Compositing in Blink: Recap, update, and squashing

January 2014

Compositing:

(in the context of rendering websites)

The use of multiple backing stores to cache and/or group chunks of the render tree.

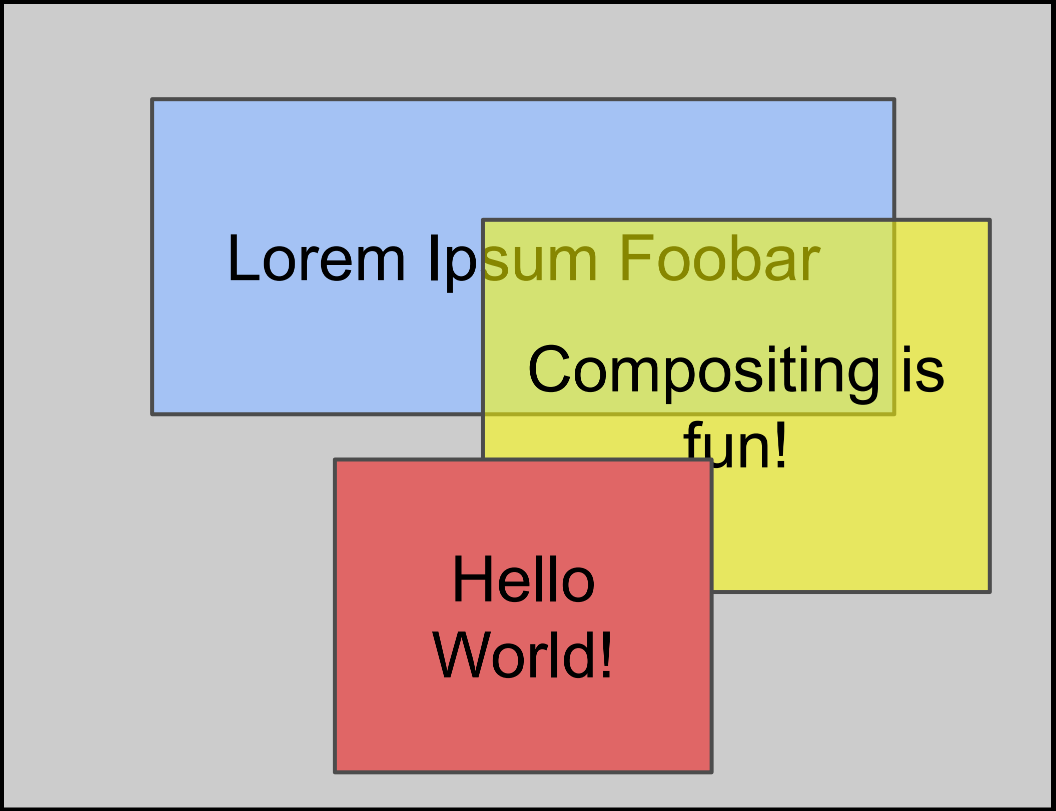
Compositing:

(in the context of rendering websites)

The use of multiple backing stores to cache and/or group chunks of the render tree.

(Backing store == GraphicsLayer)

