
ORACLE®

ORACLE

Partial Escape Analysis and Scalar Replacement for Java

Lukas Stadler

VM Research Group, Oracle Labs

The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Partial Escape Analysis and Scalar Replacement for Java

Lukas Stadler
Thomas Würthinger
Oracle Labs

Hanspeter Mössenböck
Johannes Kepler University Linz, Austria

Graal?

http://openjdk.java.net/projects/graal/

graal-dev@openjdk.java.net

```
$ hg clone http://hg.openjdk.java.net/graal/graal
$ cd graal
$ ./mx build
$ ./mx ideinit
$ ./mx vm <arguments>
```

- Graal Resources

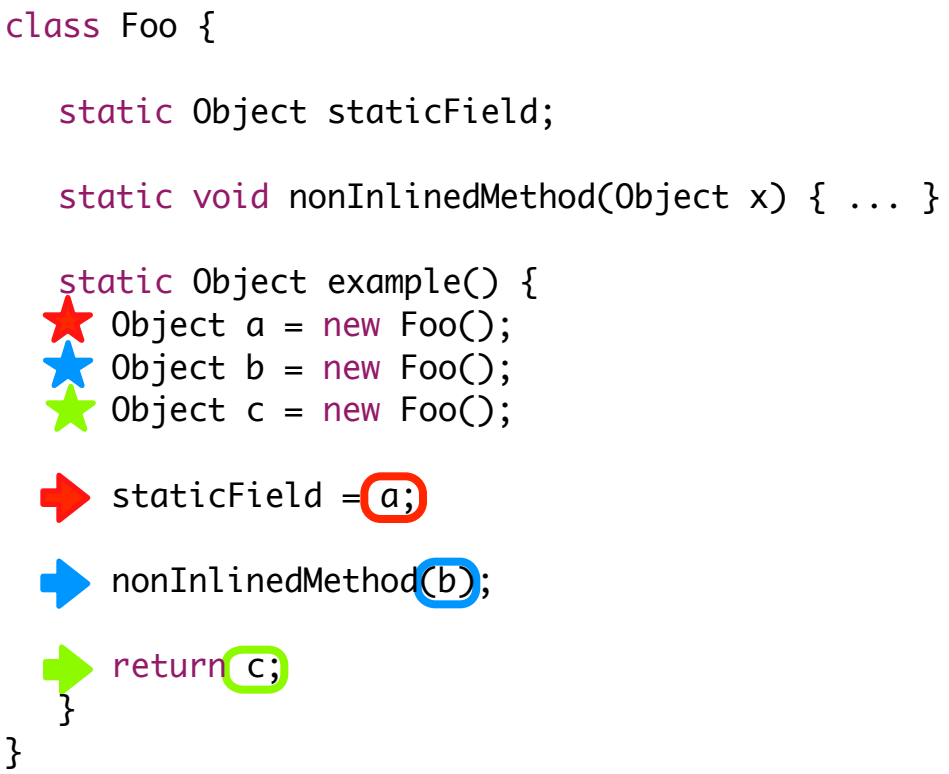
https://wiki.openjdk.java.net/display/Graal/

graal-dev@openjdk.java.net

Escape Analysis

- Escape Analysis:
  Analyzes where references to new objects flow

- Looks for “escapes”
  - Method call parameters
  - Static fields
  - Return value
  - Throws
  - …



Escape Analysis - Optimization Opportunities

- Allocated object is scope/method local
  - Scalar Replacement: replace fields with local variables
- Allocated object is thread local
  - Lock Removal: no other thread can see the object, no locking required
  - Stack Allocation: automatic stack management, destroyed on return
- Allocated object escapes
  - Escapes to other threads/methods, no optimizations possible

Escape Analysis - Example

```
public static Car getCached(int hp, String name) {
     Car car = new Car(hp, name, null);
     Car cacheEntry = null;
     for (int i = 0; i < cache.length; i++) {
          if (car.hp == cache[i].hp &&
                car.name == cache[i].name) {
               cacheEntry = cache[i];
               break;
          }
     }
     if (cacheEntry != null) {
          return cacheEntry;
     } else {
          return null;
     }
}
```

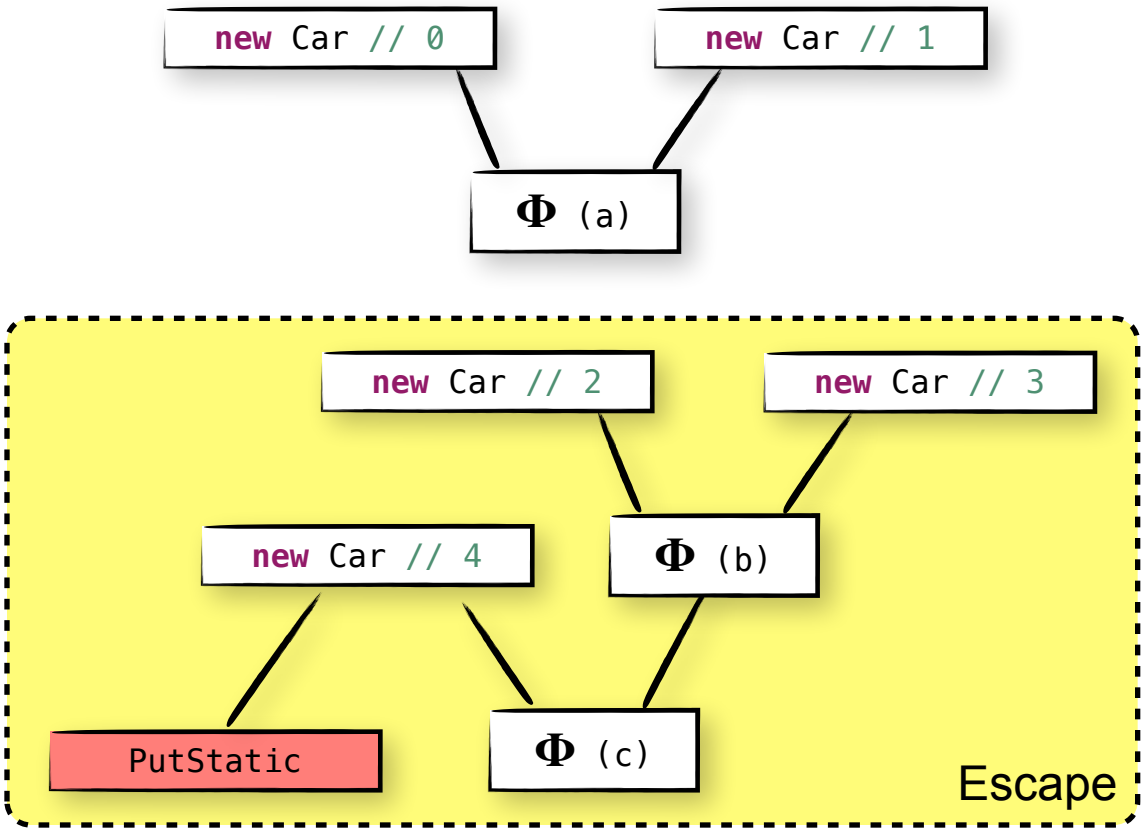
Escape Analysis - Example

```
public static Car getCached(int hp, String name) {
     Car cacheEntry = null;
     for (int i = 0; i < cache.length; i++) {
          if (hp == cache[i].hp &&
                name == cache[i].name) {
               cacheEntry = cache[i];
               break;
          }
     }
     if (cacheEntry != null) {
          return cacheEntry;
     } else {
          return null;
     }
}
```

- `new Car(...)` does not escape
- Allocation is removed
- Field loads replaced with values

Escape Analysis - e.g. Equi-Escape Sets

```
Car a, b, c;
if (...) {
     a = new Car(...) // 0
} else {
     a = new Car(...) // 1
}
if (...) {
     b = new Car(...) // 2
} else {
     b = new Car(...) // 3
}
if (...) {
     tmp = new Car(...) // 4
     staticField = tmp;
     c = tmp;
} else {
     c = b;
}
```



Thomas Kotzmann and Hanspeter Mössenböck. 2005. Escape analysis in the context of dynamic compilation and deoptimization. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments* (VEE '05).

Partial Escape Analysis

```
public static Car getCached(int hp, String name) {
    Car car = new Car(hp, name, null);
    Car cacheEntry = null;
    for (int i = 0; i < cache.length; i++) {
        if (car.hp == cache[i].hp &&
                car.name == cache[i].name) {
            cacheEntry = cache[i];
            break;
        }
    }
    if (cacheEntry != null) {
        return cacheEntry;
    } else {
        addToCache(car);
        return car;
    }
}
```

Partial Escape Analysis

```
public static Car getCached(int hp, String name) {
     Car cacheEntry = null;
     for (int i = 0; i < cache.length; i++) {
          if (hp == cache[i].hp &&
                 name == cache[i].name) {
               cacheEntry = cache[i];
               break;
          }
     }
     if (cacheEntry != null) {
          return cacheEntry;
     } else {
          Car car = new Car(hp, name, null);
          addToCache(car);
          return car;
     }
}
```

