

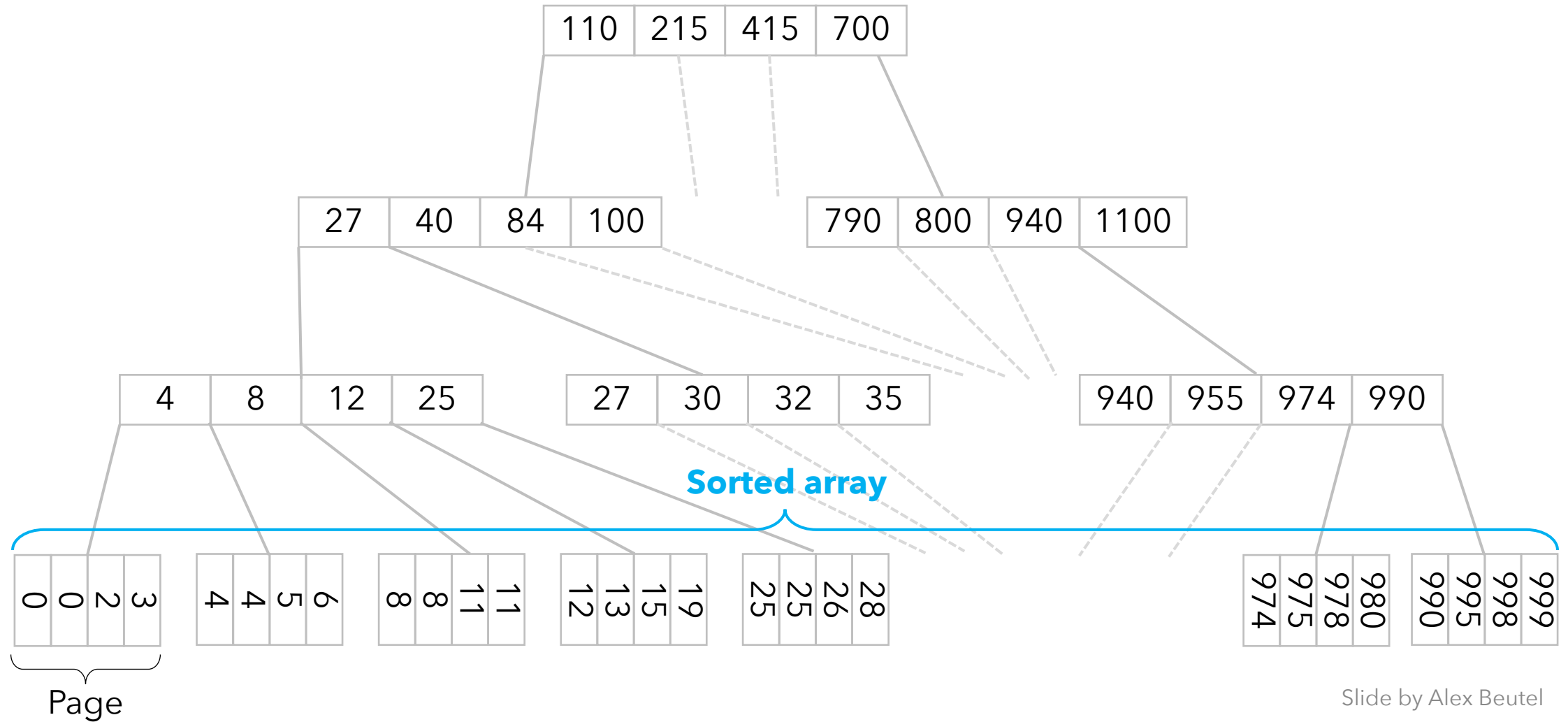
Learned index structures

Outline

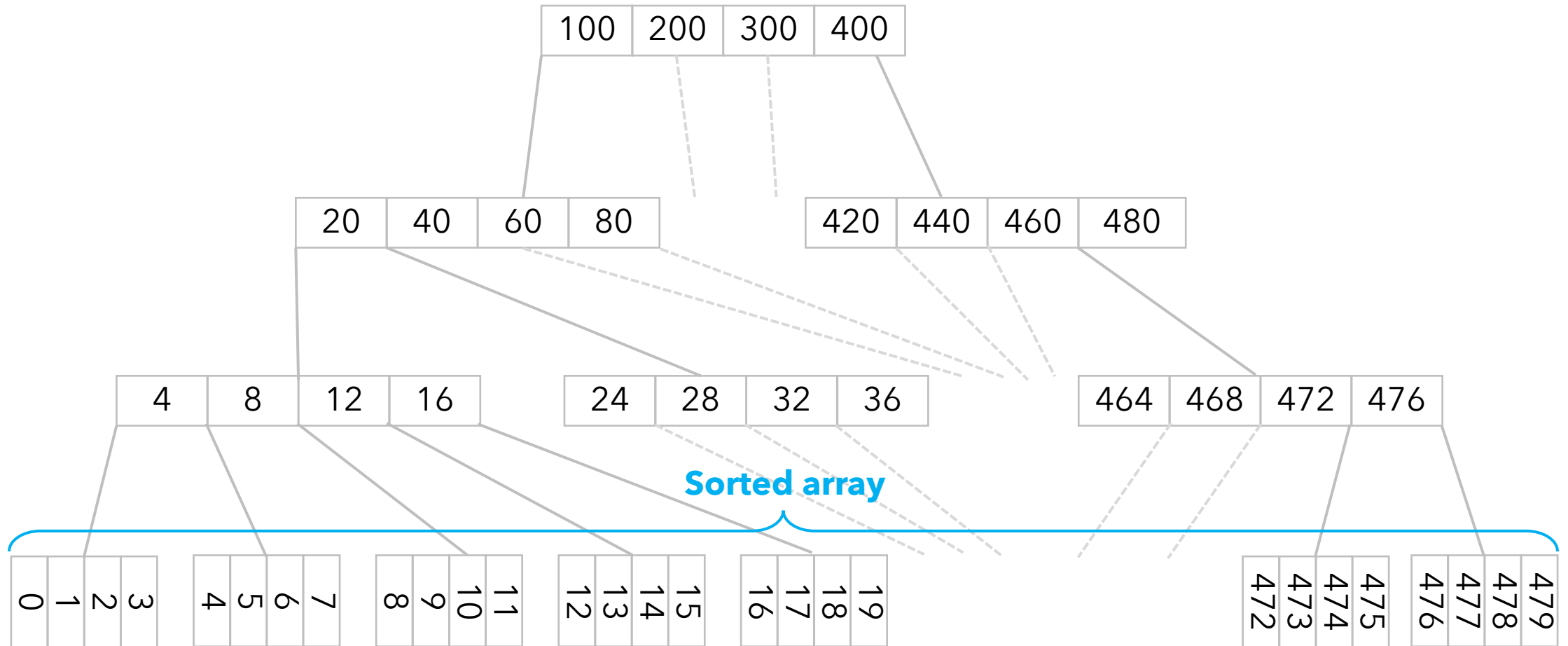
Goal: Use machine learning to augment

1. **Range index structures** (B-trees)
