Introduction to Clojure Concurrency (and data structures)

Karl Krukow,
Engineer at Trifork &
CTO LessPainful
@karlkrukow

Goto Amsterdam, May 2012

Introduction to Clojure Concurrency (and data structures)

Karl Krukow,
Engineer at Trifork &
CTO LessPainful
@karlkrukow

Goto Amsterdam, May 2012

Intro to Clojure data structures (with some parallelism added too)

Karl Krukow,
Engineer at Trifork &
CTO LessPainful
@karlkrukow

Goto Amsterdam, May 2012

About me

- PhD Computer Science, Uni. Aarhus, 2006
- Engineer at Trifork for about 6 years on Web, JavaScript, Java/JEE, Ruby/Rails, iOS, Conferences and Training.
- Clojure and Rich Hickey fan-boy!
  (and Keynote newbie)
- Recently CTO of LessPainful, automated mobile app testing: (http://www.lesspainful.com).
- Conj Labs - 3 day Clojure training in Europe
  (with Lau Jensen)

About me

- PhD Computer Science, Uni. Aarhus, 2006
- Engineer at Trifork for about 6 years on Web, JavaScript, Java/JEE, Ruby/Rails, iOS, Confer ming.
- Clojure and Rich Hickey fan-bo
  (and Keynote newbie)
- Recently CTO of LessPainful, automa mobile app testing: (http://www.lesspainful.com).
- Conj Labs - 3 day Clojure training in Europe
  (with Lau Jensen)

Why give a talk about Data Structures?

Pop-quiz

Pop-quiz

```
//m is a shared java.util.HashMap

public static void write(final int offset) {
    for (int i = 0; i < 10000; i++) {
        int k = offset+i;
        m.put(k, -k);
    }
}

public static void read(final int offset) {
     for (int i = 0; i < 10000; i++) {
        int key = offset+i;
          Integer val = m.get(key);
          if (val != null) {
            if (val.intValue() != -key) {
                System.out.println("Key and value don't match...");
            }
          }
     }
}
```

Pop-quiz

```
//m is a shared java.util.HashMap

public static void write(final int offset) {
    for (int i = 0; i < 10000; i++) {
        int k = offset+i;
        m.put(k, -k);
    }
}

public static void read(final int offset) {
     for (int i = 0; i < 10000; i++) {
        int key = offset+i;
          Integer val = m.get(key);
          if (val != null) {
            if (val.intValue() != -key) {
                System.out.println("Key and value don't match...");
            }
          }
     }
}
```

Suppose we have multiple threads calling both read and write without synchronization.

Now, what can happen?

Non-obvious bug 1

- ArrayIndexOutOfBoundsException?

```
krukow:~/workspaces/trifork/concurrency$ java -cp bin
hashmap.HashMapDemo
Exception in thread "Thread-0"

java.lang.ArrayIndexOutOfBoundsException: 23
   at java.util.HashMap.get(HashMap.java:301)
   at hashmap.HashMapDemo.read(HashMapDemo.java:17)
   at hashmap.HashMapDemo$1.run(HashMapDemo.java:32)
   at java.lang.Thread.run(Thread.java:637)
WRITE done: j = 1
READ done: j = 2
READ done: j = 4
READ done: j = 6
WRITE done: j = 3
READ done: j = 8
READ done: j = 10
```

Non-obvious bug 2

- Infinite loop!

```
krukow:~/workspaces/trifork/concurrency$ java -cp bin
hashmap.HashMapDemo
READ done: j = 0
READ done: j = 2
WRITE done: j = 1
WRITE done: j = 3
…
…
READ done: j = 12
WRITE done: j = 11
READ done: j = 14
WRITE done: j = 15
WRITE done: j = 17
READ done: j = 18
READ done: j = 16
WRITE done: j = 19
^[[A
^[[B
^[[A
^[[A
^C
```

Is this in theory only?

Is this in theory only?

- Of course not!
  - I've seen the “infinite loop” issue put down a cluster of production servers!

Is this in theory only?

- Of course not!
  - I've seen the “infinite loop” issue put down a cluster of production servers!
- **Missing technical knowledge** (e.g. JMM)

Is this in theory only?

- Of course not!
  - I've seen the "infinite loop" issue put down a cluster of production servers!
- **Missing technical knowledge** (e.g. JMM)
- **Incorrect optimizations:**
  - "I really can't pay the cost of synchronization (even though I haven't measured it), and in this particular case a data-race is safe."

Is this in theory only?

- Of course not!
  - I've seen the "infinite loop" issue put down a cluster of production servers!
- **Missing technical knowledge** (e.g. JMM)
- **Incorrect optimizations:**
  - "I really can't pay the cost of synchronization (even though I haven't measured it), and in this particular case a data-race is safe."
- **Non-obvious sharing:**
  - "I thought this object wasn't shared between multiple threads."

Is this in theory only?

- Of course not!
  - I've seen the "infinite loop" issue put down a cluster of production servers!
- **Missing technical knowledge** (e.g. JMM)
- **Incorrect optimizations:**
  - "I really can't pay the cost of synchronization (even though I haven't measured it), and in this particular case a data-race is safe."
- **Non-obvious sharing:**
  - "I thought this object wasn't shared between multiple threads."
- **Design changes.**
  - In the original design this object wasn't shared.
  - But now it is, for some reason: design change, (bad) optimizations, singleton/caching.

Is this in theory only?

- Of course not!
  - I've seen the "infinite loop" issue put down a cluster of production servers!
- **Missing technical knowledge** (e.g. JMM)
- **Incorrect optimizations:**
  - "I really can't pay the cost of synchronization (even though I haven't measured it), and in this particular case a data-race is safe."
- **Non-obvious sharing:**
  - "I thought this object wasn't shared between multiple threads."
- **Design changes.**
  - In the original design this object wasn't shared.
  - But now it is, for some reason: design change, (bad) optimizations, singleton/caching.
- **Bad library/framework.**
  - The bug may not even be in your code!

Some real-life bugs...

Some real-life bugs...

- ***Unsound optimization :*** MyFaces JSF Portlet Bridge
  - Broken lazy initialization technique (+ another bug)

Some real-life bugs...

- ***Unsound optimization :*** MyFaces JSF Portlet Bridge
  - Broken lazy initialization technique (+ another bug)
- IceFaces: **bad reasoning &** ***Design changes***
  - Store a mutable object (SwfLifecycleExecutor) in a map in application scope
  - Each requests “initializes” it setting variables
  - works in 1.8.0. broken 1.8.2

Some real-life bugs...

- ***Unsound optimization :*** MyFaces JSF Portlet Bridge
  - Broken lazy initialization technique (+ another bug)
- IceFaces: **bad reasoning &** ***Design changes***
  - Store a mutable object (SwfLifecycleExecutor) in a map in application scope
  - Each requests “initializes” it setting variables
  - works in 1.8.0. broken 1.8.2
- Spring WebFlow: ***unintended sharing***
  - Storing non thread-safe object in application scope
  - https://jira.springframework.org/browse/SWF-976

Some real-life bugs...

- ***Unsound optimization :*** MyFaces JSF Portlet Bridge
  - Broken lazy initialization technique (+ another bug)
- IceFaces: **bad reasoning &** ***Design changes***
  - Store a mutable object (SwfLifecycleExecutor) in a map in application scope
  - Each requests “initializes” it setting variables
  - works in 1.8.0. broken 1.8.2
- Spring WebFlow: ***unintended sharing***
  - Storing non thread-safe object in application scope
  - https://jira.springframework.org/browse/SWF-976
- ...

