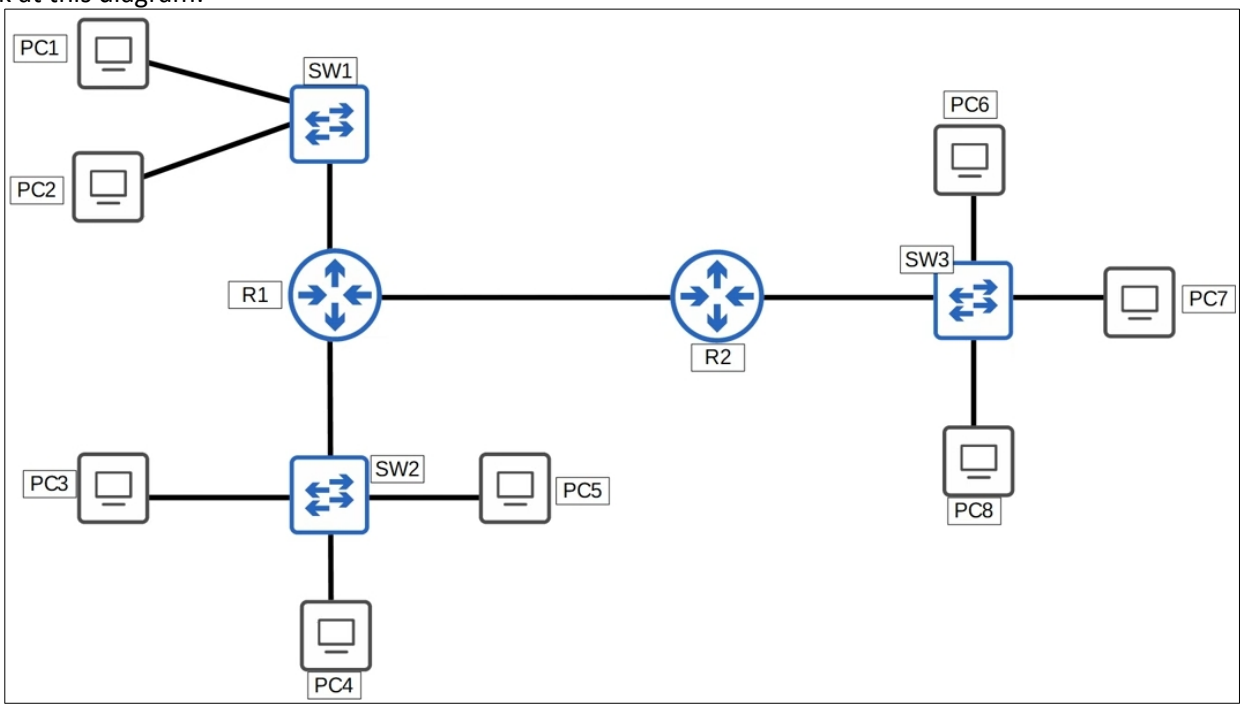
Day 16: VLANs pt.1:-

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:



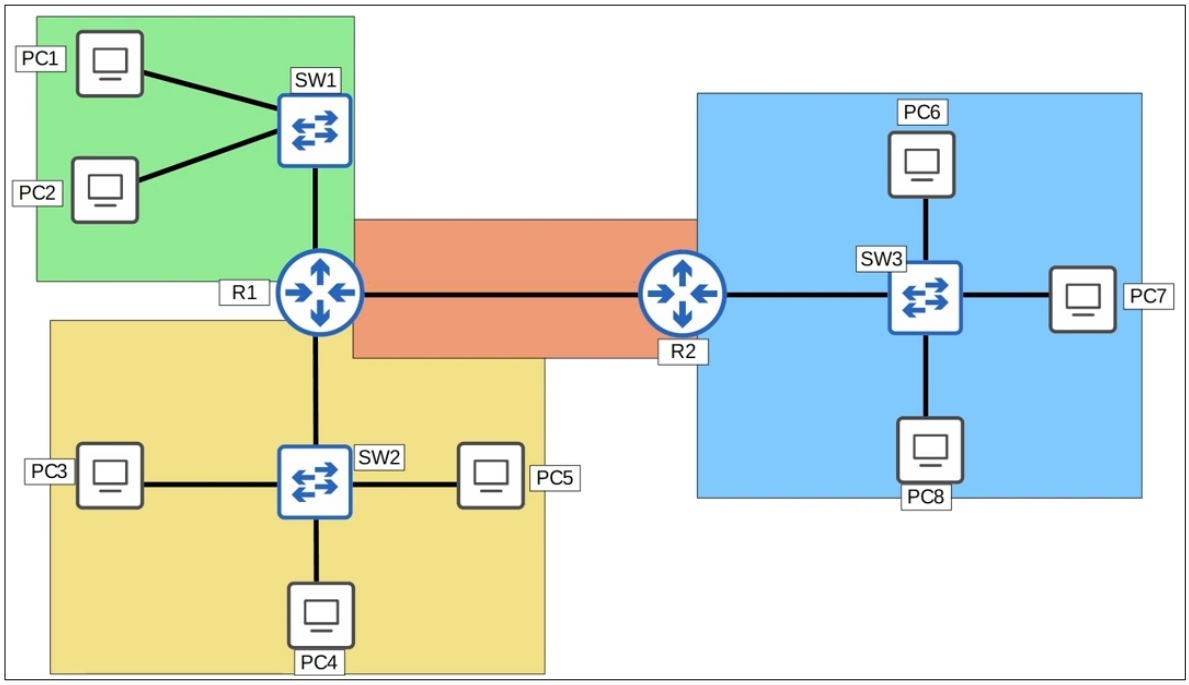
How many *broadcast domains* do you think there are?

- 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.

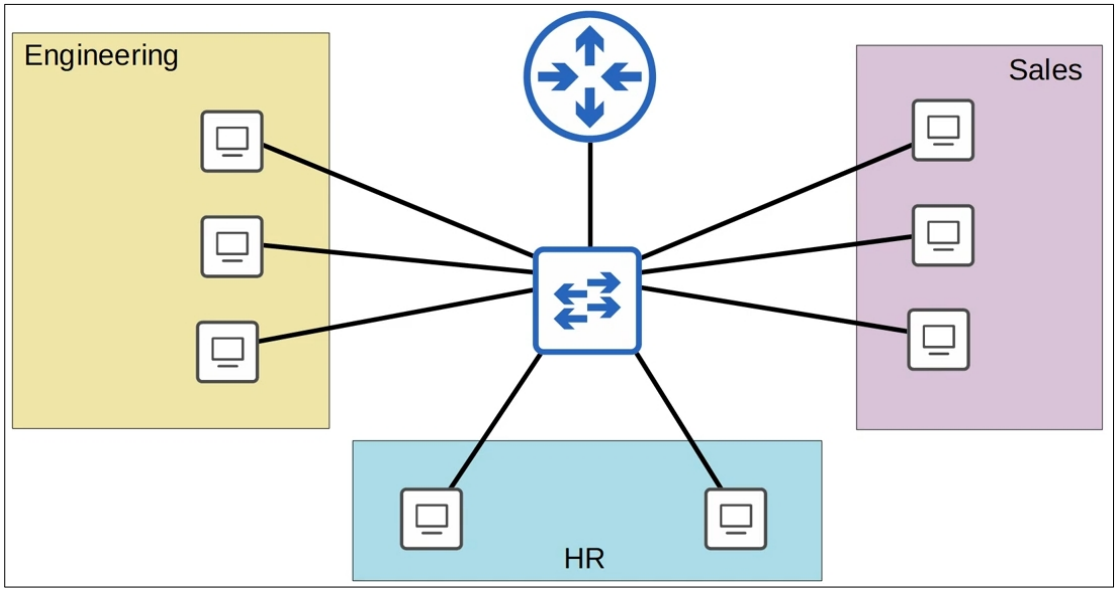


There are **4** broadcast domains in this diagram. And therefore, **4 LANS**.

