Tradeoffs in Retrofitting Security: An Experience Report

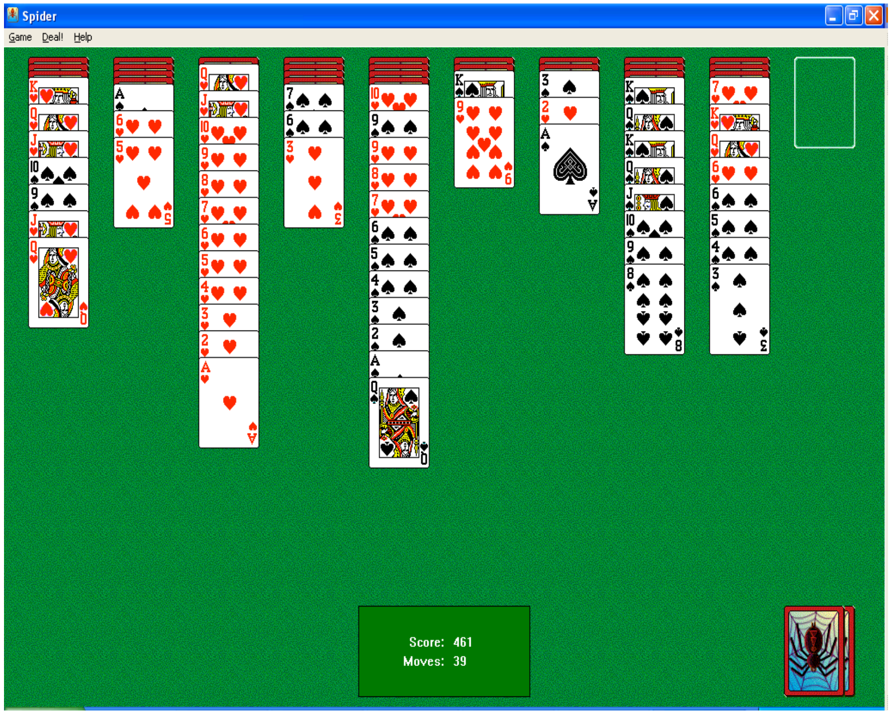
Mark S. Miller

Early Choice. Late Despair

- ACLs and OCaps start in mid ‘60s.
- DVH before specialization of CS
- ‘70s: Industry took ACL fork in road.
- ‘90s to present: Rise of Malware
- But:
  - You can’t start over again
  - You can’t add security later
- What to do?

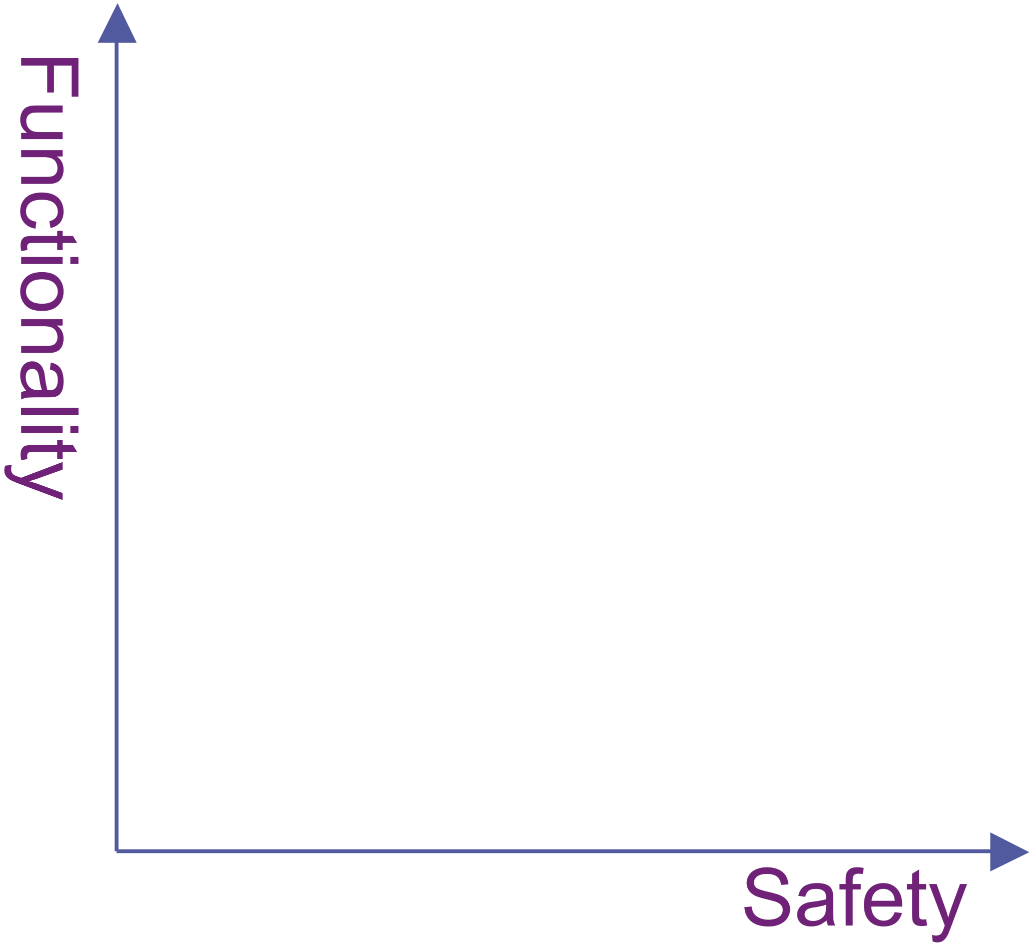
A very powerful program

A very powerful program

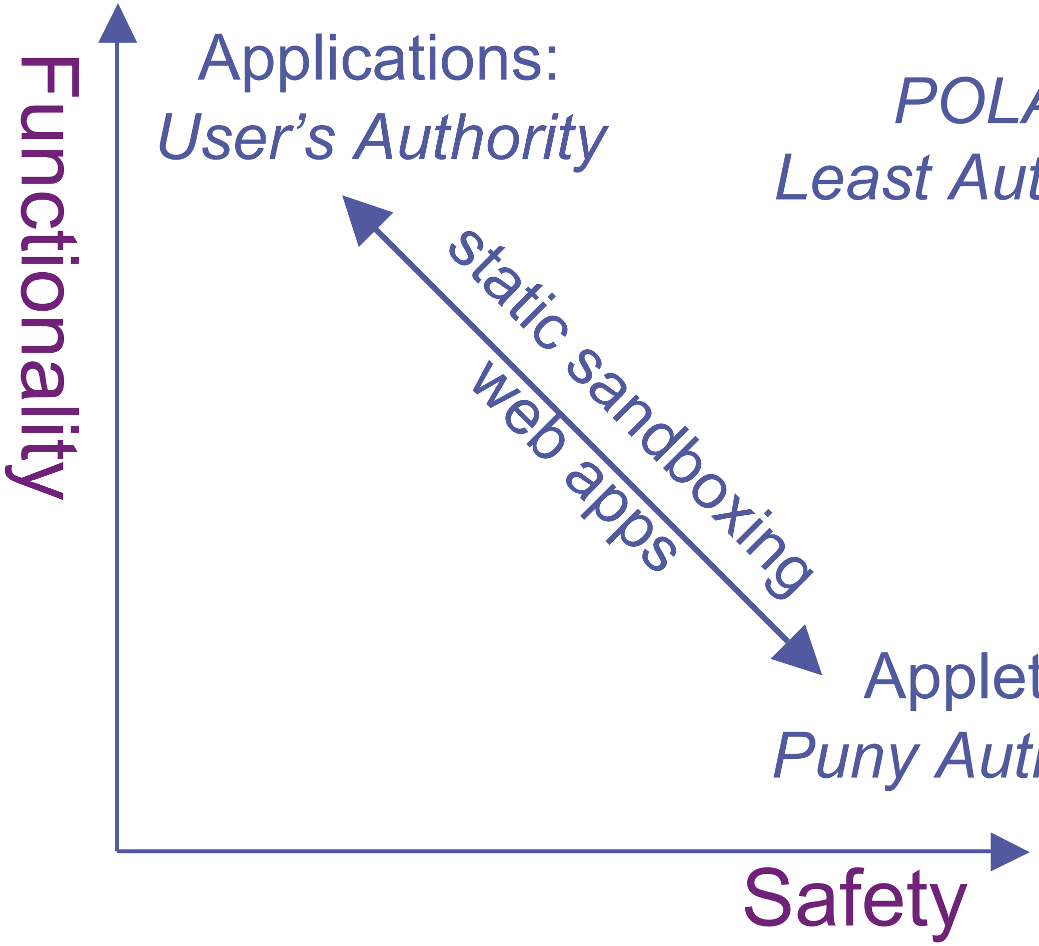


Solitaire can delete any file you can.

Functionality *vs.* Safety?



Functionality *vs.* Safety?



