Blink property trees

Philip Rogers / pdr@google.com

Blink property trees

Recap: compositor property trees

Motivation for blink property trees

How blink property trees are built & updated

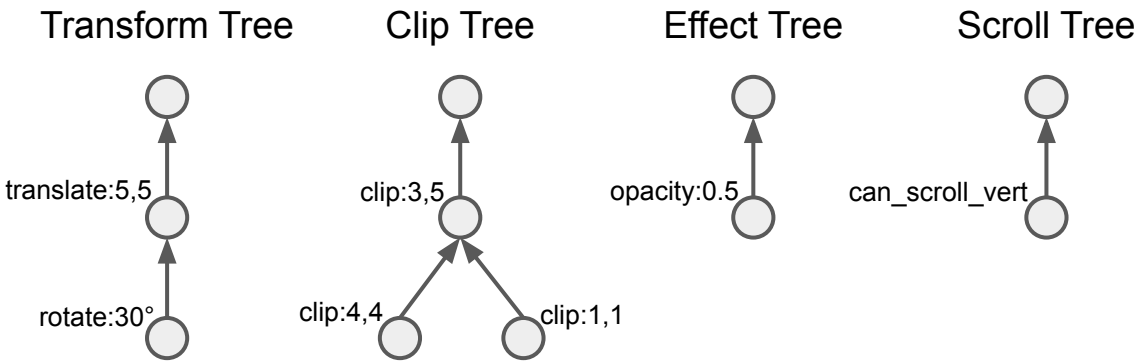
Uses of property trees in blink

Blink and compositor tree differences

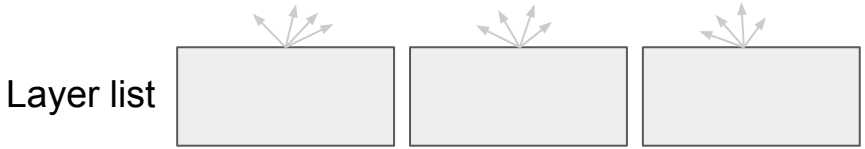
What are compositor property trees?

Sparse data structure for:

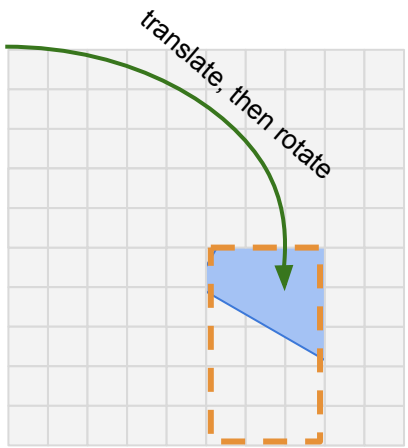
- Compositor operations
- Layer relationships

