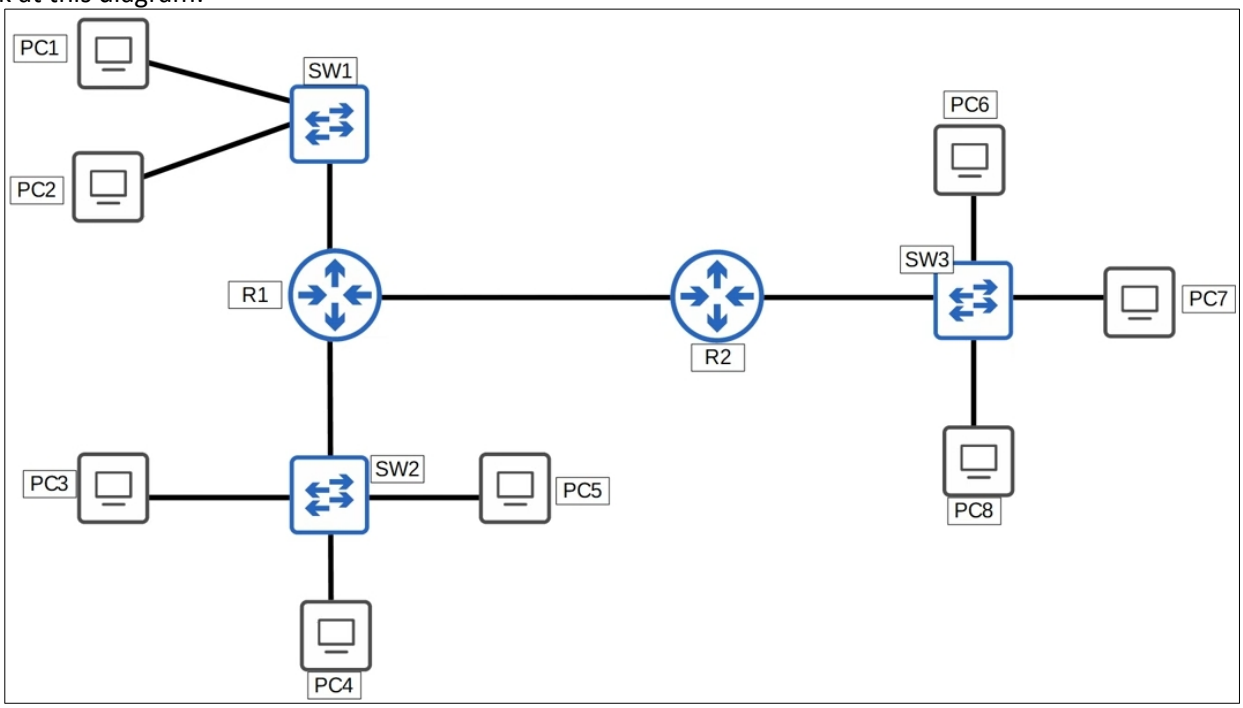
Day 16: VLANs pt.1:-

VLANs stands for, **Virtual Local Area Network**, or just **Virtual LAN**.
A VERY important Topic in the CCNA exam or when working as a Network Engineer.

→ LAN / Broadcast Domains:-

→ **LAN**: is a group of devices (PCs, Servers, Routers, Switches, etc.) in a single location (home network, office, etc.).
→ A more specific definition: A **LAN** is a single *broadcast domain*, including all devices in that broadcast domain.
→ A *broadcast domain* is the group of devices which will receive a broadcast frame (destination MAC of all Fs, **FFFF.FFFF.FFFF**) sent by any one of the group members.

→ Let's look at this diagram:



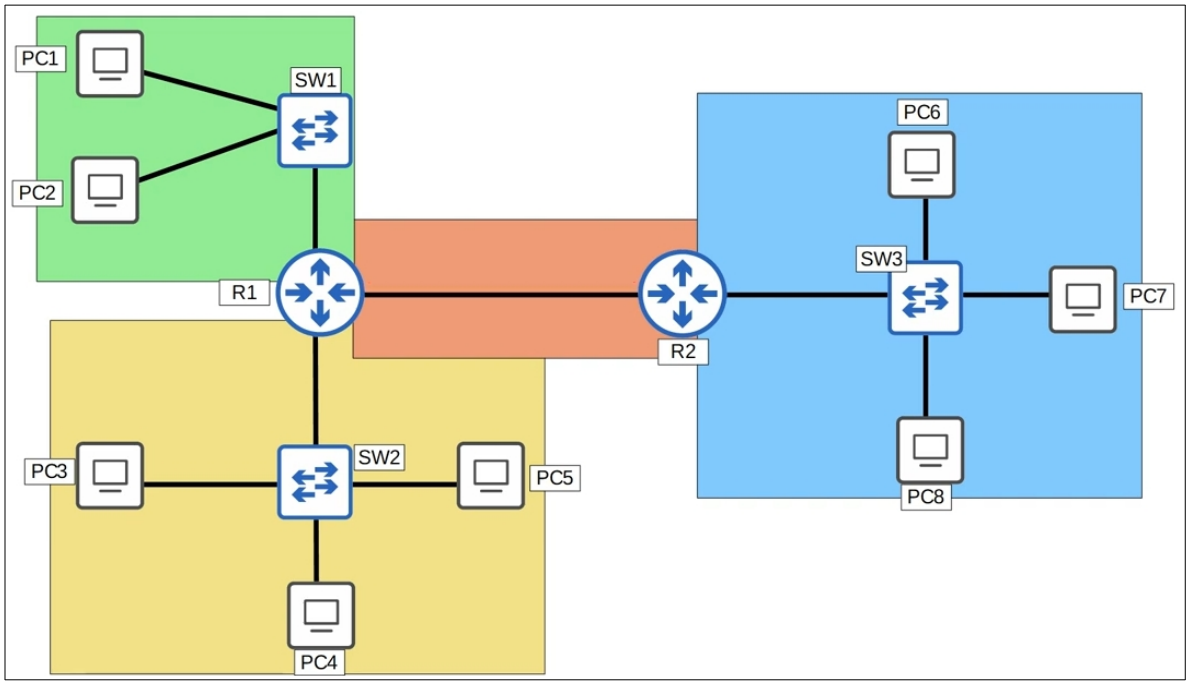
How many *broadcast domains* do you think there are?

- If **PC1** sends a broadcast frame, what will **SW1** do? It floods it out all of its interfaces except the one it was received on.
  So, **R1** receives the frame, but it doesn't forward it.
  That means this is one broadcast domain, including **PC1**, **PC2**, **SW1**, and one of **R1**'s interfaces.

- If **PC3** sends a broadcast frame, it will received by **SW2**, and then will be flooded out of all interfaces to, **R1**, **PC4**, and **PC5**.
  R1 will not forward the broadcast frame.
  So, that's the broadcast domain, **PC3**, **PC4**, **PC4**, **SW2**, and one of **R1**'s interfaces.

- If **PC6** sends a broadcast frame, **SW3** will flood the frame to **PC7**, **PC8**, and one of the **R2**'s interfaces.
  And **R2** will not forward the frame.
  So, this is a new broadcast domain, including **PC6**, **PC7**, **PC8**, **SW3**, and one of **R2**'s interfaces.

- What if **R1** sends a broadcast frame out of its interface which is connected to **R2**? It will received only by **R2**.
  However, even though this is a connection with only two devices, it is still *technically* a broadcast domain.

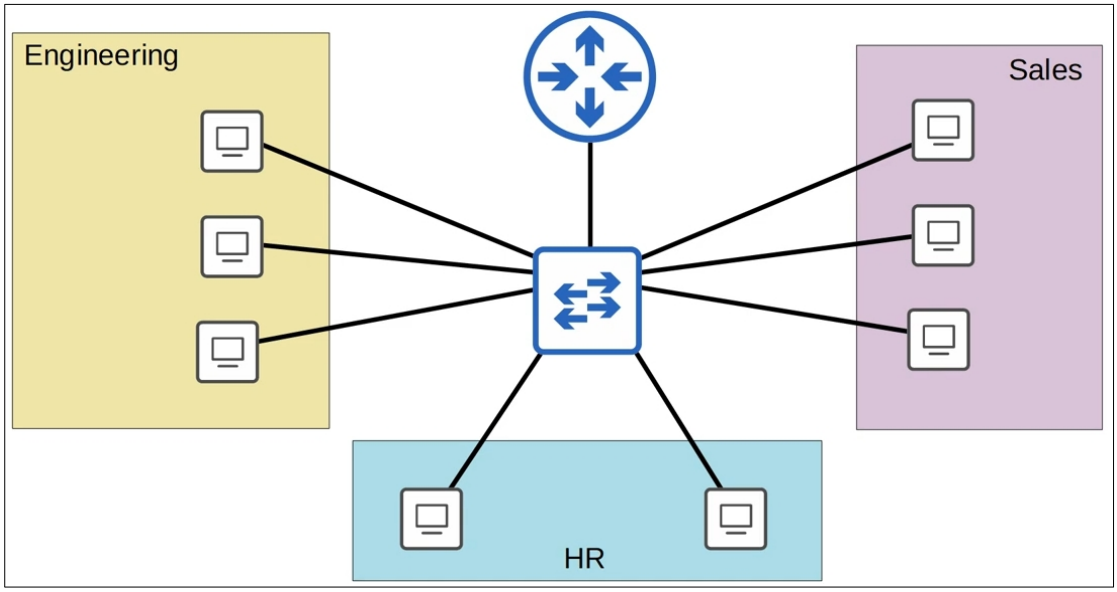


There are **4** broadcast domains in this diagram. And therefore, **4 LANS**.

→ What is a VLAN?

- Here's is a small LAN of a company.

  Let's say there are three main departments in this office, Engineering, Sales, and Human Resources.
  Also the company is using the **192.168.1.0/24** network for this LAN.



- This isn't necessarily the best setup, for both security and performance purposes.
- It would be best to split up these into separate subnets!

- For example, Let's say a PC in the engineering department sends a broadcast message intended for other PCs in the engineering department. Since it's a broadcast message, the Switch will flood it out of all interfaces.

- So, not only will the PCs in the engineering department receive the broadcast, ALL PCs, as well as the Router, will receive the broadcast message.

→ This is a BIG problem, for both security and network performance purposes:
- When it comes to performance, lots of unnecessary broadcast traffic can reduce network performance.
- Whether it's a broadcast from one end host, or a Switch that doesn't know how to reach the destination MAC address so it floods the frame. We should minimize unnecessary traffic in our network.

- As for security, Even within the same office, you want to limit who has access to what.
You can apply security policies on a Router/Firewall.
Because this is one LAN, PCs can reach each other directly, without traffic passing through the Router.
So, even if you configured security policies, they won't have any effect!!

- We should separate these hosts, so we can apply security policies that determine who can access what in the network.

→ Let's split up these departments into separate subnets:-
- **192.168.1.0/26** for the Engineering department, **192.168.1.64/26** for the HR department, and **192.168.1.128/26** for Sales.
HOWEVER, there is a problem!!
- The Router is going to need an IP address in each subnet, so it will need one interface in each subnet.

- So, let's replace this single connection between the Switch and Router with three separate connections, one in each subnet.

-- Actually, there is a more efficient way of doing this, you don't actually have to use three separate interfaces, but don't worry about that for now, We'll cover that in a future video.

- So, you may think the problems is solved now!!!

- Let's say this PC in the Engineering department has an IP of **192.168.1.1**, and this PC in the Sales department has an IP address of **192.168.1.129**.
- If **PC1** sends some data to **PC2**, **PC1** will recognize that **PC2** is in a different subnet than its own.
So, it will set the destination MAC address to its default gateway, **R1**.
This is what the frame looks like, Source IP of **PC1**, destination IP of **PC2**, Source MAC of **PC1**, destination MAC of **R1**.

- **PC1** will forward the frame to the Switch, which will send it to **R1**, which will then change the source MAC to its own MAC, and the destination to **PC2**'s MAC.

Src. IP: 192.168.1.1
Dst. IP: 192.168.1.129
Src. MAC: PC1 MAC
Dst. MAC: R1 MAC

Src. IP: 192.168.1.1
Dst. IP: 192.168.1.129
Src. MAC: R1 MAC
Dst. MAC: PC2 MAC

- It will then forward the frame back to the Switch, which will then forward it to the destination, **PC2**.

- Okay, so instead of **PC1** being able to send traffic directly to **PC2**, we *forced* it to send the traffic through **R1** first, where we would have configured some *Security Policies* and such to control exactly what traffic is allowed to pass between these subnets.

- However, there is still a problem:-
What if the frame is a broadcast or unknown unicast frame??
→ The Switch will <u>flood</u> the frame out of all interfaces.

→ For example, here is a broadcast frame:
The source IP is **PC1**'s IP, and the destination IP is its subnet's broadcast address.
So, this is a broadcast frame intended to the Engineering department.
The source MAC is **PC1**' and the destination is the broadcast MAC address of all **F**'s.
Where is the problem?????????
Well, remember that a Switch is only aware up to Layer2. It looks at Layer2 information like source and destination MAC addresses only. It doesn't care about Layer3, 4, etc.
So, even though there are three separate subnets here, the Switch doesn't know that!!!

Src. IP: 192.168.1.1
Dst. IP: 192.168.1.63
Src. MAC: PC1 MAC
Dst. MAC: FFFF.FFFF.FFFF

→ **PC1** will send the frame to the Switch, it will see the destination MAC address of all **F**'s, and then *flood* the frame!
This is *bad* in terms of both Network Performance and Security.

→ We separated the three departments into three subnets, meaning they are separated at Layer3.
They are still in the same broadcast domain, the same Layer2 network, or the same LAN.

- Now, one possible solution is to buy a separate Switch for each department.
- However, that is not very flexible, and network equipment isn't cheap. Buying one or more Switches for every single department could be too expensive, especially for a small enterprise.

- However, this is where **VLANs** come in!

→ Separating at Layer2:
- Although these PCs are all in the same LAN, we can use **VLAN**s, or **Virtual Local Area Networks** to separate them at Layer2.
- We'll assign the Engineering department to **VLAN10**, the HR department to **VLAN20**, and the Sales department to **VLAN30**.
- How exactly do we assign these hosts to **VLANs**? We configure them on the Switch, more specifically on the Switch interfaces.
- We configure the Switch interface to be in a specific **VLAN**, and then the end host connected to that interface is part of that VLAN.
- The Switch will consider each **VLAN** as a separate LAN, and will not forward traffic between **VLAN**s, including broadcast or unknown unicast traffic.

- So, if we have set up these **VLAN**s, if **PC1** sends this same broadcast frame, after the frame arrives at the Switch, it will be forwarded to all interfaces *in the same VLAN*.

Src. IP: 192.168.1.1
Dst. IP: 192.168.1.63
Src. MAC: PC1 MAC
Dst. MAC: FFFF.FFFF.FFFF

- Because the broadcast arrived on an interface configured in **VLAN10**, the Switch will only forward the frame to the other interfaces in **VLAN10**.

- If **PC1** wants to send this unicast frame to **PC2**, it will function just like before.
- It sends it to the Switch, which sends it to the Router, which changes the Source and Destination MAC addresses, and sends it back to the Switch, which sends it to the destination.

Src. IP: 192.168.1.1
Dst. IP: 192.168.1.129
Src. MAC: PC1 MAC
Dst. MAC: R1 MAC

- Note that the traffic arrives on a **VLAN10** interface is forwarded out of a **VLAN10** interface. Also traffic that arrives on a **VLAN30** interface is forwarded out of a **VLAN30** interface. Both in the same VLAN.

NOTE: *The Router is used to route between **VLANs**.*
*The Switch does not perform this **inter-VLAN Routing**. It must send the traffic through the Router*
*A Switch will never forward traffic directly between hosts in deifferent **VLAN**s.*

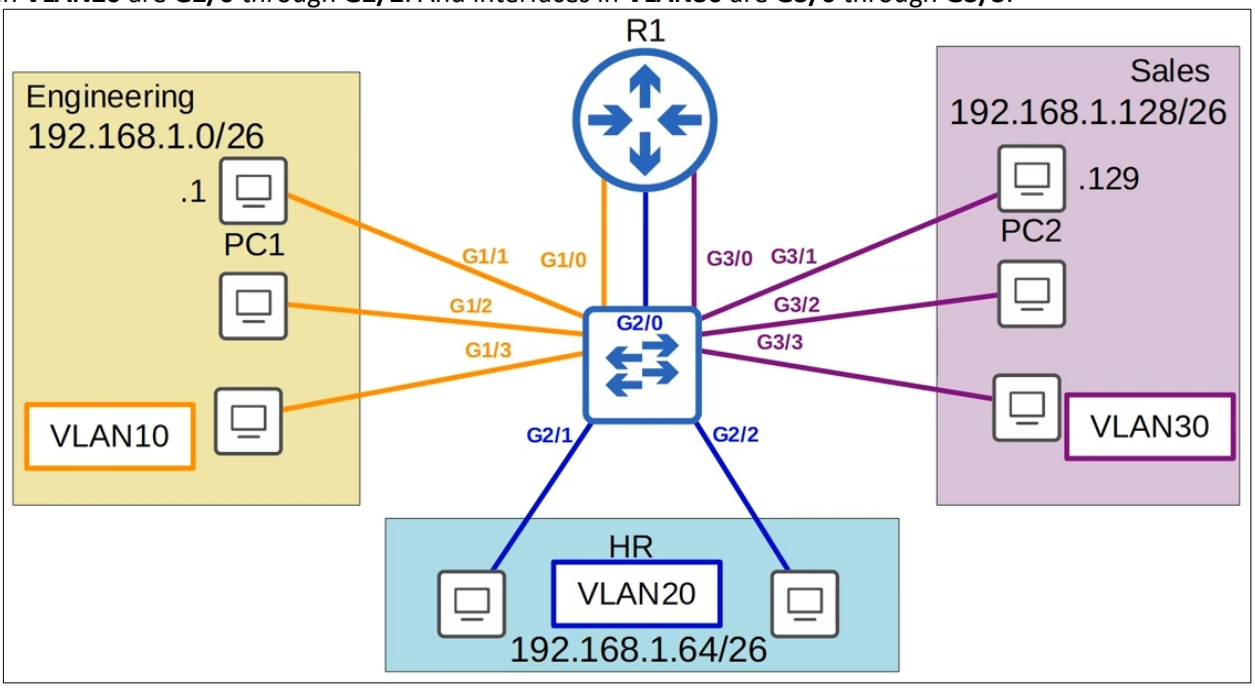
- First of all, the two hosts are in separate subnets, so **PC1** itself will send the traffic to its default gateway, **R1**.
- However, even if **PC1** and **PC2** were in the same subnet, the Switch wouldn't forward the traffic from **PC1** to **PC2**, because they are in separate **VLAN**s.

→ VLANs :-
- are configured on Switches on a *per-interface* basis.
- *logically* separate end hosts at Layer2.

→ VLAN Configuration:-

Let's take a look at basic **VLAN** configuration:

We've added the interface numbers to the diagram, interfaces in **VLAN10** are **G1/0** through **G1/3**.
Interfaces in **VLAN20** are **G2/0** through **G2/2**. And interfaces in **VLAN30** are **G3/0** through **G3/3**.



→ CLI, and put these interfaces into the proper **VLAN**s:
Let's look first at the **VLAN** that exist by default on a Switch:
In this output, we used the command (**`show vlan brief`**)

```
SW1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
                                                Gi1/0, Gi1/1, Gi1/2, Gi1/3
                                                Gi2/0, Gi2/1, Gi2/2, Gi2/3
                                                Gi3/0, Gi3/1, Gi3/2, Gi3/3
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
SW1#
```

- It displays the **VLANs** that exist on the Switch, and which interfaces are in each **VLAN**.
- We can see **VLAN1**, with the name '***default***'. This is the **VLAN** that all interfaces are assigned to by default.
- Even if we don't configure any **VLAN**s, all interfaces are in the **VLAN1** by default.
- Under **Ports**, you can see all of the interfaces on this device, from **Gi0/0** to **Gi3/3**.

- Under **VLAN1**, there are four other **VLAN**s, **1002** to **1005**, used for **FDDI** and **token-ring**.
  These are old technologies that we don't need to know for the CCNA, but feel free to google them!

- **VLAN**s **1**, **1002**-**1005** exist by default, and **cannot** be deleted!

→ Assign Interfaces to a VLAN:-

```
SW1(config)#interface range g1/0 - 3
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 10
% Access VLAN does not exist. Creating vlan 10
SW1(config-if-range)#interface range g2/0 - 2
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 20
% Access VLAN does not exist. Creating vlan 20
SW1(config-if-range)#interface range g3/0 - 3
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 30
% Access VLAN does not exist. Creating vlan 30
SW1(config-if-range)#
```

- First, we use the command (**`interface range`**) to configure all of the **VLAN10** interfaces at once.
- Use the (**`switchport mode access`**) command to set the interface as an *access port*.

→ An ***access port*** is a switchport which belongs to a single **VLAN**, and usually connects to end hosts like PCs.
That's why it's called an access port, it gives the end hosts *access* to the network.
→ There is another important type of switchports called a ***trunk port***.
Switchports which carry multiple **VLAN**s are called '***trunk ports***'.
→ ***trunk ports*** carry traffic from multiple **VLAN**s on a single interface.

- A switchport connected to an end host should enter access mode by default, however it's always a good idea to explicitly configure the setting and not rely on auto-negotiation of port type.

- And then we use (**`switchport access vlan 10`**).
This is the command that actually assigns the **VLAN** to the port.

- Notice the message that appears after this command, '*%Access VLAN does not exist. Creating vlan 10*'.
Because **VLAN10** didn't exist on the device yet, it was created automatically when we assign the interface to **VLAN10**.

- Next, we again use the command (**`interface range`**) to configure all of the **VLAN20** interfaces at once.
- We used the same (**`switchport mode access`**) command, then (**`switchport access vlan 20`**) to assign the interfaces to **VLAN20**.

- Finally, we did the same for **VLAN30**, and once again, the **VLAN** was created automatically.

- So, we used the (**`show vlan brief`**) command once again, and here we can see the three **VLAN**s we created, and the ports we assigned to each **VLAN**.

```
SW1(config)#do show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
                                                Gi2/3
10   VLAN0010                         active    Gi1/0, Gi1/1, Gi1/2, Gi1/3
20   VLAN0020                         active    Gi2/0, Gi2/1, Gi2/2
30   VLAN0030                         active    Gi3/0, Gi3/1, Gi3/2, Gi3/3
```

- Notice the default names of each **VLAN**, let's change those to make it more understandable.

```
SW1(config)#vlan 10
SW1(config-vlan)#name ENGINEERING
SW1(config-vlan)#vlan 20
SW1(config-vlan)#name HR
SW1(config-vlan)#vlan 30
SW1(config-vlan)#name SALES
```

- So, we use the (**`vlan 10`**) command to enter configuration mode for **VLAN10**.
- By the way, this is the command to crate a **VLAN**, also.
  But in this case, it was already automatically created when we assigned the interfaces.
- Next, we assign the name with this simple command, (**`name ENGINEERING`**).
- Then, we do the same with **VLAN20**, **HR**, and then with **VLAN30**, **SALES**.

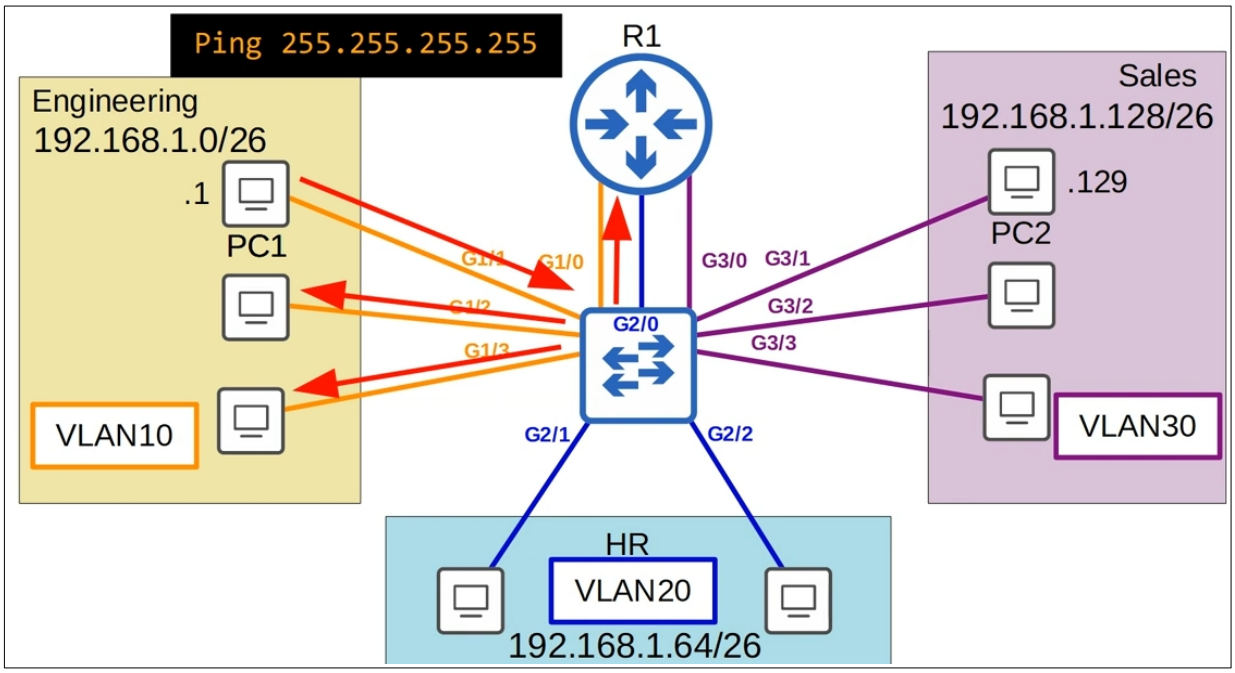
- Finally, We confirmed once more with (**`show vlan brief`**).

```
SW1(config)#do show vlan brief

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Gi0/0, Gi0/1, Gi0/2, Gi0/3
                                                 Gi2/3
10   ENGINEERING                       active    Gi1/0, Gi1/1, Gi1/2, Gi1/3
20   HR                                active    Gi2/0, Gi2/1, Gi2/2
30   SALES                             active    Gi3/0, Gi3/1, Gi3/2, Gi3/3
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
1005 trnet-default                     act/unsup
SW1(config)#
```

Notice that the names have been changed to ENGINEERING, HR and SALES.
That's all for the configurations!

