Bloom Filter

BARNABAS FORGO

Motivating Example

SAFE BROWSER

Motivating Example

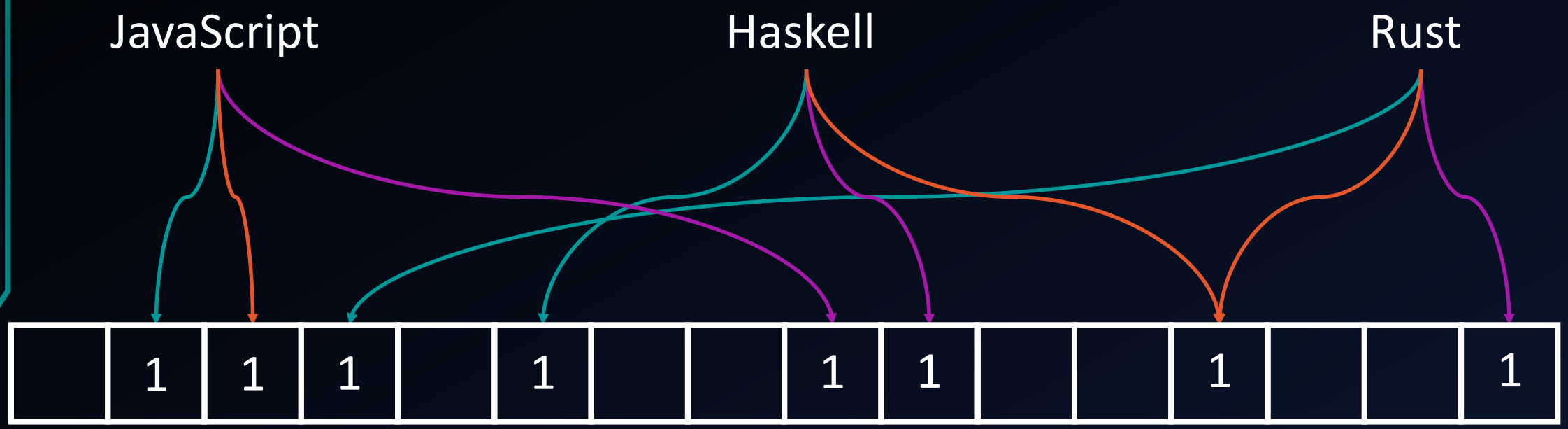
- Building a browser
- Warn about malicious URLs
- Database of millions
- 100s of MBs
- Bad lookup time, memory, download size
- Query servers
- Extra requests, cost
- What if we don’t need to store it?

Solution: Bloom Filter

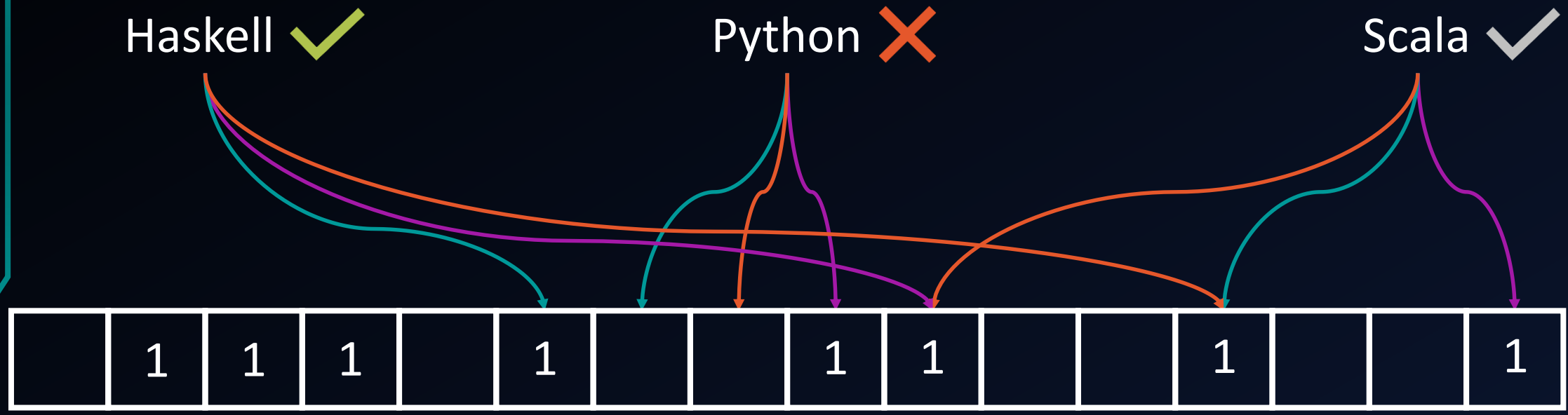
- 1970 by Burton Howard Bloom
- Probabilistic data structure
- Similar to sets
- Not storing actual data!
- “definitely not in set” or “possibly in set”
- O independent of number of elements
- Less than 10 bits per element

