The Alexander Kofkin
Faculty of Engineering
Bar-Ilan University

Distributed Computing with Signals

or,
Signal, if you can't (for the damaged)

Ran Gelles, Bar-Ilan University, Israel

ADGA'24, Madrid

What is signaling?

- When messages themselves carry no information — they **are** the information
- Information coded by the **time**, **order** of arrival

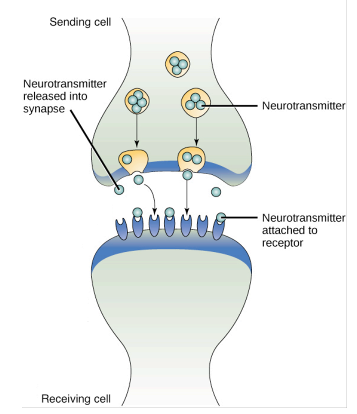
Signaling

Simple Communication

Fault Tolerance

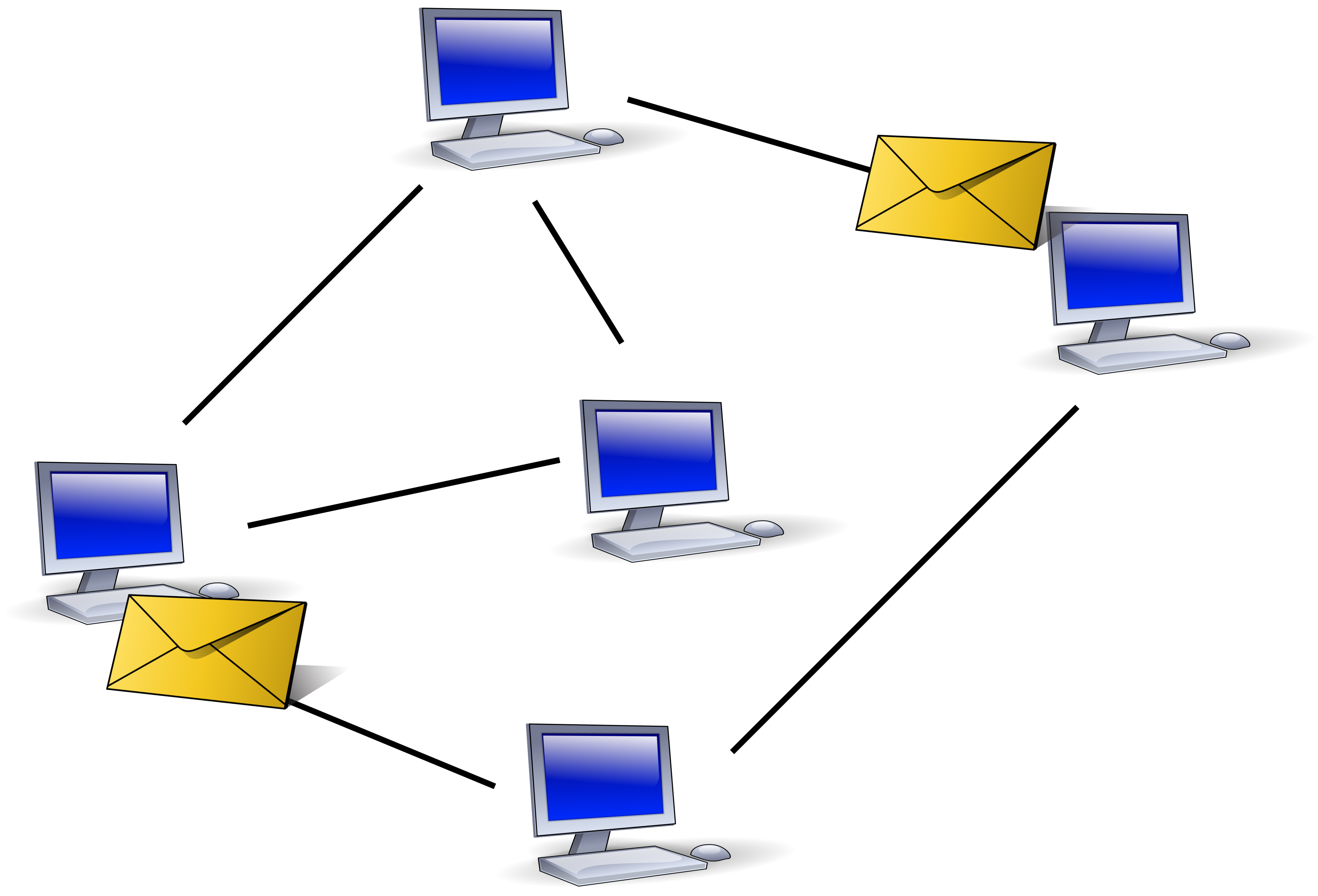
Bio-systems, Sensors, Beeping (wireless)

“Fully Defective Networks”



Content-Oblivious Computation

Synchronous Signaling



- Santoro and Widmayer 1989 , 1990:
  - In **synchronous** systems, messages arrival time carry information

Round	1	2	3	4
info	0	1	1	0

Time is not a healer

- In **Asynchronous** systems, messages suffer arbitrary delays
- Time cannot be used, and we need to employ other properties

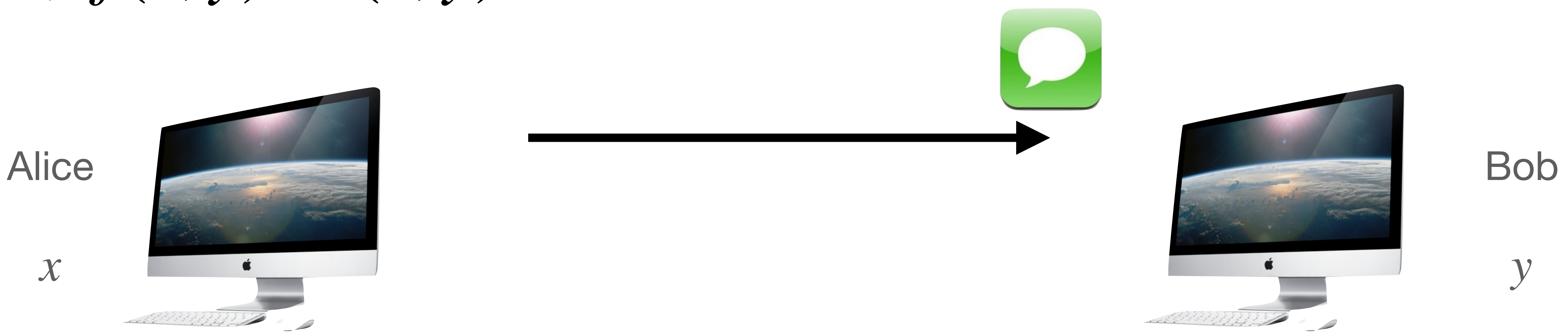
- Is *asynchronous computation* by signaling (content-oblivious) even possible?

Theorem [Santoro-Widmayer 89]**:**
No k-agreement is possible in synchronous system (over K_n) with
(i) n/2 messages corrupted per round, *if nodes always transmit*
or (ii) n-1 messages corrupted per round, *or inserted*

Impossibility of content-oblivious comp. over bridge

Theorem [CensorHillel-Cohen-Gelles-Sela 23]**:**
When nodes must terminate / finalize output and G contains a bridge,
some computation cannot be deterministically simulated over G.

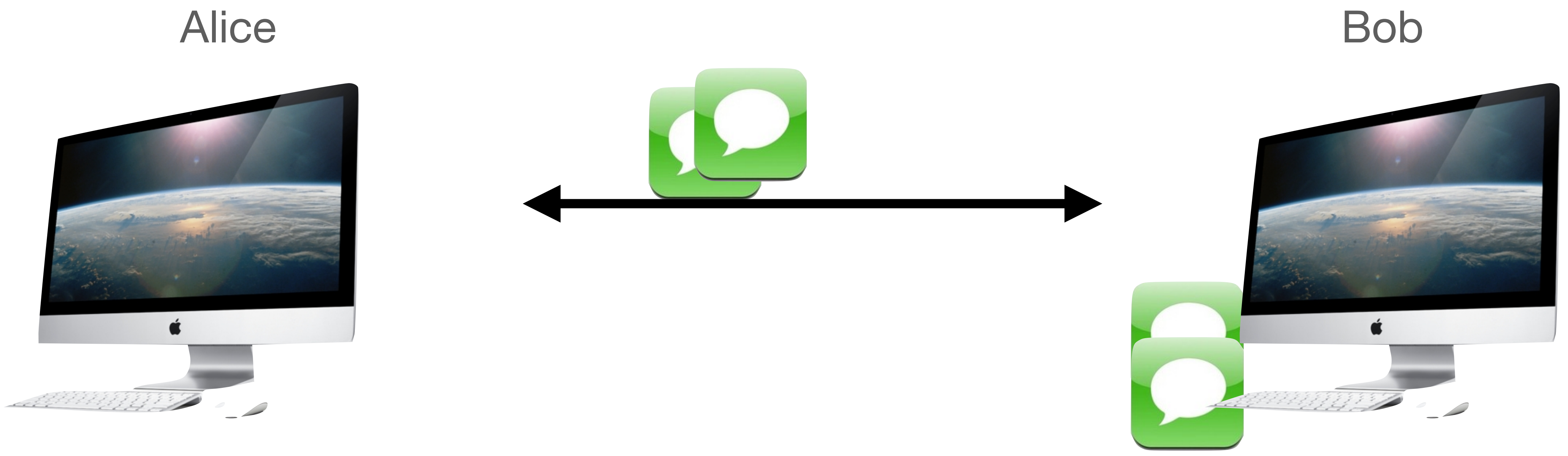
- Consider 2 parties, $f(x, y) = (x, y)$



- Bob's actions depend on the count of received messages
  - "**Upon** receiving the k-th message **do**: (send …, output …)"
- Bob's actions are the same, regardless of Alice's input
  - if on x_1 Bob outputs (x_1, y) after receiving k messages, then on x_2 he will either output (x_1, y), or never receive k messages == no output.

Impossibility of content-oblivious comp. over bridge

- note, if no termination is required, any computation is possible:



- Bob on input $y \in \mathbb{Z}$:
  - Send y messages to Alice
  - Upon receiving the k-th message from Alice, update output to $f(k, y)$

So… What can be done?

Let’s relax the model and

Assume a Leader

Content Oblivious BFS [Censor-Hillel, Gelles, Haeupler 19]

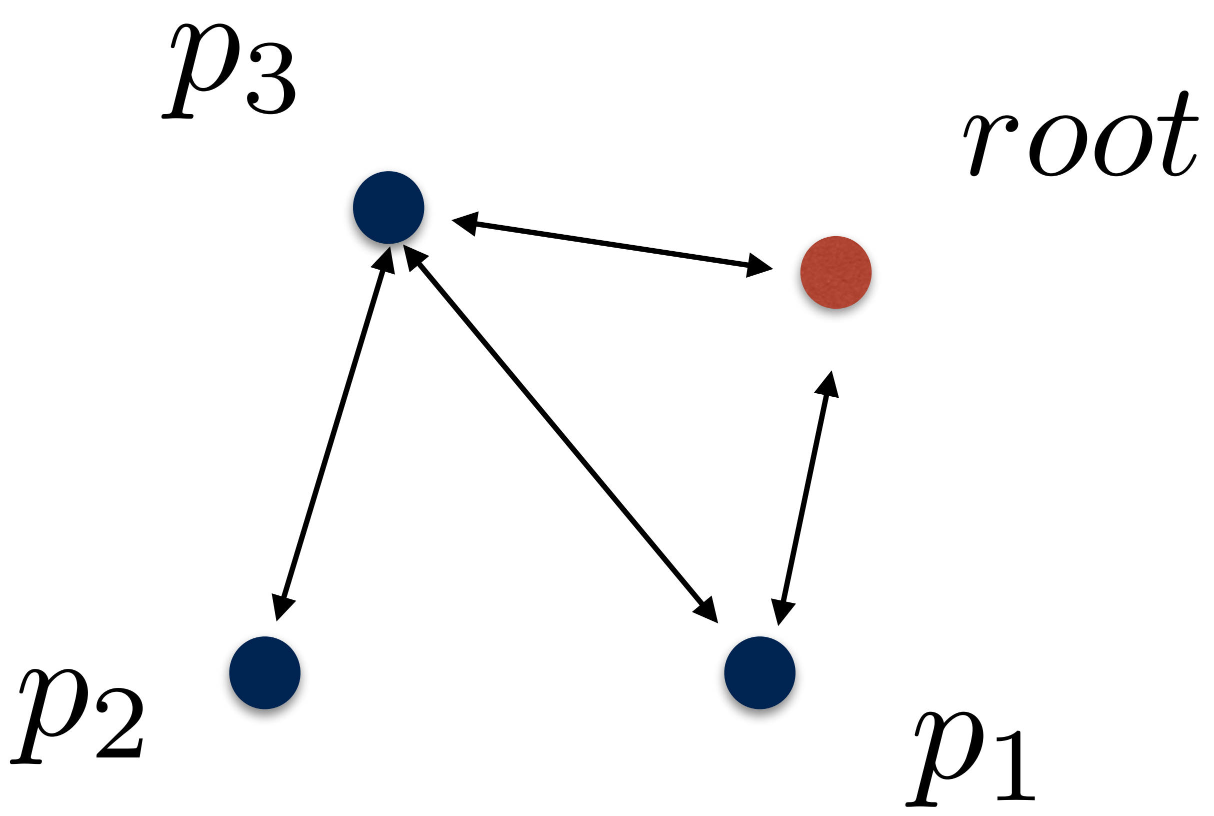
- **Reminder**: Distributed Dijkstra
  the algorithm works in "phases", initiated by the leader (root)

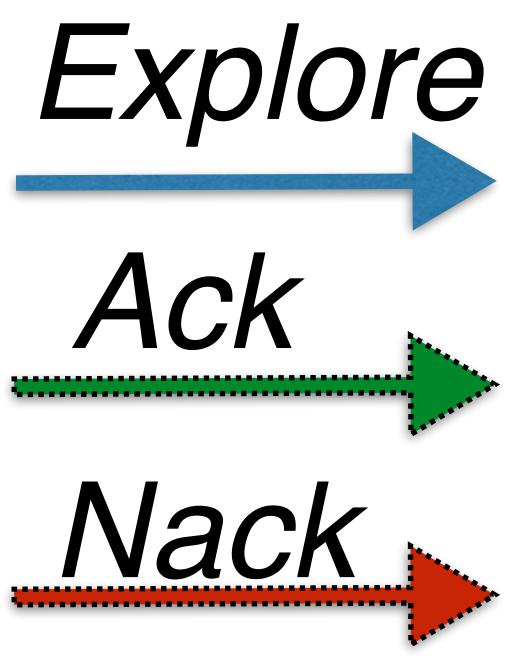
  - "***Explore***":
    send message to all neighbours (excl. parent)
    once all neighbours ***Ack***, send ***Ack*** to parent

  - upon receiving "***Explore***":
    if first time - set sender as parent**.** Reply with ***Ack***
    otherwise -
       if from parent: perform **Explore**
       if not parent: return ***Nack*** to sender

Content Oblivious BFS

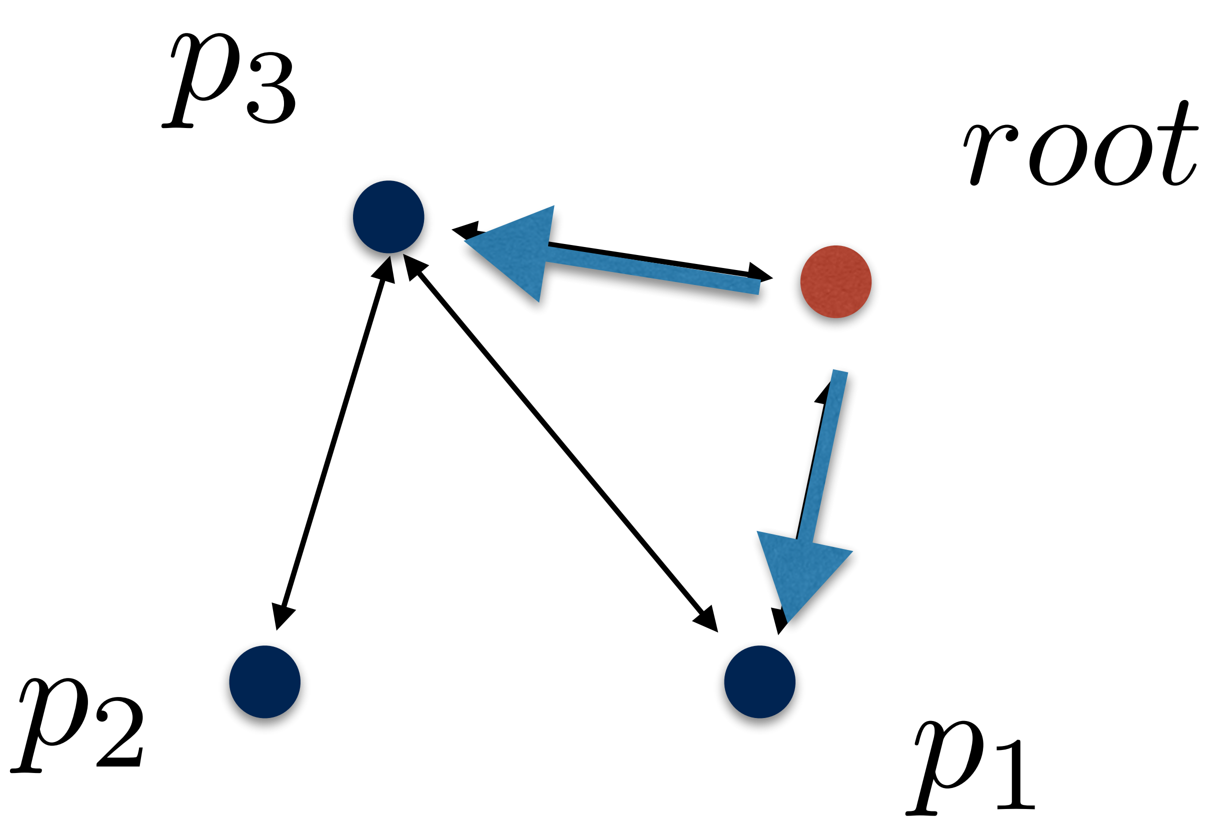
(Example)

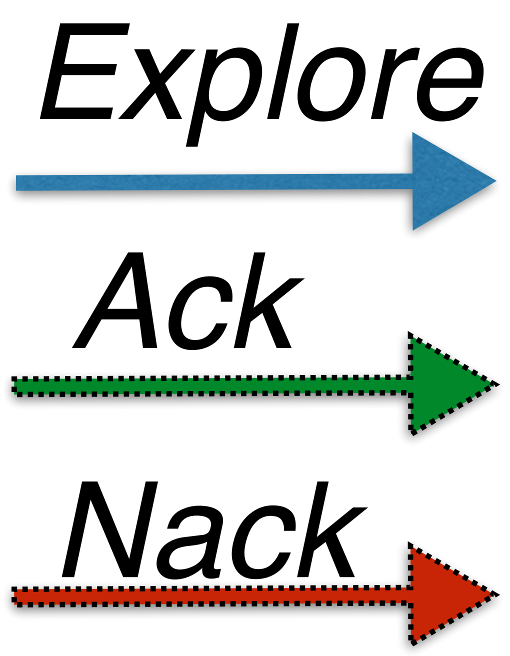




Content Oblivious BFS

(Example)





Phase 1

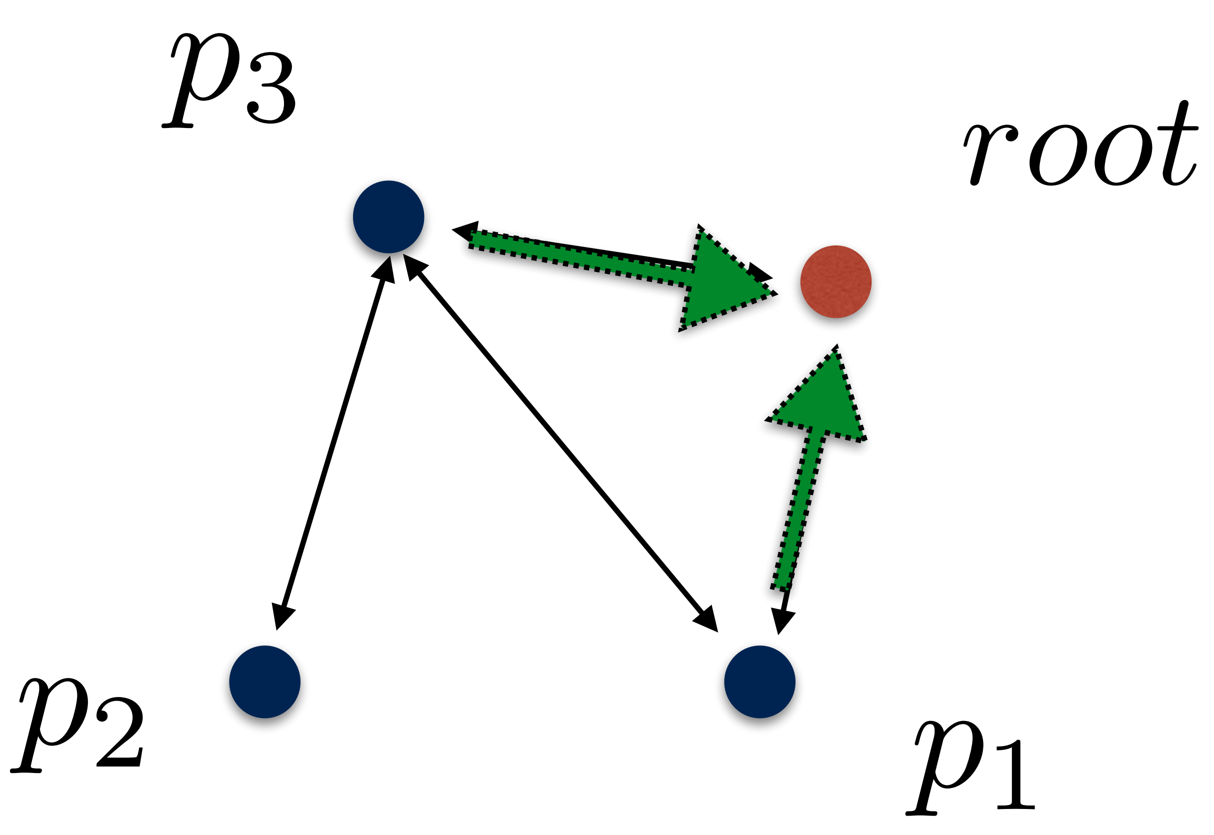
parent(p_1) = ?

