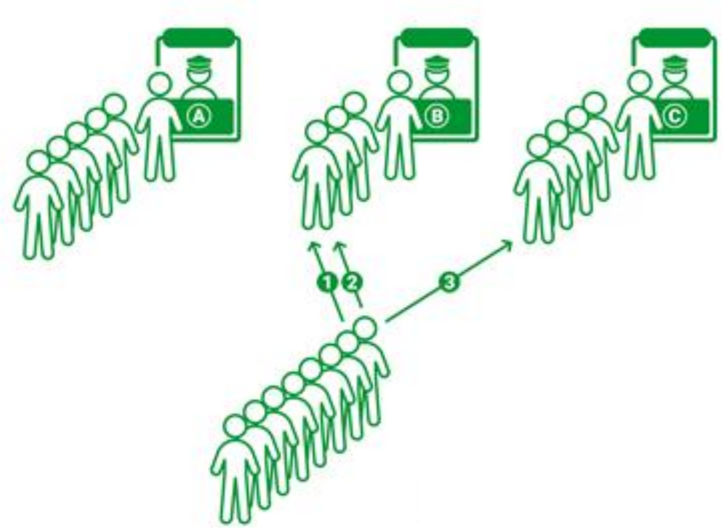
More Randomization:

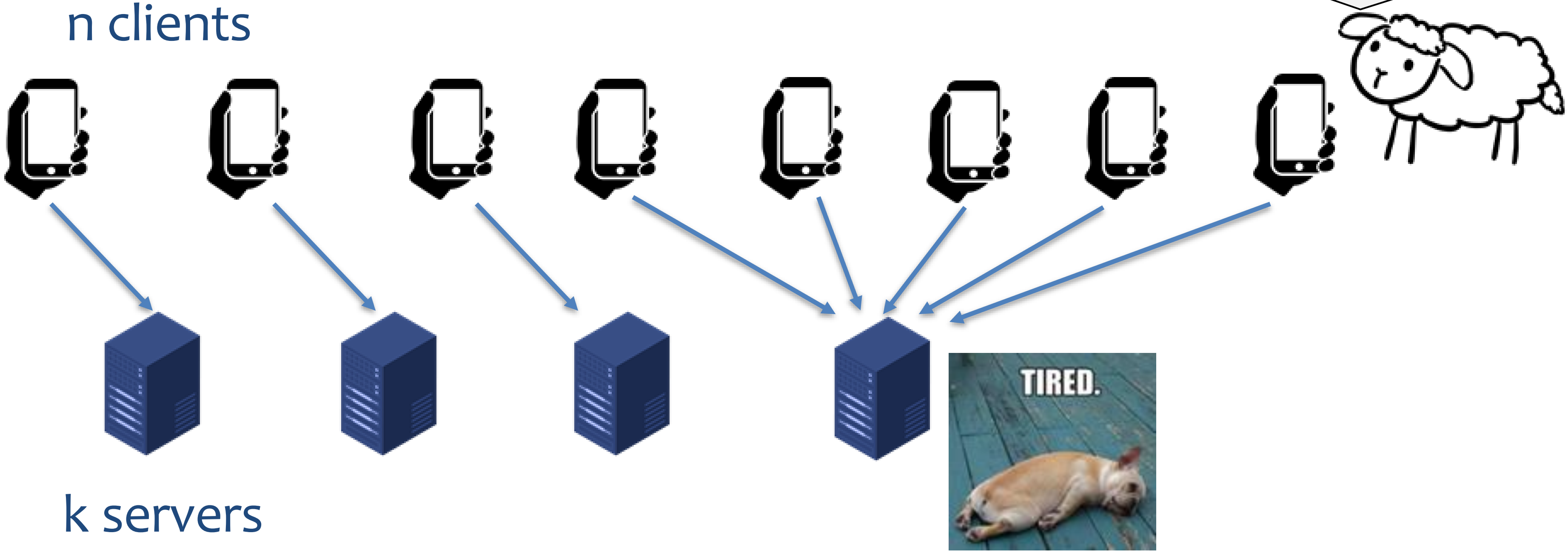
Load Balancing and Fingerprinting



Load Balancing

Imagine Google assigning search queries to servers

n clients



k servers

Goal:
- No server gets too many clients
- Each client is assigned to a server *without knowledge of the allocation of other clients to servers*

Strategy: Assign each client to a server uniformly at random!

