

CS350: Data Structures

Bloom Filters

James Moscola

Department of Physical Sciences

York College of Pennsylvania

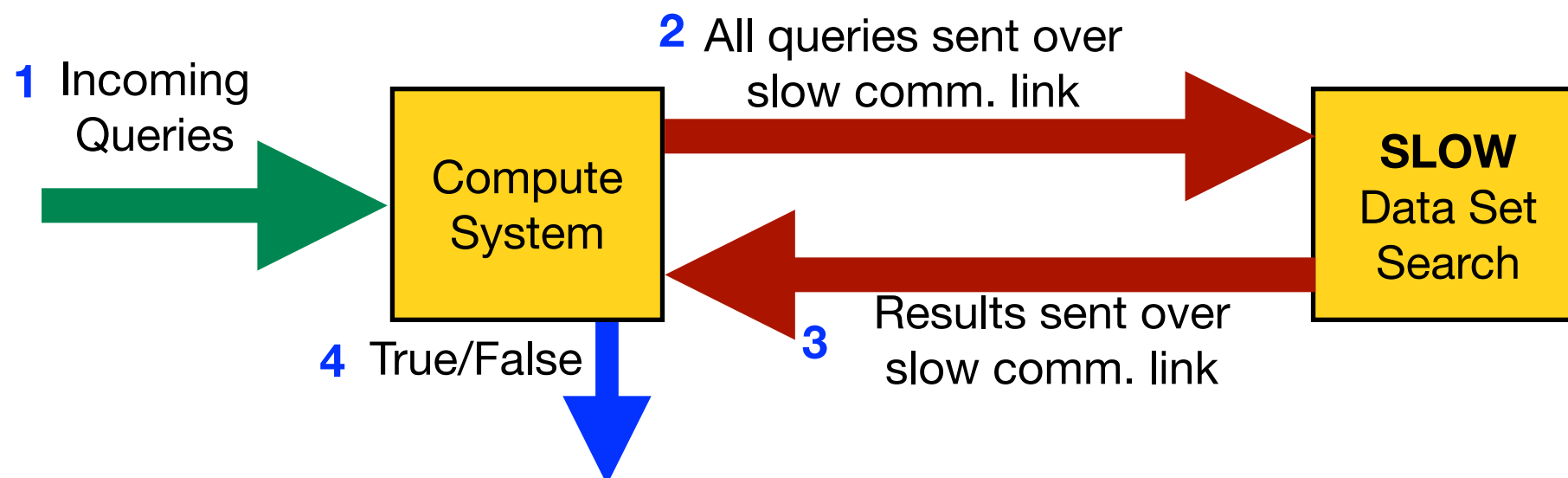


Bloom Filters

- **A space-efficient, probabilistic data structure used to see if a data element is a member of a data set**
 - A trie can also be used to check for membership in a set
- **Often used to speed up set queries by quickly filtering out all queries where the answer will definitely be 'not in set'**
 - Useful in systems where *actual* set membership can only be determined by:
 - searching a 'slow' data structure
 - reading data from a 'slow' memory
 - Don't access the 'slow' data structure/memory if you know the data isn't in there