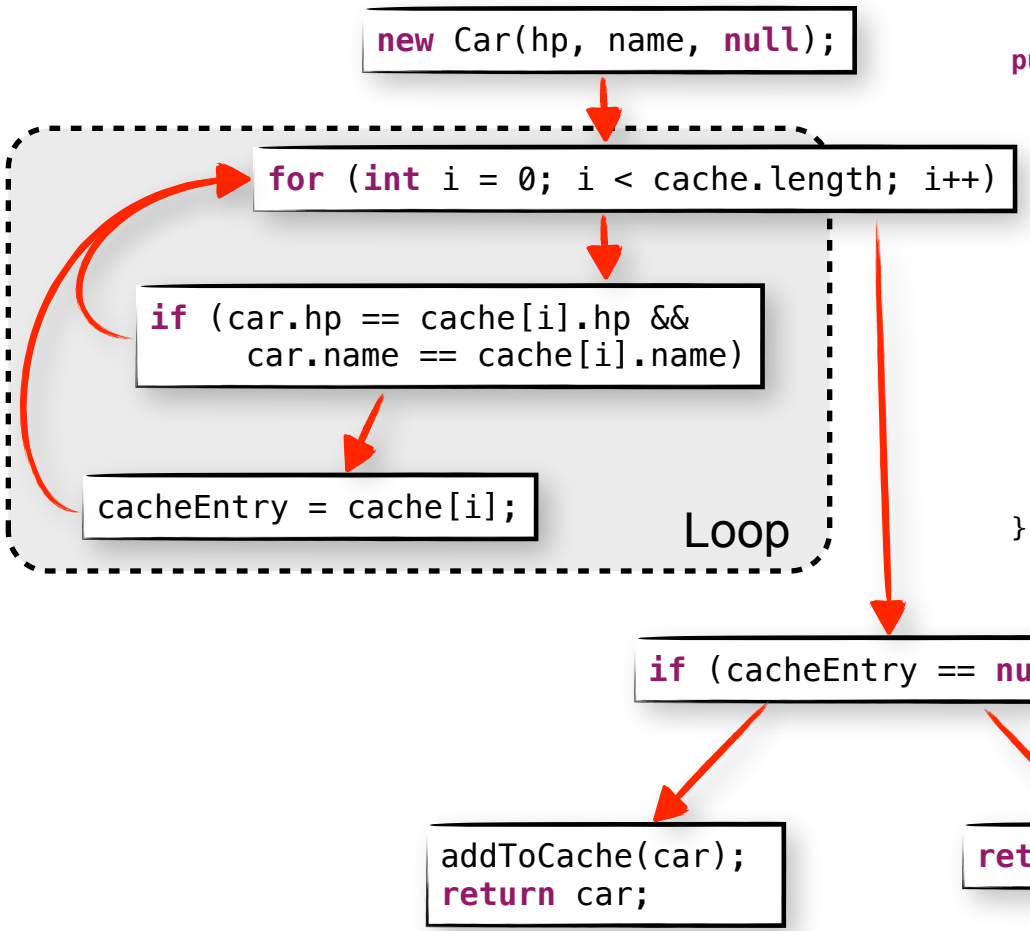
probability: ?%

- `new Car(...)` escapes at:
  - `addToCache(car);`
  - `return car;`
- Might be a very unlikely path
- No allocation in frequent path

Partial Escape Analysis

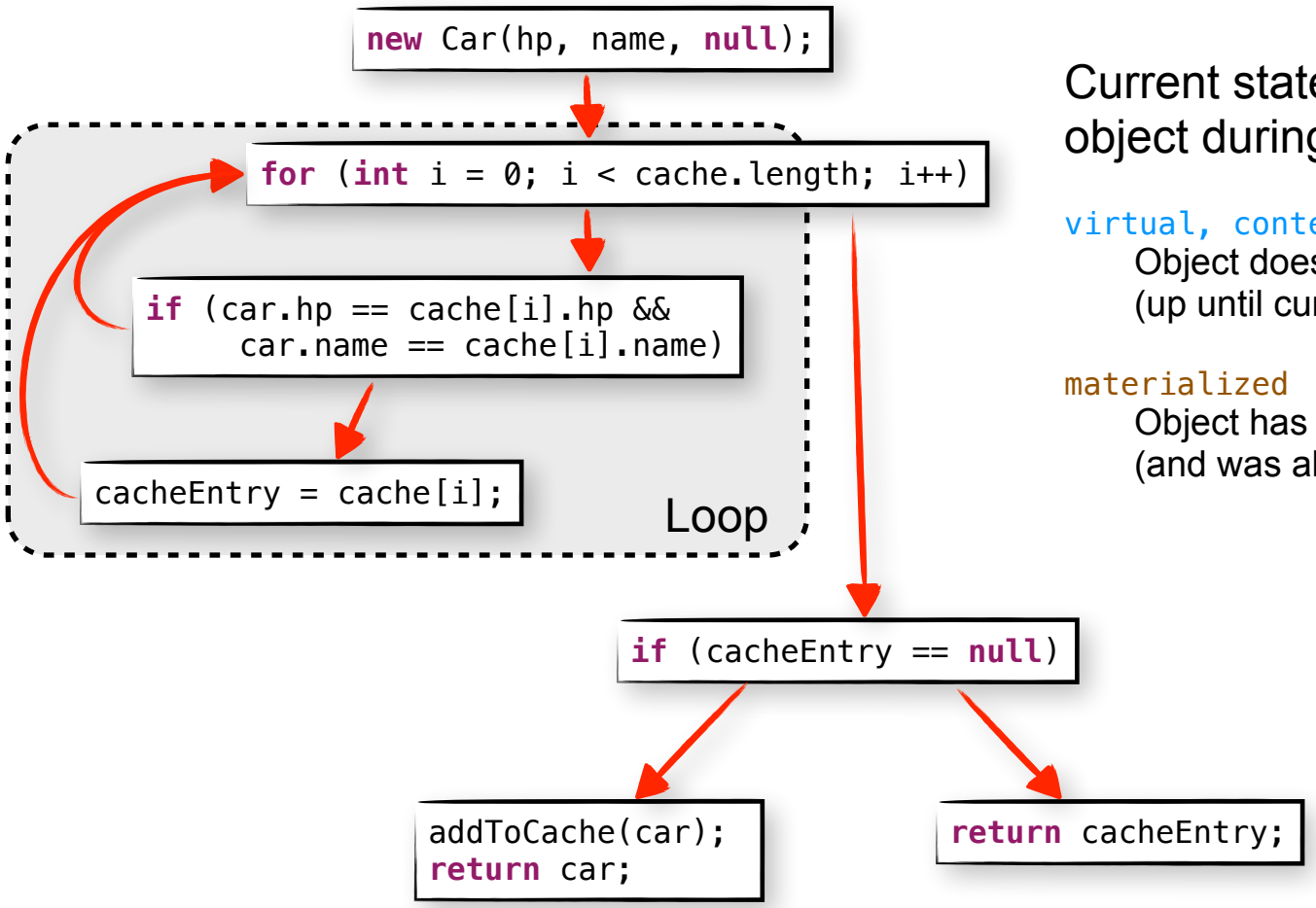
- Escape Analysis (EA): either remove allocation or not
- Partial Escape Analysis (PEA): push allocations into infrequent paths
  - Which often allows removal of other object allocations
- PEA is (inherently) control-flow sensitive
  - Analysis performs iteration over CFG

Partial Escape Analysis - Iteration




```
public static Car getCached(int hp, String name) {
      Car car = new Car(hp, name, null);
      Car cacheEntry = null;
      for (int i = 0; i < cache.length; i++) {
            if (car.hp == cache[i].hp &&
                  car.name == cache[i].name) {
                  cacheEntry = cache[i];
                  break;
            }
      }
      if (cacheEntry != null) {
            return cacheEntry;
      } else {
            addToCache(car);
            return car;
      }
}
```

