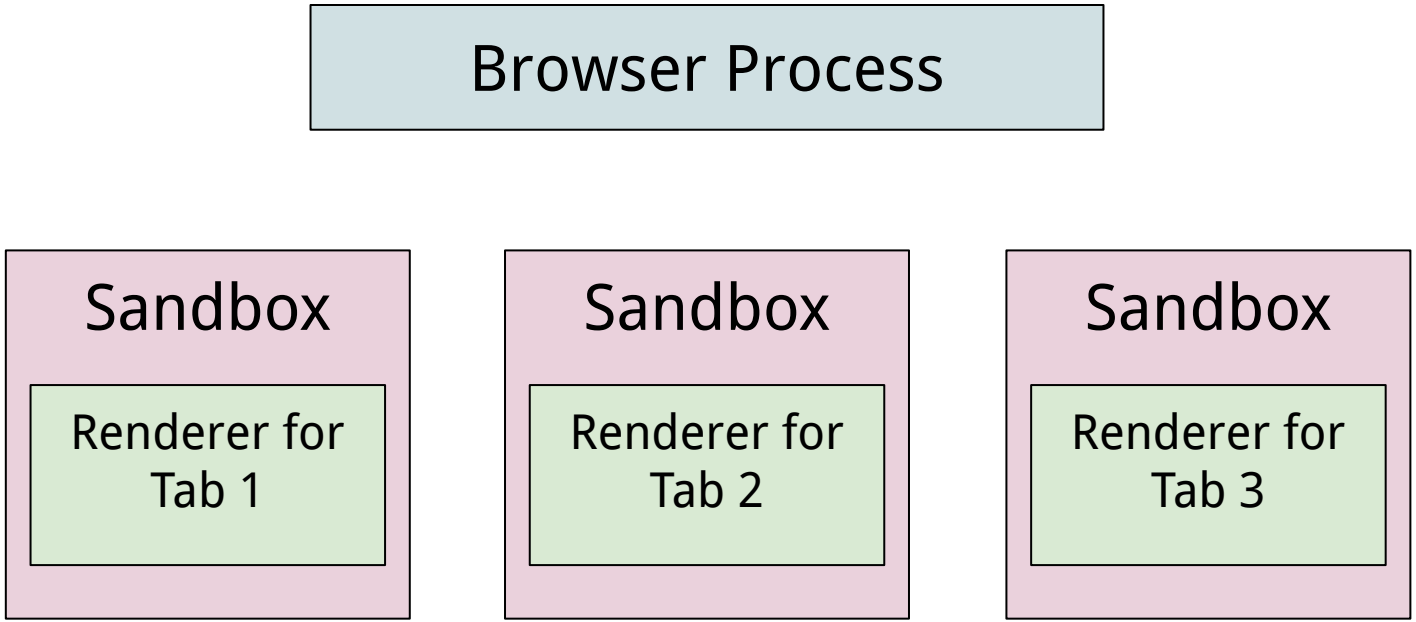
Site Isolation

Out Of Process Iframes

Chromium Today



https://crbug.com/351787

```
var ab = new ArrayBuffer(4);
ab.__defineGetter__("byteLength",
function() {
  return 0xFFFFFFFC;
});
var aaa = new Uint32Array(ab);
```

Is the sandbox sufficient?

- The renderer makes security decisions
- Cookies
- Geolocation permissions
- X-Frame-Options

Our Goals

- Site isolation with frame granularity
- Remove legacy code for swapping out `content::RenderView`
- Enforce security decisions for web content in the browser

Original Architecture

WebFrameClient

WebFrame

Frame

DOMWindow

Document

New Architecture

```
WebFrameClient WebRemoteFrameClient
 WebLocalFrame WebRemoteFrame
    LocalFrame RemoteFrame
LocalDOMWindow RemoteDOMWindow
      Document
```

New Architecture

- Frame tree holds `Frames`
- Frames have `FrameOwners`, not `HTMLFrameOwnerElements`
- Cross-origin operations are exposed by `Frame/FrameOwner/DOMWindow` interfaces.
- Most code shouldn’t need to downcast to `Local/Remote` types.

Remote vs Cross-Origin

- Remote implies cross-origin.
- Local does not imply same-origin.

Why not CrossOriginDOMWindow?

- Site isolation isn’t actually cross-origin
- Many pages have lots of frames
  - sourceforge.net: 57 frames
  - 163.com: 49 frames
- “Cross origin” is relative

Since BlinkOn 2...

- Basic frame swap
- Frame tree replication
- `content::RenderFrameProxy`

Demo!

Current Work

- Security context replication
- `WebWidget` refactoring
- `RemoteDOMWindow`
- Transition from remote to local frames.