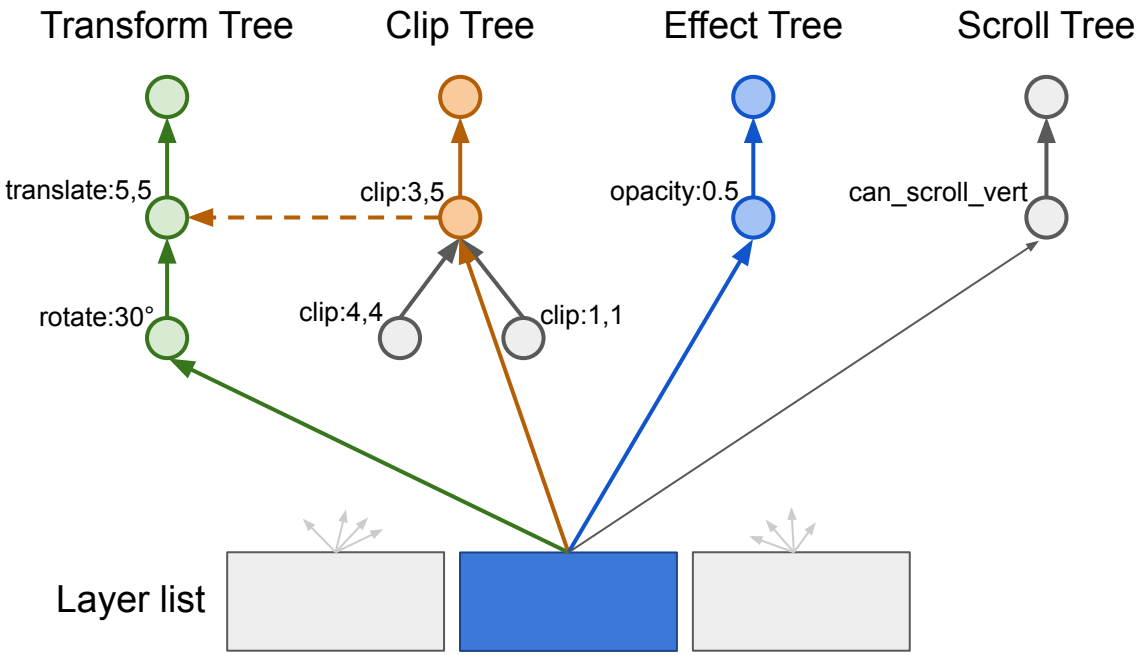


See Ali Juma's talk:
https://goo.gl/oufSqj

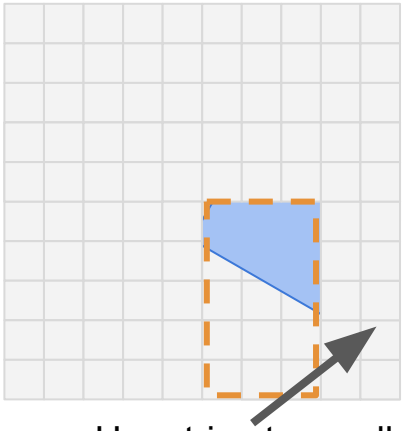


Drawing with compositor property trees



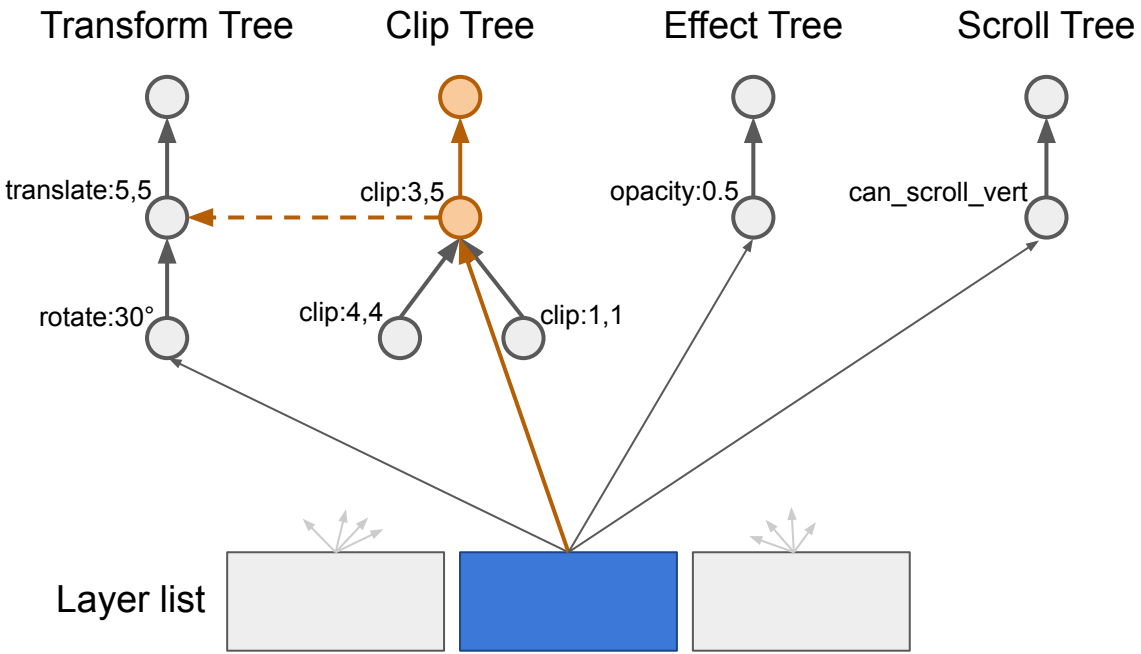


Hit testing with compositor property trees



User tries to scroll

Hit test is clipped out



Blink property trees

Recap: compositor property trees

Motivation for blink property trees

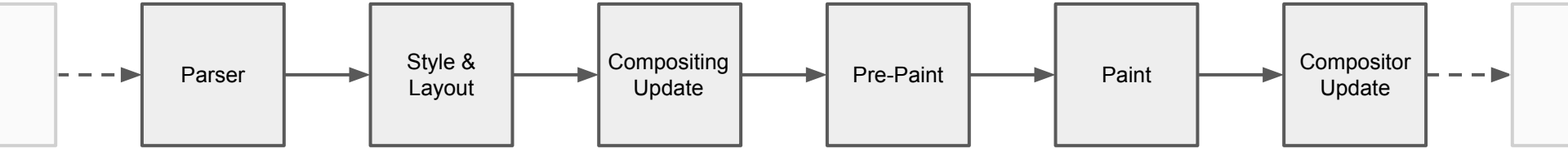
How blink property trees are built & updated

Uses of property trees in blink

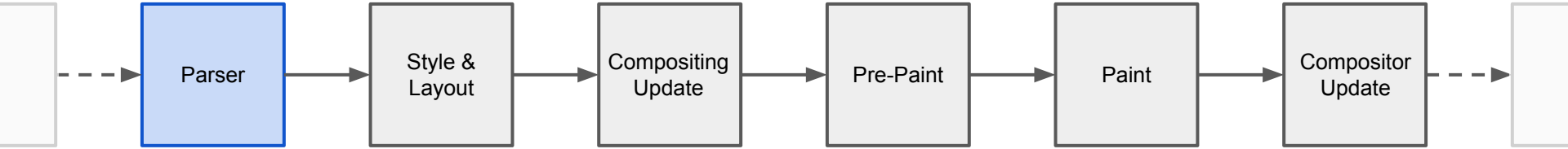
Blink and compositor tree differences

SPInvalidation paint system

- ~~M58~~ M59
- Staging towards SPV2

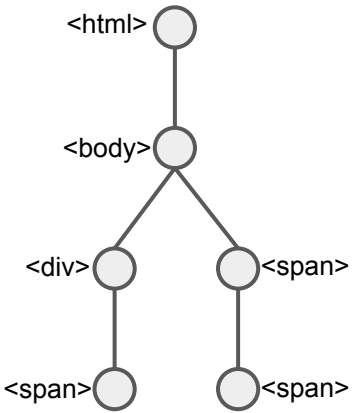


SPInvalidation paint system



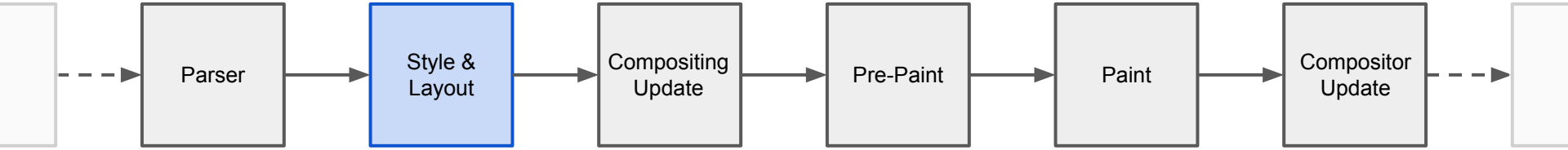

```
<html>
  <body>
    <div>
      <span/>
    </div>
    <span>
      <span/>
    </span>
  </body>
</html>
```

DOM Tree

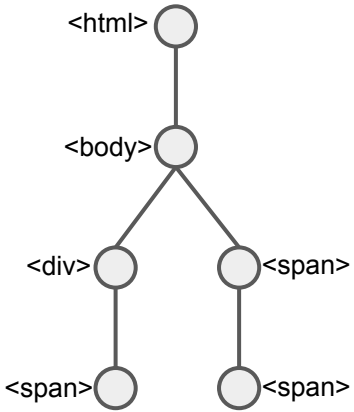


Parsing takes HTML input, produces a DOM tree output.