If we use the command (**ping 255.255.255.255**) on **PC1**, which sends a ping with the destination MAC address of all **F**s, the broadcast MAC, the broadcast will only reach hosts in **VLAN10**!
Likewise, if we do the same on **PC2**, the broadcast will only reach PCs in **VLAN30**.



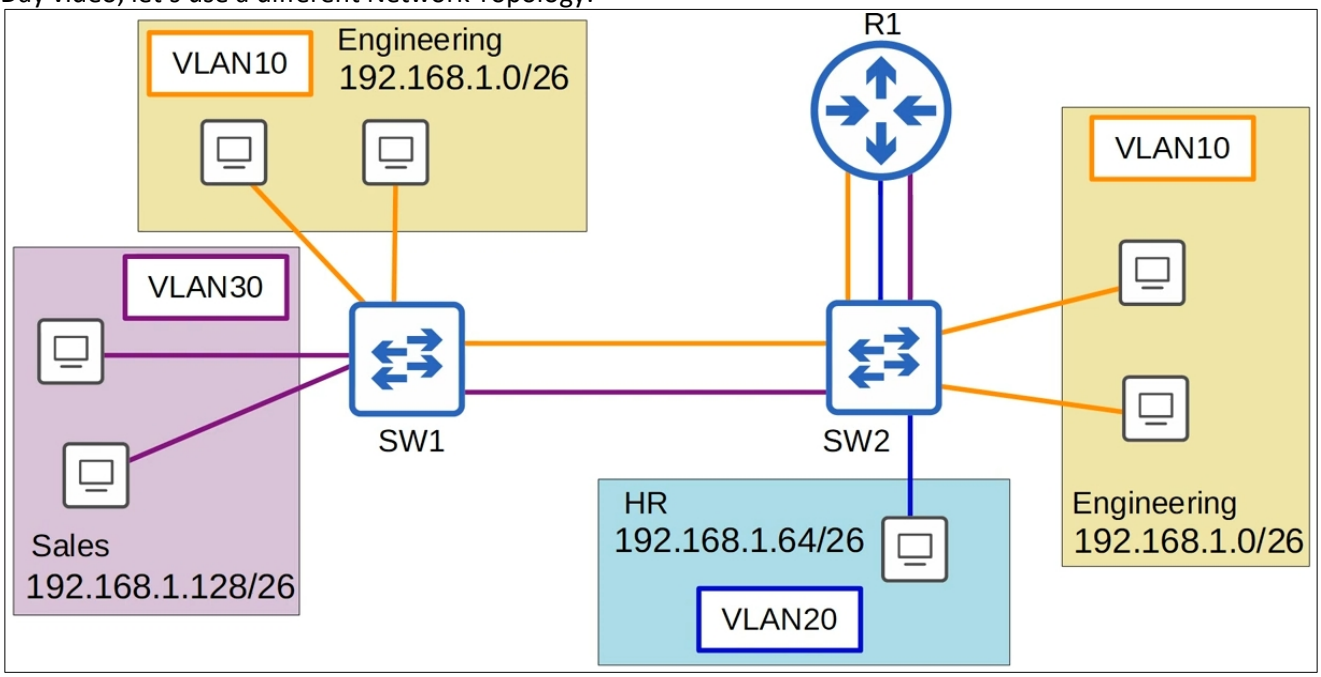
Day 17: VLANs pt.2:-

→ Trunk Ports:-

Whereas an *access port* belongs to a single VLAN, ***trunk ports*** carry traffic from multiple **VLAN**s on a single interface.

Network Topology from the Day16 video, as you remember!
All of the Switch interfaces are access ports which belongs to a single VLAN, either **VLAN10**, **VLAN20**, or **VLAN30**.
Three interfaces are used to connect to the Router, one for each VLAN.

For this Day video, let's use a different Network Topology:

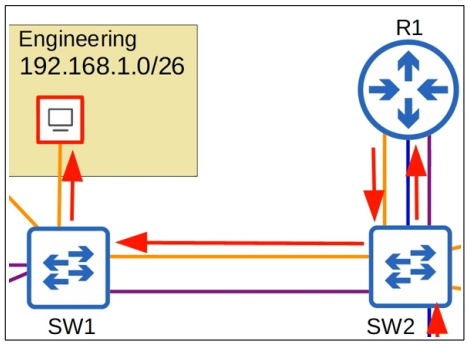


- This time there are **2** Switches used.
- Note that **VLAN10**, the VLAN for the Engineering dept, is split between the two Switches.
- This is very common, as departments in a company aren't always split up exactly by location.
  You might have some engineers on one floor of the building, for example, and some on another floor.

- We are still using only *access ports*.
- There are **2** links between **SW1** and **SW2**, one for **VLAN10**, and one for **VLAN30**.
- There must be a link in **VLAN10** between the two Switches, because **VLAN10** PCs are connected to both **SW1** and **SW2**.
  And also because the PCs connected to **SW1** need to be able to reach **R1** via **SW2**.
- As for the link in **VLAN30**, it is necessary because PCs in **VLAN30** also need to be able to reach **R1** via **SW2**.
- There is no link in **VLAN20** between **SW1** and **SW2**.
  This is because there are no PCs in **VLAN20** connected to **SW1**.
  PCs in **VLAN20** can still reach PCs connected to **SW1**, **R1** will perform ***inter-VLAN routing***.

→ inter-VLAN routing:-

- Let's say this PC in **VLAN20** wants to send traffic to one of the **VLAN10** PCs connected to **SW1**.
- It will send the frame with a destination MAC address of **R1**, its default gateway.
  R1 then forwards it back to **SW2**.
- Note that this traffic arrived at **SW2** on the **VLAN10** interface,
  the traffic is now in **VLAN10**, so it forwards it to **SW1** on the **VLAN10** connection between them, and then **SW1** forwards the traffic to the destination PC.
- You can see that, even though there isn't a **VLAN20** connection between **SW2** and **SW1**, the PC in **VLAN20** can still send traffic to the PC in **VLAN10**, because the Router performs *inter-VLAN routing*.



→ Trunk Ports:-

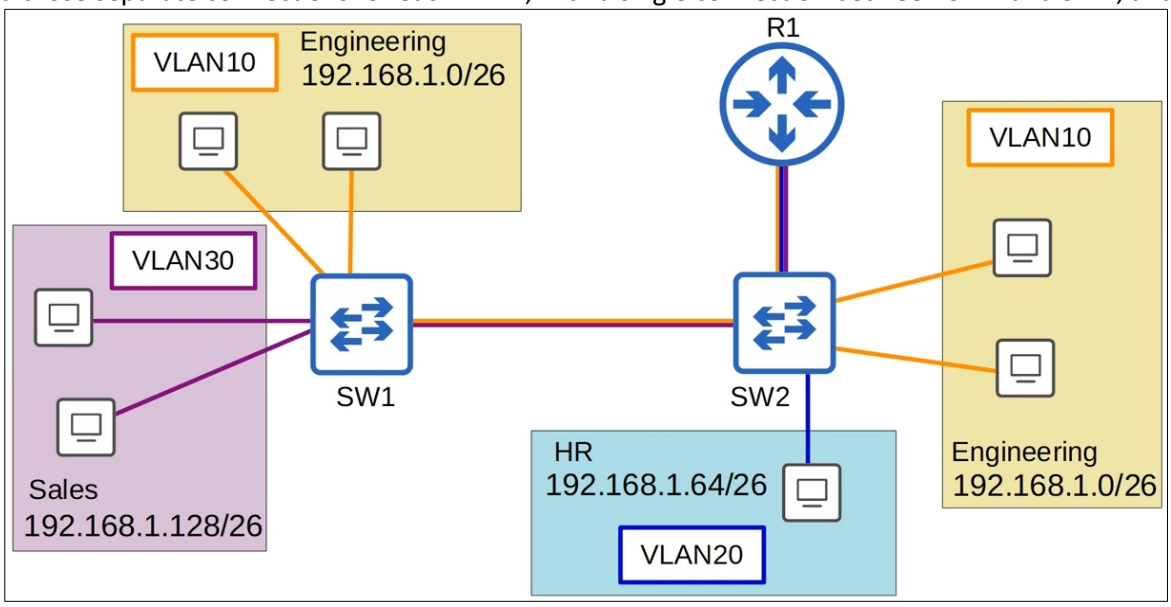
- In a small network with few VLANs, it is possible to use a separate interface for each VLAN when connecting Switches to Switches, and Switches to Routers.

- However, when the number of VLANs increases, this is not viable.
It will result in wasted interfaces, and often Routers won't have enough interfaces for each VLAN.

- We can use ***Trunk ports*** to carry traffic from multiple VLANs over a single interface!

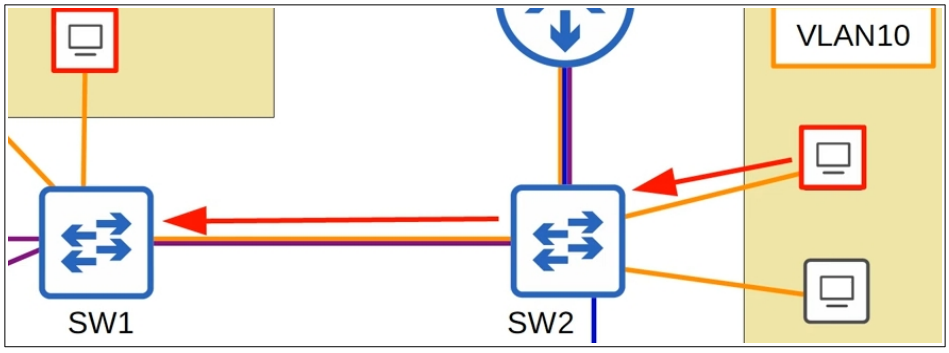
→ How Trunk Ports work?:-

We've replaced those separate connections for each VLAN, with a single connection between **SW1** and **SW2**, and **SW2** and **R1**.



- Now we can see which VLANs are allowed on each trunk.
- These are *single physical* connections, but traffic from multiple VLANs is allowed over each trunk.

→ Let's say this PC in **VLAN10** wants to send some data to this other PC in **VLAN10**.
It sends the traffic to **SW2**, which then sends it to **SW1**.



→ How does **SW1** know which VLAN the traffic belongs to??
- Both **VLANs 10** and **30** are allowed on the interface the traffic was received on, but how does **SW1** know which VLAN it belongs to?
- The answer is '***VLAN tagging***'.
- Switches will ***tag*** all frames that they send over a trunk link.
This allows the receiving Switch to know which VLAN the frame belongs to.
- Another name for a '*trunk port*' is a '***tagged port***'.
- Another name for '*access port*' is '***untagged port***'.

- Frames sent over access ports aren't tagged, they don't need to be tagged because the interface belongs to a single VLAN.
- If a frame arrives on a switchport in **VLAN10**, the Switch know the frame is in **VLAN10**.

→ VLAN Tagging:-

dot1q

- There are two main trunking protocols: **ISL**, (**Inter-Switch Link**) and **IEEE 802.1Q**.
- **ISL** is an old Cisco proprietary protocol created before the industry standard **IEEE 802.1Q**.
- **IEEE 802.1Q** is an industry standard protocol created by the **IEEE** (Institute of Electrical and Electronics Engineers).
- You will probably NEVER use **ISL** in the real world. Even modern Cisco equipment doesn't support it. For the CCNA you only need to learn **802.1Q**. You should know what **ISL** is, but you don't have to study it like **dot1Q**.

- The **dot1q** tag is actually inserted between two fields of the Ethernet header.
  Here's the Ethernet header:

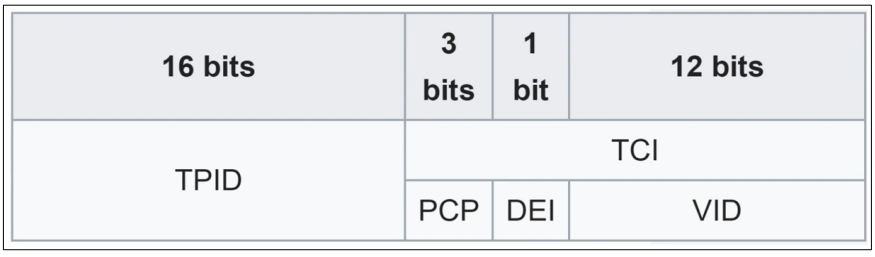
dot1Q inserts a **4**-byte, or **32**-bit field between two fields of this header.

As you can see, the **dot1Q** tag is inserted between the source MAC address and the type or length field of the Ethernet header.

→ 802.1Q Tag:-

- The **802.1Q** tag is inserted between the source MAC address and the Type/Length field of the Ethernet header.
- The tag is **4** bytes (**32** bits) in length.
- The tag consists of two main fields:
  * Tag Protocol Identifier (**TPID**)
  * Tag Control Information (**TCI**)
- The **TCI** consists of three sub-fields.

Here's a diagram of the **802.1Q** tag format:



16 bits	3 bits	1 bit	12 bits
TPID	TCI		
	PCP	DEI	VID

- It can be divided into two halves, the **TPID** and **TCI**.
- Also the **TCI** can be divided into three sub fields, the **PCP**, **DEI**, and **VID**.

→ TPID field:-

- The field is **2** bytes (**16** bits) in length, taking up half of the **802.1Q** tag's length.
- Always set to a value of **0x8100**. This indicates that the frame is **802.1Q**-tagged.
- As the **dot1Q** tag comes after the source MAC field of the Ethernet frame, This is where the TYPE field is usually located. When the Switch sees this value of **8100**, it knows it's a **dot1q**-tagged frame.

→ PCP field:-

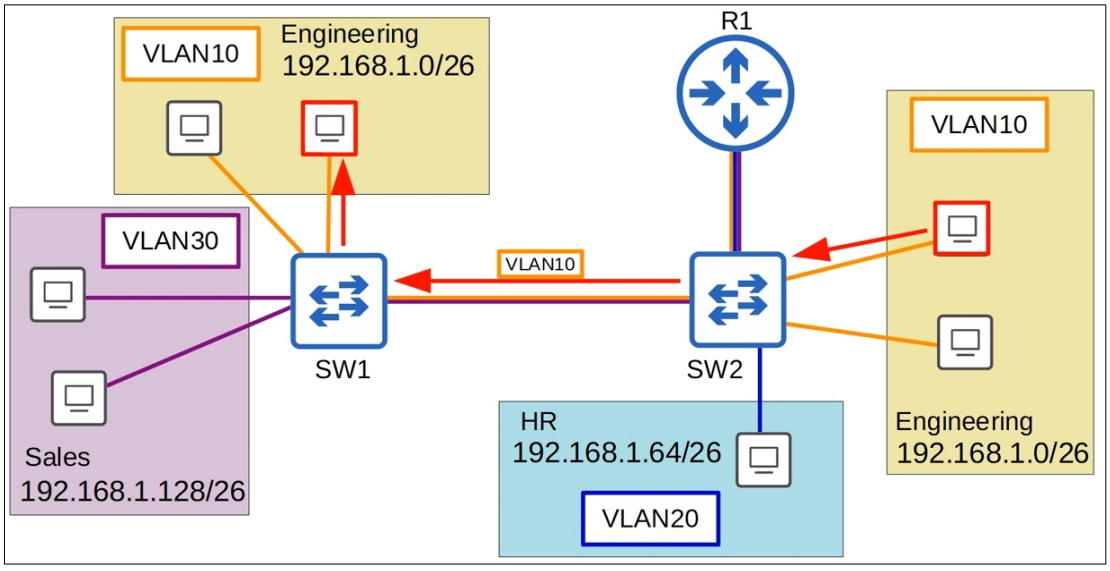
- Stands for **Priority Code Point**.
- **3** bits in length.
- Used for Class of Service (**CoS**), which prioritizes important traffic in congested networks.

→ DEI field:-
- Stands for **Drop Eligible Indicator**.
- **1** bit in length.
- Used to indicate frames that can be dropped if the network is congested.
  Which makes sure more important network traffic gets through.

→ VID field:-
- The **VLAN ID** field, the most important field in **802.1Q** tag.
- **12** bits in length.
- The field that actually identifies the VLAN the frame belongs to.
- Because it is **12** bits in length, that means there are (2^{12}), **4096** total VLANs, range of **0** – **4095**.
- However, the first and last VLANs **0** and **4095** are reserved and can't be used.
- The actual range of VLANs is **1** – **4094**.
- BTW, Cisco's proprietary **ISL**, which is an alternative protocol for VLAN tagging over trunk connections,
  also use a VLAN rage of **1 – 4094**.

→ VLAN Ranges:-
-The range of VLANs (**1 – 4094**) is divided into two sections:
  1- Normal VLANs: (**1 – 1005**)
  2- Extended VLANs: (**1006 – 4094**)
- Some older devices cannot use the extended VLAN range.
  However it's safe to expect that modern Switches will support the extended VLAN range.

→ Trunk Ports:-
Back to our diagram,
- So the PC in **VLAN10** wants to send traffic to this other PC in **VLAN10**.
- The traffic goes to **SW2**, which then forwards it to **SW1**, with a tag indicating that the traffic belongs to **VLAN10**.
- **SW1** receives the frame, and because the destination is also in **VLAN110**, it will forward the traffic to the destination

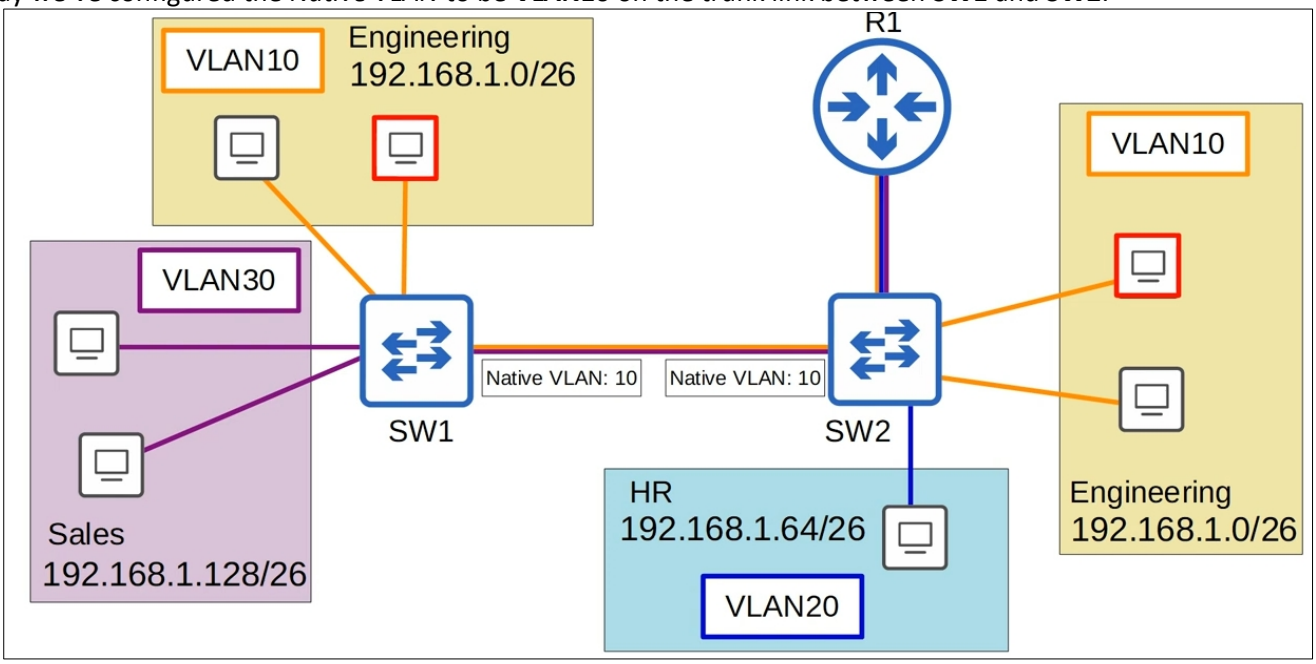


- A standard Layer2 Switch like this will only forward traffic in the same VLAN, It will not forward traffic between VLANs.

→ Native VLAN:-

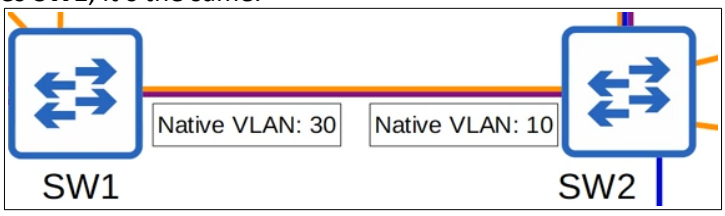
802.1Q has a feature called the *Native VLAN*. Cisco's ISL doesn't have this feature, BTW.
- The *Native VLAN* is **VLAN1** by default on all trunk ports, however this can be manually configured on each trunk port.
- ***Remember:*** This has to be configured on each trunk port separately, it's not a global configuration on the Switch.
- The Switch doesn't add an **802.1Q** tag to frames in the *Native VLAN*. It will forward the frame normally without tagging it.
- When a Switch receives an untagged frame on a trunk port, it assumes the frame belongs to *the Native VLAN.*
- So, it is very IMPORTANT that the *Native VLAN* matches between Switches.
- Switches will still forward traffic if there is a *Native VLAN* mismatch, but problems may occur.

Let's look at an example:
- Let's say we've configured the *Native VLAN* to be **VLAN10** on the trunk link between **SW1** and **SW2**.



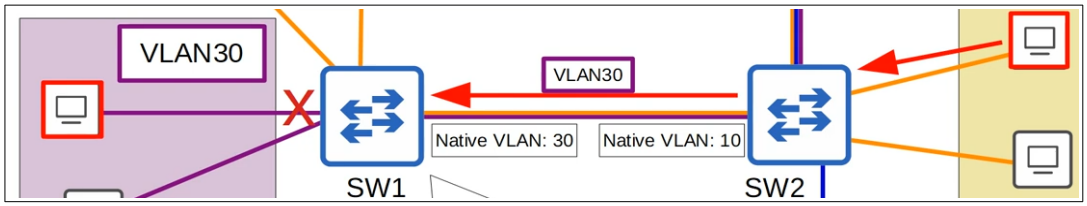
Let's follow some traffic on the same path as usual:
- The same PC from **VLAN10** sends the traffic to **SW2**.
- It will send the traffic to **SW1**, but because it is in the *Native VLAN***, VLAN10**, it won't tag it as being in **VLAN10**.
- The untagged frame arrive at **SW1**, which assumes that the traffic belongs to **VLAN10**, so it forwards it to the destination.

This time, let's look at if there is a *Native VLAN* mismatch configuration:
- On **SW2**'s interface we've configured **VLAN10** as the *Native VLAN*.
- However, on **SW1**' interface we've configured **VLAN30** as the *Native VLAN*.
- Up to the point the traffic reaches **SW1**, it's the same.



- However, when **SW1** receives the frame this is what it will think: "*This frame has no VLAN tag. It must belong to **VLAN30**.*"
  "*But the destination is in **VLAN10**, not **VLAN30**. So, It won't forward the frame.*"
- Now we can see why it is important that the *Native VLAN* configuration matches between Switches.

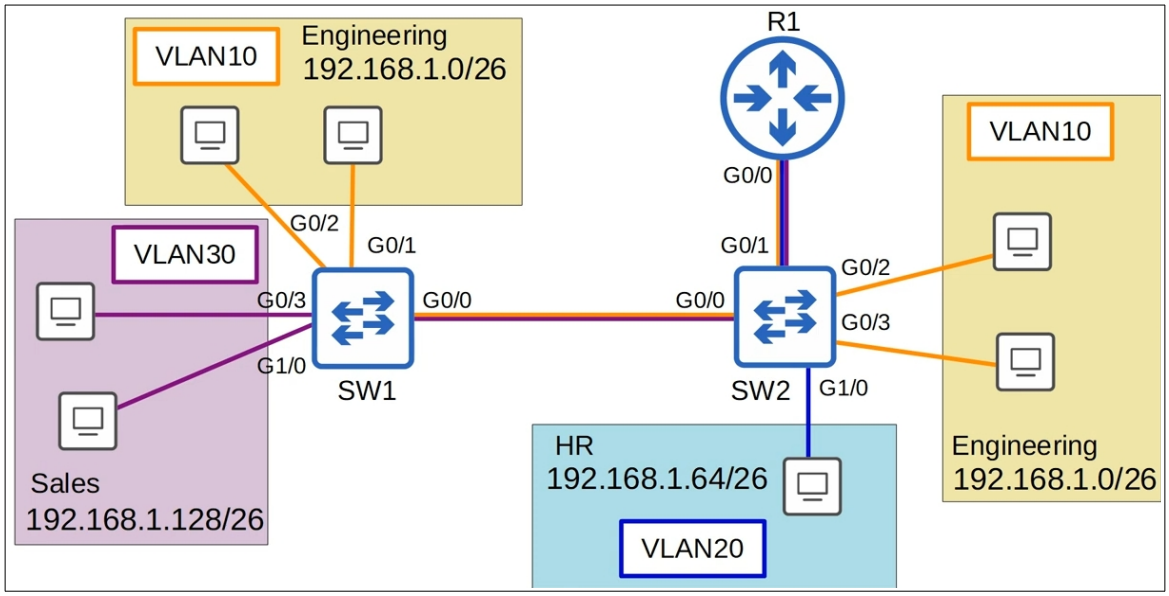
Let's look at another reason why it's important for the *Native VLANs* to match.
- This time, this PC in **VLAN10** wants to reach this PC in **VLAN30**.
- The PC sends the frame to **SW2**, which forwards it to **SW1** with a tag of **VLAN30**, since it's not the *Native VLAN* of **SW2**.
- However, **VLAN30** is the *Native VLAN* of **SW1**.
- When this frame tagged with **VLAN30** arrives, it will simply discard the frame, and will not forward it to the destination.
- Because it expects all traffic in **VLAN30** to be <u>untagged</u> on the trunk interface, it will consider the frame an error, not forward it.



