S-Core

Kicking GL out of the main T in Evas

2016.06.16

Sung W. Park
sungwoo@gmail.com

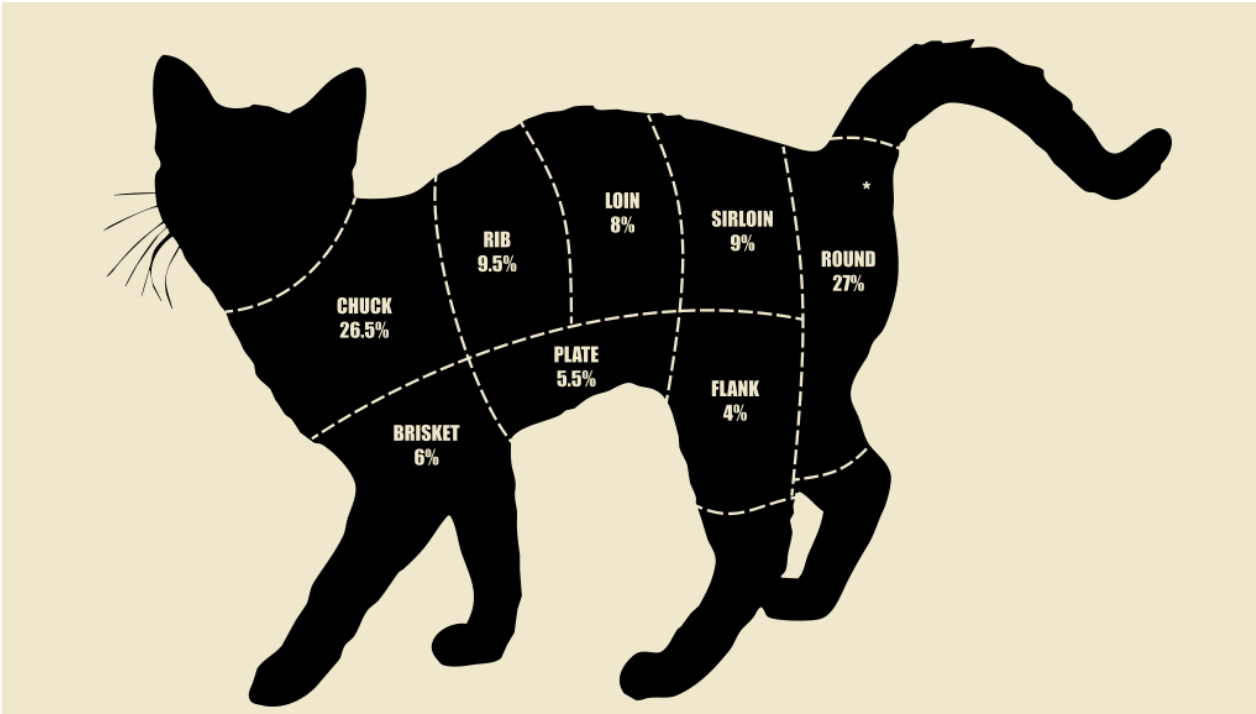


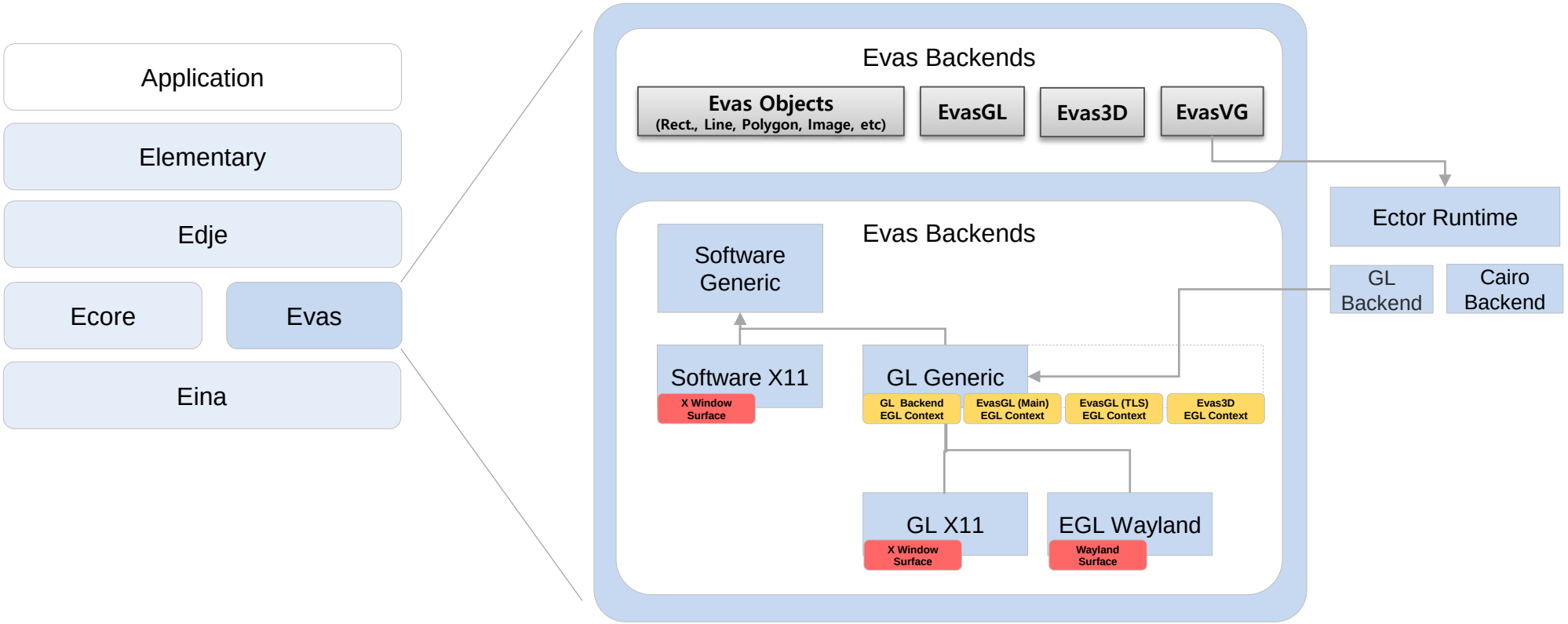
*Image source: https://wrongtees.com/show/P90/Cat-Meat

