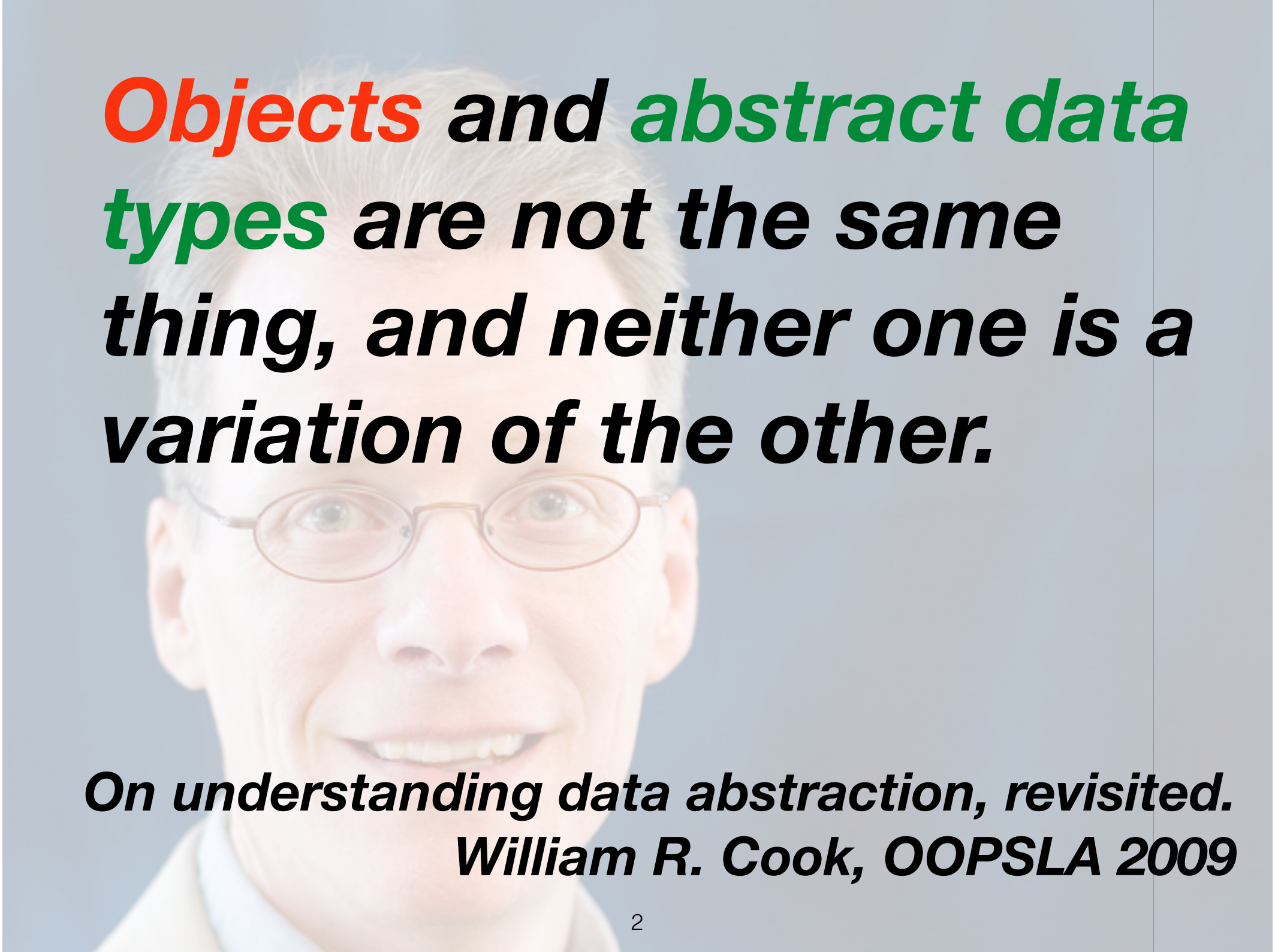
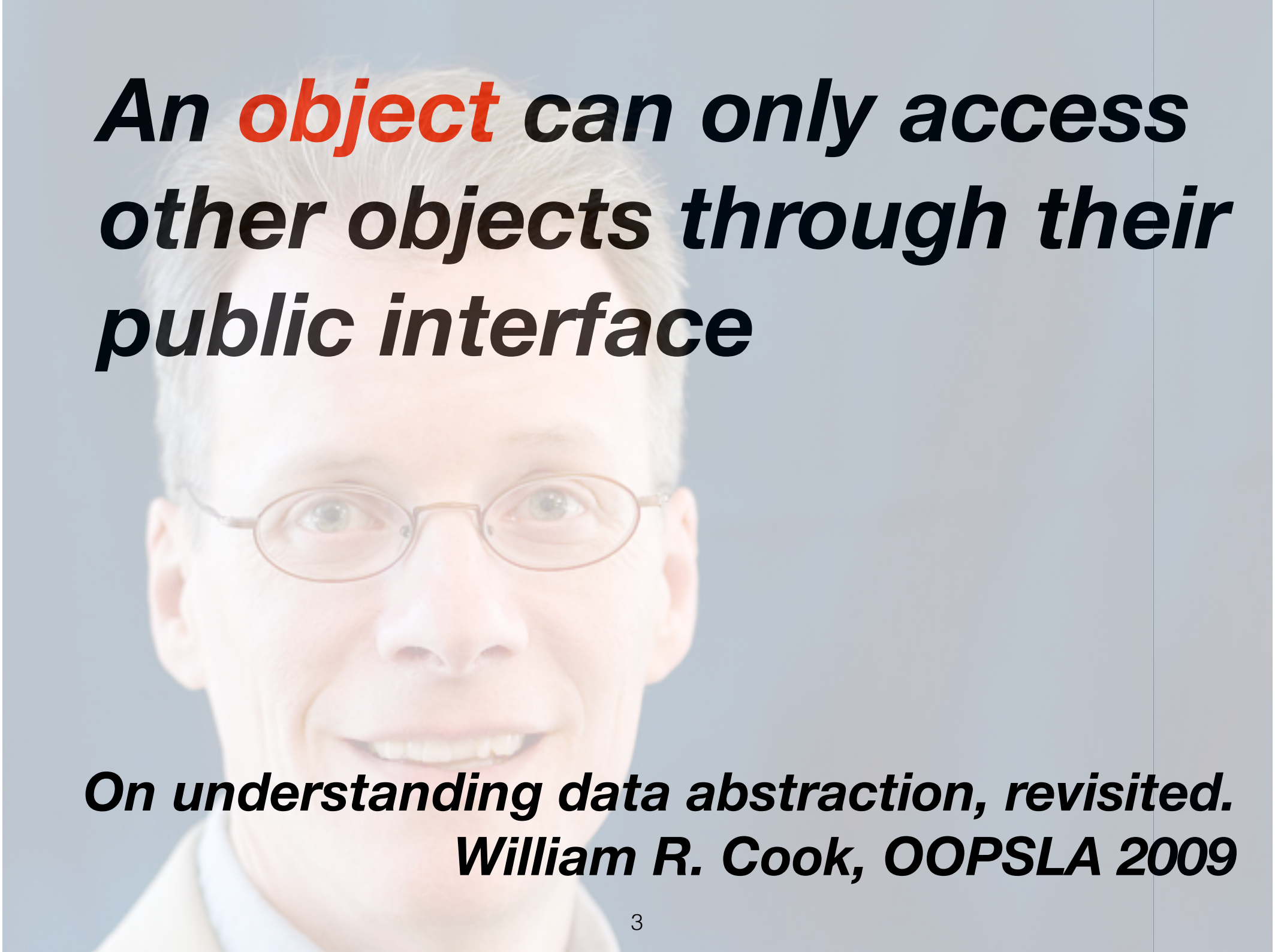
Abstract Data Types in Object Capability Systems

