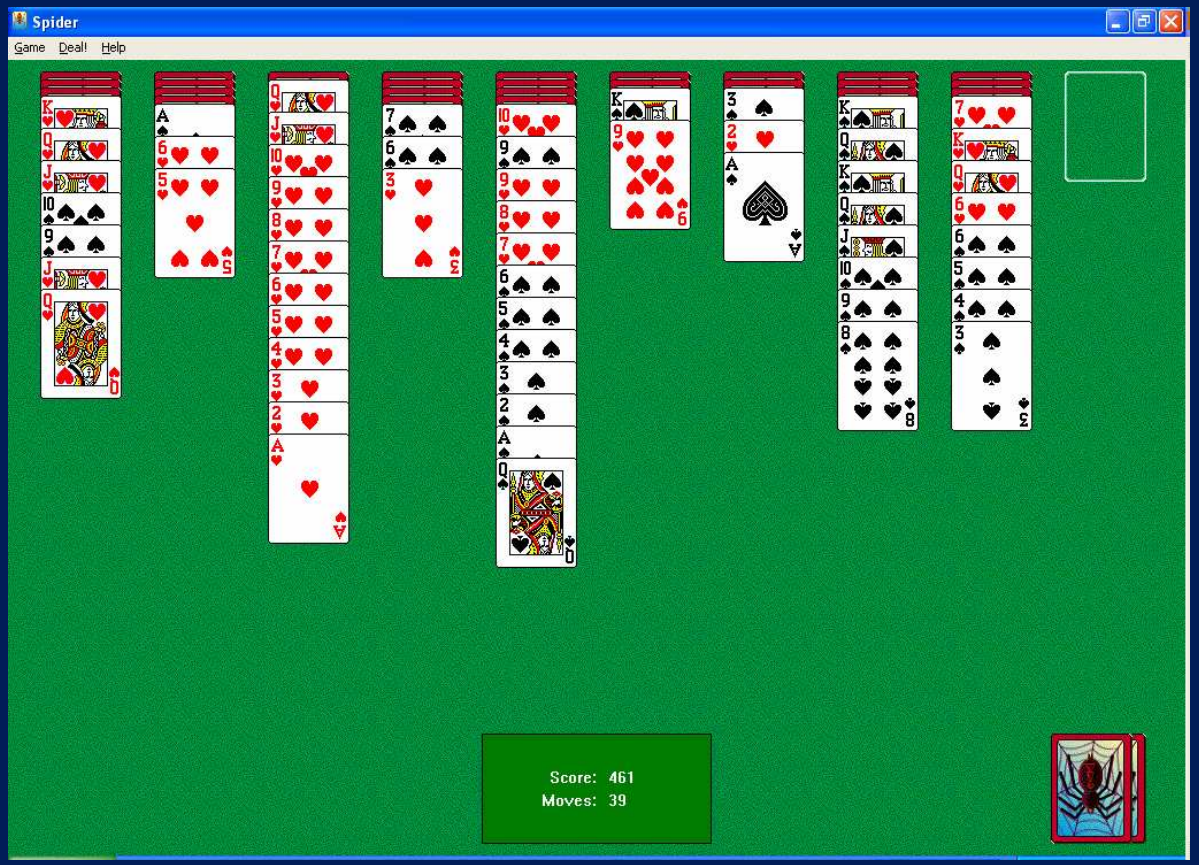
Building a Virus-Safe Platform

Don't Add Security, Remove Insecurity

Mark S. Miller
Virus-Safe Computing Initiative
Hewlett Packard Laboratories

A Very Powerful Program



This program can delete any file you can.

Functionality *vs.* Security?

Integratable

Applications:
User's Authority

"Sandboxing"

Firewalls

Applets:
No Authority

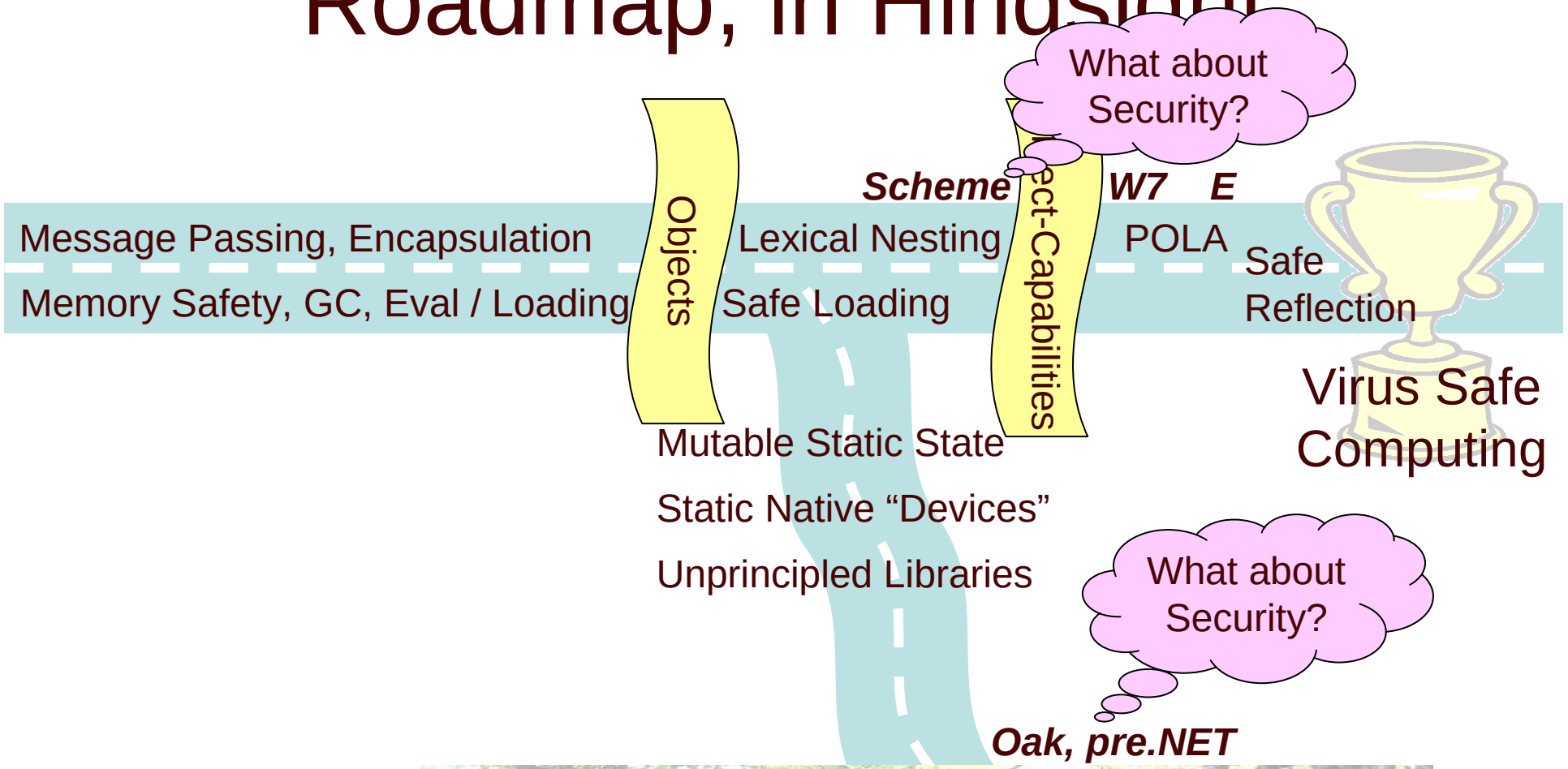
Isolated

Dangerous

Safe

- ***E*** & CapDesk
  Least Authority

Roadmap, in Hindsight



The Access Matrix

Who might endanger what?

