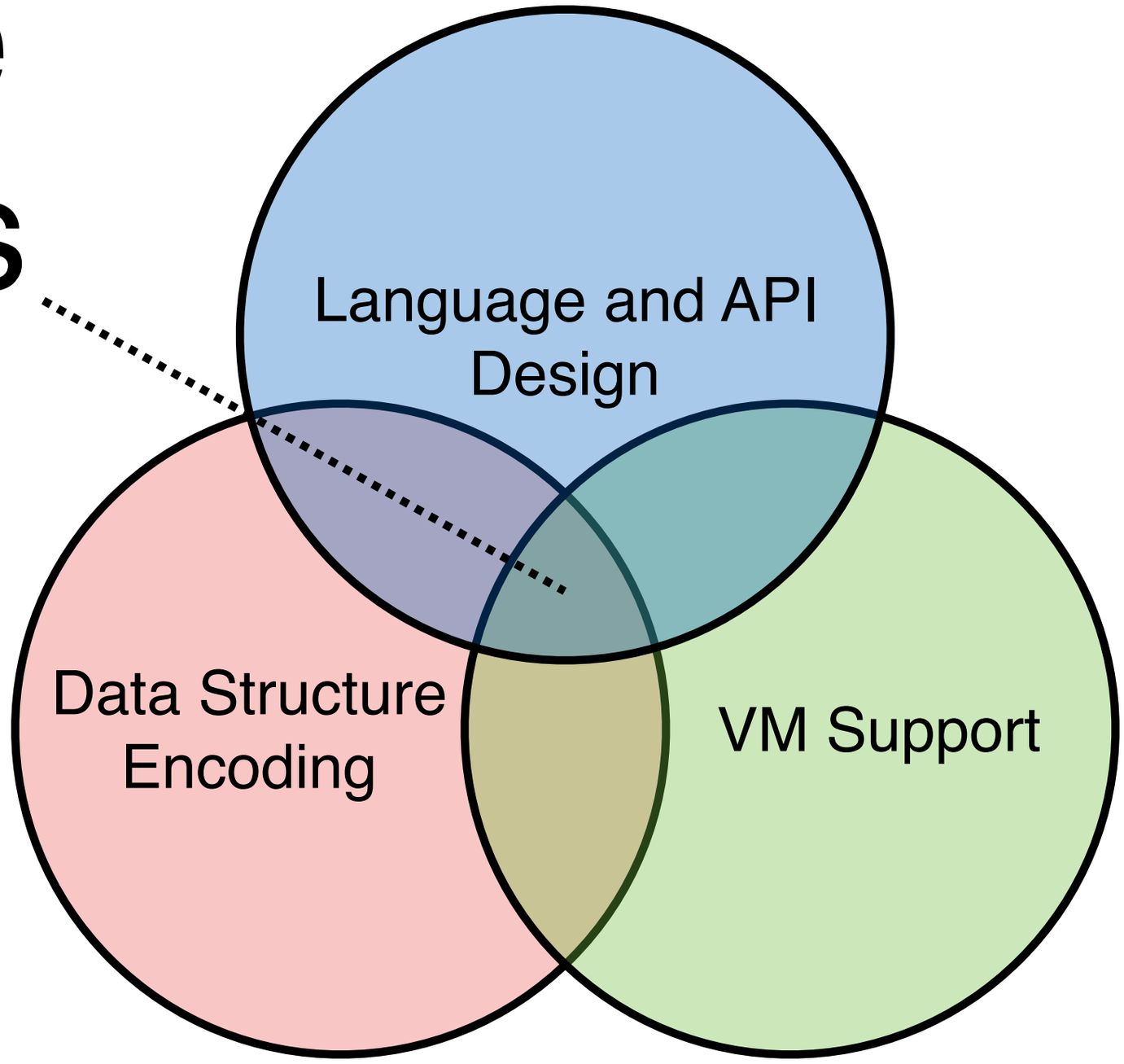
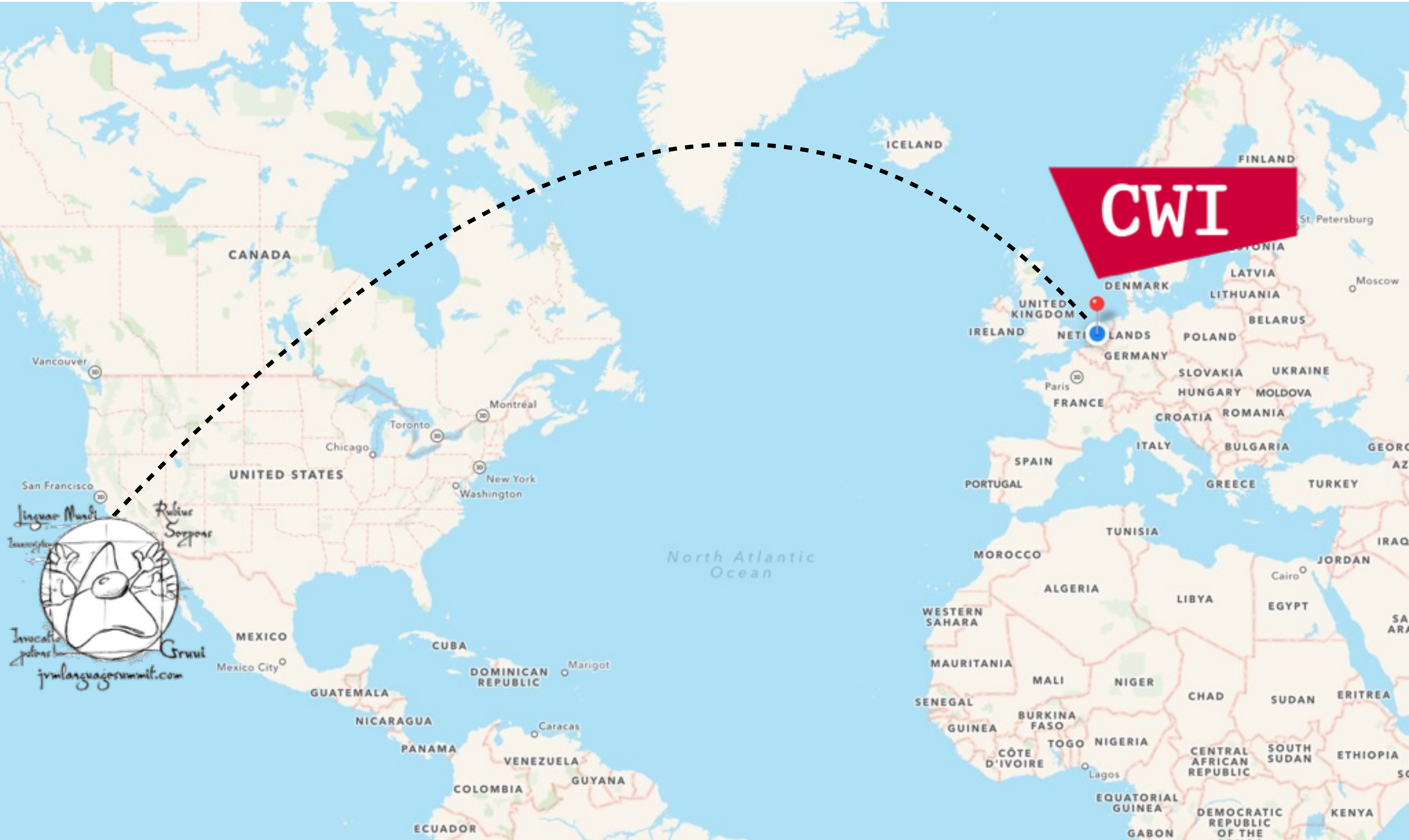
Immutable Collections

Michael J. Steindorfer

mail: michael@cwi.nl

twitter: @loopingoptimism




CWI
CANADA
UNITED STATES
MEXICO
CUBA
DOMINICAN REPUBLIC
GUATEMALA
NICARAGUA
PANAMA
VENEZUELA
GUYANA
COLOMBIA
ECUADOR
Vancouver
San Francisco
Chicago
Toronto
Montréal
New York
Washington
Mexico City
Marigot
Caracas
North Atlantic Ocean
ICELAND
FINLAND
St. Petersburg
LATVIA
LITHUANIA
Moscow
BELARUS
DENMARK
UNITED KINGDOM
IRELAND
POLAND
GERMANY
Paris
FRANCE
SLOVAKIA
UKRAINE
HUNGARY
MOLDOVA
ROMANIA
CROATIA
ITALY
BULGARIA
SPAIN
PORTUGAL
GREECE
TURKEY
TUNISIA
MOROCCO
ALGERIA
LIBYA
Cairo
EGYPT
JORDAN
WESTERN SAHARA
MAURITANIA
MALI
NIGER
CHAD
SUDAN
ERITREA
SENEGAL
GUINEA
BURKINA FASO
TOGO
NIGERIA
CÔTE D'IVOIRE
Lagos
CENTRAL AFRICAN REPUBLIC
SOUTH SUDAN
ETHIOPIA
EQUATORIAL GUINEA
GABON
DEMOCRATIC REPUBLIC OF THE
KENYA
Linguae Mundi
jvmlanguagesummit.com

"A collection represents a **group of objects** […]"

–java.util.Collection API

Data Type Semantics

lists, sets, maps, multi-maps, bags, …

Ordering

unordered (hashed), ordered

Update Semantics

mutable, transient, immutable

Processing Semantics

sequential, concurrent, parallel

Encoding

array/hashtable, linked list, tree, trie, …

Content

homogeneous, heterogeneous

…

Data Type Semantics

sets, maps, multi-maps

Ordering

unordered (hashed)

Update Semantics

immutable

Processing Semantics

sequential

Encoding

trie

Content

homogeneous, heterogeneous

...

Why Immutability?

- More Optimizations (Constant)
- Simple (Programming Model)
- Robust (Concurrency)

"**Classes should be immutable** unless there's a very good reason to make them mutable."

–Effective Java, 2nd Edition (Joshua Bloch, 2008)
Item 15: Minimize mutability

"**[Collections] should be immutable** unless there's a very good reason to make them mutable."

–Effective Java, 2nd Edition (Joshua Bloch, 2008)
Item 15: Minimize mutability

"Immutable [collections] provide **many advantages**, and their **only disadvantage is the potential for performance problems** under certain circumstances."

–Effective Java, 2nd Edition (Joshua Bloch, 2008)
Item 15: Minimize mutability

Immutable collections are **challenging to optimize**, but overall they **constitute a more sensible default**.

–Personal Opinion

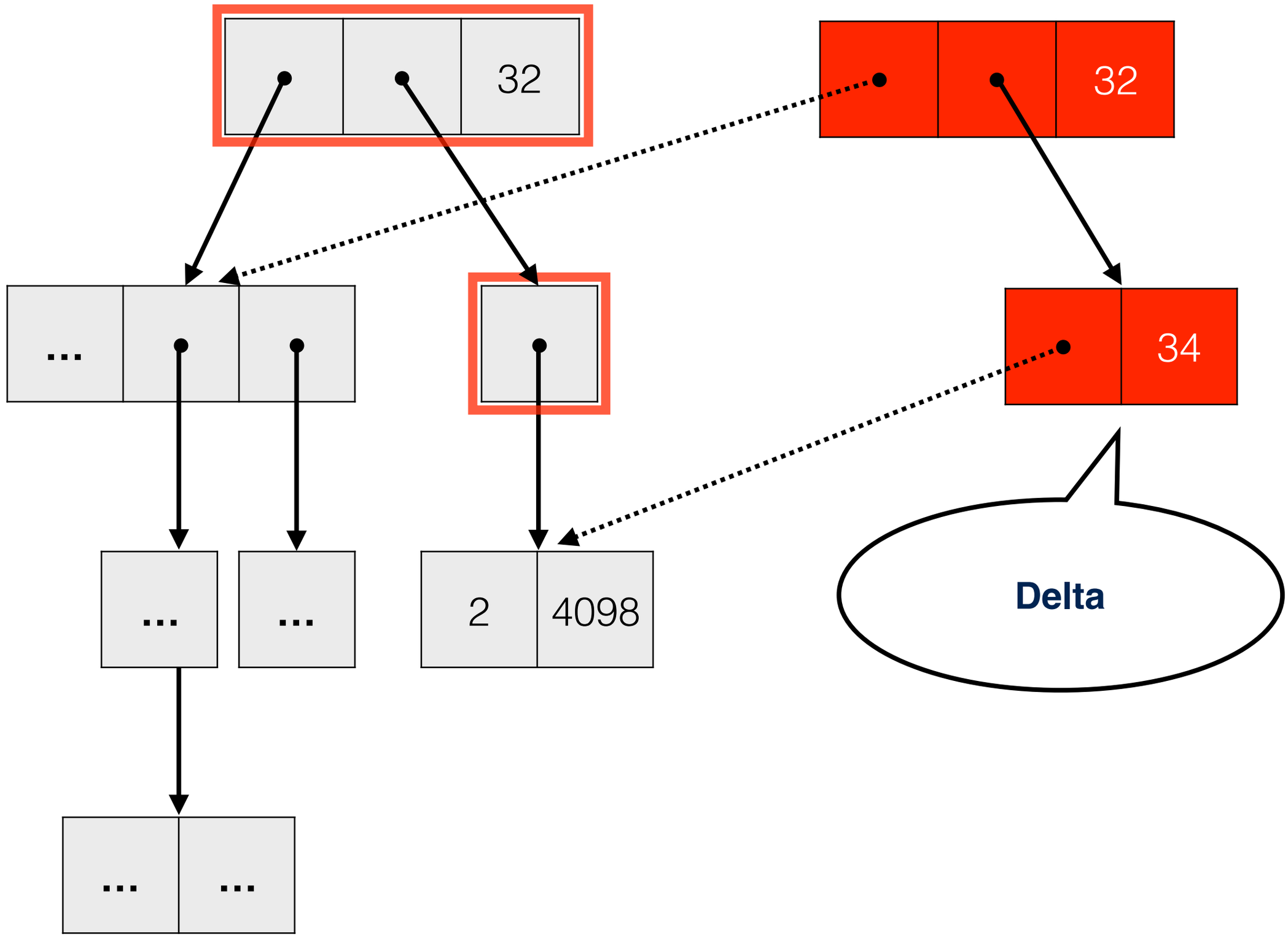
Outline

1. Core Principles & Encodings
2. Minimize Memory Footprint
3. Efficiency & Expressivity
4. Performance Challenges

Core Principles

```
Set<Integer> updatedSet =
  hugeSet.newInstanceWith(34);
```

```
Set<Integer> __tmp =
  new HashSet<>(hugeSet.size + 1);
__tmp.addAll(hugeSet);
__tmp.add(34);

Set<Integer> updatedSet =
  Collections.unmodifiableSet(__tmp);
```

