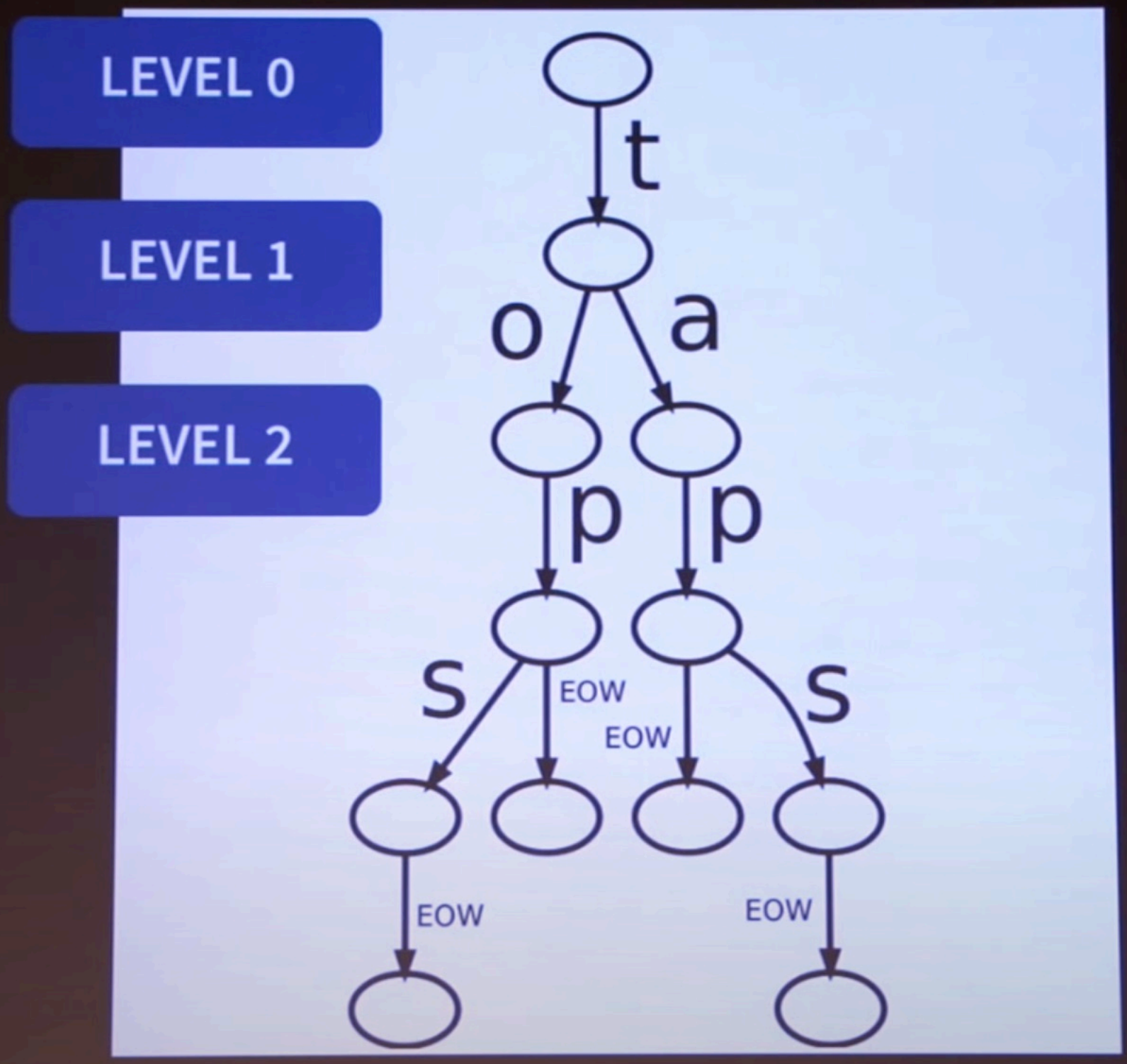
Persistent Data Structures

In Clojure Language

A Better Presentation On Clojure Will be made after Exam

- Composite values - immutable
- 'Change' is merely a function, takes one value and returns another, 'changed' value
- Collection maintains its performance guarantees
  - Therefore new versions are not full copies
- Old version of the collection is still available after 'changes', with same performance
- Example - hash map/set and vector based upon array mapped hash tries (Bagwell)

