Clojure Hash Maps:

plenty of room at the bottom

@spinningtopsofdoom
@2kliph
@bendyworks

Building an alien space ship

- Avoiding the gray goo scenario when making nano machines
- What cup of tea is best to power your Infinite Improbability Drive (earl gray hot)
- How to make the spaceship bigger on the inside then on the outside

Talk about real alien technology

CAUTION:
LISP
Made with secret alien technology

Immutability: a cornerstone of functional programming

See it's used in

- Scala
- Elixir
- Haskell
- Clojure

Why immutable?

- Deeply nested heterogeneous data
- Send data off to another part of the code: fire and forget :)
- Fast delta diffing
  - E.g. React shouldComponentUpdate

There's always a catch

- Orders of magnitude slower
- Efficient implementations have constraints, like sortable keys, storing deltas in the data structure itself
  - Increasing cognitive overhead for developers

Hash Array Mapped Tries provide performance improvements

- 2 to 3 times slower for common operations
  - That's a lot better than an order of magnitude slower
- No constraints
  - Only need a hashable key
- Reduced cognitive overhead

Optimizing Hash-Array Mapped Tries for Fast and Lean Immutable JVM Collections

by Michael J. Steindorfer and Jurgen J. Vinju

Compressed Hash-Array Mapped Prefix-tree

CHAMP

ClojureScript Implementation

https://github.com/bendyworks/lean-map

CHAMP gives you guaranteed Hash Map performance gains

- Iteration by 2x
- Equality checking by 10x to 100x

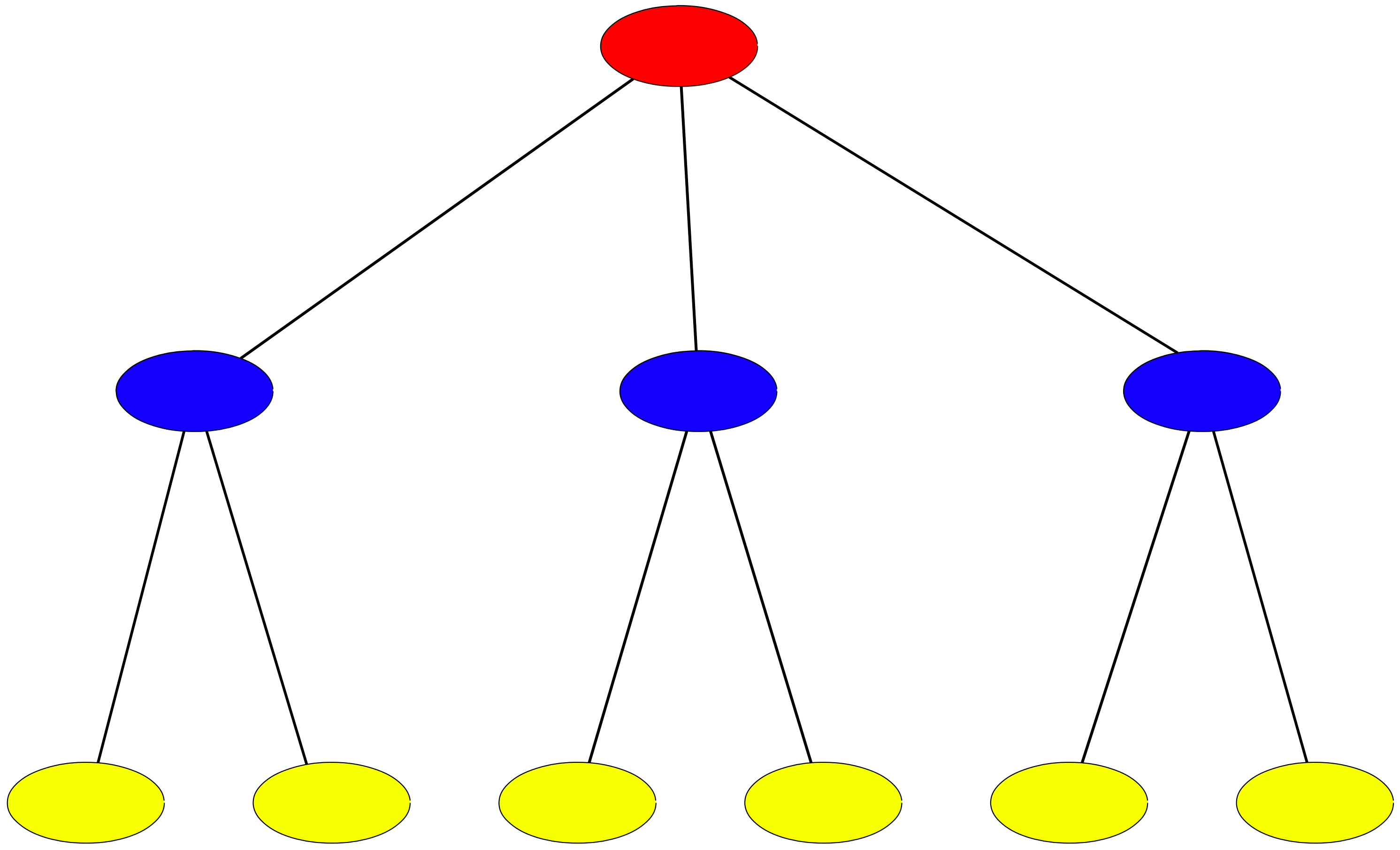
CHAMP trims your Hash Maps

CHAMP makes Hash Maps more wieldy, making them both simpler and easier

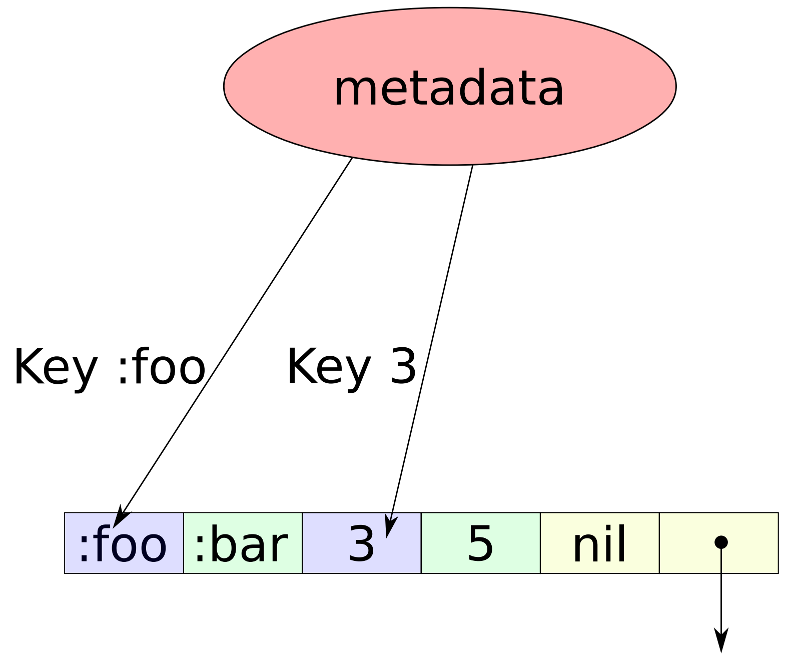
Code size is **two thirds** the size of the original implementation