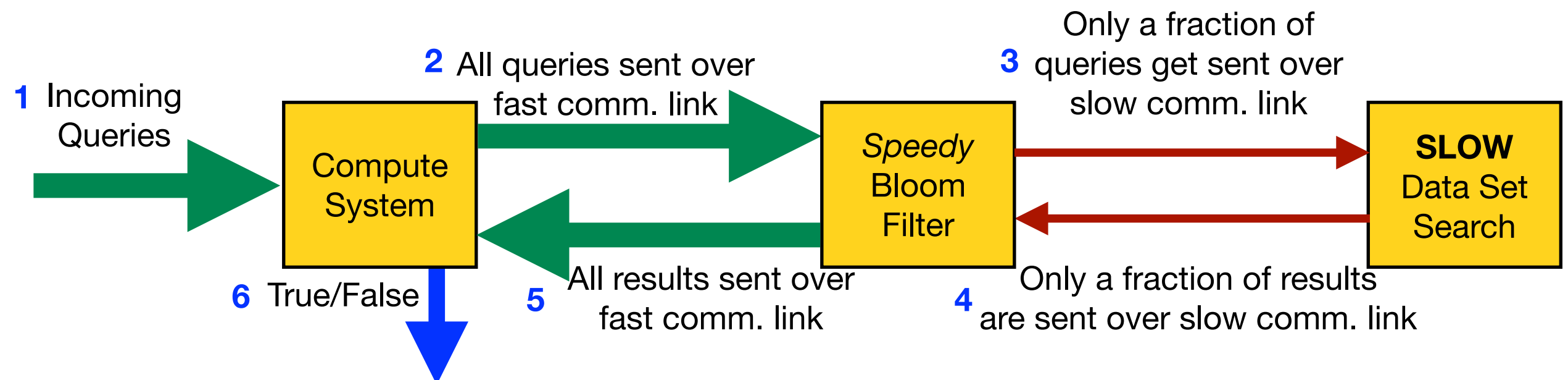
Query System Without Bloom Filters

- **A compute system requires a membership check for incoming data elements**
- **Possible problems:**
 - Communication link to main data structure/storage may be a slow link
 - Main data structure/storage may not have the performance necessary to service ALL queries
- **Solution:**
 - Only send queries to the main data structure/storage that have the possibility of existing in the data set



Query System With Bloom Filters

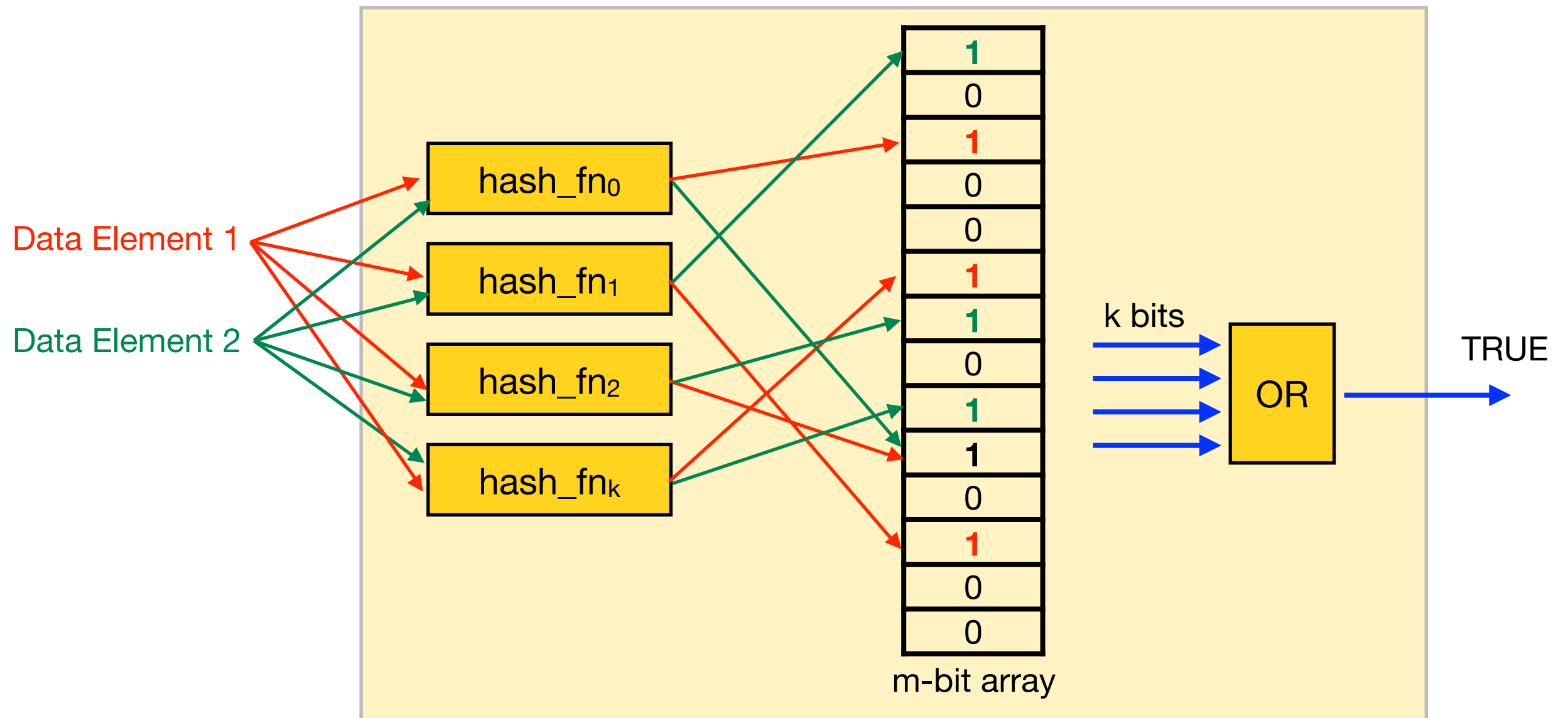
- A compute system requires a membership check for incoming data elements
- Insert a Bloom filter into the data path
 - Reduces the number of queries the need to be serviced by the slower communication links, data structures/storage
- All queries are quickly sent to and serviced by the Bloom filter
 - Bloom filter can quickly respond with a FALSE for data elements not in the set
 - For data elements that *might* be in the set, the query is sent to be serviced by the slower data set search



