technische universität
dortmund

Practical Evaluation of LZ78 and LZW Tries

Johannes Fischer

Dominik Köppl

This presentation received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941.

LZ78 with LZ trie

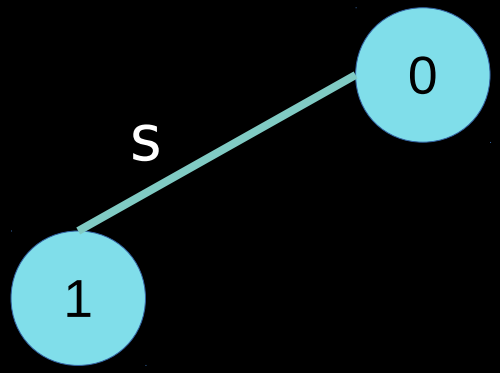
senescence

LZ78 with LZ trie

senescence

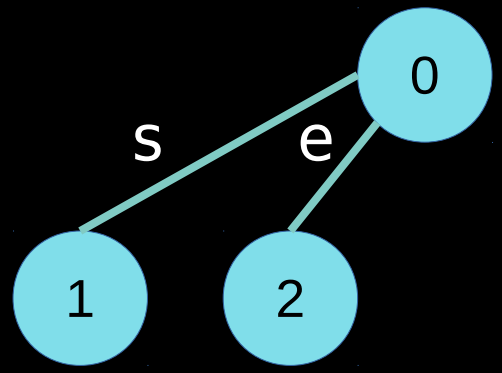
LZ78 with LZ trie

Senescence



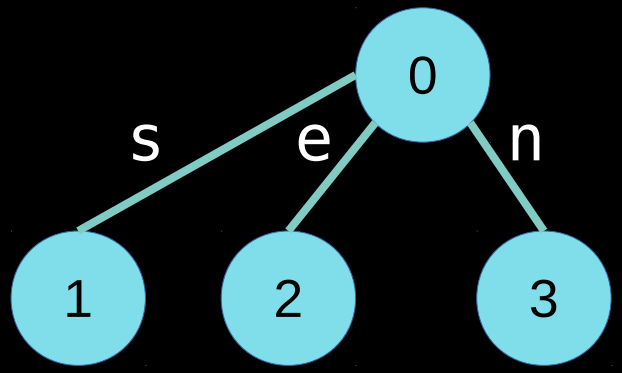
LZ78 with LZ trie

senescence



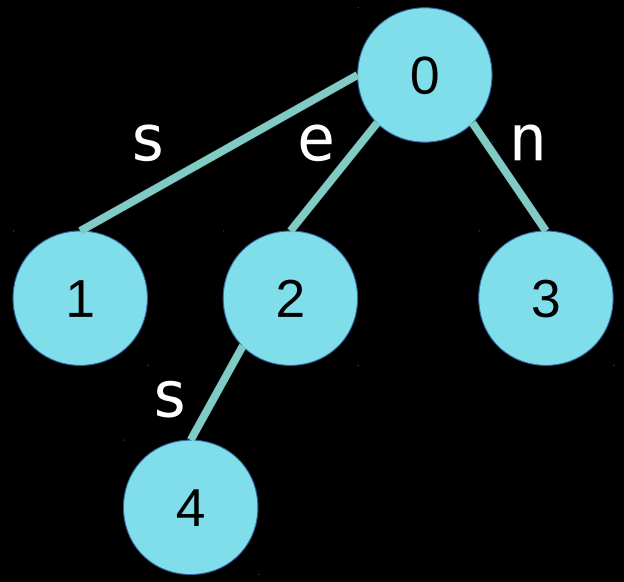
LZ78 with LZ trie

senescence



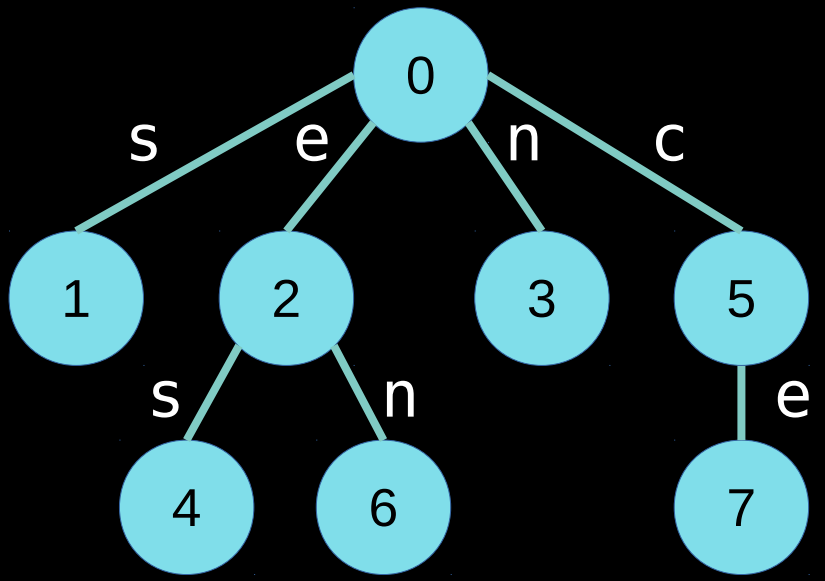
LZ78 with LZ trie

senescence



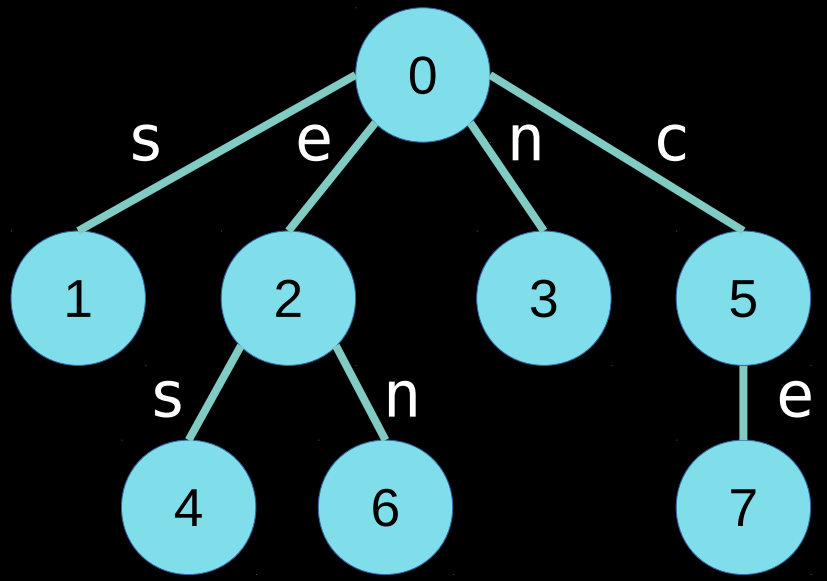
LZ78 with LZ trie

senescence



LZ78 with LZ trie

senescence



Setting

- in what time / memory build LZ trie *practically*?
- σ: alphabet size
- z: number of factors

trie representations

- binary trie [folklore]
- ternary trie [Bentley, Sedgewick'97]
- hash trie
- compact hash trie
- rolling hash trie

new

trie representations

- binary trie [folklore]
- ternary trie [Bentley, Sedgewick'97]
- hash trie
- compact hash trie
- rolling hash trie

new

all use dynamic arrays

trie representations

- binary trie [folklore]
- ternary trie [Bentley, Sedgewick'97]
- hash trie
- compact hash trie
- rolling hash trie

new

all use dynamic arrays

double space when full

trie representations

- binary trie [folklore]
- ternary trie [Bentley, Sedgewick'97]
- hash trie
- compact hash trie
- rolling hash trie

new

all use dynamic arrays

double space when full

trie representations

- binary trie [folklore]
- ternary trie [Bentley, Sedgewick'97]
- hash trie
- compact hash trie
- rolling hash trie