Index

EFL & Evas Architecture

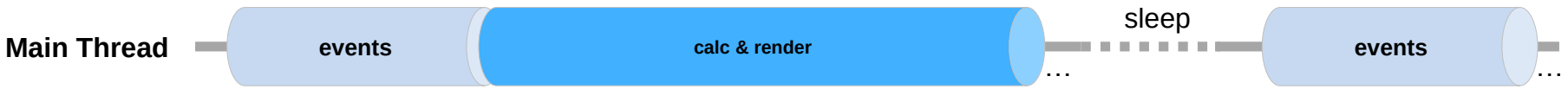
S-Core

- **Evas started as a nice 2D scene-graph library**
- **Over the years new features spaghettified things**
  - EvasGL, Evas 3D, EvasVG



Evas Rendering Flow

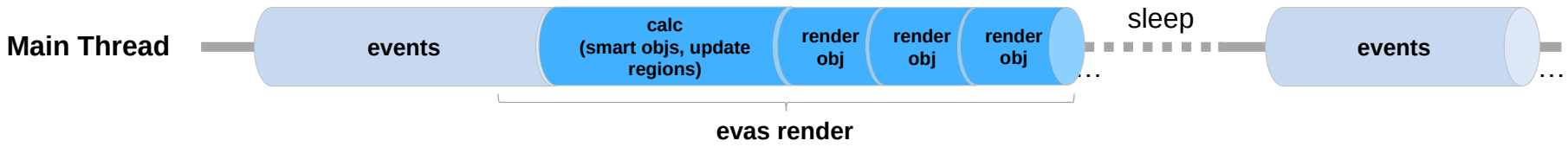
- **General Rendering Flow in Evas**



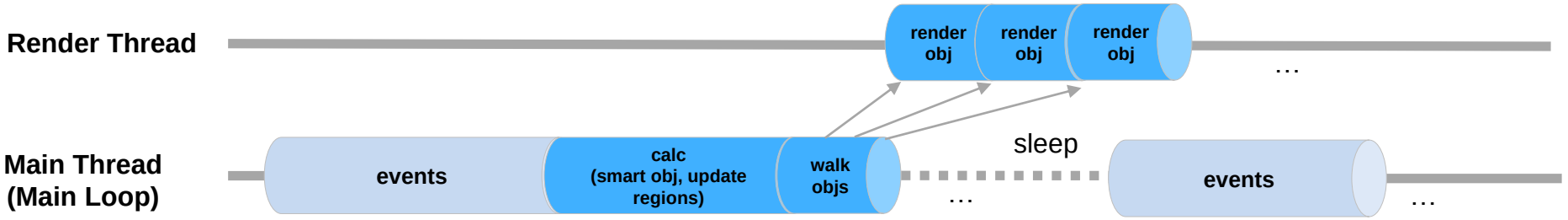
- **Heavy rendering could cause the main loop to perform poorly!**
  - This has become an issue with addition of new features all running in the main thread

