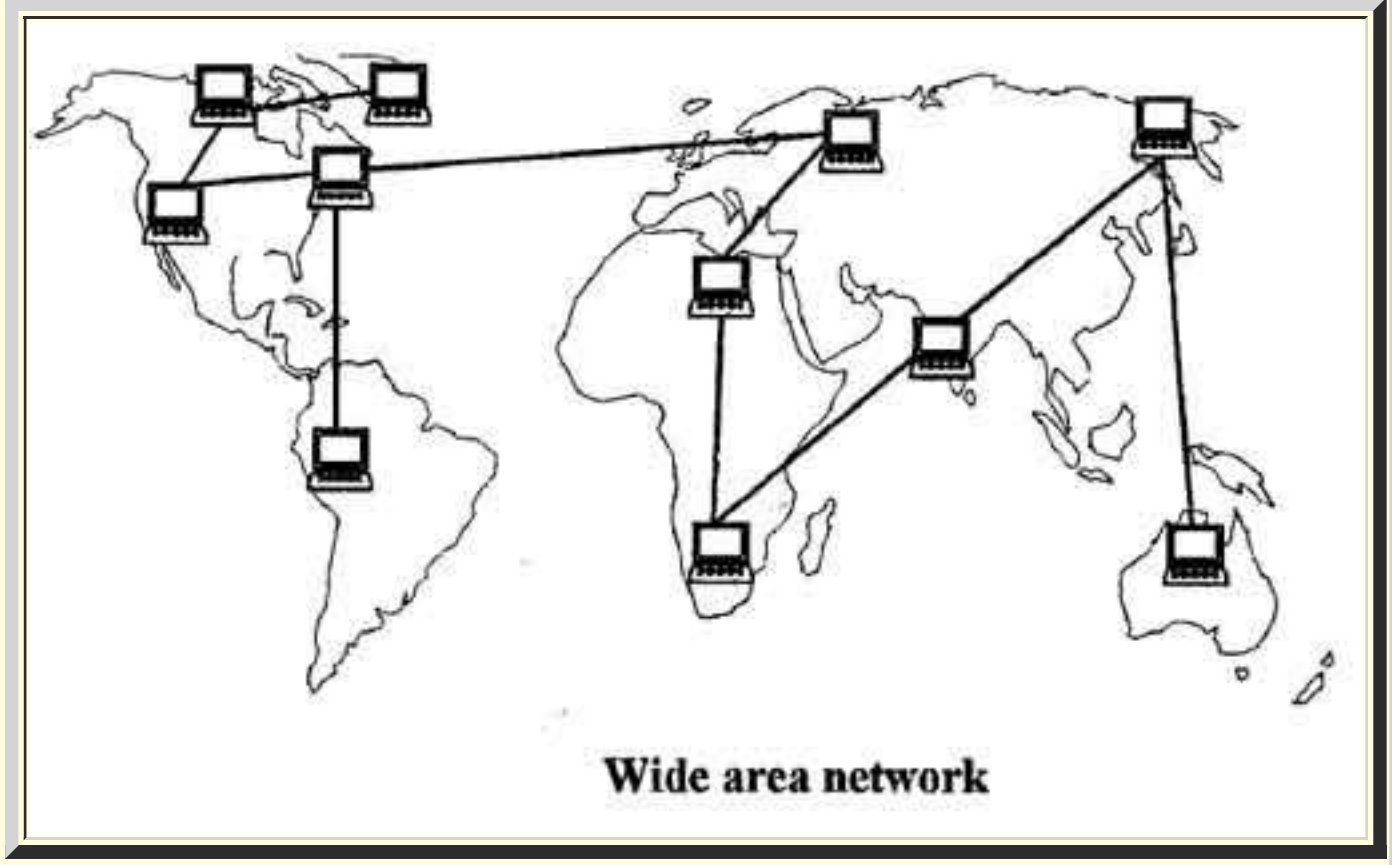
The Datalink Layer for *Broadcast* Links

- **Introduction:**
  - **Point-to-point link vs. broadcast links**

    - **Point-to-point links** ===> **switched networks**

      - These links **typically** spans ***large*** **geographical distances** (across continents)



      - Due to the **length**: **transmission errors (bit errors)** are **very likely**

        (Typical: 1 bit in ~10,000 bit for copper and 1 in ~10,000,000 in optical fiber)

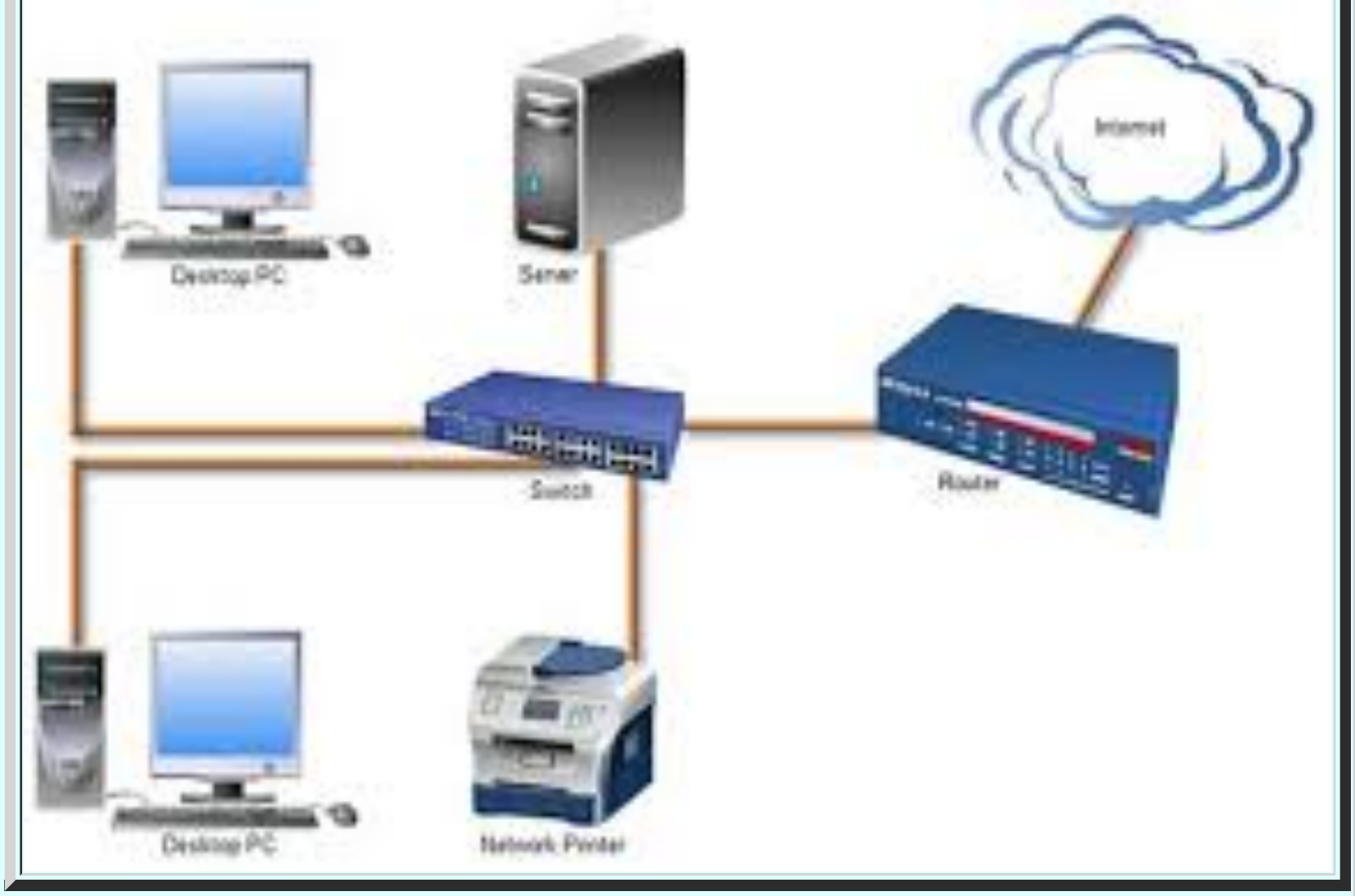
    Hence:

      - The **datalink layer** for **point-to-point link** deals **primarily** with:

        - ***reliable*** **transmission** using **Sliding Windows** protocol (in addition to framing)

        (E.g.: **HDLC** --- used in **switched networks** ---- uses ***Go-Back-N*** for **reliability**)

- **Broadcast links** ======> ***very short*** **distances (*local* area network)**



Fact:

- **Transmissions** over **short distance** are **very reliable**

Hence:

- The **data link layer** of **broadcast network** does **not** implement any ***reliable*** **communication protocol**

- **Problem** solve by the **Datalink layer** of **broadcast networks**:

  - **How** to ***share*** the **broadcast network** ***efficiently***

- **Transmission on a broadcast network**
  - **Property** of a **broadcast network**:

    - A **transmission** on a ***broadcast*** **network** will be **received** by ***all*** **nodes** on the ***broadcast*** **network**

      (**E.g.:** **radio** !!!)

  - **Consequence:**

    - When ***multiple*** **nodes transmit** at the ***same*** **time**:

      - ***All*** **transmitted frames** will be **lost** !!!

- **Collision:**

  - **Collision** = ***multiple*** **simultaneous transmission** on a **broadcast network**

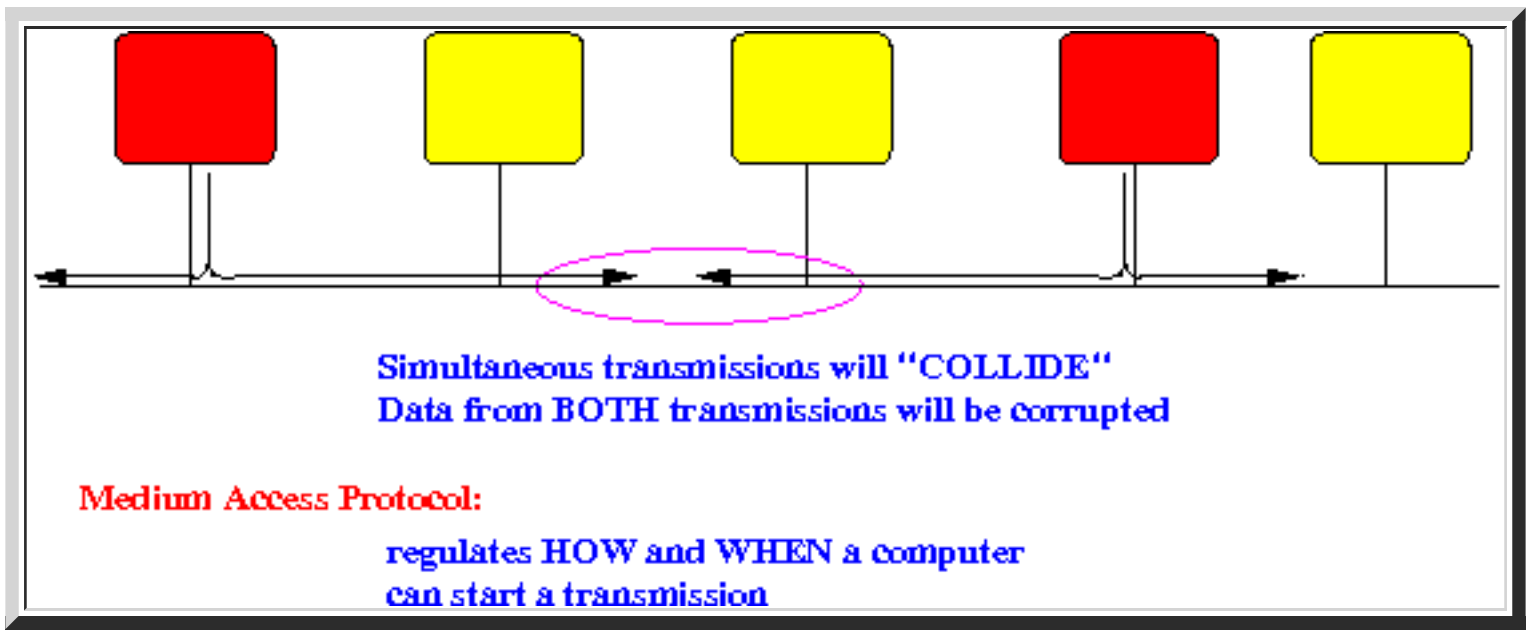
    (The ***simultaneous*** **transmission** have ***collided***)

- **Nomenclature: MAC**

  - **Medium Access Control (MAC):**

    - **Medium Access Control (Protocol)** = a **protocol (procedure)** that **nodes** on a **broadcast network** must follow ***when*** the **node** wants to **transmit** a **frame**

    Graphically:



- **Types of MAC protocols used in broadcast networks**

  - There are **2 types** of **Medium Access Control** **protocols**:

    - **Contention-based** **protocols**
    - **Contention-free** **protocols**

- **Contention-based MAC protocols**

  - **Property** of ***Contention based*** **MAC protocols**

    - **Nodes** in a **contention-based network** will **compete** to be the ***winner***

      - The **winner node** will be **able** to **transmit** its **message** on the **network** !!!

- **Operation** of **contention-based MAC networks:**

  - **Nodes** that **want to transmit** a **message** will **compete** to be the ***first*** **node** to ***start*** **transmitting** its **message**

  - The ***first*** **transmitting node** will **succeed** in **completing** its **transmission**

  - When ***other*** **(= not first) nodes** ***hears*** the **transmission** of the ***first*** **node**:

    - The **other nodes** will ***refrain*** from **transmitting**

      (Until the **transmission** of the **first node** is **complete**)

  - When the ***first*** **node** is **finish**:

    - **All active nodes** (= nodes that **want to transmit** a **message**) will **compete** ***again***

Another name: ***Multiple Access*** **protocols**

- **Competing** for **access** to the **network** can **result** in:

  - ***Simultaneous*** **transmissions**

- ***Simultaneous*** **transmissions** are **transmission "collisions"**

- **Result** of a ***collisions*** :

  - **All colliding transmissions** from **nodes** will be **lost** !!!!

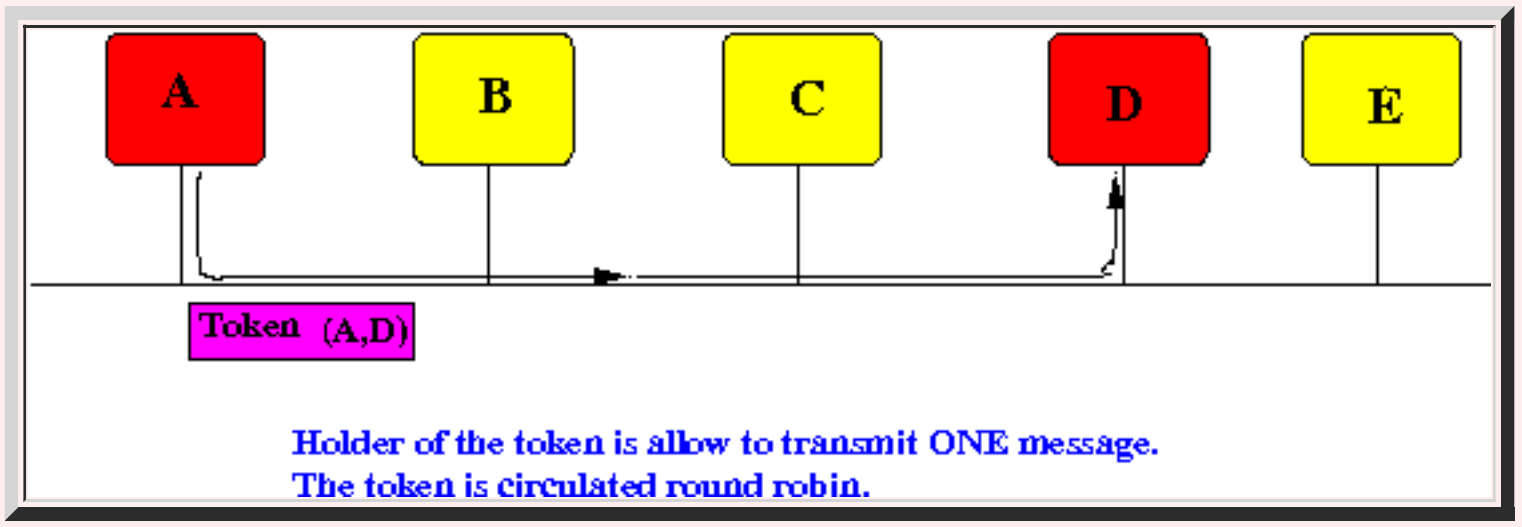
- **Nodes** involved in a **collision** will need to **transmit** the **frame** ***again***:

  - **I.e.:** nodes need ***multiple*** **attempts** before they will be **successful** in transmitting

- The **contention-based MAC protocols** are **also** called:

  - ***Multiple*** **Access Protocols**

- **Contention-free Medium Access Control protocols**
  - **Property** of **contention-free MAC**:
    - **Contention-free Medium Access Protocols *regulate*** the **network access** using a ***special*** **message** called **token**:



  - **Operation** of **content-free MAC networks**:
    - A **node** is **given** the **turn** to **transmit a message** when
      - The **node receives** a ***special*** **message** called: **"token" message**.

    - When a **node** received the **token**:
      - **If** the **node** has some **message** to **transmit**:
        - The **node** transmits ***one*** **message** and
        - The **node must "pass" (= send)** the **token** to the ***next*** **node**.

      - **If** the **node** does **not** have **any message** to **transmit**:
        - The **node must "pass" (= send)** the **token** to the ***next*** **node**.

  - Some **problems** that **contention-*free*** **protocols** must **resolve**:
    - **Maintain (= keep track of)** the ***complete*** **list** of **nodes**:
      - **How to add** a **new node** to the **network**
        - The ***new*** **node** does **not "get"** the **token** without being ***added*** !!

      - **How to remove** a **node** from the **network**

- If **some node** sends the **token (message)** to a **"deleted" node**, the **token (message)** will be ***lost*** !!!

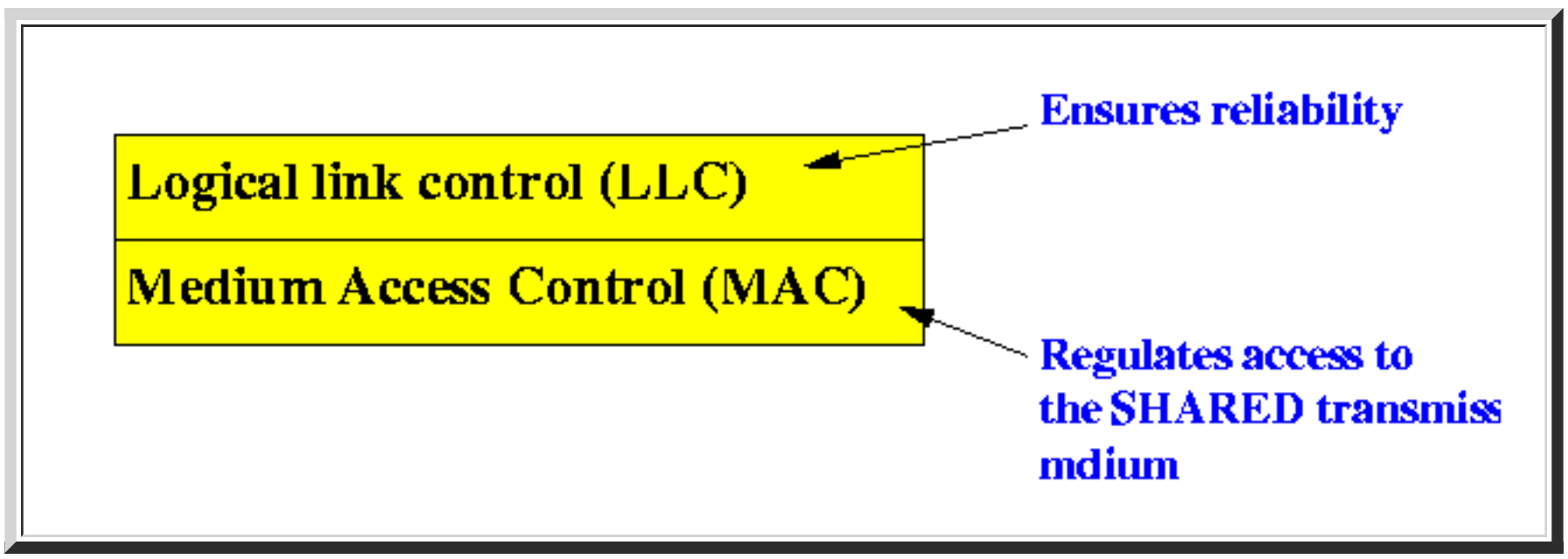
  (Then the ***whole*** **network** will **stop** working --- like a **broken traffic light** that shows **red** on ***all intersections***)

- **Structure of the Datalink layer for broadcast links**
  - The **datalink layer for *broadcast* links** consists of ***two*** **functional layers**:

    - **Logical Link Control** that **provides** ***reliable communication***

    - **Medium Access Control (MAC)** that **provides functions** to ***access*** **(= share)** the **broadcast network**

  Graphically:



- **Logical Link Control**
  - **Logical Link control:**

    - Provide **reliability functionality**

      (i.e.: **sliding window** and **ACK scheme**).

  - **Real-life Datalink layer:**

    - **Real-life datalink layers** (e.g.: **Ethernet**, **Token ring**) for **broadcast networks** do **not implement** the **Logical Link control layer (function)** !!!

- ***How*** to **recover** from **transmission errors**:

  - The **error control function** is ***also*** implemented in a ***higher*** **layer**:

    - In the **Transport Layer** (see later !!!)

  - The **Ethernet/Token Ring protocol datalink layer relies (= depends)** on the **Transport Layer (TCP !!!)** to **recover** any **lost messages** !!!

    (The **TCP layer does implement** the **reliability function**)

- **Medium Access Control**

  - **Medium Access Control:**

    - The **MAC layer** implements **functions** to **regulate *orderly* and *fair* access** to the ***shared* (broadcast) transmission medium**.

- **Next...**

  - We will study a number of **commonly used MAC**s...

Aloha: the grand-father of Multiple Access or "Contention-ful" MAC protocols

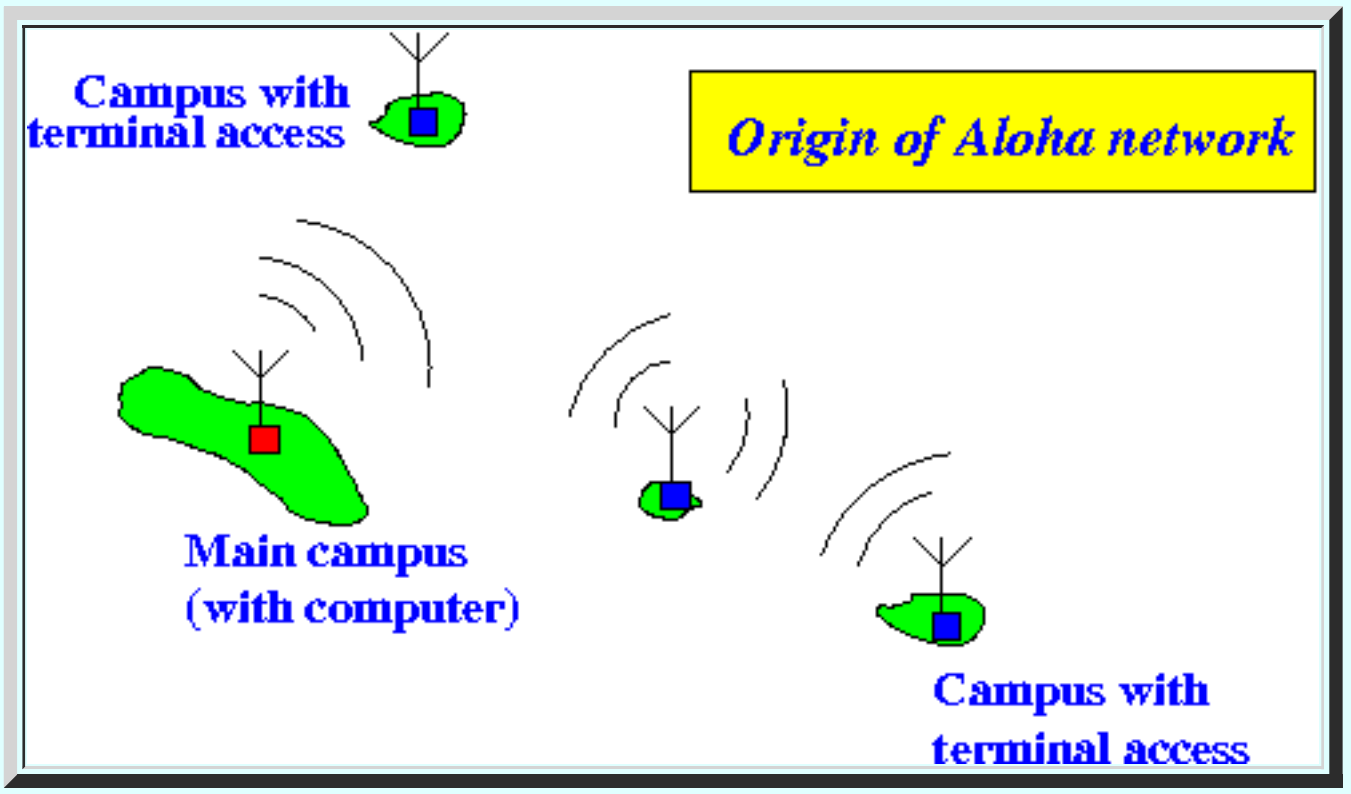
- ***Intro* to Aloha**
  - **What is Aloha:**

    - The **very first midium access protocol** developed at the **University of Hawaii** around **1970**.

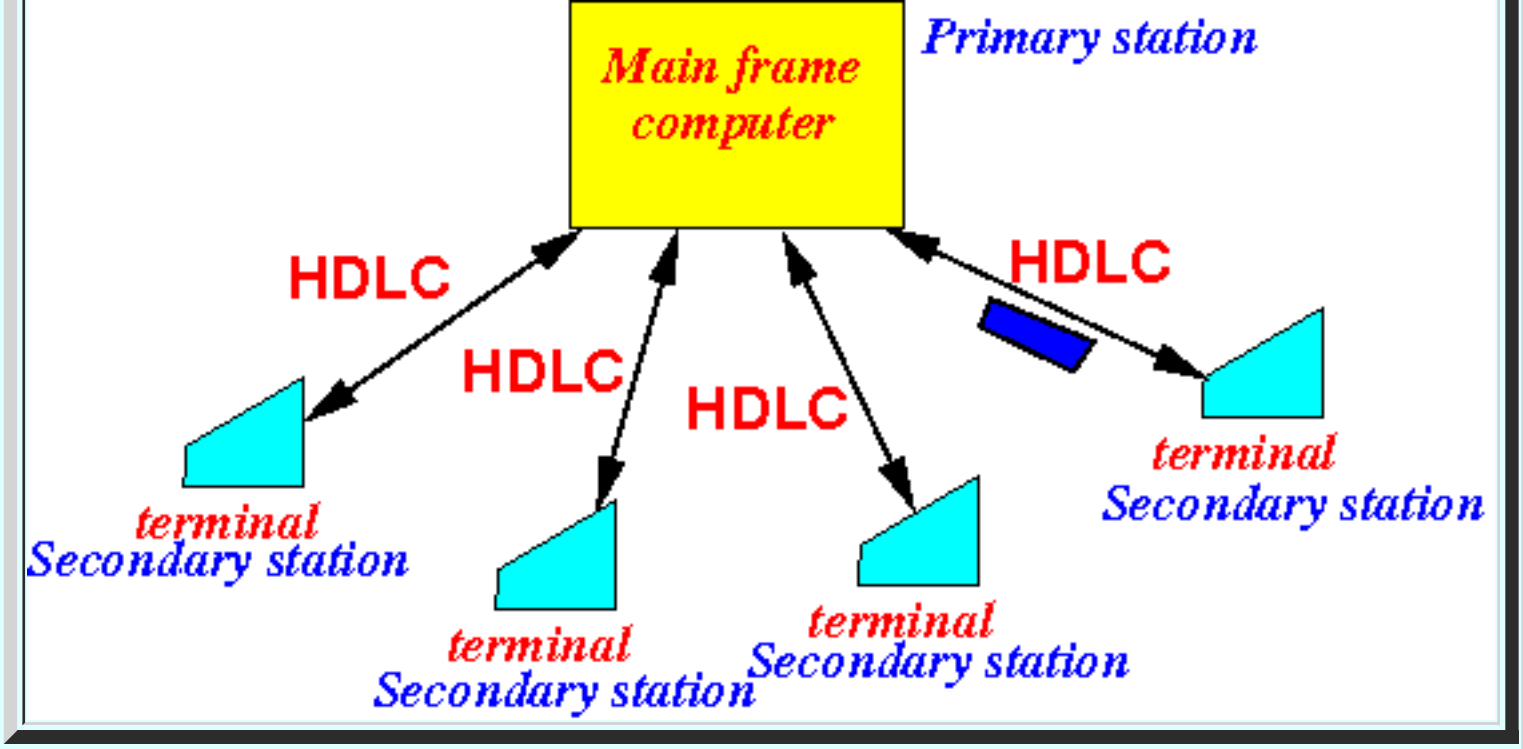
  - **Their problem:**

    - The **Uuniversity of Hawaii** had **campuses** on ***different*** **islands**

    - **Computers (a.k.a. "main frames")** were ***only*** **located** on the **main campus** (computers were very exensive in those days):



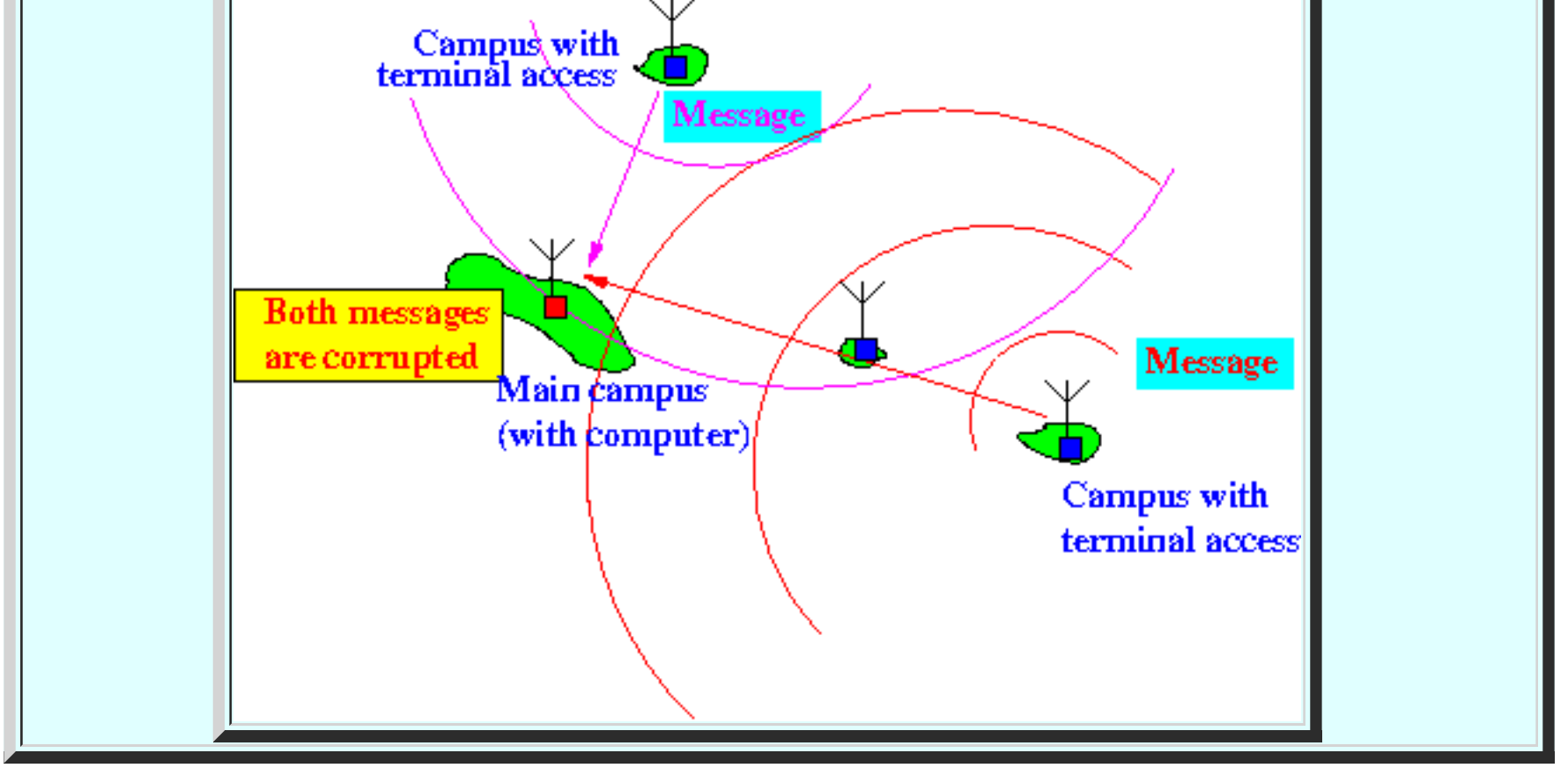
    - **In those days**, we use **terminals** connected (through **wires**) to the **main frame** to **access** the **main frame computer**:



- The **problem** facing the **Univ. of Hawaii**:
  - It's **too expensive** to run **wires** between the **main campus** and the different **satelite campussess**
    - **How** do you **connect** the **terminals** on **campusses** on ***other*** **islands** to **gain access** the **computer (main frame)** on the **main campus** ????

  - Their **solution**:
    - **Use radio communication !!!**

- **Intro to radio communication**
  - **Property** of **radio transmissions** (on the ***same*** **frequency**):
    - **Simultaneous *reception*** of ***multiple*** **transmissions** will cause ***both*** **transmissions** to be received **in error**:



- **The Aloha network protocol**
  - **Network *protocol***:

    - **Network protocol** = a **procedure** that **must** be followed by ***every*** **nodes** on a **network** to ***regulate*** the **access** to the **communication network**

  - **Operation** of an **Aloha *terminal***:

    - When the **user** on the **terminal** is ***idle***:

      - The **terminal** is **idle**

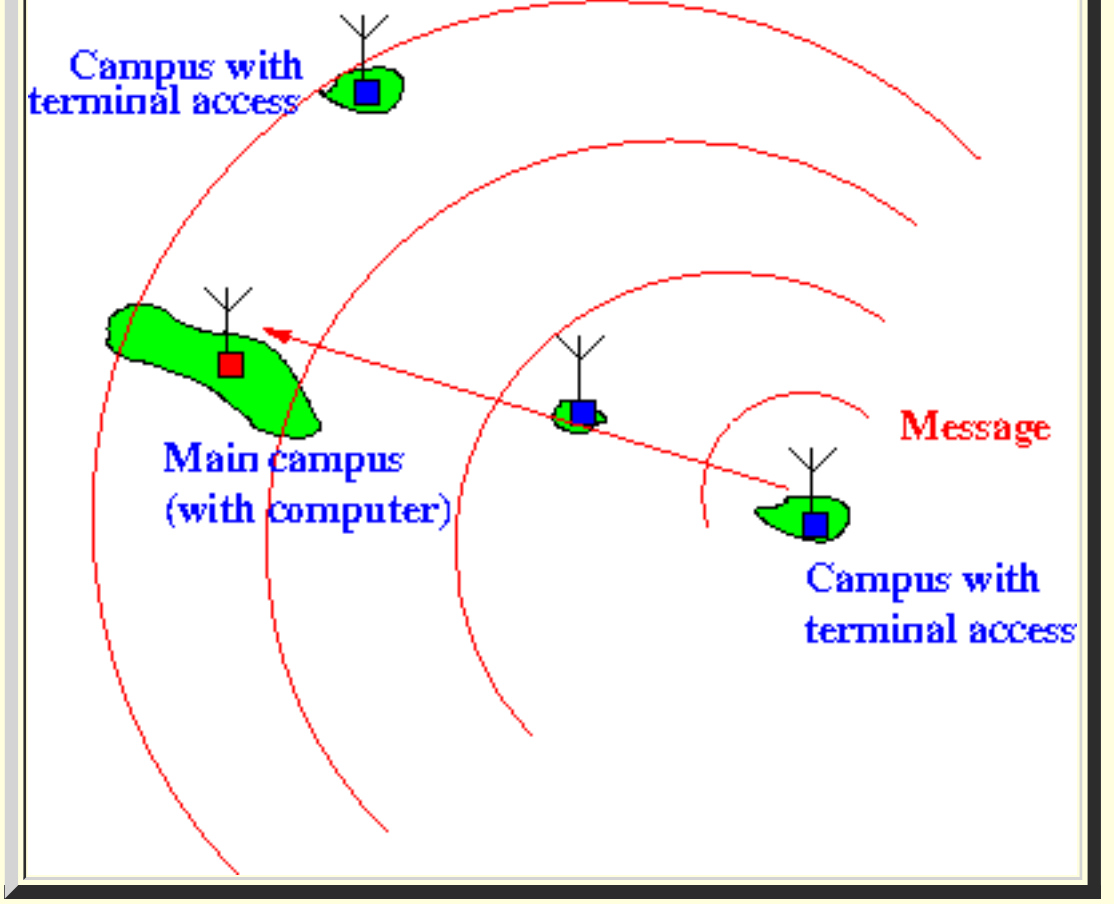
        I.e.: does **nothing**

    - When the **user** on the **terminal hits/presses** a **key**:

      - The **terminal** will **transmit** a **message** containing the **ASCII code** for the **key (letter) pressed** ***using*** the **Aloha network protocol**

  - The **Aloha network protocol**:

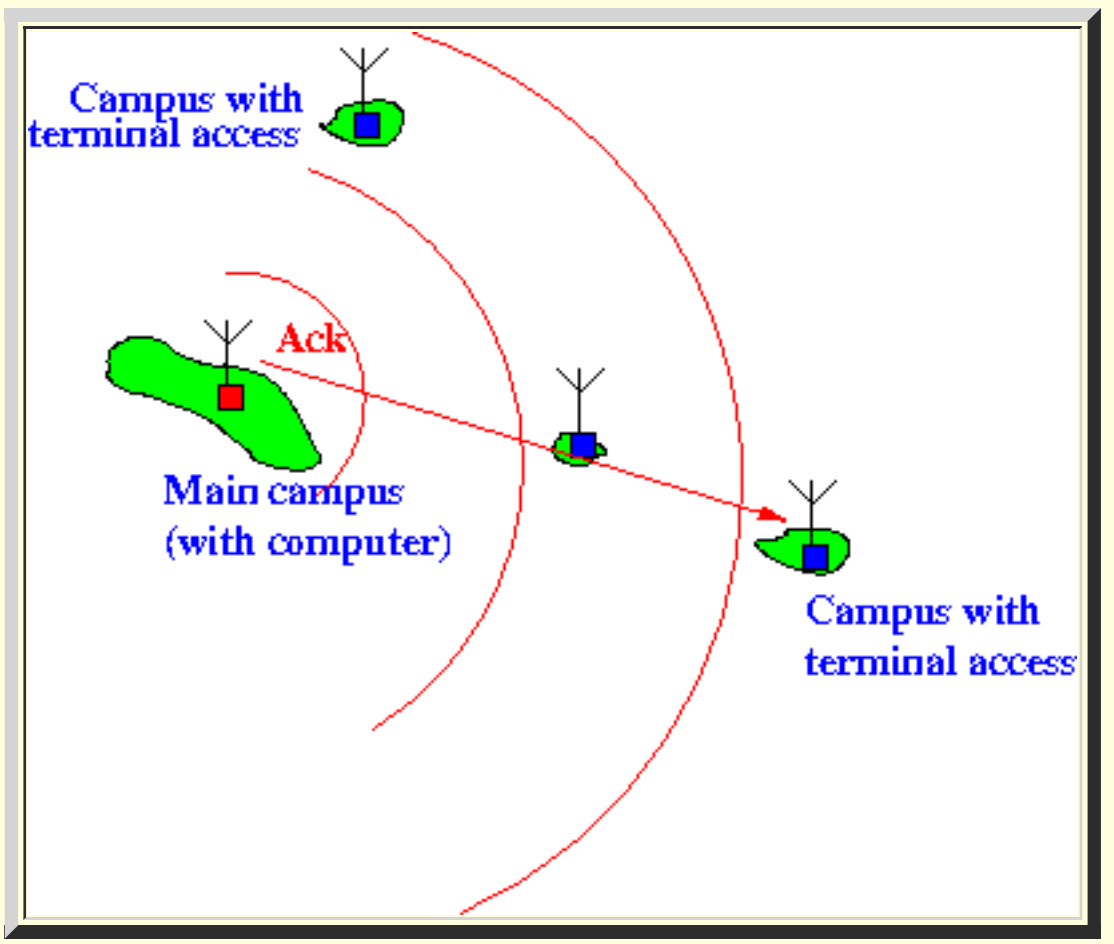
    - The **(radio) terminal** on a **satelite campus** sends the **message (= frame)** to the **main frame** on the **main campus** ***and*** then ***wait*** for an **ACK frame**:



Note:

- The **(radio) transmission** can be **"heard" (received)** by ***all*** **nodes (terminals/main frame)** in the **Aloha network** !!!

- If the **message** was **received *correctly***, the **main frame** transmits an **ACK message (frame)** back to **acknowledge** the **reception**

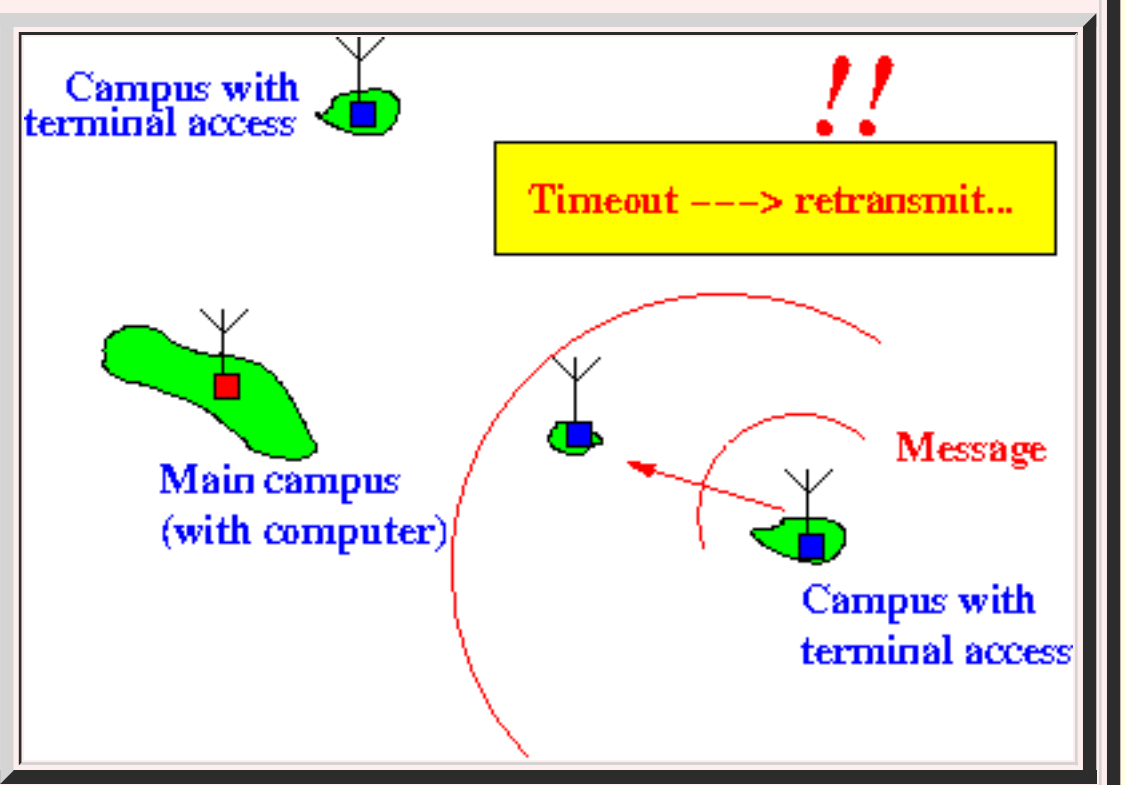


- If the **terminal** did **not** receive an **ACK message** (from the **main frame**) after a **time out period**, the **termimal** will:

1. **Wait** a ***random*** **period**

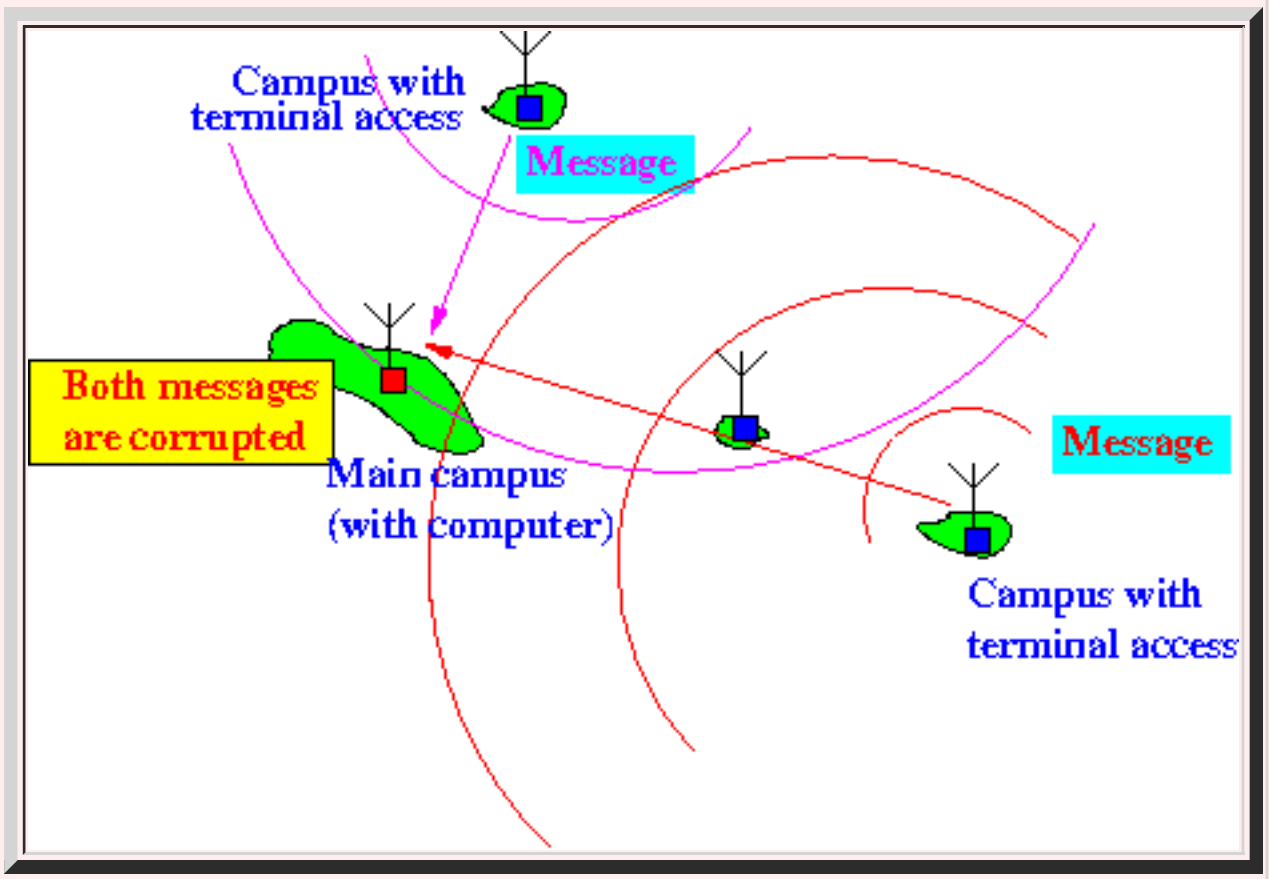
---

2. **Then** ***re-tramsmit*** the **message**



Note:

- The ***most likely*** **reason** that a **message** was **lost** is:
  - There was a **collision** due to ***simultaneous*** **message arrival/reception** at the **main campus site**:



- ***Waiting*** a ***random*** **time period** **before** **re-transmitting** the **messages** will:

- ***Decrease*** the **likelihood** of ***another*** **collision** due to ***simultaneous*** **transmissions**

  (it's **not likely** that will **wait** the ***same*** **random period** !!!)

- **Note:**

  - If the **main frame** wants to **transmit** a **message** to some **terminal**, the **main frame** will use the ***same*** **network protocol**:

    - **Main frame** sends the **data frame** and **then** **waits** for an **ACK frame** from the **terminal**

    - If **no ACK message** is **received** before the **times out**, the **main frame** will:

      - **Wait** a ***random*** **time period** and then

      - **Re-transmit** the **message**

- **Operational environment on Hawaii**

  - **Some background information** on the **operational environment** in **Hawaii**:

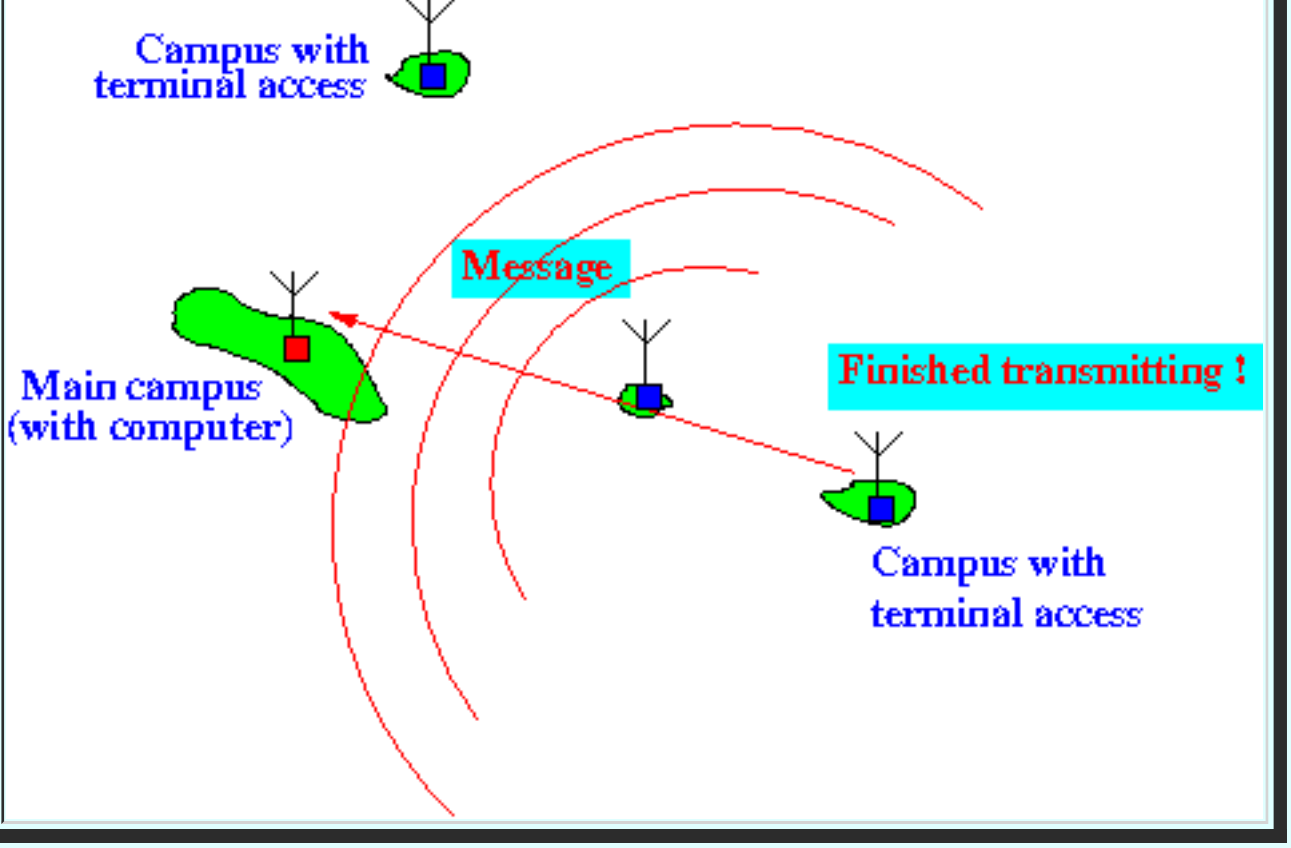
    - The ***distance*** between **island** is **very large (7-63 miles apart)**

    - **Messages** between **terminals** and **main computer** are ***very*** **short**

      (We only had **character-based** **terminals** --- ***no*** **graphics** in **1970** !!!)

Result:

- The **terminal** (on a **remote island**) will have ***finished*** transmitting the **message** ***before*** the **radio signal** has **reached** the **main campus** !!!


Campus with
terminal access
Message
Main campus
(with computer)
Finished transmitting !
Campus with
terminal access

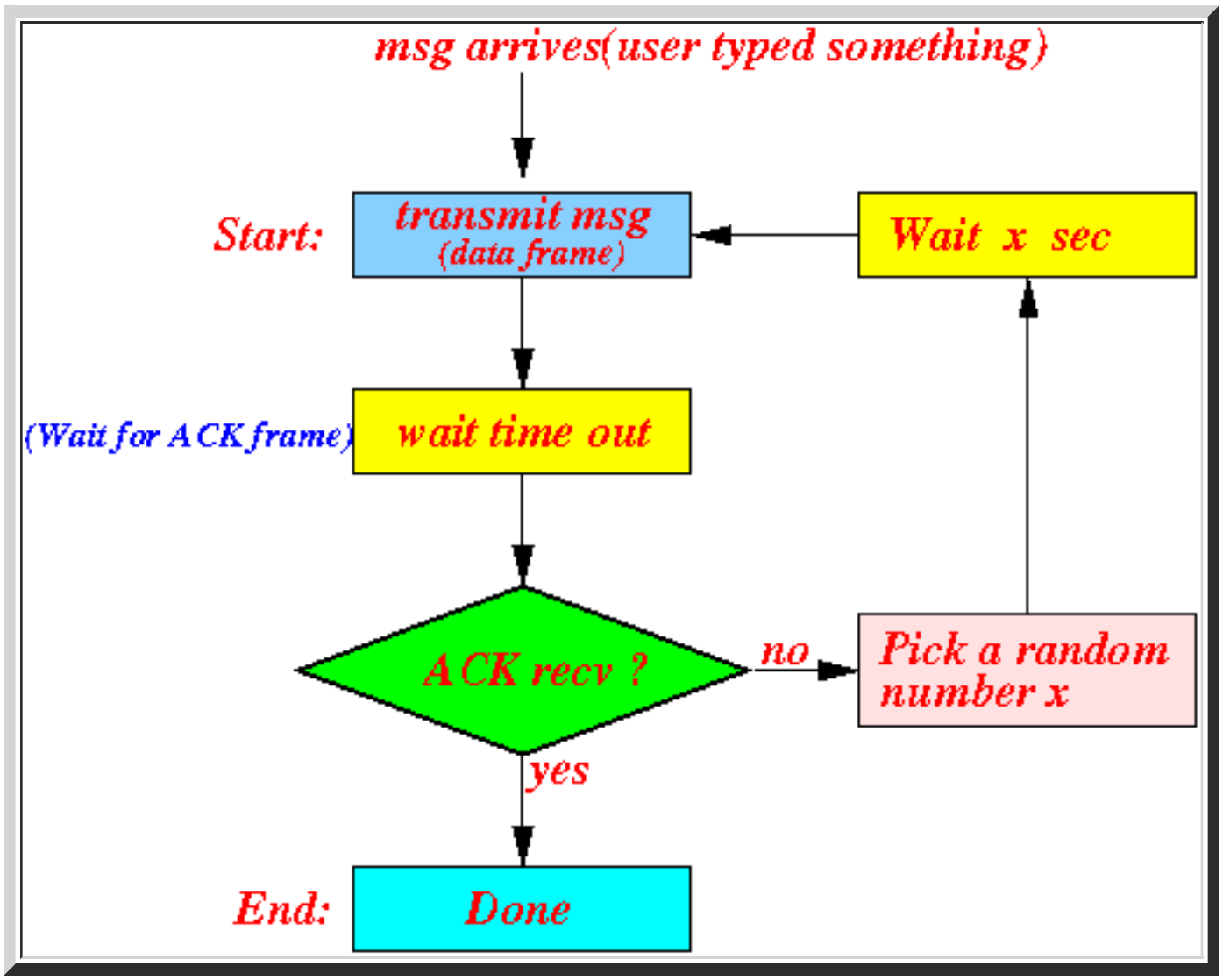
The Aloha Protocol

- **The Aloha Protocol**
  - **Fact:**

    - **Previously**, we have **described** the **Aloha network protocol** ***informally***

    - We will **now give** the **formal description** of the **Aloha network protocol**

  - The **Aloha protocol**:



Explanation:

- If a **node** has a **message** to transmit:

  - **transmit** the **message** (***immediately***)

- ***Then:***

- **Wait** a **time out** period for the **ACK frame**

- If **ACK frame** is received within **time out**:

  - **done**

  Otherwise: (the **data frame** was probably **lost** due to **collision** !!)

  - **Wait** a ***random*** **interval** (to **avoid** ***another*** **collision**)
  - **Retransmit** the **message**

- **Note:**

  - The **most likely cause** that there was ***no*** **ACK reply** is:

    - The **transmission** was **corrupted** due to **collision (simultaneous transmission)**

  - The **random delay** in included into the **Aloha protocol** to:

    - **reduce** the ***likelihood*** of a ***repeated*** **collision**

      (***Different*** **nodes** will **pick** a ***different*** **delay** before they transmit again !!!)

    When **different nodes** pick ***different*** **random delays**, ***simultaneous*** **reception** is **less likely**

- **Use sensing to help avoid potential collision**

  - **Sensing:**

    - **Sensing** = **listening** for **transmissions** on the **broadcast channel**

  - **Important fact:**

    - When ***multiple*** **nodes transmits** at the ***same*** **time**, the ***simultaneous***

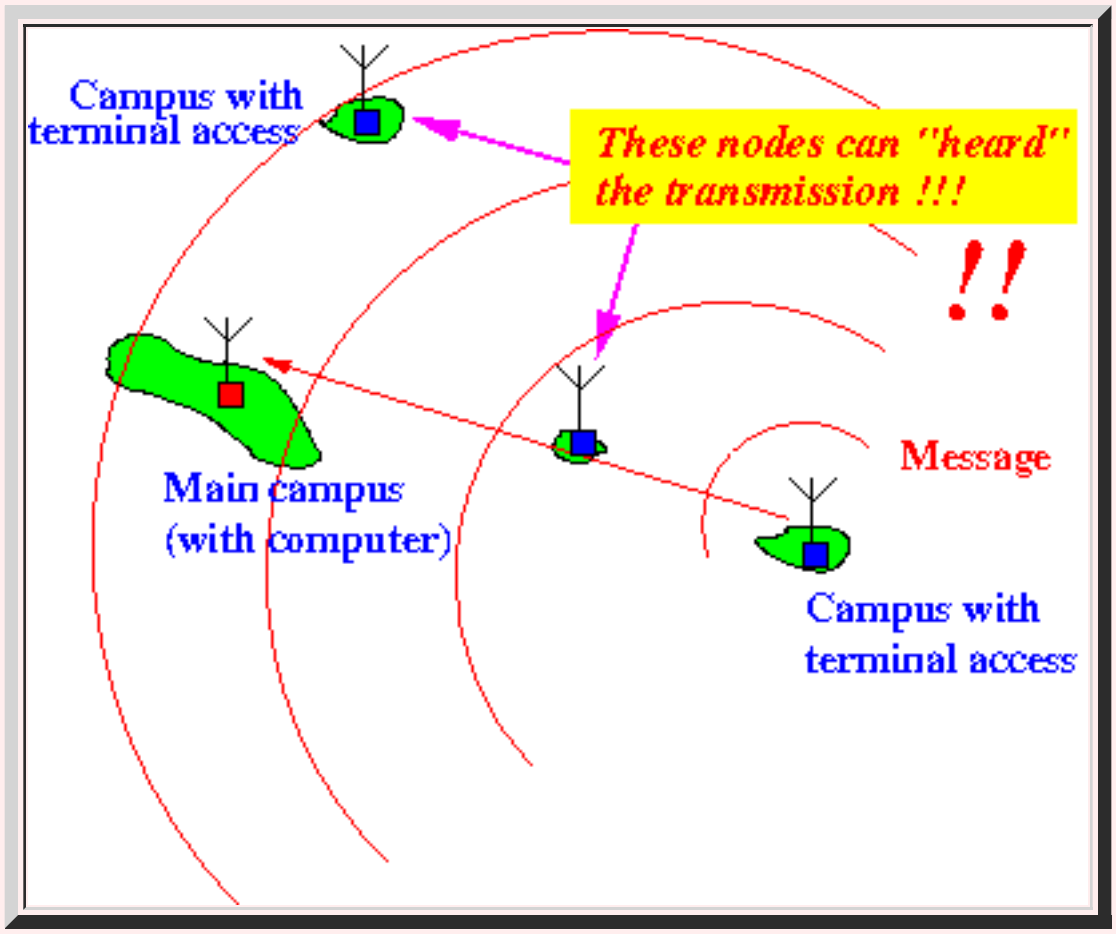
transmissions will cause a ***collision***

- ***All* frames** in a **collision** will be **lost** !!!!

- ***Broadcast* network often** use ***sensing*** to ***minimize*** the **likelihood** that **mmultiple nodes** will **transmit** at the ***same* time**

- **How sensing** can **help** avoid **collisions**:

  - ***Other* terminals (nodes)** can **"heard"** the **transmission** of a **terminal**:

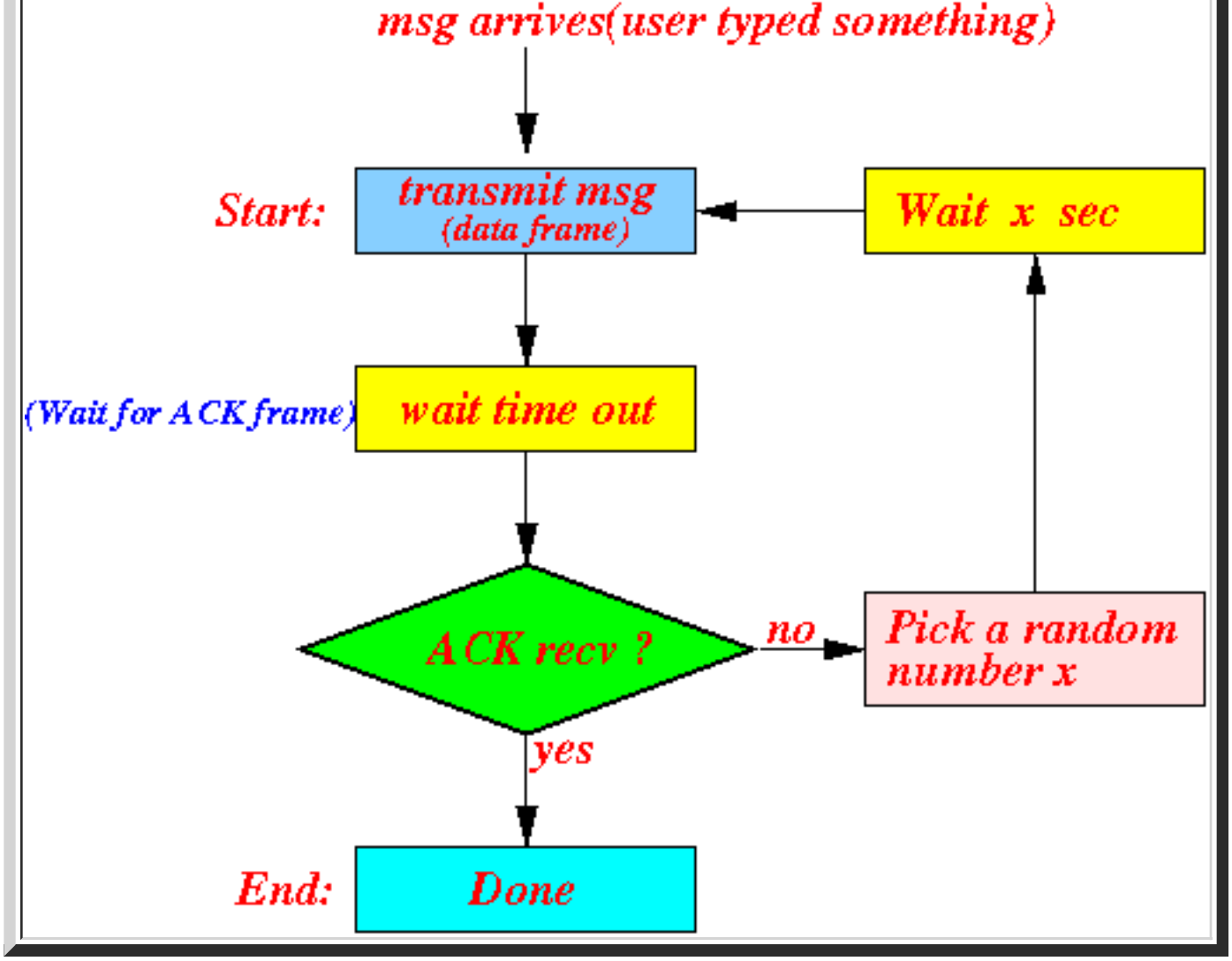


  - When a **node** detects a **transmission**, it can **avoid a collision** by:

    - ***delaying*** its **own** transmission until the **channel** is ***clear***

- **Aloha does not use sensing**

  - The **Aloha protocol**:



does **not** use **channel sensing** !!!

- **$64,000 question:**

  - Why does a **node** **not** use ***sensing*** to **avoid** a **potential collission** ????

  Answer:

  - the **messages** were **short** (nodes send **text** in those days, not **graphics**) and
  - the **distance** between the islands were **large**.

    Result:

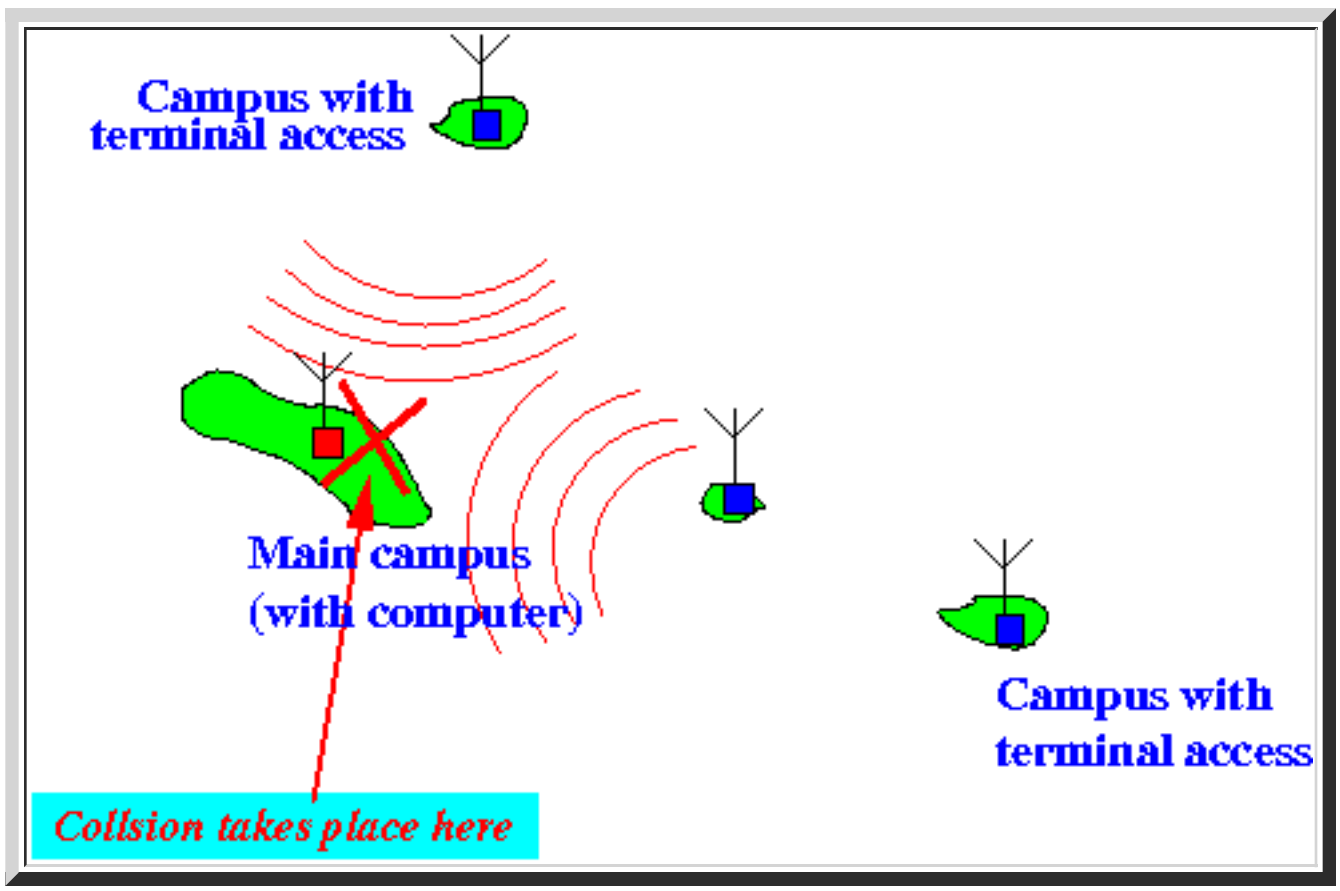
    - **Sensing** will **not** help a **node** to **avoid** a **collision**

- **Important fact:**

  - A **collision** of ***multiple*** **transmissions** take place at the **receiver (= main campus)**

(**Not** at the ***sender*** !!!)

Graphically:



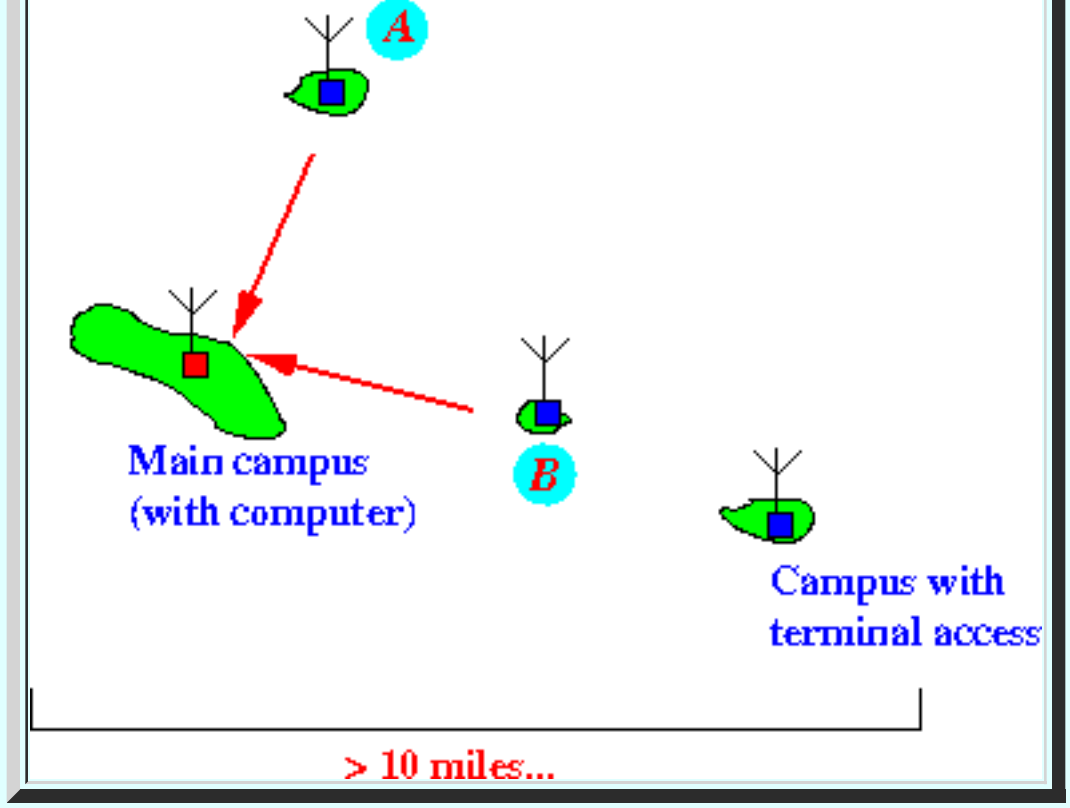
- ***Why*** does **sensing *not* help** to **avoid collision** in **Hawaii** ???

Because:

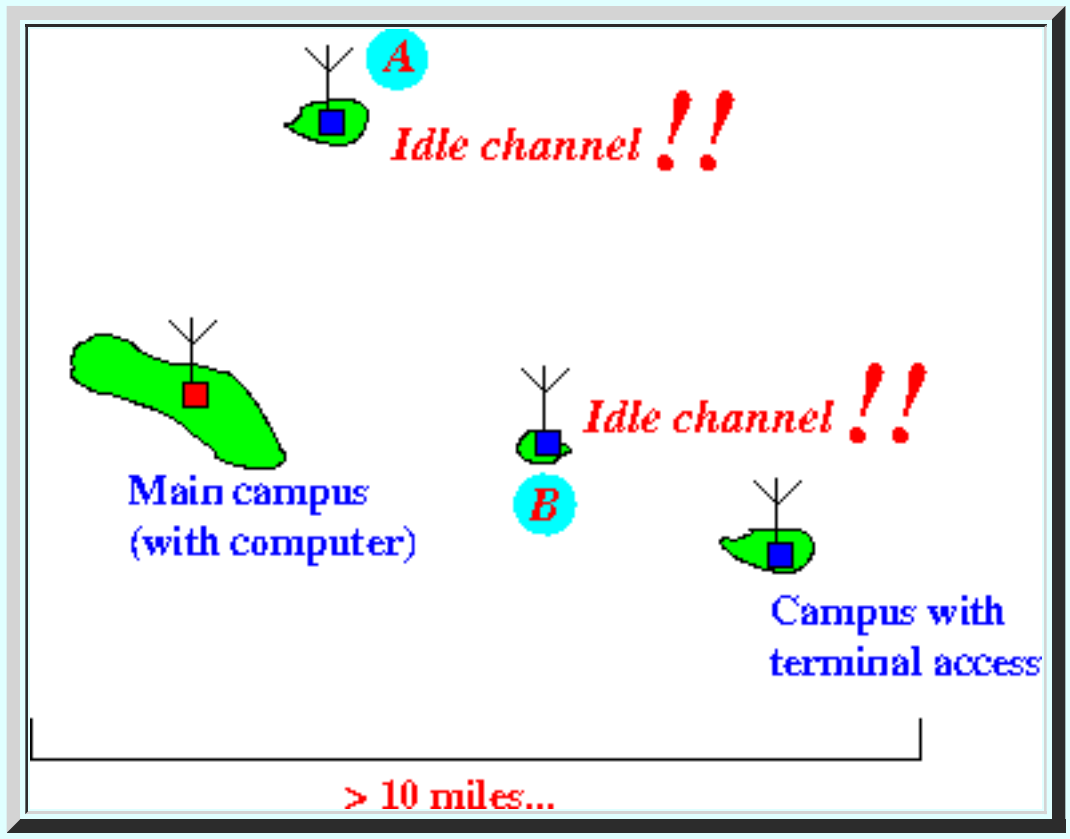
- ***If*** a **node** senses an ***idle* transmission medium** and **transmits**:
  - The **transmission may** result in a ***collision***

Example:

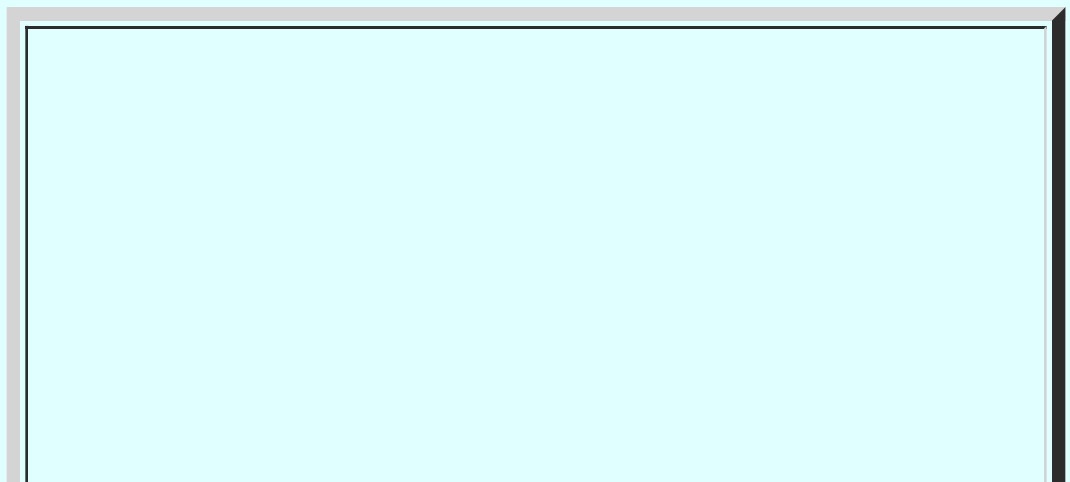
- **Suppose node *A*** and ***B*** wants to **transmit** a **data frame** to the **main campus**:

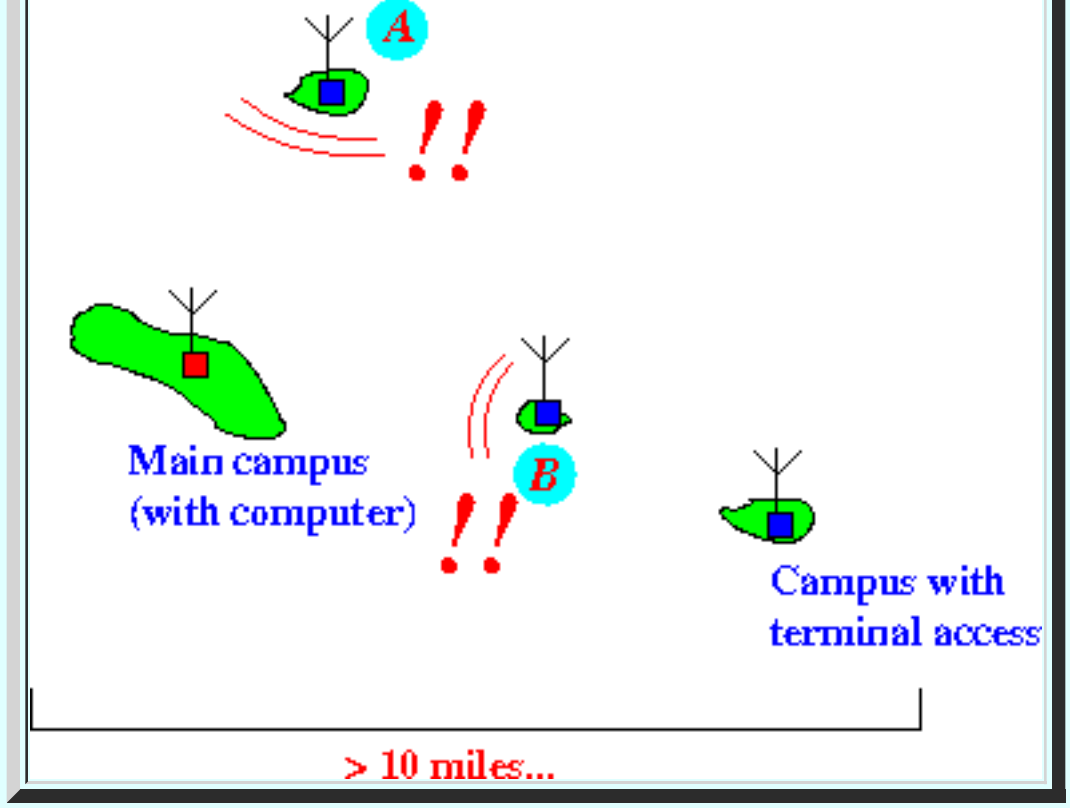


- ***A*** and ***B*** will **sense** that the **channel** is ***idle***:

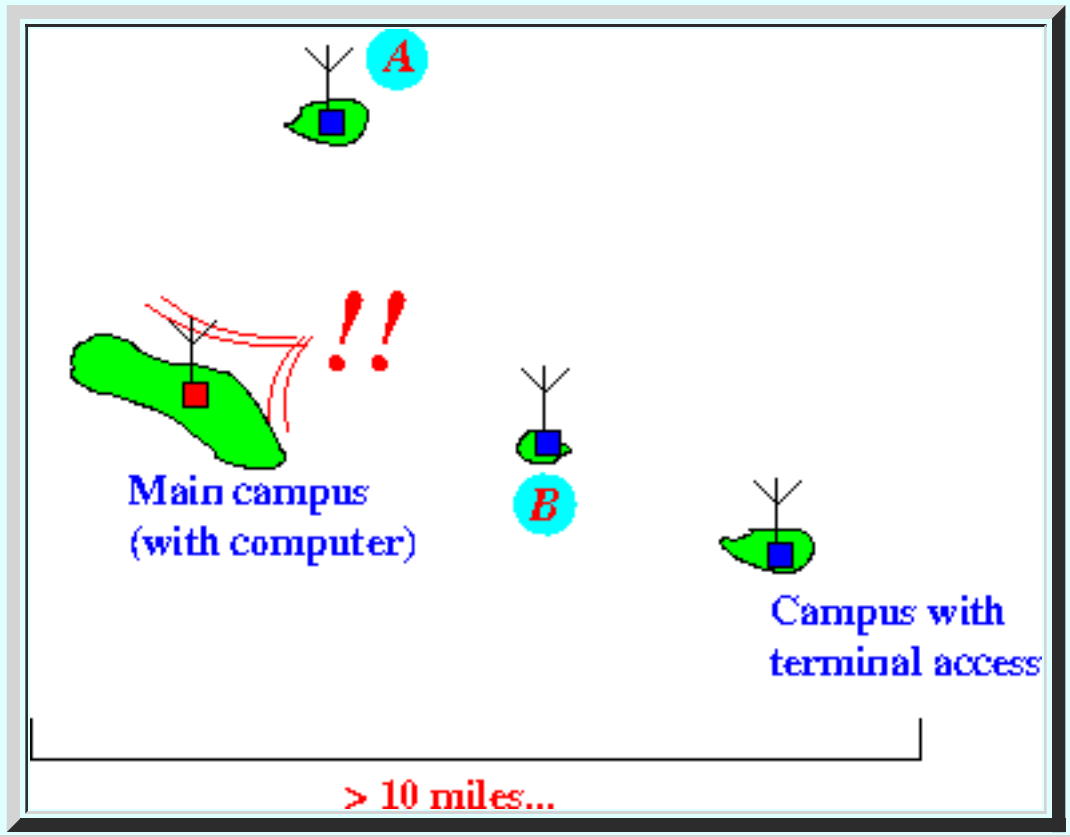


Both nodes will **start** their **transmission**:





And their **transmission** will ***collide*** at the ***receiver***:

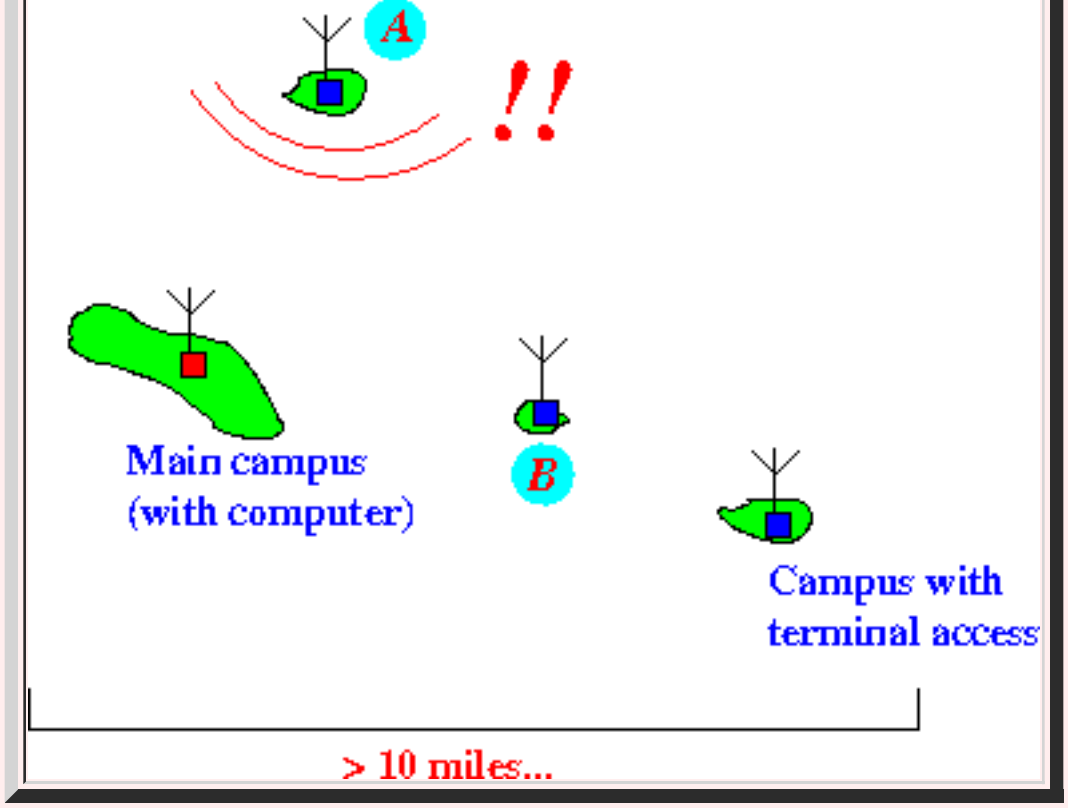


- **Furthermore:**

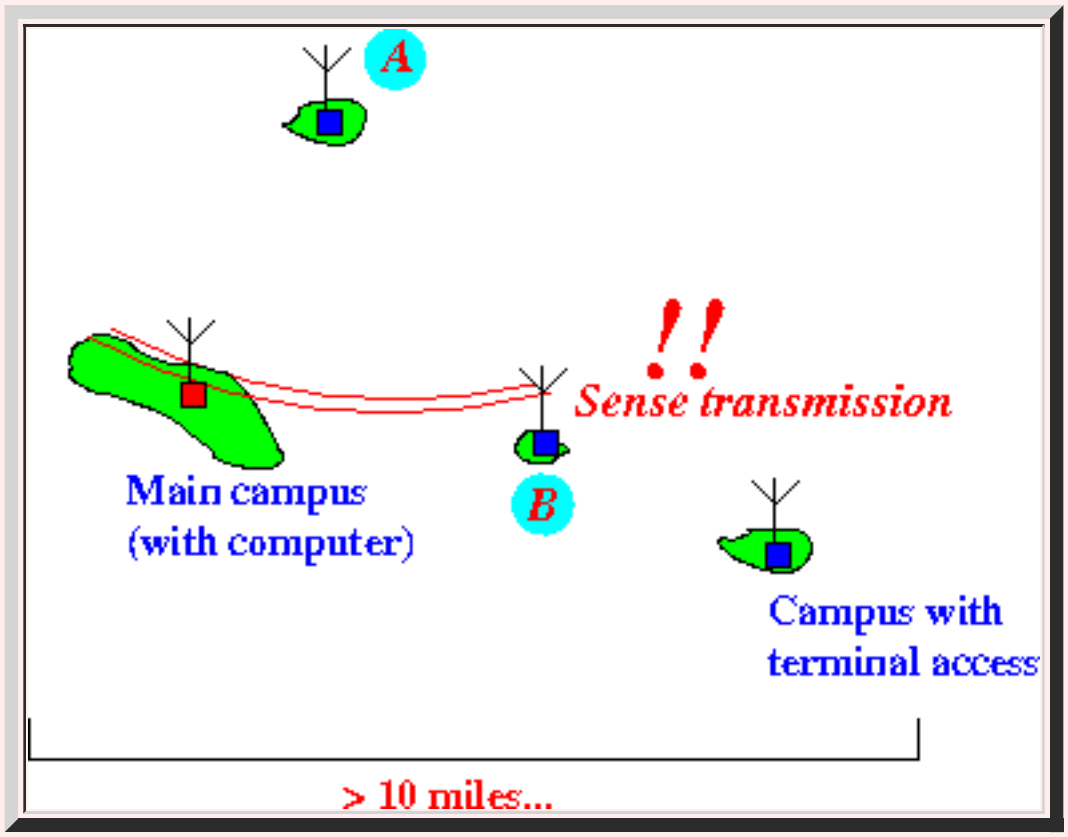
  - **Sensing** a **busy channel** is ***not*** an **indication** for a **collision** !!!

  Example:

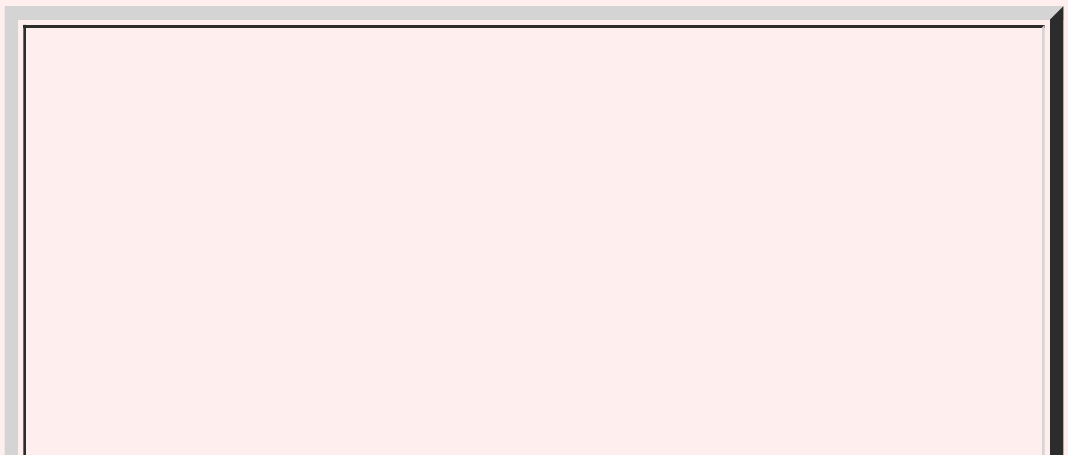
  - **Supose *A*** sends a **message** to the **main campus**:

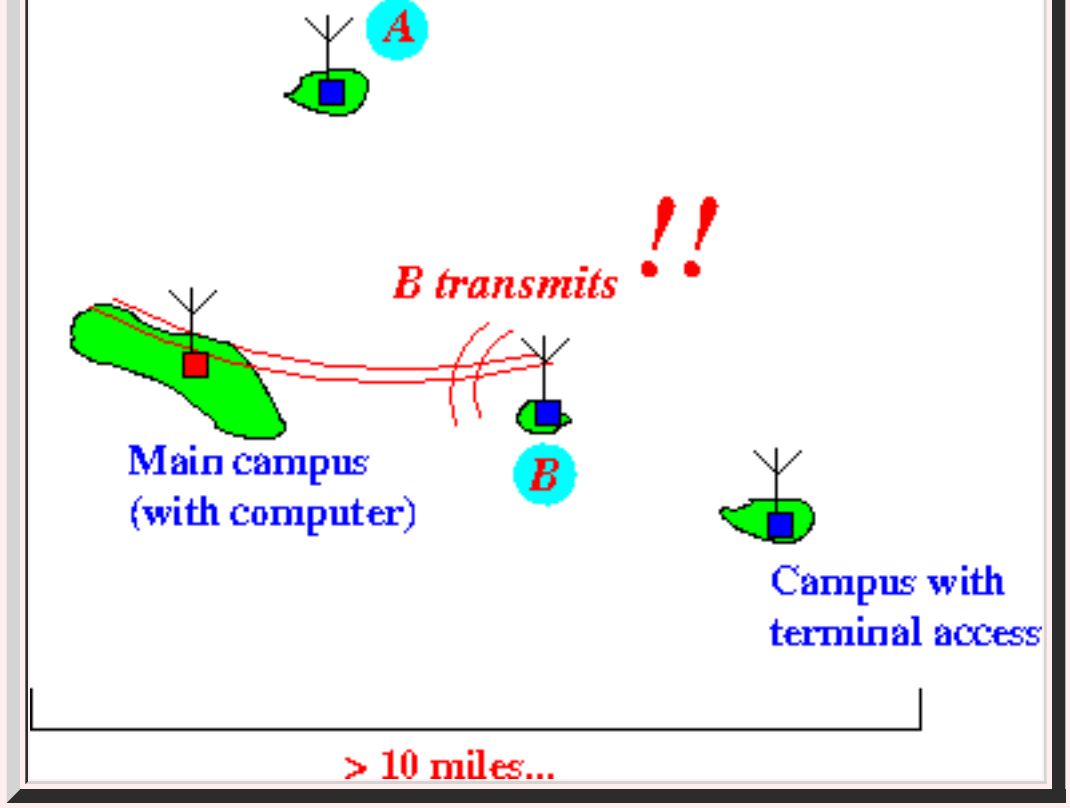


- ***A***'s messages **reaches** the node ***B***:

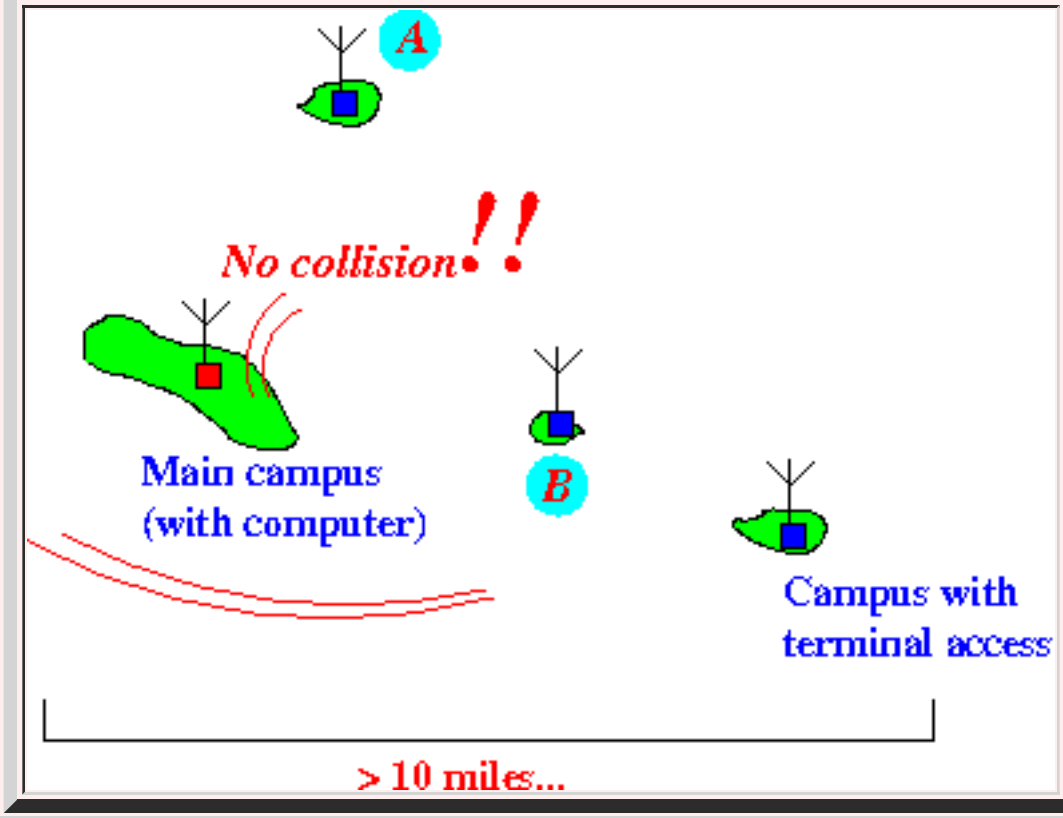


- ***If*** node ***B*** start **tranmitting** **NOW**:





The **transmission** will ***not*** **collide** at the ***reciver***:



Therefore:

- **Sensing** that the **channel** is **idle** can **not** help you do the ***right*** **thing**

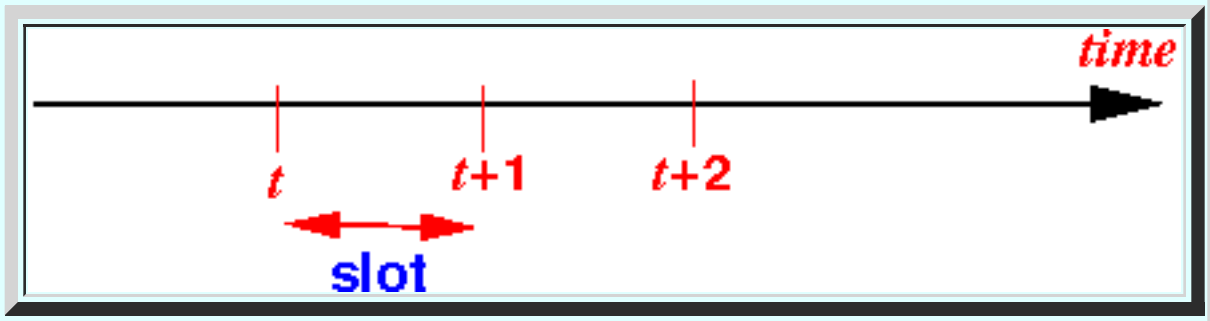
- **Sensing** that the **channel** is **busy** can **not** help you do the ***right*** **thing**

So: ***why*** **bother sensing** the **channel** at all ???

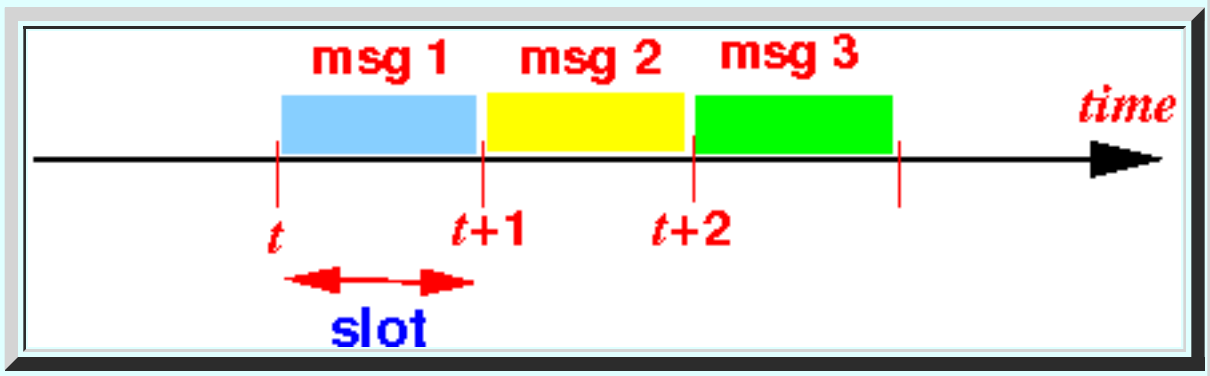
Slotted and *unslotted* transmissions

- ***Slotted* and *unslotted* transmissions**
  - **Slots** and ***slotted* transmissions:**

    - **Slot** (time slot) = the **transmission time** is **divided** into ***fixed* size "slots"**

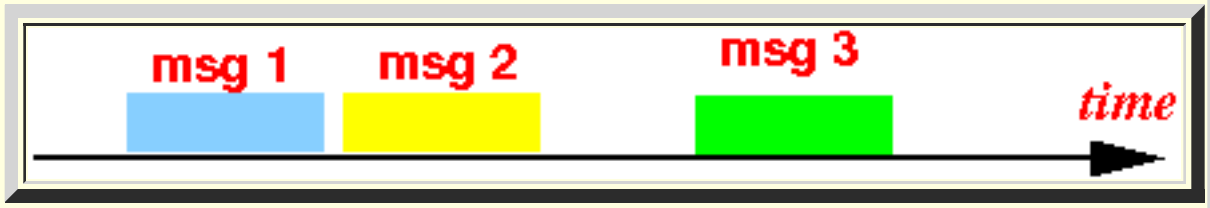


    - A **transmission** can ***only*** start at the **begin** of a **slot**:



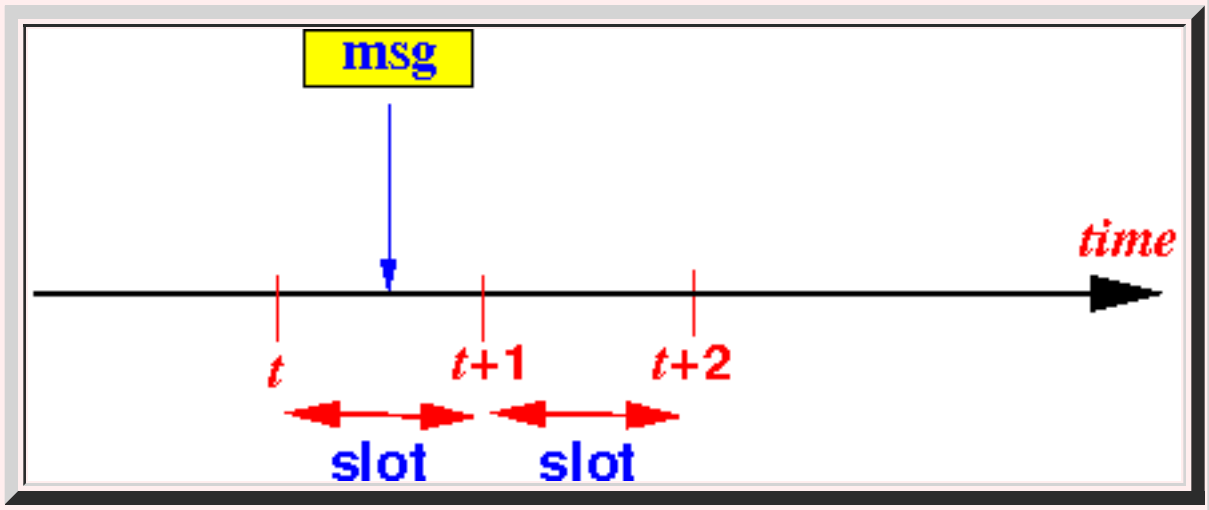
  - ***Unslotted* transmission**:

    - In ***unslotted* transmission**, a **transmission** can **start** at ***any* moment**:

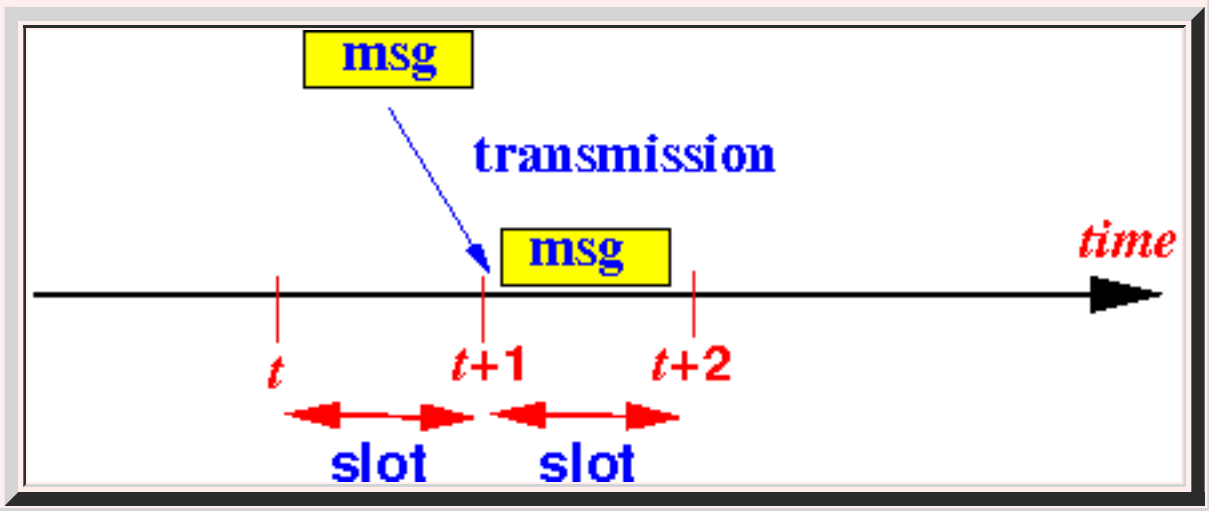


- **Slotted and unslotted transmission protocols**
  - **Slotted *transmissions***:

- A **message** that **arrives** during some **slot *t***:



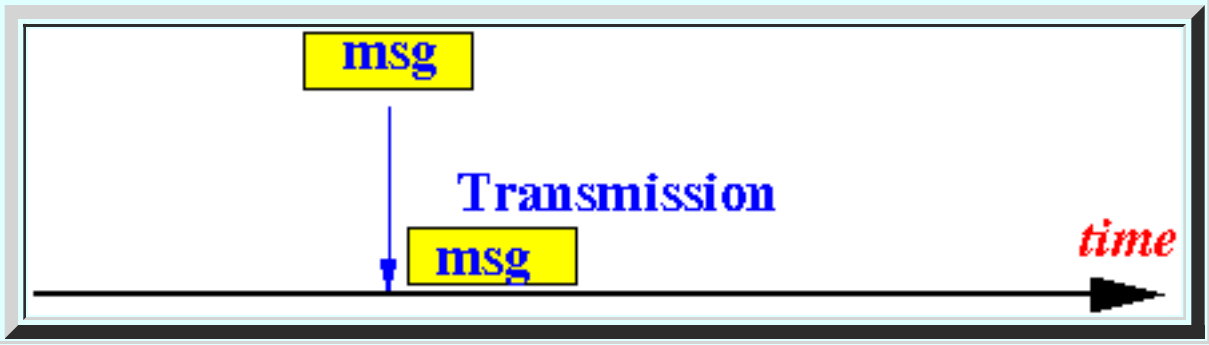
will be **transmitted** in the **slot *t*+1**:



- **Unslotted *transmissions***:

  - A **message** that **arrives**:

will be **transmitted** in the ***same* time** as its **arrival moment**:



- **Slotted and Unslotted Aloha**
  - **Slotted Aloha:**

    - **Slotted Aloha** = the **Aloha network protocol** that **uses *slotted* transmissions**

  - **Unslotted Aloha:**

    - **Unslotted Aloha** = the **Aloha network protocol** that **uses *unslotted* transmissions**

Introduction to Performance Analysis

- **Performance Analysis**
  - **Performance Analysis:**

    - **Performance Analysis** = the **branch** of **Mathematics** that study the **question**:
      - How ***efficient*** is a **system**

  - The **questions** studied include:

    - How ***many*** **customers** can the **system** **service** **per time unit (sec)**
    - How **long** does the **system** take to **service** a **customer**.
    - How **long** does a **customer** has to **wait** on **average** ???
    - And so on.

- **Terminology: *arrival rate* and *throughput***
  - **Definitions:**

    - **Arrival rate** (**Offered load**) = **number of customers** that **arrives** to the **system** **per time unit (sec)**
    - **Throughput** = **number of customers** that are **served** by the **system** **per time unit (sec)**

  - **Computer networks:**

    - The "**customers**" in a **computer network** are:
      - ***messages*** !!!
    - In **computer networking**:
      - **Arrival rate (offered load)** = **number of *messages*** that **arrives** **per time unit**
      - **Throughput** = **number of *messages*** that are **transmitted** **per time unit**

- **Performance analysis for *contention* base networks**
  - **Fact:**
    - In **Aloha**, a **message** may require ***multiple* transmission attempts** before the **message** is **transmitted *successfully***

      (This is due to **collisions**)

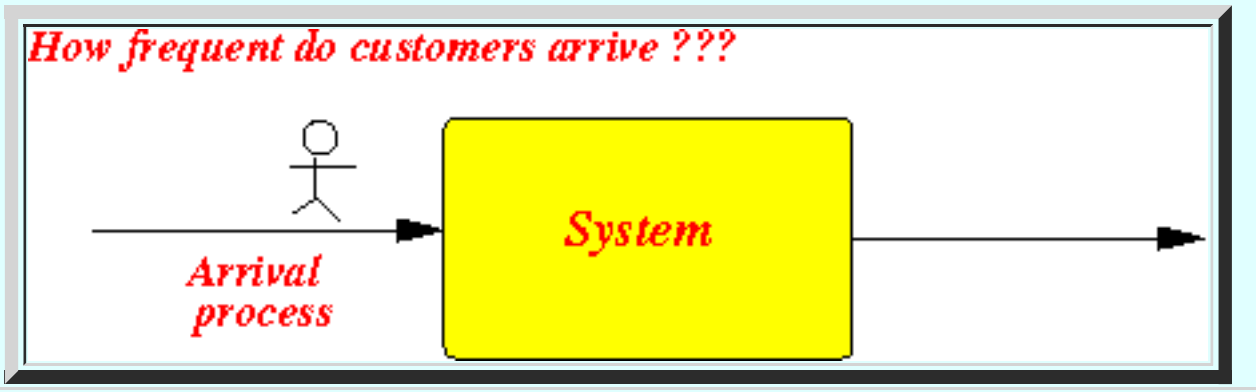
  - **Performance parameters** defined for ***centention* base networks**:
    - **G = offered load**
      - **G = number** of **packet transmission *attempts*** per **time unit**
    - **S = throughput**
      - **S = number** of ***successful* packet transmission** per **time unit**

- **Simplifying assumptions....**
  - **Fact about (Mathematical) analysis:**
    - Often ***very* difficult**
    - People ***always*** make ***assumptions*** to make the **Mathematical analysis** more **simple**
  - **Simplifying assumptions:**
    - **All packets** have the ***same* length**
      - **Result**: the **transmission time** for **every packet** is the **same duration**
    - The **packet transmission time** = **1 time unit**

- **Mathematical performance analysis**
  - **Fact:**

- In order to **study** the **performance** of a **system** with **Mathematical modeling**:

we must **model** the ***arrival*** **process** **analytically**:



- **Announcement:**

  - I will ***only*** state some **results** from **probability theory**....

    (This **material** is **beyond** the scope of a **networking** course....)

- **Mathematical modeling of message arrivals**

  - **Recall: Arrival rate (offer load)**

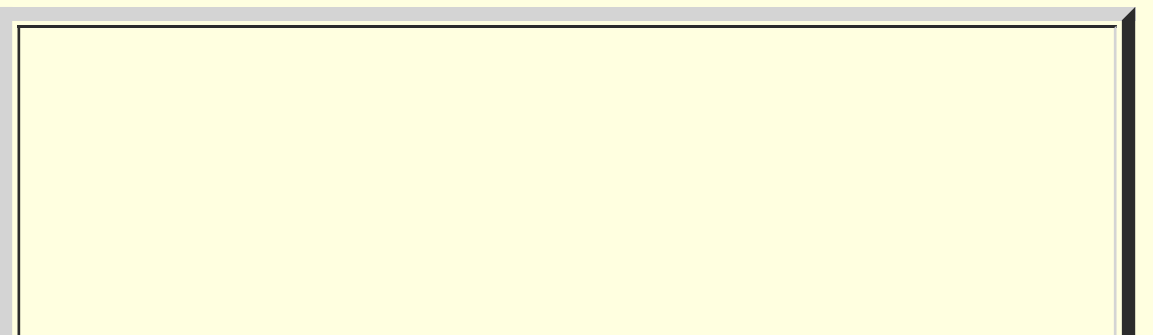
    - **Arrival rate (Offered load)** = **number of customers** that **arrives** to the **system** **per time unit (sec)**

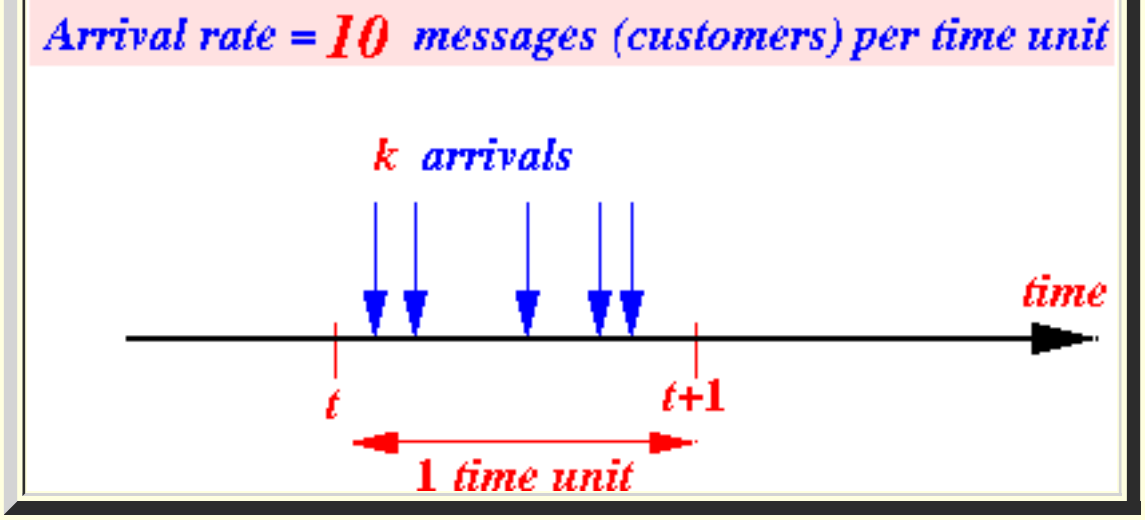
  - **Modeling** the **arrival** of **messages**:

    - **Modeling** the **arrival** of **messages** = **compute** the ***probability*** of a **certain number** of **arrivals** (for a given **arrival rate**)

    Example:

    - Suppose the ***average*** **arrival rate** of **customers** = ***10*** **customers/sec**:





Then:

- **What** is the **probability** that there are **0 customer arrival** in **1 sec** ??
- **What** is the **probability** that there are **1 customer arrival** in **1 sec** ??
- **What** is the **probability** that there are **2 customer arrivals** in **1 sec** ??
- **And so on...**

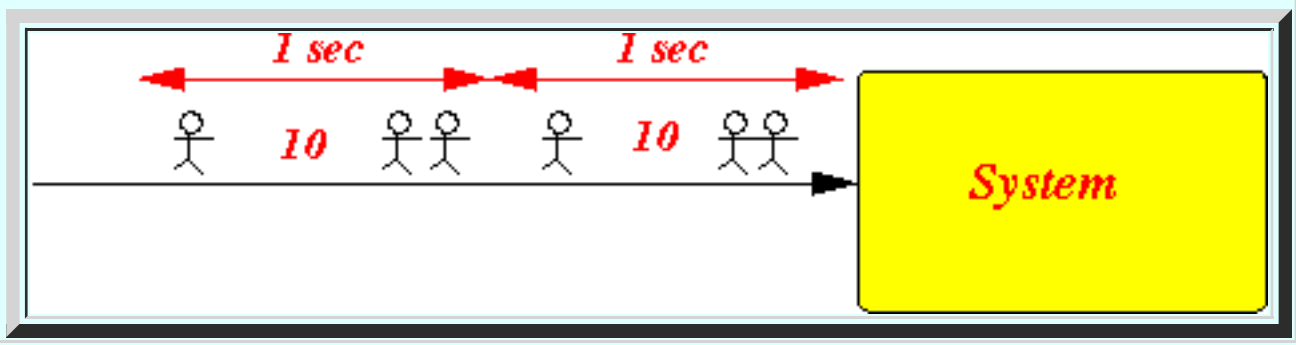
Give a ***formula*** to **compute** these ***probabilities***.

Note:

- An ***average*** **arrival rate** of **10 customer/sec** does **not** mean:

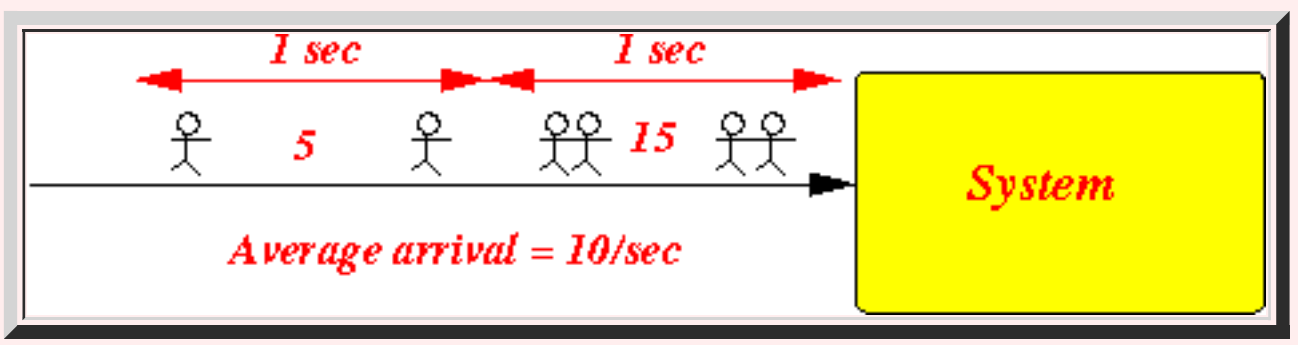
  - There are ***always*** **10 customers** arrive **every sec**.....

    Example:



- There can be **heavy** and **light** **arrival periods**

  Example:



- **Therefore:**

  - The **probability** that there are **0 customer arrival** in **1 sec** when the ***average***

arrival rate = 10 is ***not*** equal to **0** !!!

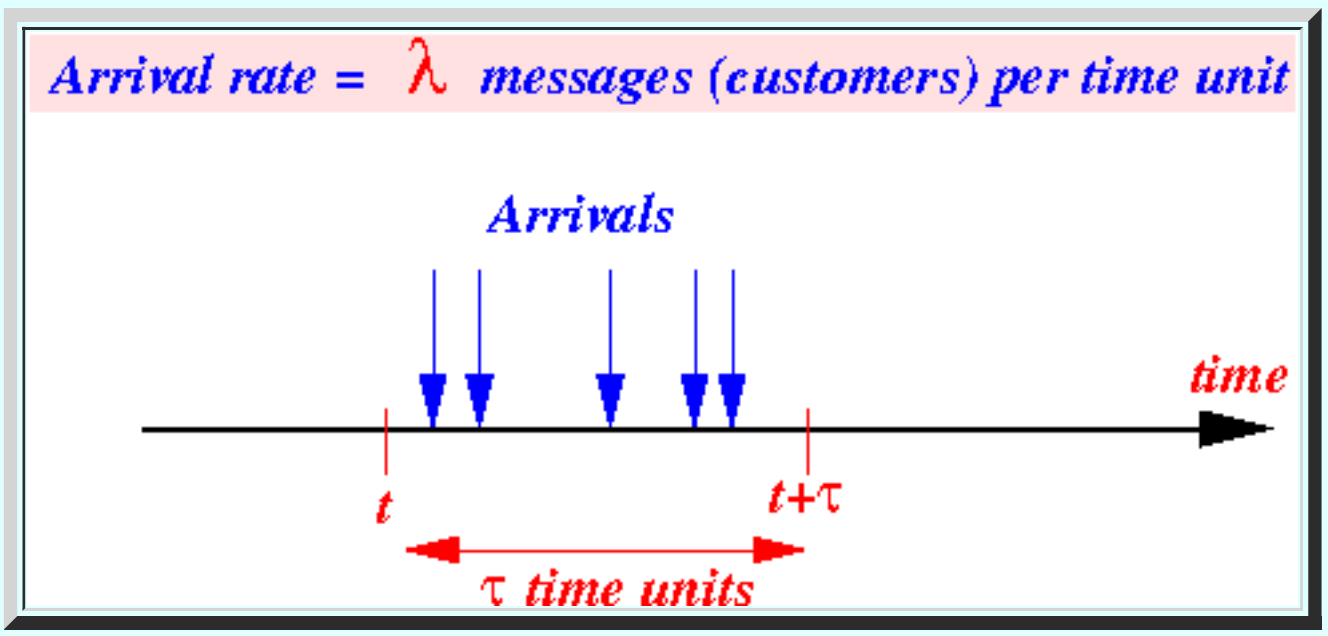
- **The Poisson arrival process**
  - The **most commonly used Mathematical model** for **customer arrivals** is:

    - **Poisson (arrival) process**

    Wikipedia page: [click here]

  - The **Poisson arrival (probability) distribution**:

    - Suppose the **(average) *arrival* rate** of **customers** = **λ**:



    Then:

    - the **probability** that there are ***k* arrivals** in a **τ sec** period is:

$$\text{Prob[} k \text{ arrivals in } \tau \text{ sec]} = \frac{e^{-\lambda\tau}\,(\lambda\tau)^k}{k!}$$

    (**e** = the **base** of the **natural *log*** function (= **0.2718281828....**))

  - **Example:**

    - **Arrival rate λ** = **10** (customers/sec)

    - **Then:**

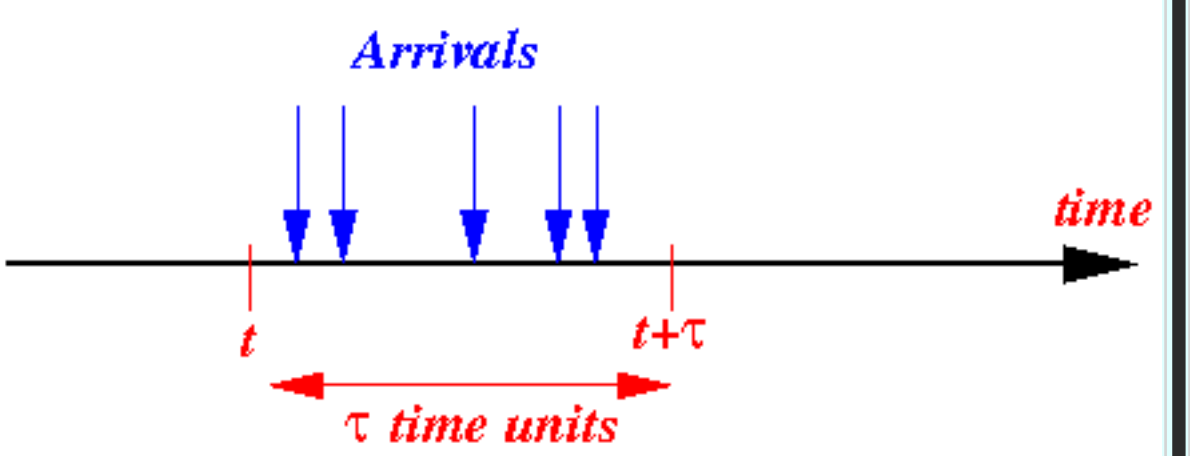
$$\text{Prob[0 arrivals in 1 sec]} = \frac{e^{-10*1}\,(10*1)^0}{0!} = e^{-10} = 0.00004539992$$

- **Aloha message arrival modeling using Poisson arrival process**
  - What does this **result** mean for **Aloha**:

    - The **arrival rate (λ)** in the **Aloha network** = ***G* messages** per **time unit**

    - **Therefore:** the **probability distribution** of the **number of arrivals** is:

      Arrival rate = G messages (customers) per time unit

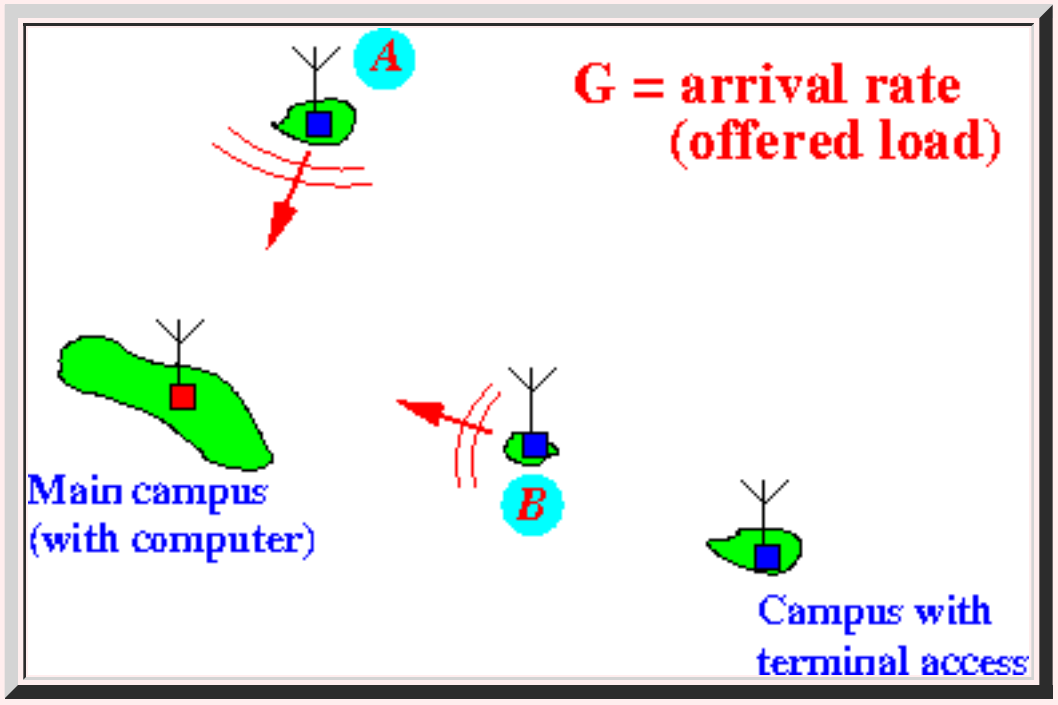
      

      $$P[\,k \text{ arrivals in } \tau\,] = \frac{(G\tau)^k\, e^{-(G\tau)}}{k!}$$

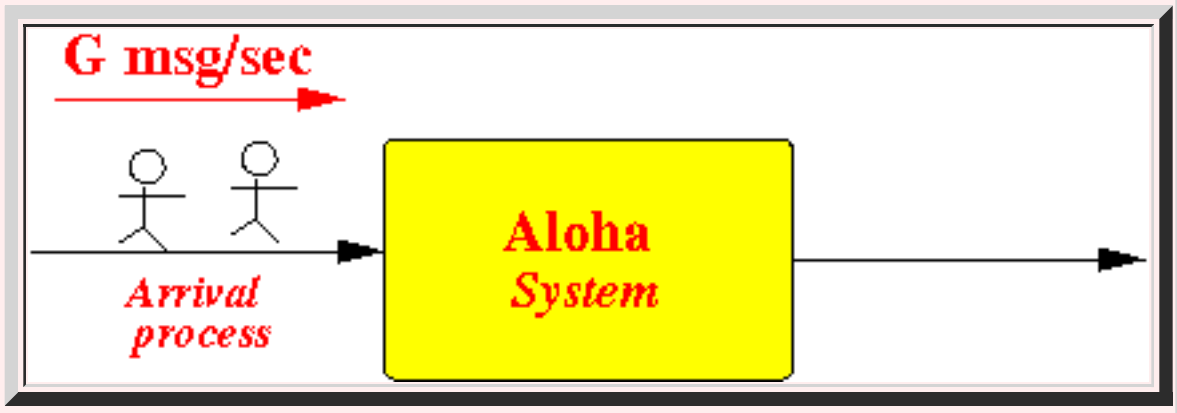
The initial performance Analysis of the Aloha protocol

- **Initial performance analysis of the Aloha protocol**
  - **Model description:**

    - An **average** of **G messages** are being **transmitted** per **sec**:



Analogy:



- We **know** that:

  - There may be **collisions**
  - Not **every transmission** will be ***successful***

  - **Define**:

```
p = the fraction of transmission attempts that are successful
```

- **Then** the **throughput *S*** is **equal** to:

```
   S  =  # successful transmissions  per sec        (definition !!!)

      =  # transmission attempts  per sec  ×  fraction that are successful
      =  G × p

So:

   S = p × G
```

- **$64,000 question**: how do we **compute** `p` ???

```
   p = ?????
```

Fact:

- `p` is ***different*** for
  - ***slotted*** Aloha
  - ***unslotted*** Aloha

- **Definition of the time unit**
  - **Fact:**
    - You can **often *simplify*** the **Mathematical analysis** by:
      - **Adopting** a ***different*** **time unit**

  - **Recall** the follow **assumption**:
    - **Each message (frame)** has the ***same*** **length** (= same **transmission time**)

  - **Time unit** used in the **performance analysis**:

- **Time unit** = **transmission time** of a **message**

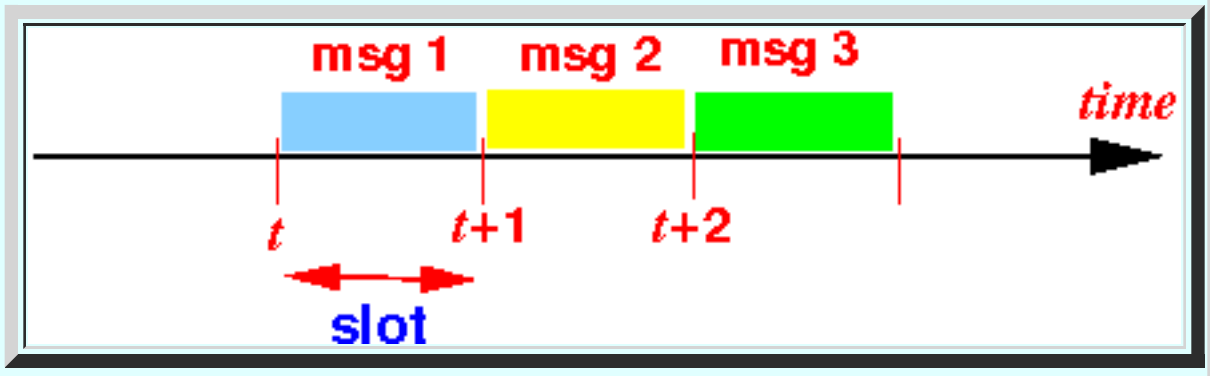
- **Note:**

  - In ***slotted*** **Aloha**, we ***also*** have:

    `1 time unit =  length of 1 slot`

  Reason:

  - **Because *one* message** is **transmitted** in ***one*** **slot**:

    We ***also*** have:

    - **Time unit** = **length (duration)** of **1 slot** !!!