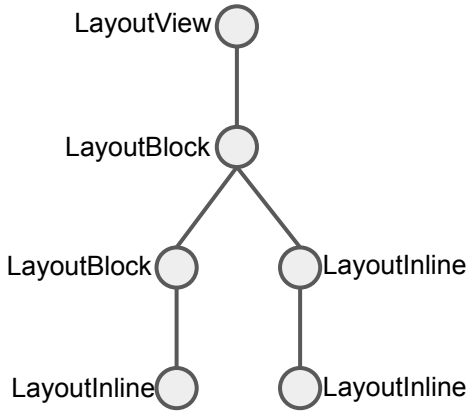
SPInvalidation paint system



DOM Tree

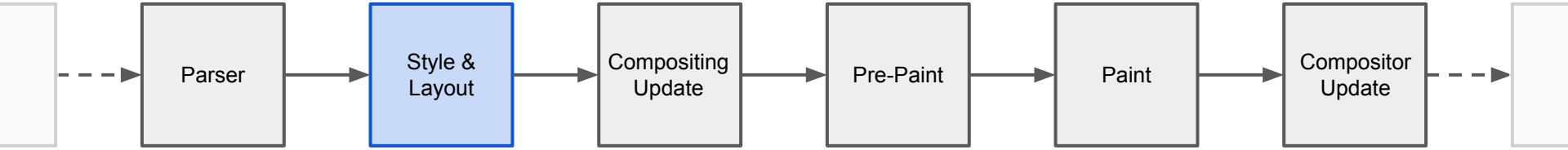


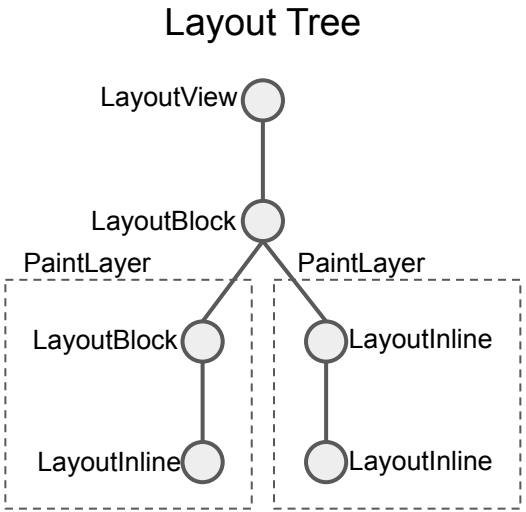
Layout Tree



Style & Layout takes DOM input, produces layout tree output.

SPInvalidation paint system

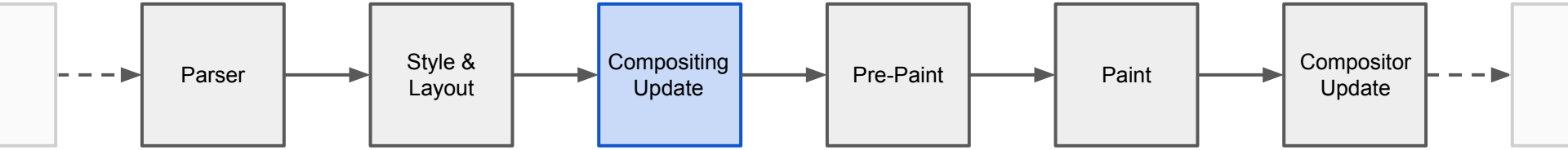


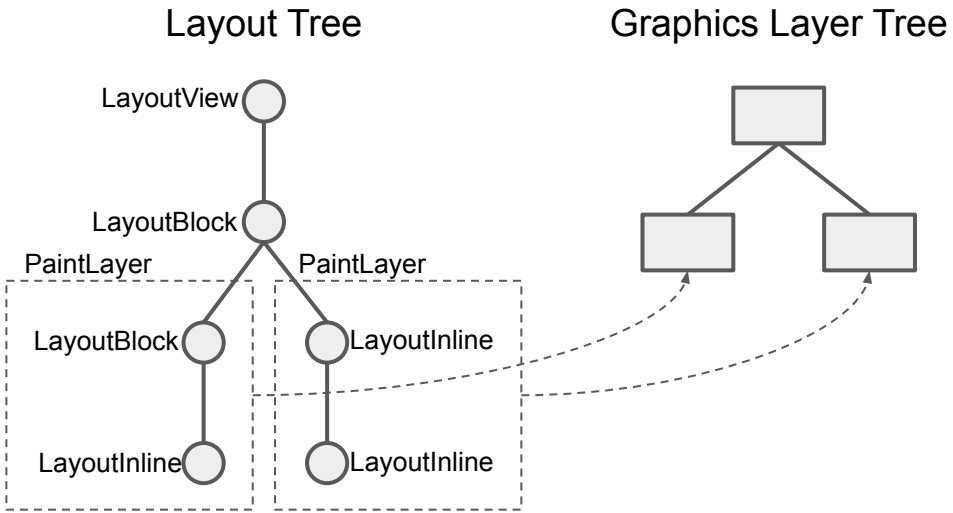


Positioning information is stored in the layout tree. Lets author control where content appears.

Stacking contexts are also stored in the layout tree. Lets author paint subtrees out of order.

SPInvalidation paint system

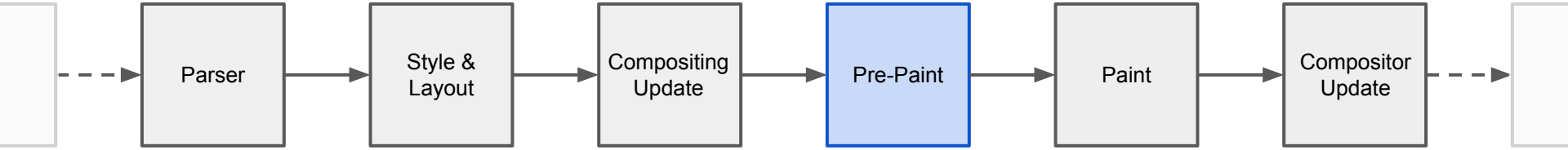




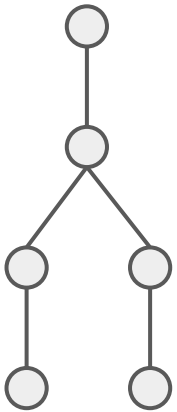
Graphics layers are created for a subset of PaintLayers.

This is a fundamental compositing bug.

