The Programming Language Gilgul

Pascal Costanza

University of Bonn
Institute of Computer Science III

http://www.pascalcostanza.de

© 2001 Pascal Costanza, Institute of Computer Science III, University of Bonn

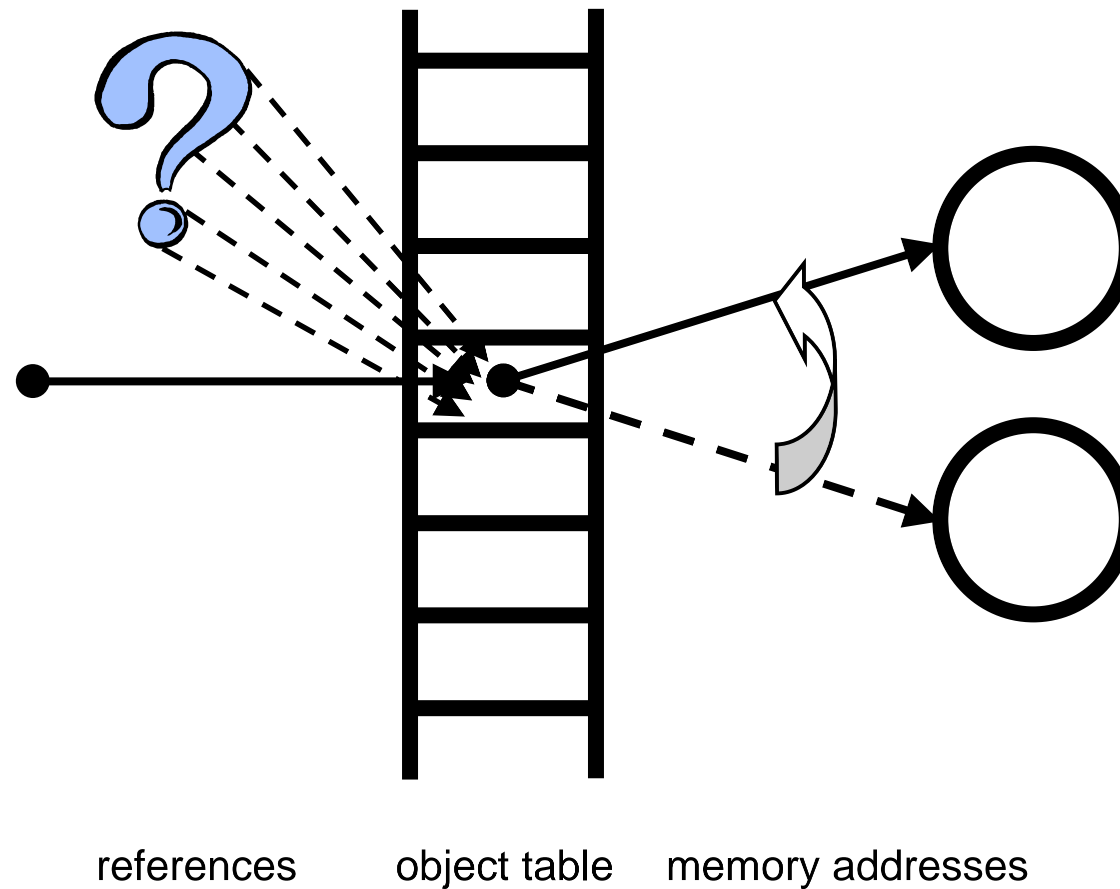
The TAILOR Project:
How to Adapt Software During Runtime?

- In principle, by manual redirection of references

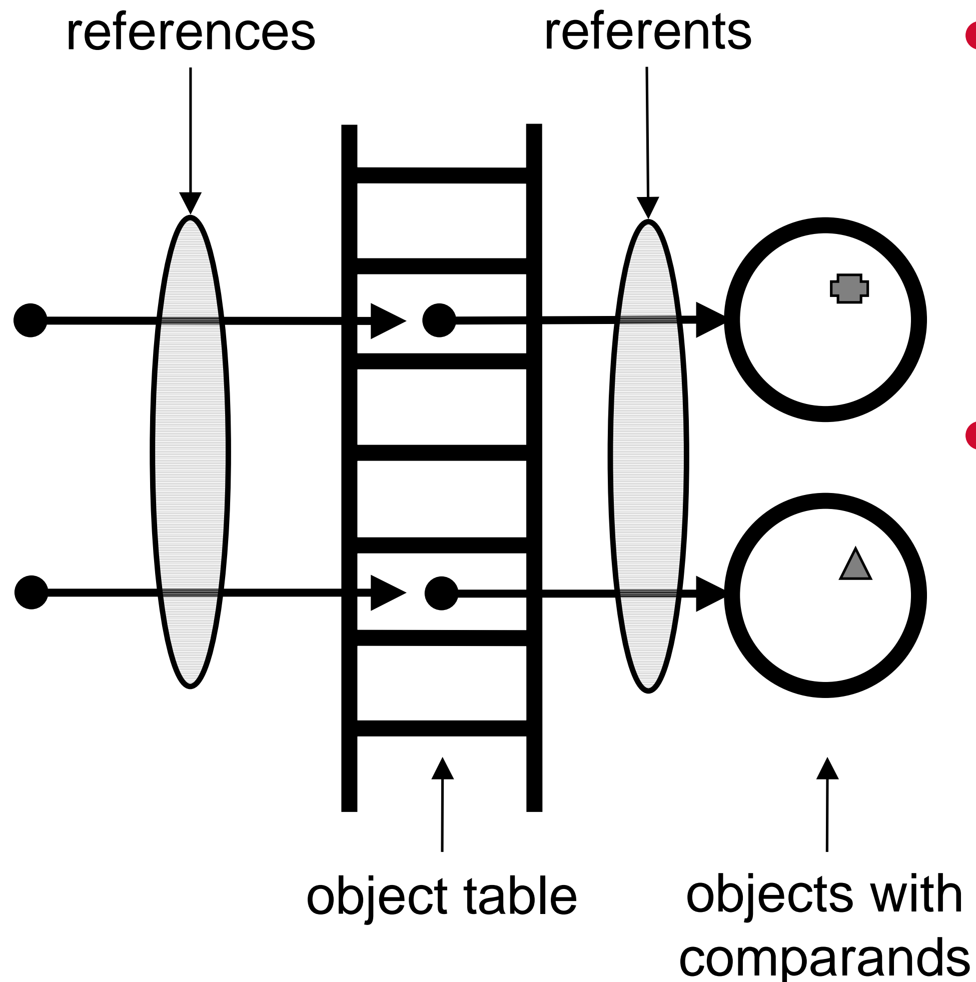
- Consistency problems:
  - All references to an object must be known.
  - They have to be redirected one by one.