- **An important fact of probability**

  - **Fact:**

    - The **probability** of some **event *X*** = the ***fraction*** of **event *X***

  - **Example:**

    - We **roll** a **die**

      Then:

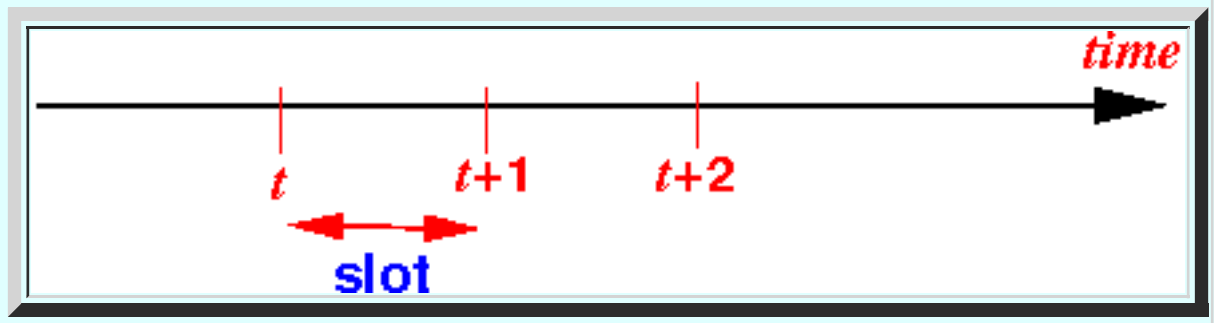
      `Probability of rolling a 6  = 1/6`

```
Fraction of rolls equal 6    = 1/6
```

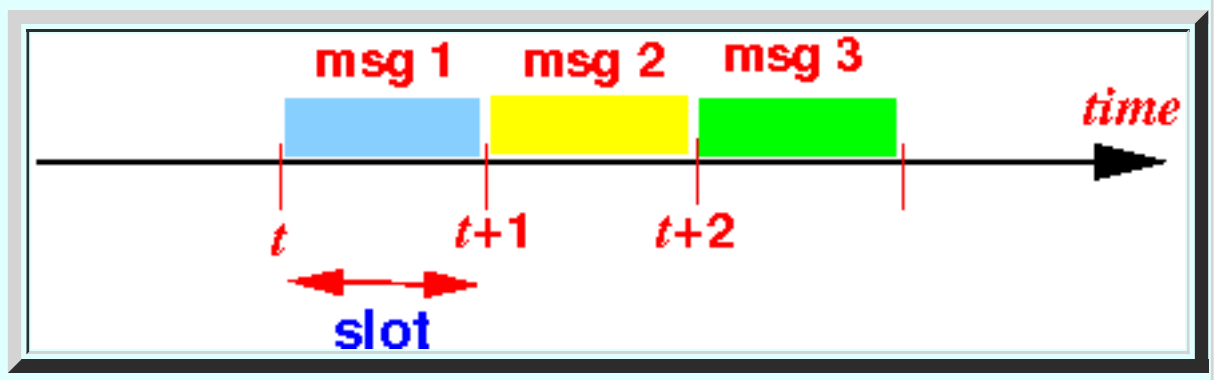
Performance Analysis of the *slotted* Aloha protocol

- **Reminder: *Slotted* Aloha**
  - **Recall**: **Slotted Aloha:**

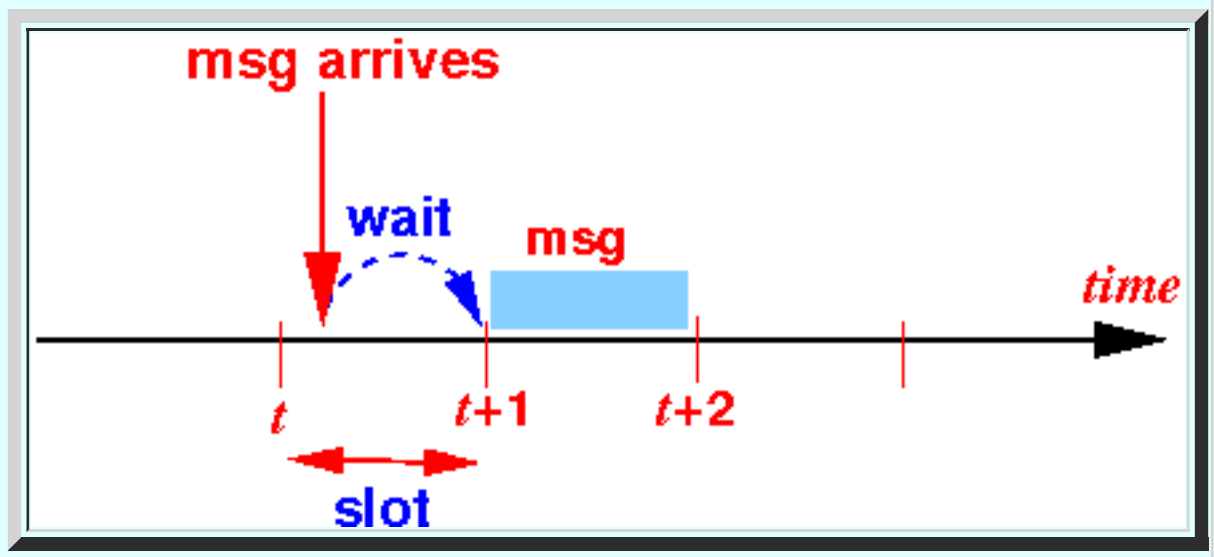
    - Time is **divided** into **slots**



    - A **transmission** can ***only*** **begin** at the **start** of a **slot**:



    - When a **message *msg*** arrives:



    the **message *msg*** must ***wait*** until the ***start* of a slot** to be **transmitted**

- **Choice of time unit**
  - **Fact:**

    - **Normally**, we use:

      - **time unit** = **second**

- **However:**

  - **Expressions** can **often** be **simplified** using a ***different*** **time unit**

- **Time unit** used in **Aloha** analysis:

  - **1 time unit** = **length** of **1 message**
    = **length of 1 slot**

- **Reminder: Throughput**

  - **Throughput S**:

    S = p × G

    Where:

    S = throughput
    G = offered load (arrival rate)
    p = fraction of transmissions that are successful
    = probability that a transmission is successful

  - **Converting S** and **G** for changing **time unit**:

    - **G** = **# messages** that **arrives** per **time unit**

      - We may need to **convert** the value when the **time unit** changes

    ---

    - **Example:**

      - Suppose **G** = **5 arrivals** per **sec**
      - **Length** of **message** = **2 sec**

        Then:

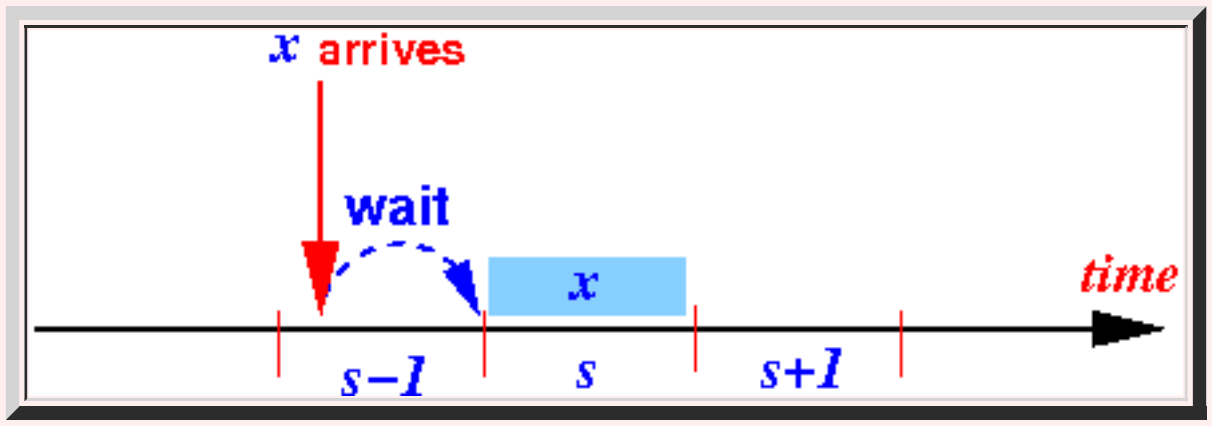
        # arrivals = 5 in 1 sec (5/sec)
        = 10 in 2 sec
        = 10 per "length of message"

        G = 10 / time unit (with message length as time unit)

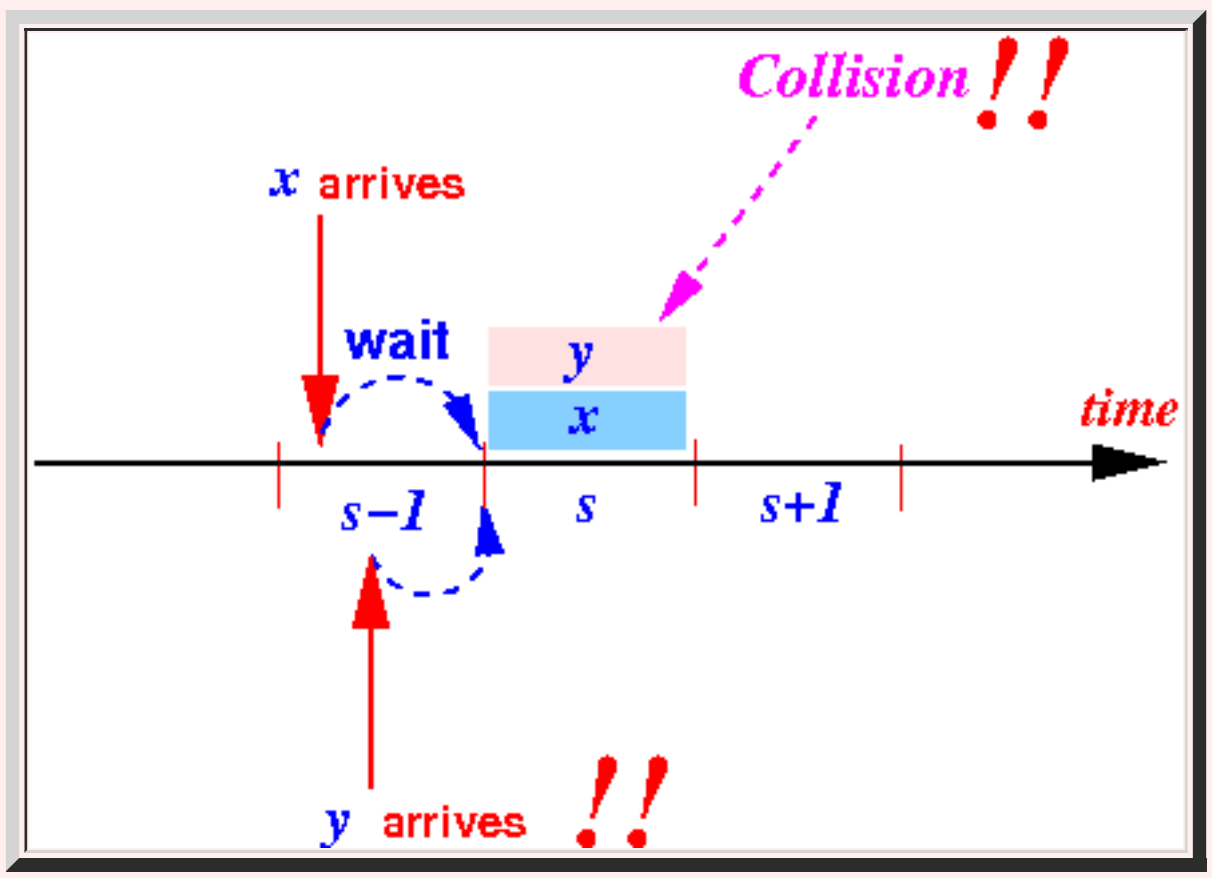
- **Performance analysis of the *slotted* Aloha protocol**

  - **When** is a **transmission *successful***:

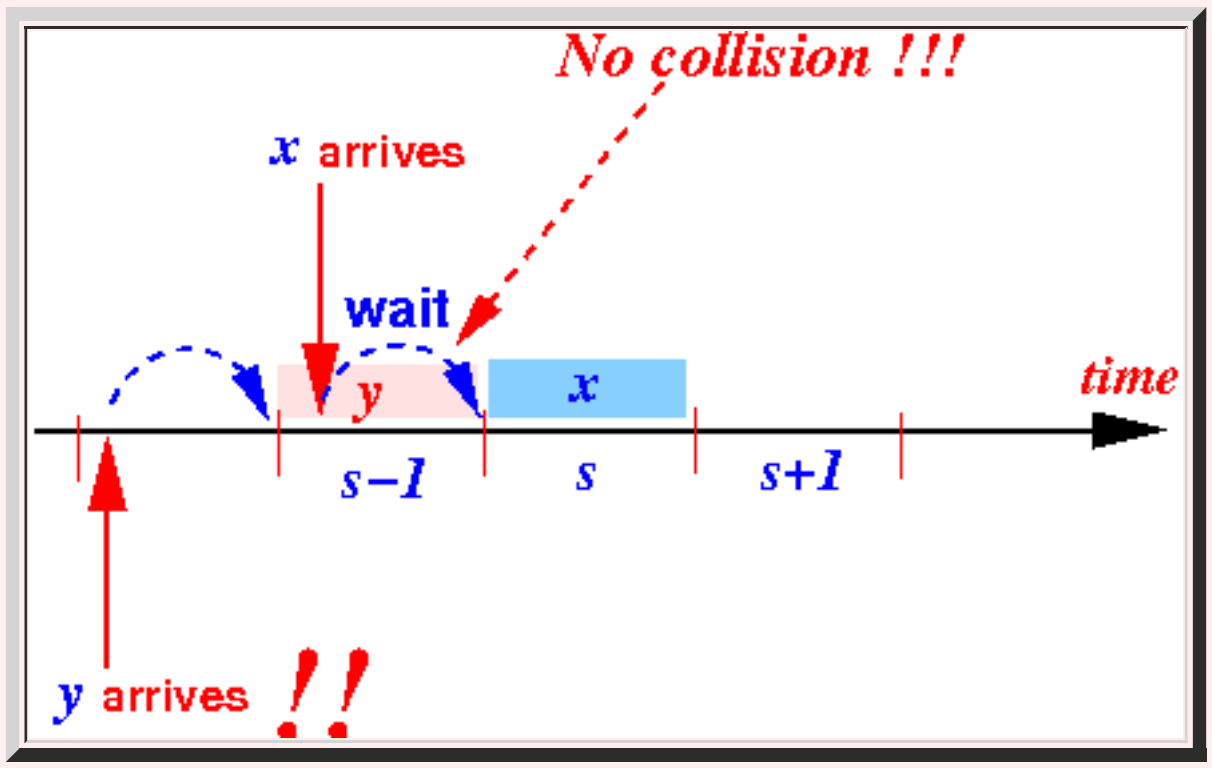
- **Consider** a **transmission *x*** that **arrives *prior*** to a **slot *s***:



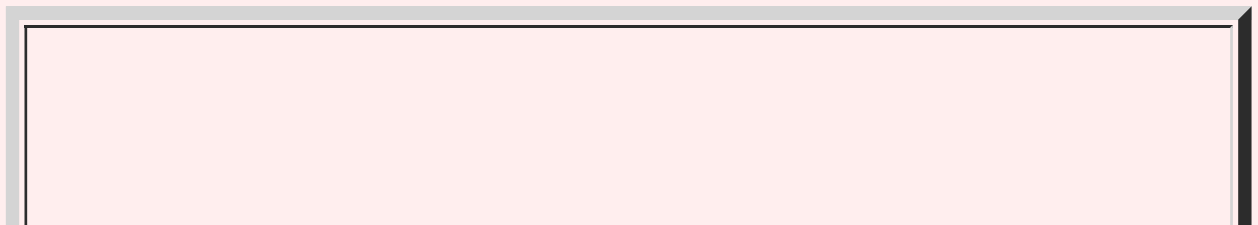
- A **tranmission *y*** that **arrive *during*** the **slot "*s*−1"** will ***collide*** with the **transmission *x***:

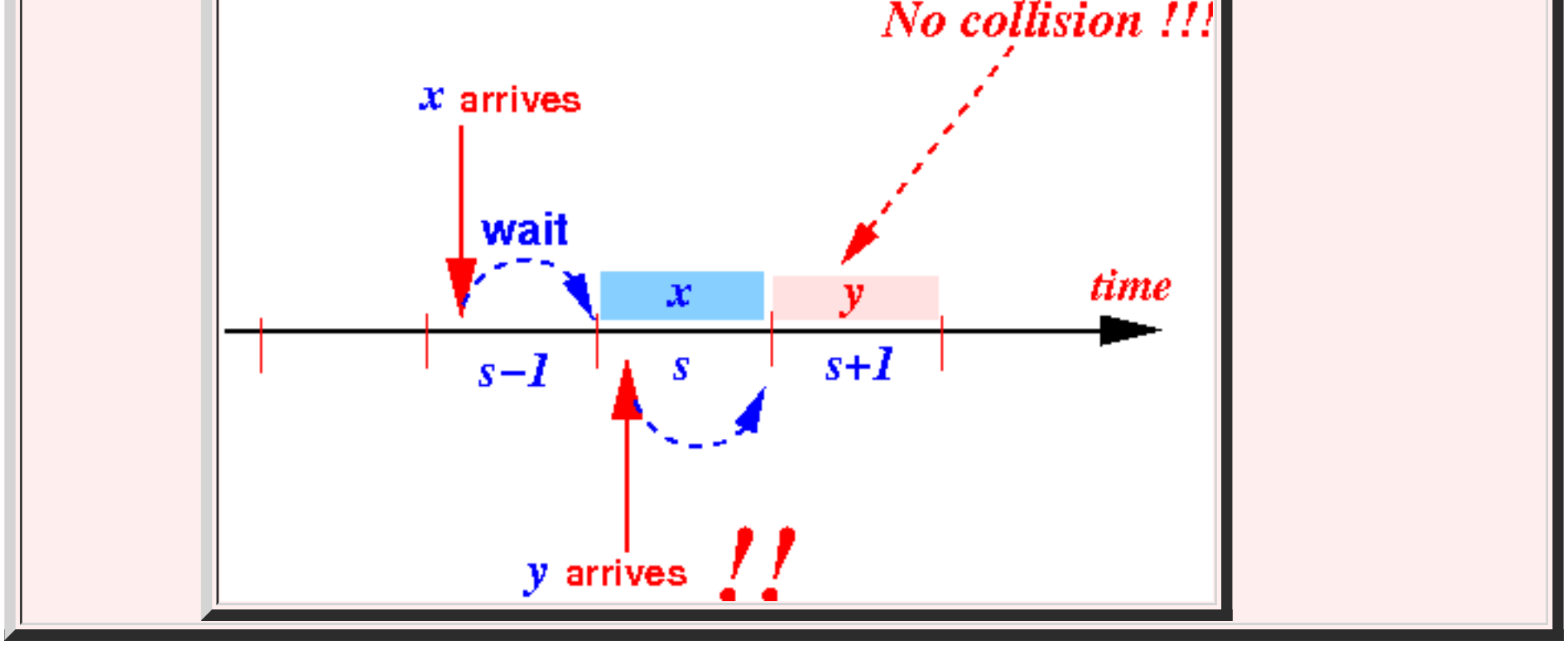


- **Transmissions** that **do *not* arrive *during*** the **slot "*s*−1"** will ***not*** **collide** with the **transmission *x***:



Or:





- ***Conclussion***:

  - A **transmission *x*** in **slot *s*** is **successful** if and only if:
    - There are ***no* (zero) message arrivals during** the **slot *s* – 1**

- **Throughput** of the ***slotted* Aloha**:

Throughput S = #successful transmissions per time unit (= slot)
= Total # transmissions × fraction successful
= G × fraction successful
= p × G

---

p = fraction of successful transmission

= probabability that a transmission in slot is successful

= probability that no (0) message arrives in previous slot

= Prob[0 arrivals in 1 time unit (t.u.)]

Recall Poisson arrival process:

$$\text{Prob[} k \text{ arrivals in } T \text{ t.u.]} = \frac{e^{-GT}\,(GT)^{k}}{k!} \qquad (G = \text{arrival rate})$$

$$\text{Prob[} 0 \text{ arrivals in } 1 \text{ t.u.]} = \frac{(G \times 1)^{0}\, e^{(-G \times 1)}}{0!}$$

$$= e^{-G}$$

Hence:

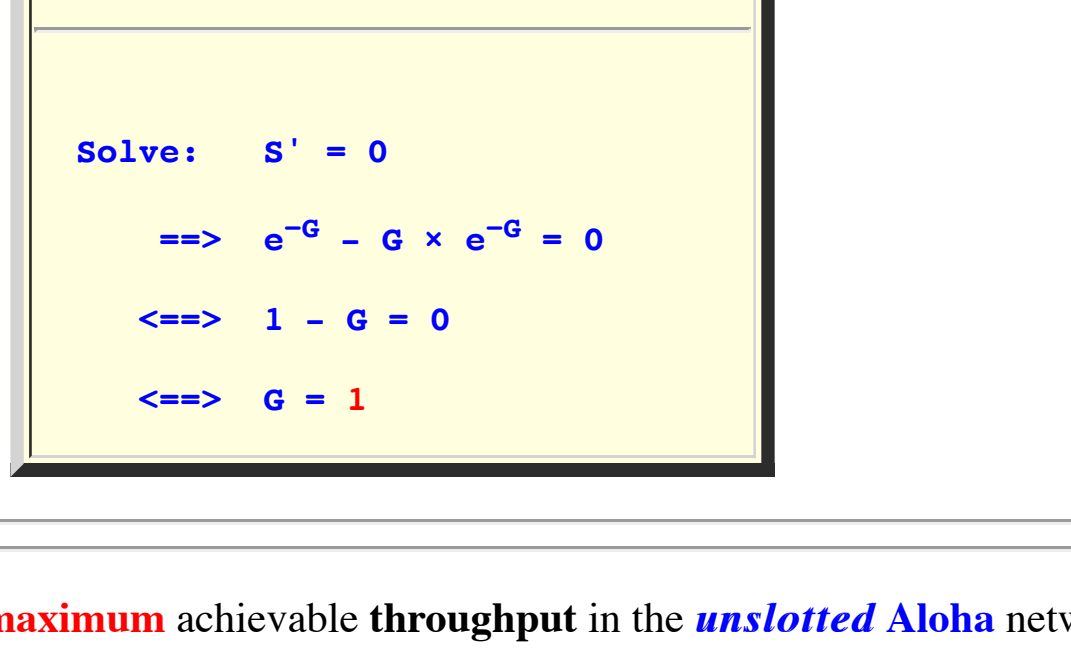
Throughput S = p × G

$$= e^{-G} \times G$$
$$= G\ e^{-G}$$

- **Summary:** throughput of the ***slotted*** **Aloha** protocol

$$S = G \times e^{-G} \quad \text{(Slotted Aloha)}$$

- **Graph of *Slotted* Aloha's throughput (S) vs. offered load (G)**:



- **Maximum throughput of the *Slotted* Aloha protocol**

- **Previously:**

Throughput $S = G \times e^{-G}$ (Slotted Aloha)

- The **maximum** can be found through ***differentiation***:

$$S = G \times e^{-G}$$

$$S' = e^{-G} - G \times e^{-G}$$

Solve: $S' = 0$

$$\Longrightarrow \quad e^{-G} - G \times e^{-G} = 0$$

$$\Longleftrightarrow \quad 1 - G = 0$$

$$\Longleftrightarrow \quad G = 1$$

- The **maximum** achievable **throughput** in the ***unslotted*** **Aloha** network is:

$$S(1.0) = 1.0 \times e^{-1.0}$$
$$= 1.0 \times 0.368$$

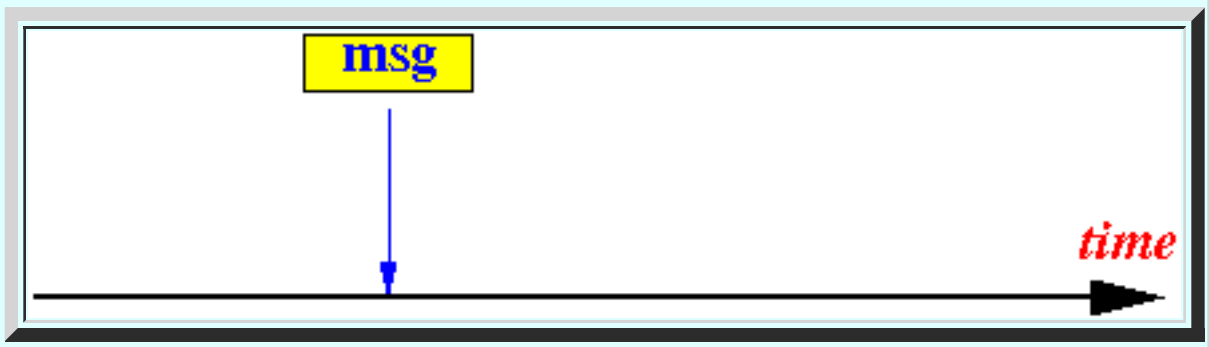
```
= 0.37
```

Maximum channel utilization with **slotted Alpoha** = **37%** (not very good)

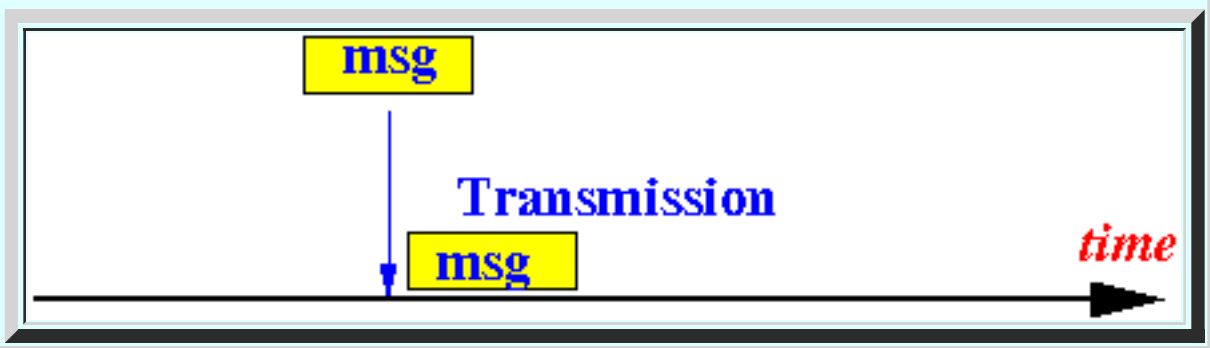
Performance Analysis of the *unslotted* Aloha protocol

- **Reminder: *unslotted* Aloha**
  - **Recall**: **unslotted Aloha:**

    - When a **message *msg*** arrives:



    it will be **transmitted** at the ***same* time** as its **arrival moment**:



- **Reminder: Throughput**
  - **Throughput S**:

$$S = p \times G$$

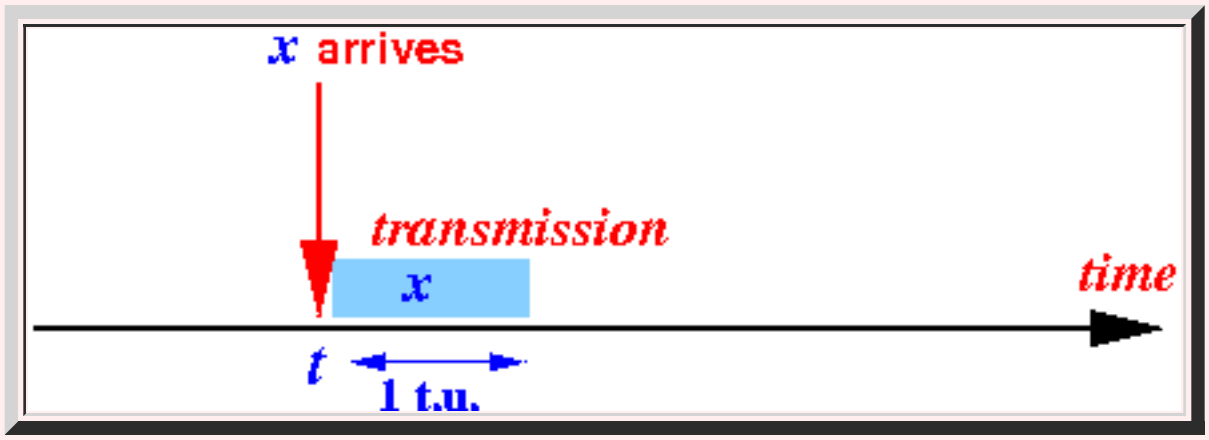
Where:

S = throughput
G = offered load (arrival rate)
p = fraction of transmissions that are successful
  = probability that a transmission is successful

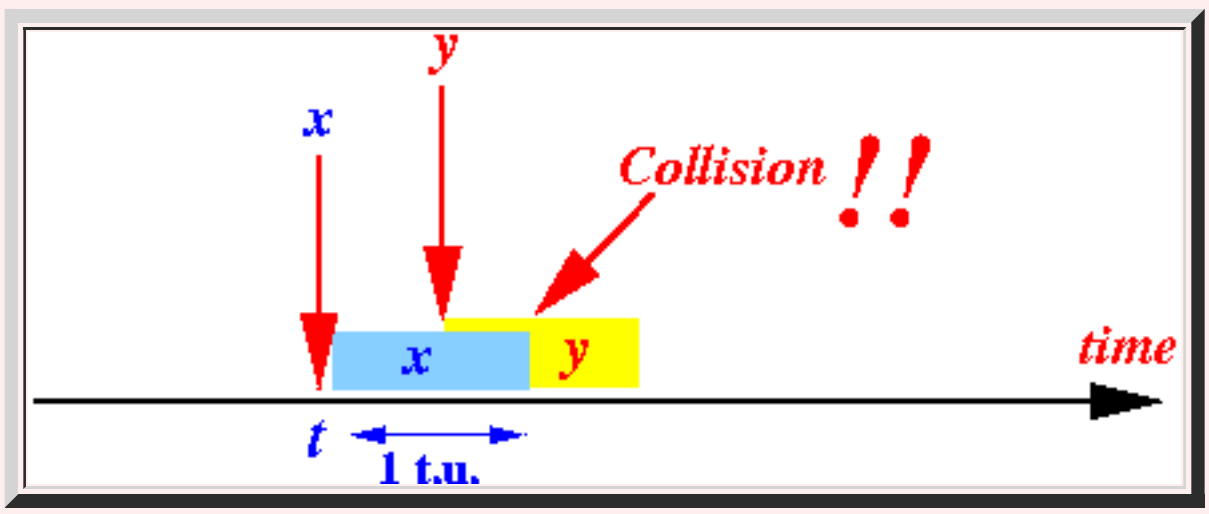
- **Performance analysis of the *unslotted* Aloha protocol**

- **When** is a **transmission** ***successful***:

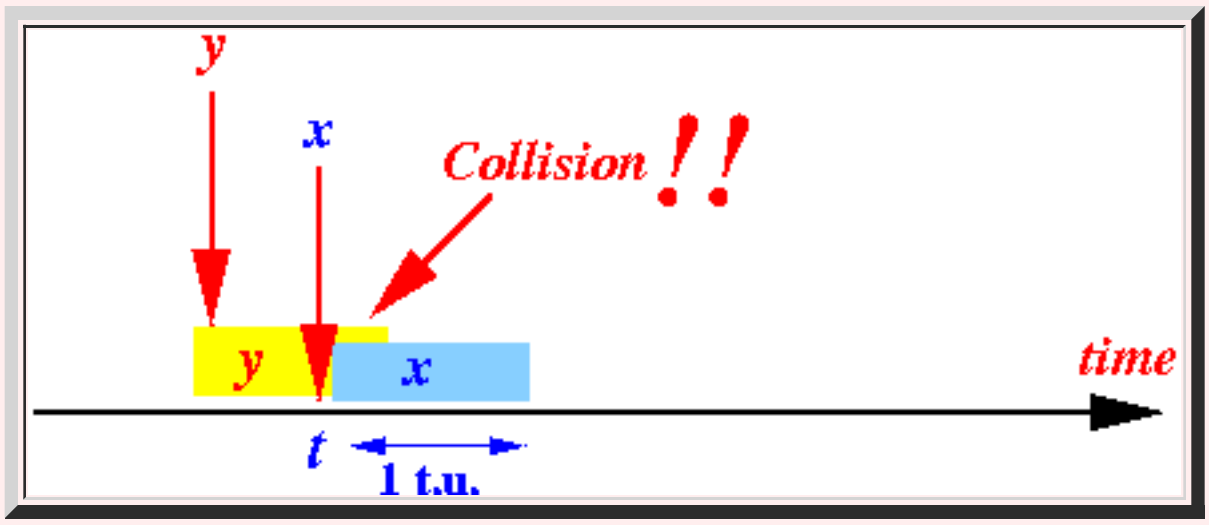
  - **Consider** a **transmission** ***x*** that **arrives** at time ***t***:



  - A **tranmission** ***y*** that **arrive** ***during*** the **transmission** of ***x*** will **collide** with the **transmission** ***x***:

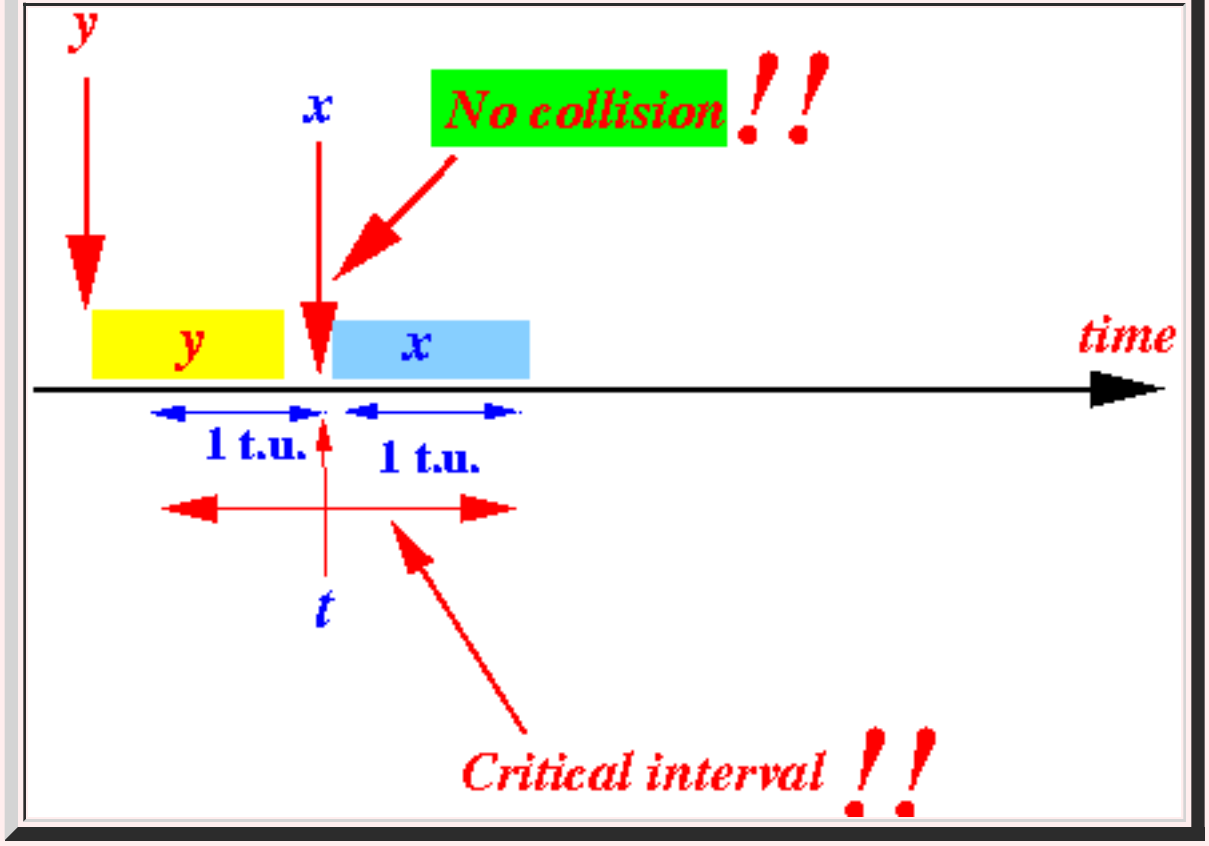


  - A **tranmission** ***y*** that **arrive 1 time unit** ***before t*** will ***also*** **collide** with the **transmission** ***x***:

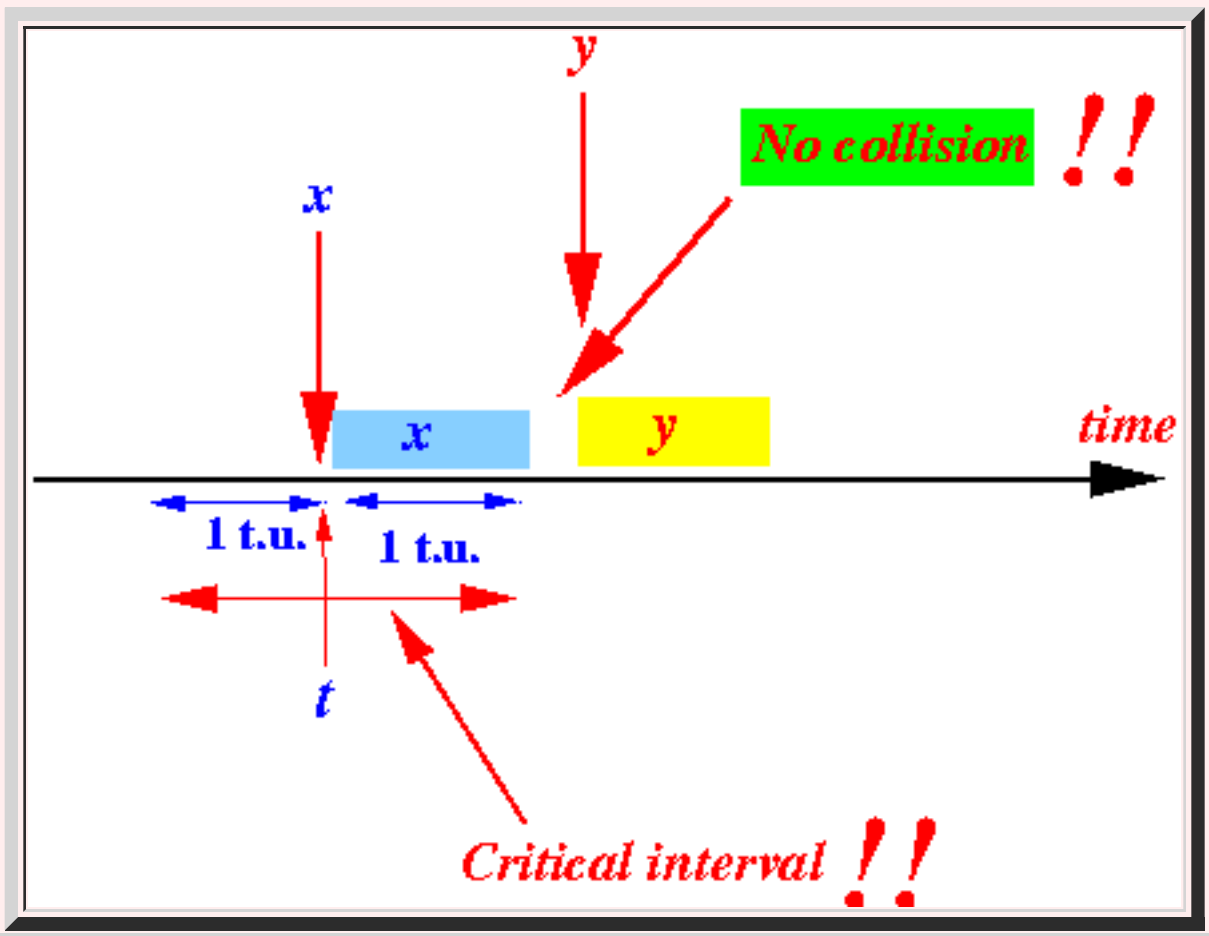


  - **Transmissions** that **arrive** ***outside*** the **critical interval (see figure below)** will ***not*** **collide** with the **transmission** ***x***.

    Before:



Or after:



- ***Conclusion***:

  - A **transmission *x*** is **successful** if and only if:

    - There are ***no* (zero) message arrivals during *2* time units**

- **Throughput** of the ***unslotted* Aloha**:

Throughput S = #successful transmissions per time unit (= slot)
= Total # transmissions × fraction successful
= G × fraction successful
= p × G

---

p = fraction of successful transmission

= probabability that a transmission in slot is successful

= probability that no (0) message arrives in previous slot

= Prob[0 arrivals in 2 time unit (t.u.)]

Recall Poisson arrival process:

$$\text{Prob[} k \text{ arrivals in } T \text{ t.u.]} = \frac{e^{-GT}\,(GT)^{k}}{k!} \qquad (G = \text{arrival rate})$$

$$\text{Prob[} 0 \text{ arrivals in } 2 \text{ t.u.]} = \frac{(G \times 2)^{0}\, e^{(-G \times 2)}}{0!}$$

$$= e^{-2G}$$

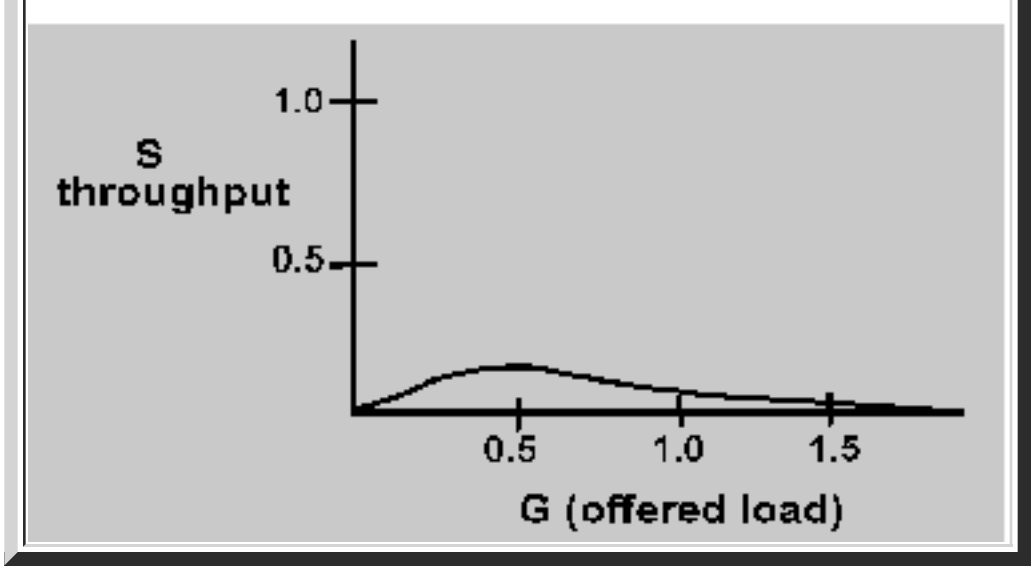
Hence:

Throughput S = p × G
= e^{-2G} × G
= G e^{-2G}

- **Summary:** throughput of the ***unslotted* Aloha** protocol

$$S = G \times e^{-2G} \quad \text{(Unslotted Aloha)}$$

- **Graph of *Slotted* Aloha's throughput (S) vs. offered load (G)**:



- **Maximum throughput of the *Slotted* Aloha protocol**
  - **Previously:**

$$\text{Throughput} \quad S = G \times e^{-2G} \quad \text{(Slotted Aloha)}$$

  - The **maximum** can be found through ***differentiation***:

$$S = G \times e^{-2G}$$

$$S' = e^{-2G} - 2G \times e^{-2G}$$

Solve: $S' = 0$

$$\Longrightarrow \quad e^{-2G} - 2G \times e^{-2G} = 0$$

$$\Longleftrightarrow \quad 1 - 2G = 0$$

$$\Longleftrightarrow \quad 2G = 1$$

$$\Longleftrightarrow \quad G = 1/2 = 0.5$$

  - The **maximum** achievable **throughput** in the ***unslotted* Aloha** network is:

$$S(G) = G \times e^{-2G} \qquad ; \text{ max for } G = 0.5$$

$$S(0.5) = 0.5 \times e^{-1.0}$$
$$= 0.5 \times 0.368$$
$$= 0.18$$

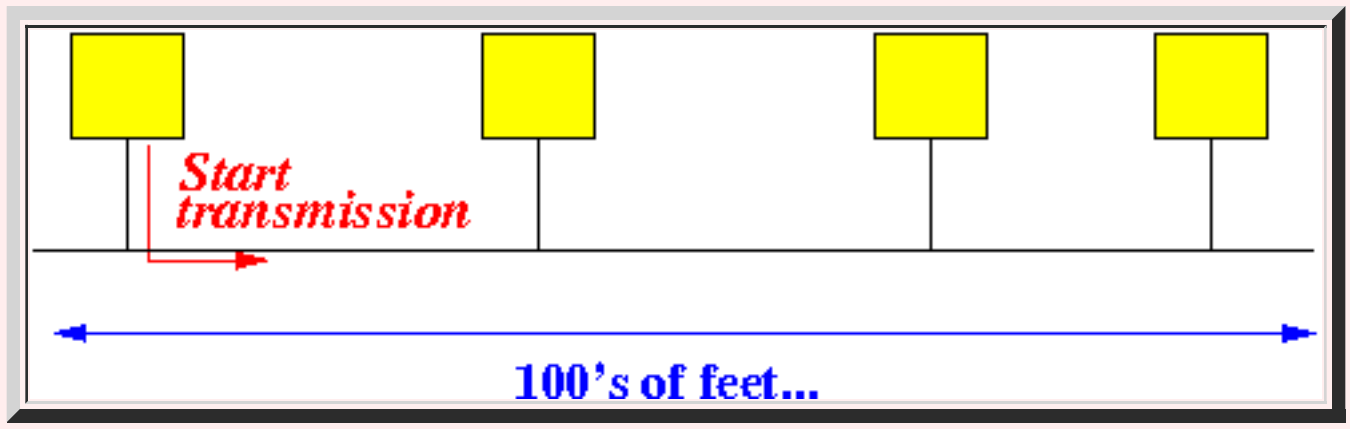
Maximum channel utilization with **unslotted Alpoha** = **18%** (not very good)

Intro to *Local Area* networks

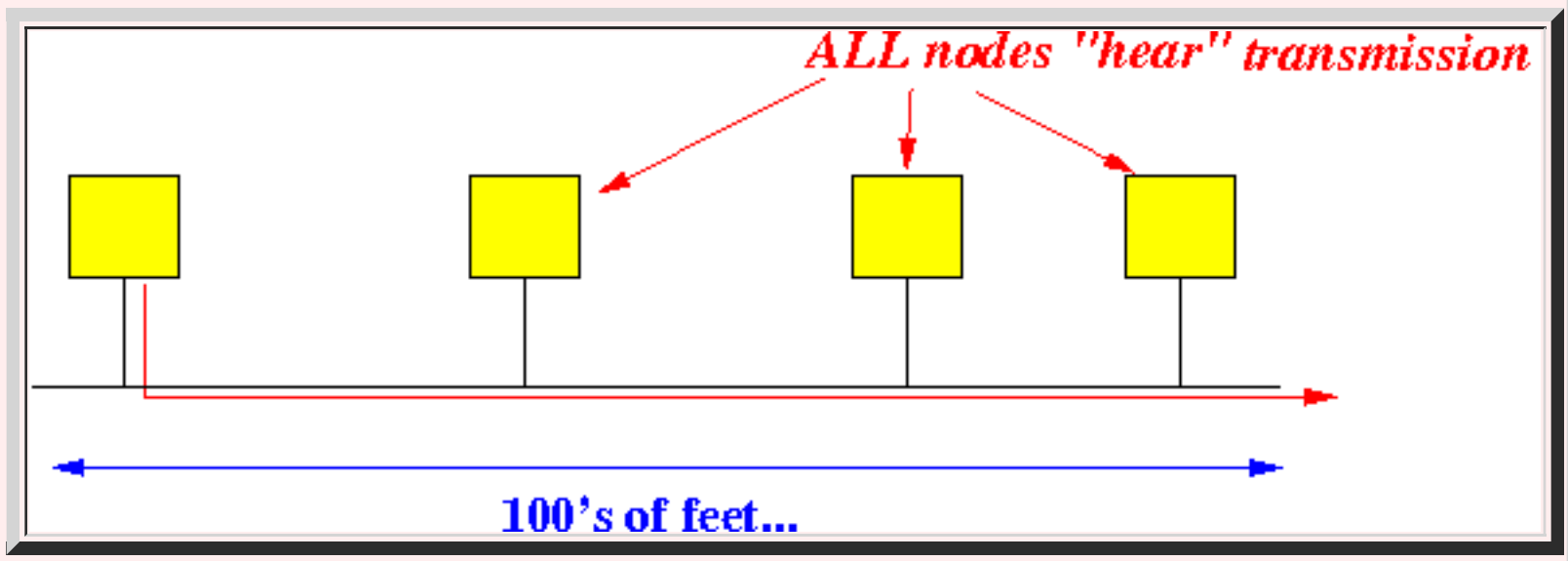
- **"Local Area" networks (LANs)**
  - **Fact:**
    - **Local Area networks** spans a ***small*** **geographic area**

    Example:
    - An (office) **building** (**≤ 500 m** ~= **1/3 mile**)
      (E.g.: **MSC** - this building...)
  - **Consequence** of a ***short*** **distance**:
    - **Transmissions** takes ***very*** **short time** to reach ***all*** **nodes** in the **network**

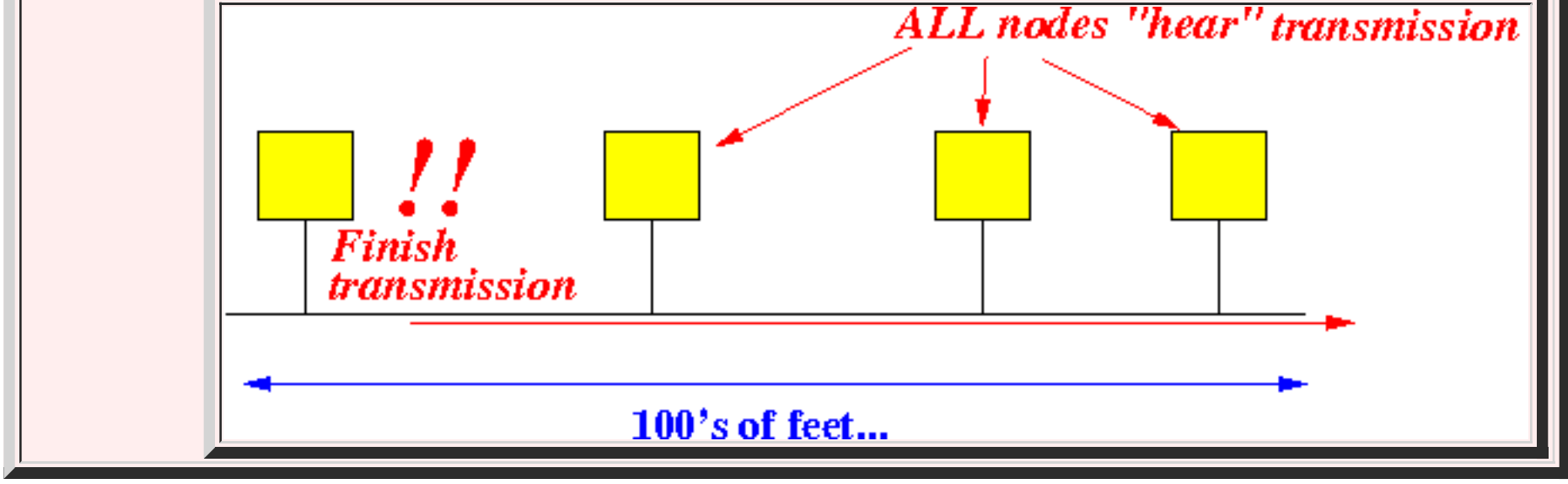
- **Important property of LANs**
  - **Important property**:
    - When a **node *starts* transmitting** a **message (data frame)**:



the **transmission** will be **"heard" (sensed)** by ***all*** **nodes**:

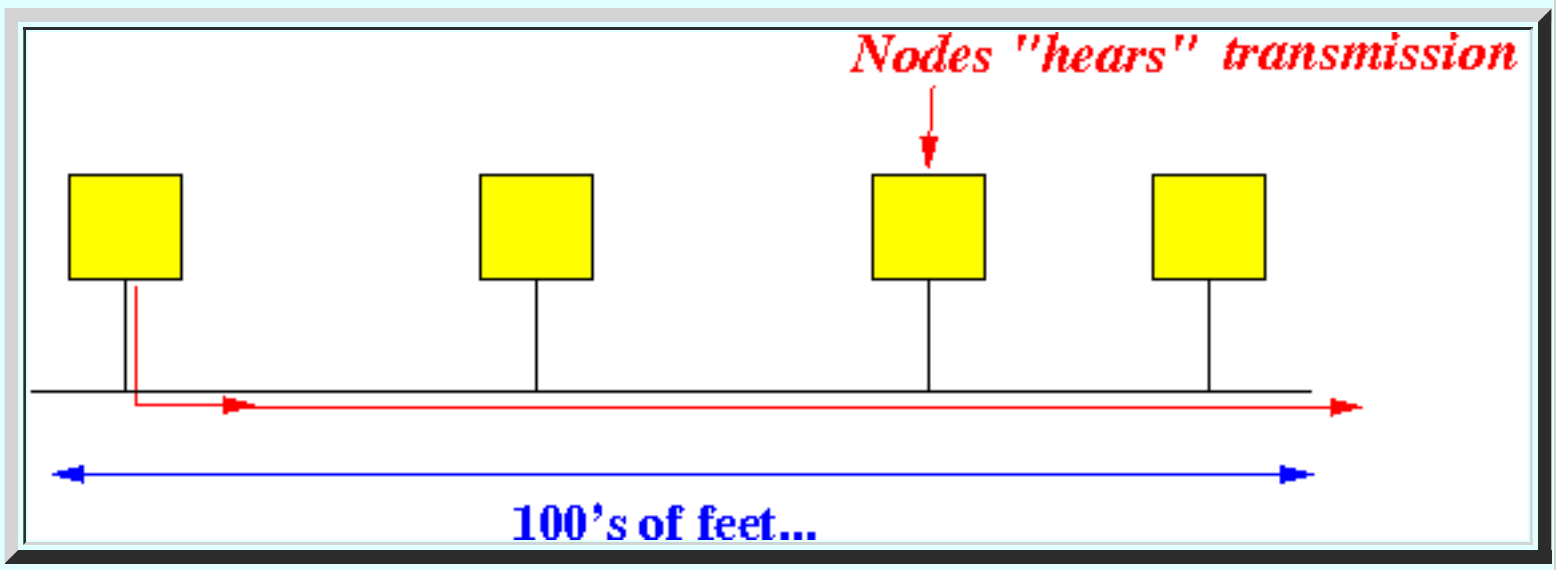


before the **node *finish* transmitting**:

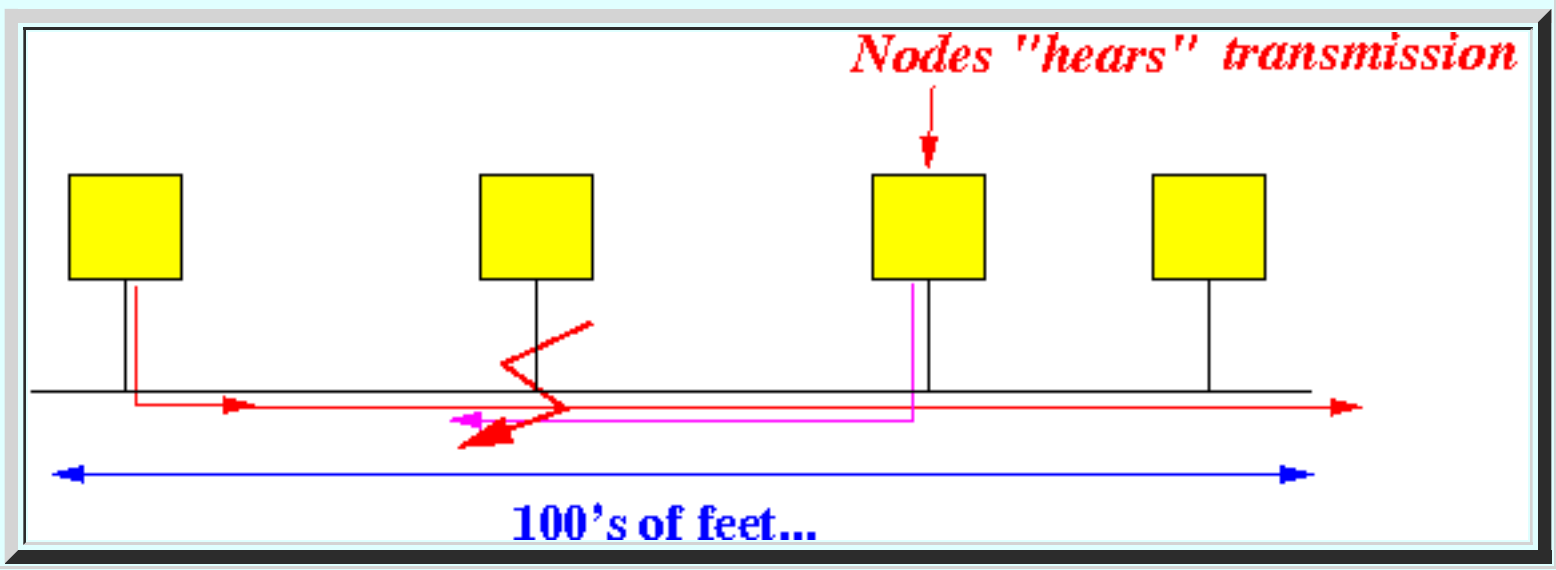


- **Therefore:**

  - **When** a **node** **"hears"** **(senses)** a **transmission**:



  Then its ***own*** **transmission** will ***very likely*** cause a **collision**:



- **Carrier sensing in LAN**

  - **Fact:**

    - **Carrier sensing** can **reduce** the **collision probability** in ***Local*** **Area Networks**

- **A property of LANs**
  - **Fact:**

    - **Message transmission** over **500 m** of **coaxial wire** or **twisted pairs** is **virtually *error free*** (if there are ***no*** **collision**)

  - **Design decision:**

    - Due to the **extremely low** **error rate**:

      - The **data link layer** of **LAN protocols** does **not** use **sliding window (ACK) schemes**

        (I.e., **retransmission methods** are ***not*** built in to the **data link layer**)

  - **Notice**

    - **Although errors** are ***extremely*** **rare**:

      - There **can** be **collision** on a **CSMA** network

      - The **sender** will ***still*** have to **recover** from these ***rare*** **frame losses**

    - The **(design) decision** is:

      - Do **not** implement **reliability** in the **data link layer**

        (Because it is **not** worth the **trouble** --- **frame losses** are **rare**)

      - The **reliability** of the **data transmissions** will be **handle** in the ***Transport*** **Layer**

Example:

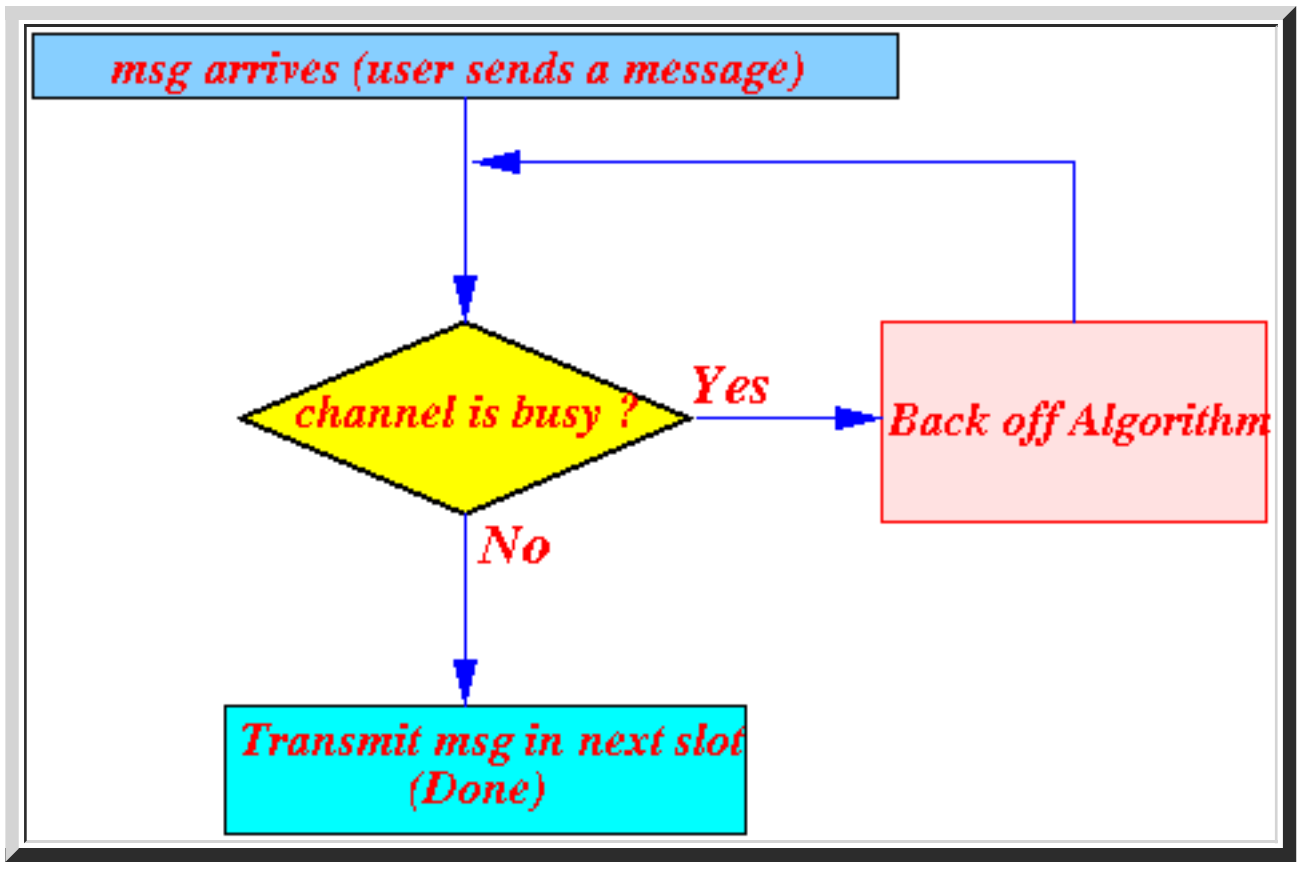
- The **TCP protocol** (in the ***Transport*** **Layer**) has **time out** and **retransmit** functions

  So:

  - The **TCP layer** will provide the **reliabilty** functionality for the **CSMA** protocol

Intro to the "Carrier Sense Multiple Access" (CSMA) protocol

- **Carrier Sense Multiple Access (CSMA) protocols**
  - **The *general* CSMA protocol:**



Explanation:

1. When a node has a **message** to transmit:

   - The **node** will ***sense*** the **(broadcast) transmission channel** (= **"carrier"**)

2. If the **(transmission) channel** is ***idle***:

   - The **node transmits** the **message**

     (The **message** will be **considered** to be **transmitted *correctly*** because **error rate** is ***low***)

Otherwise:

   - The **node** will ***back off*** (= **wait** some time)
   - **retry** after **waiting** the **back off period**

- **Different *variants* (types) of CSMA protocols**
  - **CSMA *Variants***:

    - There are **3 *different* (variants)** of **CSMA**

(They all **operate** in a ***similar*** **manner**)

- The ***different*** **variants** of **CSMA protocols** differents in the ***back off*** **algorithm** used.

- The **3 types of CSMA protocols** are:

  - **Non-persistent CSMA** (the "whimp")
  - **1-persistent CSMA** (the "stalker")
  - **p-persistent CSMA** (the "gambler")

- **Nomenclature: "Carrier Sense" and "Multiple Access"**

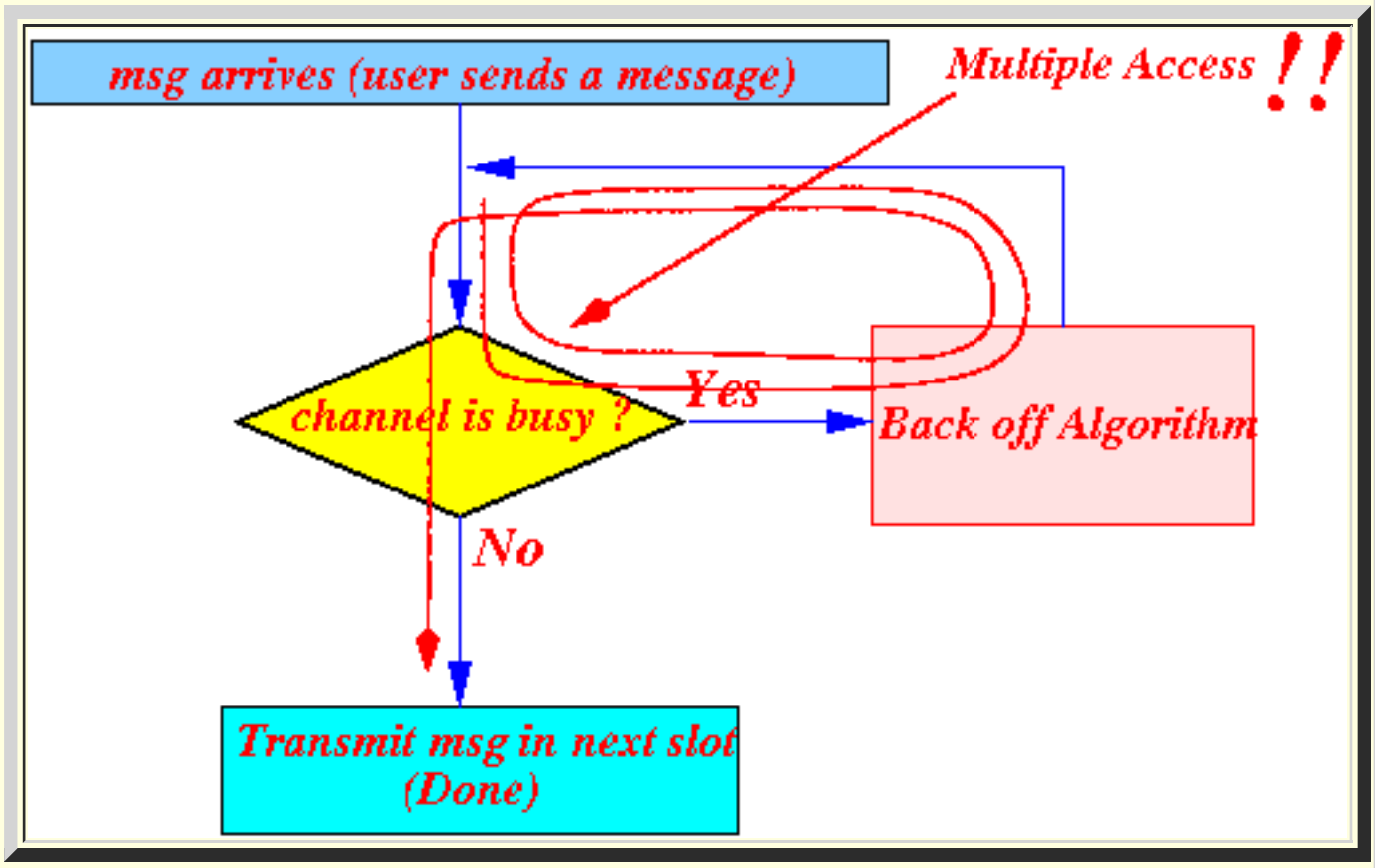
  - **Why** are these **protocols** called **Carrier Sense Multiple Access**:

    - **Carrier sense**:

      - because the **CSMA protocol** requires **nodes** to ***senses*** the **carrier (= transmission channel)**

    - **Multiple Access**:

      - because the **CSMA protocol** must try ***multiple*** **times** before it will **succeed** in **transmitting** a **message**:



(= **multiple access**)

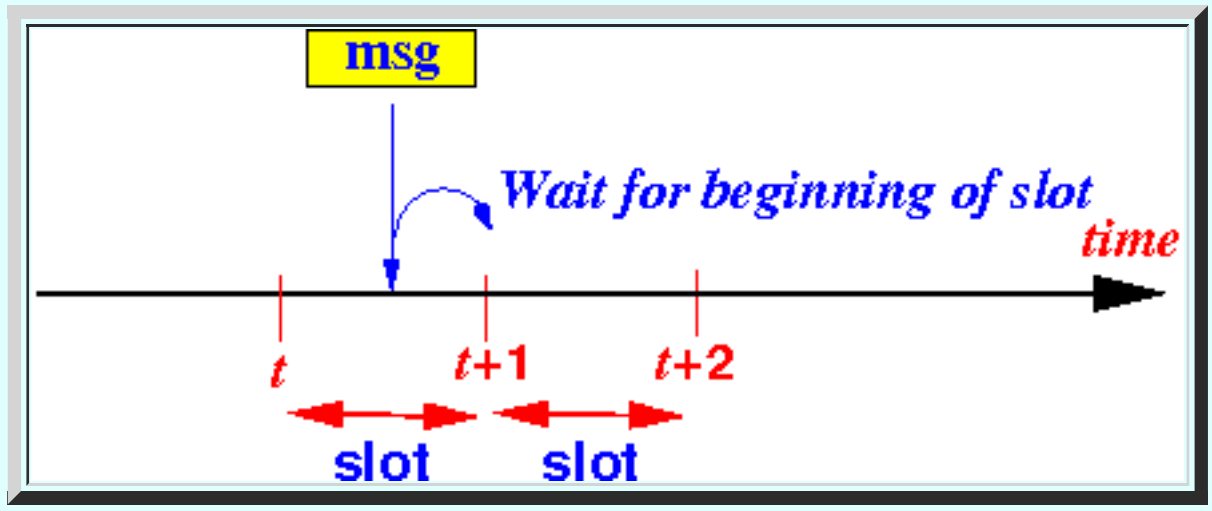
- **Slotted transmission in CSMA**
  - **Fact:**

    - **CSMA** uses ***slotted*** **transnissions** to **improve** ***performance***

      (***Slotted*** **Aloha** has **better** performance than ***unslotted*** **Aloha** !!!)

  - The **generalized** ***slotted*** **operation** of **CSMA**:

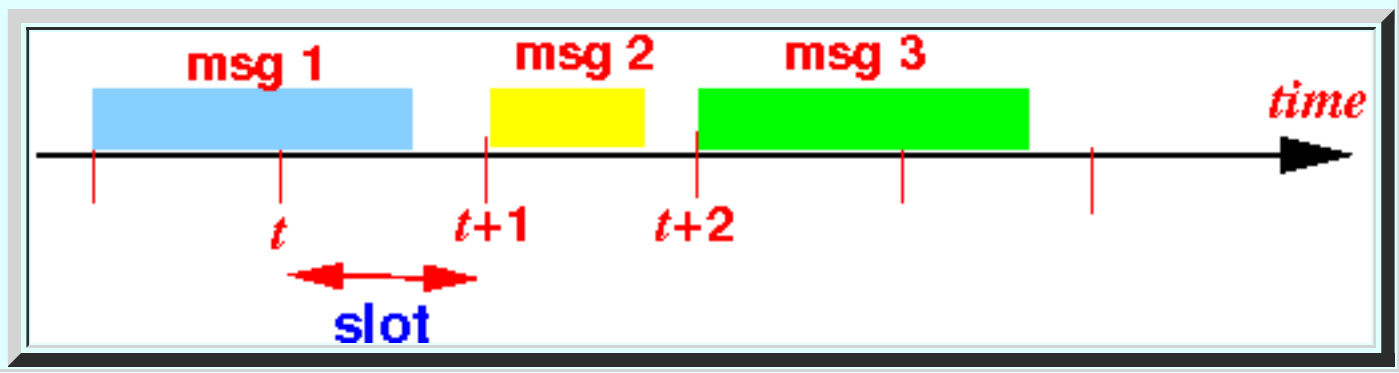
    - A **transmission** must ***begin*** at the **start** of a **(time) slot**:



    - **However:**

      - A **transmission** can last ***longer*** (or ***shorter***) than ***one*** **time slot**

Example:



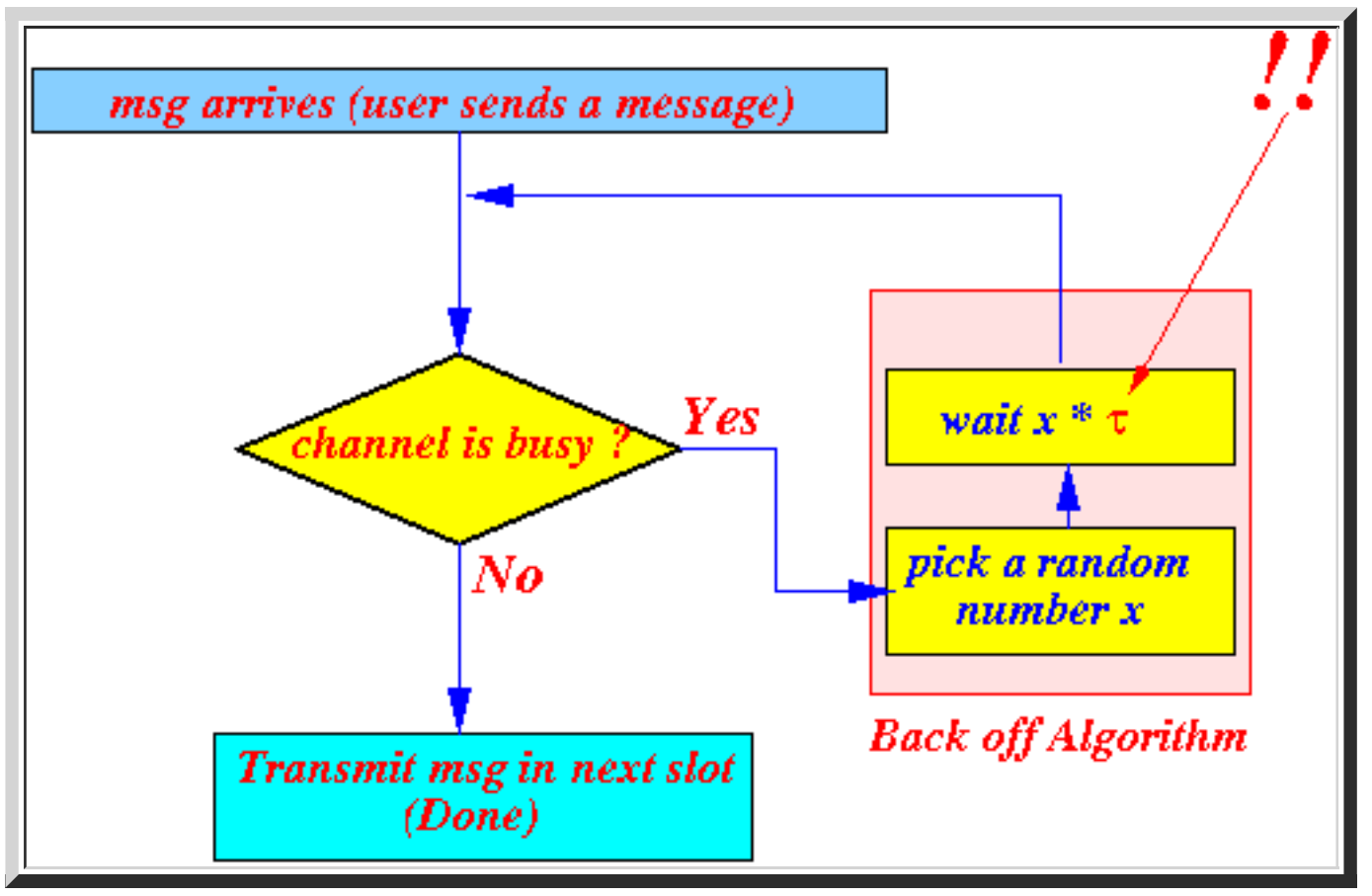
The *Non-persistent* CSMA protocol

- **Non-persistent CSMA:**
  - The ***non-persistent*** **CSMA:**

    - **go away** if the **channel** is **busy**

      (and come back later.....)

  - **Flow chart:**



Explanation:

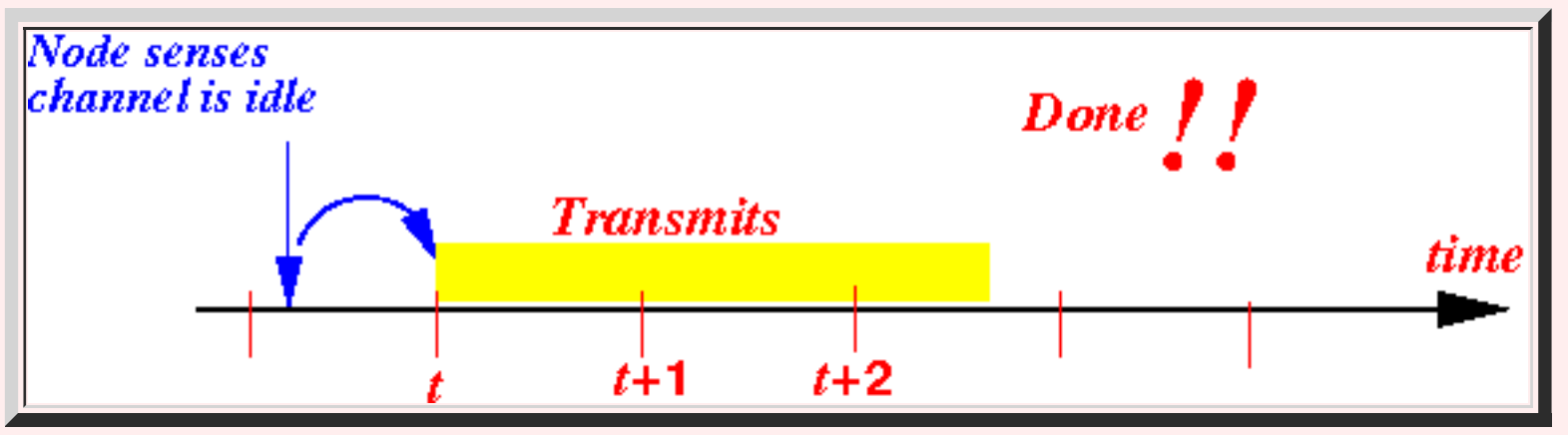
- If the node **senses** that the **carrier is *idle***:

  - the **node transmits** the **message** in the **next slot**

  After **transmitting** the **message**, the **CSMA protocol terminates** !

  (**Don't wait** for **ACK** !)

  Graphically:



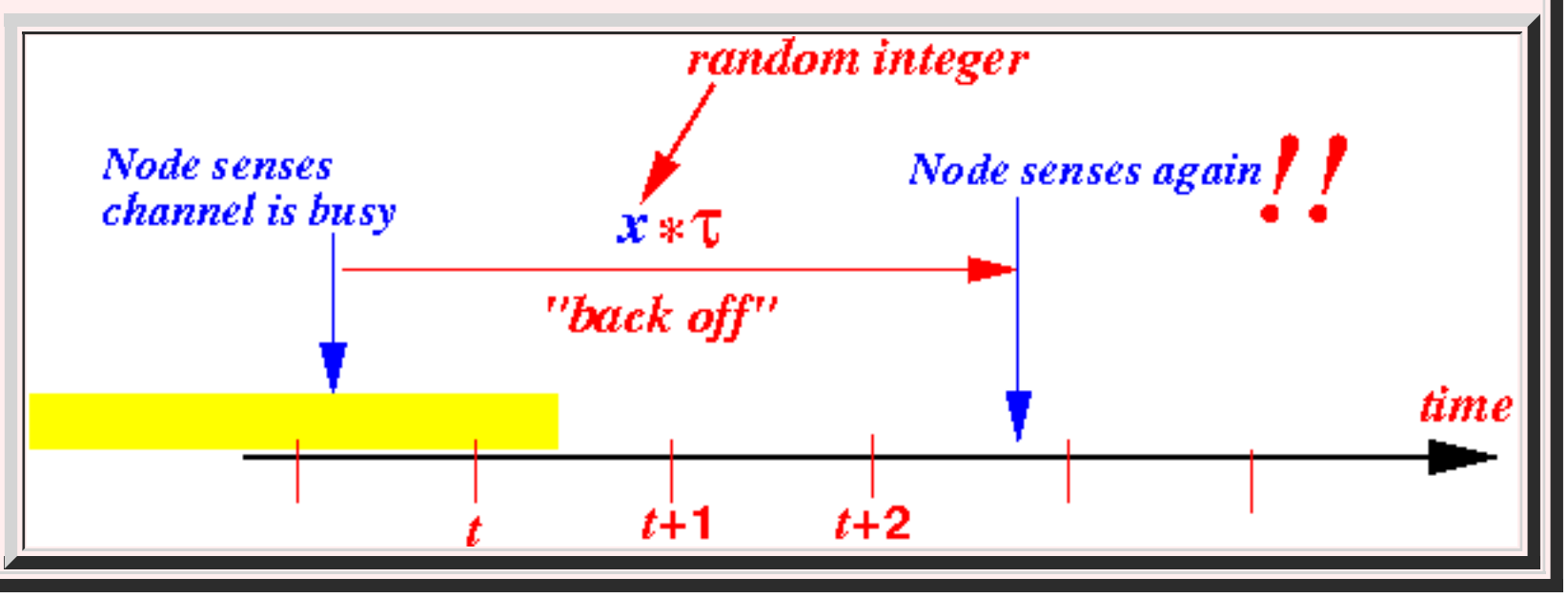
- If the node **senses** that the **carrier is *busy***, it performs the following **backoff procedure**:

  - The node **picks** a **random (integer) number *x***
  - The **node** then **waits**:

x × τ sec

- and **repeat** the **protocol** from the **start**....

Graphically:



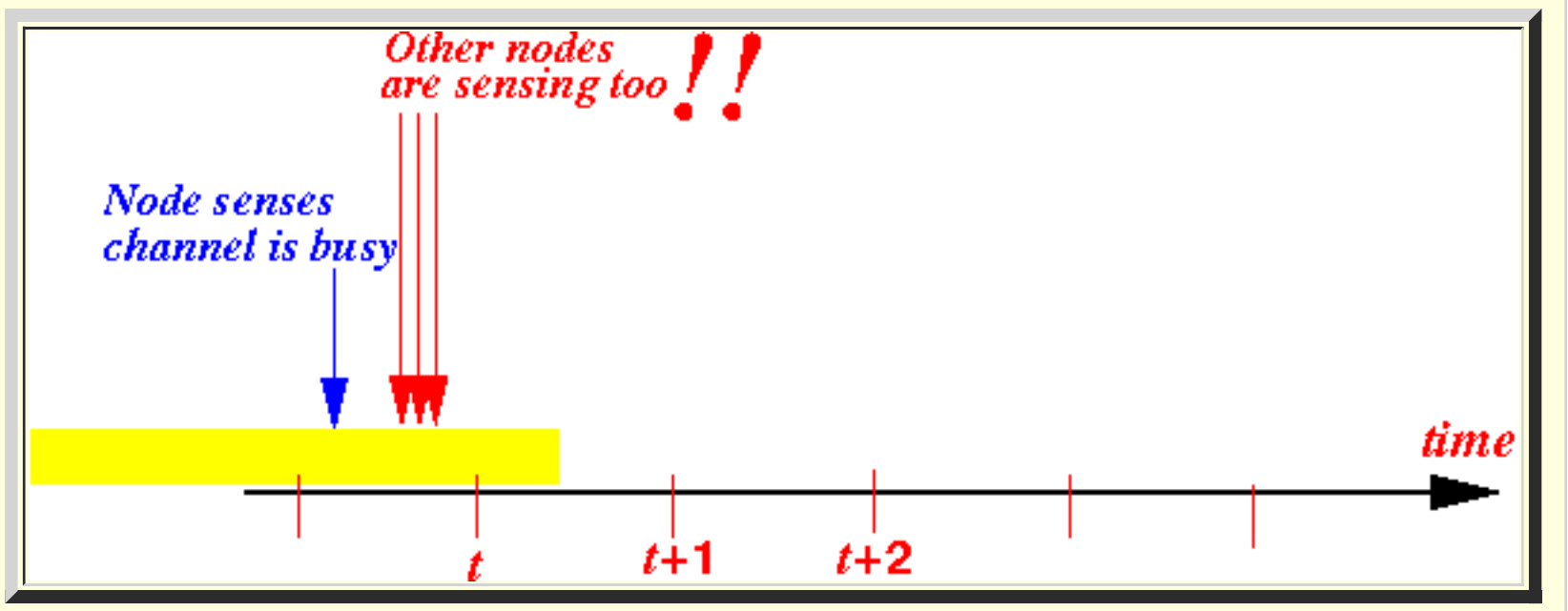
- **Rationale of the *Non-persistent* CSMA protocol**
  - **Rationele** of the ***non-persistent*** **CSMA:**

- The **non-persistent CSMA** is a **"pessimistic" protocol**:

- When a **node senses** that the **transmission channel** is **busy**, a ***pessimist*** will **think**:

  - There are ***other*** **nodes** that are ***also*** **sensing** that the **channel** is **busy**

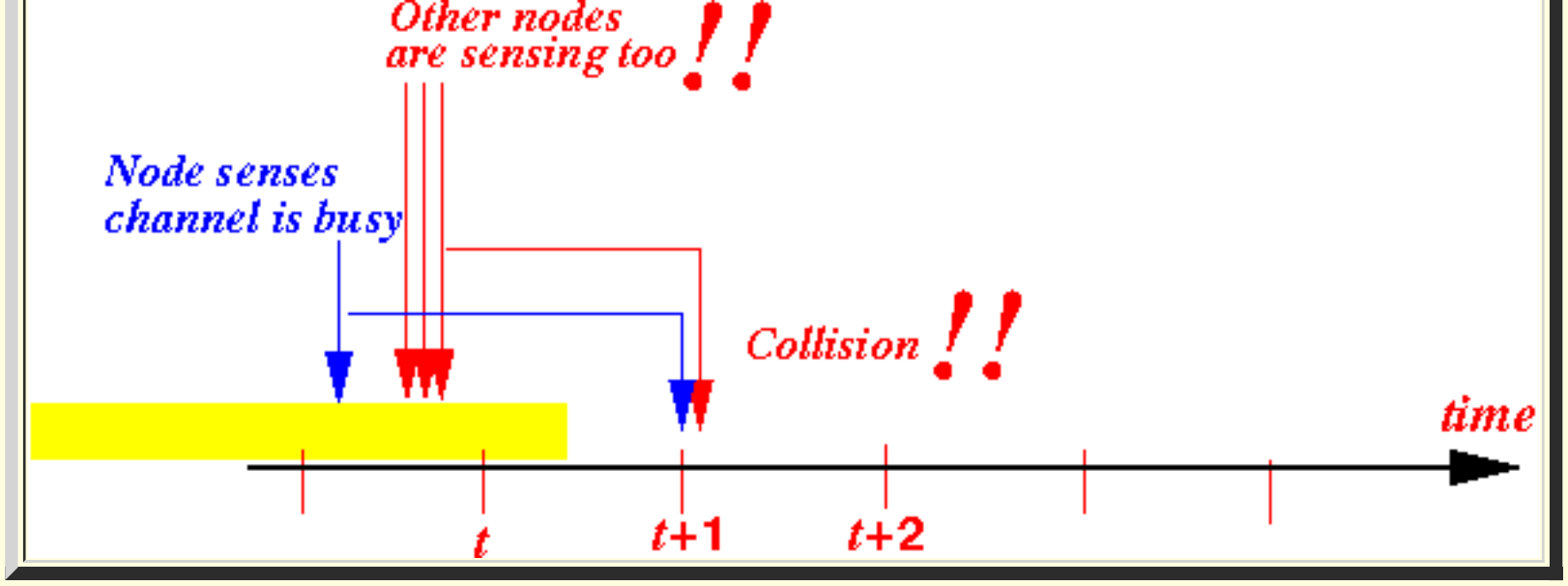
Graphically:



- So ***if*** the **node** **transmits** after the **current transmission** ***ends***, **then**:

  - There will **definitely** be a ***collision*** !!!

Graphically:



- **Therefore:**

  - The **node** will ***retry*** after some ***random*** **amount of (wait) time** to ***avoid*** the (hypothetical/potential) **collision**

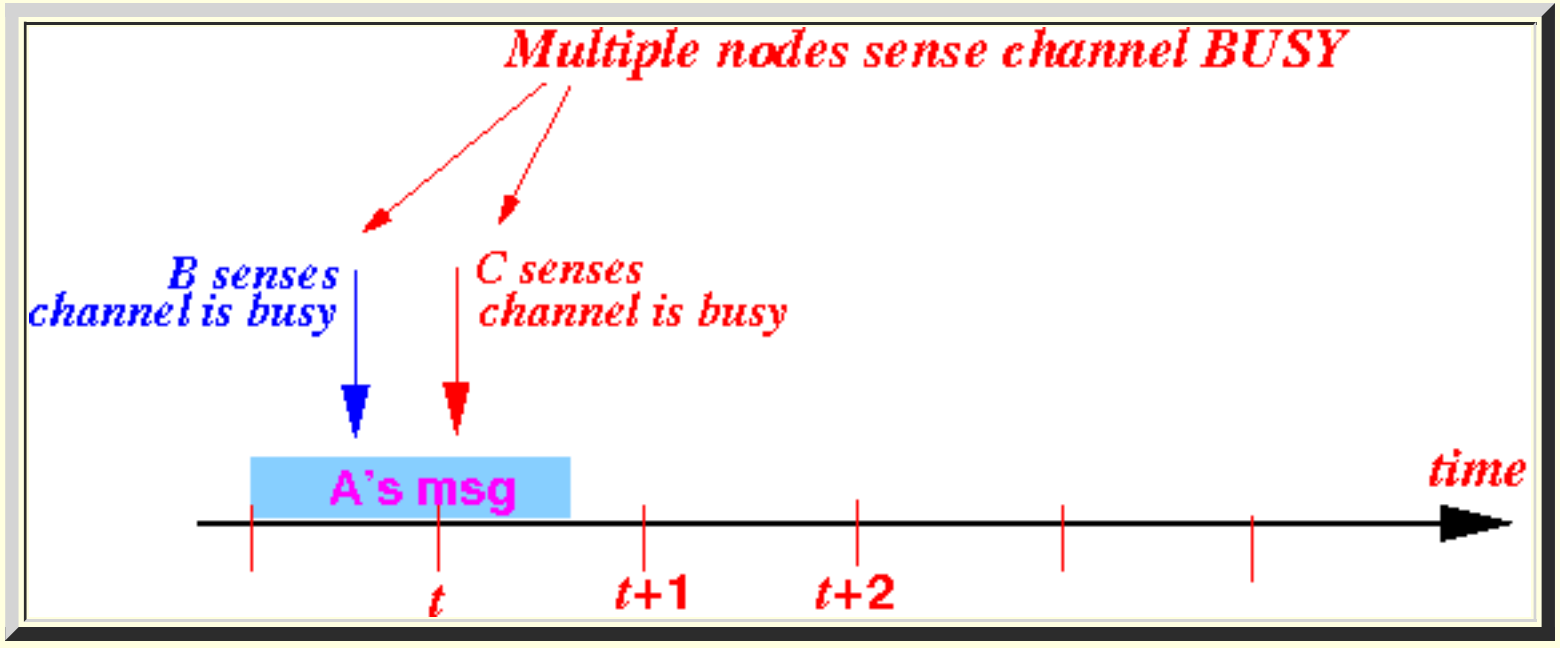
- **Strength and weakness of Non-persistent CSMA**

  - **Strength of *non-persistent CSMA***

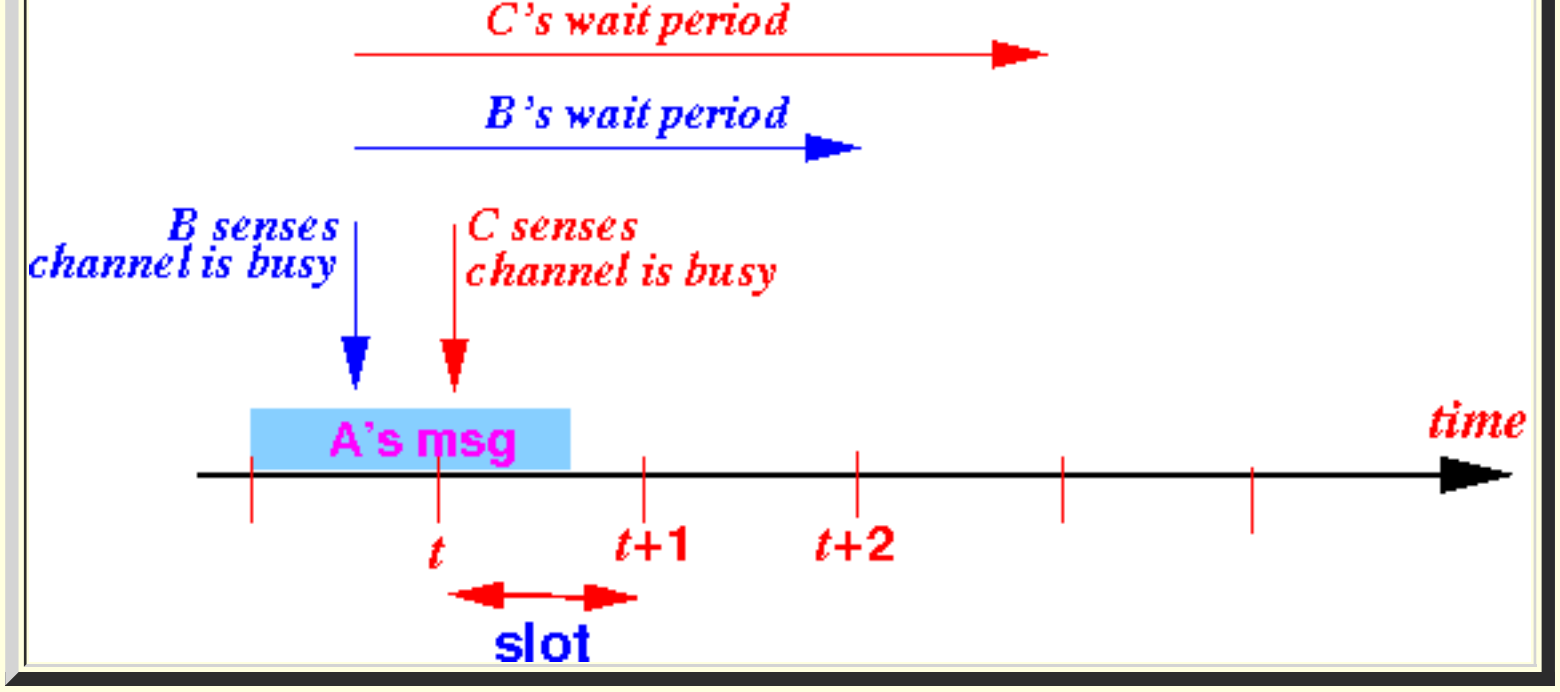
    - Achieves ***automatic*** **transmission rescheduling** in ***heavily loaded*** **systems**

      Example:

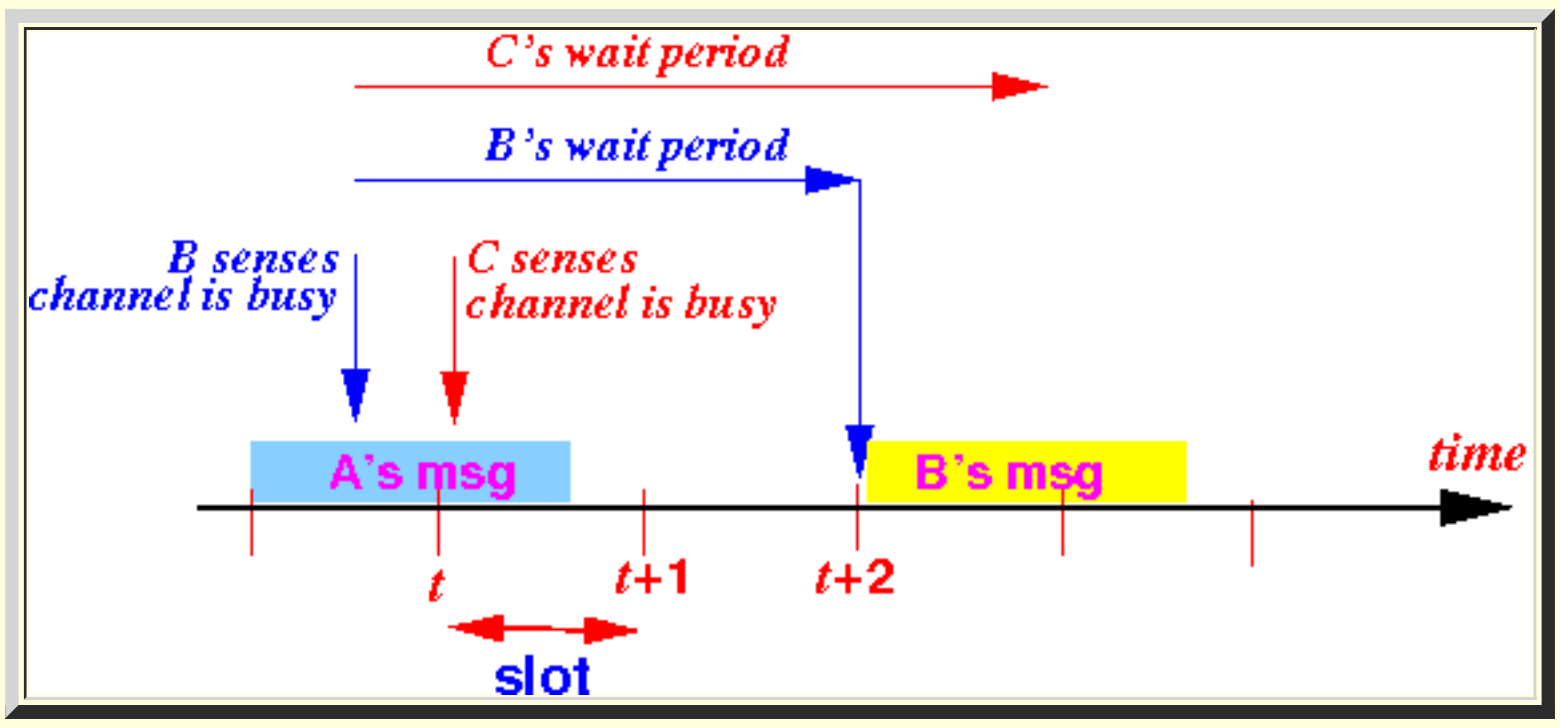
      - When ***many*** **nodes** that have messages to **transmit** (= ***heavily*** **loaded system**), it is **likely** that ***multiple*** **nodes** are **waiting**:



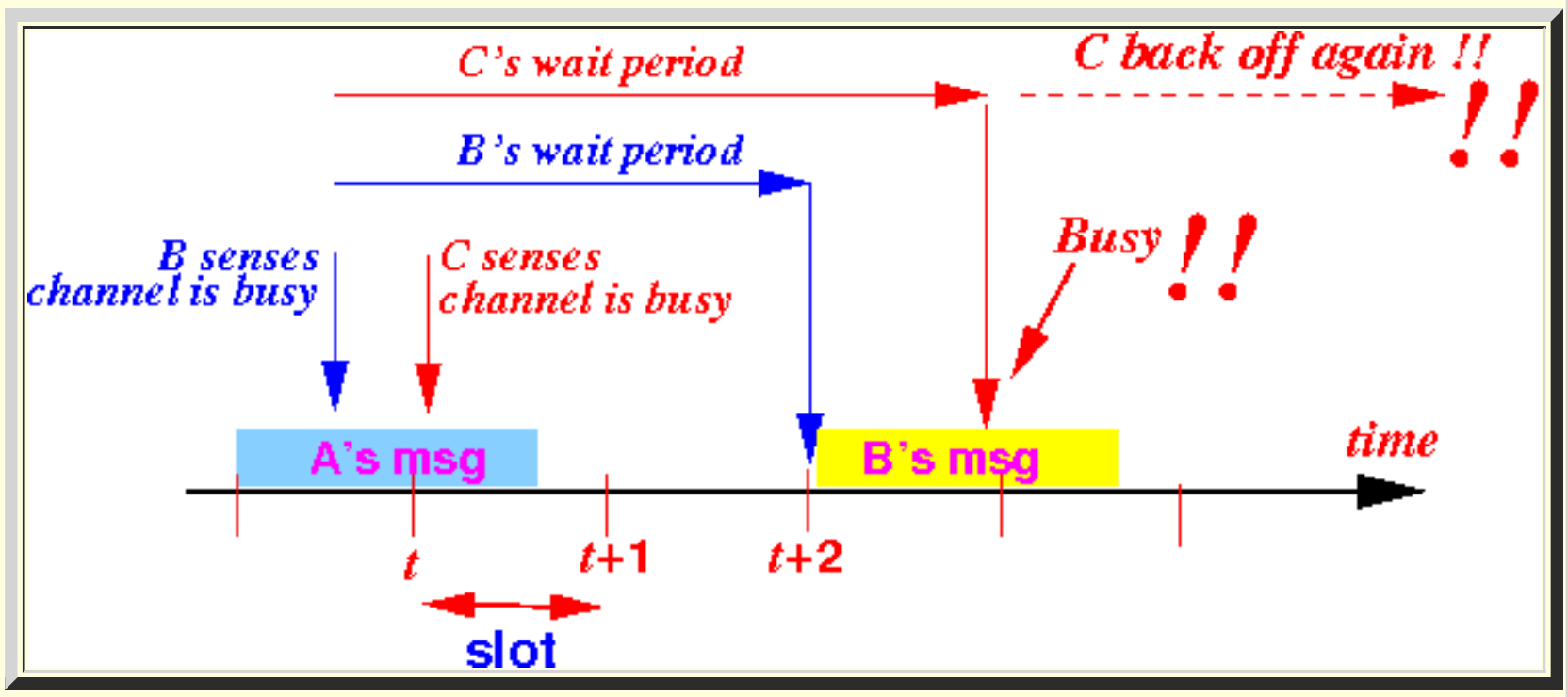
      - ***Each*** **node** will **then** pick a (different) ***random*** **number**:



- The **node** that **picks** the ***smaller* random number (= B)** will **transmits *first***:



- The **node** that **picks** the ***larger* random number (= C)** will **back off *again***:

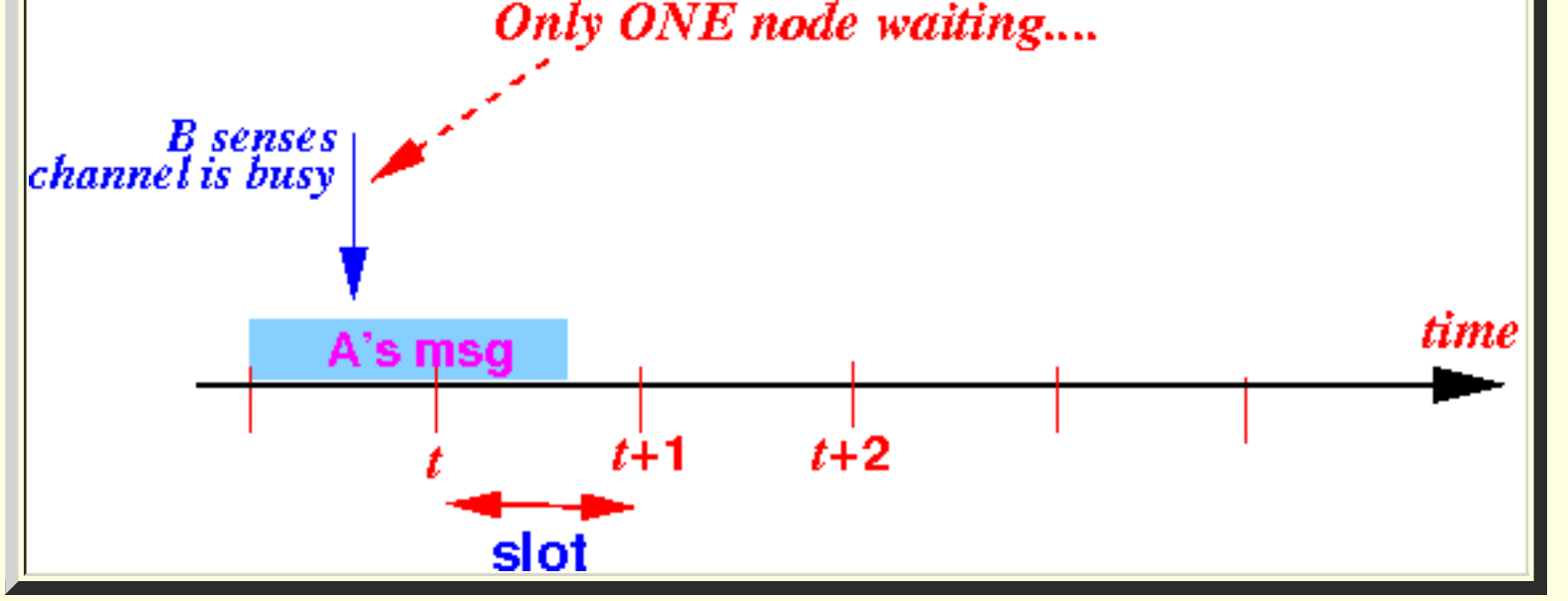


- **Weakness of *non-persistent CSMA***

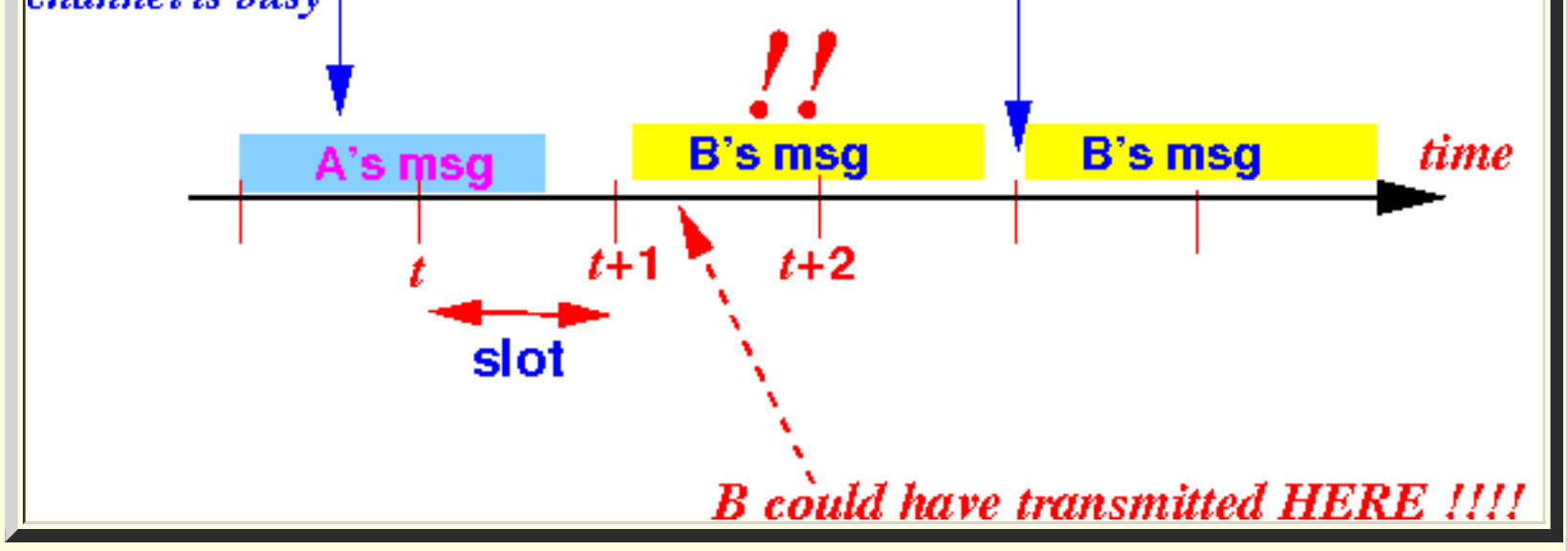
  - Possible ***channel* wastage** in a ***lightly* loaded system**

    Example:

    - In a ***light* loaded system**, there are **not many** nodes ***waiting*** to **transmit**:



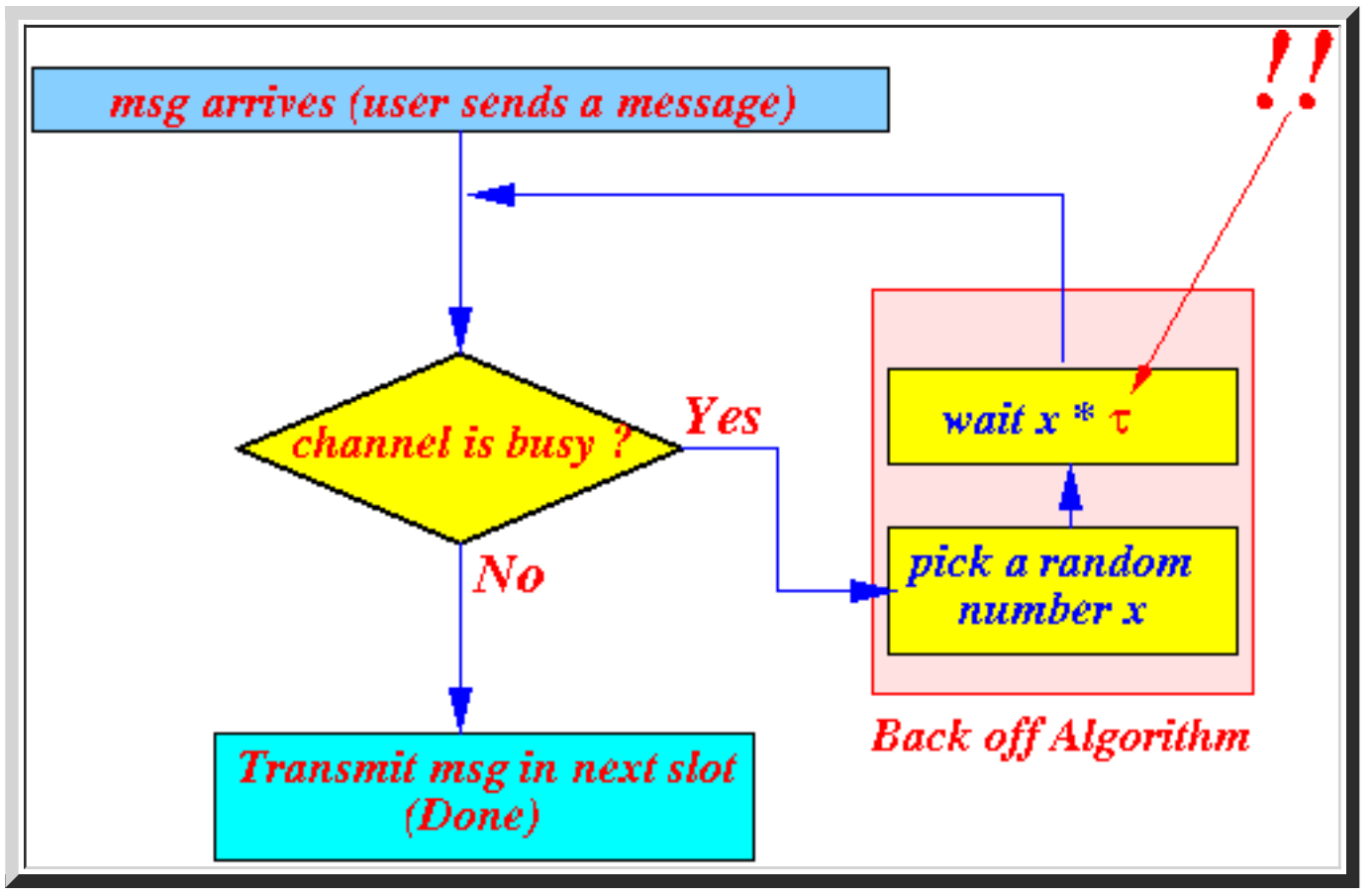
- **Suppose** the **node** pick a ***large*** **random number**:



- ***Wastage*** of **transmission time**:

How to "better" avoid collision

- **The goal of the back off algorithm**
  - **Consider** the **non-persistent** **CSMA** protocol:

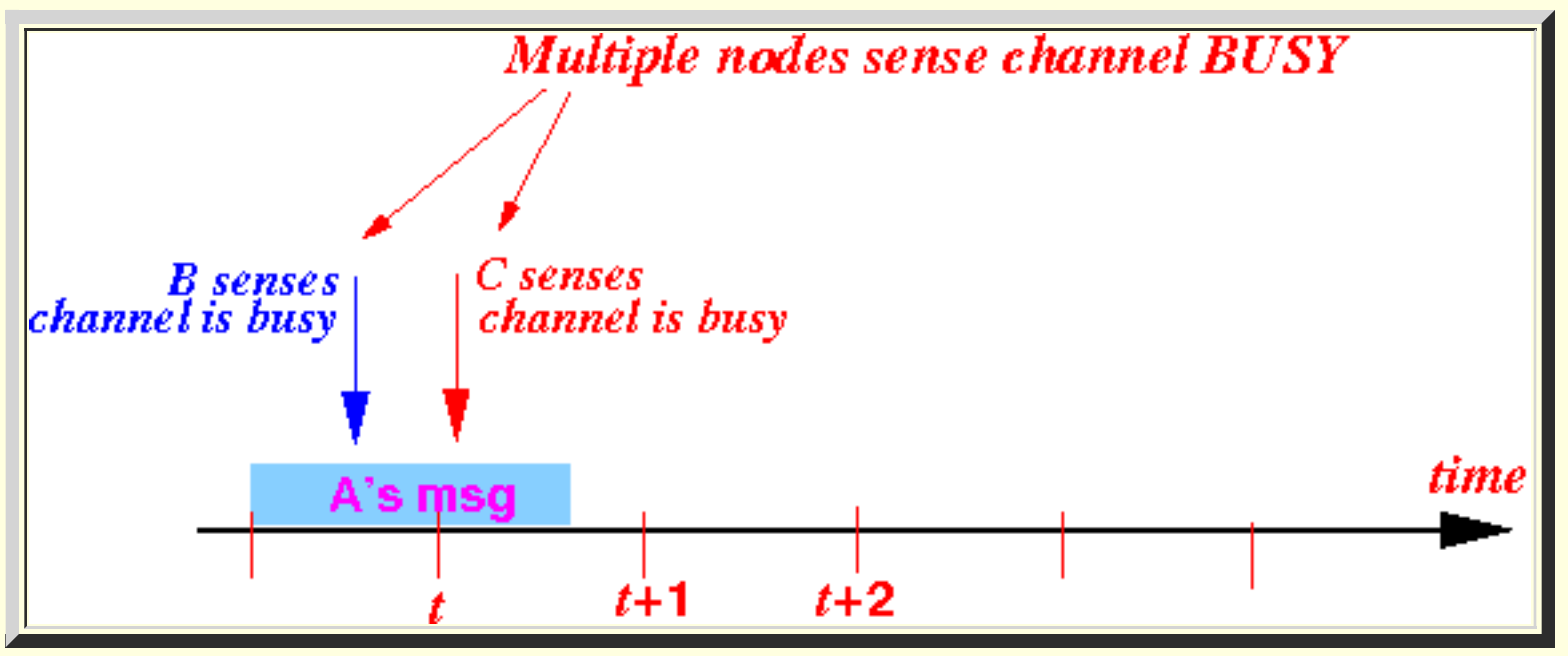


  - The **goal** of the **back off algorithm** is:

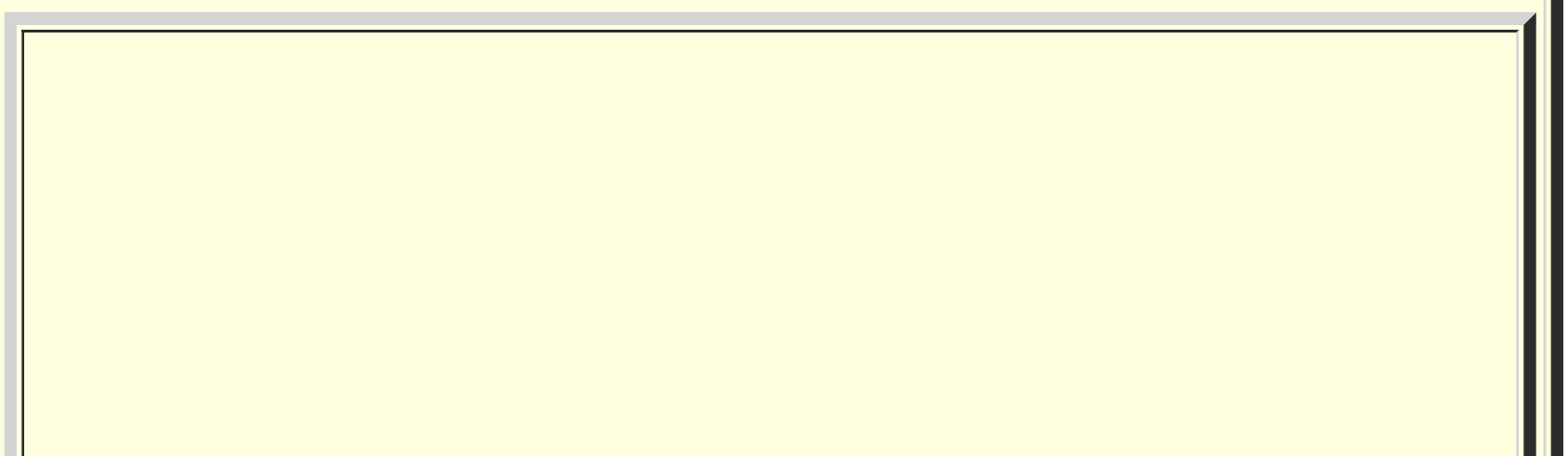
    - **Minimize** the **probability** of **collision** between ***2*** **or** ***more*** **waiting nodes**

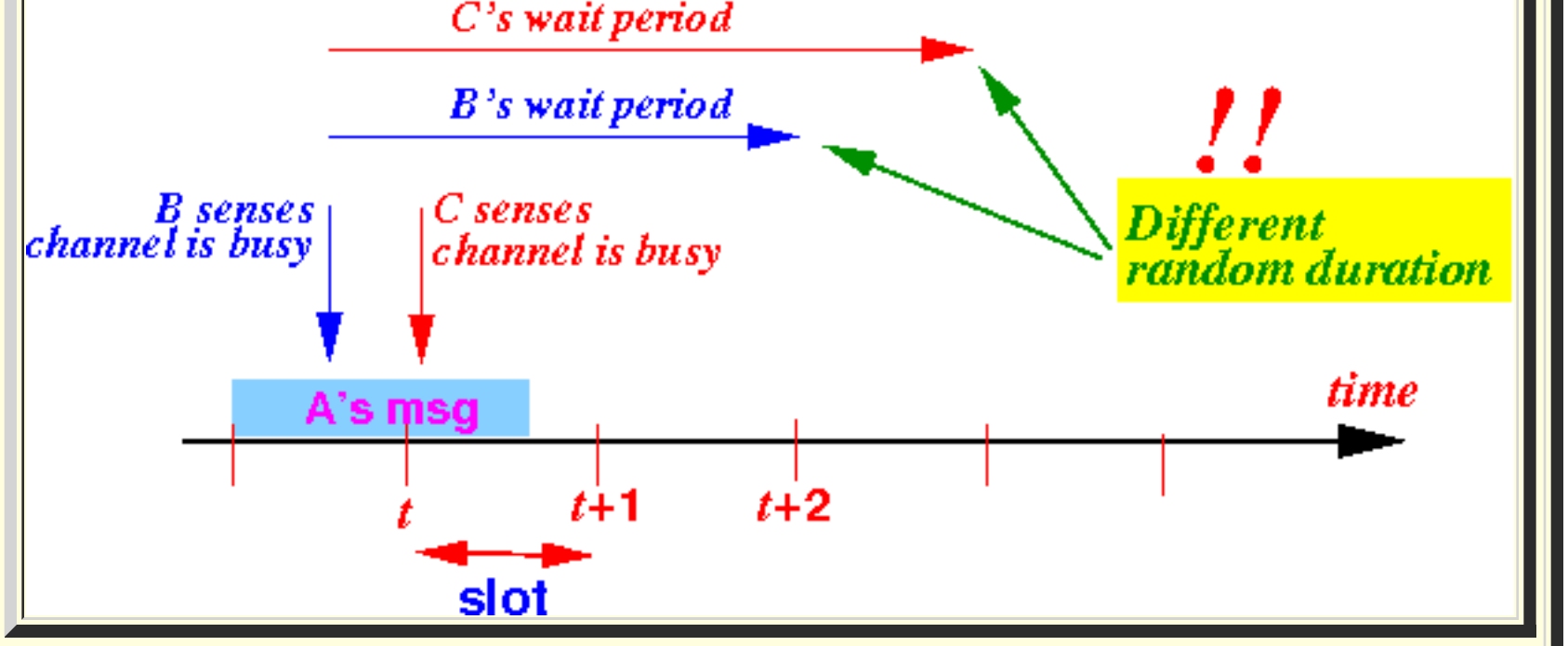
  - **Example:**

    - When **nodes** **B** and **C** ***both*** detect a ***busy*** **channel**:

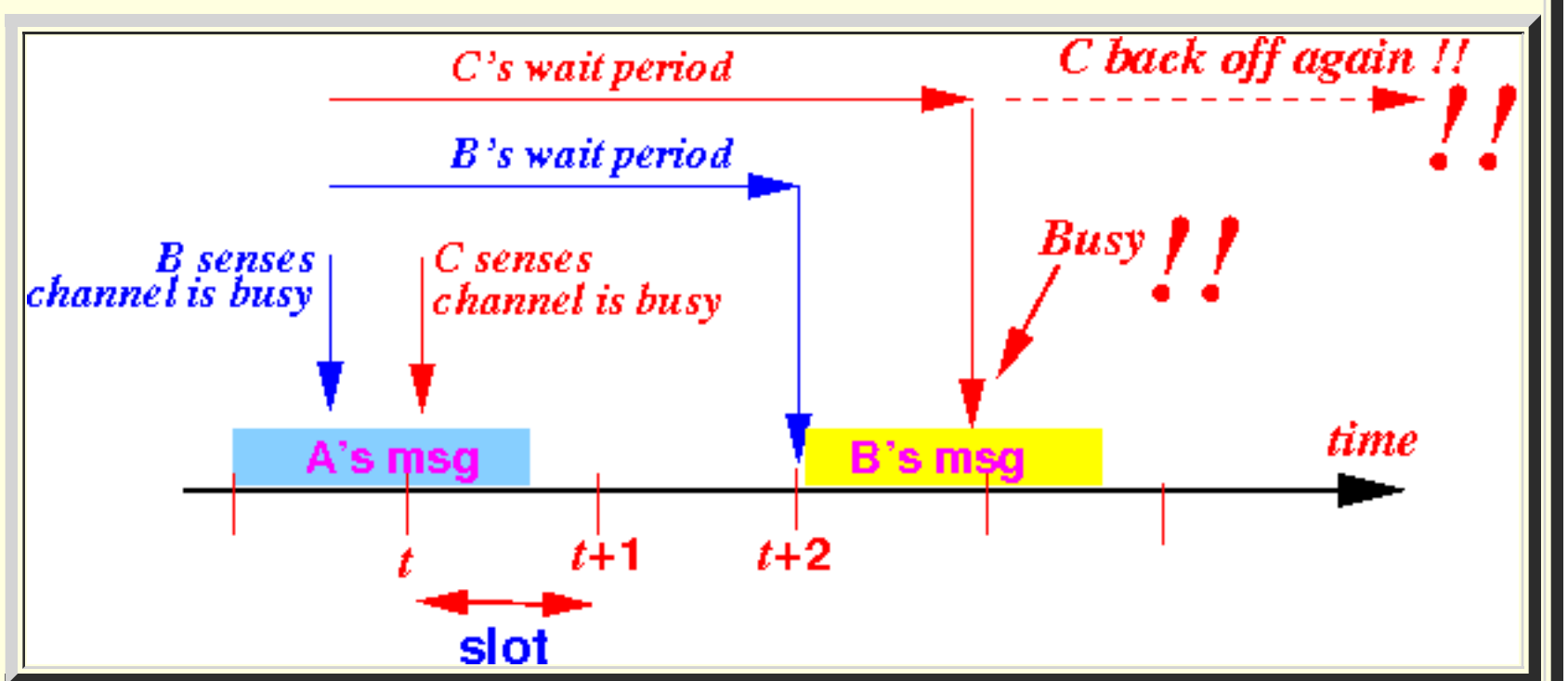


and **B** and **C** pick ***different*** **random numbers**:





then **C** can **sense B's transmission**:

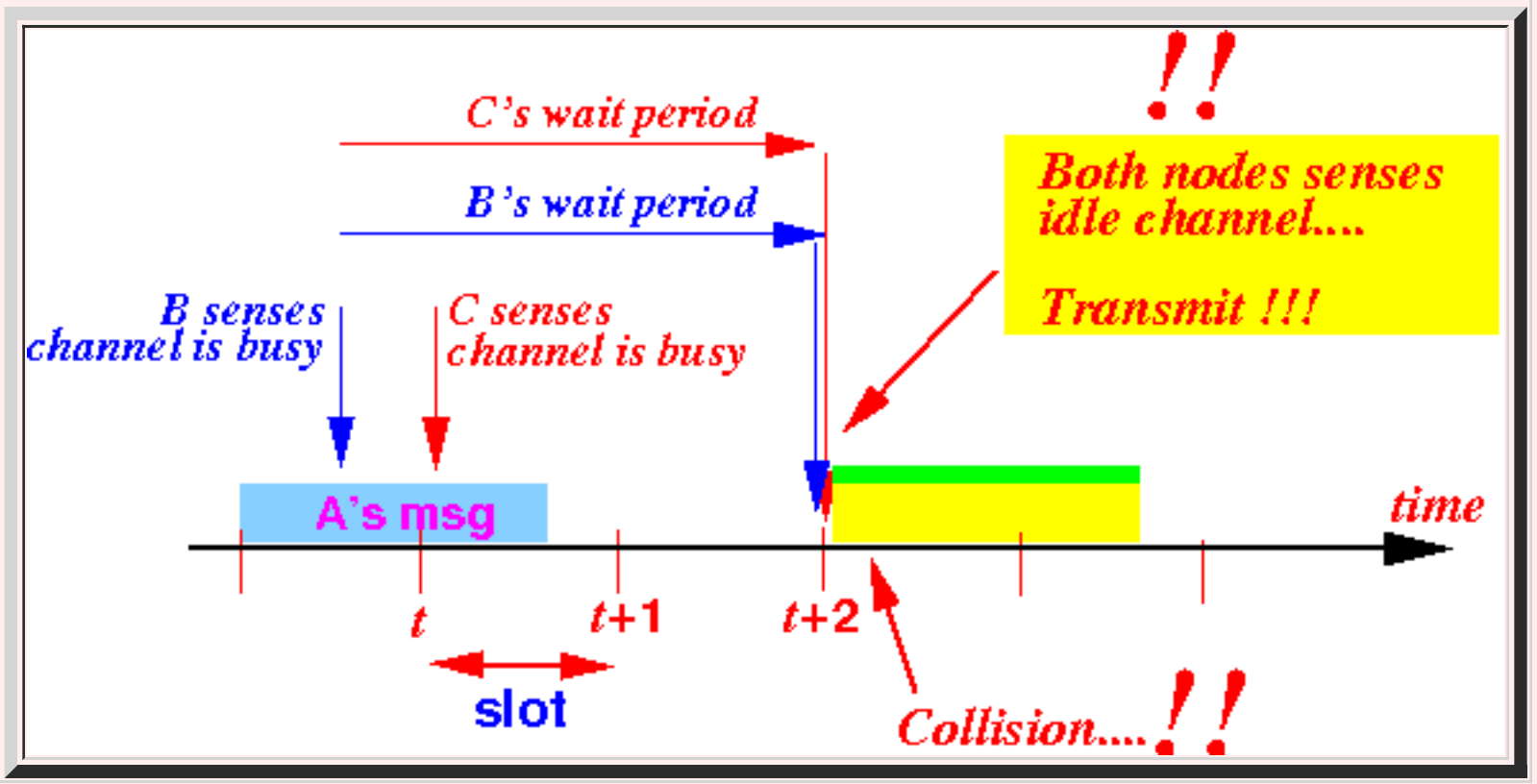


- **Comment:**

  - If ***both*** **nodes** pick the ***same*** **random number (duration)**, then:

    - Their **transmissions** will ***collide***

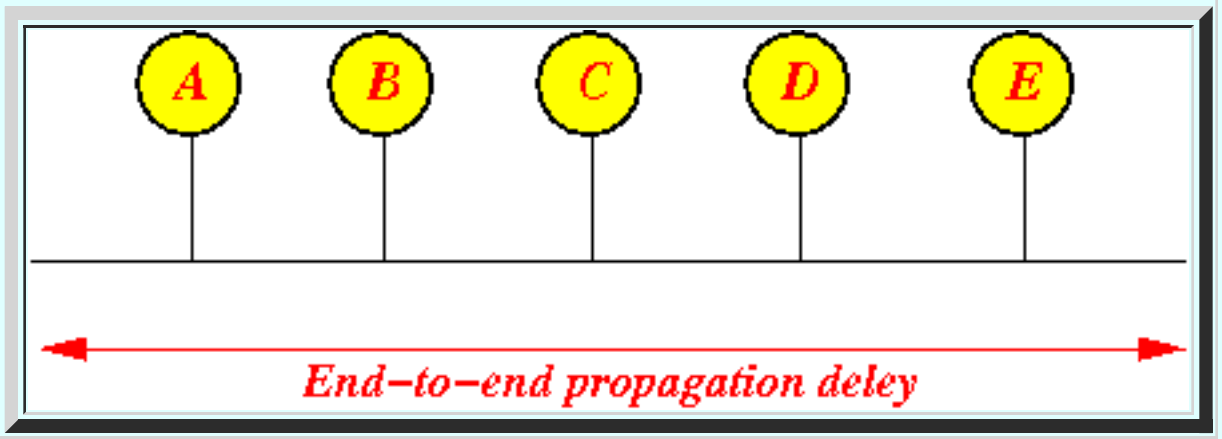
    Because:



- **The end-to-end propagation delay**
  - **Definition:**

    - **End-to-end propagation delay** = the **time** it takes for **signals** to traverse:
      - ***From*** **one end** of the **network**
      - ***to*** the ***other*** **end** of the **network**

    Graphically:



  - **Notation:**

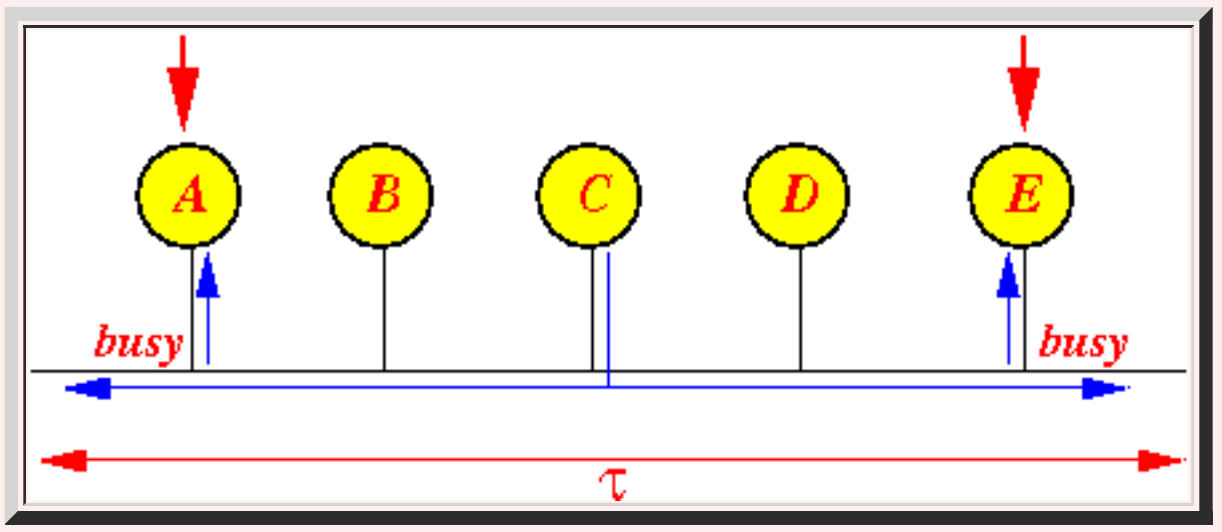
    - **τ** = the **end-to-end propagation delay**

- **Why do nodes wait (*x* × τ) delay before sensing the channel**
  - **Fact:**

    - When **2 nodes** **delay** for ***different*** **(integer) random number × τ**, then:
      - The **node** that picks the ***larger*** **integer** is **guaranteed** to **heard** the **other node's transmission**

    Proof:

    - **Suppose** the **nodes** ***A*** and ***E*** hear a ***busy*** **channel** and **backoff**:

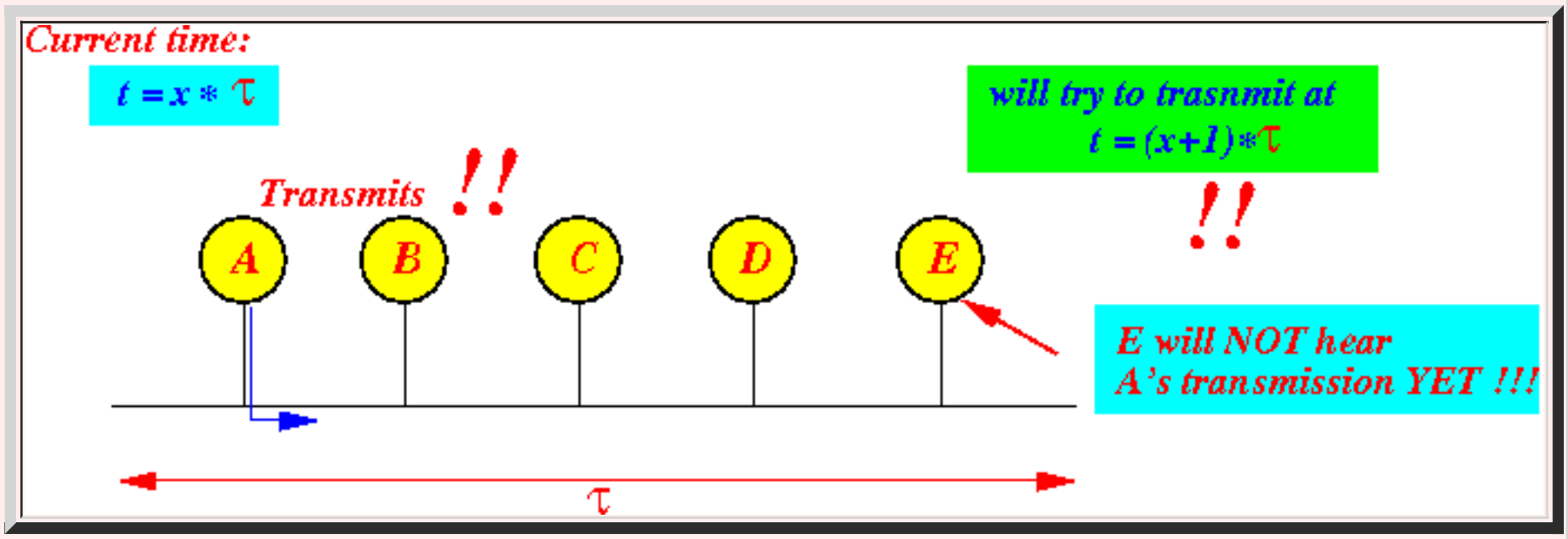


    - **Suppose**:
      - ***A*** and ***E*** pick 2 ***different*** **integers** that has the ***minimum*** **difference possible (= 1)**

    Say:

- ***A*** picks ***x***
- ***E*** picks ***x+1***

- Then **after** $x \times \tau$, the **node** ***A*** starts to **transmit**:



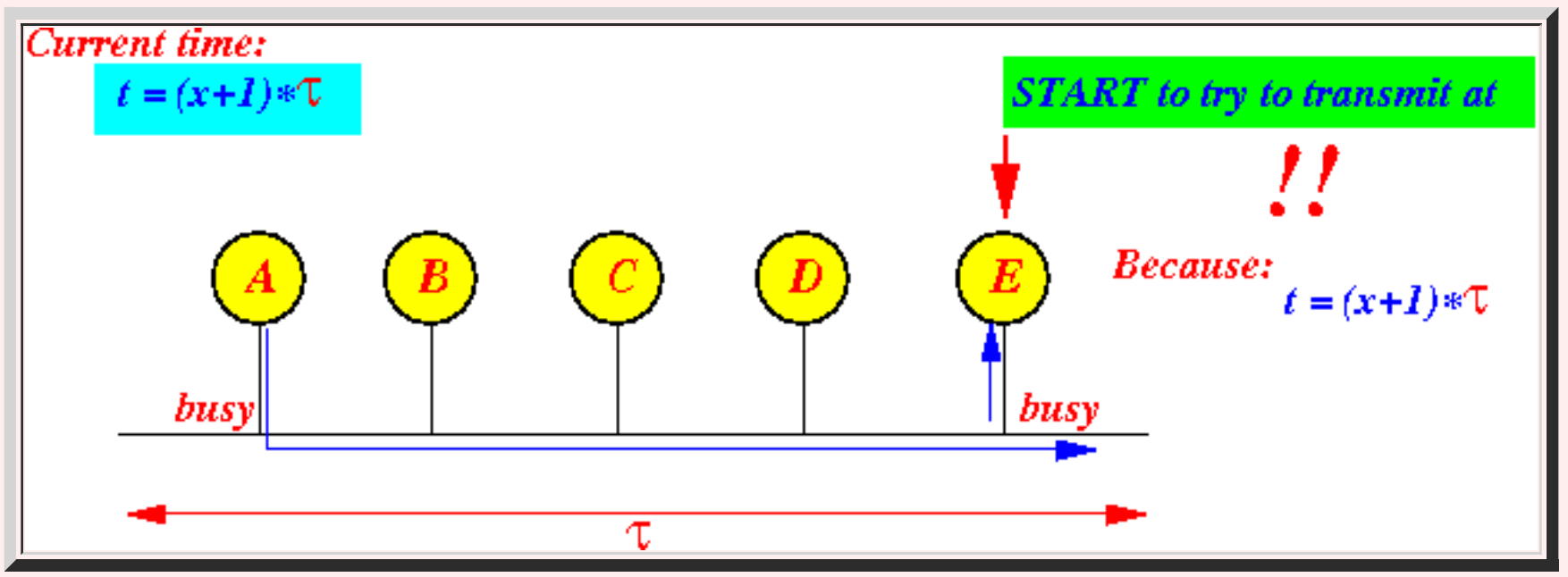
Notice that:

- ***E*** can ***not*** hear ***A*'s transmission yet** !!!

