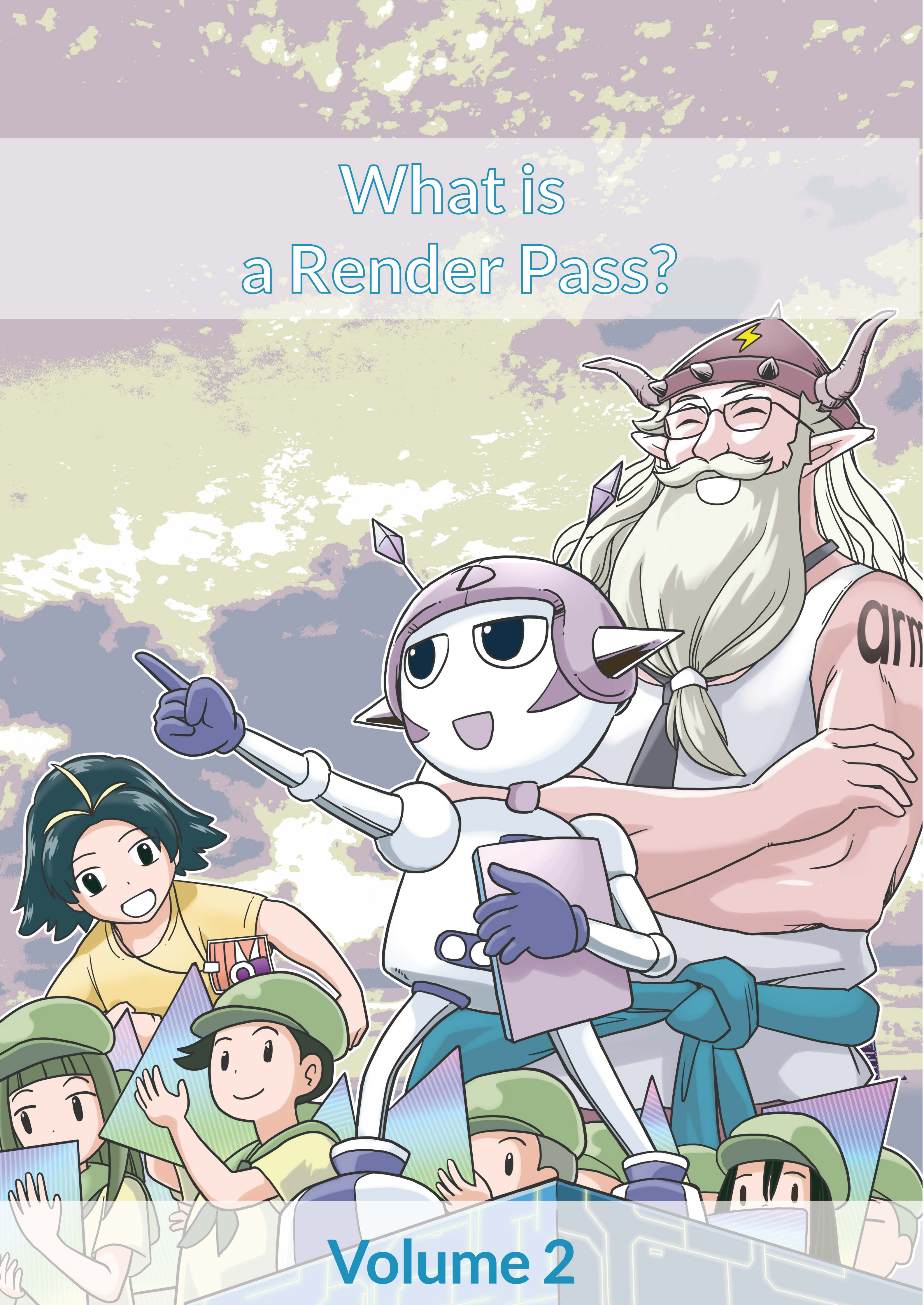
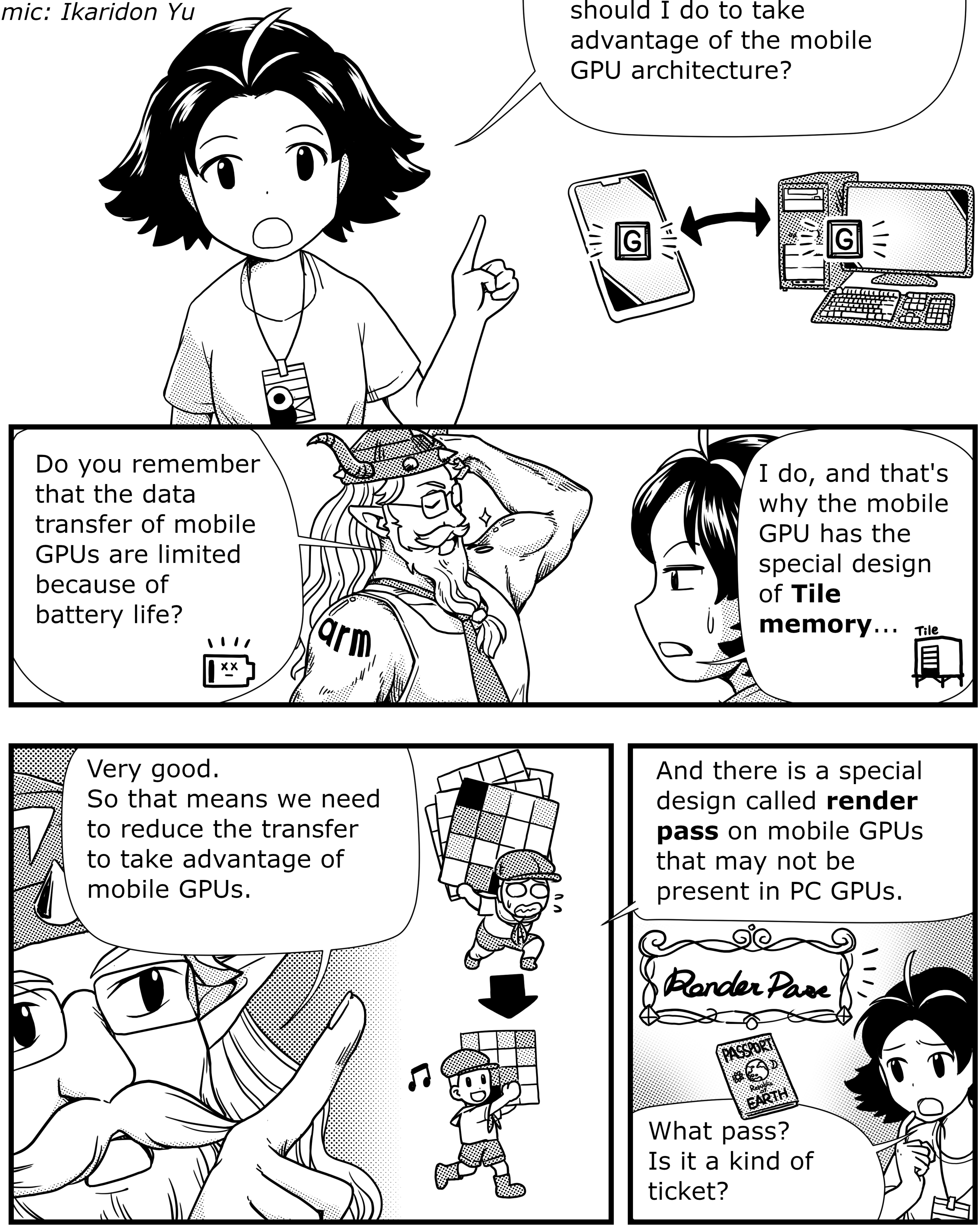
What is a Render Pass?

