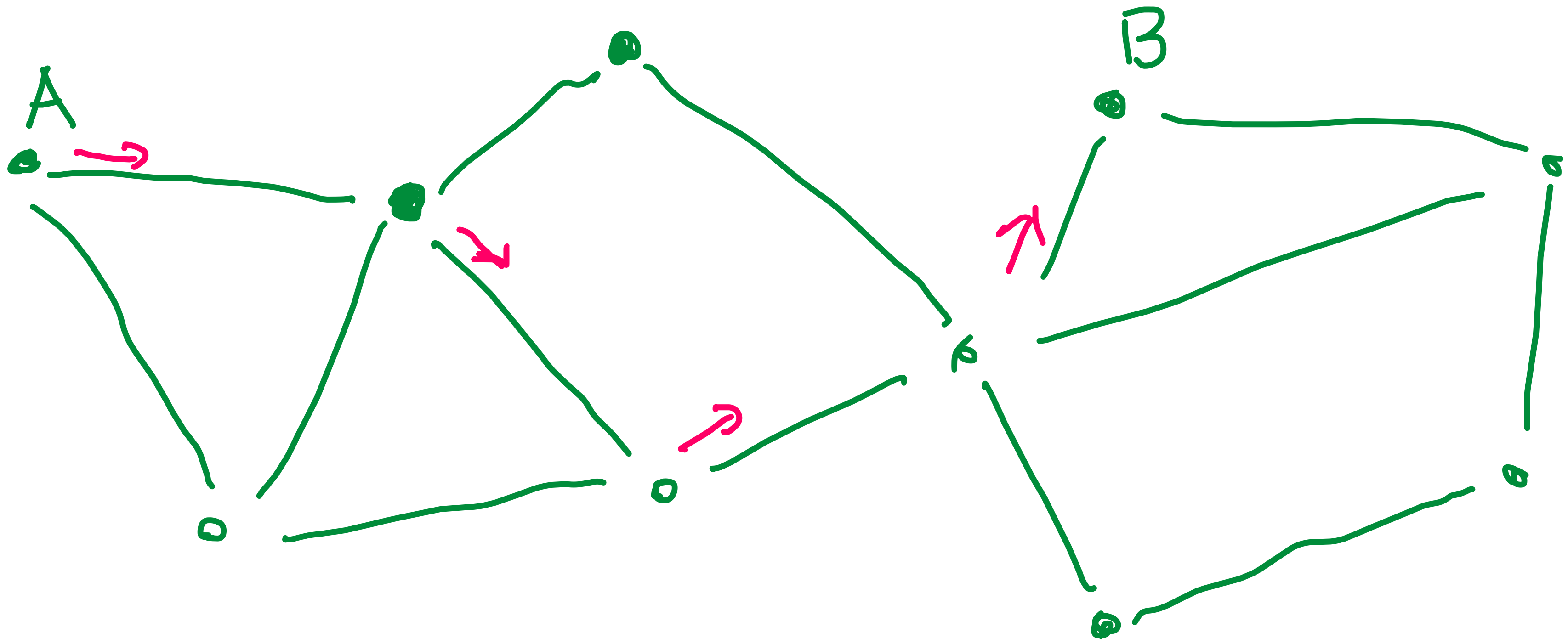
Distributed Computing
PWr, WIT 2021

Prof. Mirosław Kutyłowski

3: Communication complexity

All pairs shortest path problem (APSP)

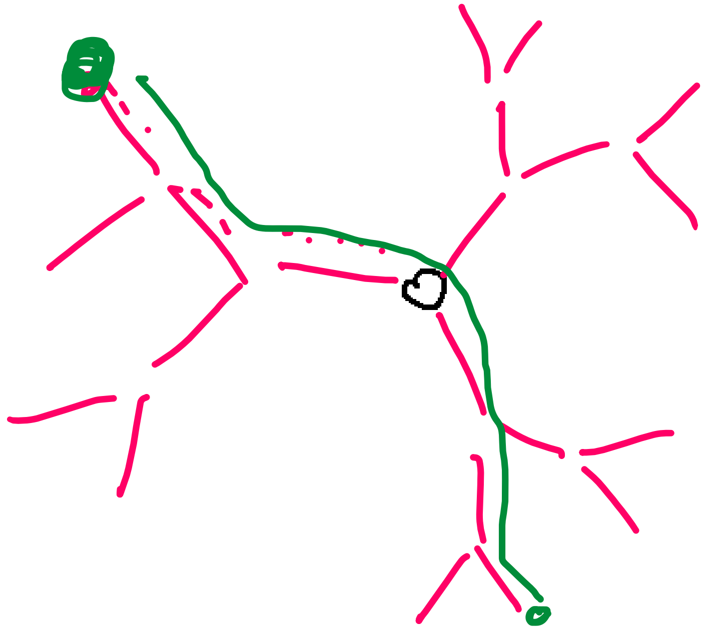
- For every pair of nodes (u,v) compute the shortest path from u to v
- Store the results so that routing along these paths is possible



Find diameter of the graph - naïve solution

Algorithm 11.1 Naive Diameter Construction

1: all nodes compute their radius by synchronous flooding/echo
2: all nodes flood their radius on the constructed BFS tree
3: the maximum radius a node sees is the diameter



Naïve solution complexity

time O(D) for diameter D

Congestion of messages – n algorithms executed in parallel

Reasonable size of messages

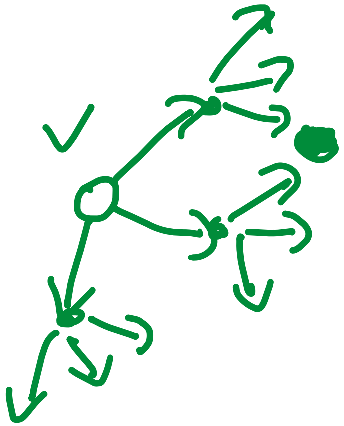
LORA

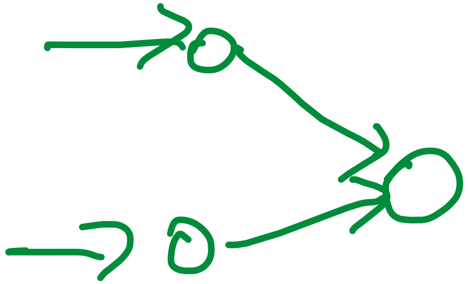
something like O(log n)

Building block -- BFS

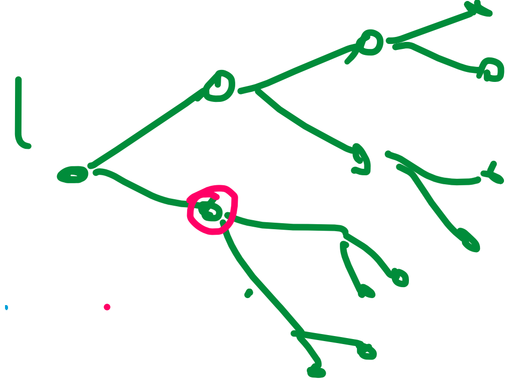
Broad First Search

Definition 11.2. *(BFS_v) Performing a breadth first search at node v produces spanning tree BFS_v (see Chapter 2). This takes time $\mathcal{O}(D)$ using small messages.*





Pebbles algorithm



Algorithm 11.3 Computes APSP on G.

