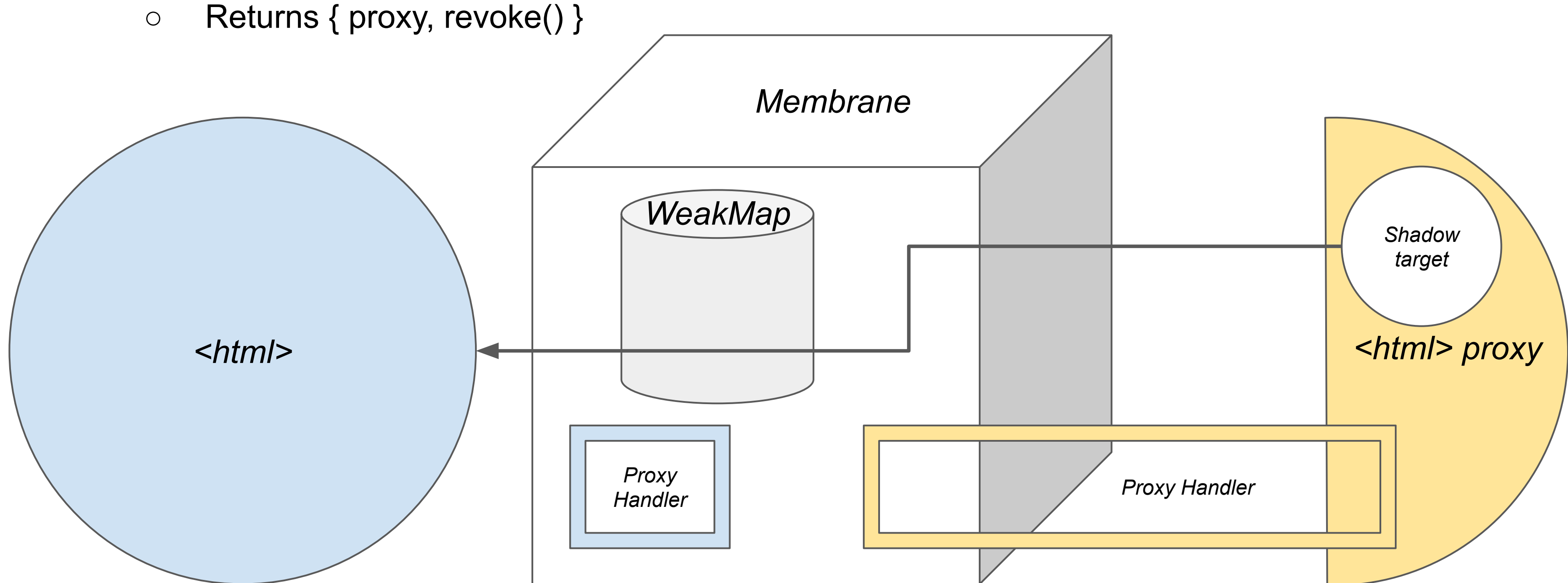
# *Mass Proxy Revocation*

Alexander J. Vincent
ajvincent@gmail.com

https://github.com/ajvincent/es-membrane

# *Proxy creation takes two arguments.*

- new Proxy(shadowTarget, proxyHandler);
- Proxy.revocable(shadowTarget, proxyHandler);
  - Returns { proxy, revoke() }



# *Background: Membranes combine proxies, maps.*

This is fine for individual proxies, but doesn’t scale well.

If you have hundreds of proxies, you have hundreds of revoker functions, or you don’t care about revoking proxies at all.

- We can reasonably expect membranes to meet this criterion.
  - Revocation of entire object graphs is a security measure: denying access.
  - Revocation means invoking every revoker function for an object graph.
- Membranes are in use *now*.
  - Mozilla Firefox has a membrane, via “cross-compartment wrappers”, for its DOM.
  - Salesforce’s observable-membrane, observing interactions in object graphs.

