Introduction to the HAMT: Opportunity for Tcl

2017 Tcl Conference
Don Porter
Tcl/Tk Release Manager

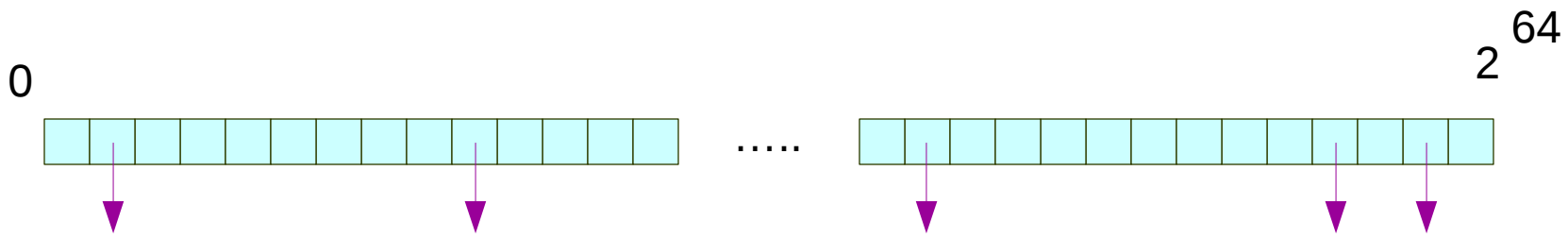
Hash Maps in Tcl

- Dictionaries
- Array variables
- Name lookups (commands, vars, etc.)
- Much much more…
  - Most make use of Tcl_HashTable.
    - Customizable

Hash Map – Giant Bucket Array

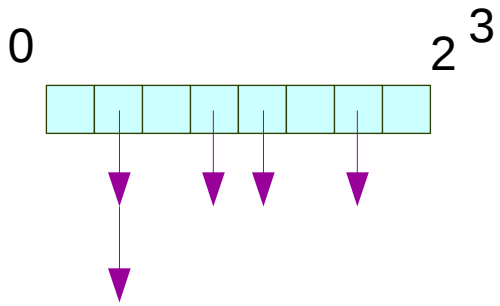
- Define Hash: Key → index
  - Efficient
  - Range evenly distributed over indices

