# Are Your Incoming Aliases Really Necessary? Counting the Cost of Object Ownership

Alex Potanin, Monique Damitio, James Noble
Victoria University of Wellington, New Zealand

# Are Your Incoming Aliases Really Necessary? Remembering the Cost of Object Ownership

Alex Potanin, Monique Damitio, James Noble
Victoria University of Wellington, New Zealand

# Aliasing

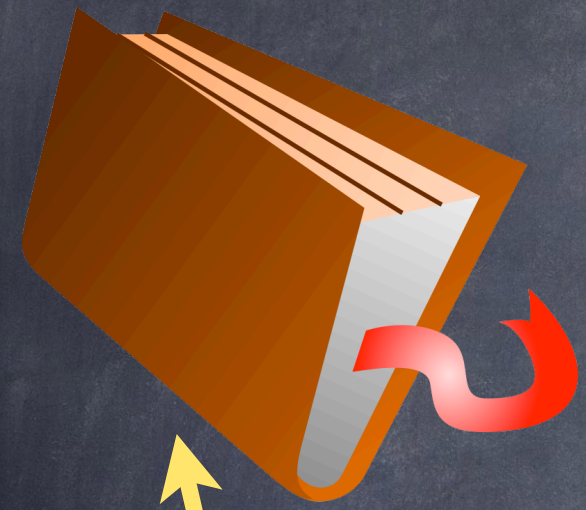
"The big lie of object-oriented programming is that objects provide encapsulation."

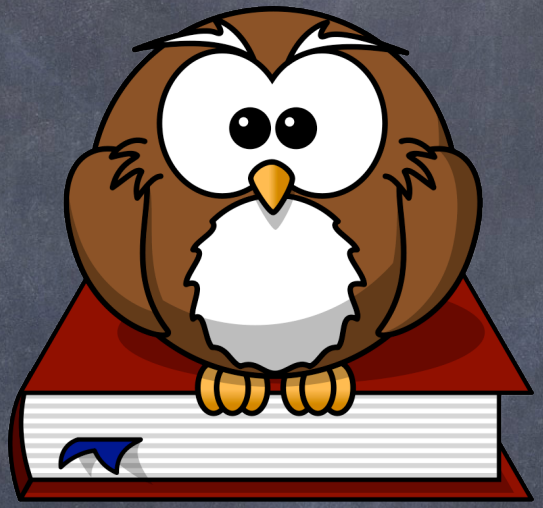
John Hogg

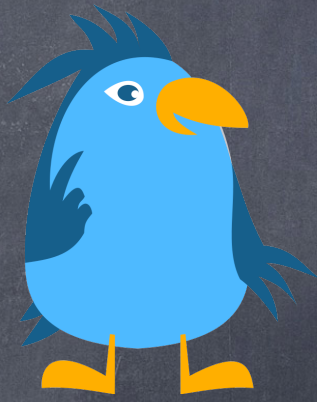
Islands: Aliasing Protection in Object-Oriented Languages

