Seminar on distributed algorithms – PN models chapter 3

Maayan Ifergan & Tom Sasson

Overview

01
Reminder - distributed networks in general.

02
The PN model.

03
Distributed algorithms in the PN model.

04
Algorithm for 3 - coloring path graphs

05
Algorithm for maximal matching in bipartite graphs.

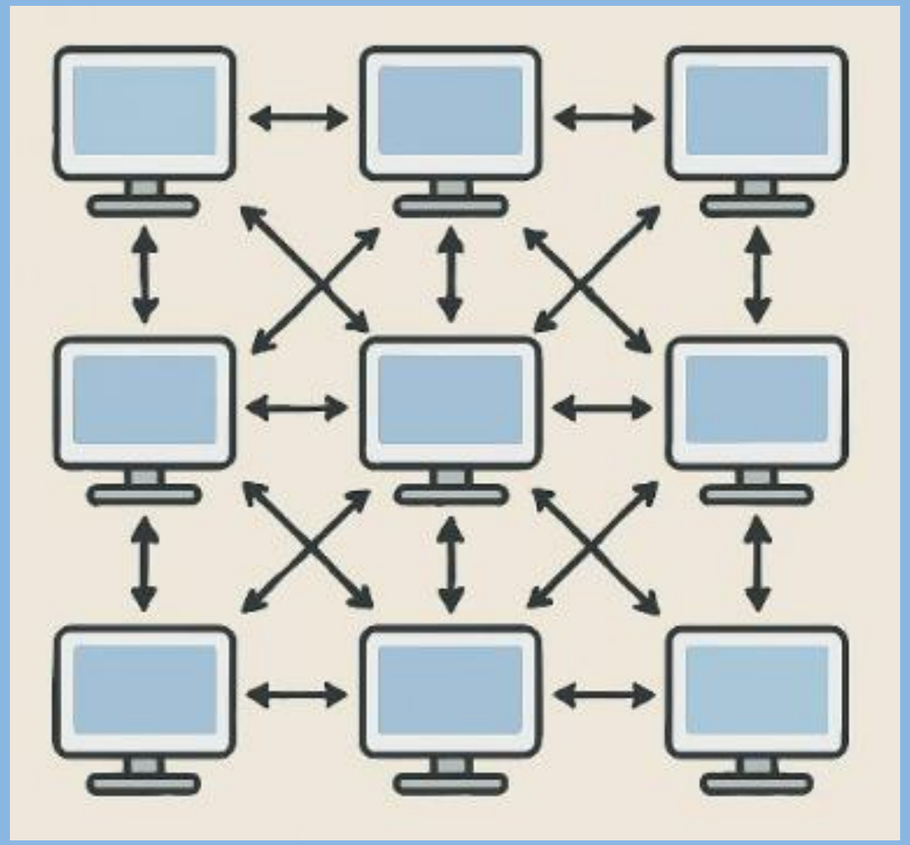
Reminder

Reminder -

distributed Networks

- A distributed system is a network where each node is a computer, and each edge is a communication link.
- Each node can exchange messages only with its neighbors. Each communication round takes fixed time.

Example



Reminder – distributed algorithms

- A distributed algorithm is a method for solving computational problems in a network.
- Distributed algorithms work with local knowledge and rely on message passing between nodes.
- We would like to minimize the number of communication rounds.

The PN model

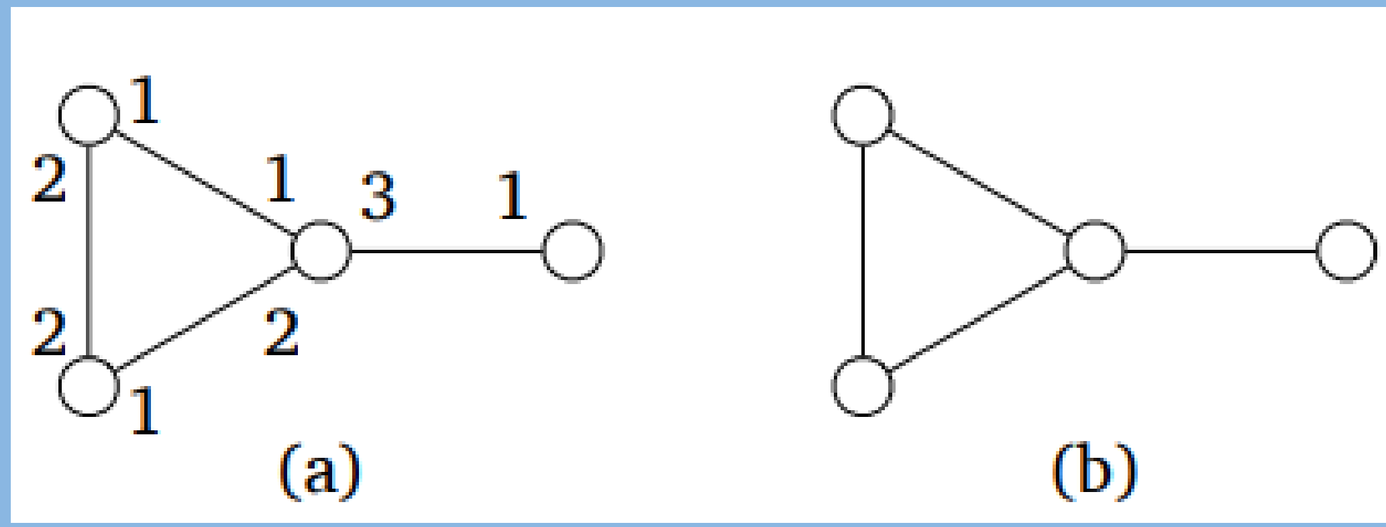
PN model definition

- A port-numbered network is a triple $N = (V, P, p)$, where V is the set of nodes, P is the set of ports, and $p: P \to P$ is a function that specifies the connections between the ports. We make the following assumptions:
- (a) Each port is a pair (v, i) where $v \in V$ and $i \in \{1, 2, \ldots \mathrm{Deg}(v)\}$.
- (b) The connection function p is an involution, that is, for any port $(u, i) \in P$ we have $p(p((u, i))) = (u, i)$.

