Data Structures

the code that isn’t there

“A data structure is just a
stupid programming language.”

- Ralph William Gosper Jr.

“A data structure is just a
~~stupid programming language~~
tiny virtual machine.”

data structures

Fundamentals:

linked list
array
hash table
binary tree

data structures

using list for a set?

ruby's require, circa 1.9.2

bit.ly/krkvJP

FAIL

"The cheapest, fastest, and most reliable components are **those that aren't there**."

- Gordon Bell

...set the path of least resistance.

implementation details
bubble up to the surface

data structures

git vs. hg

data structures

internal data formats

data structures

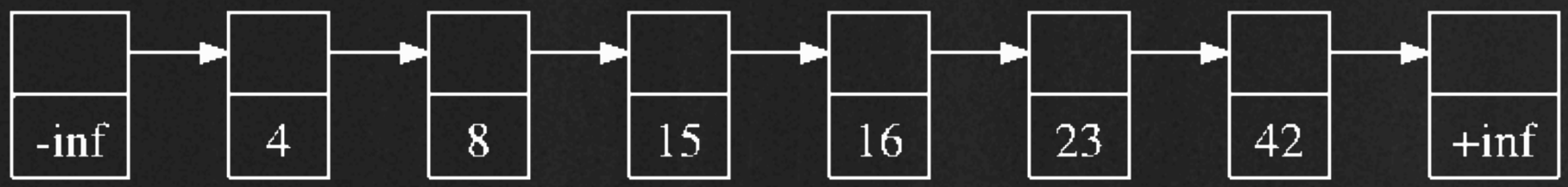
jarring segue

but, enough about the basics...

skiplists

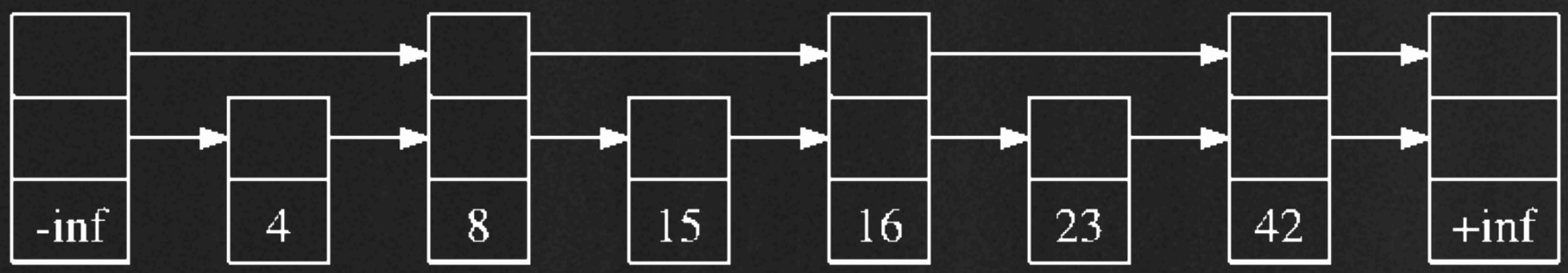
skiplists

first, take an ordered linked list...