- ***E*** will **start** trying to **transmit** at time $(x+1) \times \tau$

- It take **τ sec** for the **signal** from ***A*** to **reach** node ***E***

  So at time: $(x+1) \times \tau$:



  The **transmission** of ***A*** arrives at ***E*** in the **nick of time** !!!

  When ***E*** **starts** its **transmission attempt**, ***E*** will **hears A's transmission *just in the nick of time*** !!!

- **Conclusion:**

  - We **must** multiply the ***random*** **integer** value by the ***end-to-end*** **propagation delay** to **ensure** that the **signal** has **enough time** to **reach *all* nodes** to **avoid collision** !!!

The *1-persistent* CSMA protocol

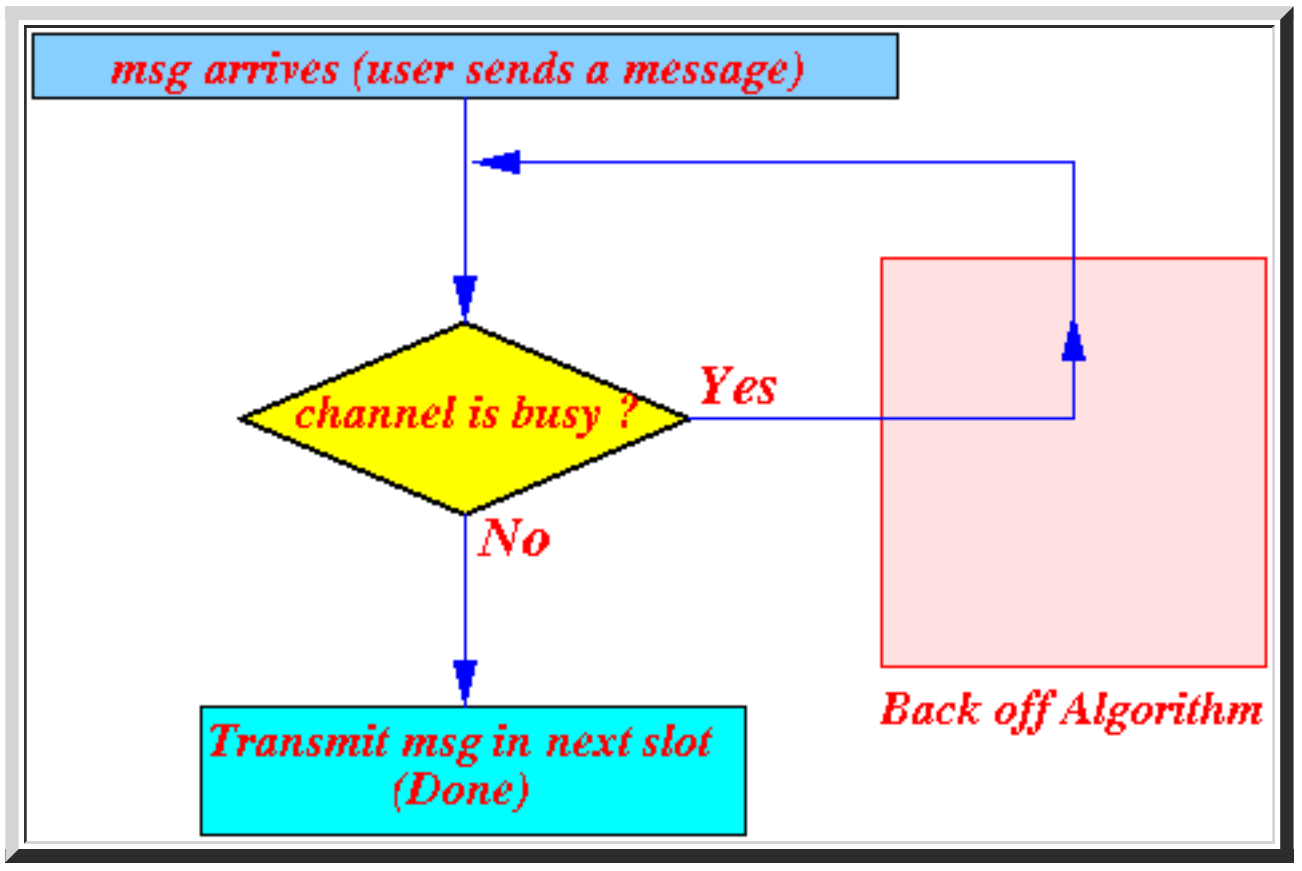
- **1-persistent CSMA:**
  - The **1-persistent CSMA** protocol:

    - If **channel is *busy***, then ***keep* sensing** until the **channel** becomes ***idle***

      - **As soon as** the **channel** becomes ***idles***, transmit the **message** in the ***next* slot**

    - (I call the **1-persistent** method: the **stalker** :))

  - **Flow chart:**



Explanation:

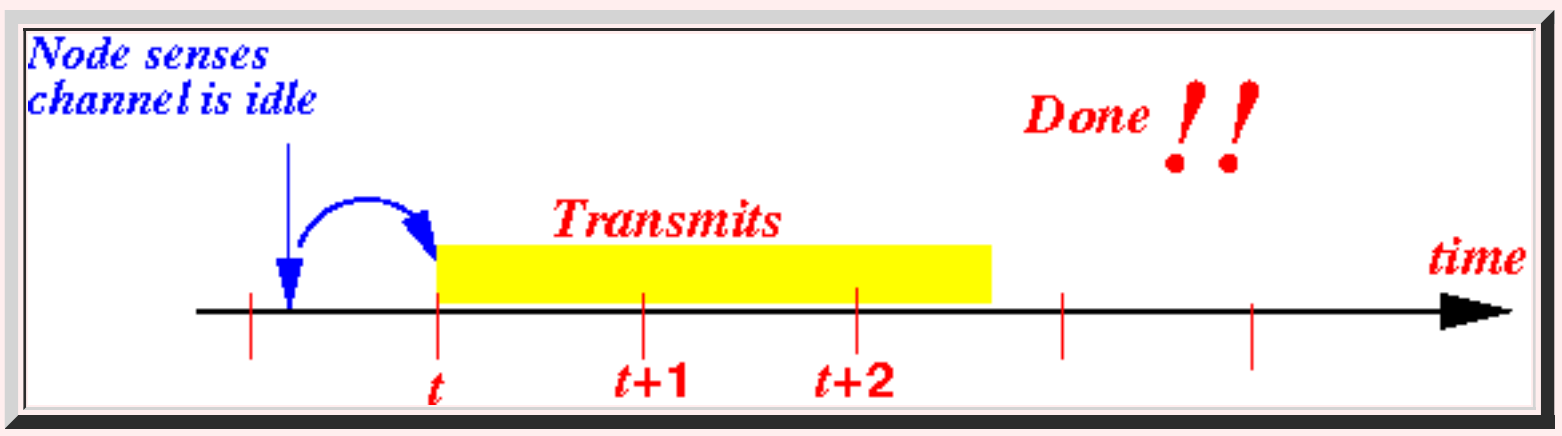
- If the node **senses** that the **carrier is *idle***:

  - the **node transmits** the **message** in the **next slot**

  After **transmitting** the **message**, the **CSMA protocol terminates** !

  (**Don't wait** for **ACK** !)

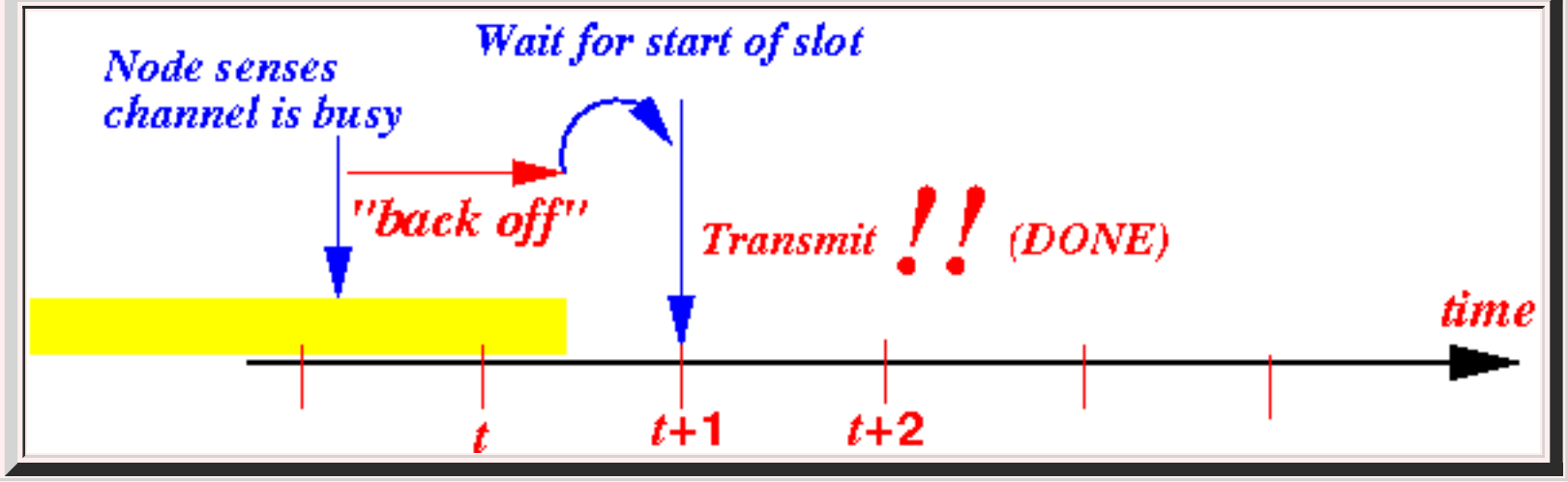
  Graphically:



- If the node **senses** that the **carrier is *busy***, **then**:

- The **node** **wait** until the **current transmission** ***ends***
- then **transmits** the **message** in the **next slot**

Graphically:



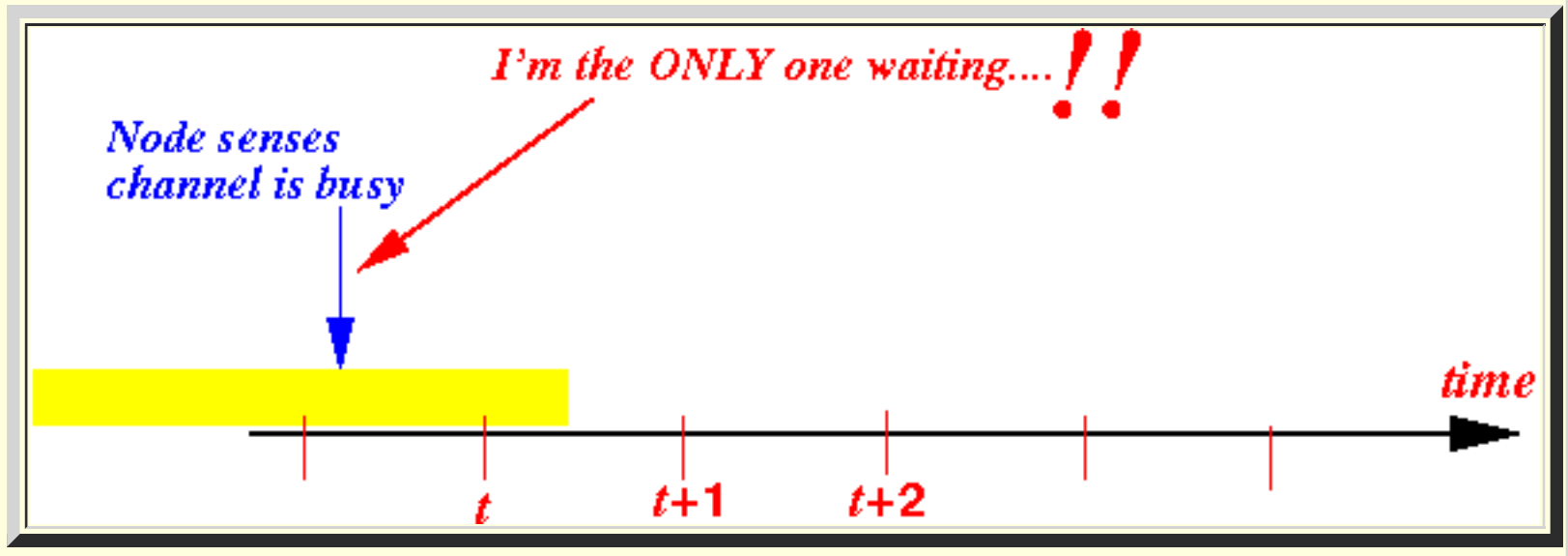
- **Rationele** of the **1-persistent CSMA:**

- **1-persistent CSMA** is an "***optimistic***" **protocol**

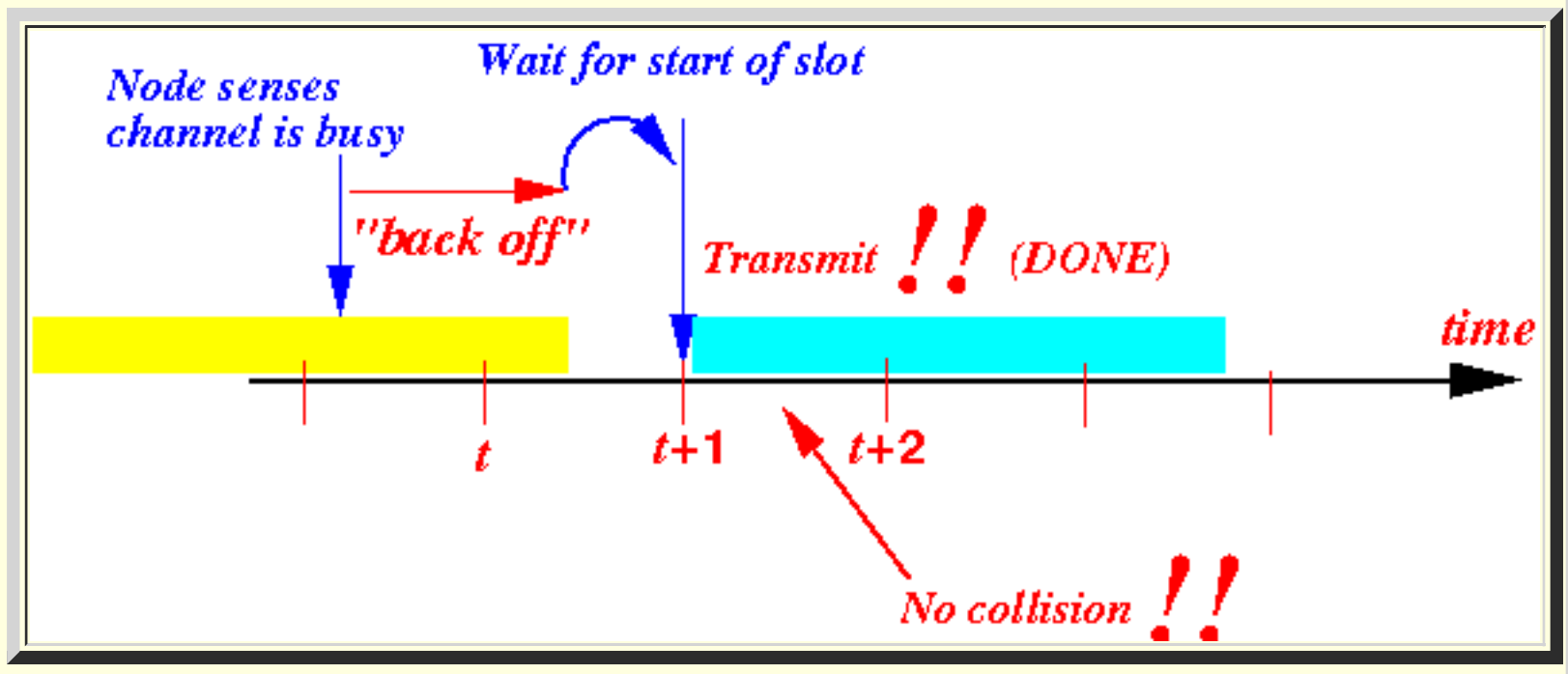
  - "I senses the channel is **busy**, and I **believe** that the system is ***lightly*** **loaded**

    - **I.e.:** I **think** that I'm the ***only*** **one** out there that is **waiting to transmit**....

    Graphically:

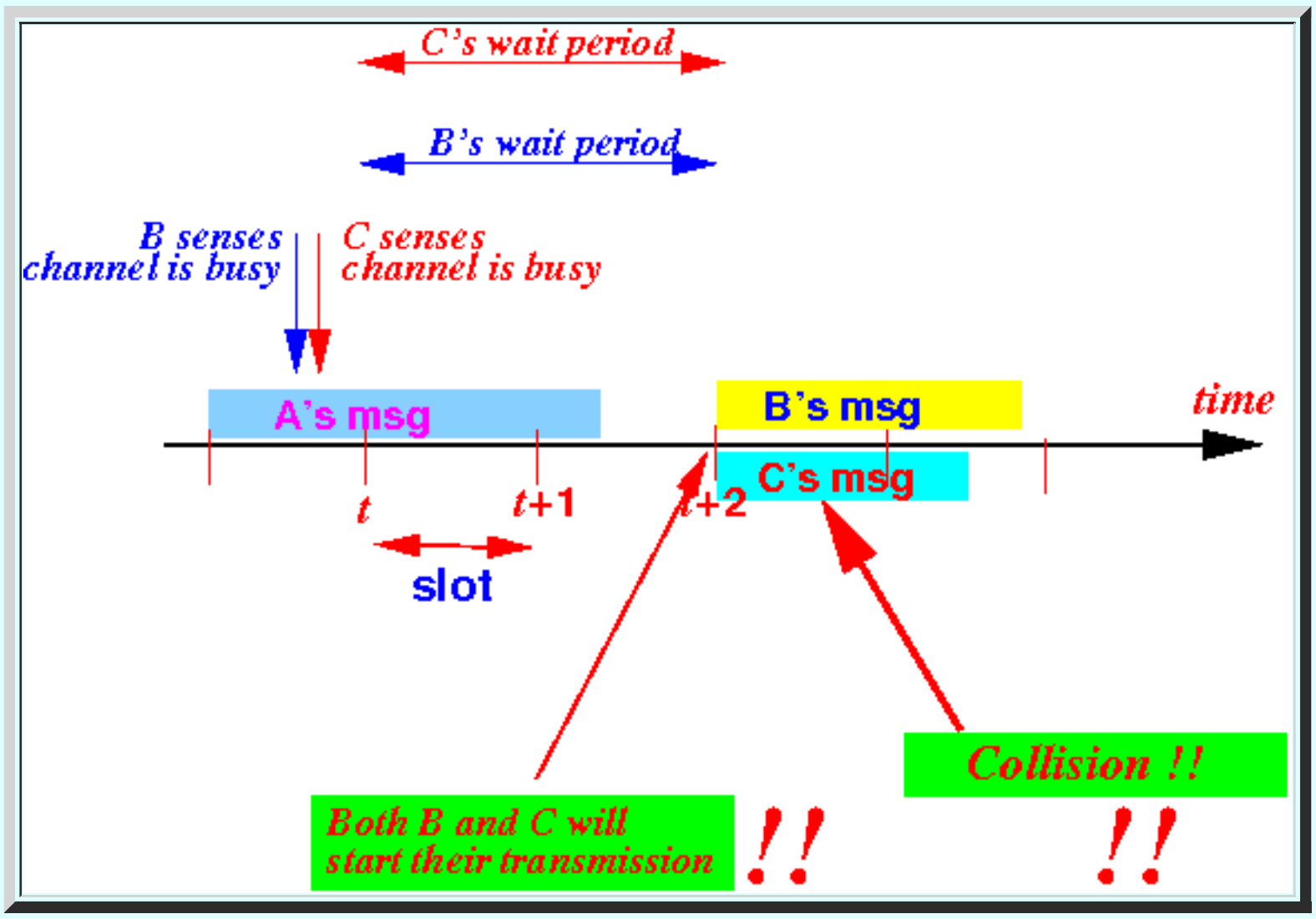


  - So there will **not** be a **collision** after the ***current*** **transmission** is **done**:



- **Note:**

  - ***If*** there are **2 or more nodes waiting** for the ***current* transmission** to **end**, the **1-persistent CSMA** protocol will ***always*** result in a **collision**:



- **Strength and weakness of *1-persistent* CSMA**

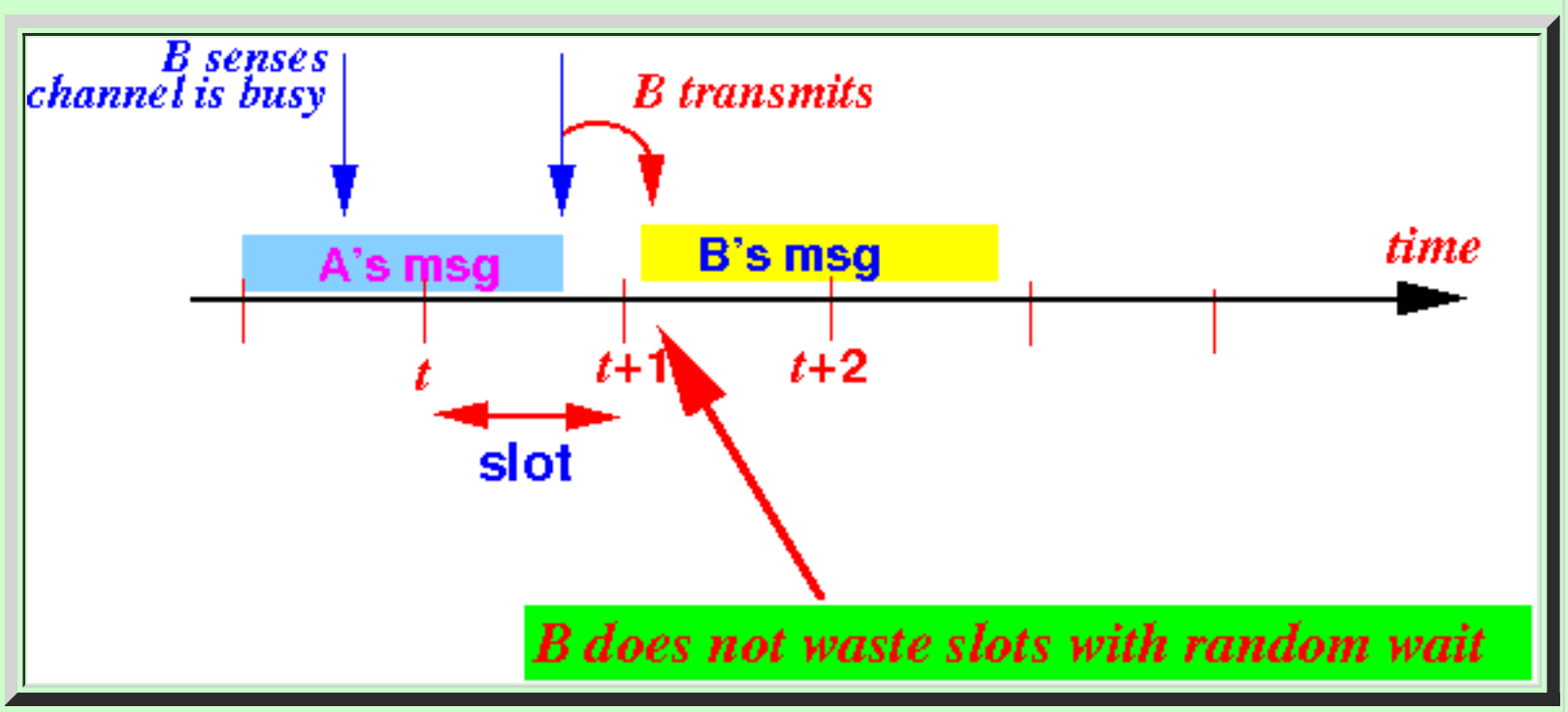
  - **Strength** of the ***1-persistent* CSMA protocol:**

    - **Performs very well** in **lightly loaded systems**

      - I.e., when there are **few nodes** that have messages to transmit

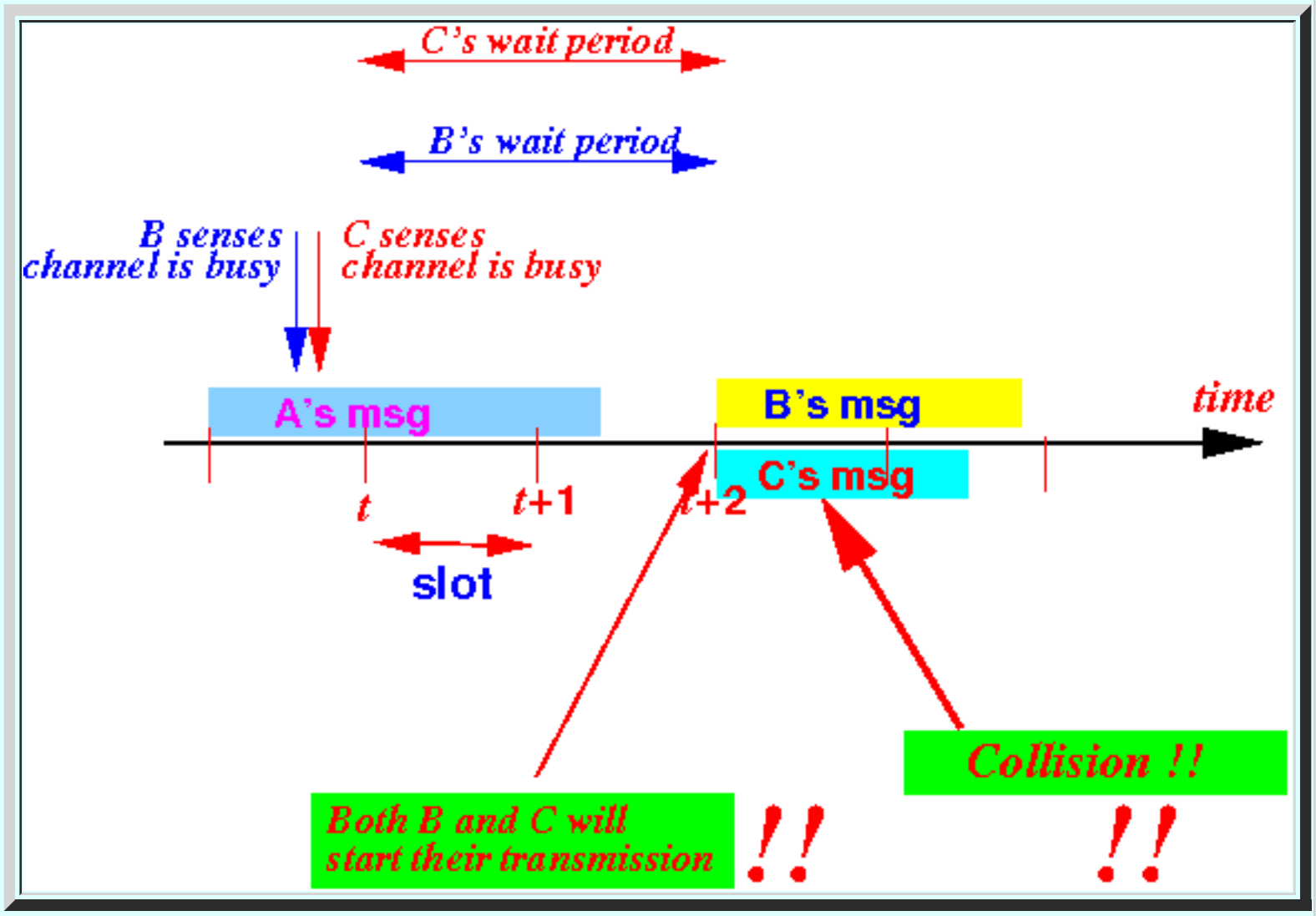
    ---

    - **1-persistent CSMA** will ***not* waste** any **slots** with a ***random* wait period**:



  - **Weakness** of the ***1-persistent* CSMA protocol:**

- **1-persistent CSMA** will ***guarantee*** **a collision** when **multiple nodes** are **waiting**



(**I.e.**, the **performance** will be **poor** under **heavy load**)

The *p-persistent* CSMA protocol

- ***p*-persistent CSMA:**
  - The **p-persistent CSMA** protocol:

    - The ***p*-persistent CSMA** protocol tries to ***fix*** the **"guaranteed" collision** problem in the ***p*-persistent CSMA** protocol

    - When a **node** can **transmit**, the **node** will:

      - **Flip** a **coin**

    - If **coin** comes up **head**, then

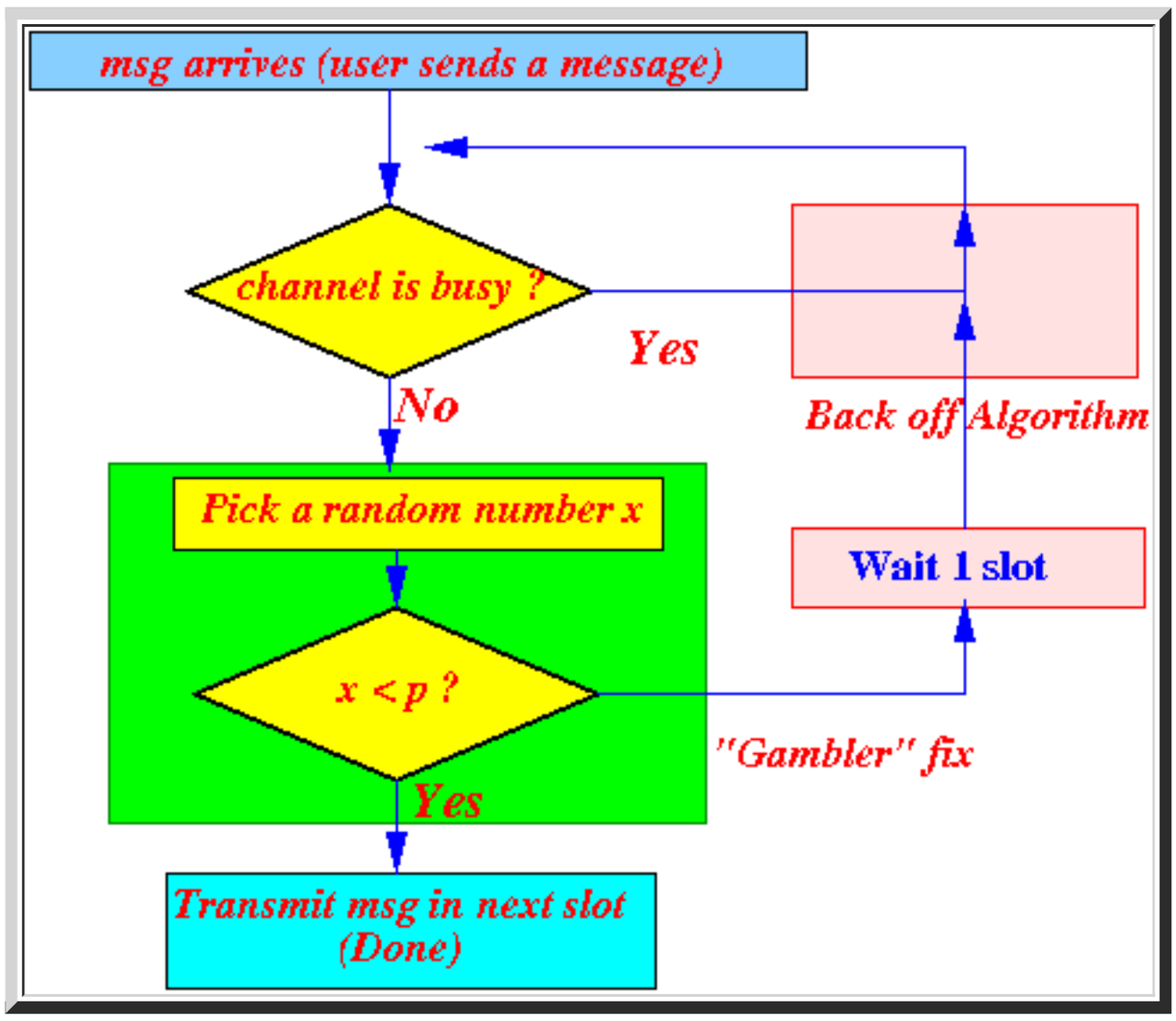
      - The **node** will **transmit**

      Otherwise

      - The **node** will **sense** again

    - (I call this **protocol** the ***gambler***...)

  - **Flow chart:**

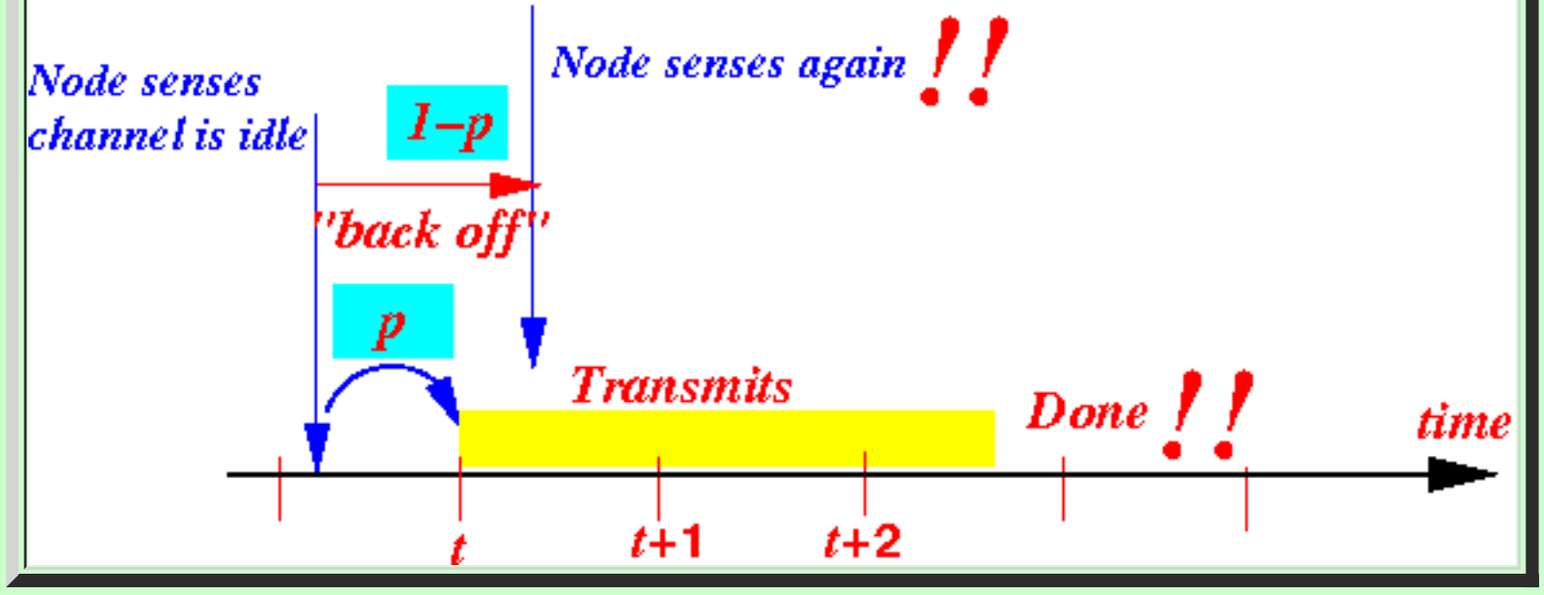


Explanation:

- If the **node** **senses** that the **carrier** is **idle**, then:

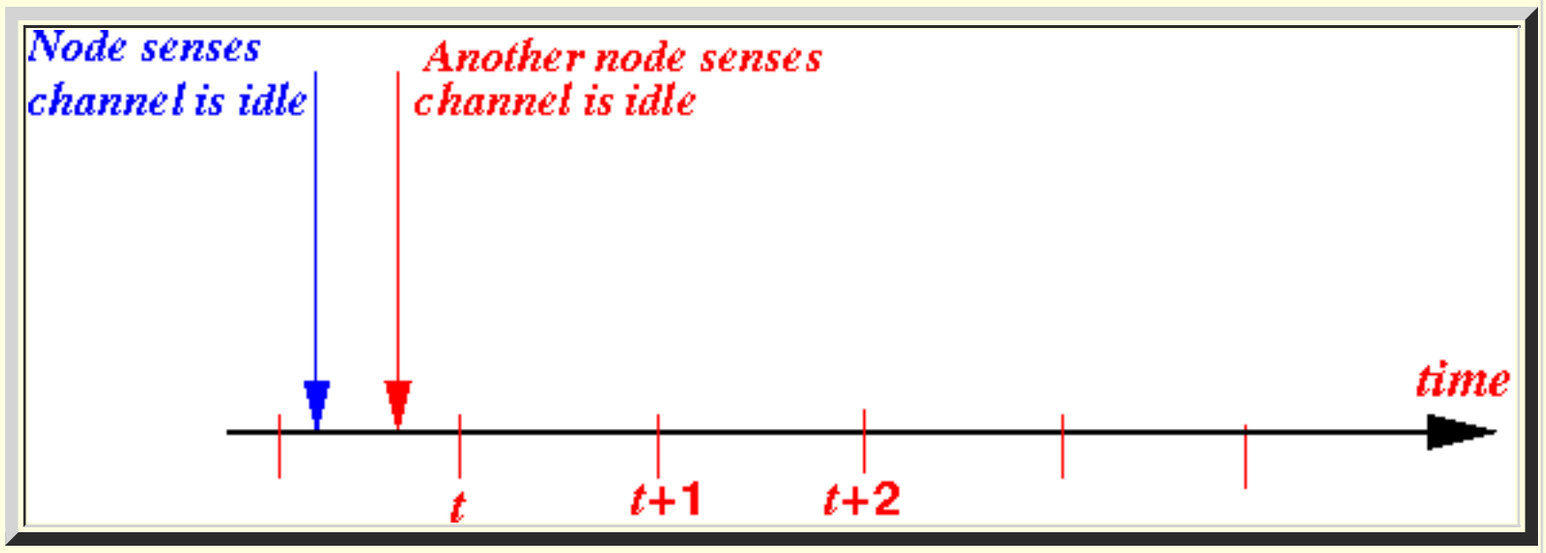
  - The **node** will **transmit** with **probability** ***p***

  Graphically:

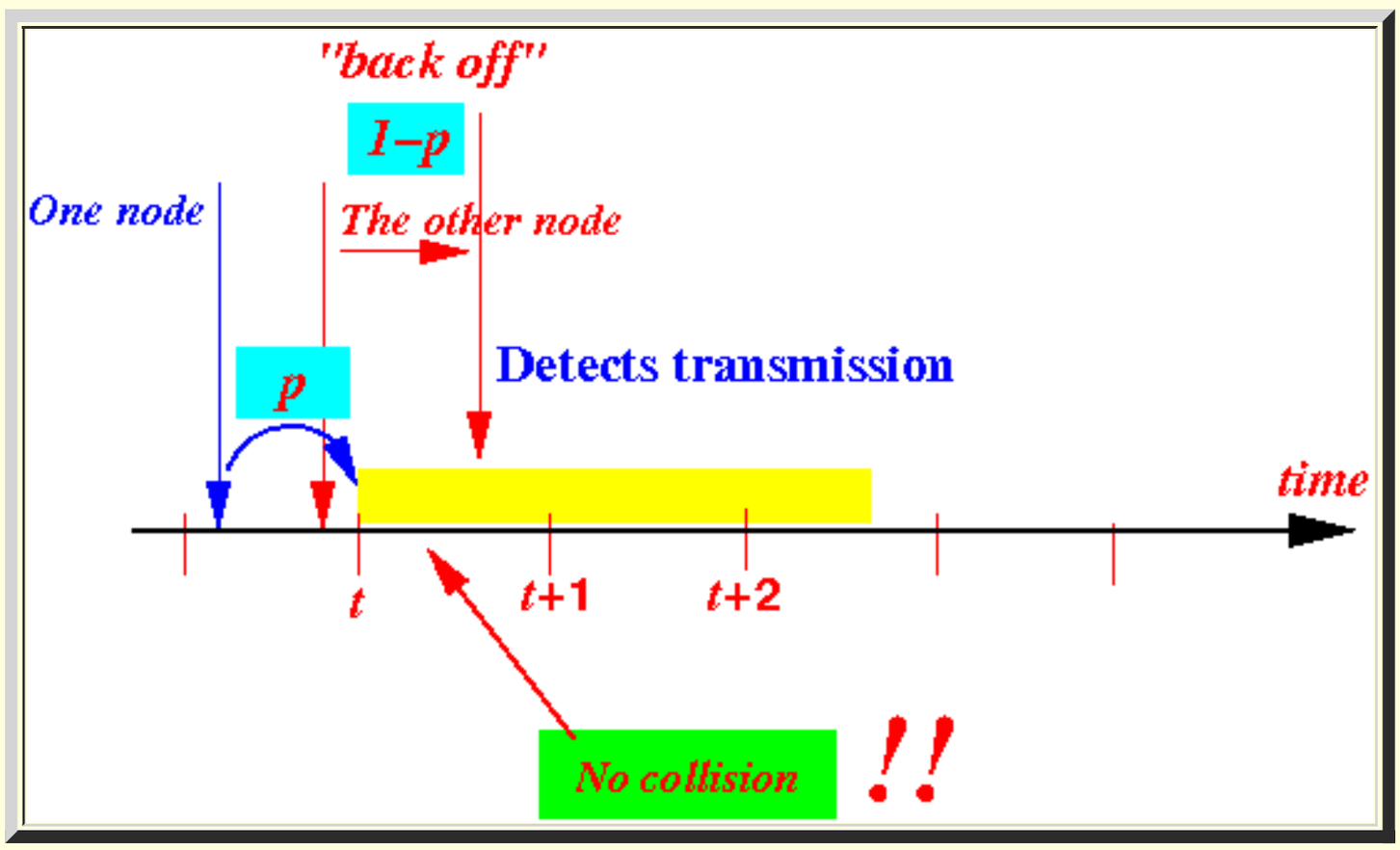


Reason:

- There **may** be ***multiple*** **nodes** sensing at the ***same*** **time**:



- **Then**: ***one*** **node** may decide to **transmit**, while the ***other*** **node** may decide to **back off**:

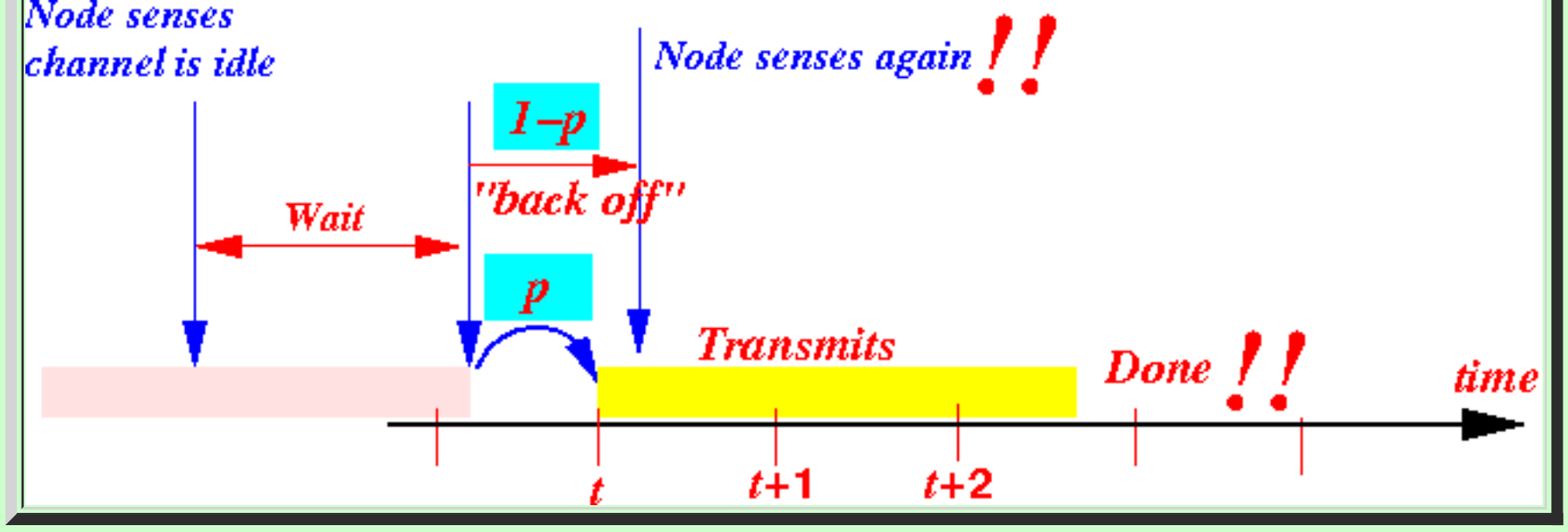


- If the **node** **senses** that the **carrier** is **busy**, then:

  - **Wait** until the ***current*** **transmission** ***ends***

  - **Transmit** with **probability** ***p***

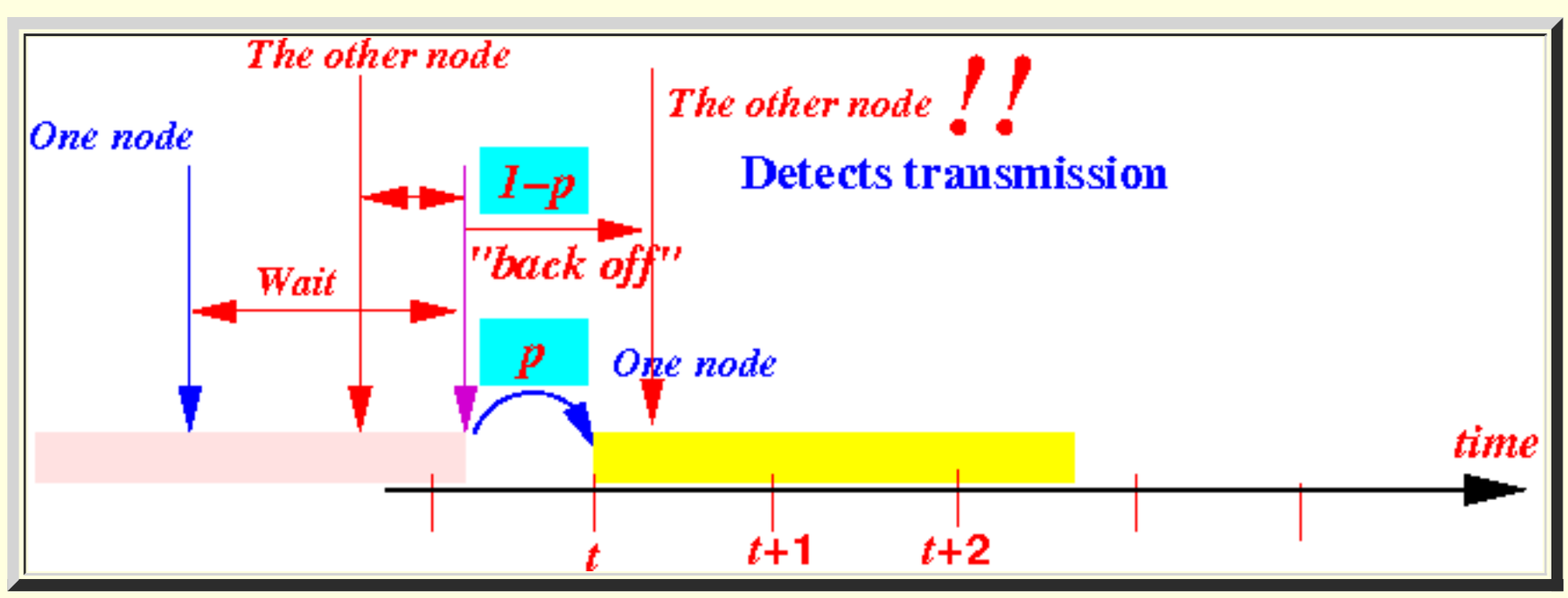
Graphically:



Reason:

- There **may** be ***multiple*** **nodes** sensing at the ***same*** **time**:

- **Then**: ***one*** **node** may decide to **transmit**, while the ***other*** **node** may decide to **back off**:



- **What's value should we use for *p* ????**
  - **Selecting** the ***p*** value in ***p*-persistent CSMA**:

    - ***p*** must be set to a value such that:
      - It is **very likely** that **exactly *one* node** will pick a **random number *x*** such that:

        `x < p`

        I.e.: ***exactly* 1 node** should **transmit** its **message**

- **Result** from **probability theory**:

  - If ***n* nodes** are **sensing**, then the **probability**:

    $$p = \frac{1}{n}$$

    will **maiximize** the **likelihood** that ***exactly* one** node will ***transmit*** (and ***(n-1)* nodes** will ***back off***)

- **Example**:

  - ***If*** there are **2 nodes** **waiting** for the **current transmission** to finish, then use:

    $$p = 0.5$$

  - **On the other hand**, if there are **3 nodes** **waiting** for the **current transmission** to finish, then use:

    $$p = 0.33333333\ (1/3)$$

    And so on...

Conclusion:

- ***p*** must be **dynamic**

It's **hard** to know **how many** nodes are **busy**...

So:

- The ***p*-persistent CSMA** protocol is **not implemented** as a real life protcol.

Performance Analysis of CSMA protocols

- **Performance Analysis of CSMA protocols**
  - **Notations:**

    - **Time unit** = length of a **slot**

    - **G** = the **offered load** (number of attempted transmissions per time unit)
    - **S** = the **throughput** (number of ***successful*** transmissions per time unit)

  - **Performance** of the various **CSMA** protocols:

We shall not derive the expressions for each of these methods. Under certain assumptions, Tobagi (Kleinrock and Tobagi, 1975) has derived the equations:

$$S_1 = \frac{Ge^{-G}(1+G)}{G+e^{-G}} \tag{3.9}$$

$$S_n = \frac{G}{1+G} \tag{3.10}$$

$$S_p = \frac{Ge^{-G}(1+pGx)}{G+e^{-G}} \tag{3.11}$$

$$\text{where} \quad x = \sum_{k=0}^{\infty} \frac{(qG)^k}{(1-q^{k+1})!} \tag{3.12}$$

Notations:

- S_1 = the **throughput** of the ***1-persistent*** **CSMA** protocol
- S_n = the **throughput** of the ***non-persistent*** **CSMA** protocol
- S_p = the **throughput** of the ***p-persistent*** **CSMA** protocol

---

- **Comparing the performance of various CSMA protocols:**

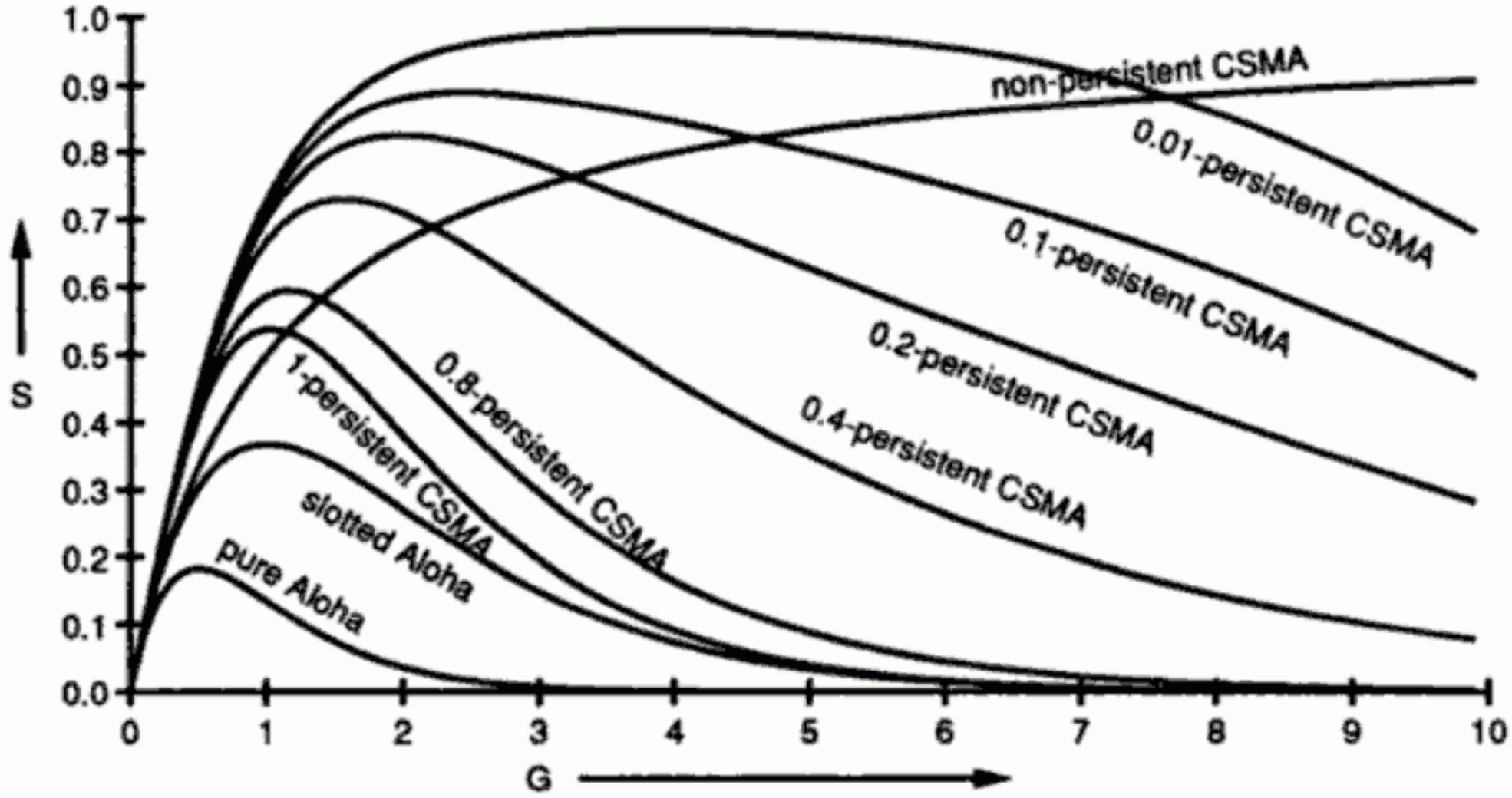
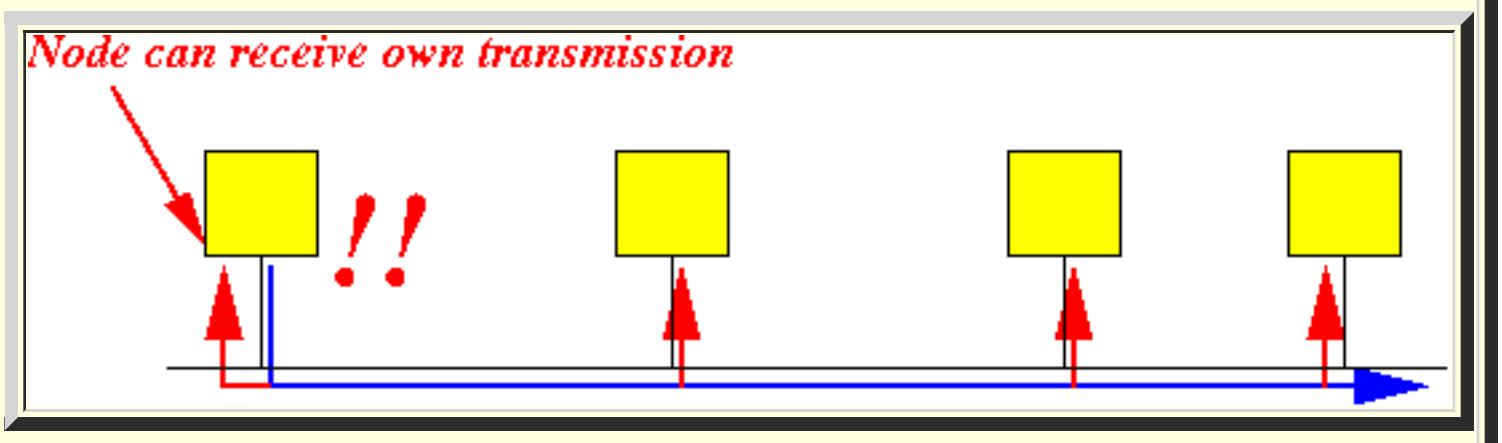


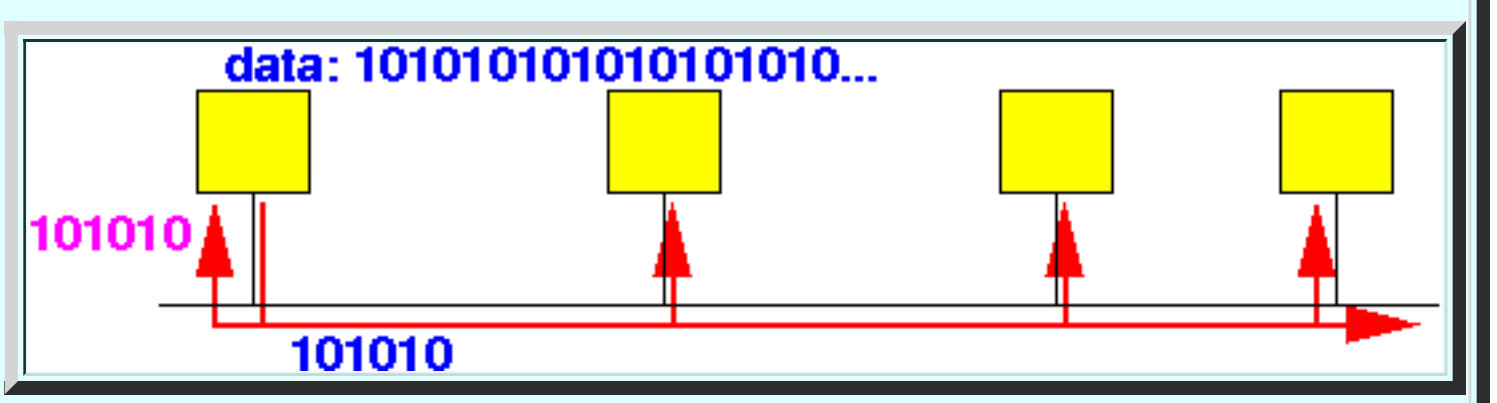
Fig 3.11. Channel Throughput: Aloha and CSMA

Intro to *Collision Dectection*

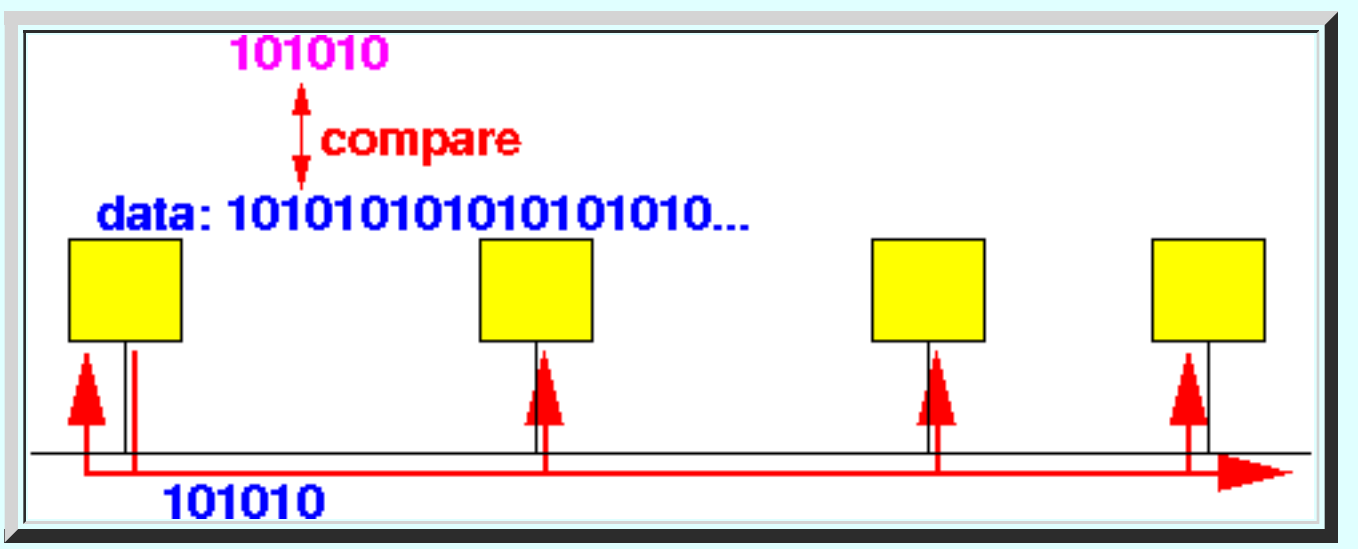
- **Collision Detection**
  - **Fact:**
    - A **transmitting node *can*** receive its ***own* transmission**:



  - **How to** detect a **collision:**
    - While **transmitting**, the **sender receives** its ***own* transmission**:



    - ***Compare*** the **data received** against the **data transmitted**:



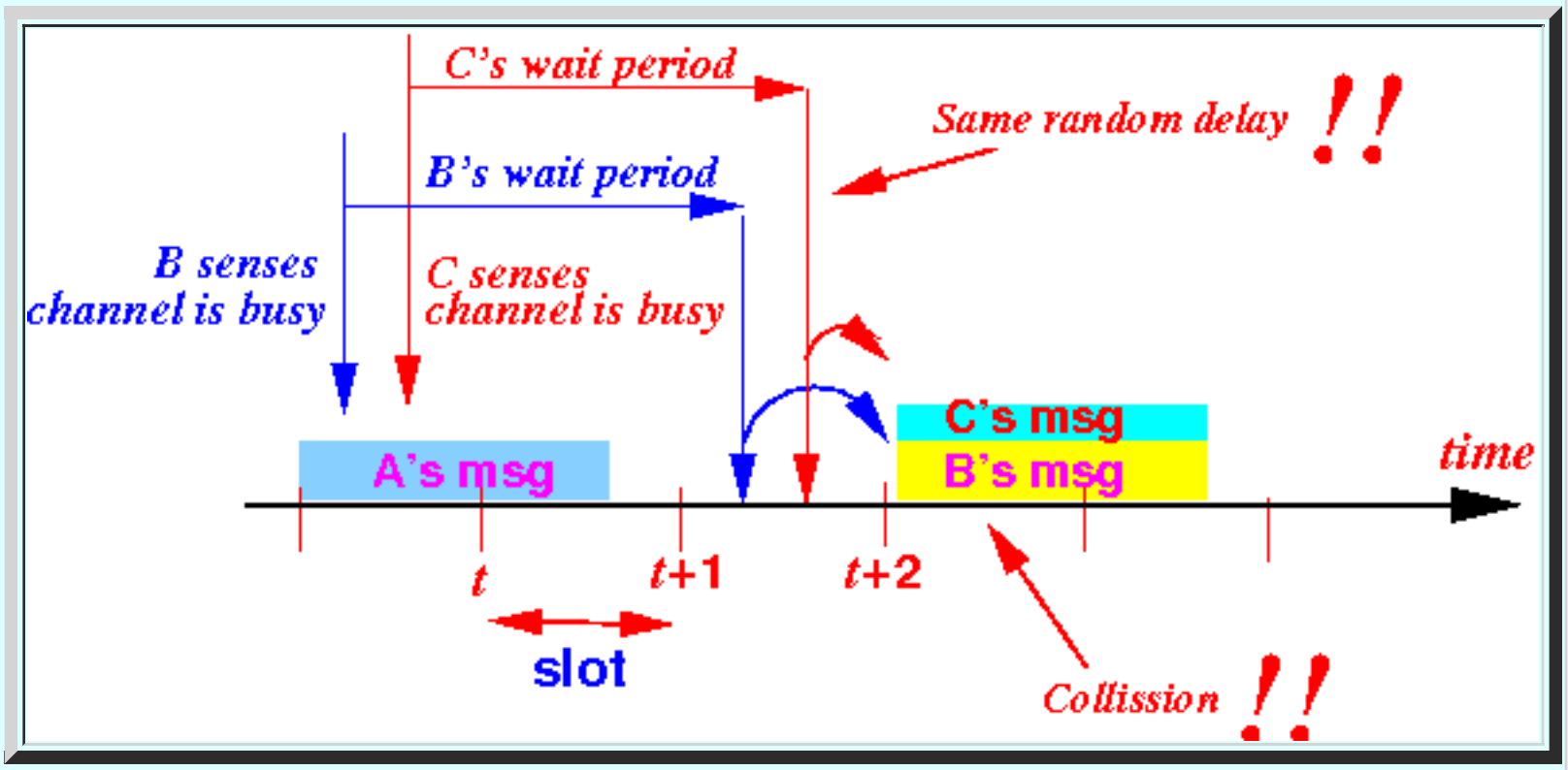
    - ***Collision*** will be **detected** through:
      - **Bit *errors***

- **Collisions in the CSMA protocols**
  - **Fact:**

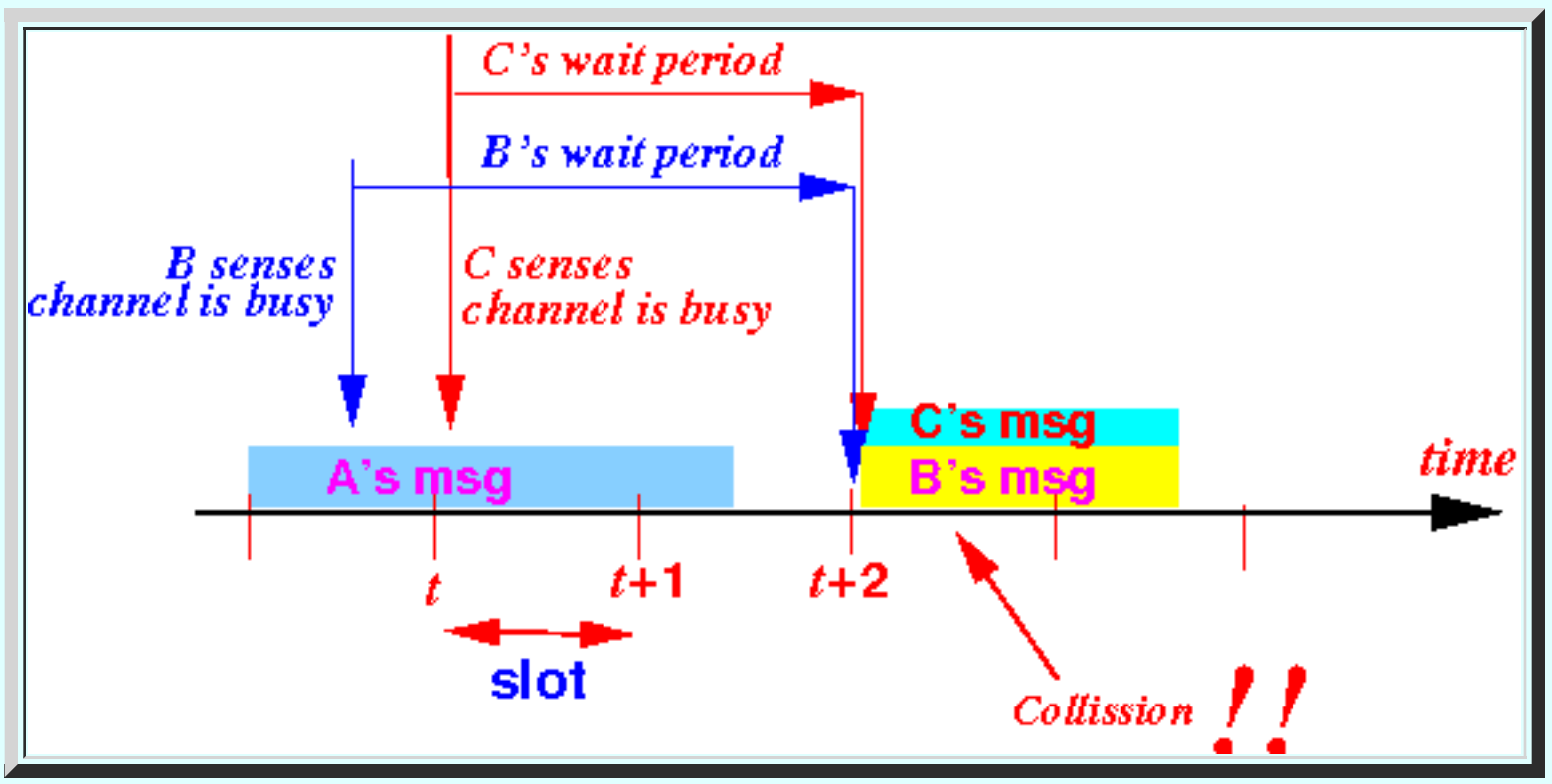
    - ***Every* variant** of the **CSMA protocols** can have **collisions**

    Examples:

    - In ***non*-persistent CSMA**, a **collision** happens when **2 nodes** pick the ***same* random number**:

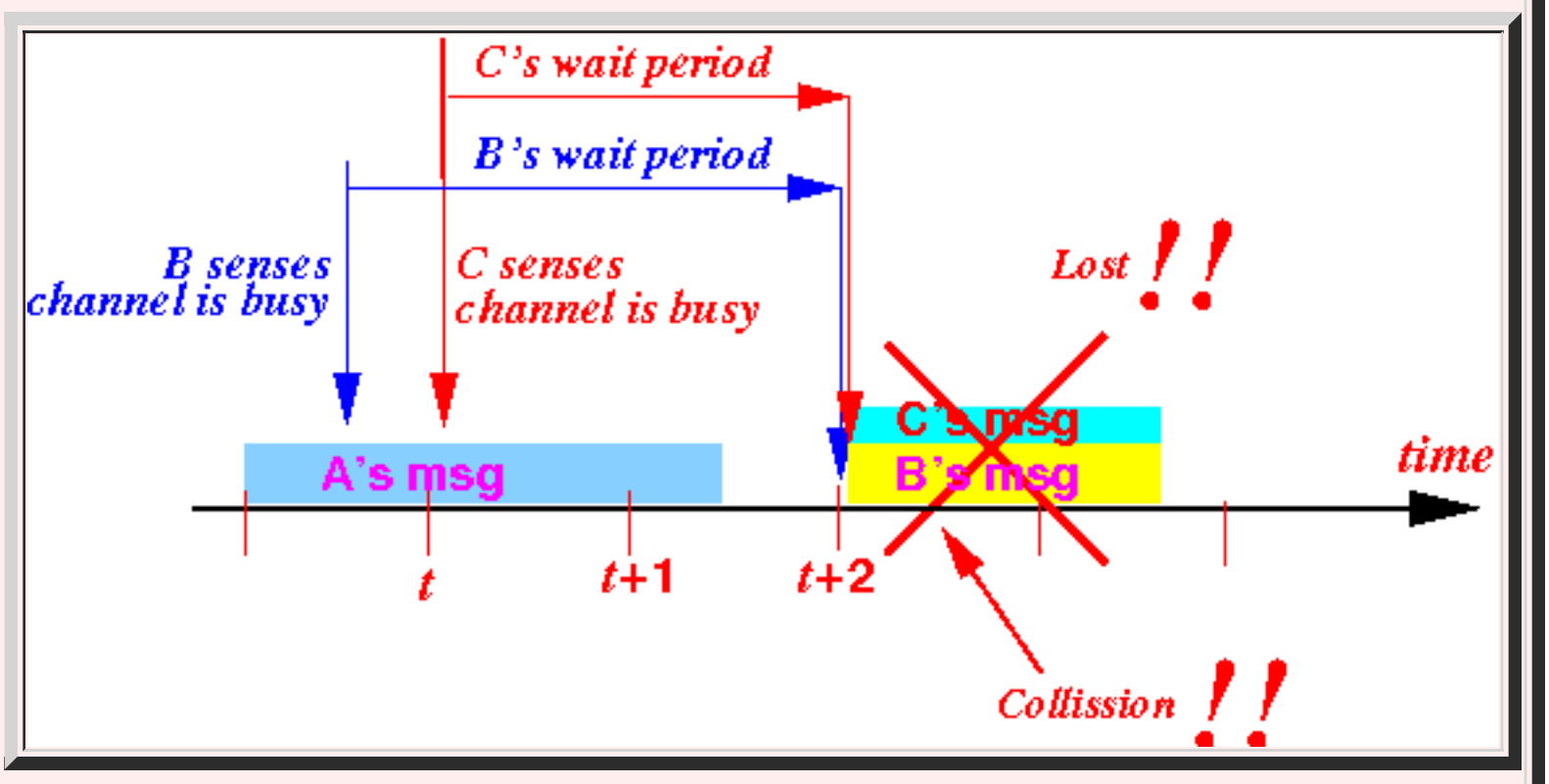


    - In ***1*-persistent CSMA**, a **collision** happens when **2 nodes starts sensing** during the ***same* transmission**:



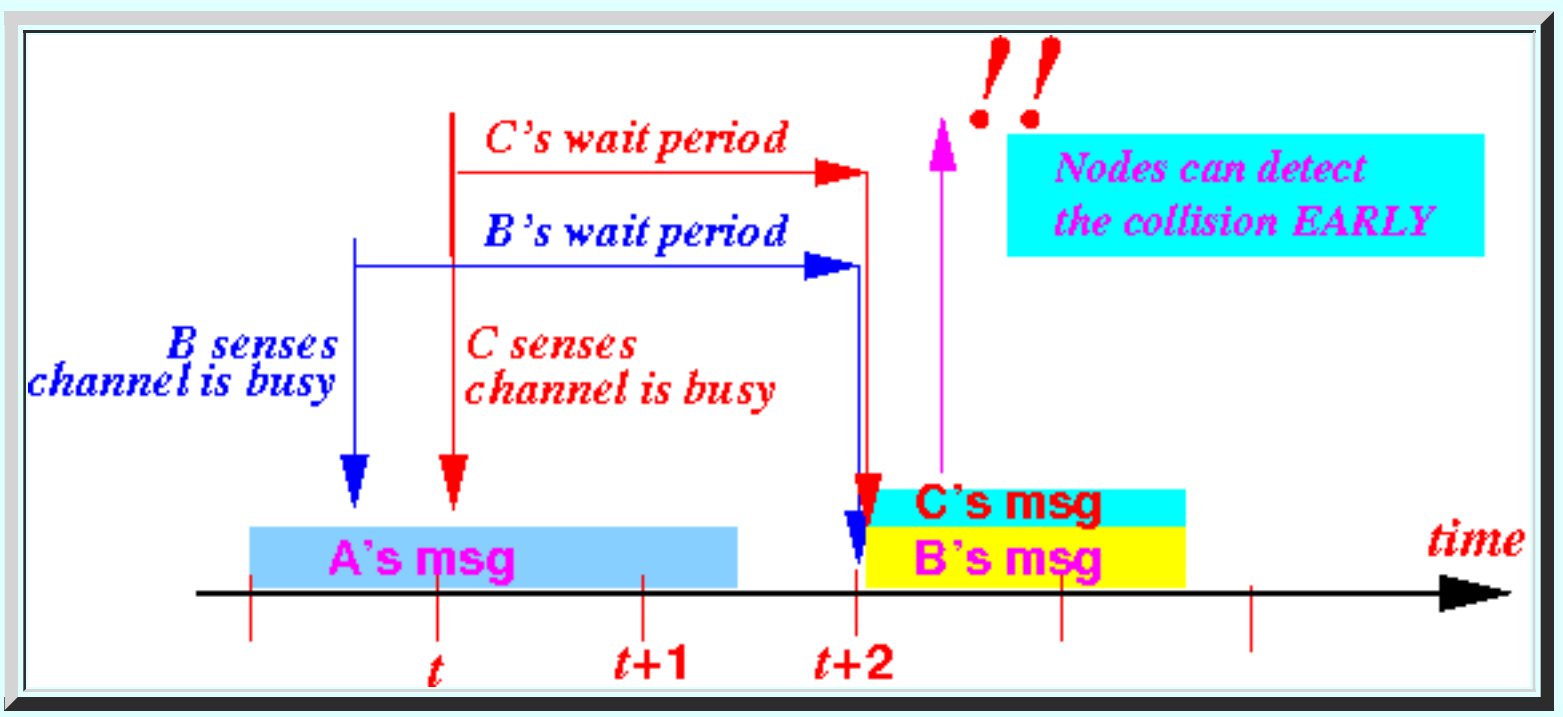
  - **Result** of **collisions**:

- The **transmission (frames)** involved in a **collission** are ***lost***:

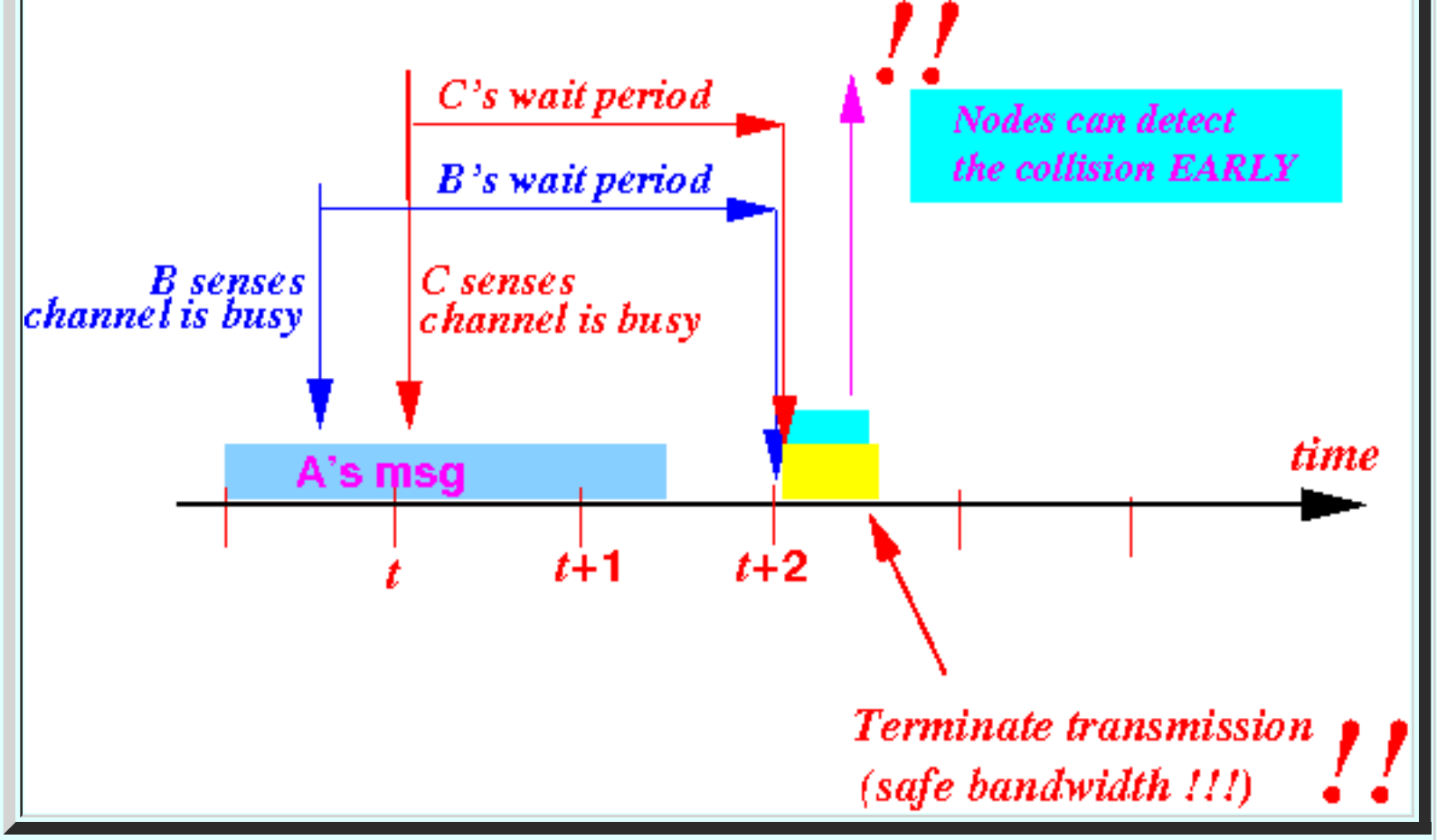


- ***Improving*** **the CSMA protocol with** ***Collision Detection***
  - **How to** improving the **performance** of **CSMA** with ***Collision Detection***:

- The **nodes** can **detect** a **collision** very ***early***:



- The **nodes** can ***terminate*** the **transmission** **prematurely**:



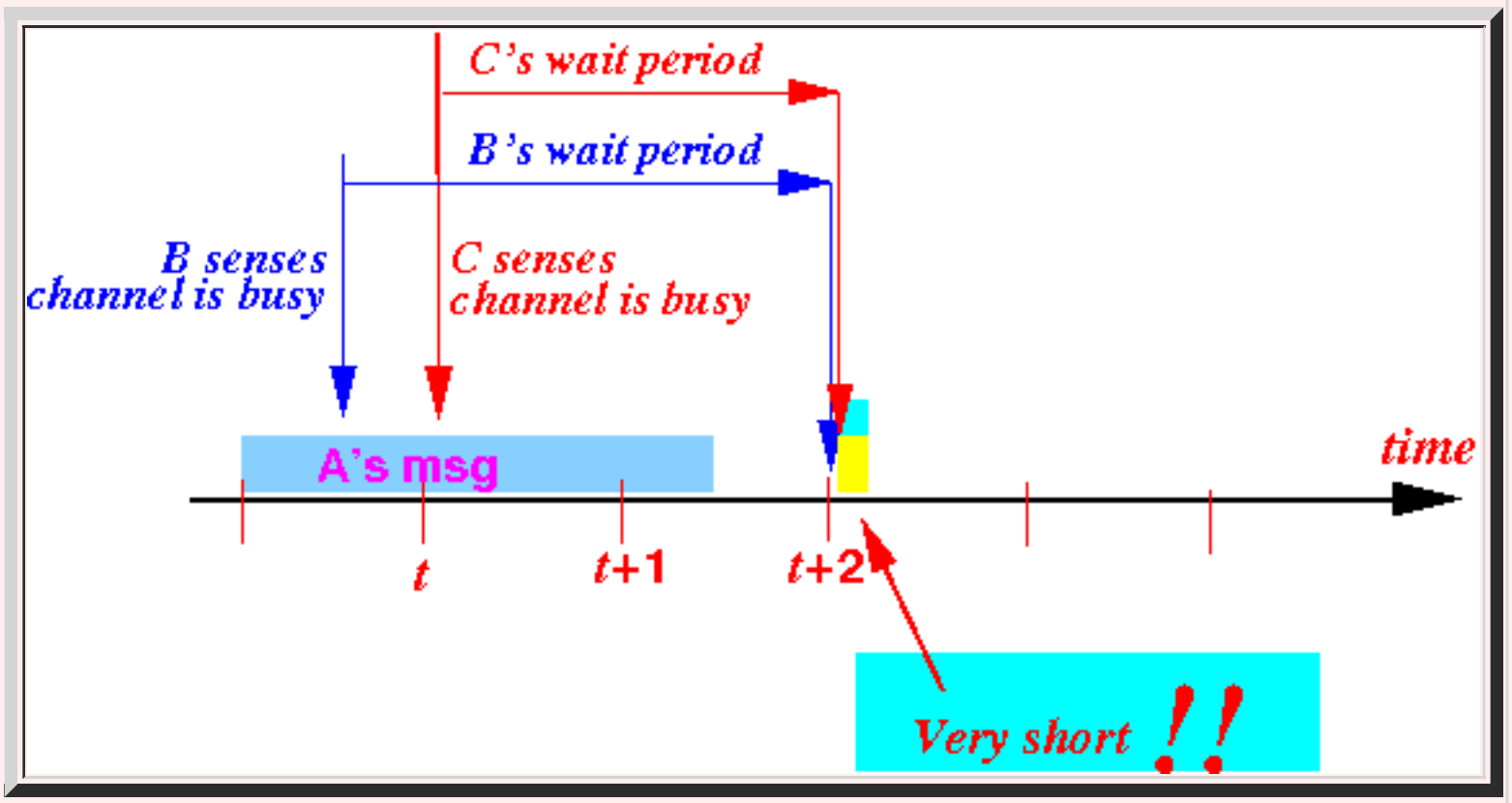
- **Result:**

  - ***Less* transmission time** are ***wasted*** in a ***useless* collission**

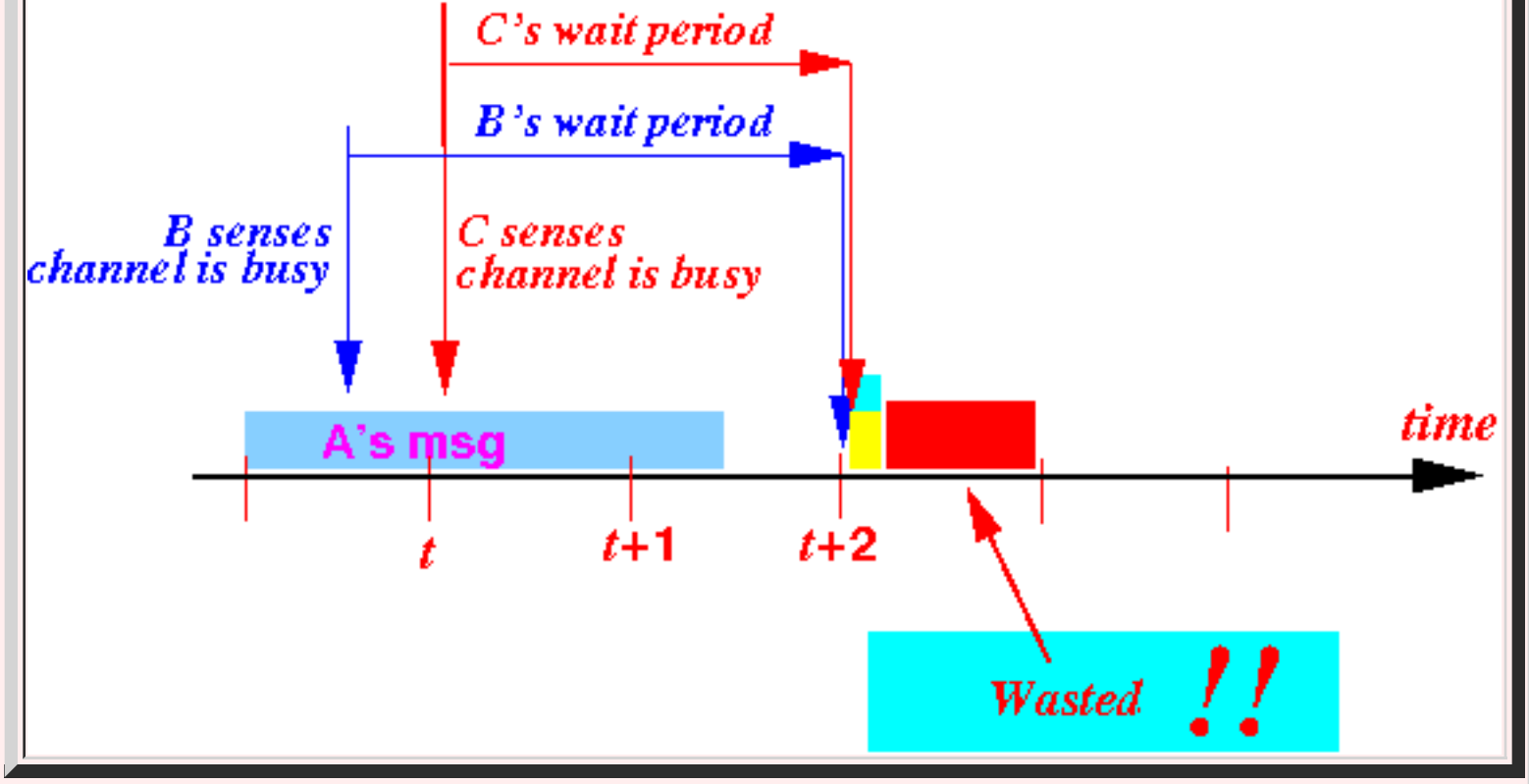
- **CSMA/CD uses unslotted transmissions**

  Important fact:

  - The ***collision* period** is ***very* short** compared to a **slot time**:



  - ***Slotted* transmissions** will ***waste*** the ***remainder*** of the **slot**:

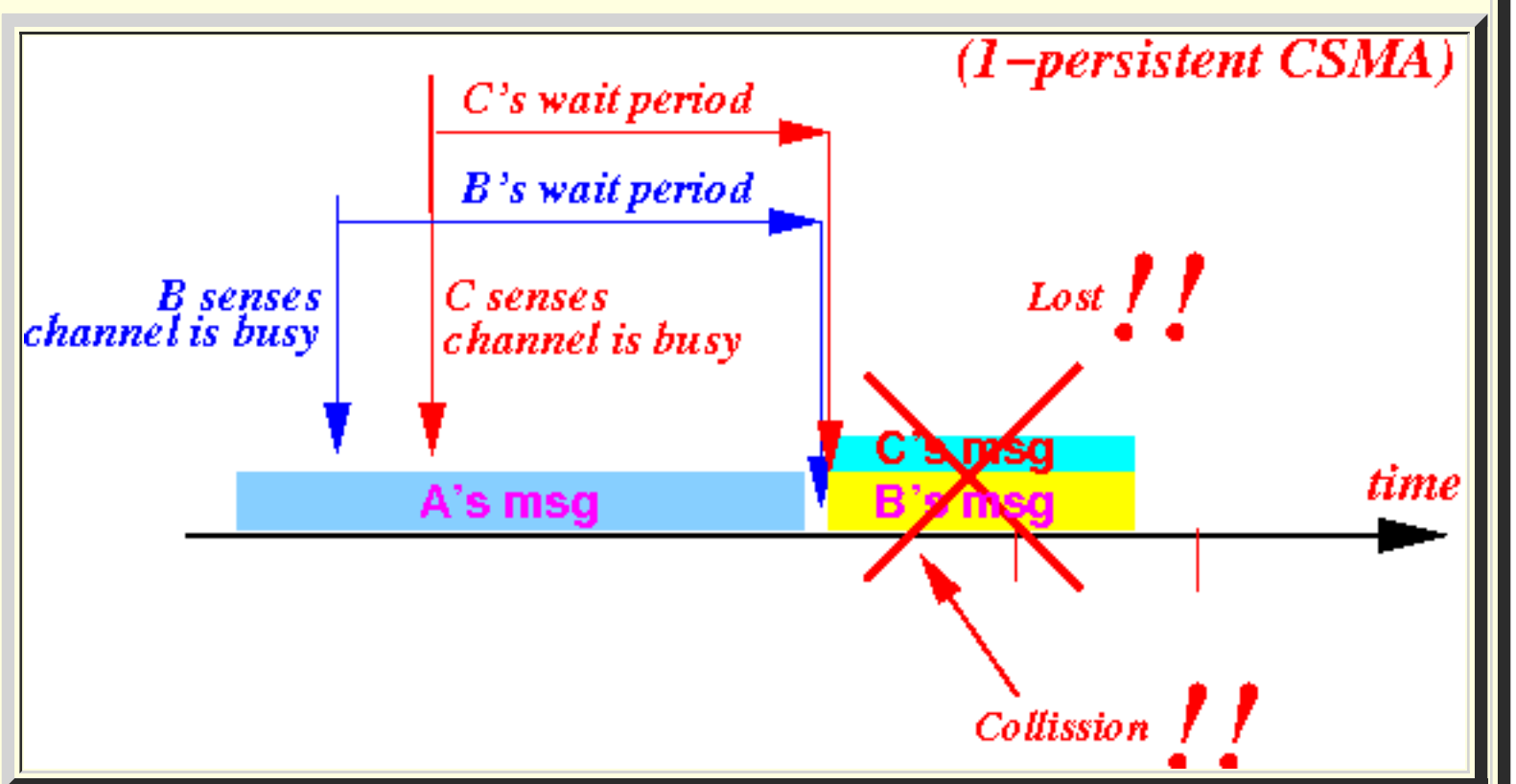


- **CSMA** with **collision detection** uses:
  - ***Unslotted*** **transmissions** to ***minimize*** **wastage** of **transmission time**

Action taken after a positive collision detection --- the CSMA/CD protocol

- **What happens *after* a node detects a *collision* ?**
  - **Fact:**

    - **Collision** will render ***all* (colliding) frame** to be ***lost***:



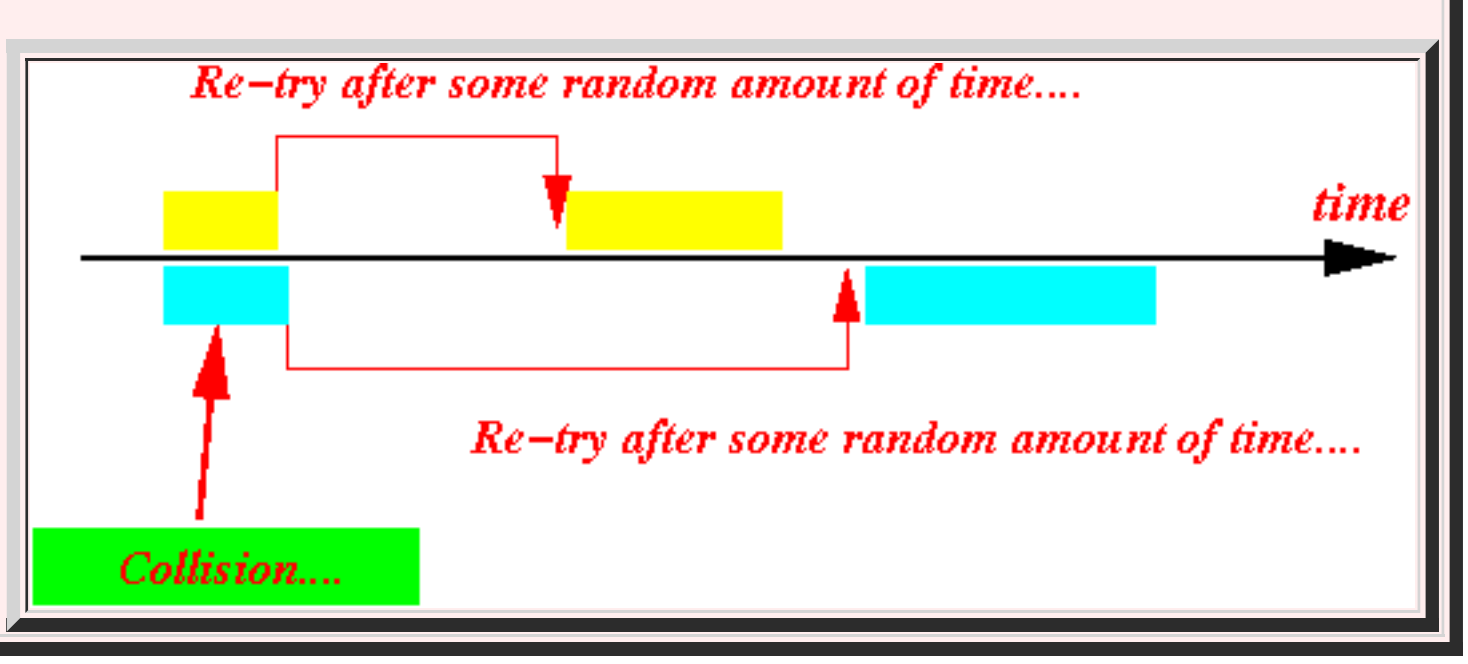
  - **What** to **do** when a **node** detects a **collision**:

    1. ***Back off*** (= **wait**) for a **random period**

       ---

    2. **Re-try** again

       Graphically:

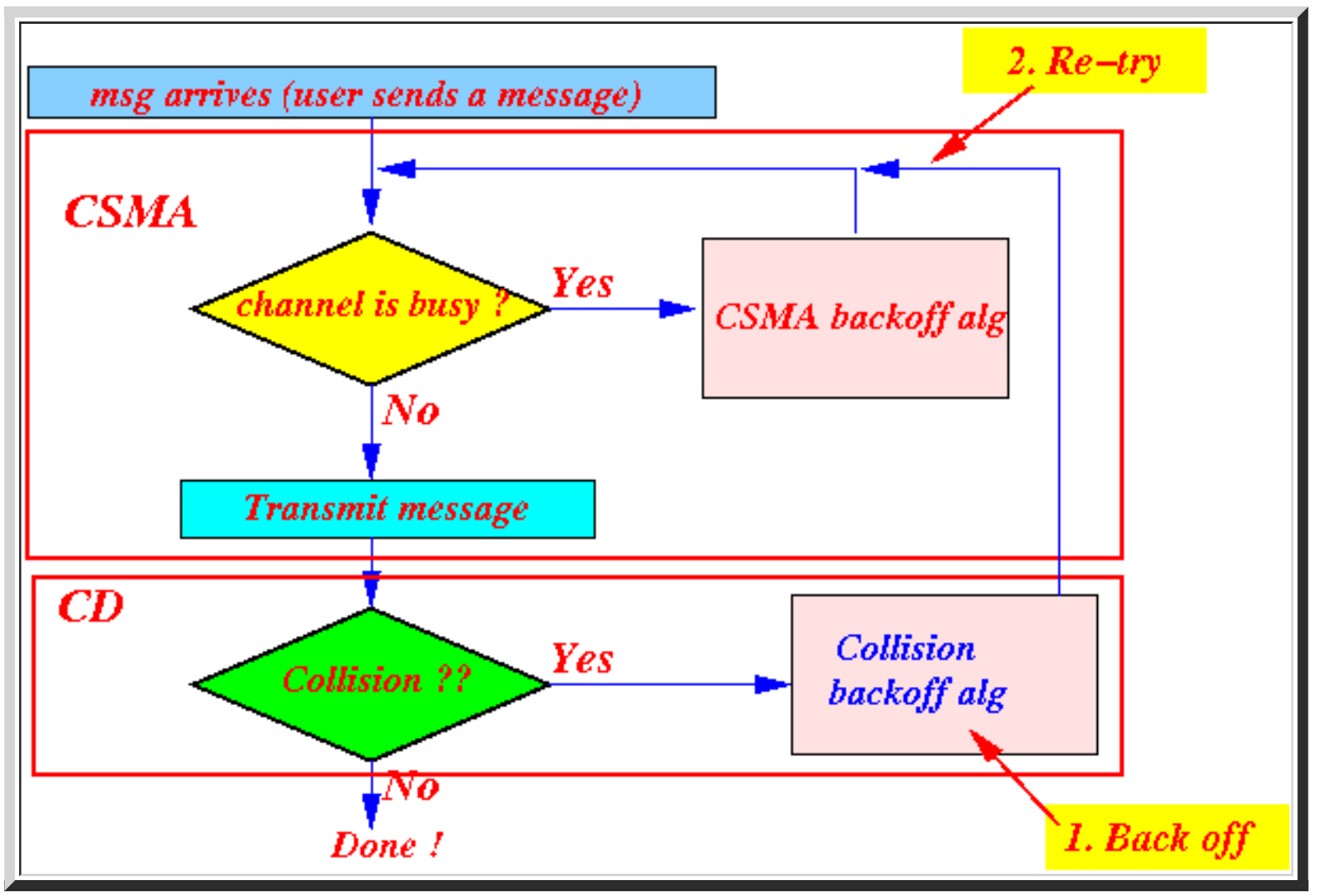


Note:

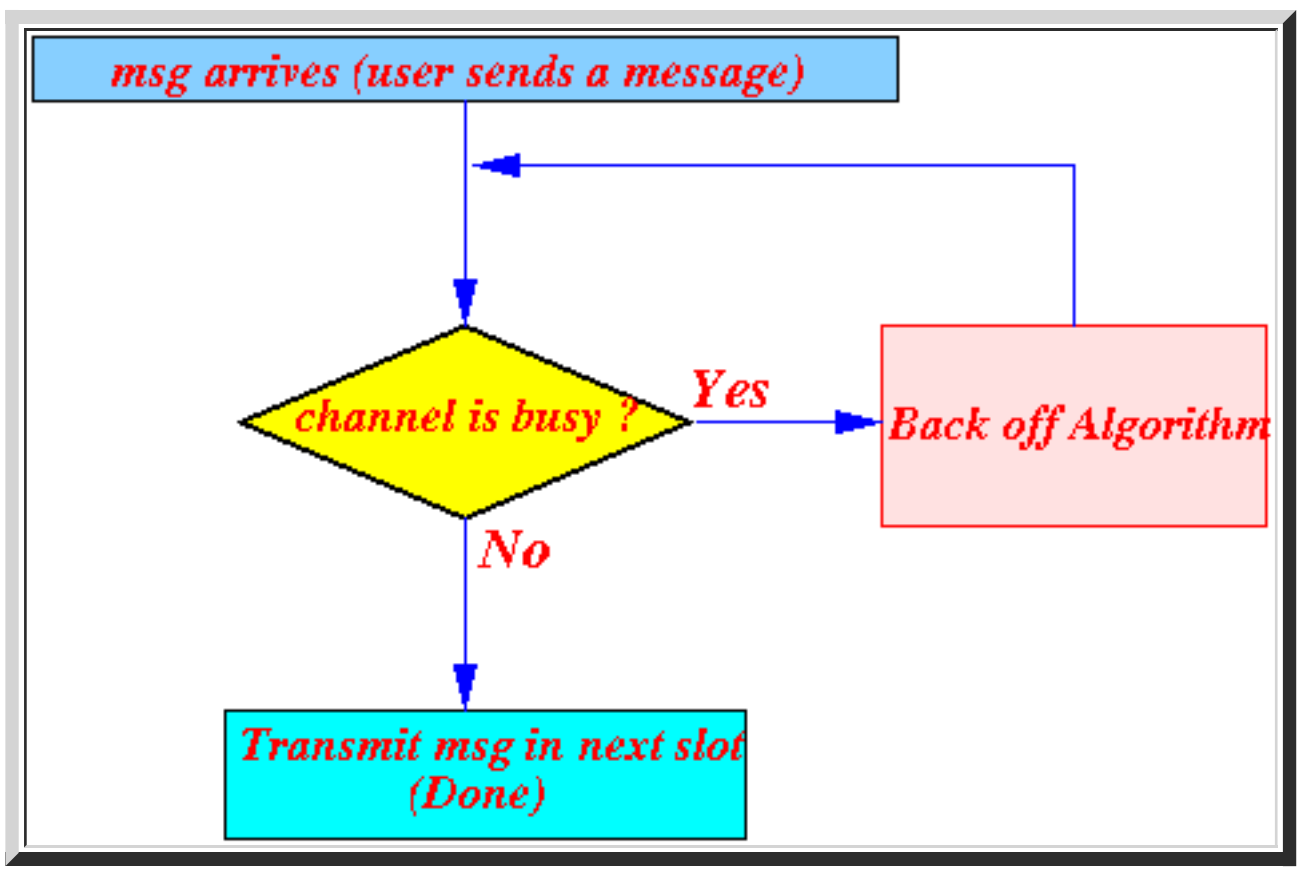
- The ***random* backoff period** will **help** the **nodes** to ***avoid*** a **new collision** when they **attempt** to **re-transmit** their **frames** !!!

- **The *CSMA* protocol with collision detection (CD)**

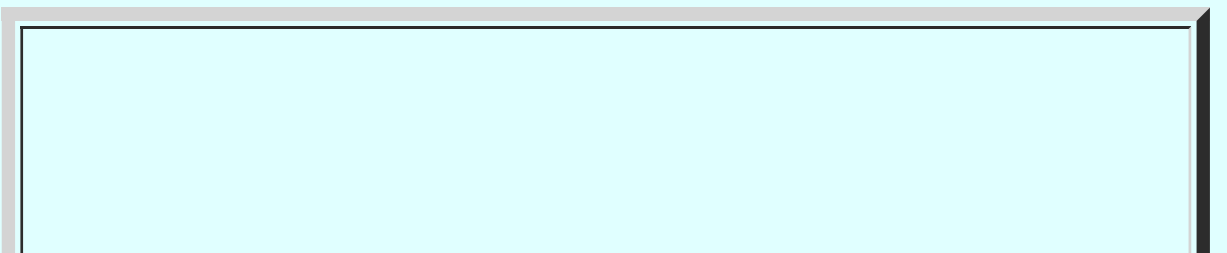
- **Flow chart** for **CSMA/CD** protocol:

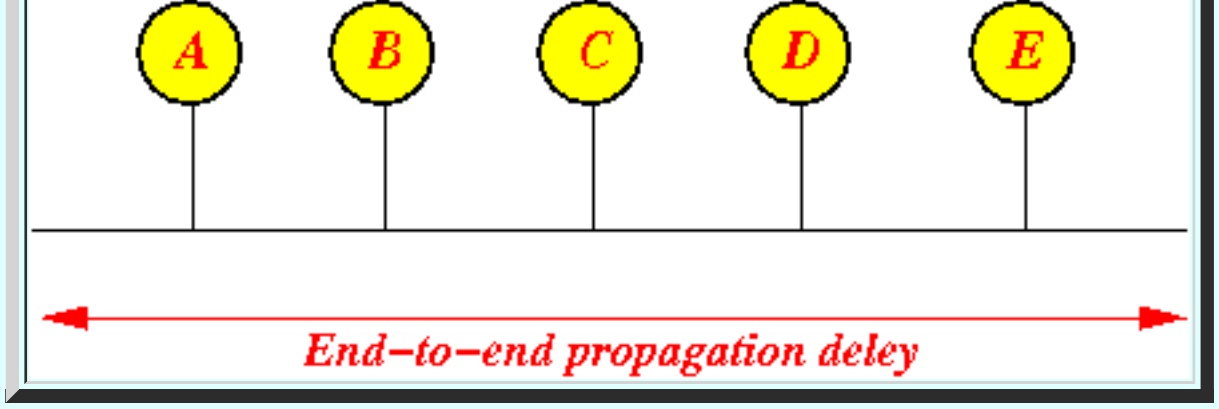


- **Compare** that to the **flow chart** for **CSMA**:



- **The time unit used in the Back Off Algorithm in CSMA/CD**
  - **Recall: End-to-end propagation delay**
    - **End-to-end propagation dela** = **amount of time** that it takes for **electric sigmals** to **travel** from **one-end** of the **network** to the ***other* end** of the **network**:





- **Notation**:

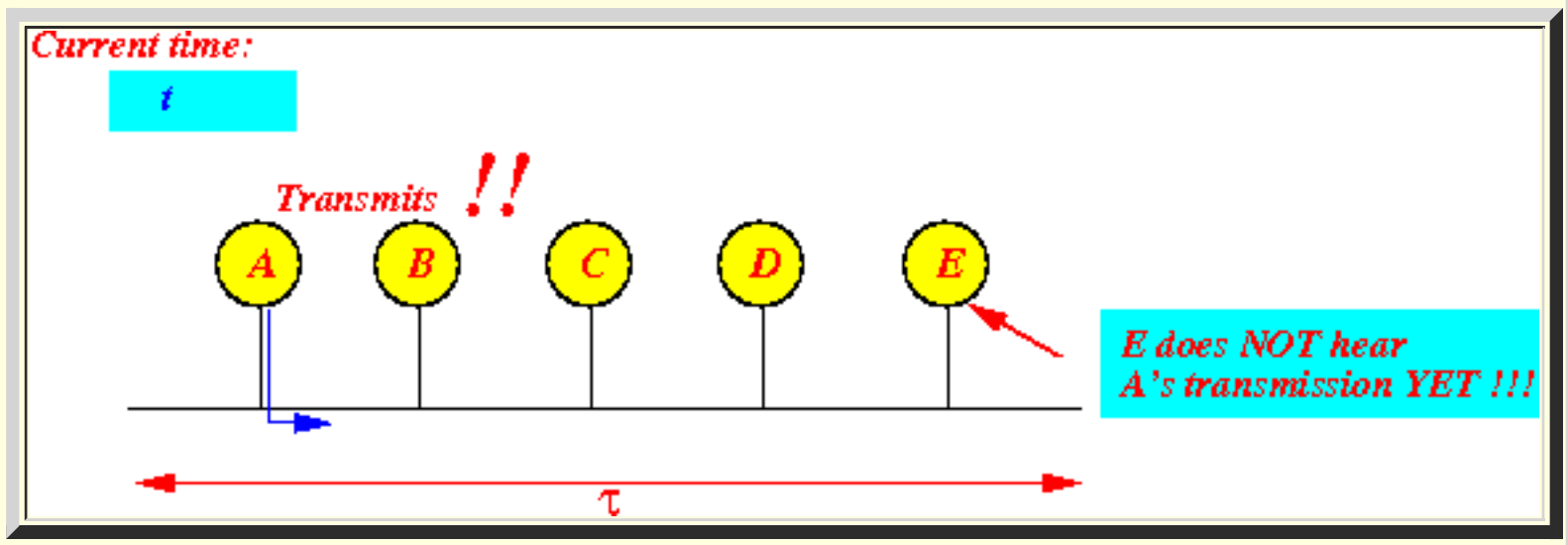
  - τ = **end-to-end propagation delay**

- **Property** of the **end-to-end propagation delay (τ)**

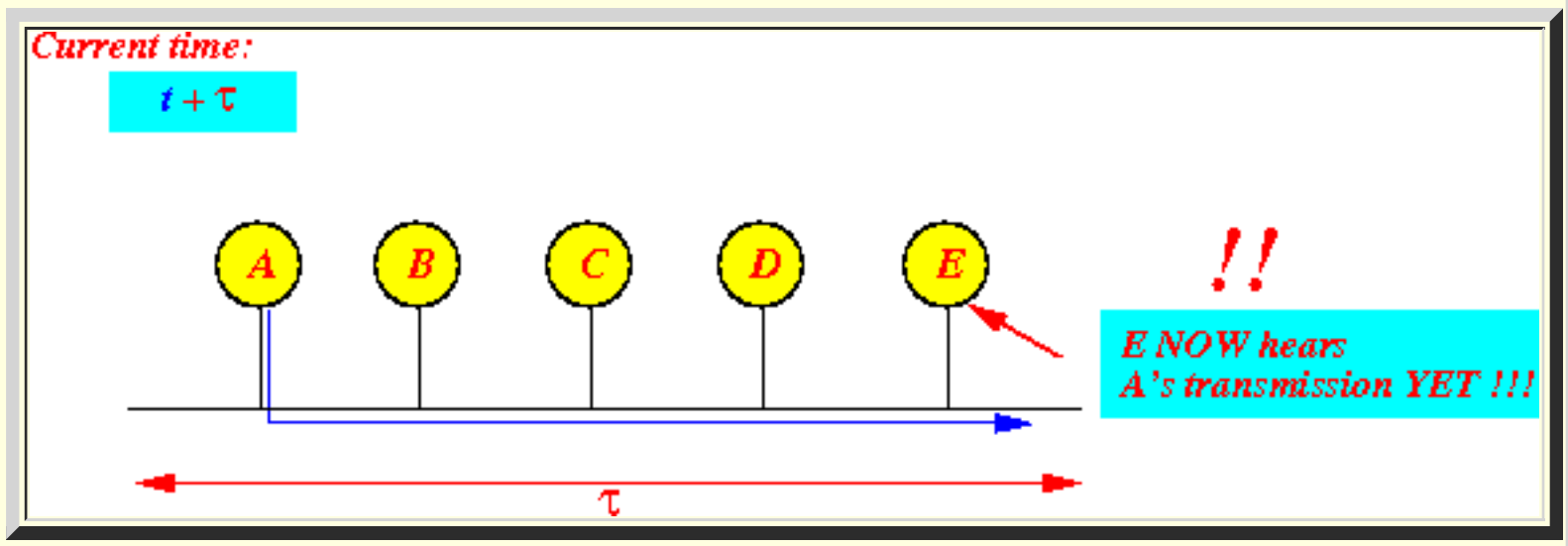
  - ***All* nodes** on the **LAN** will **hear** a **node's transmission *after* τ sec** after the **start** of the **transmission**

    Graphically:

    - A **(any) node** on the **LAN *starts* transmitting** at **time *t***:



    - At **time ($t + \tau$) sec**, ***every* node** on the **LAN** will **hear (= detect)** the **transmission**:



- **Back off time period**:

  - **Back off time duration** = $r \times \tau$ (r = a **random number**)

  We **choose** τ to be the **"time unit"** of the **back off algorithm**

  We see see the **reason why next**...