A Tale of Two Copies

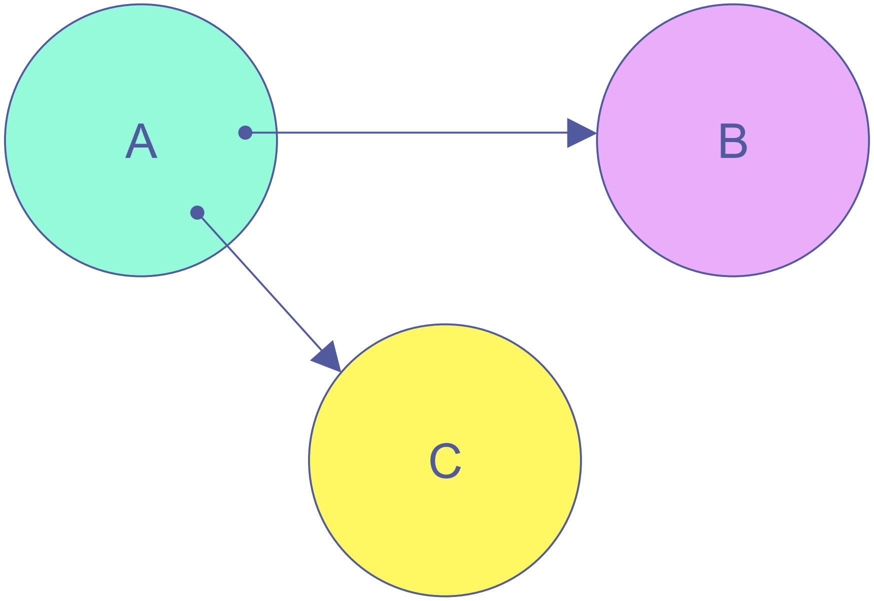
```
$ cp foo.txt bar.txt
```

vs.

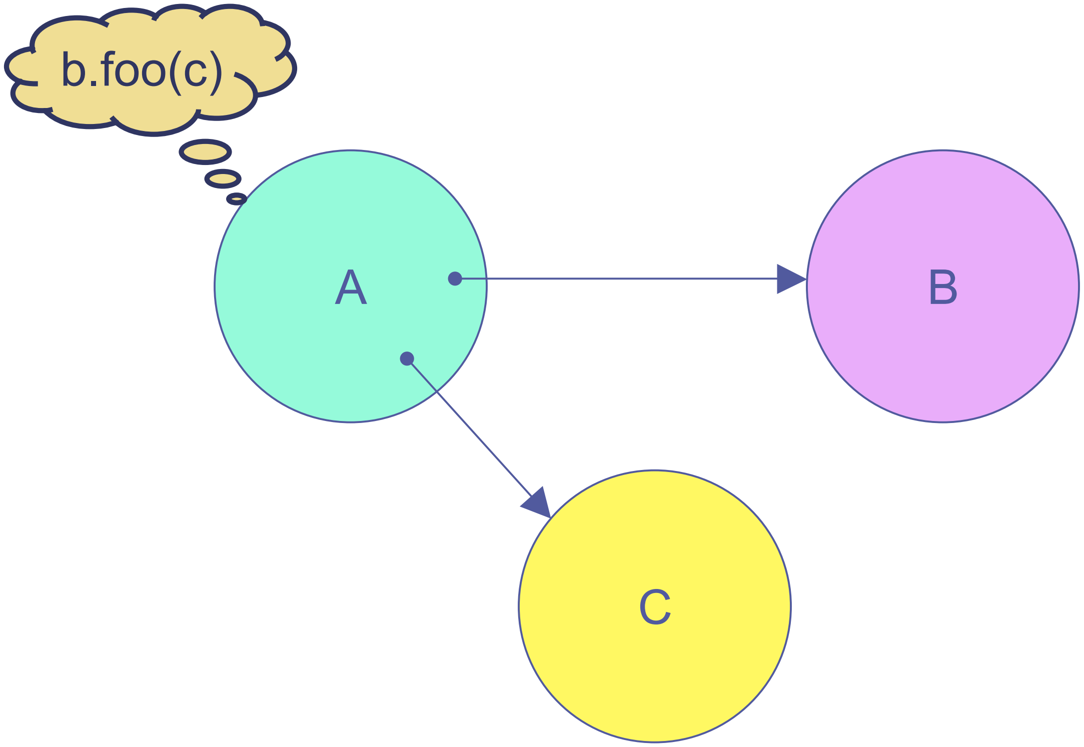
```
$ cat < foo.txt > bar.txt
```

- Bundle authorization with designation.
- Remove ambient authority.

