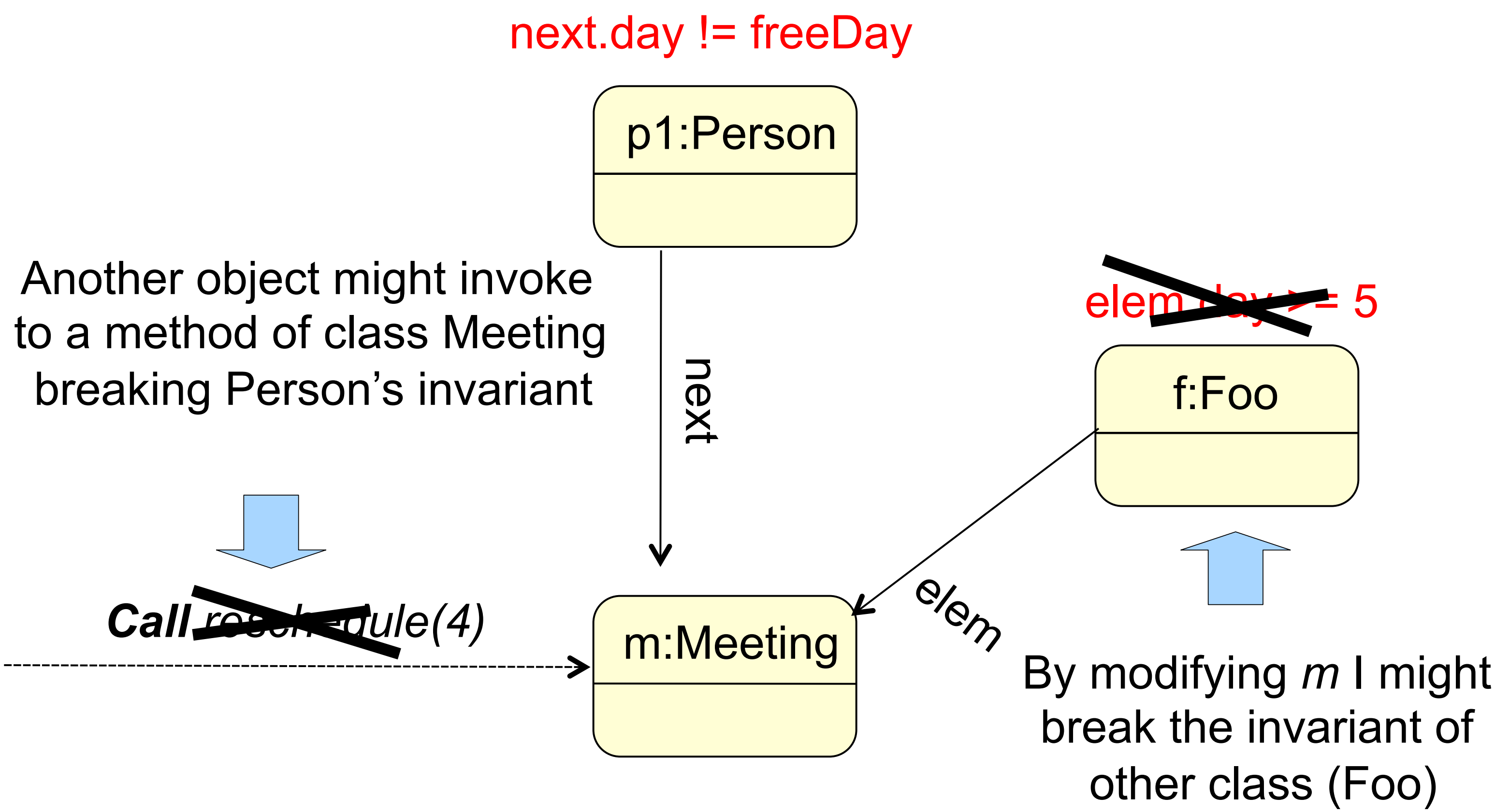
Problem: linked data structures

- What about this case?
- Can we reason about them modularly?

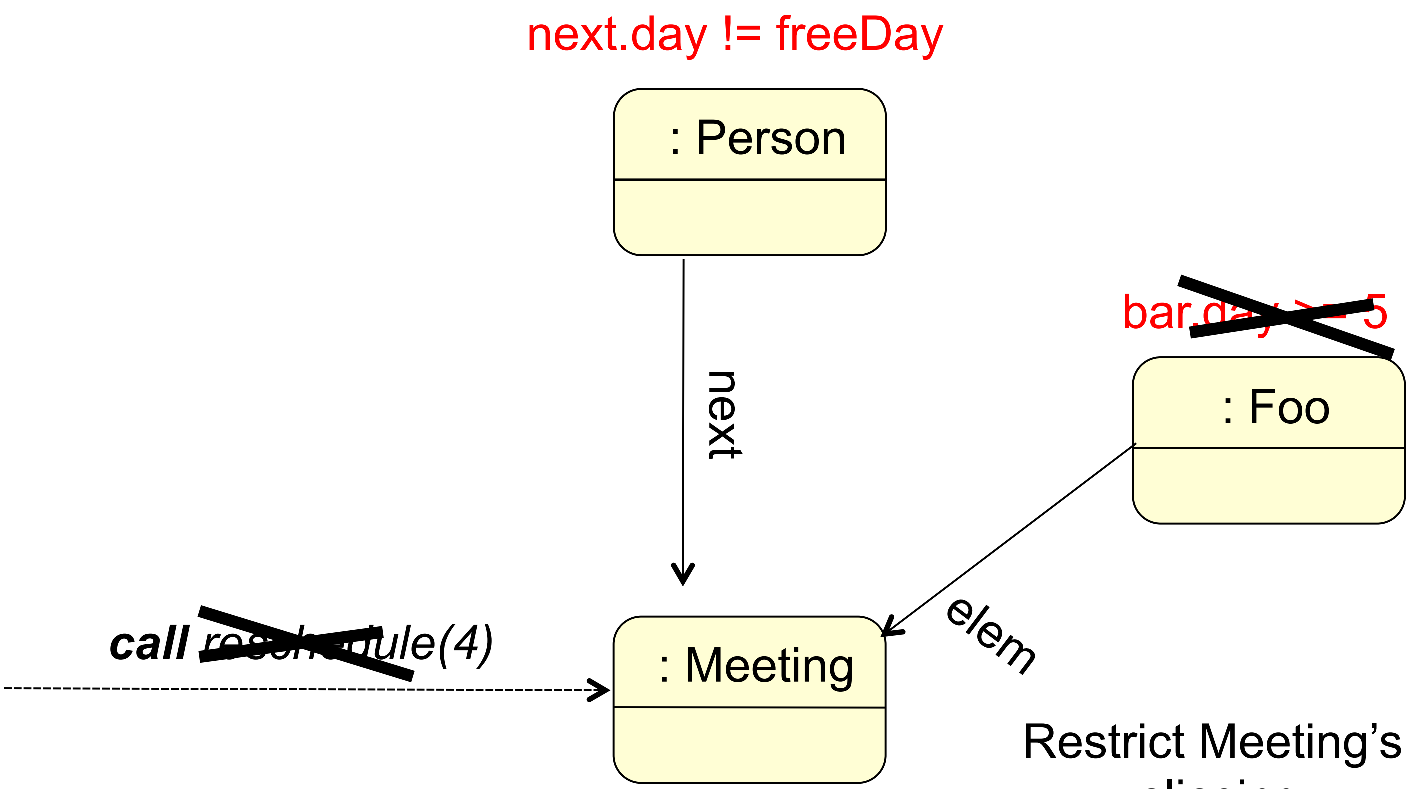
```
class Meeting {
  int day;
  invariant 0 ≤ day<7;

  void Reschedule(int d)
       requires 0 ≤ d <7;
  {
    expose(this){
      day = d;
    }
  }
}
```

```
class Person {
  int freeDay;
  Meeting next;
  invariant this.next != null
 => this.next.day != freeDay;
}
```

