Using and extending Wasm for security

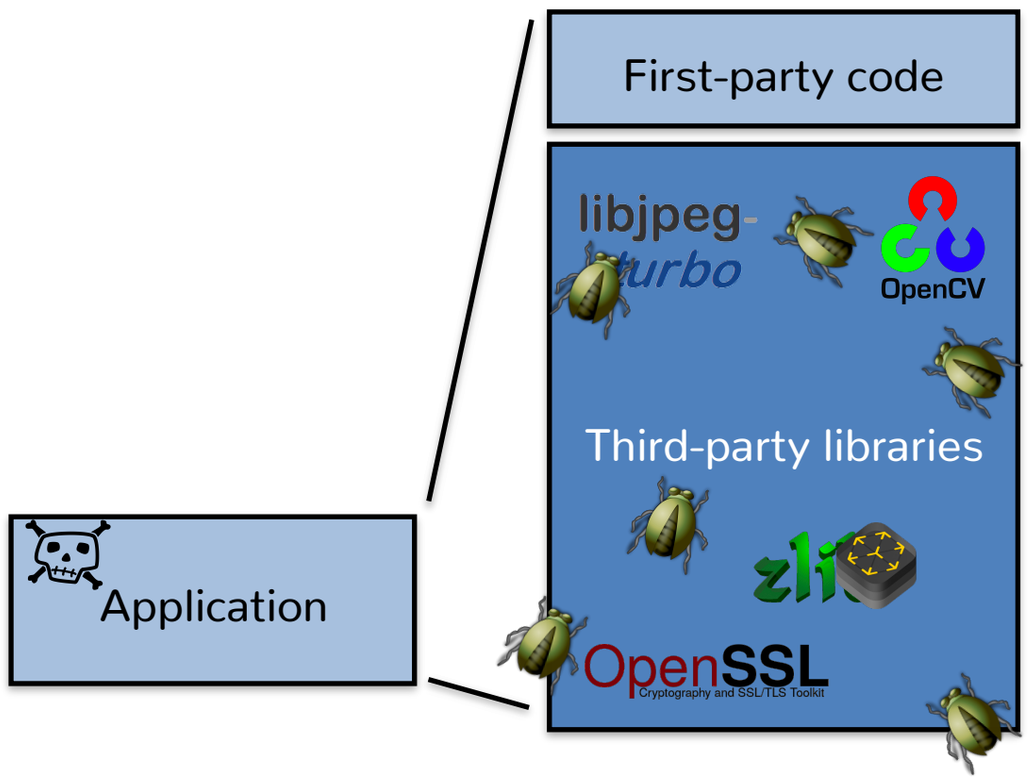
Deian Stefan

Using and extending Wasm for security

Deian Stefan

1. Sandboxing 3rd party libs

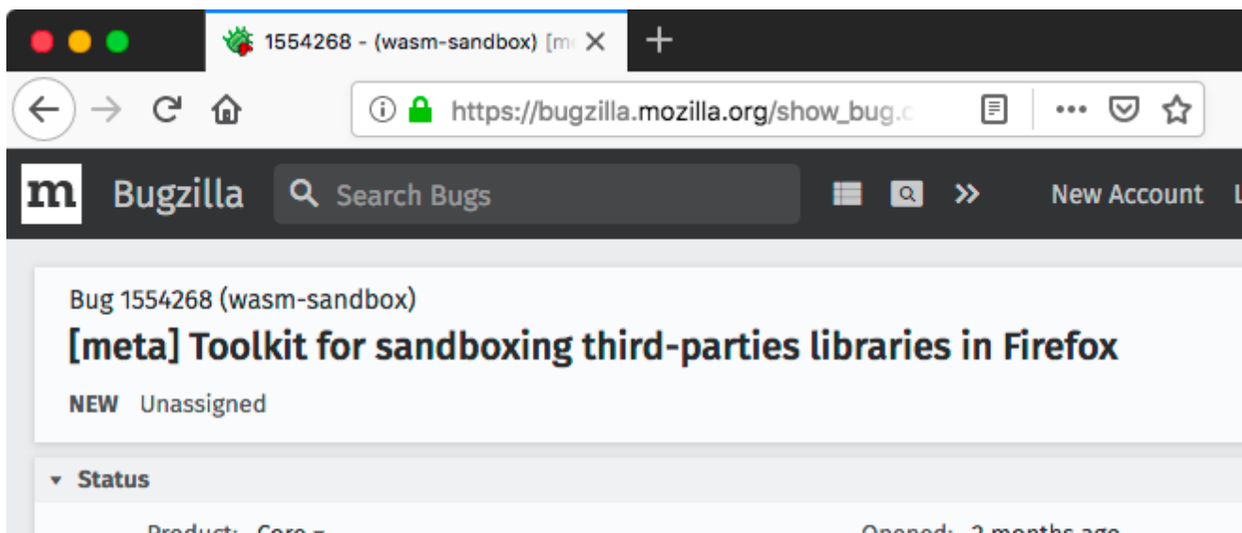
Shravan Narayan, Tal Garfinkel, Craig Disselkoen, Sorin Lerner, Hovav Shacham, Eric Rahm, Nathan Froyd



- Wasm is designed to be embedded in an app
  - I.e., designed to be sandboxed

Our approach

- Compile libraries to Wasm modules
  - Mediate Wasm→ runtime according to policy
- RLBox: toolkit for sandboxing libraries
  - Simplifies sandboxing and ensures application code cannot be abused by sandbox code
  - Available: rlbox.dev
  - Almost in production:



RLBox toolkit handles a lot...

wasm-sandbox.rlbox.dev

```
#include "rlbox_lucet_sandbox.hpp"
#include "rlbox.hpp"

int main()
{
    rlbox_sandbox<rlbox_lucet_sandbox> sandbox;
    sandbox.create_sandbox("libWasmFoo.so");
    // Invoke function bar with parameter 1
    sandbox.invoke_sandbox_function(bar, 1);
    sandbox.destroy_sandbox();
    return 0;
}
```

RLBox toolkit handles a lot...

Machines model differences, pointer swizzling, ABI issues for function calls and callbacks, how to port existing application, etc.

But sandboxing is not free

- Challenges we had/have to address:
  - AOT compilation (W)
  - x64 application, cross-platform (L,W)
  - Register and unregister callbacks (L)
  - 4gb heap alignment (L)
  - Library use case (L)
  - Re-entrant (L)
  - Threads

Going forward

- Wasm is THE way forward
  - Previous SFI toolkit (NaCl) is dead
- Library sandboxing as a use-case for Wasm
  - Compiler + sandbox: wasm-sandbox.rlbox.dev
- Need love: compiler, runtime, WASI

2. Verified Wasm compiler

- Jay Bosamiya, Bryan Parno, Benjamin Lin
- Implemented in F*: Wasm → x86/ARM
- Security property:
  - Compiled code cannot read/write outside region
- Going forward: how does this tie into runtimes? Can we implement a verified JIT vs AOT compiler?