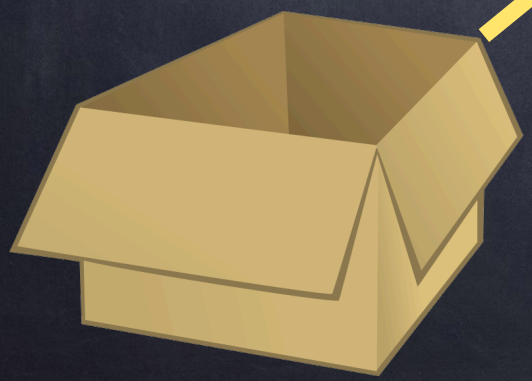
OOPSLA 1991

# Object-Oriented Programming



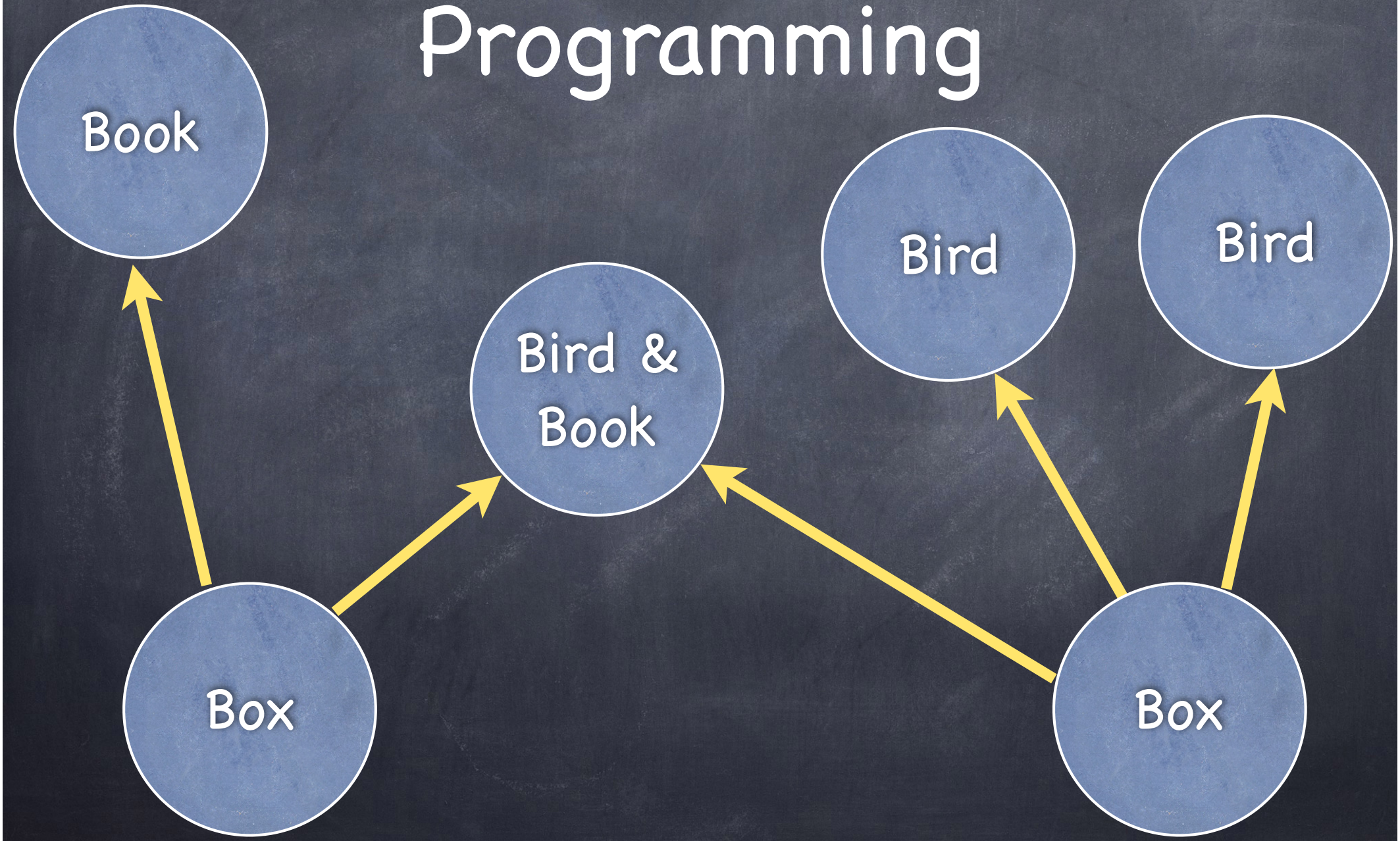




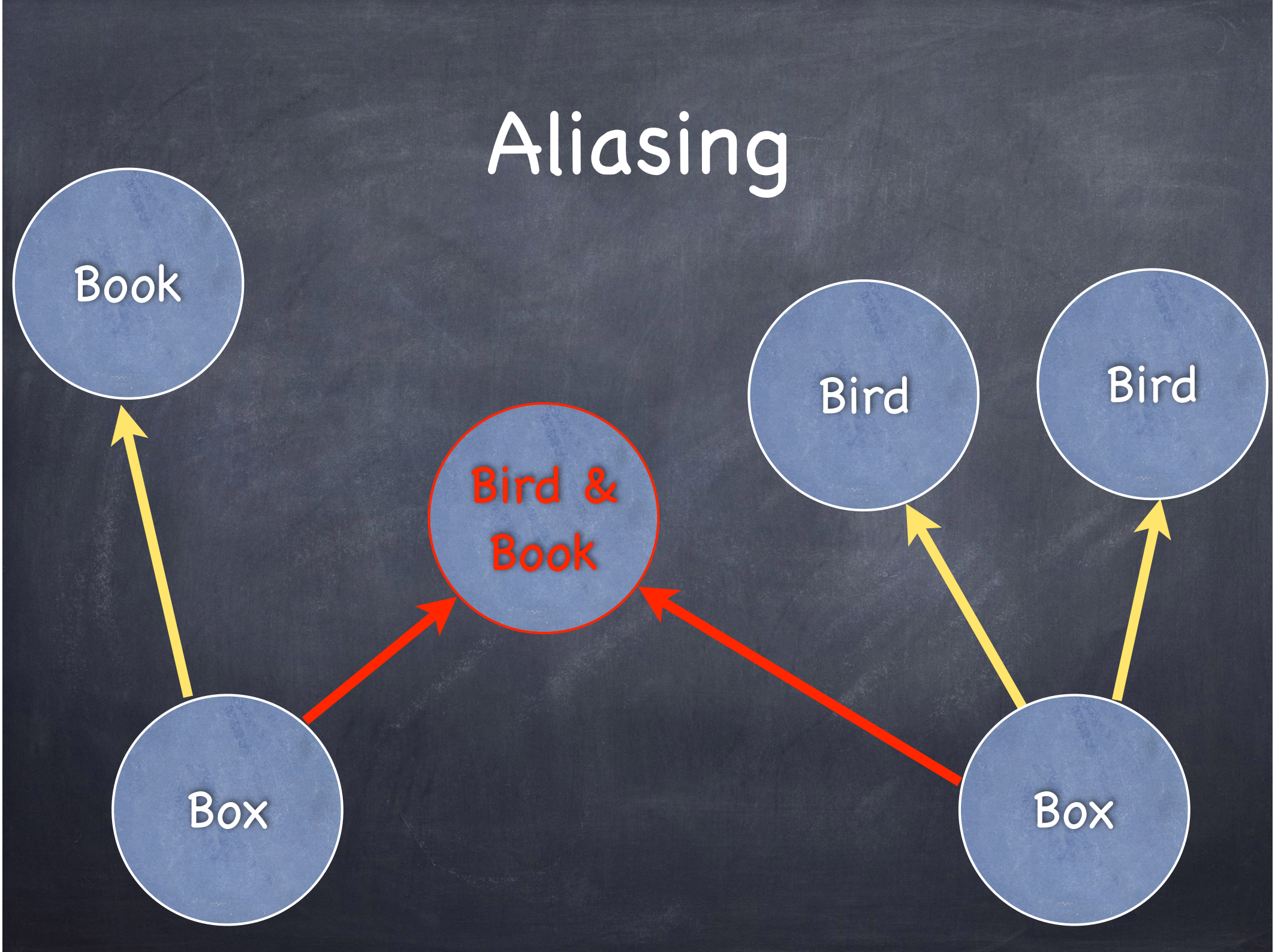


http://openclipart.org/

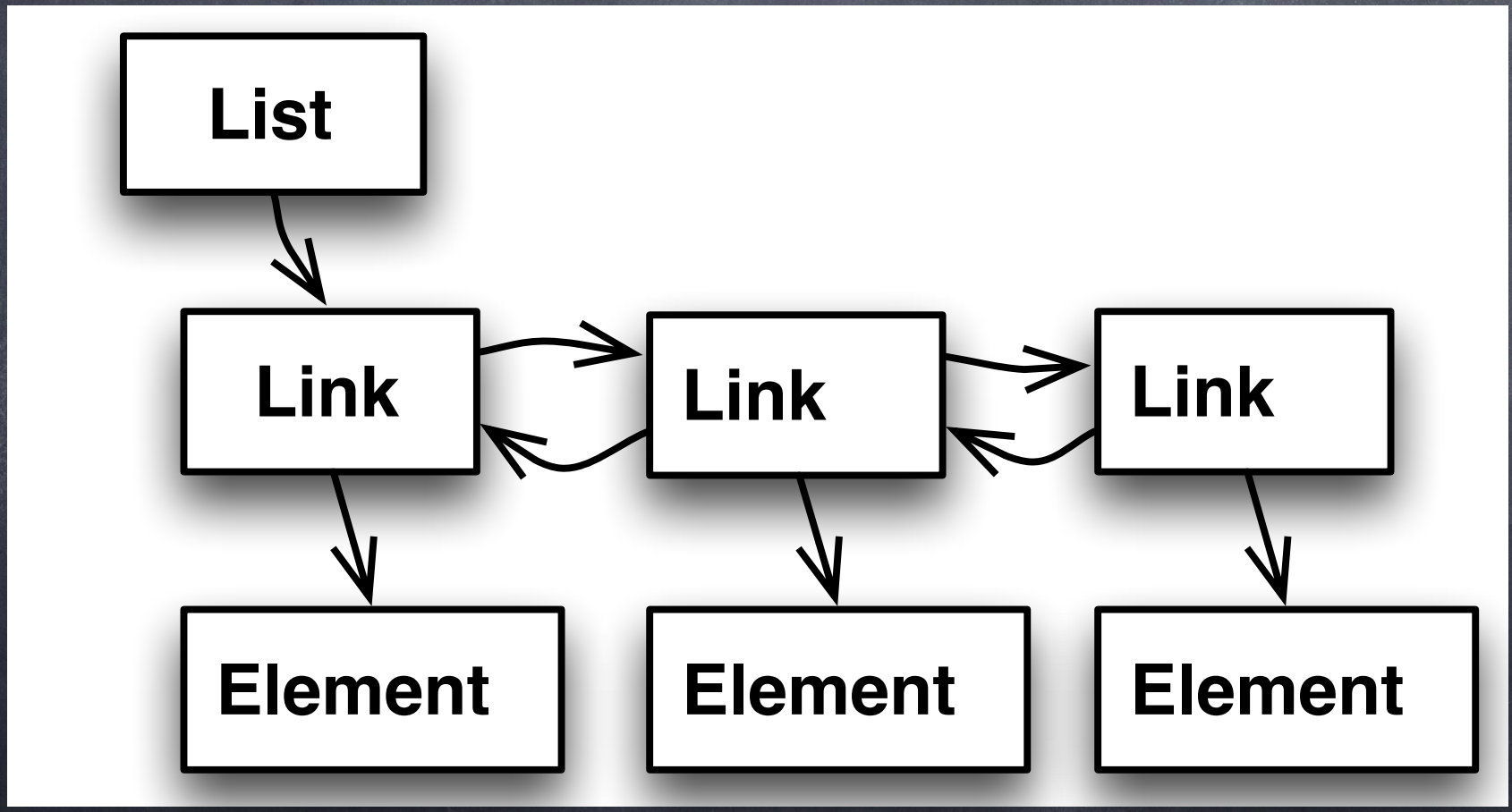
# Object-Oriented Programming



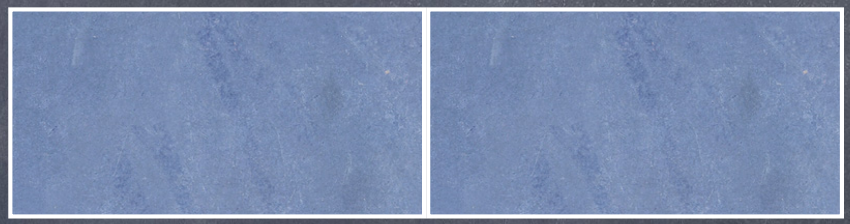
# Aliasing



# Aliasing (The Good)



# Aliasing (The Bad)

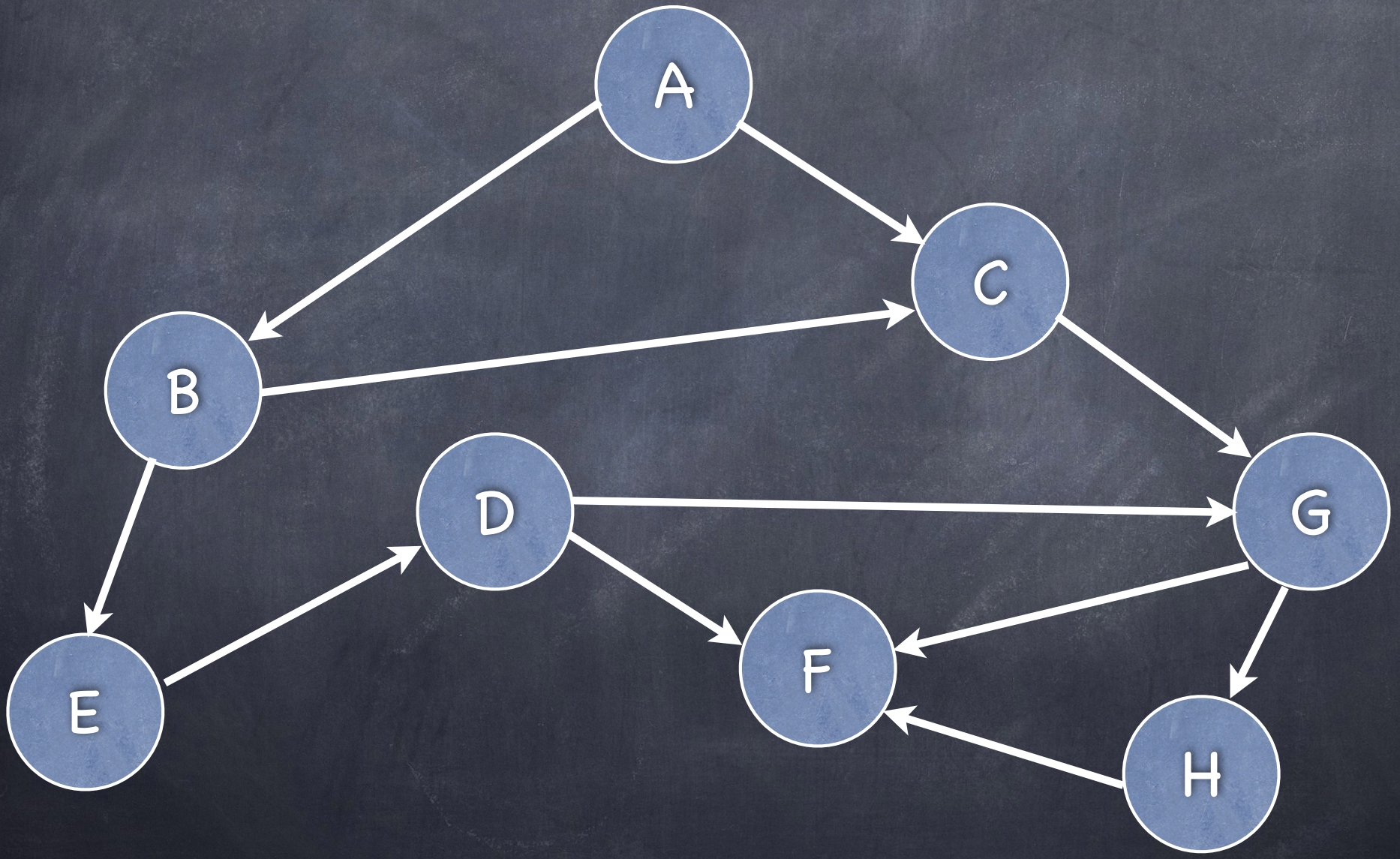


```
class Rectangle {
    private Point topLeft;
    private int w, h;
    public Rectangle (Point topLeft, int w, int h)
                { this.topLeft = topLeft; this.w = w; this.h = h; }
}
...
Point p = new Point(100, 50);
Rectangle r = new Rectangle(p, 300, 200);
...
p.setX(400);
```

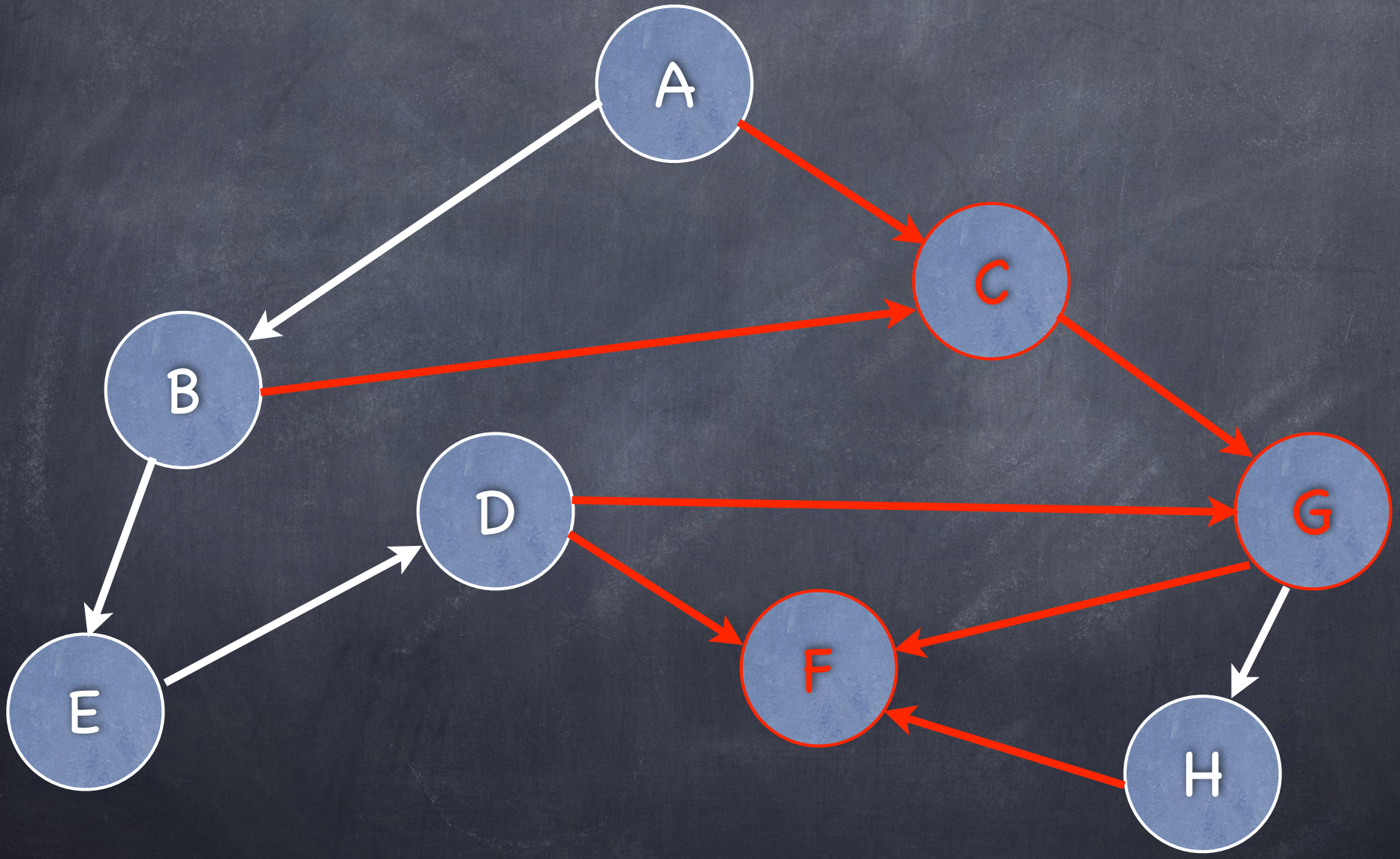
# Aliasing (The Ugly)

- Bugs due to unintentional aliasing are hard to track down (e.g. applets breaking out of JDK v1.1.1 sandbox)
- Typically enforced by coding style only (e.g. CMU SEI CERT OBJ05-J advises to "defensively copy private mutable class members before returning their references")
- Still no mainstream language support for enforcing per-object encapsulation after 20+ years... (Rust?)