Threats to Person's object invariant

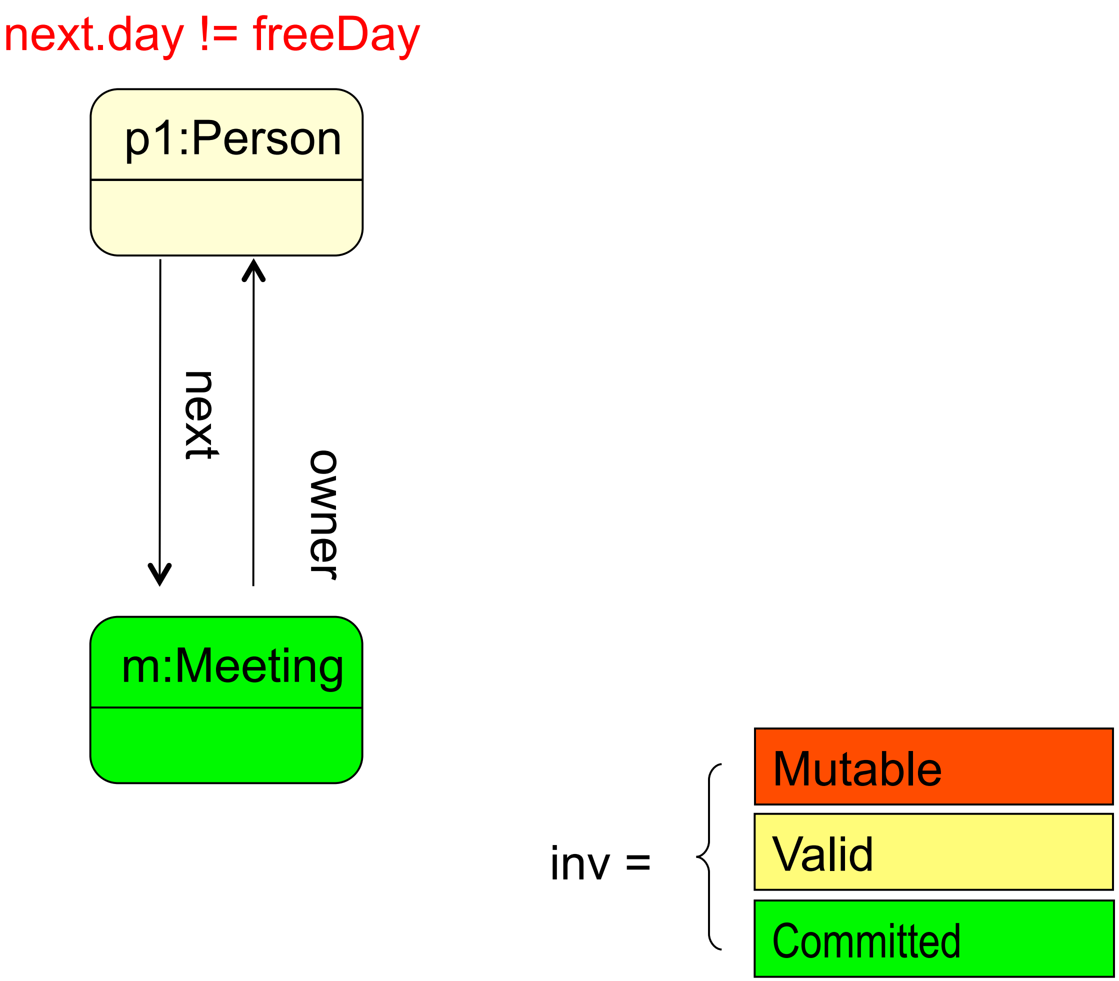


Threats to Person's object invariant

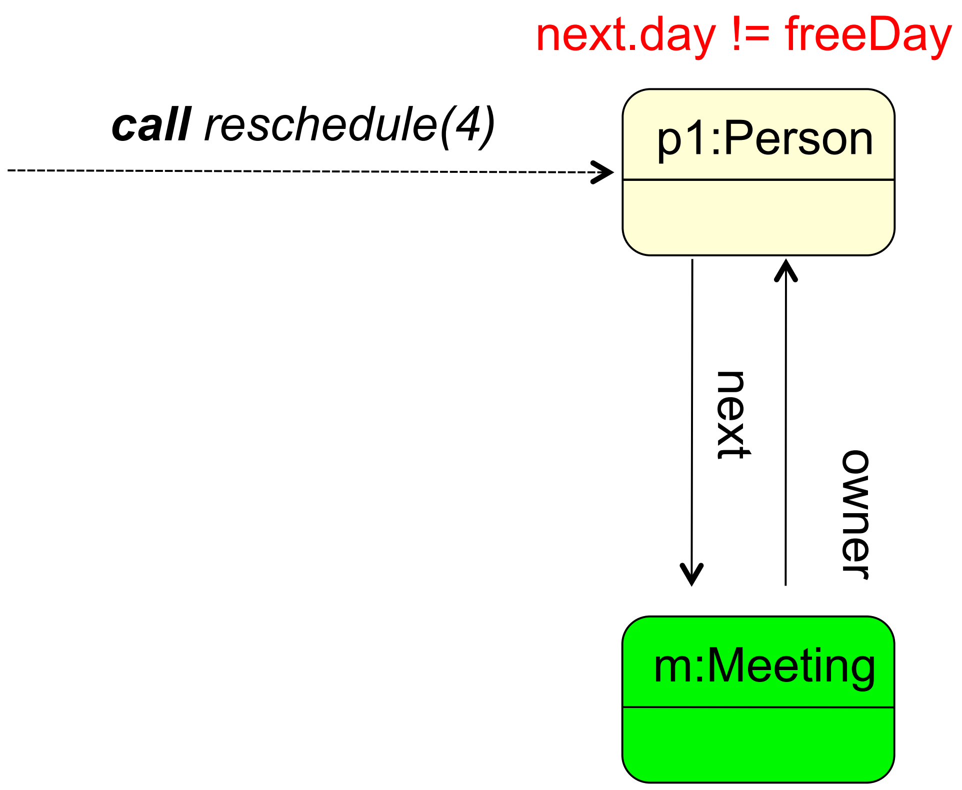


Restrict Meeting's aliasing

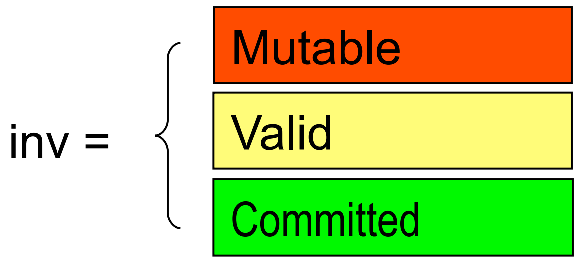