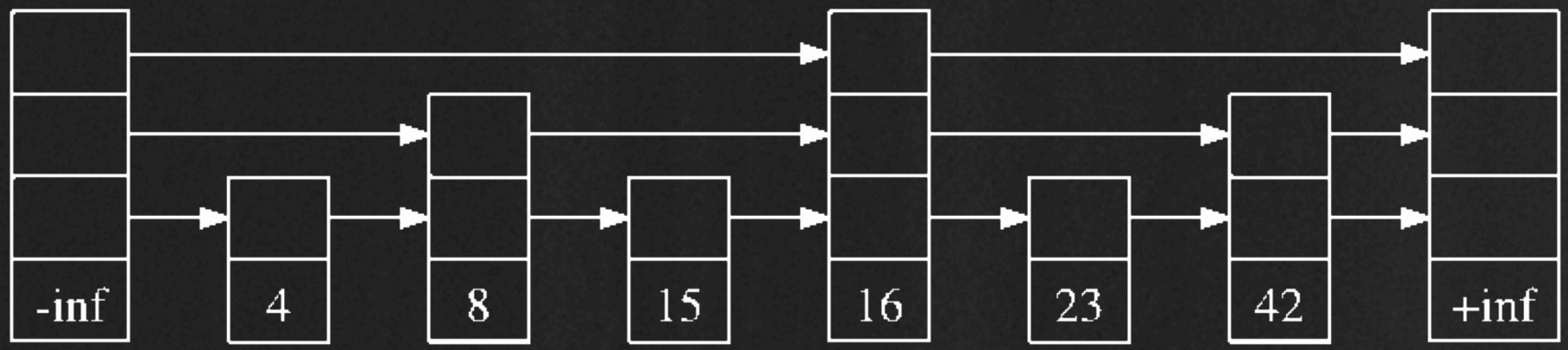


skiplists

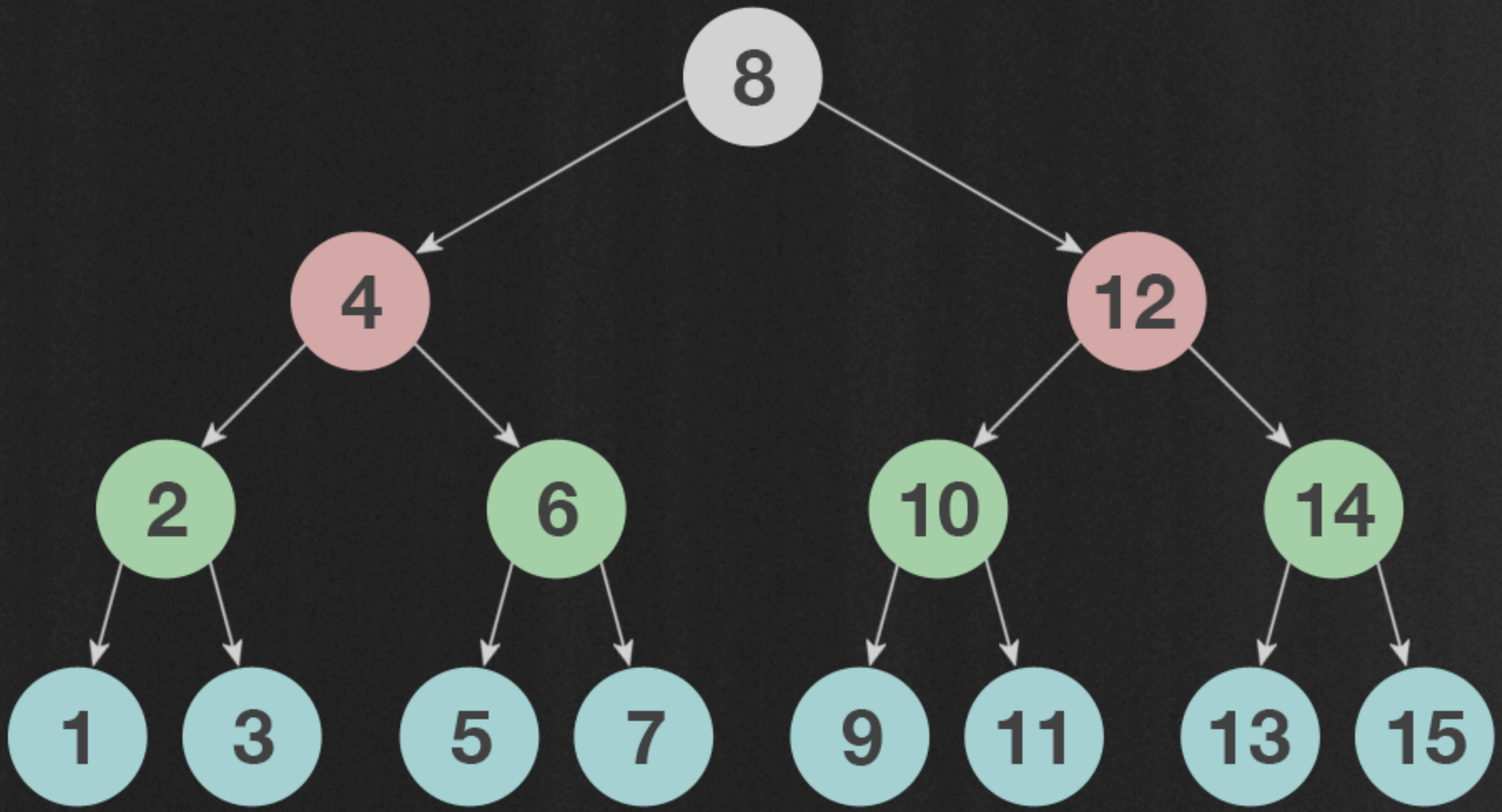
add an “express lane”...



