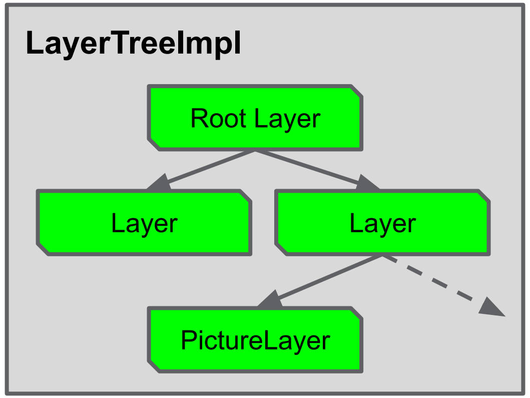
Threaded GPU Rasterization crbug.com/454500

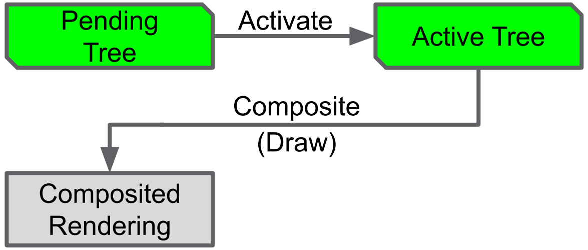
State of GPU Rasterization (M-42)

Architecture circa M-42

Quick impl-side-painting recap

- Compositor (CC) maintains snapshots of Blink world in **LayerTrees**.
- LayerTrees contain **PictureLayers**, which are content layers. They need to be rasterized to GL textures for GPU compositing to happen.
- As rasterization can be slow, we have a system with two LayerTrees:
  - **Active tree** - is the current tree used for compositing. It's visible PictureLayers are already rendered *(we hope)*, and we can scroll, zoom, or apply CSS transforms to them.
  - **Pending tree** - is an incoming update from Blink, with new or invalidated layers. We want to make this Active ASAP, but it's visible PictureLayers need to be rasterized first - so we don't *flash* empty layers to the user.

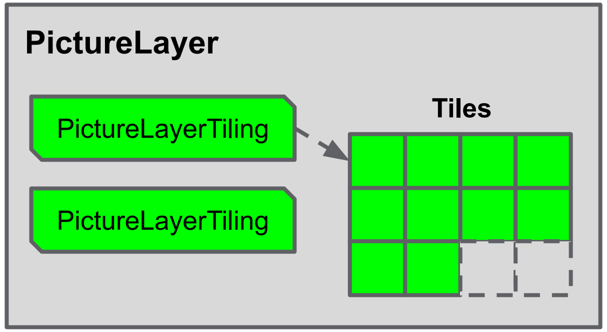




Architecture circa M-42

Quick impl-side-painting recap

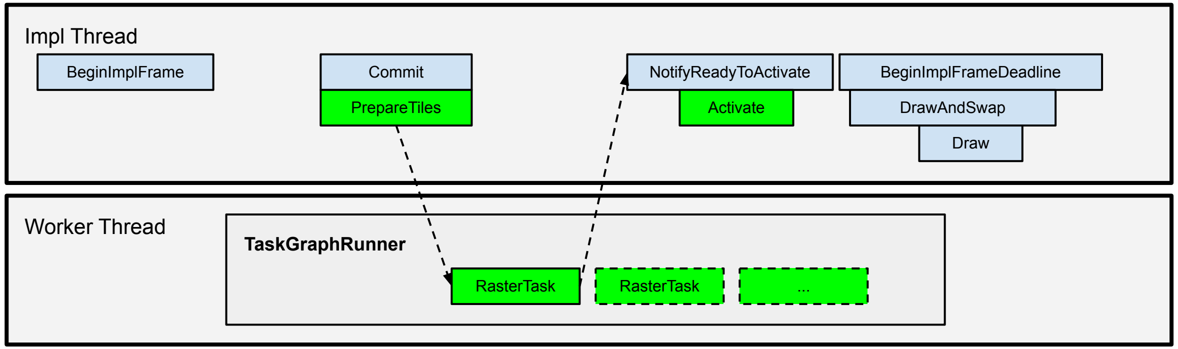
- A **PictureLayer** is cut into smaller 2D portions (**Tiles**) used for tracking the actual rendered GL textures, visibility culling.
- Content may be rasterized at more than one scale (scale animations, pinch-zoom), and PictureLayers keep Tiles of different scales in one or more **PictureLayerTilings**.
- Tiles are prioritized, allocated, and scheduled for rasterization via the **TileManager**.
- Scheduled tiles are rasterized via **GpuTileTaskWorkerPool**, **TaskGraphRunner,** and of course **Skia** rasterizer.
- More detail on impl-side-painting design: http://www.chromium.org/developers/design-documents/impl-side-painting