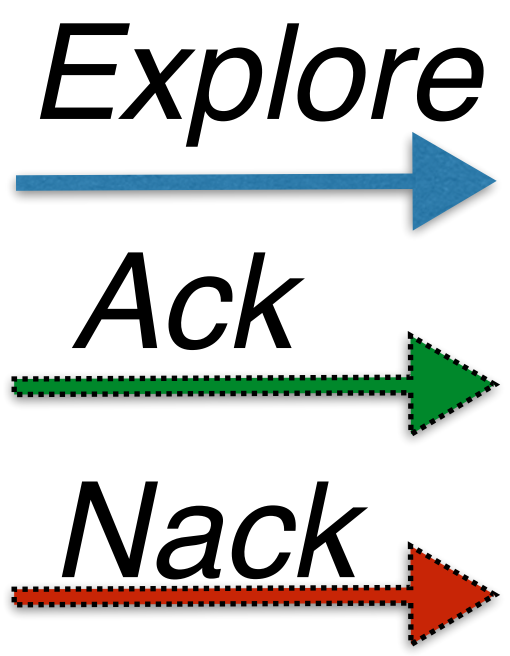
parent(p_2) = ?

parent(p_3) = ?

Content Oblivious BFS

(Example)





Phase 1

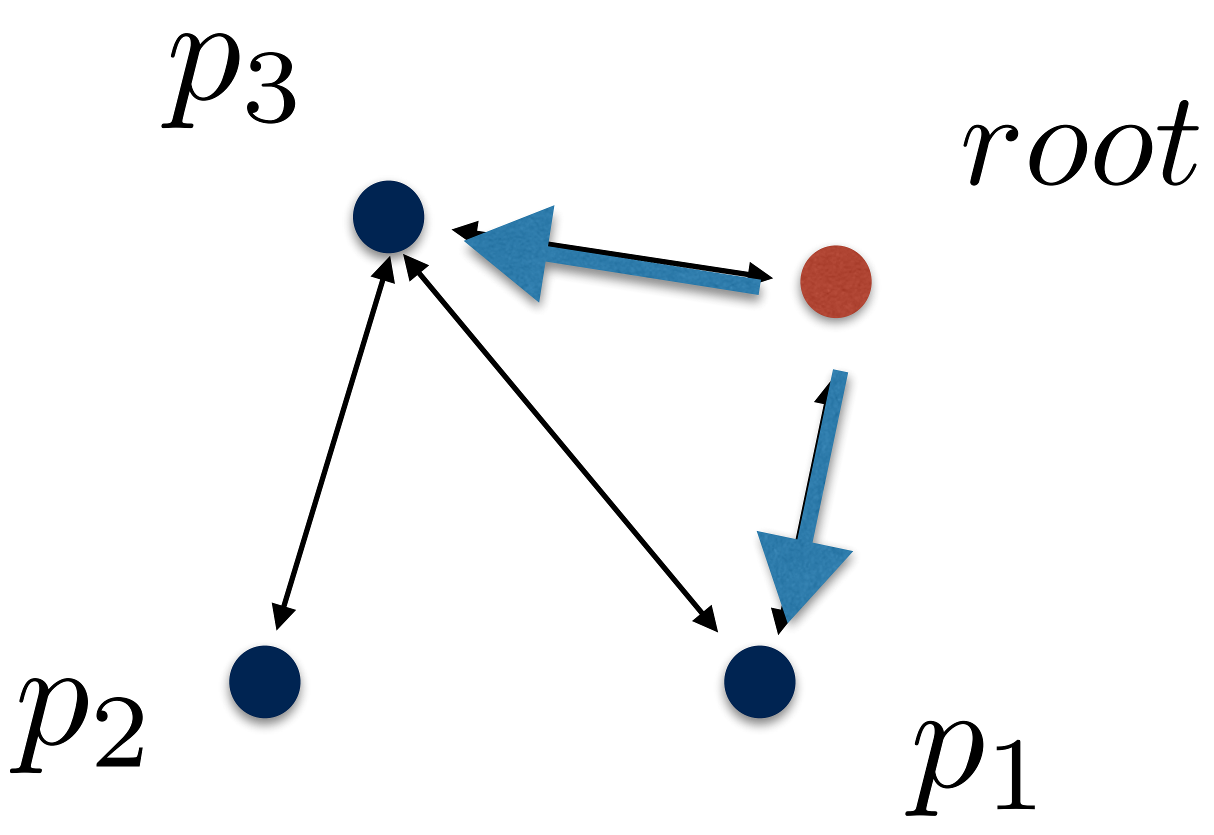
parent(p_1) = r

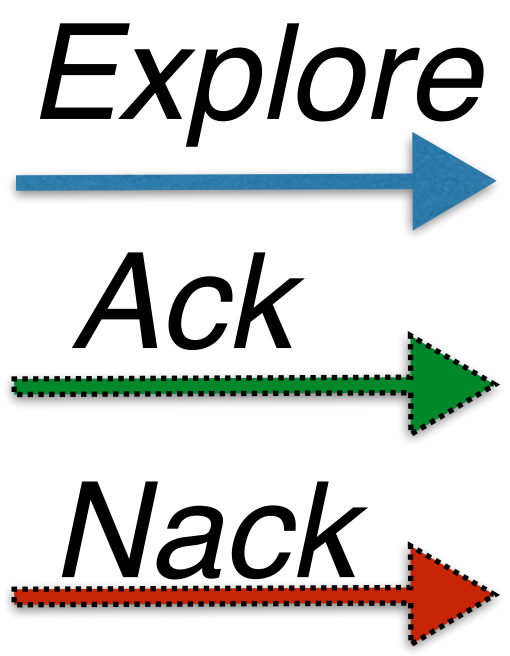
parent(p_2) = ?

parent(p_3) = r

Content Oblivious BFS

(Example)





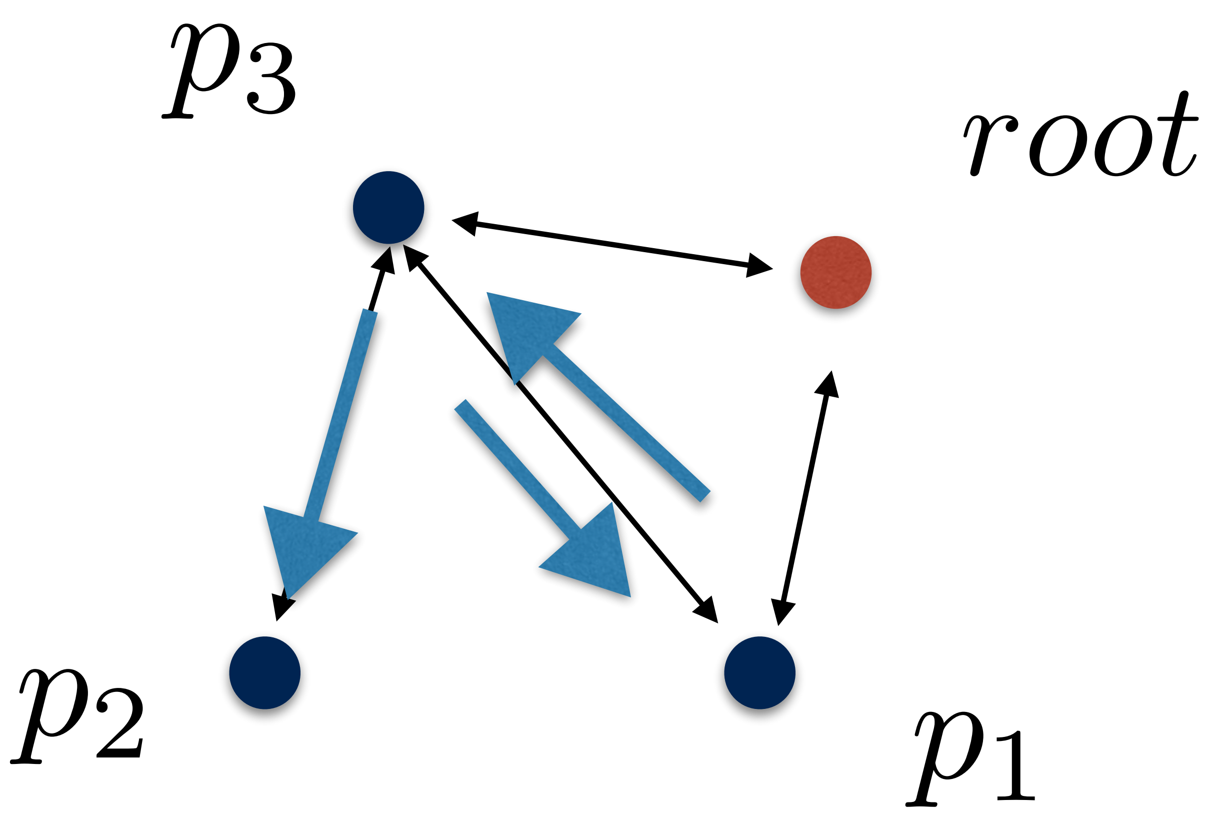
<u>Phase **2**</u>

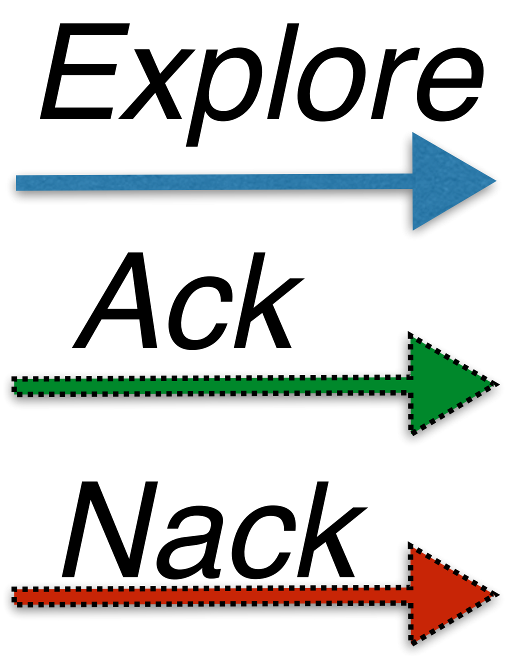
parent(p_1) = r

parent(p_2) = $?$

parent(p_3) = r

Content Oblivious BFS
(Example)





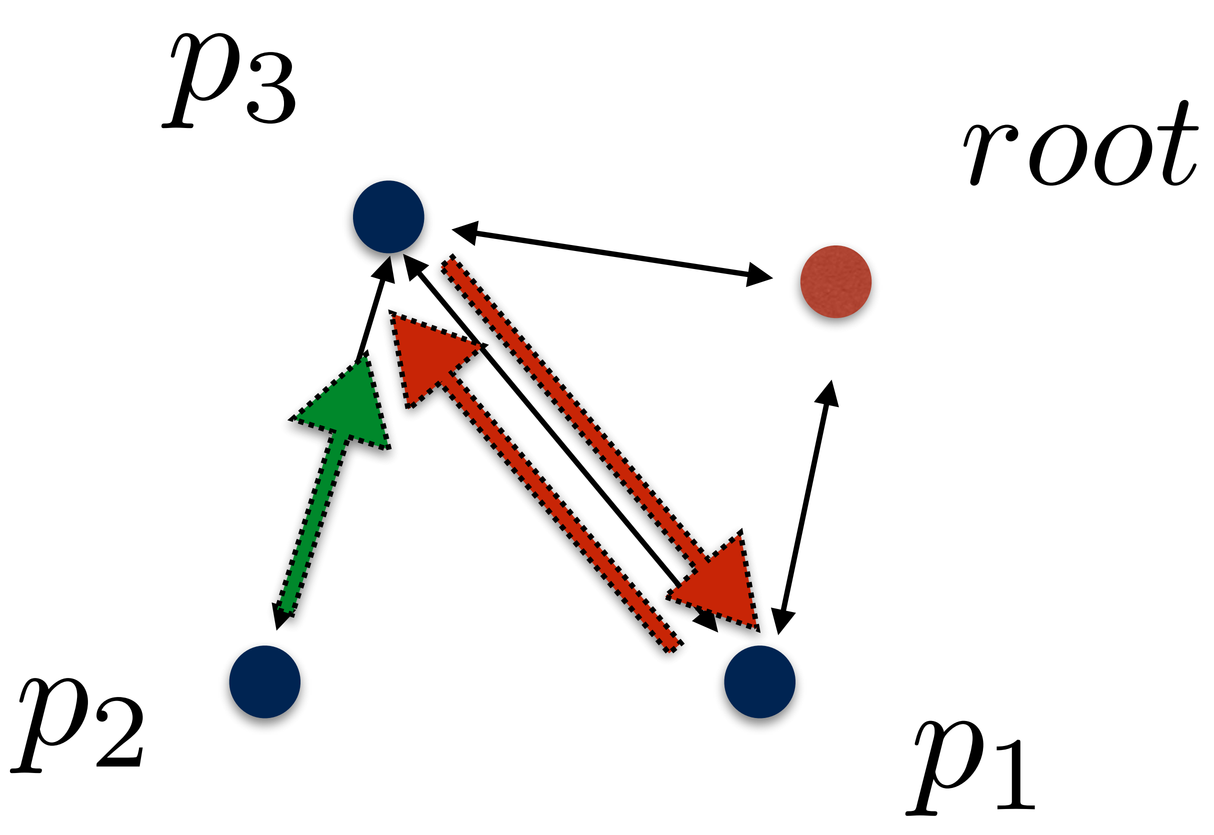
<u>Phase **2**</u>

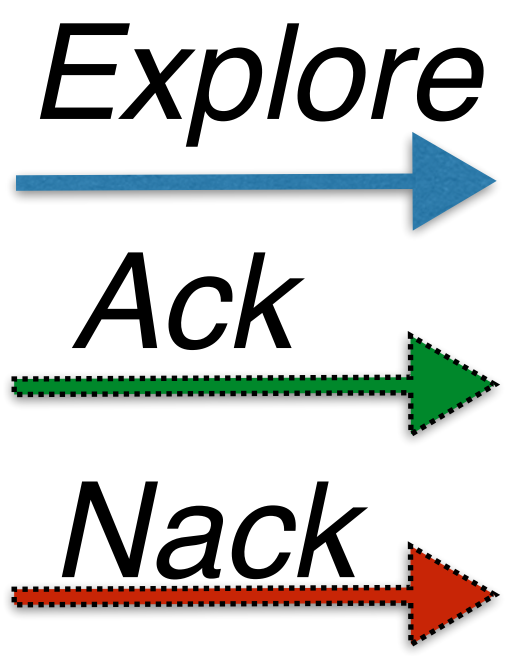
parent(p_1) = r

parent(p_2) = ?

parent(p_3) = r

Content Oblivious BFS
(Example)





<u>Phase **2**</u>

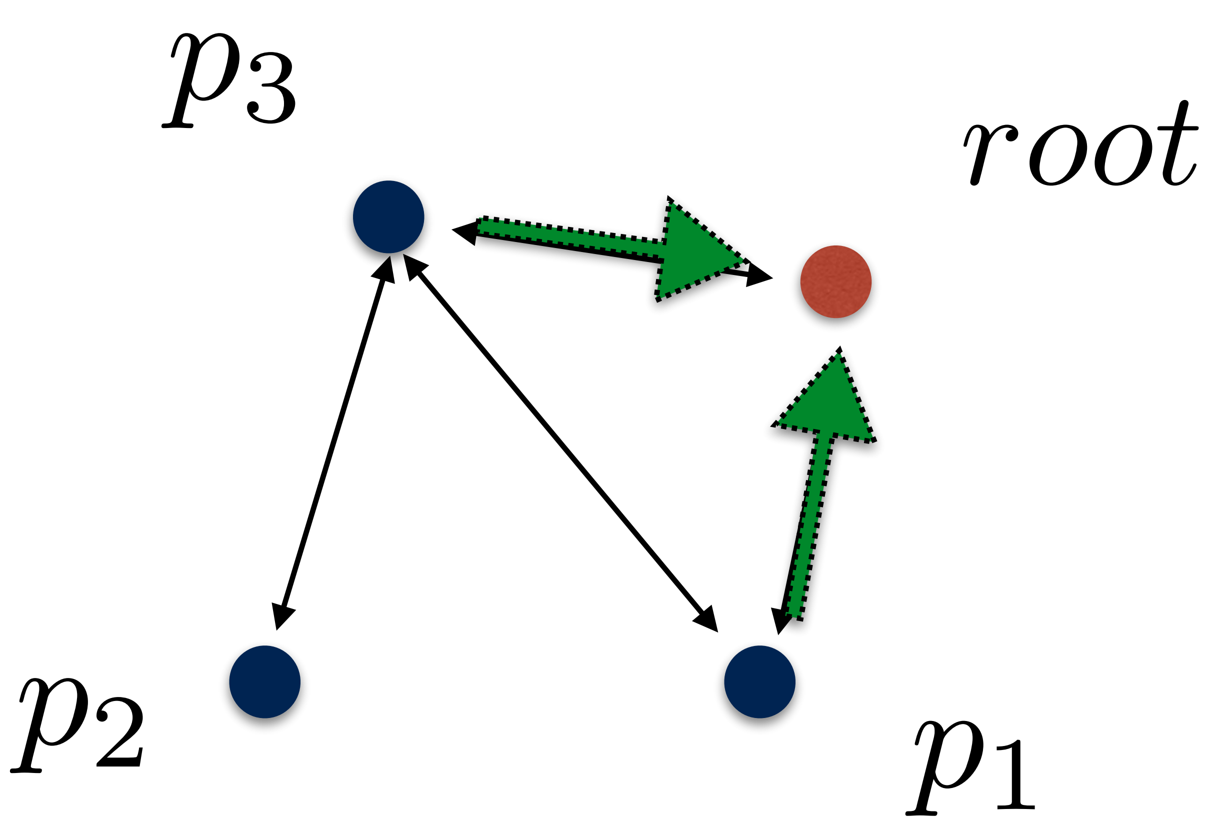
parent(p_1) = r

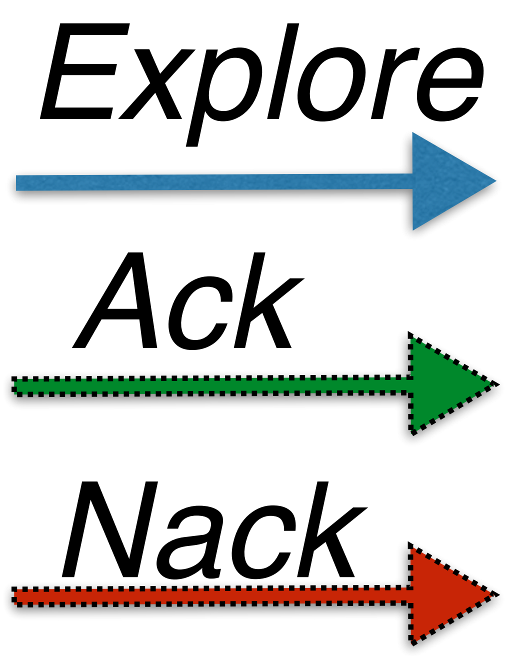
parent(p_2) = p_3

parent(p_3) = r

Content Oblivious BFS

(Example)





Phase **2**

parent(p_1) = r

parent(p_2) = p_3

parent(p_3) = r

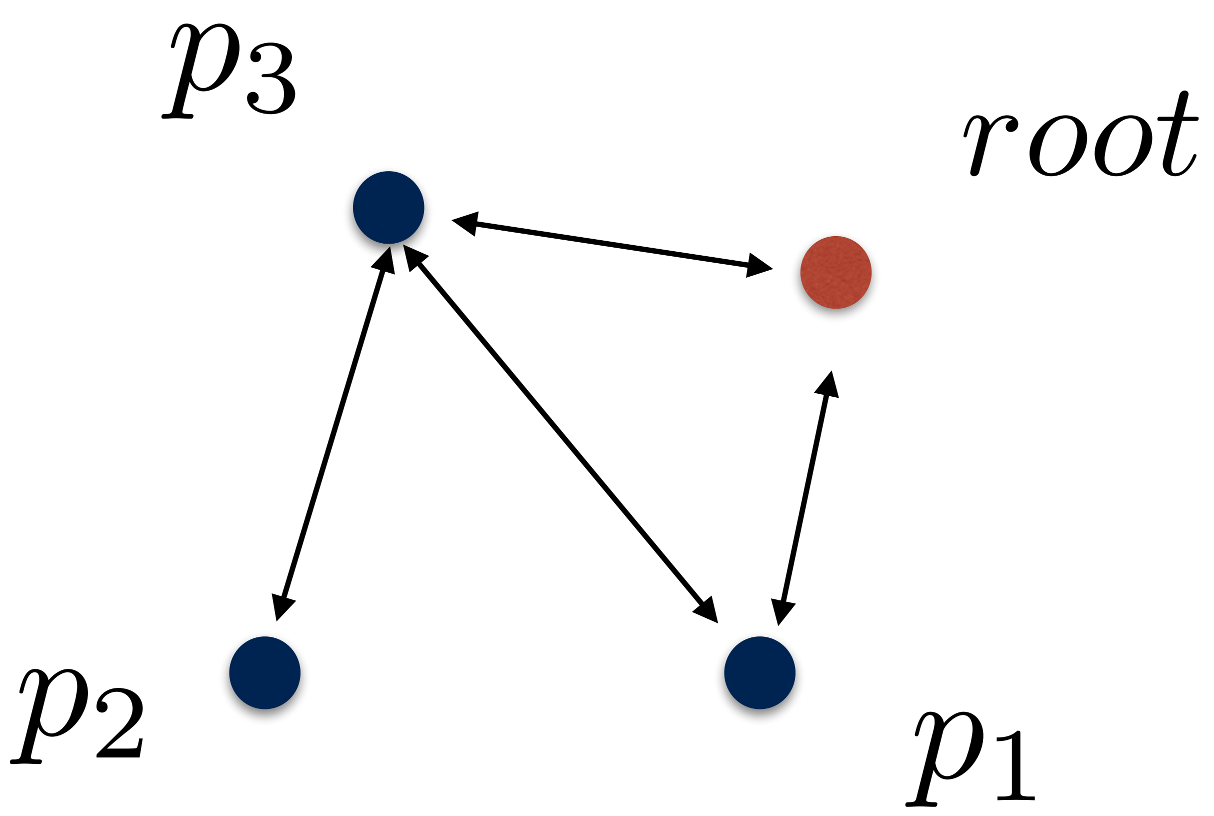
Content Oblivious BFS

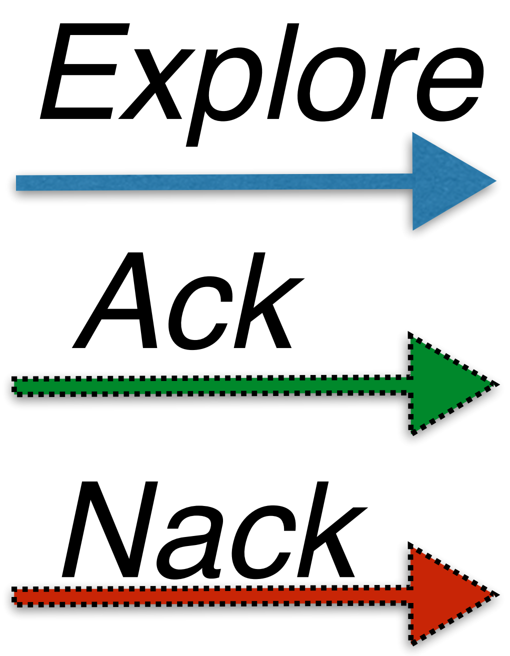
"Observations"

- One message is enough for (***Explore***, ***Ack***)
- How to distinguish ***Nack*** / ***Ack*** ?
- Work **sequentially:**
  - Explore one neighbour at a time.
    Move on to next neighbour only after ***Ack***
  - If a node gets a message from a non-parent outside an **Explore** –
    sender must be a sibling !