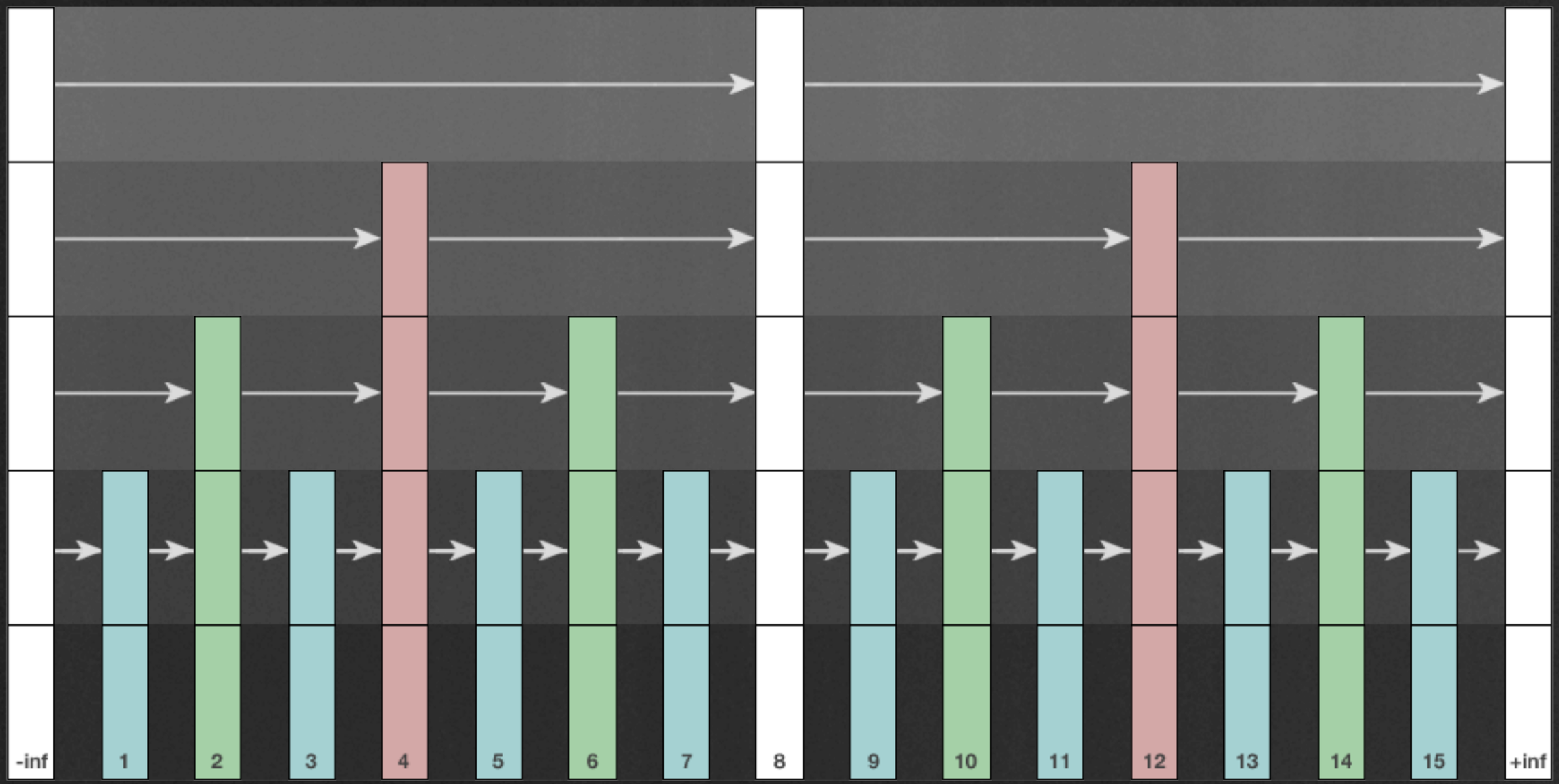
add another...



skiplists



skiplists



but, how do we balance that?