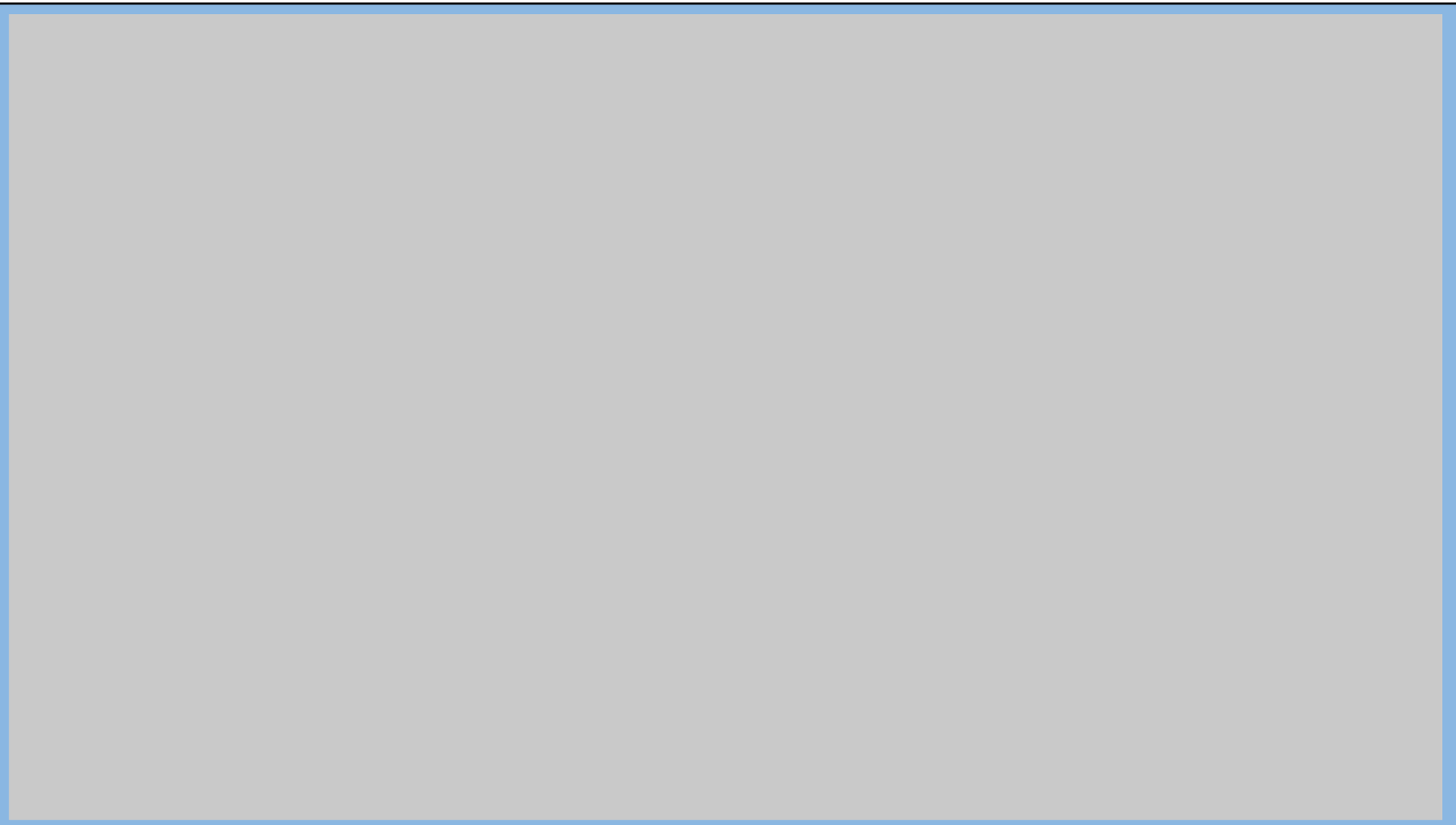
Underlying graph

- For a simple port-numbered network $N = (V, P, p)$ we define the underlying graph $G = (V, E)$ as follows:
- $\{u, v\} \in E$ if and only if u is connected to v in network N.
- Observe that $\deg_G(v) = \deg_N(v)$ for all $v \in V$.

