## **Triggered Flow Runtime Behavior**

The rest of this document describes technical details about the Flow runtime.

### **Performance Discussion: Same-Record Field Updates**

Approximately 150 billion actions were executed by Workflow, Process Builder, and Flow in April 2020, including record updates, email alerts, outbound messages, and invocable actions. Around 100 billion of those 150 billion actions were same-record field updates. Note that before-save flow triggers had only been launched the release before, so that means 100 billion after-save same-record field updates — or equivalently, 100 billion recursive saves — were executed in just one month. Imagine how much time could have been saved by before-save flow triggers

Caveat: Architects should view all performance claims with a critical eye, even when they come from Salesforce. Results in your org will likely be different than the results in our orgs.

Earlier in this guide, we noted that while Workflow Rules have a reputation for being fast, they will always be slower and more resource-hungry than a single, functionally equivalent before-save flow trigger. The theoretical side to this assertion is that, by design, before-save flow triggers neither cause DML operations nor the ensuing recursive firing of the save order, while Workflow Rules do (because they happen after the save).

But what happens in practice? We ran a few experiments to find out.

#### **[Experiment 1] Single trigger; single record created from the UI; Apex debug log duration**

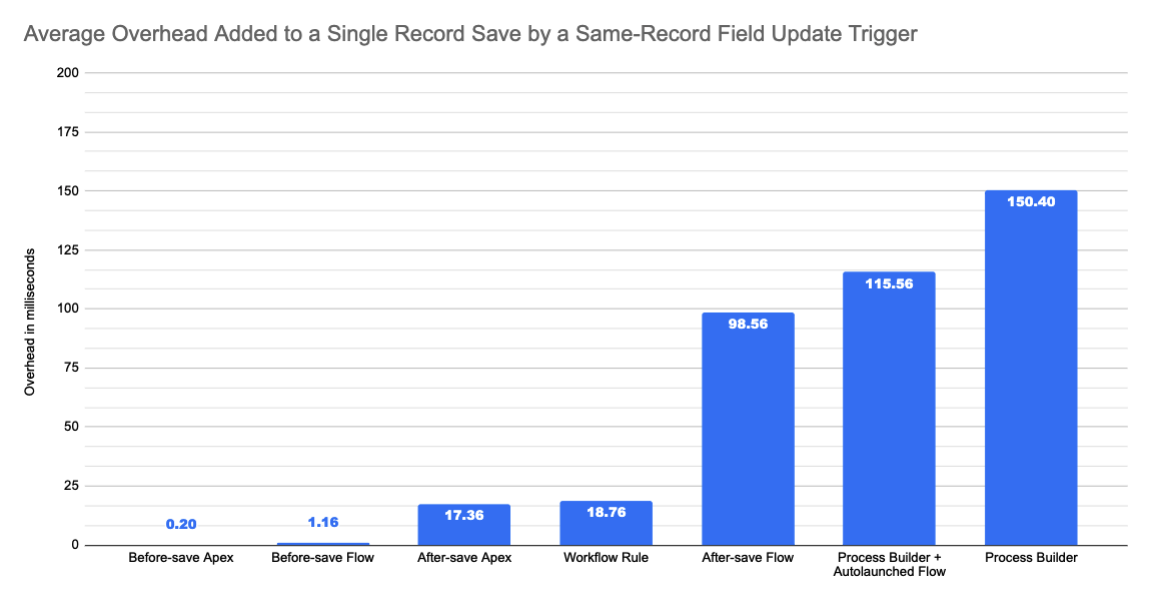
***How much longer does an end user have to wait for a record to save?***

For each of the different automation tools that can be used to automate a same-record field update, we created a fresh org, + one more fresh org to serve as a baseline org.

Then for each org, we:

1. Except for the baseline org, implemented the simplest version of a trigger on Opportunity Create that would set Opportunity.NextStep = Opportunity.Amount.
2. Enabled Apex debug logging, with all debug levels set to None except Workflow.Info and Apex Code.Debug
3. Manually created a new Opportunity record with a populated Amount value through the UI, 25 times.
4. Calculated the average duration of the log across the 25 transactions.
5. Subtracted from the average duration in #4, the average duration of the log in the baseline org.