Content Oblivious BFS

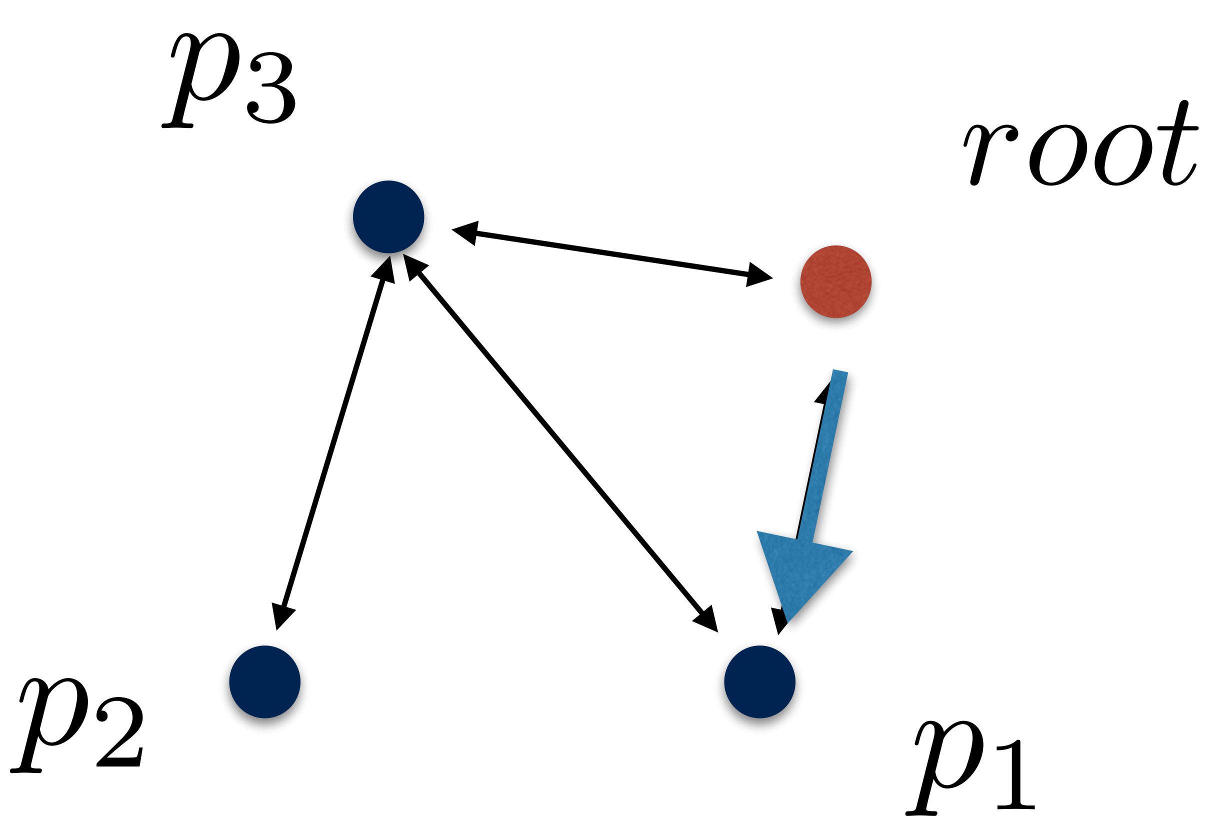
(Example)

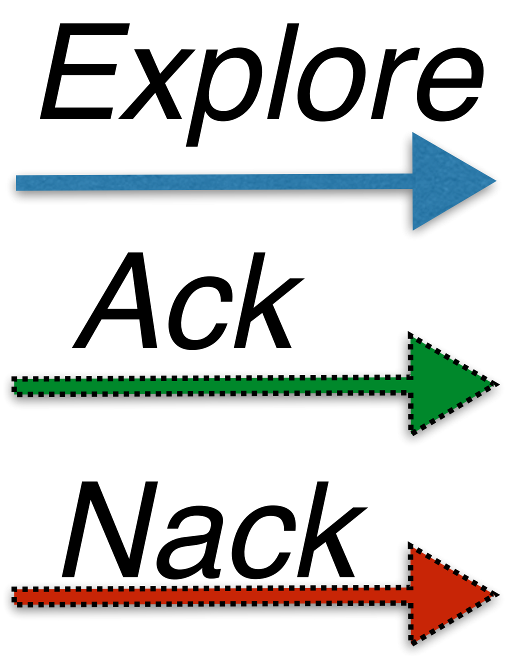




Content Oblivious BFS

(Example)





Phase 1

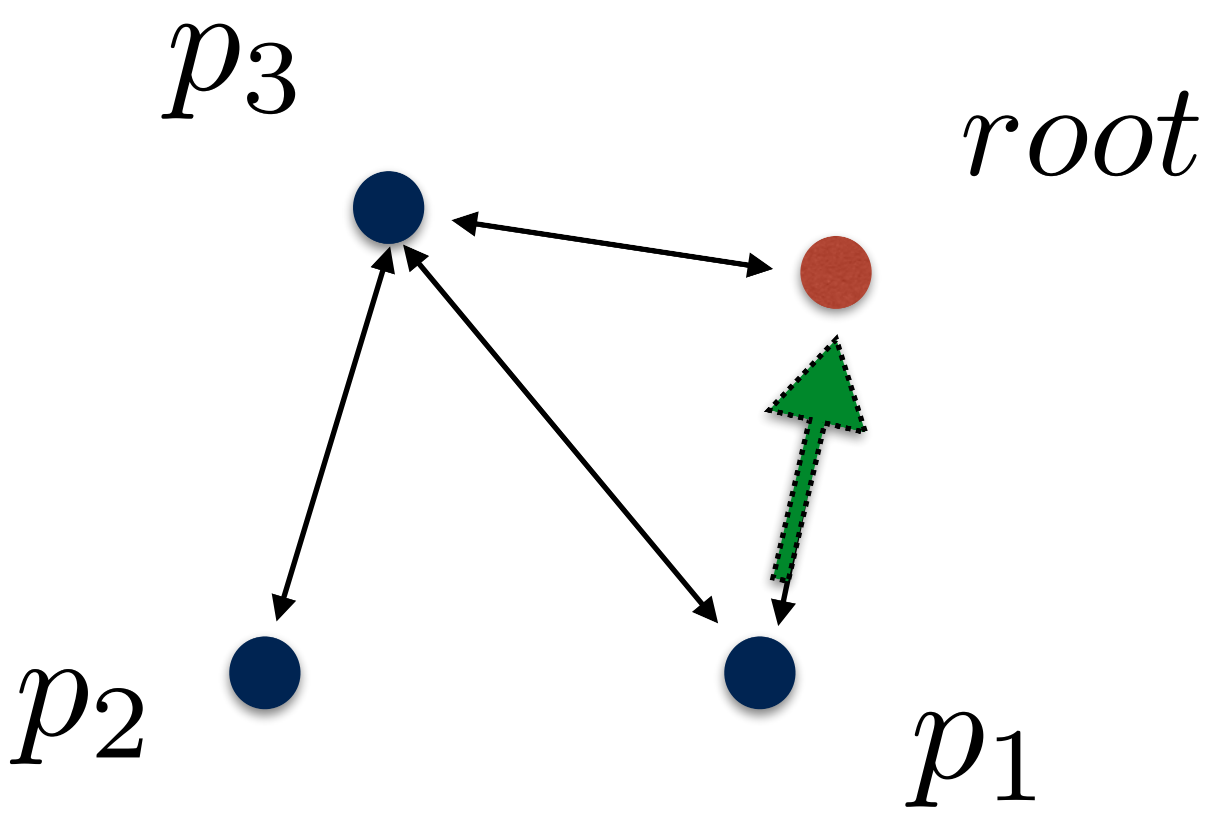
parent(p_1) = ?

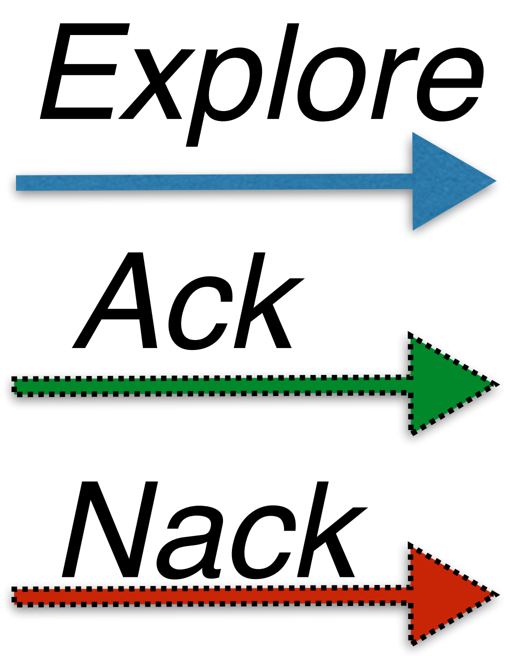
parent(p_2) = ?

parent(p_3) = ?

Content Oblivious BFS

(Example)





<u>Phase 1</u>

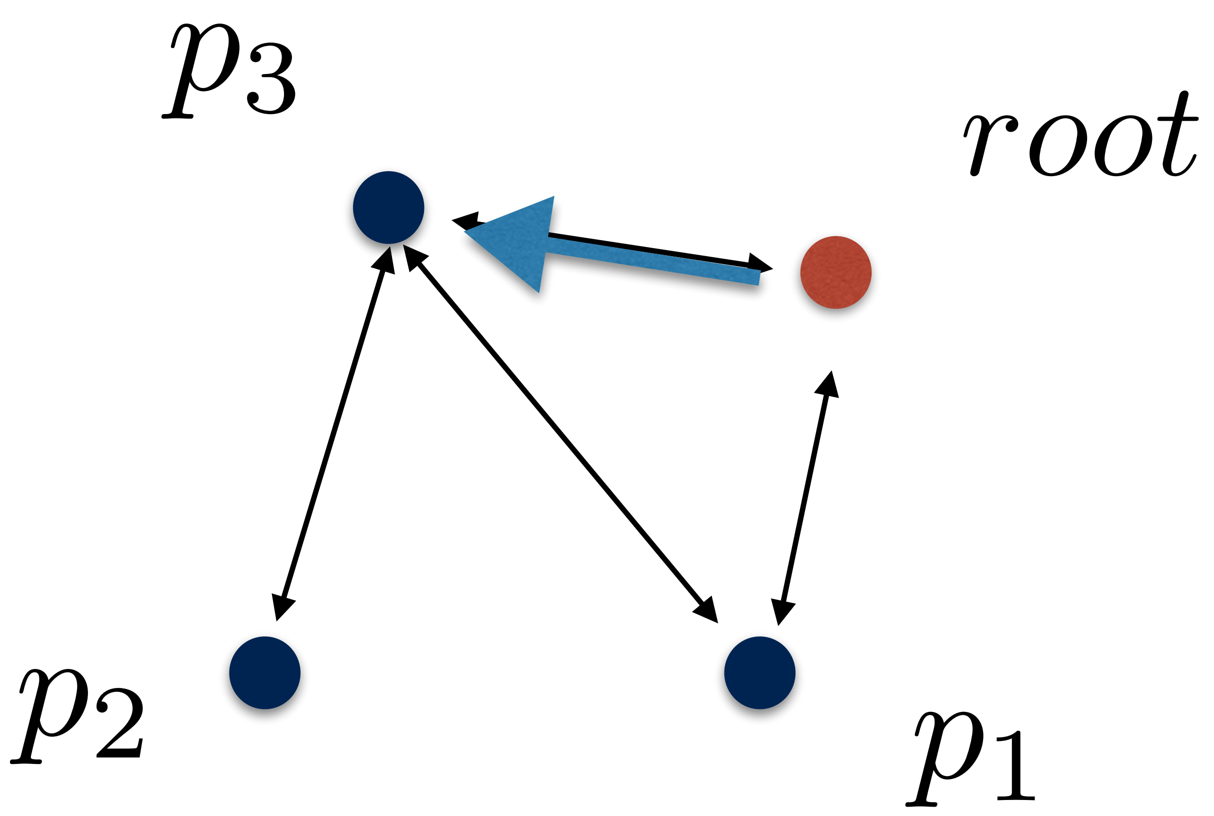
parent(p_1) = r

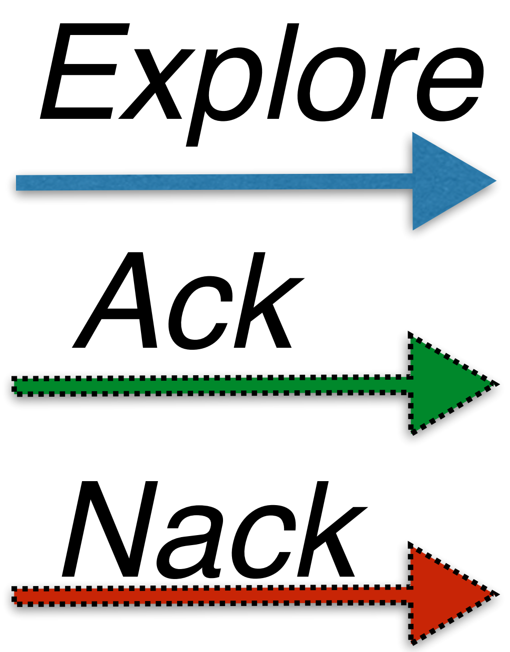
parent(p_2) = $?$

parent(p_3) = $?$

Content Oblivious BFS

(Example)





<u>Phase 1</u>

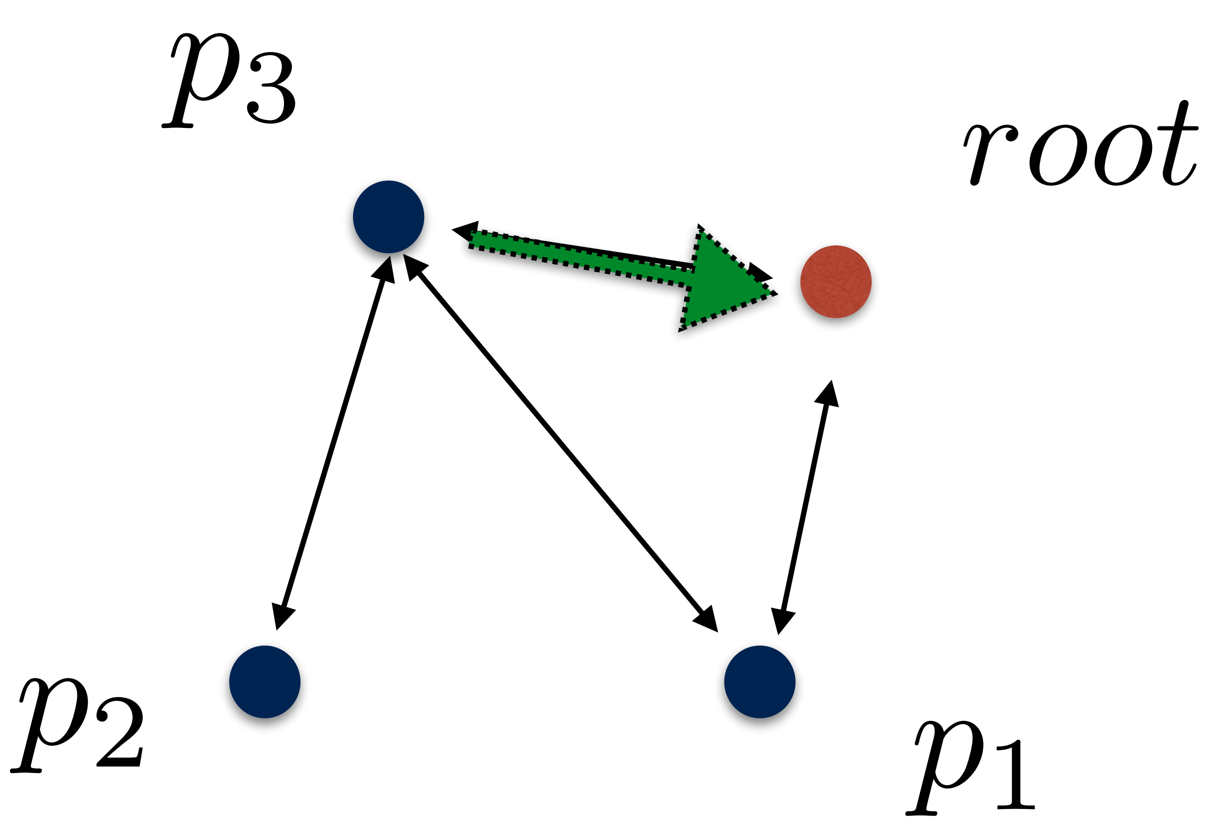
parent(p_1) = r

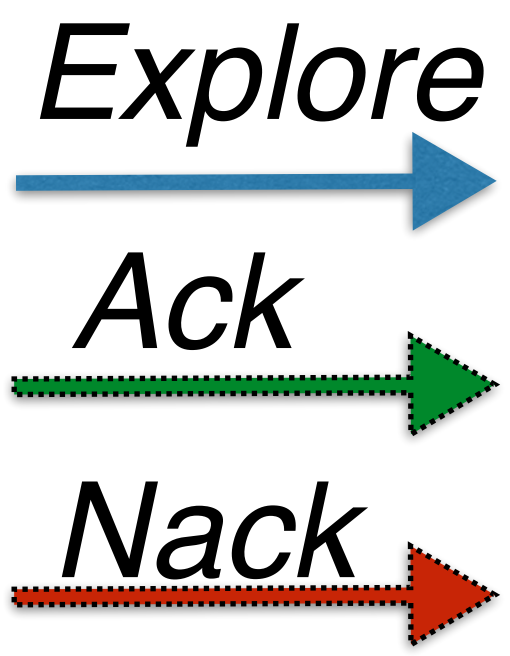
parent(p_2) = $?$

parent(p_3) = $?$

Content Oblivious BFS

(Example)