Underlying graph

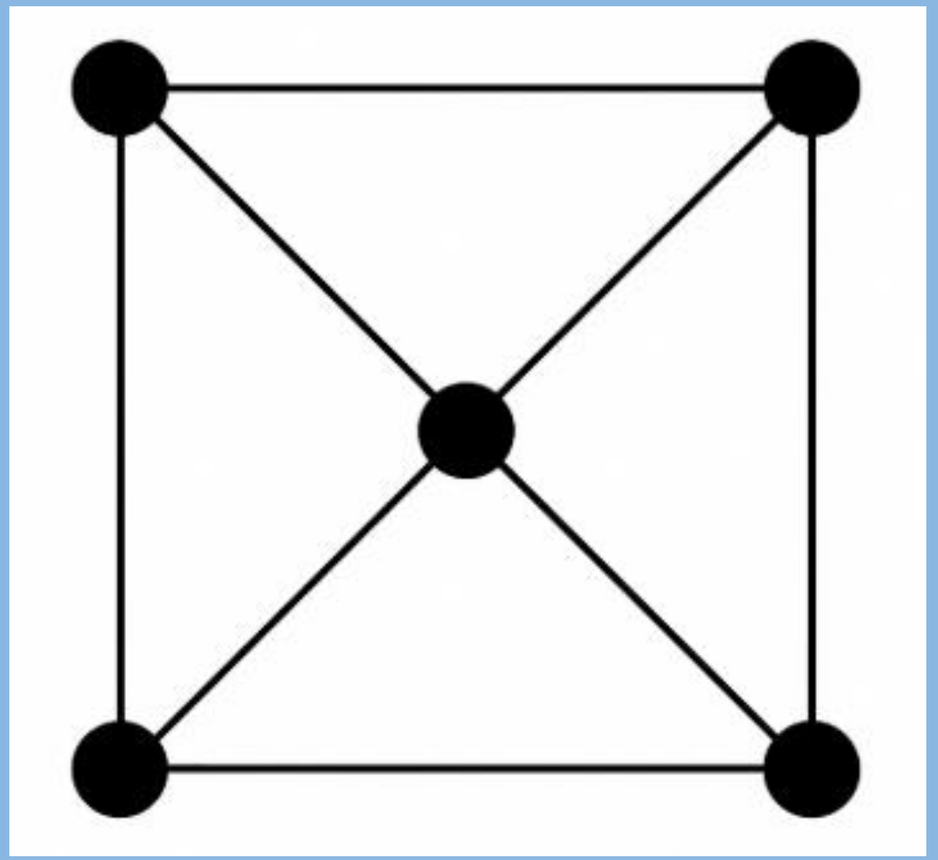


video



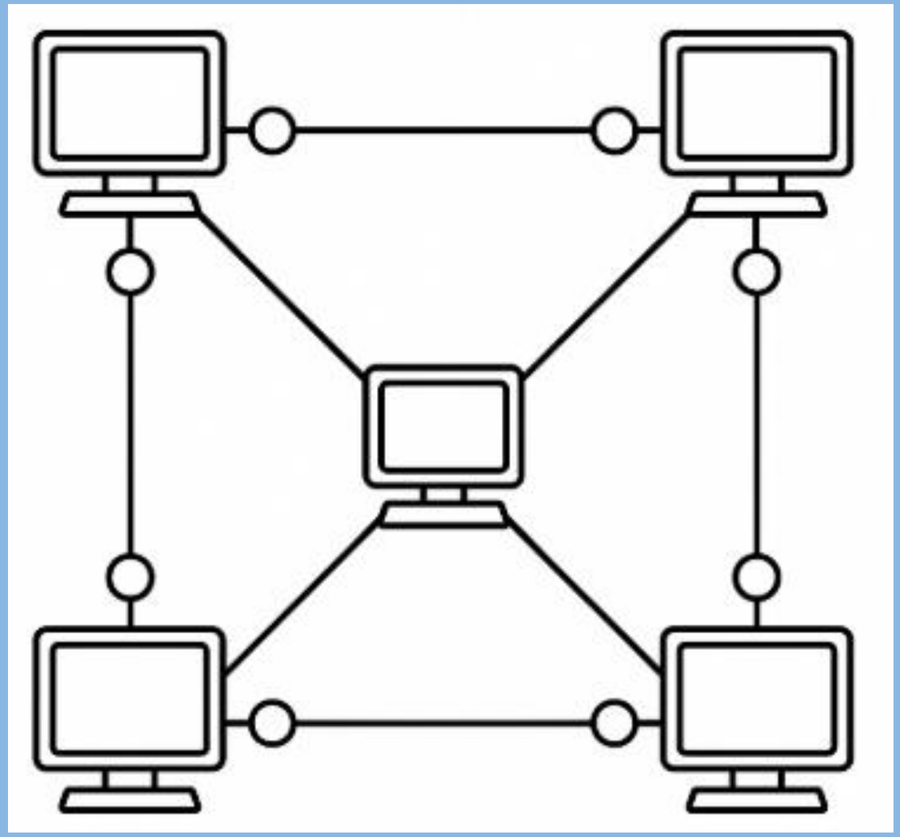
Running example

A) Start from a simple undirected graph G = (V, E).



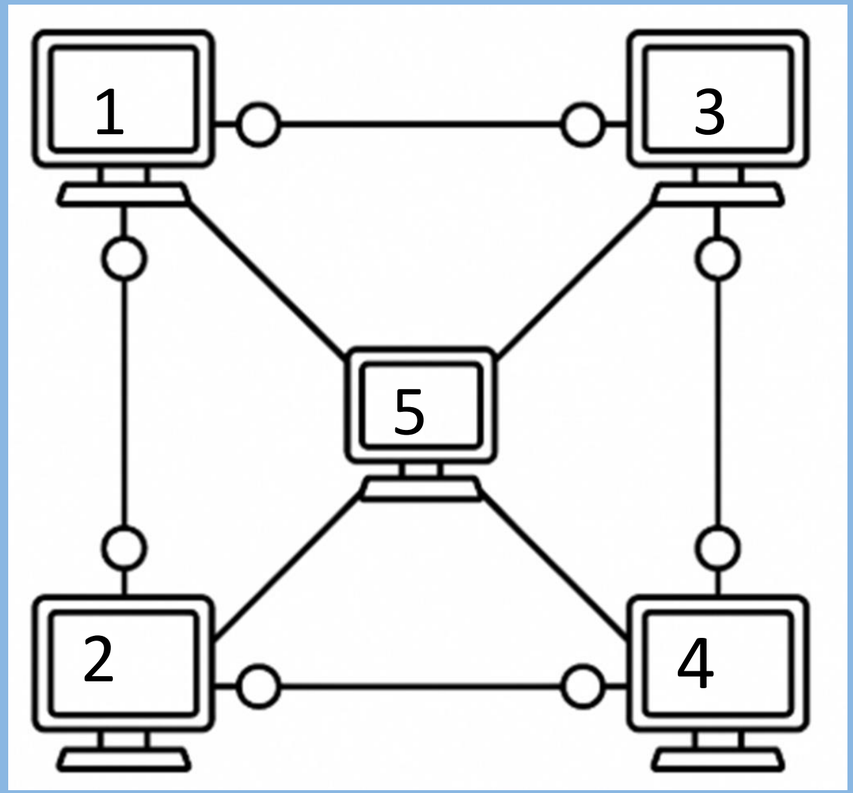
B) Put together a computer network N with the same structure as G. A node $v \in V$ corresponds to a computer in N, and an edge $\{u, v\} \in E$ corresponds to a communication link between the computers u and v.