Evas Rendering Optimization – Async Rendering for SW backend

- **Default rendering mode**



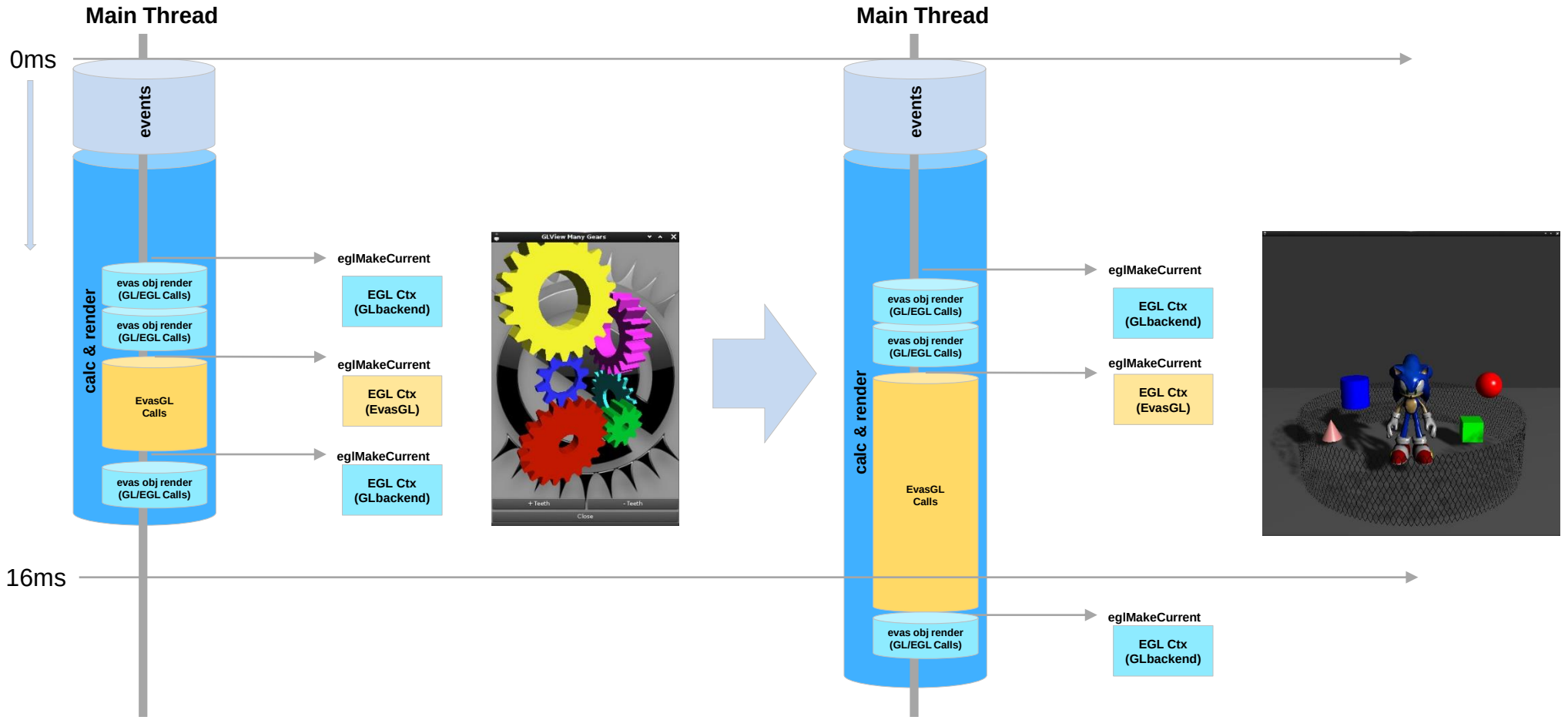
- **Async rendering mode**



- **Does it really help? (According to Raster)**
  - 90% of the time is spent setting up objects for rendering & 10% of the time is spent on rendering
  - "Bigger optimization is in the object setting/walking part and CEDRIC was SUPPOSED to work on it but NO, he had to work on stinkin Vector Graphics….!@$#@!$" -Raster
  - **But it still does help**!
- **What about GL backend? There's no such optimization currently. Does it make sense?**

Pain Points for Sammy

- **Performance / Interactivity Issues**
  - Heavy GL usage causes user interactivity to suffer: Evas Objects, **EvasGL**
  - Frequent context switches between Evas, EvasGL, Evas3D, EvasVG can degrade performance



Evas Rendering Optimization for GL Backend?

