Konzepte objektorientierter Programmierung

Prof. Dr. Peter Müller

Werner Dietl

Software Component Technology

Exercises 9: Ownership

Wintersemester 05/06

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Questions?

Exercise 2 – ArrayList

```
public class ArrayList {
  private rep int[] array;
  private int next;
  public ArrayList() {
     array = new rep int[ 20 ];
     next = 0;
  }
  public void add( int i ) {
     if( next==array.length ) { resize( ); }
     array[ next ] = i;
     next++;
  }
```

Exercise 2 – ArrayList

```
public void setElems( readonly int[] ia ) {
  if( array.length < ia.length ) {
     array = new rep int[ ia.length ];  }
  for(int i=0; i<ia.length; ++i ) {
     array[i] = ia[i];   }
  next = ia.length;
}
protected void resize() {
  if( next==array.length ) {
     readonly int[] oa = array;
     array = new rep int[2*oa.length];
     setElems( oa ); }
}
```

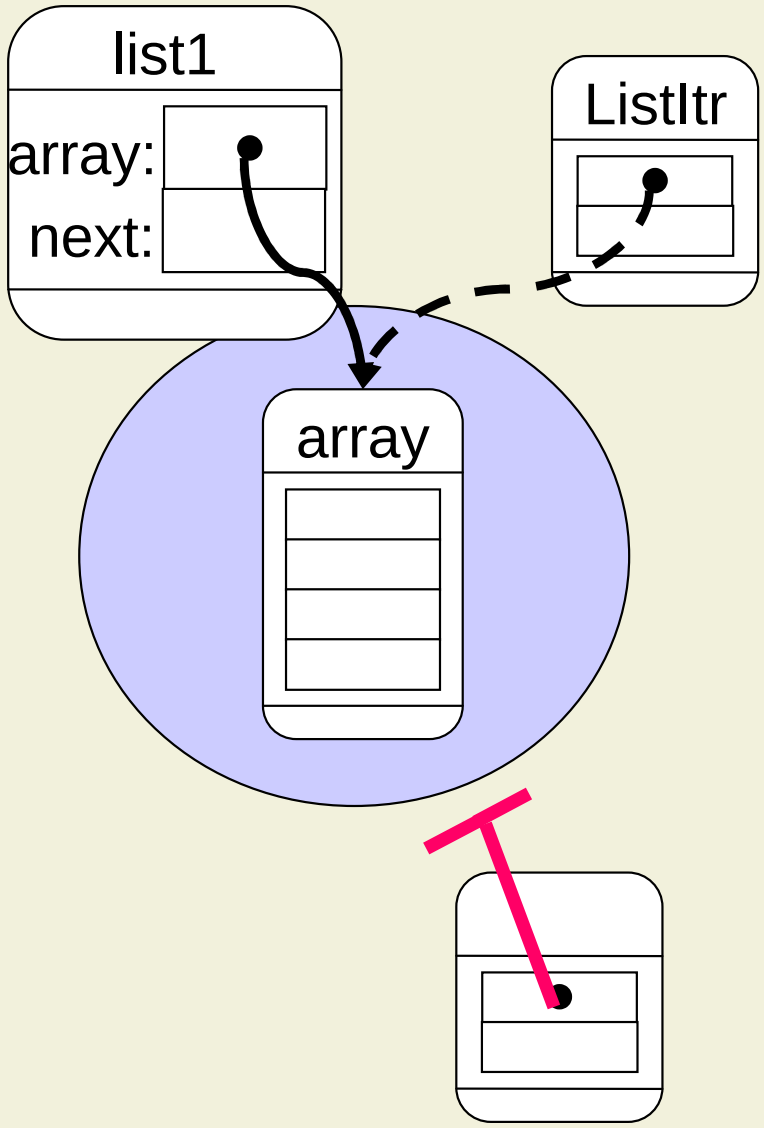
Exercise 2 – ArrayList

```
public readonly int[] getElems() {
  return array; }

public static void main( String[] args ){
  int[] myarr = new int[10];
  for(int i=0; i<myarr.length; ++i)
     myarr[i] = i;
  ArrayList al = new ArrayList();
  al.setElems( myarr );
  System.out.println("Before: " + al);
  myarr[0] = 42;
  System.out.println("After:  " + al); }
```