Use a function for node labeling $f : V \rightarrow Y$, where Y are in this case, natural numbers.

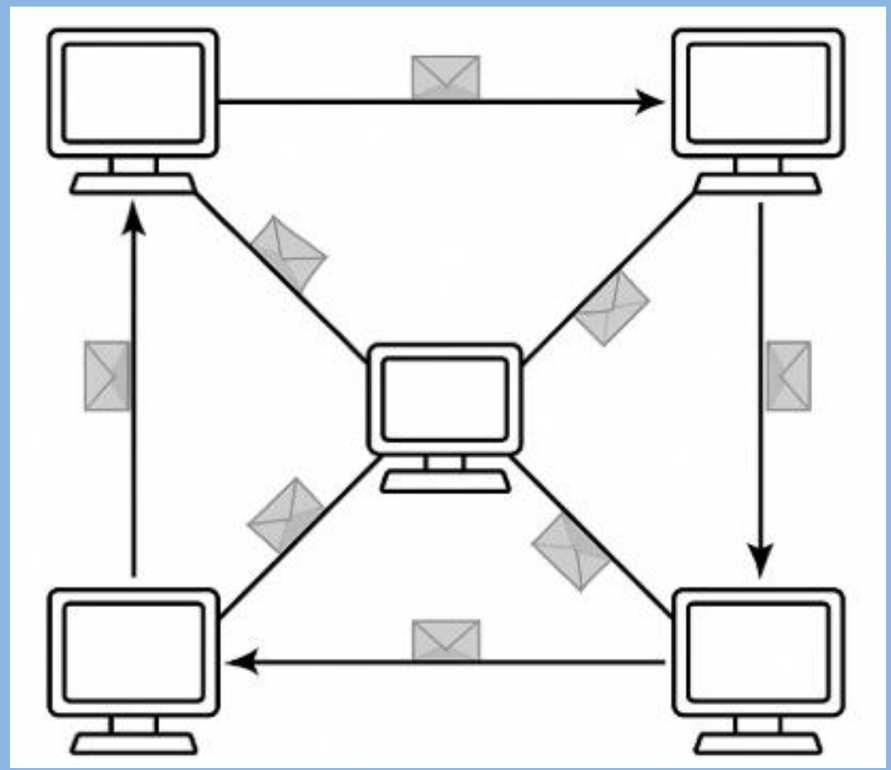


C) Communication takes place through communication ports. A node of degree d corresponds to a computer with d ports that are labeled with numbers 1, 2, . . . , d.

D) Each computer runs a copy of the same deterministic algorithm. All nodes are identical. Initially they know only their own degree.



E) All computers are started simultaneously, and they follow a deterministic algorithm synchronously in parallel. In each synchronous communication round, all computers in parallel:
(1) send a message to each of their ports.
(2) wait while the messages are propagated.
(3) receive a message from each of their ports.
(4) update their own state.



f) After each round, a computer can stop and announce its local output – will be explained afterwards.

(g) All nodes should eventually stop - the running time of the algorithm is the number of communications rounds it takes until all nodes have stopped.

Distributed algorithms in the PN model

Distributed algorithms in the pn model

- A distributed graph problem Π associates a set of solutions $\Pi(N)$ with each simple port-numbered network $N = (V, P, p)$. A solution $f \in \Pi(N)$ is a node labeling $f : V \to Y$ for some set Y of local outputs.
- For example – for the 3 – color problem, each node should be labeled using the function: $f : V \to \{1, 2, 3\}$.

State machine

- A distributed algorithm A is a state machine that consists of the following components:

(i) InputA is the set of local inputs

(ii) StatesA is the set of states

(iii) OutputA $\subseteq$ StatesA is the set of stopping states (local outputs)

(iv) MsgA is the set of possible messages.

State machine cont'

- For each possible degree $d \in \mathbb{N}$ we have the following functions:
  (i) $\text{init}_{A,d} : \text{Input}_{A,d} \to \text{States}_{A,d}$ initializes the state machine.
  (ii) $\text{send}_{A,d} : \text{States}_{A,d} \to \text{Msg}_{A,d}$ constructs outgoing messages.
  (iii) $\text{receive}_{A,d} : \text{States}_{A,d} \times \text{Msg}_{A,d} \to \text{States}_{A,d}$ processes incoming messages.
- We require that $\text{receive}_{A,d}(x, y) = x$ whenever $x \in \text{Output}_A$. The idea is that a node that has already stopped and returned its local output, no longer changes its state.

