[Understanding the performance benefits of ConstraintLayout](https://android-developers.googleblog.com/2017/08/understanding-performance-benefits-of.html)

24 August 2017

*Posted by Takeshi Hagikura, Developer Programs Engineer*

Since announcing [ConstraintLayout](https://developer.android.com/training/constraint-layout/index.html) at Google I/O last year, we've continued to improve the layout's stability and layout editor support. We've also added new features specific to ConstraintLayout that help you build various type of layouts, such as [introducing chains](https://developer.android.com/training/constraint-layout/index.html#constrain-chain) and [setting size as a ratio](https://developer.android.com/training/constraint-layout/index.html#ratio). In addition to these features, there is a notable performance benefit by using ConstraintLayout. In this post, we'll walk through how you can benefit from these performance improvements.

**How Android draws views?**

To better understand the performance of ConstraintLayout, let's take a step back and see how Android draws views.

When a user brings an Android view into focus, the Android framework directs the view to draw itself. This drawing process comprises 3 phases:

1. **Measure**

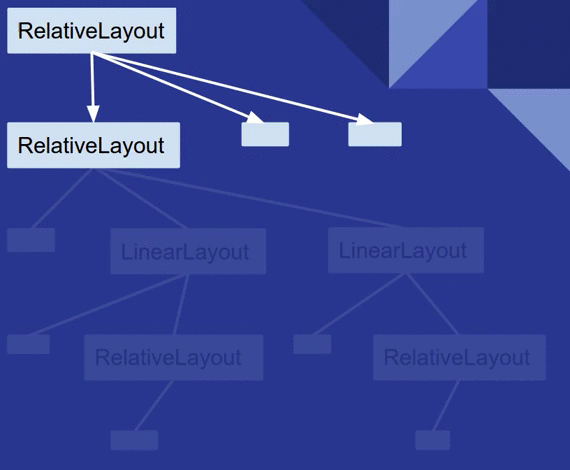
The system completes a top-down traversal of the view tree to determine how large each ViewGroup and View element should be. When a ViewGroup is measured, it also measures its children.

1. **Layout**

Another top-down traversal occurs, with each ViewGroup determining the positions of its children using the sizes determined in the measure phase.

1. **Draw**

The system performs yet another top-down traversal. For each object in the view tree, a Canvas object is created to send a list of drawing commands to the GPU. These commands include the ViewGroup and View objects' sizes and positions, which the system determined during the previous 2 phases.