Search bucket [Hash(key)] for key



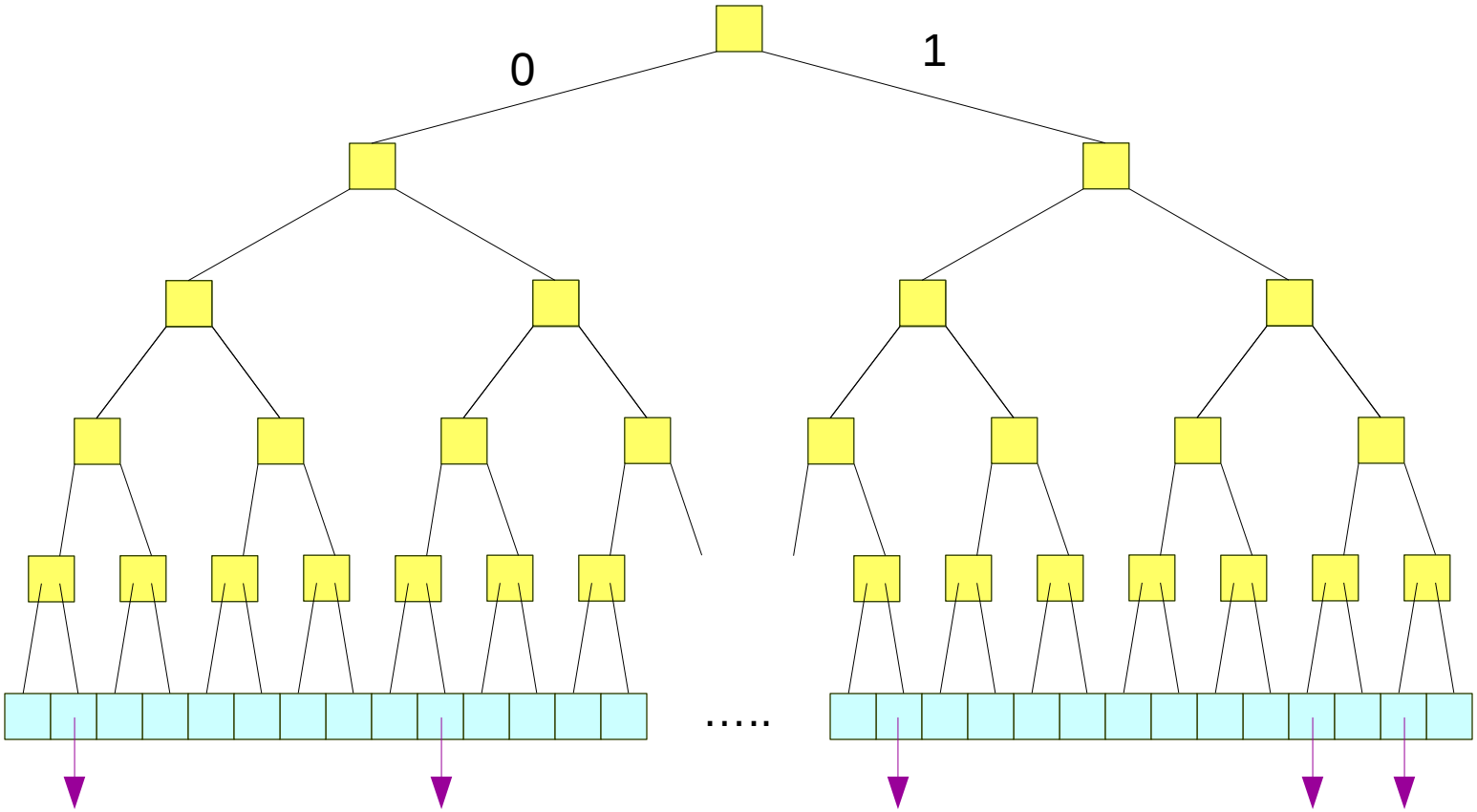
Hash Map – Tcl_HashTable

Search bucket [Hash(key) & mask] for key