2. Existence index structures (Bloom filters)

B-trees



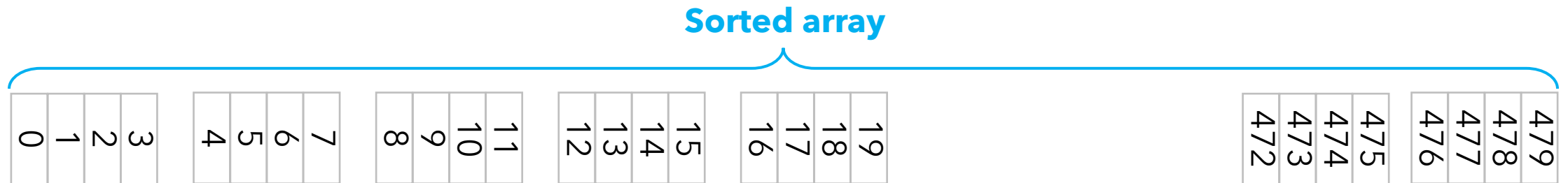
If data is all integers from 0 to 1 million?



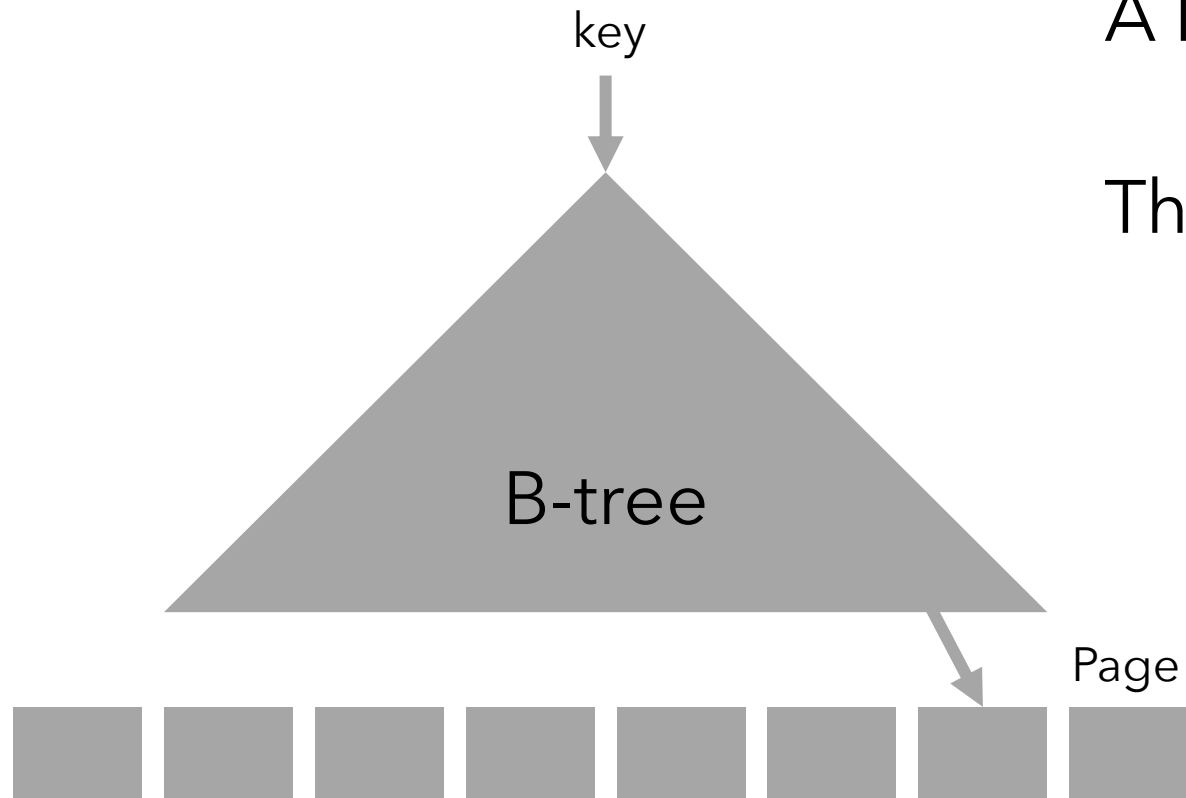
If data is all integers from 0 to 1 million?

