Ephemeron Tables
for EcmaScript Harmony and Secure EcmaScript

Mark S. Miller and the Cajadores
Google, Inc.

Shorthand: Const functions

```
{
    ...
    const foo(a) { ...; }
    ...
}
```

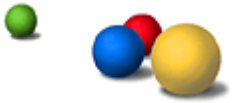
shorthand for

```
{
    const foo = function(a) { ...; };    // unitialized foo unobservable
    Object.freeze(foo);                  // mutable foo unobservable
    Object.freeze(foo.prototype);        // mutable foo.prototype unobservable
    ...
    ...
}
```

Soft Fields, Rights Amplification

```
// The "factorial" of secure programming

const makeCurrency() {
  const decr = makeEphemeronTable();
  return const makePurse(balance :Nat) {
    const purse = Object.freeze({
      getBalance: const() { return balance; },
      makePurse: const() { return makePurse(0); },
      depositFrom: const(amount :Nat, src) {
        const newBal :Nat = balance + amount;
        decr.get(src)(amount);
        balance = newBal;
    }});
    decr.set(purse, const(withdrawal) { balance -= withdrawal; });
    return purse; };}
```



Soft Fields with Inheritance

```
const makeSoftField() {
  const et = makeEphemeronTable();
  return Object.freeze({
    get: const(key) {
      while (key !== null) {
        const result = et.get(key);
        if (result !== undefined) { return result; }
        key = Object.getPrototypeOf(key);
      }
      return undefined;
    },
    set: et.set;
  });
}
```

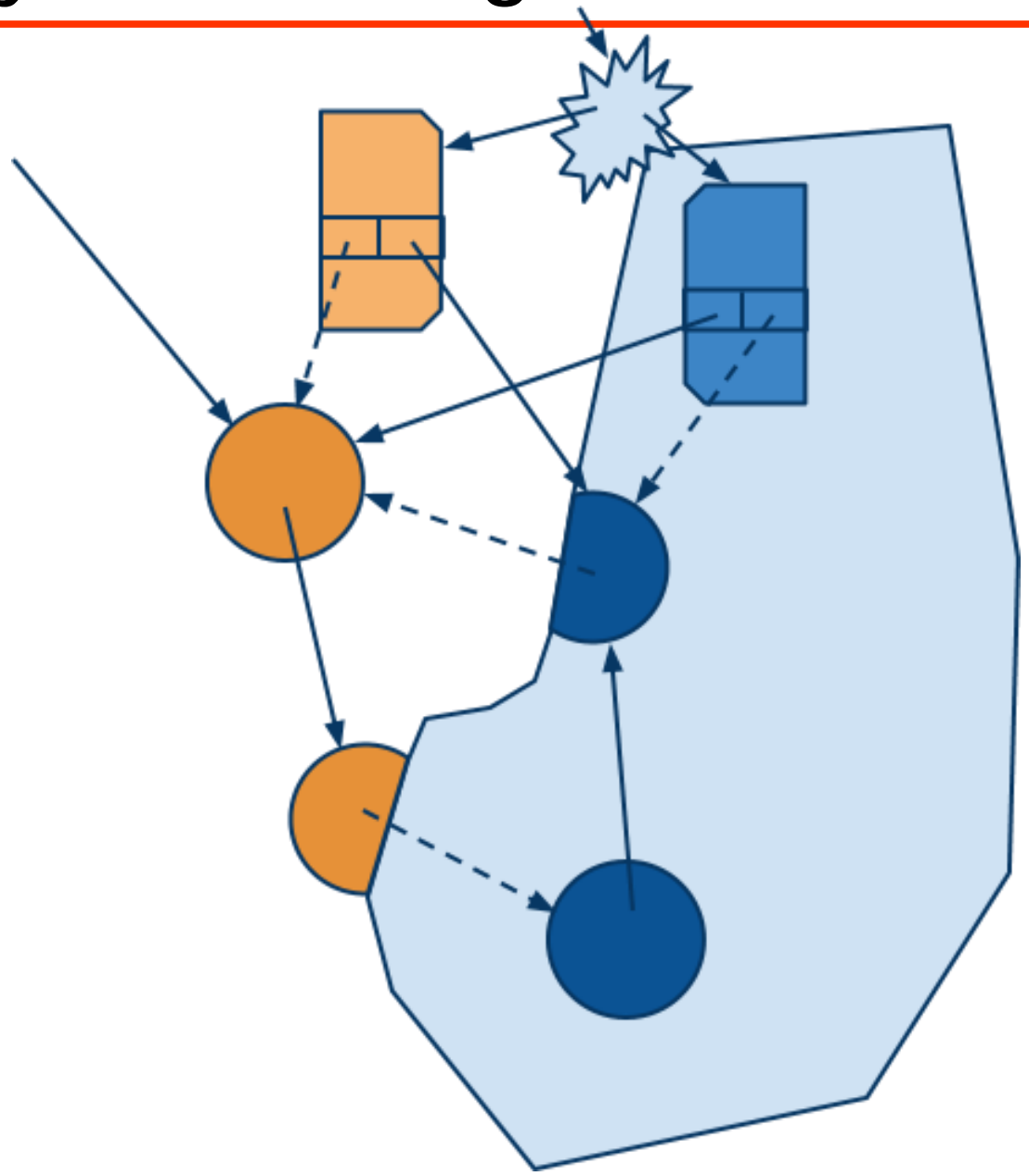
Unique Labeller

```
const makeLabeler() {
  const et = makeEphemeronTable();
  let count = 0;
  return Object.freeze({
    label: const(obj) {
      const result = et.get(obj);
      if (result) { return result; }
      et.set(obj, ++count);
      return count;
    }
  });
}
```