Solving Graph Problems

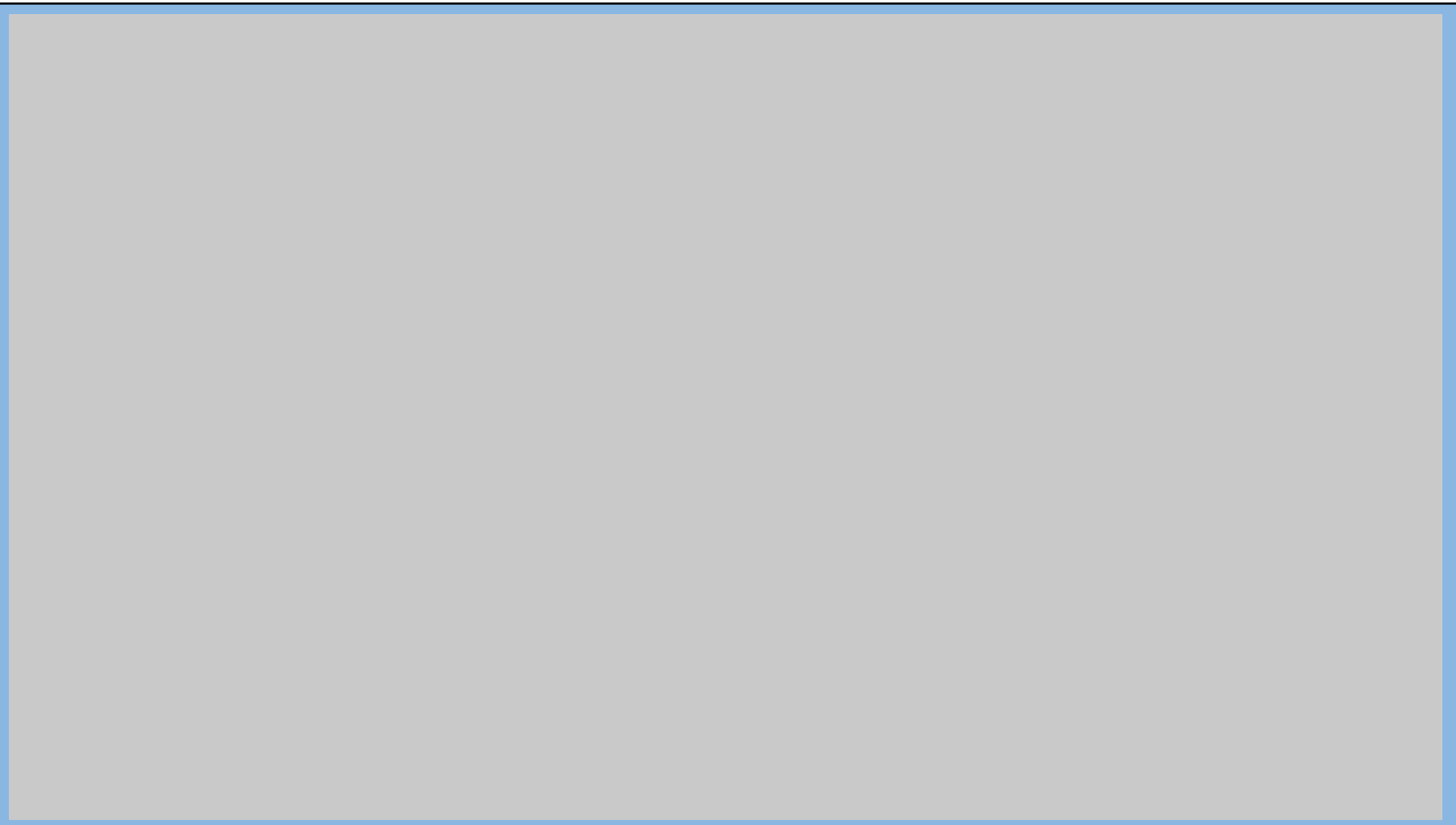
Let F be a family of simple undirected graphs. Let Π and Π’ be distributed graph problems. Assuming that:

- (a) N = (V, P, p) is a simple port-numbered network.
- (b) the underlying graph of N is in F
- (c) the input f is in Π’ (N)

We say that distributed algorithm A solves problem Π on graph family F given Π’ if the following holds:

- The execution of algorithm A on (N, f) stops and produces an output g ∈ Π(N). If A stops in time T(|V|) for some function T : N → N, we say that A solves the problem in time T.

video



Algorithm for 3 – coloring path graphs

Coloring Paths

- Let us start with the problem definition:
  - F is the family of path graphs.
  - Π is the problem of coloring graphs with 3 colors.
  - Π ’ is the problem of coloring graphs with any number of colors.
- We will present algorithm A that solves problem Π on graph family F given Π’.
- It is sufficient to assume that we have some graph coloring, i.e., a solution to problem Π’.

Coloring Paths cont'

The set of local inputs is determined by what we assume as input:

$$InputA = Z+$$

The set of stopping states is determined by the problem that we are trying to solve:

$$OutputA = \{1, 2, 3\}$$

In our algorithm, each node only needs to store one positive integer (the current color):

$$StatesA = Z+$$

Messages are also integers:

$$MsgA = Z+$$

Coloring Paths state machine

Initialization is trivial: the initial state of a node is its color. Hence for all d we have-

$$\text{init}_{A,d}(x) = x.$$

In each step, each node sends its current color to each of its neighbors. As we assume that all nodes have degree at most 2, we only need to define $\text{send}_{A,d}$ for $d \leq 2$:

$$\text{send}_{A,0}(x) = ()$$
$$\text{send}_{A,1}(x) = (x)$$
$$\text{send}_{A,2}(x) = (x, x)$$

