YO-YO

Election in arbitrary graphs:
simple but not optimal

Yo-Yo: Structure of the Algorithm

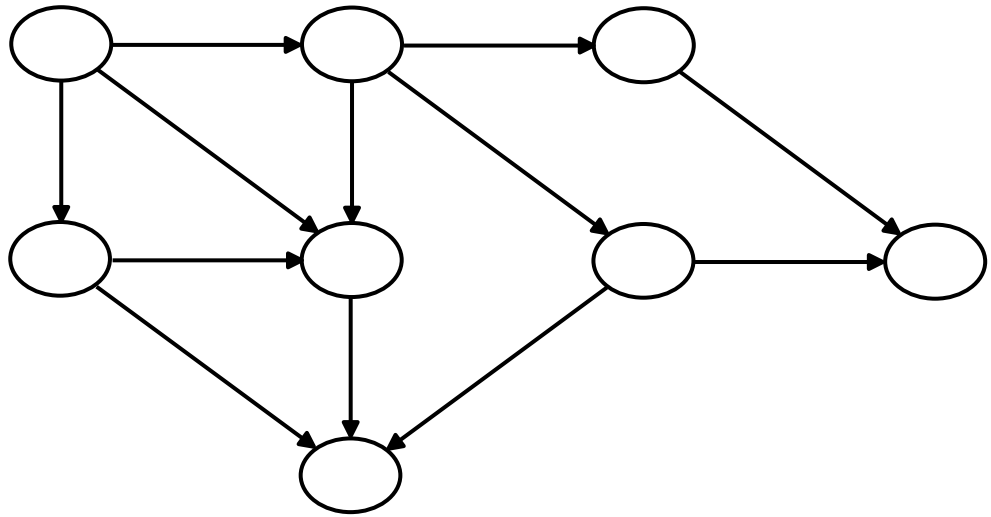
A minimum finding algorithm that consists of 2 parts

1. Pre-processing phase/setup
   - Each entity x exchanges its id with all its neighbors
   - Then, x orients its links in the direction of the entity with largest id

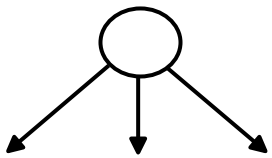
2. Sequence of iterations
   - Each iteration works as an electoral stage in which some candidates are defeated and do not continue in the next stage
   - Each iteration is divided in two steps

Yo-Yo

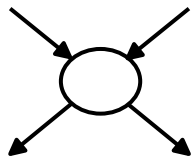
DAG =
Directed Acyclic Graph



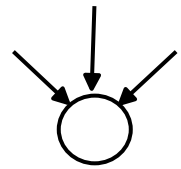
SOURCE:



INTERNAL NODE:

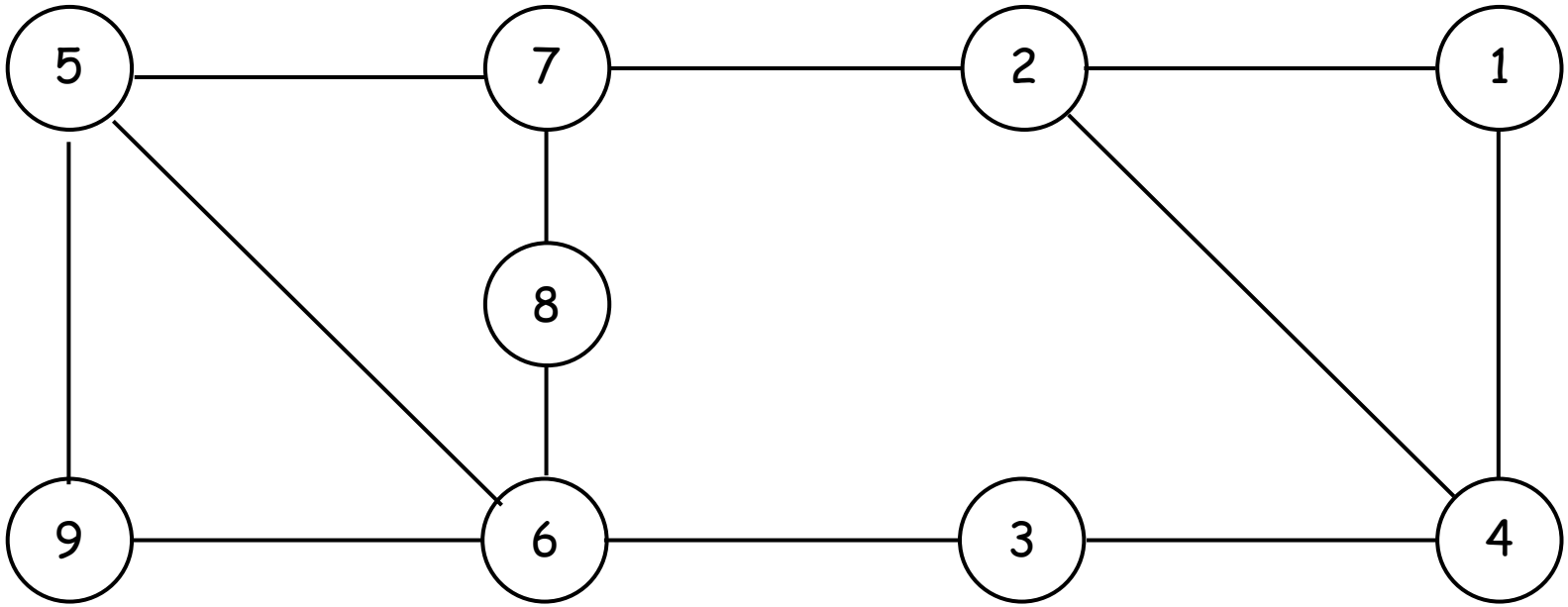


SINK:



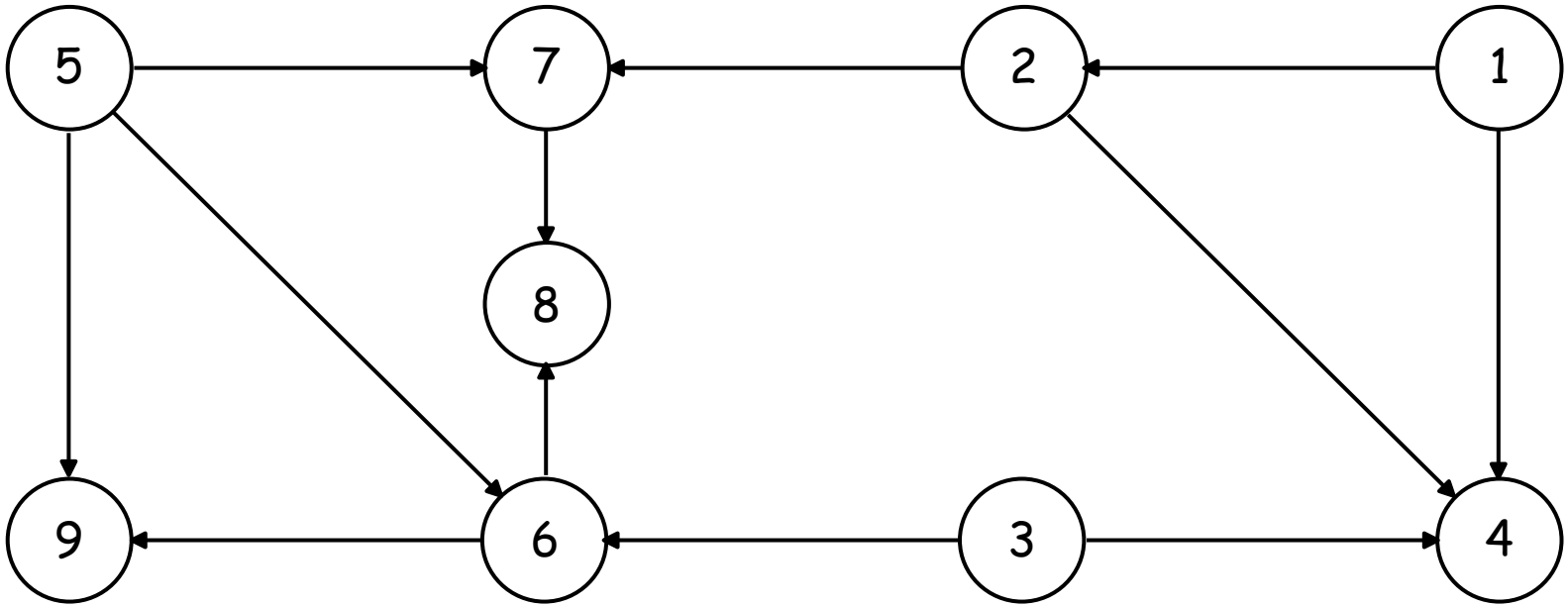
Initialization phase

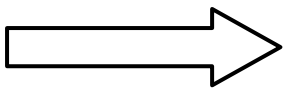
Given an arbitrary undirected graph



Initialization/setup phase

Given an arbitrary undirected graph, smaller entities are directed towards bigger (by exchanging messages with neighbours)





