# Leader Election in the Ring (Part 2)



Evangelos Kranakis, Carleton University, SCS (October 3, 2020)

## Outline



- Non-Comparison Based Algorithms
  - Time Slice
  - Variable Speeds
- Lower Bounds
- Randomized
  - identity selection
  - Leader election

## Comparison of Leader Election Algorithms



- Resulting Tradeoffs



| Algorithm | Rounds | Time | # Messages |
|---|---|---|---|
| TimeSlice | $O(u_{\min} n)$ | $O(u_{\min} n)$ | $O(n)$ |
| VariableSpeed | $O(2^{u_{\min}} n)$ | $O(2^{u_{\min}} n)$ | $O(n)$ |
| Randomized | $O(\log n)$ | $O(n)$ | $O(n \log n)$ |





where $u_i$ denotes the $i$-th node's identifier and $u_{\min}$ the minimum identifier among the node identifiers.

# Non-Comparison



# Based

## Time Slice Algorithm



- Uses the *strong* assumption that
  - the ring size $n$ is known to all the processes (non-uniform).



- It assumes unidirectional communication.
- It elects the process with the minimum user ID.



- Assumes processor IDs are natural numbers.
  - Each process $i$ has the ID $u_i$ (unknown to the rest of the processors).





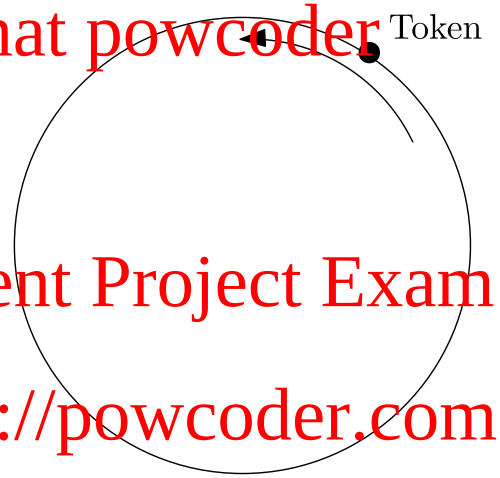
- Employs synchrony in a deeper way in that
  - it uses a token to convey information.

## Time Slice Algorithm: Searhing for IDs!



- Employs a *circulating token*, carrying an ID around the ring.





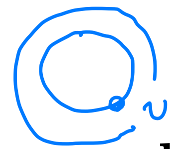



- Let $v$ denote the phase. Starting from $v = 1$ and each phase incrementing by 1, it attempts to elect $v$ as leader.
- For $v = 1, 2, \ldots$: in phase $v$ only a token carrying ID $v$ is permitted to circulate.
- Processor IDs are unknown to other processors (and can be large numbers).

## Time Slice Algorithm

1. Computation proceeds in phases $1, 2, \ldots$ and each phase consists of $n$ consecutive rounds (we use $n$ is known).

2. Each phase devoted to the possible circulation, all the way around the ring, of a token carrying a particular value. Nodes check if their ID is equal to the value of the token.




3. In **phase** $v$, which consists of rounds $(v-1)n+1, \ldots, vn$, **only a token carrying value** $v$ is permitted to circulate.

4. **If a process** $i$ **with** $ID$ **equal to** $v$ **exists**, and round $(v-1)n+1$ is reached then process $i$ elects itself the leader and sends a token carrying its ID around the ring.

5. As this token travels, all the other processes note that they have received it, which prevents them from electing themselves as leader or initiating the sending of a token at any later phase.

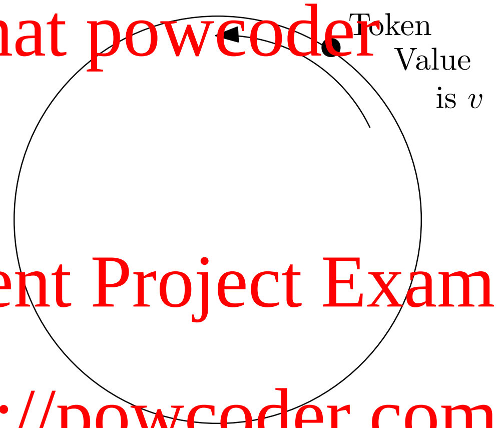
i.e., you assume the nodes have MAC addresses: in the Network Card

## Example of Time Slice Algorithm



- A token **carrying a certain value** circulates around the ring.








1. Phase $v = 1$: Token carrying ID equal to 1 circulating around the ring; nodes check if their ID is equal to 1;



2. Phase $v = 2$: Token carrying ID equal to 1 circulating around the ring; nodes check if their ID is equal to 1;
3. etc.

- Note that in each phase, it takes $n$ steps for the token to go around the ring.

## Correctness of Time Slice Algorithm



- The minimum ID $u_{\text{min}}$ eventually gets all the way around, which causes its originating process to become elected.



  – No messages are sent before round $(u_{\text{min}} - 1)n + 1$, and
  – no messages are sent after round $u_{\text{min}} n$.



- The total number of messages sent is just $n$.



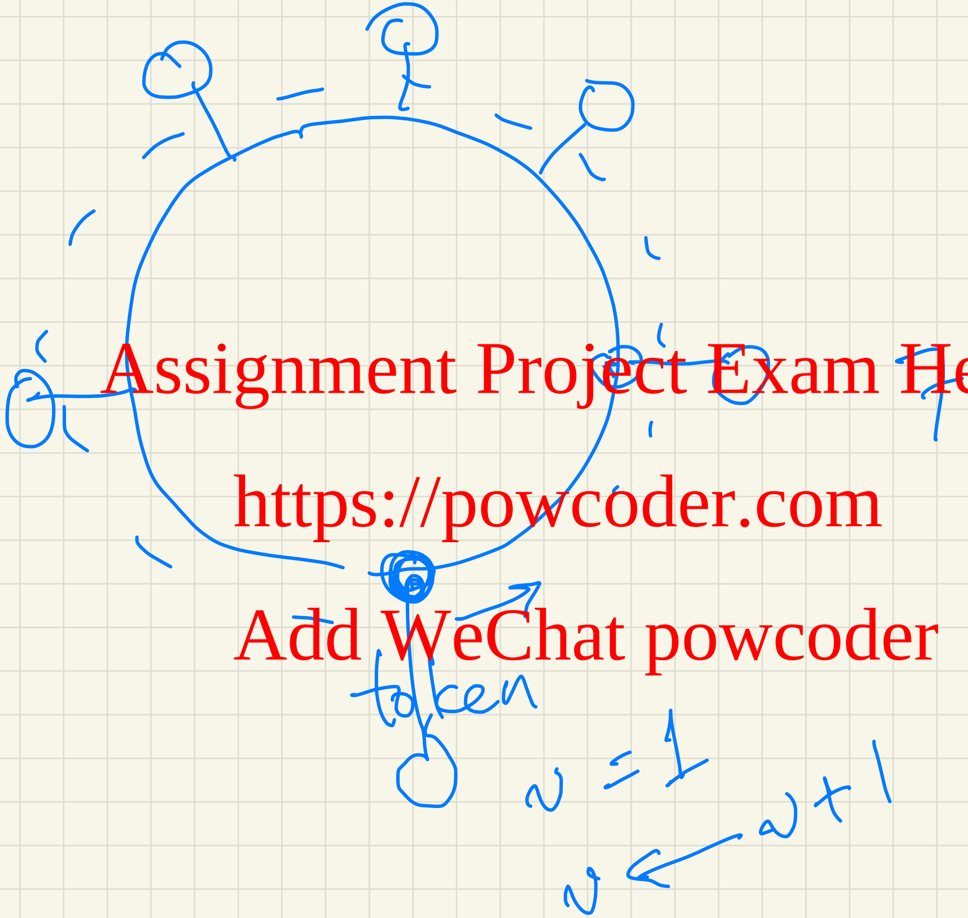
  – These are the non-null messages of the node with minimum ID $u_{min}$ claiming to be the leader.



- If we prefer to elect the process with the maximum ID rather than the process with the minimum, we can simply let the minimum send a special message around after it is discovered in order to determine the maximum.

$u_{\text{min}} \cdot n$

- The communication complexity is still $O(n)$.


token
n = 1
n ← n+1

## Correctness of Time Slice Algorithm



- The good property of the TimeSlice algorithm is that the total number of messages is $n$.



- Unfortunately, the time complexity is about $nu_{\min}$, which is an unbounded number, even in a fixed-size ring.



- This time complexity limits the practicality of the algorithm.



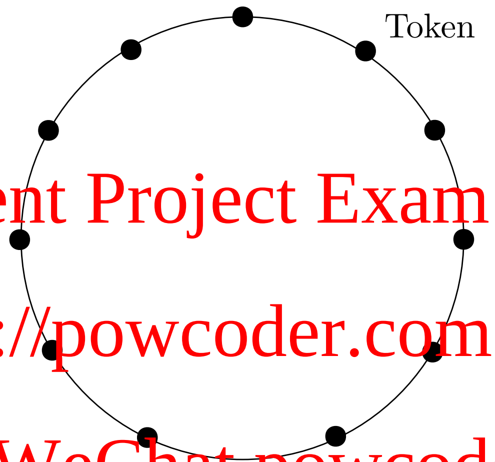
- It is only useful in practice for small ring networks in which IDs are assigned from among the small positive integers.

## Variable Speed Algorithm



- Employs *circulating tokens* (as many tokens as nodes) carrying certain IDs.








