Ruby - Feature #21353

Add shape_id to RBasic under 32 bit

05/20/2025 08:18 AM - byroot (Jean Boussier)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

Currently on 64bit systems, for every types, the shape_id is stored inside the RBasic.flags field, and is 32bit long.

However, on 32bit systems like i686 and WASM, it is much more complicated.
For T_OBJECT, T_CLASS and T_MODULE, the shape_id is stored as part of "user flags" in FL_USER4-19, and for all other types it's stored alongside the instance variable in the generic_fields_tbl, which means a hash lookup is required to access it.

This situation makes a lot of routine noticeably more complicated, with numerous codepath taken only by 32bit systems, because to avoid doing two hash-lookup per ivar access, the code need a lot of contortions.
You can look for SHAPE_IN_BASIC_FLAGS to have an idea of the added complexity.

In addition, it forces us to duplicate some bits of information. For instance RUBY_FL_FREEZE is redundant with the shape_id. The shape already record that the object is frozen, and the only reason RUBY_FL_FREEZE hasn't been eliminated is because on 32bits the shape_id isn't store inline for some objects.

Similarly, RUBY_FL_EXIVAR is redundant with the shape_id, because to know whether an object has ivars, you can simply check if shape_id == 0.

Reclaiming these bits would be very useful for Ractors, as we'd need two bits in objects to be able to implement lightwieght locks.

Yet another complication, is that on 32bit systems, the shape_id is only 16bits long.
I have the project to use the upper bits of the shape_id to store metadata, such as the frozen and too_complex status, allowing to test for this without chasing a pointer: https://github.com/ruby/ruby/pull/13289.
But this currently can't be done on 32bit sytems, both because accessing the shape_id might require a hash-lookup, and also because it's only 16bits long, so every single bit used for tagging severely restrict the maximum number of shapes.

Proposal

To simplify all this, we propose that on 32bit systems, we add a VALUE shape_id in RBasic:

```
struct RBasic {
    VALUE flags;
    const VALUE klass;
#if RBASIC_SHAPE_ID_FIELD
    VALUE shape_id;
#endif
}
```

This ensure that on 32bits, all objects have their shape_id always at the same predictable offset, and 32bits long.

As you can see on the pull request, it simplify the code quite significantly: https://github.com/ruby/ruby/pull/13341, and there's more cleanup that can be done.

The downside obviously is that on 32bit, objects would grow from 20B to 24B.

Pull Request

You can find the proposed patch at: https://github.com/ruby/ruby/pull/13341

cc @tenderlovemaking (Aaron Patterson) and @jhawthorn (John Hawthorn)

Also FYI @katei (Yuta Saito) because as the maintainer of WASM I assume this impact you the most.

Associated revisions

Revision f483befd - 05/26/2025 08:31 AM - jhawthorn (John Hawthorn)

Add shape_id to RBasic under 32 bit

This makes RBobject 4B larger on 32 bit systems
but simplifies the implementation a lot.

[Feature #21353]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision e27404af - 06/03/2025 07:15 PM - byroot (Jean Boussier)

Use all 32bits of shape_id_t on all platforms

Followup: https://github.com/ruby/ruby/pull/13341 / [Feature #21353]

Even thought shape_id_t has been make 32bits, we were still limited
to use only the lower 16 bits because they had to fit alongside attr_index_t
inside a uintptr_t in inline caches.

By enlarging inline caches we can unlock the full 32bits on all
platforms, allowing to use these extra bits for tagging.

History

#1 - 05/20/2025 08:11 PM - Dan0042 (Daniel DeLorme)

In general this sounds like a good idea, but I think it would be better to have a struct that works the same way for both 32bit and 64bit systems, and also avoids reserving an entire 32 bits, which is overkill for shape ids.

What about this?

```
struct RBasic {
    uint32 flags;
    uint32 flags2;
    const VALUE klass;
}

#define shape_id(ptr) (ptr->flags2)
```

Not relying on #if RBASIC_SHAPE_ID_FIELD would imho reduce overall complexity and simplify maintenance.

Future-proofing: if we run out of bits in flags, we can redefine the shape_id macro and use some bits from flags2

#2 - 05/20/2025 10:49 PM - jhawthorn (John Hawthorn)

@Dan0042 We've been finding a lot of cases where flags actually make sense as part of the shape rather than flags, like FL_FROZEN and FL_EXIVAR per the description, but also possibly object_id and likely more (ex. capacity, FL_EMBEDDED) but it is hard for us to follow that path when shapes are so different under 32-bit. So this is actually freeing up flag bits as you are hoping to.

Declaring RBasic that way is an interesting idea, but I think creates a lot more complexity. A lot of things currently expect two VALUE sized objects at the start of each object to be flags and klass, so that would be quite a large change. As well we'd need to redefine what flags are everywhere they're used to be a uint64_t rather than a VALUE (or make it a uint32_t?? either way a large change). So I think the approach we took with our PR is the least invasive way to get there, there aren't that many references to RBASIC_SHAPE_ID_FIELD and to me they all read as being in a place they make sense.

#3 - 05/22/2025 03:20 AM - Dan0042 (Daniel DeLorme)

jhawthorn (John Hawthorn) wrote in #note-2:

> it is hard for us to follow that path when shapes are so different under 32-bit

So we're in agreement here, and in fact I think shapes should be **exactly the same** under both 32bit and 64bit.

> Declaring RBasic that way is an interesting idea, but I think creates a lot more complexity. A lot of things currently expect two VALUE sized objects at the start of each object to be flags and klass so that would be quite a large change. As well we'd need to redefine what flags are everywhere they're used to be a uint64_t rather than a VALUE (or make it a uint32_t?? either way a large change). So I think the approach we took with our PR is the least invasive way to get there, there aren't that many references to RBASIC_SHAPE_ID_FIELD and to me they all read as being in a place they make sense.

I can only see it as simplification, certainly not complexity. And I don't think it's nearly as big a change as you imagine. On 64 bit the memory layout doesn't even change. On 32 bit I can't think of anything that requires the klass to be at exactly 4 bytes from the struct start. On 64 bit flags is defined to be a 64-bit VALUE, but only the first 32 bits are ever used (apart from the shapes stuff), so changing the type to uint32_t would not change any behavior, and would reflect the actual number of flag bits; imho using 64 bits for the flags was kind of a hack in the first place, and this is the opportunity for a long-overdue cleanup.

I actually tried changing Ruby 3.1 (before the introduction of shapes) to use 2 uint32_t fields as above, and everything compiles and works! Much less of a large change than one might think.

#4 - 05/26/2025 08:32 AM - jhawthorn (John Hawthorn)

- Status changed from Open to Closed

Applied in changeset git|f483befd9065d159d3a944b87fe26179c5373c30.

Add shape_id to RBasic under 32 bit

This makes RBobject 4B larger on 32 bit systems
but simplifies the implementation a lot.

[Feature #21353]

Co-authored-by: Jean Boussier byroot@ruby-lang.org