Assume we have a leader node l (if not, compute one first)
compute BFS_l of leader l
send a pebble P to traverse BFS_l in a DFS way;
while P traverses BFS_l **do**
    if P visits a new node v **then**
        wait one time slot; // avoid congestion
        start BFS_v from node v; // compute all distances to v
        // the depth of node u in BFS_v is $d(u, v)$
    end if
end while

Algorithm 11.3 Computes APSP on G.

Assume we have a leader node l (if not, compute one first)
compute BFS_l of leader l
send a pebble P to traverse BFS_l in a DFS way;
while P traverses BFS_l **do**
    if P visits a new node v **then**
        wait one time slot; // avoid congestion
        start BFS_v from node v; // compute all distances to v
        // the depth of node u in BFS_v is $d(u, v)$
    end if
end while

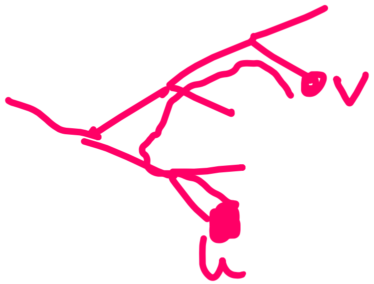
Algorithm 11.3 Computes APSP on G.

```
Assume we have a leader node l (if not, compute one first)
compute BFS_l of leader l
send a pebble P to traverse BFS_l in a DFS way;
while P traverses BFS_l do
    if P visits a new node v then
        wait one time slot;   // avoid congestion
        start BFS_v from node v;   // compute all distances to v
        // the depth of node u in BFS_v is d(u, v)
    end if
end while
```

Algorithm 11.3 Computes APSP on G.

```
Assume we have a leader node l (if not, compute one first)
compute BFS_l of leader l
send a pebble P to traverse BFS_l in a DFS way;
while P traverses BFS_l do
    if P visits a new node v then
        wait one time slot;   // avoid congestion
        start BFS_v from node v;   // compute all distances to v
        // the depth of node u in BFS_v is d(u, v)
    end if
end while
```

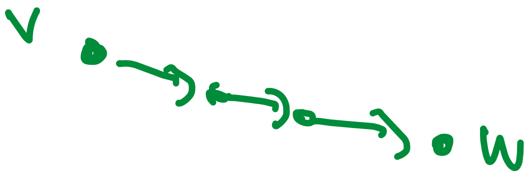
Avoiding congestions



Lemma: no node is simultaneously involved in BFS_u and BFS_v

Proof:

- Let: BFS started at u at time t_u, BFS started at v at time t_v
- A node w involved at time $t_u+d(u,w)$, so $t_v >= t_u + d(u,v)+1$
- $t_v + d(v,w) >= (t_u + d(u,v)+1) + d(v,w) >= t_u + d(u,w) + 1 > t_u + d(u,w)$



Time complexity

Theorem 11.5. *Algorithm 11.3 computes APSP (all pairs shortest path) in time* $\mathcal{O}(n)$.

Lower bound

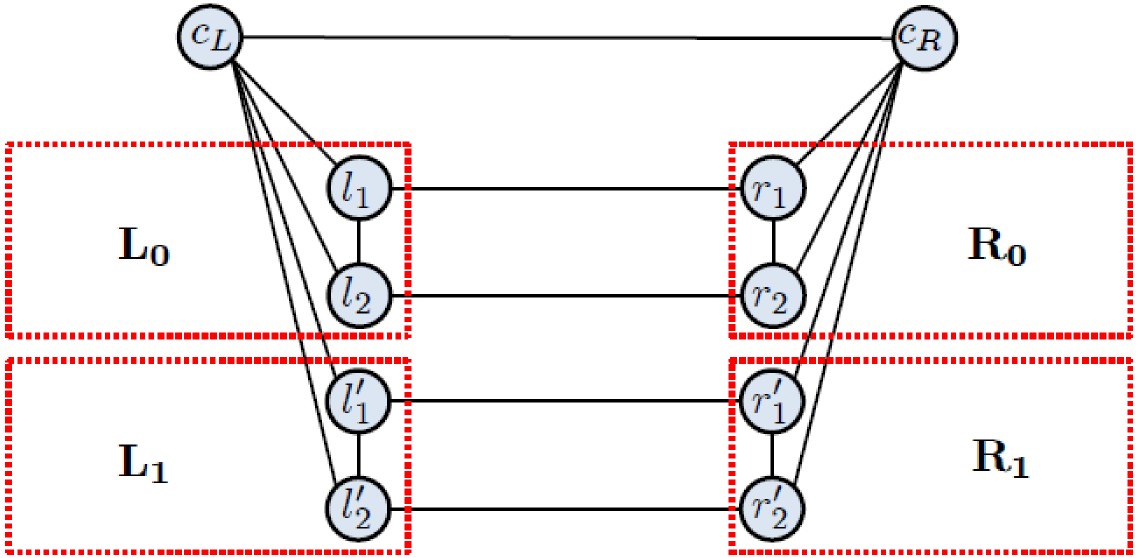
Argument showing that complexity of any algorithm is at least b, where b is the bound

Known for sequential programs (e.g. the number for steps for sorting)

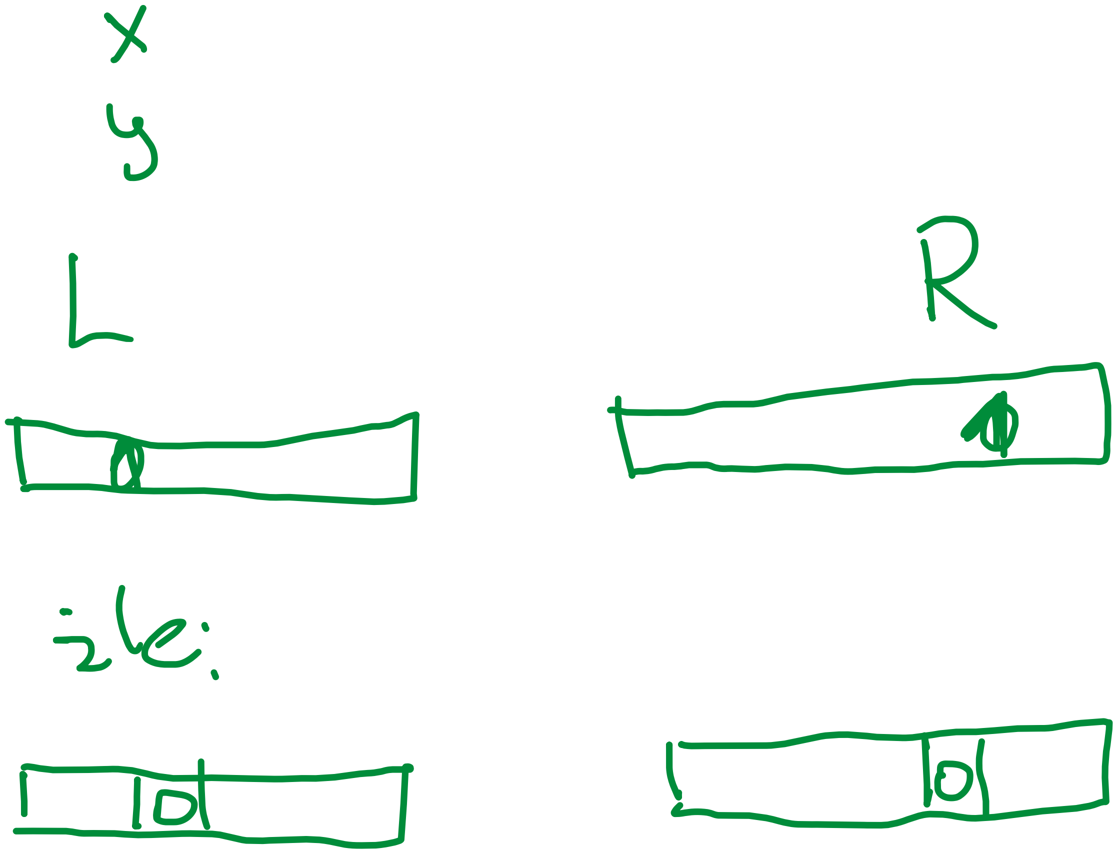
Considerably more complex to prove for distributed algorithms