<u>NOTE:</u> Make sure the *Native VLAN* matches on each Switch.

→ Trunk Configuration:-

So, we will be configuring **G0/0** on **SW1**, and **G0/0** and **G0/1** on **SW2** as trunk ports.



→ Let's go on **SW1** first:
- Fist, Let's look at the most basic trunk configuration, manually configuring the interface as a trunk.
- After entering interface configuration mode,
  use the command (**`switchport mode trunk`**) to manually configure the interface as a trunk.
- However, in this case we got an error message!
  "Command rejected: An interface whose trunk encapsulation is "Auto" can not be configured to "trunk" mode".
  This is a little tricky! Many modern Switches do not support Cisco's **ISL** at all. They only support **802.1Q**.
  Even though **ISL** is a Cisco proprietary protocol, even Cisco Switches are moving toward supporting only **dot1q**.
- However, Switches that do support both **dot1q** and **ISL** (Like the one we're using in our example)
  have a trunk encapsulation of '*Auto*' by default.
- To manually configure the interface as a trunk port, you must first set the encapsulation to **802.1Q** or **ISL**.
  On Switches that only support **802.1Q**, this is not necessary.
- After you set the encapsulation type, you then configure the interface as a trunk.

```
SW1(config)#interface g0/0
SW1(config-if)#switchport mode trunk
Command rejected: An interface whose trunk encapsulation is "Auto" can not be configured to "trunk" mode.
```

- To set the encapsulation type, use the command (**`switchport trunk encapsulation`**).
- If we use the question mark **?** To see the options, and we get (**`dot1q`**, **`isl`**, and **`negotiate`**).
  Negotiate sets it to ***Auto*** mode, so we can't choose it.

```
SW1(config-if)#switchport trunk encapsulation ?
  dot1q      Interface uses only 802.1q trunking encapsulation when trunking
  isl        Interface uses only ISL trunking encapsulation when trunking
  negotiate  Device will negotiate trunking encapsulation with peer on
             interface
```

- We set the encapsulation to **dot1q**, by the command (**`switchport trunk encapsulation dot1q`**).
- And then this time the command (**`switchport mode trunk`**) is accepted.

```
SW1(config-if)#switchport trunk encapsulation dot1q
SW1(config-if)#switchport mode trunk
SW1(config-if)#
```

NOTE: On Switches that only support **dot1q**, you will ONLY need the (**`switchport mode trunk`**) command.
But on some Switches you will need to set the encapsulation first.

- To confirm, we use the (**`show interfaces trunk`**) command.

```
SW1#show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/0       1-4094

Port        Vlans allowed and active in management domain
Gi0/0       1,10,30

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       1,10,30
SW1#
```

- First up, the trunk interfaces are listed here.
- '*Mode ON*' means that the interface was manually configured as a trunk.
- Encapsulation is **802.1q** as we configured, Status is trunking, and the Native VLAN is the default of **1**.
- Under that, the VLANs allowed on the trunk are displayed.
  By the default, ALL VLANs, **1**- **4094**, are allowed on the trunk.
- However, for security purposes, we might want to limit which VLANs can be forwarded on the trunk.
- Next up is VLANs allowed and active in management domain.
  This includes the default VLAN of **1**, as well as VLANs **10** and **30**, which I already configured on the Switch.
- Note that, although **VLAN1**, which exists by default, appears here, VLANs **1002** to **1005**, which we showed before, *do not*!

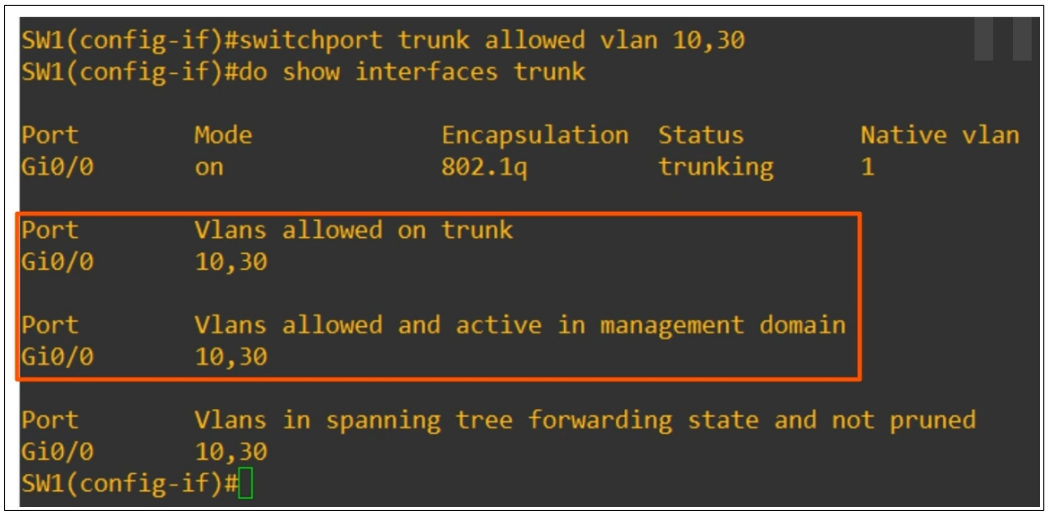
```
SW1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi1/1, Gi1/2, Gi1/3, Gi2/0
                                                Gi2/1, Gi2/2, Gi2/3, Gi3/0
                                                Gi3/1, Gi3/2, Gi3/3
10   ENGINEERING                      active    Gi0/1, Gi0/2
30   SALES                            active    Gi0/3, Gi1/0
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
SW1#
```

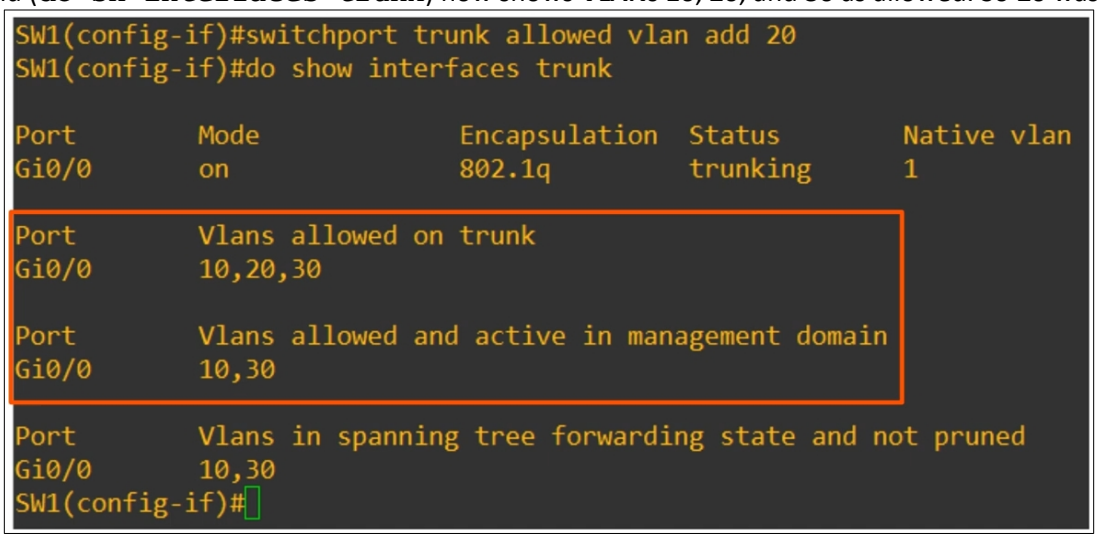
- The last field of the (**`show interfaces trunk`**) command is '*Vlans in spanning tree forwarding state and not pruned*'.

→ Here's the command to configure the VLANs allowed on a trunk
- (**`switchport trunk allowed vlan`**), and then there are some options:

1- **WORD**, allows you to simply configure the list of VLANs allowed.
VLAN IDs of the allowed VLANs when this port is in trunking mode.
So, we used the command (**`switchport trunk allowed vlan 10, 30`**)
Notice that the command (**`do sh interfaces trunk`**) now only shows **VLAN10** and **VLAN30** as being allowed on trunk.



2- **add**, allows you to add allowed VLANs to the currently existing list.
VLANs 10 and **30** are allowed, let's say we want to add **VLAN20**, even though no hosts in **VLAN20** are connected to **SW1**.
This time we use the command (**`switchport trunk allowed vlan add 20`**).
The command (**`do sh interfaces trunk`**) now shows **VLAN**s **10**, **20**, and **30** as allowed. So **20** was added to the list.



NOTE: Because we haven't actually created **VLAN20** on this Switch, **VLAN20** still isn't displayed in the VLANs allowed and active in management domain section.

3- **remove**, remove VLANs from the current list:
VLAN20 is not necessary on this trunk, so let's remove it!
We use the command (**`switchport trunk allowed vlan remove 20`**).
And then (**`do sh interfaces trunk`**).
Now it has been removed from the list of allowed VLANs. Leaving only **VLAN10** and **VLAN30**.

4- **all**, all VLANs.
This time we use (**`switchport trunk allowed vlan all`**).
Now all VLANs are allowed on the trunk.
This is the same as the default state, as all VLANs are allowed by default.

```
SW1(config-if)#switchport trunk allowed vlan all
SW1(config-if)#do show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/0       1-4094

Port        Vlans allowed and active in management domain
Gi0/0       1,10,30

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       1,10,30
SW1(config-if)#
```

5- **except**, it allows all VLANs except the ones we specify it to the command.
We use (**`switchport trunk allowed vlan except 1-5, 10`**), and then (**`do sh interfaces trunk`**).
It allows all VLANs except those, so **6-9**, and **11-4094**.

```
SW1(config-if)#switchport trunk allowed vlan except 1-5,10
SW1(config-if)#do show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/0       6-9,11-4094

Port        Vlans allowed and active in management domain
Gi0/0       30

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       30
SW1(config-if)#
```

6- **none**, no VLANs:
We use (**`switchport trunk allowed vlan none`**), and then (**`do sh interfaces trunk`**).
As we can see, no VLANs are allowed on the trunk!

```
SW1(config-if)#switchport trunk allowed vlan none
SW1(config-if)#do show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1

Port        Vlans allowed on trunk
Gi0/0       none

Port        Vlans allowed and active in management domain
Gi0/0       none

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       none
SW1(config-if)#
```

This effectively allows no traffic to pass over the trunk

→ Let's do the actual settings we want for this network:
- **SW1** has hosts in **VLAN10** and **VLAN30** connected to it. No hosts in **VLAN20** are connected.
- So there's no need to allow **VLAN20** on the trunk between **SW1** and **SW2**.
- Let's set the allowed VLANs to **10** and **30** like we did before:
  We use the command (**`switchport trunk allowed vlan 10, 30`**)
- Now the only VLANs allowed on the trunk are **VLAN10** and **vLAN30**.

→ The reason to do this is for Security Purposes, to make sure only traffic in the necessary VLANs can use that connection.
→ Also, for Network Performance Purposes, this avoid unnecessary traffic, because broadcasts and such in other VLANs won't be sent over the trunk.

→ Native VLAN:-

- For Security Purposes, it is ***best*** to change the *Native VLAN* to an **unused VLAN**.
- It is about limiting unnecessary traffic in the network, and controlling what traffic is allowed.
- **Remember:** Make the *Native VLAN* matches between Switches.

→ Change the Native VLAN:-
- The command to change it is (**`switchport trunk native vlan + n`**), while **`n`** is the VLAN number.
- We choose an unused VLAN, **1001**, (**`switchport trunk native vlan 1001`**).
- Now we use the command (**`do sh interfaces trunk`**):
- As we can see, the *Native VLAN* has now been changed to **1001.**

```
SW1(config-if)#switchport trunk native vlan 1001
SW1(config-if)#do show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1001
```

→ Trunk Configuration:-

- After configuring this trunk port, we did the command (**`show vlan brief`**).
- Notice that, **Gi0/0** is not listed anywhere. Not in **VLAN10** or **VLAN30**, even though those are the allowed VLANs on the trunk.

```
SW1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Gi1/1, Gi1/2, Gi1/3, Gi2/0
                                                Gi2/1, Gi2/2, Gi2/3, Gi3/0
                                                Gi3/1, Gi3/2, Gi3/3
10   ENGINEERING                      active    Gi0/1, Gi0/2
30   SALES                            active    Gi0/3, Gi1/0
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup
SW1#
```

- This is because the (**`show vlan brief`**) command show the access ports assigned to each VLAN,
  NOT the trunk ports that allow each VLAN.
- Use the (**`show interfaces trunk`**) command to confirm trunk ports.

→ Trunk configuration on SW2:-

- On **SW2**'s **Gi0/0** interface, we must allow **VLAN10** and **VLAN30**.
- On **SW2**'s **Gi0/1** interface, however, we must allow **VLAN20** as well.
- Here are the configurations for **SW2**'s **Gi0/0** interface, the interface connected to **SW1**:

```
SW2(config)#interface g0/0
SW2(config-if)#switchport trunk encapsulation dot1q
SW2(config-if)#switchport mode trunk
SW2(config-if)#switchport trunk allowed vlan 10,30
SW2(config-if)#switchport trunk native vlan 1001
SW2(config-if)#do show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1001

Port        Vlans allowed on trunk
Gi0/0       10,30

Port        Vlans allowed and active in management domain
Gi0/0       10,30

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       10,30
SW2(config-if)#
```

- Let's move on to **Gi0/1**, which is connected to **R1**: Here are the configurations:

```
SW2(config)#interface g0/1
SW2(config-if)#switchport trunk encapsulation dot1q
SW2(config-if)#switchport mode trunk
SW2(config-if)#switchport trunk allowed vlan 10,20,30
SW2(config-if)#switchport trunk native vlan 1001
SW2(config-if)#do show interfaces trunk

Port        Mode             Encapsulation  Status        Native vlan
Gi0/0       on               802.1q         trunking      1001
Gi0/1       on               802.1q         trunking      1001

Port        Vlans allowed on trunk
Gi0/0       10,30
Gi0/1       10,20,30

Port        Vlans allowed and active in management domain
Gi0/0       10,30
Gi0/1       10,20,30

Port        Vlans in spanning tree forwarding state and not pruned
Gi0/0       10,30
Gi0/1       none
SW2(config-if)#
```

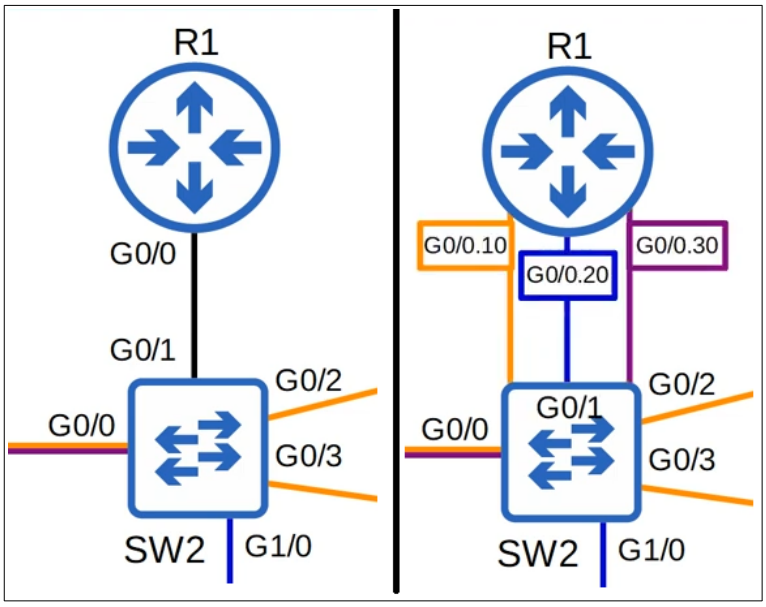
- Almost identical to **Gi0/0**, except we allowed **VLAN20** in addition to **VLAN10** and **VLAN30**.
- Now, both **Gi0/0** and **Gi0/1** are displayed in the output of the (**`show interfaces trunk`**) command.
- That's all for the Switch configurations for this lesson!

@0x0nullian

→ Router on a Stick (ROAS):-

- You may be wondering about the Router!?
- In the previous lecture, we used three separate interfaces for the connection from **SW2** to **R1**, and assigned a separate IP address to each one on **R1**. Each one served as the default gateway address for the PCs in each VLAN.
- However, now we are using only one physical connection between the two devices!
- So, we must use '***subinterfaces***' on **R1**.

- ***Router on a Stick***, is a bit of a strange name, but it's the name used for this method of **inter-VLAN Routing**.
- As there is only a single physical interface connecting the Router and the Switch, and it looks like a *Stick* on the network topology.
- So, in this case that one physical interface being used on **R1** to connect to **SW2** is **G0/0**. It is connected to **G0/1** on **SW2**.
- But, we can actually divide this one physical interface into three separate ***sub-interfaces***, which will allow us to perform ***inter-VLAN Routing*** with only one physical interface.
- So, it would look like this:



- **G0/0.10** for **VLAN10**, **G0/0.20** for **VLAN20**, **G0/0.30** for **VLAN30**.
- These three *logical* s**ub-interfaces** are really one physical interface, **G0/0**, which is connected to **SW2**'s **G0/1** interface, but they can operate like three separate interfaces.

NOTE: We don't need to do any additional configurations on **SW2**.

- We already configured **G0/1** as a trunk, and made sure that **VLAN**s **10**, **20**, and **30** are allowed.
- That's all we need to do on the Switch, configure the interface like a regular trunk.

→ **Router Configurations**:

→ First, make sure the interface is enabled with (**`no shutdown`**), as Router interfaces are disabled by default.

```
R1(config)#interface g0/0
R1(config-if)#no shutdown
R1(config-if)#
*Apr 15 04:29:49.681: %LINK-3-UPDOWN: Interface GigabitEthernet0/0, changed state to up
*Apr 15 04:29:50.682: %LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/
0, changed state to up
```

→ Next up, is the first *sub-interface*:

```
R1(config-if)#interface g0/0.10
R1(config-subif)#encapsulation dot1q 10
R1(config-subif)#ip address 192.168.1.62 255.255.255.192
```

- Notice how to enter *sub-interface* configuration mode, (**`interface g0/0.10`**)
- The *sub-interface* number doesn't have to match the VLAN number.
- However, it is highly recommended that they do match, to make it easier to understand.
- Next command is (**`encapsulation dot1q + n`**), followed by the VLAN number which is **`10`** in this case.
- This tells the Router to treat any arriving frames tagged with the specified VLAN number as if they arrived on this *sub-interface*.
- If a frame arrives tagged with **VLAN10**, **R1** will behave as if it arrived on interface **G0/0.10**.
- It will also tag all frames leaving this *sub-interface* with **VLAN10** using **802.1Q**.
- After encapsulation command, we simply assign the IP address to the *sub-interface* with the last usable IP address of the subnet.

→ Then we did the same thing with the other two *sub-interfaces*:

```
R1(config-subif)#interface g0/0.20
R1(config-subif)#encapsulation dot1q 20
R1(config-subif)#ip address 192.168.1.126 255.255.255.192
R1(config-subif)#interface g0/0.30
R1(config-subif)#encapsulation dot1q 30
R1(config-subif)#ip address 192.168.1.190 255.255.255.192
```

- If we confirmed with the (**`show ip interface brief`**) command, we can see that each of the *sub-interfaces* appears, as well as the physical interface, although the physical interface itself has no IP address assigned to it.

```
R1#show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     unassigned      YES NVRAM  up                    up
GigabitEthernet0/0.10  192.168.1.62    YES manual up                    up
GigabitEthernet0/0.20  192.168.1.126   YES manual up                    up
GigabitEthernet0/0.30  192.168.1.190   YES manual up                    up
GigabitEthernet0/1     unassigned      YES NVRAM  administratively down down
GigabitEthernet0/2     unassigned      YES NVRAM  administratively down down
GigabitEthernet0/3     unassigned      YES NVRAM  administratively down down
```

→ The Routing table:

```
R1#show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is not set

      192.168.1.0/24 is variably subnetted, 6 subnets, 2 masks
C        192.168.1.0/26 is directly connected, GigabitEthernet0/0.10
L        192.168.1.62/32 is directly connected, GigabitEthernet0/0.10
C        192.168.1.64/26 is directly connected, GigabitEthernet0/0.20
L        192.168.1.126/32 is directly connected, GigabitEthernet0/0.20
C        192.168.1.128/26 is directly connected, GigabitEthernet0/0.30
L        192.168.1.190/32 is directly connected, GigabitEthernet0/0.30
R1#
```

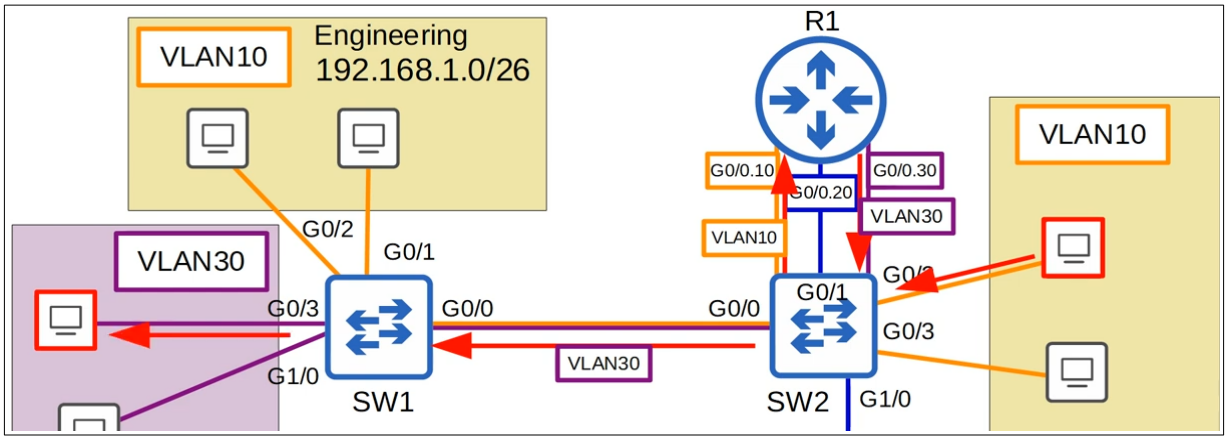
- Notice the connected and local routes are added just when IP addresses are added to regular physical interfaces.
- When **R1** sends frames out of these *sub-interfaces*, it adds the VLAN tag configured on the *sub-interface*.
- For example, if a packet arrives destined for the **192.168.1.64/26** subnet, it will send the packet out of it's **G0/0** interface tagged with **VLAN20**.

→ Let's review some important points:-

- **ROAS** is used to route between multiple VLANs using a single interface on the Router and Switch.
- The Switch interface is configured as a regular trunk.
- The Router interface is configured using *sub-interfaces*. We configure the VLAN tag and IP address on each *sub-interface*.
- The Router will behave as if frames arriving with a certain VLAN tag have arrived on the *sub-interface* configured with that VLAN tag.
- Finally, the Router will tag frames sent out of each *sub-interface* with the VLAN tag configured on the *sub-interface*.

→ How inter-VLAN routing works:-

- This PC in **VLAN10** is trying to reach this PC in **VLAN30**.
- The frame is sent to **SW2**. **SW2** sends the frame on its **G0/1** interface to **R1**, tagging it as being in **VLAN10**.
- **R1** receives it on its **G0/0** interface, identifying it as arriving on the **G0/0.10** *sub-interface* because of the **VLAN10** tag.
- The destination is in the subnet **192.168.1.128/26**, which is connected to **R1**'s **G0/0.30** *sub-interface*,
- So, it sends the frame out of it's **G0/0** interface.
- It tags it as **VLAN30** because that is what was configured on the **G0/0.30** *sub-interface*.
- **SW2** then forwards it to **SW1**, tagging it as **VLAN30** over the trunk.
- **SW1** then forwards the frame to the destination.



Day 18: VLANs pt.3:-

→ Native VLAN on a Router (ROAS):-

Best Practice: is to set the Native VLAN to an unused VLAN, as the Native VLAN feature can cause some security issues.

→ However, if you want to use the Native VLAN feature, Let' see how to use it on a Router:-
- The Native VLAN feature does have one benefit.
  Because frames in the Native VLAN aren't tagged, it's more efficient, each frame is smaller,
  so it allows the device to send more frames per second.
- In the previous video, we set the Native VLAN to **1001** on **SW1**'s **G0/0** interface, and **SW2**'s **G0/0** and **G0/1** interfaces.
- So, just for this demonstration, let's set them back to a used VLAN, **VLAN10** on all trunks:

```
SW1(config)#int g0/0
SW1(config-if)#switchport trunk native vlan 10
SW1(config-if)#
```

```
SW2(config)#int g0/0
SW2(config-if)#switchport trunk native vlan 10
SW2(config-if)#int g0/1
SW2(config-if)#switchport trunk native vlan 10
SW2(config-if)#
```

→ There are ***2 methods*** of configuring the *Native VLAN* on a Router:-
1. First up, we can use the command (**`encapsulation dot1q + "vlan-id" + native`**) on the Router interface.
- This tells the Router that this sub-interface belongs to the *Native VLAN*, and it will function just like the *Native VLAN* on a Switch.
- It will assume untagged frame belong to the *Native VLAN*, and frames sent in the *Native VLAN* will not be tagged.