Let’s see how well this does…

This is often formulated as:

"Balls and Bins"

n balls (clients)

k bins (servers)

Each ball goes into a random bin.

Question: How many balls in the fullest bin?

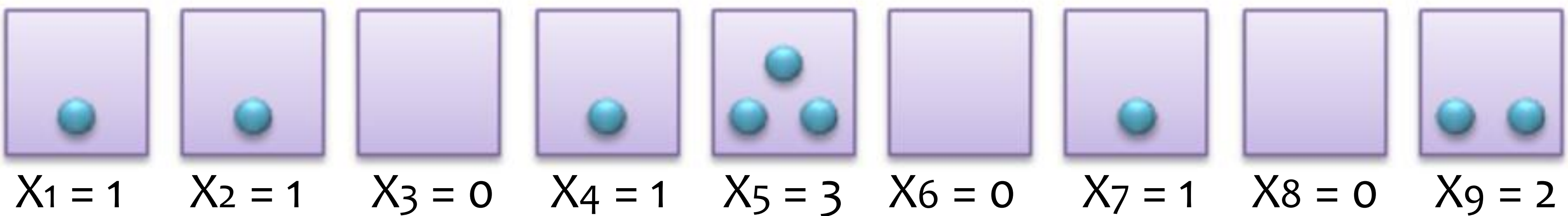
For simplicity, we will analyze the case where $n=k$.

We will prove: With prob $\geq 1-1/n$, fullest bin has $O(\log n)$ balls.

This is often formulated as:

"Balls and Bins"

n balls (clients)



n bins (servers)

First, let's calculate the expected number of balls per bin:

Let $\mathbf{X_j}$ be the number of balls in bin j.

Let $\mathbf{X_{ij}}$ be an indicator r.v. for whether ball i is in bin j.

Observation: $\mathbf{X_j} = \sum_{i=1}^{n} \mathbf{X_{ij}}$

So, $E[\mathbf{X_j}] = E[\sum_{i=1}^{n} \mathbf{X_{ij}}] = \sum_{i=1}^{n} E[\mathbf{X_{ij}}] = \sum_{i=1}^{n} Pr[X_{ij}=1] = \sum_{i=1}^{n} \frac{1}{n} = 1$

This is often formulated as:

"Balls and Bins"

n balls (clients)

$X_1 = 1$ $X_2 = 1$ $X_3 = 0$ $X_4 = 1$ $X_5 = 3$ $X_6 = 0$ $X_7 = 1$ $X_8 = 0$ $X_9 = 2$

n bins (servers)

Our goal is to bound the probability that the fullest bin has "many" balls

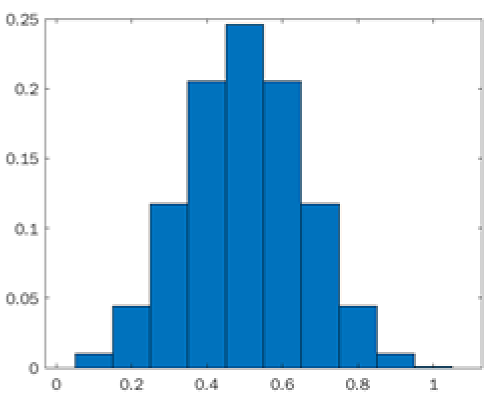
Last lecture, to bound probabilities we used *Markov's inequality*.
It turns out that won't suffice here!

We need a stronger bound, that will use the fact that $\mathbf{X_j}$ is a sum of <u>independent</u> r.v.'s.

