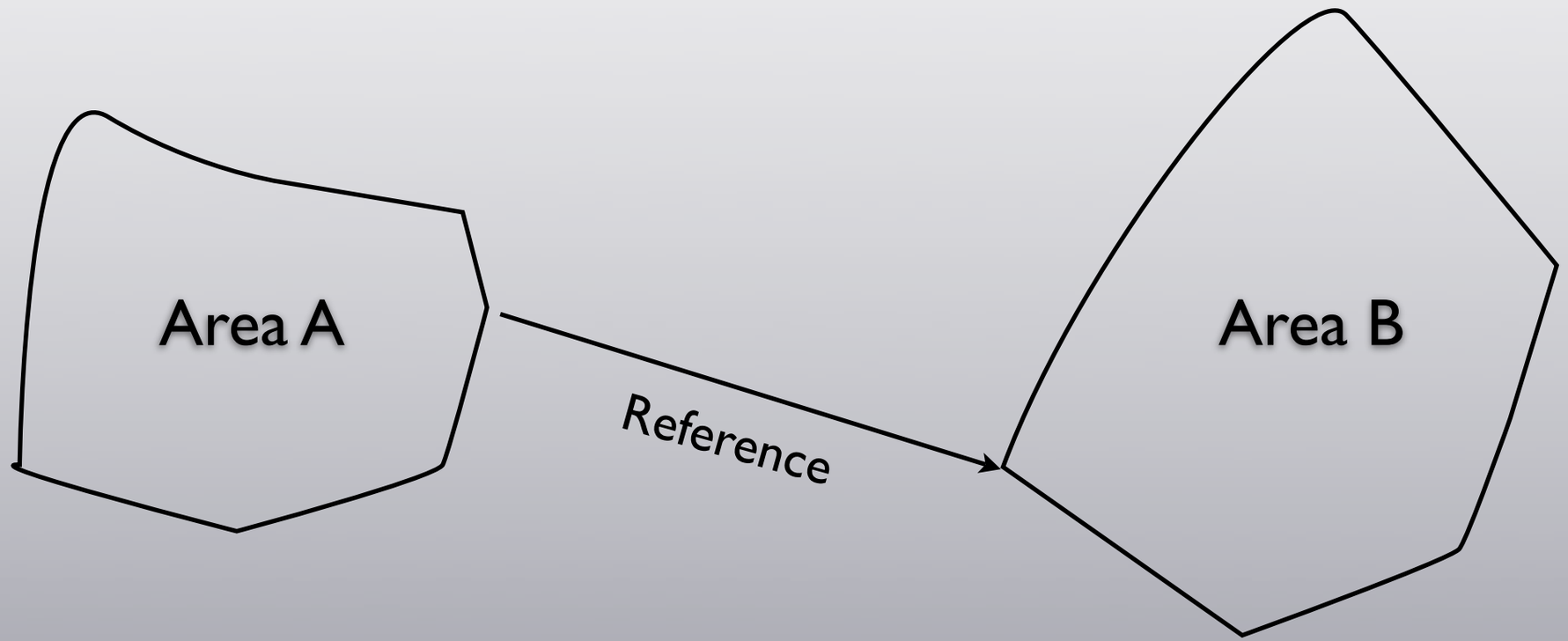
System Enforced Transitive Read-Only Objects in KeyKOS-like Systems

This talk is about system-level assurance of an isolation property.

Being a walk along one path through the design space.

There may be better paths. Go find them.

The Problem



We want objects in A to be able to get data from objects in B without being able to influence anything in B in any way.

For an Orange Book example, think of A as being Top Secret and B as being unclassified.

The Problem

```
Class Foo {
   private Object a;
   void setA(Object x) { a = x; }
   void getA(void) { return a; }
}
```


A dumb example with a getter and a setter. It is just complex enough to illustrate the point.