Phase 1

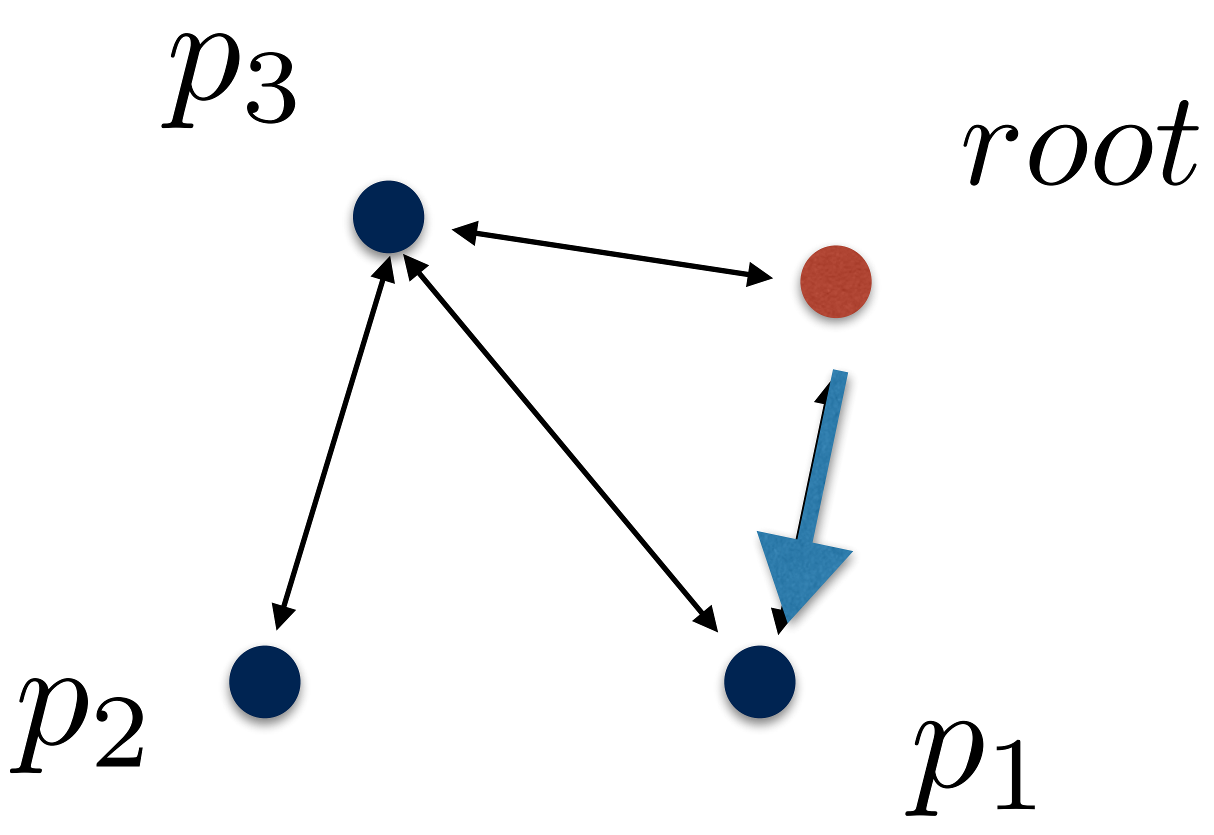
parent(p_1) = r

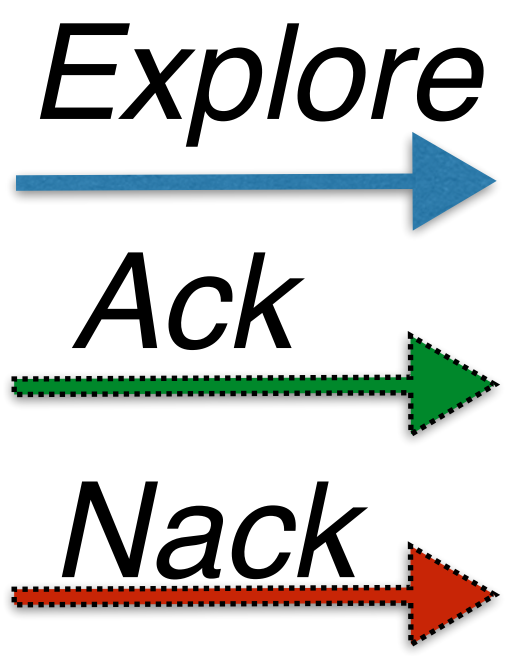
parent(p_2) = $?$

parent(p_3) = r

Content Oblivious BFS

(Example)





Phase **2**

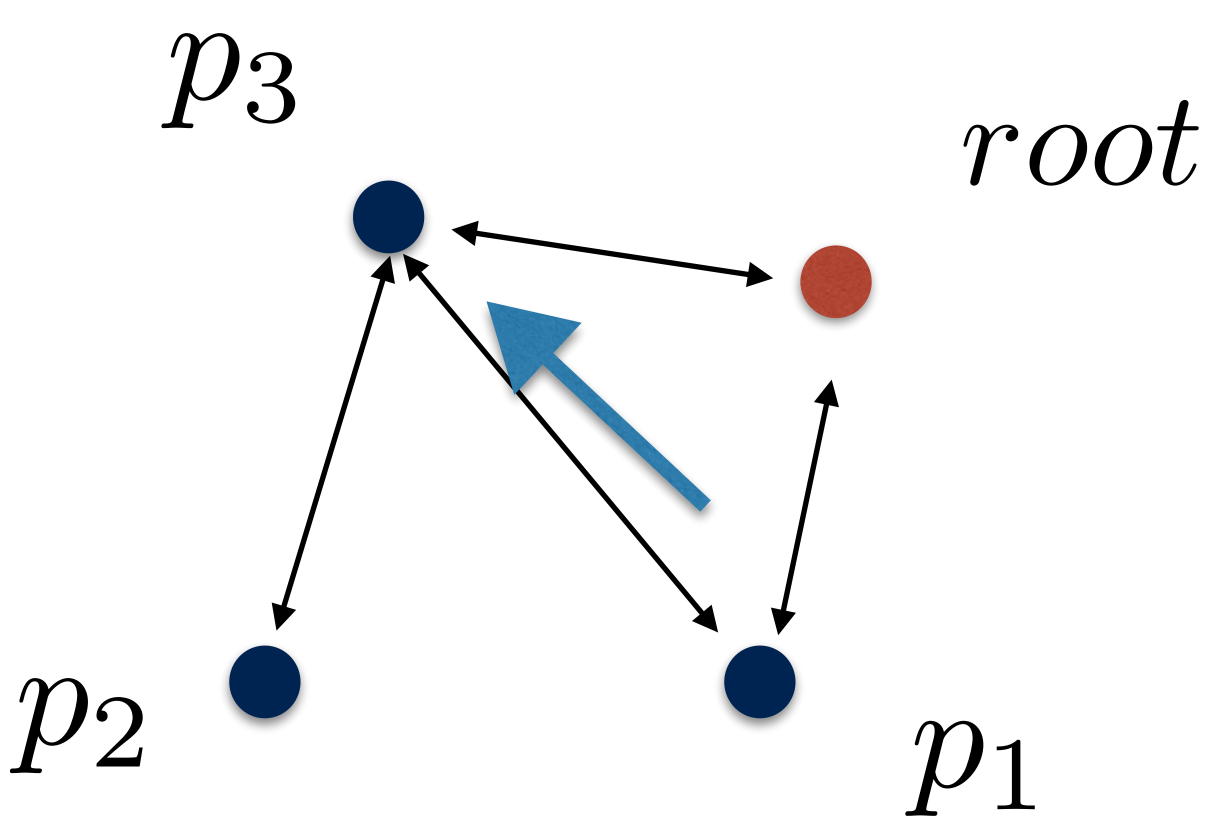
parent(p_1) = r

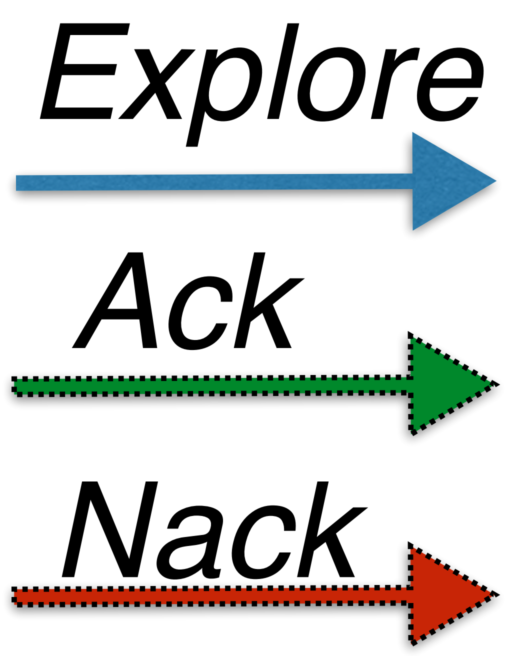
parent(p_2) = $?$

parent(p_3) = r

Content Oblivious BFS

(Example)





Phase **2**

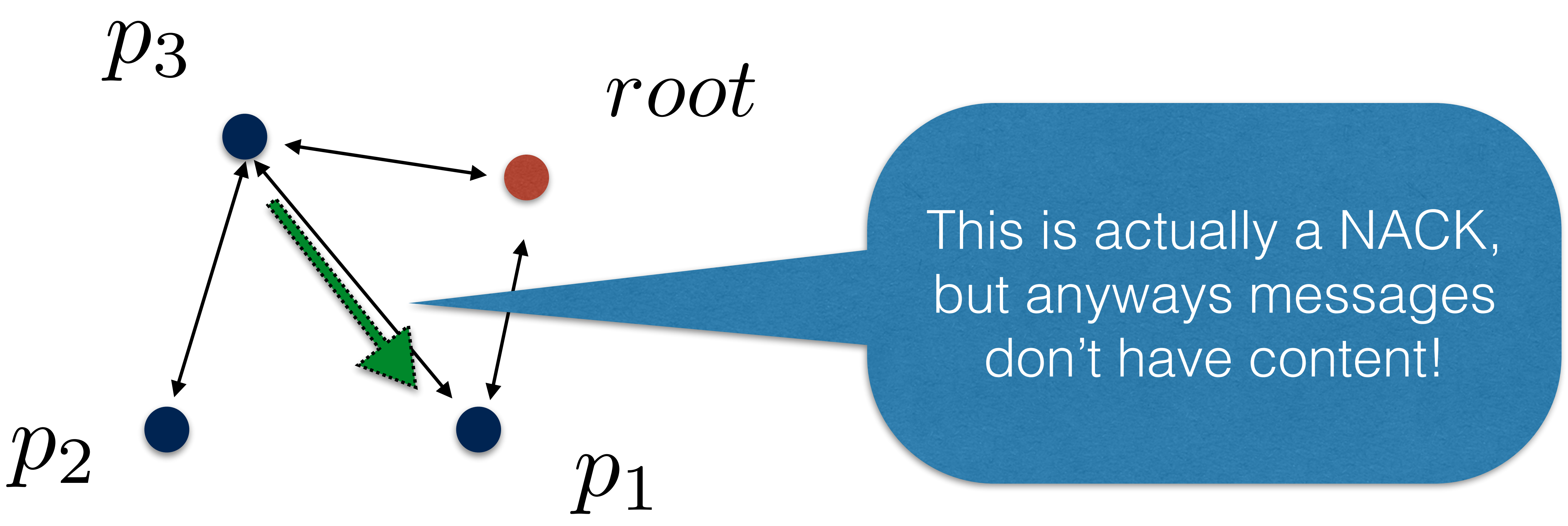
parent(p_1) = r

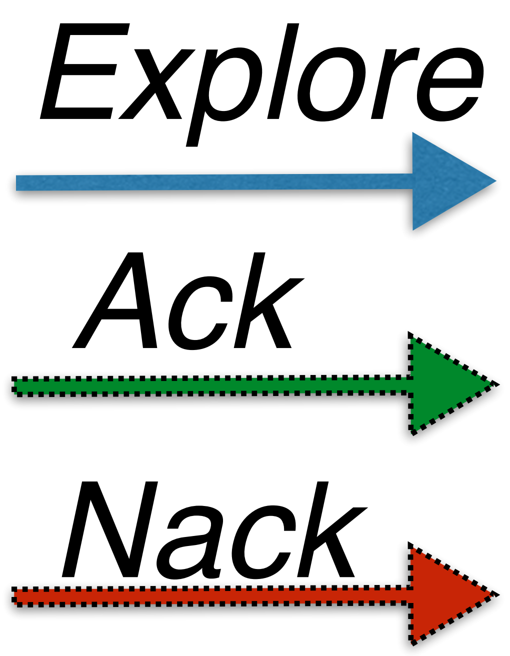
parent(p_2) = $?$

parent(p_3) = r

Content Oblivious BFS

(Example)





<u>Phase **2**</u>

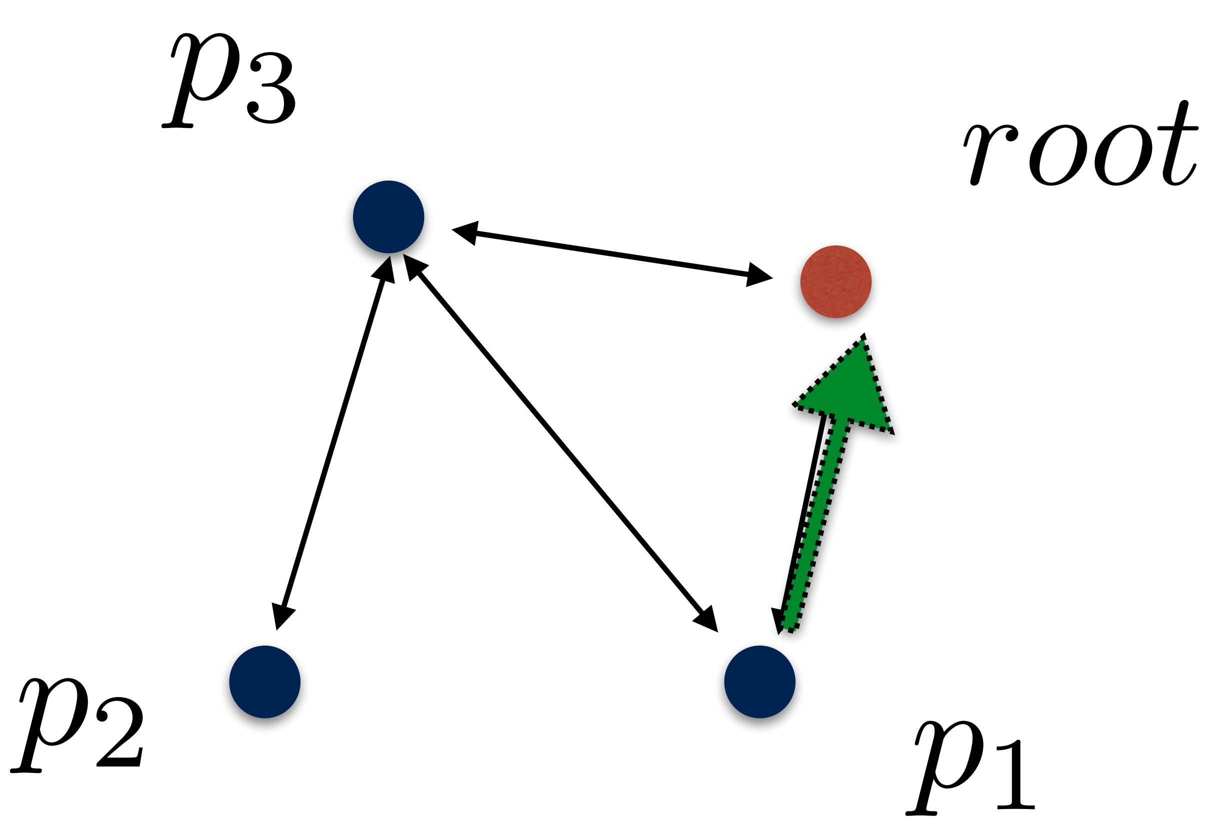
parent(p_1) = r sibling(p_1) =

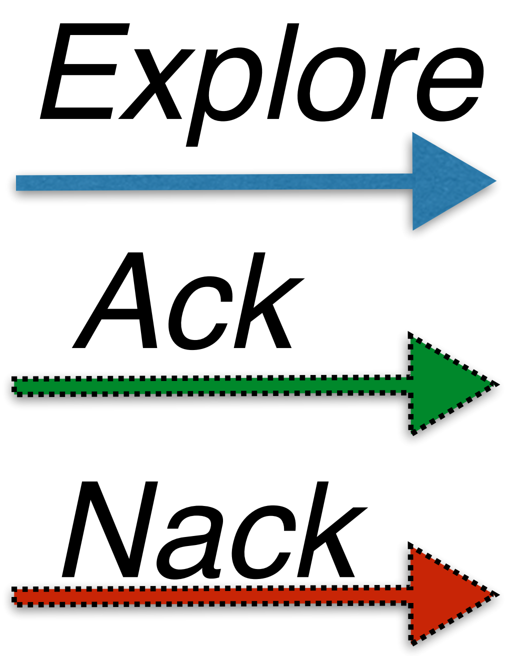
parent(p_2) = $?$ sibling(p_2) =

parent(p_3) = r sibling(p_3) = p_1

Content Oblivious BFS

(Example)





<u>Phase **2**</u>

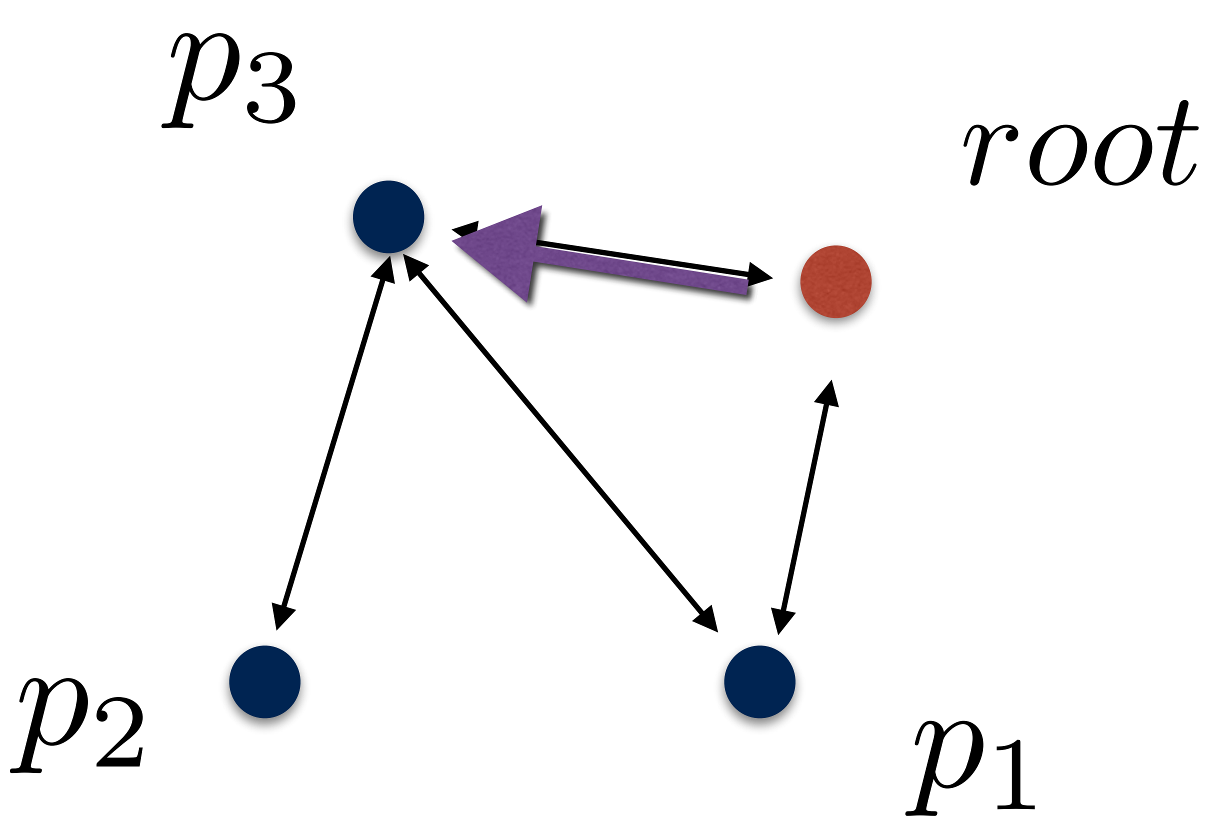
parent(p_1) = r sibling(p_1) =

parent(p_2) = $?$ sibling(p_2) =

parent(p_3) = r sibling(p_3) = p_1

Content Oblivious BFS

(Example)



<u>Phase **2**</u>

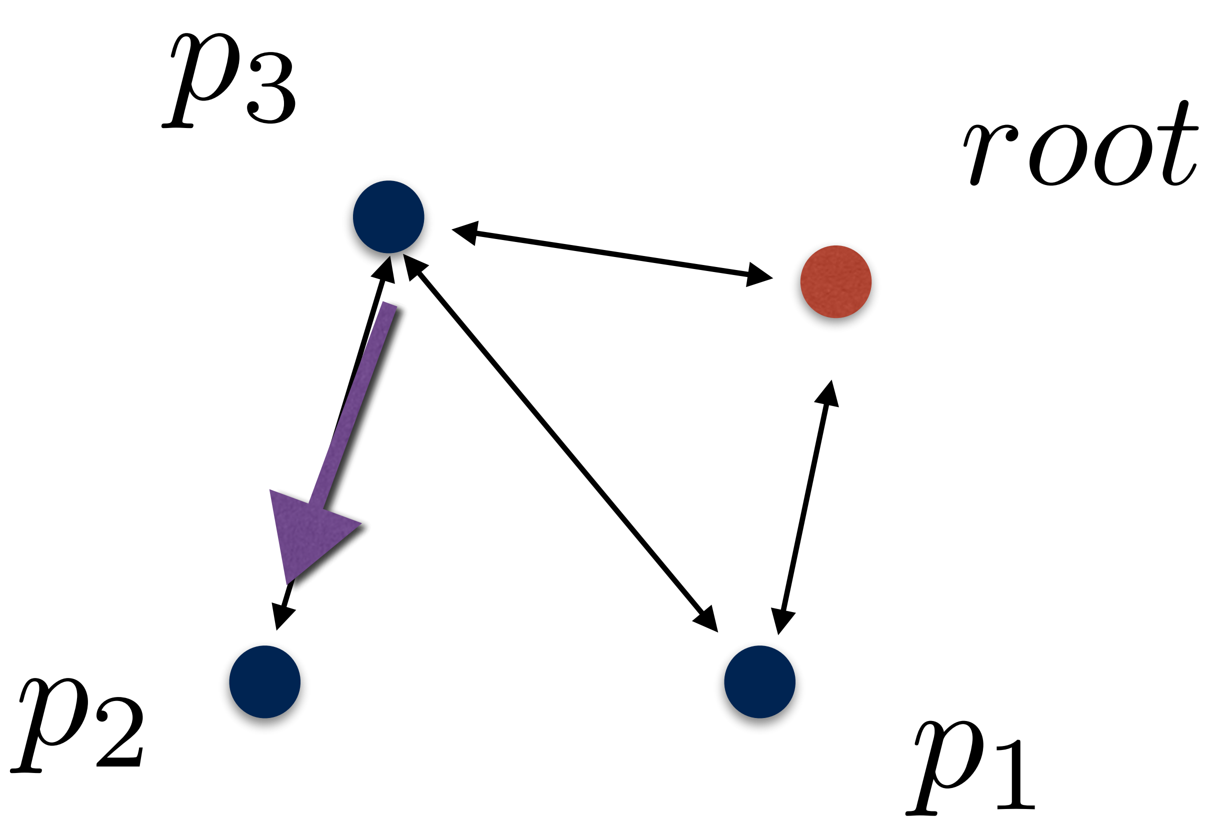
parent(p_1) = r sibling(p_1) =

parent(p_2) = $?$ sibling(p_2) =

parent(p_3) = r sibling(p_3) = p_1

Content Oblivious BFS

(Example)