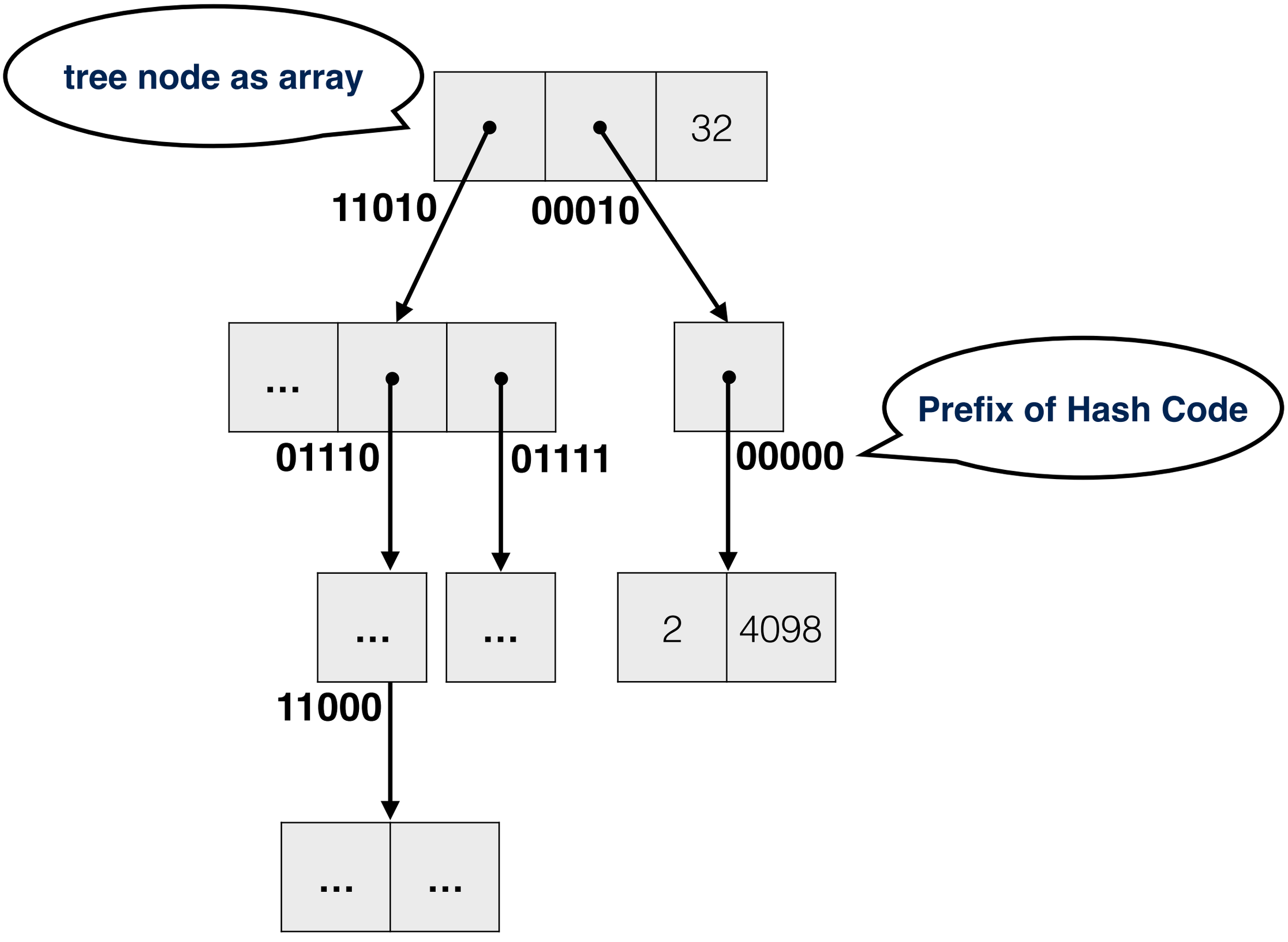

32
32
...
34
...
...
2
4098
Delta
...
...

Clojure
http://clojure.org

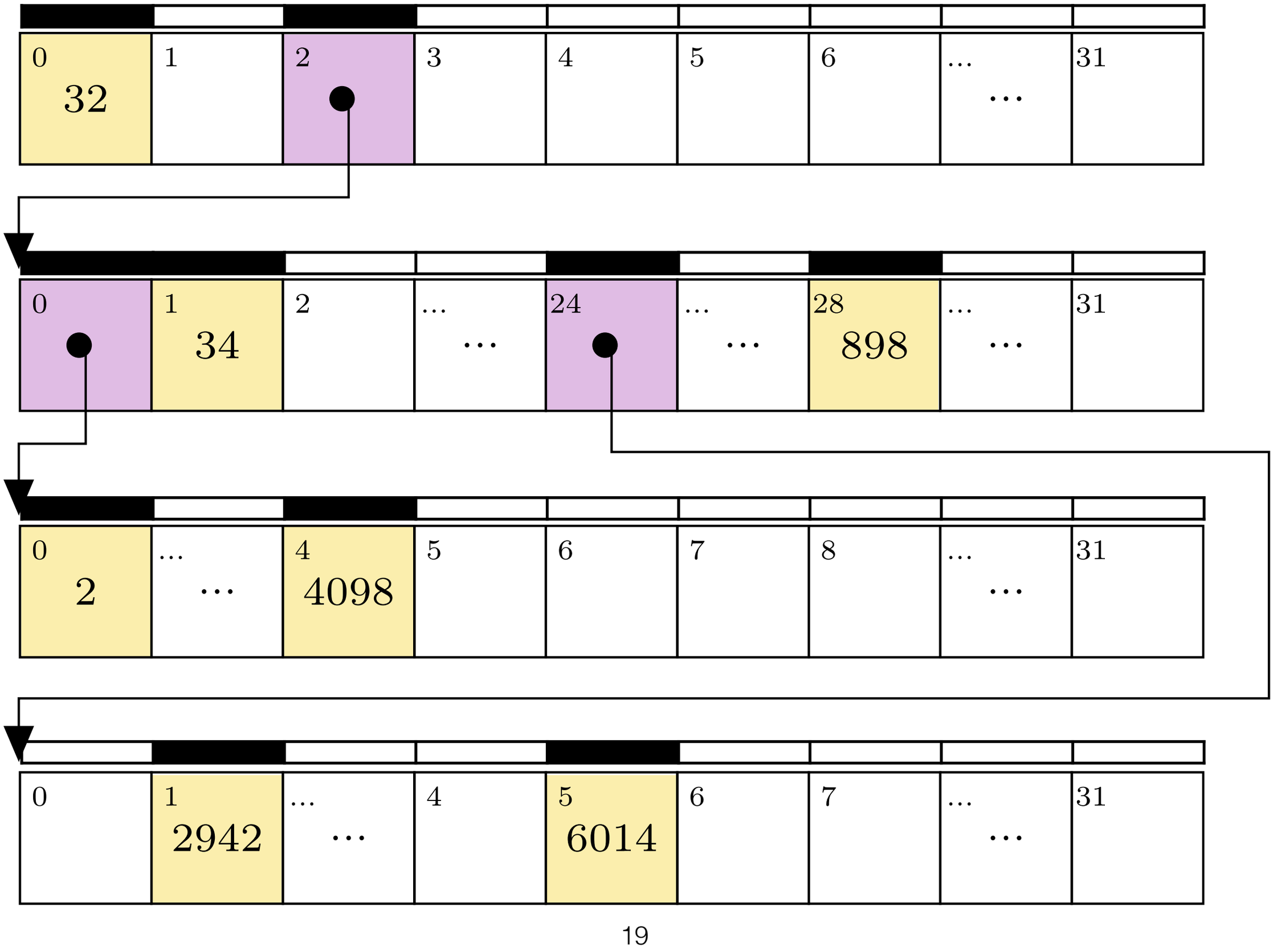
Rascal
http://rascal-mpl.org

Scala
http://scala-lang.org

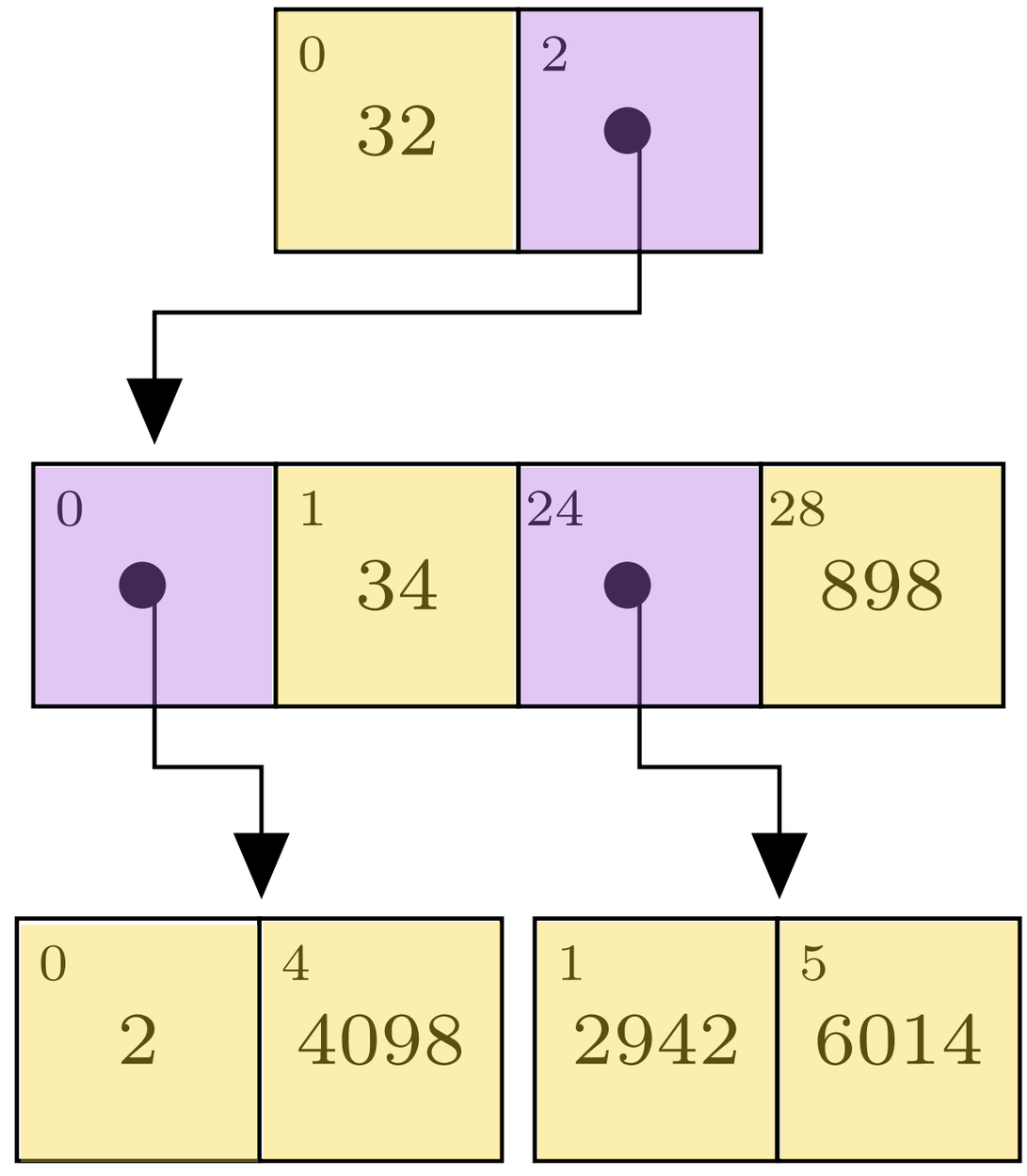
Hash Array Mapped Trie


tree node as array
32
11010
00010
...
01110
01111
00000
Prefix of Hash Code
...
...
2
4098
11000
...
...