- Atomic replacement of objects is needed!

The TAILOR Project: How to Adapt Software During Runtime?

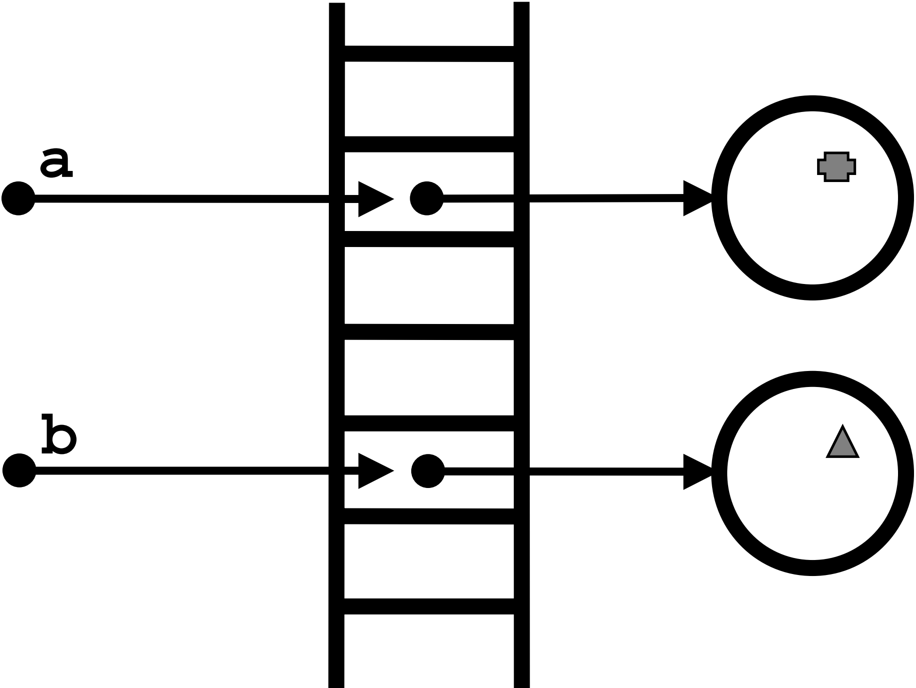


The Gilgul Model



- Strict separation of concerns!
  - References are never used for comparison.
  - Comparands are never used for reference.
- This allows new operations to be introduced in a semantically clean way.

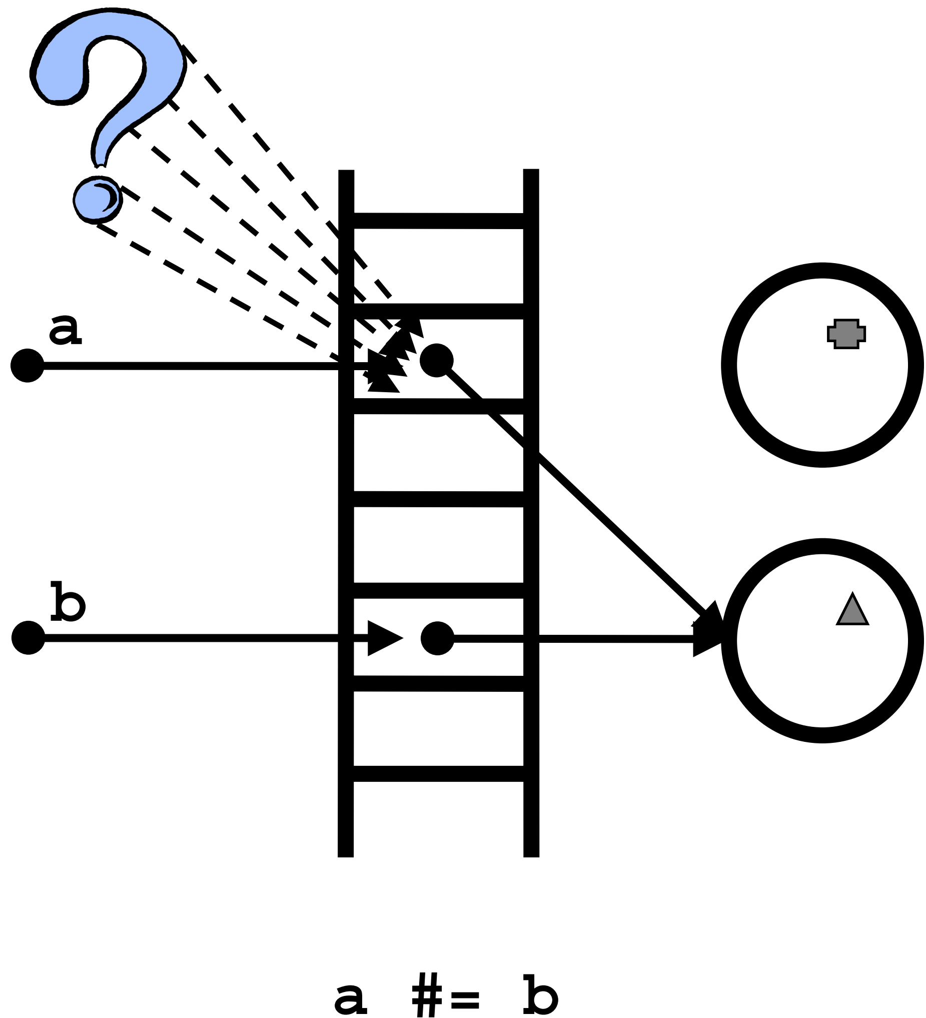
Referent Assignment in Gilgul




```
a #= b
```