- **The generic Back Off Algorithm in CSMA/CD**
  - A ***generic*** **back off** algorithm:

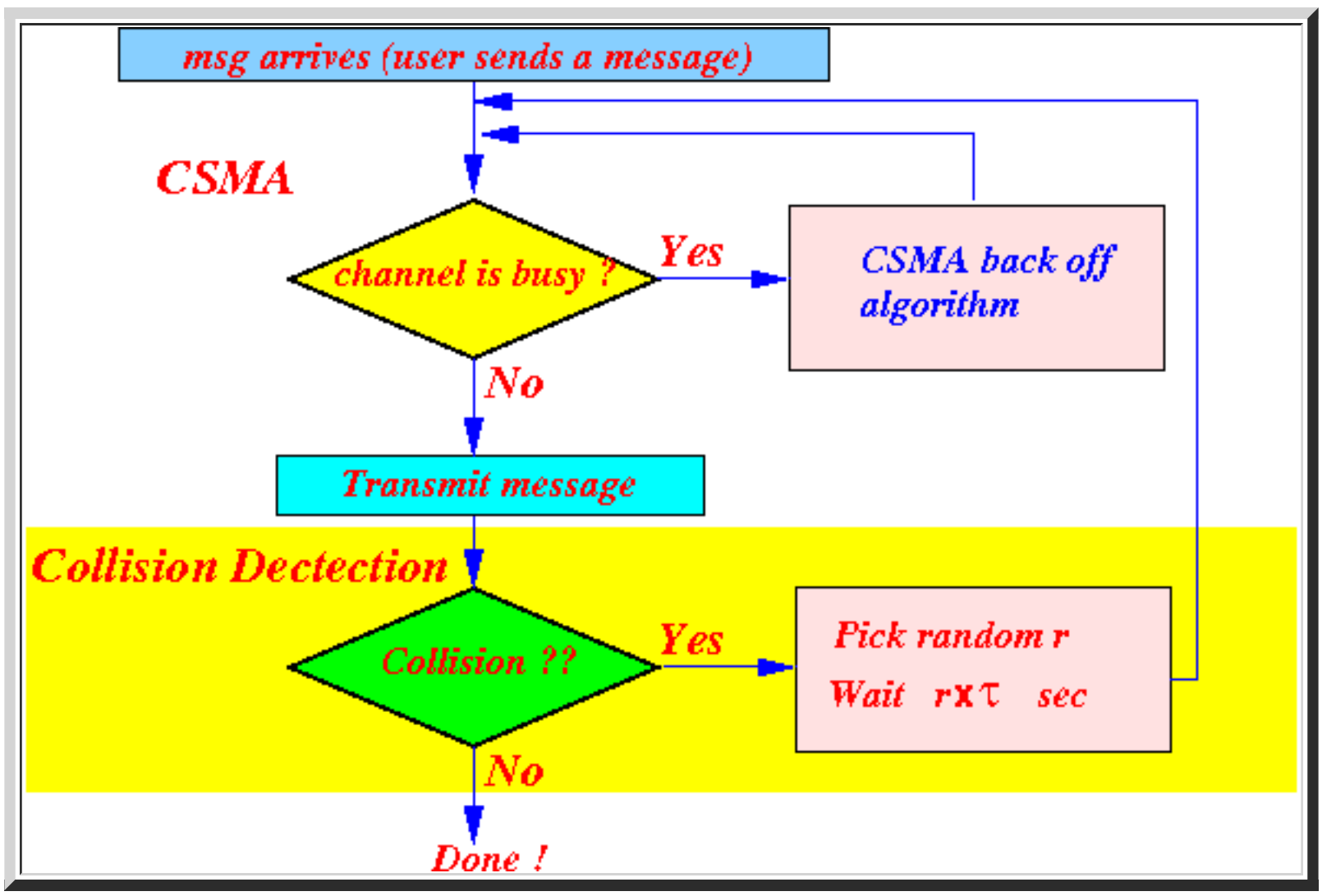
```
Pick a random integer r

Wait  r × T  seconds

(Start the transmission (CSMA/CD) protocol from the start)
```

Important fact:

- If **2 different nodes** **picks** ***different*** **random integers**:
  - The **back off periods** used by these **nodes** will **ensure (guarantee)** the ***later*** **starting node** will **hear (= detect !)** the **transmission** of the ***earlier*** **starting node** when the ***later*** **starting node re-tries** its **transmission** !!!

This will **allow** the ***later*** **starting node avoid colliding** with the ***earlier*** **starting node** !!!

  - **Updated** (**improved**) **flow chart** for **CSMA/CD**:



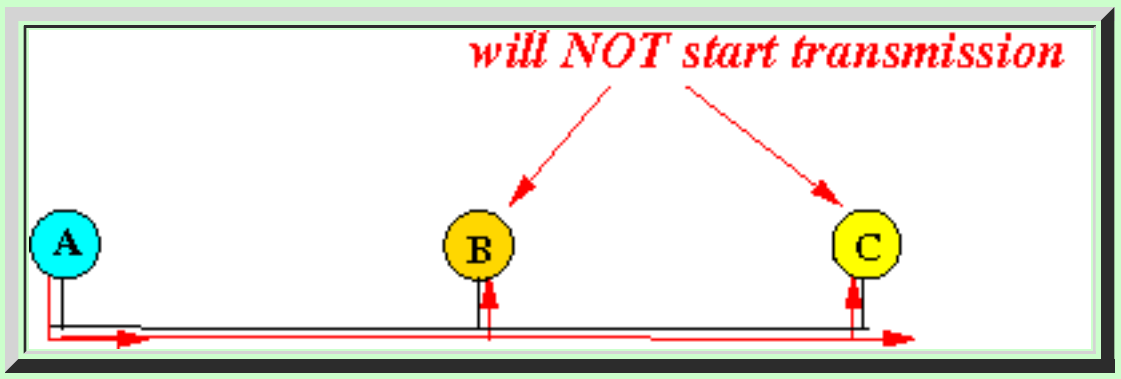
A node does not need to perform collision detection forever ....

- **A node can stop sensing for collision after $2 \times \tau$ time**
  - **Claim:**

    - ***If*** **node** has **trasnmitted** for $2 \times \tau$ (τ = end-to-end-delay), and it **did *not*** detect a **collision**, ***then***:
      - The **transmission** will **not** suffer a **collision** for ***certain*** !!!

    Proof:

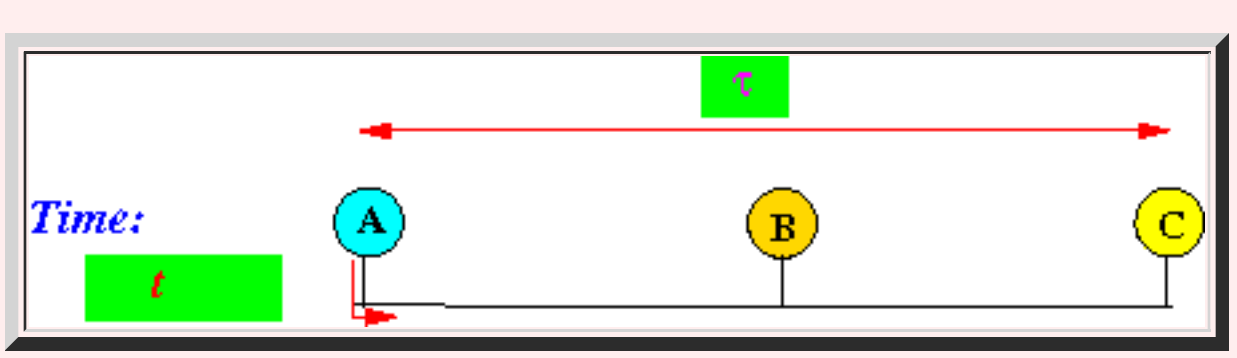
    - **Fact:**
      - In **CSMA/CD**, a **node** will **not** transmit if it **hears** an ***on-going*** **transmission**:



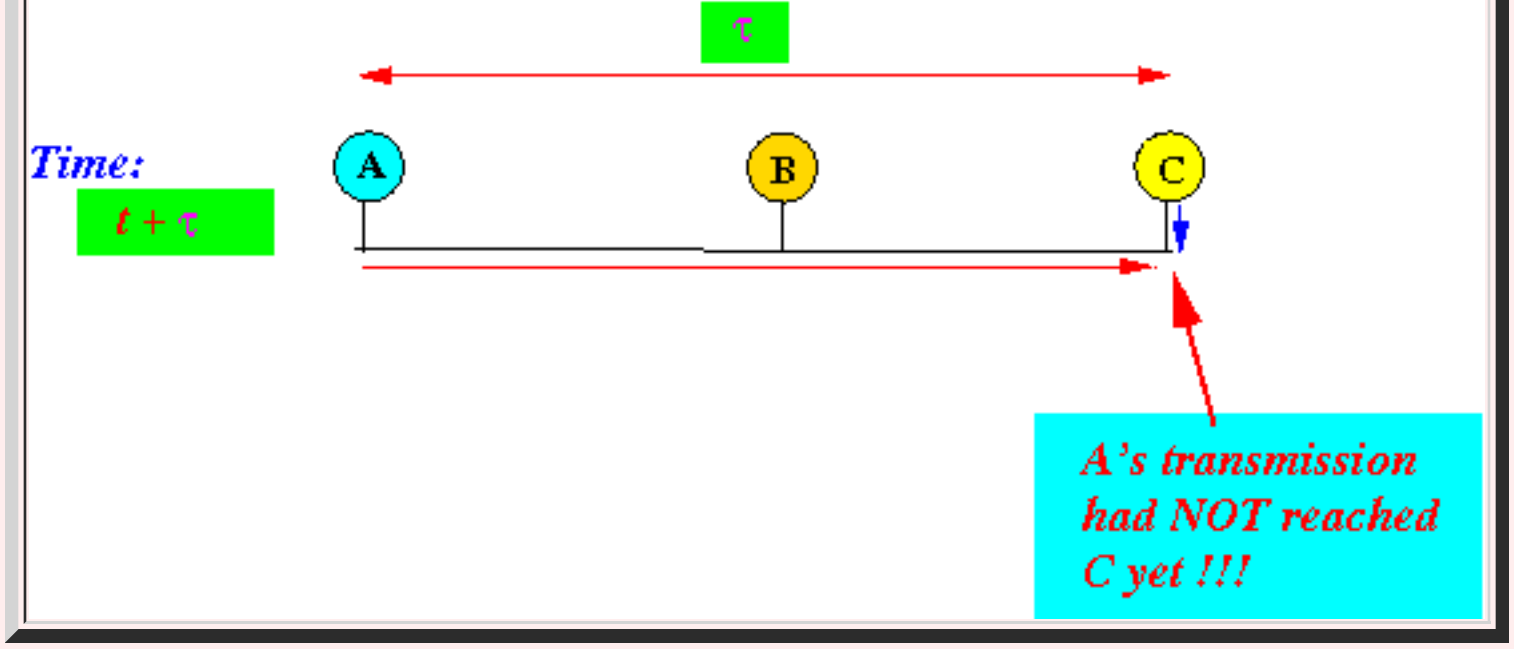
- We **consider** the ***worst case*** **scenario** where:
  - a **collision** takes place at the ***last*** **possible moment**

  This is the ***worst case*** **scenario**:

  - The **node *A*** at **one end** of the **network** start **transmitting** at so ***arbitrary*** time ***t***:



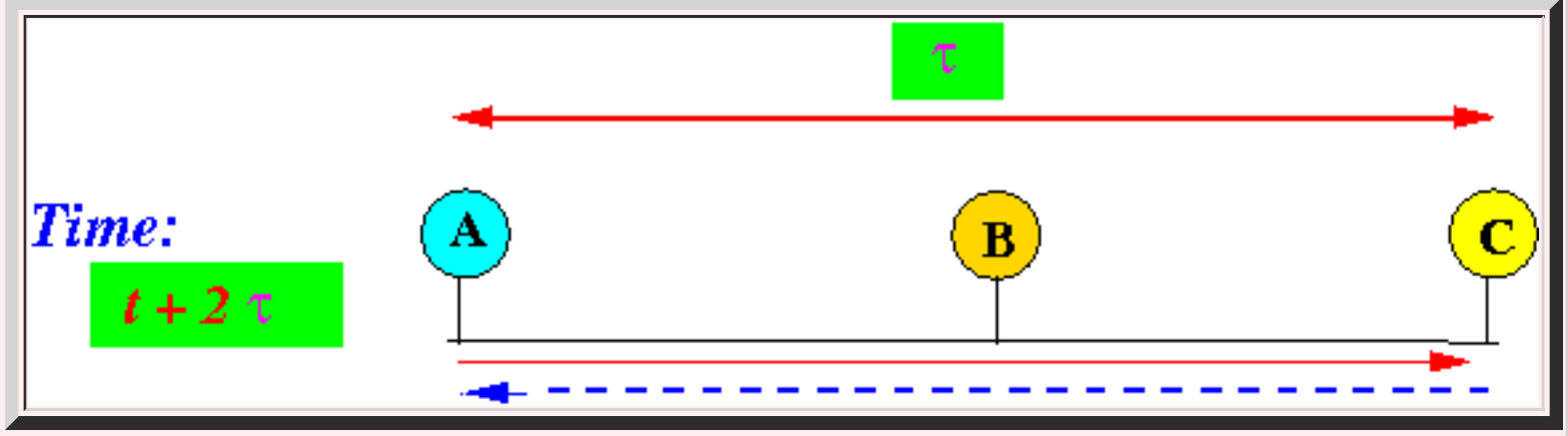
  - ***Just*** before the ***last*** **node (*C*)** hears ***A*****'s transmission**, it **starts** to **transmit**:



This will **happen** just ***before*** **t+τ** sec

- There is **now** a **collision** !!!

- The **node *A*** will **detect** the ***collision*** when ***C*'s transmission** reaches ***A***:



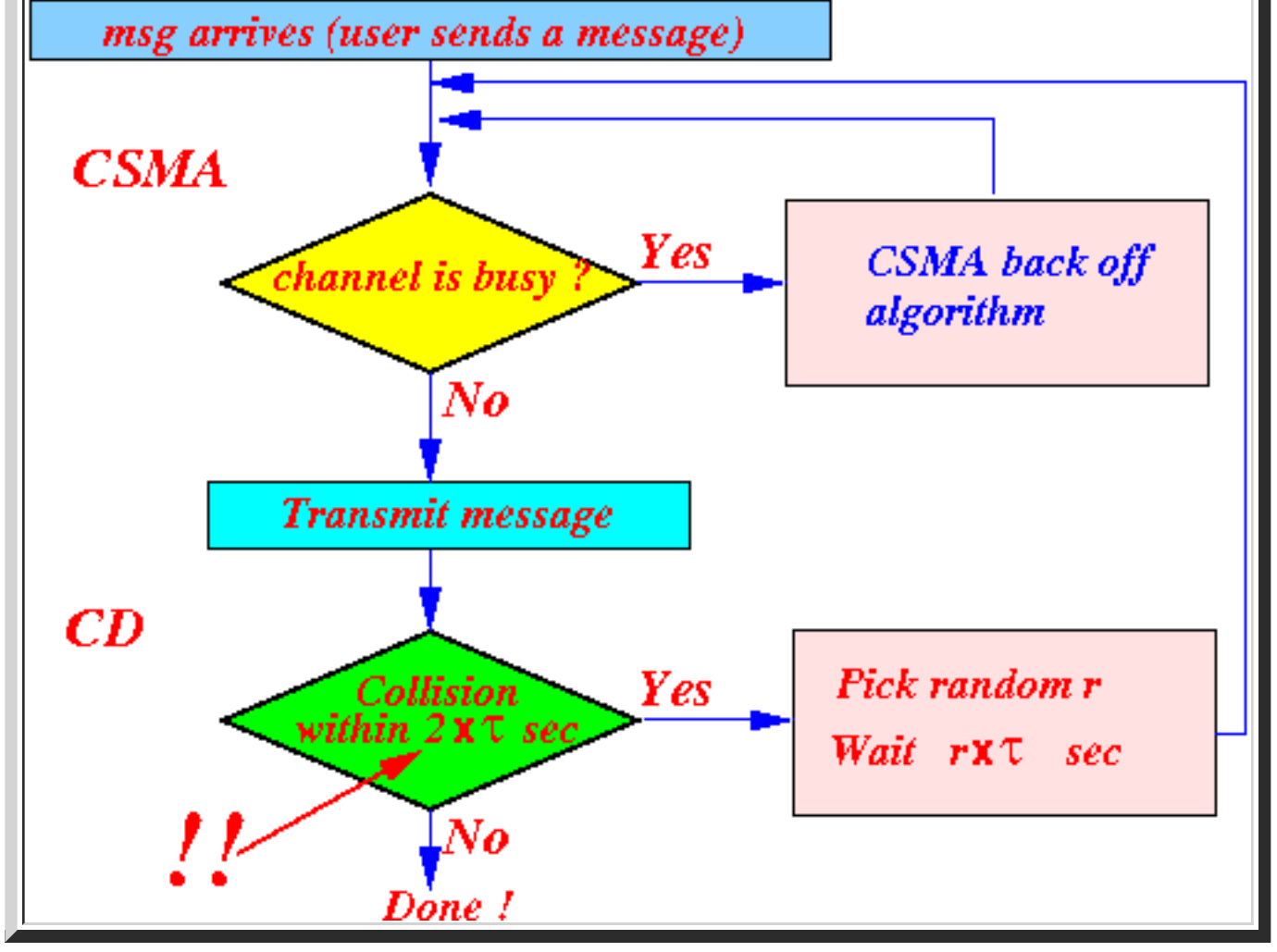
This will **happen** at time **t+2×τ** sec

- **Conclusion:**
  - ***If*** the **node *A*** has ***not*** **detect a collision** after **2× τ** time, there **will *not*** be any ***future*** **collision !!!**

- **Therefore:**
  - A **node** can **stop sensing** the **channel** after for **2 × τ** time

---

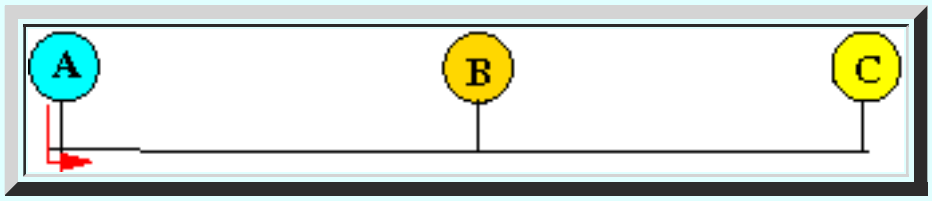
- **The general CSMA/CD protocol (chart)**
  - ***Almost*** **final flow chart** for **CSMA/CD**:


msg arrives (user sends a message)
CSMA
channel is busy ?
Yes
CSMA back off algorithm
No
Transmit message
CD
Collision within 2 x τ sec
Yes
Pick random r
Wait r x τ sec
!!
No
Done !

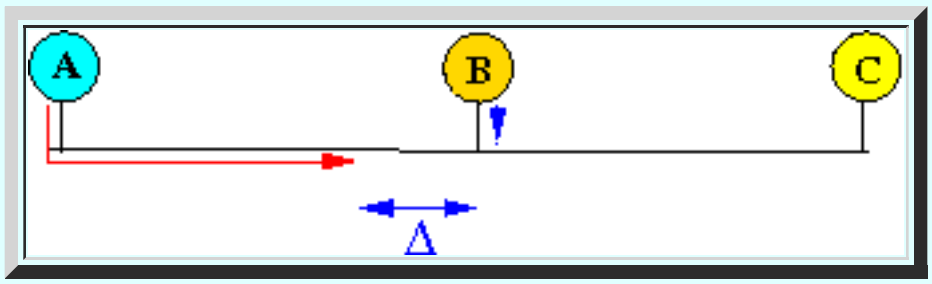
Jamming to ensure collision detection

- **Duration of a collision**
  - The **duration** of a **collision**:

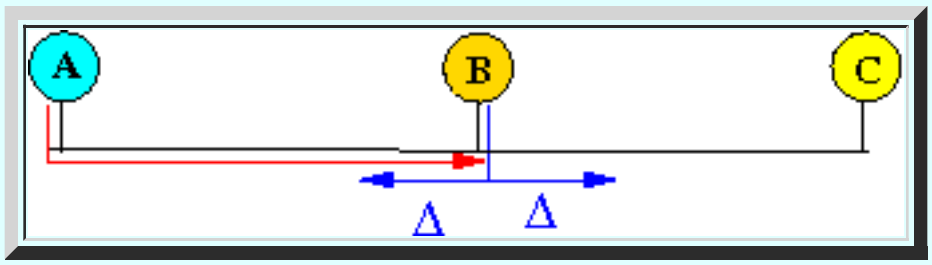
    - **Suppose node *A* starts a transmission**:



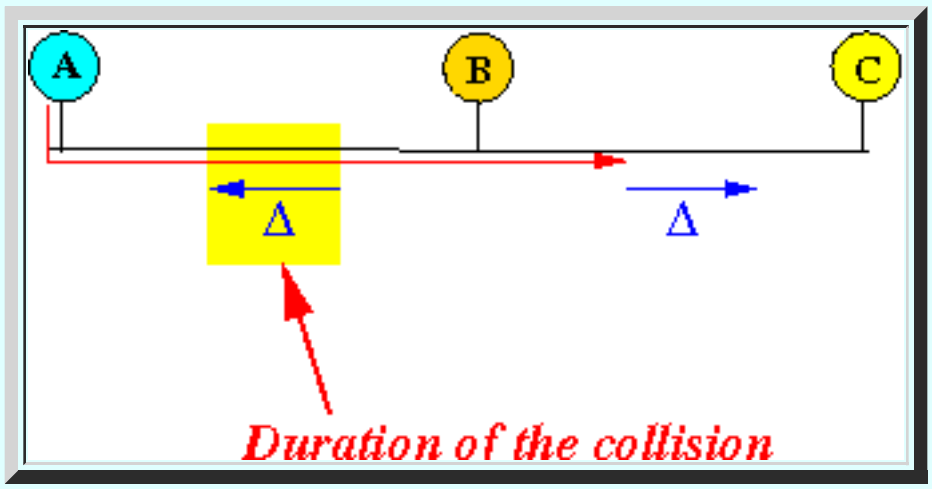
    - **Node *B* starts to transmit at Δ time *before A's transmission* arrives at *B***:



    - **Suppose node *B* will stop transmitting when *B* hears *A's transmission***:



    - **Then:**

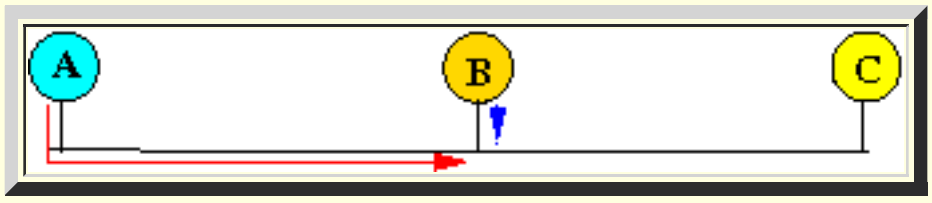


    the ***duration*** of the **collision** is **equal** to:

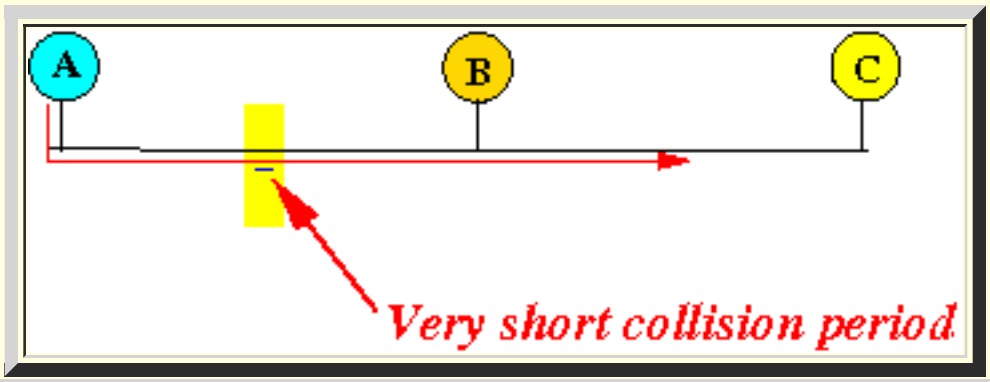
- Δ sec

- **The duration of collision can be *very* short**
  - **Fact:**

    - If a **node (*B*) *starts*** a **transmission *just* before** a **transmission *arrives***:



    the ***duration*** of **collision** will be ***very* short**:



  - **Fact:**

    - ***Short* collision periods** generates ***few* bit errors** and ***may* escape detection** !!!

- **Improving collision detection**
  - **Making sure** that a **collision** is ***easily* detectable**:

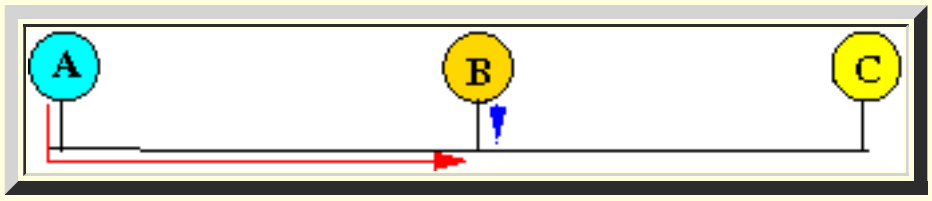
    - When a ***transmitting* node** has **detected** a **collision**, it will:

      1. **Transmit** a **(short) jam signal**
      2. **Stop** transmitting

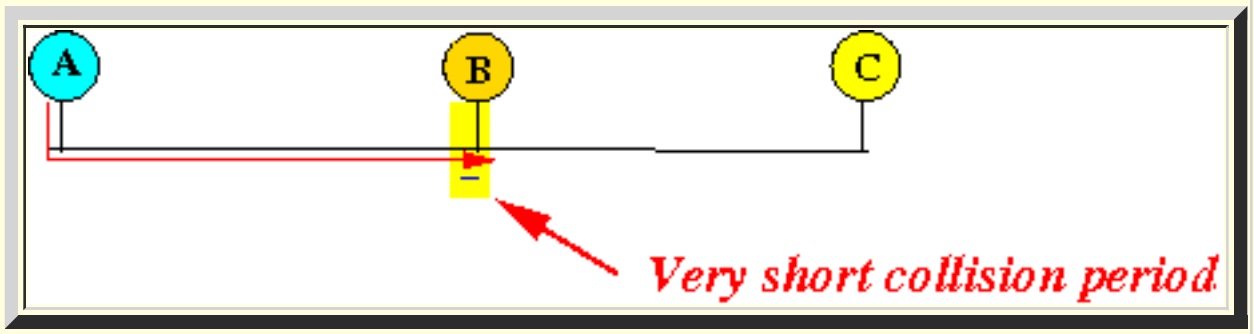
    (The **jam signal** is used to **help** the ***other* transmitting nodes** detect the **collision**)

Example:

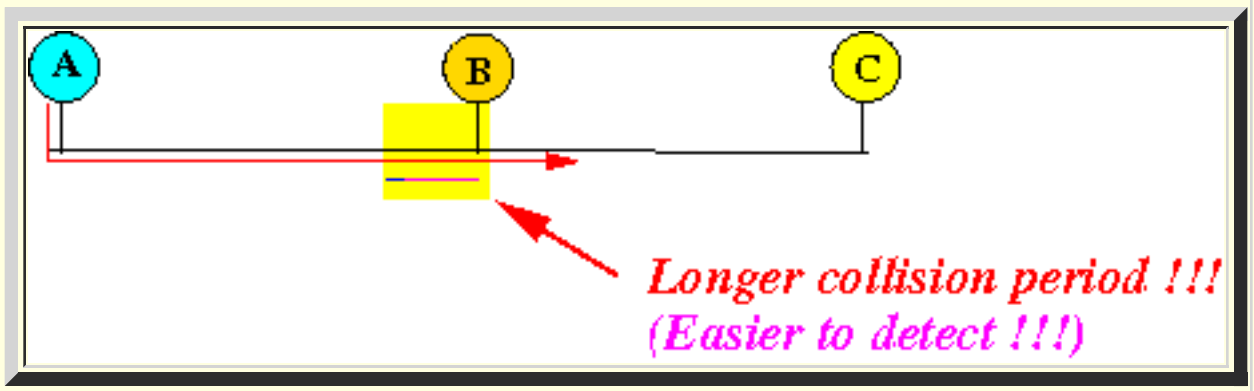
- **Node *B*** starts **just** before ***A*'s transmission** arrives:



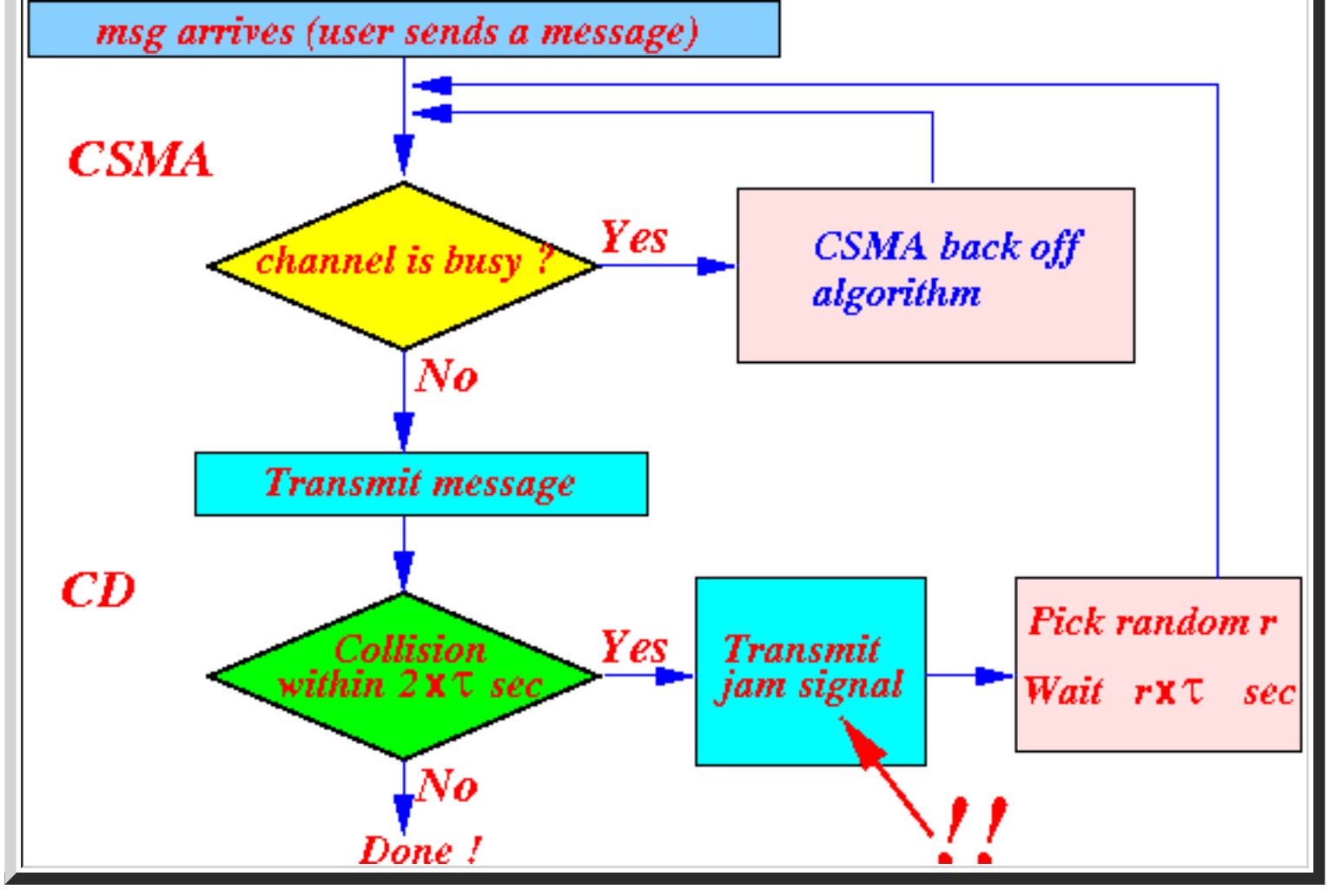
- **Node *B*** has **detected** the **collision**:



- **Node *B*** transmits a (short) **jam signal** to improve **collision detection**:

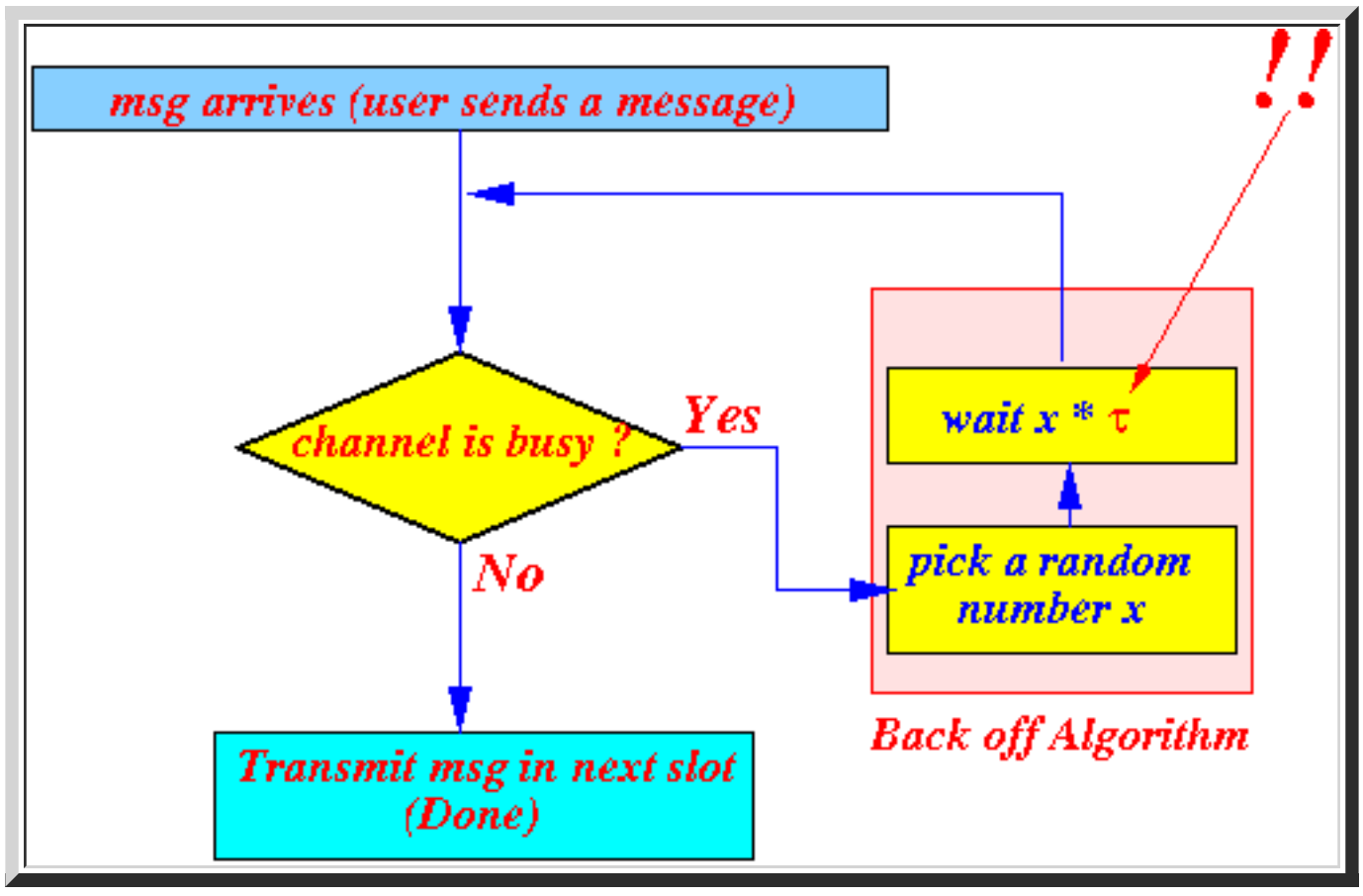


- **The final CSMA/CD procotol (chart)**
  - The **final** **CSMA/CD** flow chart:

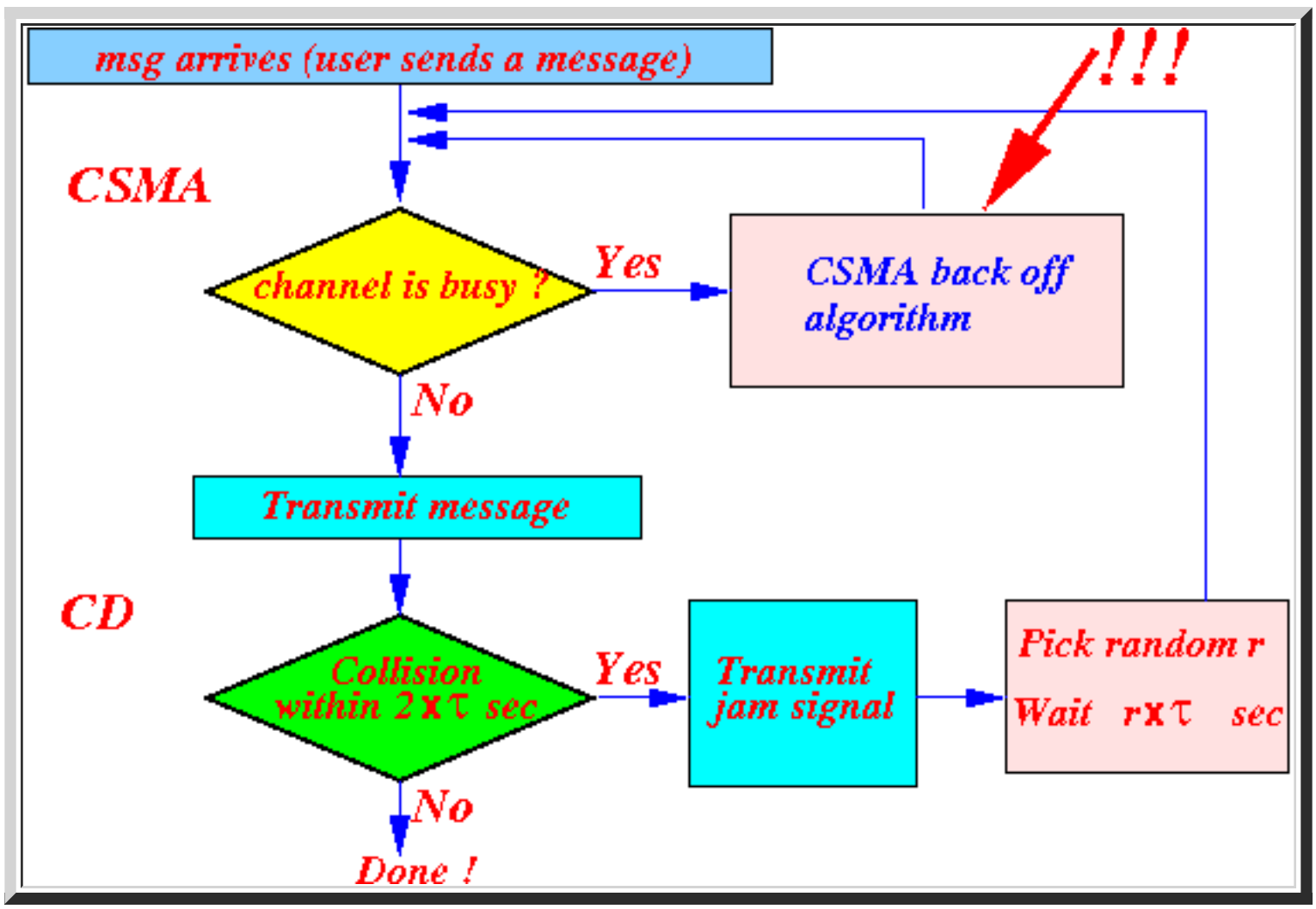

msg arrives (user sends a message)
CSMA
channel is busy ?
Yes
CSMA back off algorithm
No
Transmit message
CD
Collision within 2xτ sec
Yes
Transmit jam signal
Pick random r
Wait rxτ sec
No
Done !
!!

Variants of the CSMA/CD protocols

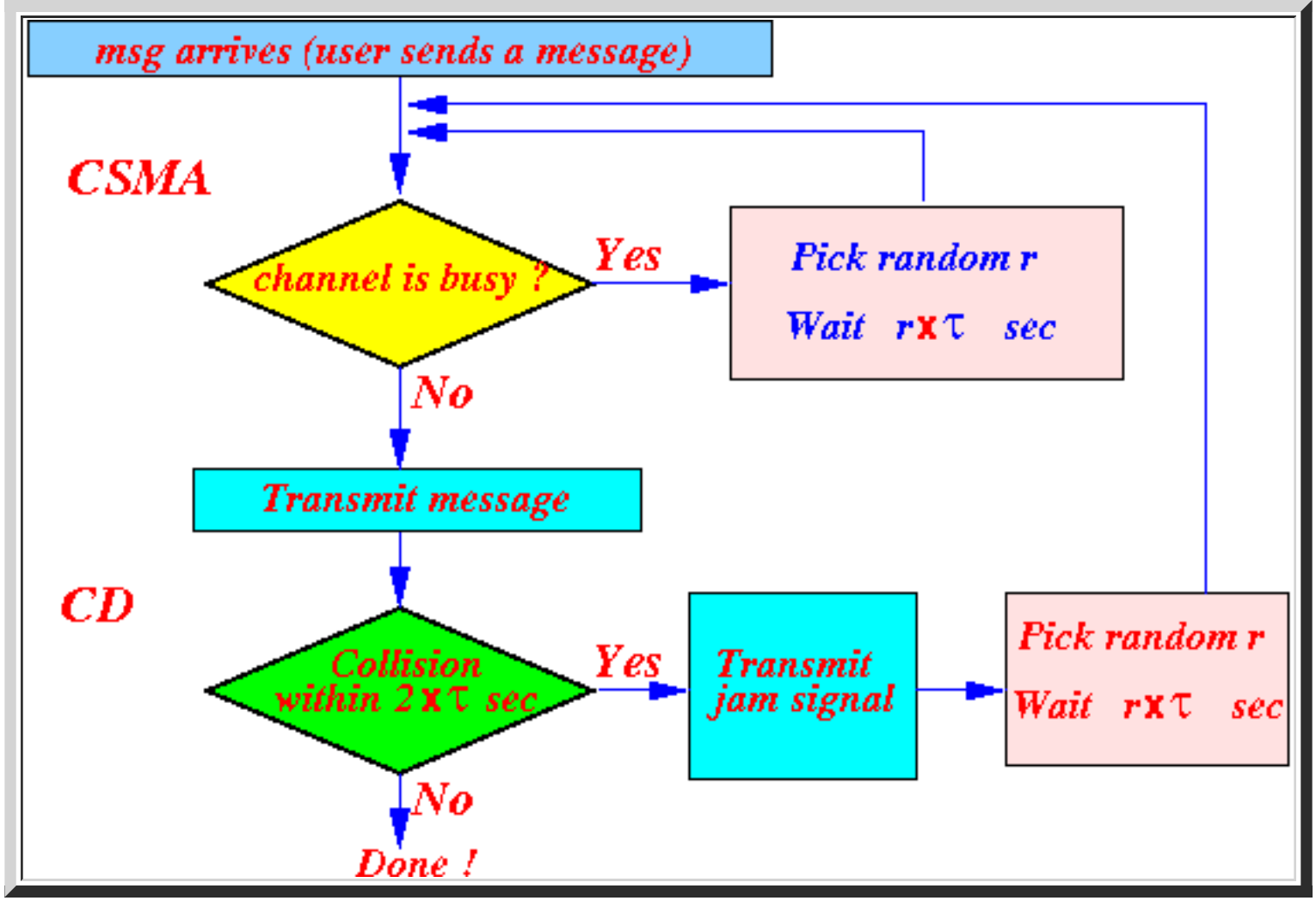
- ***Non*-persistent CSMA/CD:**
  - **Recall** the ***non-persistent*** **CSMA** protocol:



  - When we use the ***non-persistent*** **CSMA backoff algorithm** in the **CSMA/CD** flow chart:



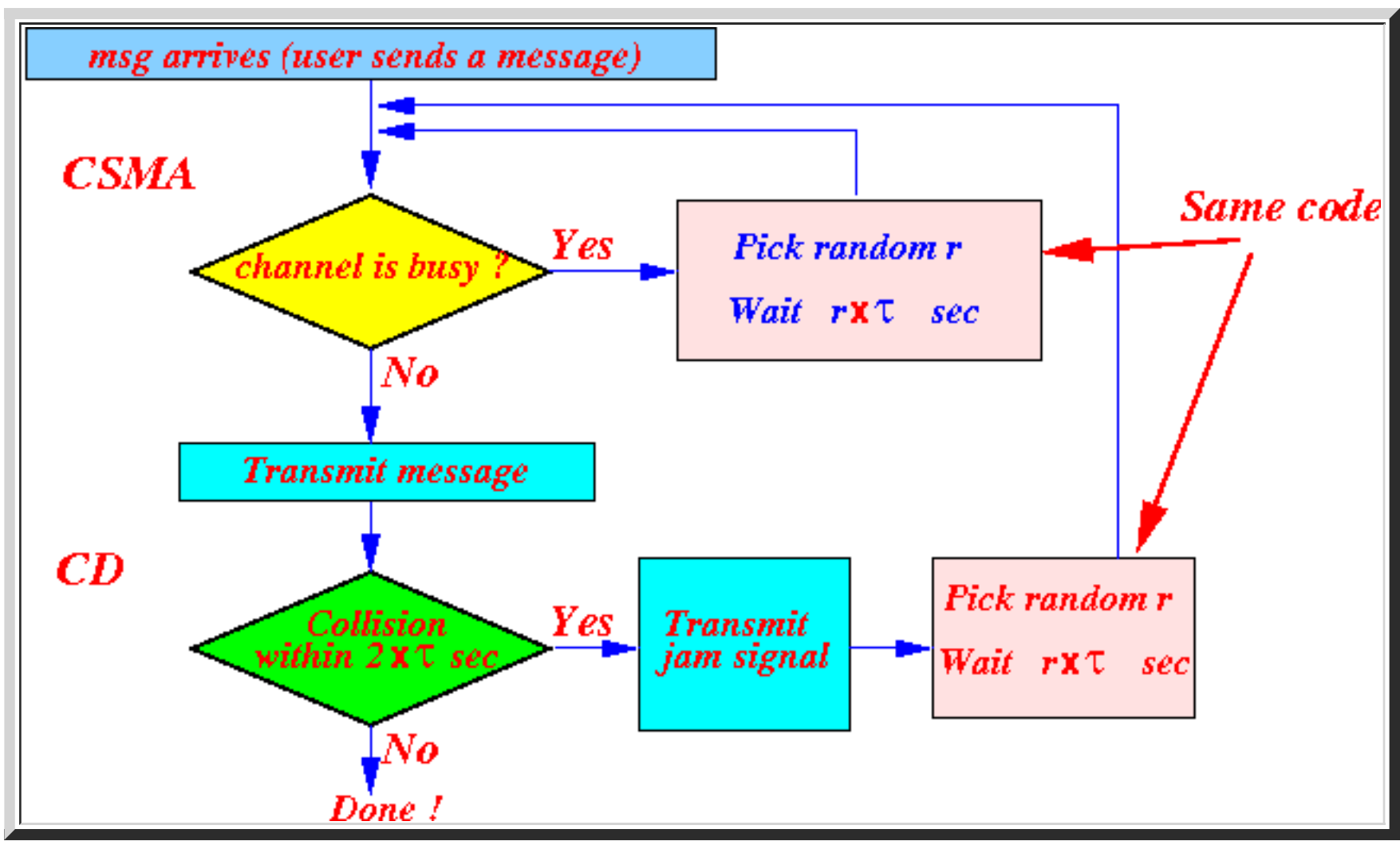
we **obtain** the ***Non-persisten*** **CSMA/CD** protocol:



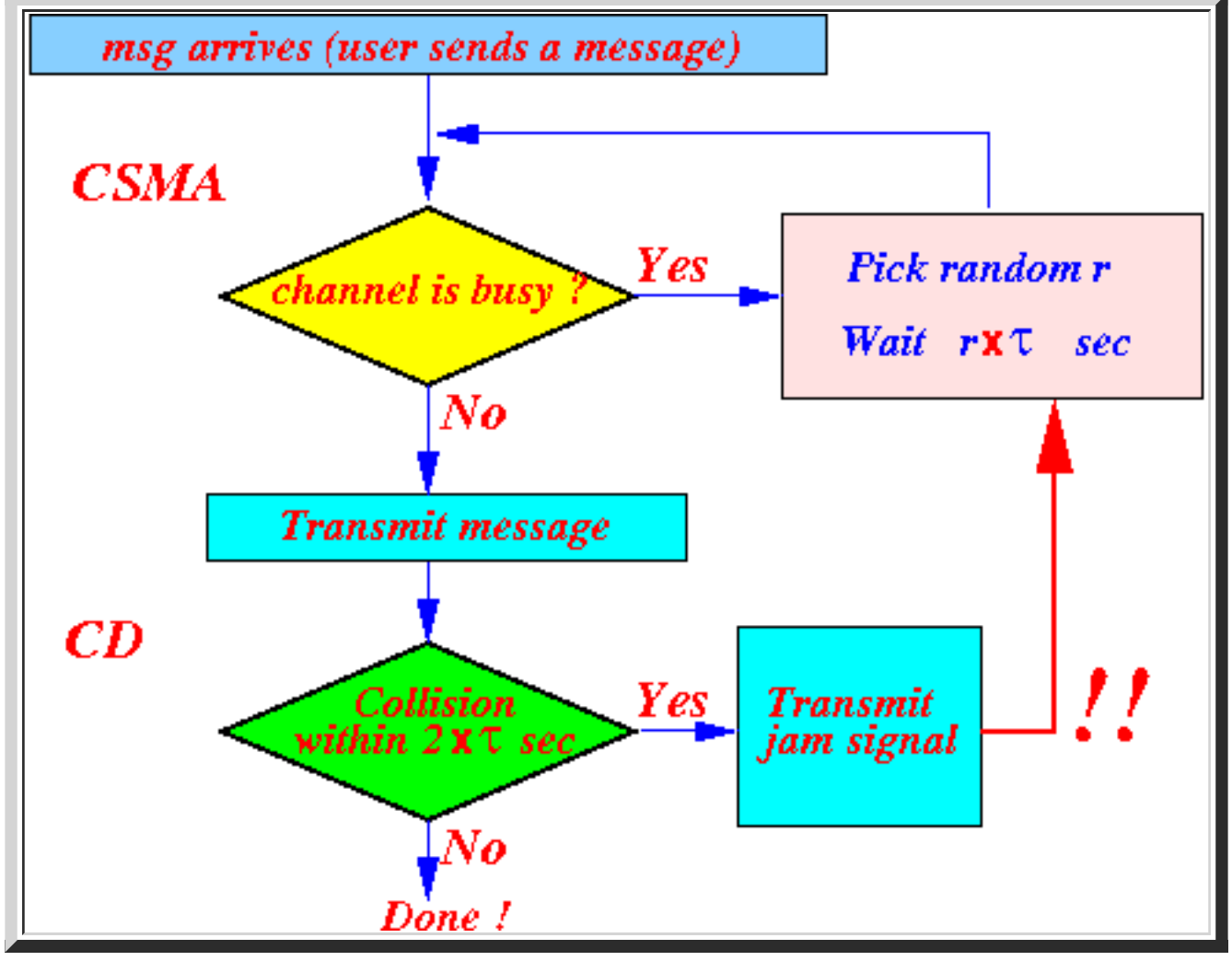
Note:

- τ = the **end-to-end-delay**

- We can **simplify** the ***previous*** **flow chart**:

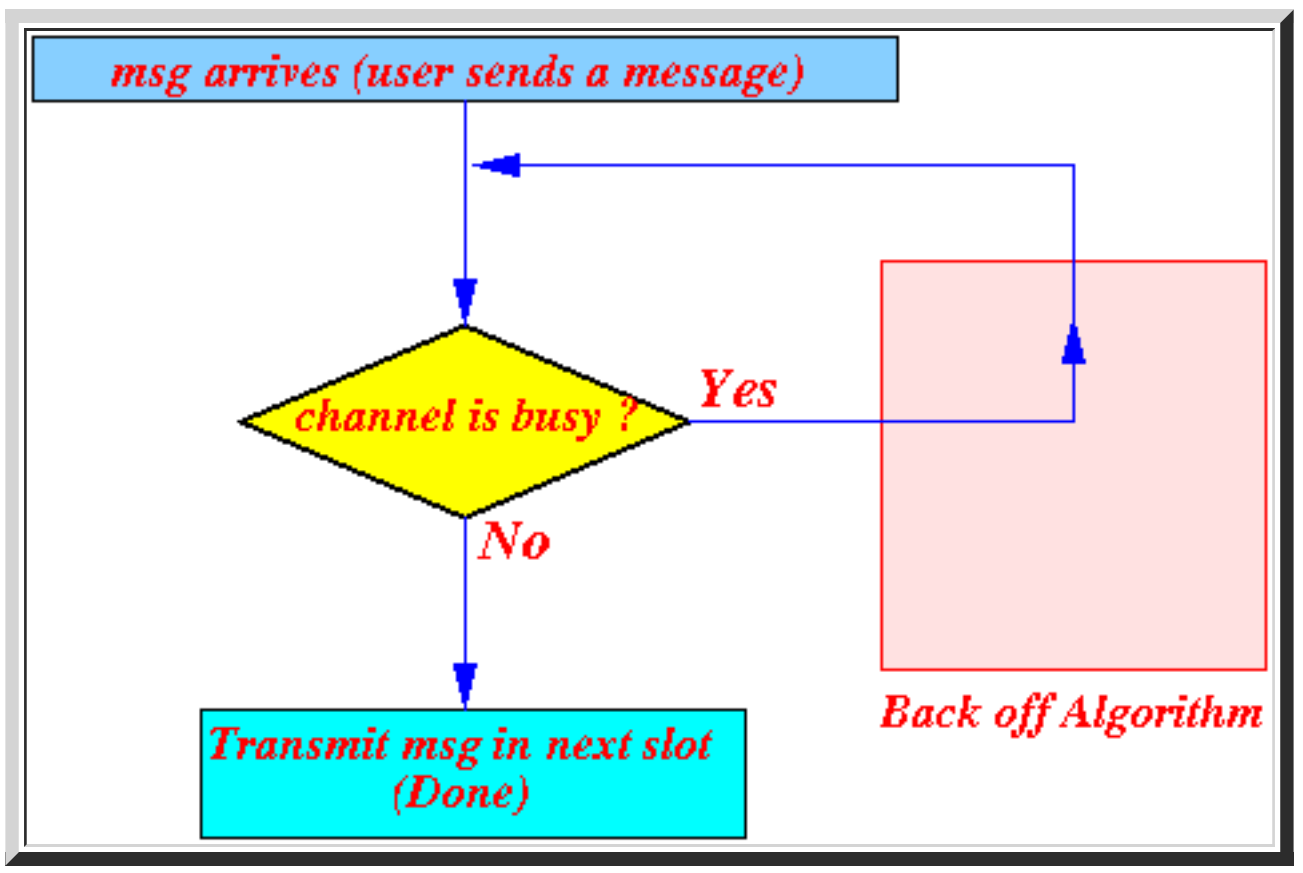


to **this**:

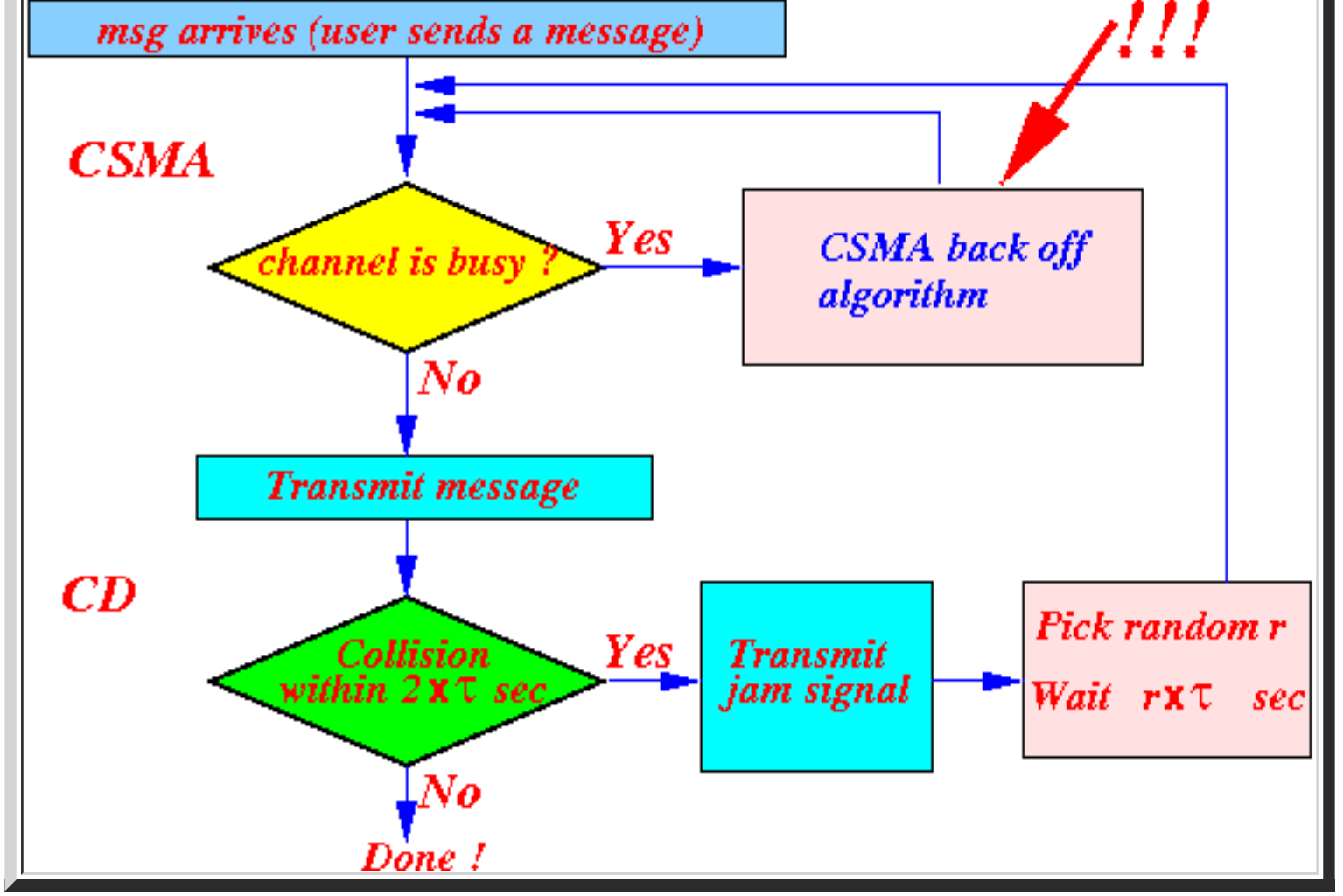


(The ***non-persistent*** **CSMA/CD** protocol)

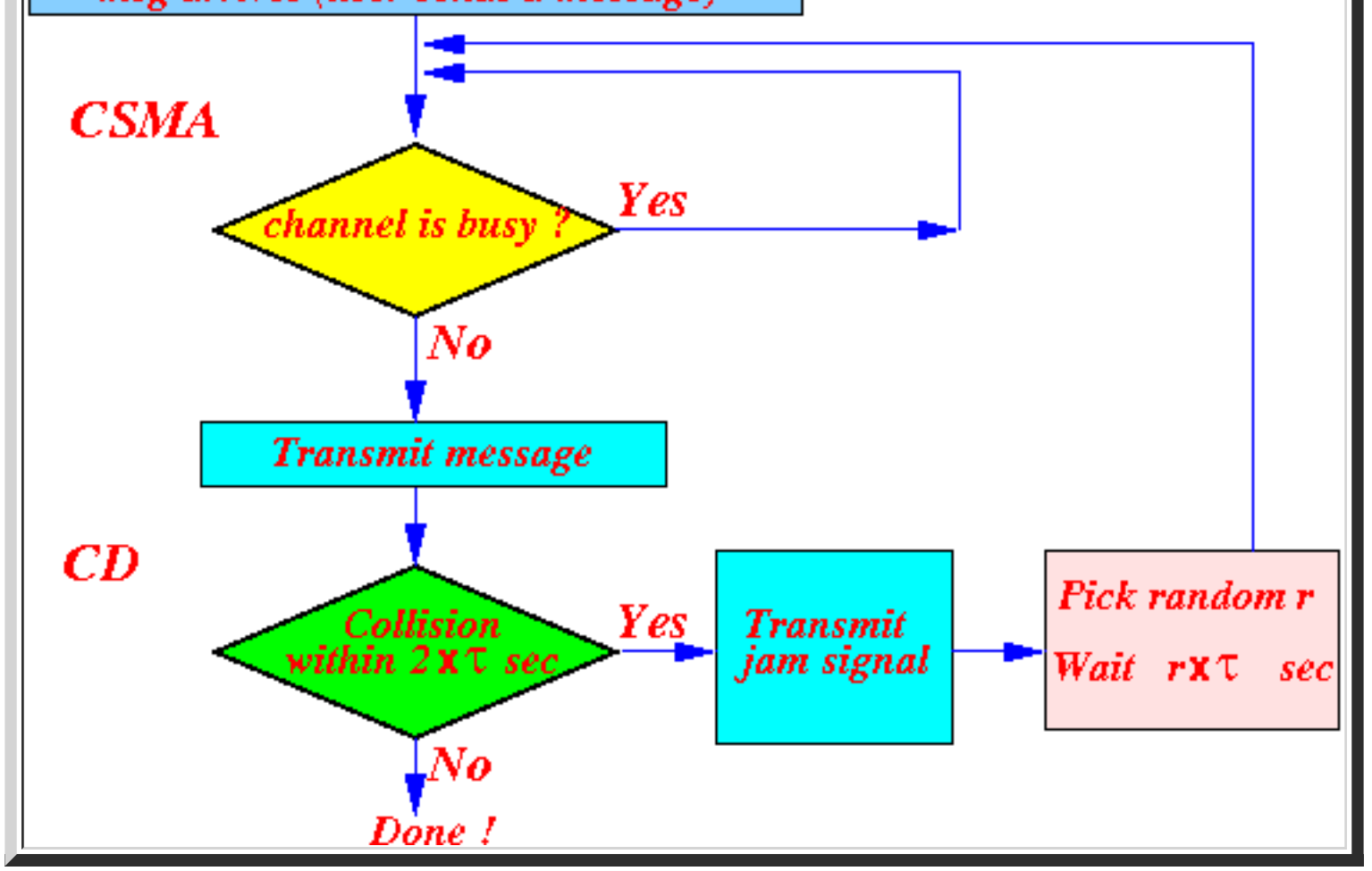
- ***1*-persistent CSMA/CD:**
  - **Recall** the ***1-persistent*** **CSMA** protocol:



  - When we use the ***1-persistent*** **CSMA backoff algorithm** in the **CSMA/CD** flow chart:



we **obtain** the ***1-persisten*** **CSMA/CD** protocol:



Intro to Ethernet

- **The Ethernet**
  - **Overview of the Ethernet:**

    - The ***most* popular network** in the **US**

    - **Ethernet** was developed at **Xerox PARC** between **1973 and 1974**

      (**P**alo **A**lto **R**esearch **C**enter Incorporated is a branch of the **Xerox company**)

    - The **Ethernet protocol** was invented by **Robert Metcalfe**: click here

    - **Metcalfe** was **inspired** by **Aloha** --- which he studied as part of his **PhD dissertation**

  - **Documentation** on the **Ethernet (standard**:

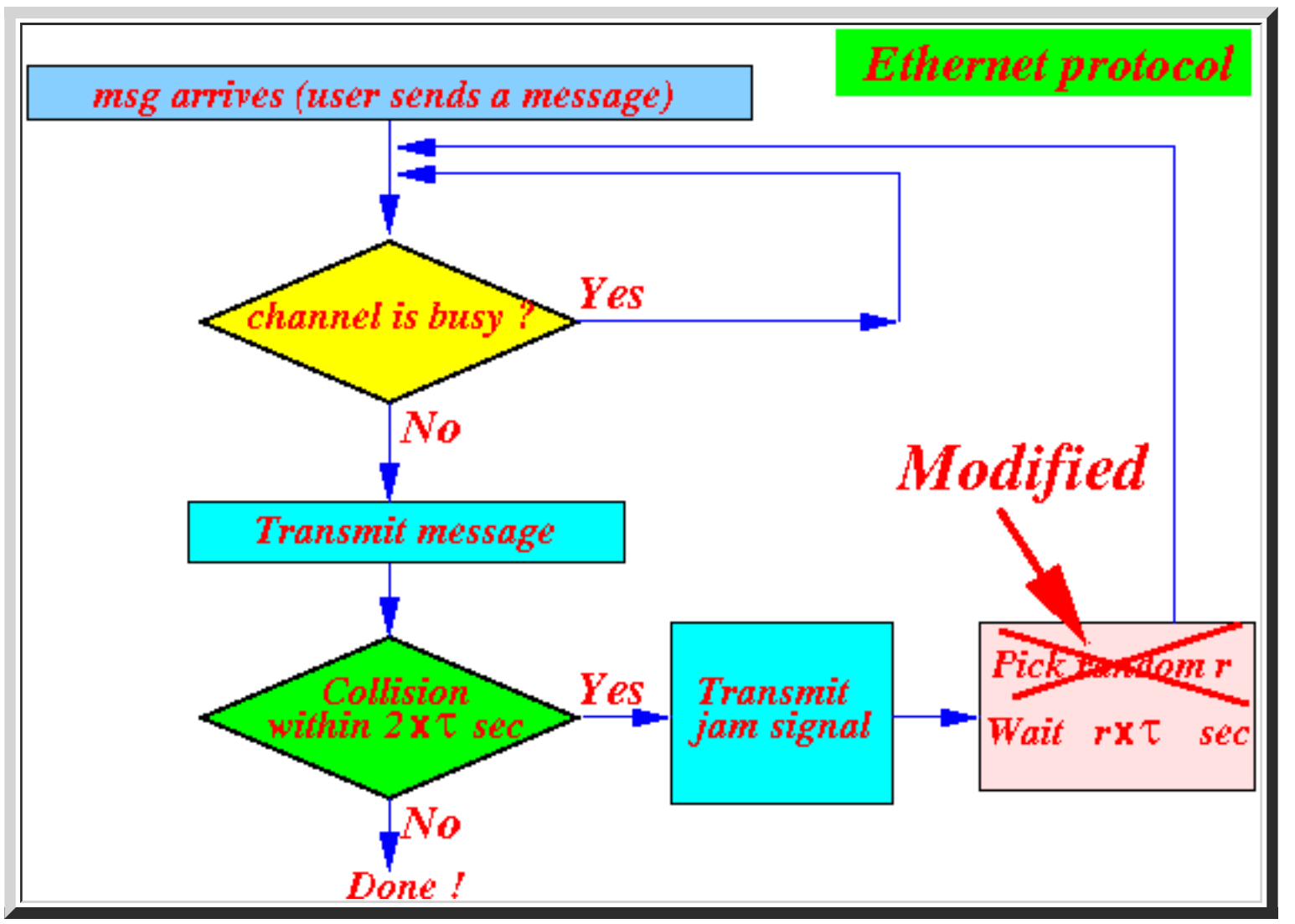
    - "The Ethernet, A Local Area Network. Data Link Layer and Physical Layer Specifications" - **1980** - **click here**

    - **Standardization dates**:

      - **International standard** in **1983**
      - The **IEEE 802.3 standard** on: **June 23, 1983**

- **The Ethernet *Protocol***
  - The **Ethernet protocol** is:
    - A ***modified 1-persistent*** **CSMA/CD protocol !!!**

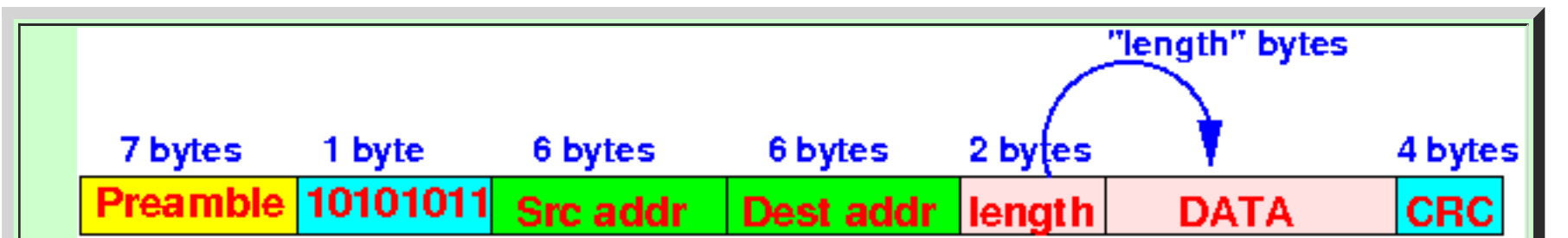
  - The **Ethernet protocol** **flow chart:**



Important **note:**

- **Ethernet** has a ***modified*** **" back off algorithm** which we will **discuss** in the **next web page**

- **The Ethernet *Frame* format**
  - **The Ethernet Frame format**




```
Meaning of the fields:

   Preamble: discussed below
   Start:    10101011           (Fixed pattern)
   Src:      Ethernet address of the sender
   Dest:     Ethernet address of the intended receiver
   Length:   Length of the data field
   Data:     Data field
   CRC:      Check sum
```

- **The *pre-amble***
  - **Pre-amble:**

    - The **pre-amble** is used to ***pad*** the **Ethernet frame** so that the frame is **long *enough*** for **collision *detection*** purposes

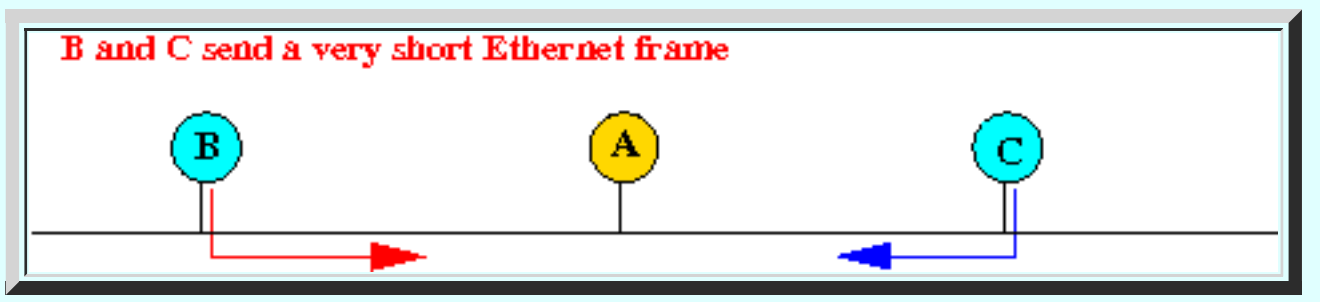
  - **Usage** of the **pre-amble:**

    - A **transmitting node** can ***only*** **detect** a **collision *while*** the **node** is (still) **transmitting**

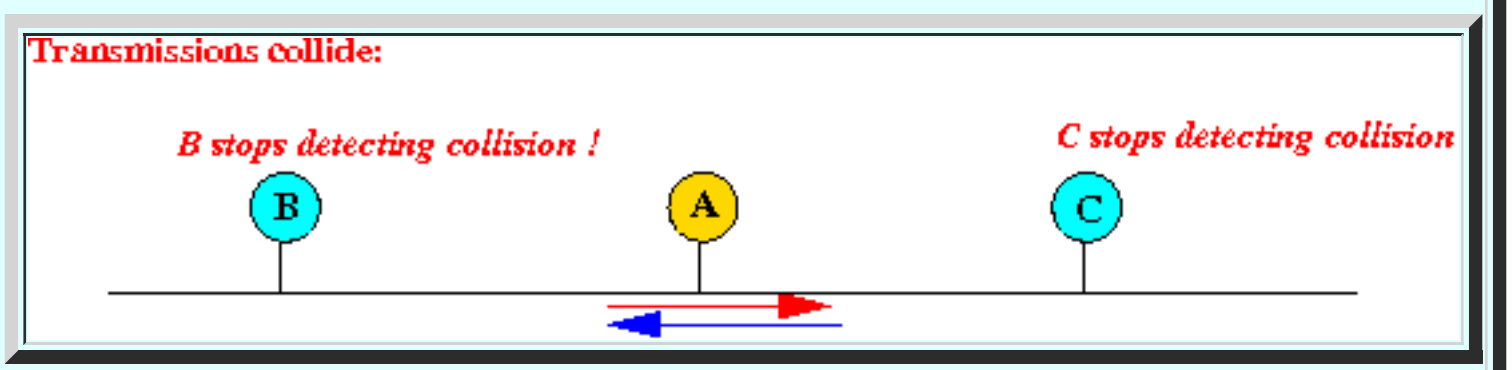
    - The **pre-amble** will **make sure** that ***simultaneous* senders** will transmit **long *enough*** that the ***simultaneous* senders** will **detect a collision**

  - **Example: how nodes** can ***miss*** a **collission**

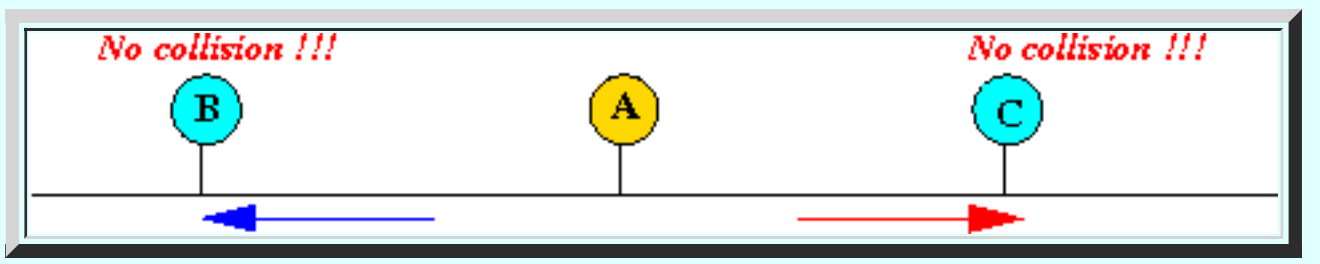
    - **Nodes *B* and *C*** transmit ***very* short frames**:



    - Their **transmissions** will **collide**:



    - **However**: the **sending nodes** (***B*** and ***C***) did **not** detect the **collision**:

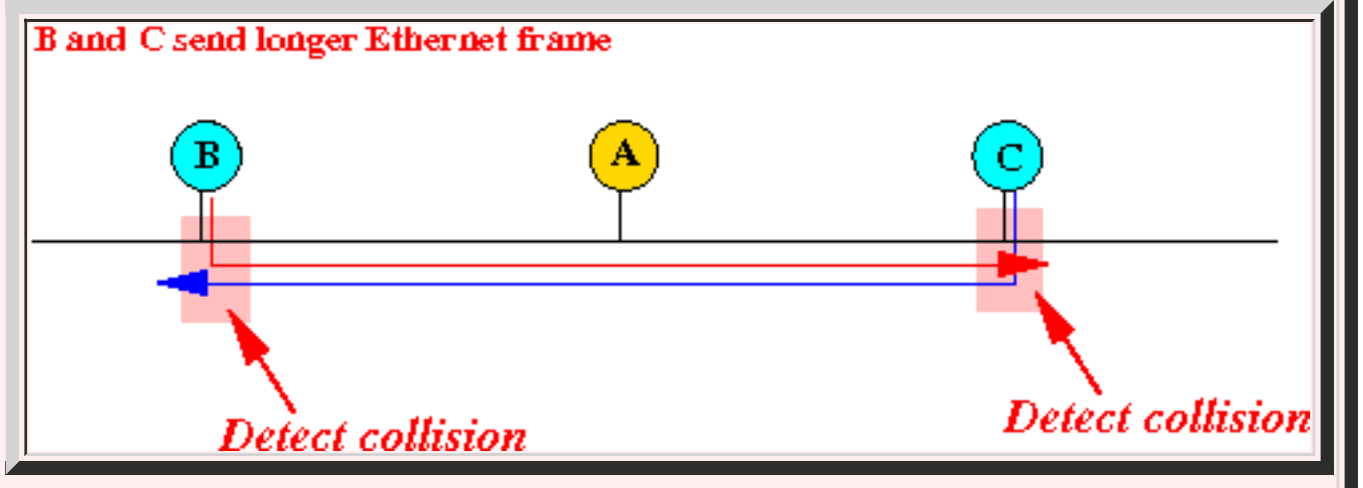


    And:

    - **Nodes *B*** and ***C*** will ***not* retransmit** !!!!

The **pre-amble** will **make sure** that **transmitting node** will be **transmitting *long enough*** to **detect** a **collission**:

- **Nodes *B* and *C*** transmit ***longer* frames**:



and **hears** the **collision**....

- **Ethernet Address**
  - **Ethernet Address:**

    - ***Each*** **nodes** on a **Ethernet** is ***identified*** by a ***unique*** **48 bits Ethernet address**

    - The **Ethernet address** is ***permanently*** **inscribed** in the **Ethernet card** of a **computer**

  - **UNIX command** to find the **Ethernet Address** on a **computer**:

```
cheung@aruba> ifconfig -a
eth0      Link encap:Ethernet  HWaddr 2C:41:38:8B:05:38
          inet addr:170.140.150.36  Bcast:170.140.151.255  Mask:255.255.254.0
          inet6 addr: fe80::2e41:38ff:fe8b:538/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:681868507 errors:0 dropped:0 overruns:0 frame:0
          TX packets:628584348 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:335075333658 (312.0 GiB)  TX bytes:540484678811 (503.3 GiB)
          Interrupt:20 Memory:fb000000-fb020000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:267964 errors:0 dropped:0 overruns:0 frame:0
          TX packets:267964 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:107815976 (102.8 MiB)  TX bytes:107815976 (102.8 MiB)
```

Explanation:

```
Ethernet address = 2C:41:38:8B:05:38        (Hexadcimal digits)

                     2    C    4    1    3    8    8    B    0    5    3    8
                 = 0010 1100 0100 0001 0011 1000 1000 1011 0000 0101 0011 1000
```

- **Why are there *no* send/receiver sequence numbers ???**

  Notice that:

    - **Ethernet frame:**

There are:

- ***No Send*** **sequence number**
- ***No ACK*** **sequence number**

in **Ethernet frames** !!!

- ***Reason:*** (it **confirms** what I **said before** --- see: **click here**)

  - The **Ethernet (LAN) protocol** **does *not*** provide **reliability** function !!!

  - **Note:**

    - **Ethernet** relies on the ***Transport*** **Layer** to ensure **reliable communication**

      - There is an **ARQ protocol** in the ***Transport*** **layer** !!!

- **Postscript: "2 Ethernets" --- the DIX Standard and the IEEE 802.3 Standard**

  - **Ethernet:**

    - **"Ethernet" originally** referred to the protocol implementation standardized by **Digital, Intel and Xerox**.

    - This **standard** is **now** known as the **DIX standard**.

  - **IEEE Standard for "Ethernet"**

    - The **IEEE 802.3 protocol standard** describes **another** **1-persistent CSMA/CD protocol** that is **very close** to the **DIX Ethernet standard**.

    - **DIX Ethernet** and **802.3** differ slightly in the **frame format** and are in **most aspects identical**.

      - The ***only*** **difference** is the **length/EtherType field**

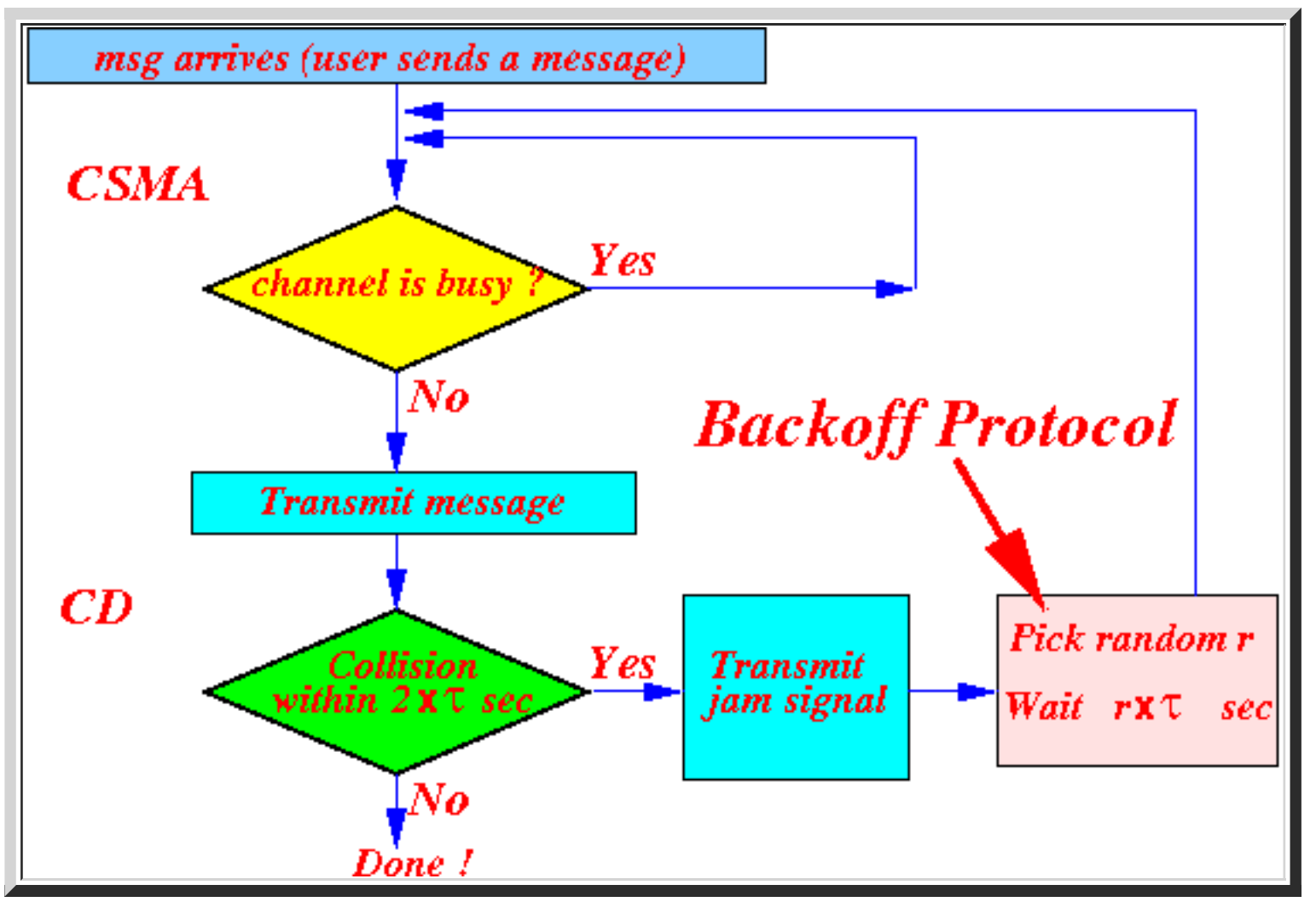
        See: **click here**

- **Today,** the term **"Ethernet"** refers generically to **both**:

  - the **DIX Ethernet implementation** and
  - the **IEEE 802.3 standard**.

The *Expontial* Backoff Algorithm of the Ethernet

- **The Backoff Protocol**
  - **Recall** the **Ethernet CSMA/CD protocol**:



Observe that:

> - The **backoff protocol** is **executed** when a **node** **have (already) detected** a **collision** !!!!

Therefore:

> - ***Multiple*** **nodes** are ***ready*** to **transmit** !!!
>
> (***One*** **transmission** can ***never*** result in a **collission**.....)

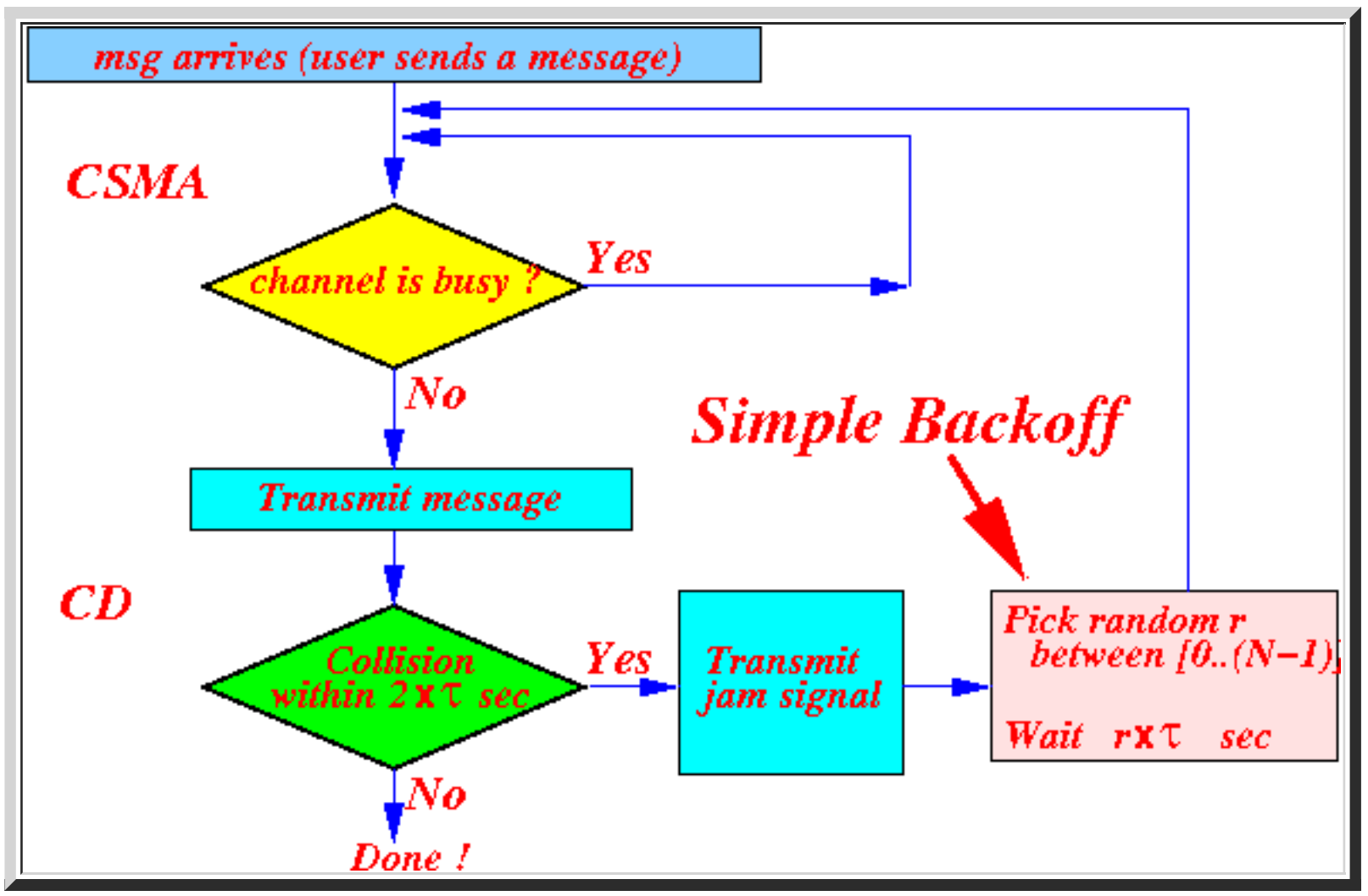
  - ***Goal*** of the **Backoff protocol**:

> - ***Re-schedule*** the **transmissions** of the ***collided*** **nodes** so that:
>
> > - the **likelihood (chance)** of ***subsequent*** **collisions** is **minimized**.

- **A *simple-minded* backoff protocol**
  - **A *simple-minded* backoff protocol:**

- **Pick** a **random number** ***r*** between **[0..(N−1)]** (**N** is **fixed**)
- **Wait** $r \times \tau$ time

Flow chart:



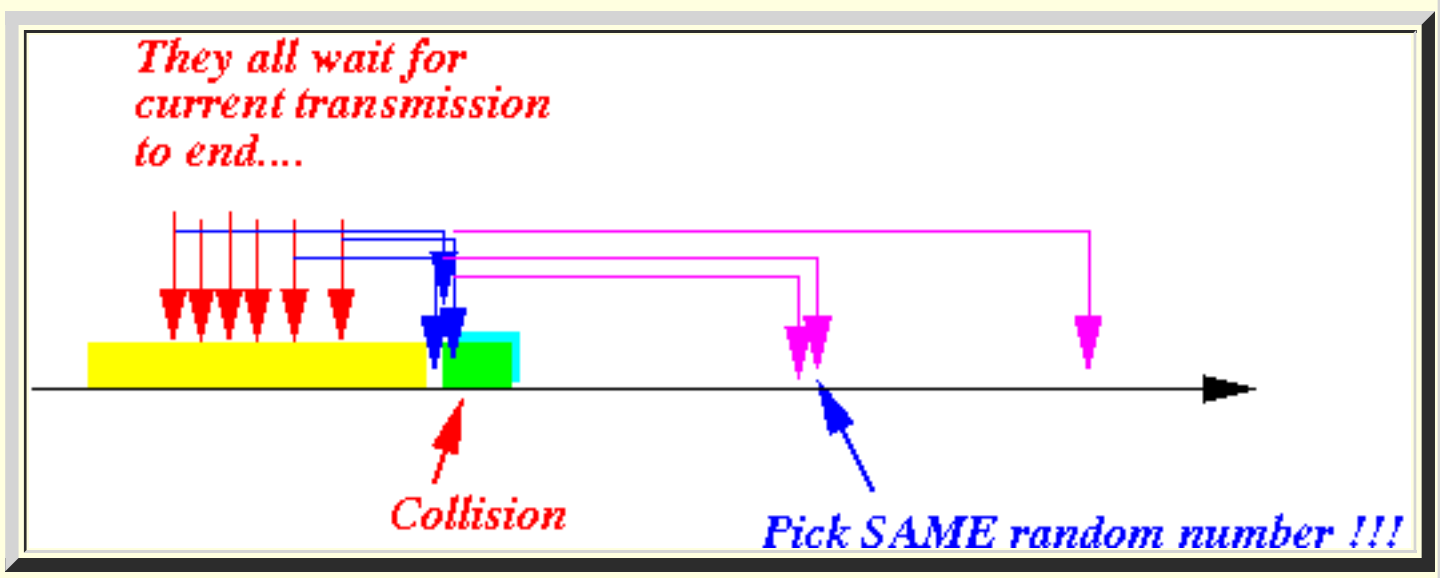
- **Weakness** of the ***simple-minded*** **backoff** algorithm:

  - If ***N*** is ***small***, and the **system** is **very busy**

    (= ***many*** **nodes** have ***collided***),

    then:

    - It will be ***very*** **likely** that ***more*** **than one node** will **select** the ***same*** **random number**

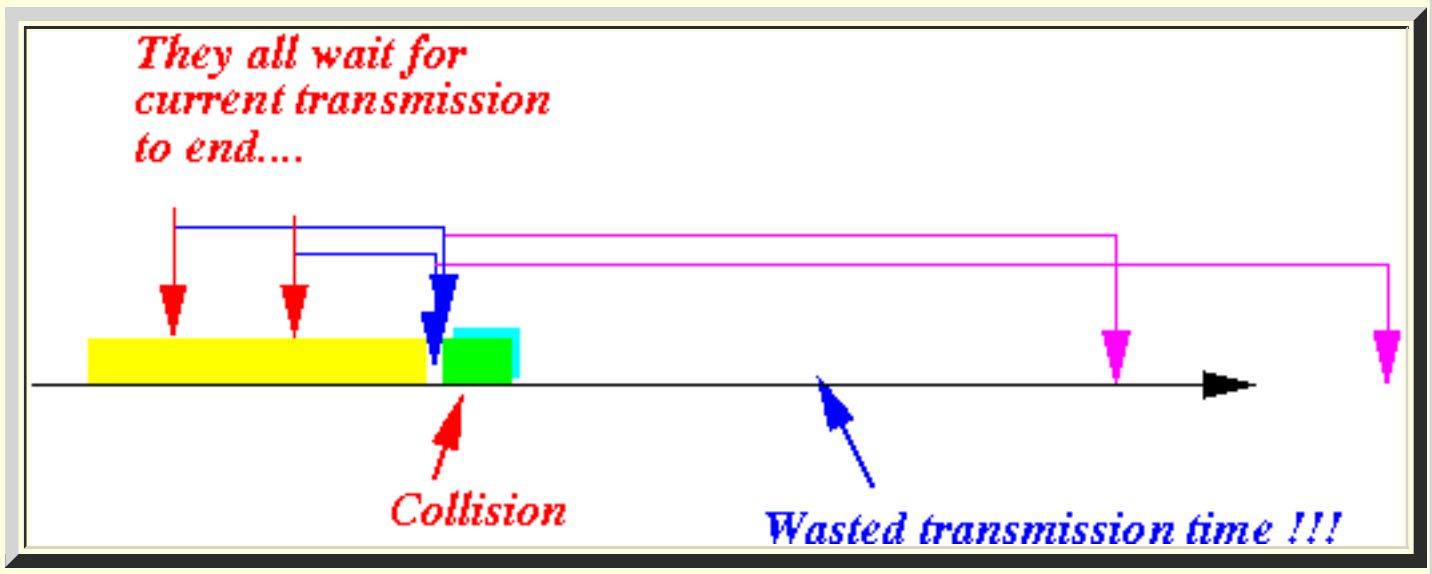


    - **Result:**

      - ***frequent*** **collisions**

- **On the other hand**, if ***N*** is ***large***, but the **system** is ***lightly*** **loaded**, **then**:

  - Nodes ***may*** select an ***large*** random value



  Result:

  - The **node** will **back off** for a ***long*** **time** (for nothing)....

  - **Result:**

    - **Wasting** of **bandwidth**.....

- **Conclusion:**

  - We need a ***dynamic*** **backoff algorithm** that can ***adjust*** to ***different*** **network operating conditions** !!!!

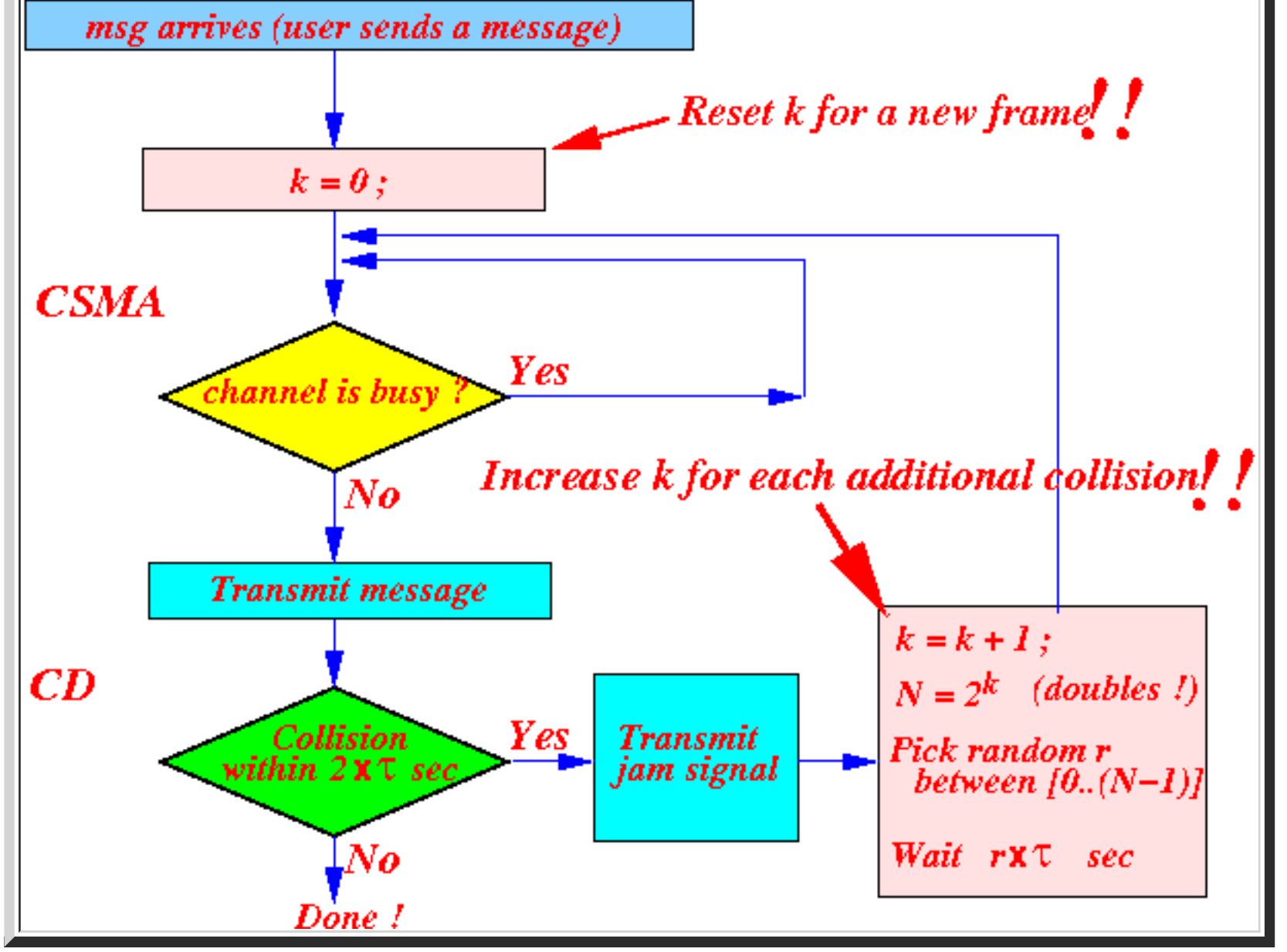
- **The Exponential Backoff Protocol**

  - ***Adaptive*** **back off**:

    - Allow **collided nodes** to pick ***small*** **back off** periods in ***earlier*** **re-transmission attempts**

    - And pick ***larger*** **back off** periods when there are ***more*** **re-transmission attempts**

  - The **exponential backoff algorithm** **flow chart**:



- **Explanation:**

- When a **frame** is **transmitted** for the ***first*** **time**:

```
Reset  k = 0
```

- ***If*** there was a **collision**, the ***first*** **retransmission attempt** will use:

```
k = k + 1  = 0 + 1  = 1
N = 2^k - 1 = 2^1 - 1 = 2 - 1 = 1
```

For the ***first*** **re-transmission attempt**, the **node** will picks a **random number** ***r*** from the range:

- $r \in [0, 1]$

This **range** is **suitable** when there are ***very few*** **(about 2) nodes** involved in the **collision**

- It the **node** is **involved** in ***another*** **collision** while trying to **transmit** the ***same*** **frame**, **then**:

```
k = k + 1  = 1 + 1  = 2
N = 2^k - 1 = 2^2 - 1 = 4 - 1 = 3
```

So the ***second*** **retransmission attempt** for the ***same*** **frame** will use a **random number** ***r*** from the range:

- $r \in [0..3]$

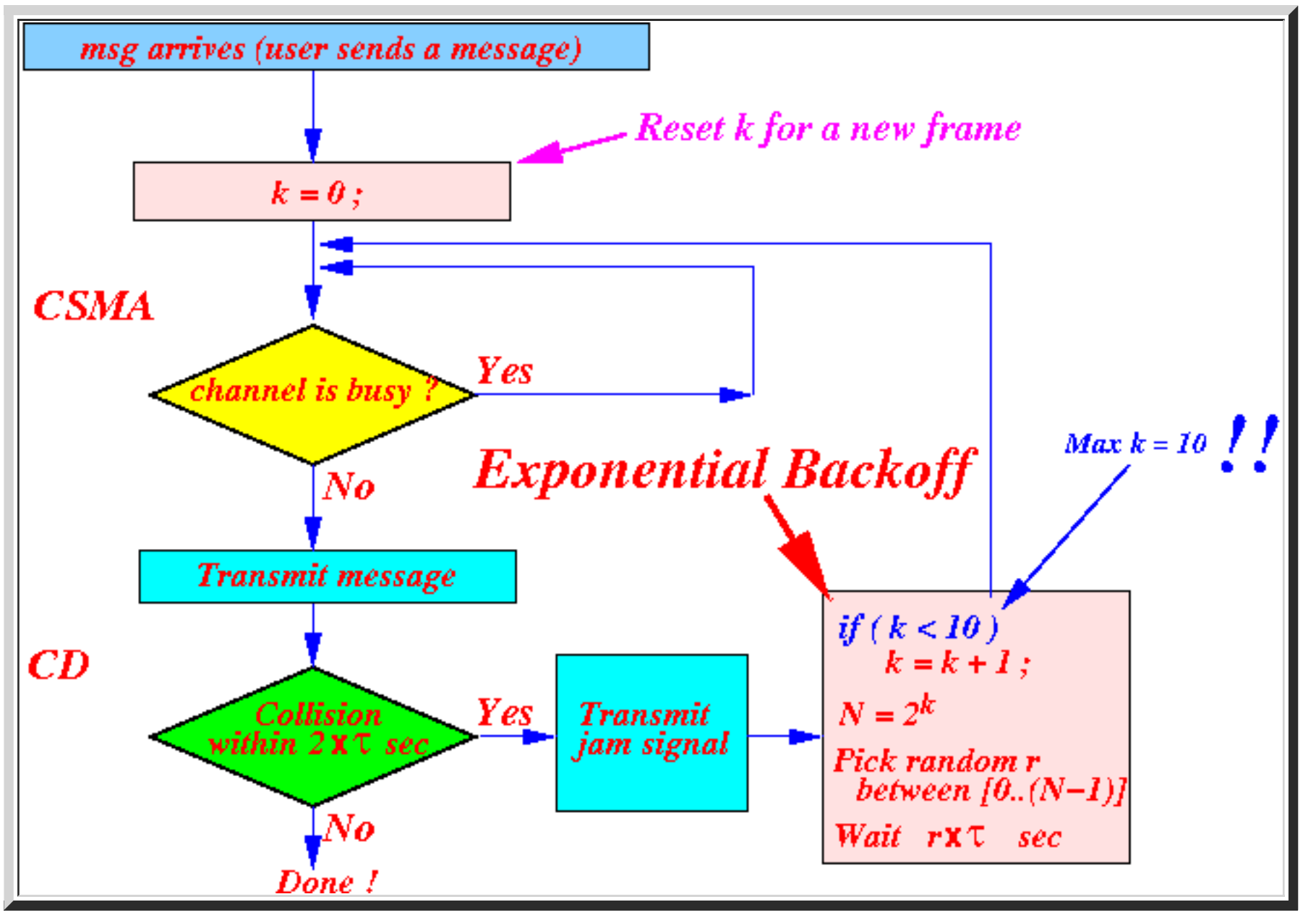
The **node** is ***adjusting*** to **accomodate** for a ***larger*** **number** of **nodes** (i.e., a ***heavier loaded***

situation)

- If there is **another collision**, the ***third*** **retransmission attempt** of the ***same*** **frame** will use:
  - $r \in [0..7]$
- **And so on...**

- **A *small* adjustment: upper bound on random range**
  - **Fact:**
    - **One Ethernet network** can **support** at most **500 nodes**

    Therefore:
    - The **maximum range** from which the **random numbers** are selected is:
      - $[0..2^{10}-1] = [0..1023]$
  - The ***final*** version of the **Ethernet Medium Access Control (MAC)** algorithm:



The *unfairness* of the Ethernet Backoff Algorithm

- **Channel capture effect**
  - **Channel capture effect:**

    - **Channel capture effect** = a **phenomenon** where **one user** of a ***shared*** **(= broadcast) medium** ***"captures"*** **(= hogs)** the **medium** for a **significant time**.

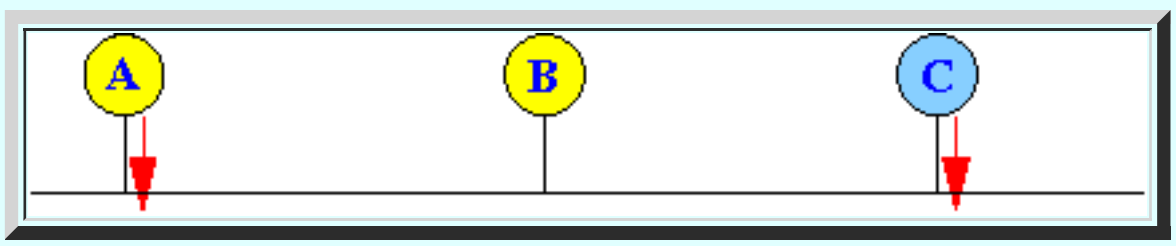
      Wikipedia: click here

- ***Unfairness*** **of the Ethernet's backoff algorithm**
  - **Suppose:**

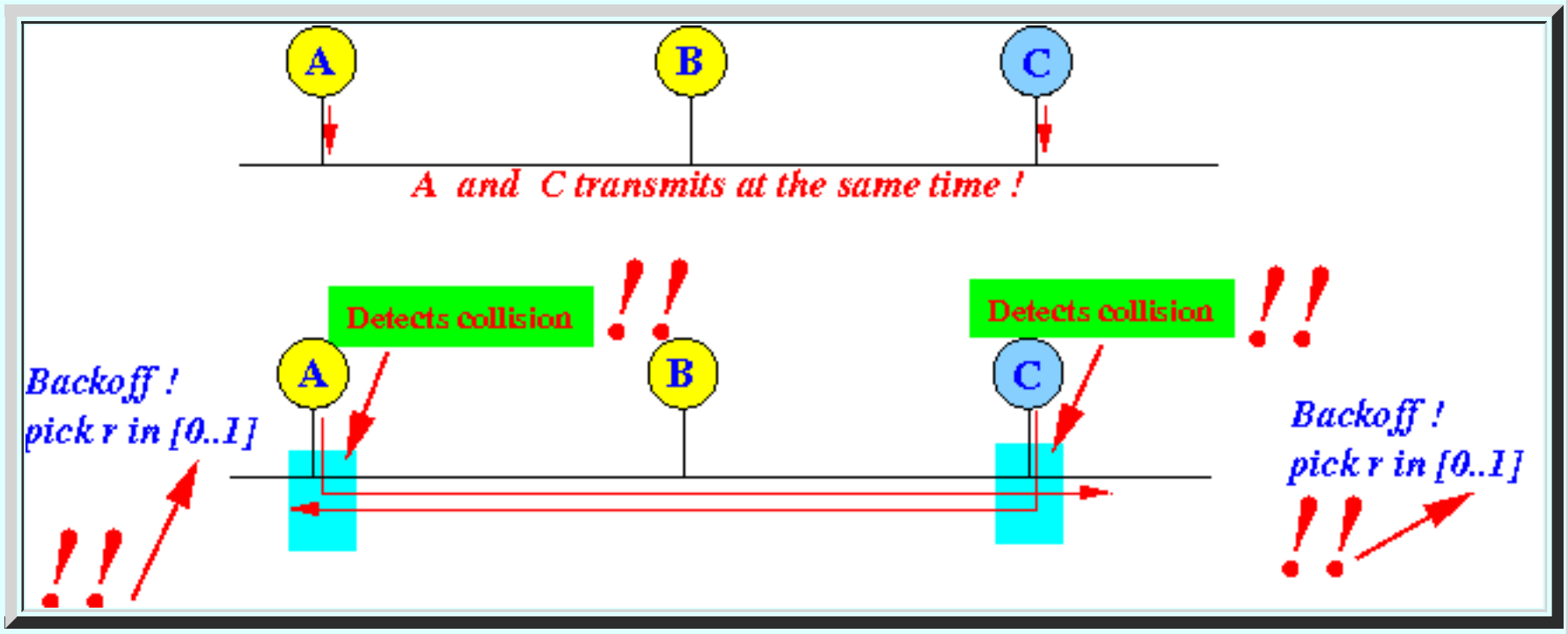
    - The **nodes *A*** and ***C*** in an **Ethernet** has a ***large*** **number** of **frames** to **transmit**

  - **Consider** the following **scenario**:

    - **Nodes *A*** and ***C*** **transmits** at the ***same*** **time**:

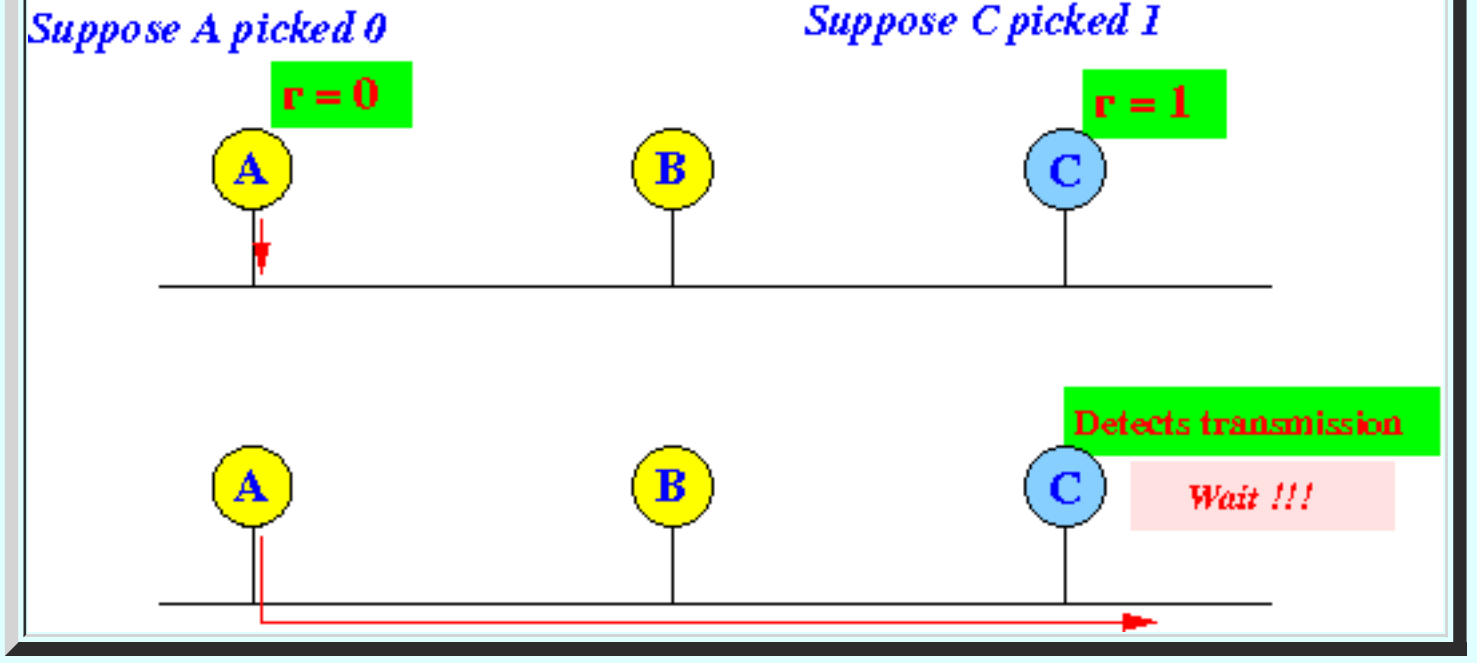
    - The **transmissions** of **nodes *A*** and ***C*** will **collide** and they will **back off**

    Suppose:

    - Node ***A*** picked **0**
    - Node ***C*** picked **1**

    Then node *A* will **transmits *first***:

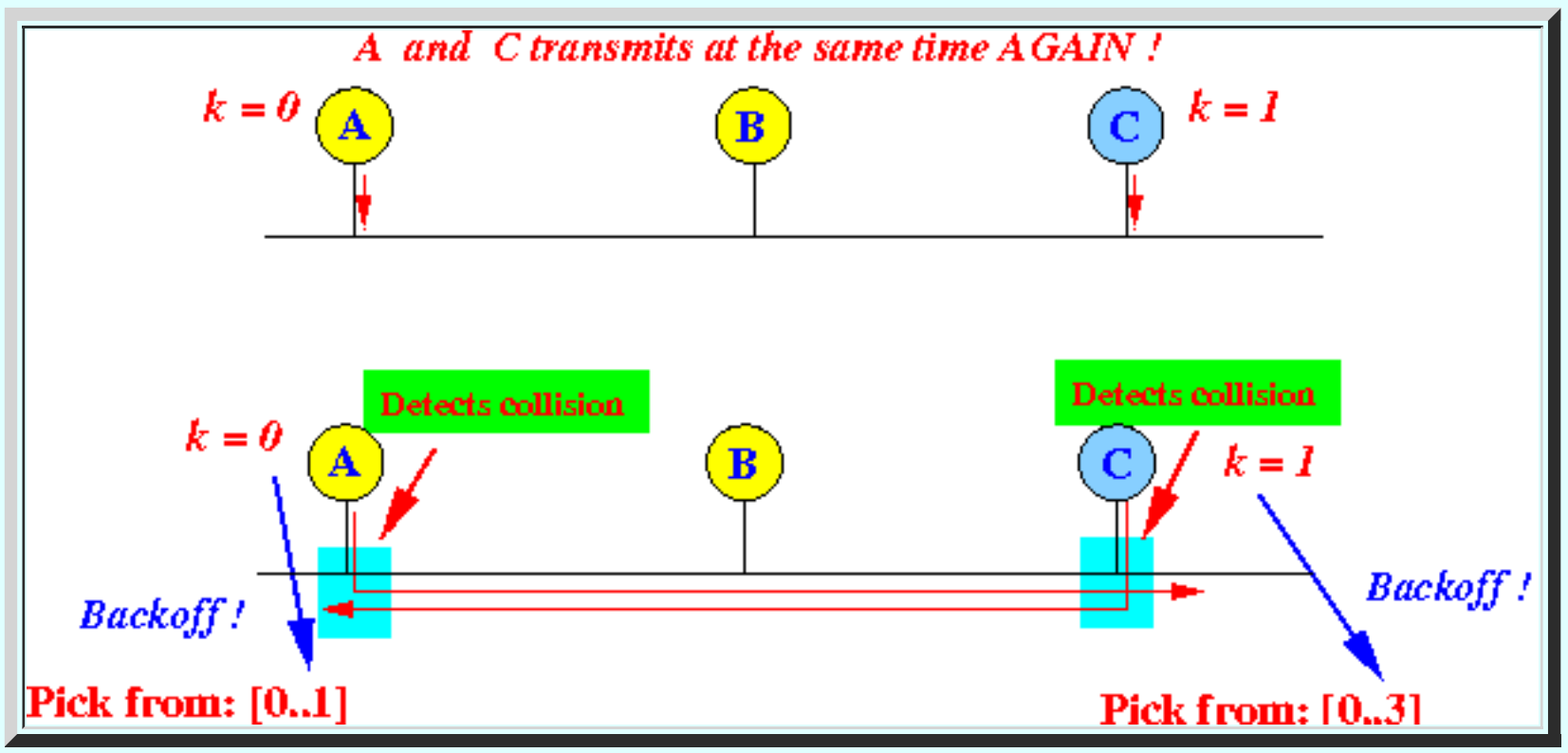


Node *A* will **complete** its **transmission**

- When node ***A*** is **done**, ***A*** will **transmit *again***

  The node ***C*** will ***also* transmit**.

  Their **transmissions collide *again*** and **back off**:



Notice that:

- **Node *A*** transmits the **frame** for the ***first* time**, so:
  - Node ***A*** will pick a **random number** from the range **range [0..1]**

- **Node *C*** transmits the **frame** for the ***second* time**, so:
  - Node ***C*** will pick a **random number** from the range **range [0..3]**

- **Problem:**
  - The ***likelihood*** that **node *A*** will **transmit *before*** the **node *C*** is ***higher*** !!!
    - The **probability** is **calculated** in the **section below**....

- This is ***unfair*** for **node *C*** !!!

- **Unfairness calculation....**
  - **Question:**

    - If **node *A*** picks from **[0..1]** and **node *C*** picks from **[0..3]**, then:
      - What is the **probability** that node ***A*** will transmit ***before*** node ***C*** ?

Pre-question: ***when*** will **node *A*** transmit **before node *C***:

- Node ***A*** transmits **before** node ***C*** if:
  - Node ***A*** picks a ***smaller* (random) value** than node ***C***

Answer:

- ***A*** will transmit ***before C*** when:
  - ***A* picks 0** and ***C* picks 1**
  - ***A* picks 0** and ***C* picks 2**
  - ***A* picks 0** and ***C* picks 3**
  - ***A* picks 1** and ***C* picks 2**
  - ***A* picks 1** and ***C* picks 3**
- The **probability** that **C** will pick **any one number** is **0.25**

  The **probability** that **B** will pick **any one number 0.5**

- The chance that **C** will transmit **before B** is:
  - **(0.25 × 0.5) + (0.25 × 0.5) + (0.25 × 0.5) + (0.25 × 0.5) + (0.25 × 0.5) = 0.625**

- **Summary:**
  - If **node *A*** picks from **[0..1]** and **node *C*** picks from **[0..3]**, then:
    - The **probability** that node **A** transmits ***before*** node **C** = **0.625**
    - The **probability** that node **C** will transmit ***before*** node **A** = **0.125**

Conclusion:

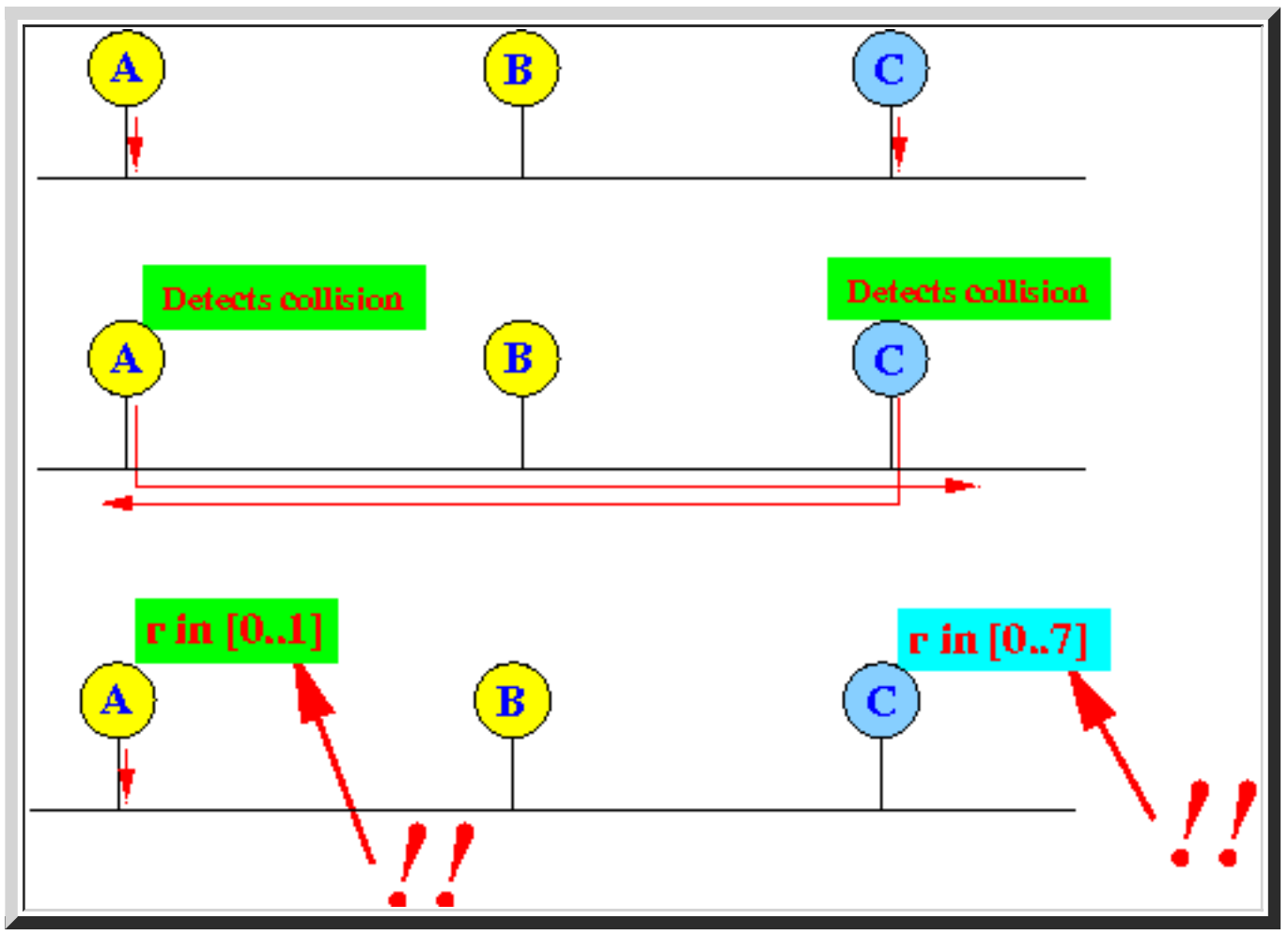
- This is ***not*** **fair** !!!

- And the **bad news** is:

  - It will get **even worse** if **C** loses **again** ...

  Suppose node **B** transmits first and during **B's transmission** the node **A**, scheduled a transmission.

  Then:



  The **likelyhood** that **node C** will **transmit first** is now **very very** ***smaller*** !!!