2. The second, is to not use the sub-interface at all, but just configure the IP address for the *Native VLAN* on the physical interface of the Router.
- The (**`encapsulation dot1q`**) is not necessary in this case.

<u>Let's look at each option:-</u>

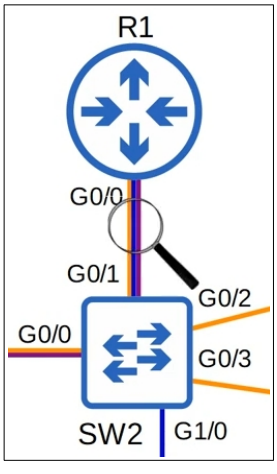
1- First, We will configure the first option:-
- On the **G0/0.10** interface, we configured (**`encapsulation dot1q 10 native`**)

<u>NOTE:</u> This is the complete topology from the previous lecture, so the IP address is already configured. The only change is that we added (**`native`**) to the (**`encapsulation dot1q`**) command.

```
R1(config)#int g0/0.10
R1(config-subif)#encapsulation dot1q 10 native
R1(config-subif)#
```

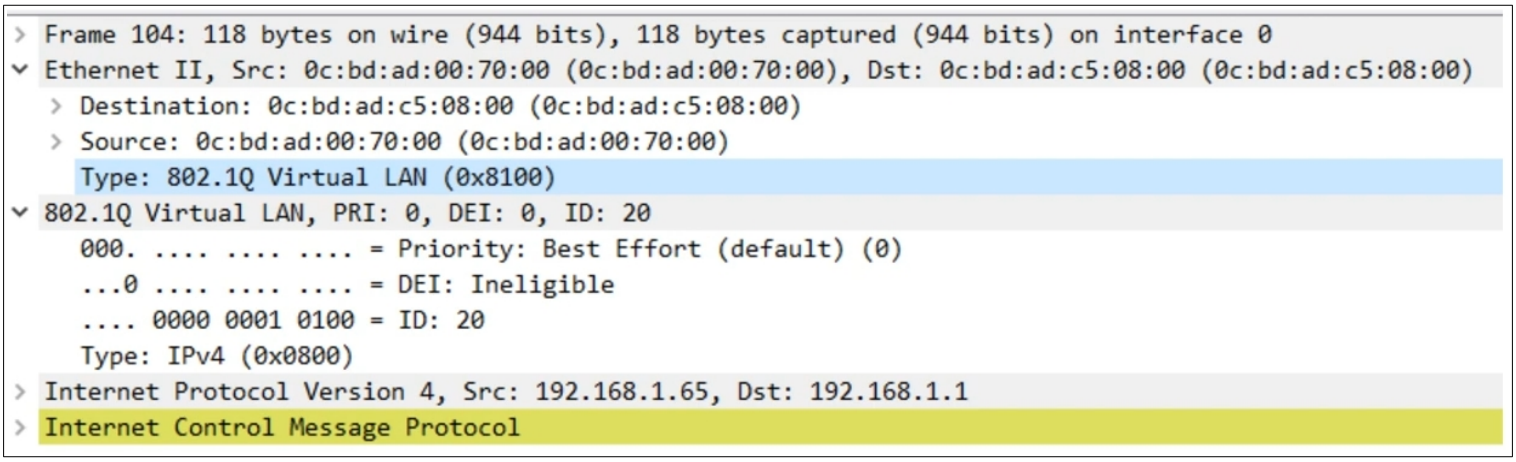
→ Wireshark Packet Capture:-

Let's take this opportunity to look at a Wireshark capture to demonstrate the *Native VLAN*.
- This PC in **VLAN20** has an IP address of **192.168.1.65**,
  and that PC in **VLAN10** has an IP address of **192.168.1.1**.
- We will use the Wireshark to monitor this connection between **R1** and **SW2**.
- Wireshark will capture all frames on this connection, in both directions,
  so we can take a look at what traffic is passing through.
- Let's send that ping. We will first look at the capture of the **ICMP echo request** message as it goes from **SW2** to **R1**.
- It ill be in **VLAN20**, and it's being sent to **R1** for *inter-VLAN routing*.

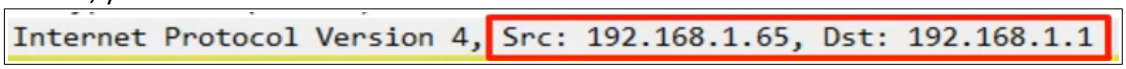


→ Wireshark Capture (SW2 → R1):-

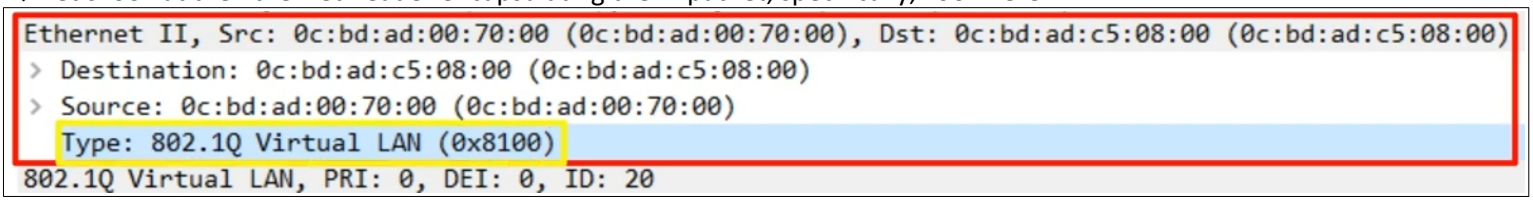
Here's the Wireshark capture for the **ICMP echo request** as it goes from **SW2** to **R1**:-



- First off, you can see the source and destination IP addresses here.

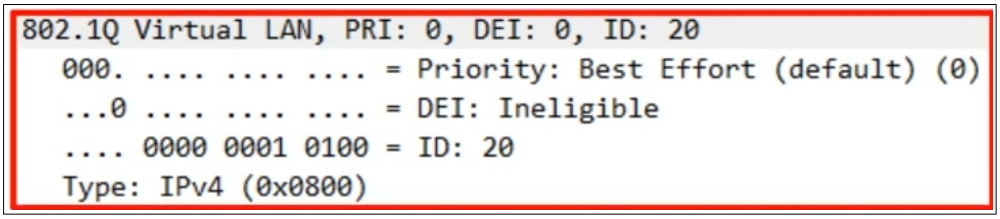


→ Let's look at the Ethernet header encapsulating the IP packet, specifically, Look here:



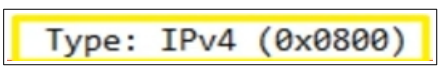
- Type: **802.1q** Virtual LAN, and notice the hexadecimal **8100** value.
- In the previous lecture, **dot1q** is inserted after the source MAC address field, and that is where TYPE field usually goes.
- This here is the '**TPID**' field of the **dot1q** tag.

Under it, these are the rest of the fields of the **802.1Q** tag:



- First is the **PCP**, Priority Code Point. It has a value of **0**, so no priority is given to this frame.
- Under it the **DEI**, Drop Eligible Indicator. A value of **0**, so it won't be dropped during times of network congestion.
- Next is the most important field, the **VID**, **VLAN ID**, which is **20**, as we would expect.
- The PC that sent the ping is in **VLAN20**m and it's not the *Native VLAN* so that's why this frame is tagged.

- Finally, under that is the normal TYPE field of the Ethernet header, indicating that an **IPv4** packet is encapsulated.



It normally comes after the source MAC address field, but now the **802.1Q** tag is between them.

→ Wireshark Capture (R1 → SW2):-

Let's look at the **ICMP echo request** going from **R1** to **SW2**:

- It will now be in **VLAN10**, because the destination is in **VLAN10**.
- **VLAN10** is configured as the *Native VLAN* on both **R1** and **SW2**, so let's see what different.

Here's he exact same **ICMP echo request**, the exact same Layer3 packet, as it is sent from **R1** to **SW2**.

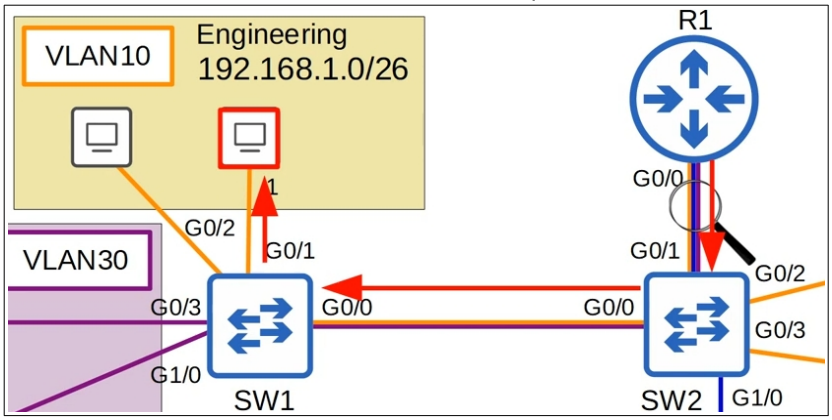
What's different??

```
> Frame 105: 114 bytes on wire (912 bits), 114 bytes captured (912 bits) on interface 0
v Ethernet II, Src: 0c:bd:ad:c5:08:00 (0c:bd:ad:c5:08:00), Dst: 0c:bd:ad:84:0a:00 (0c:bd:ad:84:0a:00)
  > Destination: 0c:bd:ad:84:0a:00 (0c:bd:ad:84:0a:00)
  > Source: 0c:bd:ad:c5:08:00 (0c:bd:ad:c5:08:00)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 192.168.1.65, Dst: 192.168.1.1
> Internet Control Message Protocol
```

- It has been encapsulated with a new Ethernet header, but this Ethernet header doesn't have a **dot1q** tag.
- This is the *Native VLAN* function at work.
- Both **R1** and **SW2** understand that untagged frames belong to **VLAN10**, so there is no need to tag each frame with **dot1q** tag.

→ Continue Native VLAN on a Router (ROAS):-

- That **ICMP echo request** will continue to the destination, untagged all the way because **VLAN10** is configured on all devices.
- When this PC in **VLAN10** sends the **ICMP echo reply**, it will be untagged until it reaches **R1**, which will then tag it in **VLAN20**, and send it back to the PC that sent the request.



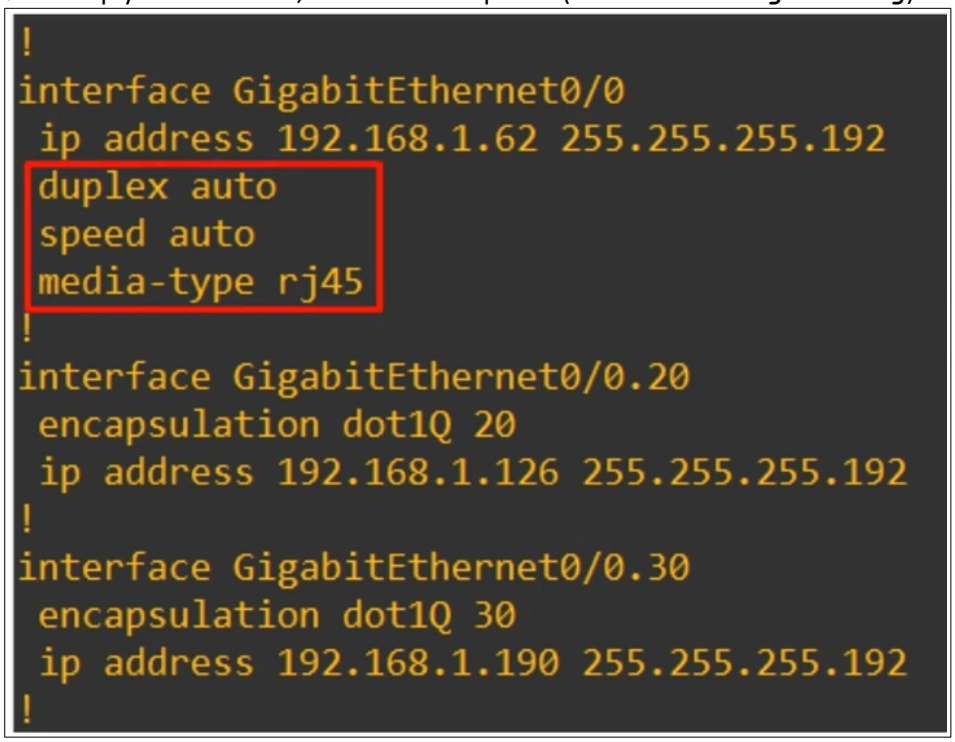
→ Configuring the Native VLAN on a Router:-

Let's look at the second method of configuring the *Native VLAN* on a Router:-

- Which is simply configuring the IP address on the Router's physical interface, no need for a sub-interface or the (`encapsulation dot1q`) command.
- Here's how to configure it:
  - First we used the command (`no interface g0/0.10`). This deletes the sub-interface
  - Then, we entered interface configuration mode from **G0/0**, and simply configured the appropriate IP address on the interface.

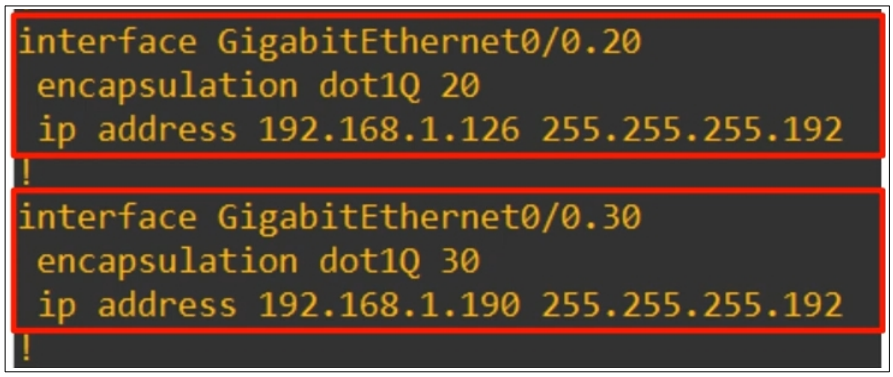
```
R1(config)#no interface g0/0.10
R1(config)#interface g0/0
R1(config-if)#ip address 192.168.1.62 255.255.255.192
R1(config-if)#
```

→ To help you visualize it, here is the output of (**`show running-config`**) for **G0/0** and its sub-interfaces:-



- First off, these commands here on the physical interface are there by default, we didn't configure them.

- The physical interface is configured normally with an IP address.
- This will be used for the *Native VLAN*, **VLAN10**.



- The other two sub-interfaces are just like we configured them in the previous video, with the (**`encapsulation dot1q`**) command and their own IP address.

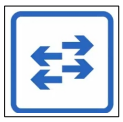
- This will function just like the first option we saw.
- **SW2** will send **VLAN10** packets in untagged frames to **R1**, and **R1** will send them in untagged frames also.

As we said before, the Best Practice is is that you just change the Native VLAN to an unused VLAN for security purposes.
But if you want to use the Native VLAN, it's important to know how to do it on a Router, so these are the two methods you can use.

@0x0nullian

→ Layer3 (MultiLayer) Switches:-

If we take a look at our Network topology, we will see that we have one Router, and two Switches.
Or we should say, two Layer2 Switches!



This is the icon we've been using for regular Layer2 Switches.

This is the icon we will use for what is called a Layer3 Switch, also known as a Multilayer Switch.

→ What exactly a MultiLayer Switch does:-

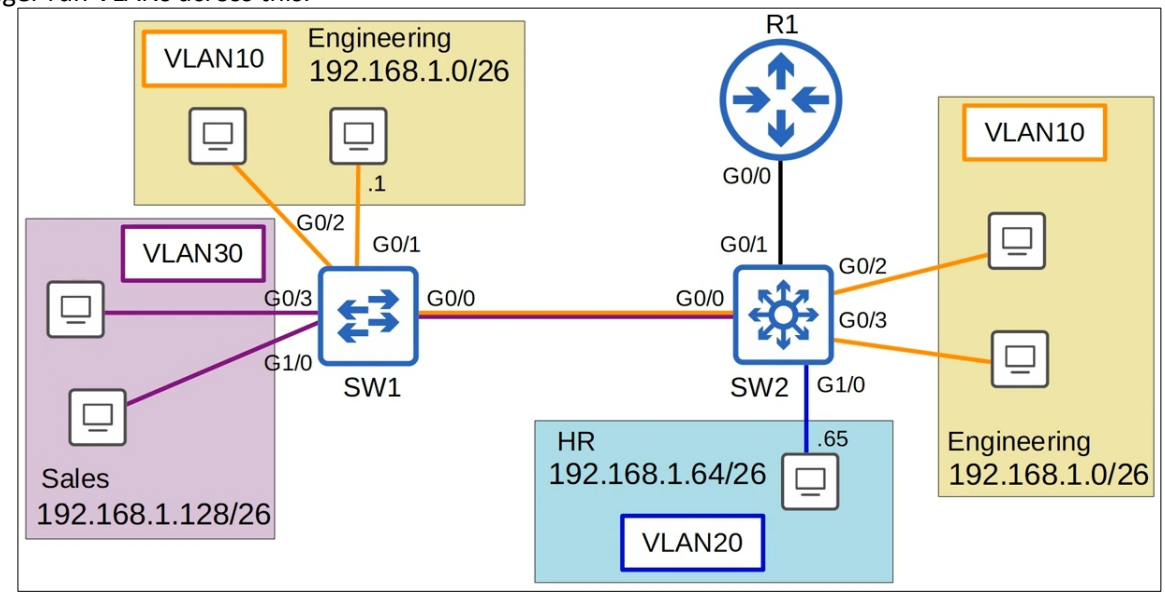
- A MultiLayer Switch is capable of both *Switching* and *Routing*.
- It is '*Layer3 aware*'.
  A regular Layer2 Witch is NOT Layer3 aware, It doesn't at all about IP addresses or anything above Layer2.
  It only cares about Layer2 information like MAC addresses.
- We can assign IP addresses to its interfaces like a Router.
  Previously, We haven't assign any IP addresses to Switches, only Routers.
  With a Layer3 Switch, we can configure '*routed ports*', which function like an interface on a Router.
- Not just physical interfaces, but we can also create virtual interfaces for each VLAN,
  and assign assign IP addresses to those interfaces.
  These are not separate physical interfaces, but virtual interfaces in the software of the Switch that can be used to route traffic at Layer3.
- We can configure routes, like static routes, on a Layer3 Switch, just like a Router.
- It can be used for inter-VLAN routing.

- We've looked at two methods of inter-VLAN routing, the first was in Day16, was using one connection for each VLAN between the Router and the Switch.
  This works, but if you have many VLANs you probably won't have enough interfaces on your Router.
- The Second method was Router on a Stick, which uses a single trunk connection which carries traffic from all VLANs between the Switch and the Router for inter-VLAN routing.
- This is efficient in terms of the number of interfaces, just one, but in a busy network all of he traffic going to the Router and back to the Switch can cause network congestion!

So, in Large Networks, a Multi-Layer Switch is the preferred method of inter-VLAN routing.

→ How Multi-Layer Switch works:-

- Here's the previous topology, but let us replace **SW2** with a *Multi-Layer* Switch.
- And let's make one more change! We've replaced the trunk link between **SW2** and **R1** with a *point-to-point* Layer3 link, we will no longer run VLANs across this.



→ For review, when we used Router on a Stick for inter-VLAN routing, traffic being routed between VLANs was sent to **R1** first, and then sent back to **SW2**, and then forwarded to the destination

→ For example, if this PC in **VLAN20** wants to ping this PC in **VLAN10**, the traffic would follow a path like this:
- From the PC to **SW2**, from **SW2** to **R1**, tagged in **VLAN20**, from **R1** to **SW2**, tagged in **VLAN10**, from **SW2** to **SW1**, tagged in **VLAN10**, and finally to the destination :D

→ However, **SW2** is a *Multi-Layer Switch*.
- It doesn't have to send the traffic to **R1** for inter-VLAN routing.
- It can do that with something called '***Switch Virtual Interfaces***'.

→ Inter-VLAN Routing via SVIs:-

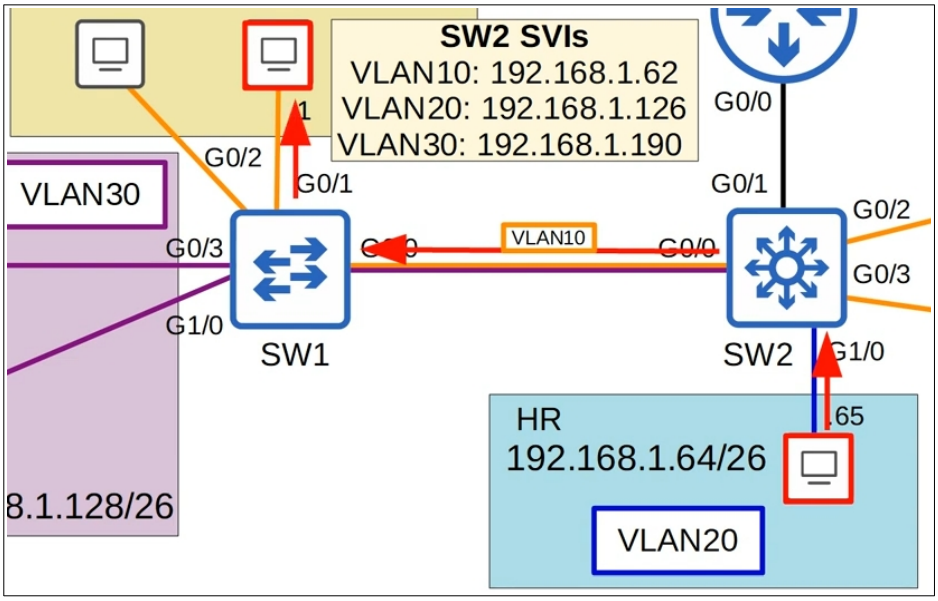
→ Switch Virtual Interfaces:-
- **SVI**s are the Virtual interfaces you can assign IP addresses to in a Multi-Layer Switch.
- Configure each PC to use the **SVI** (NOT the Router) as their gateway address.
- When using **ROAS**, the Router was used as the PC's gateway. This time, we will use the Switch's **SVI**s instead.
- To send traffic to different subnets/VLANs, the PCs will send traffic to the Switch, and the Switch will route the traffic.

→ These are the **SVI**s we configured on **SW2**.

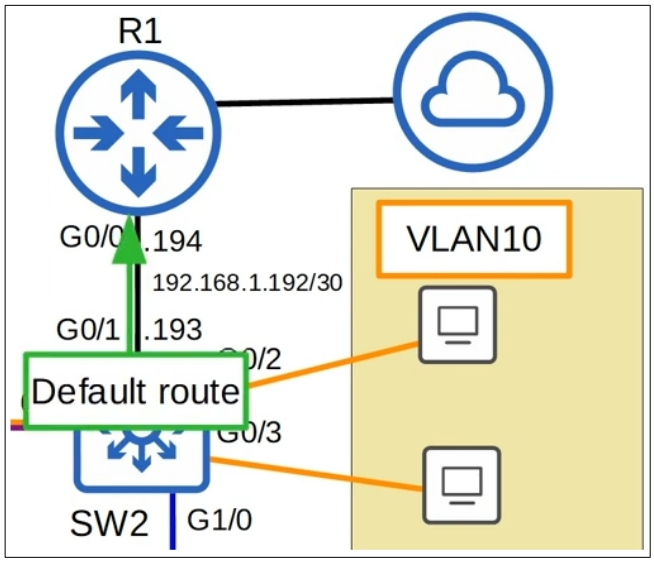
SW2 SVIs
VLAN10: 192.168.1.62
VLAN20: 192.168.1.126
VLAN30: 192.168.1.190

- These are the same IP addresses we configured on **R1** when doing Router on a Stick, the last usable IP address in each subnet.
- So, these are already configured on each PC as their gateway addresses, so there's no need to change the PC configurations.

→ So, let's take a look at the path the traffic between the last two PCs takes this time:-
- The frame is sent from the PC in **VLAN20**, arrives at **SW2**.
- The destination is in the **192.168.1.0/26** subnet.
- **SW2** now has its own routing table, so it looks up the destination in the routing table, and sees that the destination is connected to its **VLAN10 SVI**.
- So, the traffic is now routed to **VLAN10**.
- If **SW2** doesn't have the destination MAC address in its MAC address table, it will flood the frame to all **VLAN10** interfacs.
- But, let's assume it has already learned the MAC address, so it forwards it to **SW1** over its trunk interface, tagged as **VLAN10**.
- **SW1** then forwards it to the destination!



→ What if the hosts want to reach destinations outside of the LAN??
- For example, we've added a Cloud connected to **R1** to represent the Internet.
- Because **SW2** is their default gateway, any packets destined outside of their subnet will be sen to **SW2**.
- But our previous Router on a Stick configurations for the connection between **SW2** and **R1** will no longer work!
- In addition to configuring virtual interfaces, **SVI**s, on Multi-Layer Switches, we can also configure their physical interfaces to operate like a Router interface, rather than a switch port.
- So, we can assign the subnet **192.168.1.192/30** for this *point-to-point* link between **SW2** and **R1**, with **SW2**'s **G0/1** interface having an IP address of **192.168.1.193**, and **R1**'s **G0/0** interface having an IP address of **192.168.1.194**.
- Then, we configure a default route on **SW2** pointing toward **R1**, so all traffic destined outside of the LAN will be sent to **R1**.