Cycle Tolerant Graph Walking

```
const allPropNames(root) {
  const props = Object.create(null);
  const seen = makeEphemeronTable();
  const recur(node) {
    if (node !== Object(node)) { return; }
    if (seen.get(node)) { return; }
    seen.set(node, true);
    recur(Object.getPrototypeOf(node));
    Object.getOwnPropertyNames(node).forEach(const(key) {
      props[key] = true;
      recur(node[key]); });}
  recur(root);
  return Object.keys(props);
}
```

Identity Preserving Membranes

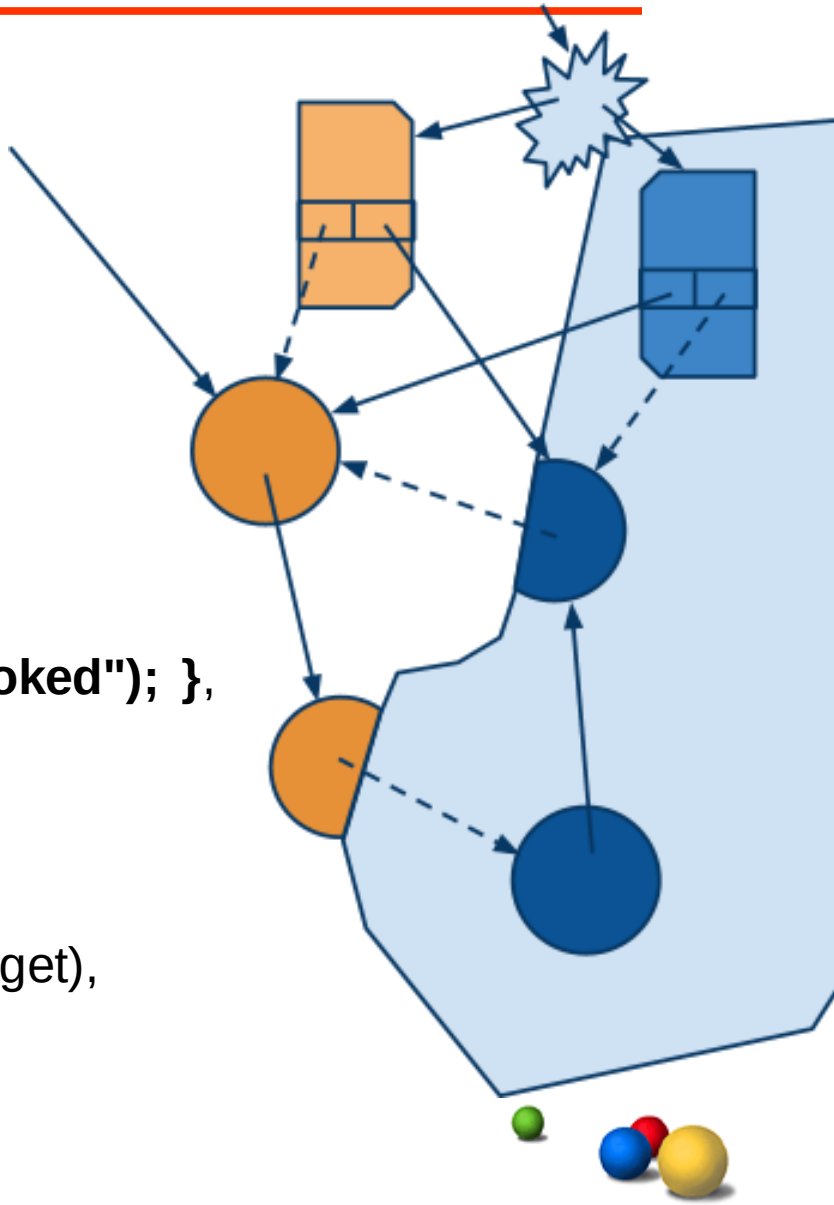


Better Membrane (Overview)

```
const makeMembrane(wetTarget) {
  let wet2dry = makeEphemeronTable(true);
  let dry2wet = makeEphemeronTable(true);

  const asDry(wet) {/*...*/}
  const asWet(dry) {/*...*/}

  const gate = Object.freeze({
    revoke: const() {
      dry2wet = wet2dry = Object.freeze({
        get: const(key) { throw new Error("revoked"); },
        set: const(key, val) {}
      });
    }
  });
  return Object.freeze({ wrapper: asDry(wetTarget),
                         gate: gate });
}
```

