# Secure ECMAScript

Name Subject To Change

Douglas Crockford

**Yahoo!**

The purpose of this workshop is to consider the feasibility and necessity of a secure replacement for ECMAScript.

## Security is our Number One Problem

All websites are under attack.

Progress is being frustrated.

Mash Up We Must!

## Three Possible Solutions

- Safe JavaScript Subset.

  Timeframe: Immediate

- Communicating Vats.

  Timeframe: Intermediate

- Secure Programming Language.

  Timeframe: Distant

- All of the Above.

## Safe JavaScript Subset

- Constrain the existing language by code rewriting and runtime repression or by static validation.
- The constrained language limits the capabilities that are given by default to a program.
- Good examples may inform the design of Ses.
- It may be good to derive a standard, but that is not the goal of this meeting.

## Vats

- Secure containers for computation.
- Constrained intervat communication.
- First steps: Google's Gears Workerpools; durable **`<iframe>`**, XDM.
- Ultimately, transmission of capabilities, futures, distributed garbage collection.
- This is out of scope for today's meeting.

## A New Language

- Similar, but not compatible.
- Retain the goodness of ECMAScript.
- Replace, repair, or remove the bad parts.

- JavaScript got a lot right.
- Minimize retraining.
- Capture programmers, not programs.

## Goals

- A computation model that allows for cooperation under mutual suspicion.
- As simple as possible. Simple systems are easier to reason about.
- Approachable. The language must be usable by ordinary web developers.
- Unsurprising. Not freaky.
- Avoid confusion, difficulty, unmanagability.

