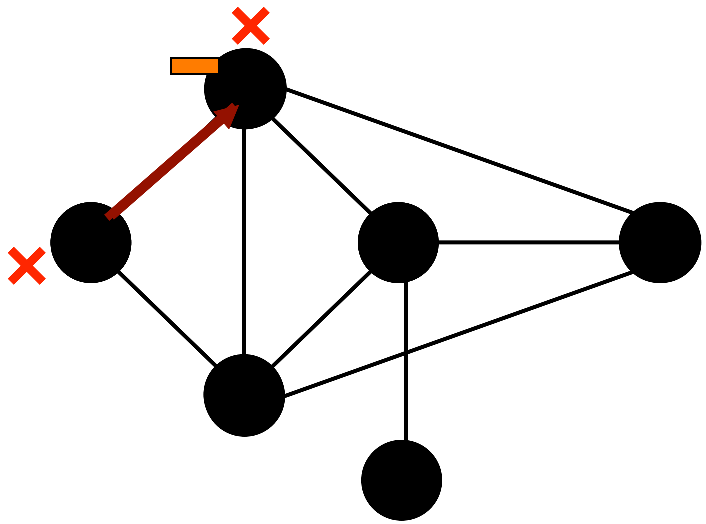
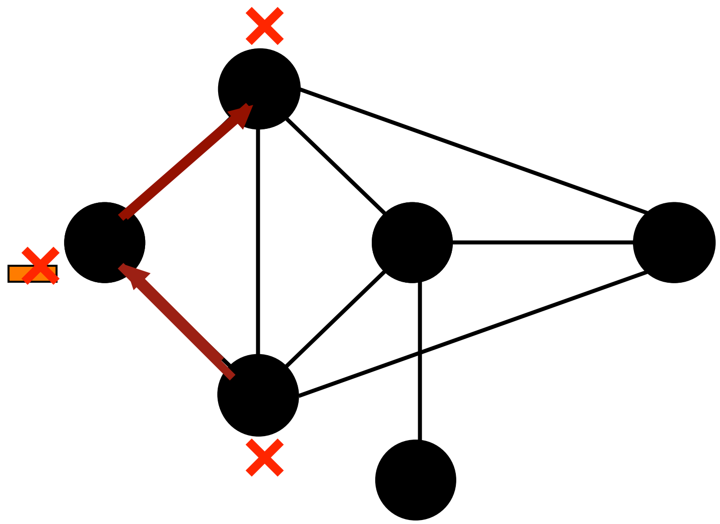
Traversal
Depth First Search

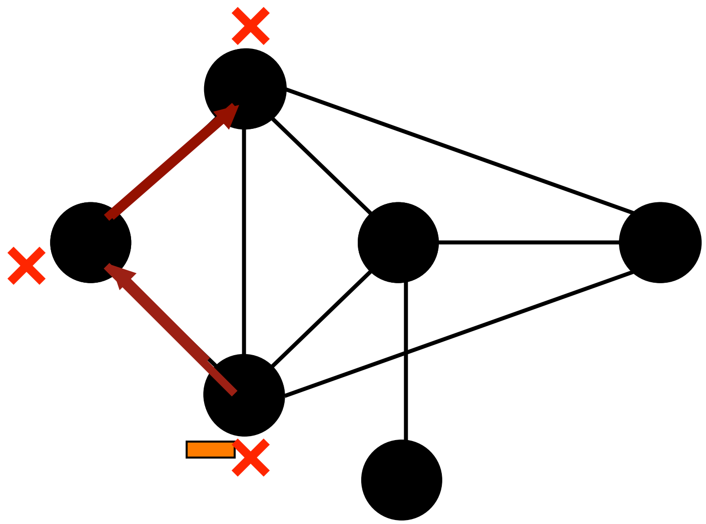
Assumptions

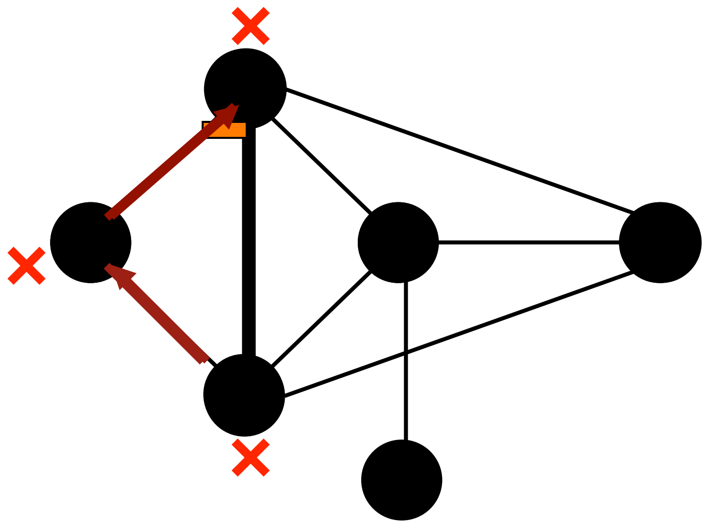
Single initiator
Bidirectional links
No faults
G connected