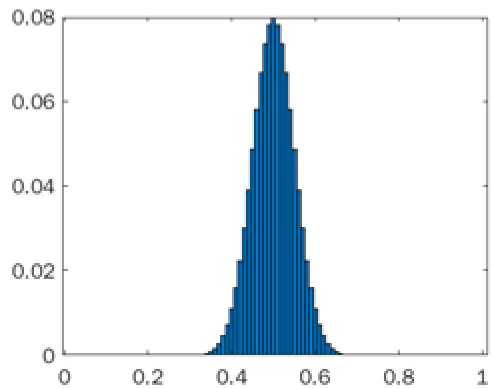
Sum of Independent r.v.'s is predictable

Flip a coin n times. What fraction of flips are heads?

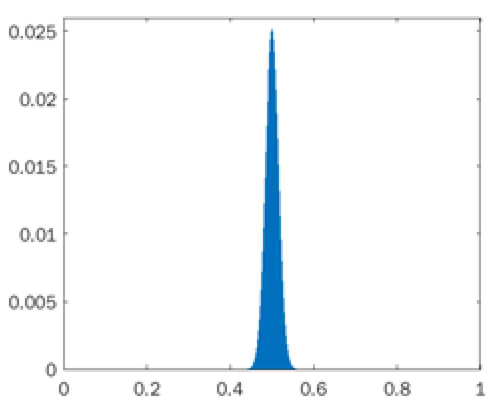
(Number of head flips is a sum of <u>independent</u> indicator r.v.'s for whether the i^{th} flip is heads)



$n = 10$



$n = 100$



$n = 1000$

Chernoff Bounds

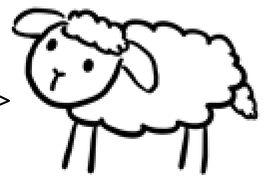
(we won't prove)

How do these compare to Markov?

Let $\mathbf{Y_1,\ldots,Y_k}$ be independent r.v.'s taking values in the range $[0,1]$.

Let $\mathbf{Y} = \sum_{i=1}^{k} \mathbf{Y_i}$. Let $\mu = E[\mathbf{Y}]$.

woohoo! I'm in the bounds



"Large Deviation" bound:

For any $\lambda \geq 1$: 1.5

$$\Pr[\mathbf{Y} - \mu \geq \lambda\mu] \leq e^{-\lambda\mu/3}$$

1

6 ln n - 1

Markov

$\frac{1}{\lambda - 1}$

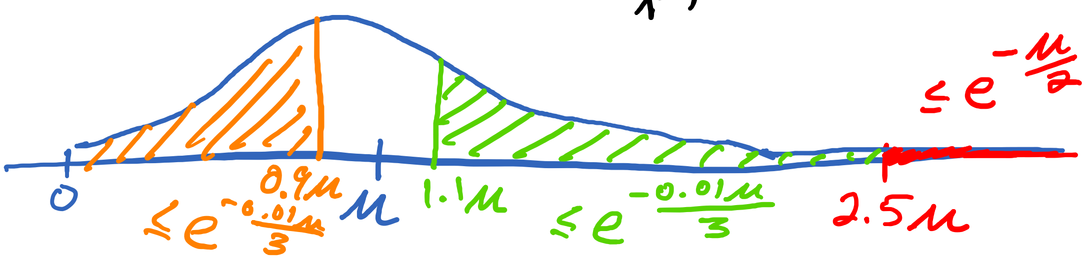
"Small Deviation" bounds:

For any $\lambda \in [0,1]$: 0.1 0.1

$$\Pr[\mathbf{Y} - \mu \geq \lambda\mu] \leq e^{-\lambda^2\mu/3}$$

$$\Pr[\mathbf{Y} - \mu \leq -\lambda\mu] \leq e^{-\lambda^2\mu/3}$$

this one we won't use in class, but you may use it on the HW



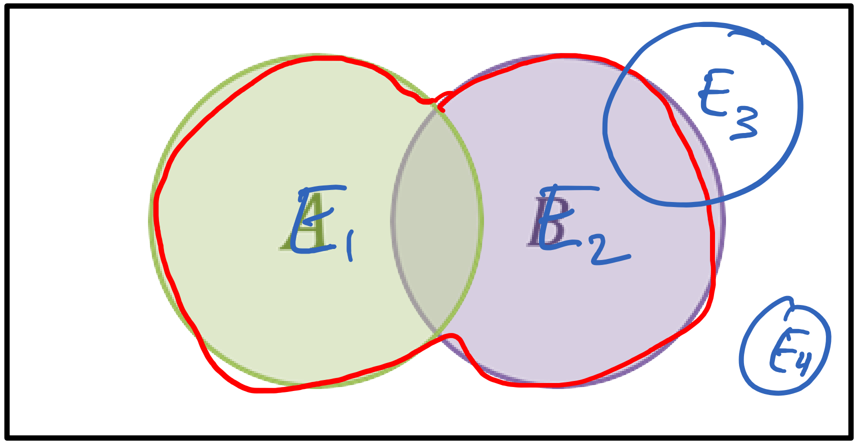
See Section 1.10 of https://arxiv.org/pdf/1801.06733.pdf for many variations of Chernoff bounds

Another useful tool:

Union Bound

- for arbitrary events A, B

$$\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$$

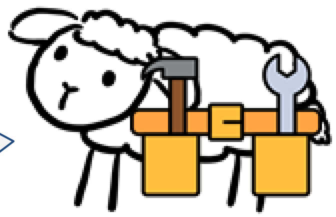


- More generally, for arbitrary events $E_1, \dots, E_n$

$$\Pr[\cup_i E_i] \leq \Sigma_i \Pr[E_i]$$

Now back to Balls and Bins

Let's apply the new tools in our pocket (Chernoff and union bounds)



n balls (clients)

$X_1 = 1$ $X_2 = 1$ $X_3 = 0$ $X_4 = 1$ $X_5 = 3$ $X_6 = 0$ $X_7 = 1$ $X_8 = 0$ $X_9 = 2$

n bins (servers)

Recall $\mathbf{X_j}$ is the number of balls in bin j. Earlier we showed: $E[\mathbf{X_j}] = 1$

Recall $\mathbf{X_{ij}}$ is an indicator r.v. for whether ball i is in bin j.

Observation: $\mathbf{X_j} = \sum_{i=1}^{n} \mathbf{X_{ij}}$.

⇒ $\mathbf{X_j}$ is the sum of <u>independent</u> r.v.'s with values in [0,1] so we can apply Chernoff.

Now back to Balls and Bins

Markov instead of Chernoff gives prob 1/log n, which becomes n/log n after union bound. Not useful!

n balls (clients)

$X_1 = 1$ $X_2 = 1$ $X_3 = 0$ $X_4 = 1$ $X_5 = 3$ $X_6 = 0$ $X_7 = 1$ $X_8 = 0$ $X_9 = 2$

~~k~~ n bins (servers)

$\mu = 1$ $\lambda = 6 \ln n$ $[Y - 1 \geq 6 \ln n]$

Apply Chernoff: Pr[**Xj** ≥ 1 + 6 ln n] ≤ $e^{-\frac{\lambda \mu}{3}} = e^{-\frac{6 \ln n}{3}} = \left(e^{\ln n}\right)^{-2} = \frac{1}{n^2}$

$Pr\left[\bigcup_j X_j \geq 1 + 6 \ln n\right] \leq \sum_{j=1}^{n} Pr[X_j \geq 1 + 6 \ln n] = \frac{n}{n^2} = \frac{1}{n}$

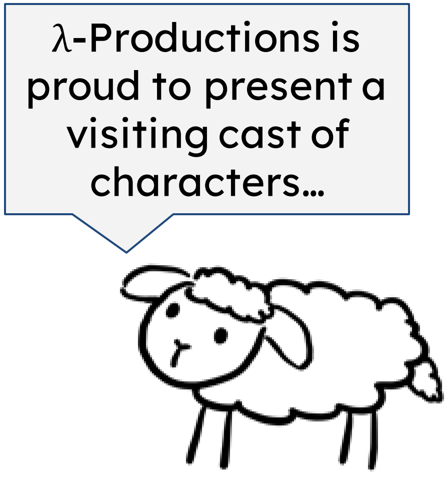
Apply Union: Pr[Exists j such that **Xj** ≥ 1 + 6 ln n] ≤

⇒ Original goal: with prob ≥ 1-1/n, fullest bin has O(log n) balls.