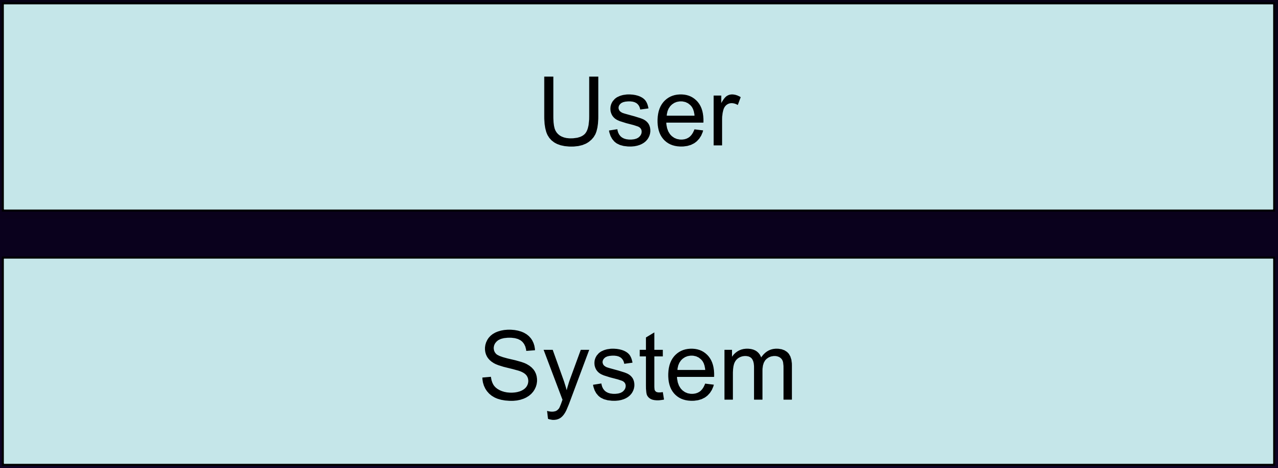
## Confusion of Interest

System Mode

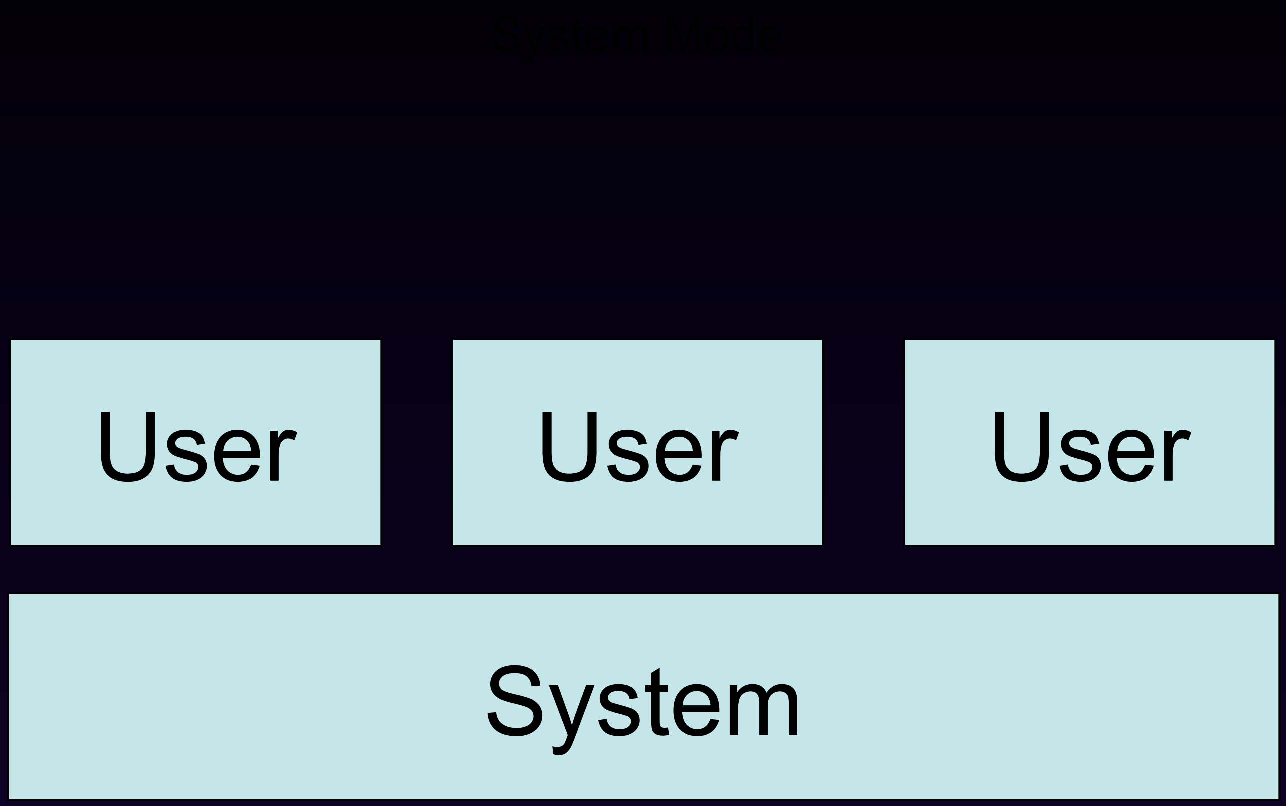
Computer

# Confusion of Interest

System Mode



## Confusion of Interest



## Confusion of Interest

System Mode

CP/M MS-DOS MacOS Windows

## Confusion of Interest

The System cannot distinguish the interest of the user from the interest of any program. This enables floppy-borne viruses.