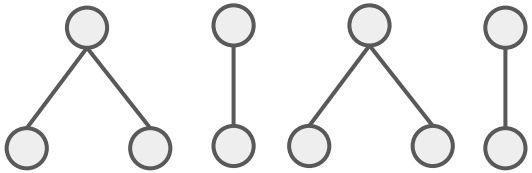
SPInvalidation paint system



Layout Tree

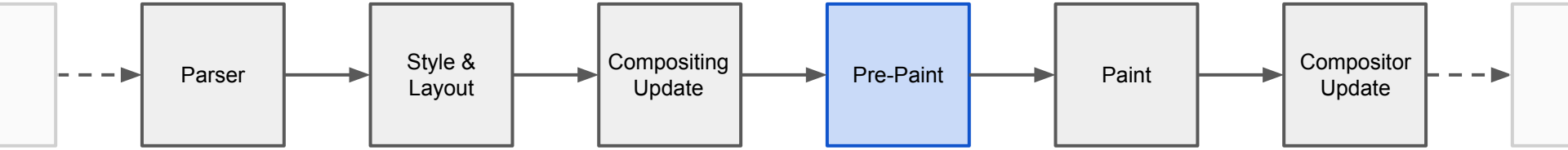


Property trees

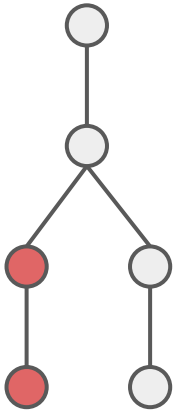


1. Builds property trees from layout tree.

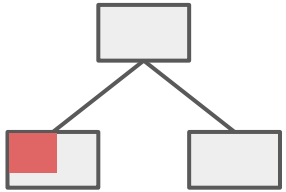
SPInvalidation paint system



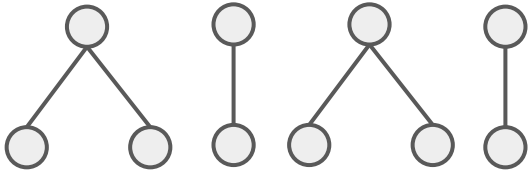
Layout Tree



Graphics Layer Tree

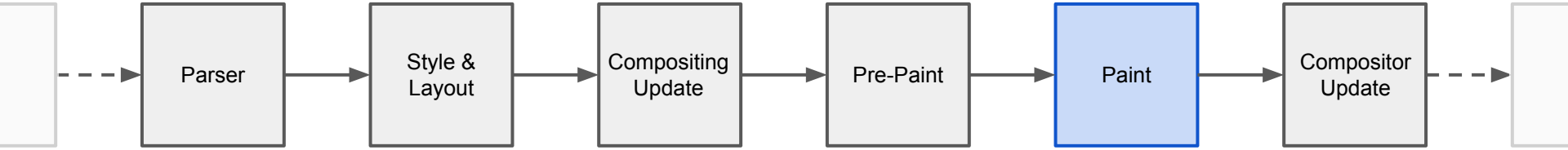


Property trees (4)

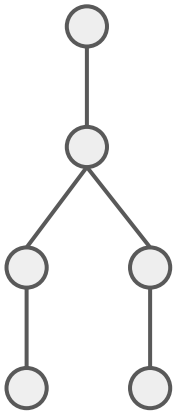


1. Builds property trees from layout tree.
2. Invalidates areas on graphics layers.

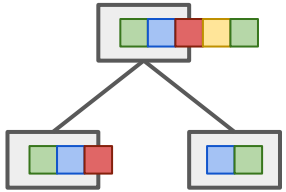
SPInvalidation paint system



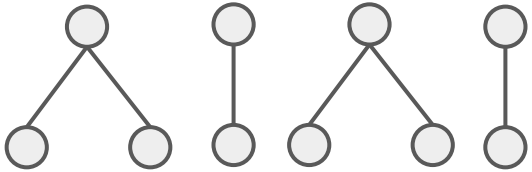
Layout Tree



Graphics Layer Tree



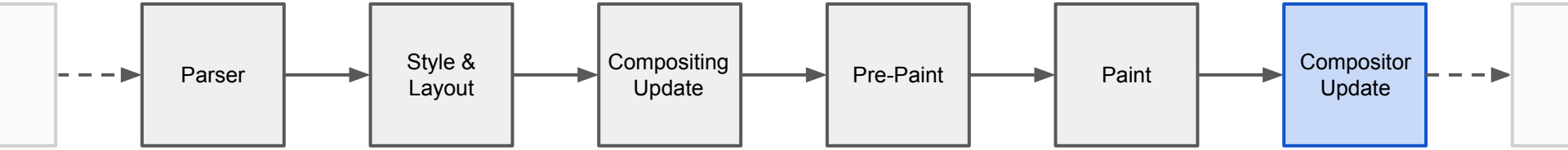
Property trees (4)



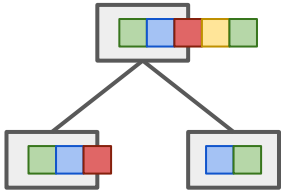
Paint walks the graphics layers and builds a display item list for each layer from the layout tree.

Property trees are used for clip calculations and cull rects.