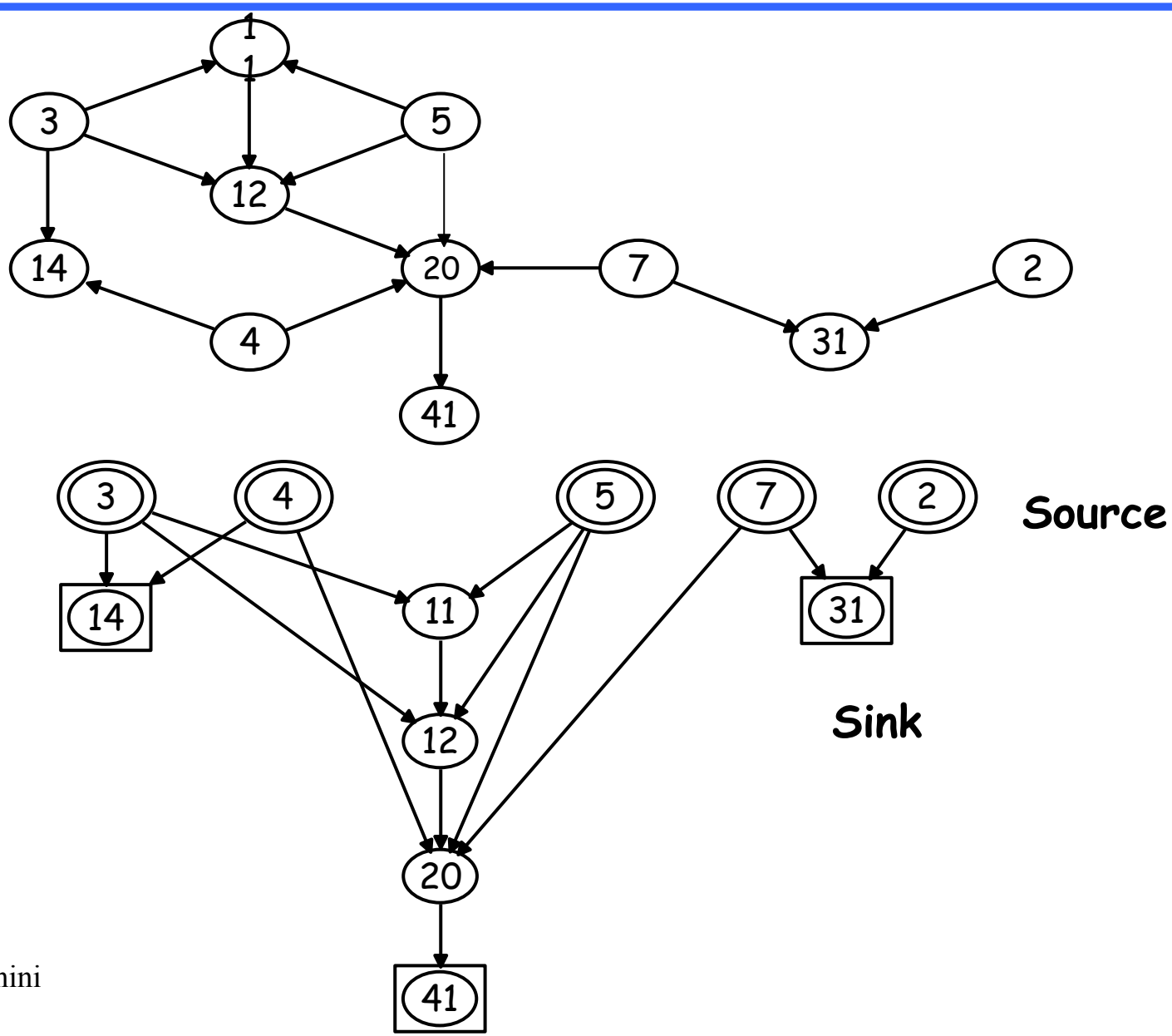
DAG

Property of the setup phase

Property. The graph resulting from the setup phase is a DAG

Proof. (Sketch)
By contradiction.
Assume a cycle: $x_0, x_1, \ldots, x_k$ where $x_k = x_0$
Thus, it holds $id(x_0) < id(x_1) < \ldots < id(x_k)$.
To be true, it must be $id(x_{k-1}) < id(x_k) = id(x_0)$,
impossible

DAG



The iteration stage

Two steps

Yo- this step is started by the sources

1. Each source sends its id to all its neighbors

2. An internal node waits until it receives all the ids from incoming neighbors. It computes the minimum and sends it to all its out-neighbors.

3. A sink waits until its receives all values from its in-neighbors. It computes the minimum and starts the second step.

The iteration stage

Two steps

-Yo this step is started by the sinks

1. Each sink answers with YES to all in-neighbors which sent the smallest value

2. An internal node waits until it receives the vote from all out-neighbors. If all votes are YES, it sends YES otherwise NO

3. A source waits until its receives the vote from all its in-neighbors. If all votes are YES, it survives and goes to next iteration

Before next iteration

Before starting the next iteration we need to modify the DAG so that sources that survived are still sources

We flip the links where a NO vote was sent

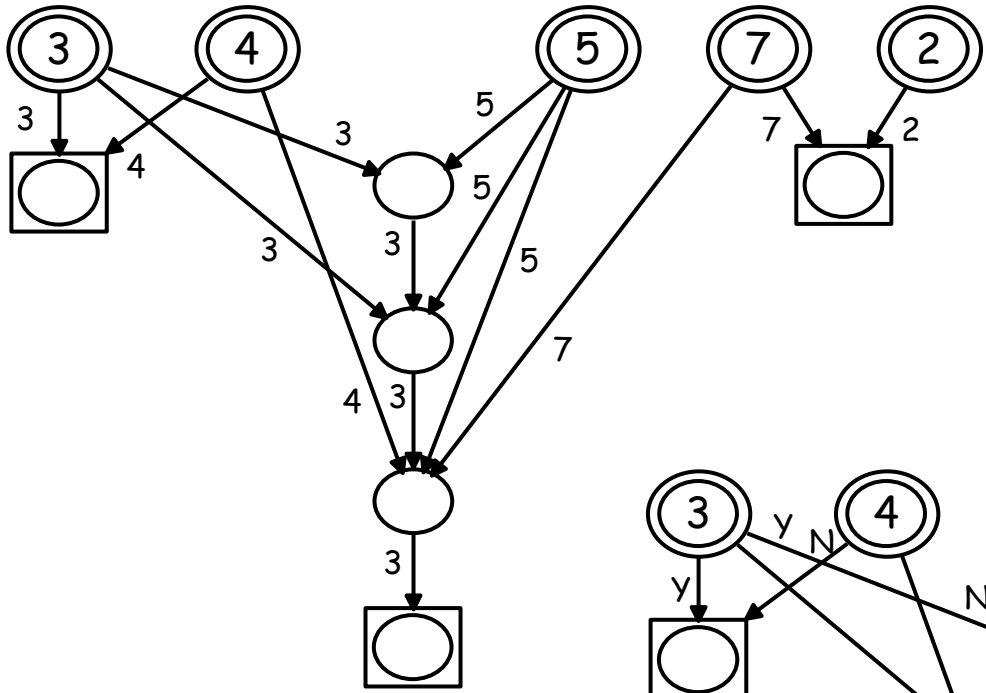
4. When a node sends NO to an in-neighbor, it reverses the direction of the link

5. When a node receives a NO on a link, it reverses the direction of that link

Any source that receives a NO will cease to be a source and becomes a sink; some sinks become internal node and some internal nodes become sink.

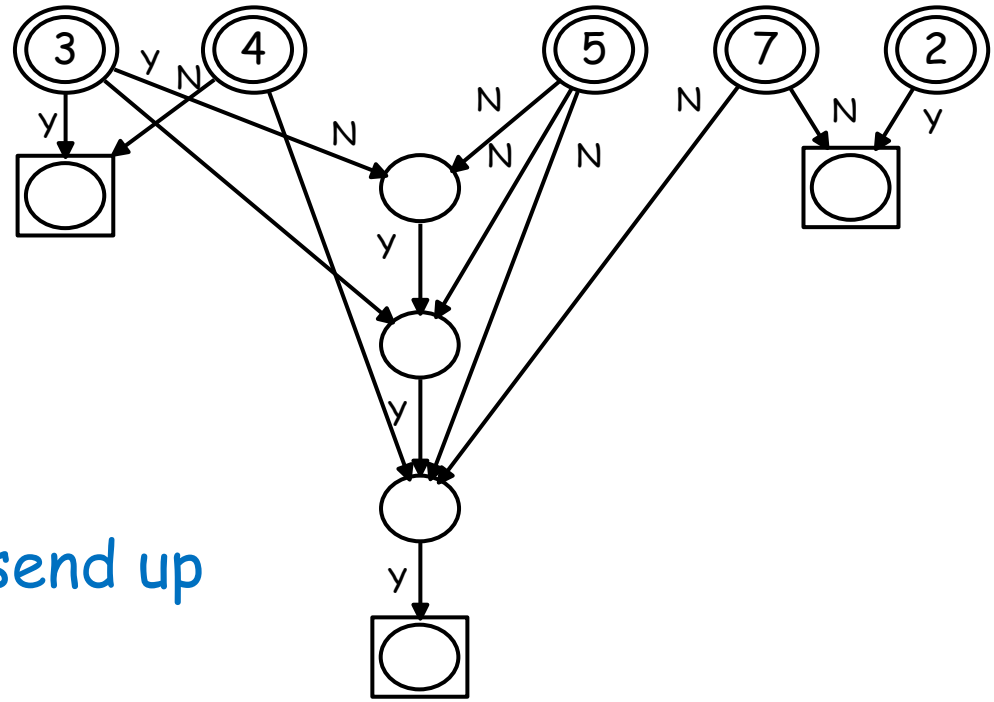
The number of sources decreases at each iteration

The protocol



Yo: send down

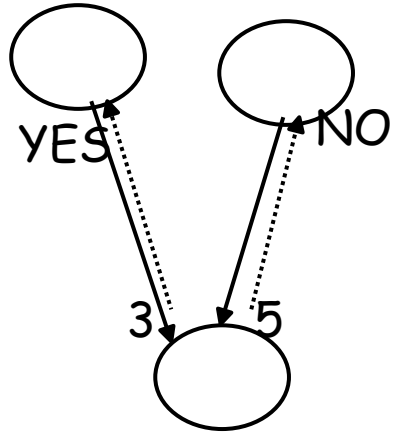
Yes to min
No to others

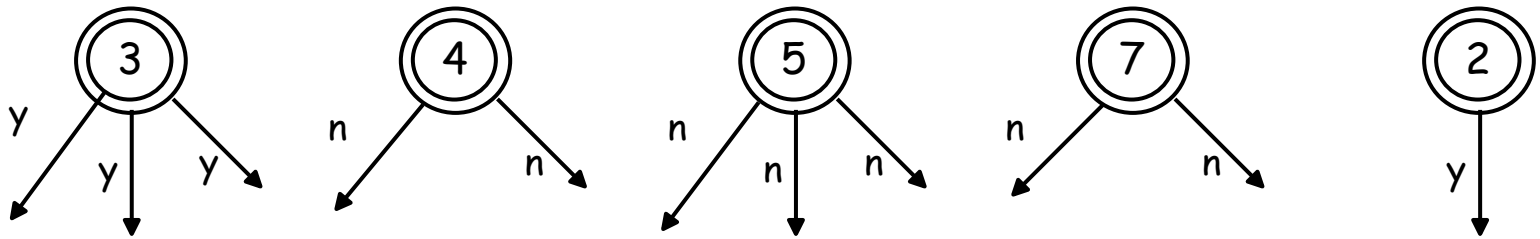


Yo: send up

The protocol

Yes to min
No to others

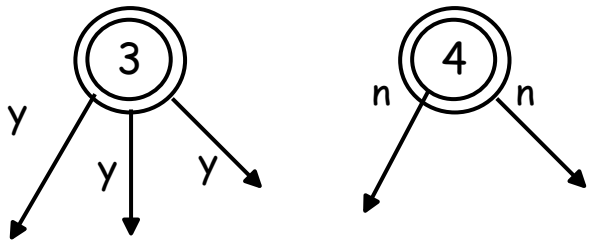




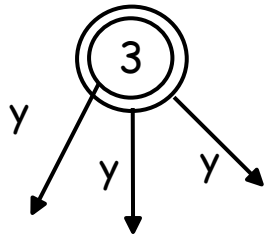
The protocol

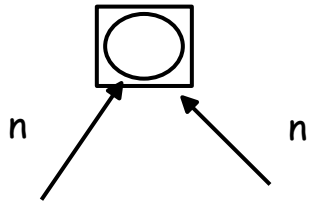
The direction of the links with "NO" will be flipped