real trees are not balanced,

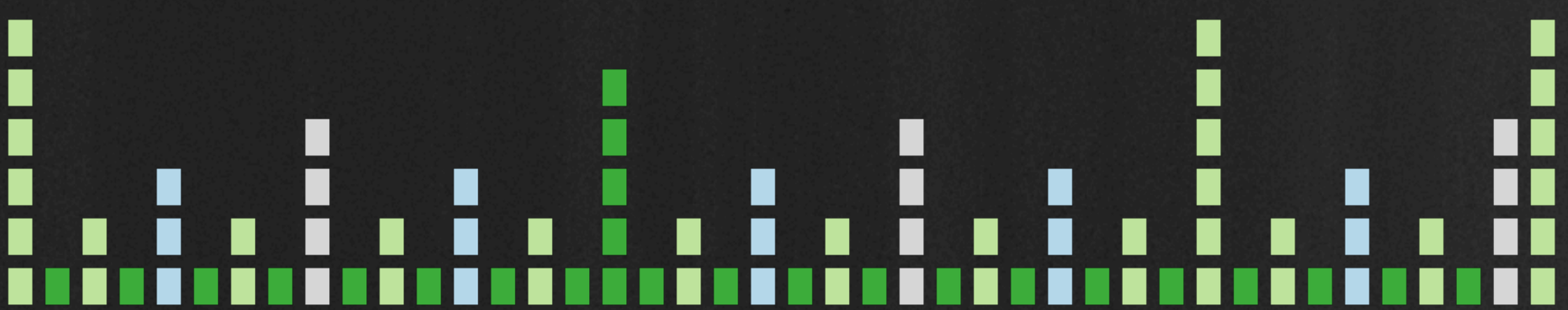
just balanced enough

skiplists

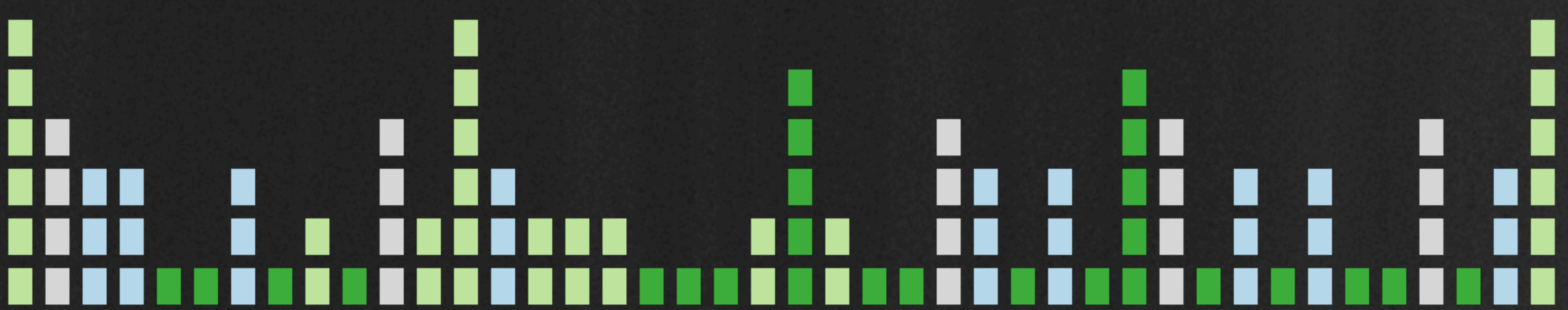
random levels

probability distribution

skiplists



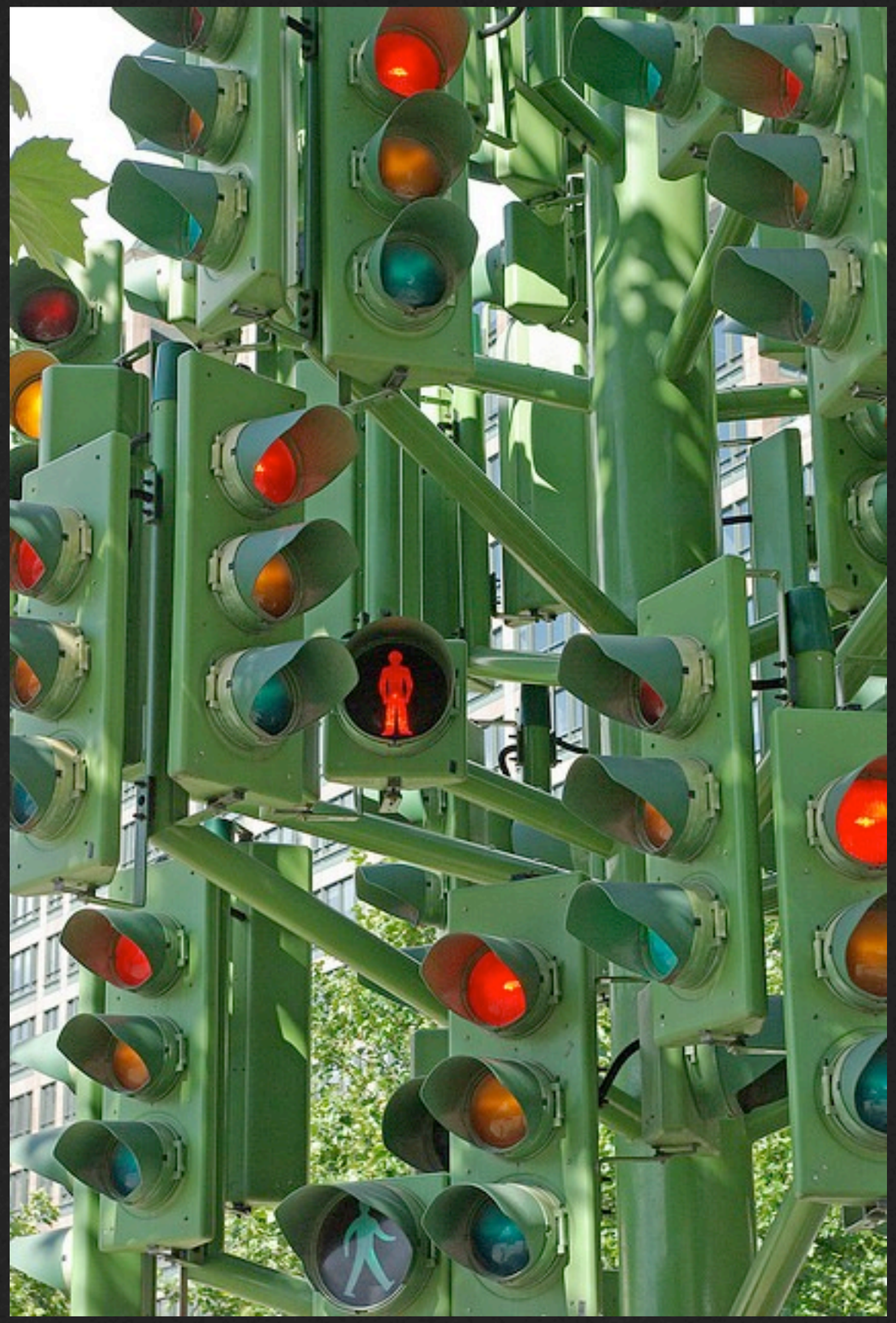
skiplists



controlled trade-offs

philosophy

“roundabouts, not traffic lights”