<u>Phase **2**</u>

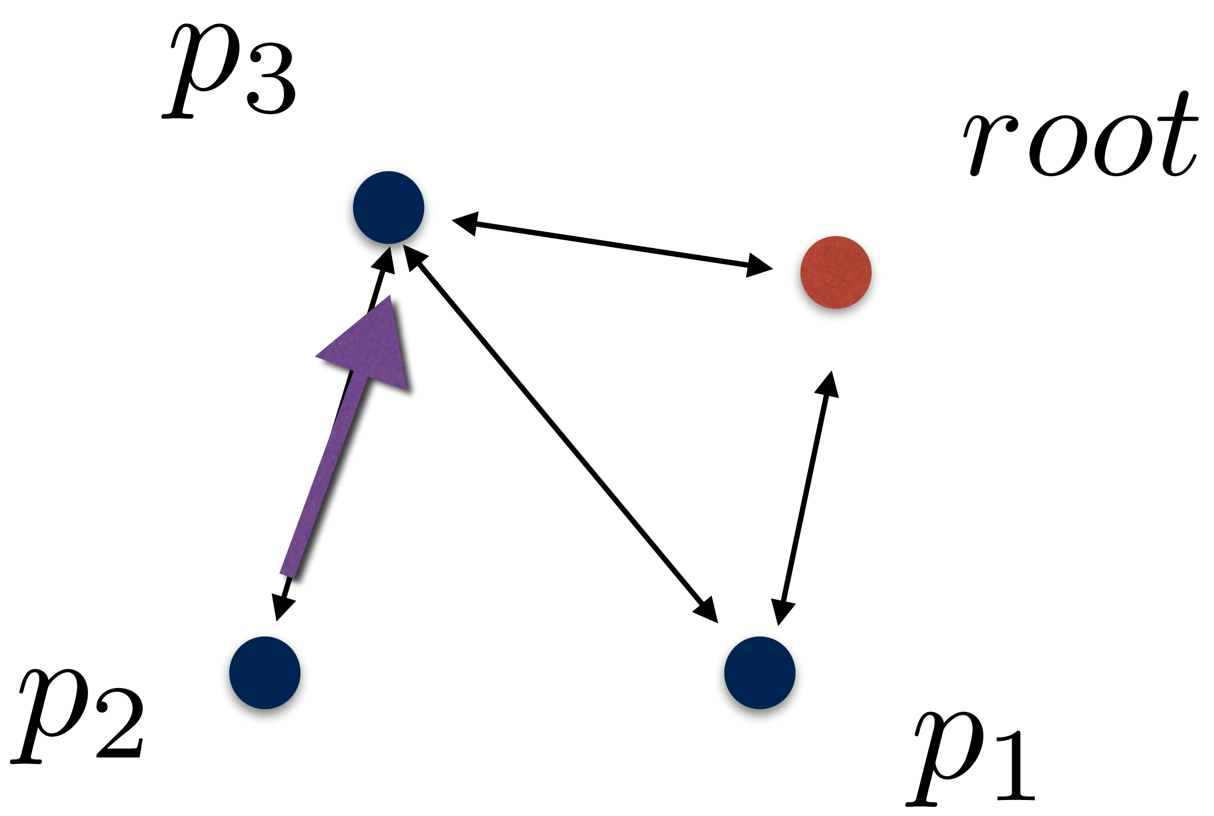
parent(p_1) = r sibling(p_1) =

parent(p_2) = $?$ sibling(p_2) =

parent(p_3) = r sibling(p_3) = p_1

Content Oblivious BFS

(Example)



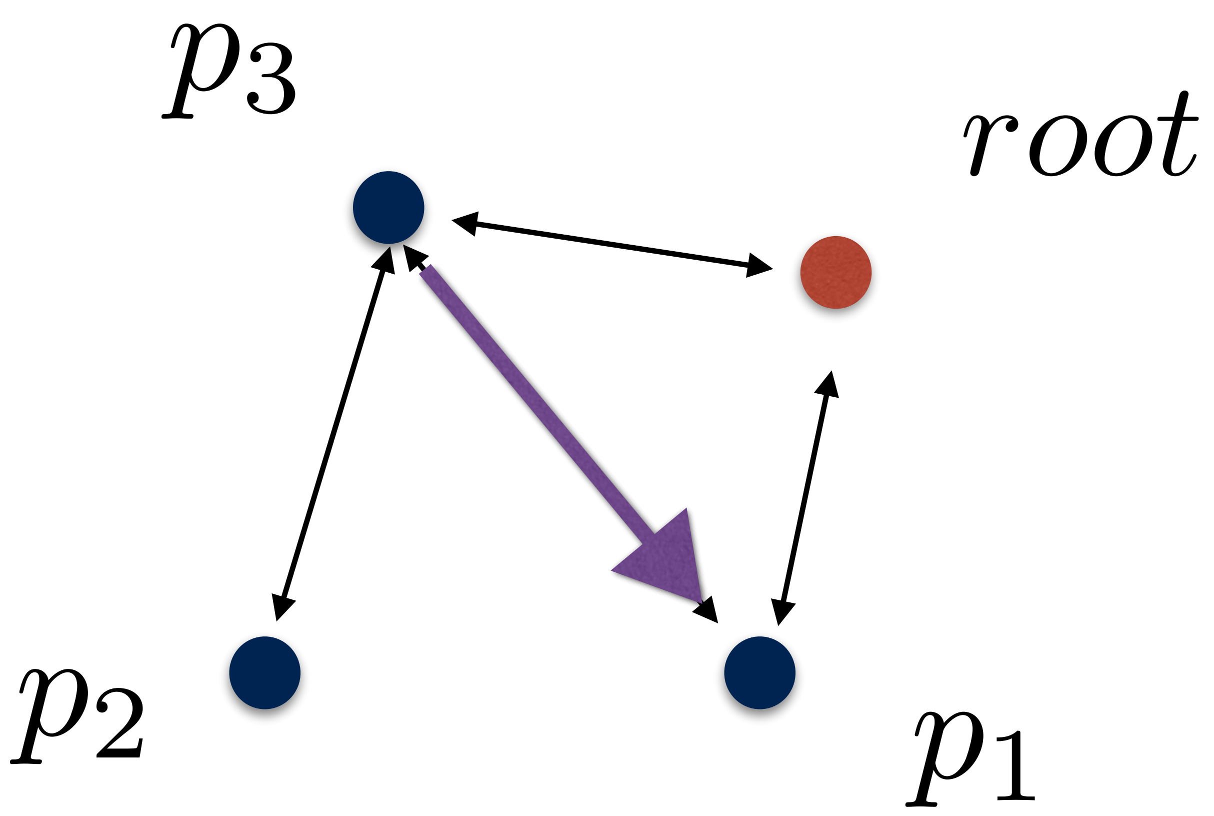
Phase **2**

parent(p_1) = r sibling(p_1) =

parent(p_2) = p_3 sibling(p_2) =

parent(p_3) = r sibling(p_3) = p_1

Content Oblivious BFS
(Example)

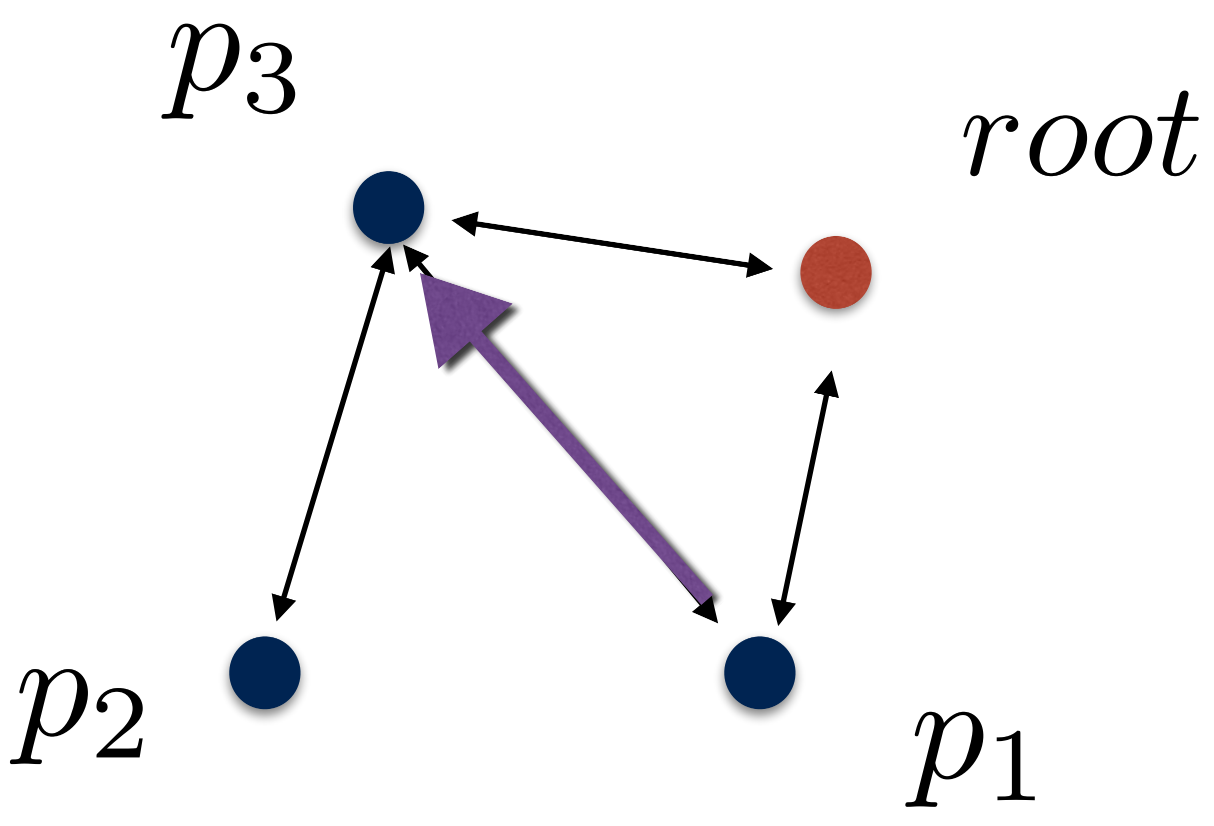


<u>Phase **2**</u>

parent(p_1) = r	sibling(p_1) =
parent(p_2) = p_3	sibling(p_2) =
parent(p_3) = r	sibling(p_3) = p_1

Content Oblivious BFS

(Example)

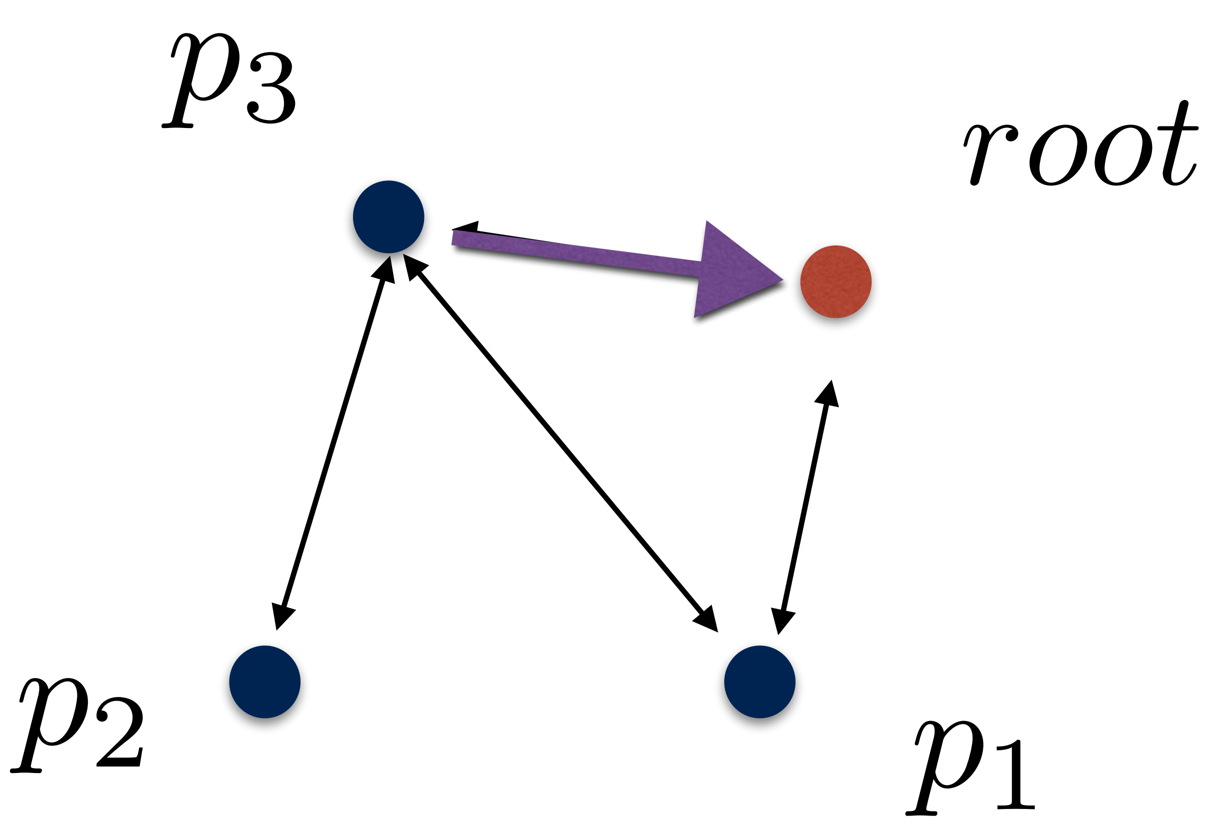


<u>Phase **2**</u>

parent(p_1) = r	sibling(p_1) = p_3
parent(p_2) = p_3	sibling(p_2) =
parent(p_3) = r	sibling(p_3) = p_1

Content Oblivious BFS

(Example)



<u>Phase **2**</u>

parent(p_1) = r sibling(p_1) = p_3

parent(p_2) = p_3 sibling(p_2) =

parent(p_3) = r sibling(p_3) = p_1

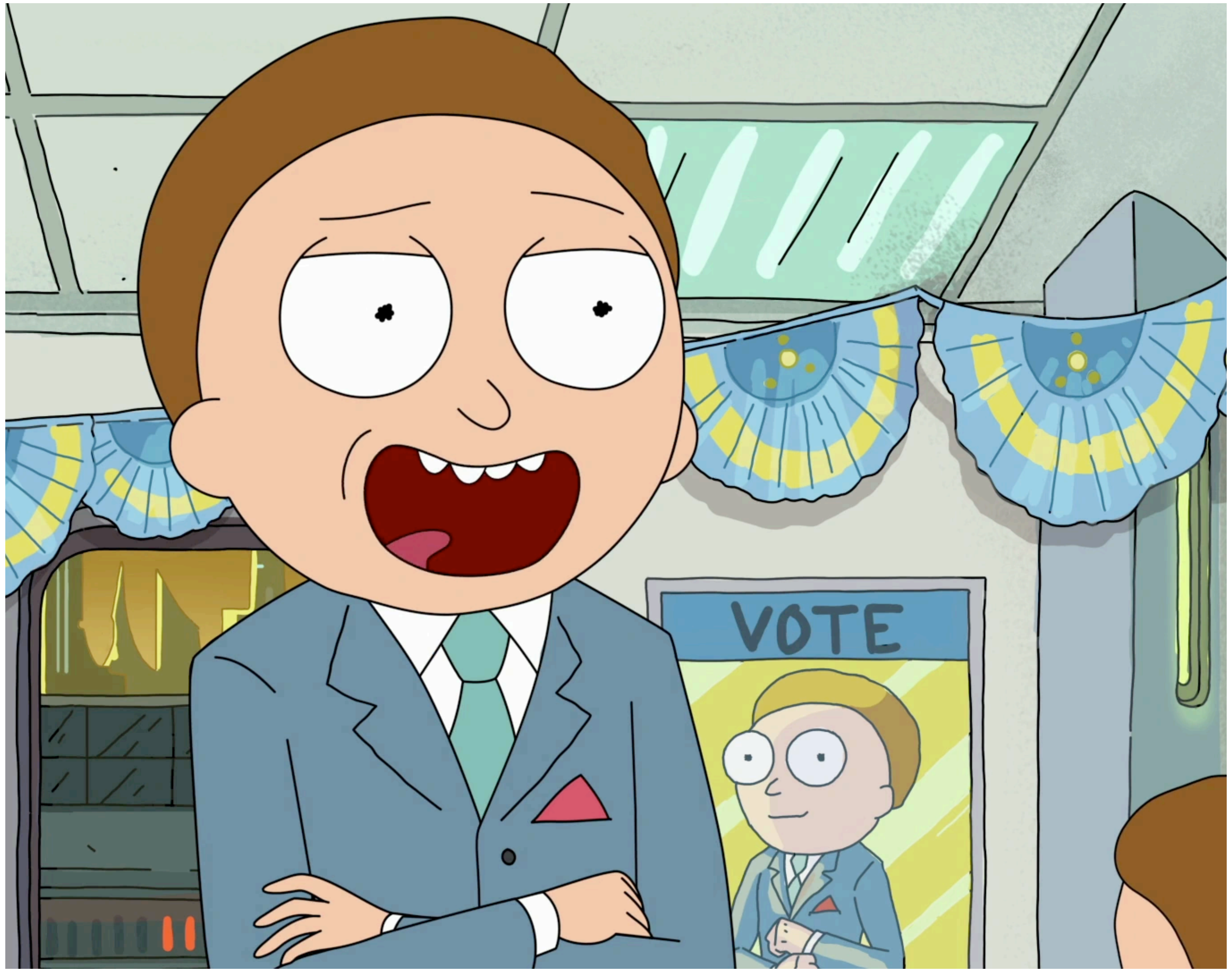
Content Oblivious BFS

Further Observations

- Requires knowing $n = |V|$ or a bound on it (for termination)
- The sequential method performs “controlled DFS”
  - Can be modified to obtain a content-oblivious DFS algorithm
- Complexity: $O(|V| \cdot |E|)$ signals
- *Requires a leader ….*

Content Oblivious

Leader Election



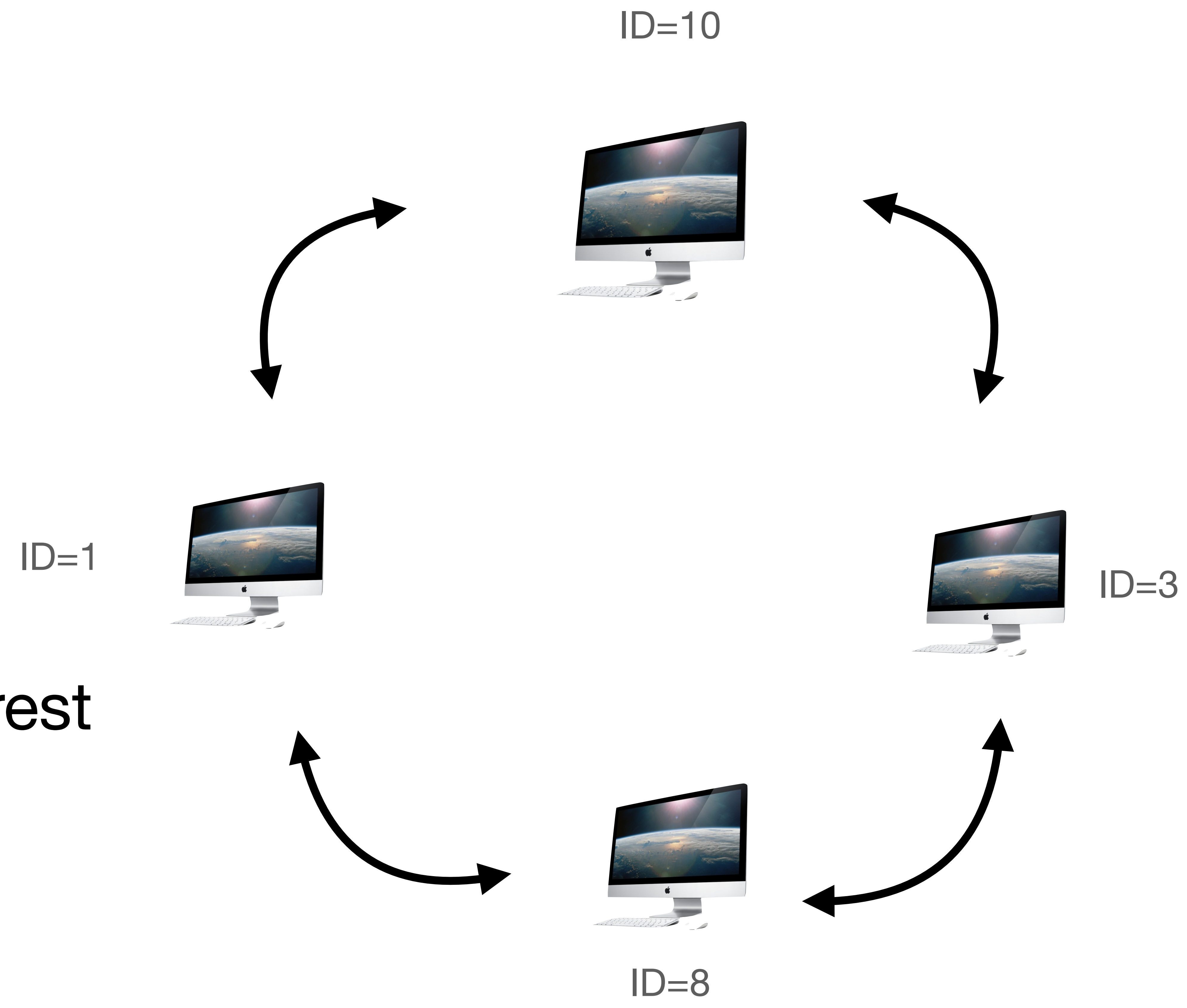
Content Oblivious Leader Election

- **Theorem:**
  Content-Oblivious Leader Election is possible in **Rings**

[Frei, Ghazi, Gelles, Nolin, DISC'24]