Expose+ownership



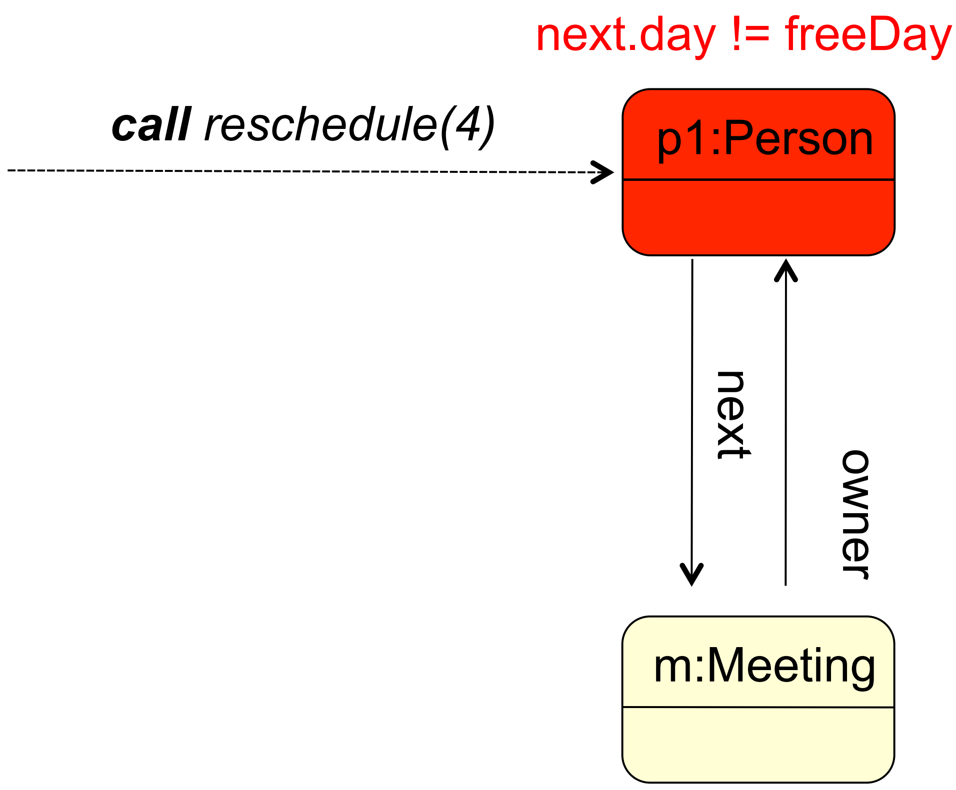
Expose+ownership

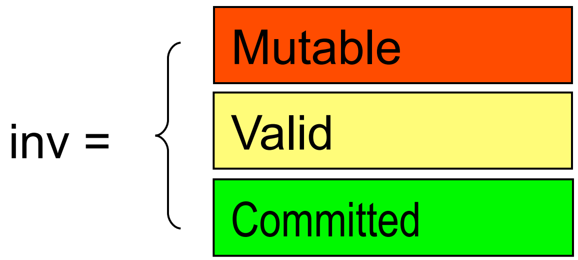


requires this.inv=Valid

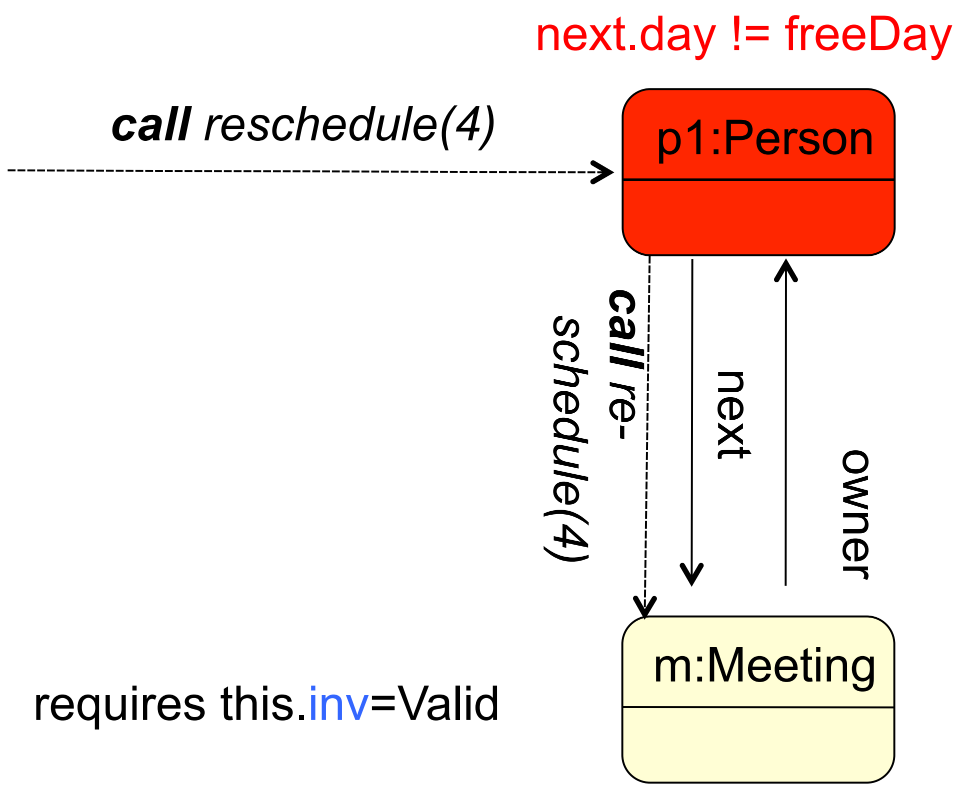