The receive function

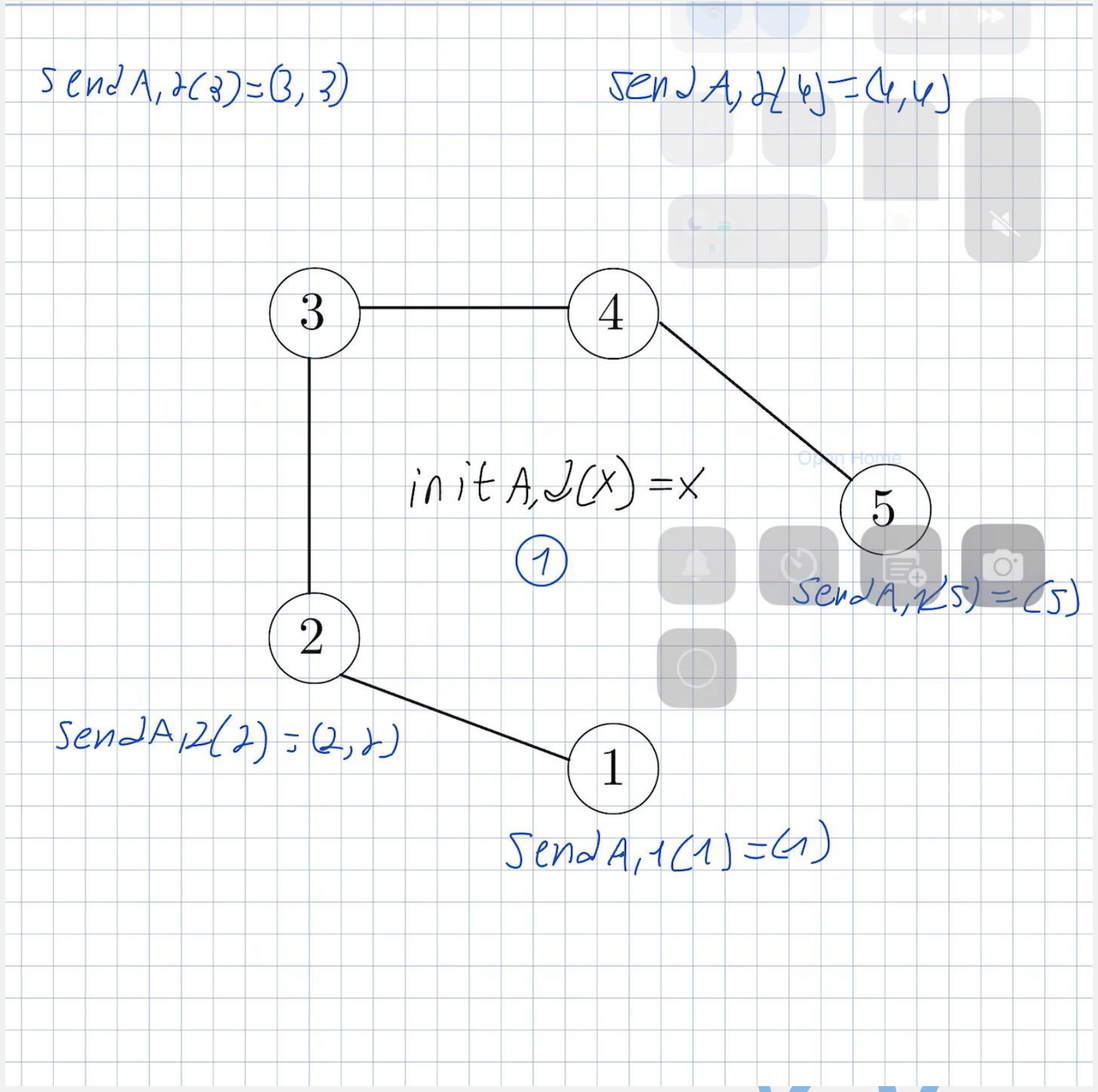
We will use the following auxiliary function that returns the smallest positive number not in X: g(X) = min(Z+\X).

$$\text{receive}_{A,0}(x,()) = \begin{cases} g(\varnothing) & \text{if } x \notin \{1,2,3\}, \\ x & \text{otherwise.} \end{cases}$$

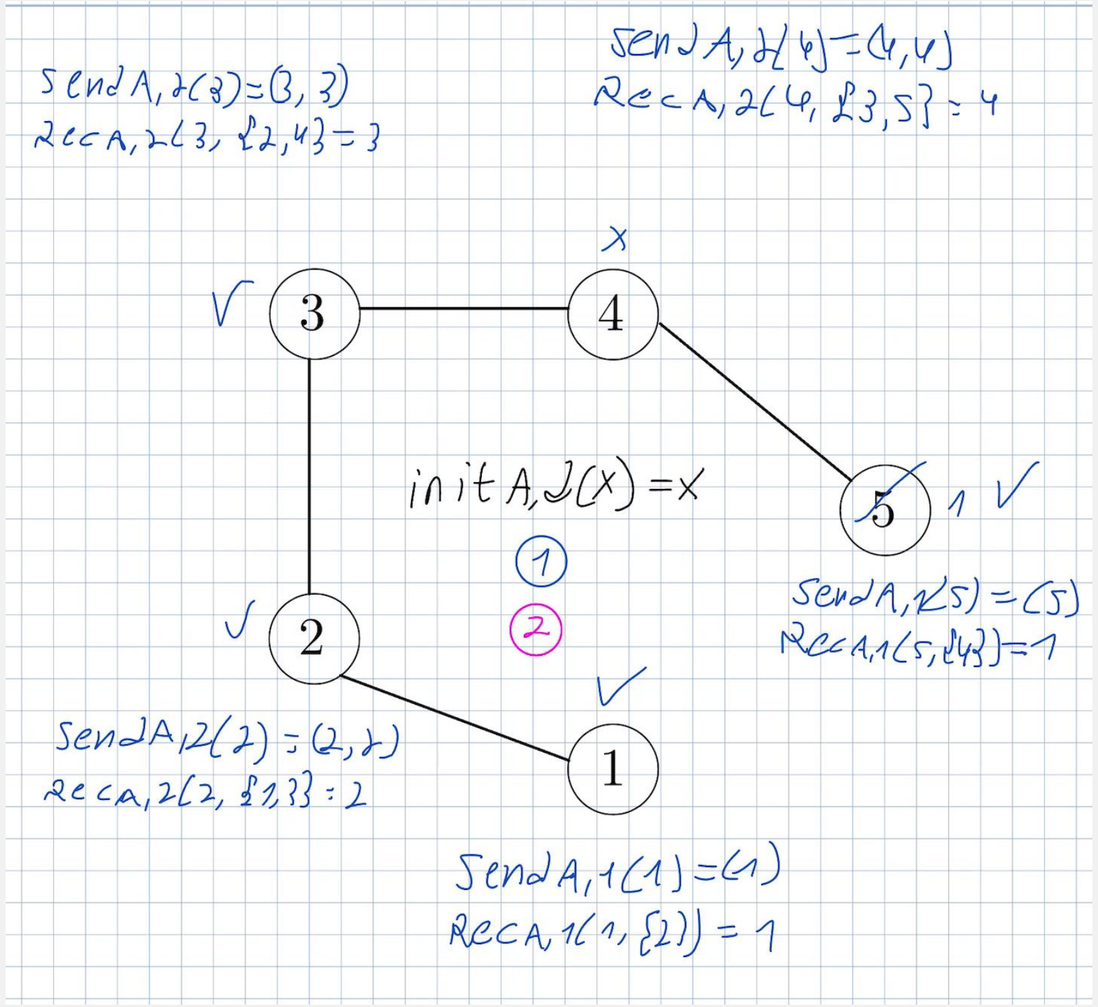
$$\text{receive}_{A,1}(x,(y)) = \begin{cases} g(\{y\}) & \text{if } x \notin \{1,2,3\} \\ & \text{and } x > y, \\ x & \text{otherwise.} \end{cases}$$