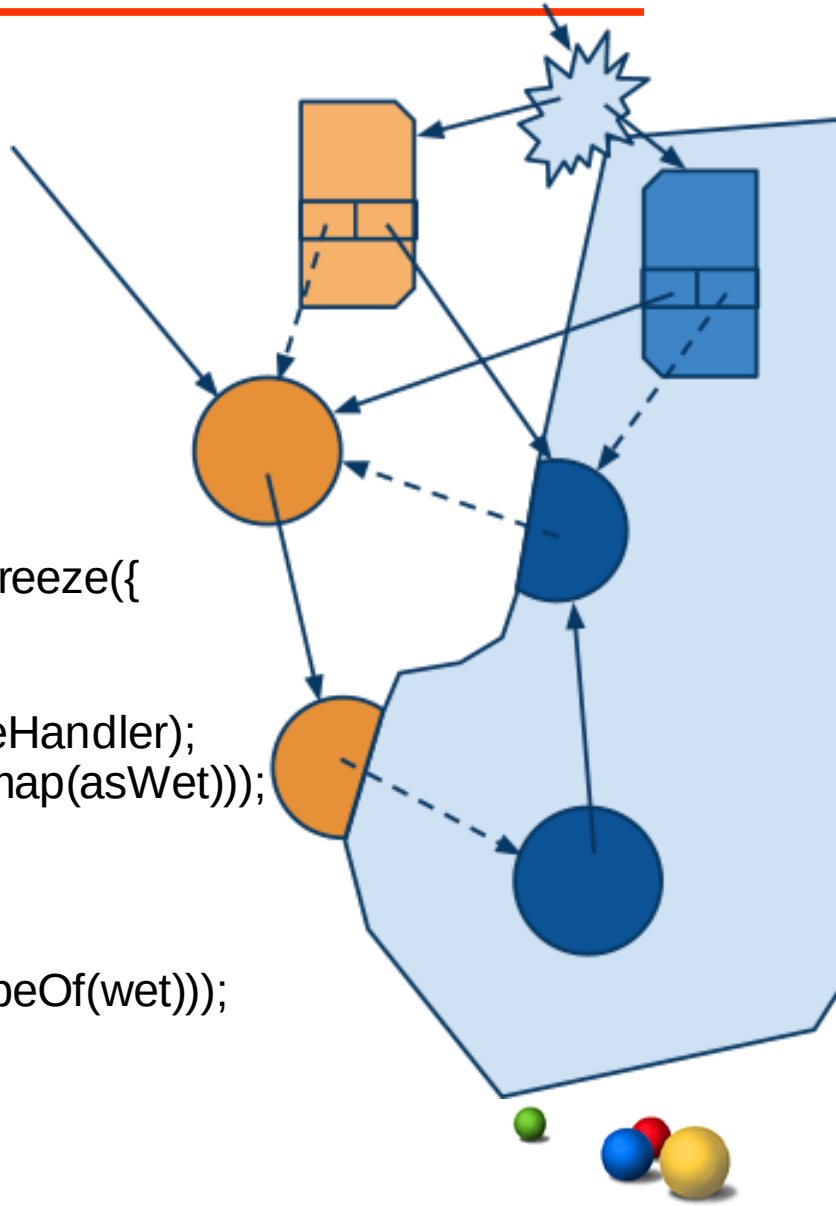


Better Membrane (Detail)

```
const makeMembrane(wetTarget) {
  let wet2dry = makeEphemeronTable(true);
  let dry2wet = makeEphemeronTable(true);

  const asDry(wet) {
    if (wet !== Object(wet)) { return wet; }
    let dryResult = wet2dry.get(wet);
    if (dryResult) { return dryResult; }

    const wetHandler = makeHandler(wet);
    const dryRevokeHandler = Proxy.create(Object.freeze({
      get: const(rcvr, name) {
        return const(...dryArgs) {
          const wetHandler2 = dry2wet.get(dryRevokeHandler);
          return asDry(wetHandler2[name](...dryArgs.map(asWet)));
      };}}));
    dry2wet.set(dryRevokeHandler, wetHandler);
    dryResult = Proxy.create(dryRevokeHandler,
                             asDry(Object.getPrototypeOf(wet)));
    wet2dry.set(wet, dryResult);
    dry2wet.set(dryResult, wet);
    return dryResult; }
....
```



Secure EcmaScript

When Alice asks: bob.foo(carol)

- Alice grants Bob access to Carol, as needed for foo

Memory-safe encapsulated objects

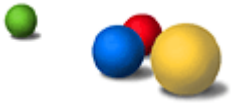
- Protect objects from their outside world

OCaps: Causality only by references

- No powerful references by default
- Protect world from objects

Reference graph === Access graph

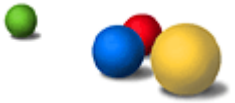
- Deny authority by witholding connectivity



Secure EcmaScript

SES ⊂ (ES5 Strict + a bit of ES-Harmony)

- Deny access to global variables, global object
- Delete non-whitelisted properties
- Freeze accessible primordials (Object, Array, Array.prototype,…)
- Restrict eval() and Function() to SES



Secure EcmaScript

SES ⊂ (ES5 Strict + a bit of ES-Harmony)

- Deny access to global variables, global object
- Delete non-whitelisted properties
- Freeze accessible primordials (Object, Array, Array.prototype,…)
- Restrict eval() and Function() to SES

SES5 trivial to implement on friendly ES5, given

```
freeVars(programSrc :string) :string[]
```

- Small init wraps eval & Function with verifier, this binder
- Uses getOwnPropertyNames() to delete all unknowns
- Freezes all whitelisted paths

Safe JavaScript Mashups Impossible?