philosophy

traffic light

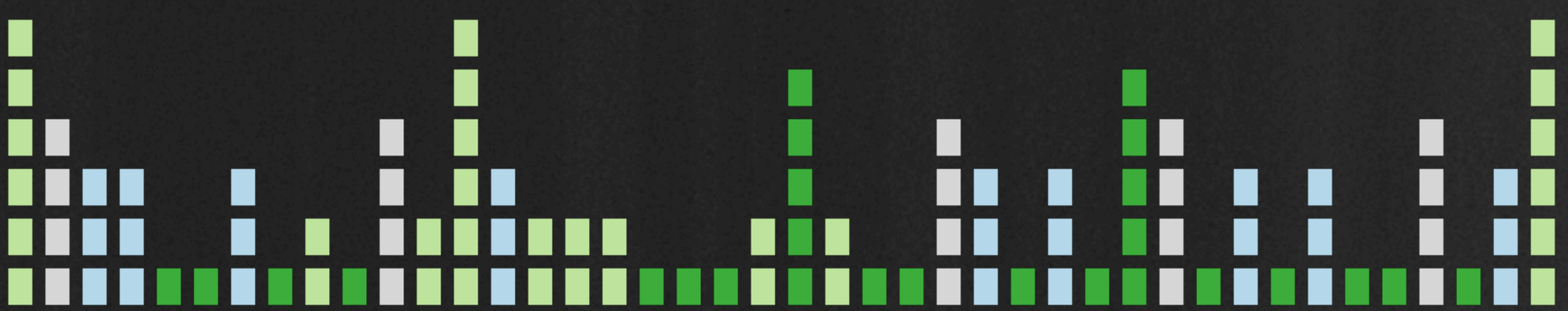
roundabout

global order emerges
from local decisions

philosophy

only immediate neighbors are affected

most locking is local, little contention



skipping on

still: mutation?

skipping on

still: mutation?

isn’t Rich Hickey in the building?

difference lists

difference lists

```
?- uses(prolog, Person).

no
```

difference lists

retroactive
immutability!

Unification

```
?- [X, Y, X] = [1, 2, Z].
```

```
X = 1
Y = 2
Z = 1

yes
```

difference lists

data structures with “holes” in them

difference lists

logic variables:

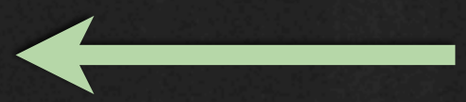
explicitly modeling partial information

difference lists

Unification

```
?- [X, Y, H] = [1, 2, Z].
```

```
X = 1
Y = 2
Z = H
```



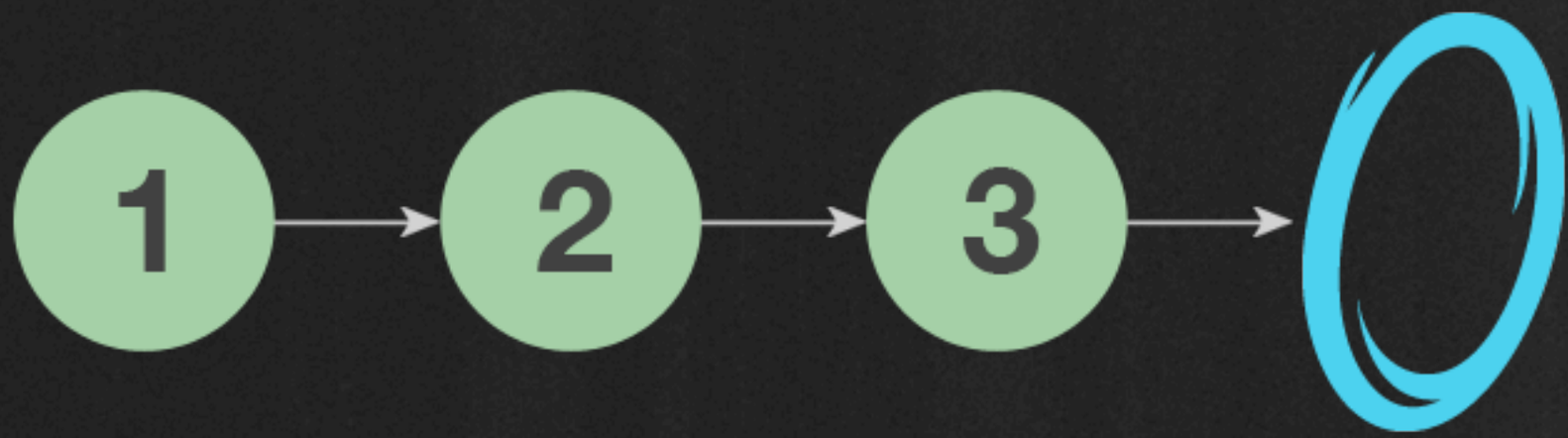
???

```
yes
```

difference lists

constant-time append
to an immutable list

difference lists



difference lists

```
?- A = [1,2|B],
     B = [3|C]
```

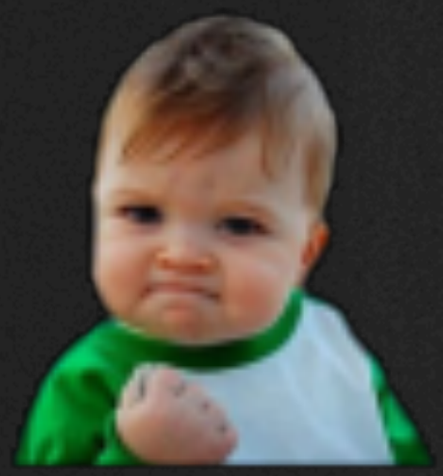
```
A = [1, 2, 3|C]

yes
```

difference lists

```
?- A = [1,2|B],
     B = [3|C]
```

```
A = [1, 2, 3|C]
```

yes



Yes!