A Better Presentation On Clojure Will be made after Exam



00000 01011 01001 11001 10111 00010 11100

0 11 9 25

ArrayNode
shift = 0

28

ArrayNode
shift = 5

2

ArrayNode
shift = 10

23

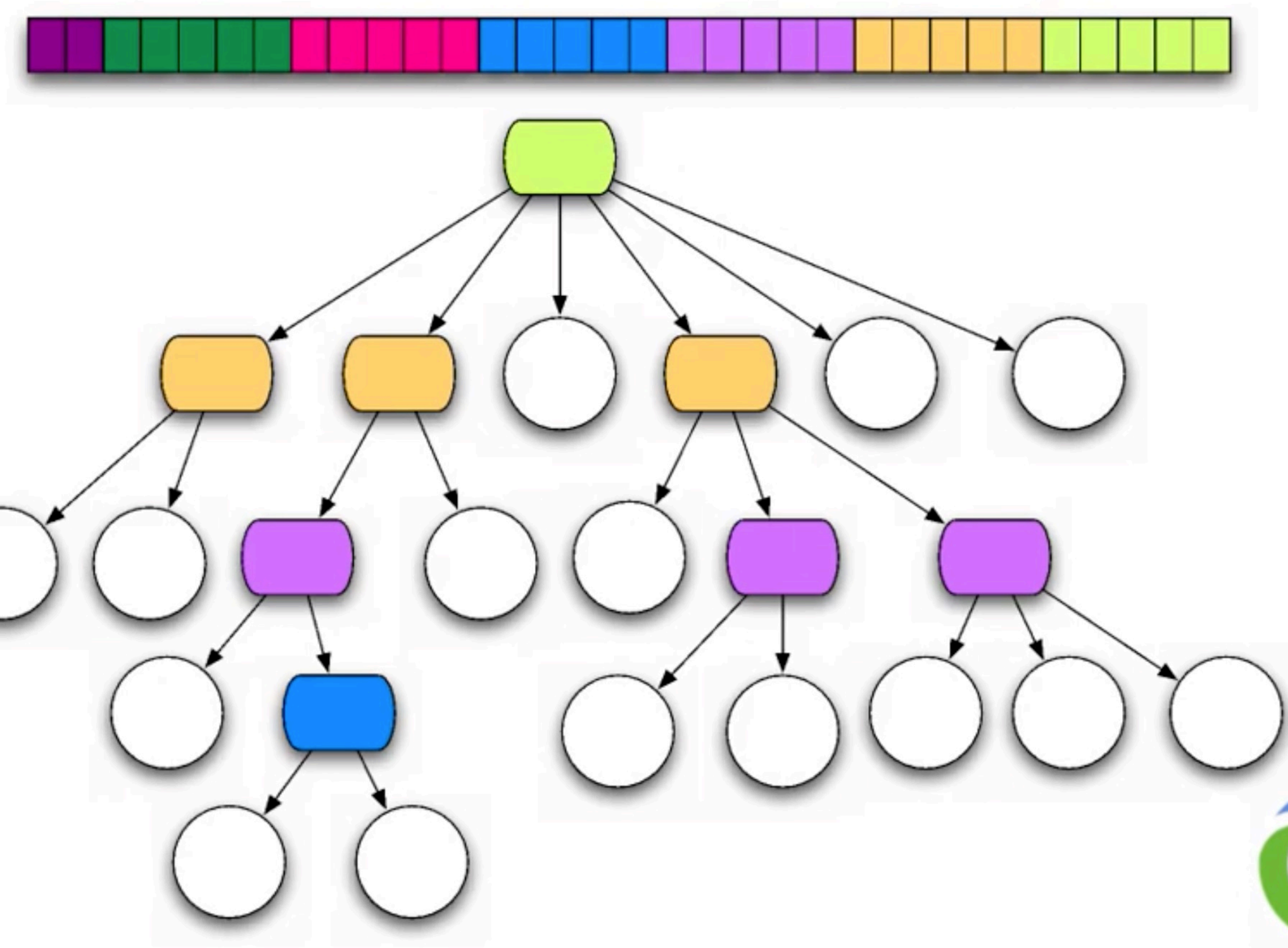
BitmapIndexedNode
shift = 15