- Each process $i$ has the ID $u_i$ (unknown to the rest of the processors).
- Process $i$ initiates a token, which travels around the ring, carrying the ID $u_i$ of the originating process $i$.
- Different tokens travel at different speeds.

## Variable Speed Algorithm



1. Each process $i$ initiates a token, which travels around the ring, carrying the value $u_i$ of the ID originating process $i$.



2. Different tokens travel at different speeds.
   - A token with value $v$ travels at the speed of one message transmission per $2^v$ rounds, that is, each process along its path waits $2^v$ rounds after receiving the token before sending it out. /* a token with value $v$ takes time $n2^v$ to circulate around the ring (if it does) */



3. Each process keeps track of the smallest value it has seen so far and simply discards any token carrying an identifier that is larger than this smallest one.
4. If a token returns to its originator the originator is elected leader.

## Correctness of Variable Speed Algorithm



- Algorithm guarantees that



  - by the time the token carrying the smallest identifier $u_{\min}$ gets all the way around the ring,
  - the second smallest identifier could only get at most halfway around, the third smallest could only get at most a quarter of the way around, and in general,




  - the $k$th smallest could only get at most $1/2^{k-1}$ of the way around.



- Therefore, up to the time of election, the token carrying $u_{\min}$ uses more messages than all the others combined.
- Since $u_{\min}$ uses exactly $n$ messages, the total number of messages sent, up to the time of election, is less than $2n$.

## Correctness of Variable Speed Algorithm



- By the time $u_{\min}$ gets all the way around the ring, all nodes know about this value, and so will refuse to send out any other tokens.



- It follows that $2n$ is an upper bound on the total number of messages that are ever sent by the algorithm (including the time after the leader output).



- The time complexity, as mentioned above, is $n2^{u_{\min}}$, since each node delays the token carrying ID $u_{\min}$ for $2^{u_{\min}}$ time units.

# Leader Election Lower Bound

## Comparison Based Algorithms



- An algorithm is comparison based if it behaves the same on rings that have the same order pattern of the identifiers.
  - in two rings with corresponding processors $p_1, p_2, \ldots, p_n$ and $q_1, q_2, \ldots, q_n$ the actions of the algorithm depend only on the order of the identifiers $ID(p_1), ID(p_2), \ldots, ID(p_n)$ and $ID(q_1), ID(q_2), \ldots, ID(q_n)$, respectively.
- We have seen that the best comparison based algorithm achieves the following bounds:
  - communication complexity of $O(n \log n)$ messages, and
  - time of $O(n)$.

Radius Growth

## Non-Comparison Based Algorithms



- Non-comparison based algorithms, on the other hand, use $O(n)$ messages, but (can) have a huge running time.



- A lower bound of $\Omega(n \log n)$ messages can be shown for

  1. *comparison based* algorithms; lower bound holds even if we assume that communication is bidirectional and the ring size $n$ is known to the processes.

  2. *non-comparison* based algorithms: with bounded time complexity (i.e., bounded number of rounds).[a]

- In the sequel we discuss only the first bound for comparison based algorithms.[b]

---

[a] This can be proved using Ramsey Theory.

[b] Greg N. Frederickson and Nancy A. Lynch. Electing a leader in a synchronous ring. Journal of the ACM, 34(1):98-115, January 1987.

## The Plan: What Are We Going to Do?



- Assume we are given a uniform ($n$ is unknown) algorithm $\mathcal{A}$ that solves the above comparison based variant of the leader election problem.



- We will show that there exists an admissible execution (i.e., an execution that conforms to the model being considered) of $\mathcal{A}$ in which $\Omega(n \log n)$ messages are being sent.
  - So not all executions will satisfy this $\Omega(n \log n)$ lower bound condition!



- **Theorem 1** *For any comparison based leader election algorithm on a ring of size $n$ there is an execution of the algorithm in which $\Omega(n \log n)$ messages are being sent.*

Rad Gro. $O(n \log n)$

## How Do You Prove a $\Omega(n \log n)$ Lower Bound?



- The quantity being considered is the following: $M(n)$ = "the number of messages required to elect a leader in an $n$-node ring".



- This means that we must find a constant $c > 0$ independent of $n$ such that



$$M(n) \geq cn \log n,$$



for all $n$.

we do not really care about c>0



c independent of n

- But how do we accomplish this task?
  - Using a recurrence!

## A Simple Recurrence



- We will show that



$$M(n) \geq 2M(n/2) + n/4 \tag{1}$$

This is what we have to prove

- What does this mean?



- Lets denote our problem $LE(n)$:
  - Leader Election in a ring of size $n$.



- One way to interpret Inequality (1) is the following:



  1. Split $LE(n)$ into two subproblems $LE(n/2)$.
  2. Show that the number of messages required to solve $LE(n)$ is at least the number of messages required to solve two $LE(n/2)$ problems plus $n/4$.

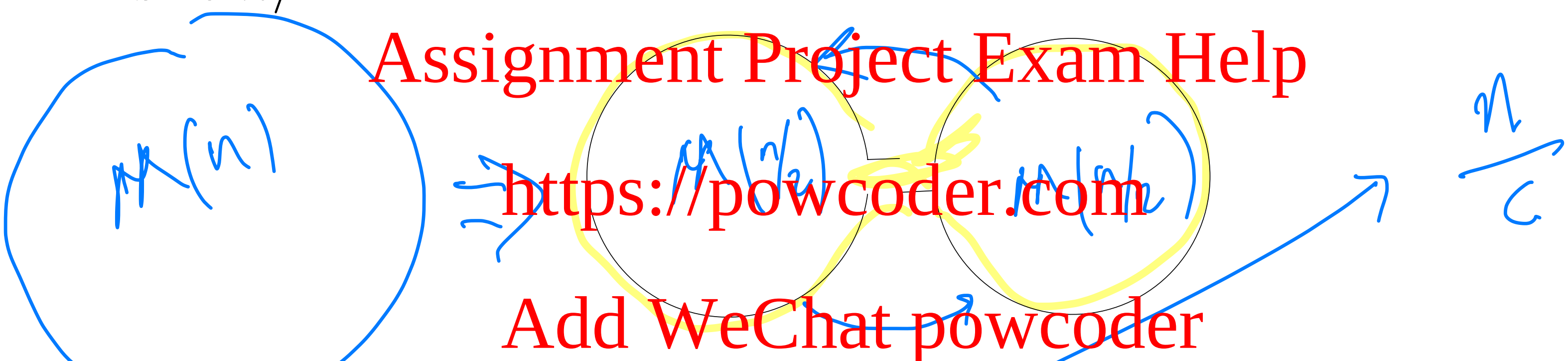
## What Does it Mean to Split?



- From our definition $M(n/2)$ = "the number of messages required to elect a leader in an $n/2$-node ring".



- So we need to split the ring of size $n$ into two subrings each of size $n/2$:

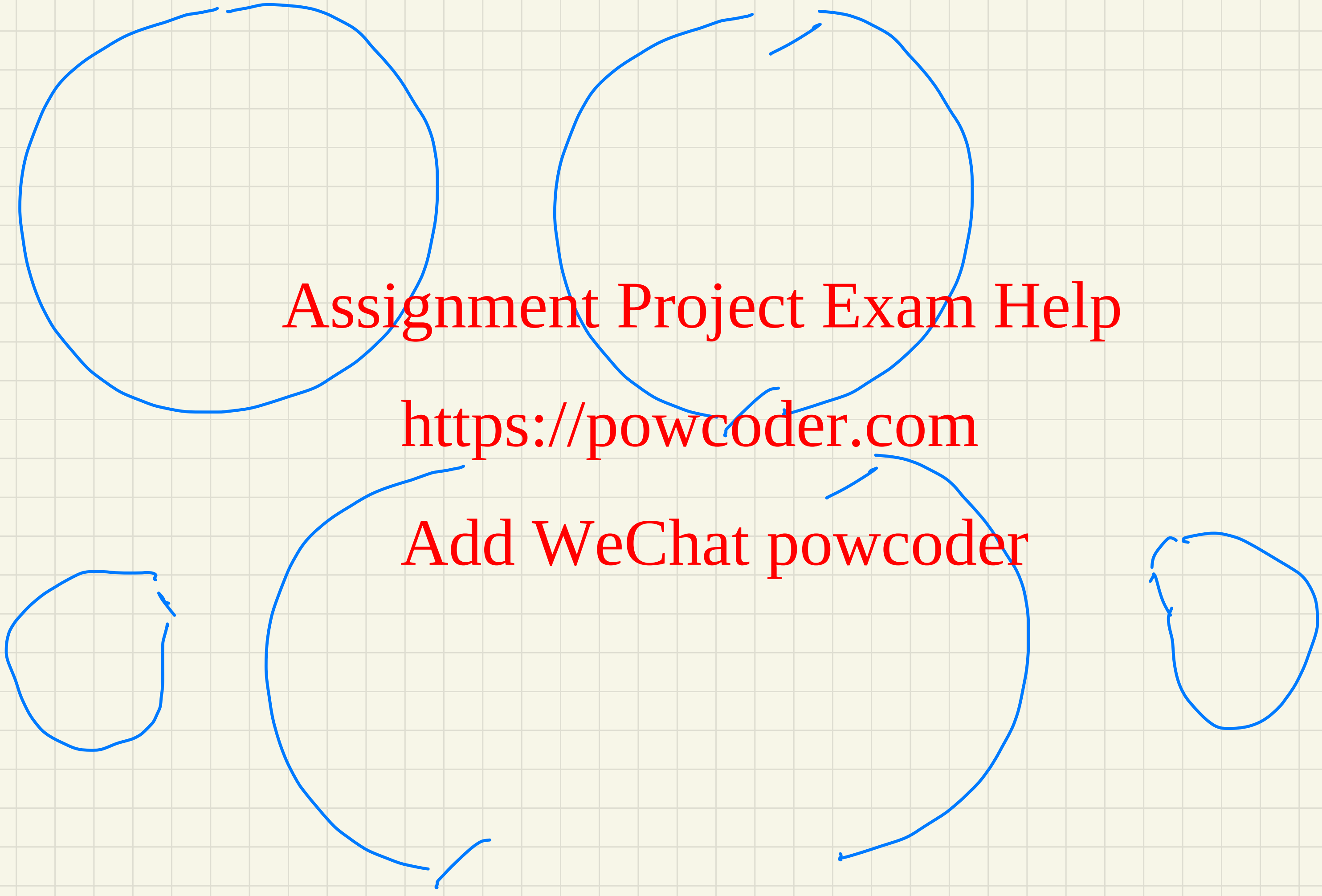





  – The term $2M(n/2)$ in Inequality (1) should come from this two subrings in this splitting!