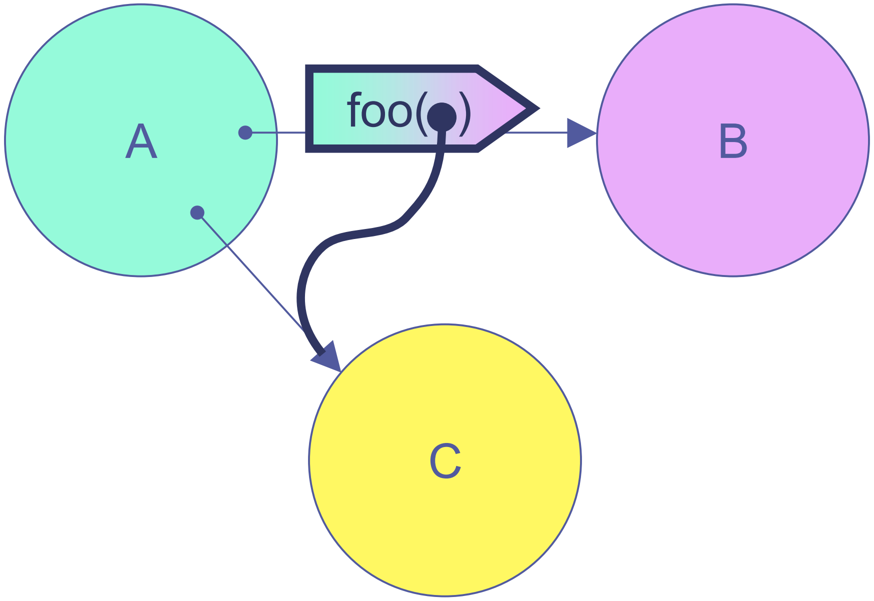
Objects



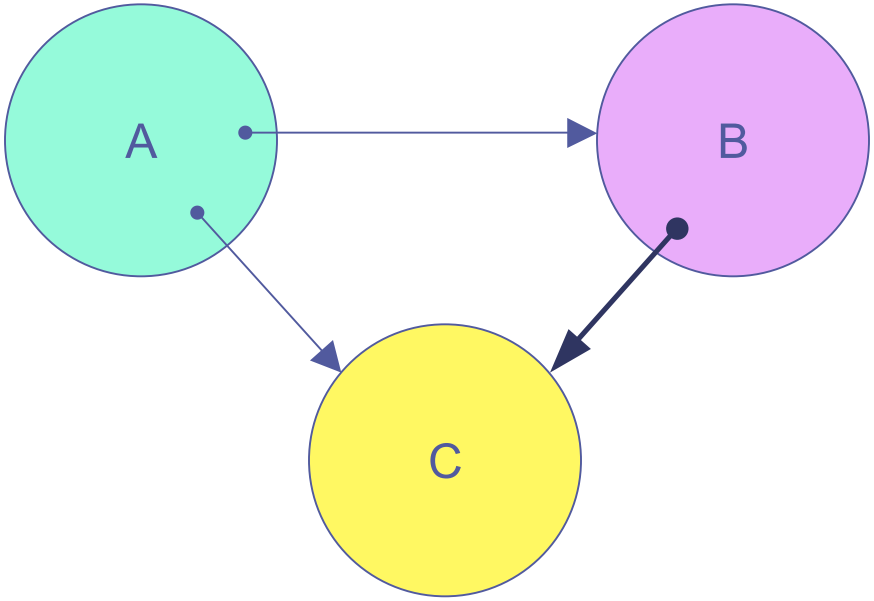
Objects



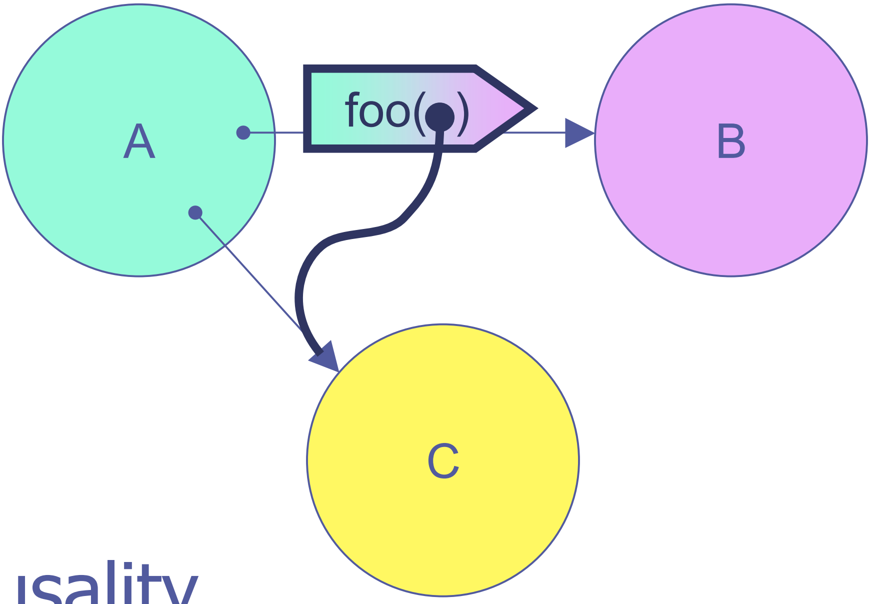
Objects



Objects



Object-Capabilities

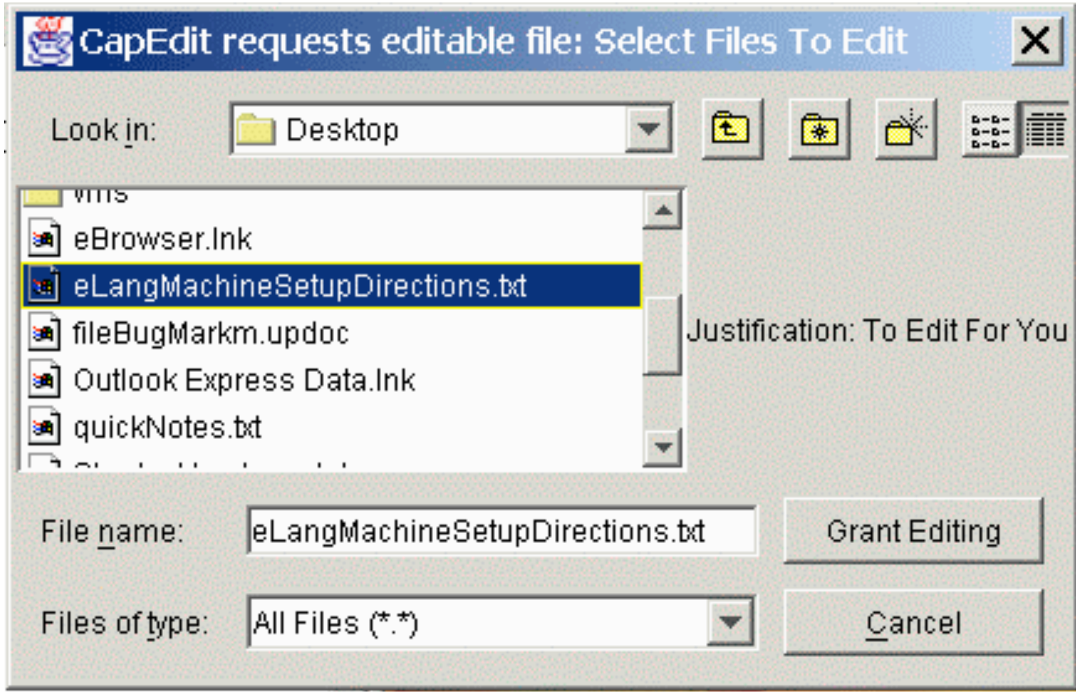


- Inter-object causality **only** by sending messages on references
- Reference graph == Access graph
- *Only connectivity begets connectivity.*

CapDesk demo

CapDesk, Polaris, BitFrost:
Usable POLA

- Double click launch
- File Explorer
- Open dialog
- Drag/Drop
- Etc...



CapEdit requests editable file: Select Files To Edit

Look in: Desktop

vms
eBrowser.lnk
eLangMachineSetupDirections.txt
fileBugMarkm.updoc
Outlook Express Data.lnk
quickNotes.txt

Justification: To Edit For You

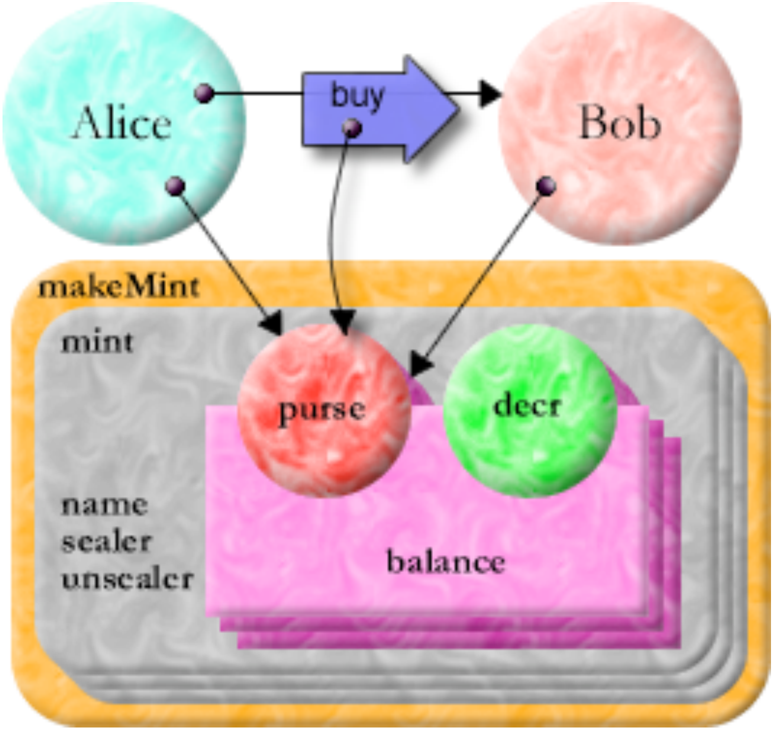
File name: eLangMachineSetupDirections.txt

Files of type: All Files (*.*)

Bundle authorization with designation

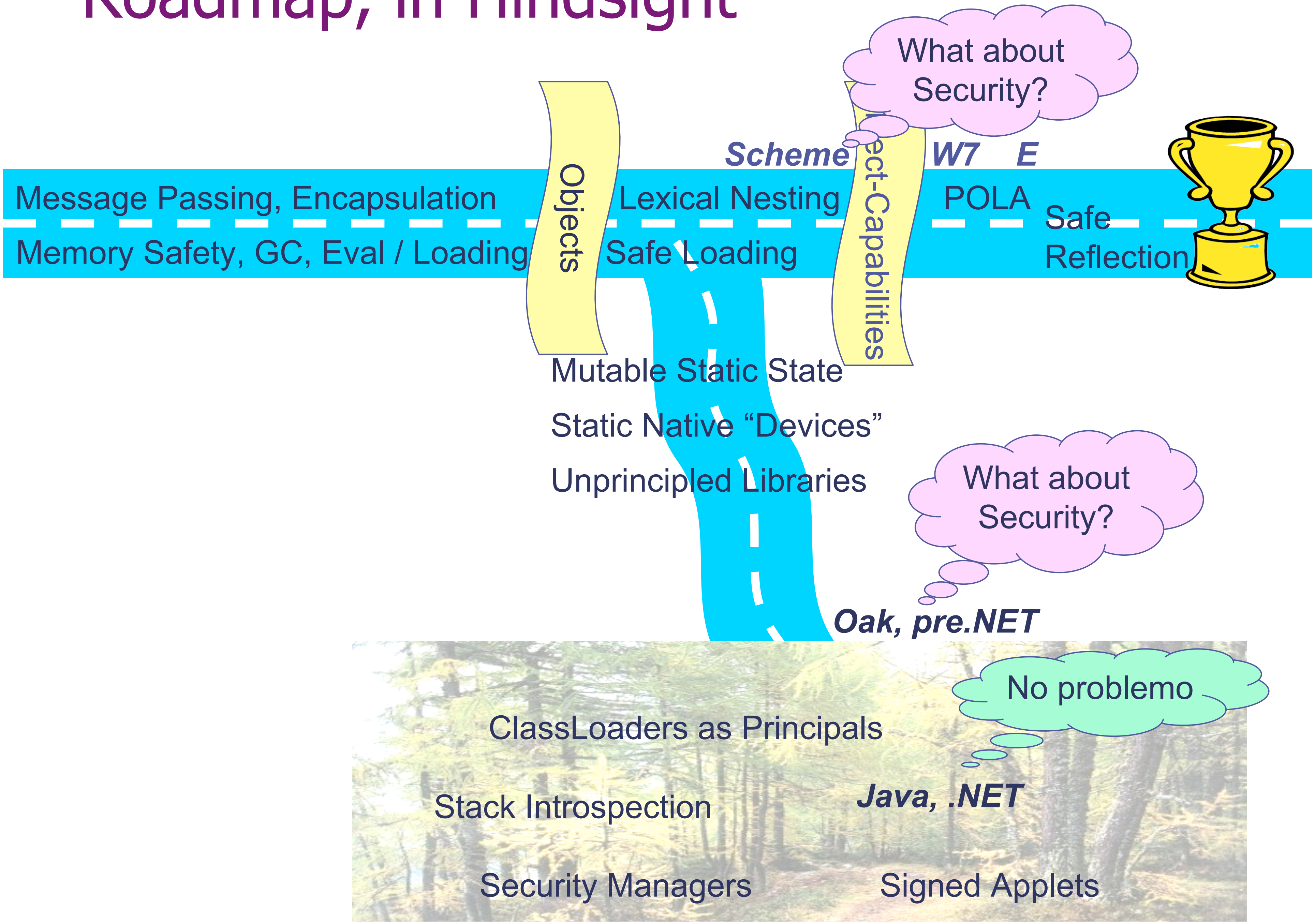
Distributed Secure Money in Caja

No explicit crypto



```
function Mint(name) {
   caja.requireType(name,'string');
   var brand = Brand(name);
   return function Purse(balance) {
      caja.requireNat(balance);
      function decr(amount) {
        caja.requireNat(amount);
        balance = caja.requireNat(balance - amount);}
      return caja.freeze({
        getBalance: function() { return balance; },
        makePurse: function() { return Purse(0); },
        getDecr:     function() { return brand.seal(decr); },
        deposit:     function(amount, src) {
           def newBal := caja.requireNat(balance+amount)
           brand.unseal(src.getDecr())(amount);
           balance := newBal;}});};;}
```

Roadmap, in Hindsight



What about Security?

