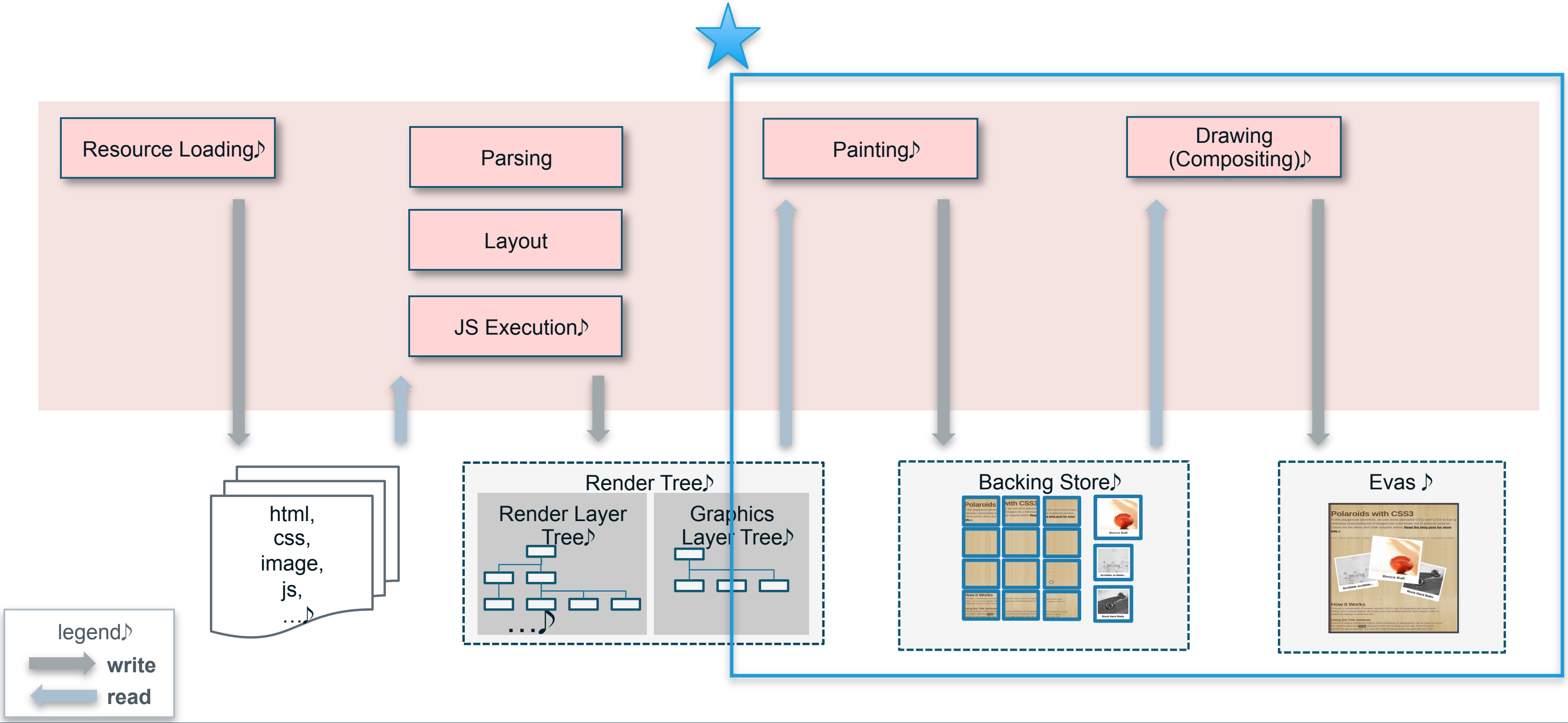
Performance Optimization of Tizen WebKit : Memory and Graphics