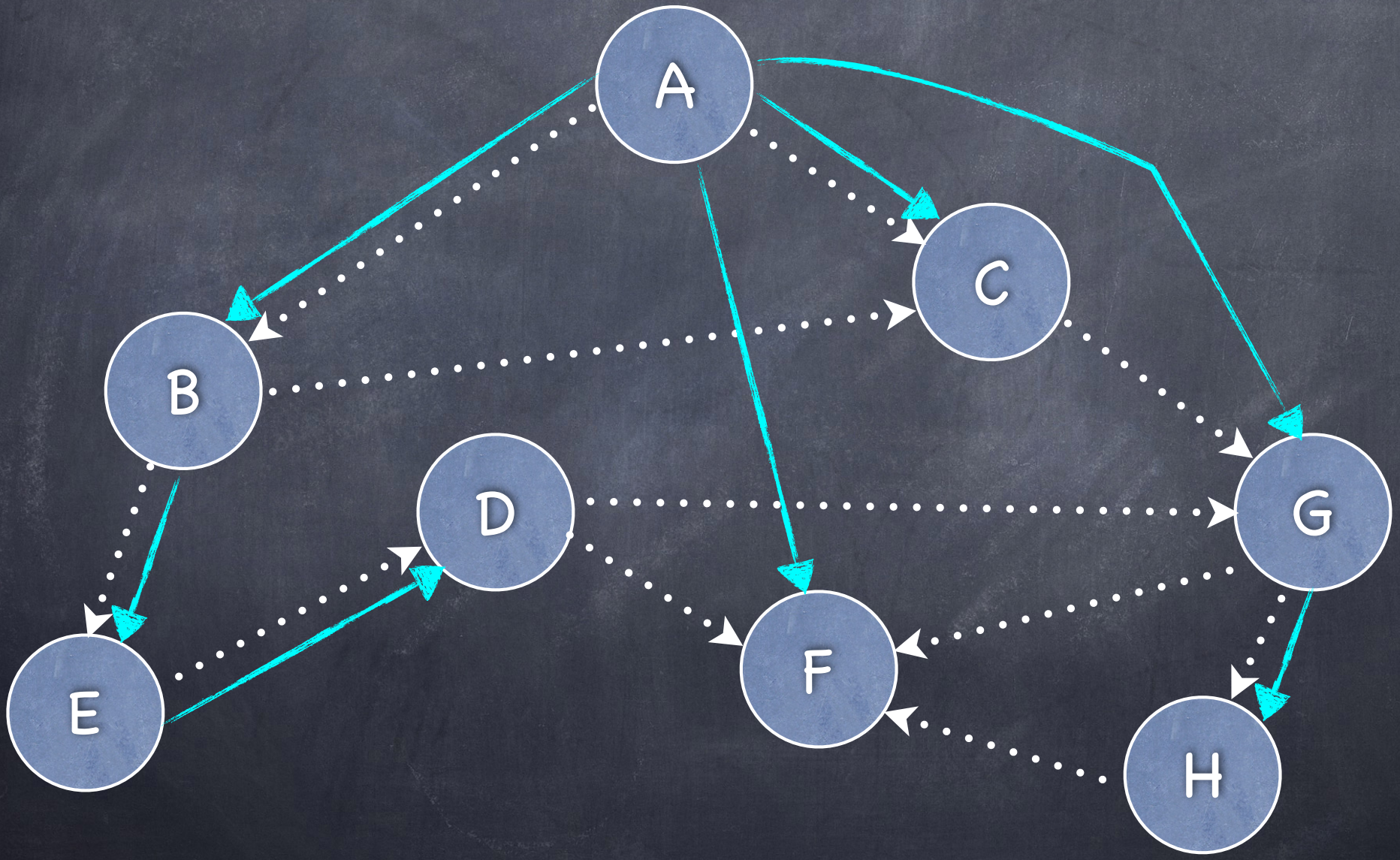
# Object Graph



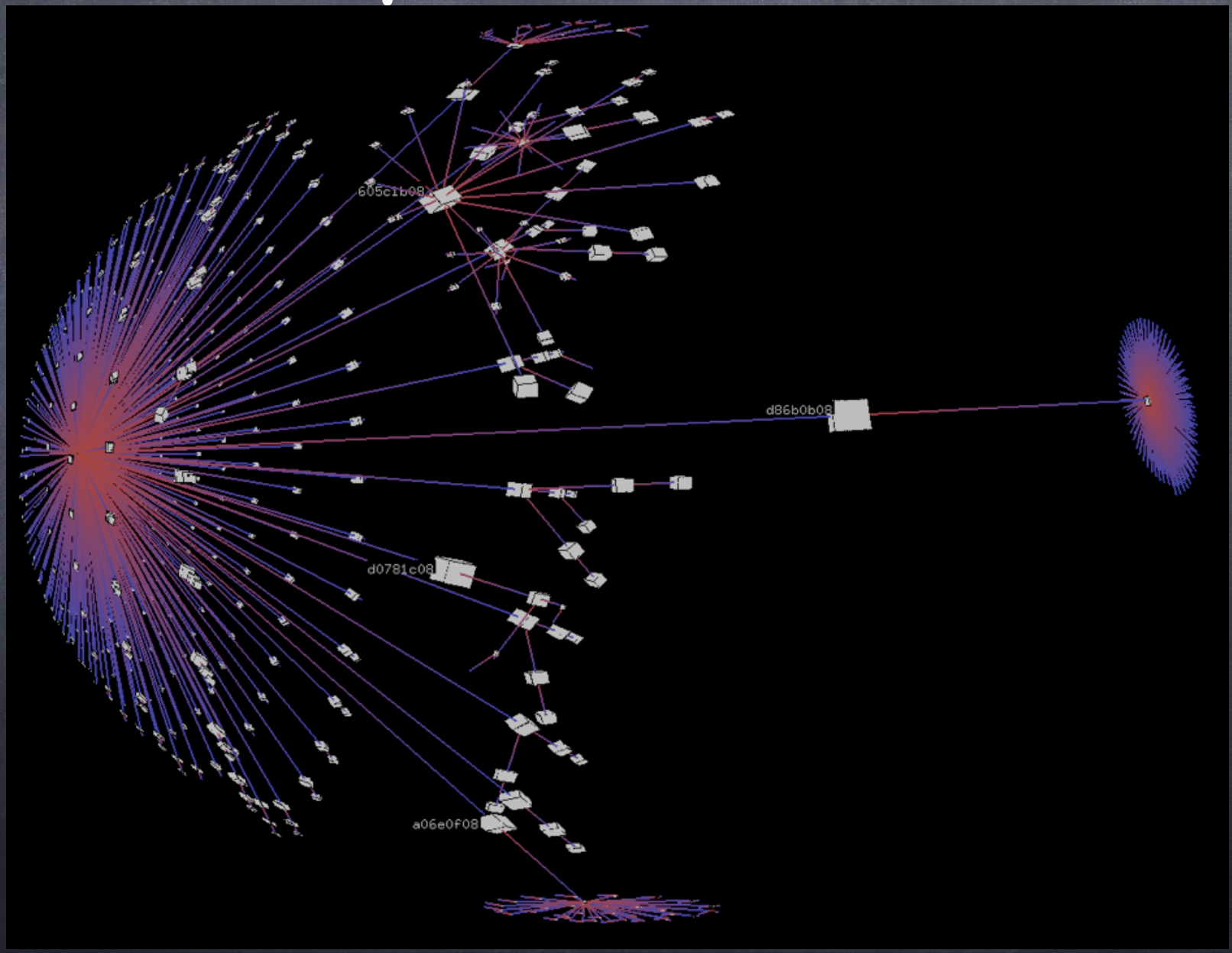
# Object Graph (Aliasing)