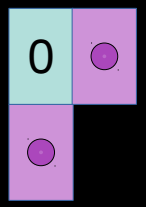
new

all use dynamic arrays

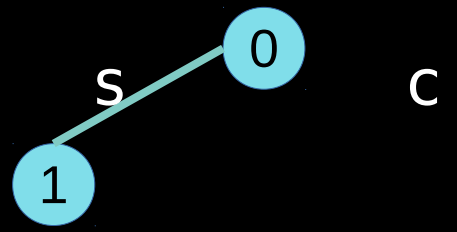
double space when full

binary trie

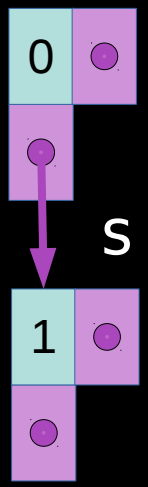
senescence



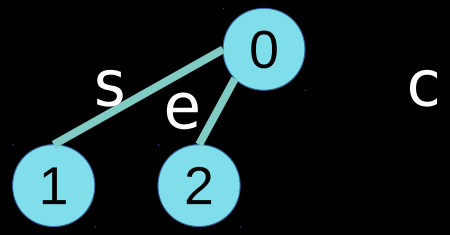
binary trie



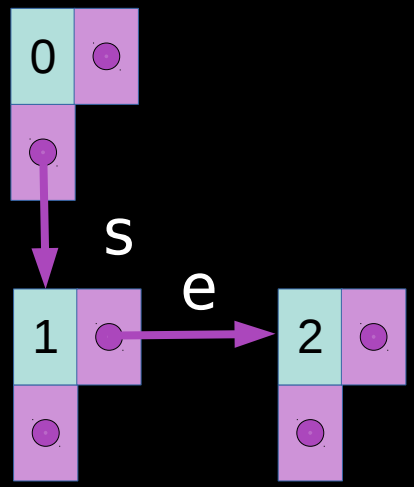
Senescence



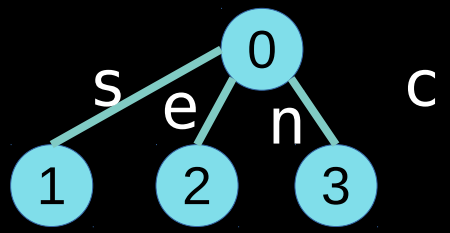
binary trie



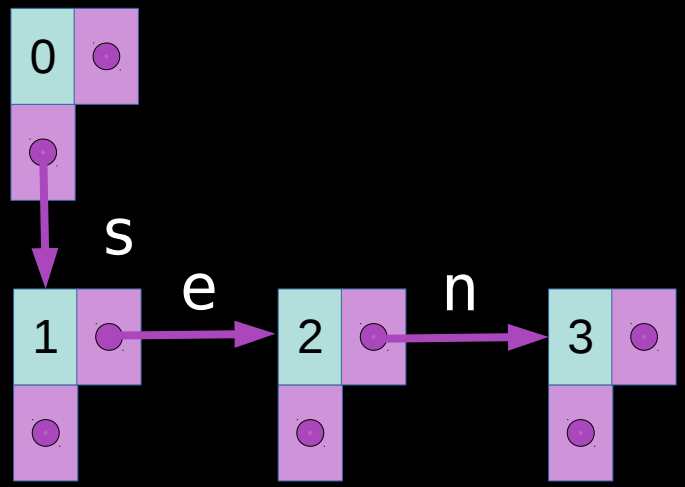
senescence



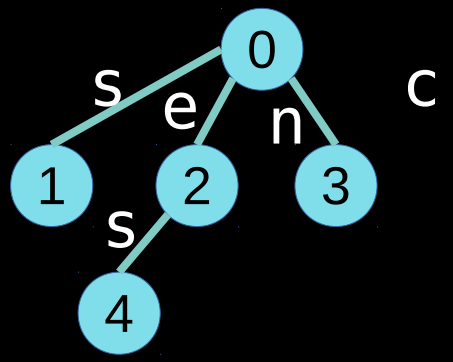
binary trie



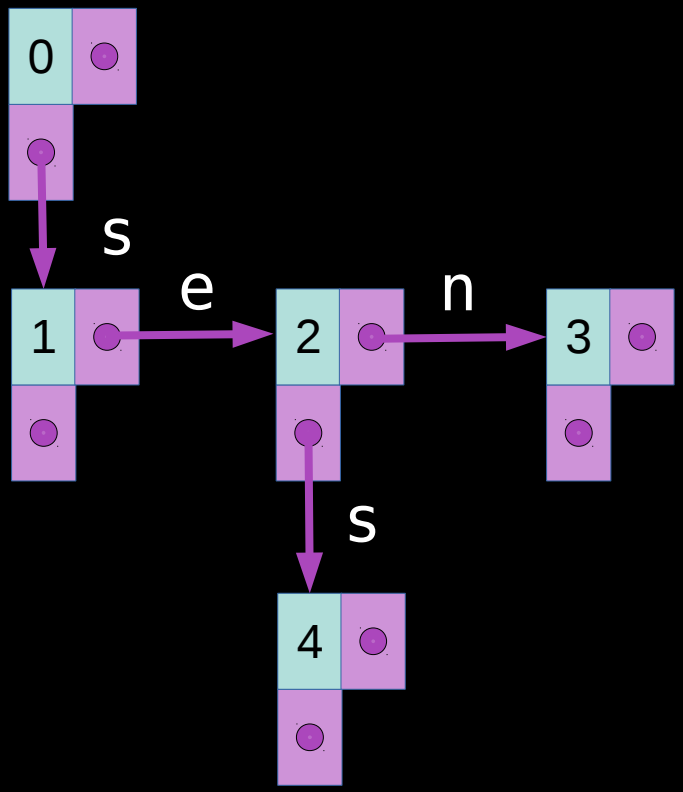
senescence



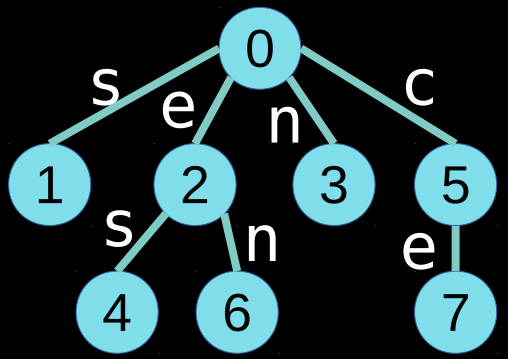
binary trie



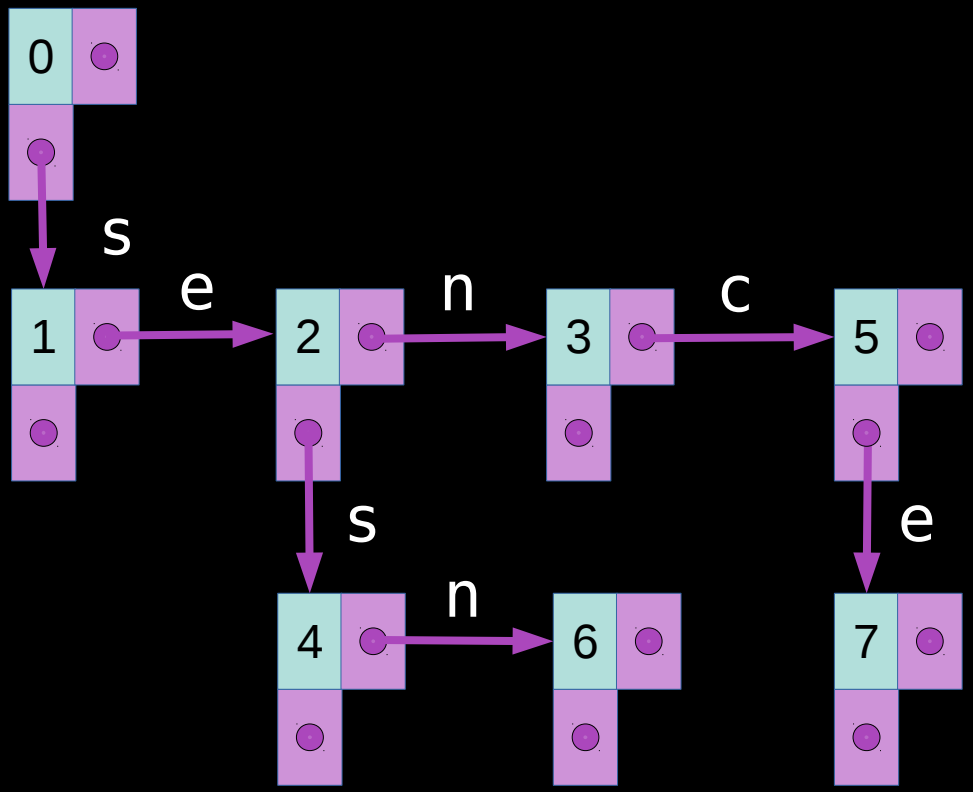
senescence



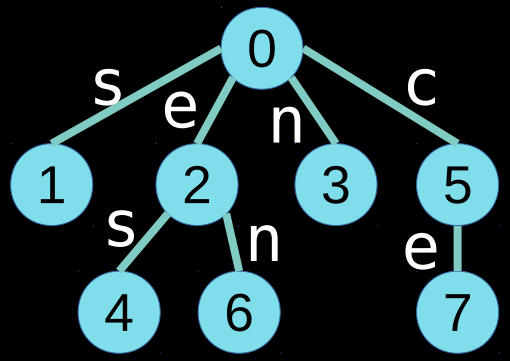
binary trie



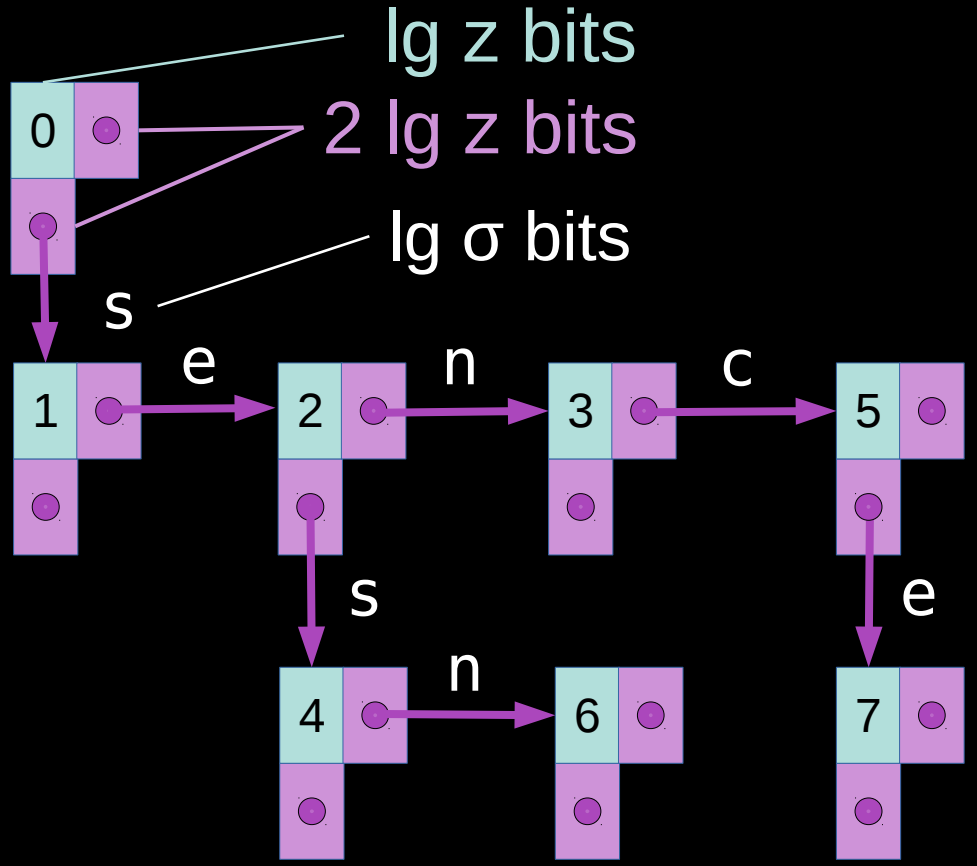
senescence