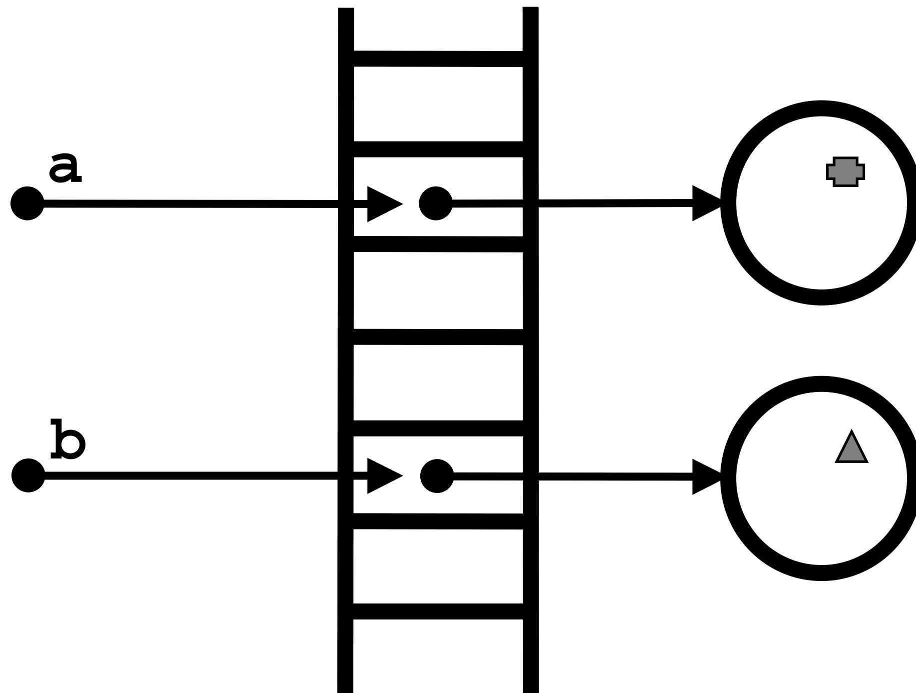
© 2001 Pascal Costanza, Institute of Computer Science III, University of Bonn

Referent Assignment in Gilgul



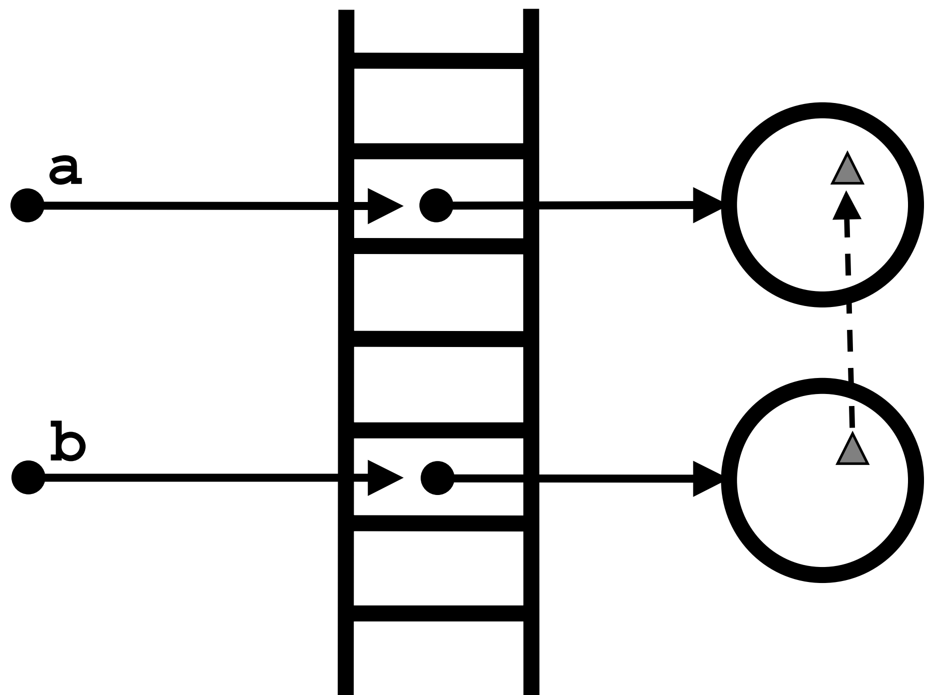
- Referent assignment does not affect the semantics of comparison.
  - `a == b` always means `a.comparand == b.comparand`
- Affects all other references pointing to the same table entry.
  - Thus allows for atomic object replacement.

Comparand Assignment in Gilgul



`a.comparand = b.comparand`

Comparand Assignment in Gilgul



- Can be used when several objects represent a conceptual entity.
  - For example: Decorator Pattern, Role Object Pattern, etc.
- An example: A decorator that counts accesses to a file.

```
a.comparand = b.comparand
```

Dealing With Critical State

- The following code seems to be problematic...
  - ```
    File f = new File("myFile.dat");
    f.open();

    ...

    f #= ...;

    ...

    f.readBytes(...);
    ```

© 2001 Pascal Costanza, Institute of Computer Science III, University of Bonn
    ```

Dealing With Critical State

- ...but decorators smoothly reuse state.

```
File f = new File("myFile.dat");
f.open();