# What is a membrane?

```
<html>
<head>
 <title>Hello</title>
 <script src="hello.js"></script>
</head>
<body>
 <p>
  <button onclick="doSomething();">Hi</button>
 </p>
</body>
</html>
```

```
function doSomething() {
  const button = document.getElementsByTagName(
    "button"
  )[0];
  button.disabled = true;

  const p = document.createElement("p");
  p.appendChild("Good day!");
  document.body.appendChild(p);
}
```

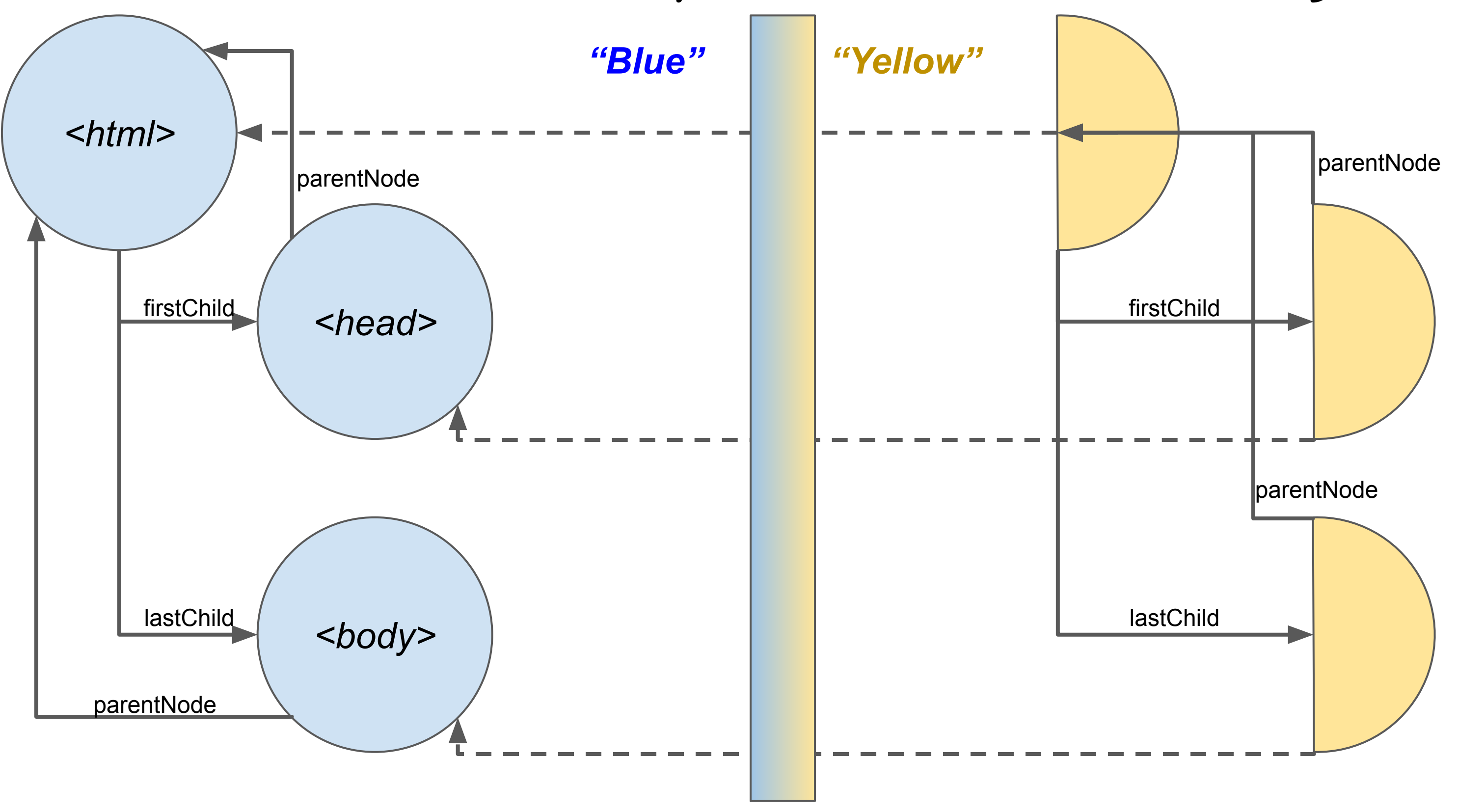
**Web browsers implement and trust the elements. The elements are "native", living inside the browser's trusted code, and are objects.**

