# Better C++ For Clang

Attention - public document!

Contact: cbruni@chromium.org
Tracking Bug: https://crbug.com/v8/11263
Status: **Inception** | Draft | Accepted | Done

## LGTMs

| Name | LGTM |
|---|---|
| verwaest@ | NoLGTM \| LGTM |
| leszeks@ | NoLGTM \| LGTM |
| $YOU | NoLGTM \| LGTM |

## Background

V8's Handles prevent essential optimizations in our C++ code. This happens all over the place and we have been very careful in the past to not write too much unhandlified code.
Additionally we see poorly generated code due to how we access heap objects from C++. Clang does perform as well as one would expect, even on seemingly trivial examples.

I suggest to dehandlify code in places where it's not necessary (and low risk) for some immediate gains before conservative stack scanning is fully implemented:

- `const` methods should not use handles
- Prevent derefing a handle more than once if no GC happens in between and we already use raw pointers
- Make sure clang knows about immovable objects and optimizes accesses to them

## Example 1: Handles

Let's have a look at Factory::NewPropertyDecriptorObject as an example of a simple handlified method that suffers from various issues.

Basic version:

```
545 Handle<PropertyDescriptorObject> Factory::NewPropertyDescriptorObject() {
546   Handle<PropertyDescriptorObject> object =
547     Handle<PropertyDescriptorObject>::cast(
548       NewStruct(PROPERTY_DESCRIPTOR_OBJECT_TYPE, AllocationType::kYoung));
549   object->set_flags(0);
550   object->set_value(*the_hole_value(), SKIP_WRITE_BARRIER);
551   object->set_get(*the_hole_value(), SKIP_WRITE_BARRIER);
```

```
552  object->set_set(*the_hole_value(), SKIP_WRITE_BARRIER);
553  return object;
```

Release disassembly:

```
...
0x0000000000a054b0 <+16>:    e8 db 8e ff ff       call 0x9fe390 <NewStruct(...)>
0x0000000000a054b5 <+21>:    48 8b 08             mov rcx,QWORD PTR [rax]
0x0000000000a054b8 <+24>:    c7 41 03 00 00 00 00 mov DWORD PTR [rcx+0x3],0x0
0x0000000000a054bf <+31>:    48 8b 08             mov rcx,QWORD PTR [rax]
0x0000000000a054c2 <+34>:    8b 93 98 00 00 00    mov edx,DWORD PTR [rbx+0x98]
0x0000000000a054c8 <+40>:    89 51 07             mov DWORD PTR [rcx+0x7],edx
0x0000000000a054cb <+43>:    48 8b 08             mov rcx,QWORD PTR [rax]
0x0000000000a054ce <+46>:    8b 93 98 00 00 00    mov edx,DWORD PTR [rbx+0x98]
0x0000000000a054d4 <+52>:    89 51 0b             mov DWORD PTR [rcx+0xb],edx
0x0000000000a054d7 <+55>:    48 8b 08             mov rcx,QWORD PTR [rax]
0x0000000000a054da <+58>:    8b 93 98 00 00 00    mov edx,DWORD PTR [rbx+0x98]
0x0000000000a054e0 <+64>:    89 51 0f             mov DWORD PTR [rcx+0xf],edx
0x0000000000a054e3 <+67>:    48 83 c4 08          add rsp,0x8
0x0000000000a054e7 <+71>:    5b                   pop rbx
0x0000000000a054e8 <+72>:    5d                   pop rbp
0x0000000000a054e9 <+73>:    c3                   ret
```

We observe the following Issues:

- `object` dereferences not constant-folded
- `*the_hole_value()` is not constant-folded

Optimized version:

```
546  Handle<PropertyDescriptorObject> object = …
...  DisallowGarbageCollectionWithGcMoleSupport no_gc; // to be decided
549  PropertyDescriptorObject raw = *object;
550  Object the_hole = *the_hole_value();
551  raw.set_flags(0);
552  raw.set_value(the_hole, SKIP_WRITE_BARRIER);
553  raw.set_get(the_hole, SKIP_WRITE_BARRIER);
554  raw.set_set(the_hole, SKIP_WRITE_BARRIER);
555  return object;
```