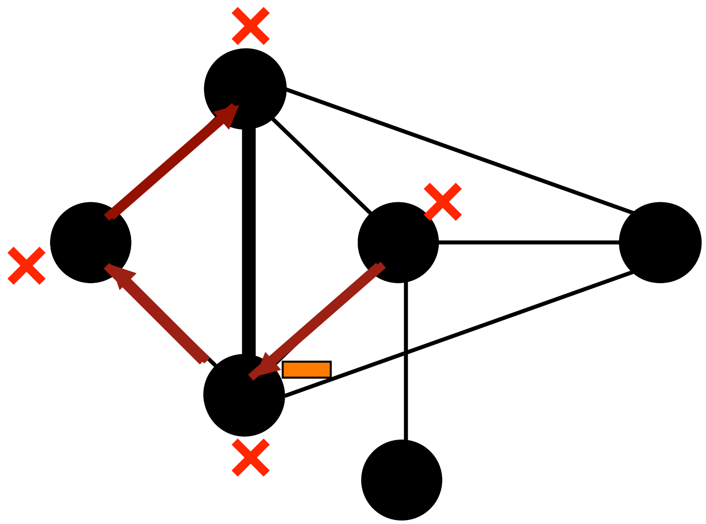
S = {INITIATOR, SLEEPING, ACTIVE, DONE}

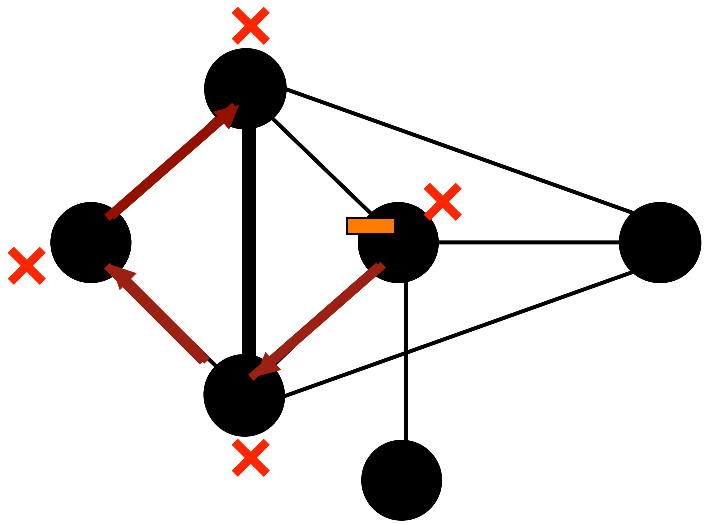


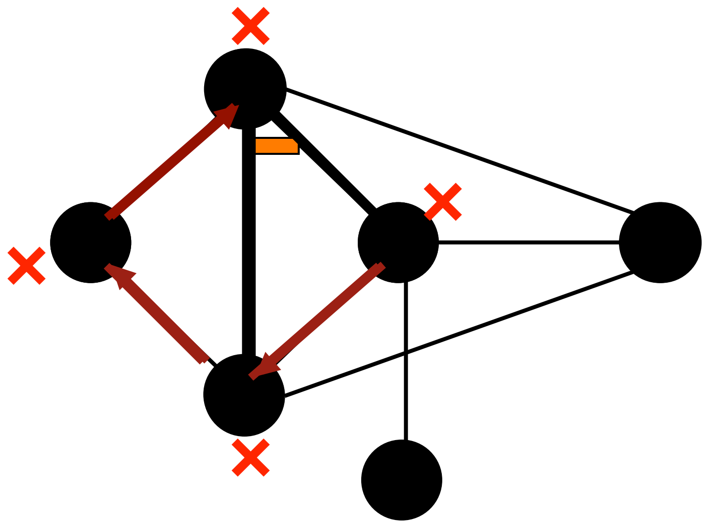


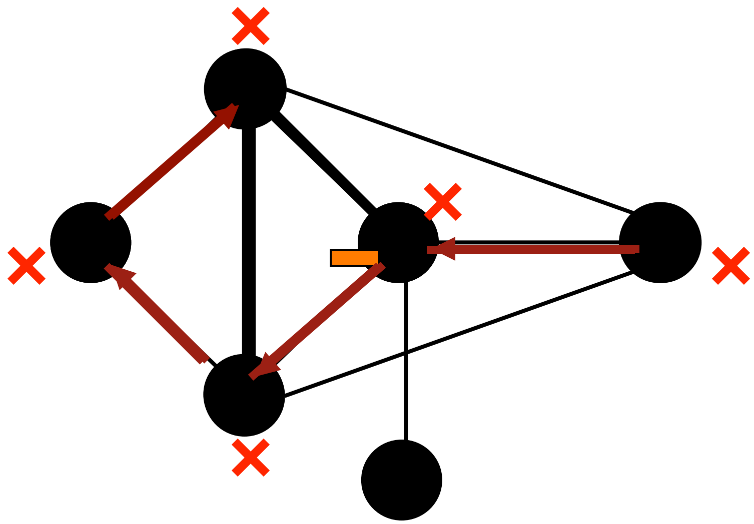


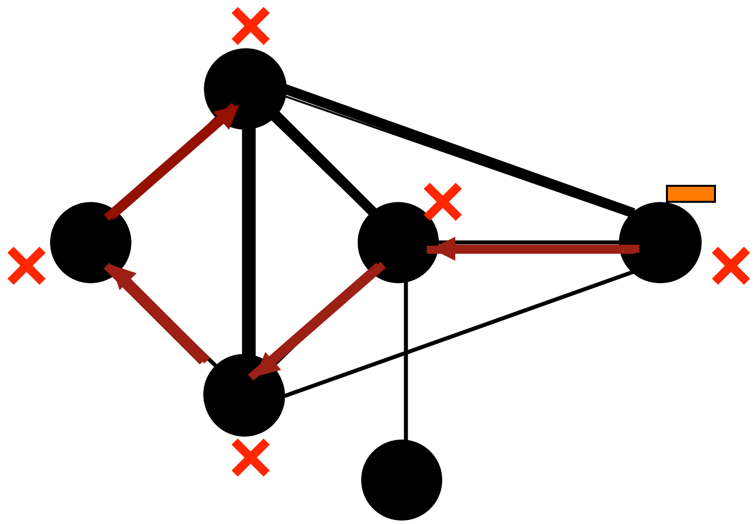


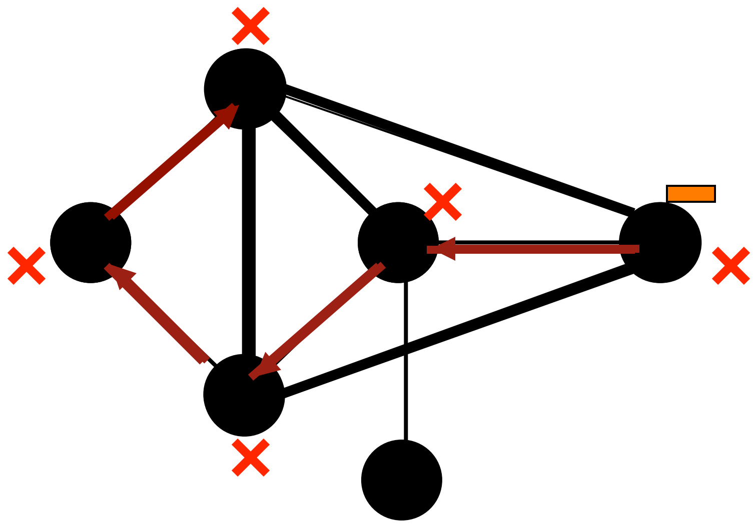


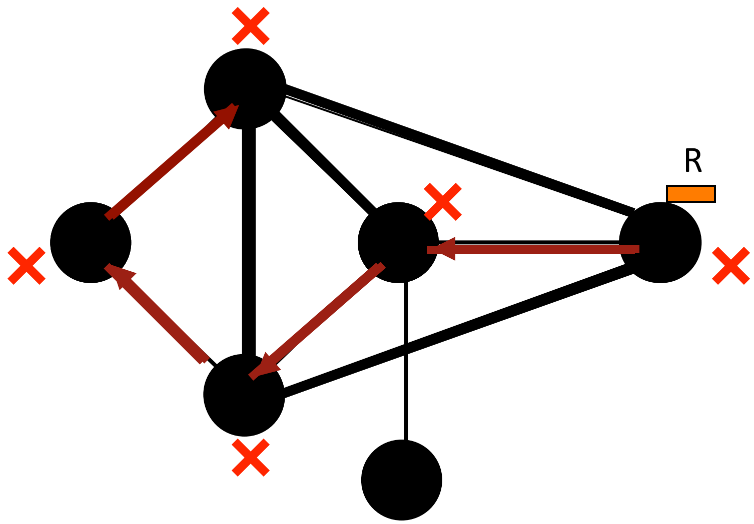


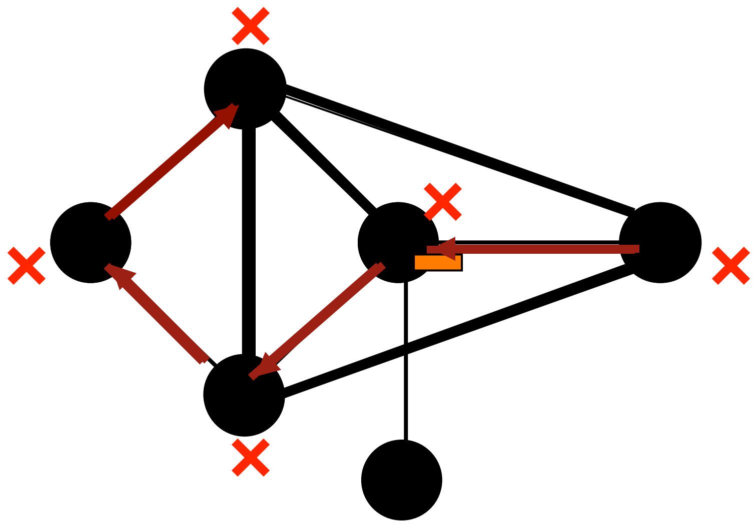


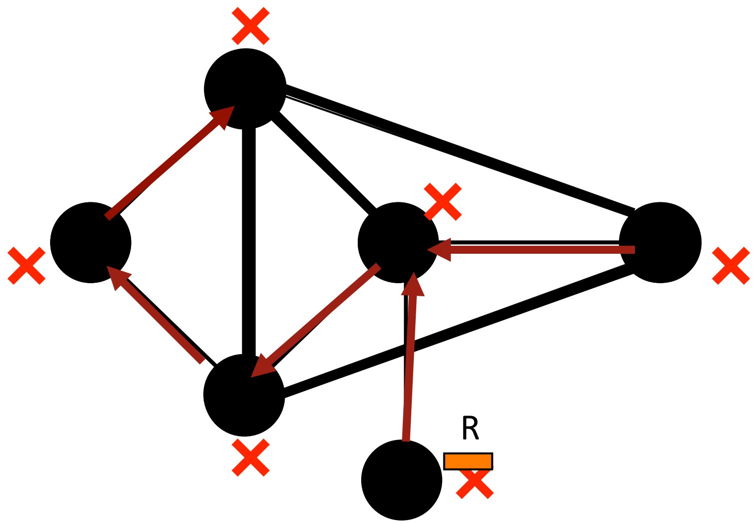


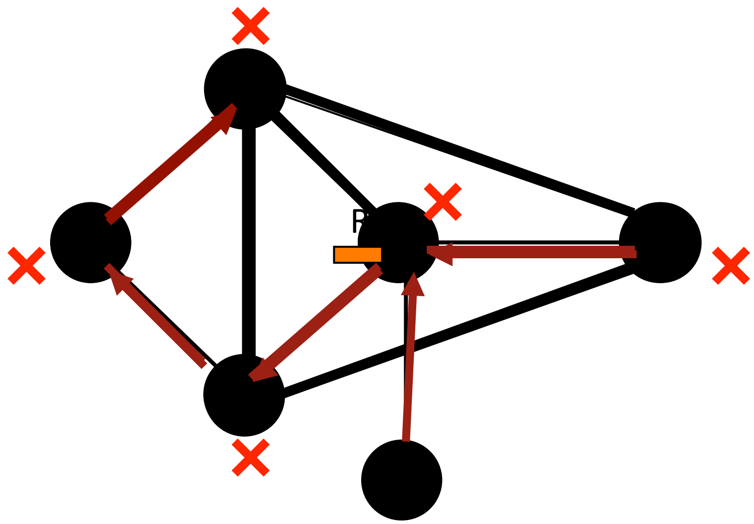