Content Oblivious Leader Election

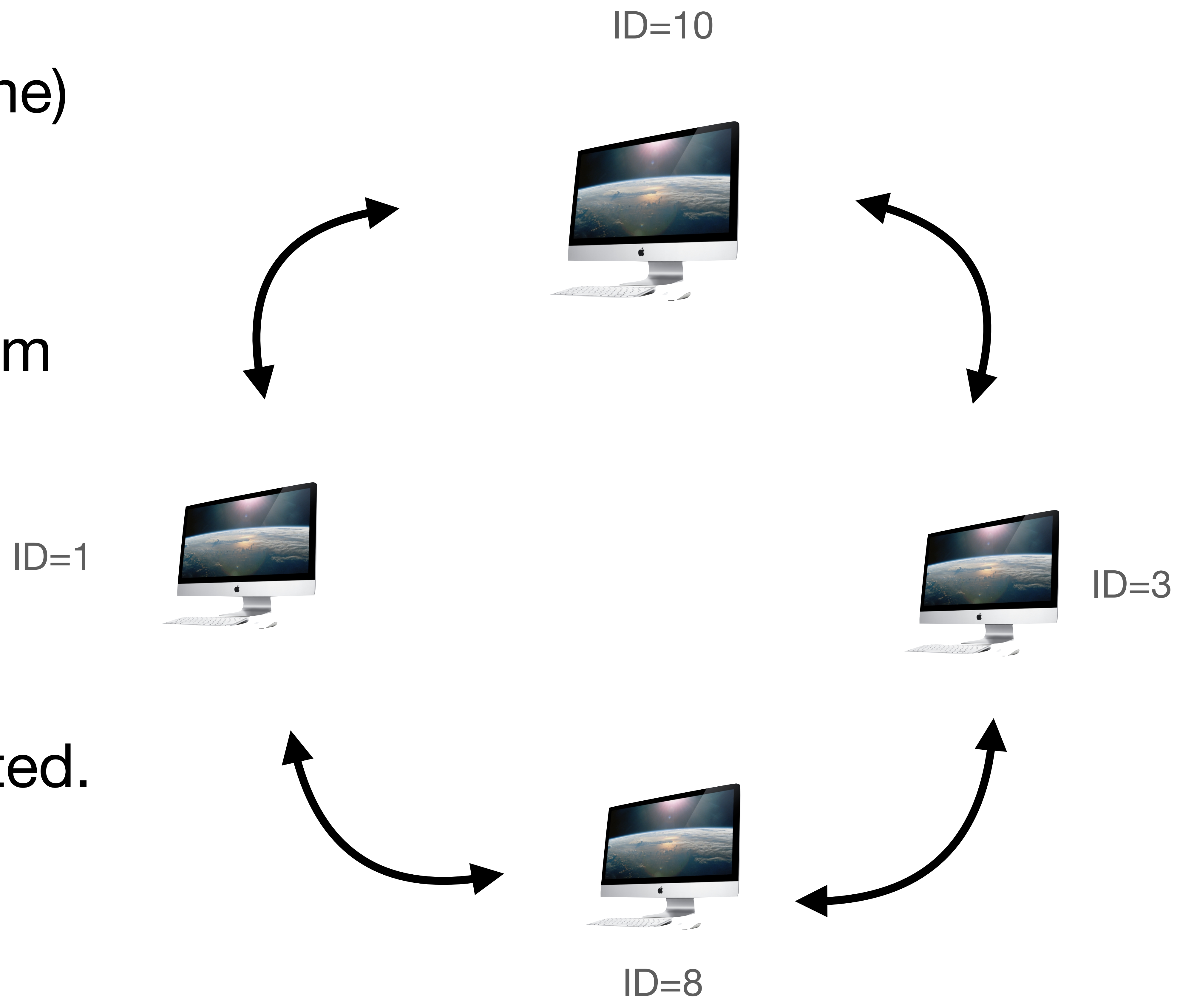
- Leader election on (content-based messages, rings):
  - Elect the node with maximal ID
    (e.g., send your ID clockwise, $O(n^2)$ messages)
  - No IDs?   Symmetry Braking is Impossible!
- Let's imitate this protocol using signals:
  - Each v sends ID_v many signals **clockwise**
  - If v receives more than ID_v signals, propagate the rest (v is not the leader)



Termination!?

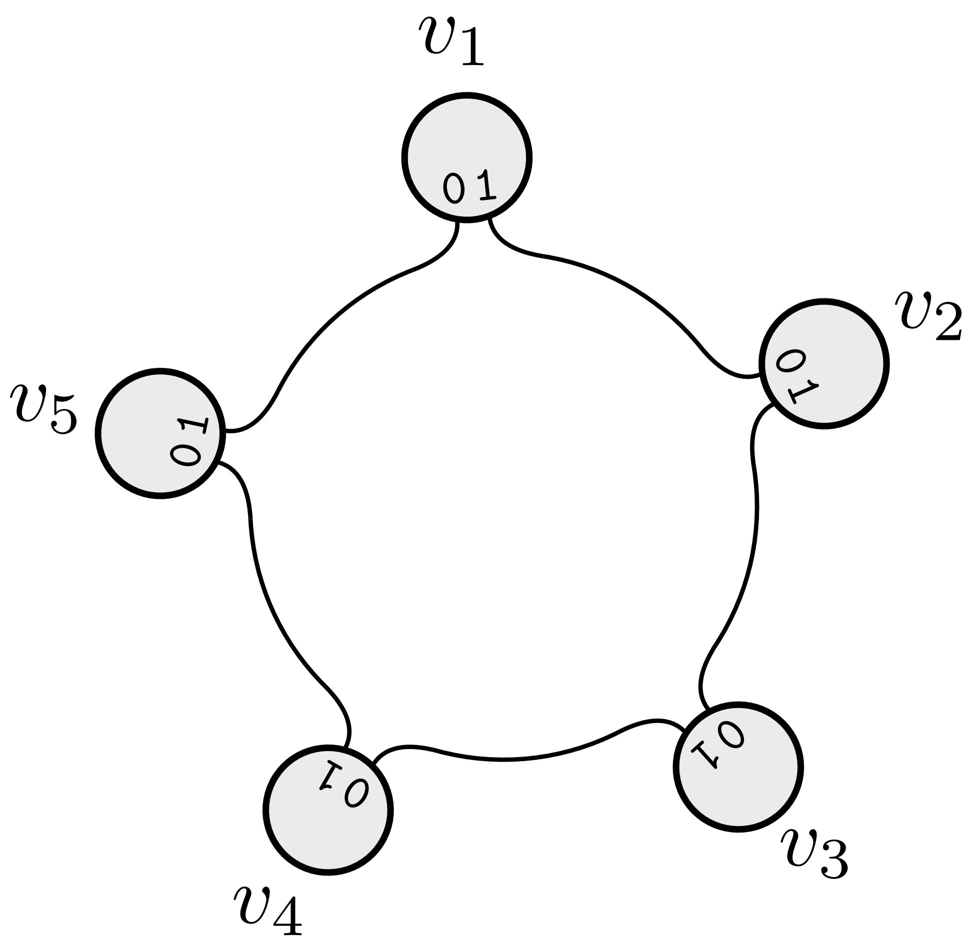
Terminating CO LE

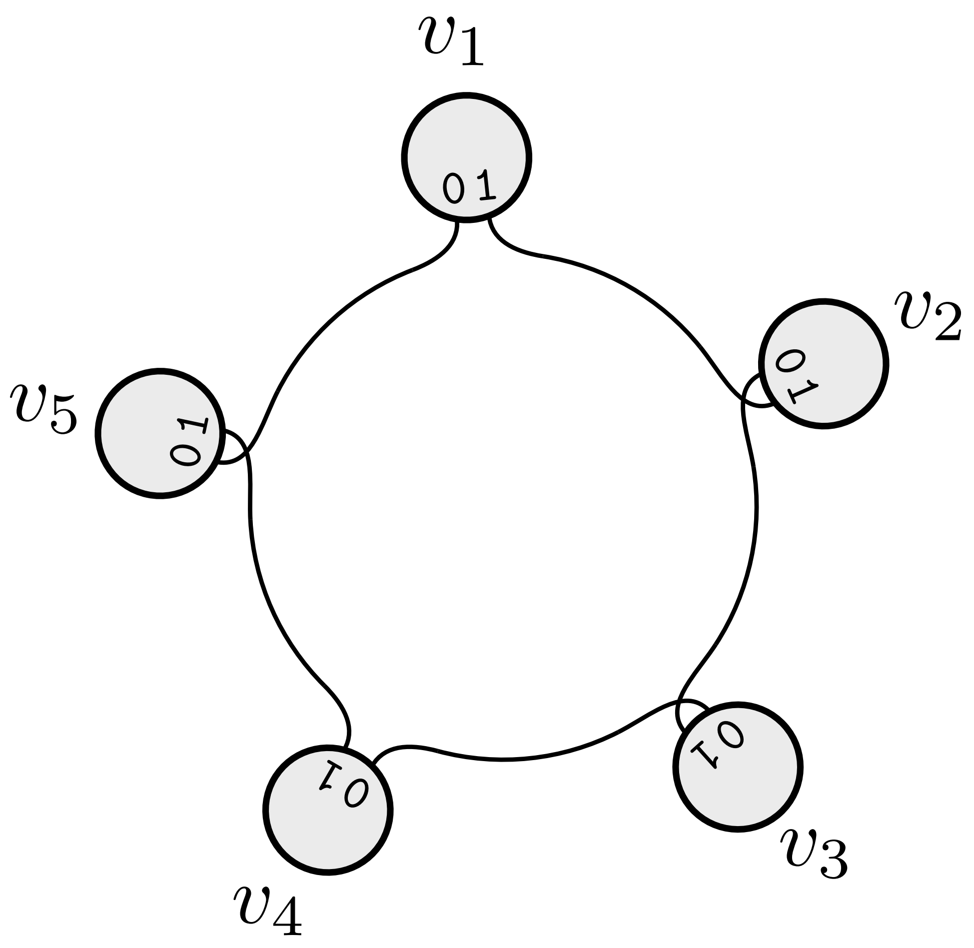
- Note: after previous phase all nodes see exactly $\max$ ID clockwise signals!
- The leader is the last to see $\max$ ID signals
  (but it does not know whether more signals are yet to come)
- We did not use the **counter-clockwise** path!
  - When #signals $= \mathrm{ID}_v$, node v starts the same algorithm in the **counter-clockwise** direction
  - When #signals$_{CCW}$ $=$ $\max$ ID, terminate
- When the node with $\max$ ID receives $\max$ ID **counter-clockwise** signals, all other nodes have terminated.
- Complexity: $n(2\mathrm{ID}_{\max} + 1)$



Orienting a ring

- We assumed that the ring is oriented:
  - Nodes distinguish the CW and the CCW directions.

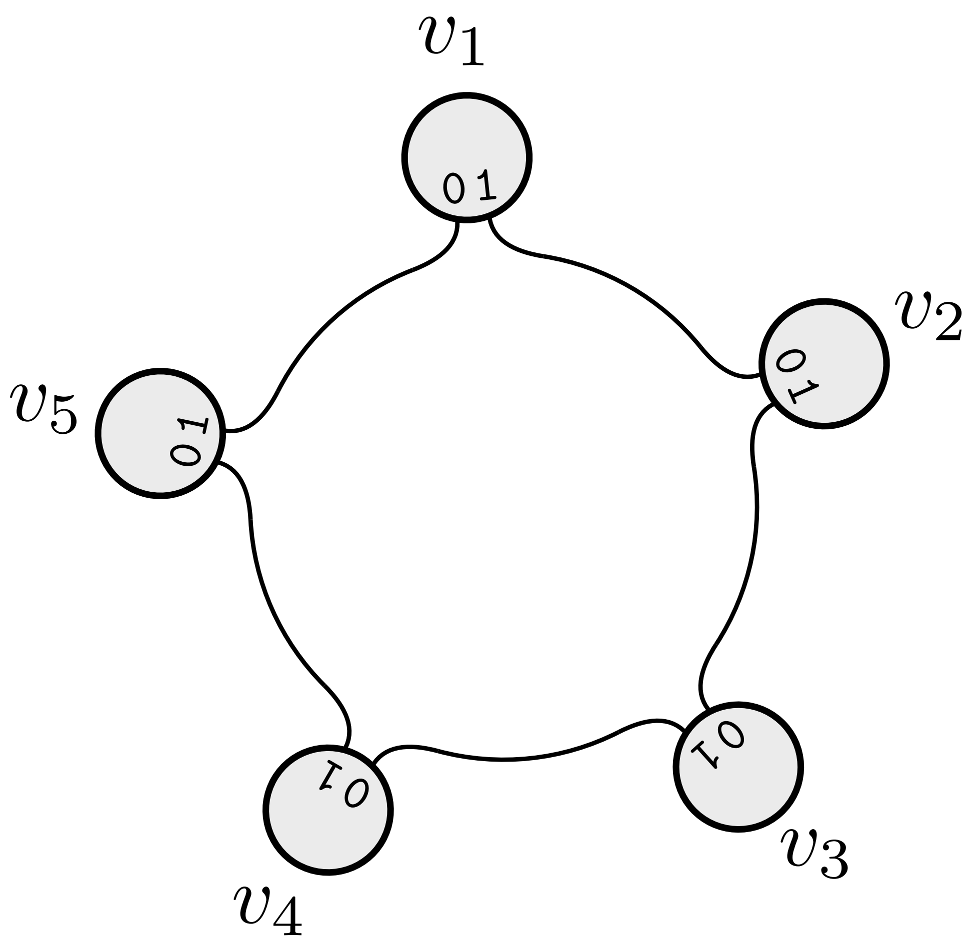


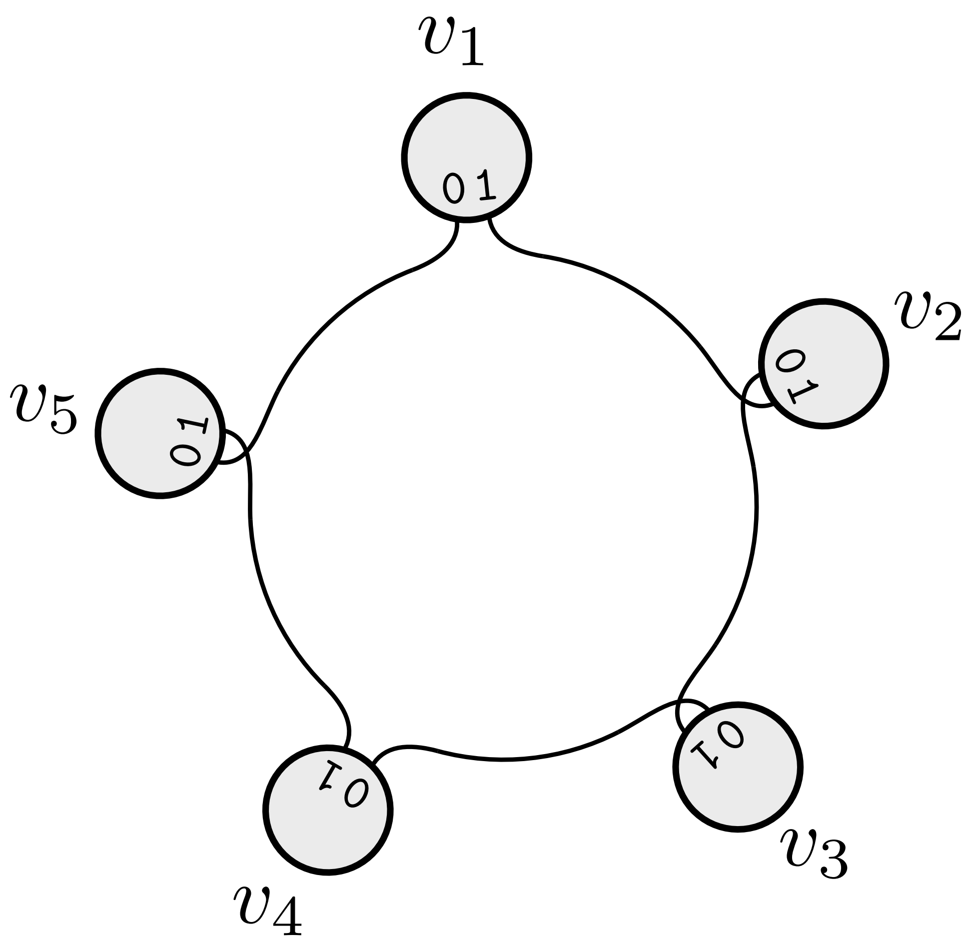


- Can we remove this assumption?

Orienting a ring

- Observation: Forwarding a signal goes along the cycle, even if the ring is un-oriented.





- Orienting a ring: send your ID to CW (propagate surplus), convert if you see more signals in the other direction
- Can achieve both Leader Election and Orientation at the same time

Content Oblivious Ring Orientation

- **Theorem:**
  Content-Oblivious Ring Orientation (+Leader Election) is possible in Rings

[Frei, Ghazi, Gelles, Nolin, DISC'24]

- Complexity: $n(2\mathrm{ID}_{\max} + 1)$
- Non-terminating! But reaching quiescence
  - We suspect this task does not have a terminating algorithm, without further assumptions

Content Oblivious

General Compilers

Content-Oblivious Computation

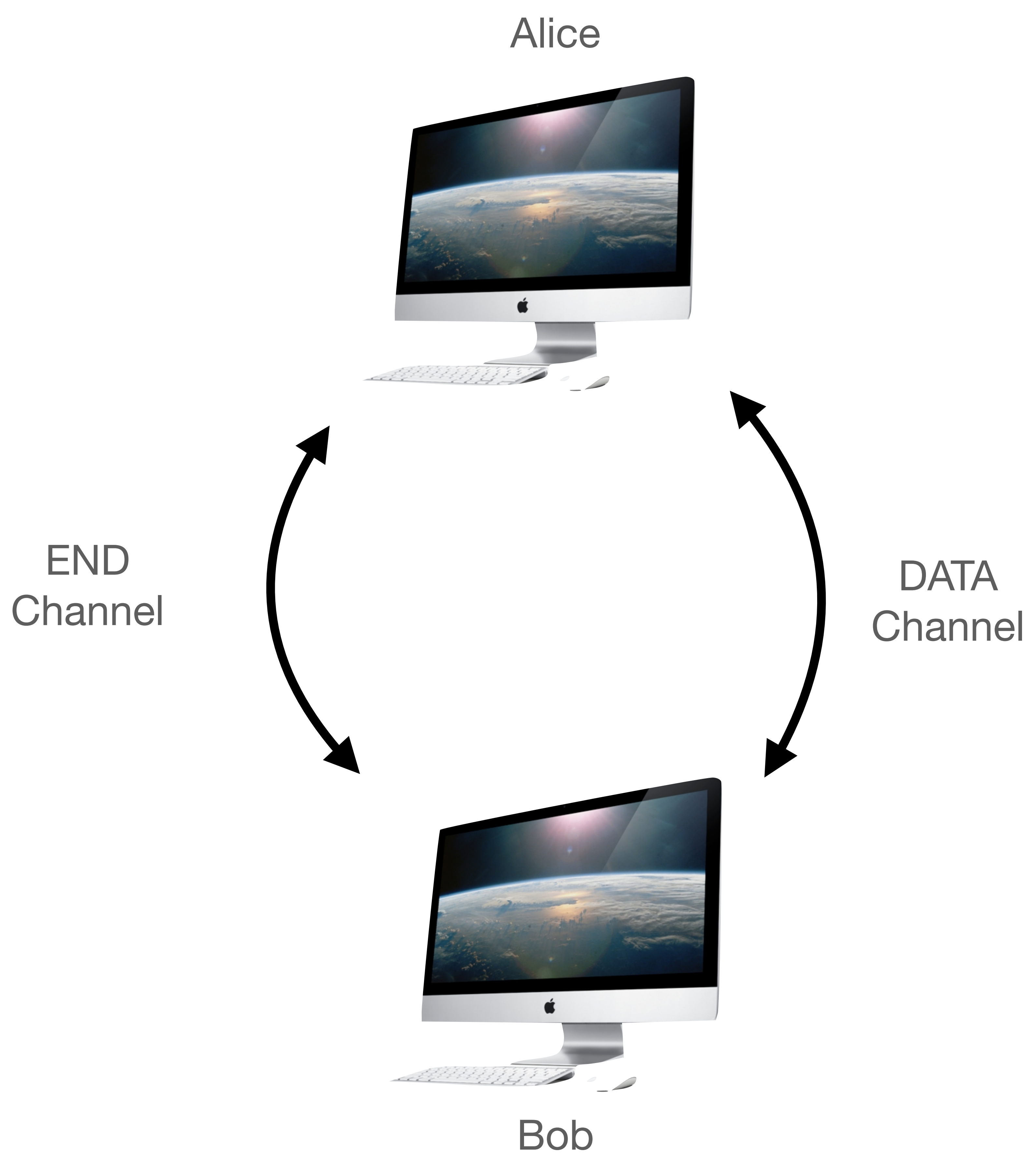
Theorem:
Any communication protocol Π can be simulated over any 2-edge connected network G, in a content-oblivious way

with $poly(n)$ overhead per bit of Π,   assuming a "root"