ACLs and Caps, equivalent so far…

Knowledge Shapes Access

"Make the Computer Recursive"
—Alan Kay

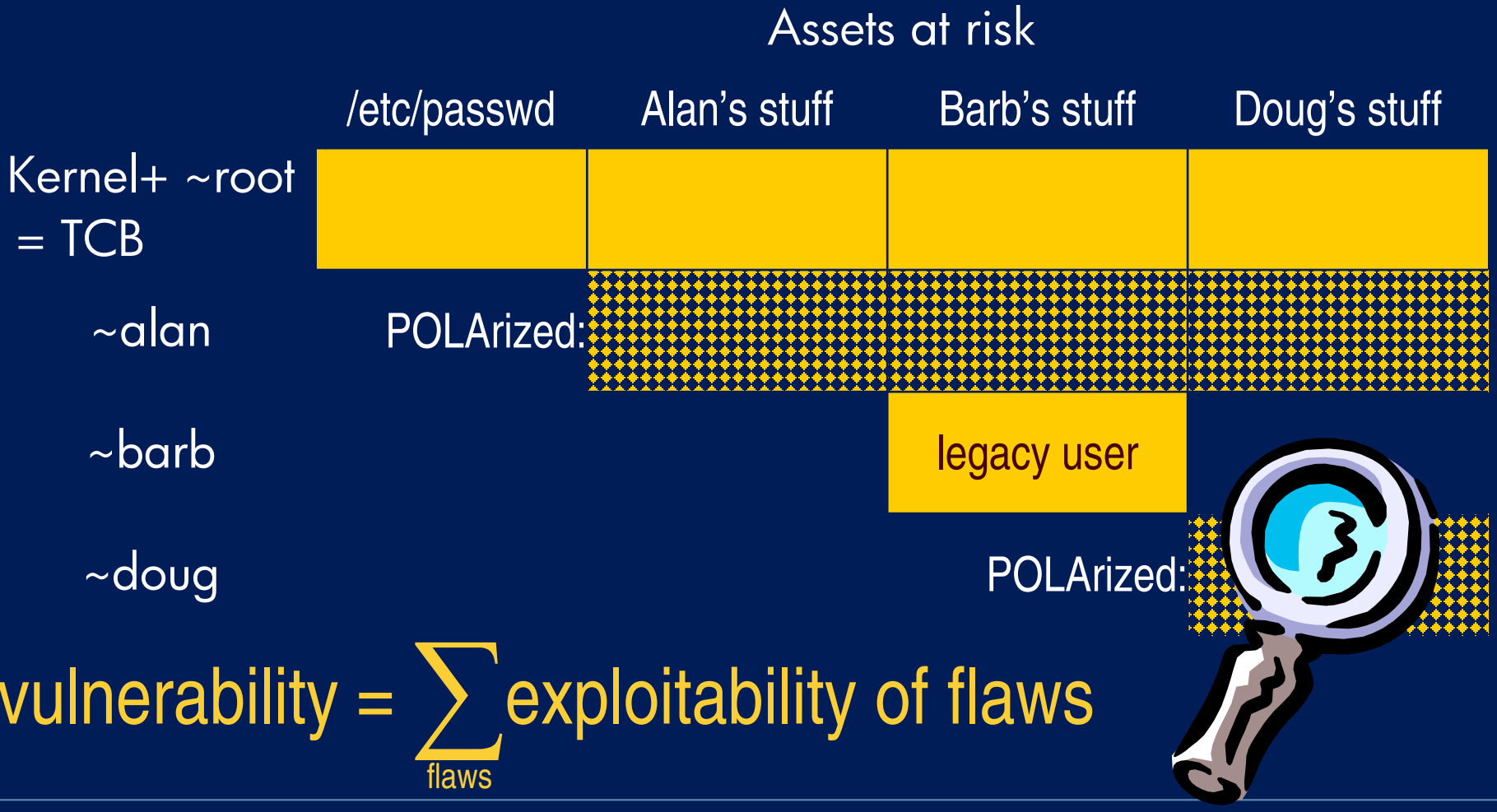
- *"Knows about"* already has a fractal structure.
  - Stepwise refinement. Simon's watchmakers.
  - People know people. Organs know organs. Cells know cells.

- Information hiding: *"Need to know"*
- POLA (Principle of Least Authority): *"Need to do"*

- Let locality of knowledge shape localization of authority
  - Dynamics of "knows about" from acts of designation
  - Provide "access to" when providing "knowledge of"
  - Make access rights similarly self-similar!

The Access Matrix, Reloaded

Who might endanger what?



$$\text{vulnerability} = \sum_{\text{flaws}} \text{exploitability of flaws}$$

The Access Matrix Revolution

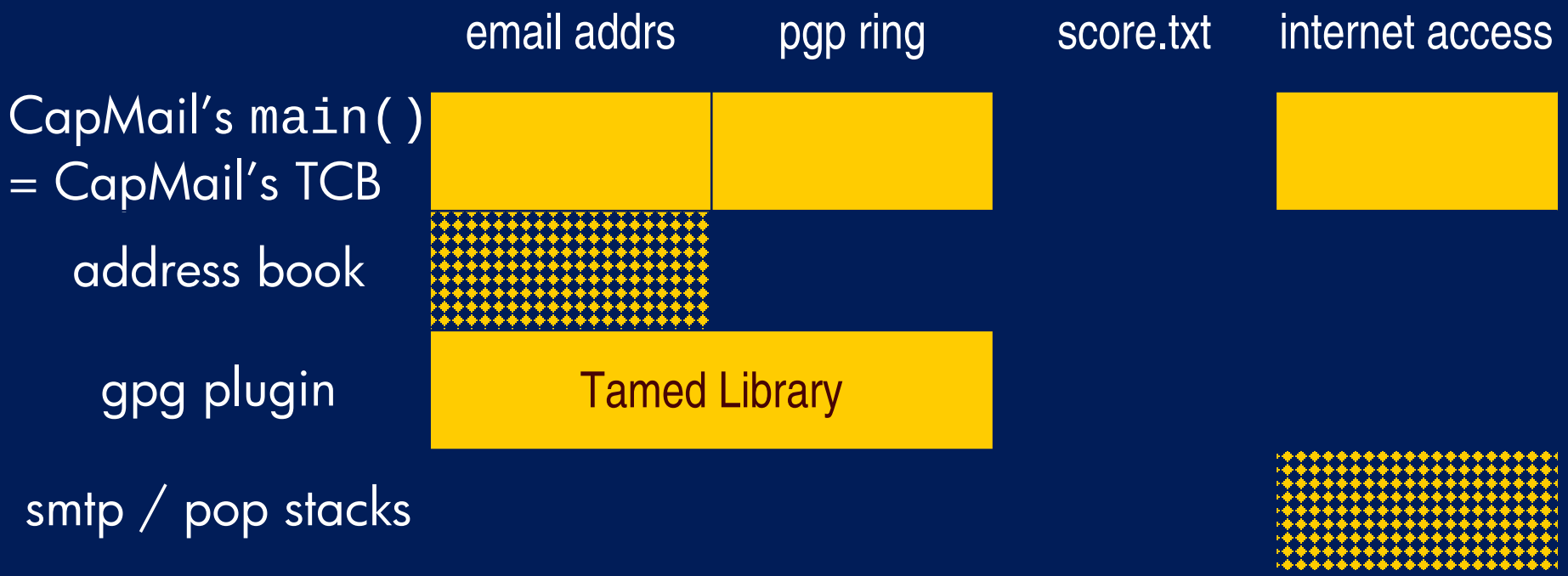
What might endanger what?