Seojin Kim
Samsung Electronics

TIZEN™
DEVELOPER CONFERENCE 2013
SAN FRANCISCO

Contents

Introduction

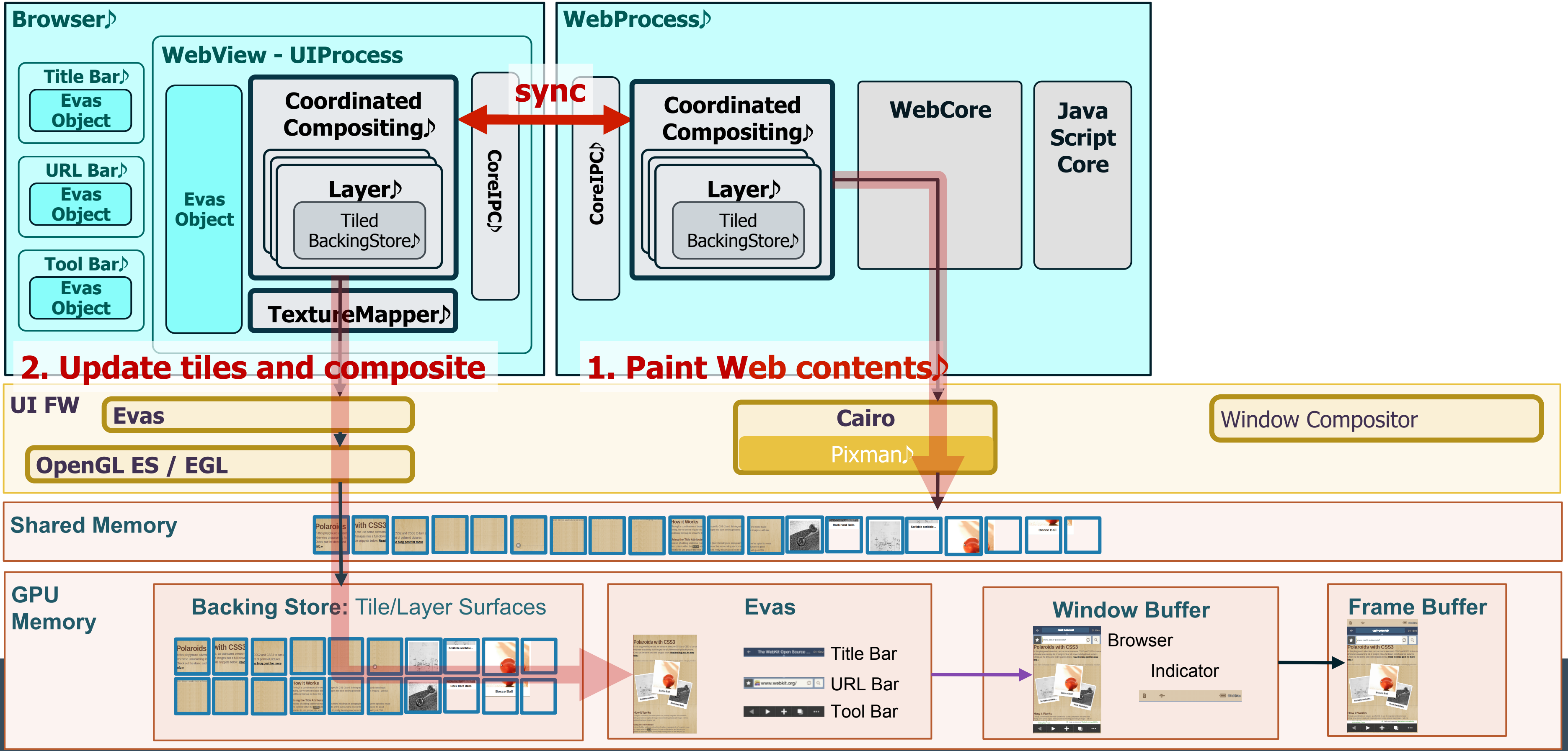


Overall Rendering Architecture

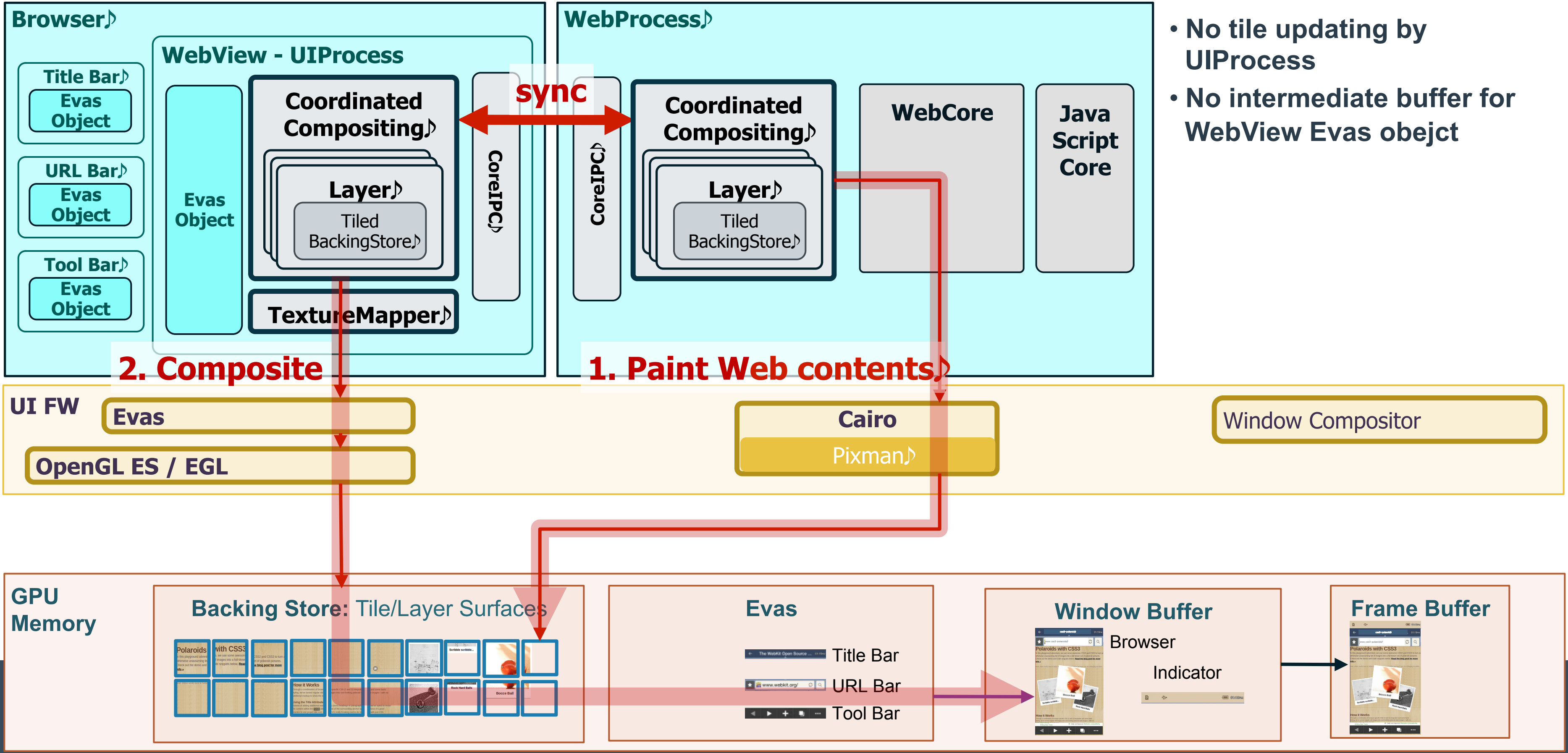
- **WebKit2-EFL**
  - UIProcess and WebProcess
- **Coordinated graphics architecture**
  - WebProcess painting to tile back buffer
  - UIProcess compositing tiles into display
- **Benefits**
  - User responsiveness: nothing blocks UIProcess from processing user event

Performance

Basic Rendering Flow



Optimized Rendering Flow



- No tile updating by UIProcess
- No intermediate buffer for WebView Evas obejct

Optimization Render Flow (1/2)

- **SharedPlatformSurfaceTizen**
  - Tile/GraphicsSurface back buffer
  - Locate in GPU memory
  - Both UIProcess and WebProcess access GPU memory directly
  - DDK support is necessary: TBM (tizen buffer manager) and EGL extension

Power consumption and drawing speed

- Zero-copy realized: no texture uploading in UIProcess

Memory consumption

- Restriction: allocation time is long → Introduce pool for recycling

Optimization Render Flow (2/2)

- **TextureMapper with Evas GL direct rendering**
  - No separate Evas object surface for WebView
  - Compositing destination is window surface itself