→ Let's get into the configurations, starting first with the *point-to-point* link between **SW2** and **R1**, and then the **SVI**s on **SW2**.
- First off, remove **R1**'s Router on a Stick configurations and configure that new IP address on **G0/0**.
- We delete each sub-interface with the command (**`no interface g0/0.10`**), (**`no interface g0/0.20`**), (**`no interface g0/0.30`**).

- Then, we use the command (**`default interface g0/0`**), to reset **G0/0** to its default settings.

- After that, we used (**`show ip interface brief`**) to check the interfaces.
- Notice the status of the interfaces, it says **deleted**.
- Although we've successfully deleted the sub-interfaces, they will remain here with a '**deleted**' status unless we reload the Router.

```
R1(config)#no interface g0/0.10
R1(config)#no interface g0/0.20
R1(config)#no interface g0/0.30
R1(config)#default interface g0/0
Interface GigabitEthernet0/0 set to default configuration
R1(config)#do show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         unassigned      YES NVRAM  up                    up
GigabitEthernet0/0.10      unassigned      YES manual deleted               down
GigabitEthernet0/0.20      unassigned      YES manual deleted               down
GigabitEthernet0/0.30      unassigned      YES manual deleted               down
GigabitEthernet0/1         unassigned      YES NVRAM  administratively down down
GigabitEthernet0/2         unassigned      YES NVRAM  administratively down down
GigabitEthernet0/3         unassigned      YES NVRAM  administratively down down
R1(config)#
```

→ Then we enter interface configuration mode for **G0/0** and configure the new IP address, with a **/30** subnet mask.
- We use the (**`show ip interface brief`**) again, and we can see that the new IP address has been successfully configured.

```
R1(config)#interface g0/0
R1(config-if)#ip address 192.168.1.194 255.255.255.252
R1(config-if)#do show ip interface brief
Interface                  IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0         192.168.1.194   YES manual up                    up
GigabitEthernet0/0.10      unassigned      YES manual deleted               down
GigabitEthernet0/0.20      unassigned      YES manual deleted               down
GigabitEthernet0/0.30      unassigned      YES manual deleted               down
GigabitEthernet0/1         unassigned      YES NVRAM  administratively down down
GigabitEthernet0/2         unassigned      YES NVRAM  administratively down down
GigabitEthernet0/3         unassigned      YES NVRAM  administratively down down
R1(config-if)#
```

→ Now let's look at the Switch's side of the *point-to-point* connection.
- First, we reset the **G0/1** interface to its default setting with the (**`default interface g0/1`**) command.
  Because it was configured as a trunk for Router on a Stick because of the previous lab.

- Next up is a very important command, one you must not forget.
  (**`ip routing`**), enables Layer3 routing on the Switch. It lets it build its own routing table like a Router.
- If you forget this command, your inter-VLAN routing will not work.

- Next up is another important command (**`no switchport`**) on the interface.
  This command configures the interface as a '**routed port**' (Layer3 port, not a Layer2/switch-port)
  Now we will be able to assign an IP address to it.

- So, we assigned **192.168.1.193/30**, and used (**`do show ip interface brief`**), and as we can see the IP address is assigned to it just like a Router interface.

```
SW2(config)#default interface g0/1
Interface GigabitEthernet0/1 set to default configuration
SW2(config)#ip routing
SW2(config)#interface g0/1
SW2(config-if)#no switchport
SW2(config-if)#ip address 192.168.1.193 255.255.255.252
SW2(config-if)#do show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     unassigned      YES unset  up                    up
GigabitEthernet0/2     unassigned      YES unset  up                    up
GigabitEthernet0/3     unassigned      YES unset  up                    up
GigabitEthernet0/1     192.168.1.193   YES manual up                    up
```

→ Last up is the default route pointing to **R1**:
- As we've already shown in a previous video, the command is (**`ip route 0.0.0.0 0.0.0.0 192.168.1.194`**), the next-hop is the **R1**.
- Then we used (**`do show ip route`**) to confirm, and we can see that **SW2** now has a routing table, with a default route pointing to **R1**, and **connected** and **local** routes for the routed interface we configured.

```
SW2(config-if)#exit
SW2(config)#ip route 0.0.0.0 0.0.0.0 192.168.1.194
SW2(config)#do show ip route
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override

Gateway of last resort is 192.168.1.194 to network 0.0.0.0

S*    0.0.0.0/0 [1/0] via 192.168.1.194
      192.168.1.0/24 is variably subnetted, 2 subnets, 2 masks
C        192.168.1.192/30 is directly connected, GigabitEthernet0/1
L        192.168.1.193/32 is directly connected, GigabitEthernet0/1
SW2(config)#
```

- One additional command to confirm (**`show interfaces status`**), which we used in a previous video on Ethernet Switching.
- Notice that, in the VLAN column, instead of a VLAN number **G0/1** displays '**routed**'.

```
SW2#show interfaces status

Port      Name               Status       Vlan       Duplex  Speed Type
Gi0/0                        connected    trunk        auto   auto unknown
Gi0/2                        connected    10           auto   auto unknown
Gi0/3                        connected    10           auto   auto unknown
Gi0/1                        connected    routed       auto   auto unknown
Gi1/0                        connected    20           auto   auto unknown
Gi1/1                        connected    1            auto   auto unknown
Gi1/2                        connected    1            auto   auto unknown
Gi1/3                        connected    1            auto   auto unknown
```

→ Configure **SVI**s on **SW2**:
- **SVI** configuration is very simple.
- Here's the configurations, use the command (**interface vlan10**), for example, to create an **SVI** for **VLAN10** and configure it.
- Then assign an IP address, and use (**`no shutdown`**) to enable it.
- **SVI**s are *shutdown* by default, so remember to use (**`no shutdown`**) command to enable them.
- Then we repeated the process for **VLAN20** and **VLAN30**, and that's all it it to configure an **SVI**. Very simple!

```
SW2(config)#interface vlan10
SW2(config-if)#ip address 192.168.1.62 255.255.255.192
SW2(config-if)#no shutdown
SW2(config-if)#interface vlan20
SW2(config-if)#ip address 192.168.1.126 255.255.255.192
SW2(config-if)#no shutdown
SW2(config-if)#interface vlan30
SW2(config-if)#ip address 192.168.1.190 255.255.255.192
SW2(config-if)#no shutdown
```

→ Just to demonstrate one problem we might encounter, we created another **SVI** for a VLAN that doesn't exist on the Switch, **VLAN40**, and assigned an IP address, **40.40.40.40/24**.

- We also made sure to enable it with (**`no shutdown`**).

```
SW2(config-if)#interface vlan40
SW2(config-if)#ip address 40.40.40.40 255.255.255.0
SW2(config-if)#no shutdown
SW2(config-if)#do show ip interface brief
Interface              IP-Address      OK? Method Status                Protocol
GigabitEthernet0/0     unassigned      YES unset  up                    up
GigabitEthernet0/2     unassigned      YES unset  up                    up
GigabitEthernet0/3     unassigned      YES unset  up                    up
GigabitEthernet0/1     192.168.1.193   YES manual up                    up
```

→ However, look at the **SVI** itself:

- It is **down/down**. Why???
- Well, it's because the VLAN doesn't exist on the Switch.

```
Vlan10                 192.168.1.62    YES manual up                    up
Vlan20                 192.168.1.126   YES manual up                    up
Vlan30                 192.168.1.190   YES manual up                    up
Vlan40                 40.40.40.40     YES manual down                  down
```

→ The conditions required for an SVI to be up/up:

1. The VLAN must exist on the Switch.
   When you assign an access port to a VLAN, if the VLAN doesn't yet exist on the Switch, it will be automatically created.
   However, if you create an **SVI** for a non-existed VLAN, the Switch WILL NOT automatically create that VLAN.
2. The Switch must have at least one access port in the VLAN in an **up/up** state, AND/OR one trunk port tat allows the VLAN that is in an **up/up** state.
   For example, in the topology we're using, **SW2** has hosts connected in **VLAN10** and **VLAN20**, so their **SVI**s can go **up**.
   There are no connected hosts in **VLAN30**, however it has a trunk port, **G0/0**, which allows **VLAN30** over it.
   So, **VLAN30**'s **SVI** is **up** as well.
3. The VLAN must not be shutdown.
   Note that this is NOT the **SVI**, but the VLAN itself.
   We can enter the VLAN configuration mode, and use the **`shutdown`** command to disable the VLAN.
   If we do this, the **SVI** for that VLAN can't become **up/up**.
   We can't do this command in Packet Tracer, So we will need a real Cisco Switch if we want to try this one out.
4. The **SVI** must not be shutdown (**SVI**s are disabled by default).
   If the **SVI** itself is shutdown, it obviously won't be **up/up**, so make sure to use the (**`no shutdown`**) command after creating an **SVI**, because they are shutdown by default.

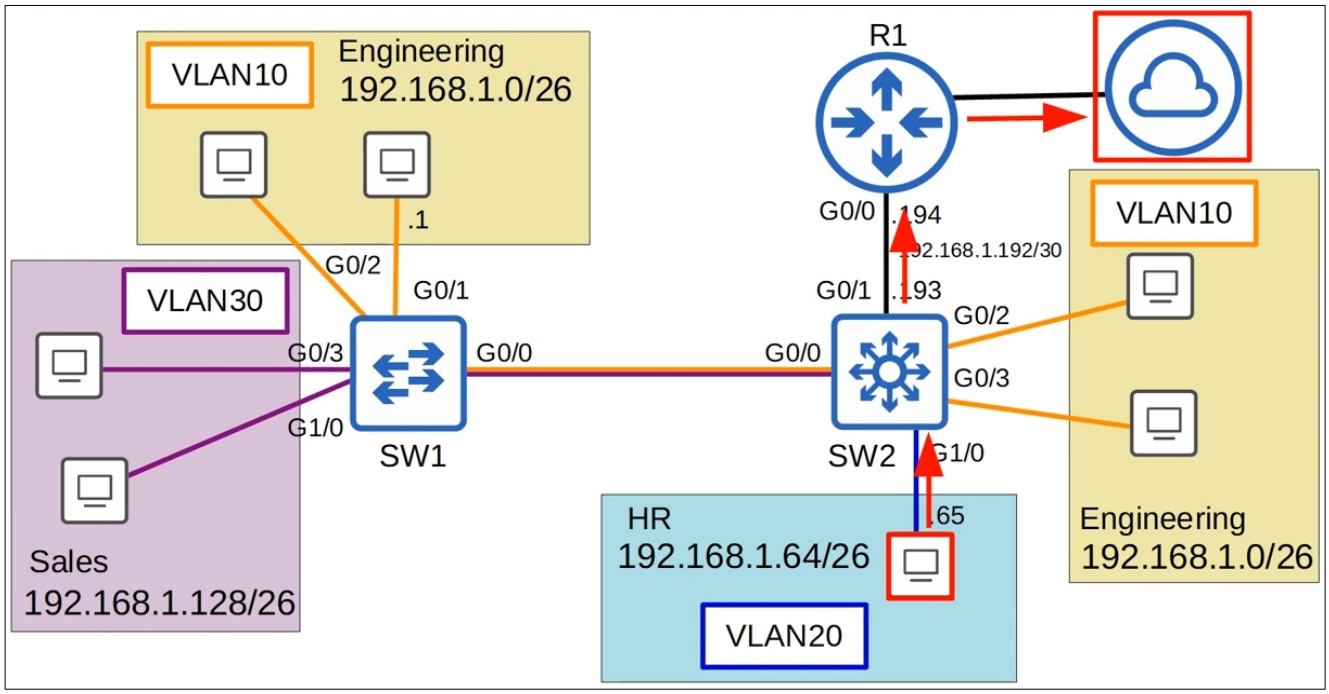
→ We used the (**`do show ip route`**) command again, and we can see **connected** and **local** routes have been added to the route table for the **SVI**s we created, all shown as directly connected to the **SVI** for each VLAN.

```
Gateway of last resort is 192.168.1.194 to network 0.0.0.0

S*    0.0.0.0/0 [1/0] via 192.168.1.194
      192.168.1.0/24 is variably subnetted, 8 subnets, 3 masks
C        192.168.1.0/26 is directly connected, Vlan10
L        192.168.1.62/32 is directly connected, Vlan10
C        192.168.1.64/26 is directly connected, Vlan20
L        192.168.1.126/32 is directly connected, Vlan20
C        192.168.1.128/26 is directly connected, Vlan30
L        192.168.1.190/32 is directly connected, Vlan30
C        192.168.1.192/30 is directly connected, GigabitEthernet0/1
L        192.168.1.193/32 is directly connected, GigabitEthernet0/1
SW2(config-if)#
```

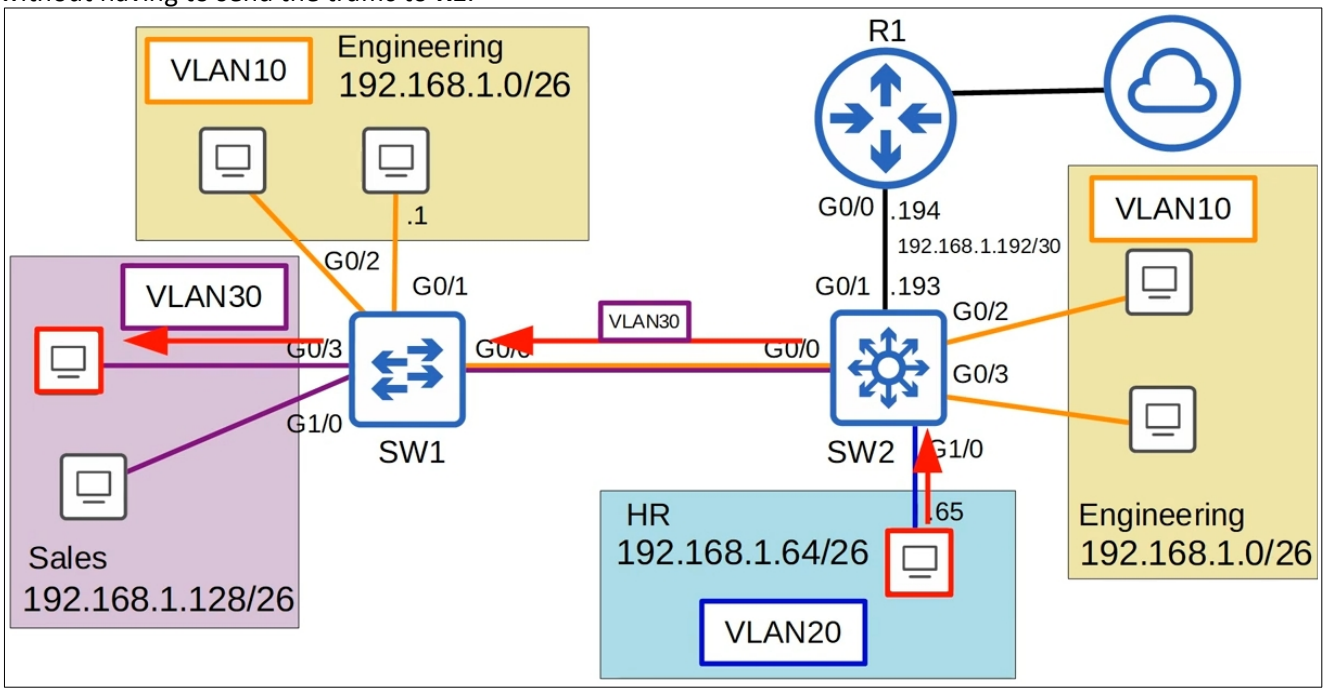
Okay, so our configurations are all done ;)

→ So, if one of our PCs wants to reach a destination outside of the LAN, it will be sent to **SW2**, which will send it to **R1**, which will take care of it from there.



We didn't actually configure any routes on **R1** in this lab, We're just focusing on inter-VLAN routing t this point

→ So, if one of our PCs wants to reach a destination in the LAN, but in a different subnet and VLAN, **SW2** will do the inter-VLAN routing without having to send the traffic to **R1**.



@0x0nullian

Day 19: DTP & VTP:-

In this Day we will cover two Cisco proprietary protocols,
DTP, aka, **Dynamic Trunking Protocol**, and **VTP**, **VLAN Trunking Protocol**.
They were developed by Cisco and they only run on Cisco devices.

> DTP and VTP were removed from the CCNA exam topics list for the new exam (200-301). However, it's important to know their function, and you may still get questions about them on the exam even though they are not on the topics list.

→ DTP (Dynamic Trunking Protocol):-

→ **DTP** is a Cisco proprietary protocol that allows Cisco Switches to negotiate the status of their switch-ports to be either *access ports* or *trunk ports*, without manually configuring them.
It allows Switches to dynamically determine their interface status (***access*** or ***trunk***) without manual configuration.
Basically, two Cisco Switches connected together can form a *trunk* but otherwise the interface will automatically be an access port.

→ **DTP** is enabled by default on all Cisco Switch interfaces.

→ So far, we've been manually configuring switch-ports using these commands:
(**`switchport mode access`**) OR (**`switchport mode trunk`**)
If we use **DTP**, we don't need to enter these commands.

→ For security purposes, manual configuration is ***recommended***. **DTP** should be disabled on all switch-ports.
It can be exploited by Attackers.

→ Let's go to the CLI :-
We're in interface configuration mode on a Cisco Switch, and we entered (**`switchport mode ?`**)
We can see the *ACCESS* and *TRUNK* options we used before, but the one we're going to look at now is this one, ***DYNAMIC***.

```
SW2(config-if)#switchport mode ?
  access        Set trunking mode to ACCESS unconditionally
  dot1q-tunnel  set trunking mode to TUNNEL unconditionally
  dynamic       Set trunking mode to dynamically negotiate access or trunk mode
  private-vlan  Set private-vlan mode
  trunk         Set trunking mode to TRUNK unconditionally
```

It says "*Set trunking mode to dynamically negotiate access or trunk mode*". That's **DTP**!

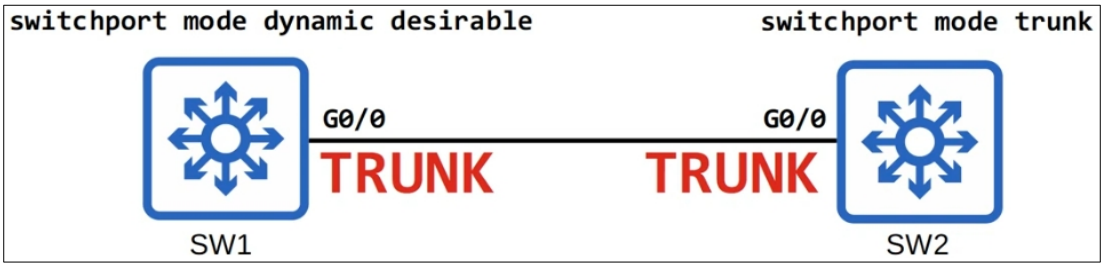
So, we entered (**`switchport mode dynamic ?`**)
There are two options, ***AUTO*** and ***DESIRABLE***.

```
SW2(config-if)#switchport mode dynamic ?
  auto       Set trunking mode dynamic negotiation parameter to AUTO
  desirable  Set trunking mode dynamic negotiation parameter to DESIRABLE
```

As from the description, it doesn't really explain their function!

→ Let's see how the *DYNAMIC DESIRABLE* mode works:-
- A switchport in *DYNAMIC DESIRABLE* mode will actively try to form a trunk from other Cisco Switches.
- It will form a trunk if connected to another switchport in the following modes:
    * **`switchport mode trunk`**
    * **`switchport mode dynamic desirable`**
    * **`switchport mode dynamic auto`**

- **SW1** and **SW2** are connected via their **G0/0** interfaces.
- **SW1**'s **G0/0** interface is in *DYNAMIC DESIRABLE* mode. **SW2**'s **G0/0** interface is manually configured as a trunk.



- So, these two Switches will both agree to operate as trunks.

- Here's a new command (**`show interface g0/0 switchport`**).

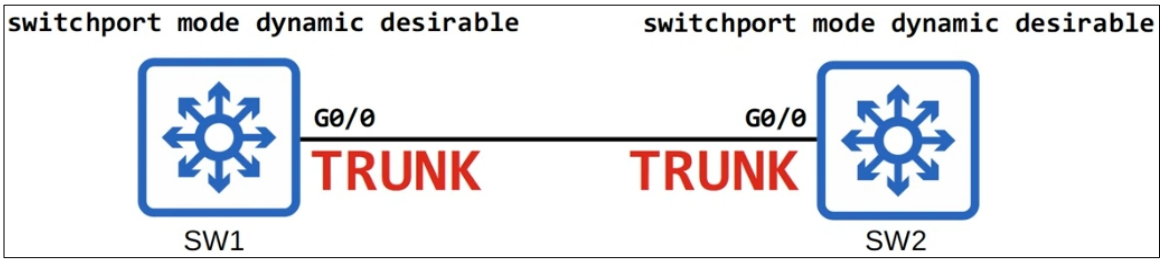
```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

- It says **Switchport: Enabled**, because it is a Layer port.
- If we want to configure a routed port with the (**`no switchport`**) command, this would display differently.
- The **Administrative Mode: dynamic desirable**. Administrative Mode is what we actually configured on the interface.
- **Operational Mode: trunk**. Displays whether it is a *trunk* or *access* port.
- Because **SW2**'s interface is a *trunk*, **SW1**'s interface become a *trunk* as well, thanks to **DTP** negotiation!

- Here on **SW2** we can see that both the administrative mode and operational mode are ***trunk***.

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

→ Now both interfaces are configure in *DYNAMIC DESIRABLE* mode.
So, they will both for a *trunk*.



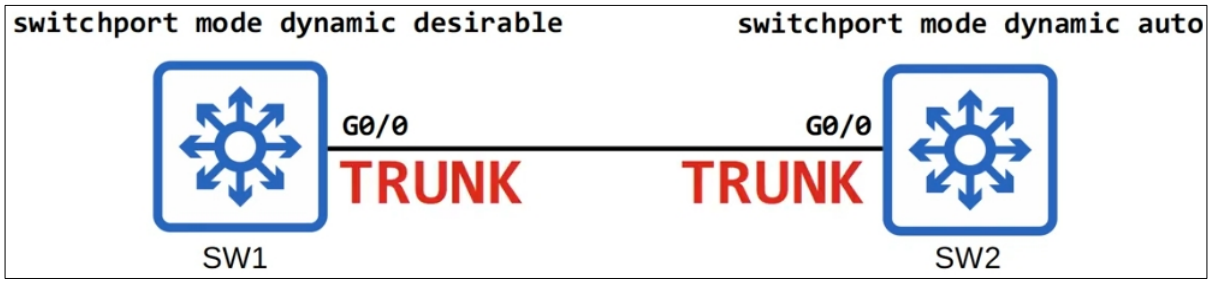
- The output of (**`show interfaces g0/0 switchport`**) is the same on **SW1**, and this time **SW2** also has an administrative mode of '*DYNAMIC DESIRABLE*'.
- But once again the operational mode is *TRUNK*, because both Switches are actively using **DTP** to try to form a trunk!

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

- Even if manually configured as a trunk, an interface still sends **DTP** frames out of the interface.

→ This time **SW2**'s interface is configured in *DYNAMIC AUTO* mode.
- A switchport in dynamic auto mode does not try to form a trunk. It's more passive!
- It will tell **SW1** 'If you want to form a trunk, I'll form a trunk, but I'm not going to actively form a trunk with you!'
- However, because **SW1** is in *DYNAMIC DESIRABLE* mode, once again a trunk will be formed!

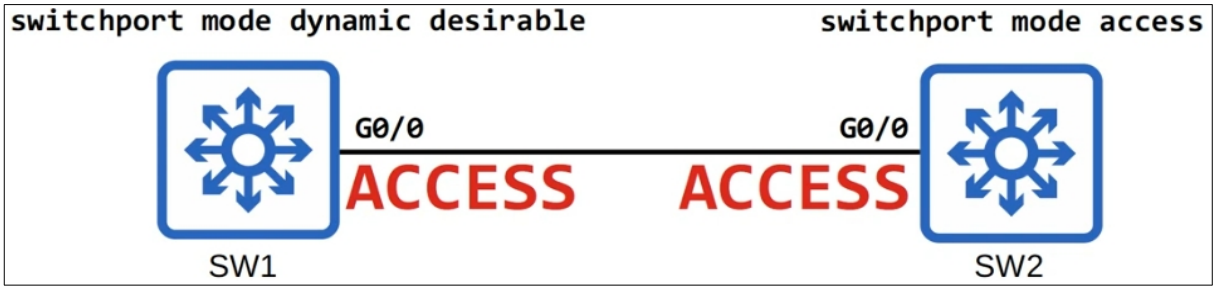


- **SW1**'s (**`show interfaces g0/0 switchport`**) output is the same, and **SW2**'s show an administrative mode of dynamic auto, and again and operational mode of a trunk!

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: trunk
```

→ One more example:
- **SW2**'s interface is now manually configured as an *ACCESS PORT* with the (**`switchport mode access`**).
- **SW1** is actively trying to form a trunk, but since **SW2** is manually configured in access mode, the trunk will not form!!
  and both will operate as access ports in the default VLAN, which is **VLAN1**.



- The output of (**`show interfaces g0/0 switchport`**) on **SW1** now shows an operational mode of *static access*.
- On **SW2**'s **G0/0** interface both the administrative and operational modes are *static access*.