Overview of Clojure Hash Maps

Clojure Hash Maps tree of nodes 32 way branching factor



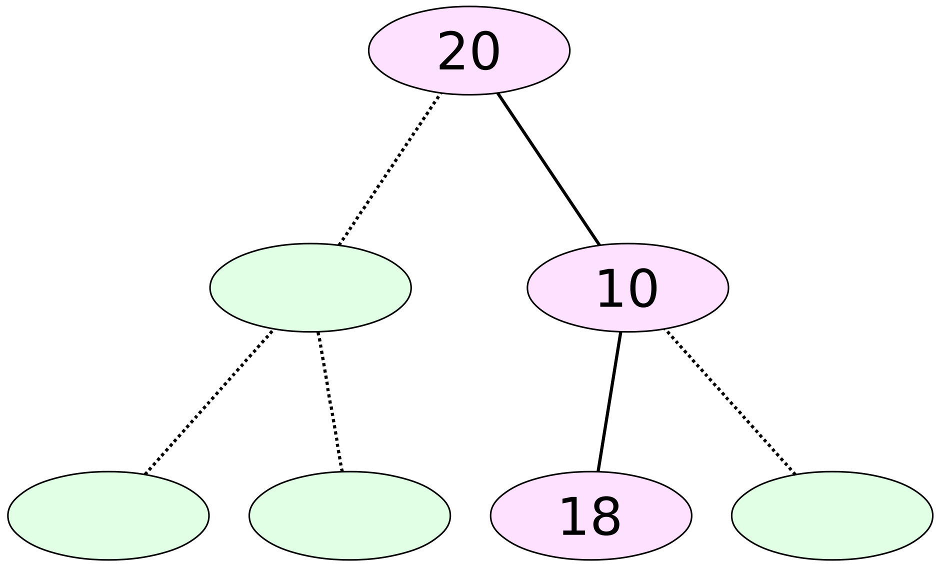
Node internals



How a key finds a node

Key: :foo Hash: 1268894036

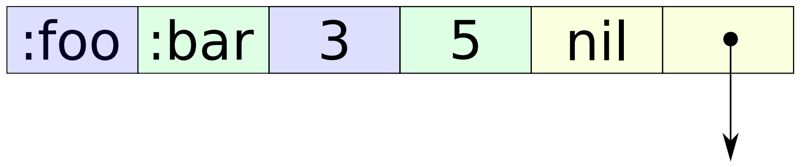
20	10	18	3	26	5	1



First major improvement

Removes problems with sub node references

Sub node reference is a psuedo Key Value pair with `nil` as the "key"

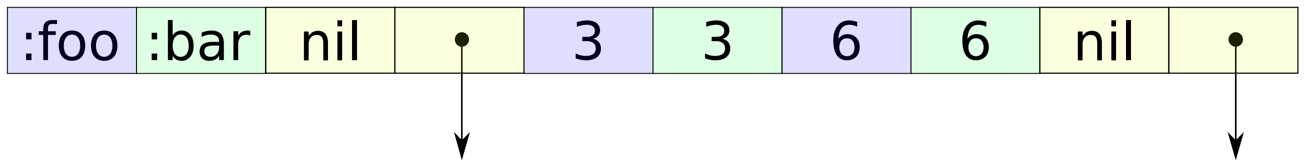


Doubles overhead for each sub node reference

Adds incidental complexity

- Needs a flag for `nil` key and field for `nil` values
- Optimized node (Array Node) just containing sub node references
  - Happens when normal node's array has 32 elements
- Further complications with second problem