Partial Escape Analysis - Iteration

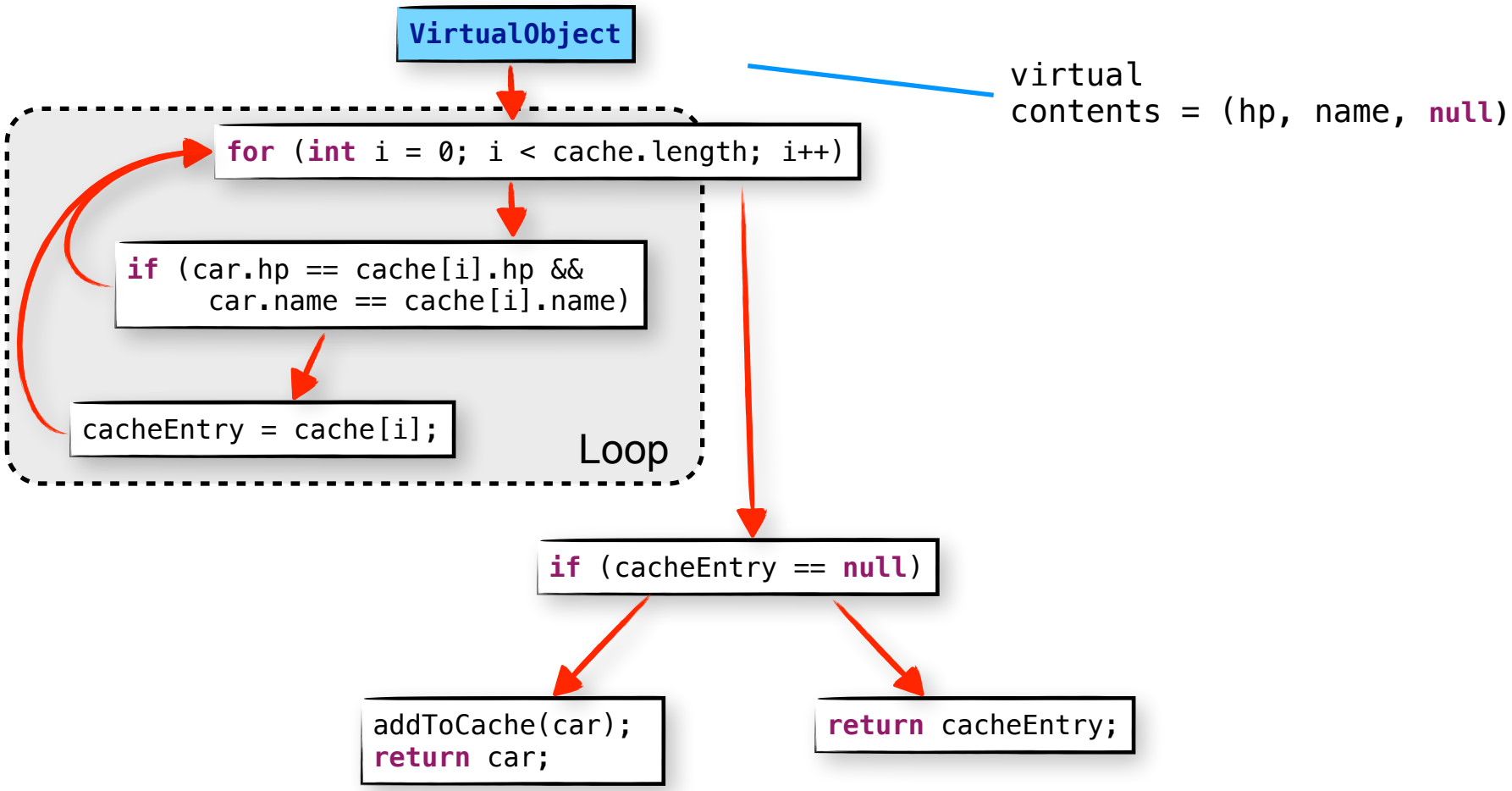


Current state of Escape Analyzed object during CFG iteration:

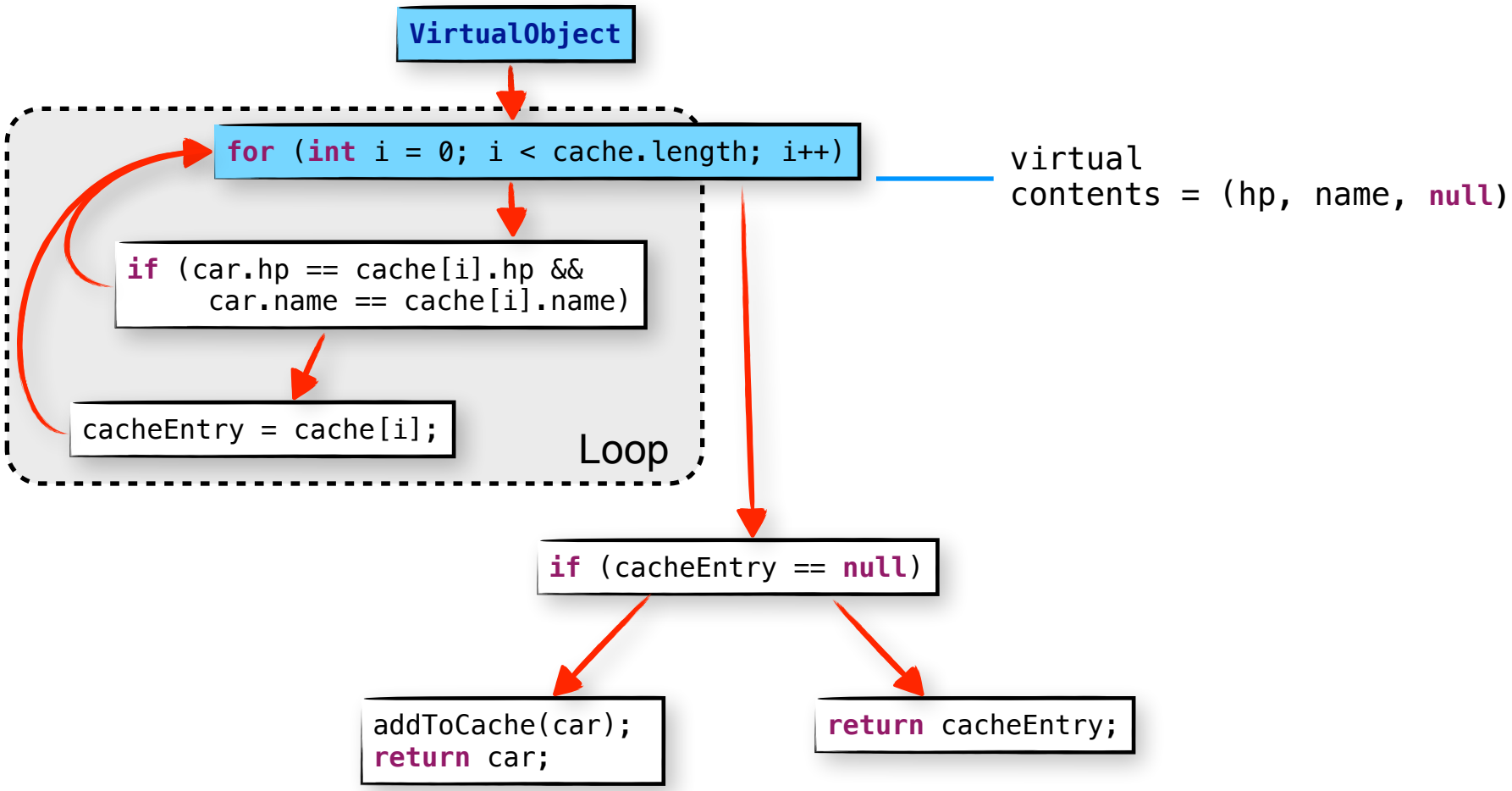
`virtual, contents = (...)`
Object does not escape
(up until current point)

`materialized`
Object has escaped
(and was allocated)