Scheme ***W7*** ***E***

Message Passing, Encapsulation

Objects

Lexical Nesting

ect-Capabilities

POLA

Safe Reflection

Memory Safety, GC, Eval / Loading

Safe Loading

Mutable Static State

Static Native “Devices”

Unprincipled Libraries

What about Security?

Oak, pre.NET

No problemo

ClassLoaders as Principals

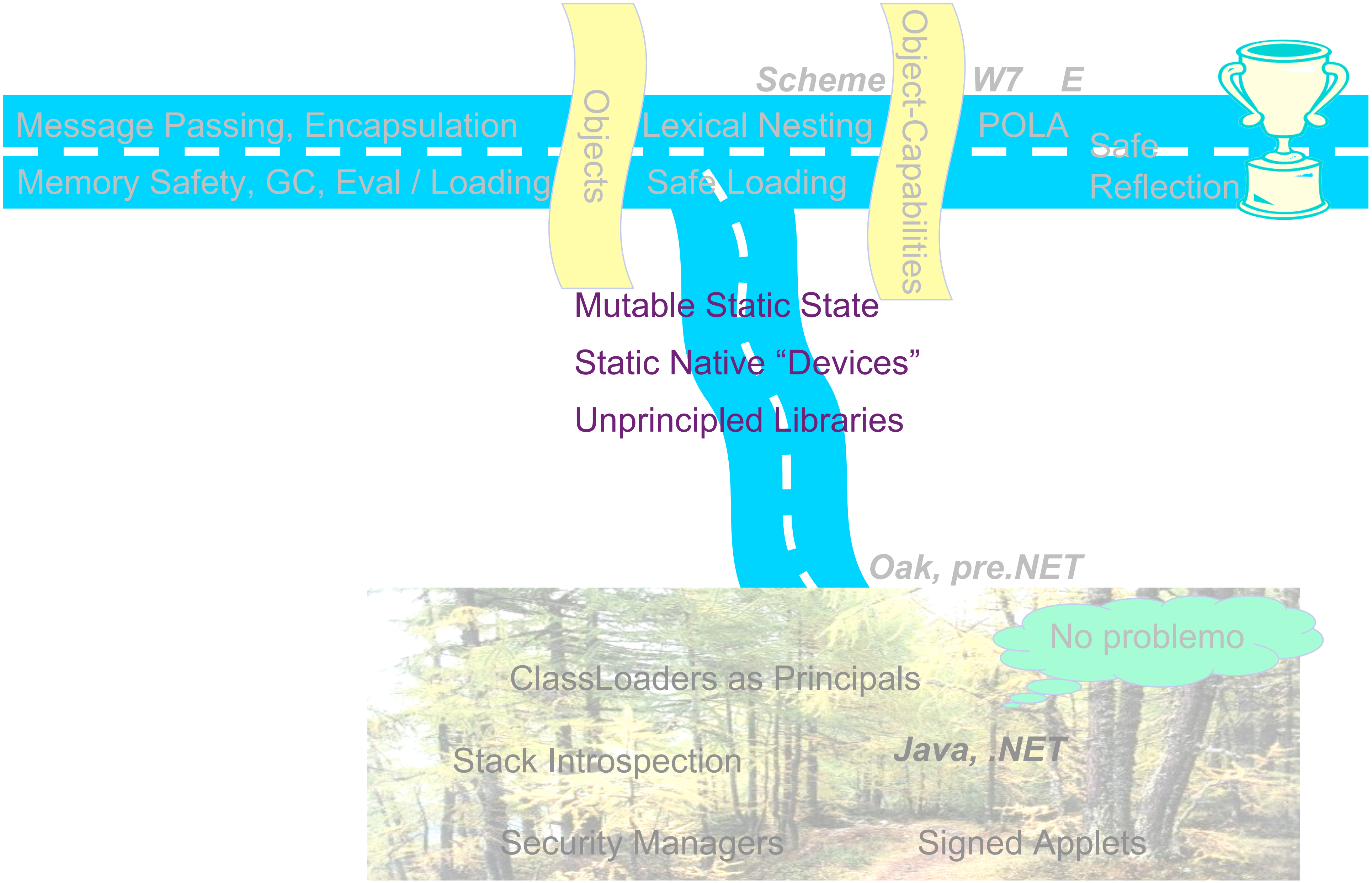
Java, .NET

Stack Introspection

Security Managers

Signed Applets

Detour is *Non-Object Causality*



Objects

Object-Capabilities

Scheme

W7

E

Message Passing, Encapsulation

Memory Safety, GC, Eval / Loading

Lexical Nesting

Safe Loading

POLA

Safe Reflection

Mutable Static State

Static Native "Devices"

Unprincipled Libraries

Oak, pre.NET

No problemo

ClassLoaders as Principals

Stack Introspection

Java, .NET

Security Managers

Signed Applets

Oak to Java

Oak (Java’s simple ancestor)
+ ClassLoaders as Principals
+ SecurityManagers
+ stack introspection
+ policy files
+ signed applets

Painful and Inflexible Security

Don’t add security.

Java to Joe-E

Java

— all those “security” mechanisms

— mutable static state

— static native “devices”

— unprincipled parts of libraries

+ library of principled replacements

Simple and Expressive Security

Remove insecurity.

But isn’t that stuff useful?

```
public class Foo {
  static private int count = 0;
  public Foo() {
    count++;
    …
} }
```

But isn’t that stuff useful?

```
public class Foo {
  static private int count = 0;
  public Foo() {
    count++;
    …
} }
```

```
public class FooMaker {
  private int[] countCell = {0};
  public class Foo {
    public Foo() {
      countCell[0]++;
      …
} } }
```

But isn’t that stuff useful?

```
public class Foo {
  static private int count = 0;
  public Foo() {
    count++;
    …
} }
```

```
public class FooMaker {
  private int[] countCell = {0};
  public class Foo {
    public Foo() {
      countCell[0]++;
      …
} } }
```

Unnecessary awkwardness.

But better engineering anyway:
All state is multiply instantiable.

Choice: Intellectual Communities

- Traditional OS access control
  - + Brilliant early history
  - - Misdirected priorities, Accumulated Myths
  - Windows -> Polaris
  - Linux -> Plash, BitFrost
- Crypto
  - + Serious about security, High standards
  - - Platform security is *Someone Else's Problem*
  - HTTPS -> Webkeys, Foolscap, Second Life
- Programming Language
  - + Abstraction, Modularity, Composition
  - - Security is *Someone Else's Problem*

Choice: How to secure a language

- New language
  - **Gedanken**, Emerald, Joule, Toontalk, E, AmbientTalk, Sebyla
- Statically verified subset
  - **Joe-E**, Emily, Backwater, JSON, ADsafe, Pthin
- Dynamic restrictions, rewriting
  - W7, Oz-E, CaPerl, **Caja**, FBJS?, Squeak-E
- Wrapper-based isolation
  - **J-Kernel**, Squeak Islands, Earlier Caja attempts
- Sandboxed virtual machines
  - **Java Isolates?**, Tweak Islands, Secure Python

New Languages

- Object-grain
- port programmers, not programs
  - Algol 60 -> Gedanken
- Pros:
  - + Ideal laboratory for new ideas
  - + Ideal teaching vehicle
- Cons:
  - - Huge barrier to adoption

Statically verified subset

- Object-grain
- No rewrite
- Static library taming

  - Joe-E Example: No non-final static variables

  - + 100% compatibility with tool chain
  - + No measurable runtime cost

  - - For dynamic languages, restrictions can be severe
    - JSON, ADsafe, Pthin

Dynamic restrictions, rewriting

- Object-grain
- Dynamically substituted scope, rewriting
- Virtualized Libraries