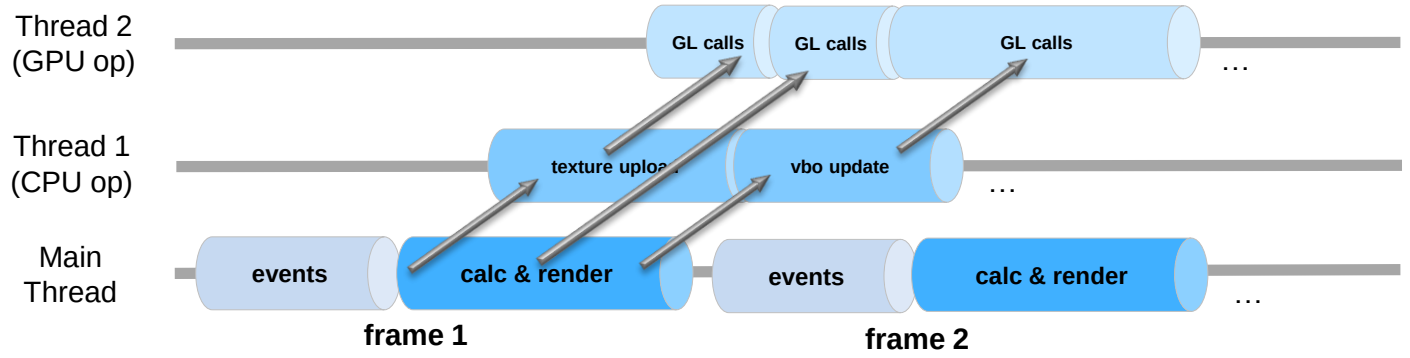
S-Core

- **Sammy's Request**
  - We want the entire rendering on a separate thread!
    - With the current rendering structure, it would be very difficult

- **Compromise ➔ Pull out all the GL calls to a separate thread**
  - It would be async rendering for GL backend
    - They felt that this would still benefit their use cases
  - **This would be a necessary step for Render 2 refactoring work in the future**

Evas Rendering Optimization for GL Backend?

- **One potential scenario…**



  - 3 Threads? Or more?
    - **Main thread** calculates draw regions and dispatches draw calls
    - **Thread 1** performs CPU operations related to GL calls
    - **Thread 2** performs all GL calls

- **Is this the best scenario?**
  - How many threads should we use? What's the best way to split jobs between threads?

Evas Async GL Rendering: Two Main Issues to Tackle

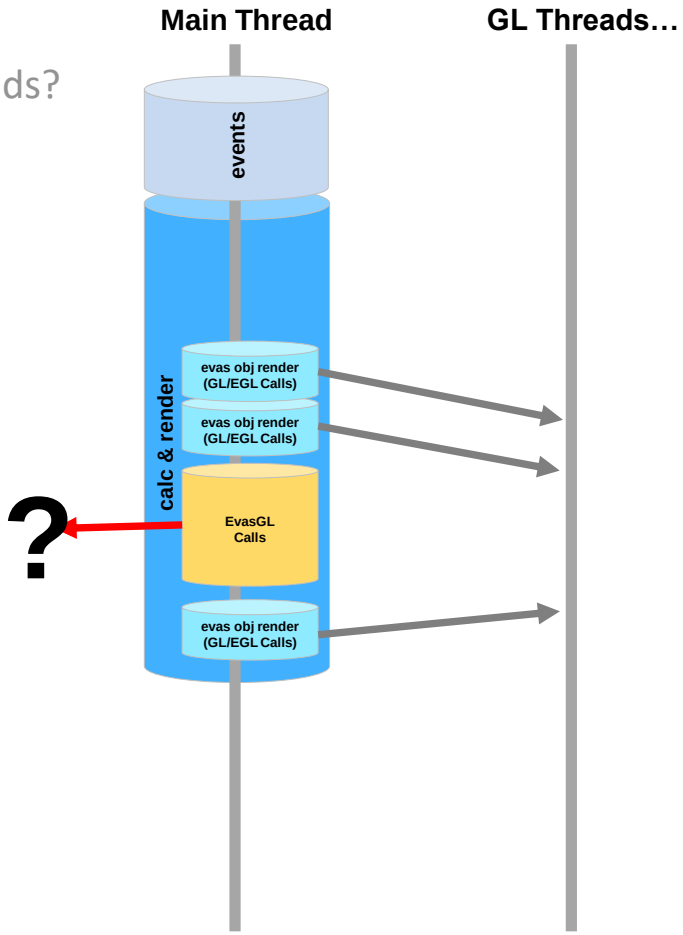
S-Core

1. Strategies to pull out GL calls for evas object rendering?

- How do we efficiently offload CPU/GPU bound calls to separate threads?
- How do we serialize memory bound calls efficiently?

2. Strategies for EvasGL (Evas3D)?

- How do we call EvasGL efficiently?
- What about direct rendering vs indirect rendering?



Strategies to pull out GL calls for evas object rendering

S-Core

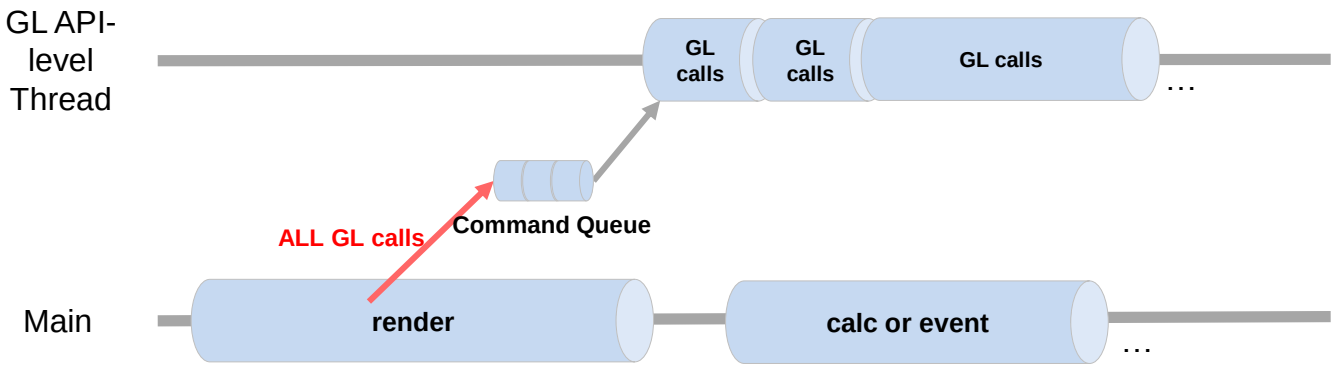
- **To find the best approach would require profiling as well as testing different approaches**