The Problem

We need two types of reference to a Foo:

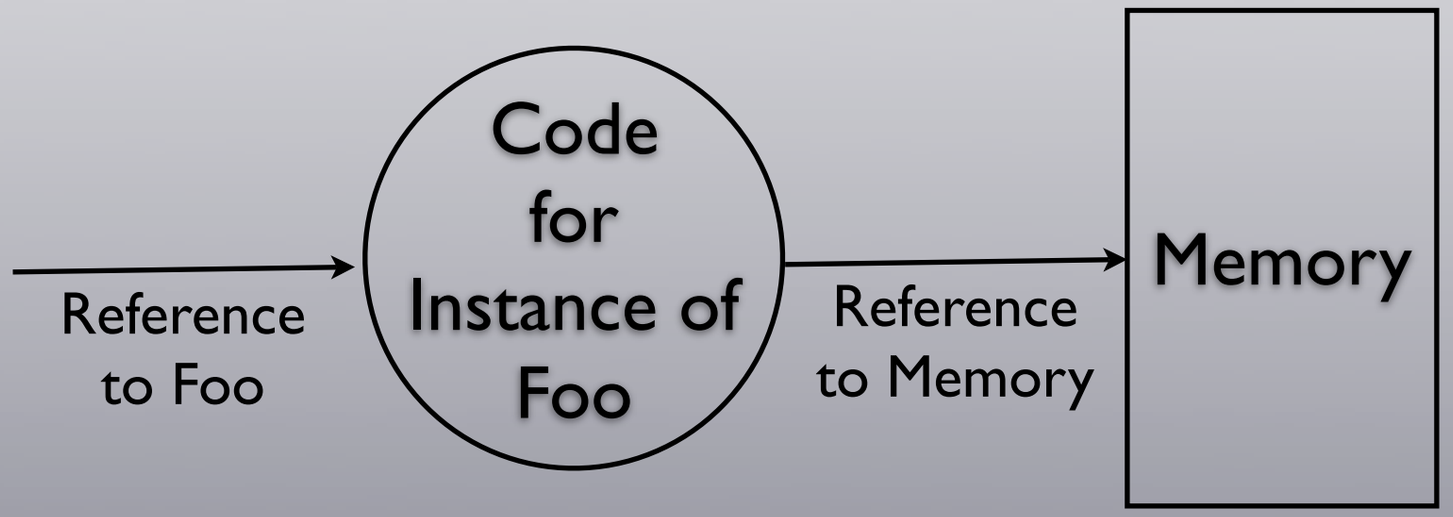
A strong reference which can use any method.

A "no-influence" reference where:

(1) setA() can't be used
(2) any object returned by getA() is transitively read-only,
(3) these restrictions are enforced by "the system".

If we were working at the language level, "the system" might be the language runtime. For this talk, "the system" is the OS kernel and the basic system objects.

The (Simplified) Object



In KeyKOS, the “Reference to Foo” is a start or resume key <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/18.html#gatetype>. The “Code for Instance of Foo” is part of a domain <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/15.html>

Existing: The Factory

- Standard object creation tool
- Allows auditing the communication paths new objects have when they are "born"
- Allows debug access with permission of both the type's "owner", and the instance's "owner"