No need for B-tree

- $O(1)$ look-up
- $O(1)$ memory



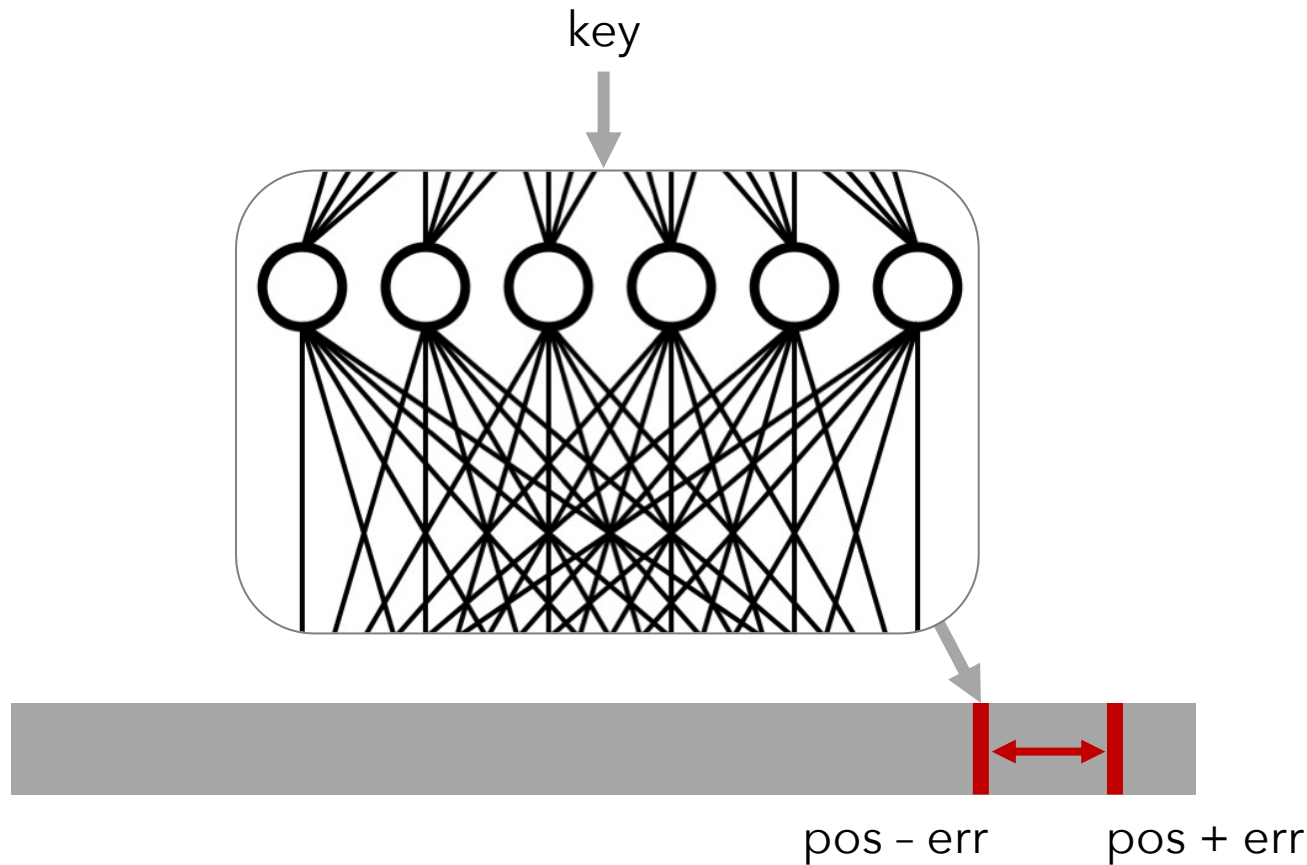
B-trees



A B-tree maps a key to a page

Then searches within the page

First attempt

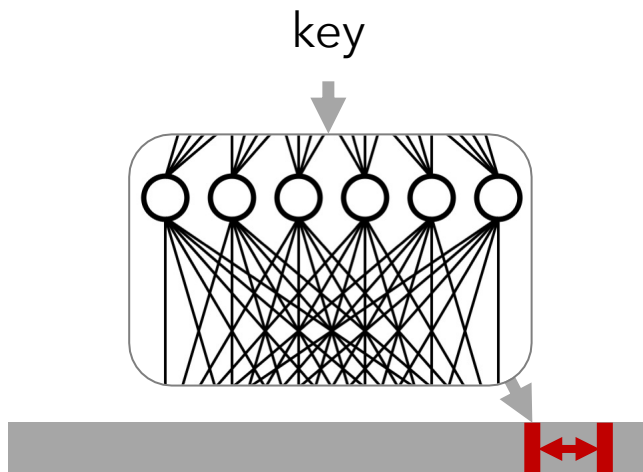


Replace B-tree
with **neural network**?