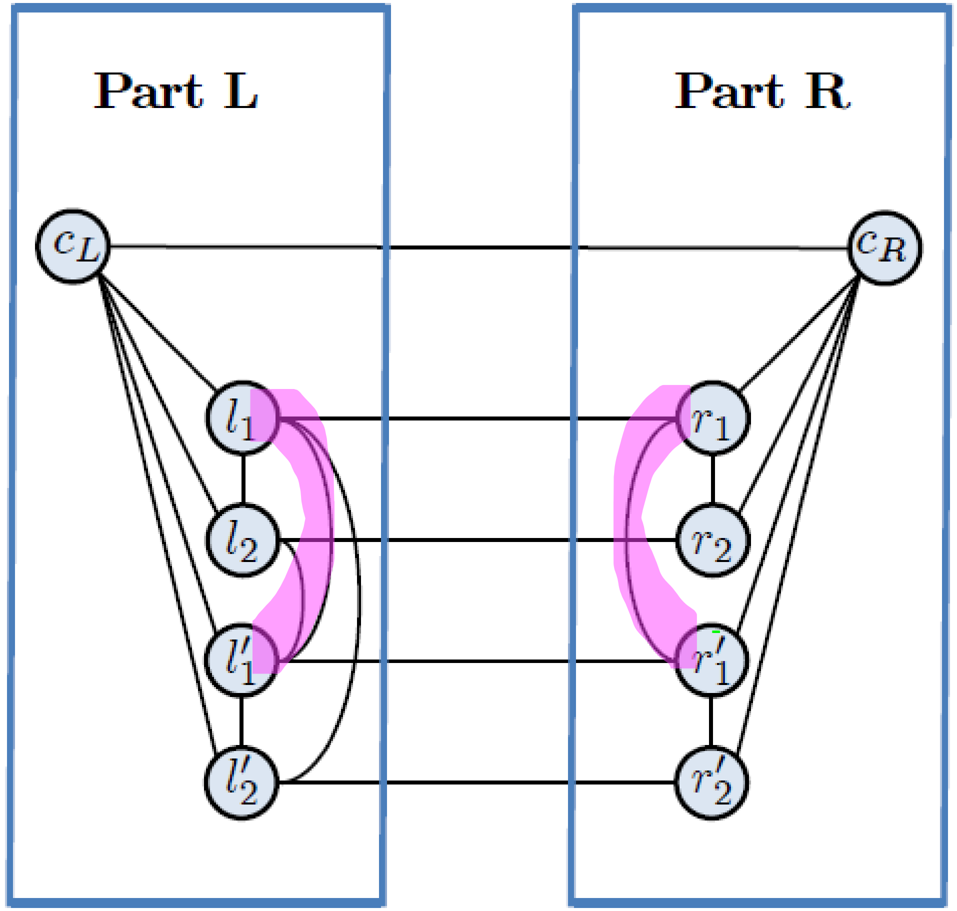
Graph used for showing a n/log n lower bound

$$
\begin{aligned}
\mathbf{L_0} &:= \{l_i \mid i \in [q]\,\} && \text{// upper left in Figure 11.6}\\
\mathbf{L_1} &:= \{l'_i \mid i \in [q]\,\} && \text{// lower left}\\
\mathbf{R_0} &:= \{r_i \mid i \in [q]\,\} && \text{// upper right}\\
\mathbf{R_1} &:= \{r'_i \mid i \in [q]\,\} && \text{// lower right}
\end{aligned}
$$

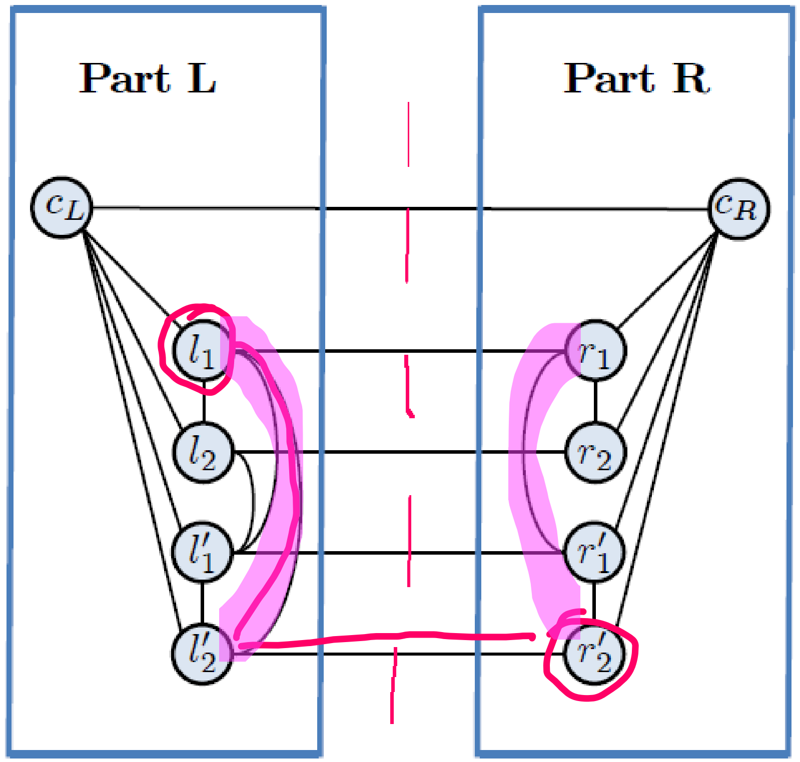


Some number of edges between L_0 and L_1,
some number of edges between R_0 and R_1