Some real-life bugs...

- ***Unsound optimization :*** MyFaces JSF Portlet Bridge
  - Broken lazy initialization technique (+ another bug)
- IceFaces: **bad reasoning &** ***Design changes***
  - Store a mutable object (SwfLifecycleExecutor) in a map in application scope
  - Each requests "initializes" it setting variables
  - works in 1.8.0. broken 1.8.2
- Spring WebFlow: ***unintended sharing***
  - Storing non thread-safe ob
  - https://jira.springframework
- ...

"This bug is very strange, because in single user development/testing everything works fine, but in production with more users, we got usually NullPointerException on bizzarre places in program like this..."

Note

Note

- All these bugs have at their core
  - Share mutable objects
  - Low-level primitives: Threads, locks + JVM Memory Model

Note

- All these bugs have at their core
  - Share mutable objects
  - Low-level primitives: Threads, locks + JVM Memory Model
- Persistent data structures are Good!
  - These bugs vanish when you restrict yourself to programming with immutable objects.
  - But how to make that practical?

Note

- All these bugs have at their core
  - Share mutable objects
  - Low-level primitives: Threads, locks + JVM Memory Model
- Persistent data structures are Good!
  - These bugs vanish when you restrict yourself to programming with immutable objects.
  - But how to make that practical?
- It turns out, many immutable persistent data structures also are quite amenable to parallel processing.

Agenda

- One-slide intro to Clojure
- Clojure Persistent Data Structures
  - More details: PersistentVector, PersistentHashMap
- A look into the future: Parallelism with reducers
- Summary and references

Clojure in one slide

- Functional dynamic language
  - Persistent data structures, pure functions, sequence library
- A unique programming & concurrency model
  - State management constructs: var, ref, atom, agent
- On-the-fly & AOT compilation: JVM bytecode
  - Deep two-way JVM interop.; embraces host
- A LISP family member
  - Meta programming, closures, interactivity

Remember, there is (much) more...

Remember, there is (much) more...

- Records, types, protocols and polymorphism
- Multi methods
- Laziness
- Concurrency support
- Parallel programming support
- Macros and meta programming
- Numeric support and type hints
- JVM language interop
- ClojureScript

Remember, there is (much) more...

- Records, types, protocols and polymorphism
- Multi methods
- Laziness
- Concurrency support
- Parallel programming support
- Macros and meta programming
- Numeric support and type hints
- JVM language interop
- ClojureScript
- Sequence library
- DSLs with Clojure
- Logic programming
- Meta data
- Persistent data structures
- Interactive/REPL-based programming
- Clojure CLR
- ...

Clojure Philosophy

Clojure Philosophy

- *Background*:
  - Most programs could have dramatically less state than they do - we tend to introduce state just because it is the language default (and because we are trained to do so).

Clojure Philosophy

- *Background*:
  - Most programs could have dramatically less state than they do - we tend to introduce state just because it is the language default (and because we are trained to do so).
- In Clojure
  - We rarely use mutable objects, instead immutable data structures and pure functions.
  - Explicitly mark the (few) parts of the program that have state - *reference types*.
  - State-change to reference types
    - is managed by Clojure (no manual locking).
    - references *atomically* change from referring one immutable piece of data to another.

Understanding Clojure's Persistent Data Structures

What are persistent data structures?

What are persistent data structures?

- Nothing to do with durability, i.e., saving to storage!

What are persistent data structures?

- Nothing to do with durability, i.e., saving to storage!
- Immutable

What are persistent data structures?

- Nothing to do with durability, i.e., saving to storage!
- Immutable
- There are efficient functions that
  - take as input a PDS, and produce as output a ”variant” of the input.
  - The input is still available after the operation and will retain its performance characteristics.

What are persistent data structures?

- Nothing to do with durability, i.e., saving to storage!
- Immutable
- There are efficient functions that
  - take as input a PDS, and produce as output a ”variant” of the input.
  - The input is still available after the operation and will retain its performance characteristics.
- For Clojure, performance characteristics usually within 1-4x of their mutable (Java) counterparts (for single ops).
  - The input and output structures share most of their structure (which is efficient and safe).

That almost feels like magic

Data structures

- Lists - singly linked, grow at front
  - `(1 2 3 4 5), (fred ethel lucy), (list 1 2 3)`
- Vectors - indexed access, grow at end
  - `[1 2 3 4 5], [fred ethel lucy]`
- Maps - key/value associations
  - `{:a 1, :b 2, :c 3}, {1 “ethel” 2 “fred”}`
- Sets `#{fred ethel lucy}`
- Everything Nests

Slide by Rich Hickey

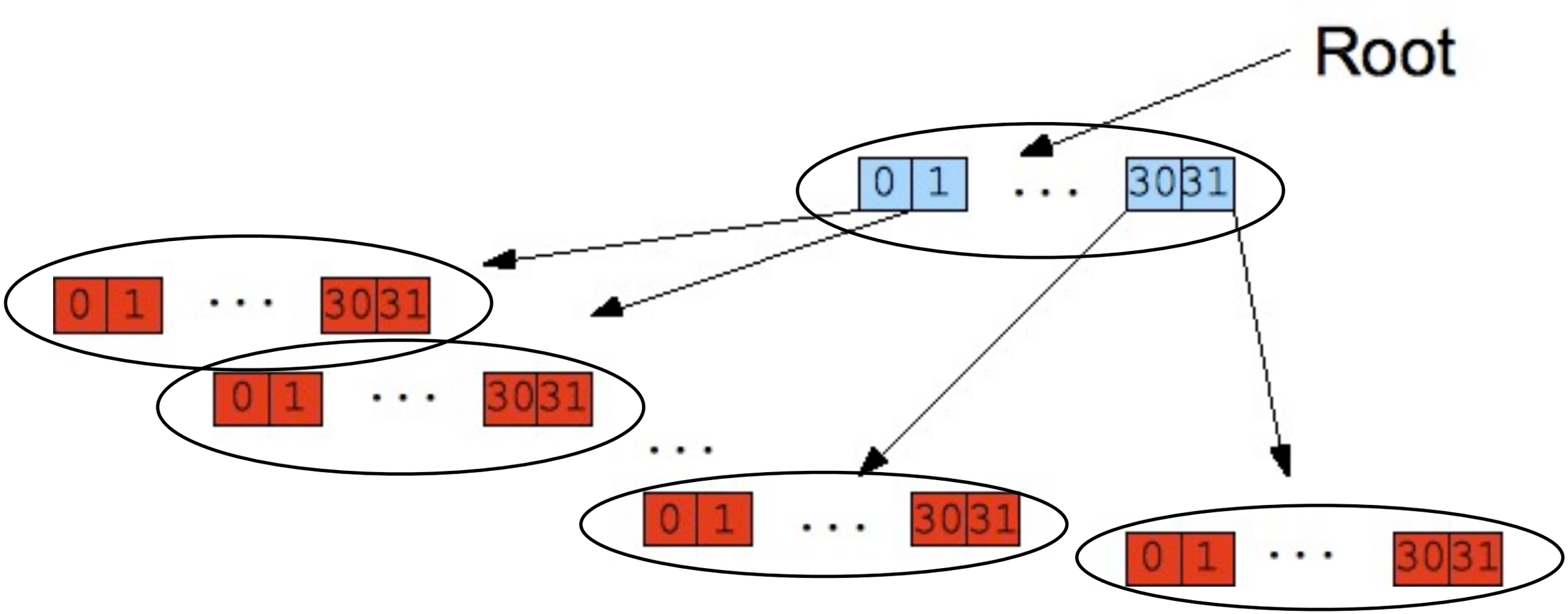
Understanding Persistent Vector

- How is a PersistentVector represented?
- How to do random access to the vector?
- How do we “add” elements to the vector?