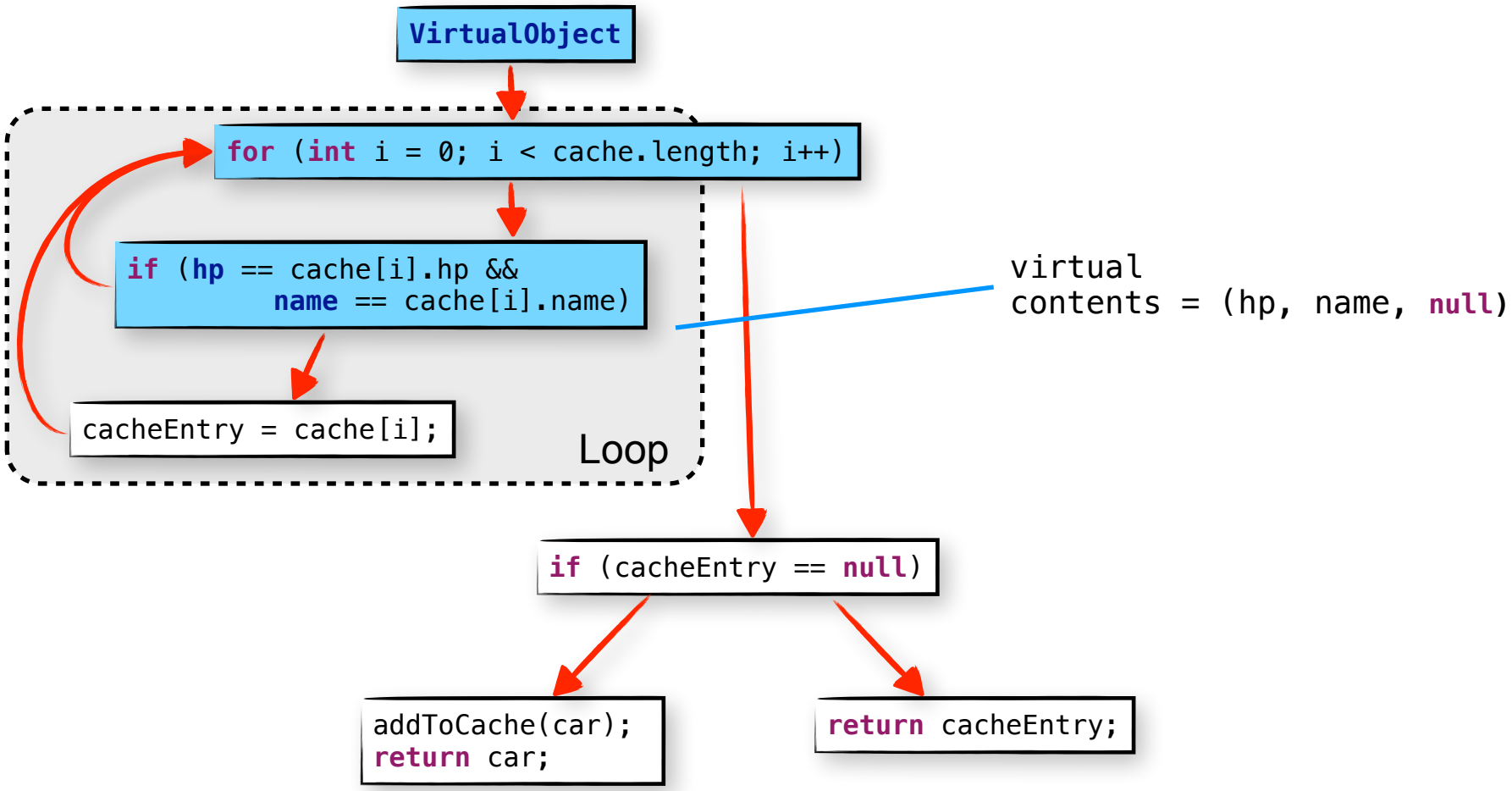
Partial Escape Analysis - Iteration



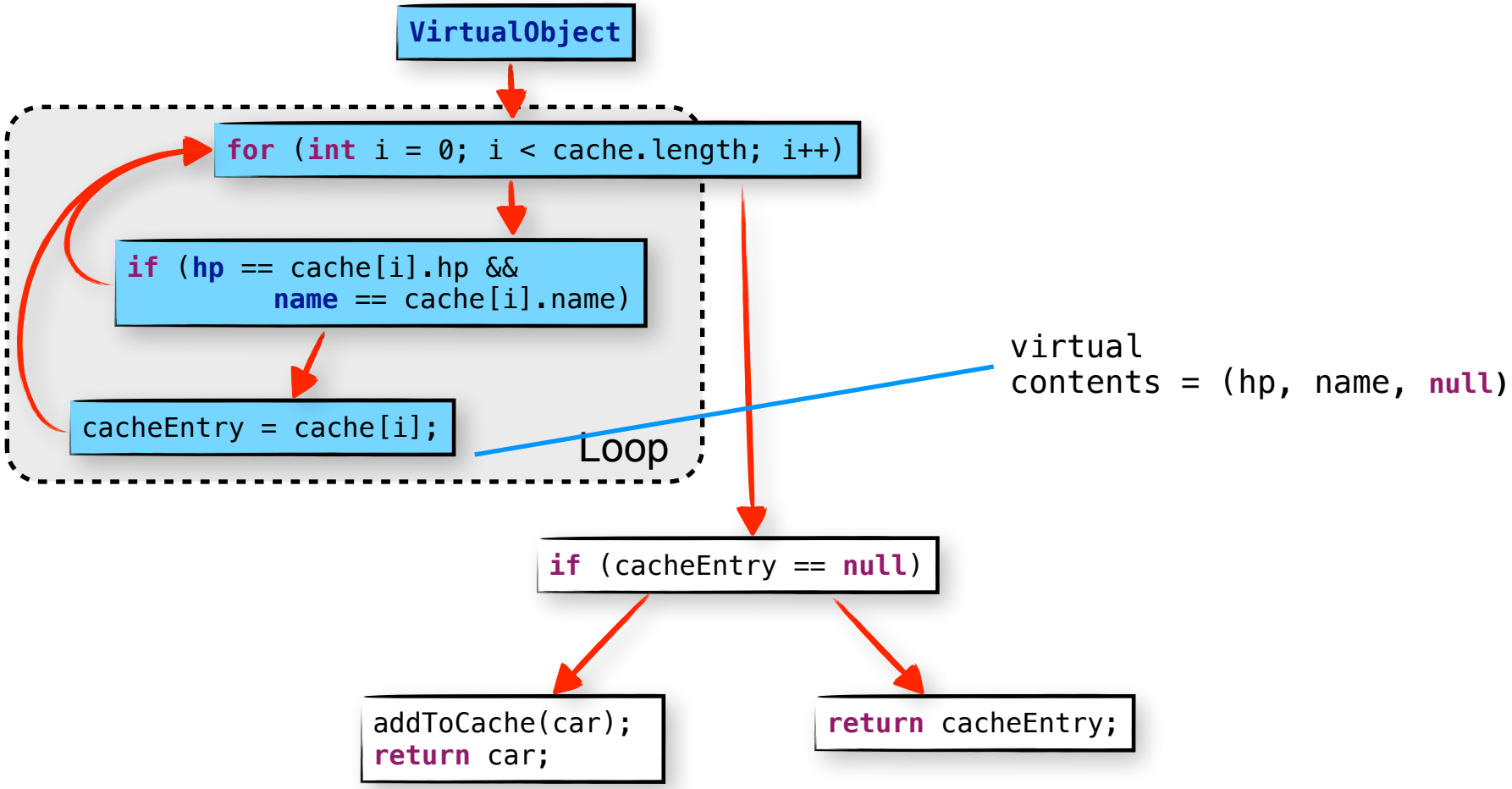
Partial Escape Analysis - Iteration



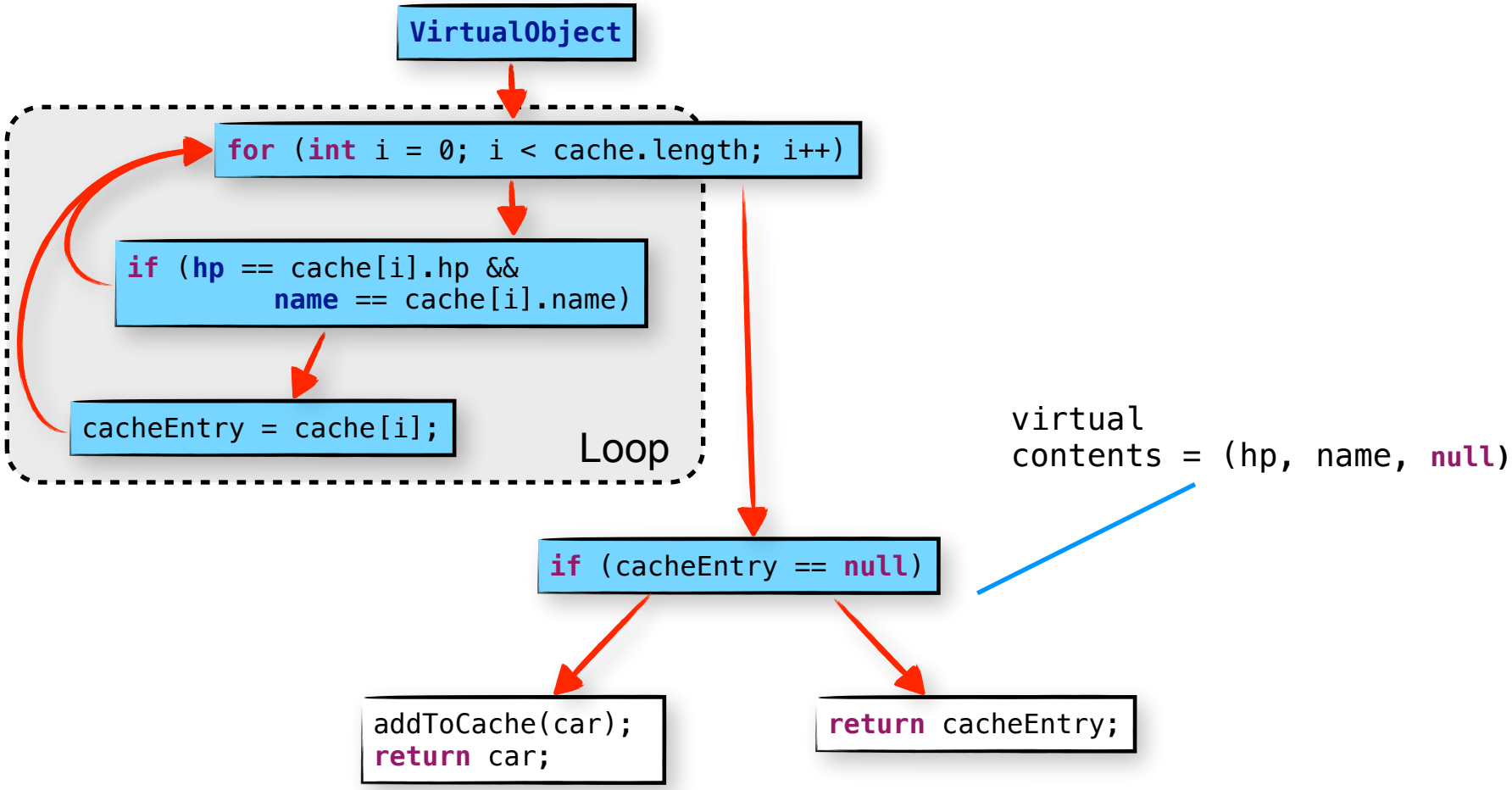
Partial Escape Analysis - Iteration



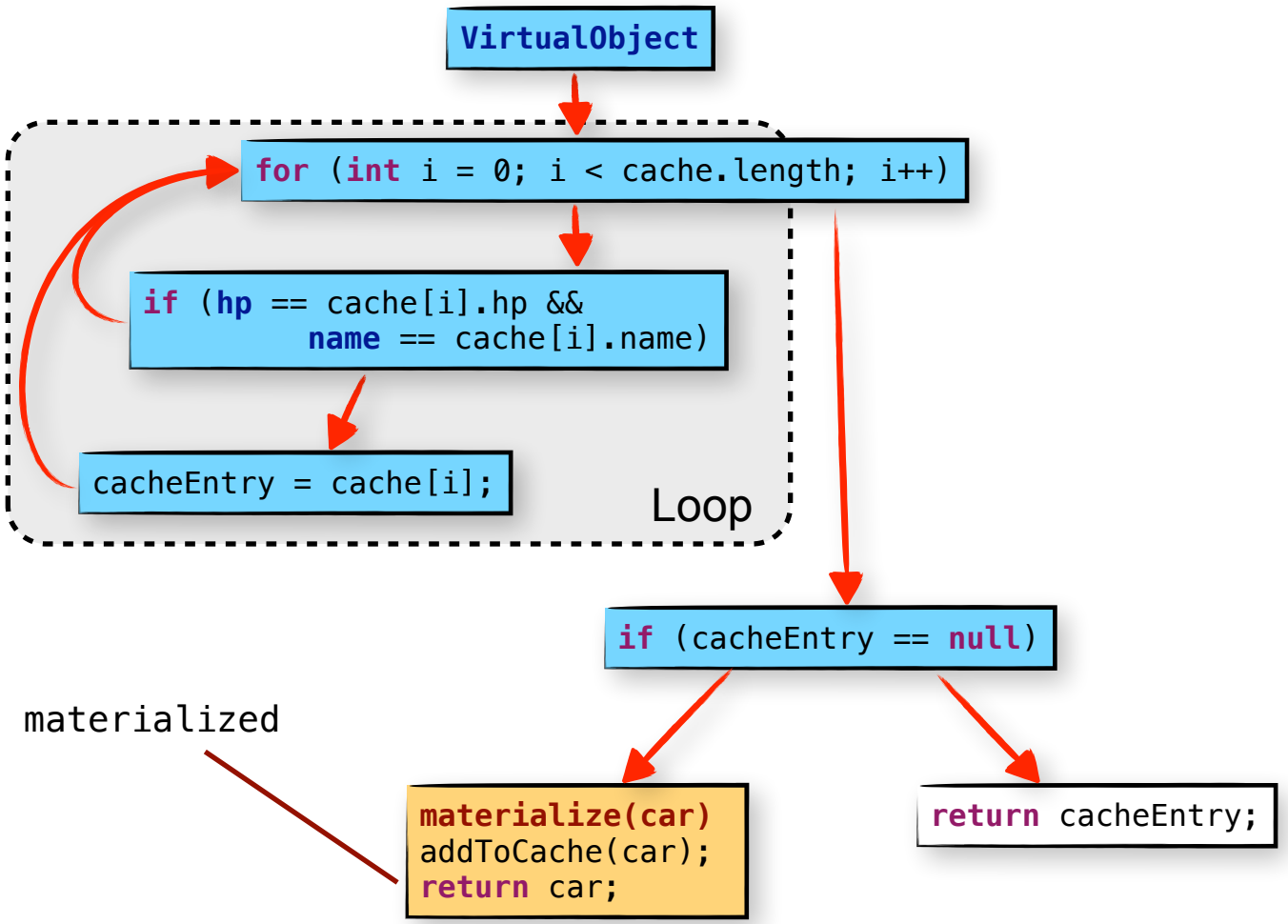
Partial Escape Analysis - Iteration



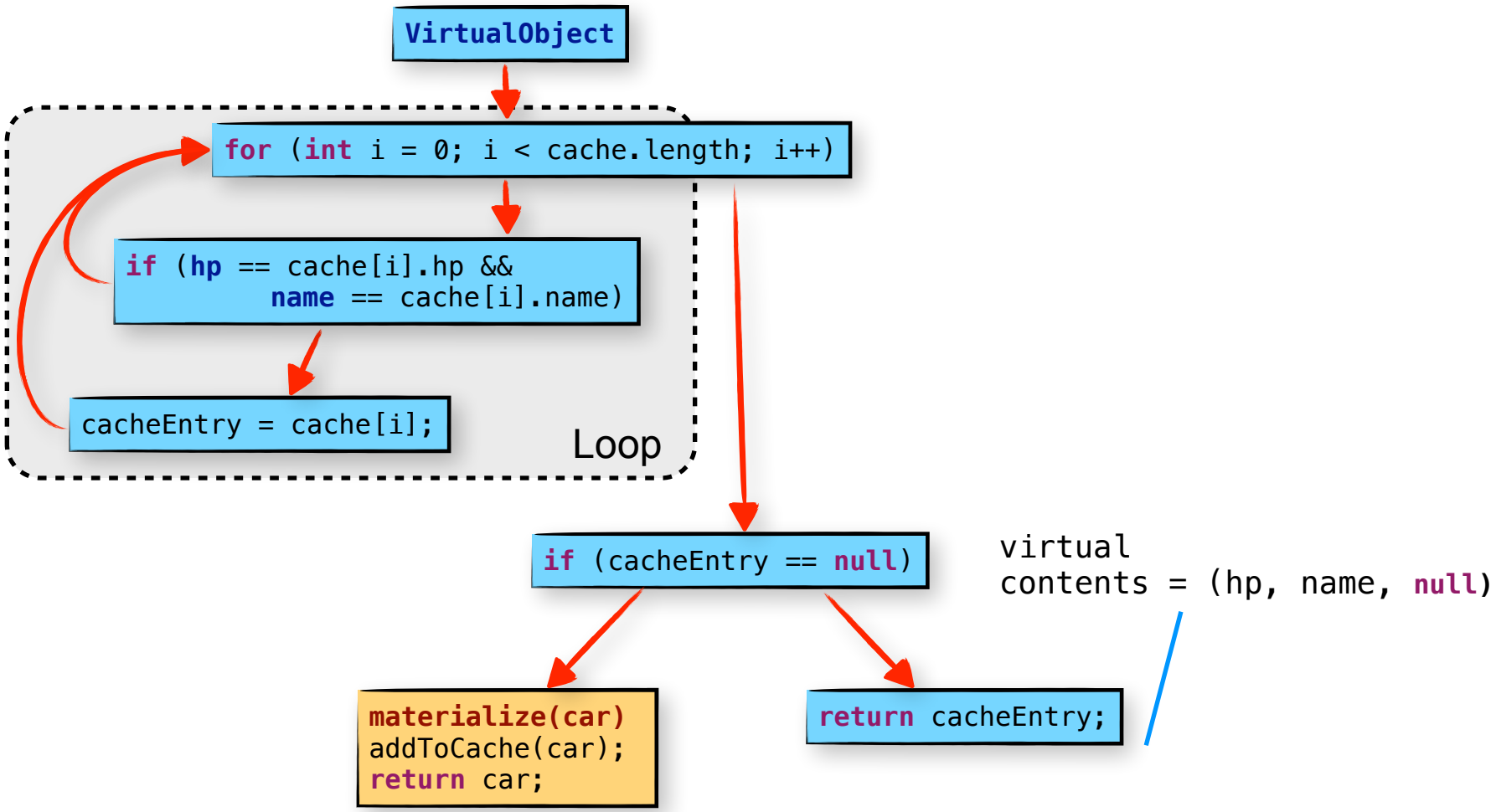
Partial Escape Analysis - Iteration



Partial Escape Analysis - Iteration