binary trie



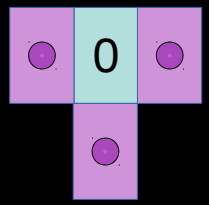
senescence



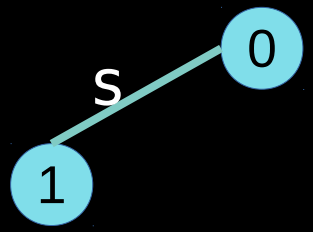
0

ternary trie

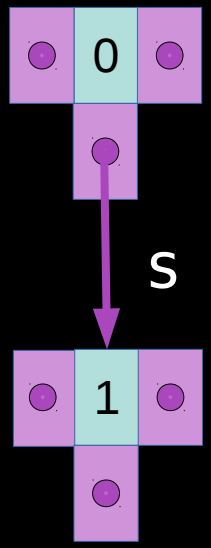
senescence



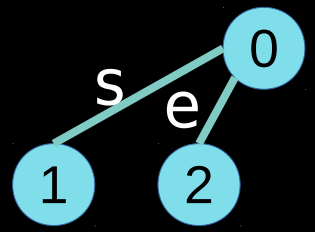
ternary trie



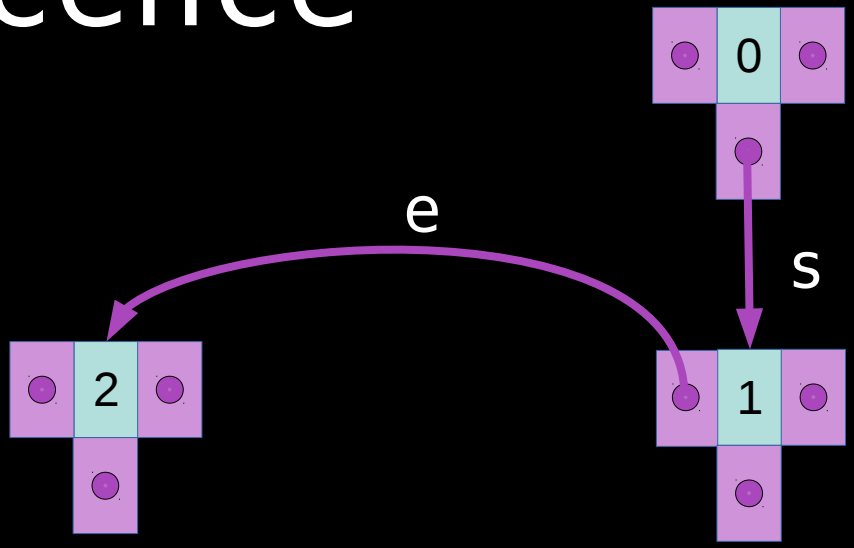
senescence



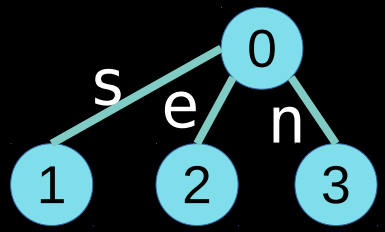
ternary trie



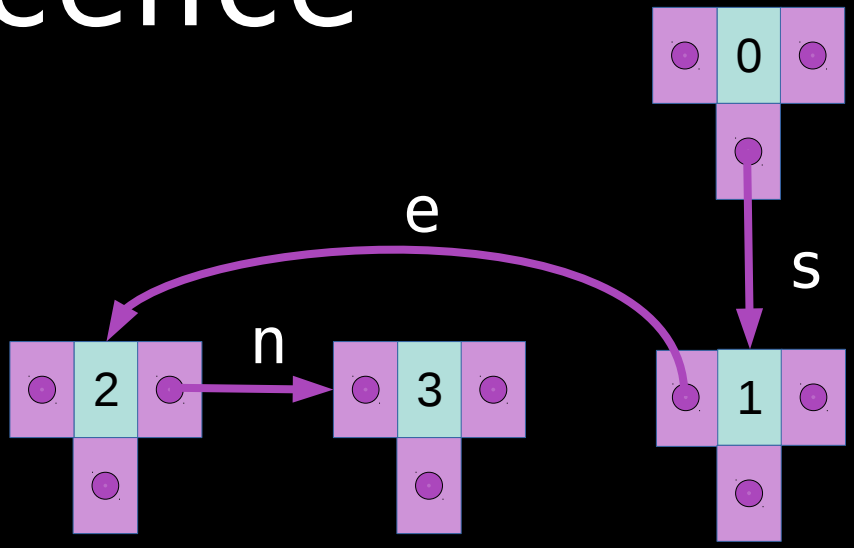
senescence



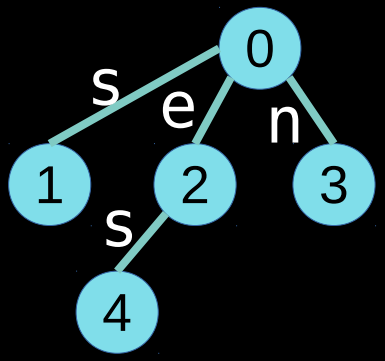
ternary trie



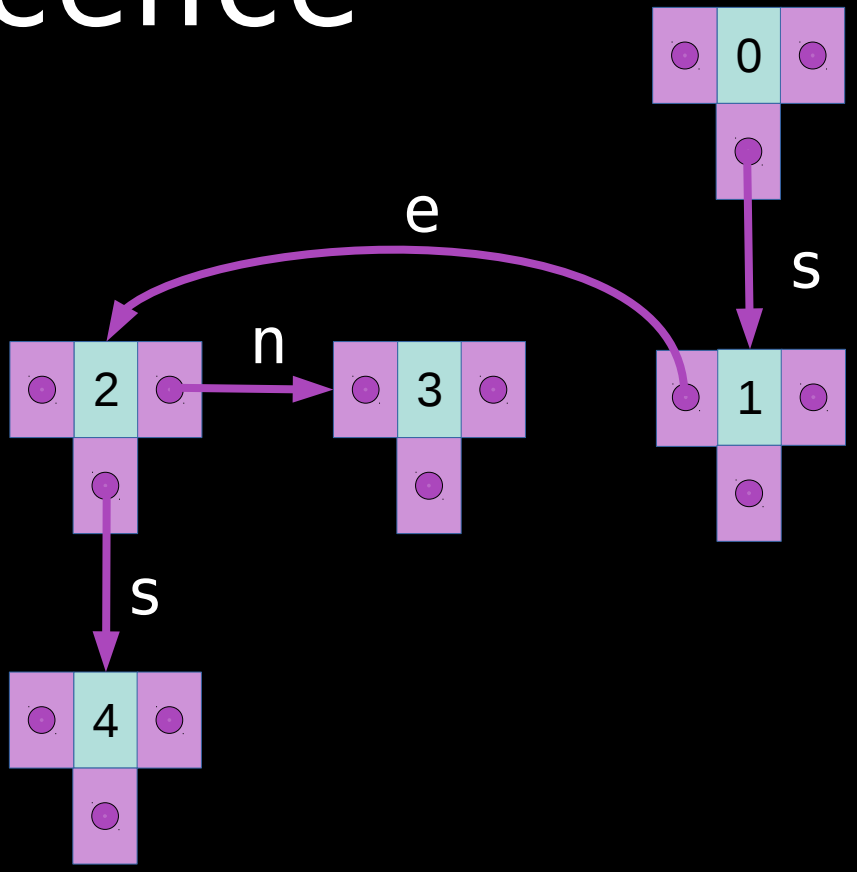
senescence



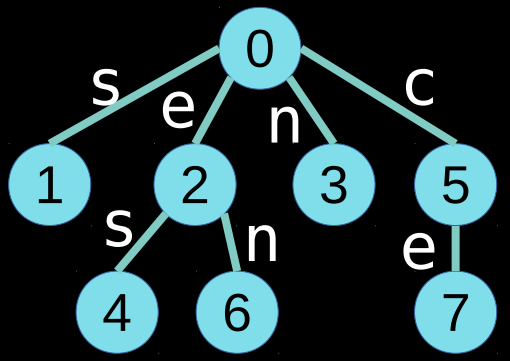
ternary trie



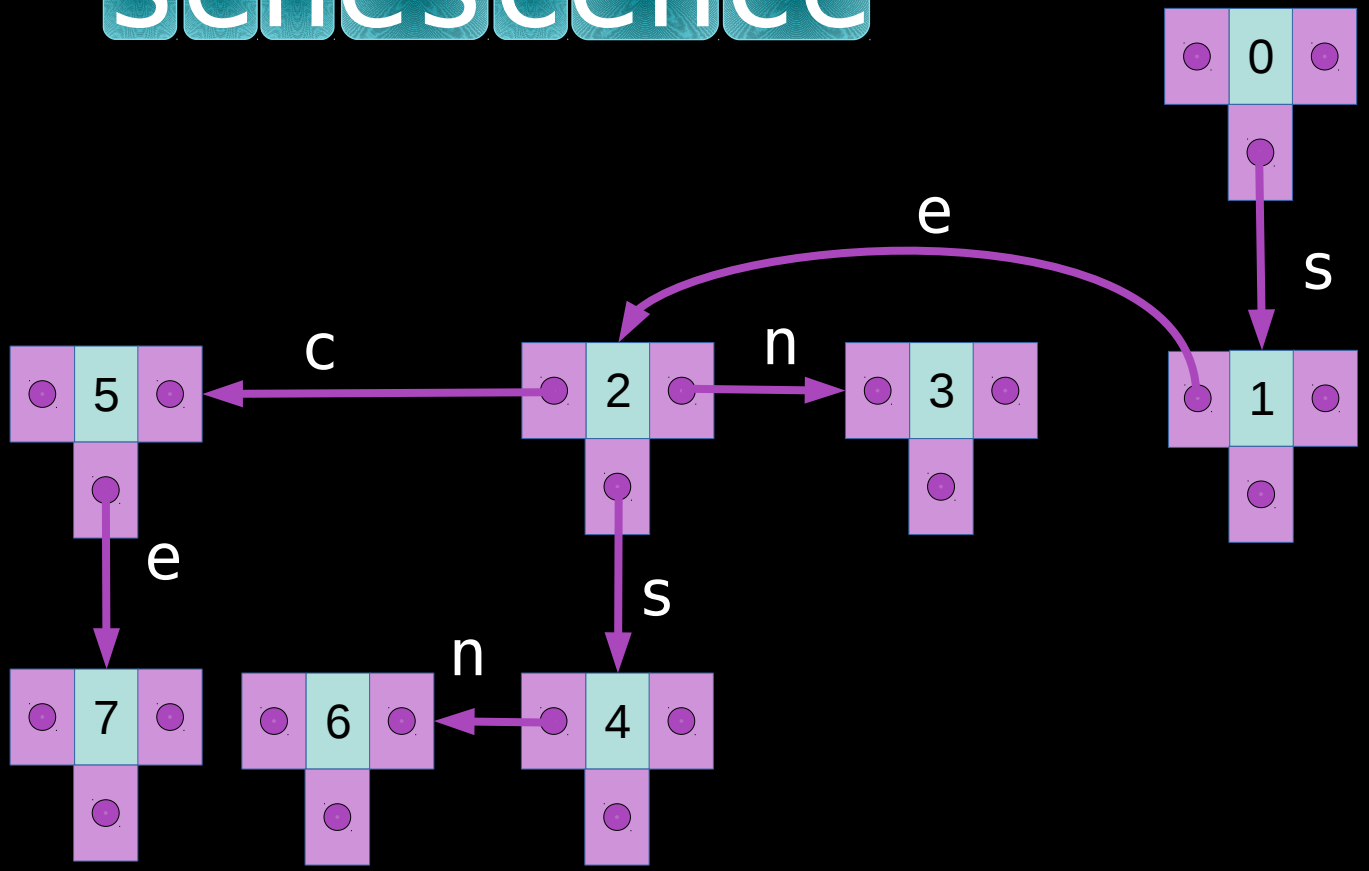
senescence



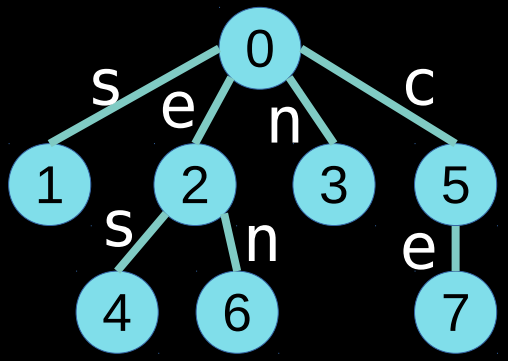
ternary trie



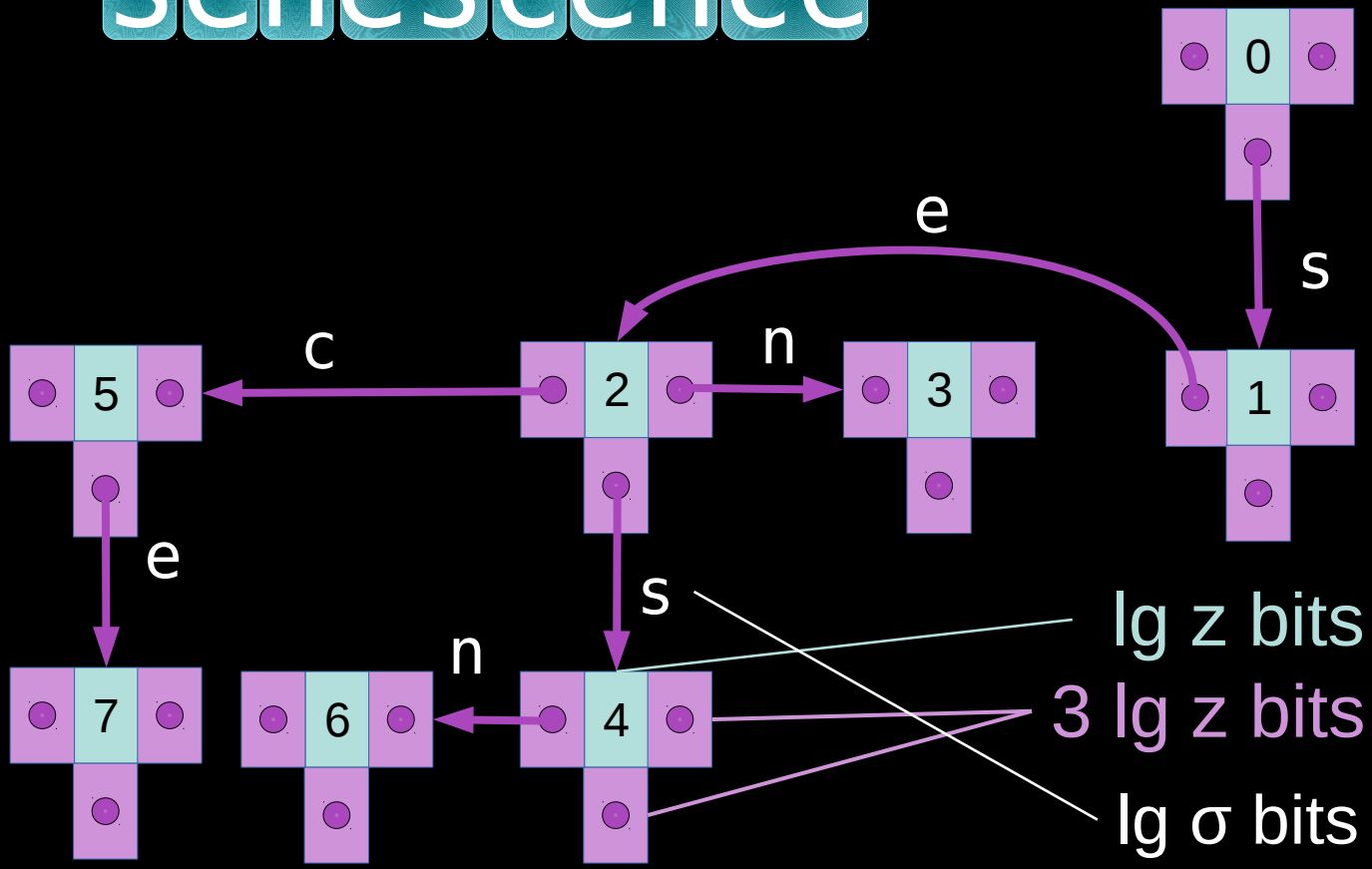
senescence



ternary trie