{2, 32, 34, 898,
2942, 4098, 6014}


0
32
1
2
3
4
5
6
...
...
31
0
1
34
2
...
...
24
...
...
28
898
...
...
31
0
2
...
...
4
4098
5
6
7
8
...
...
31
0
1
2942
...
...
4
5
6014
6
7
...
...
31

```
class Node {
  int bitmap;
  Object[] content;
}
```

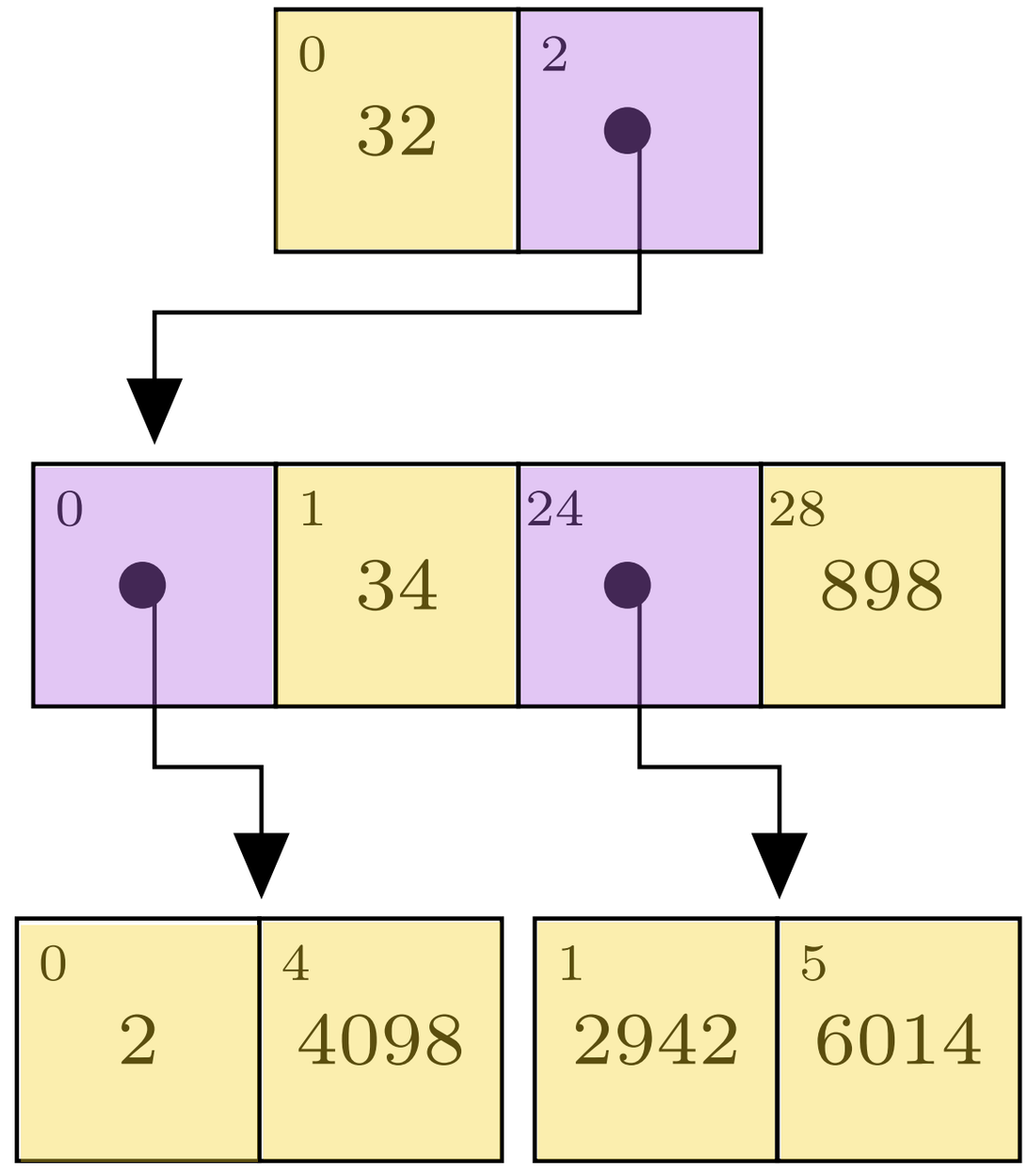


What can you expect?

- Shallow and Wide Search Trees ($\mathrm{Depth} <= 7$)
- Lookup/Insert/Delete in $\mathrm{O}(\log_{32}(\mathrm{n})) \cong \mathrm{O}(1)$
- Union/Subset (Structural Operations)

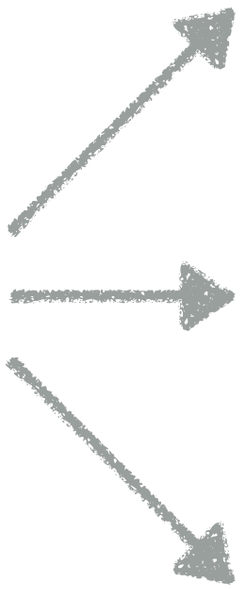
Minimize Memory Footprint by Specializing Class Layouts

```
class Node {
  int bitmap;
  Object[] content;
}
```



```
class ArrayList32 {
  /* 0 <= content.size <= 32 */
  Object[] content;
}
```

```
class ArrayList32 {
  Object[] content;
}
```



```
class List1 implements
    ImmutableList {
  Object slot0;
}

class List2 implements
    ImmutableList {
  Object slot0;
  Object slot1;
}

class List3 implements
    ImmutableList {
  Object slot0;
  Object slot1;
  Object slot2;
}

...
```

-  Small Footprints
-  Megamorphic Call Sites
-  Code Bloat

```
abstract class List2 implements ImmutableList {
  Object slot0;
  Object slot1;

  @Override
  public ImmutableList insertAtIndex(
      int index, Object item) {
    switch (index) {
      case 0:
        return new List3(item, slot0, slot1);
      case 1:
        return new List3(slot0, item, slot1);
      case 2:
        return new List3(slot0, slot1, item);
      default:
        throw new IllegalArgumentException();
    }
  }
}
```

-  Small Footprints
-  Monomorphic Call Sites
-  JIT-Friendly

Decoupling Data and Operations

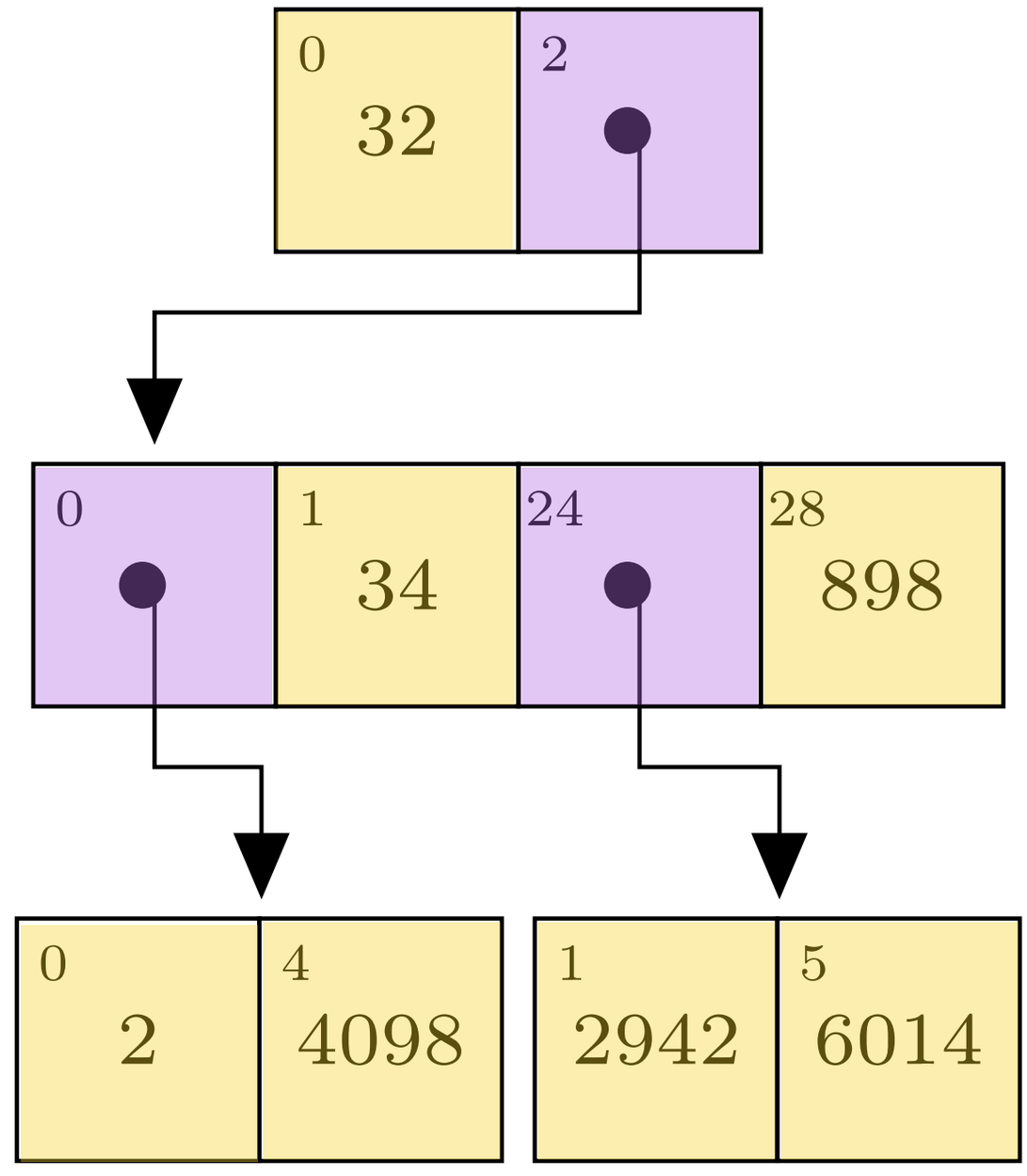
```
abstract class List2 implements ImmutableList {
  Object slot0;
  Object slot1;
}
```

```
// viewing consecutive fields as arrays
createArrayView(List2.class,
                "slot0", “slot1”, Object.class);
```

```
ImmutableList insertAtIndex(int index, Object item) {
  ArrayView<Object> src = getArrayView();
  ImmutableList newList = allocateList(src.length + 1);
  ArrayView<Object> dst = newList.getArrayView();
  // copy 'src' and insert element at position 'index'
  arrayviewcopy(src, 0, dst, 0, index);
  dst.set(index, item);
  arrayviewcopy(src, index, dst, index + 1,
    src.length - index);

  return newNode;
}
```

Efficiency & Expressivity

```
class Node {
  int bitmap;
  Object[] content;
}
```



More Performant …

Up to 6x faster …

(speedup of iteration over Clojure / Scala)

Steindorfer, M. J., & Vinju, J. J. (2015). Optimizing Hash-Array Mapped Tries for Fast and Lean Immutable JVM Collections. OOPSLA'15.