Doug's assets at risk "to Doug"

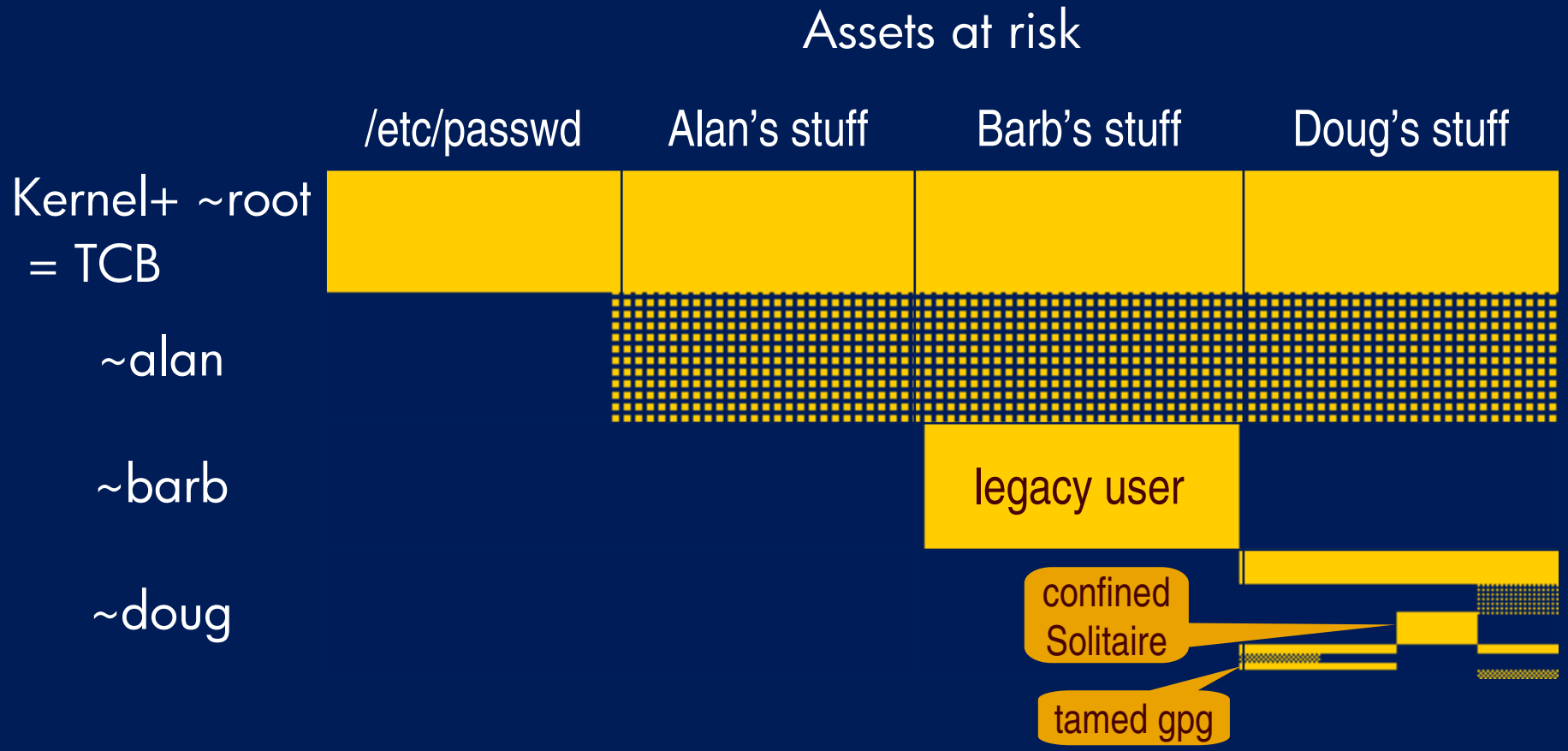
	email addrs	pgp ring	score.txt	internet access
E + CapDesk = Doug's TCB				
DarpaBrowser				
Solitaire			Machine virtualization	
CapMail				

Subsystems within Subsystems

Doug's assets at risk to his CapMail



Least Authority is Fractal*!*



"Need to do" parallels *"need to know"*, recursively reducing area of vulnerability

A Tale of Two Copies

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

vs.

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

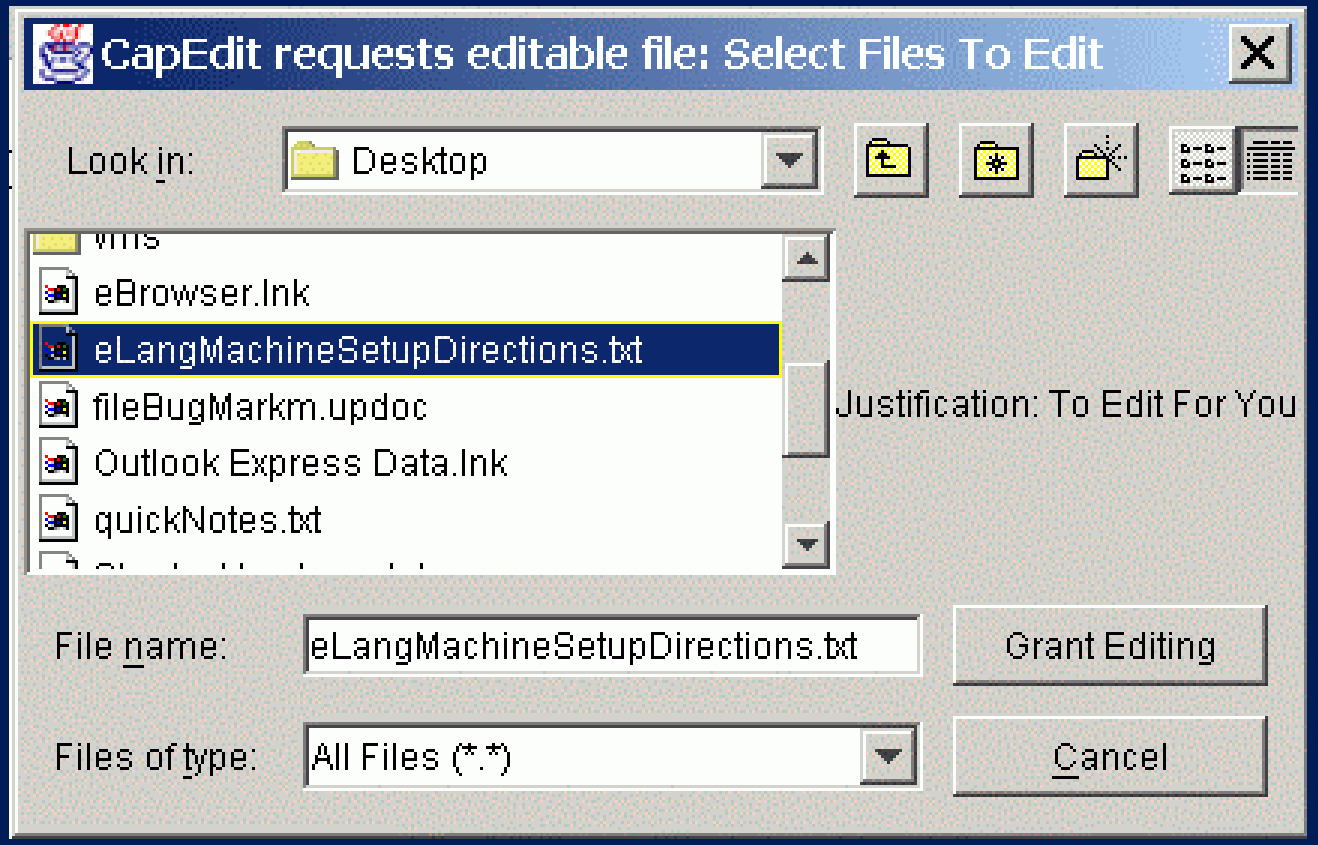
"Knowledge of" should shape *"access to"*

Moral:

Bundle permission with designation

CapDesk: Usable Security

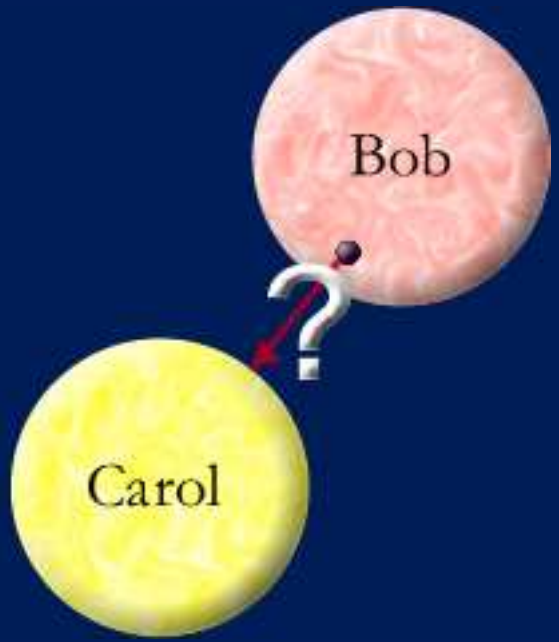
- Double click launch
- File dialog
- Drag/Drop
- Etc…



Moral:

Bundle permission with designation

How do I designate thee?