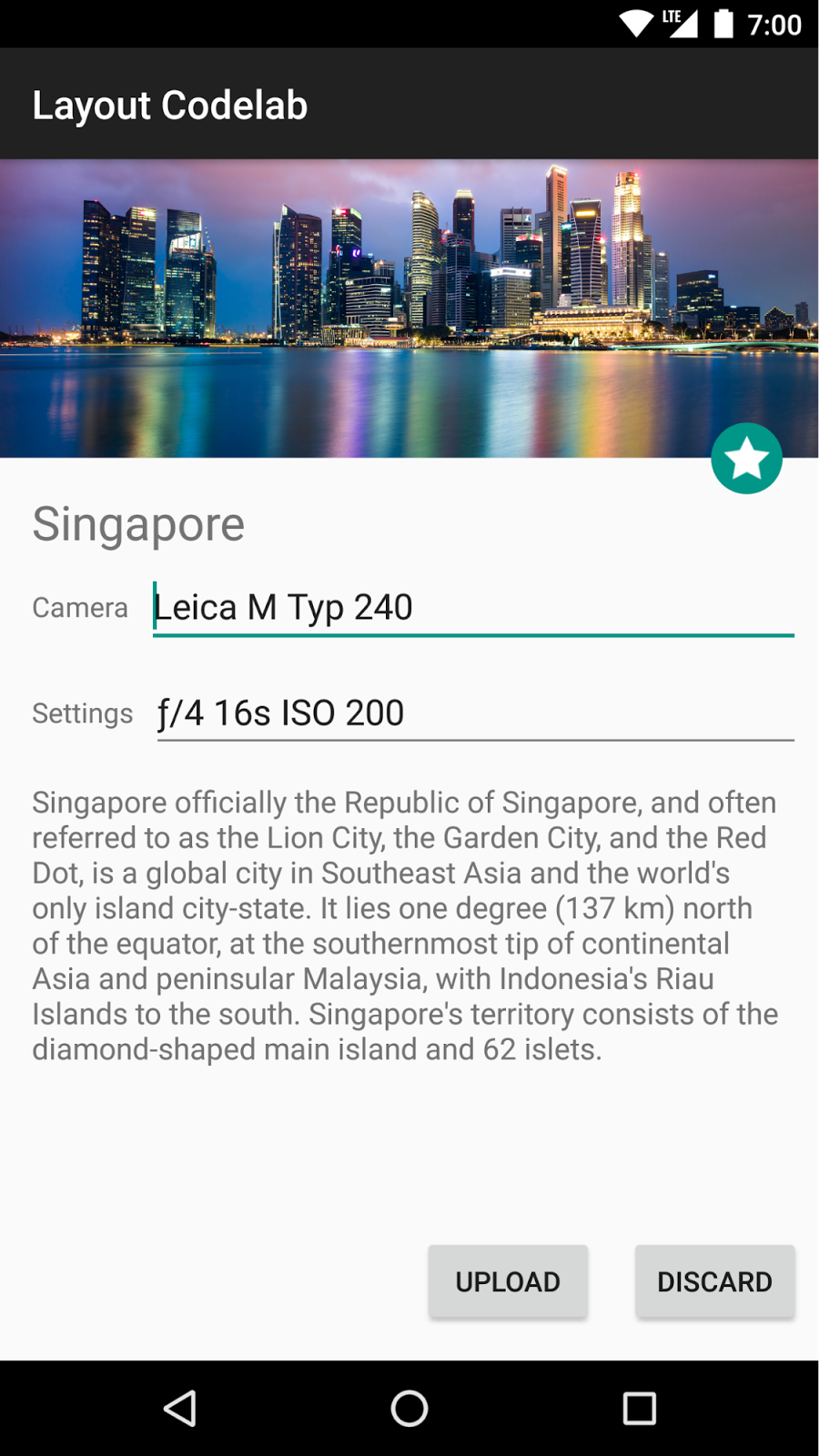
[](https://1.bp.blogspot.com/-WKiq_6Ei1Zg/WZ3iZ6A5krI/AAAAAAAAEgs/ioP7HT-izzwF4xIl_wnAqtLqTayT6aUOACLcBGAs/s1600/image6.gif)

***Figure 1.*** Example of how the measure phase traverses a view tree

Each phase within the drawing process requires a top-down traversal of the view tree. Therefore, the more views you embed within each other (or nest) into the view hierarchy, the more time and computation power it takes for the device to draw the views. By keeping a flat hierarchy in your Android app layouts, you can create a fast and responsive user interface for your app.

**The expense of a traditional layout hierarchy**

With that explanation in mind, let's create a traditional layout hierarchy that uses LinearLayout and RelativeLayout objects.

[](https://4.bp.blogspot.com/-0Q-BVkD8e_A/WZ3itMksXOI/AAAAAAAAEgw/dLRNKQj5FhE2NpSq6jDx-MlBuCEqUOwqQCLcBGAs/s1600/image5.png)

***Figure 2.*** Example layout

Let's say we want to build a layout like the image above. If you build it with traditional layouts, the XML file contains an element hierarchy similar to the following (for this example, we've omitted the attributes):

<RelativeLayout>

<ImageView />

<ImageView />

<RelativeLayout>

<TextView />

<LinearLayout>

<TextView />

<RelativeLayout>

<EditText />

</RelativeLayout>

</LinearLayout>

<LinearLayout>

<TextView />

<RelativeLayout>

<EditText />

</RelativeLayout>

</LinearLayout>

<TextView />

</RelativeLayout>

<LinearLayout >

<Button />

<Button />

</LinearLayout>

</RelativeLayout>

Although there's usually room for improvement in this type of view hierarchy, you'll almost certainly still need to create a hierarchy with some nested views.