… up to 28x faster …

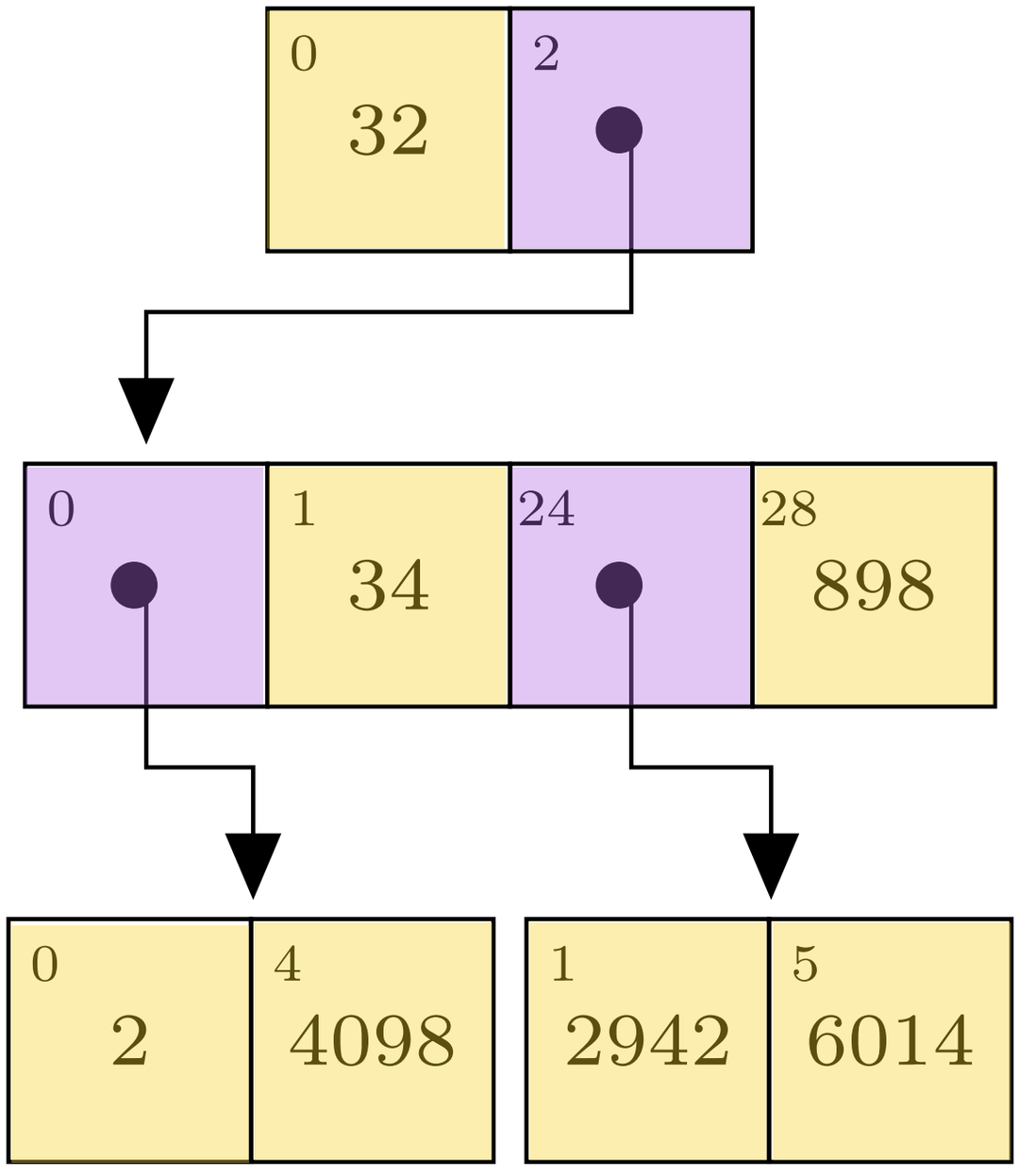
(speedup equality checking over Clojure / Scala)

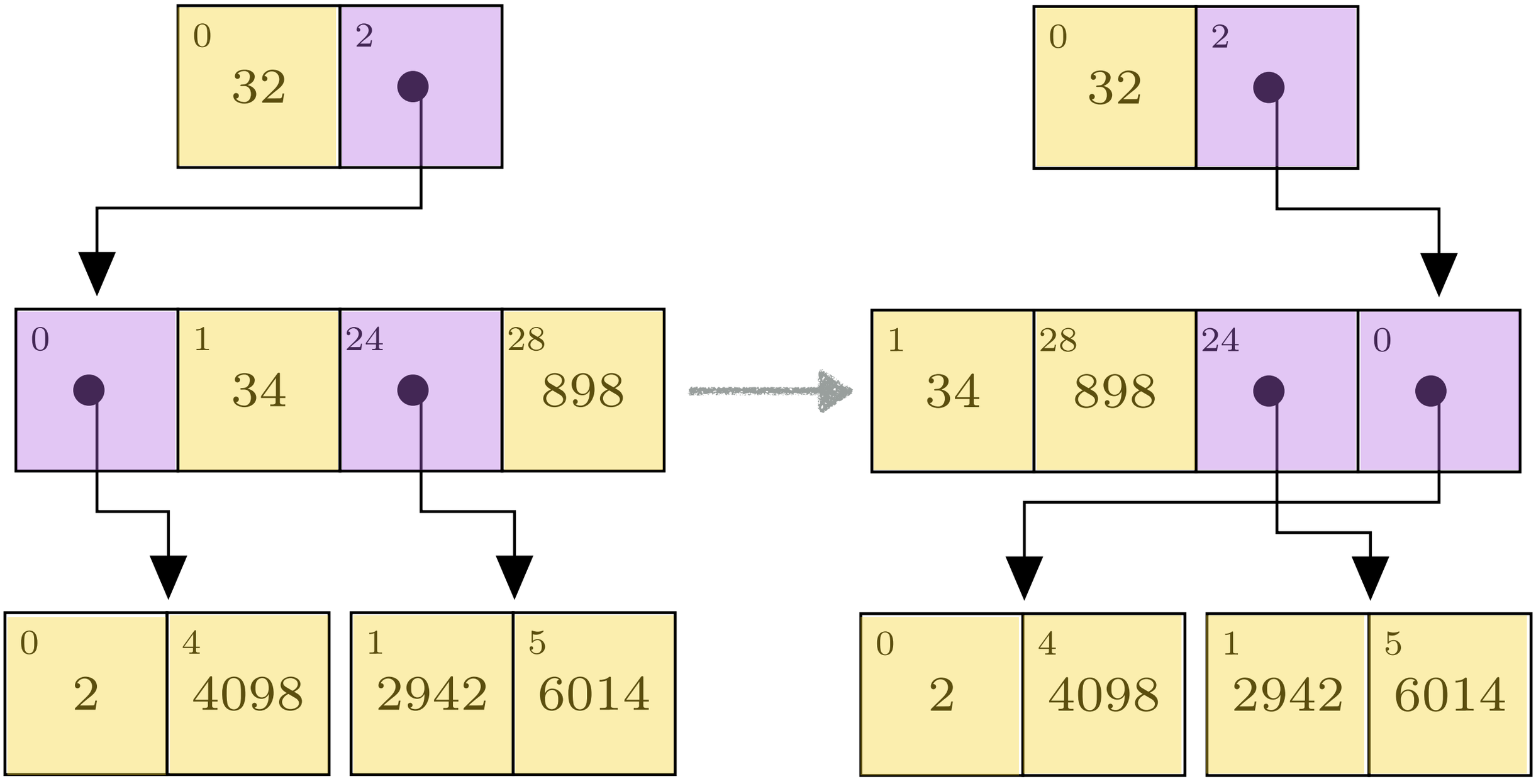
Steindorfer, M. J., & Vinju, J. J. (2015). Optimizing Hash-Array Mapped Tries for Fast and Lean Immutable JVM Collections. OOPSLA'15.

… and up to 64% smaller.

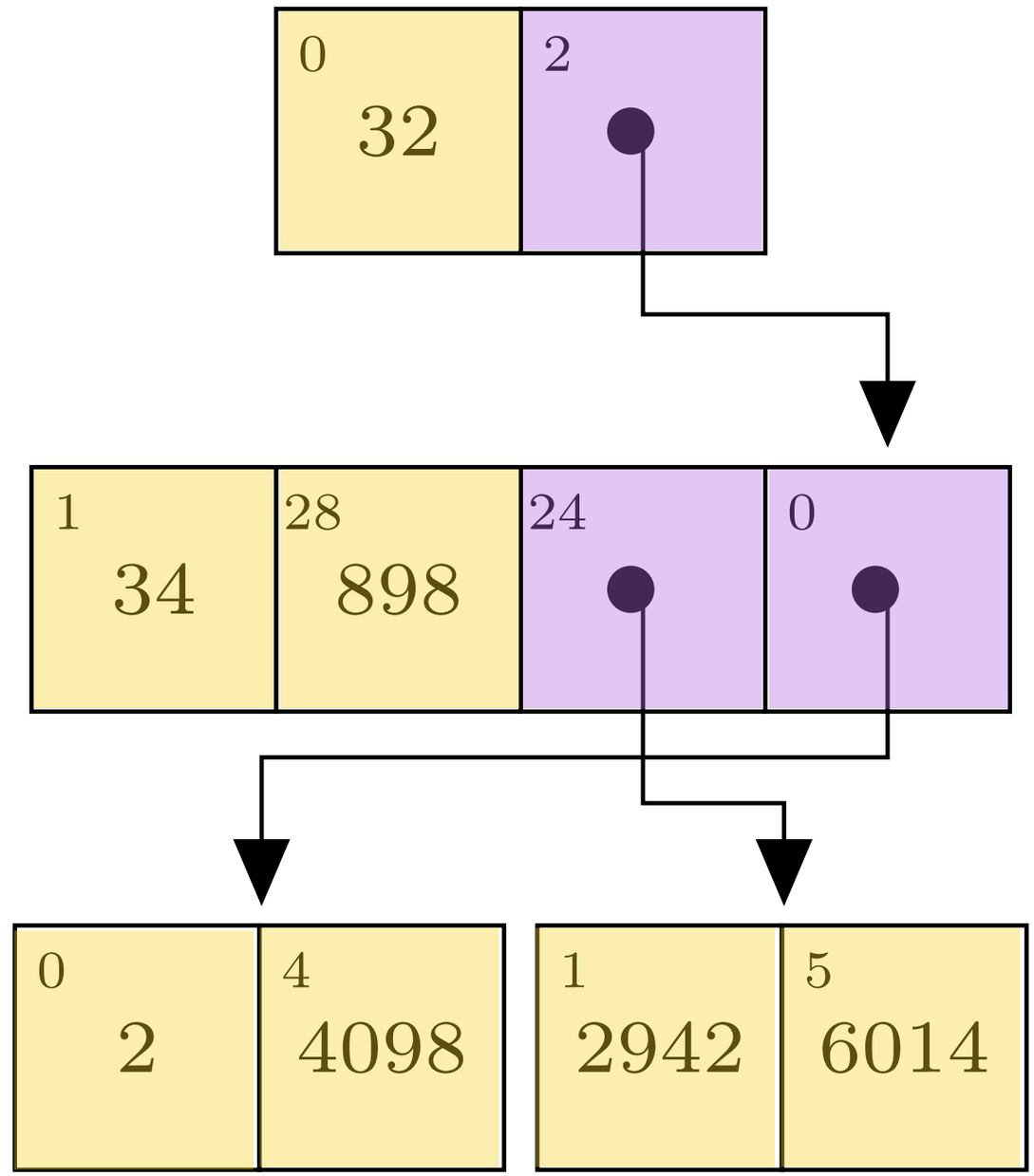
(memory footprint reduction over Clojure / Scala)

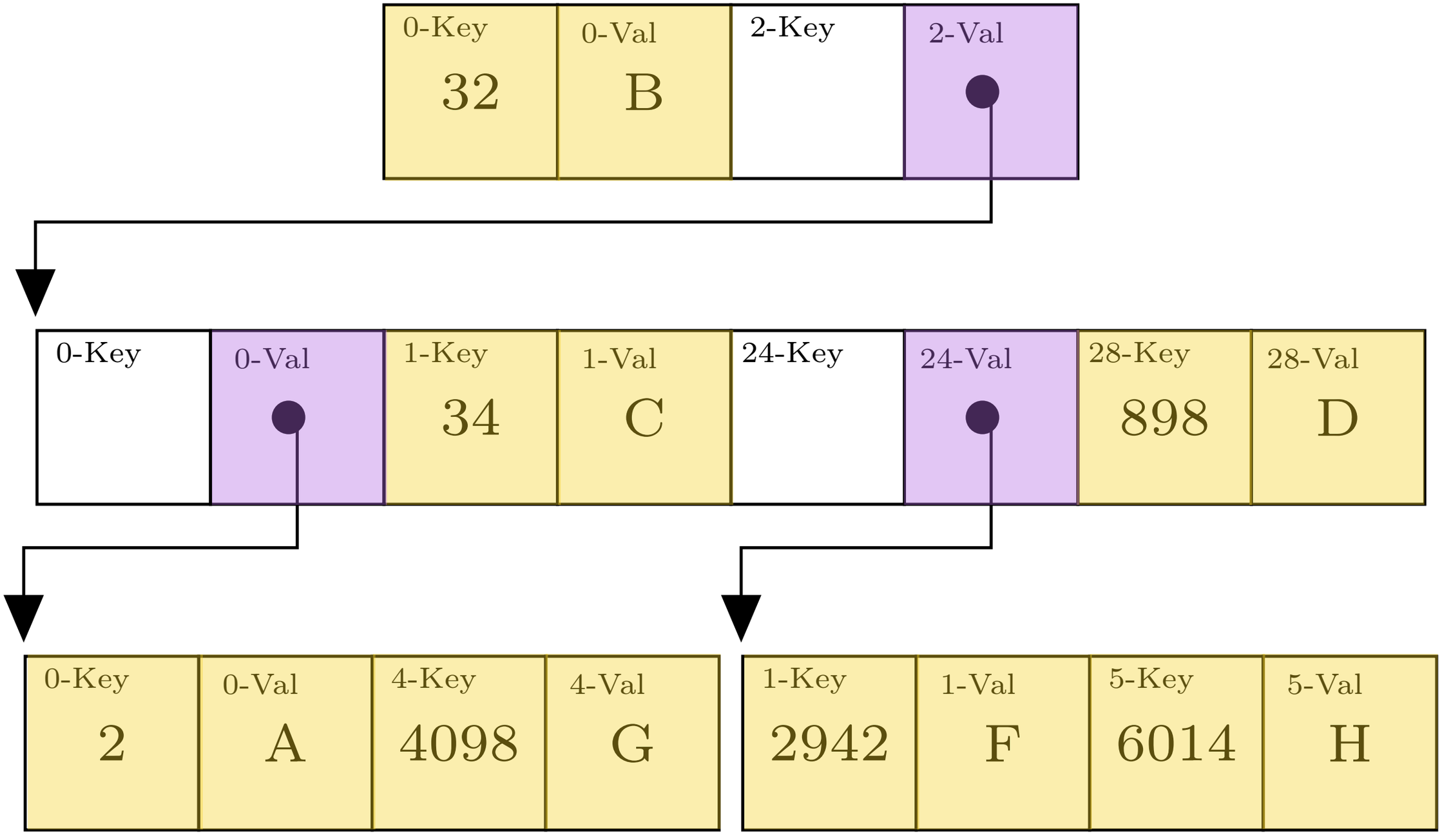
Steindorfer, M. J., & Vinju, J. J. (2015). Optimizing Hash-Array Mapped Tries for Fast and Lean Immutable JVM Collections. OOPSLA'15.

