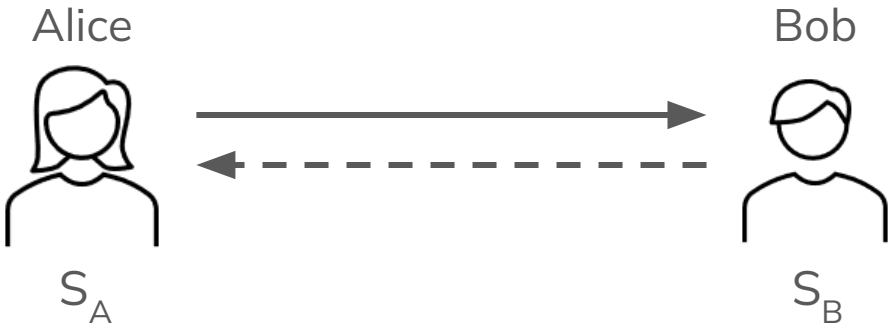
香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Communication Complexity

Wong Chun {WongChun1234}
2025-06-30

What is Communication Complexity?

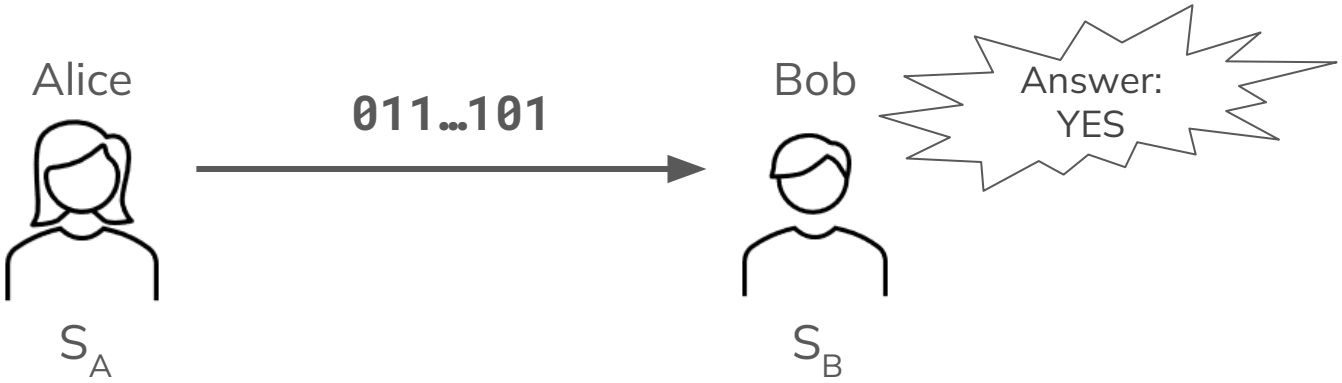


Alice owns some secret S_A while Bob owns some secret S_B.

Target: Solve some tasks collaboratively, by sending bits to each other.

Communication Complexity: The minimum amount of bits being sent.

One-way Communication Complexity

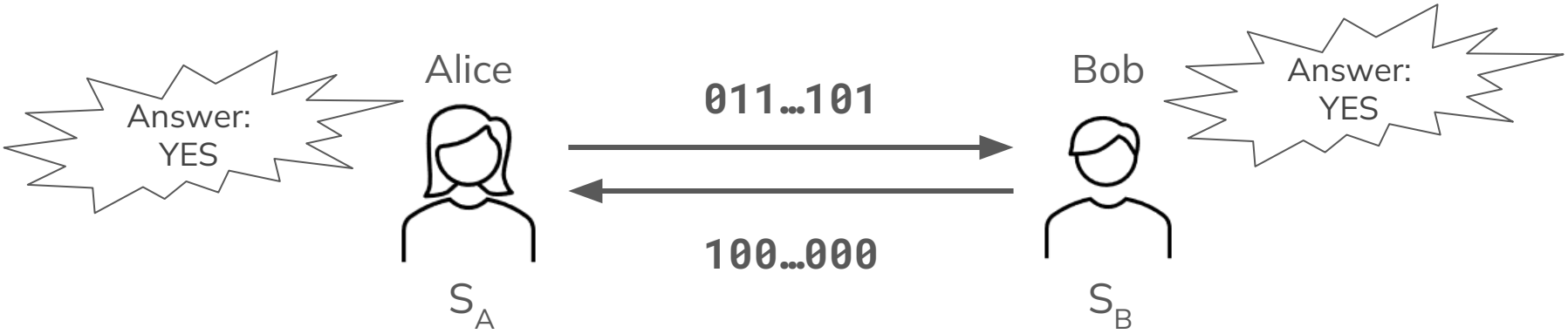


Only Alice can send bits to Bob.

Then, Bob needs to solve some task, based on the bits he got from Alice.

Communication Complexity: The minimum amount of bits being sent.

Two-ways Communication Complexity

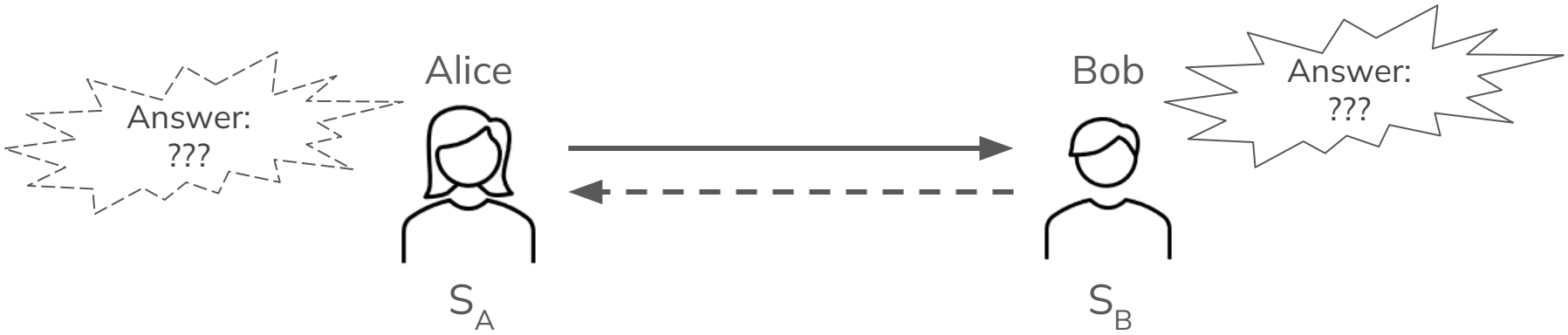


Both Alice and Bob can send bits to each other.

Both Alice and Bob need to solve the same task.

Communication Complexity: The minimum amount of bits being sent in total.

One-way vs Two-ways

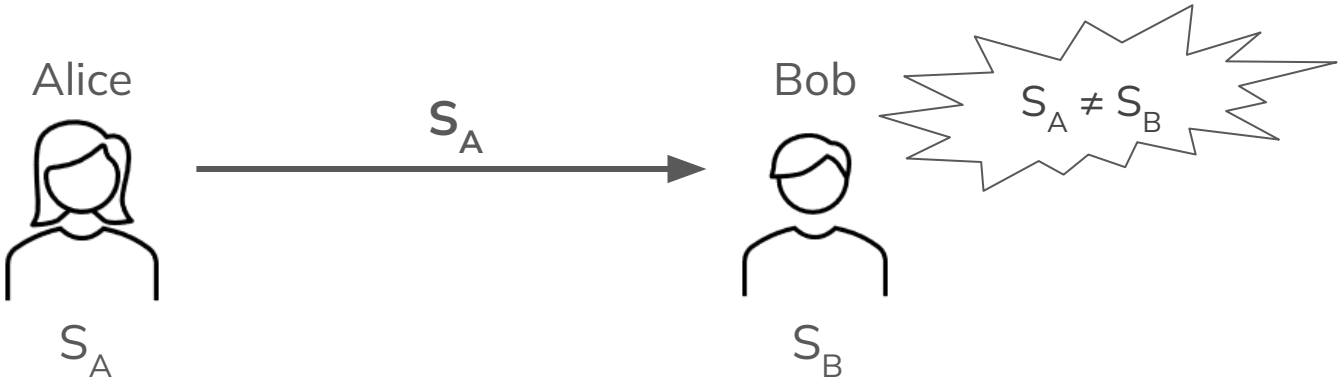


Suppose S_A and S_B are both 1-bit, and the task is XOR:

- If S_A and S_B are different, then answer YES. Otherwise, answer NO.

Think: What are the **One-way** and **Two-ways Communication Complexity**?

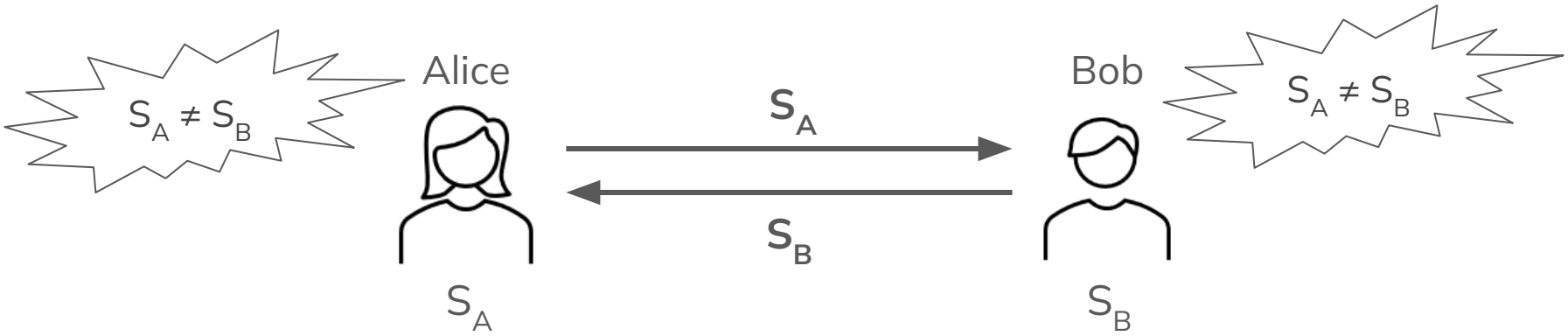
One-way vs Two-ways



One-way Communication Complexity: 1 bit

How? Alice can send S_A to Bob directly!

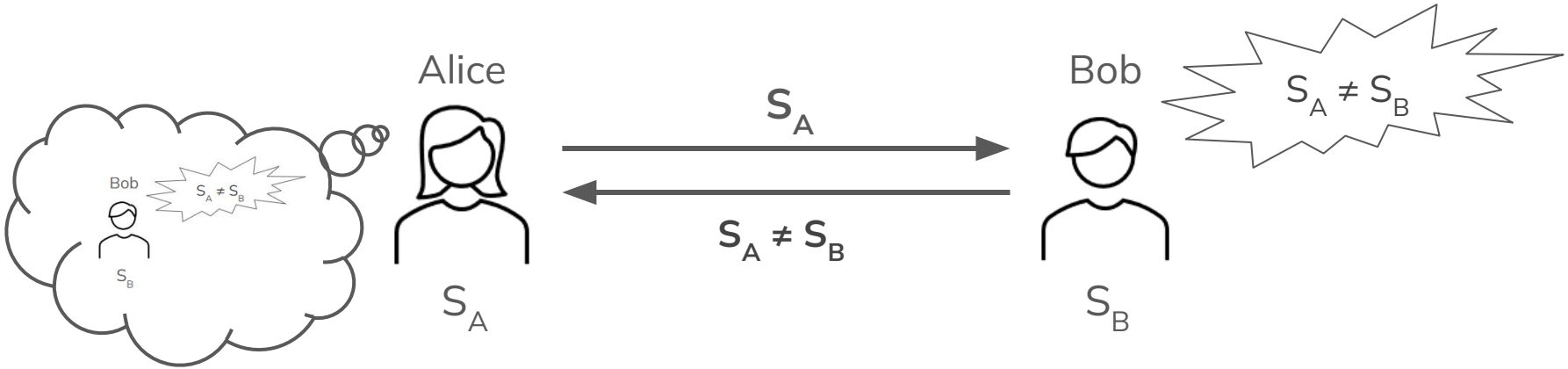
One-way vs Two-ways



Two-ways Communication Complexity: 2 bits

How? Alice can send S_A to Bob, then Bob can send S_B to Alice!