Model: $f(\text{key}) \rightarrow \text{pos}$

Then searches from
 $[\text{pos} - \text{err}, \text{pos} + \text{err}]$

First attempt



200M serve logs timestamp sorted

2-layer NN, 32-width fully-connected, ReLU
Tensorflow

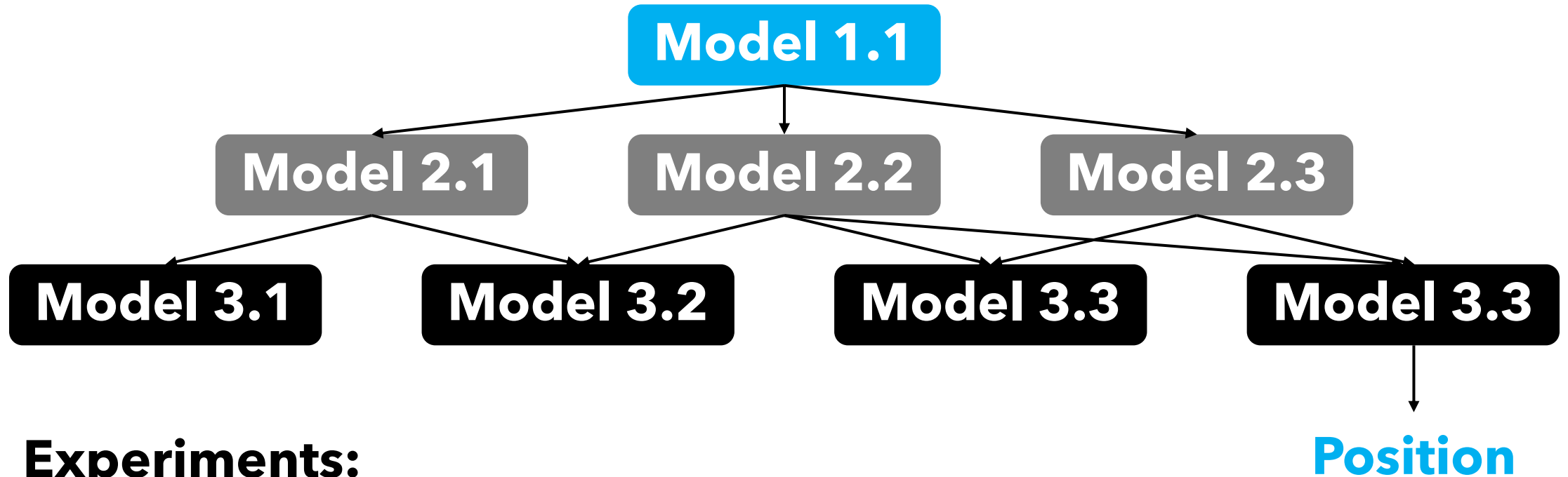
B-trees lookup time: 300ns

Model lookup time: 80,000ns

First attempt: key issues

- 1 Tensorflow designed for **big models**
Big **overhead** on small models
- 2 B-Trees **"overfit"** the data...
but NNs designed for **generalization**
- 3 B-trees are **cache-** and **operation-efficient**

Revised approach: Hierarchy of experts



Experiments:

- Up to 70% speed optimization
- Order-of-magnitude memory savings

Outline

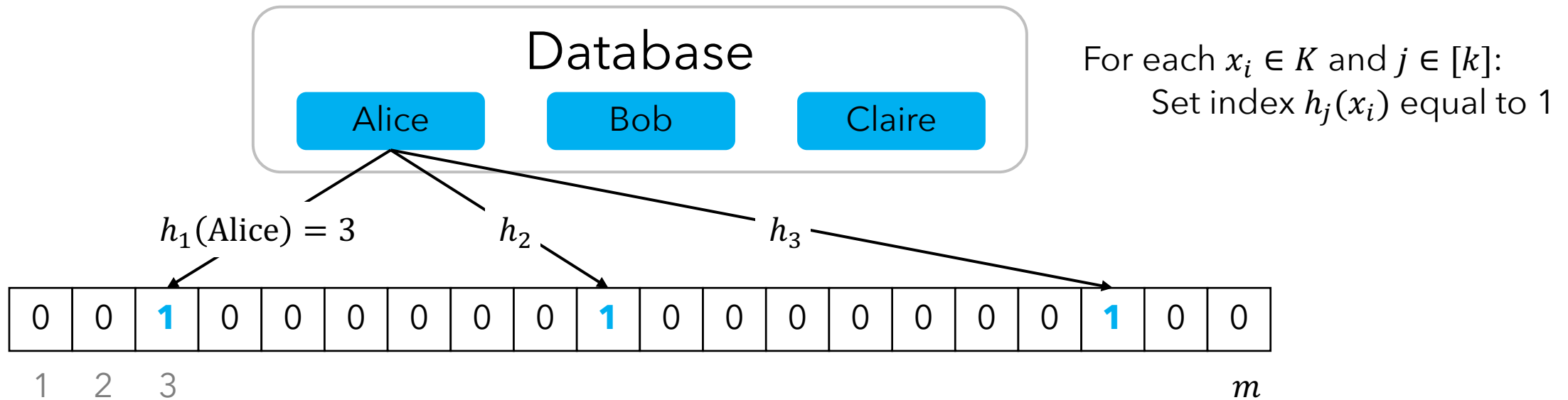
Goal: Use machine learning to augment

1. Range index structures (B-trees)

- 2. Existence index structures** (Bloom filters)