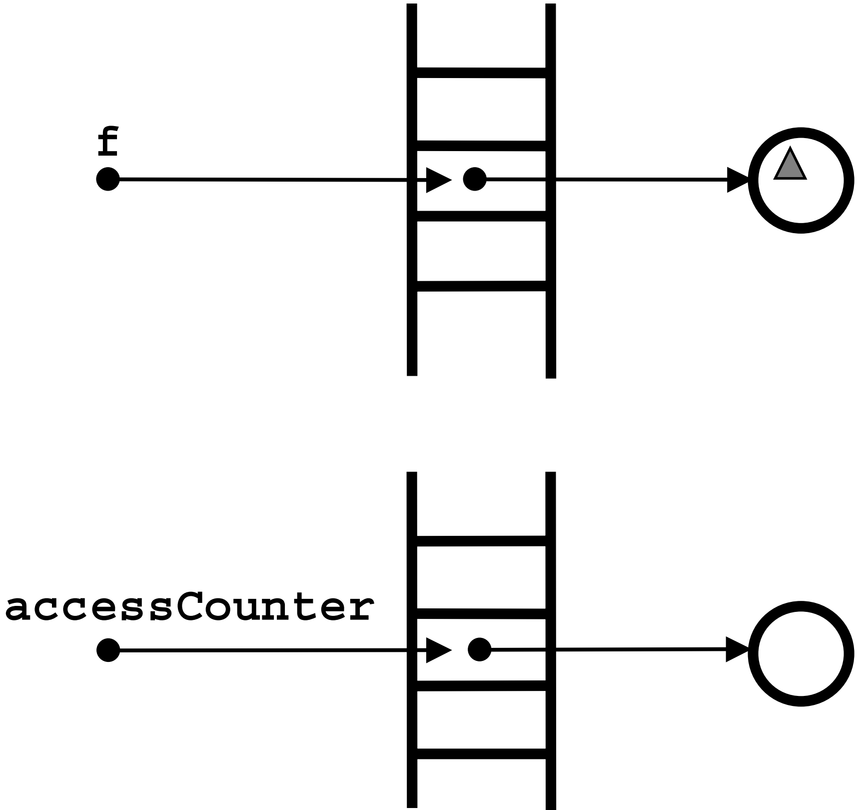
...

f #= new AccessCounterDecorator(f);

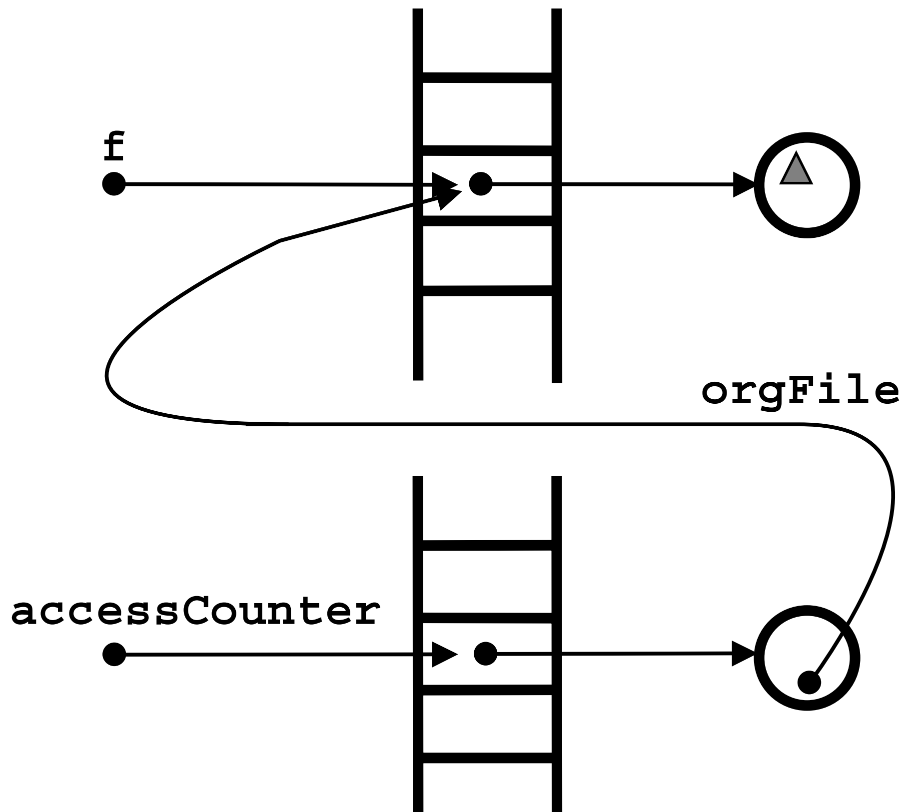
...

f.readBytes(...);
```

Naive Replacement May Lead to Cycles.

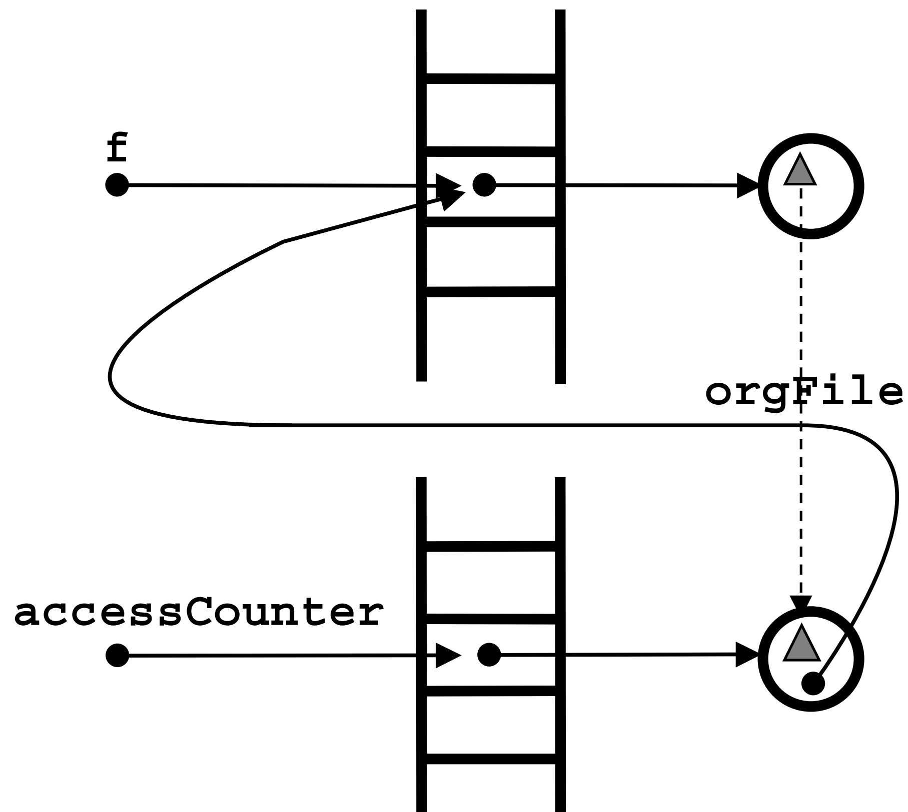


Naive Replacement May Lead to Cycles.