This gave us the average overhead that each trigger added to the log duration.



#### **[Experiment 2] 50 triggers; 50,000 records inserted via Bulk API (200 record batches); internal tooling**

***How about the other side of the spectrum: high-volume batch processing?***

We borrowed some of our performance team’s internal environments to get a sense of how well the different trigger tools scale.

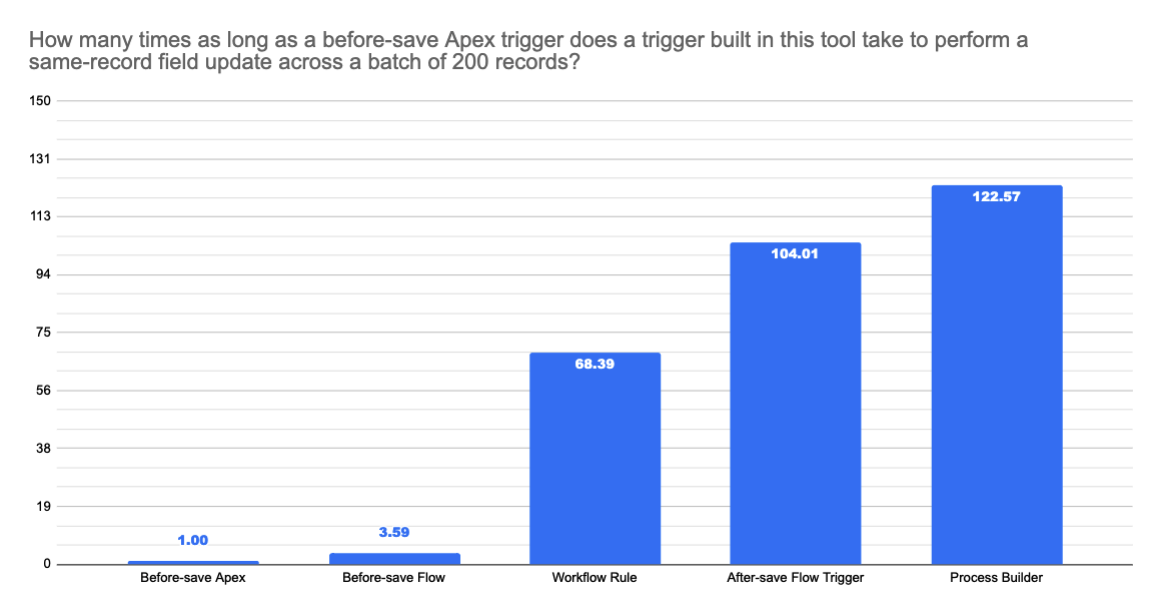
The configuration was:

* 1 org with 50 before-save flows on Account Create which each update Account.ShippingPostalCode
* 1 org with 50 before-save Apex triggers on Account Create which each update Account.ShippingPostalCode
* 1 org with 50 Workflow Rules on Account Create which each update Account.ShippingPostalCode
* 1 org with 50 after-save flow triggers on Account Create which each update Account.ShippingPostalCode
* 1 org with 50 Process Builder processes on Account Create which each update Account.ShippingPostalCode

Then each Tuesday for the last 12 weeks, we uploaded 50,000 Accounts to each org through the Bulk API, with a 200-record batch size.

Fortunately, our internal environments can directly profile trigger execution time without requiring Apex debug logging or extrapolation from a baseline.

Because our internal environments are not representative of production, we’re sharing only the relative performance timings, and not the raw performance timings.



In both single-record and bulk use cases, the before-save Flow performs extremely well. As much as we’d like to take credit for the outcomes, however, most of the performance savings come simply due to the huge advantage of being before the save.

Go forth and stop implementing same-record field updates in Workflow Rules and Process Builder!