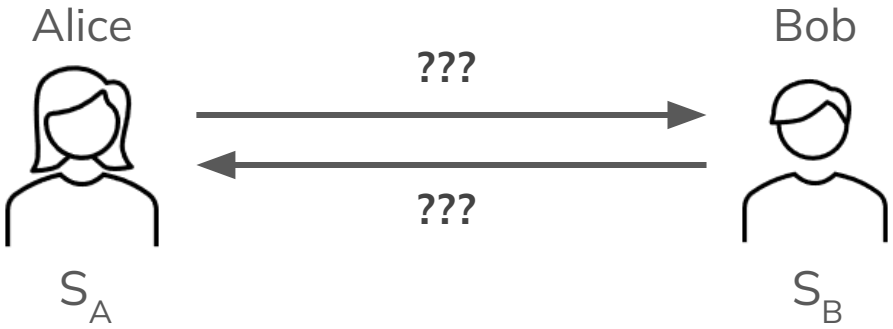
One-way vs Two-ways



Two-ways Communication Complexity: 2 bits

How? Alice can send S_A to Bob, then Bob can send **the answer** to Alice!

One-way vs Two-ways



Two-ways Communication Complexity: 2 bits

You would soon realise that there is not possible to complete this in 1 bits :(

Hence, we can **confirm** the communication complexity being 2 bits.

One-way Communication Complexity

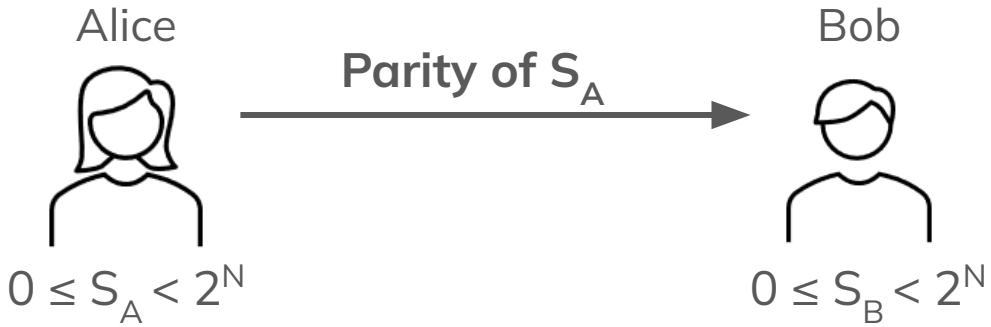
One-way | Parity

Extension on the XOR Task, `Parity`: Now S_A and S_B are N-bits binary strings.

- Answer the parity in $S_A \,||\, S_B$ (concatenation of S_A and S_B).

Think: What is the **One-way Communication Complexity**?

One-way | Parity



Simple: Alice can just send the parity of S_A to Bob in 1 bit!

Then, Bob can calculate the parity of S_B as well, then compare with that of S_A.

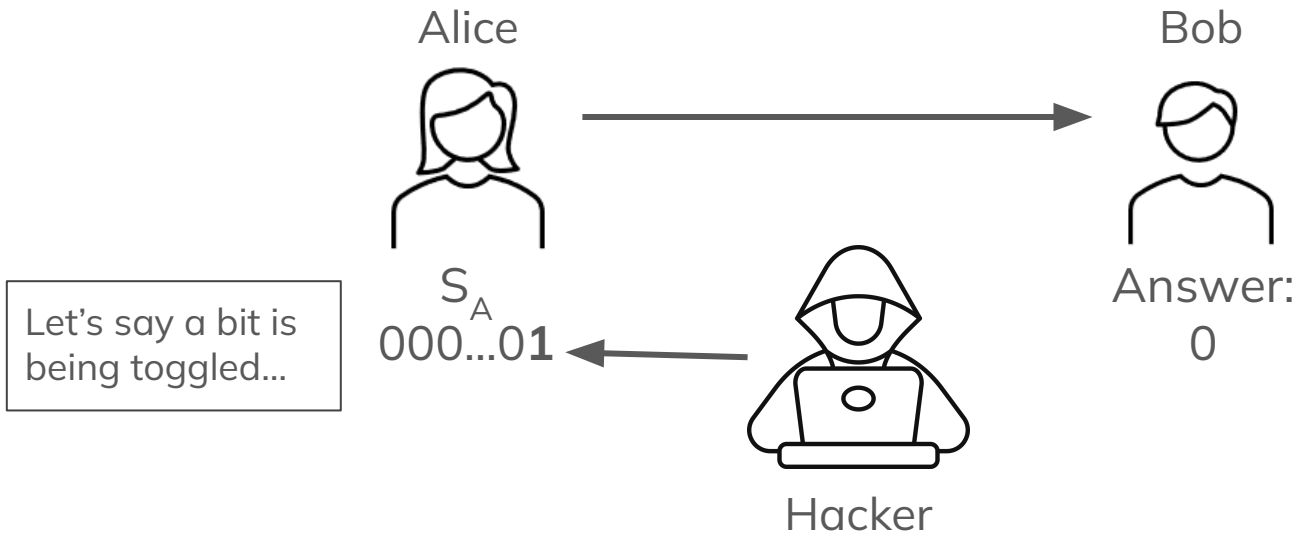
But how can we confirm that Bob cannot know with 0 bits?

One-way | Parity Hacker

One-way | Parity Hacker

The algorithm needs to be correct.

One-way | Parity Hacker



One-way | Parity Hacker

If Bob cannot distinguish two distinct S_A and S_A' and the answer changes depending on this, then Bob would get wrong in one of the cases.

Conclusion: It is not possible for Bob to derive the answer with 0 bits.

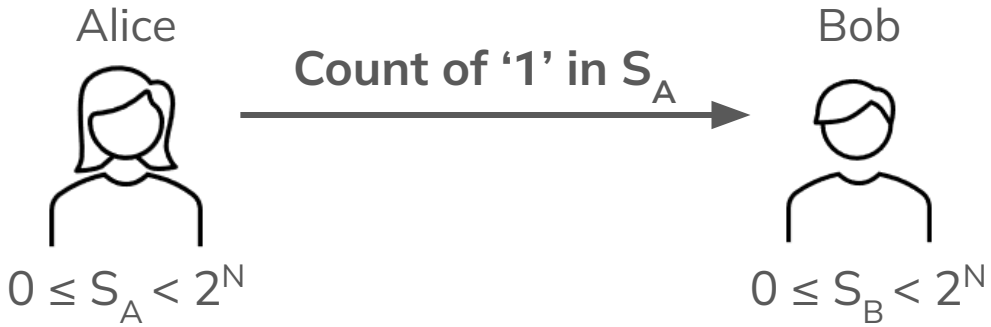
One-way | Majority

Another task, the `Majority` Task. Suppose S_A and S_B are N-bits binary string:

- In $S_A \,||\, S_B$, is it true that '1' appears strictly more than '`0`'?

Think: What is the **One-way Communication Complexity**?

One-way | Majority

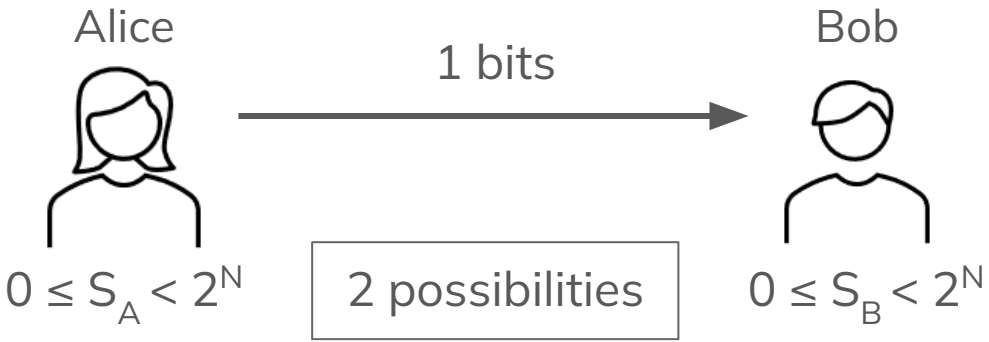


Also simple: Alice can send the count of '1' in S_A to Bob in $\lceil \log_2 (N+1) \rceil$ bits!

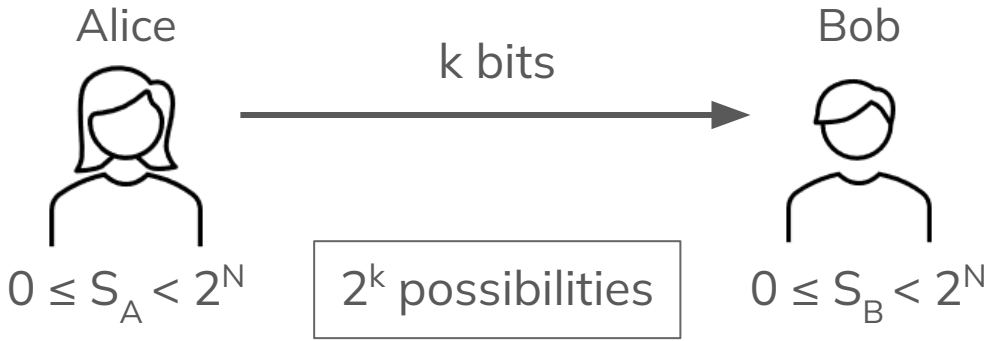
Can we do better?

One-way | Majority

One-way | Majority



One-way | Majority

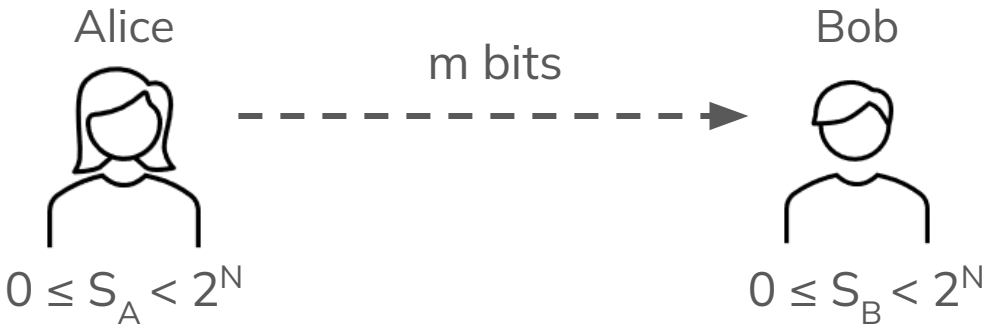


We can use the length property as well!

With k bits, we can send $1 + 2 + \ldots + 2^k = 2^{k+1} - 1$ different possibilities.

Therefore, we just need $2^{k+1} - 1 \ge N + 1$, or to send $k = \lceil \log_2(N + 2) - 1 \rceil$ bits.

One-way | Majority Hacker



Suppose Alice decides to send less bits, say $1 + 2 + \ldots + 2^m < N + 1$.

How can we hack the algorithm?

One-way | Majority Hacker

Consider the N + 1 inputs

{000...00, 000...01, 000...11, ..., 001...11, 011...11, 111...11}

Suppose the number of possible messages $1 + 2 + ... + 2^m$ is less than N + 1.

Then by **pigeonhole principle**, there must exist two inputs that maps to the same message.

Let's say the two messages M_1 and M_2 are with x '1's and y '1's (x < y) respectively. Bob can construct a length-N string with (N - x) '1's, then

- If M_1 is chosen, then answer is 0. If M_2 is chosen, the answer is 1.

The algorithm fails when less bits is used to communicate!

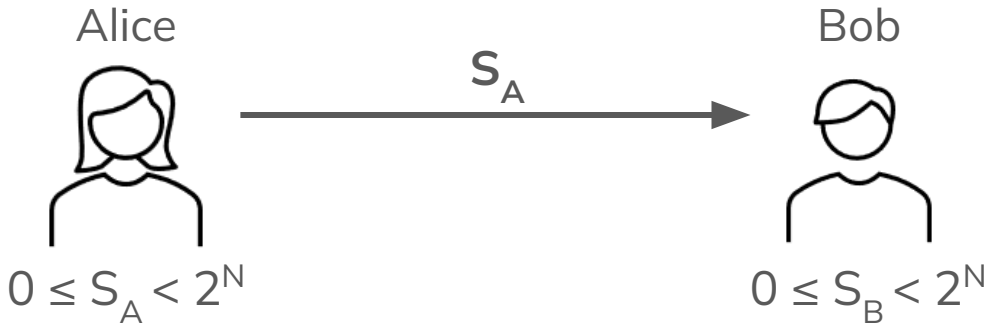
One-way | Equals

The fourth task, `Equals`. Suppose S_A and S_B are N-bits binary string:

- If S_A and S_B are equal, then answer YES. Otherwise, answer NO.