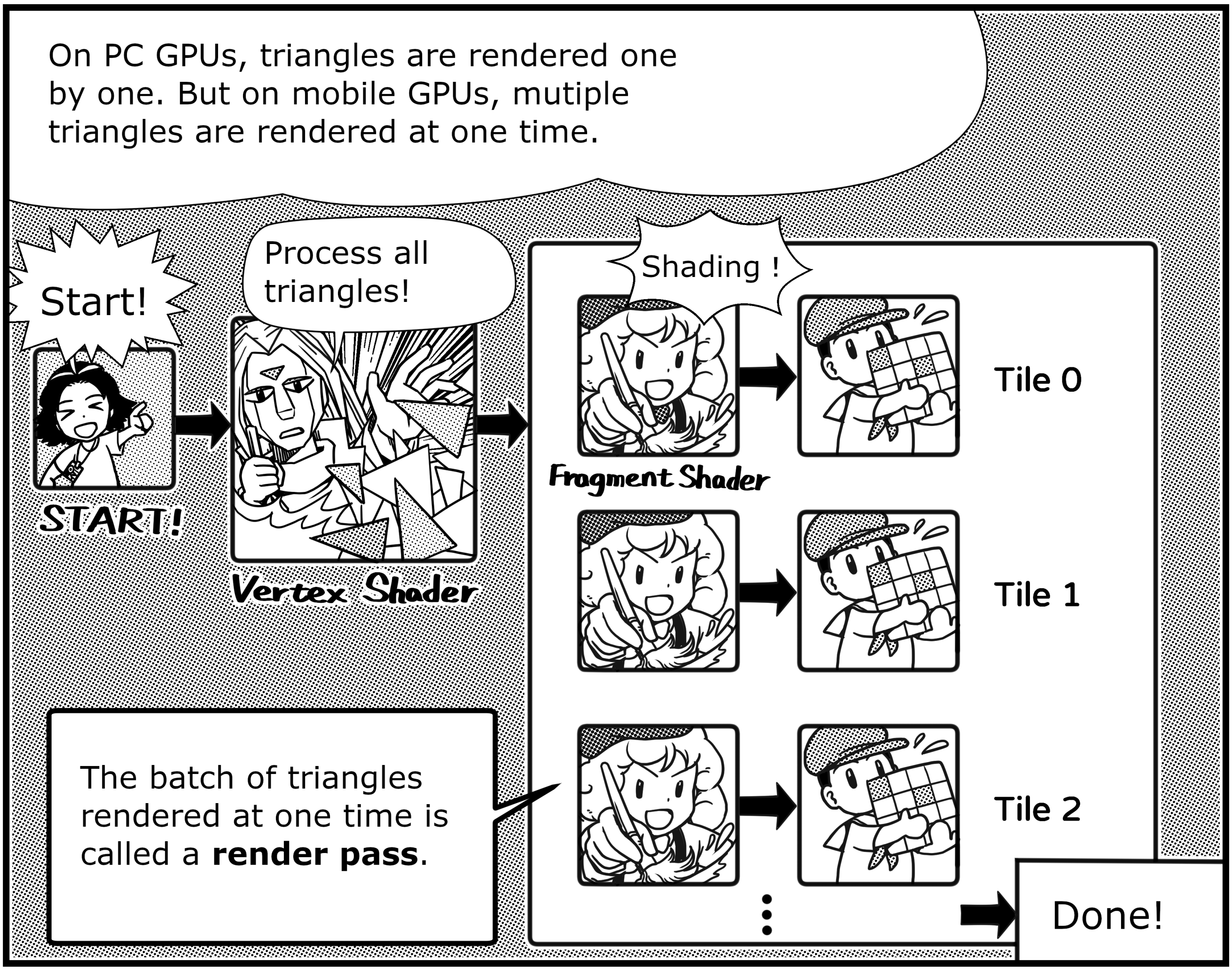


Volume 2

What is a Render Pass?

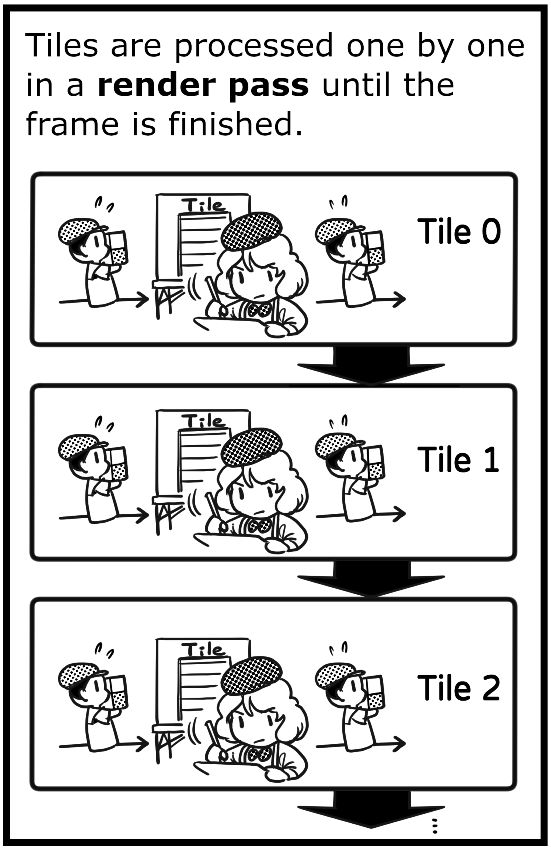
Comic: Ikaridon Yu


I have learned the difference between PC and mobile GPUs. So what should I do to take advantage of the mobile GPU architecture?
G
G
Do you remember that the data transfer of mobile GPUs are limited because of battery life?
arm
I do, and that's why the mobile GPU has the special design of **Tile memory**...
Tile
Very good. So that means we need to reduce the transfer to take advantage of mobile GPUs.
And there is a special design called **render pass** on mobile GPUs that may not be present in PC GPUs.
Render Pass
PASSPORT
EARTH
What pass? Is it a kind of ticket?