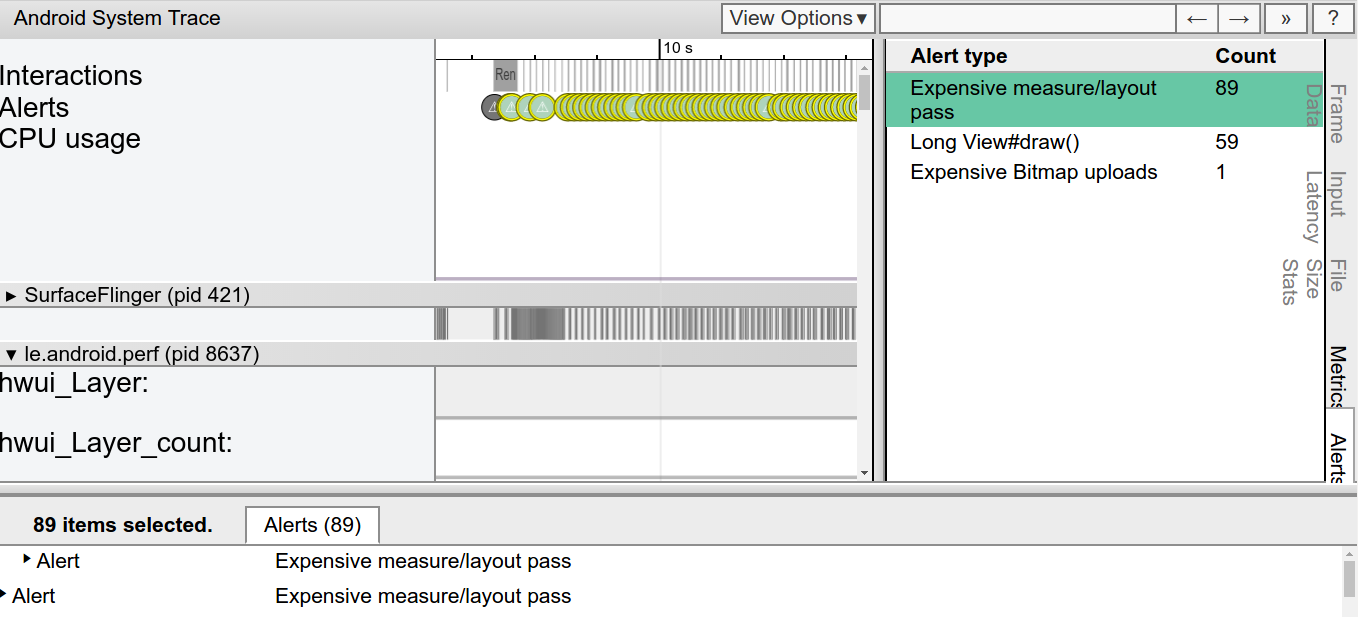
As discussed before, nested hierarchies can adversely affect performance. Let's take a look at how the nested views actually affect the UI performance using Android Studio's [Systrace](https://developer.android.com/studio/profile/systrace-commandline.html) tool. We called the measure and layout phases for each ViewGroup (ConstraintLayout and RelativeLayout) programmatically and triggered Systrace while the measure and layout calls are executing. The following command generates an overview file that contains key events, such as expensive measure/layout passes, that occur during a 20-second interval:

python $ANDROID\_HOME/platform-tools/systrace/systrace.py --time=20 -o ~/trace.html gfx view res

For more details about how you can use Systrace, see the [Analyzing UI Performance with Systrace](https://developer.android.com/studio/profile/systrace.html) guide.

Systrace automatically highlights the (numerous) performance problems with this layout, as well as suggestions for fixing them. By clicking the "Alerts" tab, you will find that drawing this view hierarchy requires 80 expensive passes through the measure and layout phases!

Triggering that many expensive measure and layout phases is far from ideal; such a large amount of drawing activity could result in skipped frames that users notice. We can conclude that the layout has poor performance due to the nested hierarchy as well as the characteristic of RelativeLayout, which measures each of its children twice.

[](https://4.bp.blogspot.com/-i-QqmN2sU8U/WZ3i8N2ktdI/AAAAAAAAEg0/jJ_0DlQjsw8mCJSTLRDkqgzlA84x7acRgCLcBGAs/s1600/image1.png)

***Figure 3.*** Looking at the alerts from Systrace for the layout variant that uses RelativeLayout

You can check the entire code on how we performed these measurements in our [GitHub repository](https://github.com/googlesamples/android-constraint-layout-performance).

**The benefits of a ConstraintLayout object**

If you create the same layout using ConstraintLayout, the XML file contains an element hierarchy similar to the following (attributes again omitted):

<android.support.constraint.ConstraintLayout>

<ImageView />

<ImageView />

<TextView />

<EditText />

<TextView />

<TextView />

<EditText />

<Button />

<Button />

<TextView />

</android.support.constraint.ConstraintLayout>

As this example shows, the layout now has a completely flat hierarchy. This is because ConstraintLayout allows you to build complex layouts without having to nest View and ViewGroup elements.

