Self-Stabilizing Silent Disjunction in an Anonymous Network

Ajoy K. Datta[1] Stéphane Devismes[2] Lawrence L. Larmore[1]

[1]University of Nevada Las Vegas

[2]Université Joseph Fourier, Grenoble

ICDCN 2013, 04/01/2012, Mumbai

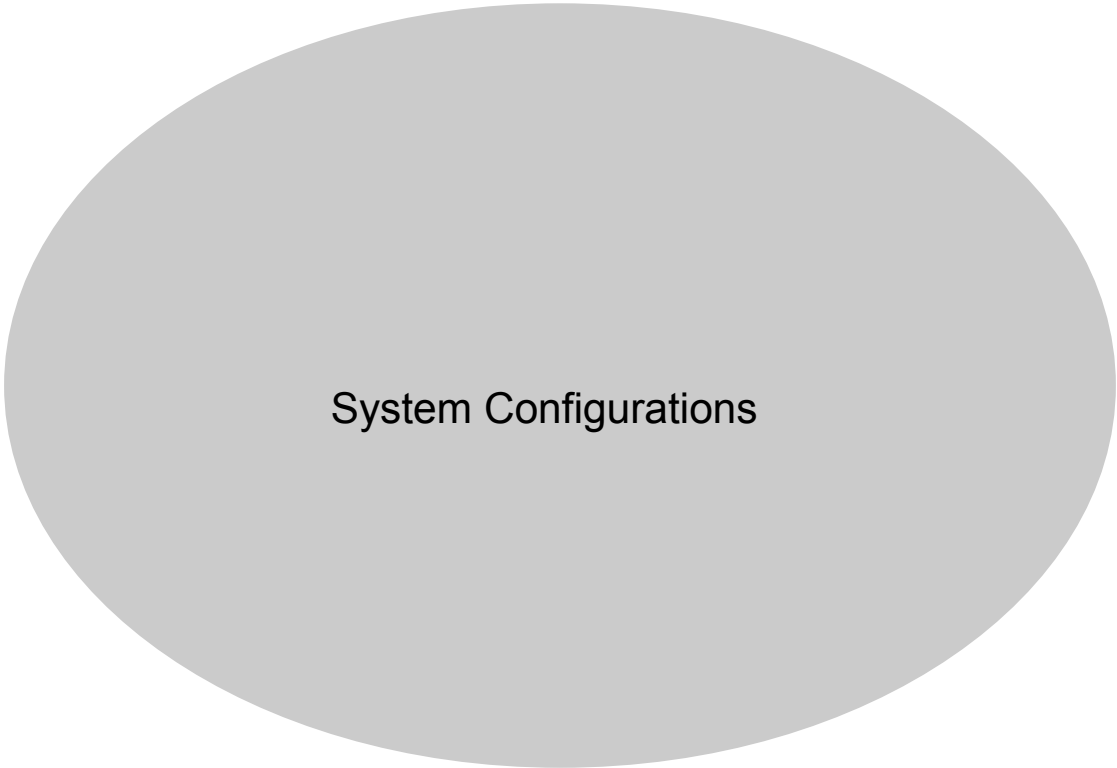
Join Work

Ajoy K. Datta

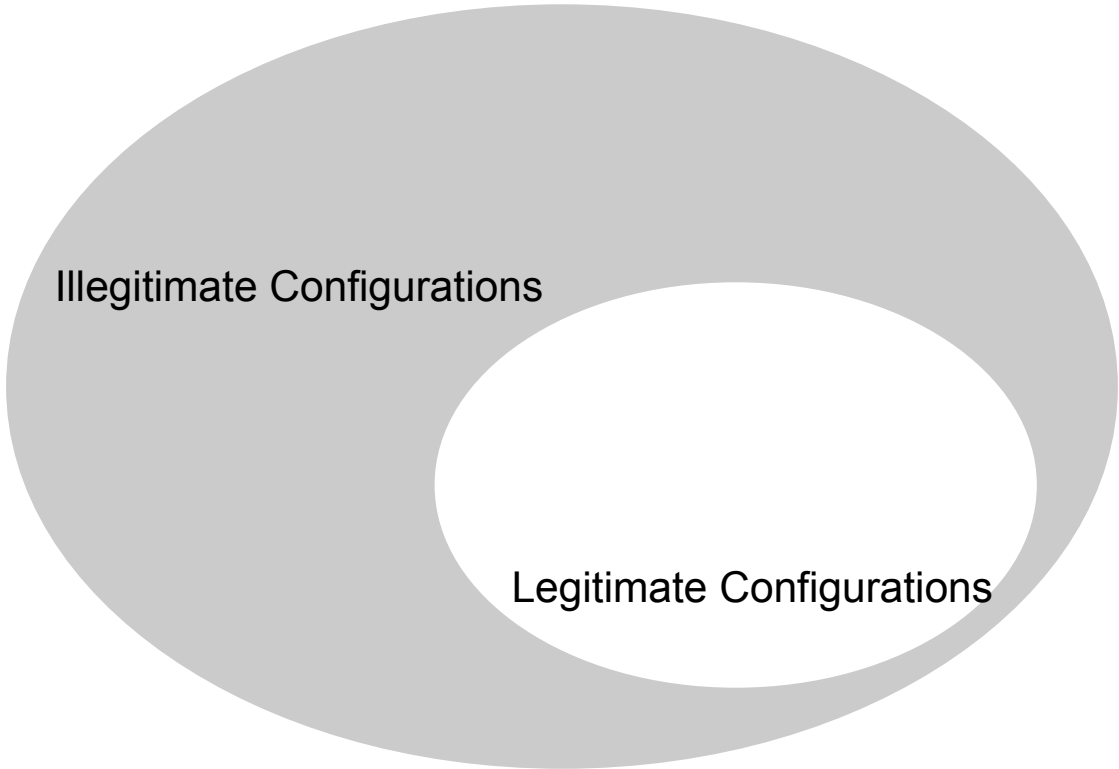
Lawrence L. Larmore

Self-Stabilization

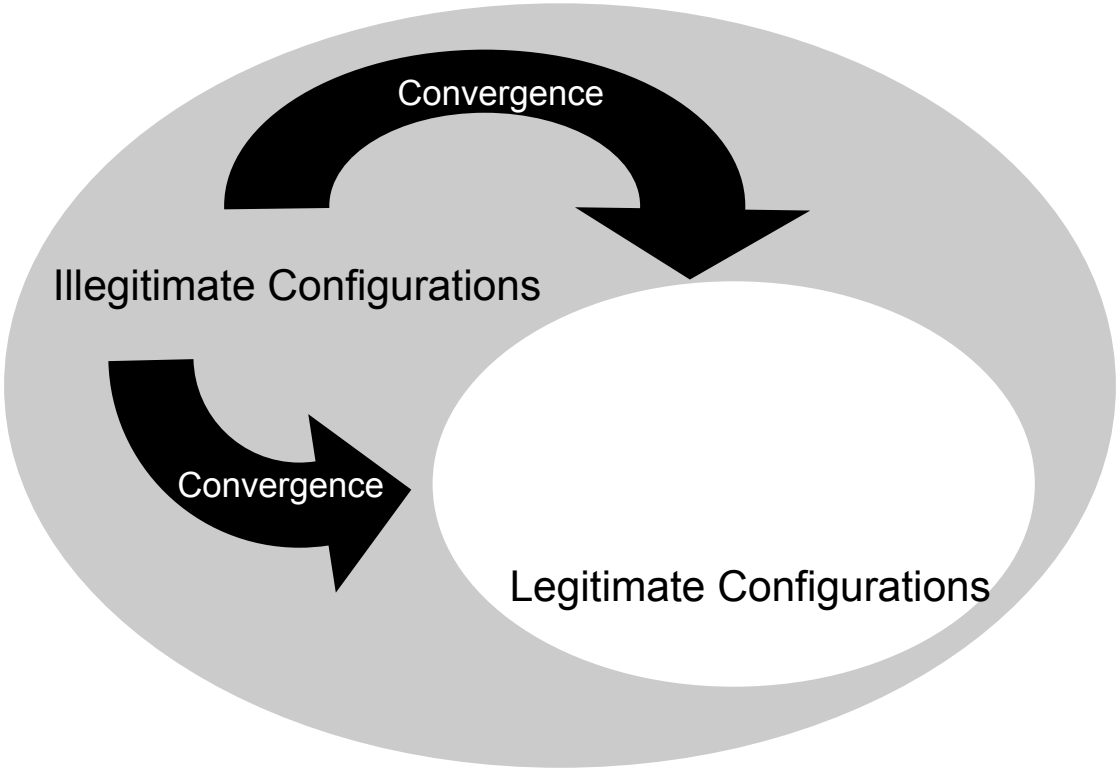
Self-Stabilization



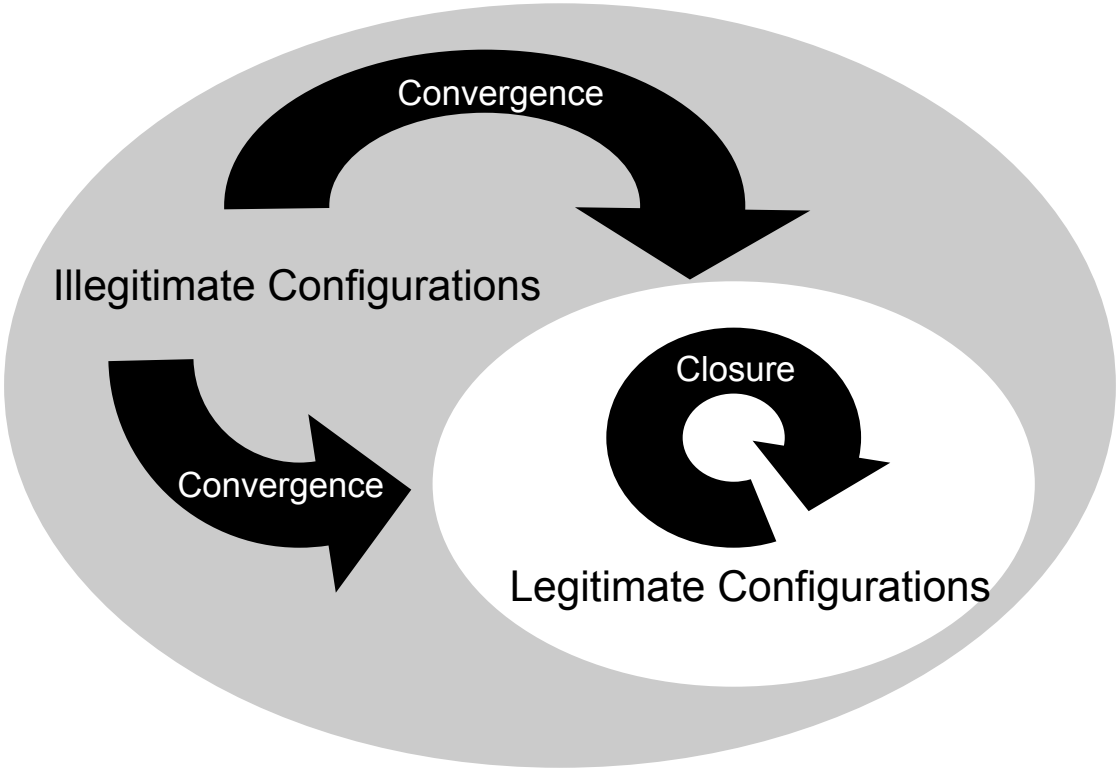
Self-Stabilization



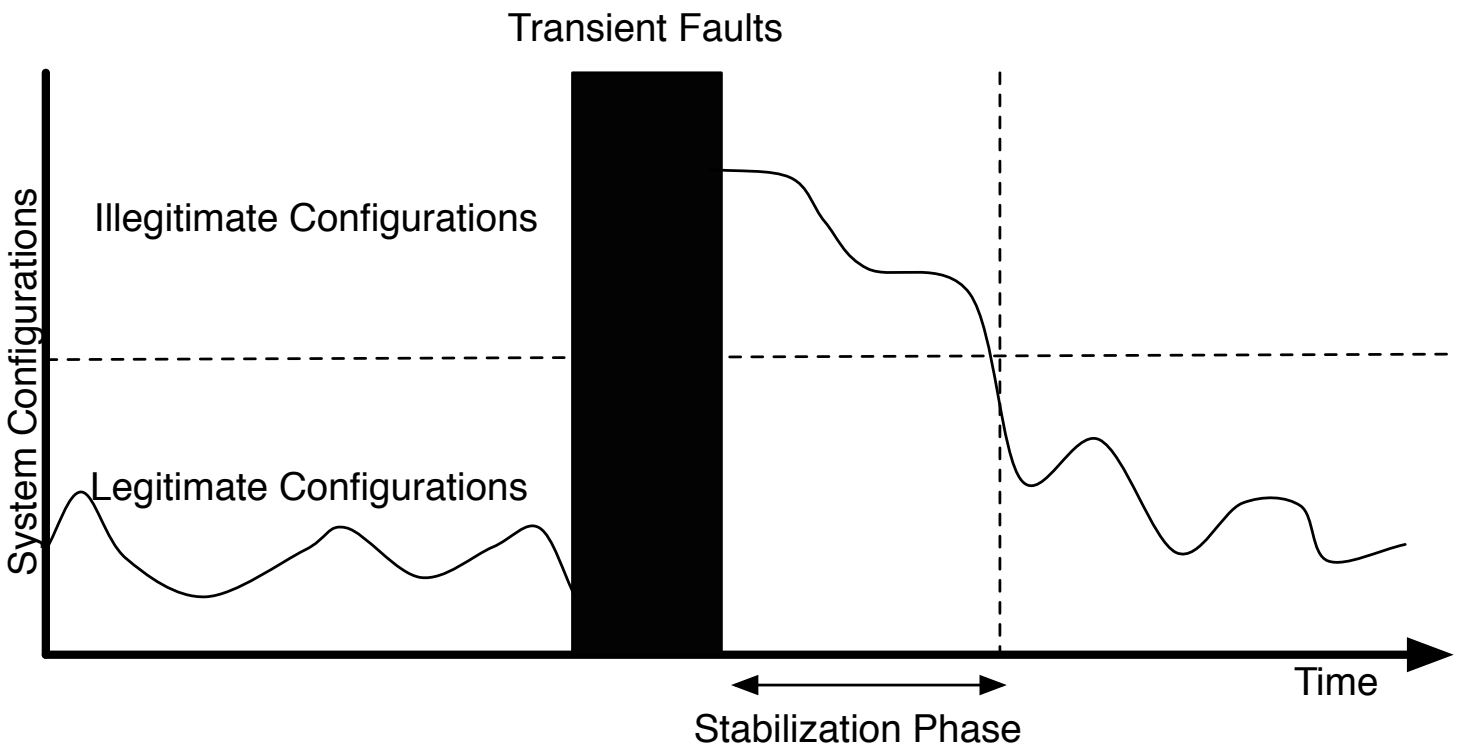
Self-Stabilization



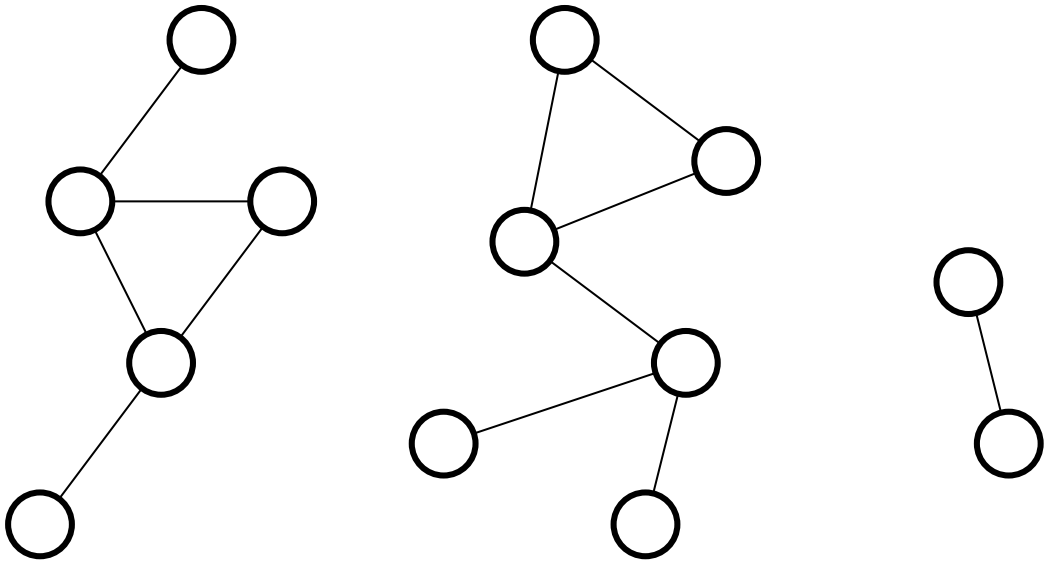
Self-Stabilization



Tolerance to Transient Faults

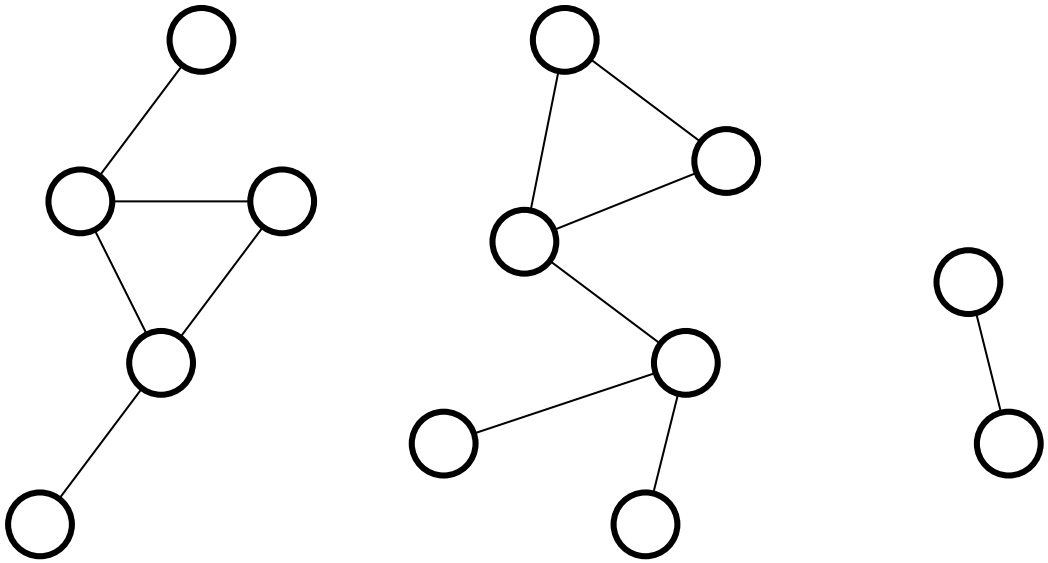


The Disjunction Problem



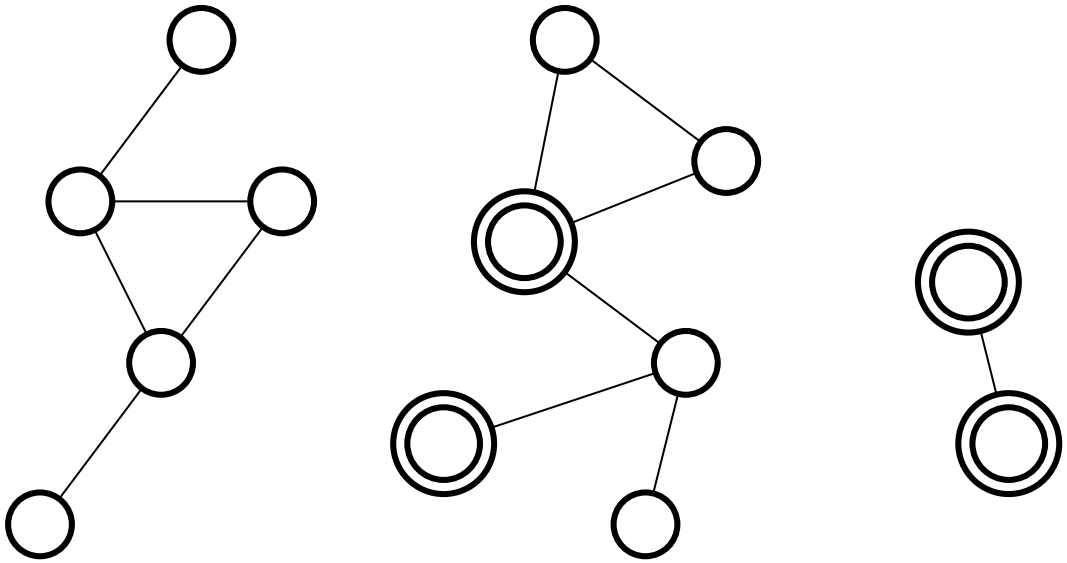
The Disjunction Problem

- **Anonymous** Network of **Processes**.