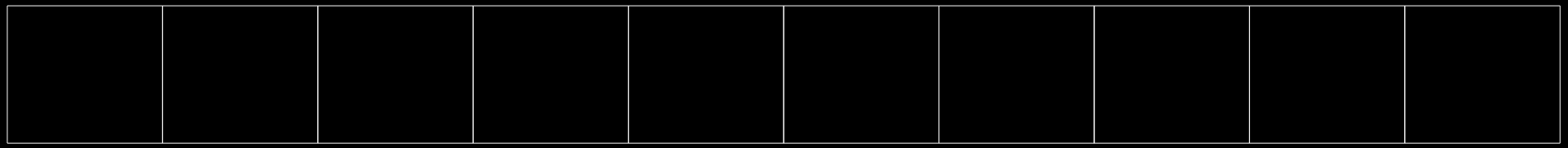


senescence



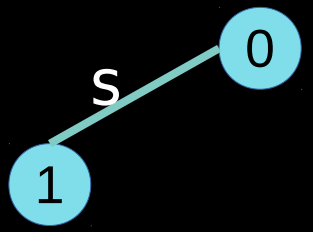
0

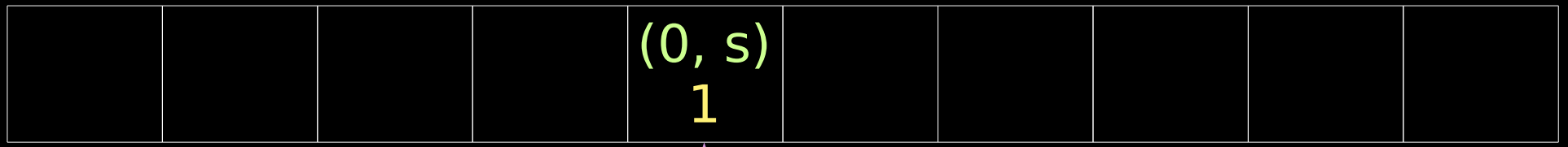
hash trie



senescence

hash trie



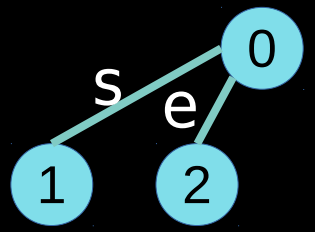


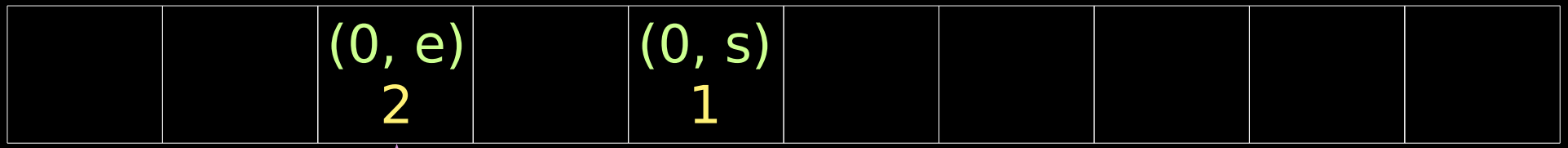
key (0, s)
value '1'

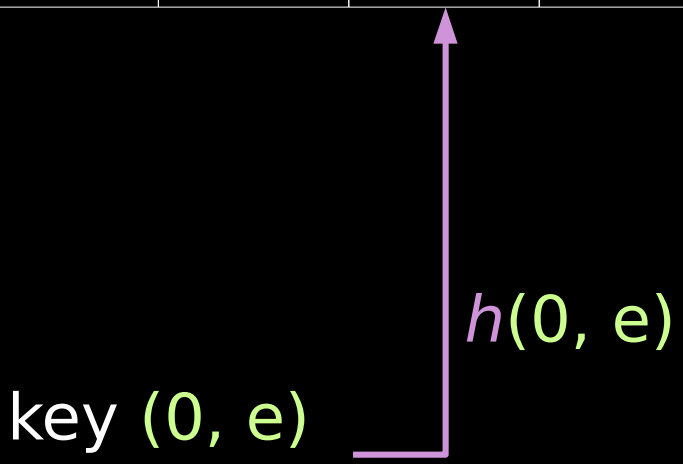
h(0, s)

senescence

hash trie

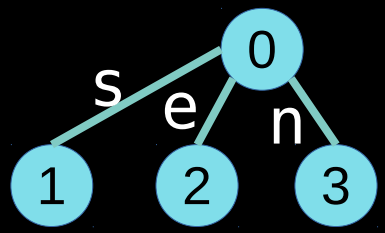






senescence

hash trie



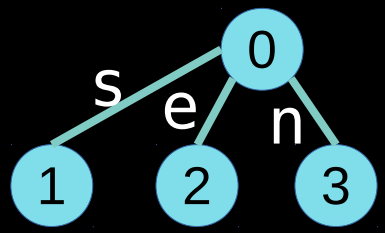
		(0, e) 2		(0, s) 1		(0, n) 3			

h(0, 2)

key (0, n)
value '3'

senescence

hash trie



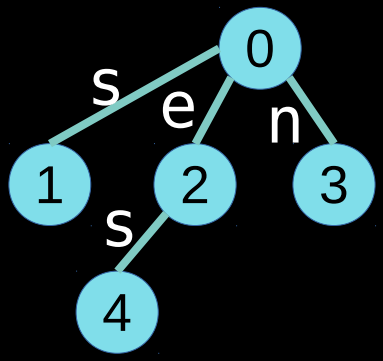
		(0, e) 2		(0, s) 1		(0, n) 3			

h(0, e)

key (0, e)
value '2'

senescence

hash trie



		(0, e) 2		(0, s) 1		(0, n) 3		(2, s) 4	

h(2, s)

key (2, s)
value '4'