Using and extending Wasm for security

Deian Stefan

Extending Wasm for security

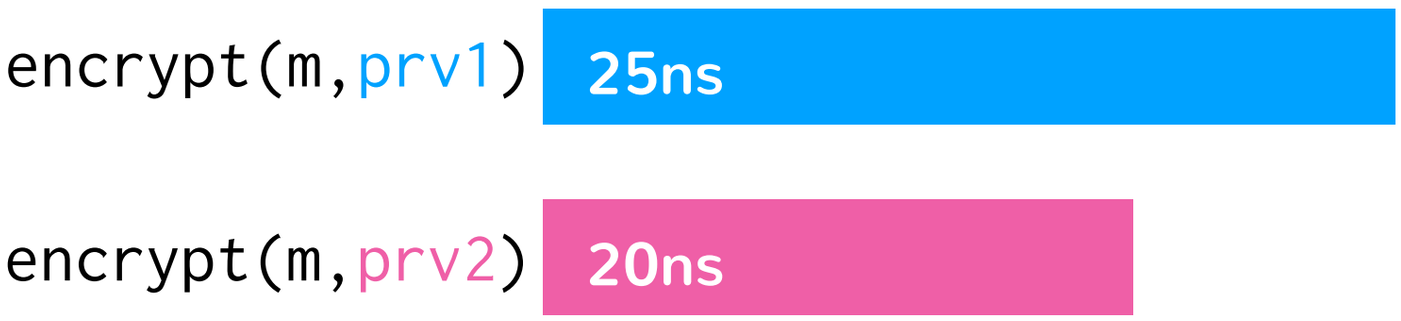
- CT-Wasm makes it possible for developers to write verifiably secure crypto algorithms
  - Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi
- MS-Wasm is a progressive approach to preventing memory safety violations in C/C++
  - Craig Disselkoen, John Renner, Conrad Watt, Tal Garfinkel, Amit Levy

Extending Wasm for security

- CT-Wasm makes it possible for developers to write verifiably secure crypto algorithms
  - Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi
- MS-Wasm is a progressive approach to preventing memory safety violations in C/C++
  - Craig Disselkoen, John Renner, Conrad Watt, Tal Garfinkel, Amit Levy

Writing crypto code is hard

- Need to ensure that code is not leaking secrets
  - Directly (e.g., by writing secrets to public memory)
  - Or indirectly, via timing channels



Wasm is close, but not enough

Doesn’t know what’s secret: can’t stop you from leaking it

CT-Wasm makes secrecy explicit

- Extend Wasm language with two new types
  - s32 and s64: secret 32- and 64-bit ints
  - All other types are public: i32, i64, f32, f64
- Extend compiler to turn leaks into type errors
  - Prevent direct leaks
  - Prevent implicit leaks (via control flow)
  - Prevent leaks via timing channels

CT-Wasm makes secrecy explicit

- Extend Wasm language with two new types
  - s32 and s64: secret 32- and 64-bit ints
  - All other types are public: i32, i64, f32, f64
- Extend compiler to turn leaks into type errors
  - Prevent direct leaks
  - Prevent implicit leaks (via control flow)
  - Prevent leaks via timing channels

Prevent explicit leaks

(via local variables)

```
(local $pub i32)
(local $sec s32)
```

```
(set_local $pub (get_local $sec))
```

Prevent explicit leaks

(via local variables)

```
(local $pub i32)
(local $sec s32)

(set_local $pub (get_local $sec))
```

```
Error: type mismatch in set_local,
       expected i32 but got s32
```

Prevent implicit leaks

(via control flow)

```
(if (get_local $sec)
  (then (set_local $pub ...))
  (else (set_local $pub ...)))
```

Prevent implicit leaks

(via control flow)

```
(if (get_local $sec)
  (then (set_local $pub ...))
  (else (set_local $pub ...)))
```

```
TypeError
‘if’ requires i32
     found    s32
```

Prevent leaks via timing

- Disallow variable-time instructions on secrets
  - E.g., we don't define `s32.div` or `s32.mod`
- Disallow secret-dependent memory accesses
  - E.g., `(s32.load (get_local $sec))`

Prevent leaks via timing

- Disallow variable-time instructions on secrets
  - E.g., we don't define `s32.div` or `s32.mod`
- Disallow secret-dependent memory accesses
  - E.g., `(s32.load (get_local $sec))`

```
TypeError
's32.load' requires i32
           found    s32
```

Simple extension ➠ secure crypto

Extending Wasm for security

- CT-Wasm makes it possible for developers to write verifiably secure crypto algorithms
  - Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi
- MS-Wasm is a progressive approach to preventing memory safety violations in C/C++
  - Craig Disselkoen, John Renner, Conrad Watt, Tal Garfinkel, Amit Levy

Extending Wasm for security

- CT-Wasm makes it possible for developers to write verifiably secure crypto algorithms
  - Conrad Watt, John Renner, Natalie Popescu, Sunjay Cauligi

➔ MS-Wasm is a progressive approach to preventing memory safety violations in C/C++

  - Craig Disselkoen, John Renner, Conrad Watt, Tal Garfinkel, Amit Levy

Wasm code isn't necessarily memory safe...

Manual memory management

- Makes it easy to compile C/C++ that runs fast!

- C/C++ code is plagued with memory safety bugs
  - Buffer overflows
  - Use after frees

70% of security bugs at Microsoft

Manual memory management

- Makes it easy to compile C/C++ that runs fast!

- C/C++ code is plagued with memory safety bugs
  - Buffer overflows
  - Use after frees

70% of security bugs at Microsoft

May be worse than native code...

- Native code can use hardware mechanisms that check certain kinds of memory safety violations

Kostya Serebryany
@kayseesee

Follow

Adopting the Arm Memory Tagging Extension in Android

Adopting the Arm Memory Tagging Extension in Android

Posted by Kostya Serebryany, Google Core Systems, and Sudhi Herle, Android Security & Privacy Team As part of our continuous commitment t...