SPInvalidation paint system

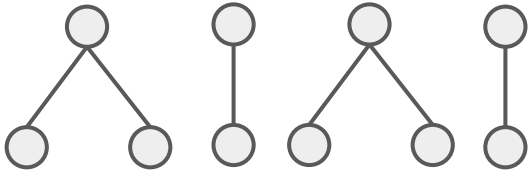


Graphics Layer Tree



Layer list

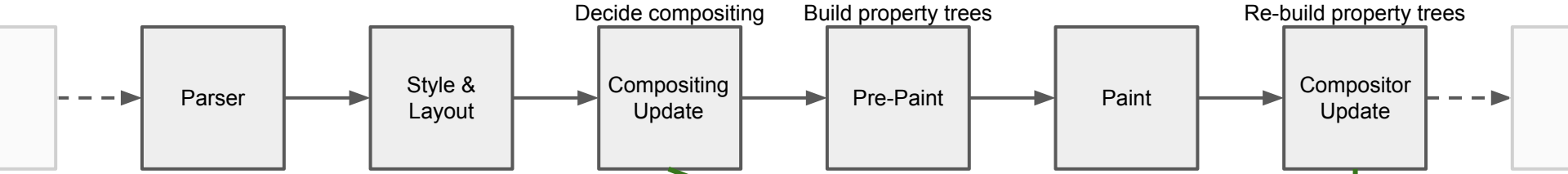
cc property trees



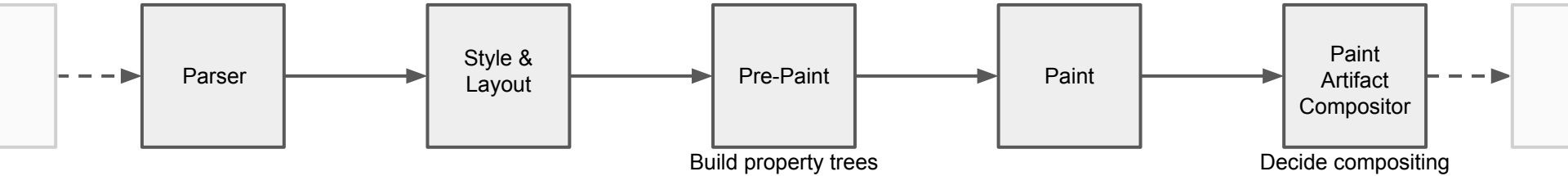
Compositor reverse engineers property trees from graphics layer tree.

Design is a staging step towards SPV2.

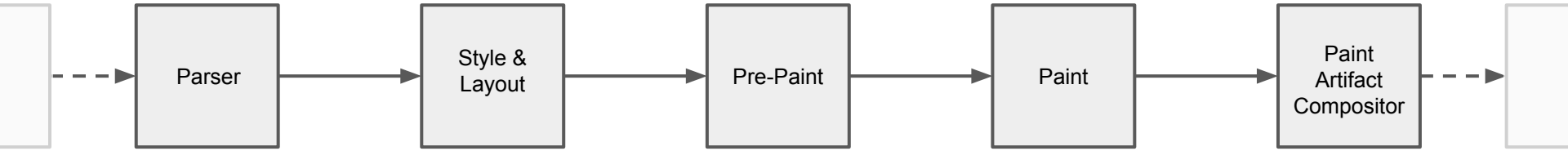
SPInvalidation



SPV2



SPV2 paint system



- As in SPInvalidation, paint properties computed in Pre-Paint step.
- New:
  - Paint step breaks up display item list when paint properties change
    - Each paint "chunk" is a potential compositing atom
  - Paint artifact compositor builds cc layer list from paint chunks
  - Paint artifact compositor directly creates cc property trees from blink trees
  - Paint layers can be much simpler, no longer related to compositing.

Blink property trees

Recap: compositor property trees

Motivation for blink property trees

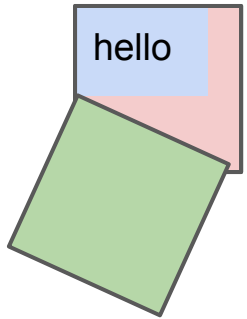
How blink property trees are built & updated

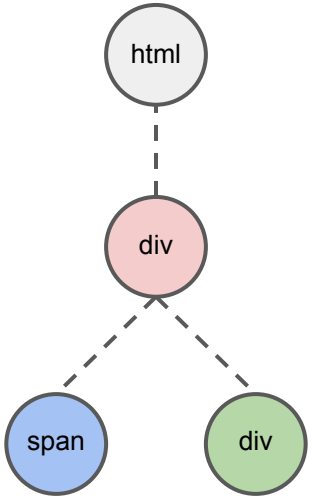
Uses of property trees in blink

Blink and compositor tree differences

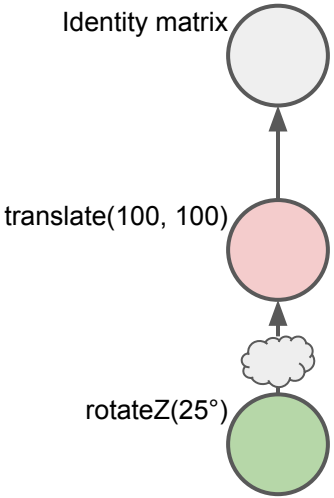
Building blink property trees

```
<div bg: pink; transform: translate(100, 100)>
    <span bg: blue>hello</span>
    <div bg: green; transform: rotateZ(25deg)>
```





Layout tree

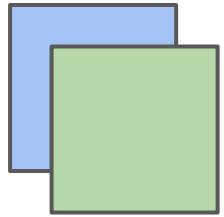


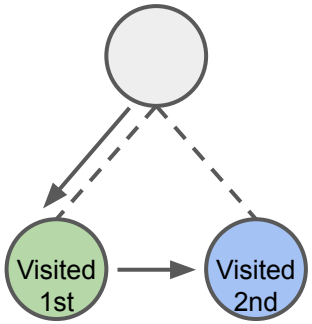
Transform tree

http://jsbin.com/tejimox

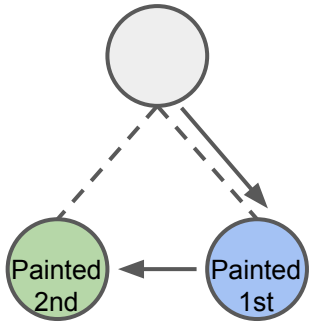
Layout treewalks

```
<html>
    <div bg: green; position: absolute; top: 25px; left: 25px>
    <div bg: blue>
```





DOM or "primary" order

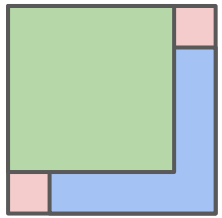


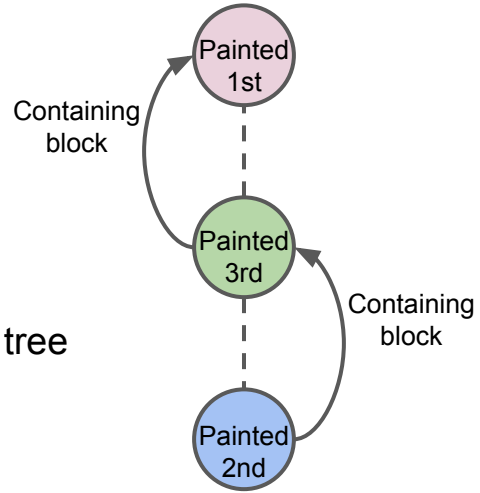
Paint order

http://jsbin.com/napisa

Layout treewalks

```
<div bg: pink; pos: relative; z-index: -1>
    <div bg: green; pos: absolute>
        <div bg: blue; pos: relative; top: 25px; left: 25px; z-index: -1>
```