- **NOTE:**

  - This **problem** is **very well-known**, and it is called the:

    - **"Ethernet Capture Effect"**

Contention-free (Token-based) Medium Access Protocols (MAC)

- **Contention-free protocols**
  - **Contention-free medium access protocols**:

    - **Network access** is ***regulated*** by a ***special*** **frame**:
      - **Token frame**

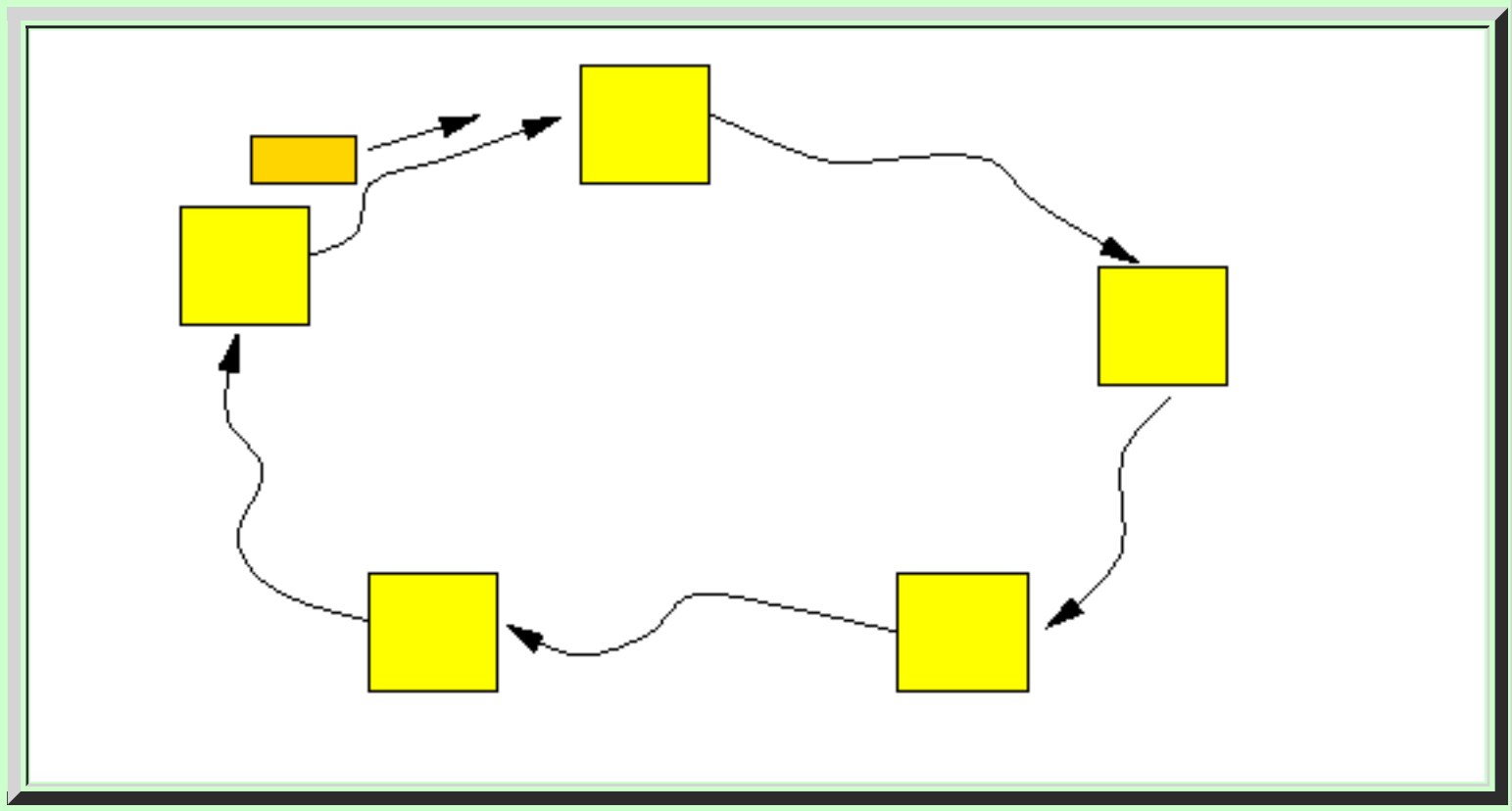
    - A **node** is ***only*** allowed to **transmit** a **data frame (message)** when:
      - The **node** has **received** the ***token*** **frame**

- **Token-based protocols**
  - There are **2 standards** for **token-based (contention-free) protocols**:

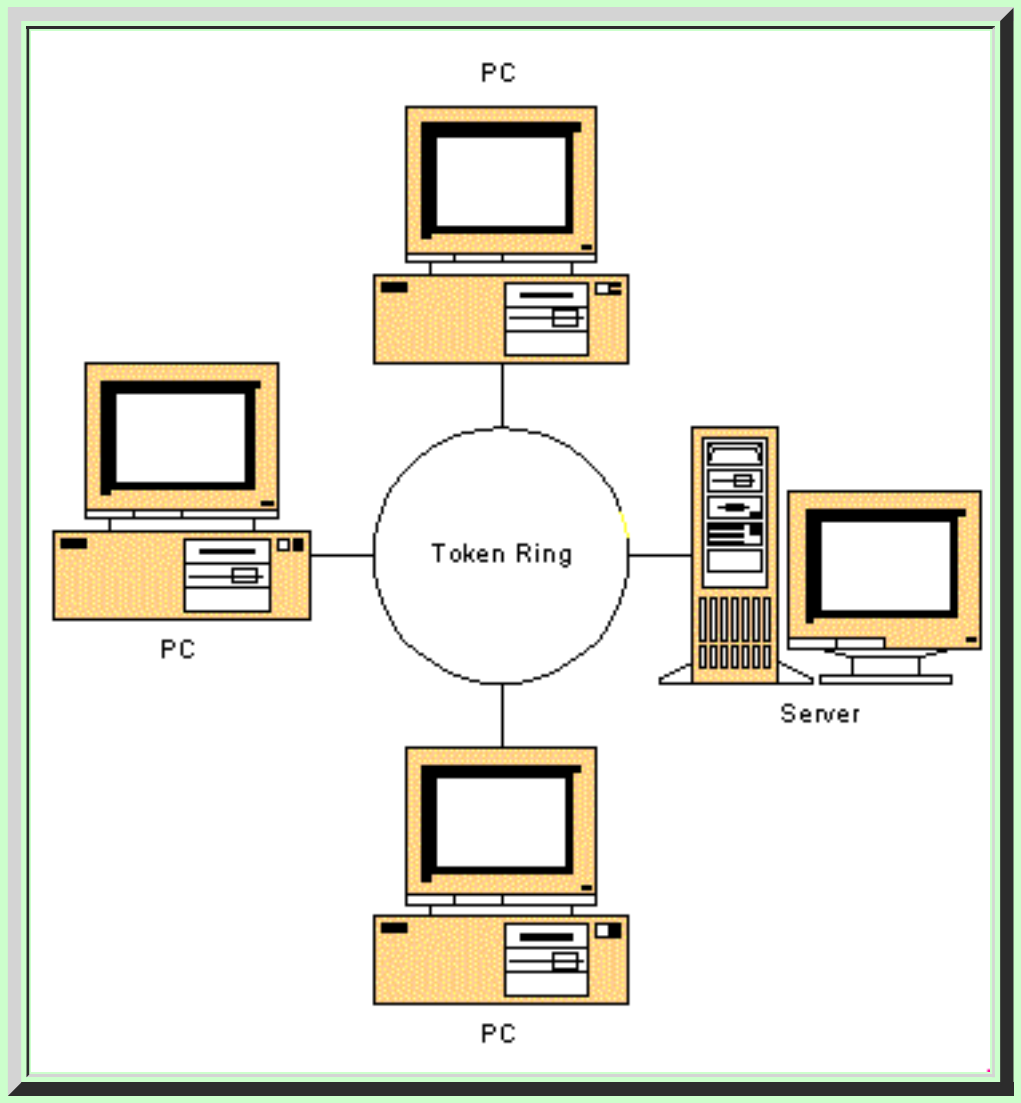
    - **Token ring** (**IEEE 802.5**)
    - **Token *bus*** (**IEEE 802.4**)

Architecture of the 802.5 - Token Ring

- **IEEE802.5 - Token ring *architecture***
  - **Architecture:**
    - **Computers** are organized into a **physical ring**:



    - **Messages** are circulated through the **ring network**:



Frame format used in Token Ring

- **Frames used in the Token Ring protocol**
  - A **node** can **send 2 types** of **frames** in the **Token Bus** network:

    - A **token frame**
    - A ***data*** **frame**

- **Format of a token frame**
  - **Token ring**'s ***token*** **frame format**:

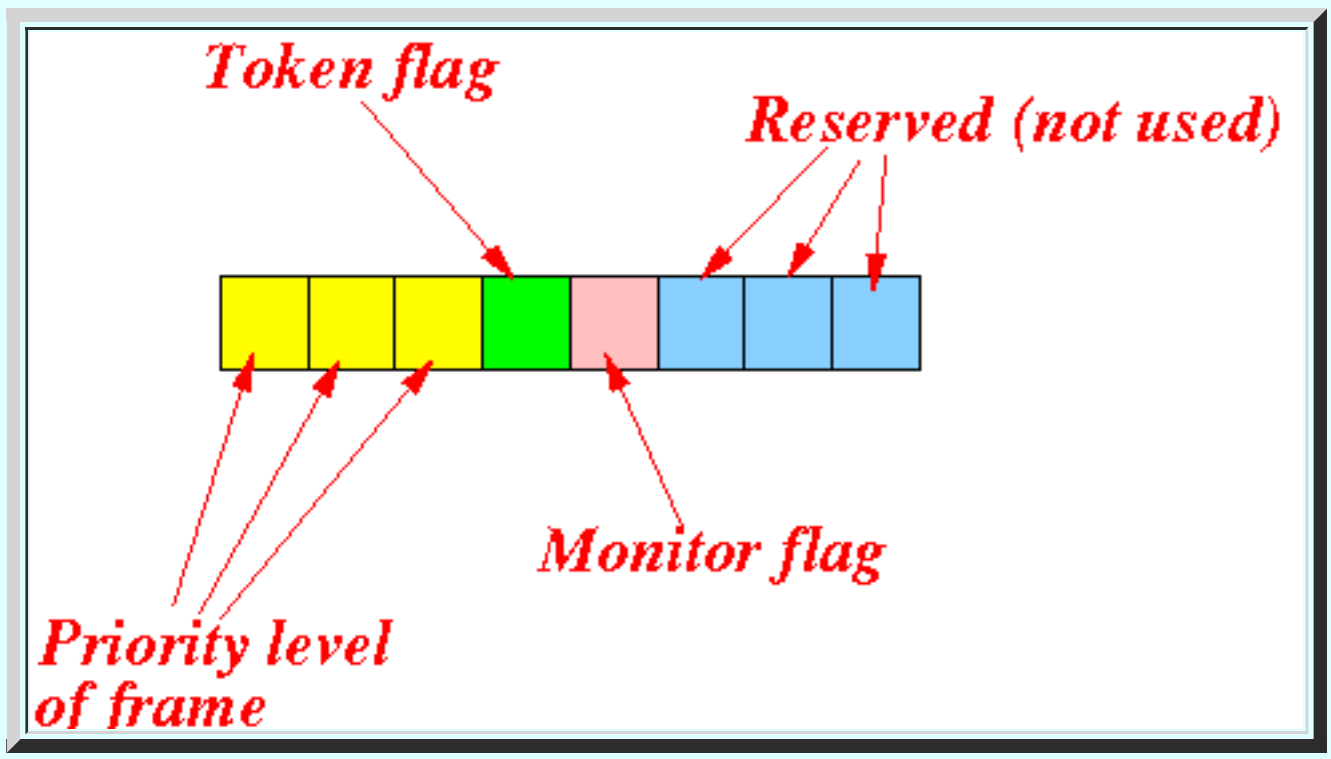
Meaning of the **fields**:

- **Start delimiter**:

  - Signals the **start** of a **frame**

    (Use like a **start flag**)

- **Access-control byte** contains



Description:

- **Priority field**: specifies the **priority level** of the **frame**

- **Token bit**: when **bit** is **set (= 1)**, the **frame** contains the **token**

- **Monitor bit**: used by the ***activity*** **monitor** to **monitor** whether some **frame** is **circling** the ring ***indefinitely*.**

- **Reserved bits: reserved for future use**

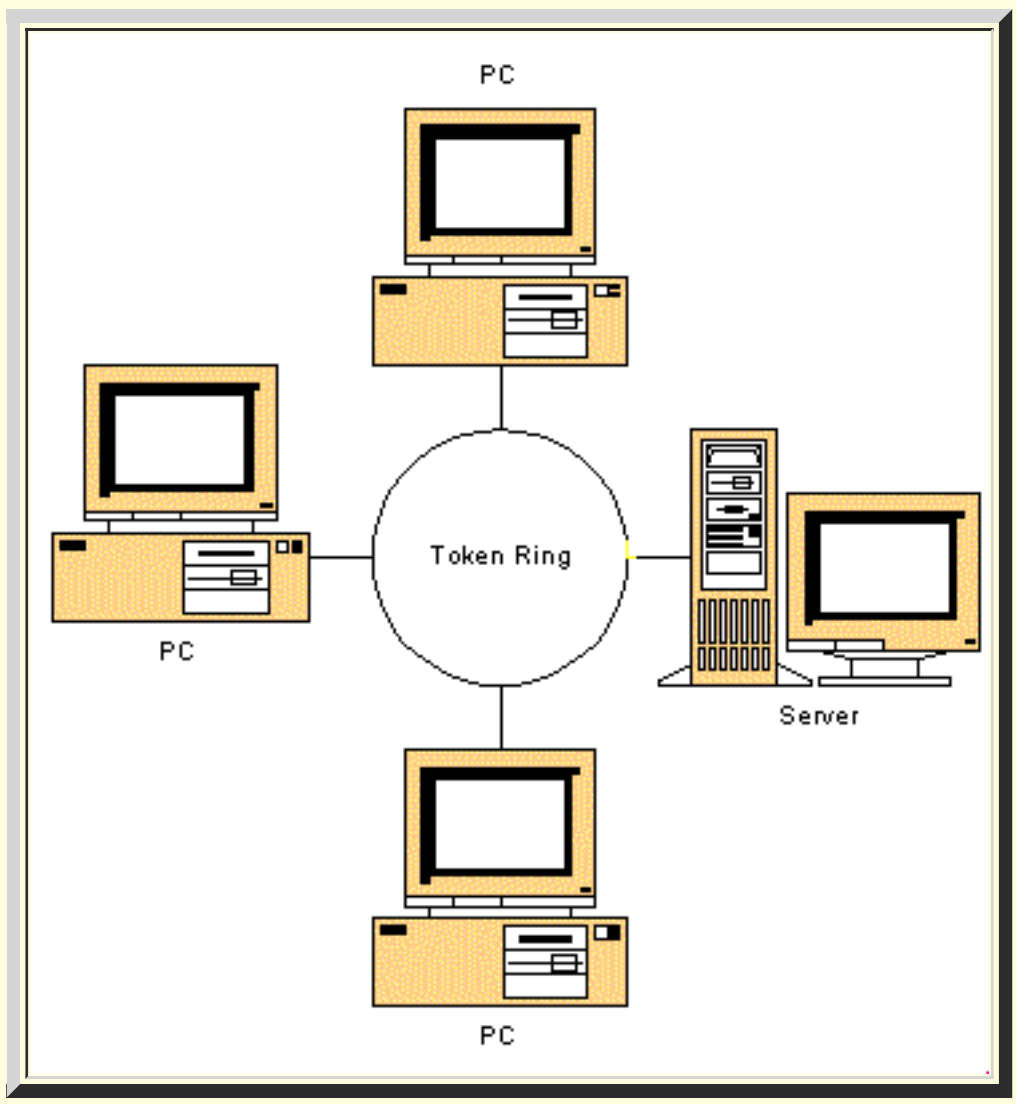
- **End delimiter**:

  - Signals the **end** of a **frame**

    (It's like an **"end flag"**)

- **Token passing:**

  - **Nodes** in the **token ring continously** pass the **token** (= a **frame** that **contains token bit = 1**) to **each other**:



  - **Only** the **node** that has **received** the **token** is **allowed** to **transmit** on the **token ring network**

- **Format of a data frame**

  - **Token ring**'s ***data*** **frame format**:

Meaning of the **fields**:

- **Start delimiter**:
  - Has been **discussed** before

- **Access control**:
  - Has been **discussed** before

- **Frame-control byte**
  - Indicates whether the **frame** contains **data** or **control** information.

- **Destination** and **source addresses**
  - Consists of two 6-byte **address fields** that **identify** the **destination** and **source** station.

- **Frame Status**
  - Is a 1-byte field **terminating** a **command/data** frame.

  - The **Frame Status** field **includes**:
    - the **address-recognized** indicator **bit**
    - the **frame-copied** indicator **bit**

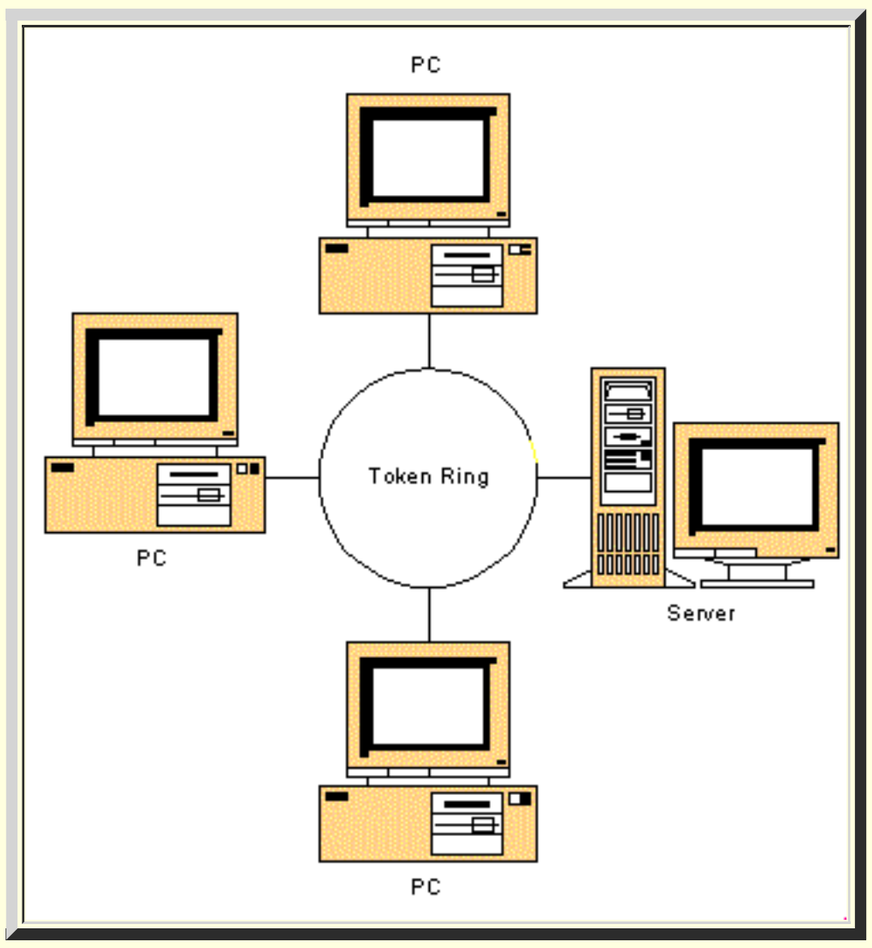
End delimiter:

- Has been **discussed** before

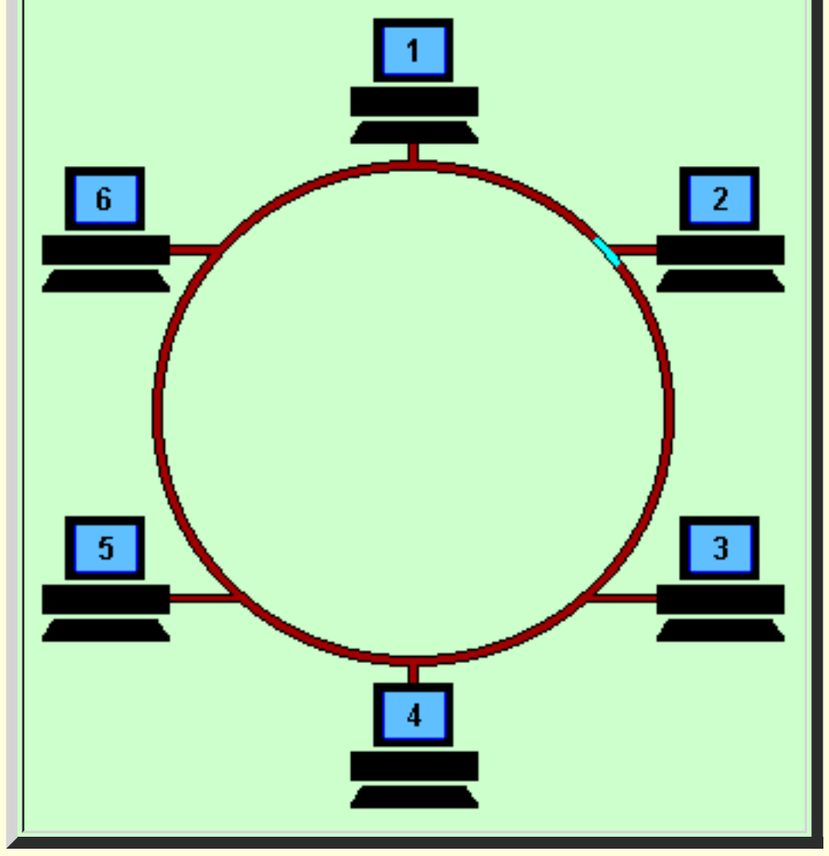
- **Note:**
  - **Only** the **node** that **has received** the **token (frame)** is **allowed** to **transmit** a **data frame** !!!

The Token Ring network protocol

- **Token holder**
  - **Token holder:**
    - **Token holder** = the **node** that has **received** the **token frame**
      - There is ***only*** **one token holder** in the **entire token ring**
    - **Only** the **token holder** is **allowed** to **transmit** a **frame** !!!
  - **Note:**
    - The **token frame** is ***also*** a **frame**
    - **Only** the **token holder** is **allowed** to **transmit** the **token frame** (to ***another*** **node**)
  - **Comment:**
    - When **nodes** are **idle** (= have **no data frames** to **transmit**), the **nodes** will **transmit token frames** to **each other**:

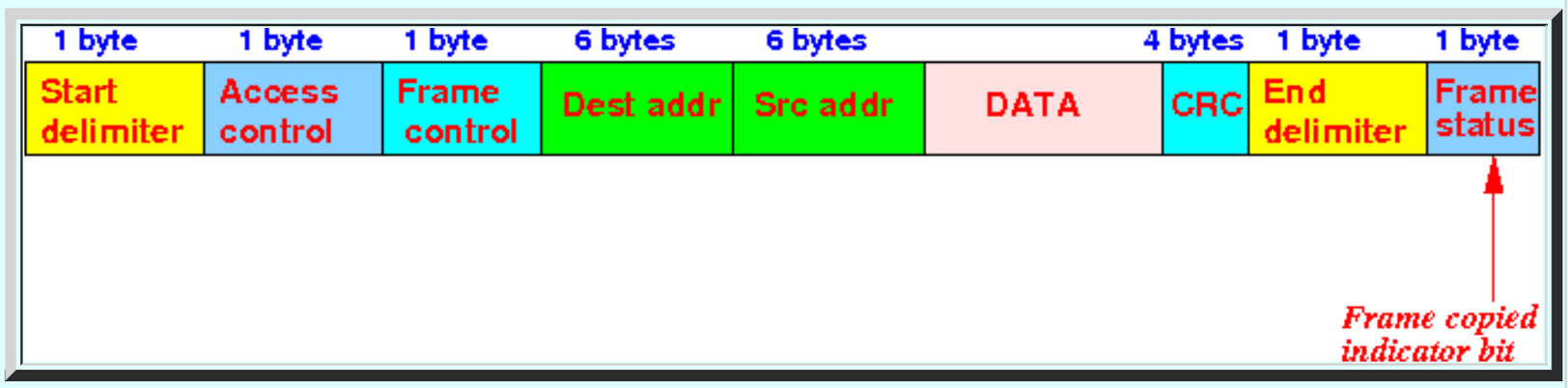


- **The IEEE 802.5 Token Ring *protocol***
  - **Operation:**
    - The **token holder** is **allowed** to **transmit** (only) ***one*** **message**
    - A **transmitted messages** will **circulated through the entire ring** and **returned back to the sender**



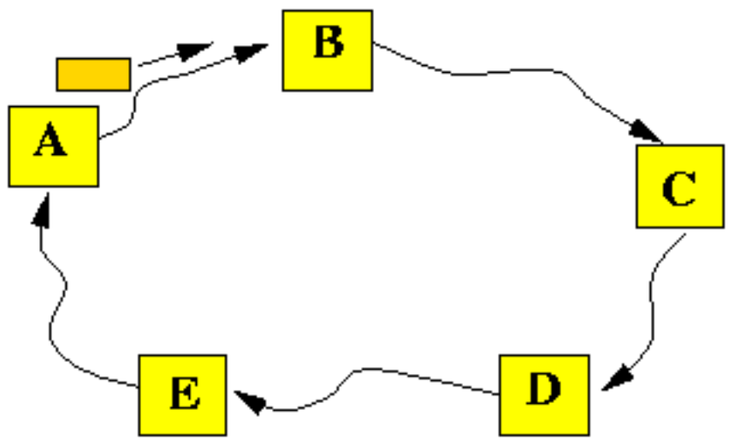
- A ***non*-destination node** will:
  - **Forward** the **data frame** to the ***next*** **node**

- When the **transmitted message** arrives at the ***destination*** **node**:
  - The **destination node** will:
    - **Copy** the **data frame**
    - **Set** the **frame copied *indicator bit*** at the ***end*** of the **data frame**:

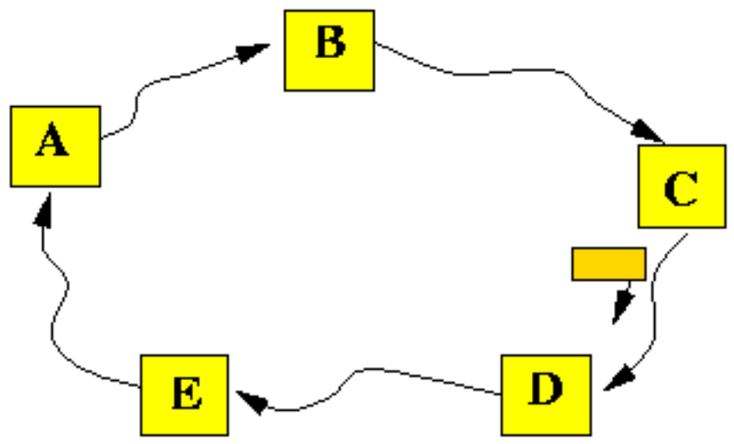


    - **Forward** the **data frame** to the ***next*** **node**
  - When the **data frame** returns to the ***sender***:
    - The **sender** transmits a ***token*** **frame** to the ***next*** **node**

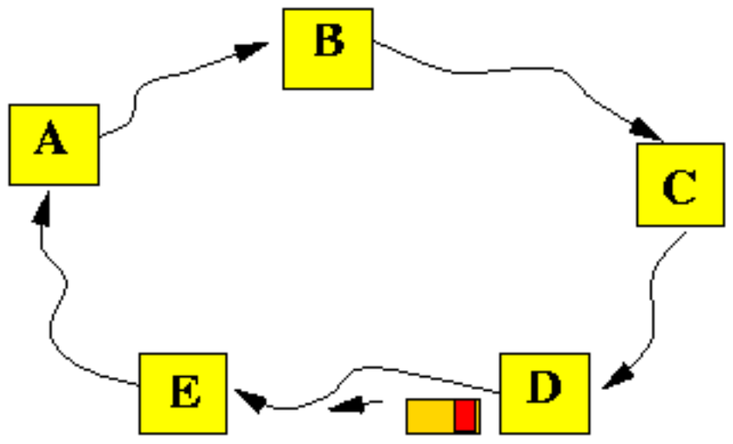
- **The frame copied indictor bit**
  - The **token ring** has a **"built-in" ACK method** because:
    - A **frame** with the **frame copy indicator bit** that has been **set** by the **receiver**, will travel **back** to the **sender** !!!
    - This **works** like an **acknowledge frame** !!!
  - The **frame copied bit** explained ***graphically***:



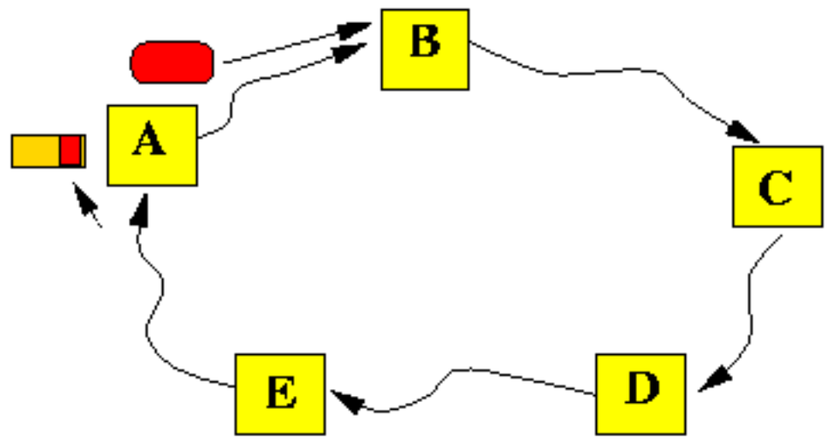
A sends a message to D



Nodes B and C will simply pass on the message



D will copy the message AND pass it on
D will also SET the ACK bit in message !



A receives the (ACKED) message back
and then send a TOKEN to its neighbor

Explanation:

1. **Suppose** node ***A*** is the **token holder**

---

2. **Node *A*** can **transmit** a **frame** (intended for **node *D***)

---

3. When **node *D*** received the **message**, it **sets** the **ACK bit** in the **tail** of the **message**

   - **Node *D*** can do this ***while*** the **message *passes* by** the **node** !

---

4. The **message** will **return** back to the **sender node *A***

---

5. **Then** the **node *A*** transmits a ***token* frame (message)**

   - The **next node (*B*)** in the **ring** will become the **token holder**

   - If **node *B*** has ***no*** **message** to transmit, it will transmit a ***token* frame**

     Otherwise, it sends a ***data* frame** and follow the **procedure** above.

- **Strength and Weakness of the Token ring protocol**
  - **Strengths:**
    - **Good performance** in ***highly loaded*** **situation**

    - Has a **built-in ACK scheme**

  - **Weaknesses:**

- **Poor performance** in **lightly loaded situation**:

  - When **most nodes are *idle***, the **token** will **circulate by** many nodes **"in vain"**

- ***Difficult*** to **add/delete** nodes (computers) from the **token *ring***

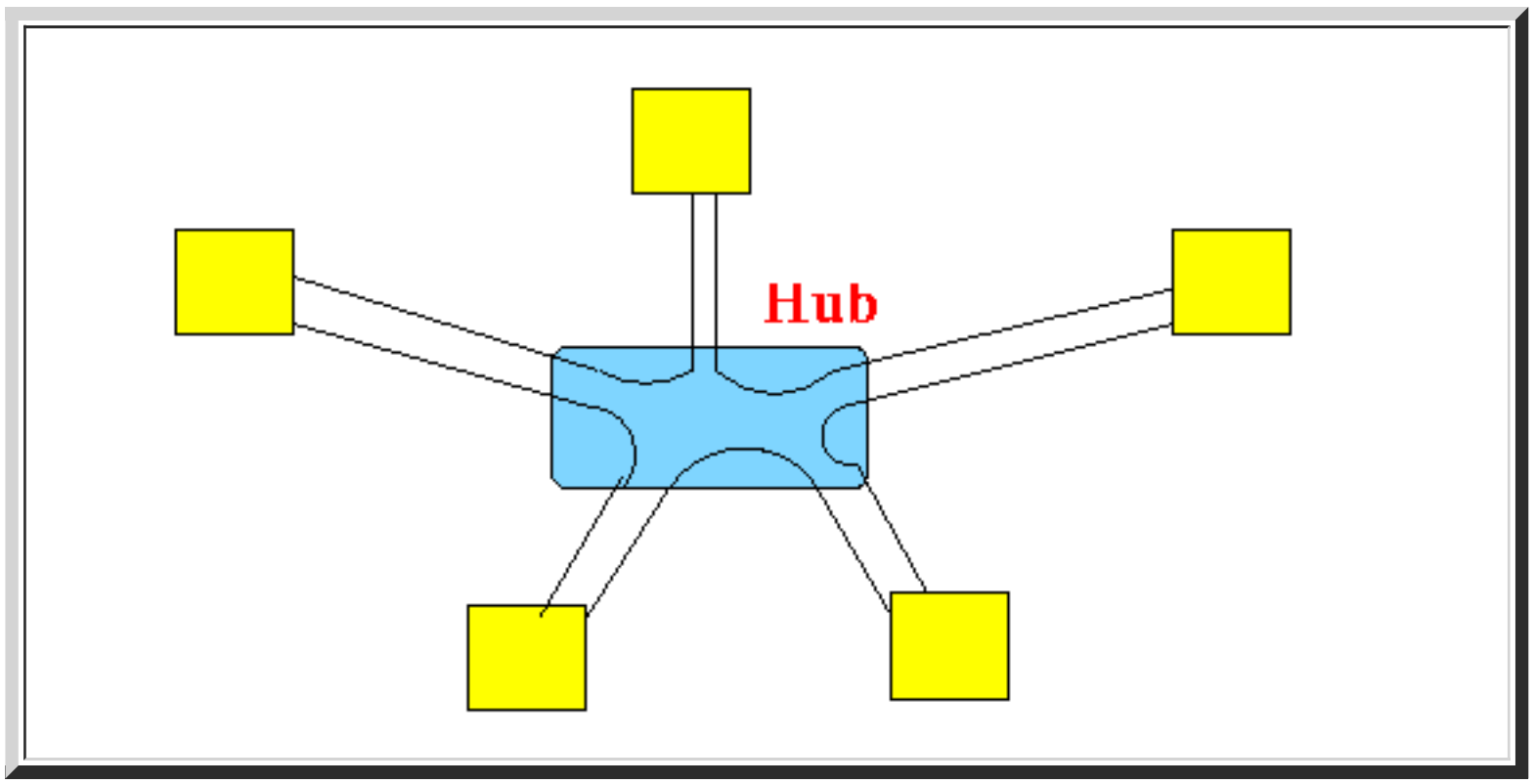
  - You need to ***wired*** the **node** into the **ring** !!!

- **Cable breakage** will cause the ***entire*** **network** to fail...

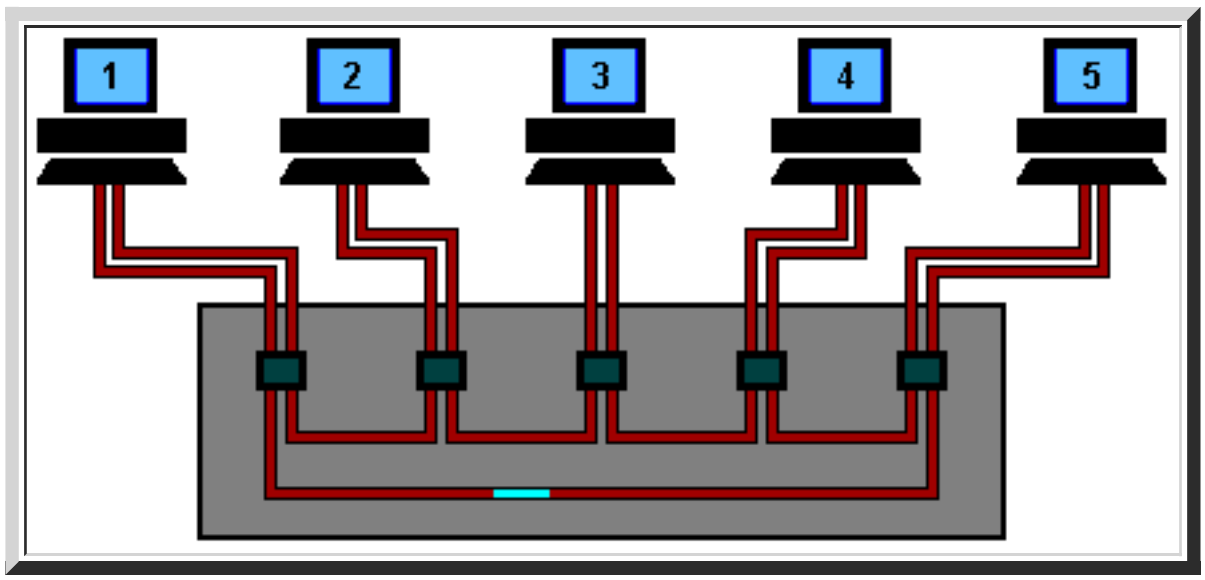
  (Because the **ring** is **disconnected** !!!!)

The "Star-Ring" configuration

- **The *Star*-Ring configuration**
  - The ***star-shaped* ring topology** was proposed to **overcome** the **cable failure problem**:

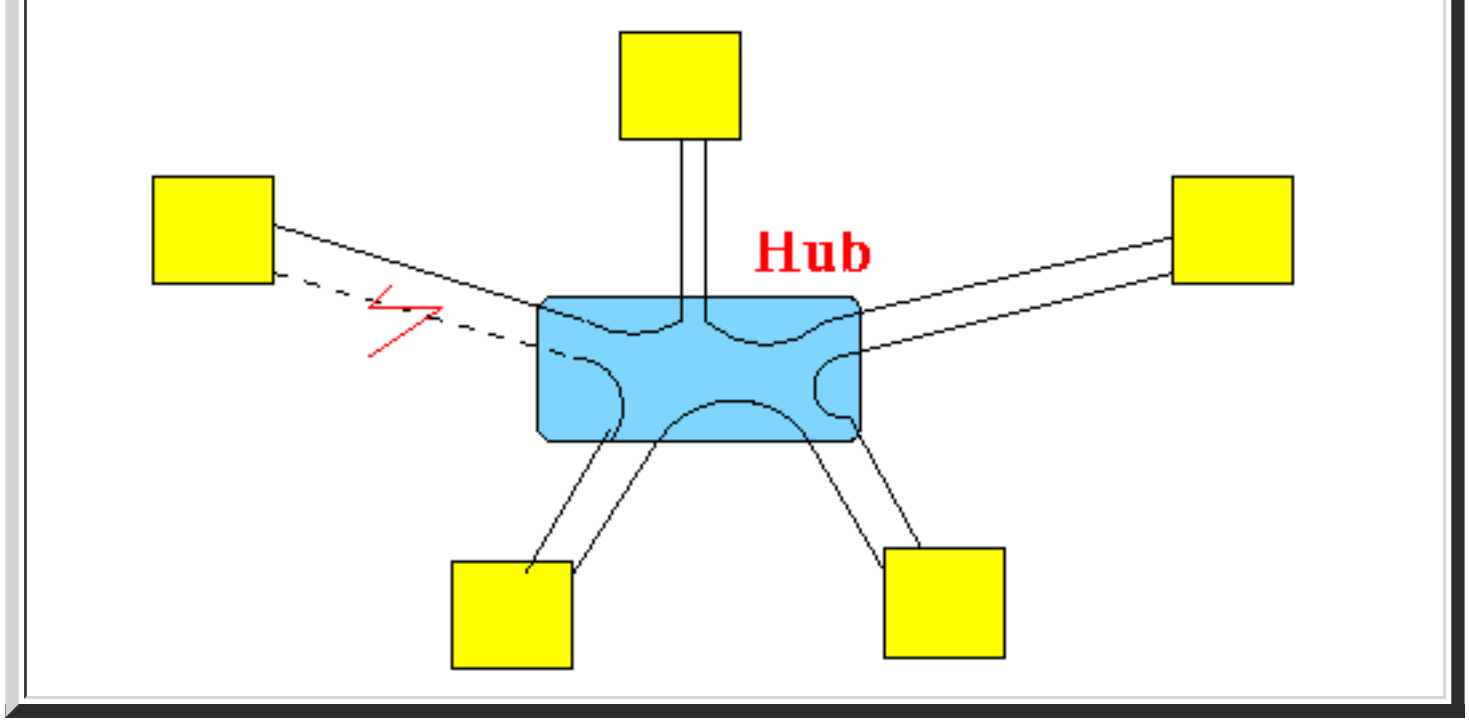


The **actual shape** of the **hub** or **concentrator** is a **box**:



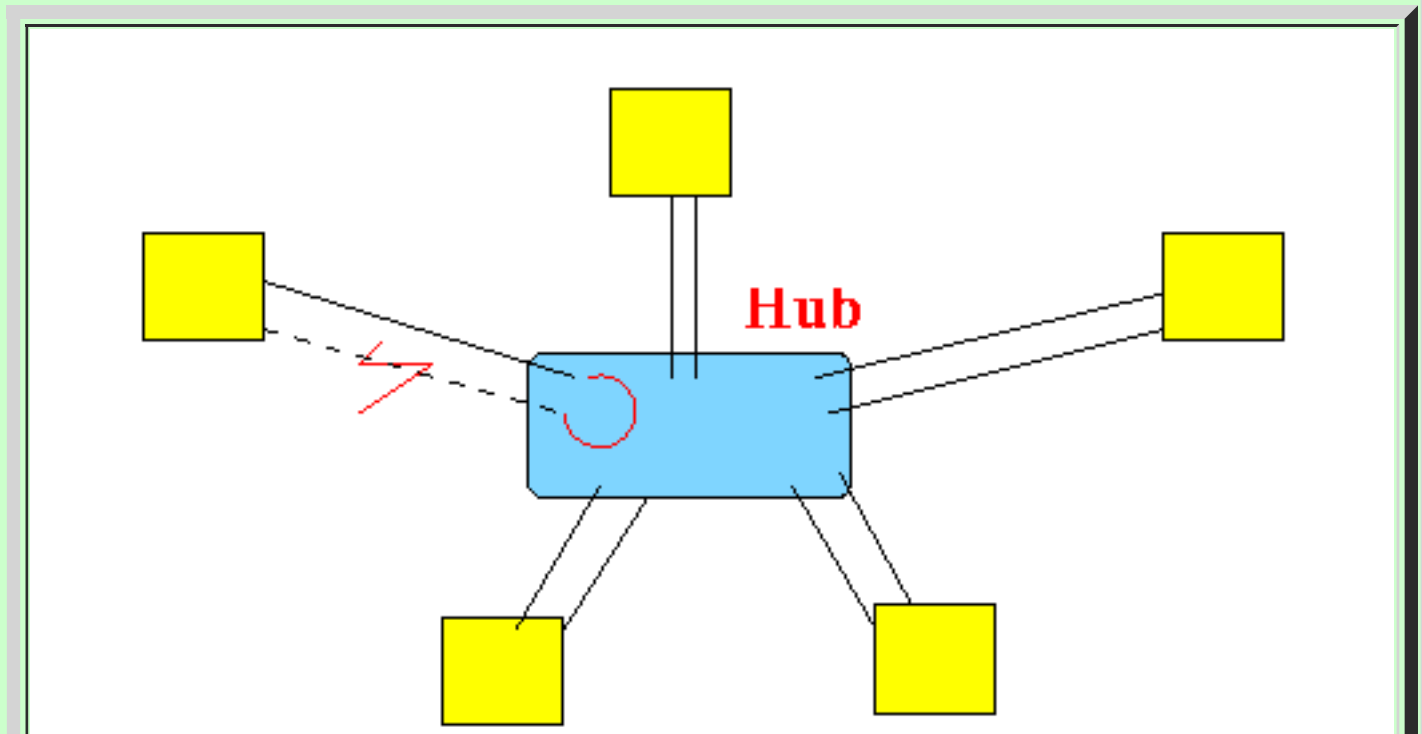
The **connection** forms a **physical ring** !!!

- **Suppose** a section of the **Token Ring** was **disrupted**:



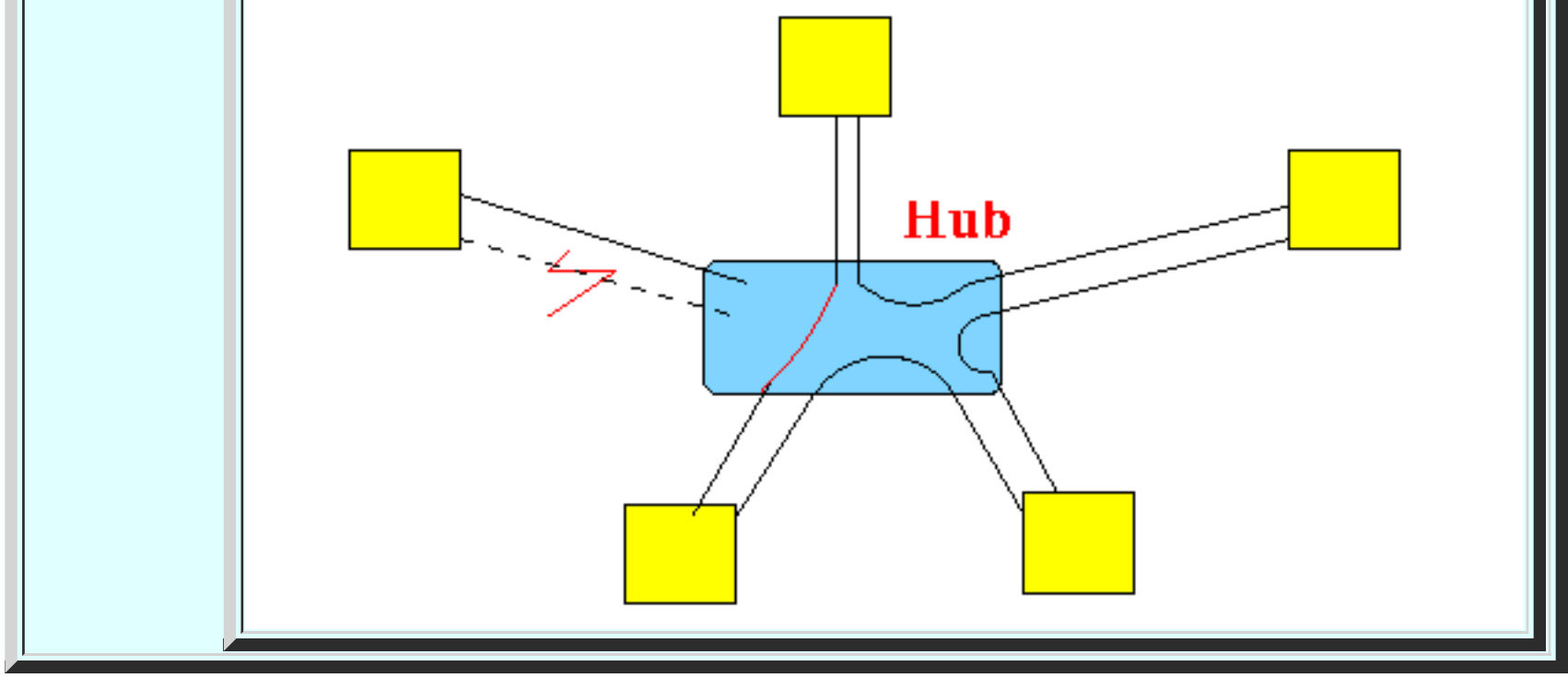
Locating the **cable disruption** is now **relatively easy**:

- Just find the 2 end points that **does *not*** for a **closed loop**:



Fixing the **token ring** is **also relatively easy**:

- **Omit** the **disconnected segment**:

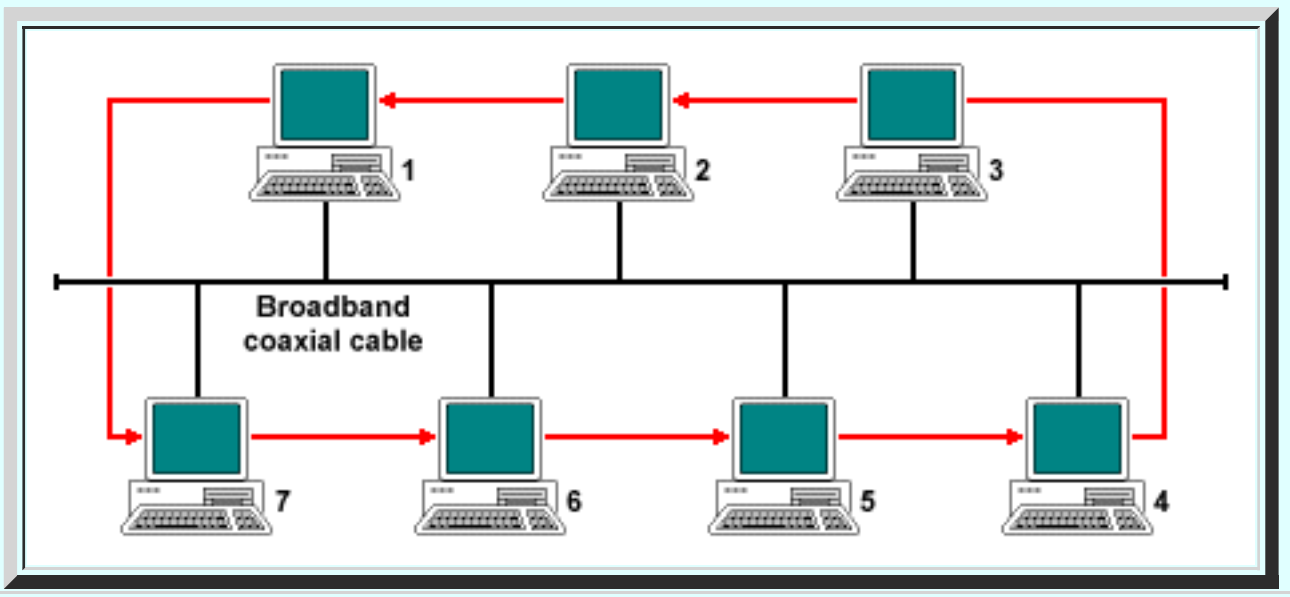

Hub

Intro to 802.4 - Token Bus

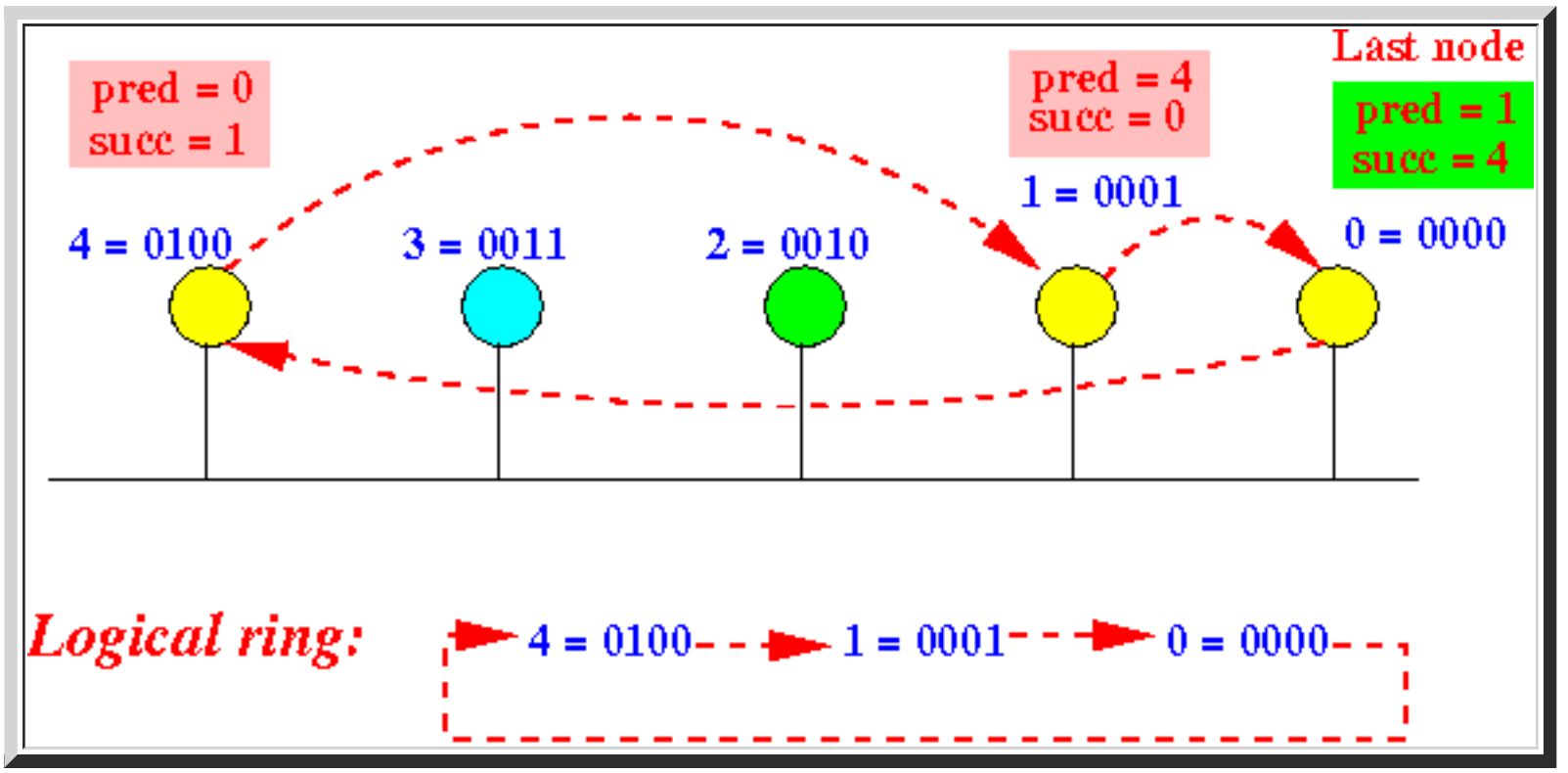
- **IEEE 802.4 - Token Bus *architecture***
  - **Architecture** of the **IEEE 802.4 Token bus:**

    - The **nodes** in **IEEE 802.4** are:
      - **Physically *connected*** using a ***bus*** **network** (e.g., using **Ethernet cables**)
      - ***Logically organized*** into a **ring struture**

    Example:



  - **Schematically:**



  - **Information** maintain by each **node**:

- **nodeID** = the **ID** of the **node**

- **pred** = the **ID** of its **logical *predecessor***

  - **Predecessor** = the **node** with the **next *smaller* ID** that is a **member** of the **token bus**

    (**Except** for the **node** with the ***smallest* ID**

    The **predecesser** of the **node** with the ***smallest* ID** is the **node** with the ***largest* ID** that is a **member** of the **token bus**)

- **succ** = the **ID** of its **logical *successor***

  - **Successor** = the **node** with the **next *larger* ID** that is a **member** of the **token bus**

    (**Except** for the **node** with the ***largest* ID**

    The **predecesser** of the **node** with the ***largest* ID** is the **node** with the ***smallest* ID** that is a **member** of the **token bus**)

Note:

- Some **nodes** may ***not* participate** in the **token bus** !!!

- A **node** must be ***inserted*** into the **Token bus** to **become a *member* node** of the **Token bus**

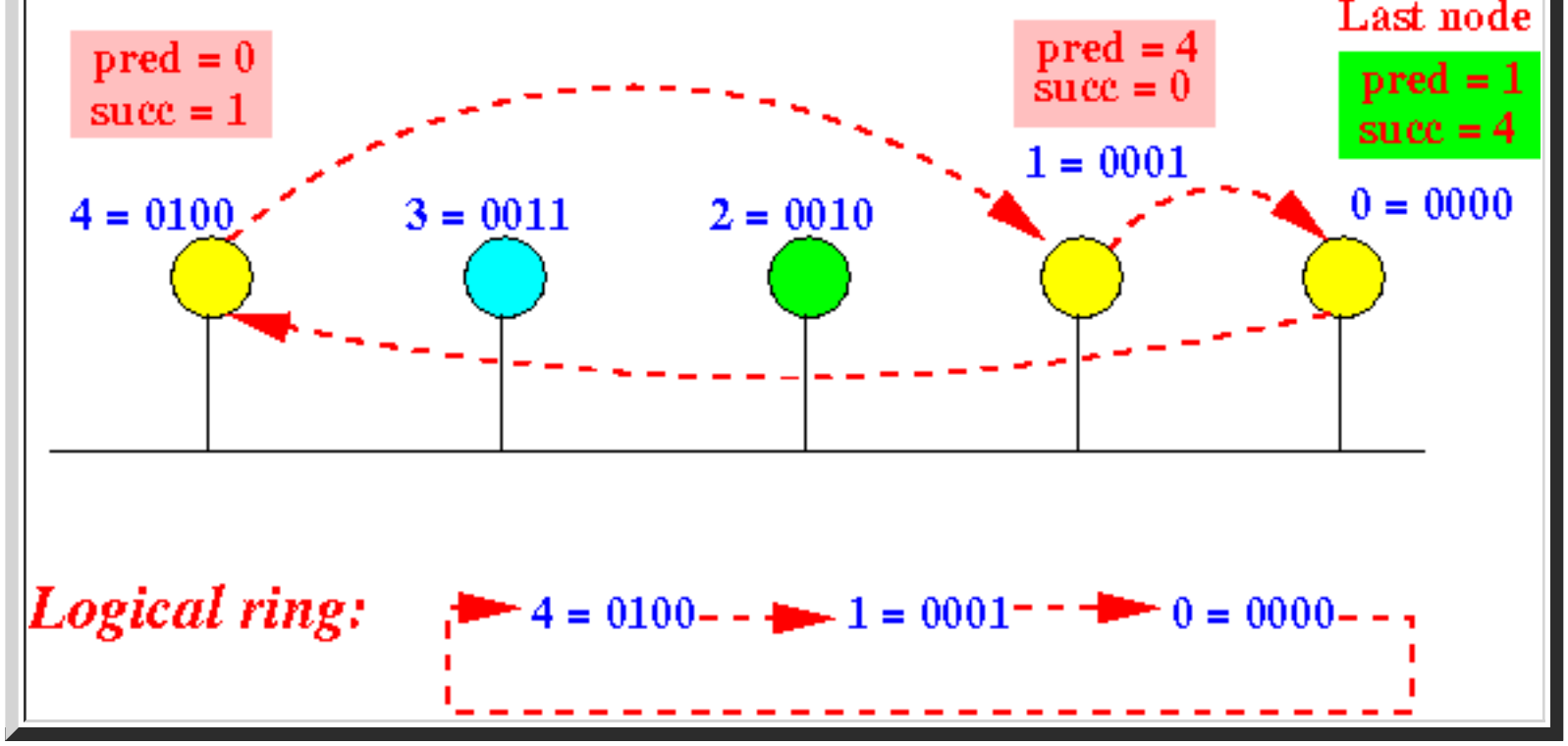
- **Node insertion** uses a ***insertion* protocol**

- **The *logical* ordering of nodes in the Token Bus**

  - The ***order*** of the **nodes** in the ***logical* ring** is **always**:

    - by ***descending* node ID**

    Example:



The ***ring*** consist of the **following nodes** (in this ***specific*** **order**):

1. **Node #4**
2. **Node #1**
3. **Node #0**

Node #0 is the ***last*** **node** in the **token bus**

- **Successor node**
  - **Important Property**:
    - The ***successor*** **ID** of a **node** is ***always*** **smaller** that the **node's nodeID**

      (**Because** the **ordering** is ***descending*** in **nodeID**)

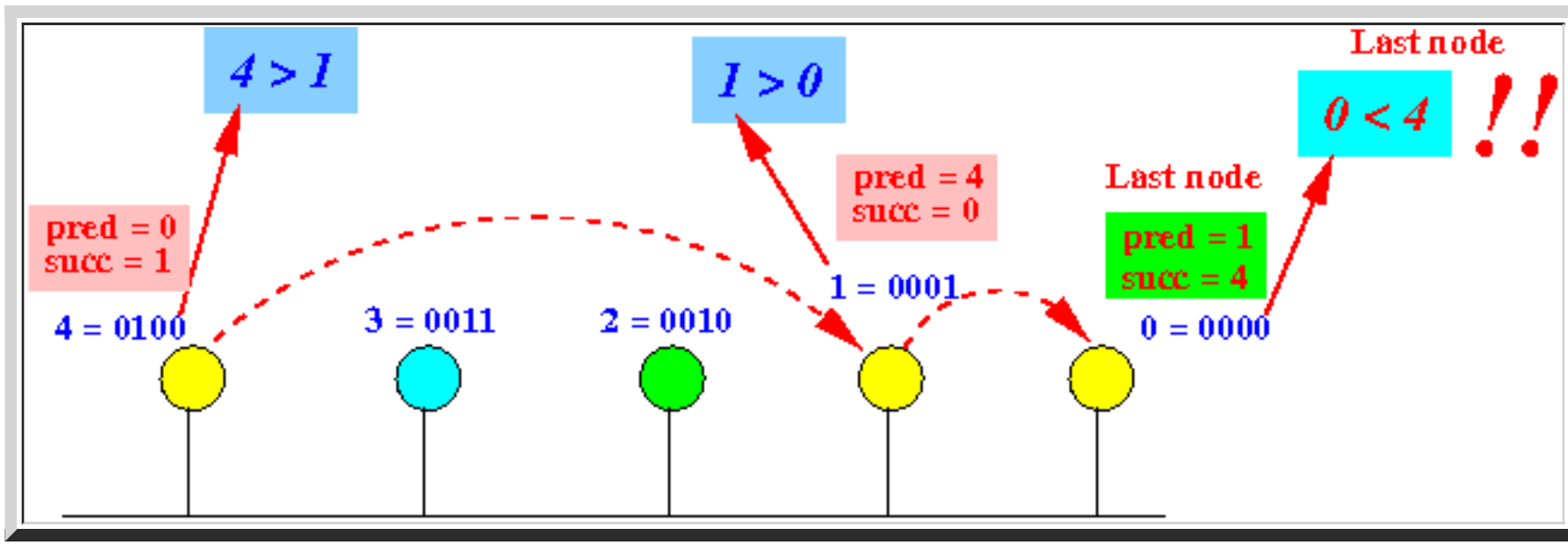
      Exception:
      - The ***last*** **node** in the **token ring** !!!!

- **How a node can tell whether it is the *last* node in the ring**
  - **"Normal" nodes** (= ***non*-last nodes**) will have:
    - **node ID > successor ID**

  - ***Only*** the ***last*** **node** will have:

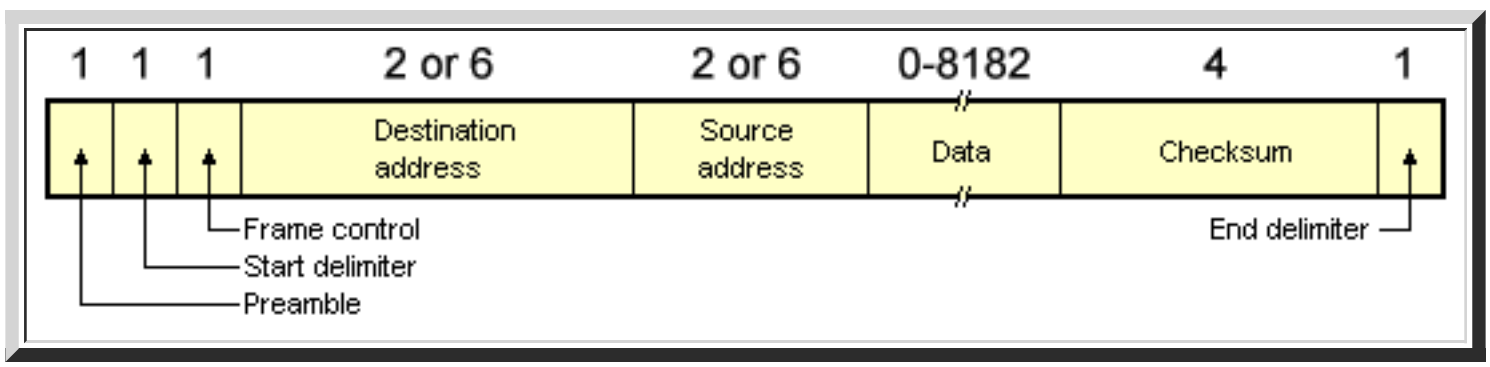
- **node ID < successor ID**

- **Example:**



- **Frame format of the token bus**

  - **Frame format:**



(**Unit** = **# bytes**)

Meaning of the **fields**:

- **Preamble** = used to **synchronise** the **receiver's clock**.

- **Start Delimeter** and **End Delimeter** = used to **mark** the **start** and **end** of the **frame**

- **Frame control** = indicates the **type** of the **frame** (see below)

- **Frame control codes**

  - **Sample token Bus Control Frames:**

Frame control field	Name	Meaning

00000000	`Claim_token`	**Claim** the **token** during **ring initialization**
00000001	`Solicit_successor_1`	**Invite** stations to **enter** the **ring**
00000010	`Solicit_successor_2`	**Invite** stations to **enter** the **ring** (used by the ***last*** **node**)
00000011	`Who_follows`	**Recover** a ***lost*** **token** (due to **node failure**)
00000100	`Resolve_contention`	**Used** when ***multiple*** **stations** want to **enter the ring**
00001000	`Token`	**Pass** the **token** **Also** used as a `Set_predecessor` control frame
00001100	`Set_successor`	Allow stations to **leave** the **ring** (used in ***various*** **protocols**)
01MMMPPP	`Data_frame`	**Data frame** (**PPP** are **priority level** bits)

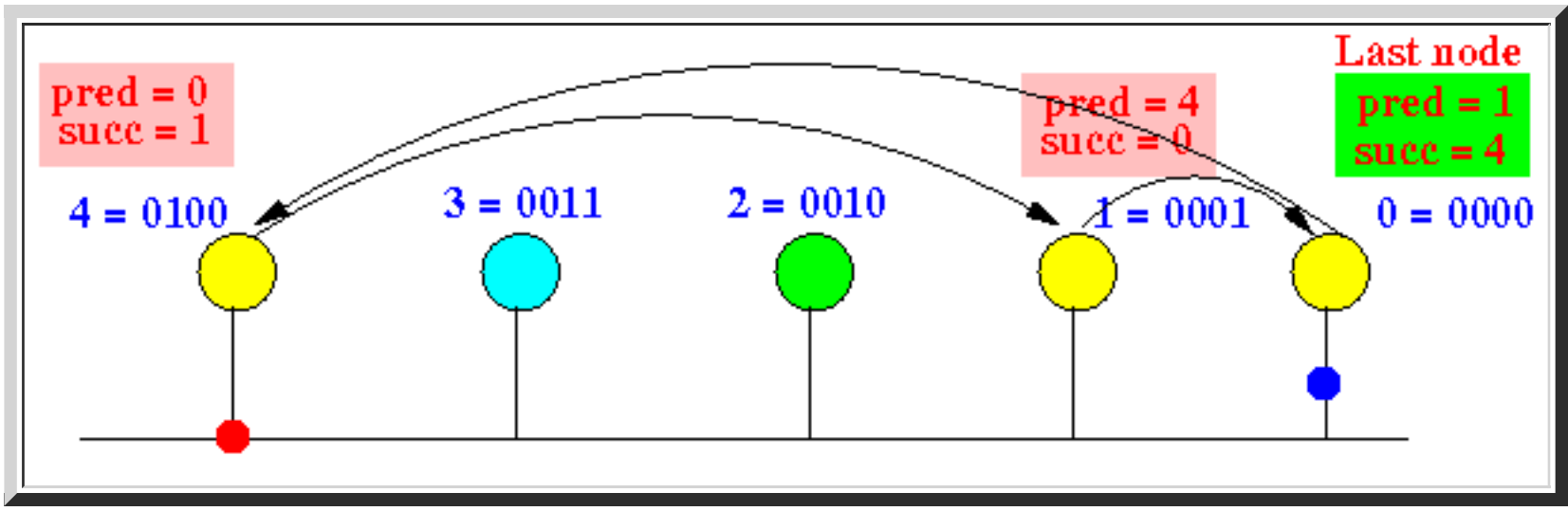
Note:

- The **use** of the **control frames** will be **explaned** ***later***...

The 802.4 - Token Bus protocol

- **Transmission order in token bus**
  - **Transmission order**:
    - **After** a **node** has **transmitted** a **data frame**:
      - The **node** will **pass** the **token (frame)** to its ***successor*** **node**

- **IEEE 802.4 - Token Bus *protocol***
  - **Token bus Operation**: (**frame transmission**)



Explanation:

- **Node 0** is the ***current*** **token holder**
  - **Successor** of **node 0** = **node 4**

- **Node 0** sends a **token message** to it ***successor*** **(node 4)**
  - **Token** = the ***red*** **dot** in the **figure**

- When **node 4** receives the **token** message:
  - **Node 4** can **transmit** a **message**

    (**message** = the **blue dot**)

- ***After*** transmitting the **message**, the **node 4** will send a **token message** (**red** dot) to its **successor**

- **Token bus vs. Ethernet**
  - **Advantage** of **Ethernet** (over **Token bus**):
    - **Ethernet** performs **much better** in ***low*** **traffic conditions**
      - **Low traffic** will have **few *simultaneous* transmitters**
      - **Fewer *simultaneous* transmitters** will result in **low probability** of **collision**
      - ***Most* transmissions** will be **received *correctly*** on the ***first* try** !!!
  - **Short-coming** of **Ethernet**:
    - **Ethernet** performs **terribly** in ***high*** **traffic conditions**
      - **High traffic** will have **many *simultaneous* transmitters**
      - **More *simultaneous* transmitters** will result in ***high*** **probability** of **collision**
      - ***Most* transmissions** will be ***lost*** on the ***first* try** !!!

  - **Advantage** of **Token bus**:
    - **Token bus** performs **much better** in a ***high*** **traffic load** condition:
      - There are **no collisions** in **Token bus** !!
  - **Short-coming** of **Token bus**:
    - **Token bus** performs **terribly** in a ***low*** **traffic load** condition:
      - A **node** must **pass** the **token** after the **node** has **transmitted** its **frame**

        (This is **done** for **fairness** sake !!!)

- The **node must pass** the **token *even*** if its the ***only*** **node** that is **transmitting** !!!

    - The **node** will ***waste*** **time waiting** for the **token** to **come back**....

- **Analogy**

    - **Analogy:**

        - **Ethernet** is like a **stop sign**

            - In **low traffic condition**, **intersections** with a **stop sign** performs ***much better*** than **intersections** with a **stop light**

        - **Token bus** is like a **stop *light***

            - In ***high*** **traffic condition**, **intersections** with a **stop *light*** performs ***much better*** than **intersections** with a **stop sign**

            - In ***low*** **traffic condition**, you will **have to wait** for the **stop light** to **turn green** (= **receive** the **token**) before you can **cross** the **intersection**

- **Token *bus* vs. Token *ring***

    - **Advantages** of **Token *bus*** over **token *ring***:

        - **Token bus *network*** is ***easier*** to **maintain**:

            - **Bus network** is **easier** to **connect**

        - **Token bus** is ***more*** **flexible**:

            - **Nodes** can be ***added*** to the **network** more ***easily***:

(Just **connect** the **node** to the **bus network**.

No need to **split** the **ring network opne** !!!)

- **Disadvantages** of the **Token bus network**:

  - **Complicated maintenance protocols** to **insert/delete nods** from **token bus**

- **Token bus *maintenance***

  - The **token bus standard** include **protocols** for:

    - **Token bus membership maintenance**

  - **How to add** a **node** to the **token bus network**:

    1. **Connect** the **node** to the **(token) bus network**

       ---

    2. The **node** will be **inserted *itself*** into the **token bus network** (i.e.: become a **member** of the **token bus**) using the ***node insertion*** **protcol**

We will study these **protocols**: **token bus:**

- (Automatic) **node deletion** (this is the **easiest**)
- (Automatic) **node insertion**
- (Automatic) **ring initialization**

Control messages for ring maintainance in Token Bus

- **Maintaining the logical ring in the token bus network**
  - **Ring maintenance:**
    - **Ring maintenance** = **functions** to **add/delete nodes** to the **logical ring structure** in the **Token bus network**

- **Maintenance operations**
  - **Maintenance operations:**
    - **Remove** a **node** from the **existing logical ring**
    - **Add** a **node** to the **existing logical ring**
    - **Recover** from a **node failure**
      - **Node failure** will ***break*** the **logical ring**
    - **Ring *initialization***
      - **Procedure** to ***start*** **up** the **token bus network**

- **How is the ring maintenance operation performed ?**
  - **Fact:**
    - **Nodes** will ***exchange*** **"control messages (frames)"** with **each other** to **perform** the **ring maintenance functions**

- **Control messages (frames) used in ring maintenance operations**
  - **Control messages** used to **maintenance** of the **token bus network**:

- **`S.Token( )`**: the **token frame**

  - The **sender `S`** sends this **`S.Token( )`** message to a **receiver `R`**

  - When **receiver `R`** receives the **`S.Token( )` message**, it set:

    ```
    R.predecessor = S
    ```

    Reason:

    - A **sender `S`** will **always** send the **token** to its ***successor***

    - ***If*** the **node `S`** sends the **token** to **node `R`**, **then:**

      ```
      S's successor = R

      But also:

      R's predecessor = S !!!
      ```

- **`S.Set_successor( X )`**:

  - The **sender `S`** sends this **`S.Set_successor( X )`** message to a **receiver `R`**

  - When a **node `R`** receives the **`S.Set_successor( X )`** message, the **node `R`** will:

    1. **Update** its **`Successor`** variable:

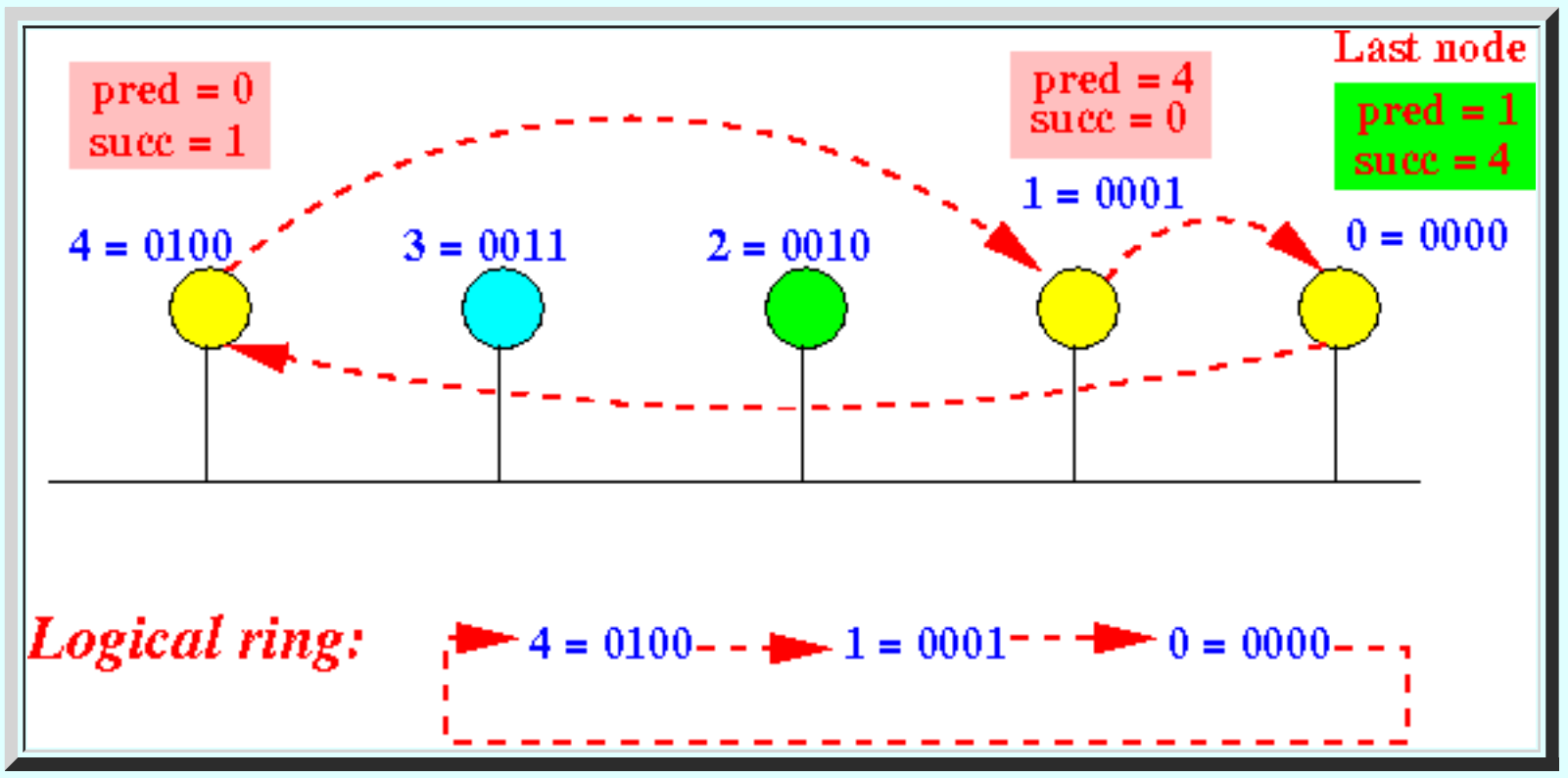
       ```
       R.successor = X
       ```

    2. **Send** the **token** to its ***new*** **successor**

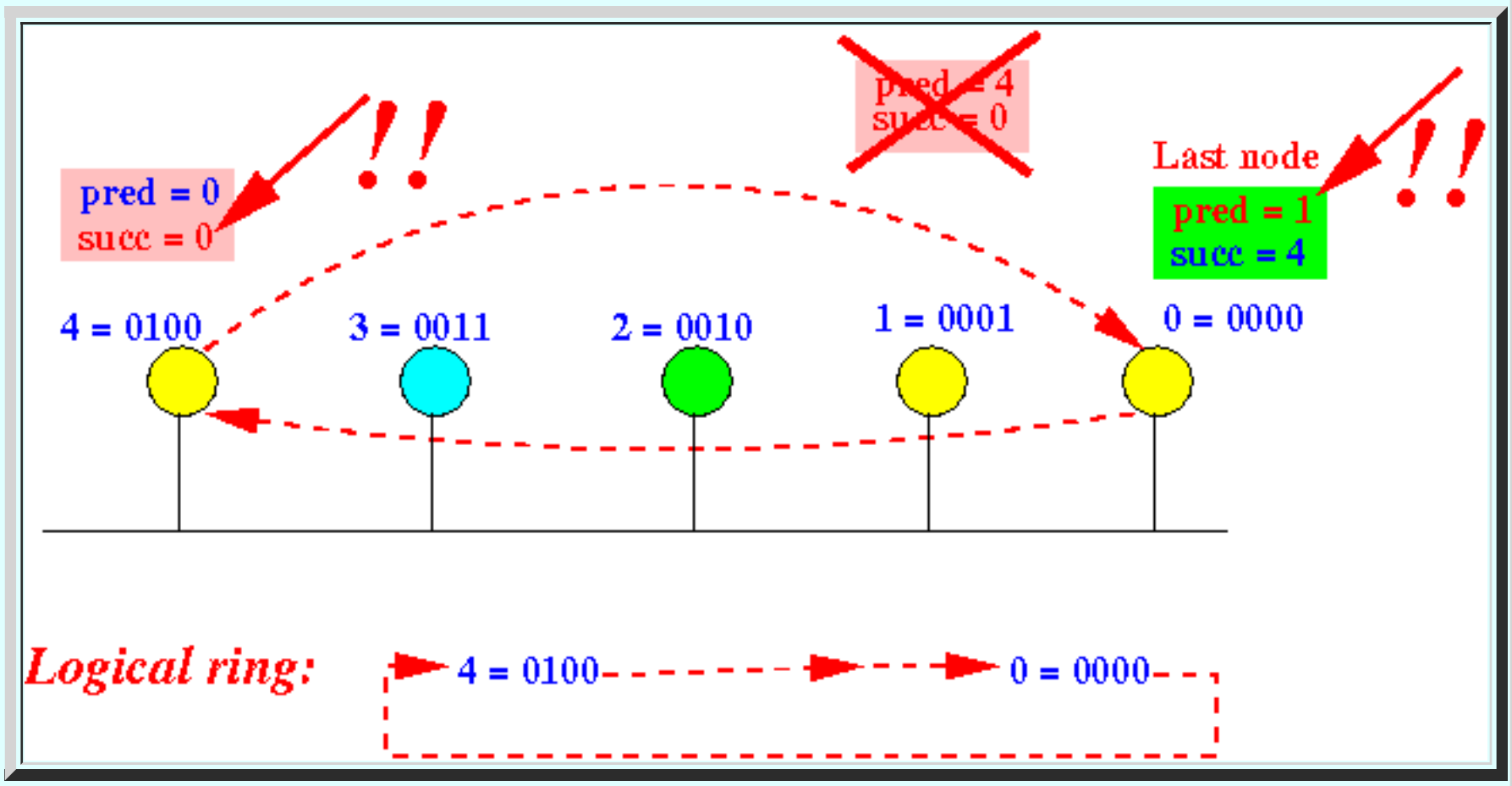
Node *deletion*

- **Removing a node from the token bus**
  - **Effect** of a **deletion**:

    - **Suppose initially** the **(logical) ring** in the **token bus network** consist of the **nodes 4, 1, 0**:



    - **After removing** the **node *1***, the ***logical* ring** will become:



  - **Note:**

    - **Node 1 can** still be **connected** to the **bus *network***

    - **Node 1** will ***not* be part** of the ***logical* ring**:

      - **Node 4** will ***not* receive** the **token frame** any more !!!

- **The Token-bus node deletion *protocol***
  - **Fact:**
    - ***If*** a **node** wants to ***leave*** the **token bus network**:
      - The **node initiates** the **node *deletion* protocol**
  - **Prelude** to **initiating** the **node deletion protocol**:
    - A **node** that wants to ***leave*** must **first**:
      - **Become** the **token holder** !!!

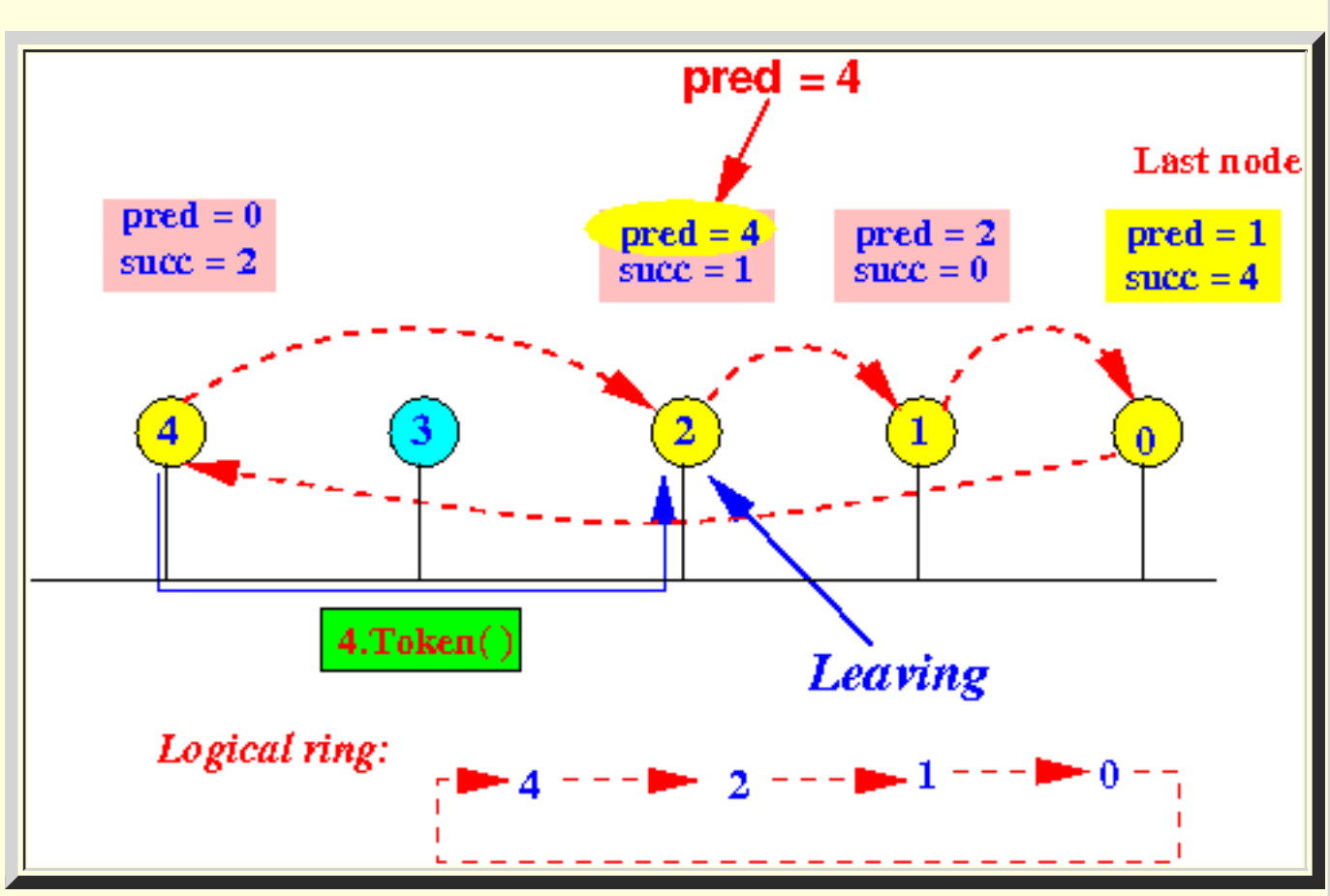
      (**Because only** the **token holder** can **transmit control messages** !!!)
  - The **node *deletion* protocol**:

1. The **"leaving" node X sends**:

   `X.Set_successor( X's succ ) control frame to X.predecessor`

   Example:

   - **Initial state**:

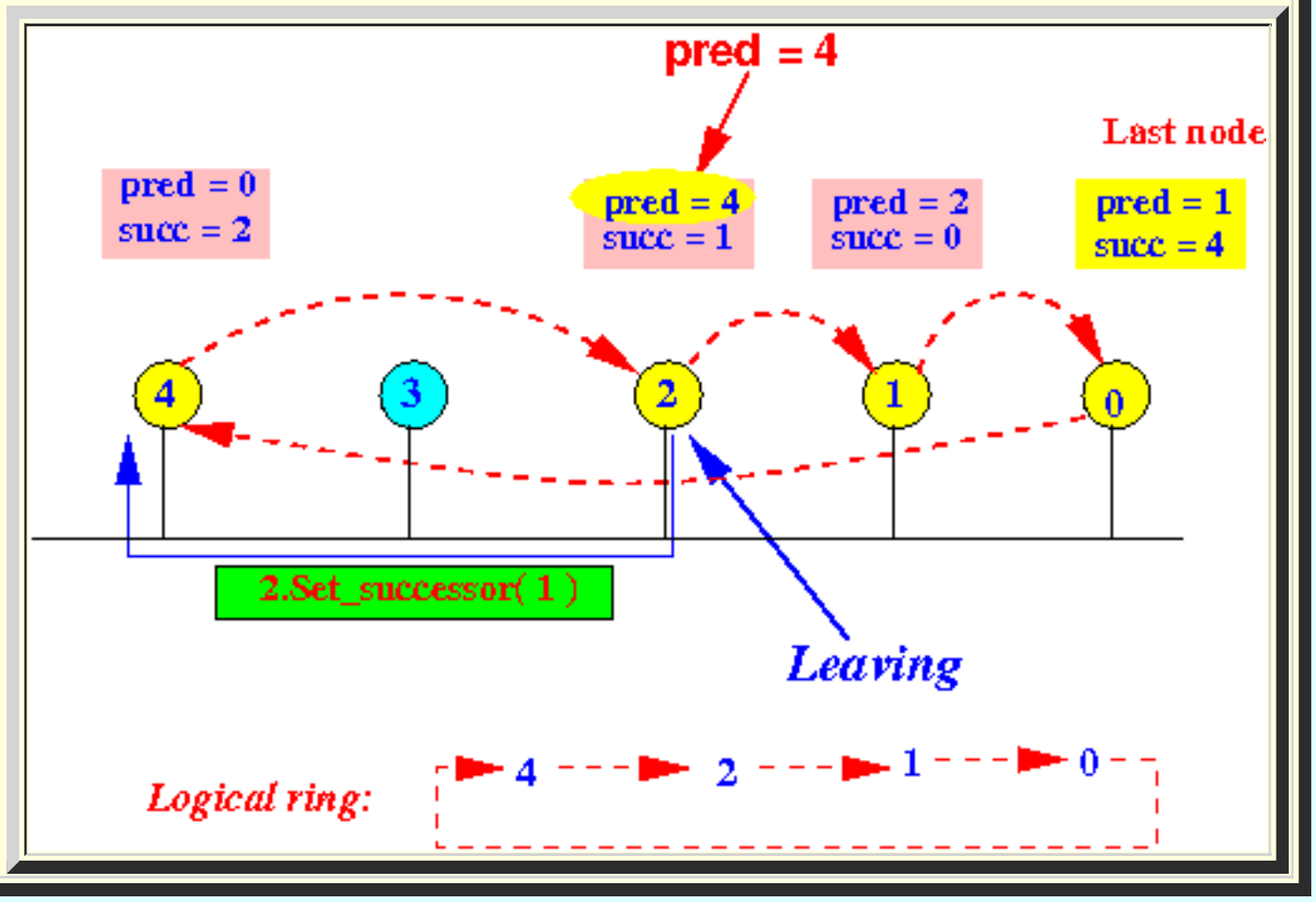


Node 2 (who wants to **leave**) has just **received** the **token** from **node 4**....

- **Node 2** that wants to ***leave*** will **send**:

```
2.SetSuccessor( 1 )     back to node 4
```

Graphically:



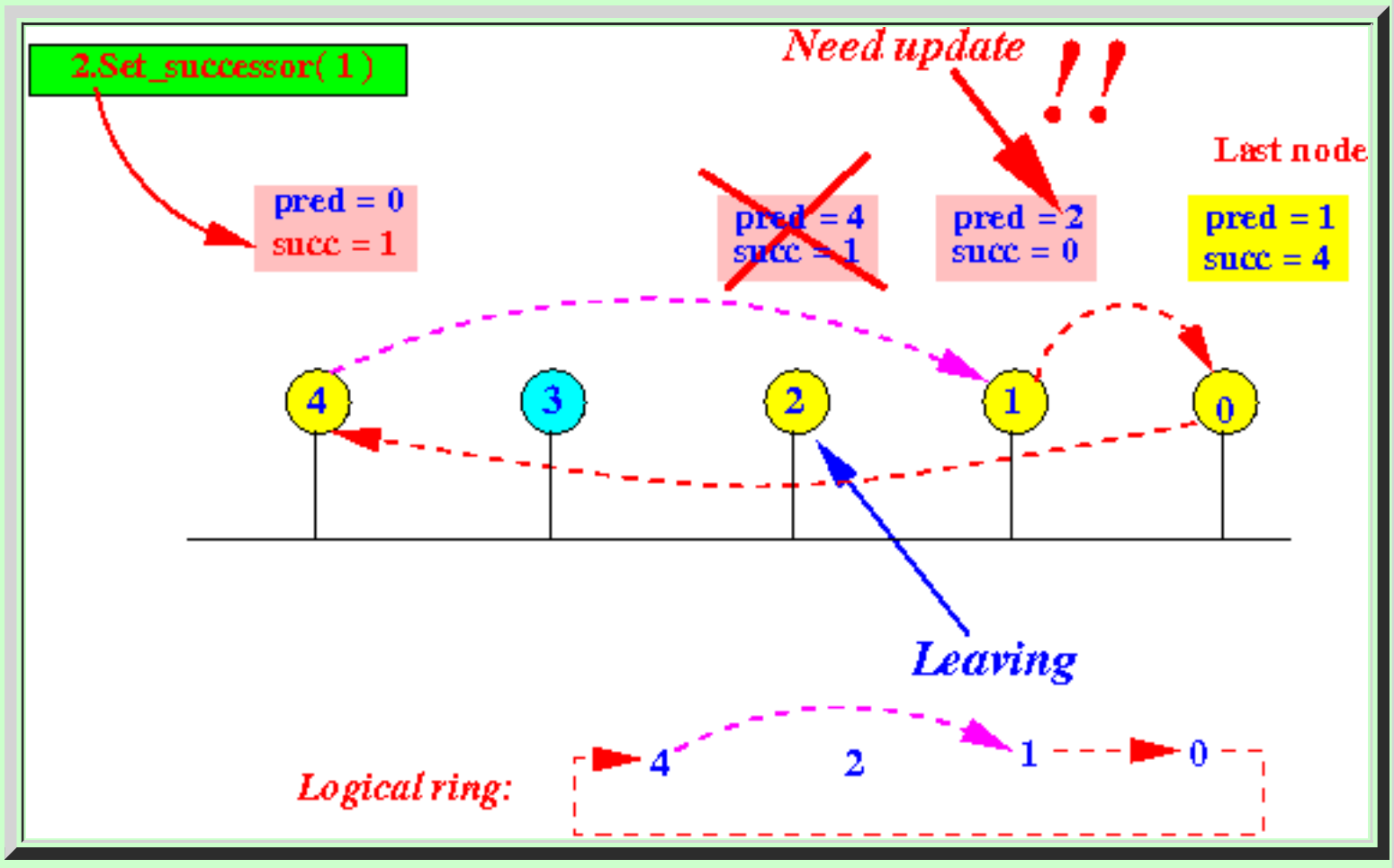
---

2. **Node** that **receives** a `Set_successor(X)` message will:

1. **Update** its **successor** to the **node ID** to `x`
2. **Send** the **token** to its ***new*** **successor node** `x`

Example: (**continued**)

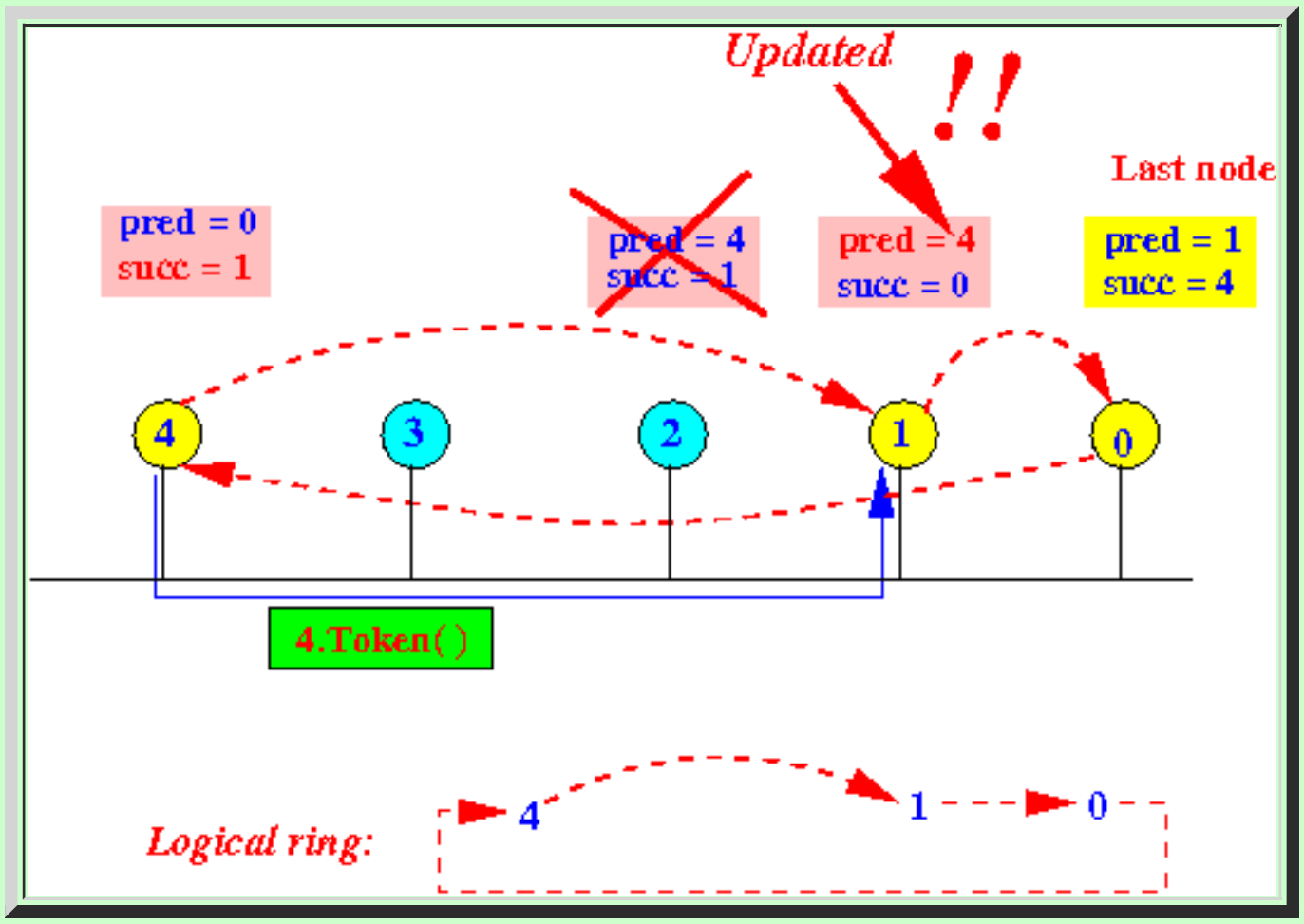
- **Node 4** received `2.Set_successor( 1 )` will **update** `Successor = 1`:



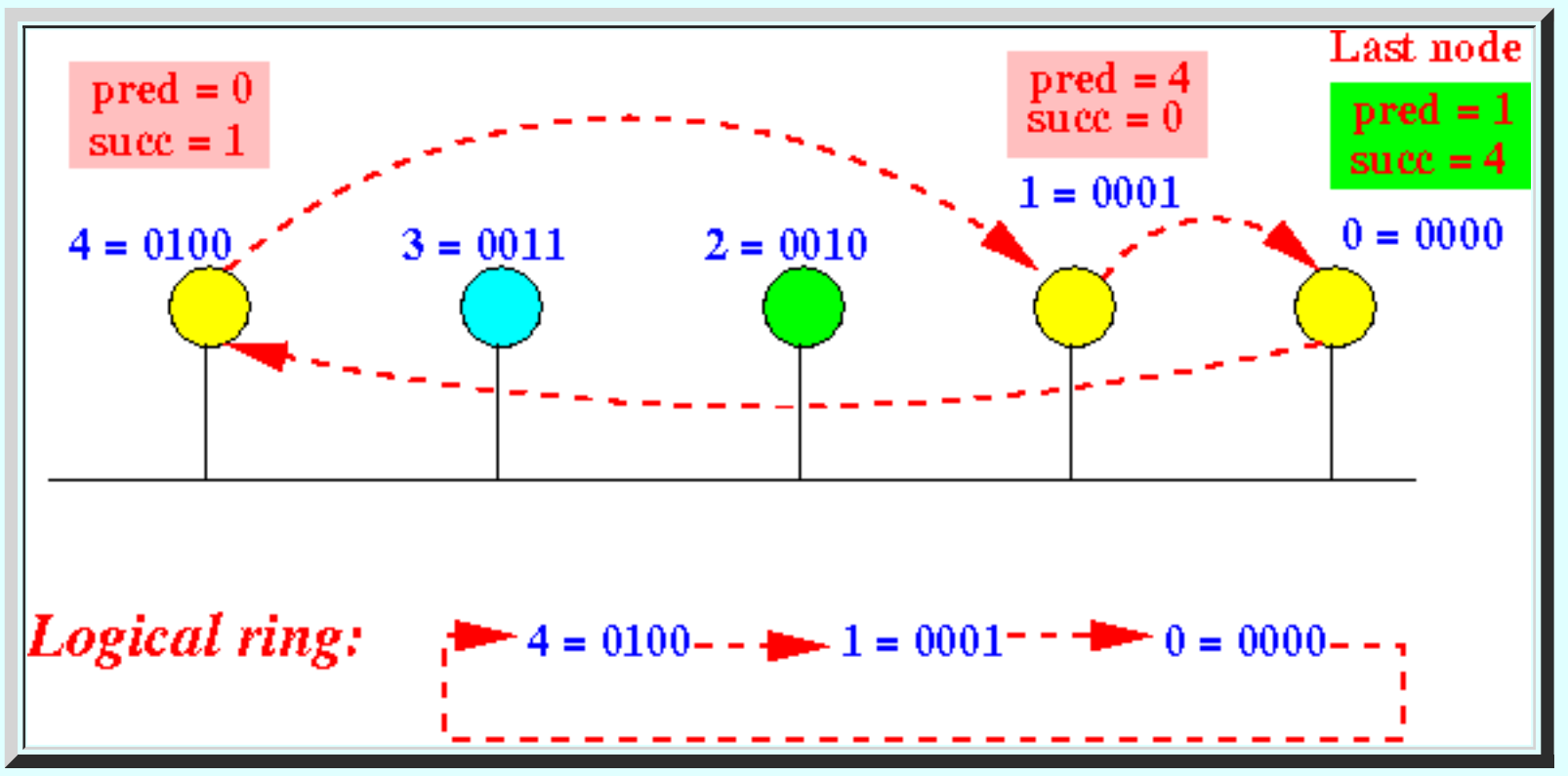
Notice that:

- We **still** need to **update** `predecessor` in the **node 1** !!!

- **Node 4** will **send (pass)** the **token message** `4.Token( )` to its ***new*** **successor node 1**:



3. **Done**:



Note:

- The **ring** is **complete** !!

- **Note:**

  - **Node 2** that has **left** the **logical ring** accomplished the **following**:

    - **Node 2** made ***its*** **successor** the ***new*** **token holder** !!!

Node insertion protocol of the Token bus

- **Inserting a node into the logical ring**
  - **Facts:**

    - A **node** that is **member** of the **logical ring** can ***invite*** a **non-member node** to **join** the **logical ring** (and become **part** of the **token bus network**)

      - Each **member node** of the **token bus network** has a **timer**

      - When the **timer *expires***:

        - The **node** will **send** an ***invite* message** to **invite** a ***range* of nodes** to **join**

- **Node insertion in Token-bus**
  - ***Simplifying* assumption:**

    - To **simplify** the **discussion** on **node *insertion***, we **assume** that:

      - There is **exactly *one* node** that wants to **join** the **token bus network**

  - The **Insertion *protocol*** (with **example**):

    1. **Invitation step**:

       - A **node** `x` that is **not** the ***last* node** will send the **following control frame** to **invite non-members** to **join**:

         ```
         X.Solicit_successor1( X.succ )

              will invite nodes in range (X .. X.succ) to join
         ```

       - The ***last* node** `L` will send the **following control frame** to **invite non-members** to **join**:

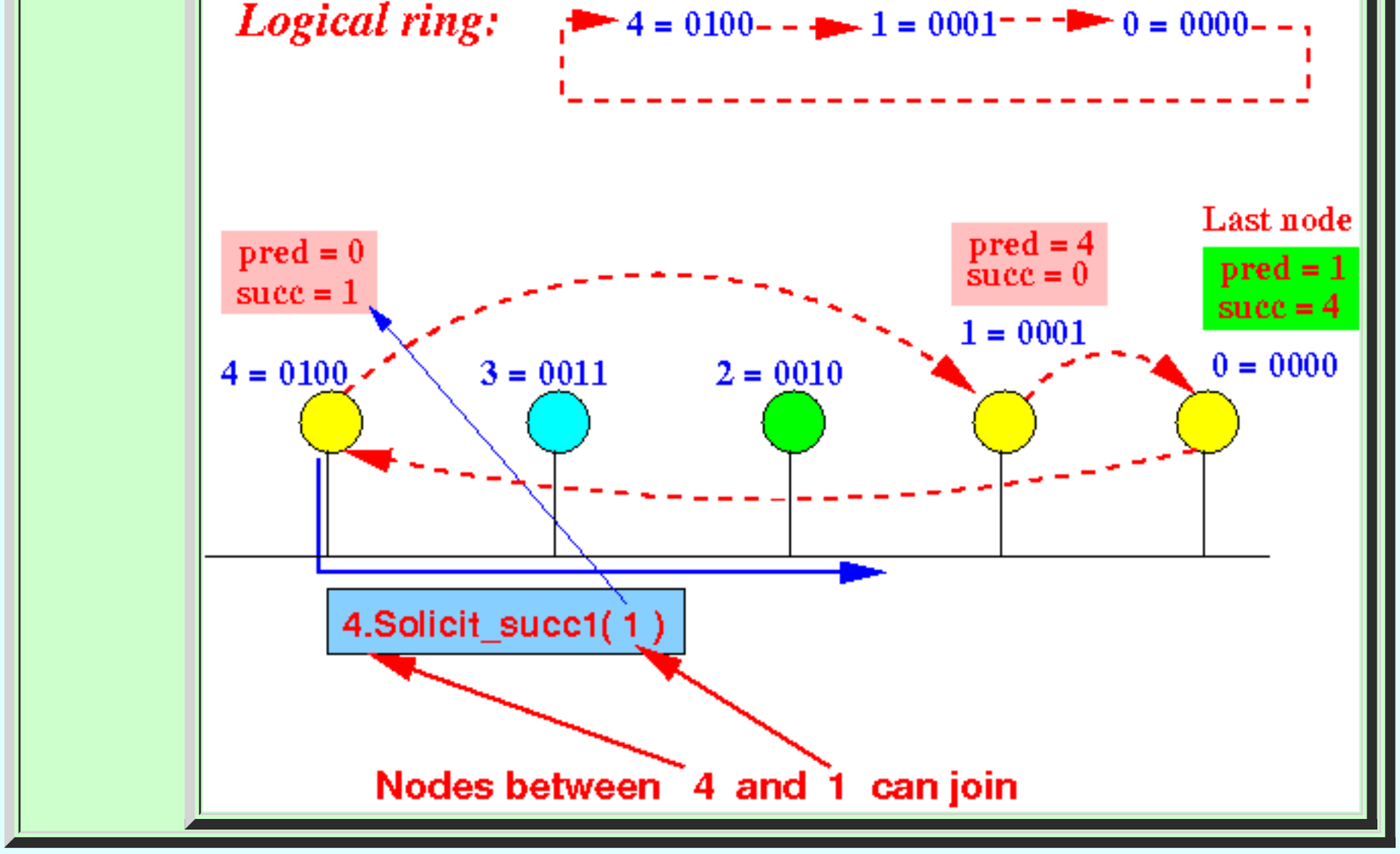
         ```
         L.Solicit_successor2( L.succ )

              will invite nodes in ranges (∞ .. L.succ)
                                      and (L .. 0)      to join
         ```

    Example:

    - **Current nodes** in **logical ring**:

      ```
      +--> 4  -->  1  --> 0 --+   (4 is token holder)
      |                       |
      +-----------------------+
      ```

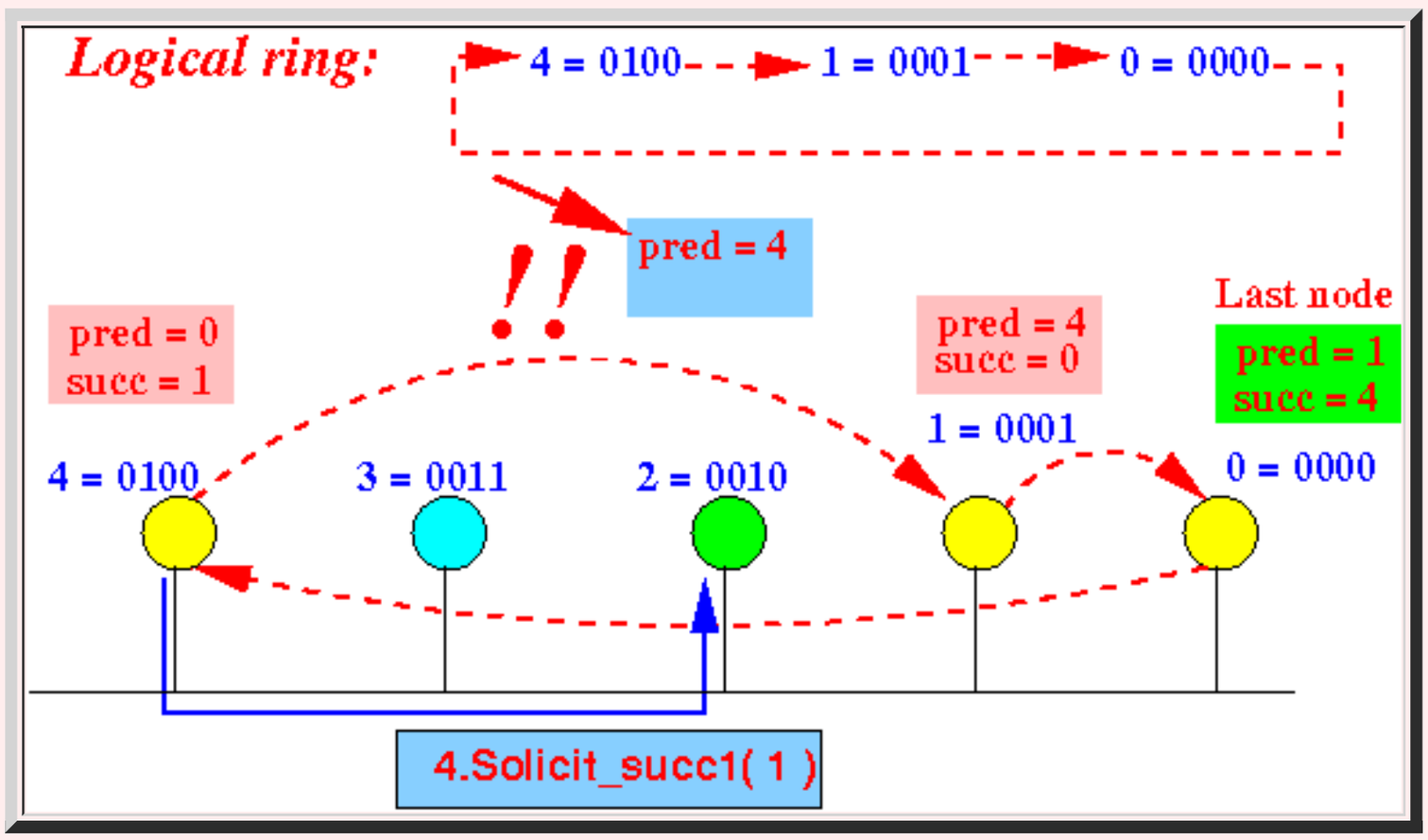
      Node 4 sends out an **invitation** using **Solicit_successor1(1)** :

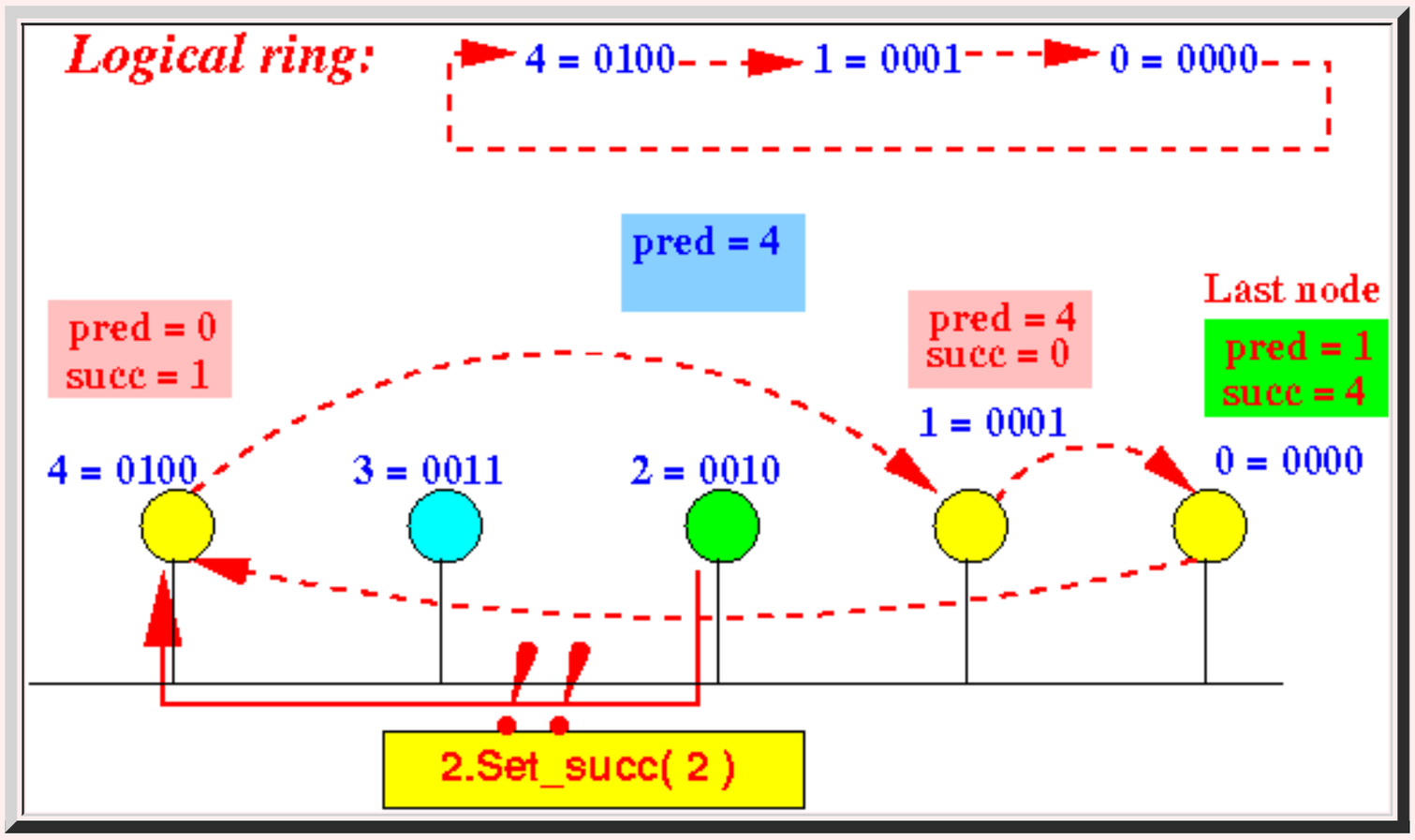


2. **Node insertion step** (or **node joining step**):

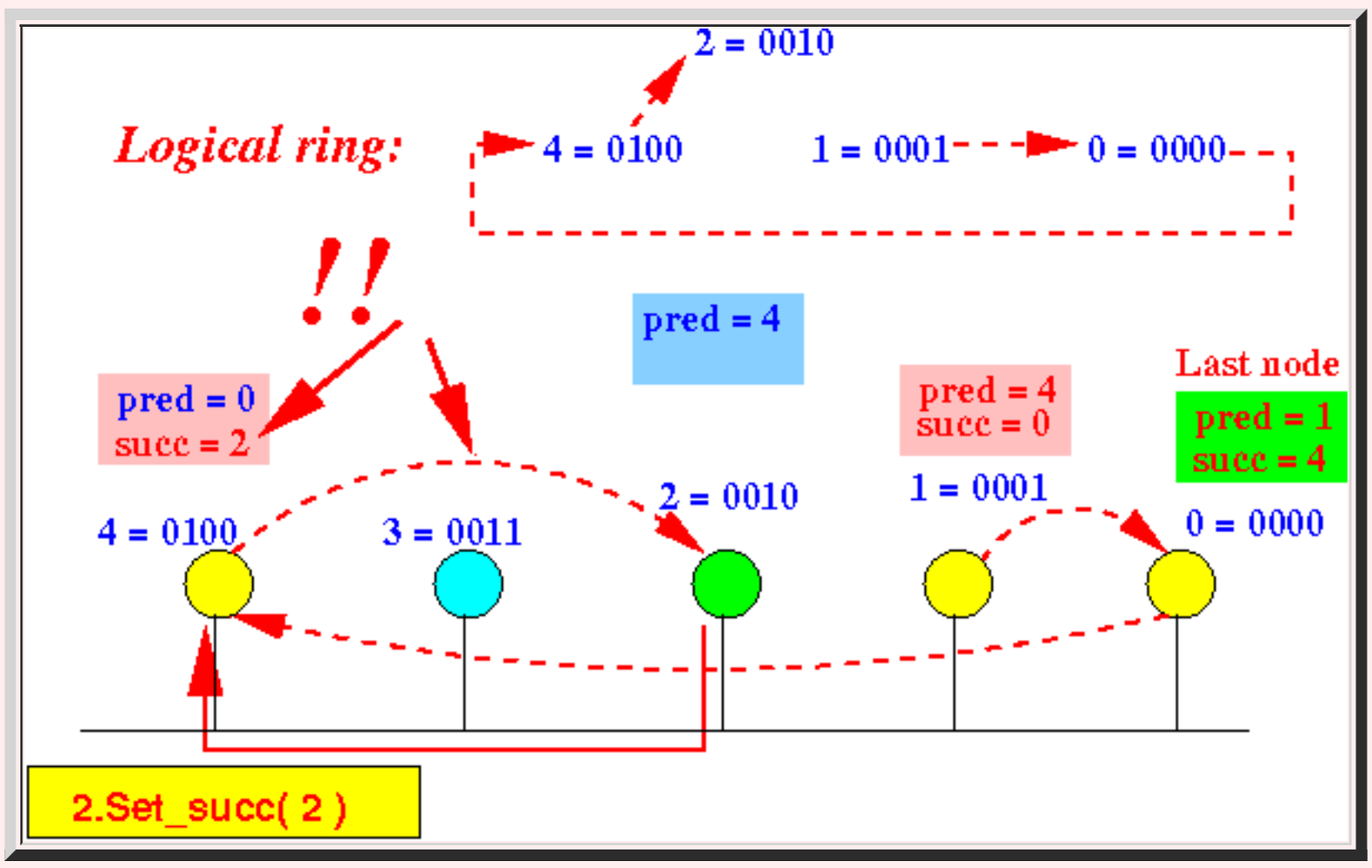
> 1. The ***joining*** **node** will:
>
> > 1. **Update** its `Predecessor` **variable** to the ***inviter's*** **successor** (= `x.succ`)
> >
> > 2. **Send** the `Set_successor( own_ID )` **message** to the ***inviter*** **node**
>
> 2. When the ***invitor*** **node receives** the `Set_successor(joining_ID)` **message**, the ***invitor*** **node** will:
>
> > 1. **Update** its `Successor` **variable** to the ***joining*** **node ID**
> >
> > 2. **Send** the `Set_successor(invitorSucc)` to its ***new*** **successor**

Example: **Suppose node 2 joins** (i.e.: **node 2** is the ***joining*** **node**)

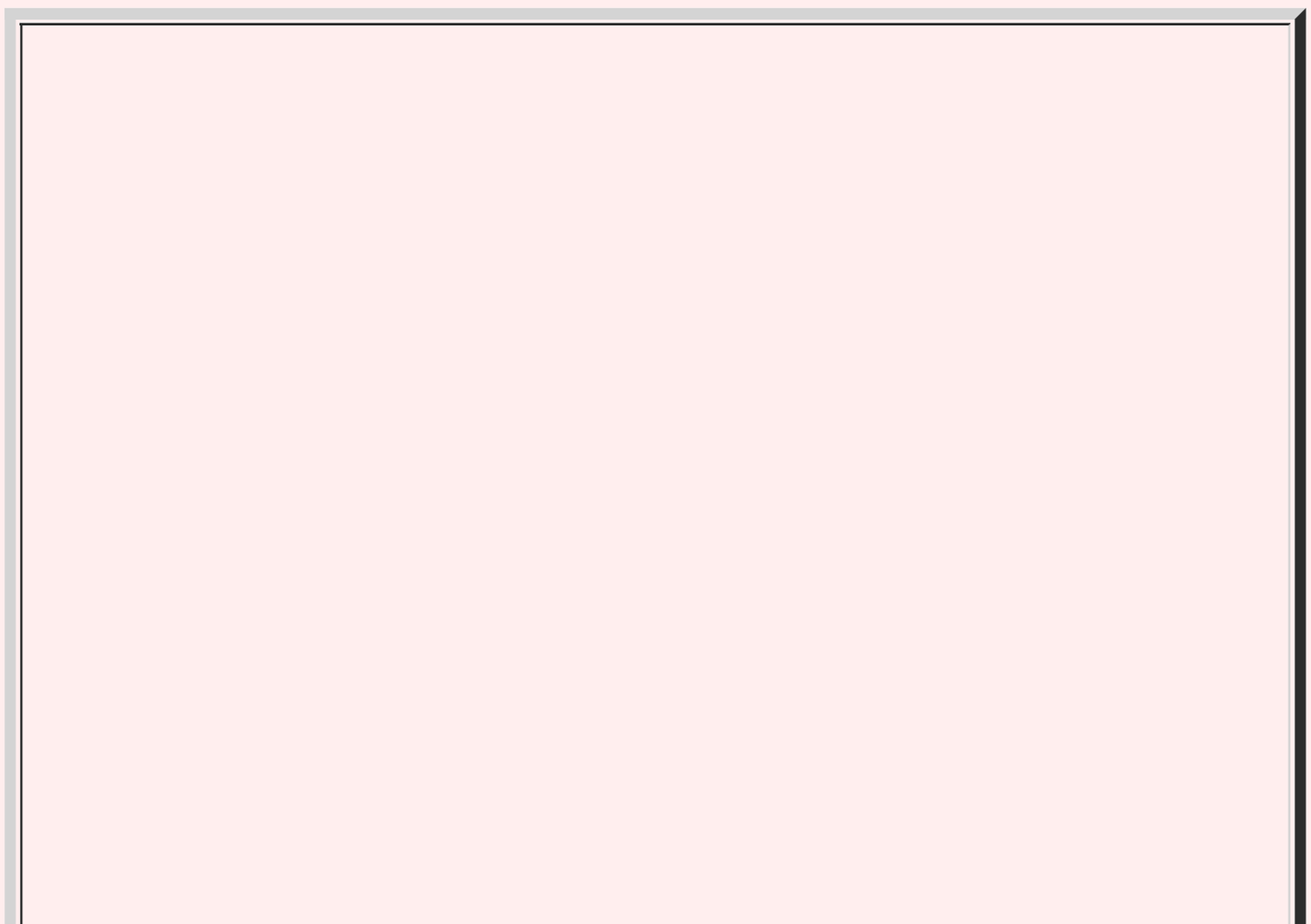
> 1. **Node 2 updates** its **variable predecessor = 4**:
>
> 
>
>
> 2. **Node 2 then sends** the **message** `Set_successor( 2 )` to **node 4**:

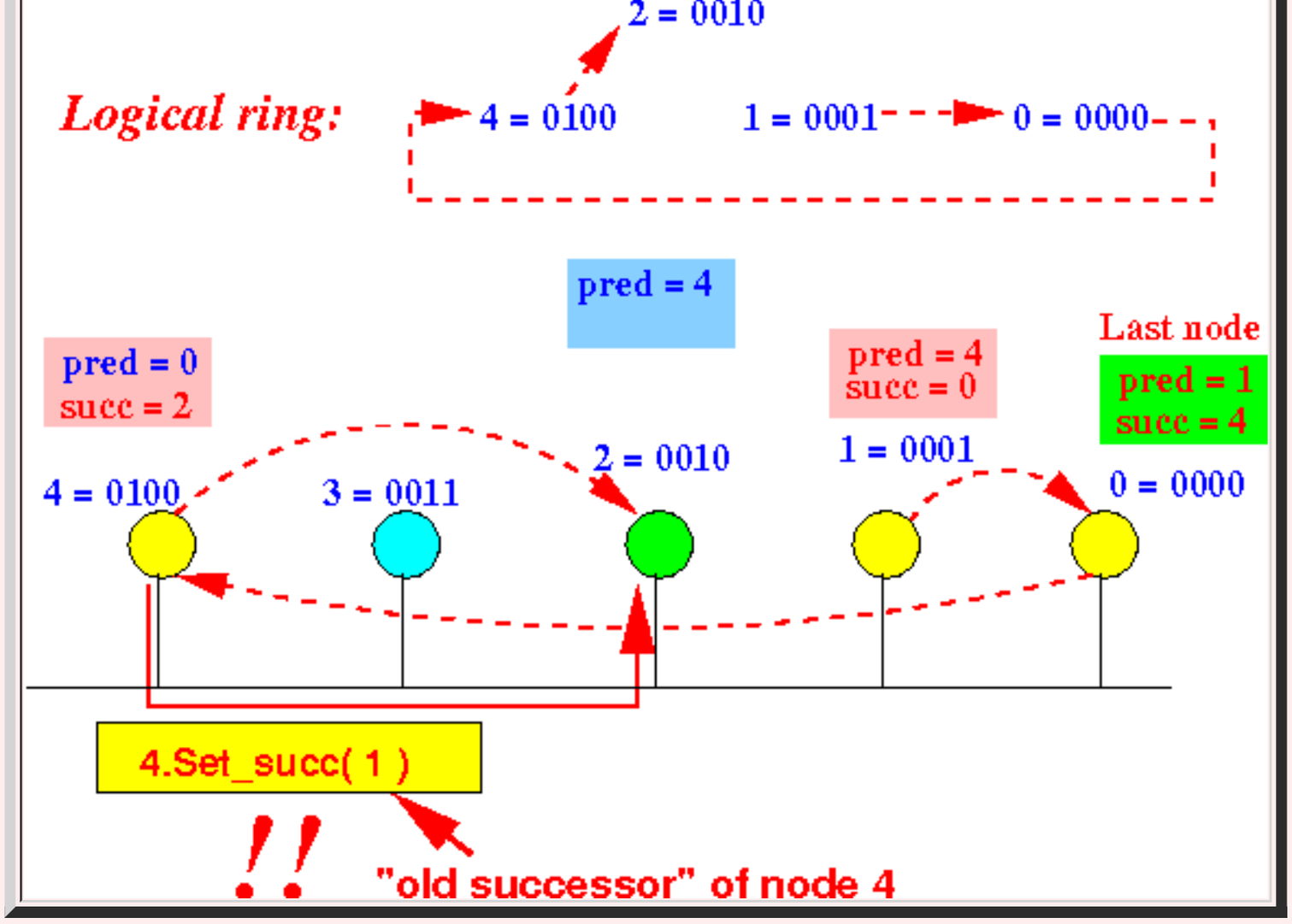


3. **Node 4** will **update** its **`Successor = 2`**:



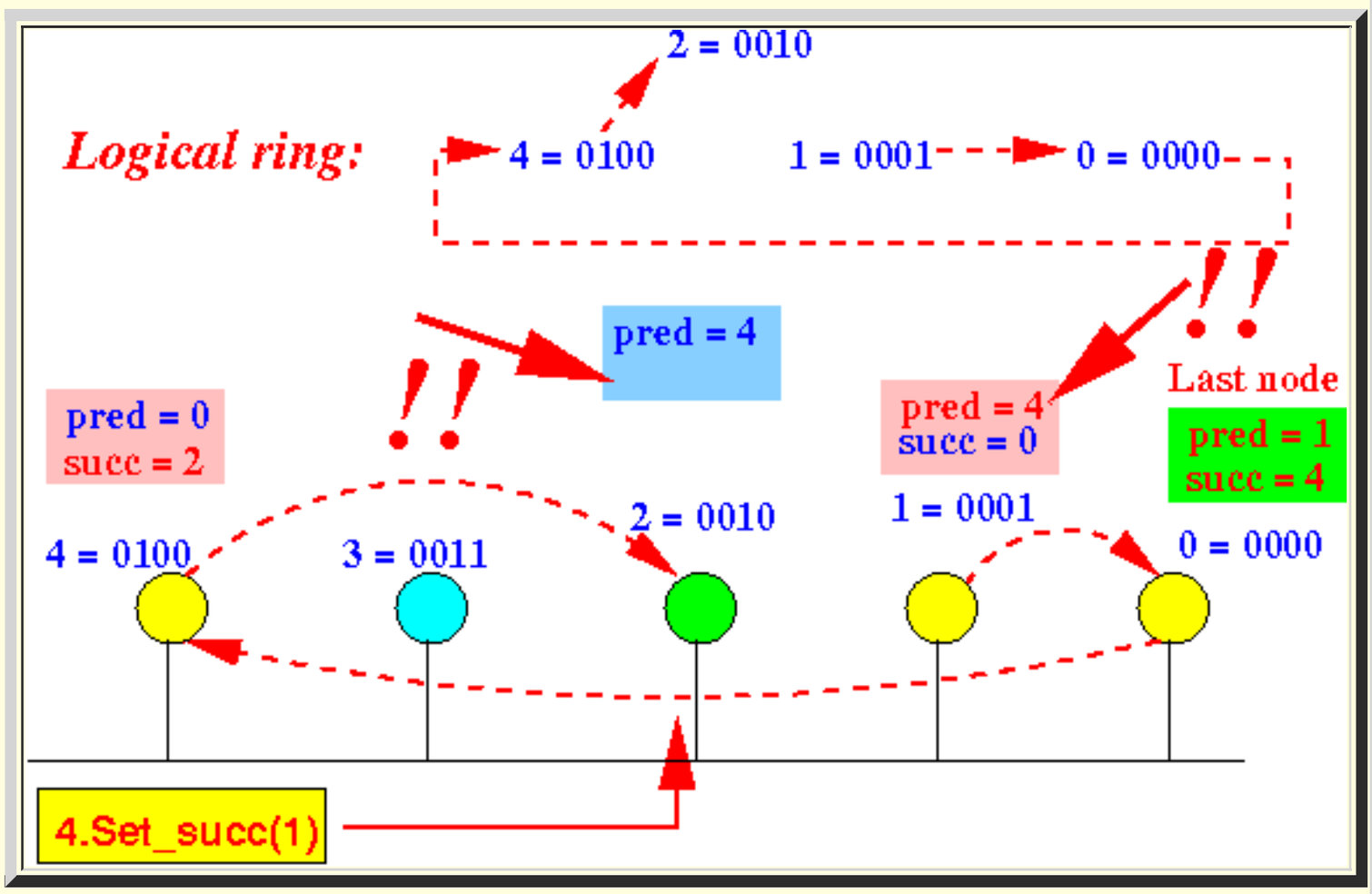
4. **Node 4** will **send** the **`"4.Set_successor(1)"`** **message** its ***new*** **successor (node *2*)**:





Note:

- The ***logical* ring** is **still *not* complete**:



Node 2 must still **set** its **successor** to **node 1**
Node 1 must **update** its **predecessor** to **node 2**

- **Processing** the `4.Set_successor(1)` **message** will **complete** these **updates** !!!