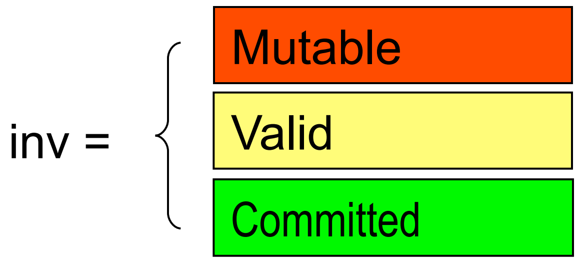
Expose+ownership



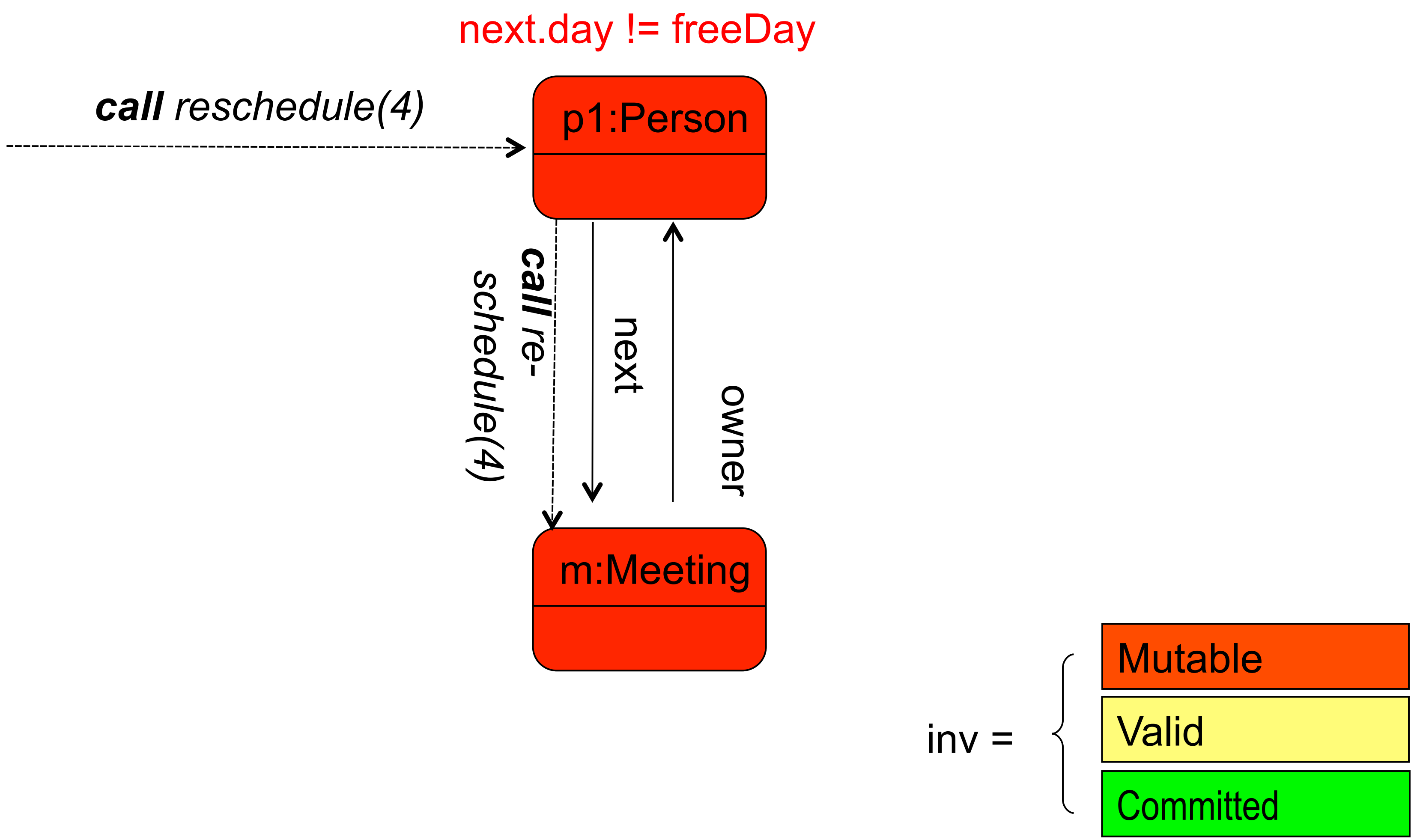


Expose+ownership

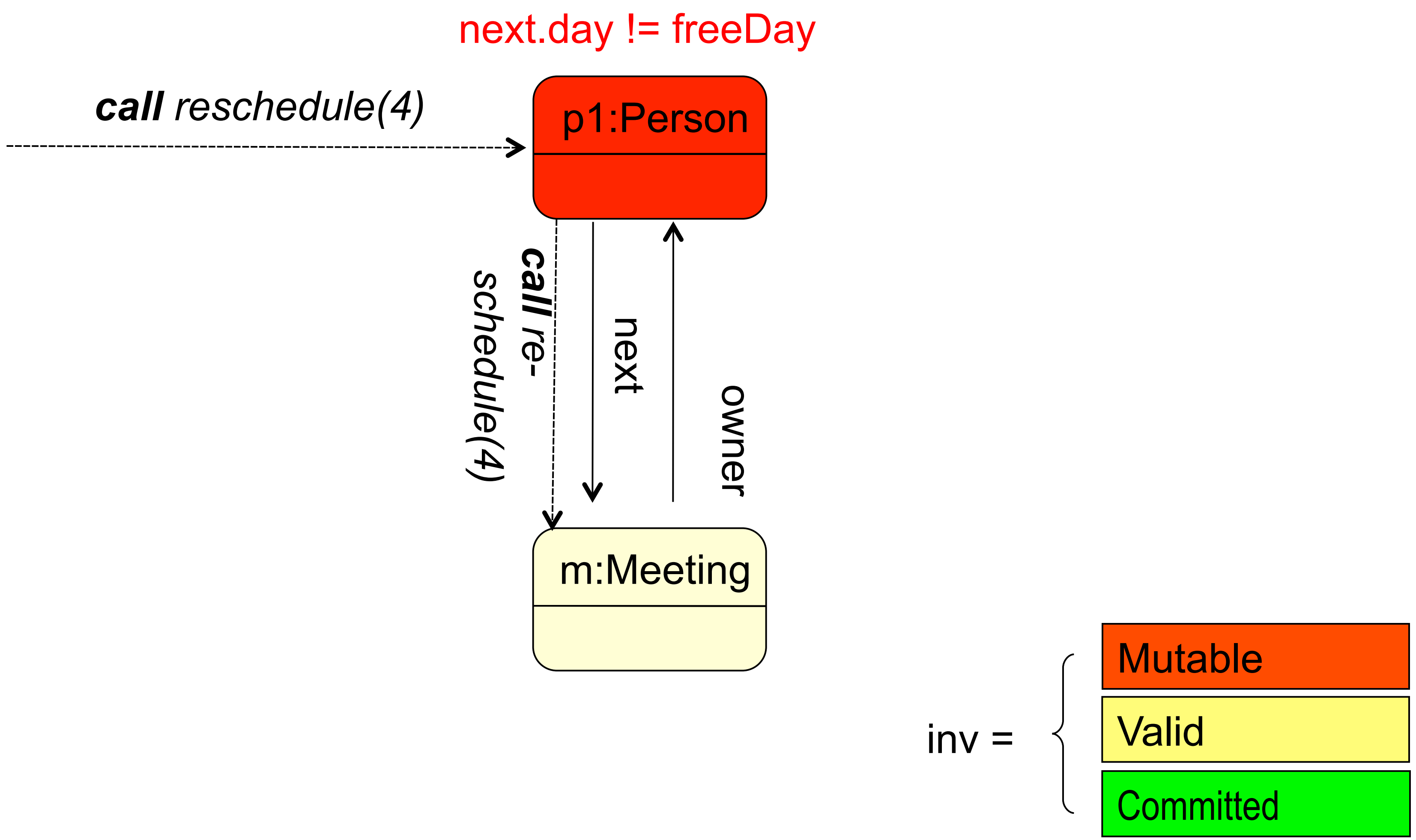




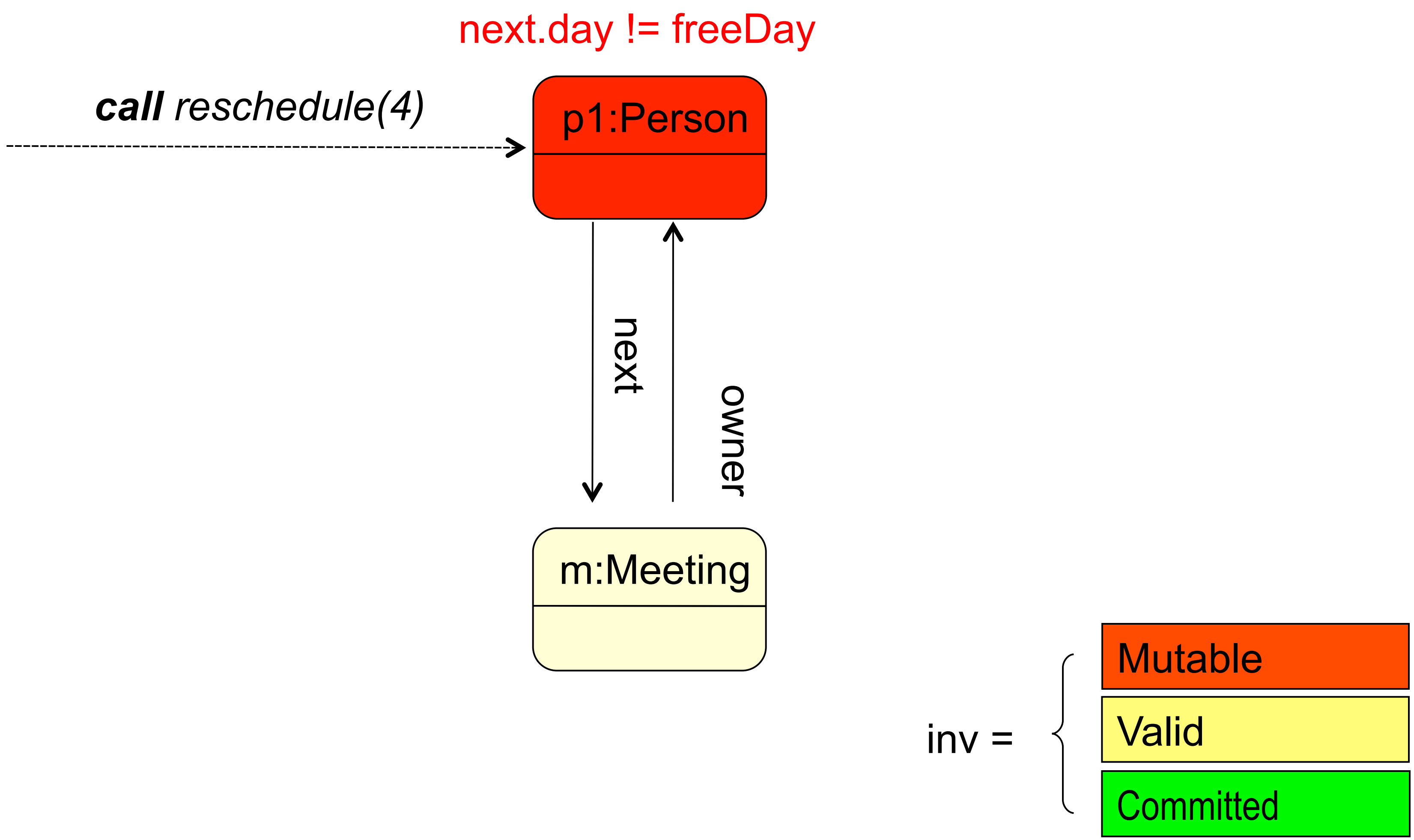