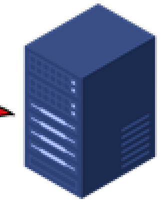
Fingerprinting

The scenario:

- You download a large file from a *untrusted* remote server
- The original file is from your friend
- You want to check that the version you downloaded hasn't be tampered with.
- The file is large so your friend can't send the whole thing to you directly, but they can send you a small **"fingerprint"** to help verify the authenticity



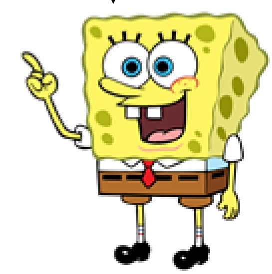
Beforehand, Alice and Bob agree on a **protocol** for how Alice will choose **M**, given **x**.



malicious adversary who knows protocol

y (downloaded file)

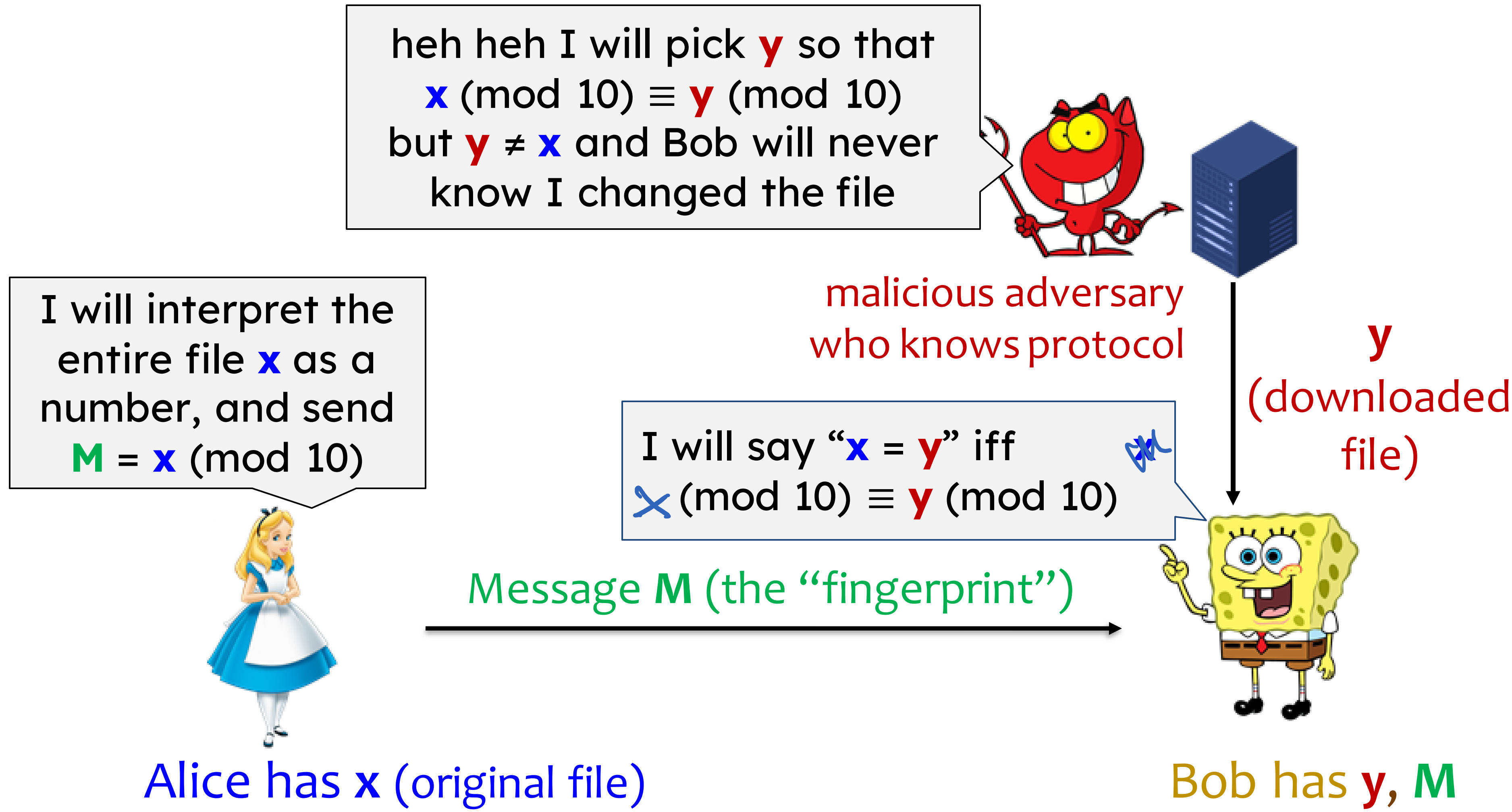
Message **M** (the "fingerprint")