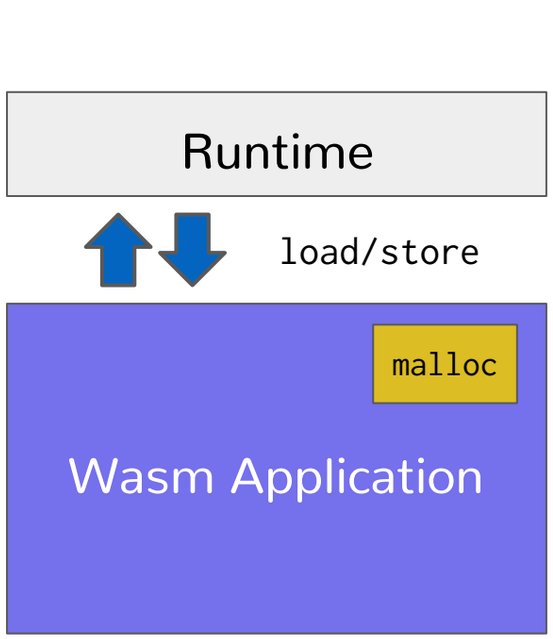
security.googleblog.com

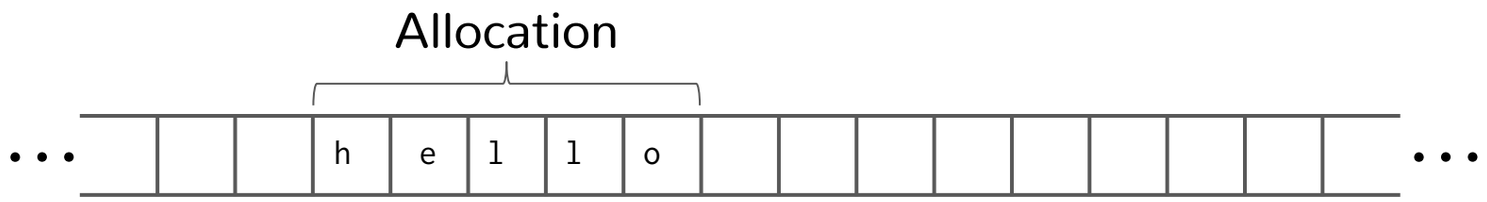
MS-Wasm: progressive safety

- Extend Wasm to capture the essence mem safety
  - Memory allocation is explicit
  - Typed handles associate pointers to allocations
- Leave enforcement up to the runtime
  - Depends on available HW and application demands

Challenge with Wasm today

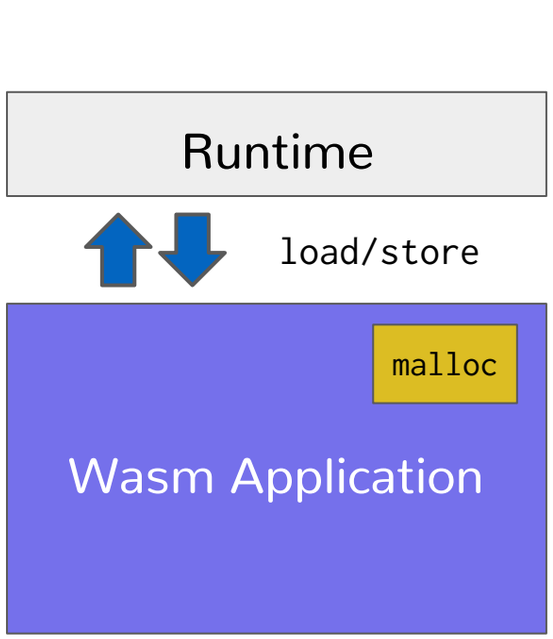
- Wasm runtime isn't aware of allocations
  - Can't enforce bounds checks!

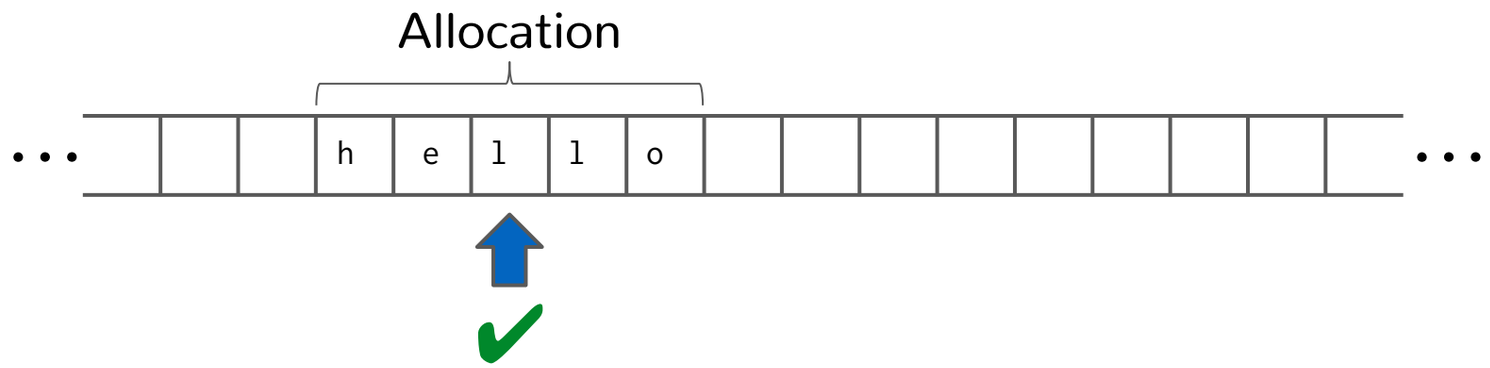




Challenge with Wasm today

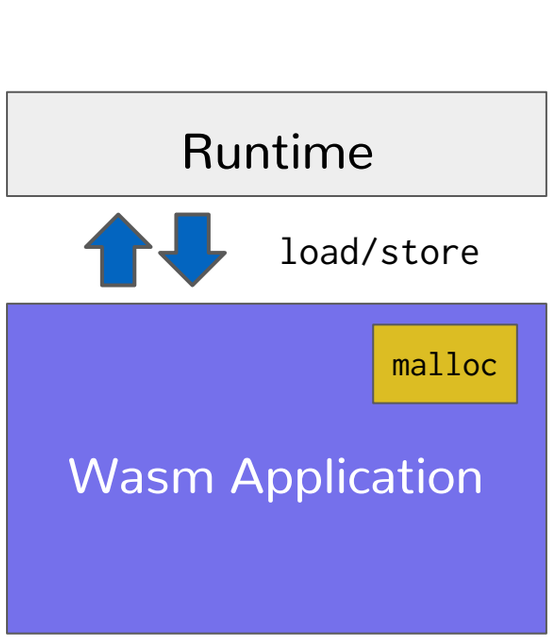
- Wasm runtime isn't aware of allocations
  - Can't enforce bounds checks!