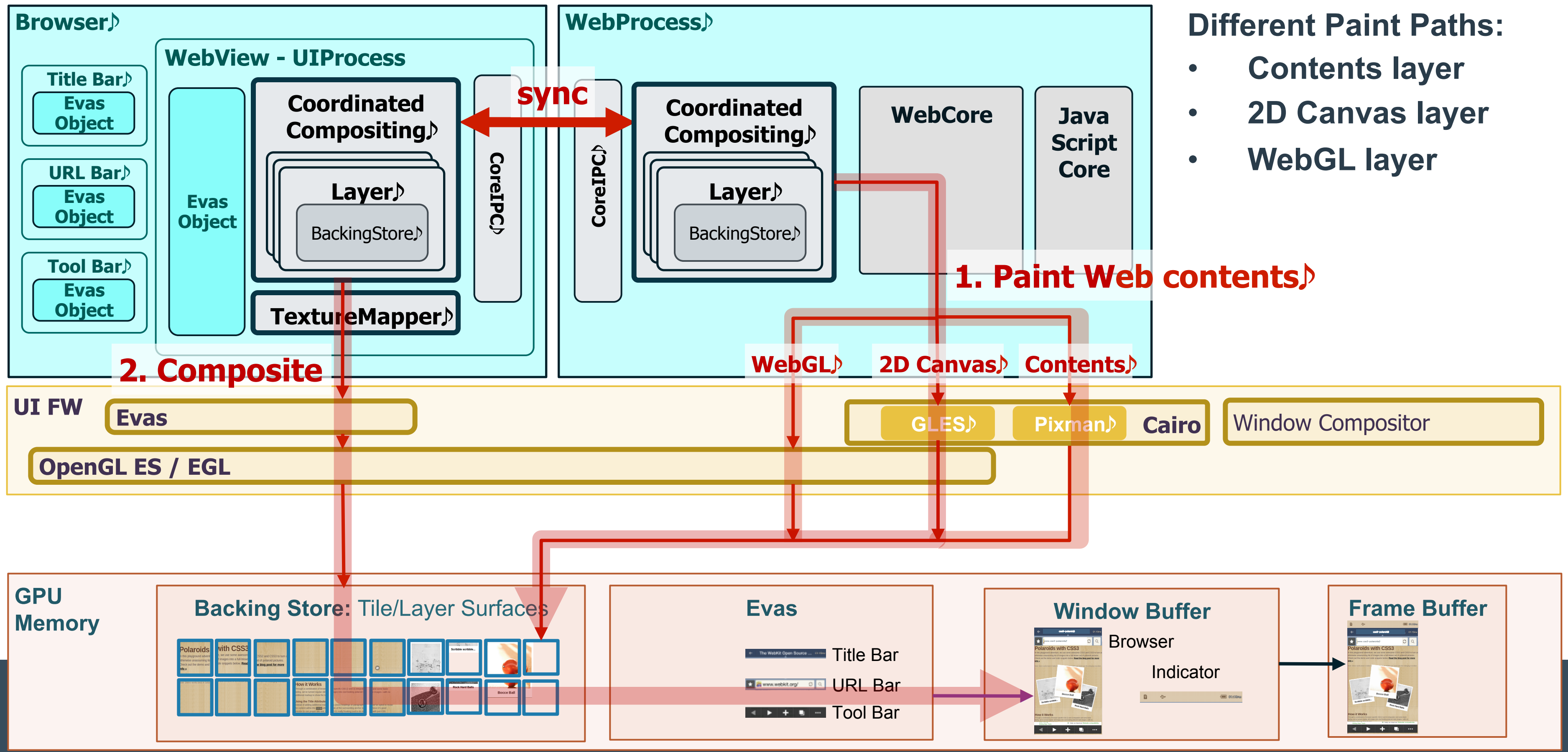
Power consumption & Drawing speed

- Reduce compositing count

Memory consumption

- No separate memory space for WebView Evas object

Support Other Graphics Feature

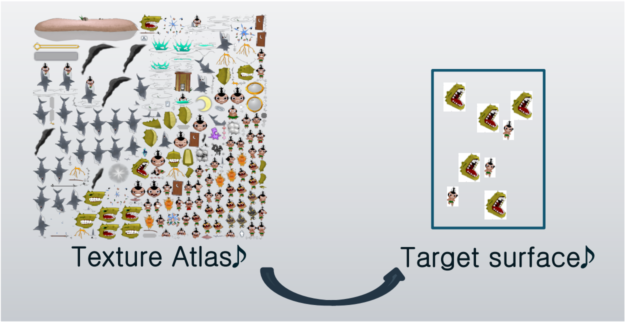


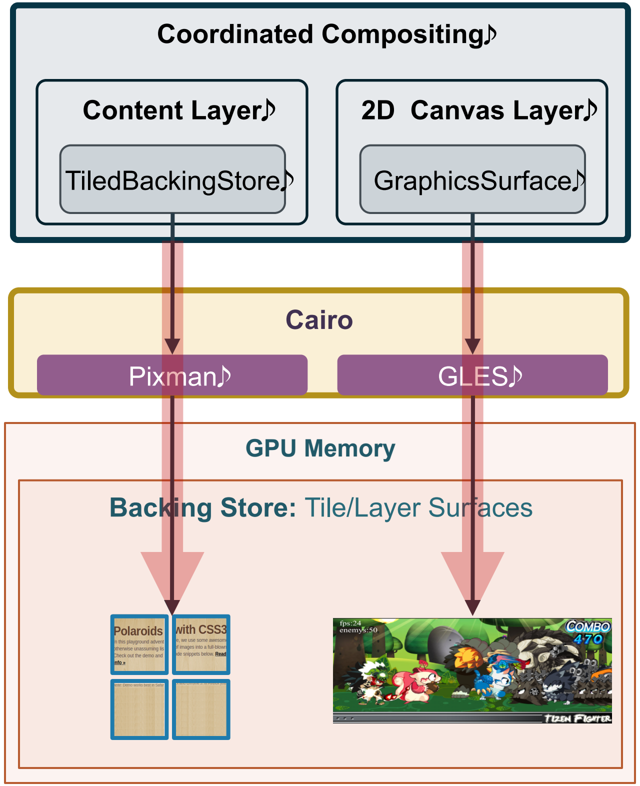
Different Paint Paths:

- Contents layer
- 2D Canvas layer
- WebGL layer

Accelerated 2D Canvas

- **Paint with Cairo-GLES**
- **Optimize for 2D graphics game**
  - Texture Atlas

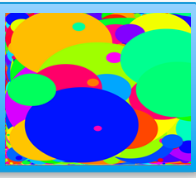




CanvasPerf Score

2.38

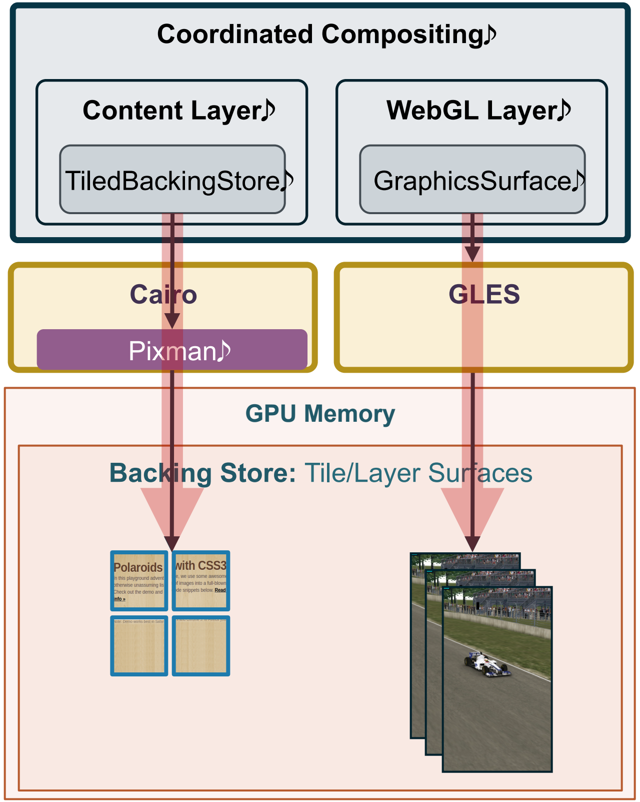