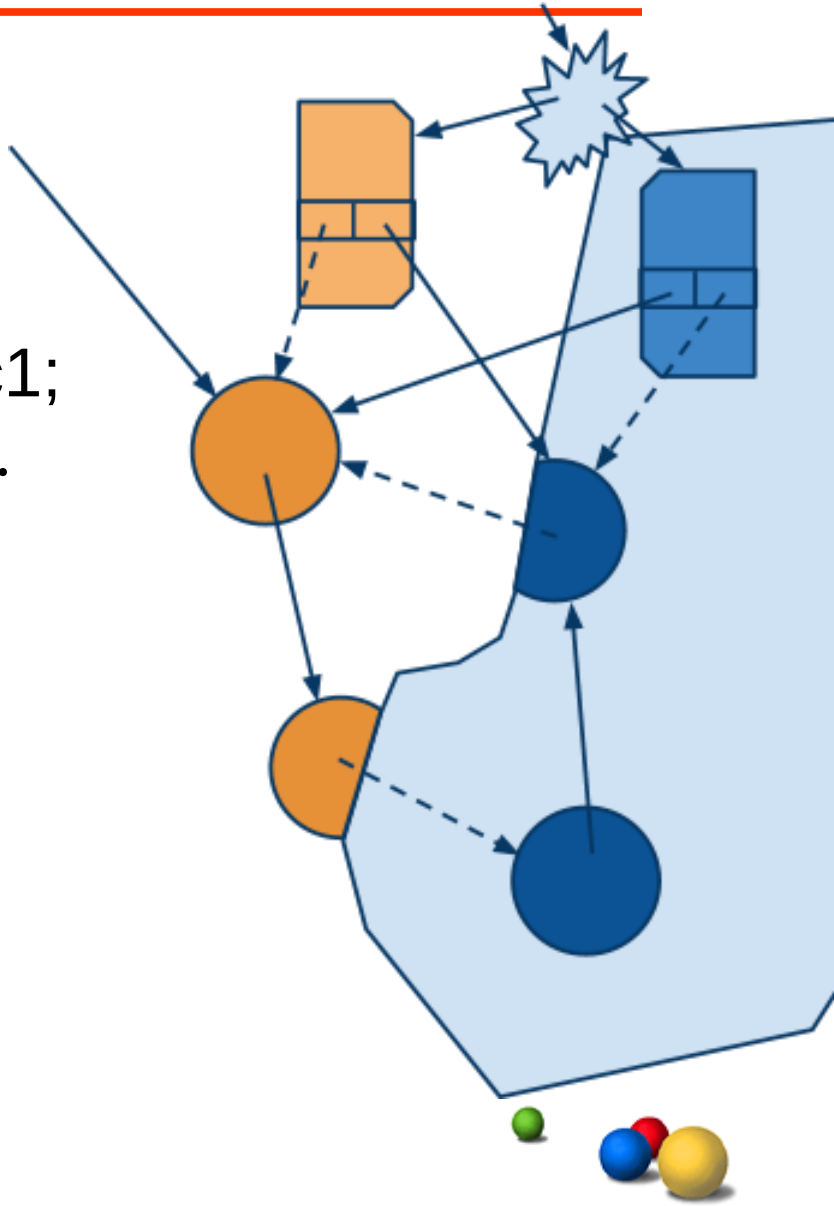
The counter example:

```
const bobPowers = Object.freeze({counter: makeCounter(0)});
const bobMakerCode = //...get potentially bad code...
const bob = eval(bobMakerCode).make(bobPowers);
```

