

# Improve the progress tracking of Naiad (Timely Dataflow)

Victor

Nov 2020

# Timely Dataflow

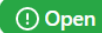
- Timely dataflow is a powerful dataflow framework (and under-appreciated IMO)
- Supports both high-throughput batch processing and low-latency stream processing
- Supports static cyclic graph
- Uses stateful workers with less fault tolerance capabilities
- Use timestamps for progress tracking and batch synchronization
- Still under active development and usage (by Differential Dataflow and Materialize)

# Personal interests

- Timely dataflow (and differential dataflow) is interesting and powerful
- Dissertation project involves combining Timely Dataflow and enclaves
- Shift some operators to enclaves to make them confidential on untrusted cloud
- Additional opportunity to learn about the implementation
- (And also learn Rust)

# What's the problem here?

## Progress tracking needs to be reigned in. #190



frankmcherry opened this issue on Sep 19, 2018 · 1 comment



frankmcherry commented on Sep 19, 2018 · edited

Member



Recent measurements, using [eintopf](#) as a basis and grokking logs [with this program](#) reveals that by volume (bytes) there is a crap-ton of progress traffic going on. As we increase the workers up to 32, the volume even dominates the amount of real data.

Naiad had exactly this problem and instituted several measures for optimizing the traffic, including switching to edge-based transmission where accumulations are only sent out when discrete changes in the global frontier are observed, and aggregation at various levels that makes this even more effective.

We should probably do the same thing... Sigh.

As examples, using four processes each with one worker produces communication channel by counts of

```
MESSAGE (11, (Root, Duration { secs: 100, nanos: 0 }), 1799904)
MESSAGE (15, (Root, Duration { secs: 100, nanos: 0 }), 98112480)
MESSAGE (22, (Root, Duration { secs: 100, nanos: 0 }), 1065273136)
MESSAGE (26, (Root, Duration { secs: 100, nanos: 0 }), 534406272)
```

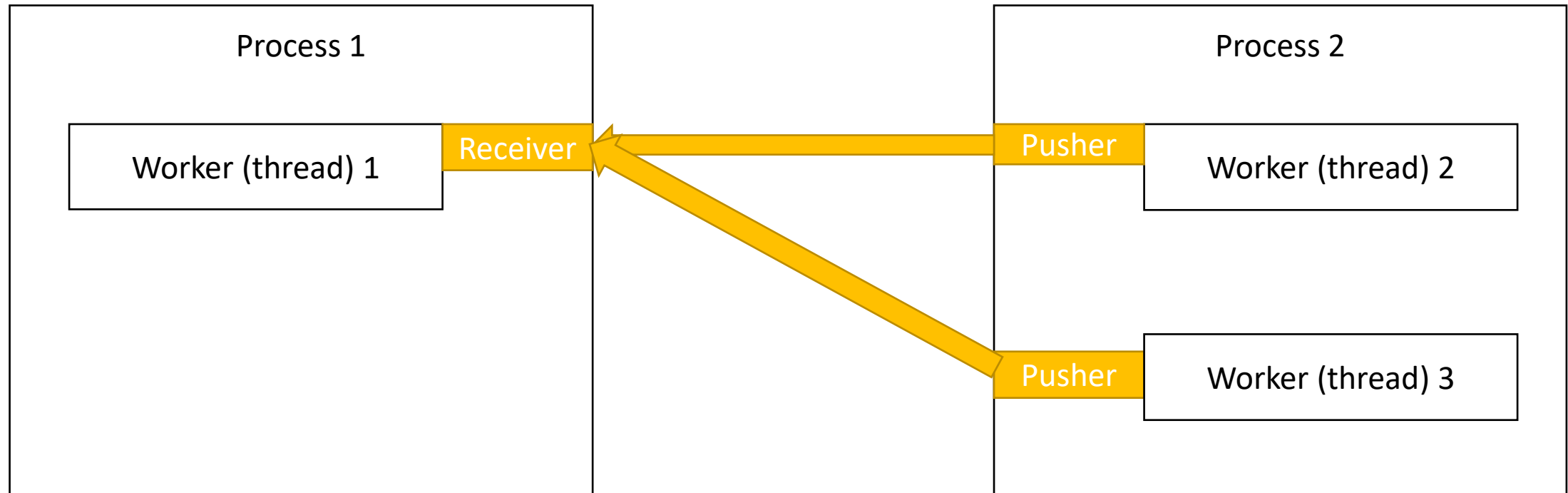
where 26 is the progress channel and others are data channels.