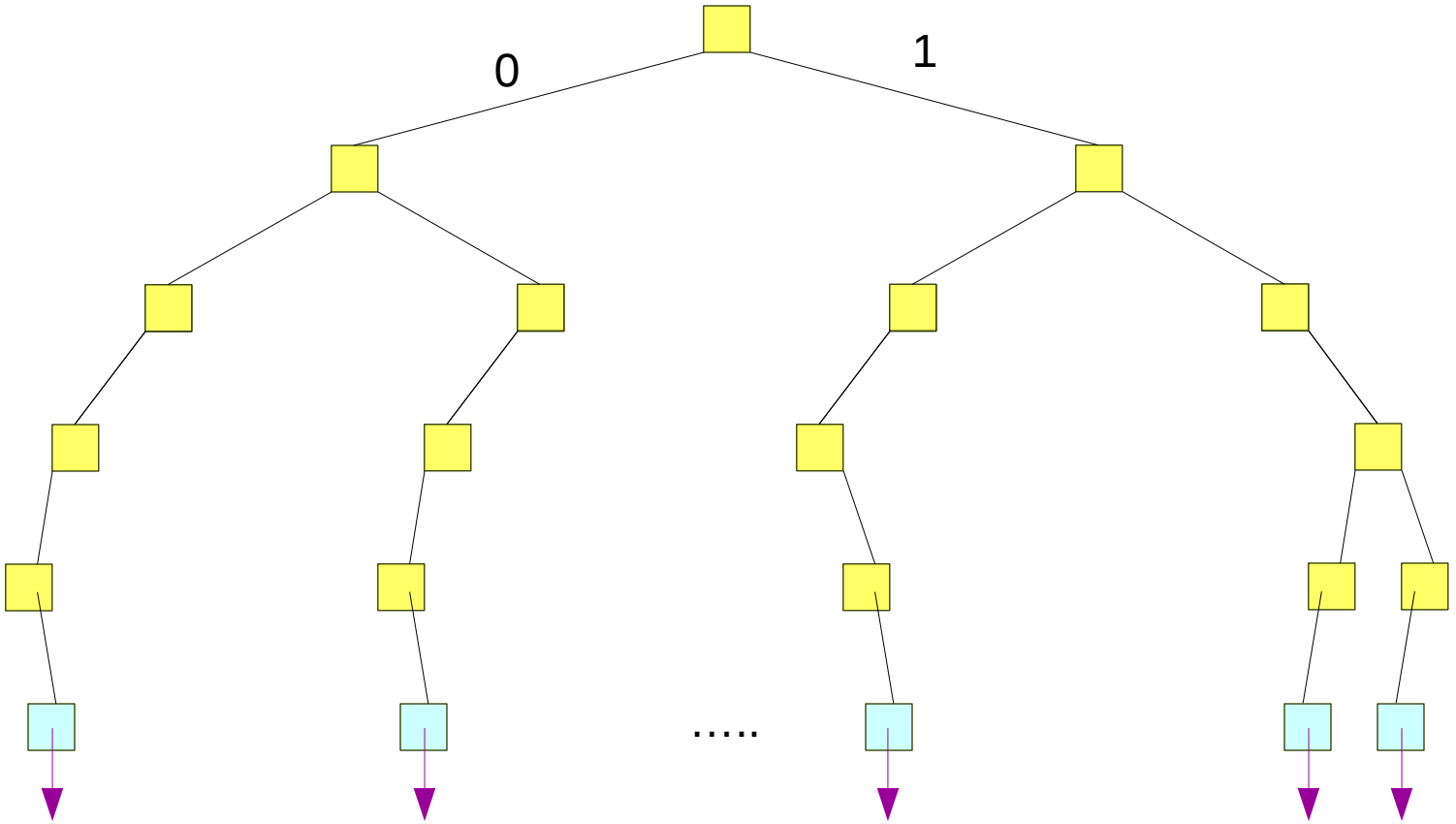
Hash Map – Hash Trie

Follow Hash(key) path to leaf storing key



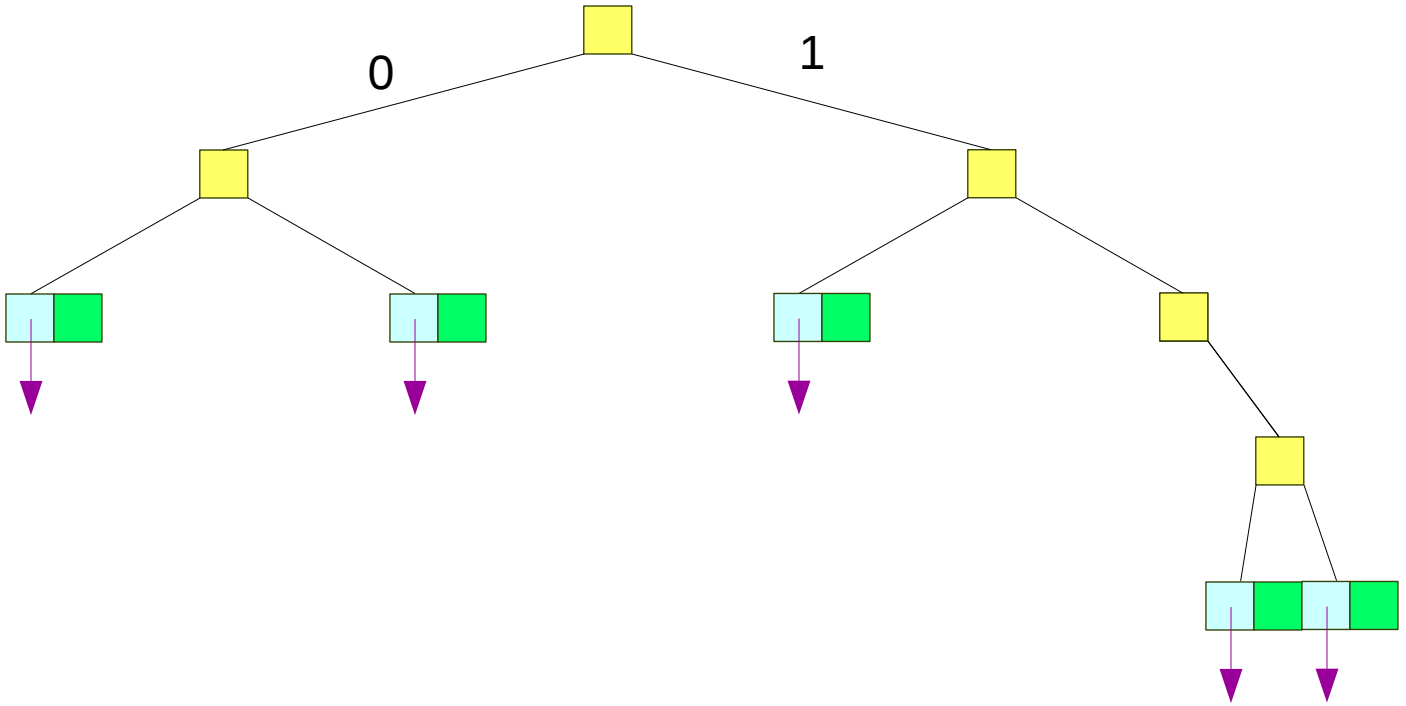