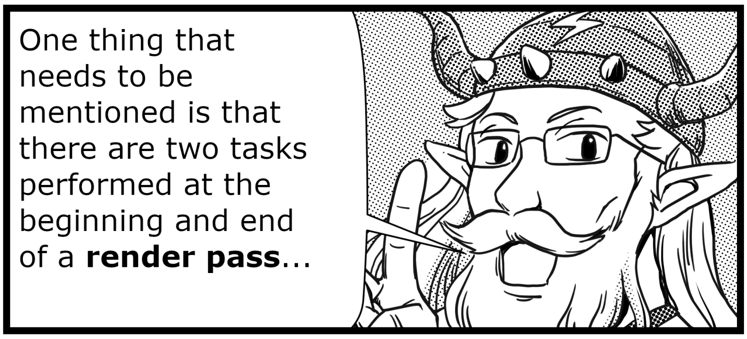


On PC GPUs, triangles are rendered one by one. But on mobile GPUs, mutiple triangles are rendered at one time.

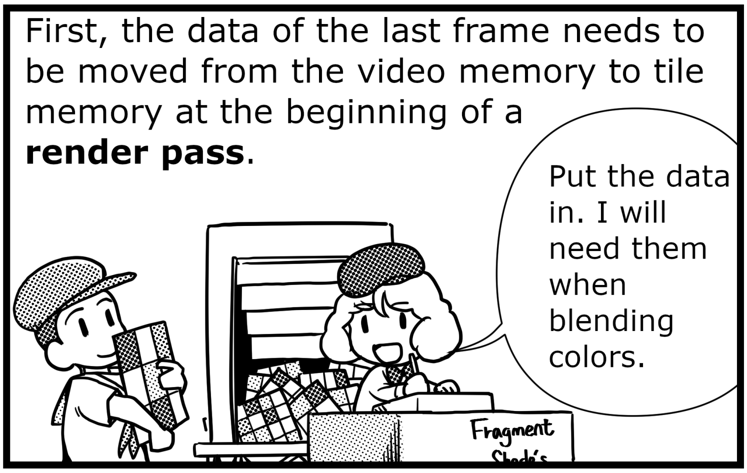
The batch of triangles rendered at one time is called a **render pass**.



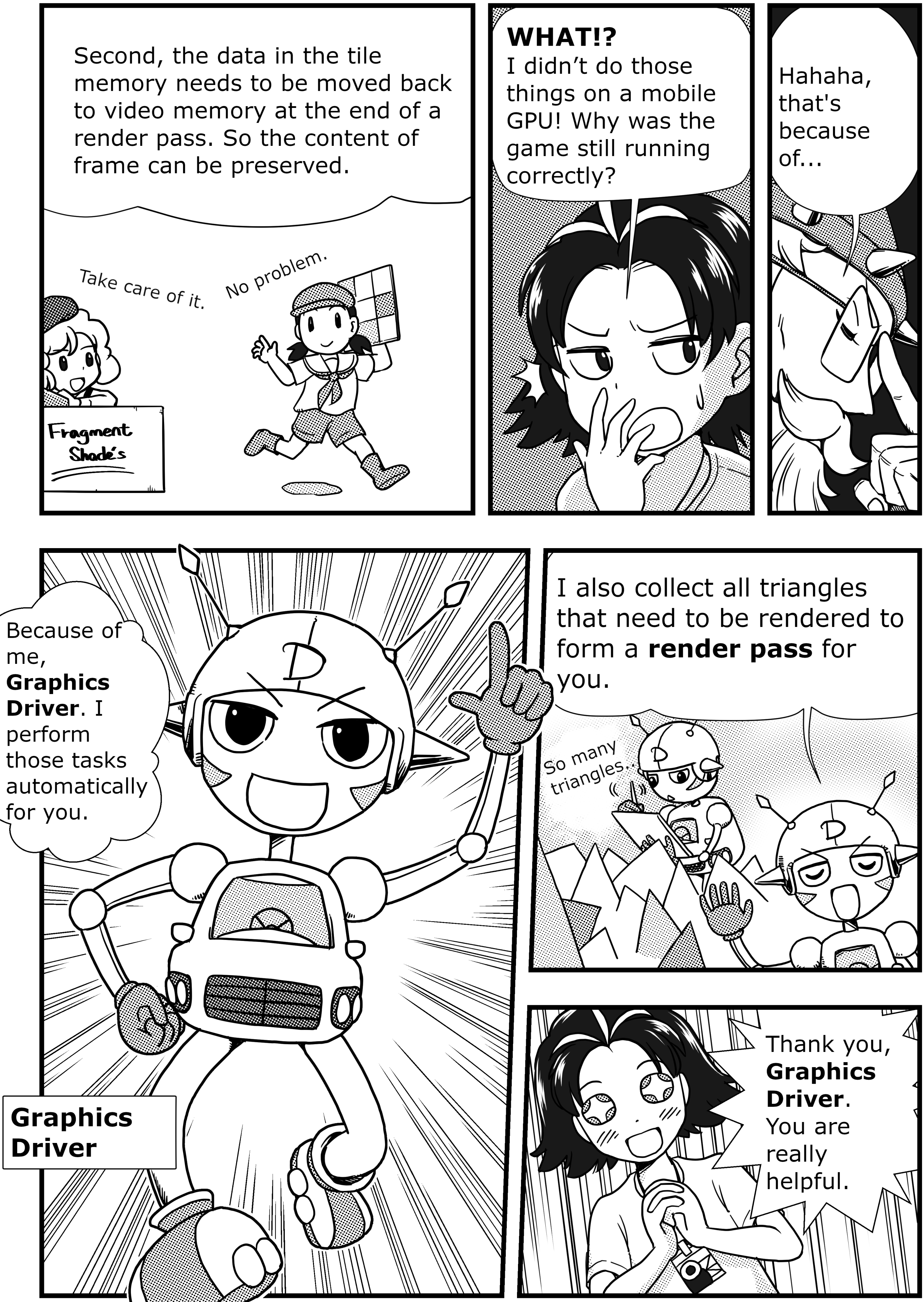
Tiles are processed one by one in a **render pass** until the frame is finished.