James Noble
Sophia Drossopoulou
Mark S. Miller
Toby Murray
Alex Potanin



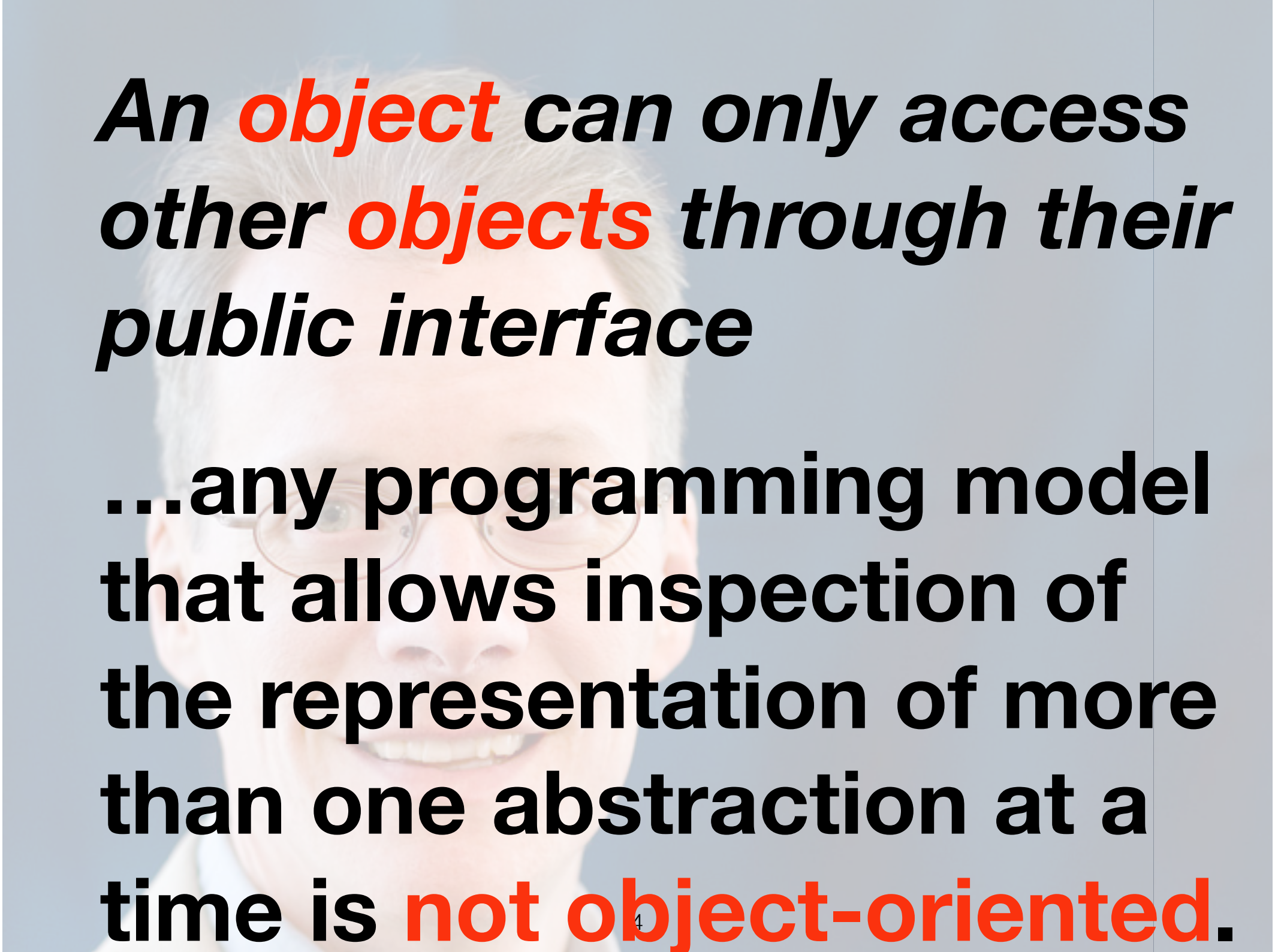
Objects and abstract data types are not the same thing, and neither one is a variation of the other.

On understanding data abstraction, revisited.
William R. Cook, OOPSLA 2009



An object can only access other objects through their public interface

On understanding data abstraction, revisited.
William R. Cook, OOPSLA 2009



An object can only access other objects through their public interface

…any programming model that allows inspection of the representation of more than one abstraction at a time is not object-oriented.