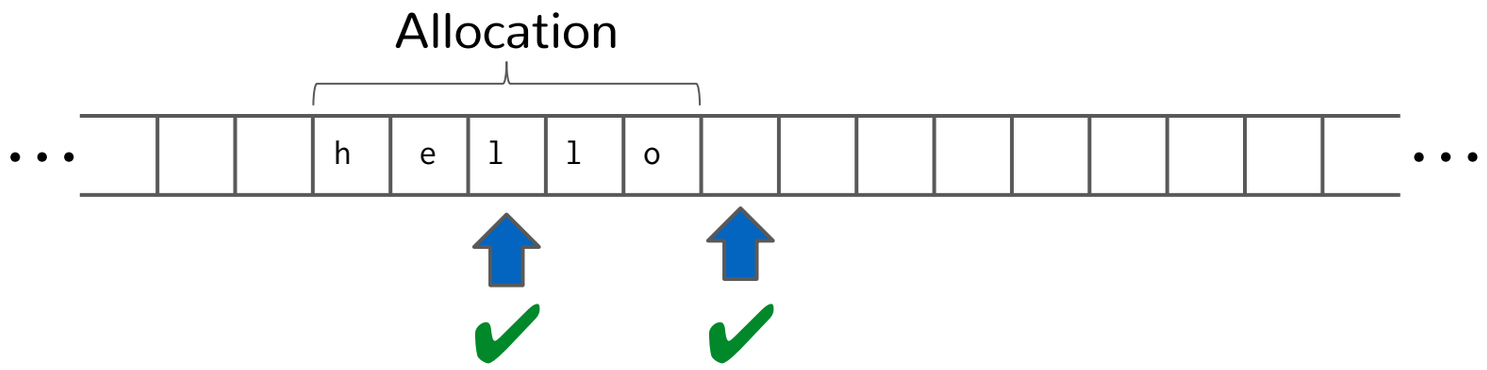




Challenge with Wasm today

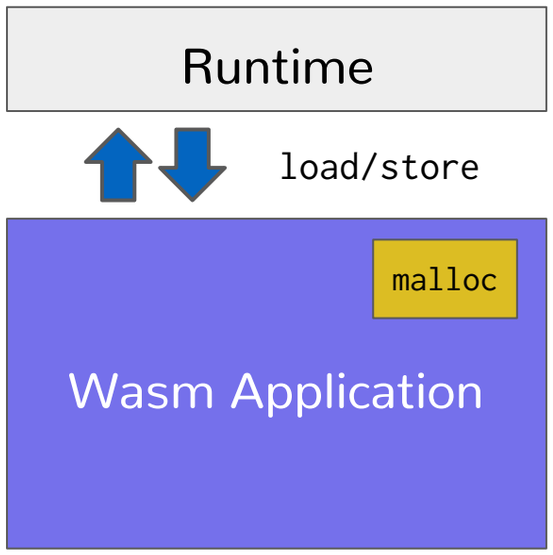
- Wasm runtime isn't aware of allocations
  - Can't enforce bounds checks!

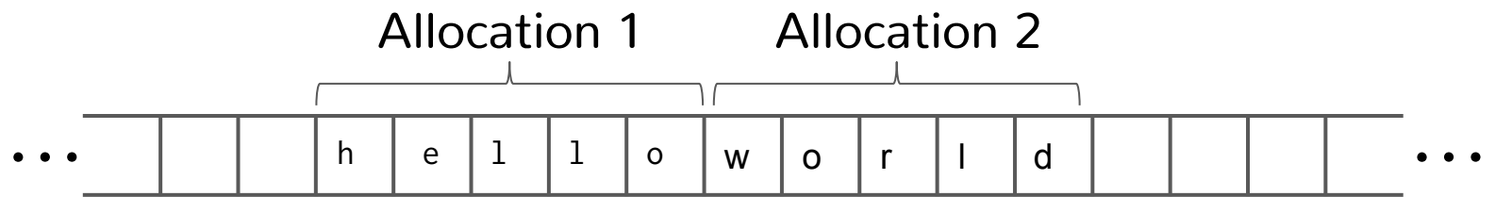




MS-Wasm: make safety explicit

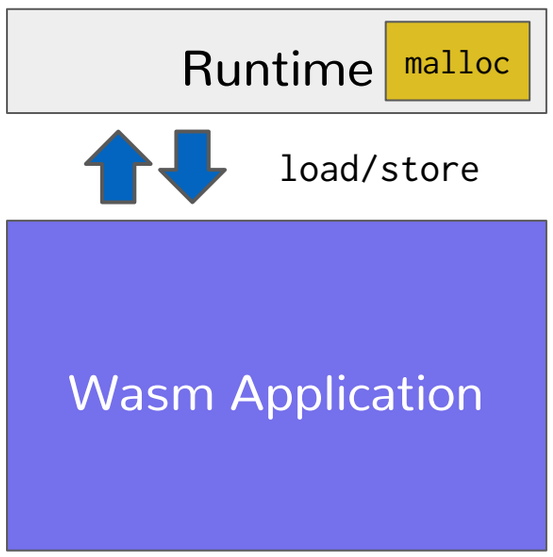
- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations

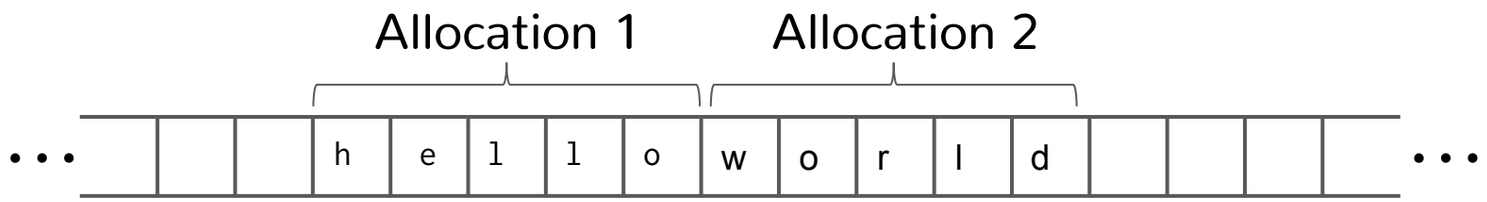




MS-Wasm: make safety explicit

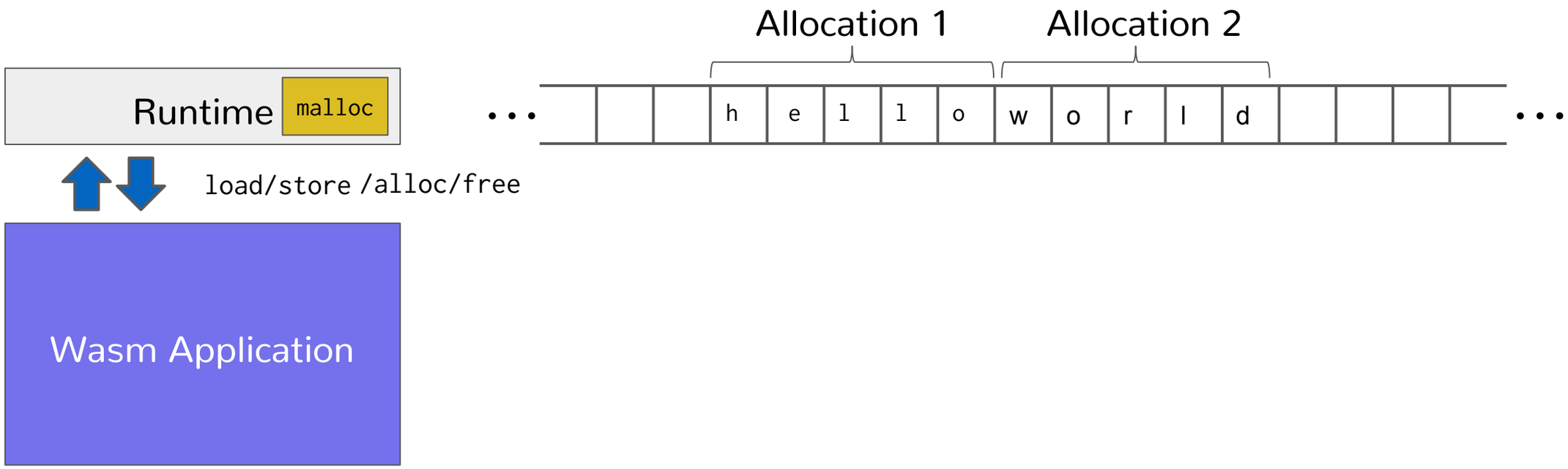
- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations





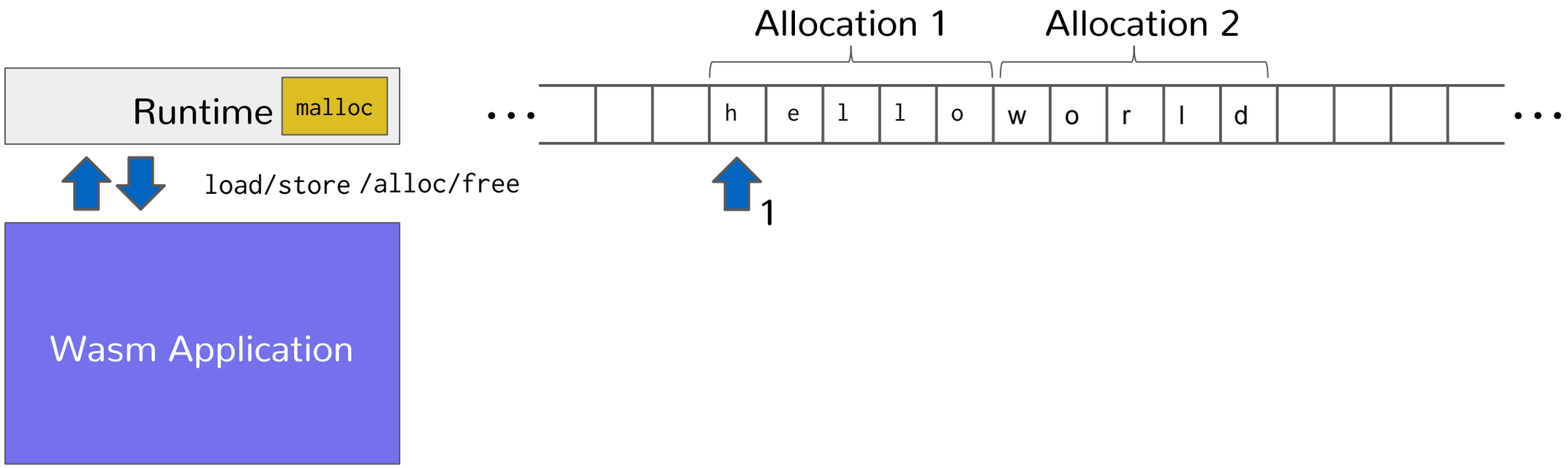
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



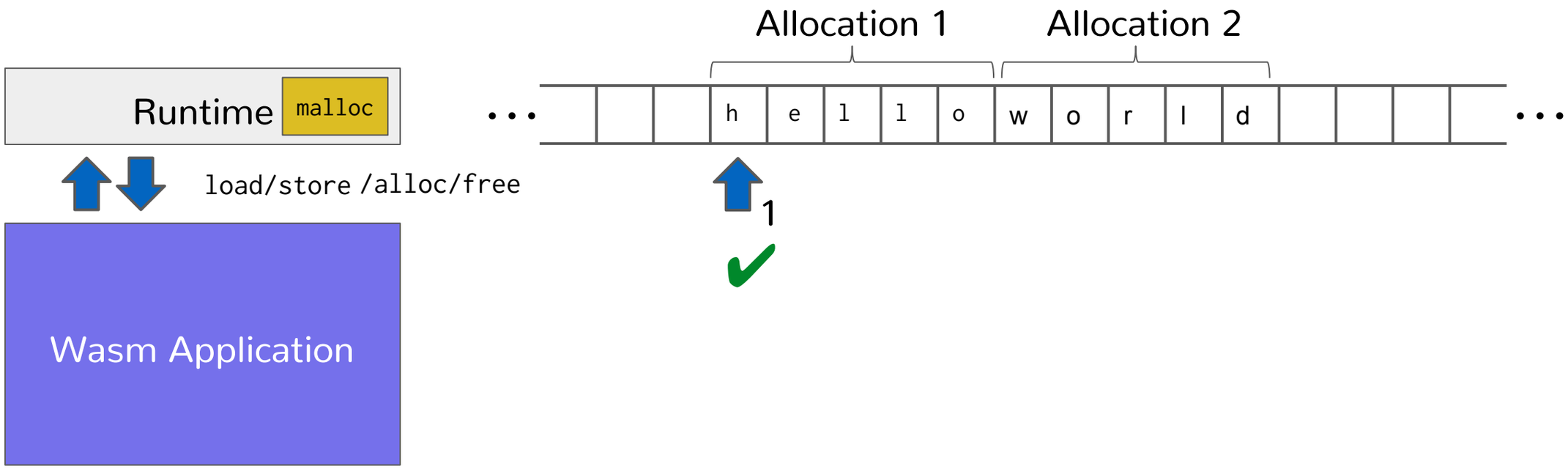
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