CP/M MS-DOS MacOS Windows

## Confusion of Interest

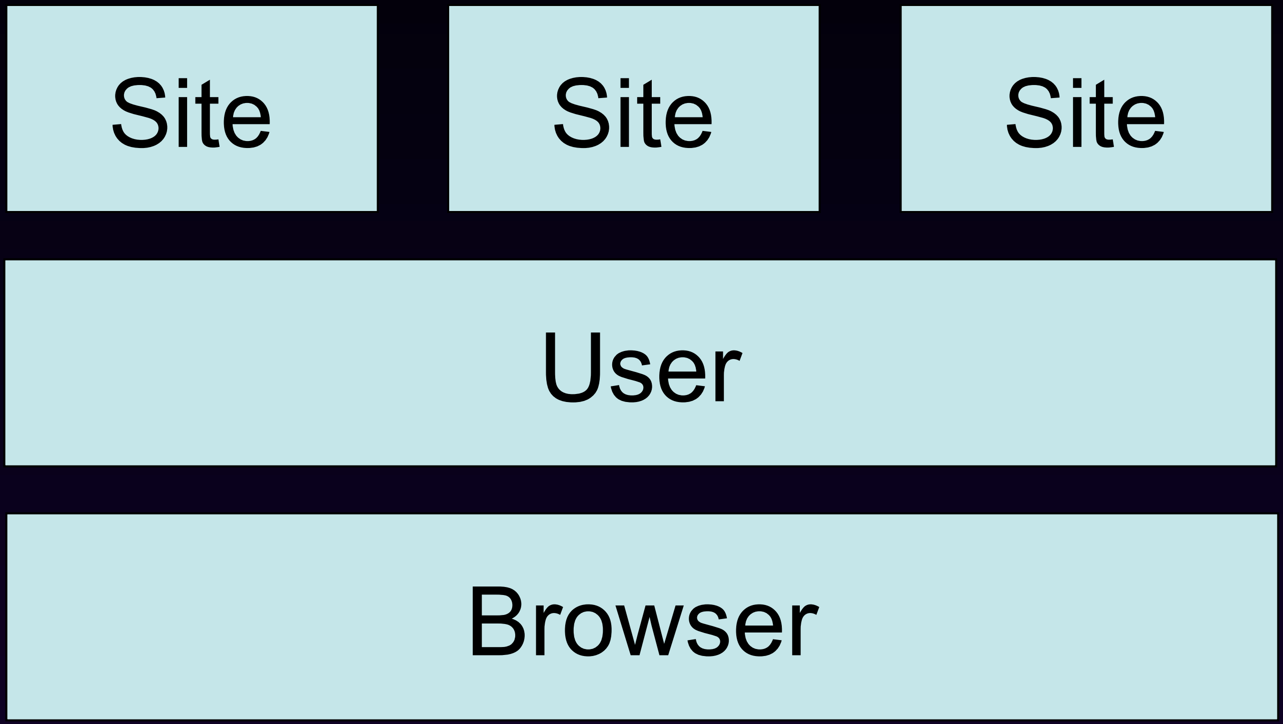
System Mode

When networking is introduced, network-borne viruses are enabled.

CP/M MS-DOS MacOS Windows

## Confusion of Interest

The browser is a significant improvement, able to distinguish the interests of users and sites in some cases.



## But within a page, interests are confused.

An ad or a widget or an Ajax library gets the same rights as the site's own scripts.

## JavaScript got close to getting it right.

Except for the Global Object.

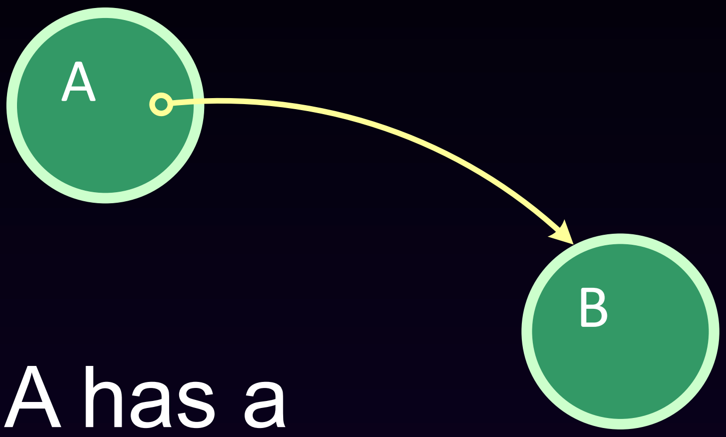
It can be repaired, becoming an object capability language.

## An Introduction to Object Capabilities

A is an Object.

Object A has state and behavior.