For example, let's look at the TextView and EditText in the middle of the layout:

[](https://2.bp.blogspot.com/-m3b0oznyHnk/WZ3jMX6rZfI/AAAAAAAAEg4/tlui5o6219E4VDUZ7hACve0Nv3dAIg07gCLcBGAs/s1600/image4.png)

When using a RelativeLayout, you need to create a new ViewGroup to align the EditText vertically with the TextView:

<LinearLayout

android:id="@+id/camera\_area"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:orientation="horizontal"

android:layout\_below="@id/title" >

<TextView

android:text="@string/camera"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_gravity="center\_vertical"

android:id="@+id/cameraLabel"

android:labelFor="@+id/cameraType"

android:layout\_marginStart="16dp" />

<RelativeLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<EditText

android:id="@+id/cameraType"

android:ems="10"

android:inputType="textPersonName"

android:text="@string/camera\_value"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_centerVertical="true"

android:layout\_marginTop="8dp"

android:layout\_marginStart="8dp"

android:layout\_marginEnd="8dp" />

</RelativeLayout>

</LinearLayout>

By using ConstraintLayout instead, you can achieve the same effect just by adding a constraint from the baseline of the TextView to the baseline of the EditText without creating another ViewGroup:

[](https://4.bp.blogspot.com/-MJ9yc4BzEoE/WZ3jYv1RlpI/AAAAAAAAEg8/4c-a7w5rTpE7n1SNZCERfcmxwIKyRAdPQCLcBGAs/s1600/image7.png)

***Figure 4.*** Constraint between EditText and TextView

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

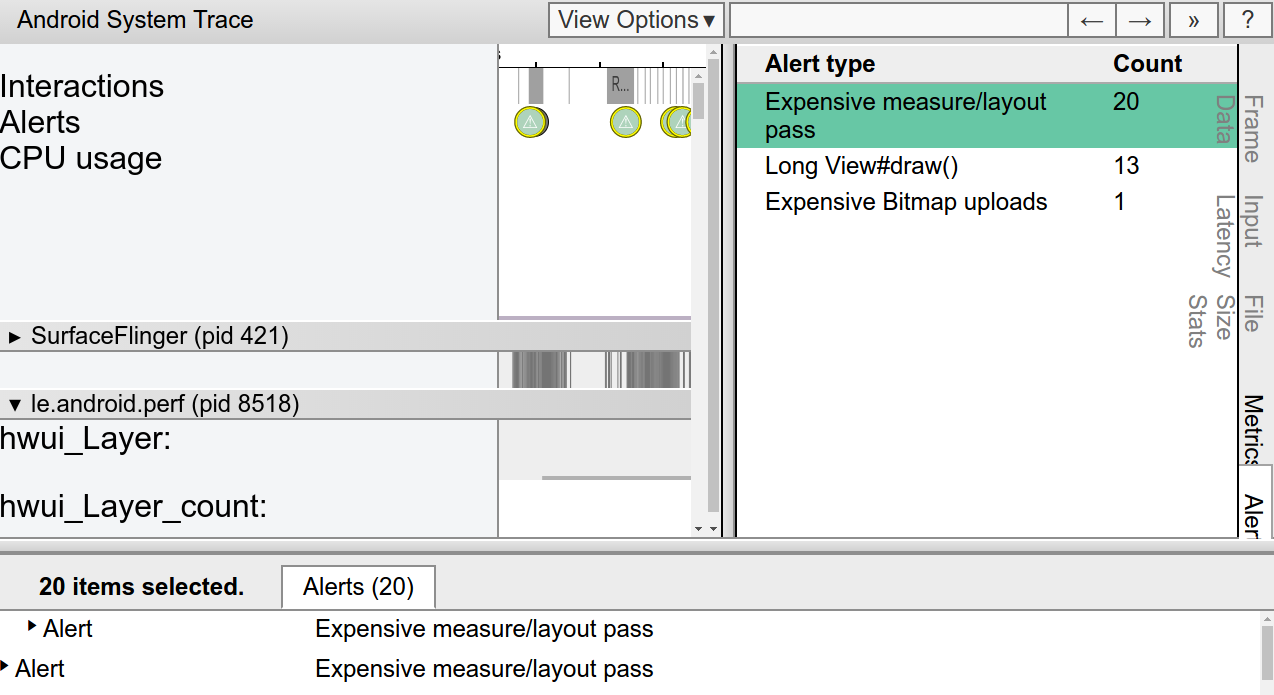
app:layout\_constraintLeft\_creator="1"

