Multi-threaded Rasterization

aka impl-side painting
aka multi-threaded painting

enne@, nduca@, &c

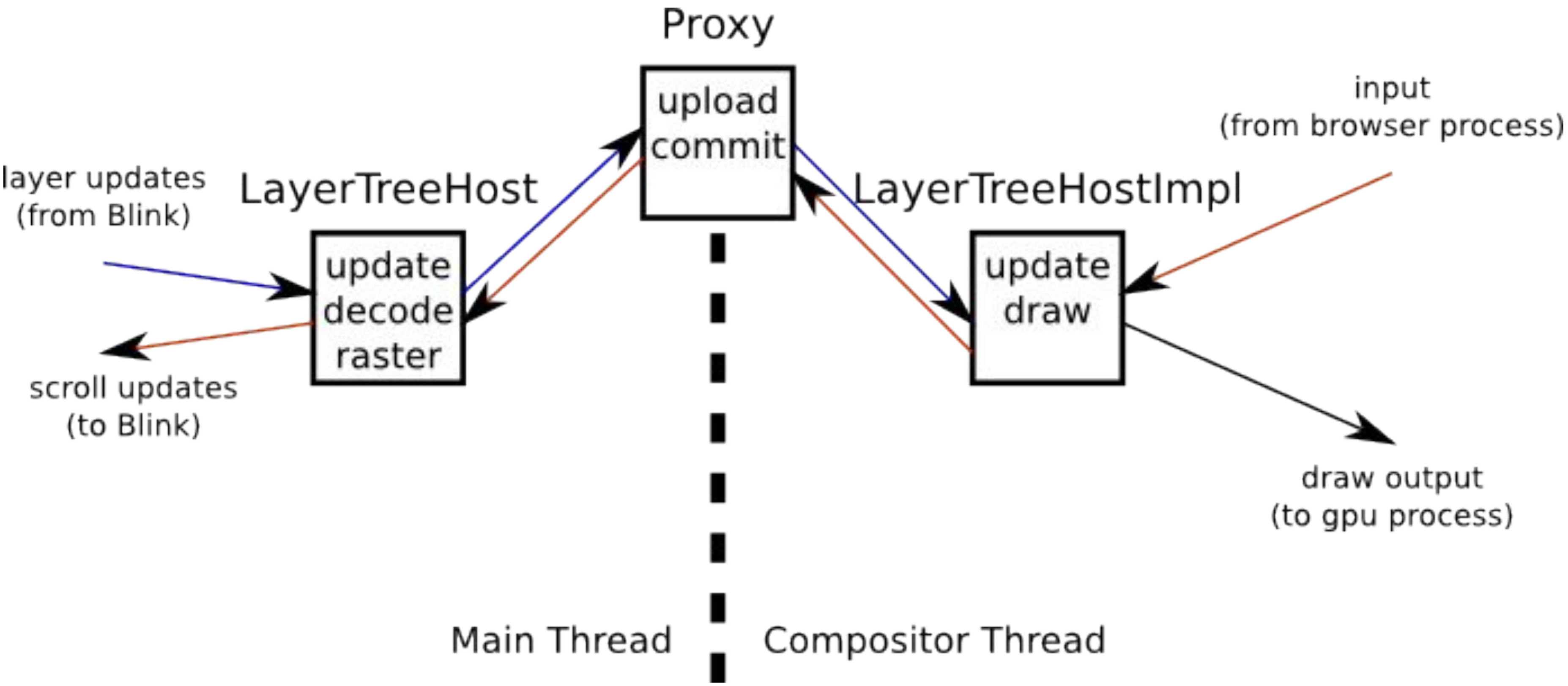
Outline

- Background: threaded compositing
- Motivation
- v1.0 Architecture overview
    - Two trees & activation
    - PicturePile recordings
    - PictureLayerTiling rasterization
    - Tile Manager, prioritization, and uploads
- Work still to be done for v2.0

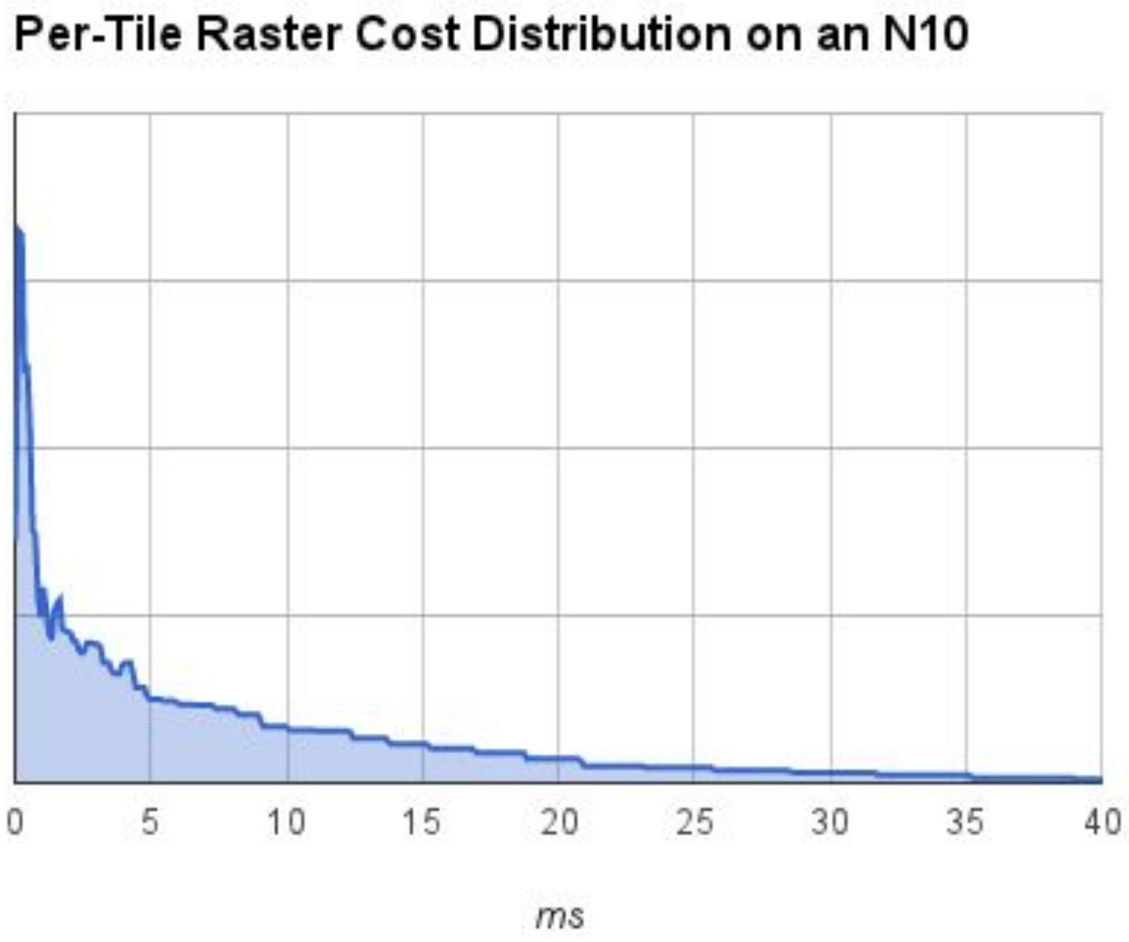
Intended audience

- Chromium cc hackers
- Chromium gpu hackers
- Blink & Skia hackers
- Hardcore webdevs

Threaded Compositing Background



Long tail of raster costs is slooow



Threaded Compositing Wrap-up

Positives:

- Super architecture for threaded scrolling.
- Jank-free hotness.