senescence

key (i, s)
value j

lg z bits
lg σ bits
lg z bits

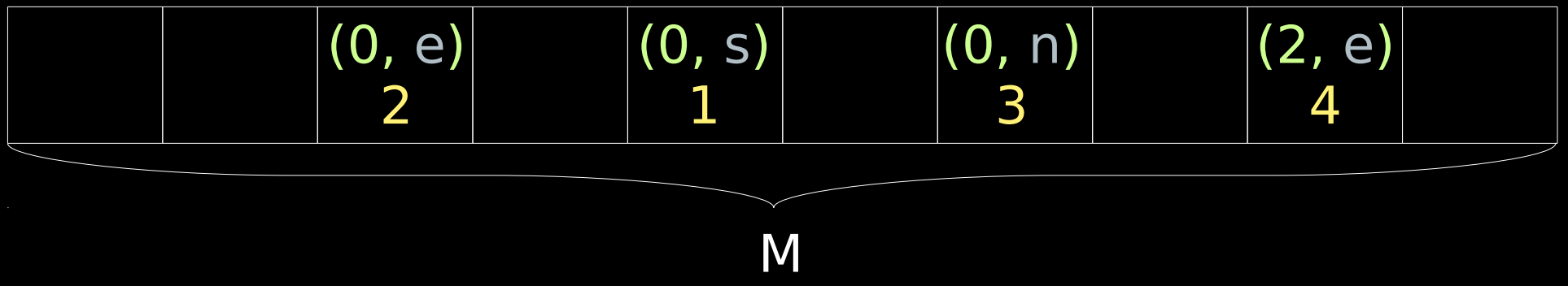
$2\lg z + \lg \sigma$ bits

		(0, e) 2		(0, s) 1		(0, n) 3		(2, e) 4	

key (i, s)
value j

lg z bits
lg σ bits
lg z bits

$2\lg z + \lg \sigma$ bits



- table size M
- load factor α
- $z \leq \alpha M$

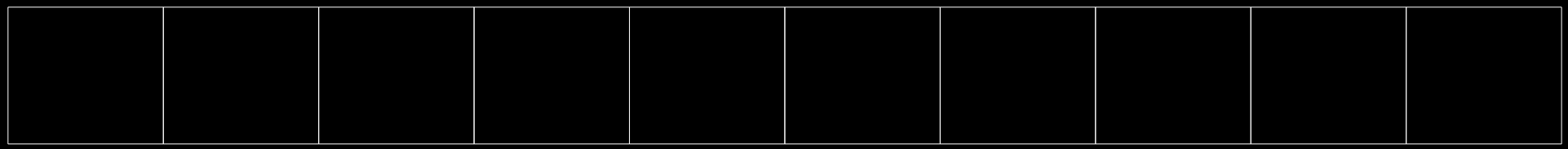
$$\geq \frac{z}{\alpha}(2\lg z + \lg \sigma) \text{ bits}$$

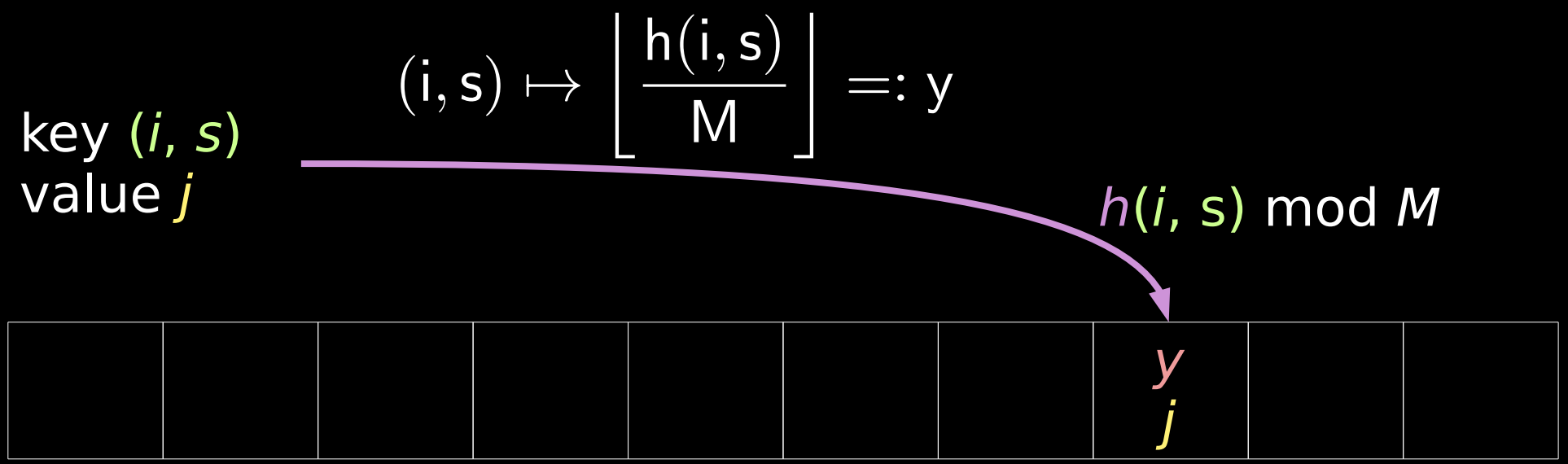
compact hash trie

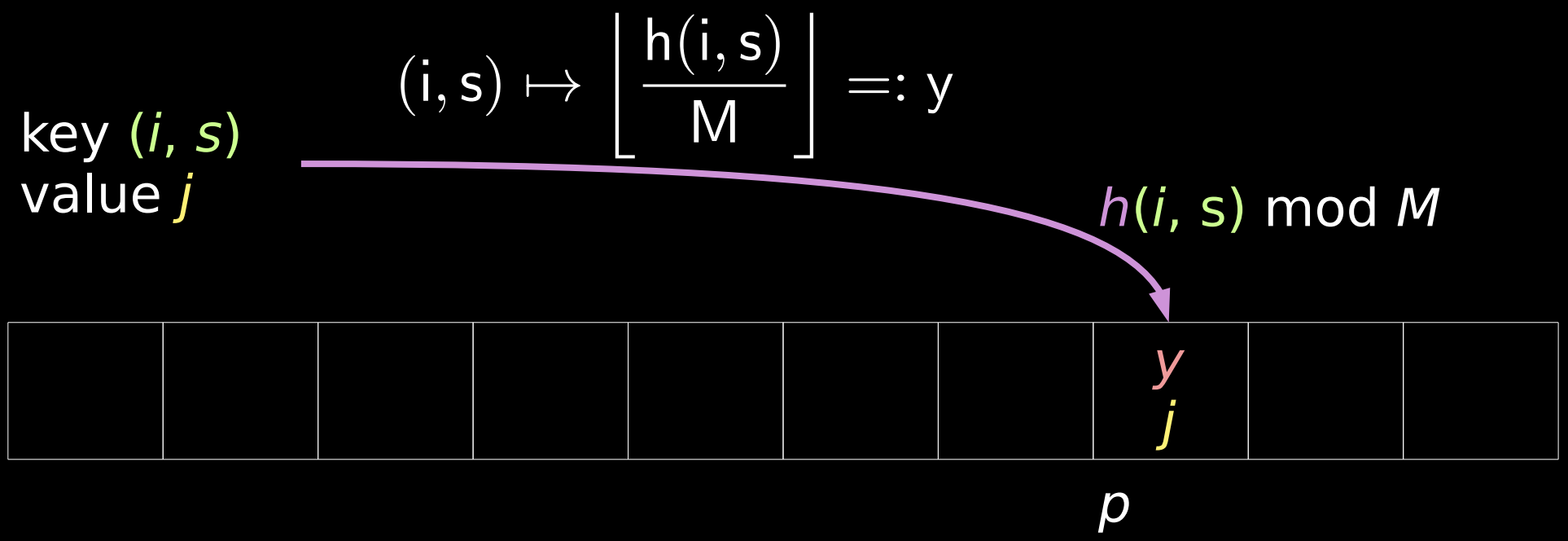
- bijective hash function
  - $(ax + b) \bmod q$
  - xorshift $x \longrightarrow x \oplus (x \gg j)$
- choose j, q so $h : [0..\sigma M] \longrightarrow [0..\sigma M]$ is bijective:
  - invariant: $z \leq \alpha M$
  - $| [0..z\text{-}1] \times [0..\sigma\text{-}1] | = z\sigma \leq \alpha\sigma M \leq \sigma M$

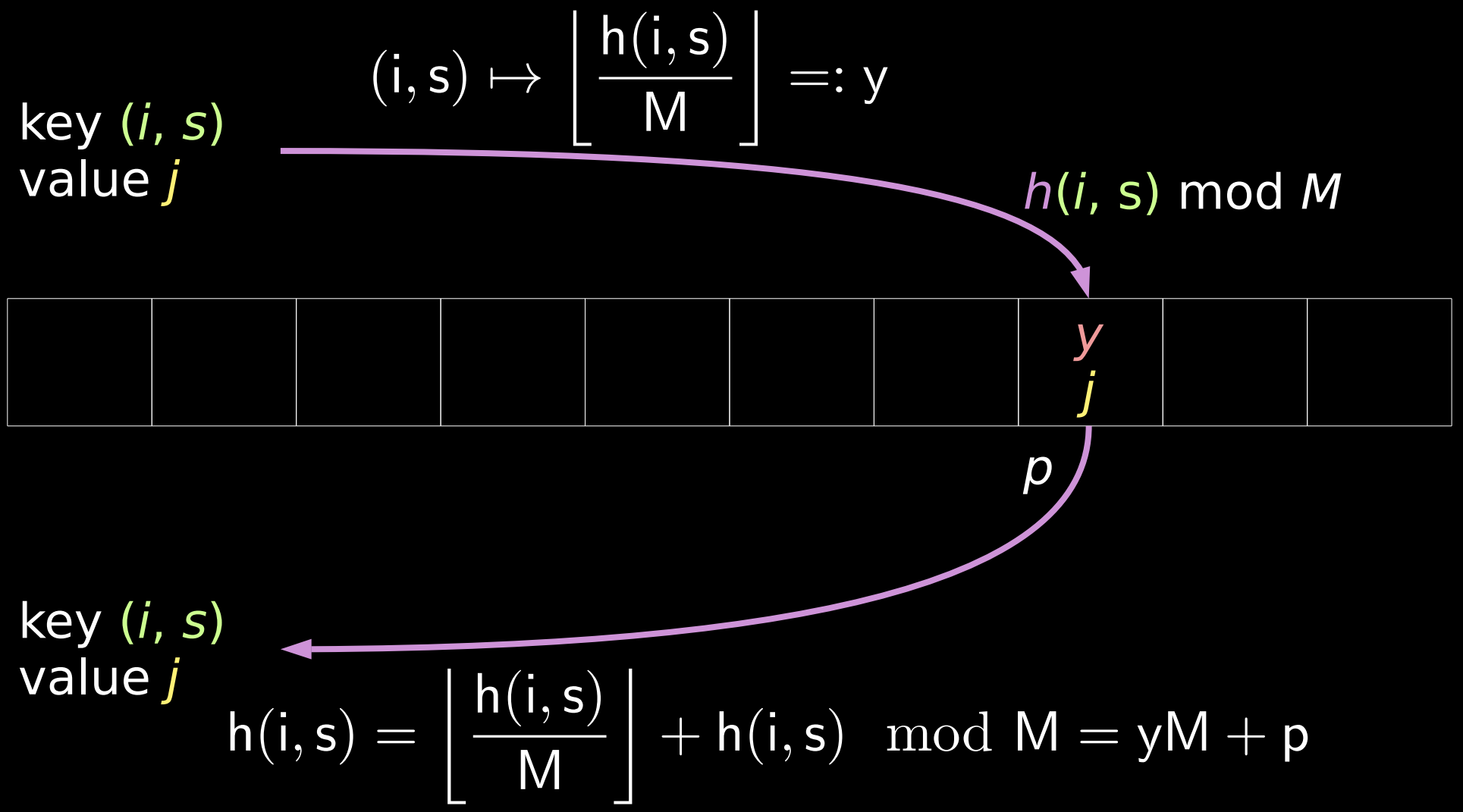
based on [Poyias and Raman'17]