Exercise 2 – ArrayList

- Array is encapsulated
- Readonly access possible
- No external modifications



Exercise 3 – Entries

```
class Entry {
  readonly Object element;
  peer Entry previous, next;

  Entry( readonly Object o,
         peer Entry p, peer Entry n ) {
    element = o;
    previous = p;
    next = n; }
}
```

Exercise 3 – LinkedList

```
public class LinkedList {
  private rep Entry header;
  private int size;

  public LinkedList() {
     header =
        new rep Entry(null, null, null);
     header.next = header;
     header.previous = header;
     size = 0;
  }
```

Exercise 3 – LinkedList

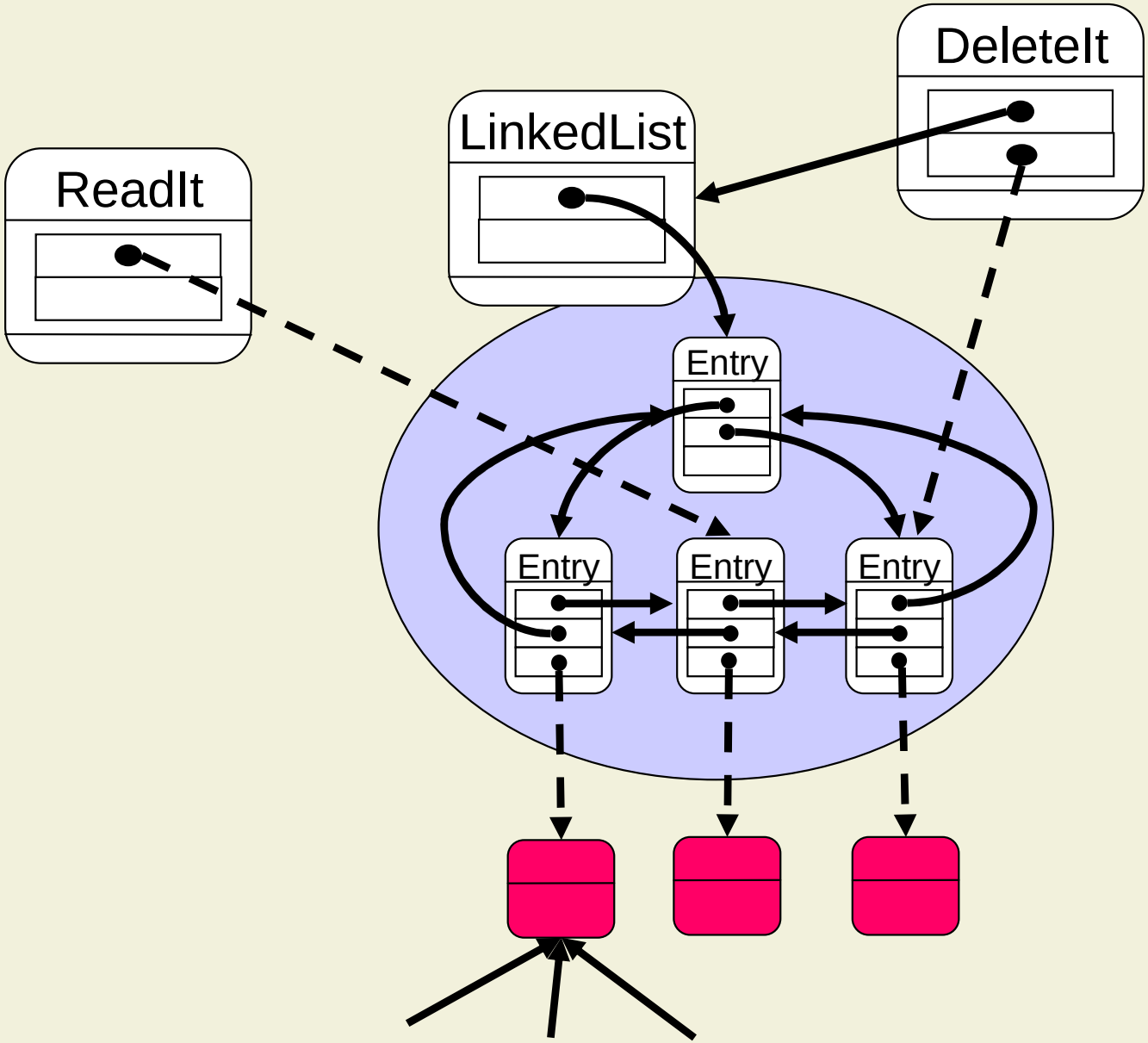
```
public void add( readonly Object o ) {
  rep Entry newE =
    new rep Entry(o,header,header.next);
  header.next.previous = newE;
  header.next = newE;
  ++size;
}
```