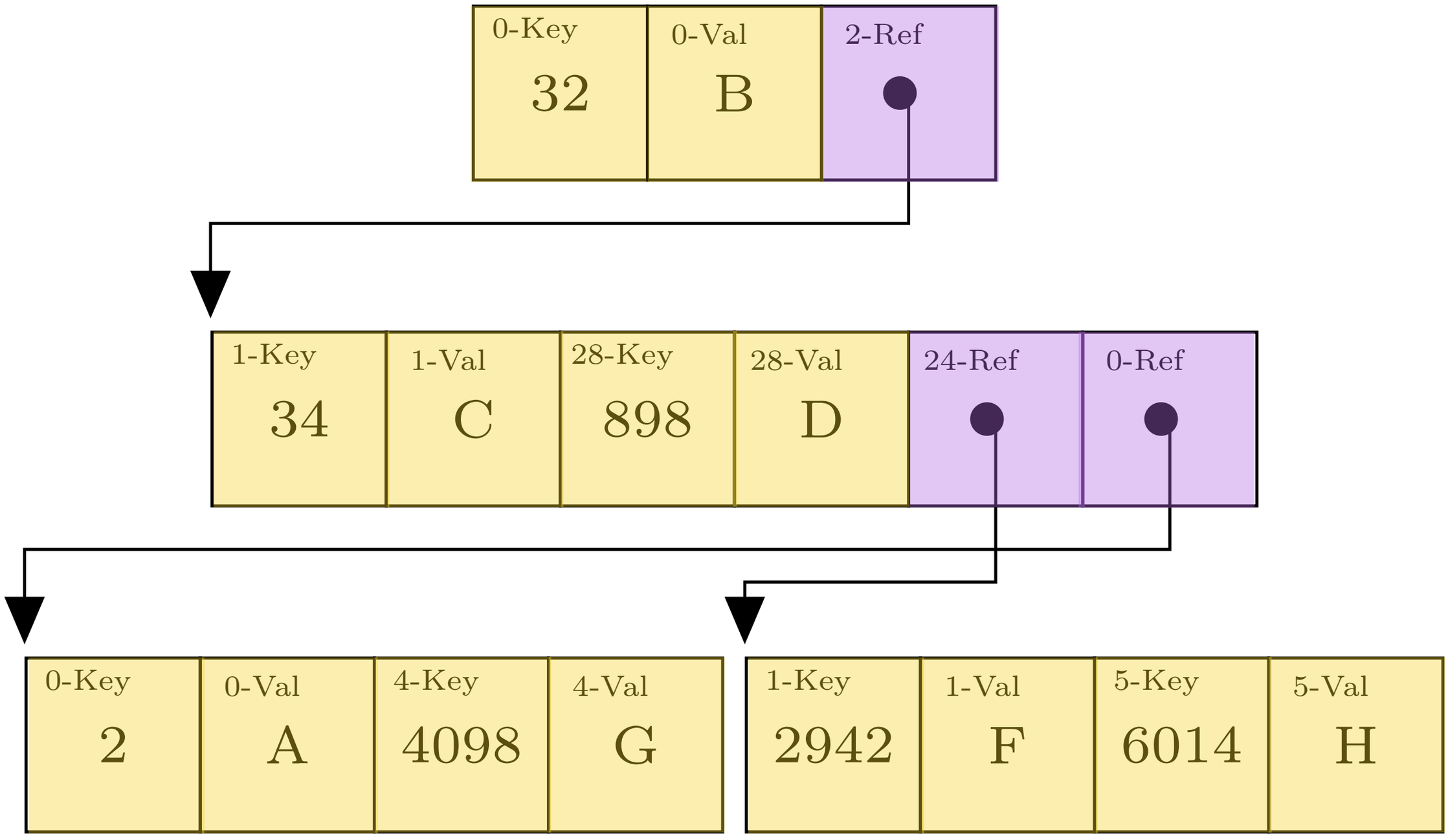

0
32
2
0
34
1
24
898
28
0
2
4
4098
1
2942
5
6014


0
32
2
0
2
0
34
24
898
28
0
4
2
4098
1
2942
5
6014
1
34
28
898
24
0
4
2
4098
1
2942
5
6014

```
class Node {
   int datamap;
   int nodemap;
   Object[] content;
}
```




0-Key
32
0-Val
B
2-Key
2-Val
0-Key
0-Val
1-Key
34
1-Val
C
24-Key
24-Val
28-Key
898
28-Val
D
0-Key
2
0-Val
A
4-Key
4098
4-Val
G
1-Key
2942
1-Val
F
5-Key
6014
5-Val
H

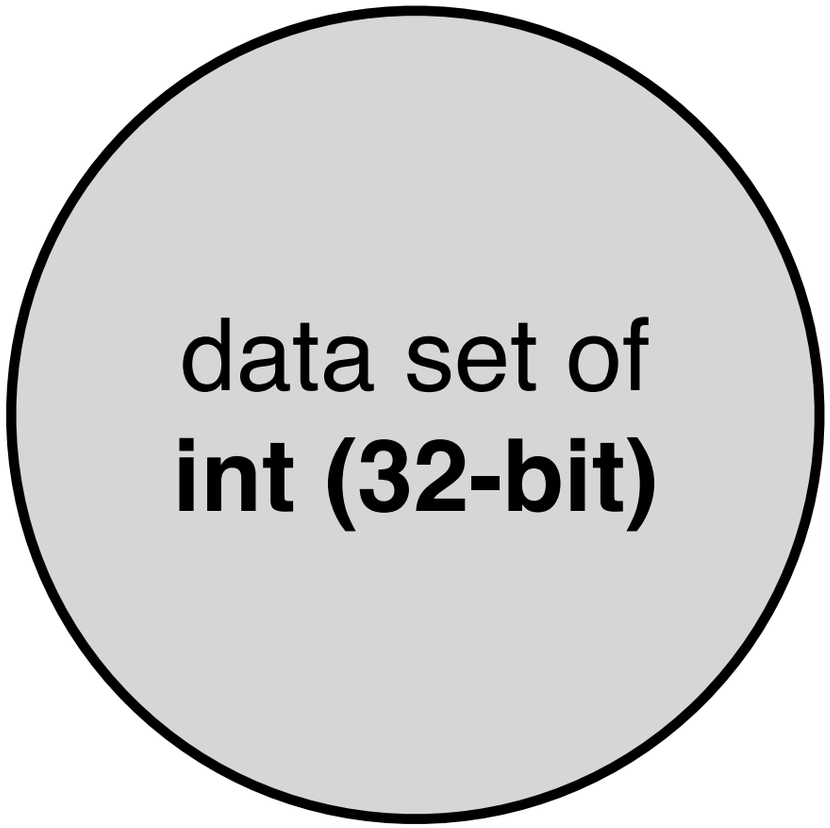

0-Key
32
0-Val
B
2-Ref
1-Key
34
1-Val
C
28-Key
898
28-Val
D
24-Ref
0-Ref
0-Key
2
0-Val
A
4-Key
4098
4-Val
G
1-Key
2942
1-Val
F
5-Key
6014
5-Val
H