```
accessCounter.orgFile = f;
```

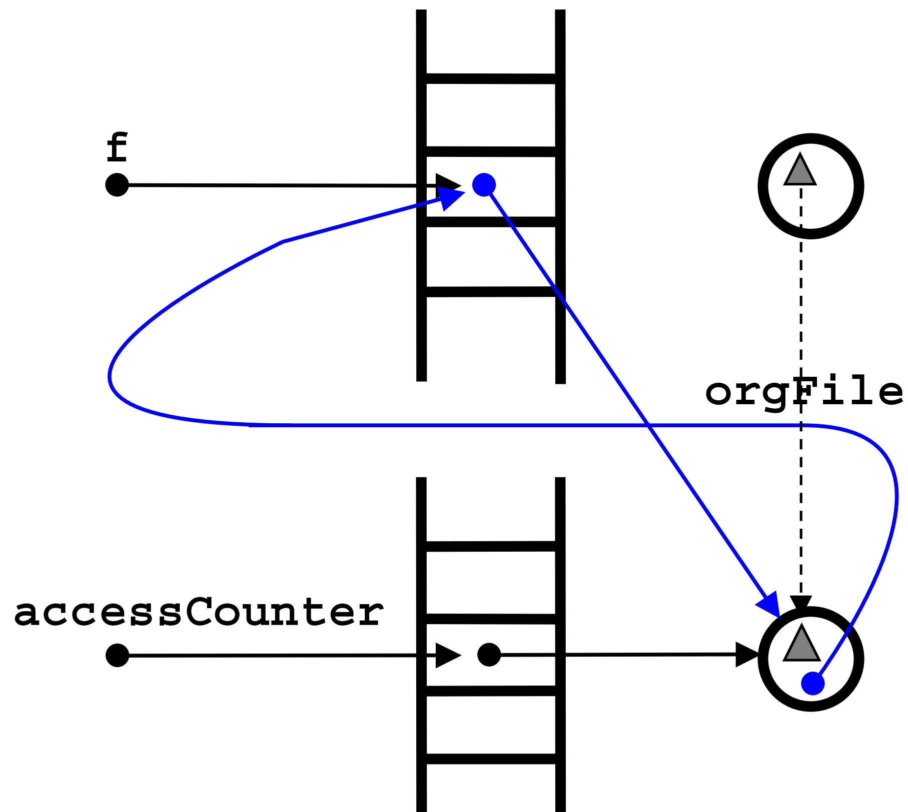
Naive Replacement May Lead to Cycles.




```
accessCounter.orgFile = f;

accessCounter.comparand
                   = f.comparand;
```

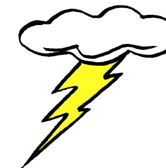
Naive Replacement May Lead to Cycles.




```
accessCounter.orgFile = f;

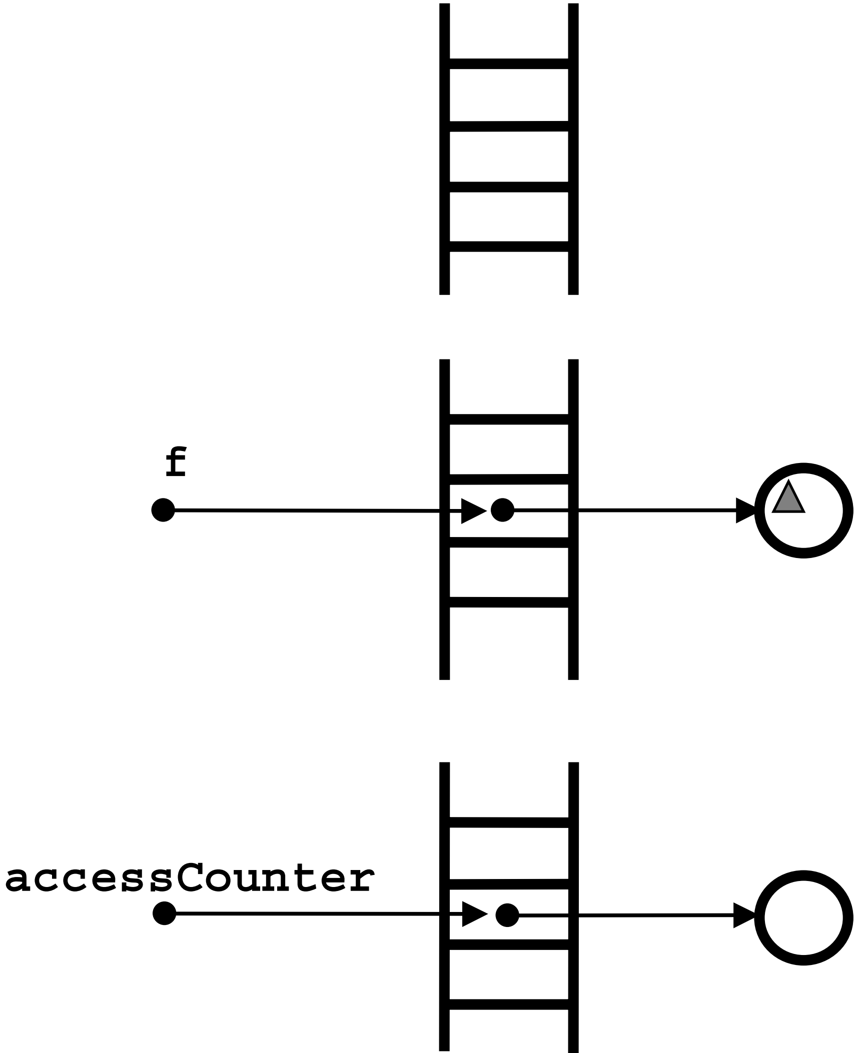
accessCounter.comparand
               = f.comparand;