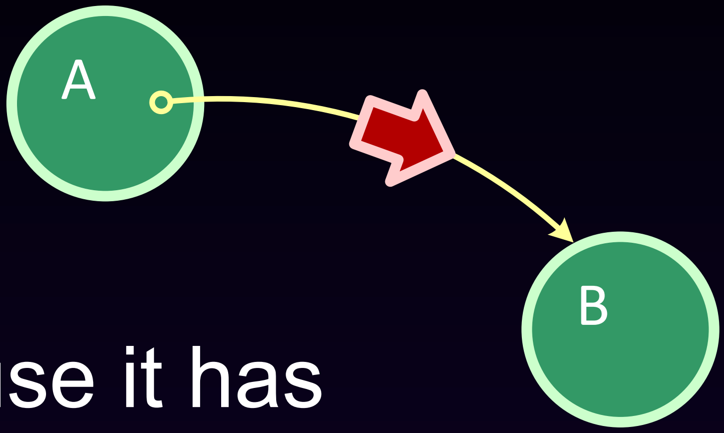
# has-a



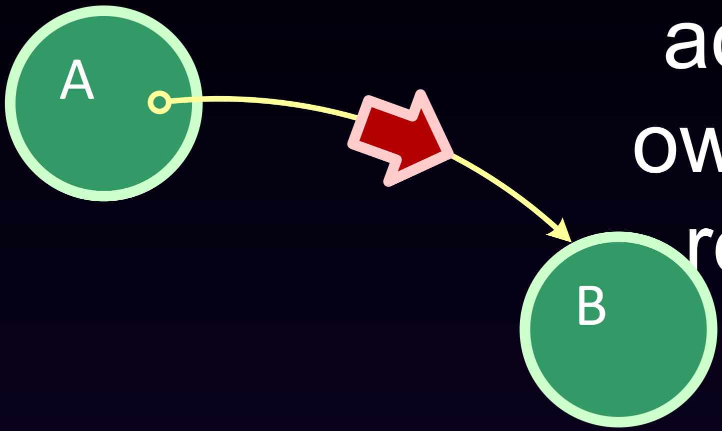
Object A has a reference to Object B.

An object can have references to other objects.

Object A can communicate with Object B...



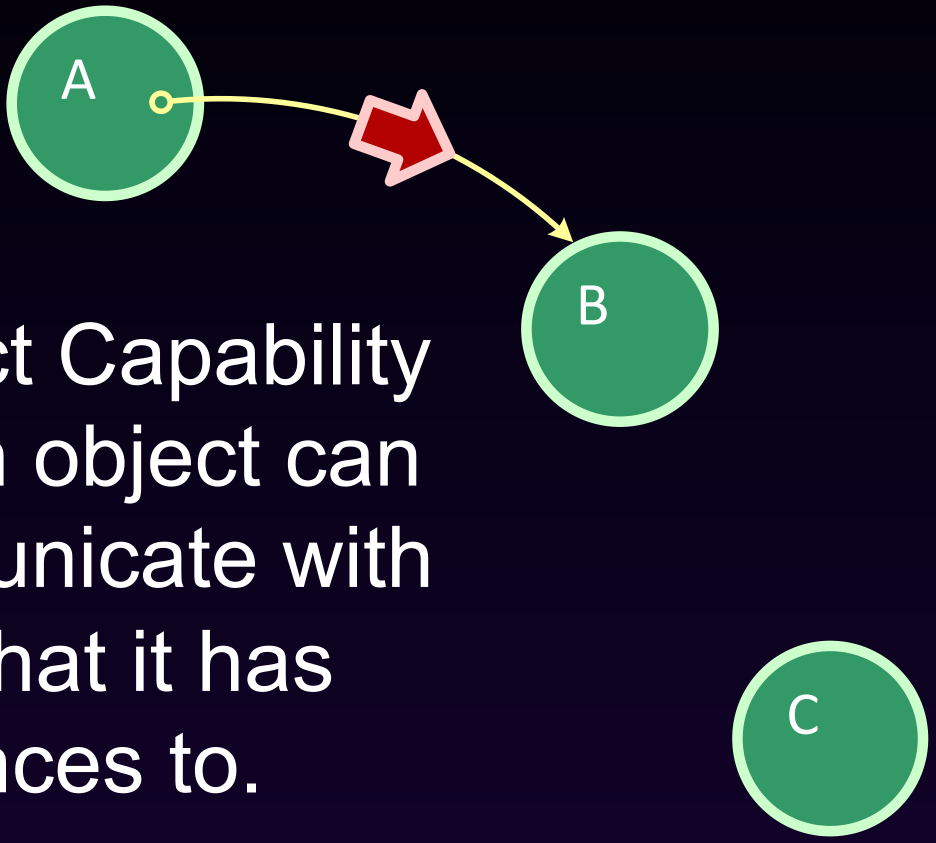
...because it has a reference to Object B.