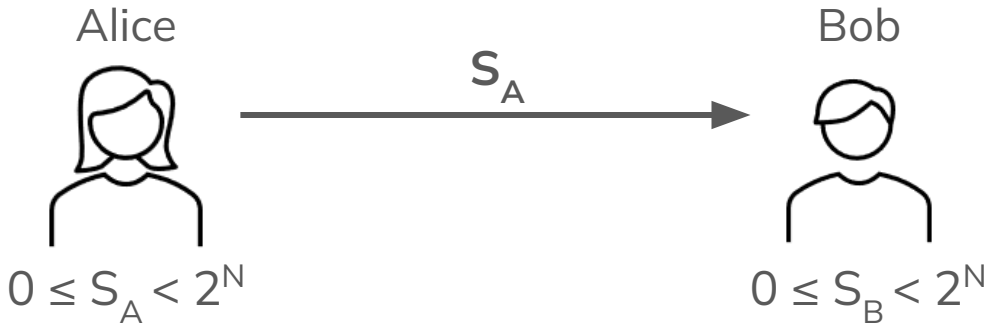
Think: What is the **One-way Communication Complexity**?

One-way | Equals



The best way seems to be directly sending the whole bitstring S_A with N bits!
With N - 1 bits, there are only $2^0 + 2^1 + \ldots + 2^{N-1} = 2^N - 1$ different possibilities.

One-way | Equals

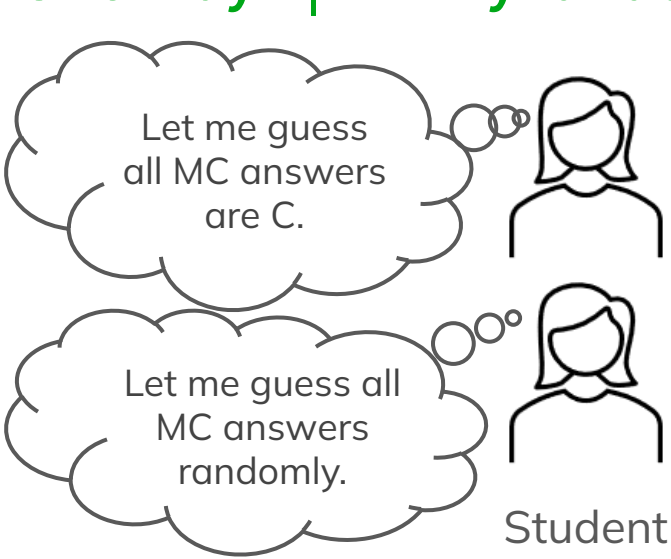


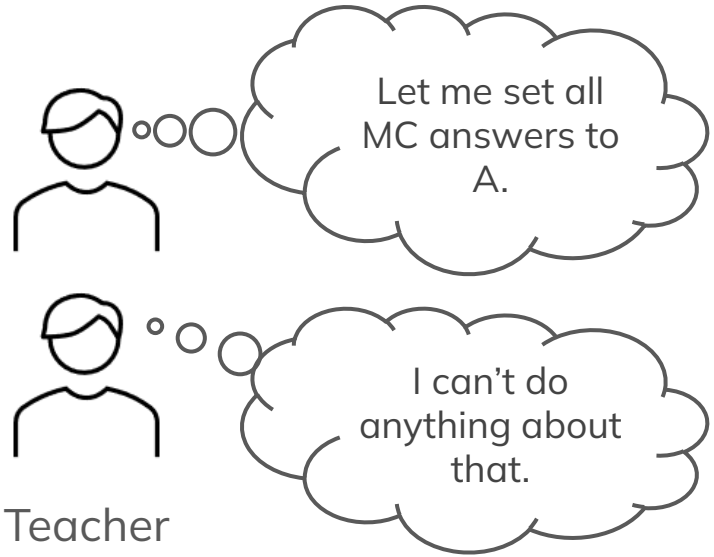
On the other hand, by **pigeonhole principle**, some message would give at least two possibilities of S_A. If S_B is either of them, Bob can't tell the answer. The **One-way Communication Complexity** is exactly $\lceil \log_2(N + 2) - 1 \rceil$ bits.

One-way | Equals, Randomized

Actually, if S_A and S_B are not equal, then likely we would know that early on. How can we use this to reduce the communication complexity, by using some randomized algorithms that is **allowed to fail for a small probability**?

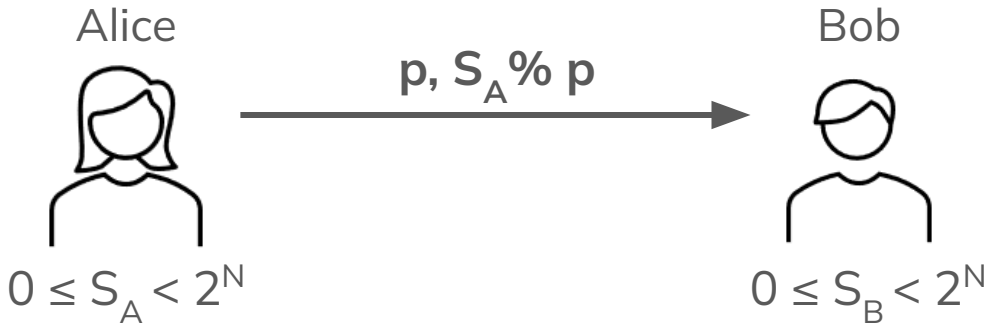
One-way | Why randomized may work?





With random, regardless of the answer you can get the answer correct 25% of the time. Deterministic gets 0% of the time.

One-way | Equals, Randomized



"Hash"-like Idea: Alice can send a **random** prime p and S_A mod p.

- If S_A mod p = S_B mod p, then answer YES. Otherwise, answer NO.
- How this works?

One-way | Equals, Randomized

"Hash"-like Idea: Alice can send a **random** prime p and S_A mod p.

- If S_A mod p = S_B mod p, then answer YES. Otherwise, answer NO.

Suppose the first N^2 primes are $2 \le p_1 < p_2 < \ldots < p_{N^2}$.
Let's say we pick p from $\{p_1, p_2, \ldots, p_{N^2}\}$ randomly.

How likely does the algorithm give correct result?

One-way | Equals, Randomized

How likely does the algorithm give correct result?

- If $S_A = S_B$, then algorithm always returns YES.

- If $S_A \neq S_B$, the algorithm can returns either YES or NO.
  - Note that $S_A \bmod p = S_B \bmod p \Rightarrow S_A - S_B$ is divisible by p.
  - How many bad choices of p are there?
  - $|S_A - S_B| < 2^N$ has less than N prime divisors, so at most N bad choices for p.
  - We choose p from $\{p_1, p_2, ..., p_{N^2}\}$, so there are at least $N^2 - N$ good choices.
  - Algorithm returns NO without probability at least $(N^2 - N) / N^2 = 1 - 1 / N$.

One-way | Equals, Deterministic vs Randomized

Deterministic

- Lower bound of N bits.
- Must give correct result for all input (Otherwise can be hacked)

Randomized

- Send O(log p) = O(log N) bits.
- Correctness depends on **random choice** p (regardless of the input)

Two-way Communication Complexity

Two-ways | Equals

Once again, let S_A and S_B be N-bits binary string.

- Question: Are S_A and S_B equal?

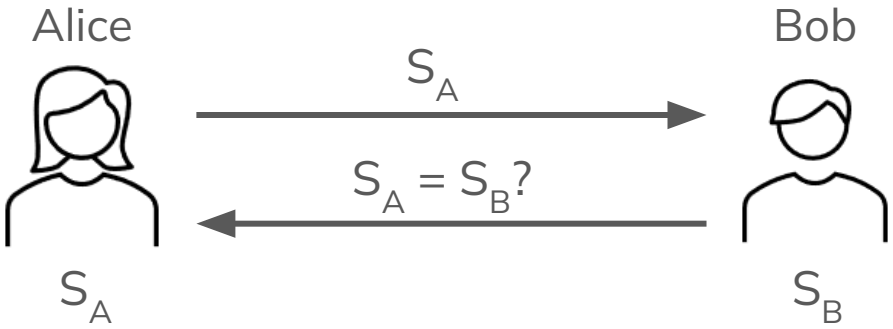
This time, both Alice and Bob need to answer.

What is the **Two-ways Communication Complexity**?

What we need:

- Upper bound (Construction by algorithm)
- Lower bound (Proof of why less bits won't work)

Two-ways | Equals



The algorithm is easy: Alice can simply send S_A to Bob, and then Bob can send back the answer. It takes N + 1 bits for communication.

Two-ways | Disjointness

Once again, let S_A and S_B be N-bits binary string.

- Question: Are S_A and S_B disjoint (bitwise AND = 0)?

Both Alice and Bob need to answer.

What is the **Two-ways Communication Complexity**?

What we need:

- Upper bound (Construction by algorithm)
- Lower bound (Proof of why less bits won't work)

Two-ways | Disjointness

Once again, a simple algorithm: Alice can simply send S_A to Bob, and then Bob can send back the answer. It takes N + 1 bits for communication.

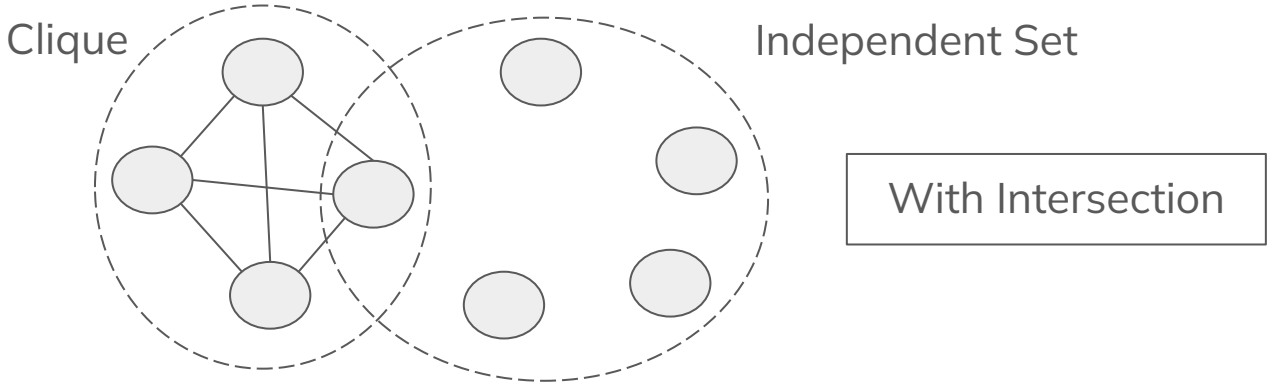
Two-ways | Clique Independent-Set

Now, the third task. Given a graph of N nodes, which:

- Alice is given a **Clique** (exists an edge between every nodes)
- Bob is given an **Independent Set** (no edge exists between every nodes)

Both Alice and Bob knows the structure of the graph.

Determine if the clique and the independent set has intersection.