Negatives:

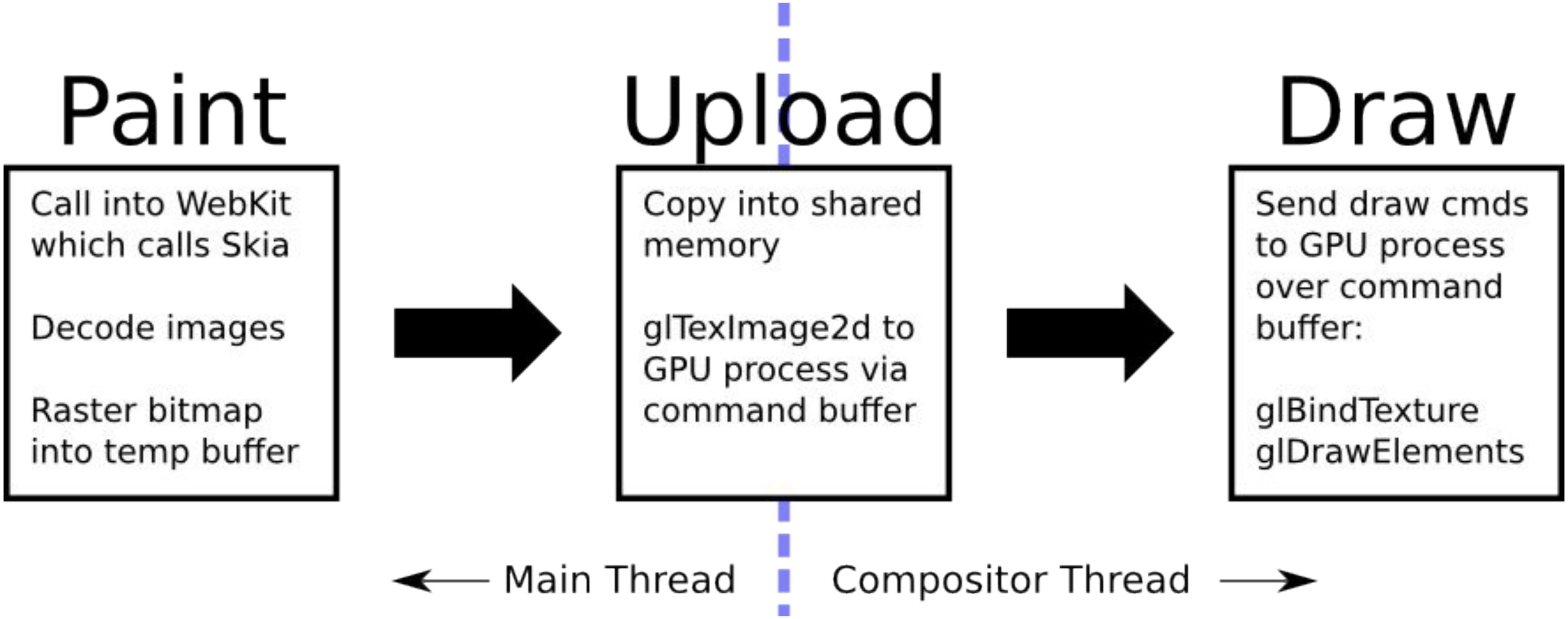
- Rastering additional content requires a slow round trip through the main thread.
- Prepainting is awkward and out of date.
- Awkward upload throttling.
- Long commits

Why, impl-side painting?

P0 bug: Reduce checkerboarding

(and try to clean up architectural messes while we're at it)

Life of a texture (before)



Paint

Call into WebKit which calls Skia

Decode images

Raster bitmap into temp buffer

Upload

Copy into shared memory

glTexImage2d to GPU process via command buffer

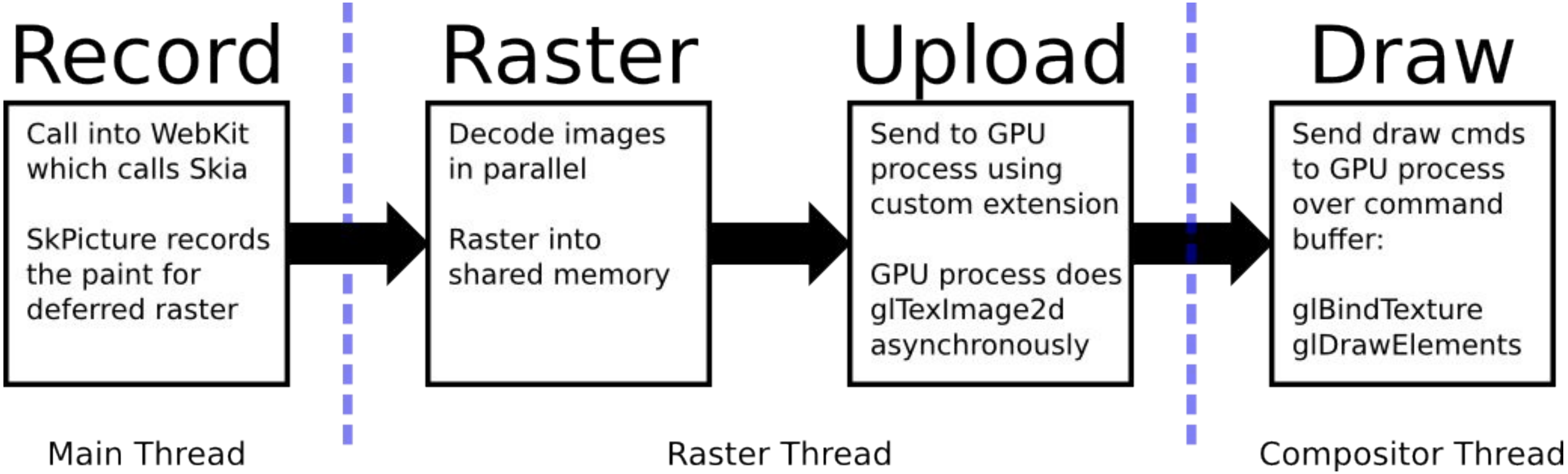
Draw

Send draw cmds to GPU process over command buffer:

glBindTexture
glDrawElements

← Main Thread | Compositor Thread →

Life of a texture (after)



Record

Call into WebKit which calls Skia

SkPicture records the paint for deferred raster

Raster

Decode images in parallel

Raster into shared memory

Upload

Send to GPU process using custom extension

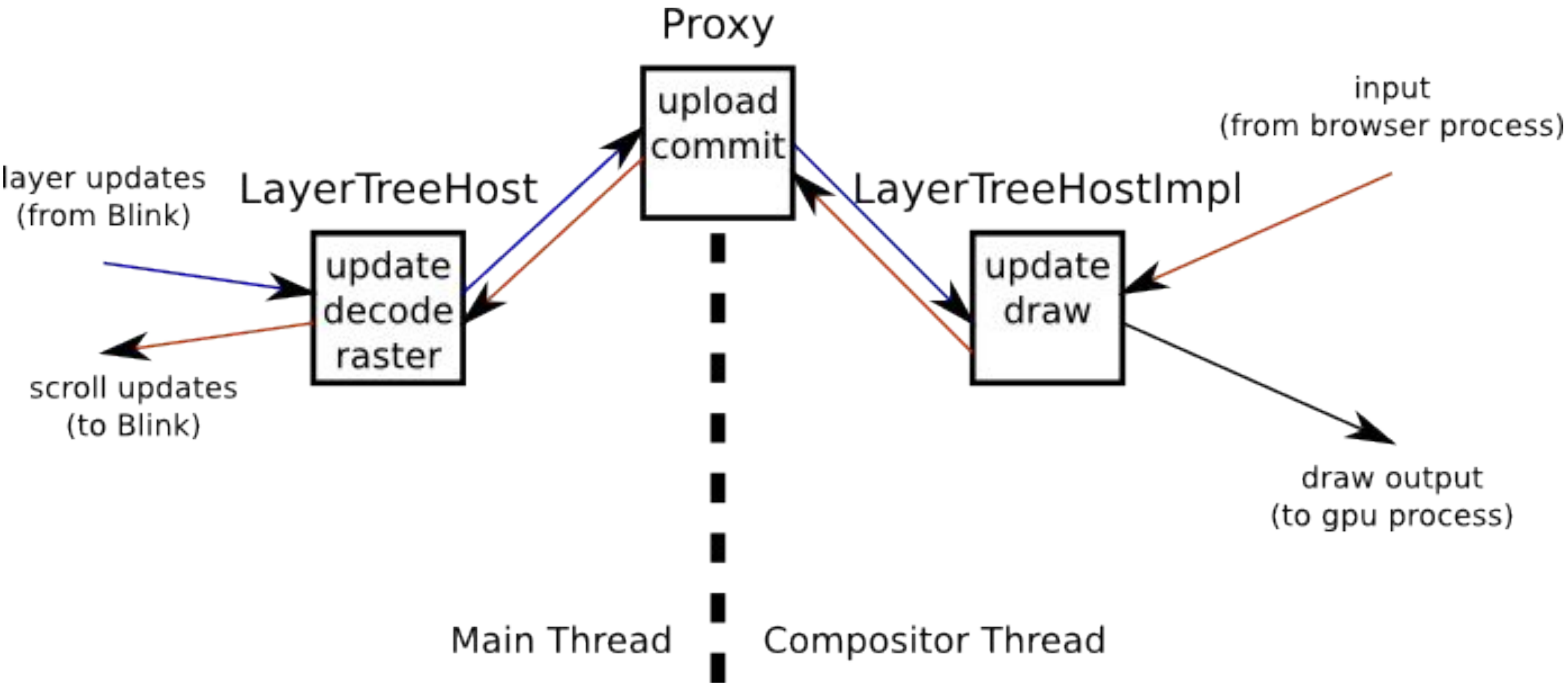
GPU process does glTexImage2d asynchronously