→ 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.*

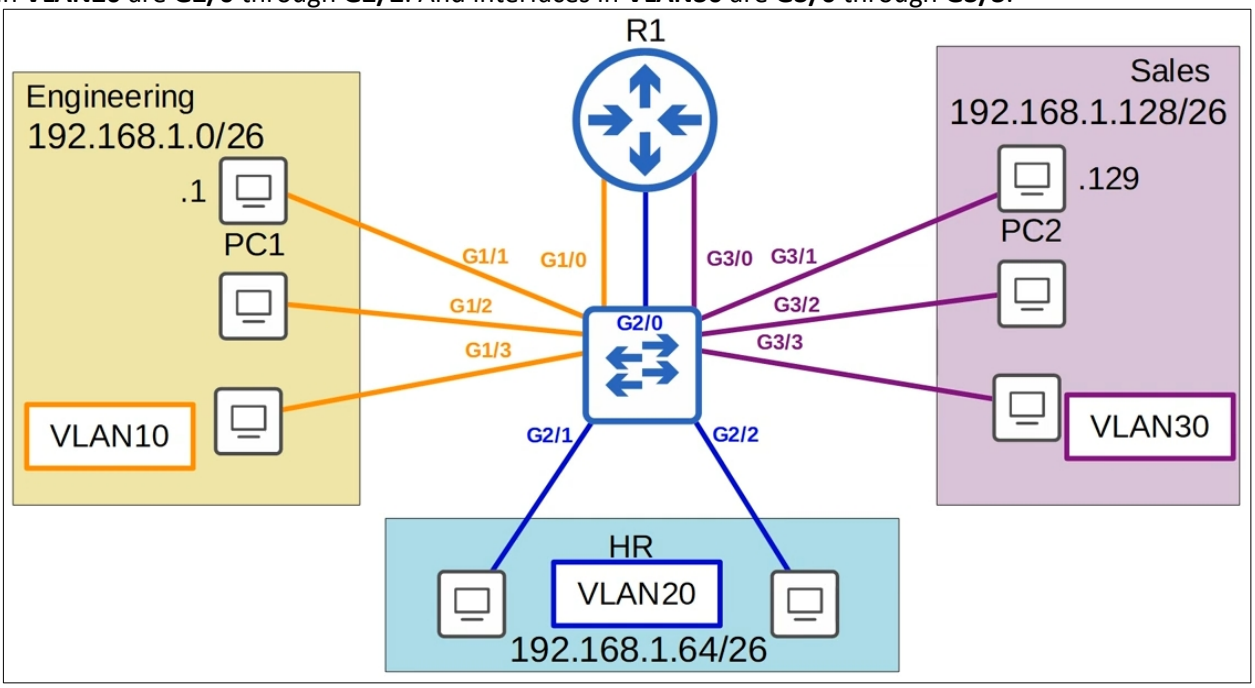
- First of all, the two hosts are in separate subnets, so **PC1** itself will send the traffic to its default gateway, **R1**.
- However, even if **PC1** and **PC2** were in the same subnet, the Switch wouldn't forward the traffic from **PC1** to **PC2**, because they are in separate **VLAN**s.

→ 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**.