app:layout\_constraintBaseline\_creator="1"

app:layout\_constraintLeft\_toLeftOf="@+id/activity\_main\_done"

app:layout\_constraintBaseline\_toBaselineOf="@+id/cameraType" />

When running the Systrace tool for the version of our layout that uses ConstraintLayout, you see far fewer expensive measure/layout passes during the same 20-second interval. This improvement in performance makes sense, now that we're keeping the view hierarchy flat!

[](https://4.bp.blogspot.com/-1OKtg8DSnmM/WZ3jrmE2KrI/AAAAAAAAEhA/Avij5jmePpob8fRnd8l9PqOewuDIrk-GQCLcBGAs/s1600/image3.png)

***Figure 5.***Looking at the alerts from Systrace for the layout variant that uses ConstraintLayout

On a related note, we built the ConstraintLayout variant of our layout using just the [layout editor](https://developer.android.com/training/constraint-layout/index.html) instead of editing the XML by hand. To achieve the same visual effect using RelativeLayout, we probably would have needed to edit the XML by hand.

**Measuring the performance difference**

We analyzed how long every measure and layout pass took for two type of layouts, ConstraintLayout and RelativeLayout, by using [OnFrameMetricsAvailableListener](https://developer.android.com/reference/android/view/Window.OnFrameMetricsAvailableListener.html), which was introduced in Android 7.0 (API level 24). This class allows you to collect frame-by-frame timing information about your app's UI rendering.

By calling the following code, you can start recording per-frame UI actions:

window.addOnFrameMetricsAvailableListener(

frameMetricsAvailableListener, frameMetricsHandler);

After timing information becomes available, the app triggers the frameMetricsAvailableListener() callback. We are interested in the measure/layout performance, so we call [FrameMetrics.LAYOUT\_MEASURE\_DURATION](https://developer.android.com/reference/android/view/FrameMetrics.html#LAYOUT_MEASURE_DURATION) when retrieving the actual frame duration.

Window.OnFrameMetricsAvailableListener {

\_, frameMetrics, \_ ->

val frameMetricsCopy = FrameMetrics(frameMetrics);