R

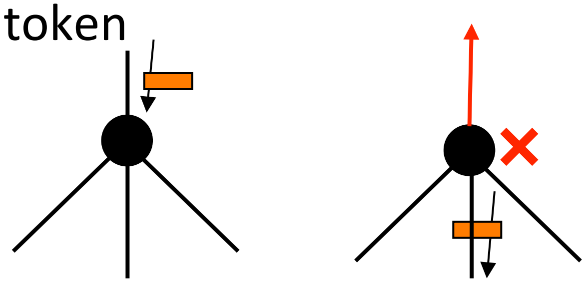



R

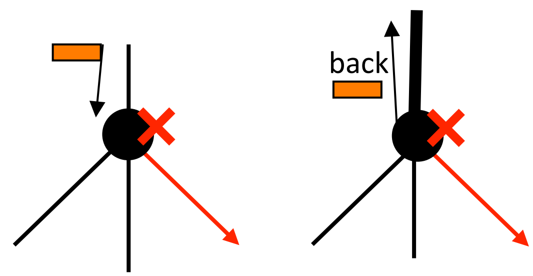


DF

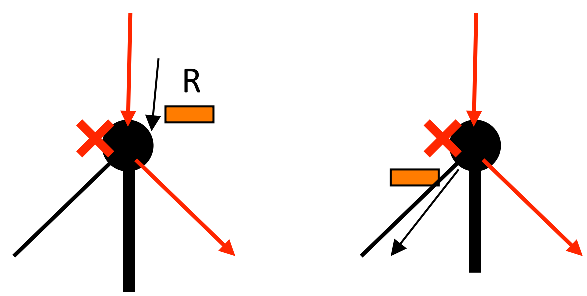
1) When receiving the token: *if it is the first time*, remember who sent (my parent), forward the token to one of the unvisited neighbours, become VISITED wait for the return token



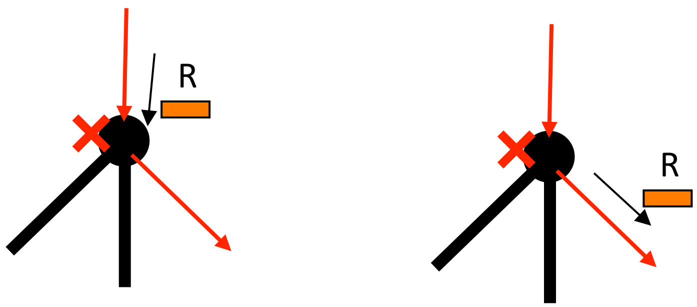
2) When receiving a token: *if already visited,* send the token back saying it is a back edge



3) When receiving a return token: send the token to an unvisited neighbour (if any)



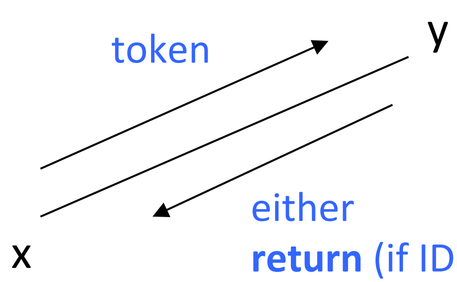
4) When receiving a return token: If there are *no more unvisited neighbours*, return the token to the parent



Complexity

Message Complexity:

Type of messages: `token, back, return`



either
return (if IDLE when received the token first)
oback

$2m = O(m)$

Time Complexity:
(ideal time)