**JavaScript code is *foreign* to the browser, and the browser treats that code with suspicion.**

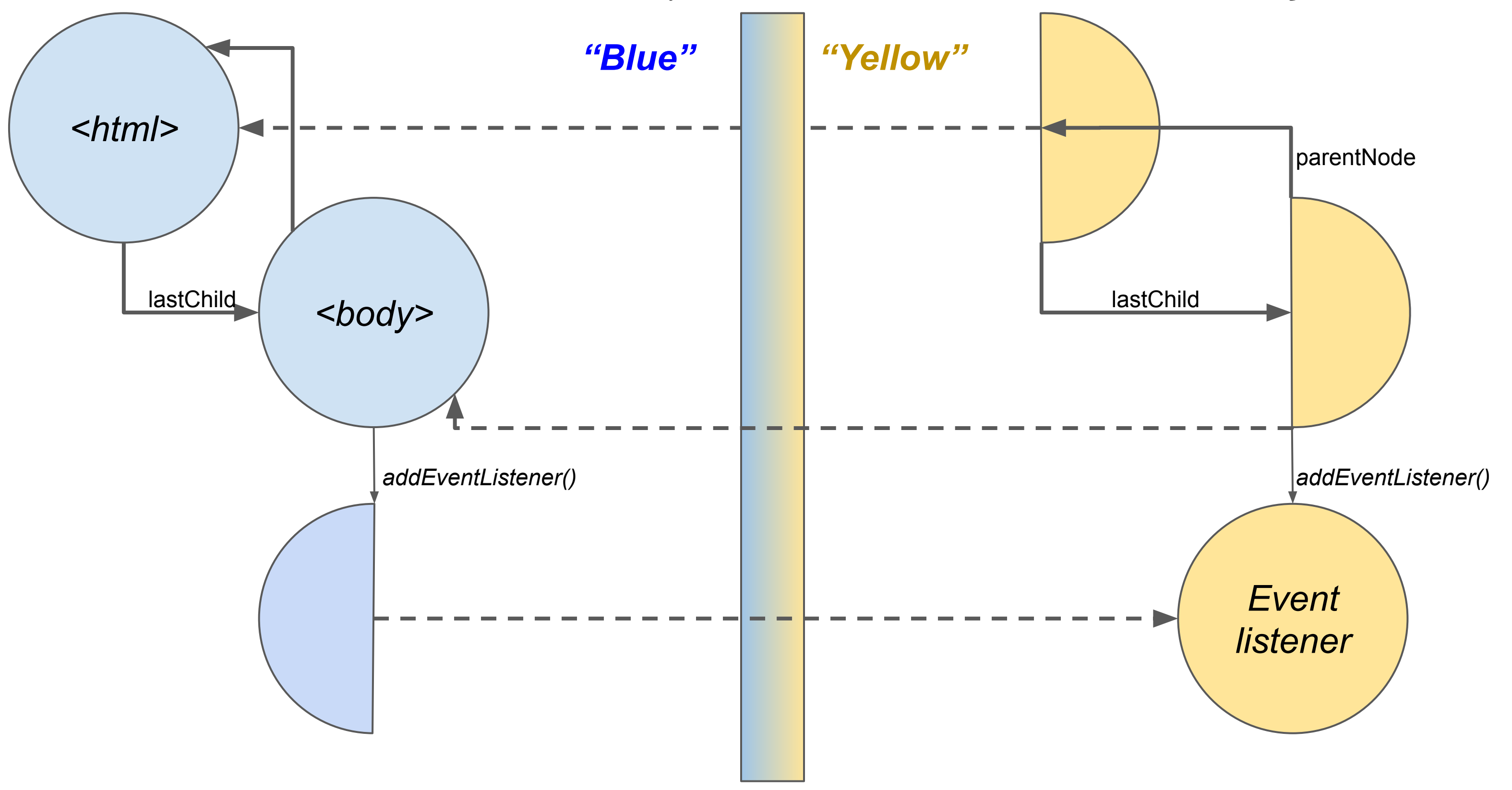
**The browser provides some access to the objects, but it doesn't dare provide unrestricted access.**