- **To get there we adopt two stages**
  - Stage 1 ➔ API-level threading
  - Stage 2 ➔ Functional-level threading

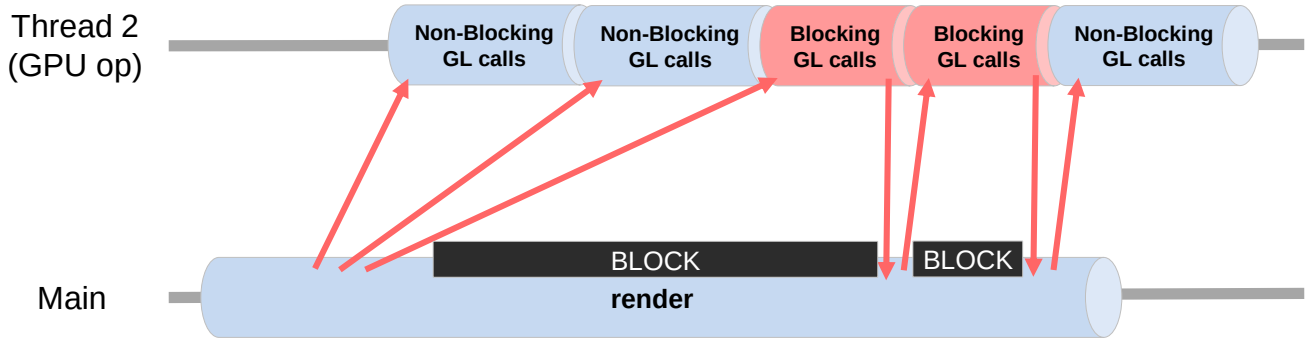
Stage 1: GL API-level threading

S-Core

- **Create an infrastructure to be able record GL calls and call them from a different thread**



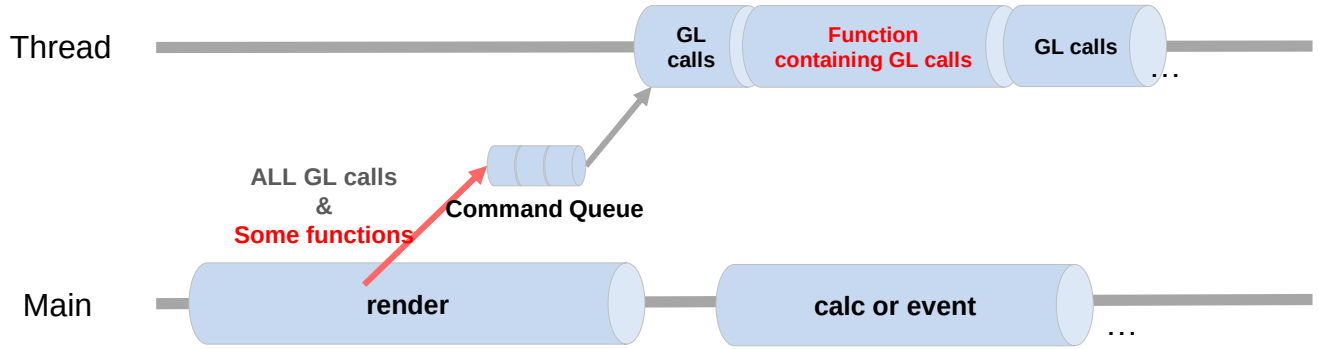
- **Some calls require blocking by the main thread:**
  - Calls that have return values or memory access: ie. glGet, Texture/Buffer uploads
- **Other calls may return right away**



Stage 2: Functional-level threading

S-Core

- **Once we have an infrastructure to call GL calls we can start testing and optimizing them**
- **Strategy**
  - Benchmark ➔ Group/ Reorder etc. ➔ Apply
  - Decide # of threads and etc as we go along



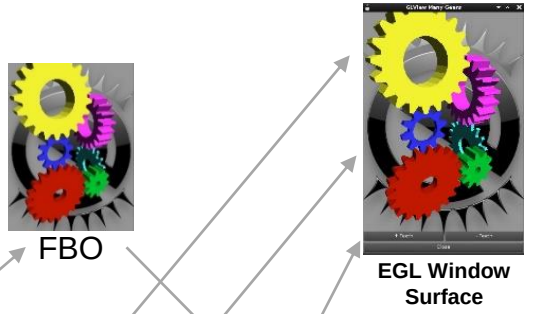
2. Strategies for EvasGL (Evas3D)?