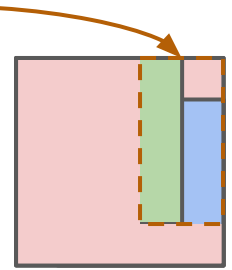


Paint order can jump around DOM tree

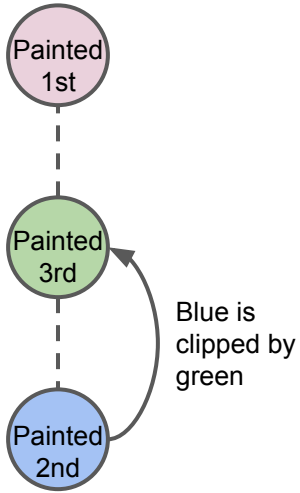
http://jsbin.com/puxuna

Layout treewalks

```
<div bg: pink; pos: relative; z-index: -1>
    <div bg: green; pos: absolute; clip: rect(0, 125px, 100px, 75px);>
        <div bg: blue; pos: relative; top: 25px; left: 25px; z-index: -1>
```



In a paint-order layout tree walk, blue is traversed before its clipping ancestor.



http://jsbin.com/halilez

Building blink property trees

- Built during the pre-paint tree walk
  - A DOM-order layout tree walk for efficiency (not paint-order)
  - All 4 trees constructed simultaneously

- Parent context object passed around during recursive walk
  - Easy access to ancestor properties in O(1)
  - Some additional context needed for out-of-flow (fixed pos, abs pos, etc)

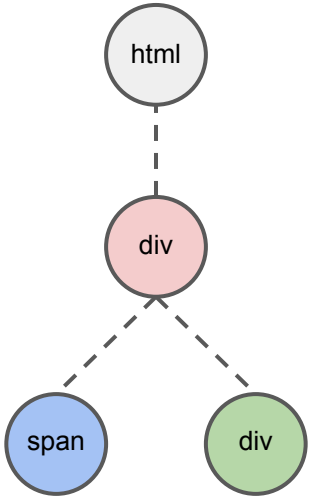
Building the transform and scroll trees

- LayoutObjects can create multiple transforms properties:
  - Paint offset translation - establishes an origin for later transforms
  - CSS transform
  - CSS perspective
  - SVG local to border box - used for SVG's viewbox
  - Scroll translation
    - Scroll tree built from scroll translations

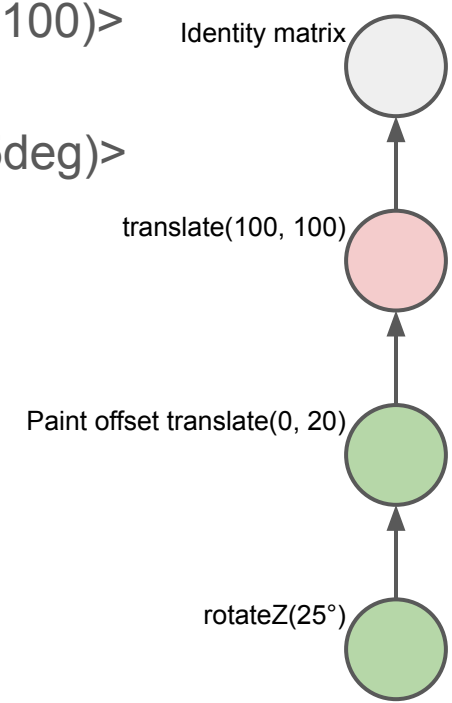
- Stored in ObjectPaintProperties.h
  - Rare-data pattern, stored on LayoutObject
  - How these properties nest is documented in header

Building the transform tree

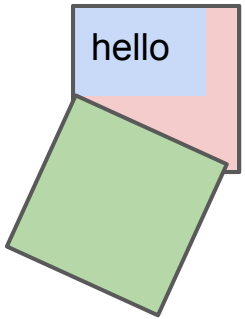
```
<div bg: pink; transform: translate(100, 100)>
    <span bg: blue>hello</span>
    <div bg: green; transform: rotateZ(25deg)>
```



Layout tree



Transform tree



http://jsbin.com/tejimox

Building the effect and clip trees

- LayoutObjects can create multiple effect properties:
  - Main effect - isolated group for CSS opacity, mix-blend-mode, others
  - CSS filter
  - CSS mask

- LayoutObjects can create multiple clip properties:
  - Mask clip - clip for CSS mask
  - CSS clip
  - Inner border radius clip
  - Outer border radius clip
  - Overflow clip - For overflow: scroll and overflow: hidden
  - CSS clip for fixed position objects

Building property trees - memory usage

ObjectPaintProperties: 128 bytes
TransformPaintPropertyNode: 208 bytes
ScrollPaintPropertyNode: 48 bytes
EffectPaintPropertyNode: 104 bytes
ClipPaintPropertyNode: 88 bytes

Amazon.com search for "chromecast":
  5,546 LayoutObjects
  3 transform nodes
  141 clip nodes

Updating property trees

- Clean/dirty bit pattern used
  - Does LayoutObject need own paint properties updated?
    Ex: CSS transform value changed? => setNeedsPaintPropertyUpdate();
  - Does LayoutObject descendant need paint properties updated?

- Optimization: pre-paint tree walk pruned if no updates needed
  - Because some paint properties depend on paint offset, paint offset changes can force updates of subtrees.

- Parent pointer design requires rebuilding subtrees on structural changes
  - If a node is added/removed, a descendant property might still reference it, force update
  - If a node's value just changes, just set the value in-place

Stable ids between cc and blink trees

- 1:1 blink::GraphicsLayer to cc:Layer mapping going away
  - Old: cc layer is scrolled, need to update blink GraphicsLayer
  - New: cc scroll node changes value, which blink node is updated?