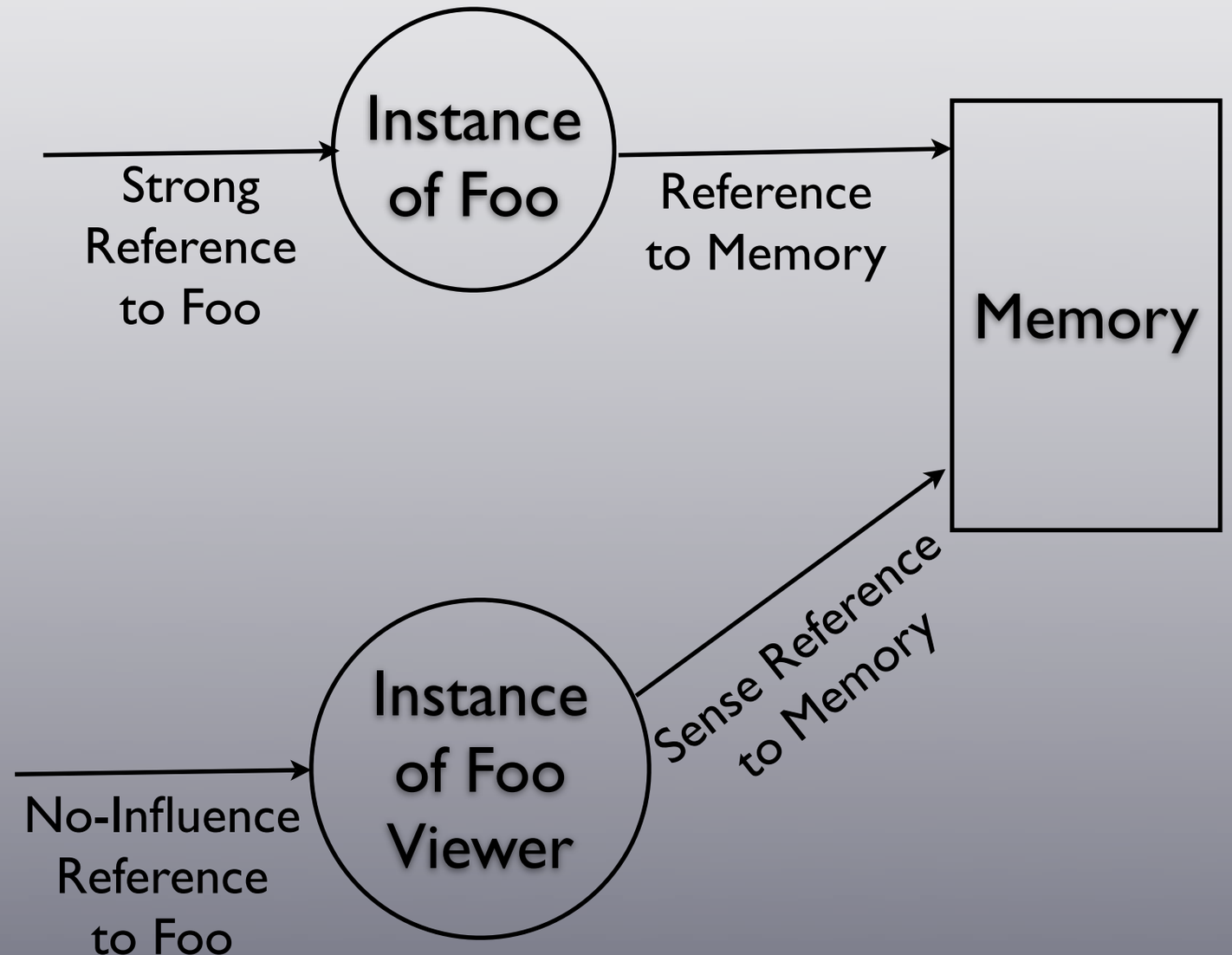
The notation "Existing" indicates that this facility exists in current versions of KeyKOS.
The factory: <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/68.html>

Existing: The Sense Reference

- Allows transitive read/only access to memory ☝
- Allows reference to a (very) few other objects ☝
- Sets general object references to null 👇
- Is known by the Factory ☝

See: <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/38.html>

Today, We Can:


Instance of Foo
Strong Reference to Foo
Reference to Memory
Memory
Instance of Foo Viewer
No-Influence Reference to Foo
Sense Reference to Memory

What It Does Right

- Provides both strong and no-influence accesses.
- Provides system assurance of transitive read/only.

What It Doesn't Do

- Provide for verifying the no-influence reference is indeed no-influence.
- Provide for calling sub-objects. i.e. It doesn't compose
- Handle race conditions for memory access