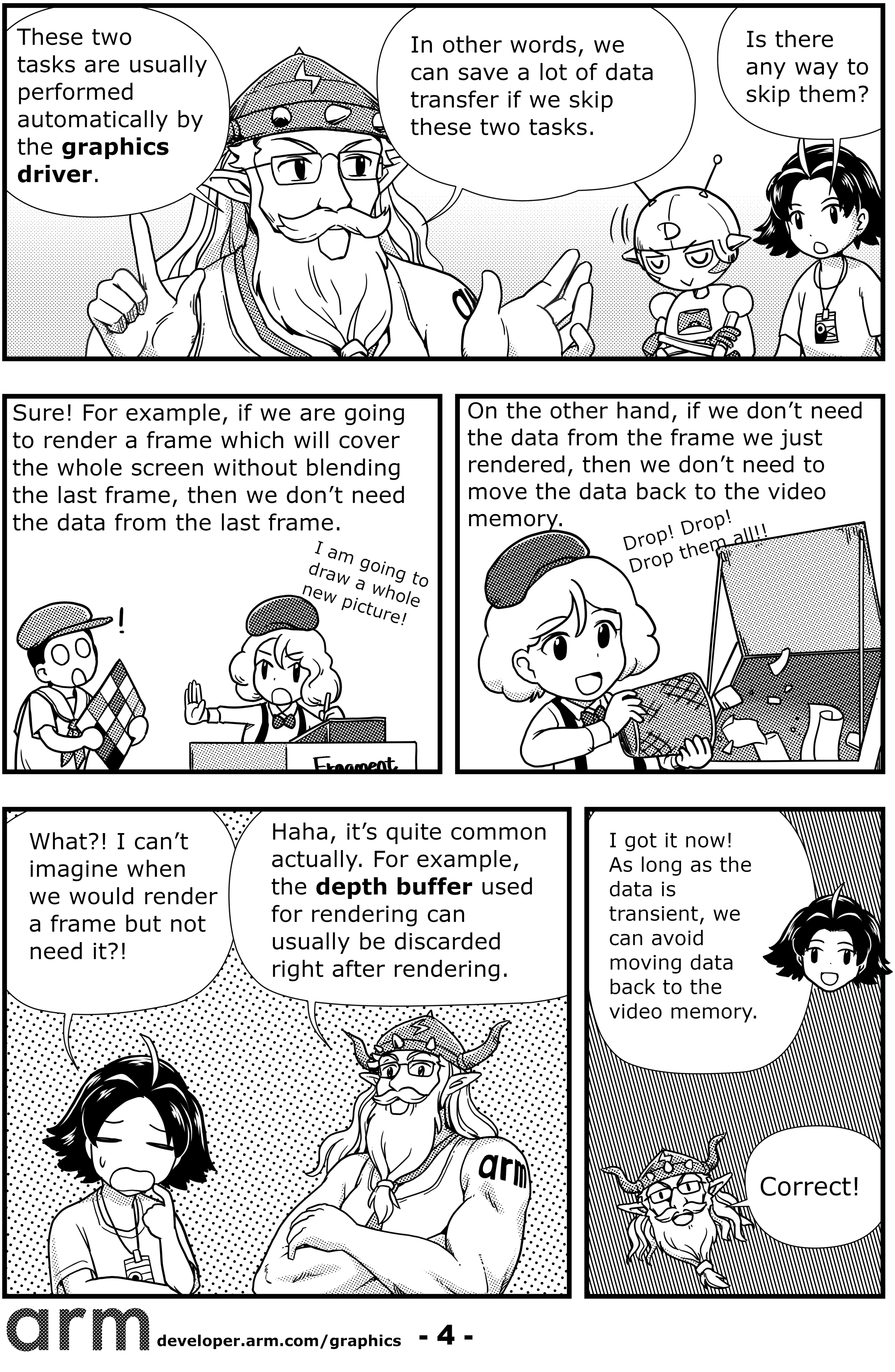


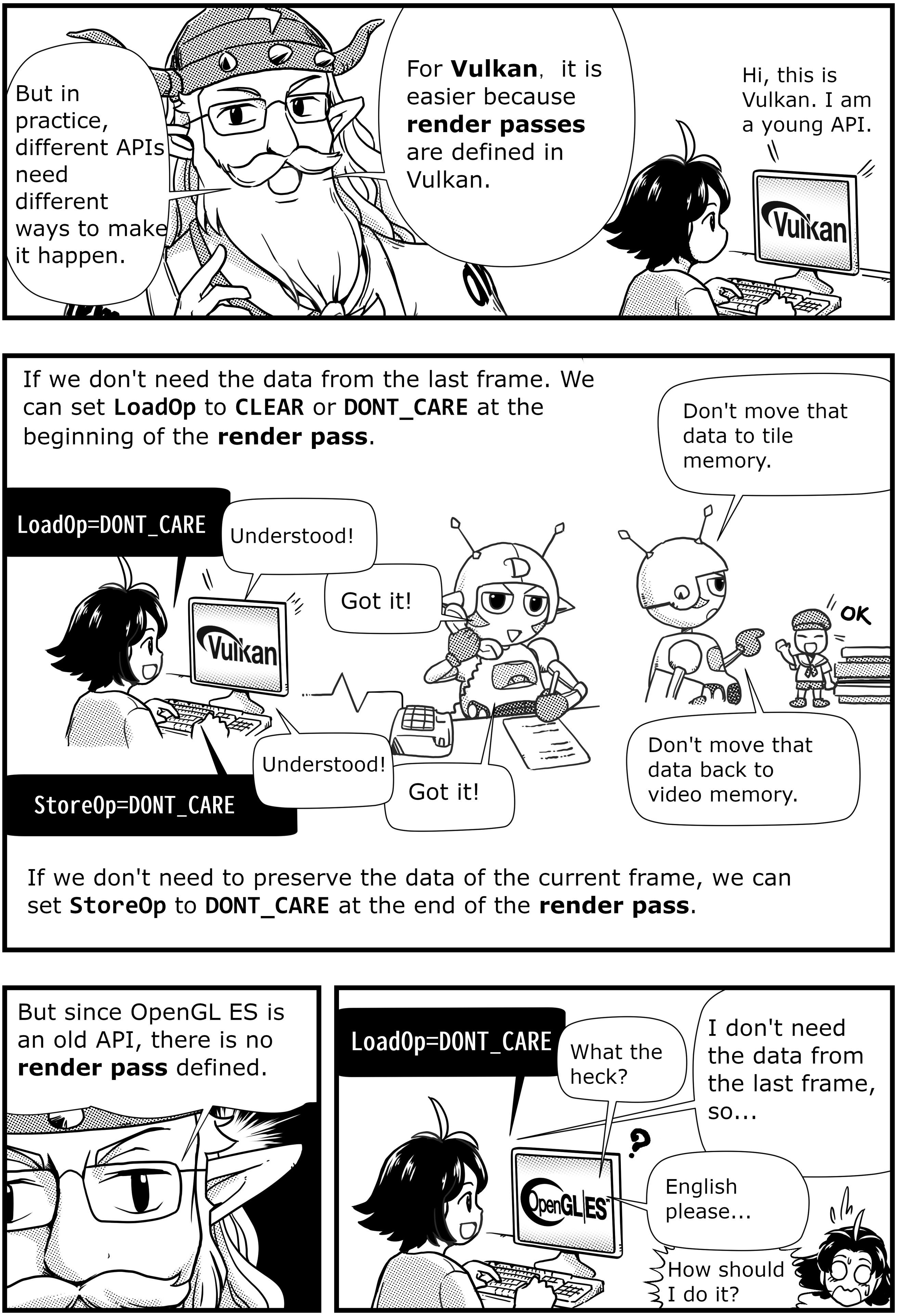
One thing that needs to be mentioned is that there are two tasks performed at the beginning and end of a **render pass**...