- How about the term $n/4$ in Inequality (1)?
  – The term $n/4$ in Inequality (1) should come from messages which are in transit from one subring to the other and needed to elect a leaser! (Discussed later.)

$\cancel{\log(n/c)}$

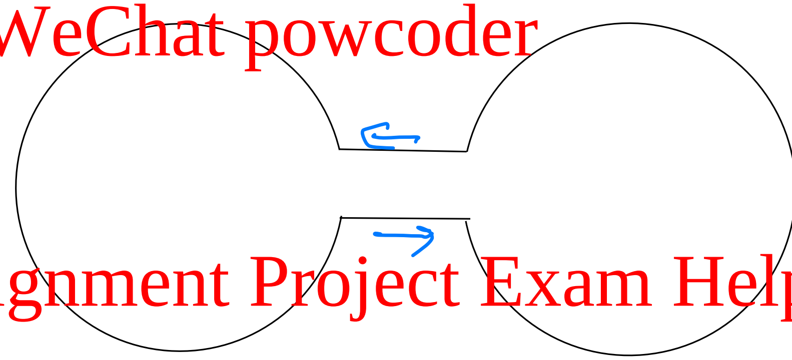


**Glueing!**



- How do you glue two subrings into a bigger ring?







- You must find two respective schedules in the subrings of size $n/2$ which send at least $M(n/2)$ messages in each subring and also leave at least one edge unused!
  - You need this so that you can do the glueing!



- We will call such schedules which leave an edge of the ring unused *open*
  - *open* because they leave an edge unused.

## **Inequality** (1) **Implies** $M(n) \geq \frac{n}{4}(\log n + 1)$

$c = \frac{1}{4}$



- For simplicity, assume $n$ is a power of 2.



- Assume that $M(n) \geq 2M(n/2) + n/4$ has been proved[a]
- By induction: we will prove
  - Base case:



$$M(2) \geq \frac{2}{4}(\log 2 + 1) = 1$$

$\frac{2}{4} \cdot 2 = 1$

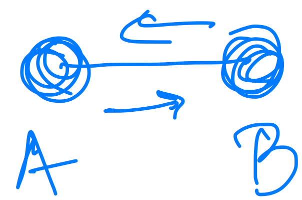


  - Inductive Case:

    Assuming the inductive assumption



$$M(n/2) \geq \frac{n}{8}\left(\log \frac{n}{2} + 1\right)$$




we'll prove

$$M(n) \geq \frac{n}{4}(\log n + 1)$$

[a] We have not proved this yet! This is our goal in this lecture!

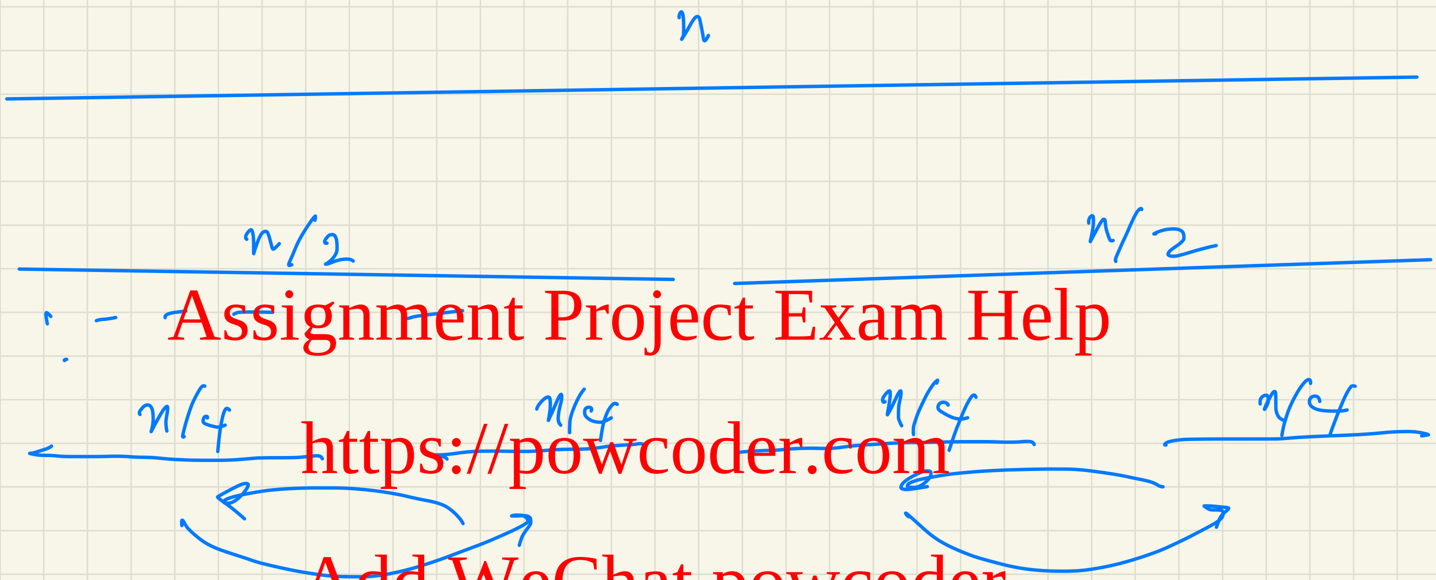
$$S(n) \le 2\,S\left(\frac{n}{2}\right) + \frac{n}{4}$$

$S(n)$ = # of steps

You don't know what the explicit formula for $S(n)$ is

Example Radius Growth satisfies this inequality

what is $S(n)$?






$$\boxed{S(n) \le \underbrace{2\,S(n/2)}_{} + \frac{n}{4}}$$

$$
\begin{aligned}
S(n) &\le 2S(n/2) + n/4 \\
&\le 2\left(2S(n/4) + n/8\right) + n/4 \\
&= 2^2 S(n/2^2) + 2 \cdot \frac{n}{4} \\
&= 2^2\left(2 \cdot S(n/2^3) + n/2^4\right) + 2 \cdot \frac{n}{4} \\
&= 2^3 \cdot S(n/2^3) + 3\,\frac{n}{4} \\
&\;\;\vdots \\
&= n \cdot S(1) + \log n \cdot \frac{n}{4}
\end{aligned}
$$

$$S(1) = 1$$

$$S(2) \le 2 \cdot S(1) + \frac{1}{4} \; ;$$



$$S(4) \le 2 \cdot S(2) + \frac{\cancel{2}\,4}{4}$$





$$S(8) \le 2 \cdot S(4) + \frac{8}{4}$$

$$\vdots \qquad\qquad \vdots$$

$+$

## Base Case



- Here we must show that



$$M(2) \geq \frac{2}{4}(\log 2 + 1) = 1$$

- Somehow this seems like a simple statement about rings of just 2 nodes.



  – Lets postpone it for a moment!

## Inductive Step



- Assumie the inductive assumption



$$M(n/2) \geq \frac{n}{8}(\log \frac{n}{2} + 1)$$

- Therefore



$$\begin{aligned} M(n) &\geq 2M(n/2) + n/4 \quad \text{(By Inequality (1))} \\ &\geq 2\left(\frac{n}{8}\left(\log \frac{n}{2} + 1\right)\right) + \frac{n}{4} \quad \text{(Inductive Assumption)} \\ &= \frac{n}{4}\log n + \frac{n}{4} \\ &= \frac{n}{4}(\log n + 1) \end{aligned}$$




$\log n - 1$

- Lets move now to the details! We are missing the proofs of
  - the base case, and
  - Inequality (1).

## Basic Concepts/Assumptions



- We prove the lower bound for a special variant of the leader election problem, where the elected leader must be the processor with the maximum identifier in the ring;



  - in addition, all the processors must know the identifier of the elected leader.[a]

$$\boxed{M(n) \geq 2M\left(\frac{n}{2}\right) + \frac{n}{4}}$$



- We only accept uniform algorithms where the node with the maximum identifier can be the leader.





- Additionally, every node that is not the leader must know the identity of the leader.

- Ring is asynchronous: nodes may wake up at arbitrary times (but at the latest when receiving the first message).