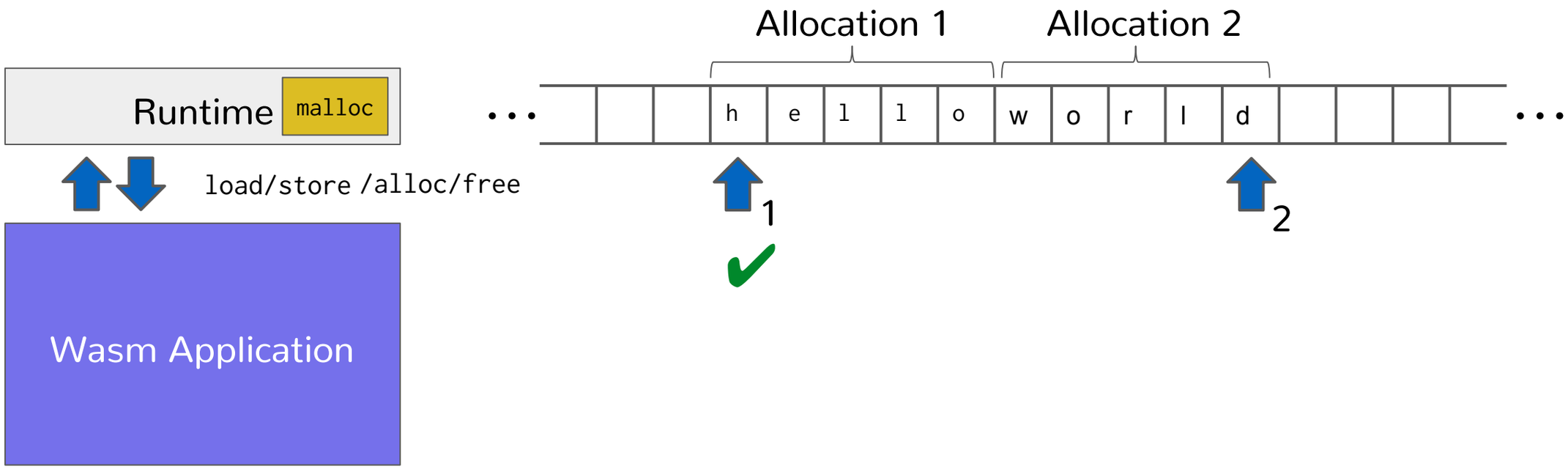
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



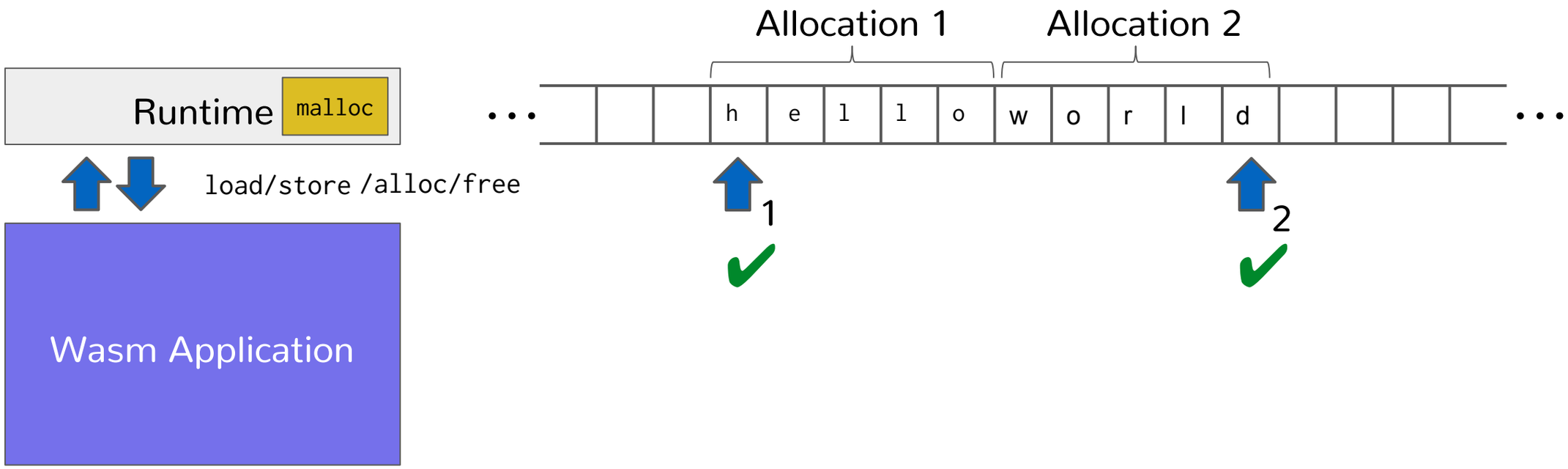
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



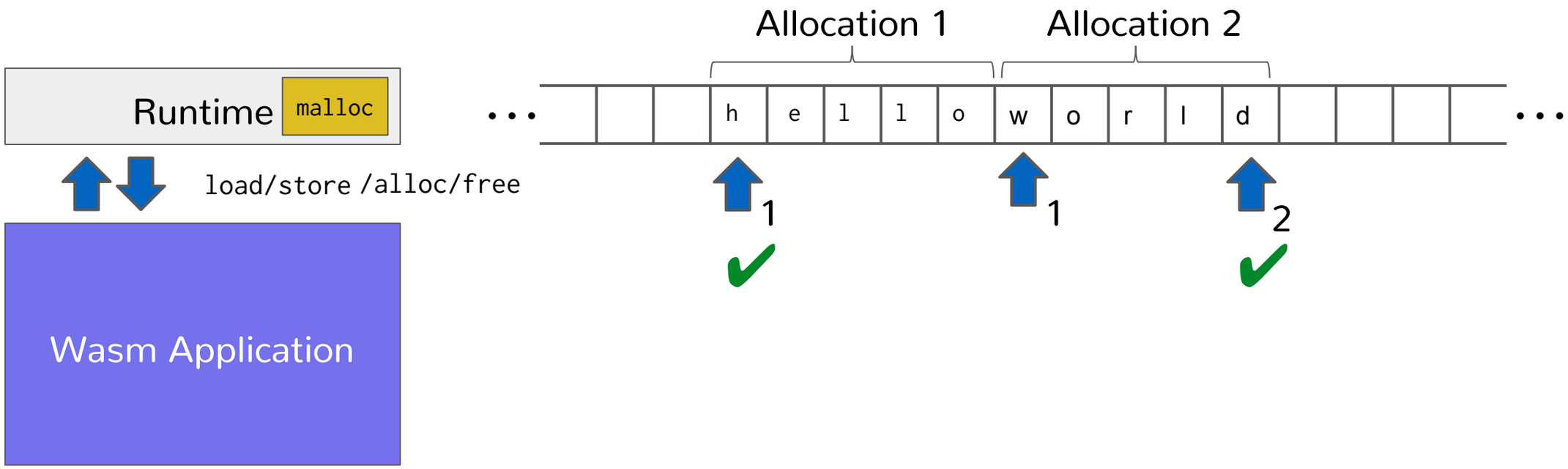
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