$$\text{receive}_{A,2}(x,(y,z)) = \begin{cases} g(\{y,z\}) & \text{if } x \notin \{1,2,3\} \\ & \text{and } x > y,\ x > z, \\ x & \text{otherwise.} \end{cases}$$

Demonstration



Demonstration cont’



Coloring Paths proof

To prove the correctness of the distributed algorithm for 3-coloring path graphs, we need to show that:

- **Each node eventually stabilizes** (i.e., reaches a final state that no longer changes).
- **The final coloring is proper**: no two adjacent nodes share the same color.
- **The coloring uses only colors in {1, 2, 3}**.

Coloring Paths proof

Each node eventually stabilizes:

Each node only reduces its color when it is:
Not yet valid (i.e., not in {1,2,3}), And greater than neighbors.

Let $u \in V$ be a node not in {1, 2, 3}, with smaller color then at least one of its neighbors. Because it is a graph path, the algorithm will eventually find a node that is bigger from his neighbors – assume for contradiction that there isn't, and because there a two leaves, we reach this situation.

$u1 < u2 < u3 < \ldots < un\text{-}2 \mathbf{<\ un\text{-}1\ >}\ un$

Coloring Paths proof

Proper Coloring:

Let’s argue that once all nodes stabilize, no two neighbors have the same color. Suppose for contradiction that two neighbors u and v end up with the same color $c \in \{1,2,3\}$.

Let’s assume without loss of generality that node u made the last change to color c after seeing its neighbor v already had color c. Then it should have chosen a different color via $g(\{c\}) \neq c$ (since $c \in X$), which contradicts the definition of the receive function.
Thus, no two adjacent nodes can share a color in a fixed point.

Coloring Paths proof

3-Color Validity:

The g(X) function always returns a color from {1,2,3} because:
The maximum size of X is 2 (degree ≤ 2), so the missing smallest element from {1,2,3} is always ≤ 3.

Therefore, all nodes eventually get a color in {1,2,3}.

Coloring Paths runtime

Let n be the number of nodes, and C be the maximum initial value of a node in the network.

The worst case occurs when: All nodes start with large unique values (e.g., C ,C – 1, …,C – n + 1).

The node with the largest color must reduce first, then the next largest, and so on (like the example!).

This makes the behavior similar to a wavefront propagating from one side to the other.
So, in the worst case, the update must travel across the entire path from one end to the other and back – O(n).

Algorithm for maximal matching in bipartite graphs.

Maximal Matching in Two-Colored Graphs

- Now we will present a distributed bipartite maximal matching algorithm.
- It finds a maximal matching in 2 - colored graphs. That is, F is the family of bipartite graphs, we are given a 2 - coloring $f : V \to \{1, 2\}$, and the algorithm will output an encoding of a maximal matching $M \subseteq E$.
- We say that a node $v \in V$ is white if $f(v) = 1$, and it is black if $f(v) = 2$.

Preview for the algorithm

- During the execution of the algorithm, each node is in one of the states:
  $\{ UR, MR(i), US, MS(i) \}$
- These stand for “unmatched and running”, “matched and running”, “unmatched and stopped”, and “matched and stopped”, respectively.
- US and MS(i) are stopping states.
- If the state of a node v is MS(i) then v is matched with the neighbor that is connected to port i. Initially, all nodes are in state UR.
- Each black node v maintains variables M(v) and X(v), which are initialized: $M(v) \leftarrow \emptyset$, $X(v) \leftarrow \{1, 2, \ldots, deg(v)\}$.

The algorithm

Round 2k – 1, white nodes:

- State UR, k ≤ degN (v): Send ‘proposal’ to port (v, k).
- State UR, k > degN (v): Switch to state US.
- State MR(i): Send ‘matched’ to all ports. Switch to state MS(i).

Round 2k – 1, black nodes:

- State UR: Read incoming messages. If we receive ‘matched’ from port i, remove i from X(v). If we receive ‘proposal’ from port i, add i to M(v).