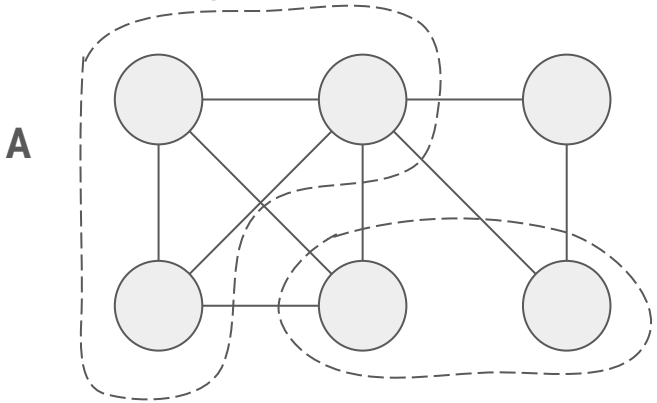
Two-ways | Clique Independent-Set

Now, the third task. Given a graph of N nodes, which:

- Alice is given a **Clique** (exists an edge between every nodes)
- Bob is given an **Independent Set** (no edge exists between every nodes)

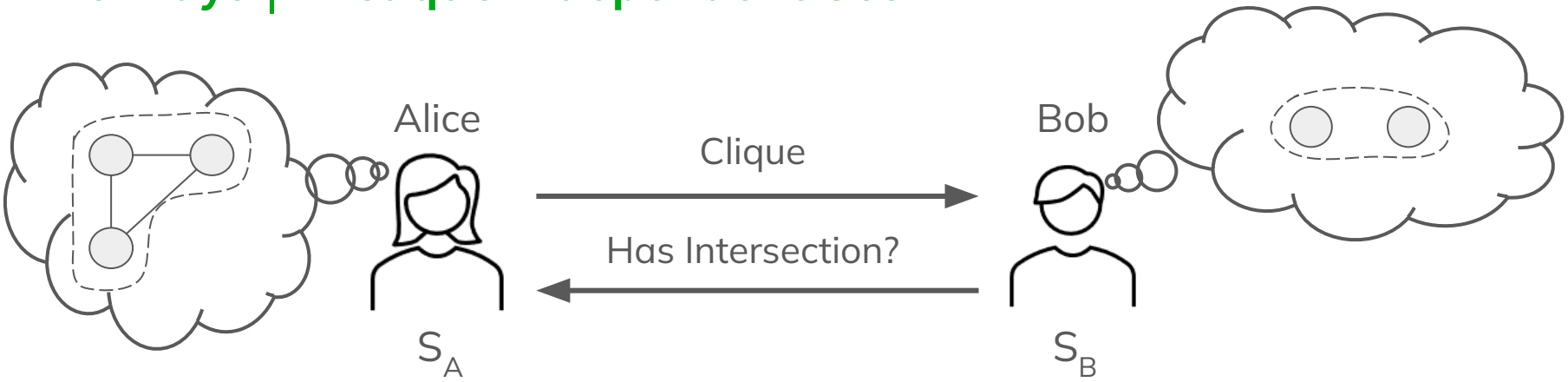
Both Alice and Bob knows the structure of the graph.

Determine if the clique and the independent set has intersection.



Without Intersection

Two-ways | Clique Independent-Set



Once again, a simple algorithm: Alice can send a “boolean array” of N-bits to Bob, where the nodes in her clique is marked as true. Then, Bob can send back the answer. It takes N + 1 bits for communication.

Group Discussion!

Task 1. `Equals`

- Alice, Bob: N-bit binary strings; Answer: are the two strings equal

Task 2. `Disjointness`

- Alice, Bob: N-bit binary strings; Answer: bitwise AND = 0 or not

Task 3. `Clique Independent-Set`

- Alice: Clique, Bob: Independent set; Answer: has intersection or not

Actually, one of the algorithms described in the previous pages is non-optimal.
Which of the above tasks is with **two-way communication complexity of ≤ N**?
Fastest group who gets the answer (**with algorithm**) gets a card pack!

Group Discussion!

Hint 1.

- One problem is a subset of another. Which problem is the relatively easier one?

Hint 2.

- Try a D&C approach. Eliminate half of the possibilities in one round

Hint 3.

- Clique has high degree while independent set has low degree. How do we exploit this?

Group Discussion!

Task 1. `Equals`

- Alice, Bob: N-bit binary strings; Answer: are the two strings equal

Task 2. `Disjointness`

- Alice, Bob: N-bit binary strings; Answer: bitwise AND = 0 or not

Task 3. `Clique Independent-Set`

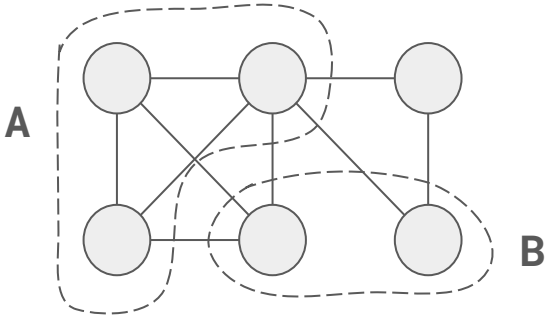
- Alice: Clique, Bob: Independent set; Answer: has intersection or not

Let's go through the tasks one-by-one!

Two-ways | Clique Independent-Set

How to solve the task `Clique Independent-Set`?
Yes, we can definitely do better than N + 1 bits.

Let K be the remaining nodes that can be intersection.

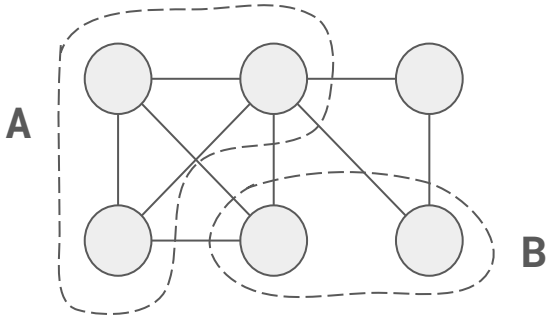


Two-ways | Clique Independent-Set

Let K be the remaining nodes that can be intersection.

Case I. Bob owns a node of degree > K / 2.

- Bob sends the node label to Alice.
- If Alice owns that, report YES.
- Otherwise, we **remove the node and its neighbours**.
- K becomes at most half of original after the removal.

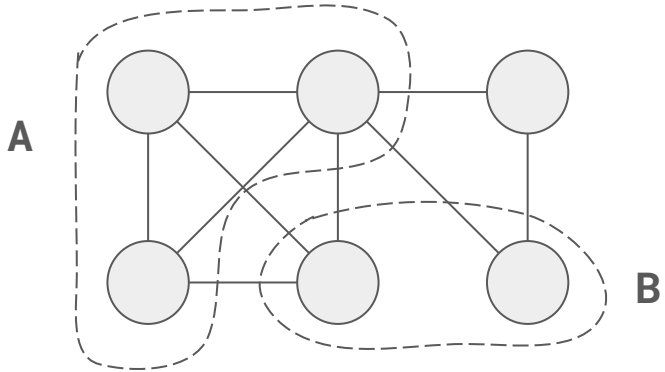


Two-ways | Clique Independent-Set

Let K be the remaining nodes that can be intersection.

Case II. Alice owns a node of degree ≤ K / 2.

- Alice sends the node label to Bob.
- Then, Bob knows the **intersecting node must be adjacent to that node**.
- Once again, we can restrict the graph to around K / 2 vertices.



Two-ways | Clique Independent-Set

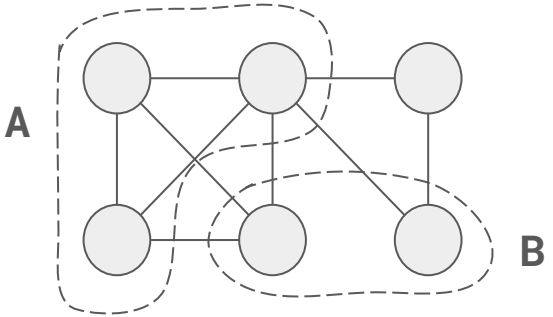
Let K be the remaining nodes that can be intersection.

Case I. Bob owns a node of degree > K / 2.

Case II. Alice owns a node of degree ≤ K / 2.

Case III. Neither of above.

- How to handle?



Two-ways | Clique Independent-Set

Let K be the remaining nodes that can be intersection.

Case I. Bob owns a node of degree > K / 2.

Case II. Alice owns a node of degree ≤ K / 2.

Case III. Neither of above.

- Simply return **NO**, since the nodes own by Alice and Bob must be disjoint!

Since we need to

- do this O(log N) times, and
- sending each node label costs us O(log N) bits,

In total we would need $O(\log^2 N)$ bits to communicate.

Note: This doesn't mean that the Communication Complexity is $O(\log^2 N)$!

The current best known lower bound is $\log^2 N$ / polylog(log N), but this is not known to be the upper bound.

Two-ways | Clique Independent-Set

Since we need to

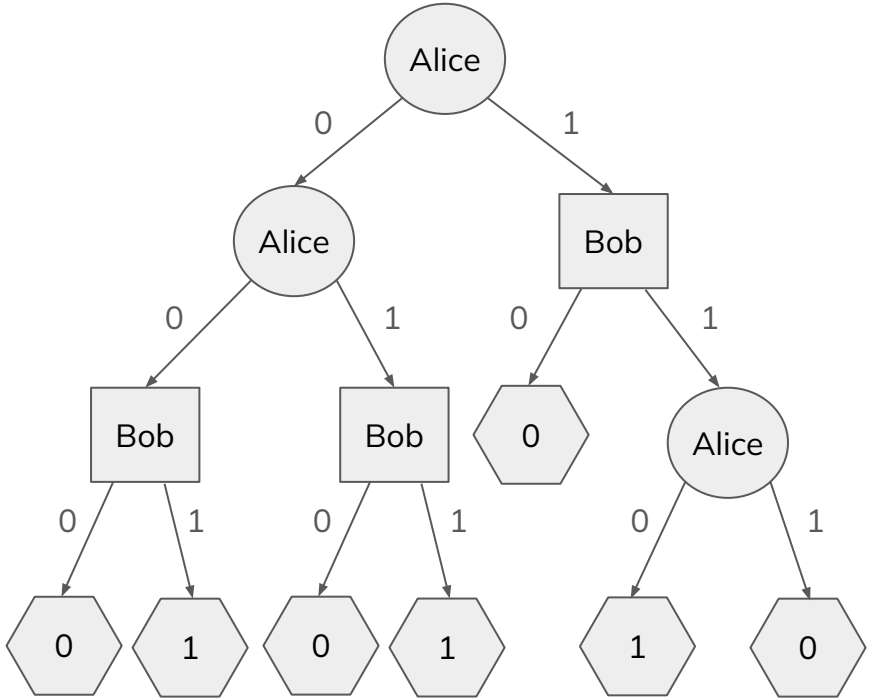
- do this O(log N) times, and
- sending each node label costs us O(log N) bits,

In total we would need $O(\log^2 N)$ bits to communicate.

Conclusion: Two-way communication is not just a trivial extension of one-way communication. Interaction could make two-way easier than one-way!

Let's move on to some tools that we can use in general for two-ways communication complexity...

Protocol Trees

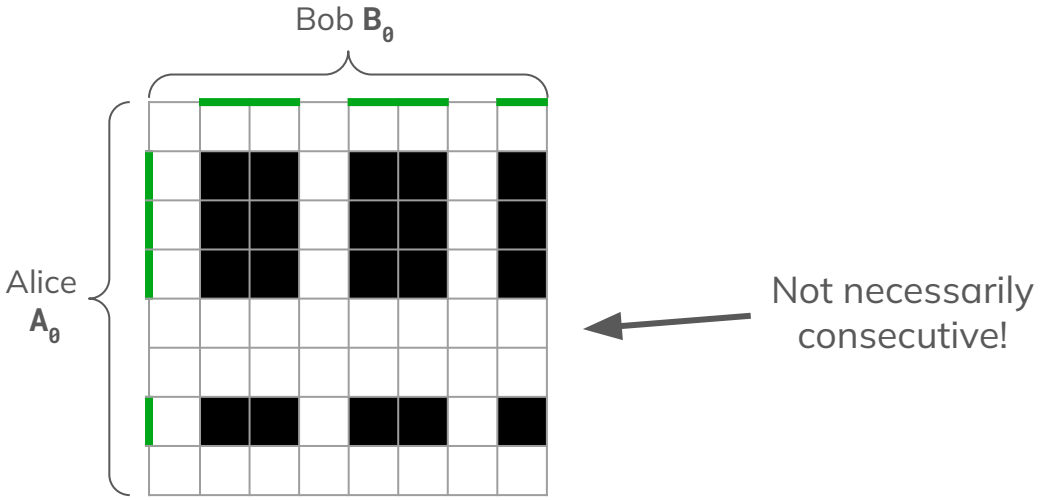


In fact, we can visualise the communication process with a tree.

Alice and Bob can follow the edges of this protocol tree to determine the answer in a deterministic way.

Rectangles

Next, let's introduce the concept of **Rectangles**.