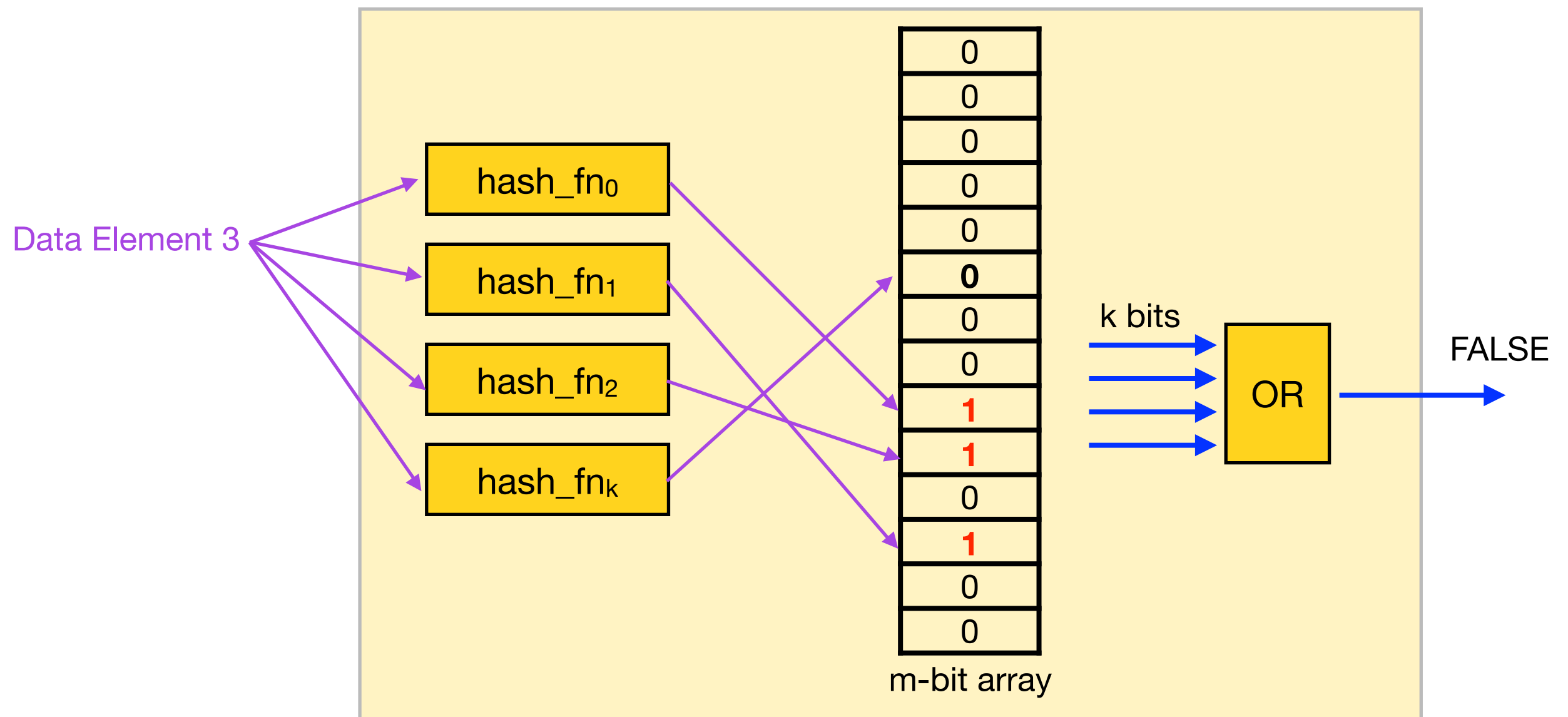
Bloom Filters

- **Utilizes multiple hash functions and an array of bits to define set**
 - Incoming data elements are hashed by **ALL** k hash functions
 - Each hash function generates an address that is used to lookup a single bit in the bit array
 - With k hash functions, k addresses are computed and k bits are read/set in the bit array
 - Each of the k bits is used to determine if a data element exists in the data set
 - If all k bits are 1, then the data element *might* exist
 - If any of the k bits are 0, then the data element definitely does not exist in the set