3. **Completion step:**

- When the ***joining* node** receives the `Set_succesor( )` **frame**:

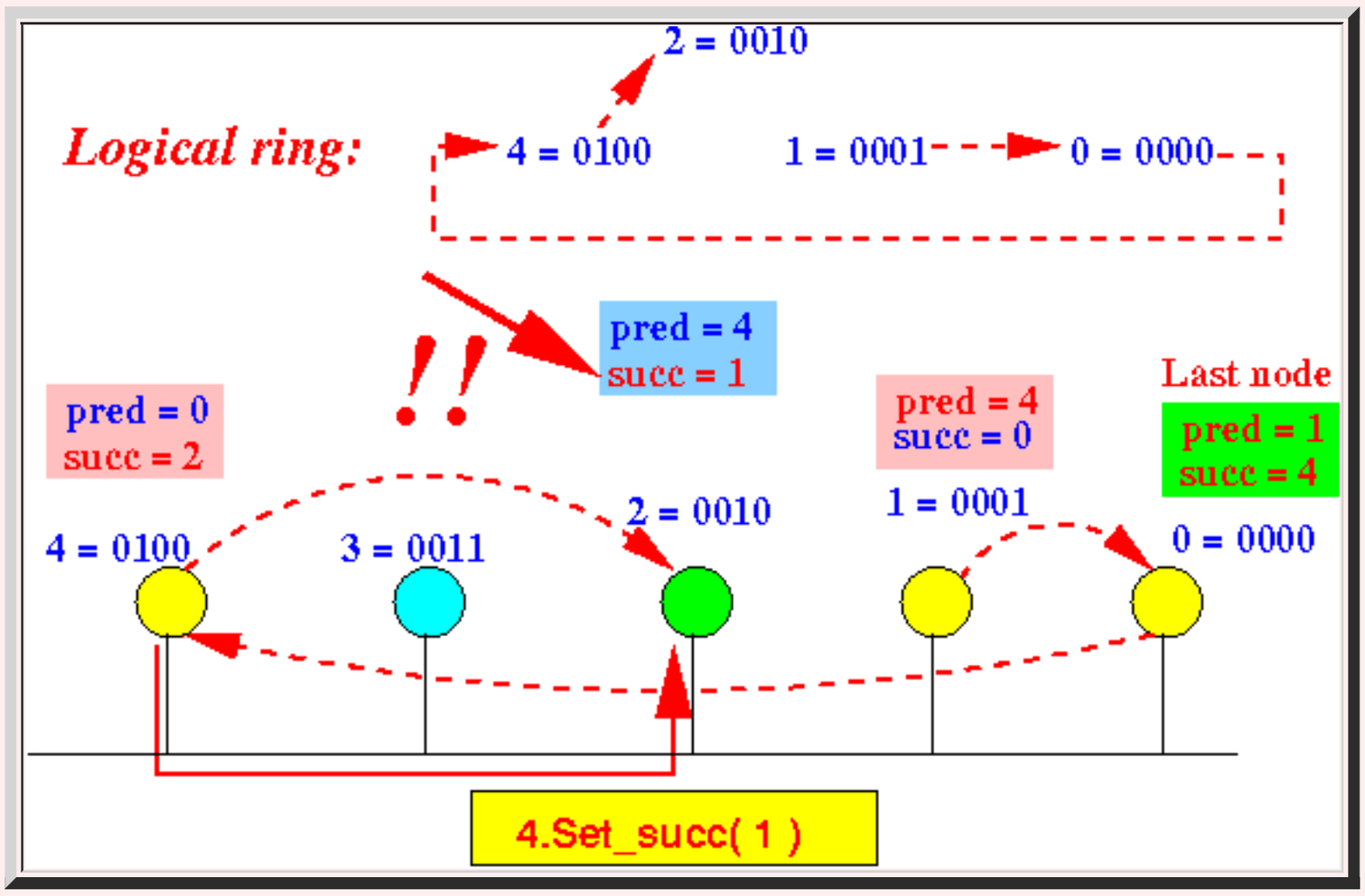
1. The ***joining* node** set its `succesor` **= invitor node ID**
2. Then ***joining* node sends** a `token` to its ***new* successor node**

3. When its ***new*** **successor node receives** the `token`, it will **update** its **successor**

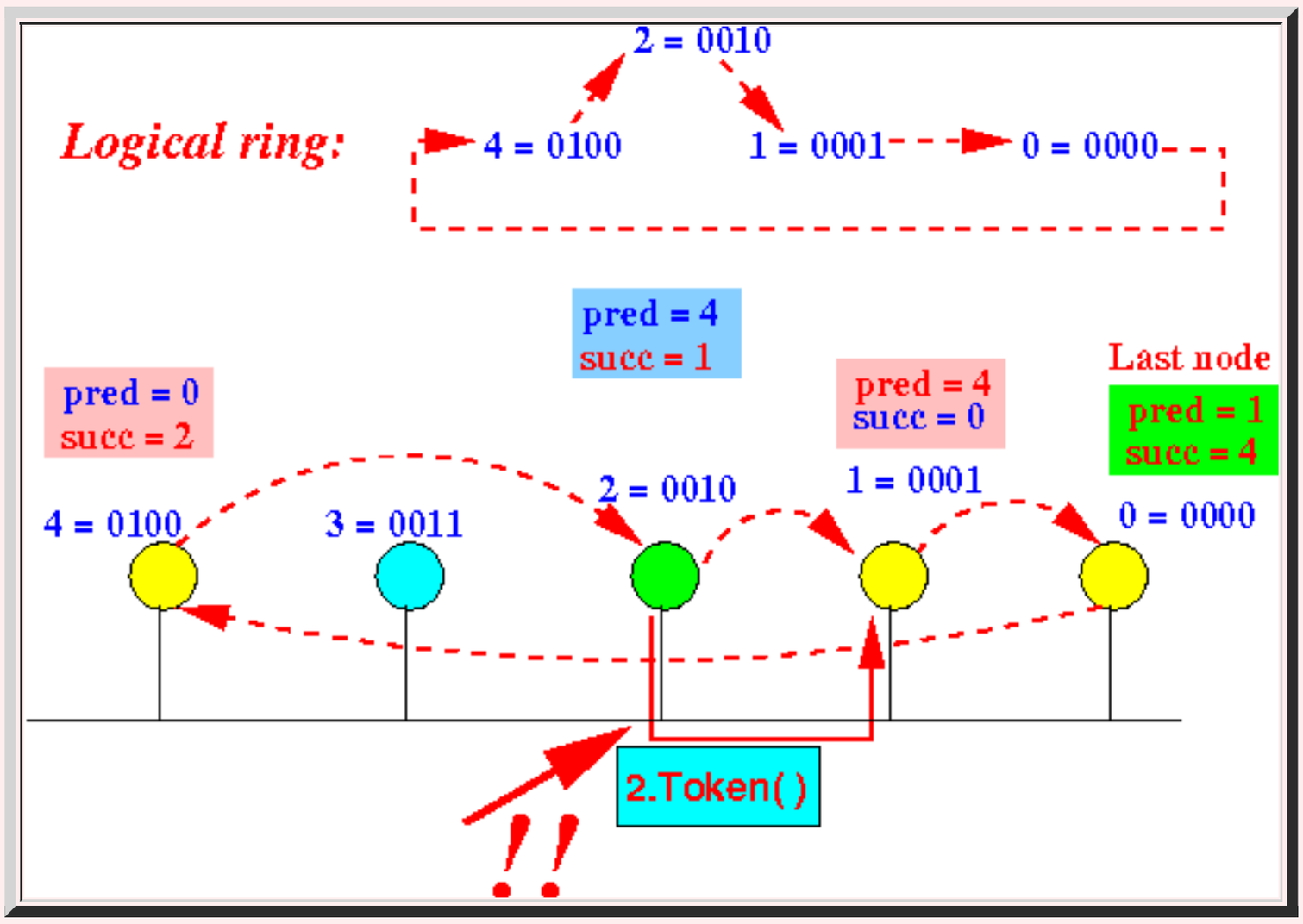
- The **logical ring** is then **complete** !!!

Example: continued with the above...

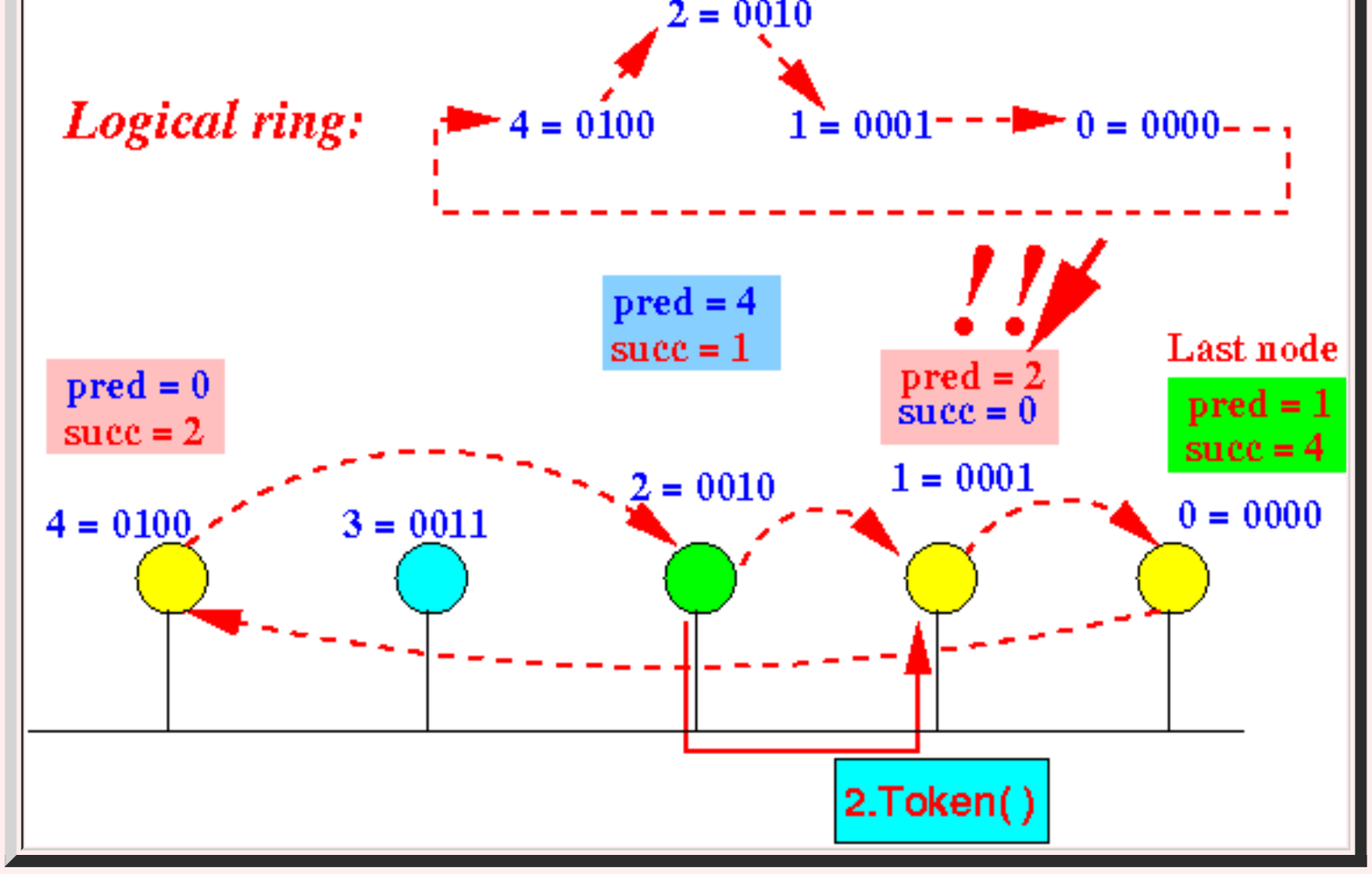
- **Node 2 received** the `4.Set_successor( 1 )` **message** and **updates** its **successor variable**:



- **Node 2** then **sends (= passes)** the **token** (in `2.Token( )` **message**) to its **successor**:



- When **node 1** has **received** the **token** (in `2.Token( )` **message**), **node 2** will **update** its **predecessor**:



(**Because *only* your predecessor can pass a token to you !!!**)

- **Special cases: (1) no joining node and (2) too many joining bnodes**
  - There are **2 special cases** that the **node insertion protocol** must **handle**:

    1. There are ***no* (= zero) nodes** that wish to **join** the **token bus network**

    2. There are ***more than 1* node** that wish to **join** the **token bus network**

- **The node insert protocol of the Token-bus --- no reply case**
  - **Fact:**
    - When there are ***no* nodes** that wish to **join** the **token bus network**:
      - There will be ***no* reply** for the `Solicit_successor(1 or 2)` **message** !!!
  - **Solution:**
    - The **invitor node** sets a **time out**
    - When **no reply** is **received** before **time runs out**:
      - The **invitor node** simply **passes** the **token** to its ***current* successor**

        (Because **no *new* node** wants to be ***its* successor** !!!)

- **The node insert protocol of the Token-bus --- the *multiple* reply case**
  - **Recall:**

- The `X.Solicit_successor(1 or 2)(Y)` **message** will **invite** a ***range* of nodes** to **join** the **logical ring**

  - It is **possible** for ***multiple* nodes** to **join (= send control messages)** at the ***same* time** !!!

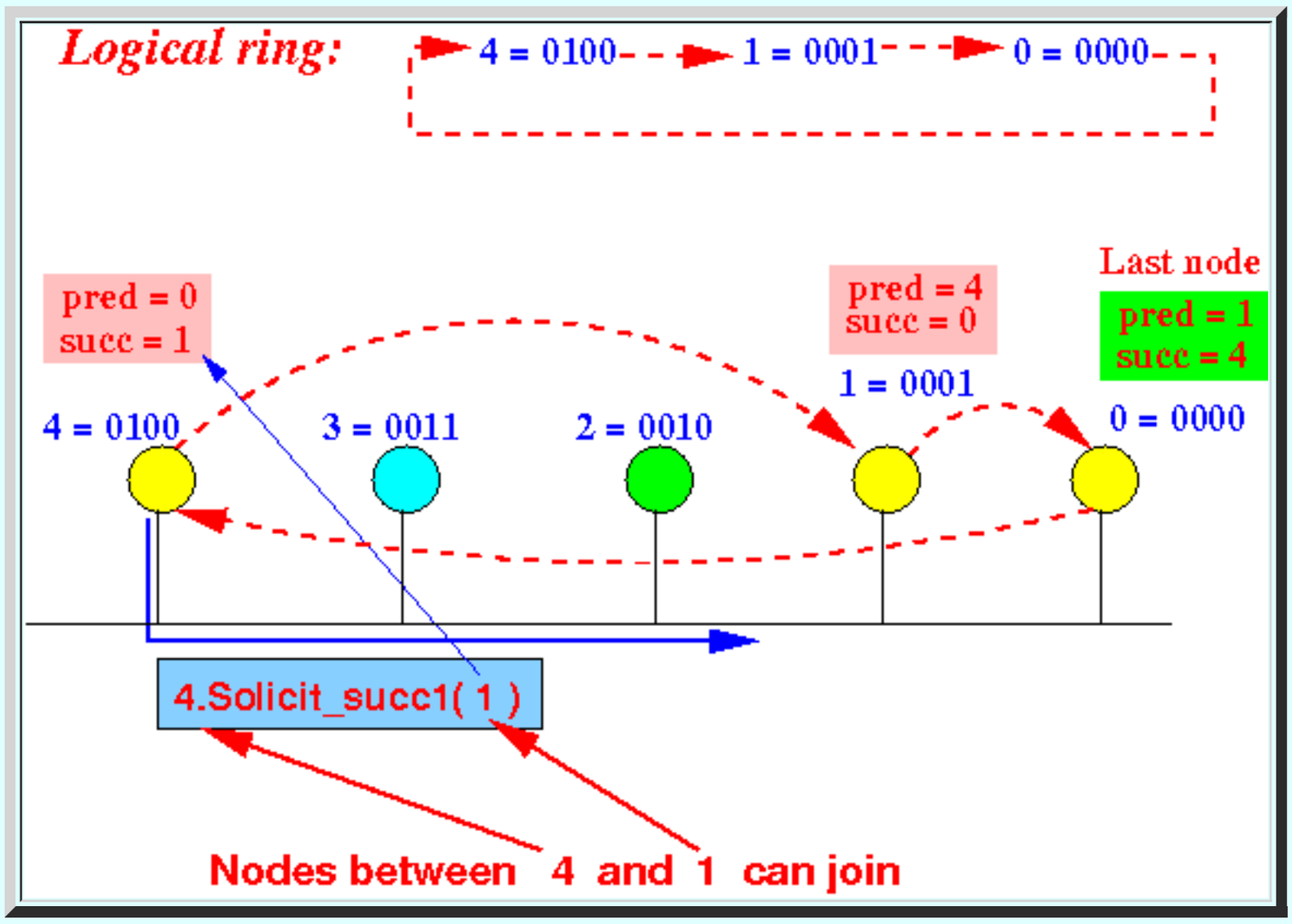
- The **Node *insertion* protocol** of the **token bus**: the ***multiple* reply** case

  - If ***multiple* nodes** try to **join** the **token bus network**, then:

    - Their **control messages** will result in a ***collision*** on the **token bus**

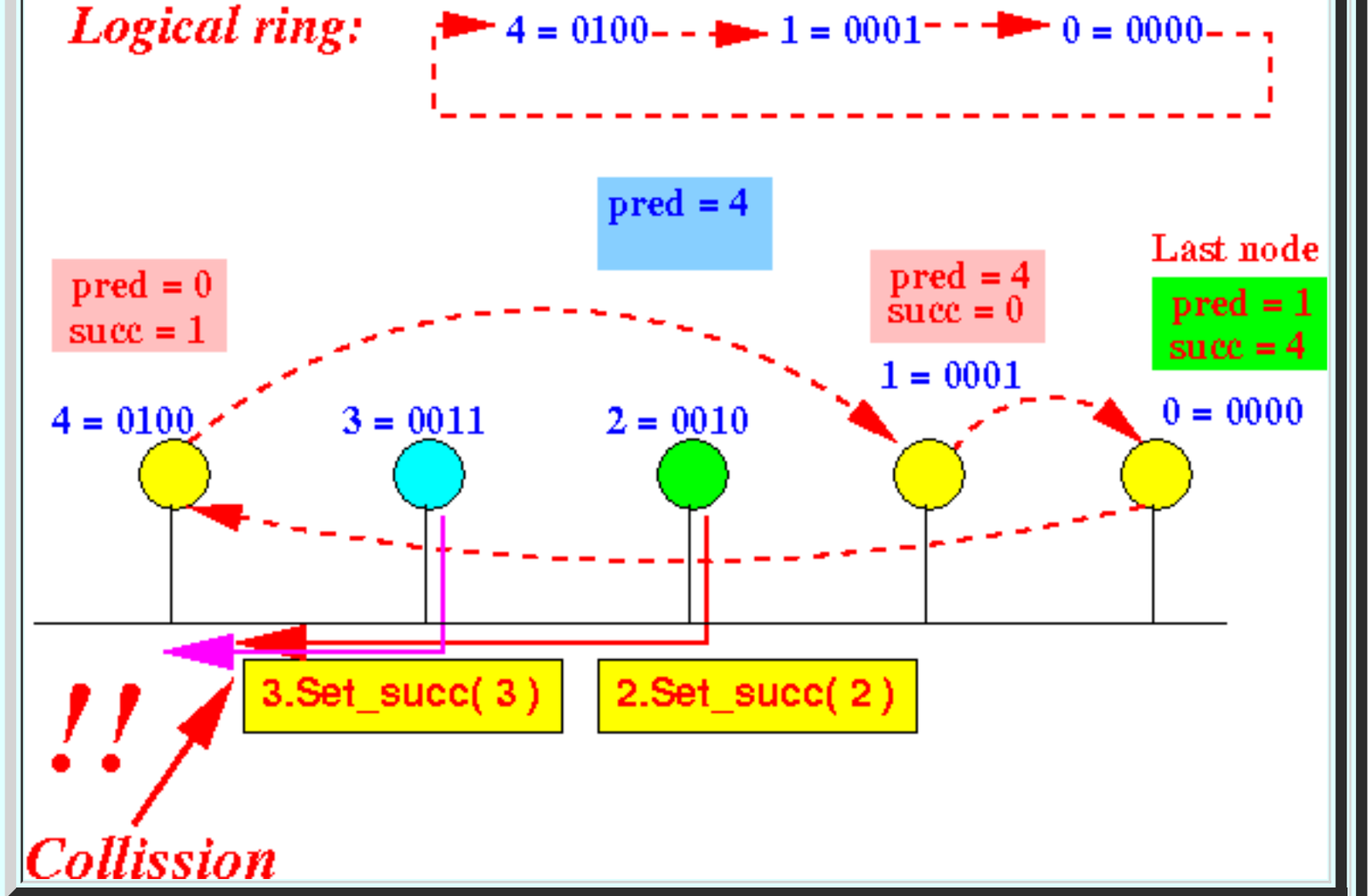
Example:

- **Node 4** sends `Solicit_successor` **control frame**:



Nodes 2 *and* 3 are **invited** to **join** !!!

- When **both node 2 *and* 3** tries to **join** (by **sending** a `Set_successor( )` message):



- The **transmitting nodes** will **detect the collision** and then **execute**:

  - A **collision *resolution* protocol** to:

    - **Select** a **"winner"** among the ***colliding*** **nodes**

  - The **winner** will then **execute** the **insertion protocol** (discussed **above**)

  We will **discuss** the **collision resolution protocl** in the **next webpage**

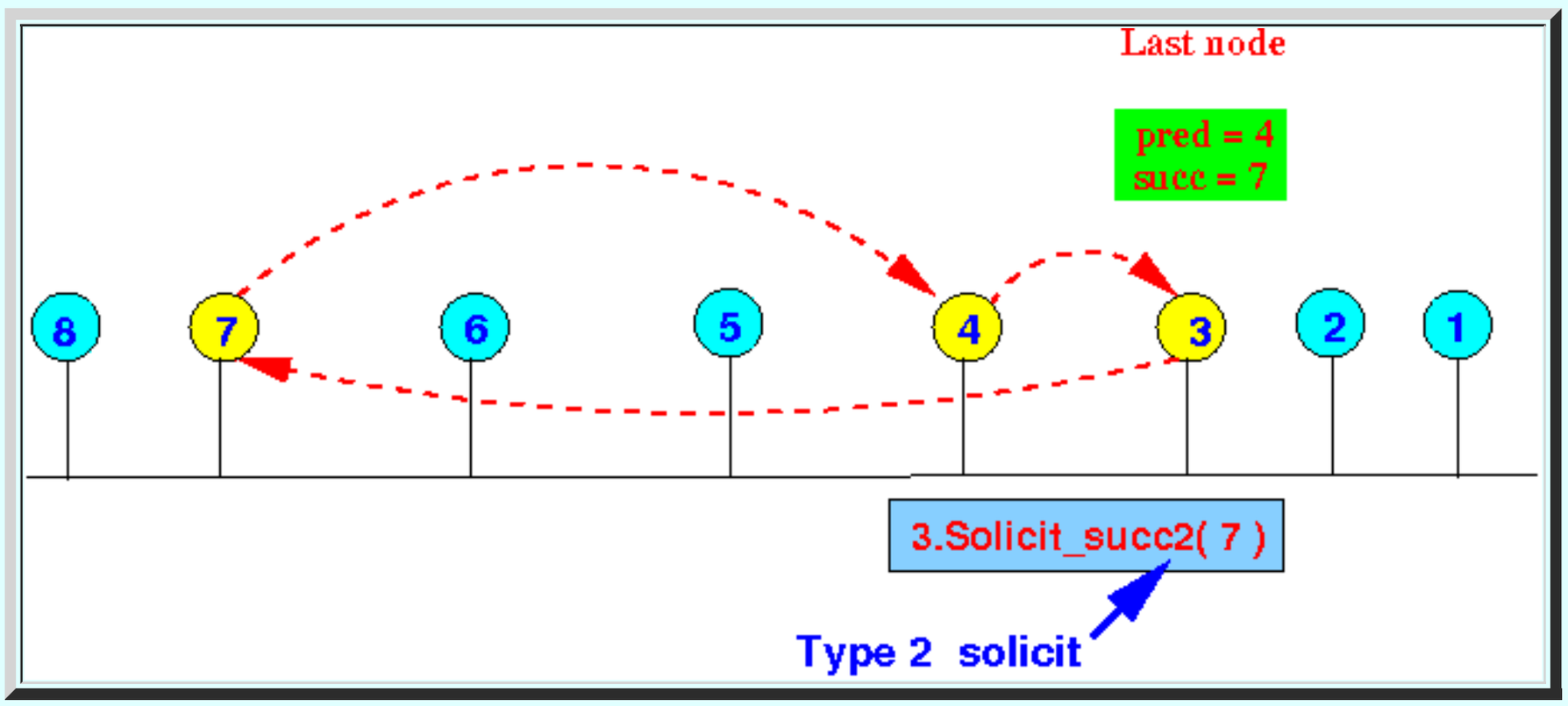
- **The last node in the token ring**

  - **Comment**:

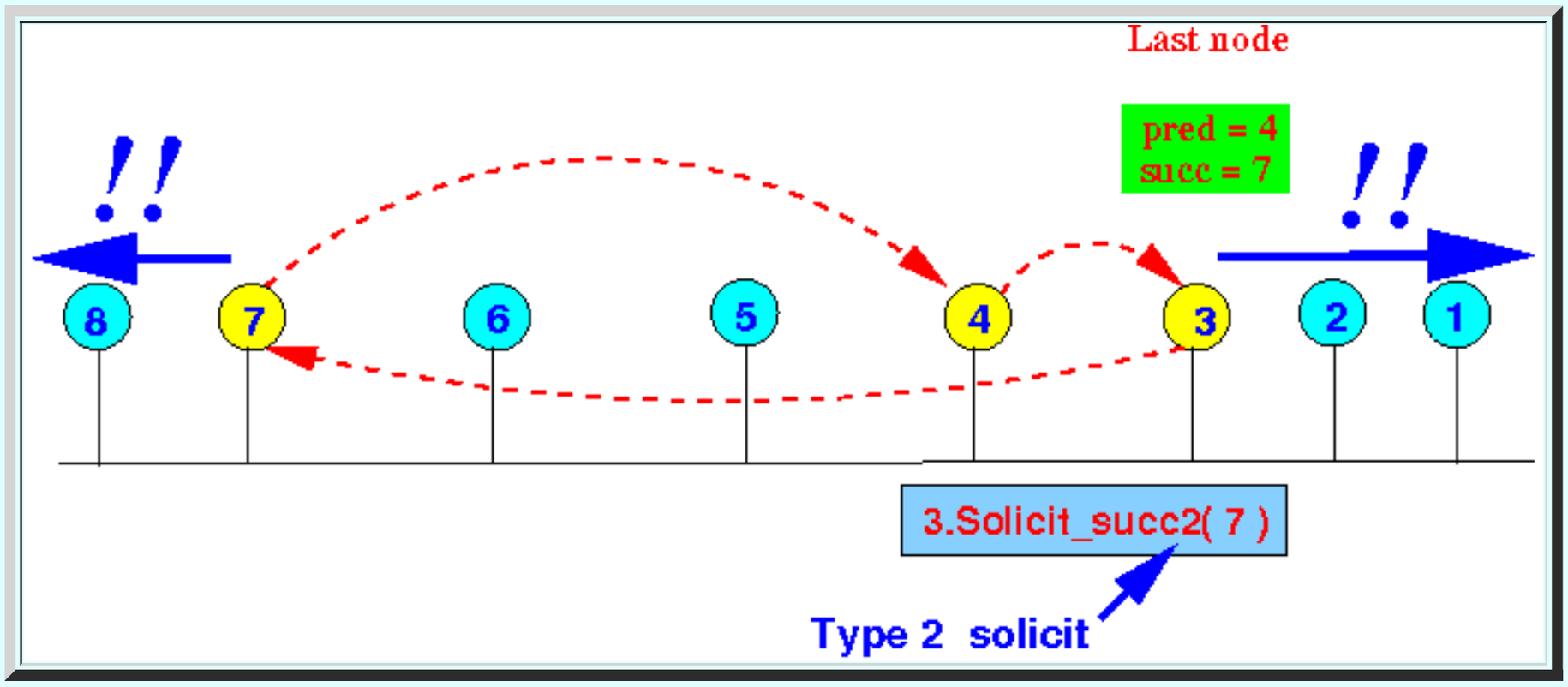
    - The **last node** will send the ***second*** **type** of **Solicitation message**:

      ```
      Solicit_successor2( )
      ```

Example: **last node 3** sends **"`3.Solicit_successor2(7)` "**



The **3.Solicit_successor2(7)** message will **invite** the **following nodes** to **join** the **token bus**:



Nodes 1, 2 (with **id < 3**) and **node 8** (with **id > 7**) are **invited**

Furthermore:

- The **insertion protocol** used is **identical** to the **insertion protocol** used for `Solicit_successor1( )` described above...

The collision resolution protocol of the Token Bus network

- **Collisions on the token bus network**
  - **Recall:**

    - ***Multiple* nodes** can be **invited** to **join** the **token bus network**

    - When ***multiple* nodes** tries to **join**:

      - Their ***simultaneous* message transmissions** will result in a **collision**

  - **Cause** of the **collision**:

    - **Collisions** on the **token bus network** is **cause** by:

      - ***Multiple* nodes believe** they have the **permission** to **transmit** on the **network**

  - ***Resolving* collision**:

    - **Resolving collision** = making the ***colliding* nodes** decide on ***which* (one) node** has the **permission** to **transmit** on the **network**

    Fact:

    - The **Token Bus network** defines a ***distributed* algorithm** to ***resolve* a collision**

- **The collision resolution protocol of the Token Bus network**
  - The ***Collision Resolution* Protocol:** (Executed by **nodes** that are **involved** in the **collision**)

```
1. Let NodeID = address (ID) of the node in binary

2. Divide the NodeID into groups of 2 bits each:

      NodeID = 11223344556677.....
```

```
/* ============================================================
   This distributed algorithm will identify the node
   with the largest node ID !!!
   ============================================================ */
while ( true )
{
   x = next 2 bits in NodeID;    // x = bits 11 or bits 22, and so on

   Transmit a jam signal for (x + 1)*T sec  // T = end to end delay

   Listen on the transmission medium

   if ( transmission medium is busy for > T sec )
   {
      /* ============================================
         Some other node has an x value > my x value
         ============================================ */
      winner = false;    // Node will stop competing !!
```

```
            exit;
        }
        else ( transmission medium is busy for < T sec )
        {
            /* ================================================
               Some other node has an x value == my x value
               ================================================ */
            continue;       // Try again with next group of bits
        }
        else // The transmission medium is clear !!!
        {
            /* ================================================
               This node transmits the jam for the longest time !!
               I.e.: This node has the highest x value !!!
               ================================================ */
            winner = true;   // This node is the winner

            exit;
        }
    }
```

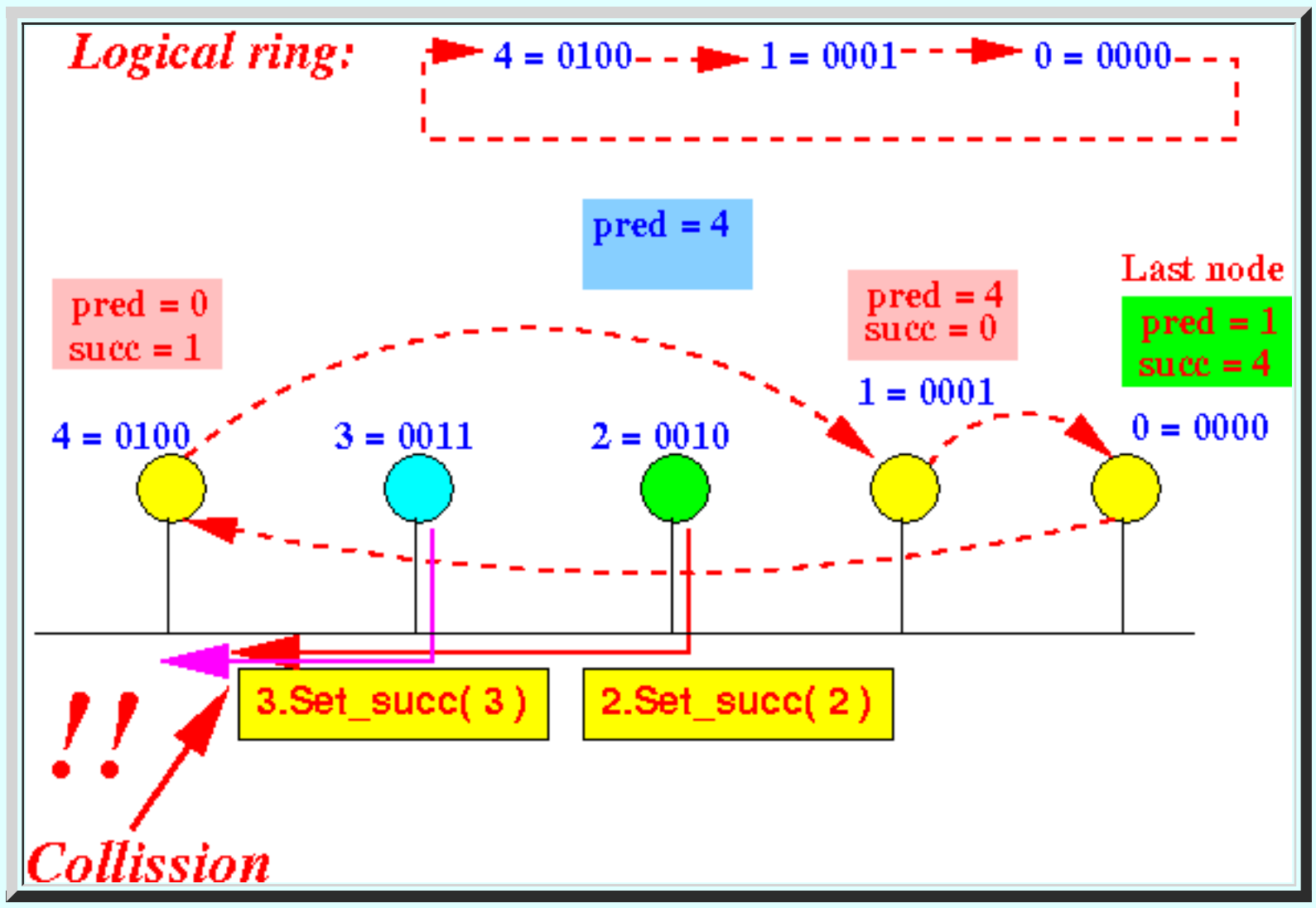
- ***Outcome*** of the **Collision Resolution Protocol:**

  - The **node** that has the ***largest value*** **for ID** will be the **winner** of the **Collision Resolution protocol** !!!

    (And ***this*** **node** will then **execute** the **node *insertion* protocol** to **add** itself into the **ring**)

- **Example:**

  - **Suppose** nodes **2** and **3** want to **insert** into the **ring**:



Their **transmissions** will **collide**....

  - We will use **4 bits** (**binary numbers**) for **node IDs** in the **example**

- **Round 1:**

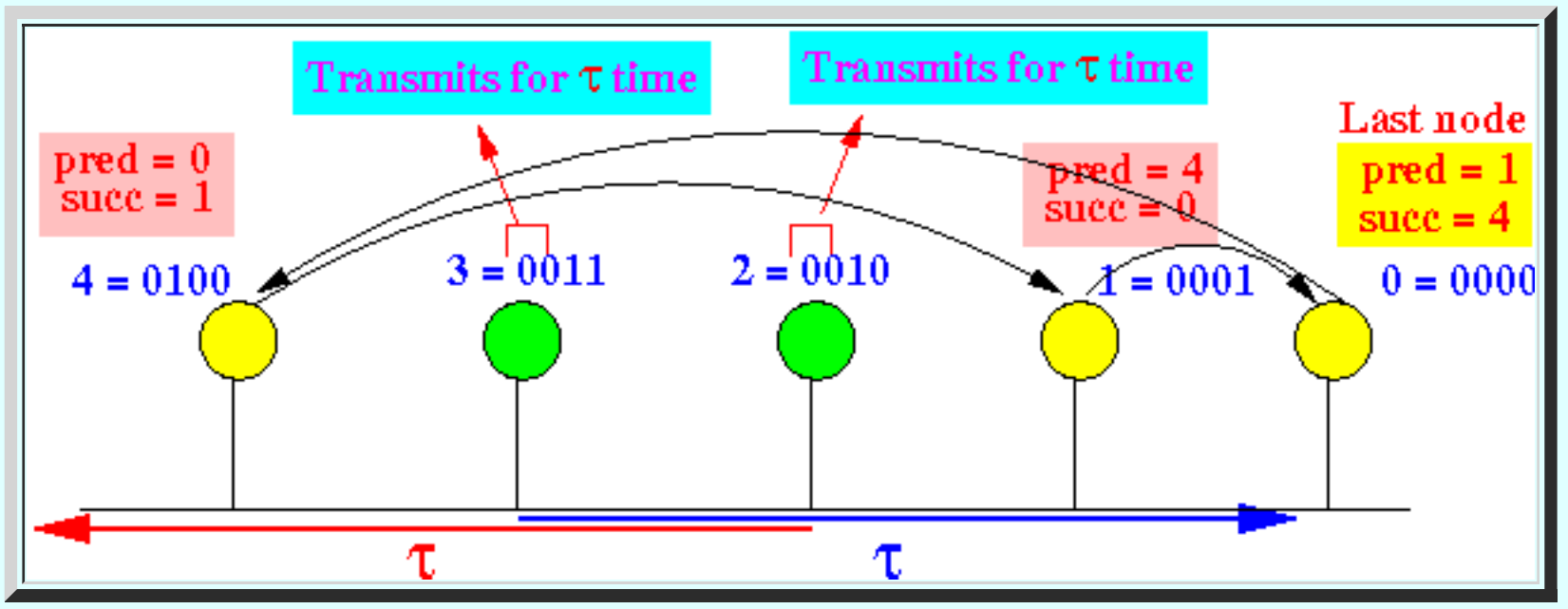
  - **Node 0010 (2)** will use **x = 00**

- **Node 0011 (3)** will use **x = 00**

Therefore:

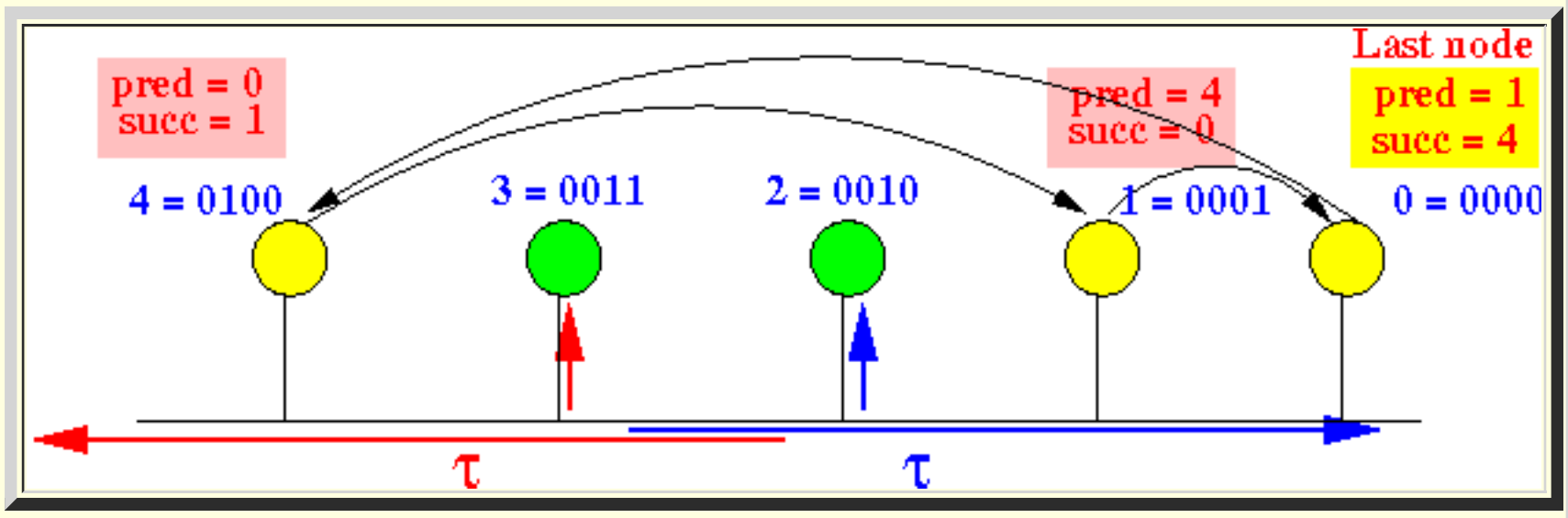
- **Node 0010 (2)** will transmit for **1×τ sec**
- **Node 0011 (3)** will transmit for **1×τ sec**

Graphically:



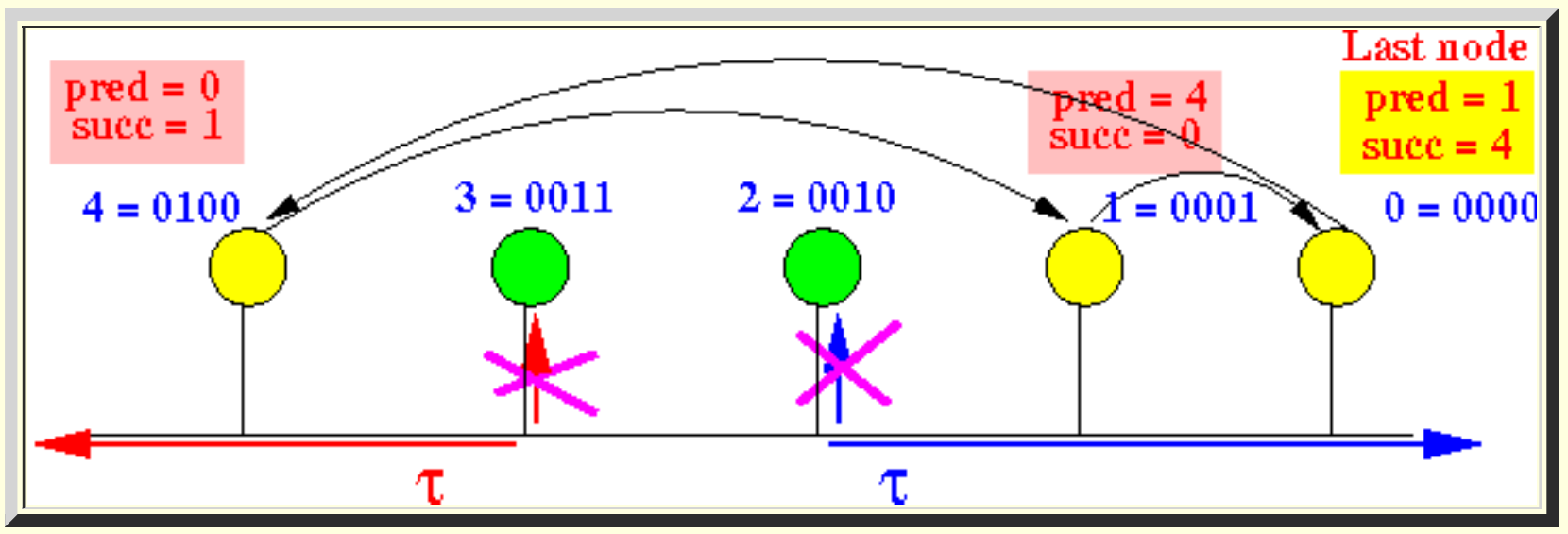
Then:

- ***Both* nodes** will ***listen*** on the **transmission medium**:



and **both nodes** will **detect** a **transmission**

- The **transmission** will ***end*** in **< τ sec**:



Therefore:

- The nodes **2 and 3** will **repeat** and use their **next group (pair)** of **bits**

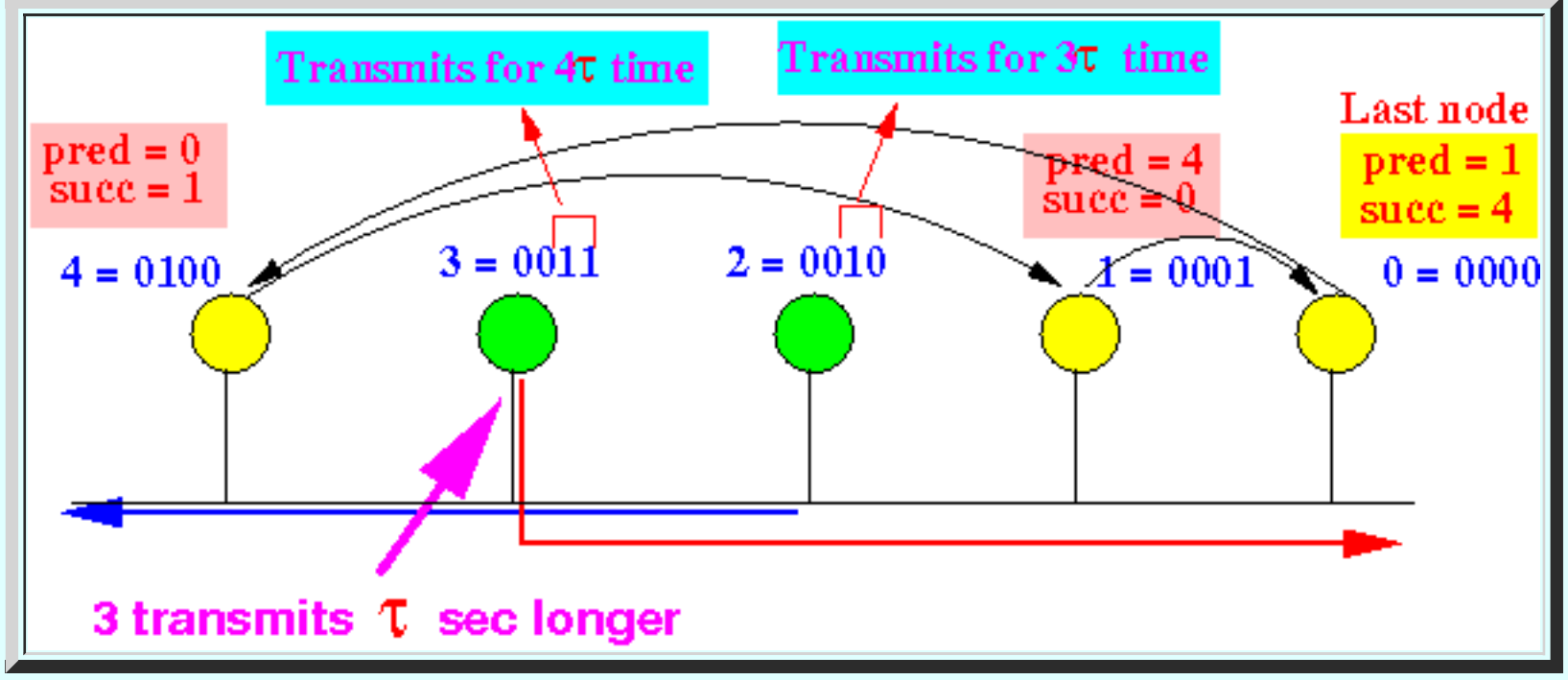
- **Round 2:**

- **Node 0010 (2)** will use **x = 10 (= 2)**
- **Node 0011 (3)** will use **x = 11 (= 3)**

Therefore:

- **Node 0010 (2)** will transmit for **3×τ sec**
- **Node 0011 (3)** will transmit for **4×τ sec**

Graphically:



Result:

- **Node 2** will hear a **transmission** for **> τ sec after node 2** has **stopped**:
  - **Node 2** will **withdraw** itself from the **collision resolution protocol**

- **Node 3** will hear ***no*** **transmission after node 3** has **stopped** !!!
  - **Node 3** will **detect** that **itself** is the ***winner*** !!!

Node initialization and token bus initialization

- **Node initialization**

  - **Node initialization**:

    - When a **node** first ***powers up***:

      - The **node** is ***not*** **a member** of the **token bus network** !!!

Observation:

- A ***starting*** **node** will **not know**:

  1. The **token bus network** has be **started** ***already***

     - In **this case**, the ***starting*** **node** must **join** the ***existing*** **token bus network**

  2. There is ***no*** **token bus network** running

     - In **this case**, the ***starting*** **node** must ***initiate*** the **token bus network**

  - **Node** ***initialization*** **protocol** of the **token bus network**:

    - When a **node** is **powered up**:

      - The **node** will **listens** on the **token bus** for a **time out period**

    - If the **node** **hears** a **transmission** within given **time out** period:

      - The **node** will ***wait*** for a `Solicit_successor` message to ***insert*** itself into the **token bus network**

- If the **node** does **not hear** a **transmission** within given **time out** period:

  - The **node** will ***execute*** the **token bus *initialization* protocol**

    (Discussed next)

- **Token Ring *initialization***

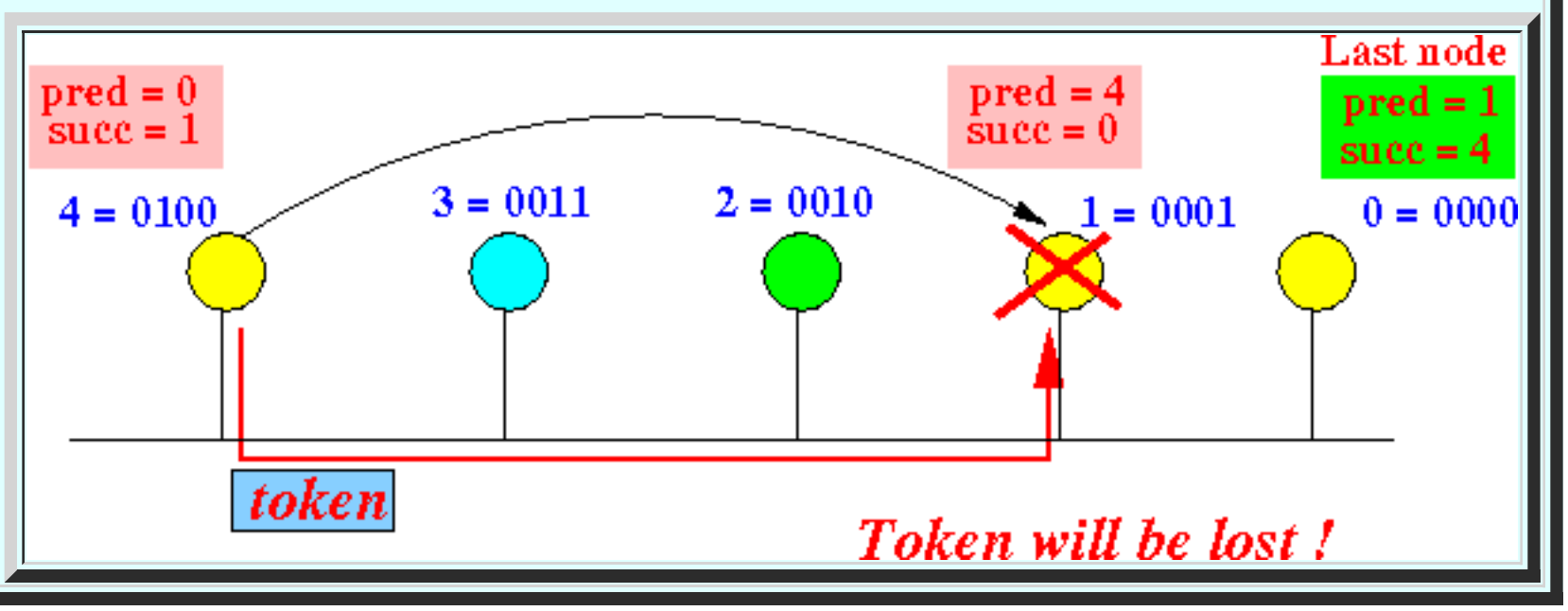
  - The **token ring initialization** protocol:

```
(Executed when a node is powered up and does not hear
 any transmission for Bus-Idle Timer period)

Transmit the Claim_token control frame;

if ( no collision )
{
   Node = token holder

   Invite other nodes to join token bus network;
}
else
{
   /* ======================================
      Multiple nodes try to claim token...
      ====================================== */

    Execute the collision resolution protocol;

    if ( winner == true )
    {
       Node = token holder

       Invite other nodes to join token bus network;
    }
    else
    {
        Wait for Solicit_successor control frame to join;
    }
 }
```

Recovery from node failure

- **Node failure**
  - **Effect** of a **node failure**:
    - When a **node fails**, the **token frame** sent ***to*** the **node** will be ***lost***:

      Example:



      Losing the **token** is ***detrimental*** to the **token bus network** !!!

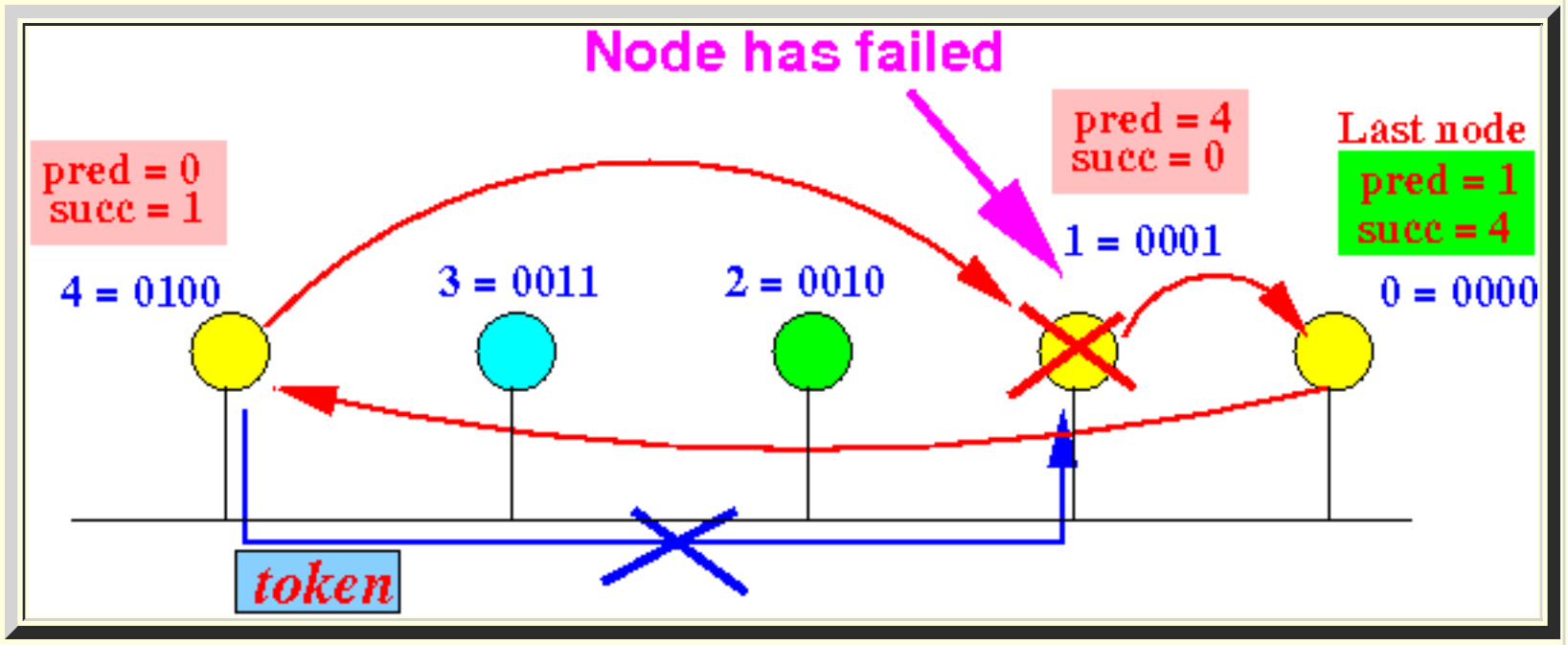
- **Protecting the token from being lost through node failure**
  - **Fact:**

    - **Nodes** must use the **token passing protocol** to **send** a **token (frame)** to its **successor**

  - The **token passing protocol** of the **token bus network**:

```
/* ============================
   1. Initial attempt
   ============================ */
Transmit token to successor node;

Set timer;

if ( transmission heard within time out )
   return;   // Token passed successfully

/* ============================
   2. Backup attempt
   ============================ */
Transmit token to successor node;

Set timer;

if ( transmission heard within time out )
   return;   // Token passed successfully

/* ============================================
   3. Give up: try to repair the logical ring
   ============================================ */
Execute the ring recovery protocol;
```

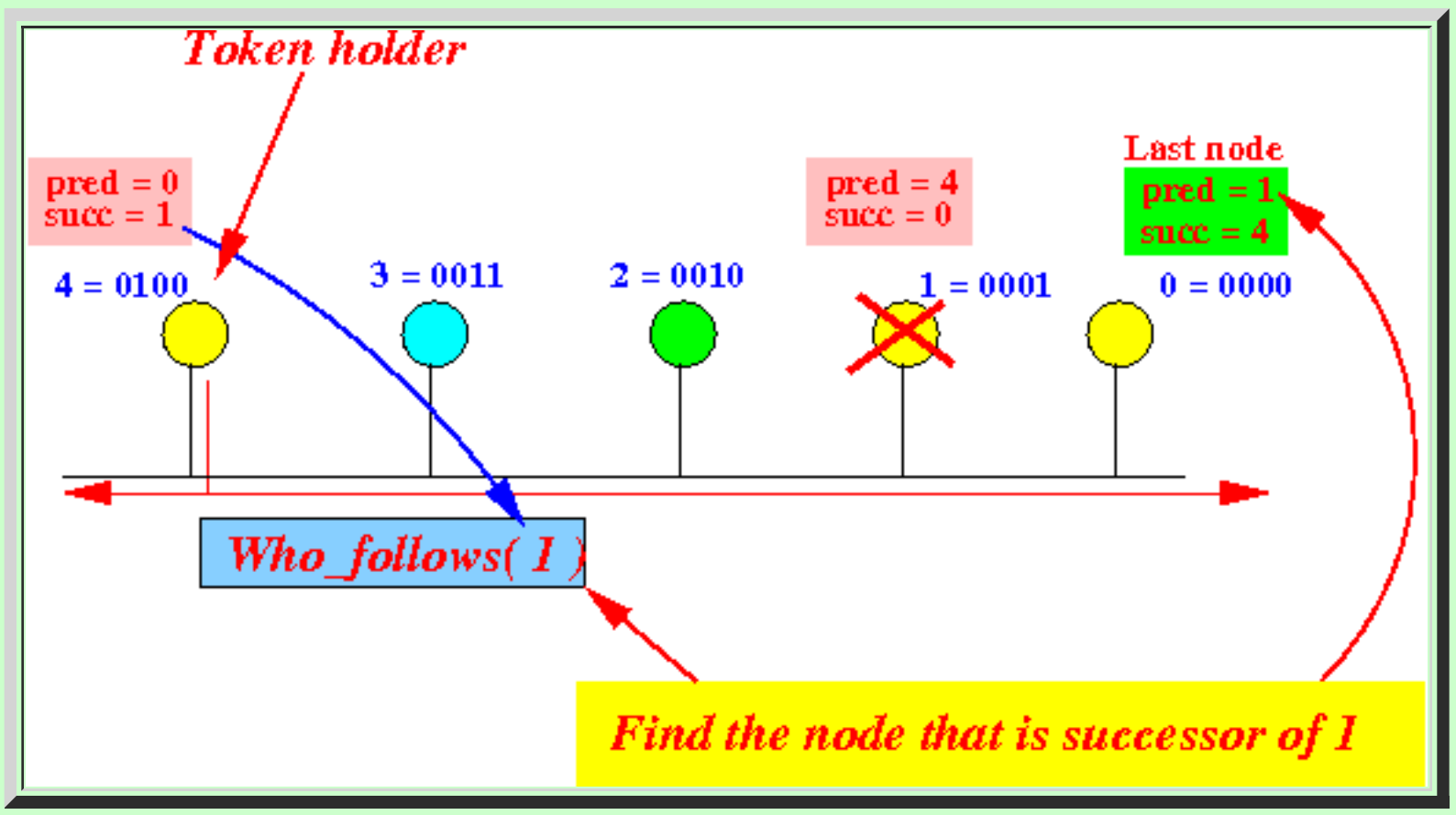
- **The ring recovery protocol of the token bus network**
  - The **ring recovery protocol**:
    - The **current token holder** will **execute** the **ring recover protocol** when:
      - The **token holder cannot pass** the **token** to its **successor** due to **node failure**:



  - The **ring recovery protocol**:
    - The **token holder transmits** a

```
Who_follows( my successor node ID )  frame
```

Example: **node 4** will transmit a `Who_follows(1)` control frame



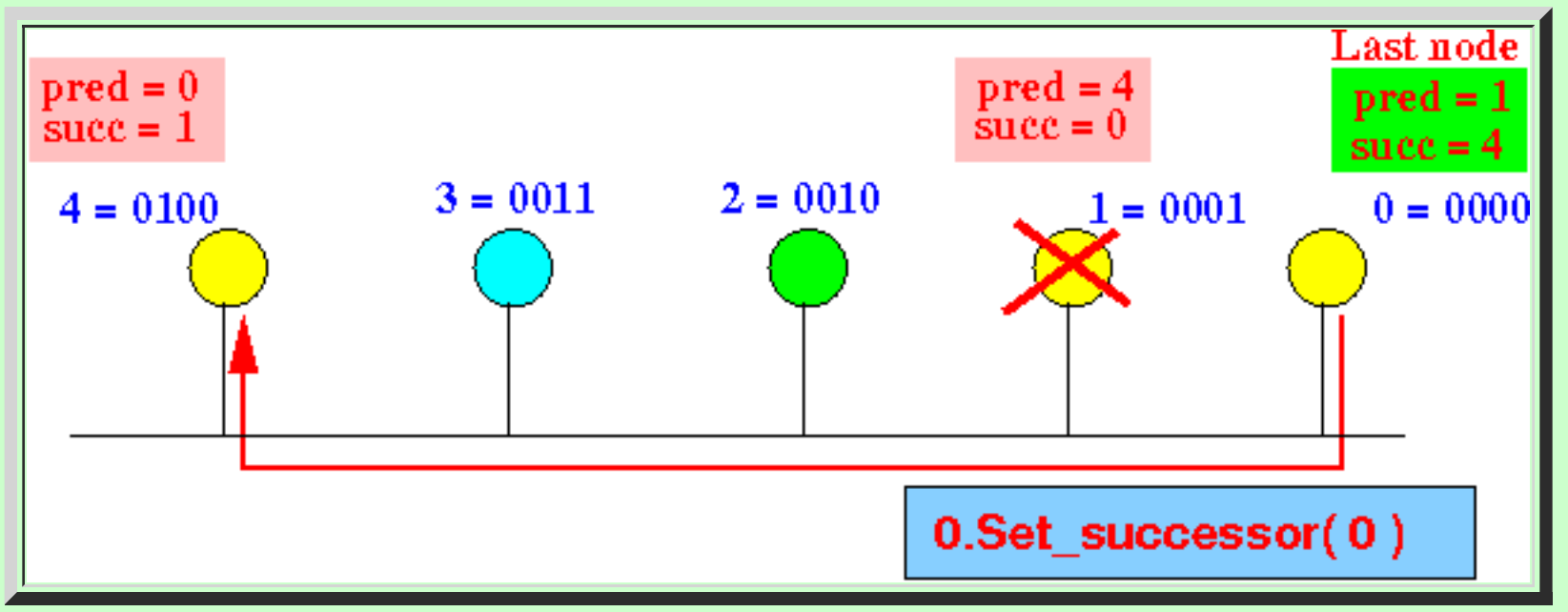
    - The **node** that has:

```
predecessor == nodeID in "Who_follows" frame
```

will **send back**:

```
Set_successor( myID )
```

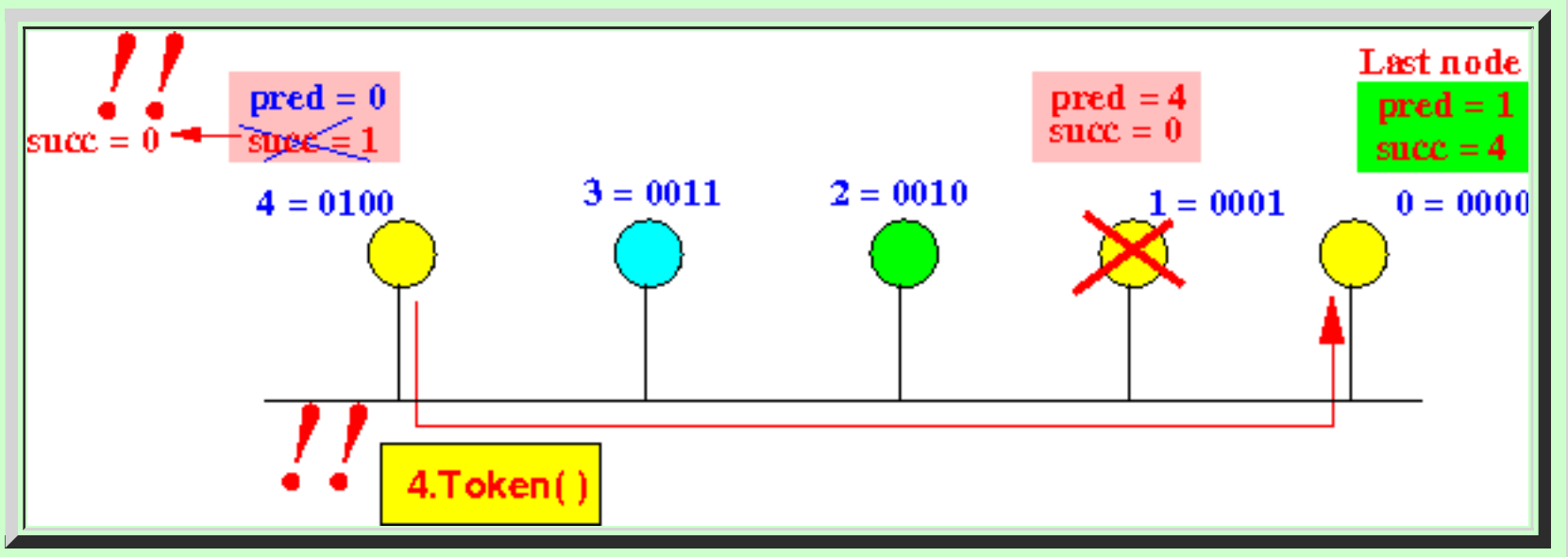
Example: **node 0** sends back `Set_successor(0)` to **node 4**



- The **token holder** will:

1. **Set** its `Successor node` **variable** **and**
2. **Send** the `Token` to its ***new*** **successor**:

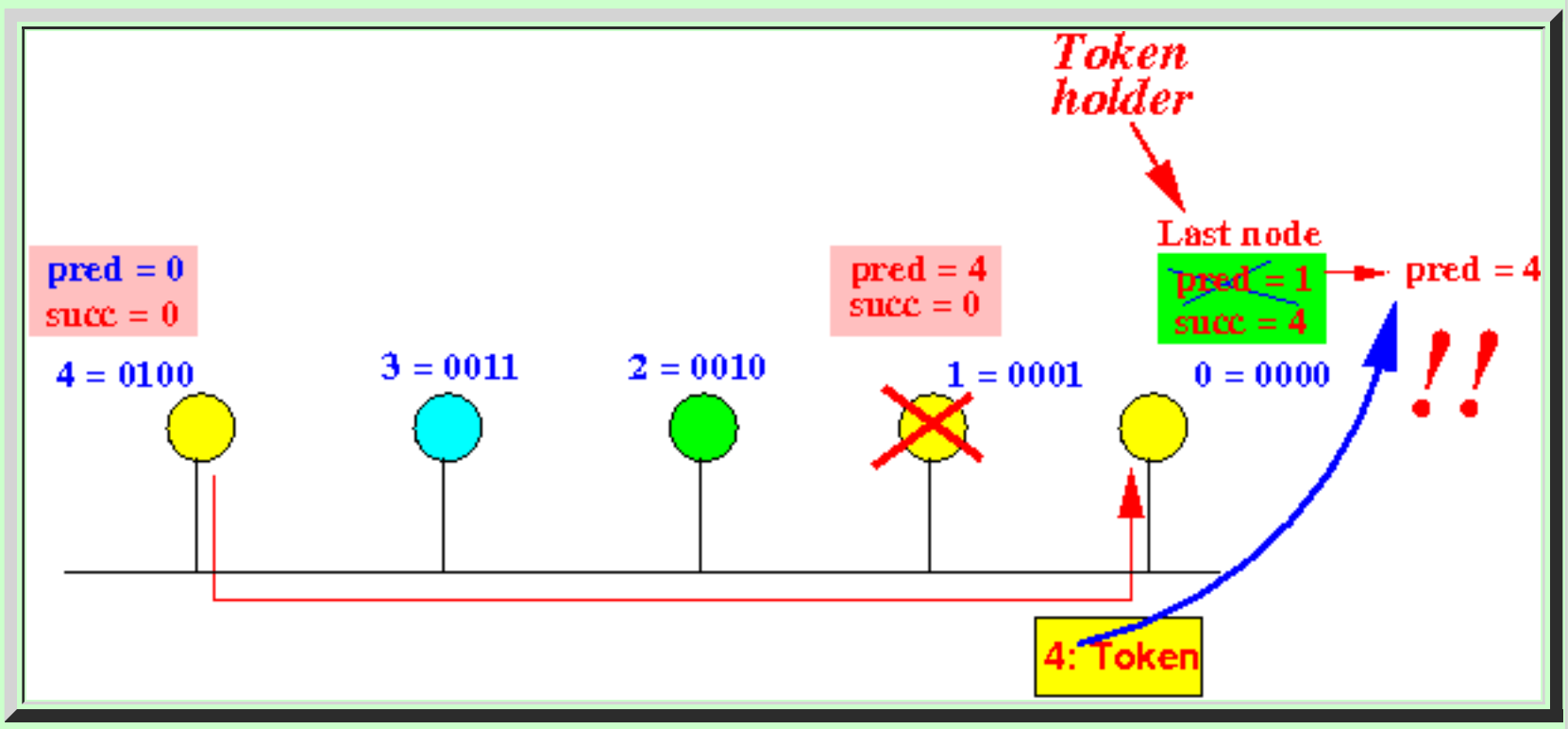
Example: **node 4** updates `succ` and **transmits** `Token` to **node 0**:



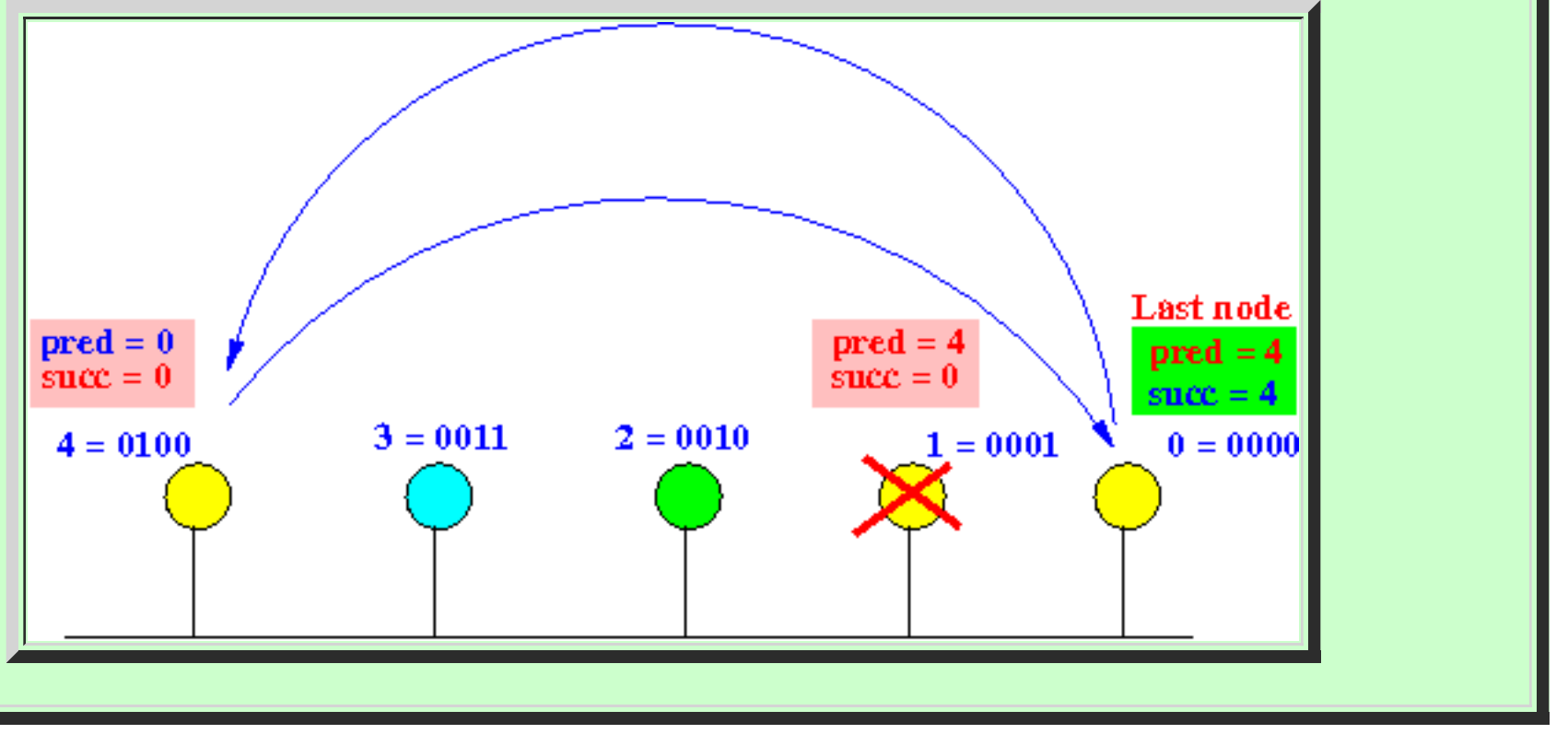
- When the **node** receives the **token (frame)**: frame:

- The **node** will **update** its `predecessor` **variable**
- **Becomes** the **token holder** (and **transmits** a **frame** if it has some)

Example: **node 0** updates its `pred` variable and becomes **token holder**



- And the **ring** is ***fixed***:


pred = 0
succ = 0
pred = 4
succ = 0
Last node
pred = 4
succ = 4
4 = 0100
3 = 0011
2 = 0010
1 = 0001
0 = 0000

Intro to IEEE 802.11 --- Wireless LAN

- **IEEE 802.11 Standard**
  - The **IEEE wireless LAN standard** is:

    - **802.11**

    A.k.a.: ***Wi-fi***

  - **Different *variants*** of the **standards** are **differentiated** with a **letter**

    Example:

    - **802.11a**
    - **802.11b**
    - **802.11g**
    - **802.11n**
    - **802.11ac**

    - And so on... they keep **changing** the **standard** (**higher speed**)....

  - The **802.11 protocols** are the **most commonly used** **home/office wireless** networks today:

- **Brief history....**
  - ***History:***

- **802.11a**: (1999):

  - operates in the **5 GHz band** with a maximum net data rate of **54 Mbit/s**
  - ***Not* popular....** (equipment was **expensive**)
  - Does not **inter-operate** with **802.11b**

- **802.11b**: (1999)

  - operates in the **2.4 GHz band** with a maximum net data rate of **11 Mbit/s**
  - ***Very* popular** (maybe because **equipment** was **cheap** :))

  (BTW, **microwave ovens** operates in the **2,4 GHz** range !!!)

- **802.11g**: (2003)

  - operates in the **2.4 GHz band** with a maximum net data rate of **54 Mbit/s**
  - It is fully **backwards compatible** with **802.11b**.

- **802.11n**: (2009)

  - Not really a new standard....
  - Is an **amendmentment** to the **existing 802.11** standards....
  - maximum data rate of **600 Mbit/s !!!**

  ***Most* important amendment**:

  - **MIMO** (Multiple Input, Multiple Output -- multiple antennas)

- **More wireless standards** are **still being proposed**....

MIMO: multiple input, multiple output

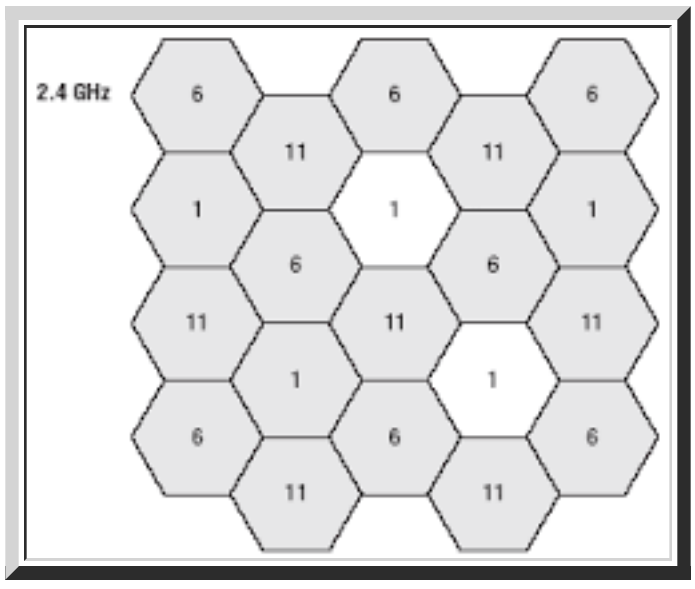
- **MIMO *Signal Processing*** uses **multiple *antennas*** that **send** and **receive data** at the ***same* time** to improve signal coherence.

- This **technique** increase the **transmission rate** significantly.

This is not a **Electrical Engineering class**...

If you want to know **more** about **MIMO**, see: **click here**

- **802.11 network architecture**
  - An **802.11 Lan** is subdivided into **cells**



  - **Each cell** uses a ***different*** **transmission frequency** !!!

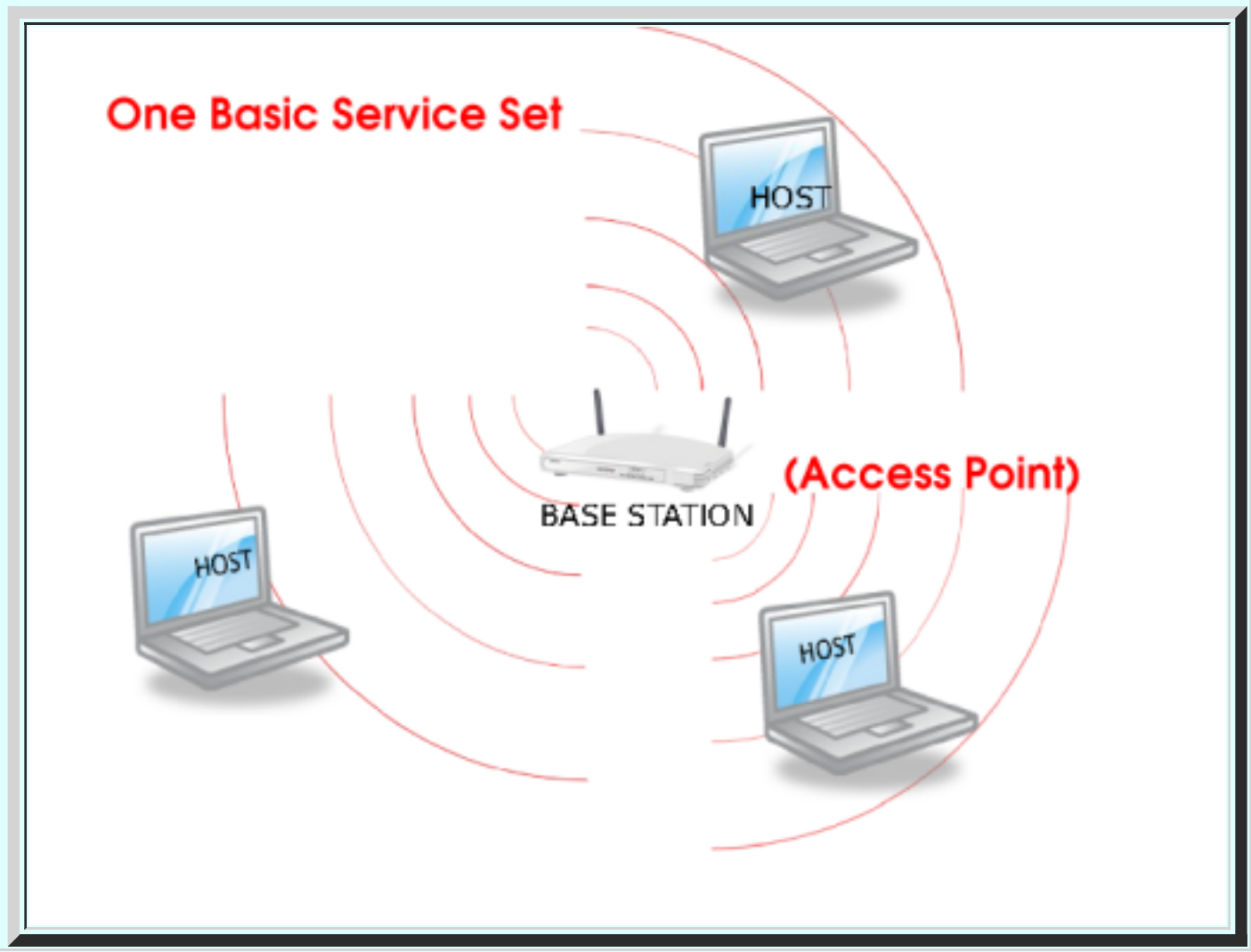
    - **Signals** of ***different*** **frequency** do **not interfere** with **each other**

      (**E.g.: radio stations** on **different frequencies FM 90.1** and **FM 94.9** can **transmit *simultaneously***)

  - Each **cell** is called a **Basic Service Set (BSS)**

    - A **BSS** is **serviced** by an **Access Point (AP)** (**a.k.a.** a **base station**)

      Example:

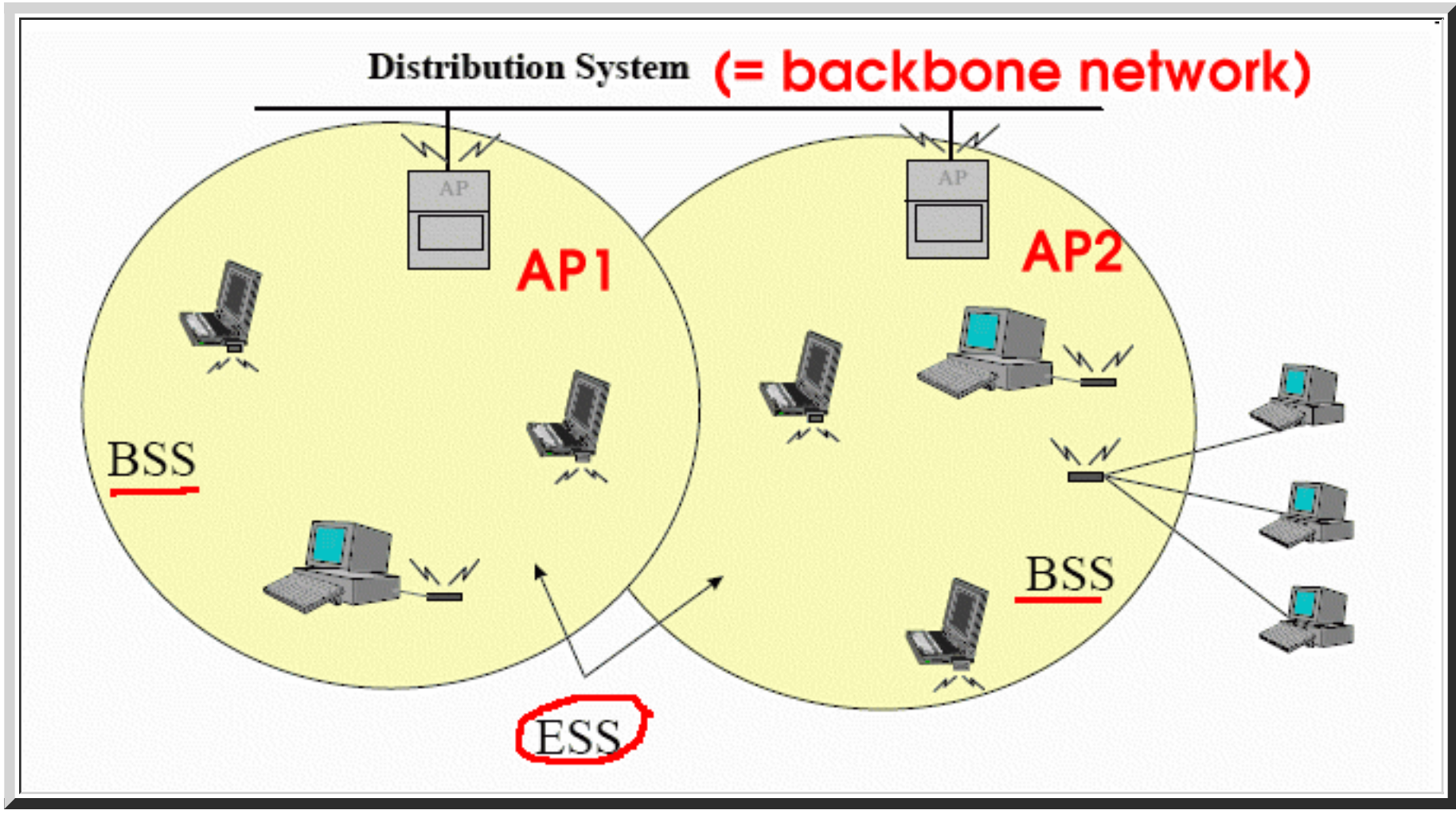


- Most **802.11 networks** consist of ***multiple* APs**

  - The **Access Points** are **interconnected** together using a **"backbone" network**.

    - The **backbone network** is **usually** an ***Ethernet* network**

Example:



The **backbone network** is called:

- The **distribution system** in **802.11 literature**

The ***entire* interconnected wireless LAN** is called:

- a ***Extended* Service Set (ESS)**

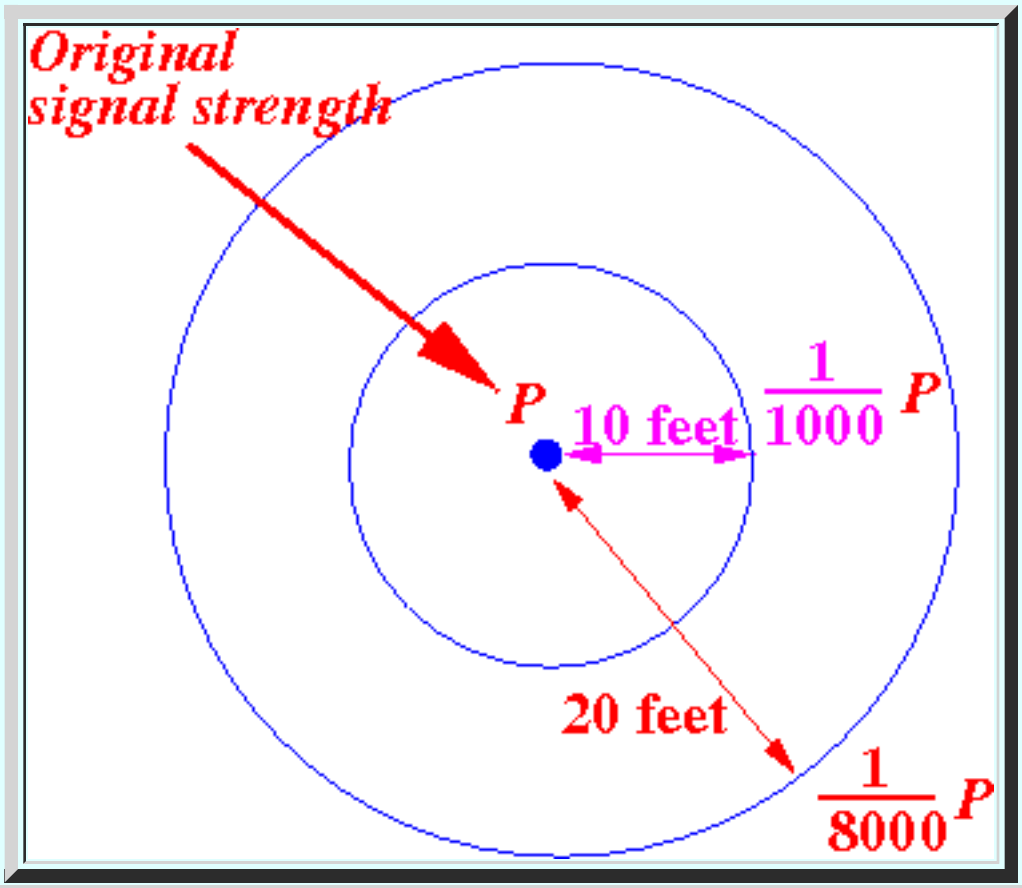
Physics of wireless communication

- **The Wireless Networking environment (Physics...)**
  - **Physics** of **wireless transmissions**:

    - The **signal strength** of wireless transmission **attenuates (= weakens) very rapidly**:

      - **Signal strength *decreases*** at a rate of **$1/r^3$** where **r is the distance** to the **source**

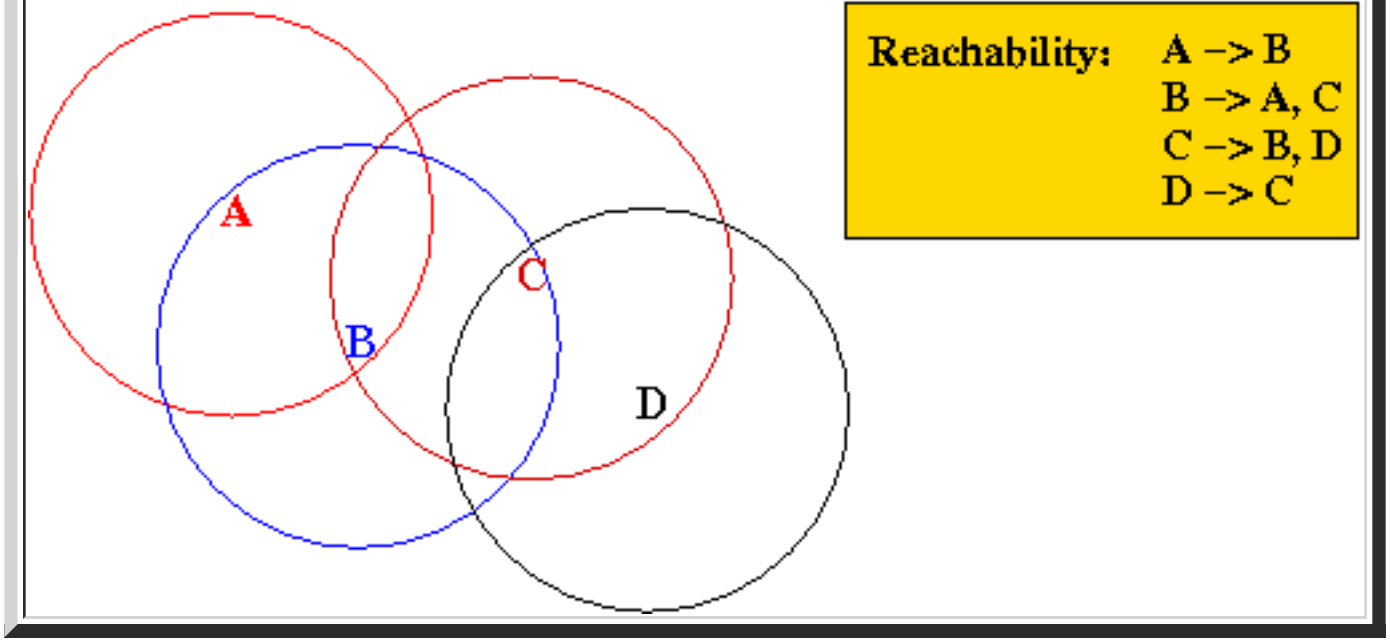
    Example:



  - ***Rapid* signal strength attenuation** results in:

    - a ***limited* range** of **reception**.

    Example:



Notes:

- The letters indicate the physical location of each node
- The **circles** indicate the **transmission range** of each node
  - Node **B** is in **A**'s range.
  - Nodes **A** and **C** are in **B**'s range.
  - Nodes **B** and **D** are in **C**'s range.
  - Node **C** is in **D**'s range.

- **Distance covered by 802.11**
  - **Range** of **802.11 devices:**
    - The **range** is **relatively short**:
      - about a ***hundred*** **of feet**.

  - **Because** of the **short distance**:
    - A **dode's *transmissions*** will **reach other nodes almost *instantaneously***
      - **Nodes** will ***still*** **be transmitting** (a **frame**) when ***other*** **nodes detect** the **transmission**

In other words:

- **Channel sensing can** help you ***avoid*** **collisions** !!!

  (Just like **Ethernet**)

- **Problem with wireless networks**
  - **Fact:**
    - Due to ***rapid*** **signal attenuation** (= **weakening** of the **transmission signal**) in **wireless transmission**:
      - The **effectiveness** of **channel sensing** is **diminished**

  - **Problems** that ***wireless*** **networks** have to **deal with**:
    - The **hidden node problem**:
      - A **node** that is ***out of range*** can **interfere** with **your transmissions**

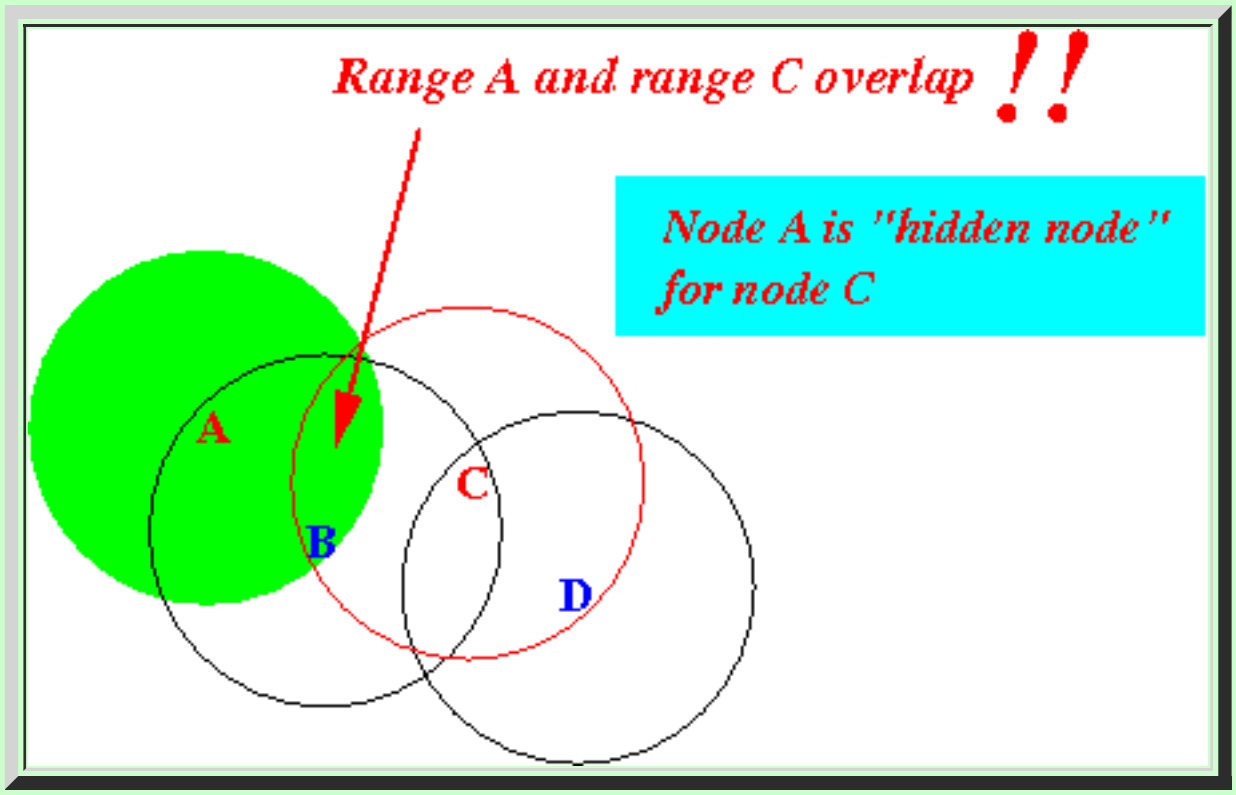
    - The **exposed node problem**:
      - A ***transmitting*** **node** that is ***in range*** but **do *not* interfere** with **your transmissions**

The *hidden* node problem

- **"Hidden" Node**
  - **Hidden node:**
    - A **hidden node** = a **node** that is ***outside*** **your range** and ***interferes*** with **your transmissions**

      Note:
      - **Because** the **node** is ***outside*** **your range**, you **cannot** be **aware** of **the *hidden* node** !!!

  - **Example:** (**hidden node**)
    - **Node A** is ***outside*** **the range** of **node C**



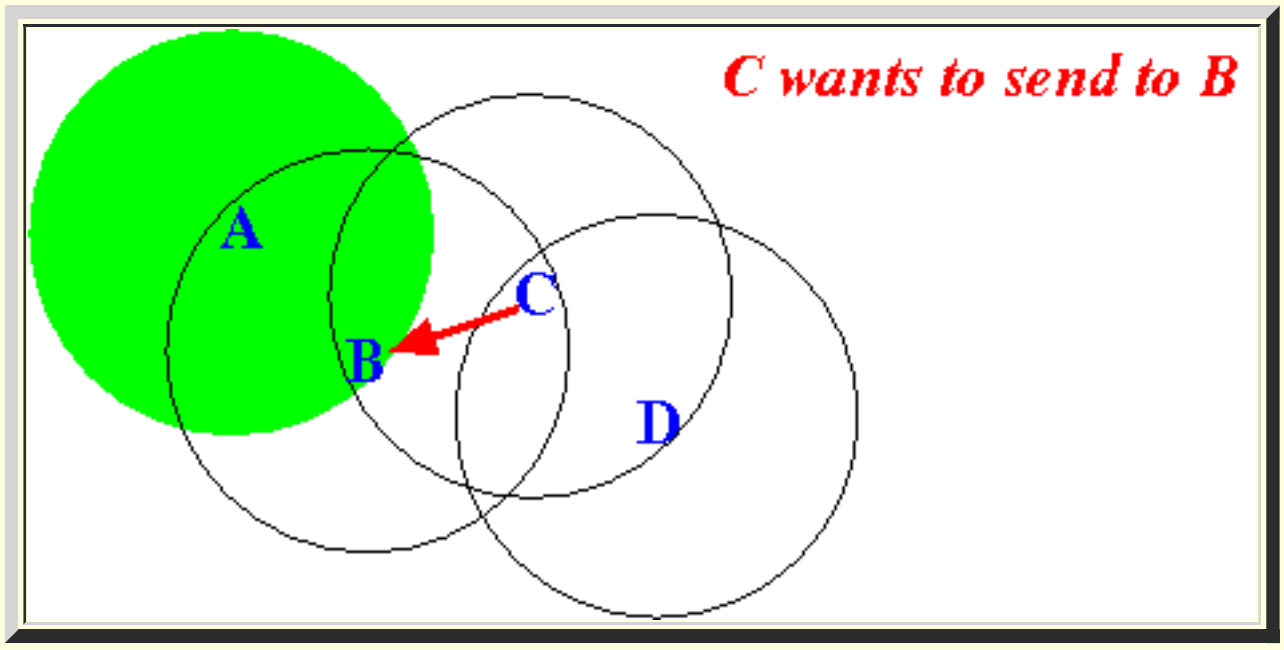
      Node A can ***interfere*** with the **transmissions** from **Node C** (they have ***overlapping* reach area** !!!)

- **The "Hidden" node problem**
  - The ***hidden*** **node problem:**
    - The **transmissions** of a **node** can ***collide*** with **transmissions** from **hidden nodes**
      - These **collisions** are ***undetectable*** because a **node** can **not** hear the

transmissions from its ***hidden*** **nodes** !!

Example: (hidden node problem)

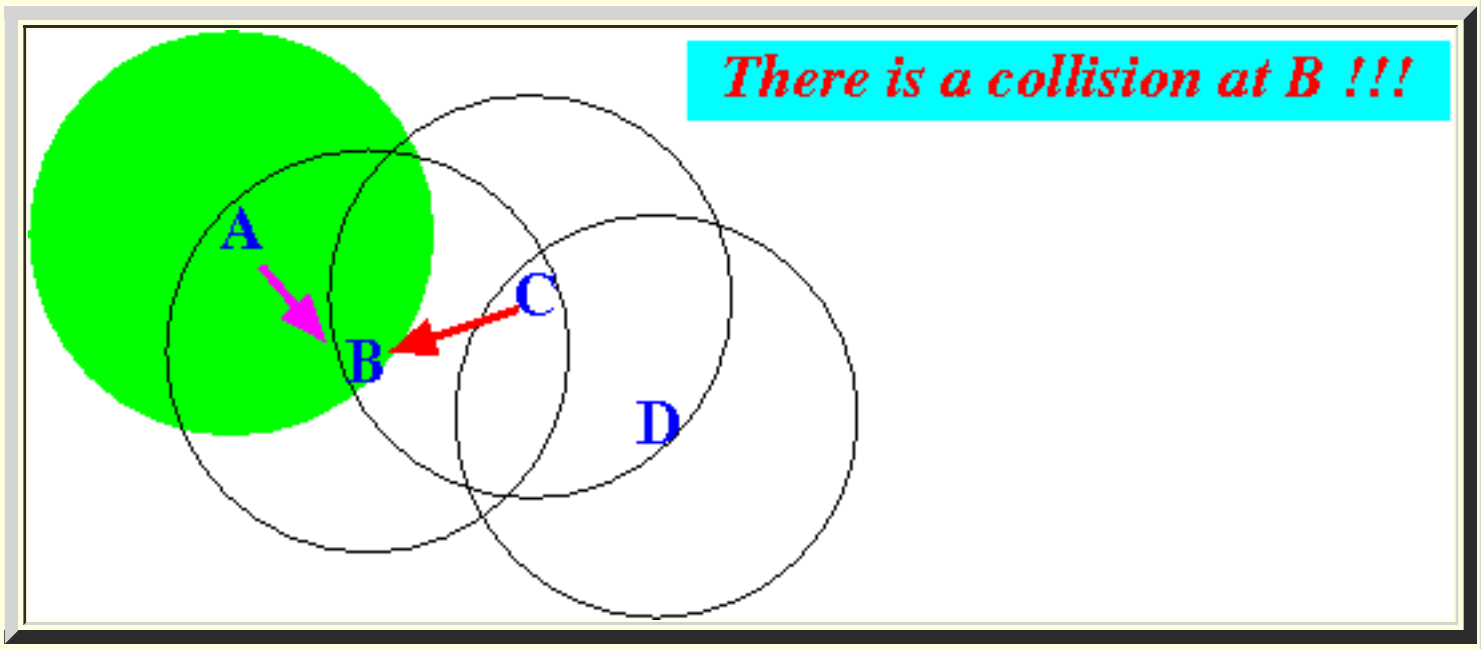
- **Suppose: Node C** wants to send a message to **B**:



Use **channel sensing**:

- **Node C** senses that the **channel** and find that it is **clear (no transmission)** ....

- When **node C** transmits to **node B**, it **can** result in a ***collision***:



Observe that:

- **C cannot hear A**'s transission
- **But**, when **node C does transmit**, it will cause an ***collision*** at node **B** !!!

- **Conclusion:**

  - ***If*** a **node** senses that the **channel** is **clear**, this **fact** does ***not*** guarantee that there will **not** be a **collision**

    - due to the ***hidden*** **node** problem

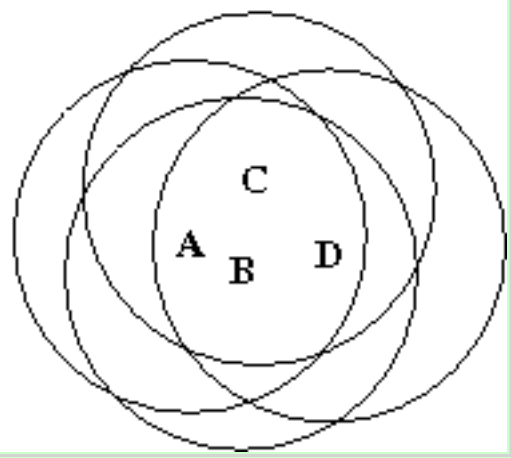
- **Comment**

  - **Fact:**

    - The ***hidden*** **node** problem is ***not*** **always present** in a **wireless network** !!!

      Example:

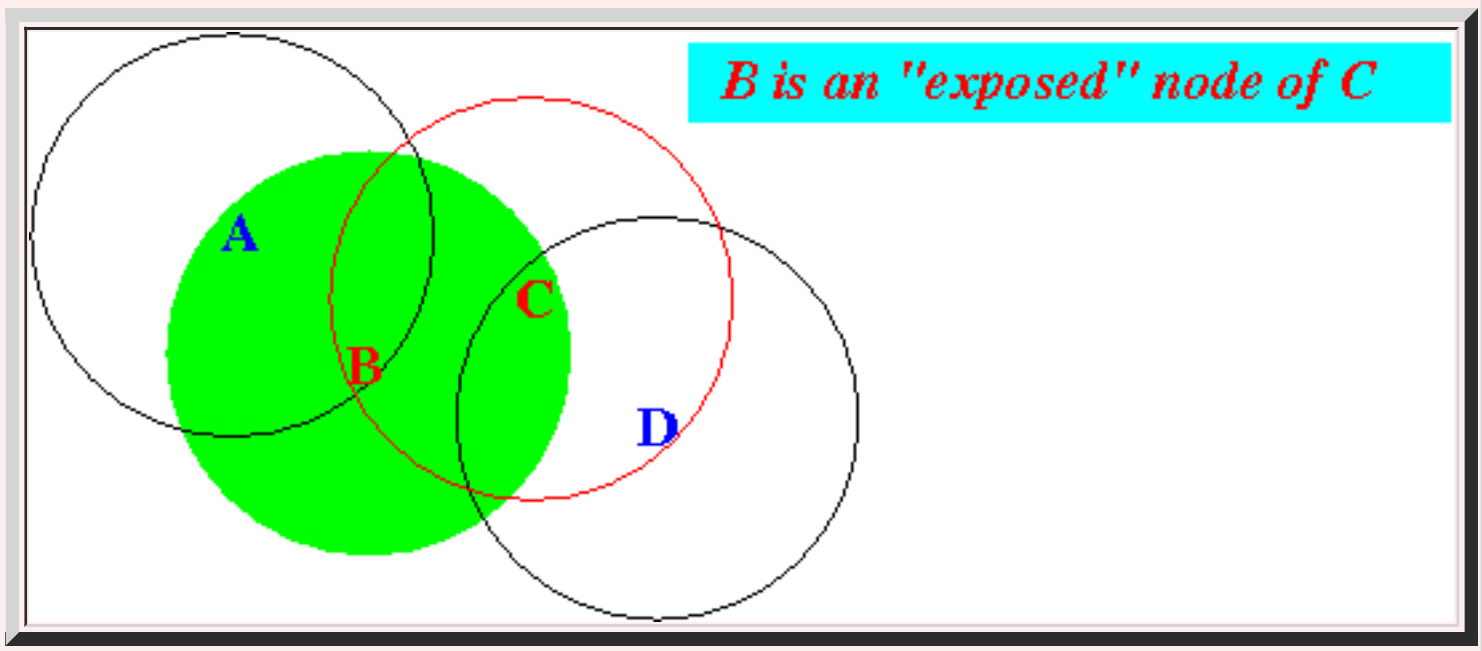
      NO hidden node problem:

    - **Carrier sensing** will ***avoid*** **collision** if there are ***no*** **hidden nodes** !!!

The *exposed* node *annoyance* (not so much a "problem")

- **Exposed Node**
  - **Exposed node:**
    - **Exposed node** = a **node** that is ***within*** **your range** **but** does ***not*** cause a **collision**
  - **Example:** (**exposed node**)
    - **Node C** is **inside the range** of **node B**



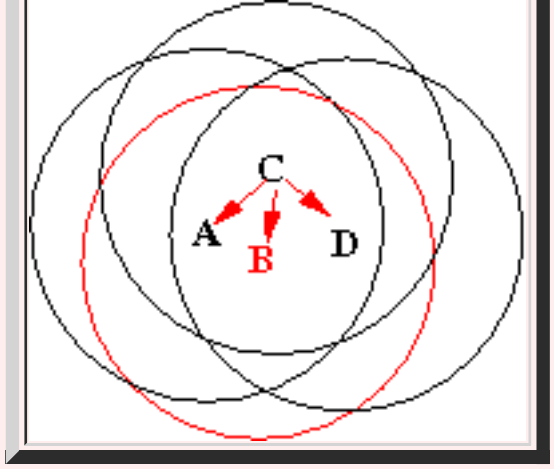
But:

- **Node C** does ***not*** **interfere** with the **transmissions** from **Node B**

- **Review: sense to avoid collision**
  - **Recall**: **collision avoidance**
    - ***If*** a **node** hears an **on-going transmission** (from another node):
      - the **node** ***must delay*** from **transmitting**

      because:
      - The **transmission** will **collide** in the ***on-going*** **transmission**

    Example:
    - **Node C** is **currently** **transmitting** to **node A**:



- **Node B** can **hear** the **transmission**

  - ***If* node B** starts **transmitting**, the **transmission** will ***collide*** with **node C**'s **transmission** !!!

- **The *Exposed* node "annoyance"**

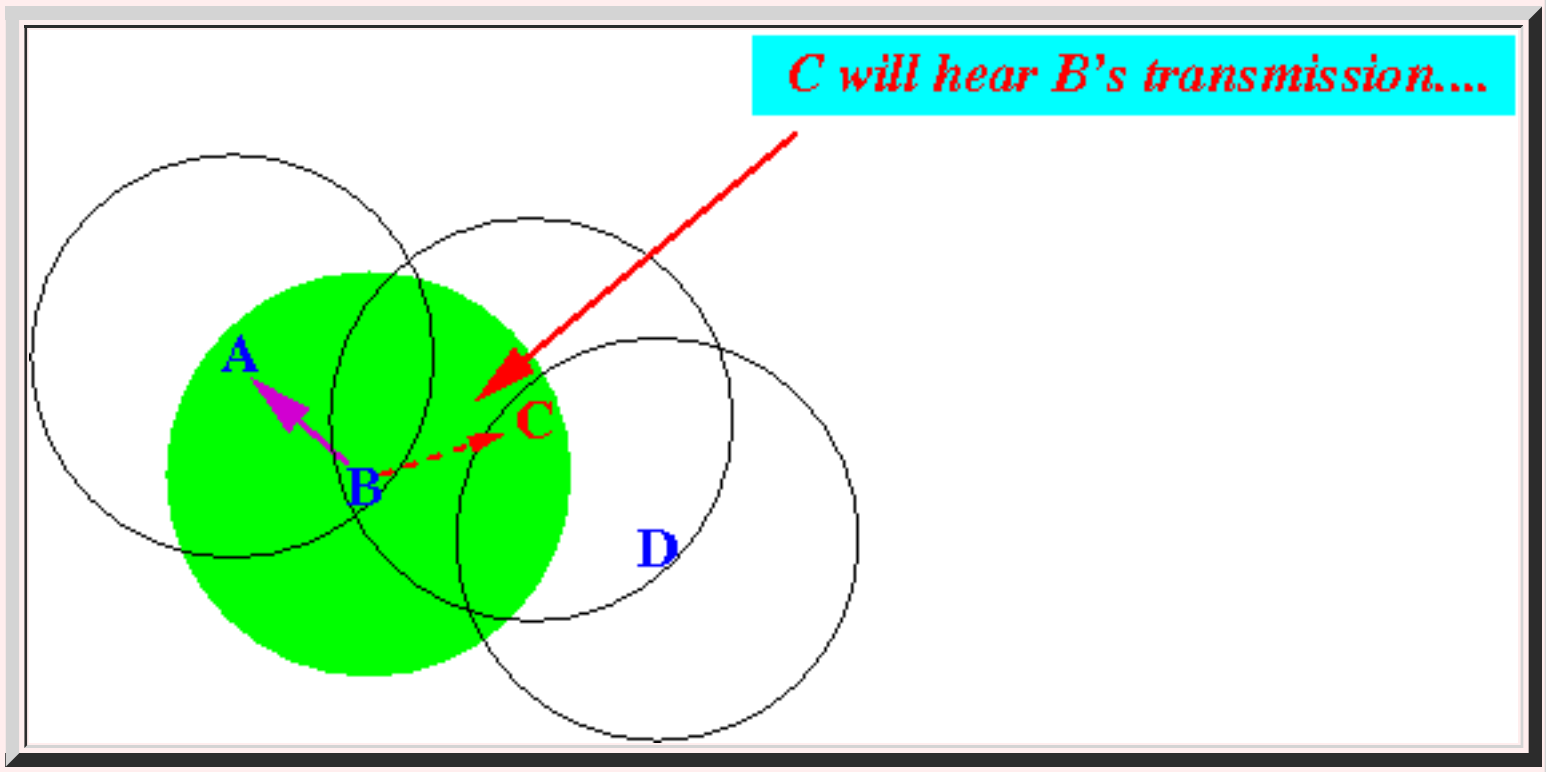
  - **Exposed node "problem" (*annoyance*):**

    - A **node** can **hears** an ***on-going* transmission** from its **exposed node**, and:

      - This ***on-going* transmission** (from the **exposed node**) will ***not* collide** with ***its own* transmission**

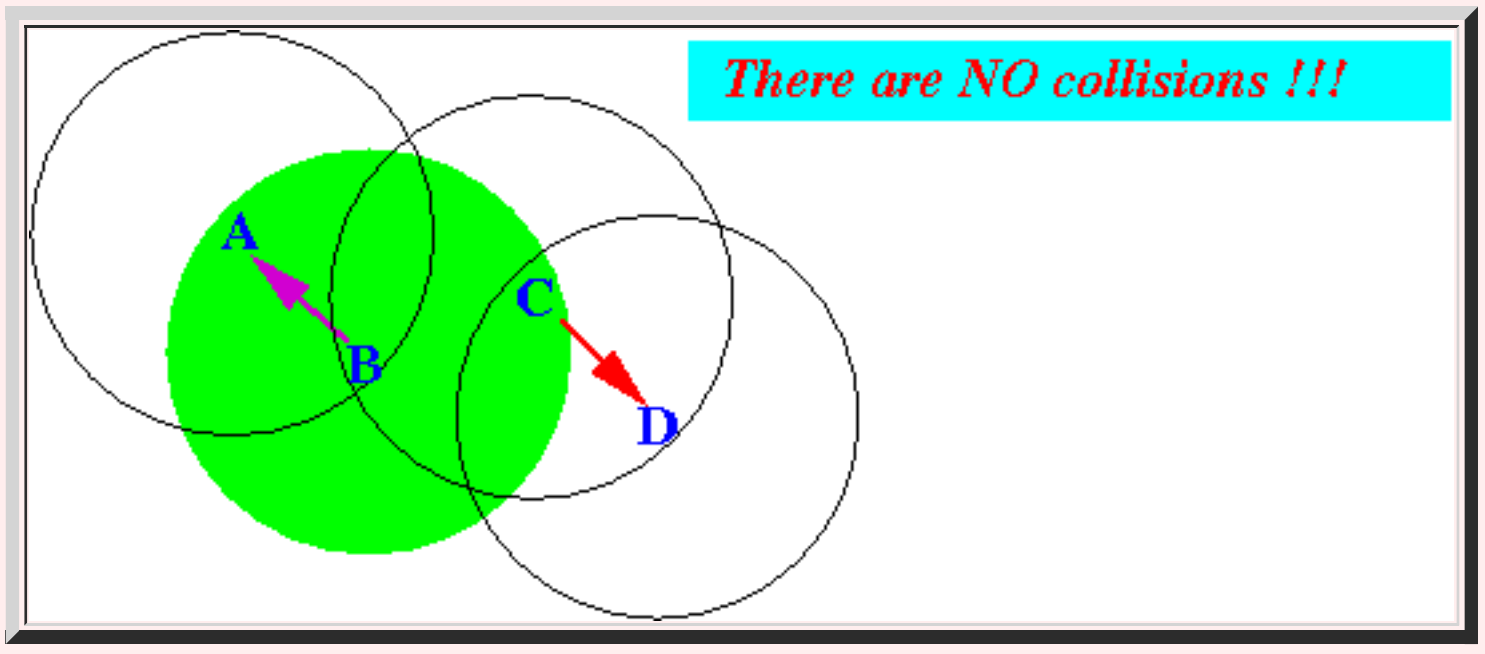
Example: (**exposed node *annoyance***)

- Suppose **Node B** is currently **transmitting** to **Node A**:



Node C will **hear *B*'s transmission**

- ***If* node C would transmit** to **Node D**:



There will be ***no* collisions**:

- **B's signal** can ***not*** reach **node D** to cause a **collision** !!!

Remember:

- A **collision** is **caused** by ***multiple* signal** reaching the ***same* destination**

  - **Conclusion:**

- ***If*** a **node** senses that the **channel** is ***busy***, this **fact** does ***not*** guarantee that there ***will*** be **collision**

- **Post script**
  - The **exposed node** is ***not* a problem**:

- The ***exposed* node** is **not** a ***problem*** because:
  - **Hearing** an **on-going transmission** will **prevent** a **node** from ***transmitting***
    - It is ***losing*** some **opportunity** to **transmit**

That's **why** I call it an ***annoyance***...

  - The ***hidden* node** is a ***real* problem**:

- A **node** is **not aware** of its ***hidden*** **nodes**

- The **node's transmissions** can be ***lost*** due to **collisions** with those from its ***hidden*** **nodes** !!!

Summary: channel sensing in wireless commnucation

- **Effectiveness of channel sensing in Wireless network**
  - **Summary of the above results:**

    - If a **wireless node** finds the channel **idle**, there is the **possibility** than its transmission will **collide** with a current transmission from a **hidden node**...

    - If a **wireless node** finds the channel **busy**, there is the **possibility** than the **current transmission** will ***not*** **collide** with its own transmission...

    Conclusion:

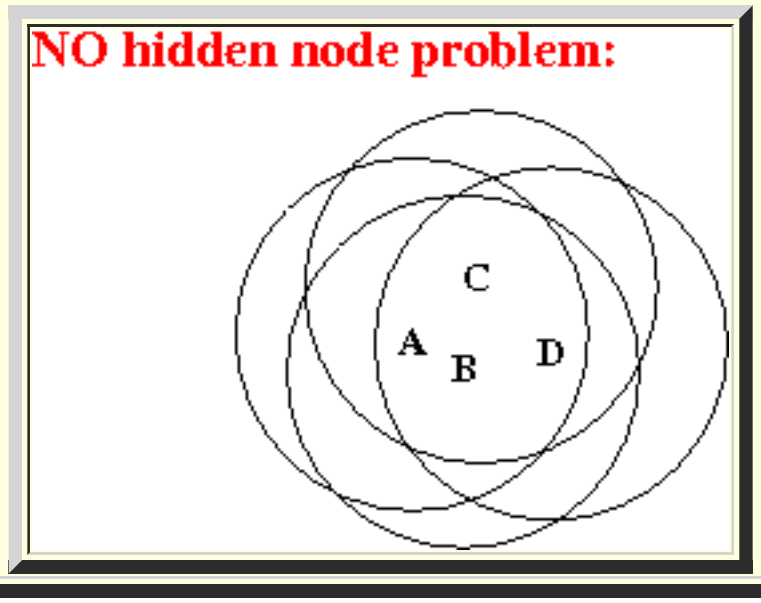
    - **Sensing the carrier** is ***not*** **100% effective** in **wireless networks**...

      (because the information that a node get from sensing the carrier does not tell him whether his transmission will or will not interfere with another transmission)

    - **But...**, **sensing** is ***not*** **completely useless** !!!!

      - **especially** if there are ***no*** **hidden nodes** !!!

    Example:

- **The 802.11 MAC protocol**

- **802.11 Medium Access Control (MAC) protocol:**

  - **802.11** will **use carrier sensing** to **reduce collisions**

  - **In addition**, the **802.11 protocol** will **also** use:

    - ***Frame* acknowledgements** to **detect** frame **losses**

      - **I.e.:** Just like **Aloha** !!!

    - A ***military-grade* communication technique** to ***overcome* collisions** !!!

      - **802.11** uses (battle-field-like) ***anti-jamming*** communication techniques !!!

    - **Optionally** use a **reservation-base protocol** to solve the ***hidden* nodes** problem

Intro to Spread Spectrum Communication

- **"Normal" radio communication**
  - **Normal** way to use **radio waves (wireless)** to **communicate**:
    - The **sender** transmits **messages** on
      - ***One* frequency**
    - The **receiver** **listens** on ***one* (same) frequency**
  - **Consequence:**
    - When ***another* node transmits** the ***same* frequency**:
      - **Transmissions** on the ***shared* frequence** will be **lost**

- **Military grade communication**
  - **Battlefield scenario:** (***jamming***)
    - **Soldiers** at the **front line** use **radio communication** to report the ***current* situation** back to the **head quarters** (tug away safely behind many line of defenses)
    - The **enemy** will ***always*** try **prevent** the **soldiers** from **reporting**:
      - The **enemy** will **jam** the **transmission** (= **transmit** garbage on the ***same* frequency**)
  - ***Anti*-Jamming** communication:
    - **Anti-jamming** technique:

- **Transmit** the **signal** using ***multiple*** **frequencies**

- This **technique** is called:

  - ***spread-spectrum*** **communication**

  (Spread the **signal** over a ***spectrum*** **of frequencies**)

- **Wikipedia page: click here**

- **Spread spectrum communication**

  - **Spread spectrum transmission:**

    - **Spread-spectrum transmission**: transmits a signal using ***multiple*** **frequencies**

      I.e.: the **data** is ***spread*** across ***different*** **frequencies**

  - **Techniques** used to achieve **"spread spectrum" transmission:**

    - **Frequency hopping**
    - **Direct sequence**

Frequency Hopping Spread Spectrum (FHSS) Communication

- **Frequency Hopping**
  - The **frequency hopping** spread spectrum **(FHSS)** **transmission technique**:

    - The **sender** and **receiver** ***first*** agree on a **list of frequences**:

      $f_1 \quad f_2 \quad f_3 \quad f_4 \quad ... \quad f_N$

    - The **sender** and **receiver** then **pick (agree)** on a **random sequence**:

      $r_1 \quad r_2 \quad r_3 \quad r_4 \quad .... \quad$ (indefinitely long)

      This can be **achieve** by using the ***same*** **seed** in a common **random number generator**.

    - **Sender:**

      - Send the **first bit** of the message on **frequence** f_{r_1}
      - Send the **bit 2** of the message on **frequence** f_{r_2}

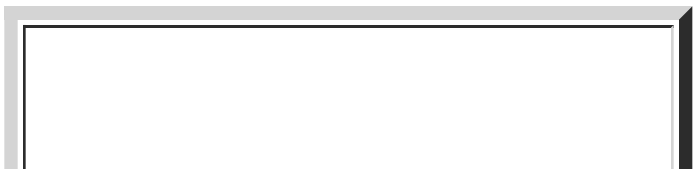
        ...

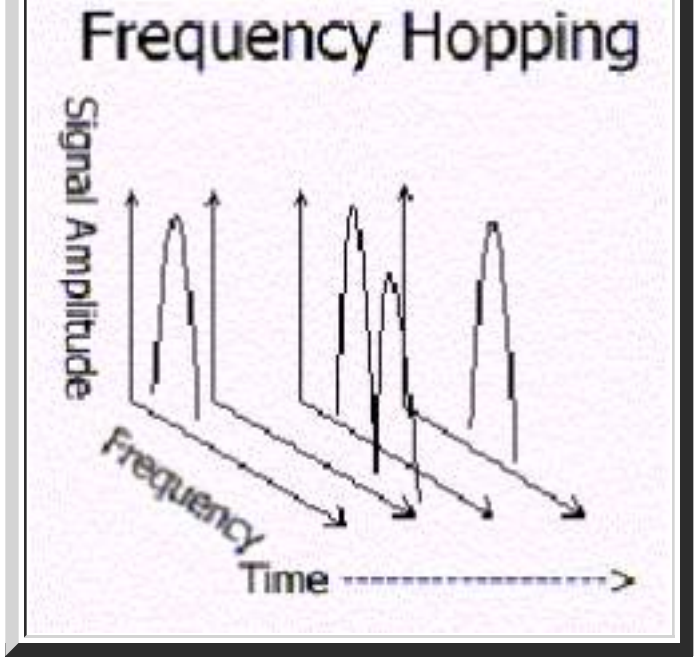
    - **Receiver:**

      - Receive the **first bit** of the message on **frequence** f_{r_1}
      - Receive the **bit 2** of the message on **frequence** f_{r_2}

        ...

  - **Graphically:**





Explanation:

- The picture above shows **4 different times** that **FHSS** transmits

- **Each transmission** uses a ***different*** **frequency**

NOTE:

- **Some *data* can still be lost** due to **jamming**

  - **Data** transmitted on **frequencies** that is being **jammed** will be **lost**

  But:

  - The **data** transmitted on the ***unjammed*** **frequencies** will be **received** !!!

- **Voice communication is extremely redundant**

  - **Fact:**

    - ***voice*** **communication** is **highly "redundant"**

      I.e.:

      - It is **possible** to **lose** part of the **conversation** and the **rest** is ***still*** **comprehensible** !!!