@ Tizen Reference Target + Tizen 2.1

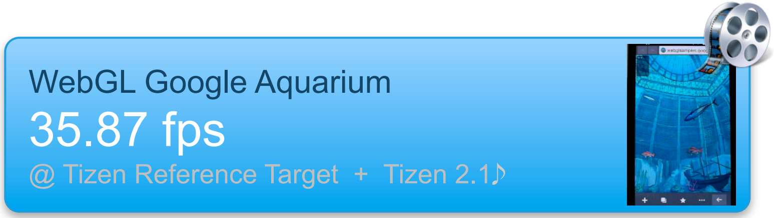


WebGL

- **Directly use GLES**
- **Triple buffering for WebGL layer surface**
  - Pipelining of 3 stages:
  - Painting by WebProcess
  - Issuing commands for compositing by UIProcess
  - Actual compositing by GPU

→ Fully utilize GPU





Memory

Memory Optimization

Objectives

Ensure that foreground application runs smoothly

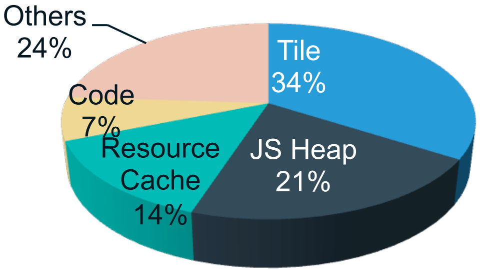
- **Foreground: WebView is visible**
  - Active state
  - Ensure certain performance quality with optimal memory usage