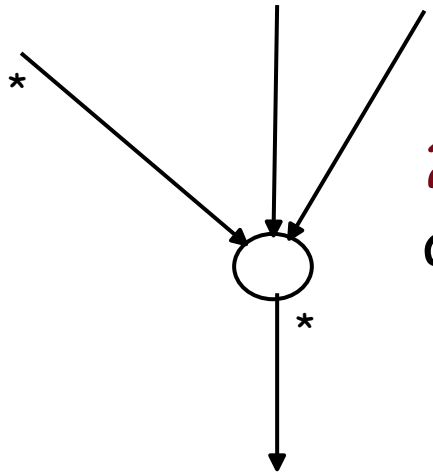
The sources with at least a "NO" will become sinks or internal nodes



1. At each step at least a source becomes either a sink, or internal





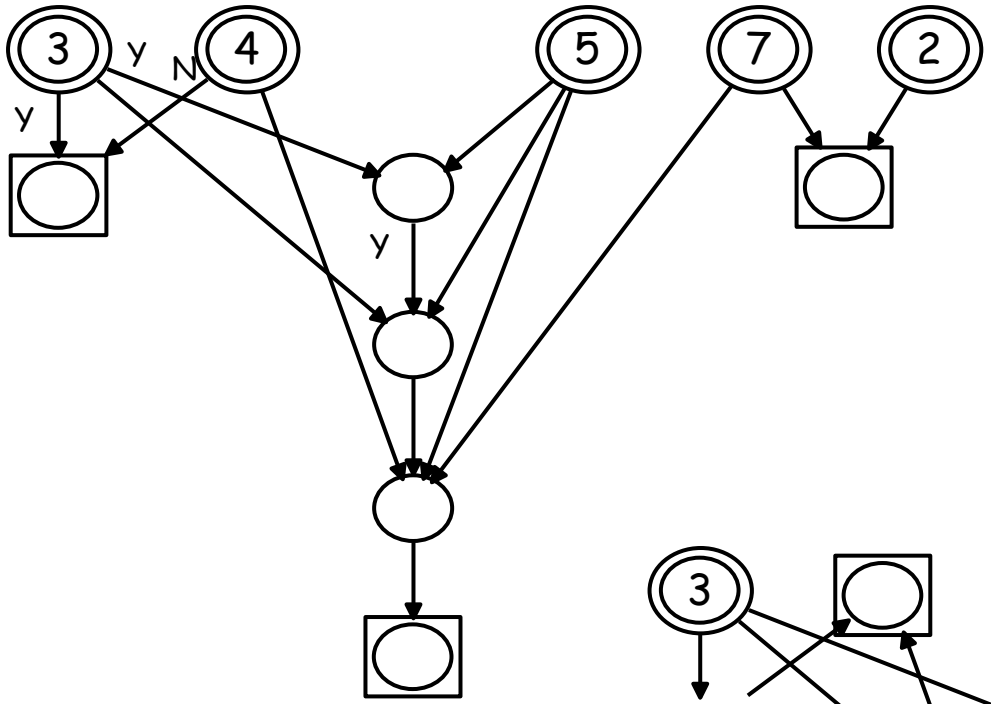


2. An internal node cannot become source after the flipping

3. A sink cannot become source after the flipping

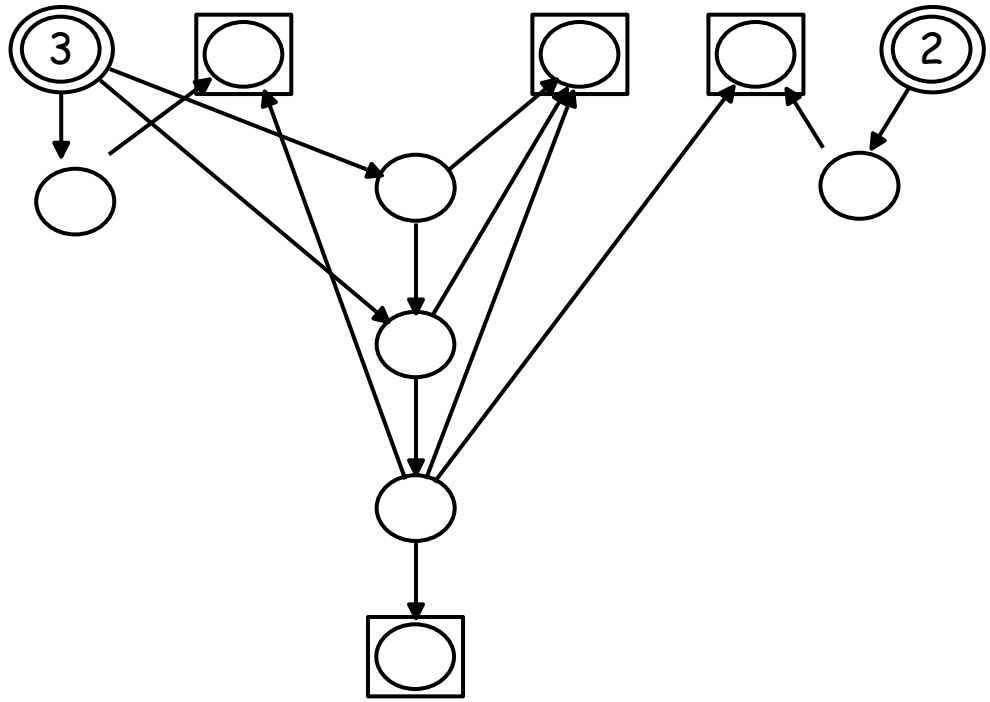
From 1. 2. and 3.:

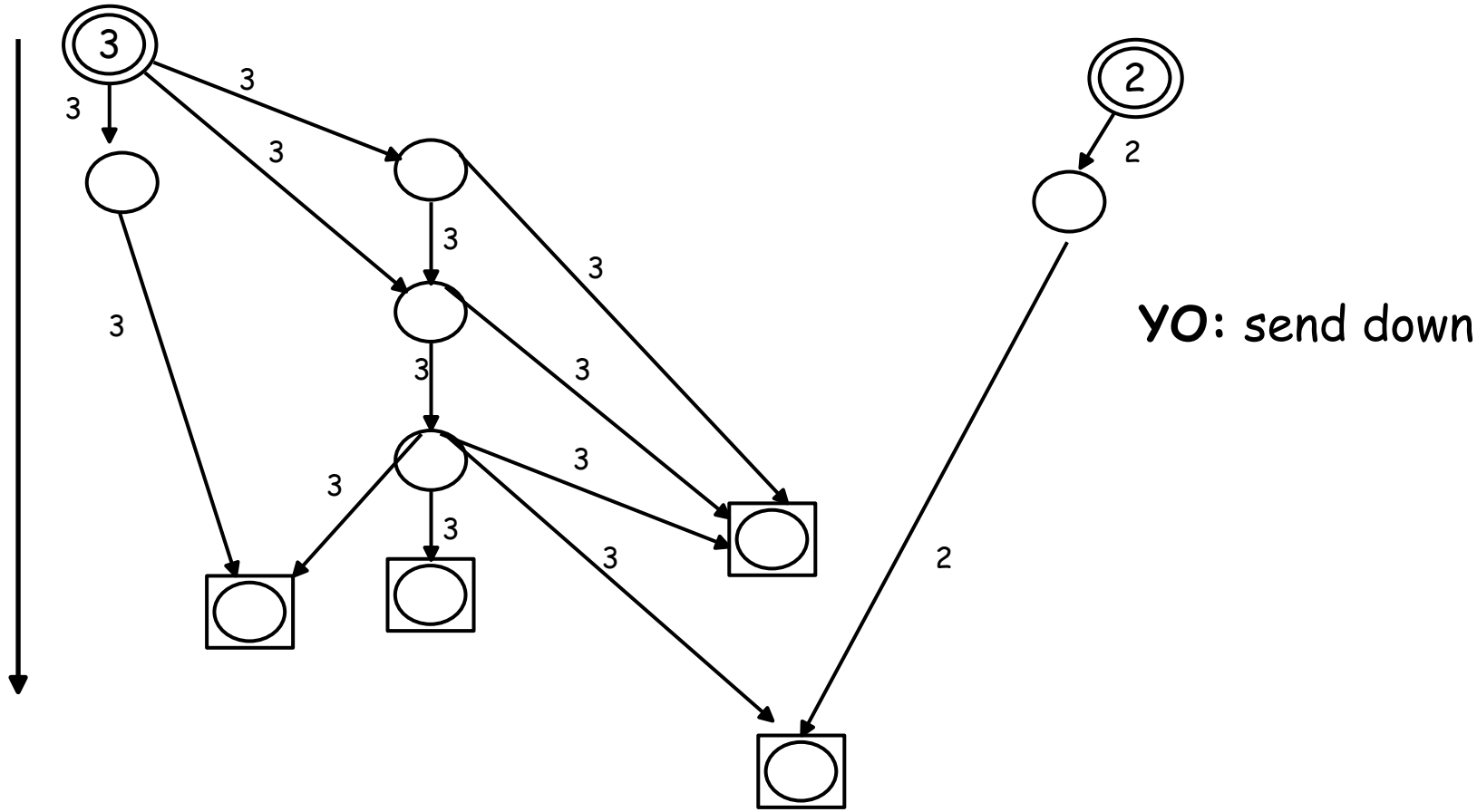
The number of sources is reduced monotonically