Object Capabilities

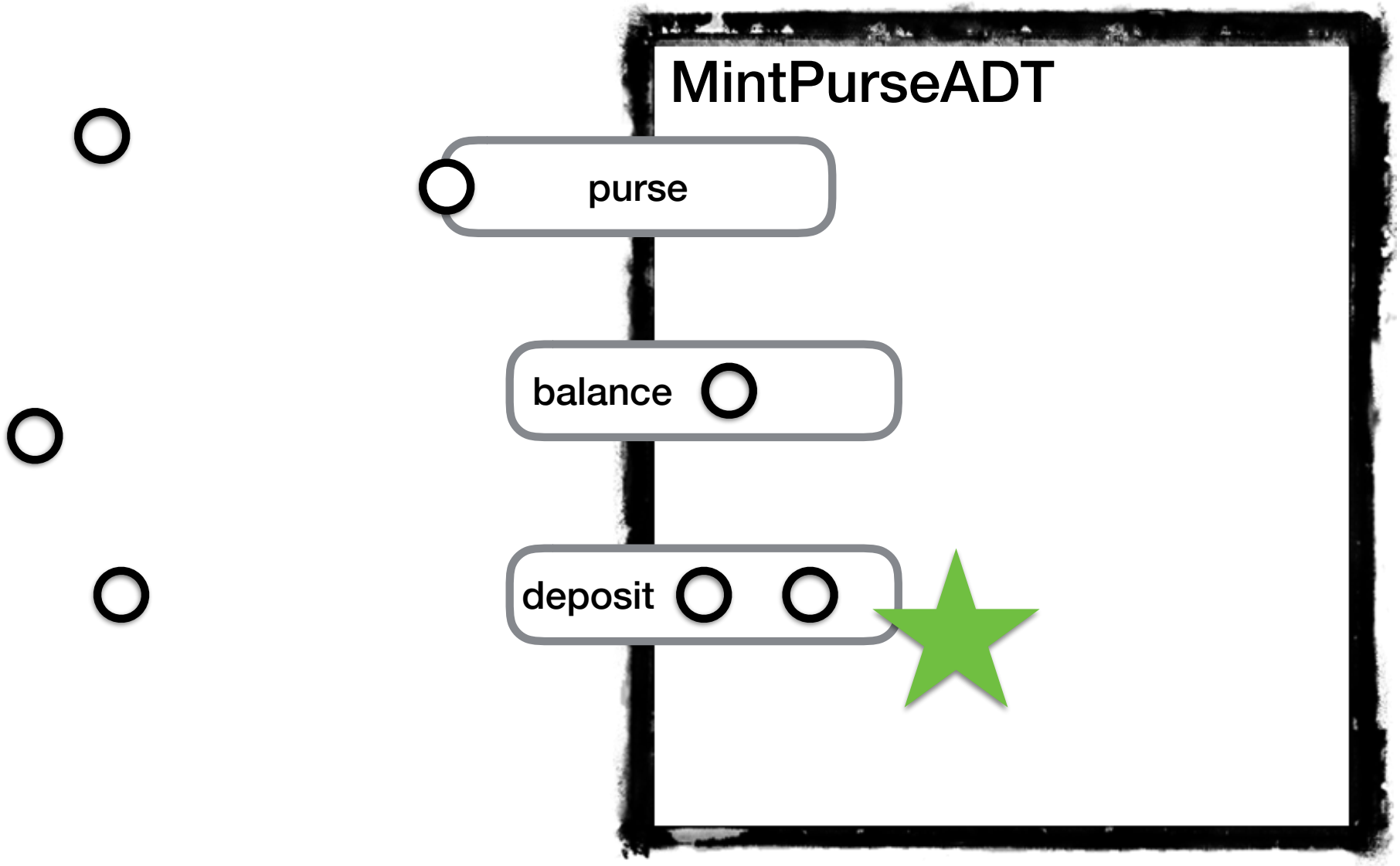
- **Only connectivity begets connectivity**
  - *"No ambient authority"*
- **Access implies permission**
  - *"No access control lists"*
- Dynamic Loading

Capability Based Financial Instruments ***Miller, Morningstar, Frantz.*** **Fin. Cryp. 2000**

Grodula-3

```
type MintPurseADT = interface {
  type T <: Object
  purse(amount : Number) -> T
  balance(of : T) -> Number
  deposit(to : T,
          amount : Number,
          from : T) -> Boolean
  sprout -> T
}
```

Grodula-3



Abstract Data Type

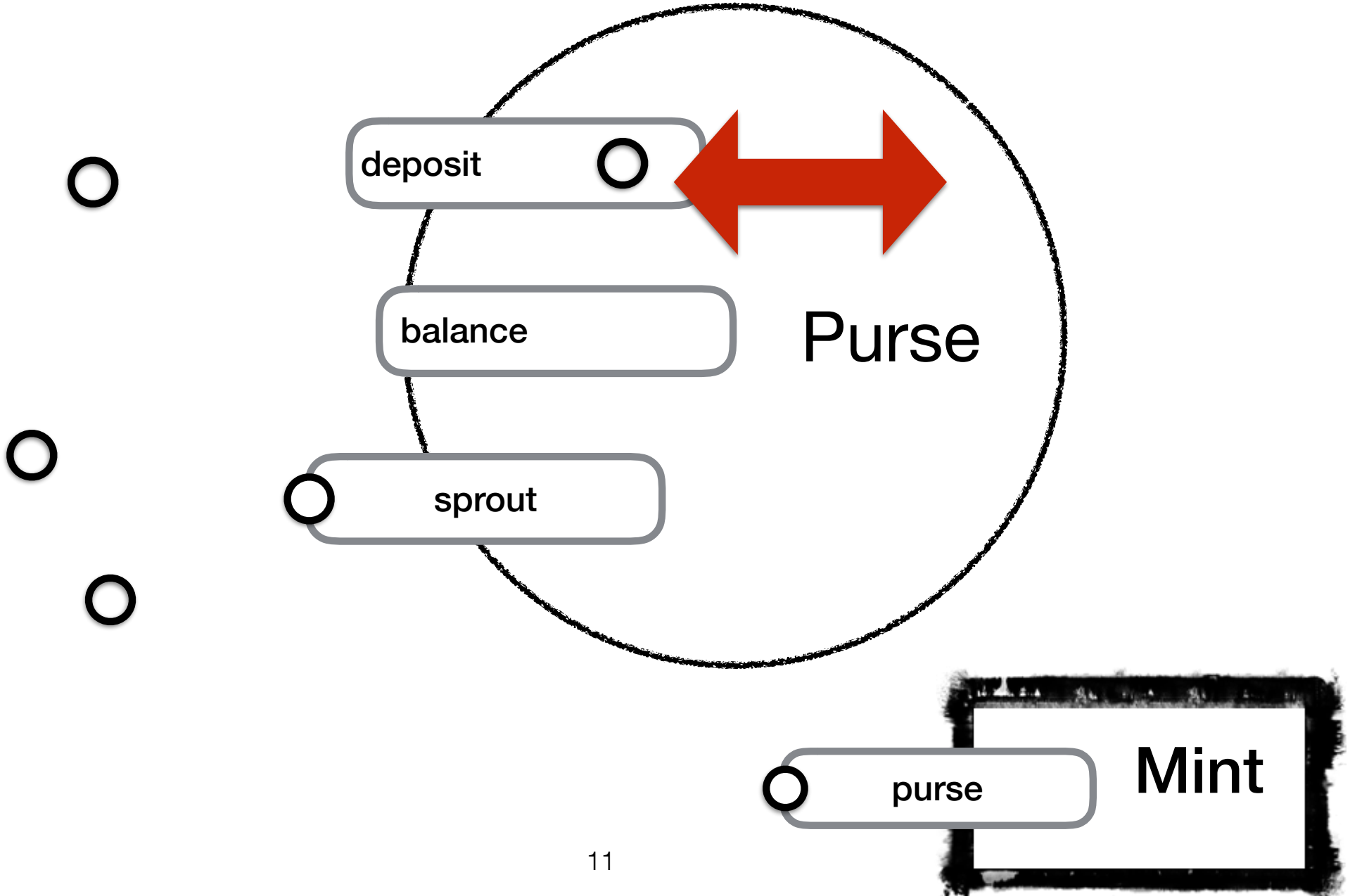
```
module MintPuse {
  reveal T = Purse

  type Purse is private = interface {
    balance -> Number
    balance:= ( n : Number )
  }

  method purse(amount : Number) -> T {
    return object { var balance is public := amount }

  method balance(of : Purse) -> Number { of.balance }
```

```
  method deposit(to : Purse,
                 amount : Number,
                 from : Purse) -> Boolean {
    if ((amount >= 0) && (from.balance >= amount))
      then {
        from.balance := from.balance - amount
        to.balance := to.balance + amount
        return true
      } else {return false}
  }



  method sprout { purse(0) }
}
```

