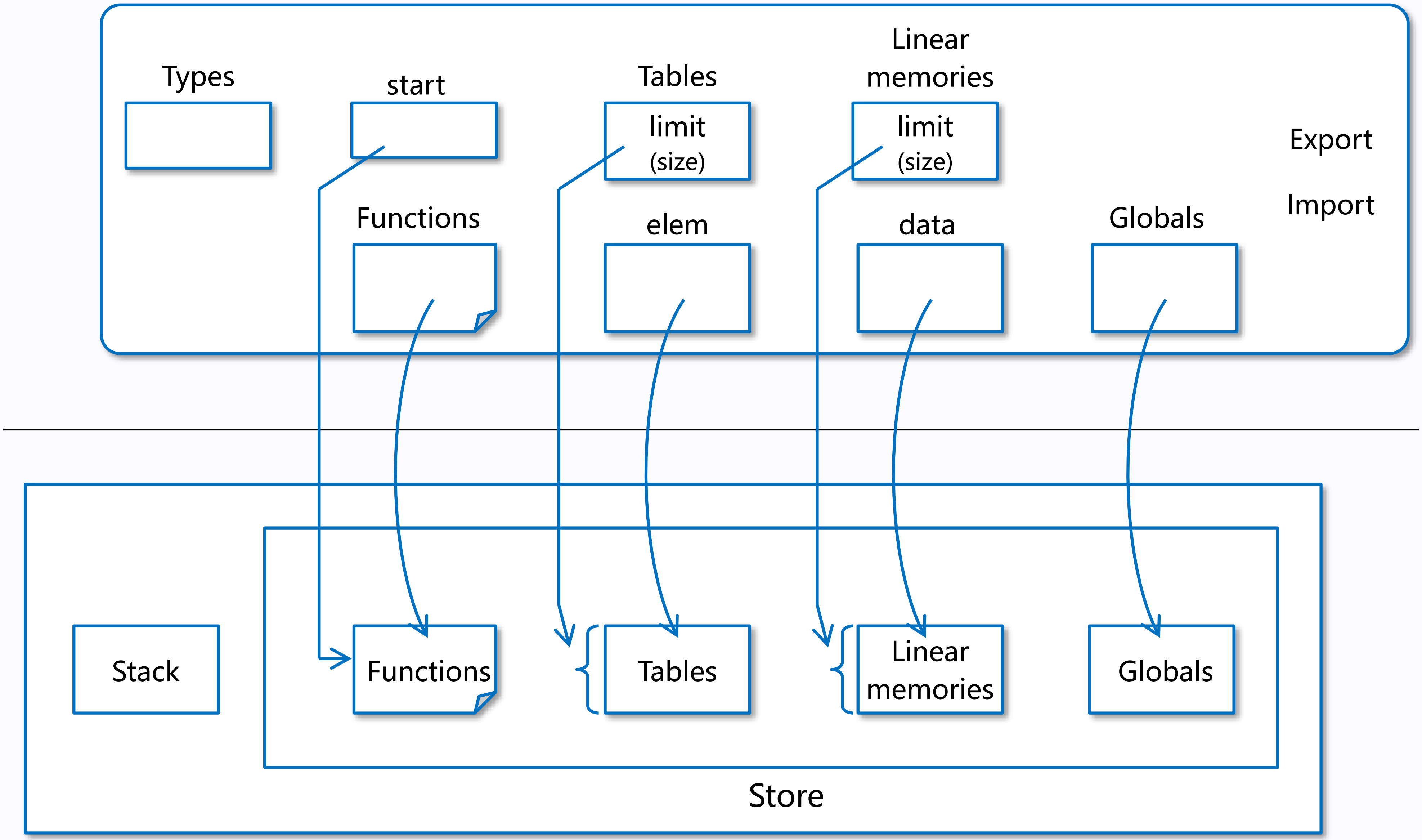
Invocation

?

References : [1] Ch.1.2

Instantiation from module binary to abstract machine

WebAssembly module binary format



WebAssembly abstract machine

References : [1] Ch.1.2, Ch1.4

Sections

Sections

[spec, 5.5]

The binary encoding of modules is organized into sections.