Draw

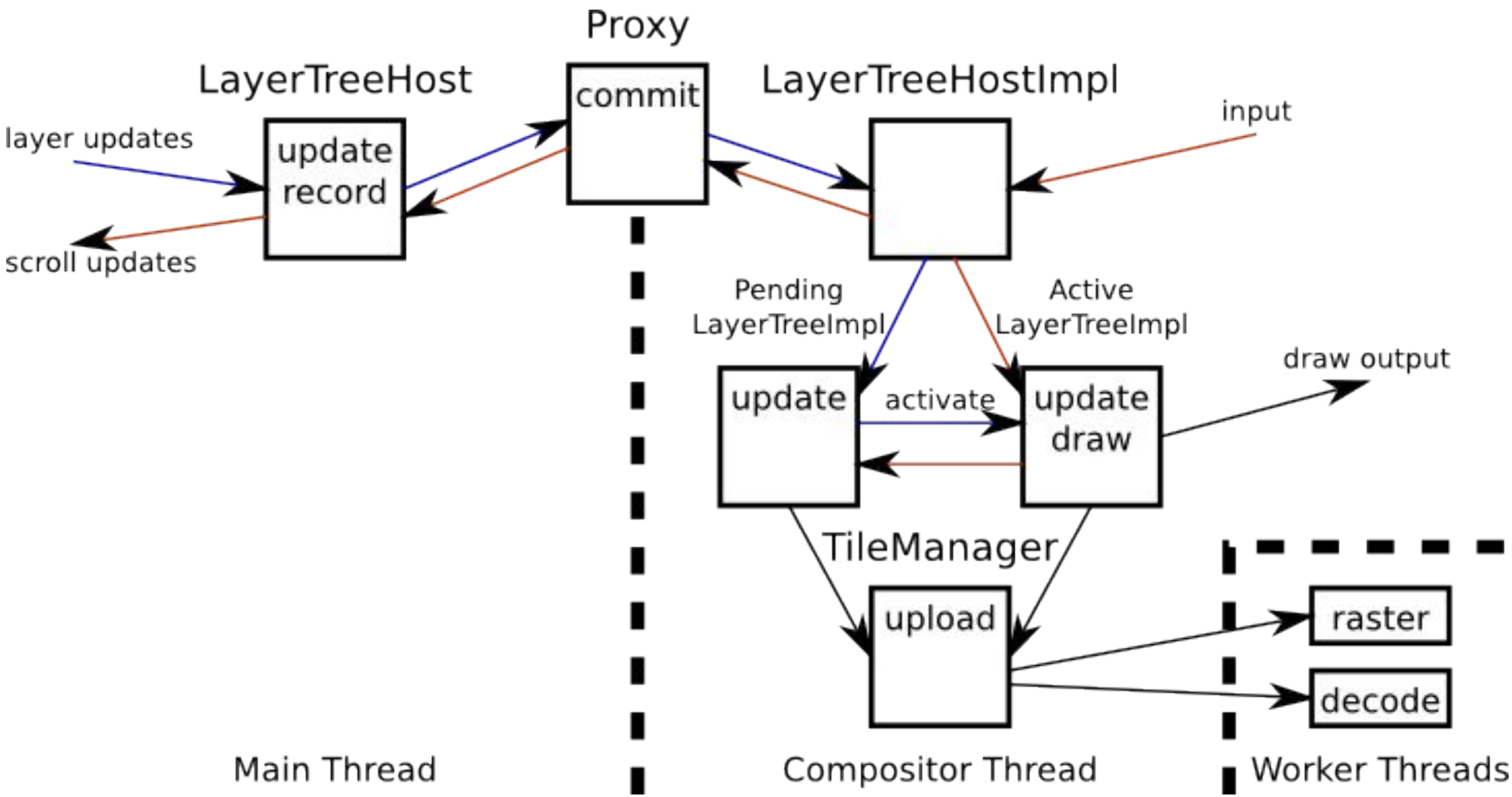
Send draw cmds to GPU process over command buffer:

glBindTexture
glDrawElements

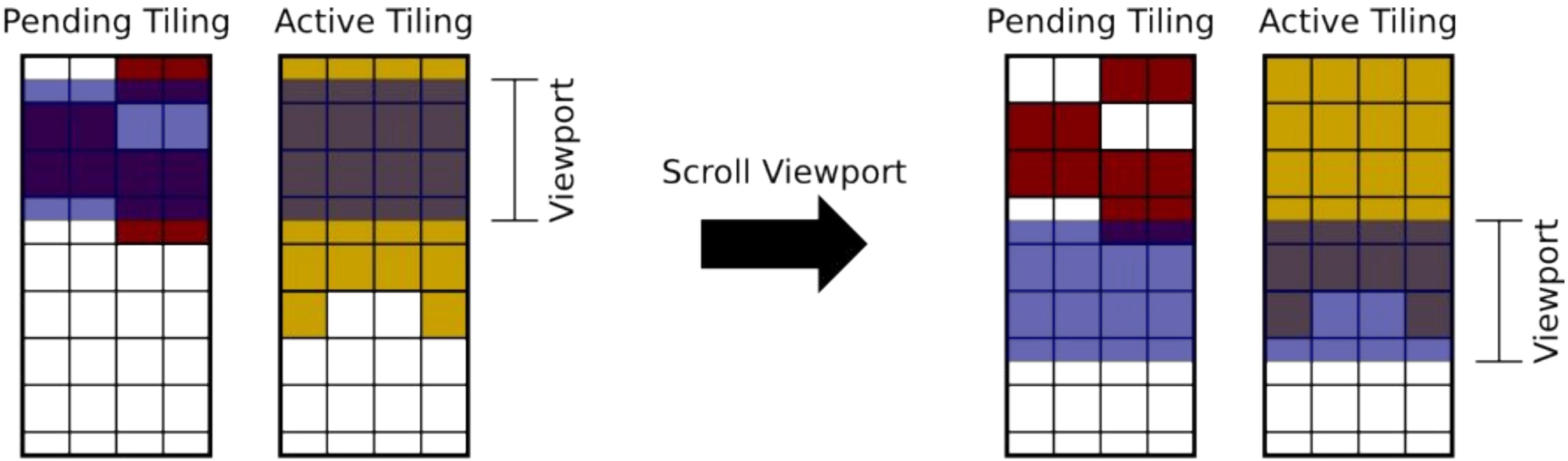
Data flow with trees (before)



Data flow with trees (after)



Rastering from two trees

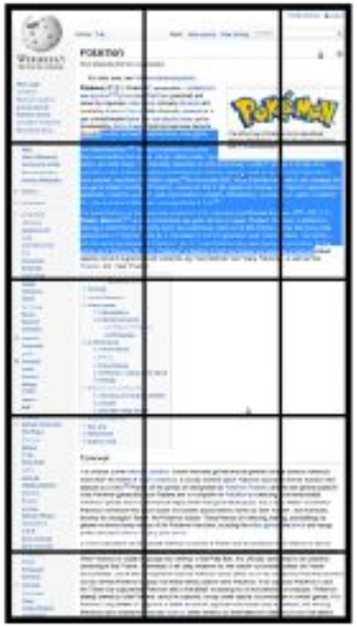


Pending tiling needs to raster 6 tiles.
Active tiling has no missing visible tiles.
Activation not possible due to missing tiles.

Pending tiling missing 14 tiles.
Active tiling is now missing 6 tiles.
Activation still not possible.