Hash Map – Hash Trie

Eliminate empty buckets and paths



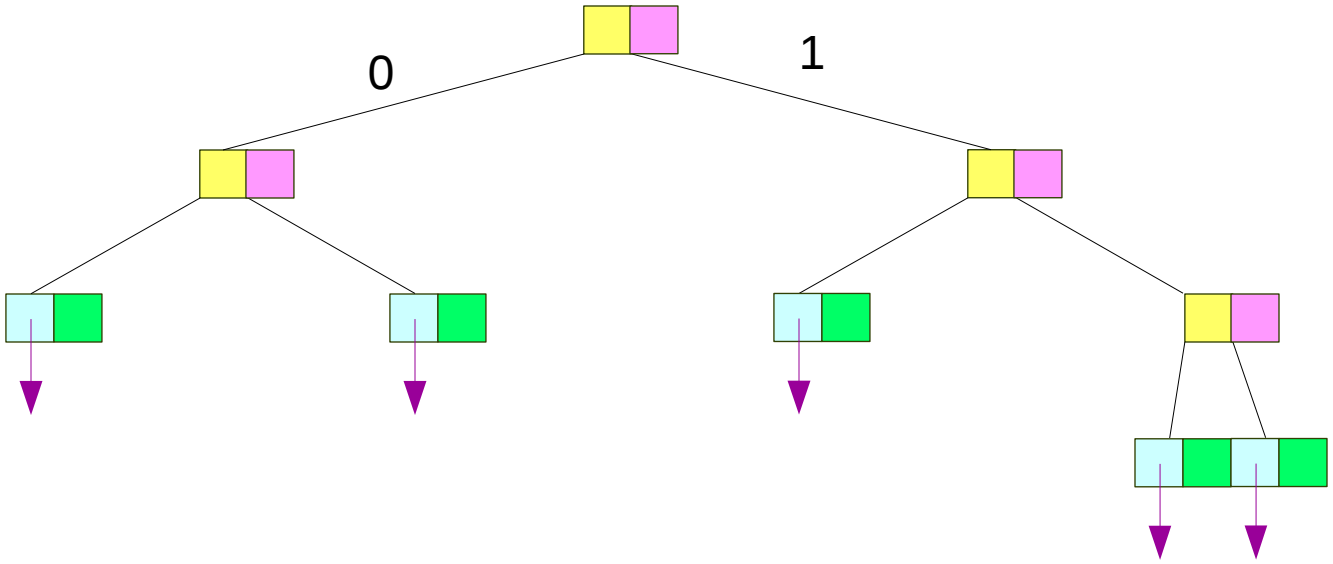
Hash Map – Hash Trie

Store hashes – shorten paths w/o branches