$2m = O(m)$

Totally sequential

$\Omega(m)$ is also a lower bound

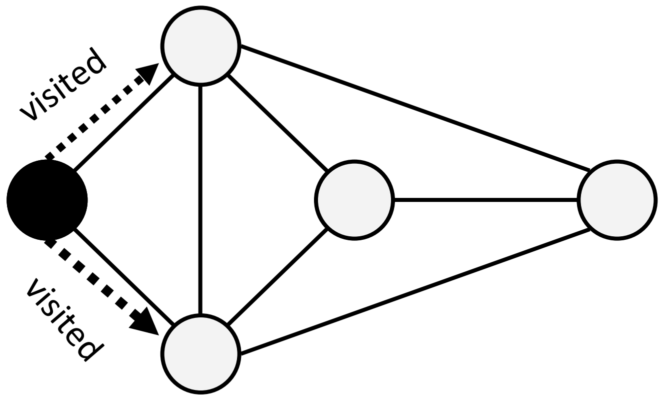
Note:
most messages are on Back Edges

---> most time is spent on Back Edges

Idea: avoid sending messages on back edges

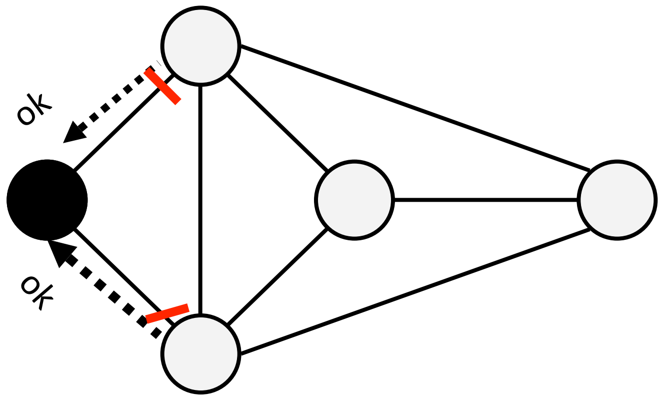
How ?

DF+ Improving Time



After being visited, I tell my neighbours so that they know that I already received the token !!!

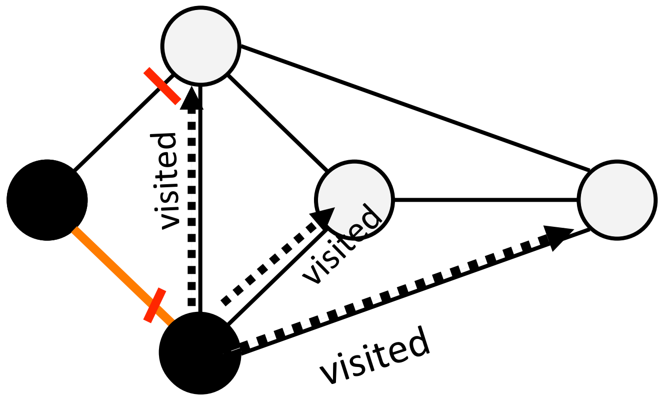
DF+ Improving Time



After being visited, I tell my neighbours so that they know that I already received the token !!!

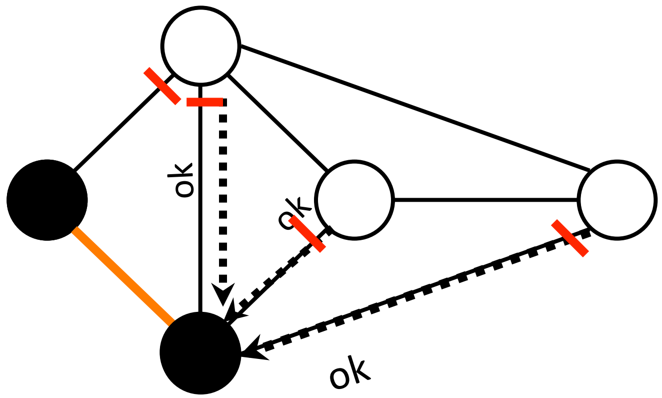
I wait for their reply to be sure that they know

DF+ Improving Time

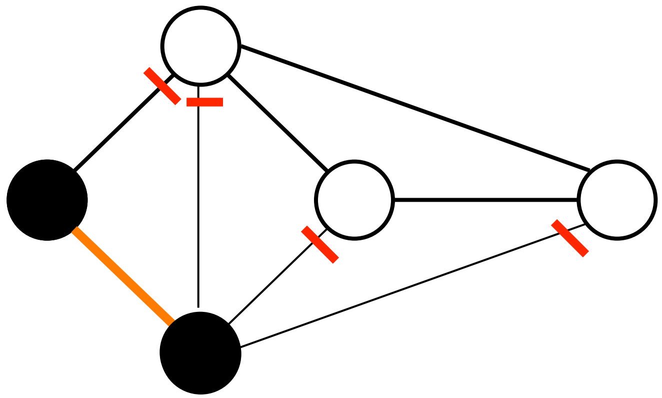


Then I send the token to an unvisited neighbour

DF+ Improving Time

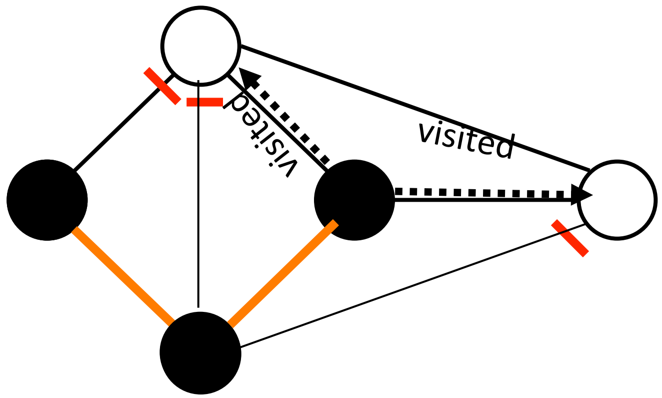


DF+ Improving Time



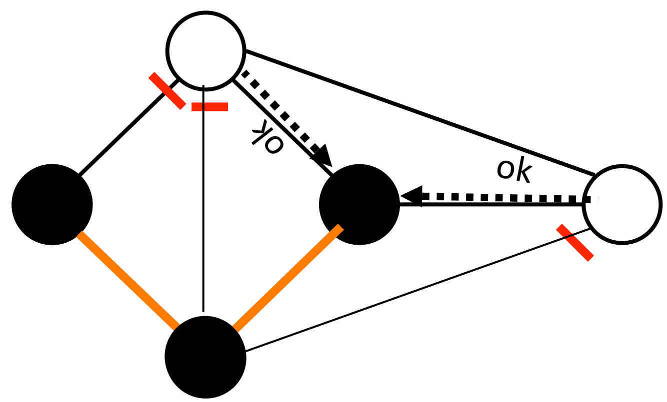
At any time anyone knows who, among its neighbours, has been already visited and who has not !!!!