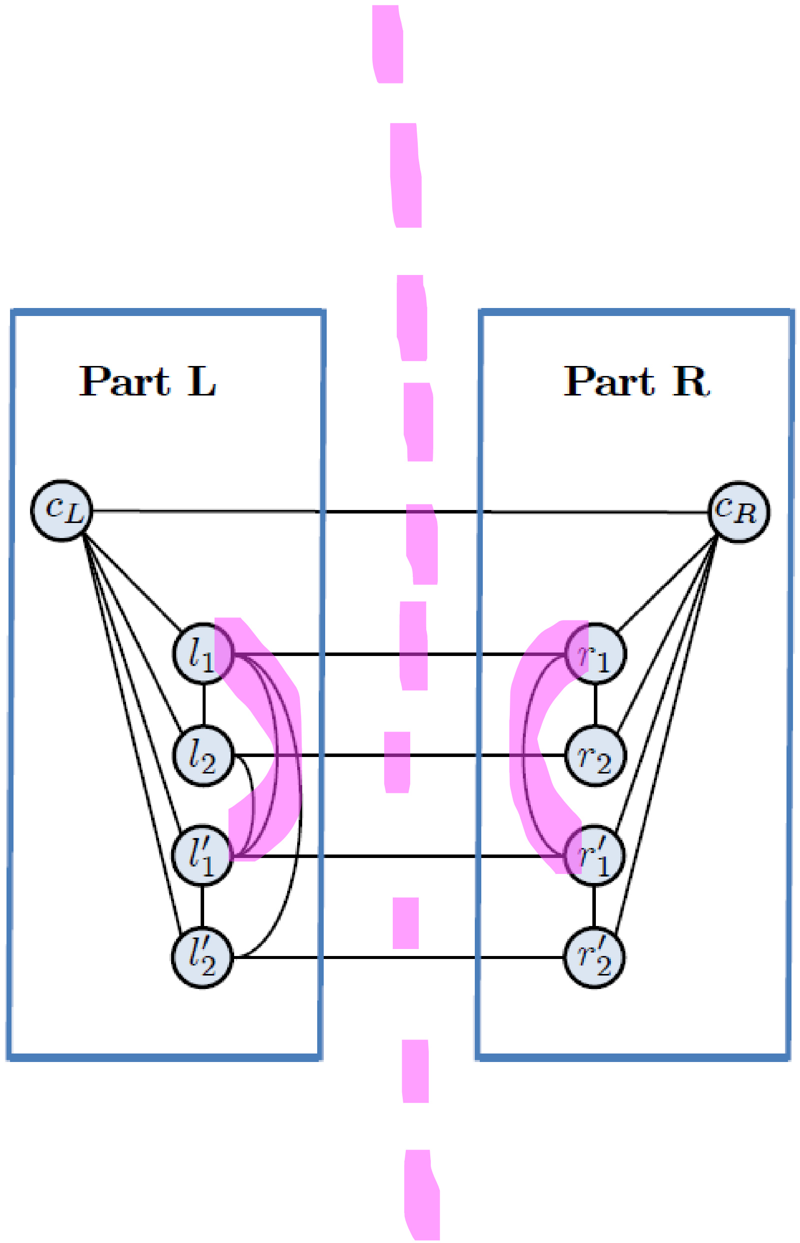




Diameter 2 or 3



Cutsize



Informal argument

One has to check that for each (i,j) there is a connection either on the left or on the right side

2-party communication model

- Alice gets x, Bob gets y
- The goal is to compute $f(x,y)$
- Alice and Bob exchange messages, finally both Alice and Bob must learn $f(x,y)$

Communication complexity

The simplest solution:

1. Alice sends x to Bob
2. Bob computes f(x,y)
3. Bob sends f(x,y) to Alice

Communication complexity

length(x) + length(f(x,y))

$\leq n$

Equality, its complexity?

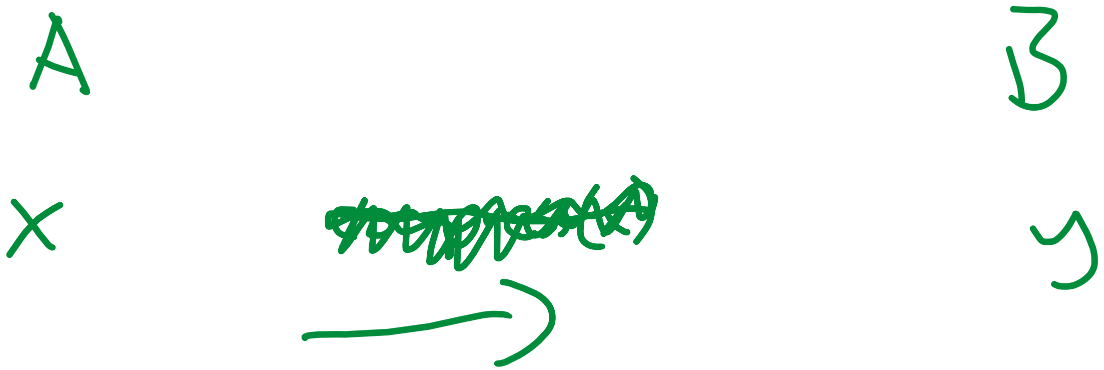
(Equality.) We define the equality function EQ *to be:*

$$\mathrm{EQ}(x, y) := \begin{cases} 1 & : x = y \\ 0 & : x \neq y \end{cases}.$$

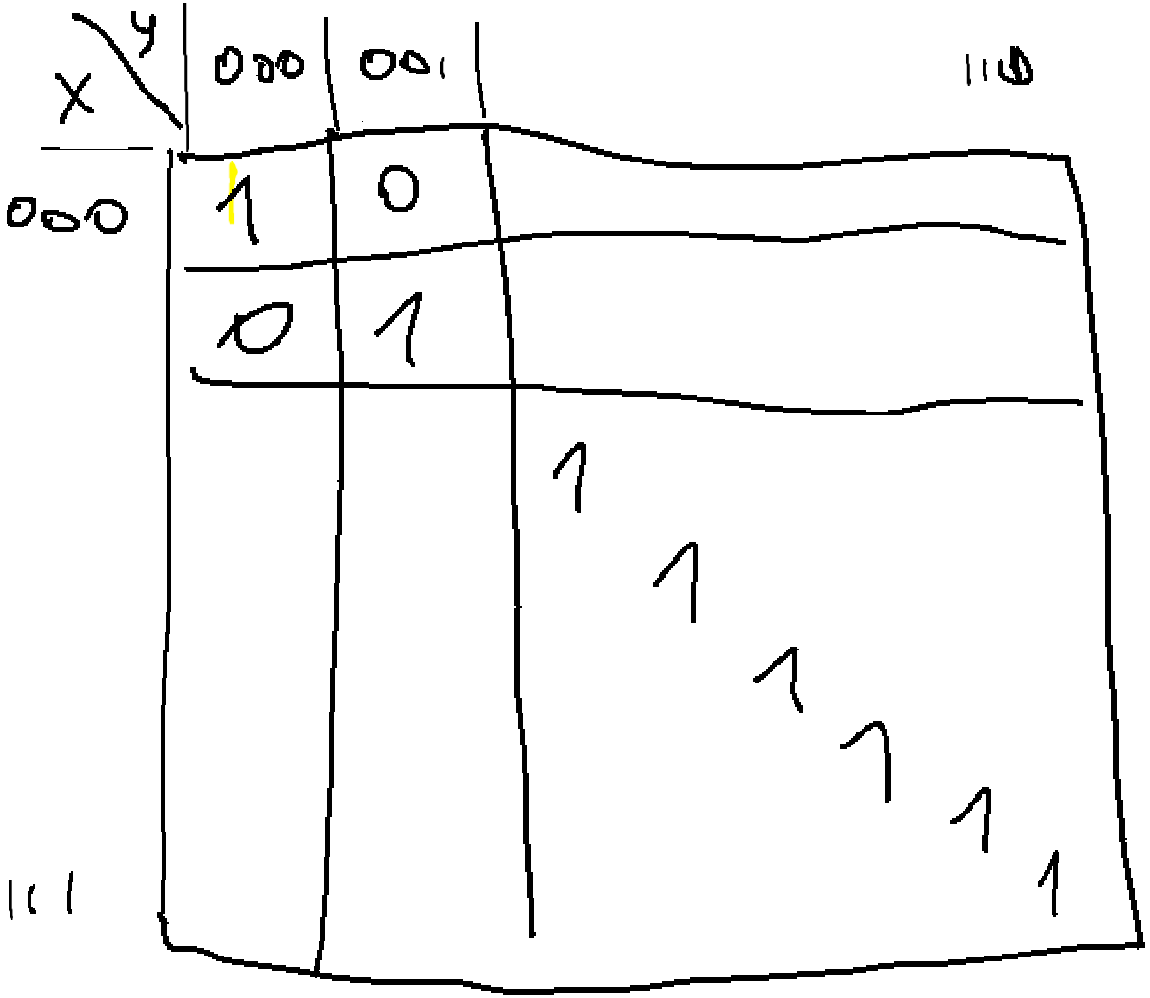
Formal definition of communication complexity