*A membrane separates "mutually suspicious" object graphs from each other, using proxies to tunnel through.*

# Membranes interpose between object graphs.



# Membranes interpose between object graphs.



# *Why are membranes important?*

- ❏ Security against accessing API which an object graph shouldn’t expose.
- ❏ Limiting impacts on objects you don’t control.
- ❏ Integrating components from different sources.

# *Membranes motivated Proxy and WeakMap.*

- A Document Object Model often has hundreds, if not thousands, of nodes: elements, and text mostly, but they're *everywhere*.

- Blocking access to internal API requires *one proxy per underlying object.*
    - Thus, one *revoker* function per proxy.
    - We now have "hyper-membrane" models, with multiple object graphs and different capabilities per graph, which I'll explain shortly. This means *one proxy per underlying object, per graph*, and one revoker function *per proxy.*

- Other complex data structures may have similar requirements.

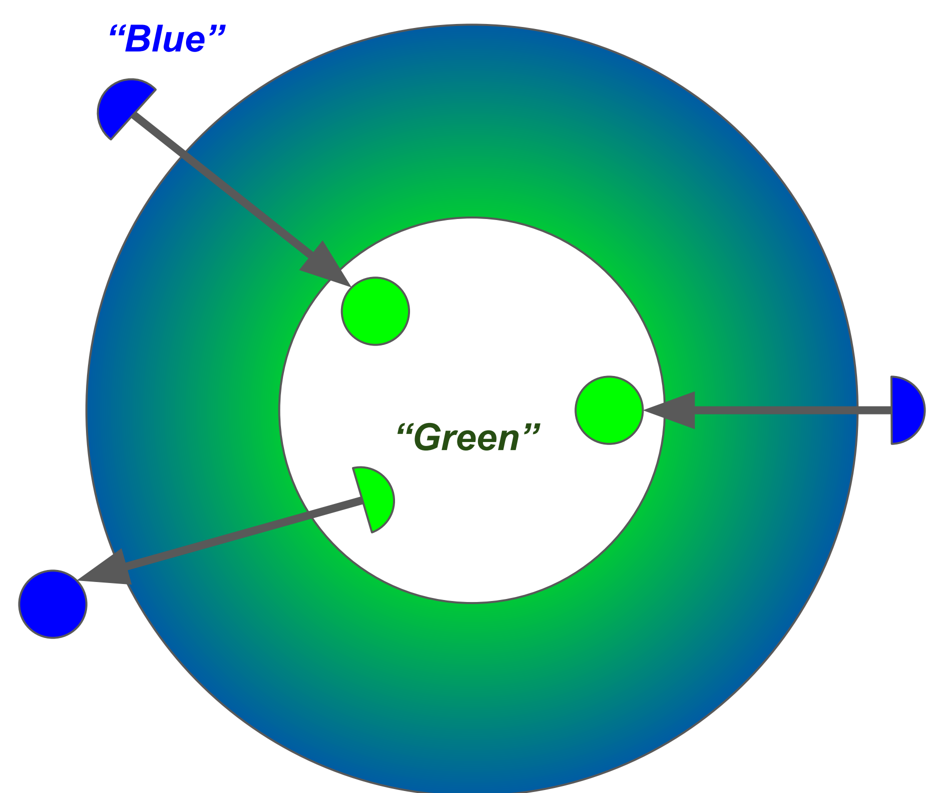
# *Membranes motivated [Proxy]() and [WeakMap]().*

Now, imagine revoking *hundreds* or *thousands* of proxies synchronously, with one revoker function for every proxy. That's where we are *right now* in membranes.