Security context replication

- Security decisions can involve other frames
  - Navigation
  - postMessage()
- Not just origins; sandboxed iframe attribute
- Currently unhandled; most security decisions just silently fail for `RemoteFrames`

Security context replication

- Browser replicates security information
- Add `SecurityContext` accessor to `Frame`
- `LocalFrame` simply returns `Document`
- `RemoteFrame` has its own `SecurityContext`
  - populated with replicated data
  - browser maintains canonical information

Security context replication

- `frame->document()->canDo(...)` becomes `frame->securityContext()->canDo(...)`
- CSP won't be replicated
- Caution needed to prevent time of check to time of use bugs

Security context replication

- Time of check to time of use issues?
- Renderer might send an IPC assuming the target renderer has one origin, e.g. `postMessage()`
- Renderer might send an IPC to the browser for a frame that’s already swapped to another process

WebWidget refactoring

- Entry point for input event dispatch
- Viewport size
- Provides drawing surface for layout
- Sends frames to compositor

WebWidget refactoring

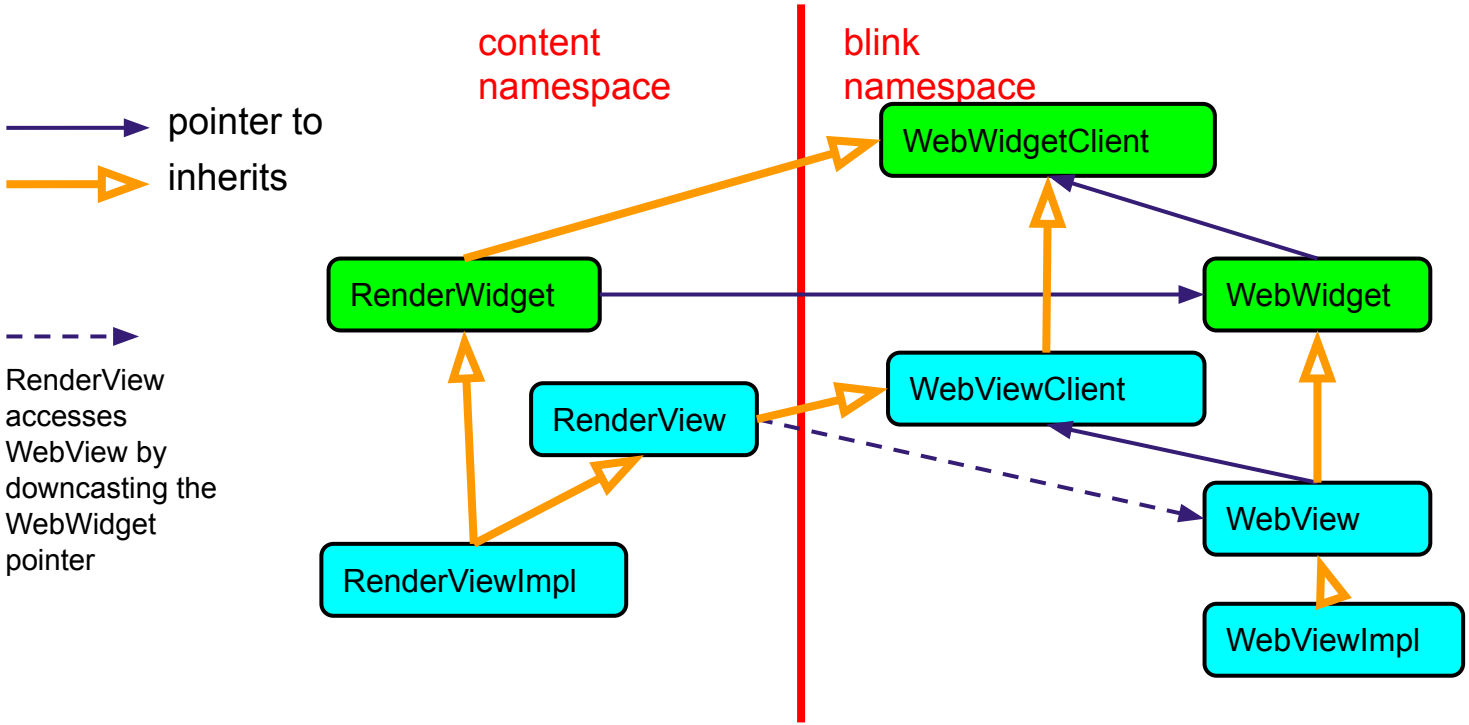
- Each subframe renderer pretends it’s a top level renderer.
- Works as long as you hold it the right way
- Breaks down with two subframes from the same site

WebWidget refactoring

- Current code just picks the first local frame it finds for rendering purposes.
- In the future, rendering and input will be based off “local root”
- Each local root will have a `WebWidget`