“Classical” “OO” ADT

```
type Mint = interface {
  purse(amount : Number) -> Purse
}

type Purse = interface {
   balance -> Number
   deposit(amount : Number, src : Purse) -> Boolean
   sprout -> Purse
}
```

“Classical” “OO” ADT

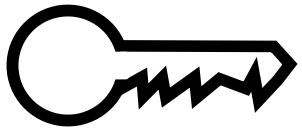


“Classical” “OO” ADT

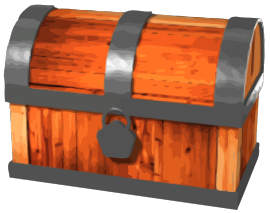
```
class purse(amount : Number) -> Purse {
  var balance is readable, private := amount

   method deposit(amt : Number, src : Purse) -> Boolean
     { if ((amount >= 0) && (src.balance >= amount))
        then {
            src.balance := src.balance - amount
            balance := balance + amount
            return true
         } else {return false}
     }
```

OCaps—Brand Pairs

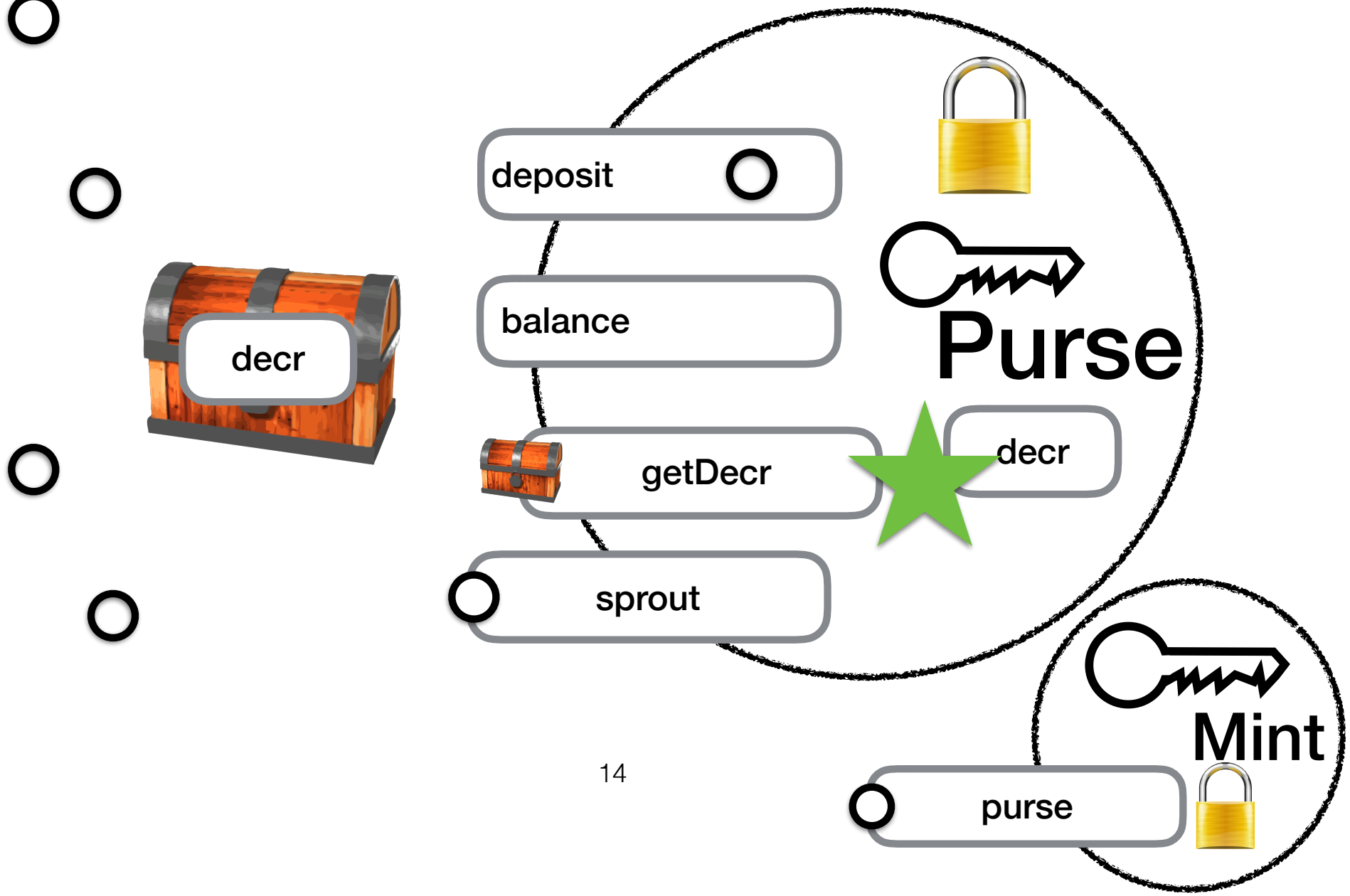
```
type Sealing = interface {
  makeBrandPair -> interface {
    sealer -> Sealer
    unsealer -> Unsealer
  }
}
```



```
type Sealer = interface { seal(o : Object) -> Box }

type Unsealer = interface { unseal(b : Box) -> Object }

type Box = interface { }
```