- We can probably fudge a little with custom proxy handlers, but we still need to *create* the revokers, and *hold* them somewhere, some way.
  - A revoker holds a reference to the proxy in an internal slot.
  - The proxy holds a strong reference to the shadow target.
  - Think WeakMap<shadowTarget, revoker>. This would allow revocation during trap invocation, *if* we emulate the traps for revoked proxies in our proxy handler.

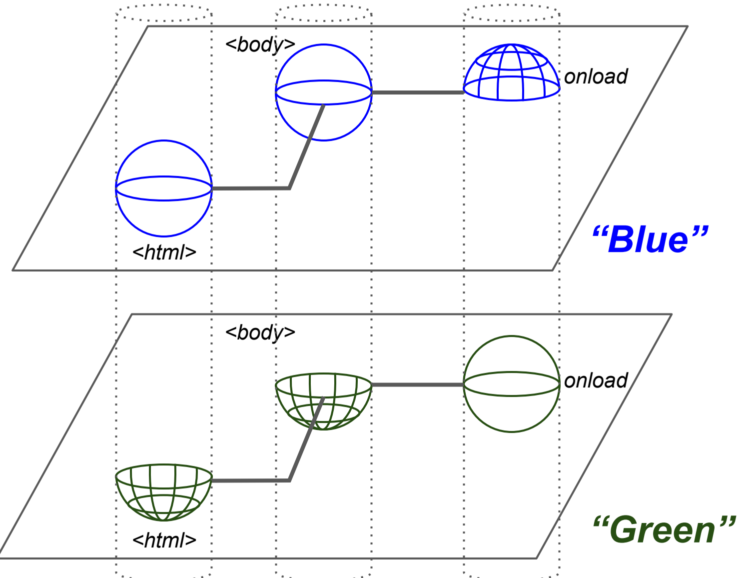
- Did I mention membranes are *really hard* to implement correctly?
  - This is why abstraction libraries like observable-membrane and es-membrane exist.

# *Membranes:  The cell model*



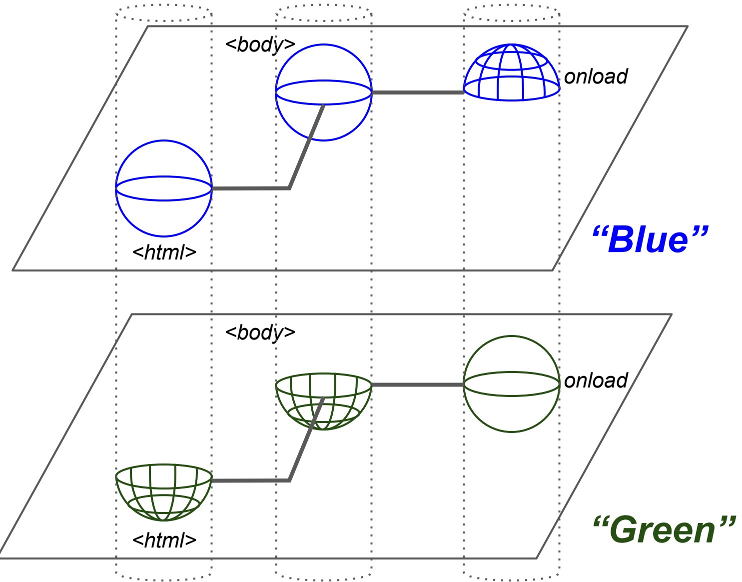

- The first model of a membrane was based around *cell membranes* in biology:  an impenetrable (except by Proxy) boundary to separate what’s inside the boundary (here, “green”) from what’s outside (“blue”).
- Understanding biology isn’t necessary. This is just to illustrate the model Dr. Mark Miller and Dr. Tom van Cutsem used for their first membranes.
- The original concept was one-to-one mappings, with a maximum of two sides.
- We represent objects by circles, proxies by semicircles.