A Better Presentation On Clojure Will be made after Exam

… and then follow the AMT down

A Better Presentation On Clojure Will be made after Exam

Bit-partitioned hash tries



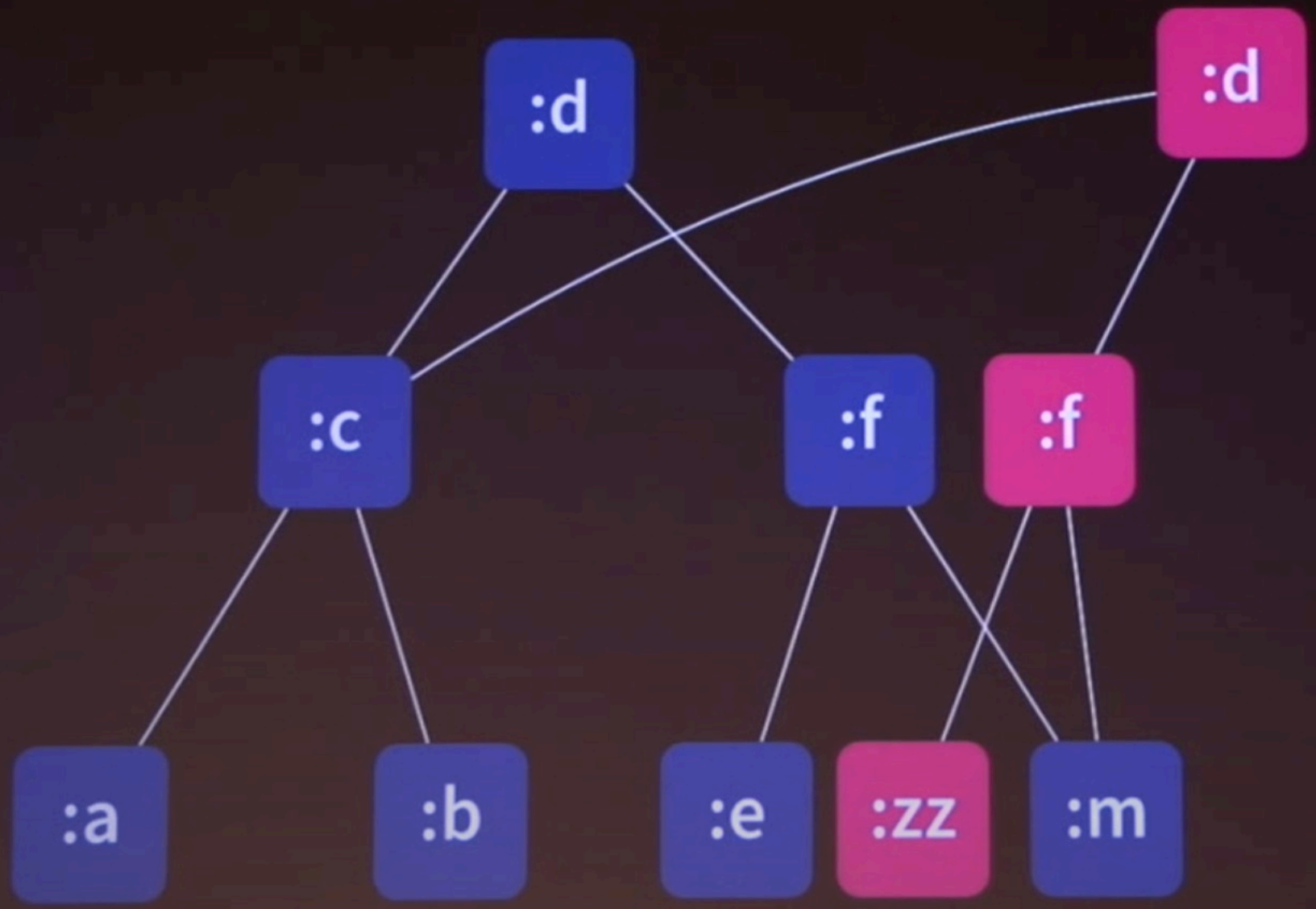
A Better Presentation On Clojure Will be made after Exam