- **Background: WebView is invisible**
  - Usually in suspended state
  - Yield as much memory as possible

Foreground Web App Memory Optimization

Contents Characteristics♪	Major Memory Consumer
Dynamic user experience (such as CSS animation, contents layering)♪	Tile, JS Heap♪
Image resource-heavy♪	Resource Cache♪
Large content size♪	Tile♪

- **Memory Optimization Parameters:**
  - Tile cover area size
  - Tile size
  - JS engine garbage collection period
  - Resource cache size

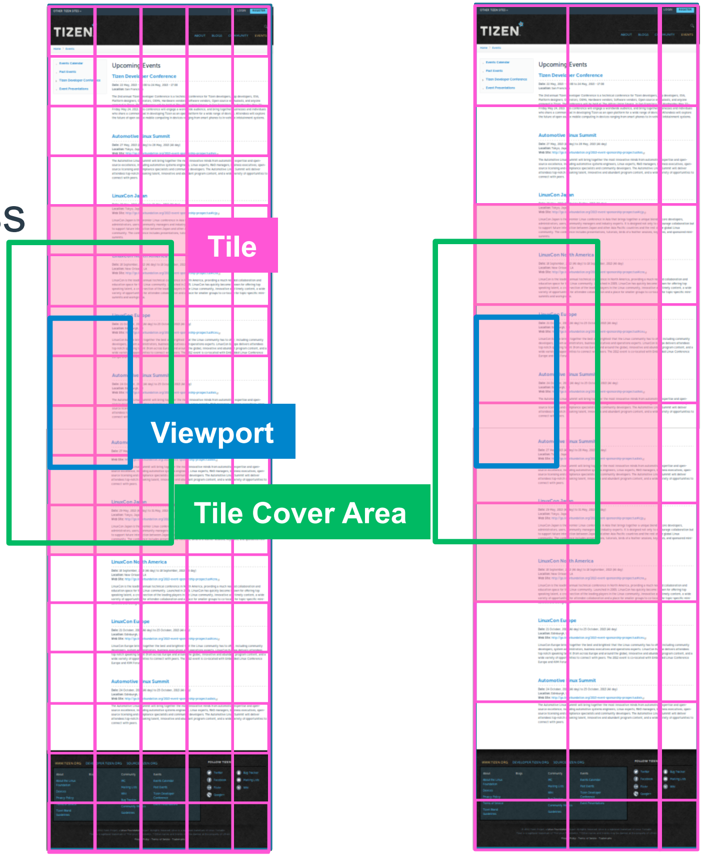


Memory Usage Breakdown
for a web content with very complex UX

Performance against Memory Trade-off

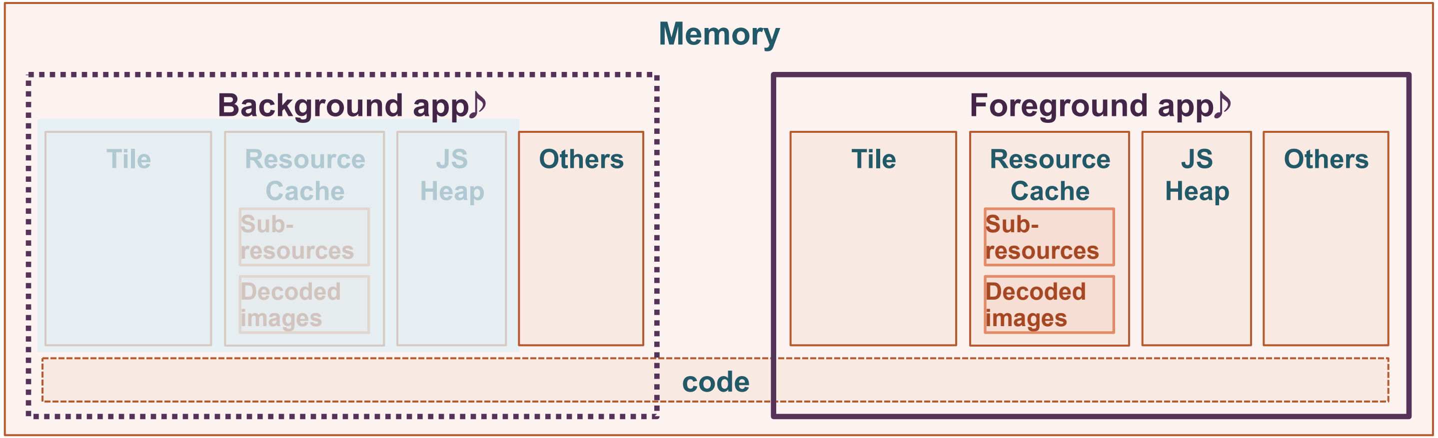
- Tile cover area size
  - Larger means better scrolling responsiveness
  - Smaller means better memory usage