Brand Pairs

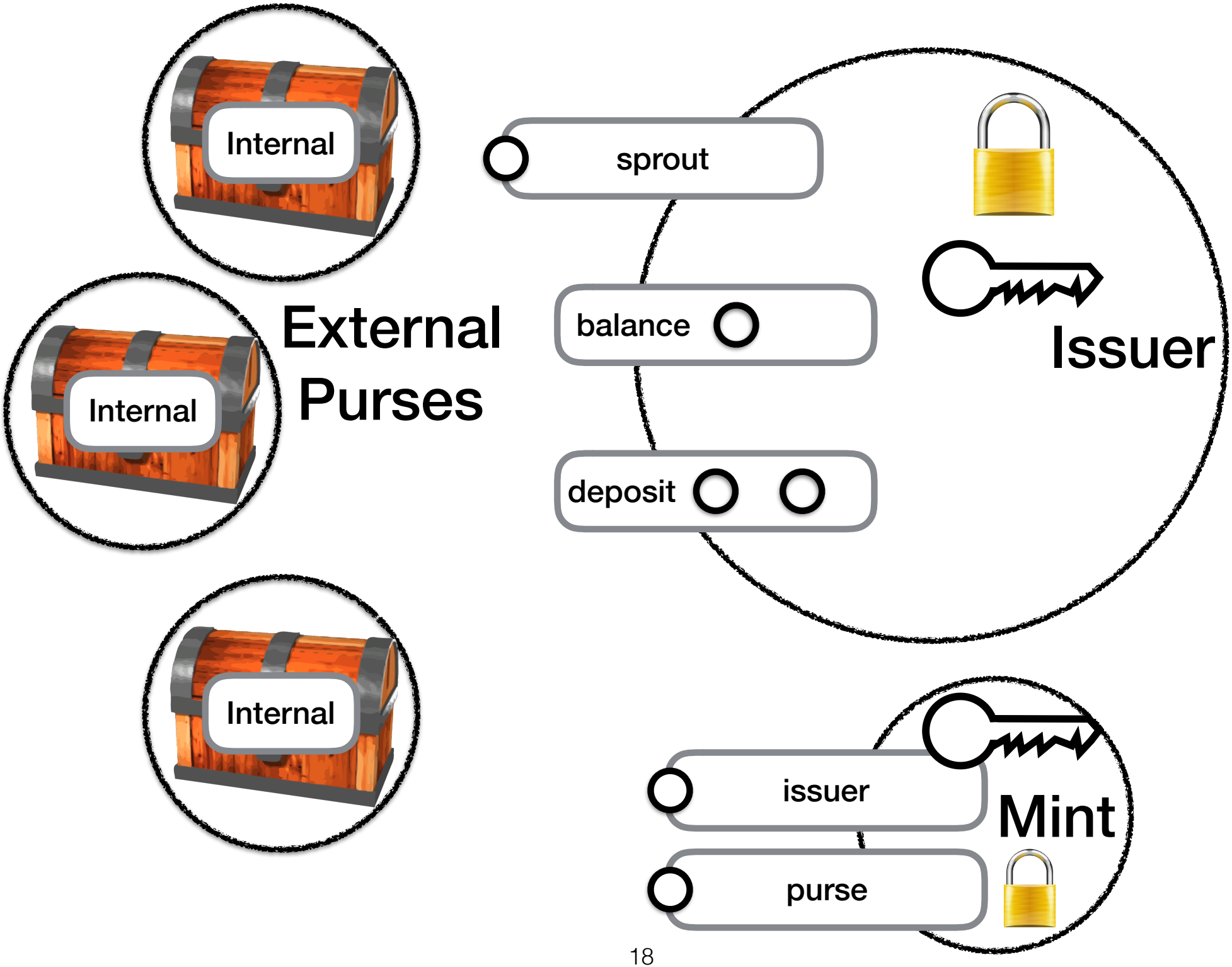


```
class mint -> Mint is public {

  def brandPair = sealing.makeBrandPair

  class purse(amount : Number) -> Purse {

    var balance := amount

    method decr(amt : Number) -> Boolean is confidential {
      if ((amt < 0) || (amt > balance)) then {
          return false }
      balance := balance - amt
      return true }

    method getDecr
      {brandPair.sealer.seal { amt -> decr(amt) } }
```

```
method deposit(amt : Number, src : Purse) -> Boolean {
  if (amt < 0) then { return false }
  var srcDecr
  try { srcDecr := brandPair.unsealer.unseal(src.getDecr) }
    catch { _ -> return false }
  if (srcDecr.apply( amt )) then {
     balance := balance + amt
     return true }
  return false }
```

Generalising the Design

```
type ExternalPurse = interface { }

type InternalPurse = interface {
   balance -> Number
   balance:= ( n : Number )
}

type Issuer = interface {
  balance(of : ExternalPurse) -> Number
  deposit(to : ExternalPurse,
          amount : Number,
          from : ExternalPurse ) -> Boolean
  sprout -> ExternalPurse
}
```


Internal
Internal
Internal
External
Purses
sprout
balance
deposit
Issuer
issuer
purse
Mint

Generalising the Design

```
class mint -> Mint {

  def myBrandPair = sealing.makeBrandPair

  method seal(protectedRep : InternalPurse) -> ExternalPurse
    is confidential { myBrandPair.sealer.seal(protectedRep) }

  method unseal(sealedBox : ExternalPurse) -> InternalPurse
    is confidential { myBrandPair.unsealer.unseal(sealedBox) }

  method purse(amount : Number) -> ExternalPurse {
      seal( object { var balance is public := amount } ) }
```

```
def issuer is public = object {
  method balance(of : ExternalPurse) -> Number {
    return unseal(of).balance}
  method deposit(to : ExternalPurse,
                 amount : Number,
                 from : ExternalPurse) -> Boolean {
    var internalTo
    var internalFrom
    try {
        internalTo := unseal(to)       // throws if fails
        internalFrom := unseal(from)   // throws if fails
    } catch { _ -> return false }
    if ((amount >= 0) && (internalFrom.balance >= amount))
      then {
        internalFrom.balance := internalFrom.balance - amount
        internalTo.balance := internalTo.balance + amount
        return true
      } else {return false}
  }
```

Splitting Mint from Issuer

```
type Mint = interface {
  purse(amount : Number) -> ExternalPurse
  issuer -> Issuer
}
```