difference lists

still immutable

...just unstuck in time

difference lists

more fundamental than lazy evaluation or lazy streams

(a bit closer to futures/promises)

difference lists

...apply to more than just lists

difference trees?
difference dictionaries?

rolling along...

rolling hash

find matching/overlapping
sequences in binary data

rolling hash

hash everything against everything?

md5, sha1, etc. are **too slow**

rolling hash

“Fast, Cheap, and Able to Roll”

rolling hash

```
The_quick_brow     0xba5eba11
 he_quick_brown    0x0b5e55ed
  e_quick_brown_   0xdeadbeef
   _quick_brown_f  0xf005ba11
    quick_brown_fo 0xbea2ded1
     uick_brown_fox 0xdefea7ed
```

rolling hash

fill then window buffer, then:

drop 1, take 1, new hash
repeat

rolling hash

rsync

rolling hash

sync data across slow network

minimize passes & bandwidth

rolling hash

just send the hash for each block, eh?

S 

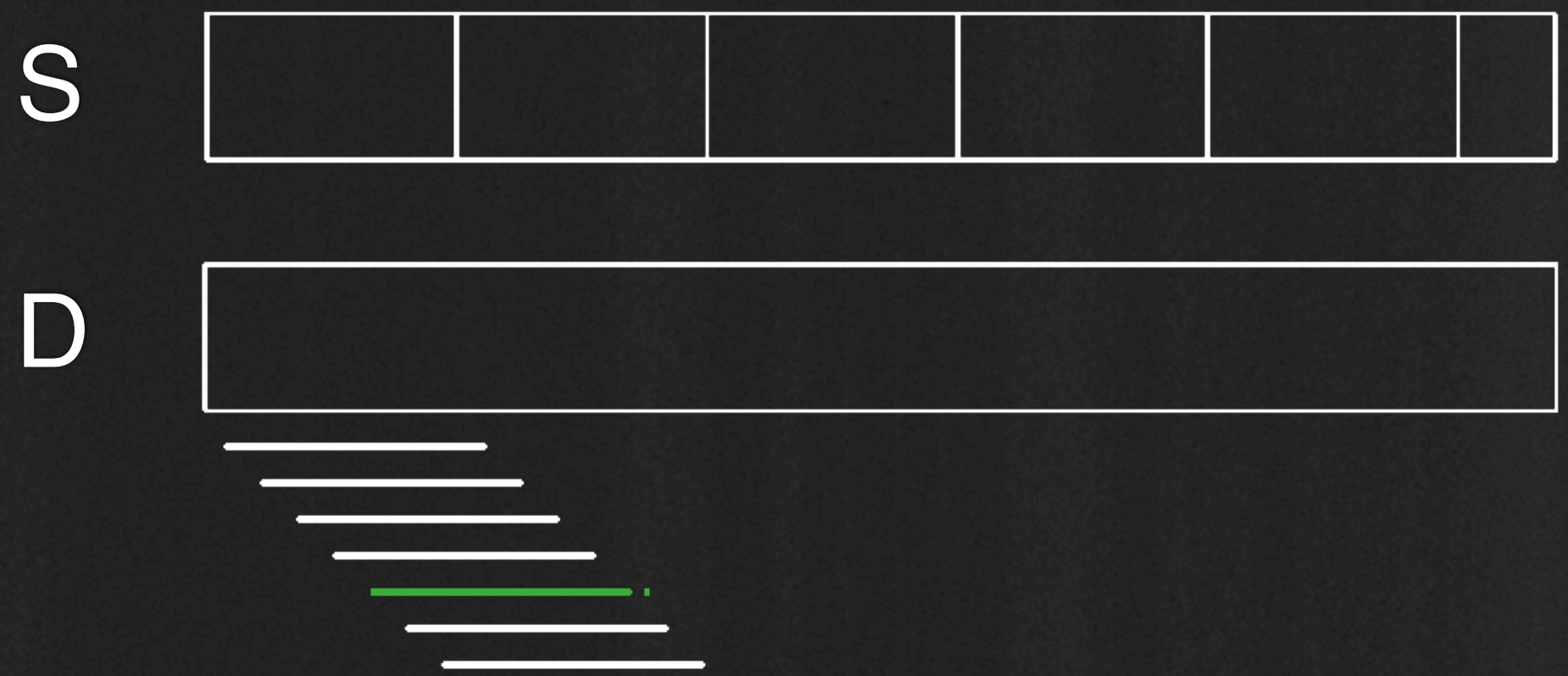
D 

not so fast:
inserts/deletes shift each block!