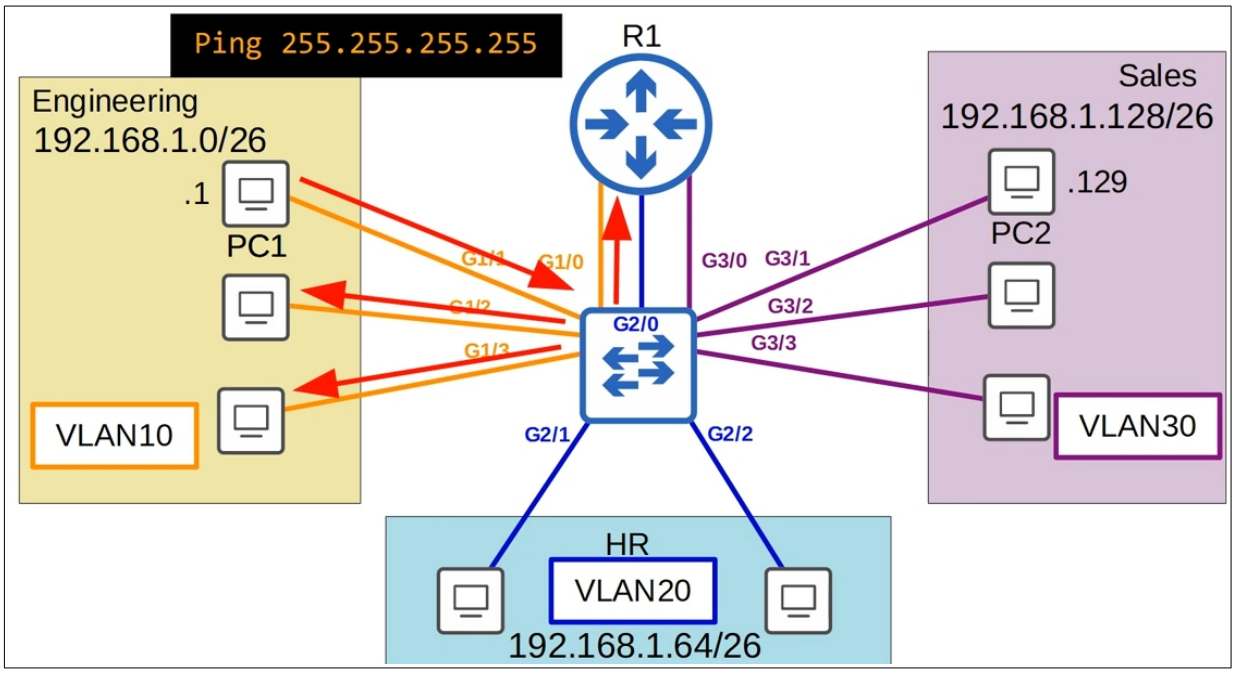
- Finally, We confirmed once more with (**`show vlan brief`**).

```
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**.



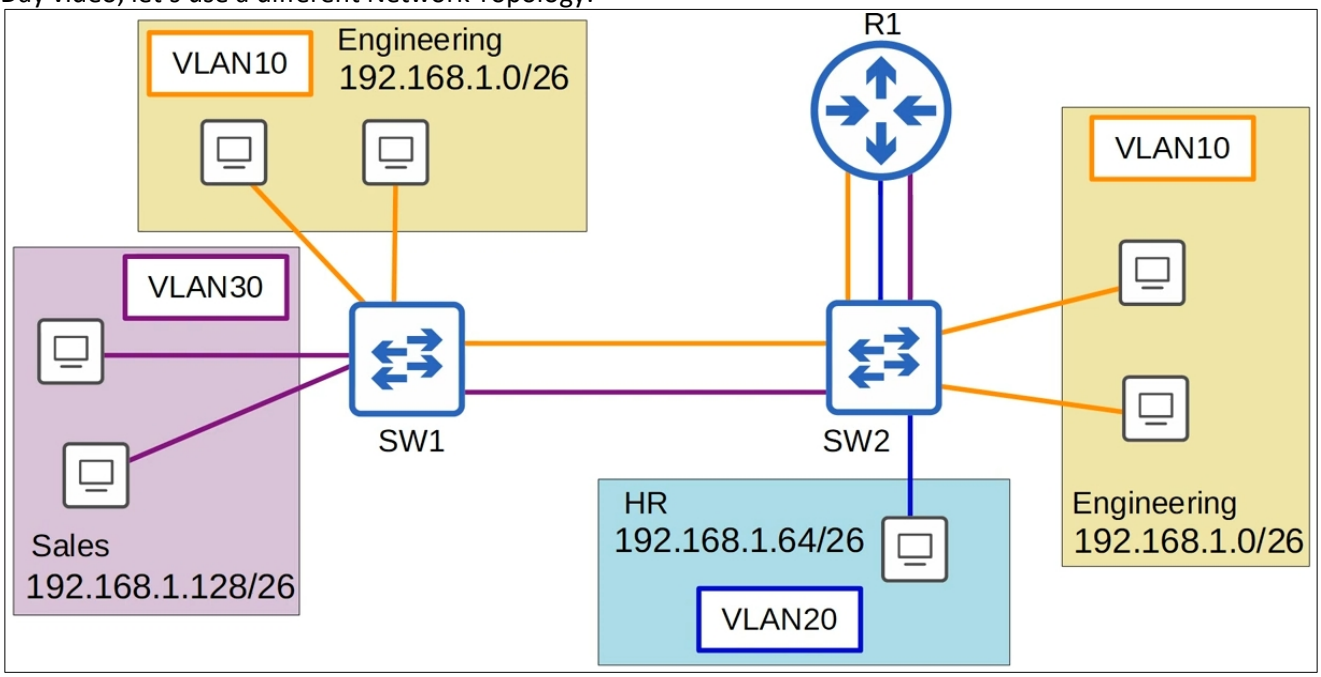
Day 17: VLANs pt.2:-

→ 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:

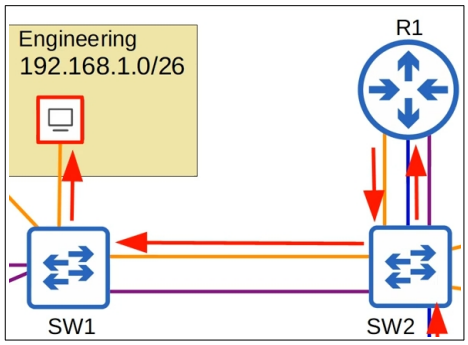


- This time there are **2** Switches used.
- Note that **VLAN10**, the VLAN for the Engineering dept, is split between the two Switches.
- This is very common, as departments in a company aren't always split up exactly by location.
  You might have some engineers on one floor of the building, for example,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

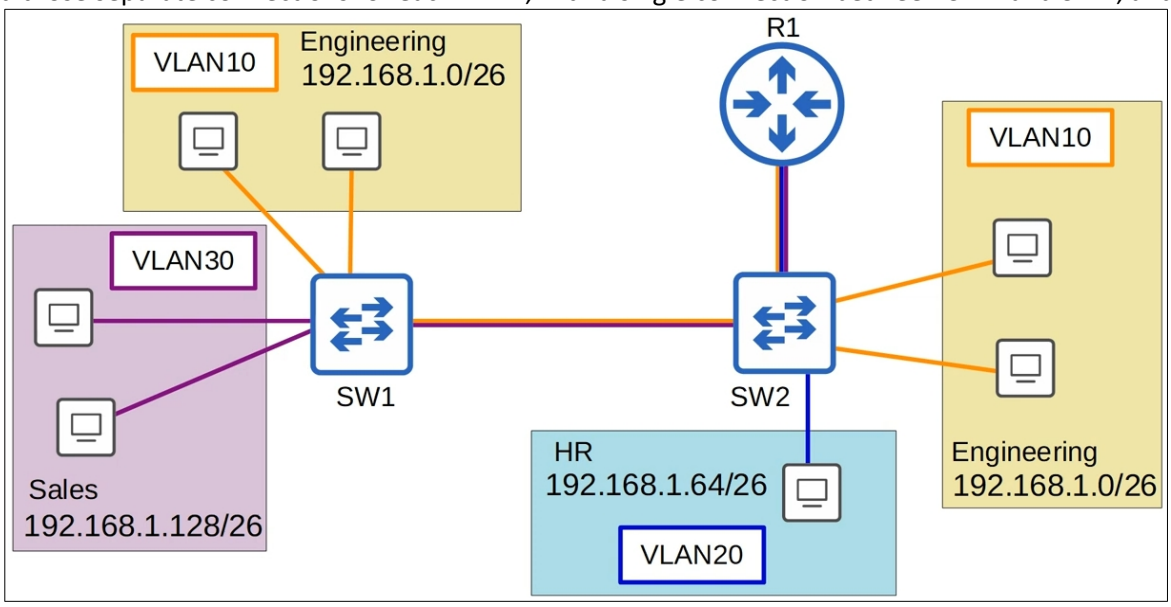
- In a small network with few VLANs, it is possible to use a separate interface for each VLAN when connecting Switches to Switches, and Switches to Routers.

- 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!

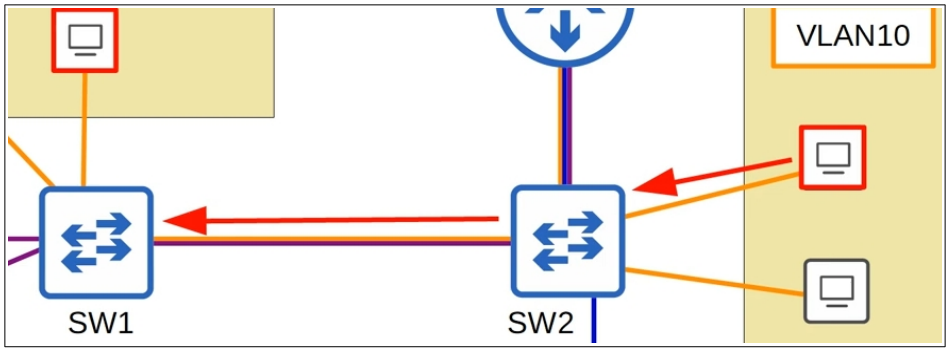
→ How Trunk Ports work?:-

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:

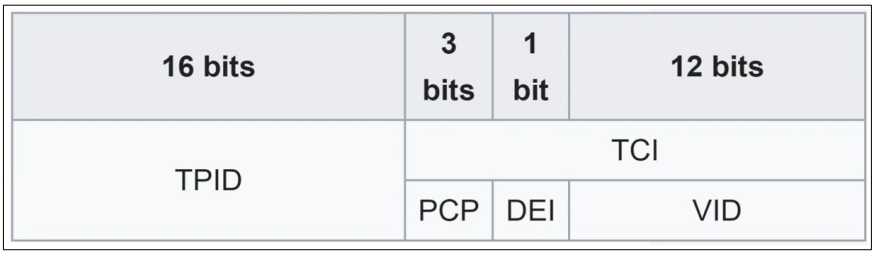
dot1Q inserts a **4**-byte, or **32**-bit field between two fields of this header.

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:-

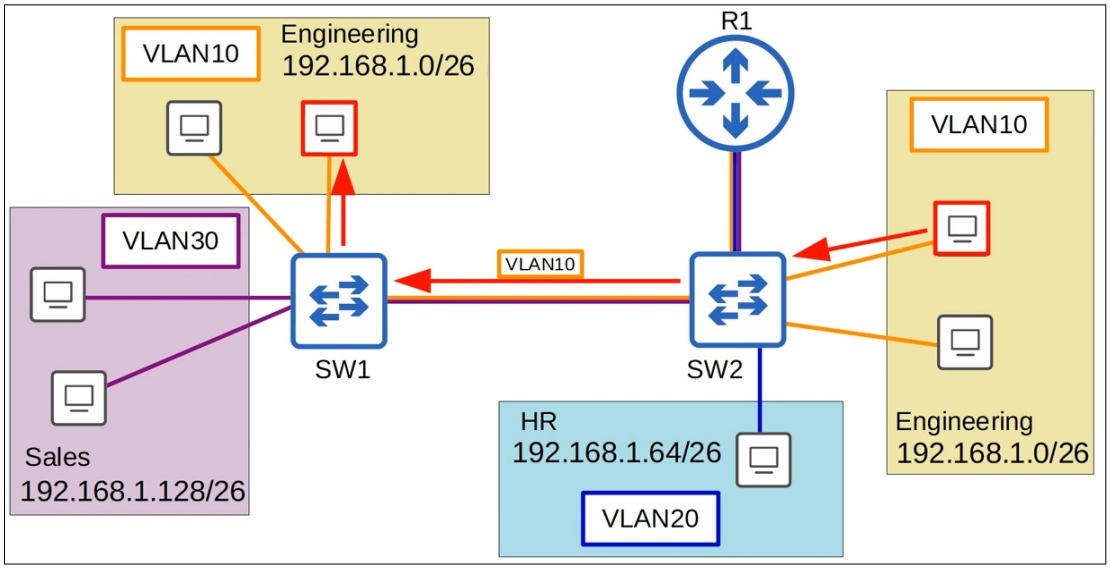
- Stands for **Priority Code Point**.
- **3** bits in length.
- Used for Class of Service (**CoS**), which prioritizes important traffic in congested networks.

→ 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

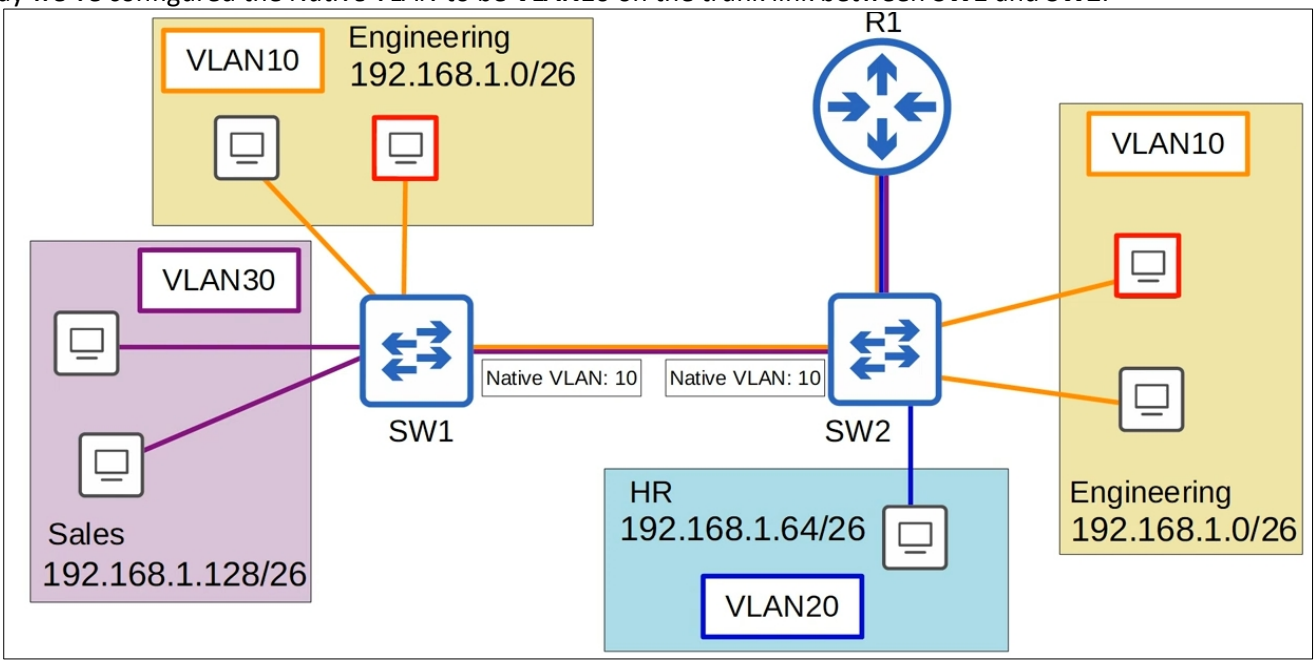


- A standard Layer2 Switch like this will only forward traffic in the same VLAN, It will not forward traffic between VLANs.

→ Native VLAN:-

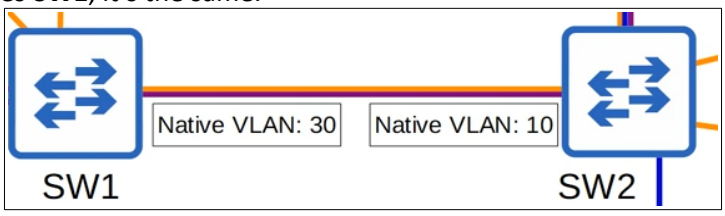
802.1Q has a feature called the *Native VLAN*. Cisco's ISL doesn't have this feature, BTW.
- The *Native VLAN* is **VLAN1** by default on all trunk ports, however this can be manually configured on each trunk port.
- ***Remember:*** This has to be configured on each trunk port separately, it's not a 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**.



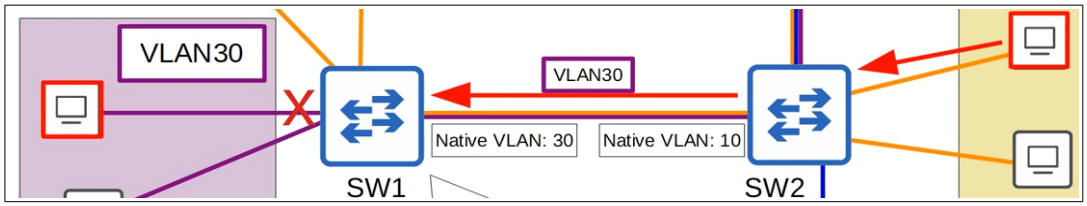
Let's follow some traffic on the same path as usual:
- The same PC from **VLAN10** sends the traffic to **SW2**.
- It will send the traffic to **SW1**, but because it is in the *Native VLAN***, VLAN10**, it won't tag it as being in **VLAN10**.
- The untagged frame arrive at **SW1**, which assumes that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

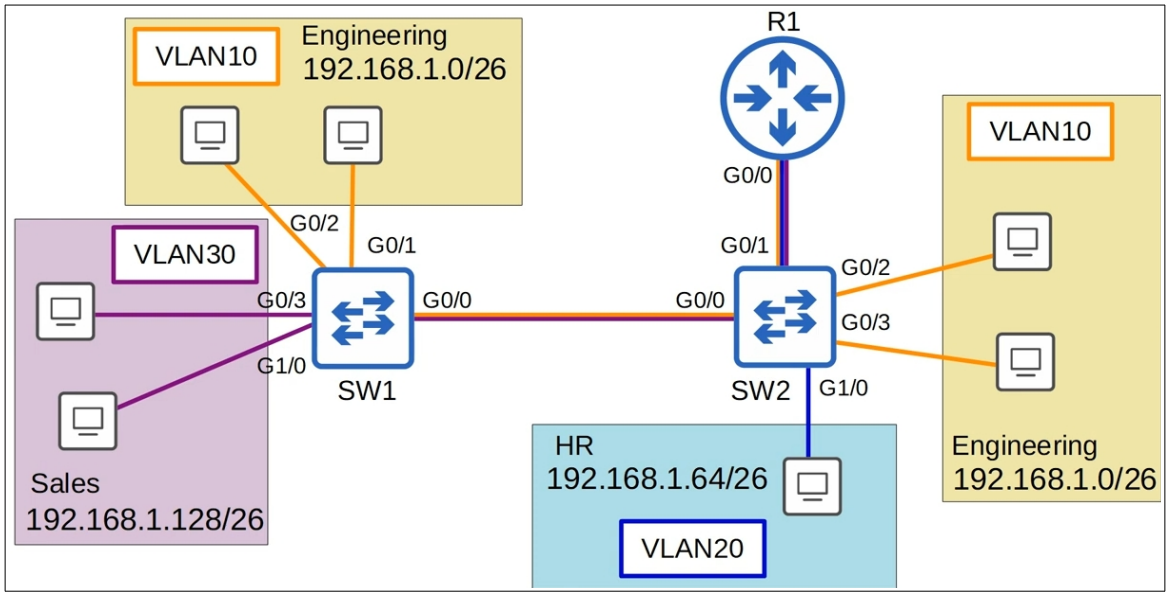
Let's look at another reason why it's important for the *Native VLANs* to match.
- This time, this PC in **VLAN10** wants to reach this PC in **VLAN30**.
- The PC sends the frame to **SW2**, which forwards it to **SW1** with a tag of **VLAN30**, since it's not the *Native VLAN* of **SW2**.
- However, **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*!

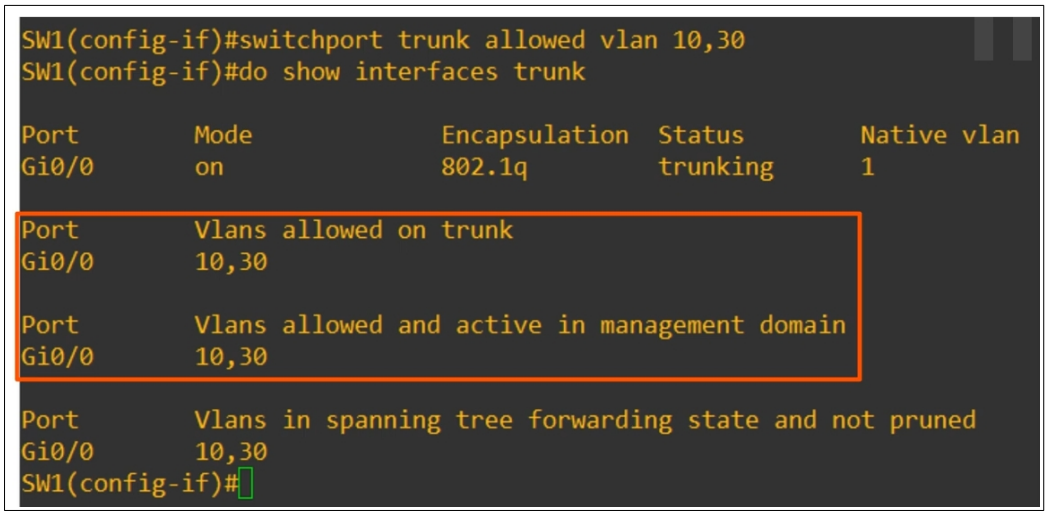
```
SW1#show vlan brief