Expose+ownership



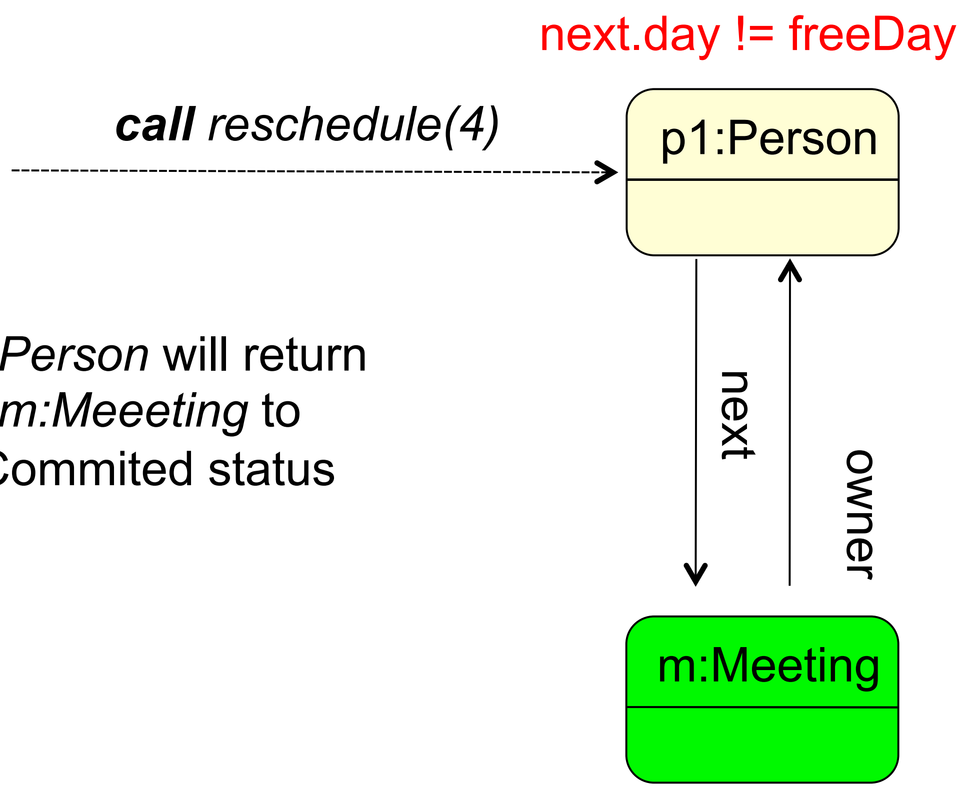
Expose+ownership



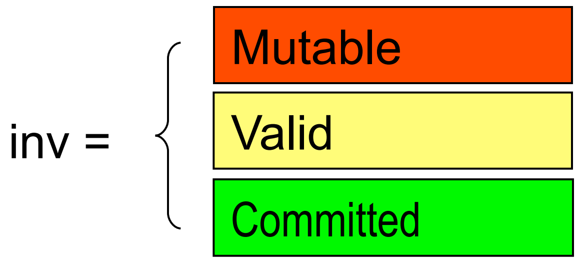
Expose+ownership



Expose+ownership

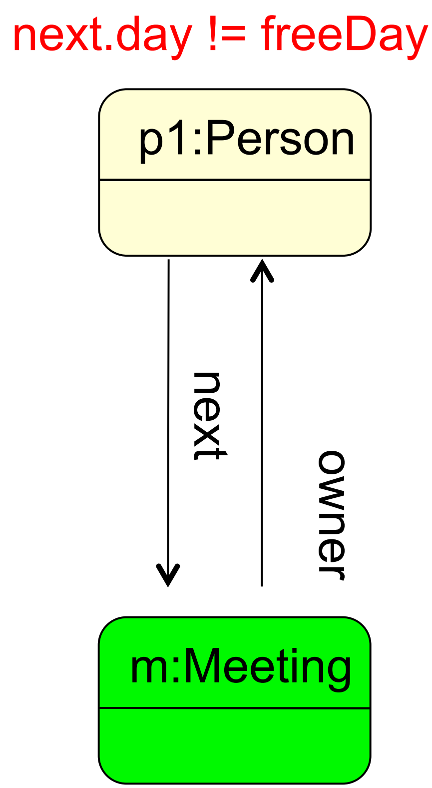