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: static access
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
```

→ Okay, Now we've learned about dynamic desirable mode and seen that an interface in dynamic desirable mode will use **DTP** negotiation to form a trunk if the connected interface on the other device is in ***trunk***, ***dynamic desirable***, or ***dynamic auto*** mode.

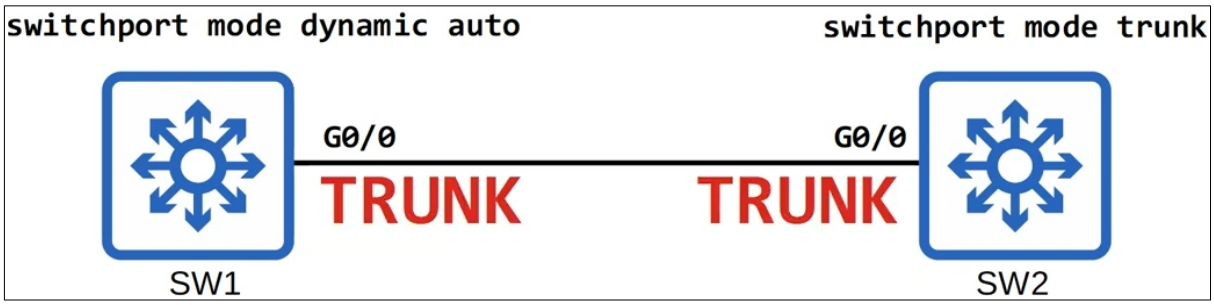
→ However, if the other interface is in access mode, it will not form a trunk, it will be an access port.

- ***Static Access*** means an access port that belongs to a single VLAN that doesn't change (unless you configure a different VLAN).
- There are also '***Dynamic Access***' ports, in which a server automatically assigns the VLAN
  depending on the MAC address of the connected device.
  <u>This is out of the scope of the CCNA.</u>

→ Let's Look at *DYNAMIC AUTO* mode:-

- A switchport in *DYNAMIC AUTO* mode will NOT actively try to form a trunk with other Cisco Switches.
- However, It will form a trunk if the Switch connected to it is actively trying to form a trunk.
- It will form a trunk with a switchport in the following modes:
    * **`switchport mode trunk`**
    * **`switchport mode dynamic desirable`**

→ So,
- **SW1**'s **G0/0** interface is configured in dynamic auto mode, and **SW2**'s is manually configured as a trunk.
- Therefore, **DTP** negotiation will cause them to form a trunk link.



- Here we can see the administrative mode of dynamic auto and operational mode of trunk.
- Whereas on **SW2** both are trunk.

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: trunk
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

-- Now we already so what happens when a switchport in dynamic auto mode is connected to a switchport in dynamic desirable mode, they form a link!

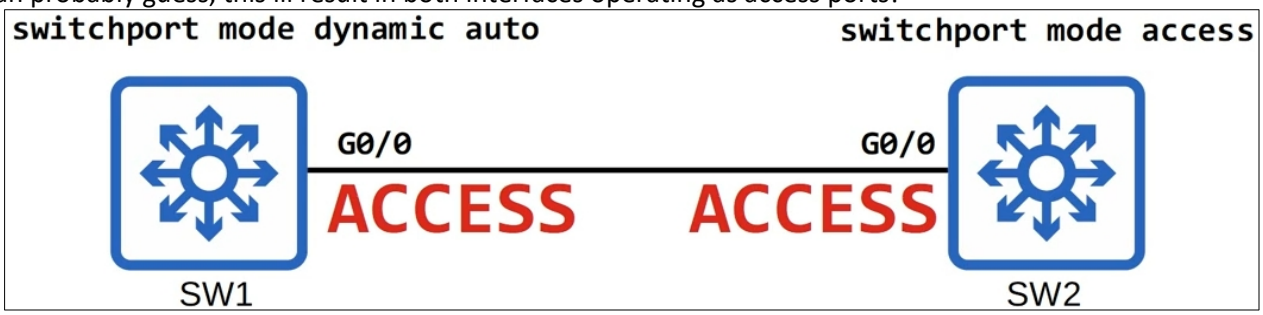
→ Let's look at two switchports in dynamic auto mode:-
- Neither is actively trying to form a link, so both operate as *access ports* in the default VLAN, **VLAN1**.

- They have the same output for the (**`show interfaces g0/0 switchport`**) command, administrative mode of dynamic auto and operational mode of *static access*.

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
```

→ Next up, dynamic auto and access mode:-
- As you can probably guess, this ill result in both interfaces operating as access ports!



- Here's the output of (**`show interfaces g0/0 switchport`**) on each Switch.

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic auto
Operational Mode: static access
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
```

→ What happens if a manually configured trunk is connected to a manually configured access port??
- Well, since both are manually configured, they are ***forced*** to operate ***matched*** in trunk and access modes!
- However, this configuration does not work! It should result in an error! And traffic will not pass between these Switches.

- Here's the output of (**`show interfaces g0/0 switchport`**) command for each.
- However, this configuration does not work! It should result in an error! And traffic will not pass between these Switches.

```
SW1#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: trunk
Operational Mode: trunk
```

```
SW2#show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: static access
Operational Mode: static access
```

→→ Here's a ***chart*** summarizing the resulting operational mode given two administrative modes:

Administrative Mode	Trunk	Dynamic Desirable	Access	Dynamic Auto
Trunk	Trunk	Trunk	X	Trunk
Dynamic Desirable	Trunk	Trunk	Access	Trunk
Access	X	Access	Access	Access
Dynamic Auto	Trunk	Trunk	Access	Access

- For example, a switchport in dynamic desirable mode will form a trunk with an interface in any administrative modes except access mode.

IMPORTANT POINT: **DTP** will not form a trunk with a Router, PC, etc. The switch-port will be in access mode.
→ So, if you want to configure router on a stick, you must manually configure the interface connected to the Router as a trunk, You cannot put it in dynamic desirable mode and expect it to become a trunk.

→ A few more points about DTP:-

→ On older Switches, **`switchport mode dynamic desirable`** is the default administrative mode.
They will actively try to form trunk links.

→ On newer Switches, **`switchport mode dynamic auto`** is the default administrative mode.

→ You can disable **DTP** negotiation on in interface with this command (**`switchport nonegotiate`**).
If you use this command, the interface will stop sending **DTP** negotiation frames.

→ Configuring an access port with **`swithcport mode access`** also disables **DTP** negotiation on an interface.
- It will also stop sending **DTP** frames.
- If you manually configure an interface in trunk mode, however, it does not stop it from sending **DTP** frames, unless you also issue the (**`switchport nonegotiate`**) command.

→ It is recommended to disable **DTP** on all switch-port and manually configure them as *access* or *trunk* ports.

→ Trunk Encapsulation Negotiation:-

- Switches that support both **802.1Q** and **ISL** trunk encapsulation can use **DTP** to negotiate the encapsulation they will use.
- This negotiation is enabled by default, as the default trunk encapsulation mode is:
  `switchport trunk encapsulation negotiate`.
- If you want to manually configure a trunk interface on a Switch that supports both **802.1Q** and **ISL**,
  You must first change the encapsulation mode to **dot1q** or **ISL**, you can't leave it in negotiation mode.
- **ISL** is favored **dot1q**, so if both Switches support **ISL** it will be selected.
- **DTP** frames are sent in **VLAN1** when using **ISL**, or in the native VLAN when using **802.1Q** (the default native VLAN is **VLAN1**).
  Unless you changed the native VLAN they will be sent in **VLAN1** for **802.1Q** also.

→ To show this negotiation of trunking encapsulation, here is a little more of the output from (**`show interfaces switchport`**):

- We set the interfaces on both Switches to dynamic desirable mode so they would form a trunk.
- Notice that the default trunking encapsulation mode of **negotiate** results in an operational trunking encapsulation of **ISL**.
- By the way, this field down here, negotiation of trunking, shows whether **DTP** is enabled,
  whether the interface is sending **DTP** frames or not.
- If the interface is in dynamic desirable mode, dynamic auto, or trunk mode, this will be **On**.
- If it's in access mode, or if you use the (**`switchport nonegotiate`**) command, this will be **Off**.

```
SW1(config-if)#switchport mode dynamic desirable
SW1(config-if)#do show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: isl
Negotiation of Trunking: On
```

```
SW2(config-if)#switchport mode dynamic desirable
SW2(config-if)#do show interfaces g0/0 switchport
Name: Gi0/0
Switchport: Enabled
Administrative Mode: dynamic desirable
Operational Mode: trunk
Administrative Trunking Encapsulation: negotiate
Operational Trunking Encapsulation: isl
Negotiation of Trunking: On
```

Here we end from the **DTP** ;)

@0x0nullian

→ VTP (VLAN Trunking Protocol):-

→ **VTP** allows you to configure VLANs on a central **VTP** server switch, and other Switches (called **VTP** clients) will synchronize their VLAN database to the server.

→ **VTP** is designed for large networks with many VLANs, so that you don't have to configure each VLAN on every single Switch.

→ However, like **DTP**, it is rarely used, and it is recommended that you don't use it.

→ There are three versions of **VTP**: **1**, **2** and **3**.
Most modern Cisco Switches support all three, but older ones might only support **1** and **2**.

→ There are three **VTP** modes that a Switch can operate in: **server**, **client**, and **transparent**.

→ Cisco Switches operate in **VTP server** mode by default.

- **VTP** does not automatically assign interfaces to VLANs

→ VTP Modes:-

1- VTP Servers:-
- Can add/modify/delete VLANs.
- The default mode in the Cisco Switches by default.
- You can modify the database on Cisco Switches by default.
- They store the VLAN database in non-volatile RAM, also called NVRAM.
  This means the VLAN database is saved even if the Switch is turned off or reloaded.
- They will increase the **revision number** every time a VLAN is added, modified, or deleted.

- This revision number is a very important part of **VTP**.
- It's what **VTP** uses to determine the newest version of the VLAN database, the version that the Switch will synchronize to.
- **VTP** servers will advertise the latest version of the VLAN database on trunk interfaces,
  and the **VTP** clients will synchronize their VLAN database to it.
- So, **VTP** advertisements aren't sent on access ports, only on trunk ports.

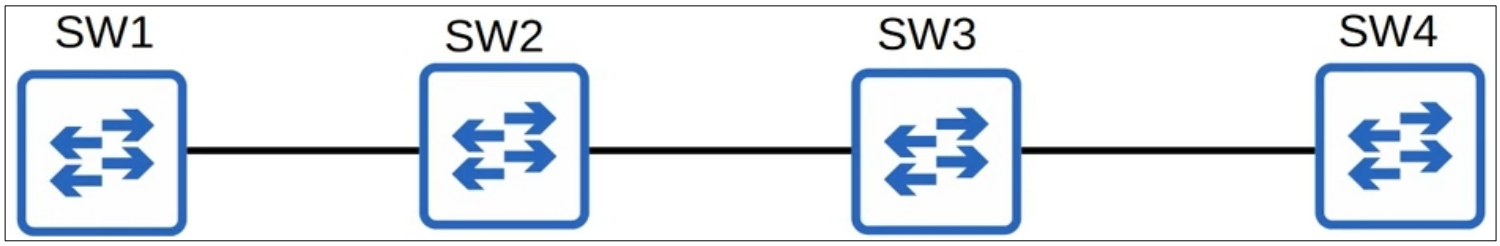
Important point: **VTP** servers also function as **VTP** clients.
It means that a **VTP** server will synchronize to another **VTP** server with a higher **revision number**.
Because the highest **revision number** is considered the newest, most accurate version of the VLAN database.

2- VTP Clients:-
- Cannot add/modify/delete VLANs. IF you tried to, the command will be rejected.
- Do not store the VLAN database in NVRAM. (however in the newest **VTP** version, in **VTPv3**, they do).
- **VTP** clients will synchronize their VLAN database to the server with the highest **revision number** in their **VTP** domain.
- They will advertise their database, and forward VLAN advertisements to other clients over their trunk ports.

We will talk about **VTP Transparent** soon.. ;)

→ How VTP Works? :-

- These are four Switches, and we've configured all of their interfaces as trunk, so they will send and receive **VTP** advertisements between each other.
- All of these Switches have the default configuration, so their output will be mostly the same.



- Here's he output of a very useful command (**`show vtp status`**), on **SW1**:

```
SW1#show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 1
VTP Domain Name                 :
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : 0c09.f956.1300
Configuration last modified by 0.0.0.0 at 0-0-00 00:00:00
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 5
Configuration Revision          : 0
MD5 digest                      : 0x57 0xCD 0x40 0x65 0x63 0x59 0x47 0xBD
                                  0x56 0x9D 0x4A 0x3E 0xA5 0x69 0x35 0xBC
```

→ Let's look at some of these fields:

- These fields here shows that the Switch is capable of running **VTP** version **1**, **2**, or **3**.
- But it is running version **1** at the moment, the default.

```
VTP Version capable             : 1 to 3
VTP version running             : 1
```

- Notice that there is no domain name.
- By default the domain name is **NULL**, there is no domain name.
- If we want **VTP** to synchronize among these devices, we will need to configure them all with the same **VTP** domain name.

- Looking down, we can see the default **VTP** operating mode of **Server**.
- The maximum number of VLANs supported locally is **1005**.
  This is because **VTPv1** and **VTPv2** do not support the extended VLAN range of (**1006 – 4094**), only **VTPv3** supports them.
  If you want to use the extended VLAN range you'll need to use **VTPv3**.
- The number of existing VLANs is **5**, those are the VLANs that exist by default in the Switch (**1**, **1002**, **1003**, **1004** and **1005**).
- Finally, look at the configuration revision number. It is **0** at the moment.
  If we add, modify, or delete a VLAN, this revision number will increase to **1**,
  and **SW1** will advertise this to **VTP** clients in the same domain.
  SW1 will also update its own VLAN database if it receives a **VTP** advertisement with a higher revision number.
  Because **VTP** servers function as **VTP** clients also.

```
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 5
Configuration Revision          : 0
```

- So, we use this command (**`vtp domain + name`**), the name is **`cisco`**, to change the **SW1**'s domain name.
- And then we made a VLAN, **VLAN10**, and named it engineering.

```
SW1(config)#vtp domain cisco
Changing VTP domain name from NULL to cisco
SW1(config)#
*May  4 02:14:47.276: %SW_VLAN-6-VTP_DOMAIN_NAME_CHG: VTP domain name changed to cisco.
SW1(config)#vlan 10
SW1(config-vlan)#name engineering
SW1(config-vlan)#exit
```

- So, because we added a VLAN, if we do (**`show vtp status`**) again, we should see that the revision number has increased!

```
SW1#show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 1
VTP Domain Name                 : cisco
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : 0c09.f956.1300
Configuration last modified by 0.0.0.0 at 5-4-20 02:18:27
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 6
Configuration Revision          : 1
MD5 digest                      : 0x9F 0xE0 0xAB 0x7A 0x78 0x62 0x68 0x70
                                  0x2D 0xD5 0x5A 0xBE 0x21 0x5D 0x56 0x49

SW1#
```

- We can see now that the **VTP** domain name has changed to **cisco**, the number of existing VLANs is now **6**, and the revision number has increased to **1**.

→ Now, Let's go on and check the other Switches:-

- Okay, so something interesting has occurred!

```
SW2#show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 1
VTP Domain Name                 : cisco
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : 0c09.f9ab.0800
Configuration last modified by 0.0.0.0 at 5-4-20 02:18:27
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 6
Configuration Revision          : 1
MD5 digest                      : 0x9F 0xE0 0xAB 0x7A 0x78 0x62 0x68 0x70
                                  0x2D 0xD5 0x5A 0xBE 0x21 0x5D 0x56 0x49

SW2#
```

- Without any configuration, **SW2** has changed its domain name to **cisco** and updated its VLAN database to match **SW1**'s, with a revision number of **1**.

> If a switch with no VTP domain (domain NULL) receives a VTP advertisement with a VTP domain name, it will automatically join that VTP domain.

- So, **SW2** automatically joined the domain **cisco**.

> If a switch receives a VTP advertisement in the same VTP domain with a higher revision number, it will update it's VLAN database to match.

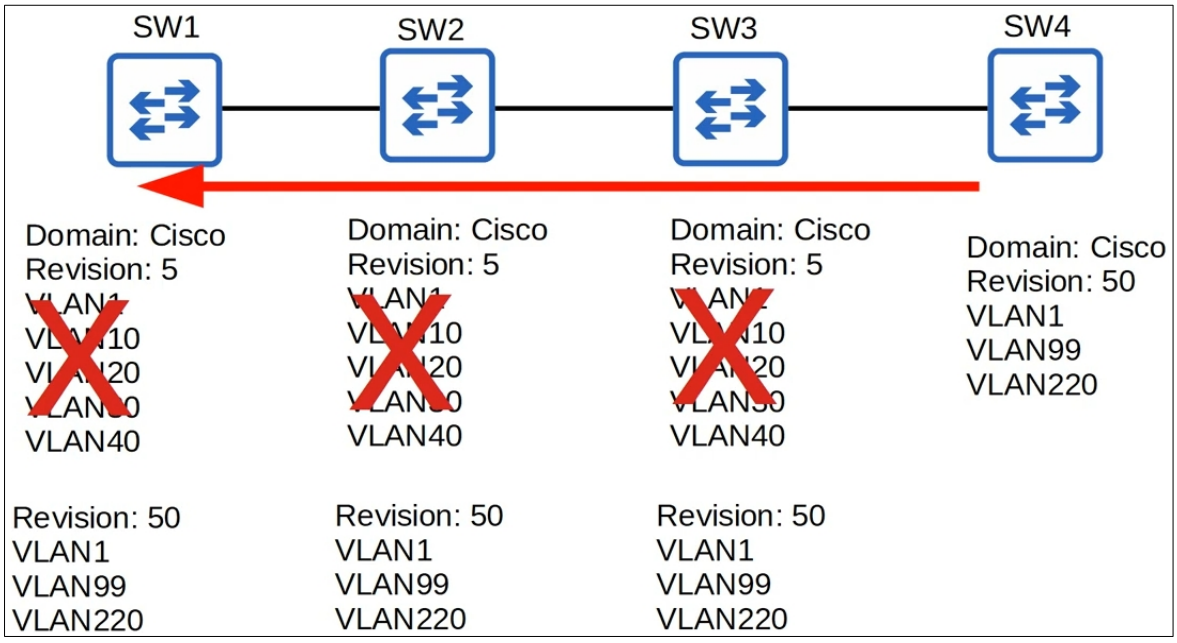
- If we do (**`show vlan brief`**) on **SW2**, we can see that **VLAN10**, with the name '**engineering**', was added.

```
10   engineering
1002 fddi-default
1003 token-ring-default
1004 fddinet-default
1005 trnet-default
SW2#
```

- If we did (**`show vtp status`**) on **SW3** and **SW4**, we will see the advertisement were passed along to them as well, and they joined the domain and updated their VLAN database as well.

NOTE: **VTP** only syncs the VLAN database, we still have to configure interfaces on each Switch separately. For example, **`switchport access vlan10`**, etc.

→ Since we have seen how **VTP** works, There is one danger of **VTP**:
- If you connect an old Switch with a higher revision number to your network (and the **VTP** domain name mathces), all Switches in the domain will sync their VLAN database to that Switch.
- This could cause all of the hosts on you network to instantly lose connectivity, because the Switches could sync to a totally different **VTP** database, and the VLANs you were using could disappear.
- This is one reason why **VTP** is usually not used in modern networks.



- Just to demonstrate, let's say this **VTP** domain Cisco has a revision number of **5**, and VLANs **1**, **10**, **20**, **30**, and **40.**
- Then we take an old Switch our company used to use, to add to the network, However it has a revision number of **50**, and VLANs **1**, **99**, and **220**.
- The newer Switch, will send **VTP** advertisements with this revision number, which will be forwarded throughout the domain.
- All of these Switches will update their VLAN database to match, and all hosts in VLANs **10**, **20**, **30**, and **40** will suddenly lose connectivity.

3- VTP Transparent mode:-

- Switches in **VTP** transparent mode do not participate in the **VTP** domain, they do not sync their VLAN database to the **VTP** server.
- Maintains its own independent VLAN database In NVRAM.
- It can add/modify/delete VLANs, but they won't be advertised to other Switches.
- Although it doesn't sync its VLAN database, it will forward **VTP** advertisements over its trunk ports, if the **VTP** advertisements is in the same domain as it, but it won't advertise its own VLAN database.

→ Let's compare the functionality of server, client, and transparent mode Switches:
- We first set **SW2** to client mode with the command (**`vtp mode client`**).
- Afterward, we tried to create **VLAN20** on the Switch, but as we can see it was rejected! Because **SW2** is now in client mode!

```
SW2(config)#vtp mode client
Setting device to VTP Client mode for VLANS.
SW2(config)#vlan 20
VTP VLAN configuration not allowed when device is in CLIENT mode.
SW2(config)#
```

- Then we set **SW3** to transparent mode with (**`vtp mode transparent`**).
- To show you that a transparent mode Switch won't forward advertisements if its in a different domain, we changed the domain name to Juniper.

```
SW3(config)#vtp mode transparent
Setting device to VTP Transparent mode for VLANS.
SW3(config)#vtp domain juniper
Changing VTP domain name from cisco to juniper
SW3(config)#
```

- So, we created **VLAN20**, named **Sales**, o **SW1**, and we can see it appears in the output of (**`show vlan brief`**).

```
SW1(config)#vlan 20
SW1(config-vlan)#name sales
SW1(config-vlan)#exit
SW1(config)#do show vlan brief

VLAN Name                              Status
---- --------------------------------- ---------
1    default                           active



10   engineering                       active
20   sales                             active
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
1005 trnet-default                     act/unsup
SW1(config)#
```

- Then we did (**`show vtp status`**), and we can see that the configuration revision number is **4**.
- It should be **2**, but we made a few other changes as we was trying things out in the lab for this video.

```
Number of existing VLANs          : 7
Configuration Revision            : 4
```

- Let's check **SW2**: (**`show vlan brief`**)
- As we can see, the **VTP** client **SW2** has indeed added **VLAN20** to its VLAN database, and it now has the same revision number, **4**.

`show vlan brief`

```
10   engineering                       active
20   sales                             active
```

`show vtp status`

```
Number of existing VLANs          : 7
Configuration Revision            : 4
```

→ How about the transparent Switch, **SW3**?
- As expected, on the transparent Switch **SW3**, **vLAN20** was not added, and now it has a revision number of **0**.

```
show vlan brief
10   engineering                       active
```

```
show vtp status
Number of existing VLANs          : 6
Configuration Revision            : 0
```

-- Changing the **VTP** domain to an unused domain will reset the revision number to **0**.
-- Changing the **VTP** mode to transparent will also reset the revision number to **0**.

→ So, if you're going to plug an old Switch with a high revision number into a network that uses **VTP**, make sure to reset the revision number with one of these methods first, so it doesn't overwrite your network's VLAN configurations.

→ Whether **SW4** will have added **VLAN20** to its VLAN database??

- **SW3** is in transparent mode in a different domain, so it shouldn't forward the **VTP** advertisements to **SW4**.
- Indeed, **SW4** doesn't have **VLAN20**, and it is still on revision number **3**.

```
show vlan brief
10   engineering                       active
```

```
show vtp status
Number of existing VLANs          : 6
Configuration Revision            : 3
```

→ So, what we can do to make **SW3** start forwarding the **VTP** advertisements to **SW4**??

- If we change the **VTP** domain on **SW3** back to Cisco, it should start forwarding advertisements to **SW4**.
- Even though **SW3** itself won't sync its own VLAN database based on those advertisements.
- So, we changed the **VTP** domain on **SW3** back to Cisco.

```
SW3(config)#vtp domain cisco
Changing VTP domain name from juniper to cisco
SW3(config)#
*May  4 04:06:00.101: %SW_VLAN-6-VTP_DOMAIN_NAM
SW3(config)#
```

- We also created some new VLANs on **SW1** to increase the revision number ad send more advertisements, by the way.
- And now we can see that **SW3** did indeed forward the advertisements to **SW4**,
  and **SW4** synced its VLAN database to **SW1** and **SW2**.

```
SW4#show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 1
VTP Domain Name                 : cisco
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : 0c09.f972.8700
Configuration last modified by 0.0.0.0 at 5-4-20 04:15:14
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 11
Configuration Revision          : 12
MD5 digest                      : 0xDB 0x14 0xEF 0x30 0
                                  0xEC 0x6C 0x96 0xAD 0
SW4#
```

→ VTP version:-

- To change the **VTP** version, use the command (**`vtp version + n`**), while **`n`** is the version.
- Changing the **VTP** version increases the revision number, by the way, and advertisements with the new number will be sent.

```
SW1(config)#vtp version 2
SW1(config)#do show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 2
VTP Domain Name                 : cisco
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : 0c09.f956.1300
Configuration last modified by 0.0.0.0 at 5-4-20 04:19:30
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 11
Configuration Revision          : 13
MD5 digest                      : 0xE4 0xC9 0x65 0x0A 0x
                                  0x99 0xB2 0x16 0x81 0x

SW1(config)#
```

- Other servers and clients will then sync and start operating in version **2** as well!
- For example, here is **SW4**, it is now running version **2** and has a revision number of **13**, just like **SW1**.