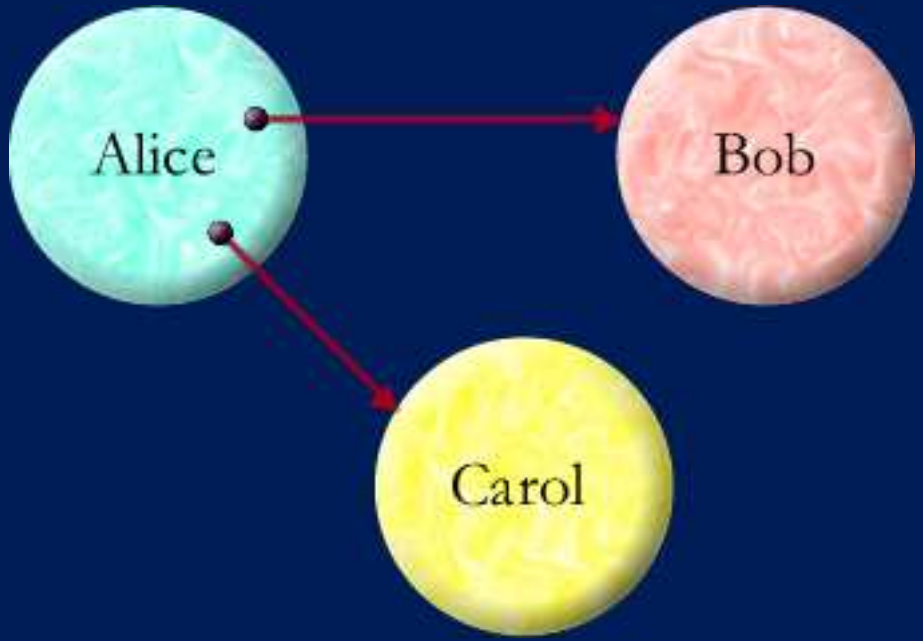


- by Introduction
  - ref to Carol
  - ref to Bob
  - decides to share
- by Parenthood
- by Endowment
- by Initial Conditions

How might object Bob come to know of object Carol?

How do I designate thee?

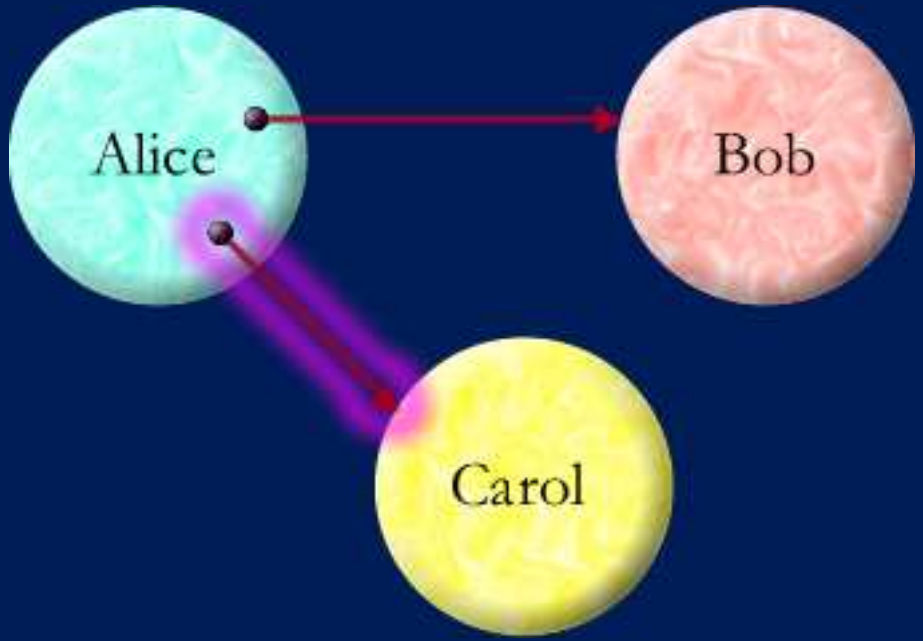
Alice says: bob.foo(carol)



- ***by Introduction***
  - ref to Carol
  - ref to Bob
  - decides to share
- by Parenthood
- by Endowment
- by Initial Conditions

How do I designate thee?

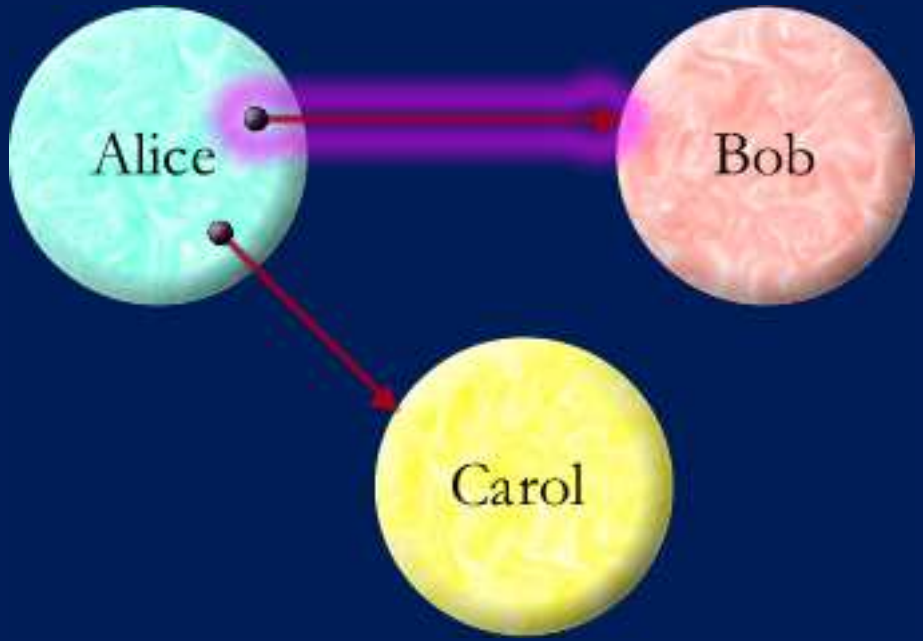
Alice says: bob.foo(carol)



- *by Introduction*
  - ***ref to Carol***
  - ref to Bob
  - decides to share
- by Parenthood
- by Endowment
- by Initial Conditions

How do I designate thee?

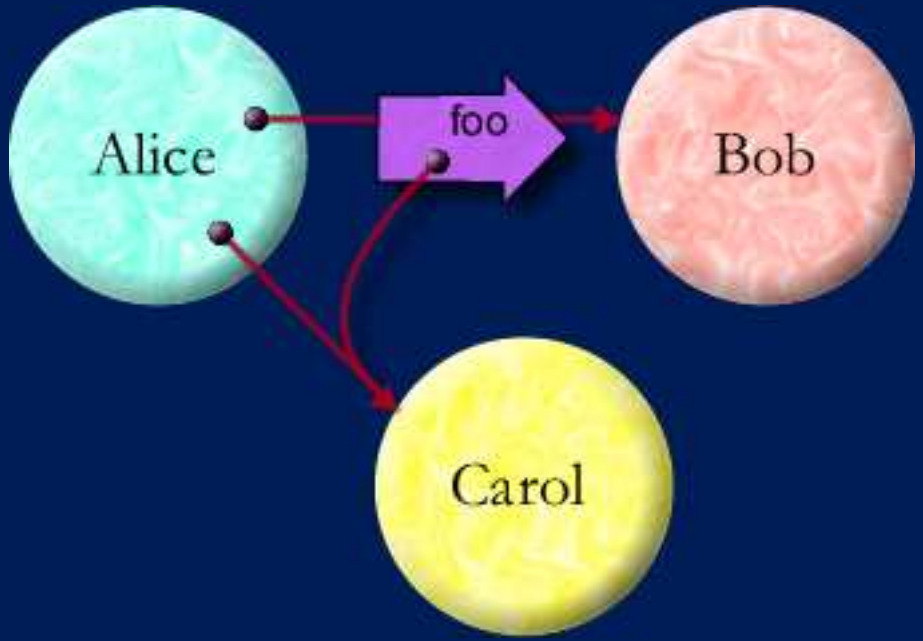
Alice says: bob.foo(carol)



- *by Introduction*
  - *ref to Carol*
  - ***ref to Bob***
  - decides to share
- by Parenthood
- by Endowment
- by Initial Conditions

How do I designate thee?