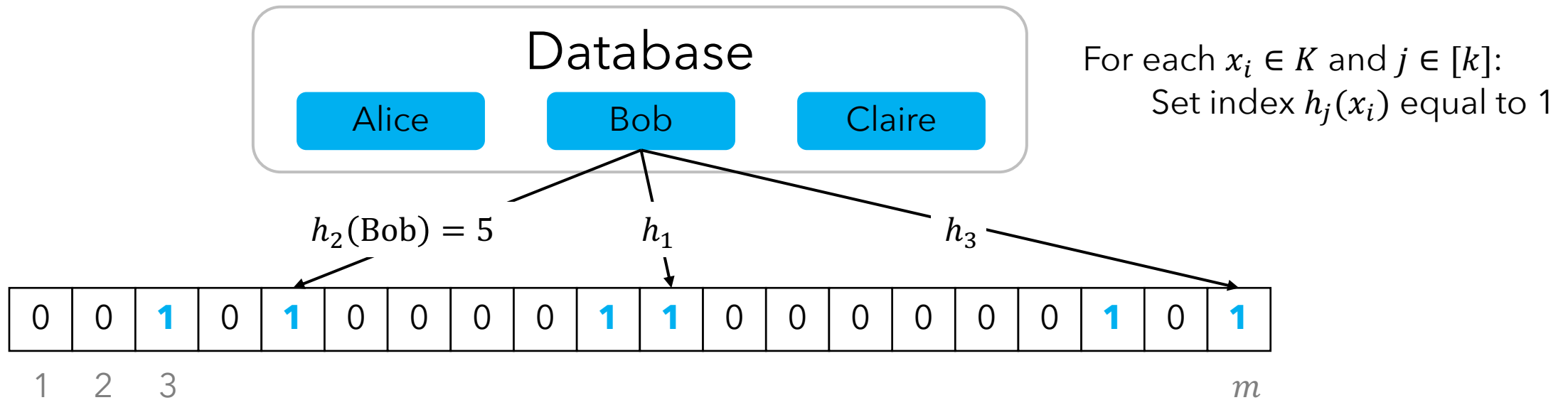
Bloom filters

- Database is a set $K \subseteq U$
- **Goal:** DS allowing us to quickly determine if any $x \in U$ is in K
 - Use hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$



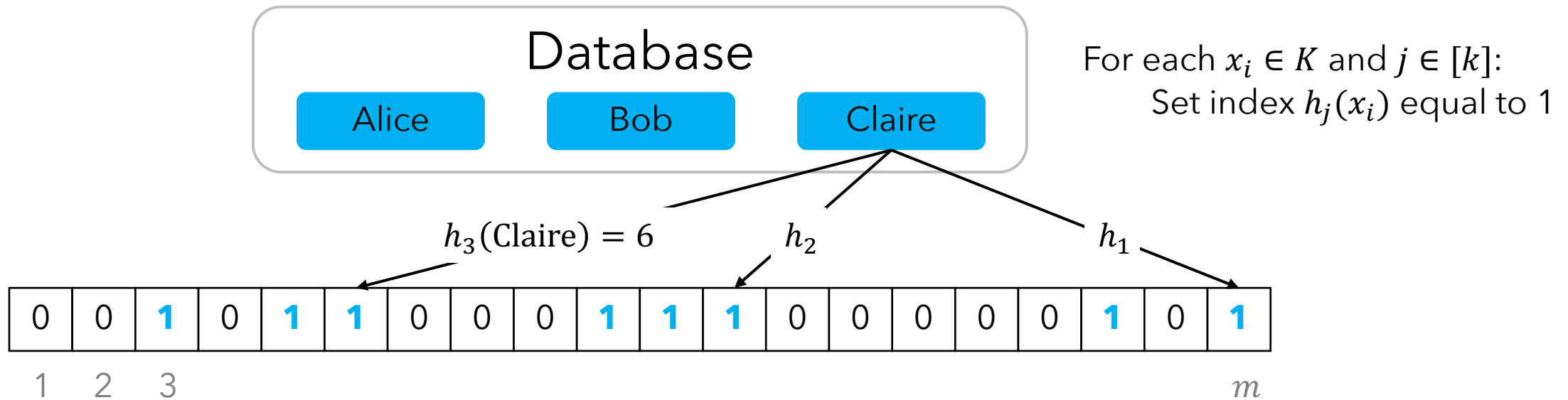
Bloom filters

- Database is a set $K \subseteq U$
- **Goal:** DS allowing us to quickly determine if any $x \in U$ is in K
 - Use hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$



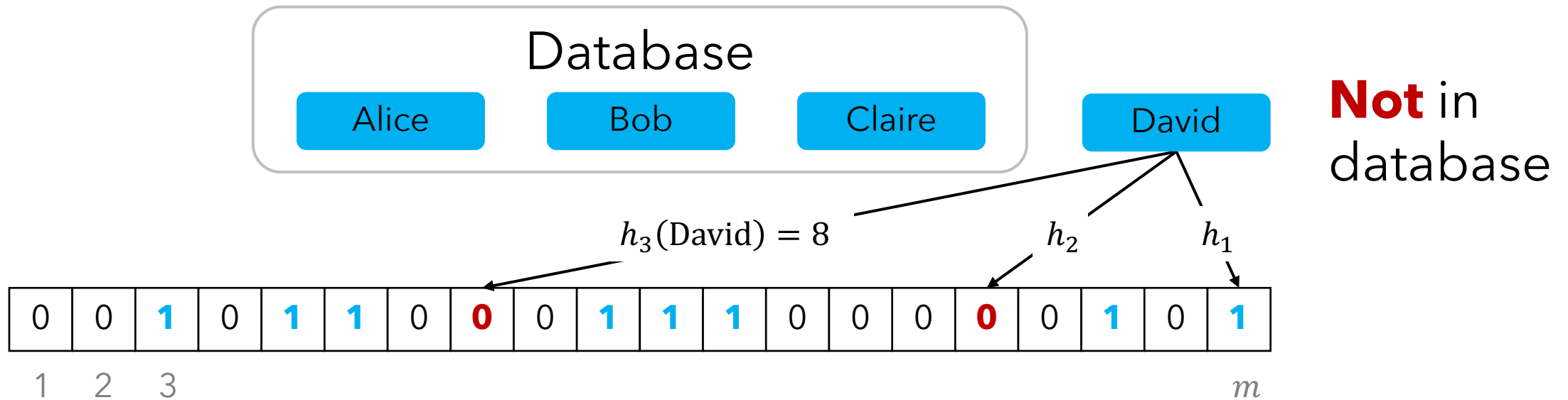
Bloom filters

- Database is a set $K \subseteq U$
- **Goal:** DS allowing us to quickly determine if any $x \in U$ is in K
 - Use hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$