Checking Membership

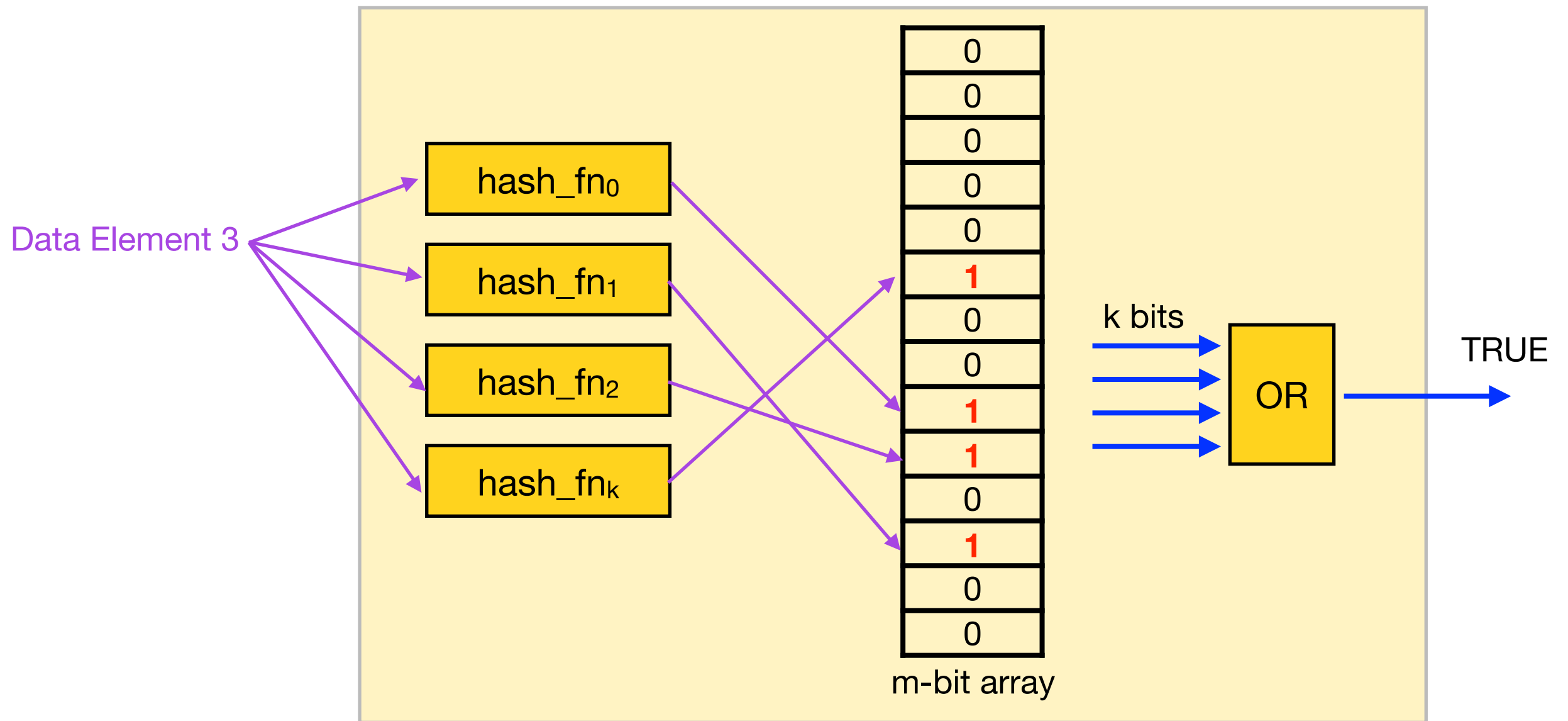


Checking Membership