[a]Unless this is done, no leader has been elected.

## Rules of the Game



- Recall execution model



- Nodes wake up at the latest when receiving first message
- Algorithms must be uniform
- We assume we have an algorithm and show it cannot complete faster than $O(n \log n)$ time





- Algorithm needs to do this regardless of how messages are scheduled
  - And when nodes wake up
  - Otherwise it is not a solution
- But communication links must be FIFO

## Schedules



- An execution of a distributed algorithm is a list of events, sorted by time.



  – An event is a record (time, node, type, message) where type is "send" or "receive".



- A schedule $\sigma$ of $\mathcal{A}$ for a particular ring is open if there exists an edge $e$ of the ring such that in $\sigma$ no message is delivered over the edge $e$ in either direction;

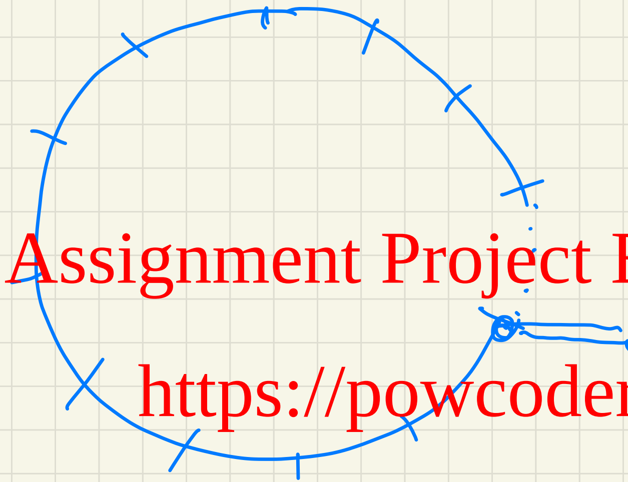


  – **Edge is open** if no message which is traversing the edge has been received so far.



  – **Schedule is open** if there is an open edge in the ring.

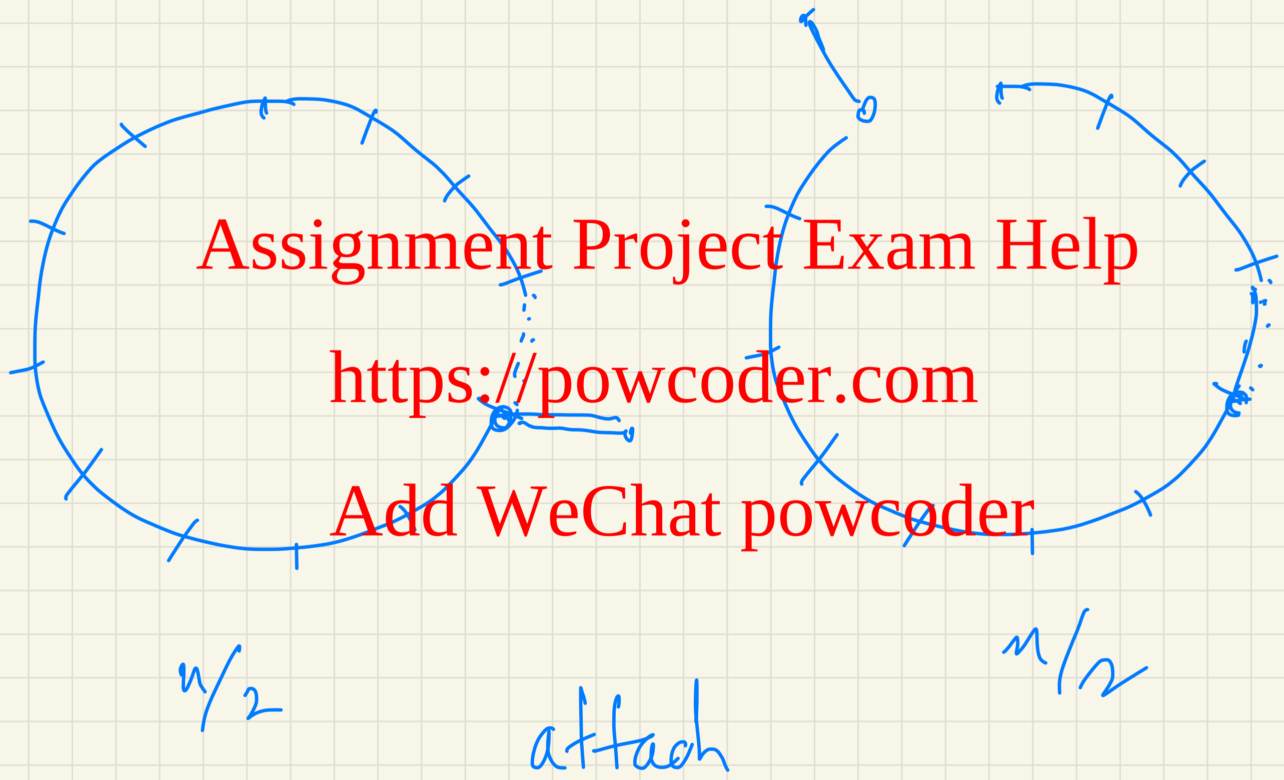
We need to "cut" the ring into two subrings



A schedule in which edges are "left open".

Open means unused.

We need to ensure that an execution will take place that will transmit "enough" messages and still leave an edge unused.


n/2
n/2
attach

## Scheduler



- We assume no two events happen at exactly the same time.



- During the proof we can "play god" and specify which message in transmission arrives next in the execution.
- If more than one message is in transit, the scheduler can choose which one arrives first.




- If two messages are transmitted over the same directed edge, then it is sometimes required that the message transmitted first will also be received first.
  - We respect the FIFO conditions for links.

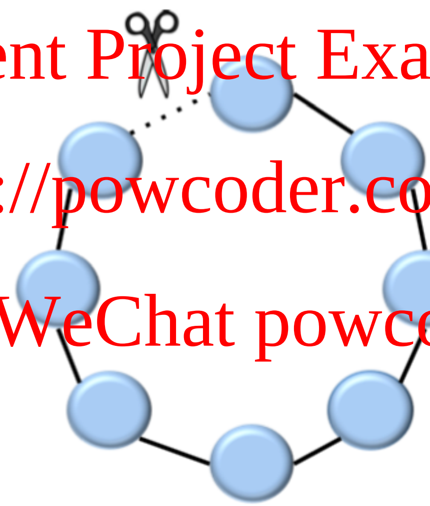
## Open Schedules



- Schedule: Execution chosen by the scheduler



- Open schedule:
  - Schedule with an open edge / communication link





- Open edge:
  - Edge along which no message has yet been scheduled

## Main Idea: Ring of $n$ nodes



- We want to count how many messages we need to send to elect a leader and at the same time maintain an open schedule, i.e. there is an edge for which no message has been received so far.
  - We prove it by induction on $n$
  - Will assume $n$ is a power of 2.
- The proof is by induction on $n$:
  - The base case is for $n = 2^1$;
  - The inductive step is for $n = 2^i$, where $i > 1$.



$$\underbrace{M(n)}_{\checkmark} \geq \underbrace{M\left(\frac{n}{2}\right)}_{\checkmark} + \underbrace{M\left(\frac{n}{2}\right)}_{\checkmark} + \frac{n}{4}$$

### Base Case: Starting the recursion in a 2-node Ring



- **Lemma 1** *Given a ring $R$ with two nodes, we can construct an open schedule in which at least one message is received.*



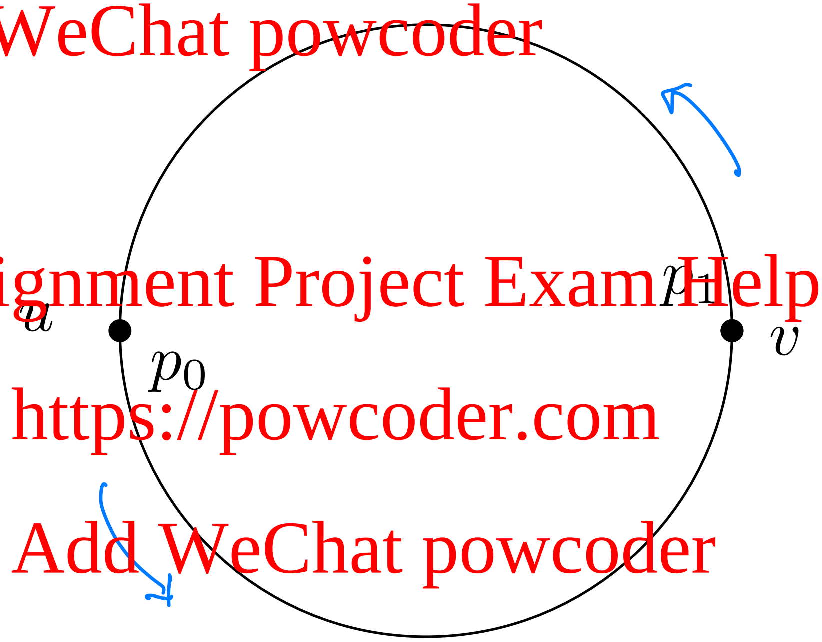
  – *The nodes cannot distinguish this schedule from one on a larger ring with all other nodes being where the open edge is.*

## 2-node Ring



- Two processors $p_0, p_1$ have identifiers $u$ and $v$ s.t. $u > v$.