Google Confidential and Proprietary

Architecture circa M-42

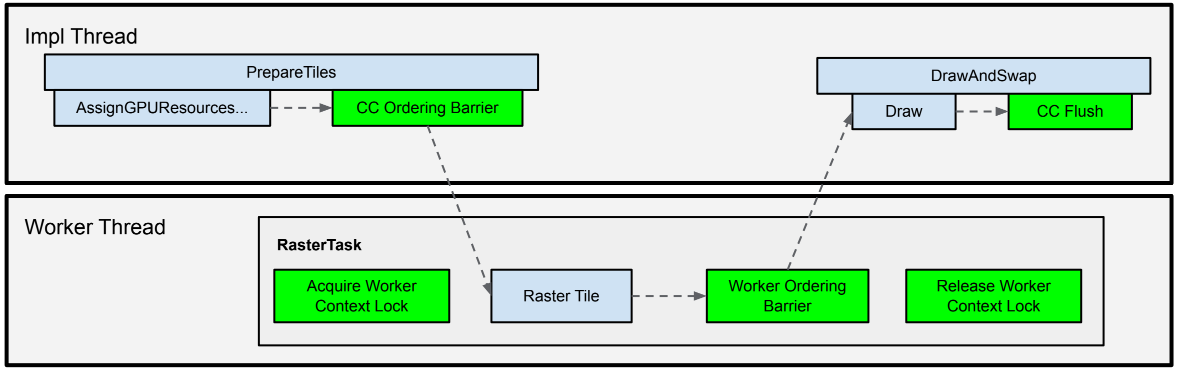
Threaded GPU Rasterization



- Threaded GPU Rasterization looks like Threaded Software Rasterization:
  - GPU **RasterTasks** are prioritized and scheduled by **PrepareTiles**
  - **TaskGraphRunner** runs scheduled tasks on a worker thread.
- RasterTasks are non-blocking. If needed tasks complete within the frame deadline, we notify and can **Activate** trees at 60fps, otherwise we keep drawing the old Active tree.
- This model matches Software Rasterization, and enables smooth scrolling & CSS animations while RasterTasks may be slow.

Architecture circa M-42

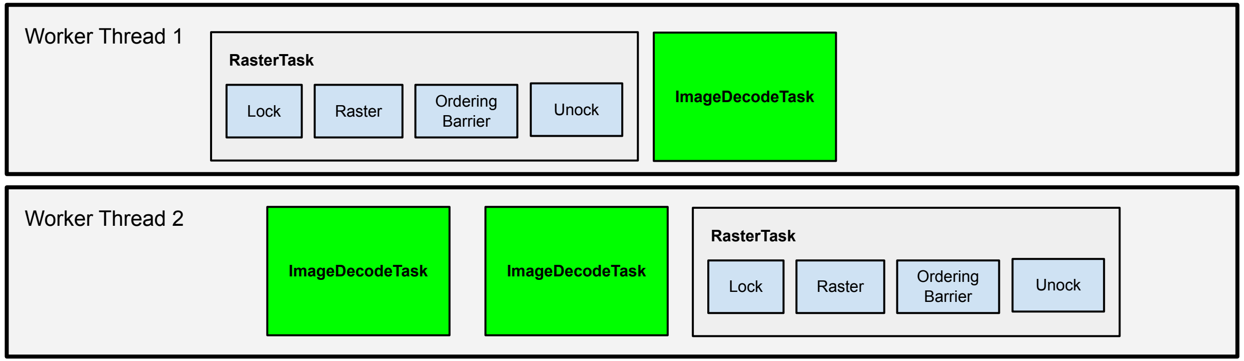
Threaded GPU Rasterization - synchronization detail



- Much of what was newly introduced for Threaded GPU is multiple GL context setup and synchronization.
  - Two GL contexts: **CC Context** and **Worker Context**.
  - Ordering between CC & Worker GL contexts is maintained by **OrderingBarrier() & Flush() APIs** [indicated by - - - >]
  - OrderingBarrier is a new Chromium GL API which allows lighter weight synchronization than ShallowFlush().
  - Access to Worker Context requires holding a **Worker Context Lock**.
    - In practice no other threads contend for the lock other than exceptional cases like handling loss of the GL context.

Architecture circa M-43

Multi-threaded GPU Rasterization (for Desktops)



- On Desktop with 4+ CPU cores, **ImageDecodeTasks** can be multi-threaded and pipelined with RasterTasks.
- RasterTasks can run on either thread, but only one at a time.