Exercise 3 – LinkedList

```
public pure readonly Object
            get( int idx ) {
  if( idx > size ) return null;

  readonly Entry e = header.next;
  for( int i=0; i<idx; ++i ) {
            e = e.next; }
  return e.element;
}
```

Exercise 3 – LinkedList



Exercise 3 – LinkedList

```
public pure peer ReadIterator
  getReadIterator() {
     return new peer ReadIterator(
                                header );
}
public pure peer DeleteIterator
  getDeleteIterator() {
     return new peer DeleteIterator(
                             this, header );
}
```

Exercise 3 – LinkedList

```
// precondition that the node belongs to us
/*@ requires e.owner == this; @*/
protected void delete( readonly Entry e ) {
  rep Entry re = (rep Entry) e;
  if( re.previous != null )
     re.previous.next = re.next;
  if( re.next != null )
     re.next.previous = re.previous;
  --size;
}
```

Exercise 3 – ReadIterator

```
public class ReadIterator {
  public ReadIterator(readonly Entry h)
  {
     // the header is a dummy that is
     // never null
     current = h;
     header = h;
  }
  public pure boolean hasNext() {
     return current.next != header; }
```

Exercise 3 – ReadIterator

```
  public void moveNext() {
      current = current.next;
  }
  public pure readonly Object element() {
      return current.element;
  }

  protected readonly Entry current;
  protected readonly Entry header;
}
```

Exercise 3 – DeleteIterator

```
public class DeleteIterator
                  extends ReadIterator {
  public DeleteIterator( peer LinkedList l,
                         readonly Entry h ) {
     super( h ); list = l;
  }
  public void delete() {
     list.delete(current);
     current = current.next;
  }
  private peer LinkedList list;
}
```

Exercise 3 – LinkedList Main

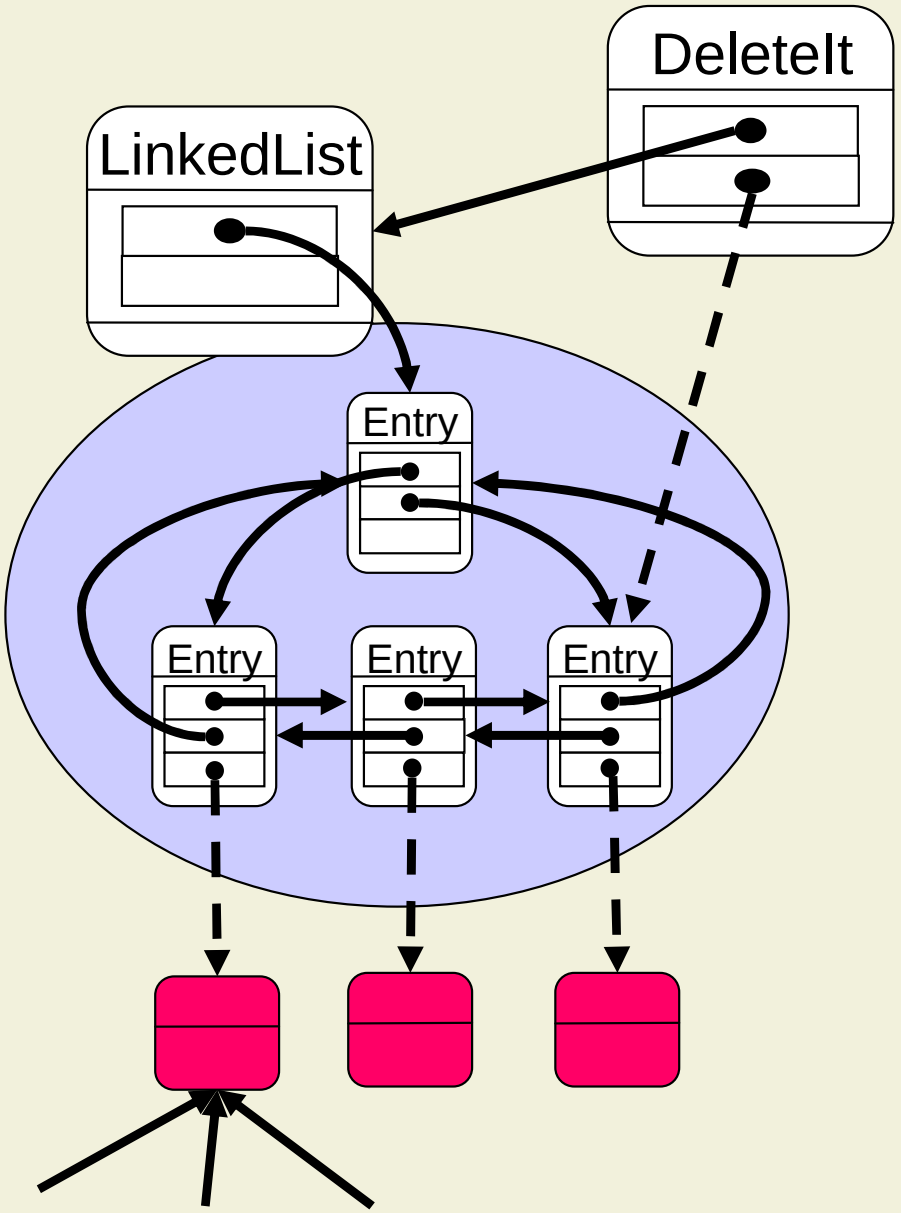
```
public static void main( String[] args ) {
  LinkedList ll = new LinkedList();
  ll.add( new Integer(20) );
  ll.add( "xyz" );
  ll.add( new Float(2.2f) );
  ll.add( "Hello World" );
  ll.add( new Object() );
  ReadIterator itr = ll.getReadIterator();
  int i = 0;
  while( itr.hasNext() ) { itr.moveNext();
     System.out.println("Element[" +i+ "]: " +
                                  itr.element() );
     ++i;
  }
```