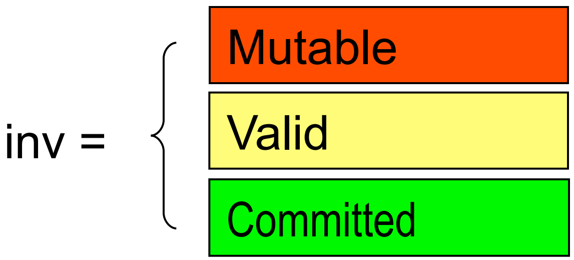
p1:Person will return
m:Meeeting to
Commited status



Expose+ownership

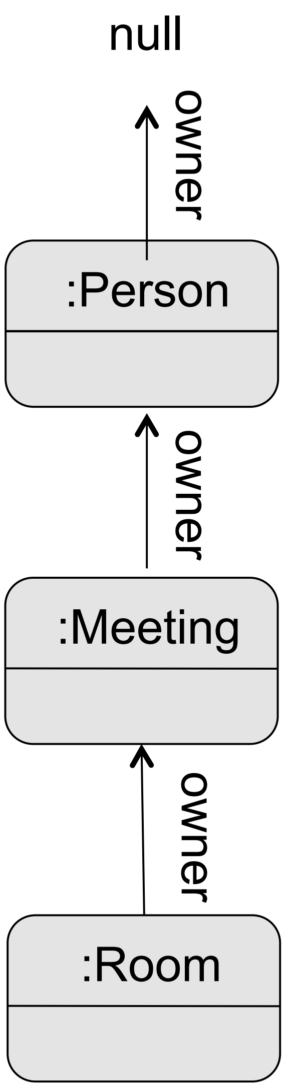
In case it exists, P1's owner will return this object to Commited status