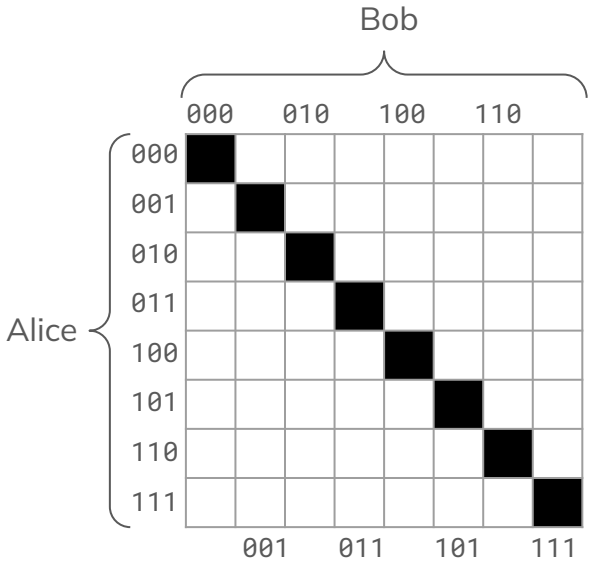
A **Rectangle** is the set of intersecting cells between some rows and columns.

Rectangles

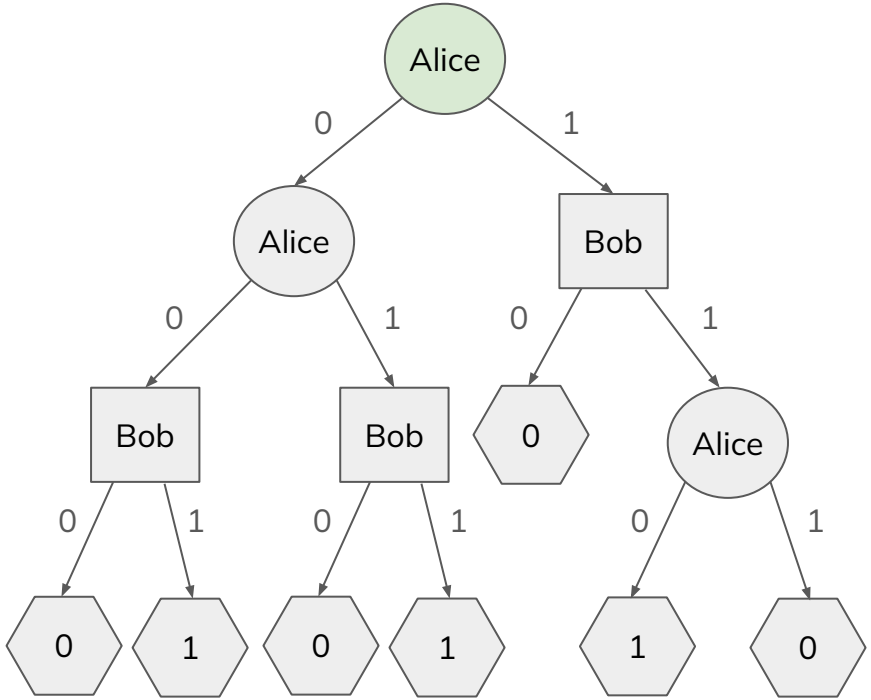
Black cells correspond to YES, whereas
White cells correspond to NO.

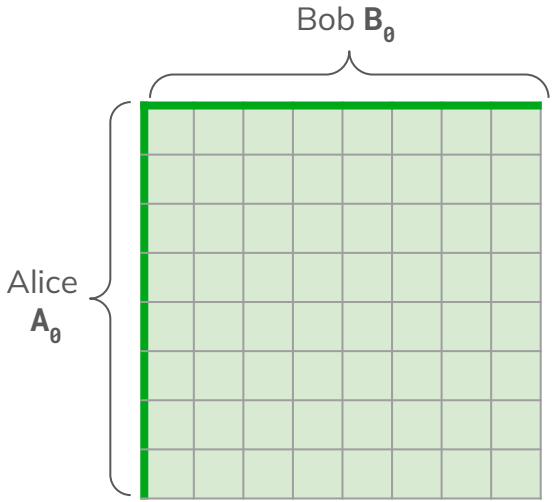
Observation 1: At each moment, the remaining possibilities corresponds to a **Rectangle**.

Observation 2: For each bit sent, we can cut the rectangle into at most **two sub-rectangles**.