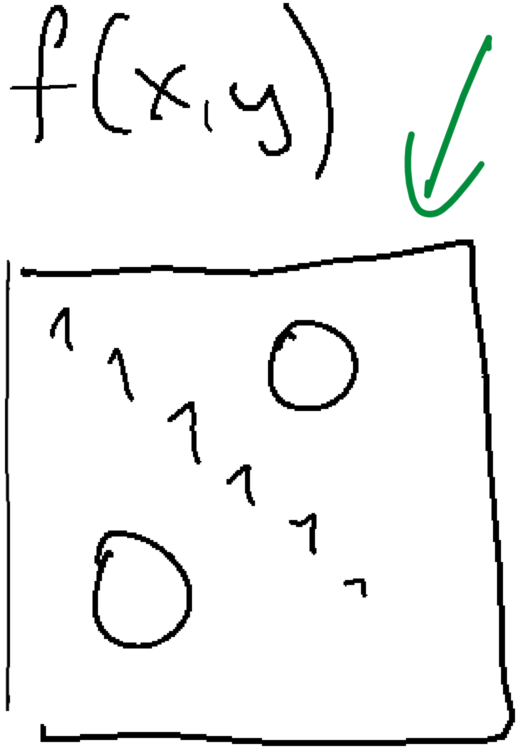
The total size of all messages exchanged …

… in the worst case.



Matrix representation of function f

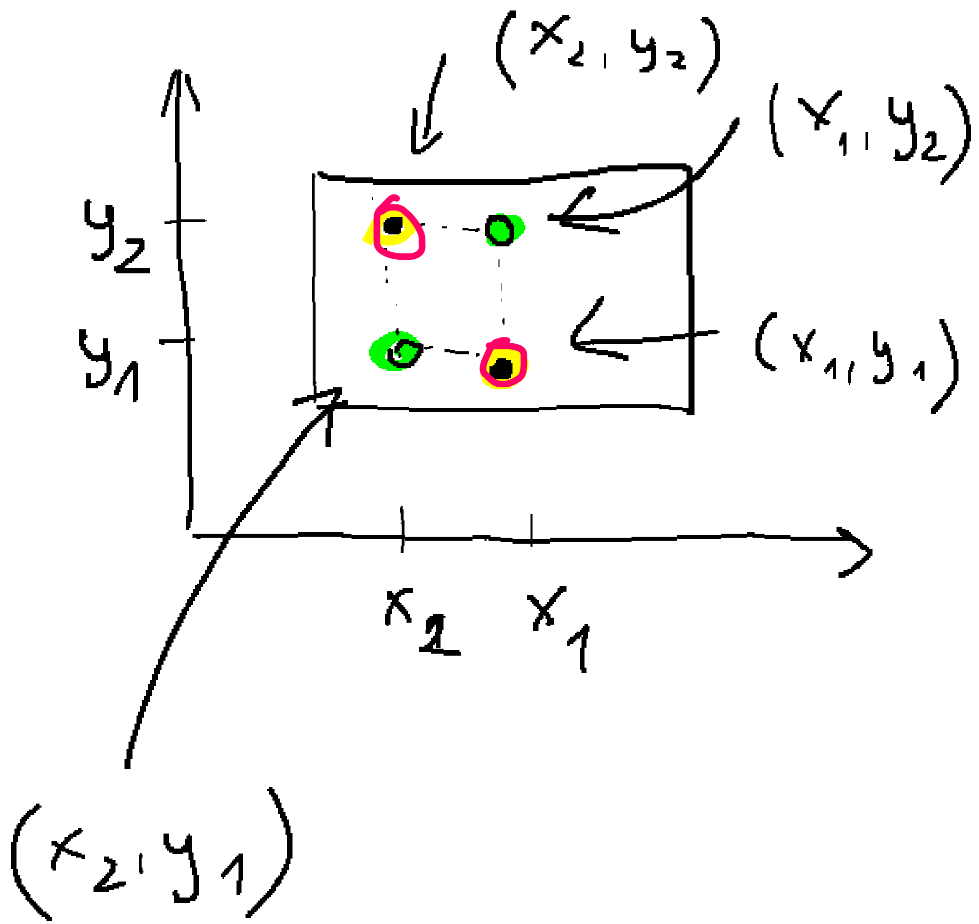


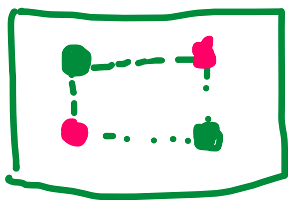


“rectangles”

A set S of pairs is a rectangle iff

If (x_0,y_0) and (x_1,y_1) belong to S, then (x_0,y_1) and (x_1,y_0) belong to S as well





Rectangles

EQ	000	001	010	011	100	101	110	111	← x
000	1	0	0	0	0	0	0	0	
001	0	1	0	0	0	0	0	0	
010	0	0	1	0	0	0	0	0	
011	0	0	0	1	0	0	0	0	
100	0	0	0	0	1	0	0	0	
101	0	0	0	0	0	1	0	0	
110	0	0	0	0	0	0	1	0	
111	0	0	0	0	0	0	0	1	
↑ y									

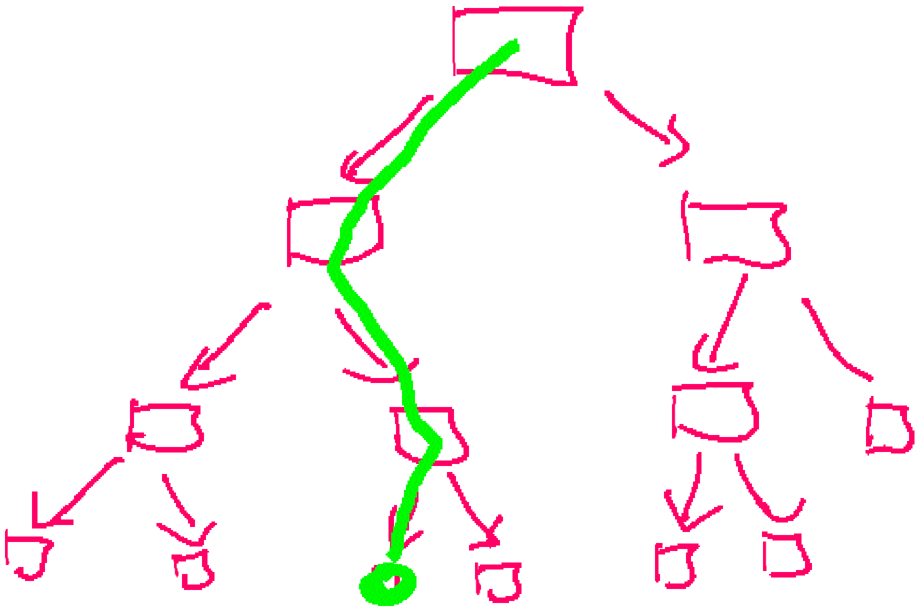
Importance of rectangles

For a given set of messages exchanged, the set of possible inputs (x,y) is a rectangle