f #= accessCounter;
```

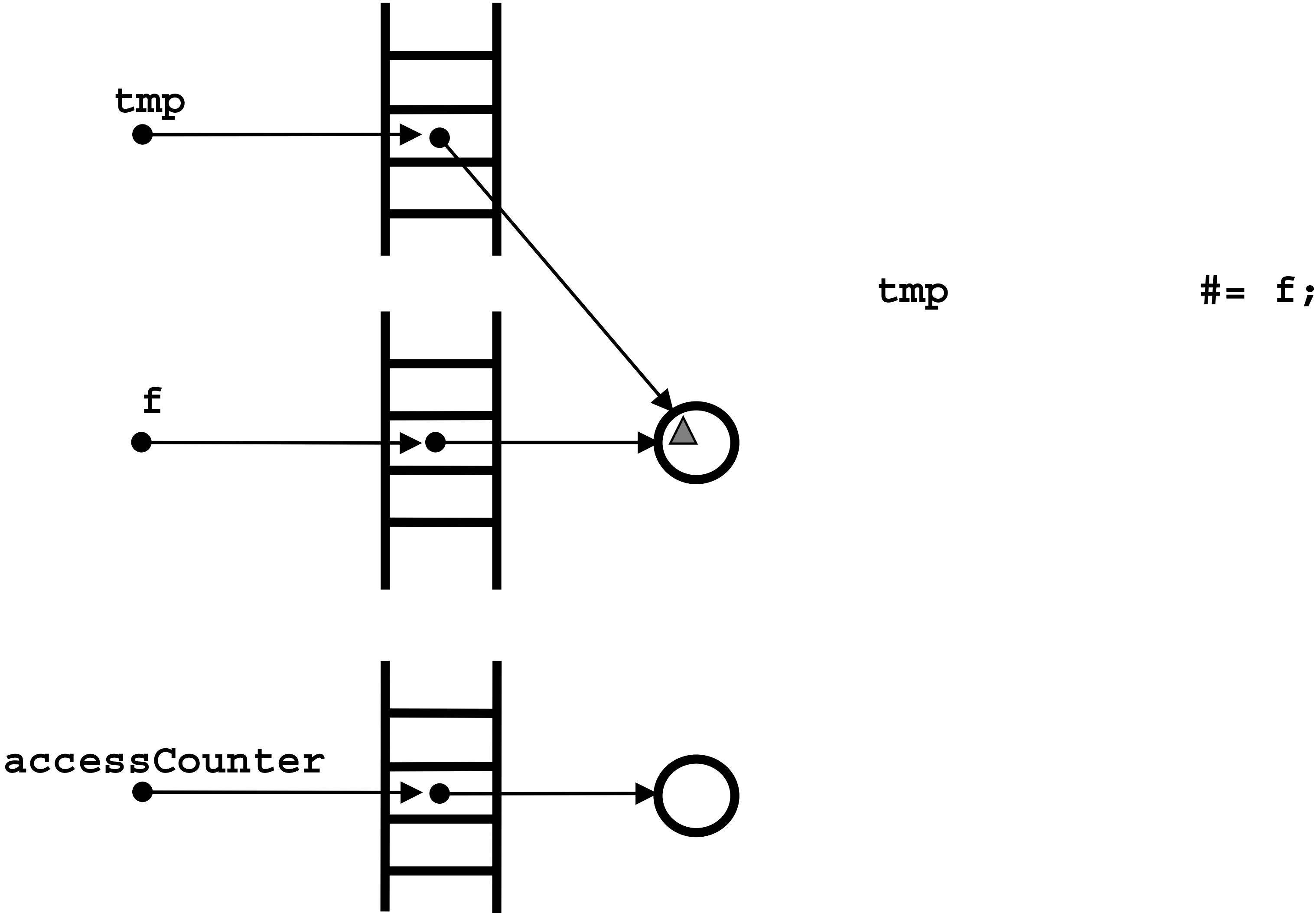


© 2001 Pascal Costanza, Institute of Computer Science III, University of Bonn

Correct Replacement Without Cycles

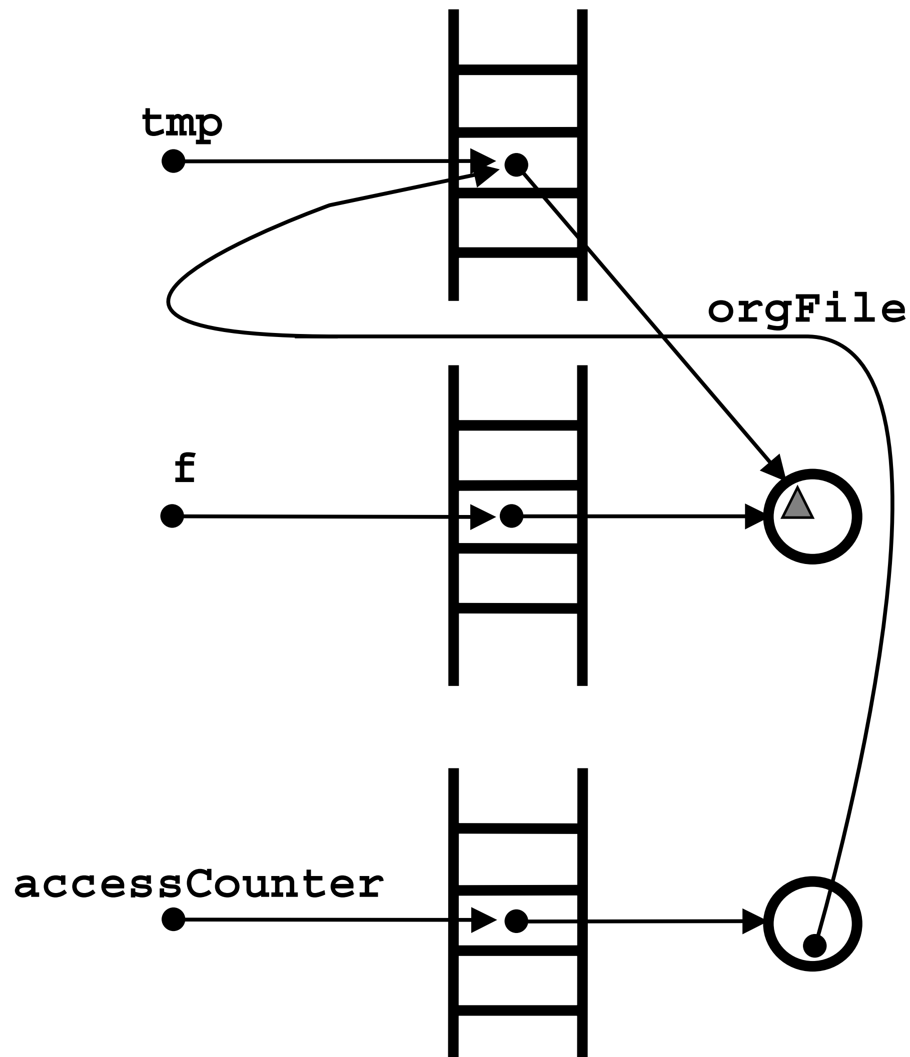


Correct Replacement Without Cycles



`tmp #= f;`

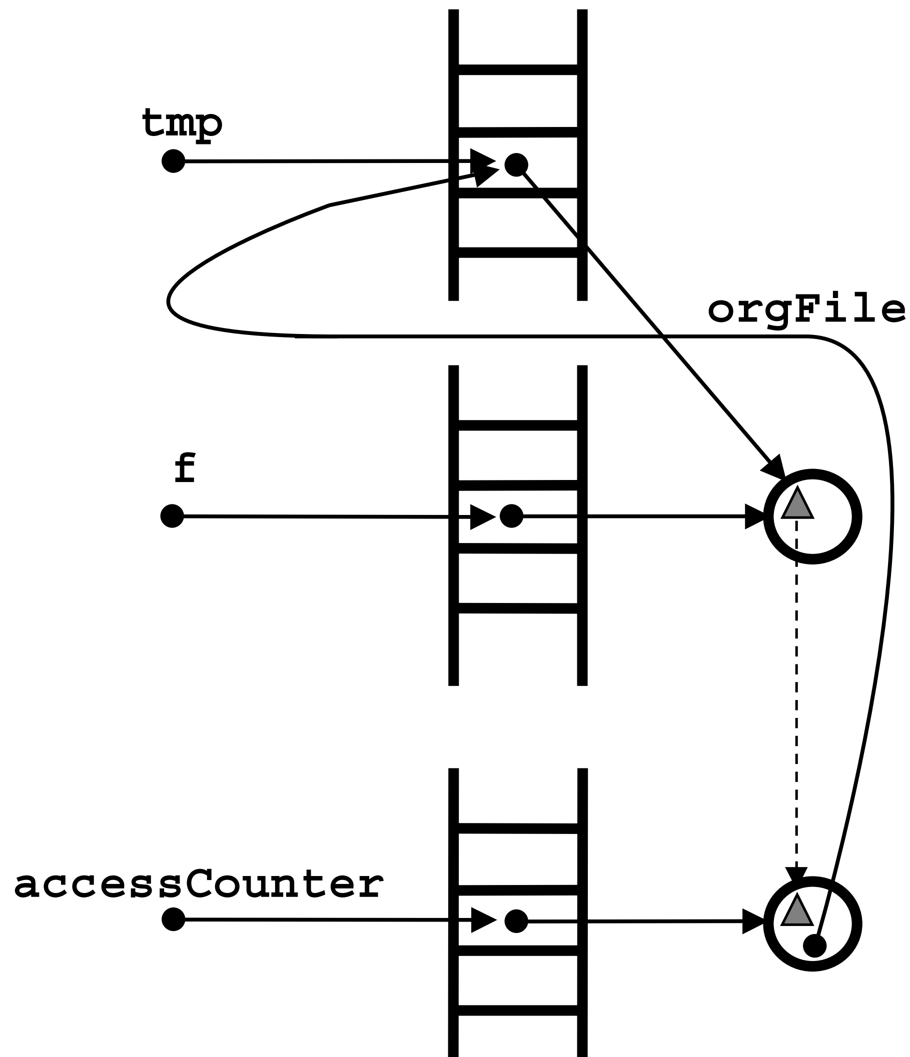
Correct Replacement Without Cycles