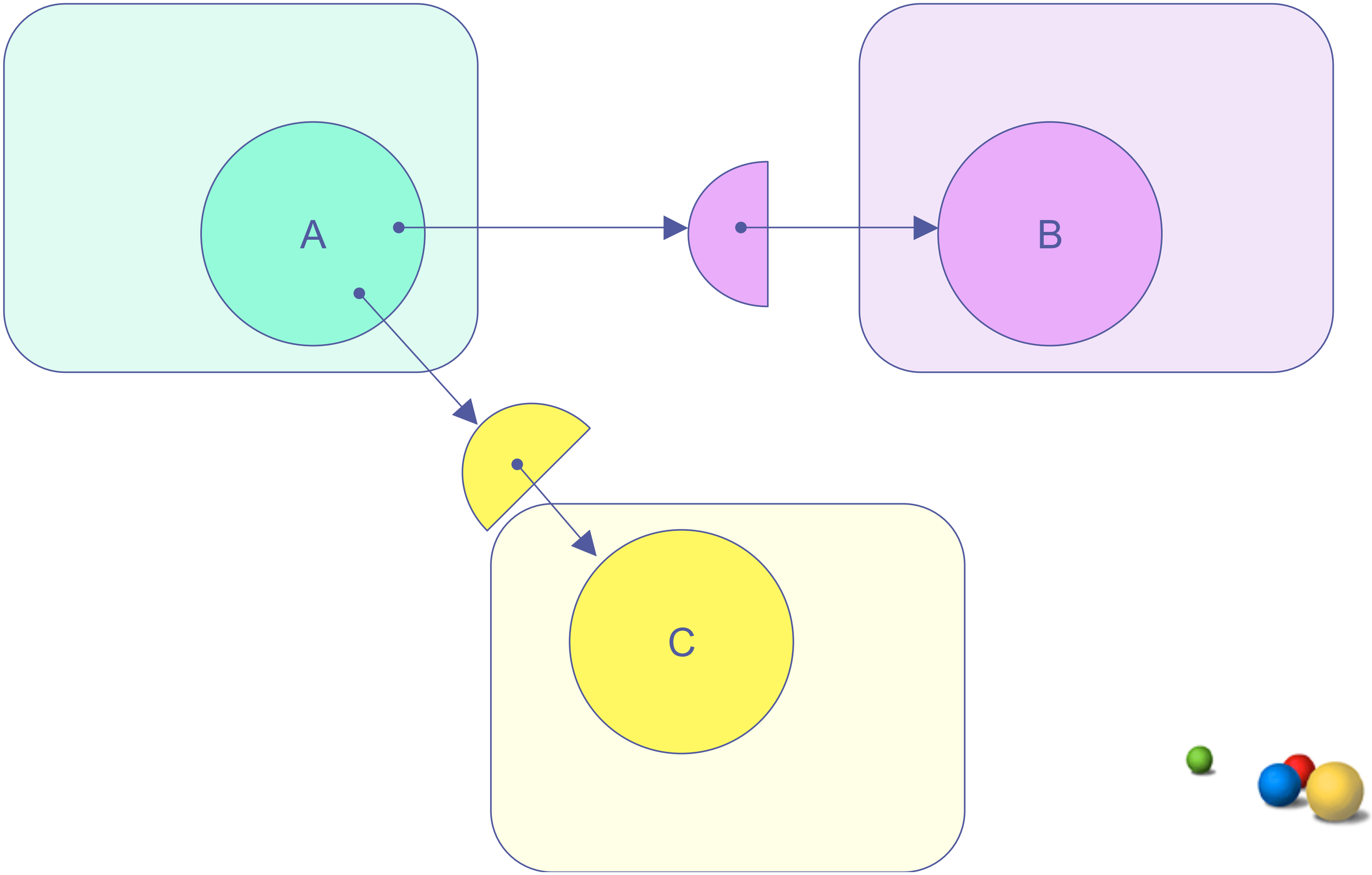
  - Caja Example:

    ```
      foo.bar
    ➔ foo.bar_canRead___ ? foo.bar : ___.read(foo,”bar”)
    ```