Structural Sharing

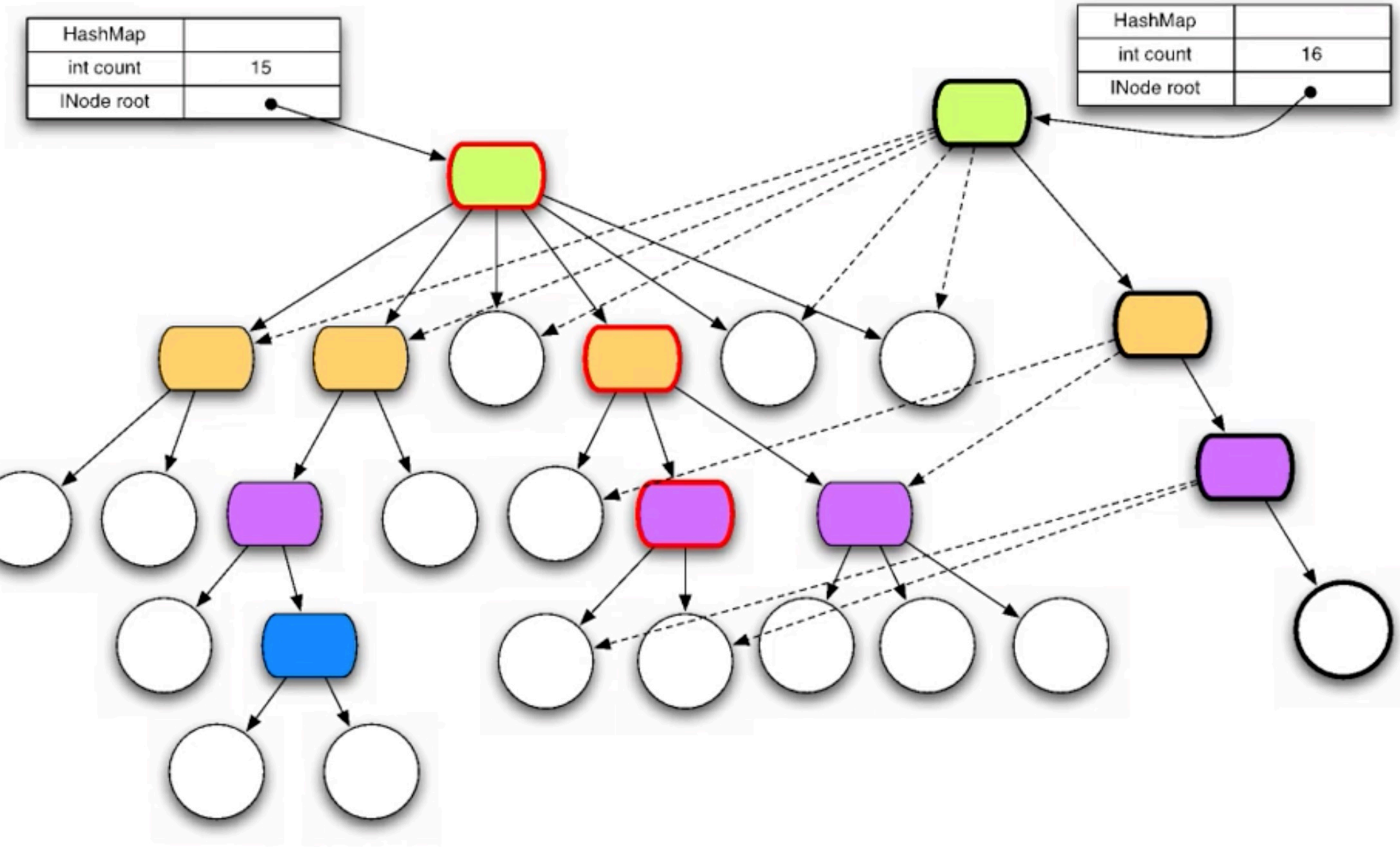
- Key to efficient 'copies' and therefore persistence
- Everything is immutable so no chance of interference
- Thread safe
- Iteration safe

STRUCTURAL SHARING

(assoc v 4 :zz)