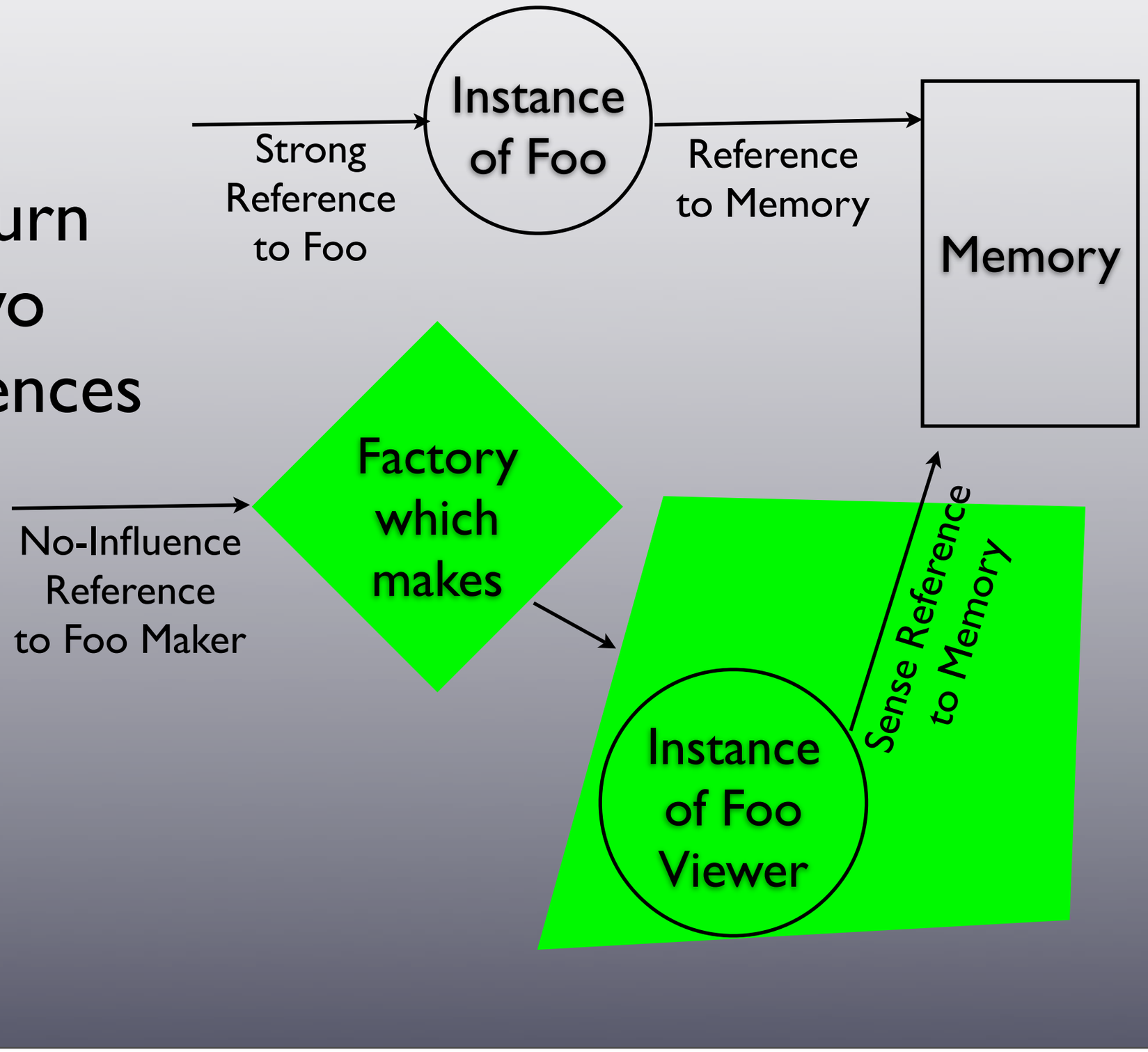
Features

- Instance of Foo Viewer can have its own state from call to call, perhaps making an Instance of Foo Viewer Instance Viewer desirable.
- We're glossing over the question of how the viewer's stack and heap addresses are allocated. Having an Instance of Foo Viewer Instance Viewer makes the answer more complex.

This slide mentions a potential can of worms which is not necessary to solve the basic problem.

Verifiability Today

Return two references



Since instead of getting a no influence reference, we get a reference to a factory which will create a no influence reference, we can use the standard factory mechanism <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/72.html#discr> to verify that the reference we get is indeed “no influence”.

Verifiability Today

Viewer Factory Components:

(1) Sense reference to instance of Foo's memory

(2) Read/Only reference to Instance of Foo Viewer code

This is a "no hole" factory

Hole: <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/73.html#hole> is a reference that factory can't say doesn't pass data. The factory knows about both read-only memory references and sense references.