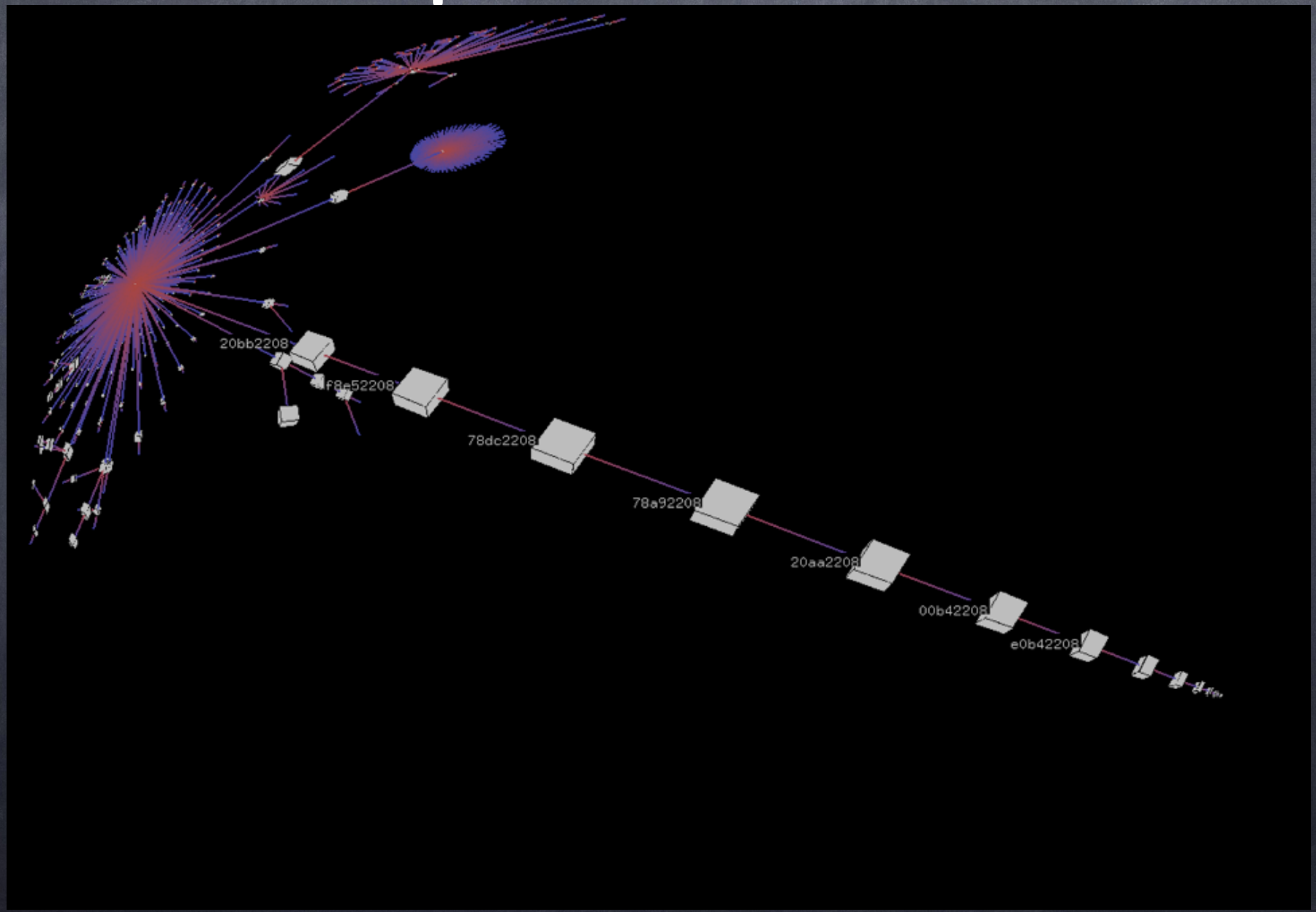
# Ownership Tree



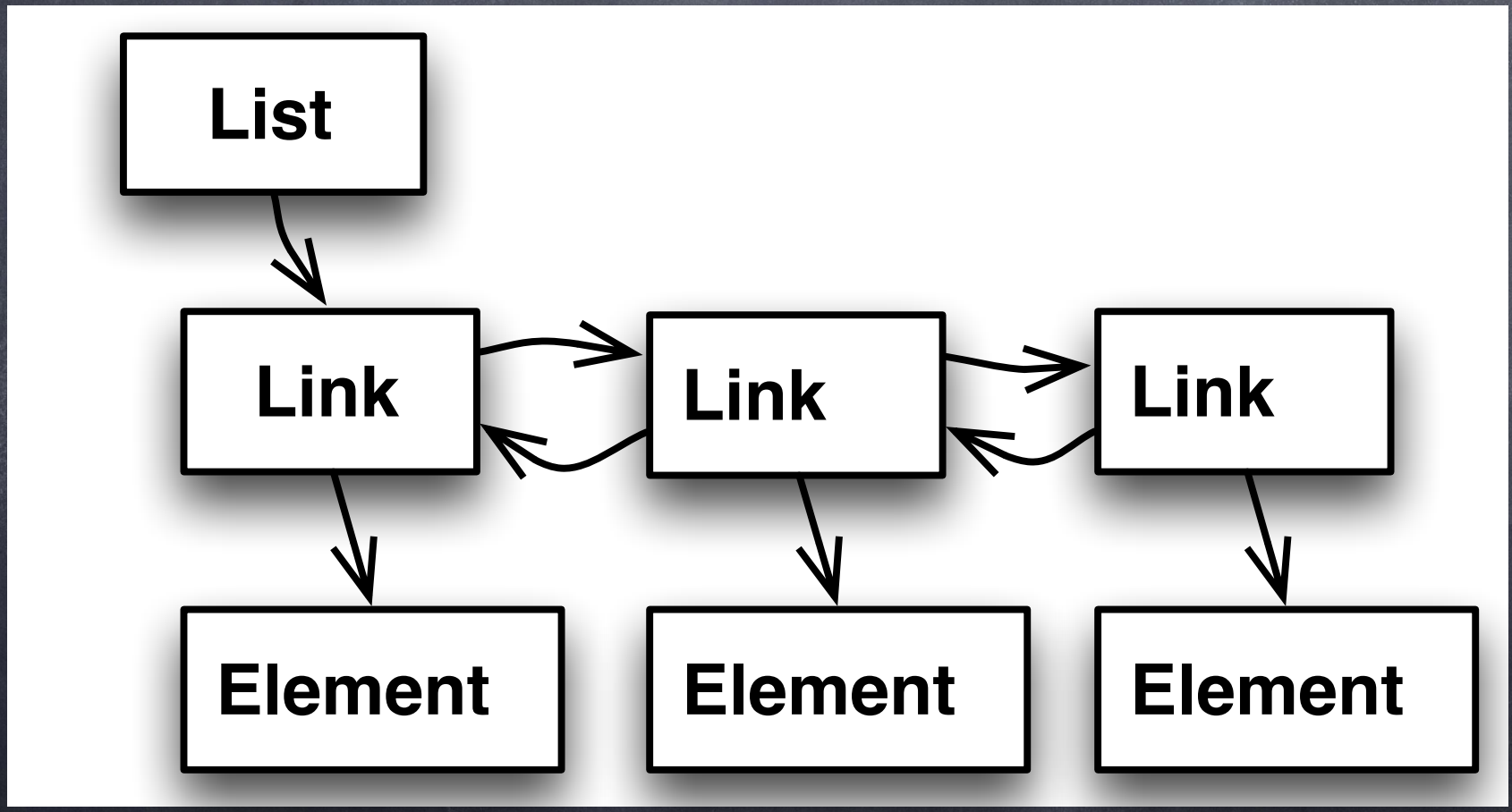
# Ownership Tree: HelloWorld



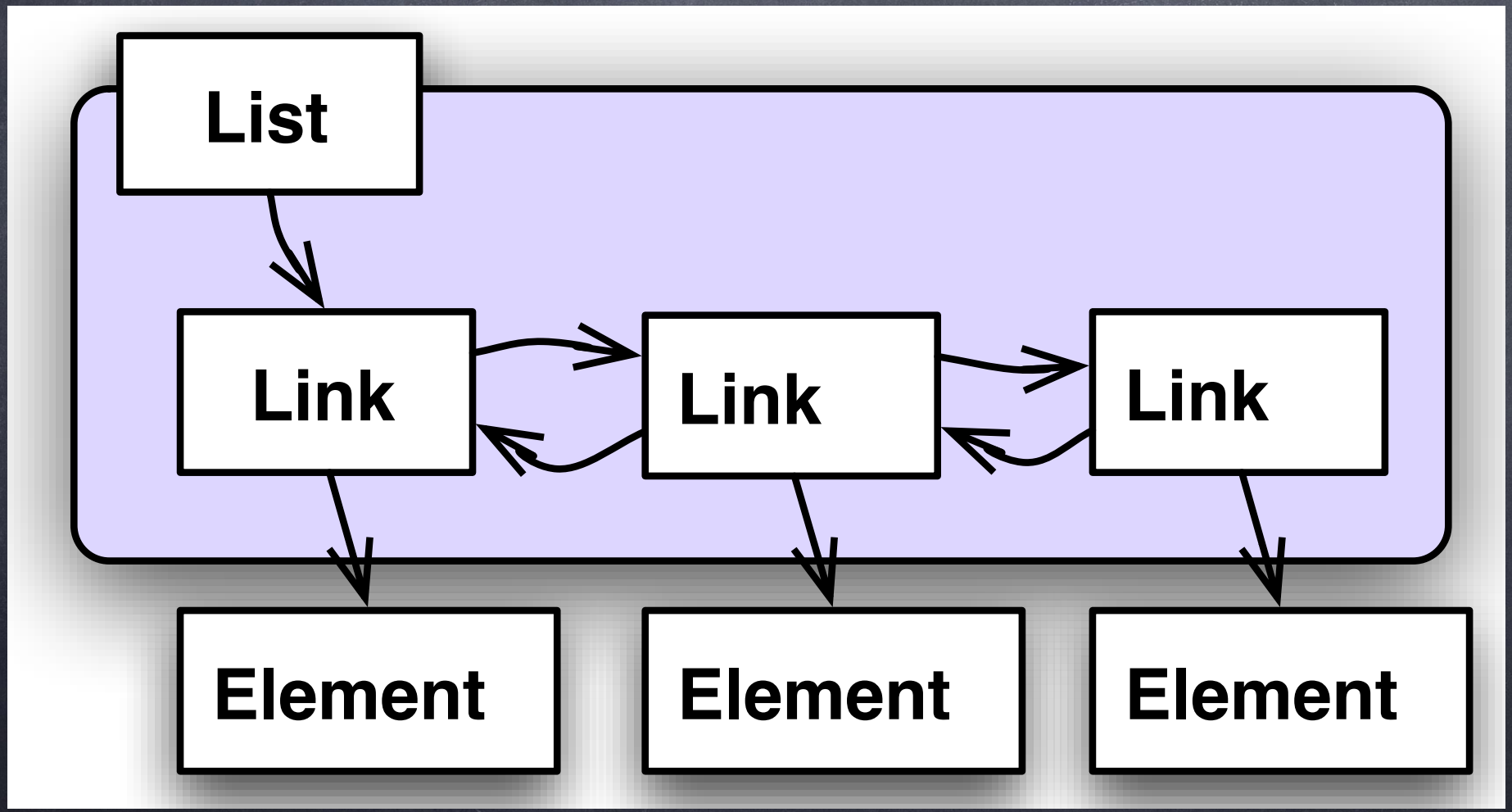
# Ownership Tree: LinkedList



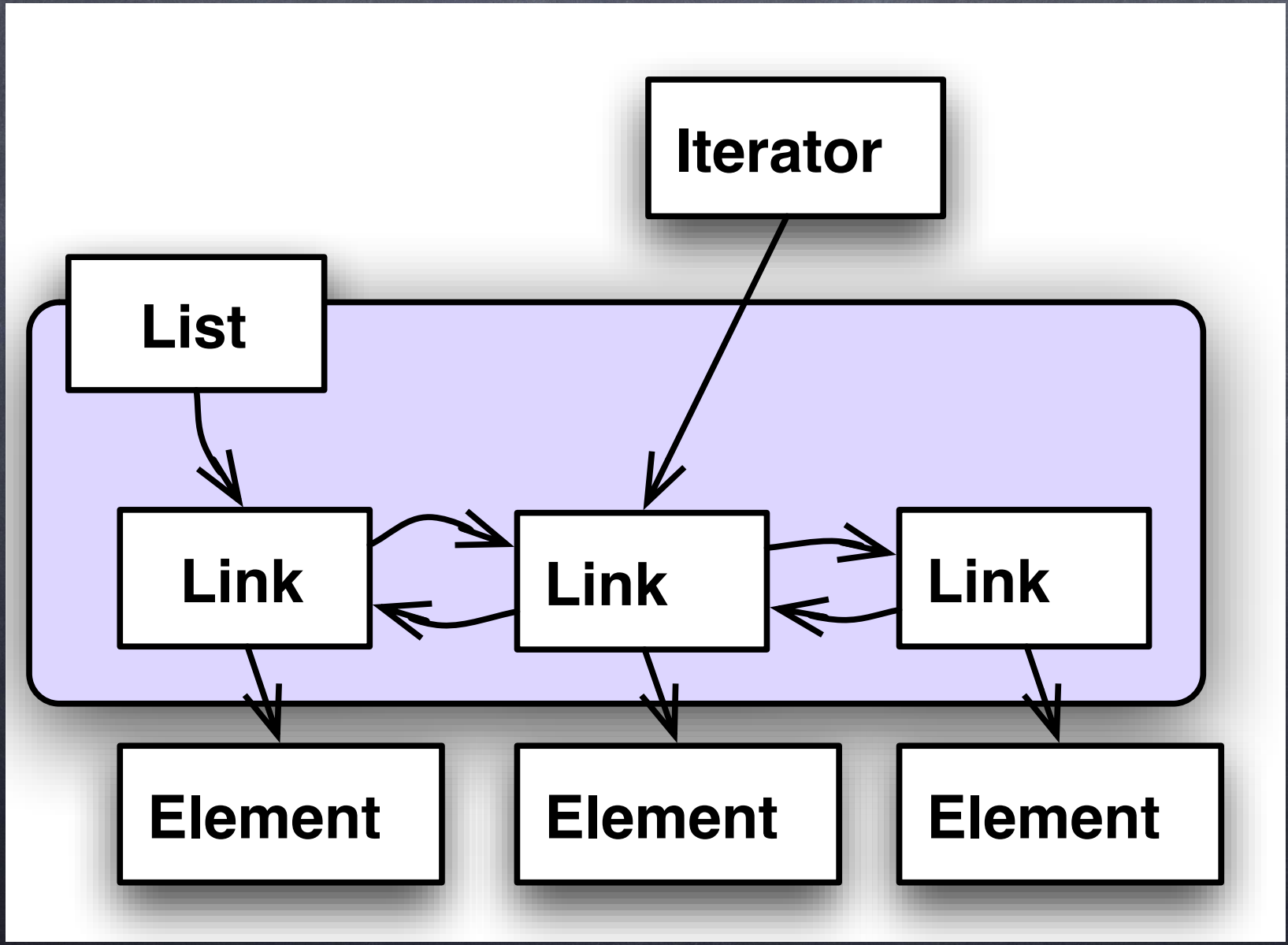
# Back to LinkedList



# Encapsulated LinkedList



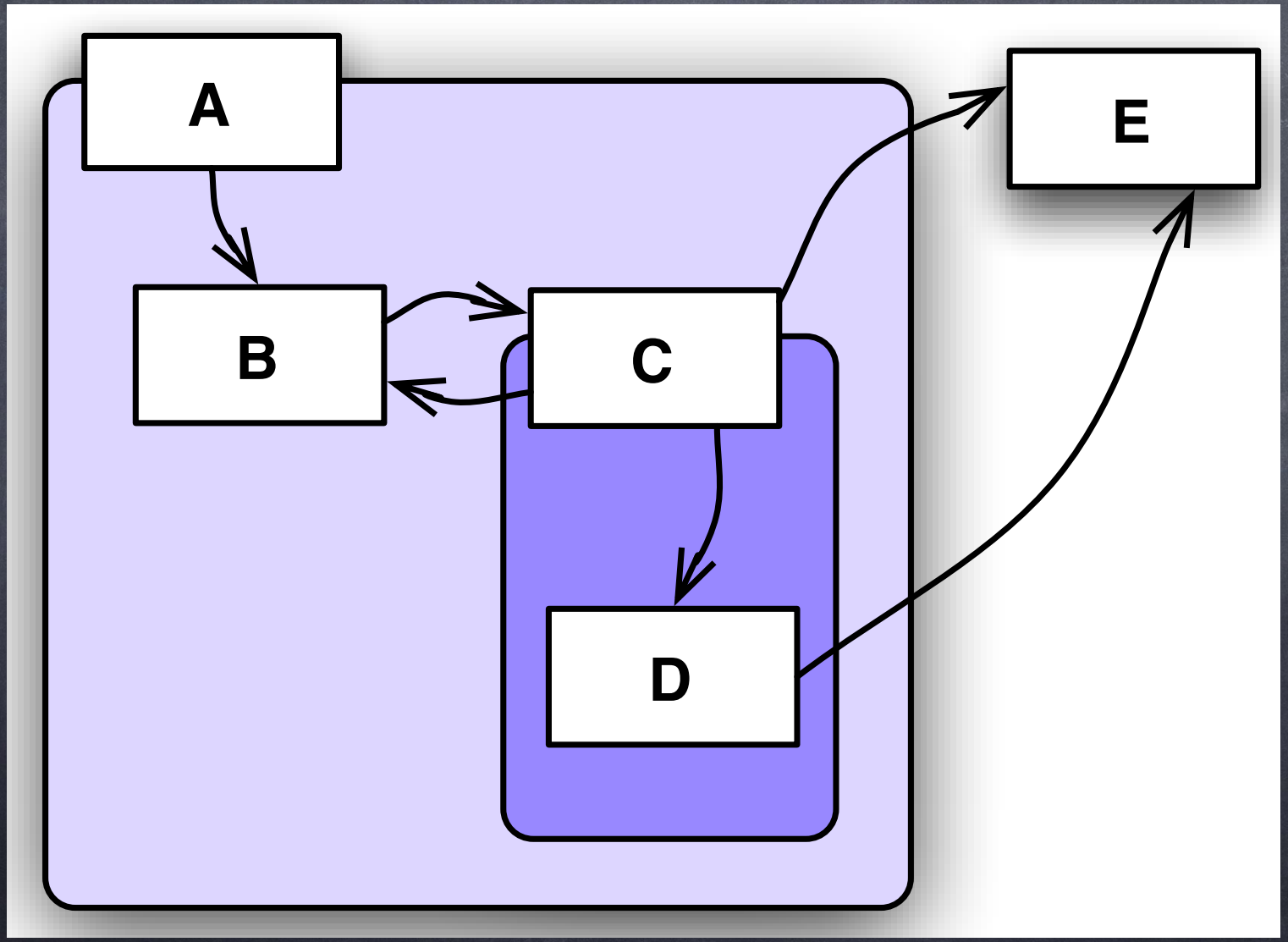
# LinkedList with Iterator



# Our Questions

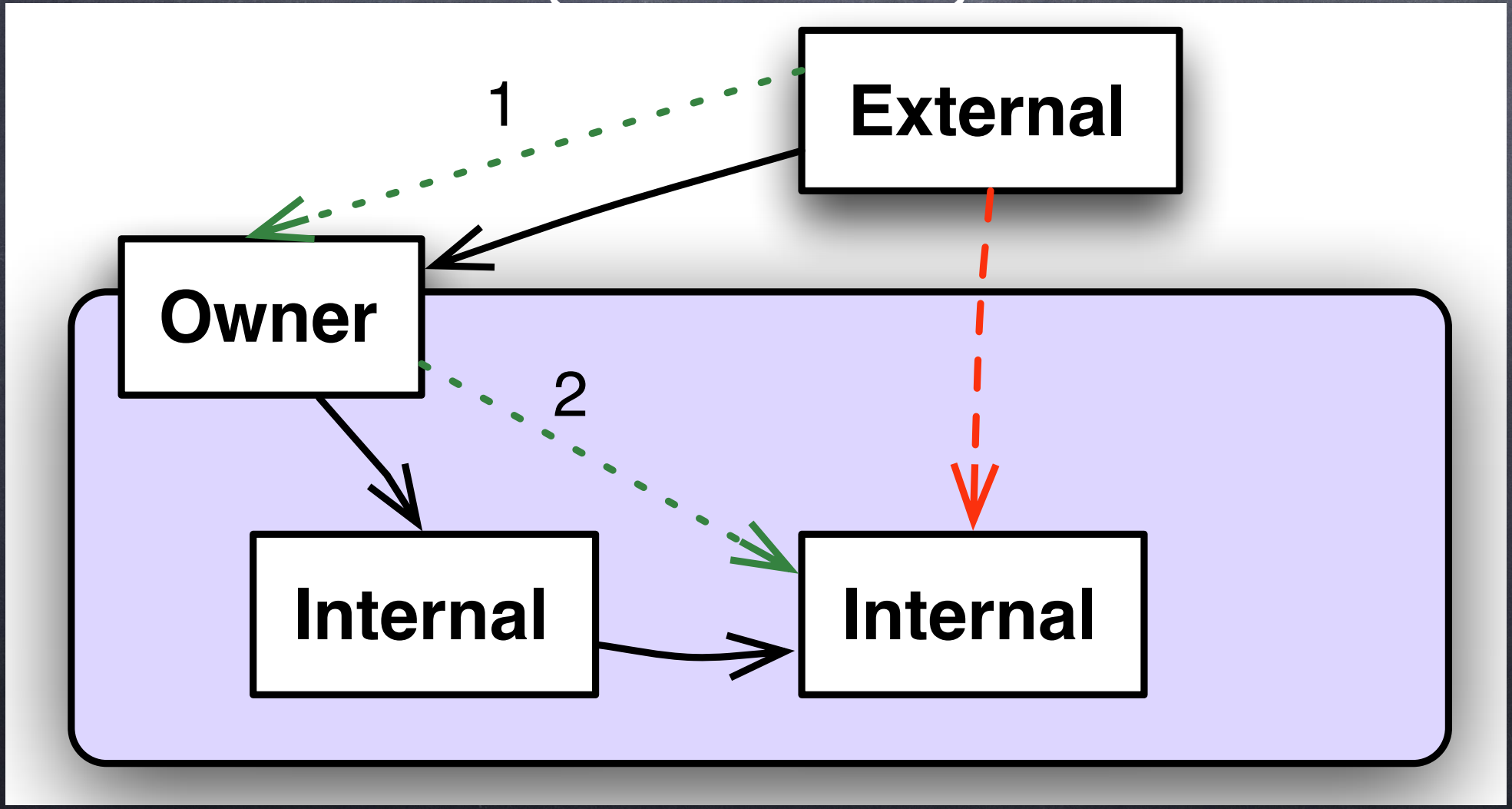
- How must designs change to respect encapsulation?
- What performance cost do these changes impose?
- How does this impact programs' performance?

# More on Ownership



# Owners as Accessors

(IWACO 2014)



# Owner-as-Accessor Example

```
class Internal { String s = “abc”; }
class Owner {
  Internal i = new Internal();
  void modifyI() { i.s = “def”; } }
class External {