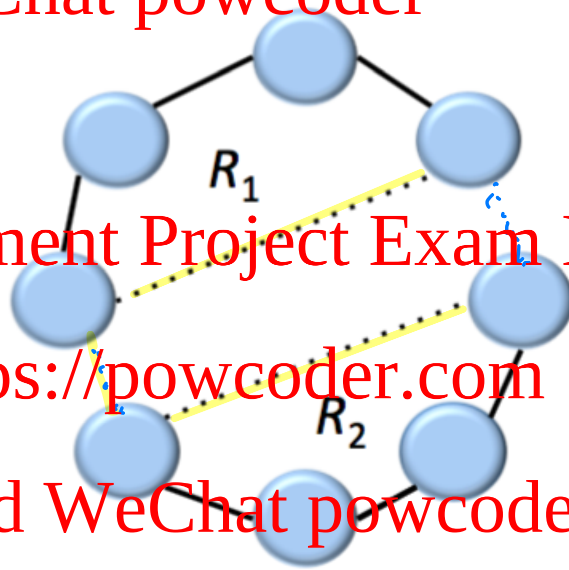

- Processor $p_1$ must learn the identity of node $v$, thus receive at least one message.
- We stop the execution of the algorithm as soon as the first message is received.

## Induction Step



- Assume two rings of size $n/2$ with open schedules





## Induction Step

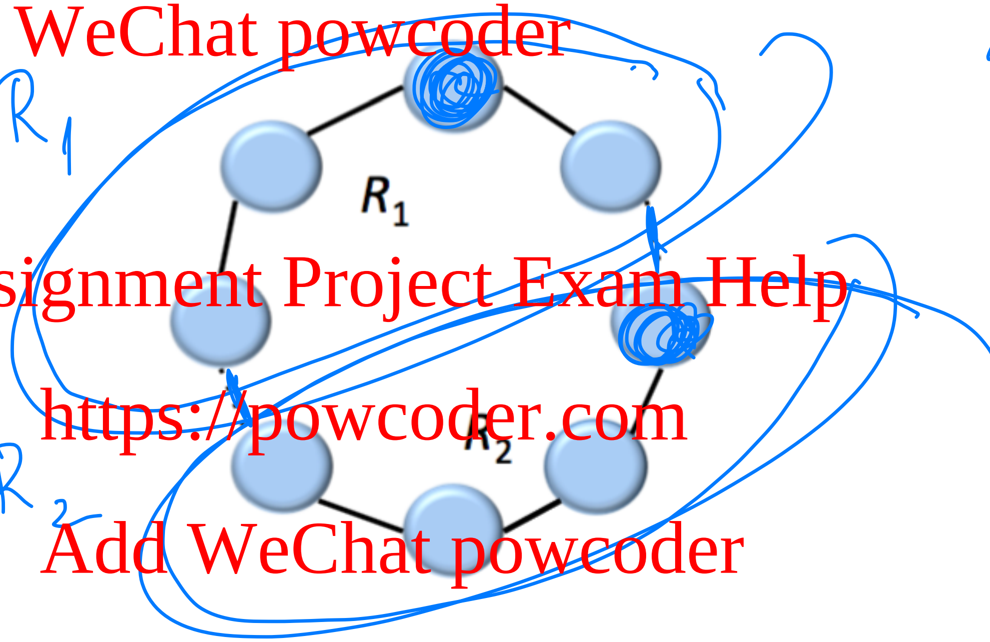


- Assume two rings of size $n/2$ with open schedules

Def. of LE problem is critical



leader in $R_1$






leader in $R_2$





We can construct an open schedule on a ring of size $n$

- If $M(n/2)$ is number of messages can construct schedule with $2M(n/2)$ without scheduling either of the two open edges
- Remember: We decide when edges are scheduled and when nodes wake up

## Induction Step



- Each node in, say $R_1$, must learn of at least one node in $R_2$



- At least $n/2$ messages must be passed from $R_1$ to $R_2$[a]
- But some messages use $e_1$ and others use $e_2$ is not good enough as an argument!

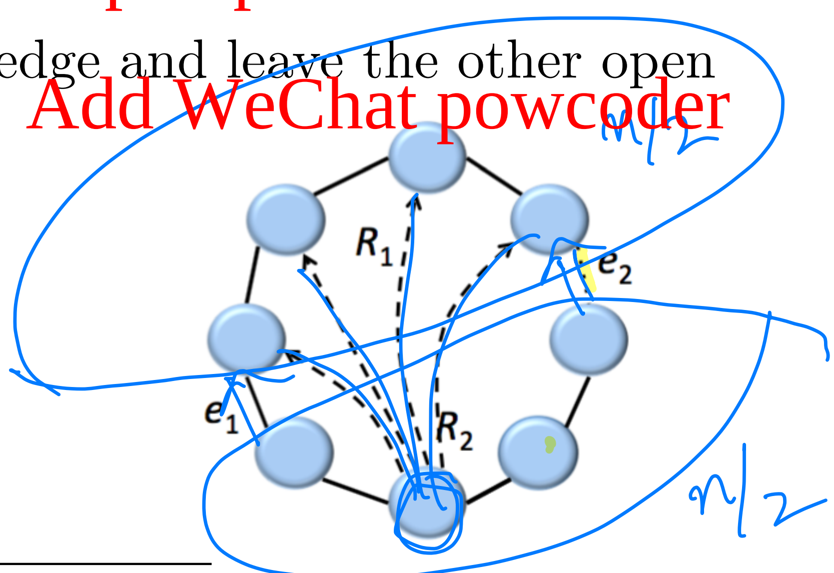


- Closing one of the edges will cause at least $n/4$ messages to be passed (not necessarily over the closed edge though)



- Schedule this edge and leave the other open






$\frac{n}{2}$

At least $n/4$ messages must pass through $e_1$ or $e_2$

[a] This is crucial to the leader election process!

## Gluing Together



- We can show that

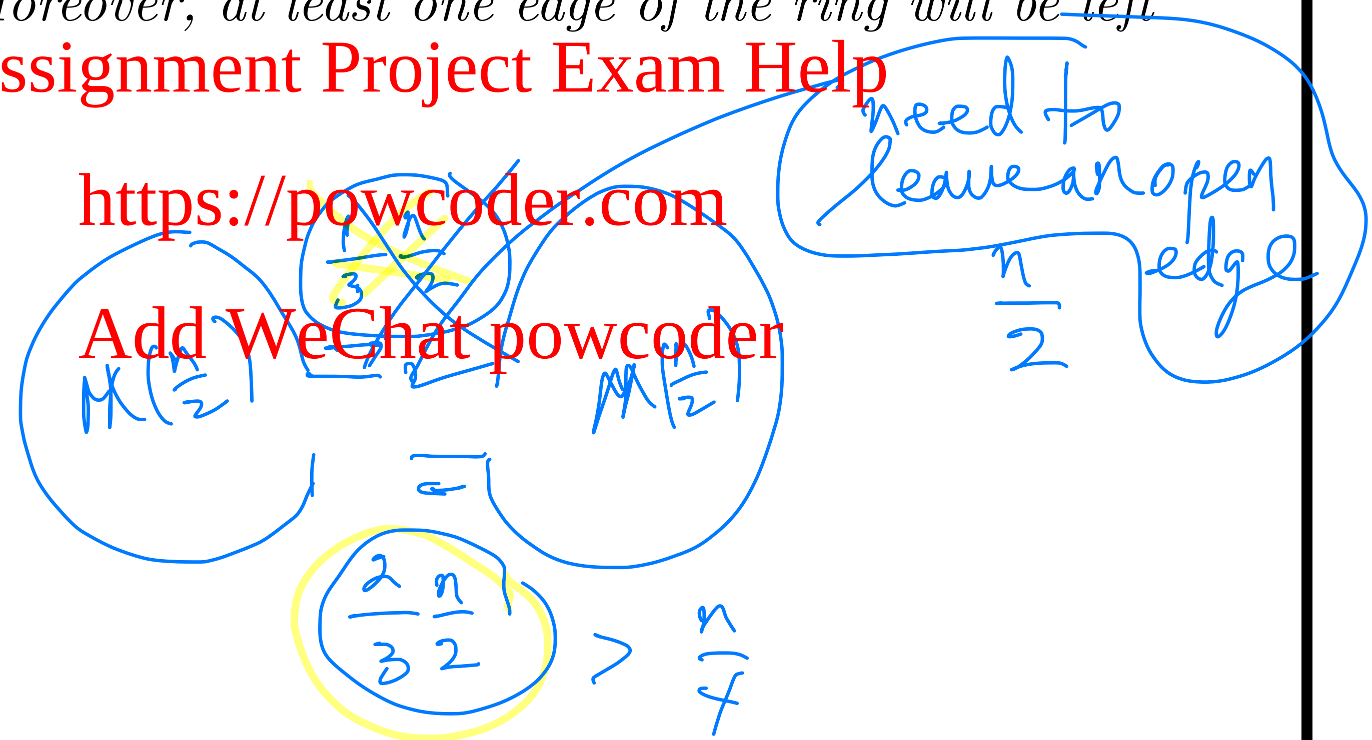
  

  **Lemma 2** *Gluing together $R_1$ and $R_2$, at least $2M(n/2) + n/4$ messages must be exchanged to solve leader election on the ring of size $n$. Moreover, at least one edge of the ring will be left open.*[a]









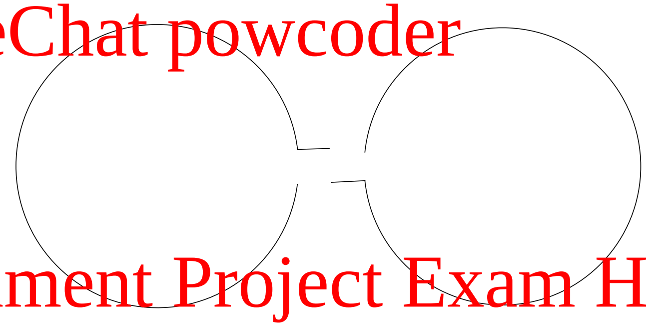

[a]This is crucial to the validity of the induction step.