Sub node references are scattered throughout a nodes array



Combined with `nil` marker value makes that you you have to ask

"Is it a Key Value pair or sub node reference?"

for **every** operation

Makes iteration a wiki walk

The Roman Empire was the post-**Roman Republic** period

The Roman Republic was the period of **ancient Roman civilization** beginning with the

Lots more link clicking...

Awareness is the ability to perceive, to feel, or to be conscious of events, objects, thoughts, emotions, or sensory patterns

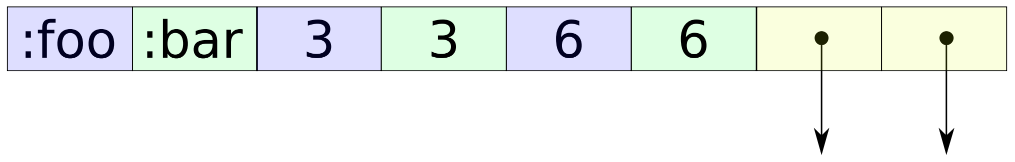
What was the next word after **Roman Republic**?

Wiki Walk Iteration

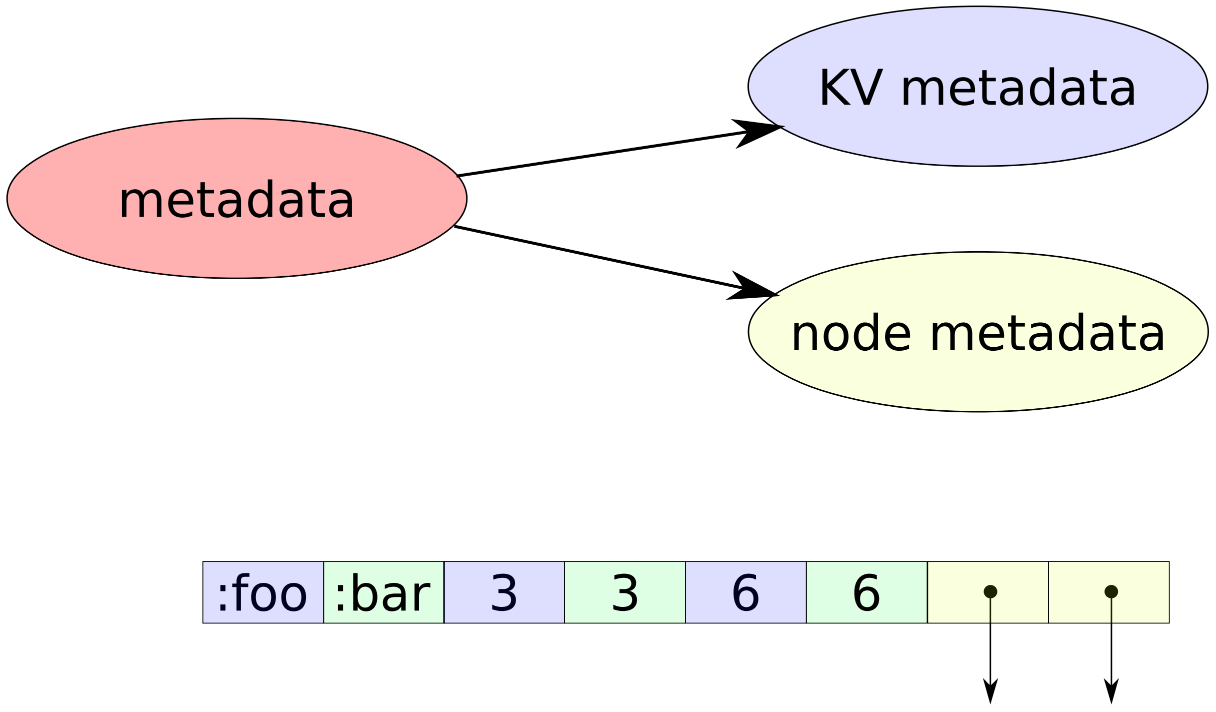
- Bad locality
  - Blows the stack
  - CPU caches are never hot