// Layout measure duration in nanoseconds

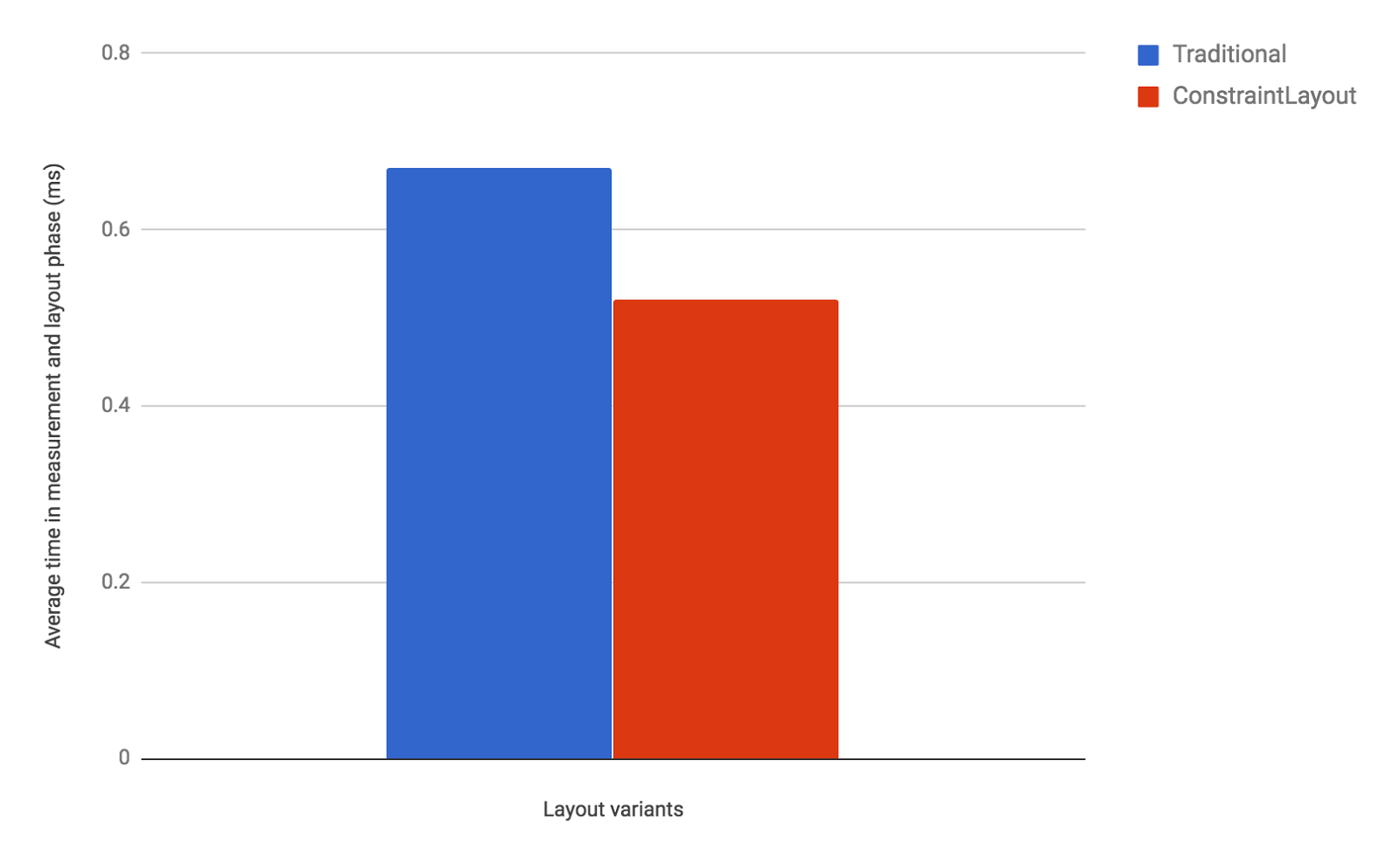
val layoutMeasureDurationNs =

frameMetricsCopy.getMetric(FrameMetrics.LAYOUT\_MEASURE\_DURATION);

To learn more about the other types of duration information that FrameMetrics can receive, see the [FrameMetrics](https://developer.android.com/reference/android/view/FrameMetrics.html) API reference.

**Measurement results: ConstraintLayout is faster**

Our performance comparison shows that ConstraintLayout performs about 40% better in the measure/layout phase than RelativeLayout:

[](https://4.bp.blogspot.com/-iZSeC2m1Ti8/WZ3j3rAiq6I/AAAAAAAAEhE/4bdOKMUvUSsyx563WZ7lIgZoeYdGmLgTwCLcBGAs/s1600/image2.png)

***Figure 6.*** Measure / Layout (unit: ms, average of 100 frames)

As these results show, ConstraintLayout is likely to be more performant than traditional layouts. Moreover, ConstraintLayout has other features that help you build complex and performant layouts, as discussed in the [benefits of a ConstraintLayout object](https://android-developers.googleblog.com/2017/08/understanding-performance-benefits-of.html#bookmark=kix.qjly41rrtsa9) section. For details, see the [Build a Responsive UI with ConstraintLayout](https://developer.android.com/training/constraint-layout/index.html) guide. We recommend that you use ConstraintLayout when designing your app's layouts. In almost all cases when you would have previously need a deeply-nested layout, ConstraintLayout should be your go-to layout for optimal performance and ease of use.

**Appendix: Measurement environment**

All the measurements above were performed in the following environment.

|  |  |
| --- | --- |
| Device | Nexus 5X |
| Android Version | 8.0 |
| ConstraintLayout version | 1.0.2 |

**What's next**

Check out the [developer guide](https://developer.android.com/training/constraint-layout/index.html), the [API reference documentation](https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html), and the [article on Medium](https://medium.com/google-developers/building-interfaces-with-constraintlayout-3958fa38a9f7) to fully understand what ConstraintLayout can provide for you. And once again, thank you to all who submitted feedback and issues over the months since our alpha release of ConstraintLayout. We're truly grateful that we were able to release the production-ready [1.0 version](http://tools.android.com/recent/constraintlayout10isnowavailable) of ConstraintLayout earlier this year. As we continue to improve ConstraintLayout, please continue to send us feedback using the [Android issue tracker](http://b.android.com/).