## Recursive Step: Stitching



- Take two size $n/2$ subrings $R_1$ and $R_2$ with open schedules.







  – Take the open edges in each sub-schedule and use these edges to glue the subrings into a ring of size $n$.





- Electing a leader in the resulting "glued" ring involves messages that

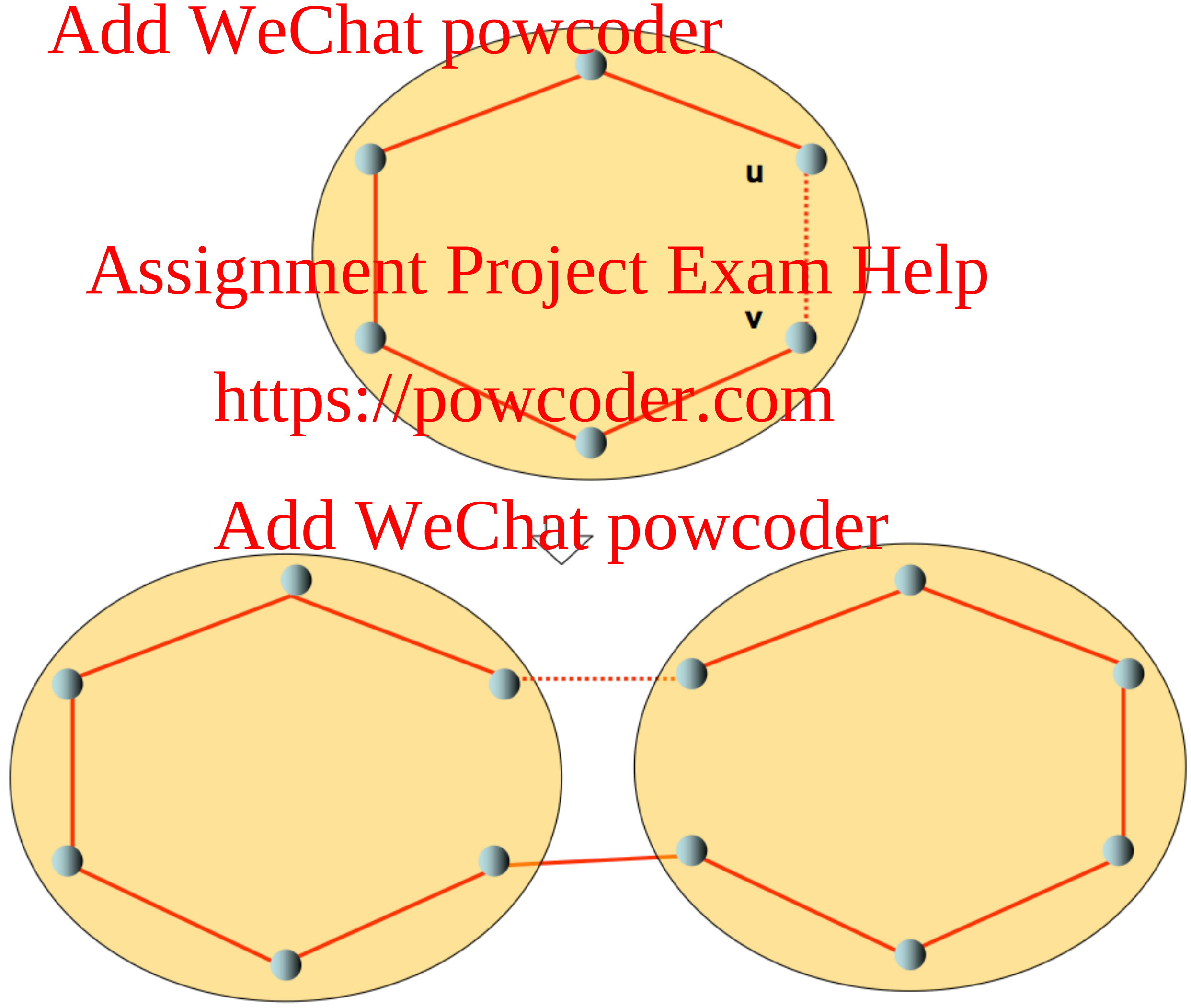
  1. stay within each subring, plus
  2. move from one ring to the other.

## Stitching



- Glue two rings of size $n/2$:





## $n$-node Ring: Idea of Inductive Hypothesis



- **Lemma 3** *By glueing together two rings of size $n/2$ for which we have open schedules, we can construct a new open schedule on a ring of size $n$.*
  - *If $M(n/2)$ denotes the number of messages already received in each of these schedules, at least $n/4$ additional messages have to be exchanged in order to solve leader election.*

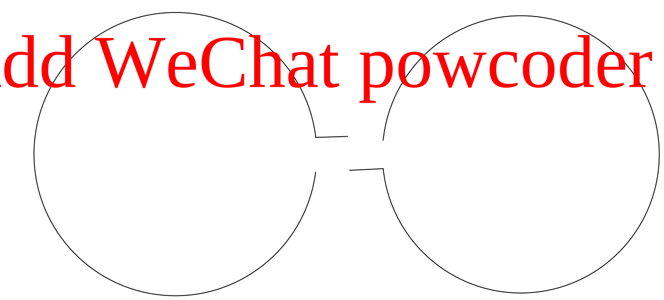




- Divide the ring into two subrings $R_1$ and $R_2$ of size $n/2$.







- These subrings cannot be distinguished from rings with $n/2$ nodes if no messages are received from "outsiders".
- Can ensure this by not scheduling such messages until we want.

## $n$-node Ring: Idea of Inductive Hypothesis

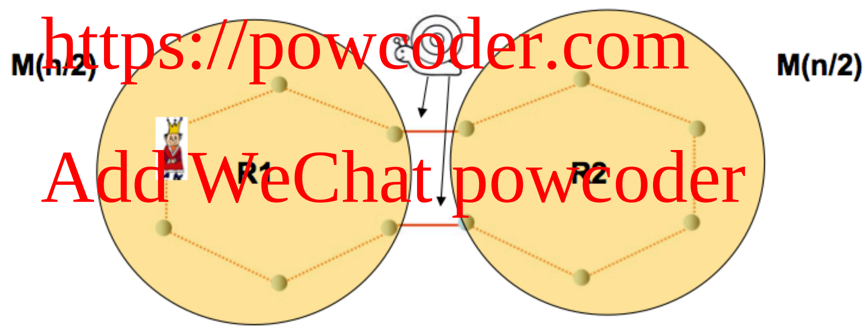


- Executing both given open schedules on $R_1$ and $R_2$ "in parallel" is possible because we control not only the scheduling of the messages, but also when nodes wake up.



- This ensures that $2M(n/2)$ messages are sent before the nodes in $R_1$ and $R_2$ learn anything of each other!










Without loss of generality, $R_1$ contains the maximum identifier.

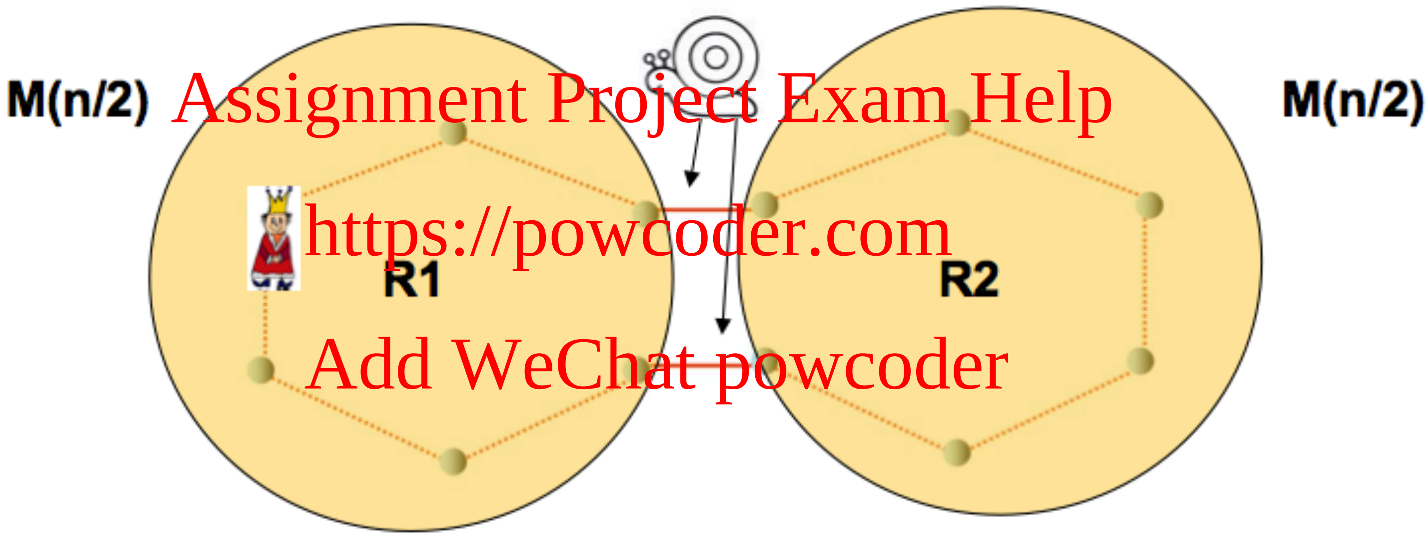
- Each node in $R_2$ must learn the identity of the max identifier, thus at least $n/2$ additional messages must be received.