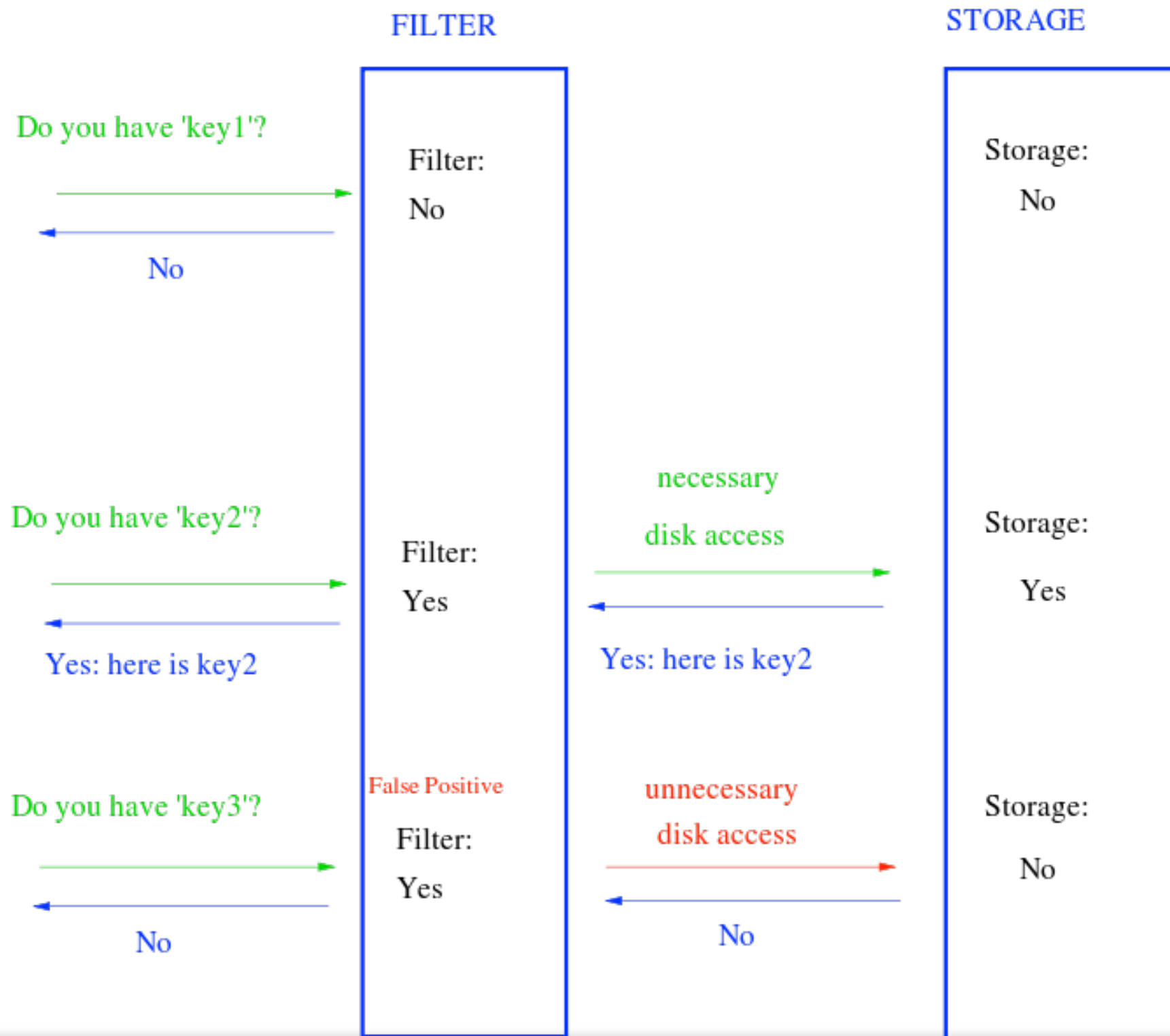
Red bits set by some other data element, not by data element 3