CHAMP node improvements

Key Value Pairs in front, Sub Node references in back



Decomplect metadata



Lower memory overhead by removing `nil` marker values

Removes all sub node incidental complexity

- nil key flag
- nil value field
- Array Node
- Check for Key Value or Sub node reference

2X speedup by changing iteration from wiki walk to a linear scan

Original Hash Map iteration algorithm (pseudocode)

- If nil flag is true return `[nil, <nil value>]`
- For normal nodes
  - If key is not `nil` then return the Key Value pair
  - Otherwise go to sub node and repeat
- For Array node
  - If element is `nil` continue
  - Otherwise go to sub node and repeat

CHAMP iteration algorithm

1. Iterate though Key Value pairs
2. Iterate through sub node(s) repeating step one

Comparison

- Seven lines vs two lines
- Three conditionals vs none
- Polymorphism vs no polymorphism

CHAMP Equality Check improvements

Clojure Puzzler

Sloppy Cleaning

```clojure
(def base-map (hash-map))
(def one-million 1000000)

(def full-map
  (reduce
    (fn [m i] (assoc m i 0))
    base-map
    (range one-million)))

(def same-map
  (reduce
    (fn [m i] (dissoc m I))
    full-map
    (range one-million)))

(= base-map same-map) ;; true
(time (into {} base-map)) ;; 140 microseconds
(time (into {} same-map)) ;; ??? microseconds
```