“monochromatic” rectangle

A rectangle where the value f(x,y) is fixed

One cannot stop the computation unless a rectangle is monochromatic



Monochromatic rectangles

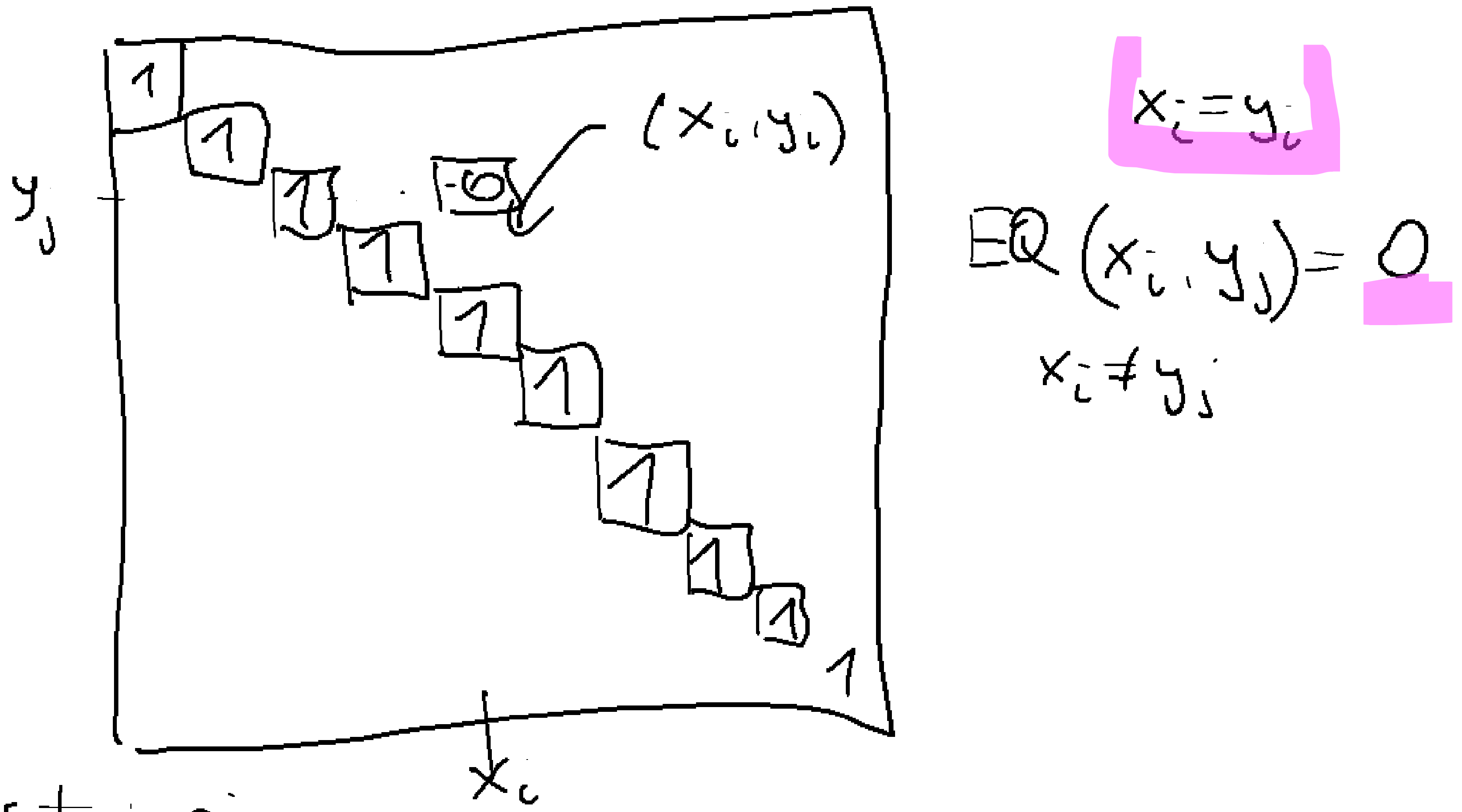
EQ	000	001	010	011	100	101	110	111	$\leftarrow x$
000	1	0	0	0	0	0	0	0	
001	0	1	0	0	0	0	0	0	
010	0	0	1	0	0	0	0	0	
011	0	0	0	1	0	0	0	0	
100	0	0	0	0	1	0	0	0	
101	0	0	0	0	0	1	0	0	
110	0	0	0	0	0	0	1	0	
111	0	0	0	0	0	0	0	1	
$\uparrow y$									

Fooling set

(x_1,y_1), (x_2,y_2), …, (x_n,y_n) is a fooling set iff

- $f(x_1,y_1)=f(x_2,y_2)= \ldots = f(x_n,y_n)=x$
- for any $i \neq j$ we have $f(x_i,y_j) \neq x$