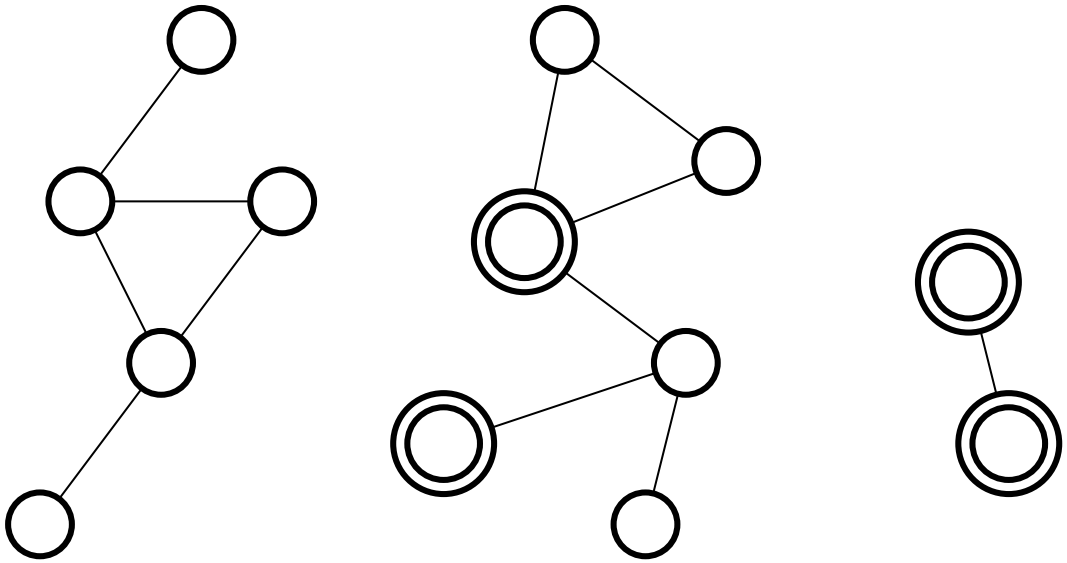
The Disjunction Problem

- Anonymous Network of Processes.
  - Each Process has **Input Bit**.



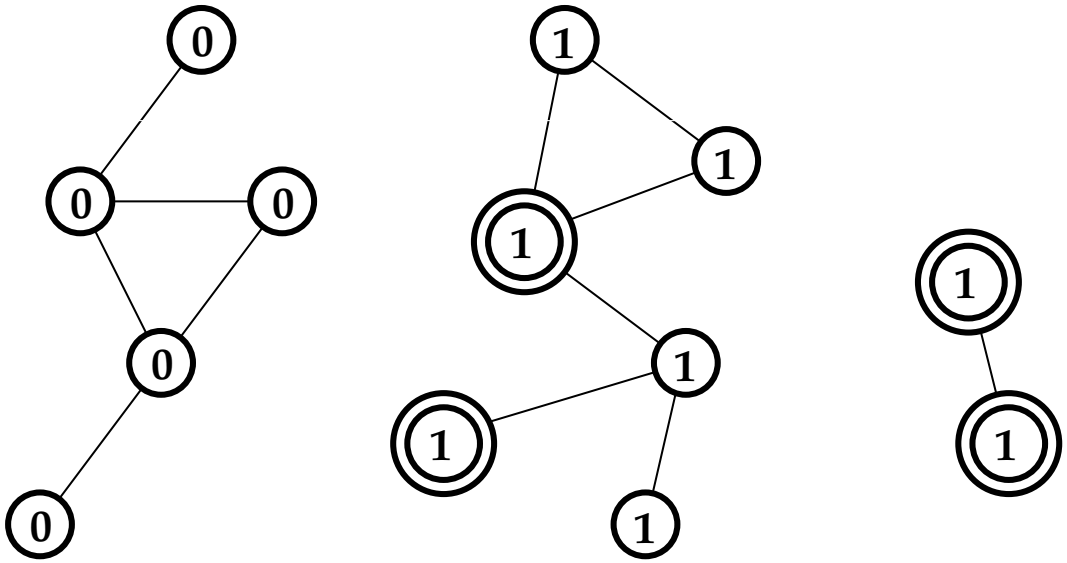
The Disjunction Problem

- Anonymous Network of Processes.
  - Each Process has Input Bit.
  - **Double Circle** $\Longleftrightarrow$ Input Bit = 1. Others are 0.



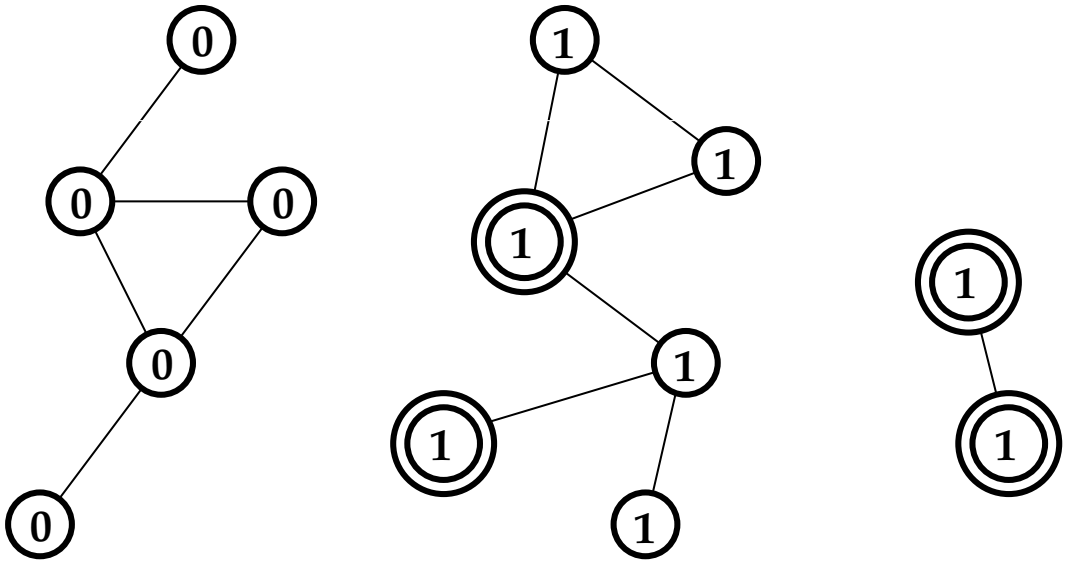
The Disjunction Problem

- Anonymous network of Processes.
  - Each Process has **Input Bit**.
  - Double Circle $\Longleftrightarrow$ Input Bit = 1. Others are 0.
  - **Output Bit** shown inside circle.



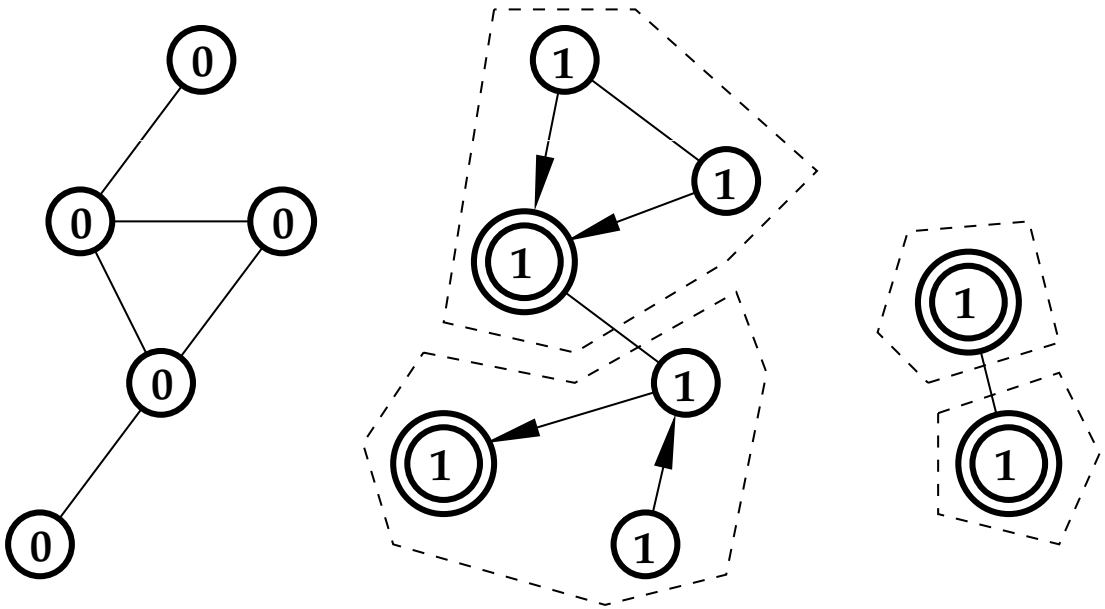
The Disjunction Problem

- Anonymous network of Processes.
  - Each Process has **Input Bit**.
  - Double Circle $\Longleftrightarrow$ Input Bit = 1. Others are 0.
  - Output Bit shown inside circle.
  - **Output Bit** of each Process is **Disjunction** of all Input Bits of Processes in its Component.



The Disjunction Problem

- Anonymous network of Processes.
  - Each Process has Input Bit.
  - Double Circle $\Longleftrightarrow$ Input Bit = 1. Others are 0.
  - Output Bit shown inside circle.
  - Output Bit of each Process is Disjunction of all Input Bits of Processes in its Component.
- (Construct **BFS Forest** Rooted at Processes with Input 1.)



Simple Flooding

Simple Flooding

- Works if **Clean Start.**

Simple Flooding

- Works if Clean Start.
- If **Arbitrary Start:**

Simple Flooding

- Works if Clean Start.
- If **Arbitrary Start:**
  - Works if Some Process has **Input 1.**

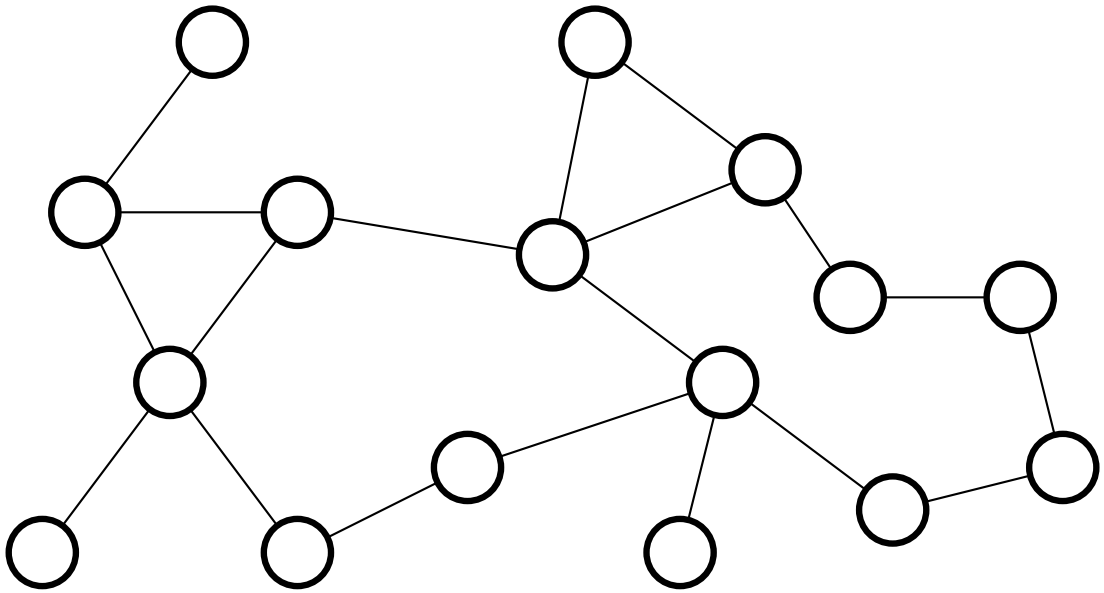
Simple Flooding

- Works if Clean Start.
- If **Arbitrary Start:**
  - Works if Some Process has Input 1.
  - All Processes have **Input 0:** might go into **Endless Loop!**