More Expressive …

Numeric Data Sets

Storing and Processing with Collections

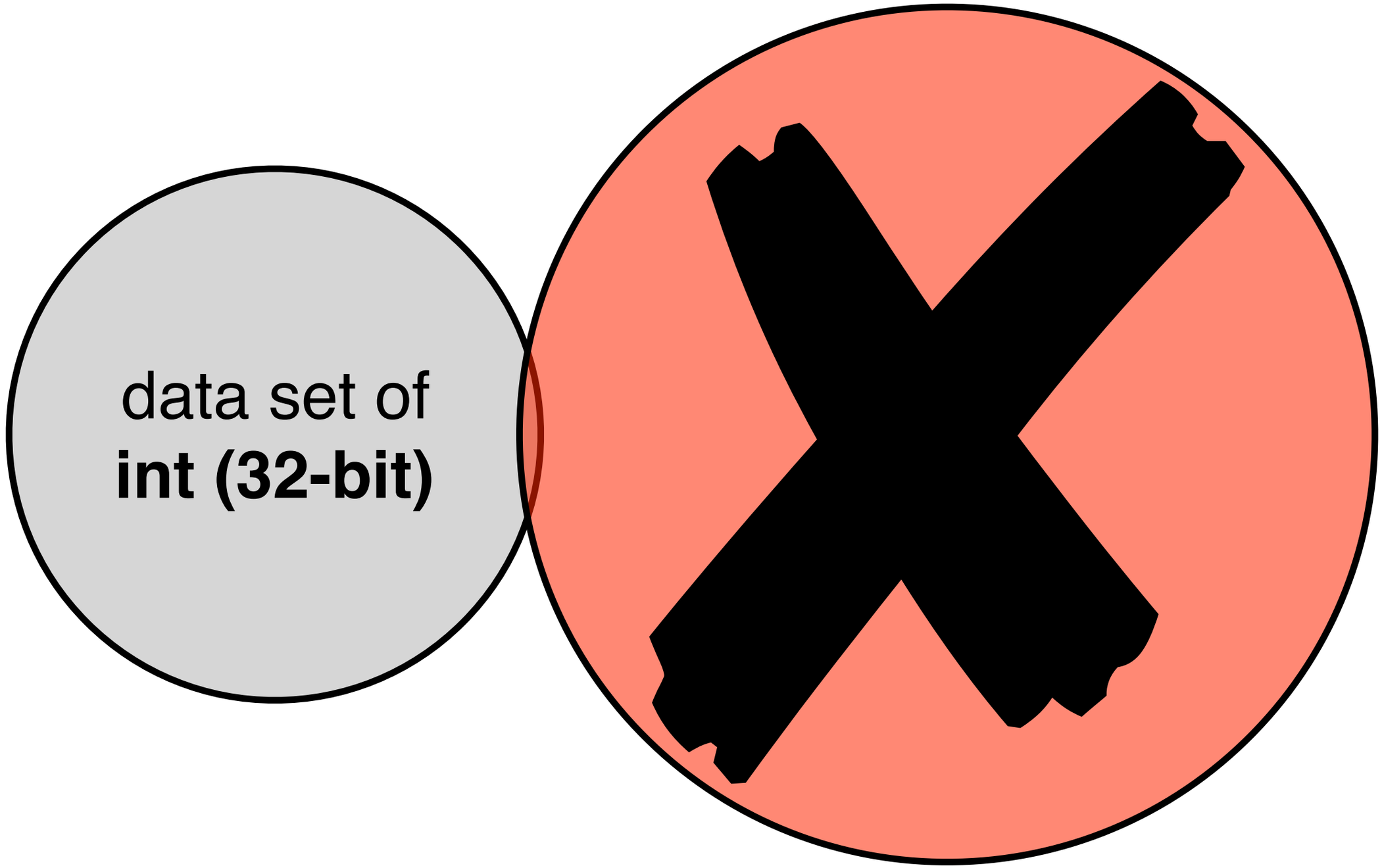
SPECIALIZED PRIMITIVE COLLECTION



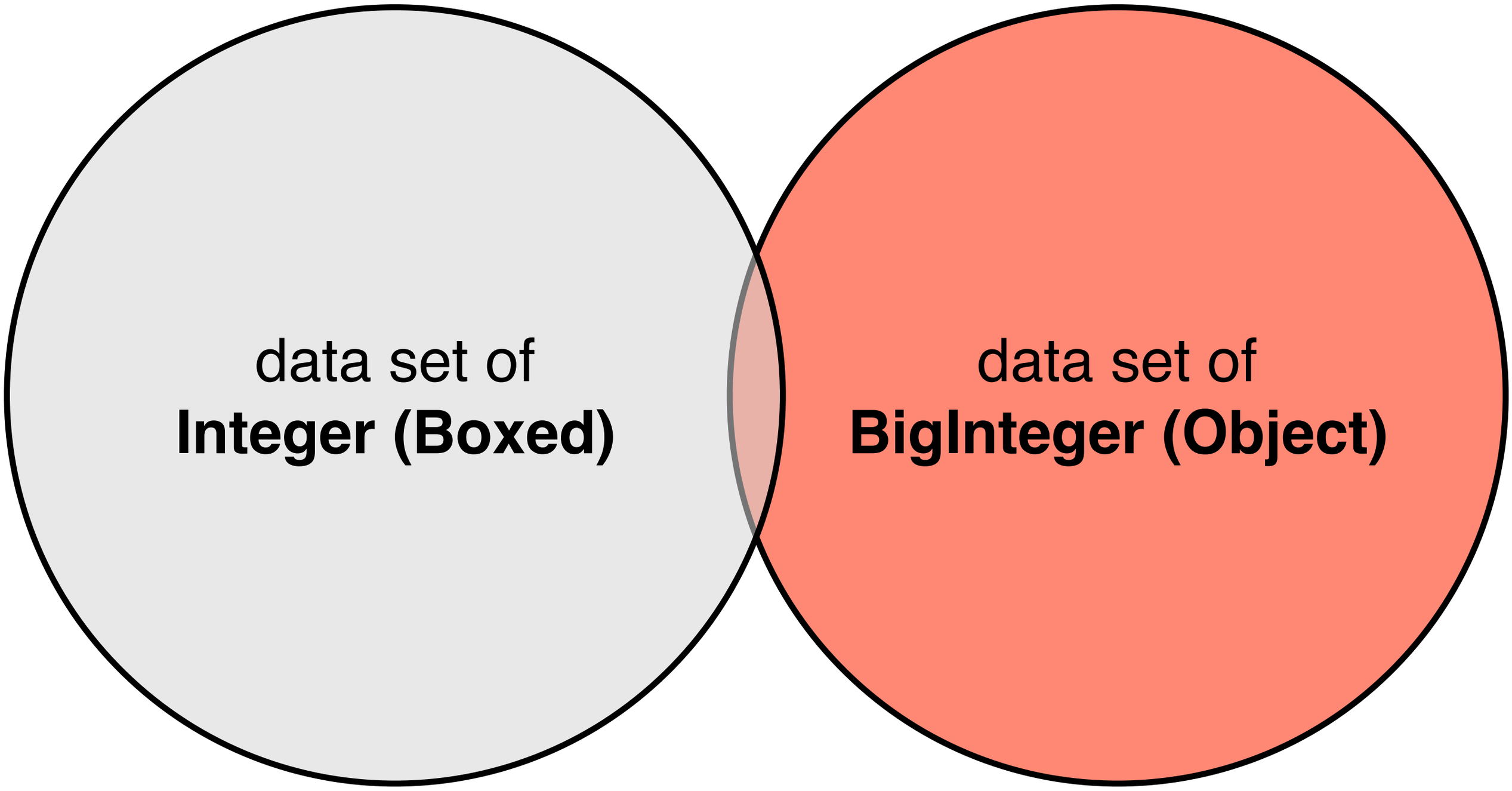
data set of
BigInteger (Object)

GENERIC COLLECTION

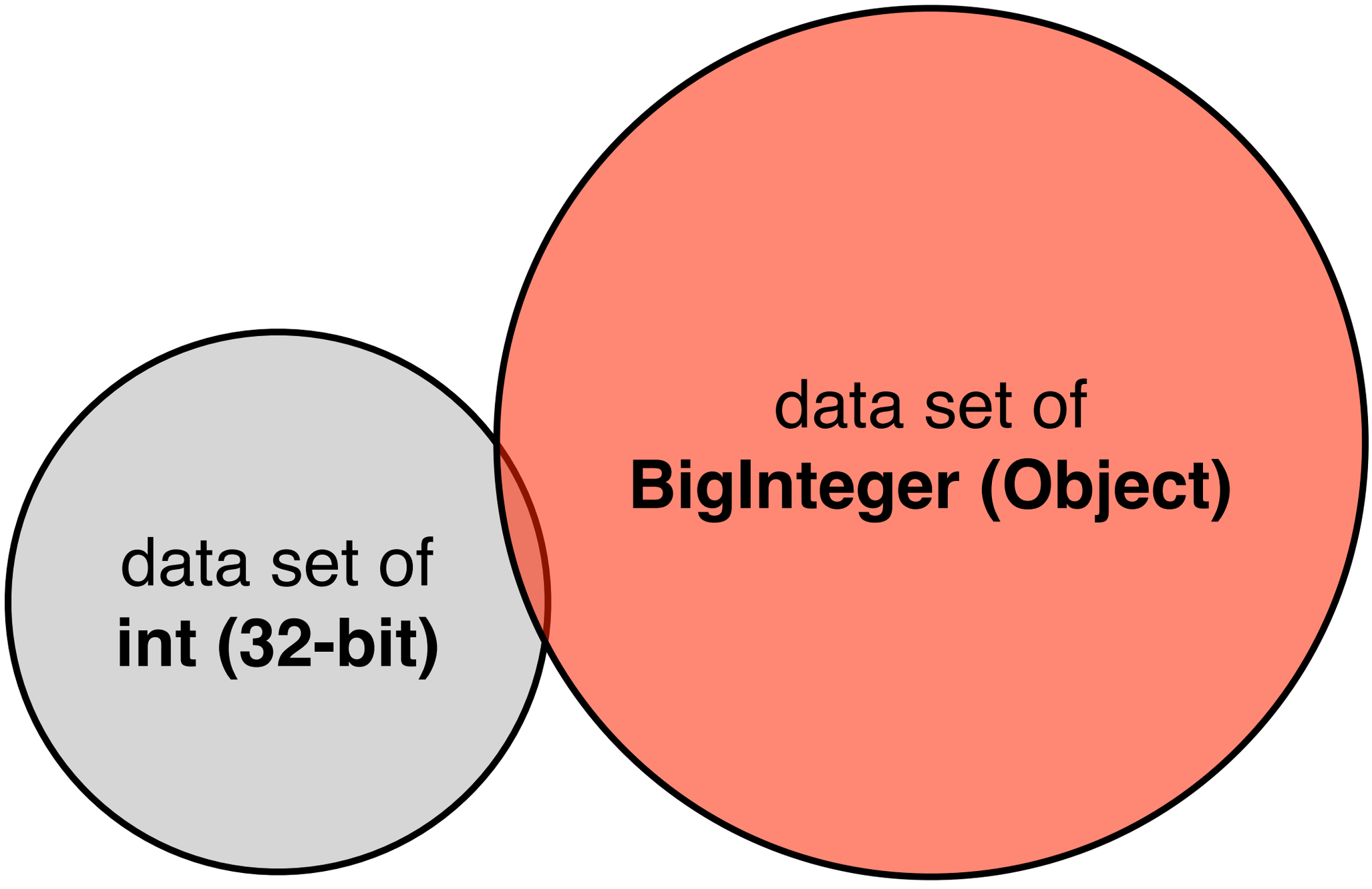
PRIMITIVE COLLECTION ???

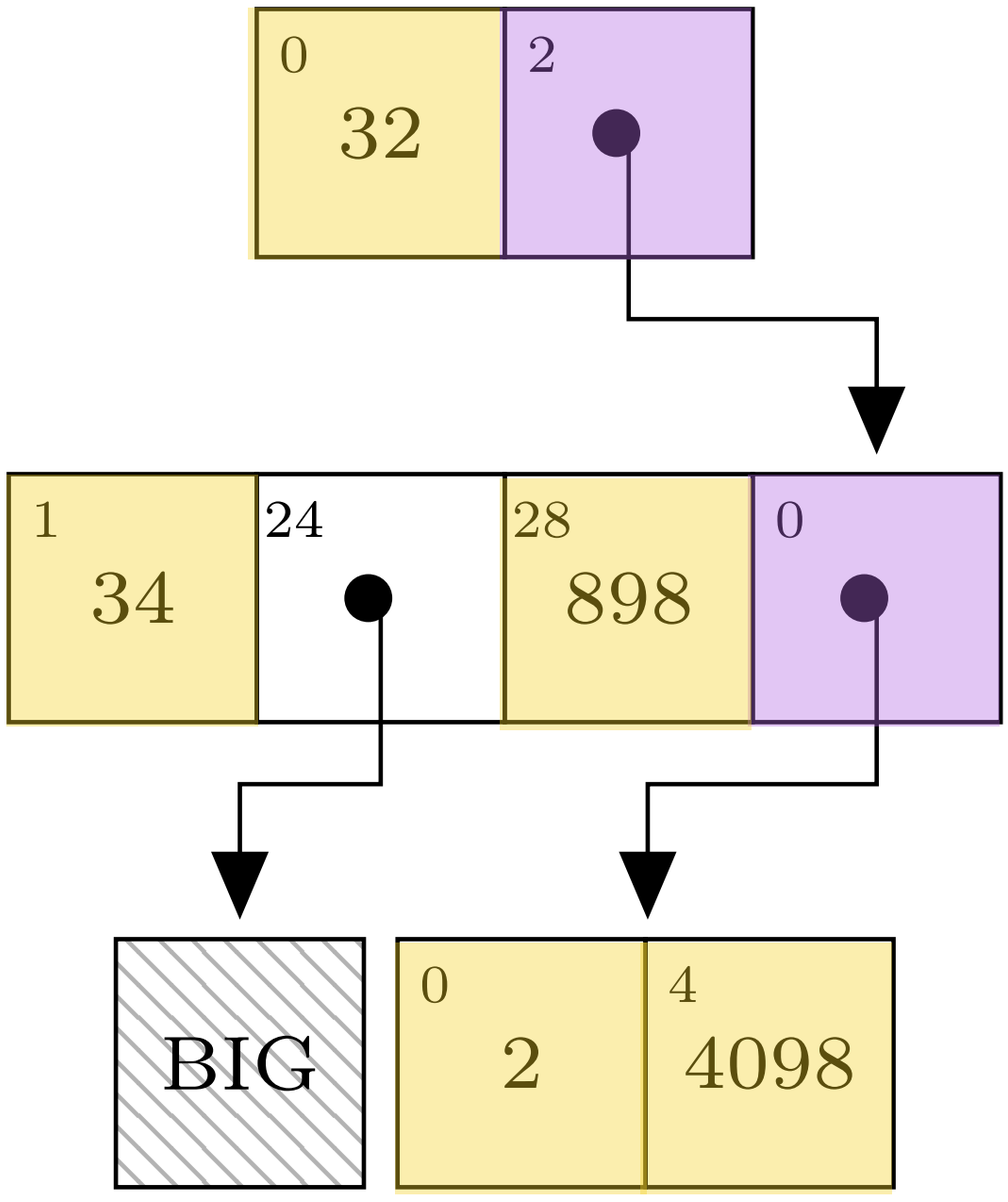


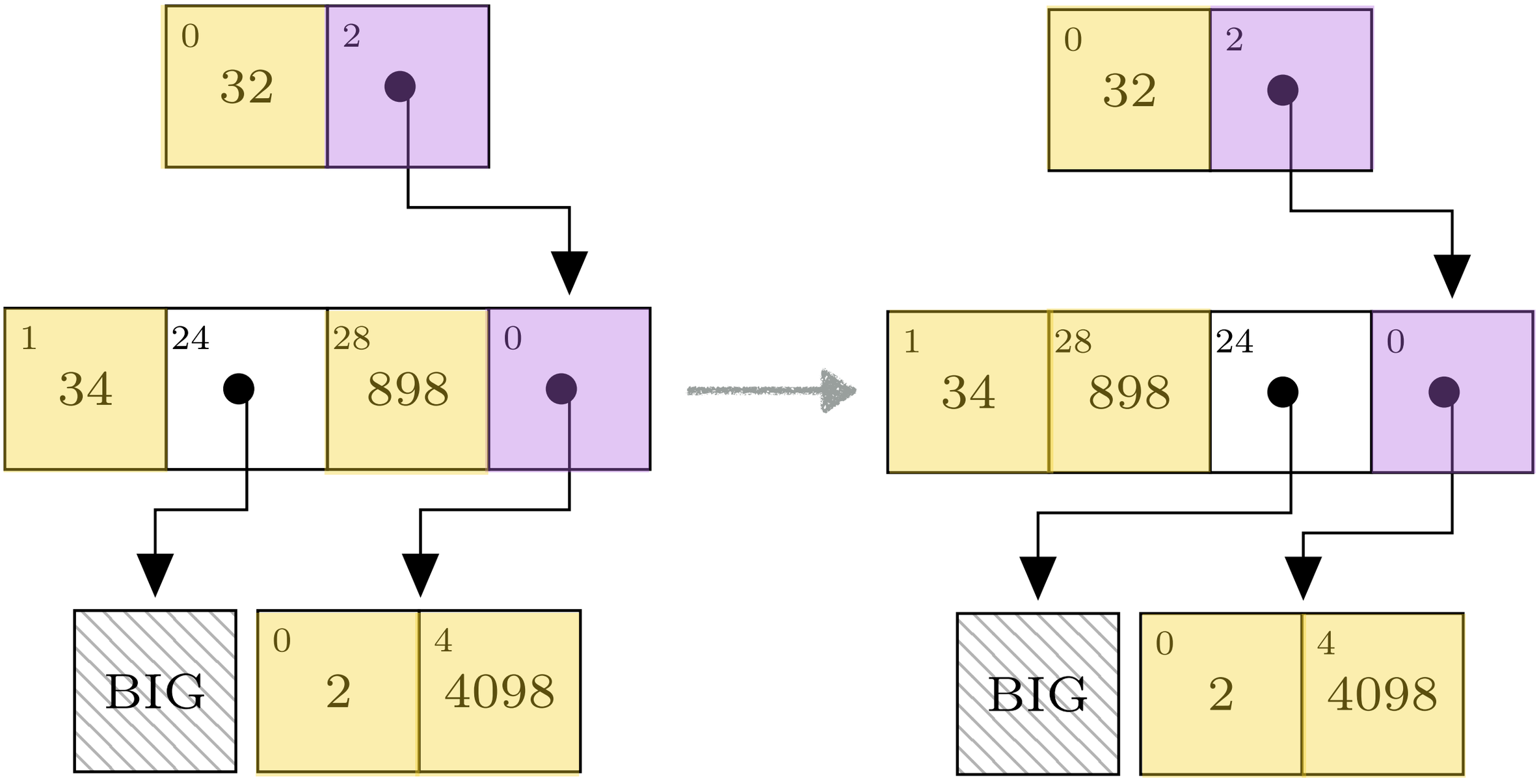
GENERIC COLLECTION ???