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#
```

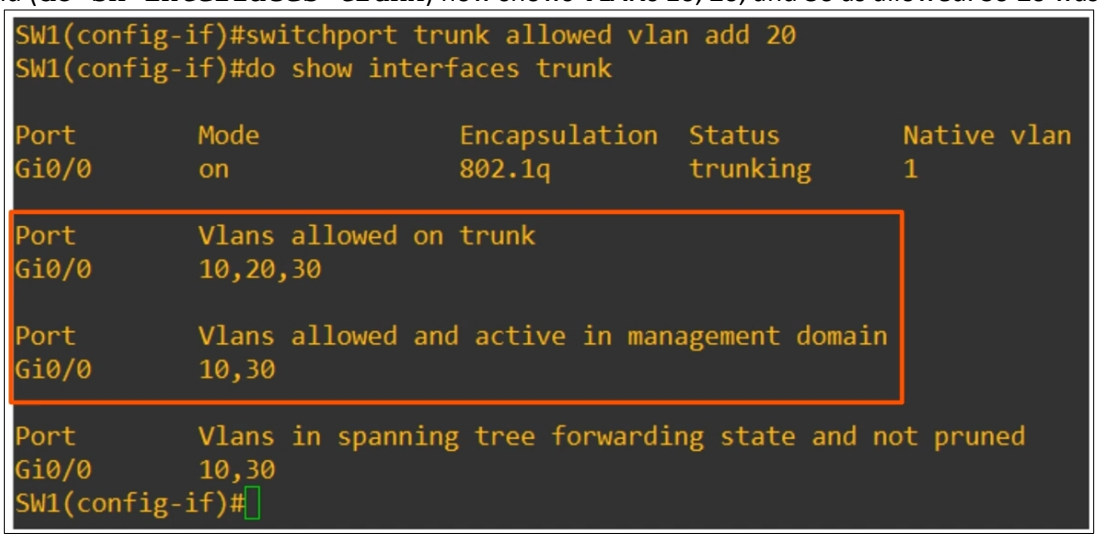
- The last field of the (**`show interfaces trunk`**) command is '*Vlans in spanning tree forwarding state and not pruned*'.

→ 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)#
```