Release disassembly:

```
...
0x0000000000a05490 <+16>:    e8 db 8e ff ff       call 0x9fe370 <NewStruct(...)>
0x0000000000a05495 <+21>:    48 8b 08             mov rcx,QWORD PTR [rax]
0x0000000000a05498 <+24>:    8b 93 98 00 00 00    mov edx,DWORD PTR [rbx+0x98]
0x0000000000a0549e <+30>:    c7 41 03 00 00 00 00 mov DWORD PTR [rcx+0x3],0x0
0x0000000000a054a5 <+37>:    89 51 07             mov DWORD PTR [rcx+0x7],edx
0x0000000000a054a8 <+40>:    89 51 0b             mov DWORD PTR [rcx+0xb],edx
0x0000000000a054ab <+43>:    89 51 0f             mov DWORD PTR [rcx+0xf],edx
0x0000000000a054ae <+46>:    48 83 c4 08          add rsp,0x8
0x0000000000a054b2 <+50>:    5b                   pop rbx
0x0000000000a054b3 <+51>:    5d                   pop rbp
0x0000000000a054b4 <+52>:    c3                   ret
```

# Example 2: Type Checks

The second example covers a simple const method that does instance type checking:

```
bool Object::IsFoo(Isolate* isolate) const {
  if (IsJSGeneratorObject()) return false;
  if (IsJSPromise()) return false;
  return true;
}
0x0000000000c330d0 <+0>:  55                push   rbp
0x0000000000c330d1 <+1>:  48 89 e5          mov    rbp,rsp
0x0000000000c330d4 <+4>:  48 8b 07          mov    rax,QWORD PTR [rdi]
0x0000000000c330d7 <+7>:  a8 01             test   al,0x1
0x0000000000c330d9 <+9>:  74 21             je     0xc330fc <+44>
0x0000000000c330db <+11>: 48 b9 00 00 00 00 ff ff ff ff
                                            movabs rcx,0xffffffff00000000
0x0000000000c330e5 <+21>: 48 21 c1          and    rcx,rax
0x0000000000c330e8 <+24>: 8b 50 ff          mov    edx,DWORD PTR [rax-0x1] // Load Map
0x0000000000c330eb <+27>: 0f b7 54 11 07    movzx  edx,WORD PTR [rcx+rdx*1+0x7]
0x0000000000c330f0 <+32>: 81 c2 e9 fb ff ff add    edx,0xfffffbe9 // 0xffffffff - 0x417
0x0000000000c330f6 <+38>: 66 83 fa 03       cmp    dx,0x3         // range check
0x0000000000c330fa <+42>: 73 07             jae    0xc33103 <+51>
0x0000000000c330fc <+44>: a8 01             test   al,0x1
0x0000000000c330fe <+46>: 0f 94 c0          sete   al
0x0000000000c33101 <+49>: 5d                pop    rbp
0x0000000000c33102 <+50>: c3                ret
0x0000000000c33103 <+51>: 48 83 c9 07       or     rcx,0x7
0x0000000000c33107 <+55>: 8b 40 ff          mov    eax,DWORD PTR [rax-0x1] // load Map
0x0000000000c3310a <+58>: 0f b7 04 01       movzx  eax,WORD PTR [rcx+rax*1]
0x0000000000c3310e <+62>: 3d 33 04 00 00    cmp    eax,0x433
0x0000000000c33113 <+67>: 0f 95 c0          setne  al
0x0000000000c33116 <+70>: 5d                pop    rbp
0x0000000000c33117 <+71>: c3                ret
```

Issues:

- Smi check is not hoisted
- Map gets loaded twice

```
bool Object::IsFoo(Isolate* isolate) const {
  if (IsSmi()) return false;
  if (IsJSGeneratorObject(isolate)) return false;
  if (IsJSPromise(isolate)) return false;
  return true
}
0x0000000000c330d0 <+0>:  55                push   rbp
0x0000000000c330d1 <+1>:  48 89 e5          mov    rbp,rsp
0x0000000000c330d4 <+4>:  48 8b 07          mov    rax,QWORD PTR [rdi]
```

```
0x0000000000c330d7 <+7>:  a8 01              test   al,0x1
0x0000000000c330d9 <+9>:  74 14              je     0xc330ef <E+31>
// load map
0x0000000000c330db <+11>: 8b 48 ff           mov    ecx,DWORD PTR [rax-0x1]
0x0000000000c330de <+14>: 0f b7 4c 0e 07     movzx  ecx,WORD PTR [rsi+rcx*1+0x7]
0x0000000000c330e3 <+19>: 81 c1 e9 fb ff ff  add    ecx,0xfffffbe9 // 0xffffffff - 0x417
0x0000000000c330e9 <+25>: 66 83 f9 03        cmp    cx,0x3         // range check
0x0000000000c330ed <+29>: 73 04              jae    0xc330f3 <+35>
0x0000000000c330ef <+31>: 31 c0              xor    eax,eax
0x0000000000c330f1 <+33>: 5d                 pop    rbp
0x0000000000c330f2 <+34>: c3                 ret
// load map
0x0000000000c330f3 <+35>: 8b 40 ff           mov    eax,DWORD PTR [rax-0x1]
0x0000000000c330f6 <+38>: 0f b7 44 06 07     movzx  eax,WORD PTR [rsi+rax*1+0x7]
0x0000000000c330fb <+43>: 3d 33 04 00 00     cmp    eax,0x433
0x0000000000c33100 <+48>: 0f 95 c0           setne  al
0x0000000000c33103 <+51>: 5d                 pop    rbp
0x0000000000c33104 <+52>: c3                 ret
```

If we manually tweak the code, and use more low-level helpers we get even better results. The resulting code is faster, but also quite a bit more verbose:

```
bool Object::IsCodeLike(Isolate* isolate) const {
  if (IsSmi()) return false;
  InstanceType type = HeapObject::cast(*this).map(isolate).instance_type();
  if (InstanceTypeChecker::IsJSGeneratorObject(type)) return false;
  if (InstanceTypeChecker::IsJSPromise(type)) return false;
  return true;
}
0x0000000000c330d0 <+0>:  55                 push   rbp
0x0000000000c330d1 <+1>:  48 89 e5           mov    rbp,rsp
0x0000000000c330d4 <+4>:  48 8b 07           mov    rax,QWORD PTR [rdi]
0x0000000000c330d7 <+7>:  a8 01              test   al,0x1
0x0000000000c330d9 <+9>:  74 14              je     0xc330ef <+31>
0x0000000000c330db <+11>: 8b 40 ff           mov    eax,DWORD PTR [rax-0x1]
0x0000000000c330de <+14>: 0f b7 44 06 07     movzx  eax,WORD PTR [rsi+rax*1+0x7]
0x0000000000c330e3 <+19>: 8d 88 e9 fb ff ff  lea    ecx,[rax-0x417] // GENERATOR TYPE
0x0000000000c330e9 <+25>: 66 83 f9 03        cmp    cx,0x3 // range check
0x0000000000c330ed <+29>: 73 04              jae    0xc330f3 <+35>
0x0000000000c330ef <+31>: 31 c0              xor    eax,eax
0x0000000000c330f1 <+33>: 5d                 pop    rbp
0x0000000000c330f2 <+34>: c3                 ret
0x0000000000c330f3 <+35>: 0f b7 c0           movzx  eax,ax
0x0000000000c330f6 <+38>: 3d 33 04 00 00     cmp    eax,0x433 // JS_PROMISE_TYPE
0x0000000000c330fb <+43>: 0f 95 c0           setne  al
0x0000000000c330fe <+46>: 5d                 pop    rb
0x0000000000c330ff <+47>: c3                 ret
```

By manually loading the instance type we get the expected result. The map and instance type is loaded only once. However, most of this work should happen automatically in order to keep the code more maintainable.