- **Consequently:**

  - **Frequency hopping Spread spectrum** can ***effectively*** **protect audio messages** from being **jammed**

- **Applications of Frequency Hopping Spread Spectrum**

  - **Applications:**

    - **Anti-jamming**:

      - **Most** of the **audio conversation** will be ***unjammed***
      - **Audio** with **some *clipping*** is **highly understandable**

    - **Anti-eavesdropping**:

      - An **eavesdropper tunes (= listens)** in to the transmission in a ***certain*** **frequency**
      - By **"spreading"** the transmission over **multiple *randomized* frequencies**, **most** of the transmitted **audio signal** will ***not*** **be eavesdropped**

  - **Note:**

    - This technique is **high effective** in protecting **audio transmission**

    - **Fact:**

      - **Audio** is **highly redundant**

      (you don't need to hear **everything** to understand what is communicated)

- So **losing a few bits** in the **audio stream** due to **jamming** will not **prevent the communication**
- Having a **few bits** **"stolen"** will not allowed the thief to **understand** the **entire conversation**

- **Frequency-hopping spread spectrum - Wikipedia:** click here

Direct Sequence Spread Spectrum (DSSS) Communication

- **Direct Sequence Spread Spectrum (used in 802.11)**
  - **Definitions:**

    - **Pseudo Noise (PN) code** = a ***random*** **sequence** of ***N*** **bits**

      Example:

      ```
      01001011
      ```

      (The **number** of **0's** and **1's** in the **PN code** should be **approximately equal**)

    - **Chip** = a **popular name** for the **PN code**

    - **Chip length** = the **number of bits** in the **chip (PN code)**

  - The **Direct Sequence Spread Spectrum** (**DSSS**) **transmission technique**:

    - The **sender** and the **receiver** select a ***common Pseudo Noise*** (**PN**) **code**:

    - **Sender:**

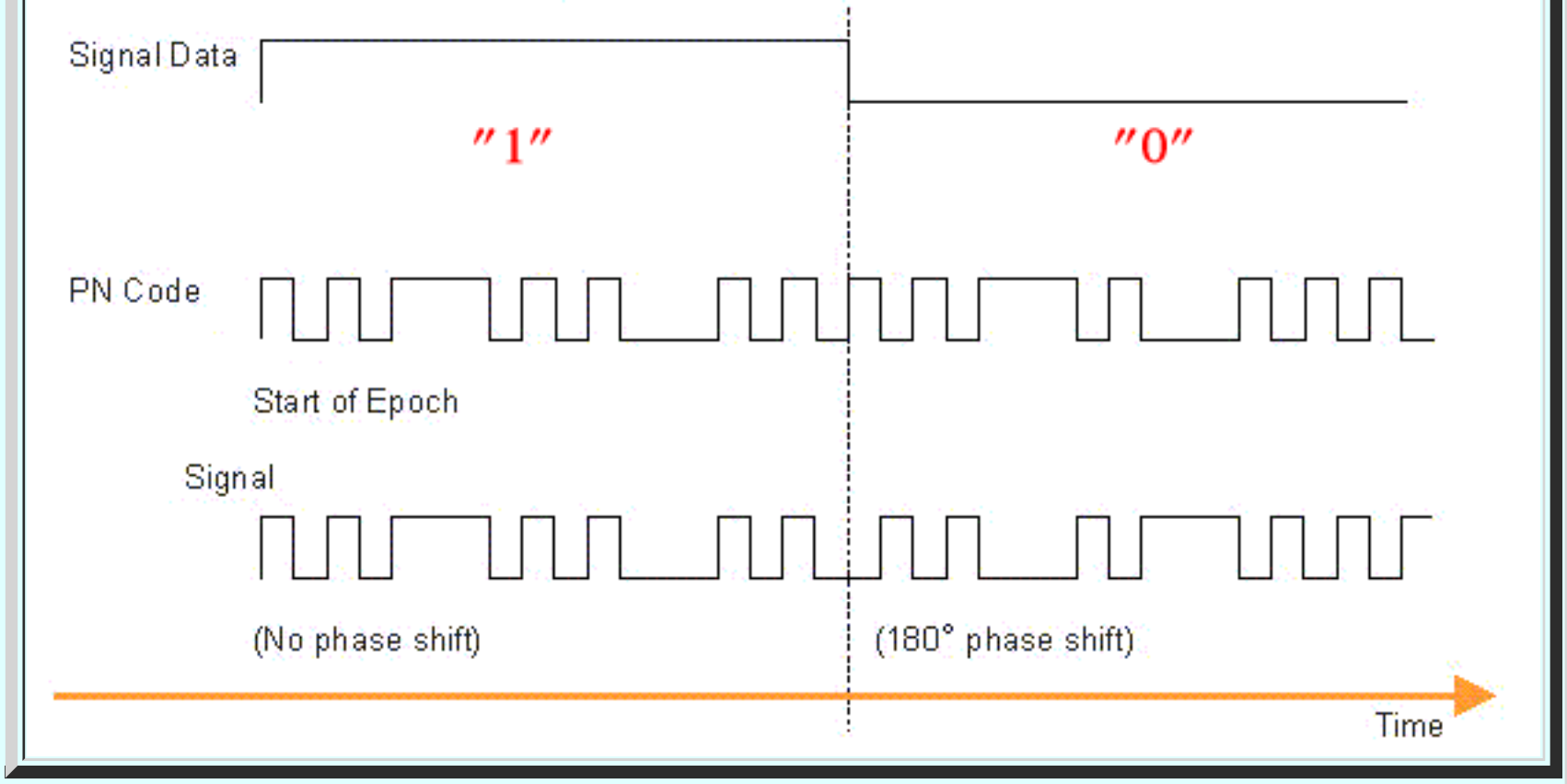
      - **Code** for transmitting **1**:

        - the **chip (PN code)**

      - **Code** for transmitting **0**:

        - the ***inverse*** of the **chip (PN code)**

    Example:



- **Receiver:** (***decoding* algorithm**):

  - Receive ***N* bits** (***N*** = the **chip length**)

  - If a ***majority*** of the received ***N* bits *matches*** the **PN code**:

    - The **receiver decodes** to a **"1" bit**

  - If a ***majority*** of the received ***N* bits *fails* to match** the **PN code**:

    - The **receiver decodes** to a **"0" bit**

- **Example decode process**

  - **Suppose** the **nodes A and B** uses the following **PN code**:

    ```
    PN code = 0011011
    ```

  - ***If*** the **node B** receives:

    ```
    B:        1011011
    ```

    then **node B** will **decode** as **follows**:

    ```
    received:   1011011
    PN code:    0011011   XOR
             ------------------
                1000000   NOT
             ------------------
                0111111
    ```

```
Majority = 1   ===>  1
```

(You can **compare** the **received data** with the **PN code** using an ***Exclusive OR*** operation)

- **A *simplified* explanation of collision**
  - The **interference** of **radio transmission** is **very complex**

    - I will present a ***simplified*** **explanation** on ***how*** the **PN code** can overcome **collisions** (= ***multiple simultaneous transmissions***)

  - **Simplification:**

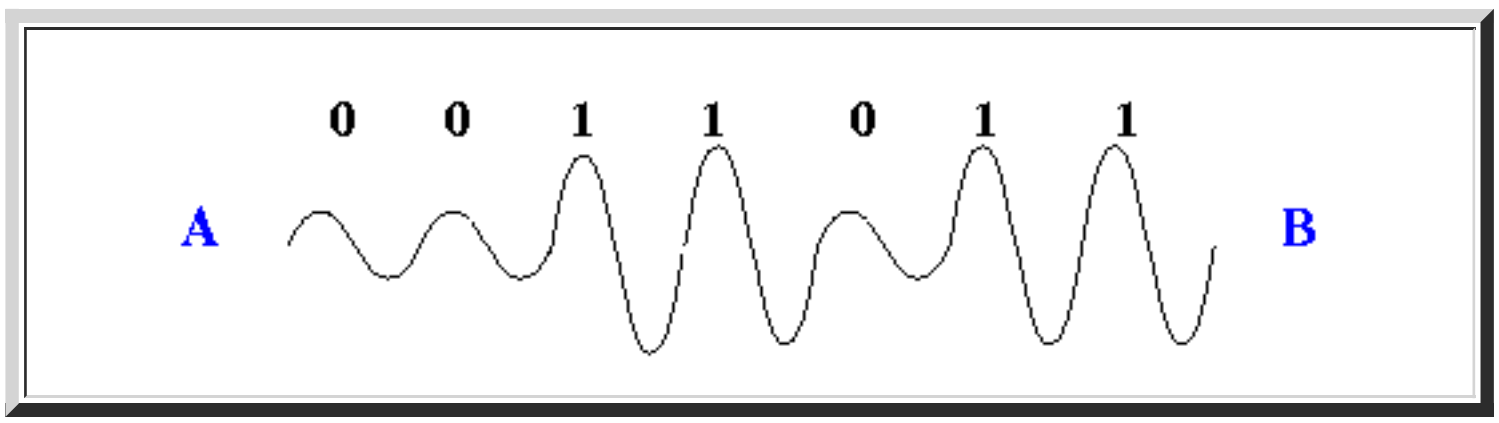
    - I will use **Amplitude Modulation** to show you the ***effect*** of a **collision**

      - In a **later section**, I will **sketch** what is really going on; but not very in depth --- this is a **Electrical Engineering** subject...

  - **Suppose** the **PN code** used is:

    ```
    PN code = 0011011
    ```

  - The **PN code** **transmitted** using **Amplitude Modulation** will look like this:

    (***Low*** **amplitude** = **0**, and ***high*** **amplitude** = **1**)

  - **Effect** of **collision (jamming)**:

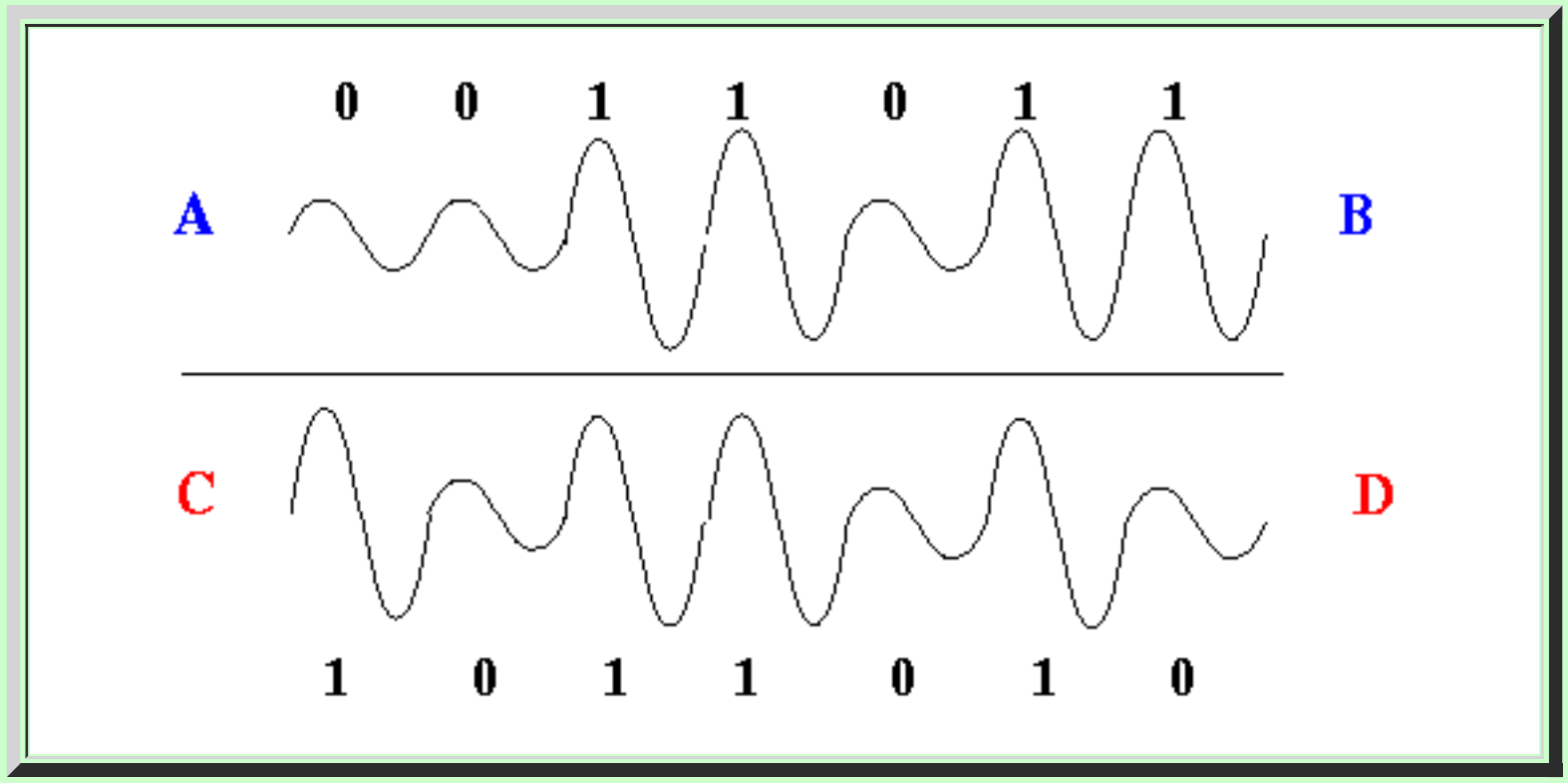
    - **Suppose** there are **2 pairs** of **senders/receivers**:

      - **A/B** uses **PN code = 0011011**
      - **C/D** uses **PN code = 1011010**

    - **Suppose** the following **transmissions** happen ***simulataneously***:

      - **A** sends **"1"** to **B** (i.e.: **A** will send `0011011` --- because its **PN code "0011011"**)
      - **C** sends **"1"** to **D** (i.e.: **C** will send `1011010` --- because its **PN code "1011010"**)

Graphically: (using ***Amplitude*** **Modulation**)

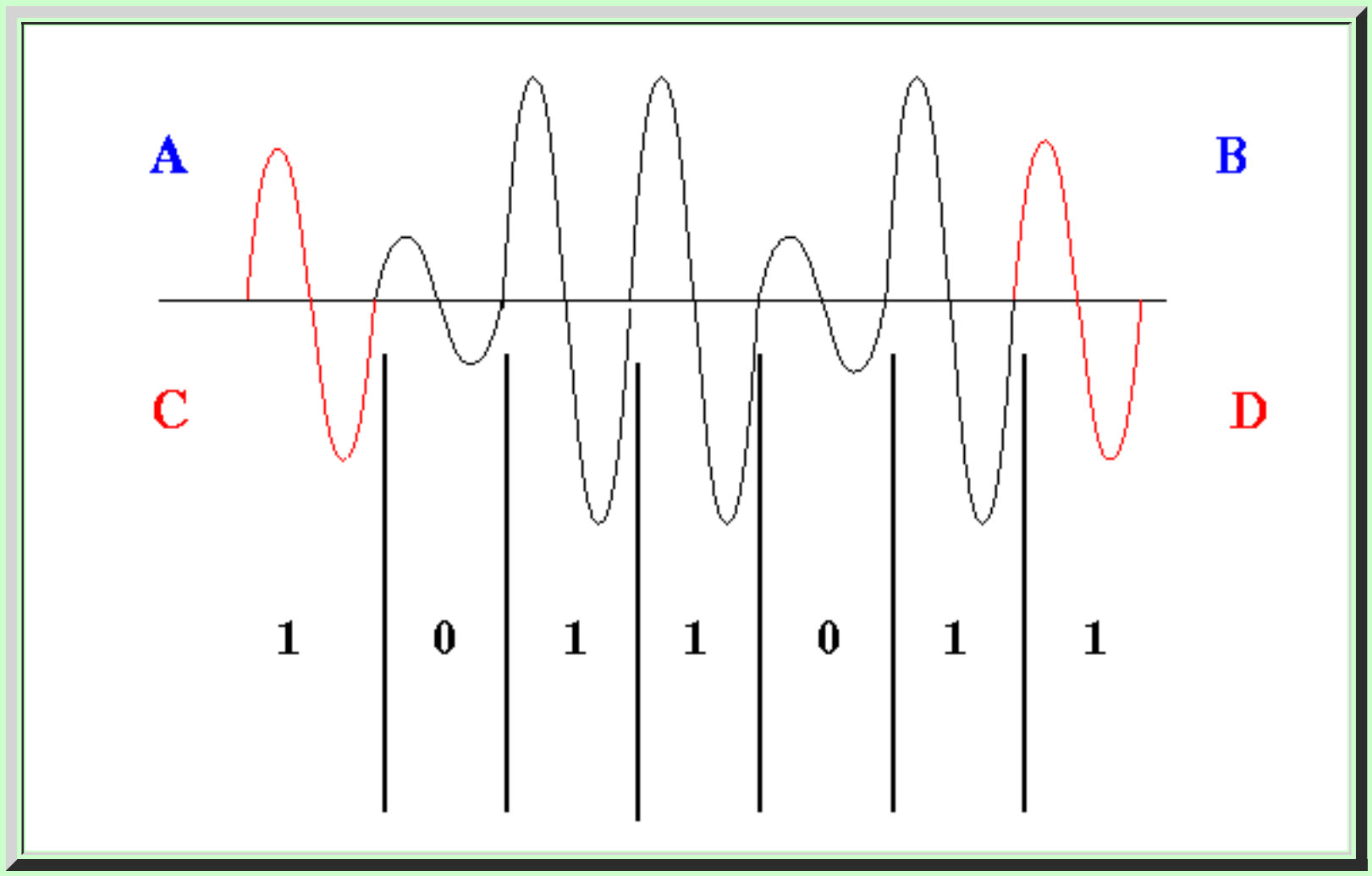


- **Physics:**

  - **Simultaneous waves** result in ***super-imposition***

    (I.e.: the **amplitudes** are **added** together)

The ***super-imposed*** **wave** is as **follows**:



The **receivers** will ***both*** **"hear" (receive)** the **above pattern** !!!!

When **both receiver** try to ***decode*** the reception (using ***Amplitude*** **Modulation**), they will **obtain** this **(*corrupted*) pattern**:

- **1011011**

Because:

- A ***large*** **amplitude signal** will be **decoded** as **1**

- A ***low*** **amplitude signal** will be **decoded** as **0**

- **Observation**:

  - Using ***amplitude*** **modulation** (= my ***simplified*** **explanation**), a **collision** will result in:

    - ***or*****-ing** the **transmitted** **data bits**

    In **reality**, the **receivers** will use **frequency** ***filters*** to perform the **decoding**....

  - The ***simplified*** **explanation** will **still** be **helpful** in **understanding** ***how*** the ***protection*** is provided:

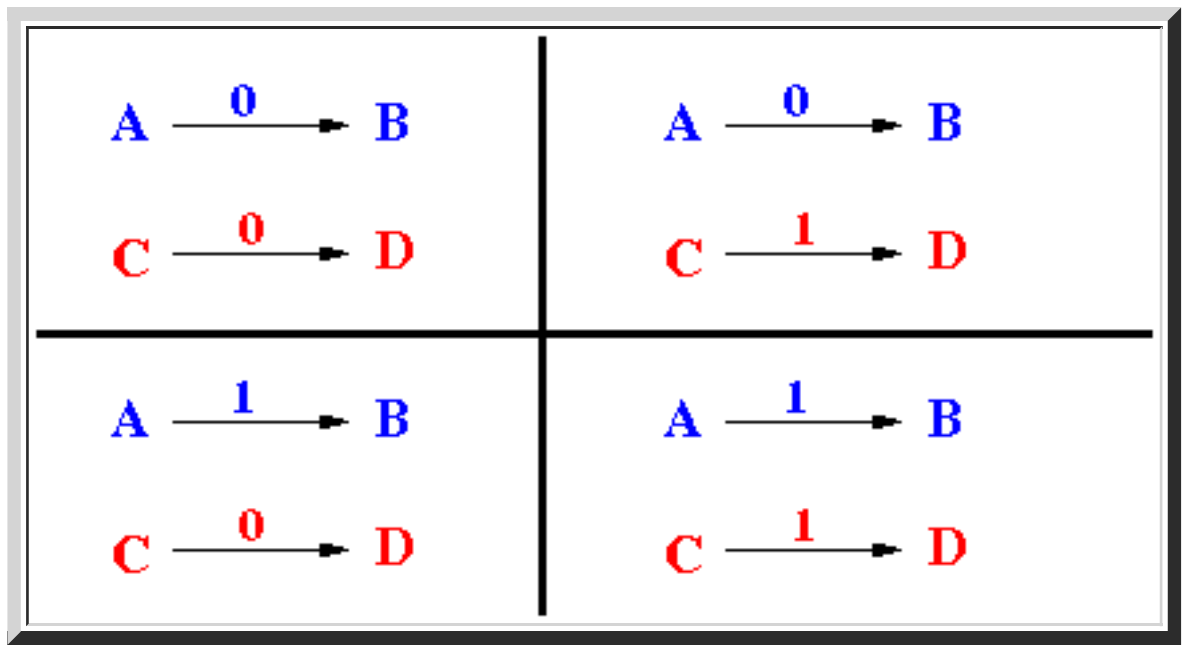
    - **Through** the ***redundancy*** (of **data**) in the **PN code**

- **How the PN code protect against collision (a "highly simplified" explanation using *Amplitude* modulation)**

  - Consider ***all*** **possible scenarios** that the transmissions **A ⇒ B** and **C ⇒ D** can **interact** with each other:

    - At the **bit level**, the **senders** **A** and **C** can transmit either a **"1" bit** or a **"0" bit**

  - There are then **4 possible interference patterns**:



Namely:

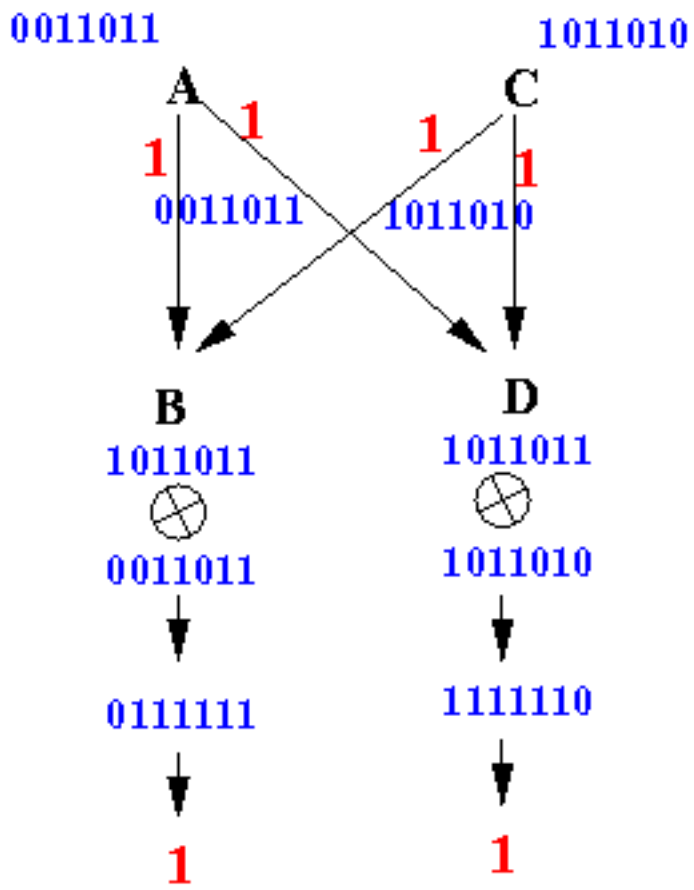
- **A** transmits **0** to **B** and **C** transmits **0** to **D** at the ***same*** **time**
- **A** transmits **0** to **B** and **C** transmits **1** to **D** at the ***same*** **time**
- And so on...

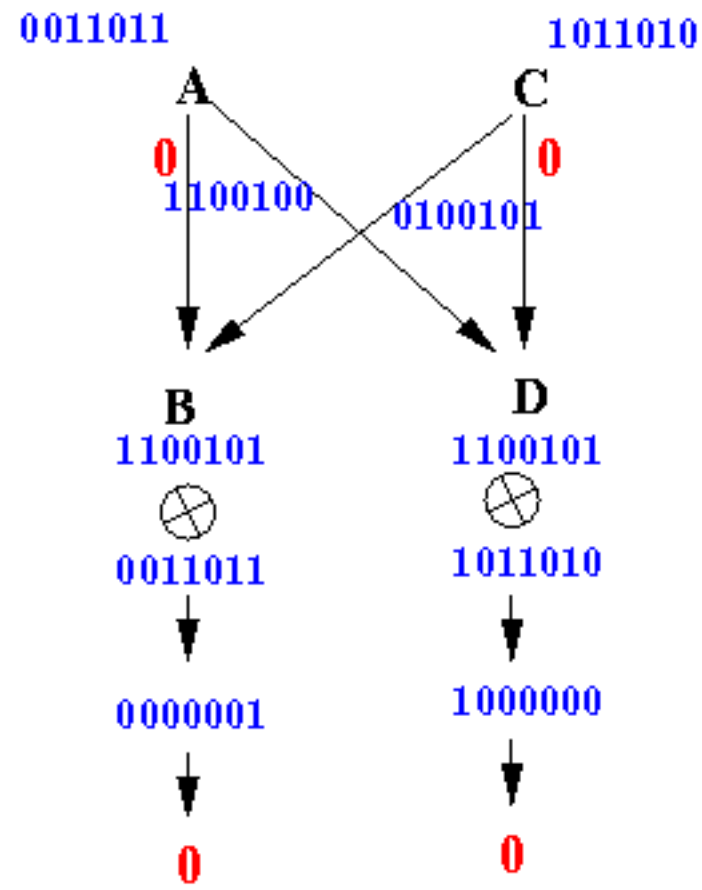
Let's consider for **each case** whether the **receivers** **B** and **D** can **decode** the transmission **correctly**....

- The **following figure** consider these **interferences:**

- **A** sends "**1**" and **C** sends "**1**" (at the ***same* time**)
- **A** sends "**0**" and **C** sends "**0**" (at the ***same* time**)

What the receivers **B** and **D** will **decode**:





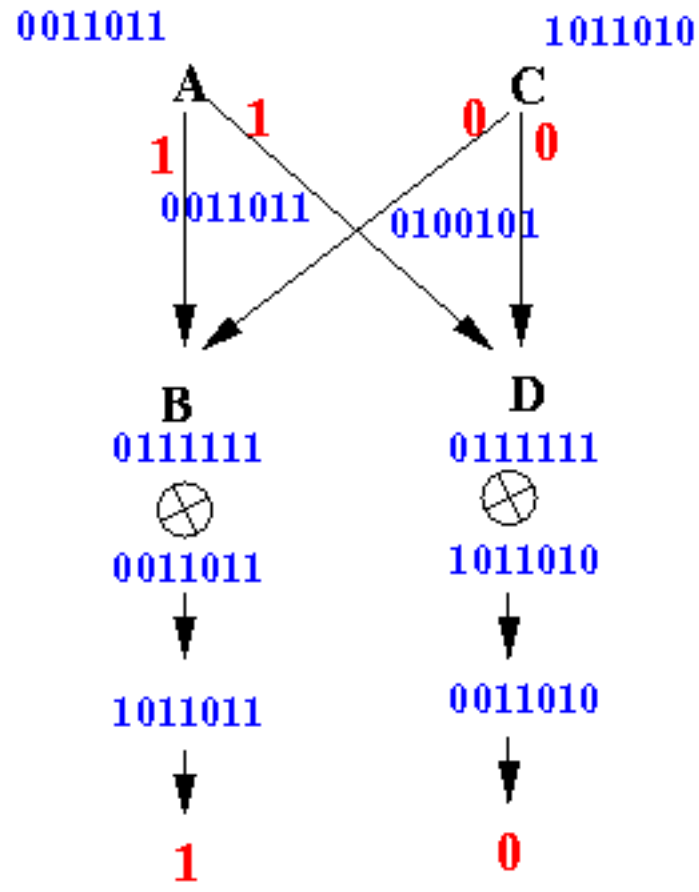
Result:

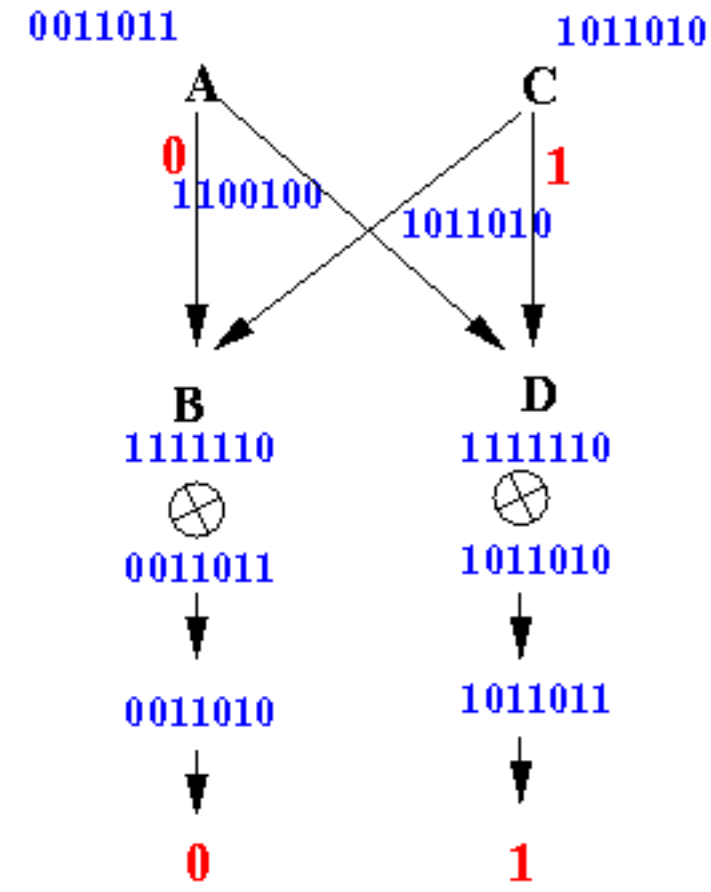
- **Both** receivers **B** and **D** will **decode *correctly* despite the interference**

- The **following figure** consider these **interferences:**

  - **A** sends "**1**" and **C** sends "**0**" (at the ***same* time**)
  - **A** sends "**1**" and **C** sends "**0**" (at the ***same* time**)

What the receivers **B** and **D** will **decode**:





Result:

- **Again, both receivers B and D will decode *correctly* despite the interference**

- **Conclusion:**

  - Using these **specific chips**, each **sender/receiver pair** is able to **decode the transmitted bit** correctly.
  - Therefore, the **transmitting nodes** are able to **share** the **transmission medium**

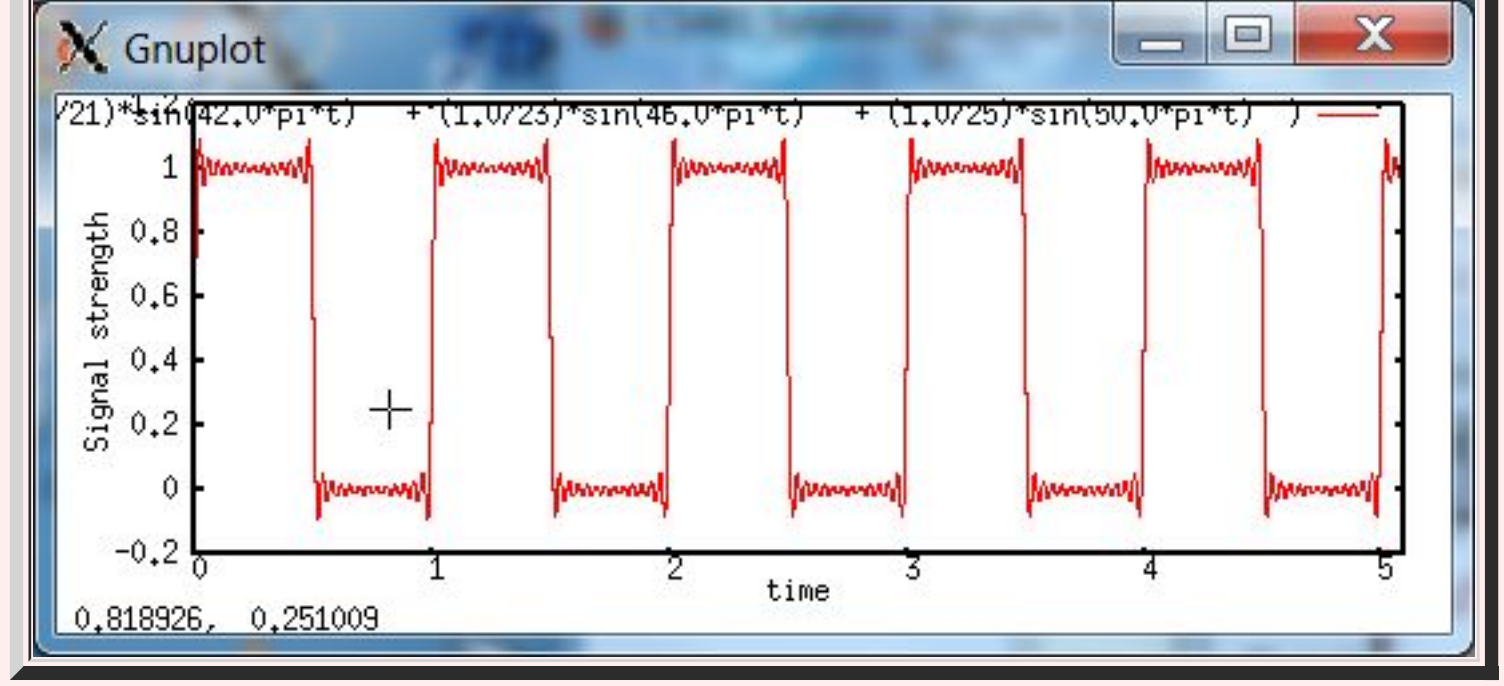
    (The **direct-sequence spread spectrum** has the **"anti-jamming"** property !!!).

- **Comment...**

  - The **real explanation** is much more **complex**...

  - **Direct Sequence Spread Spectrum** trasnmission:

    - A **digital signal (e.g.: 10101010...)** is **composed** of ***many* different frequencies** (see: **click here**)

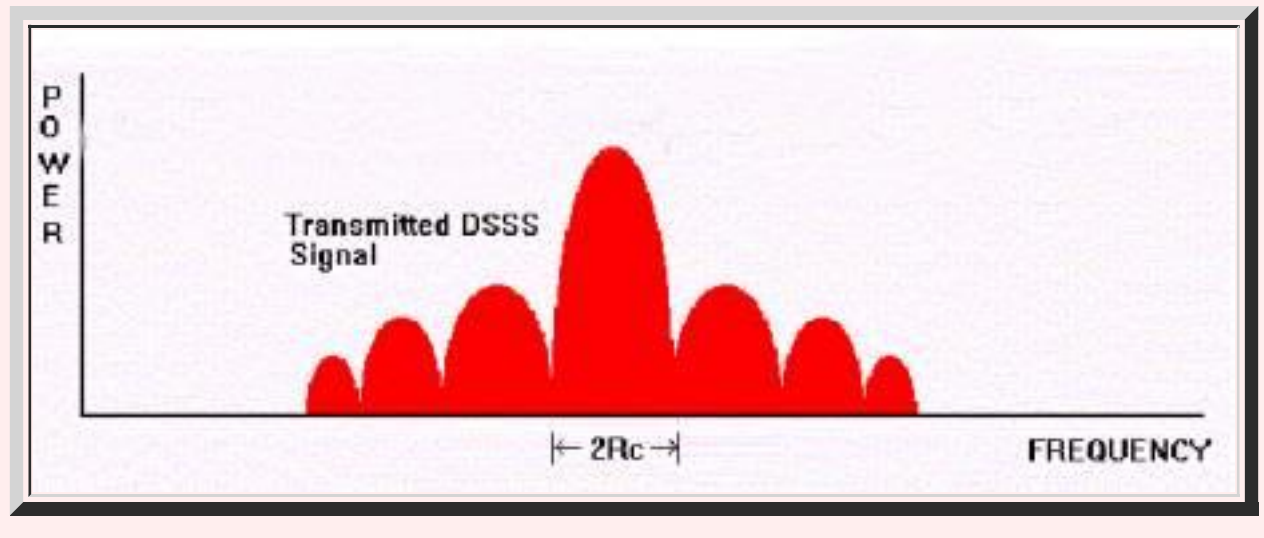


(This **decomposition** is called the ***Fourier* transform**)

- The **result** of using the **Psuedo Noise (PN) code** to **transmit** the **data** is:

  - **Transmit** the **data** using a **spectrum** of **frequencies**

  The **signal strength** used in **DSSS** transmissions look like this:



- A **sender** and **reeiver** will use ***filters*** to **target** the **spectrum** of **frequencies** contained in the **PN code** to **extract** the **transmitted bits**

  - **Receiver:**

    - The **receiver** will **tune in** to the ***same* set** of **frequencies** defined by the **Pseudo Noise (PN) code** to pickup the **data**

- **Code Correlation**

  - ***Different* PN codes** can be **alike** or **different**

  - The **measure** of **"likeness"** is the ***correlation***, defined as:

$$\frac{1}{\infty}\int$$

$$R_{corr}(T) = \frac{1}{T} \int_0^T p_i(t)\, p_j(t+T)\, dt$$ (See: click here)

The function $p_i(t)$ and $p_j(t)$ are the **wave forms** of the **two PN codes**.

(Don't worry, you don't need to know the details of **correlation** - it's there for completeness sake....)

- The **correlation function** has the following **properties**:

  - $0 \le R_{corr} \le 1$
  - $R_{corr} = 1$ if the **PN codes** ate **identical**
  - $R_{corr} = 0$ if the **PN codes** ate ***uncorrelated***

Intermediate values (between 0 and 1) indicate **how much** the **codes have in common**.

- **Property:**

  - If **2 PN codes** are ***highly*** **correlated**, their **collisions** will be more **likely** to cause ***decoding*** **errors**

Code Division Multiplexing

- **Code Division Multiplexing**
  - We **saw** in the **previous example** that:

    - ***Multiple* transmissions (= collision)** can be ***decoded* correctly** using **Direct Sequence Spread Spectrum** technique
      - **BTW**, this does **not** come for **free**

      - The **bandwidth** used to **transmit 1 bit** in **DSSS** is ***N* times** higher

        (***N*** = the **chip length**)

  Therefore:

    - **Direct Sequence Spread Spectrum (DSSS)** can allow ***multiple* sender/receiver** to ***share*** the **broadcast** transmission medium

  - **Code Division multiplexing (CDM):**

    - **Code Division Multiplexing (CDM)** = **sharing** a **broadcast transmission medium** using **PN code** transmission

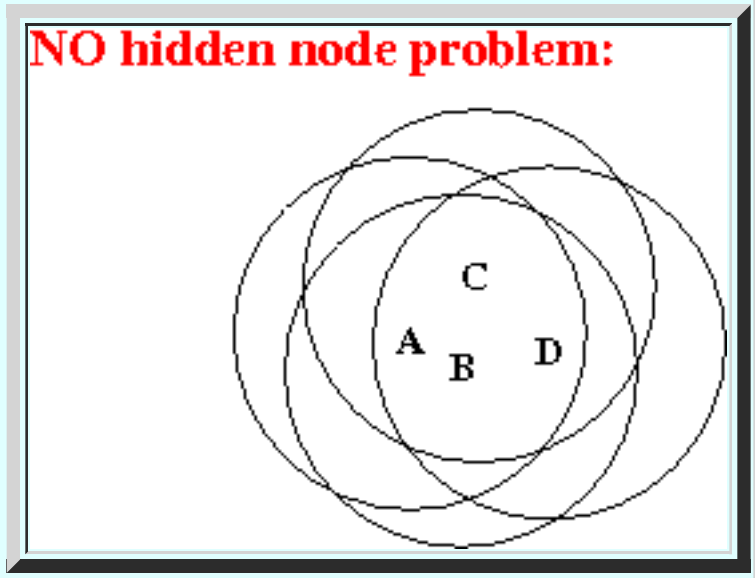
  - **Recall** we have learn **2 medium sharing techniques** previously:

    - **Time Division Multiplexing (TDM)**
    - **Frequency Division Multiplexing (FDM)**

    CDM is the **third** way to **share** a **transmission medium**

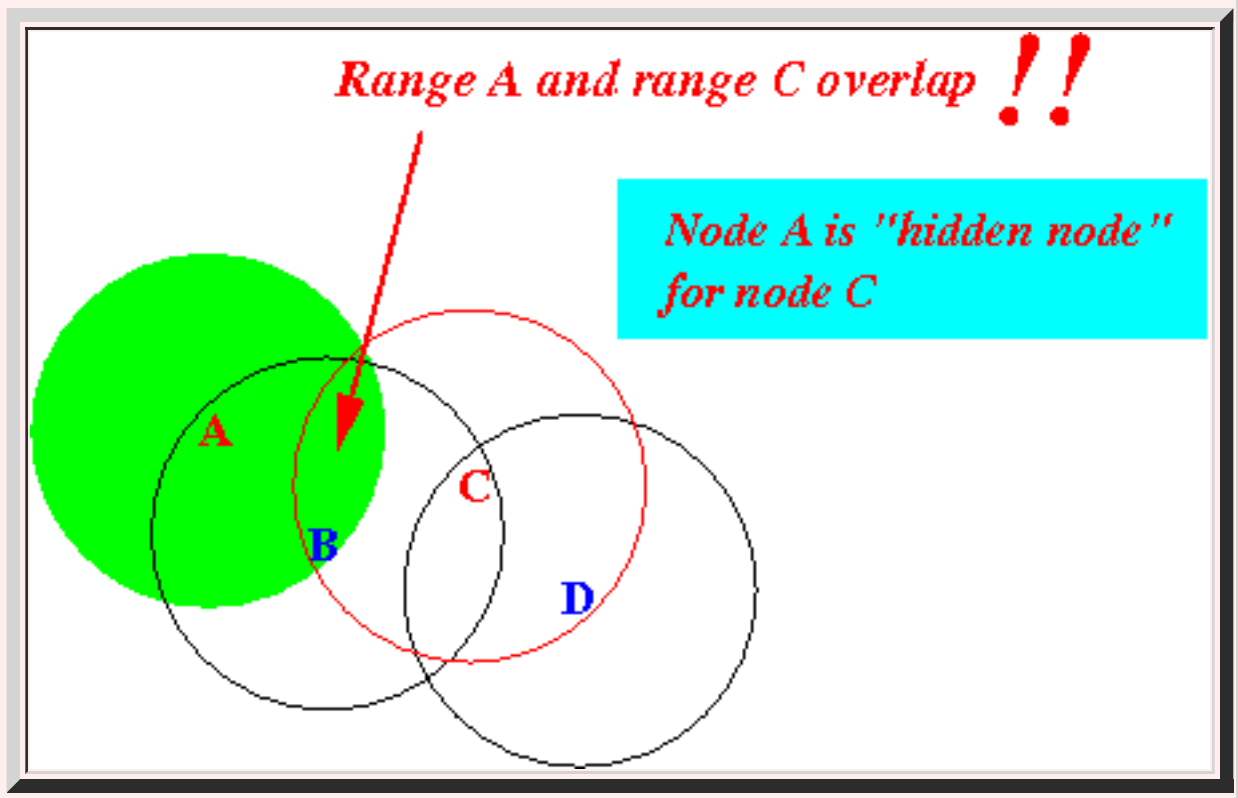
The 802.11 Medium Access Protocol - Channel sensing and no collision detection

- **Recall: 802.11 operational environment**
  - **Channel sensing** is ***useful***:
    - If there are **no hidden node**:



carrier sensing will ***avoid*** a **collision**

  - **Channel sensing** is ***not*** **foolproof**:
    - Due to the ***hidden*** **node problem**



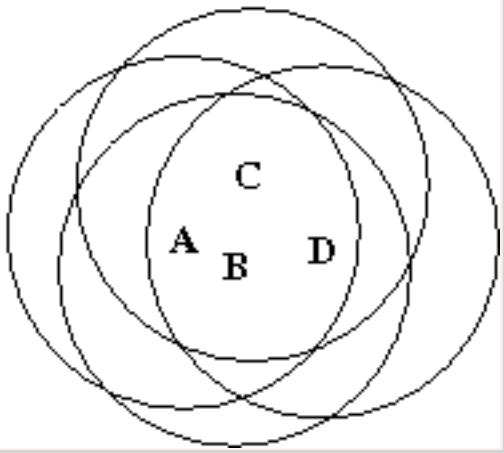
- **Channel sensing in 802.11**
  - **IEEE 802.11** uses **channel sensing**:

    - When a **node** ***senses*** **(detects)** an **ongoing transmission**, it will ***not*** start a **transmission**

      Reason:

      - The **configuration** may be like **this**:

        NO hidden node problem:

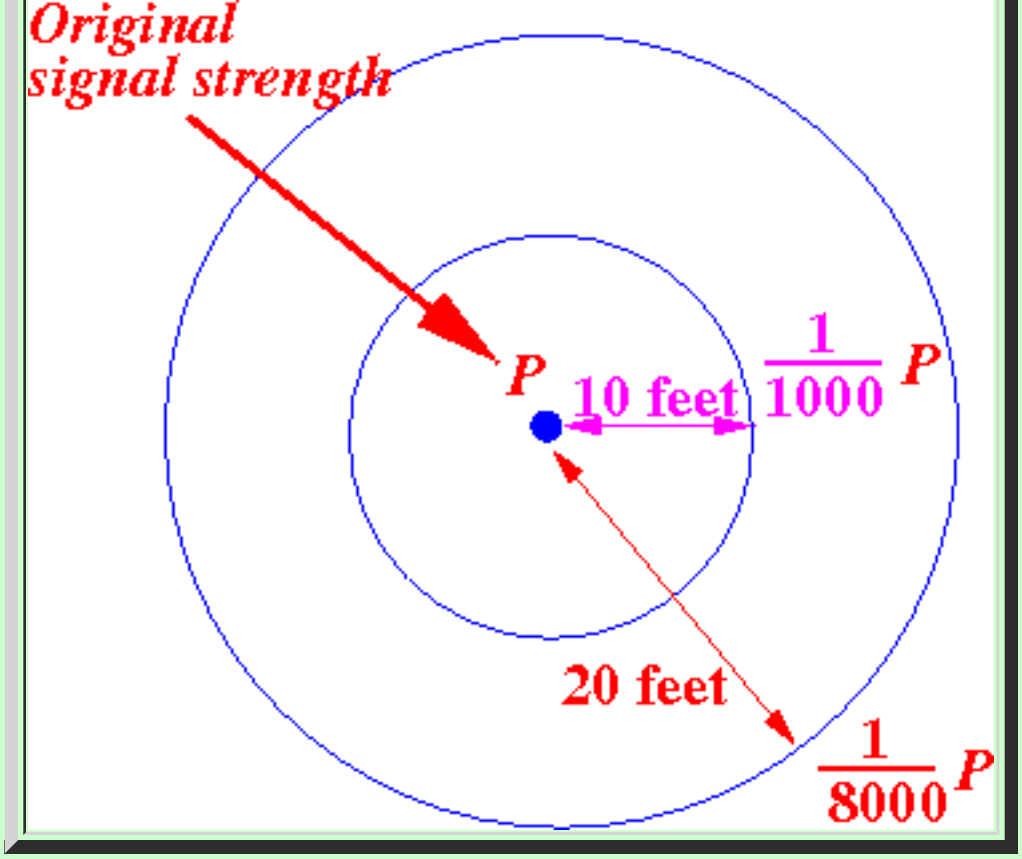
        Sensing will be ***highly*** **effective** in **avoiding collision**

- **IEEE 802.11 does not use collision detection**
  - **Reason:**

    - It is ***technically*** **not possible** to **detect collision** in **wireless transmissions** due to the **hugh** ***difference*** in **signal strength**

  - **Detailed explanation**:

    - **Signal** ***strength*** **attentuates** ***very*** **rapidly** in **wireless transmissions** (proportional to $distance^3$):



- The **signal level** of the **node's *own* transmission** is ***extremely* high**:
  - **because** the ***send* antenna** of the **node** is ***close* by** to its ***receive* antenna**....

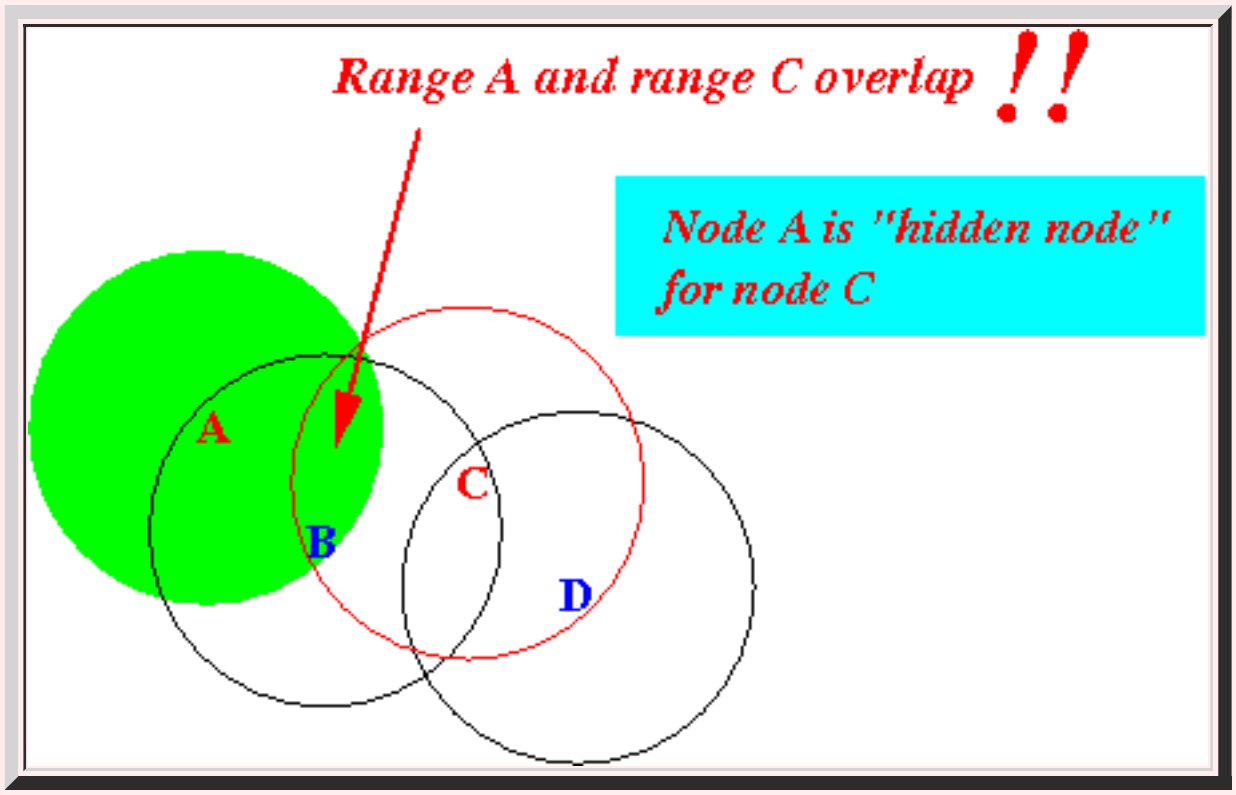
- **But**: the **signal level** of ***another* (colliding) transmission** is ***extremely* weak**
  - **because** the ***send* antenna** of the ***other* node** is ***far* away** from the **node's *receive* antenna**....

- **Result:**
  - The **node's *own* transmission** will ***overwhelm*** the **signal** from the ***other* node(s)**

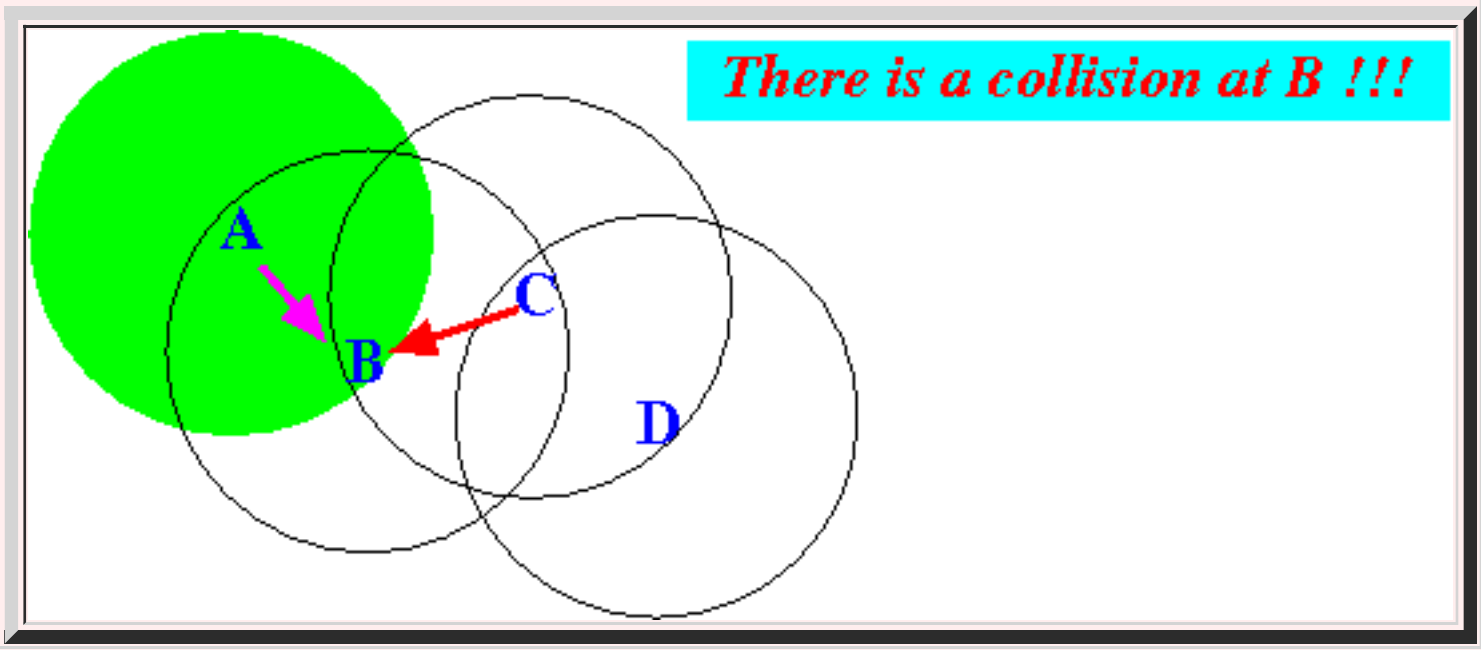
    ===> **Collision detection** in **802.11** is **not technically feasible**

MAC level Acknowledgement --- but prioritized ACK transmission

- **Recall: 802.11 operational environment**
  - **Channel sensing** is ***not*** **foolproof**:
    - Due to the ***hidden*** **node problem**



    - There can be **collisions** that a **node** can **not detect (hear)**:



- **Faster recovery of lost frames**
  - To **recover** ***lost*** **frame** as **quickly** as possible:
    - The **receiver** must send an **ACK** for each **correctly received frame**

- If **sender** does **not receive** an **ACK** for the **transmitted frame** within a **timeout period**:
  - The **sender** will **retransmit** the **frame**

The **acknowledgement scheme** is call **MAC level Acknowledgement** because:

- the **protocol specification** is **defined** in the **Medium Access Control (MAC) layer**

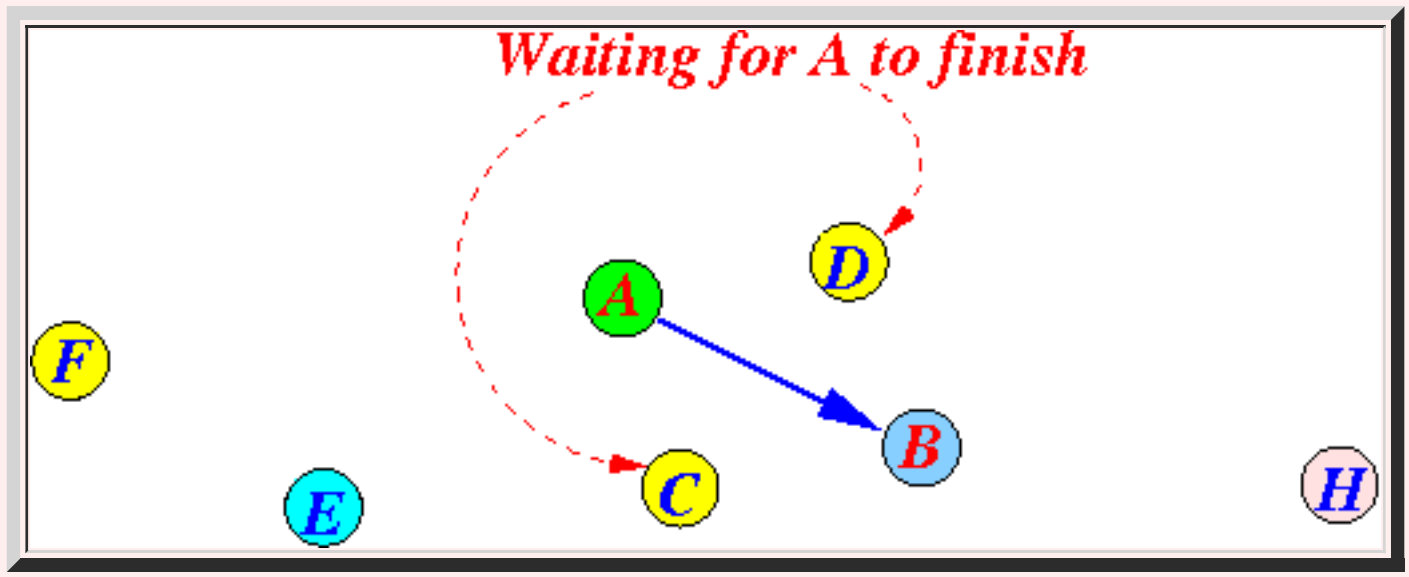
  (**IEEE 802.11** is **just like Aloha**)

- The **MAC level ACK protocol** used in **IEEE 802.11**:
  - **Sender**:
    - transmits a **frame** to **receiver**
    - **Wait** for an **ACK frame** within a **time out period**

    - If **time out** expires, **retransmit** the **frame**

  - **Receiver**:
    - **receive** a **frame**
    - **Send** an **ACK frame** back to the **sender**

- **Problem implementing the MAC level ACK scheme**
  - **Consider** the following **scenario**:
    - **Node *A*** is currently **transmitting** to **node *B***:



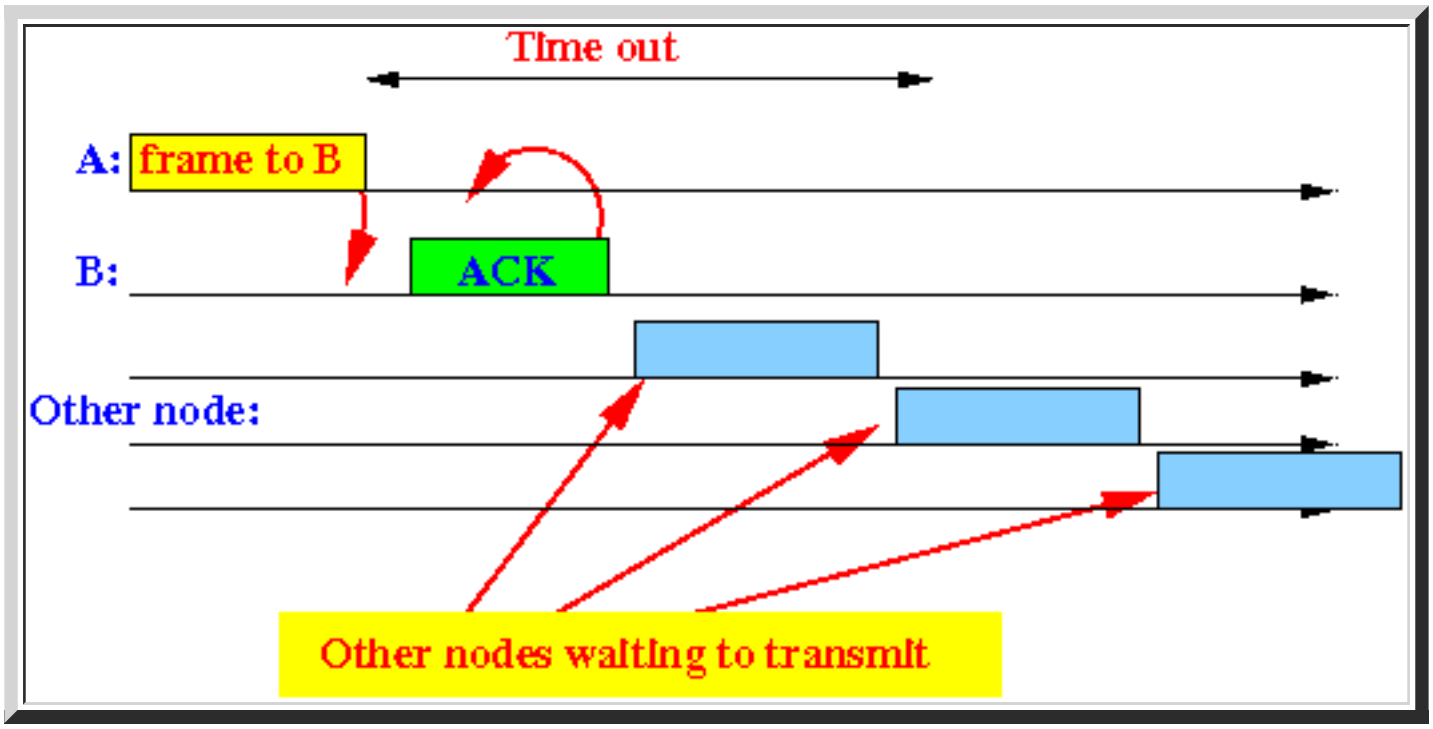
Nodes *C* and ***D*** are ***waiting*** for **node *A*** to finish.

- We have the following ***interesting* scenario**:

  - **Node *B*** will **transmit** the **ACK frame** (to ***A***) as soon as **node *A*** is ***finished***

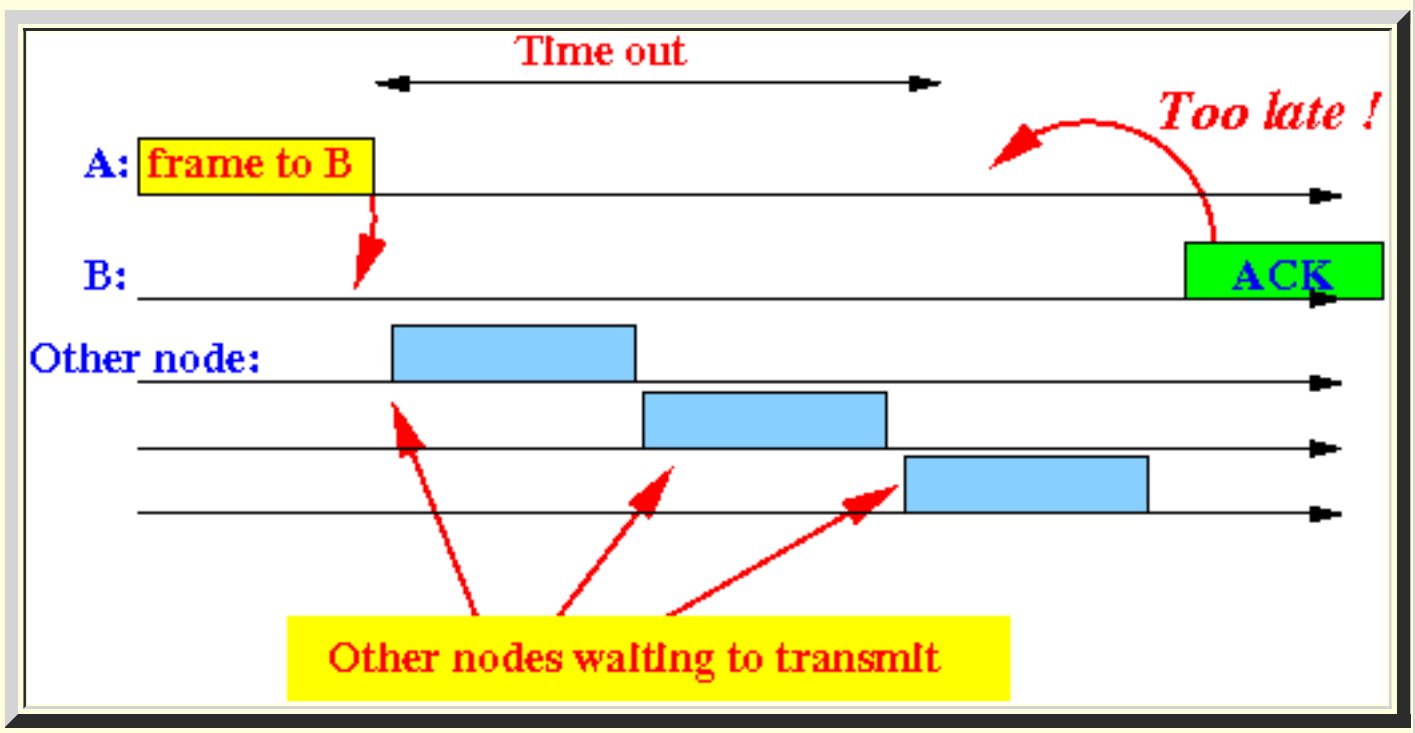
  - **But**: **nodes *C*** and ***D*** will ***also* transmit as soon as node *A*** is ***finished*** !!!

We **need** to **ensure** that the **ACK transmission** goes **before** all ***other* transmissions**:



- ***Possible* transmission order**:

  - **Transmissions** from ***other* nodes** gets ***ahead*** of **node *B***:



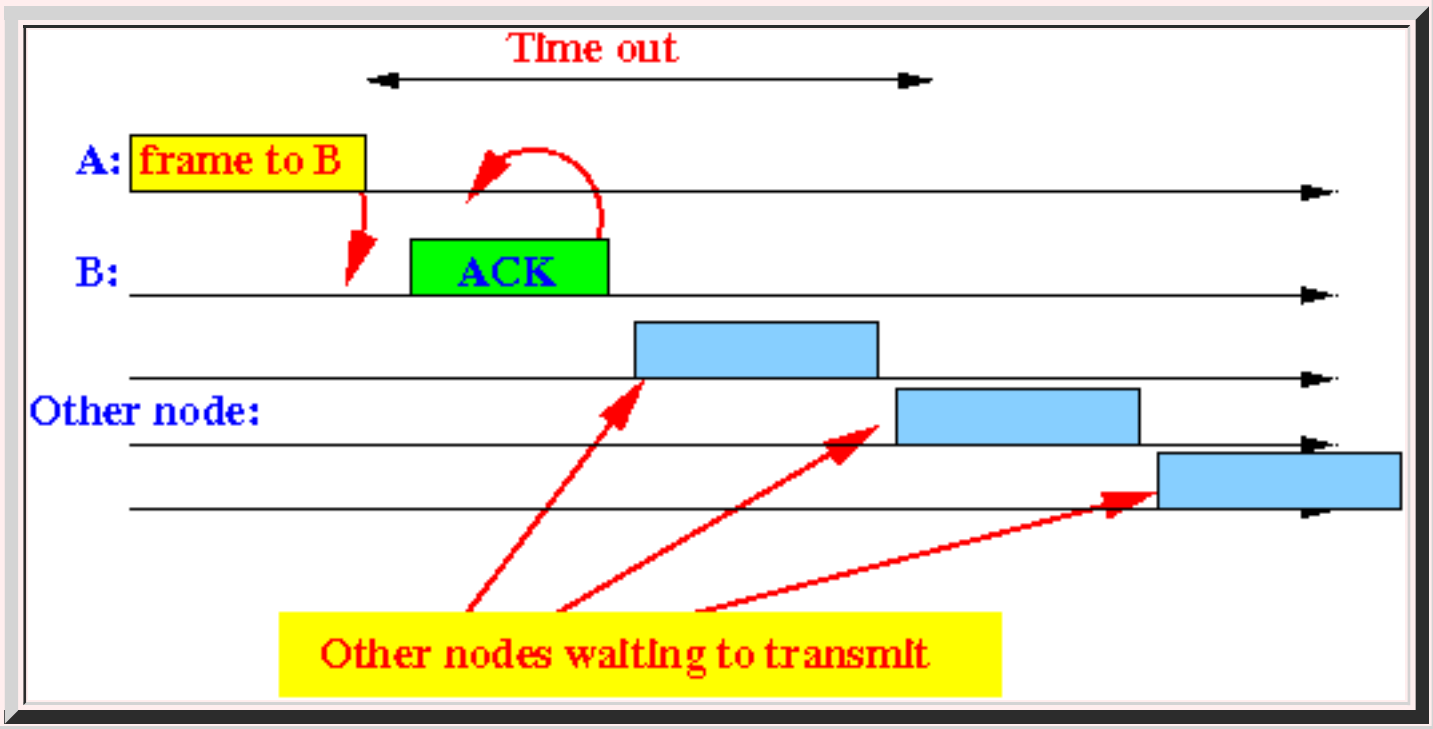
The **ACK frame** of **B** will be **too late** !!!

- **Node *A*** will ***timed out*** !!!!

- **$64,000 question:**

  - **How** can we **make sure** that an **ACK frame** is **transmitted *first*** ???

    Graphically:



- **Note:**

  - The ***main* difficulty** of this **problem** is the **fact** that:

    - **Nodes** that are ***ready* to transmit** are ***not* aware** of **each other** !!!

    - **Futhermore**:

      - The **nodes** must **not send messages** to **each other** to **solve** this **problem** !!!

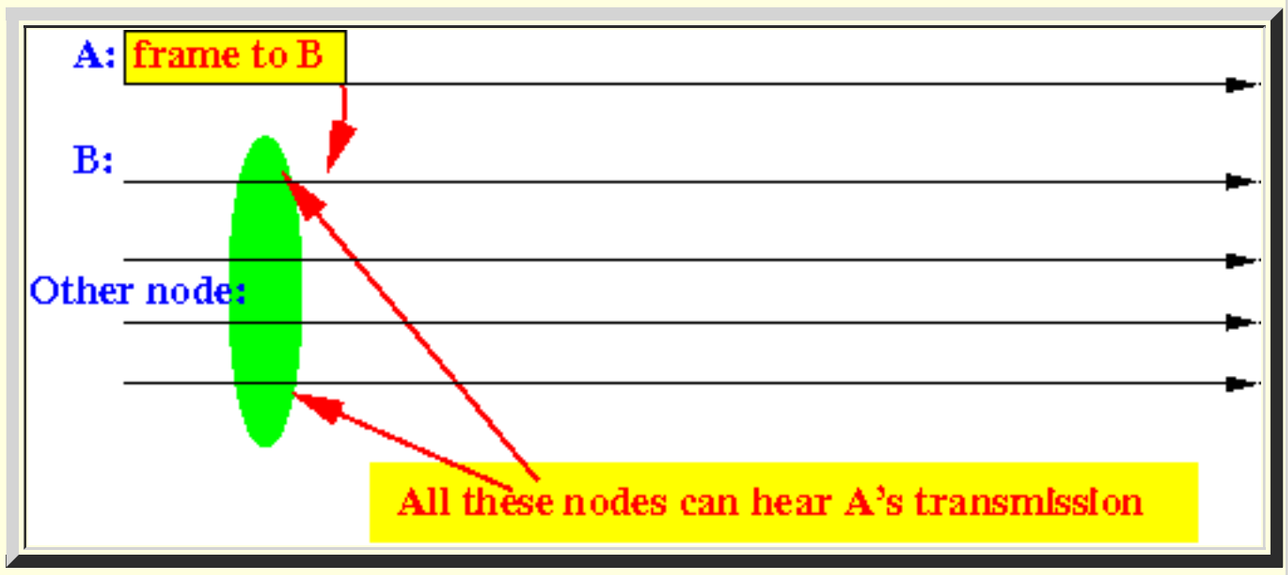
        (Because the **time** it takes to **send messages** to **each other** will **increase** the **delay** --- the **ACK frame** will to **too late** !!!)

The **solution** is **pretty ingenious**....

Prioritizing transmissions in (wireless) IEEE 802.11

- Assigning *priority* in a *distributed* manner
  - **Scenario:**

    - ***Multiple*** **nodes** are ***waiting*** for a **transmission** (**node *A***) to **finish**:



The **(multiple) nodes** will **transmit** when the **transmission** is **complete**

  - **Problem description:**

    - The ***multiple*** **nodes** are **not aware** of **each other**

    - We must **make sure** that **node *B*** transmits ***first***

    - We must do so ***without*** using ***any*** **messages** communicated between the **nodes**

  - **Solution:**

    - When a **node** want to **transmit** a **frame**:

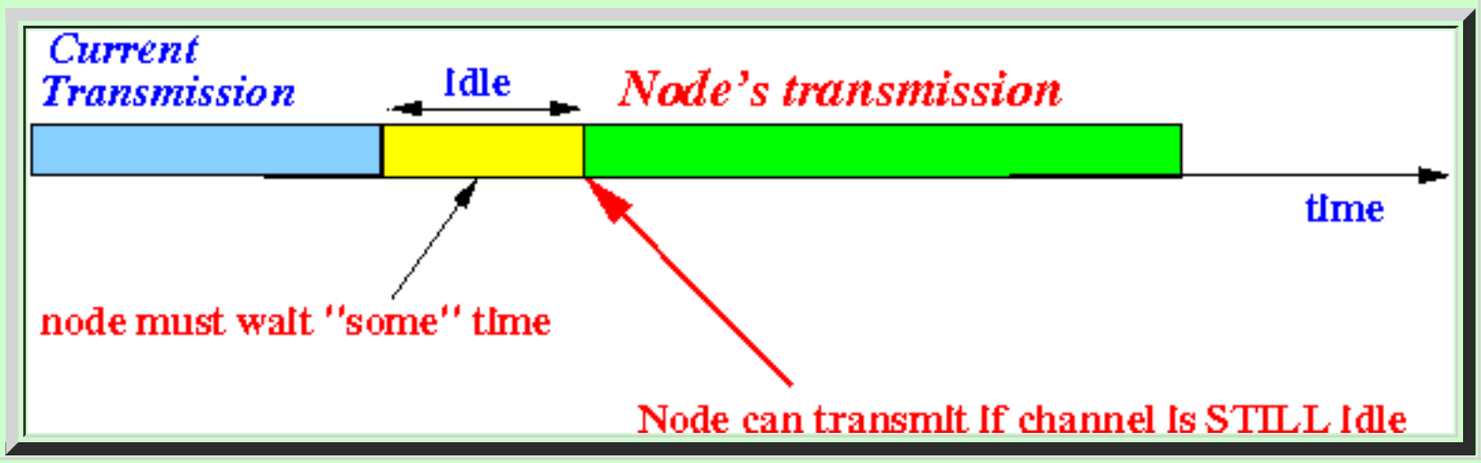
      1. The **node** must ***first*** **wait ("defer")** a ***predefined*** **amount of time** before the **node** can **transmit** its **frame**

      ---

      2. If the **channel** is **idle** after **waiting** the **predefined time**:

         - The **node** will **transmit** the **frame** (immediately)

           Schematically:



      Otherwise:

         - The **node** must **wait** until the **current transmission** is **over**

         - **Then** go **back** to **step 1**: ***defer*** the **predefined wait time** again before **transitting** !!!

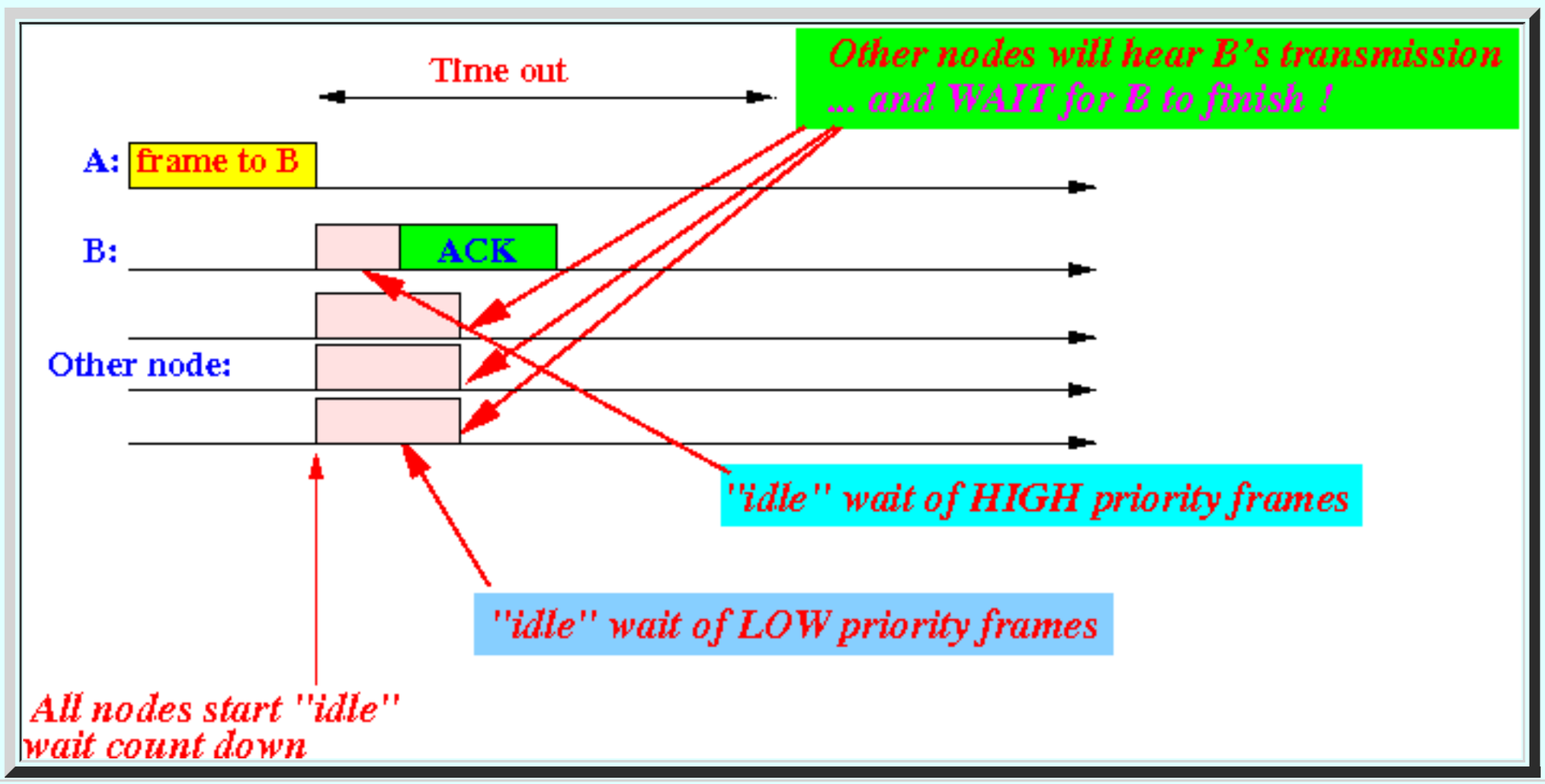
- **Key** to **prioritizing** the **ACK transmission**:

  - ***Different*** **types** of **frames** will use ***different*** **waiting (defer) time** !!!

    In fact:

    - ***Higher*** **priority frames** will use a ***shorter*** **waiting time** !!!

Example:

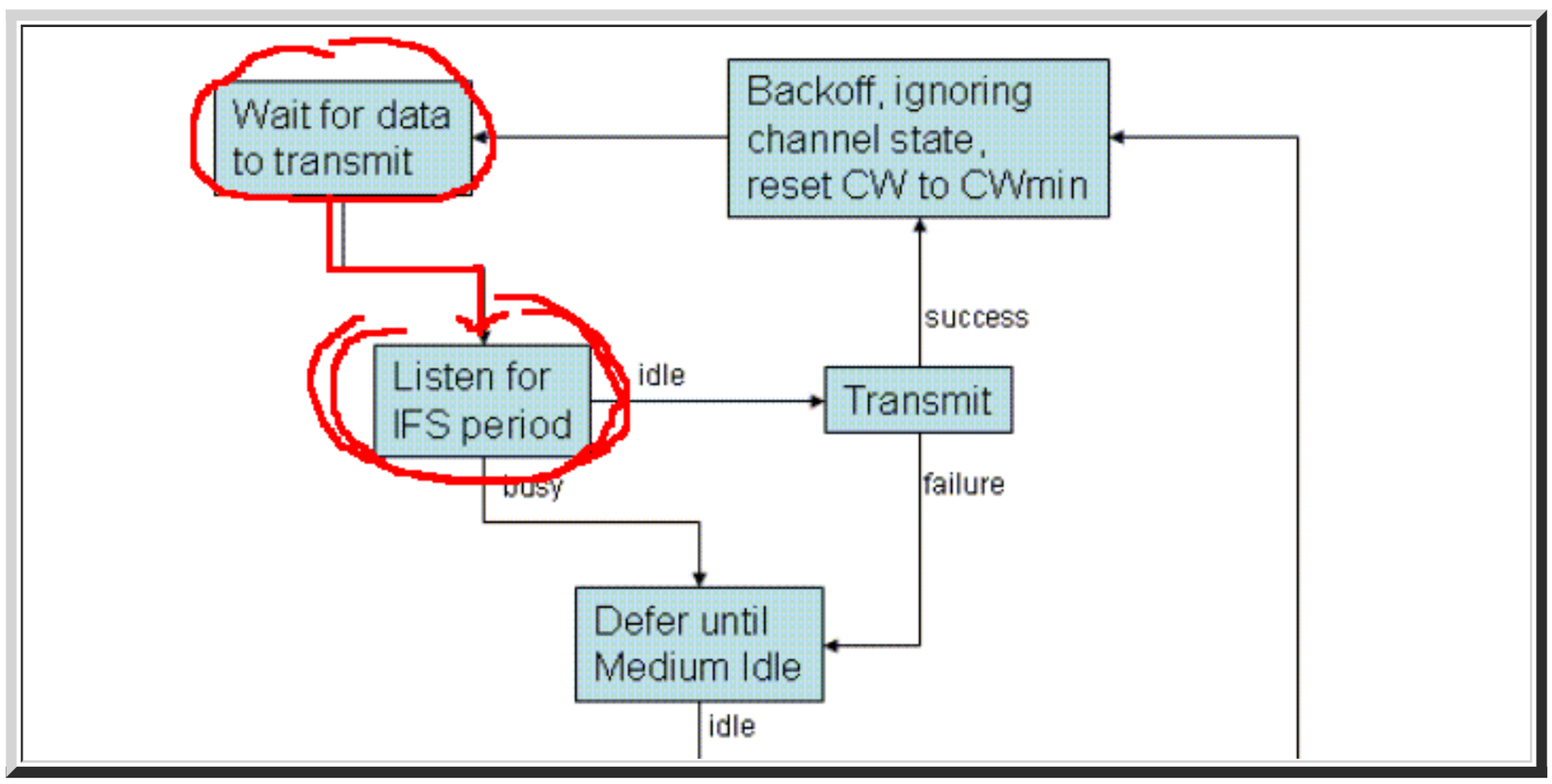


- **Interframe Spacing**

  - ***Interframe*** **Spacing**

    - **Interframe Spacing (IFS)** = the **period** of time that a **(transmitting) node** must ***wait*** **(and listen)** before the **node** can ***start*** **transmitting** a **frame**

  - ***Partial*** **flow chart** of the **802.11 Medium Access Protocol**:



Explanation:

- A **(transmitting) node** must **listen (monitor)** the **transmission medium** for ***IFS*** **amount of time**

- The **IFS** for ***different*** **types** of **frames** are ***different*** !!!

- If there are ***no*** **transmissions** for ***IFS*** **amount of time**, the **node** will ***transmit*** its **frame**

- If the **node** detect a **transmission during** the ***IFS*** **wait time**:

  - The **node** will ***back off*** and **try again** later

- **The various types of IFS in 802.11**
  - **Types of IFS:**

    - **SIFS** = **Short** Interframe Spacing (has the ***shortest*** **duration**)

      - **SIFS = 28 µsec**

    Usage:

      - **SIFS** is used as **sensing delay** for transmitting **ACK frames** !!!

    - **PIFS** = **Point Coordination Function (PCF)** Interframe Spacing (has the ***second*** **shortest duration**)

      - **PIFS = SIFS + 1 Slot Time = 78 µsec**

    Usage:

      - **PIFS** is used as **sensing delay** by a **base station** (= ***coordination point***) that ***operates*** in a ***special*** **coordination mode**

    - **DIFS** = **Distributed Coordination Function (DCF)** Interframe Spacing (has the **"*normal*" duration**)

      - **PIFS = SIFS + 2 × Slot Time = 128 µsec**

    Usage:

      - **DIFS** is used as **sensing delay** for transmitting **data frames**

    - **EIFS** = **Extended** Interframe Spacing (has the **longest duration**)

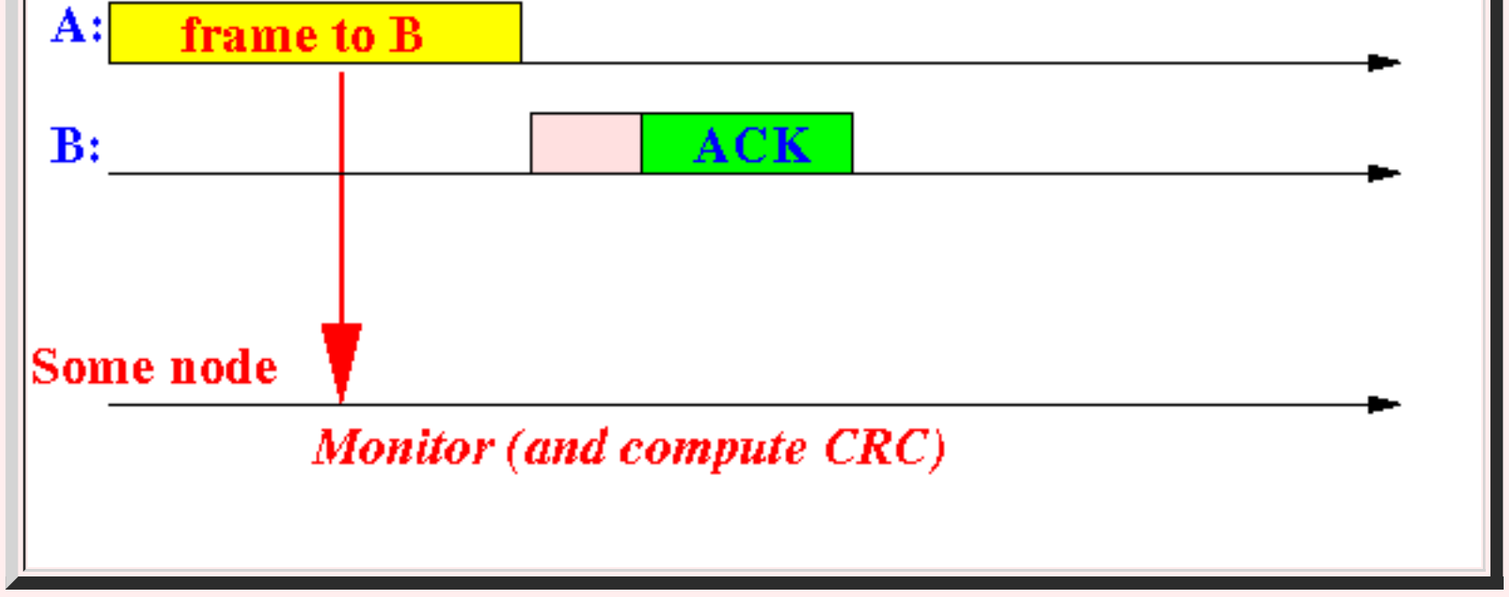
      - **EIFS** = **Ack frame duration + SIFS + DIFS**

    Usage:

      - **EIFS** is used by a **transmitting node** that **received** a ***corrupted*** **data frame**

    Explanation:

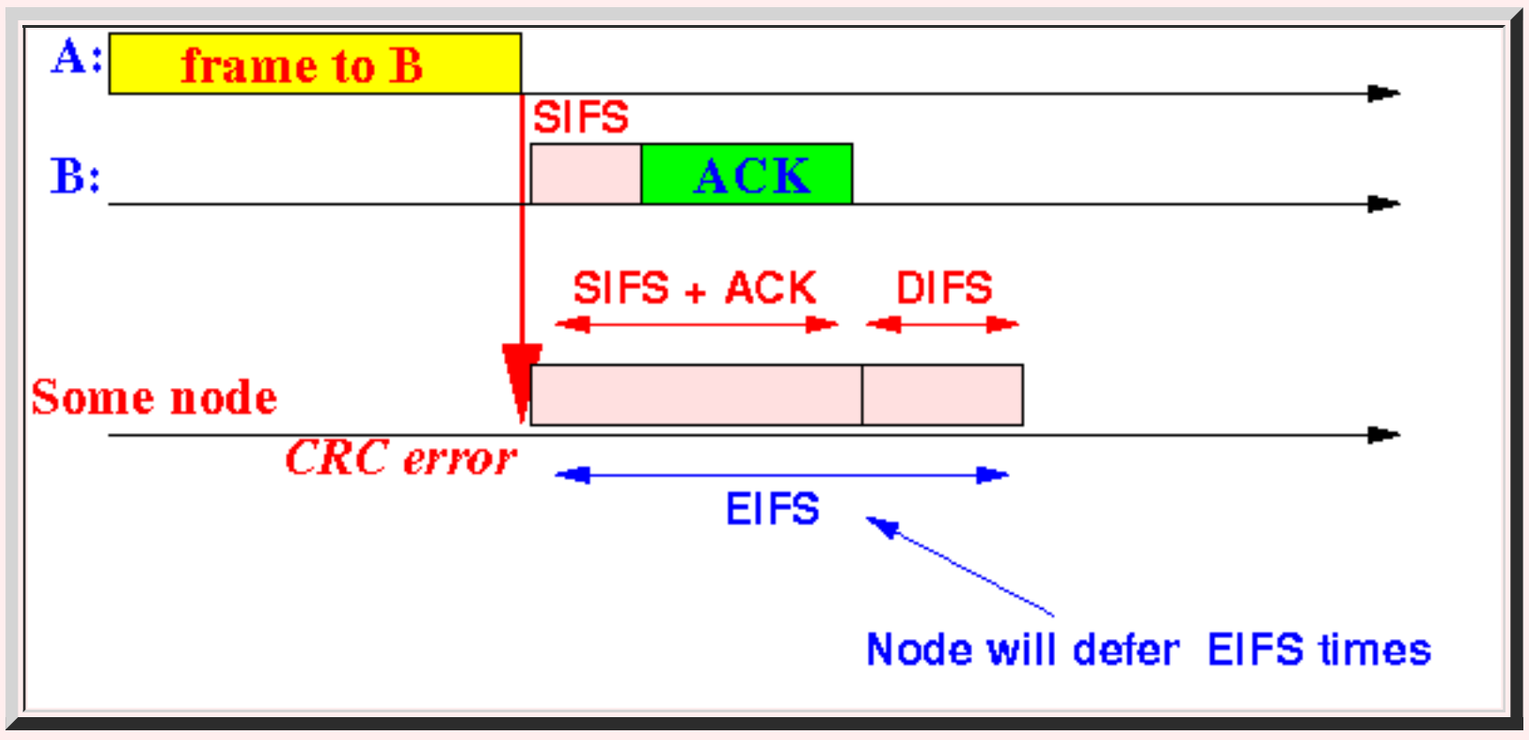
      - ***Every*** **node** in **IEEE 802.11** will **continuously receive** and **CRC check** ***all*** **frames** !!!



- If a **received frame** contains an **(CRC) error** then:

  - A **node** will defer **EIFS** duration **instead** of **DIFS** before **transmitting** a **data frame**.

  Schematically:

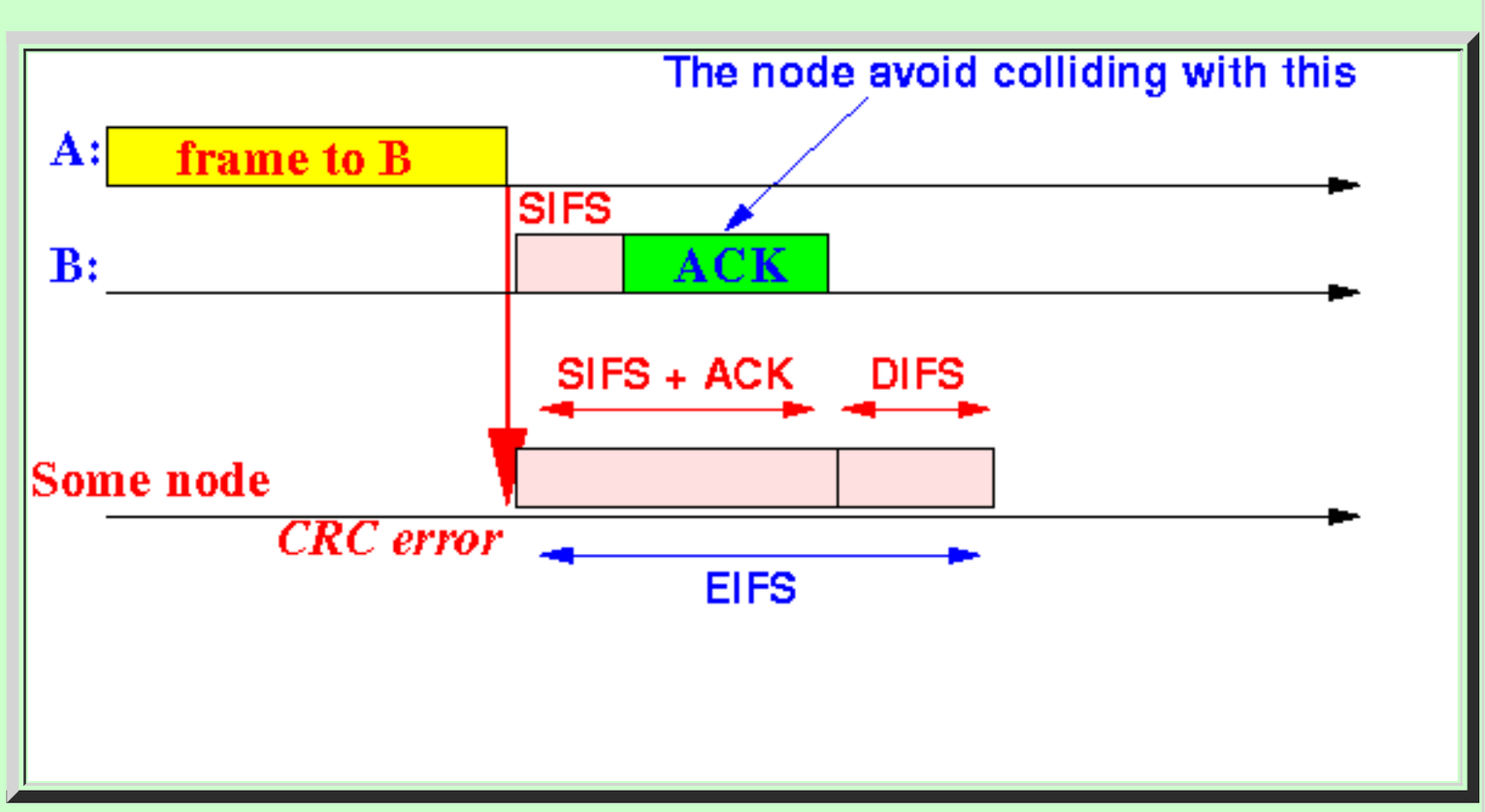


- **Reason**:

  - The ***corrupt*** **frame** may be a **data frame** for ***another*** **node** (far away -- that's why it was corrupt)

  - ***That*** **(other) node** may have **received** the **data frame** ***correctly*** !

  - By ***waiting*** **EIFS**, the **node** will let the ***other*** **node** transmit the **ACK frame without collision** !!!



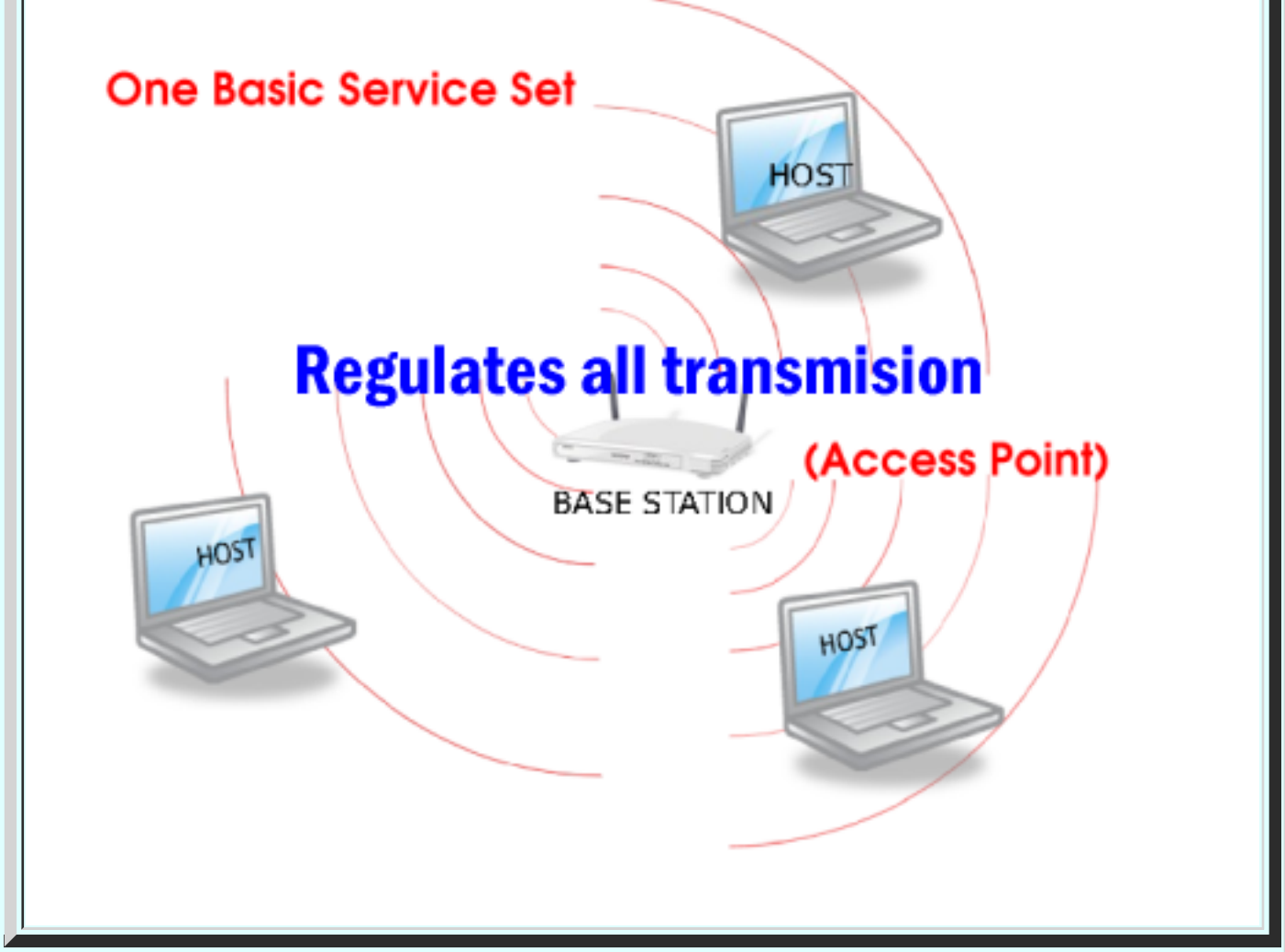
Overview of 802.11 Medium Access Control protocol

- **IEEE 802.11 uses slotted transmission**
  - **802.11 network** uses ***slotted*** **transmissions**:
    - **Slotted transmissions** (in general) will **reduce** the **likelihood** of **collisions**
    - Since **nodes** in the **802.11** network ***cannot*** detect **collisions**, there is no **"wasting"** of slots when **collision** is **detected**....
  - **Duration** of a **slot** in **802.11** is:
    - **σ = 20 µsec**

    (A message can be longer than one slot).

- **Modes of MAC protocol operations**
  - The **802.11 MAC protocol** can operate in **2 different modes**:
    - **Centralized mode without contention**
    - **Distributed mode with contention**

- **The Point Coordinated Function**
  - The **Point Coordinated Function (PCF)**:
    - **Point Coordinated Function** = the **centralized (operation) mode** of the **802.11** protocol
  - **Operational overview:**
    - in **Point Coordinated Function**, the ***access*** **(= permission)** to the **transmission medium** is **controlled** by the **Access Point (AP)/base station**:



- **Medium access coordination**:

  - The **AP point coordinator transmits CF-*Poll* frame** to **stations (= wireless clients, laptops)**.

  - A **node (e.g., laptop)** that received a **CF-Poll frame *must* transmit (exactly) one frame**.

    - ***If*** a **node** does **not** have **any frames** to transmit, then it ***must*** transmit a ***null* frame**.

- **Property:**

  - There are ***no* collisions** in the **PCF mode** --- because **transmissions** are ***regulated*** by the **access point** !!!

Note:

  - **Most 802.11 devices** do ***not* support** the **Point Coordinated Function operational mode** !!!

- **The Distributed Coordinated Function**
  - The **Distributed Coordinated Function (DCF)**:

    - **Distributed Coordinated Function** = the ***distributed*** **(operational) mode** of the **802.11** protocol

      - The **DCF** is the ***most*** **commonly used operational mode**

  - **Operational overview:**

    - **Each node determine/decide on its own** whether it is **safe** to **transmit** a **frame**

    - In the **distributed mode**, **nodes/stations** will **content (= "fight") for the channel**

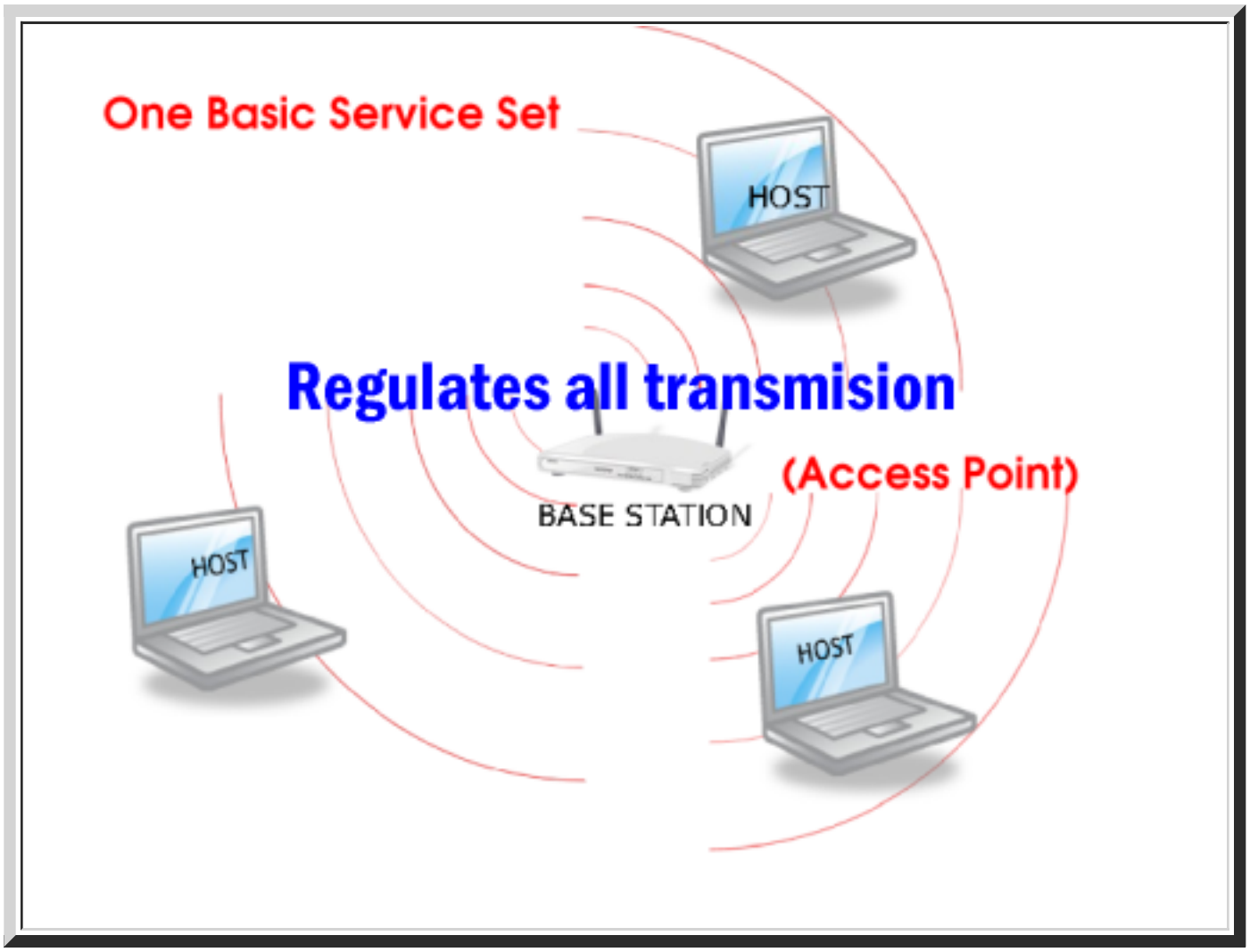
      - There will be **collisions** in the **distributed mode**

Note:

- ***All*** **802.11 devices** support the **DCF function**

Intro to the Point Coordinated Function

- **The Point Coordinated Function**
  - In the **PCF mode**, the ***Access Point*** will ***coordinate* (= regulate)** the **access** to the **transmission medium**:



  - **Base station:**
    - A **base station maintains** a **list** of **nodes** that is ***associated*** with the **base station**

      (The ***registration* protocol** will be **explained later**)

- **Node start up**
  - **Initial operational mode** of a **node** that **starts up**:
    - When a **node** in **IEEE 802.11** first **starts up**:

- The **node** will **operate** in the **Distributed Coordinated Function (DCF) mode**

We will discuss the **DCF** mode **later**....

Entering the Point Coordinated Function operational mode

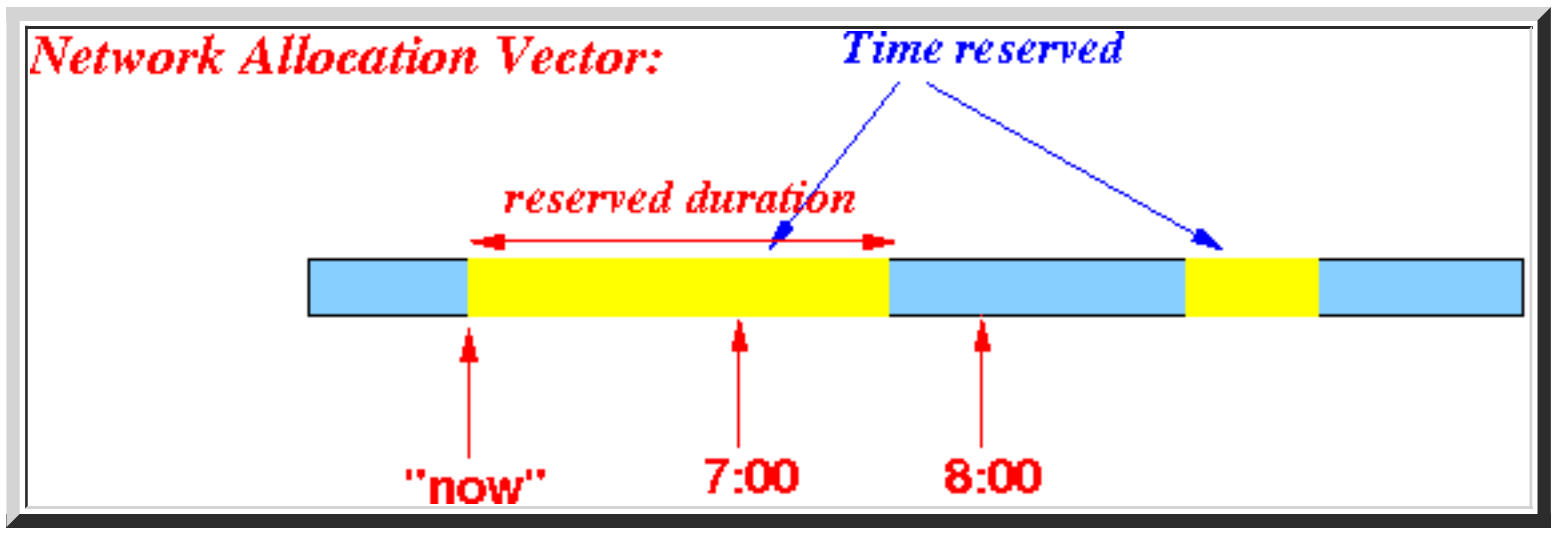
- **Preliminary: the "Network Allocation Vector"**
  - **Network Allocation Vector:**

    - **Network Allocation Vector** = a **state variable** maintained by a **wireless node (e.g., laptop)** for **IEEE 802.11 protocol**

    - When a **period** in the **Network Allocation Vector** is **"allocated"**:

      - The **wireless** will ***not*** **initiate** a **data frame transmission** on the **IEEE 802.11 network** on its own.

Example:



Explanation:

- There is a **reservation** in the **NAV** from **now** until **7:30**

- This **node** will **not start** a **data frame transmission on its own** during the **reservation period**

  - The **node** will ***still*** transmit **ACK frames** (because an **ACK frame must** be sent **ASAP** !!!)

  - The **node** will **only transmit** a **data frame** when it is ***polled*** **(= given permission to do so)**

- ***Entering*** **into the PCM operational mode**
  - **Initial (default) operation:**

    - **Initially**, the **IEEE 802.11 network** operates in the **DCF mode**

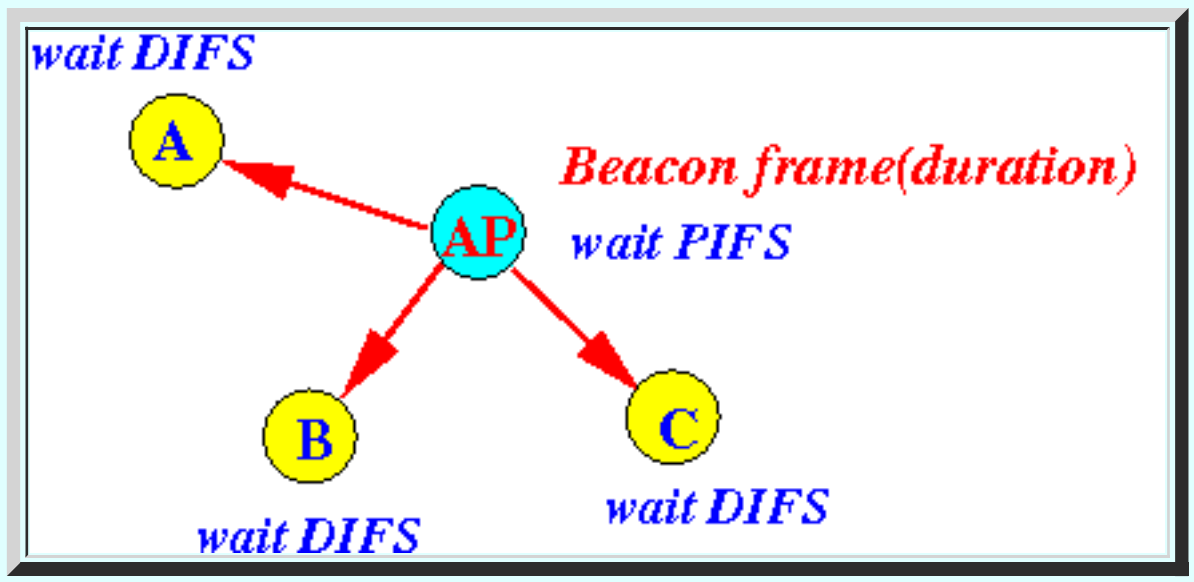
  - **Entering** the **PCM mode**:

    - The **Access Point (AP)** (coordinator) ***defers*** **(= listens)** for **PIFS** **(= PCF interframe spacing** time and **transmits** a **beacon frame**

- The **beacon frame** will contain a **PCF duration value** that is the **length** of the **contention-free period**

- **All nodes** that are **associated** with the **base station** (and in its range) will **receive** the **beacon frame**

Graphically:

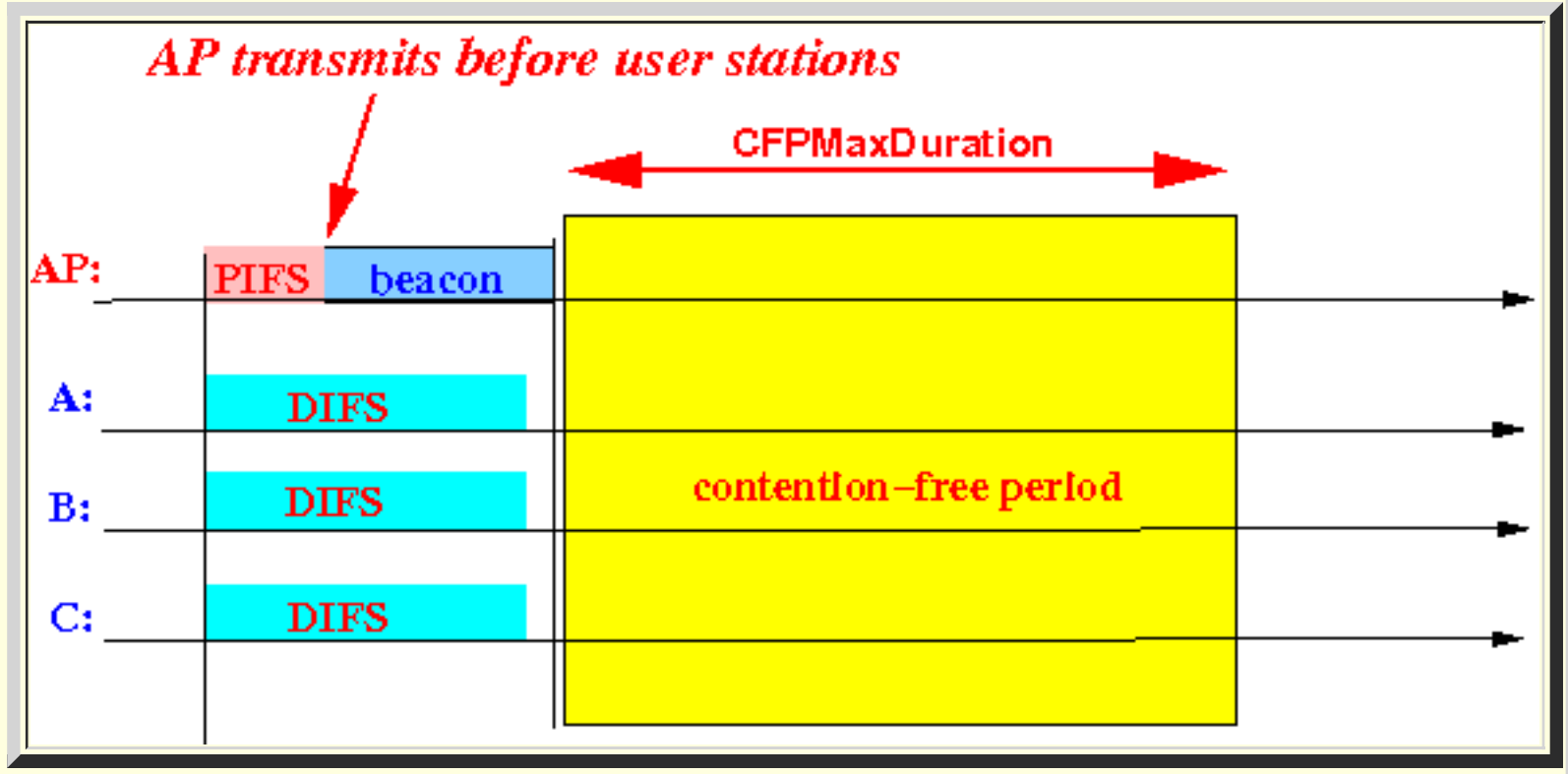


- **Note:**

  - Because a **data frame transmission** must ***defer*** **DIFS** time (and **PIFS < DIFS**):

    - The **beason frame** has a ***higher*** **transmission priority** than **data frames** !!!

    Graphically:

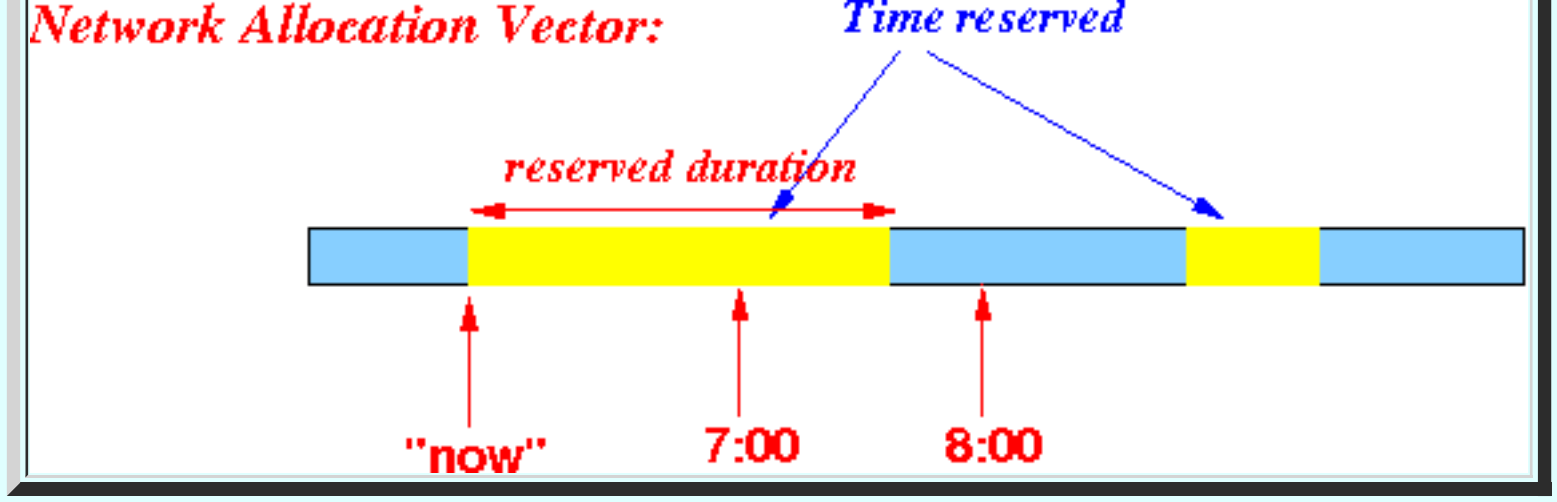


    (The **contention period duration** parameter is called **CFPMaxDuration**)

- When **stations (e.g., laptops)** receive the **beacon frame**:

  - The **node** will **reserve** the **given duration** in their ***Network Allocation Vector*** **(NAV)**

  Graphically:



The **time reservation** made in the **Network Allocation Vector** will:

- **Prevent** a **node (station)** from **transmitting** during the **reserved period**

(More on the **NAV** when we discuss **DCF**)

- **Result:**

  - **Nodes** will **not transmit** by its **own initiative**

  - **Nodes** will ***only*** **transmit** ***data*** **frames** when the **node** receives a ***permission***

    - The **permission** is **given** as a **poll frame** (**originating** from the **Access Point**)

The Point Coordinated Function MAC protocol

- **IEEE 802.11 MAC protocol operation in PCM mode**
  - **IEEE 802.11 MAC protocol** for an **Access Point**:
    - **Transmitting** a **data/poll frame**:
      - **Access Point** must ***defer*** **PIFS** time.
        - To **allow** an **ACK frame** to be **transmitted** as **early** as possible !!!
      - When the **channel** is **idle** for **PIFS** time:
        - The **Access Point** can **transmit** one **poll/data frame**

        Otherwise:
        - **Back off** and **re-try**
    - **Transmitting** a ***ACK*** **frame**:
      - **Access Point** must ***defer*** **SIFS** time.
      - When the **channel** is **idle** for **PIFS** time:
        - The **Access Point** can **transmit** the **ACK frame**

        Otherwise:
        - **Back off** and **re-try**
  - **IEEE 802.11 MAC protocol** for an **user node (station):**
    - **Transmitting** a **data frame**:

- The **station** must **wait for** a ***poll* message** from the **Access Point**

- When the **station** received the **poll frame**, the **station** can ***immediately*** (without waiting) **tranmsmit** the **data frame**

- **Transmitting** an **ACK frame**:

  - A **station** can **transmit** an ***ACK* frame** ***without*** being **polled** !!!

    (**Because** the **ACK frame** must be **transmitted ASAP** !!!)

  - The **user node (station)** must ***defer* SIFS** time.

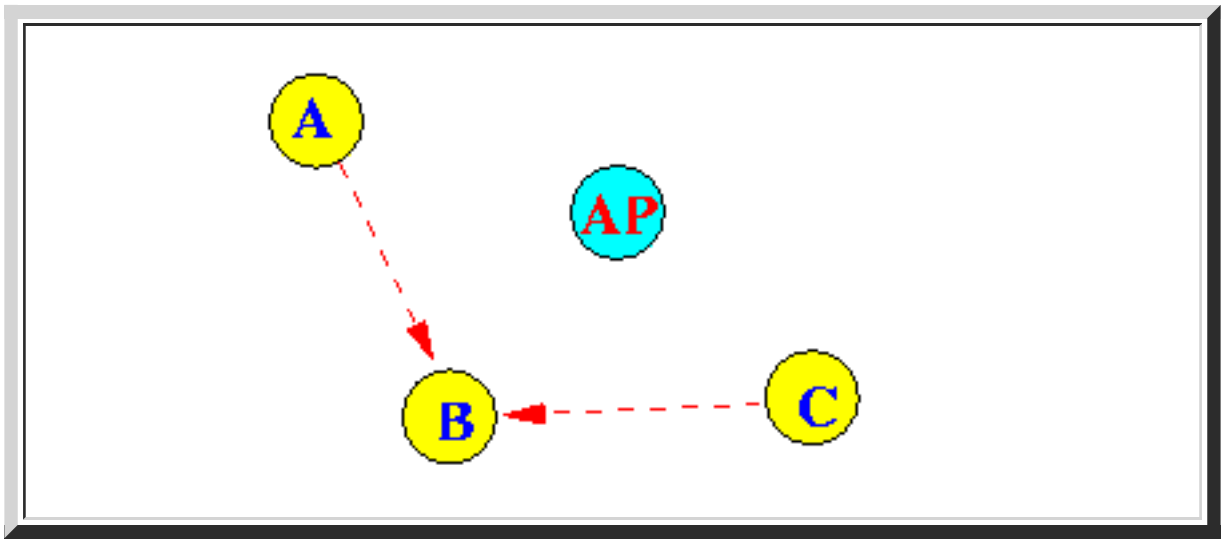
  - When the **channel** is **idle** for **PIFS** time:

    - The **Access Point** can **transmit** the **ACK frame**

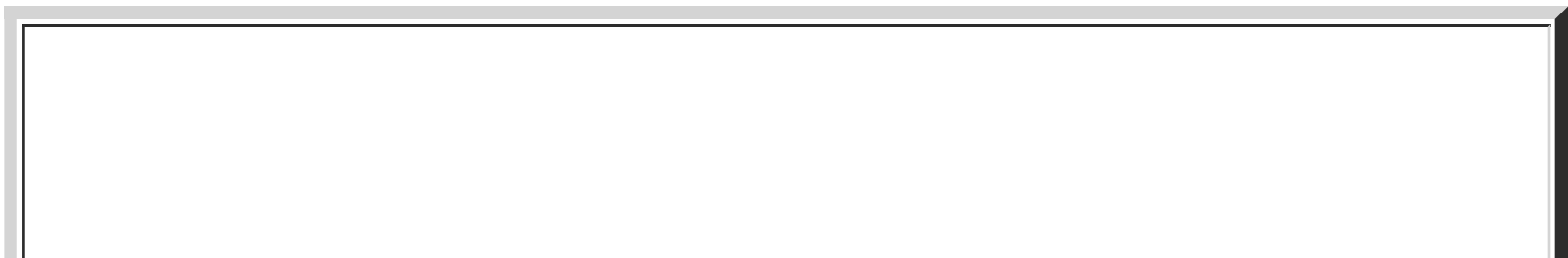
    Otherwise:

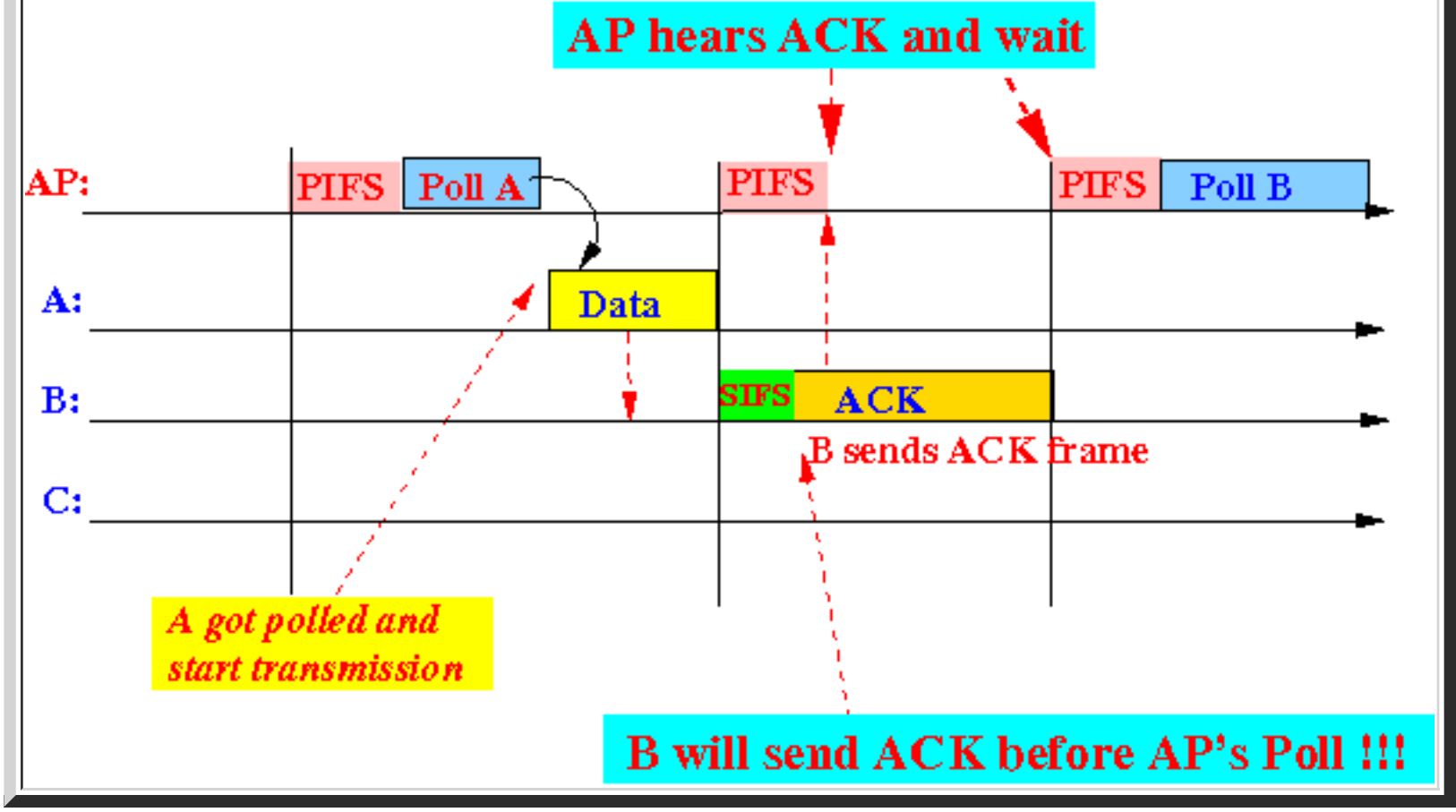
    - **Back off** and **re-try**

- **Example:** Suppose **A** and **C** want to send a frame to **B**



Sequence of events:





NOTE:

- **Node B** can send its **ACK frame** ***before*** the **AP** sends the next **poll frame** because:
  - **SIFS** duration is ***shorter*** than the **PIFS** duration !!!