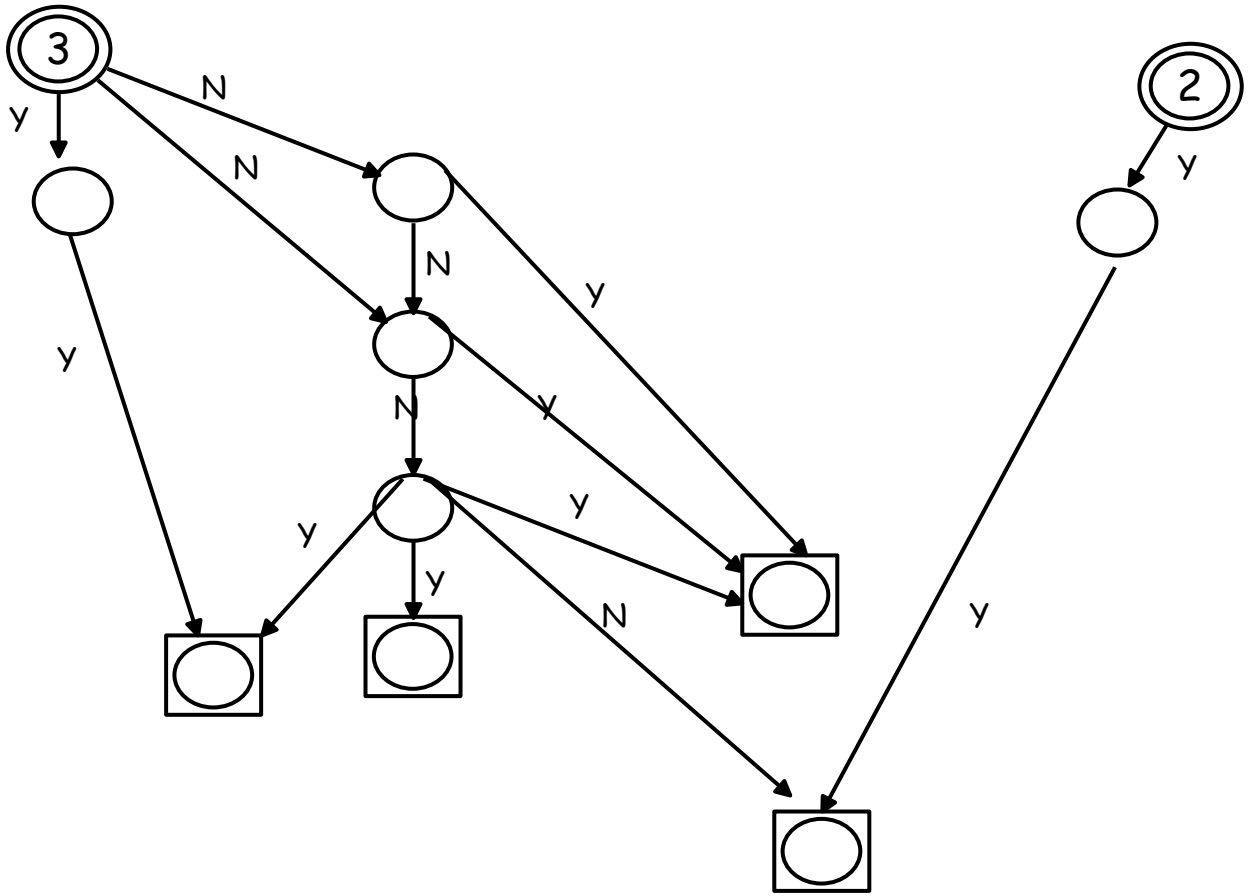


YO: send up

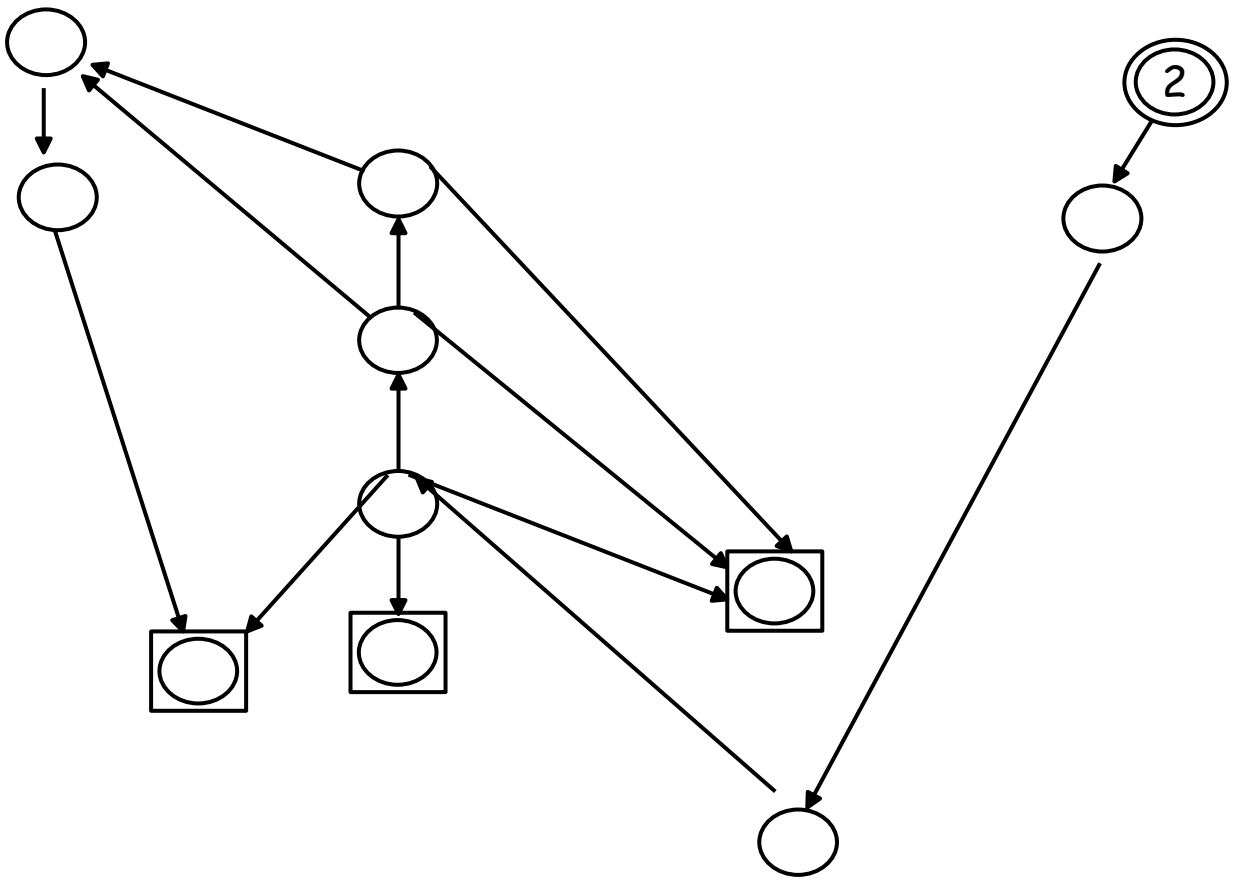
FLIP




3
3
3
3
3
3
3
3
3
3
3
3
3
2
2
2
YO: send down



YO: send up



FLIP

Termination?

The idea: the node with the minimum id will survive to all iterations

Question: how can this node understand that it is the winner?
It will receives YES messages in each iteration, even when it defeated all other candidates.

Solution: Pruning
Remove from computation nodes and links that are useless. At the end only one node is "alive", the winner

Pruning

Pruning is achieved through two rules

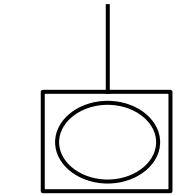
1. If a sink is a leaf (only one neighbor), then it is useless and can be pruned. The parent may become a sink.

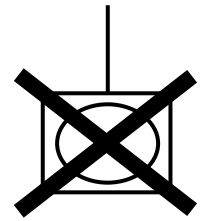
2. If a node in the yo- step receives the same value from many in-neighbor, it can ask all of them except one to prune the links

Pruning can be performed during the voting, by putting information inside the messages

Pruning Rules

1)

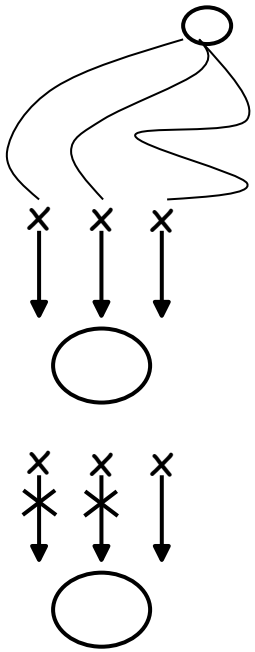




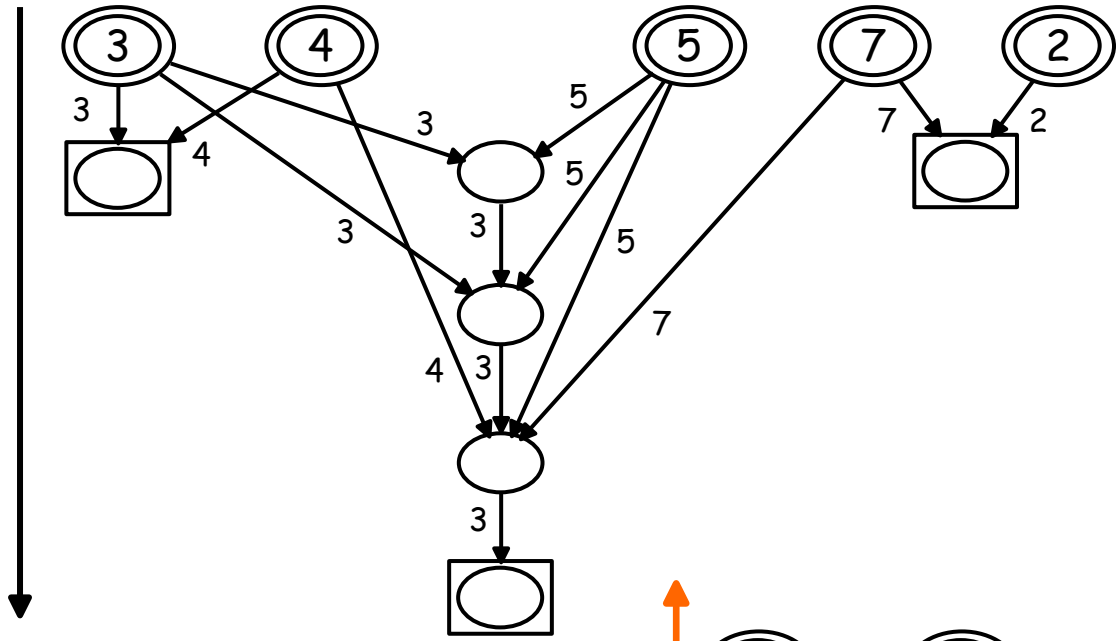
The sinks with one incoming link can be eliminated

(The decision of a sink is the same as the one of the parent)

2)