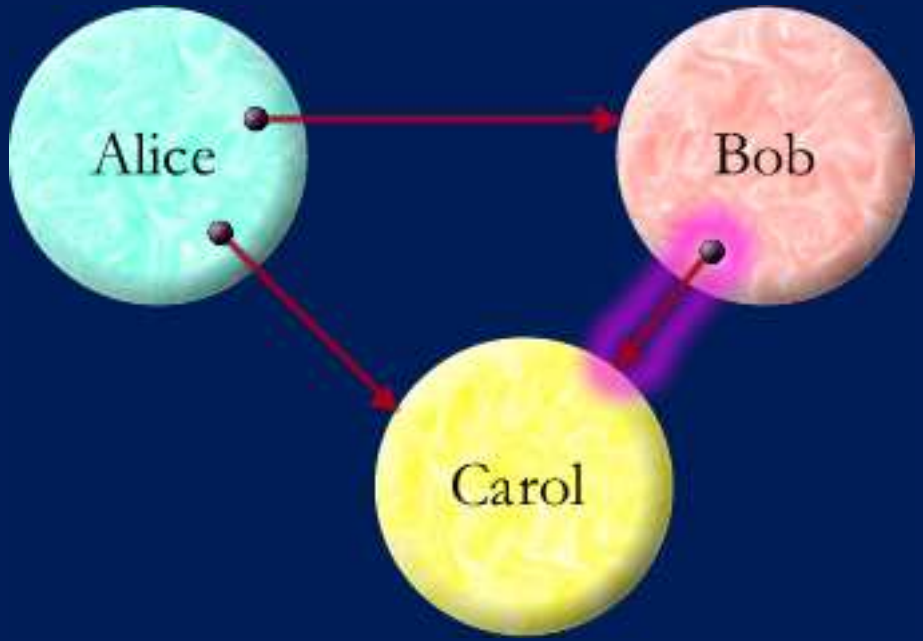
Alice says: bob.foo(carol)



- *by Introduction*
  - *ref to Carol*
  - *ref to Bob*
  - ***decides to share***
- by Parenthood
- by Endowment
- by Initial Conditions

How do I designate thee?

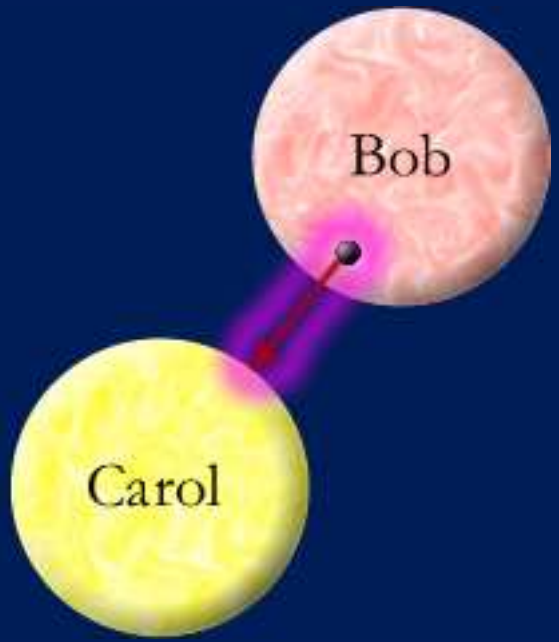
Alice says: bob.foo(carol)



- *by Introduction*
  - *ref to Carol*
  - *ref to Bob*
  - *decides to share*
- by Parenthood
- by Endowment
- by Initial Conditions

How do I designate thee?

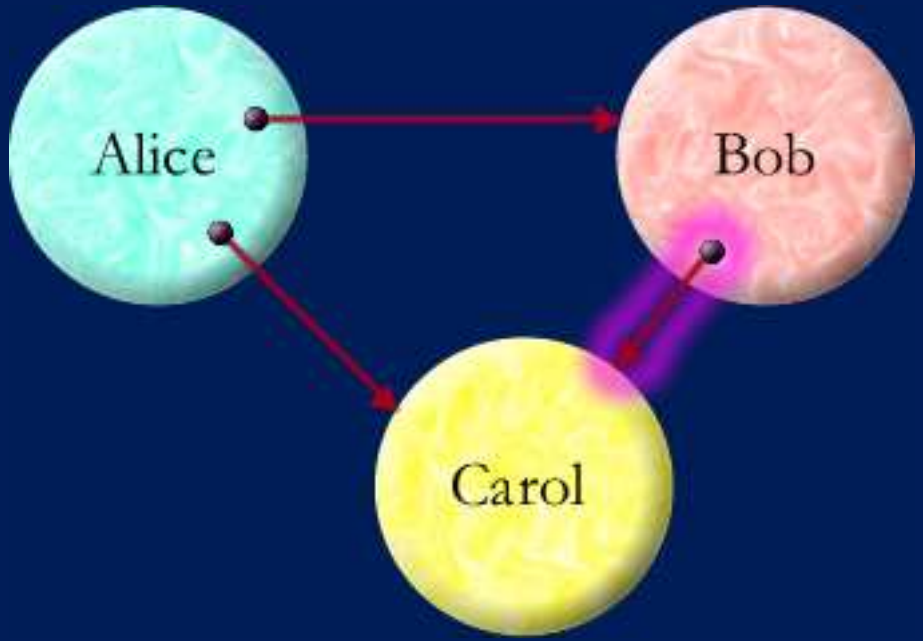
Bob says: def ***carol*** { ... }



- by Introduction
  - ref to Carol
  - ref to Bob
  - decides to share
- ***by Parenthood***
- by Endowment
- by Initial Conditions

How do I designate thee?

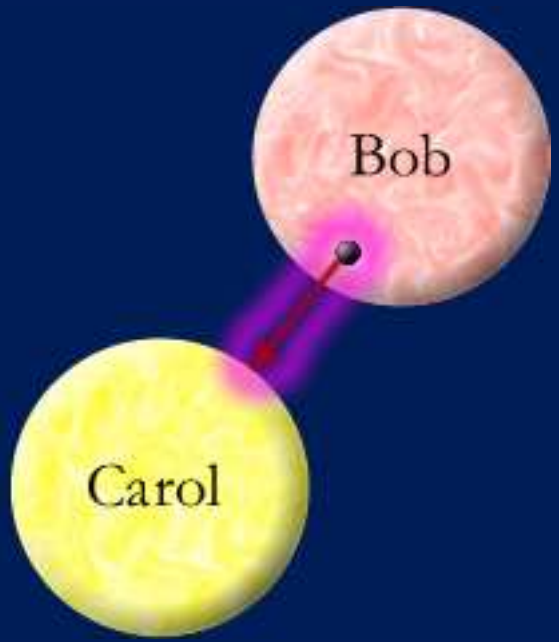
Alice says: def ***bob*** { ... carol ... }



- by Introduction
  - ref to Carol
  - ref to Bob
  - decides to share
- by Parenthood
- ***by Endowment***
- by Initial Conditions

How do I designate thee?