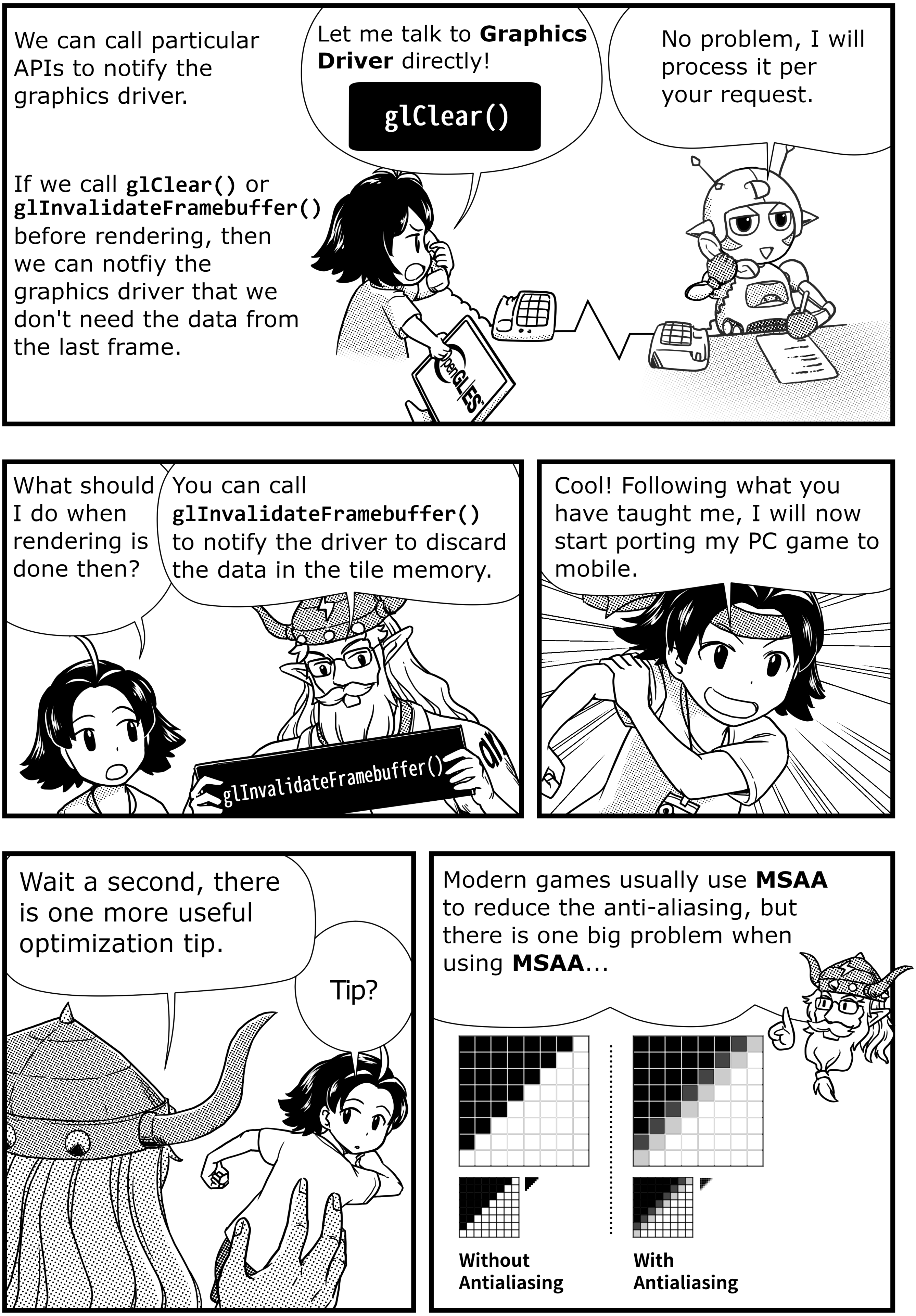


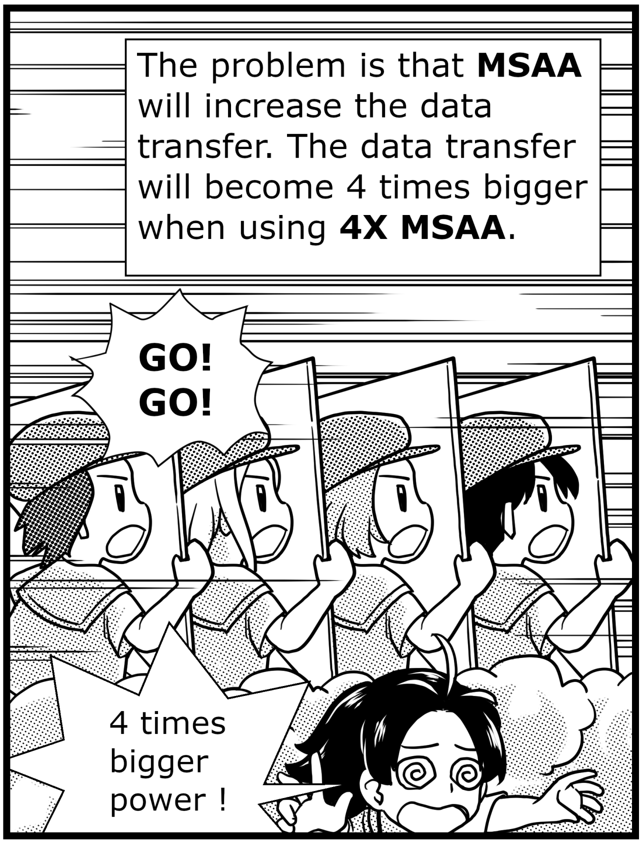
First, the data of the last frame needs to be moved from the video memory to tile memory at the beginning of a **render pass**.