- This will **prevent** the **node A** from **timing out** (and **re-transmit** the data frame) !!!

Frames used in the Point Coordinated Function MAC protocol

- **Frames sent by the Access Point in PCF mode**
  - The **AP** can send the following **frames** when it **operates** in the **PCF mode**:

    - **Data frame**
      - Frame contains data intended for one of the **station**

    - **Poll frame** (called the **CF poll frame** in **IEEE 802.11**)
      - contains a **poll message** for a **station**
        - The **station** receiving the **poll message must** transmit ***one*** **frame**
        - If the **station** has **no frame** to transmit, it **must send** a ***null*** **message**

    - **(Data + Poll) frame**
      - This is a **combined message** containing a **message *and*** a **poll** for the ***same*** **station**
        - The **station** will **receive (and process)** the **message**
        - Then the **station must** send a **message** (because it was **polled**)
        - If the **station** has **no data** to transmit, it sends a ***null*** **message**

    - **End frame**

- This **frame** will **terminate** the **PCM mode**

Effect:

- The **Access Point** ***and*** the **nodes** will use **Distributed Coordinated Function (DCF)** as the **operational mode**

Postscript

- **Postscript**
  - The **PCF mode** is **not supported** by the **Wi-Fi Alliance**:

    - **Wi-Fi Alliance** = a **trade association** that promotes **Wi-Fi (802.11) technology** and **certifies Wi-Fi products** if they conform to certain standards of **interoperability**.

      See: **click here**

  - **Consequently:**

    - ***Most* Wi-Fi devices** does **not** have the **PCF mode** built into the **device**

  - So let move on and study the **more popular operational mode**:

    - The **DCF mode**

The collision detection method

- **Collisions in 802.11**
  - **Recall**: ***anti*-jamming (collision)** in **802.11**:
    - When **2 nodes transmits** at the ***same* time**:
      - The **receivers** can ***decode*** the ***corrupted* data frame** by the use of **Direct Sequence Spread Spectrum** transmission !!!

    See: **click here**

  - **However:**
    - **Collisions** that **cause frame loss** are ***still* possible** !!!

    Frame loss can **happen** when:
    - ***More* than 2** nodes **transmit** at the ***same* time**
    - The **PN (Pseudo Noise) code** used **colliding nodes** are ***similar***

- **How does a node detects a collision in 802.11 ??**
  - ***Not*** like **Ethernet**:
    - **Recall** that the **node's own transmission** will ***overwhelm*** any **transmission** from ***other* (remote) nodes** !!!
      - **Because** the **node's receive antenna** is ***close* by** the **node's own transmit antenna** !!!
  - **Collision "detection"** in **802.11**:
    - A **node** that transmitted a ***data* frame** will **expect** an ***ACK* frame** (within a **time**

out period)

- A **collision** is ***assumed*** when:

    - The **node** does ***not*** receive an **ACK frame** within the **time out** period

The back off procedure

- **Back-off after a collision**
  - **Back off** algorithm of **IEEE 802.11**:

    - A **node** picks a **random number**

    - the **nodes** must **defer** the **(random)** ***back off*** **period** before **re-try** the **transmission**

  - **Backoff protocol** used in **802.11**:

    - **802.11** uses a ***similar*** **binary exponential backoff algorithm** of **802.3 (*Ethernet*)**

- **The 802.11 binary exponential back off algorithm**
  - The ***initial*** **range** of **random values** used in **802.11** is:

```
[ 0 ..   CWmin ]

 CWmin = 31   in 802.11b
 CWmin = 15   in 802.11g
```

  - The ***maximum*** **range** of **random values** used in **802.11** is:

```
[ 0 ..   1023 ]   (CWmax = 1023)
```

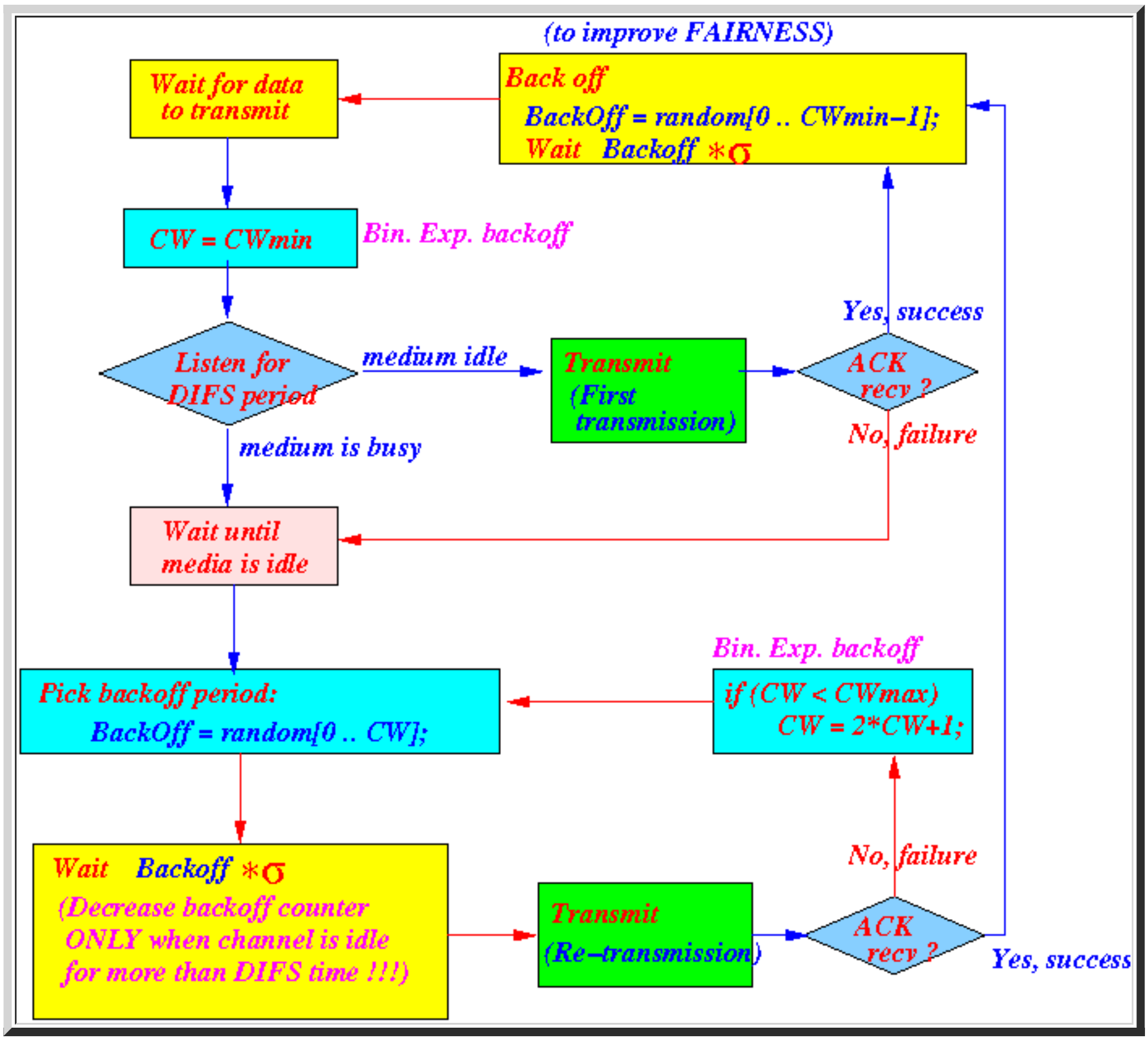
  - **Doubling algorithm:**

```
CW = CWmin;          // Initialize the range of Congestion Window values

if ( CW < CWmax (= 1023) )
   CW = 2×CW + 1;    // Doubling algorithm
```

```
Example:

CW = 15
CW = 2×CW + 1 = 2×15 + 1 = 31
CW = 2×CW + 1 = 2×31 + 1 = 63
and so on
```

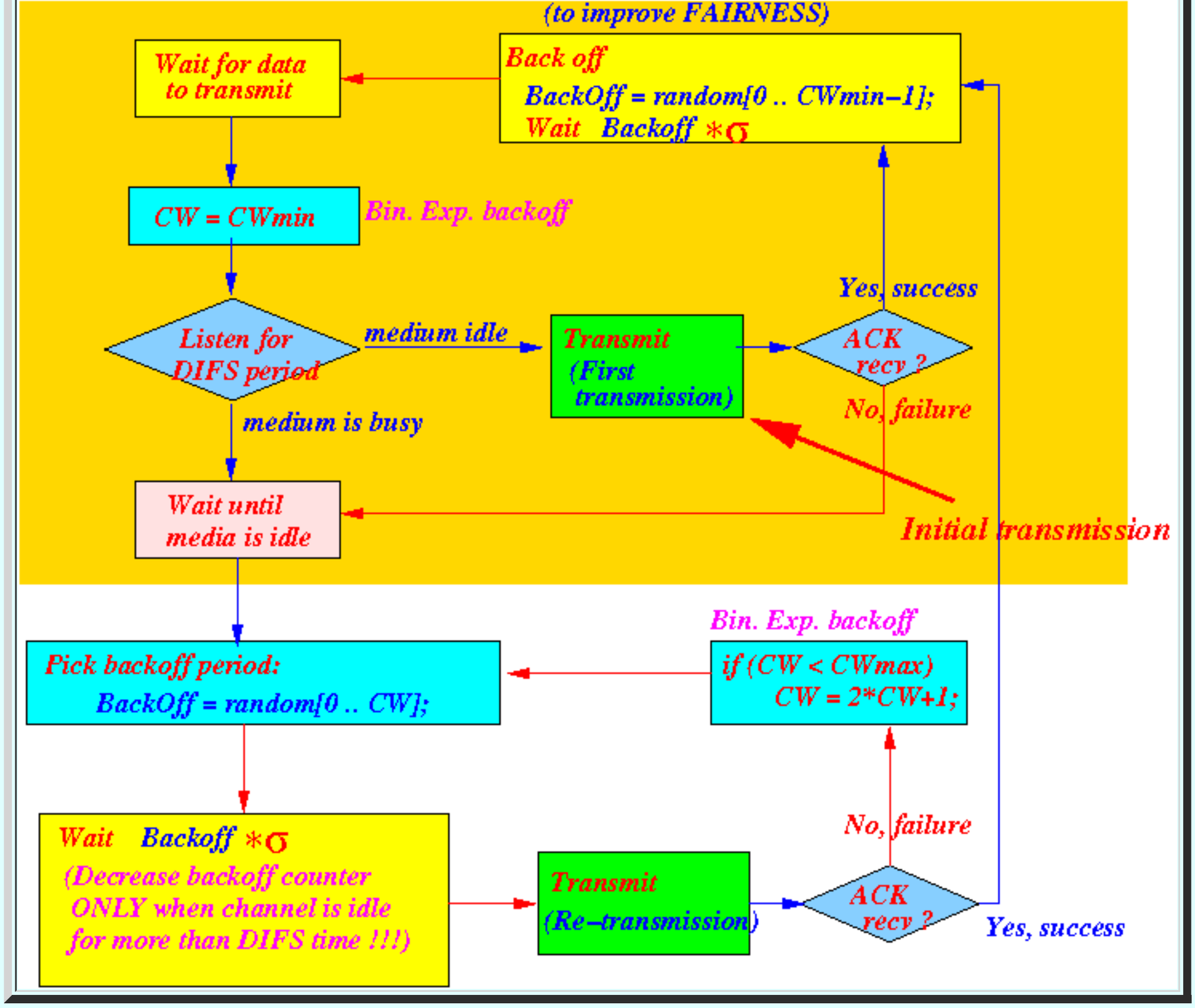
The complete IEEE 802.11 DCF MAC protocol

- **The 802.11 MAC algorithm**
  - The **complete 802.11 MAC algorithm:**

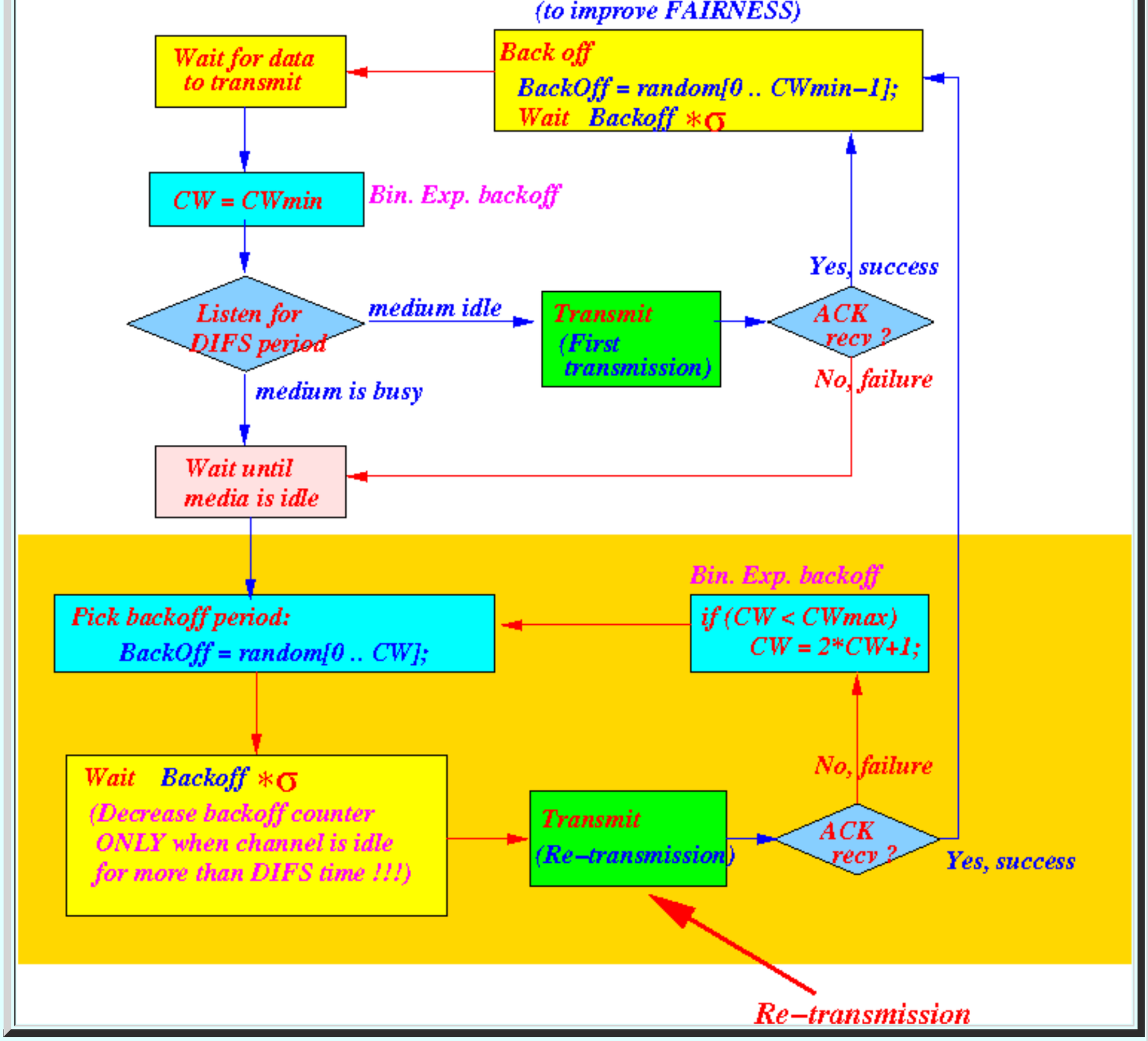


  - I will **explain** the **DCF protocol** in the following **pieces** in the **next few webpages**:

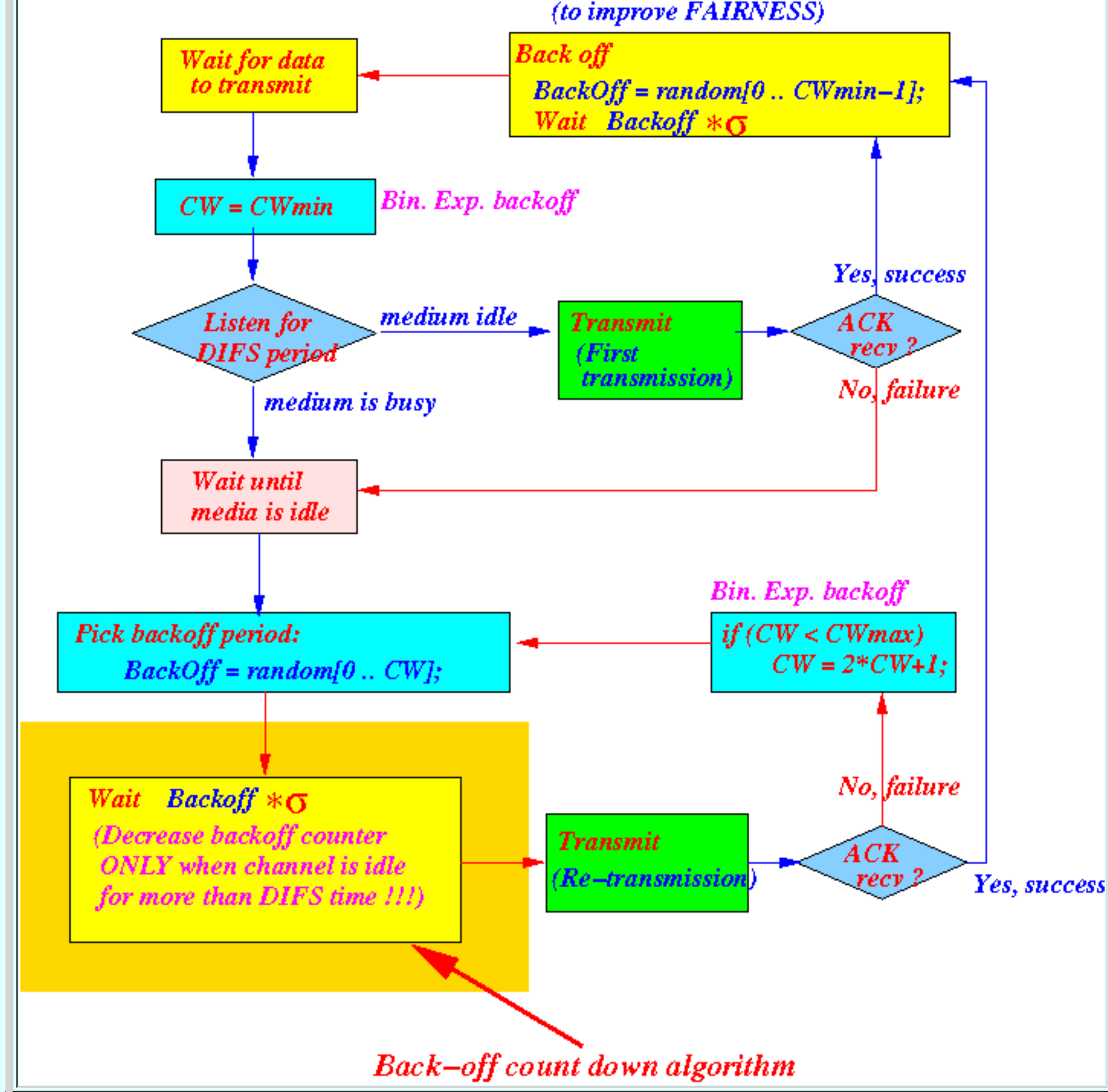
    - The ***initial*** **transmission attempt** (for a ***newly*** **arriving frame**):



- The **re-transmission attempts**

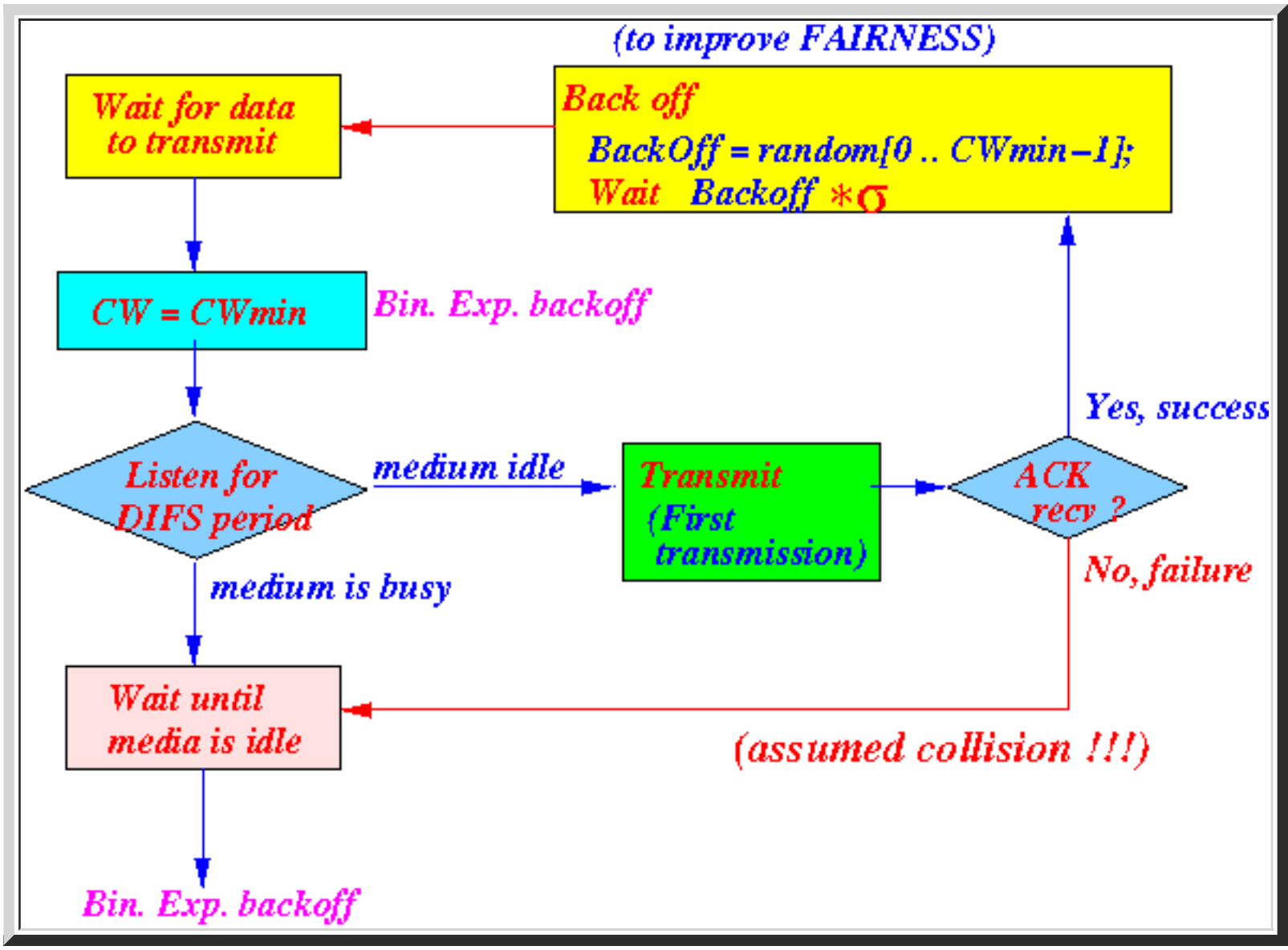


- The **"do not count busy time"** **back off** procedure used in **802.11**


(to improve FAIRNESS)
Wait for data to transmit
Back off
BackOff = random[0 .. CWmin–1];
Wait Backoff *σ
CW = CWmin
Bin. Exp. backoff
Yes, success
Listen for DIFS period
medium idle
Transmit (First transmission)
ACK recv ?
No, failure
medium is busy
Wait until media is idle
Bin. Exp. backoff
Pick backoff period:
BackOff = random[0 .. CW];
if (CW < CWmax)
CW = 2*CW+1;
No, failure
Wait Backoff *σ
(Decrease backoff counter ONLY when channel is idle for more than DIFS time !!!)
Transmit (Re–transmission)
ACK recv ?
Yes, success
Back–off count down algorithm

The first transmission attempt (new frame)

- **The *initial* transmission of a *new* frame**
  - The **algorithm** to transmit a ***new* frame** is as **follows**:



Explanation:

- When a **node** transmits a ***new* frame** (i.e., a **frame** for the ***first* time**:
  - The **node always reset** the the **random number** range (**CW (collision window)** to the **initial value (CWmin)**

- **Before** the **node** can **transmit** the **frame**:
  - The **node** must **"defer" (= listen)** for a **DIFS** seconds (to let **high priority frames** like **ACK frames**) go **first**)

- If the **node** did **not** sense (= **receive**) any **transmissions** for **DIFS** duration:

  - The **node** will **transmit** the **data frame**
  - **Wait** for an **ACK frame** !!!

  If the **node** received an **ACK frame**, it is **done**

  Otherwise (**no ACK** received):

  - The **node waits until** the **current transmission (if any)** has **ended**
  - **Then** the **node** will execute a ***binary exponential*** **back off algorithm** (discussed after this)

- If the **node senses** (= **receives**) a **transmission** during its **DIFS sense period**:

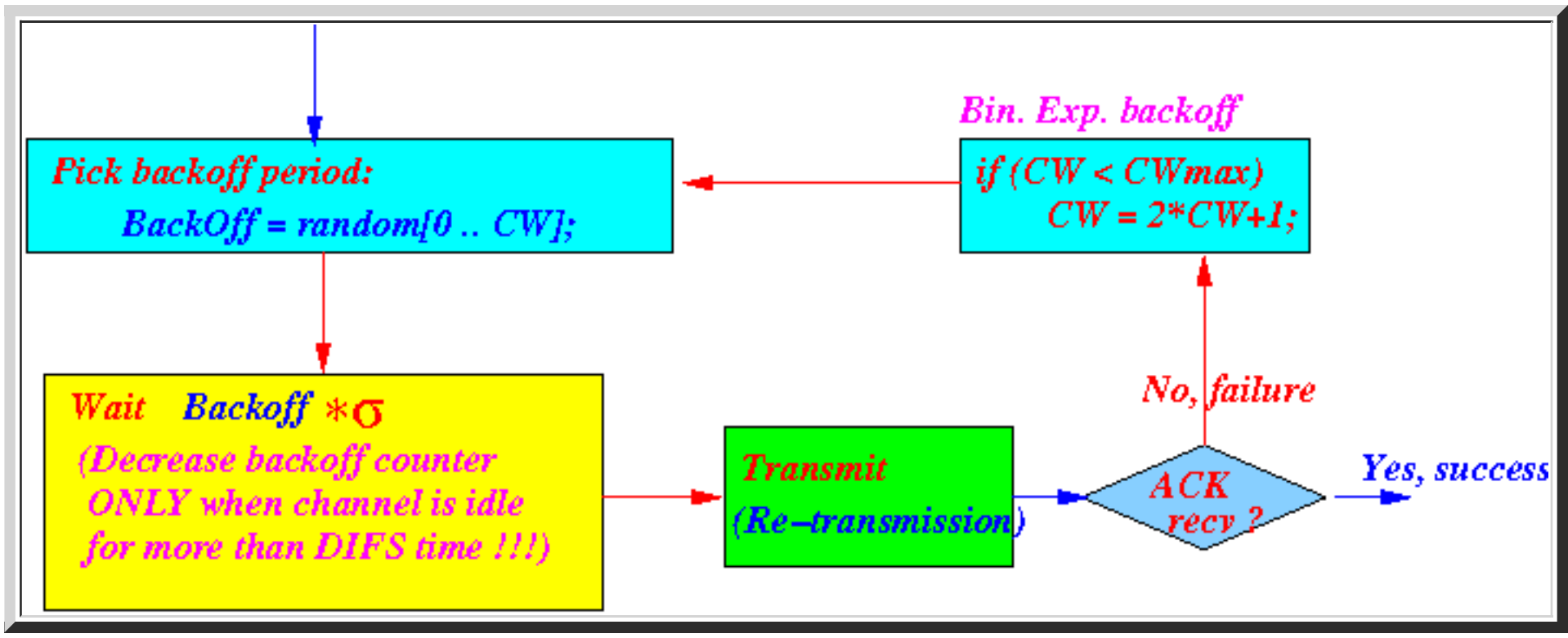
  - The **node waits until** the **current transmission (if any)** has **ended**
  - **Then** the **node** will execute a ***binary exponential*** **back off algorithm** (discussed after this)

  Note:

  - The **node** porforms a **binary exponential back off** to **minimize** the **collision probability**

The subsequent transmission attempts (re-transmissions)

- **Subsequent (retransmission) attempts**
  - The **algorithm** to **re-transmit** a ***old*** **frame** is as **follows**:



Explanation:

- The **protocol** **enter** this **section** when:

1. The **node** has detected a **ongoing transmission** or
2. The **node did *not*** receive an **ACK frame** (***assumed*** **collision** !!!)

Conclusion:

- The **node** now **knows** that there are ***other*** **node(s)** that is trying to **transmit**

- The **node** will first **select** a **random number *x*** from the current **back off range [0 .. cw]** (See: click here)

- The **node** tries to ***auto*-schedule** its **transmission** so the ***new*** **attempt** will ***not*** **collide** with that of ***other*** **node(s)**

- The **node** will **wait (= count down)** for ***x*×σ sec** of ***idle*** **channel time**

- **Idle channel time** = the **time** when there are ***no*** **transmissions** on the **wireless network**

- **σ = slot time (20 μ sec)**

More **info** on the **count down procedure**:

- The **node** keeps **sensing** the **transmission channel** while **counting down**

- If the **node** detect a **transmission** while **counting down**:

  - the ***node will stop the count down until the channel is idle (clear)***

  (because the **channel** is ***not* idle**, the **back off countdown *only*** counts the ***idle* channel time** !!)

- The **count down** will ***only*** be ***re-started***:

  - After the **transmission channel** is **idle** for **DIFS** sec !!!

(This **count down procedure** will be **explained** in ***more* details** in the ***next* webpage**)

- **At the end** of the **back off count down**:

  - The **node** transmits the frame ***immediately*** --- **without** having to wait **IFS period**

Possible outcome of the **trasnmission**:

- If the **node** is **successful** (= received an **ACK frame** in time):

  - **Done** !!!

- If the **node** is ***unsuccessful*** (= ***no* ACK frame**):

  - The **node increases** the **back off range** (unless it reach the maximum range) and

    ---

  - **Select** another **random number**

  - **redo** the **back off** step.

How IEEE 802.11 counts down a back off period

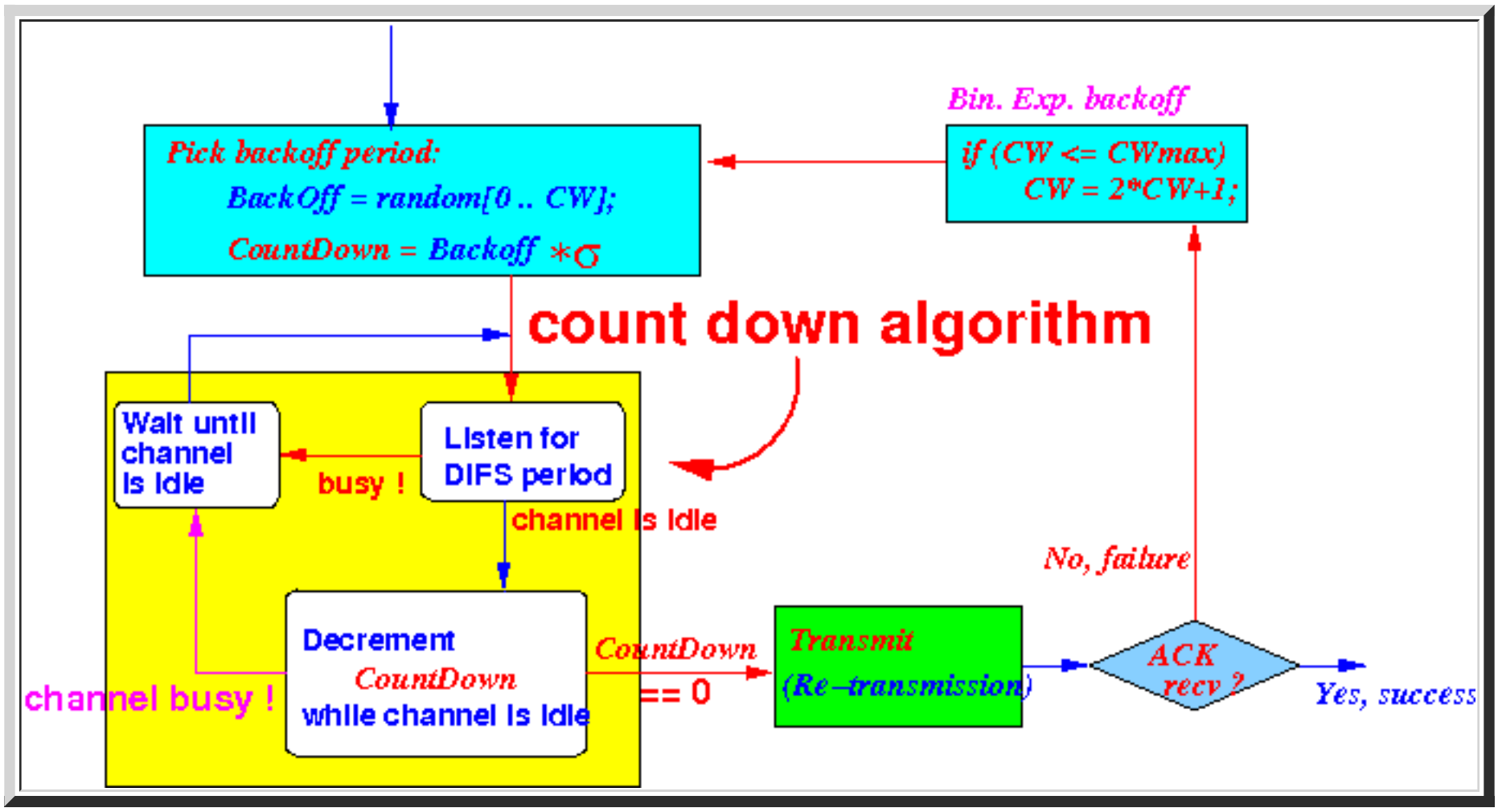
- **The back off count down counter (timer)**
  - **Back off count down timer:**

    - **Back off *count down* timer** = a **couting variable** that **contains** the **time remaining** before a **node** will **transmit** a **frame**

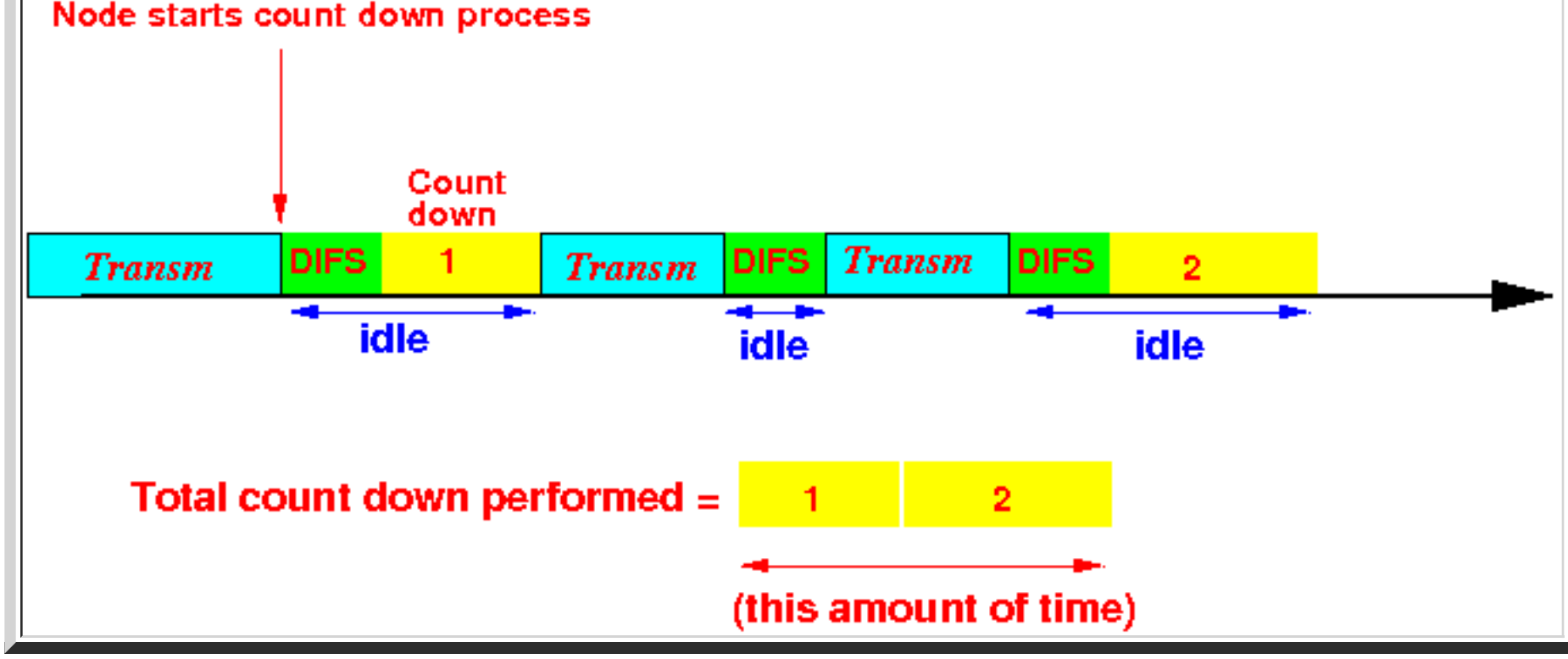
  - **Counting down procedure** for the **back off** timer:

    - The **back off count down timer** is ***initialized*** to a **random value**
    - The **back off count down timer** is then ***decremented*** (more later)
    - When the **back off count down timer** becomes **0 (zero)**, the **node transmits** a **frame**

- **How to update the count down timer**
  - ***How*** the **back off count down** timer is ***decrement***:



The **count down process illustrated**:



Explanation:

- A **node** must ***first*** **listen** for ***DIFS*** duration ***before*** counting down the **count down timer**
  - **If** the **channel** is **idle** for **DIFS** sec:
    - **Start** counting down (= **decrement**) the **count down timer**

  Otherwise:
    - **Wait** for the **channel** to become **idle**

    - **Repeat** the **waiting** for **DIFS** duration

Important fact:

- The **node** will ***only*** **decrement** the **count down timer** when the **channel** is **idle** **after** listening for **DIFS sec** !!!

- **Furthermore**: the **count down** can be ***interrupted*** be a **transmission**:
  - When the **node** hears a **transmission**, it will ***stop*** the **count down**
    - The **counter** is **frozen** (i.e., ***not*** **reset** !!!)

  - The **node** will **wait** for the **transmission channel** to become **clear** again....

  - When the **channel** become **idle**:

- The **node** will **listen (defer)** for **DIFS** ***again***

- **Then:**

  - **If** the **channel** is **idle** for **DIFS** sec:

    - ***Continue*** counting down the **count down timer**

  Otherwise:

    - **Wait** for the **channel** to become **idle**

    - **Repeat listening** for **DIFS** duration

Improving fairness in the binary back off algorithm

- **Improving fairness**
  - **Recall:**
    - The **binary exponential back off algorithm** has the ***channel capture*** **effect**
      - The **bin. exp. back off algorithm** is ***unfair*** to **node** that perform ***frequent*** **retransmissions**

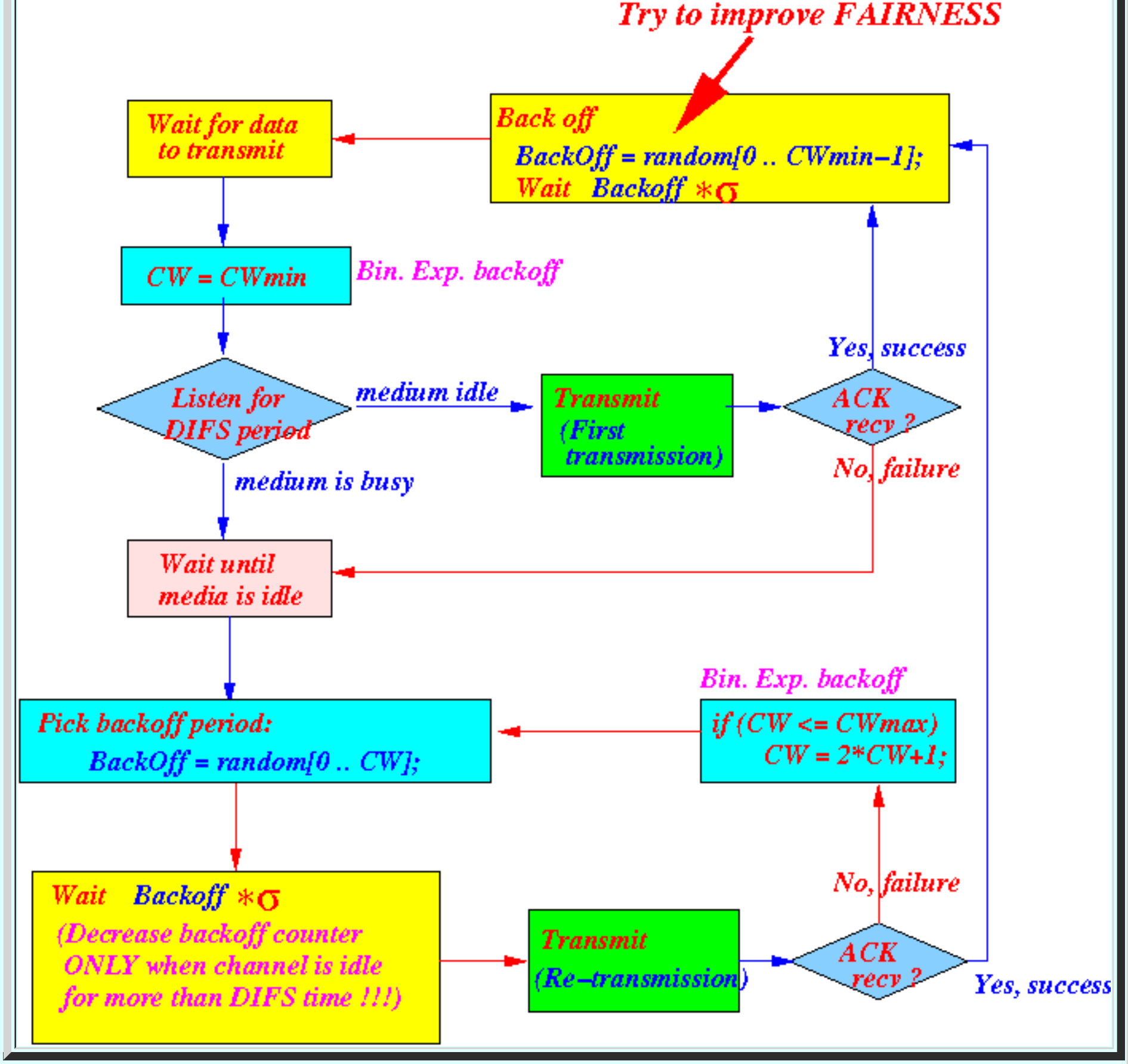
    See: **click here**

  - **Improving fairness:**

- **Fairness fix**:
  - The **802.11 protocol** do **not allow** a **node** to **transmit** frame ***continuously***

- After each **successful** transmission, a **node** will perform a **back off**
  - The **node** "tries" to **allow (let)** nodes that are ***currently*** **waiting (= performing count down)** to **transmit** ***before*** it,

See:



- ***Hopefully***, this will **alleviate** the **channel capture phenomenon** (because other senders would have **larger back off periods** and the current sender will keep winning...)
- This is only a **band-aid remedy** - providing **fairness** in **802.11** is an active **research area**...

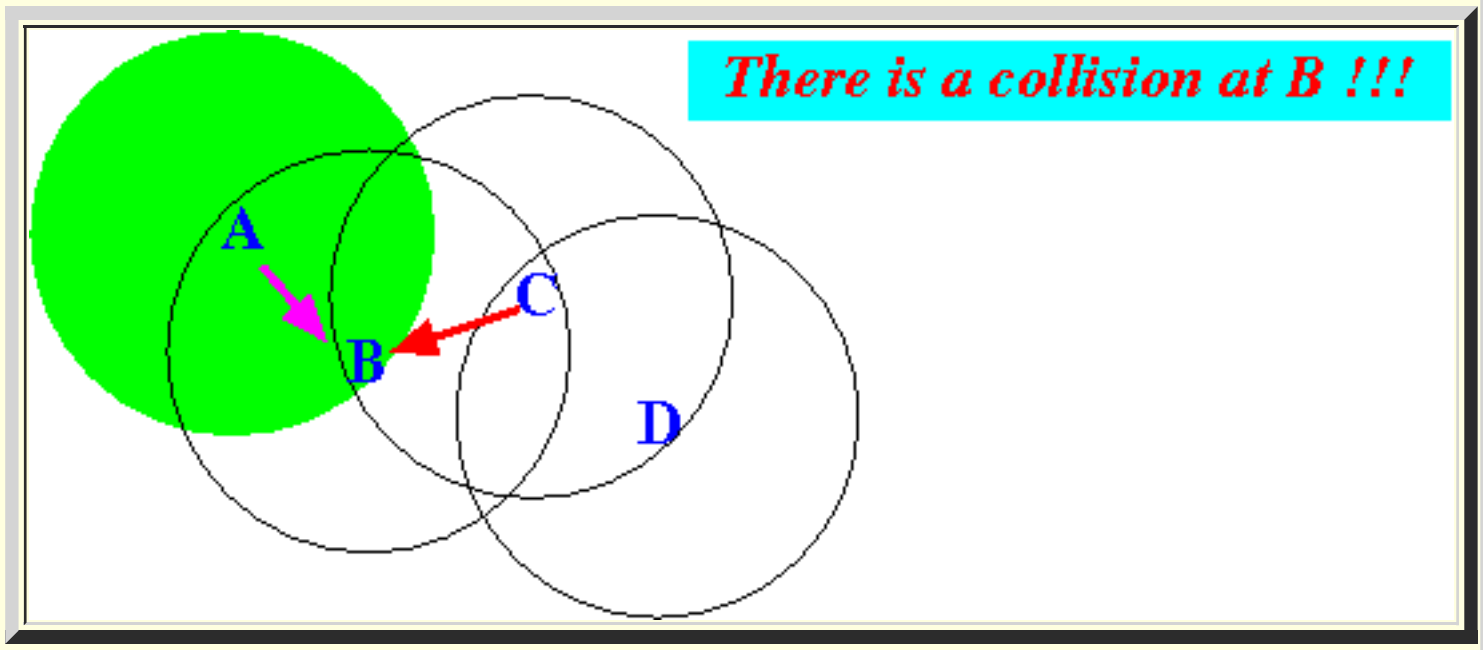
- **Other online materials**
  - 802.11 MAC: click here

The (optional) *Virtual* carrier sensing of DCF (a.k.a. CSMA/CA)

- **Introduction**
  - **Problem** with **DCF** protocol when there is a ***hidden* node**:

    - The **DCF protocol** (using **SIFS** and **DIFS** mechanism) will have **problems** if there is a **hidden node** problem

      - The **hidden node** will **cause** ***collision*** at the **receiver** that the **sender** ***cannot*** **detect**:

      When ***C*** transmits to **B**, the ***hidden* node *A*** can **interfere**

  - ***If*** a **node** detects that **some node** ***frequently*** **fails** to reply with an ***ACK* frame**:

    - The **node** may (**optionally**) use the ***Virtual* Carrier Sense mechanism** to **transmit** ***data* frames**

- **The *Virtual* Carrier Sensing protocol**
  - ***Control* frames** used in the ***Virtual* Carrier Sensing** protocol:

    - ***Request* to send (RTS) frame**
    - ***Clear* to send (CTS) frame**

    (The **RTS** and **CTS** **frames** are ***short* frames** --- like an **ACK frame**)

  - **Usage:**

    - A **node** that wants to **send a data frame** will **transmit**:

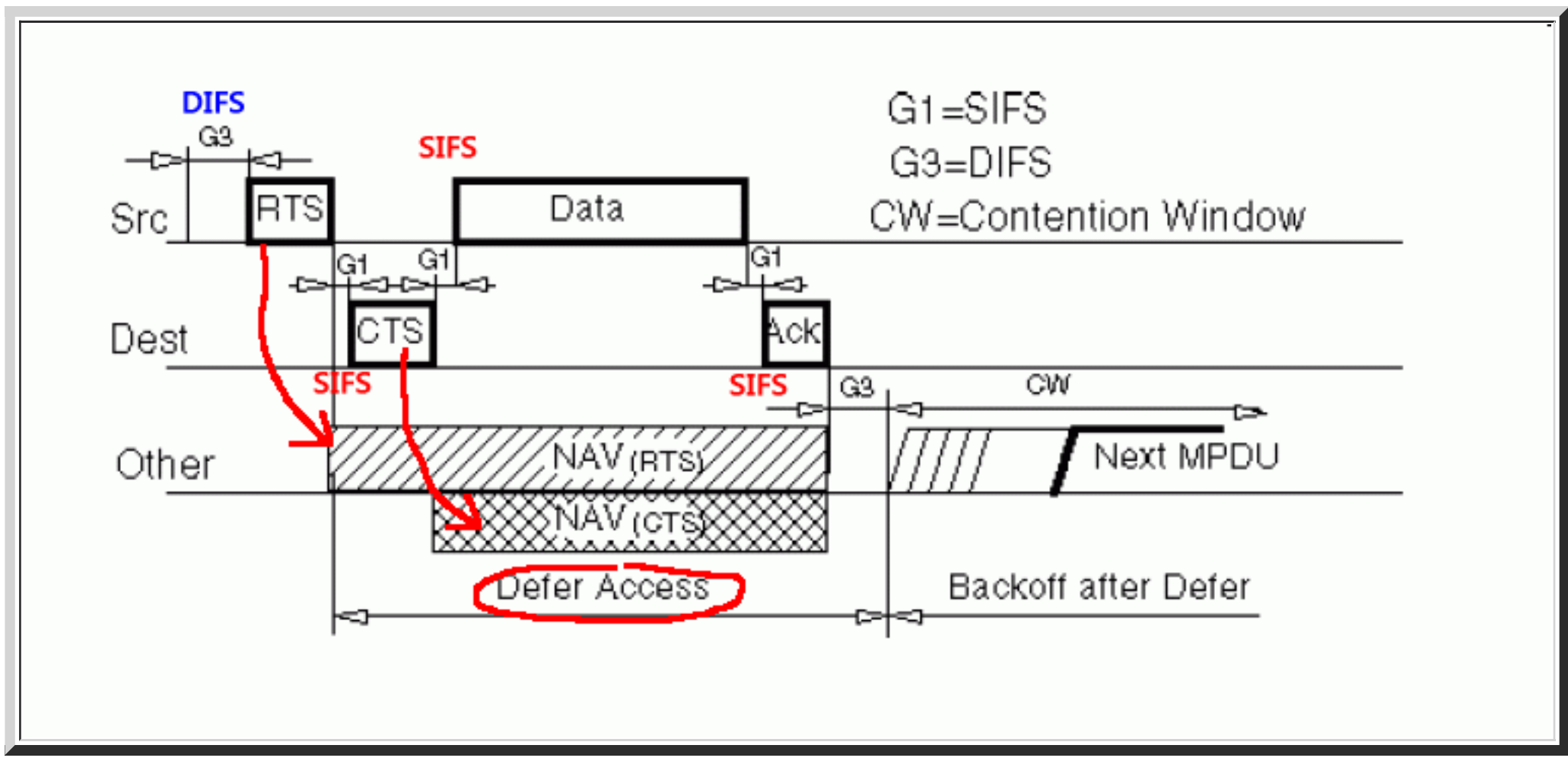
      ```
      RTS  frame  to  its receiver
      ```

    - The **receiver** will **send back** a **(high priority) CTS frame**

- The **sender** will then **send** a **(high priority) data frame**

(There are **other** ***side effect*** that will be **explained soon**)

Graphically:



- **The Request to send control frame**
  - The **IFS** for the **RTS frame**:

    DIFS

    Note:

    - The **node** ***actually*** wants to **transmit** a ***data*** **frame**

    - **Instead** of **sending** a ***data*** **frame** (which **can** be **lost** due to **collision** with a ***hidden*** **node**):

      - The **node** transmits the **RTS frame** to **solve** the ***possible*** **hidden node problem** !!!

  - The ***Request To Send (RTS)*** **frame**:

    Request to Send frame format:

SrcID	DestID	duration

    Explanation:

    - The ***Request To Send*** **(RTS) frame** contains:

- **Source ID**
- **Destination ID**
- a **(time) duration value**

- **Effect:**

  - **Desitination Node**:

    - The **destination node** must **respond** with a **(high priority)** ***Clear to Send*** **frame** (see below)

  - The ***other*** **nodes** that received a **RTS frame**:

    - The **other node** must ***refrain*** **from transmitting** for `duration` **sec** (given in **RTS frame**)

      (This is **implemented** using the **Network Allocation Vector** - see below)

- **The Clear to send control frame**

  - The **IFS** for the **RTS frame**:

    ```
    SIFS  !!!    (High priority, like an ACK frame)
    ```

    Note:

    - The **CTS frame** will **follow** the **RTS frame** ***immediately***

      - Because `SIFS` is the ***shortest*** **IFS** time

  - **Reason** to use **CTS**:

    - The **destination node** uses the **CTS** **frame** to ***silent*** **all** ***hidden*** **nodes** in its **neighborhood** !!!

  - The ***Clear To Send (CTS)*** **frame**:

    Explanation:

    - The ***Clear To Send*** **(CTS) frame** ***also*** contains:

- **Source ID**
- **Destination ID**
- a **(time) duration value**

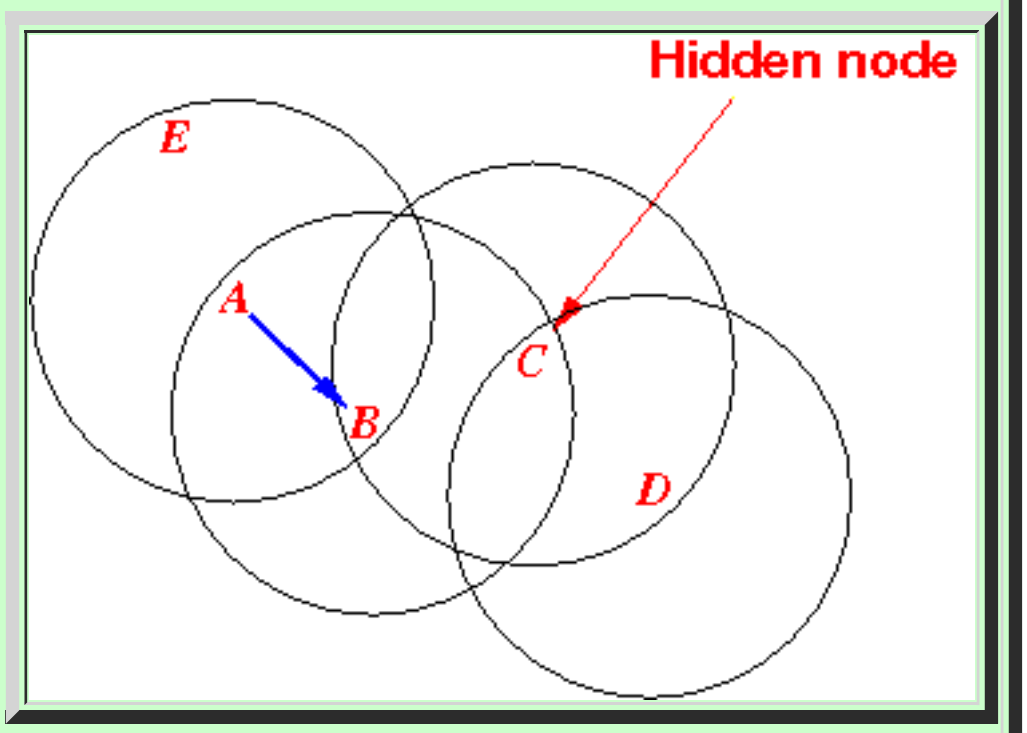
- **Effect:**

  - The ***destination*** **node** that receives the **CTS frame** must **transmitting** for the **duration time** in **CTS frame**

  - ***All*** **node** that receives the **CTS frame** must ***refrain*** **from transmitting** for the **duration time** in **CTS frame**

- **How to solve the hidden node problem using RTS/CTS frames**
  - **Consider** the **following scenario**:

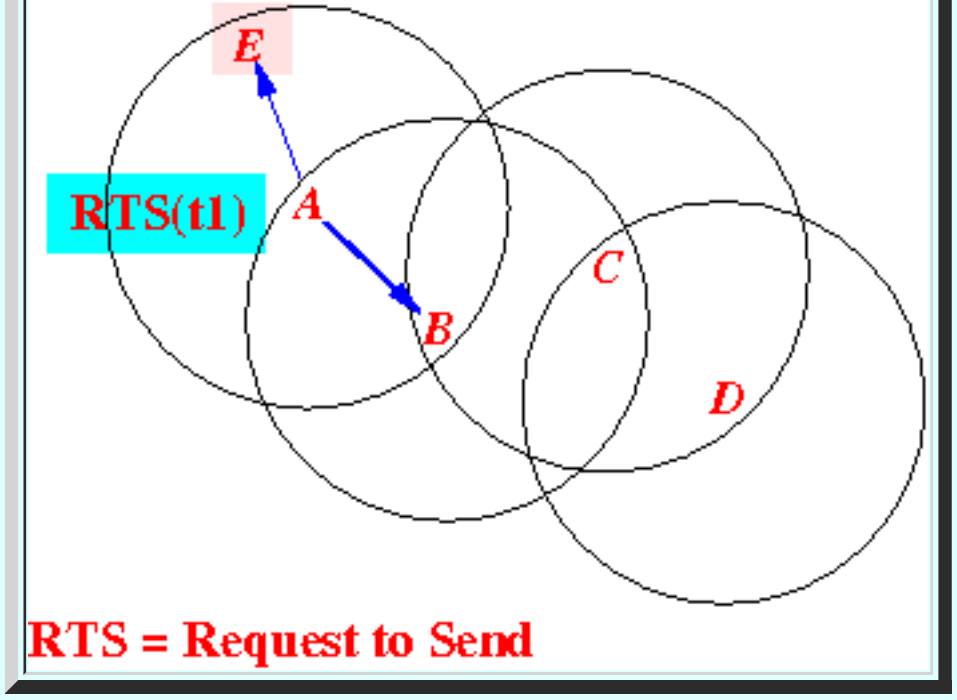
    - Suppose **node *A*** wants to send a ***data*** **frame** to **node *B***



Node *C* is a ***hidden*** **node** that can cause a **collision** !!!

  - **Solving** the ***hidden*** **node** problem:

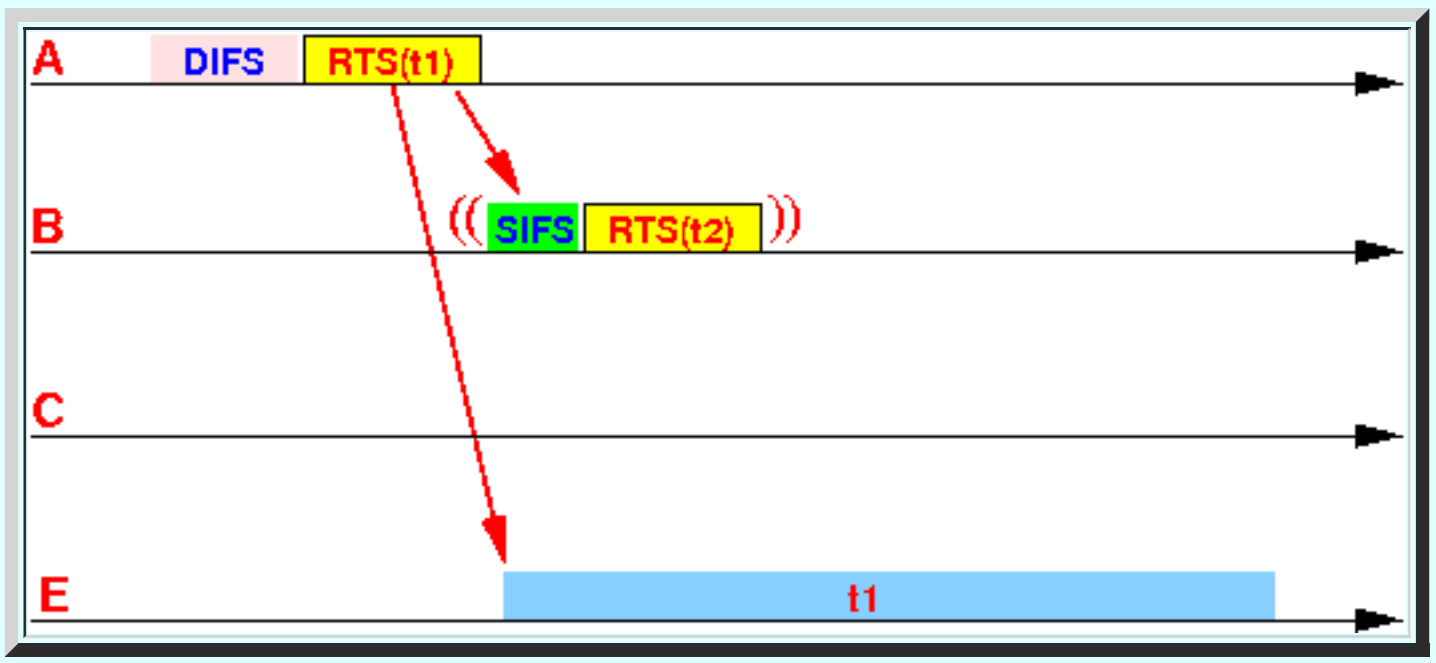
    - **Node *A* *first*** send a **RTS(*t*) frame** to **node *B***:



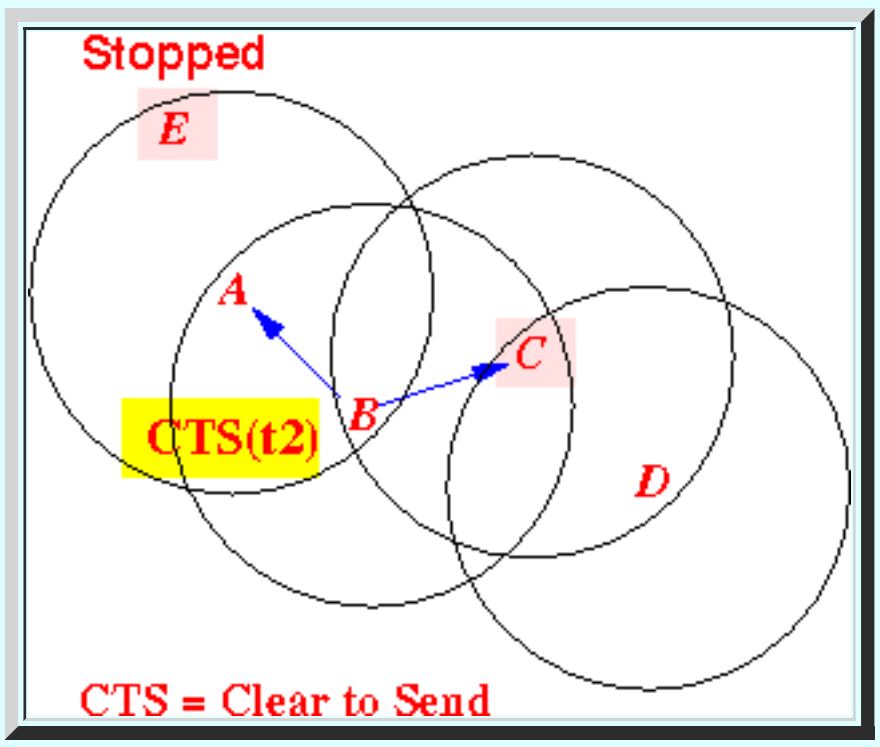
Result:

- **Node *B*** will **respond** with a **CTS frame**
- **Node *E*** will **refrain** from **transmission** for ***t1*** **seconds**
  - **Node *E*** will ***not*** **collide** with the **subsequent CTS frame** !!!

Graphically:



- **Node *B* transmits** a **CTS(*t*) frame back** to **node *A***:

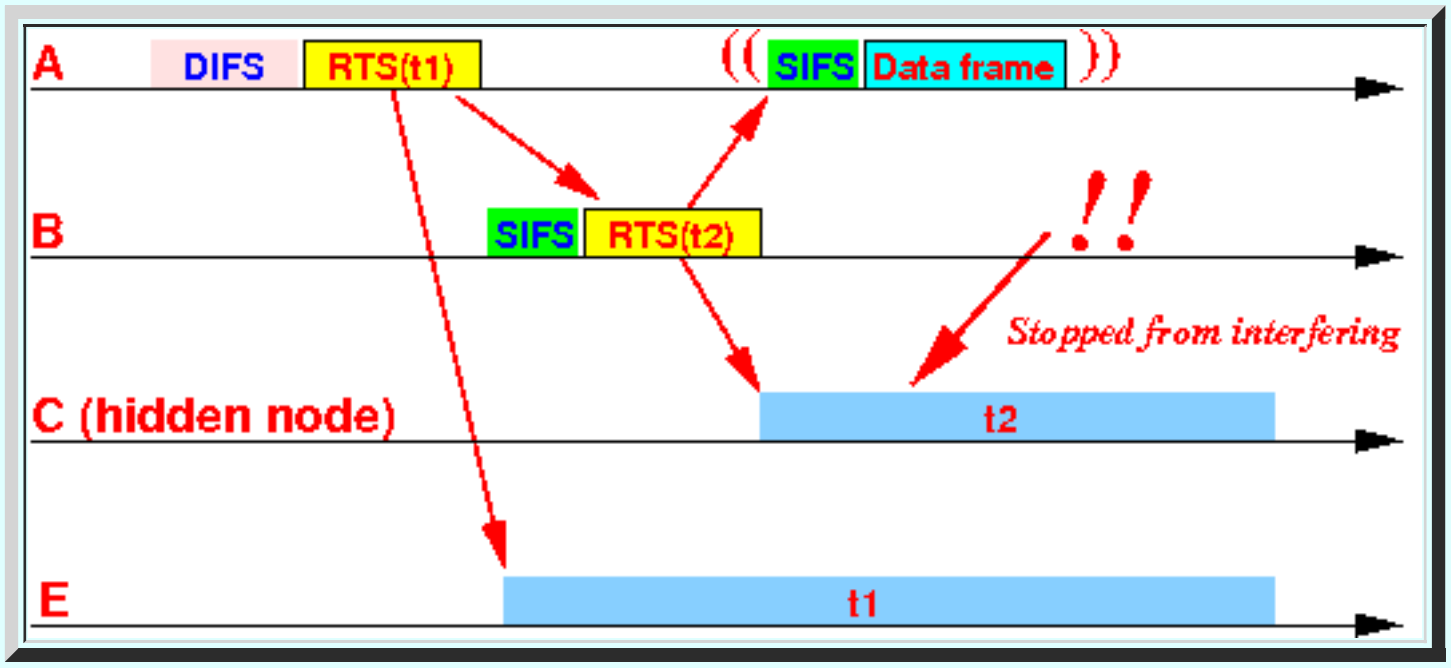


Result:

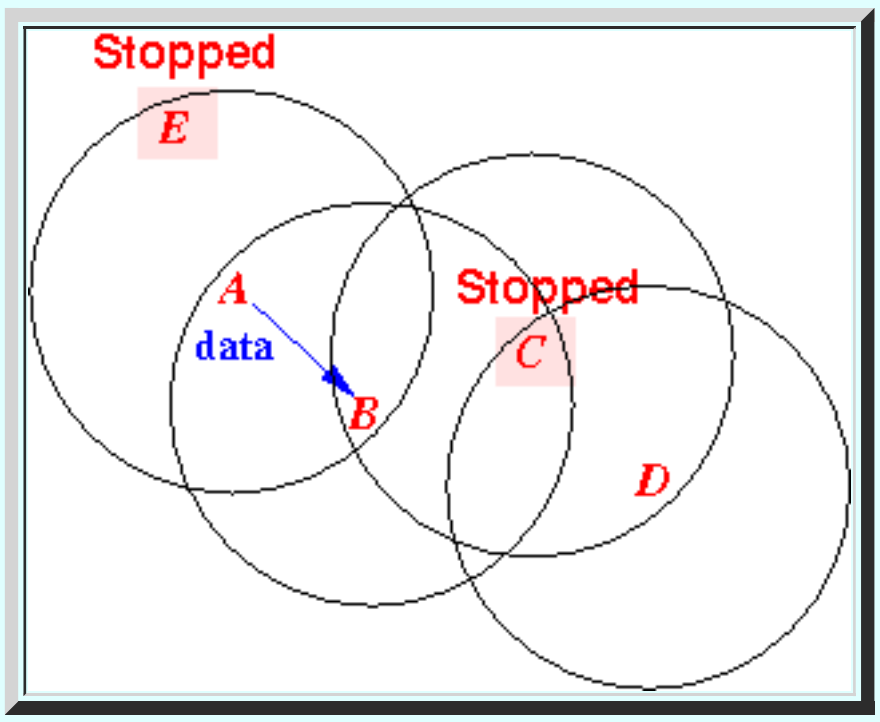
- **Node *A*** will **now *transmit*** a ***data* frame** (after sensing for ***SIFT* duration**)
- **Node *C*** will **refrain** from **transmission** for ***t2* seconds**

- The ***hidden* node *C*** will ***not* interfere** !!!

Graphically:



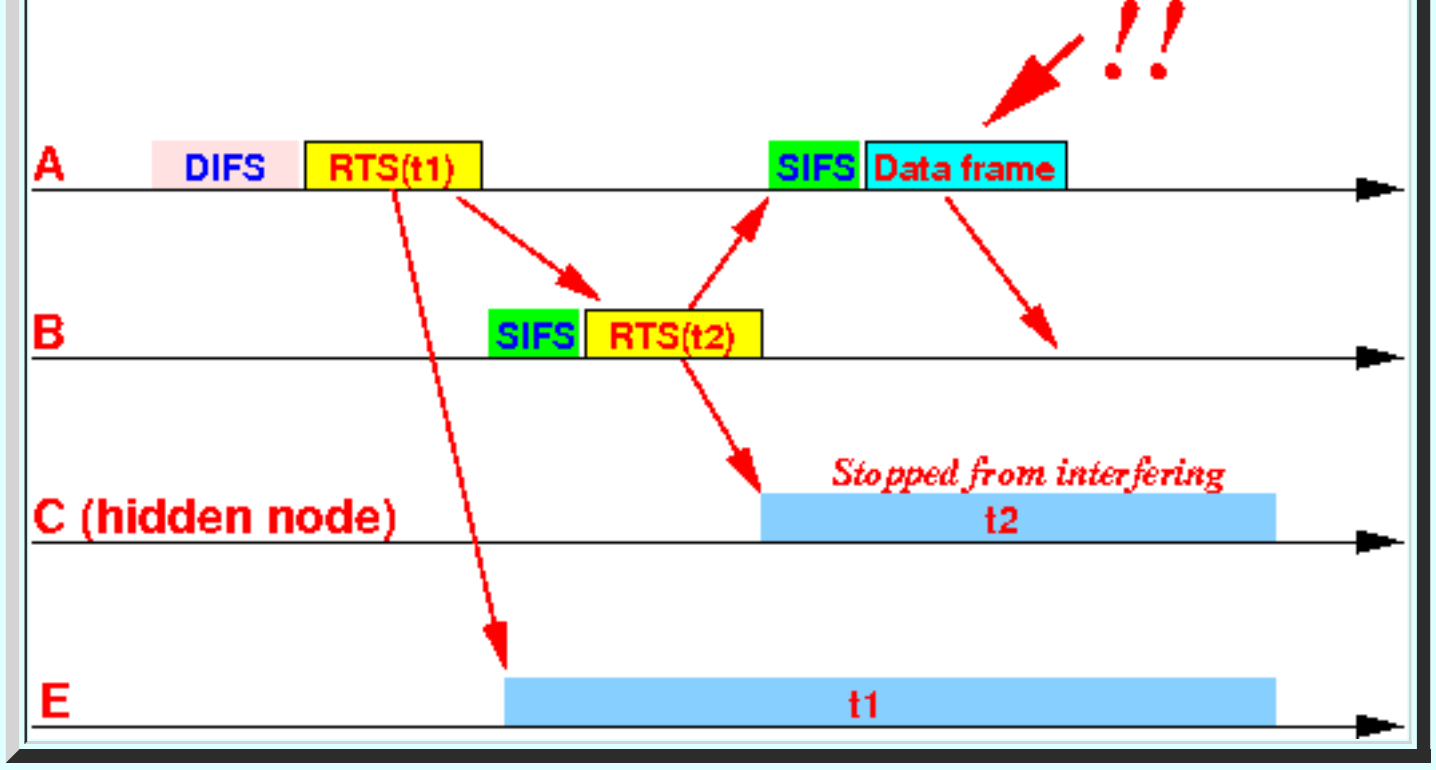
- **Node *A* now** sends its ***data* frame** to **node *B***:



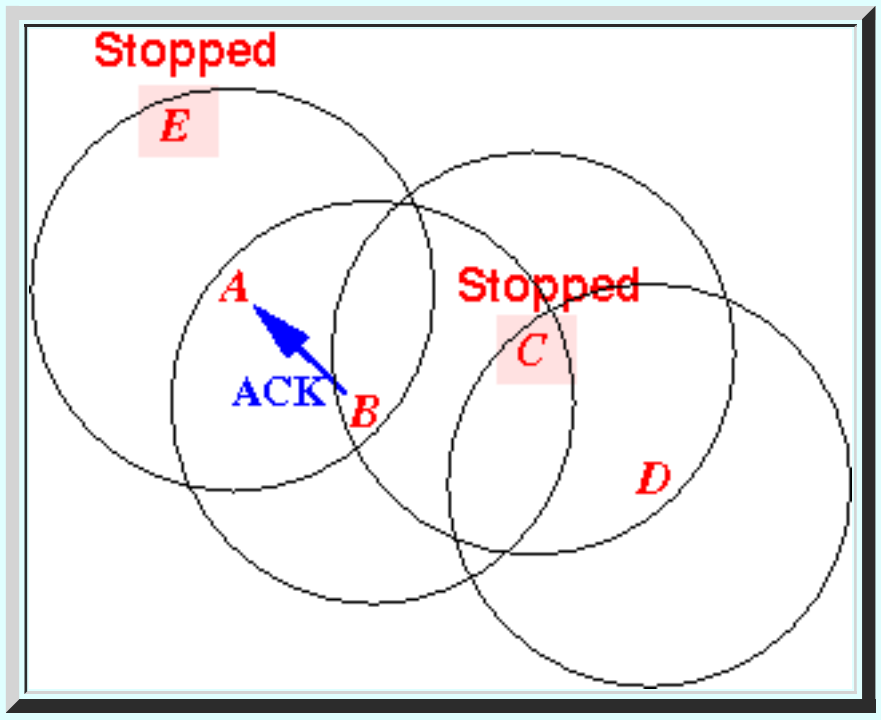
Notice that:

- The ***hidden* node (*C*)** will ***not* interfere** with **node *A*'s transmission** !!!

Graphically:

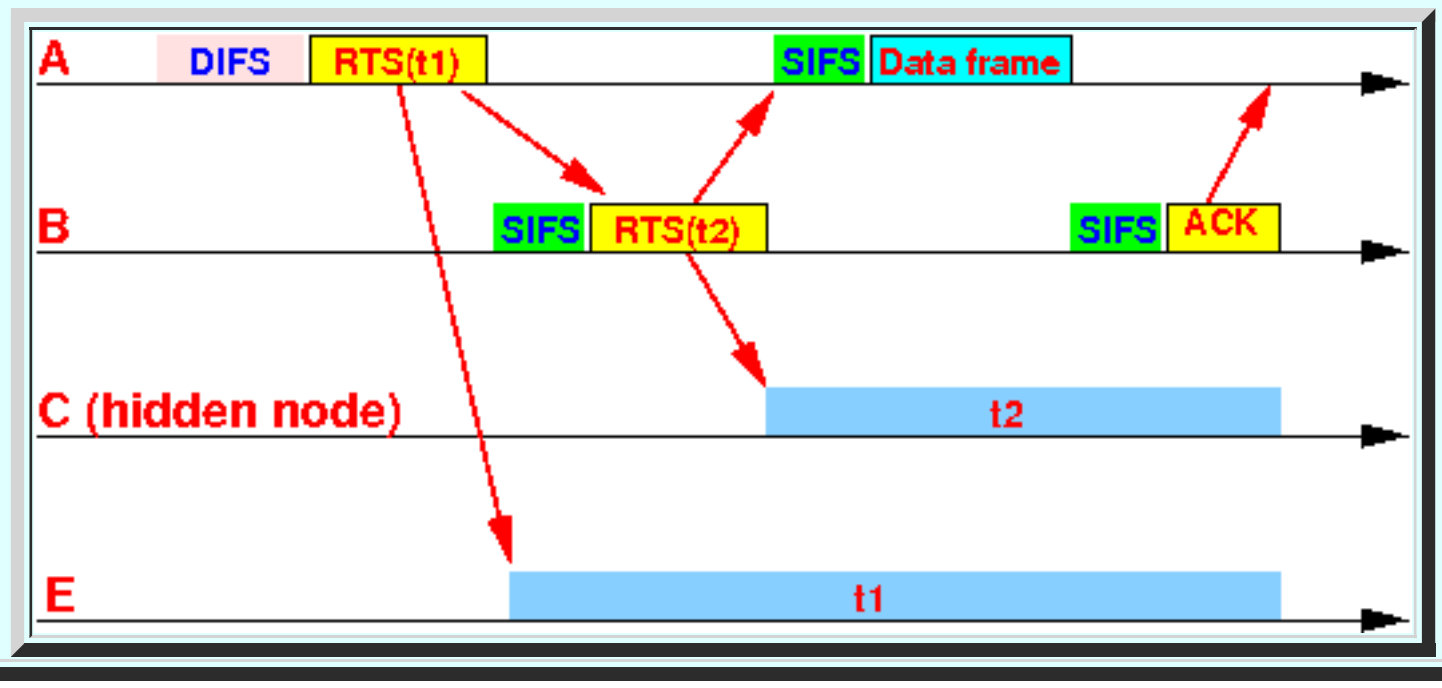


- **Finaly**: **Node *B*** sends the ***ACK* frame** **back** to **node *A***:



Without any node **interfering** !!!

Graphically:



- ***How* to implemenet the *Virtual* Carrier Sensing protocol (using the Network Allocation Vector)**
  - **Network Allocation Vector:**

    - **Network Allocation Vector** = a **data structure (= variable)** in the **802.11 protocol** that **records**

reservation **periods**

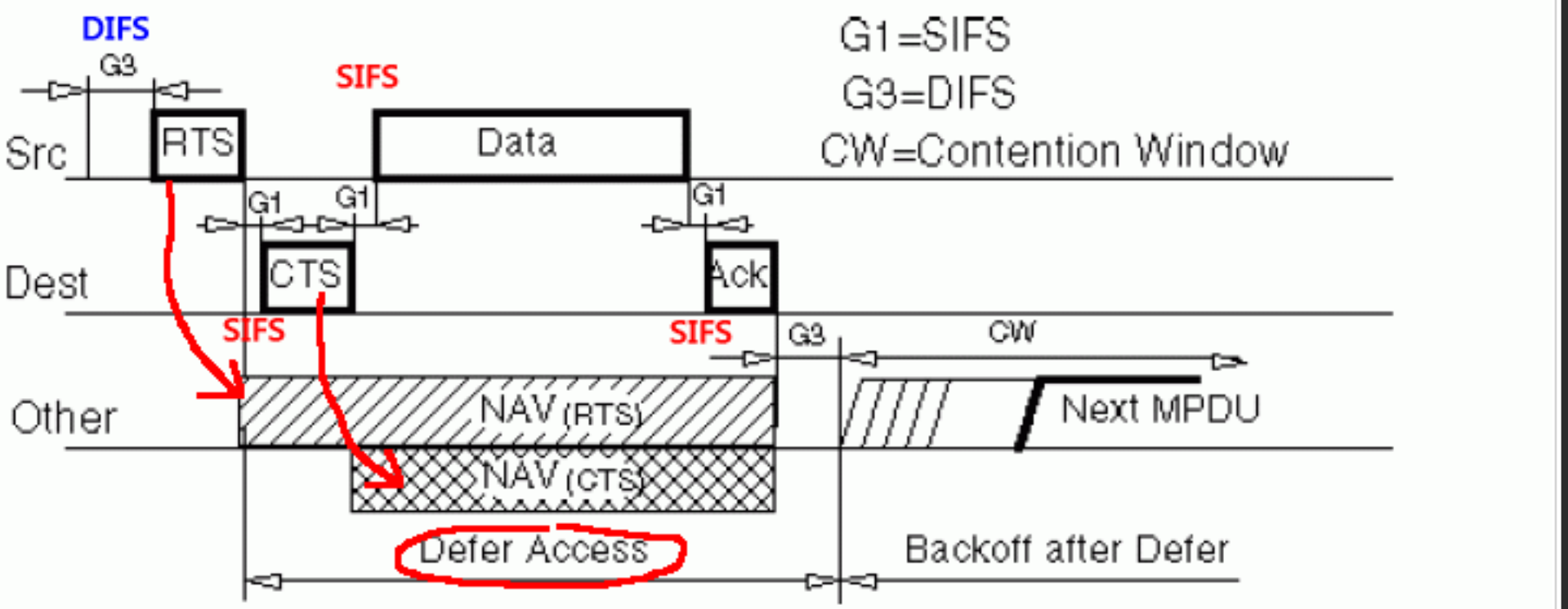
- **Reserved period** = a **period (time interval)** where a **node** ***cannot*** transmit ***any*** **frame** (***except ACK*** **frames**)

- **How to** implement ***Virtual*** **Carrier Sensing** using the **NAV (variable)**:

  - When a **node** receives a **RTS(*t*)** or **CTS(*t*)** frame, a **node**:

    - **Mark off** the **reservation period** in its **NAV**.

- **Example**:



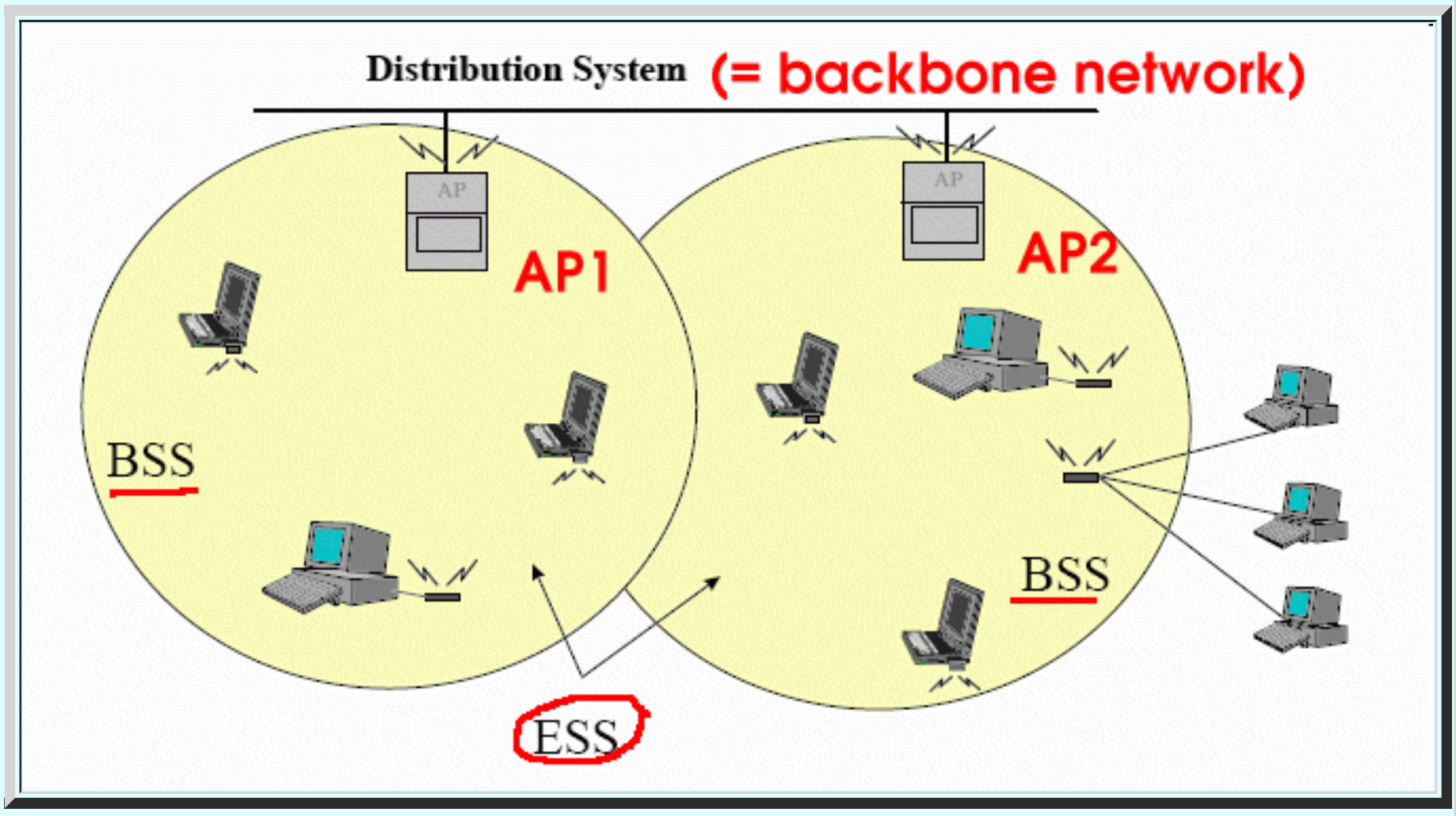
How to set the values of **t1** in **RTS** and **t2** in **CTS**:

```
t1 = (SIFS + CTS) + (SIFS + DATA) + (SIFS + ACK)    // Sender knows DATA length !!!
t2 = t1 - (SIFS + CTS)                              // Receiver uses t1 to compute t2
```

Intro: roaming

- **Recall: *Extended* Service Set**
  - **Extended Service Set**:

    - A **wireless (802.11)** network with ***multiple* Access Points**:



- **Mobile computing**
  - **Mobile nodes:**

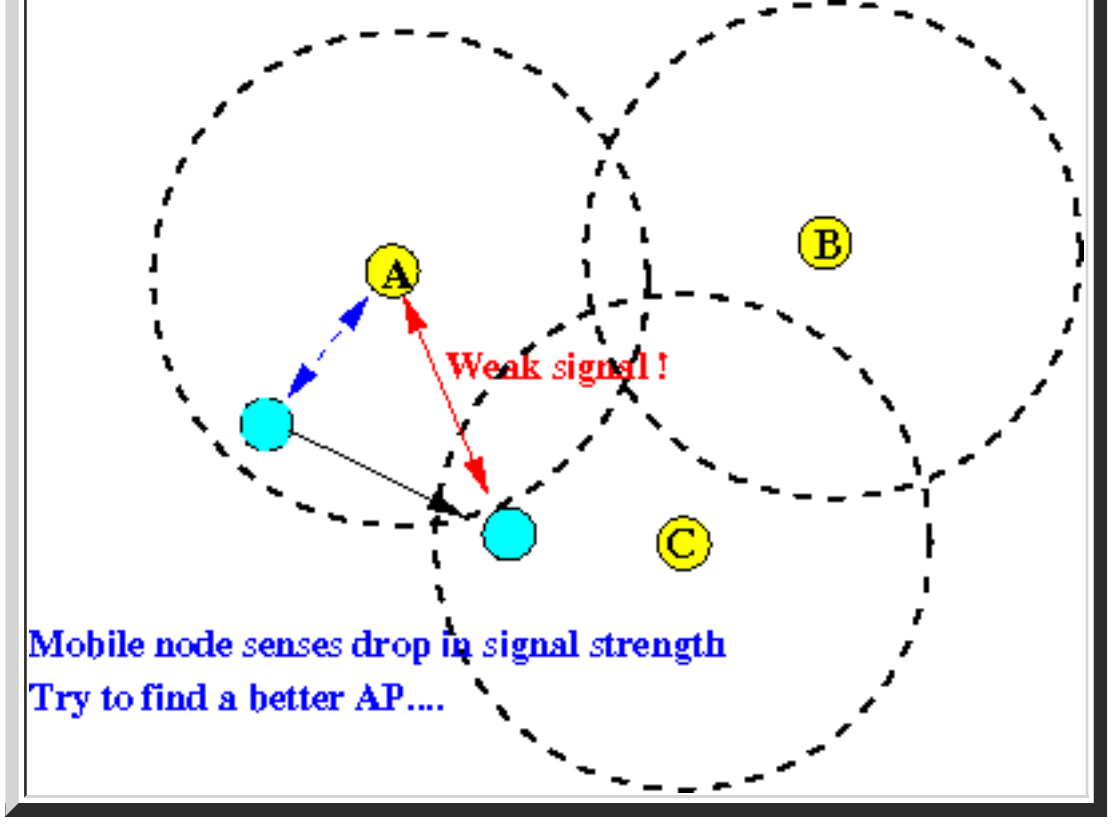
    - ***Wireless* nodes** is **mobile**

    - A **node** can **move** across ***different* Basic Service Sets** (served by ***different* APs**)

- **Roaming**
  - **Roaming:**

    - **Roaming** = the ***ability*** to ***switch*** from one ***one* AP** to ***another* AP *without losing* connectivity**

    When mobile host **roam** (travel), it will cross the **coverage area** of **different APs**:



- **Implementation** of **Roaming**:

  - A **mobile host** must **disconnects** from its ***current* AP** and
  - **Then connect** to ***another* AP**

Connectng to and switching Access Points

- **Connecting to an AP**
  - **Communication** with an ***Access Point***:

    - A **user node (device)** can ***only*** communicate with an **Access Point** **after**:
      - **Authenication** and
      - **Association**

    (E.g.: your **laptop** will ***not*** work on the **MathCS wireless network** without the **correct pass code**)

- **Authentication**
  - **Authentication:**

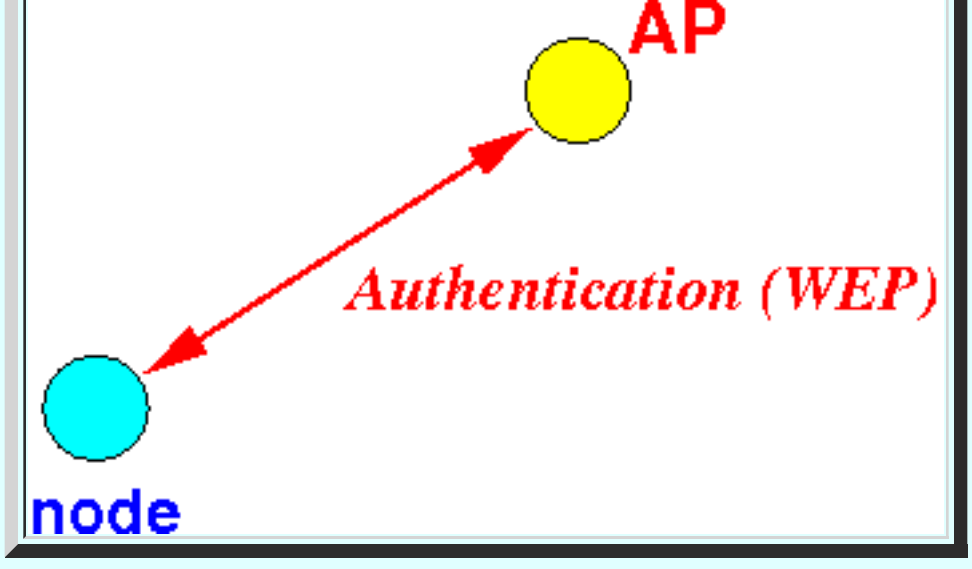
    - **802.11 authentication** is the **first step** in **network attachment**.

    - **802.11 authentication** requires a **mobile device (station)** to **establish its identity** with an **Access Point (AP)**

  - **Authentication Procedure**:

    - A **node** transmits its
      - **MAC address** (= identity)
      - **Pass phrase**

    to an **Access Point (AP)** (or broadband wireless router):



- The **node** can use one of the **secure protocols** to **send** its **authentication information**:

  - **Wired Equivanlent Privacy (WEP)**,
  - **Wi-Fi Protected Access (WPA)** or
  - **Wi-Fi Protected Access 2 (WPA2)**

(**Encryption methods** is **outside** the **scope** of this **course**)....

- **Association**
  - **Association:**

    - **Once authentication** is **complete**:

      - **Mobile devices** can **associate (register)** with an **AP/router** to gain **full access** to the network.

    - **Association** allows the **AP/router** to **record** each **mobile device** so that **frames** may be **properly delivered**.

      - **Association *only* occurs** on **wireless *infrastructure* networks**
      - **Association** does **not** happen in ***ad hoc* (peer-peer) mode**.

    - A **station** can ***only* associate** with ***one* AP/router at a time**.

Detailed explanation:

- The **Access Point** is usually connected to the ***Internet***

- **If** a **wireless node** wants to **send data** to **nodes on the Internet**:

  - the **node** must ***first*** **associate** with an **Access Point**.

- An **Access Point** will **reserve *buffers*** (= **precious resource**) ***only*** for ***associated*** **nodes** !!!

- **Association Prcedure**:

  - The **node** sends an ***Asscoiation*** **request** to the **AP**

  - When an **AP/router grants association**, the **AP** will responds with

    - The **status code** of **0 (successful)**
    - The **Association ID (AID)**.

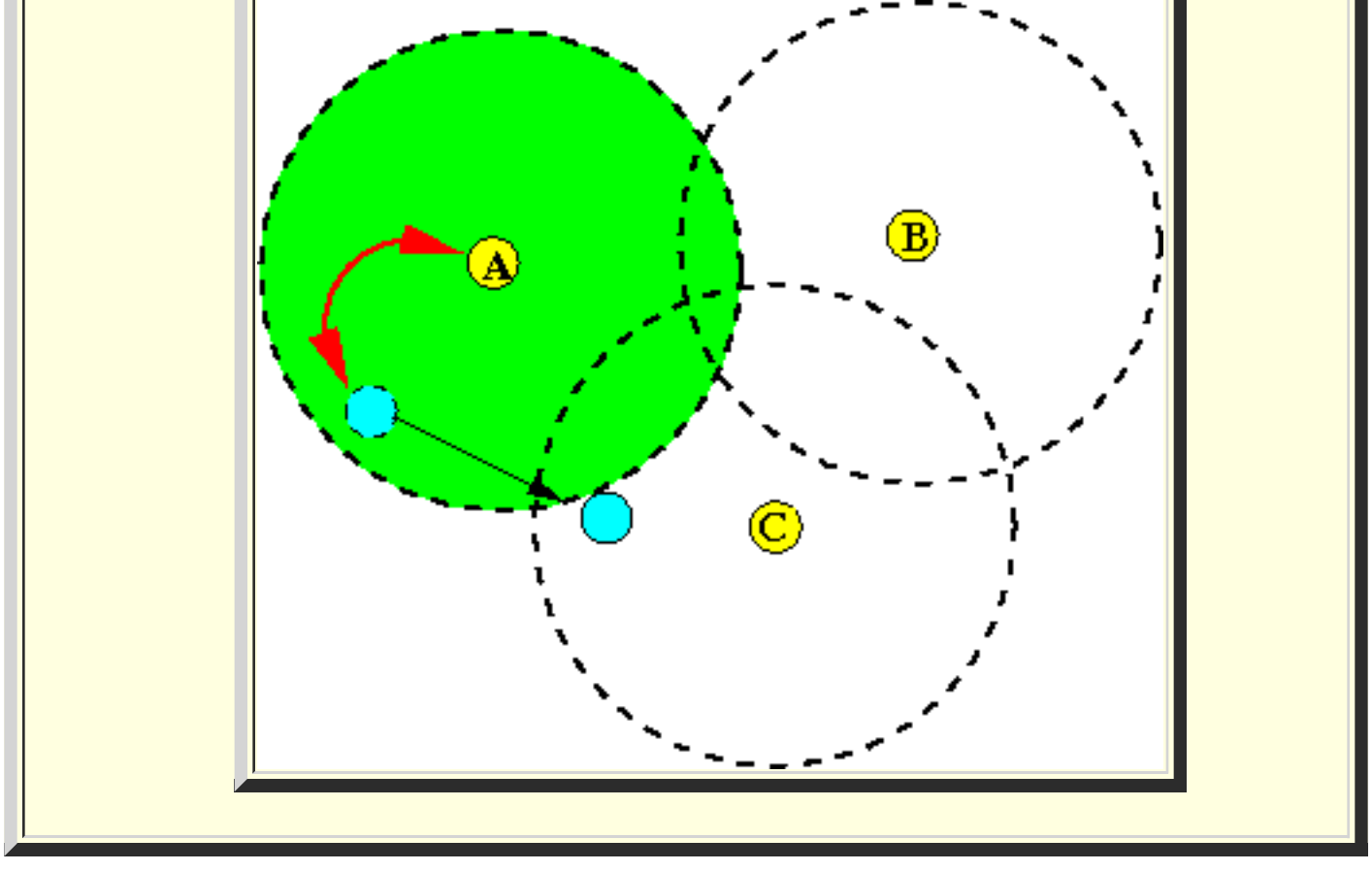
    The **Association ID** is used to **identify the station** for **delivery** of ***buffered*** **frames**

    - **Data frame** from the **Internet** must be ***buffered*** at the **Access Point** before the **Access Point** can ***deliver*** the **frames** !!!

- ***Switching*** **Access Points**

  - **Mobile nodes** and **signal quality**:

    - **Mobile nodes** can ***move*** outside the **coverage area** of an **Access Point**:



Result:

- The **mobile node** must ***switch*** to another **Access Point** that can provide **connection** with a ***stronger*** **signal strength**

- ***Switching*** **Access Points**:

  - The **802.11 protocol** allows **moible nodes** to ***switch*** **Access Points** using a
    - **Hand off protocol**

- There are **2 hand off protocols** defined in **IEEE 802.11**:
  - **Active Scanning**
  - **Passive Scanning**

Active Scanning

- **Active Scanning**
  - **Active scanning**:

    - In ***active*** **scanning**, the **mobile node** takes the ***inititive*** in ***finding*** a **better AP**

    - When the ***signal*** **strength** to the ***associated*** **AP drops** ***below*** **a threshold:**

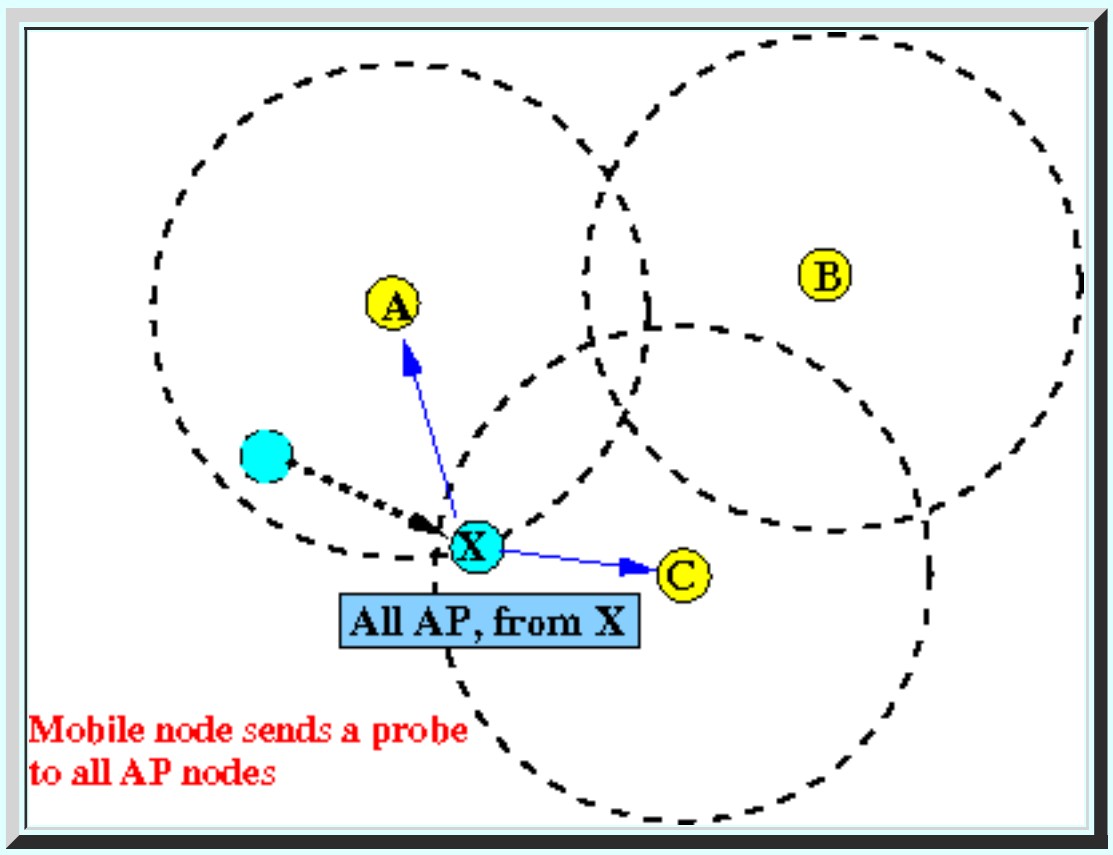
      - The **moible node** will start the ***active*** **scanning procedure**

- **The active scanning procedure**
  - The **Active Scanning** procedure:

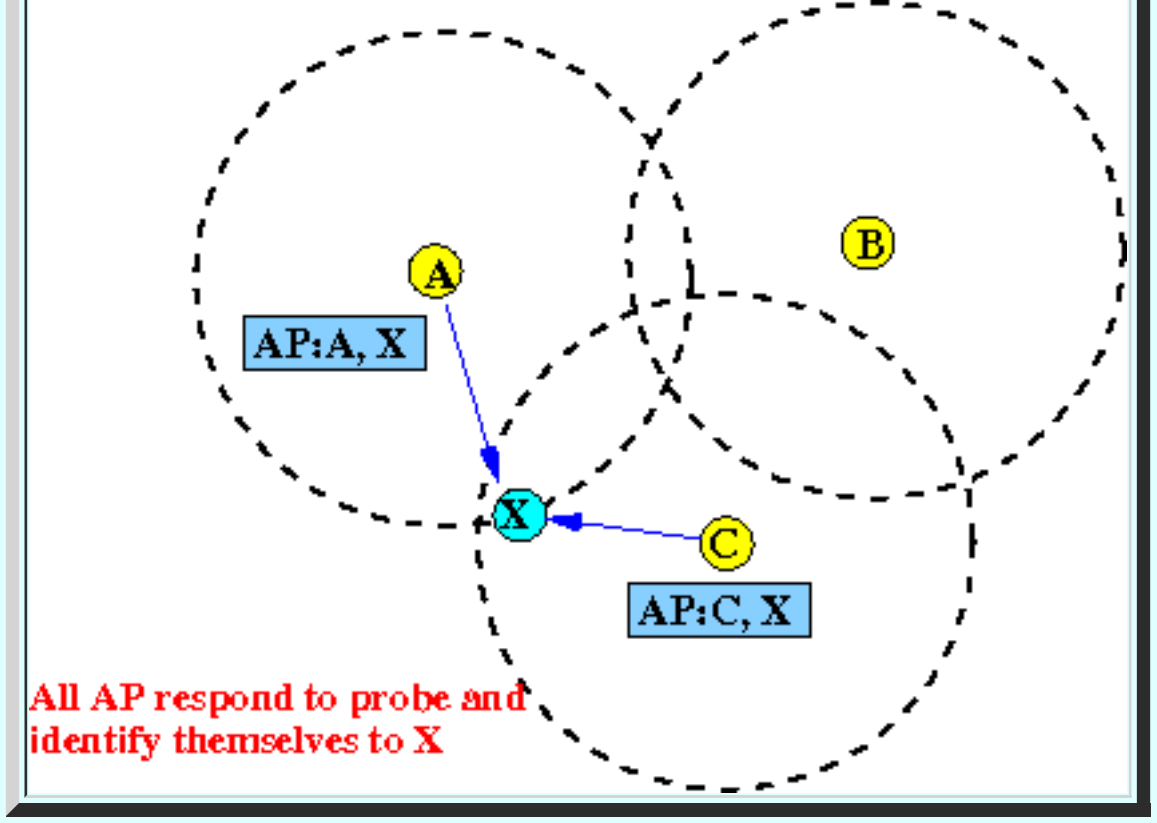
    - The **mobile node** ***broadcasts*** a **probe message** to ***all*** **access points**:

      - The **destination address** of the **probe frame** contains the special ***"All AP" MAC address***

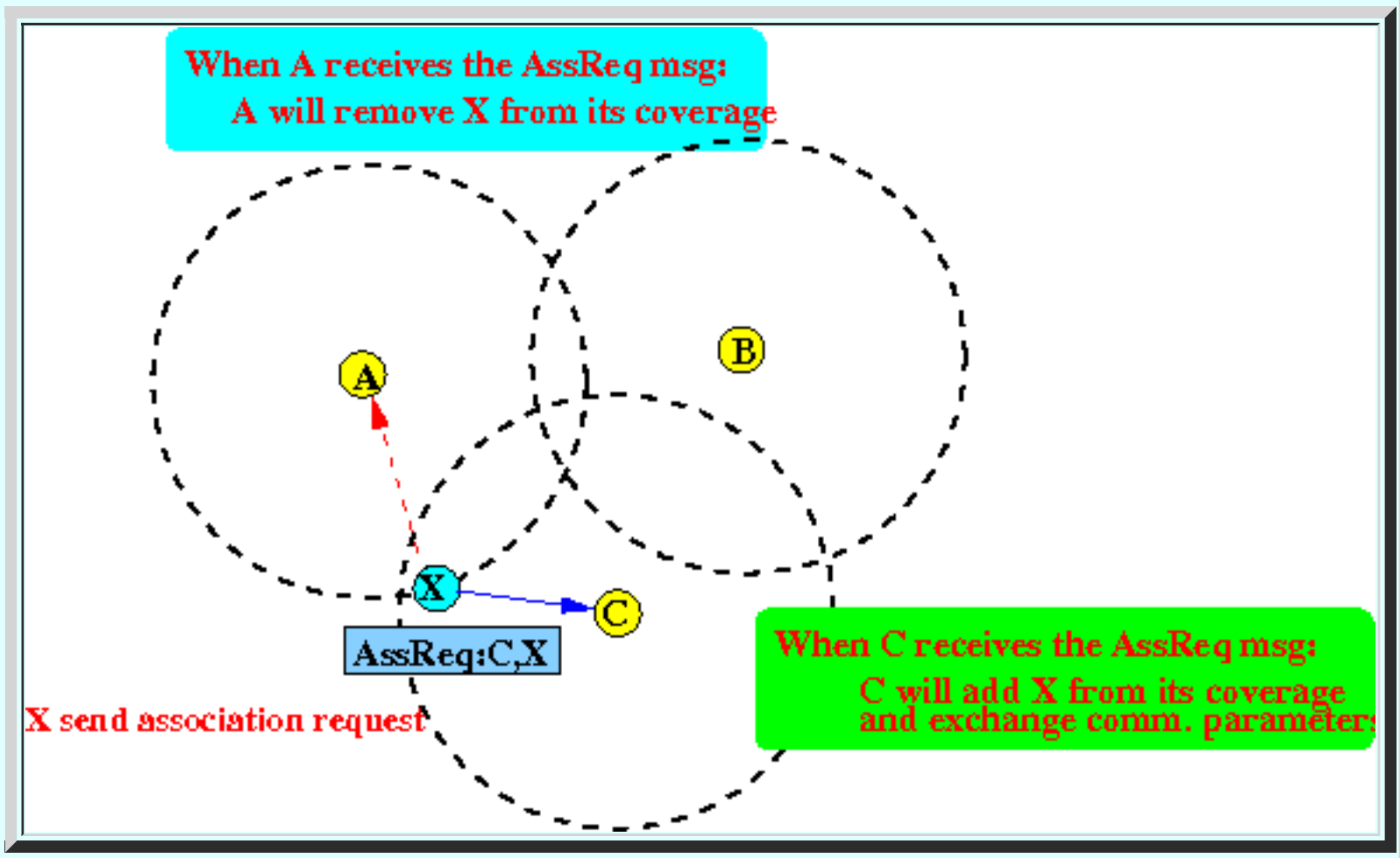
    Graphically:



    - When an **AP receives the probe**, it transmits a **response** to the requester to **identify itself**:



- The **requester node** will **associates** itself with the **AP** whose **signal strength** is the **highest** by sending an ***Association Request***:



The ***old*** **associated node** will ***also*** receive the **association request**:

  - This is an **indication** for the **old associated AP** to **remove** the **mobile node** (and **free buffers**)

- **Operational remarks**
  - **Fact:**
    - **By default:**

- A **wireless node** will perform ***both*** **types of scans** on **all channels** allowed by the **country of operation**

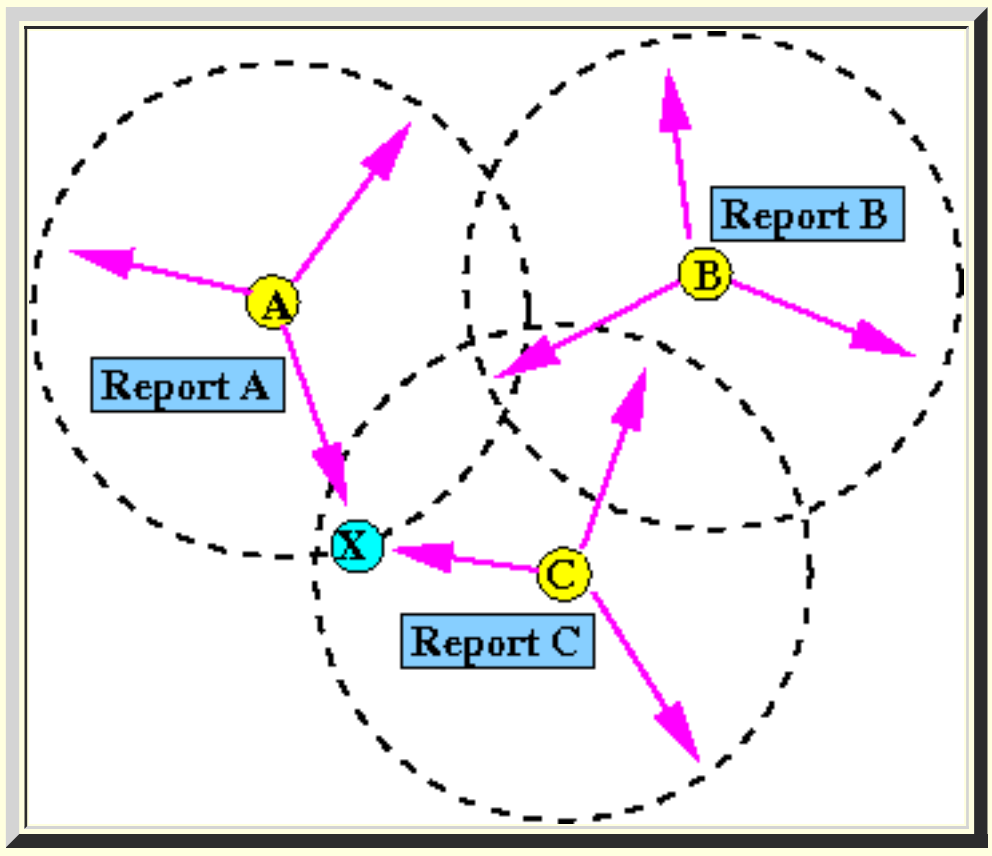
However:

- (While both types of scanning are on by default) ***active*** **scanning** is **performed** ***only*** on **channels** on which **local government regulations** allow **mobile node** to **transmit**.

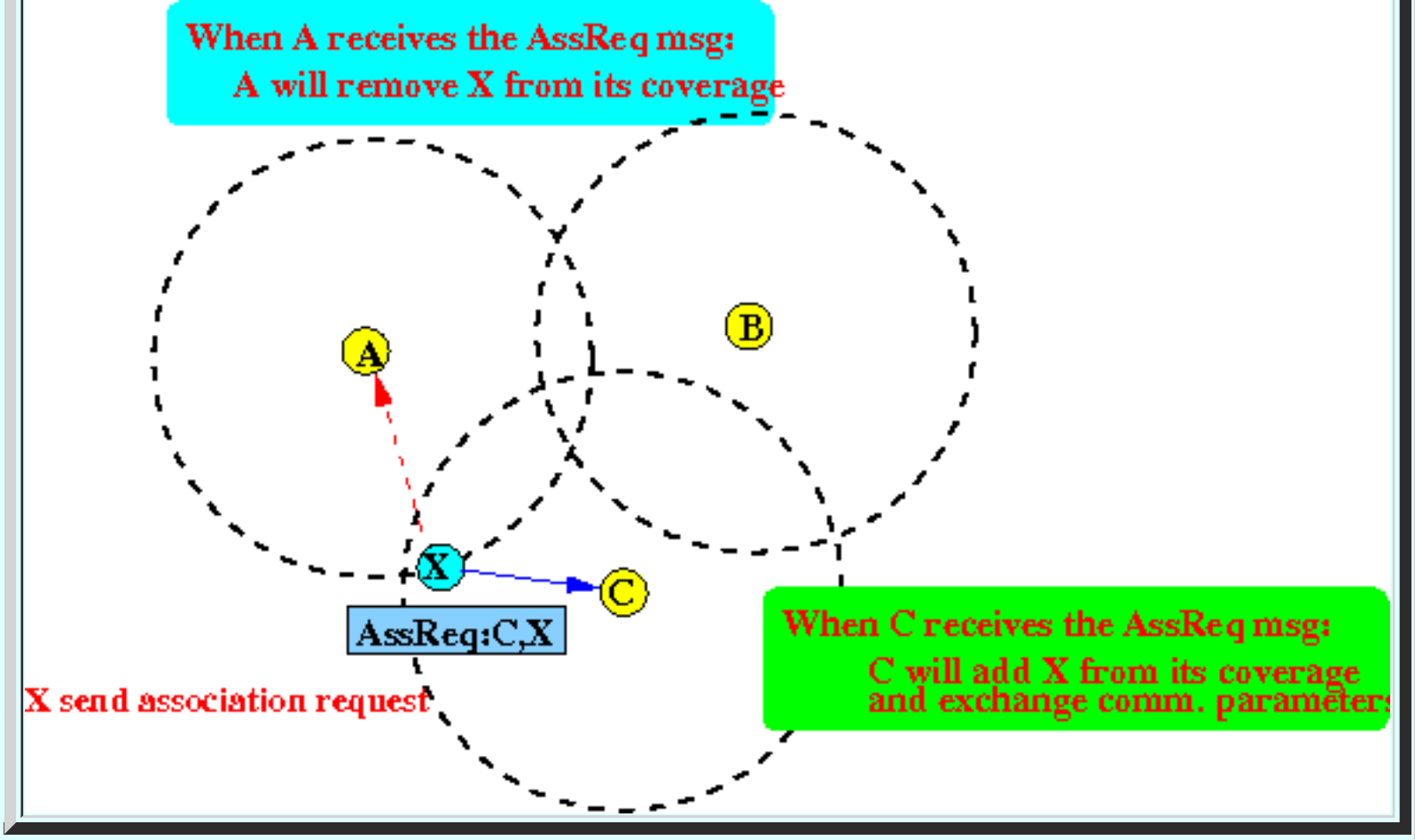
- **No active scanning** on these **channels** allowed:
  - **Channels** that are **not authorized** for **unlicensed use**
  - **Channels** that require **radar detection** with **dynamic frequency selection (DFS)**

Passive Scanning

- **Passive Scanning**
  - **Passive scanning**:
    - **Passive scanning** is ***always*** **enabled** and ***cannot*** be **disabled** !!!
    - In ***passive*** **scanning**, the **mobile node** will ***passively*** **listen** for **AP transmissions**
- **The passive scanning procedure**
  - The **passive scanning procedure**:
    - ***Each*** **AP periodically broadcasts** a ***capacity report*** **message** to **enable mobile nodes** to **"find the best AP"**:



    - A **mobile node** will **continuously** **scans (monitor)** for the **capacity reports** transmitted by **APs**
  - **Switching AP** in **Passive Scanning**:
    - When a **node** discovers a ***better*** **AP** (an **AP** with a ***stronger*** **signal**, it will **transmit** a ***Association Request*** **message** to that **AP**:



The ***old*** **associated node** will ***also*** receive the **association request**:

- This is an **indication** for the **old associated AP** to **remove** the **mobile node** (and **free buffers**)

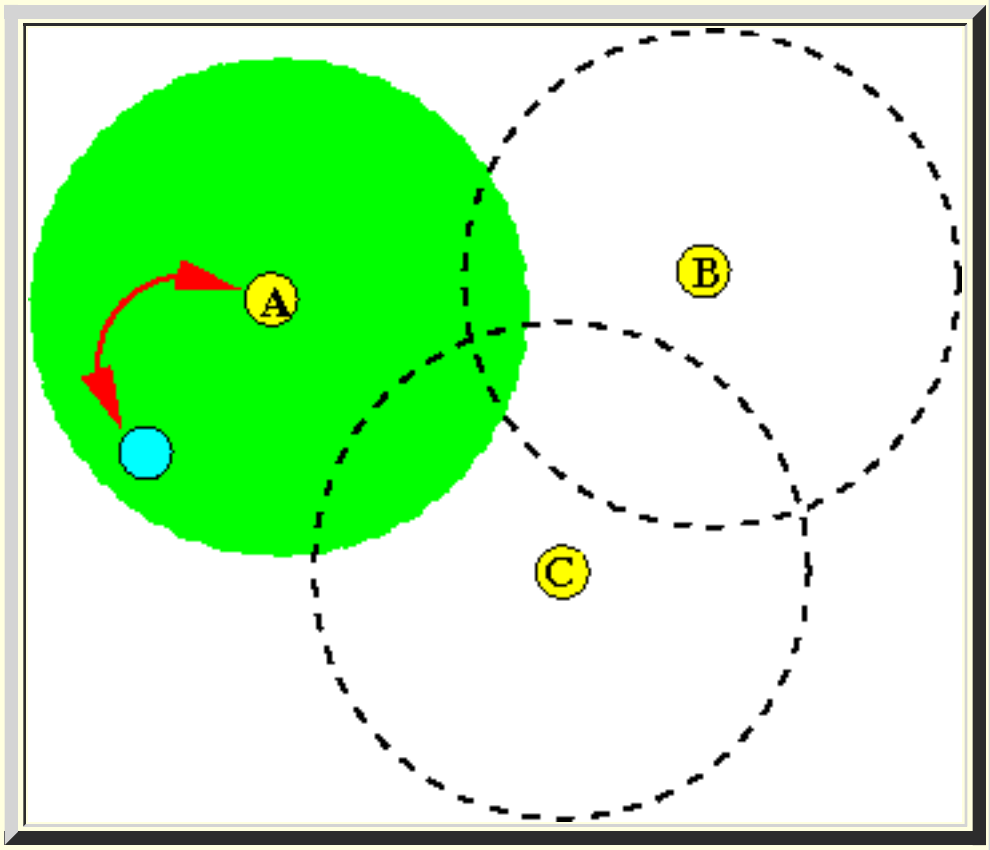
Node association clean up --- time out

- **Notatble fact**
  - **Notable fact:**

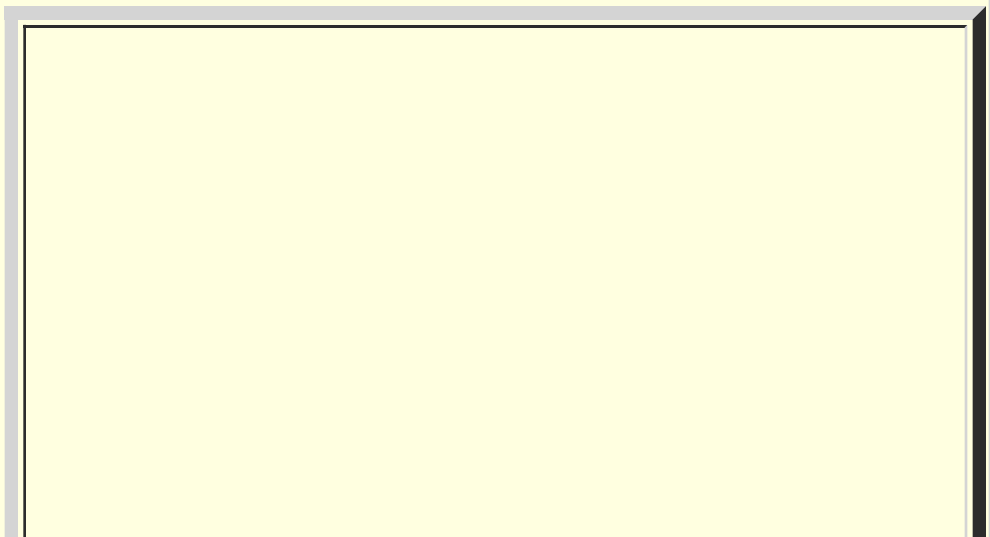
    - A **mobile node** can ***leave*** the **coverage area** of its ***associated*** **AP** ***without*** the **AP knowing** !!!

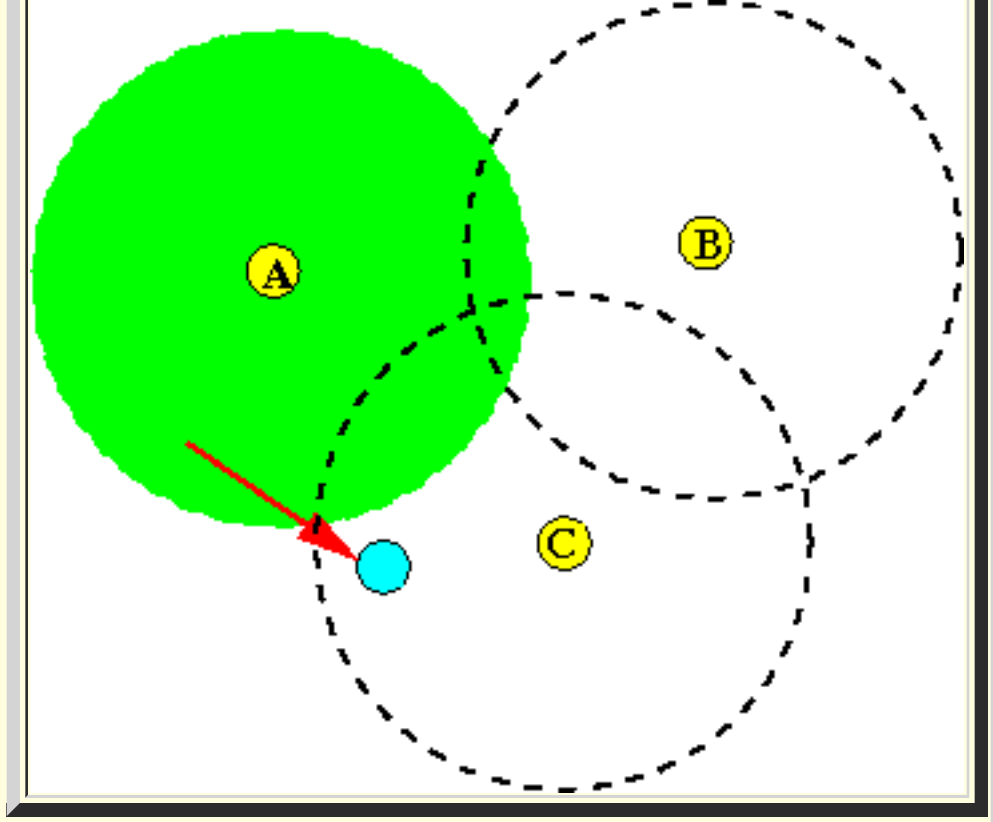
  - **Example:**

    - The **node** is **currently** **associated** with ***A***:



    - The **node** can **move** out of **range**:





without transmitting a **association frame** !!!

- **Question:**

  - **How** can the ***old*** **associated AP** **remove** a **mobile node** that has ***silently*** **moved out of range** ???

- **Time outs on AP associations**

  - **Solution** of the **"*out of range*" mobile node**:

    - An **associations** with **mobile node** has an **expiration time**

    - When the **Access Point** receives a **frame** from the **mobile node**:

      - The **Access Point** will ***refresh*** the **expiration time**

    - When the **expiration time** has **expired** (**i.e.**, the **Access Point** has **not heard** from that **node** for ***longer*** than the **expiration time**):

      - The **AP** will **remove** the **node's association** !!!