- ElementId is the new stable mapping
  - Blink: CompositorElementId, created from a node
  - Cc: ElementId

- Interesting bug: "opacity: 0" subtrees
  - Current layer system creates layers but skips paint steps
  - New ElementId approach requires painting to ensure ElementIds are correct for animations

Blink property trees

Recap: compositor property trees

Motivation for blink property trees

How blink property trees are built & updated

Uses of property trees in blink

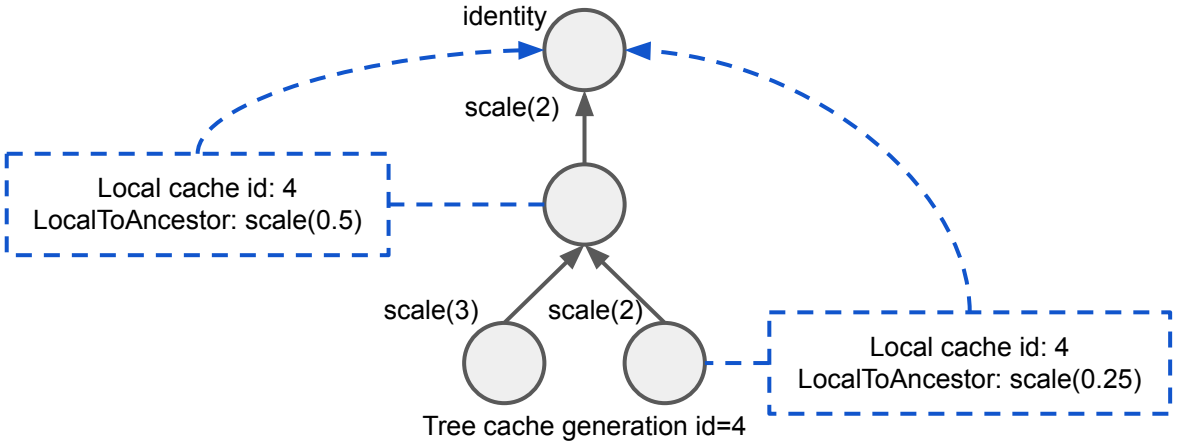
Blink and compositor tree differences

Using blink property trees - GeometryMapper.h

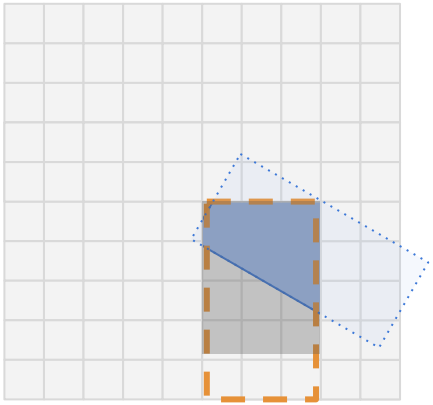
- Common code for mapping between property tree states
- Used for paint invalidation
  - What an object changes, compute the dirty regions
- Used for paint layer clipper
  - What is clip rect applies to an element
- Stores cached results in the property tree itself
  - Uses "generation id" cache invalidation strategy

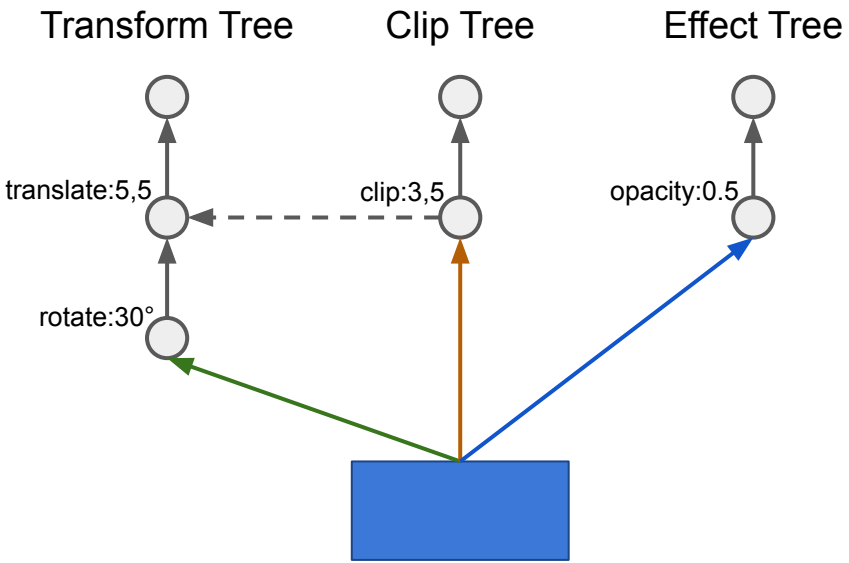
Using blink property trees - GeometryMapper.h

- Invalidating caches using "cache generation id"
  - Cache ids for all cached data
  - Cache id for entire tree
  - Cache is valid iff data id == tree id
  - Lets you invalidate entire tree of cached data in O(1)



Using blink property trees - GeometryMapper.h





localToAncestorVisualRect(localState, rootState, blue rect)

= grey shaded region

Using blink property trees - future

- SPV2
  - Decide layerization and squashing
  - Directly create cc property trees

- Greater usage of GeometryMapper:
  - Hit testing
  - Accessibility APIs
  - Javascript APIs:
    - IntersectionObserver: is an element visible or not
    - getBoundingClientRect: returns the size of an element

Blink property trees

Recap: compositor property trees

Motivation for blink property trees

How blink property trees are built & updated

Uses of property trees in blink

Blink and compositor tree differences

Differences between cc and blink trees

- Ultimate goal: same tree representation used by both.

- High-level differences between the trees:
  - Ownership:
    - cc: PropertyTree<NodeType> which stores a vector NodeType
    - blink: LayoutObjects own nodes through m_paintProperties.
  - Lookup:
    - cc: ints for lookup and references (e.g., int parent_id)
    - blink: pointers for lookup and references (e.g., PropertyNode* parent())
  - Types:
    - cc: ui/gfx types
    - blink: platform/geometry types

Differences between cc and blink trees

- Ultimate goal: same tree representation used by both.

- Transform tree major differences:
  - cc: 3 transforms per node (pre_local, local, post_local)
  - blink: 1 transform per node, plus a transform origin offset
  - resolution: cc will likely move more towards blink's model

  - cc: scroll offset stored in transform node
  - blink: scroll offset transform stored as separate transform node

  - cc: bit for whether a transform node is scrollable
  - blink: transform node owns scroll node

Differences between cc and blink trees

- Ultimate goal: same tree representation used by both.

- Clip tree major differences:
  - cc: has expansion clip type for calculating visible rects
  - blink: has a bug due to not expanding clip
  - resolution: blink will switch to cc's model

  - cc: only rects
  - blink: rect + rounded corners
  - resolution: blink will handle converting these to masks during property tree building

Differences between cc and blink trees

- Ultimate goal: same tree representation used by both.