  Internal i; Owner o = new Owner();
  void doIt() {
    this.i = o.i;  // OK. Stores external ref.
    o.modifyI(); // OK. Modifies Internal.
    // this.i.s = “ghi”; // WRONG. Not owner!
}  }
```

# Java Collections Framework v1.5.0

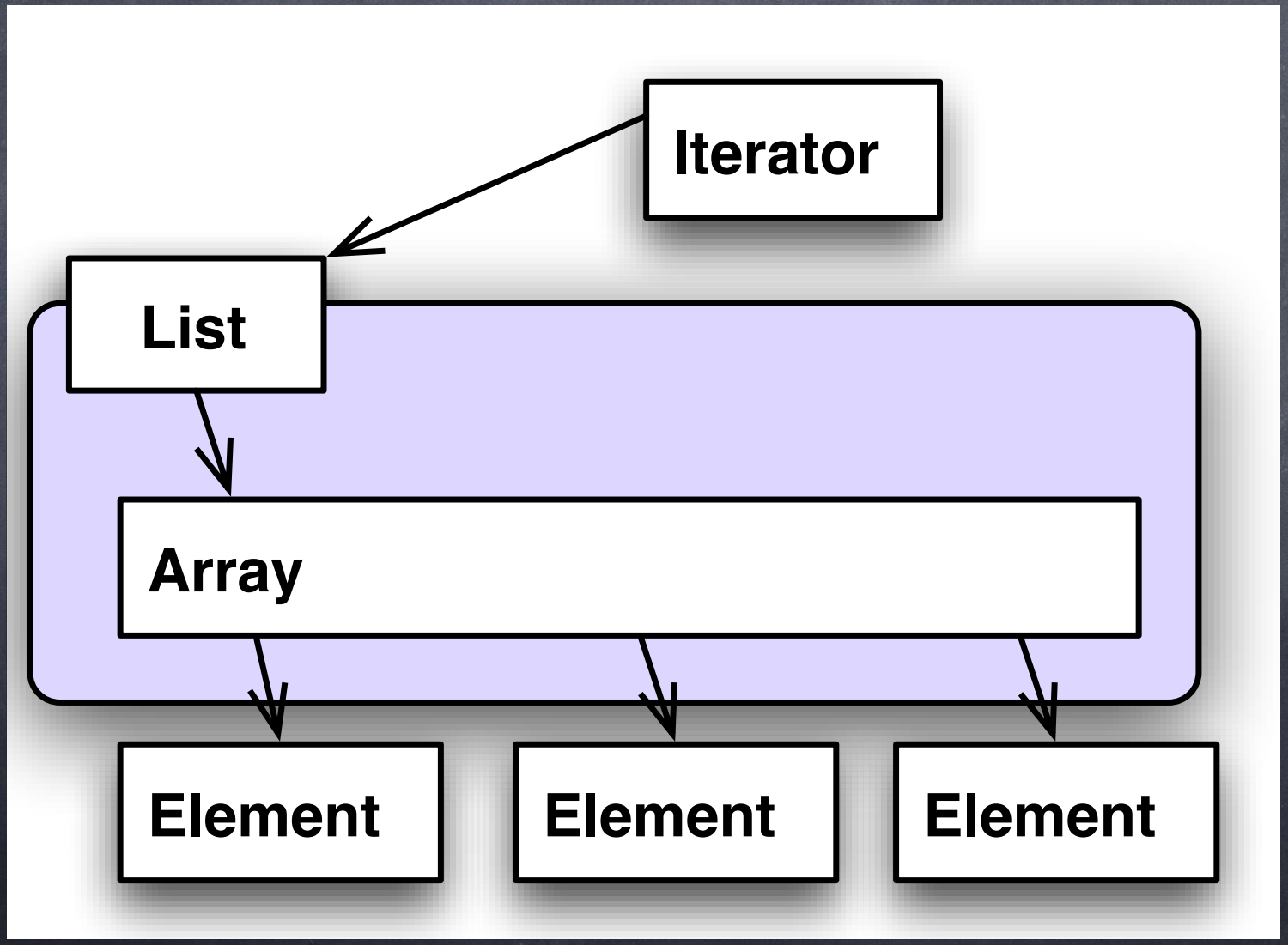
| | | Implementations | | | | |
|---|---|---|---|---|---|---|
| | | Hash Table | Resizable Array | Balanced Tree | Linked List | Hash Table + Linked List |
| Interfaces | Set | HashSet | | TreeSet | | LinkedHashSet |
| | List | | ArrayList | | LinkedList | |
| | Map | HashMap | | TreeMap | | LinkedHashMap |

- Effectively: Lists and Maps (implemented using Hash, Array, List or Tree).
- Our Implementations Available Online:

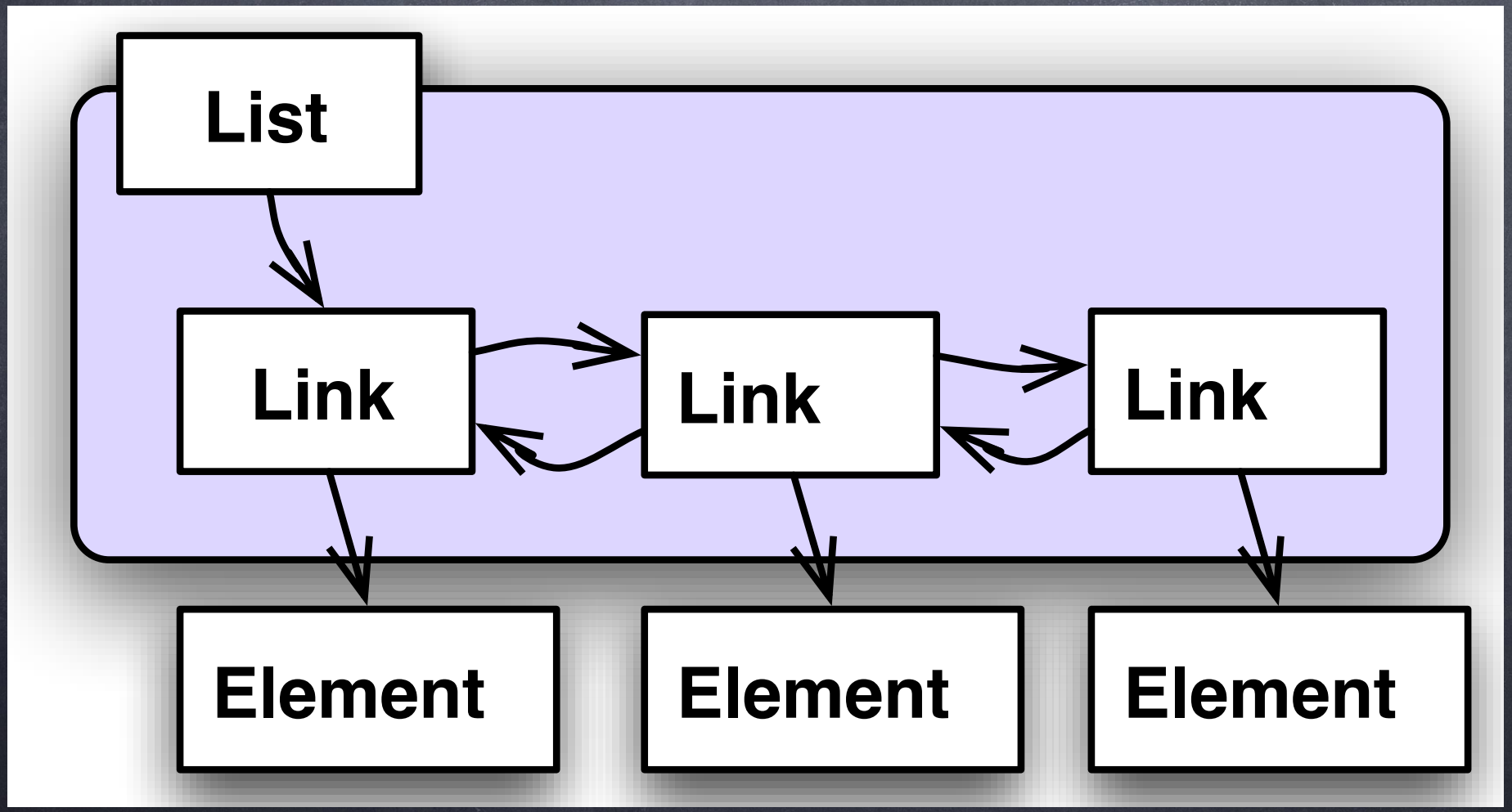
  http://homepages.ecs.vuw.ac.nz/~alex/software/

