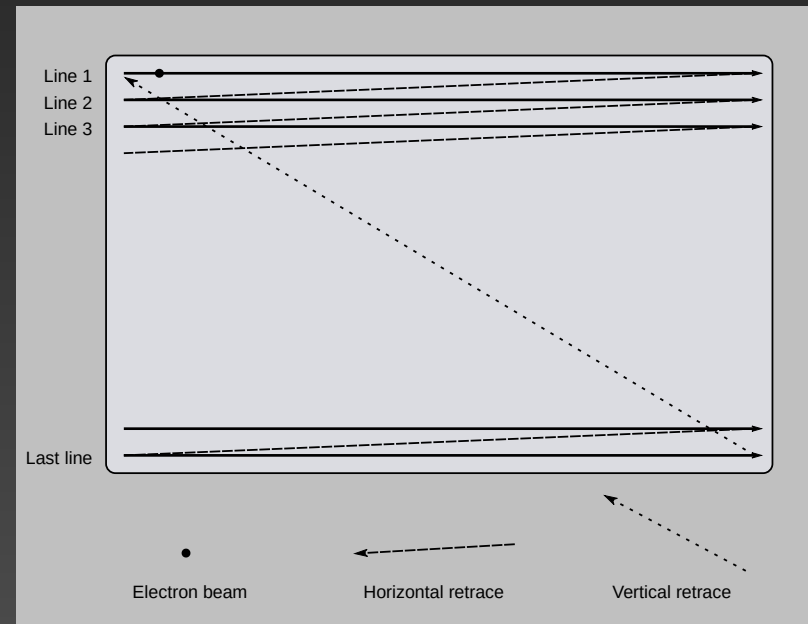
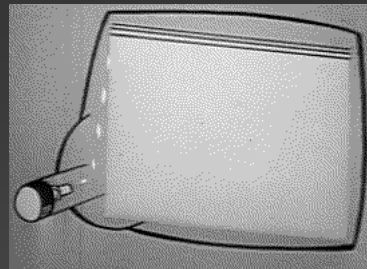
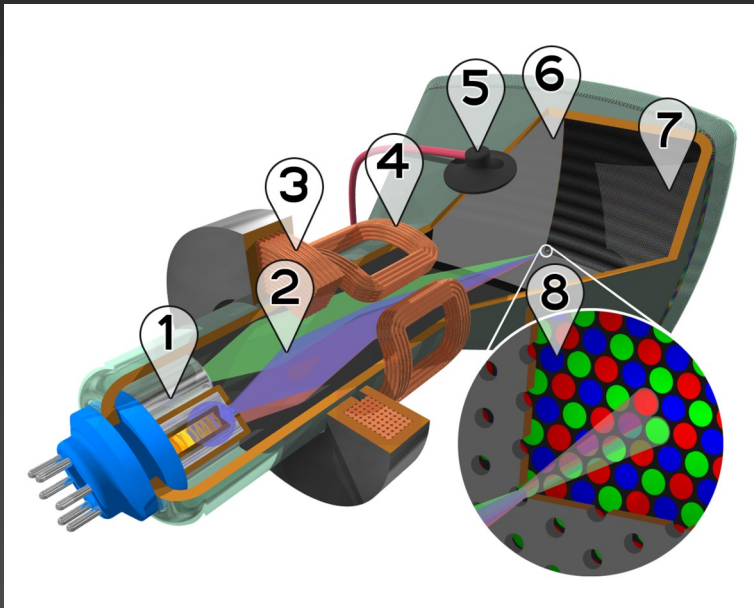


Display synchronization

Vsync,
double-buffering, triple-buffering

Changes in picture on V-sync

- V-sync is (was) signal for vertical retrace of electron beam to upper left corner
 - Retrace is the best time for changing displayed picture
 - without Vsync is old picture in top part and new picture in bottom part of screen – **tearing**
 - faster GPU → more tearing
- Constant RGB data rate = pixel clock



Tearing

Tear Point #1 --->

Tear Point #2 --->



V-Sync in GLFW

- in `init()` or during runtime

```
// Wait for 'n' V-Sync events before swapping color buffers  
glfwSwapInterval(n);
```

```
// Do not wait for V-Sync = Set V-Sync OFF  
glfwSwapInterval(0);
```

