Folly of Scheduling

(“It seemed like a good idea at the time...”)

skyostil@ / alexclarke@

BlinkOn 6

Scheduling one-pager

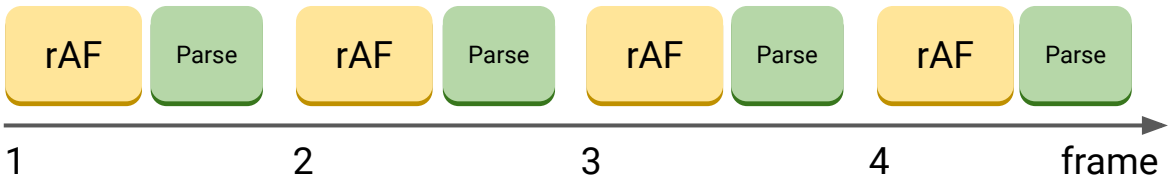
- Cooperative, single thread scheduling
- Tasks generally aren’t interruptible
- Unpredictable task run times
  - 99th %ile run time < 1ms, but tail extends to several seconds
- No explicit notion of priorities, but implicit ordering constraints
- Scheduler’s job: pick the most appropriate task to run next
- Policies based on
  - Task type
  - User model
  - Performance metrics

Bad idea #1: Always prioritize rendering

- Intuitively should increase animation smoothness

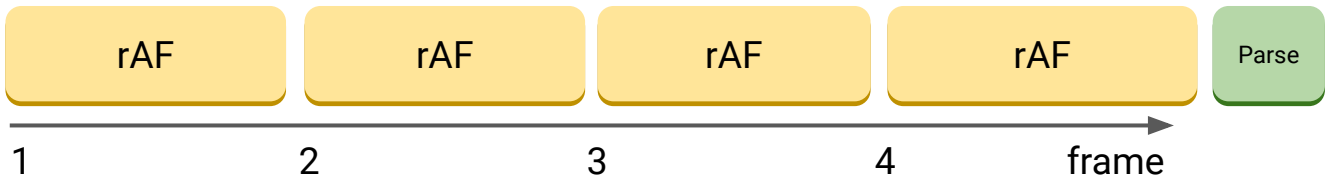
Bad idea #1: Always prioritize rendering

- Intuitively should increase animation smoothness
- Problem: if rendering starts taking >16ms, you'll be rendering **all the time**



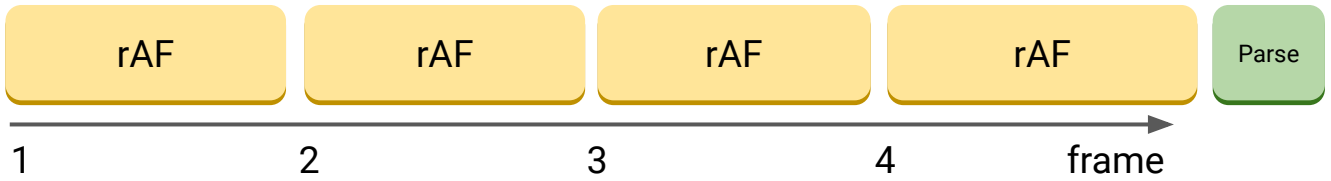
Bad idea #1: Always prioritize rendering

- Intuitively should increase animation smoothness
- Problem: if rendering starts taking >16ms, you'll be rendering **all the time**



Bad idea #1: Always prioritize rendering

- Intuitively should increase animation smoothness
- Problem: if rendering starts taking >16ms, you'll be rendering **all the time**
- Regressed page load performance
- Anti-starvation
  - High priority and normal priority round-robin with a 5:1 ratio
  - Other work can still be heavily starved
- Solution: choose priorities dynamically based on the use case



Bad idea #2: JS touch handling => Prioritize input & rendering

- Helps keep animation smooth

Bad idea #2: JS touch handling => Prioritize input & rendering

- Helps keep animation smooth
- Problem: remote desktop style applications
  - Assumed network tasks weren't important
  - Here network tasks are on the critical path, but get starved by rendering & input
- Solution: more fine grained use case detection
  - Main thread scrolling: prioritize input and rendering
  - Main thread touch handler: ¯_(ツ)_/¯

Bad idea #3: rAF means there’s an animation

- Assumed rAF callbacks always cause visual updates, so we ran the entire painting and compositing pipeline after them
- Don’t run rAF in hidden frames -- the user won’t see them, right?

Bad idea #3: rAF means there’s an animation

- Assumed rAF callbacks always cause visual updates, so we ran the entire painting and compositing pipeline after them
- Don’t run rAF in hidden frames -- the user won’t see them, right?
- Problem: Many rAF callbacks are passive, or drive logic unrelated to animation.
- Solutions:
  - Terminate painting/compositing pipeline if there were no updates
  - Don’t run rAF in hidden **cross-origin** frames

Bad idea #4: Animation means there’s a rAF

- Surely everyone is using rAF these days to drive their animations
- Let’s defer other work (timers) to make animations smoother

Bad idea #4: Animation means there's a rAF

- Surely everyone is using rAF these days to drive their animations
- Let's defer other work (timers) to make animations smoother
- Problem: lots of setTimeout()-based animation code out there -- even in popular frameworks
- Solution: Make fewer assumptions about what timers are being used for