?

```
public Object nth(int i)
{
 if(i >= 0 && i < cnt)
 {
  if(i >= tailoff())
    return tail[i & 0x01f];
  Object[] arr = root;
  for(int level = shift; level > 0; level -= 5)
    arr = (Object[]) arr[(i >>> level) & 0x01f];

  return arr[i & 0x01f];
 }
 throw new IndexOutOfBoundsException();
}
```

Implemented as wide trees



Bit partitioning

Vector index'es are ints (32 bit numbers)

Partition bit-representation in blocks of 5.

Each block corresponds to a level in the tree

(note, A block is also number in range [0,31])

Bit partitioning

Vector index'es are ints (32 bit numbers)

Partition bit-representation in blocks of 5.

Each block corresponds to a level in the tree

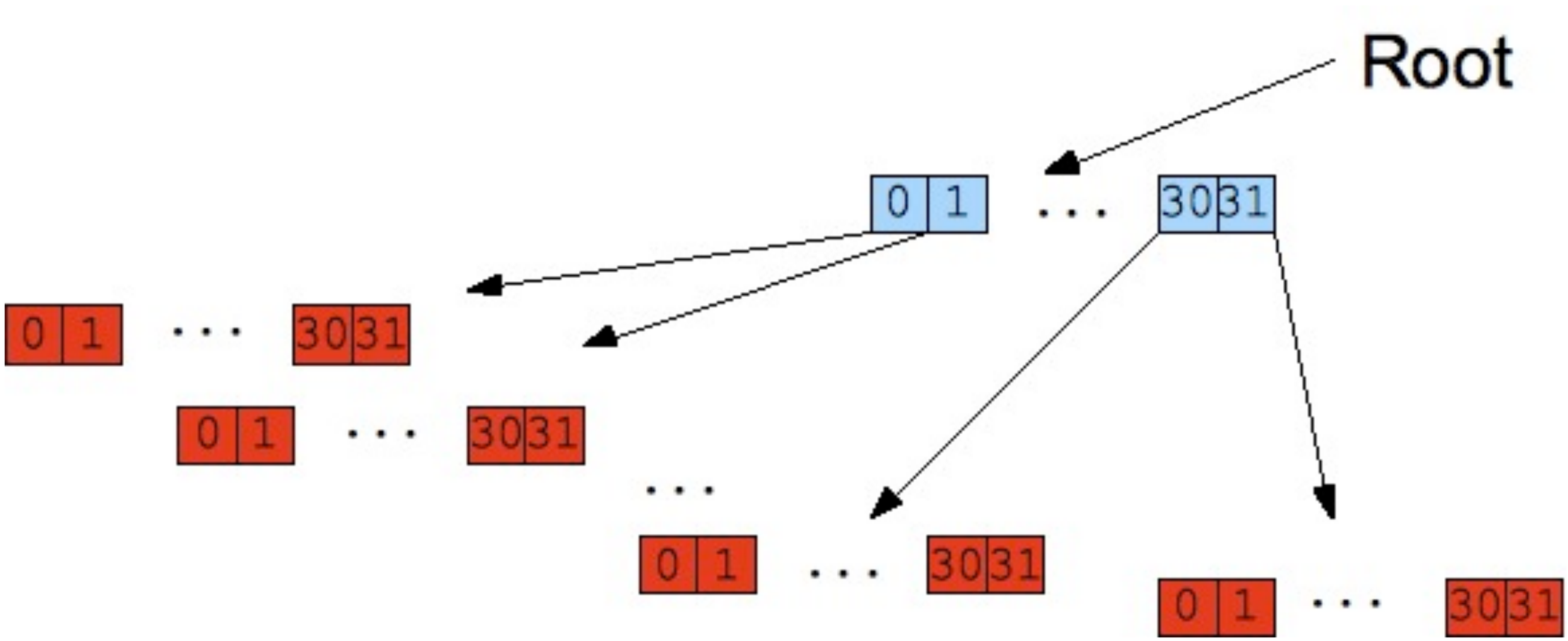
(note, A block is also number in range [0,31])

Examples

```
1:      [00][00000][00000][00000][00000][00000][00001]
234:    [00][00000][00000][00000][00000][00111][01010](10,7)
1258:   [00][00000][00000][00000][00001][00111][01010](10,7,1)
```