## Close One and Open the Other



- The only problem is that we cannot connect the two subrings with both edges since the new ring needs to remain open.








- Thus, only messages over one of the edges can be received.
- We look into the future: we check what happens when we close only one of these connecting edges.

## Count Messages

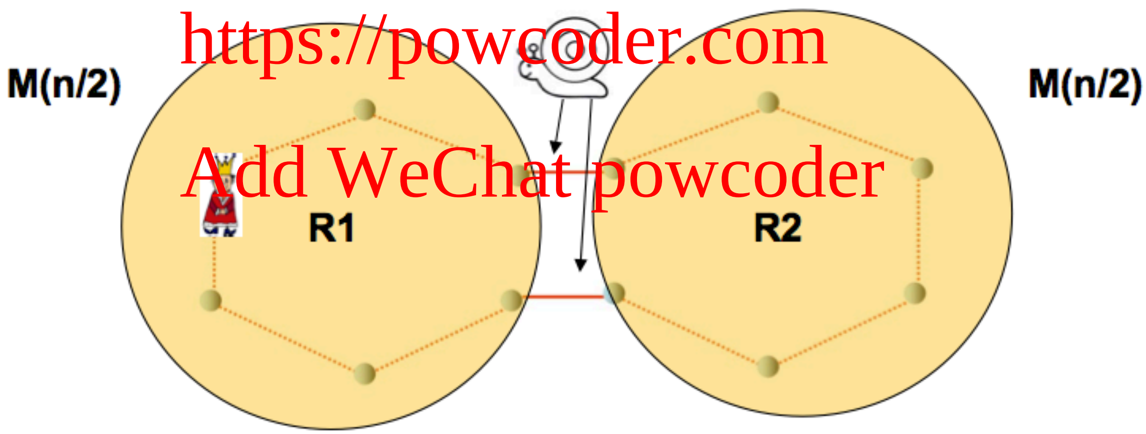


- Since we know that $n/2$ nodes have to be informed in $R_2$, there must be at least $n/2$ messages that must be received by $R_2$.



- Closing both edges must inform $n/2$ nodes, thus for one of the two edges there must be a node in distance $n/4$ which will be informed upon creating that edge.





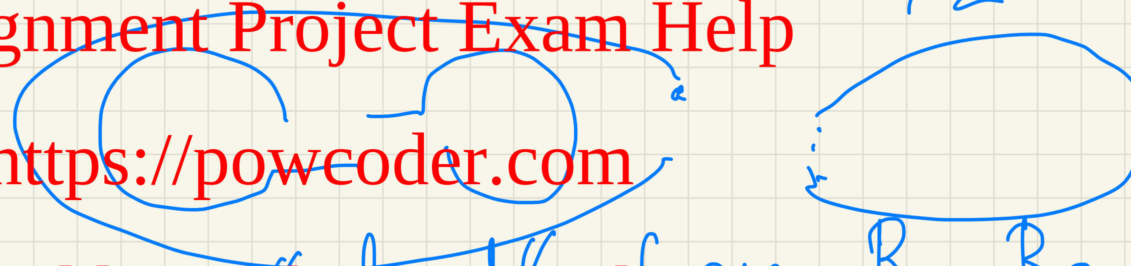



- This results in $n/4$ additional messages. Thus, we pick this edge and leave the other one open which yields the claim.

We know that $n/2$ messages must pass through either via $e_1$ or via $e_2$

why isn't the overhead $n/2$





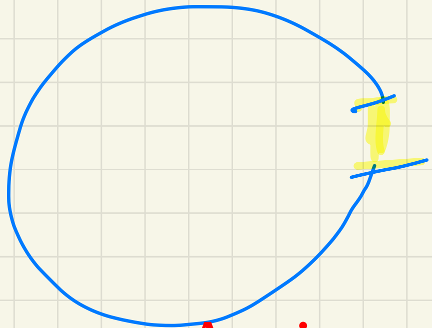


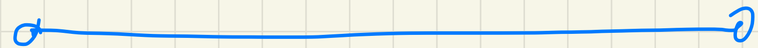
In the "new ring glued" from $R_1, R_2$

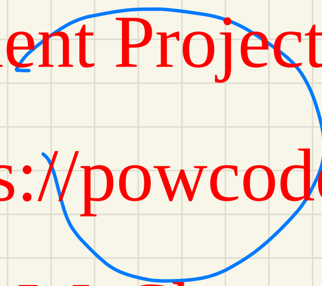


~~wee~~ we need to leave an edge open!

$$M(n) \geq M(n/2) + M(n/2) + \frac{n}{4}$$







## $\Omega(n \log n)$ messages



- So we have proved.



- **Theorem 2** *Any comparison based leader election algorithm on a ring of size $n$ needs at least $\Omega(n \log n)$ messages.*

# Randomized



# Leader Election



Determ. Lead. El.

Det. LE with random labels

## A Simple Way to Break Symmetry



- Assume that each node is equipped with a generator of random bits.



- Each node $i$ can flip a fair coin $X_i$, for $i = 0, 1, \ldots, n-1$.
  - Warning: we are not using $i$ as an identity!
  - $X_i$ is a "fair coin" means we assume that $\Pr[X_i = 0] = \Pr[X_i = 1] = \frac{1}{2}$.



- The coins are independent of each other.



- Observe that for $i \neq j$,

$$\Pr[X_i = X_j] = \frac{1}{2}$$

$\underbrace{X_i = X_j = 0}_{1/4}$ or $\underbrace{X_i = X_j = 1}_{1/4}$

## Randomized Identity Selection (1/2)



- For simplicity assume the ring is unidirectional.

  1. Each node "flips a coin" and chooses a random bit 0 or 1; the selection of each node is independent of the others.
  2. **For** $c \log n$ rounds each node sends and receives bits from its neighbour.[a] /* We must specify how the hidden constant $c$ in $c \log n$ is selected */ $c > 4$
  3. Each node uses as identity the number whose binary representation is the sequence of $c \log n$ bits it has collected, in the order received.

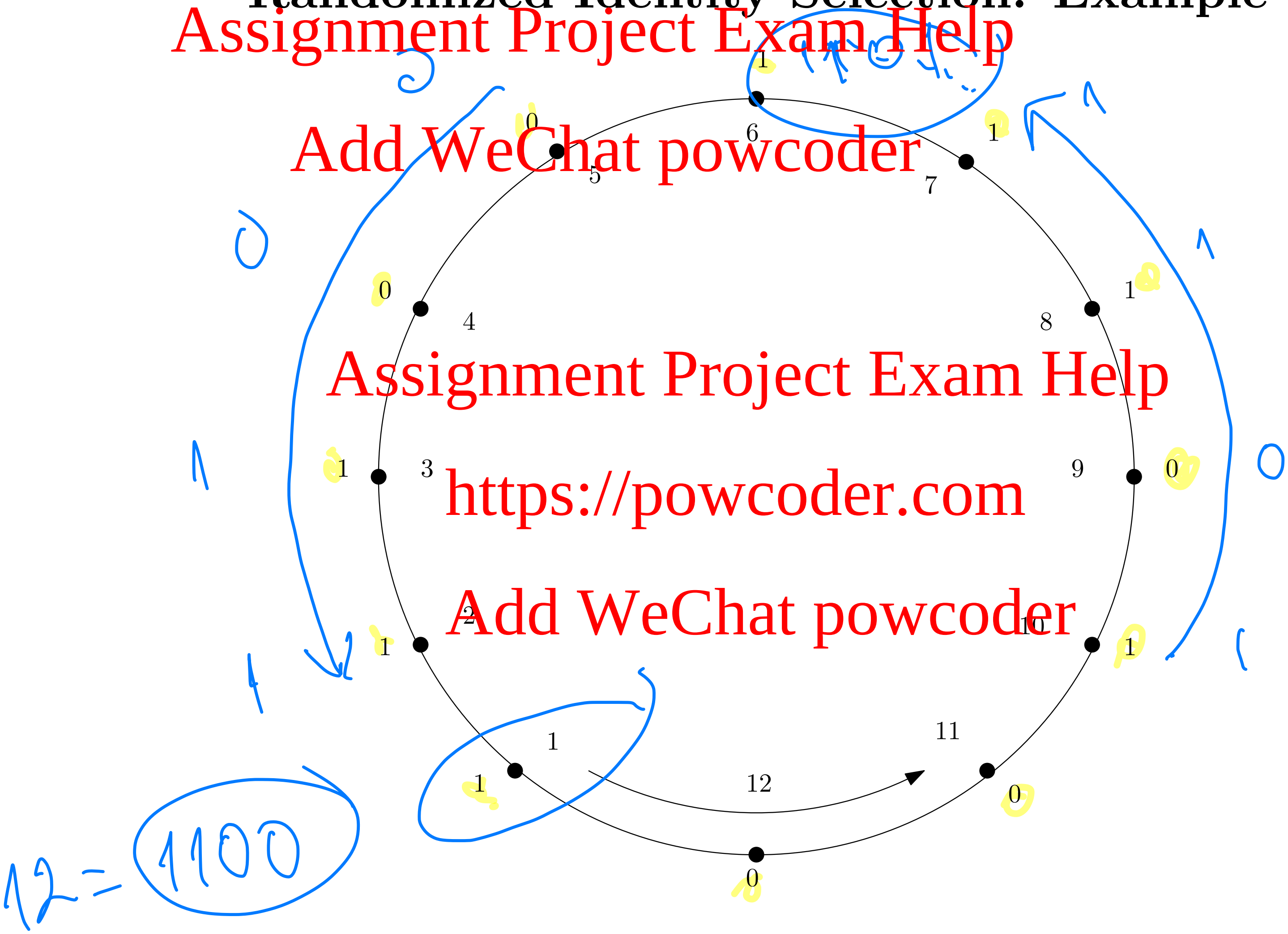
- **NB:** The input collection phase is $c \log n$ rounds.