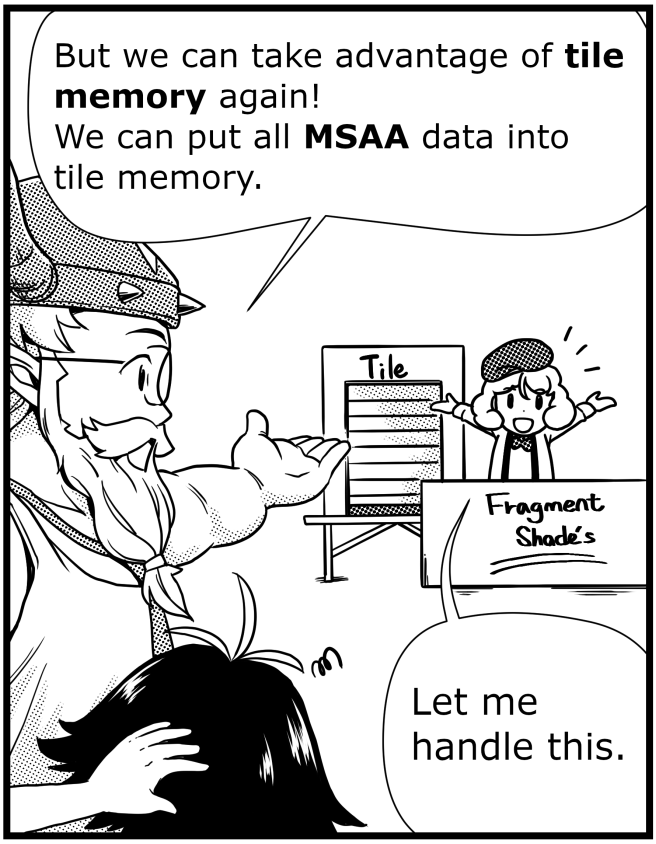

Second, the data in the tile memory needs to be moved back to video memory at the end of a render pass. So the content of frame can be preserved.
Take care of it.
No problem.
Fragment Shade's
WHAT!? I didn't do those things on a mobile GPU! Why was the game still running correctly?
Hahaha, that's because of...
Because of me, **Graphics Driver**. I perform those tasks automatically for you.
Graphics Driver
I also collect all triangles that need to be rendered to form a **render pass** for you.
So many triangles...
Thank you, **Graphics Driver**. You are really helpful.

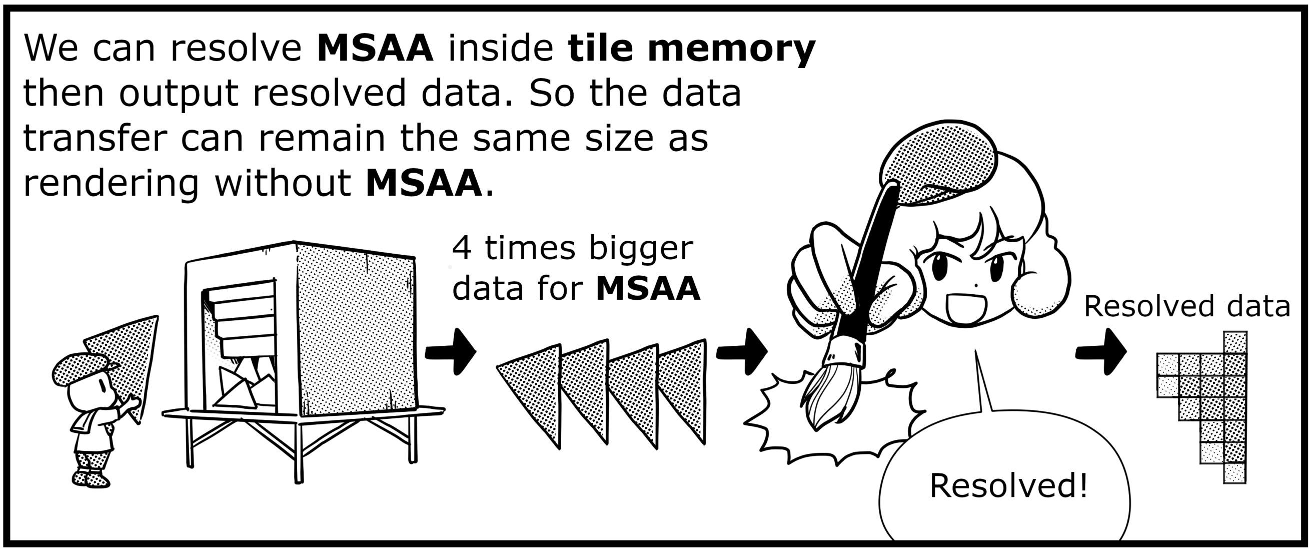

These two tasks are usually performed automatically by the **graphics driver**.
In other words, we can save a lot of data transfer if we skip these two tasks.
Is there any way to skip them?
Sure! For example, if we are going to render a frame which will cover the whole screen without blending the last frame, then we don't need the data from the last frame.
!
I am going to draw a whole new picture!
Fragment
On the other hand, if we don't need the data from the frame we just rendered, then we don't need to move the data back to the video memory.
Drop! Drop! Drop them all!!
What?! I can't imagine when we would render a frame but not need it?!
Haha, it's quite common actually. For example, the **depth buffer** used for rendering can usually be discarded right after rendering.
arm
I got it now! As long as the data is transient, we can avoid moving data back to the video memory.
Correct!