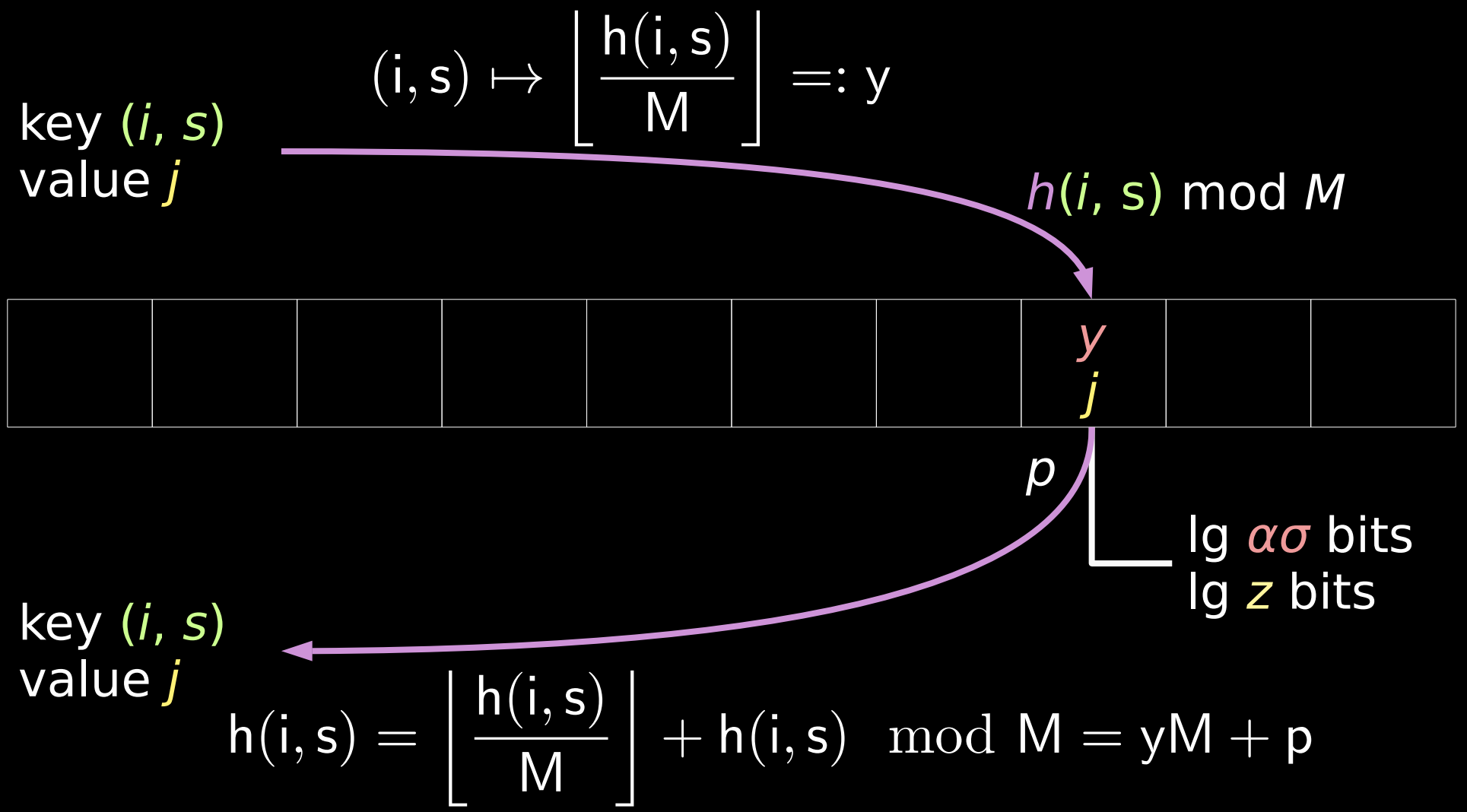
key (i, s)
value j




$(\mathrm{i}, \mathrm{s}) \mapsto \left\lfloor \frac{\mathrm{h}(\mathrm{i}, \mathrm{s})}{\mathrm{M}} \right\rfloor =: \mathrm{y}$
key (*i*, *s*)
value *j*
h(*i*, s) mod *M*
y
j


$(\mathrm{i}, \mathrm{s}) \mapsto \left\lfloor \frac{\mathrm{h}(\mathrm{i}, \mathrm{s})}{\mathrm{M}} \right\rfloor =: \mathrm{y}$
key (*i*, *s*)
value *j*
h(*i*, s) mod *M*
y
j
p

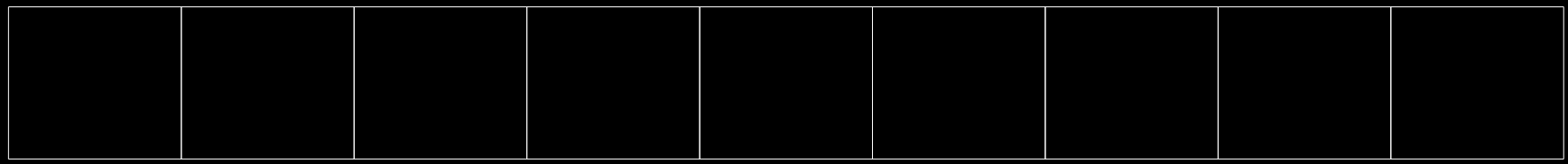

$(\mathsf{i},\mathsf{s}) \mapsto \left\lfloor \frac{\mathsf{h}(\mathsf{i},\mathsf{s})}{\mathsf{M}} \right\rfloor =: \mathsf{y}$
key (*i*, *s*)
value *j*
h(*i*, s) mod *M*
y
j
p
key (*i*, *s*)
value *j*
$\mathsf{h}(\mathsf{i},\mathsf{s}) = \left\lfloor \frac{\mathsf{h}(\mathsf{i},\mathsf{s})}{\mathsf{M}} \right\rfloor + \mathsf{h}(\mathsf{i},\mathsf{s}) \mod \mathsf{M} = \mathsf{yM} + \mathsf{p}$


$(\mathrm{i},\mathrm{s}) \mapsto \left\lfloor \frac{\mathrm{h}(\mathrm{i},\mathrm{s})}{\mathrm{M}} \right\rfloor =: \mathrm{y}$
key (*i*, *s*)
value *j*
h(*i*, s) mod *M*
y
j
p
lg *ασ* bits
lg *z* bits
key (*i*, *s*)
value *j*
$\mathrm{h}(\mathrm{i},\mathrm{s}) = \left\lfloor \frac{\mathrm{h}(\mathrm{i},\mathrm{s})}{\mathrm{M}} \right\rfloor + \mathrm{h}(\mathrm{i},\mathrm{s}) \mod \mathrm{M} = \mathrm{yM} + \mathrm{p}$