Checking Membership / False Positive



Red bits set by some other data element, not by data element 3

Checking Membership / Slow Storage Access



From Wikipedia

False Positives / False Negatives

- **Using more hash functions will result in fewer false positives**
- **Using a larger bit array will result in fewer false positives**
- **The more elements in the data set the greater the probability that a false positive will occur**
 - Just like the more elements in a hash table, the more likely a collision will occur
- **False negatives cannot happen in a Bloom filter**
 - If the data element was added to the filter, the filter will NEVER say the data doesn't exist

Insertion Into a Bloom Filter

- Hash data element with k unique hash functions to compute k different addresses into the m -bit array
- Set k bits in m -bit array based on the results of the k hash functions

```
bit_array[ hash_fn0(data) ] = 1;  
bit_array[ hash_fn1(data) ] = 1;  
bit_array[ hash_fn2(data) ] = 1;  
bit_array[ hash_fn3(data) ] = 1;
```

- Time complexity for insert is $O(1)$

Checking for Membership in Filter

- Hash data element with k unique hash functions to compute k different addresses into the m -bit array
- AND together k bits in m -bit array based on the results of the k hash functions

```
return bit_array[ hash_fn0(data) ] &  
        bit_array[ hash_fn1(data) ] &  
        bit_array[ hash_fn2(data) ] &  
        bit_array[ hash_fn3(data) ];
```

- Time complexity for insert is $O(1)$ for initial lookup
 - Does not include lookup in slower storage
 - Slower storage access would have its own complexity

Deleting a Data Element

- **Elements cannot be deleted from a standard Bloom filter**
 - Cannot clear bits in the bit array since multiple data elements may have 'set' the same bit
 - Unsetting a bit may have the unintended consequence of removing other data elements from the filter
- **A variation of the Bloom Filter exists that allows deletion**
 - **Counting Bloom Filter** uses small counters at each hash location instead of a single bit
 - When inserting data elements, increase counter
 - When deleting data elements, decrease counter
 - When checking membership, check to see that count is $\neq 0$

Some Math

- The probability that a false positive is reported by a Bloom Filter is approximated by the following:

$$f \approx \left(1 - e^{\frac{-nk}{m}}\right)^k$$

- where

f is the probability of a false positive

m is the size of the bit-array

k is the number of hash functions

n is the number data elements inserted into the filter

- Decrease probability of false positive by increasing either m or k
- Can *tune* desired false positive based on system resources