```
SW4#show vtp status
VTP Version capable             : 1 to 3
VTP version running             : 2
VTP Domain Name                 : cisco
VTP Pruning Mode                : Disabled
VTP Traps Generation            : Disabled
Device ID                       : 0c09.f972.8700
Configuration last modified by 0.0.0.0 at 5-4-20 04:19:30
Local updater ID is 0.0.0.0 (no valid interface found)

Feature VLAN:
--------------
VTP Operating Mode              : Server
Maximum VLANs supported locally : 1005
Number of existing VLANs        : 11
Configuration Revision          : 13
MD5 digest                      : 0xE4 0xC9 0x65 0x0A 0x
                                  0x99 0xB2 0x16 0x81 0x

SW4#
```

→ As for the difference between **VTP** version **1** and **2**, here is a quote directly from Cisco:

> "**VTPv2** is not much different than **VTPv1**. The major difference is that **VTPv2** introduces support for **Token Ring VLANs**. If you use **Token Ring VLANs**, you must enable **VTPv2**. Otherwise , there is no reason to use **VTPv2**."

→ **Token Ring** is an old technology, so really there is no reason to use version **2**!

→ As for version **3**, it has quite a few differences and new features, but it's certainly beyond the scope of the CCNA, so we'll leave it here!

Day 20: Spanning Tree Protocol pt.1:-

→ The CCNA exam topics list mentions **Rapid Spanning Tree**, an updated and superior version of **STP**.
However, to understand **Rapid STP**, it's important to understand **Classic STP** first.

→ Network Redundancy :-

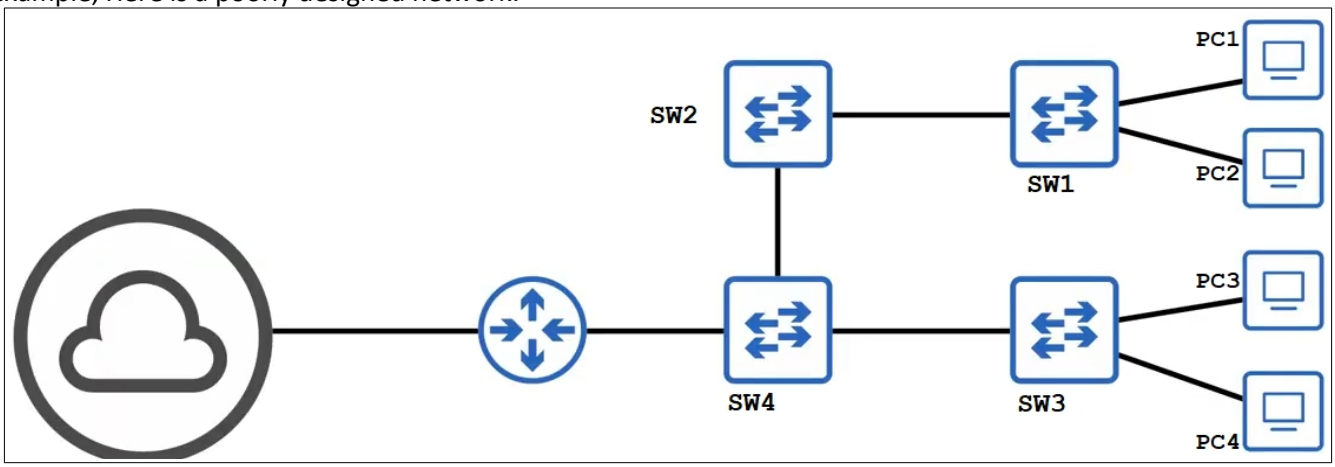
→ Redundancy is an essential part of network design.
A network that is not redundant is simply not acceptable.

→ Modern networks are expected to run **24/7/365**. Even a short downtime can be disastrous for a business.

→ If one network component fails, you must ensure that other components take over with little or no downtime.

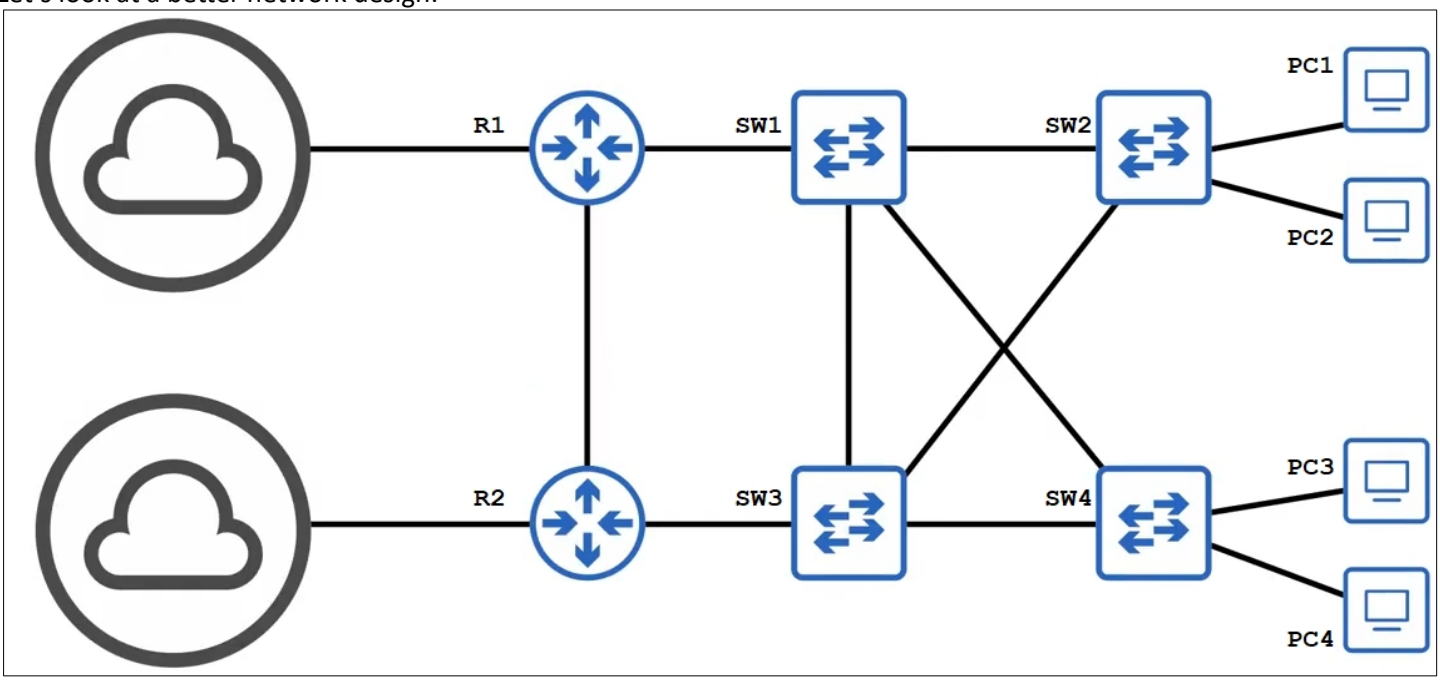
→ As much as possible, you must implement redundancy at every possible point in the network.
As Network Engineers, we are responsible for business-critical infrastructure.
So we have to make sure that that infrastructure is resilient to failures as much as possible.

→ For example, Here is a poorly designed network:



- There are many points of failure here which could cut off connectivity.
- For example, if the connection between the Router and the Internet is cut due to a hardware failure, this entire network loses connectivity to the Internet.
- Or, if the connection between **SW1** and **SW2** is cut off due to a hardware failure, these hosts, **PC1** and **PC2**, lose connectivity within a LAN, and out to the Internet.

→ Let's look at a better network design:-



- If **PC4** wants to reach the Internet, it might use this path in a normal situation (**SW4** → **SW3** → **R2** → **Internet**).
  However, even if **R2** has a hardware failure and goes down completely, **PC4** can reach the Internet via this or another alternate path (**SW4** → **SW1** → **R1** → **Internet**).

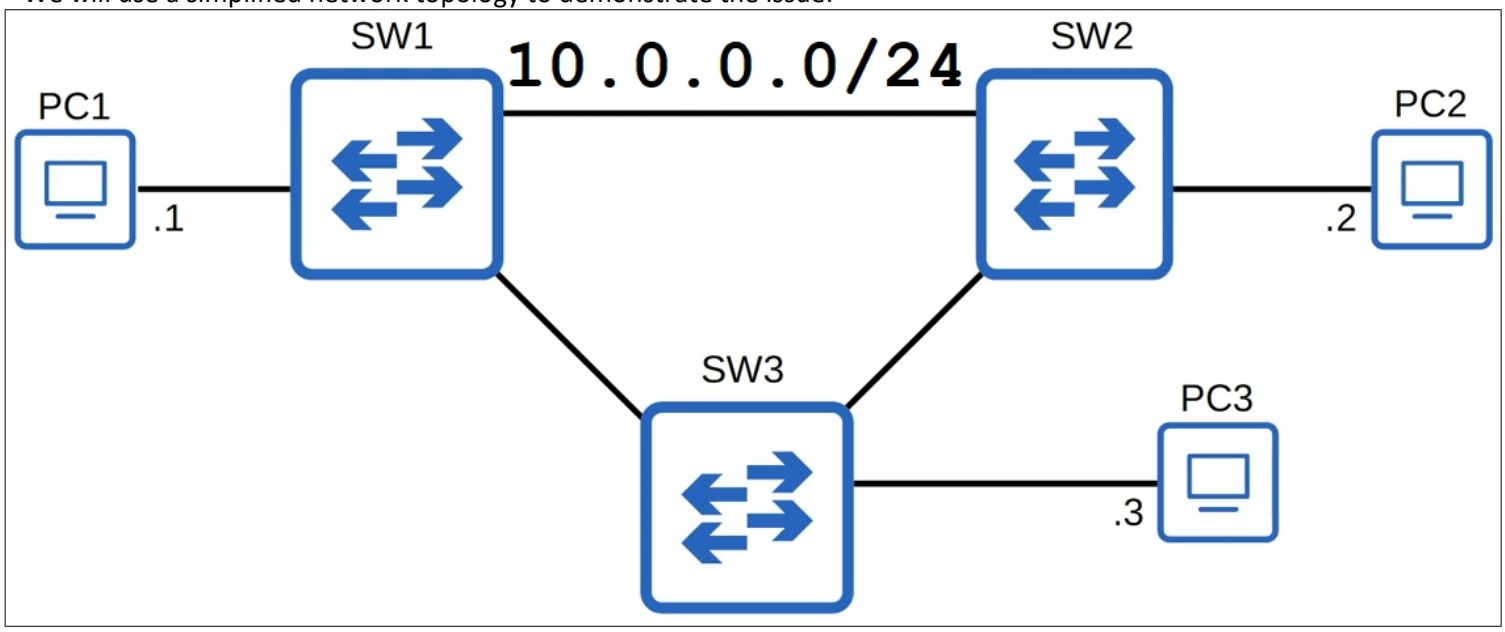
- Perhaps traffic from **PC4** to this other PC, **PC2**, in the LAN usually follows this path to the destination (**SW4** → **SW1** → **SW2** → **PC2**).
  What if **SW1** fails?? That's not a problem, because this alternate path is available (**SW4** → **SW3** → **SW2** → **PC2**).

- So what if **SW4** fails?? All hosts (**PC3** and **PC4**) connected to this Switch would lose connectivity.
  Unfortunately, Most PCs only have a single Network Interface Card, **NIC**, so they can only be plugged into a single Switch.
  However, important Servers typically have multiple **NIC**s, so they can be plugged into multiple Switches for Redundancy.

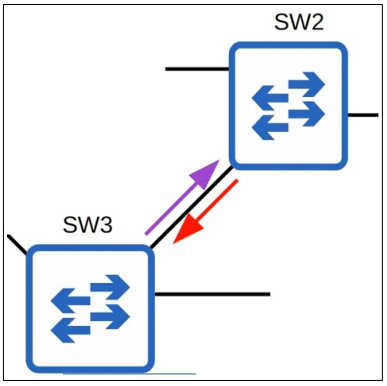
- We will cover many protocols that are used to enable Network Redundancy throughout the course, and the **STP** is one of them.
  STP is a Layer2 protocol, BTY, it enables redundant Layer2 networks.
  So, within the LAN in here, not routing out to the Internet and or to Routers between networks at Layer3.

- We just saw the benefits of a redundant LAN, having multiple paths between those Switches provides alternate paths if one connection fails.

→ However, without **STP**, there is a MAJOR problem here which can destroy our Network! So, where is the problem?
Let's introduce the concept of **Broadcast Storms**.
We will use a simplified network topology to demonstrate the issue:



- We already know what a Switch does with a broadcast frame or an unknown uni-cast frame.
- Let's say for example, **PC1** wants to send some traffic to **PC2.**
To do that, it needs to know **PC2**'s MAC address. So, let's say **PC1** sends an **ARP request** frame, which is a broadcast frame, it uses the broadcast MAC address of all **FFFF**'s as it is Layer2 address.
If **SW1** receives the frame, it will flood it out of all interfaces, except the one it was received on.
So, **SW2** and **SW3** both receive a copy of the frame. They then do the same thing, they flood it out all interfaces except the one.
So, **PC2** receives the **ARP request** and will reply with a uni-cast **ARP reply**.
All good?? Actually no, NOT all good!!
Although **PC2** receives the **ARP request** and sent its reply, these broadcast frames still remain on the network!!!



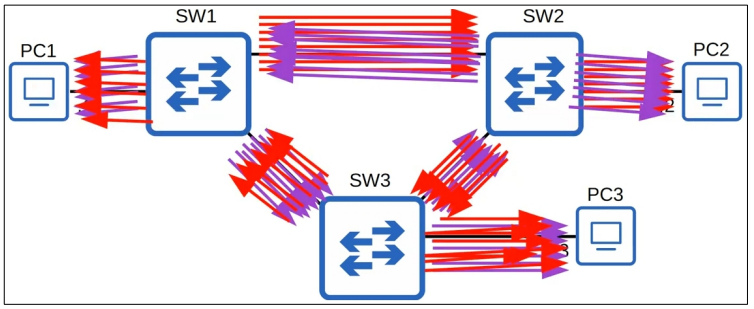
- As we just said, **PC2** received the **ARP request** and sent the **ARP reply**, but what about these broadcast frames in the network?
The Switches will continue flooding them. So, what will happen after this????

- **SW1** just received two broadcast frames, on two different interfaces.
It will once again flood them!
SW2 and **SW3** both just received broadcast frames, what will they do? They will flood them!
Now you should get the point, this will continue forever!!

- Do you remember the **TTL**, or **Time To Live**, field of the **IPv4** header?
It is used to prevent infinite loops at Layer3.

> The Ethernet header doesn't have a TTL field. These broadcast frames will loop around the network indefinitely. If enough of these looped broadcasts accumulate in the network, the network will be too congested for legitimate traffic to use the network. This is called a **broadcast storm**.

- Eventually, our network will look like this, so full of looping broadcast frames that no regular traffic can pass through the network.
- Red arrows represent clock-wise loop between the **3** Switches, and the Purple arrows the counter-clockwise loop.



- However, Network Congestion isn't the only problem!
- Each time a frame arrives on a switchport, the Switch uses the Source MAC address field to ***learn*** the MAC address and update its MAC address table.
- When frames with the same source MAC address repeatedly arrive on different interfaces, the Switch is continuously updating the interface in its MAC address table!
- This is known as **MAC Address Flapping**.

→ How can we design a network with redundant paths that doesn't result in Layer2 loops?
- **Spanning Tree Protocol** is one answer to this problem!!

→ Spanning Tree Protocol:-

→ What we now call '**Classic Spanning Tree Protocol**' is an industry standard protocol, **IEEE 802.1D**.
This is the type of **STP** we will focus on in this Day. We will focus on the newer, **Rapid STP** later.

→ Switches from ALL vendors run **STP** by default.
It is so important to prevent Layer2 loops.

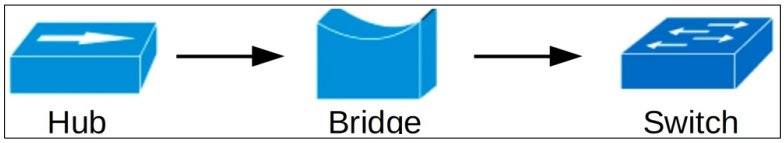
→ **STP** prevents Layer2 loops by placing redundant ports in a blocking state, essentially disabling the interface.

→ These interfaces act as backups that can enter a forwarding state if an active interface, meaning an active interface that is currently forwarding, fails.

→ Interfaces in a forwarding state behave normally. They send and receive normal traffic.

→ However, Interfaces in a blocking state only send or receive **STP** messages and some other specific traffic.
STP messages called **BPDU**s, **Bridge Protocol Data Units**

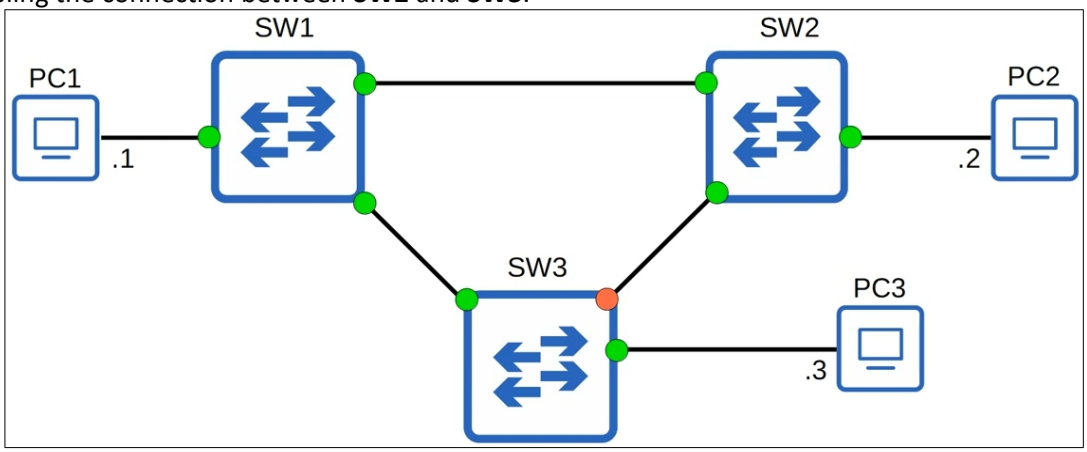
→ Let's talk about **Bridge**:-
- We talked about Hubs in previous videos, They were used before Switches were invented.
- Instead of learning MAC addresses to forward frames to correct destinations, they simply flooded frames out of all interfaces.
- But actually before Switches, there was another kind of device called a Bridge!
- The Bridges are an old technology. They're like a transitional stage between the Hub and the Switch.



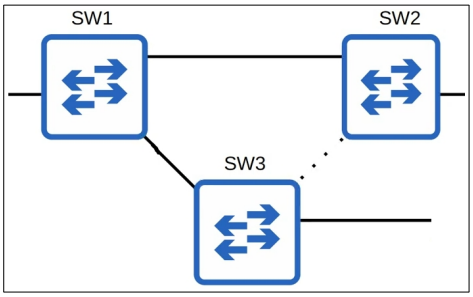
- We're talking about the Bridge, because the **STP** still uses the term '**bridge**'.
- However, when we use the term '**bridge**' we really mean '**switch**'. Bridges are not used in modern networks.

→ Back to our Network Topology:

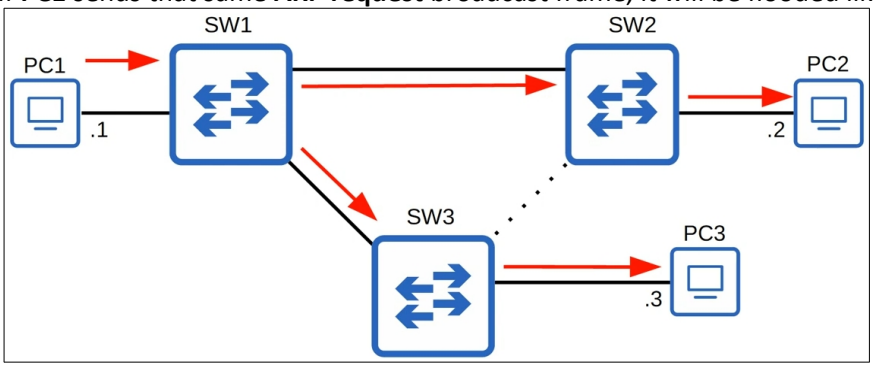
- The interfaces with the GREEN point are in a forwarding state, but the other one with RED on **SW3** is in a blocking state, effectively disabling the connection between **SW2** and **SW3**.



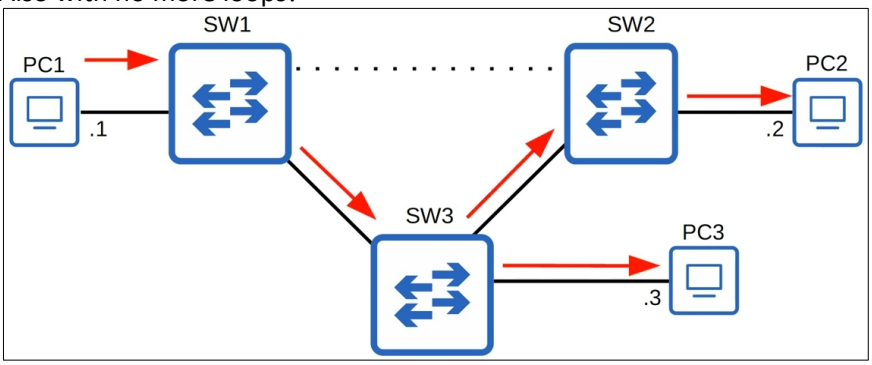
- So, effectively its like that link (between **SW2** and **SW3**) doesn't exist.



→ If **PC1** sends that same **ARP request** broadcast frame, it will be flooded like this: No more loops!



→ However, if at some point another interfaces fails, perhaps the one on the **SW2**,
The Switches will automatically adjust the topology, and the broadcast frame would be flooded like this:
Also with no more loops!



- This is just a basic outline of the purpose of **Spanning Tree Protocol**.

→ How STP Works? :-

- By selecting which ports are *forwarding* and which ports are *blocking*, **STP** creates a single path to/from each point in the network.
  This prevents Layer2 loops.

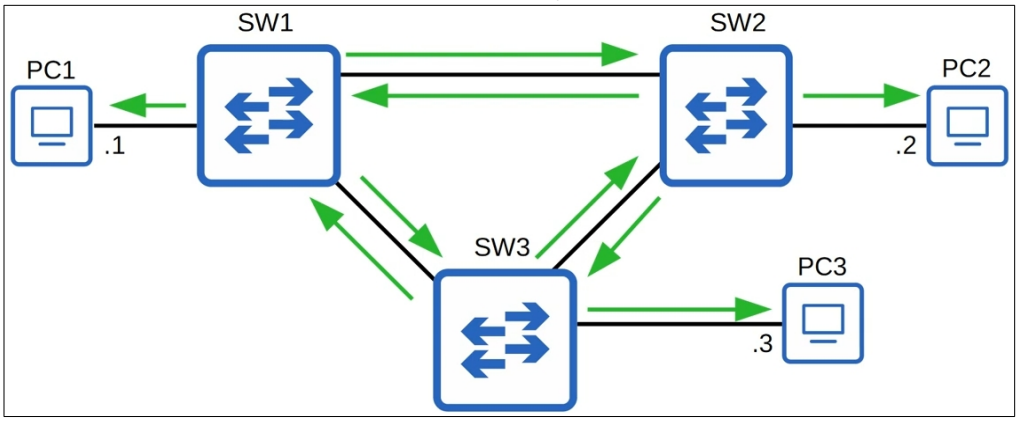
- There is a set process that **STP** uses to determine which ports should be forwarding and which should be blocking.

- **STP**-enabled Switches send/receive Hello **BPDU**s out of all interfaces, the default timer is **2** seconds.
  (The Switch will send a Hello **BPDU** out of every interface, once every **2** seconds).

- If a Switch receives a Hello **BPDU** on an interface, it knows that interfaces is connected to another Switch
  (Routers, PCs, etc do not use **STP**, so they don't send Hello **BPDU**s).

→ Back to our Topology:

- These Switches will send **BPDU**s out of each interface, like this:
- They use these **BPDU**s to advertise themselves to other Switches, and to learn about other Switches.

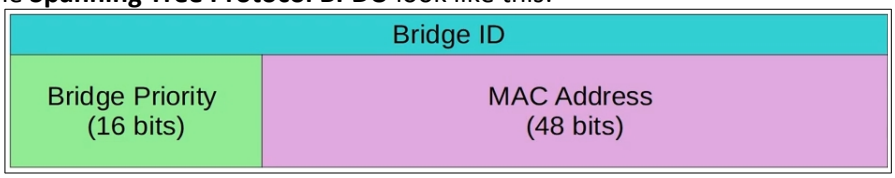


→ What are the BPDUs used for? :-

- First of all, Switches use one field in the **STP BPDU**, the **Bridge ID** field, to elect a ***root bridge*** for the network.
- The Switch with the lowest **Bridge ID** becomes the ***root bridge***.
- ALL ports in the ***root bridge*** are put in a forwarding state,
  and other Switches in the topology must have path to reach that ***root bridge***.
- **STP** puts ports in either a blocking or forwarding state, to avoid Layer2 loops in the network.
  However, as we just said, on the ***root bridge***, ALL ports are forwarding, and all other Switches must have a path to reach the ***root***.

→ Bridge ID field:-

→ The **Bridge ID** field of the **Spanning Tree Protocol BPDU** look like this:



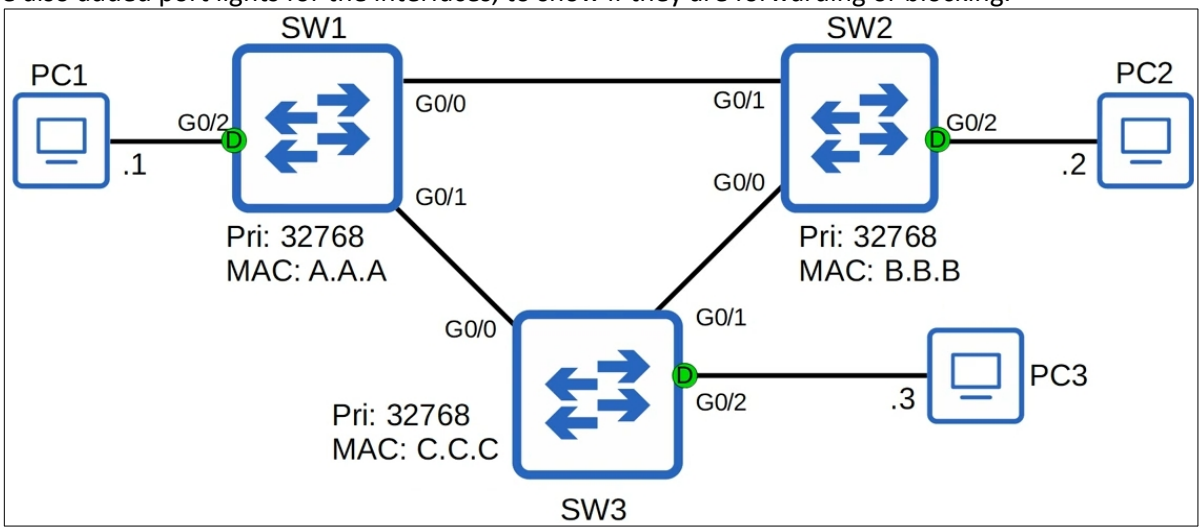
- There is a *Bridge Priority* field, which is **16** bits in length.
- And then there is the MAC address of the Switch, which is already known is **48** bits in length.
- The default *Bridge Priority* is **32768** on all Switches, so by default the MAC address is used the tie-breaker.
  Lowest MAC Address becomes the ***root bridge***.

- As we said before, the Switch with the lowest **Bridge ID** becomes the **root bridge**.
  So, by default the Switch with the lowest MAC address becomes the ***root bridge***.

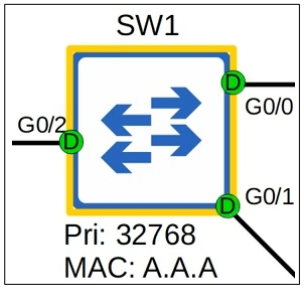
** The *Bridge Priority* is compared first, If they tie, the MAC Address is then compared. **

→ Back to our Topology, with the Priority and MAC address for each Switch:

- MAC addresses are **12** hexadecimal, but we shorten them to only **3**.
  We've also added port lights for the interfaces, to show if they are forwarding or blocking.



- The **G0/2** interface on each Switch is connected to a PC, so because it isn't receiving any **BPDU**s,
  It knows it is safe to go into forwarding mode, there is no risk of creating a Layer2 loop, So, these port light are all green!
- All three Switches have the default priority of **32768**, so in order to know which one will be the ***root bridge***,
  we will have to compare the MAC Address. **<u>Remember:</u>** The LOWEST Bridge ID wins!!
- Now, which of these MAC addresses is the lowest?
  Hexadecimal **A** is equal to **10**, **B** is equal to **11**, and **C** is equal to **12**, so **SW1** has the LOWEST MAC address.
  Therefore, **SW1** will become the ***root bridge*** of this network :)
- All ports on the ***root bridge*** become designated ports, in a forwarding state.



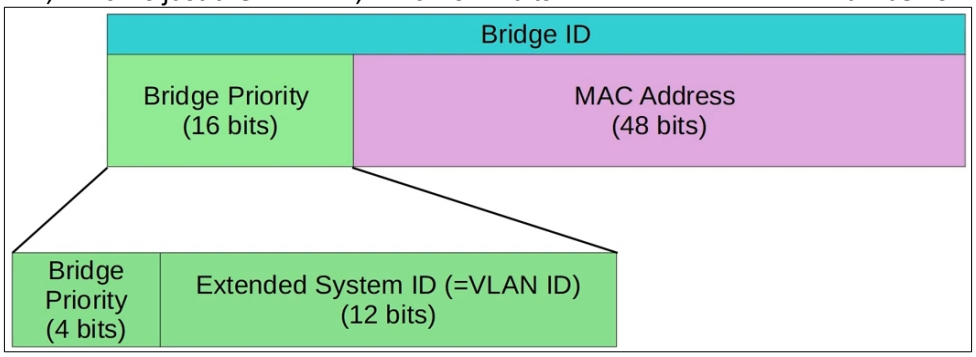
- So, that was the traditional *Bridge ID*.

→ However, the *Bridge ID* was was actually updated to look like this:

- In reality, the *Bridge ID* has been updated to be made of two parts:

1- The *Bridge Priority* which is **4** bits.

2- The *Extended System ID*, which is just the VLAN ID, which is **12** bits. A VLAN number is **12** bits in length.



- Why to include a VLAN ID in the *Bridge Priority*??
  Well, Cisco Switches use a version of **STP** called **PVST**, stands for **Per-VLAN Spanning Tree**.
- **PVST** runs a separate **STP** *instance* in each VLAN, so in each VLAN different interfaces can be forwarding/blocking.
- One interface could be forwarding in **VLAN1**, but blocking in **VLAN2**, for example.

- By adding the VLAN ID into the *Bridge Priority*, the Switch will have a different **Bridge ID** in each VLAN.

→ Here's a deeper look at the ***Bridge Priority*** field:-

Bridge Priority				Extended System ID (VLAN ID)											
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

- You may have wondered why **32768** is the default bridge priority?
- It is because this total field is **16** bits in length, and the most significant bit is set to **1** by default.
  Therefore, the default bridge priority was **32768**.
- However, with the addition of the Extended-System ID, adding the VLAN number to the Bridge Priority, that changed.
  So, the default VLAN ID is **1**, therefore the Bridge Priority in total actually ISN'T **32768**, it's **32769**
- In the default VLAN of **1**, the default bridge priority is actually **32769**, which is **32768 + 1**.

> In the default VLAN of 1, the default bridge priority is actually **32769** (32768 + 1).

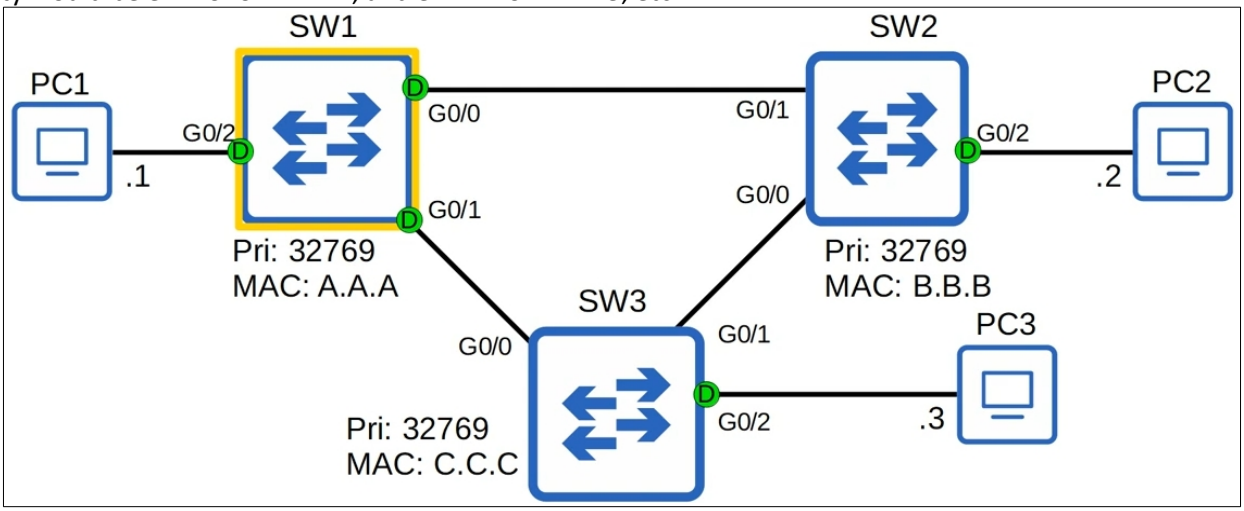
→ Here's a question, If we want to increase the Switch's bridge priority (without changing VLAN numbers), what is the minimum unit of increase/decrease?

- The *Bridge Priority + Extended System ID* is a single field of the ***Bridge ID***.
  However the Extended System ID is set and cannot be changed (because it is determined by the VLAN ID).
- Therefore, You can only change the total *Bridge Priority* (*Bridge Priority + Extended System ID*) in units of **4096**,
  The value of the least significant bit of the *Bridge Priority* portion.

- Currently, the Bridge Priority is **32769**. Let's reduce it to make this Switch the ***root bridge***.
- If we want to reduce it just a little, we can reduce it to **28673**. Which is (**16384 + 8192 + 4096 + 1**).
- We can reduce it more, of course, but the point is this:

> The STP bridge priority can only be changed in units of 4096.
>
> The valid values you can configure are:
> 0, 4096, 8192, 12288, 16384, 20480, 24576, 28672, 32768, 36864, 40960, 45056, 49152, 53248, 57344, or 61440.
> The Extended System ID will then be added to this number to make the total bridge priority.

→ Back t Our Topology again:
- We'll just be looking at the **STP** topology for a single VLAN, **VLAN1**, so the priority for each Switch is **32769**.
- But if there are multiple VLANs, **VLAN1**, **VLAN2**, and **VLAN3** in this network,
  The priority would be **32770** for **VLAN2**, and **32771** for **VLAN3**, etc.



- We could also change the Bridge Priority on the Switches for a specific VLAN, so for example, **SW1** is the ***root bridge*** in **VLAN1**, **SW2** could be the **root bridge** in **VLAN2**, and **SW3** could be the ***root bridge*** for **VLAN3**.
  We'll talk about this in the Next Video, just to know som of the possibilities.

→ So, here in **VLAN1**, **SW1** is the ***root bridge***.

All interfaces on the root bridge are **designated ports**.
Designated ports are in a forwarding state.

→ Designated Port is one of the port roles in **Spanning Tree Protocol**.
There are couple other port roles.

→ The Root Bridge:-

- When a Switch is powered on, it assumes it is the *root bridge*.
- It will only give up its position if it receives a 'Superior' **BPDU**, and Superior means a **BPDU** from a Switch with a lower Bridge ID.
- Once the Topology has converged and all Switches agree on the ***root bridge***, only the *root bridge* sends **BPDU**.
  The reason all Switches send **BPDU**s at first is because they all think they are the ***Root Bridge***!!
- Other Switches on the network will forward **BPDU**s from the *root bridge*, but they will not generate their own original **BPDU**s.

→ So far we've covered the first step of **Spanning-Tree Protocol**'s process of creating loop-free Layer2 LANs:
Step1: The Switch with the lowest bridge ID is elected as the root bridge. All ports on the root bridge are **designated ports**.
The rest of the steps depending on knowing which Switch is the root bridge.
Step2: All other Switches will select ONE of its ports to be its ***root port***.
That means there is one root port on each Switch in the network, EXCEPT on the root bridge.
The interface with the lowest *root cost* will be the root port. Root ports are also in a forwarding state.

→ What 'Root Cost' is? :-

Each interface has an associated Spanning Tree 'Cost'. We can refer to these costs with this chart:

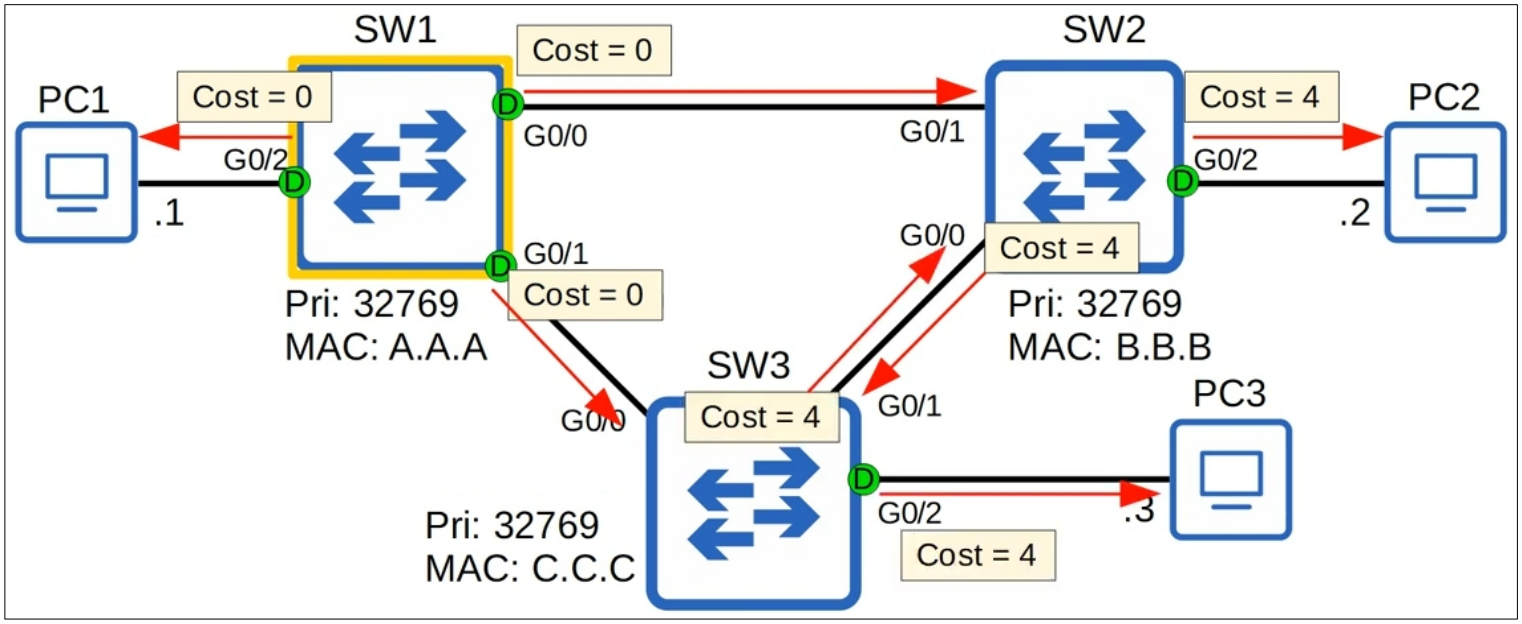
Speed	STP Cost
10 Mbps	100
100 Mbps	19
1 Gbps	4
10 Gbps	2

- The ***Root Cost*** is the total of the outgoing interfaces along the path to the *Root Bridge*.

__ However, what if a Switch has multiple ports with the same root cost?
__ In that case, the interface connected to neighbor with the lowest Bridge ID will be the Root port..

→ Back to our Topology:
- All ports are Gigabit Ethernet ports, so they all have a cost of **4**.
- The ***Root Cost*** is the total of the outgoing interfaces along the path to the *Root Bridge*.
- **SW1** is the *Root Bridge*, so it has a cost of **0** on all interfaces.
- They're Gigabit Ethernet interfaces, but we don't count the cost of the receiving interface, just the sending, the outgoing interface.

→ So, **SW1** advertises its root cost of **0** in its **BPDU**s.
→ **SW2** will receive the **BPDU** and add the cost of its outgoing interface, **G0/1**,
which is **4**, when it floods those **BPDU**s out of its interface.
→ **SW3** will do the same.



→ Which port **SW2** will choose as its *root port*? Here's its logic:

- It was advertised a cost of **0** on its **G0/1** interface, however the cost of its interface is **4**, therefore the total cost via **G0/1** is **4**.
- It was advertised a cost of **4** on **G0/0**, from **SW3**. However, its interface also has a cost of **4**, so the total root cost via **G0/0** is **8**.
- So, it will select **G0/1** as the *Root Port*.

→ **SW3**' logic follows the same process:
- It has a total cost of **4** via **G0/0**, and a total cost of **8** via **G0/1**. So, it will select **G0/0** as its *Root Port*.

SW2's logic:
I was advertised a cost of 0 on G0/1. My interface cost = 4.
Total cost via G0/1 = 4

I was advertised a cost of 4 on G0/0. My interface cost = 4.
Total cost via G0/0 = 8

SW3's logic:
I was advertised a cost of 0 on G0/0. My interface cost = 4.
Total cost via G0/0 = 4

I was advertised a cost of 4 on G0/1. My interface cost = 4.
Total cost via G0/1 = 8

- In this case, the ports directly across from each Root Port are the root bridge, so they are already designated ports.
- However, Keep in mind that the port connected to another Switch's root port MUST be designated.
Because the root port is the Switch's path to the Root Bridge, another Switch must not block it.

The ports connected to another switch's root port MUST be designated. Because the root port is the switch's path to the root bridge, another switch must not block it.

→ Spanning Tree Summary:-

1- One Switch is elected as the Root Bridge. All ports on the Root Bridge are **designated ports** (forwarding state).
There is only one step in selecting the Root Bridge:
1. Lowest Bridge ID

2- Each remaining Switch will select ONE of its ports to be its ***root port*** (forwarding state).
Ports across from, ports connected to, the Root Port are always **designated ports**.
Root port selection:
1. Lowest root cost
2. Lowest neighbor Bridge ID
3. Lowest neighbor port ID

→ However, there is ONE more tiebreaker (number **3**) that might be needed to select the root port.
- What if two Switches have two connections between them, so both the root cost and the Neighbor Bridge ID are the same??
We get to the final tie-breaker, the interface connected to the interface on the neighbor Switch with the lowest port ID will become the Root Port.

→ Port ID briefly:-
- Here's the output of the (**`show spanning-tree`**) command :
- We just need to see that section of **Prio.Nbr**, or **Priority Number**:
This section lists the **Spanning Tree Port ID** of each interface on the Switch.