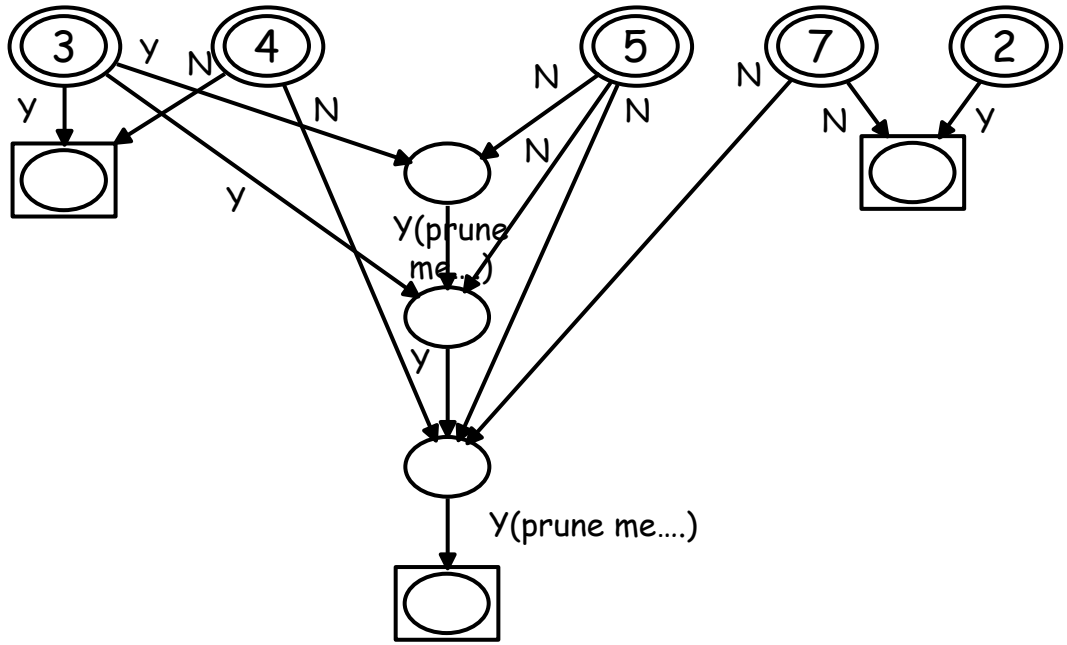


Nodes that receive several identical values:
All the links with the same value can be eliminated except one
(They come from the same source - the decision is the same)