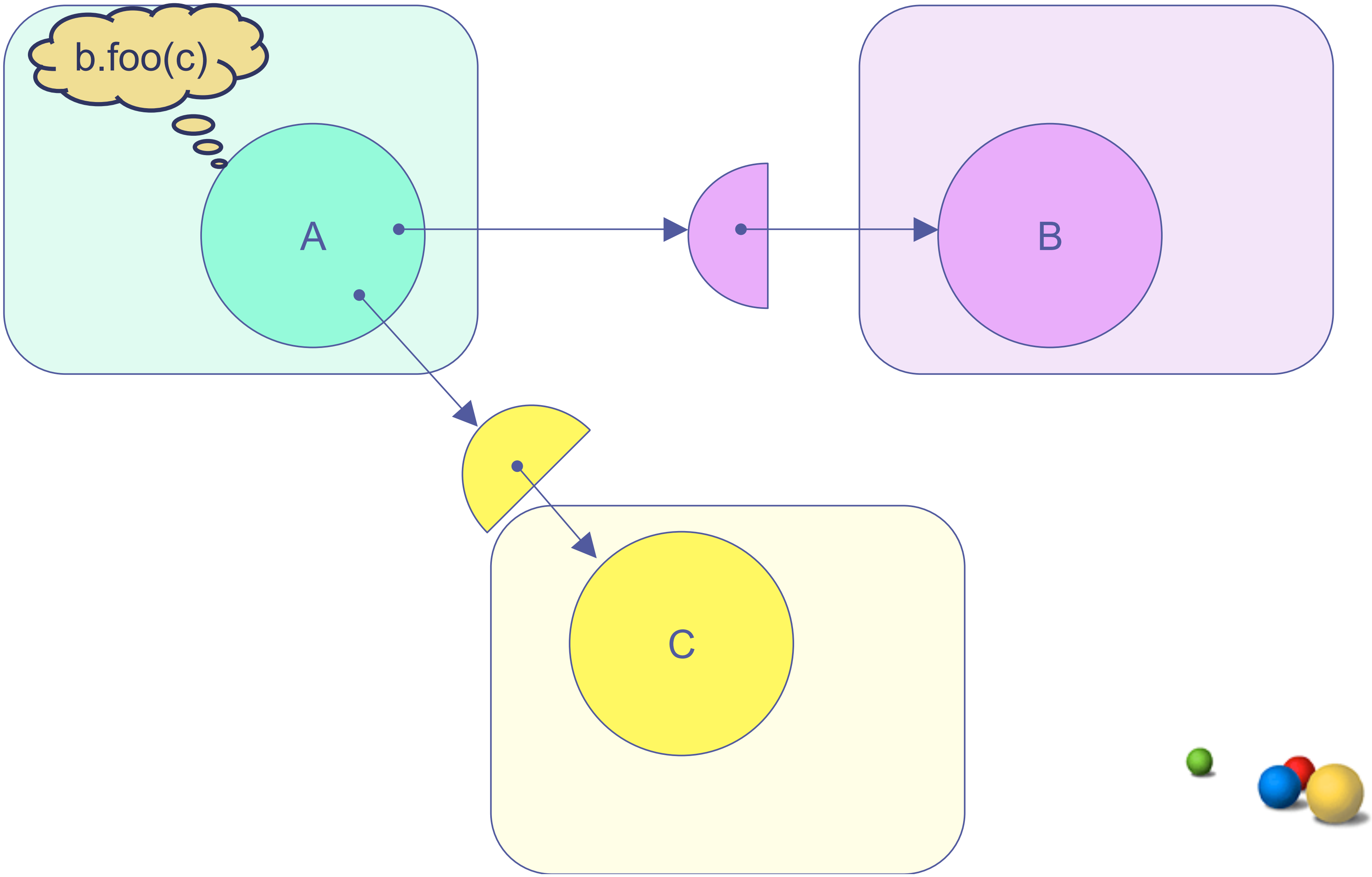
  - + More permissive rules possible

  - - Src is one transform removed from IDE’s view
  - - Runtime cost

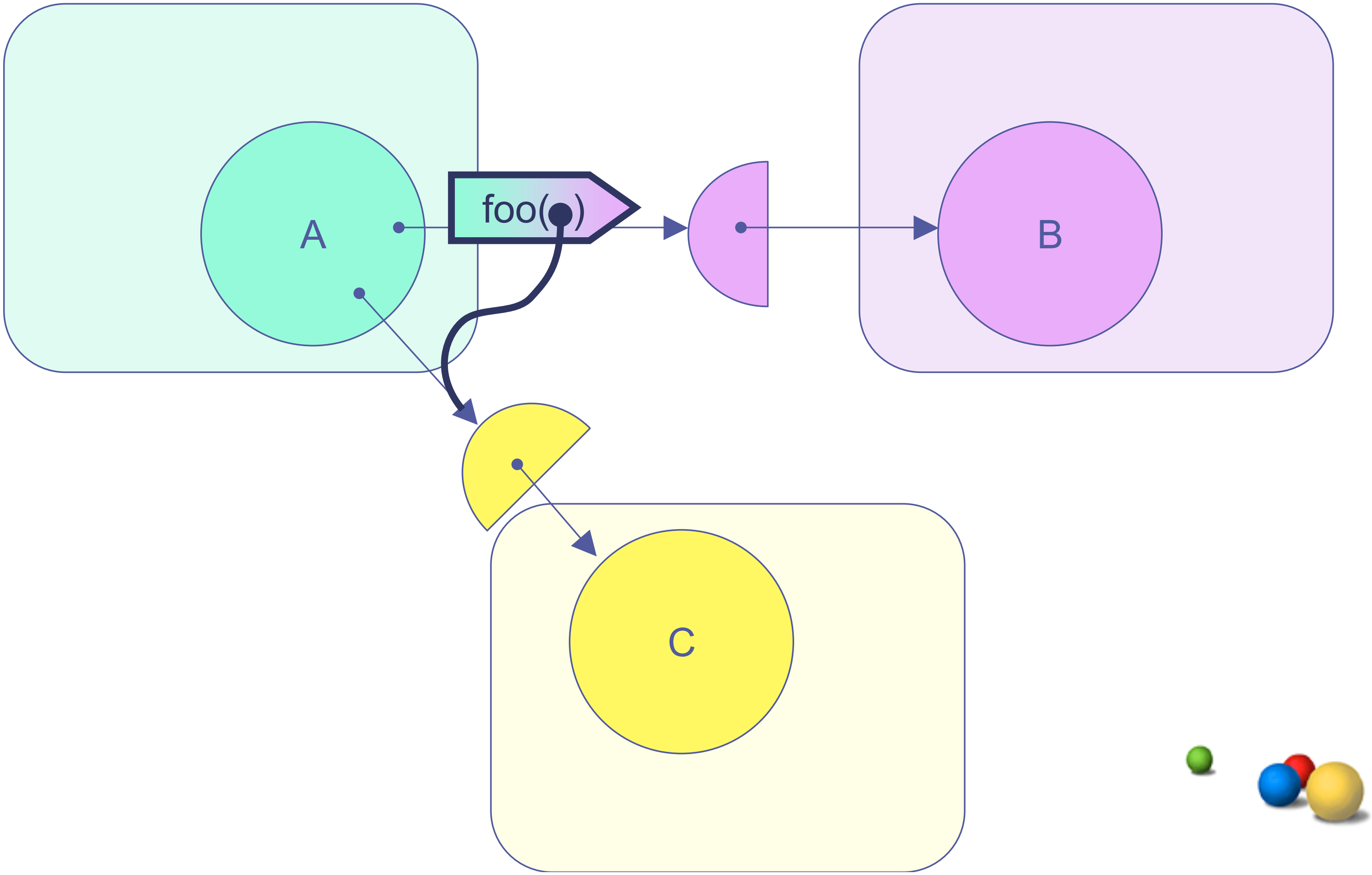
Wrapper-based Isolation



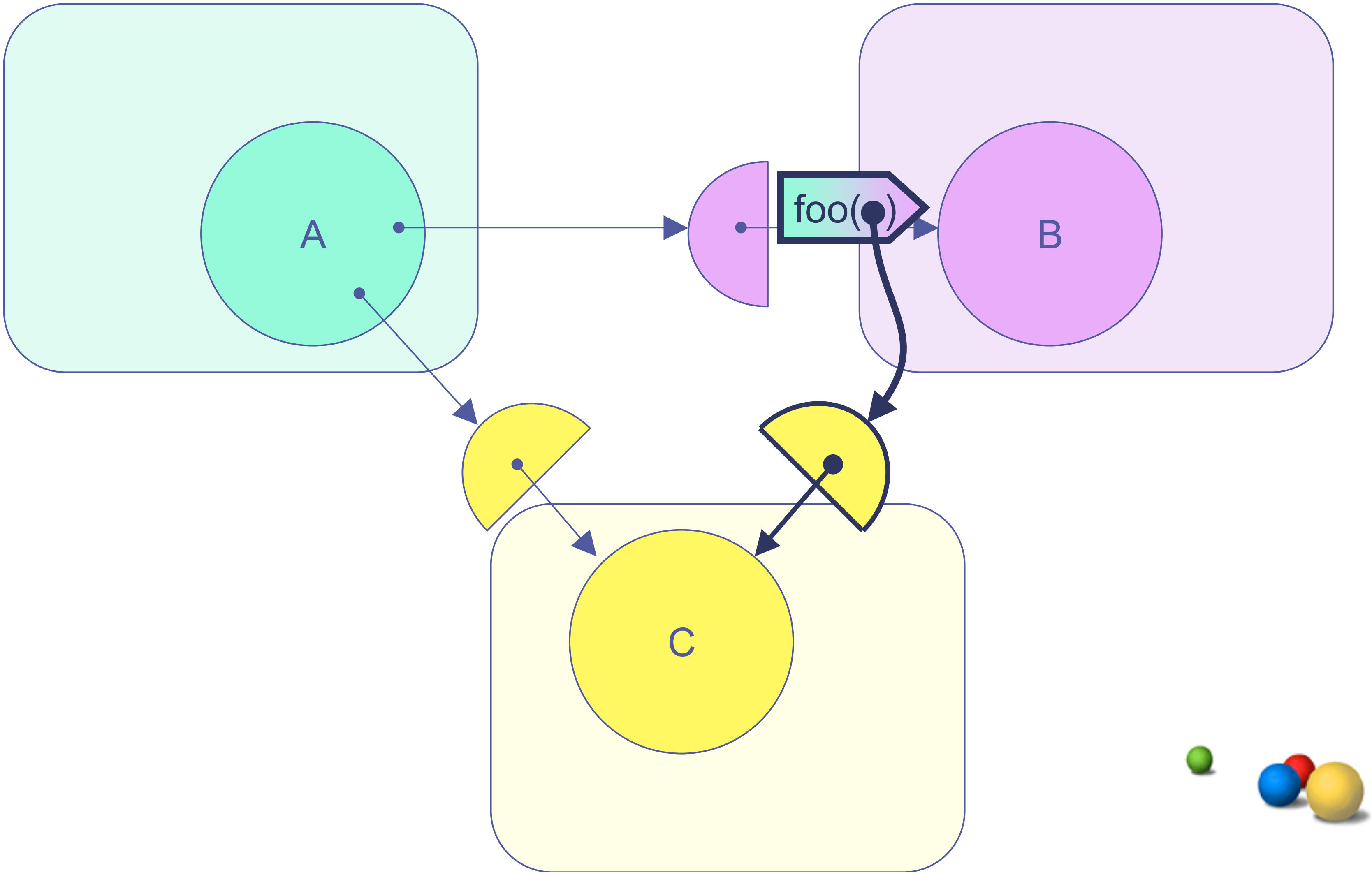
Wrapper-based Isolation



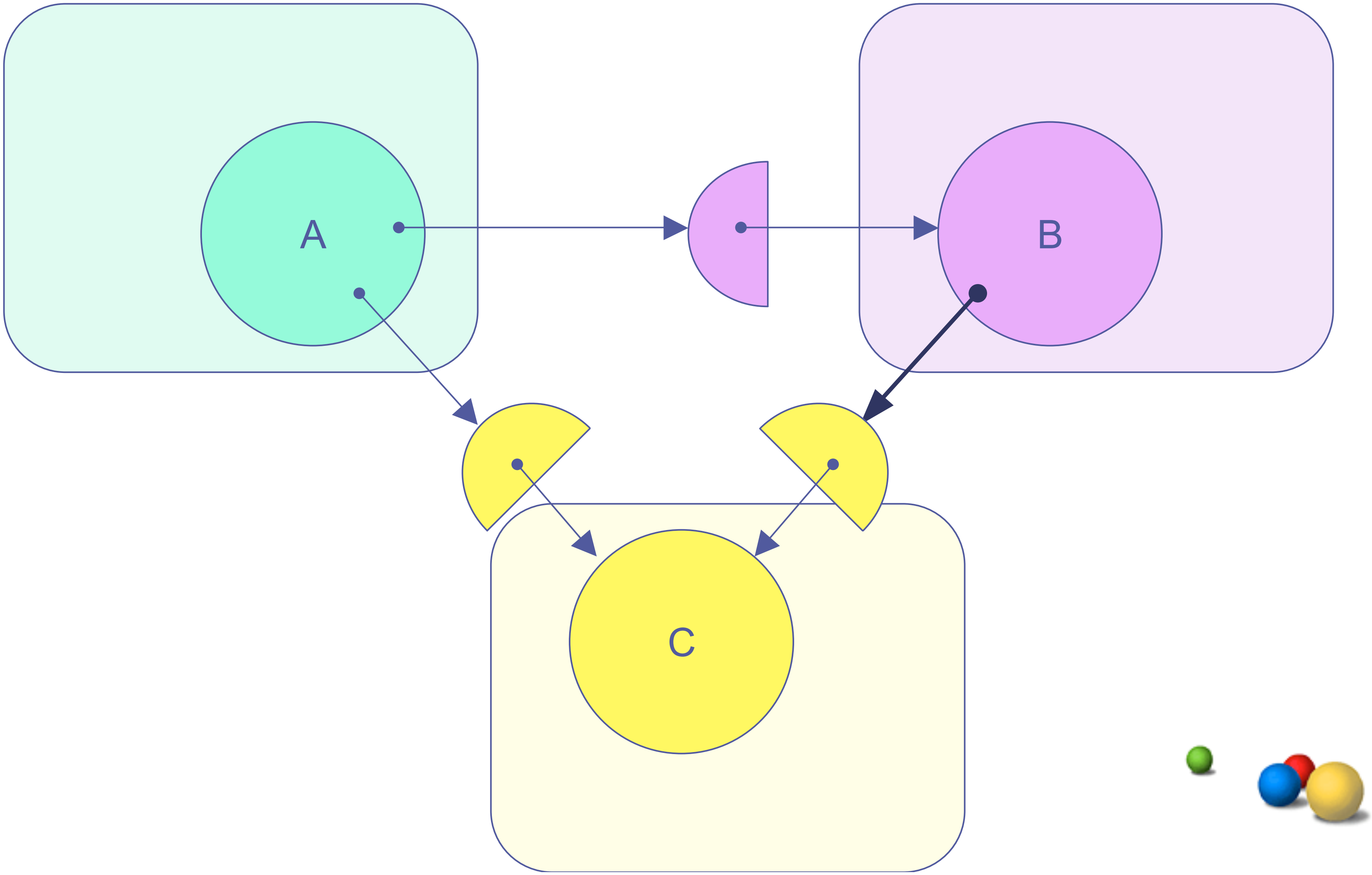
Wrapper-based Isolation



Wrapper-based Isolation



Wrapper-based Isolation



Wrapper-based Isolation

- Component-grain
- Synchronous membrane/wrappers
- Virtualized Libraries, Rewriter?
  - Java 1.1 -> J-Kernel
  - + More compatible with old code
  - - Domain switching overhead leads to bad designs
  - - Programmer codes in two models, don’t mix well