Clean Start

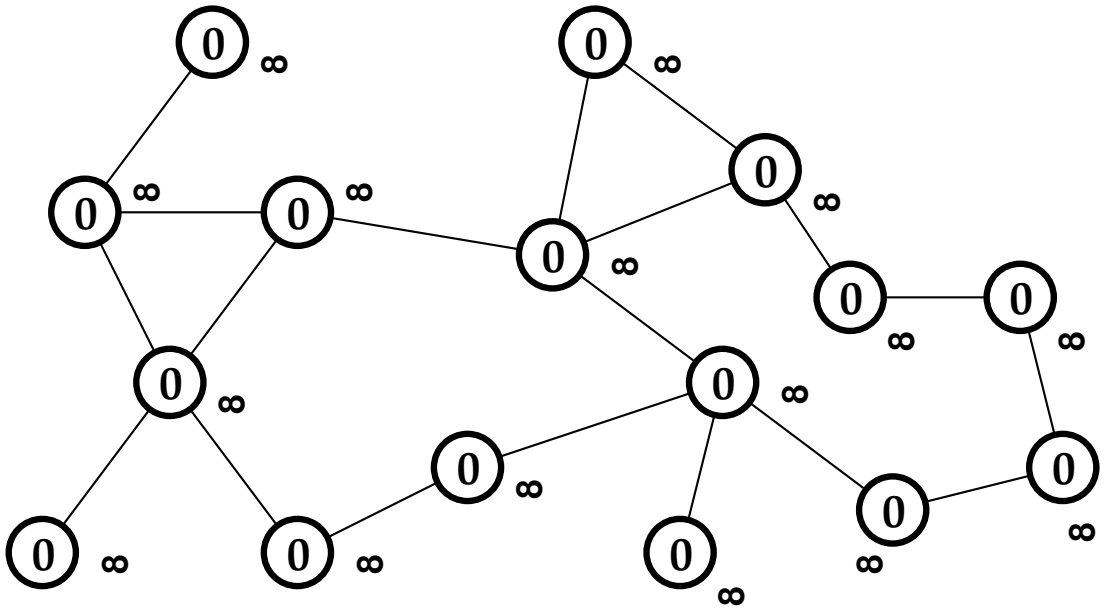
Clean Start

- **All Input Bits 0.**



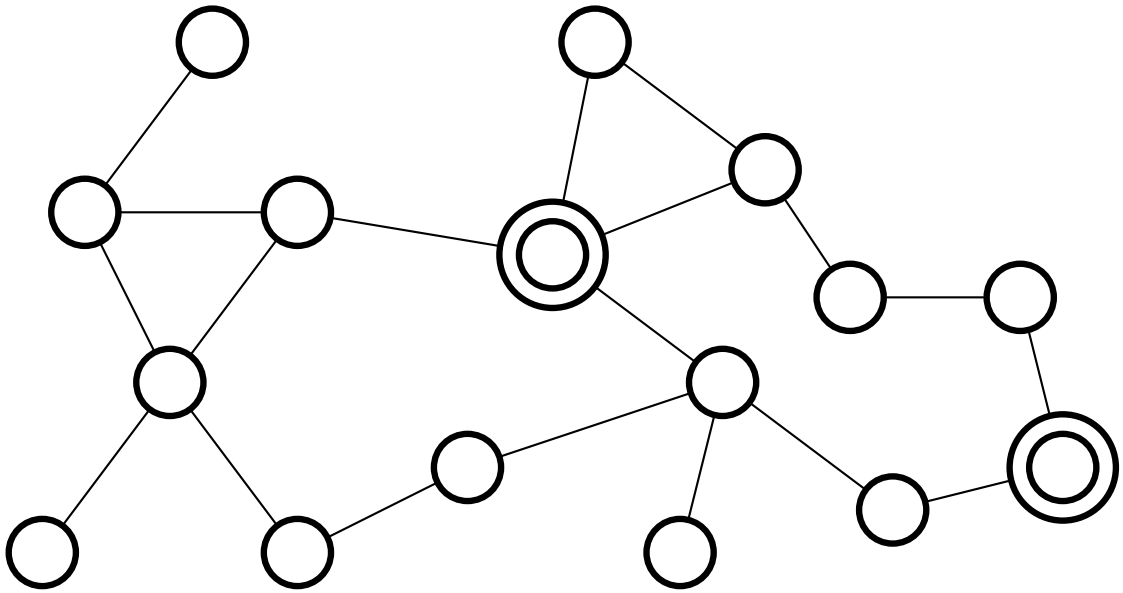
Clean Start

- All Input Bits 0.
- Clean Configuration is **Final.**



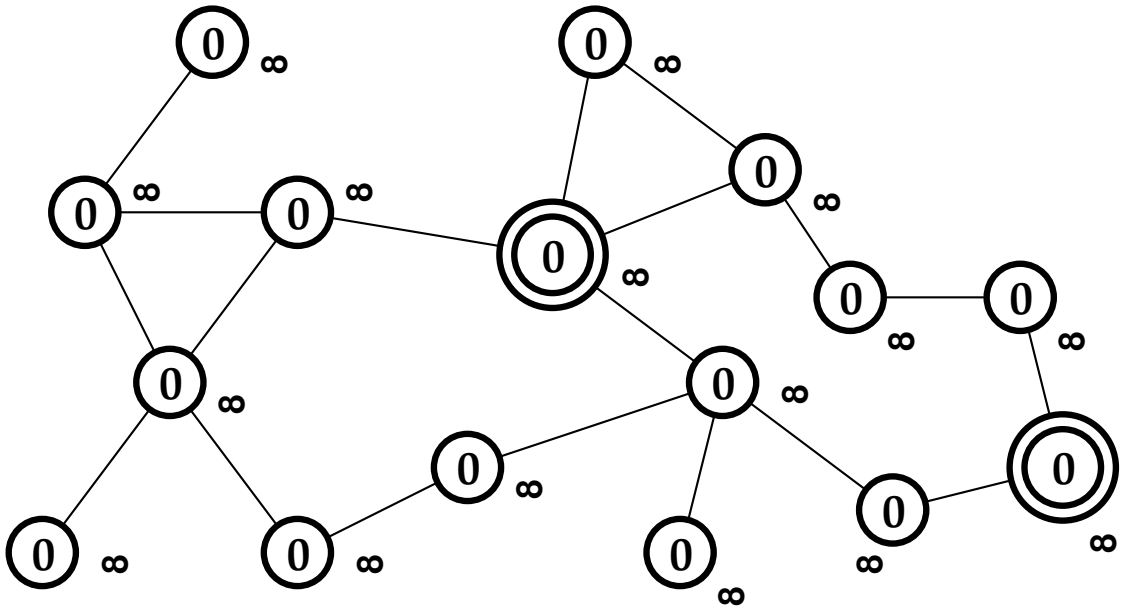
Clean Start

- **Some Input Bits = 1.**



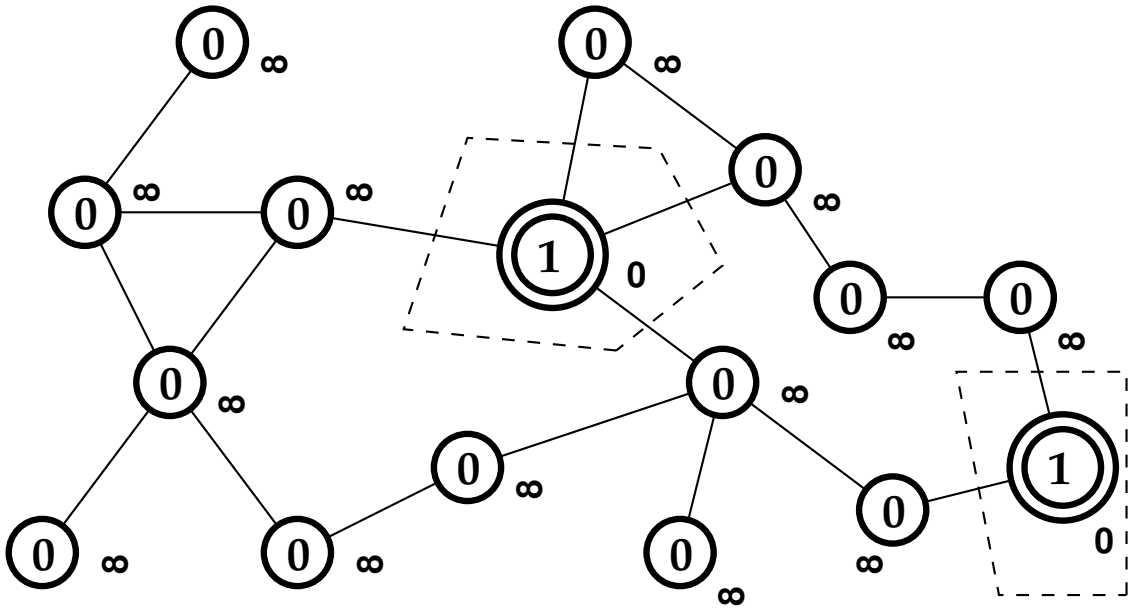
Clean Start

- Some Input Bits = 1.
- Clean Configuration is **Not Final.**



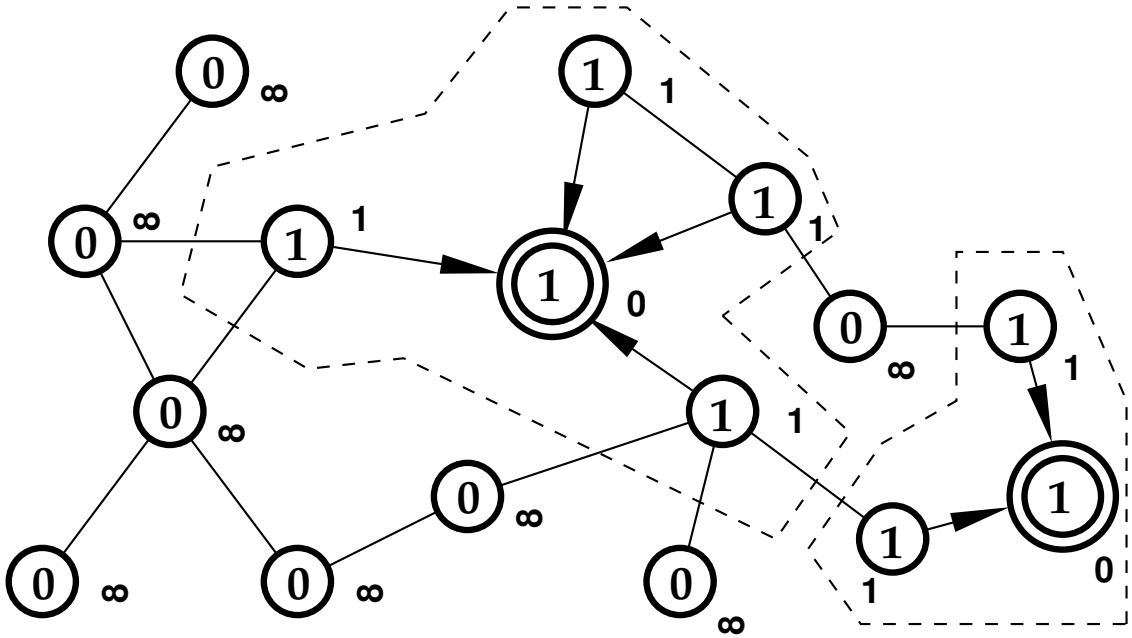
Clean Start

- Some Input Bits = 1.
- Clean Configuration is not Final.
- Clusterheads Execute **Reset.**



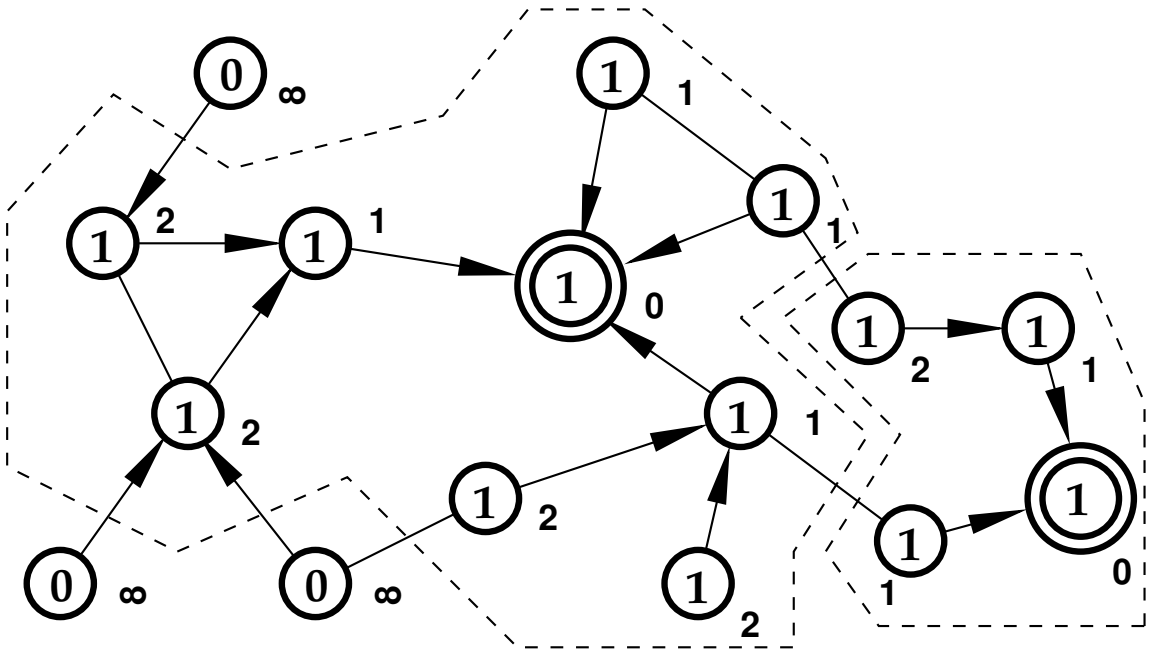
Clean Start

- Some Input Bits = 1.
- Clean Configuration is not Final.
- Clusterheads Execute Reset.
- Adjacent Processes Execute **Join.**



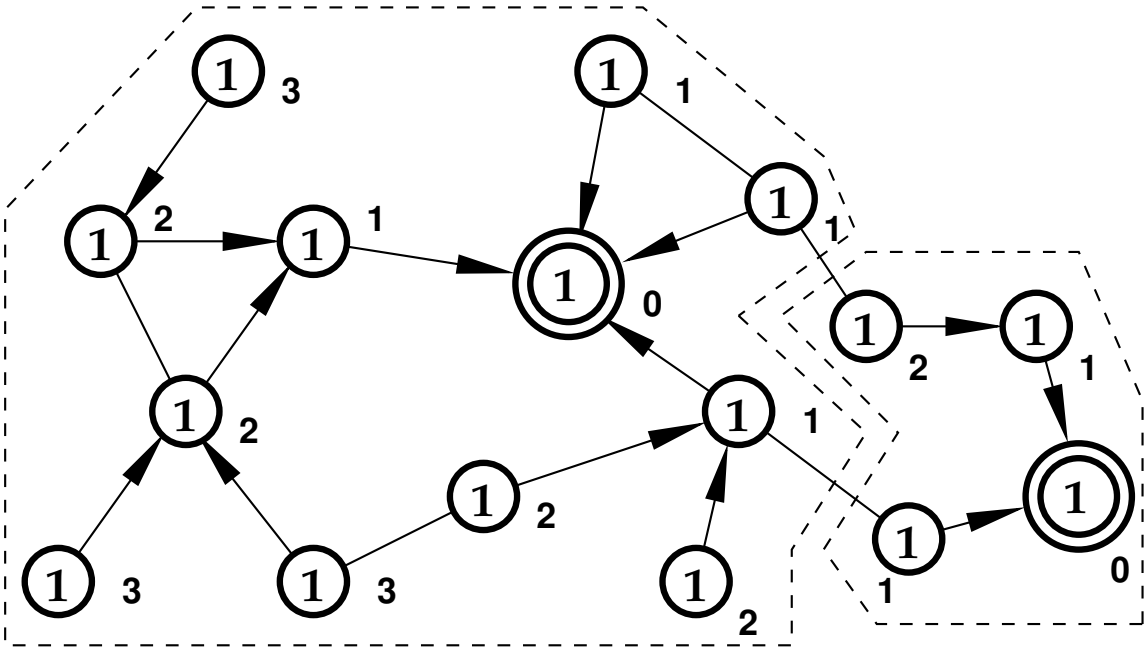
Clean Start

- Some Input Bits = 1.
- Clean Configuration is not Final.
- Clusterheads Execute Reset.
- Adjacent Processes Execute **Join.**



Clean Start

- Some Input Bits = 1.
- Clean Configuration is not Final.
- Clusterheads Execute Reset.
- Adjacent Processes Execute Join.
- **Flooding:** All Parent Pointers and Levels are Computed in at Most $(1 + Diam)$ **Rounds.**



Simple Flooding Is Not Self-Stabilizing!

Simple Flooding Is Not Self-Stabilizing!

- **Arbitrary Initial Configuration?**

Simple Flooding Is Not Self-Stabilizing!

- Arbitrary Initial Configuration.
- No Problem if Some Process has **Input 1.**

Simple Flooding Is Not Self-Stabilizing!

- Arbitrary Initial Configuration.
- No Problem if Some Process has Input 1.
- Otherwise, might go into **Endless Loop.**

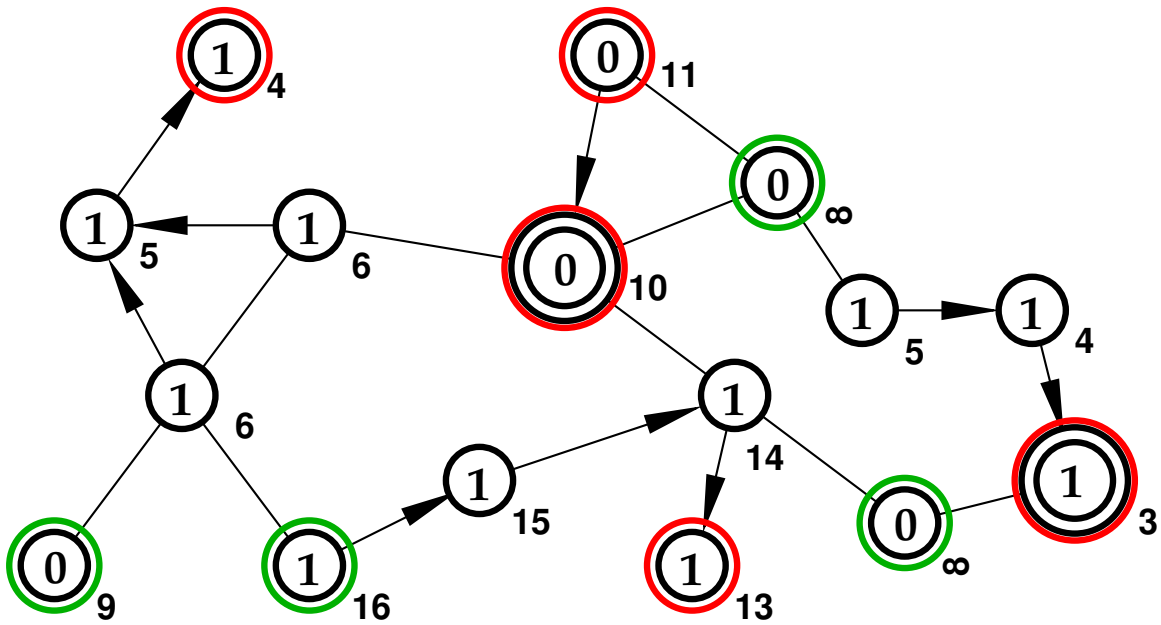
Arbitrary Start: Output = 1: No Problem!

Arbitrary Start: Output = 1: No Problem!

- Some Input Bits 1.

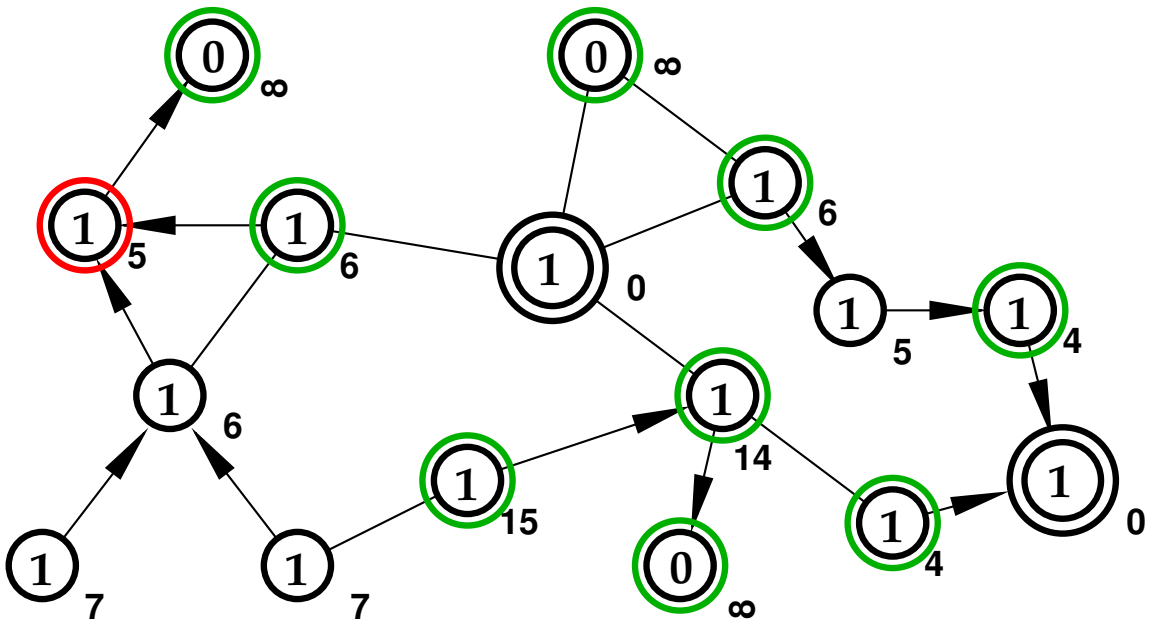
Arbitrary Start: Output = 1: No Problem!

- Some Input Bits 1.
- Red = Enabled to Reset.
- Green = Enabled to Join.