A) 140 microseconds

B) 280 microseconds

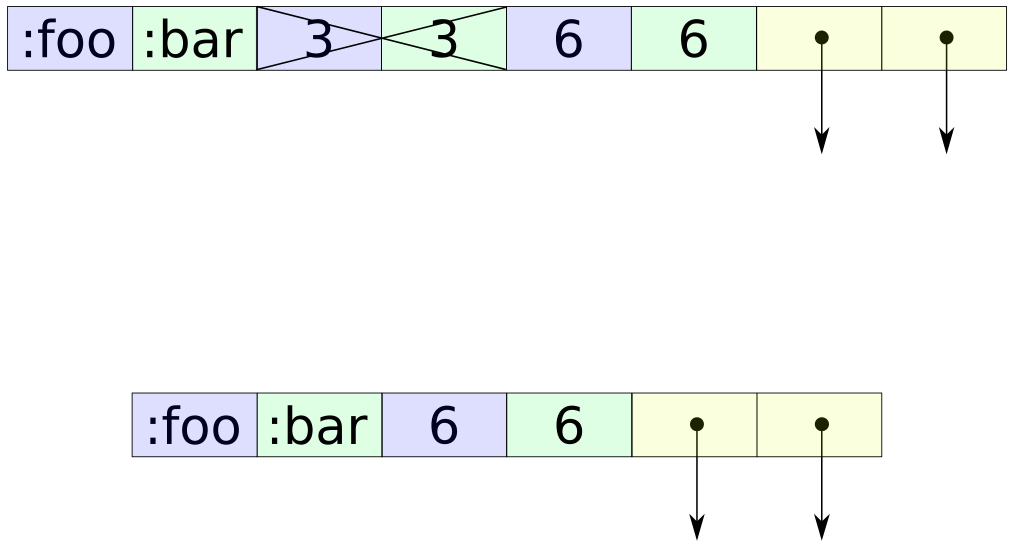
C) 1400 microseconds

D) 14000 microseconds

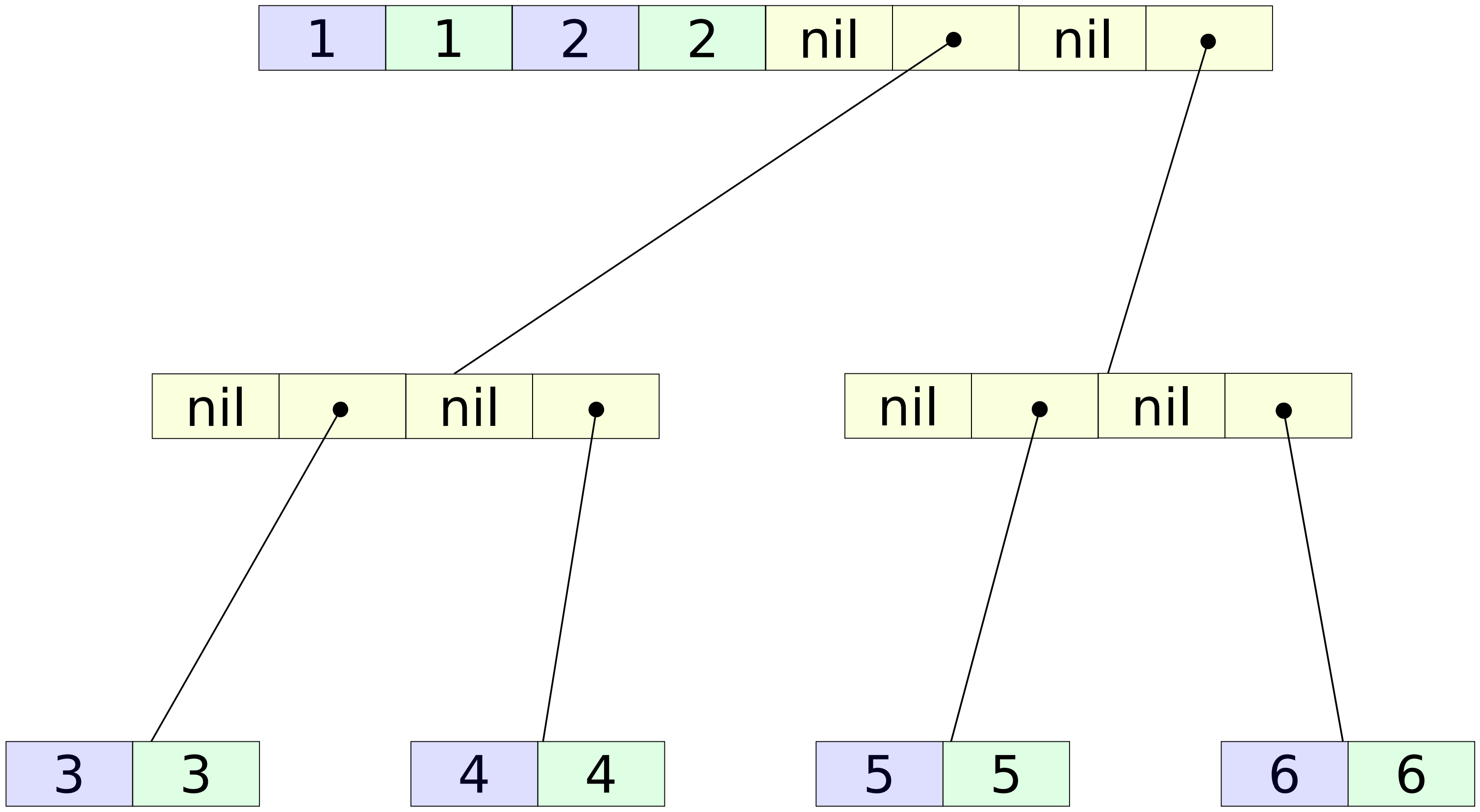
E) 31000 microseconds

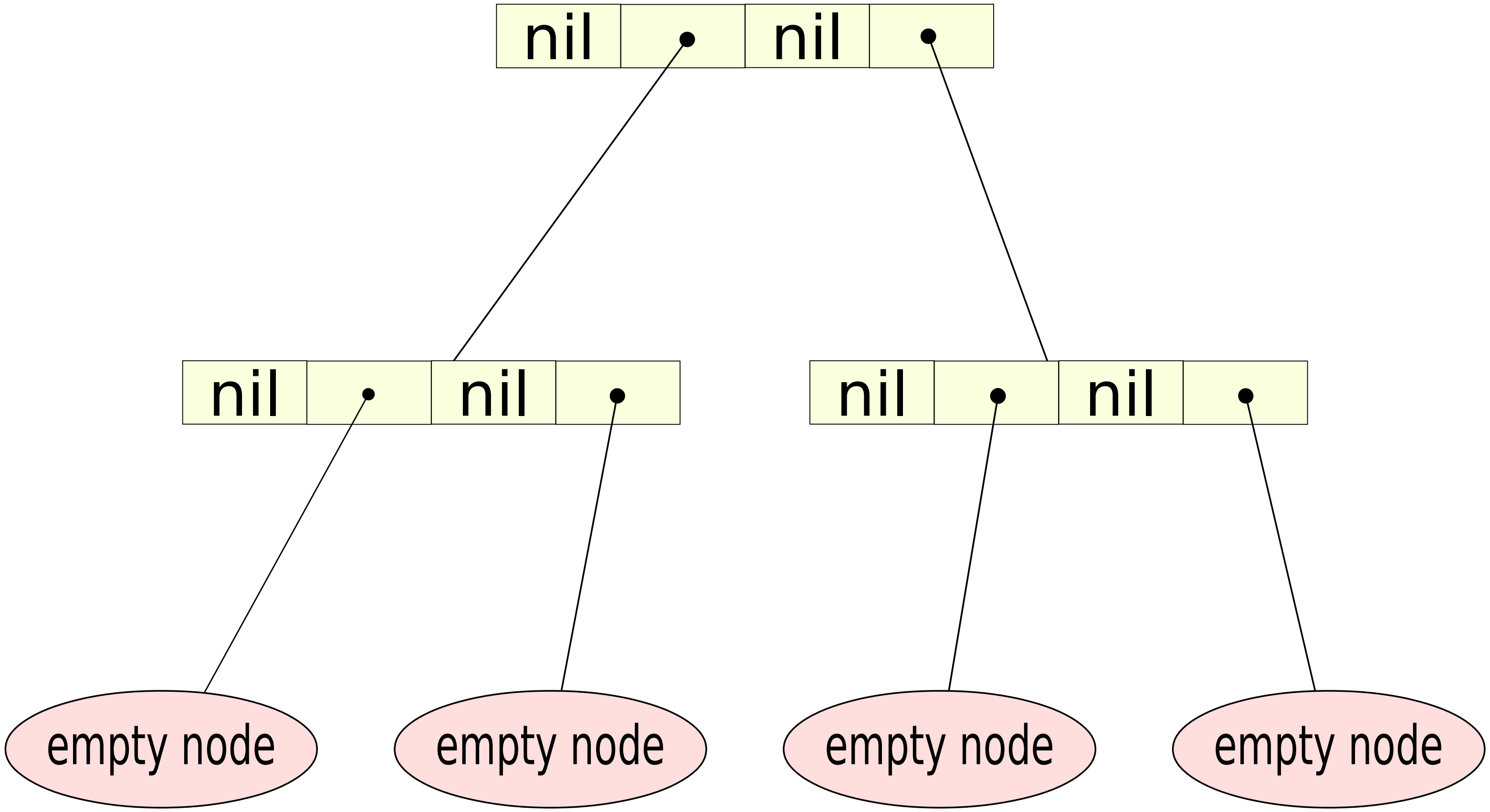
E) 31000 microseconds

Original Delete Algorithm

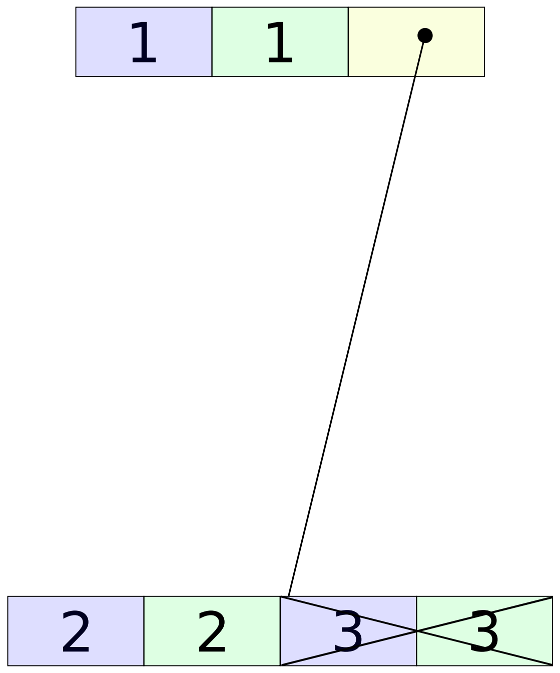


This leads to

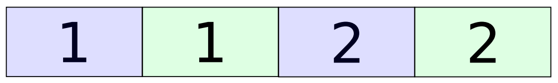

1
1
2
2
nil
nil
nil
nil
nil
nil
3
3
4
4
5
5
6
6


nil
nil
nil
nil
nil
nil
empty node
empty node
empty node
empty node

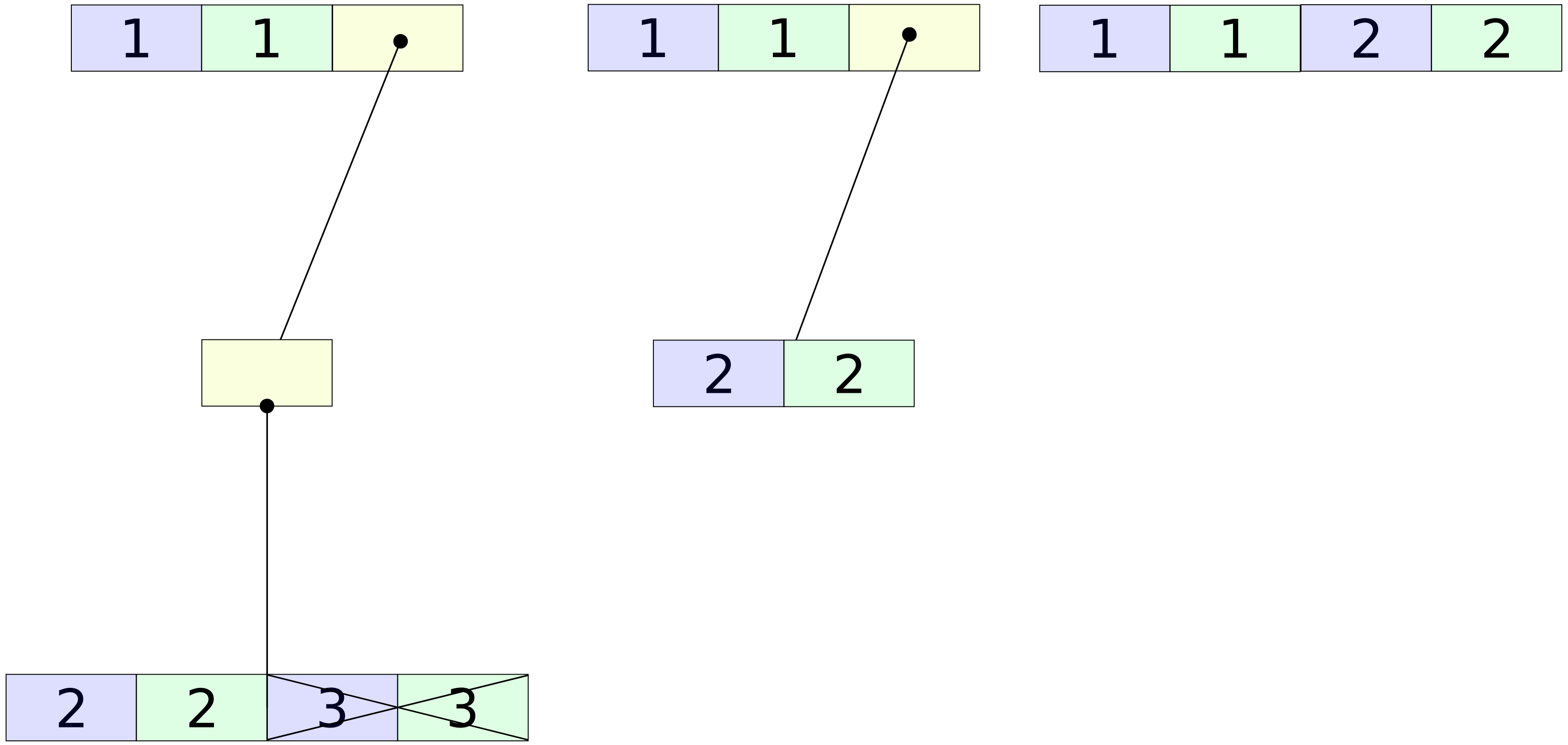
CHAMP Delete Algorithm



1
1
2
2
3
3



1
1
2
2


1
1
1
1
1
1
2
2
2
2
2
2
3
3

Lowers memory overhead that occurs from deletion

So what? This only really matters in pathological cases