External Purse is a pain

```
type ExternalPurse = interface { }
```

Let's be **evil**

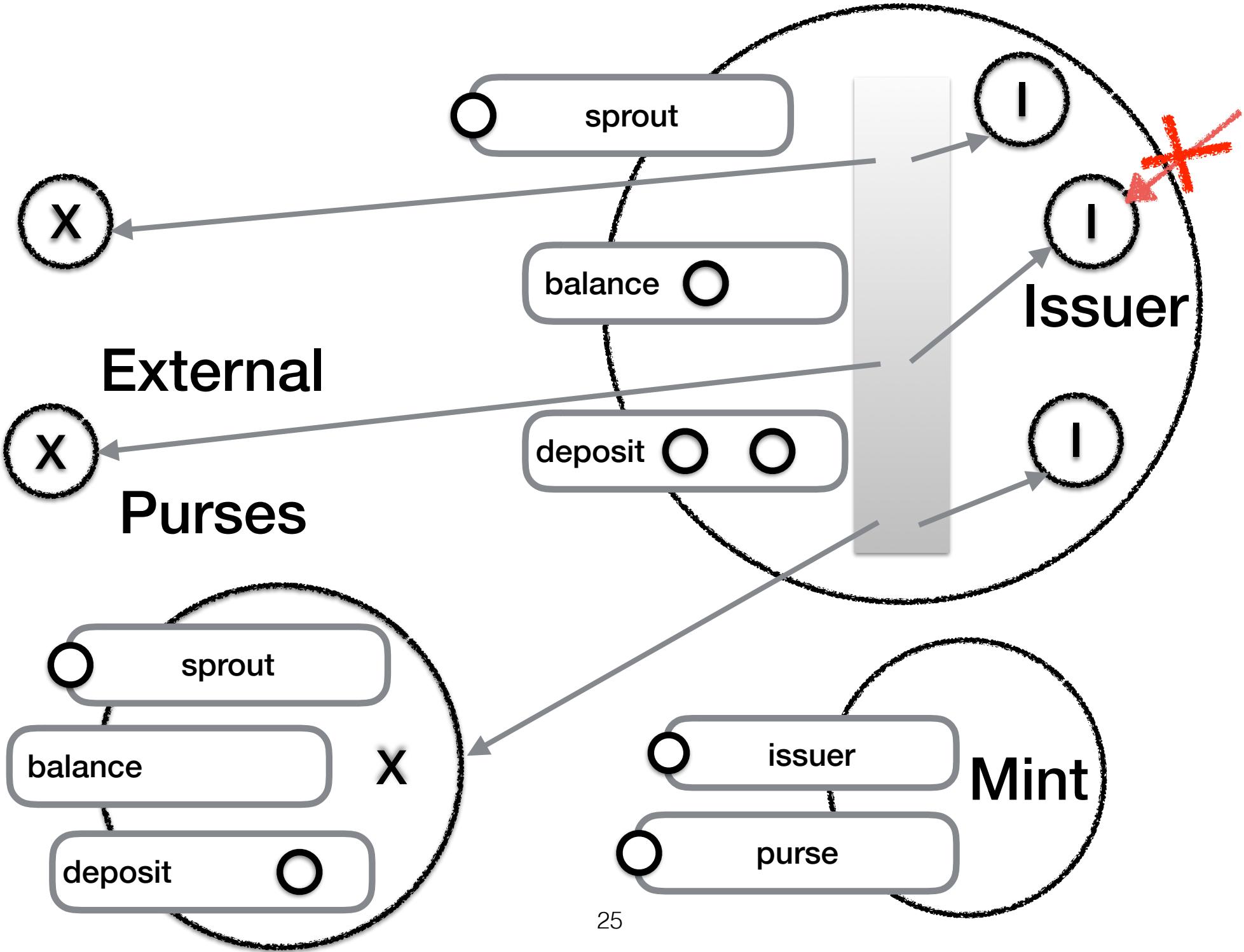
```
type ExternalPurse = interface {
  balance -> Number
  deposit(amount : Number, src : ExternalPurse) -> Boolean
  sprout -> ExternalPurse
}
method seal(protectedRep : InternalPurse) -> ExternalPurse
      is confidential {
     object {
      inherit myBrandPair.sealer.seal(protectedRep)
       method balance { issuer.balance(self) }
       method sprout { issuer.sprout }
       method deposit(amt,src) { issuer.deposit(self,amt,src) }
     }
}
```

```
type ExternalPurse = interface {
  balance -> Number
  deposit(amount : Number, src : ExternalPurse) -> Boolean
  sprout -> ExternalPurse
}
type Mint = interface {
  purse(amount : Number) -> ExternalPurse
  deposit(to : ExternalPurse,
          amount : Number,
          from : ExternalPurse) -> Boolean
  balance(of : ExternalPurse) -> Number
  sprout -> ExternalPurse
}
type InternalPurse = interface {
  balance -> Number
  balance:= (Number) -> Done
  deposit(amount : Number, src : ExternalPurse) -> Boolean
}
```

Hash Table

```
class mint -> Mint {

  def instances = collections.map[[ExternalPurse,InternalPurse]]

  method valid(prs : ExternalPurse) -> Boolean
    { instances.contains(prs) }

  method internal(prs : ExternalPurse) -> InternalPurse
    { instances.get(prs) }

  method purse(amount : Number) -> ExternalPurse {
    def ext = externalPurse(self)
    def int = internalPurse(amount)
    instances.put(ext, int)
    return ext
  }
```

Hash Table


sprout
balance
deposit
I
I
I
Issuer
X
X
External
Purses
sprout
balance
deposit
X
issuer
purse
Mint

Hash Table

```
  method deposit(to : ExternalPurse,
                 amount : Number,
                 from : ExternalPurse) -> Boolean {
    if ((valid(to)) && (valid(from))) then {
        return internal(to).deposit(amount, internal(from))}
    return false
  }
  method balance(prs : ExternalPurse) -> Number
    { internal(prs).balance }

  method sprout -> ExternalPurse {purse(0)}
}
```