[Censor-Hillel, Cohen, Gelles, Sela, 23]

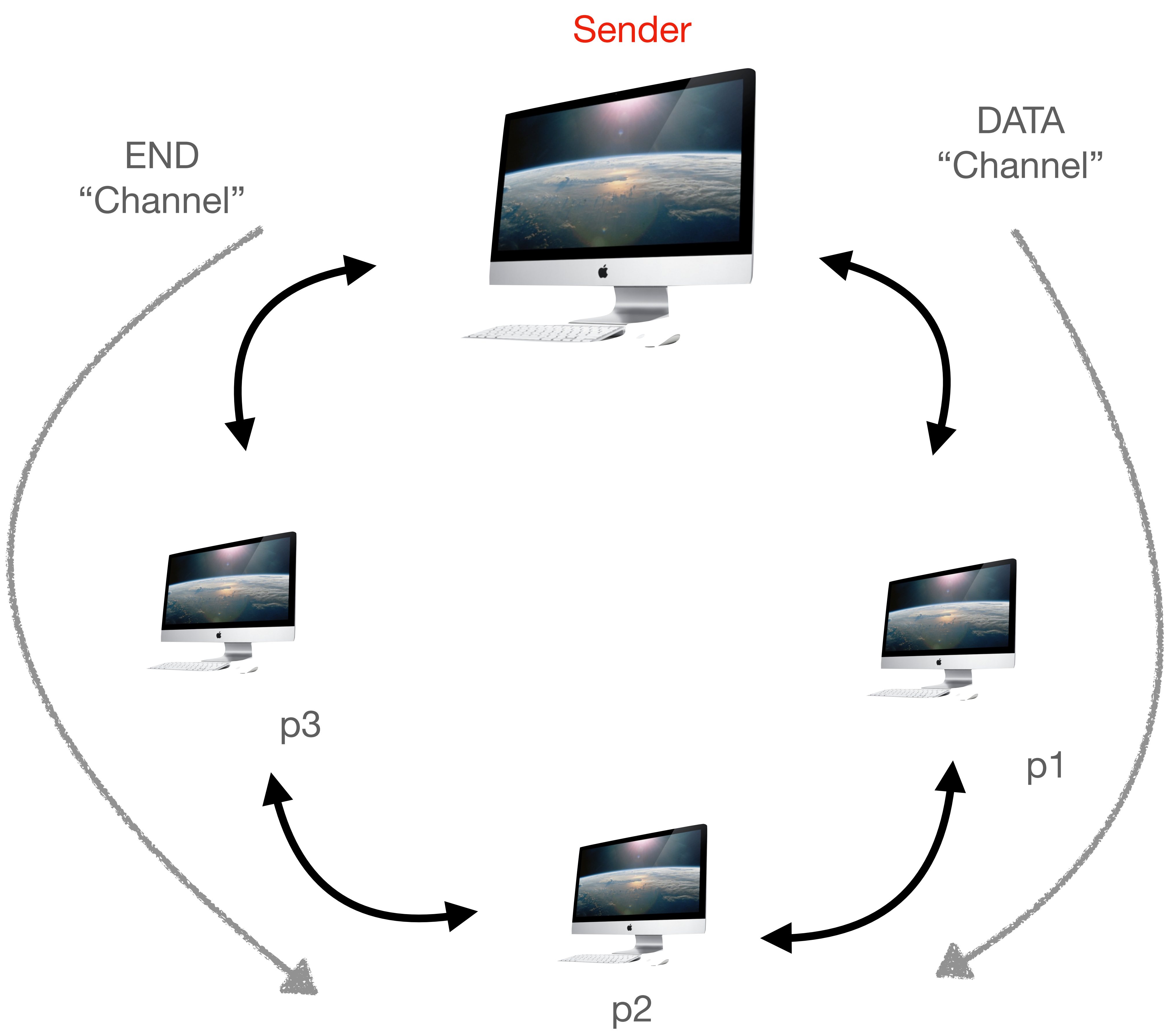
Idea: two channels

- Assume we have two channels:
  - DATA channel -
    Unary encoding of the information
    (1 message per symbol)
  - END channel -
    marks the end of the transmission
    (a single message)
- Each message must be acknowledged otherwise, END might be wrong
- END also changes parties' roles



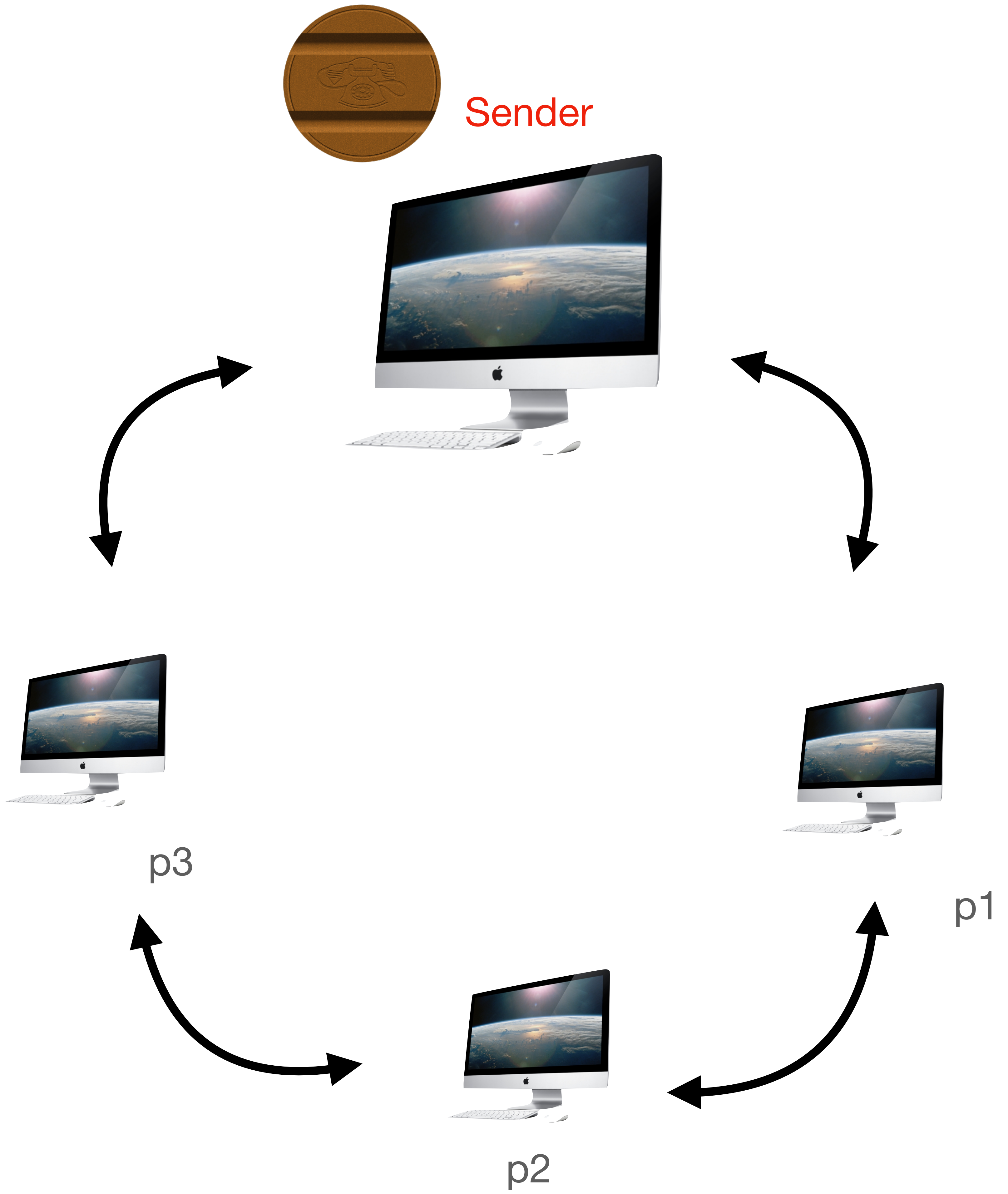
Content-oblivious comm. in simple cycles

- Extension to a cycle is possible as long as there is a single sender
  - Nodes relay any received message
  - Information is carried out by direction:
    - Clockwise: DATA
    - CounterClockwise: END
- Overhead: O(n) per (unary) symbol



Communication over a Fully-Defective Cycle

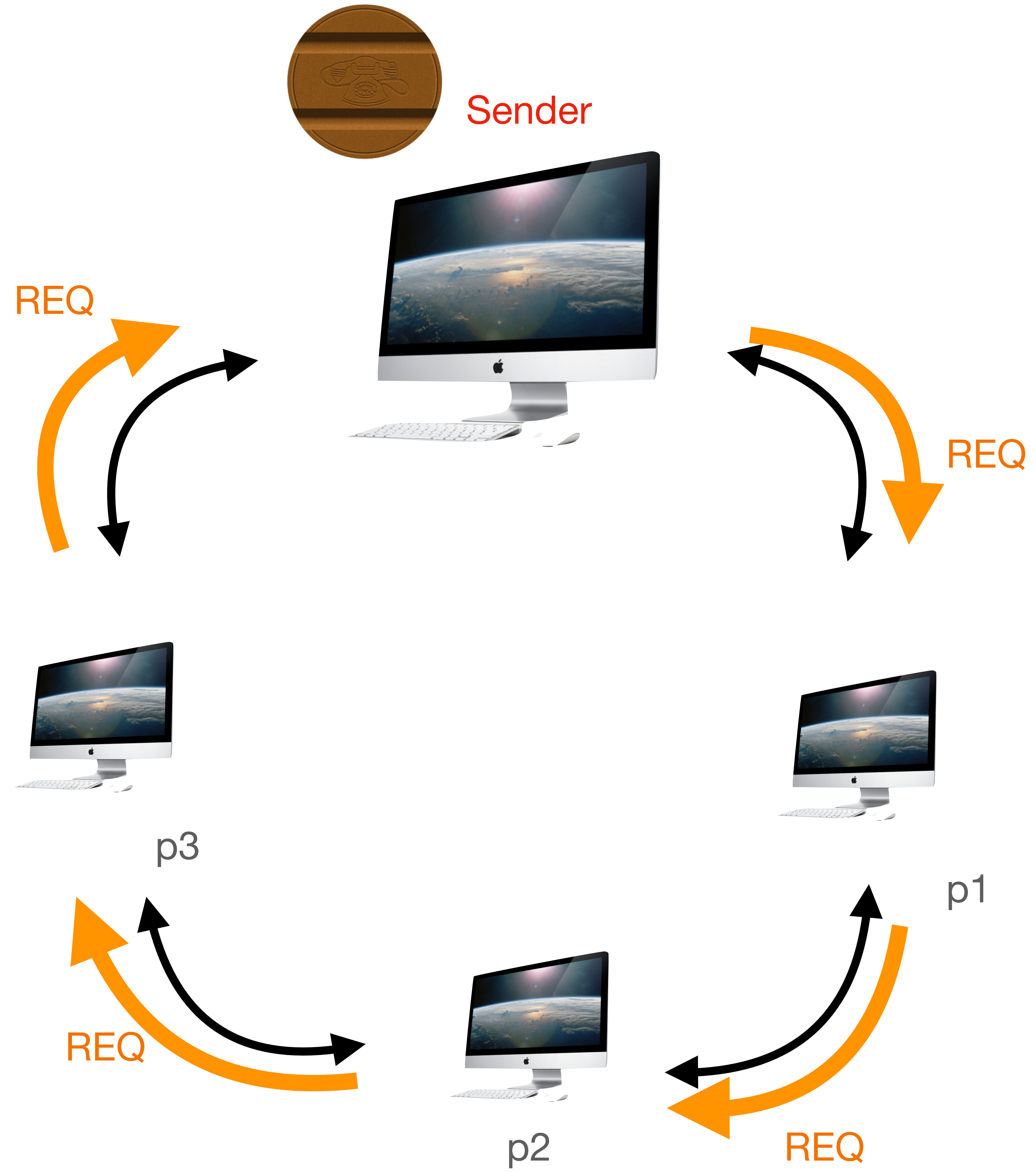
- What if another node wishes to speak?
- TOKEN exchange mechanism:
  - after an END message, meaning of messages changes:
    - **Clockwise**: request for token
    - **CounterClockwise**: TOKEN



Communication over a Fully-Defective Cycle

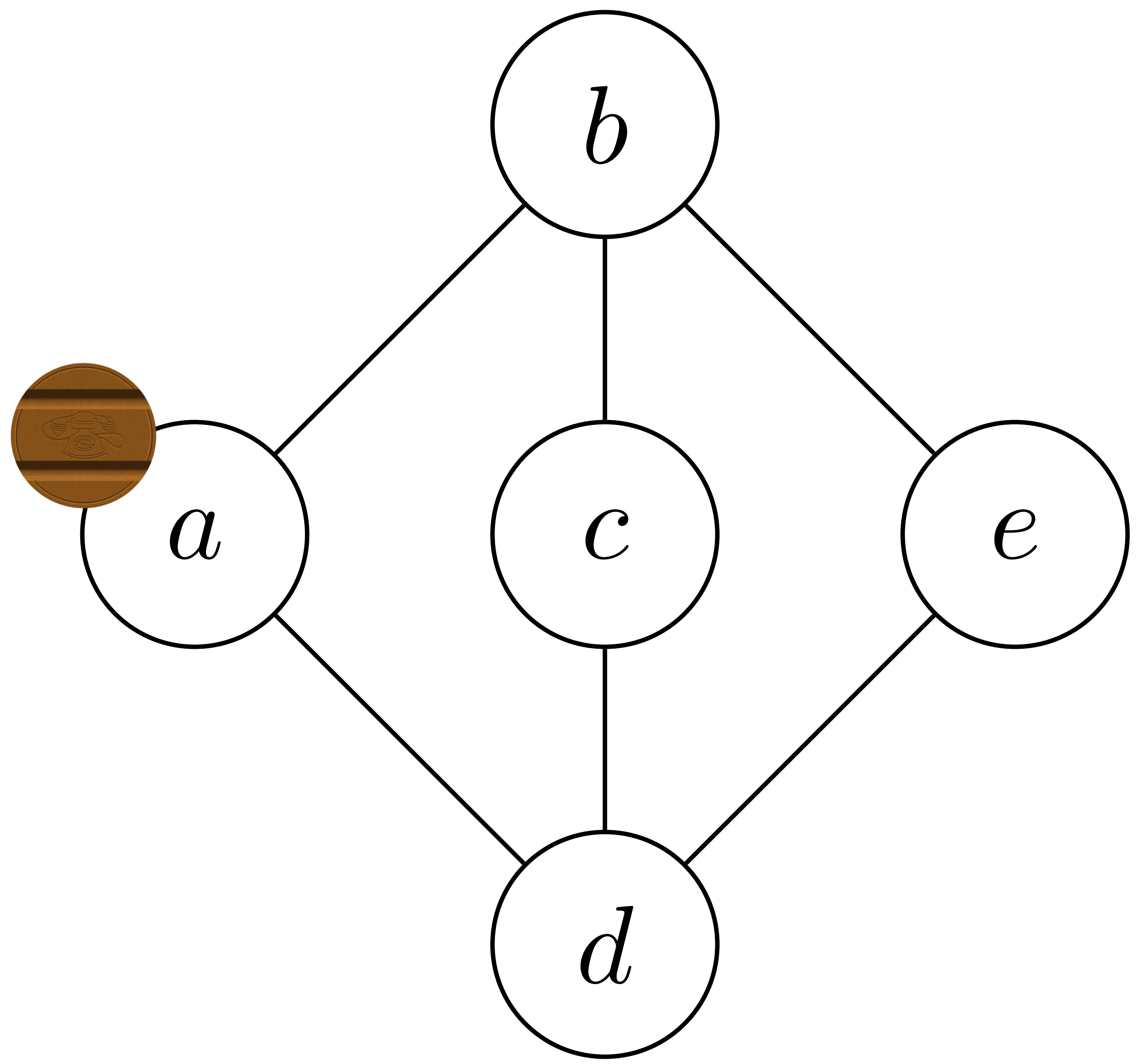
Token Mechanism

- Request (REQ):
  - Nodes request token asynchronously
  - **Invariant**: every node must send 1 REQ and receive 1 REQ before continuing
- Once the current sender sent and received REQ, it releases the token (TOK)
  - If TOK reaches node that wants the token, it becomes the new sender,
- sender initiates communication (sends DATA) (triggers other to quit TOKEN phase)



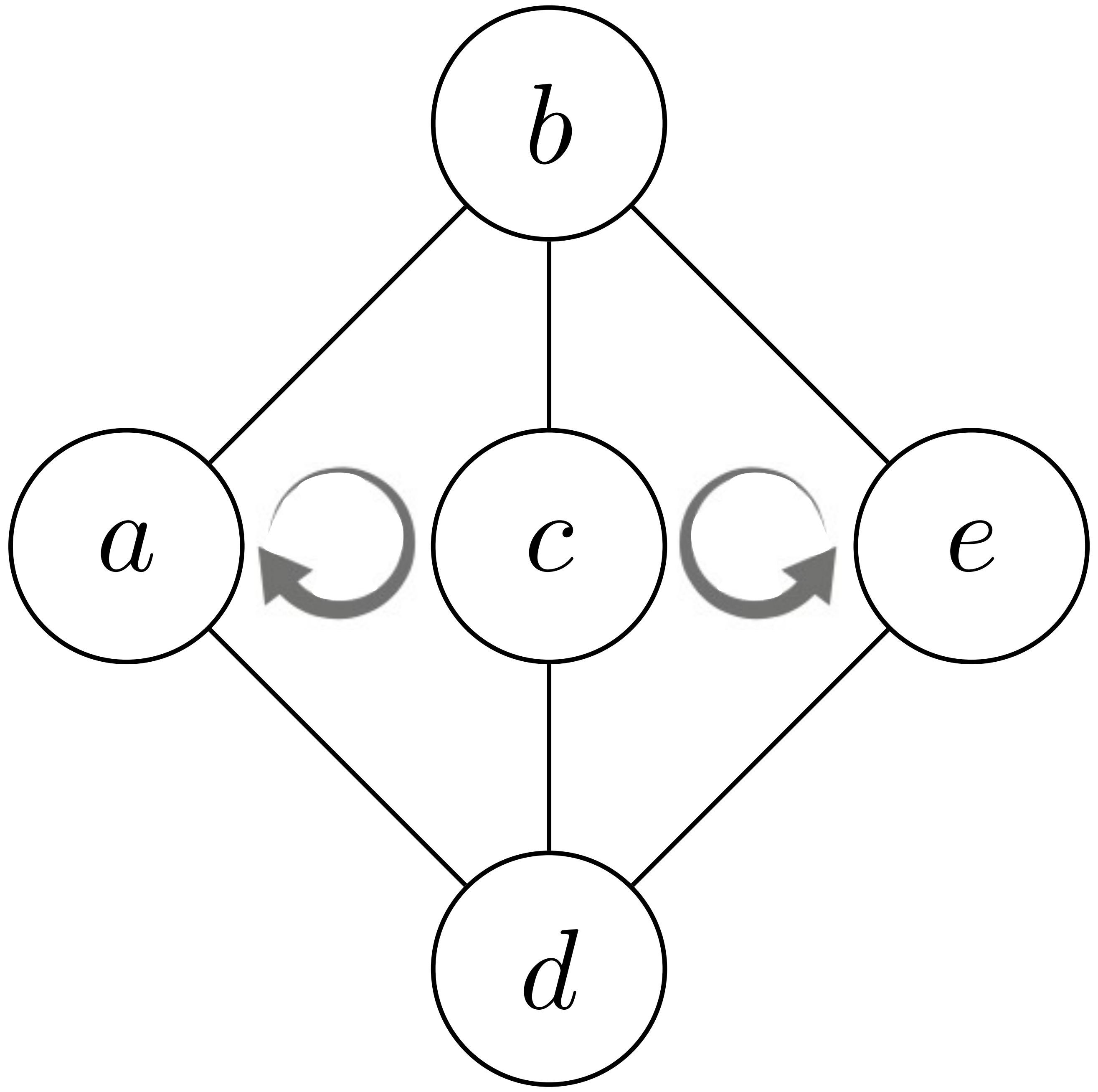
The General Case

- How to communicate over arbitrary 2-edge connected graphs?