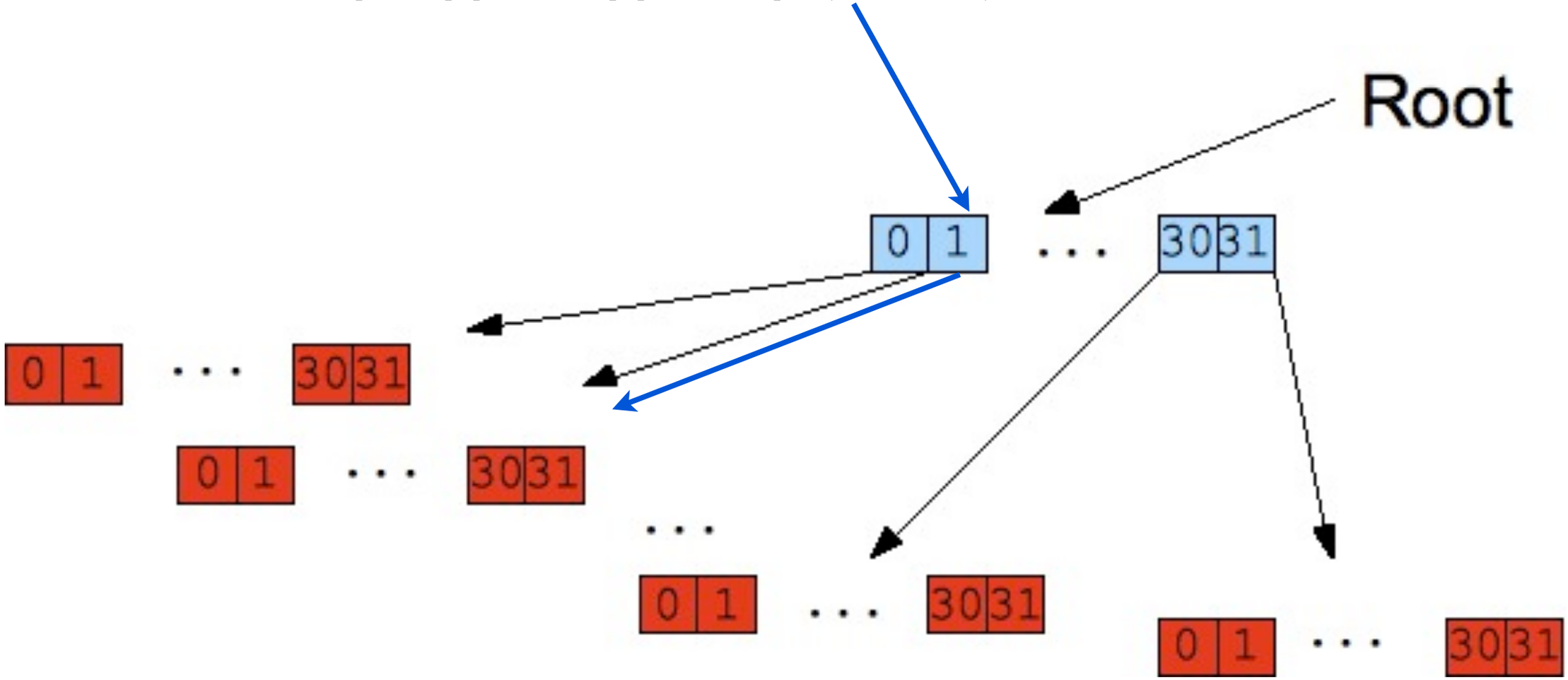
Illustration

- `49:  [...][00001][10001] (1 + 17)`



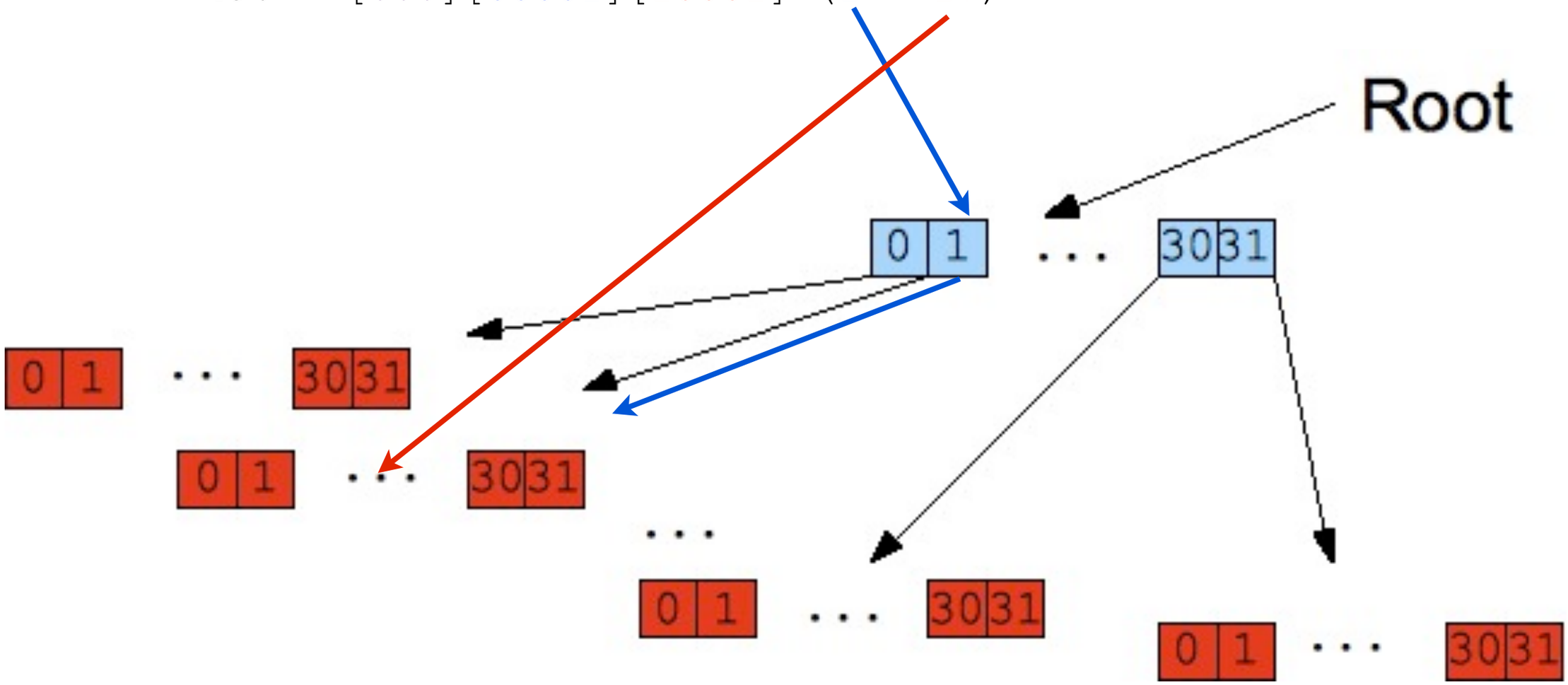
Illustration

- `49:  [...][00001][10001] (1 + 17)`



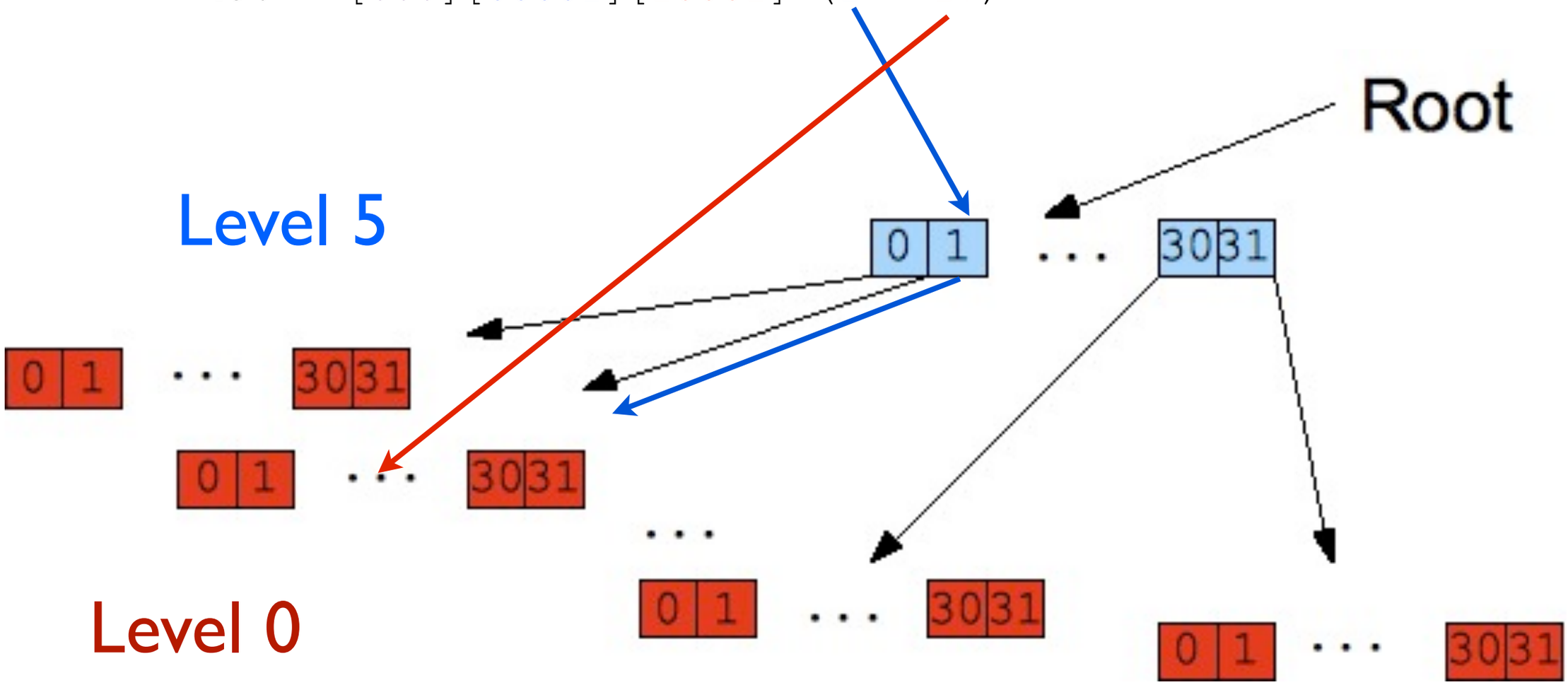
Illustration

- `49:  [...][00001][10001] (1 + 17)`