# Lists

# ArrayList (and Vector)



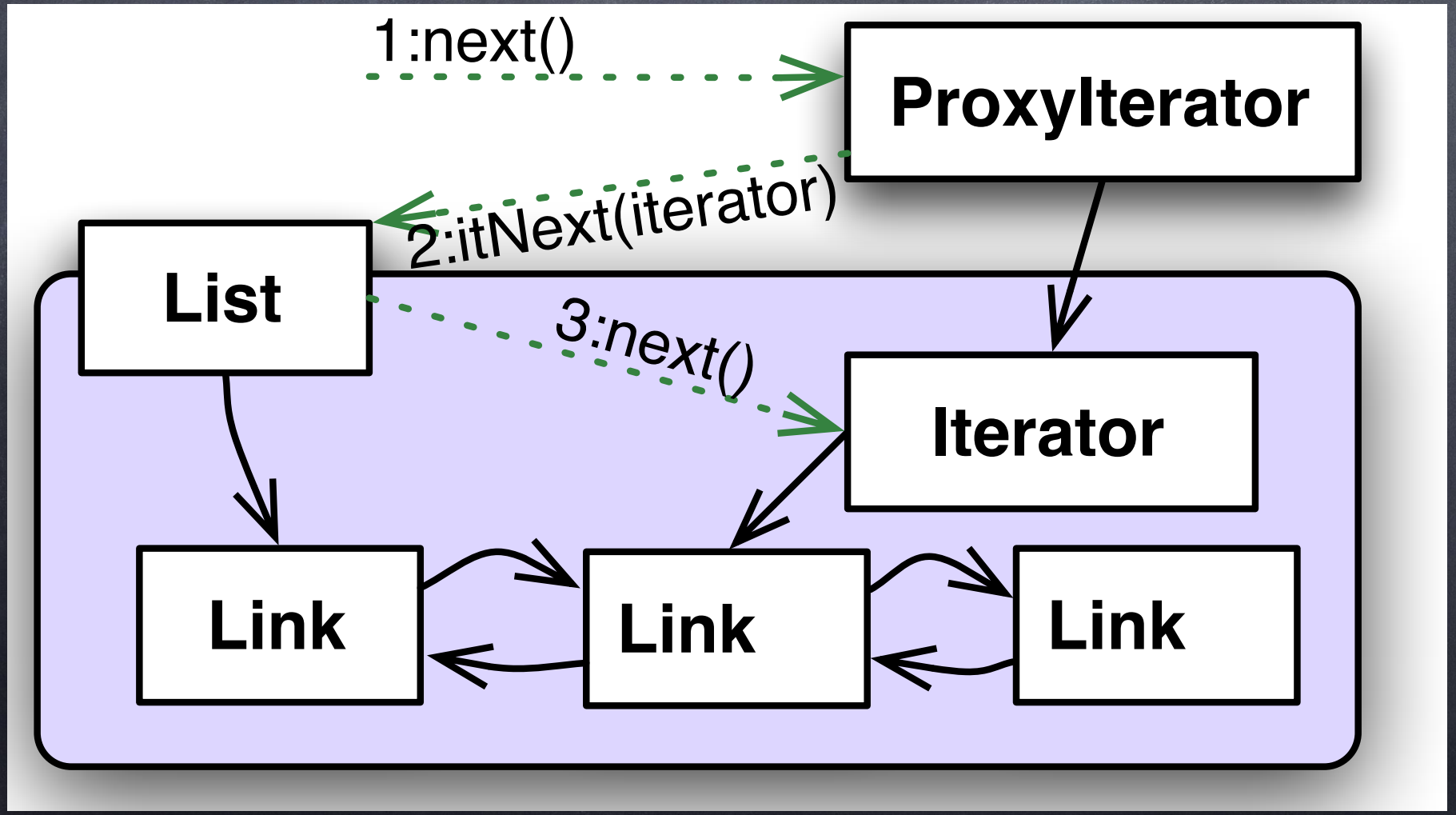
# What About LinkedList?



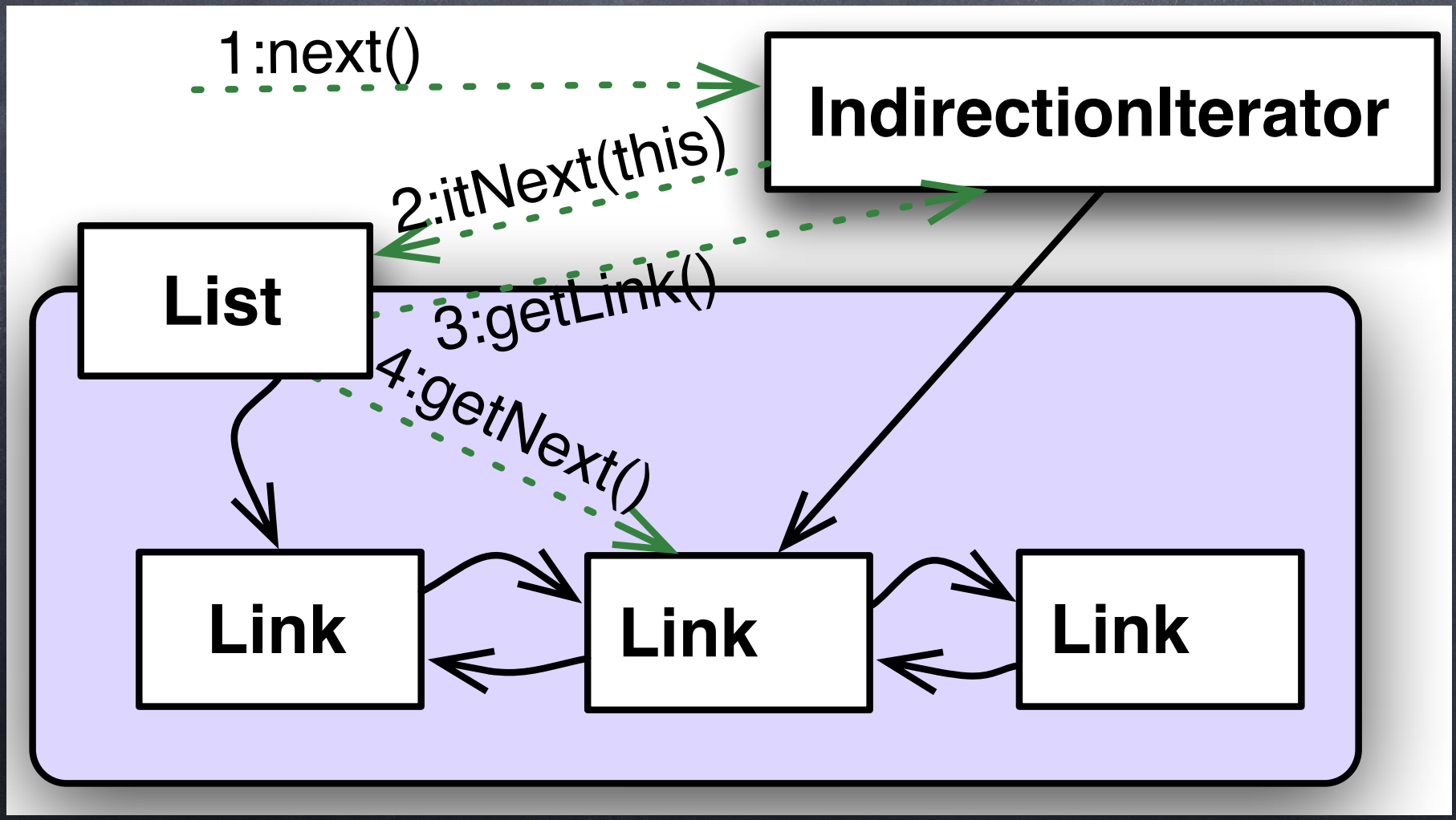
# Owner-as-Dominator

- Naive implementation that uses just the interface gives $O(N^2)$ iteration of the whole list (duh!)
- Single Place Cache (caching last accessed item and its index) improves it back to O(N) for non-random iteration - complies with specification

# Owner-as-Accessor (Proxy Iterator)



# Owner-as-Accessor (Indirection Iterator)

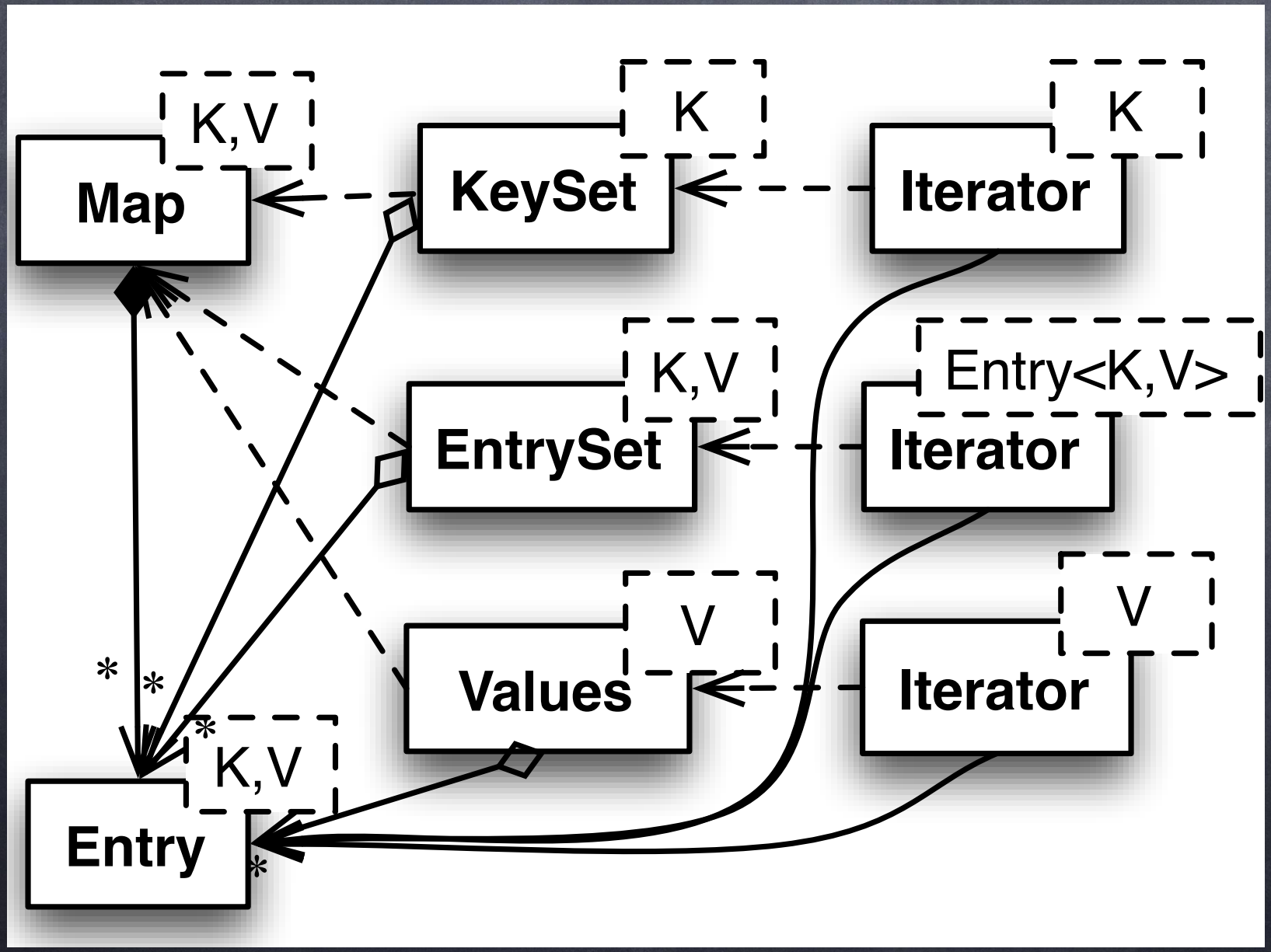


# Other Possible Iterators

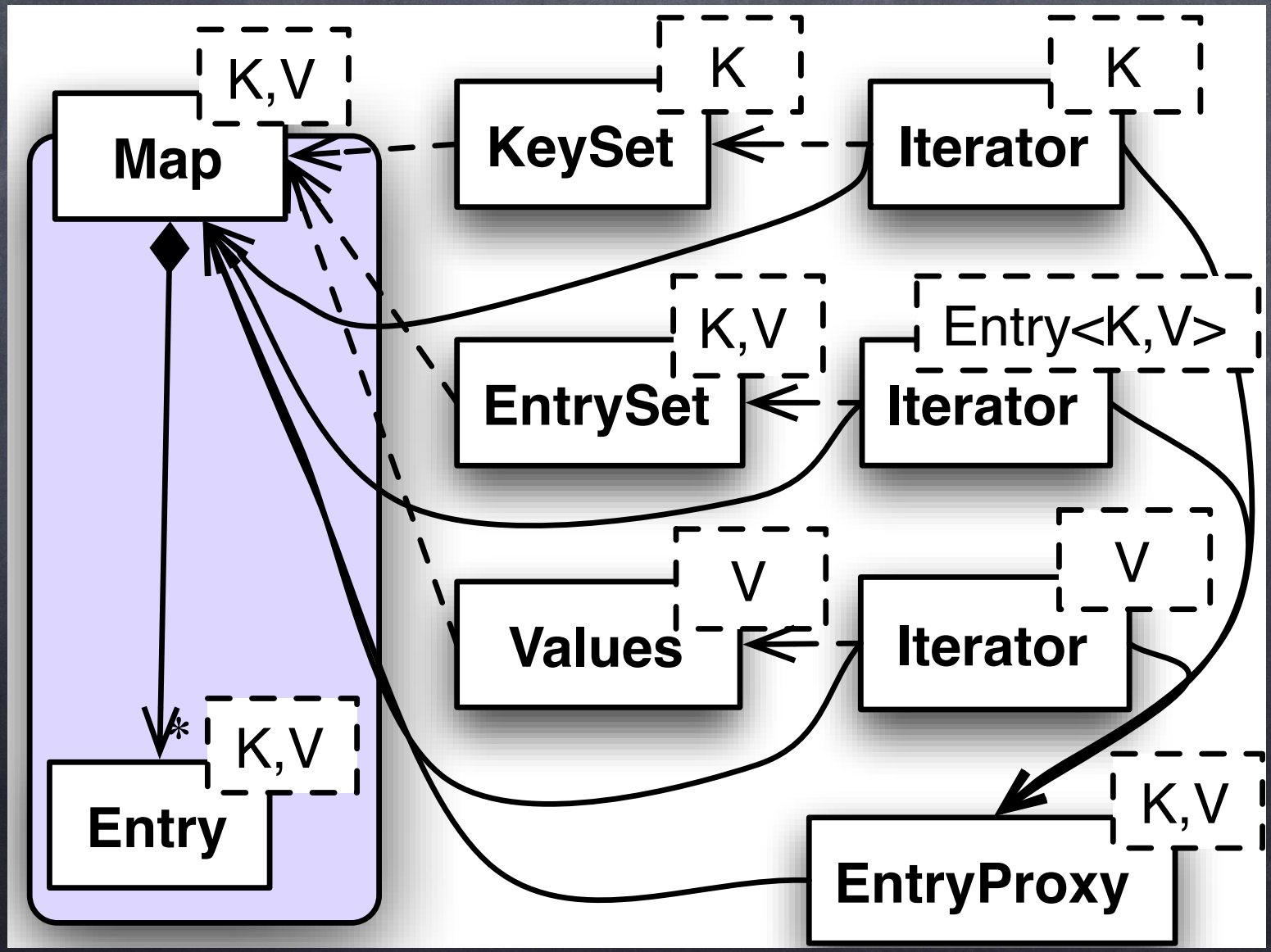
- "Magic Cookie" that uses a unique ID for each Iterator and stores its internal state inside a list in a hash map
- See: "Iterators and Encapsulation" by James Noble in TOOLS2000