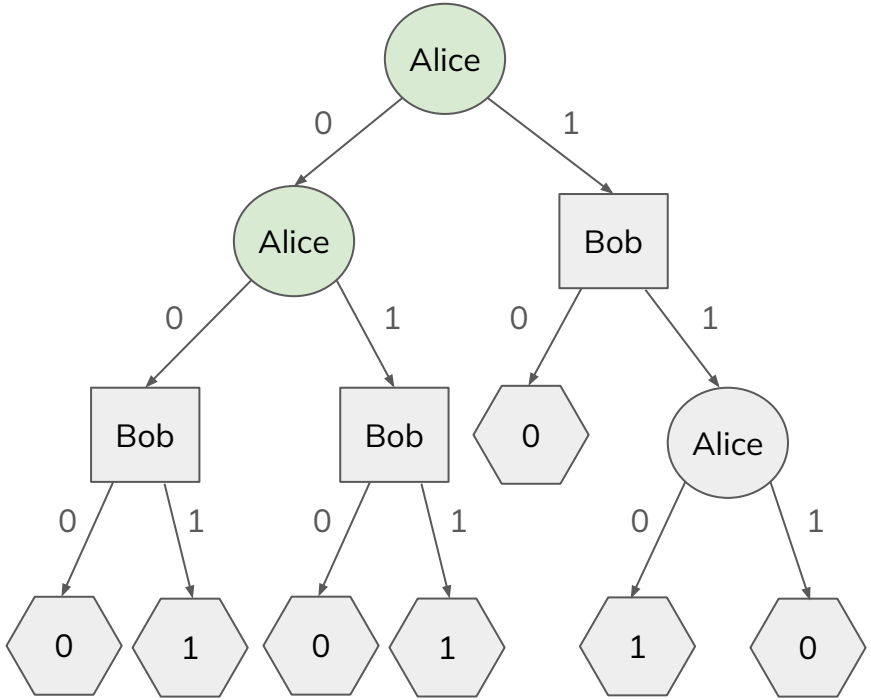


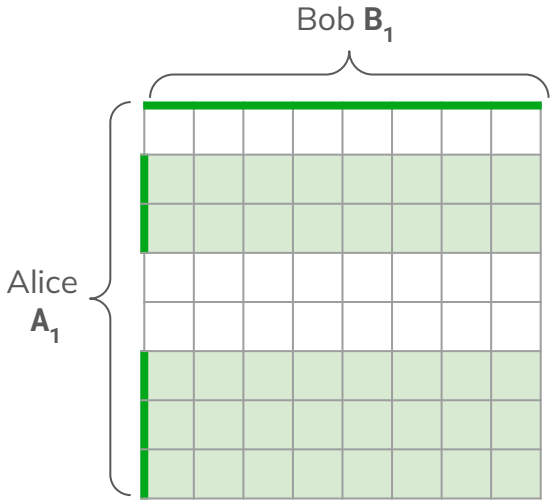
Protocol Trees and Rectangles



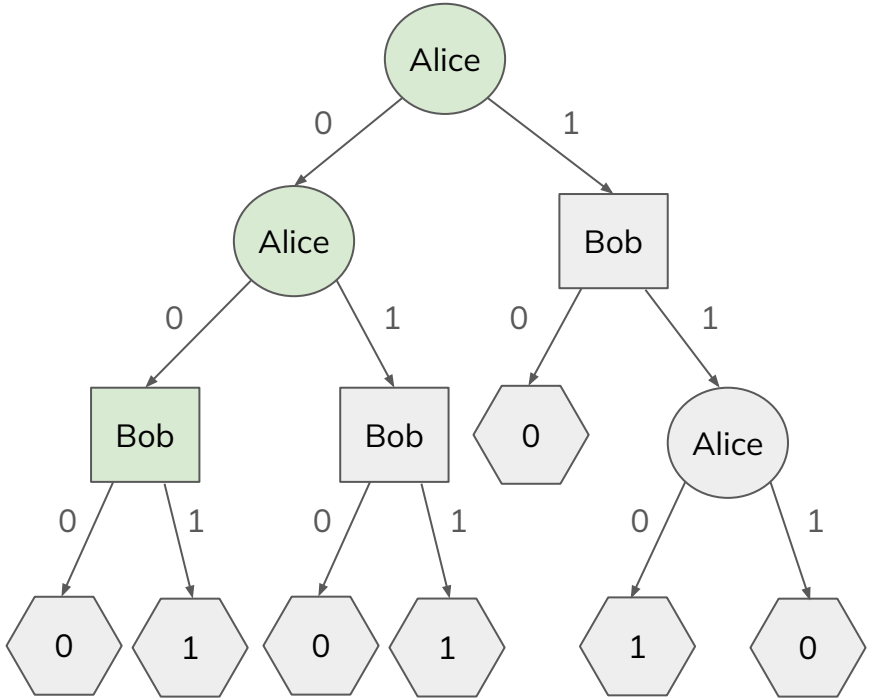


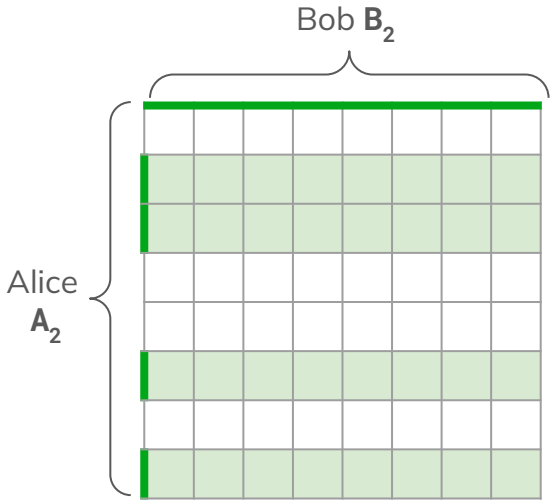
Protocol Trees and Rectangles



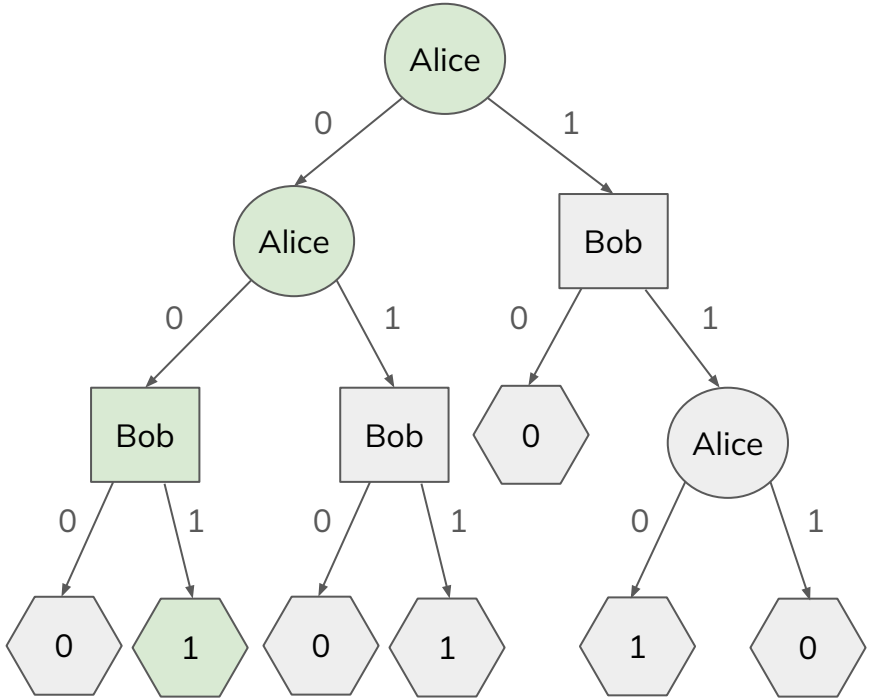


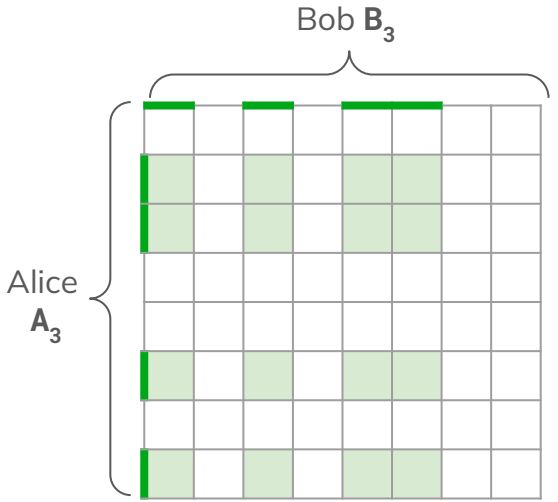
Protocol Trees and Rectangles



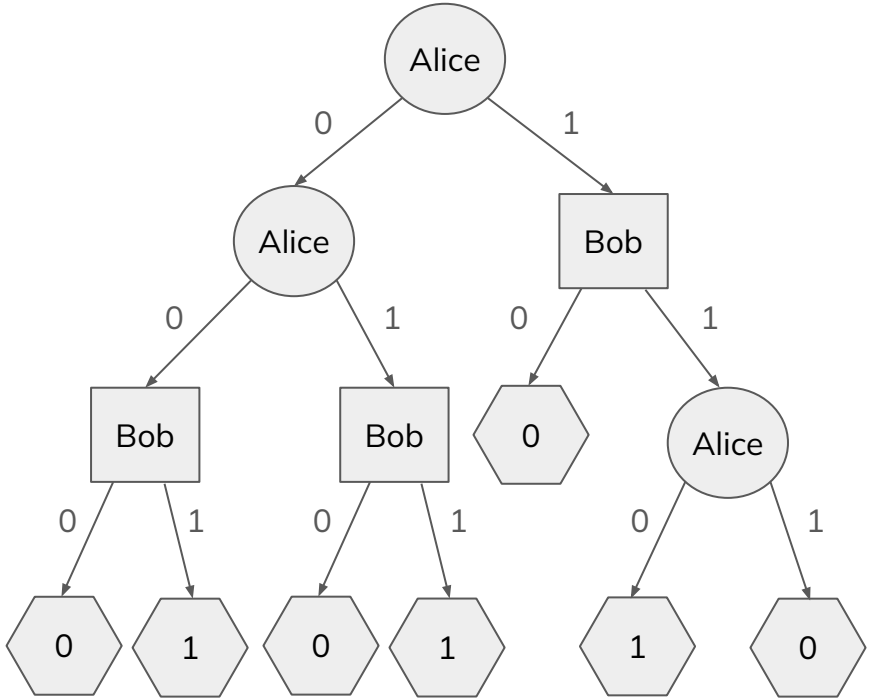


Protocol Trees and Rectangles





Protocol Trees and Rectangles



In sense of Protocol Tree, we can say

Observation 1 becomes

Each node is a Rectangle.

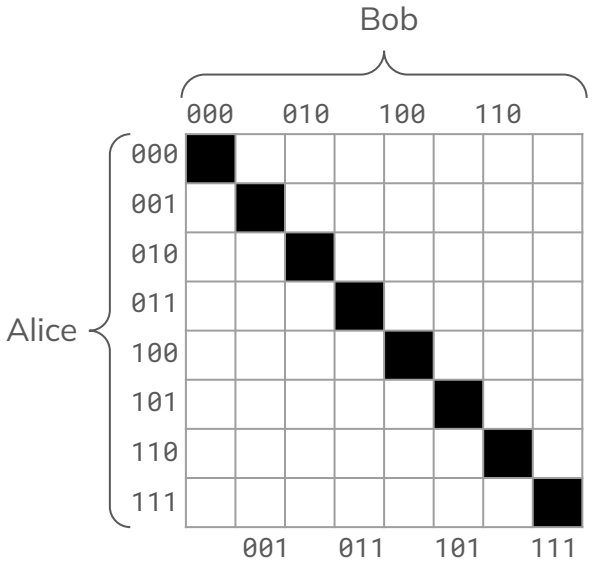
Observation 2 becomes

Protocol Tree is a Binary Tree.

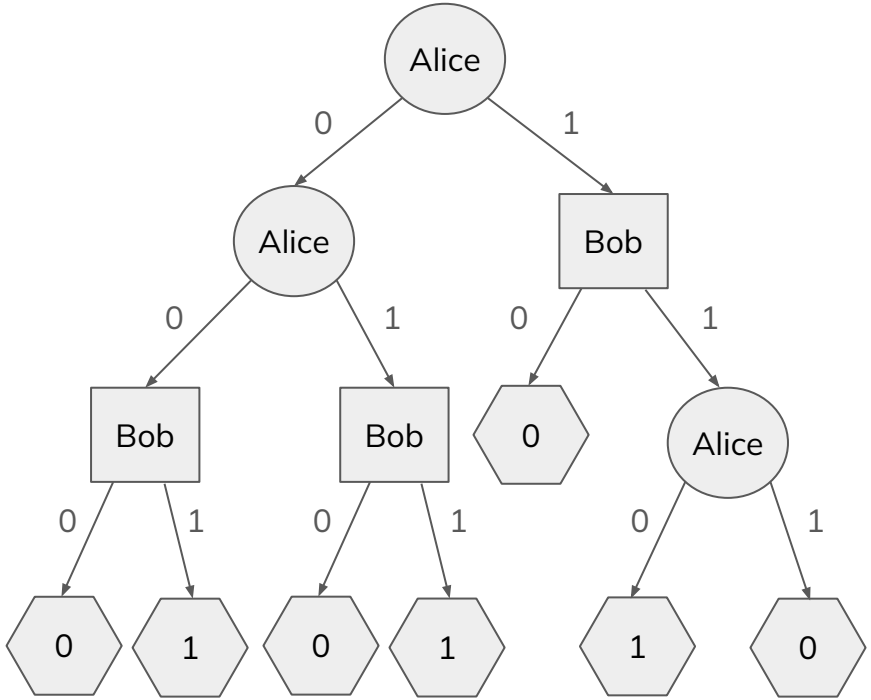
Monochromatic Rectangles

Observation 3: We can stop dividing iff the **rectangle** becomes **monochromatic**.

If the rectangle contains both black and white, Then the answer can be either YES or NO.
Alice and Bob cannot give a certain answer.



Monochromatic Rectangles

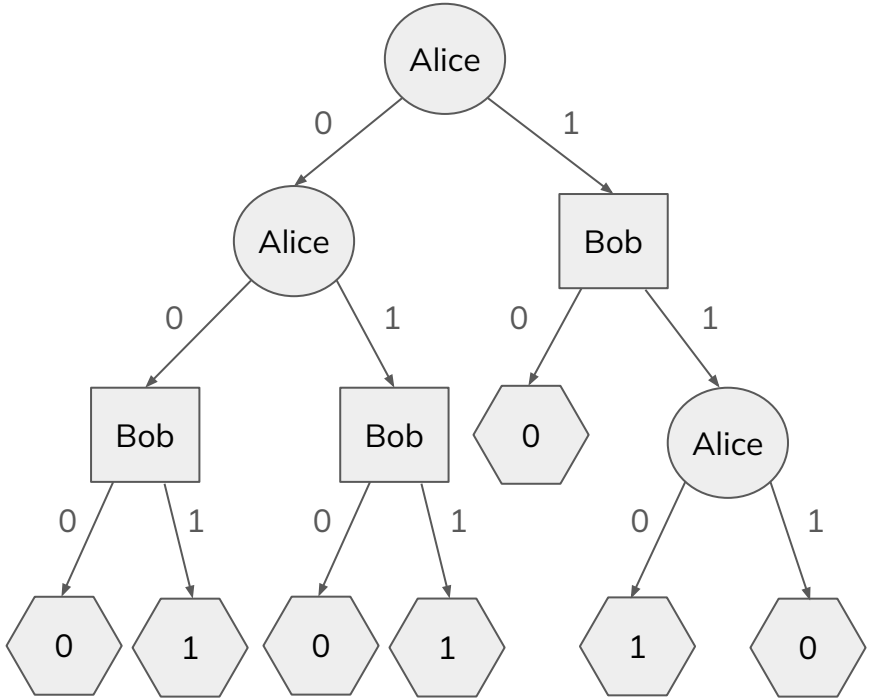


Translate back to Protocol Trees:
Observation 3 becomes
Each leaf node is monochromatic.

In other words, **Protocol Tree** essentially **cuts the Rectangle into monochromatic rectangles**!

For a communication complexity of H bits (height of protocol tree = H), at most 2^H monochromatic rectangles are formed.

Rectangle Partitions

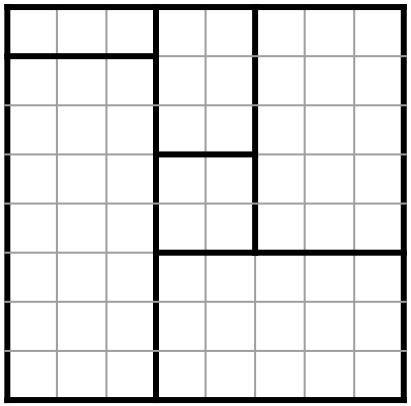


For a **communication complexity of H bits** (height of protocol tree = H), at most 2^H monochromatic rectangles are formed.

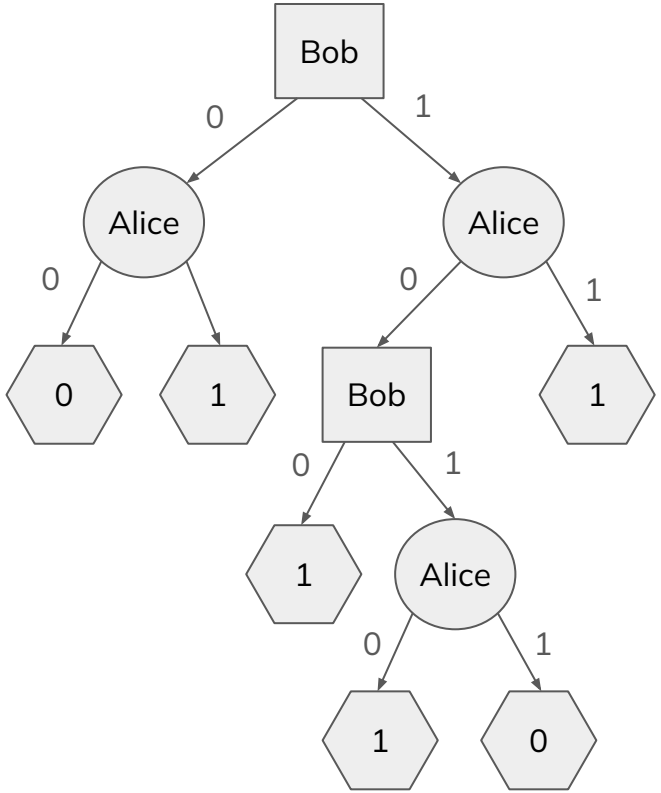
In other words:

If we cannot cut the rectangle into **≤ 2^H monochromatic rectangles**, its complexity is greater than H bits.

Rectangle Partitions



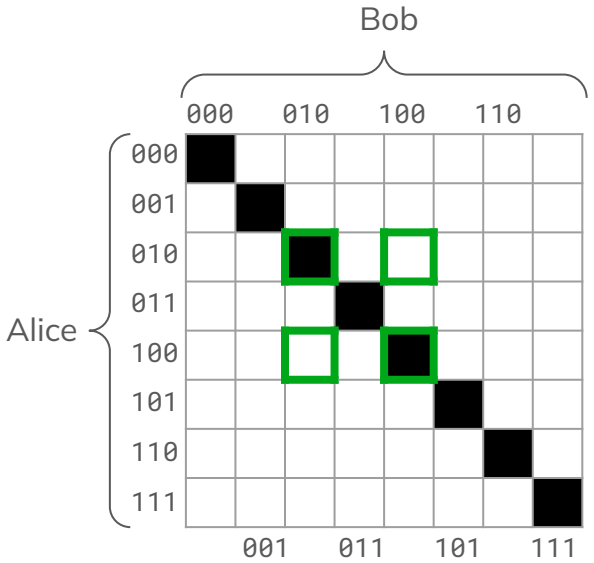
⇔



Two-ways | Equals

How can we apply those ideas back to our tasks?
First, the task EQUALS:

Note that black cells (x, x) and (y, y) where x ≠ y cannot be in the same **monochromatic** rectangle, since that rectangle must also contain white (x, y).