Object B provides an interface that constrains access to its own state and references.

Object A does not get access to Object B's innards.

Object A does not have a reference to Object C, so Object A cannot communicate with Object C.



In an Object Capability System, an object can only communicate with objects that it has references to.

An Object Capability System is produced by constraining the ways that references are obtained.

A reference cannot be obtained simply by knowing the name of a global variable or a public class.

## There are exactly three ways to obtain a reference.

1. By Creation.

2. By Construction.

3. By Introduction.

## 1. By Creation

If a function creates an object, it gets a reference to that object.

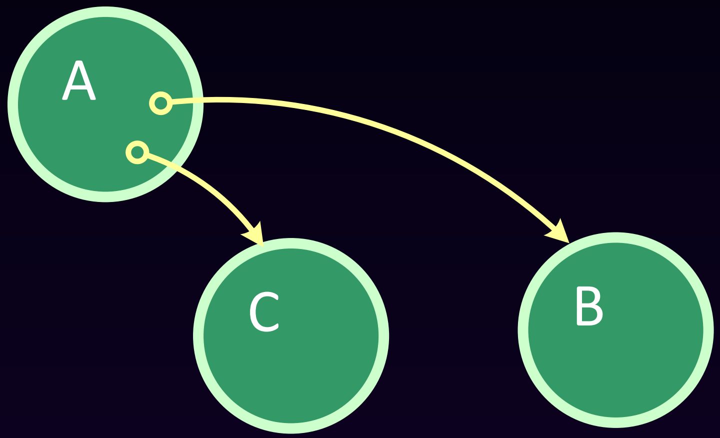
## 2. By Construction

An object may be endowed by its constructor with references.

This can include references in the constructor's context and inherited references.

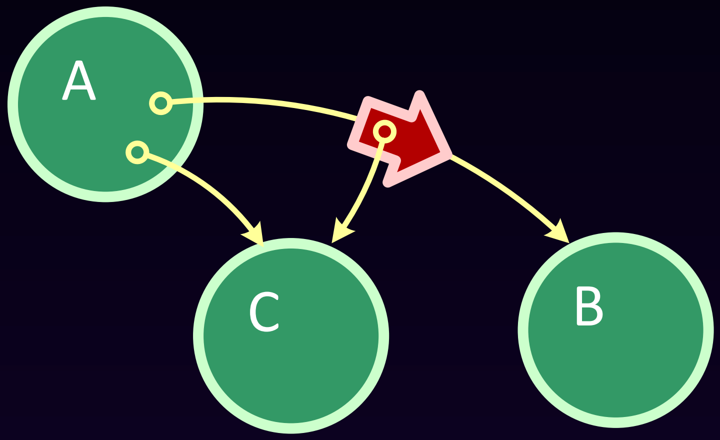
## 3. By Introduction

A has a references to B and C.
B has no references, so it cannot communicate with A or C.
C has no references, so it cannot communicate with A or B.



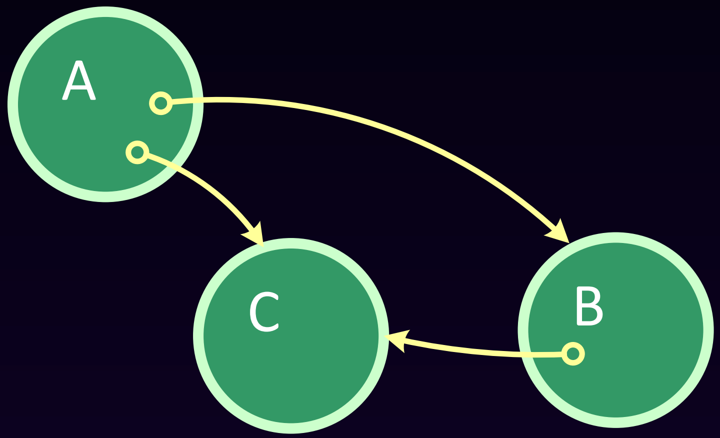
# 3. By Introduction

A calls B, passing a reference to C.



## 3. By Introduction

B is now able to communicate with C.



It has the capability.

If references can only be obtained by Creation, Construction, or Introduction, then you may have a safe system.