# Maps

# Map Interfaces



# [Linked]HashMap and Hashtable



# Extended Map Interface

```
/*****************************************************************/
/* OWNERSHIP additions for map external iterators                */
/*****************************************************************/
int getModificationCount();
K getFirstKey();
K getNextKey(K key);
boolean hasNextKey(K key);
Map.Entry<K,V> entryProxyForKey(K key);
void forall(Procedure<K,V> proc);
```

# Summary of Map Changes

- HashMap's HashIterator was changed to utilise the extended Map interface
- LinkedHashMap was easier as the "getNextKey" could make use of linked list woven through the map's entries
- Hashtable (like Vector) was fundamentally the same but used different interfaces and was fully synchronised
- Finally, we provided 4 more owner-as-accessor refactorings following the ones described in LinkedList

# TreeMap

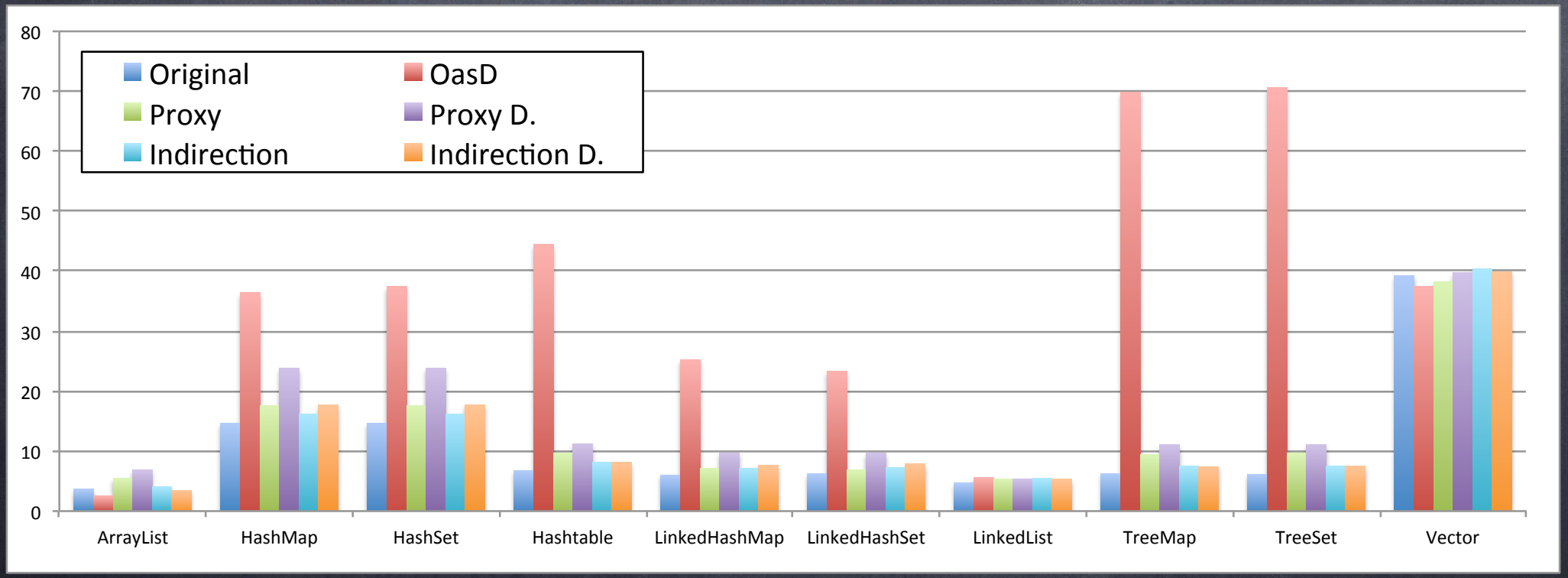
- Originally planned to refactor separately...
- Turns out we could re-use all the iterator, view, and entry objects from HashMap as they were just using Map's public interface!
- Calling getNextKey(currentKey) results in tracing down from the root of Red-Black tree (more costly and complex operation), but improvements with caching are possible

# Measurements

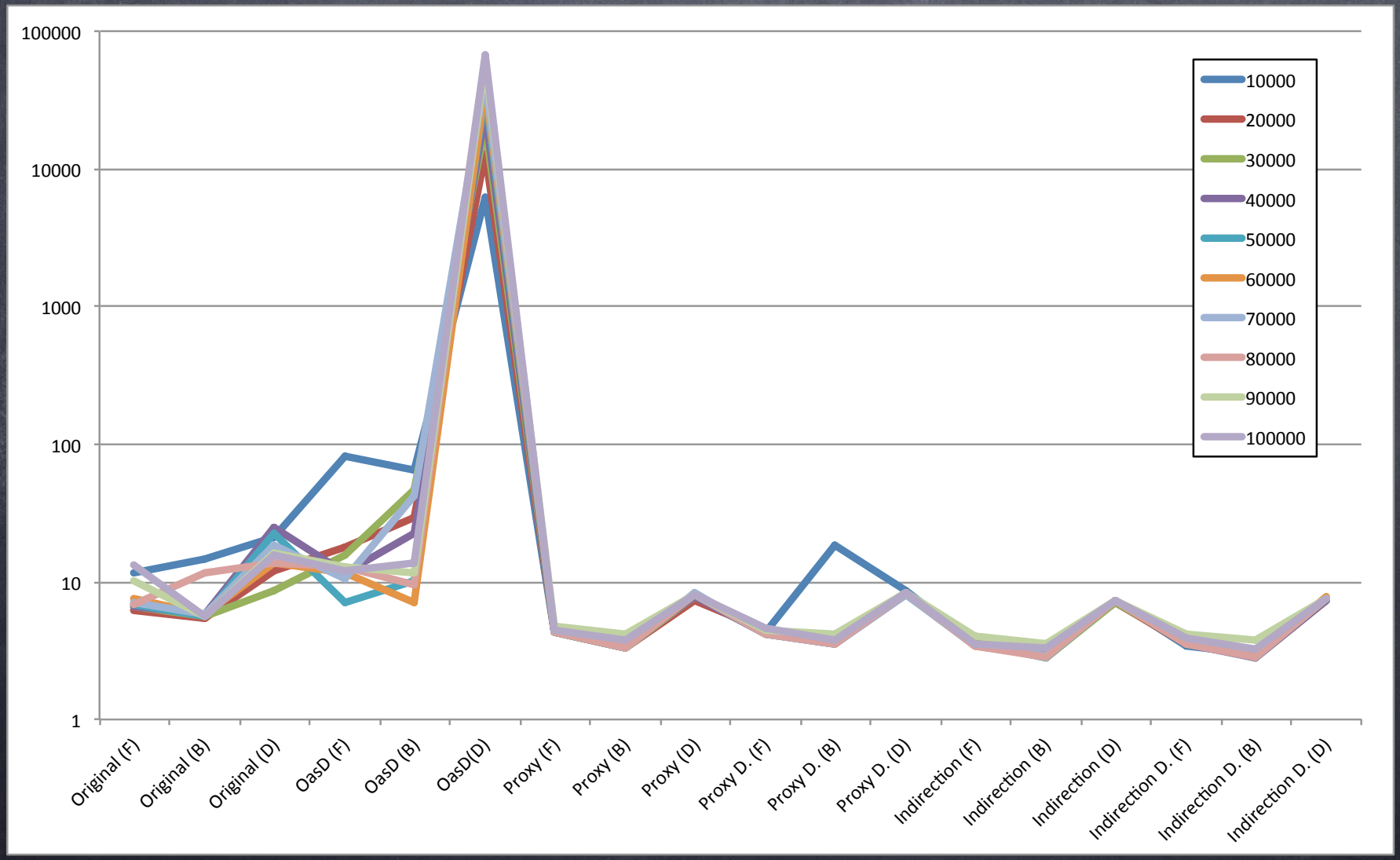
# Three Microbenchmarks

- Doug Lea's IteratorLoops from JSR166
- LinkedList Iteration: forward/backward/ disruptive
- Doug Lea's MapMicroBenchmark from JSR166
- NB! We ensured that GC and JIT did not interfere with our tests and ran each 25 times (see Georges et al OOPSLA2007)

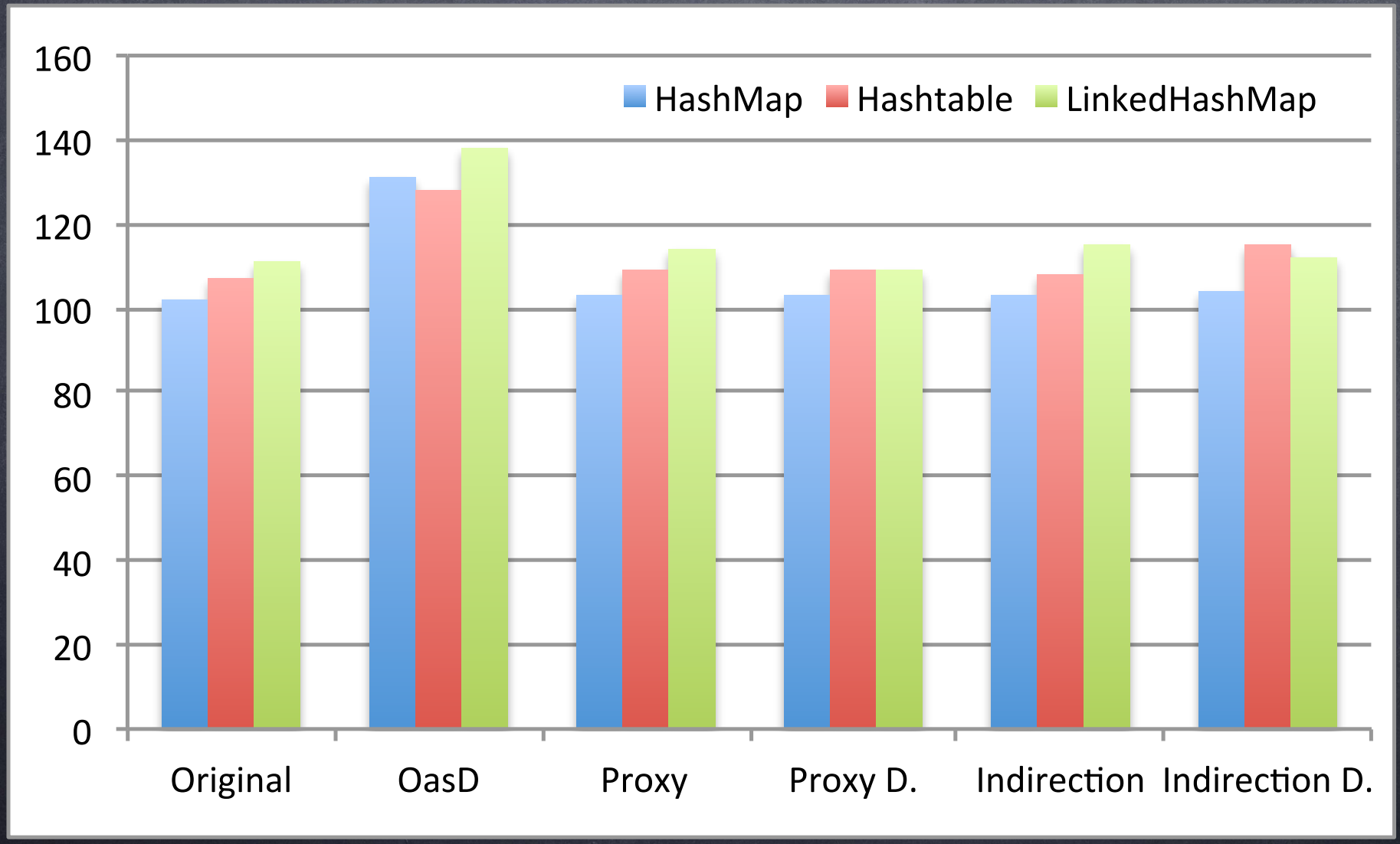
# IteratorLoops



# LinkedList Iteration



# MapMicroBenchmark



# Three Macrobenchmarks

- DaCapo (Typical Usage, Improves on SPEC)
- SPECjbb2005
- SPECjvm2008

TABLE I

FULL RESULTS TABLE (FORMAT: MEAN|SD; STATISTICALLY SIGNIFICANTLY DIFFERENT VALUES SHOWN IN BOLD)