Primary benefit of compositing

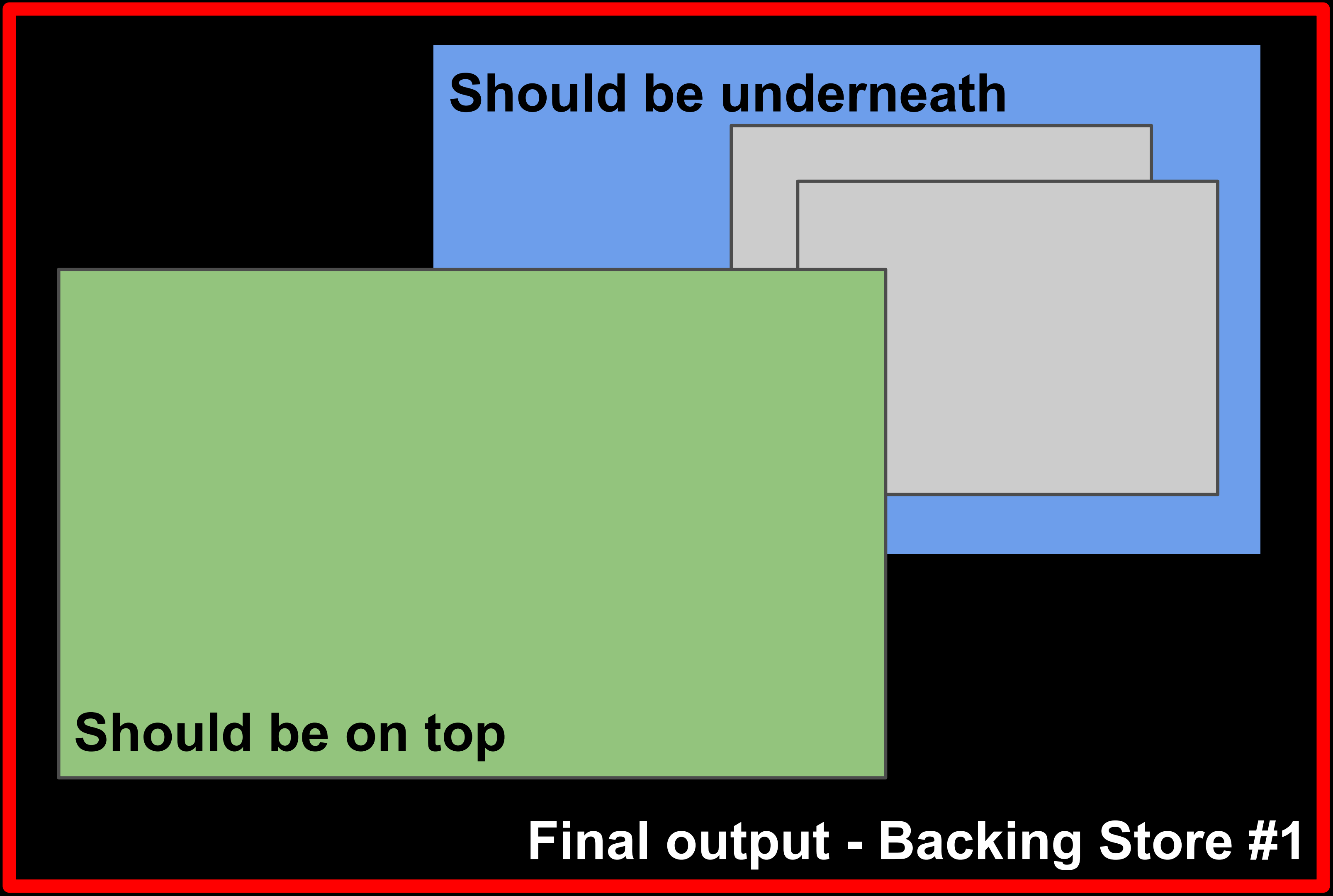


What needs to be repainted as this animates?