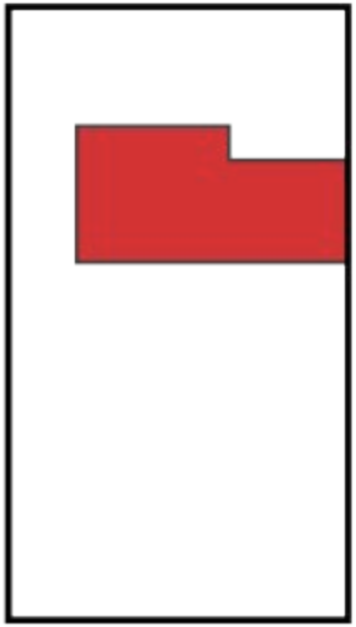
Handling Incremental Invalidation

Pending Tiling



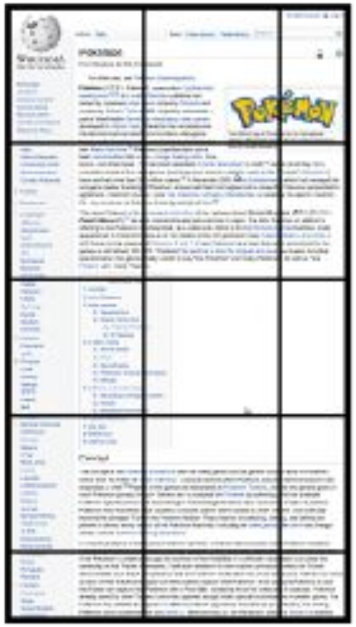
Main thread frame n+1

Layer Invalidation



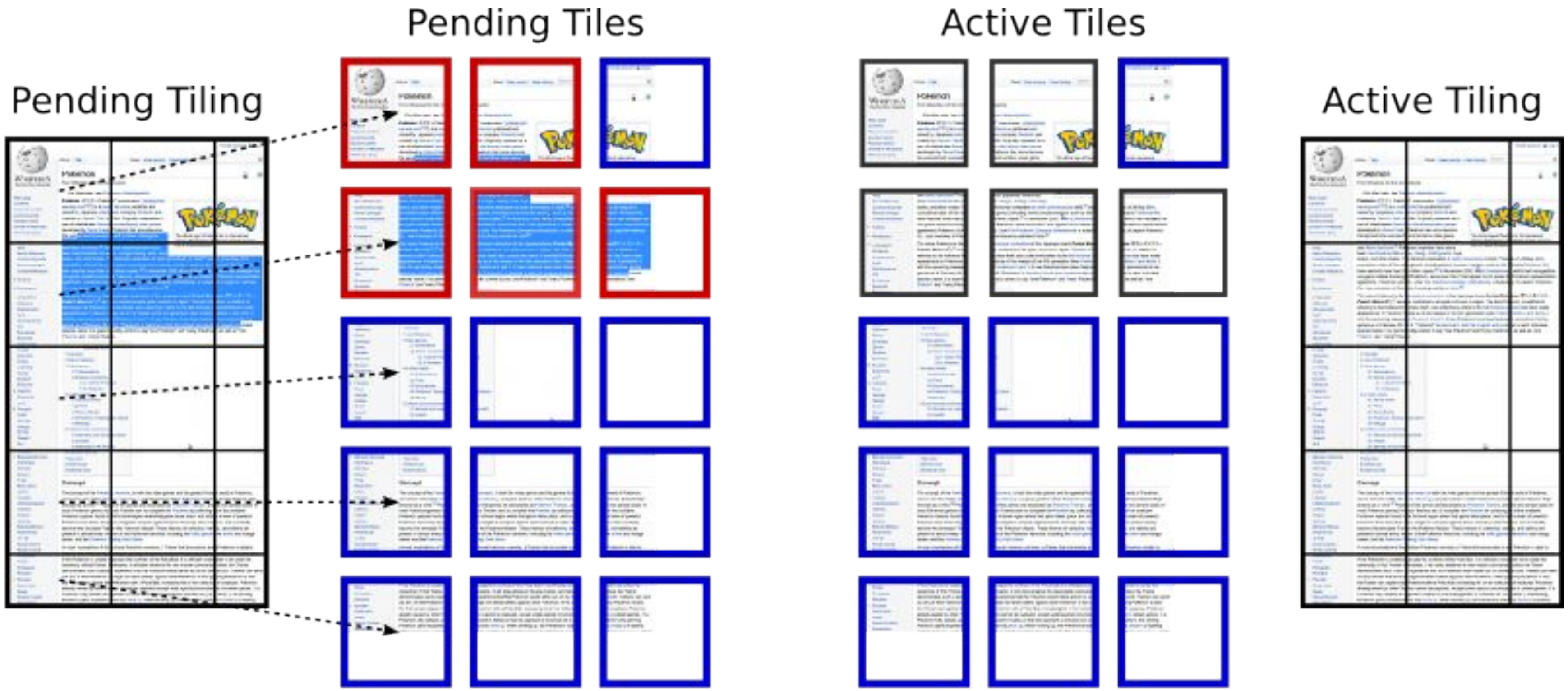
Diff between frames

Active Tiling



Main thread frame n

Tile sharing between trees

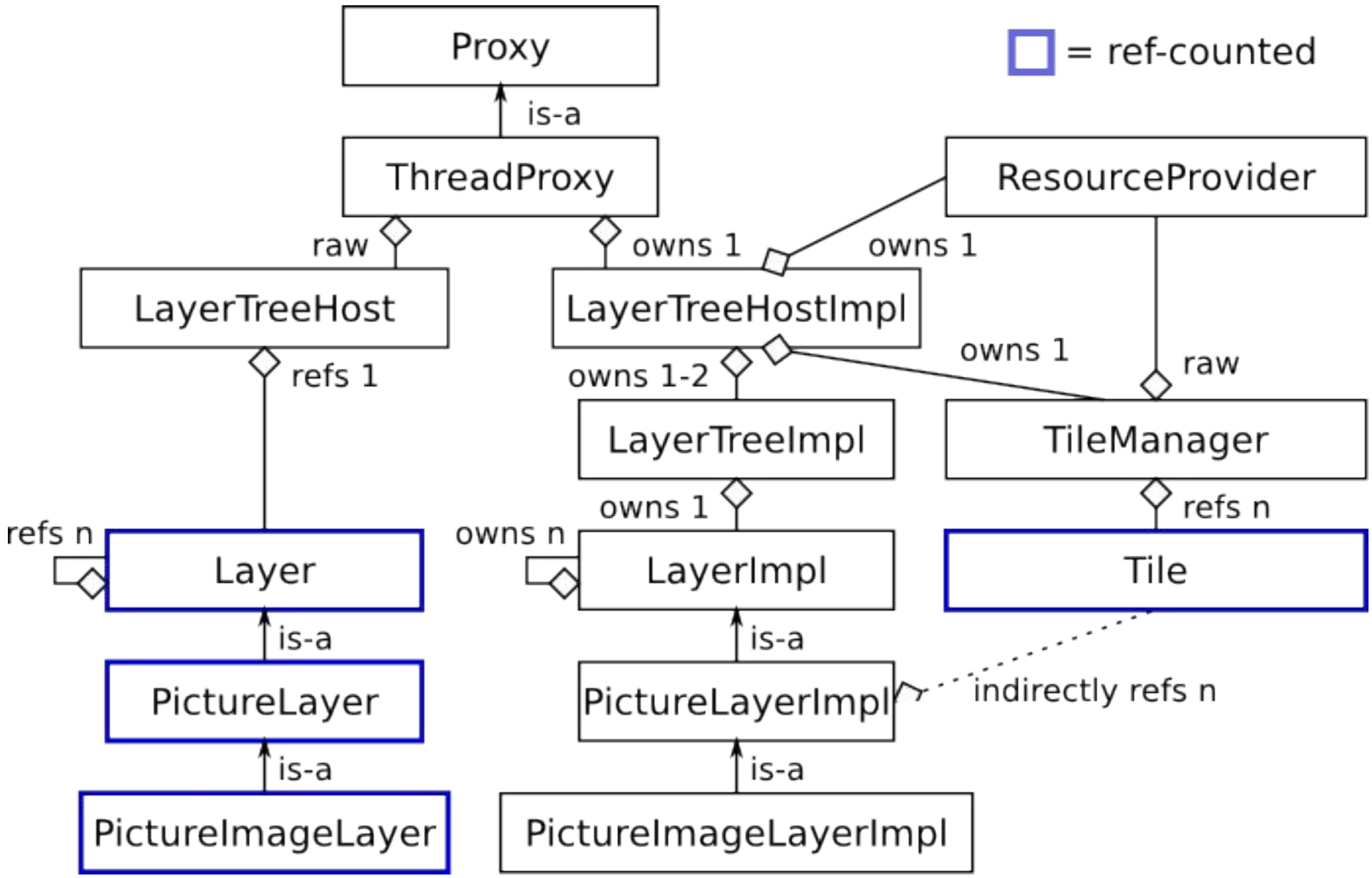


🟦 = both tilings share the same tile object

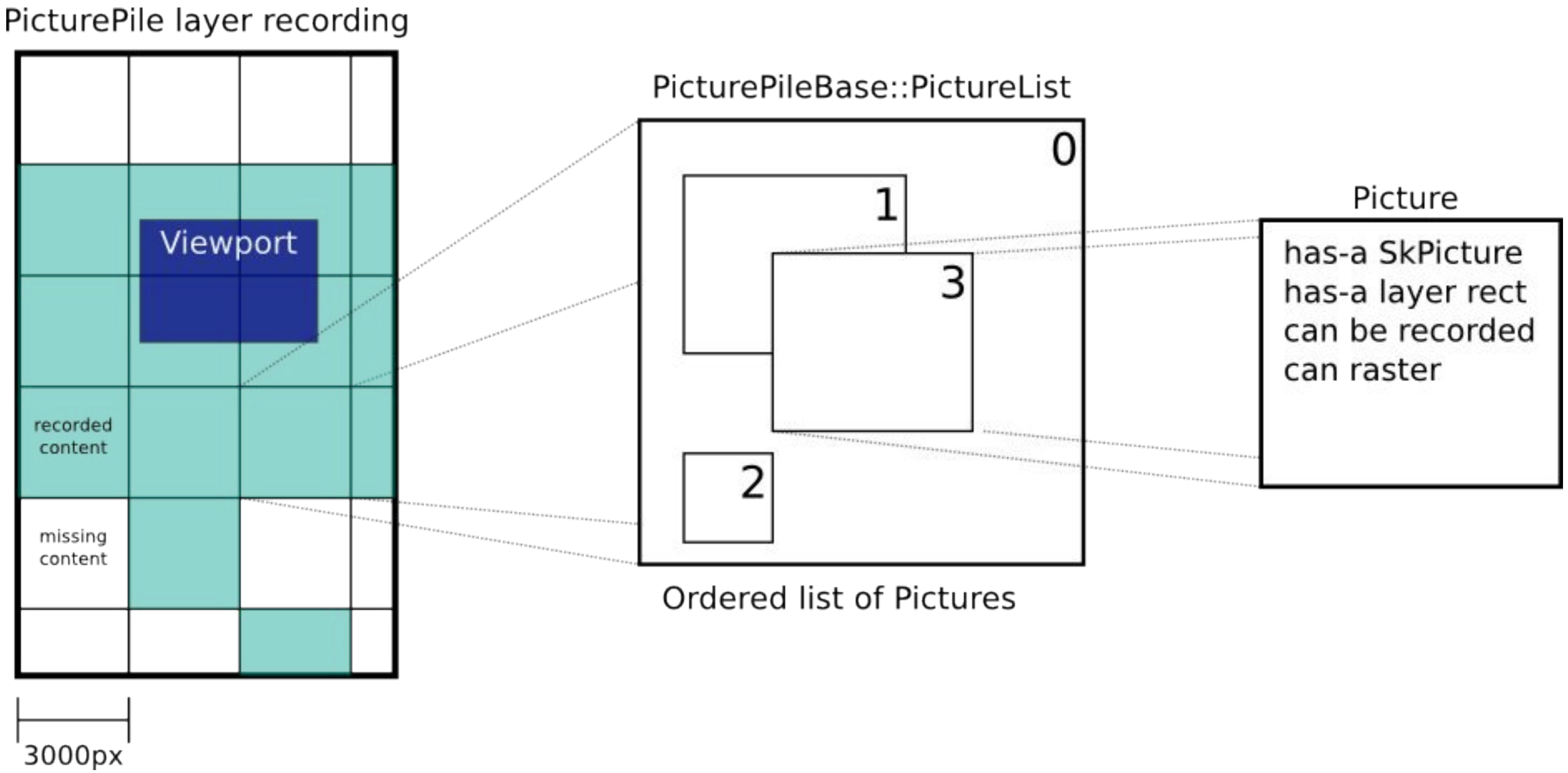
🟥 = new pending tiles due to invalidation

