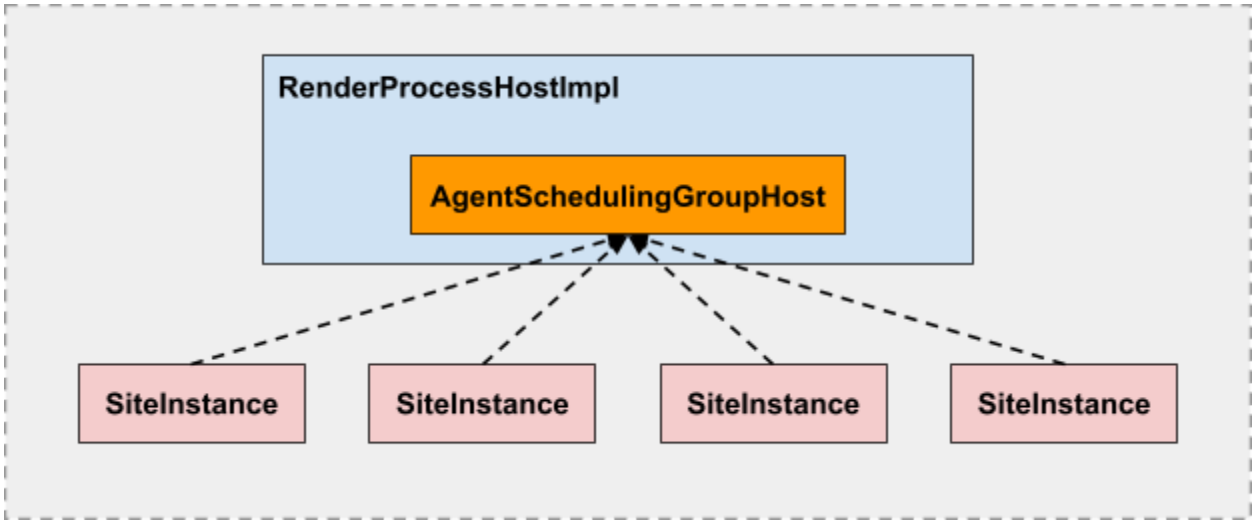
AgentSchedulingGroupHost per SiteInstance

domfarolino@
Bug: crbug.com/1149830

Background

Currently AgentSchedulingGroupHost is owned by RenderProcessHostImpl's `|user_data_|` map [cs]. The UserData owns a *single* AgentSchedulingGroupHost, therefore AgentSchedulingGroupHost is 1:1 with RenderProcessHost, and multiple SiteInstances may share the same AgentSchedulingGroupHost on platforms where they share the same RenderProcessHost (i.e., where SiteIsolation is not fully enabled).

Configuring AgentSchedulingGroupHost to be 1:1 with RenderProcessHostImpl is the best way to isolate potential IPC & mojo ordering issues that are introduced by "MBI" mode, which dissociates the AgentSchedulingGroupHost interface/legacy IPC channel from the process-global legacy IPC channel.



This nicely ties the lifetime of an AgentSchedulingGroupHost to its associated RenderProcessHostImpl. The AgentSchedulingGroupHost is destructed when the RenderProcessHostImpl is destroyed (asynchronously [cs]), as this is when its `|user_data_|` map is destroyed.

Design: 1:1 with SiteInstance

This is currently being implemented in CL 2536756.

While the above (current) configuration is ideal for isolating potential scheduling issues, we want to implement a mode where AgentSchedulingGroupHost is 1:1 with SiteInstance, to support more granular per-AgentCluster scheduling (especially where Site Isolation isn't fully enabled). In order to do this, we'll need to allocate an AgentSchedulingGroupHost per SiteInstance, and associate each one of them with the process host.

Parameterizing the flag

Currently we have a flag [cs] that essentially amounts to turning on MBI mode when enabled, or reverting back to the "legacy" pre-MBI mode when disabled. When enabled, MBI mode simply disassociates the AgentSchedulingGroupHost interface from the process-global legacy IPC channel. The AgentShedulingGroup communication is effectively scheduled independently.