Alice has **x** (original file)

Bob has **y**, **M**

Goal: We want message **M** as short as possible, while still ensuring that no matter how the adversary changes the file, Bob can check if **x** = **y**, given **M**.



Goal: We want message **M** as short as possible, while still ensuring that no matter how the adversary changes the file, Bob can check if **x** = **y**, given **M**.

Deterministic Protocols Don't Work

If $|M| < |x|$, then by the pigeonhole principle, there are two files $x1$, $x2$ that deterministically cause Alice to send the same message M_{bad}.

If Bob receives the message M_{bad} and downloads the file $y = x1$, Bob doesn't know if the original file was $x1$ or $x2$! He deterministically says either "$x = y$" (wrong if $x=x2$), or "$x \neq y$" (wrong if $x=x1$).

Therefore, all deterministic protocols require $|M| = |x|$ in the worst case.
This is useless because the file is too large to send!

But there is a good **randomized** protocol!

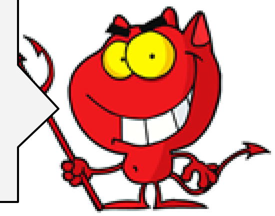
We will show:

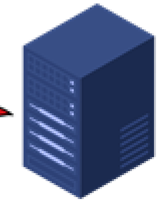
There is a randomized protocol such that for every **x**, **y**:

- $|\mathbf{M}| = O(\log \mathbf{n})$ ⇐ n is #bits of x
- If **x** = **y**, then Bob always says "**x** = **y**"
- If **x** ≠ **y**, then Bob detects that "**x** ≠ **y**" with prob ≥90%

1st attempt

heh heh I will pick **y** so that
$x \pmod p = y \pmod p$ for all p in [1..10].
$|x-y|$ is a multiple of every number in [1...10]



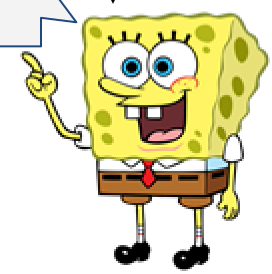


malicious adversary who knows protocol

y (downloaded file)

I will pick a random number p in [1..10] and send
M = (p, **x** (mod p))

I will say "**x** = **y**" iff
x (mod p) ≡ **y** (mod p)



Message **M** (the "fingerprint")

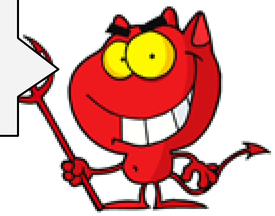
Alice has **x** (original file)

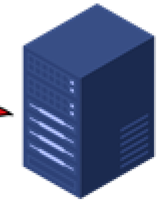
Bob has **y**, **M**

Goal: We want message **M** as short as possible, while still ensuring that no matter how the adversary changes the file, Bob can check if **x** = **y**, given **M**.

2nd attempt

I want to pick **y** so that |**x**-**y**| has as many factors in [1..n] as possible.





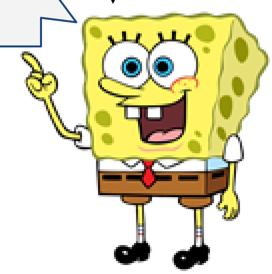
malicious adversary who knows protocol

y (downloaded file)

I will pick a random number p in [1..n] and send **M** = (p, **x** (mod p))

I will say "**x** = **y**" iff **x** (mod p) $\equiv$ **y** (mod p)

Message **M** (the "fingerprint")



Alice has **x** (original file)

Bob has **y**, **M**

Goal: We want message **M** as short as possible, while still ensuring that no matter how the adversary changes the file, Bob can check if **x** = **y**, given **M**.