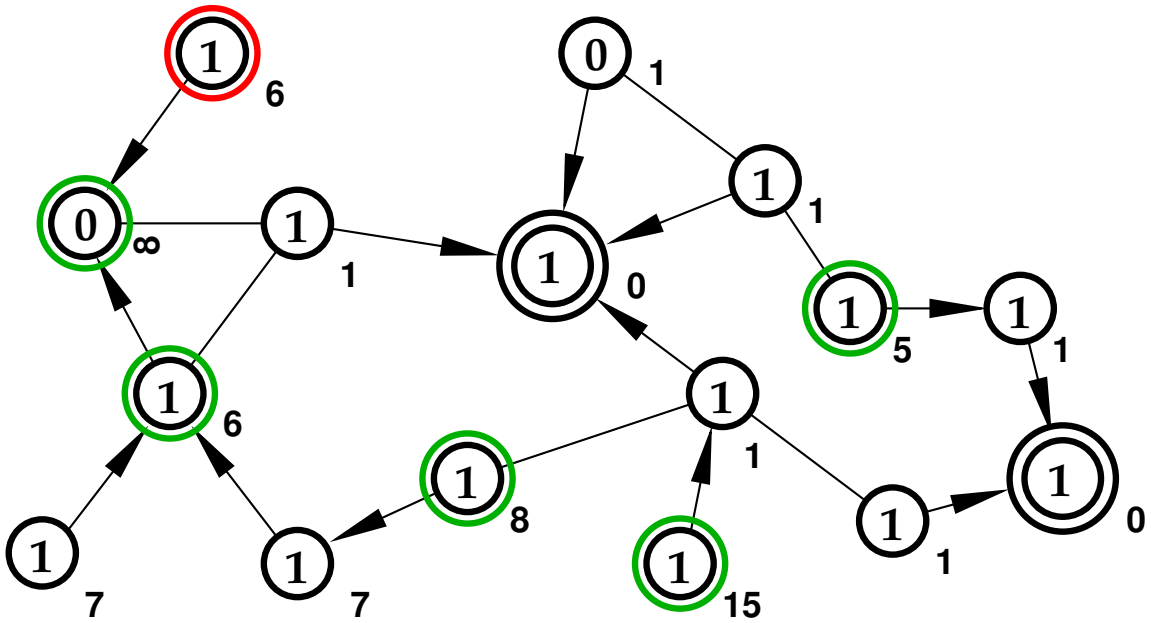
Arbitrary Start: Output = 1: No Problem!

- Some Input Bits 1.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- **Clusterheads Reset**.



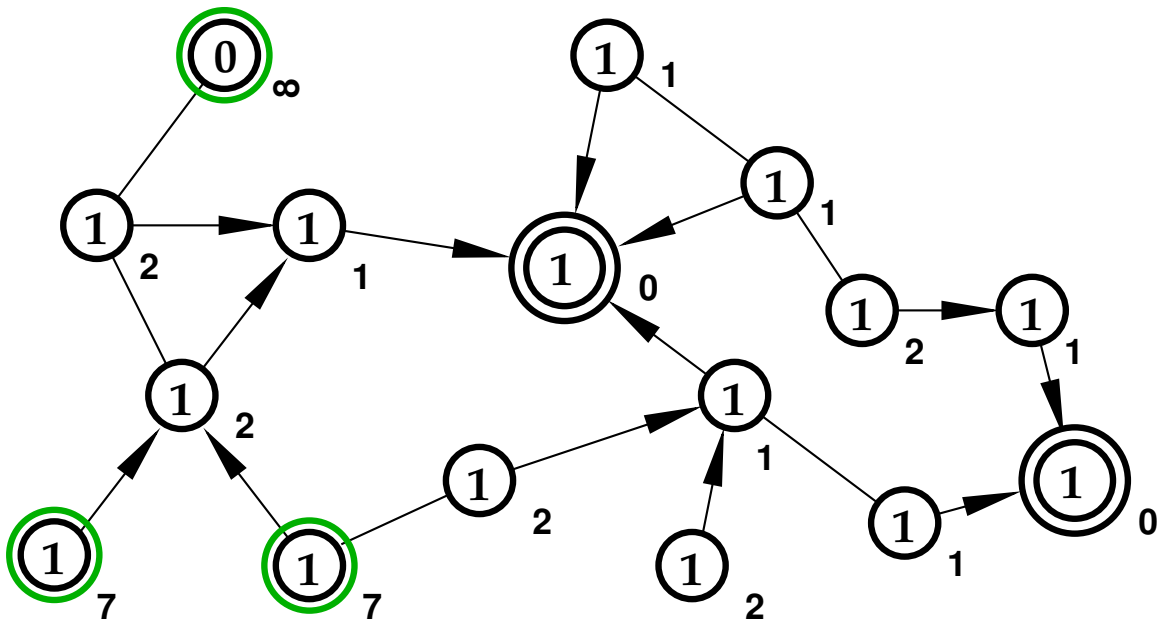
Arbitrary Start: Output = 1: No Problem!

- Some Input Bits 1.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Clusterheads Reset.
- **Flooding** Moves at Speed 1.



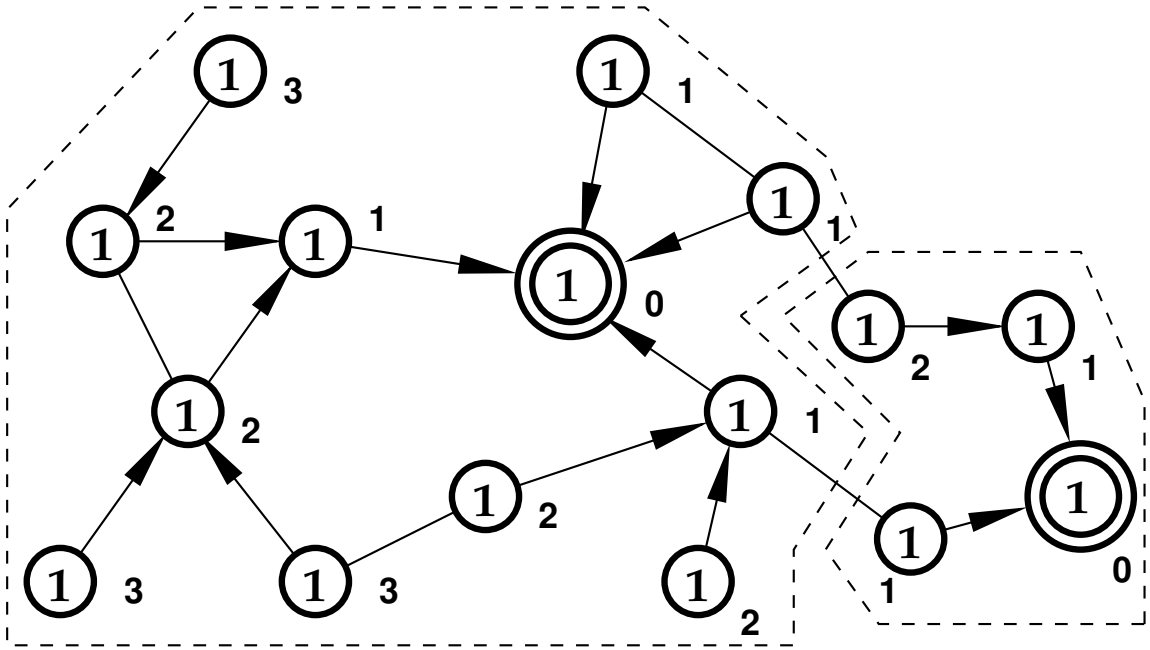
Arbitrary Start: Output = 1: No Problem!

- Some Input Bits 1.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Clusterheads Reset.
- Flooding Moves at Speed 1.



Arbitrary Start: Output = 1: No Problem!

- Some Input Bits 1.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Clusterheads Reset.
- Flooding Moves at Speed 1.
- **Convergence** within $1 + Diam$ **Rounds**.



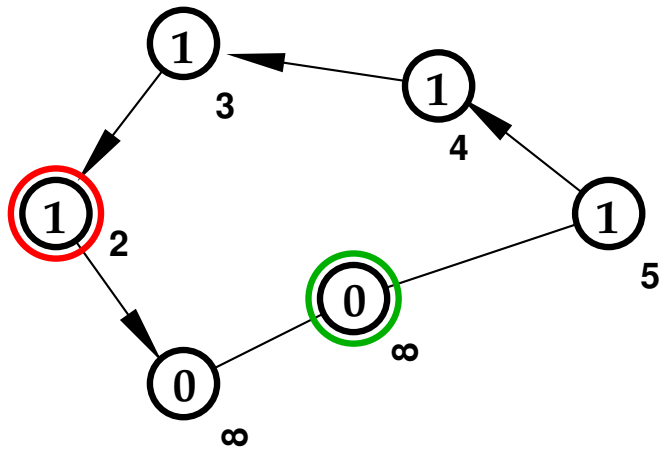
Arbitrary Start: Output = 0: Serious Problem!

Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.

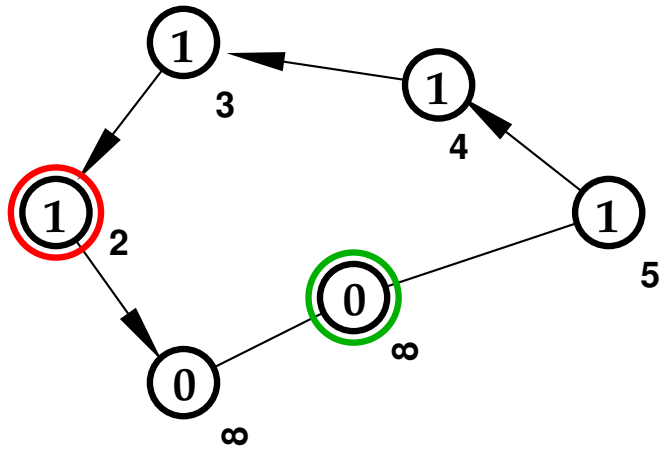
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- **Output Bits Inside Circles.**



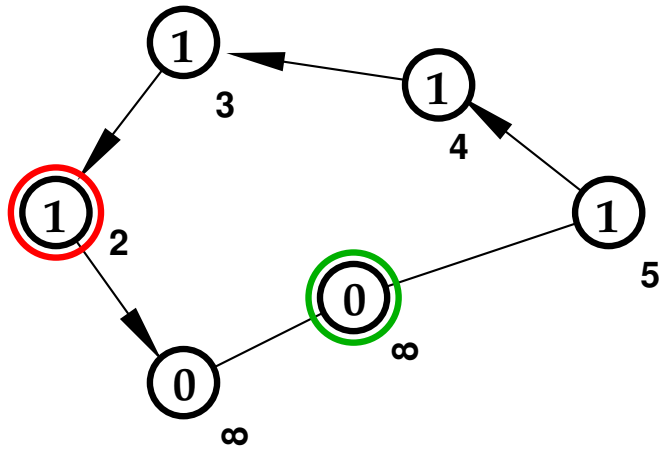
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.



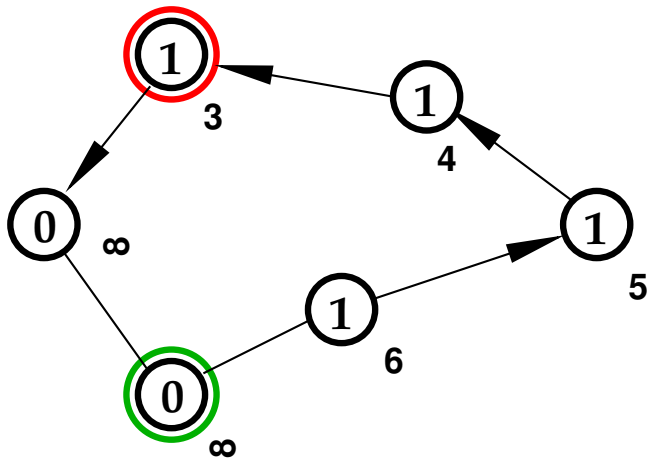
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.



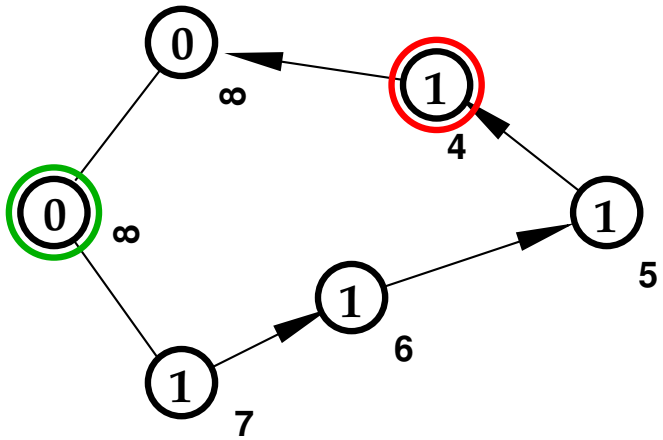
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- **Chain Deletes** at Head End.
- **But Recruits** at Tail (Leaf) End.



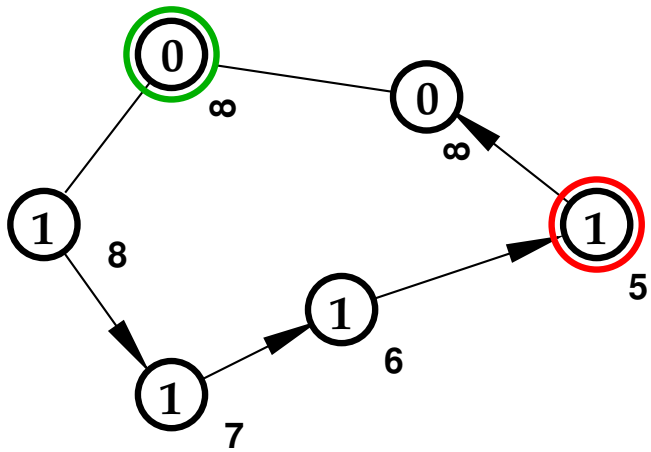
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Chain Deletes at Head End.
- But Recruits at Tail (Leaf) End.
- Keeps Going **Around!**



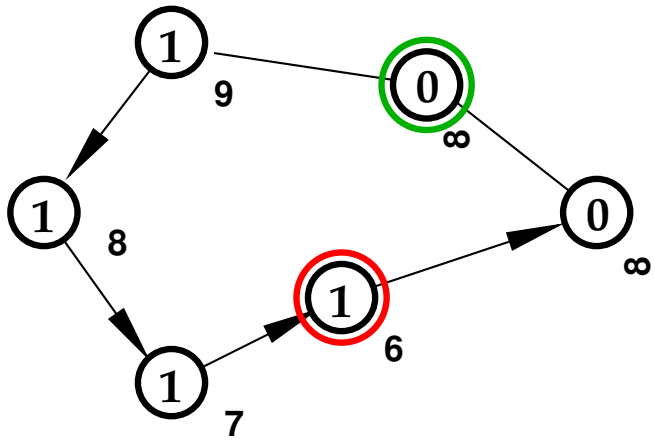
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Chain Deletes at Head End.
- But Recruits at Tail (Leaf) End.
- Keeps Going **Around!**



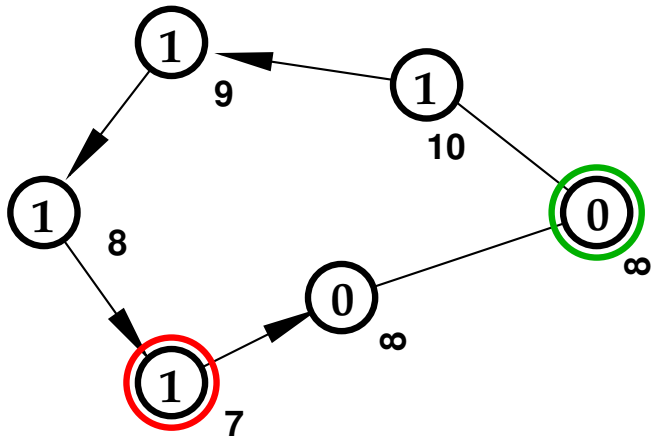
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Chain Deletes at Head End.
- But Recruits at Tail (Leaf) End.
- Keeps Going **Around!**



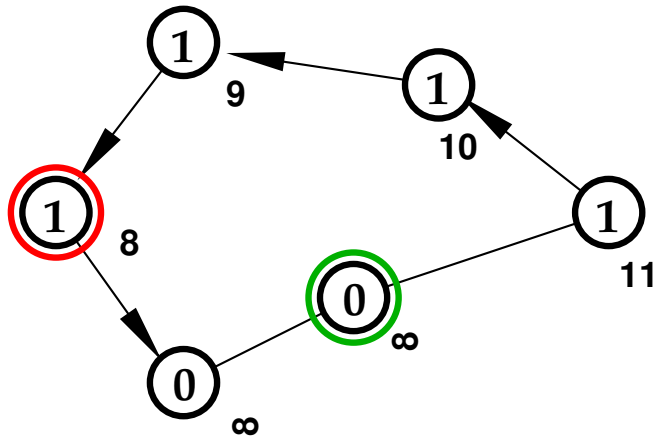
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Chain Deletes at Head End.
- But Recruits at Tail (Leaf) End.
- Keeps Going **Around!**



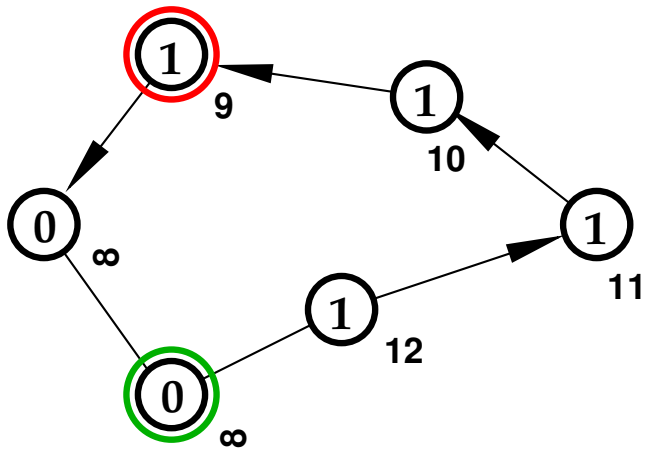
Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Chain Deletes at Head End.
- But Recruits at Tail (Leaf) End.
- Keeps Going Around.
- **Return** to **First Configuration**, Except for Levels.



Arbitrary Start: Output = 0: Serious Problem!

- All Input Bits 0.
- Output Bits Inside Circles.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Chain Deletes at Head End.
- But Recruits at Tail (Leaf) End.
- Keeps Going Around.
- Return to First Configuration, Except for Levels.
- **Endless!**



Purpose of Color Waves

Purpose of Color Waves

- **Prevent Indefinite Growth of Fictitious Trees.**

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- **Slow Down Algorithm by a Factor of Three**

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- Slow Down Algorithm by a Factor of Three
- **After Legitimacy, Color Waves could Run Forever.**

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- Slow Down Algorithm by a Factor of Three
- After Legitimacy, Color Waves could Run Forever.

Counteract Effect with Done Waves

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- Slow Down Algorithm by a Factor of Three
- After Legitimacy, Color Waves could Run Forever.

Counteract Effect with Done Waves

- **Convergecast: Leaves Detect Algorithm Done**

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- Slow Down Algorithm by a Factor of Three
- After Legitimacy, Color Waves could Run Forever.