- Tile size
  - Larger means better painting speed
  - Smaller means better memory usage



Background Web App Memory Optimization

- **Release memory**
  - Suspend tile painting and purge backing store
  - Utilize disk cache for JS heap and resource cache (to-be)



※ code memory is shared among all web app instances

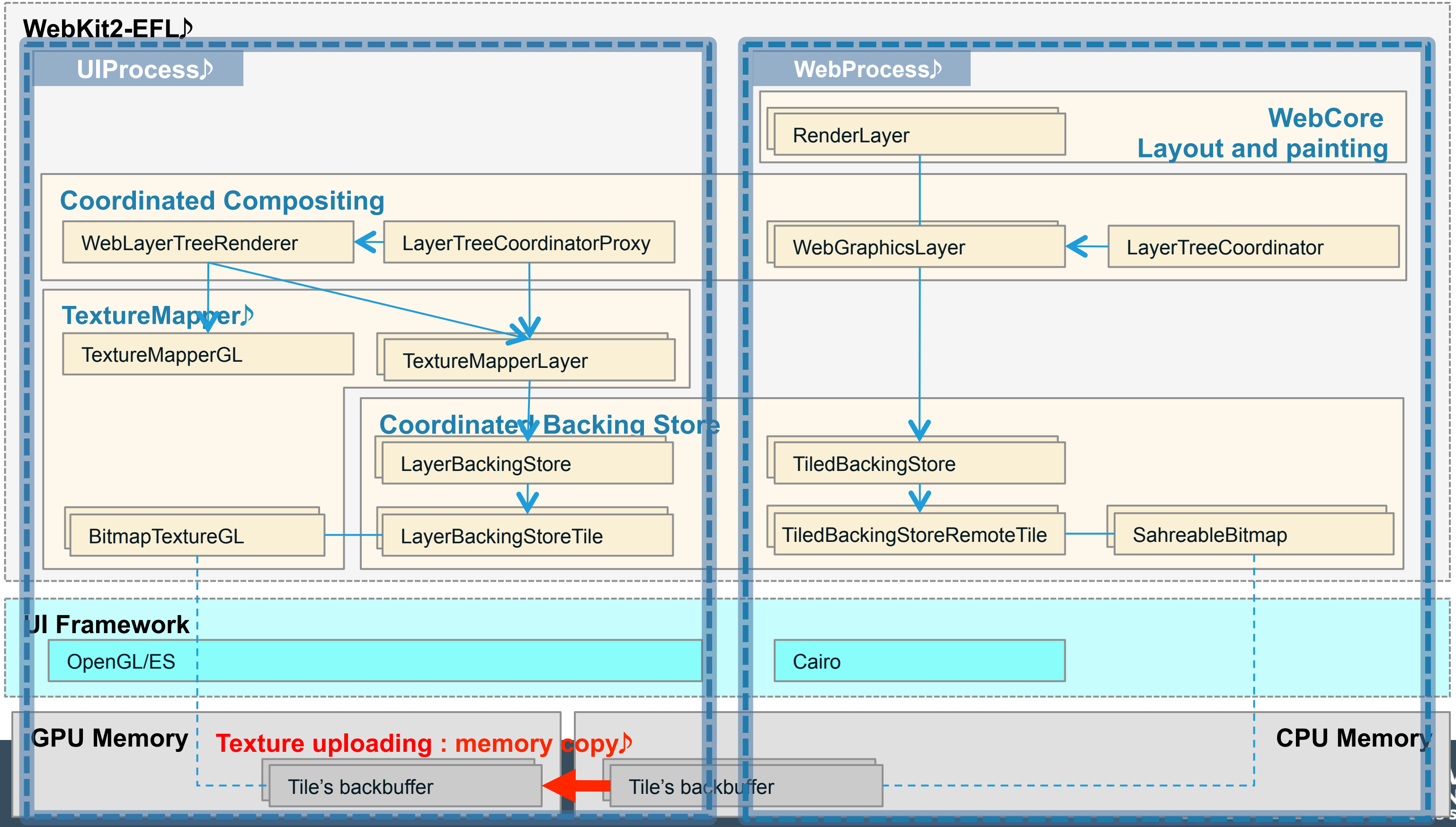
Conclusions

Conclusions