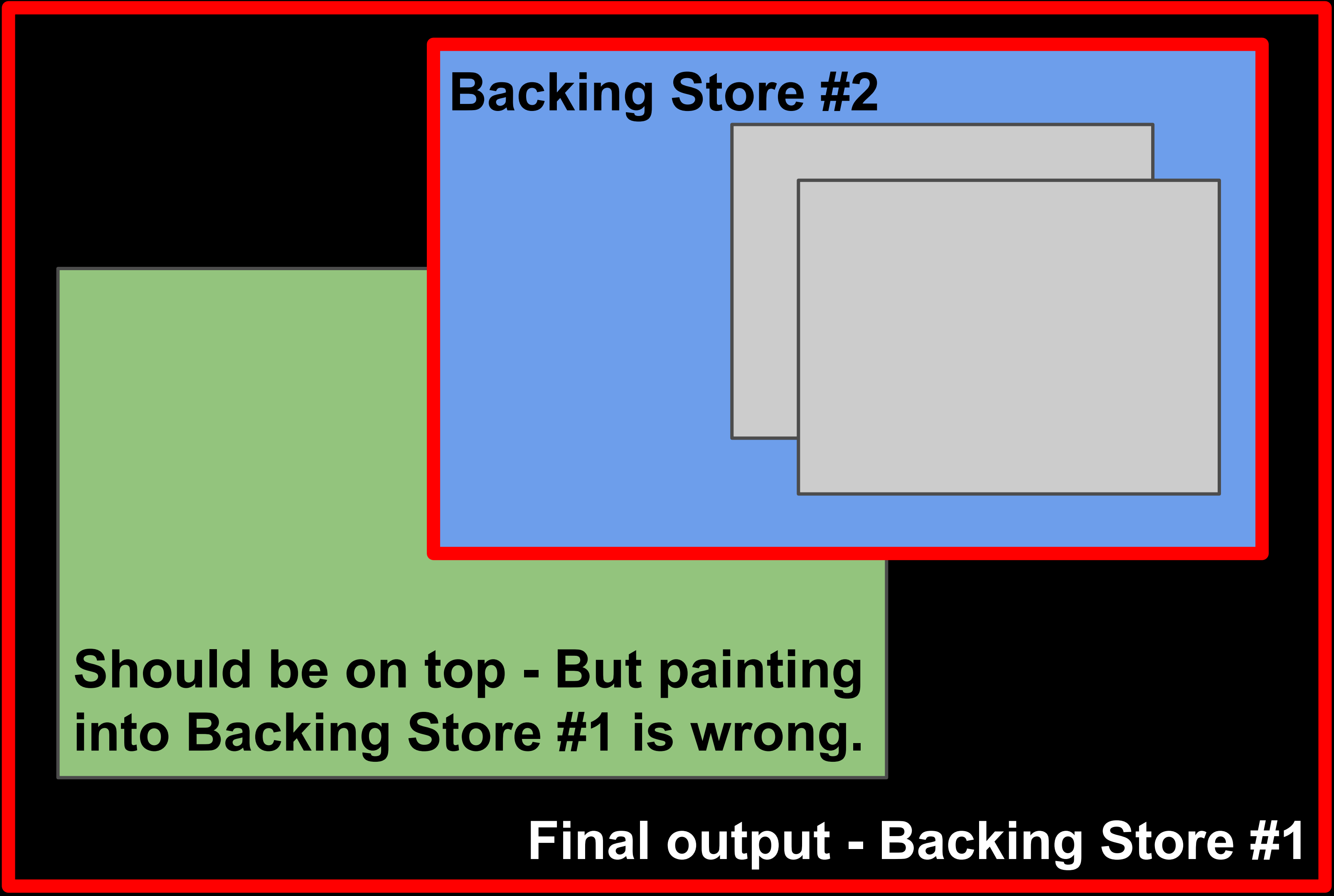
With one backing store (pixel buffer):

Portions of all four layers!