Future work

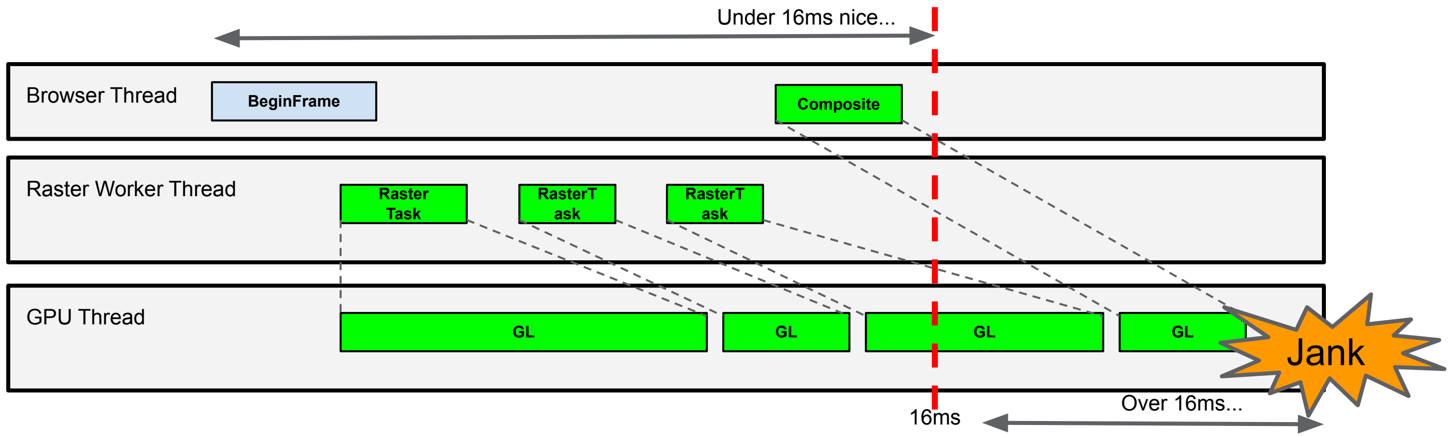
- Skia GPU projects to improve raster performance
  - Faster Path rendering - tesselated paths
  - Distance Field Fonts for fast scalable text
  - Image scaling to screen resolution
  - Draw call reordering and better batching
  - Canvas reordering and atlassing
  - More caching and reuse, e.g. of text layout

- GPU Integration projects to improve raster performance
  - Applying SkMultiPictureDraw in context of Threaded GPU Rasterization.
  - Tuning tiling sizes
  - Optimizing PicturePile analysis and playback

- Threaded GPU Architecture
  - Issues and ideas on following slides

Google Confidential and Proprietary

Towards GPU Raster on all content



- Handling all content means dealing with arbitrary content complexity - i.e. RasterTasks can take an arbitrary amount of time.
- On the Worker Thread, it's OK for tasks to run beyond 1 frame.
- On the GPU Thread, if the Compositor can't run every 16ms then we Jank.
- Need to prevent tasks that can take more than a frame: uploading lots of large images, complex path rendering, costly filters.

- **Requires effective throttling of RasterTasks.**
  a. Throttle based on criteria such as # image bytes uploaded, # pixels rendered, # render targets used.
  b. Back-pressure from GPU Thread.

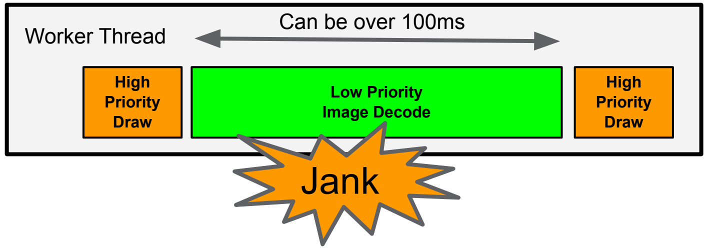
Towards preparing new content at 60fps

- Skia GPU currently draws tiles in first-in-first-out (FIFO) order.
- If we commit to doing a slow operation like preparing a tile with a large image that takes 100ms, it blocks any other rasterization.
- To run a dynamic Web App UI at 60fps we can't allow 100ms tasks on the thread that is rendering that UI.

- **Requires either:**
  a. **Run long tasks on separate threads.**
  b. **Break tasks down into small, schedulable pieces.**
  c. **Have pre-emptable tasks.**

- Picture rasterization with GrGPU is not threadable so (a) is not currently an option, but pictures are composed of many smaller rendering operations which means (b) breaking down into smaller tasks or (c) pre-empting larger rendering tasks are feasible.

- Image decoding may use codecs outside of our control, so (b) or (c) may not be an option, leaving (a) threading. This requires investigation.


Worker Thread
Can be over 100ms
High Priority Draw
Low Priority Image Decode
High Priority Draw
Jank