HETEROGENEOUS COLLECTIONS

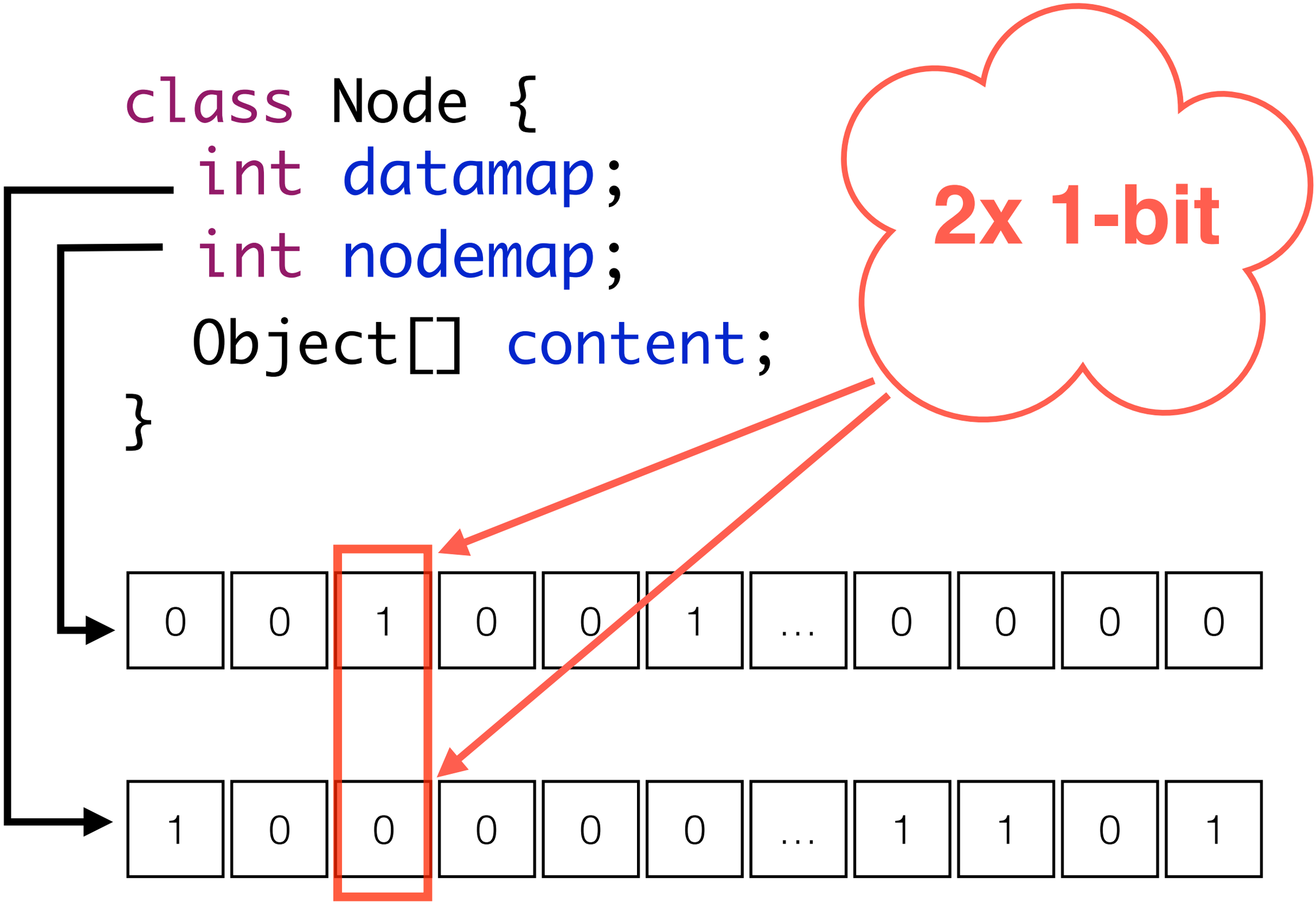



0
32
2
1
34
24
28
898
0
BIG
0
2
4
4098



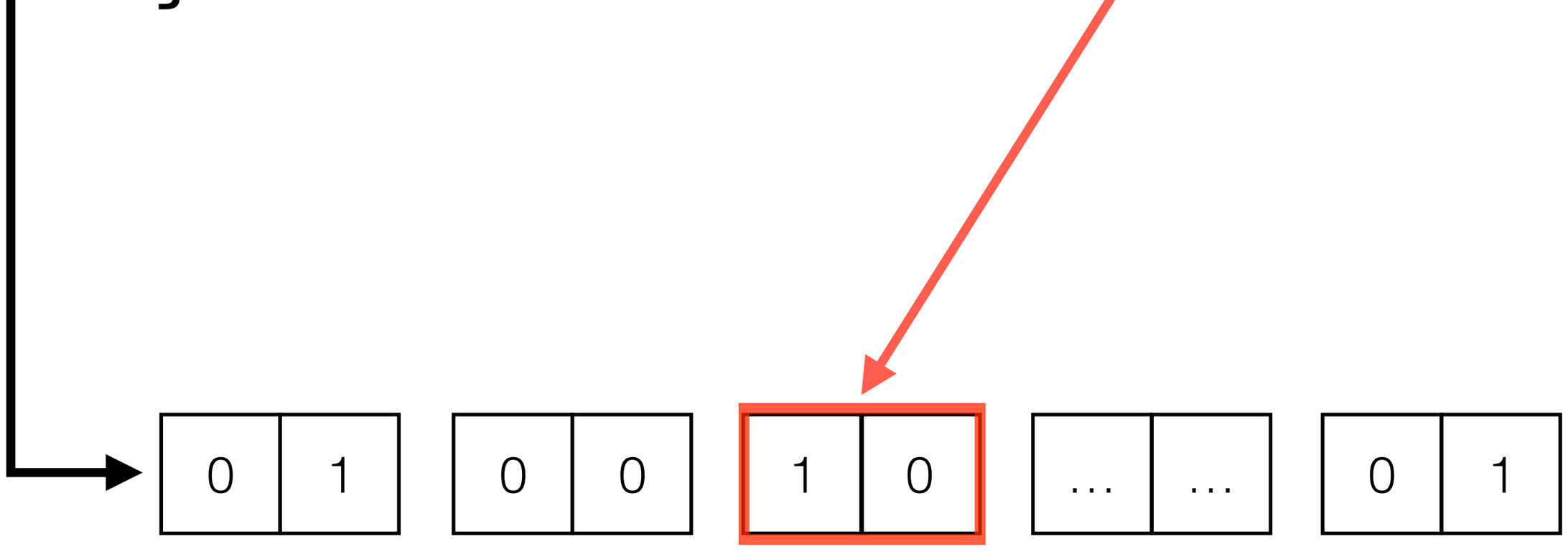
Steindorfer, M. J., & Vinju, J. J. (2016). Fast and Lean Immutable Multi-Maps on the JVM based on Heterogeneous Hash-Array Mapped Tries. To Appear.

```
class Node {
  int datamap;
  int nodemap;
  Object[] content;
}
```



```
class Node {
  BitVector bitmap;

  Object[] content;
}
```

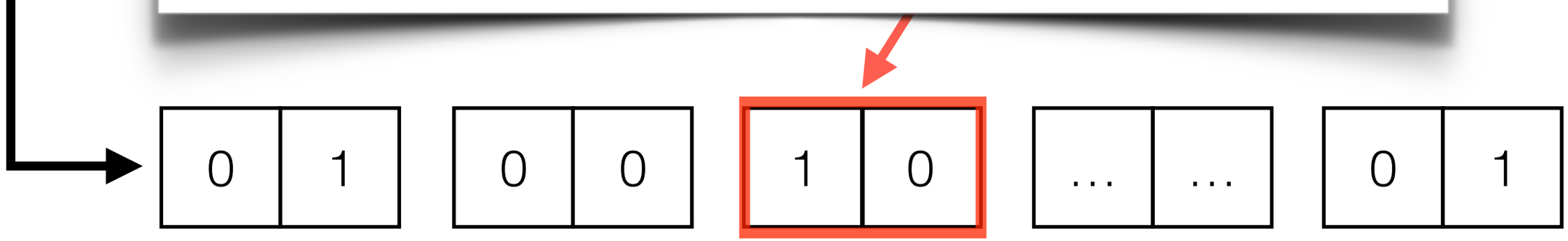
1x n-bit



```
class Node {
    BitVector bitmap;
```

Orthogonal To:

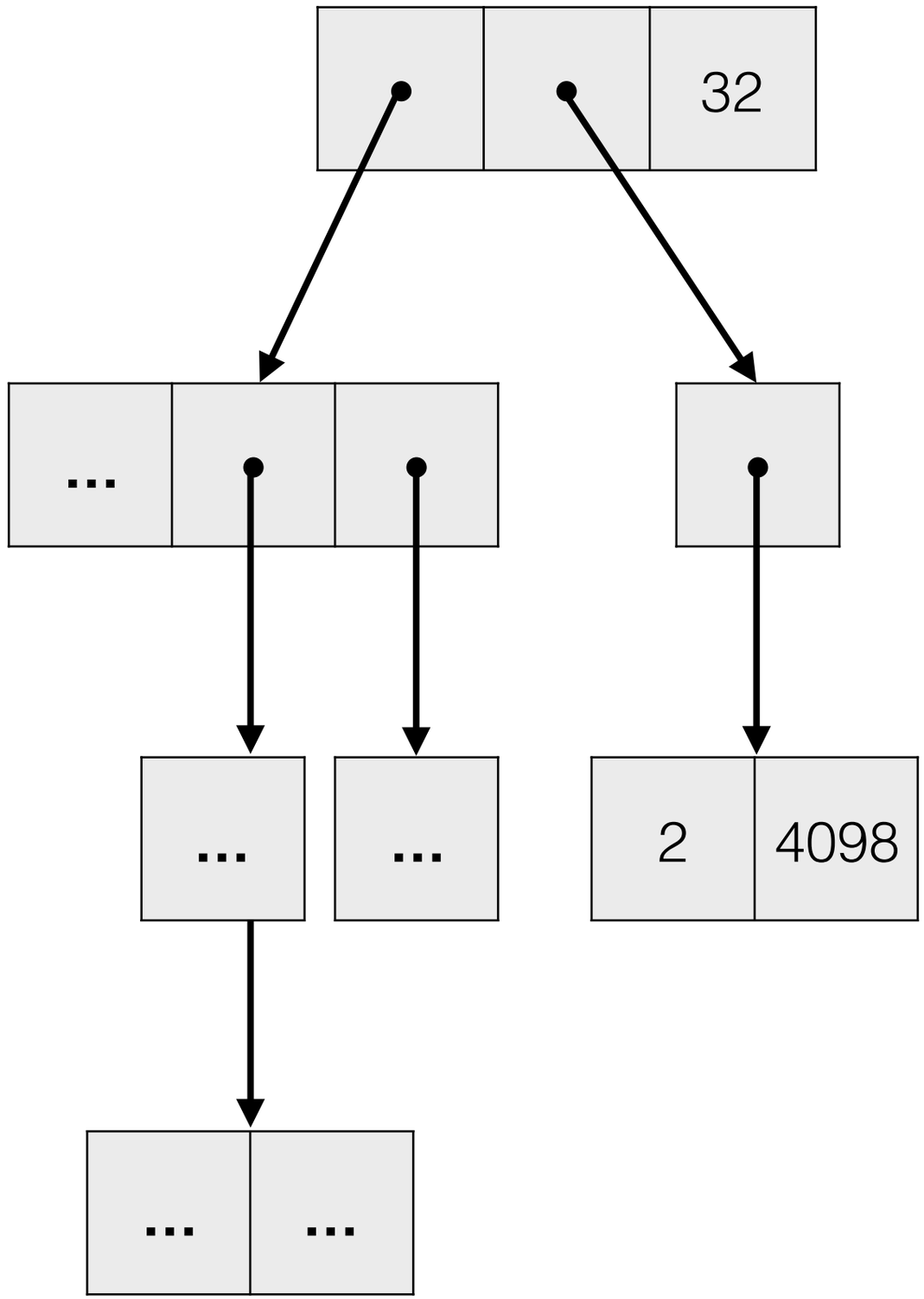
- Scala's Union Types
- Valhalla's Primitive Generics