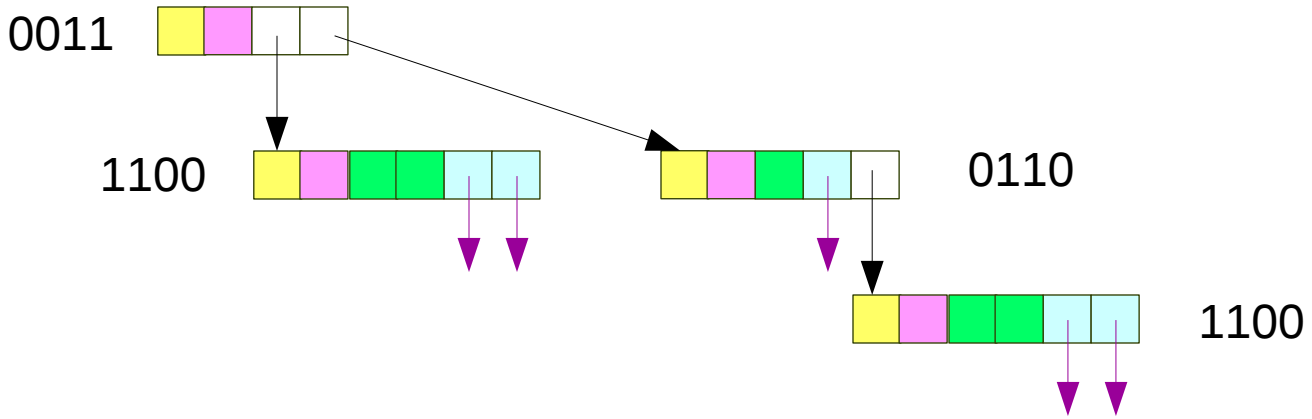
Hash Map – Hash Trie

Store node IDs – shorten paths w/o branches



Hash Array-Map Trie (HAMT)