[a] $c > 0$ is a constant that will be determined later.

## Randomized Identity Selection: Example








- Each node collects the $k$ bits it receives and converts the result to decimal.

## Randomized Identity Selection: Example for 4 Steps



| | | |
|---|---|---|
| Node 1 | 1100 | 12 |
| Node 2 | 1001 | 9 |
| Node 3 | 0011 | 3 |
| Node 4 | 0111 | 7 |
| Node 5 | 1110 | 14 |
| Node 6 | 1101 | 13 |
| Node 7 | 1010 | 10 |
| Node 8 | 0100 | 4 |
| Node 9 | 1001 | 9 |
| Node 10 | 0011 | 6 |
| Node 11 | 0111 | 7 |
| Node 12 | 1110 | 14 |

## Correctness of Randomized Identity Selection



- **Theorem 3** *For $c \geq 3$, w.h.p. (i.e., with probability $\geq 1 - 1/n$) Algorithm Randomized ID Selection ensures that the identities selected are pairwise distinct. Moreover, the algorithm*

  1. *uses a total of $n$ random bits,*
  2. *terminates in $c \log n$ rounds,*
  3. *the total number of bits transmitted is $cn \log n$.*

## Correctness of Randomized Identity Selection



- Consider the $i$-th node. Lets use the notation $k = c \log n$.



- After $c \log n$ rounds node $i$ will have received the following sequence of bits

$$X_i, X_{(i+1) \bmod n}, X_{(i+2) \bmod n}, \cdots, X_{(i+k) \bmod n}$$



and form its identity



$$ID_i := X_i X_{(i+1) \bmod n} X_{(i+2) \bmod n} \cdots X_{(i+k) \bmod n}$$



$X_j X_{j+1} \cdots$

- We now ask the question.

  How likely is it that two different nodes $i \neq j$ of the ring will obtain the same identity?

## Correctness of Randomized Identity Selection



- Assume $i \neq j$.



- Observe that

$$
\begin{aligned}
ID_i = ID_j \text{ iff } & X_i X_{(i+1) \bmod n} \cdots X_{(i+k) \bmod n} \\
& = X_j X_{(j+1) \bmod n} \cdots X_{(j+k) \bmod n} \\
& \text{iff } X_{i+l} = X_{j+l}, \text{ for all } 1 \leq l \leq k.
\end{aligned}
$$





- Therefore



$$
\begin{aligned}
\Pr[ID_i = ID_j] &= \Pr[\forall l \leq k (X_{i+l} = X_{j+l})] \\
&= \prod_{l=1}^{k} \Pr[X_{i+l} = X_{j+l}] \\
&= 2^{-k} = \frac{1}{n^c} \text{ (since } k = c \log n)
\end{aligned}
$$

$1/2$

$(1/2)^k$

## Correctness of Randomized Identity Selection



- However since there are at most $\binom{n}{2}$ pairs $i \neq j$ we get



Union Rule

$$\begin{aligned}
\Pr[\exists i \neq j (ID_i = ID_j)] &\leq \sum_{i \neq j} \underbrace{\Pr[ID_i = ID_j]} \\
&\leq \frac{\binom{n}{2}}{n^c} \\
&\leq \frac{1}{n^{c-2}}
\end{aligned}$$

Boole's Rule





- Hence,



$$\underbrace{\Pr[\forall i \neq j (ID_i \neq ID_j)]} \geq \underbrace{1 - \frac{1}{n^{c-2}}}.$$

$c = 4$

$1 - \frac{1}{100^2}$

## Randomized Leader Election (2/2)



- A leader election algorithm in a ring of size $n$ has to run for $n$ rounds so as to ensure that every node is informed of the leader.



- $\boxed{\textbf{Algorithm Randomized Leader Election}}$

1. Each node chooses a random bit 0 or 1 independently of each other.
2. **For** $n$ rounds each node sends and receives bits from its neighbour.
3. Each node computes its identity as the sequence of $n$ bits it receives /* in the order received */
4. A node becomes a leader if its identity is the largest among everybody else's identities.

## Randomized Identity Selection: Example



- Lets run the algorithm for 4 steps:



| | | |
|---|---|---|
| Node 1 | 1100 | 12 |
| Node 2 | 1001 | 9 |
| Node 3 | 0011 | 3 |
| Node 4 | 0111 | 7 |
| Node 5 | 1110 | 14 |
| Node 6 | 1101 | 13 |
| Node 7 | 1010 | 10 |
| Node 8 | 0100 | 4 |
| Node 9 | 1001 | 9 |
| ... | ... | ... |



- Which one of the 12 nodes receives the largest identifier?

## Correctness of Randomized Leader Election



- **Theorem 4** *With probability at least* $\geq 1 - n^2/2^n$ *Algorithm Randomized Leader Election ensures that a unique leader is elected. The algorithm uses a total of* $n$ *random bits, terminates in* $n$ *rounds and the total number of bits transmitted is* $n^2$.[a]





- The algorithm terminates in $n$ rounds, because Step 2 of the algorithm runs for $n$ rounds (not $c \log n$ as in identity selection).



- In Step 4, how does each node compute the identity of every other node without additional communication?



[a]Notice that the IDs constructed by this algorithm can be as large as $2^n$.

## Correctness of Randomized Leader Election



- For the sake of simplicity, assume addition below is mod$n$



- If

$$ID_i = X_i X_{i+1} \cdots X_{i+n-1}$$

is $i$'s identity as computed by the algorithm then $i$ can compute $ID_{i+k \bmod n}$ by simply rotating it $k$ positions.





- Indeed



$$\begin{aligned}
ID_i &= X_i X_{i+1} X_{i+2} X_{i+3} \cdots X_{i+n-2} X_{i+n-1} \\
ID_{i+1} &= X_{i+1} X_{i+2} X_{i+3} \cdots X_{i+n-2} X_{i+n-1} X_i \\
ID_{i+2} &= X_{i+2} X_{i+3} \cdots X_{i+n-2} X_{i+n-1} X_i X_{i+1} \\
\vdots &= \qquad\qquad \vdots
\end{aligned}$$

- No additional communication round is needed.

## Correctness of Randomized Identity Selection



- Finally, the claim on the probability.
- Just repeating the previous argument with $k = n$ we see that



$$
\begin{aligned}
\Pr[ID_i = ID_j] &= \Pr[\forall l \le k (X_{i+l} = X_{j+l})] \\
&= \prod_{l=1}^{k} \Pr[X_{i+l} = X_{j+l}] \\
&= 2^{-n}
\end{aligned}
$$



- Therefore since there are at most $\binom{n}{2}$ pairs $i \neq j$ we get

$$
\begin{aligned}
\Pr[\exists i \neq j (ID_i = ID_j)] &\le \sum_{i \neq j} \Pr[ID_i = ID_j] \\
&\le \binom{n}{2} / 2^n \\
&\le n^2 / 2^n .
\end{aligned}
$$

## Sources




- H. Attiya, J. Welch, Distributed Computing, John Wiley and Sons, 2E, 2004.





- R. Wattenhofer, Lecture Notes on Principles of Distributed Computing, ETH Spring 2012.





- N. Lynch, Distributed Algorithms, Morgan-Kaufmann, 1996.

## Exercises[a]



1. Can paiwise distinct identifiers be selected if the nodes have independent random generators?



2. Why are we allowed to interpret an event $E$ that is valid "with probability at least $\geq 1 - n^2/2^n$" as "it is valid with high probability"?



3. Justify the validity of the approximation $e^{-1} \approx 1 - x$, for $|x|$ sufficiently small. (Use the Taylor series expansion of the function $e^u$, where $u$ is a real number.)





4. Consider the following variant of randomized ID selection: Each node selects $k$ random bits $b_1, b_2, \ldots, b_k$ and makes the sequence $b_1 b_2 \cdots b_k$ its identifier. Show that by choosing $k$ appropriately with high probability the identifiers chosen by the nodes are pairwise distinct. **NB.** This algorithm differs

[a] Do not submit!

from the one discussed in class in that it does not require any message exchanges.



5. How do "TimeSlice" and "VariableSpeed" differ from the traditional "C;lock-wise" and "RadiusGrowth" algorithms?