| Benchmark | Original | | OasD | | Proxy | | Proxy D. | | Indirection | | Indirection D. | | Number | Percent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DaCapo (Time in *ms*; lower is better) | | | | | | | | | | | | | | |
| avrora | 23003 | 300 | **22781** | **415** | **22816** | **236** | 22867 | 179 | 22948 | 389 | 22873 | 323 | 219049309 | 78.08% |
| batik | 2516 | 34 | 2517 | 19 | 2520 | 25 | 2519 | 19 | 2519 | 25 | **2528** | **30** | 26507124 | 31.37% |
| eclipse | 53793 | 1031 | 53480 | 936 | 53716 | 1010 | 53812 | 1329 | **53554** | **1406** | 53608 | 738 | 355465429 | 33.63% |
| fop | 393 | 27 | 397 | 29 | 394 | 20 | 397 | 23 | 396 | 23 | 399 | 20 | 1874892 | 18.07% |
| h2 | 24133 | 580 | 24238 | 593 | 24141 | 517 | 24188 | 380 | 23967 | 320 | 23934 | 375 | 90175446 | 8.04% |
| jython | 15041 | 215 | **15476** | **100** | **15725** | **107** | **15719** | **53** | **15837** | **110** | **17050** | **145** | 159700109 | 7.49% |
| luindex | 705 | 18 | **687** | **23** | 714 | 22 | 713 | 19 | 710 | 40 | 718 | 51 | 327466 | 36.66% |
| lusearch | 7251 | 184 | 7334 | 105 | **7181** | **189** | **7120** | **198** | 7226 | 299 | 7325 | 78 | 11979688 | 5.45% |
| pmd | 3944 | 47 | **3992** | **39** | **4046** | **59** | **4065** | **72** | **4005** | **64** | **4054** | **67** | 10712544 | 36.55% |
| sunflow | 22656 | 518 | 22560 | 365 | **23365** | **126** | 22970 | 711 | 22851 | 523 | **22331** | **207** | 171198077 | <0.01% |
| tomcat | 7576 | 108 | 7641 | 135 | **7687** | **134** | **7736** | **111** | **7733** | **88** | **7661** | **118** | 16726923 | 13.95% |
| tradebeans | 27952 | 409 | **27556** | **494** | **28258** | **506** | 28142 | 275 | 27998 | 328 | 28020 | 340 | 1621619 | 33.00% |
| tradesoap | 64476 | 1251 | 65193 | 1549 | 65042 | 1712 | 65119 | 1463 | 64390 | 1378 | 65111 | 1499 | 1631193 | 32.82% |
| xalan | 26604 | 384 | 26692 | 318 | **26383** | **247** | **26173** | **258** | **26125** | **251** | **26310** | **291** | 61153799 | 13.23% |
| SPECjbb2005 (Throughput; higher is better) | | | | | | | | | | | | | | |
| SPECjbb2005 | 29598 | 405 | **14062** | **181** | **28825** | **860** | **28540** | **619** | **28959** | **764** | **28394** | **641** | 35542855 | 4.87% |
| SPECjvm2008 (Time in *ms*; lower is better) | | | | | | | | | | | | | | |
| compress | 46.56 | 0.81 | 46.77 | 0.59 | 46.71 | 0.59 | 46.82 | 0.42 | 46.83 | 0.42 | 46.84 | 0.57 | 199478 | 20.21% |
| crypto.aes | 18.49 | 0.18 | 18.40 | 0.13 | 18.46 | 0.10 | 18.48 | 0.18 | **18.42** | **0.10** | 18.48 | 0.19 | 254853 | 22.00% |
| crypto.rsa | 35.04 | 0.25 | 34.89 | 0.28 | 34.92 | 0.31 | 34.95 | 0.36 | 34.94 | 0.25 | 34.95 | 0.27 | 6535358 | 11.18% |
| crypto.signverify | 53.06 | 0.27 | 52.97 | 0.31 | 53.11 | 0.29 | 53.16 | 0.36 | 53.09 | 0.38 | 53.12 | 0.29 | 3290783 | 2.13% |
| derby | 21.55 | 0.48 | 21.63 | 0.40 | 21.56 | 0.50 | 21.73 | 0.43 | 21.59 | 0.38 | 21.61 | 0.33 | 89061937 | 3.04% |
| mpegaudio | 15.08 | 0.05 | 15.10 | 0.05 | 15.08 | 0.06 | 15.11 | 0.05 | 15.10 | 0.06 | 15.07 | 0.06 | 209089 | 20.32% |
| fft.large | 23.61 | 0.27 | 23.53 | 0.30 | 23.49 | 0.27 | **23.43** | **0.28** | 23.55 | 0.24 | 23.51 | 0.38 | 168290 | 23.78% |
| fft.small | 82.75 | 4.49 | 84.24 | 5.21 | 84.98 | 3.94 | 83.35 | 4.01 | 84.46 | 4.70 | 83.57 | 4.26 | 439646 | 9.22% |
| lu.large | 6.98 | 1.44 | 6.69 | 1.24 | 6.72 | 1.22 | 7.05 | 1.39 | 6.34 | 0.96 | 7.02 | 1.42 | 166728 | 23.92% |
| lu.small | 107.98 | 0.87 | 107.61 | 0.92 | 107.48 | 0.75 | 107.58 | 0.84 | 107.94 | 0.70 | 107.82 | 0.85 | 642713 | 6.39% |
| monte_carlo | 15.33 | 1.49 | 15.33 | 1.48 | 15.65 | 0.10 | 15.67 | 0.11 | 15.63 | 0.03 | 15.29 | 1.47 | 179253 | 22.58% |
| sor.large | 13.01 | 0.02 | 13.01 | 0.02 | 13.01 | 0.01 | 13.02 | 0.02 | **12.99** | **0.05** | 13.01 | 0.01 | 167449 | 23.92% |
| sor.small | 57.47 | 0.11 | 57.47 | 0.10 | 57.49 | 0.10 | 57.47 | 0.12 | 57.43 | 0.07 | 57.44 | 0.10 | 193425 | 21.16% |
| sparse.large | 11.51 | 0.23 | **11.97** | **1.22** | 11.70 | 0.28 | 11.82 | 0.80 | 11.71 | 0.34 | 11.77 | 0.36 | 166691 | 23.98% |
| sparse.small | 43.87 | 0.11 | 43.79 | 0.12 | 43.80 | 0.18 | 43.83 | 0.11 | 43.85 | 0.18 | 43.80 | 0.15 | 182676 | 22.17% |
| serial | 33.22 | 0.98 | **32.46** | **0.96** | 33.27 | 0.81 | 33.04 | 0.92 | 32.88 | 1.31 | 33.31 | 0.89 | 51123977 | 5.68% |
| sunflow | 20.51 | 0.50 | 20.48 | 0.50 | 20.55 | 0.32 | 20.42 | 0.52 | **20.22** | **0.63** | 20.51 | 0.47 | 42506559 | 0.12% |
| xml.validation | 63.38 | 1.09 | 63.01 | 0.96 | 63.36 | 1.03 | 63.19 | 1.42 | 63.75 | 1.39 | 63.23 | 1.03 | 10318760 | 3.33% |

# Macrobenchmarks

- Full table available in our ICSE2013 paper and an accompanying Technical Report ECSTR12-22:

  http://ecs.victoria.ac.nz/Main/TechnicalReportSeries

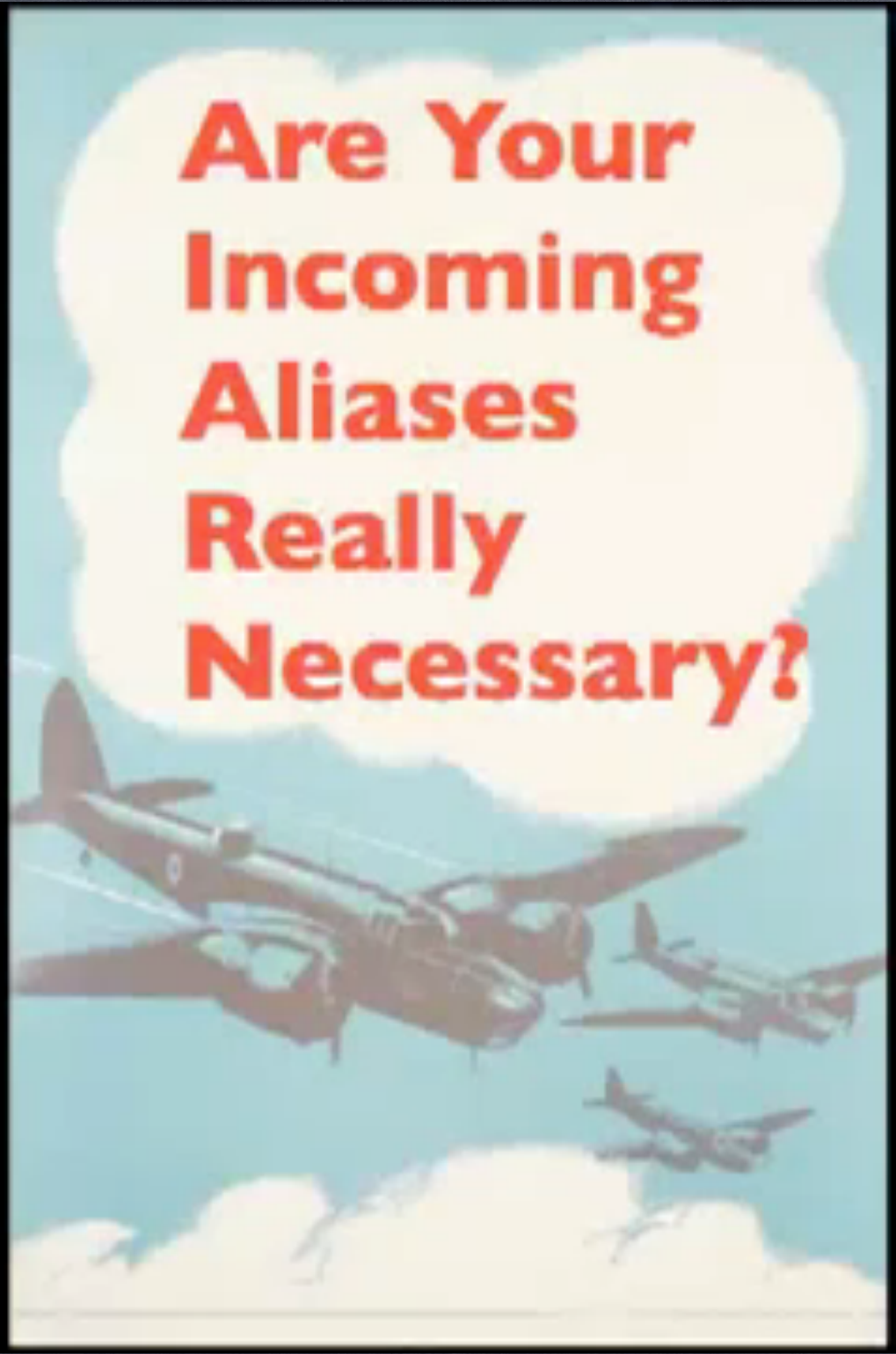
- In only 40 out of 165 refactored benchmarks have we detected a statistically significant slow down!
- SPECjbb2005 is the heaviest user of collections (around 8% of its running time spent in java.util.* methods according to option `-Xrunhprof:cpu-times`) and as a result slowed down the most

# Conclusions

- Encapsulation reduces performance by factors of 2 to 8 (in particular on microbenchmarks)
- Owner-as-dominator is the worst (as expected)
- Owner-as-accessor (even with dynamic checking) only produces less than 3% slowdown on macrobenchmarks
- **We hope these results may encourage object-oriented designers to consider object encapsulation more carefully when designing their programs - especially their use of incoming aliases to circumvent encapsulation - and to ask themselves: are their incoming aliases really necessary?**

# Since 2013...

- State of the macro benchmarks is still "to be improved": some recent 2019 developments on DeCapo update *as well as* a competing performance corpora by Oracle Labs just released at PLDI 2019 (Renaissance - controversial)
- New common data structure libraries but still not necessarily treating aliasing more than an algorithmic side effect.


Are Your
Incoming
Aliases
Really
Necessary?


# Are your Incoming Aliases Really Necessary?

Counting the Cost of Object Ownership

Alex Potanin,
Monique Damitio,
James Noble

**Fri 24 May Session 2.3**