Bad idea #5: Scrolling => Prioritize input & rendering

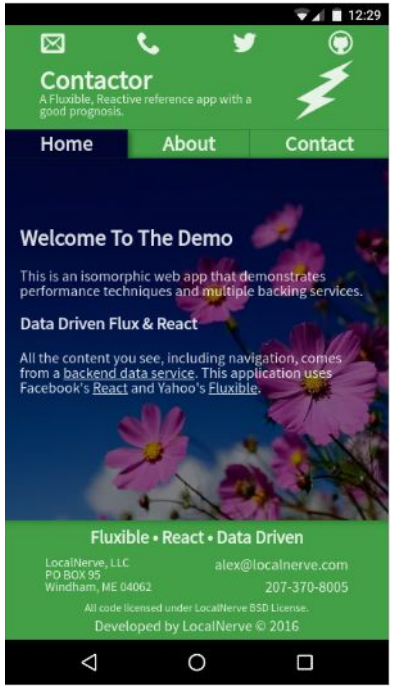
- Makes scroll animation smoother and more responsive

Bad idea #5: Scrolling => Prioritize input & rendering

- Makes scroll animation smoother and more responsive
- Problem: Imperfect use case detection
  - Page had wheel event handler, which did not preventDefault() => we thought we were scrolling
  - On low end machines rendering was expensive => WebSocket tasks got starved
- Solution: only prioritize rendering if it’s fast

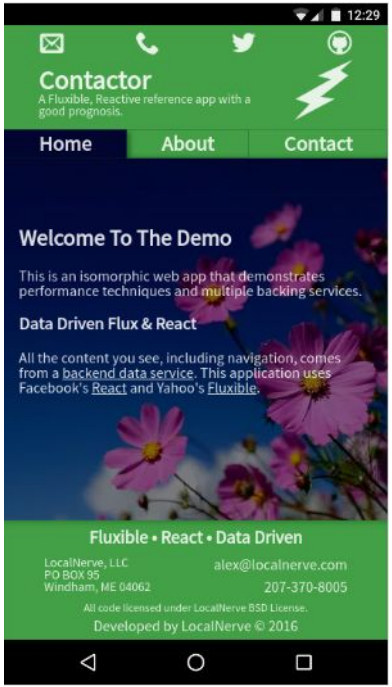
Bad idea #6: Defer tasks we think are unnecessary

- When expecting latency critical work (input), don’t start any unnecessary long running tasks
- Problem: it’s very difficult to work out which tasks are actually necessary
- Examples of breakage
  - Tab switching delay
  - Navigation delay after clicking on a link



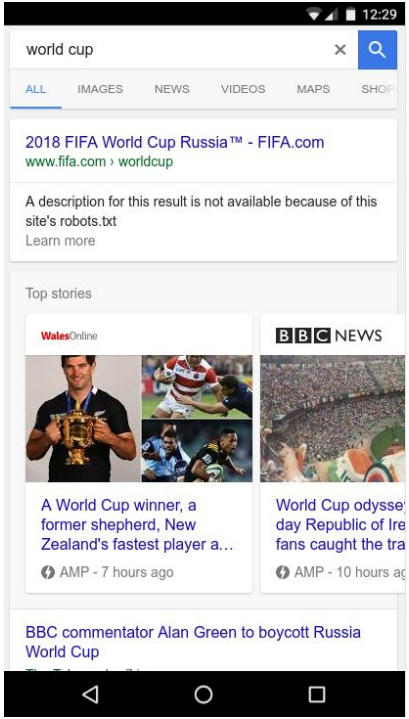
Bad idea #6: Defer tasks we think are unnecessary

- Solution: better use case detection
  - **Waiting for touch start:** block everything except touch handling
  - **During compositor scrolling:** block expensive things if expecting another gesture
  - **During main thread scrolling:** block expensive things if expecting another gesture, otherwise throttle to 1 Hz
  - **Idle:** block expensive things if expecting a new gesture



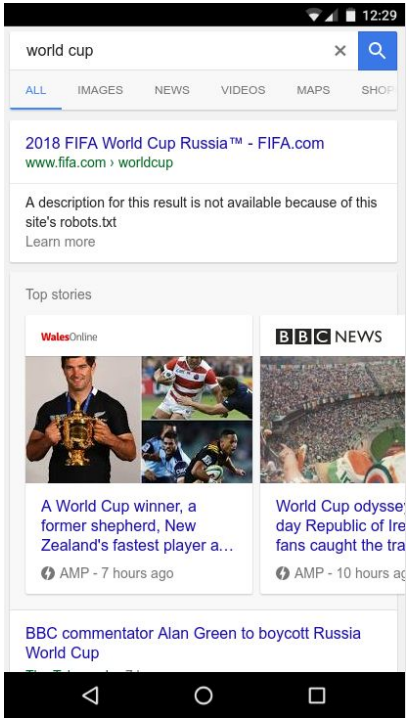
Bad idea #7: Stop network tasks while the user's finger is on the glass

- Use case: swiping between AMP search results
  - Javascript drives swipe animation
- New result brings in expensive JS (e.g., Youtube embed), causing an animation jank



Bad idea #7: Stop network tasks while the user's finger is on the glass

- Problem: sometimes network tasks are integral to the page's functionality
  - e.g., server side rendering
- (Tentative) solution: split network tasks into categories
  - e.g., HTML parsing is safer to defer than an XHR response
  - Deferring navigations is never safe



Lessons learned

- Scheduling the web’s hard yo
- If there exists an obscure corner case you didn’t think of, lots of pages are guaranteed to hit it
- Upside: with perseverance it’s possible to turn bad ideas into good ones