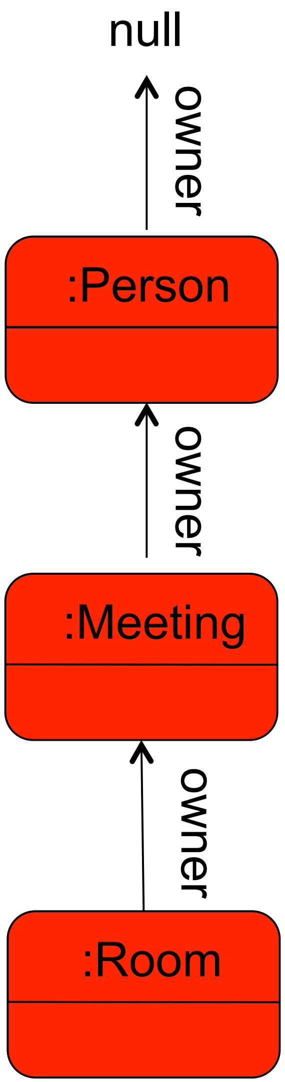
Objects invariants + ownership

- Which objects must be in Mutable status if Room is Mutable?
- From `Person`'s perspective, what fields can I access?



Object invariants + ownership

- ownership is an <u>acyclic</u> relation
  - I can not own my owner
  - Each object has at most one owner
- **Ownership rule**:
  - If o.inv = Mutable, then *owner*(*o*), *owner*(*owner*(*o*)), … are Mutable.
- The object invariant of *o* can only depend on:
  - The fields of *o*
  - Any field of any other object which o <u>owns</u> (recursively)



Supporting Ownership

- A new ghost field is added:
  - owner: reference to the “owner” of the object
- Field inv values are ∈ {Committed, Valid, Mutable}
- An object status is Committed if:
  - The object invariant holds
  - Its owner is not in Mutable status
- Commited: acts as a lock to guarantee validity

Rep References - Example

- The ***rep*** (representation) modifier introduces implicitly *ownership* invariants

```
class Person {
  int freeDay;
  [rep] Meeting next;

  /*implicit invariant
  next ≠ null ⇒ next.owner = this;
  */
  …
}
```

Pack/Unpack+Ownership

- pack/unpack is extended to support this new protocol

```
unpack o:

assert o.inv = Valid;
o.inv := Mutable;
foreach (c |c.owner = o)
   { c.inv := Valid; }
```

```
pack o:

assert o.inv = Mutable;
assert ∀c: c.owner = o ⇒
   c.inv = Valid;
foreach (c |c.owner = o)
   { c.inv := Committed; }
assert Invariant(o);
o.inv := Valid
```

Invariants+Ownerships/Rep

Memory state:

$\forall o$: o.inv $\neq$ mutable $\Rightarrow$

Inv(o) $\wedge$

($\forall c$: c.owner = o $\Rightarrow$ c.inv = Committed))

Admissible Invariants:

Only accesses to fields

- this.f_1. … .f_n, where f_1 … .f_{n-1} are fields of "rep" references

Example (reloaded)

```
class Person {
  int freeDay;
  rep Meeting next;

  invariant next ≠ null ⇒
   next.day ≠ freeDay;

  int doTravel(int td)
    requires inv==valid;
    modifies this.*;
  {
    expose(this) {
      freeDay = td;
      if (next!=null) {
       next.reschedule((td+1)%7);
    };
  }
```

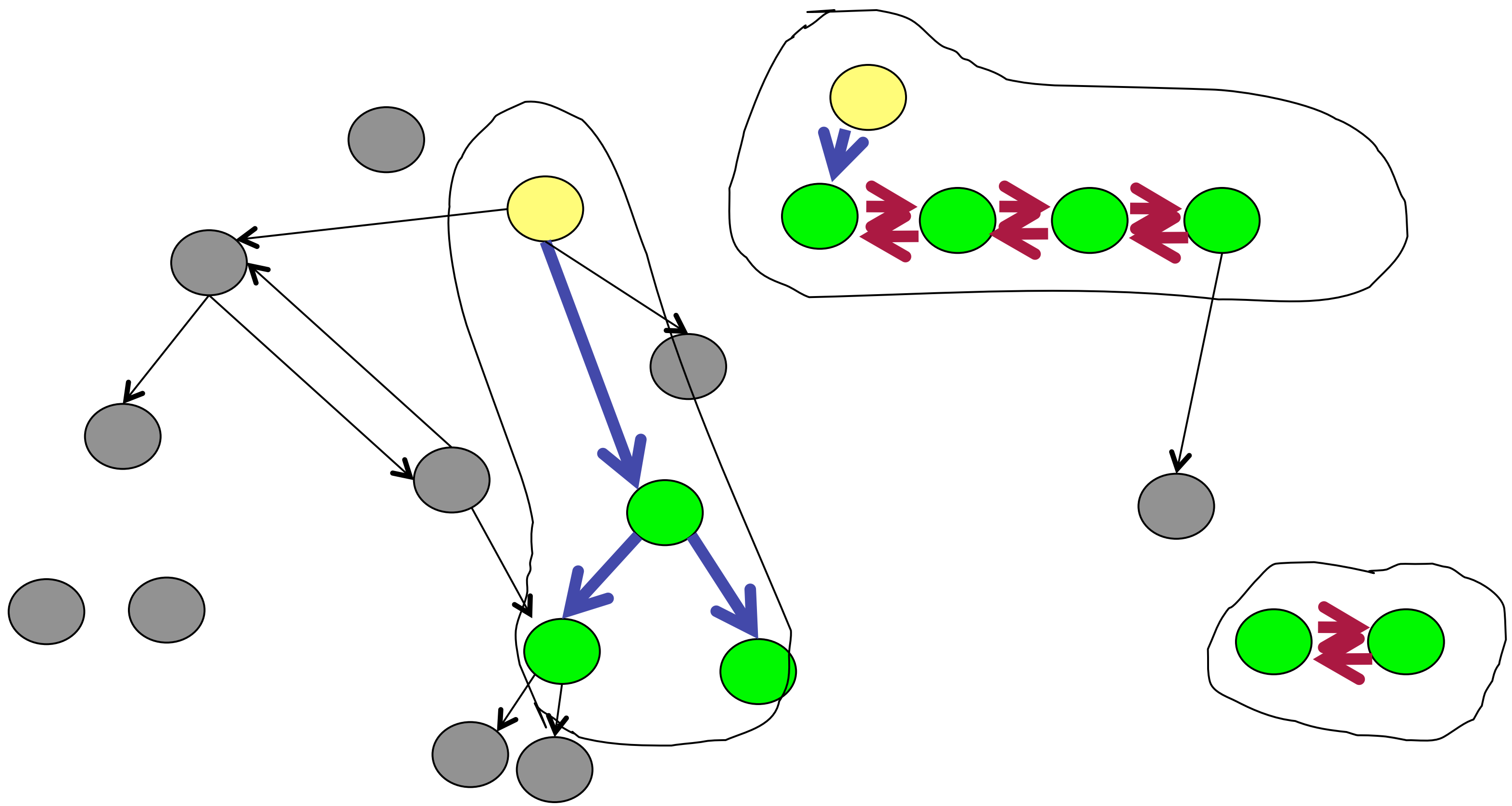
```
class Meeting {
  int day;
  void reschedule( int d )
    requires inv==valid; {
       expose(this) {
         day = d;
       }
    }
  }
```

```
Person person = ... ;
Meeting meeting = ... ;
person.next := meeting ;
```