Partial Escape Analysis - Iteration

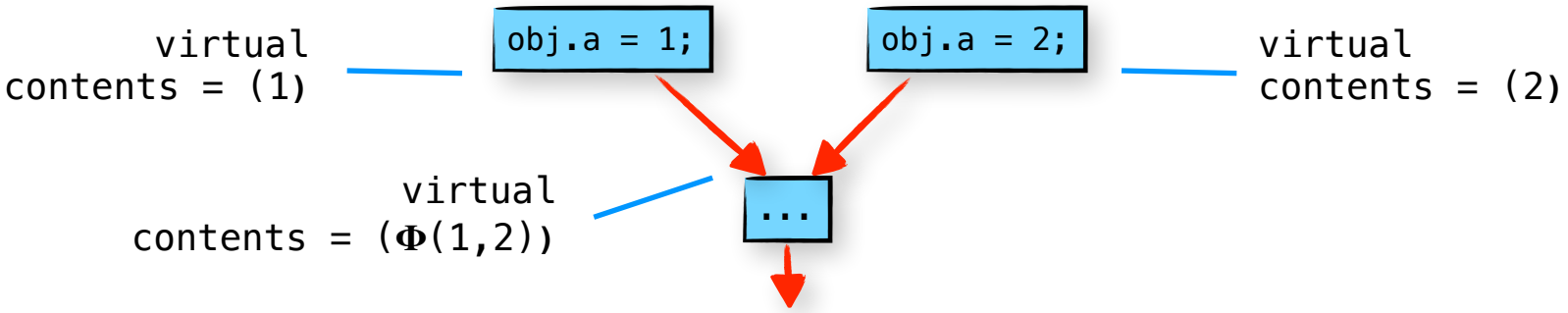


Partial Escape Analysis - Iteration

- Performs EA of all allocations in one iteration
  - Replaces IR nodes with virtualization effect
- Interfaces on IR node classes:
  - `interface VirtualizableAllocation`
    - Nodes that produce a virtualizable object (NewInstance, NewArray, ...)
  - `interface Virtualizable`
    - Nodes that have a virtualizable effect (StoreField, LoadIndexed, ...)