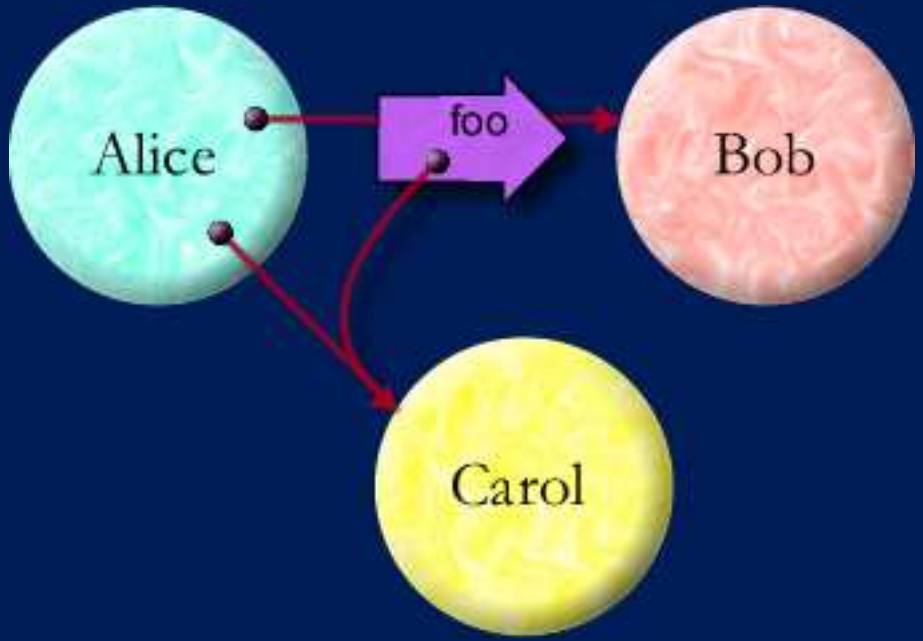
At t_0:



- by Introduction
  - ref to Carol
  - ref to Bob
  - decides to share
- by Parenthood
- by Endowment
- ***by Initial Conditions***

What are Object-Capabilities?

Reference Graph == Access Graph



- by Introduction
  - ref to Carol
  - ref to Bob
  - decides to share
- by Parenthood
- by Endowment
- by Initial Conditions

- Absolute encapsulation—causality ***only*** by messages
- ***Only*** references permit causality

Security is Just Extreme Modularity

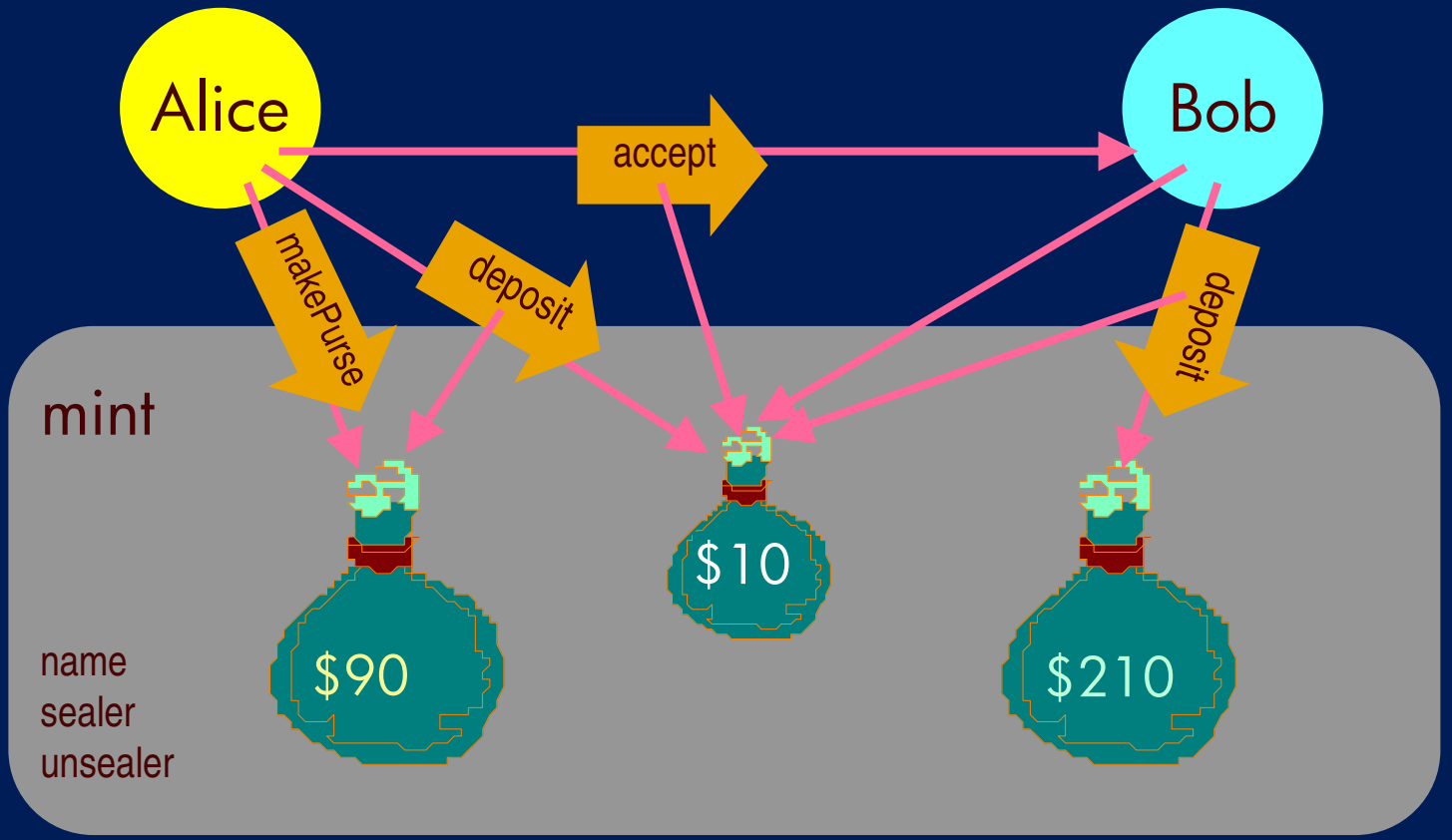
- Good software engineering
  - Responsibility driven design
  - Omit needless coupling
  - `assert(..)` preconditions
- Information hiding
  - Designation, need to know
  - Dynamics of knowledge
- Lexical naming
  - Think names, speak refs
  - Avoid global variables
- Abstraction
  - Procedural, data, control, …
  - Patterns and frameworks
  - Say what you mean

- Capability discipline
  - Authority driven design
  - Omit needless vulnerability
  - Validate inputs
- Principle of Least Authority
  - Permission, need to do
  - Dynamics of authorization
- No global name spaces
  - Think names, speak refs
  - Forbid mutable static state
- Abstraction
  - … and access abstractions
  - Patterns of safe cooperation
  - Mean only what you say