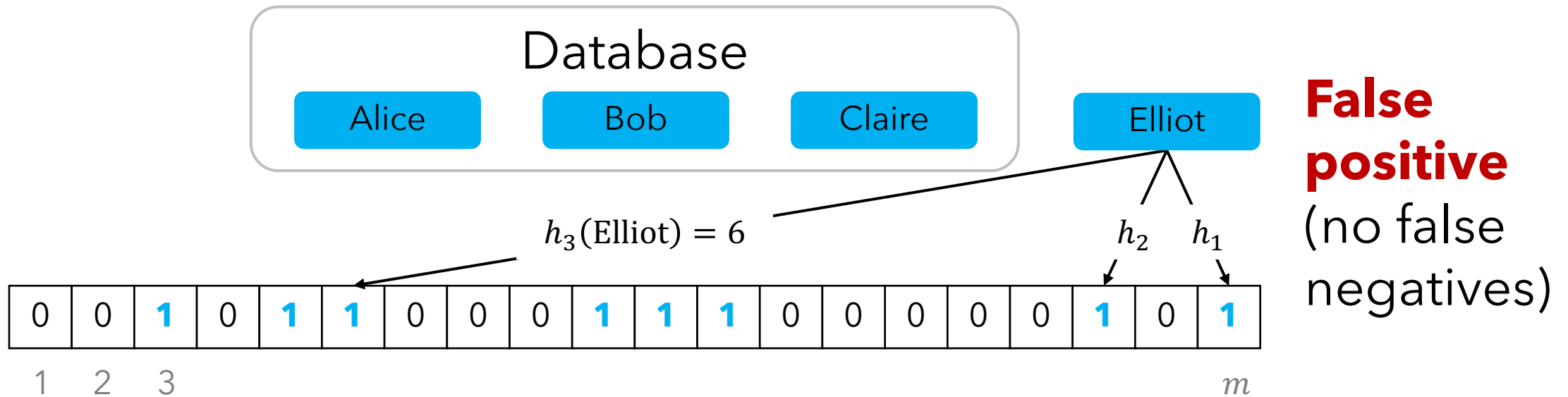
Bloom filters

- Database is a set $K \subseteq U$
- **Goal:** DS allowing us to quickly determine if any $x \in U$ is in K
 - Use hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$



Bloom filters

- Database is a set $K \subseteq U$
- **Goal:** DS allowing us to quickly determine if any $x \in U$ is in K
 - Use hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$



Learned Bloom filters

Idea 1: Replace Bloom filter with a classifier?

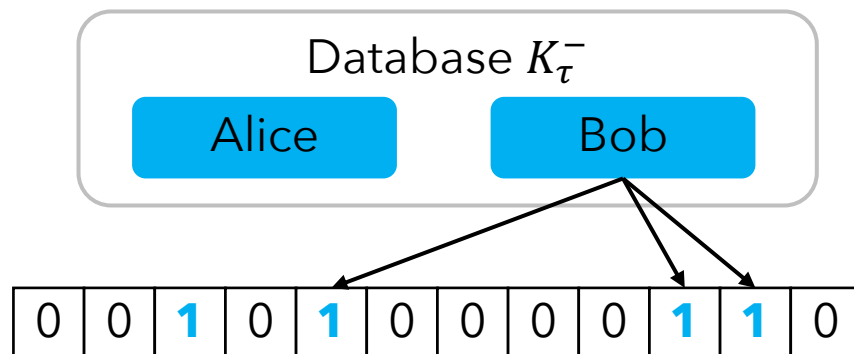
- Train ML model $f: U \rightarrow [0,1]$ with threshold τ so (hopefully)
$$f(x) \geq \tau \iff x \in K$$
- Training set: $(x, 1)$ for some $x \in S$, $(x, 0)$ for some $x \notin S$

Key issue: May be false negatives ($f(x) < \tau$ for some $x \in K$)

Learned Bloom filters

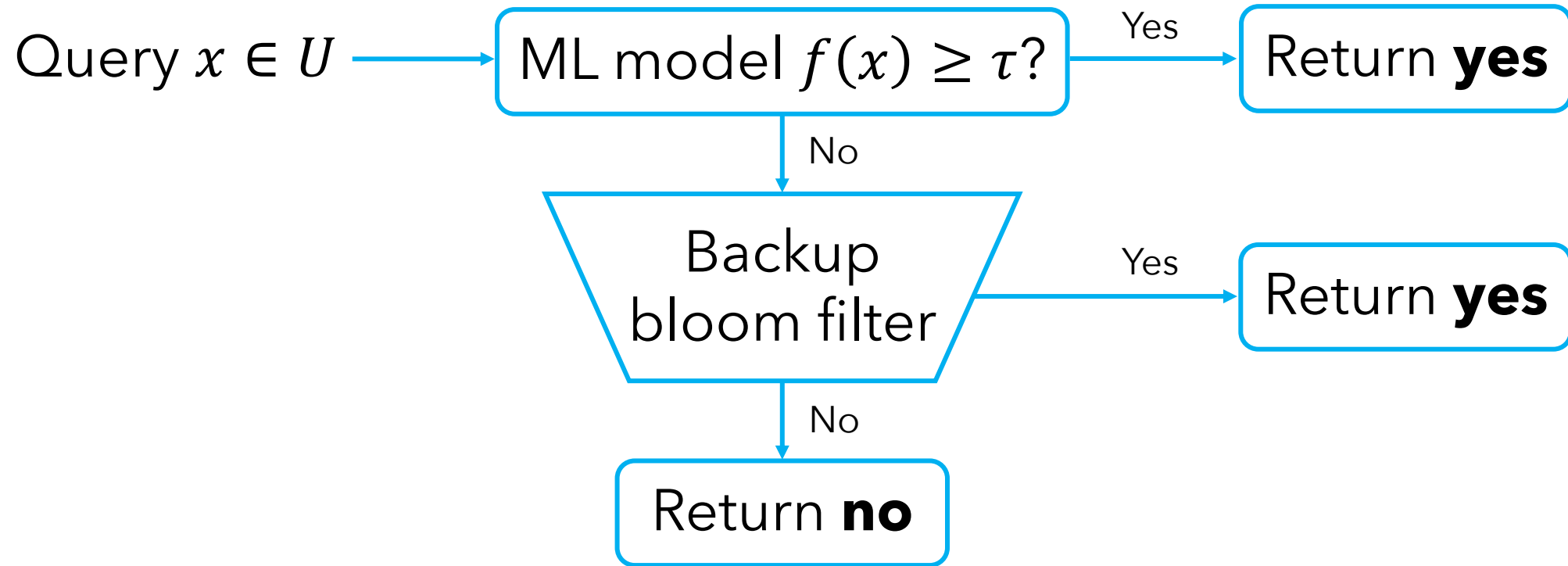
Idea 2:

- Train ML model $f: U \rightarrow [0,1]$ with threshold τ so (hopefully)
$$f(x) \geq \tau \iff x \in K$$
- Training set: $(x, 1)$ for some $x \in S$, $(x, 0)$ for some $x \notin S$
- Construct **backup Bloom filter**:
 - Bloom filter for set $K_\tau^- = \{x \in K : f(x) < \tau\}$



*Ideally much smaller than
Bloom filter for K*

Learned Bloom filters



Setting τ

- D is a distribution over $U \setminus K$
- FPR_B = false positive rate of **backup Bloom filter**

Next class: how to set Bloom filter size to control FPR_B

- False positive rate of **learned Bloom filter**:

$$\text{FPR}_O = \underbrace{\mathbb{P}_{x \sim D}[f(x) > \tau]}_{\text{Return \textbf{yes}}} + \underbrace{\mathbb{P}_{x \sim D}[f(x) \leq \tau]}_{\text{Send to backup Bloom filter}} \cdot \text{FPR}_B$$

- If want $\text{FPR}_O \leq \epsilon$:
 - Choose τ s.t. $\mathbb{P}_{x \sim D}[f(x) > \tau] \leq \frac{\epsilon}{2}$
 - $\text{FPR}_B \leq \frac{\epsilon}{2}$

Experiments

Keys: 1.7M URLs from a Google dataset

Non-keys: Random URLs and phishing URLs

Goal: 1% FPR

- Normal Bloom filter: 2.04MB
 - Learned Bloom filter:
 - ML model (RNN) requires 0.0259MB
 - Backup Bloom filter requires 1.31MB
- } 36% improvement

Outline

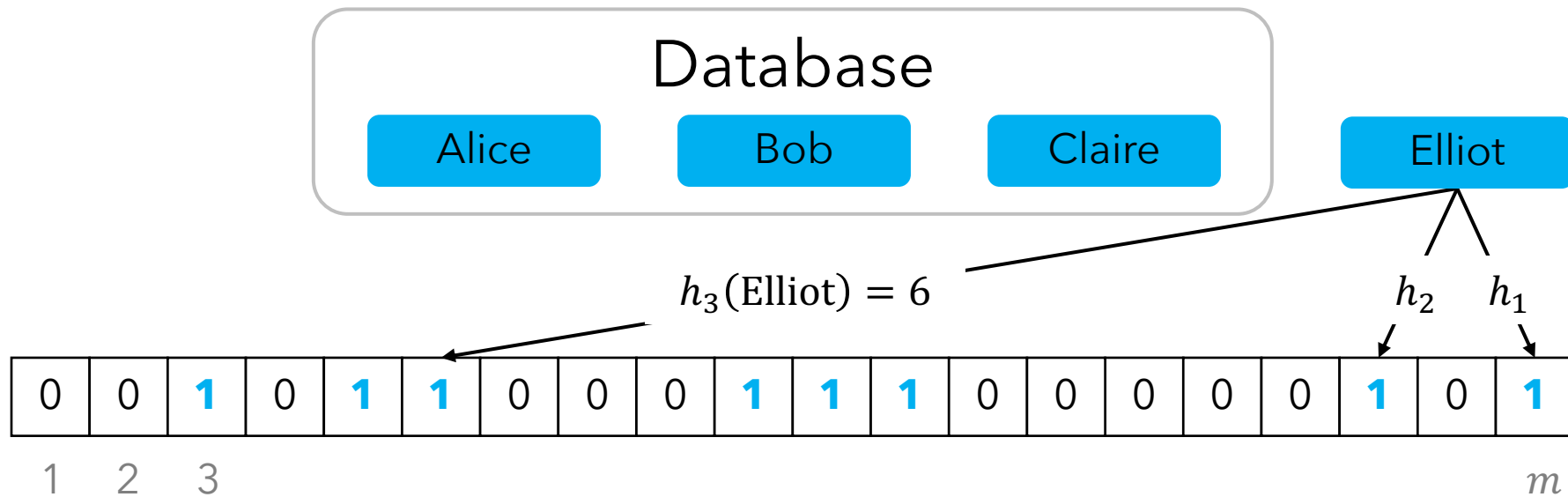
Goal: Use machine learning to augment

1. Range index structures (B-trees)
2. Existence index structures (Bloom filters)
 - i. Approach
 - ii. Theoretical guarantees/insights**