Operations

Insert



Membership test



Contains:
JavaScript
Haskell
Rust

Implementation & analysis

Implementing a Bloom Filter

```
class BloomFilter<T> {
  private bits: Array<Bit>;
  private hashFunctions: Array<(value: T) => number>;

  constructor(
    size: number,
    hashFunctions: Array<(value: T) => number>
  ) {
    this.bits = new Array(size);
    this.hashFunctions = hashFunctions;
  }
```

Implementing Insert

```
private computeHashes(value: T): number[] {
  return this.hashFunctions
    .map(fn => fn(value))
    .map(newHash => newHash % this.bits.length)
    .map(Math.abs);
}

insert(value: T) {
  this.computeHashes(value)
    .forEach(hash => this.bits[hash] = 1);
}
```

Analysing Insert

- *k* hash functions
- Hashing once: O(1)
- Number of elements irrelevant
- Constant == speed of hashing
- Fast, non-crypto preferred
- Hashing *k* times: O(*k*)
- Setting bits: O(*k*)
- Insert: O(*k*)

```
private computeHashes(value: T): number[] {
  return this.hashFunctions
    .map(fn => fn(value))
    .map(newHash => newHash % this.bits.length)
    .map(Math.abs);
}

insert(value: T) {
  this.computeHashes(value)
    .forEach(hash => this.bits[hash] = 1);
}
```

Implementing Membership Test

```
private computeHashes(value: T): number[] {
  return this.hashFunctions
    .map(fn => fn(value))
    .map(newHash => newHash % this.bits.length)
    .map(Math.abs);
}

query(value: T): boolean {
  return this.computeHashes(value)
    .map(hash => this.bits[hash])
    .reduce((result, bit) => bit === 1 && result, true);
}
```

Analysing Membership test

- *k* hash functions
- Hashing *k* times: O(*k*)
- Getting bits: O(*k*)
- Membership test: O(*k*)

```
private computeHashes(value: T): number[] {
  return this.hashFunctions
    .map(fn => fn(value))
    .map(newHash => newHash % this.bits.length)
    .map(Math.abs);
}

query(value: T): boolean {
  return this.computeHashes(value)
    .map(hash => this.bits[hash])
    .reduce((result, bit) => bit === 1 && result, true);
}
```

Less Hashing, Same Performance

```
private hashFunctions: number;

private computeHash(value: T): number[] {
  const hash1: number = md5(value);
  const hash2: number = crc(value);

  return range(0, this.hashFunctions)
    .map(index => hash1 + index * hash2)
    .map(newHash => newHash % this.bitmap.length)
    .map(Math.abs);
}
```

Analysing Less Hashing, Same Performance

- Kirsch & Mitzenmacher 2007
- $g_i(x) = h_1(x) + i\, h_2(x)$
- Insert, Query still O(k)
- Much lower constant factor

```
private hashFunctions: number;

private computeHash(value: T): number[] {
  const hash1: number = md5(value);
  const hash2: number = crc(value);

  return range(0, this.hashFunctions)
    .map(index => hash1 + index * hash2)
    .map(newHash => newHash % this.bitmap.length)
    .map(Math.abs);
}
```

Summary

- Checking whether an element is in a set with probability p
- Insert, Query: O(k)
- Space: O(n)
- No delete
- Lots of data
- “Have we seen this?”

Questions?

References

- Bloom, Burton H. ‘Space/Time Trade-Offs in Hash Coding with Allowable Errors’. *Communications of the ACM* 13, no. 7 (1970): 422–26.
- Blustein, James, and Amal El-Maazawi. ‘Bloom Filters. a Tutorial, Analysis, and Survey’. *Technical Report CS-2002–10. Faculty of Computer Science, Dalhousie University*, 2002.
- Kirsch, Adam, and Michael Mitzenmacher. ‘Less Hashing, Same Performance: Building a Better Bloom Filter’. In *ESA*, 6:456–67. Springer, 2006.
- Roughgarden, Tim. ‘Bloom Filters: The Basics - Stanford University’. Coursera. Accessed 5 December 2017. https://www.coursera.org/learn/algorithms-graphs-data-structures/lecture/riKfa/bloom-filters-the-basics.