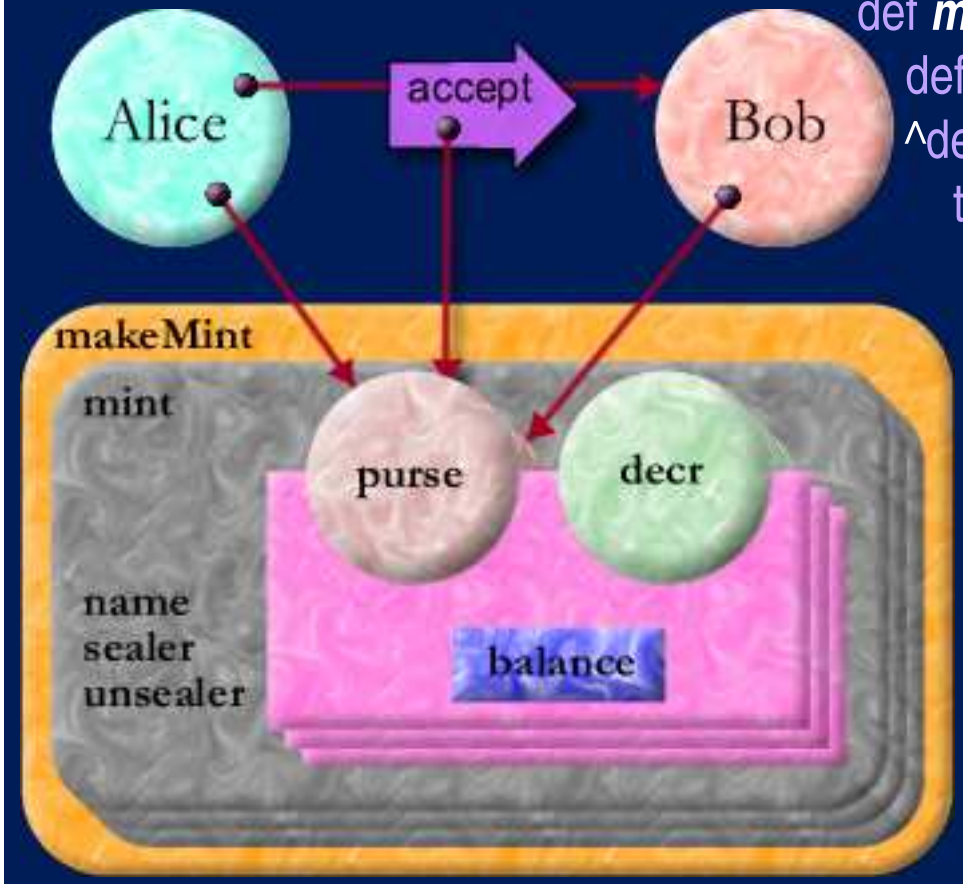
Cashing in on Distributed Objects

```
def payment := myPurse <- makePurse()
payment <- deposit(10, myPurse)
bob <- accept(payment)
```

```
when (myPurse <- deposit(10, payment)) ... {
    ...
}
```

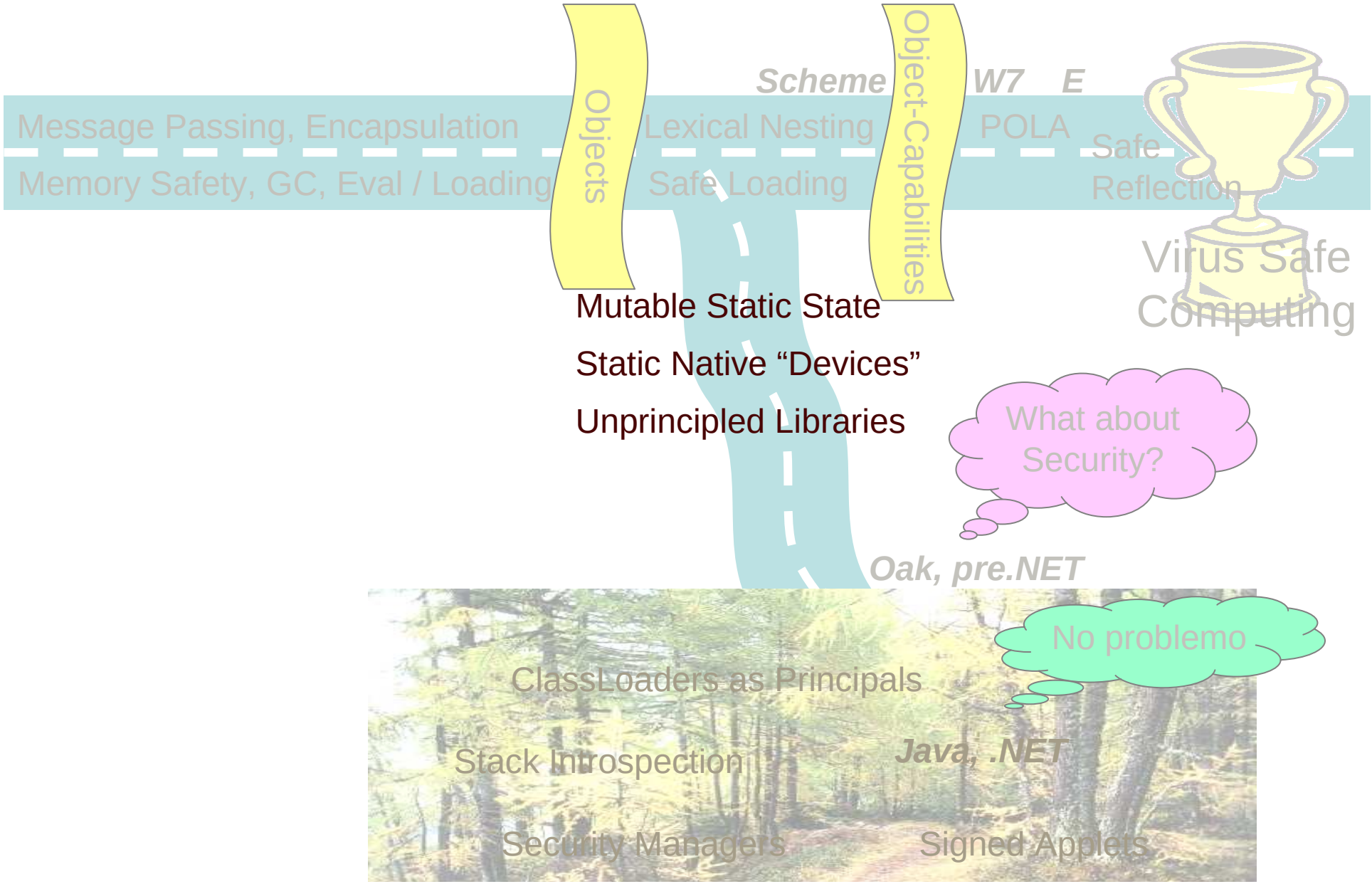


Distributed Secure Money in *E*




```
def makeMint(name :String) :any {
  def [sealer, unsealer] := makeBrandPair(name)
  ^def mint {
    to makePurse(var balance :(int >= 0)) :any {
      def decr(amount :(0..balance)) :void {
        balance -= amount
      }
      ^def purse {
        to getBalance() :int   { ^balance }
        to makePurse() :any { ^mint.makePurse(0) }
        to getDecr()       :any { ^sealer.seal(decr) }
        to deposit(amount :int, src) :void {
          unsealer.unseal(src.getDecr())(amount)
          balance += amount
} } } } }
```

Detour is *Non-Object Causality*



Transcend the Arms Race

- Generals and virus scanners fight the last war
  - Time to exploit dropping to hours
  - Time to epidemic measured in minutes
- Firewalls fight the enemy
  - But what about your “friends”?
- Two fronts
  - Reduce number of exploitable flaws – QA, Patching
  - Reduce exploitability of inevitable flaws – POLA
- “The best way to solve a problem is to avoid it.”
  - Give subsystems only the authority they need, recursively
  - Limits the damage a virus can do
- Make virus writing uninteresting