Broder, Mitzenmacher [Internet Mathematics '04]
Mitzenmacher [NeurIPS'18]

Bloom filters

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$



**False
positive**
(no false
negatives)

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]$
 - Map $x \in U$ to a random number in $[m]$, independently and uniformly
- ρ = fraction of bits set to 1
- For any $y \notin K$,
$$\begin{aligned} & \mathbb{P}[y \text{ yields a false positive} \mid \rho = q] \\ &= \mathbb{P}[h_i(y) = 1, \forall i \in [k] \mid \rho = q] \\ &= \mathbb{P}[h_1(y) = 1 \mid \rho = q] \cdots \mathbb{P}[h_k(y) = 1 \mid \rho = q] \\ &= q^k \end{aligned}$$

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1

$$\mathbb{E}[\rho] = \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m \mathbf{1}_{\{i^{\text{th}} \text{ bit set to 1}\}}\right] = \frac{1}{m} \sum_{i=1}^m \mathbb{P}[i^{\text{th}} \text{ bit set to 1}]$$

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1

$$\begin{aligned}\mathbb{E}[\rho] &= \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m \mathbf{1}_{\{i^{\text{th}} \text{ bit set to 1}\}}\right] = \frac{1}{m} \sum_{i=1}^m \mathbb{P}[i^{\text{th}} \text{ bit set to 1}] \\ &= \frac{1}{m} \sum_{i=1}^m (1 - \mathbb{P}[i^{\text{th}} \text{ bit set to 0}])\end{aligned}$$

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1

$$\mathbb{E}[\rho] = \frac{1}{m} \sum_{i=1}^m (1 - \mathbb{P}[i^{\text{th}} \text{ bit set to 0}])$$

$$\mathbb{P}[i^{\text{th}} \text{ bit set to 0}] = \mathbb{P}[h_j(x) \neq i, \forall x \in K, \forall j \in [k]]$$

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1

$$\mathbb{E}[\rho] = \frac{1}{m} \sum_{i=1}^m (1 - \mathbb{P}[i^{\text{th}} \text{ bit set to 0}])$$

$$\mathbb{P}[i^{\text{th}} \text{ bit set to 0}] = \mathbb{P}[h_j(x) \neq i, \forall x \in K, \forall j \in [k]]$$

$$\mathbb{P}[h_j(x) \neq i] = 1 - \mathbb{P}[h_j(x) = i] = 1 - \frac{1}{m}$$

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1

$$\mathbb{E}[\rho] = \frac{1}{m} \sum_{i=1}^m \left(1 - \mathbb{P}[i^{\text{th}} \text{ bit set to 0}]\right) = \frac{1}{m} \sum_{i=1}^m \left(1 - \left(1 - \frac{1}{m}\right)^{|K|k}\right)$$

$$\begin{aligned} \mathbb{P}[i^{\text{th}} \text{ bit set to 0}] &= \mathbb{P}\left[h_j(x) \neq i, \forall x \in K, \forall j \in [k]\right] \\ &= \left(1 - \frac{1}{m}\right)^{|K|k} \end{aligned}$$

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1

$$\begin{aligned}\mathbb{E}[\rho] &= \frac{1}{m} \sum_{i=1}^m \left(1 - \mathbb{P}[i^{\text{th}} \text{ bit set to 0}]\right) = \frac{1}{m} \sum_{i=1}^m \left(1 - \left(1 - \frac{1}{m}\right)^{|K|k}\right) \\ &\approx 1 - \exp\left(-\frac{|K|k}{m}\right)\end{aligned}$$

- With high probability, $\rho \approx \mathbb{E}[\rho]$ (Chernoff bound)

False positive analysis

- Database is a set $K \subseteq U$
- Hash functions $h_1, h_2, \dots, h_k: U \rightarrow [m]; \mathbb{P}[h_i(x) = j] = \frac{1}{m}$
- ρ = fraction of bits set to 1
- For any $y \notin K$,

$$\mathbb{P}[y \text{ yields a false positive}] \approx \mathbb{E}[\rho]^k \approx \left(1 - \exp\left(-\frac{|K|k}{m}\right)\right)^k$$

- If $m \approx |K| \log \frac{1}{\epsilon}$ and $k = \log \frac{1}{\epsilon'}$,

$$\mathbb{P}[y \text{ yields a false positive}] \approx \epsilon$$

Differences between FPRs

Bloom filter: **for any** $y \notin K$, $\mathbb{P}[y \text{ yields a false positive}] \leq \epsilon$

Learned Bloom filter:

- D is a distribution over $U \setminus K$
- FPR_B = false positive rate of **backup Bloom filter**
- False positive rate of **learned Bloom filter**:

$$\text{FPR}_O = \mathbb{P}_{y \sim D}[f(y) > \tau] + \mathbb{P}_{y \sim D}[f(y) \leq \tau] \cdot \text{FPR}_B$$

Return **yes**

Send to backup Bloom filter

- FPR_O is with respect to a **random draw** of $y \sim D$

Differences between FPRs

Bloom filter:  **Robust to distribution shift** $\mathbb{P}[f(y) > \tau | y \in K] \leq \epsilon$

Learned Bloom filter:  **Not robust to distribution shift**

- D is a distribution over $U \setminus K$
- FPR_B = false positive rate of **backup Bloom filter**
- False positive rate of **learned Bloom filter**:

$$\text{FPR}_O = \underbrace{\mathbb{P}_{y \sim D}[f(y) > \tau]}_{\text{Return yes}} + \underbrace{\mathbb{P}_{y \sim D}[f(y) \leq \tau]}_{\text{Send to backup Bloom filter}} \cdot \text{FPR}_B$$

- FPR_O is with respect to a **random draw** of $y \sim D$