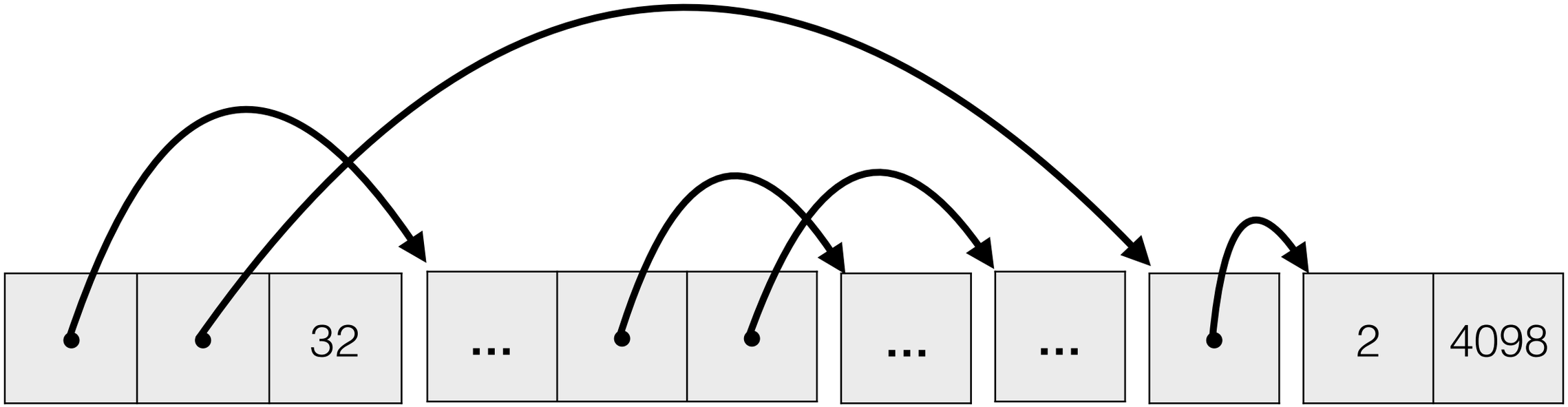


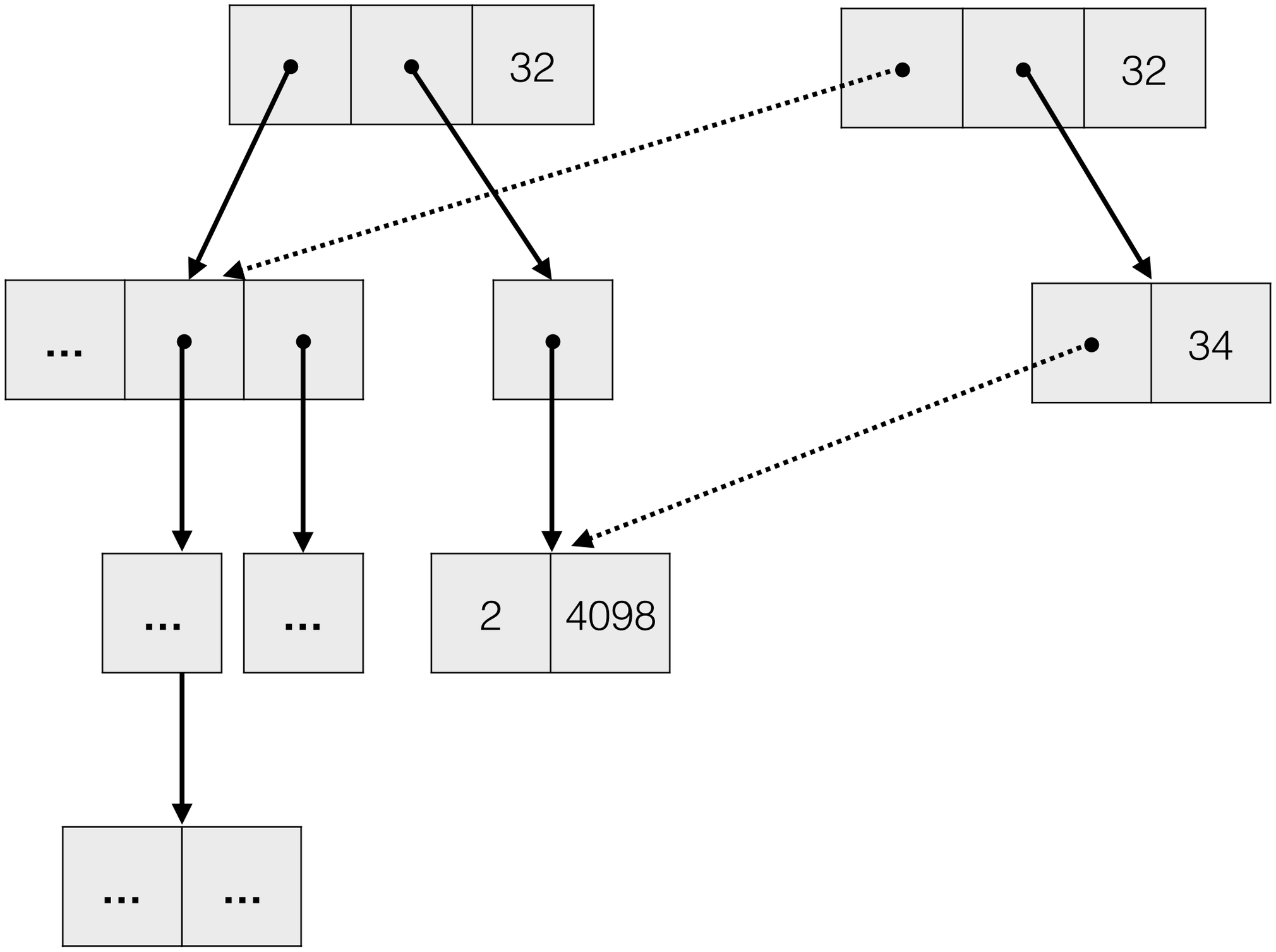
Performance Challenges

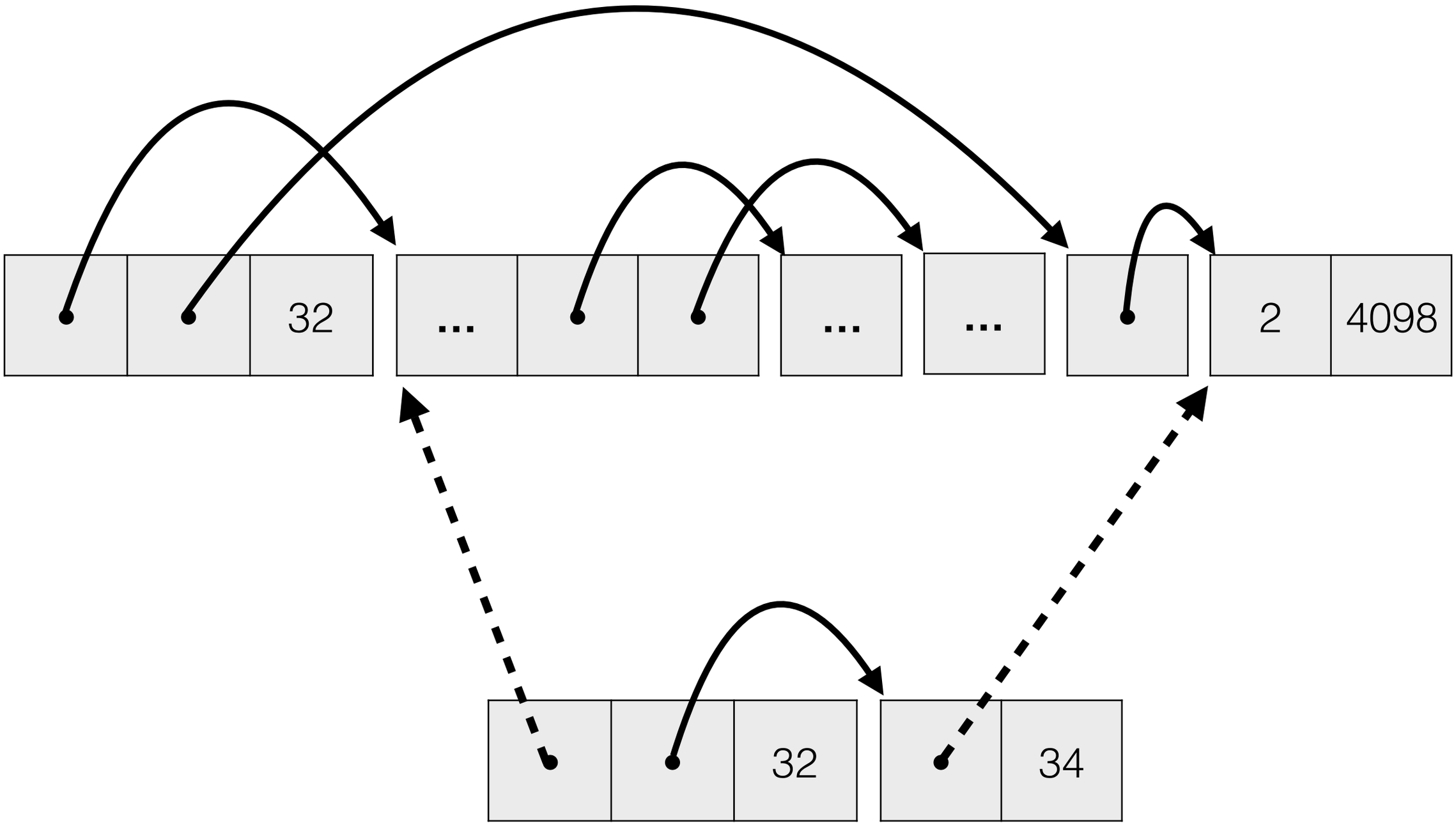
Trees (as fast) as Arrays?

Improving Locality between Nodes.


32
...
...
...
2
4098
...
...


32
...
...
...
2
4098


32
32
...
34
...
...
2
4098
...
...


32
...
...
...
2
4098
32
34

Trees (as fast) as Arrays!

Summary

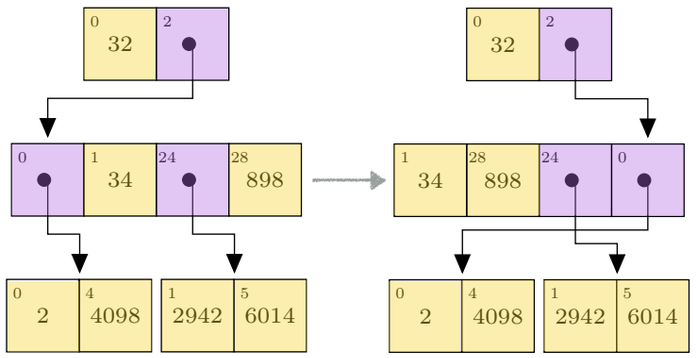
Clojure
http://clojure.org

Rascal
http://rascal-mpl.org

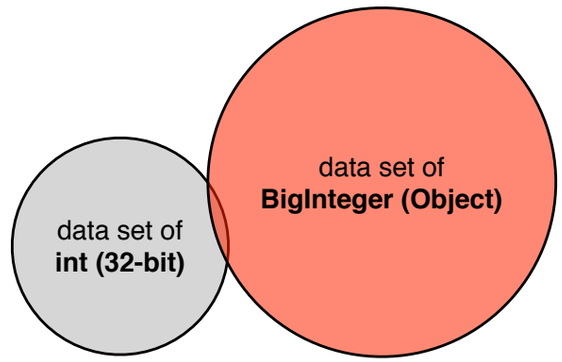
Scala
http://scala-lang.org

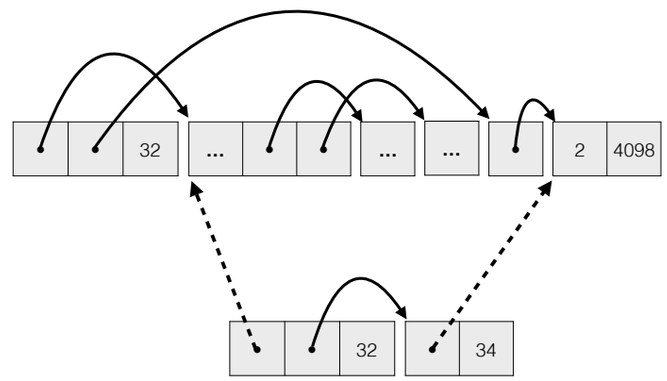

```
class ArrayList32 {
  Object[] content;
}
```

```
class List1 implements
    ImmutableList {
  Object slot0;
}

class List2 implements
    ImmutableList {
  Object slot0;
  Object slot1;
}

class List3 implements
    ImmutableList {
  Object slot0;
  Object slot1;
  Object slot2;
}

...
```

HETEROGENEOUS COLLECTIONS





```
Map<String, Long> wordCount = phrases.stream()
    .flatMap (toWordStream)
    .filter  (word -> word.length() > 0)
    .map     (word -> new Tuple<>(word, 1L))
    .collect (groupingBy(Tuple::getKey, counting()));
```

MISSING: IMMUTABLE COLLECTIONS

```
Map<String, Long> wordCount = phrases.stream()
    .flatMap (toWordStream)
    .filter  (word -> word.length() > 0)
    .map     (word -> new Tuple<>(word, 1L))
    .collect (groupingBy(Tuple::getKey, counting()));
```

usethesource/capsule

The Capsule Hash Trie Collections Library

Immutable Collections

Michael J. Steindorfer

mail: michael@cwi.nl

twitter: @loopingoptimism