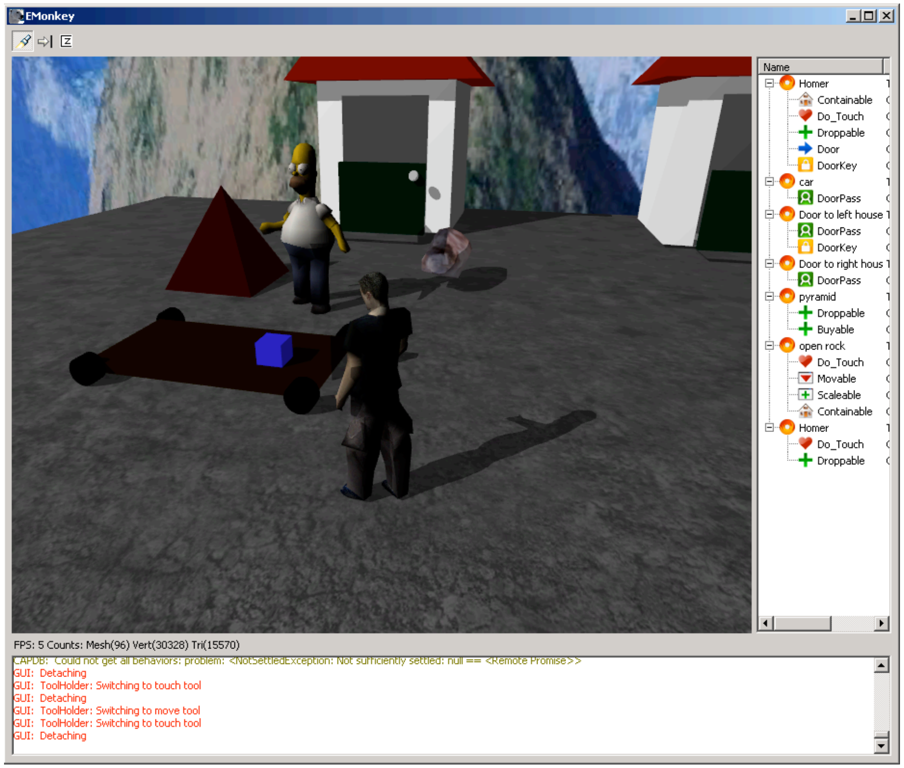
Sandboxed Virtual Machine

- VM-grain
- Alternative Libraries
  - Java Isolates?
  - + Technically sound: OS-like isolation
  - - Maintaining a forked version
  - - Difficult deployment demands

Need hostile environment

- Clean languages are more secureable.
  - Scheme, ML, Pict
- Academics too friendly, so no adoption.

- Virtual Realities
  - EC Habitats, Den, eMonkey
  - Croquet?
- Web/App Server
  - Waterken/Joe-E
- Javascript in web pages
  - ADsafe, FBJS, Caja×6



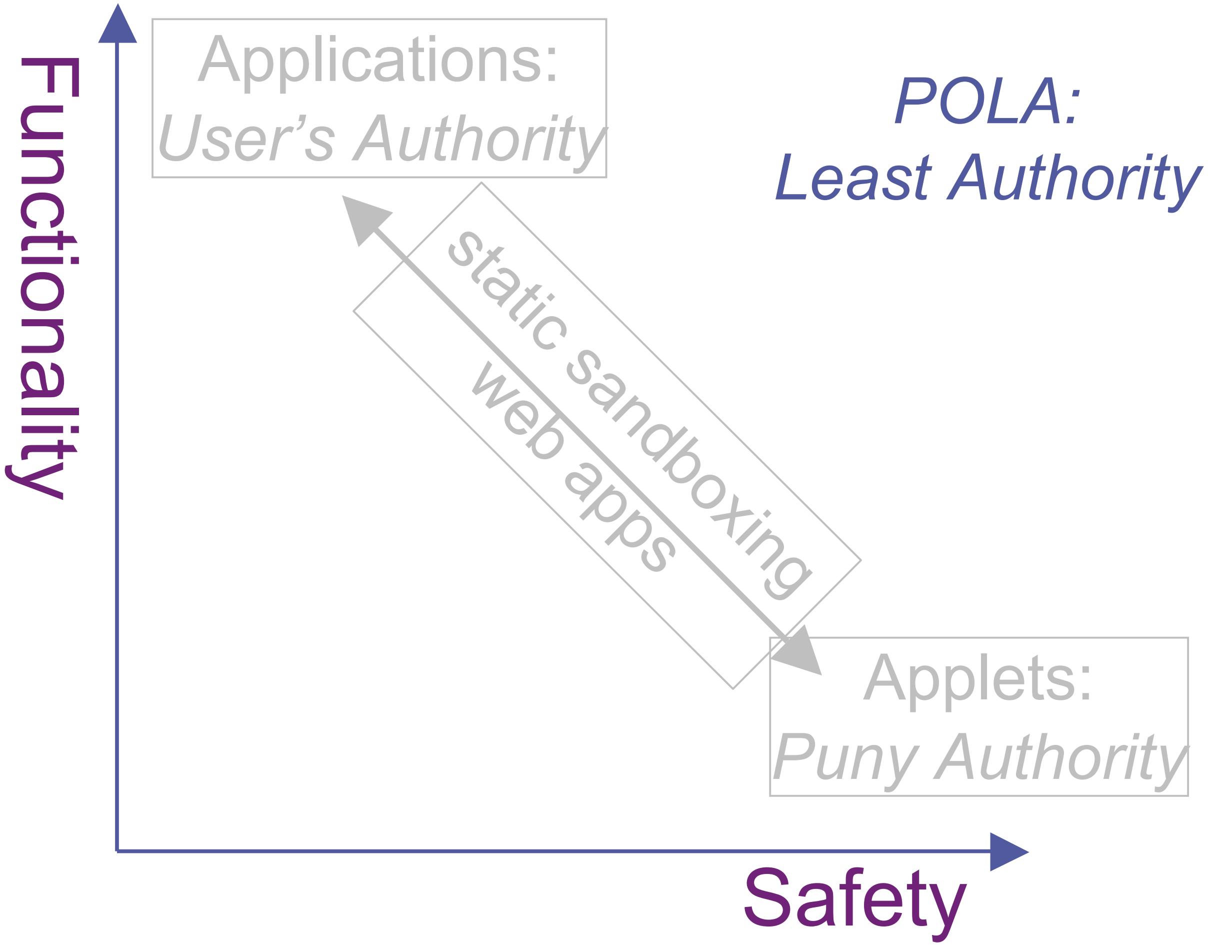
```
FPS: 5 Counts: Mesh(96) Vert(30328) Tri(15570)
```

```
CAPDB:  Could not get all behaviors: problem: <NotSettledException: Not sufficiently settled: null == <Remote Promise>>
GUI:  Detaching
GUI:  ToolHolder: Switching to touch tool
GUI:  Detaching
GUI:  ToolHolder: Switching to move tool
GUI:  ToolHolder: Switching to touch tool
GUI:  Detaching
```

Language design by subsetting

- Design to change the world
  - New language -> no adoption
- Languages already too large
  - “Extra” features destroy useful formal properties
- Insiders can’t subtract. Outsiders can’t add.
  - Old code *vs.* old tools: contravariant compatibility
- Discover the simple language struggling to get out.