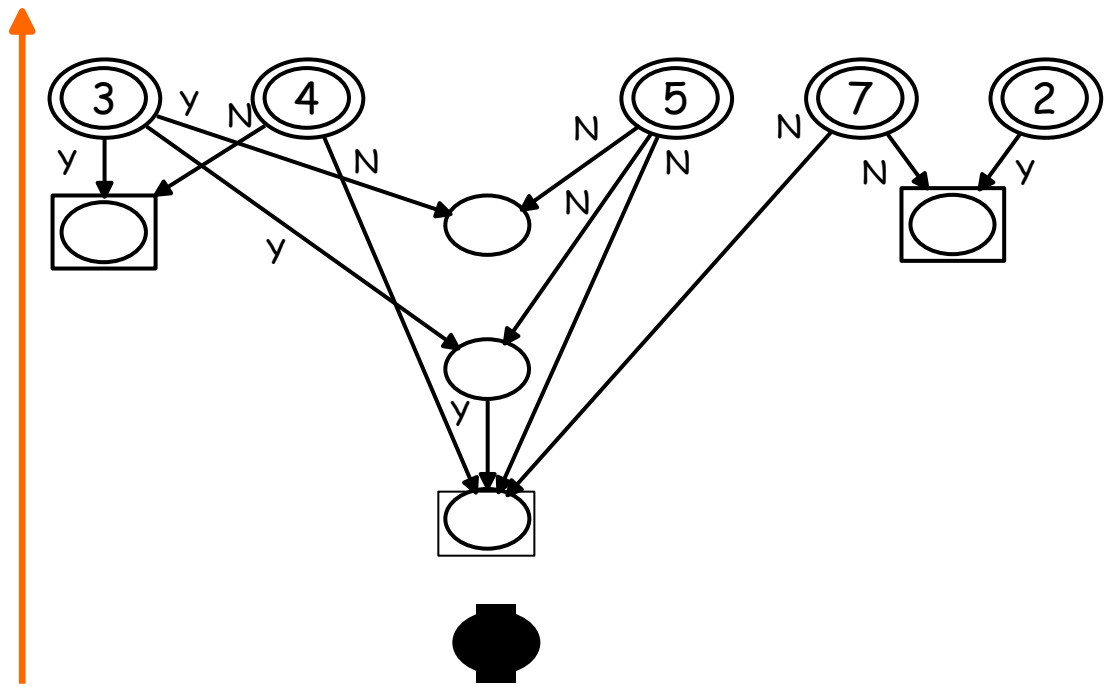


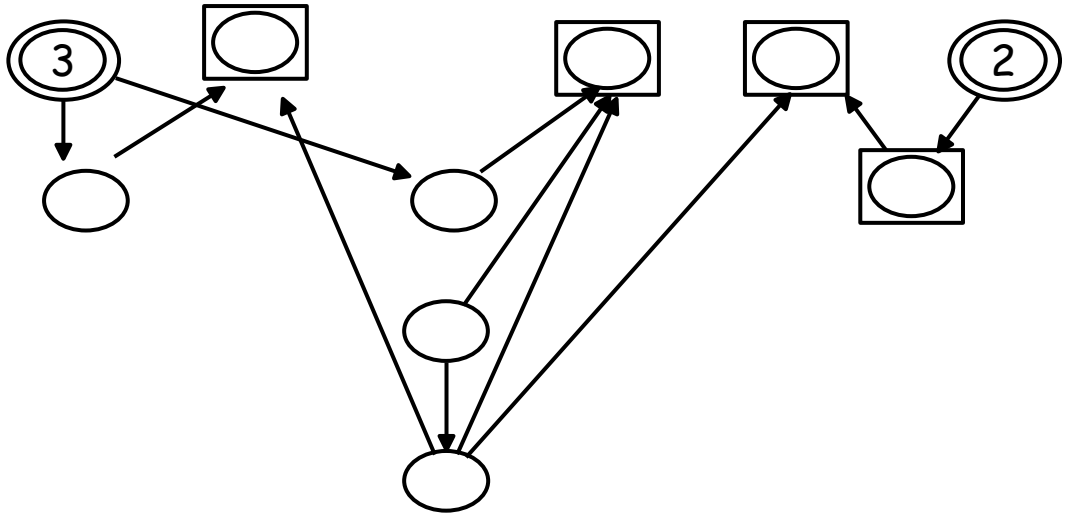
YO -down

YO - up + prune info

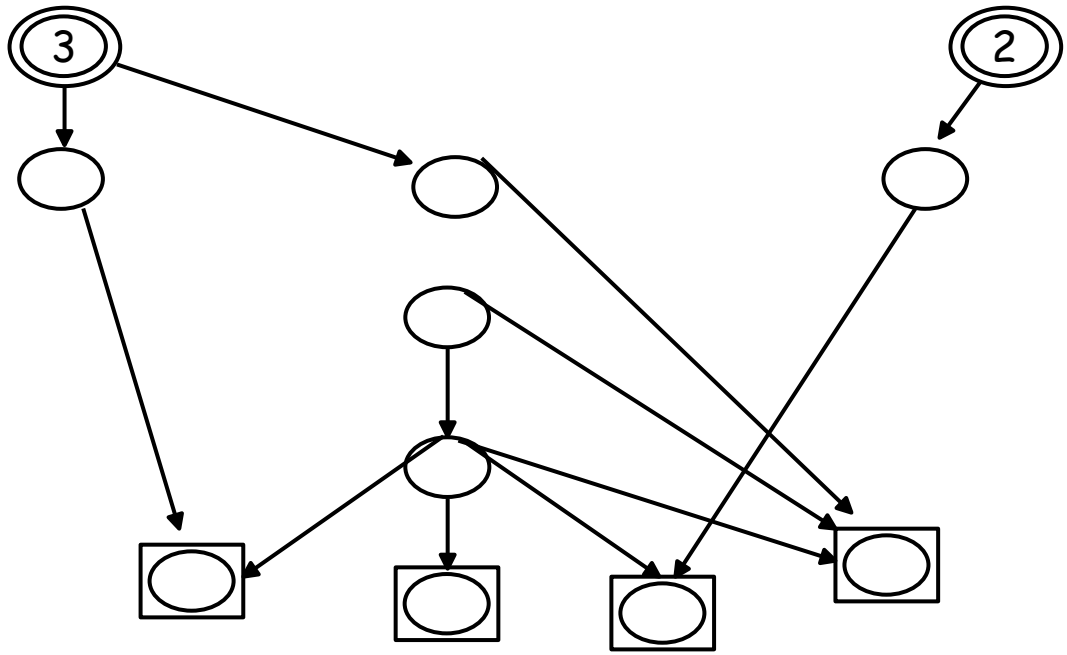


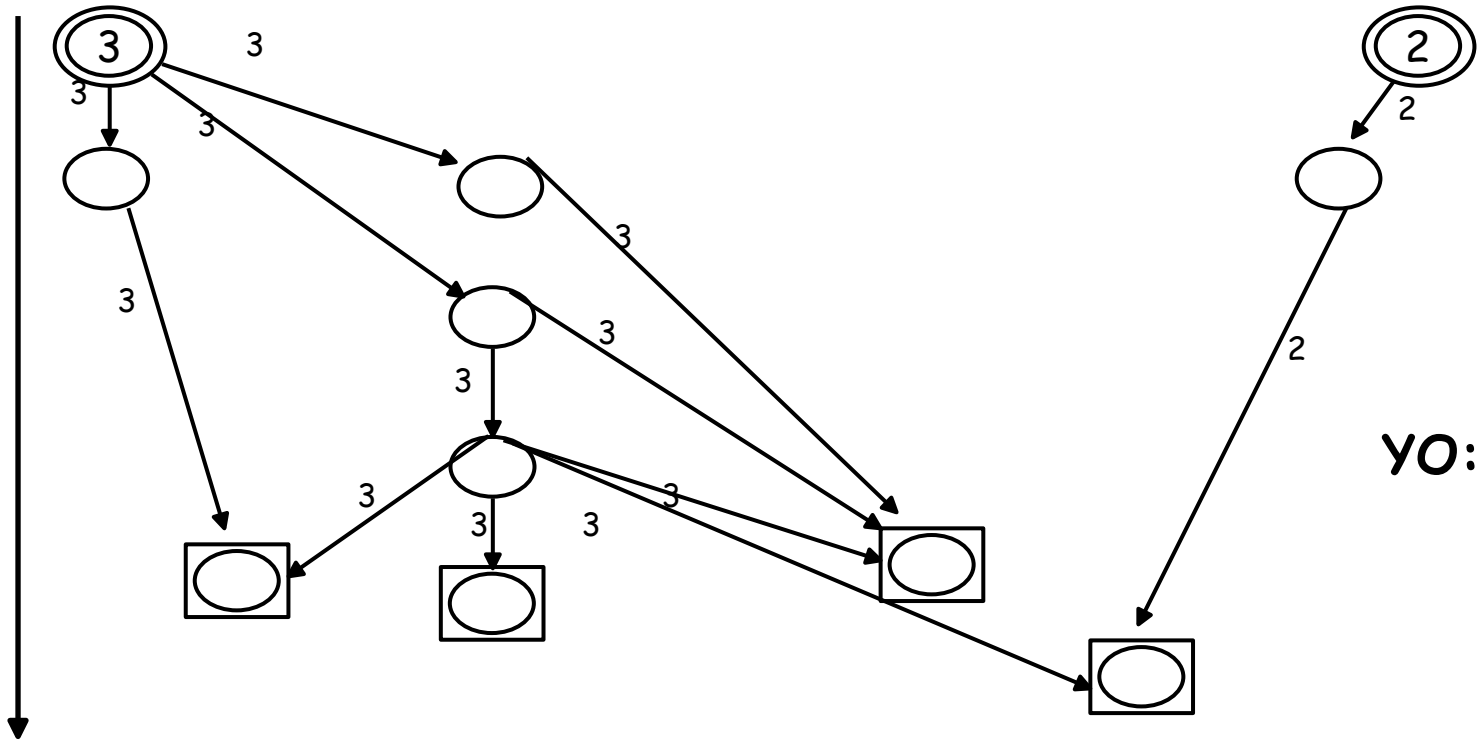
At the same time …. prune



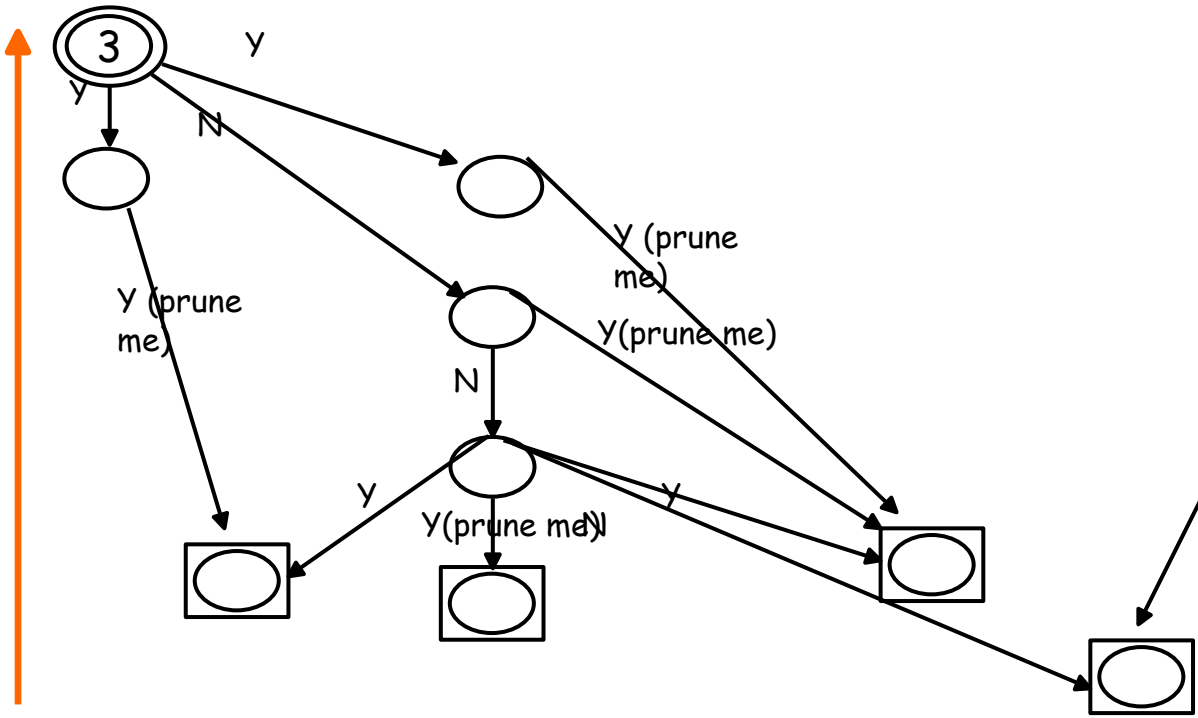


FLIP

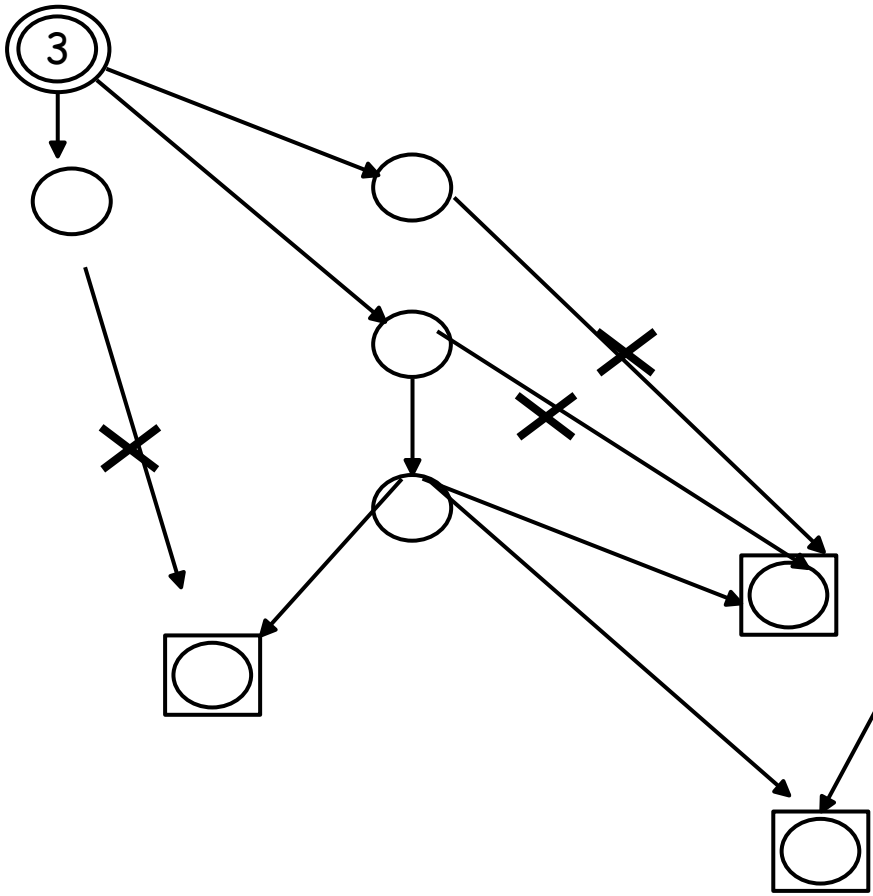




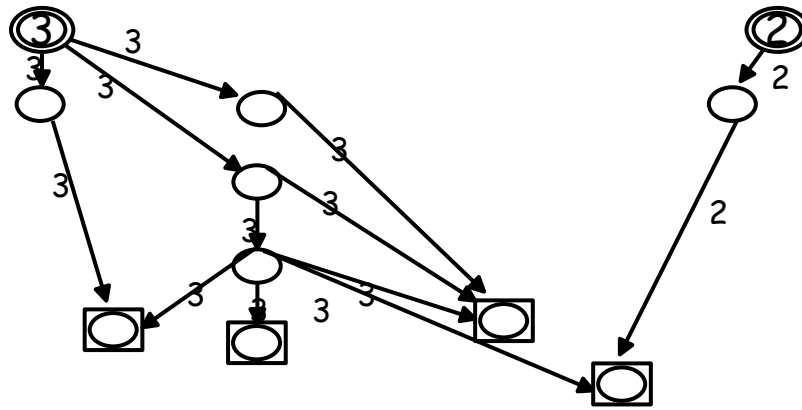