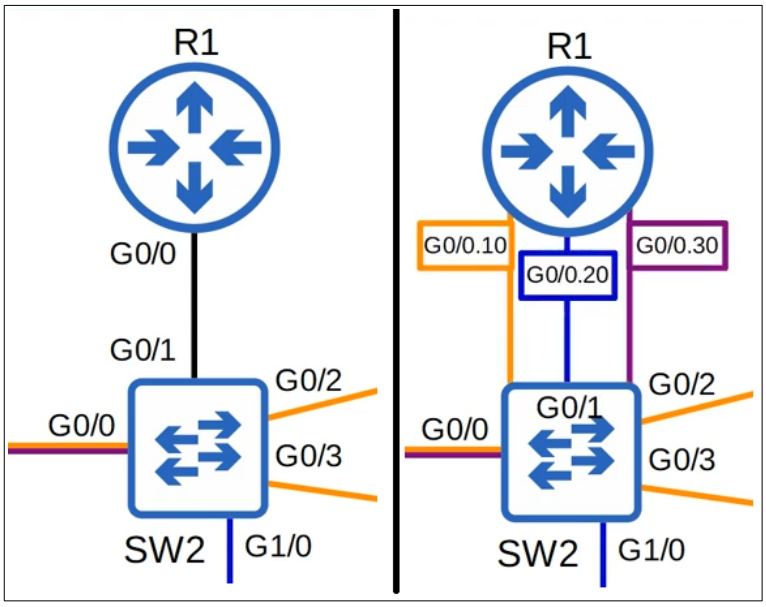
- Almost identical to **Gi0/0**, except we allowed **VLAN20** in addition to **VLAN10** and **VLAN30**.
- Now, both **Gi0/0** and **Gi0/1** are displayed in the output of the (**`show interfaces trunk`**) command.
- That's all for the Switch configurations for this lesson!

@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#
```

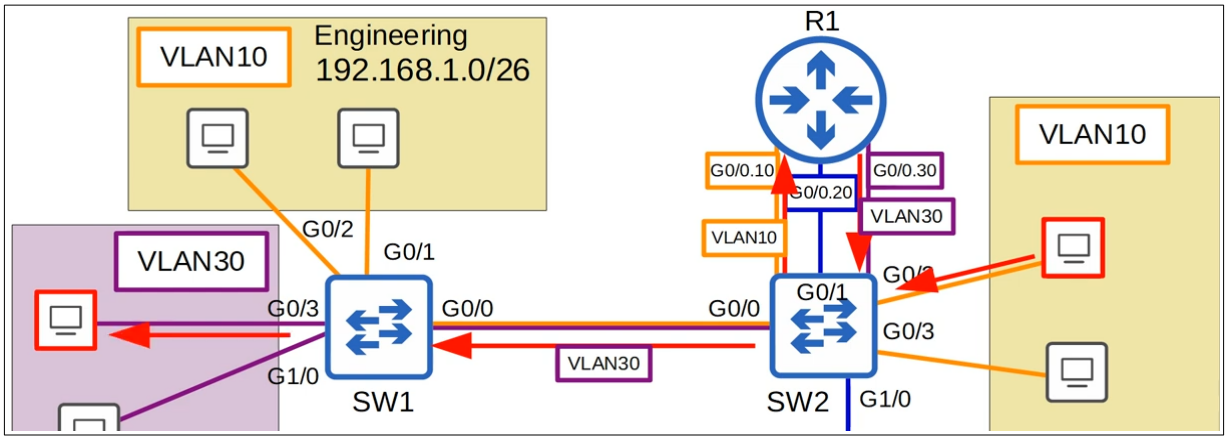
- Notice the connected and local routes are added just when IP addresses are added to regular physical interfaces.
- When **R1** sends frames out of these *sub-interfaces*, it adds the VLAN tag configured on the *sub-interface*.
- For example, if a packet arrives destined for the **192.168.1.64/26**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

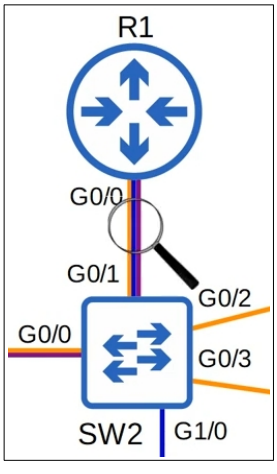
1- First, We will configure the first option:-
- On the **G0/0.10** interface, we configured (**`encapsulation dot1q 10 native`**)

<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)#
```