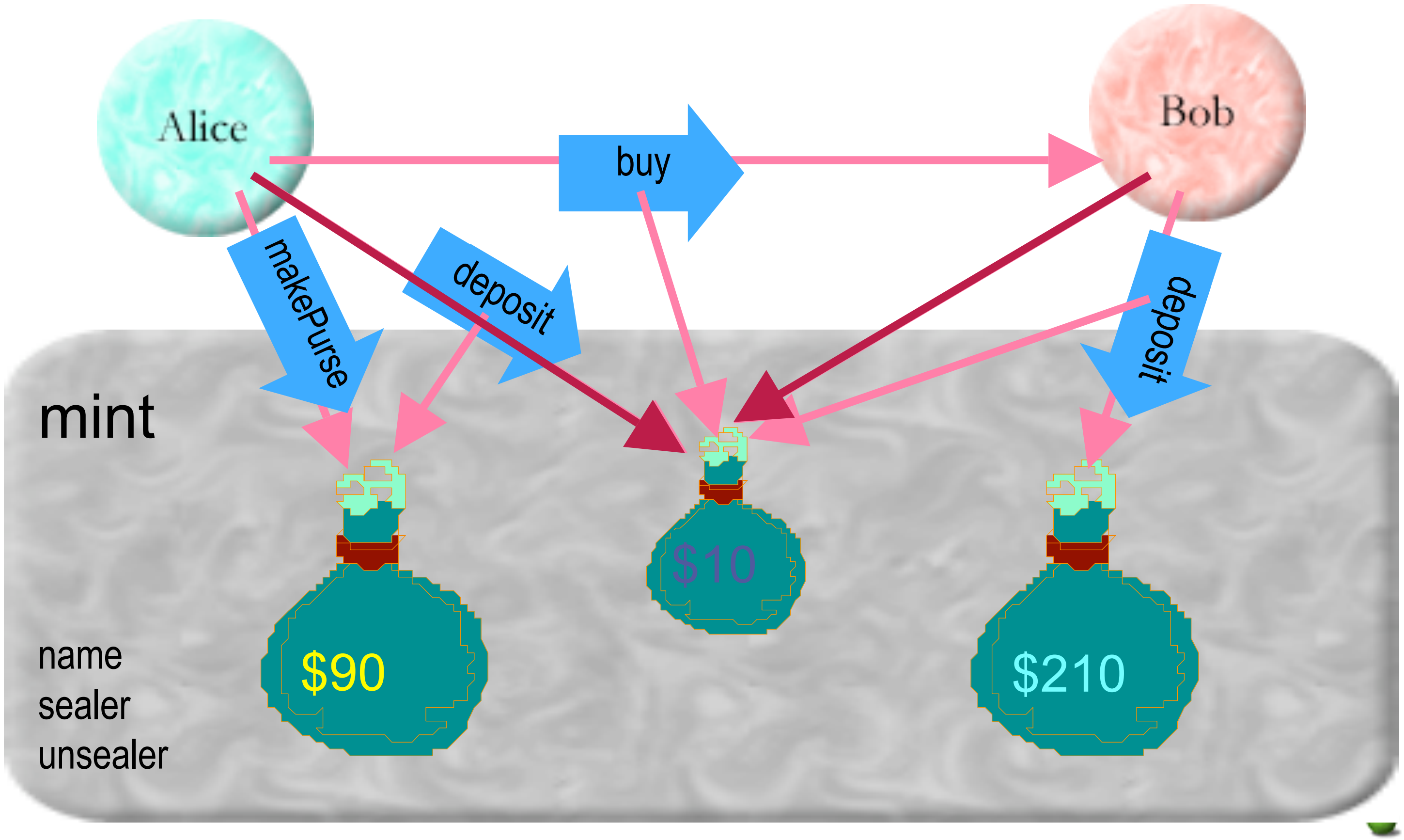
Stop Malware with OO Security



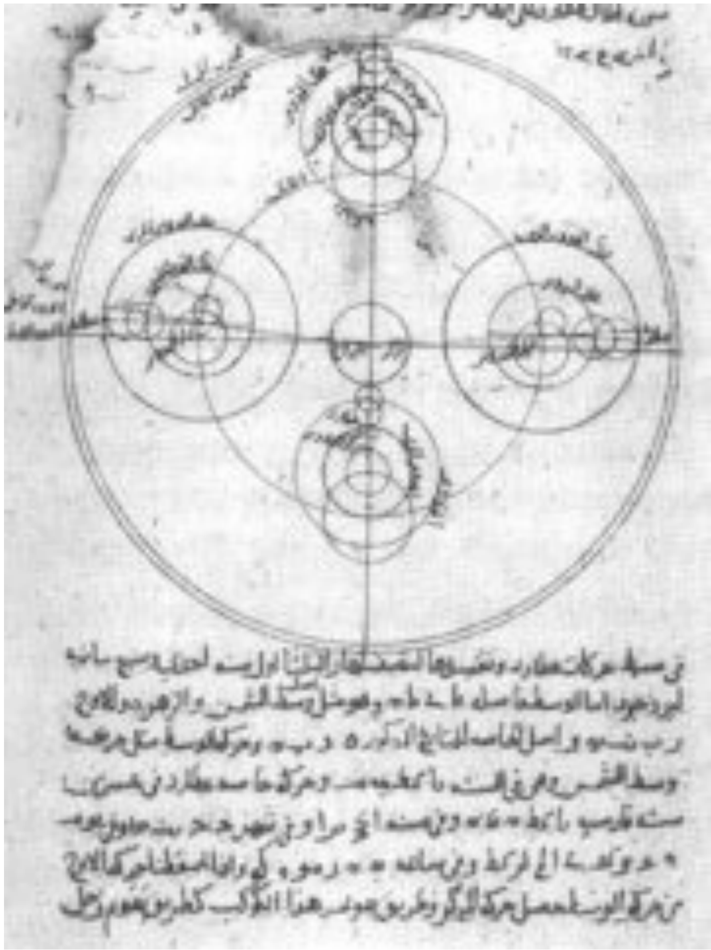
Alice pays Bob

```
var payment = myPurse.makePurse();
payment.deposit(10,myPurse);
bob.buy(..., payment);
```

```
Q.when(payment, function() {
   Q.when(myPurse.deposit(10,payment), function() {
      ... # dispense value});});
```



ACL Epicycles



New Languages

- Object-grain
- port programmers, not programs
    - **Algol 60 -> Gedanken**
    - Prolog+Actors -> FCP, Vulcan
    - -> Joule, Toontalk
    - Java -> E
    - C# -> Sebyla
    - ?? -> Eden, Emerald

Statically verified subset

- Object-grain
- No rewrite
- Static library taming
  - Javascript -> JSON (like S-expression)
  - Pict -> Backwater
  - OCaml -> Emily
  - Python -> Pthin (like Pascal)
  - **Java -> Joe-E**
  - Javascript -> ADsafe (blacklisting)
  - Java -> Original-E

Dynamic restrictions, rewriting

- Object-grain
- Dynamically substituted scope, rewriting
- Virtualized Libraries
  - **Scheme -> W7**
  - Mozart/Oz -> Oz-E
  - Perl -> CaPerl
  - Javascript -> Wrapperless Caja^{x}3 (FBJS?)
    - 1) blacklisting, 2) property name lifting, **3) Caja with whitelisting flags**
  - Smalltalk -> Squeak-E
  - ~~CommonLisp -> CL-E~~

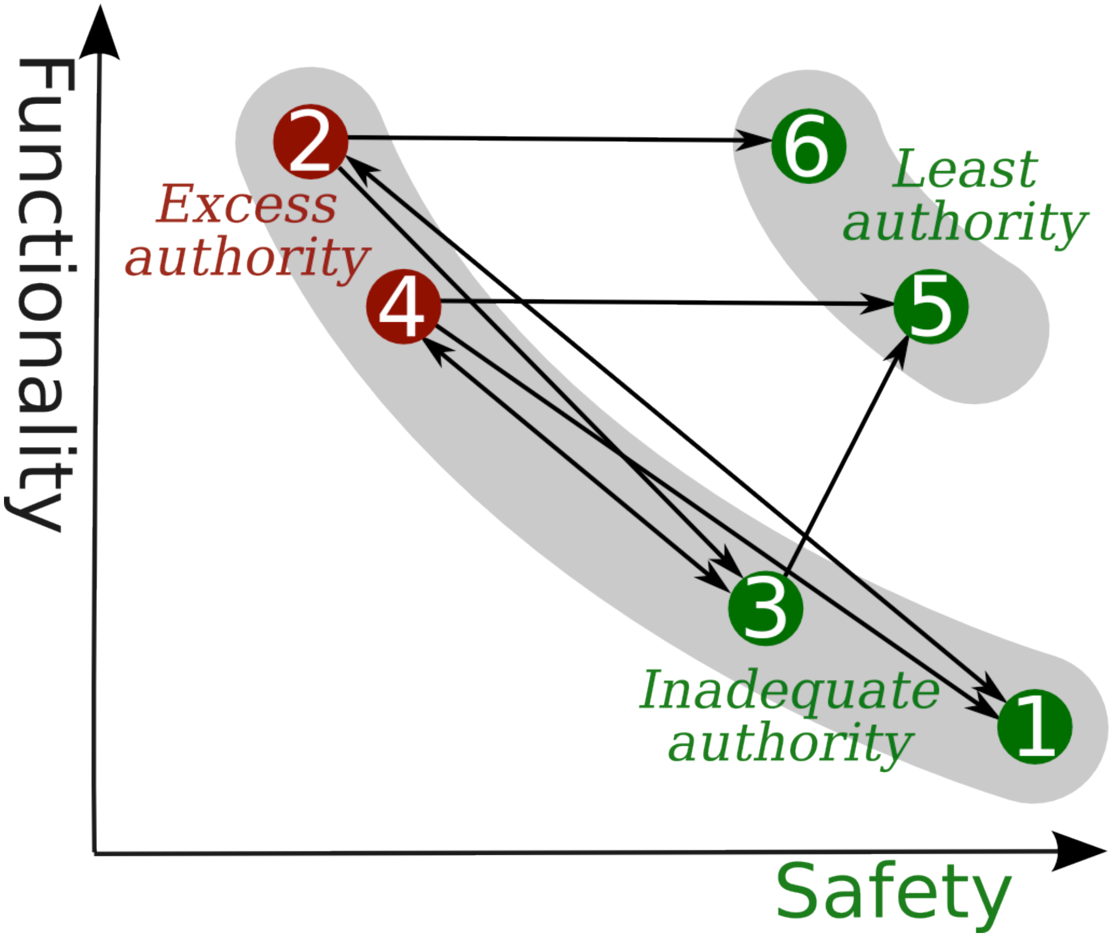
Wrapper-based Isolation

- Component-grain
- Synchronous membrane/wrappers
- Virtualized Libraries, Rewriter?
    - **Java(1.1) -> J-Kernel (ClassLoader tricks + RMI)**
    - Javascript -> Wrapper-based Caja^{x}2
        - 1) Asymmetric suspicion
        - 2) Mutual suspicion
    - Smalltalk -> Lex Spoon's Islands

Sandboxed Virtual Machine

- Vat-grain
- Modified VM, Async wrappers
- Alternative Libraries
  - Erlang -> Erly
  - **Java -> Java Isolates**
  - Javascript -> Vats on Gears Workers
  - Python -> Brett Canon’s “Secure Python”
  - Smalltalk -> Tweak Islands

Escape the Dilemma



Design enforceable language subsets

- "You can't start over again"
- "You can't add security later"
- Don't add security, remove insecurity

- Vendors can only grow their language
  - Non-vendors can only shrink it
  - Old tools *vs.* old code: contravariant compatibility