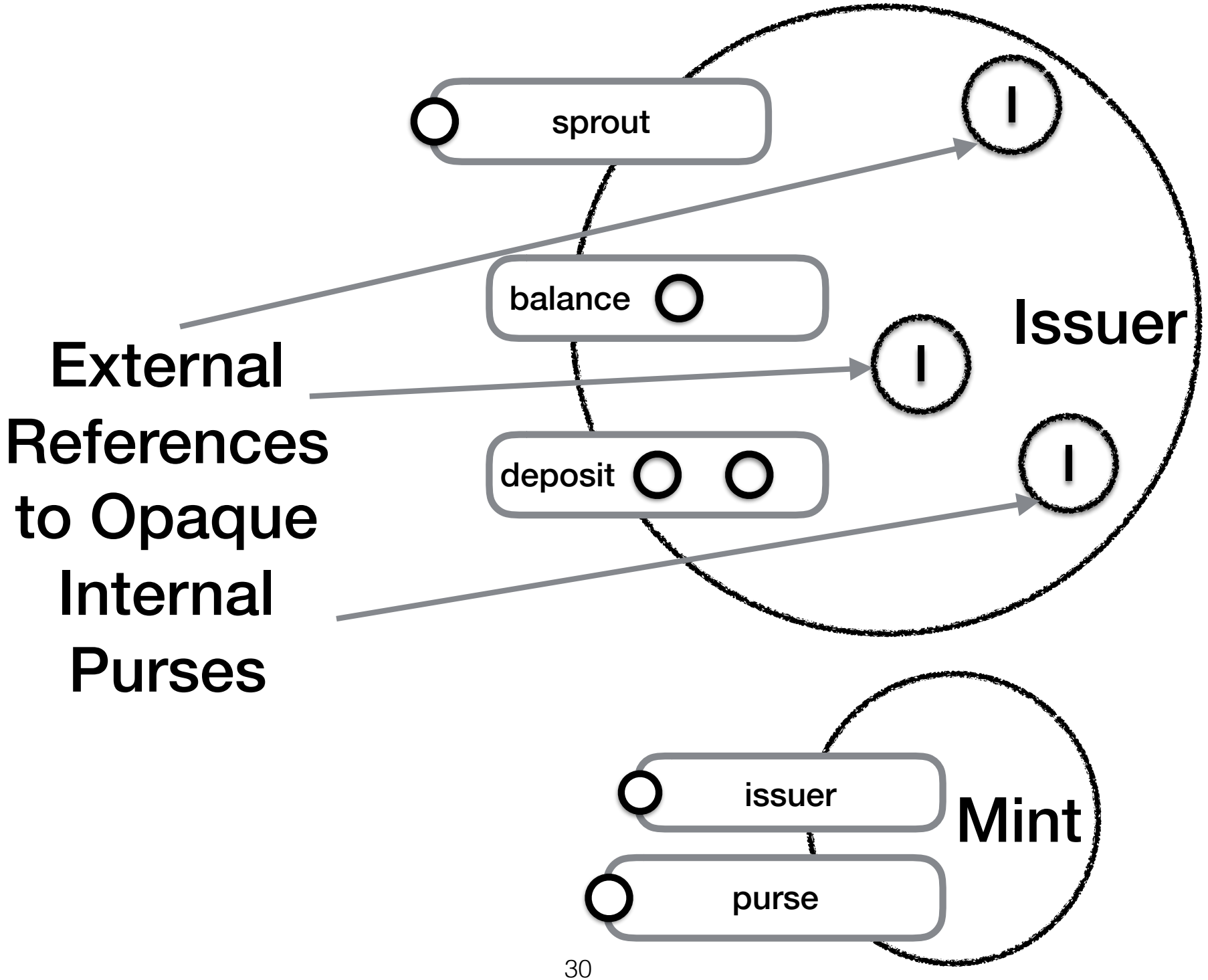
```
class internalPurse(amount : Number) −> InternalPurse {
  var balance is public := amount
  method deposit(amount : Number, src : InternalPurse)
          −> Boolean
    { if ((amount >= 0) && (src.balance >= amount)) then {
        src.balance := src.balance − amount
        balance := balance + amount
        return true }
      return false
    }
}
```

```
class externalPurse(mint' : Mint) −> ExternalPurse {
  def mint = mint'
  method balance {mint.balance(self)}
  method sprout −> ExternalPurse { mint.sprout }
  method deposit(amount : Number, src : ExternalPurse)
          −> Boolean { return mint.deposit(self, amt, src)
  }
}
```

"*you are in a maze of twisty little* ***objects****, all alike*"

```
type Mint = interface {
  purse(amount : Number) −> Purse
  issuer −> Issuer
}

type Issuer = interface {
  balance(of : Purse) −> Number
  deposit(to : Purse,
          amount : Number,
          from : Purse) −> Boolean
  sprout −> Purse
}

type Purse = interface {
   balance −> Number
   balance:= ( n : Number )
}
```

Owners as Readers


External References to Opaque Internal Purses
sprout
balance
deposit
Issuer
issuer
purse
Mint

```
class mint −> Mint {
  class purse(amount : Number) −> Purse is owned {
    var balance is public := amount
  }
  def issuer is public = object {
    method deposit(to : Purse is owned, amount : Number,
           from : Purse is owned) −> Boolean {
      if ((amount >= 0) && (from.balance >= amount))
        then {
          from.balance := from.balance − amount
          to.balance := to.balance + amount
          return true
        } else {return false}
    }
    method balance(of : Purse is owned) −> Number {
      return of.balance}
    method sprout −> Purse { purse(0) }
  }
}
```

```
type Mint = interface {
  purse(amount : Number) −> Purse
  issuer −> Issuer
}

type Issuer = interface {
  balance(of : Purse) −> Number
  deposit(to : Purse,
          amount : Number,
          from : Purse) −> Boolean
  sprout −> Purse
}

type Purse = interface {
   balance −> Number
   balance:= ( n : Number )
}
```

Owners as Readers ?