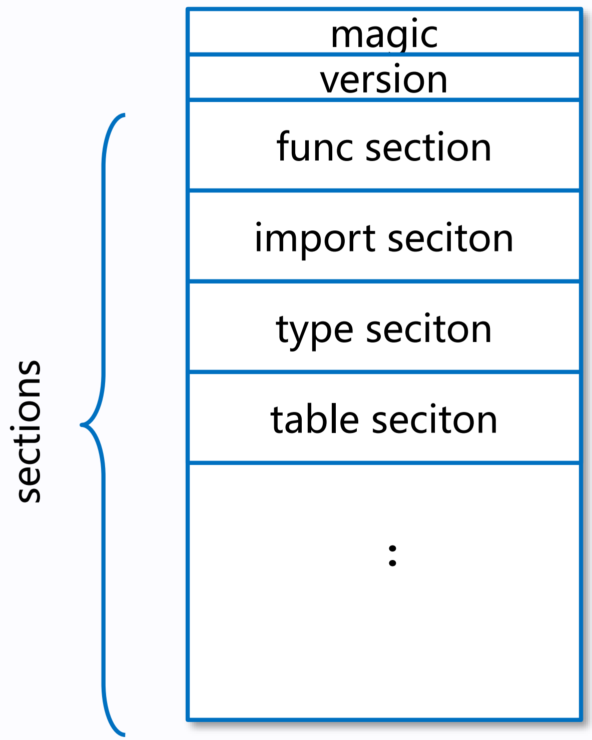
[spec, 5.5.2]

Each section consists of
- a one-byte section id,
- the u32 size of the contents, in bytes,
- the actual contents, whose structure is depended on the section id.

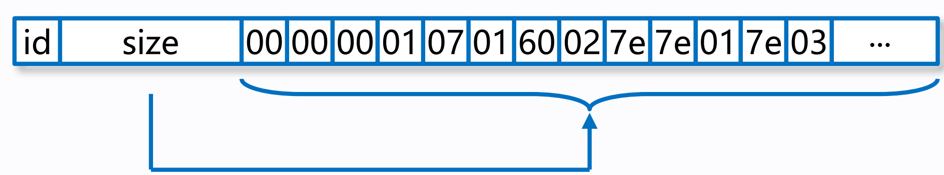
References : [1] Ch.1.2

Sections

Binary encoding of modules



[spec, 5.5.2]



References : [1] Ch.1.2

4. WebAssembly instructions

Simple instructions

Numeric instructions

[spec, 2.4.1]

@@@ see tegaki, No.8

References : [1] Ch.2.4

Parametric instructions

[spec, 2.4.2]

@@@ see tegaki, No.8

References : [1] Ch.2.4

Variable instructions

[spec, 2.4.3]

@@@ see tegaki, No.8

References : [1] Ch.2.4

Memory instructions

[spec, 2.4.4]

@@@ see tegaki, No.8

References : [1] Ch.2.4

[4. WebAssembly instructions]

Control instructions

Control instructions

[spec, 2.4.5]

@@@ see tegaki, No.8-9

[rational, why]

structured control flow allows simpler and more efficient verification

[rational, control flow]

Structured control flow provides simple and size-efficient binary encoding and compilation.

References : [1] Ch.2.4

Byte order

Endian

[spec, 4.3.1]

When a number is stored into memory, it is converted into a sequence of bytes in little endian23 byte order:

References : [1] Ch.2.4

Appendix A

Appendix A

Operational semantics

Operational semantics

[spec, 4.1-]

@@@ see tegaki, No.11-12, A1

[spec, 4.1]

Execution behavior is defined in terms of an abstract machine that models the program state. It includes a stack,
which records operand values and control constructs, and an abstract store containing global state.

LEB128

[spec, 5.2.2]

All integers are encoded using the LEB12828 variable-length integer encoding, in either unsigned or signed variant.

Unsigned integers are encoded in unsigned LEB128 format.

Signed integers are encoded in signed LEB12830 format.