Path Copying

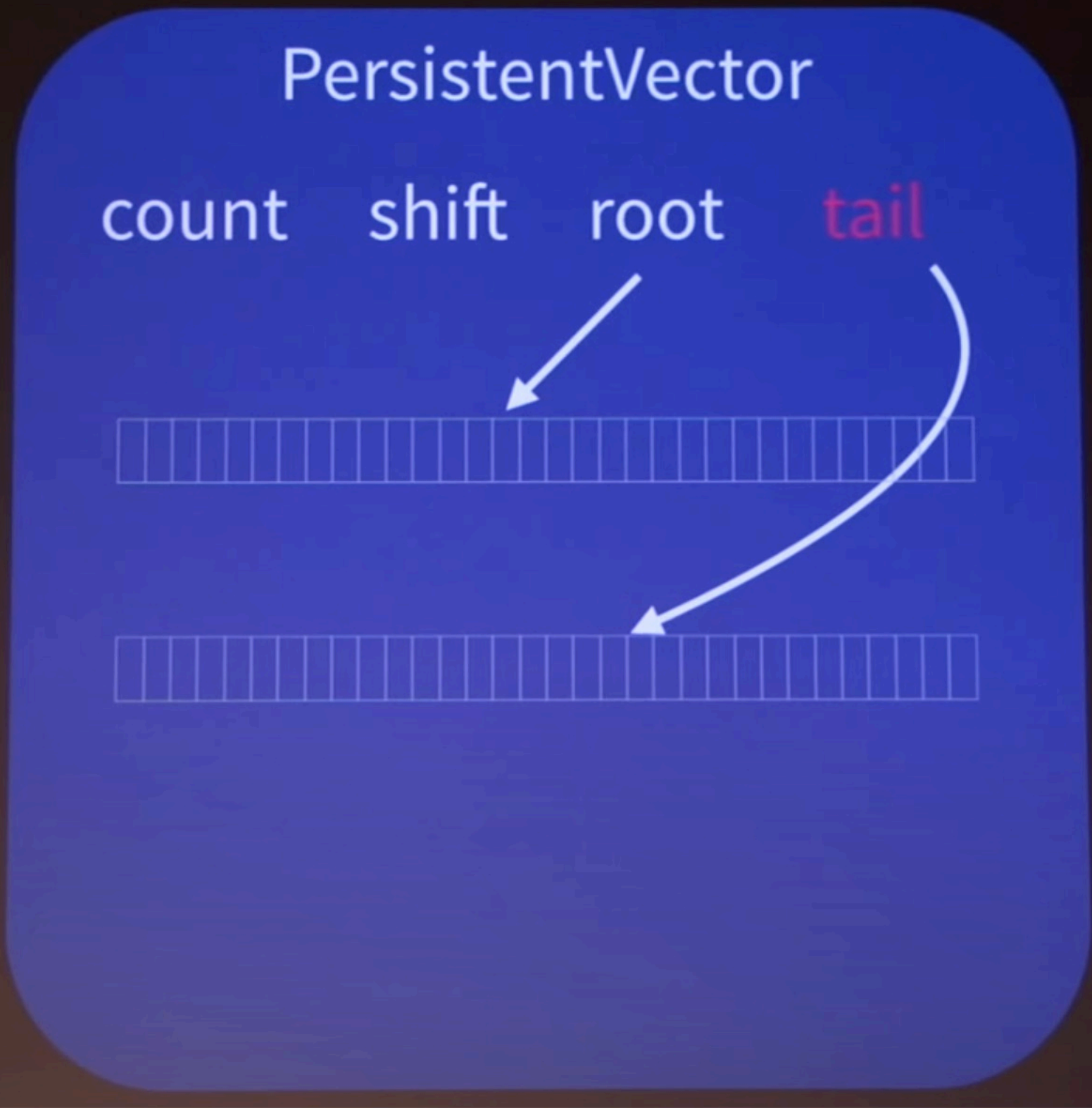


REGULAR HASH TABLE?

NEED ROOT RESIZING

NOT AMENABLE TO STRUCTURAL SHARING

THE TAIL OPTIMIZATION



Slides are taken from, the seminars:

1. What Lies Beneath - A Deep Dive Into Clojure's Data Structures - Mohit Thatte
2. Persistent Data Structures and Managed References - Rich Hickey