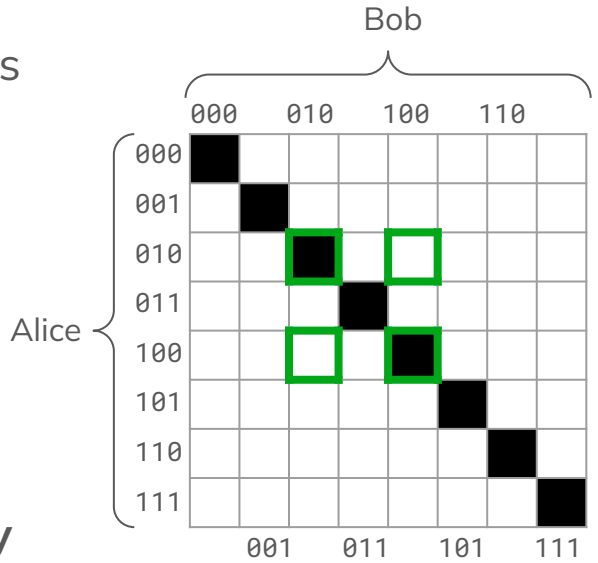
Two-ways | Equals

Therefore, we need to cut the rectangle into at least **2^N (black) + 1 (white) rectangles** - which requires us at least N + 1 bits to communicate.

Since we have an algorithm using N + 1 bits:

- Alice send to Bob: **S_A in N bits**
- Bob send to Alice: result of **$S_A = S_B$ in 1 bit**

We can confirm that the **communication complexity** for the task is **exactly N + 1 bits**.



Two-ways | Disjointness

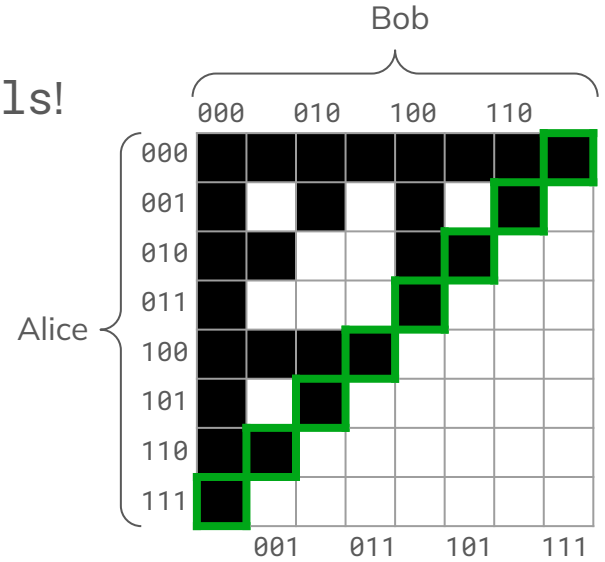
How about the task `Disjointness`?

Let's revisit the strategy we used to prove for `Equals`!

Claim 1. Cell (x, 2^N - 1 - x) is black for all x.

Examples (N = 3):

x	101	110	000
2^N - 1 - x	010	001	111

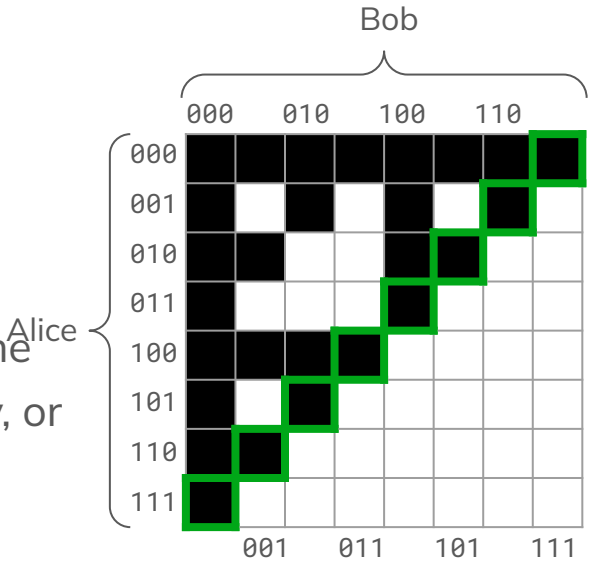


Two-ways | Disjointness

Claim 1. Cell (x, 2^N - 1 - x) is black for all x.

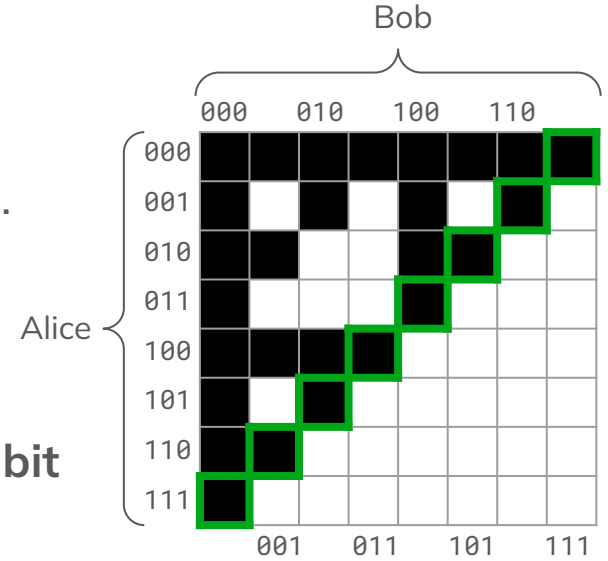
Claim 2. Cell (x, 2^N - 1 - x) and (y, 2^N - 1 - y) cannot be both in the same rectangle, for x ≠ y.

- For each bit position, '1' appears twice in
  x, 2^N - 1 - x, y, 2^N - 1 - y
- Since x + (2^N - 1 - y) ≠ 2^N - 1, there must exist some position where '1' appears in both x and 2^N - 1 - y, or In both y and 2^N - 1 - x!



Two-ways | Disjointness

Once again, we come to the conclusion that we need to cut the rectangle into at least **2^N (black) + 1 (white) rectangles**, which means the communication complexity is **at least N + 1 bits**.



Since we have an **algorithm using N + 1 bits**:

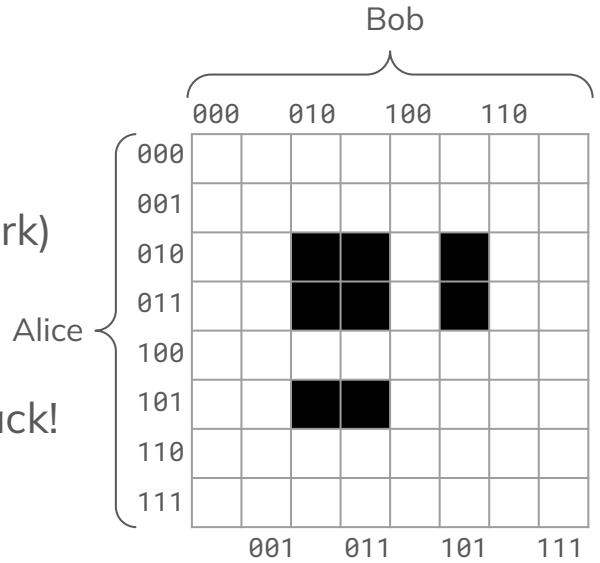
- Alice send to Bob: **S_A in N bits**
- Bob send to Alice: result of **S_A AND S_B = 0 in 1 bit**

We can confirm the two-ways communication complexity is **exactly N + 1 bits**.

Two-ways | Exercise!

- Given a **Rectangle** that looks like the right one.
- Find its **Communication Complexity**.
- Two parts:
  - Upper bound (Construction by algorithm)
  - Lower bound (Proof of why less bits won't work)
- Both parts are required!

- Fastest group who answer correctly gets a card pack!



Rectangle Partition to Protocol Tree

Given a **Protocol Tree**, we can determine a **Rectangle Partition**.

Can we convert **Rectangle Partition** to **Protocol Tree** instead?

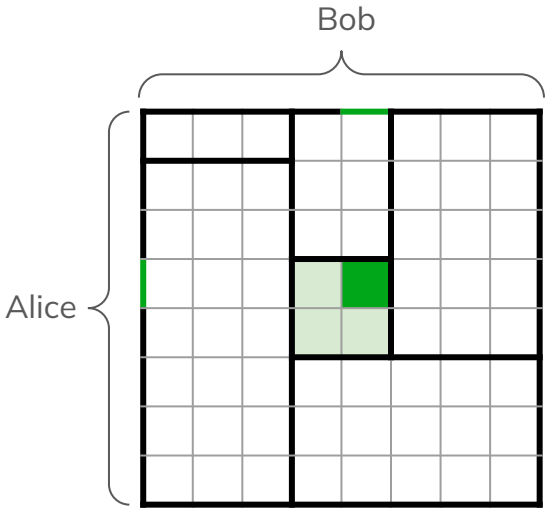
Turns out yes!

Claim. Given a Rectangle Partition of 2^c rectangles, we can construct a Protocol Tree of height $O(c^2)$.

- Fact. Alice or Bob can specify any rectangle in c bits, by sending its label.
- How can Alice and Bob pinpoint the exact rectangle?

Rectangle Partition to Protocol Tree

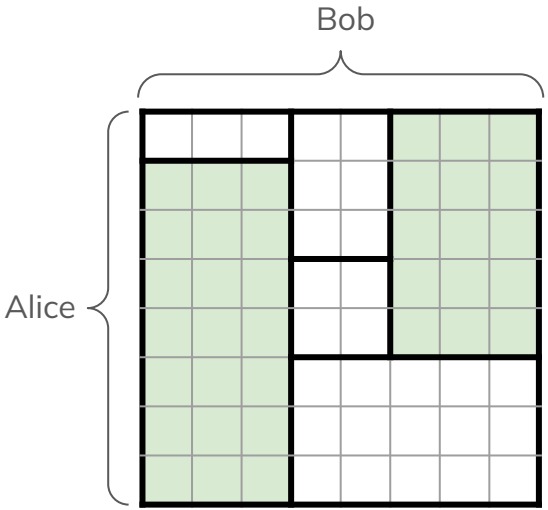
How can Alice and Bob pinpoint the exact rectangle?



Rectangle Partition to Protocol Tree

Definition:

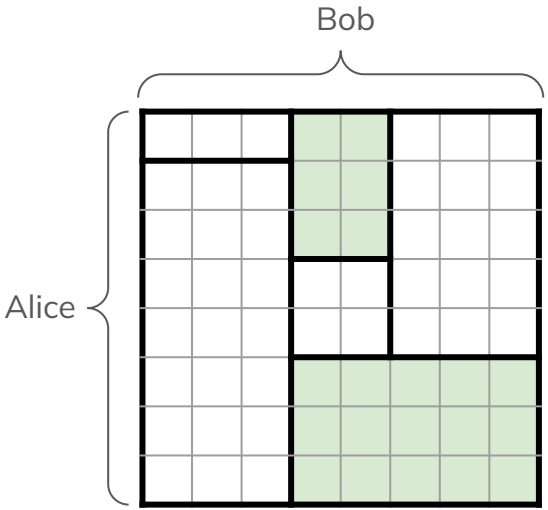
- Two rectangles **horizontally intersect** if some row(s) intersects both rectangles.



Rectangle Partition to Protocol Tree

Definition:

- Two rectangles **vertically intersect** if some column(s) intersects both rectangles.



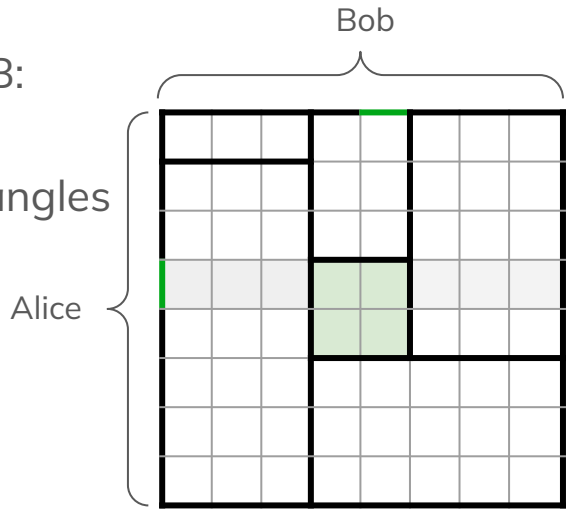
Rectangle Partition to Protocol Tree

Fact: No two distinct rectangles both intersect vertically and horizontally.