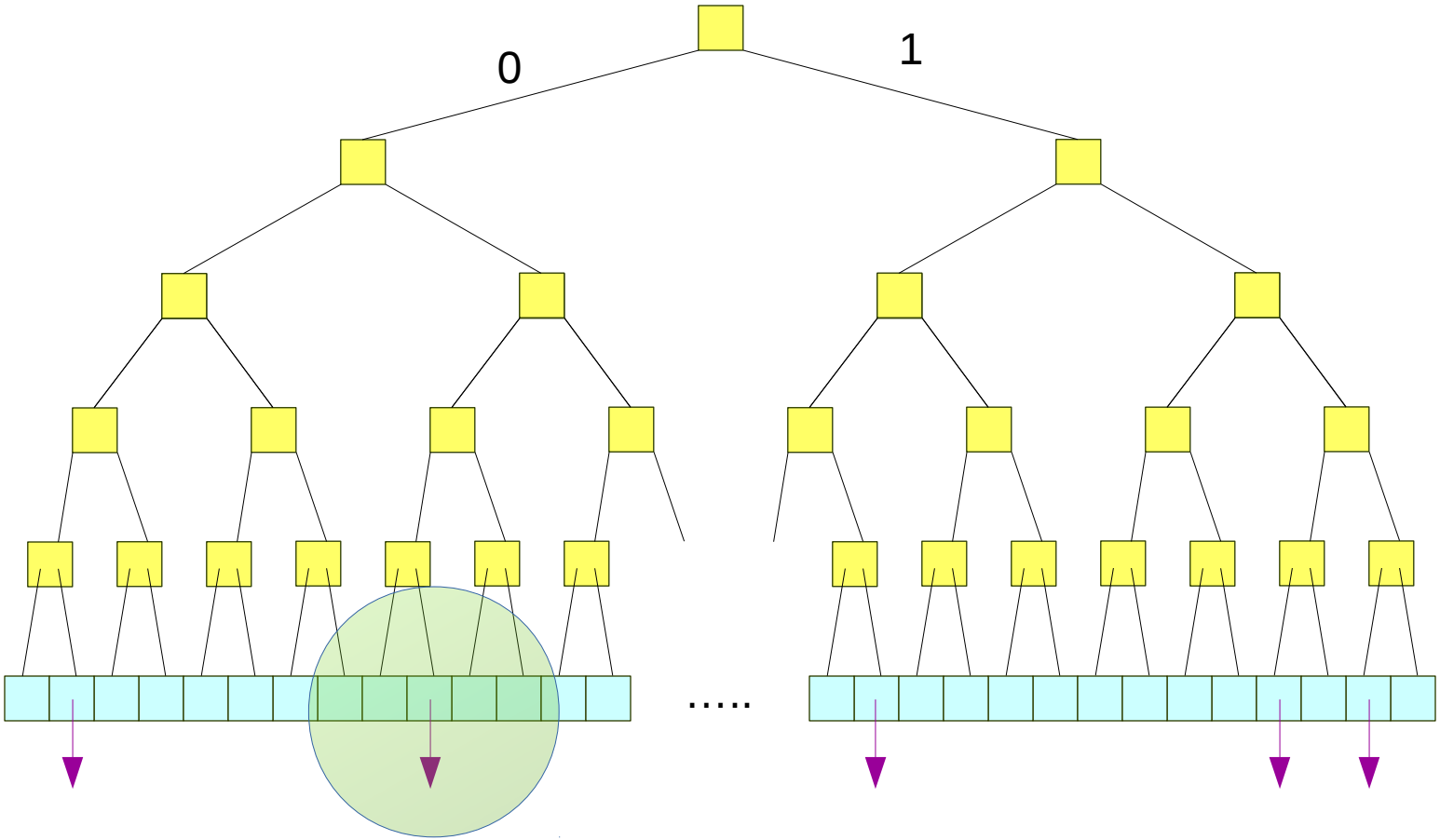
Structure nodes as array maps



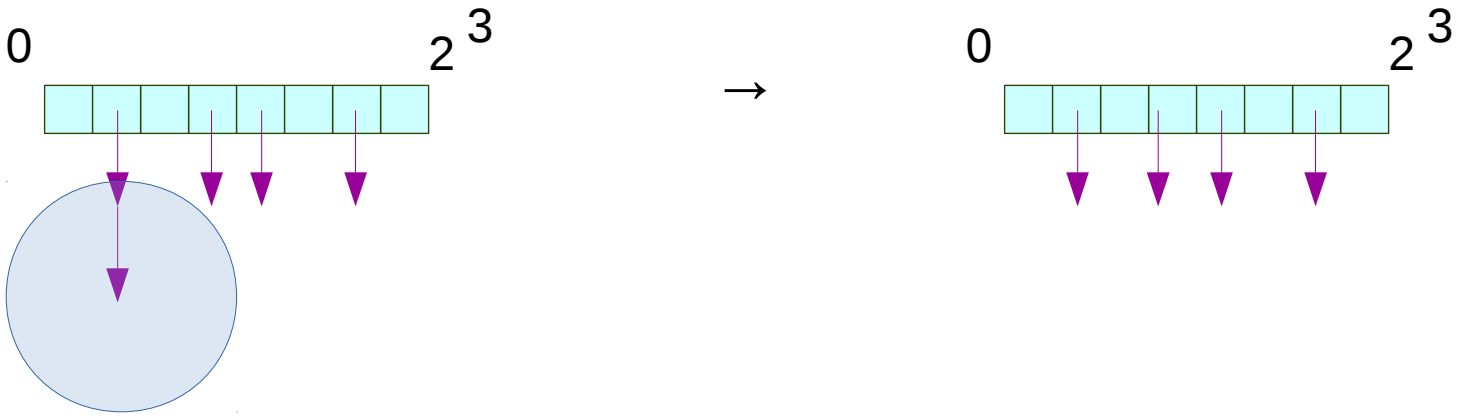
Array Map Encoding

- Two bits encoding bucket leaf children
  - Bit n is set → child n is a bucket
    - Hash and leaf pointer are stored in array
- Two bits encoding subnode children
  - Bit n is set → child n is a subnode
    - Pointer to subnode is stored in array