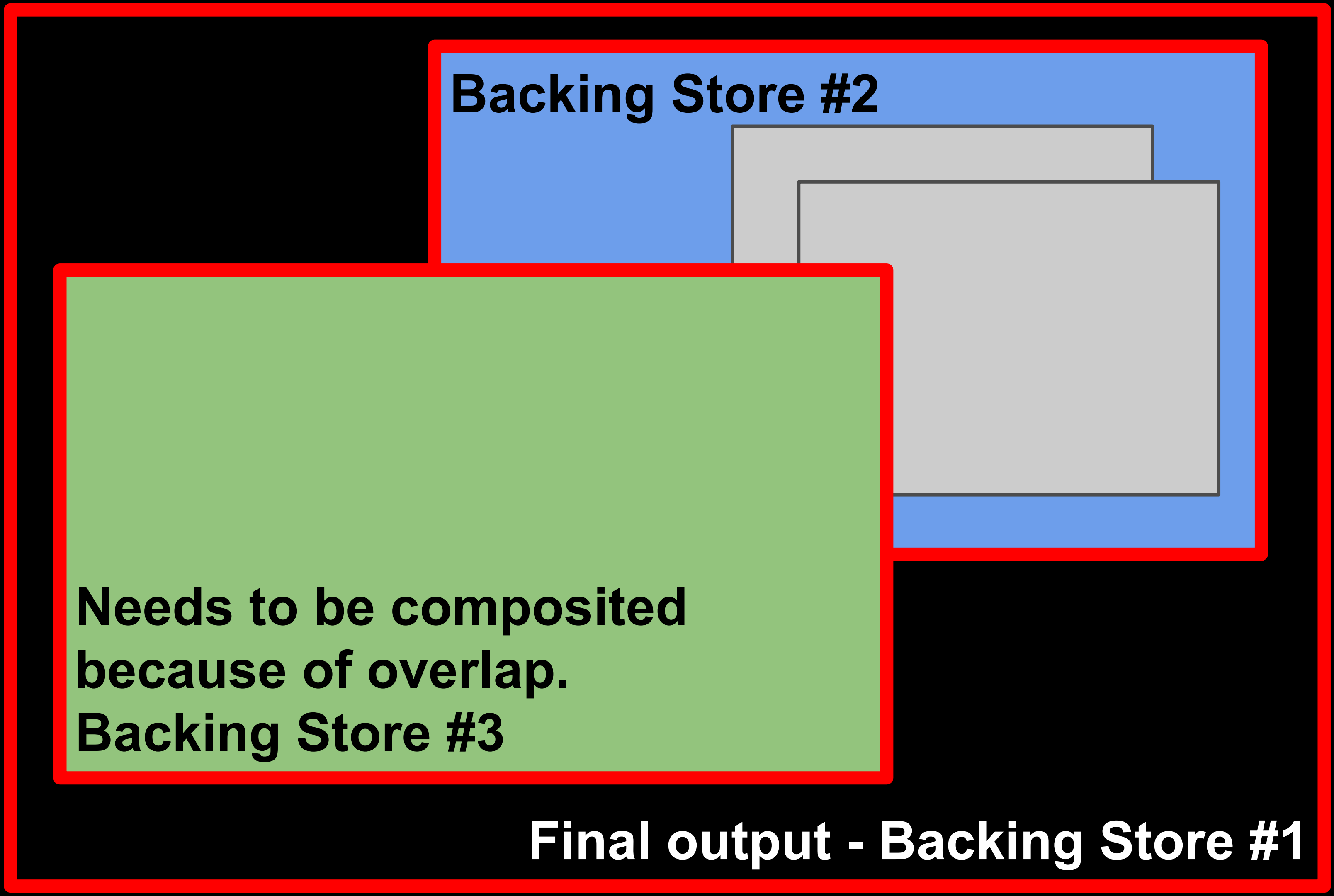
Example #2 - Desired



Example #2 - Wrong!!


Backing Store #2
Should be on top - But painting into Backing Store #1 is wrong.
Final output - Backing Store #1

Example #2 - Correct


Backing Store #2
Needs to be composited because of overlap.
Backing Store #3
Final output - Backing Store #1

Reasons to Make a Composited Layer

Composite when the render subtree could benefit from being cached or grouped:

- Easier to apply certain effects to a subtree
  - e.g. opacity, transforms, filters, reflections
- Elements can move without repainting
  - e.g. scrolling, fixed-position elements
- More practical for hardware accelerated content
  - e.g. video, webGL
- Potentially isolate content that repaints a lot
  - Just speculation at this point

Reasons to Make a Composited Layer

Composite when it is necessary to maintain correctness:

- To maintain correct paint order
    - Like the previous example just shown
- To ensure style properties correctly propagate to the composited layer tree
    - Discussion for another time

Two Fundamental Constraints

Must maintain correct paint order

- Currently, this is done by ***overlap testing***

Only one RenderLayer subtree can contribute to a composited GraphicsLayer

- This is an ***artificial constraint*** imposed by current code design

Silk? more like Burlap!

Overlap testing:

- Innocent intentions: save memory by not creating unnecessary composited layers.
- O(n^2) implementation
- Requires costly converting bounds for most RenderLayers
- Makes compositing reasons depend on layout
- Makes incremental updates impractical
  - Because compositing changes require more overlap testing
- Can incur jank as elements flip between compositing and non-composited

Silk? more like Burlap!

One RenderLayer subtree per GraphicsLayer:

- Potentially creates unnecessary composited layers if things overlap
- Definitely creates unnecessary composited layers for universal composited scrolling

Major computations in compositing

1. Determine reasons we might want to composite each RenderLayer

2. Assign each RenderLayer to a composited backing

3. Create the composited GraphicsLayer tree
   a. allocate/destroy GraphicsLayers as needed
   b. stitch together the layers into a tree
   c. initialize all the properties of the tree (e.g. position, size, opacity, transform, backface-visibility, etc.)

Next-gen Compositing: Squashing

- Step 1: Support for mutiple RenderLayers squashed onto a single GraphicsLayer
    - At least we can get universal overflow scroll nicely supported without layer explosions everywhere
    - Remove other layer explosions problems

- Step 2: Maintain paint order without overlap testing
    - ETA 1-2 more weeks for data making a solid case for this

Next-gen Compositing: Squashing

Long-term possible vision:

- Totally flexible assignment of RenderLayer paint phases to GraphicsLayers

- enables us to be more agile about experimenting with compositing strategies