Counteract Effect with Done Waves

- Convergecast: Leaves Detect Algorithm Done
- **Root (Clusterhead) Color Freezes.**

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- Slow Down Algorithm by a Factor of Three
- After Legitimacy, Color Waves could Run Forever.

Counteract Effect with Done Waves

- Convergecast: Leaves Detect Algorithm Done
- Root (Clusterhead) Color Freezes.
- **Color Lock Results Within $O(Diam)$ Rounds.**

Purpose of Color Waves

- Prevent Indefinite Growth of Fictitious Trees.

Side Effects of Color Waves

- Slow Down Algorithm by a Factor of Three
- After Legitimacy, Color Waves could Run Forever.

Counteract Effect with Done Waves

- Convergecast: Leaves Detect Algorithm Done
- Root (Clusterhead) Color Freezes.
- Color Lock Results Within $O(Diam)$ Rounds.
- **Silence.**

Color Wave Details:

Color Wave Details:

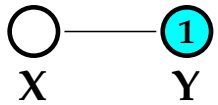
- Process with **Output Bit = 1** has **Color:** 0 = ①, 1 = ①.

Color Wave Details:

- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes **Join**, Attaching to Process **Y**:

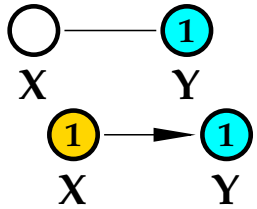
Color Wave Details:

- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes **Join**, Attaching to Process **Y**:
  - **Y** must have Color 1:



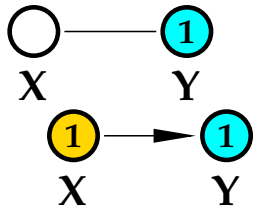
Color Wave Details:

- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes **Join**, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:



Color Wave Details:

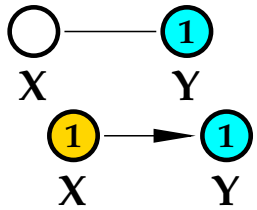
- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes Join, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:



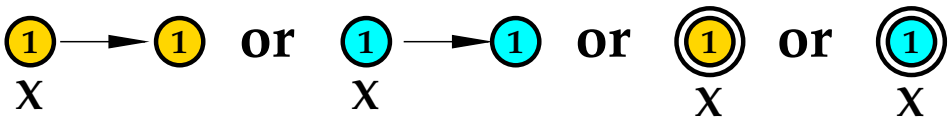
- Process **X** can **Change Color** if the **Following Conditions Hold:**

Color Wave Details:

- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes Join, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:

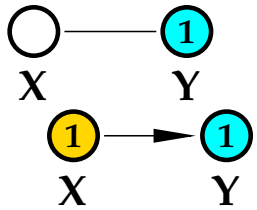


- Process **X** can **Change Color** if the **Following Conditions Hold:**
  - **X** has **Same Color** as its **Parent**, or is **Clusterhead:**

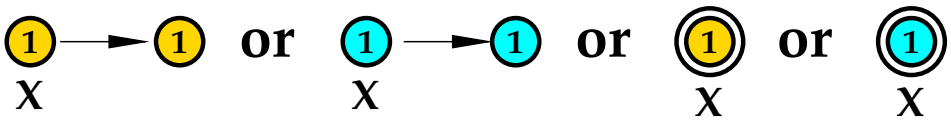


Color Wave Details:

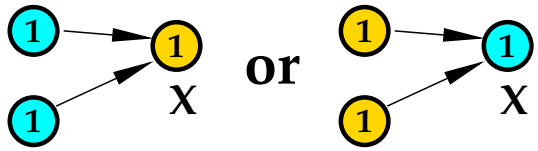
- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes Join, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:



- Process **X** can **Change Color** if the **Following Conditions Hold:**
  - **X** has Same Color as its Parent, or is Clusterhead:

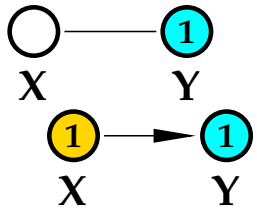


  - **Children** have **Opposite Color**:

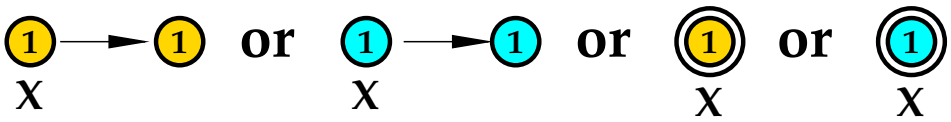


Color Wave Details:

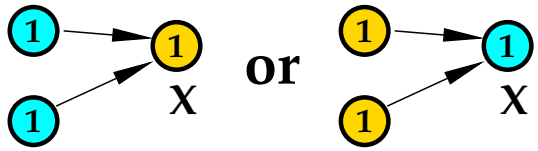
- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes Join, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:



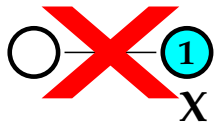
- Process **X** can **Change Color** if the **Following Conditions Hold:**
  - **X** has Same Color as its Parent, or is Clusterhead:



  - Children have Opposite Color:

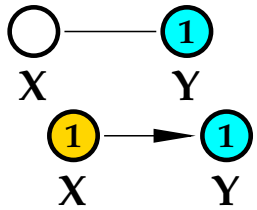


  - If **Color = 1, X Cannot Change Color** if Any Neighbor is Enabled to Attach to **X:**

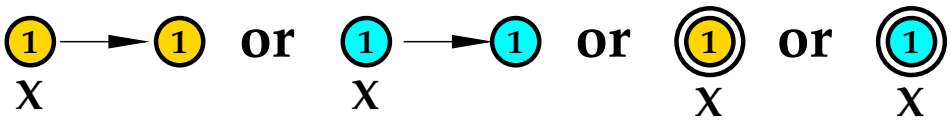


Color Wave Details:

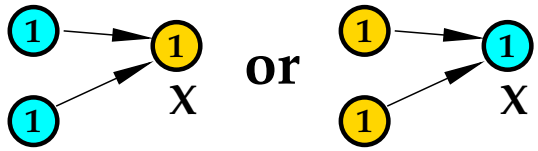
- Process with Output Bit = 1 has Color: 0 = (1), 1 = (1).
- If Process **X** Executes Join, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:



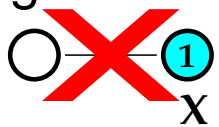
- Process **X** can Change Color if the Following Conditions Hold:
  - **X** has Same Color as its Parent, or is Clusterhead:



  - Children have Opposite Color:



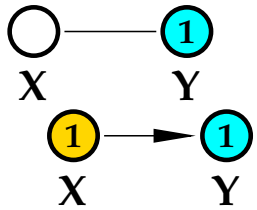
  - If Color = 1: **X** Cannot Change Color if Any Neighbor is Enabled to Attach to **X:**



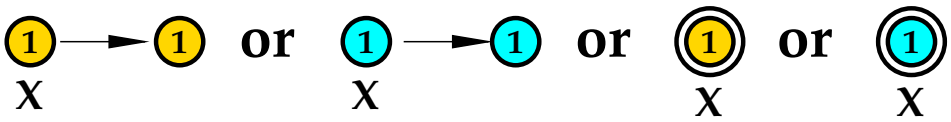
- When **Clusterhead** Changes Color, a **Color Wave** is **Absorbed.**

Color Wave Details:

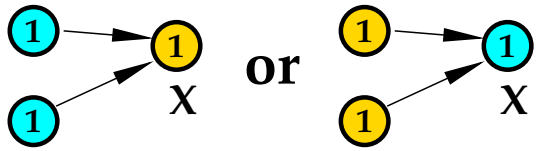
- Process with Output Bit = 1 has Color: 0 = ①, 1 = ①.
- If Process **X** Executes Join, Attaching to Process **Y**:
  - **Y** must have Color 1:
  - Color of **X** Becomes 0:



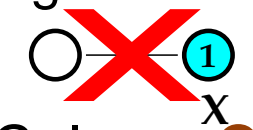
- Process **X** can Change Color if the Following Conditions Hold:
  - **X** has Same Color as its Parent, or is Clusterhead:



  - Children have Opposite Color:



  - If Color = 1: **X** Cannot Change Color if Any Neighbor is Enabled to Attach to **X:**

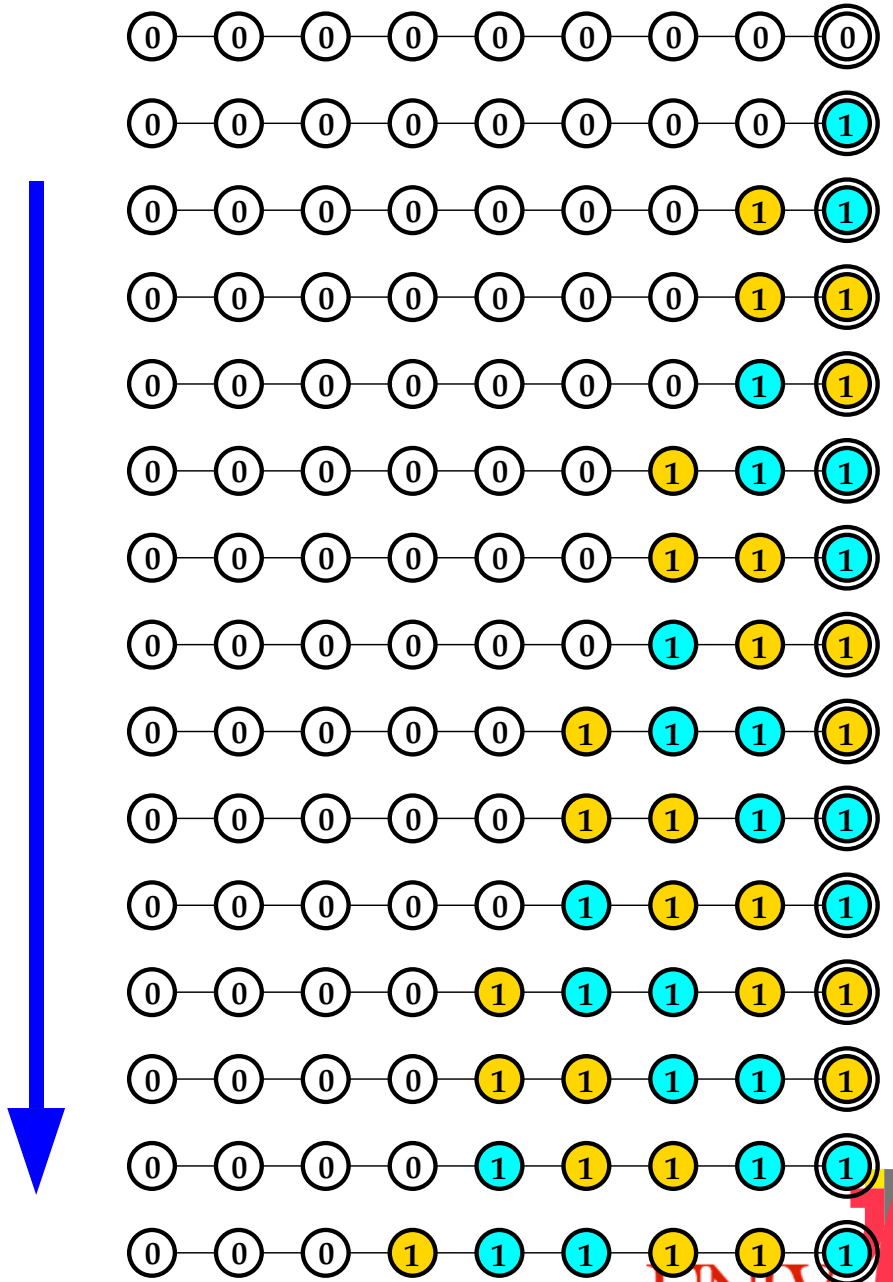


- When **Clusterhead** Changes Color, a **Color Wave** is **Absorbed**.
  - **False Roots Cannot Absorb Color Waves**.

Color Waves:

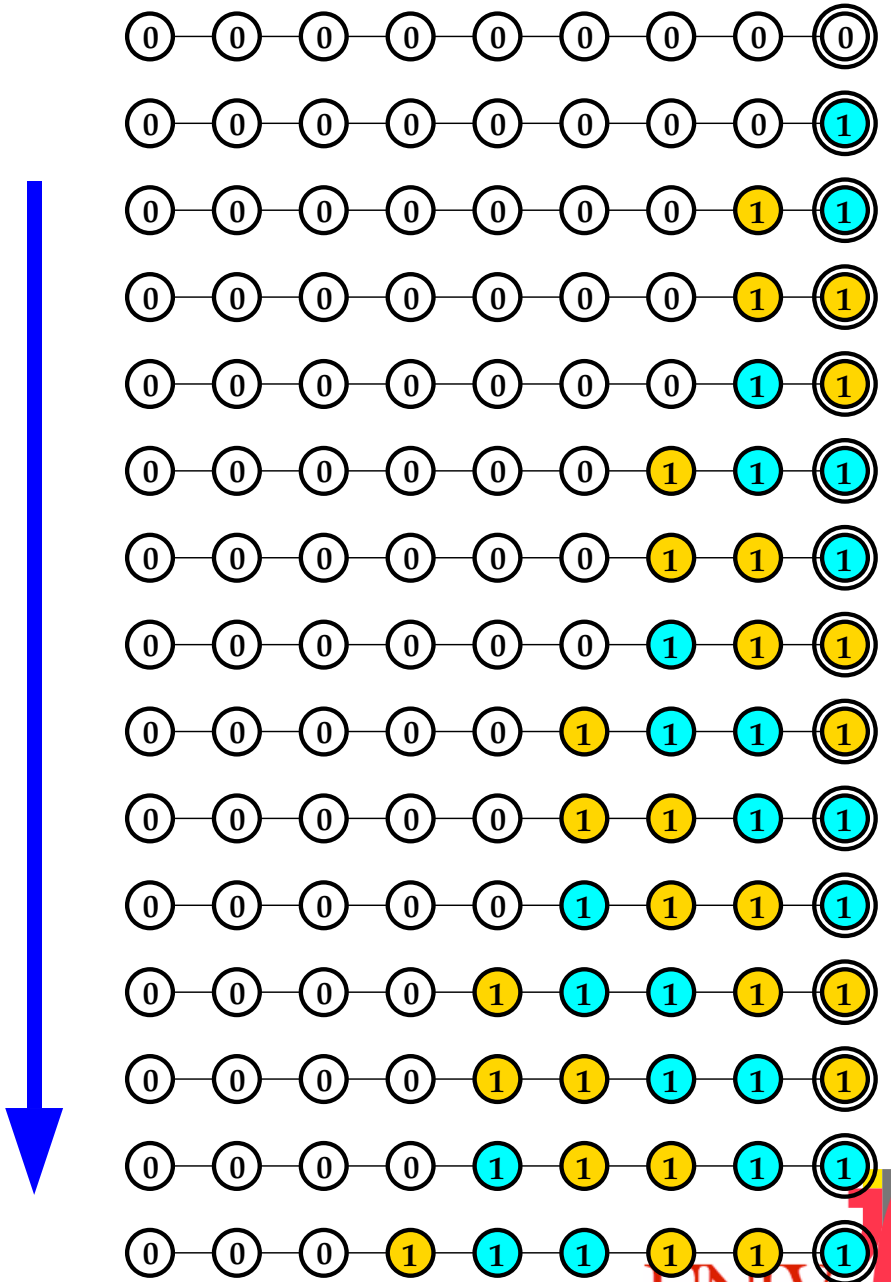
Color Waves:

- **Chain Example.**



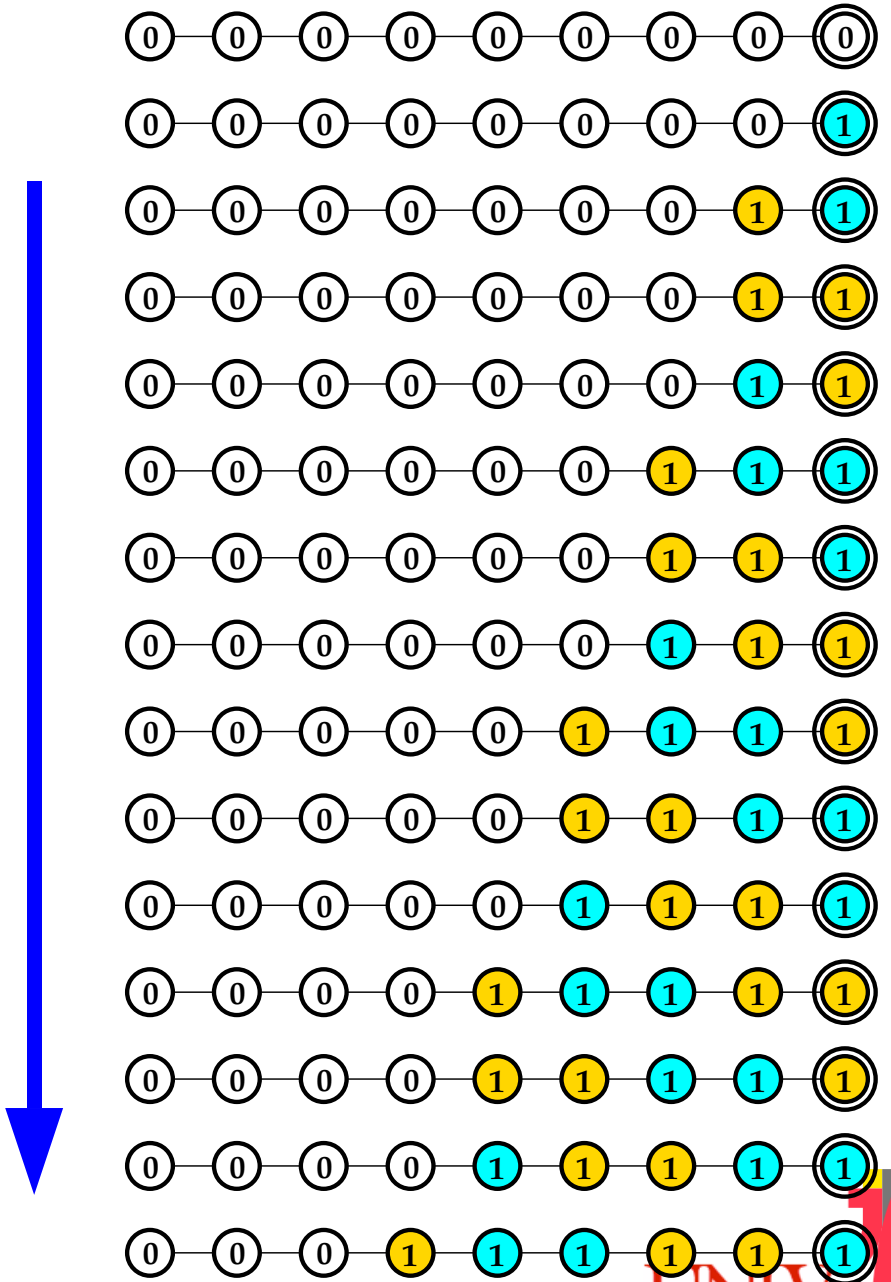
Color Waves:

- Chain Example.
- **One Process has Input = 1.**



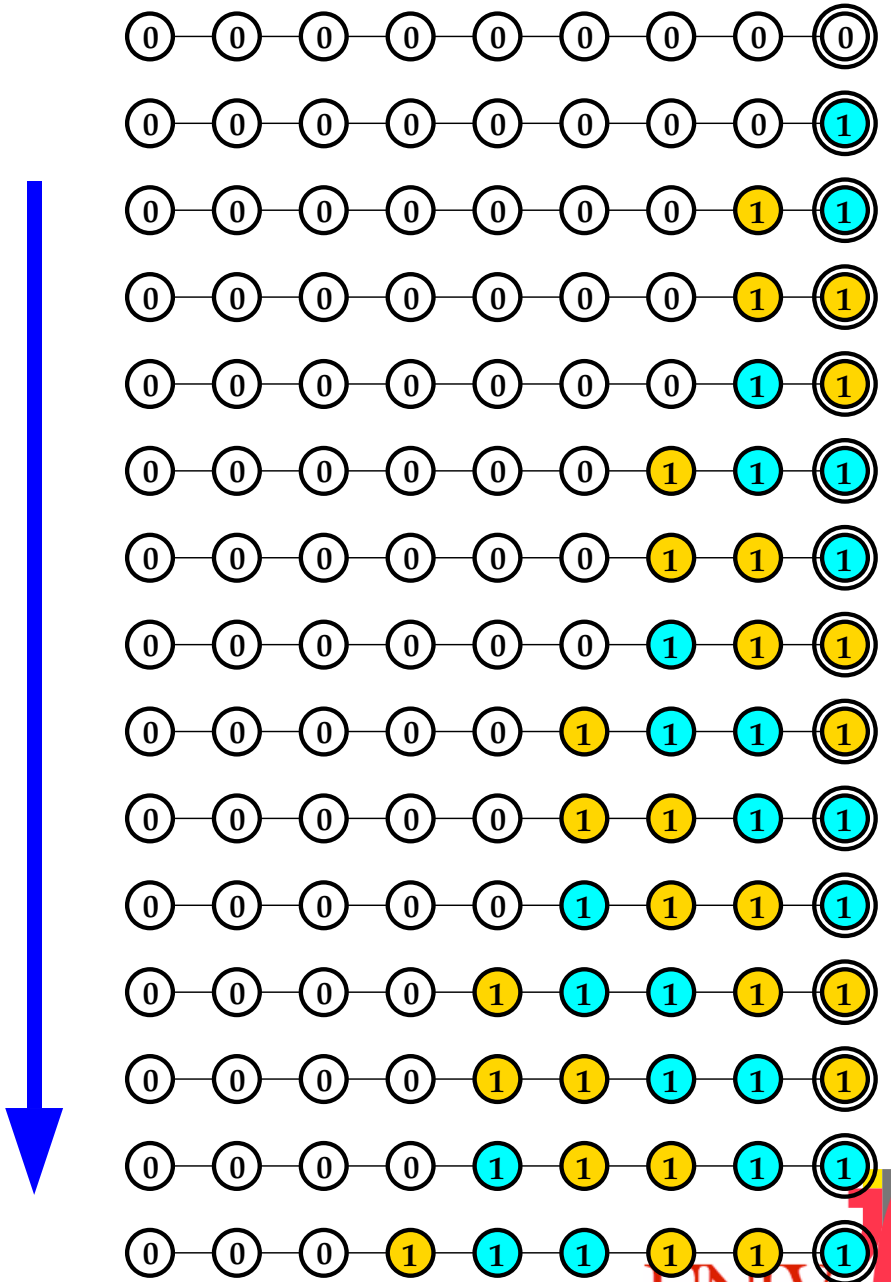
Color Waves:

- Chain Example.
- One Process has Input = 1.
- **Arrow** Shows Flow of Time.



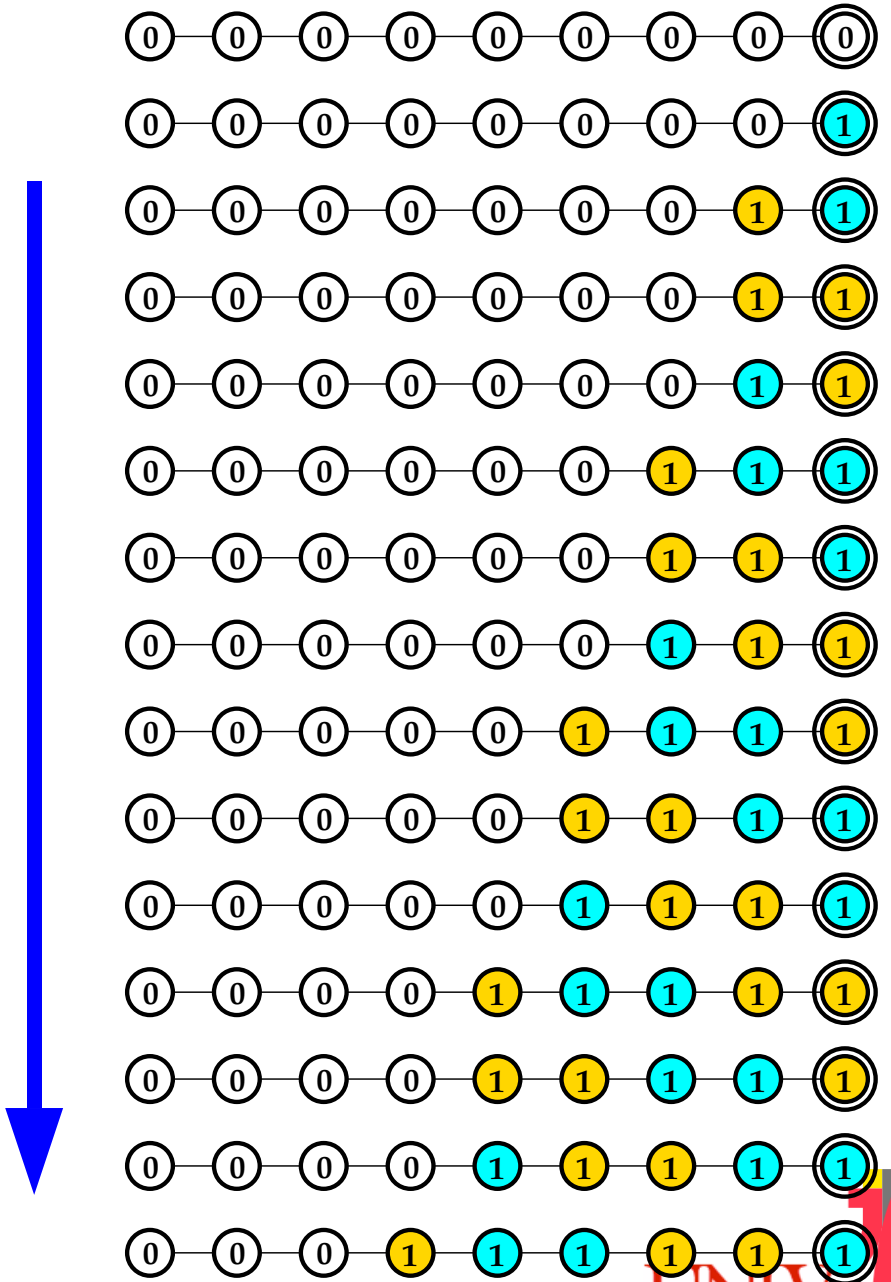
Color Waves:

- Chain Example.
- One Process has Input = 1.
- Arrow Shows Flow of Time.
- **Color Waves** Move
  Toward Clusterhead



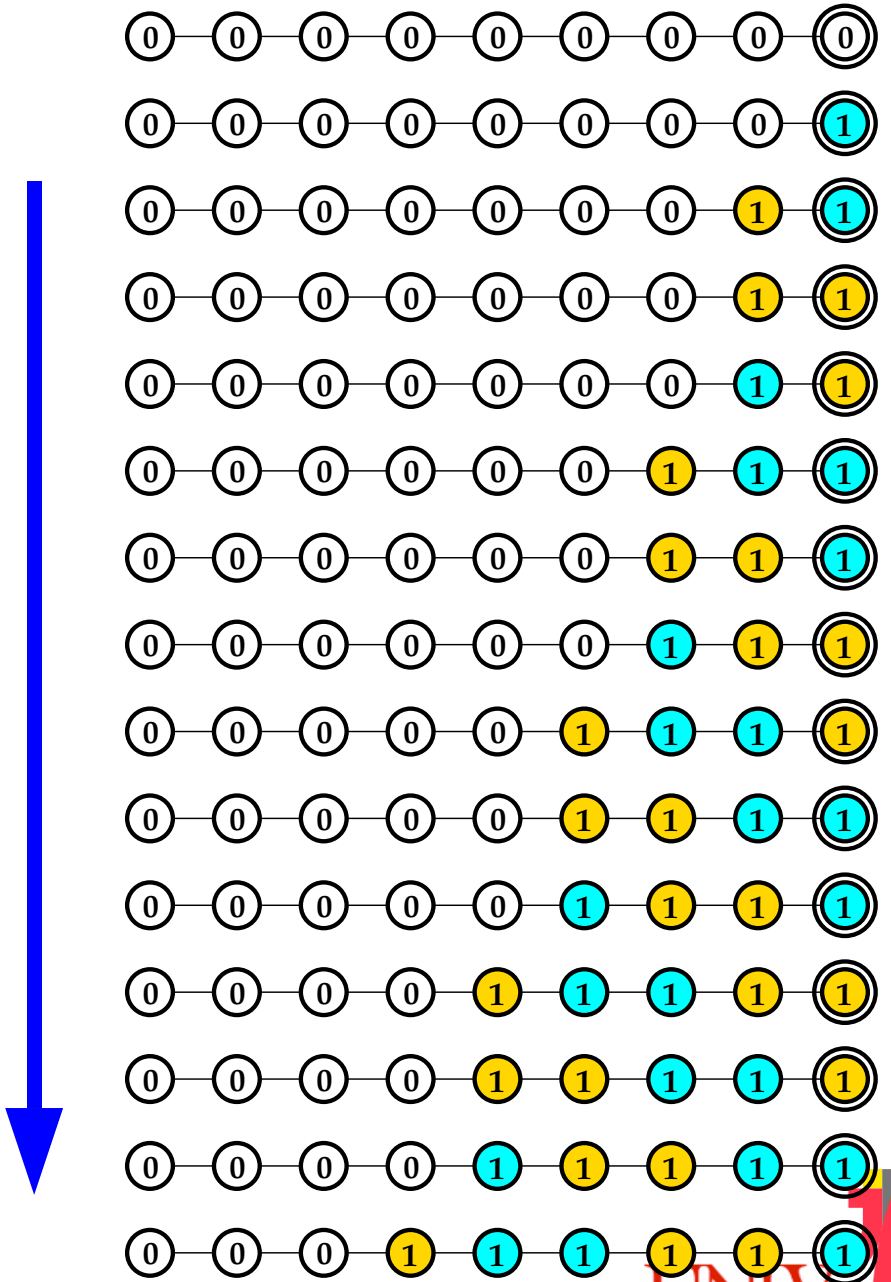
Color Waves:

- Chain Example.
- One Process has Input = 1.
- Arrow Shows Flow of Time.
- Color Waves Move
  Toward Clusterhead
- **Growth Rate = 1/3**.



Color Waves:

- Chain Example.
- One Process has Input = 1.
- Arrow Shows Flow of Time.
- Color Waves Move
  Toward Clusterhead
- Growth Rate = 1/3.



Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Getting Rid of False Roots:

- Consider a **False Root**, **R**.
- **R** is Enabled Only to **Reset**, and thus Cannot Change Color.
- Eventually **Color Lock**.
- Tree Rooted at **R** Cannot Grow Forever.
- **How can you Prove That?**
- **Use Energy!**

Energy

Energy

- **Energy(X): Positive Integer** for **X** of **Output = 1**.

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined **Recursively.**

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined **Recursively.**
  - **Energy(X)** $= 1$ if **X** is **Leaf** of **Color = 0.**

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined **Recursively.**
  - Energy(**X**) $=$ 1 if **X** is Leaf of Color = 0.
  - **Energy(X)** $=$ 2 if **X** is **Leaf** of **Color = 1.**

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined **Recursively.**
  - Energy(**X**) $=$ 1 if **X** is Leaf of Color = 0.
  - Energy(**X**) $=$ 2 if **X** is Leaf of Color = 1.
  - **X Not Leaf: Energy**(**X**) = **Maximum** of:

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined **Recursively.**
  - Energy(**X**) $=$ 1 if **X** is Leaf of Color = 0.
  - Energy(**X**) $=$ 2 if **X** is Leaf of Color = 1.
  - **X Not Leaf: Energy**(**X**) = **Maximum** of:
    - **1 + Energy** of any **Child of Opposite Color**.

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined **Recursively.**
  - Energy(**X**) $=$ 1 if **X** is Leaf of Color = 0.
  - Energy(**X**) $=$ 2 if **X** is Leaf of Color = 1.
  - **X Not Leaf: Energy**(**X**) = **Maximum** of:
    - 1 + Energy of any Child of Opposite Color.
    - **2 + Energy** of any **Child of Matching Color**.

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined Recursively.
  - Energy(**X**) $= 1$ if **X** is Leaf of Color = 0.
  - Energy(**X**) $= 2$ if **X** is Leaf of Color = 1.
  - **X** Not Leaf: Energy(**X**) = Maximum of:
    - 1 + Energy of any Child of Opposite Color.
    - 2 + Energy of any Child of Matching Color.
- **Theorem:** No Action of a **Process of Input = 0** can **Increase Maximum Energy** of the Network.

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined Recursively.
  - Energy(**X**) $= 1$ if **X** is Leaf of Color = 0.
  - Energy(**X**) $= 2$ if **X** is Leaf of Color = 1.
  - **X** Not Leaf: Energy(**X**) = Maximum of:
    - 1 + Energy of any Child of Opposite Color.
    - 2 + Energy of any Child of Matching Color.
- Theorem: No Action of a Process of Input = 0 can Increase Maximum Energy of the Network.
- **Theorem:** If **All Processes have Input = 0:** **Maximum Energy** of the Network Decreases every **Round.** Hence **Convergence** After $O(n)$ **Rounds.**

Energy

- Energy(**X**): Positive Integer for **X** of Output = 1.
- Defined Recursively.
  - Energy(**X**) $= 1$ if **X** is Leaf of Color = 0.
  - Energy(**X**) $= 2$ if **X** is Leaf of Color = 1.
  - **X** Not Leaf: Energy(**X**) = Maximum of:
    - 1 + Energy of any Child of Opposite Color.
    - 2 + Energy of any Child of Matching Color.
- Theorem: No Action of a Process of Input = 0 can Increase Maximum Energy of the Network.
- Theorem: If All Processes have Input = 0: Maximum Energy of the Network Decreases every Round, Hence Convergence After $O(n)$ Rounds.

Silence

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**
- How do we Stop **Color Waves** from **Continuing Forever?**

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**
- How do we Stop Color Waves from Continuing Forever?
- **Done Waves:**

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**
- How do we Stop Color Waves from Continuing Forever?
- **Done Waves:**
  - **Leaf** Initiates when it **Detects** (Local) **Legitimacy.**

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**
- How do we Stop Color Waves from Continuing Forever?
- **Done Waves:**
  - Leaf Initiates when it Detects (Local) Legitimacy.
  - Done Wave **Convergecast.**

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**
- How do we Stop Color Waves from Continuing Forever?
- **Done Waves:**
  - Leaf Initiates when it Detects (Local) Legitimacy.
  - Done Wave Convergecast.
  - **Clusterhead** (That is, Process with Input = 1) Becomes **Color Frozen.** Cannot Change Color.

Silence

- **Legitimate Configuration** After $O(n)$ **rounds**
- How do we Stop Color Waves from Continuing Forever?
- **Done Waves:**
  - Leaf Initiates when it Detects (Local) Legitimacy.
  - Done Wave Convergecast.
  - Clusterhead (That is, Process with Input = 1) Becomes Color Frozen. Cannot Change Color.
  - **Color Lock:** Within $O(Diam)$ Rounds: **Silence** is Achieved.

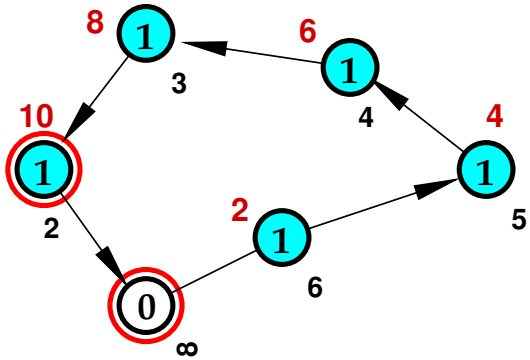
Arbitrary Start: Output = 0: Problem Solved!

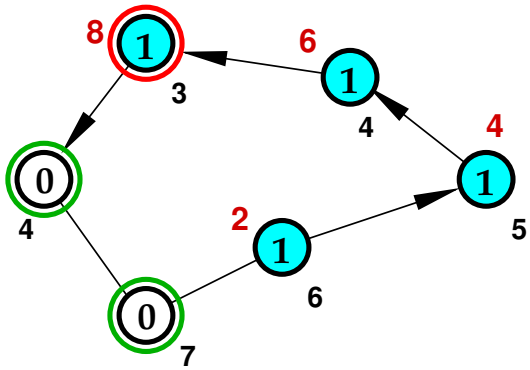
Arbitrary Start: Output = 0: Problem Solved!

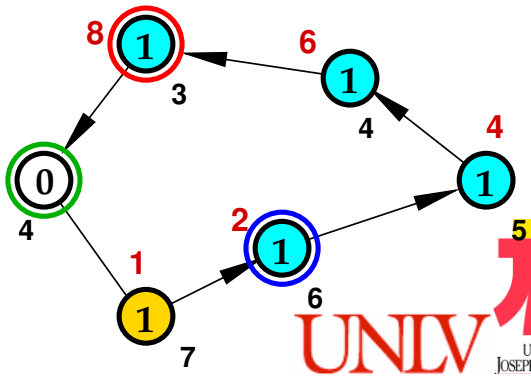
- **Asynchronous Example Computation**

Arbitrary Start: Output = 0: Problem Solved!