The algorithm – cont'

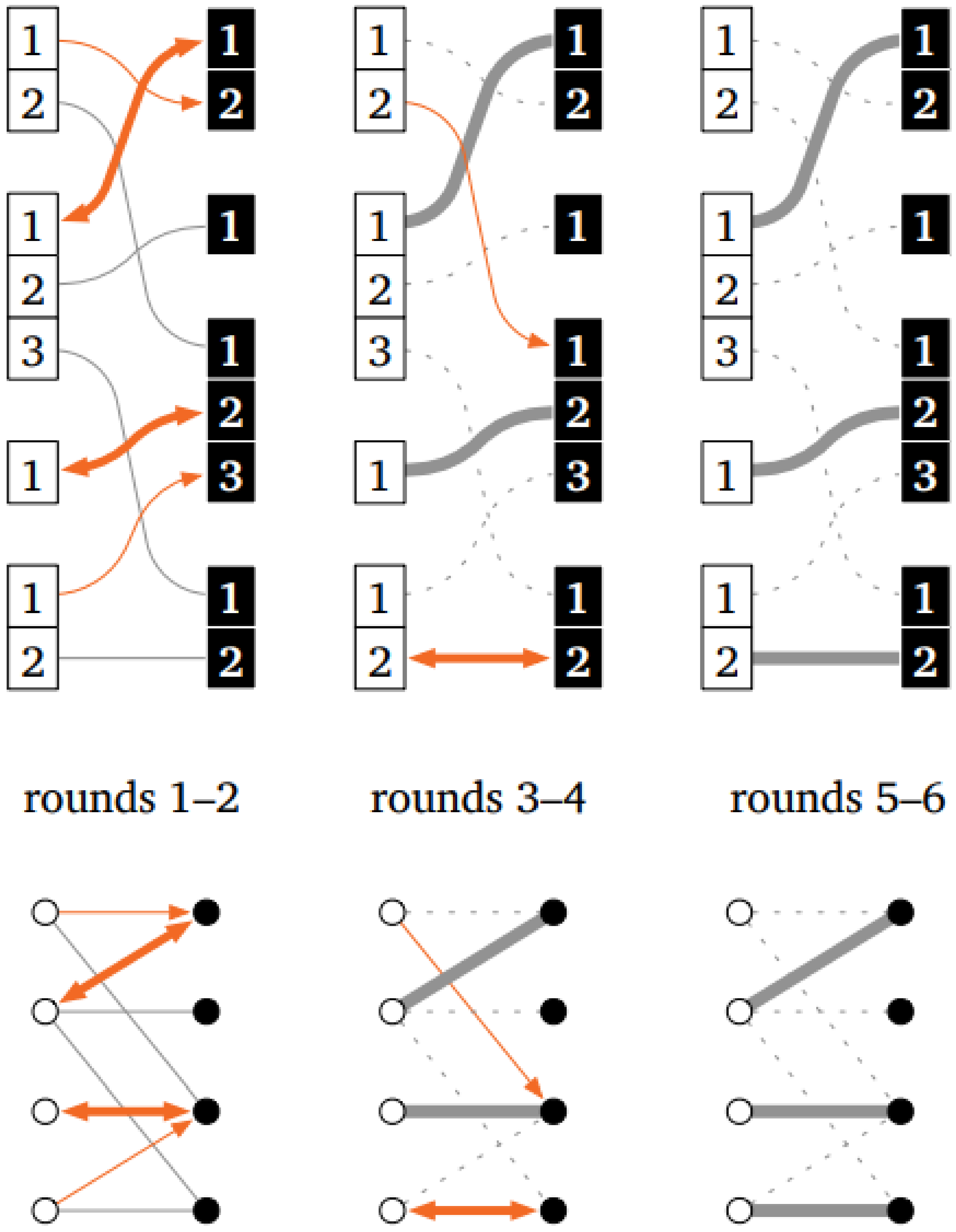
Round 2k, black nodes:

• State UR, M(v)≠ ∅: Let i = min M(v). Send ‘accept’ to port (v, i). Switch to state MS(i).

• State UR, X(v) = ∅: Switch to state US.

Round 2k, white nodes:

• State UR: Process incoming messages. If we receive ‘accept’ from port i, switch to state MR(i).


rounds 1–2
rounds 3–4
rounds 5–6

Lemma 1

Assume that u is a white node, v is a black node, and (u, i) = p(v, j). Then at least one of the following holds:
(a) element j is removed from X(v) before round 2i.
(b) at least one element is added to M(v) before round 2i.

Proof: Assume that we still have $M(v) = \emptyset$ and $j \in X(v)$ after round $2i - 2$. This implies that v is still in state UR, and u has not sent 'matched' to v. In particular, u is in state UR or MR(i) after round $2i - 2$. In the former case, u sends 'proposal' to v on round $2i - 1$, and j is added to M(v) on round $2i - 1$. In the latter case, u sends 'matched' to v on round $2i - 1$, and j is removed from X(v) on round $2i - 1$.

Lemma 2 – Correctness

The bipartite maximal matching algorithm finds a maximal matching in any two-colored graph.

Proof: Let us first verify that the output correctly encodes a matching. In particular, assume that u is a white node, v is a black node, and p(u, i) = (v, j). We have to prove that u stops in state MS(i) if and only if v stops in state MS(j). If u stops in state MS(i), it has received an ‘accept’ from v, and v stops in state MS(j). Conversely, if v stops in state MS(j), it has received a ‘proposal’ from u and it sends an ‘accept’ to u, after which u stops in state MS(i).

Let us then verify that M is indeed maximal. If this was not the case, there would be an unmatched white node u that is connected to an unmatched black node v. However, the first Lemma implies that at least one of them becomes matched before or during round 2Δ.

Lemma 3 – Runtime

The bipartite maximal matching algorithm stops in time $2\Delta + 1$, where Δ is the maximum degree of N.

Proof: A white node of degree d stops before or during round $2d + 1 \leq 2\Delta + 1$. Now let us consider a black node v. Assume that we still have $j \in X(v)$ on round 2Δ. Let $(u, i) = p(v, j)$; note that $i \leq \Delta$. By the first Lemma, at least one element has been added to M(v) before round 2Δ. In particular, v stops before or during round 2Δ.

Thank you very much!

Questions?