New Sense References

- Provide sensory access to an object
- Have “discretion” for use by the factory
- Pass factories at least as discreet as it is

Having seen what we can do with today’s system. Lets see how those ideas can take us a bit further. We describe a “new sense reference” with some factory-like properties. The properties are continued on the next slide.

New Sense Function

- References to nodes become sensory references to the same node
- Factories must be at least as discreet as the sense reference or they become null
- Discrim, sensory references, etc. are passed unchanged
- All other references become null

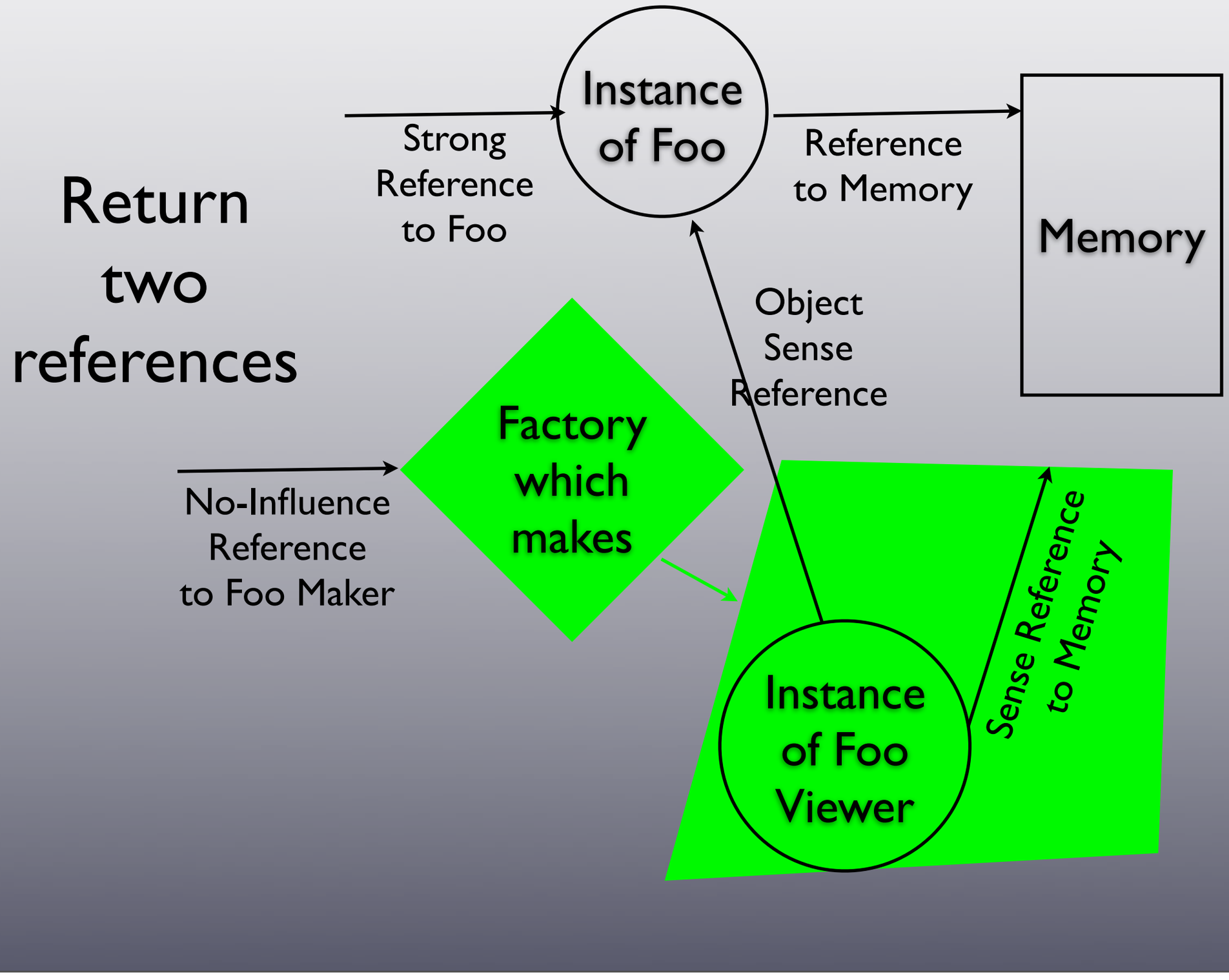
Discrim, sensory references etc. are mentioned to be pedantically complete. See <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/38.html> for a list of the current sense reference's special cases.