- Asynchronous Example Computation
- All Input Bits 0.

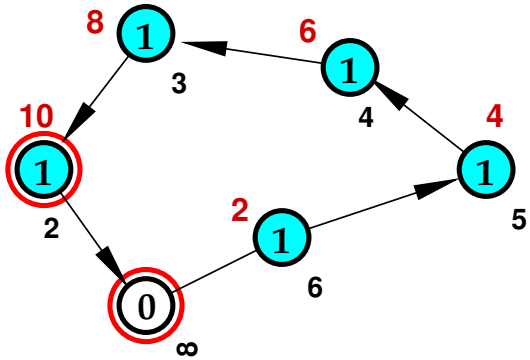


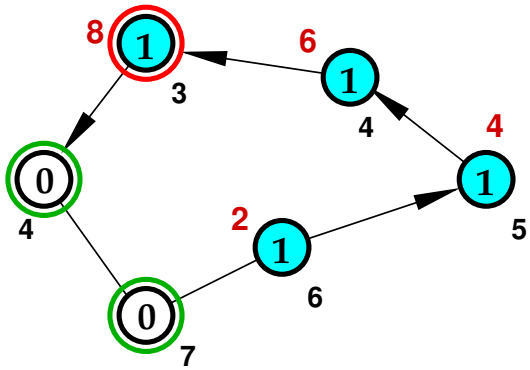


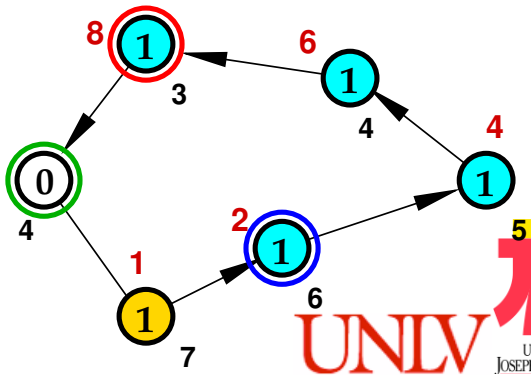


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.

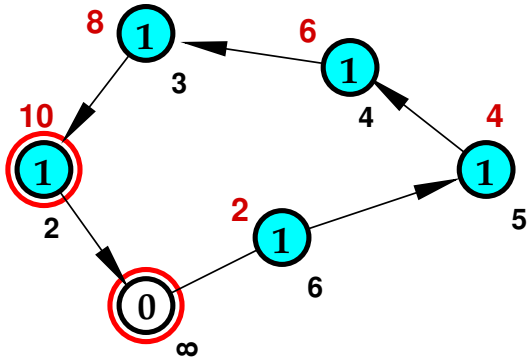


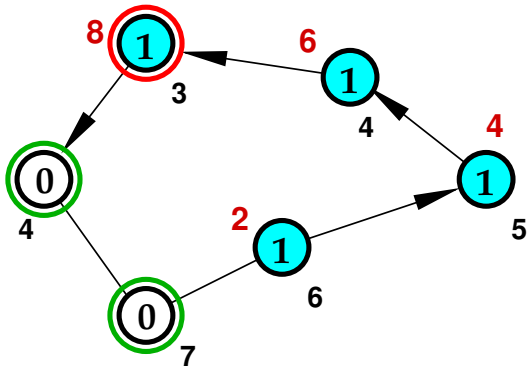


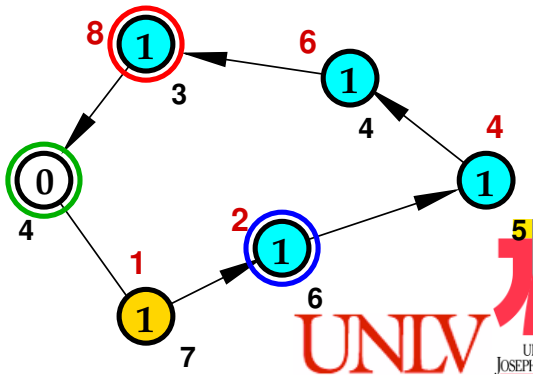


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.

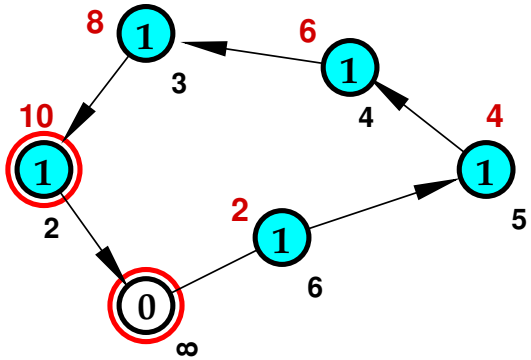


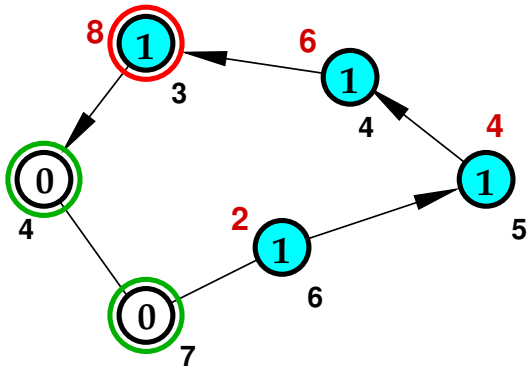


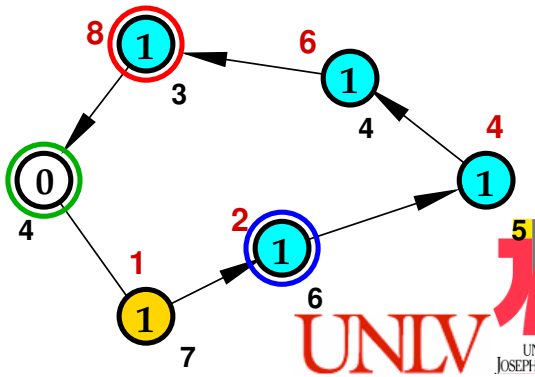


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.

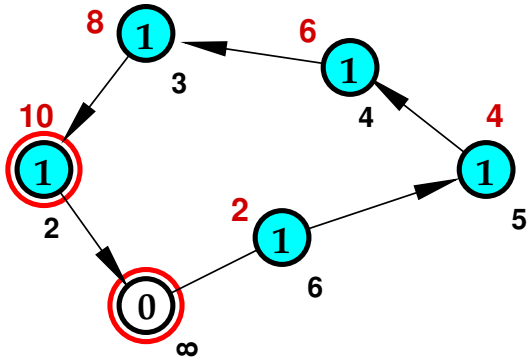


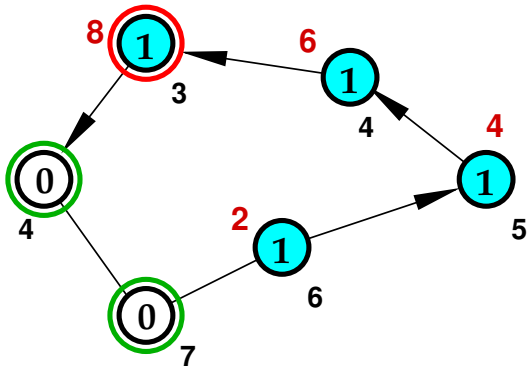


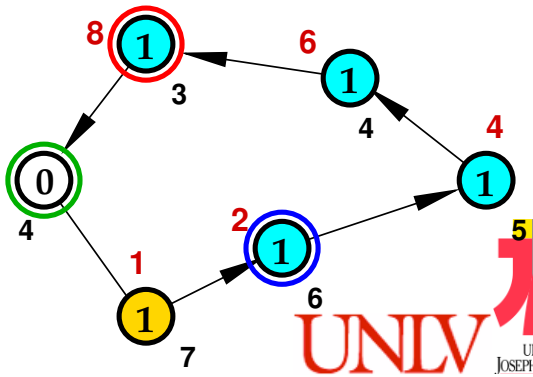


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum **Energy** Initially = 10.

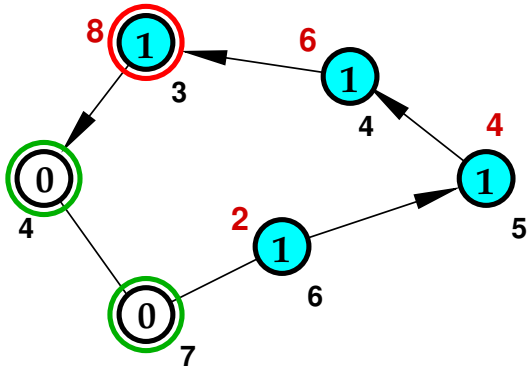


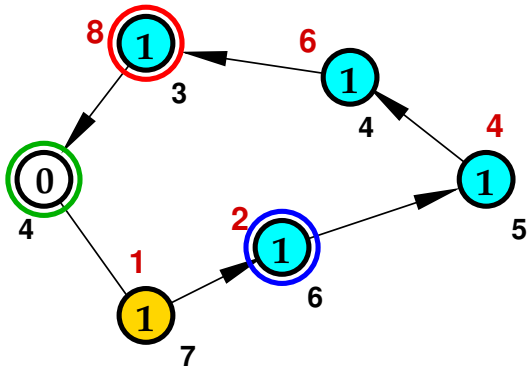


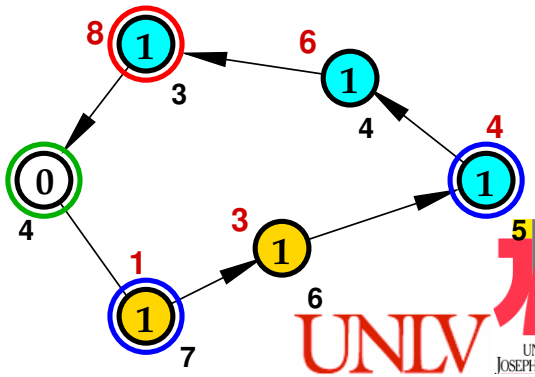


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum **Energy** Initially = 10.

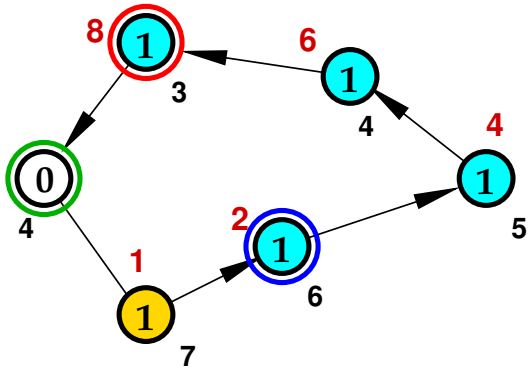


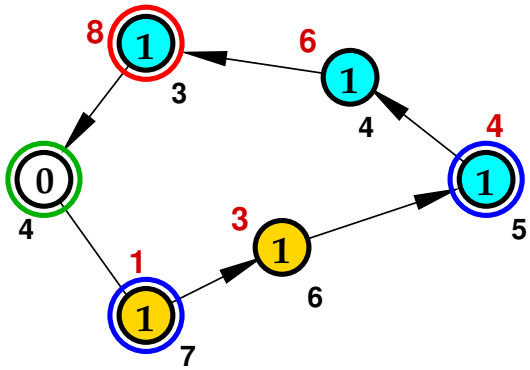


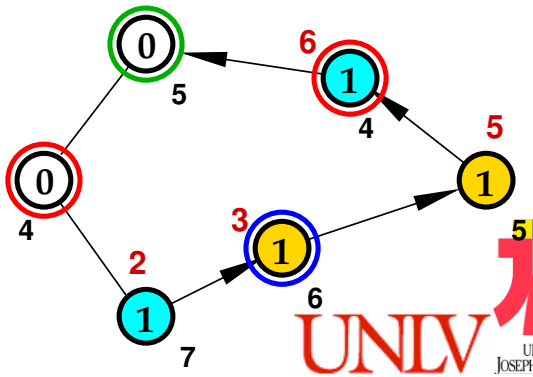


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum **Energy Decreases** each **Round.**

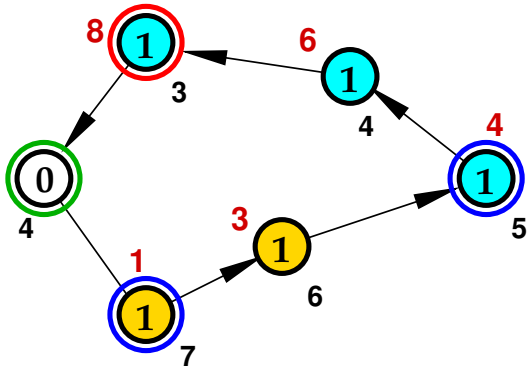


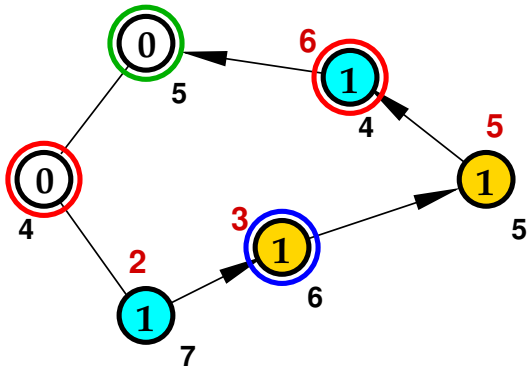


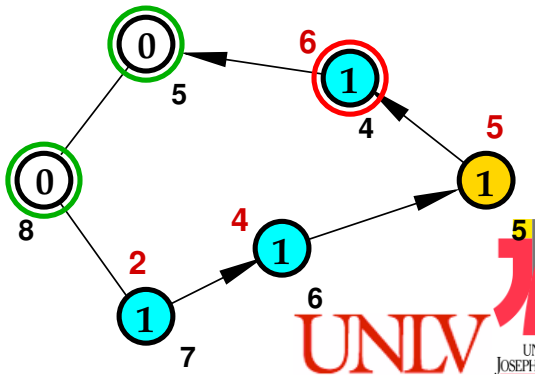


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum **Energy Decreases** each **Round.**

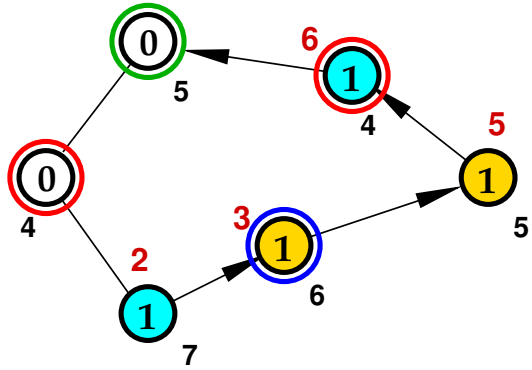


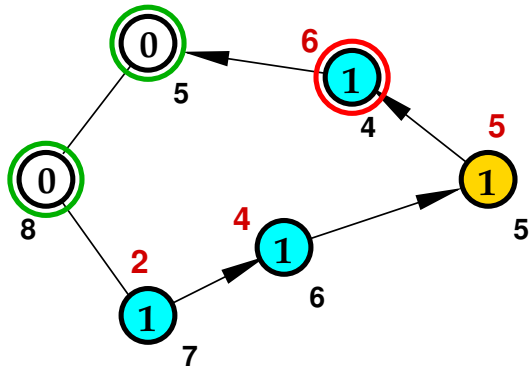


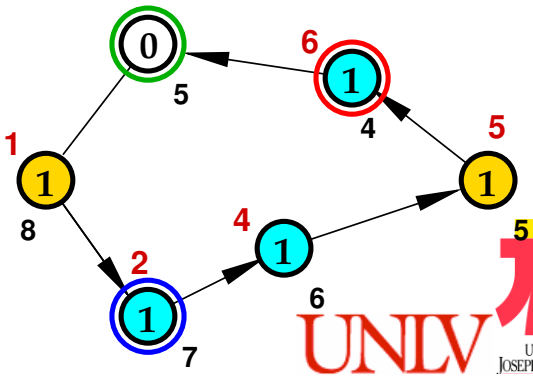


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum **Energy Decreases** each **Round.**

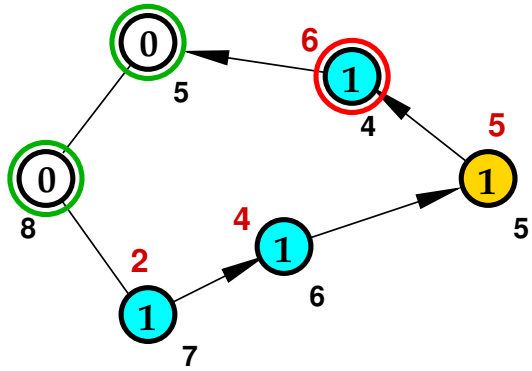


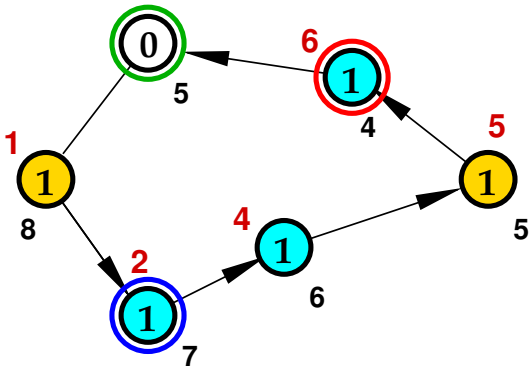


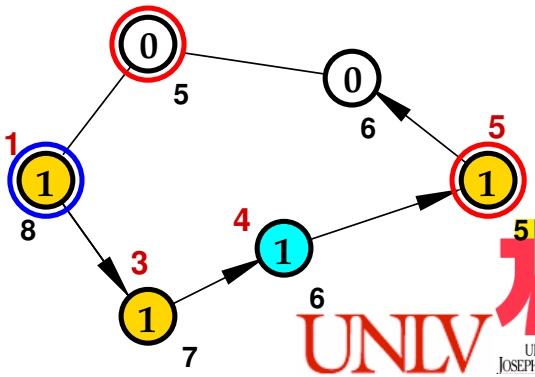


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum **Energy Decreases** each **Round.**