DF+ Improving Time



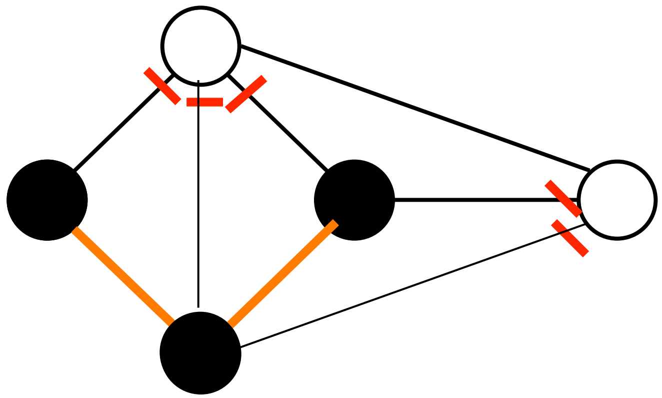
At any time anyone knows who, among its neighbours, has been already visited and who has not !!!!

DF+ Improving Time



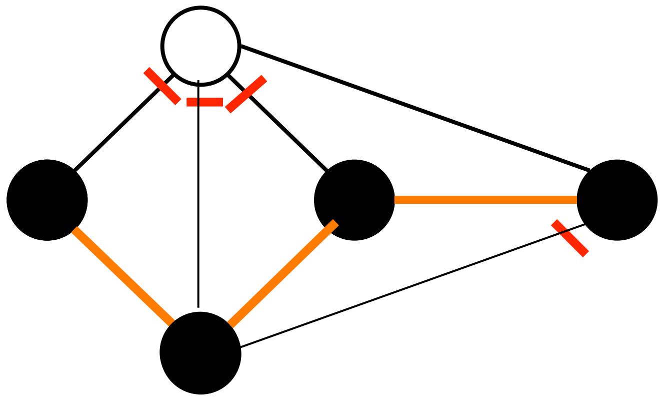
At any time anyone knows who, among its neighbours, has been already visited and who has not !!!!

DF+ Improving Time



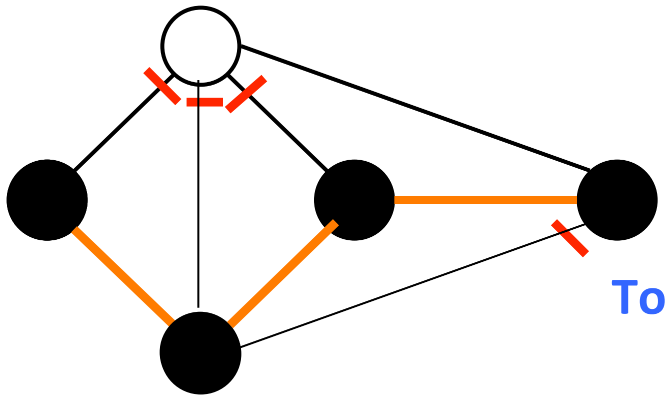
At any time anyone knows who, among its neighbours, has been already visited and who has not !!!!

DF+ Improving Time



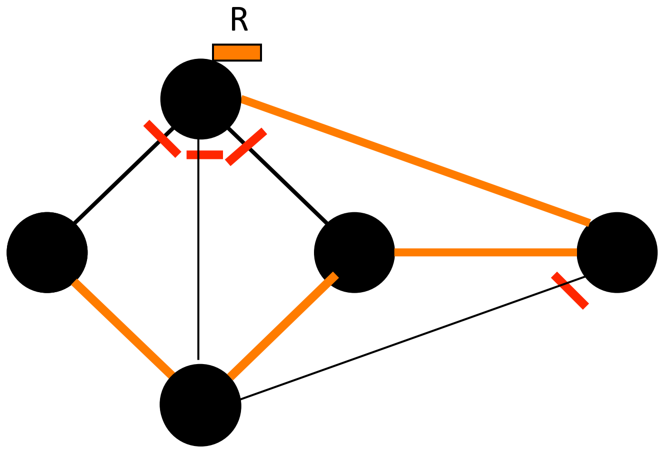
At any time anyone knows who, among its neighbours,
has been already visited and who has not !!!!

DF+ Improving Time



I know where
to send the token
To find an unvisited neighbour

DF+ Improving Time



DF+ Complexity

Message

Messages: **`Token`**, **`Return`**, **`Visited`**, **`Ack (ok)`**

Each entity (except init): receives 1 **`Token`**, sends 1 **`Return`**:

2(n-1)

Each entity:
1 **`Visited`** to all neighbours except the sender

Let s be
the initiator

$$|N(s)| + \sum_{x \neq s} (|N(x)|-1)$$

$$= 2m - (n-1)$$

(same for **`Ack`**)

TOT: **4m**

DF+ Complexity

Time (ideal time)

Token and Return are sent sequentially: 2(n-1)

Visited and Ack are done in parallel: 2n

TOT: **4n -2**

Summarizing:

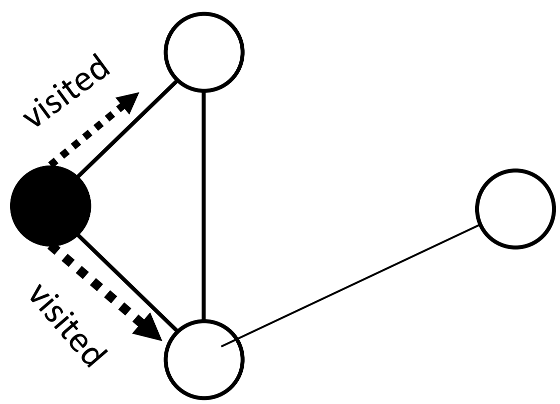
DF Traversal

	Messages	Ideal Time
DF:	2m	2m
DF+:	4m	4n -2

DF++

Do not send the `Ack`

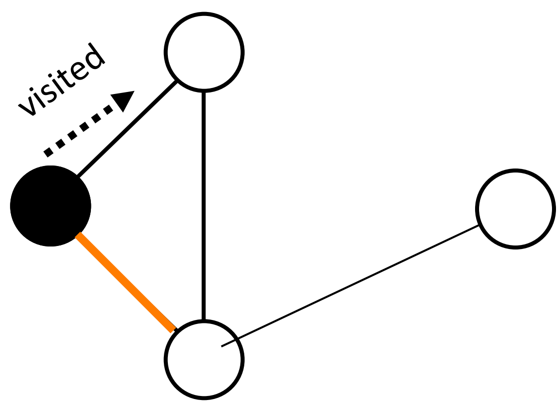
What happens ?



DF++

Do not send the `Ack`

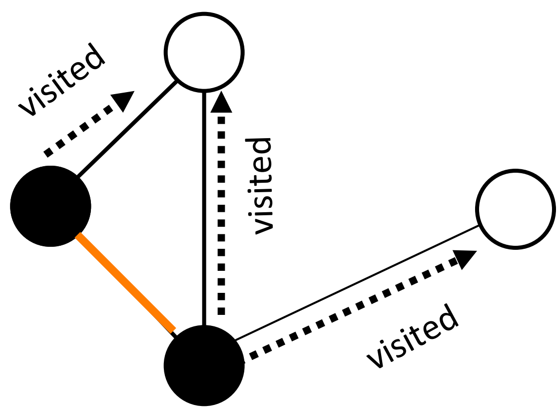
What happens ?



DF++

Do not send the `Ack`

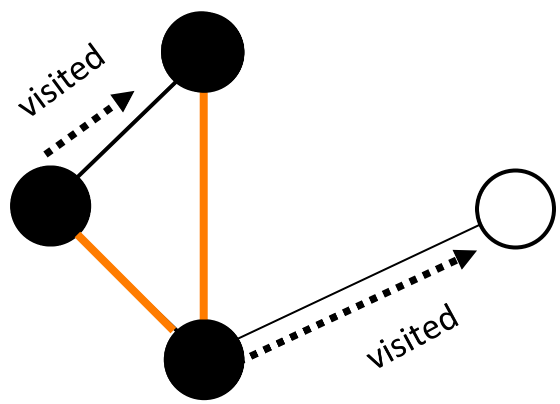
What happens ?



DF++

Do not send the `Ack`

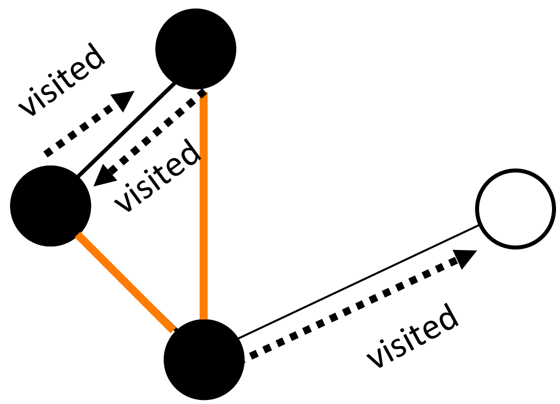
What happens ?



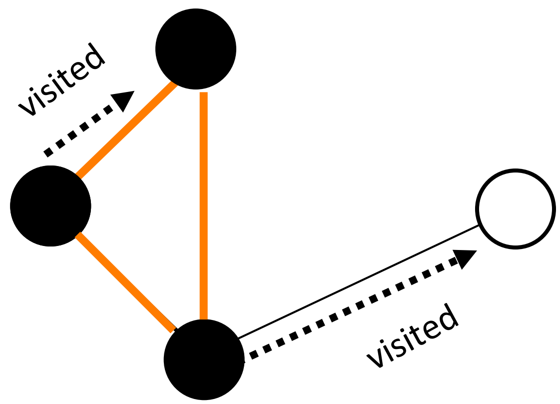
DF++

Do not send the `Ack`

What happens ?



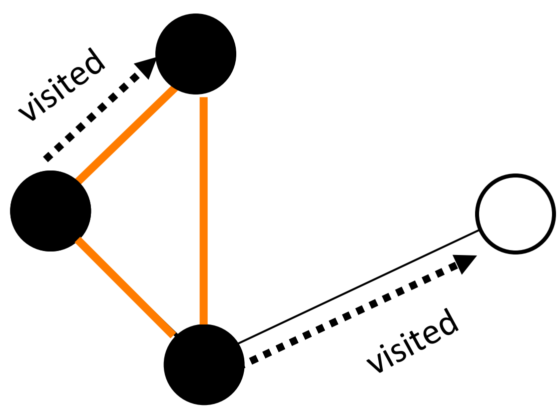
Do not send the `Ack`
What happens ?



Mistake: A token is sent to an already visited node (= back edge)

Do not send the `Ack`

What happens ?