References : [1] Ch.4.1

Appendix B

Appendix B

Implementation

References

References

[1] WebAssembly Specification Release 1.0 (Draft, last updated Oct 31, 2018)
https://webassembly.github.io/spec/core/

[2] Bringing the Web up to Speed with WebAssembly
https://github.com/WebAssembly/spec/blob/master/papers/pldi2017.pdf

[3] WebAssembly High-Level Goals
https://webassembly.org/docs/high-level-goals/

[4] Design Rationale
https://webassembly.org/docs/rationale/

[5] Modules
https://webassembly.org/docs/modules/

[6] WebAssembly Concepts
https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts

[7] Understanding WebAssembly text format
https://developer.mozilla.org/en-US/docs/WebAssembly/Understanding_the_text_format

References

[C1] Spec: WebAssembly Interpreter
https://github.com/WebAssembly/spec/tree/master/interpreter

[C2] Binaryen
https://github.com/WebAssembly/binaryen

[C3] WABT: The WebAssembly Binary Toolkit
https://github.com/WebAssembly/wabt

[C4] WebAssembly Virtual Machine
https://github.com/WAVM/WAVM

Drop figures

Architecture layer

Abstract machine
(Virtual machine)

WebAssembly Virtual ISA

* Instruction set
* Binary encoding
* Validation
* Execution semantics

Physical processor

x86 ISA, ARM ISA, ...

[spec, 1.1.2]

WebAssembly is a virtual instruction set architecture (virtual ISA).

References : [1] Ch.1.1

WebAssembly is a virtual instruction set architecture

Code

WebAssembly Bytecode

Instruction set architecture
(abstract machine)

WebAssembly Virtual ISA

Platform, Environment

Web browser, Other environment
(FireFox, Chrome, Safari, Edge, Node.js, WAVM,...)

Software

Hardware

Physical Processor
(x86, ARM, ...)

References : [1] Ch.1.1

WebAssembly Spec of Core and API

Core ISA

WebAssembly Core

ISA
Binary encoding
Validation
Exec

Environment

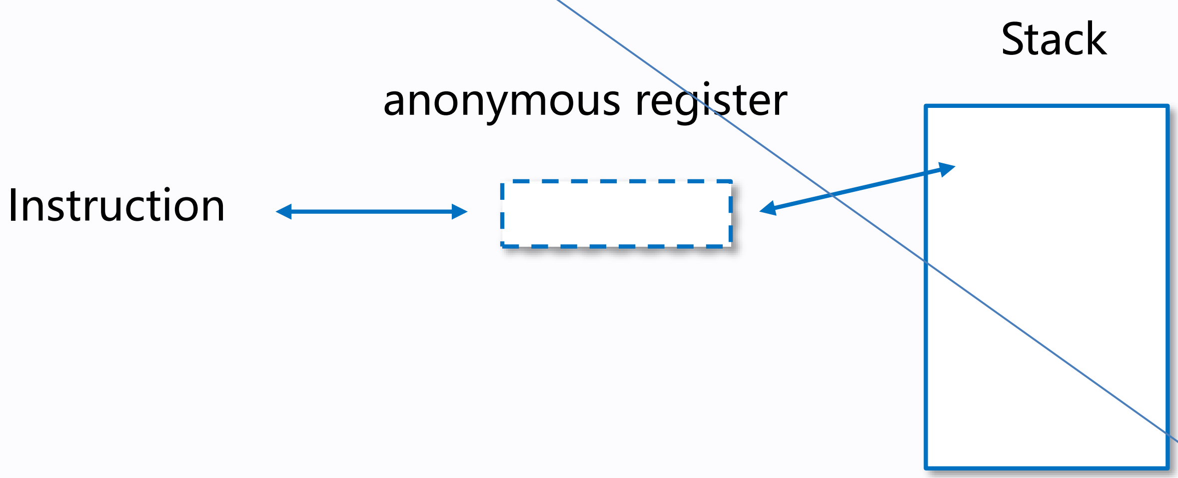
WebAssembly API

invoke
interact

References : [1] Ch.1.1

Stack machine

The computational model of WebAssembly is based on a stack machine.
Code ha meirei no sequence. inorder.
Anmokuno operand stack manipulate.
Stack, anonymous register