If references can be obtained in any other way, you do not have a safe system.

## Potential weaknesses include

1. Arrogation.
2. Corruption.
3. Confusion.
4. Collusion.

## 1. Arrogation

- To take or claim for oneself without right.
- Global variables.
- public static variables.
- Standard libraries that grant powerful capabilities like access to the file system or the network or the operating system to all programs.
- Address generation.

## 2. Corruption

It should not be possible to tamper with or circumvent the system or other objects.

## 3. Confusion

It should be possible to create objects that are not subject to confusion. A confused object can be tricked into misusing its capabilities.

## 4. Collusion

- It must not be possible for two objects to communicate until they are introduced.
- If two independent objects can collude, they might be able to pool their capabilities to cause harm.
- For example, I can give gasoline to one object, and matches to another. I need to be confident that they cannot collude.

## Rights Attenuation

- Some capabilities are too dangerous to give to guest code.
- We can instead give those capabilities to intermediate objects that will constrain the power.
- For example, an intermediate object for a file system might limit access to a particular device or directory, or limit the size of files, or the number of files, or the longevity of files, or the types of files.

Ultimately, every object should be given exactly the capabilities it needs to do its work.

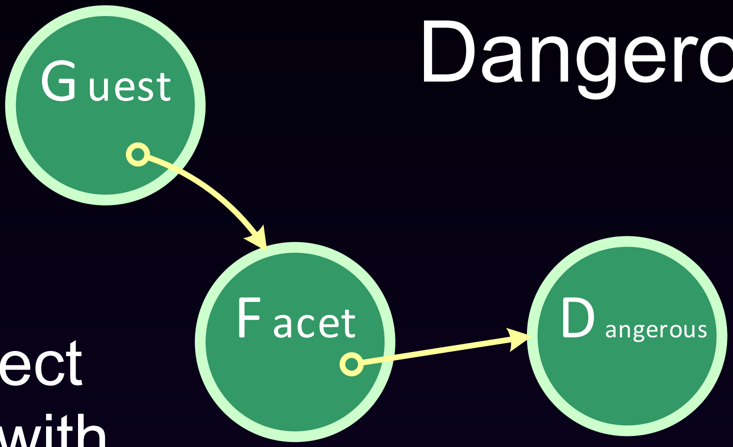
Capabilities should be granted on a need-to-do basis.

Information Hiding - Capability Hiding.

## Intermediate objects, or facets, can be very light weight.

Class-free languages can be especially effective.

The Facet object limits the Guest object's access to the Dangerous object.

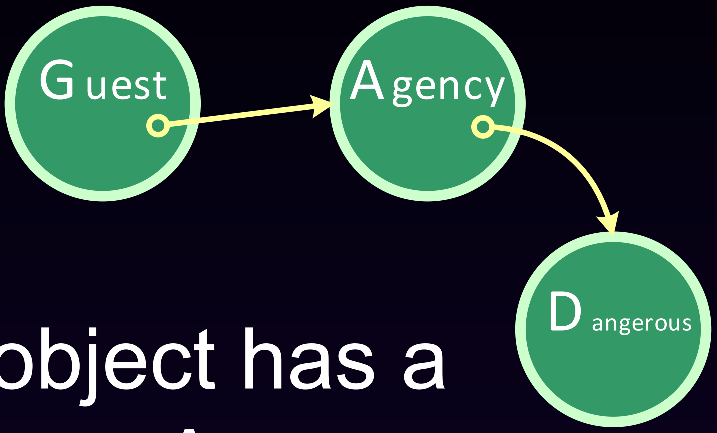


The Guest object cannot tamper with the Facet to get a direct reference to the Dangerous object.

## References are not revocable.

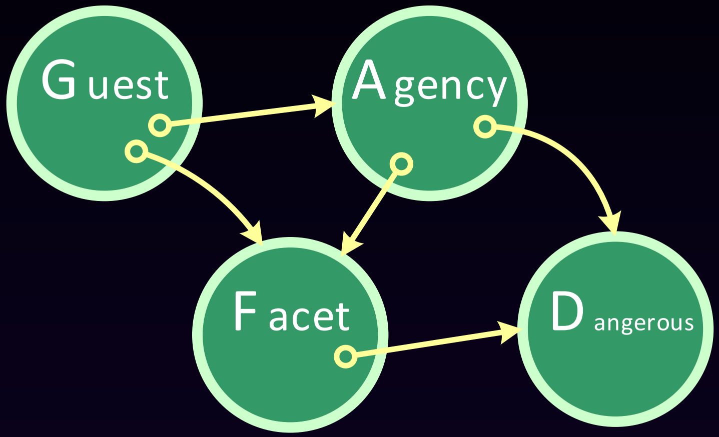
Once you introduce an object, you can't ask it to forget it.