□ = unshared active-only tiles

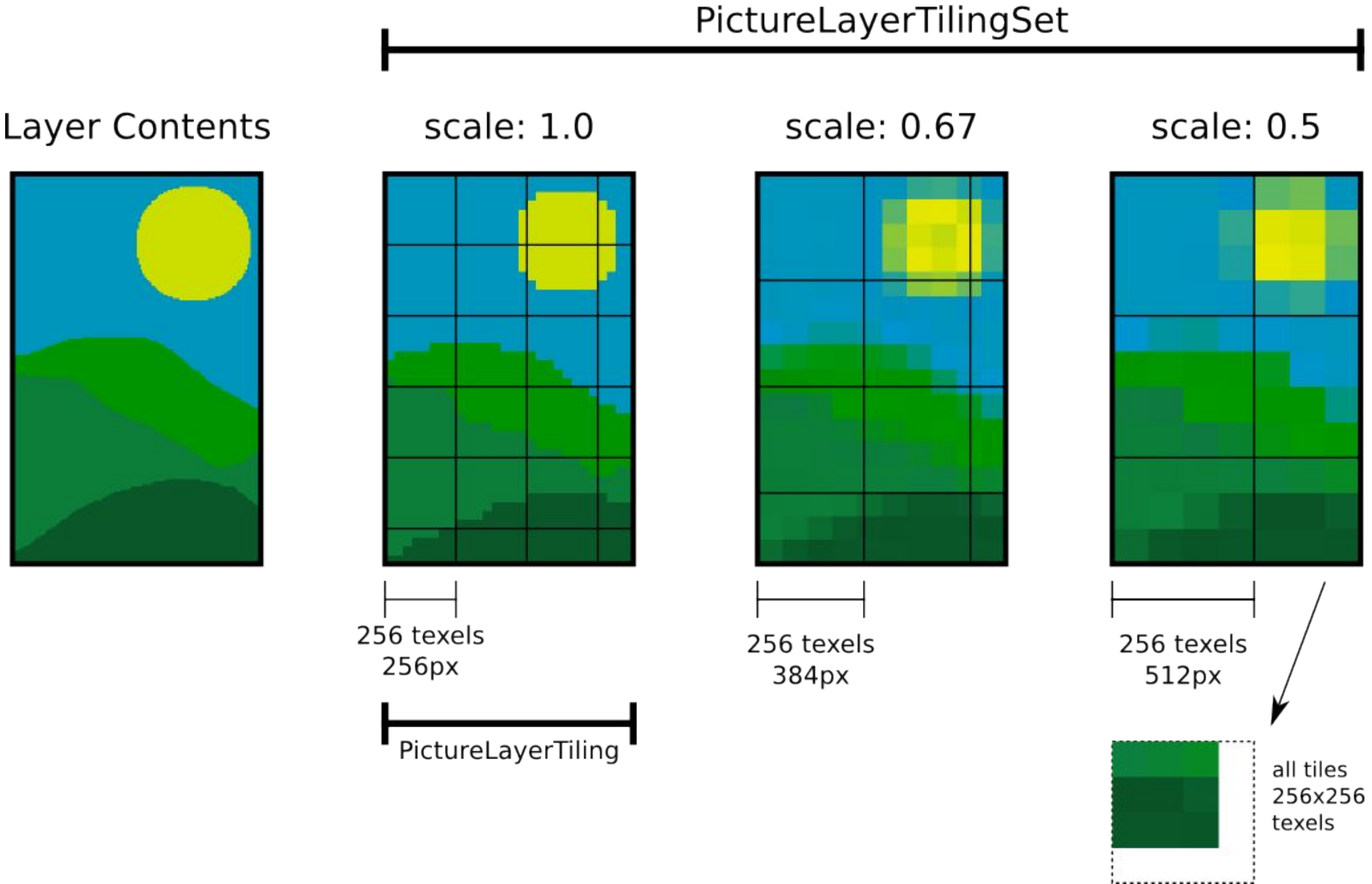
Hosts and Trees Class Diagram



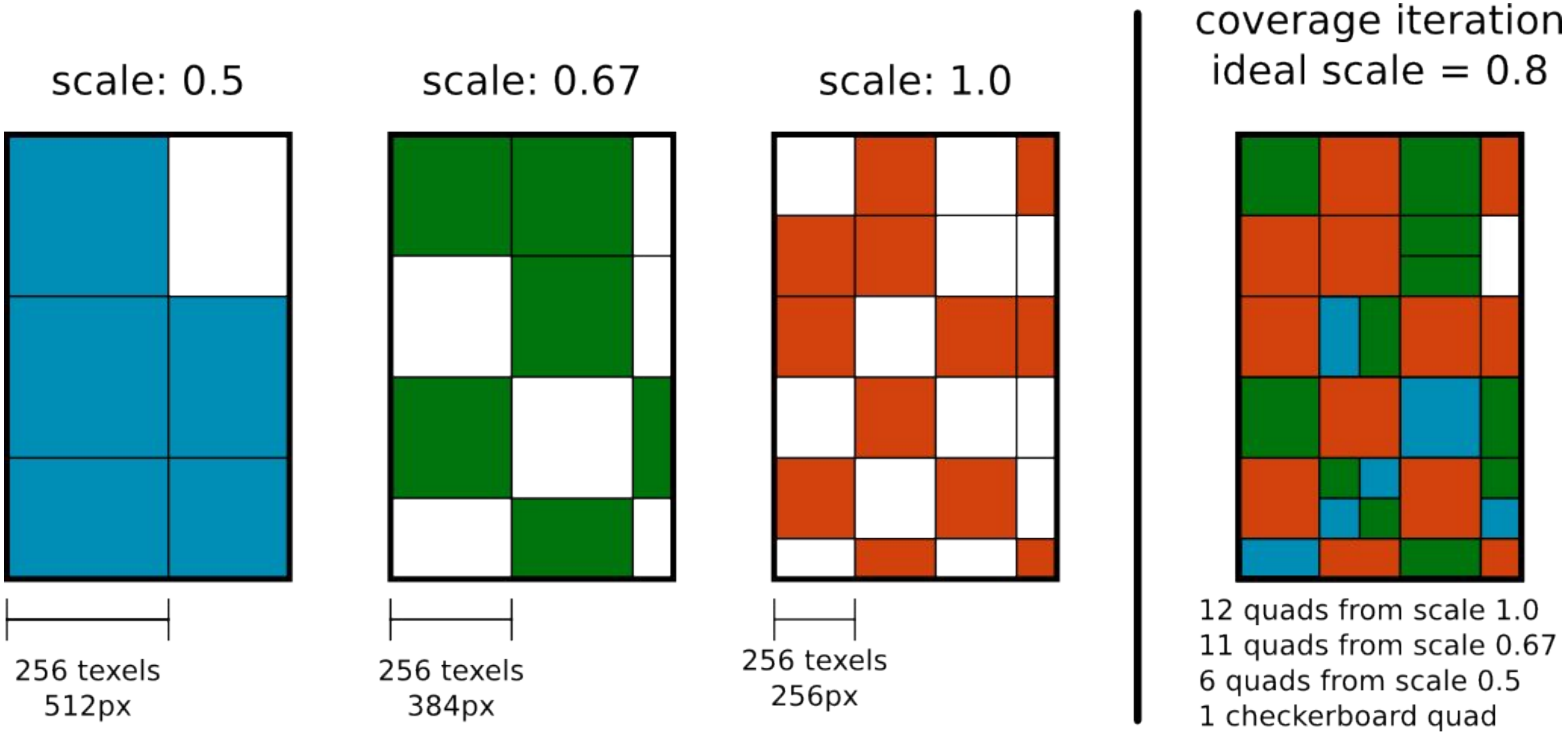
Picture Pile overview



PictureLayerTiling Overview



Iterating Through Tilings



scale: 0.5

256 texels
512px

scale: 0.67

256 texels
384px

scale: 1.0

256 texels
256px

coverage iteration
ideal scale = 0.8

12 quads from scale 1.0
11 quads from scale 0.67
6 quads from scale 0.5
1 checkerboard quad

A Very Common Misunderstanding