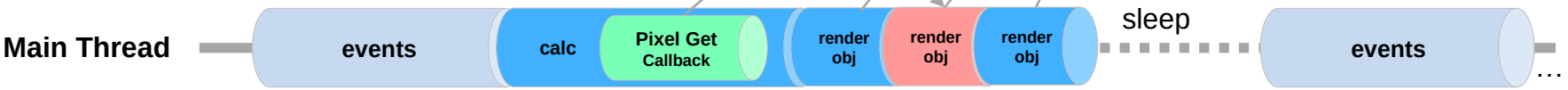
S-Core

- **We need to handle two rendering modes separately**

Indirect Rendering

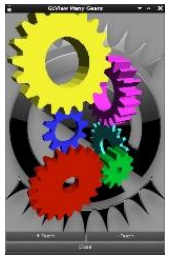
- Normal EvasGL mode of operation
- Requires extra memory space & buffer copy
- Unchanging scene benefits from rendering once to a buffer





Direct Rendering

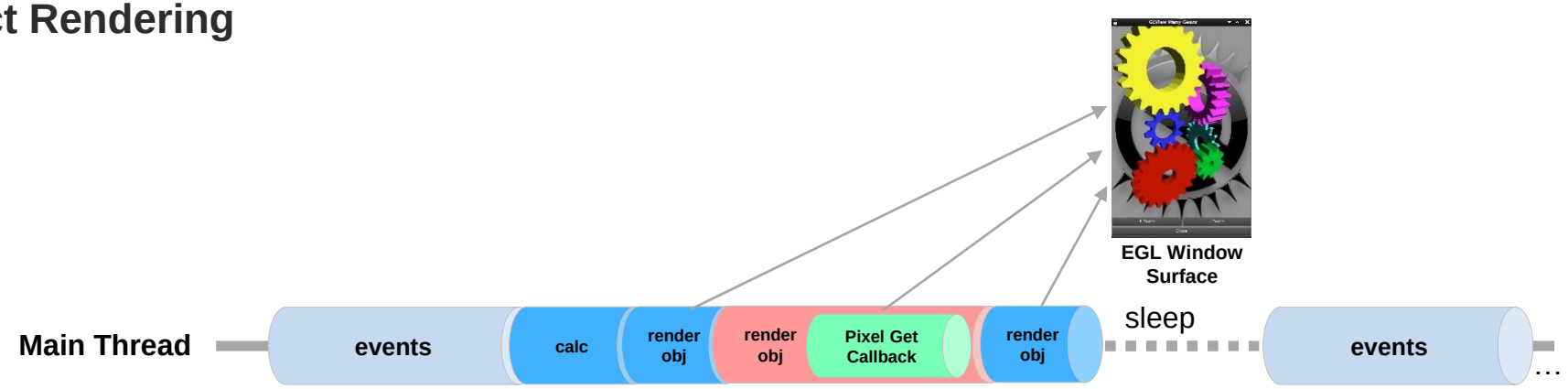
- EvasGL optimization ➔ **Must be called from Pixel Get Callback**
- Saves on memory and buffer copy and thus faster
- Even if the scene doesn't change it needs to re-render every time
- This optimization is used often in Tizen