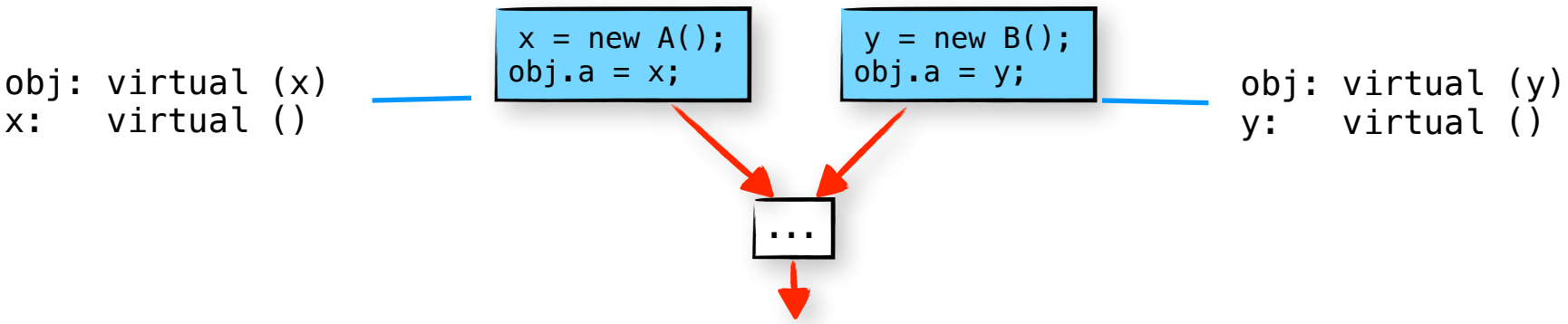
Partial Escape Analysis - Iteration

- Control Flow Merge
  - New Phi function



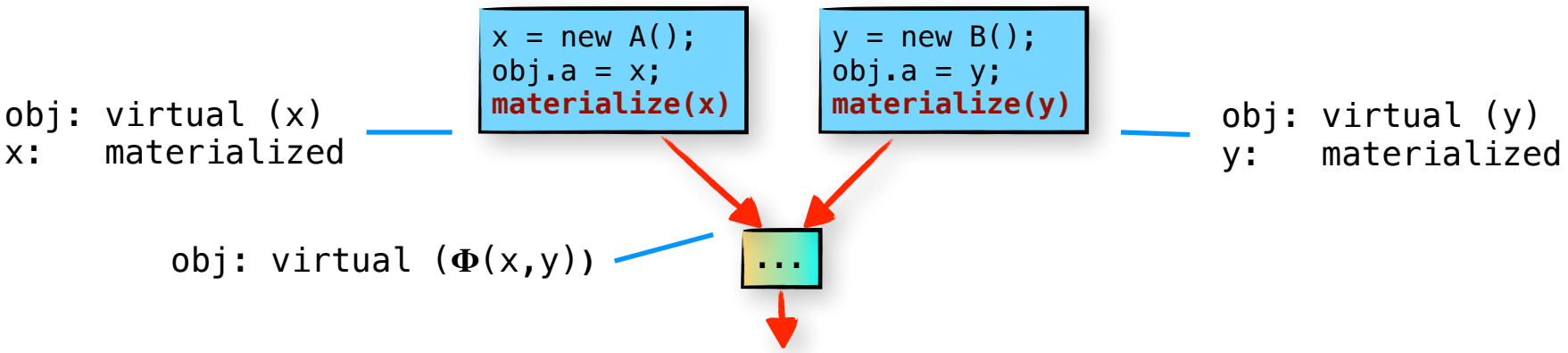
Partial Escape Analysis - Iteration

- Control Flow Merge
  - Merge of virtualized objects



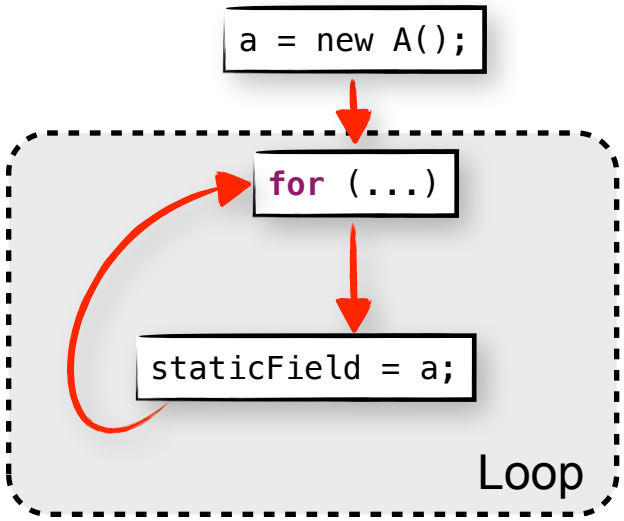
Partial Escape Analysis - Iteration

- Control Flow Merge
  - Merge of virtualized objects



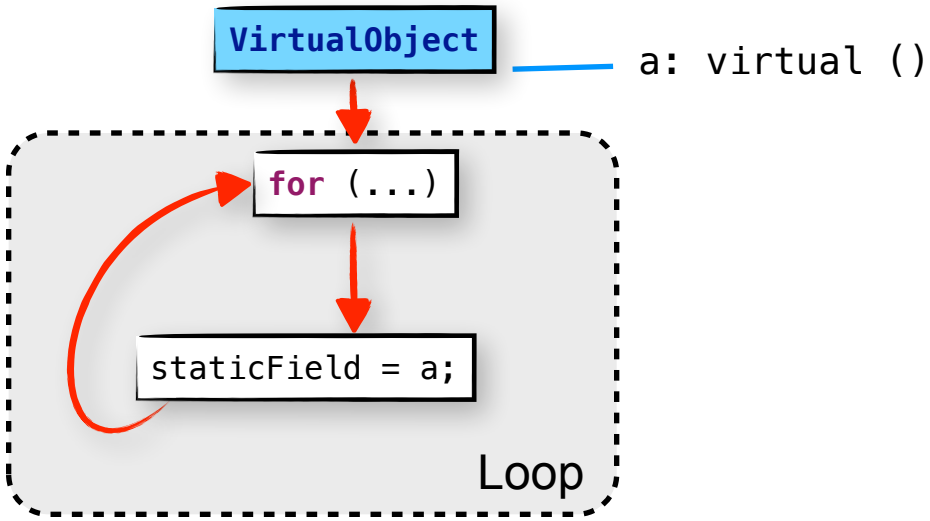
Partial Escape Analysis - Iteration

- Loops
  - Requires backtracking



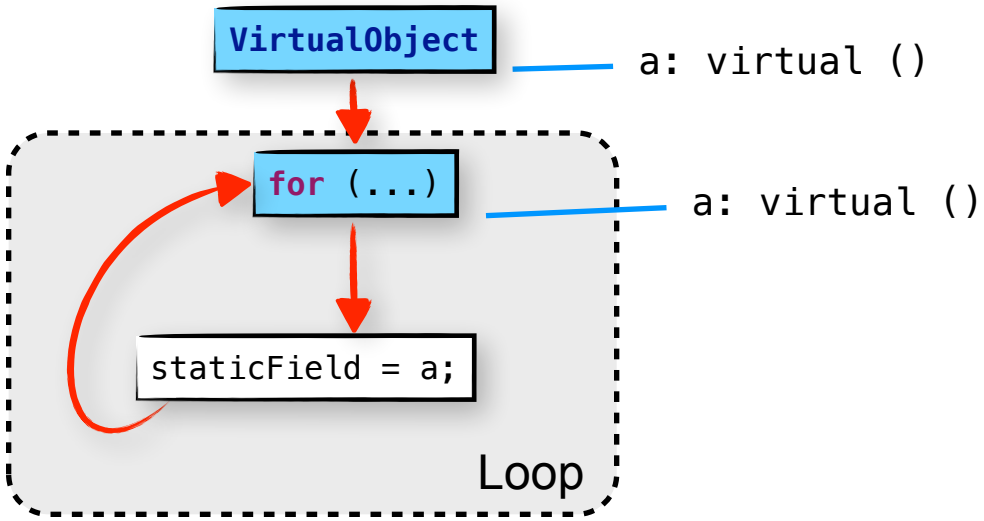
Partial Escape Analysis - Iteration

- Loops
  - Requires backtracking



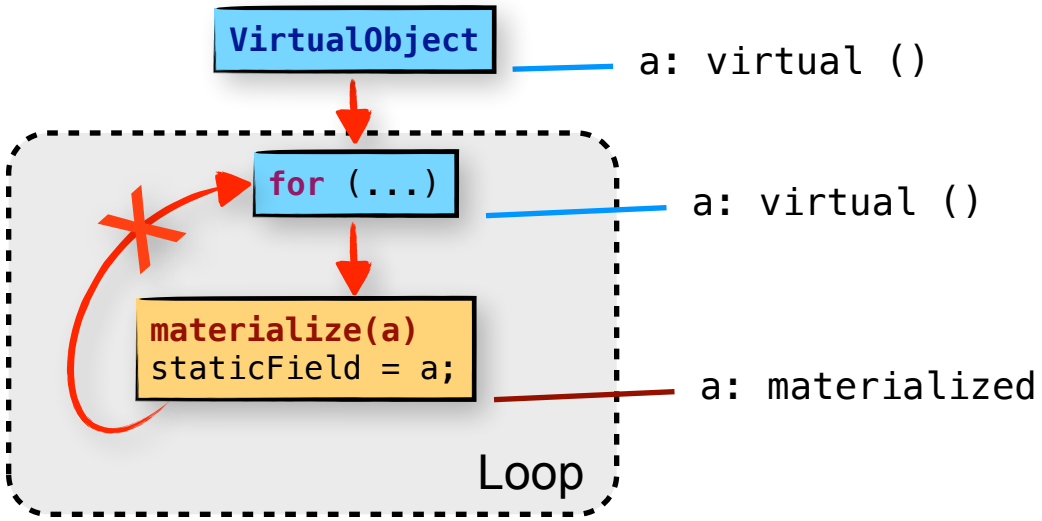
Partial Escape Analysis - Iteration

- Loops
  - Requires backtracking



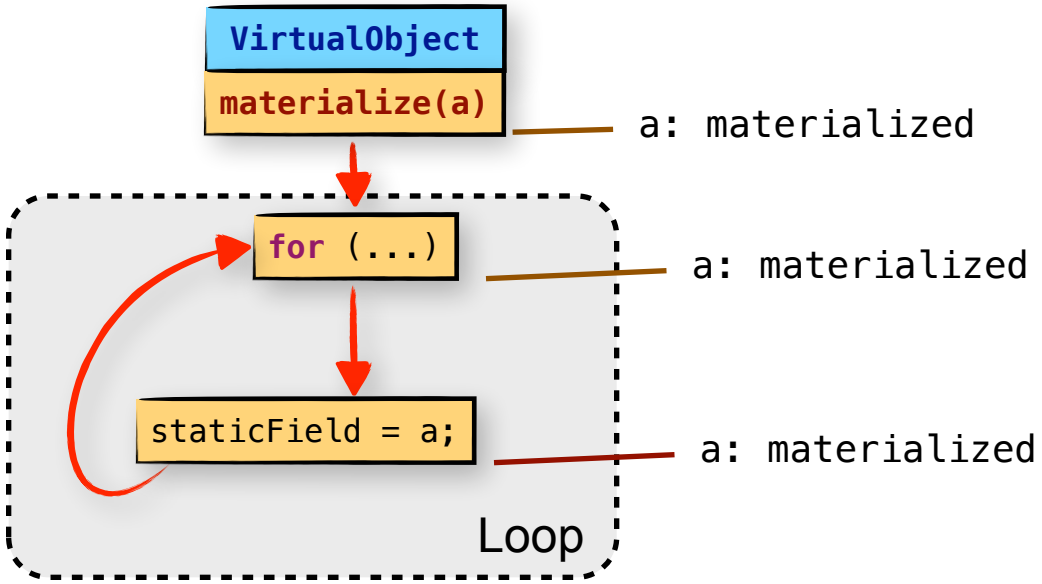
Partial Escape Analysis - Iteration

- Loops
  - Requires backtracking



Partial Escape Analysis - Iteration

- Loops
  - Requires backtracking



Evaluation

- Effects of Partial Escape Analysis:
  - Fewer allocations: less code
  - Fewer allocations: less GC work, less work for allocations
  - Fewer lock / unlock operations
  - Scalar Replacement: remove accesses
  - Coalescing allocations
  - Values not flowing through objects: easier for compiler
  - Clever handling of Boxing/Unboxing operations