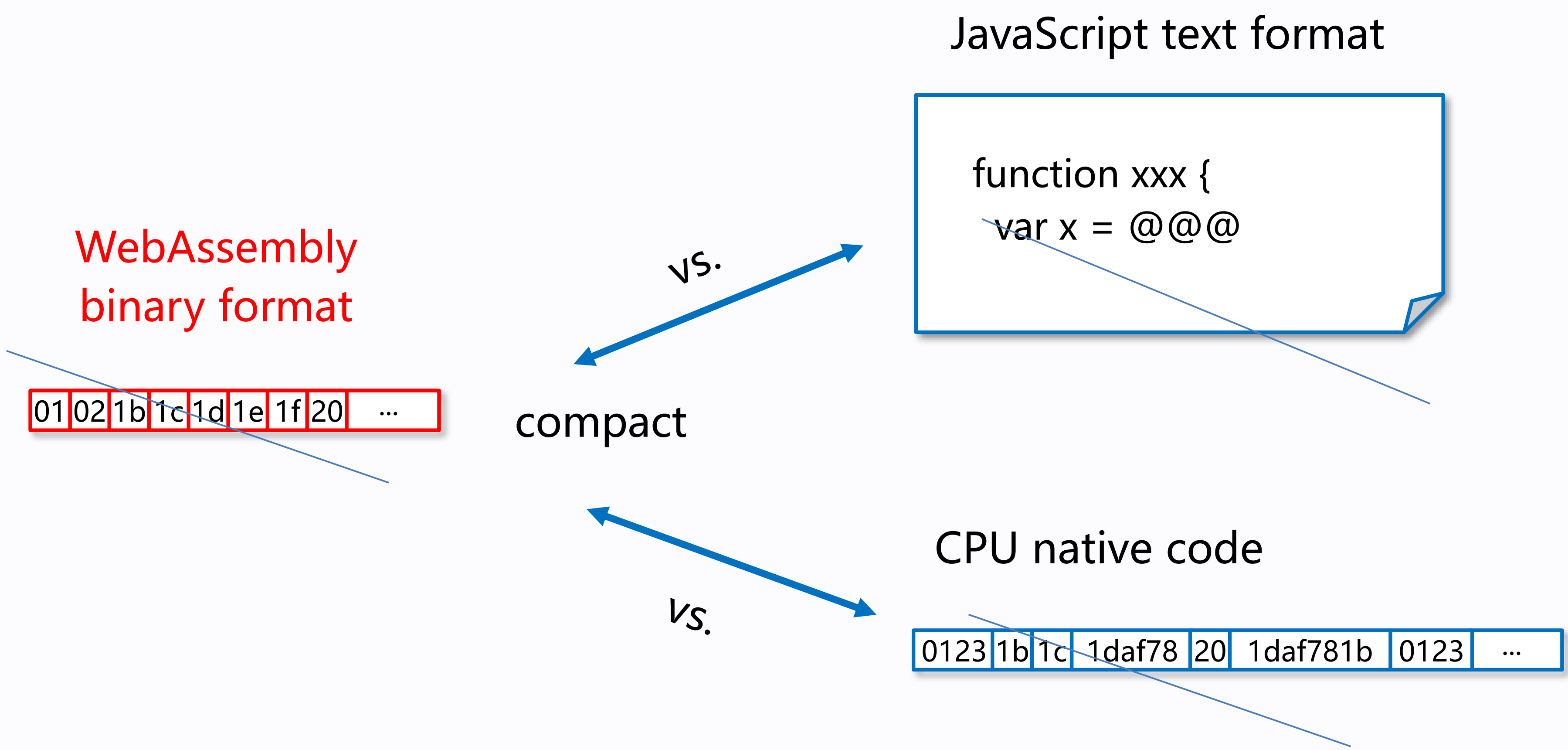


[spec, 1.2.1]

The computational model of WebAssembly is based on a stack machine.