Object Sense Reference

- Allows sensory access to an object's reference registers
- Allows sensory access to an object's address space
- Allows sensory access to an object's extra slots, C10 and C11
- Allows access to an object's data registers

The "object sense reference" is a sensory version of a domain key <http://www.cis.upenn.edu/~KeyKOS/agorics/KeyKos/Gnosis/41.html>

New Sense References



The “object sense reference” allows the viewer to fetch sensory versions of any references that Foo holds.

How Do We Get No-Influence References to Sub-Objects?

- If our object uses other objects, we can use the sensory reference to fetch from it's reference registers to access them.
- But we don't have a way of getting their no-influence factories.

But typically Foo will not be using sensory references to its sub-objects. But the viewer can only fetch sensory references.

Two ways to get References to Sub-Objects

- A new kernel function which gets it from the R/W object
- By cooperation between the R/W object and the R/O viewer

We could have a convention where each R/W object stores the reference to its viewer factory in a standard location and have a kernel function which fetches that location. Or we can leave that level of cooperation to the objects' programmer.

The Get No-Influence Factory Tool

- Define an object register which holds the No-Influence Factory (NIF) reference
- No such reference: this register holds Null
- Get NIF Tool takes an object reference and returns the object's NIF register contents
- Tool is used by the Object sense reference

This is a look at the requirements for the kernel function in the last slide. Note that it is closely held by the "object sense reference" which will perform discretion tests on any references it fetches via this tool.

New Sense Function

- References to nodes become sensory references to the same node
- Object references use the NIF Tool
- Factories must be at least as discreet as the sense reference or they become null
- Discrim, sensory references, etc: unchanged
- All other references become null

And this is how the kernel function and the “object sense reference” work with the new sense function.

Cooperative Option

- Factories return both the R/W reference and the NIF reference
- When a sub-object is built, the builder saves the NIF reference for use by its NIF during construction of a viewer
- Requires more complex reference structures for complex object graphs
- Get NIF Tool is an optimization

Assume no special kernel tool and lay the work on the programmer. This is what would have to happen.

Issues and Annoyances

- Need a new factory for each object instance
- An object can execute concurrently with its viewers

Potentially lots of factories, with one for each R/W object instance.
Race conditions accessing memory.

Need A Factory For Each Object instance

- Factory can be one node with a debugging reference to object + other data for factory with factory code as “keeper”
- A new object reference type which, with conventions, eliminates the need for a node

Ways to make the multiple factories cheaper.

Concurrent Execution

- Wall banging is an issue with most solutions
- All solutions have significant issues

These next few slides try to show the options for the concurrent execution issue.

Concurrent Execution Approaches

- Meter manipulation
- One object w/separate references
- “Canned” front end that vectors requests
- Virtual copy memory for stability
- Object entry + exit counts

Meter manipulation

- Meter keeper ensures either R/W object's meter is on or the viewer's meters are on
- Trap on no objects running under a meter has never been implemented
- Timing tests can notice viewers running

One object with separate references

- Change references for object or for each viewer
- Major kernel changes
- Not clear how to handle mutually suspicious viewer objects
- Natural for a Vat oriented system

By “vat oriented system” think of the E language runtime <http://www.erights.org/> or Waterken <http://waterken.sourceforge.net/>.

“Canned” front end that vectors requests

- Timing busy state allows view use to be detected by R/W object’s users
- Limits flexibility of object interface
- Vectoring object needs to be trusted
- Adds additional gate jumps to the path

Virtual copy memory for stability

- Cost of virtual copy for each change in object
- Doesn't provide synchronization with sub-objects

Entry + exit counts

- Object increments a counter on entry and another on exit
- Viewers repeat their methods until both counters are unchanged
- Viewers not assured of termination
- Good from a wall banging point of view
- Transient invalid states cause viewers to trap