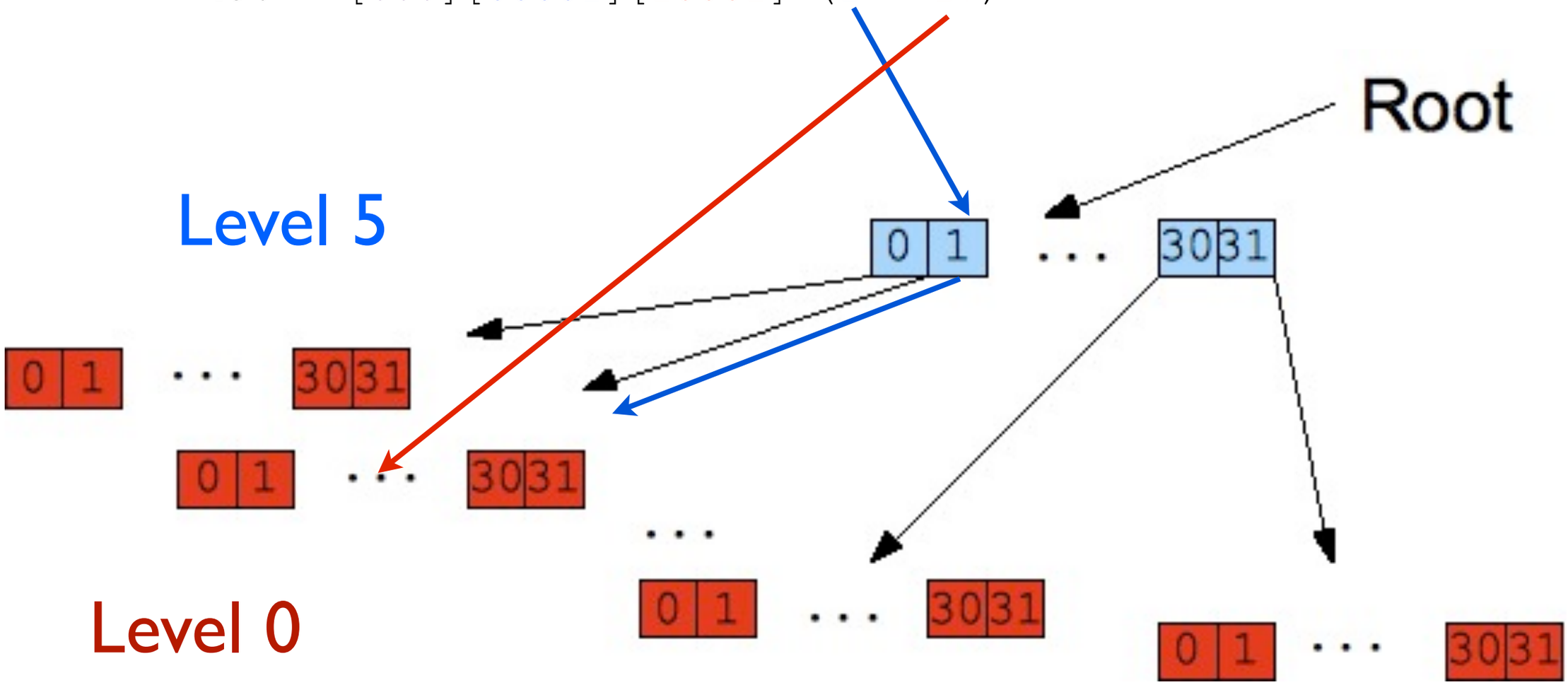
Illustration

- `49:  [...][00001][10001] (1 + 17)`



Illustration

- `49:  [...][00001][10001] (1 + 17)`



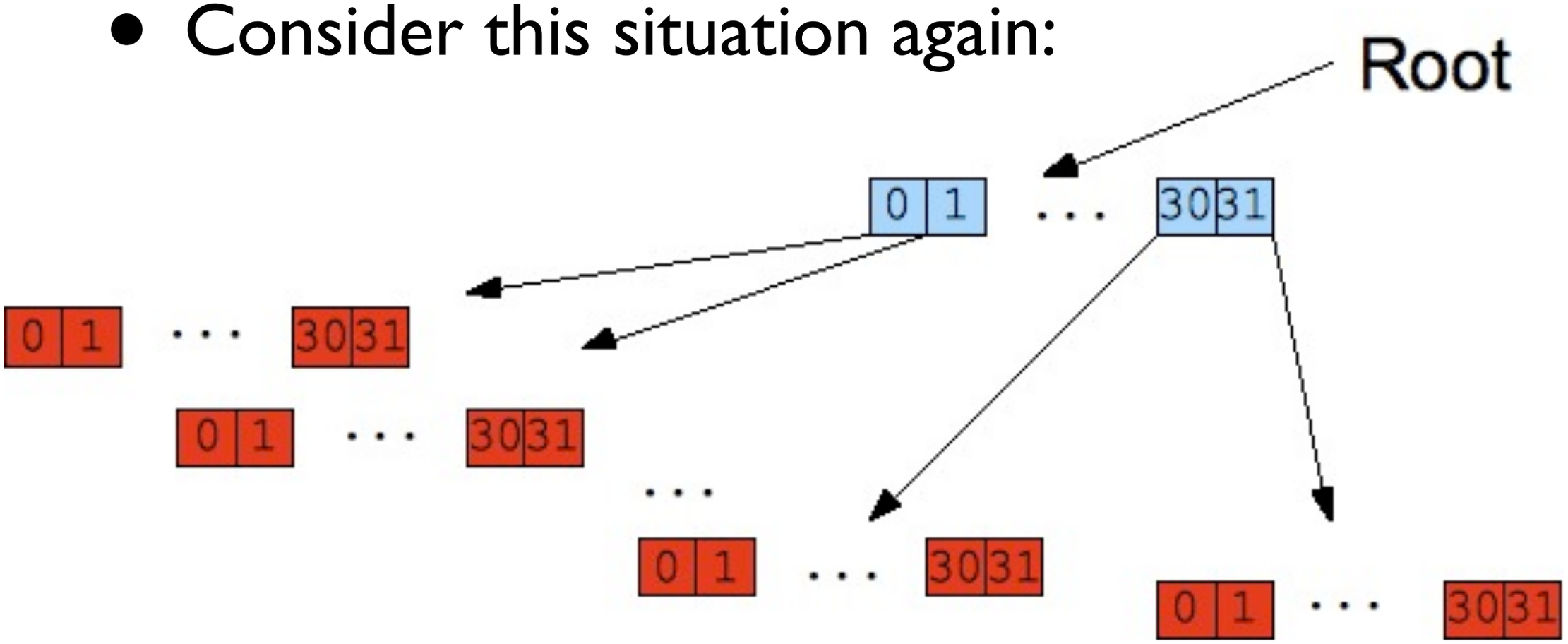
(i >>> level) & 0x01f

“Insertion” / Conjoin

- Consider this situation again:

“Insertion” / Conjoin

- Consider this situation again:



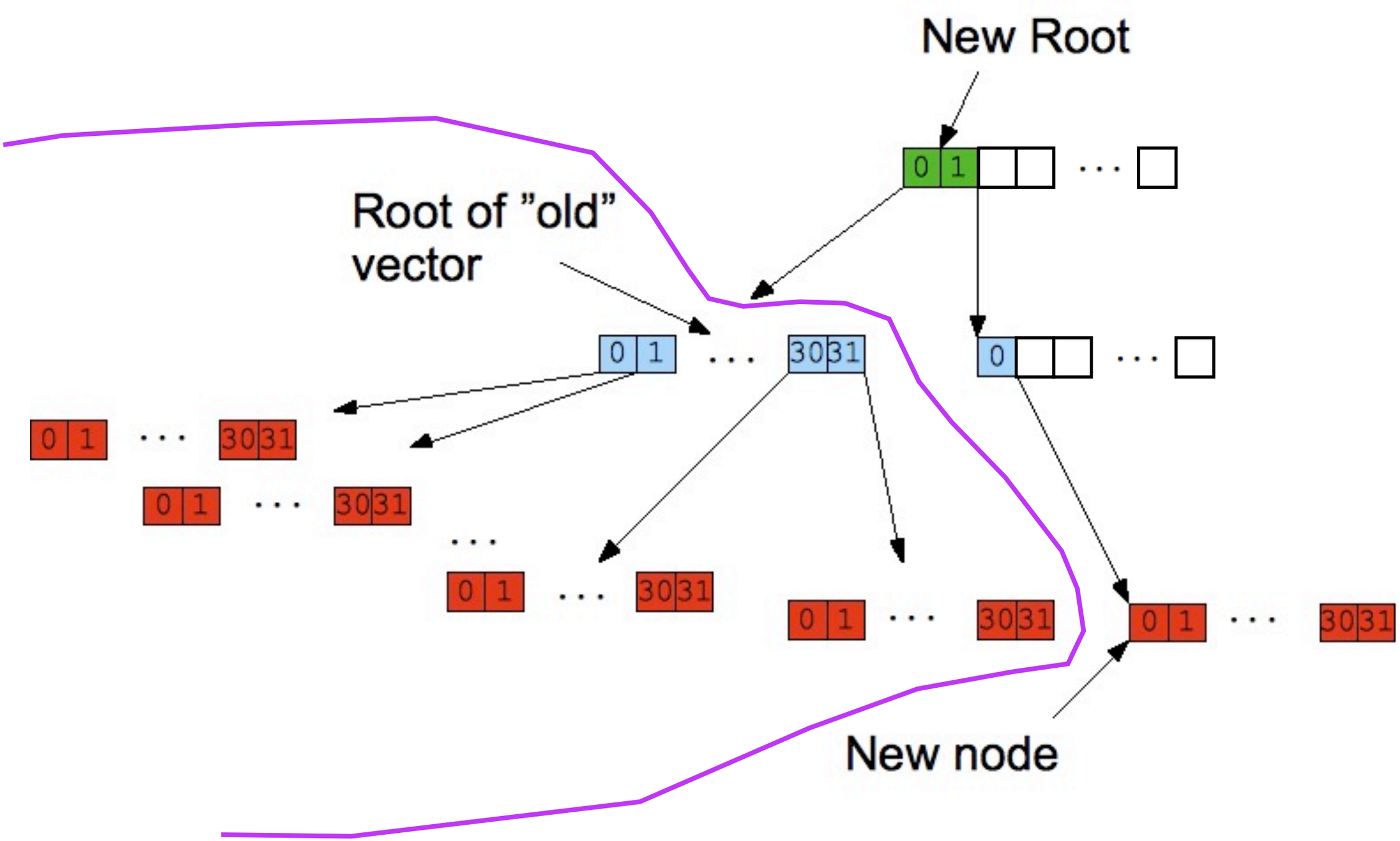
Easy-Peacy :)

```
public Object nth(int i)
{
 if(i >= 0 && i < cnt)
 {
  if(i >= tailoff())
    return tail[i & 0x01f];
  Object[] arr = root;
  for(int level = shift; level > 0; level -= 5)
    arr = (Object[]) arr[(i >>> level) & 0x01f];
  return arr[i & 0x01f];
 }
 throw new IndexOutOfBoundsException();
}
```

initial level (aka shift) is 10

```
1258:   [00][00000][00000][00000][00001][00111][01010]
```

Adding a level



What about HashMaps? PersistentHashMap

- How is a PersistentHashMap represented?
- How to lookup the value for a key?
- How to “add” a new key-value pair?

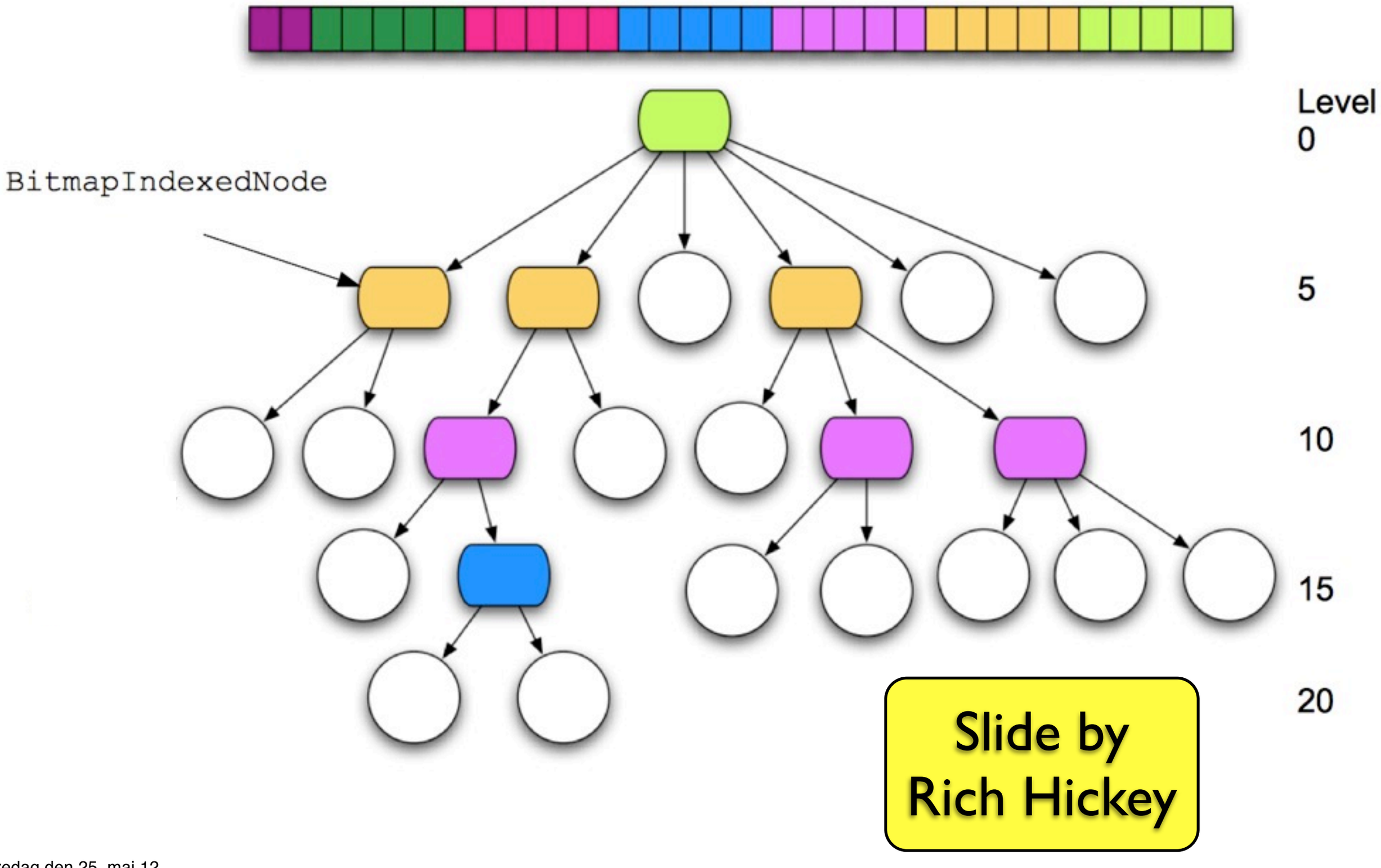
- Note this presentation talks about version 1.2 of Clojure.
  - There are changes in later versions.
  - These are not essential to this talk.