MBI mode is still limited to one AgentSchedulingGroupHost per RenderProcessHostImpl, so we'll need to parameterize the flag to allow the following configurations:

1. Disabled: Legacy
2. Enabled: AgentSchedulingGroupHost-per-RenderProcessHostImpl
3. Enabled: AgentSchedulingGroupHost-per-SiteInstance [NEW]

Changing the AgentSchedulingGroupHost Allocation Model

Currently RenderProcessHost owns an AgentSchedulingGroupHost indirectly, via base::SupportsUserData. The Agent~~Scheduling~~GroupHostUserData [cs] is a class holding a single `std::unique_ptr<AgentSchedulingGrouphost>`. This pointer is populated once per RenderProcessHostImpl the first time a SiteInstance calls `AgentSchedulingGroupHost::Get()`. The AgentSchedulingGroupHost then simply early-returned for all subsequent calls:

```
AgentSchedulingGroupHost* AgentSchedulingGroupHost::Get() {
    if (RPHIUserData(kAgentSchedulingGroupHost))
        return RPHIUserData(kAgentSchedulingGroupHost)->asgh();

    // The user data doesn't exist, so we have to:
    //    1.) Create it
    //    2.) Create a new AgentSchedulingGroupHost (user data will own it)
    auto new_user_data = new ASGUserData(new AgentSchedulingGroupHost());
    return new_user_data->asgh();
};
```

Instead of having AgentSchedulingGroupHostUserData hold a single `std::unique_ptr<AgentSchedulingGrouphost>`, I propose maintaining a `std::set<std::unique_ptr<AgentSchedulingGroupHost>>`, one host for each SiteInstance:

```
class AgentSchedulingGroupHostUserData : base::SupportsUserData::Data {
    std::set<std::unique_ptr<AgentSchedulingGroupHost>> owned_host_set_;
#if DCHECK_IS_ON()
    std::set<const SiteInstance*> host_set_;
#endif
};
```

Whenever an AgentSchedulingGroupHost is created, regardless of the MBI mode parameter state, it always gets inserted into and owned by the `|owned_host_set_|`.

Therefore the lifetime of an AgentSchedulingGroupHost will still be tied to its RenderProcessHost.

In AgentSchedulingGroup-per-RenderProcessHost mode, this makes things simple since we still have a single AgentSchedulingGroupHost owned by the RenderProcessHost, that outlives all associated SiteInstances, and goes away when the RenderProcessHost is destroyed.

In AgentSchedulingGroup-per-SiteInstace mode, this creates a "leak" when a SiteInstance goes away before the RenderProcessHost. This is because the AgentSchedulingGroupHost will still be kept alive by the RenderProcessHost, and therefore the renderer-side AgentSchedulingGroup will still be alive, though never used from then on. kouhei@ is investigating the non-trivial AgentSchedulingGroup shutdown sequence to avoid this leak, and properly kill the renderer-side objects. Note that experimenting in Canary/Dev is **absolutely blocked** on finishing the AgentSchedulingGroup shutdown flow and getting rid of this "leak". See the Integrating With Shut Down Flow section for details on this.

When the MBI mode parameter is in the AgentSchedulingGroup-per-SiteInstance state and DCHECK_IS_ON(), we'll also maintain the `|host_set_|` data structure solely to maintain invariants. The section below covers this.

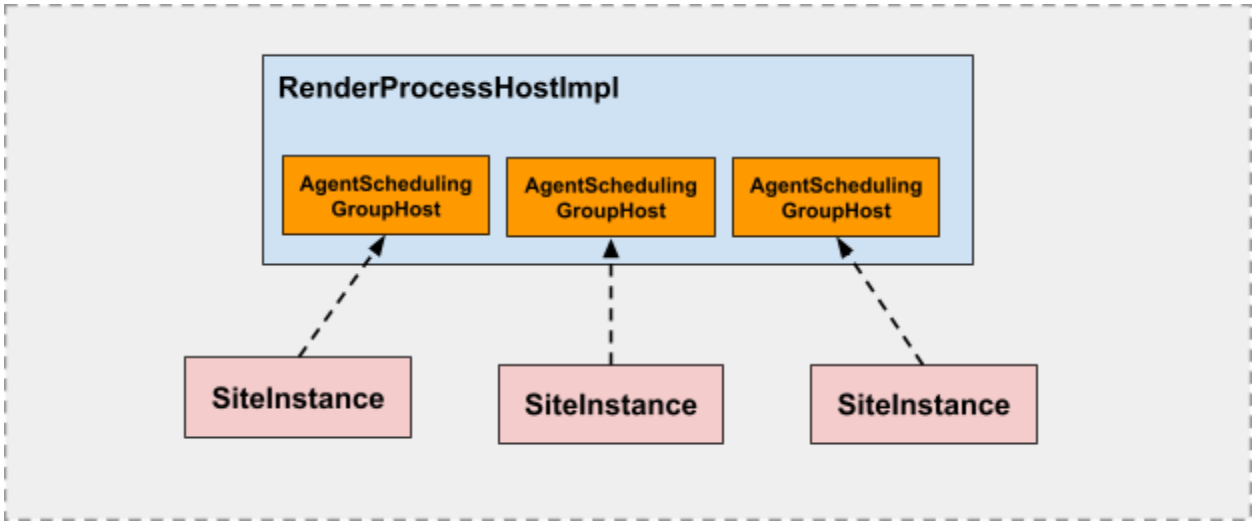
Checking & Adding Entries

Before this change, we queried the AgentSchedulingGroupHostUserData [cs] to see if the one AgentSchedulingGroupHost associated with the RenderProcessHost was available. If so, we return it. If not, we create it once.