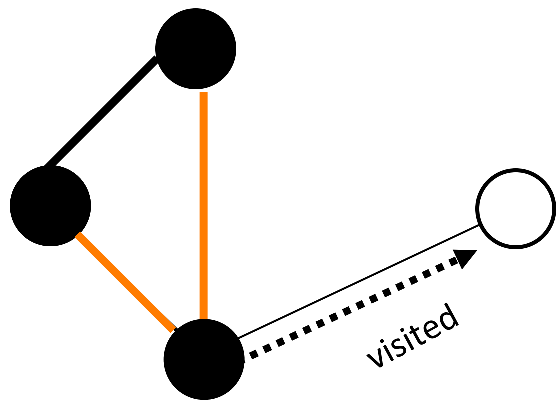


Mistake: A token is sent to an already visited node (= back edge)

Both nodes will eventually understand the "mistake"

Do not send the `Ack`

What happens ?



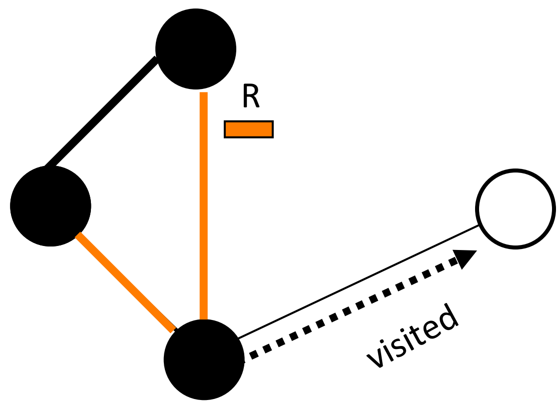
Mistake: A token is sent to an already visited node (= back edge)

Both nodes will eventually understand the "mistake"

and pretend nothing happened

Do not send the `Ack`

What happens ?



Both nodes will eventually understand the "mistake"

pretend nothing happened

and continue with the algorithm ....

DF++ Complexity Message

Messages: **`Token,`** **`Return,`** **`Visited,`**

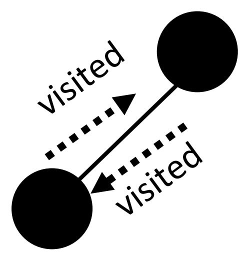
Each entity (except init): receives 1 **`Token`**, sends 1 **`Return`**:

2(n-1)

Each entity:
1 **`Visited`** to all neighbours except the sender

2m - (n-1)

In the worst case there are two "mistake-token"
on each link except for the tree links **2(m-n+1)**



TOT: ≤**4m –n +1**

DF++ Complexity Time

BUT when we measure ideal time:

“mistakes” will not happen

Time = 2(n-1)

Summary

	Messages	Ideal Time
DF:	2m	2m
DF+:	4m	4n -2
DF++	4m-n+1	2n-1

Observations

Time ...

Termination ...

An application:
access permission problems, e.g., Mutual Exclusion

Any Traversal does a Broadcast (not very efficient)
The reverse is not true.

Another Traversal: Smart Traversal

1- Build a Spanning Tree with SHOUT+

Messages = 2m

2- Perform DF Traversal

Messages = 2(n-1)

Total Messages = **2(m+n-1)**

Another Traversal: Smart Traversal

1- Build a Spanning Tree with SHOUT+

Time ≤ d+1 d: diameter

2- Perform DF Traversal

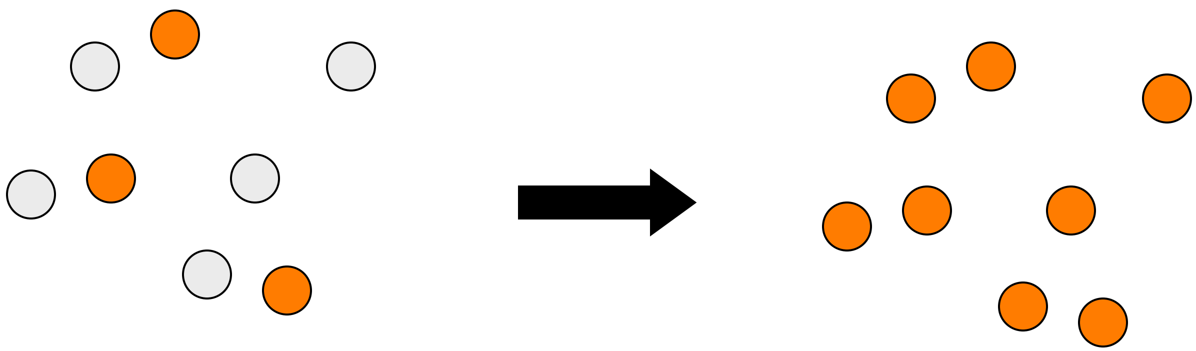
Time = 2(n-1)

Total Time **≤ 2n+d-1**

Summary

	Messages	Ideal Time
DF:	2m	2m
DF+:	4m	4n -2
DF++	4m-n+1	2n-1
Smart	2m+2n-2	2n+d-1

Computations with Multiple initiator: WAKE-UP



FLOOD solves the problem.

General FLOOD algorithm: O(m)

More precisely: 2m -n + k* WHY ?

↘ n. of initiators

1 init = broadcast = 2m -n+1

All init = 2m

Computations with Multiple initiator: WAKE-UP

In special topologies ?

TREE

Flood is optimal

$n + k^* - 2$

Computations with Multiple initiator: WAKE-UP

COMPLETE GRAPH

Broadcast

Flood $O(n^2)$ → Specific $O(n)$

Wakeup

Flood Specific

$\Omega(n^2)$

Need additional assumptions to reduce the complexity

HYPERCUBE

Broadcast

Flood $O(n \log n)$ → Specific $O(n)$

Wakeup

Flood Specific

$\Omega(n \log n)$