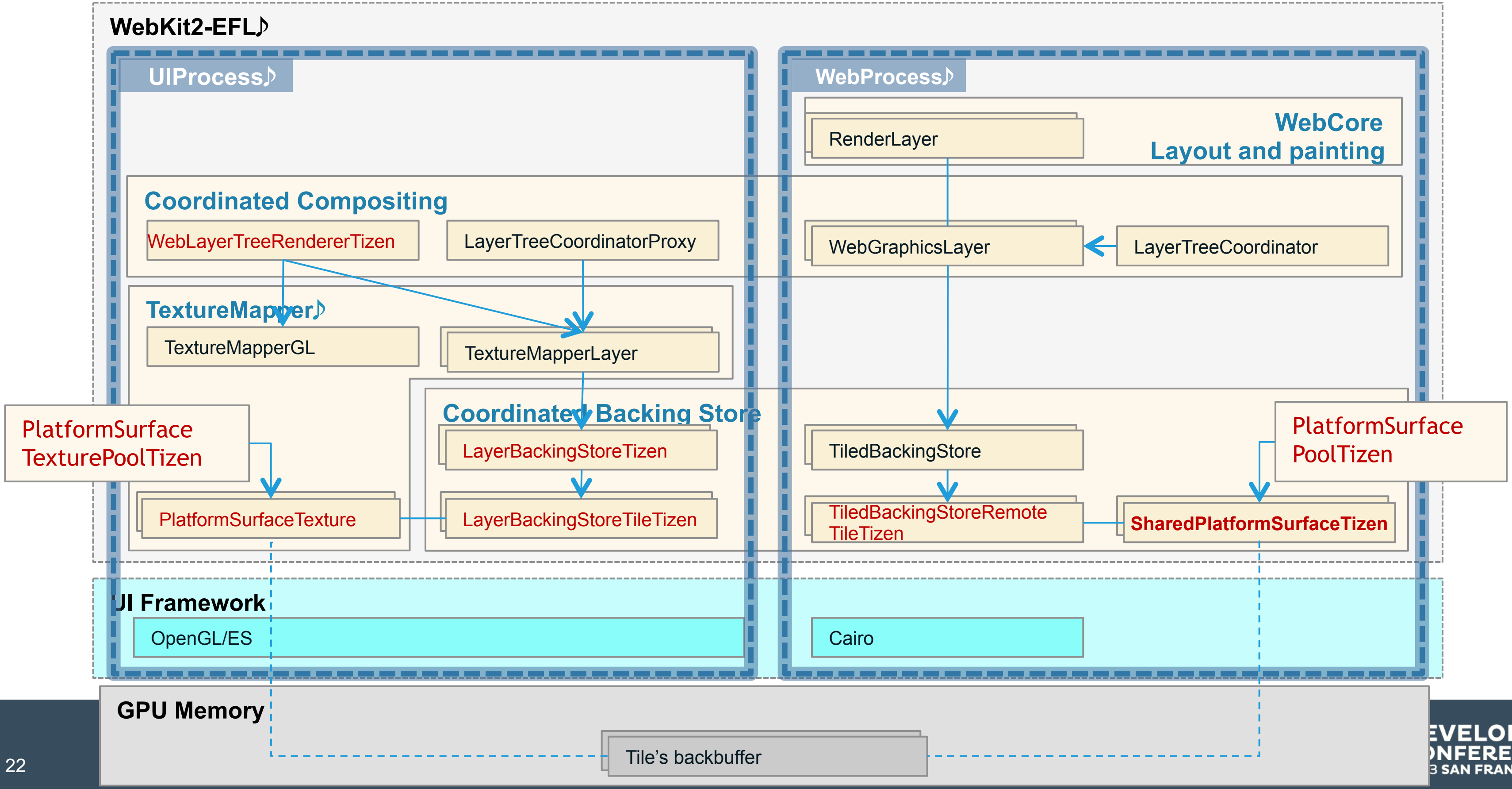
- **Graphics performance optimization**
  - Reduce memory operations
  - Different paint path per contents' characteristics
- **Memory optimization**
  - Performance-Memory trade-off
  - Release as much memory as possible in the suspended state

Appendix

Coordinated Graphics



Tizen Coordinated Graphics



TIZEN™

DEVELOPER CONFERENCE

2013

SAN FRANCISCO