- Impact on Compilation Time in Graal: 3.5 - 4%
  - Half of this is spent on scheduling
- Easy to implement PEA for new constructs:
  - Simply add `virtualize` method to new node type

Example - DaCapo sunflow

```
public Color getIrradiance(ShadingState state, Color diffRefl) {
  float b = (float) Math.PI * c / diffRefl.getMax();
  Color irr = Color.black();
  Point3 p = state.getPoint();
  Vector3 n = state.getNormal();
  int set = (int) (state.getRandom(0, 1, 1) * numSets);
  for (PointLight pl : virtualLights[set]) {
    Ray r = new Ray(p, pl.p);
    float dotNlD = -(r.dx * pl.n.x + r.dy * pl.n.y + r.dz * pl.n.z);
    float dotND = r.dx * n.x + r.dy * n.y + r.dz * n.z;
    if (dotNlD > 0 && dotND > 0) {
      float r2 = r.getMax() * r.getMax();
      Color opacity = state.traceShadow(r);
      Color power = Color.blend(pl.power, Color.BLACK, opacity);
      float g = (dotND * dotNlD) / r2;
      irr.madd(0.25f * Math.min(g, b), power);
    }
  }
  return irr;
}
```

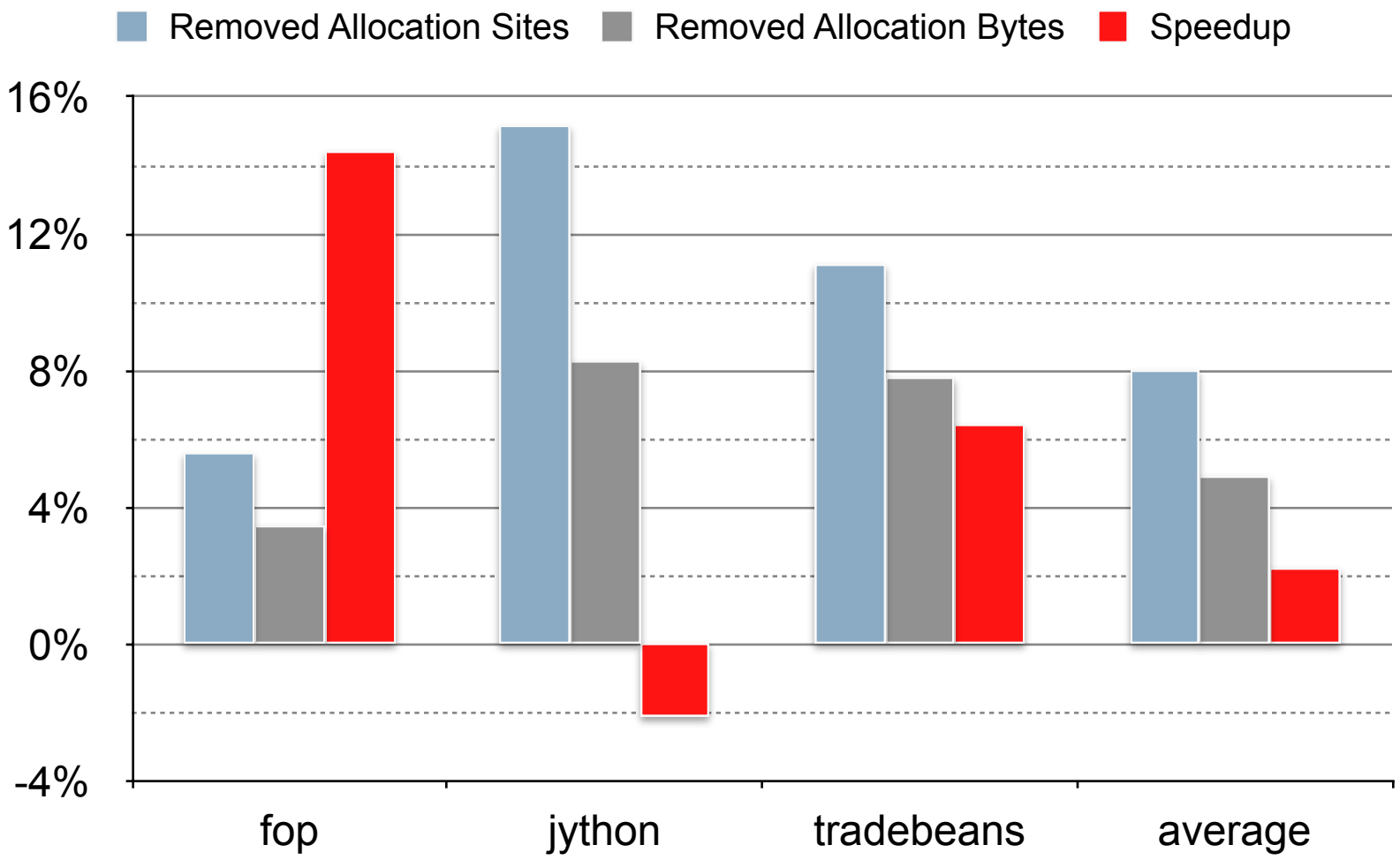
count: ~130

probability: 56%

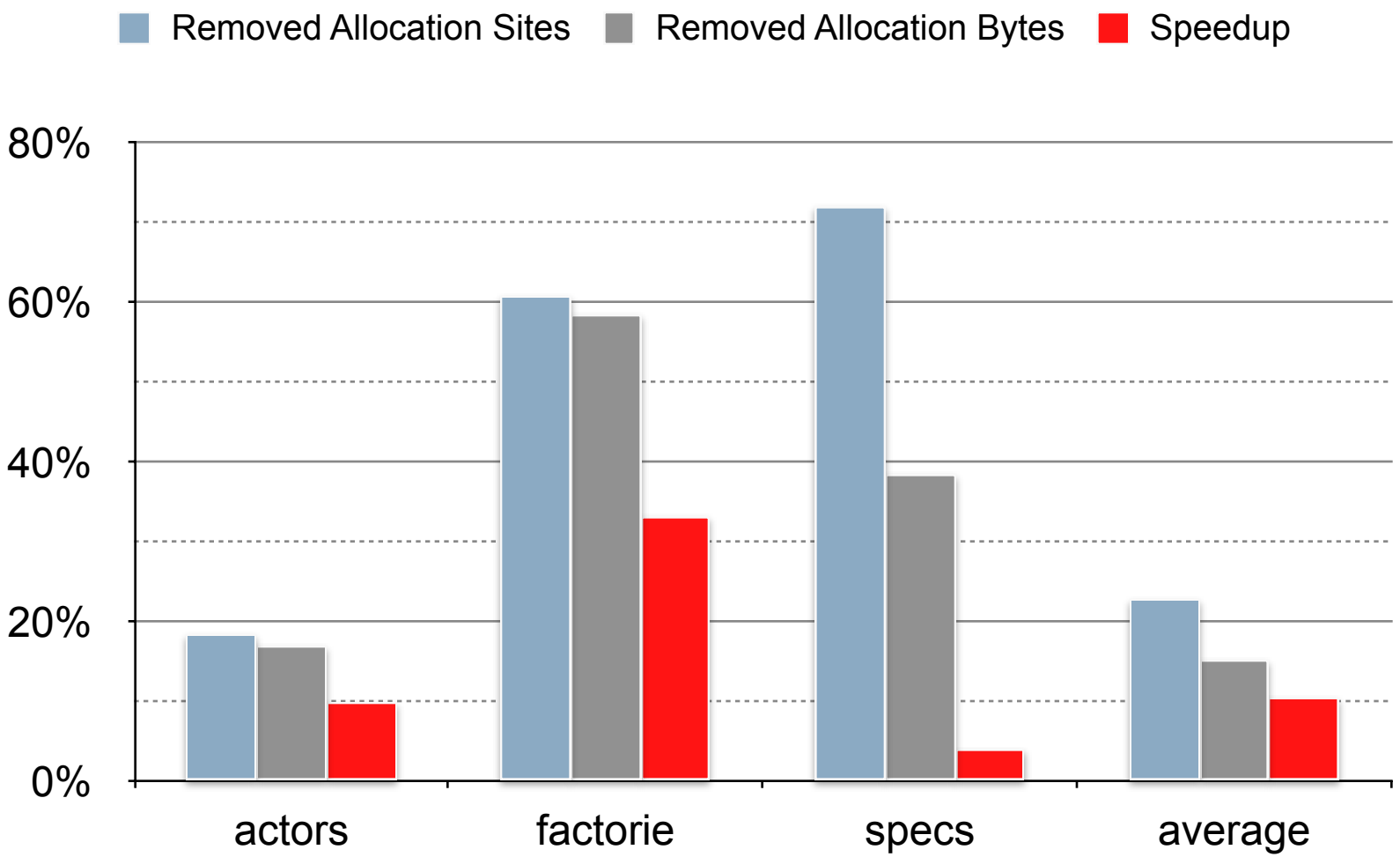
Partial Escape Analysis:
… removed 33% of allocation sites
… removed 64% of dynamic allocations (EA: 36%)
… reduced size of method by 18%