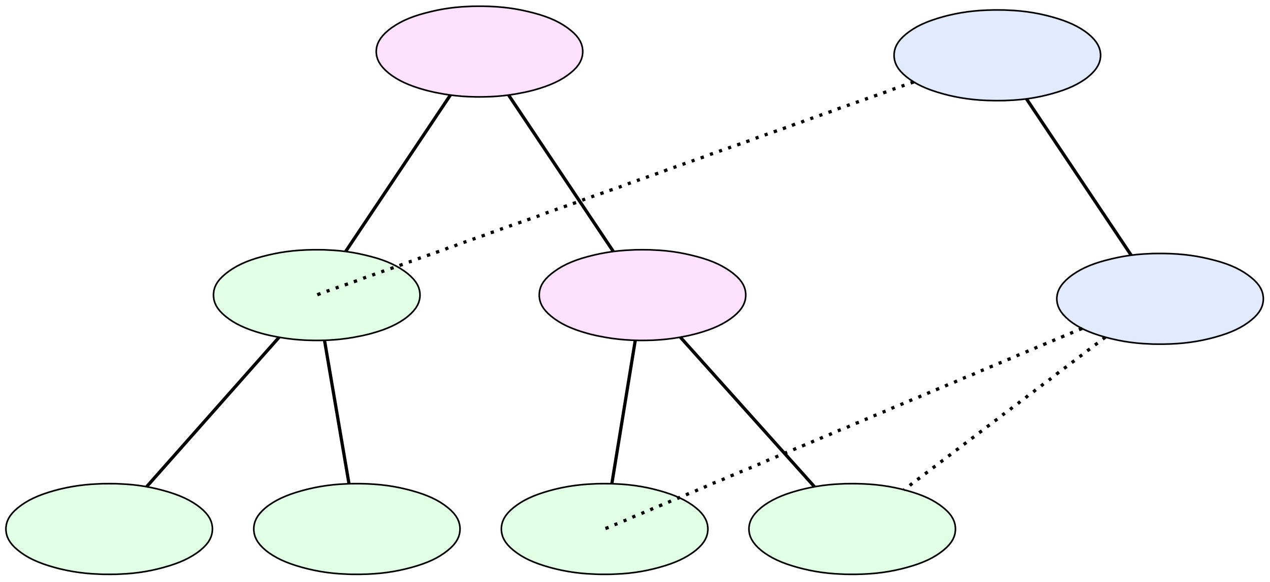
Equal CHAMP maps have the exact same layout in memory

We don't have to compare all Key Values we can compare nodes (pointer equality)

Equality check is now O(log n) vs O(n) leading to 100x performance improvement

Assuming maps share structure

Structural Sharing



We still get 10x performance boost for maps don't share any structure

- Original comparison has overhead due to Clojure abstractions (sequences and lookup)
- CHAMP comparison is only comparing two arrays

Caveats

- Javascript version: addition: 8% slower; deletion: 10 - 20% slower
  - Compared to current ClojureScript version
- JVM version: comparable speed to HAMT
  - Used in Rascal (Steindorfer & Vinju)
  - Christopher Grand has ported CHAMP to Java using Clojure's hashing functions