```
tmp                  #= f;

accessCounter.orgFile   =   tmp;
```

Correct Replacement Without Cycles

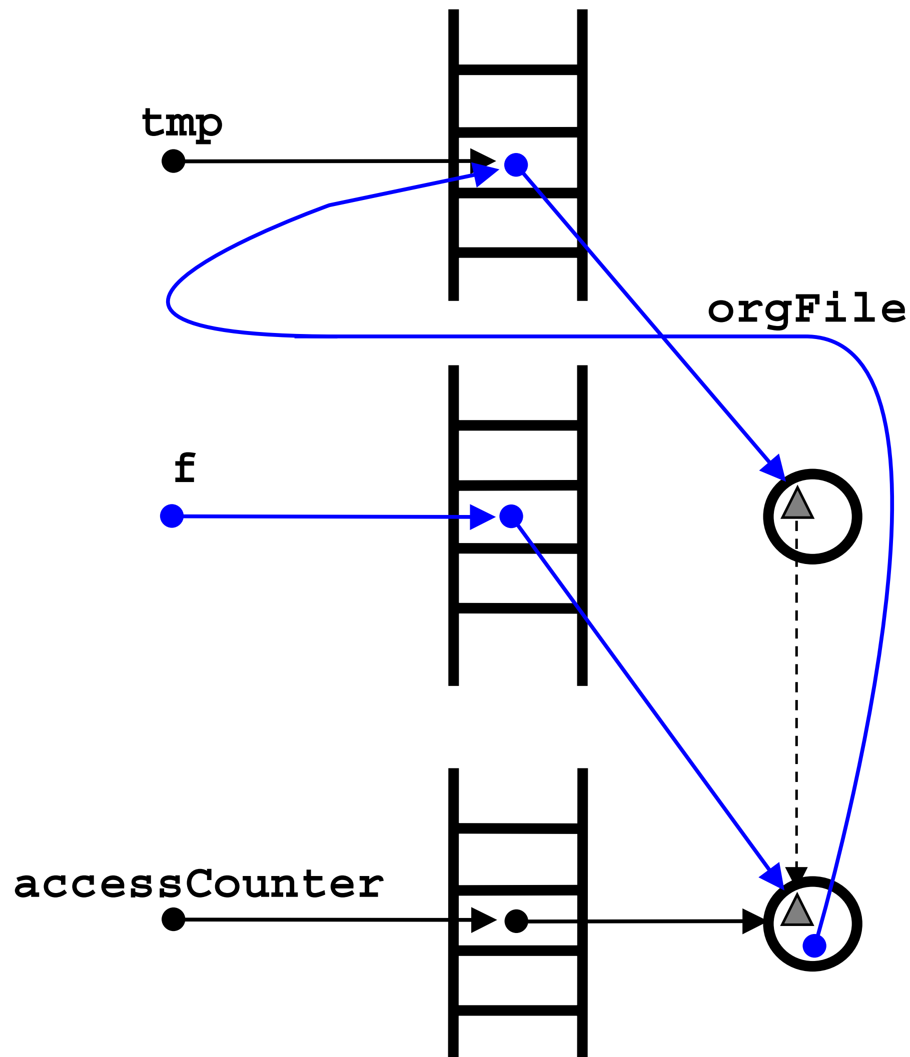