Evaluation - DaCapo

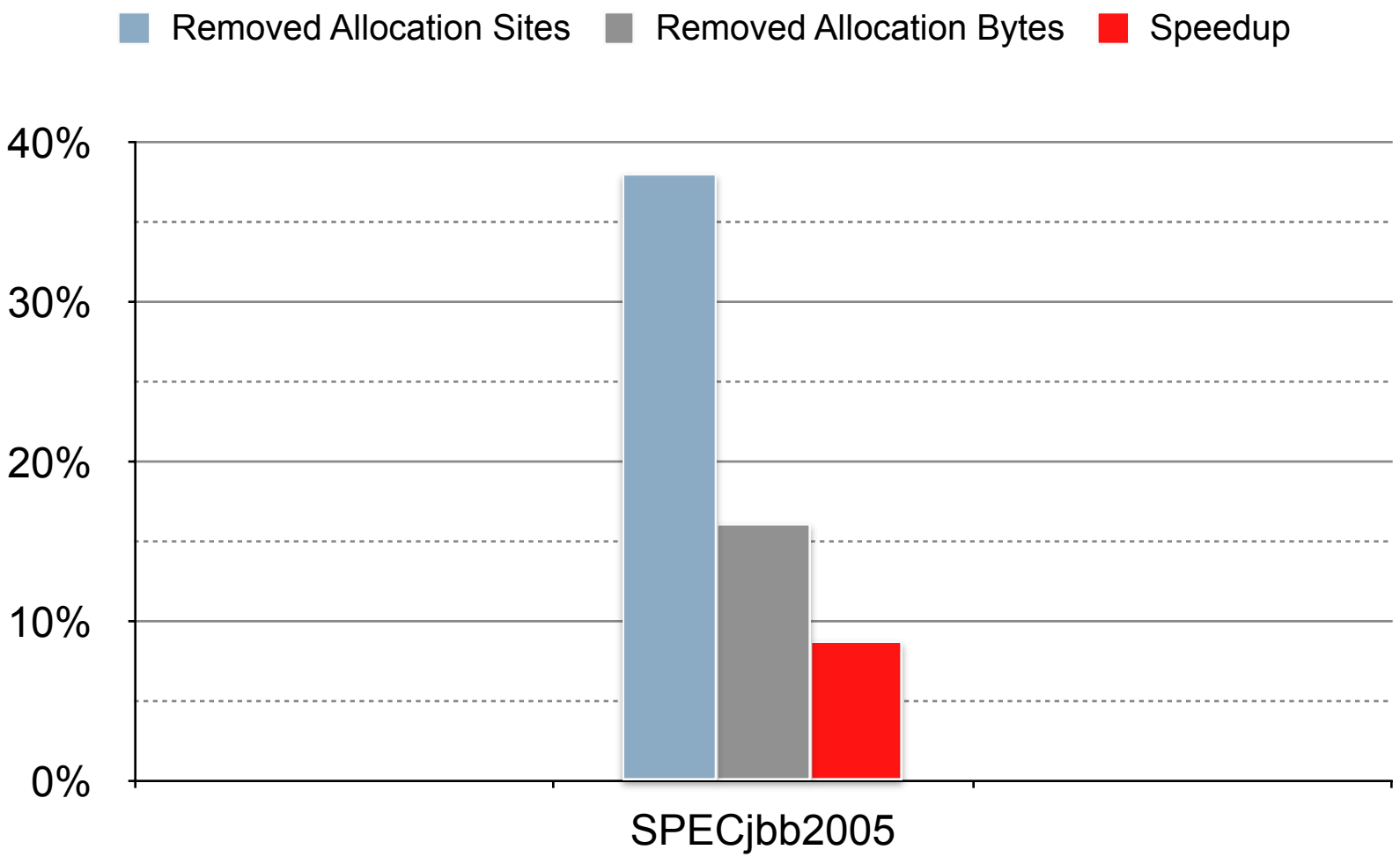
Xeon E5-2690, 2GB heap



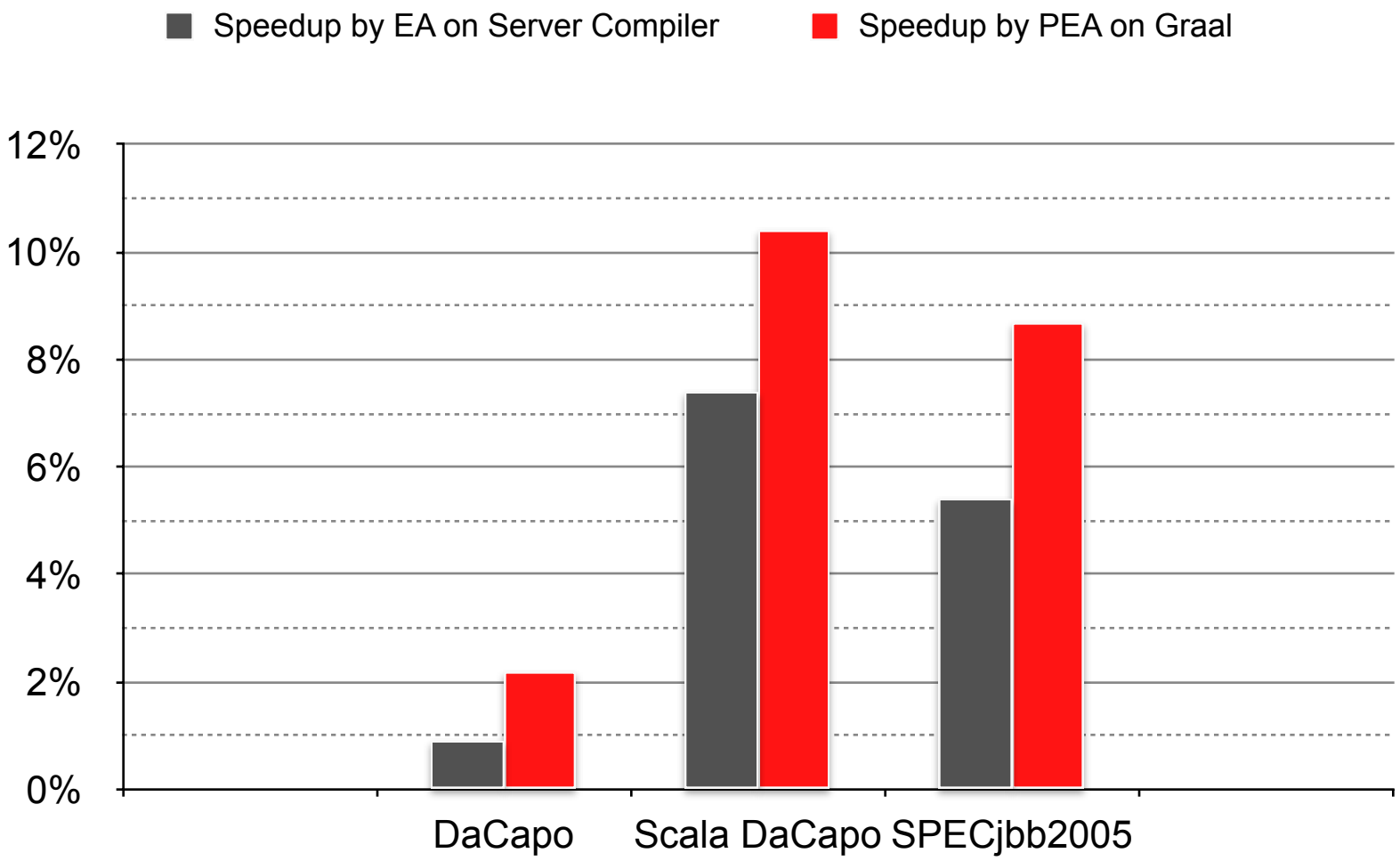
Evaluation - Scala DaCapo



Evaluation - SPECjbb2005



Evaluation - Comparison to Server Compiler



Conclusions

- Scheduling is costly
  - Non-scheduling version
  - Special scheduling that only places some nodes
- Efficient way to perform Escape Analysis
- Very important for Truffle framework
  - Can be applied multiple times during compilation

Acknowledgements

The Graal Team:
Gilles Duboscq (JKU)
Christian Häubl (JKU)
Christos Kotselidis (Oracle Labs)
Prof. Hanspeter Mössenböck (JKU)
Roland Schatz (Oracle Labs)
Doug Simon (Oracle Labs)
Lukas Stadler (Oracle Labs)
Bernhard Urban (JKU)
Christian Wimmer (Oracle Labs)
Thomas Würthinger (Oracle Labs)

Q&A

Hardware and Software

ORACLE

Engineered to Work Together


ORACLE®