While this might seem like a trivial example, a lot of the small helper functions get inlined and we quickly end up in a situation where multiple type checks happen but they are not optimized properly.

# Example 3: Map Checks

```
bool Object::Check() const {
  if (IsSmi()) return false;
  HeapObject o = HeapObject::cast(*this);
  Map map1 = o.map();
  Map map2 = o.map();
  return map1.ptr() == map2.ptr(); // should be no-op
}
0x0000000000c330d0 <+0>:  55                push   rbp
0x0000000000c330d1 <+1>:  48 89 e5          mov    rbp,rsp
0x0000000000c330d4 <+4>:  48 8b 07          mov    rax,QWORD PTR [rdi]
0x0000000000c330d7 <+7>:  a8 01             test   al,0x1
0x0000000000c330d9 <+9>:  75 04             jne    0xc330df <+15>
0x0000000000c330db <+11>: 31 c0             xor    eax,eax
0x0000000000c330dd <+13>: 5d                pop    rbp
0x0000000000c330de <+14>: c3                ret
0x0000000000c330df <+15>: 48 b9 00 00 00 00 ff ff ff ff
                                            movabs rcx,0xffffffff00000000
0x0000000000c330e9 <+25>: 48 21 c1          and    rcx,rax
0x0000000000c330ec <+28>: 8b 50 ff          mov    edx,DWORD PTR [rax-0x1]
0x0000000000c330ef <+31>: 48 09 ca          or     rdx,rcx
0x0000000000c330f2 <+34>: 8b 40 ff          mov    eax,DWORD PTR [rax-0x1]
0x0000000000c330f5 <+37>: 48 09 c8          or     rax,rcx
0x0000000000c330f8 <+40>: 48 39 c2          cmp    rdx,rax
0x0000000000c330fb <+43>: 0f 94 c0          sete   al
0x0000000000c330fe <+46>: 5d                pop    r
```

Changing `HeapObject::map_word` from `MapField::Relaxed_Load` to `MapField::load` yields the expected result:

```
DEF_GETTER(HeapObject, map_word, MapWord) {
  return MapField::load(isolate, *this); // WARNING HACK!
}
bool Object::Check() const {
 ...
}
0x0000000000c32100 <+0>:  55                push   rbp
0x0000000000c32101 <+1>:  48 89 e5          mov    rbp,rsp
0x0000000000c32104 <+4>:  48 8b 07          mov    rax,QWORD PTR [rdi]
0x0000000000c32107 <+7>:  a8 01             test   al,0x1
```

```
0x0000000000c32109 <+9>:  24 01             and    al,0x1
0x0000000000c3210b <+11>: 5d                pop    rbp
0x0000000000c3210c <+12>: c3                ret
```

# Fixes

## GCMole & DisallowGarbageCollection

Gcmole currently ignores everything that happens inside a DisallowGarbageCollection scope. This is contrary to the practice of documenting that a function cannot cause a GC.
However, there are valid cases where gcmole should skip validation since it would find false positives.

## Low Risk Items:

- Introduce distinction between DisallowGarbageCollection with and without “skip gcmole”
- Avoid handles when initializing objects in the factory
- Consistently use SKIP_WRITE_BARRIER for ReadOnlyRoots
- Consistently use SKIP_WRITE_BARRIER where possible when initializing objects
- Dehandlify simple API accessors
- Use const for instance methods where possible
- Make sure Clang handles ReadOnlyRoots as constants
- Make sure Clang handles Maps as non-moving objects

## Higher Risk Items:

- Avoid dereferencing handles more than once if no GC can happen in between (requires more adequate tooling support, probably from gcmole)