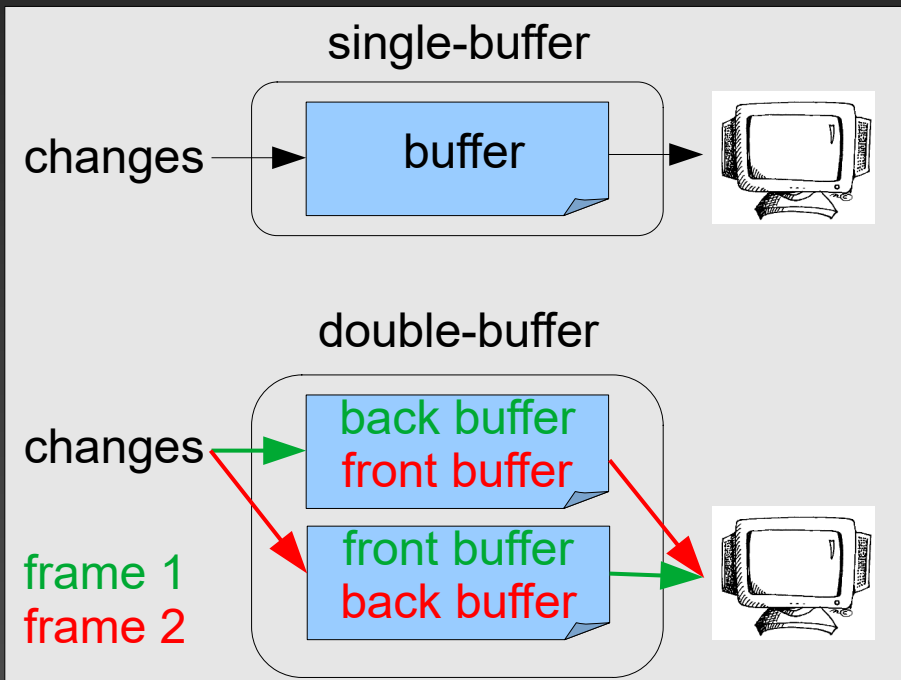
```
// Wait for ONE V-Sync = Set V-Sync ON  
glfwSwapInterval(1);
```

```
// Wait for TWO V-Sync events = show result every other display refresh = 30 FPS limit  
glfwSwapInterval(2);
```

```
// etc.
```

Single vs. Double-buffering

- Drawing frame takes time
 - visible flicking caused by sequention of triangles etc.; we can see build of the scene
- Two color-buffers, front and back
 - drawing changes to back buffer, displaying from front buffer
 - swap buffers (HW accelerated)
 - without V-sync: tearing – horizontal picture tear



Init: GLFW creates double-buffer by default

Frame finished:

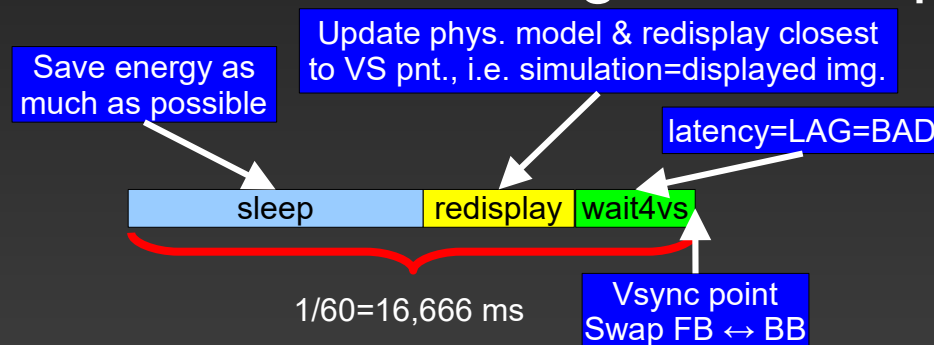
`glfwSwapBuffers(window)`

(contains `glFlush();`)

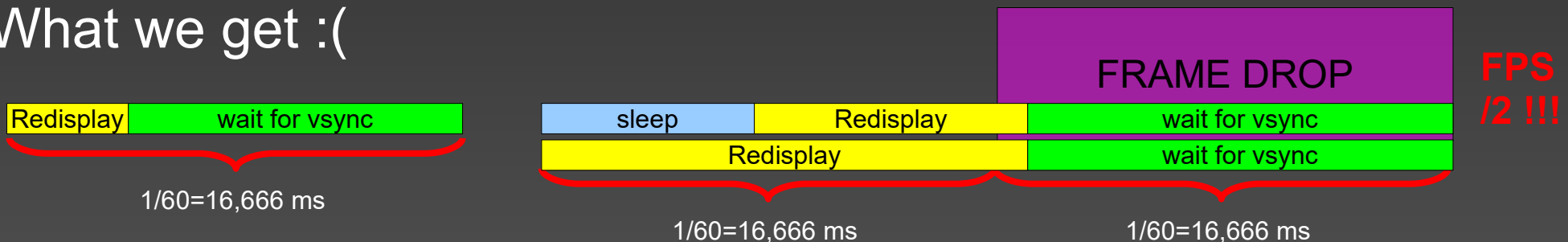
Frame drop

- min. 24 (16) fps necessary, double-buffering, vsync
- Infinite loop: update simulation → redraw
- redraw usually as fast as possible (no explicit wait)
 - 100% system load
 - can you use Sleep(milisec) etc.? How many ms?
 - using Sleep() blocks thread, CPU(GPU) time wasted, if we wait too long, FPS drop :-(

What we want:



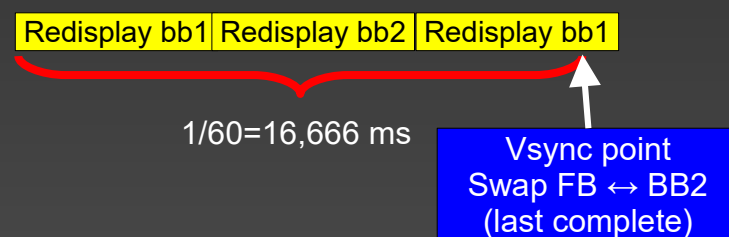
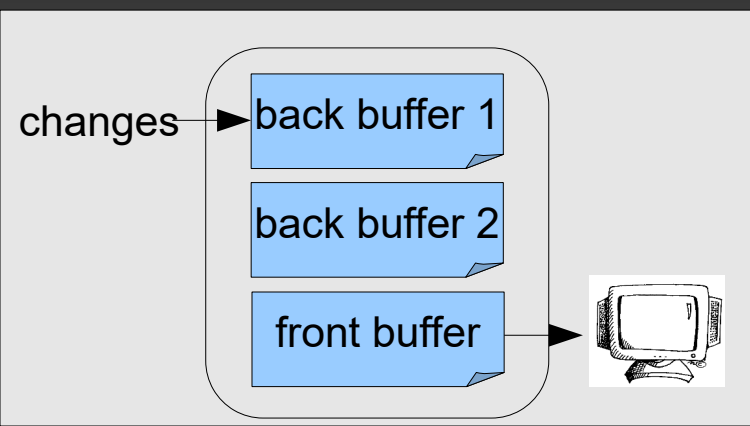
What we get :(



Double vs. Triple-buffering

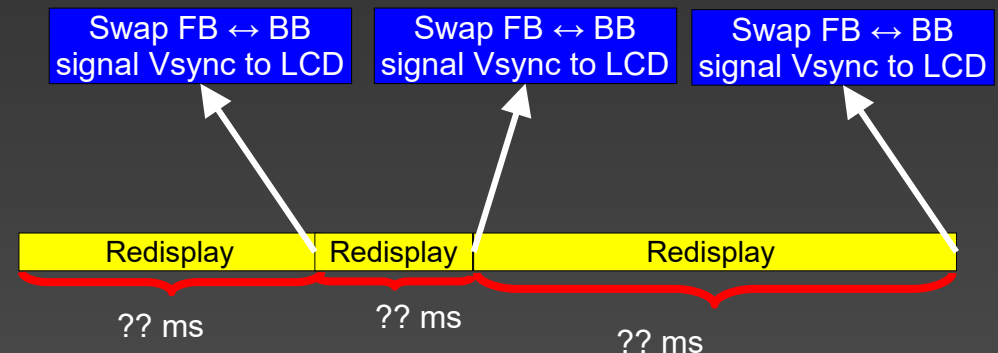
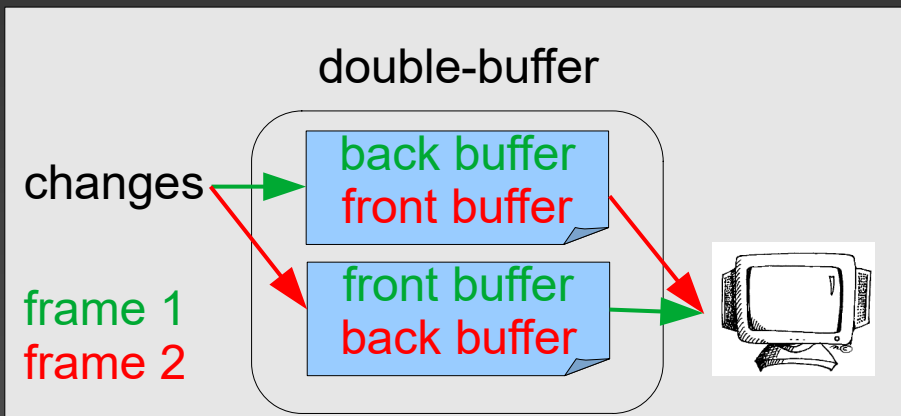
- Double-buffer disadvantage
 - significant FPS drop
 - $1/1, 1/2, 1/3, 1/4, \dots \rightarrow 60, 30, 20, 15, \dots$
 - higher latencies
- Triple buffering in GL – add third buffer
 - draw new frame to the second back buffer immediately after previous is finished
 - + increase speed, lowers latency
 - increase GPU+MEM load

(D3D: serial buffers – adds latency even more)
- Vulkan = mailbox; nvidia = FastSync



The best solution

- Variable refresh rate (+ high FPS monitor, 120, 144, 160, ...)
 - Redisplay image + swap buffers when ready
 - LCD waits for vsync... it does NOT start new refresh itself
 - Send vsync signal to LCD
 - Send data as fast as possible (max supported LCD speed)
 - No fixed FPS, variable within technological limits, e.g. 40...160Hz
 - Use only one back buffer (like double-buffering)
 - No waiting = draw as fast as you can
 - If $\text{actual_fps} < \text{max_lcd_fps}$
 - vsync=on → no tearing
 - Else: vsync = off → tearing, but max possible fps
- AMD FreeSync = VESA Adaptive Sync (Displayport) = HDMI 2.1 VRR, Nvidia G-Sync, ...



Comparison