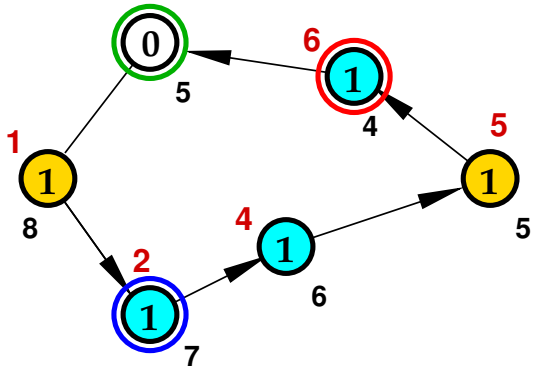


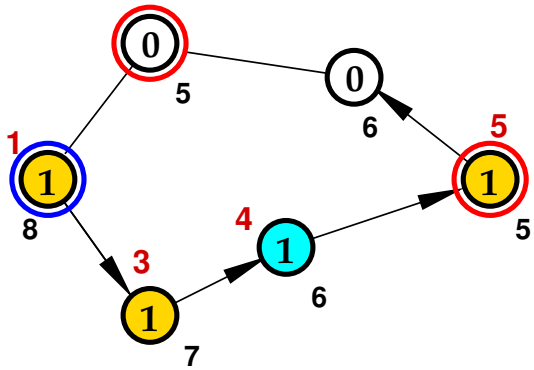


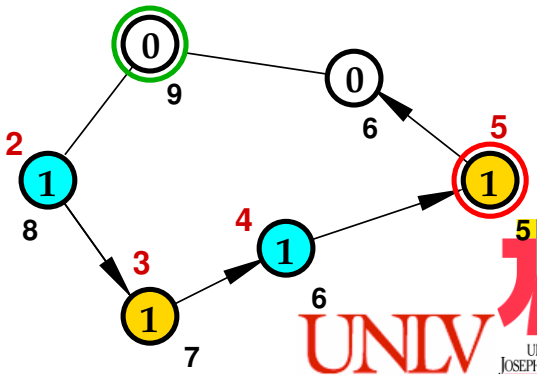


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum **Energy Decreases** each **Round.**

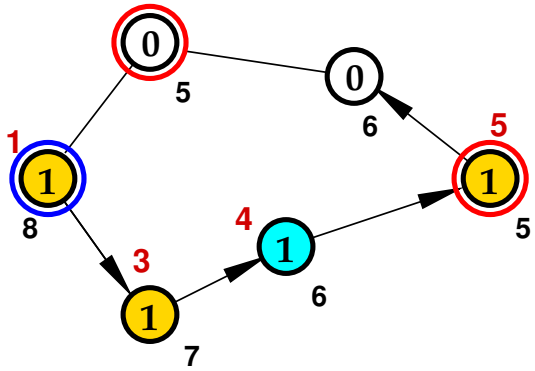


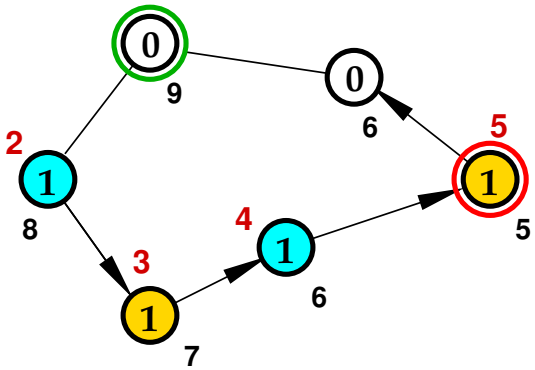


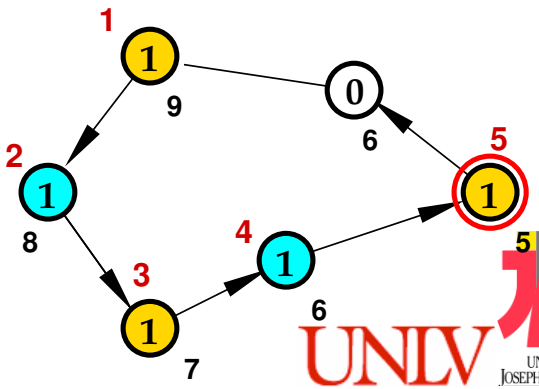


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- **No Further Growth Possible.**

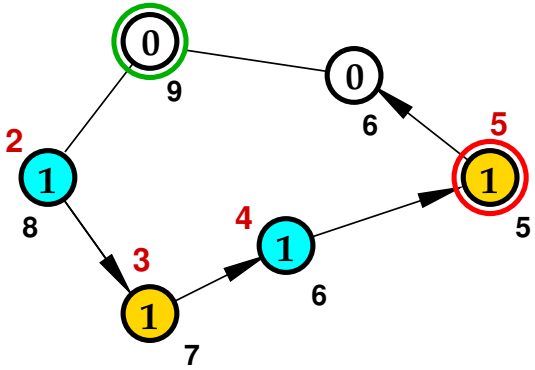


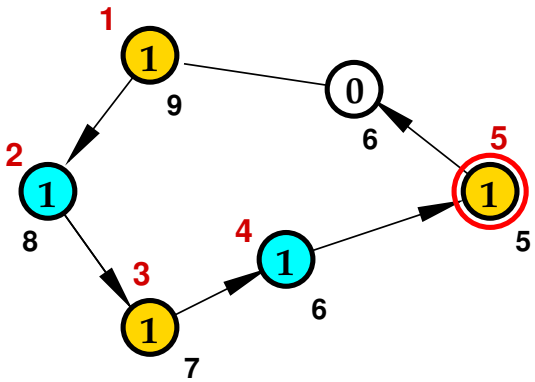


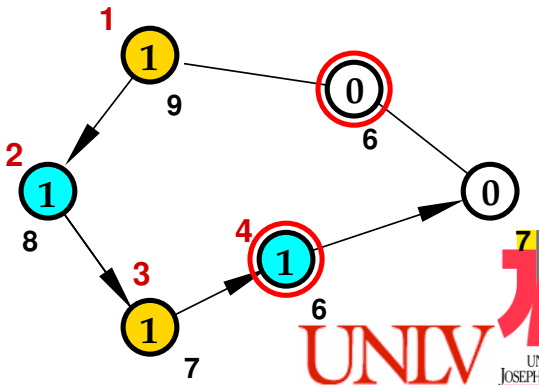


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- **No Further Growth Possible.**

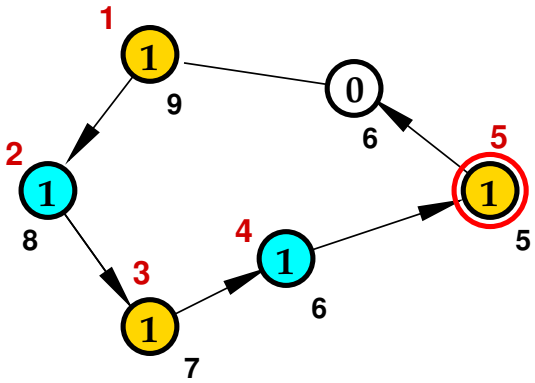


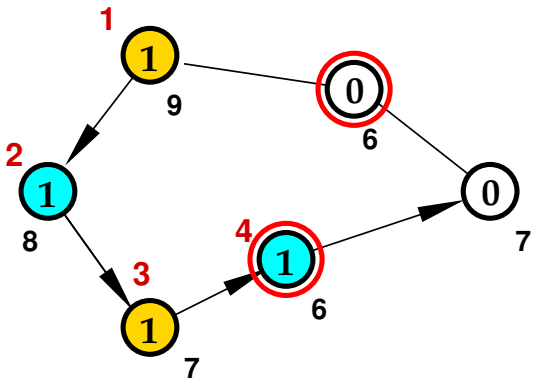


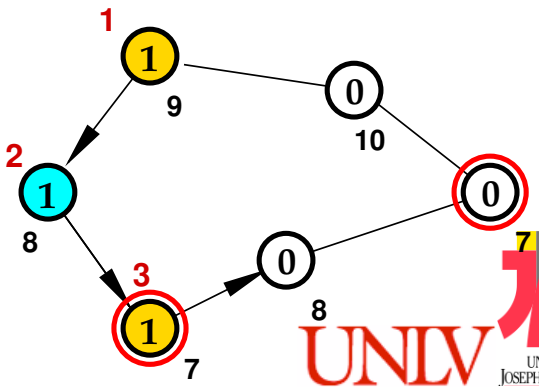


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- **No Further Growth Possible.**

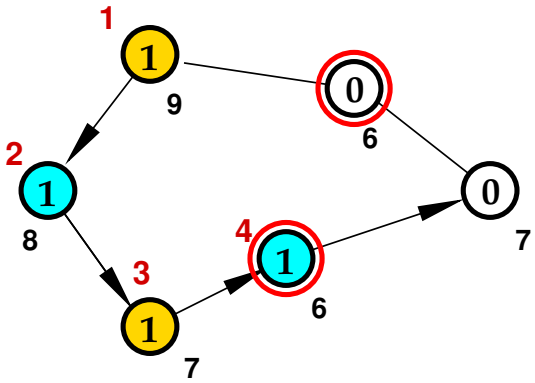


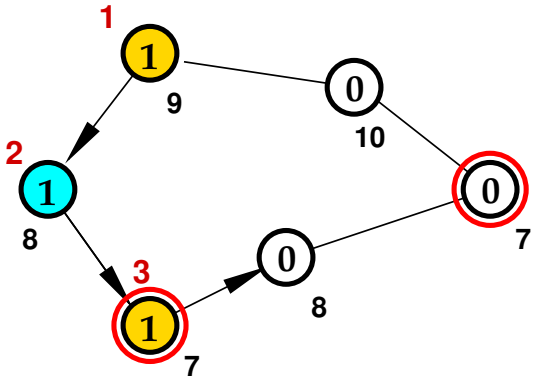


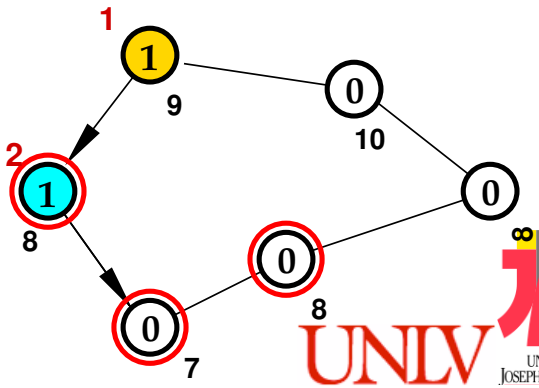


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- **No Further Growth Possible.**

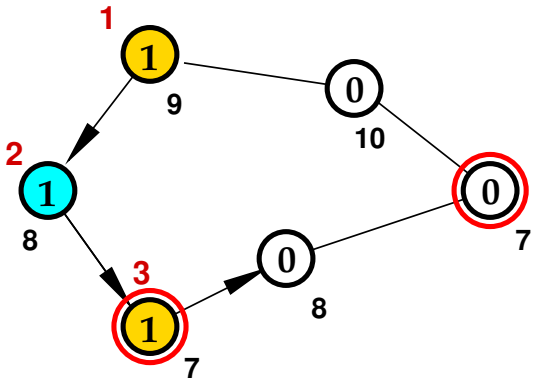


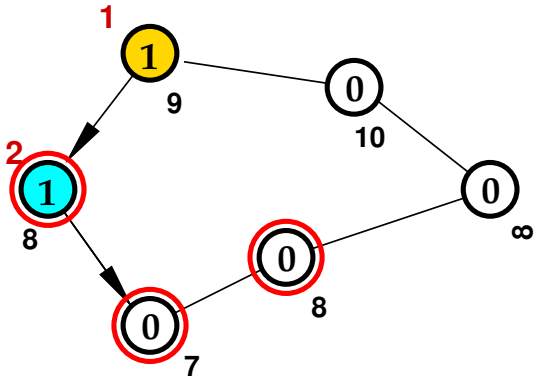


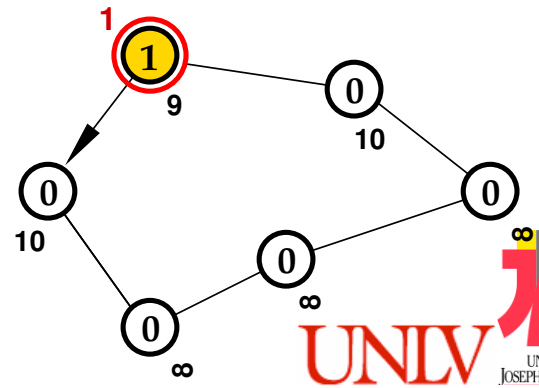


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- **No Further Growth Possible.**

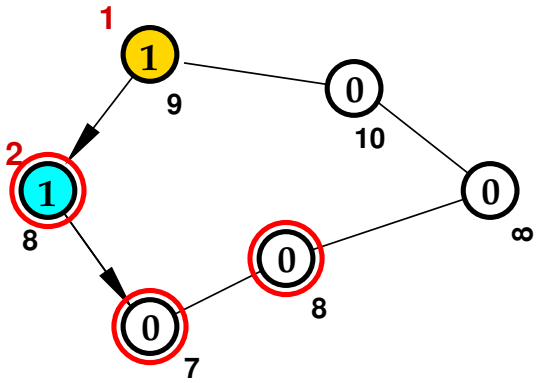


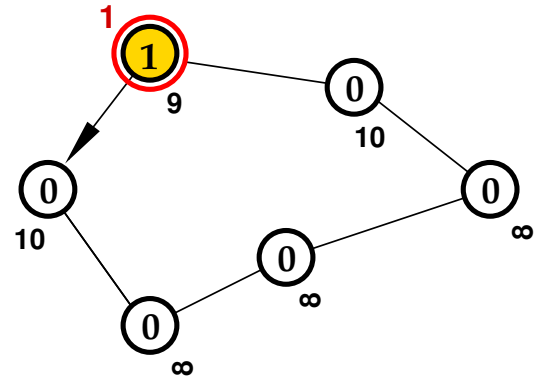


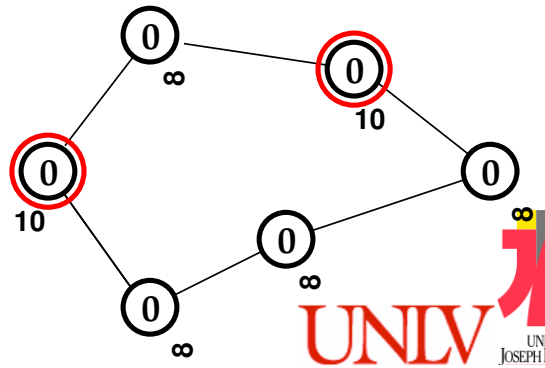


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- No Further Growth Possible.
- Configuration is Now **Legitimate.**

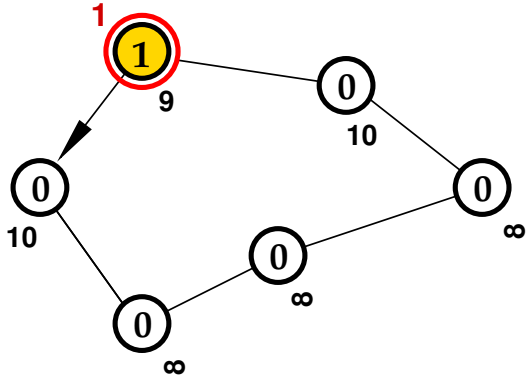


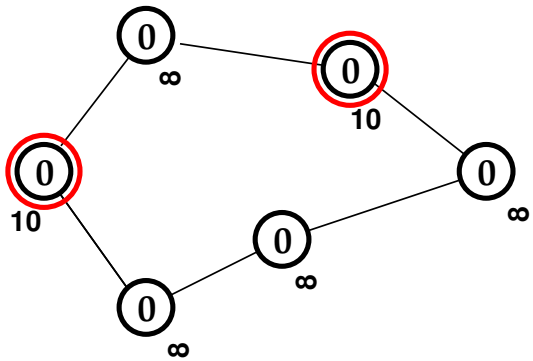


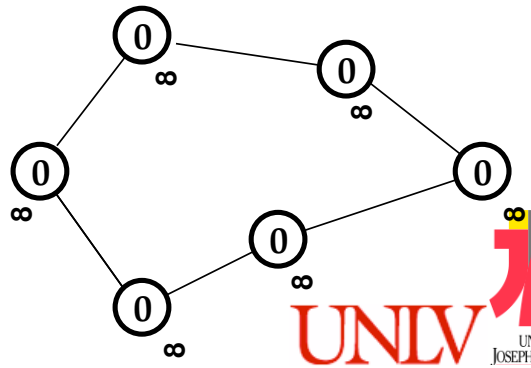


Arbitrary Start: Output = 0: Problem Solved!

- **Asynchronous Example Computation**
- All Input Bits 0.
- Red = Enabled to Reset.
- Green = Enabled to Join.
- Blue = Enabled to Change Color.
- Maximum Energy Initially = 10.
- Maximum Energy Decreases each Round.
- No Further Growth Possible.
- Configuration is Now Legitimate.
- Configuration is Now **Final. Silent.**







Conclusion

- We introduced the Disjunction Problem
- Application: Clustering
- Solution:
  - Locally Shared Memory Model (Unfair Daemon)
  - $O(\log Diam + \log \Delta)$ space per process
  - Stabilization Time: $O(n)$ rounds
- Stabilization Time in $O(Diam)$ rounds ?
  - Should be hard!
  - As difficult as *Self-Stabilizing Leader Election in identified networks* in $O(Diam)$ rounds ...

Conclusion

- We introduced the Disjunction Problem
- Application: Clustering
- Solution:
  - Locally Shared Memory Model (Unfair Daemon)
  - $O(\log Diam + \log \Delta)$ space per process
  - Stabilization Time: $O(n)$ rounds
- Stabilization Time in $O(Diam)$ rounds ?
  - Should be hard!
  - As difficult as *Self-Stabilizing Leader Election in identified networks* in $O(Diam)$ rounds ...

Conclusion

- We introduced the Disjunction Problem
- Application: Clustering
- Solution:
  - Locally Shared Memory Model (Unfair Daemon)
  - $O(\log Diam + \log \Delta)$ space per process
  - Stabilization Time: $O(n)$ rounds
- Stabilization Time in $O(Diam)$ rounds ?
  - Should be hard!
  - As difficult as *Self-Stabilizing Leader Election in identified networks* in $O(Diam)$ rounds ...

Conclusion

- We introduced the Disjunction Problem
- Application: Clustering
- Solution:
  - Locally Shared Memory Model (Unfair Daemon)
  - $O(\log Diam + \log \Delta)$ space per process
  - Stabilization Time: $O(n)$ rounds
- Stabilization Time in $O(Diam)$ rounds ?
  - Should be hard!
  - As difficult as *Self-Stabilizing Leader Election in identified networks* in $O(Diam)$ rounds ...

Conclusion

Thank You!