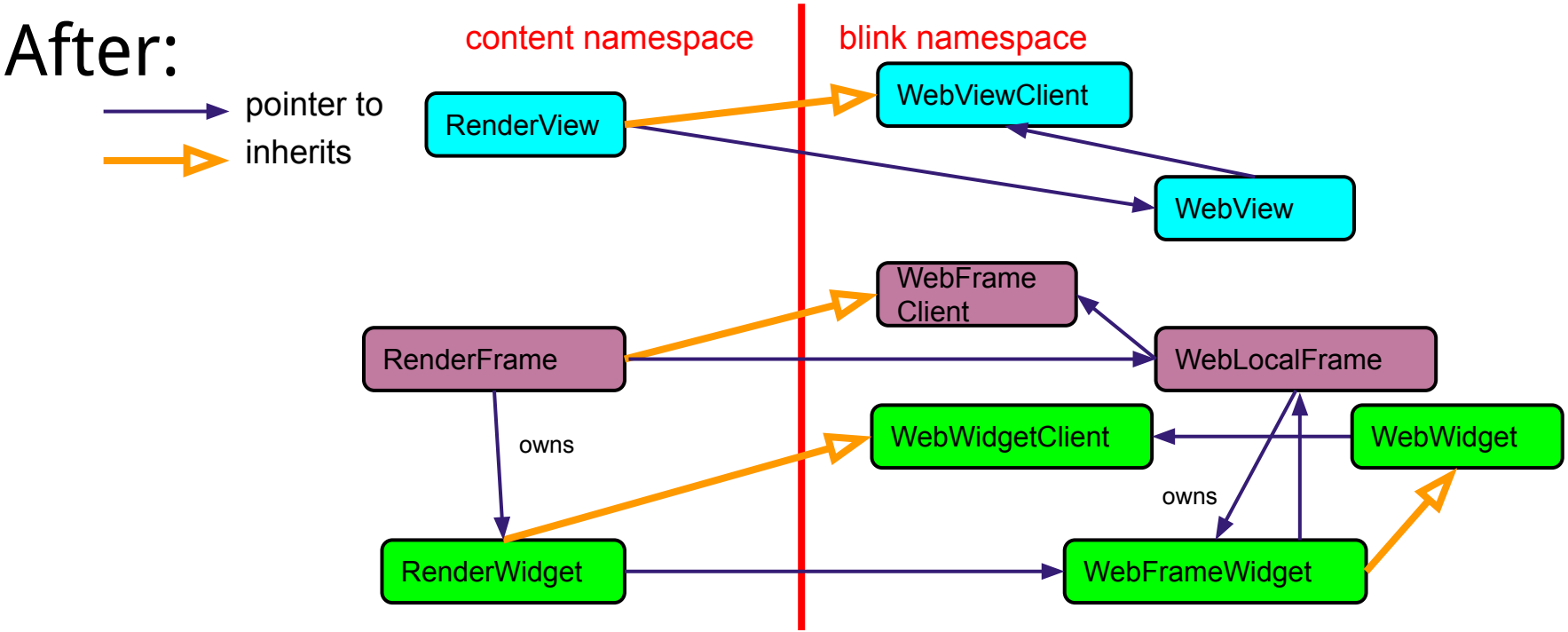
WebWidget refactoring

Before:



WebWidget refactoring

After:



DOMWindow

- Original `DOMWindow` renamed to `LocalDOMWindow`
- `RemoteFrame` currently has no `DOMWindow`
- `DOMWindow` interface factored out for bindings

WindowProxy

- For security, each `Document` has its own associated `Window` object.
- Hidden from the web by using `WindowProxy` to redirect to the `Window` object for the active `Document`.
- Managed by custom `WindowProxy` bindings

WindowProxy

```
var w = window.frames[0];
w.location = "https://example.com/";
window.setTimeout(function () {
  w.location =
      "https://foo.example.com";
});
```

Current solution: swappedout://

- Hacky: navigates to a blank page and pretends it’s not there.
- Fragile: requires IPC filtering to ignore swapped out frames.
- Dangerous: security bugs from `RenderFrameHost` confusion in browser.

Frame swap

- Implemented in `WebFrame::swap()`
- Magic that understands how to transfer relevant state from one `WebFrame` to another
  - `FrameOwner`
  - Child, parent, opener, opened frames.
  - Eventually, `DOMWindow` handoff.

Frame swap

- Cross-process transitions are hard.
- Currently fixing remote to local transitions.
- Tricky because of provisional navigations.

What is a provisional frame...?

- A renderer-initiated navigation might not actually result in a navigation.
- Plz Navigate will move navigation logic to browser process.

Anti-patterns

- Assuming the main frame is a local frame
- Checking if a `WebFrame/Frame` is local
- Checking if a `DOMWindow` is local
- Saving page state on the main frame
- Adding `ASSERT_NOT_REACHED()` methods to `WebRemoteFrameImpl`

What’s Next?

- Measuring performance
- Coordinating unload
- Input events
- Better testing framework
- Process per pixel

What’s Next

- Need your help to update Blink features
- Mailing list: site-isolation-dev@chromium.org

Questions?

This slide intentionally left blank.

Security context replication

- `Frame` will get an accessor to expose its `SecurityContext`
  - `LocalFrame` will redirect to Document
  - `RemoteFrame` will have its own security context initialized with replicated data
    - browser process will maintain origin info on its frame tree nodes and send it along whenever it is creating a new `RemoteFrame`
    - if `LocalFrame` alters its origin (e.g., when setting `document.domain`), it will need to notify the browser process

- `frame->document()->securityOrigin()->*()` will become `frame->securityContext()->securityOrigin()->*()`