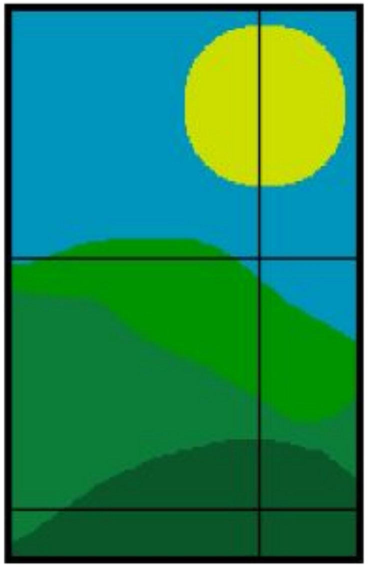
PicturePile(Impl)

- Sparse recording in layer space

- Each tile is a list of cc::Pictures (aka SkPicture)

- Very coarsely tiled (3000px) in layer space

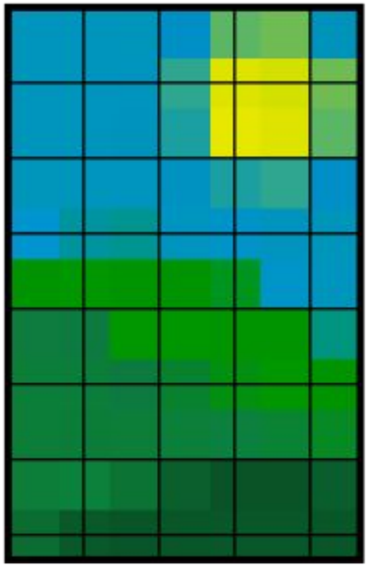
- Used as an input for rasterizing by multiple tilings



Recording

VS

PictureLayerTiling



Rasterization

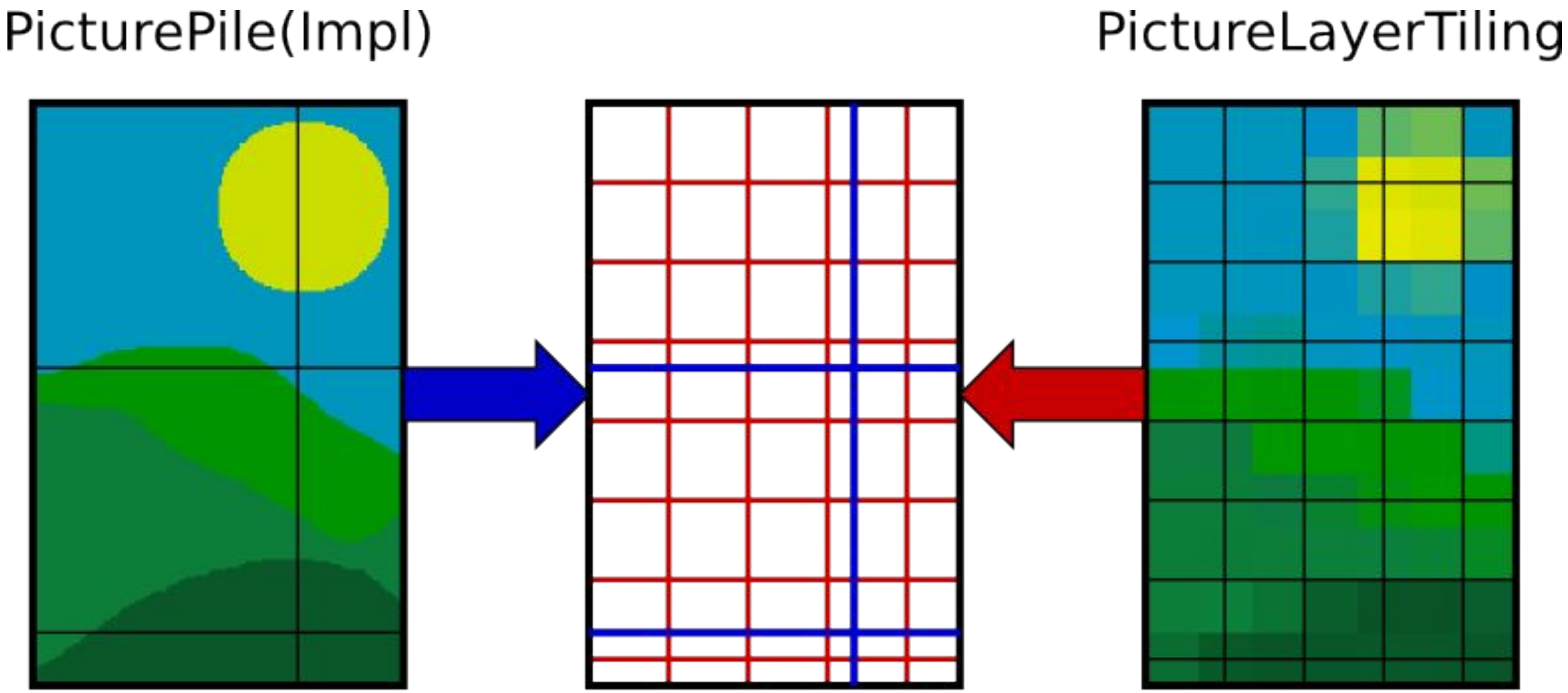
- Sparse rasterization in content space at an arbitrary scale

- Each tile is a cc::Tile resource (often a texture)