References : [1] Ch.1.2

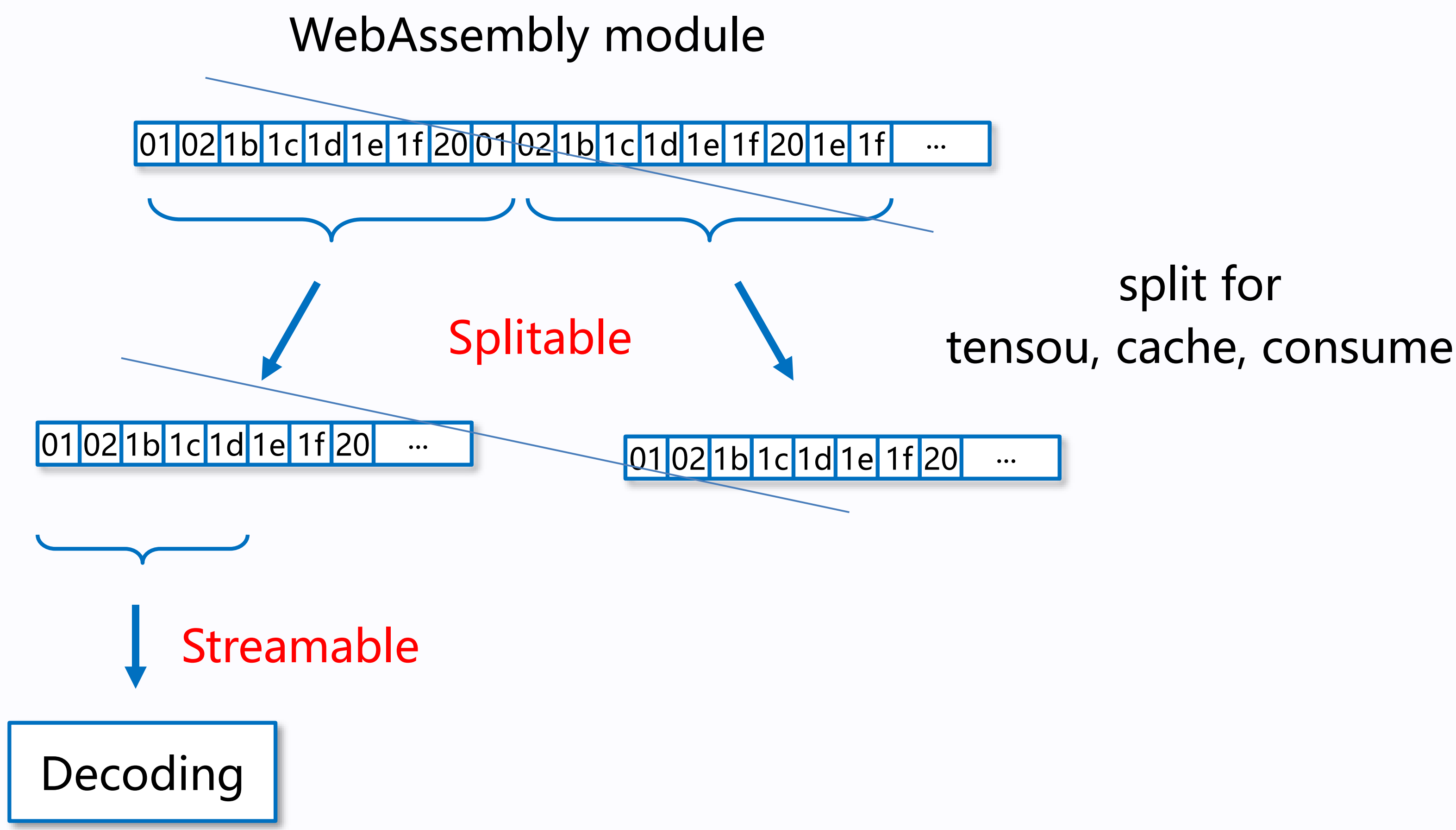
WebAssembly code is compact



@@@

References : [1] Ch.1.1

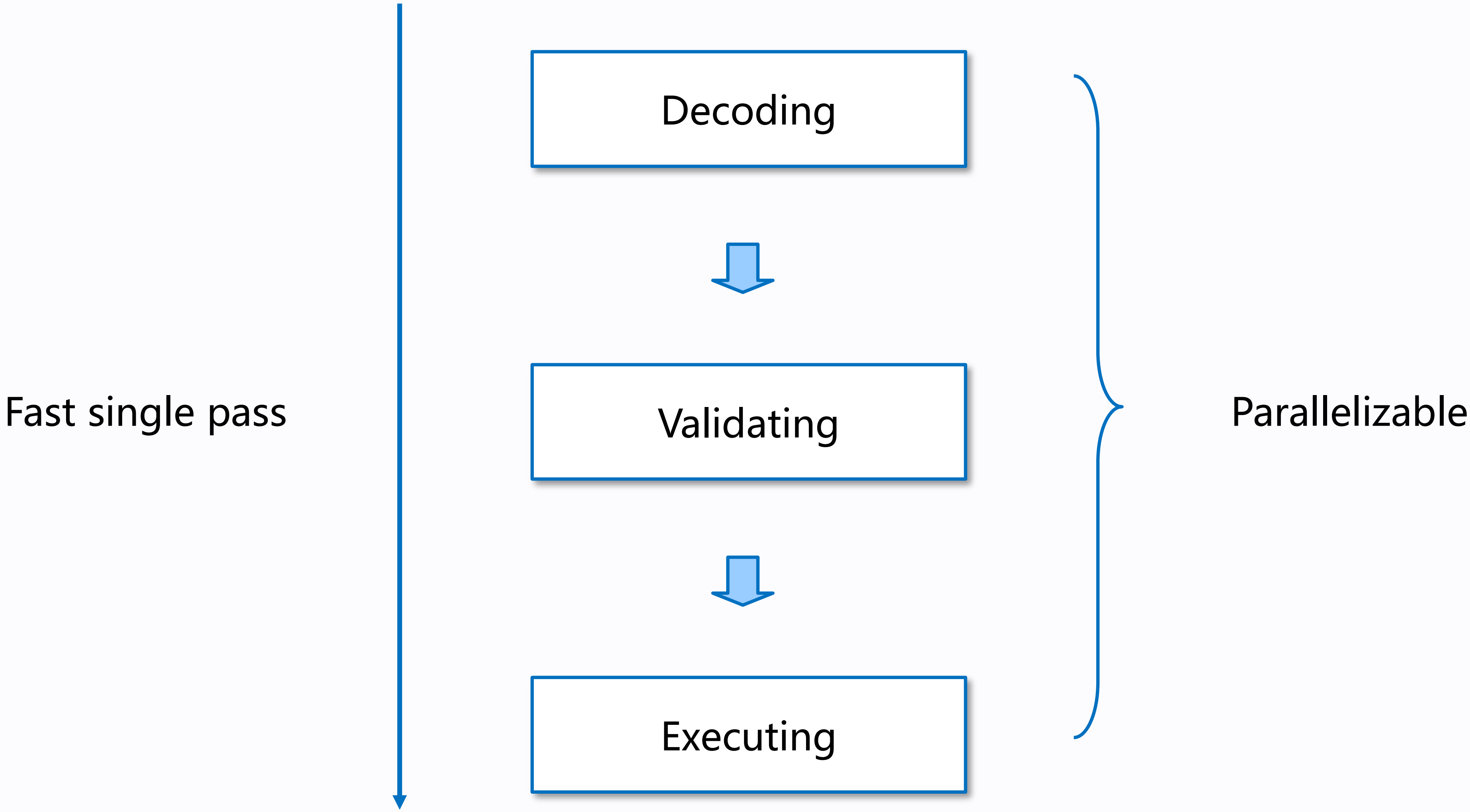
Split and streamable



@@@

References : [1] Ch.1.1

Efficient semantics



[spec, 1.2.2]

Conceptually, the semantics of WebAssembly is divided into three phases.

References : [1] Ch.1.1