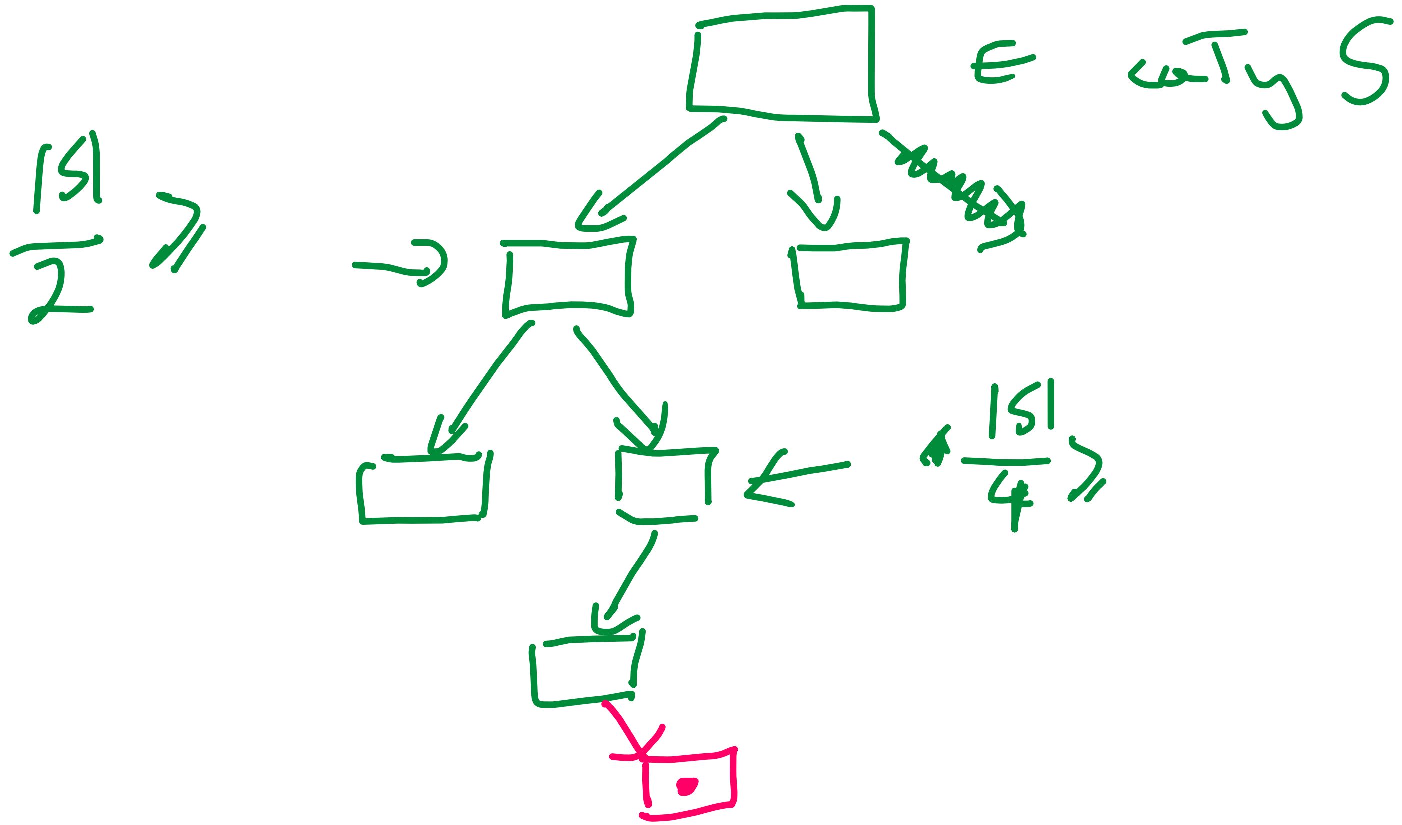
Fooling set for Equality

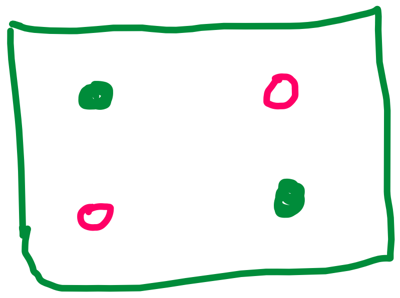


fooling set: size n

Fooling set lemma

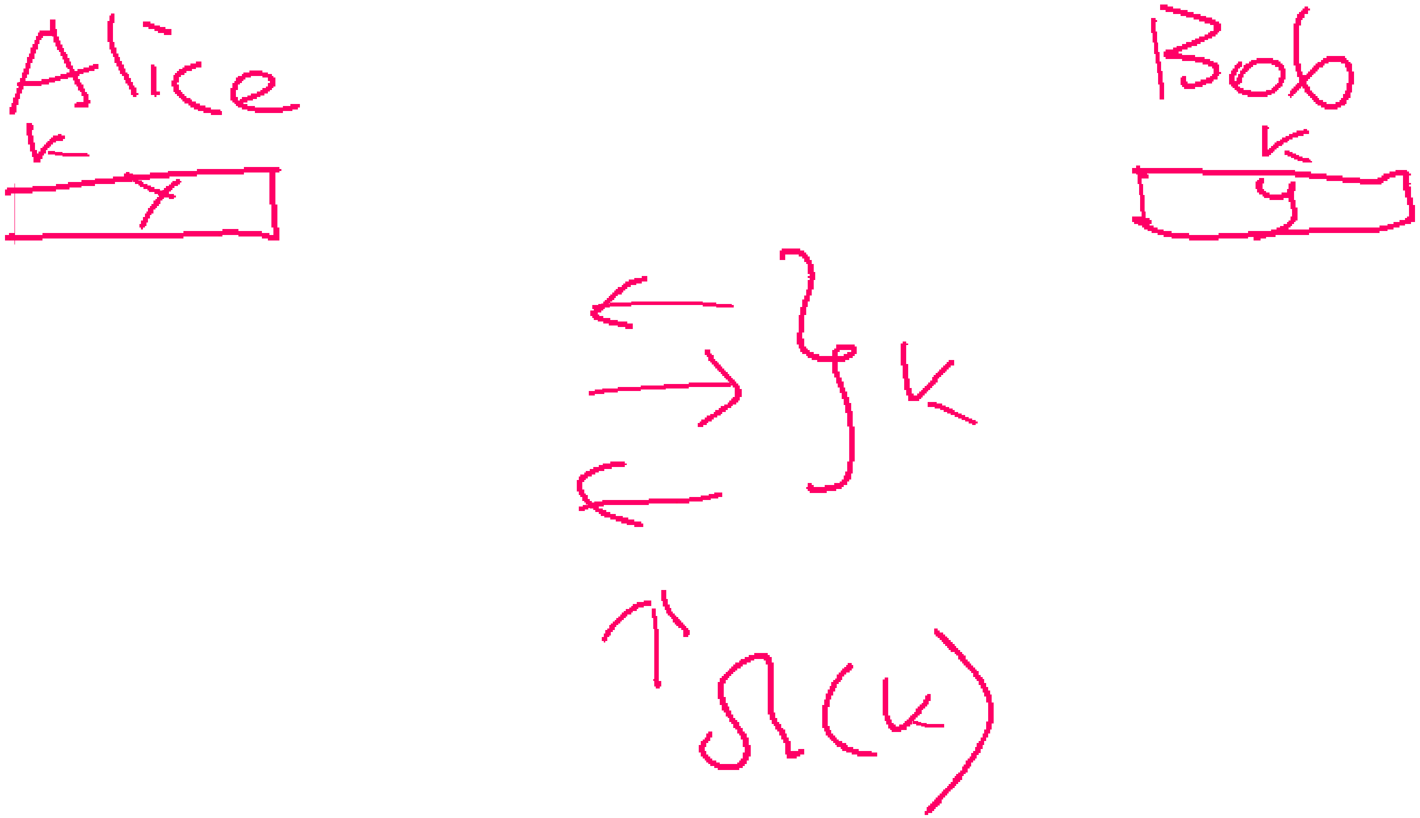
If S is a fooling set for f, then $CC(f) = \Omega(\log |S|)$.





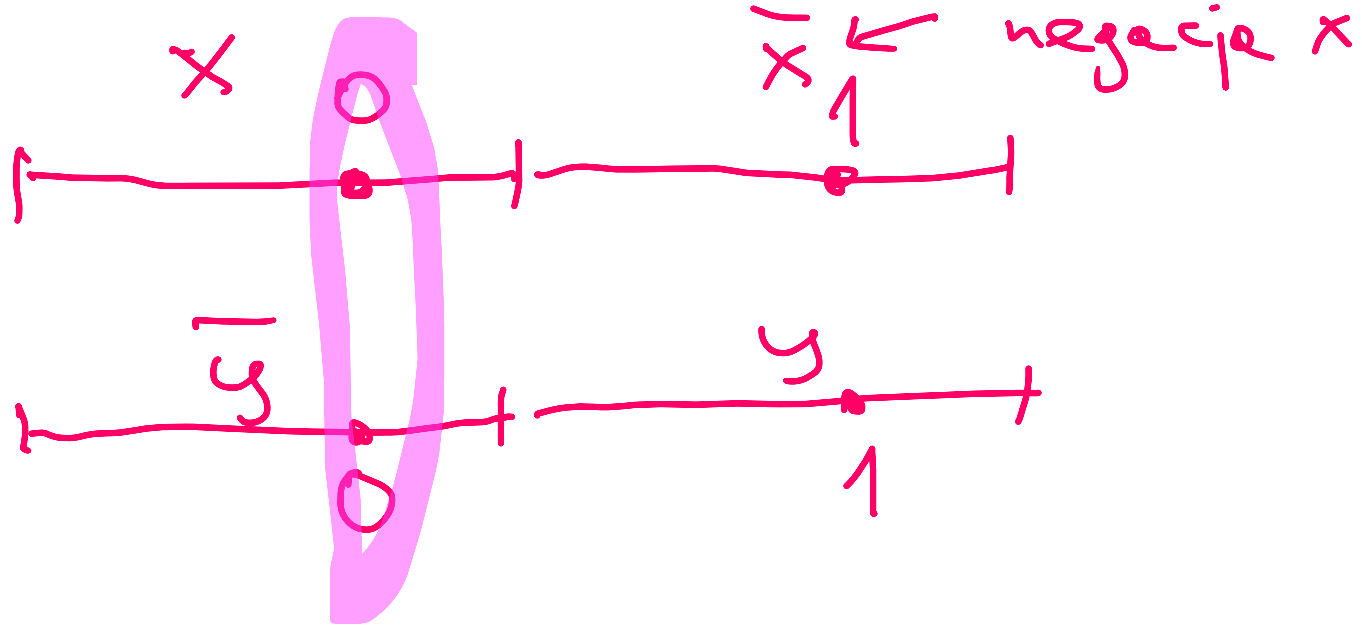
CC of EQ for k-bit numbers is

$$\Omega(\log(2^k)) = \Omega(k)$$



Auxiliary fact

Lemma 11.21. *Let x, y be k-bit strings. Then $x \neq y$ if and only if there is an index $i \in [2k]$ such that the i^{th} bit of $x \circ \overline{x}$ and the i^{th} bit of $\overline{y} \circ y$ are both 0.*