You can ask, but you should not depend on your request being honored.



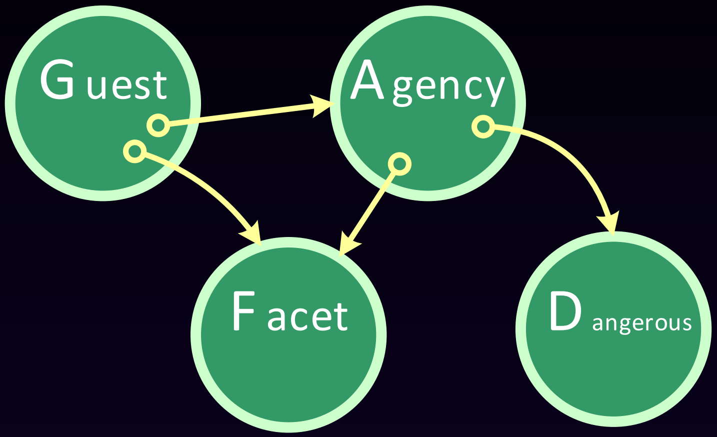
The Guest object has a reference to an Agency object. The Guest asks for an introduction to the Dangerous object.

## The Agency object makes a Facet, and gives it to the Guest.



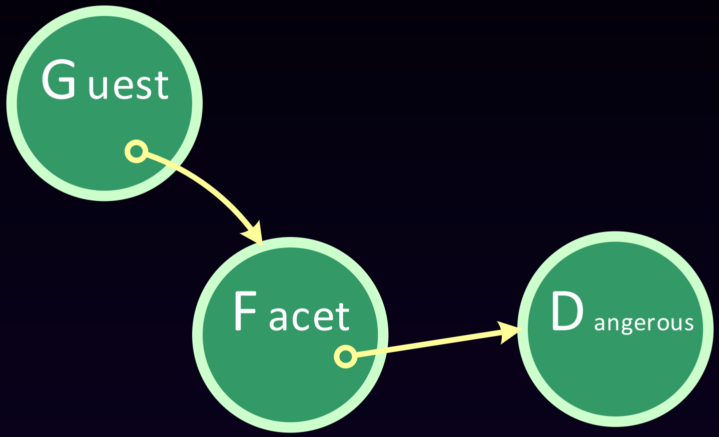
The Facet might be a simple pass through.

## When the Agency wants to revoke the capability, it tells the Facet to forget its capability.



The Facet is now useless to the Guest.

## A Facet can mark requests so that the Dangerous object can know where the request came from.



## Facets

- Very expressive.
- Easy to construct.
- Lightweight.
- Power Reduction.
- Revocation.
- Notification.
- Delegation.
- The best OO patterns are also capability patterns

Good Object Capability Design

is

Good Object Oriented Design

## Secure ECMAScript Must Be Incompatible With ES3

- If it were compatible, it would share the weaknesses of ES3.
- Incompatibility gives us license to correct many of the problems that ES3.1 must preserve.
- Lacking compatibility in the design process could lead to a lack of feature discipline.

## Minimal

- An elegant, minimal language is easier to reason about than an over-featured, maximal language.
- Committees are generally unable to produce minimal designs.
- We should avoid a Design-by-committee.

## Competition

- We invite members to submit designs.
- We drafts rules for the competition, and select a winner based on the criteria of security, expressiveness, and minimalism.
- Over several rounds of evaluation and influence, we may find either a clear winner or convergence on an ideal approach.

## Review of Current Work

- ES3.1
- FBJS
- Caja, Cajita (and E)
- Jacaranda
- dojox.secure
- JSON, ADsafe (and Misty)