After this change in **MBI AgentSchedulingGroup-per-SiteInstance mode**, whenever SiteInstance queries AgentShedulingGroupHost::Get() to get a new AgentSchedulingGroupHost, we perform the following steps:

1. *If DCHECK is on*: DCHECK that no entry for the given SiteInstance already exists in the `|host_set_|`
    a. This is important, because a SiteInstance should never request multiple AgentSchedulingGroupHosts from the same RenderProcessHost. The |host_map_| protects this invariant
2. Create a new AgentSchedulingGroupHost
3. Insert it into the `|owned_host_set_|` as usual
4. *If DCHECK is on*: Insert the requesting SiteInstance into the `|host_set_|`

5. Return the AgentSchedulingGroupHost to the requesting SiteInstance



This provides us with a way to know when to create new AgentSchedulingGroupHosts (and thus remotely AgentSchedulingGroups) based on the SiteInstance. Removing AgentSchedulingGroupHosts from the UserData and destroying them at the right time depend on the full shutdown flow being investigated by kouhei@. See the section below.

After this change in **MBI AgentSchedulingGroup-per-RenderProcessHost mode**, multiple SiteInstances may query AgentSchedulingGroupHost::Get() to get an AgentSchedulingGroupHost. This method will always return the same AgentSchedulingGroupHost to all requesting SiteInstances under the same RenderProcessHost. The flow will look like this:

1. If there are no entries in the `|owned_host_set_|`, create a new AgentSchedulingGroupHost and insert it into the set
2. DCHECK that there is at most one entry in the set
3. Return the single entry in the set

Basically not much has changed in this front.

Integrating With Shut Down Flow

As mentioned above, having the AgentSchedulingGroupHost lifetime tied to RenderProcessHost maintains the simplicity that we currently have, but creates a “leak” for when SiteInstances are destroyed.

When a SiteInstance gets destroyed, we must:

- Notify the renderer-side AgentSchedulingGroup object to delete itself, or else we essentially leak the object, as it will never be used again.
- Remove the relevant entry from the `AgentSchedulingGroupHostUserData` data structures when the host can safely be deleted.

A proper shut down flow will address both points above, but is very non-trivial. kouhei@ has been investigating this [design doc, prototype CL].

Experimenting with AgentSchedulingGroupHost-per-SiteInstance on even Canary/Dev **is absolutely blocked on** correctly solving the shutdown sequence that kouhei@ is exploring. However we don't believe that it strictly blocks implementing AgentSchedulingGroup-per-SiteInstance in general, since all of the work is behind a flag. Additionally, landing an implementation with this will help us:

- Set up a custom trybot with the MBI flag enabled in various configurations, so we can quantify how many things are broken
- Locally test and investigate breakages on real-world sites and tests

Hence we propose to implement AgentSchedulingGroup-per-SiteInstance while maintaining the "too-long" lifetime logic. The implementation of a proper shutdown flow will correct and solidify the lifetime problem, while not blocking the progress here.

Alternative Design: Ref Counting the AgentSchedulingGroupHost

This approach has to do with making AgentSchedulingGroupHost a ref-counted class (via `base::RefCounted`); it came out of a quick discussion between domfarolino@ and talp@. The idea is that the AgentSchedulingGroupHostUserData class would hold a map like so:

```
{SiteInstance* ⇒ scoped_refptr<AgentSchedulingGroupHost>}
```

Since we may have multiple SiteInstances referencing the same AgentSchedulingGroupHost, this allows us to populate a single UserData data structure with as many SiteIntsance/AgentSchedulingGroupHost pairs as come up. However, after conversation with kouhei@ about the shutdown flow, domfarolino@ is convinced that this is not the right design to go with, at least at this time.

It attempts to solve two problems at once:

1. The entire shutdown flow, by tying the lifetime of AgentSchedulingGroupHost to SiteInstance
2. The ability to support AgentSchedulingGroup-per-SiteInstance

For problem (1), after discussing with kouehi@, it seems reasonable to try and tie the lifetime of AgentSchedulingGroupHost with the frames and R*Host objects that reference it via IPC::Listener routes. Given this, it makes sense to solve the two problems separately.