# Membranes: The geometric model



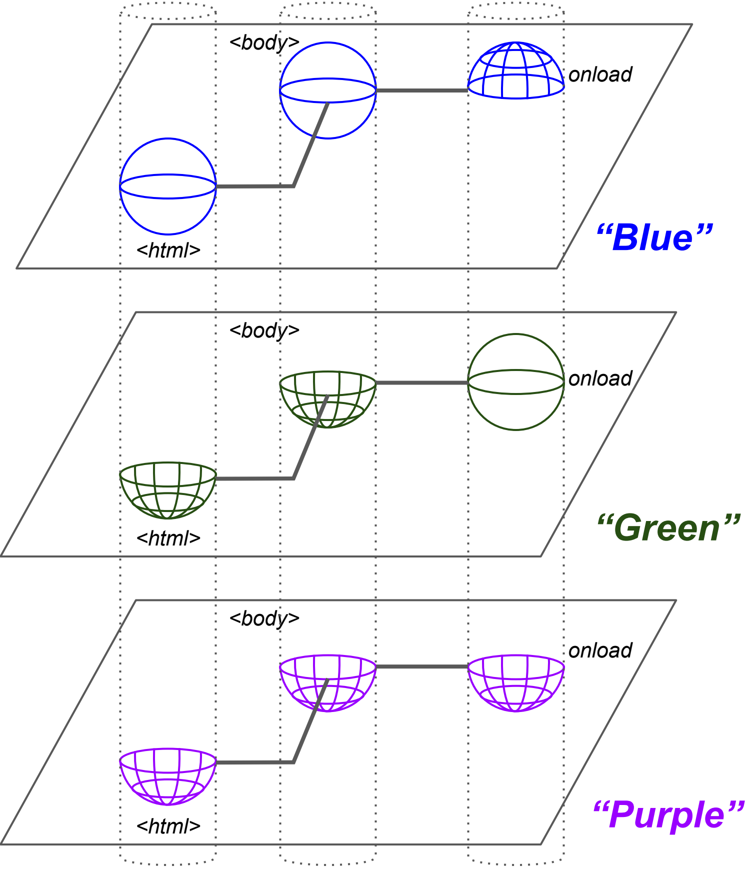

- Each object graph is in a plane, and each object graph plane is parallel to the other plane.
- We now describe objects by spheres instead of circles, embedding their centers in each plane.
- We describe proxies by hemispheres instead of semicircles, embedding their flat edges in each plane.
- Cylinders perpendicular to the object graph planes connect each proxy to its underlying object, showing the relations.