CHAMP improvements paves the way for future improvements

CHAMP internals are much easier to work with and reason about

Two Future possibilities

- Merge and Diff operations could have greatly increased performance
- Similar to RRB Vectors for Vectors

Interesting work on merging

- Christopher Grand is investigating using CHAMP as a basis for confluent hash maps
  - Uses node metadata to mark transient / persistent nodes
  - Removes marker objects needed for addition and deletion
  - Makes CHAMP able to merge hash maps in O(log n) time

CHAMP is not as cool as working nanobots

CAUTION:
LISP
Made with secret
alien technology

CHAMP shows Hash Maps have plenty of room at the bottom compared to original ClojureScript HAMT implementation

- 2x performance for iteration
- 10 - 100x performance for equality checking
- Lower memory overhead

For Peter biggest win is making Hash Maps much easier to understand and implement

Clojure Hash Maps is one of Clojure's best exports

- Scala (base hash map)
- Elixir (base hash map)
- Haskell (unordered-containers)
- Ruby (hamster)
- JavaScript (immutable.js)

Thanks

- Bendyworks for supporting my work on this
- Michael J. Steindorfer and Jurgen J. Vinju for the CHAMP Paper
- Zach Tellman for writing Collection Check
- Martin Klepsch for porting Collection Check to ClojureScript
- Nicolás Berger for helping me setup test harness
- David Nolen for performance and profiling suggestions

Fin

Questions?