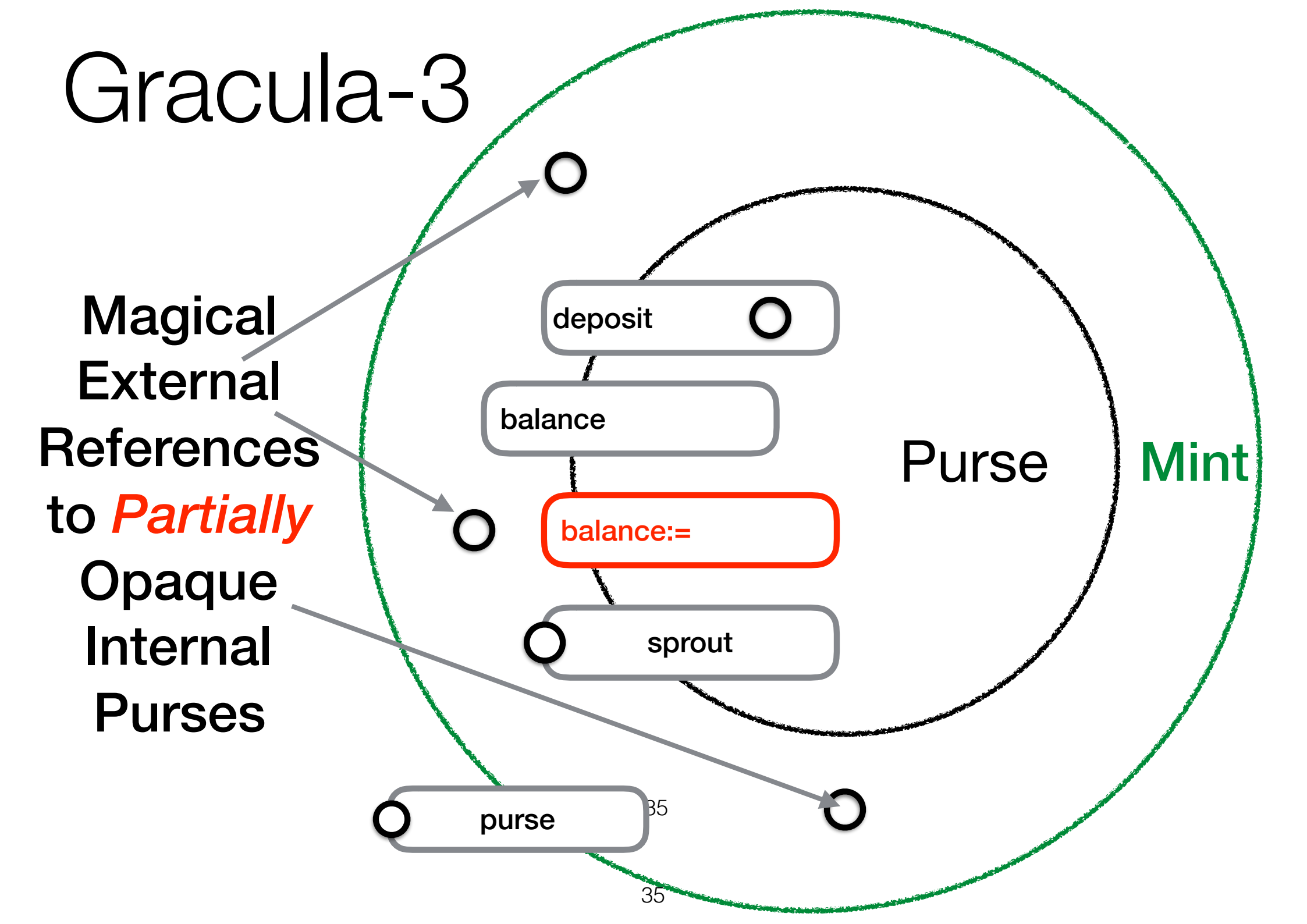
ADTs, Objects, Owners

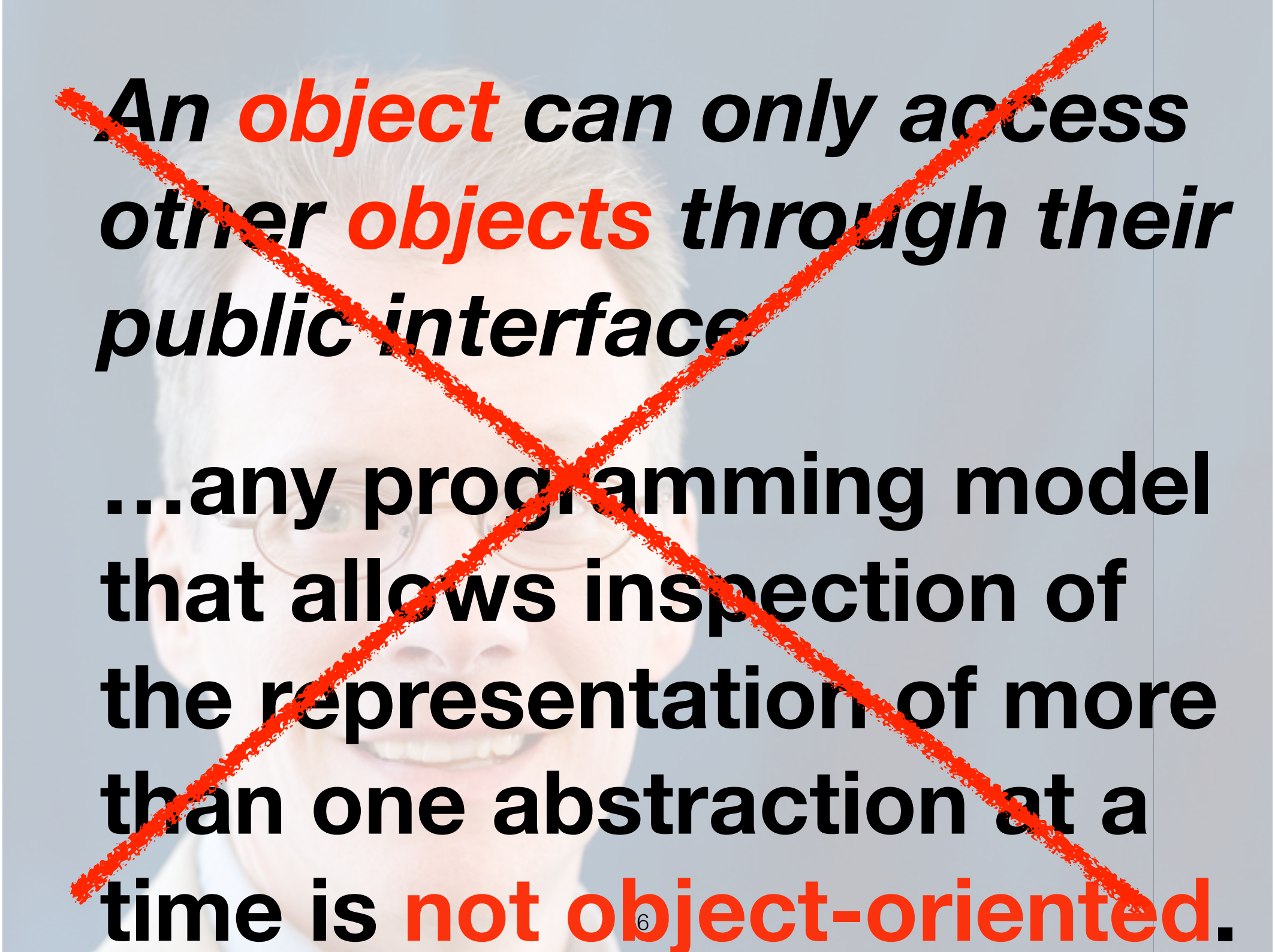
```
type Mint = interface {
  purse(amount : Number) -> Purse
}

type Purse = interface {
   balance -> Number
   deposit(amount : Number, src : Purse) -> Boolean
   sprout -> Purse
}
```

```
class mintADT -> Mint is public {

  class purse(amount : Number) -> Purse is owned {
    var balance is readable, owner := amount

    method deposit(amt : Number,
                   src : Purse is owned) -> Boolean
      { if ((amount >= 0) && (src.balance >= amount))
          then {
              src.balance := src.balance - amount
              balance := balance + amount
              return true
          } else {return false}
      }

    method sprout -> Purse is owned { purse(0) }
  }
}
```

Gracula-3

Gracula-3


Magical
External
References
to *Partially*
Opaque
Internal
Purses
deposit
balance
balance:=
sprout
purse
Purse
Mint



An object can only access other objects through their public interface

…any programming model that allows inspection of the representation of more than one abstraction at a time is not object-oriented.

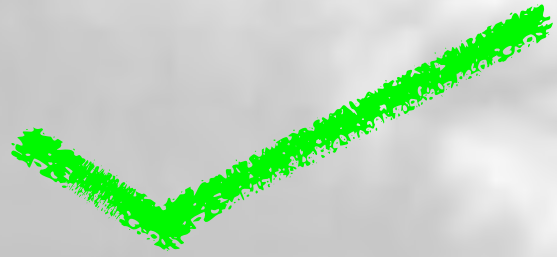
Object Capabilities

- **Only connectivity begets connectivity**
  - *"No ambient authority"*
- **Access implies permission**
  - *"No access control lists*
- Dynamic Loading

Capability Based Financial Instruments
Miller, Morningstar, Frantz. **Fin. Cryp. 2000**

Object Capabilities

- **Only connectivity begets connectivity**
  - *"No ambient authority"*
- **Access implies permission**
  - *"No access control lists"*
- Dynamic Loading



Capability Based Financial Instruments ***Miller, Morningstar, Frantz.*** **Fin. Cryp. 2000**