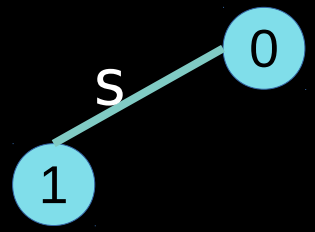
0

rolling hash trie

senescence

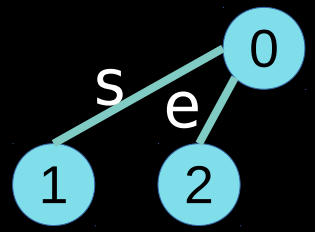


rolling hash trie

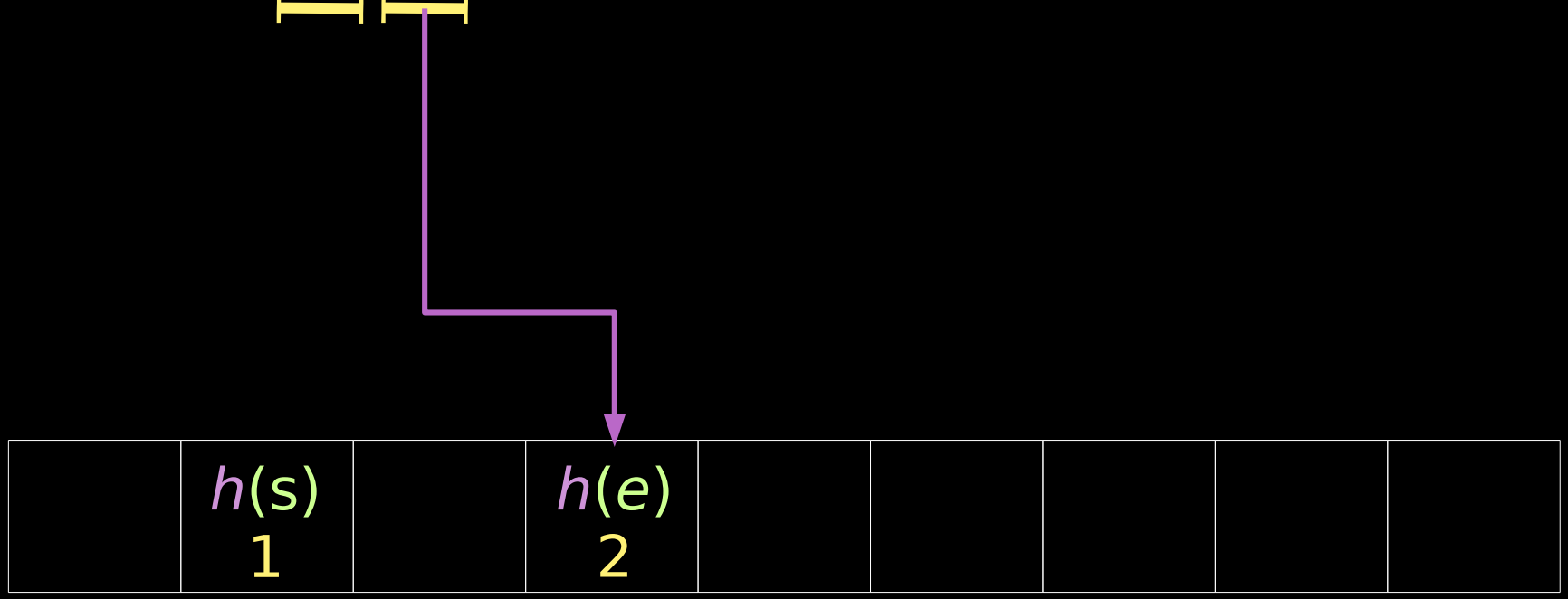


senescence

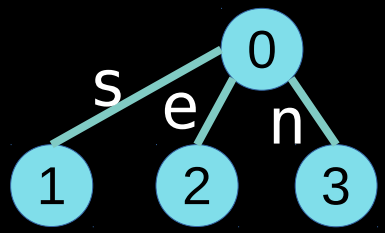
rolling hash trie



senescence



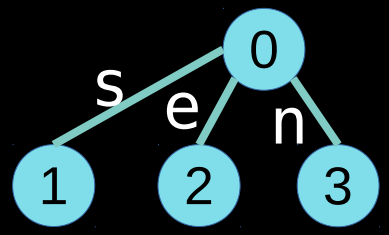
rolling hash trie



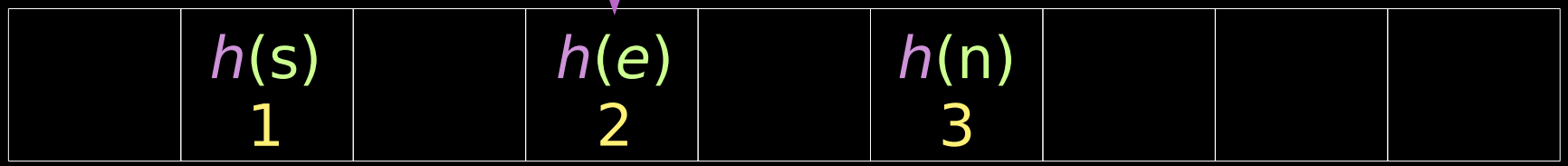
senescence

	h(s) 1		*h*(*e*) 2		*h*(n) 3			

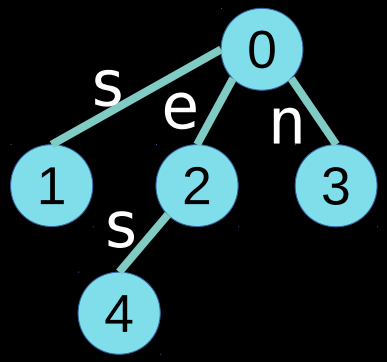
rolling hash trie



senescence



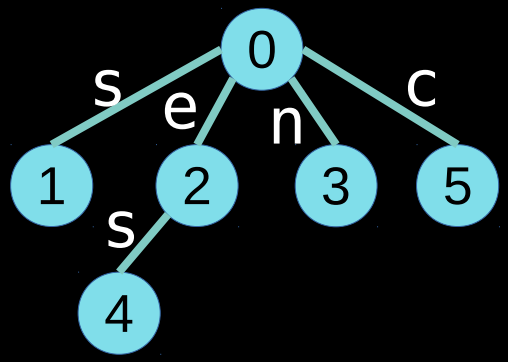
rolling hash trie



senescence

	h(s) 1		*h*(*e*) 2		*h*(n) 3		*h*(es) 4	

rolling hash trie



senescence

h(c) = *h*(es)

	h(s) 1		*h*(e) 2		*h*(n) 3		*h*(es) 4	

properties

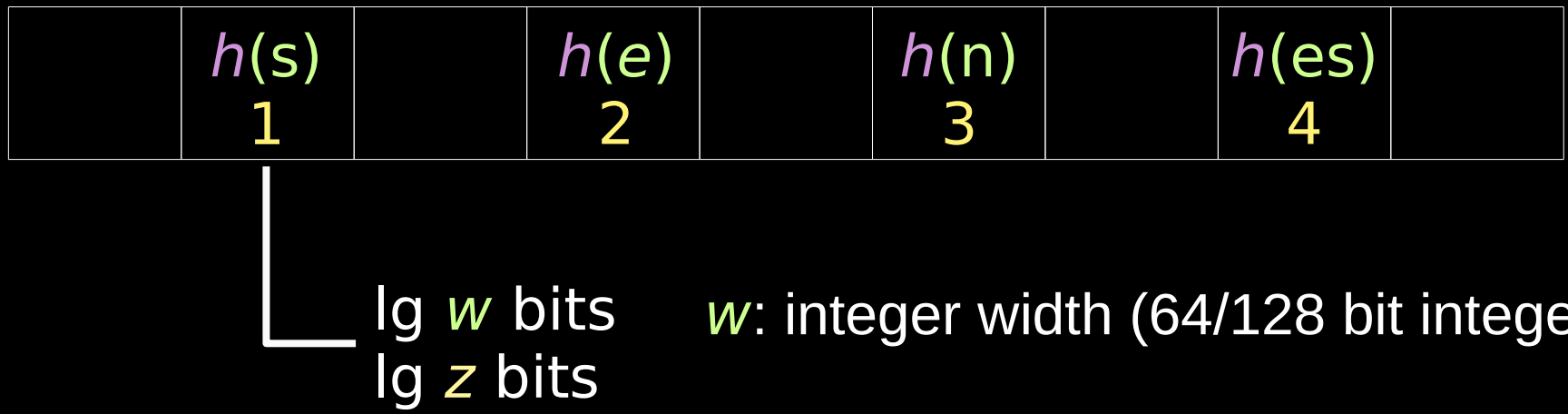
- Monte Carlo algorithm
- store fingerprint instead of keys

	h(s) 1		*h*(*e*) 2		*h*(n) 3		*h*(es) 4	

w: integer width (64/128 bit integer)

properties

- Monte Carlo algorithm
- store fingerprint instead of keys

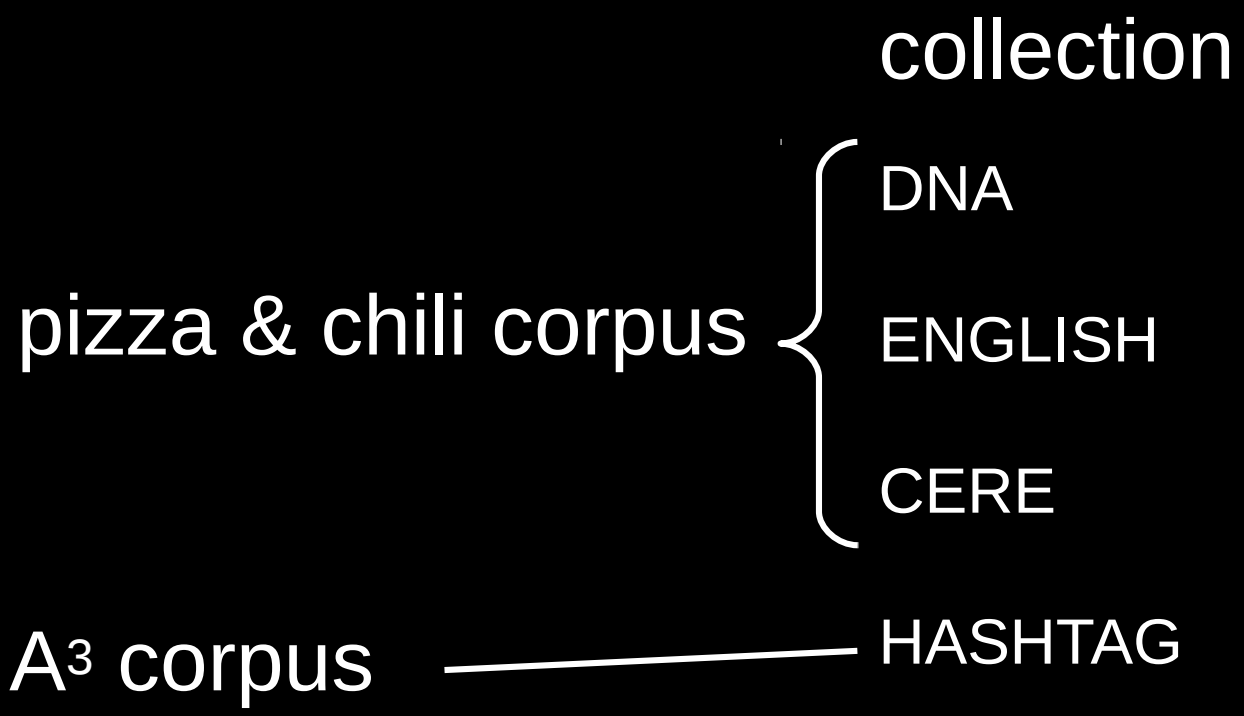


space

data structure	bits
binary trie	$z\,(2 \lg z + \lg \sigma)$
ternary trie	$z\,(3 \lg z + \lg \sigma)$
hash trie	$z/\alpha\,(2 \lg z + \lg \sigma)$
compact hash trie	$z/\alpha \lg \alpha z\sigma$
rolling hash trie	$z/\alpha \lg wz$

datasets

200 MiB text files



technical details

type sizes

- byte alphabet ($\sigma = 256$)
- $w = 64$ bits

technical details

type sizes

- byte alphabet ($\sigma = 256$)
- $w = 64$ bits

hashing

- linear probing
- hash functions:
  - splitmix64 (Vigna'15)
  - rolling:

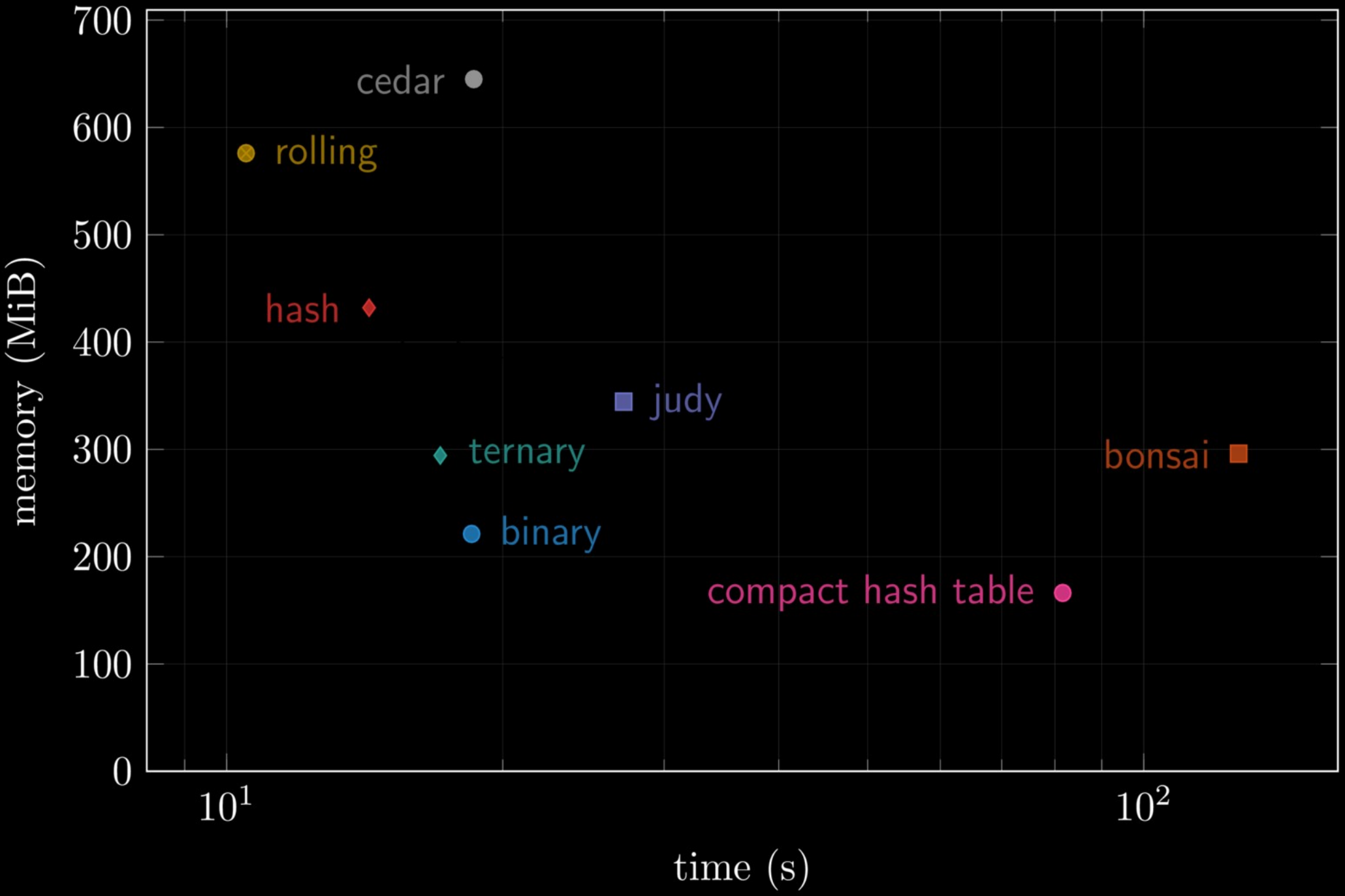
$$h(T) := \sum_{i=1}^{|T|} T[i](\sigma+1)^{|T|-i} \mod 2^{64}$$