Bob can *only* count.

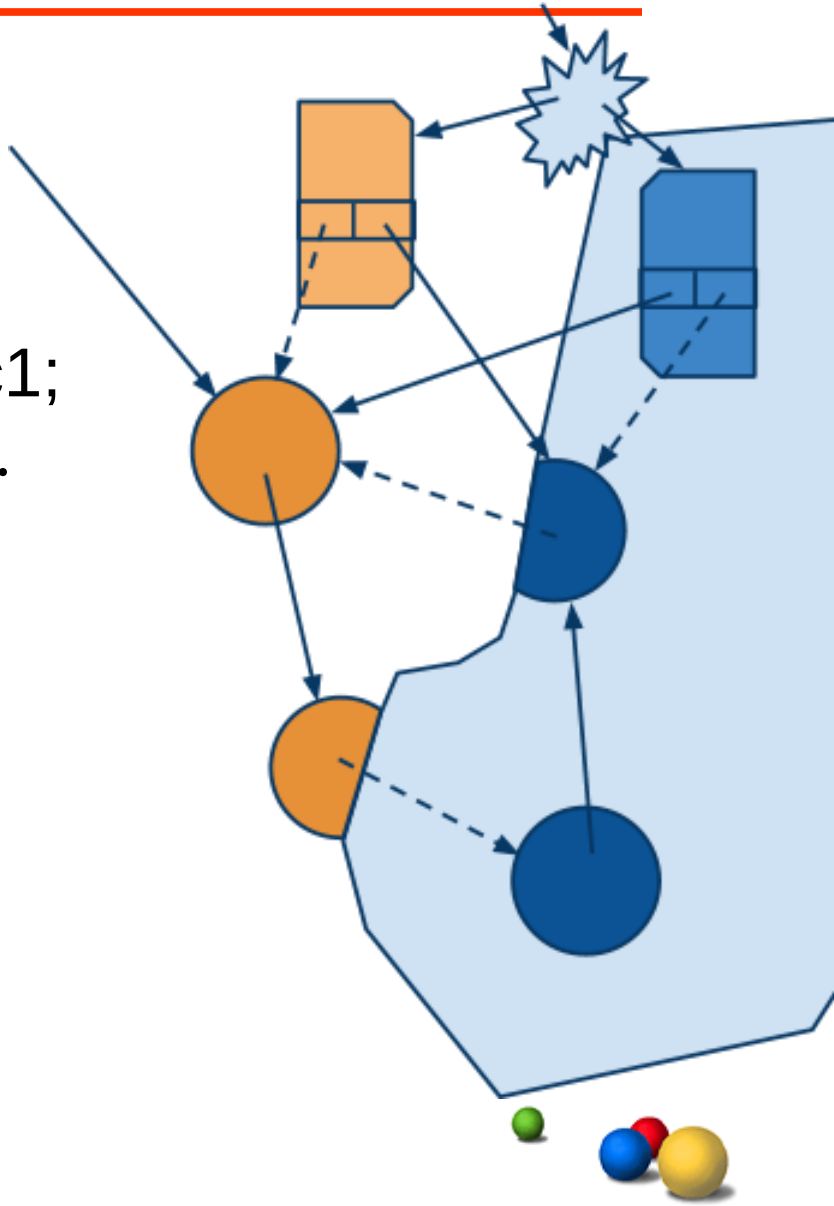
Mashup Compartments

```
const c1 = makeMembrane(eval);
const {wrapper: ev1, gate: g1} = c1;
const badCode = //...get bad code...
const result = ev1(badCode);
//...use result...
```