?? (clojure 1.2 version)

```
static int mask(int hash, int shift){
    return (hash >>> shift) & 0x01f;
}

static int bitpos(int hash, int shift){
        return 1 << mask(hash, shift);
}

final int index(int bit){
    return Integer.bitCount(bitmap & (bit - 1));
}

public LeafNode find(int hash, Object key){
        int bit = bitpos(hash, shift);
        if((bitmap & bit) != 0)
            {
            return nodes[index(bit)].find(hash, key);
            }
        else
            return null;
}
```

Bit-partitioned Hash Trie

Bit-partitioned Hash Trie



Slide by
Rich Hickey

BitmapIndexedNode

BitmapIndexedNode

- Holds an array of size < 32, pointing to children

BitmapIndexedNode

- Holds an array of size < 32, pointing to children
- Hard part is to only use as much space as is needed:
  - If node has ***n*** children, only use size ***n*** array;
  - and, doing a lookup on a BitmapIndexedNode to find a child must be fast constant time

BitmapIndexedNode

- Holds an array of size < 32, pointing to children
- Hard part is to only use as much space as is needed:
  - If node has ***n*** children, only use size ***n*** array;
    - and, doing a lookup on a BitmapIndexedNode to find a child must be fast constant time
- The trick is to find an efficiently computable function to map between a 5-bit number (i.e., a bit block) and index, $0 \leq i < n$ in child array.

BitmapIndexedNode

The bitmap

BitmapIndexedNode

The bitmap

- Consider the mapping
  - `bitpos: [0, 31] => integer`
  - `bitpos(n) = 2^n` (as bitpattern: `10^n`)
    - e.g., 13 `bitpos(13) = 00000010000000000000`

BitmapIndexedNode

The bitmap

- Consider the mapping
  - `bitpos: [0, 31] => integer`
  - `bitpos(n) = 2^n` (as bitpattern: `10^n`)
    - e.g., 13 `bitpos(13) = 00000010000000000000`
- A `bitmap` is maintained which is a bit-pattern
  - e.g., `00000100000001100010001000000001`

BitmapIndexedNode

The bitmap

- Consider the mapping
  - `bitpos: [0, 31] => integer`
  - `bitpos(n) = 2^n` (as bitpattern: `10^n`)
    - e.g., 13 `bitpos(13) = 00000010000000000000`
- A `bitmap` is maintained which is a bit-pattern
  - e.g., `00000100000001100010001000000001`
- To check if an bitblock is in the array just match:
  - `00000100000001**1**00010001000000001`
  - `00000000000001**1**00000000000000`

Bitmap: indexing

Bitmap: indexing

- For a given bitmap, e.g.,
  - 00000*1*0000000*1***1**00**1**0000**1**000000001

Bitmap: indexing

- For a given bitmap, e.g.,
  - 00000*1*0000000*1***1**00**1**0000**1**000000000**1**
- The index of an bit block, call bitpos first, e.g.:
  - 000000000000***1***00000000000000000000

Bitmap: indexing

- For a given bitmap, e.g.,
  - `00000`*`1`*`0000000`*`1`*`1`**`1`**`00`**`1`**`0000`**`1`**`000000000`**`1`**
- The index of an bit block, call bitpos first, e.g.:
  - `000000000000`***`1`***`0000000000000000000`
    - Is the number of `1`'s below this bitpos, in the bitmap, in the above example: 4.

Bitmap: indexing

- For a given bitmap, e.g.,
  - 00000*1*0000000*1***1**00**1**0000**1**000000000**1**
- The index of an bit block, call bitpos first, e.g.:
  - 000000000000***1***00000000000000000000
    - Is the number of 1's below this bitpos, in the bitmap, in the above example: 4.
  - 000000000000***1***00000000000000000000

Bitmap: indexing

- For a given bitmap, e.g.,
  - `00000`***`1`***`0000000`***`1`***`00`**`1`**`0000`**`1`**`000000000`**`1`**
- The index of an bit block, call bitpos first, e.g.:
  - `0000000000000`***`1`***`0000000000000000000`
    - Is the number of `1`'s below this bitpos, in the bitmap, in the above example: 4.
  - `0000000000000`***`1`***`0000000000000000000`
  - `0000000000000`***`0`***`1111111111111111` `(index - 1)`

Bitmap: indexing

- For a given bitmap, e.g.,
  - `00000`*`1`*`0000000`*`1`*`1`**`1`**`00`**`1`**`0000`**`1`**`000000000`**`1`**
- The index of an bit block, call bitpos first, e.g.:
  - `000000000000`***`1`***`0000000000000000000`
    - Is the number of `1`'s below this bitpos, in the bitmap, in the above example: 4.
  - `000000000000`***`1`***`0000000000000000000`
  - `000000000000`***`0`***`1111111111111111` `(index - 1)`
  - `00000`*`0`*`0000000`*`0`*`1`**`1`**`00`**`1`**`0000`**`1`**`000000000`**`1`**