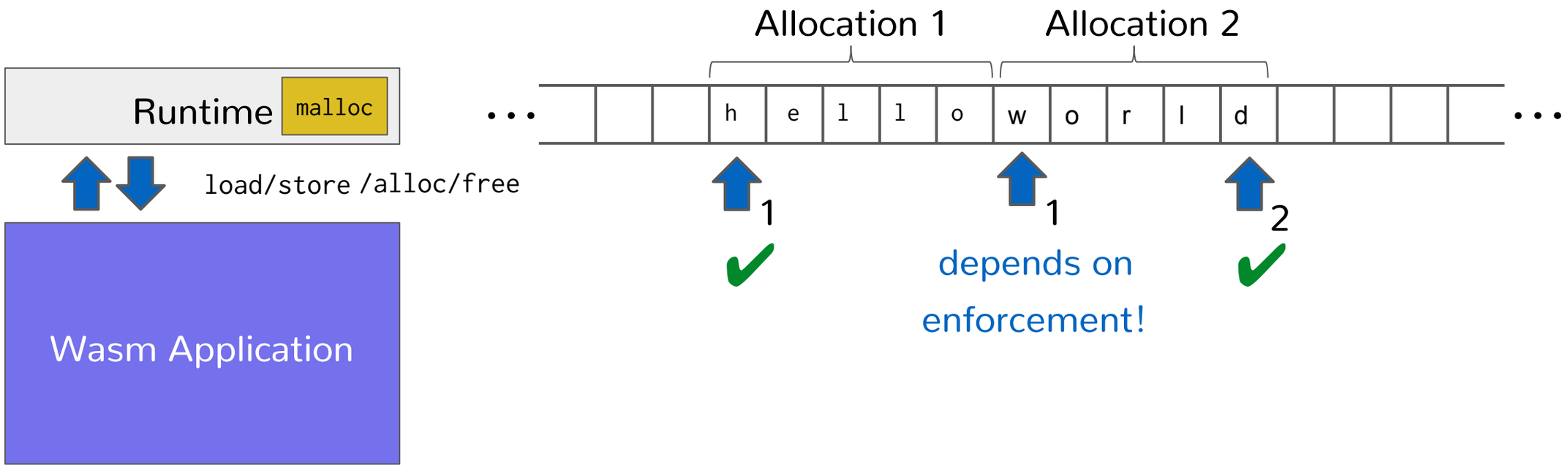
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



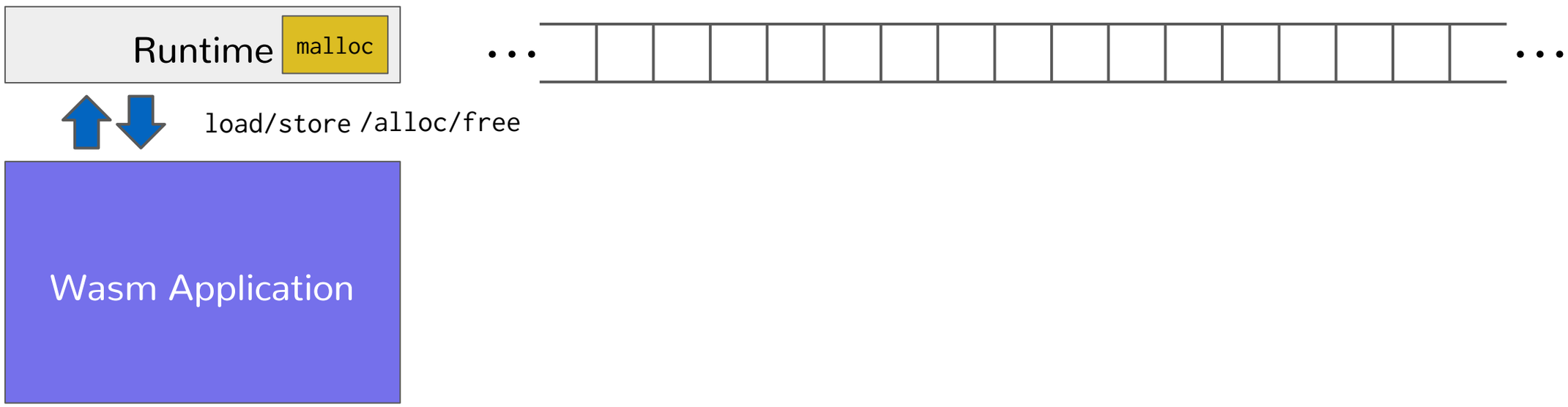
MS-Wasm: make safety explicit

- MS-Wasm runtime is aware of allocations
  - Allocations are explicit
  - Typed handles associate wtih allocations