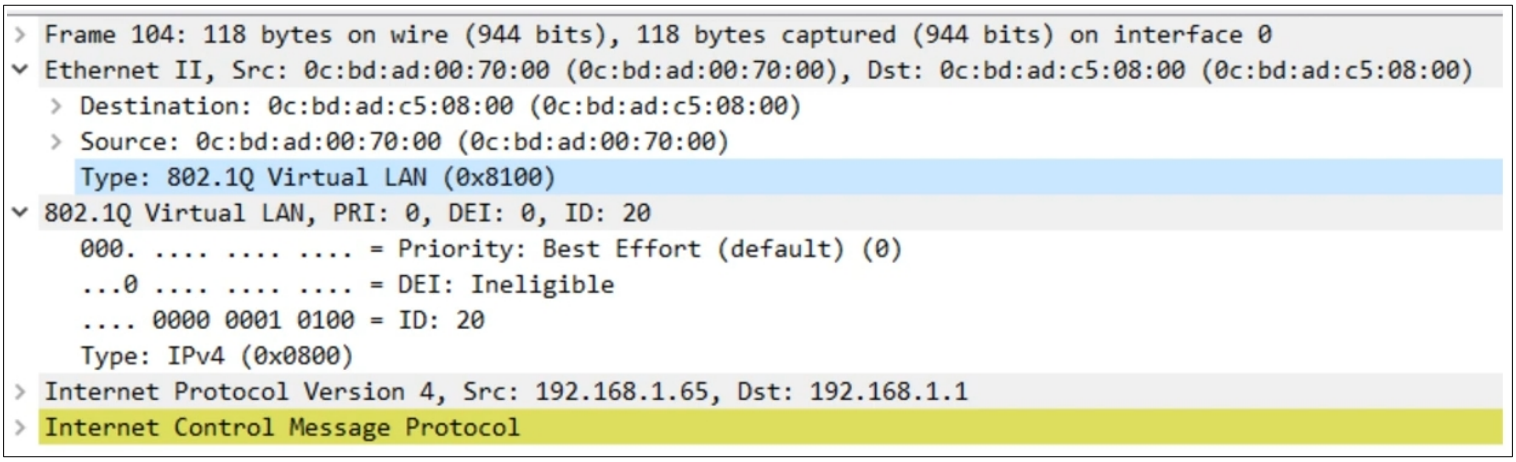
→ Wireshark Packet Capture:-

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*.

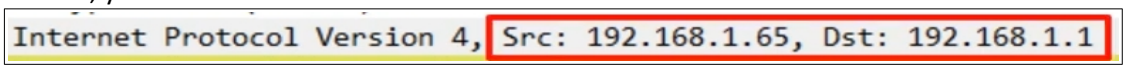


→ Wireshark Capture (SW2 → R1):-

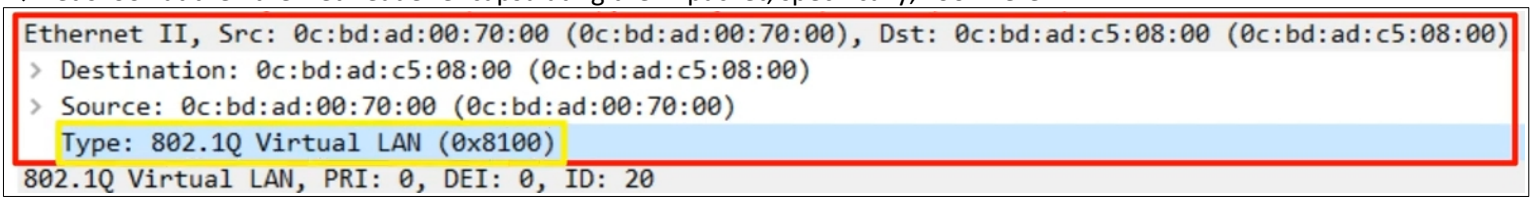
Here's the Wireshark capture for the **ICMP echo request** as it goes from **SW2** to **R1**:-



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

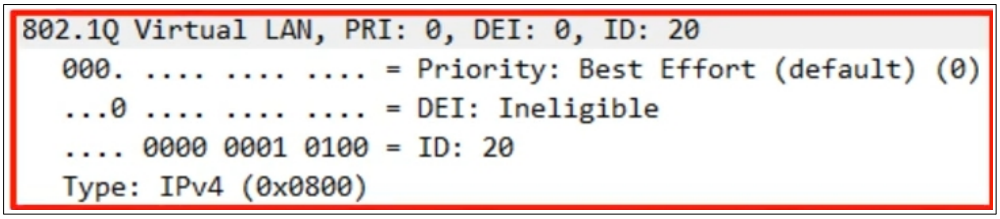


→ Let's look at the Ethernet header encapsulating the IP packet, specifically, Look here:



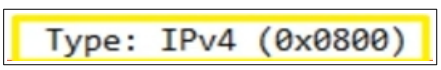
- Type: **802.1q** Virtual LAN, and notice the hexadecimal **8100** value.
- In the previous lecture, **dot1q** is inserted after the source MAC address field, and that is where TYPE field usually goes.
- This here is the '**TPID**' field of the **dot1q** tag.

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**.