```
SW1#show spanning-tree

VLAN0001
  Spanning tree enabled protocol ieee
  Root ID     Priority    32769
              Address     aaaa.aaaa.aaaa
              This bridge is the root
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec

  Bridge ID   Priority    32769  (priority 32768 sys-id-ext 1)
              Address     aaaa.aaaa.aaaa
              Hello Time   2 sec  Max Age 20 sec  Forward Delay 15 sec
              Aging Time  15  sec

Interface           Role Sts Cost      Prio.Nbr Type
------------------- ---- --- --------- -------- --------------------------------
Gi0/0               Desg FWD 4         128.1    Shr
Gi0/1               Desg FWD 4         128.2    Shr
Gi0/2               Desg FWD 4         128.3    Shr
Gi0/3               Desg FWD 4         128.4    Shr
Gi1/0               Desg FWD 4         128.5    Shr
Gi1/1               Desg FWD 4         128.6    Shr
Gi1/2               Desg FWD 4         128.7    Shr
Gi1/3               Desg FWD 4         128.8    Shr
```

- As the name is Prio.Number, So, each port has a default priority of **128**, and then a unique port number,
1 for **G0/0**, **2** for **G0/1**, etc on this Switch.
- The **STP Port ID** equals the Port Priority plus the port number.

STP Port ID = port priority (default 128) + port number

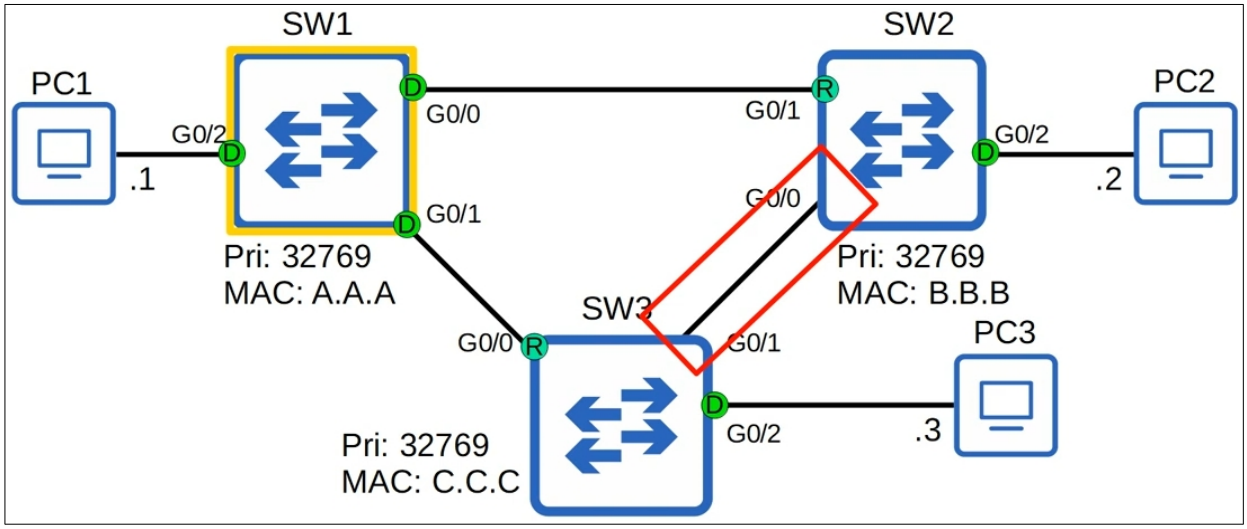
- Similar to the Bridge ID, where the MAC address is used as a tie-breaker if the priorities tie.
- In this case, the Port number is used as a tie-breaker if the priorities tie!

The NEIGHBOR switch's port ID is used to break the tie, not the local switch's port ID.

→ This our Process so far, But it's not complete! We still haven't blocked any ports, and we need to block some ports to prevent Layer2 loops.

→ Let's return to our previous topology:

- All that left is this connection between **SW2** and **SW3**.

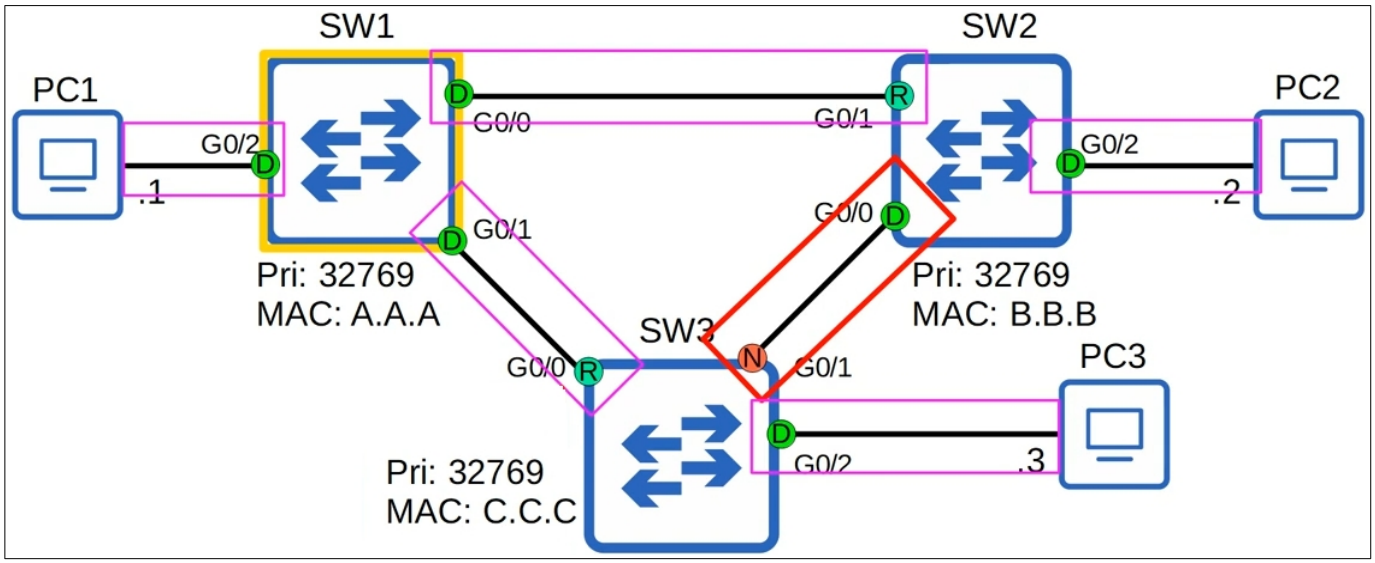


- So far, all of our ports are in a forwarding state, both root ports and designated ports are always in a forwarding state.
- So, to prevent loops do we block both of these ports?? **SW2**' **G0/0** and **SW3**'s **G0/1**??
  Actually NO!, Here's an important rule to remember:
  "<u>Every collision domain has a single Spanning Tree designated port</u>".

- Remember, unlike old Ethernet Hubs, when we use Switches, each link is a separate collision domain
- This collision domain between **SW1** and **SW2** has one designated port, **SW1**'s **G0/0**.
- This connection between SW1 and SW3 has one, SW1's G0/1 interface.
- And the connection between the PCs, are all designated ports in the forwarding state, because PCs don't participate in **STP**.
- So, we need one designated port in the connection between **SW2** and **SW3**.

→ How do we determine which port will be designated, in a forwarding state?
1- The Switch with the lowest root cost will make its port designated.
2- If the root cost is the same, the Switch with the lowest Bridge ID will make its port designated.
3- The other Switch will make its port non-designated (blocked).

- In our case, both Switches have the same root cost, **4** for **SW2** via its **G0/1** interface, and **4** for **SW3** via its **G0/0** interface.
- For the tie-breaker, we compare the Bridge ID. **SW2** has the lower Bridge ID, so its **G0/0** interface will be designated.
- Finally, the other Switch will make its port non-designated, which means it is in a blocking state.
- So, **SW3**'s **G0/1** is non-designated, it blocks the port to prevent Layer2 loops!



→ Process of selecting the different port roles and states in Spanning Tree Protocol, summary again:-

1- One Switch is elected as the Root Bridge. All ports on the Root Bridge are **designated ports** (forwarding state).

There is only one step in selecting the Root Bridge:

1. Lowest Bridge ID

2- Each remaining Switch will select ONE of its ports to be its ***root port*** (forwarding state).

Ports across from, ports connected to, the Root Port are always **designated ports**.

Root port selection:

1. Lowest root cost

2. Lowest neighbor Bridge ID

3. Lowest neighbor port ID

3- Each remaining collision domain will select ONE interface to be a **designated port** (in a forwarding state).

The other port in the collision domain will be **non-designated** (in a blocking state).

Designated Port selection:

1. Interface on the Switch with the lowest root cost.

2. Interface on the Switch with the lowest Bridge ID.

@0x0nullian