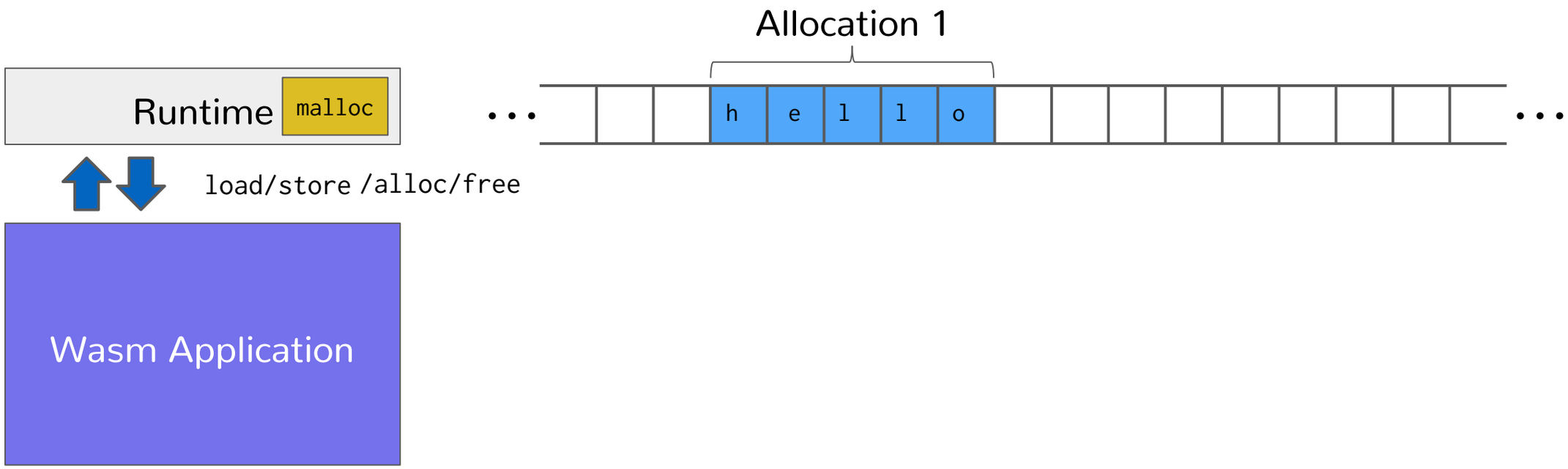
Leveraging available hardware

- Example: ARM's memory tagging extensions
  - Color allocations and handles
  - Enforcement: compare handle and allocation colors



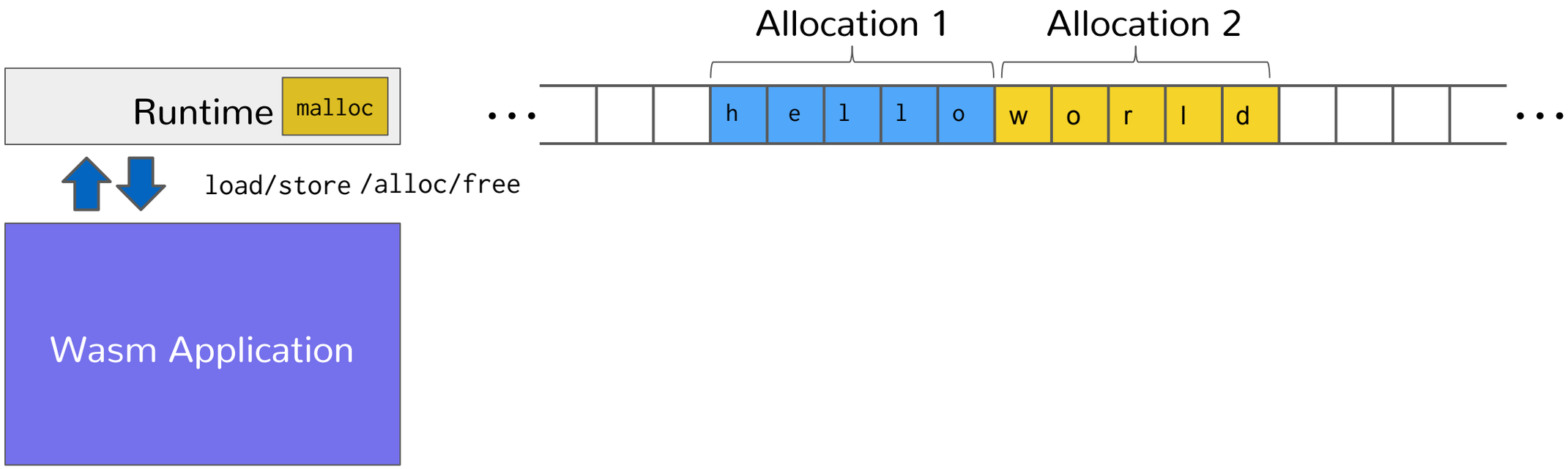
Leveraging available hardware

- Example: ARM's memory tagging extensions
  - Color allocations and handles
  - Enforcement: compare handle and allocation colors



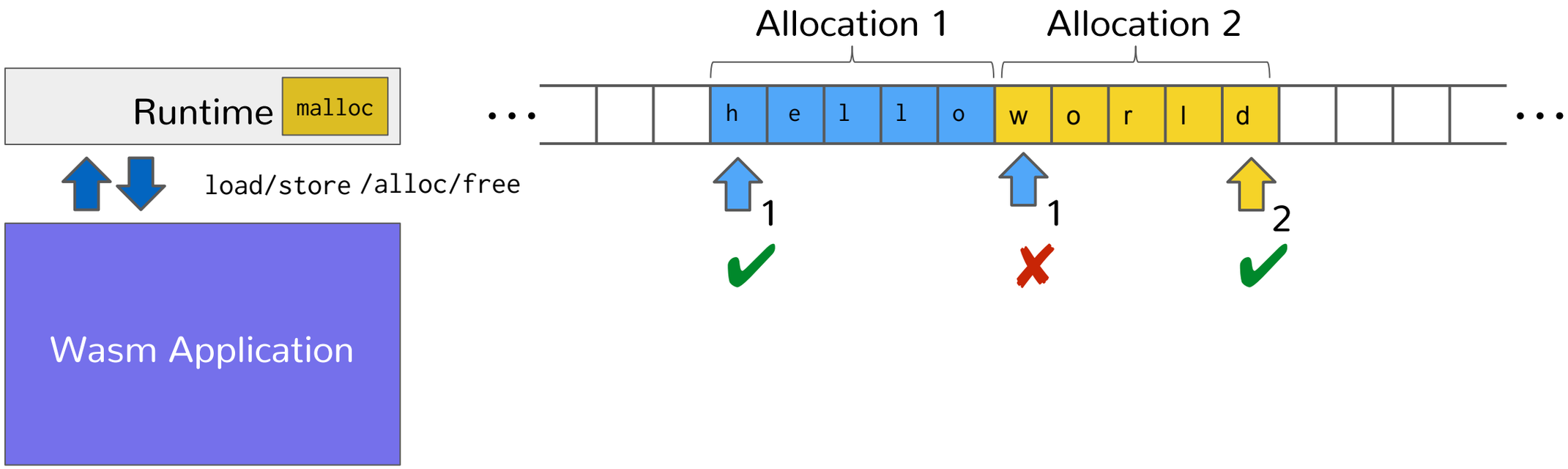
Leveraging available hardware

- Example: ARM's memory tagging extensions
  - Color allocations and handles
  - Enforcement: compare handle and allocation colors



Leveraging available hardware

- Example: ARM's memory tagging extensions
  - Color allocations and handles
  - Enforcement: compare handle and allocation colors



Gateway to memory safety

- Compile program to MS-Wasm once
- Security can improve over time
  - No enforcement today
  - MTE tomorrow
  - CHERI next year
  - MTE+CHERI or ??? the year after

- Prevent the repeat of the last 30 years of attacks

Going forward

- What are other extensions that can improve security?
- Can we do this without complete buy in?
  - Can we make Wasm extensible? (Microcode for the Web?)