Why? Since that would mean two rectangles intersect!

For a specific input (x, y), we define for rectangle R = A × B:

- **Horizontally good**:
  - **Horizontally intersect** with at most **half** of rectangles
  - **A contains x**



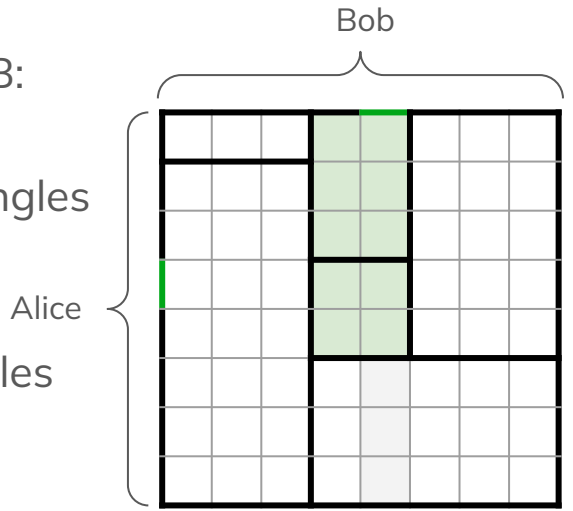
Rectangle Partition to Protocol Tree

Fact: No two distinct rectangles both intersect vertically and horizontally.

Why? Since that would mean two rectangles intersect!

For a specific input (x, y), we define for rectangle R = A × B:

- Horizontally good:
  - Horizontally intersect with at most half of rectangles
  - A contains x
- **Vertically good:**
  - **Vertically intersect** with at most **half** of rectangles
  - **B contains y**



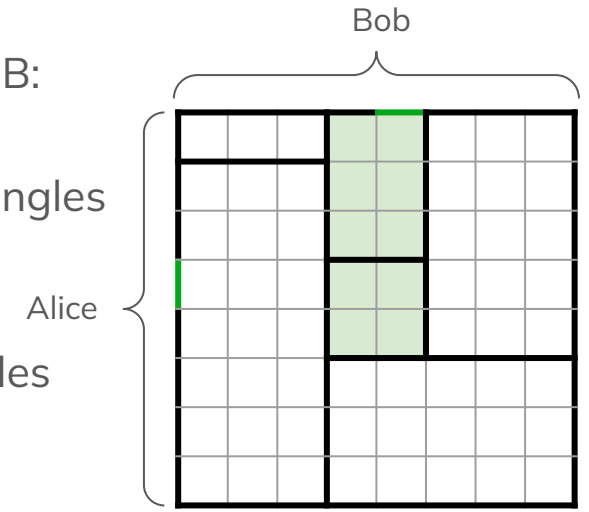
Rectangle Partition to Protocol Tree

Fact: No two distinct rectangles both intersect vertically and horizontally.

Why? Since that would mean two rectangles intersect!

For a specific input (x, y), we define for rectangle R = A × B:

- Horizontally good:
  - Horizontally intersect with at most half of rectangles
  - A contains x
- Vertically good:
  - Vertically intersect with at most half of rectangles
  - B contains y
- **Good: Either horizontally good or vertically good**

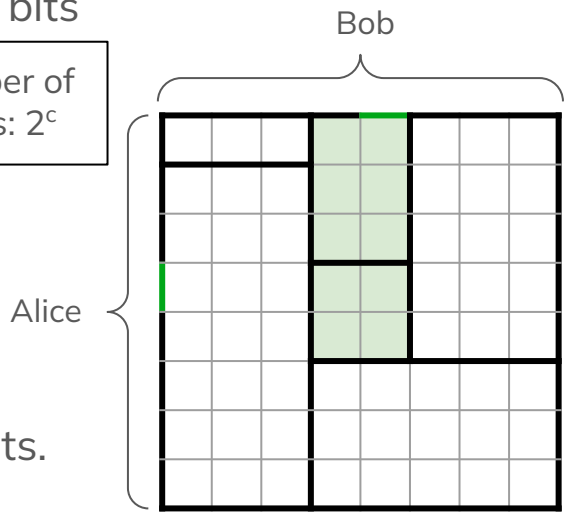


Rectangle Partition to Protocol Tree

What can we do if we find a rectangle that is either **horizontally or vertically good**?

- Suppose a rectangle is **horizontally good**:
  - Alice can send the index of the rectangle in O(c) bits
  - Eliminate every rectangle that does not horizontally intersected
  - **Less than half of rectangles remain**
- Symmetric case for **vertically good**!

Total number of rectangles: 2^c

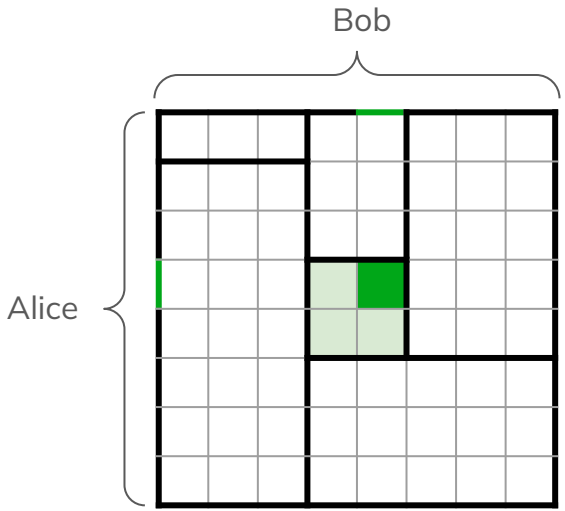


Conclusion: If **we can always find a good rectangle**, then by O(c) rounds, the rectangle can be pinpointed in $O(c^2)$ bits.

Rectangle Partition to Protocol Tree

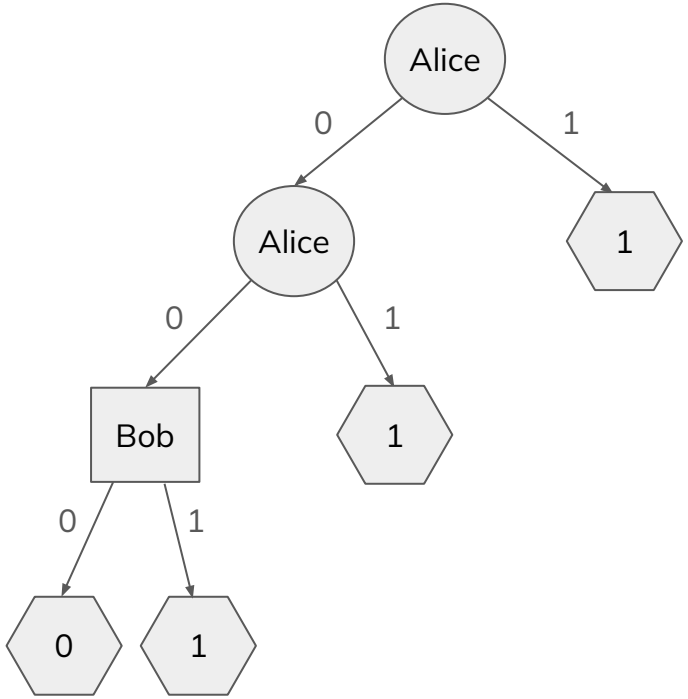
In fact, the rectangle containing (x, y) is always **good**.

Why? Since **no rectangle can intersect with it both vertically and horizontally**, and so by **pigeonhole principle** there must be either direction that intersect with < half of rectangles!



Conclusion: A rectangle partition of 2^c rectangles can be converted to a protocol tree of height $O(c^2)$.

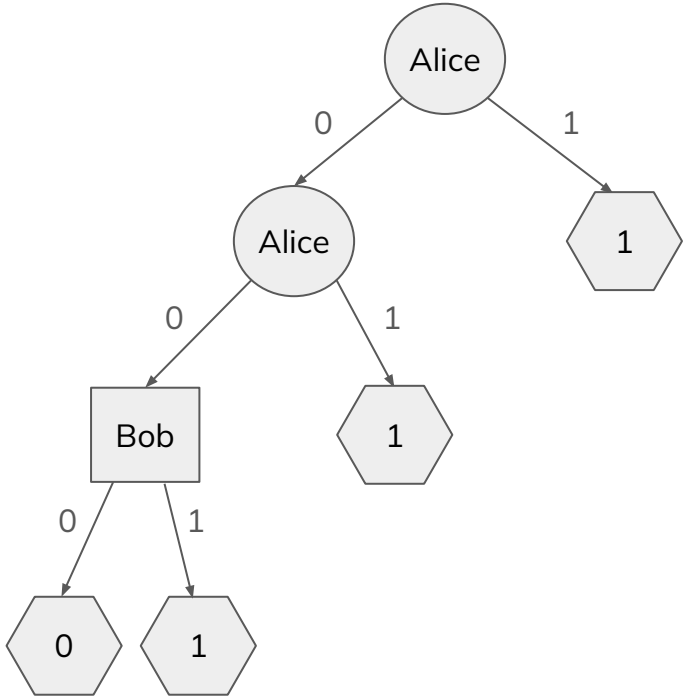
Optimising Protocol Tree



Of course, we want the height of our Protocol Tree to be as short as possible, since that would mean we need less queries.

Think: How can we “change” the structure of a highly non-balanced tree?

Optimising Protocol Tree



Suppose the current Protocol Tree has **N leaves, but with arbitrary structure**.

How well can we limit the tree height to be?

Optimising Protocol Tree

Let's employ some centroid mindset!

Claim. We can always find some subtree which contains **N/3 ~ 2N/3 leaves**.

Proof. Descend on subtree while any children contains > N/3 leaves.

- After descent, the current subtree must have size > N/3.
- Meanwhile, since each child contains ≤ N/3 leaves, in total the current subtree must contain ≤ 2N/3 leaves.

What can we do with the "centroid"?

Optimising Protocol Tree

Note that for each rectangle A × B, we can ask the question “Is (x, y) in the rectangle?” In 2 queries, by Alice sending whether x in A to Bob, then by Bob sending whether y in B to Alice.

Key Idea: Pick a subtree that contains **N/3 ~ 2N/3 leaves**. Then, “pull the node up”, i.e., check if (x, y) in that subtree first in 2 queries. Then, the protocol tree would be splitted up into subtrees containing ≤ 2N/3 leaves.

That means, recursively, we can ensure the height is at most $\mathbf{2 \log_{3/2} N}$!

Applications

Number of distinct elements

Suppose there are N integers which you read them one-by-one.

We want to find the **minimum space complexity** to count the number of distinct elements. Obvious enough, the complexity seems to be Ω(N).

But how to prove?

Number of distinct elements

Suppose there are N integers which you read them one-by-one.

We want to find the **minimum space complexity** to count the number of distinct elements. Obvious enough, the complexity seems to be Ω(N).

But how to prove?

Formulate it into communication complexity task!

Number of distinct elements

Let's say we have an algorithm that can find the number of distinct elements in o(N) space complexity.

Then we can do the following:

- Alice run the algorithm for first N / 2 elements.
- Alice sends the memory in her program to Bob. The number of bits is o(N).
- Bob continues to run the algorithm for the remaining elements.

Think: What is the issue for the above to happen?

Number of distinct elements

Let's say we have an algorithm that can find the number of distinct elements in o(N) space complexity.

Then we can do the following:

- Alice run the algorithm for first N / 2 elements.
- Alice sends the memory in her program to Bob. The number of bits is o(N).
- Bob continues to run the algorithm for the remaining elements.

This solves the `DISJOINTNESS` task for one-way communication complexity in o(N) - but we have proven that it is at least N!

Range Minimum Query

Actually, the time complexity of many other things can be proven by communication complexity.

Range Minimum Query: Given an array of N integers A[1], A[2], ..., A[N].
Support two operations:

- Update: Change A[i] to X
- Query: Find the minimum value among A[L] ... A[R].

It can be proven to take at least O(log N) time per operation.

Dynamic Connectivity

Dynamic Connectivity: Given N nodes which initially disconnected.

Support three operations:

- Link: add edge between node u and node v
- Cut: remove edge between node u and node v
- Query: determine if node u and node v are connected.

It can be proven to take at least O(log N) time per operation.

If you are interested, you can research more information about them as the proof is very hard and out of today's scope.

Thank you!