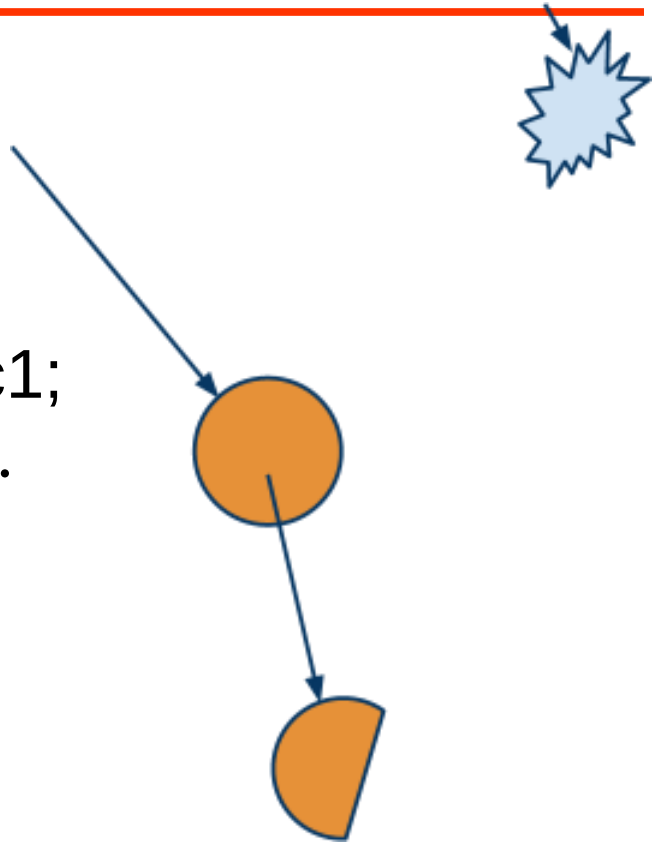


Mashup Compartments

```
const c1 = makeMembrane(eval);
const {wrapper: ev1, gate: g1} = c1;
const badCode = //...get bad code...
const result = ev1(badCode);
//...use result...
g1.revoke();
```