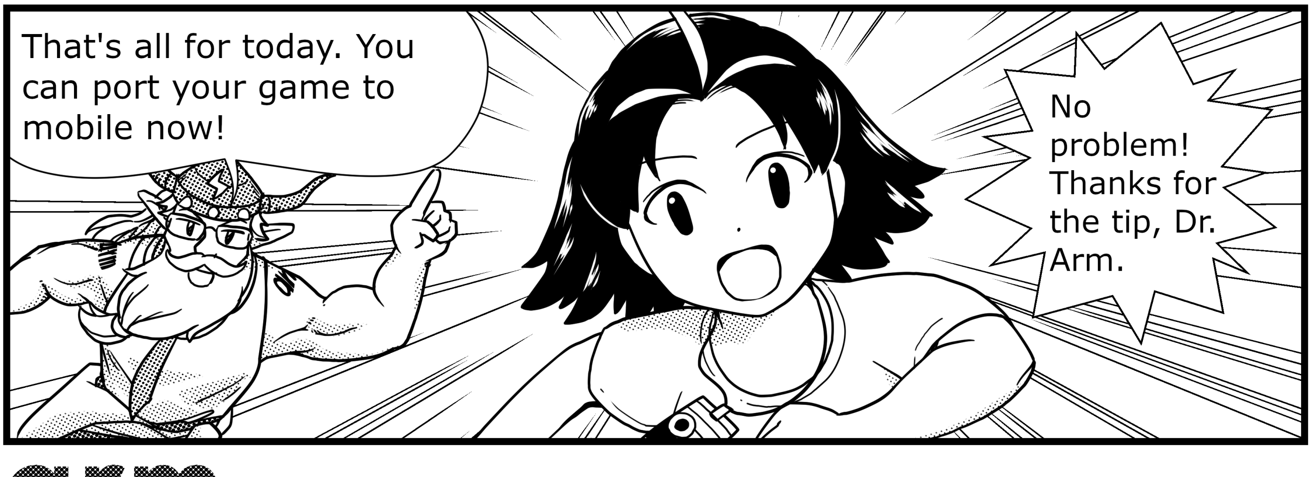

But in practice, different APIs need different ways to make it happen.
For **Vulkan**, it is easier because **render passes** are defined in Vulkan.
Hi, this is Vulkan. I am a young API.
Vulkan
If we don't need the data from the last frame. We can set **LoadOp** to **CLEAR** or **DONT_CARE** at the beginning of the **render pass**.
LoadOp=DONT_CARE
Understood!
Got it!
D
Don't move that data to tile memory.
OK
StoreOp=DONT_CARE
Understood!
Got it!
Don't move that data back to video memory.
If we don't need to preserve the data of the current frame, we can set **StoreOp** to **DONT_CARE** at the end of the **render pass**.
But since OpenGL ES is an old API, there is no **render pass** defined.
LoadOp=DONT_CARE
What the heck?
I don't need the data from the last frame, so...
?
OpenGL ES
English please...
How should I do it?

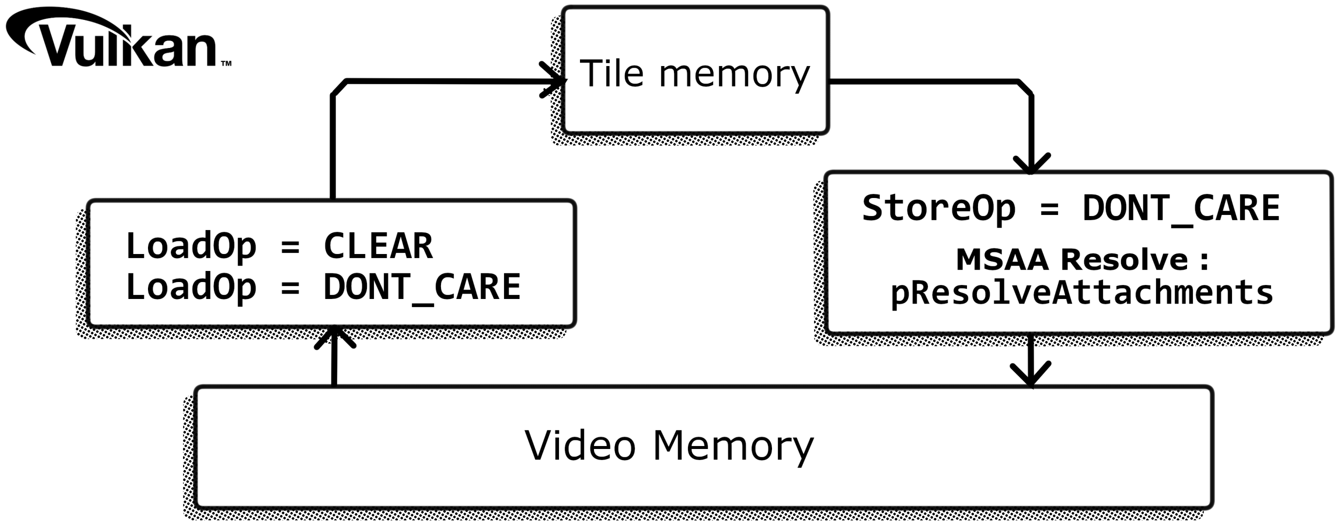

We can call particular APIs to notify the graphics driver.
Let me talk to **Graphics Driver** directly!
glClear()
No problem, I will process it per your request.
If we call **glClear()** or **glInvalidateFramebuffer()** before rendering, then we can notfiy the graphics driver that we don't need the data from the last frame.
OpenGL ES
What should I do when rendering is done then?
You can call **glInvalidateFramebuffer()** to notify the driver to discard the data in the tile memory.
glInvalidateFramebuffer()
Cool! Following what you have taught me, I will now start porting my PC game to mobile.
Wait a second, there is one more useful optimization tip.
Tip?
Modern games usually use **MSAA** to reduce the anti-aliasing, but there is one big problem when using **MSAA**...
Without Antialiasing
With Antialiasing


The problem is that **MSAA** will increase the data transfer. The data transfer will become 4 times bigger when using **4X MSAA**.
GO! GO!
4 times bigger power !


But we can take advantage of **tile memory** again! We can put all **MSAA** data into tile memory.
Tile
Fragment Shade's
Let me handle this.


We can resolve **MSAA** inside **tile memory** then output resolved data. So the data transfer can remain the same size as rendering without **MSAA**.
4 times bigger data for **MSAA**
Resolved data
Resolved!


That's all for today. You can port your game to mobile now!
No problem! Thanks for the tip, Dr. Arm.