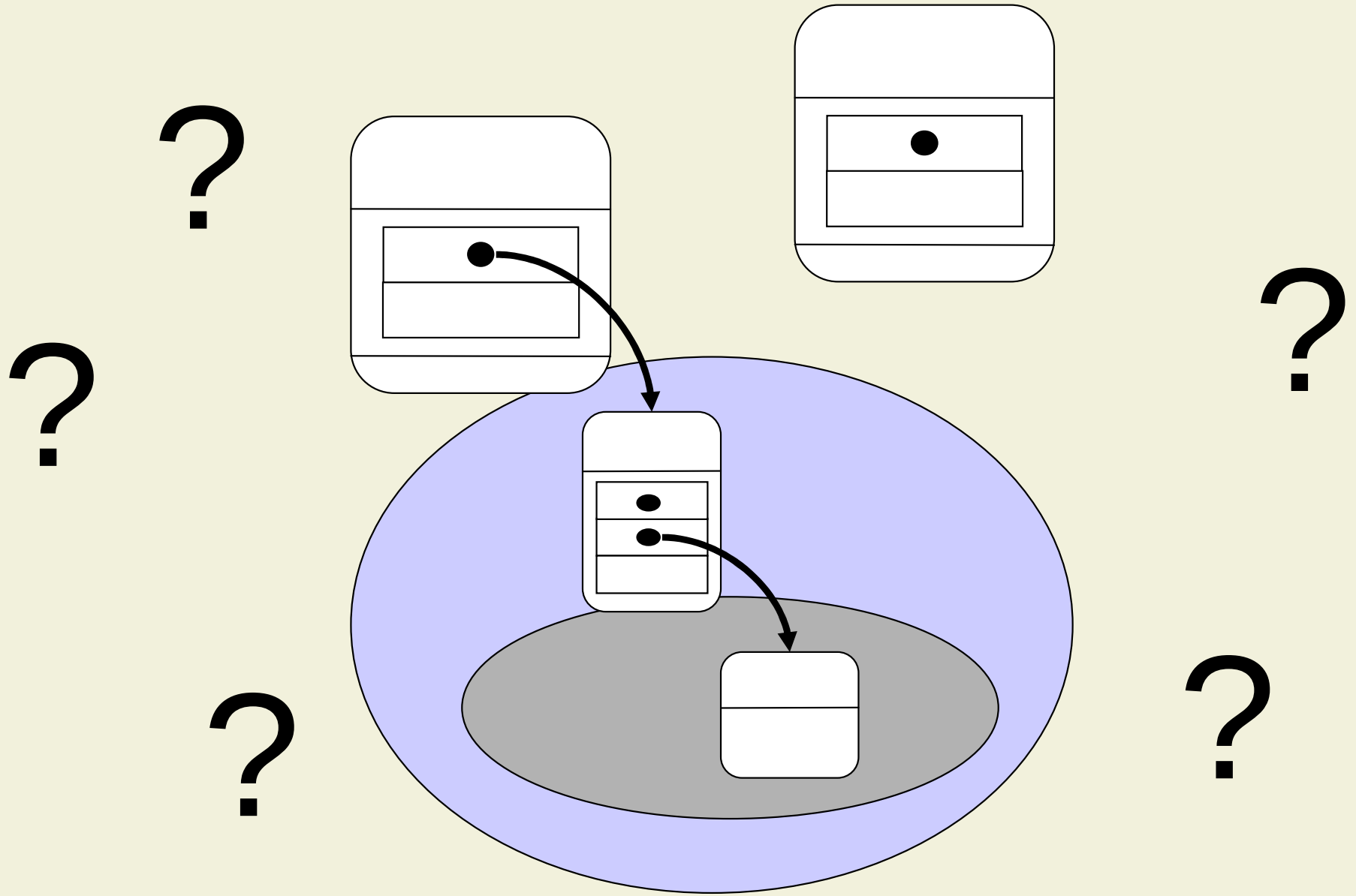
Exercise 3 – LinkedList Main

```
DeleteIterator itd = ll.getDeleteIterator();
itd.moveNext();
itd.moveNext();
itd.delete();
System.out.println("Deleted second element.");
itr = ll.getReadIterator();
i = 0;
while( itr.hasNext() ) { itr.moveNext();
  System.out.println("Element[" +i+ "]: " +
                                  itr.element() );
  ++i;
}
```