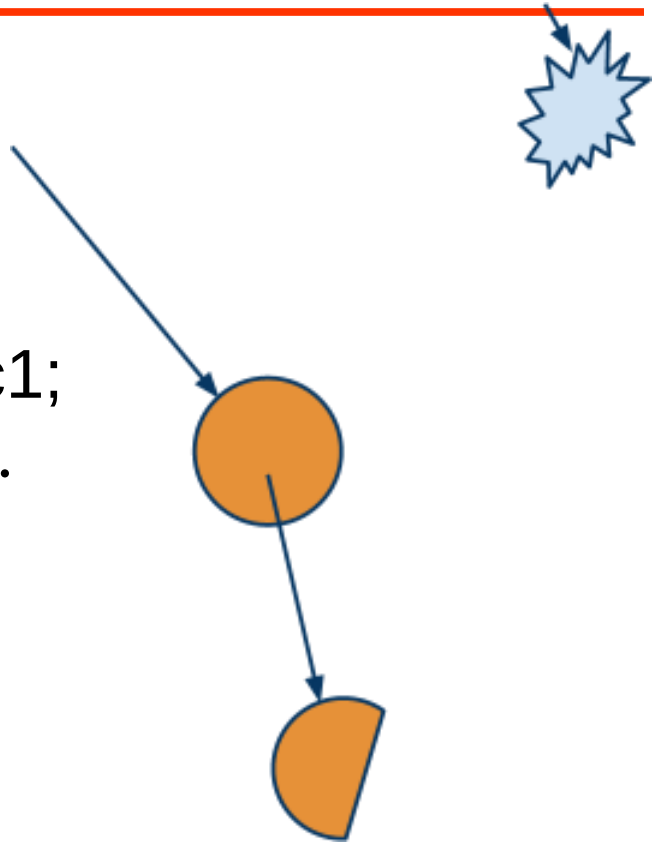
Mashup Compartments



```
const c1 = makeMembrane(eval);
const {wrapper: ev1, gate: g1} = c1;
const badCode = //...get bad code...
const result = ev1(badCode);
//...use result...
g1.revoke();
//...compartment gone...
```

Bob no longer counts.

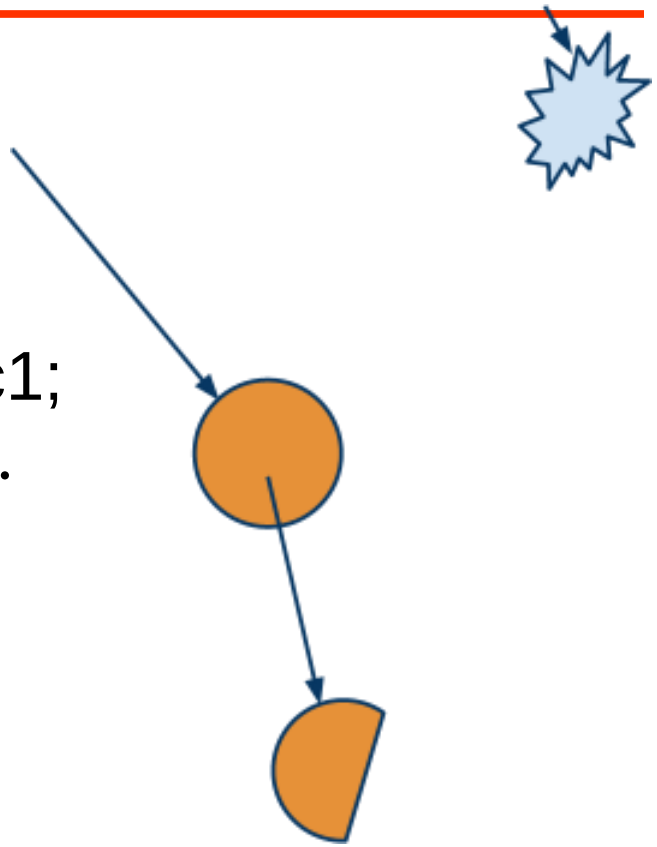
Mashup Compartments



```
const c1 = makeMembrane(eval);
const {wrapper: ev1, gate: g1} = c1;
const badCode = //...get bad code...
const result = ev1(badCode);
//...use result...
g1.revoke();
//...compartment gone...
```

SES = SES5 + Proxy + makeEphemeronTable

Mashup Compartments



```
const c1 = makeMembrane(eval);
const {wrapper: ev1, gate: g1} = c1;
const badCode = //...get bad code...
const result = ev1(badCode);
//...use result...
g1.revoke();
//...compartment gone...
```

Hopefully even smoother using modules!

Questions?