Vulkan

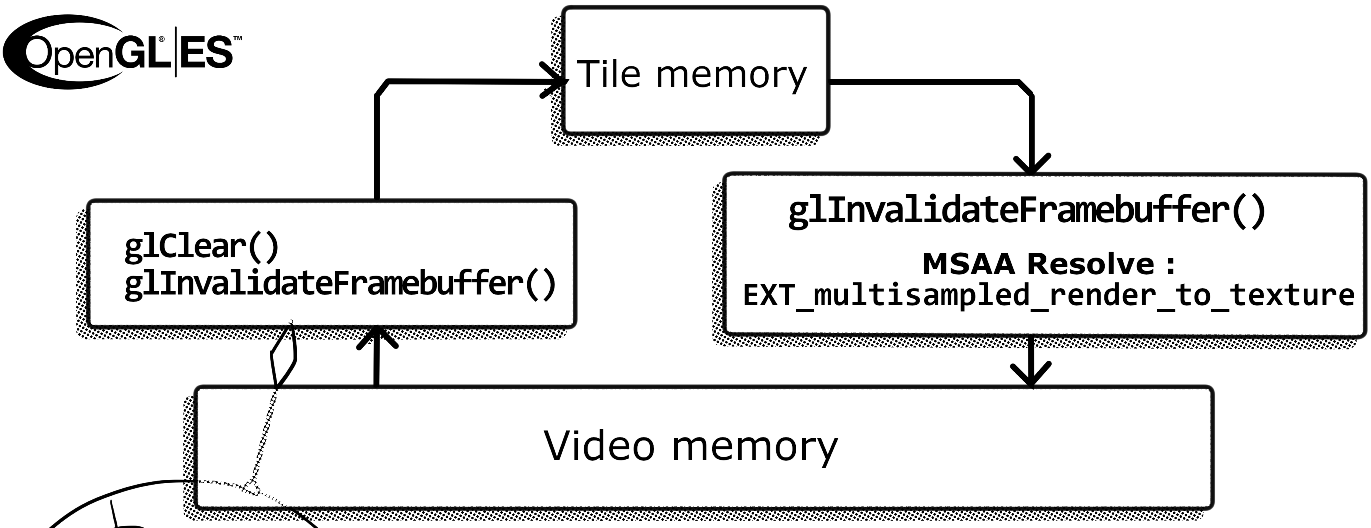
Tile memory

LoadOp = CLEAR
LoadOp = DONT_CARE

StoreOp = DONT_CARE

MSAA Resolve :
pResolveAttachments

Video Memory



OpenGL ES

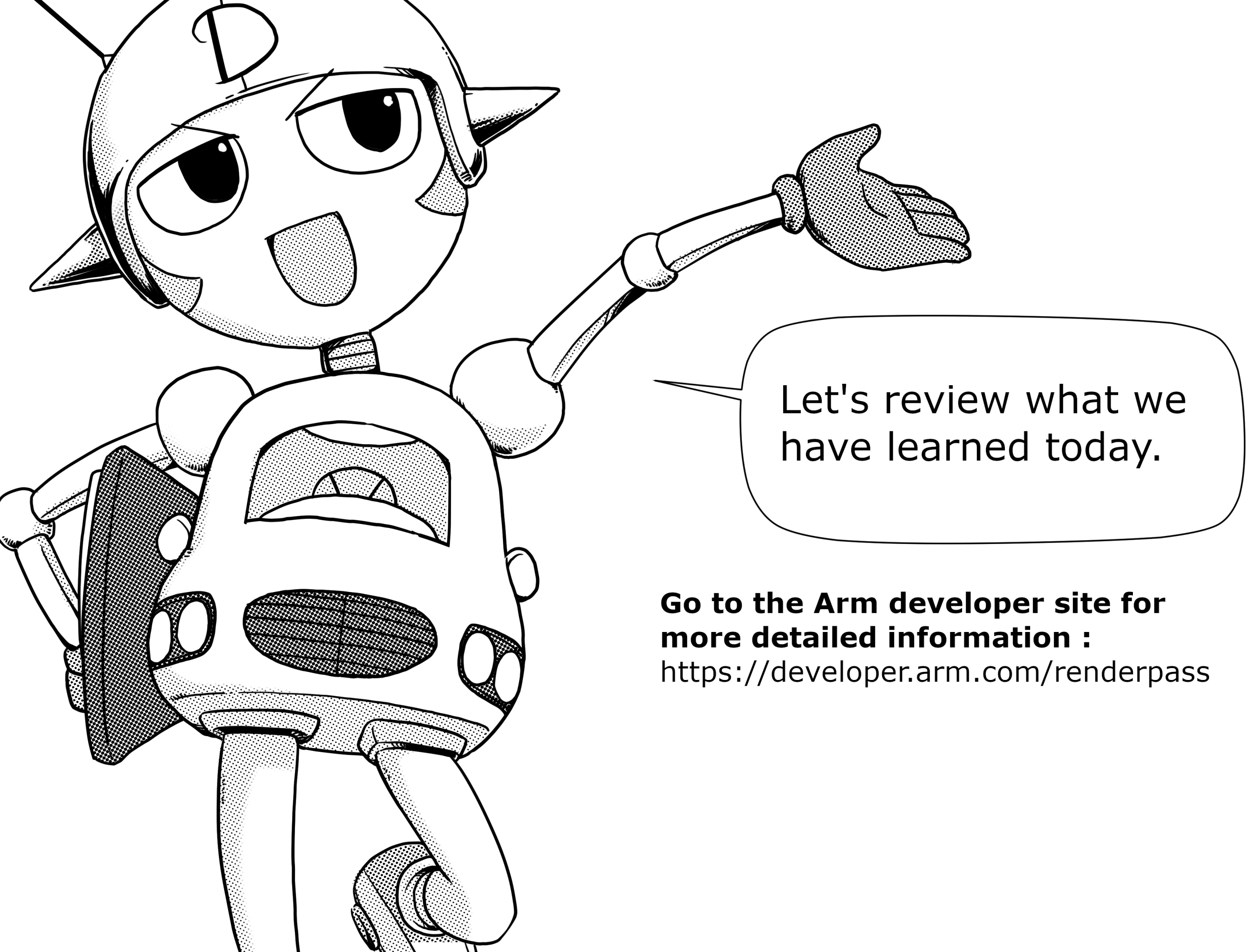
Tile memory

glClear()
glInvalidateFramebuffer()

glInvalidateFramebuffer()

MSAA Resolve :
EXT_multisampled_render_to_texture

Video memory



Go to the Arm developer site for more detailed information :
https://developer.arm.com/renderpass