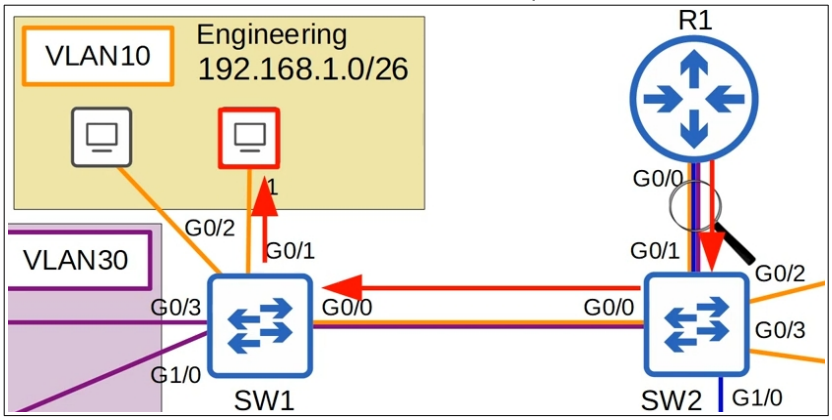
What's different??

```
> 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.



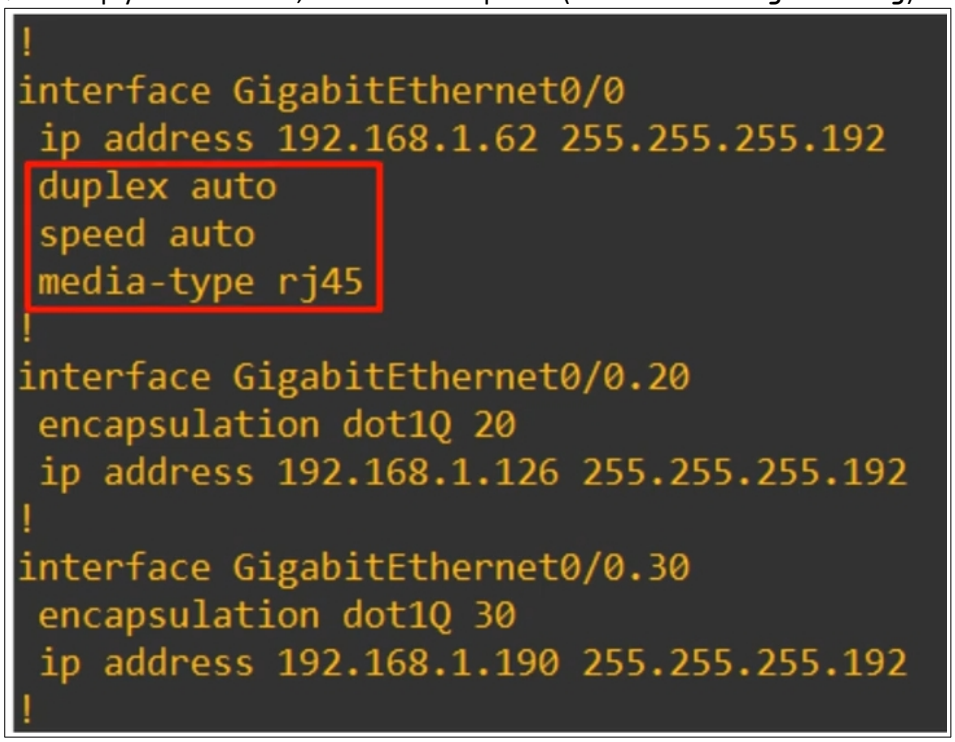
→ Configuring the Native VLAN on a Router:-

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.

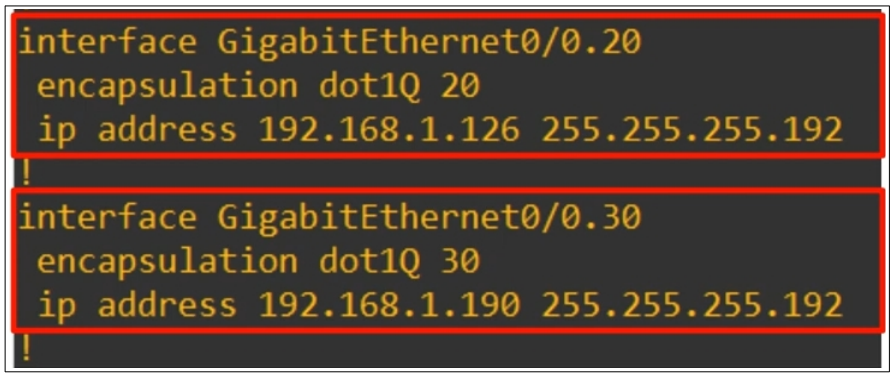
```
R1(config)#no interface g0/0.10
R1(config)#interface g0/0
R1(config-if)#ip address 192.168.1.62 255.255.255.192
R1(config-if)#
```

→ 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.

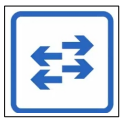
- This will function just like the first option we saw.
- **SW2** will send **VLAN10** packets in untagged frames to **R1**, and **R1** will send them in untagged frames also.

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:-

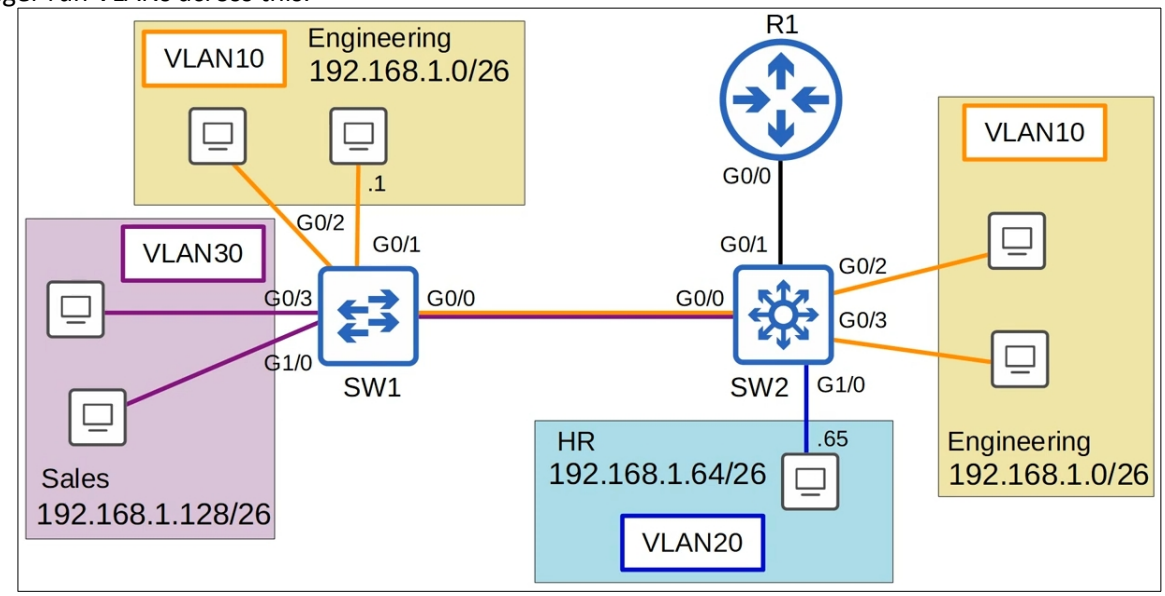
- A MultiLayer Switch is capable of both *Switching* and *Routing*.
- It is '*Layer3 aware*'.
  A regular Layer2 Witch is NOT Layer3 aware, It doesn't at all about IP addresses or anything above Layer2.
  It only cares about Layer2 information like MAC addresses.
- We can assign IP addresses to its interfaces 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:-

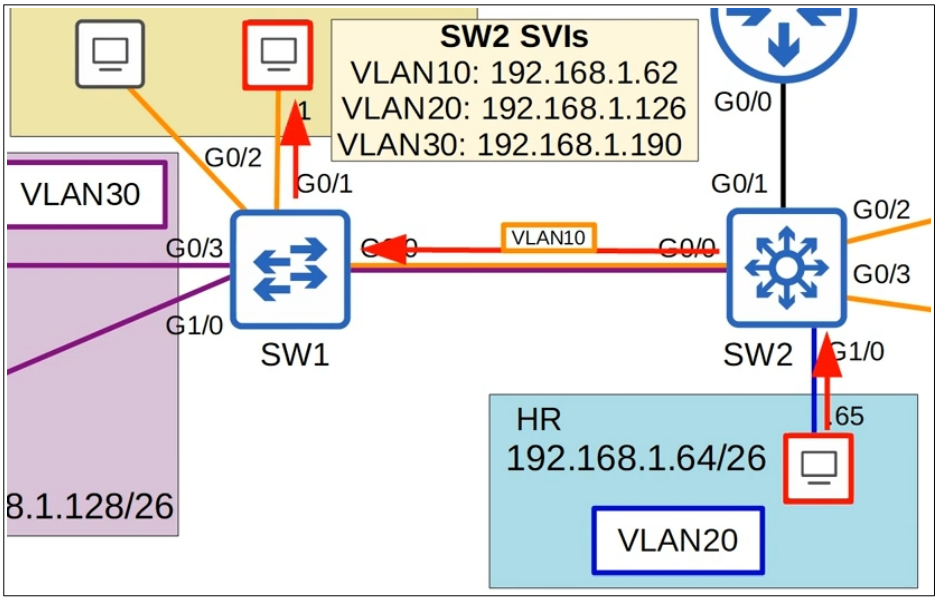
→ Switch Virtual Interfaces:-
- **SVI**s are the Virtual interfaces you can assign IP addresses to in a Multi-Layer Switch.
- Configure each PC to use the **SVI** (NOT the Router) as their gateway address.
- When using **ROAS**, the Router was used as the PC's gateway. This time, we will use the Switch's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

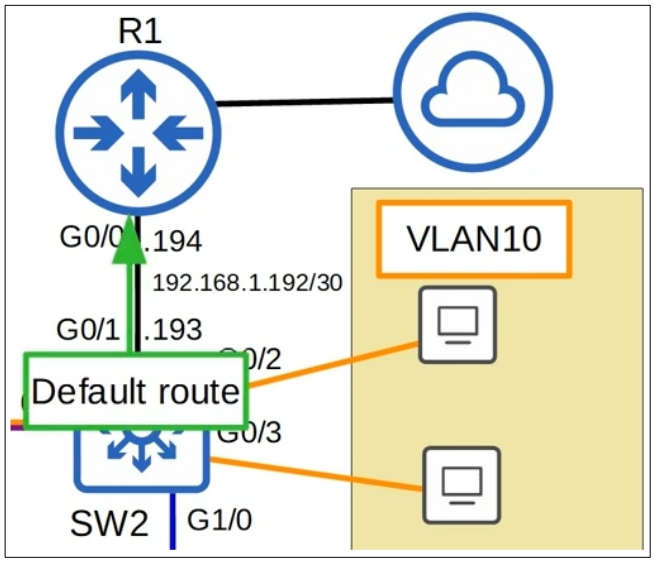
SW2 SVIs
VLAN10: 192.168.1.62
VLAN20: 192.168.1.126
VLAN30: 192.168.1.190

- 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.

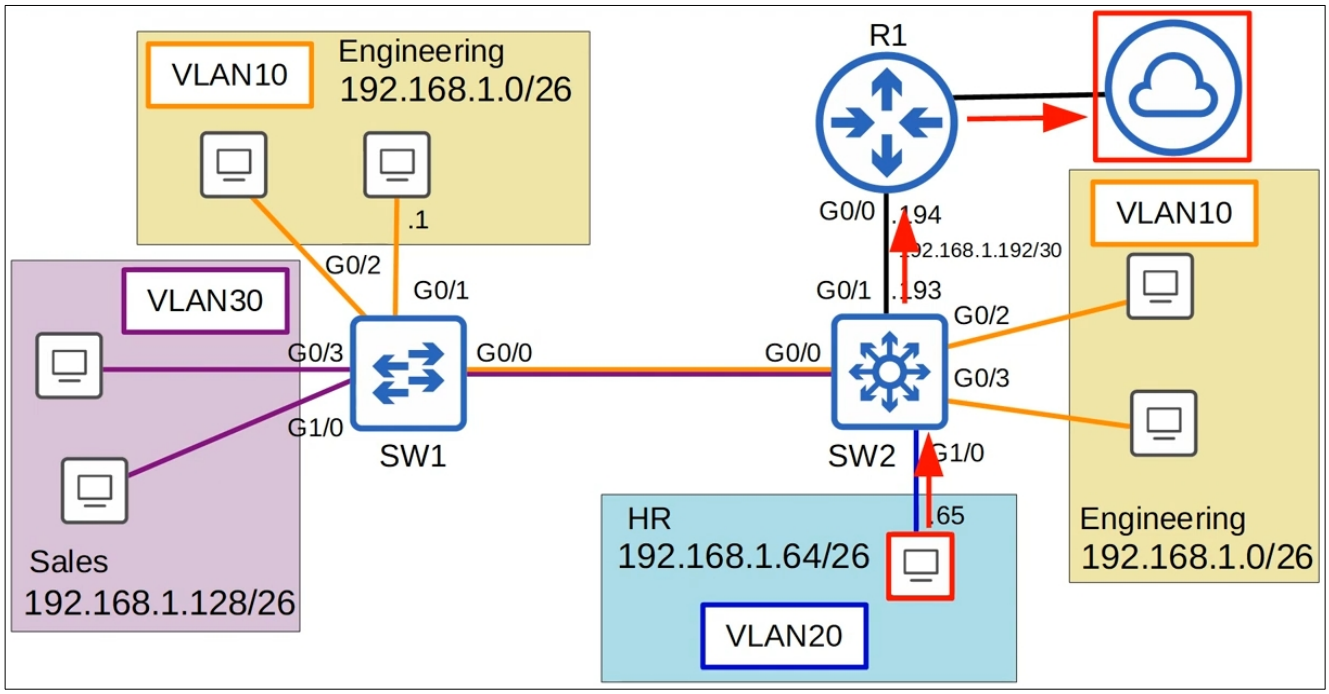
→ We used the (**`do show ip route`**) command again, and we can see **connected** and **local** routes have been added to the route table for the **SVI**s we created, all shown as directly connected to the **SVI** for each VLAN.