- Usually tiled at 256 or 512 texels in content space

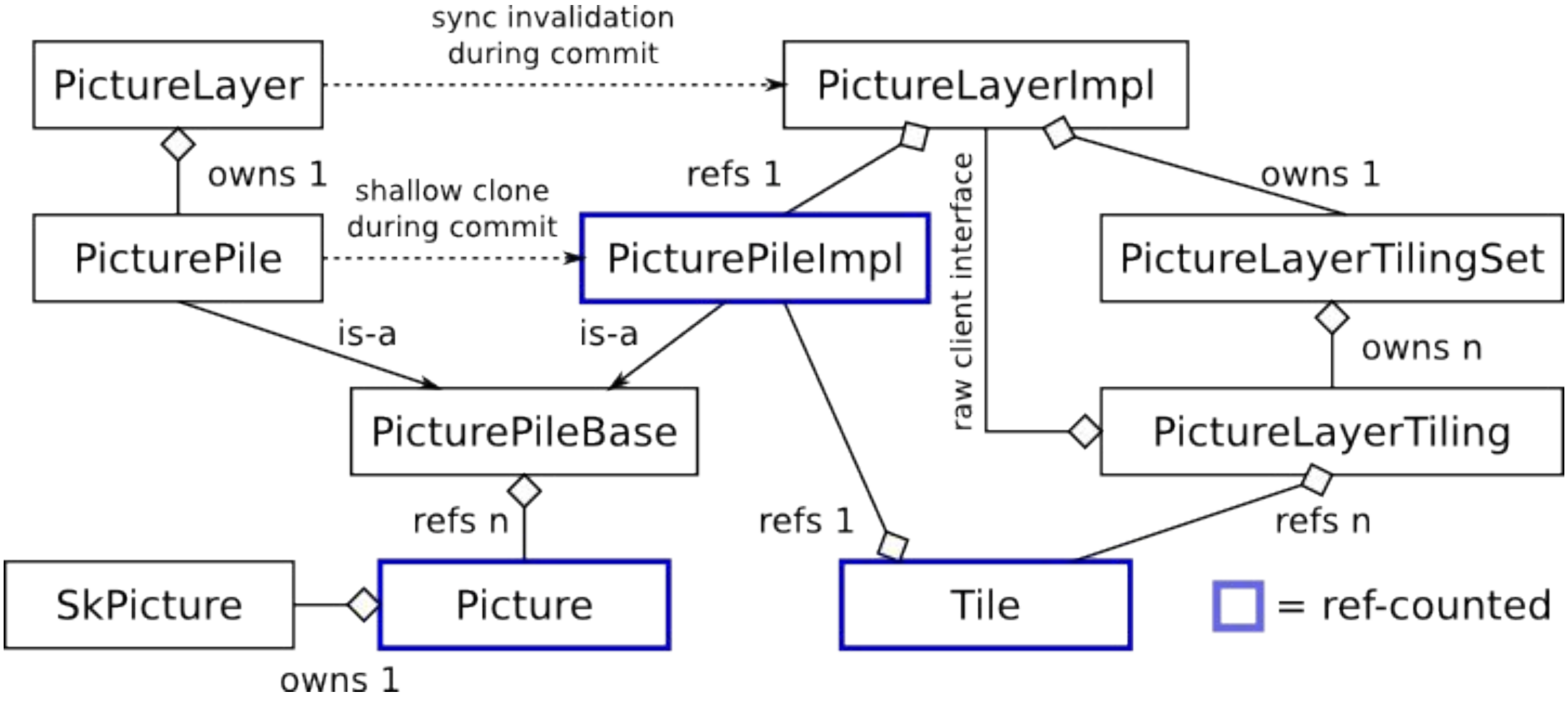
-Used as an input for drawing

A Very Common Misunderstanding



Both tilings cover the entire layer, but their tile size is entirely unrelated.

Picture Layer & Pile Class Diagram

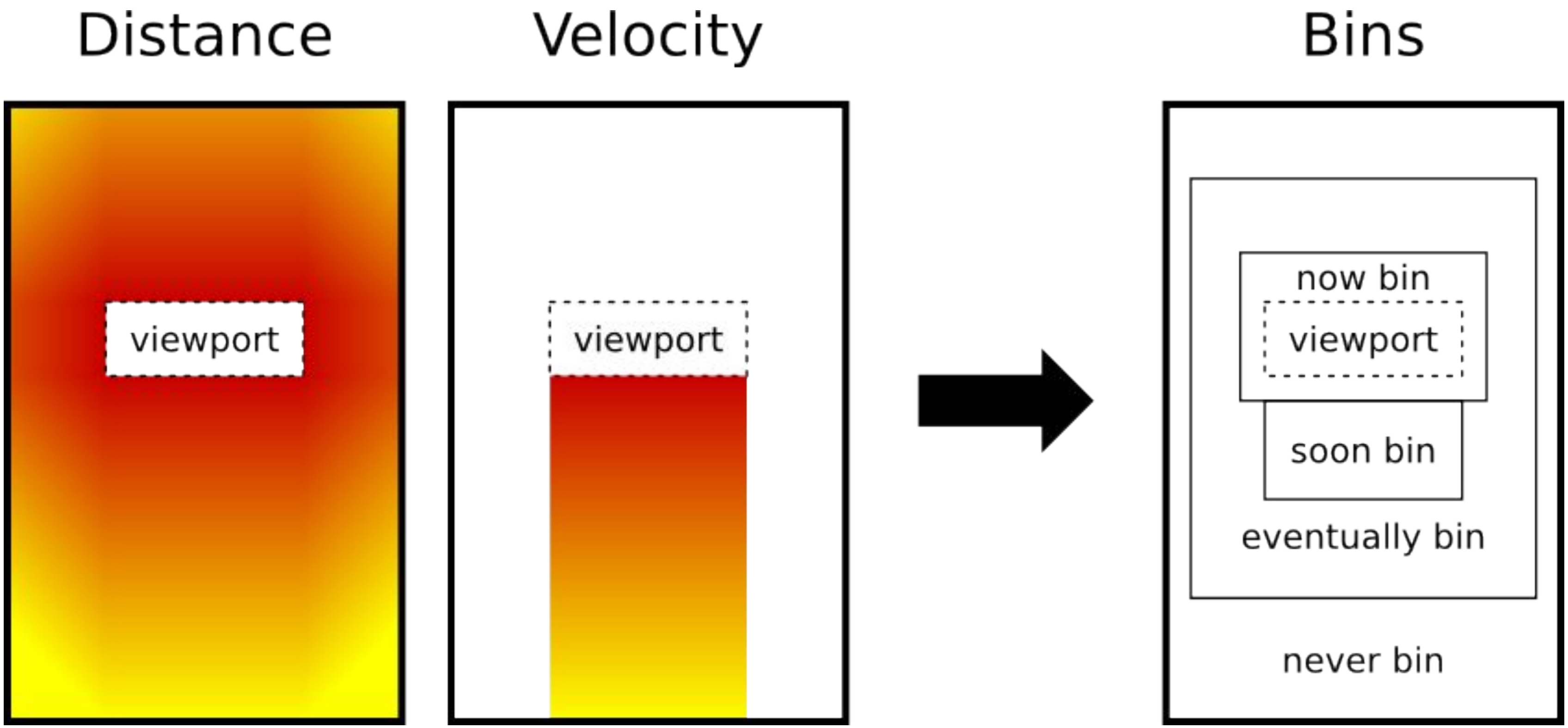


Deferred rasterization means

Tile Manager Overview

- Knows about all tiles in the universe
- Ignorant about layers
- Sorts tiles based on global state and individual tile priority
- Assigns GPU memory in order of highest priority
- Kicks off raster / decode tasks to RasterWorkerPool