```
tmp              #= f;

accessCounter.orgFile  =  tmp;

accessCounter.comparand  =
                tmp.comparand;
```

Correct Replacement Without Cycles




```
tmp                    #= f;

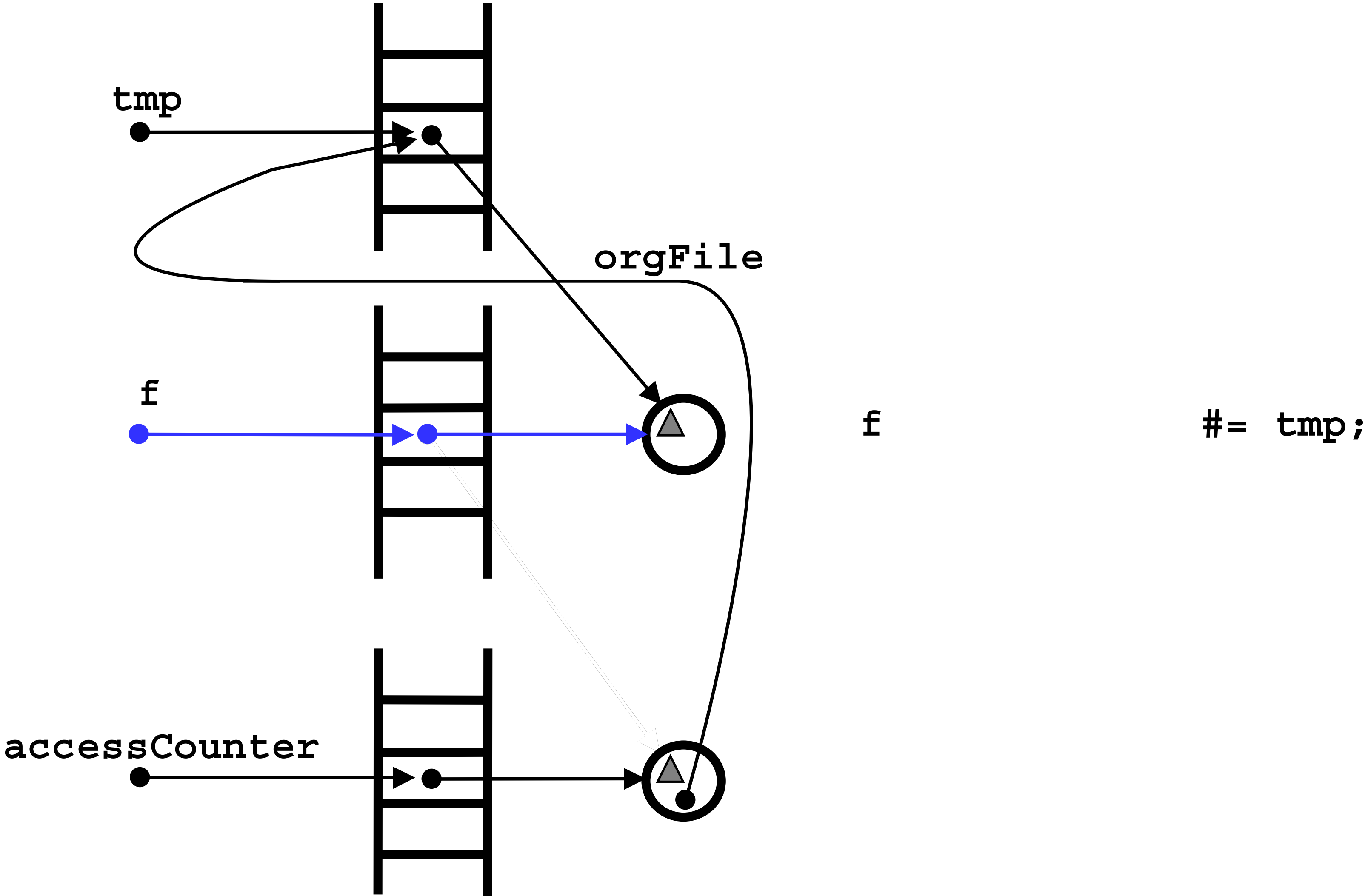
accessCounter.orgFile   =  tmp;

accessCounter.comparand   =
                       tmp.comparand;

f                      #= accessCounter;
```

- There is no need to recover state!

...and back again!



f #= tmp;

© 2001 Pascal Costanza, Institute of Computer Science III, University of Bonn

Other Design Issues

- Declaration of Restrictions
  - Even in Gilgul, the Security Manager should not be replaceable!
- Replacement of Active Objects
  - in the presence of active methods
  - both in multi-threaded and single-threaded scenarios
- Type Soundness of Referent Assignment
  - **`implementationonly`** classes effectively widen the range of unanticipated adaptations

Other Topics

- The Comparand Pattern (EuroPLoP 2001)
  - describes known uses of comparands
    - CORBA, RMI, JPDA, etc.
- Implementation (work in progress)
  - based on the Kaffe Virtual Machine
  - Goal: Use direct pointers as long as possible!
- Formal Proof of Type Soundness (future work)

Visit our websites!

- TAILOR Project
  - http://javalab.cs.uni-bonn.de/research/tailor/
- My homepage
  - http://www.pascalcostanza.de