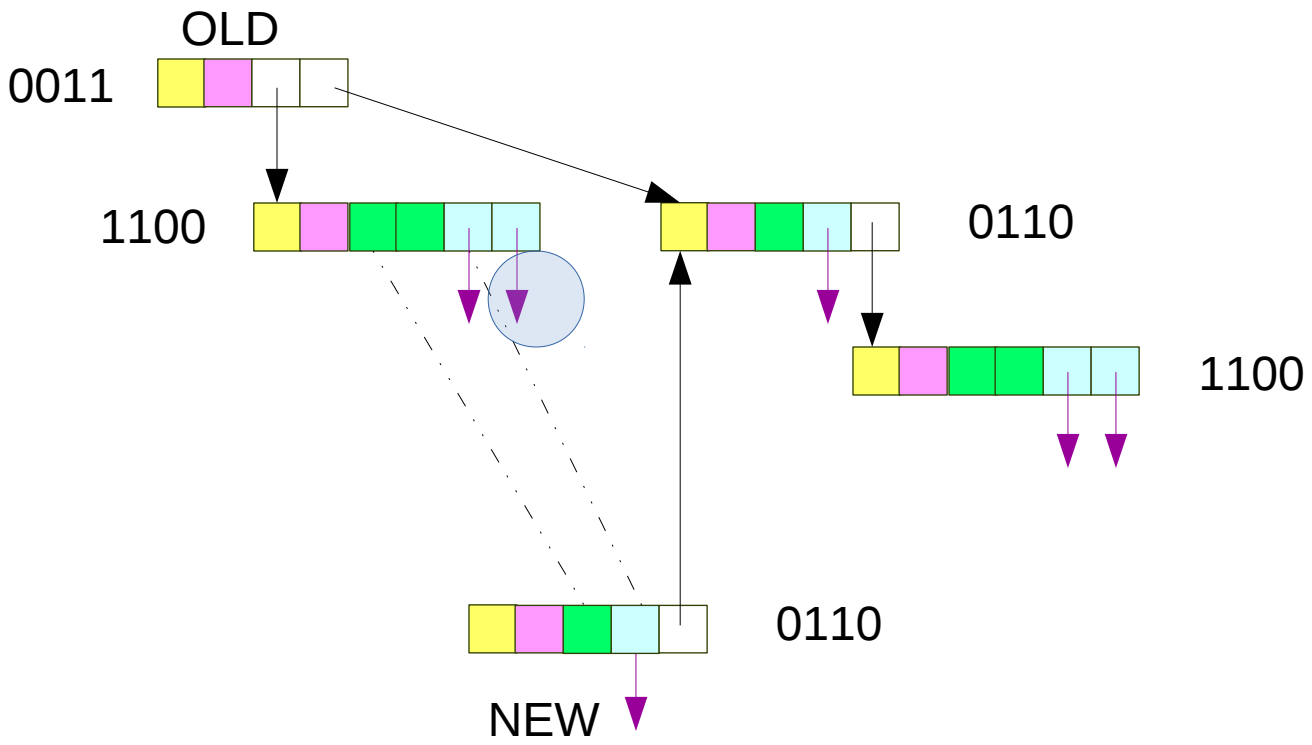
Removal Operation



Removal Operation – Tcl_HashTable (Destructive)



Removal Operation – HAMT (non-destructive)



IMMUTABILITY

- Values as Read-only structures
- Matches value semantics of Tcl
- Alternative to Copy on Write
  - CoW is a discipline to implement immutable values out of mutable foundations

...on Steroids

- Presented as binary tree
  - Two two-bit encoding maps per node
  - Easy to draw and explain
  - Inessential
- Implemented as 64-ary tree
  - Two 64-bit encoding maps per node
  - Shallow, wide trees → few hops in lookup
  - Depth of 11 covers entire 16 exbibyte capacity

Demo: `dict` vs `hamt`

```
% set data [lmap _ [lrepeat 20000 {}] tcl::mathfunc::rand]
% set d [dict create {*}$data]
% time {foreach {k v} $data {set d [dict remove $d $k]}}
-> 23839420 microseconds per iteration

% set h [hamt create {*}$data]
% time {foreach {k v} $data {set h [hamt remove $h $k]}}
-> 77113 microseconds per iteration

% set d [dict create {*}$data]
% time {foreach {k v} $data {dict unset d $k}}
-> 28610 microseconds per iteration
```

The Enemy

Merge Demo

```
% time {set d [dict merge $d1 $d2]}
→ 681783 microseconds per iteration

% time {dict merge $d $d}
→ 1032838 microseconds per iteration

% time {dict merge $d $d1}
→ 927085 microseconds per iteration

% time {set h [hamt merge $h1 $h2]}
→ 294936 microseconds per iteration

% time {hamt merge $h $h}
→ 65 microseconds per iteration

% time {hamt merge $h $h1}
 → 218641 microseconds per iteration
```

More `dict` vs `hamt`

- For one hashmap, `hamt` uses more memory.
- For set of related hashmaps, will use less.
- Operation speeds are competitive. (oom)
- Avoids copy catastrophe by design
- Still prototype quality
  - Known improvement avenues
- Immutability benefits...

Immutable Hashmap Benefits

- Read-only values share easily
  - Think “threads”
- Keep useful checkpoints
  - Think built-in command set of an interp.
- Controlled teardowns
  - Think namespace delete
- Caching and Epochs
  - No epoch for something that does not change
- Scaling?

How can I try it?

- Branch dgp-refactor in the Tcl fossil repository.
  - https://core.tcl.tk/tcl
- [hamt info] reports interesting details.
- Comments welcome.

Relaxed Radix Balance (RRB) Tree

- HAMT : Hashmap :: RRB : Sequence
  - Think “list”
  - Think “string” (list of characters)
- Foundation of the Clojure Vector
- Stay Tuned!

Conclusions

- Protoype HAMT implementation underway
  - Basic functions complete.
- Initial testing shows promise
  - Not yet a clear failure.
- Immutable structures are useful tools.
- Other immutable structure opportunities.
- Further work is needed.