```
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)#
```

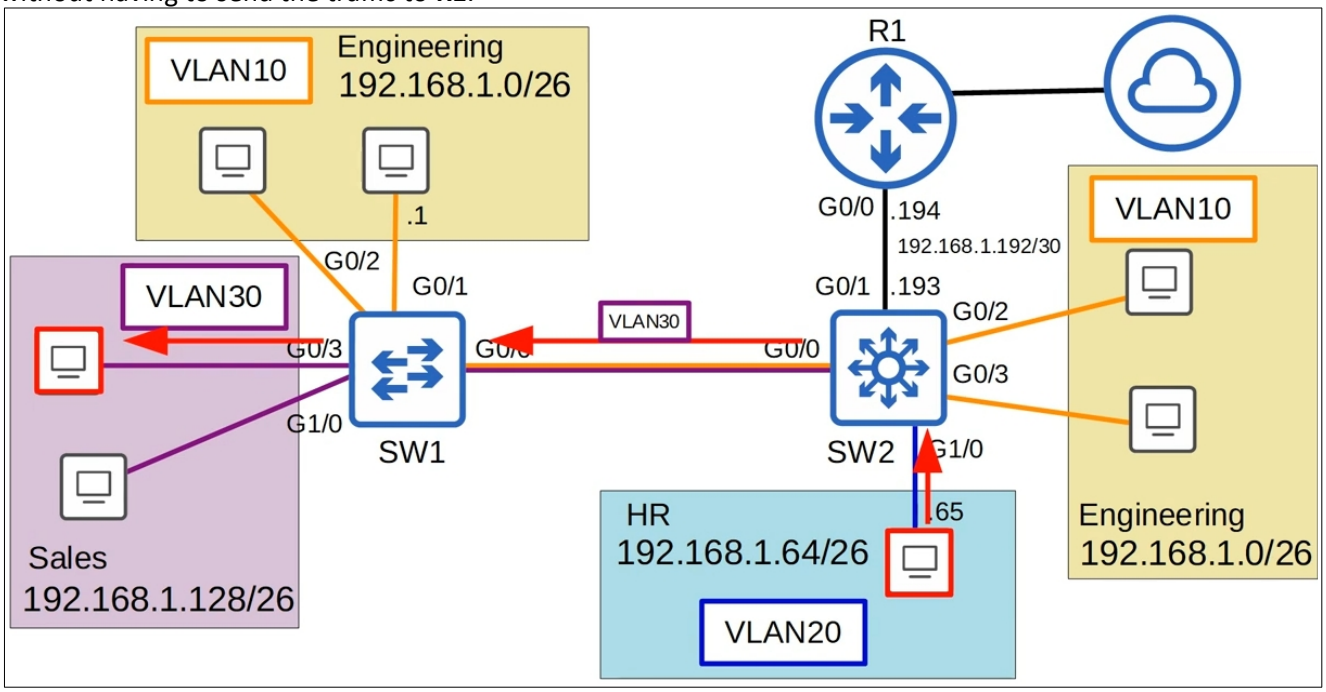
Okay, so our configurations are all done ;)

→ 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