Bitmap: indexing

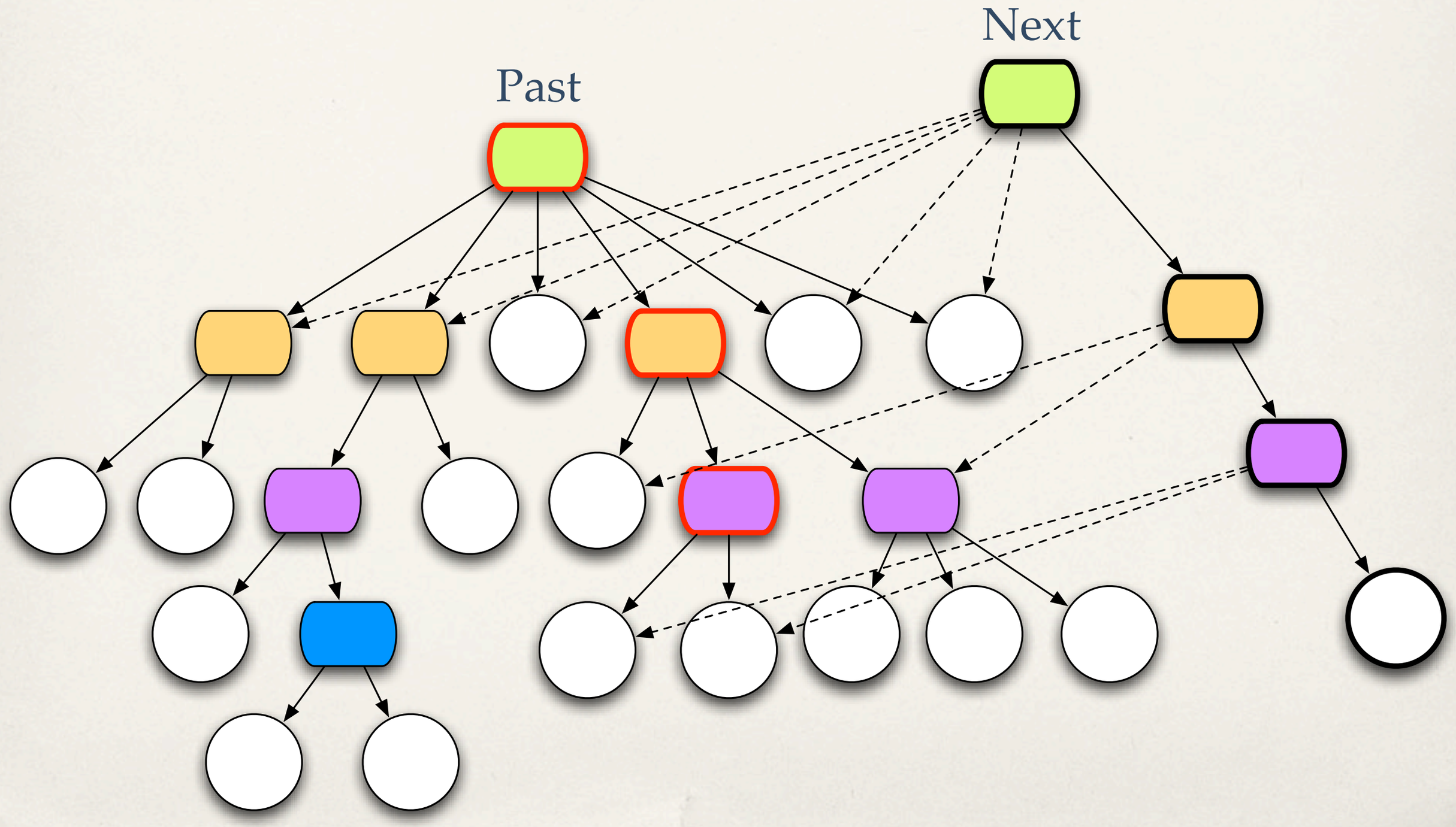
- For a given bitmap, e.g.,
  - `00000`*`1`*`0000000`*`1`***`1`**`00`***`1`**`0000`***`1`**`000000000`***`1`**
- The index of an bit block, call bitpos first, e.g.:
  - `000000000000`***`1`***`0000000000000000000`
  - Is the number of `1`'s below this bitpos, in the bitmap, in the above example: 4.
  - `000000000000`***`1`***`0000000000000000000`
  - `000000000000`***`0`***`1111111111111111` `(index - 1)`
  - `00000`*`0`*`0000000`*`0`***`1`**`00`***`1`**`0000`***`1`**`000000000`***`1`**
  - On many modern processors there is an instruction CTPOP/ POPCNT (count population)

Finding a node

```
static int mask(int hash, int shift){
    return (hash >>> shift) & 0x01f;
}

static int bitpos(int hash, int shift){
        return 1 << mask(hash, shift);
}

final int index(int bit){
    return Integer.bitCount(bitmap & (bit - 1));
}

public LeafNode find(int hash, Object key){
        int bit = bitpos(hash, shift);
        if((bitmap & bit) != 0)
            {
            return nodes[index(bit)].find(hash, key);
            }
        else
            return null;
}
```

Structural Sharing

Slide by Rich Hickey



Other benefits

Other benefits

- PersistentHashMap and PersistentVector amenable to parallel processing.
  - Divide and conquer (you already did half the work :)
  - No special parallel collection types needed

Other benefits

- PersistentHashMap and PersistentVector amenable to parallel processing.
  - Divide and conquer (you already did half the work :)
  - No special parallel collection types needed
- Clojure 1.5 will (likely) have a parallel processing function
  - ***fold*** which takes a combining fn, a reducing fn and a collection
  - Uses Java's Fork/Join framework for parallel processing.
  - Code has the same shape as existing (serial) clojure code

Other benefits

- PersistentHashMap and PersistentVector amenable to parallel processing.
  - Divide and conquer (you already did half the work :)
  - No special parallel collection types needed
  - Clojure 1.5 will (likely) have a parallel processing function
    - ***fold*** which takes a combining fn, a reducing fn and a collection
    - Uses Java's Fork/Join framework for parallel processing.
    - Code has the same shape as existing (serial) clojure code
  - Live Example?

Summary

- For your own sake, please start doing functional programming. Benefits
  - Sanity
  - Concurrency
  - Parallelism
  - (and it's fun too)
- I recommend trying out Clojure :)
- Option clj-ds: https://github.com/krukow/clj-ds
  - Port of Clojure's persistent data structures to use (when stuck) with other JVM languages
  - To be updated to support the upcoming parallel processing in Clojure

References

- Rich Hickey
  - Are we there yet? http://www.infoq.com/presentations/Are-We-There-Yet-Rich-Hickey
  - Clojure concurrency http://blip.tv/clojure/clojure-concurrency-819147
  - http://clojure.com/blog/2012/05/08/reducers-a-library-and-model-for-collection-processing.html
  - http://clojure.com/blog/2012/05/15/anatomy-of-reducer.html
- My Blog
  - http://blog.higher-order.net/2009/02/01/understanding-clojures-persistentvector-implementation
  - http://blog.higher-order.net/2009/09/08/understanding-clojures-persistenthashmap-deftwice/

References

- Chas Emerick, Brian Carper, & Christophe Grand: *Clojure Programming*, 2012, O'Reilly

- clj-ds: https://github.com/krukow/clj-ds
  - Port of Clojure's persistent data structures to use (when stuck) with other JVM languages
  - To be updated to support parallelism (via Clojure's reducers)

- Conj Labs - 3 day Clojure training in Europe
  - Lau Jensen & Karl Krukow
  - Contact me karl@lesspainful.com for details

Questions?

Making app testing less painful...
Please contact us with any questions:
contact@lesspainful.com
karl@lesspainful.com - iOS
jonas@lesspainful.com - Android

http://www.lesspainful.com