σ+1 = 257 prime number

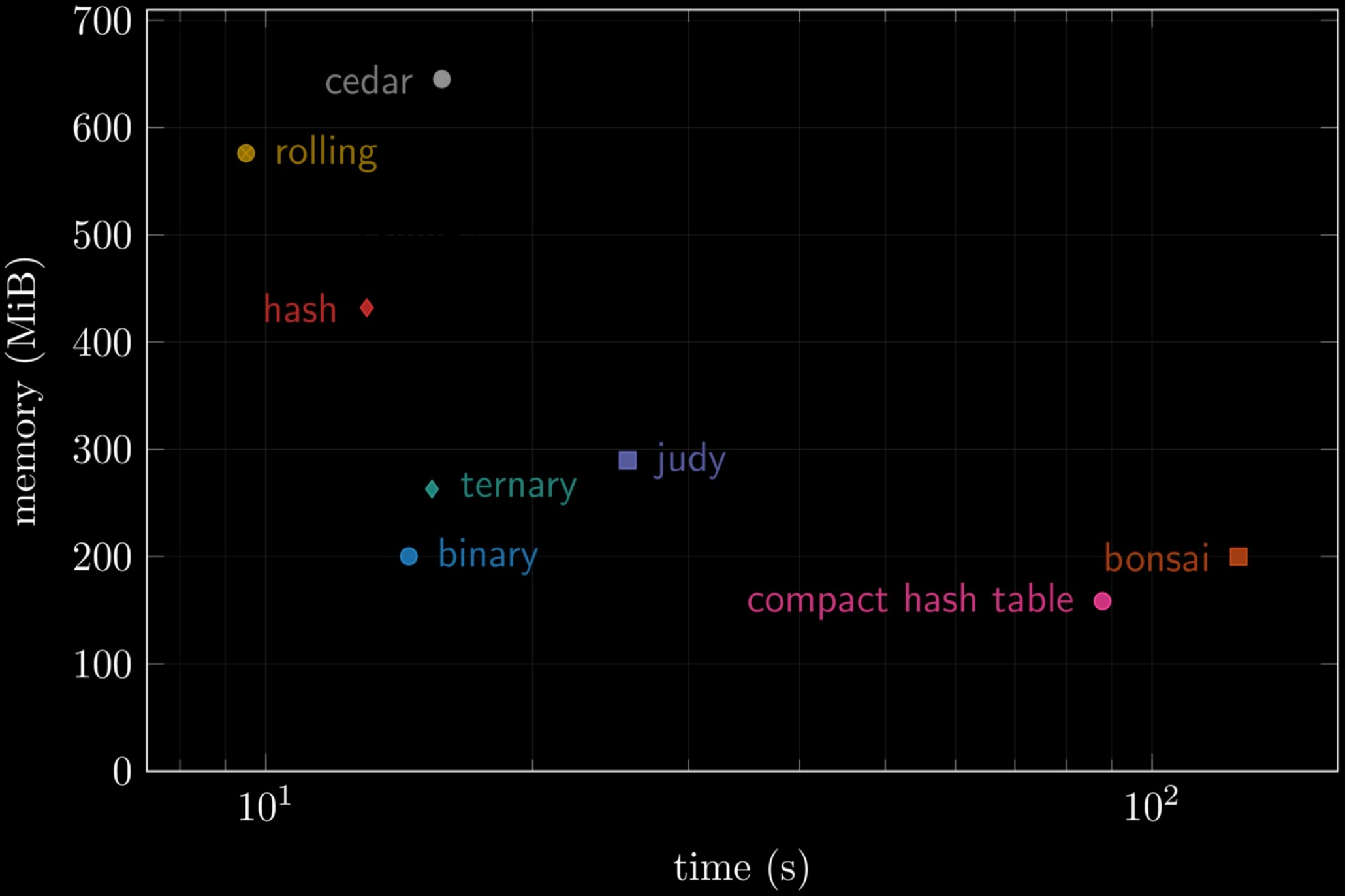
competitors

- judy array [Baskins'02]
  - 256-ary radix tries
- cedar [Yoshinaga and Kitsuregawa'14]
  - double array variant
- bonsai [Poyias and Raman'15]
  - trie with compact hashing

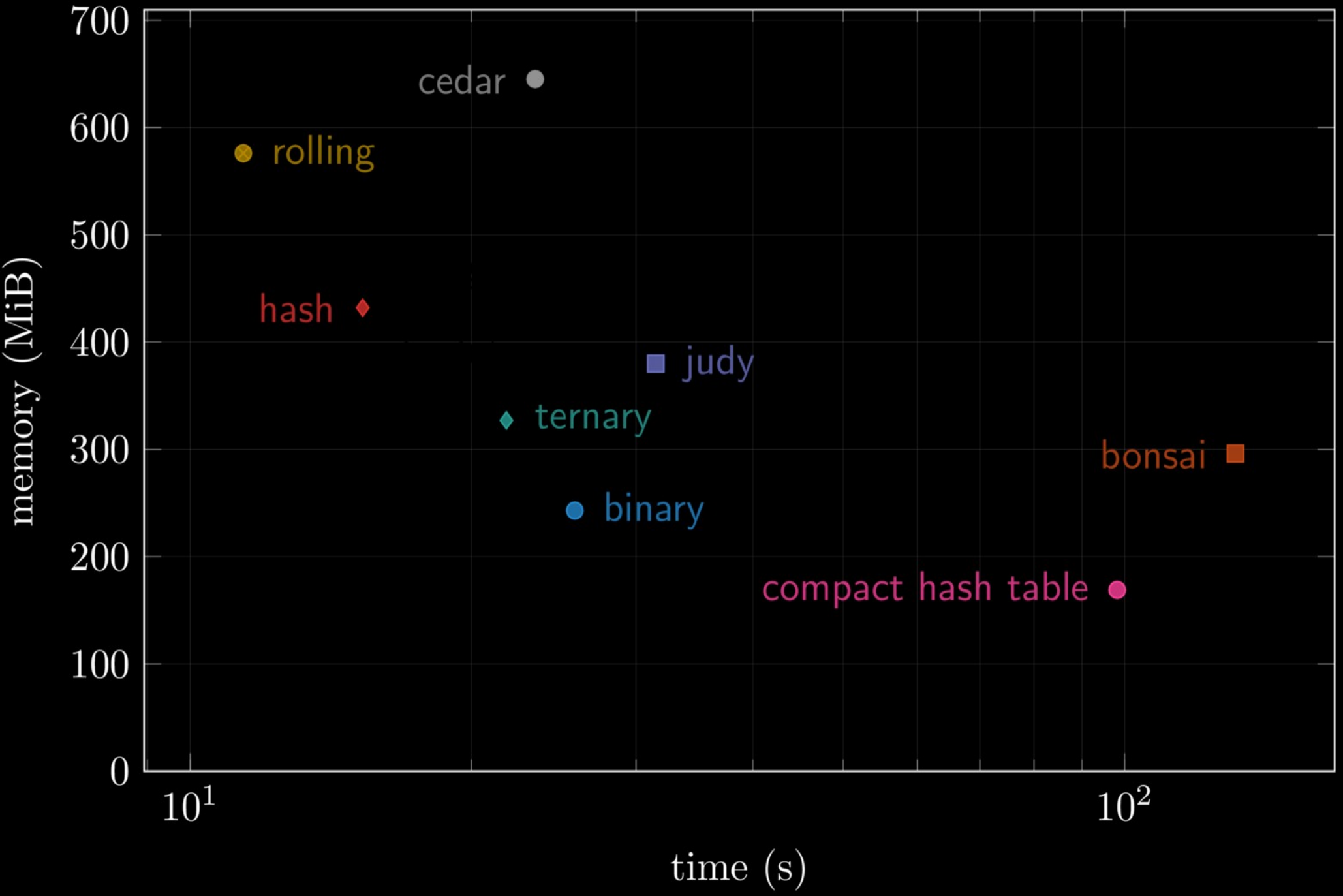
HASHTAG


700
600
500
400
300
200
100
0
memory (MiB)
cedar
rolling
hash
judy
ternary
bonsai
binary
compact hash table
10^1
10^2
time (s)

PC-DNA


700
600
500
400
300
200
100
0
memory (MiB)
cedar
rolling
hash
judy
ternary
binary
bonsai
compact hash table
10^1
10^2
time (s)

PC-ENGLISH


700
600
500
400
300
200
100
0
memory (MiB)
cedar
rolling
hash
judy
ternary
bonsai
binary
compact hash table
10^1
10^2
time (s)

PCR-CERE


600
500
400
300
200
100
0
memory (MiB)
rolling
hash
cedar
judy
ternary
binary
bonsai
compact hash table
10^1
10^2
time (s)

conclusion

- rolling fastest, but output maybe corrupt
- hash fast + correct
- ternary best trade-off speed ↔ memory
- binary slower, but less memory
- compact hash trie memory friendliest
- competitors not in Pareto front

conclusion

- rolling fastest, but output maybe corrupt
- hash fast + correct
- ternary best trade-off speed ↔ memory
- binary slower, but less memory
- compact hash trie memory friendliest
- competitors not in Pareto front

Thank you for your attention. Any questions are welcome!