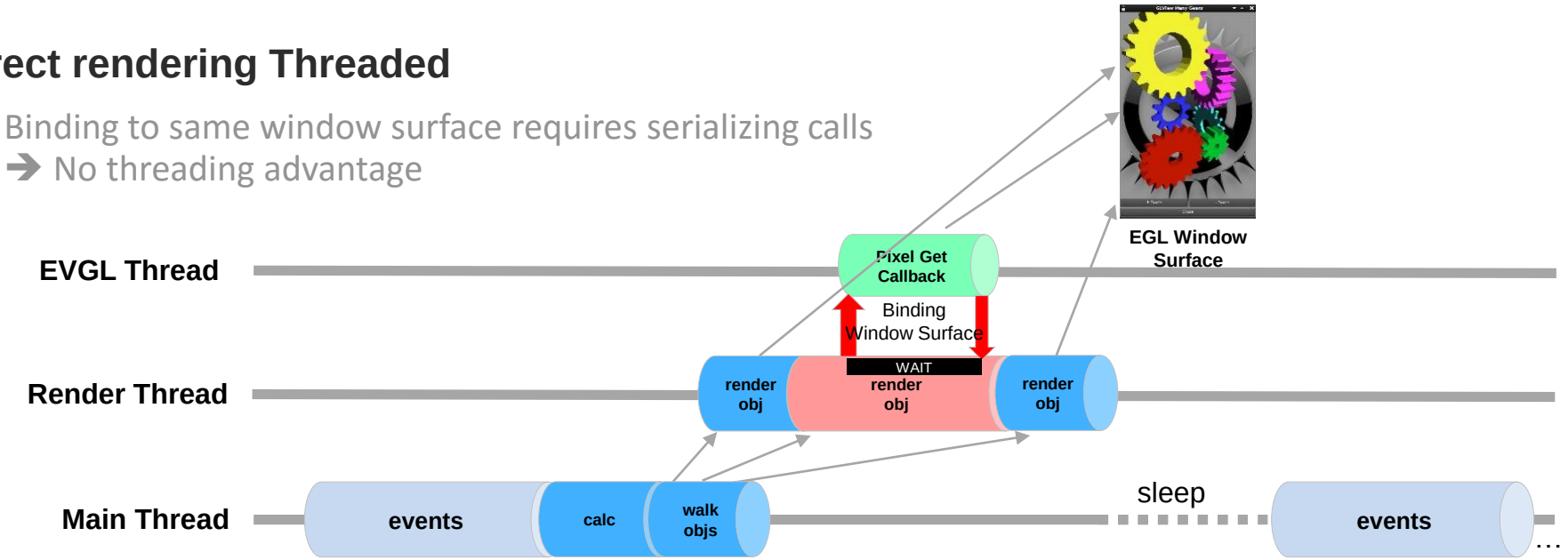
Direct Rendering Threading Strategy

S-Core

- **Direct Rendering**



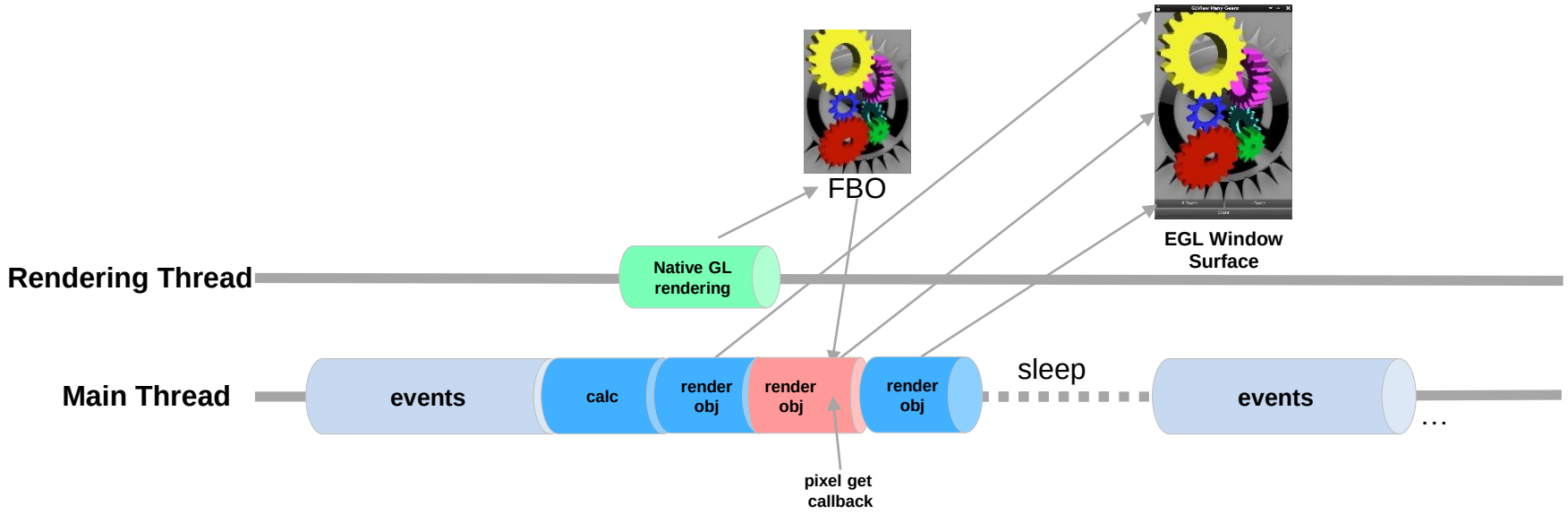
- **Direct rendering Threaded**
  - Binding to same window surface requires serializing calls
    ➔ No threading advantage



One Partner's Case

S-Core

- **Separate GL thread + FBO + Direct Rendering**
  - Do everything on a separate thread
  - Use EvasGL Direct rendering for compositing