The General Case

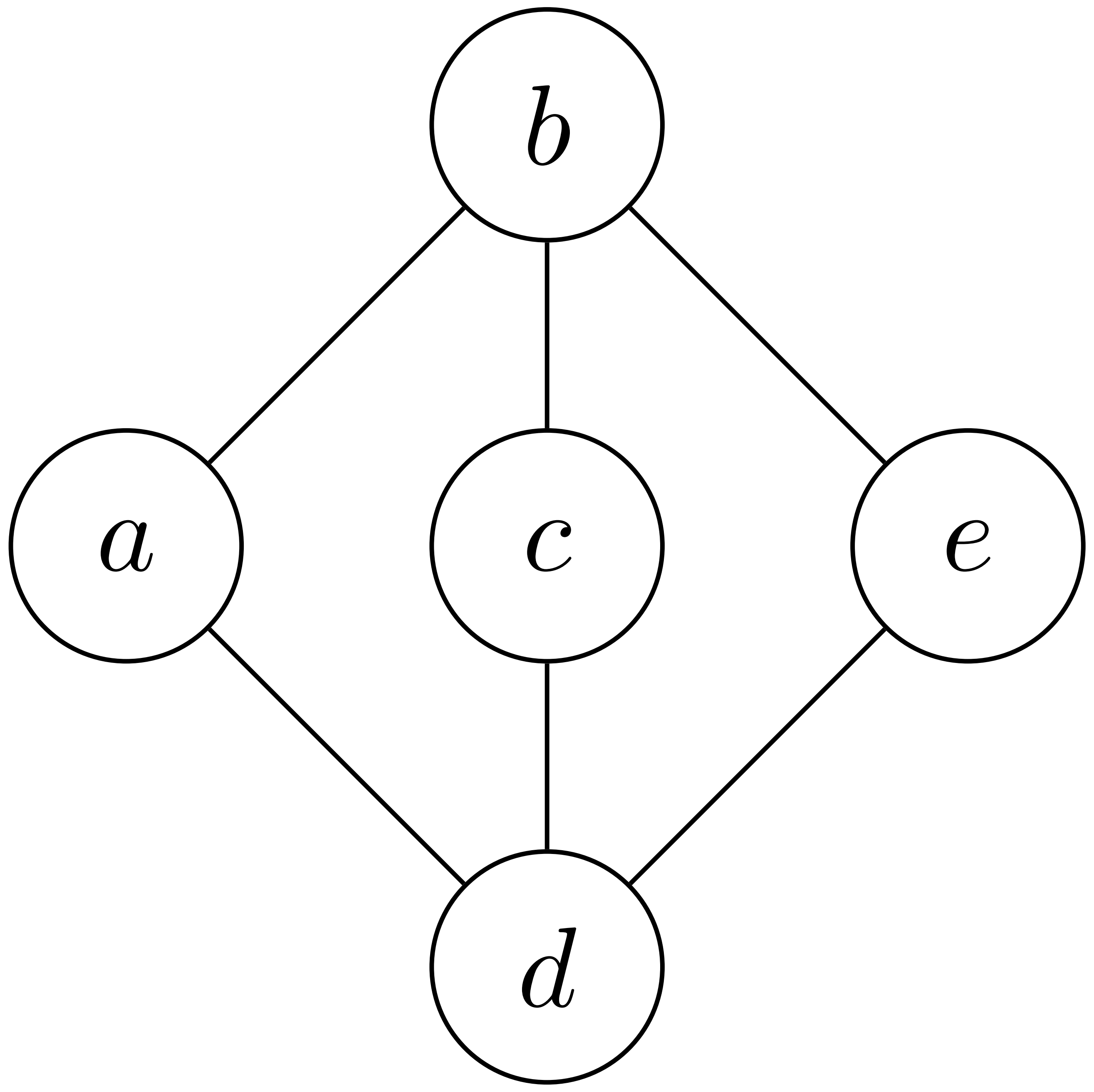
- How to communicate over arbitrary 2-edge connected graphs?



- Combining non-disjoint cycles?
  - When d gets a message, where should it propagate it to?
- How to construct the cycles?

The General Case

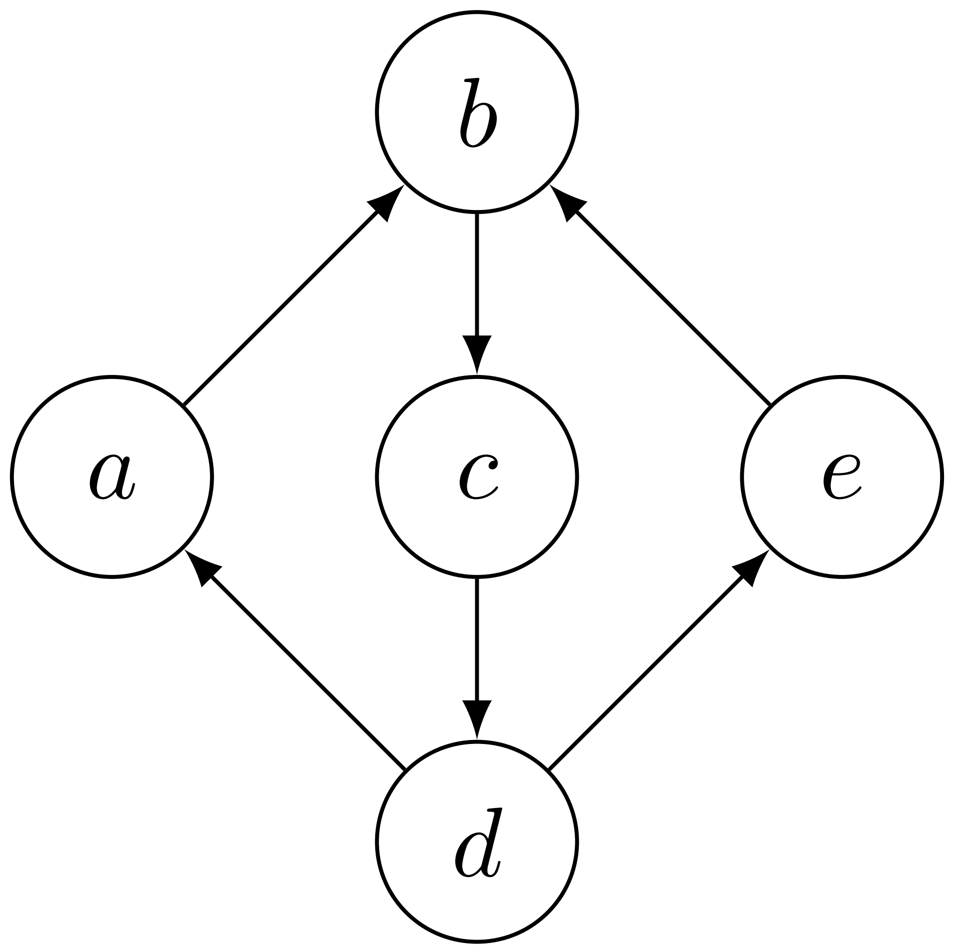
A Robbins Cycle

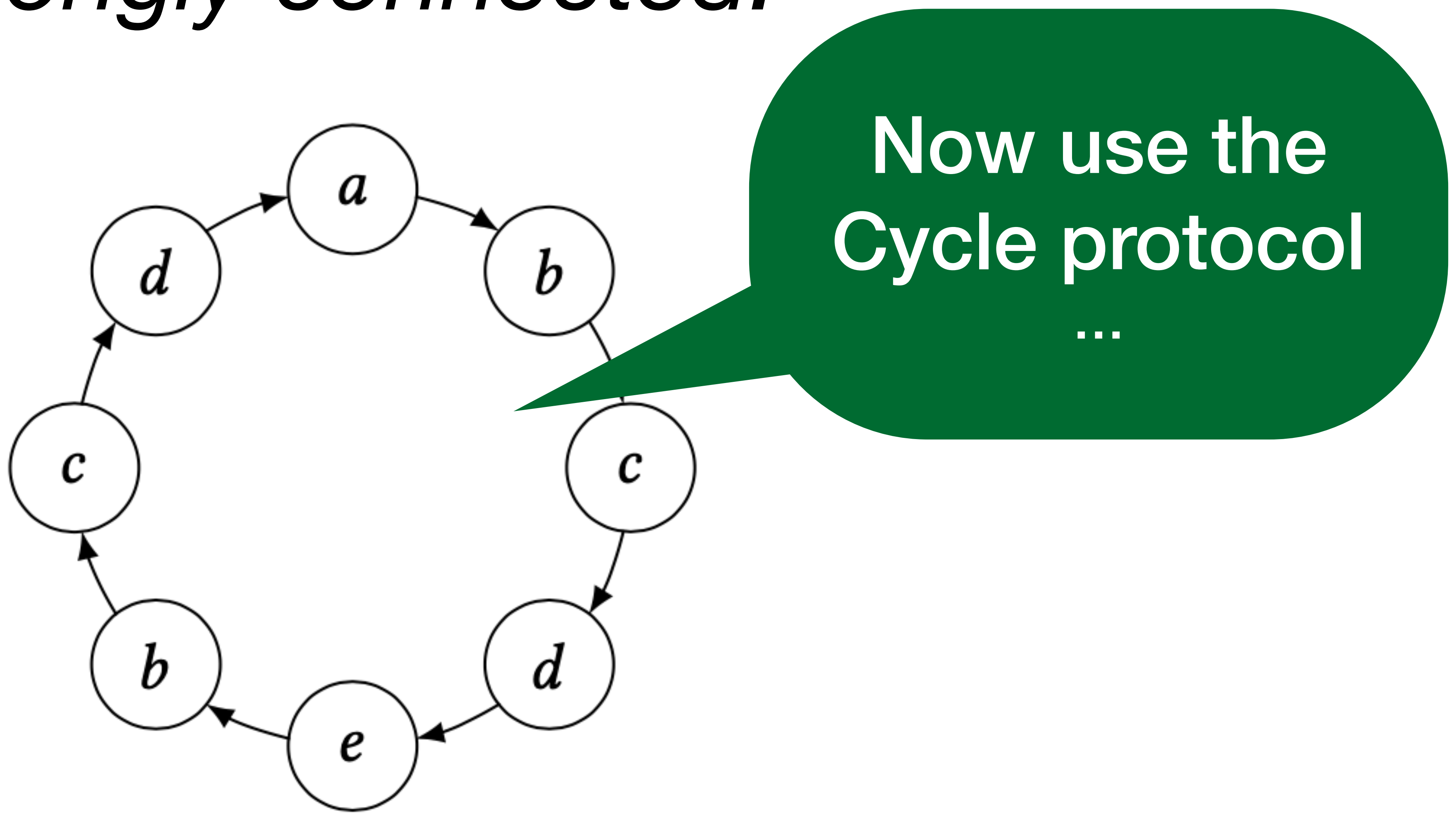


- **Theorem** [Robbins'39]

  every 2-edge-connected graph is **orientable:**

  there exists a way to orient all the edges so that the yielded directed graph is *strongly connected.*





The General Case

Constructing a Robbins Cycle

- But, how can we construct this orientation (content-obliviously)?
- **Ear-Decomposition Theorem** [Whitney'32]:
  any 2-edge-connected graph can be decomposed into

$$G = C_0 \cup E_1 \cup E_2 \cup \cdots \cup E_k$$

with

C_0 being a simple <u>cycle</u> and

E_i being a simple <u>path</u> whose endpoints belong to $C_0 \cup E_1 \cup \cdots \cup E_{i-1}$

Content-Oblivious Robbins Cycle Const.

Theorem:
Suppose one of the nodes is a designated root.
Then, there exists a content-oblivious Robbins-Cycle construction algorithm (via ear-decomposition)

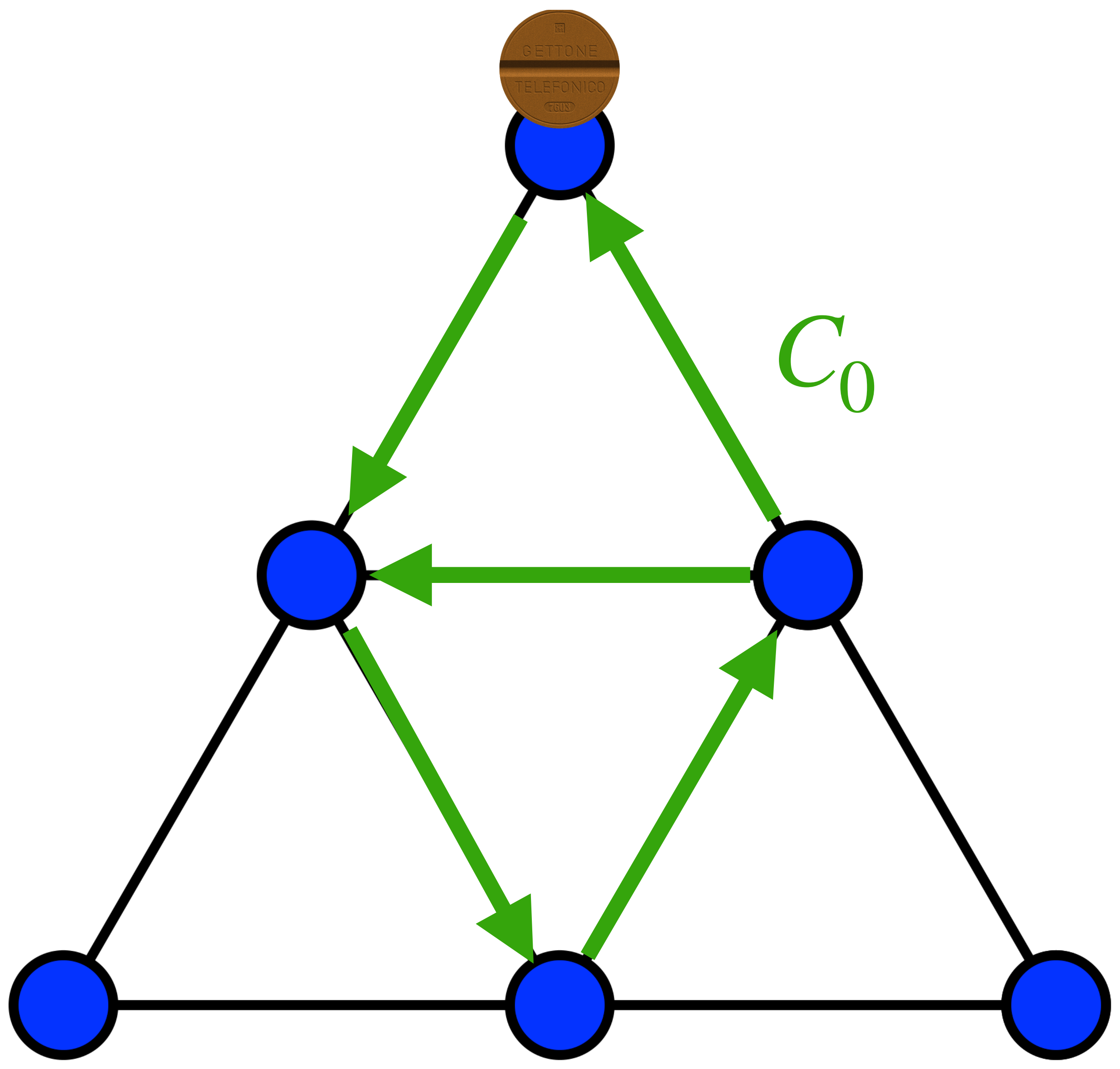
[Censor-Hillel, Cohen, Gelles, Sela, Dist.Comp'23]

- Complexity: $O(n^8)$

Ear decomposition

Constructing C_0

- The construction begins at a designated node $root$
- Nodes propagate a token in a DFS-like manner:
  - forward the token to an unused edge
  - if no unused edge **or** if reached $u \neq root$ twice
    -> send token back to parent (“retract”)
  - **until** the token reaches $root$ again
- Non-retracted edges form the cycle C_0
  Since 2-edge connected, $root$ will be reached again



Ear decomposition

Constructing E_1

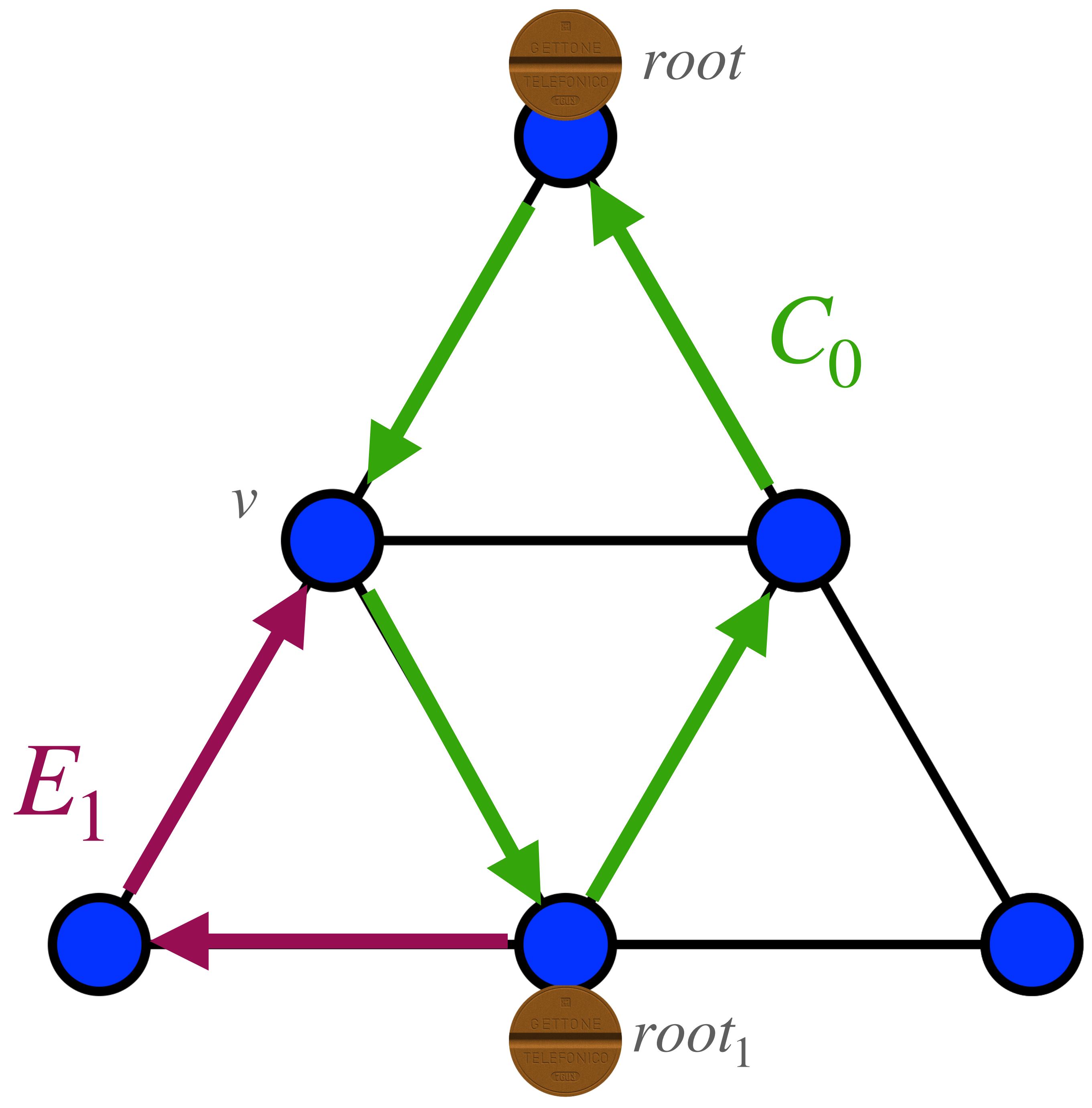
- C_0 is a simple cycle,
  its nodes can run the Cycle communication protocol

- If some $u \in C_0$ has unexplored edges, it requests to be the next root ($root_1$)

  - Begin a new DFS-exploration (on $E \setminus C_0$)
    until hitting a node already on the cycle

  - $C_0 \cup E_1$ form a (non-simple) cycle C_1 :
    ($root \to root_1 \to v \to root$)

- The decomposition then recurses on $E \setminus C_1$



Summary
and Open Questions

Summary

- **Content Oblivious Computation**
  - means of communication in networks of “simple” devices
  - fault tolerance towards potential content corruption / noise

- Some tasks can be done, under different assumptions
  - BFS/DFS (leader, knowledge of n)
  - LE (ring topology)
  - general communication (2-edge connectivity, leader)
  - In **non** 2-edge-connected networks, impossibility result if nodes give output

Open Questions

- What are the minimal assumptions for content-oblivious computation?
  - assumptions for termination?
  - weaker notions of termination?   (e.g., stabilization, finalizing outputs)
- Can we deal with insertions and deletions of signals?
  - even a tiny amount?

Open Questions

- Efficiency and Overhead?
  - BFS $O(n^3)$
  - Leader Election $O(n \max ID)$
    lower bound $\Omega(n \log(\max ID/n))$. Can we show $\Omega(n^2)$?

[Diniz,Moran,Rajsbaum '07]
[Frei, Ghazi, Gelles, Nolin '24]

  - General Compiler $\approx O(n^8) + O(n^3 \cdot \mathrm{CC}_{Alg})$

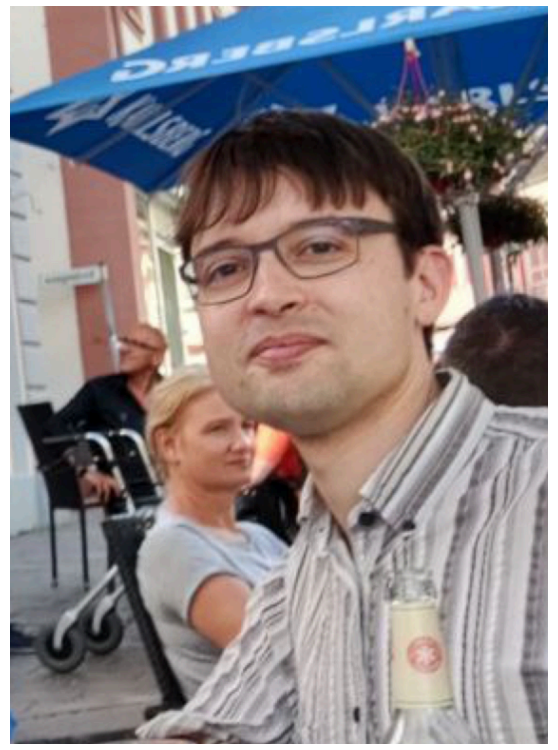
Thanks to my co-authors!!

Keren Censor-Hillel

Shir Cohen

Gal Sela

TECHNION
Israel Institute of Technology



Fabian Frei

Ahmed Ghazi

Alexandre Nolin

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY