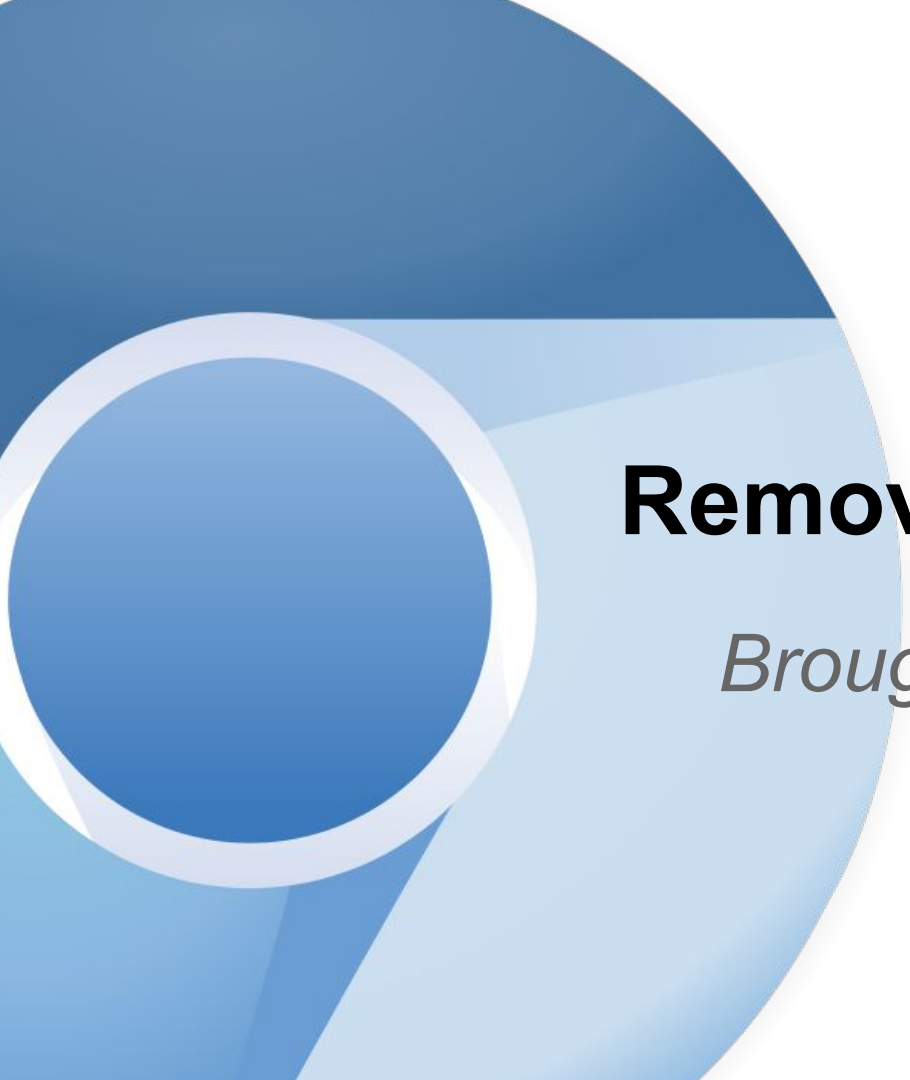


Removing Blink Scrolling

Brought to you by BlinkON!

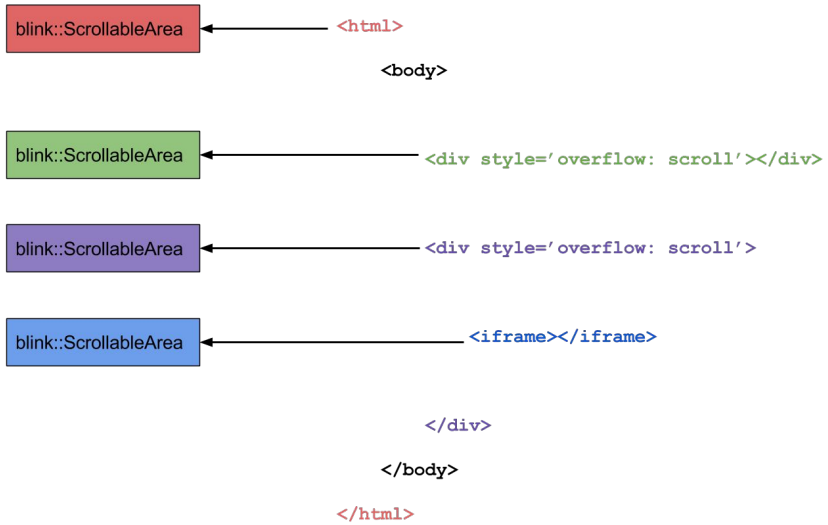
What am I talking about?

- “Scroll Unification”
  - Detailed doc: link
  - Bug
- Code Health - Refactoring to remove duplication
- Clear division of responsibilities
- Remove **USER** scrolling from Blink
  - Blink still knows about scroll
  - JS can still modify scroll offsets

But first: some background on scrolling and input...

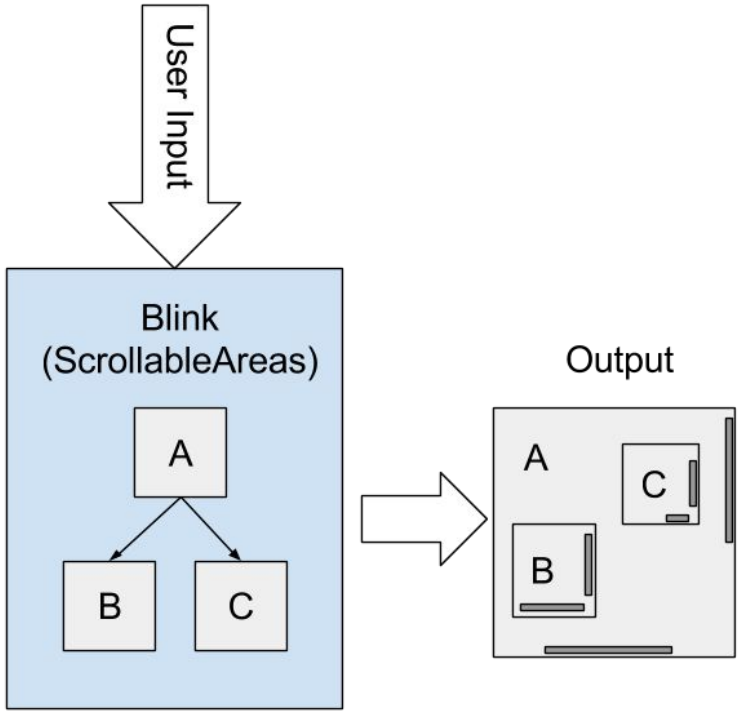
Scrolling - Simplified

- Scrolling Elements, Frames, Controls produce ScrollableArea
- Gesture event (MouseWheel) modifies ScrollableArea
- Blink produces new frame using scroll offsets in ScrollableArea

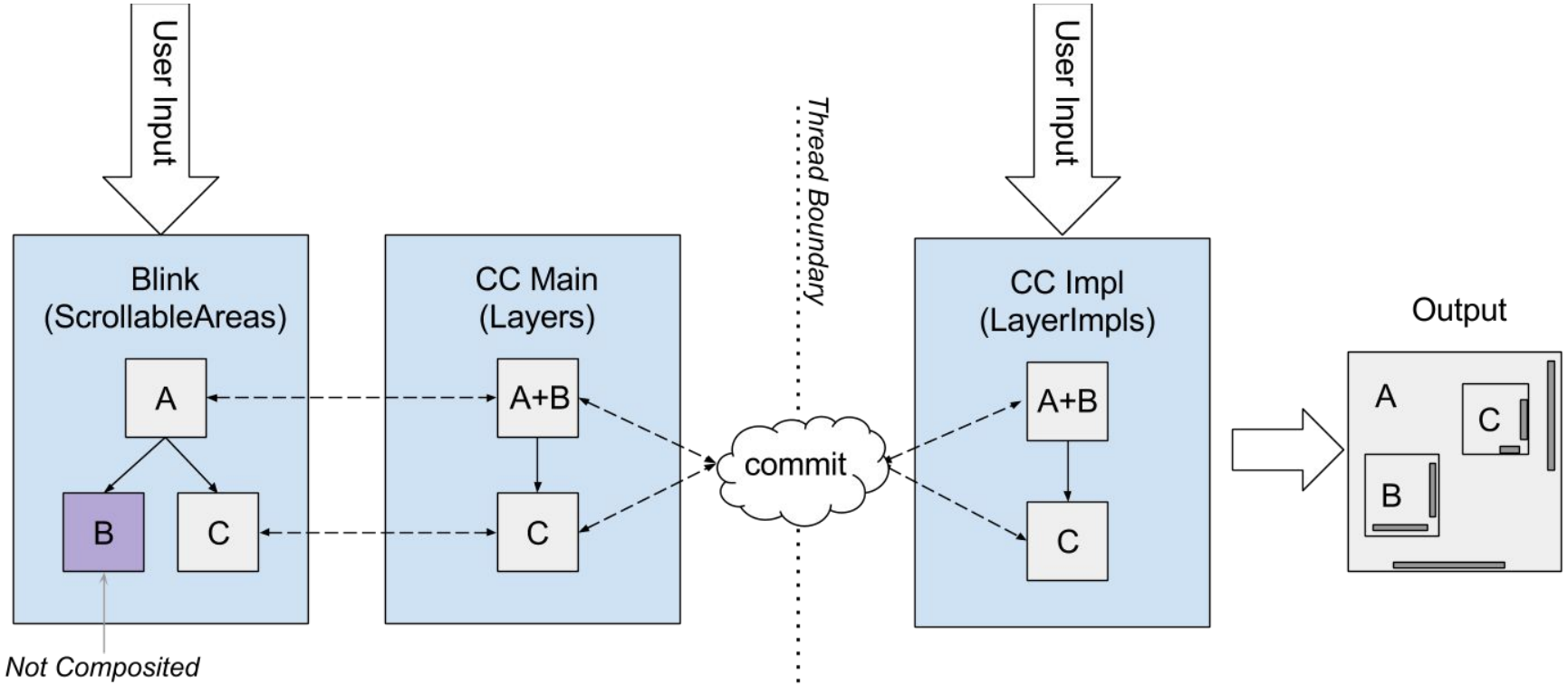


Scrolling - Simplified

- Obvious problem: busy main thread
- “Jank” when Blink/JS is busy

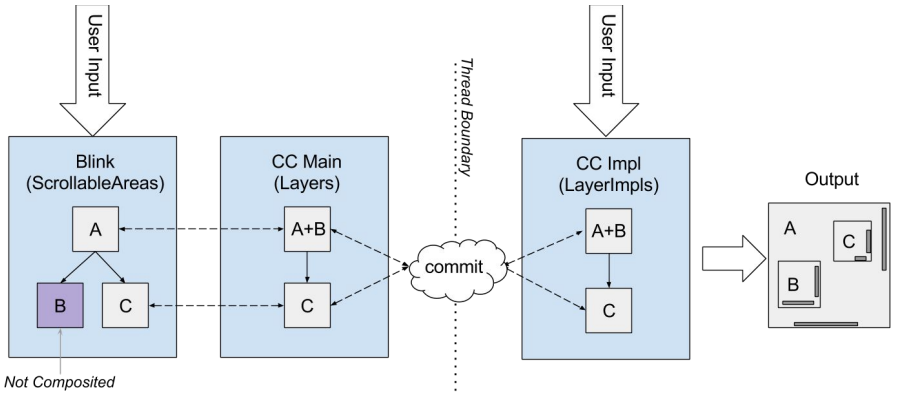


Scrolling - Complicated (and improved)



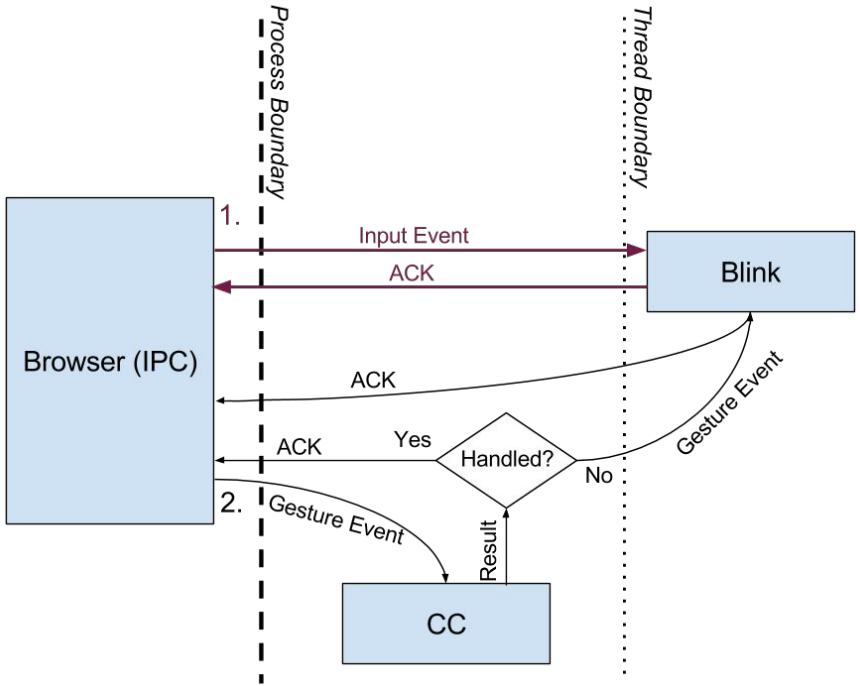
Scrolling - Complicated (and improved)

- Add a new thread (“impl” thread)
- New Chrome component: Chrome Compositor (CC)
- Composite eligible* areas
- “Commit” changes in Blink to the impl thread



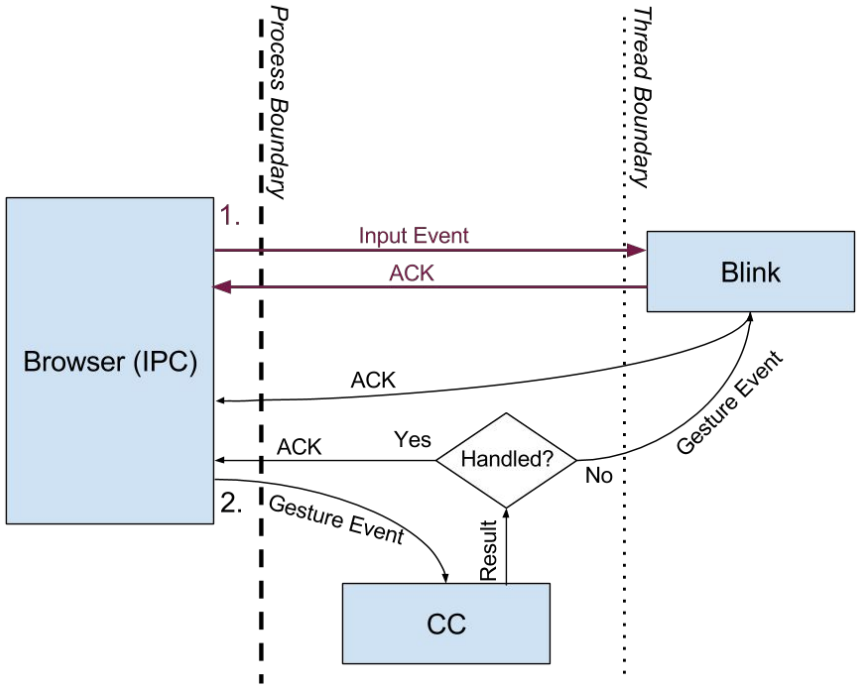
Quick Aside: Events and Gestures

- Events come first (e.g. MouseWheel)
  - Go to Blink only
  - Converted to JS event
- ACK back to Browser
  - Did page call `preventDefault()` ?
  - No? We should scroll, generate a gesture



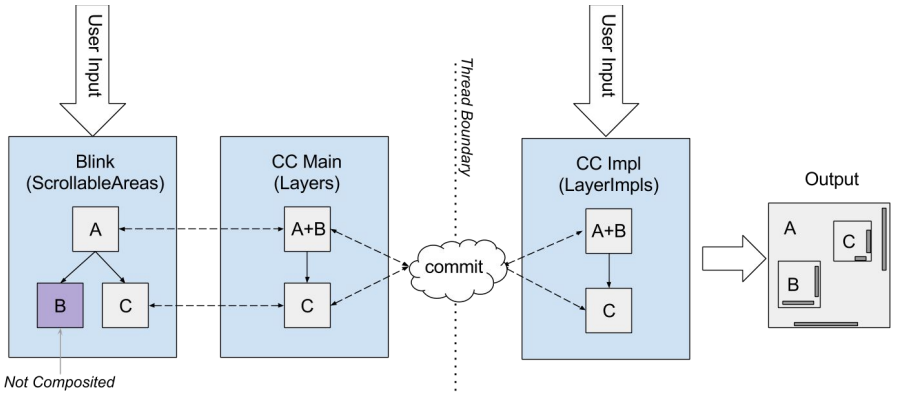
Quick Aside: Events and Gestures

- Gestures cause scrolling
  - e.g. GestureScrollUpdate
- CC gets first crack at GestureEvents
- Sometimes CC can't handle it
  - Crazy transforms, regions without a layer
- ¯_(ツ)_/¯: let Blink deal with it



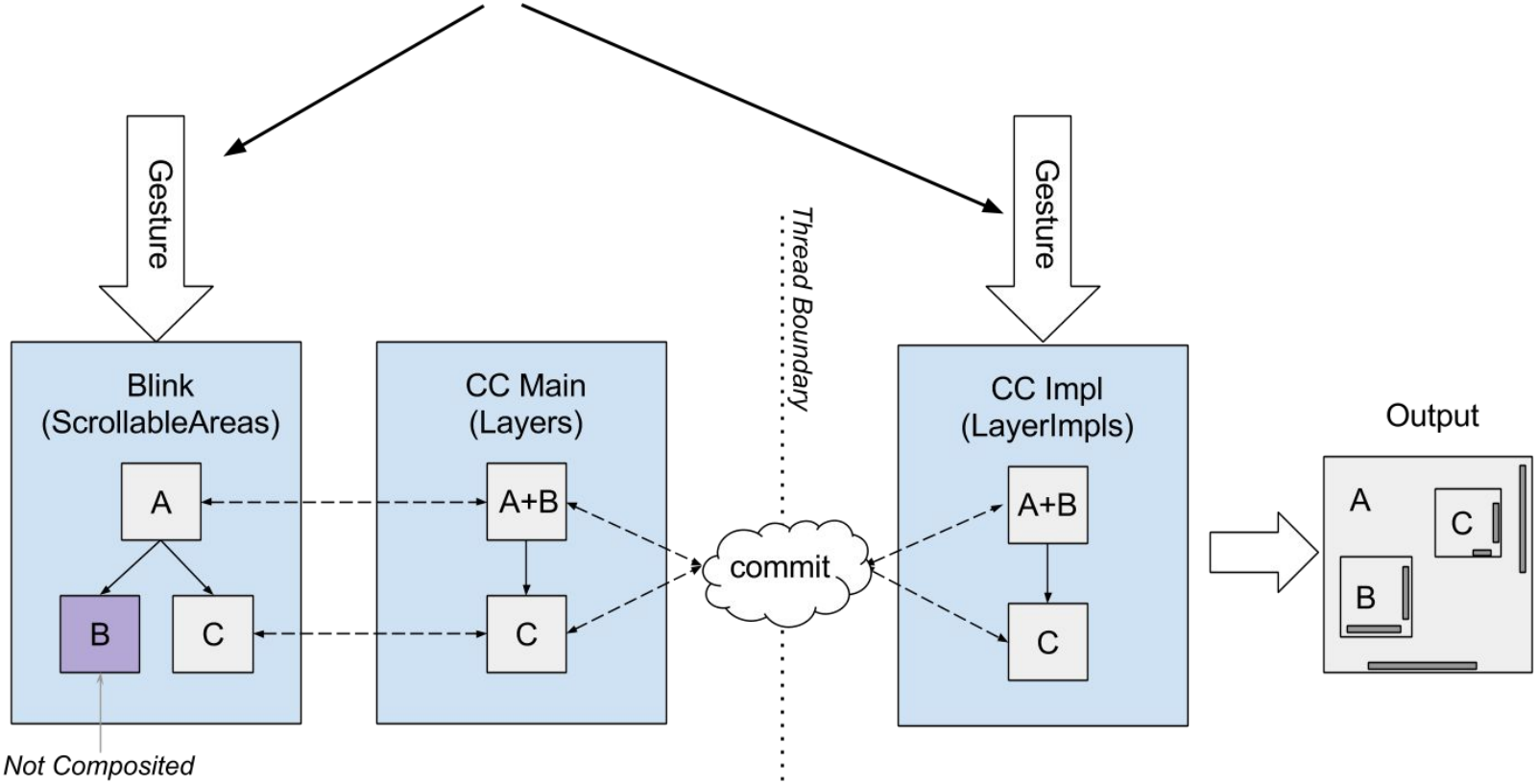
Scrolling - Complicated (and improved)

- Best Case: gesture handled entirely on impl thread
  - No Jank!
- Blink must still handle gestures
- Sometimes we can’t composite
  - CSS border-radius
  - LCD text anti-aliasing



Now that we all know how scrolling works....

This Duplication: PROBLEM



Why is it a problem?

- New feature? 2X the effort
  - Implement it both in Blink and in CC
  - scroll-boundary-behavior, document.rootScroller, scroll latching, etc.
- CC-centric features
  - e.g. Overscroll glow / rubber banding, Android/ChromeOS overlay scrollbars
  - Non-composited scrollers don’t get these features
  - Blocking new work

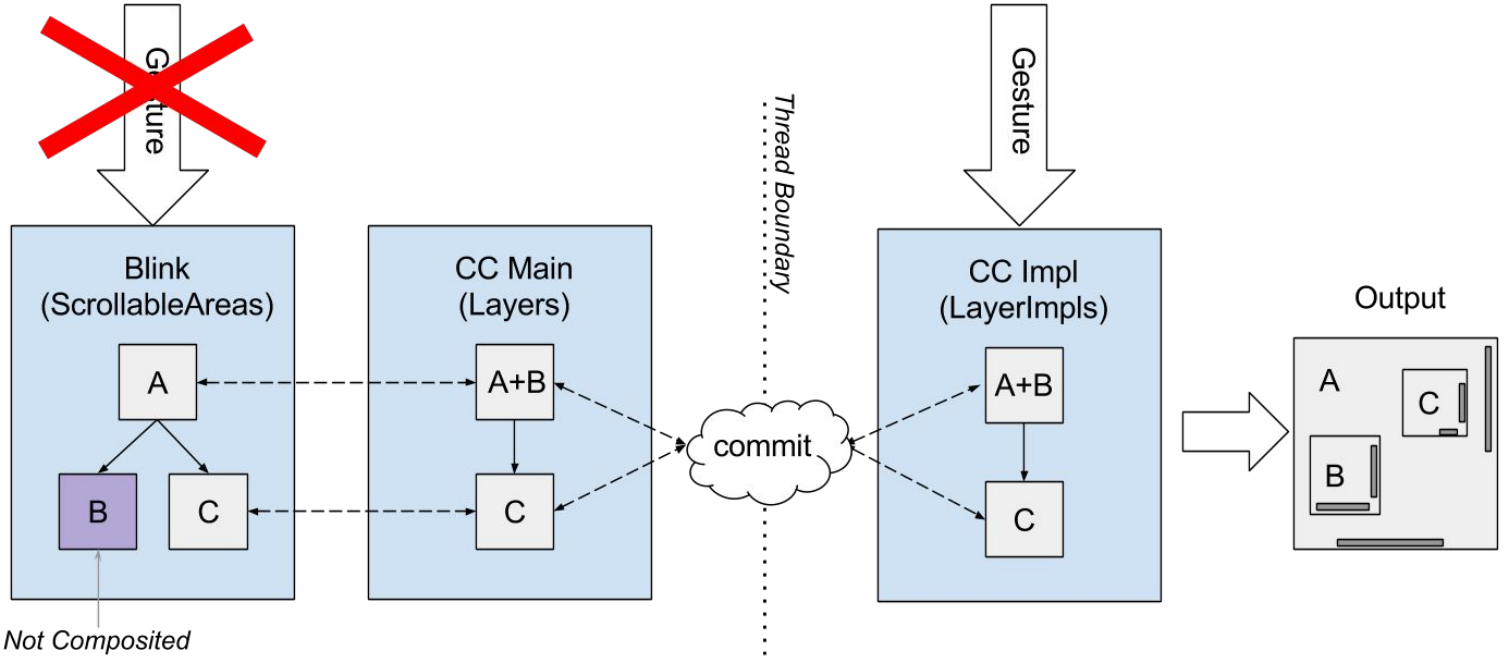
Why is it a problem?

- Implementation Inconsistencies
  - Blink and CC behave differently
  - Debugging is harder (e.g. ever see: “bug occurs only with Retina screen?”)
- Synchronization complexity
  - Scrollers can change from composited → non-composited and back arbitrarily
  - E.g. CC animation synchronization heavy machinery
- Everything is more time consuming
  - How does *X* work? Test both cases...
  - Got a bug? The haystack is bigger

Why is it a problem? (...continued)

- Difficult to test
  - Double the testing surface
  - Traditional tests (Blink unit tests, LayoutTests) completely miss CC
  - CC testing is limited/hard
- Difficult to explain
  - Increased system complexity
  - Blink and CC overlap in responsibility

Solution:


Gesture
Blink
(ScrollableAreas)
A
B
C
Not Composited
CC Main
(Layers)
A+B
C
Thread Boundary
commit
Gesture
CC Impl
(LayerImpls)
A+B
C
Output
A
C
B

Easy, right? Now the hard part...

How? A few problems stand in the way...

Problem 1: Some scrollers aren’t composited

- Remember: LCD text antialiasing, CSS border-radius, etc.
- CC is missing information
- If CC doesn’t know about it, CC can’t scroll it

Solution - Property Trees

- ScrollTree to be precise
- Already in development
- CC can know about (and can mutate) all scrollers
- Independent from DisplayList, compositing decisions

Problem 2: Scrolling isn’t a simple translate

- Sometimes, scrolling isn’t just `translateY(Δ px)`
- This is the reason we don’t composite everything in the first place
- Changing the offset requires modifying pixels
  - LCD text antialiasing requires reraster, blending with pixels below
  - CC layers don’t support complicated clip paths
- CC only knows how to scroll by applying translate

Solution

- No need to duplicate gesture handling code
  - Worst case: we could teach CC that some ScrollNodes need to cause a full BeginMainFrame/Commit cycle
- Performance-wise - no worse than today
  - Send to CC impl → Forward to Blink main, handle gesture → Repaint, Commit
- Above idea
  - Send to CC impl, Handle Gesture → Repaint, Commit

We can do better - Raster-inducing Scrolls

- CC already receives unrastered DisplayLists from Blink
- Today: CC rasters entire “interest rect” from Blink, translates to scroll
- Teach CC to scroll special layers by inducing raster
  - Raster only what’s currently visible
  - On scroll, reraster the new visible area
  - Multiple raster sources → One backing
- Performance: less thread hopping, less repaint - Less jank!
- Prevent excessive checkerboarding (e.g. fast scrollbar dragging)
- Status: threaded-rendering-dev already exploring it

Problem 3: CC can’t hit test everything

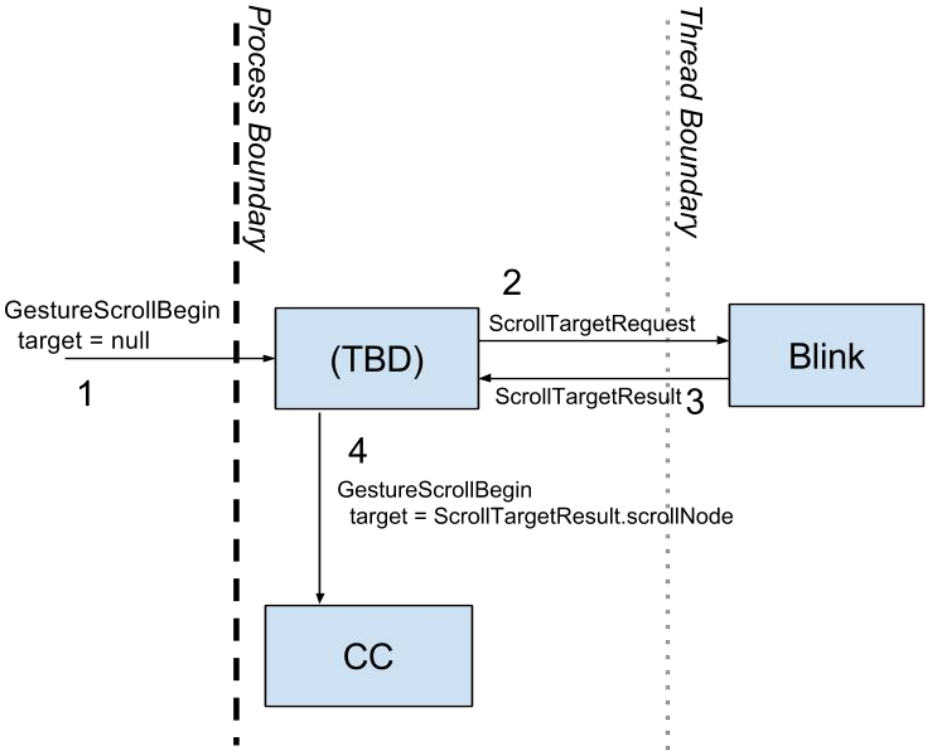
- Don’t know *what* to scroll
- ScrollTree lacks geometry information
- In theory, CC should be able to determine scroller from DisplayLists
- Invisible (but interactive!) items may not be sent by Blink at all

Solution - Blink Hit Testing Service

- Need to know which scroller a GestureEvent should affect
- Only Blink can answer that question comprehensively
- Let CC/Browser ask Blink for the hit testing result
- Surprise! Mus+Ash project also wants/needs this. Already in the works

HTaaS: Hit Testing as a Service

- Gesture first ask Blink for a hit test
- Gestures get to CC with a ScrollNode already populated
- Optimization: Include the result in InputEvent (MouseWheel) ACK



Problem 4: Main-thread only scrolling (Pri-3)

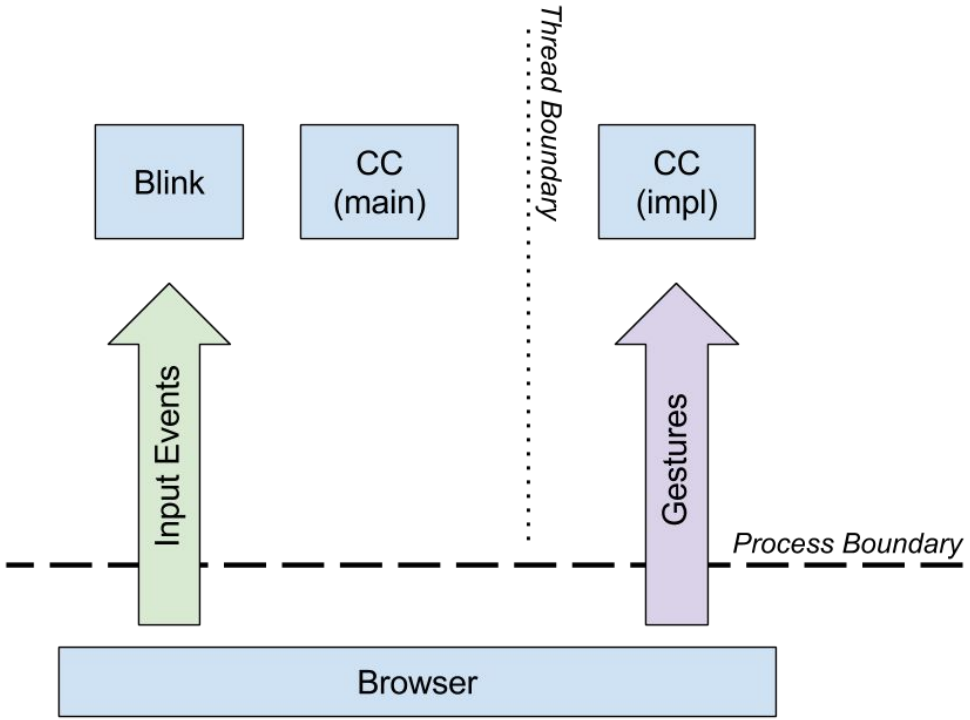
- E.g. Keyboard, scrollbar scrolling.
- Today, implemented in Blink only

Solution

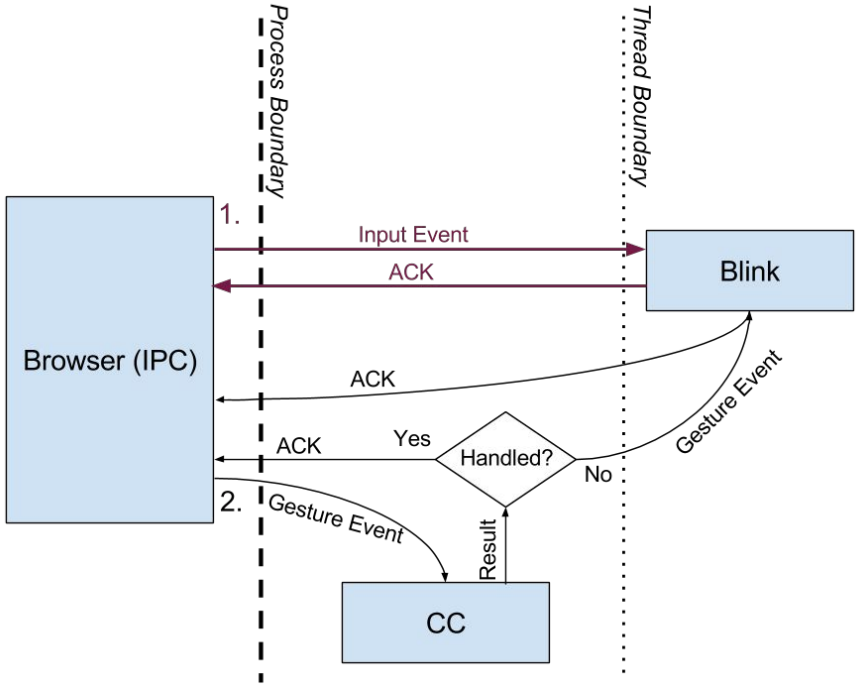
- Simple but tedious
- Move/implement the scroll modes in CC
  - Keyboard gestures
  - Autoscroll: Windows pan scrolling, drag scrolling, game pad, etc.
  - Scrollbars
- Not *strictly* necessary
  - Most benefit will come from other modes that are duplicated
  - Performance win! No jank!
- Maybe delay this
  - Input handling should really be outside CC

How does all this look?
Happy pictures:

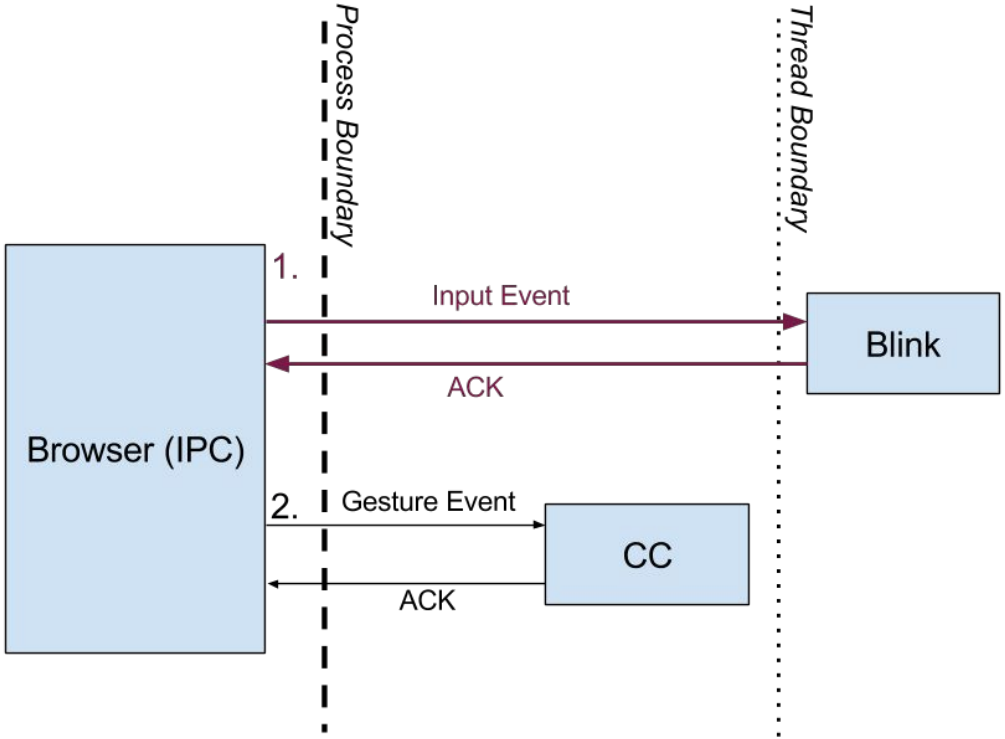
Responsibilities: Clearly delineated



Remember Input Pipeline?



Beautiful


Process Boundary
Thread Boundary
1.
Input Event
Blink
ACK
Browser (IPC)
2.
Gesture Event
CC
ACK

Code be gone!

- All Blink's handleGestureScroll*
- All of blink::ScrollManager (~1000 LOC)
- Big chunk of core/page/scrolling
  - Scroll Customization
  - OverscrollController, ViewportScrollCallback
  - Probably large chunks of ScrollingCoordinator
- Blink side of scrolling features like scroll-boundary-behavior

Questions and Comments?