Adversary wants |x-y| to have many factors, we want few factors.

How many factors does a number have? $N = 2 \cdot 3 \cdot 7^2 \cdot 13 \cdot 17^3$

Exponential in the number of prime factors.

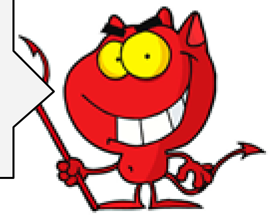
That seems like a lot…

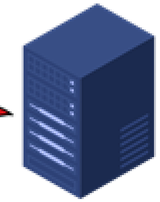
Insight: Alice picks only from the set of prime numbers!

Let's see why this works…

The actual protocol

I want to pick **y** so that |**x**-**y**| has as many **prime** factors as possible.
But no matter what **y** I pick I can't seem to fool Bob…





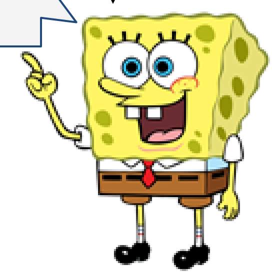
malicious adversary who knows protocol

y (downloaded file)

I will pick p randomly from the first 10n **prime** numbers and send **M** = (p, **x** (mod p))

I will say "**x** = **y**" iff **x** (mod p) ≡ **y** (mod p)

Message **M** (the "fingerprint")



$x = 253, \ y = 258$

$p = 2, 5, 11$

$\neq \ = \ \neq$

Alice has **x** (original file)

Bob has **y**, **M**

Goal: We want message **M** as short as possible, while still ensuring that no matter how the adversary changes the file, Bob can check if **x** = **y**, given **M**.

Our goal is to show:

For all **x**, **y** the protocol it such that:

- $|M| = O(\log n)$ ⇽ n is #bits of x
- If **x** = **y**, then Bob always says "**x** = **y**"
- If **x** ≠ **y**, then Bob detects that "**x** ≠ **y**" with prob ≥90%

M = (p, **x** (mod p)) where p is among the first 10n primes.

Question: How big is the kth prime number?

Answer: O(k log k) (complicated proof from number theory)

So, p = O(n log n).

⇒ |**M**| = O(#bits in p) = O(log (n log n)) = O(log n).

$\log(a \cdot b) = \log a + \log b$

$\log n + \cancel{\log \log n}$

Our goal is to show:

For all x, y the protocol it such that:

- $|M| = O(\log n)$ ⇐ n is #bits of x
- If x = y, then Bob always says "x = y"
- If x ≠ y, then Bob detects that "x ≠ y" with prob ≥90%

If x ≠ y, then Bob wrongly answers "x = y" if x (mod p) ≡ y (mod p), i.e. if p divides |x-y|.

Question: How many primes divide |x-y|?

Answer: ≤n. **Why?** $x \le 2^n$ $|x-y| \le 2^n$ every prime ≥ 2.

10% → $\frac{n}{10n}$ prob. Bob wrong

Alice chooses from 10n primes, and ≤n of them cause Bob to wrongly answer "x = y". Thus, Bob is right with prob ≥90%.

What if we wanted Bob to succeed 99% of the time instead of 90%?

Repeat the protocol

$\Pr[\text{fail 1 trial}] \le 0.1$

$\Pr[\text{fail 2 trial}] \le 0.01$

$\Pr(\text{succeed}] \ge 0.99$

Generally: c trials: $\Pr[\text{succeed}] \ge 1 - \frac{1}{10^c}$

Some takeaways from today

- Union & Chernoff bounds: used frequently in probabilistic analysis
  - Chernoff: “sum of independent r.v.’s is predictable”
  - Union: for calculating the prob that no “bad” event happens

- Randomization is necessary for frequently arising situations:
  - Load Balancing: allocating clients to servers
  - Fingerprinting: verifying authenticity of file from remote server