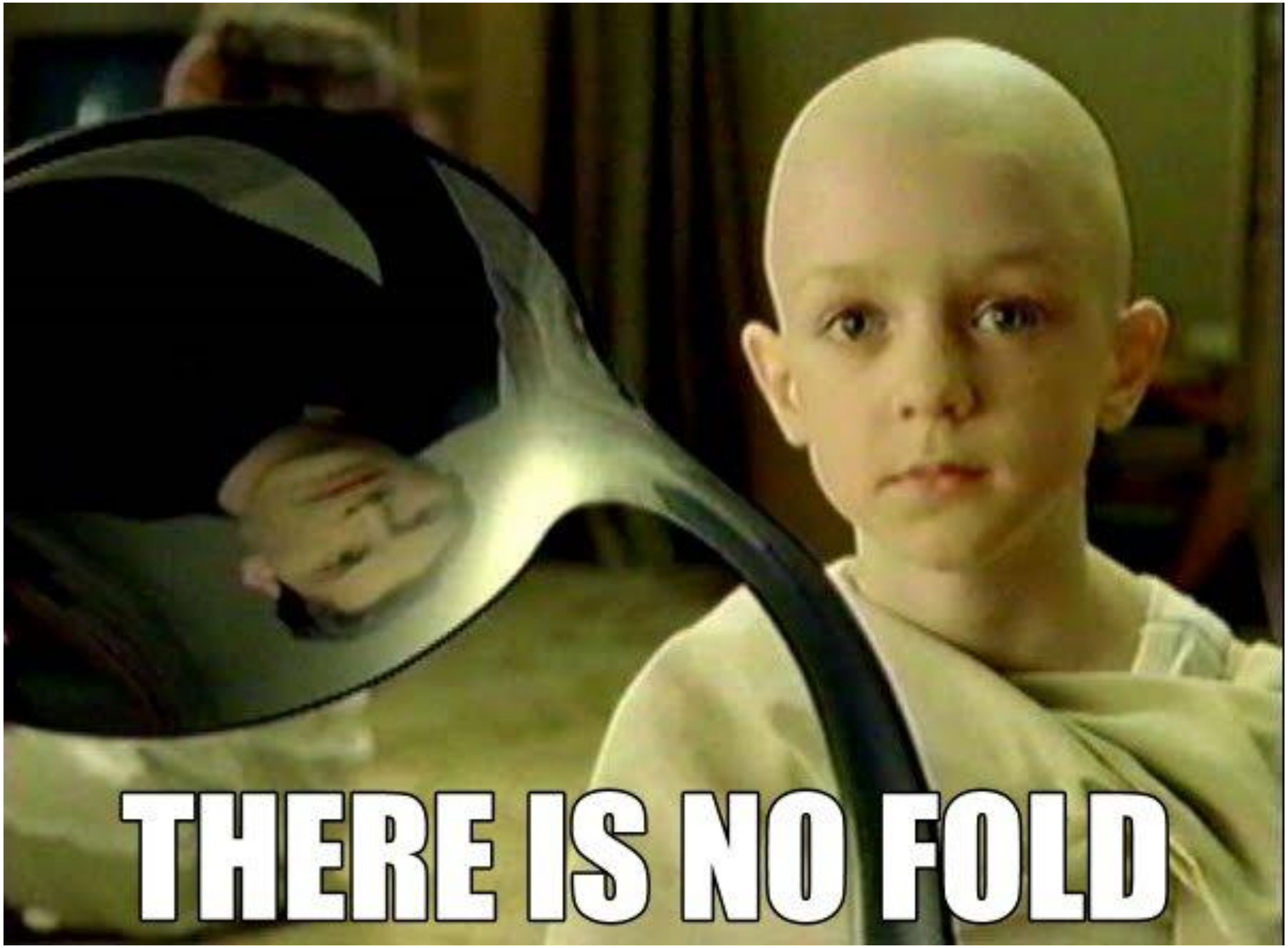
Tile Priorities



Tile Priority Philosophy

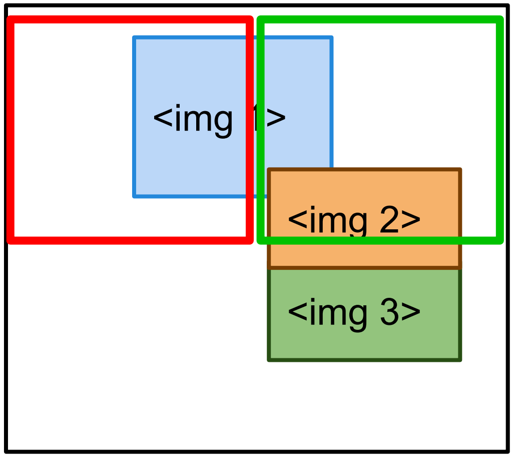
NO SPECIAL CASES

An aside to Blink engineers

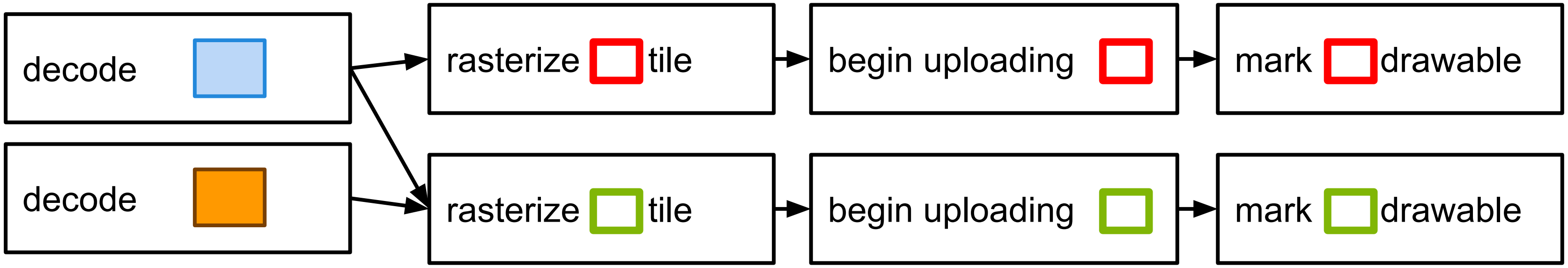


Tile Manager Jobs

If we rasterize two tiles:



then this is our task graph:


decode 🟦

decode 🟧

rasterize 🟥 tile

rasterize 🟩 tile

begin uploading 🟥

begin uploading 🟩

mark 🟥 drawable

mark 🟩 drawable

Uploading Woes of the Past

Observations:

- Uploads are slow, highly variable in length.
- Uploads are serialized with other GPU commands from the compositor.
- Too many uploads janks the frame.
- Compositor can't make good decisions about how many uploads to not jank.

Asynchronous Upload (Android)

Add an asynchronous texture upload extension:

asyncTexImage2DChromium

Poll completion with custom occlusion queries.

Upload on a separate thread in GPU process.

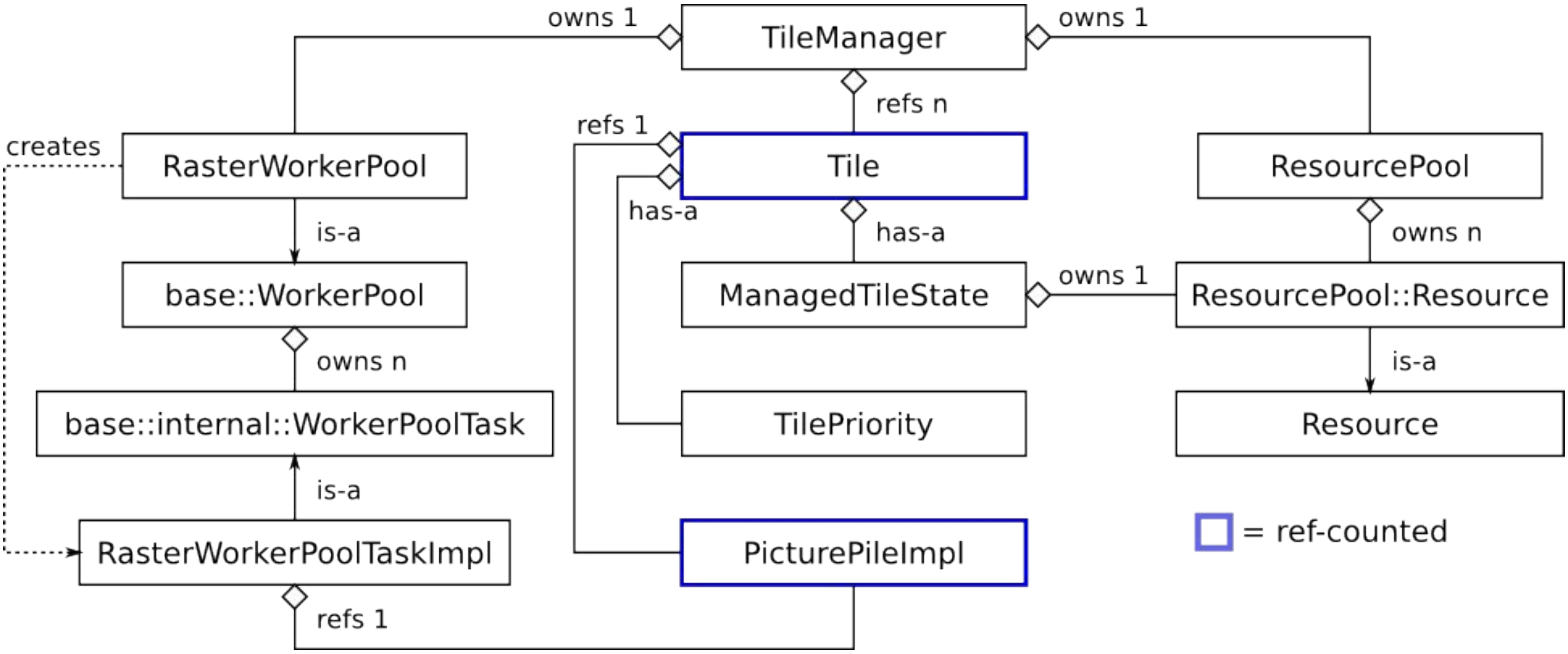
Use EGL Image extension in the GPU process to avoid having to swizzle textures on upload.

Asynchronous Upload (ChromeOS)

Same asynchronous upload extension, but runs on the main GPU thread.

Uploads are done "when idle", with some logic to force uploads to prevent starvation.

Tile Manager Class Diagram



□ = ref-counted

Tile Manager Analysis Opportunities

Solid Color Tile Optimization:

If a tile will end up being just a solid color, don't raster it or waste GPU memory on it.

Cheap Tile Latency Optimization:

If a tile is really cheap to raster, raster and synchronously on the compositor thread to avoid paying for the latency of a worker.

Recap Architectural Wins

- Less checkerboarding by doing rasterization on the compositor thread
- Prepainting happens much more naturally ("no special cases")
- Memory management all done on one thread
- Uploads much faster and more naturally throttled

Some Architectural Losses

- Occlusion culling is awkward architecturally
- More parallelism often means more latency
- Invalidates pixel-based optimizations in Blink

Work To Be Done (in rough order)

- Decode images at the ideal resolution
- Measure and optimize (everything)
- Add low quality mode, for faster low res
- Get this turned on for desktop Chromium
- Improve tile manager performance

The EndImpl