YO: send down

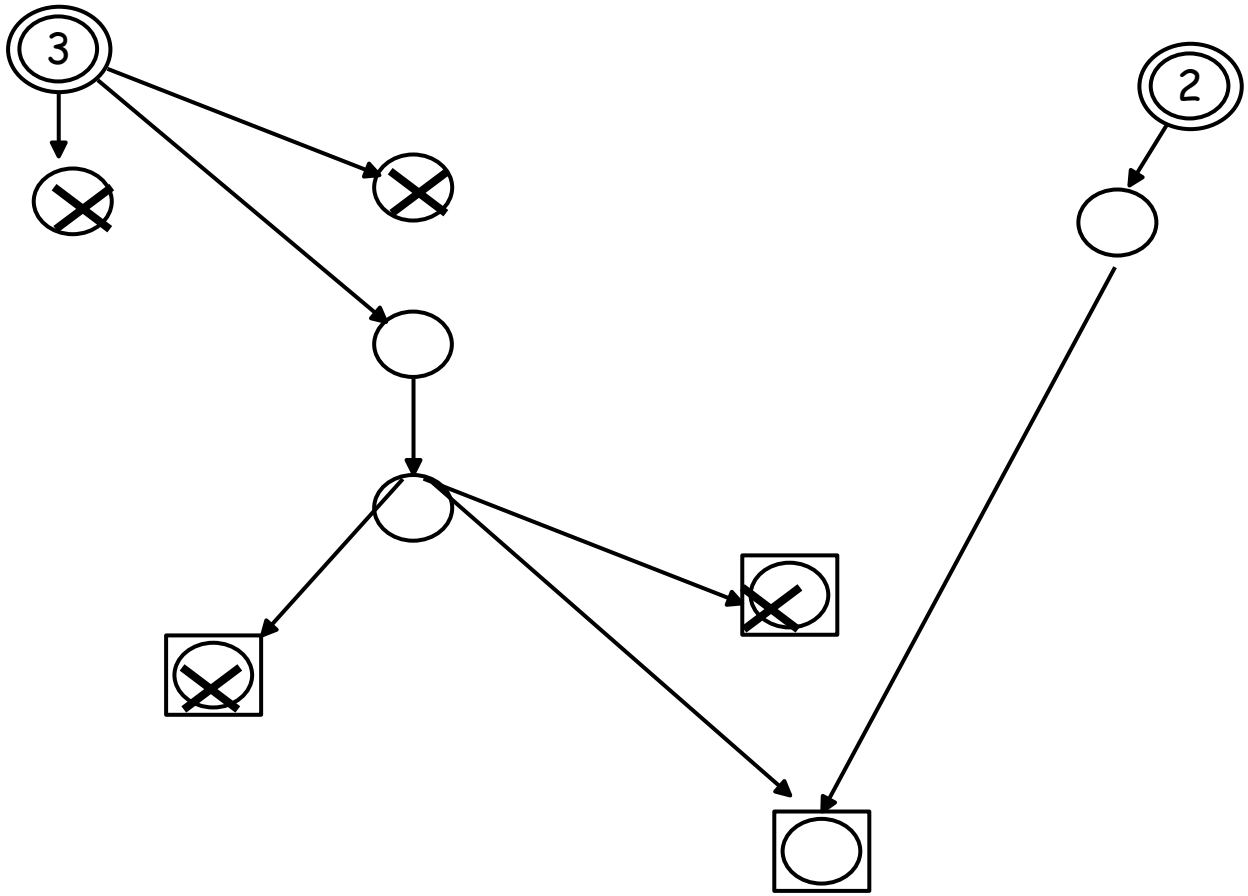


YO: send up

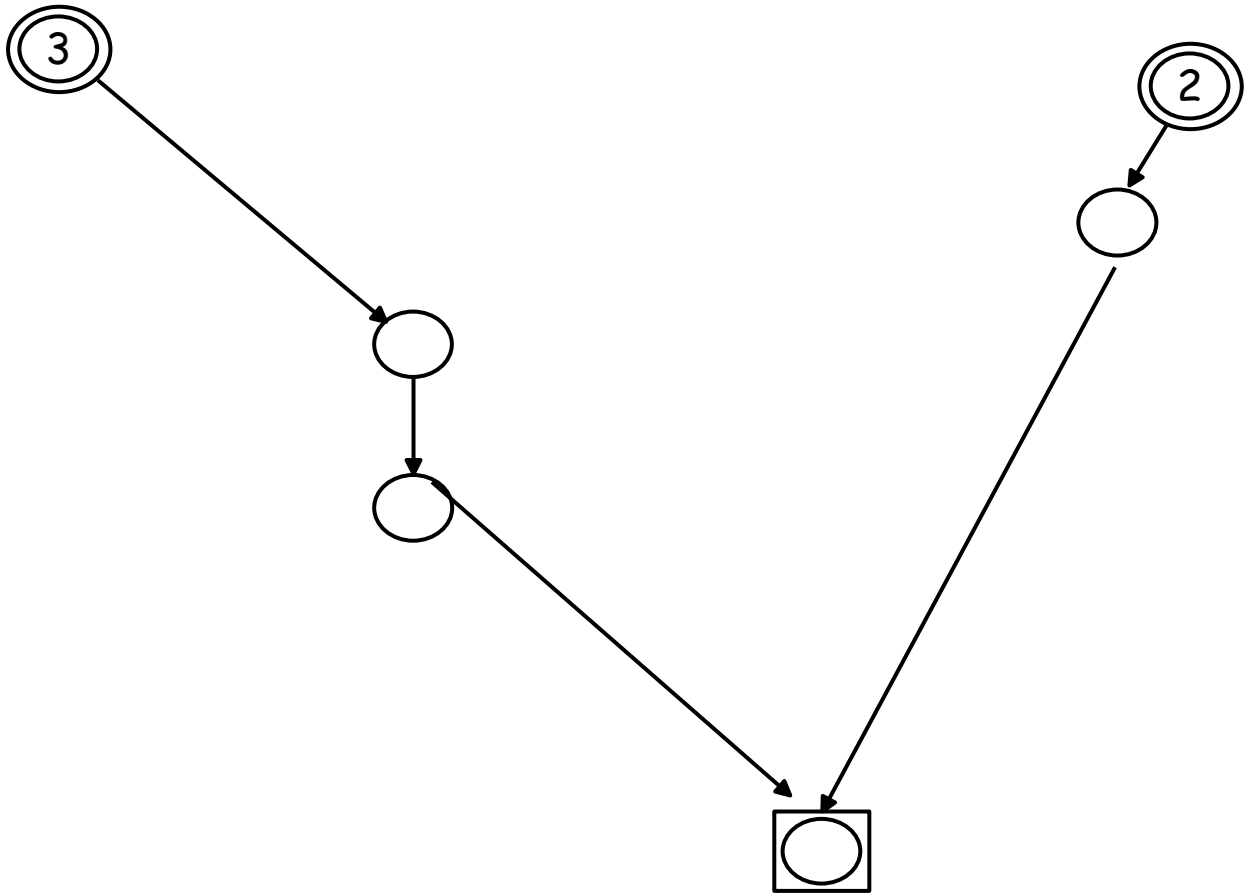


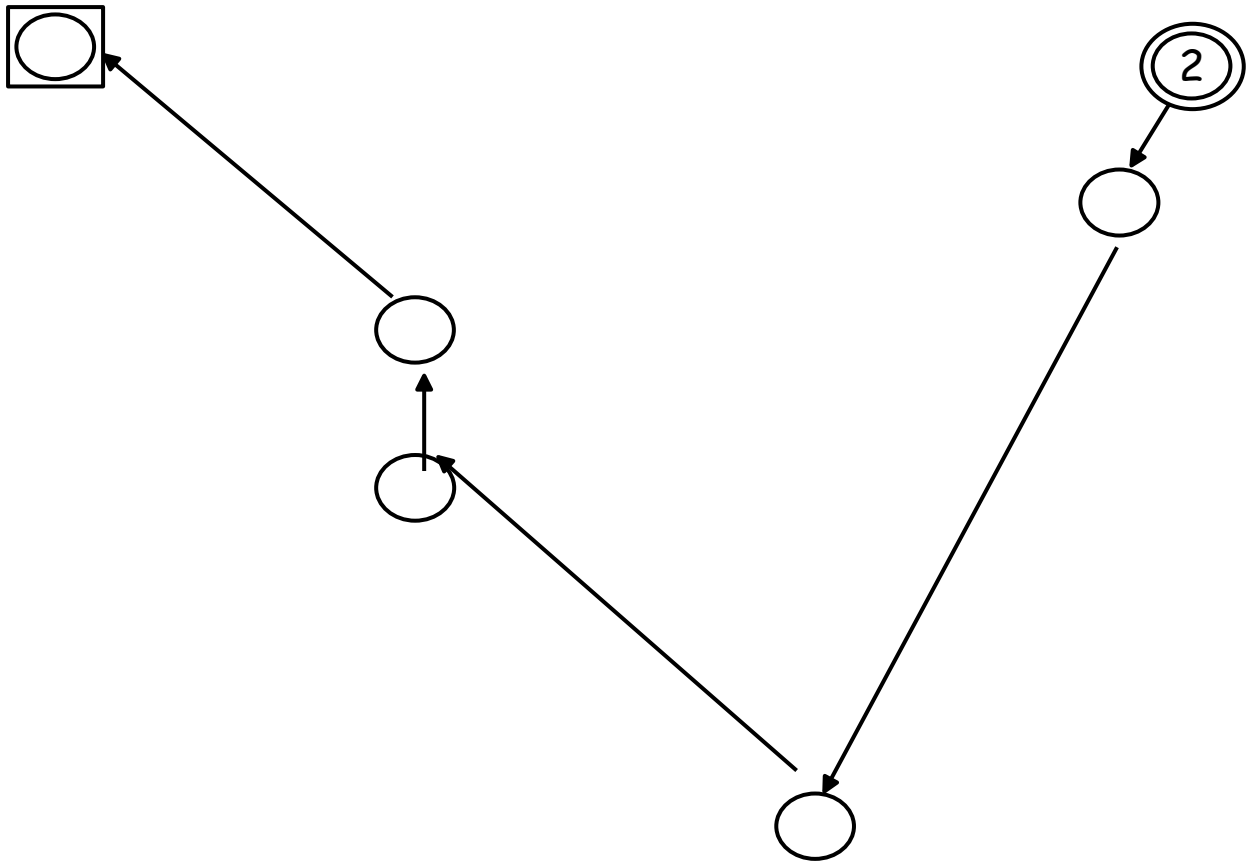
At the same time …. prune



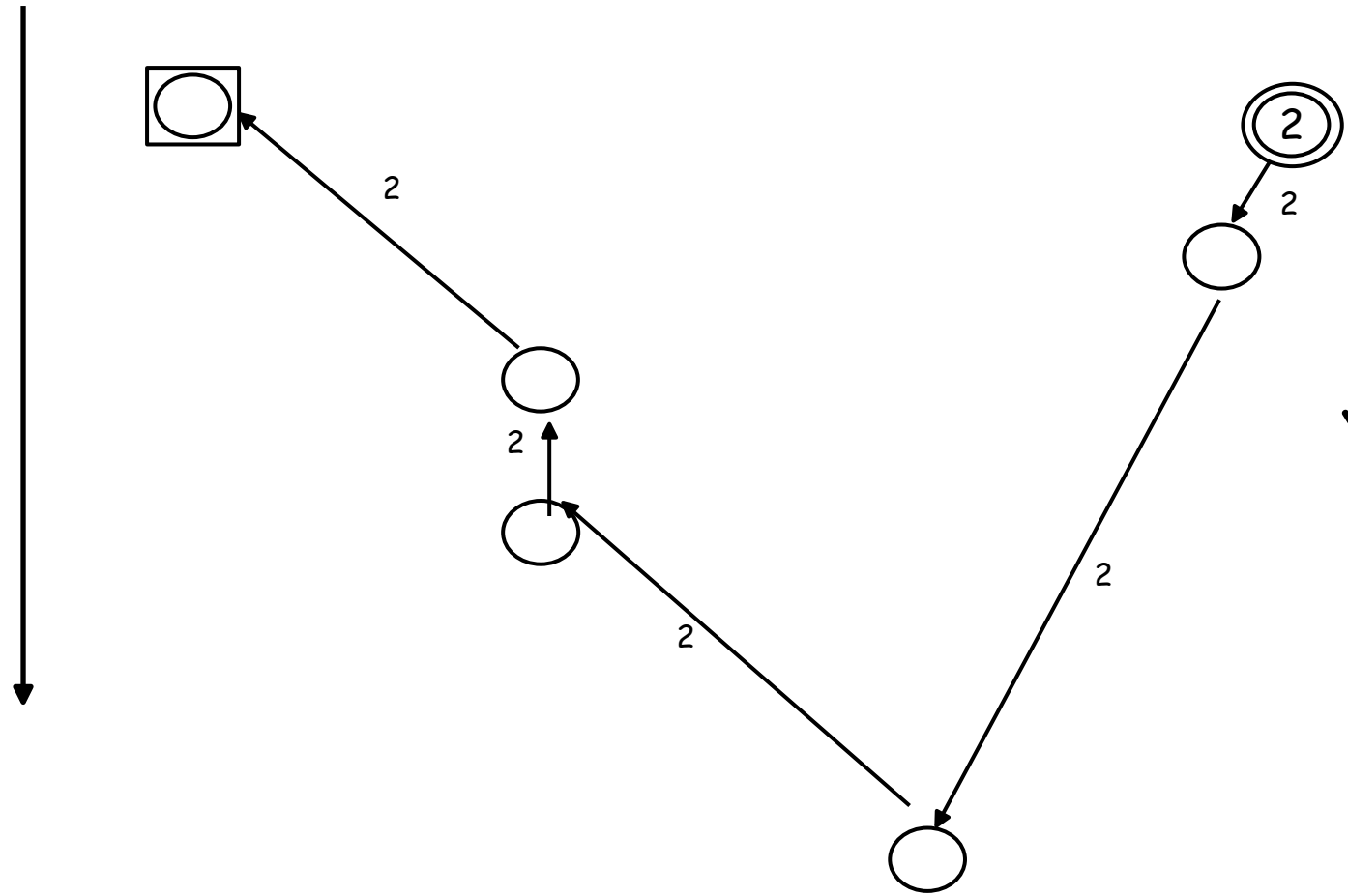


… More pruning

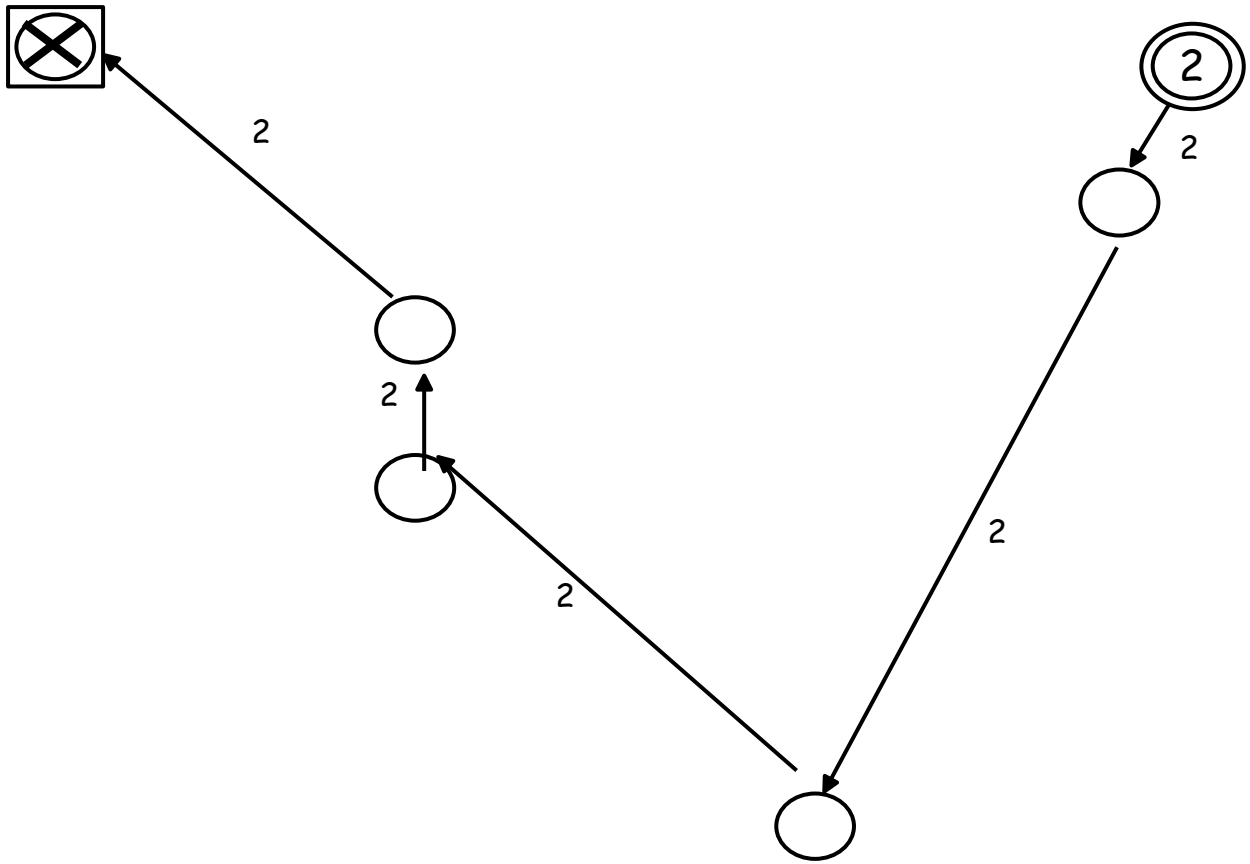

3
2



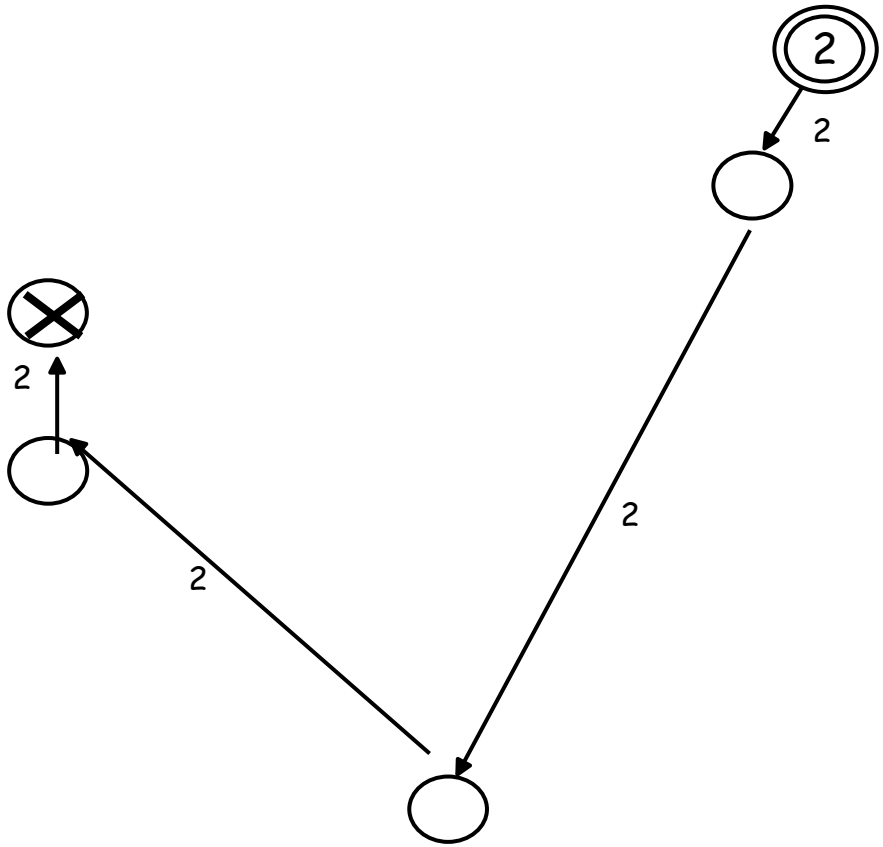
FLIP