# Membranes: The geometric model



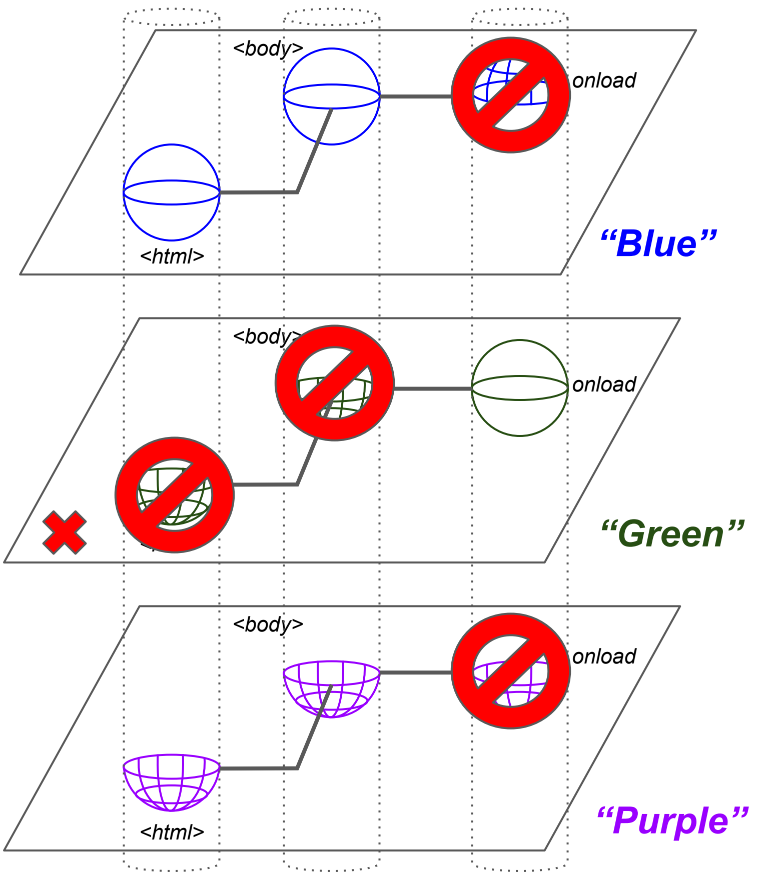

- Physical distance has no meaning within each object graph plane and between planes, except that spheres and hemispheres shall not intersect.
- This model is equivalent to the cell membrane model, with two differences:
    - The idea of "inside versus outside" goes away.
    - There's plenty of space for more object graph planes...

# *Membranes: The geometric model*




- Now we can move from two object graphs to *n* object graphs, as many as we want.
- Graph planes are unordered, swappable.
- Technically, by creating links between object graphs, we’ve created a type of *hypergraph*.
- Mark proposed calling Alex’s invention a *hyper-membrane*, but Alex thinks *hypergraph membrane* is a better name.
- Before this naming debate (and this model), the convention was to call these “multi-sided membranes”. These are still new enough that the names are interchangeable.

# Cleaning up an object graph is why we're here.




- Imagine we want to revoke the "green" object graph. This requires:
  - Creating and holding revokers for every proxy in the "green" object graph.
  - *Executing* all those revokers. Synchronously.
  - Revoking the "onload" proxies in the "blue" and "purple" object graphs. (Including overhead to track these revokers and tie them to the "green" graph.)
- In the ECMAScript spec, each revoker is a function to clear two slots: the target and the handler. *That's it.*

# *Proposal: Add an options argument to proxies.*

- new Proxy(shadowTarget, proxyHandler, options = {});
- Proxy.revocable(shadowTarget, proxyHandler, options = {});

The options object would initially support one property, a “revocation signal”.

```
// The finalization registry inspired this API.  Names of static methods open to change.
// A symbol, so we can pass it across realms if necessary.
const signal = Proxy.createSignal();
return new Proxy(shadowTarget, proxyHandler, { signal });
// ...
Proxy.finalizeSignal(signal); // kills the proxy.
```

- We could share the signal among many proxies.
- Proxies would have a new internal slot pointing to this signal.
  - If the signal is revoked, proxies can clear their slots and throw.
  - Garbage collection can treat proxies holding a revoked signal as dead, and clear the slots.

# *Proposal: Add an options argument to proxies.*

This would mean membranes might not need Proxy.revocable().  So we wouldn't create so many revokers in the first place.

- Less memory allocation, less garbage collection pressure.
- Hundreds of revoker functions reduce to far fewer: at most one for each element of the power set of object graphs.
  - "blue" + "green", not "blue" + "green" + "purple" unless a proxy has a dependency on more than two graphs

# *Cross-cutting concern: Cancellation Proposal*

- The exact *shape* of the “revocation signal” API is flexible at this point.

- The “revocation signal” could be a cancellation API, presuming the latter moves forward.

# *Mass Proxy Revocation*

Thank you.  Stage 1?

*Alexander J. Vincent*
*ajvincent@gmail.com*

*https://github.com/ajvincent/es-membrane*