S 

D 

Match with a rolling hash, then sha1



request unmatched regions

S

D

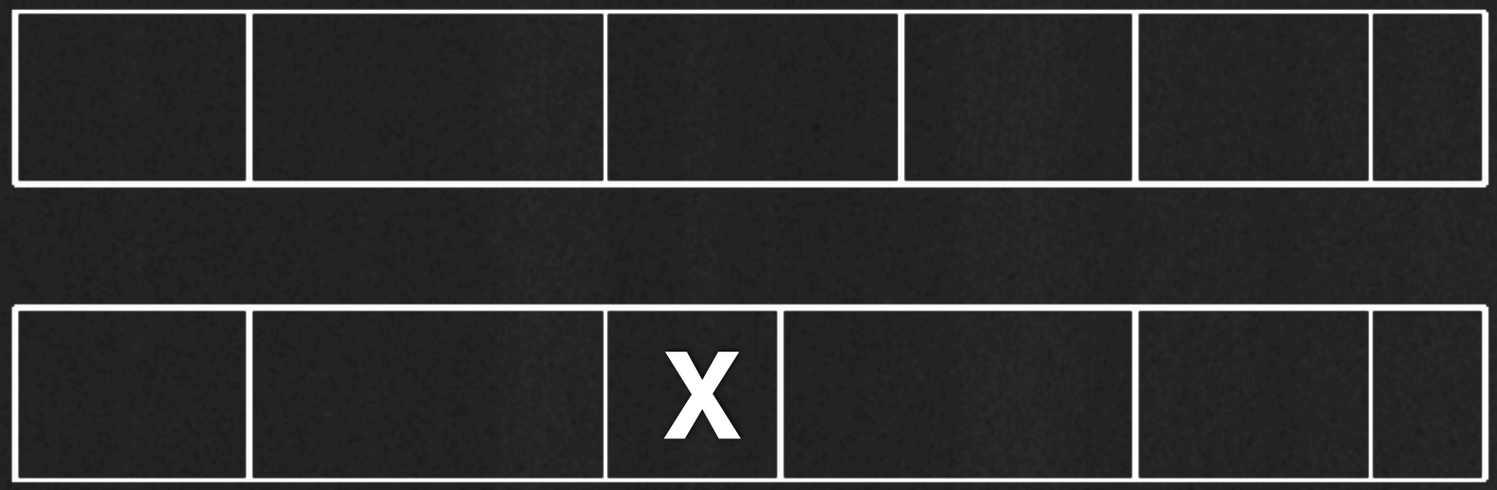
(for more details, check out Andrew Tridgell's thesis:

Efficient Algorithms for Sorting and Synchronization, pg. 64)

can also be used for chunking data

rolling hash

even with changes, falls back in sync



rolling hash

...finds deterministic breaks

...cheaply matches blocks

jumprope

jumprope

storage for large binary strings

(where “string” can mean “file”)

jumprope

“content addressable storage”

no pointers, easy distribution

jumprope

persistent and immutable

can be freely cached

jumprope

de-duplication for free

jumprope

...kind of like a git repo

three structural elements:

leaf

raw_data

limb

jumprope

trunk

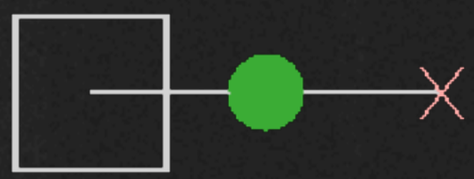
then, it just needs a key/value store

get(hash) => jr_node

set(hash, jr_node) => OK | ERROR

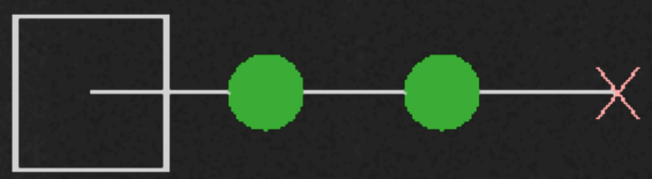
jumprope

building it up



jumprope

add another

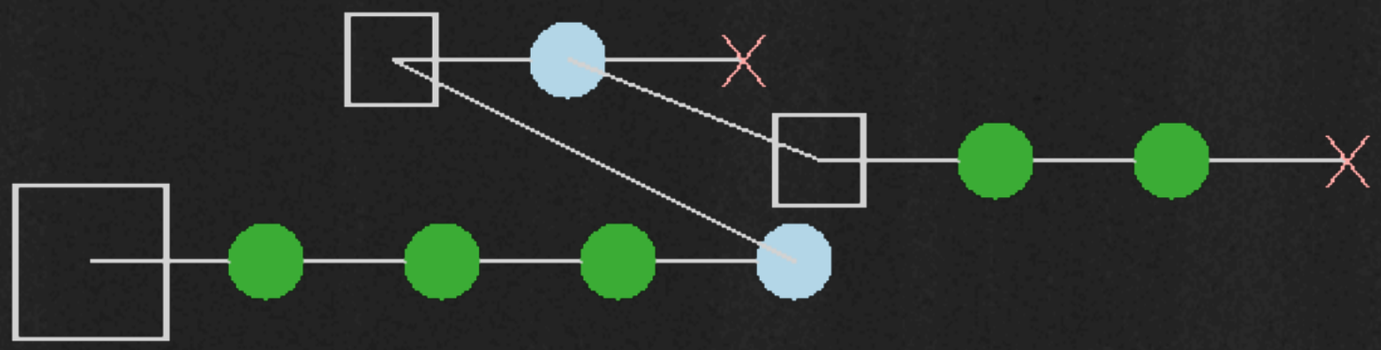


jumprope

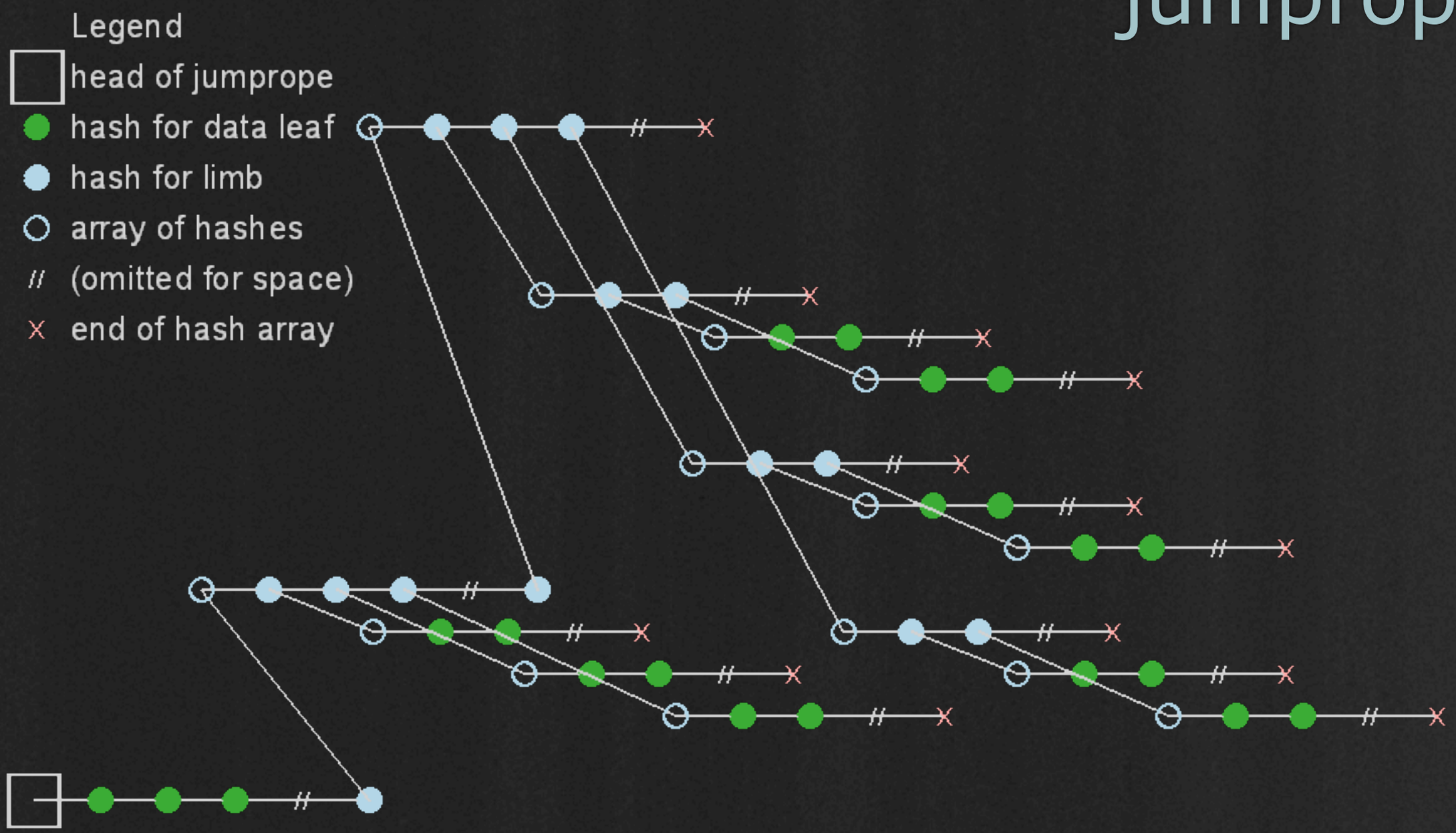
and another

jumprope

grow a level, then add a limb



jumprope


Legend
head of jumprope
hash for data leaf
hash for limb
array of hashes
// (omitted for space)
end of hash array

seek into it

jumprope

duplicated nodes are shared

free de-duplication

jumprope

low metadata overhead

jumprope

fetch it (in parallel)

stream it

mirror it

jumprope

use case?

scatterbrain

properties

randomization

persistence & immutability

emergence from local behavior

lack of pointers (CAS)

We're Hiring!

cc attribution

the php hammer:
http://www.flickr.com/photos/raindrift/sets/72157629492908038/

jumprope:
http://www.flickr.com/photos/adwriter/5826718940/

traffic light:
http://www.flickr.com/photos/bexross/2636921208/

cc attribution

roundabout:
http://www.flickr.com/photos/trendscout/4000299787/

tree:
http://www.flickr.com/photos/neilspicys/2348959159/

leek:
http://www.flickr.com/photos/richardnorth/7319091956/

Questions?

Scott Vokes

@silentbicycle

github.com/silentbicycle