Our Logo

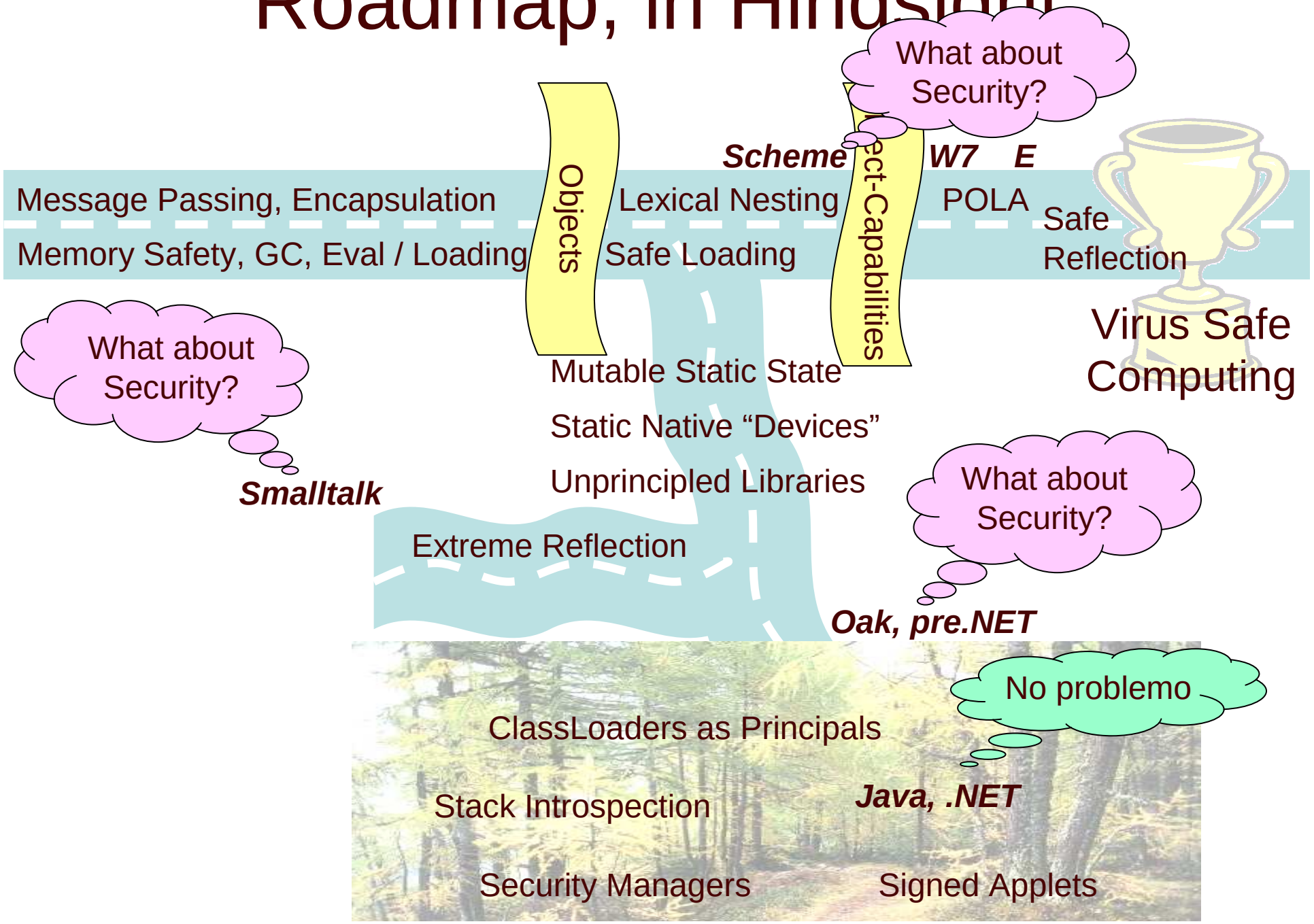
The POLA Bear

Bibliography

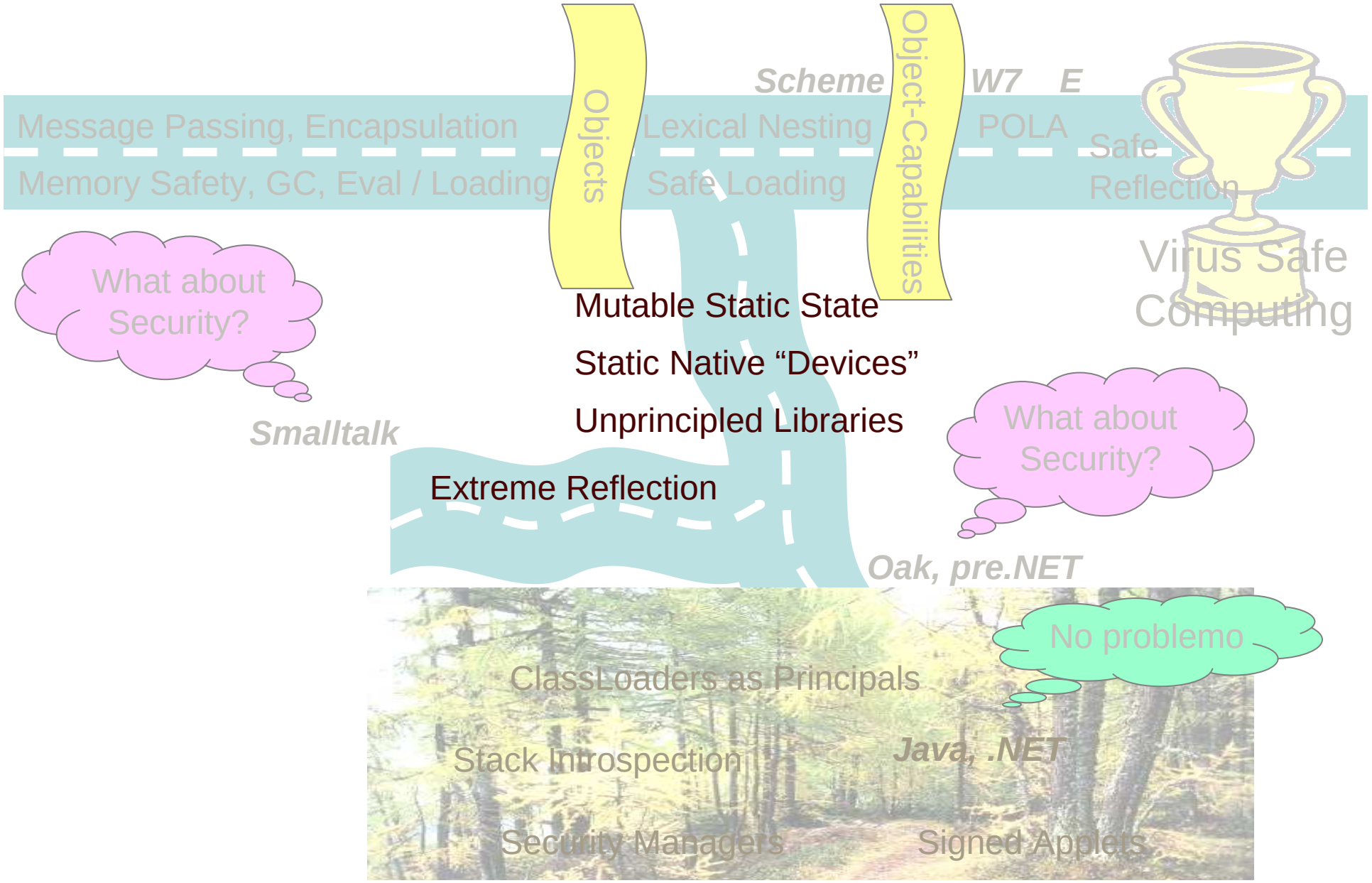
- ***E*** in a Walnut skyhunter.com/marcs/ewalnut.html
  Download ***E*** *from erights.org and try it!* (It's open source.)
- Paradigm Regained (HPL-2003-222) erights.org/talks/asian03/
- A Security Kernel Based on the Lambda-Calculus mumble.net/jar/pubs/secureos/
- Capability-based Financial Instruments (the "Ode") erights.org/elib/capability/ode/index.html
- Intro to Capability-based Security skyhunter.com/marcs/capabilityIntro/index.html
- Statements of Consensus erights.org/elib/capability/consensus-9feb01.html
- Web Calculus www.waterken.com/dev/Web/Calculus/
- Web sites: erights.org , combex.com , eros-os.org , cap-lore.com/CapTheory , www.waterken.com


hp
invent

Roadmap, in Hindsight



Detour is *Non-Object Causality*



Example: Oak's Detour

- Mutable static state (class variables)
  - even when private, prevents confinement
- Static, native, authority-bearing methods
  - example: File opening, clock
- Ambient access to non-determinism
  - `System.identityHashcode(obj)`, threads
- Locks as communication channels
  - `synchronized ("foo".intern()) {...}`
- Non-POLA legacy libraries

Stay on the Pure Object Road

- Pure object (instance) model is fine as is
  - No features need be added or removed
  - Though some new primitives are convenient
- Non-object causality must be prohibited
  - Authority only according to references held & used
- Loading separately provided code and state
  - No implicit state bindings, no global scopes
  - Must support lexical nesting in the large
  - All free variables are virtualizable
  - Only `main()` starts with all authority, as instances