Exercise 3 – LinkedList

- Entries are encapsulated
- Readonly access possible
- No external modifications
- Iterator has to delegate the modification to the LinkedList
- LinkedList is in control





Type Scheme Combinator

$$* : TypeScheme \times TypeScheme \rightarrow TypeScheme \cup \{undef\}$$

	grndS(C)	repS(C)	roS(C)	boolS	intS	nullS
grndS(D)	grndS(C)	repS(C)	roS(C)	boolS	intS	nullS
repS(D)	repS(C)	*undef*	roS(C)	boolS	intS	nullS
roS(D)	roS(C)	roS(C)	roS(C)	boolS	intS	nullS

Object Creation

Compilation Error!

```
readonly T v = new readonly T();
```

Objects have to be created in a specific Universe – this is needed e.g. for dynamic casts from readonly to readwrite types.

$$\frac{\mathrm{TS} \preceq_s [\mathrm{v}], \mathrm{TS} \quad \textit{is } grndS \textit{ or } repS}{\vdash \texttt{v=new TS();}}$$

Attribute Access on **this**

```
rep T f;
T v = this.f;
this.f = new rep T();
```

Compilation Error!

$$\frac{[\text{this}] * [\text{f}] \preceq_S [\text{v}]}{\vdash \texttt{v=this.f;}}$$

$$\frac{[\text{this}] \; is \; no \; roS, [\text{e}] \preceq_S [\text{this}] * [\text{f}]}{\vdash \texttt{this.f=e;}}$$

Attribute Access

```
class C { rep T f; … }
C w = …;
rep T v;
v = w.f;
```

Compilation Error!

$$\frac{[\mathrm{f}]\ is\ repS \Rightarrow [\mathrm{w}]\ is\ roS, [\mathrm{w}] * [\mathrm{f}] \preceq_S [\mathrm{v}]}{\vdash \texttt{v=w.f;}}$$

$$\frac{[\mathrm{v}]\ is\ no\ roS, [\mathrm{f}]\ is\ no\ repS, [\mathrm{e}] \preceq_S [\mathrm{v}] * [\mathrm{f}]}{\vdash \texttt{v.f=e;}}$$

Method Calls

```
class C { T m( T par ); … }
rep C w = …;
rep T v;
v = w.m( new rep T() );
```

$$\frac{\mathrm{m}\ \textit{is not functional},\quad \textit{par}(\mathrm{m})\ \textit{is no repS}, \textit{res}(\mathrm{m})\ \textit{is no repS},\quad [\mathrm{w}]\ \textit{is no roS},\quad [\mathrm{e}] \preceq_S [\mathrm{w}] * \textit{par}(\mathrm{m}), [\mathrm{w}] * \textit{res}(\mathrm{m}) \preceq_S [\mathrm{v}]}{\vdash \texttt{v=w.m(e);}}$$