- Effect tree major differences:
  - cc: implemented
  - blink: work in progress

  - cc: uses filter origin
  - blink: uses local transform node

  - cc: has some surface rect knowledge
  - blink: no surface rect knowledge yet

  - cc: have mask layers
  - blink: no mask layers, use mixed blend mode instead
  - resolution: cc can be refactored to match blink's model

Differences between cc and blink trees

- Ultimate goal: same tree representation used by both.

- Scroll tree major differences:
  - cc: scroll tree freestanding
  - blink: scroll tree inside transform tree

  - No major differences in the nodes % work to remove cc node->Layer pointer.

Blink property trees

Recap: compositor property trees

Motivation for blink property trees

How blink property trees are built & updated

Uses of property trees in blink

Blink and compositor tree differences

Useful links

CSS spec about painting order:
https://www.w3.org/TR/CSS22/zindex.html#painting-order

Ali's talk about compositor property trees:
https://docs.google.com/presentation/d/1V7gCqKR-edNdRDv0bDnJa_uEs6iARAU2h5WhgxHyejQ/edit#slide=id.p

Slimming paint invalidation design doc:
https://docs.google.com/document/d/1M669yu7nsF9Wrkm7nQFi3Pp2r-QmCMqm4K7fPPo-doA/edit

Incremental property tree updates design doc:
https://docs.google.com/document/d/1_GkBfvameyhnLV7ODIRsOoTedQEG5liAcHwAxmwS-Vk/edit

Tien-ren's talk about treewalk orders:
https://docs.google.com/presentation/d/12_0F64xUgp0MwxEalrx0XkNuJ5Lnw3T6GIzNWOWRyoE/edit#slide=id.p

Incremental property tree updates design doc:
https://docs.google.com/document/d/1_GkBfvameyhnLV7ODIRsOoTedQEG5liAcHwAxmwS-Vk/edit

Debugging - pretty printing

```
#include "core/paint/PaintPropertyTreePrinter.h"

void FrameView::PrePaint() {
  ...
  showAllPropertyTrees(*this);
}
```

```
root ... transform=identity origin=0,0,0
  PreTranslation (FrameView) ... transform=identity origin=0,0,0
    PaintOffsetTranslation (LayoutBlockFlow (positioned) DIV id='green')
      Transform (LayoutBlockFlow (positioned) DIV id='world') ... transform=translation(2,4,0)
      Transform (LayoutBlockFlow (positioned) DIV id='hello') ... transform=translation(1,3,0)
```

Debugging - pretty printing

```
if (layoutObj.paintProperties() && layoutObj.paintProperties()->transform()) {
  fprintf(stderr, "%s transform property:\n", layoutObj.debugName().utf8().data());
  fprintf(stderr, "%s", layoutObj.paintProperties()->transform()->toTreeString().utf8().data());
}
```

```
LayoutBlockFlow (positioned) DIV id='banana' transform property:
root transform=identity origin=0,0,0
  PreTranslation (FrameView) ... transform=identity origin=0,0,0
    PaintOffsetTranslation (LayoutBlockFlow (positioned) DIV id='banana')
      Transform (LayoutBlockFlow (positioned) DIV id='banana') ... transform=translation(2,4,0)
```

Debugging - pretty printing

```
layoutObject()->showLayoutTreeForThis();
```

```
*LayoutView                 #document
  LayoutBlockFlow           HTML
    LayoutBlockFlow         BODY
      LayoutBlockFlow (positioned)    DIV id="green"
        LayoutBlockFlow     DIV id="blue"
```

Debugging - pretty printing

```
showLayerTree(layoutView());
```

```
*layer at (0,0) size 800x600 (composited, bounds=at (0,0) size 800x600, drawsContent=1)
  LayoutView at (0,0) size 800x600
 positive z-order list(1)
   layer at (0,0) size 800x100
    LayoutBlockFlow {HTML} at (0,0) size 800x100
      LayoutBlockFlow {BODY} at (0,0) size 800x100
        LayoutBlockFlow {DIV} at (0,0) size 100x100 [bgcolor=#ADD8E6] id="blue"
   positive z-order list(2)
     layer at (25,25) size 100x100 (composited, bounds=at (0,0) size 100x100, drawsContent=1)
      LayoutBlockFlow (positioned) {DIV} at (25,25) size 100x100 [bgcolor=#90EE90] id="green"
     layer at (50,50) size 100x100 transparent
      LayoutBlockFlow (positioned) {DIV} at (50,50) size 100x100 [bgcolor=#FFC0CB] id="pink"
```

Debugging

- Paint under-invalidation:
  Catch cases where display items change without being marked as invalid:
  --enable-blink-features=PaintUnderInvalidationChecking

- Paint property under-invalidation:
  Catch cases where properties change without needing an update:
  FindPropertiesNeedingUpdate (always on in DCHECK builds)

Code for GeometryMapper example

```
TEST_F(GeometryMapperTest, TestCompositorExample) {
  TransformationMatrix transformMatrixTranslation;
  transformMatrixTranslation.translate(5, 5);
  RefPtr<TransformPaintPropertyNode> transformTranslation =
      TransformPaintPropertyNode::create(rootPropertyTreeState().transform(),
                                         transformMatrixTranslation, FloatPoint3D());
  TransformationMatrix transformMatrixRotation;
  transformMatrixRotation.rotate(30);
  RefPtr<TransformPaintPropertyNode> transformRotation =
      TransformPaintPropertyNode::create(transformTranslation,
                                         transformMatrixRotation, FloatPoint3D(2.25, 1.25, 0));

  FloatRoundedRect clipRect(
      FloatRect(0, 0, 3, 10),
      FloatRoundedRect::Radii(FloatSize(), FloatSize(), FloatSize(),
                              FloatSize()));
  RefPtr<ClipPaintPropertyNode> clip = ClipPaintPropertyNode::create(
      ClipPaintPropertyNode::root(), transformTranslation,
      clipRect);

  PropertyTreeState localState = rootPropertyTreeState();
  localState.setTransform(transformRotation.get());
  localState.setClip(clip.get());

  FloatRect input(0, 0, 5.5, 2.5);

  LOG(INFO) << "input: " << input.toString();
  geometryMapper->localToAncestorVisualRect(localState, rootPropertyTreeState(), input);
  LOG(INFO) << "localToAncestorVisualRect: " << input.toString();
}
```