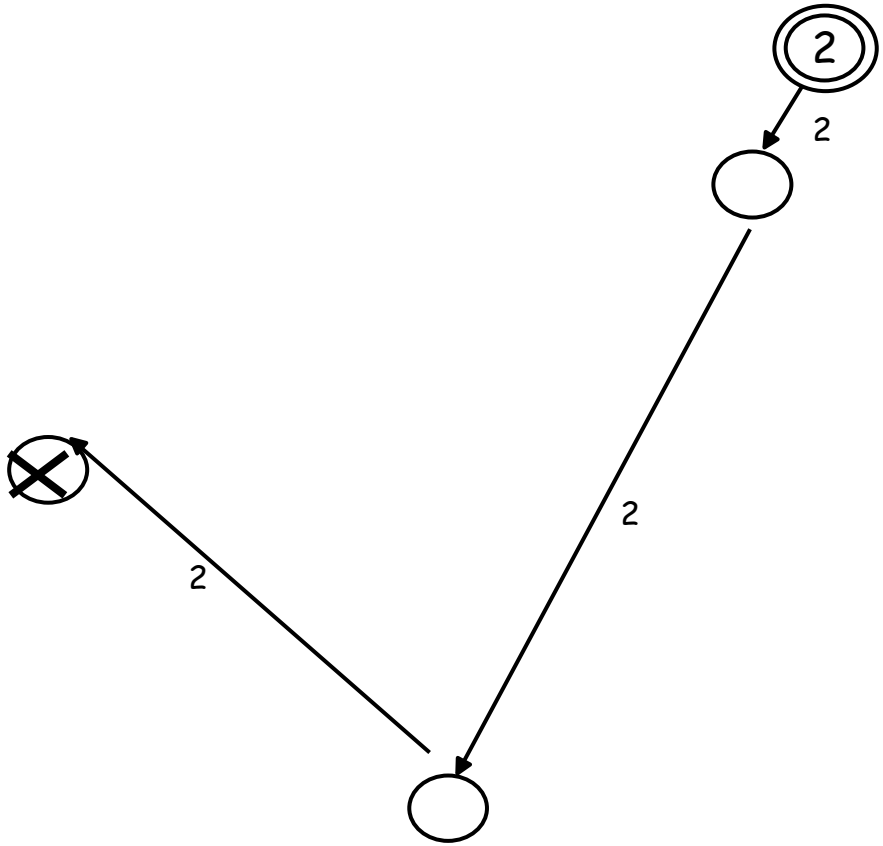
YO: send down



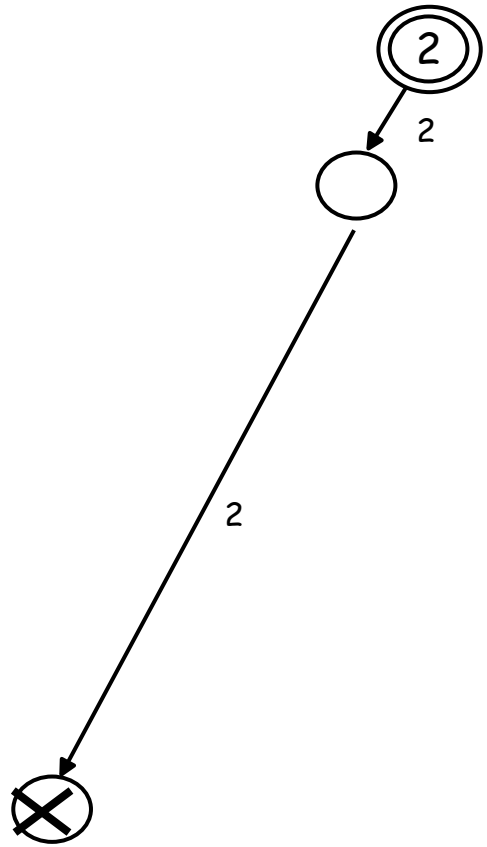
PRUNE



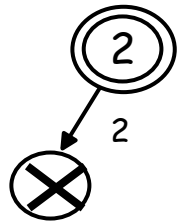
PRUNE



PRUNE



PRUNE



PRUNE

PRUNE

Complexity

Without Pruning

There are 2 messages for each link at each phase

- The number of phases is: log(# sources)

(s = # sources)

TOT: m + 2m log s

For initialization

With Pruning: ?