The **only** owner of meeting is person

Rep references

- *[Rep] defines* an object hierarchy
- What happens to other (recursive) structures?

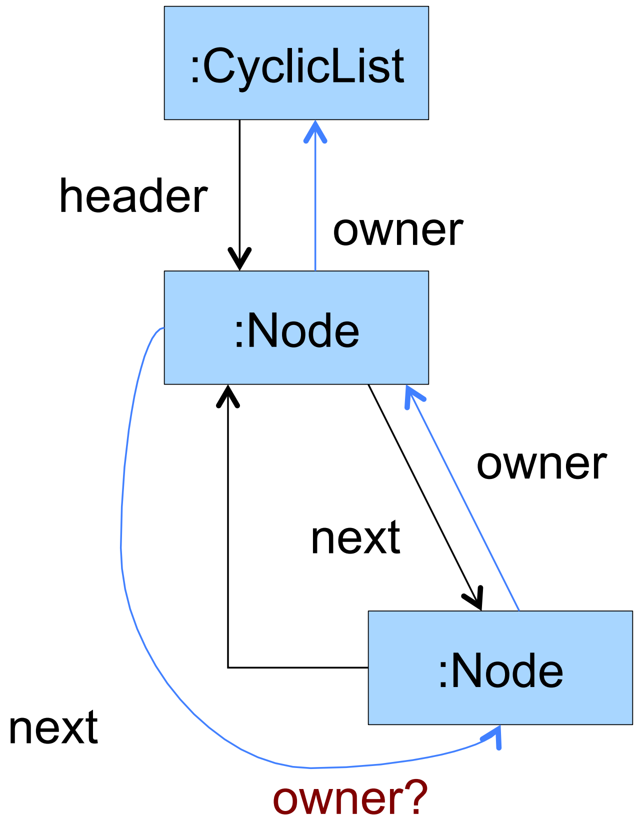


Example: cyclic list?

```
class CyclicList {
 [rep] Node header;
 //implicit invariant header.owner == this
}
class Node {
 [rep] Node next;
 //implicit invariant next.owner == this
}
```

Problem with cyclic lists

```
class CyclicList {
  [rep] Node header;
  //header.owner == this
}
class Node {
  [rep] Node next;
  //next.owner == this
}
```



Peer references

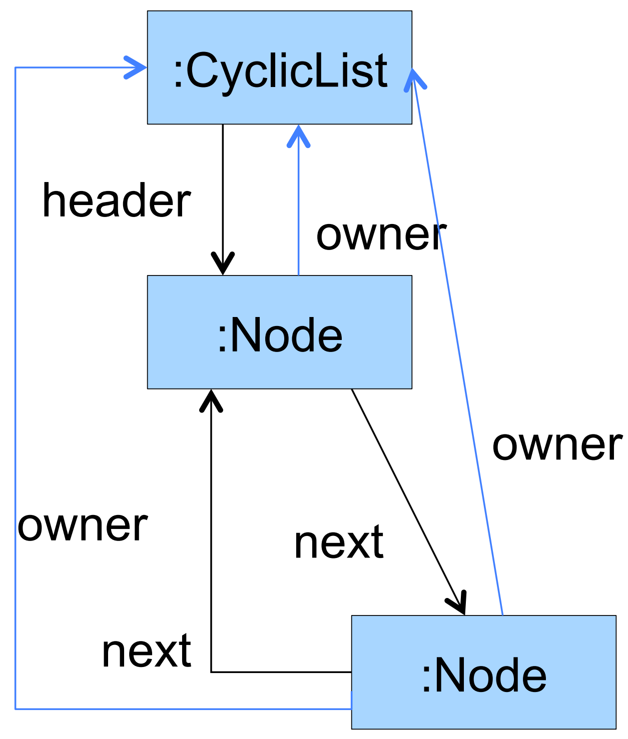
```
class T1 {
   [rep] Object f1;
   /*f1.owner=this*/
   …
}
```

```
class T1 {
   [peer] Object f1;
   /*f1.owner=this.owner*/
   …
}
```

- The **[rep]** modifier states I am the owner of the reference
- The **[peer]** modifier states the reference and I share the same owner

Example: cyclic lists

```
class CyclicList {
  [rep] Node header;
  //header.owner == this
}
class Node {
  [peer] Node next;
  // next.owner == this.owner
}
```



Modular verification of invariants in Spec#

- This methodology deals with
  - Re-entrancy (using the “inv” field value)
  - Nested structures (using ownership)
- It handles :
  - Recursive linked structures (lists)
  - Recursion, ownership transference (not seen today)
- It allows a modular verification
  - Check only the invariant of the class under analysis
  - Access protocol (inv field)
  - Aliasing is not restricted

Some references

- Tutorial Spec#
  - http://www.cs.nuim.ie/~rosemary/ETAPS-SpecSharp-Tutorial.pdf
- Paper:
  - M. Barnett et al. **Boogie: A modular reusable verifier for object-oriented programs**. 2006