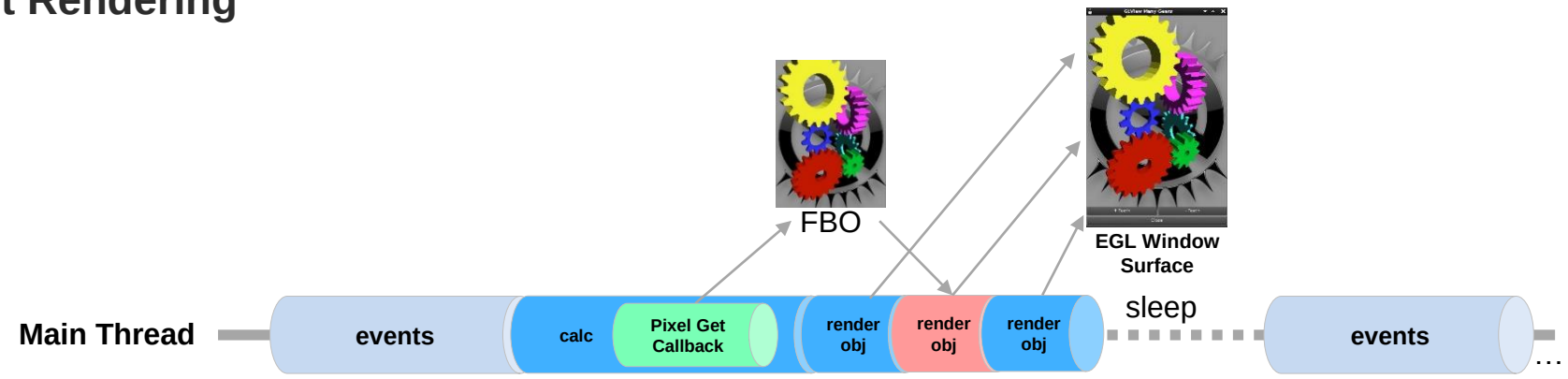


- **They saw great performance gain from such approach**
  - This is something that we can employ with our indirect rendering approach

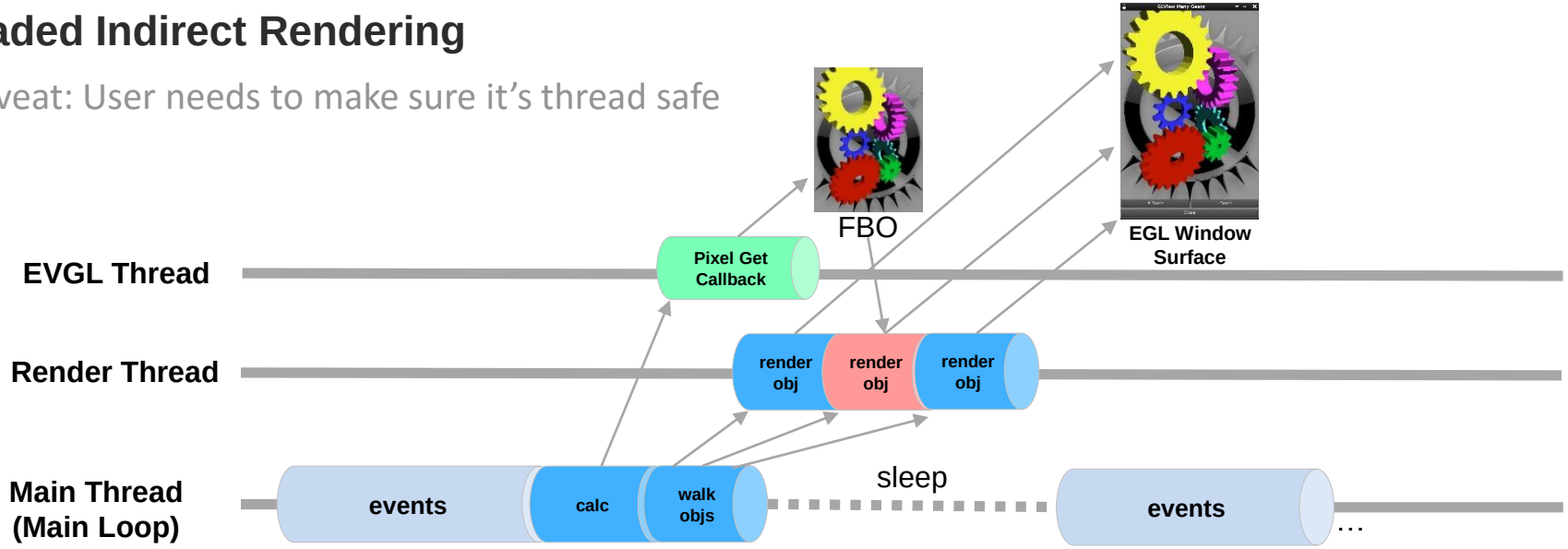
Indirect Rendering Threading Strategy

S-Core

- **Indirect Rendering**



- **Threaded Indirect Rendering**
  - Caveat: User needs to make sure it's thread safe



Current Status

S-Core

- **Currently testing GL API-Level threading implementation**

- **To Do's…**
  1. Functional-level threading (Additional optimization with threading)
     - Use the infrastructure from stage 1 to do additional grouping optimization
  2. Deal with additional EvasGL strategies

Discussion?

- **Async Evas GL rendering?**
- **Future for Evas refactoring? Render 2 ???**
- **Evas Vulkan backend?**

S-Core

» *Value Creator through Advanced Technology*