Mapping to graph

Definition 11.22. *Using the parameter q defined before, we define a bijective map between all pairs x, y of q^2-bit strings and the graphs in $\mathcal{G}$: each pair of strings x, y is mapped to graph $G_{x,y} \in \mathcal{G}$ that is derived from skeleton G' by adding*

- *edge (l_i, l'_j) to* ***Part L*** *if and only if the $(j + q \cdot (i-1))^{th}$ bit of x is 1.*
- *edge (r_i, r'_j) to* ***Part R*** *if and only if the $(j + q \cdot (i-1))^{th}$ bit of y is 1.*

Mapping to graph

Lemma 11.23. *Let x and y be $\frac{q^2}{2}$-bit strings given to Alice and Bob.*[1] *Then graph $G := G_{x \circ \overline{x}, \overline{y} \circ y} \in \mathcal{G}$ has diameter 2 if and only if $x = y$.*

Lower bound

It follows that computing APSP for a graph
requires exchanging $\Omega(n)$ bits between the left and the right part,
i.e. $\Omega(n/\log(n))$ messages of size $\log(n)$

Randomized complexity of equality

Algorithm 11.25 Randomized evaluation of EQ.

Alice and Bob use public randomness. That is they both have access to the same random bit string $z \in \{0,1\}^k$

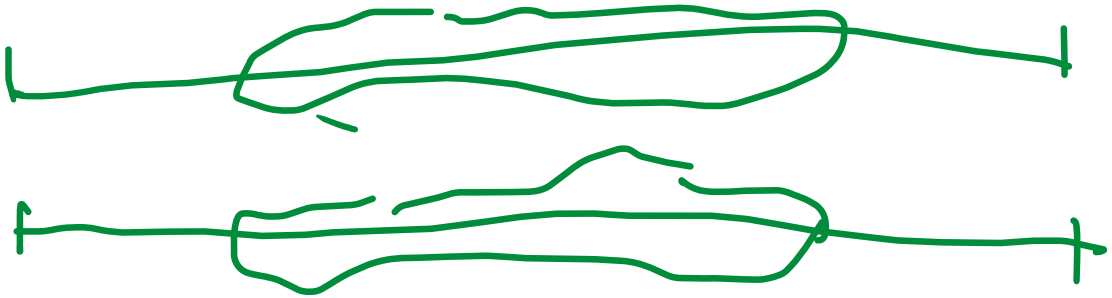
Alice sends bit $a := \sum_{i \in [k]} x_i \cdot z_i \mod 2$ to Bob

Bob sends bit $b := \sum_{i \in [k]} y_i \cdot z_i \mod 2$ to Alice

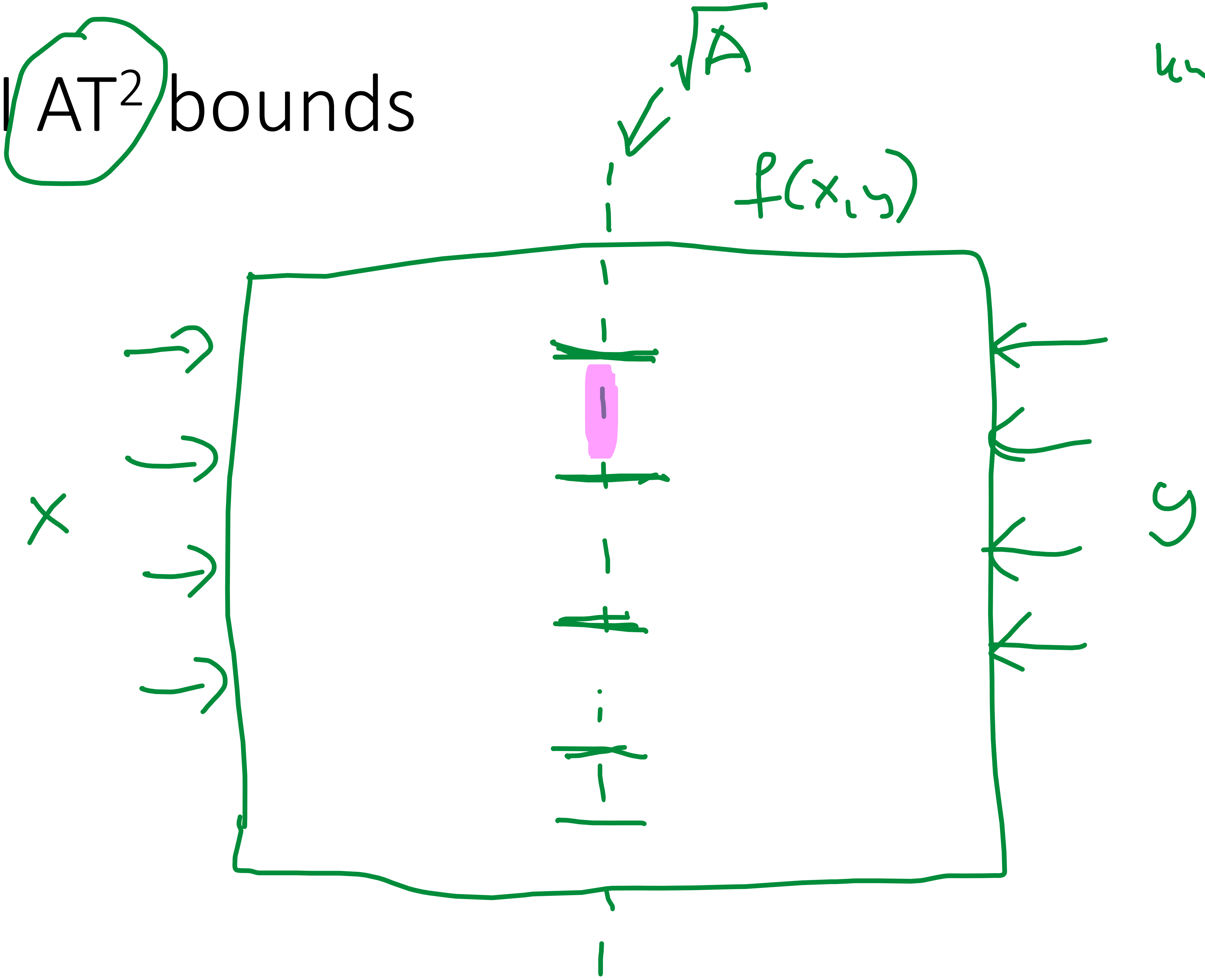
if $a \neq b$ **then**

  we know $x \neq y$

end if



VLSI AT^2 bounds



kwadrat o polu A

$\sqrt{A} \cdot T \geq n$

$AT^2 \geq n$