Functional Method Calls

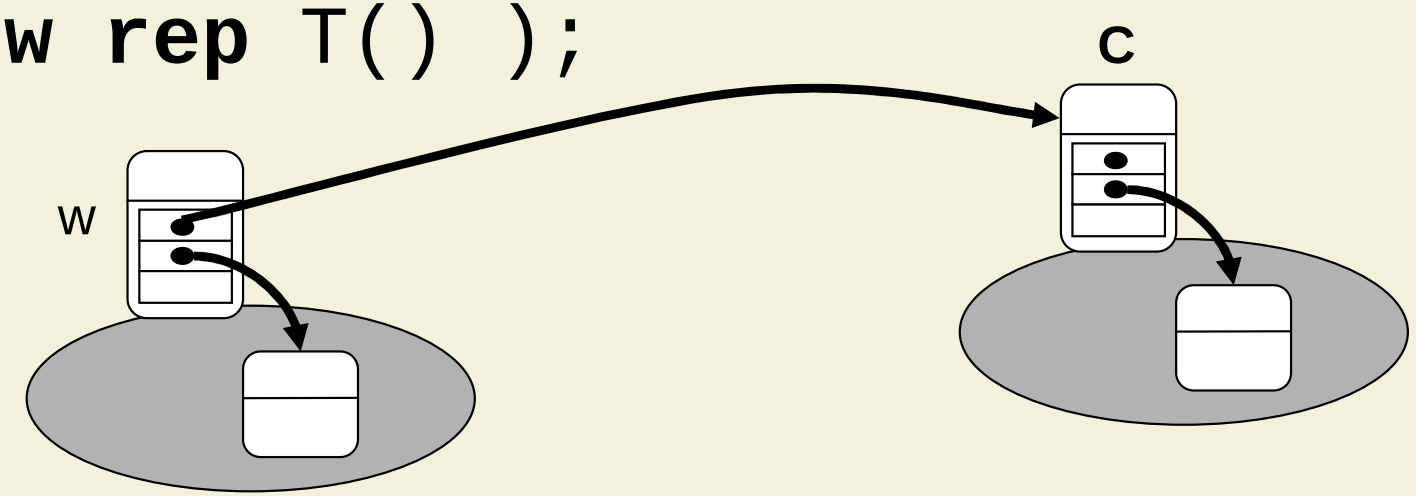
```
class C {
   functional rep T m( readonly T par ); … }
rep C w = …;
rep T v;
v = w.m( new rep T() );
```

Compilation Error!

$$\frac{\mathrm{m}\ \mathit{is}\ \mathit{functional}^5,\quad \mathit{res}(\mathrm{m})\ \mathit{is}\ \mathit{repS} \Rightarrow [\mathrm{w}]\ \mathit{is}\ \mathit{roS},\quad [\mathrm{e}] \preceq_S [\mathrm{w}] * \mathit{par}(\mathrm{m}), [\mathrm{w}] * \mathit{res}(\mathrm{m}) \preceq_S [\mathrm{v}]}{\vdash \texttt{v=w.m(e);}}$$

Parameters of Functional Methods

```
class C {
   functional void m( rep T par ); … }
readonly C w = …;
w.m( new rep T() );
```



➔ all parameters of functional methods have to be `readonly`

Method Calls on **this**

```
class C { rep T m( rep T par ); … }
rep T v;
v = this.m( new rep T() );
```

$$\frac{\mathrm{m}\ \textit{is not functional},\quad [\mathrm{e}] \preceq_s [\mathrm{this}] * \textit{par}(\mathrm{m}), [\mathrm{this}] * \textit{res}(\mathrm{m}) \preceq_s [\mathrm{v}]}{\vdash \texttt{v=this.m(e);}}$$

Functional Method Calls on **this**

```
class C {
  functional rep T m( readonly T par );
  … }
rep T v;
v = this.m( new rep T() );
```

$$\frac{\mathrm{m}\ \mathit{is}\ \mathit{functional},\quad [\mathrm{e}] \preceq_s [\mathrm{this}] * \mathit{par}(\mathrm{m}), [\mathrm{this}] * \mathit{res}(\mathrm{m}) \preceq_s [\mathrm{v}]}{\vdash \texttt{v=this.m(e);}}$$