As we increase the workers to 4x8 this increases to

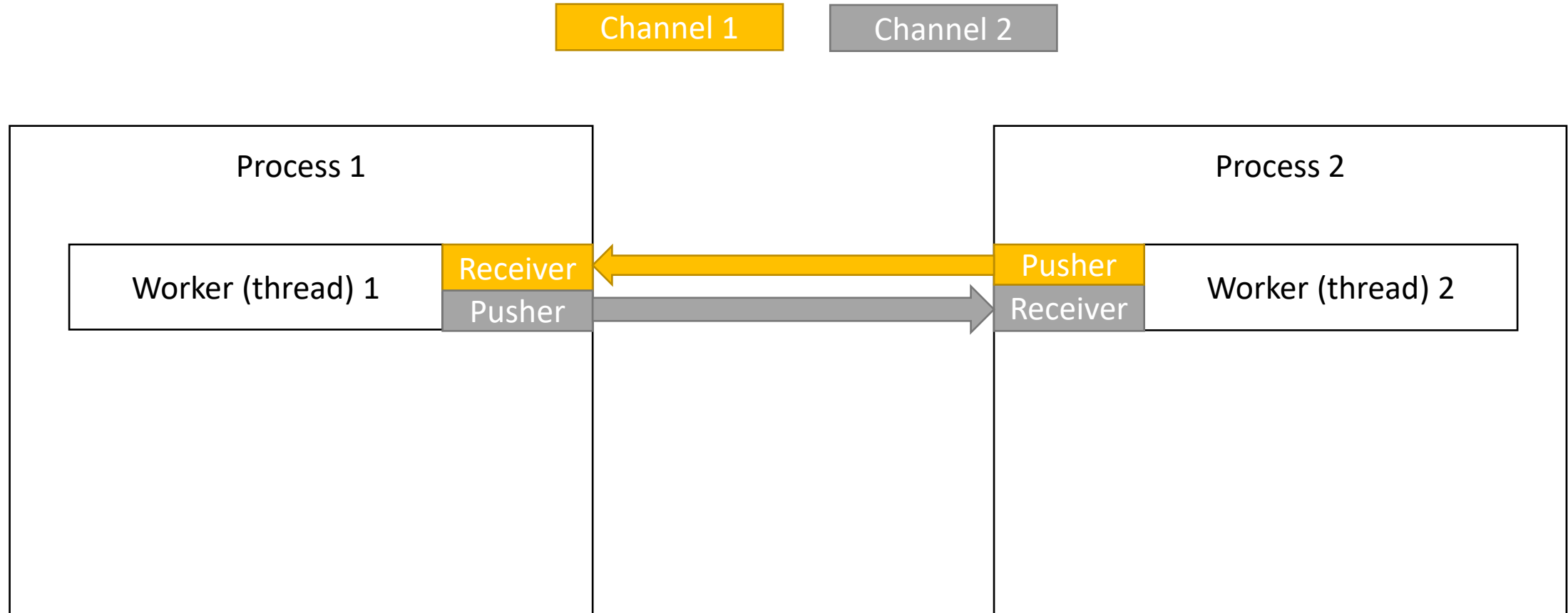
```
MESSAGE (11, (Root, Duration { secs: 100, nanos: 0 }), 1818912)
MESSAGE (15, (Root, Duration { secs: 100, nanos: 0 }), 24392448)
MESSAGE (22, (Root, Duration { secs: 100, nanos: 0 }), 162786032)
MESSAGE (26, (Root, Duration { secs: 100, nanos: 0 }), 675490816)
```

# What's the problem here?

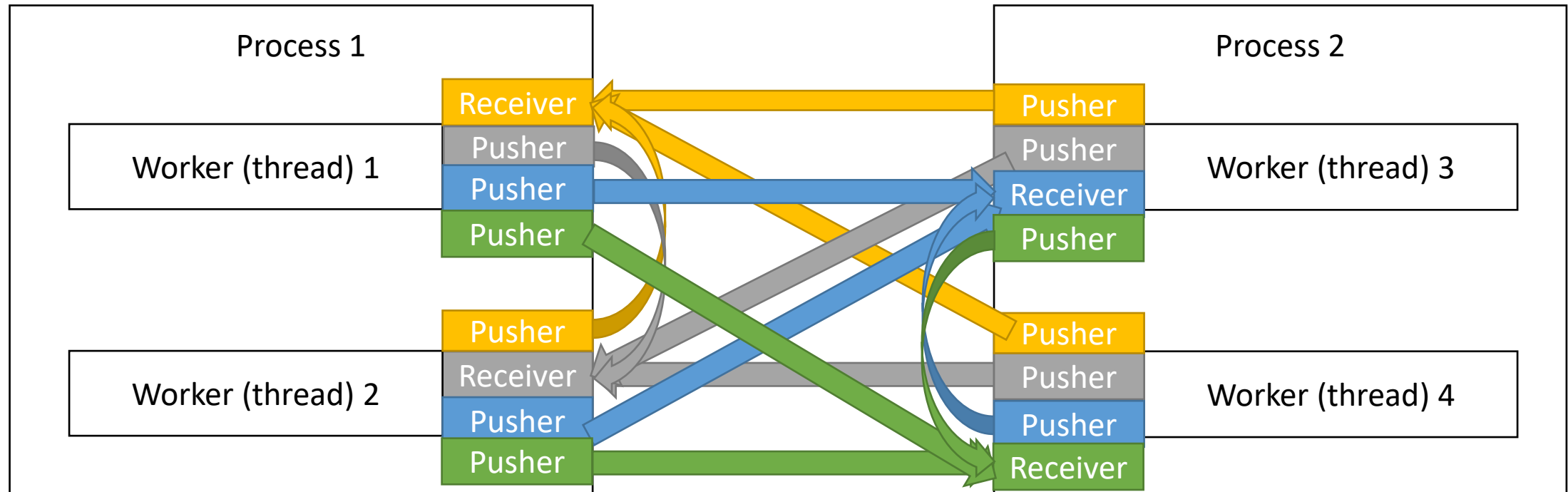
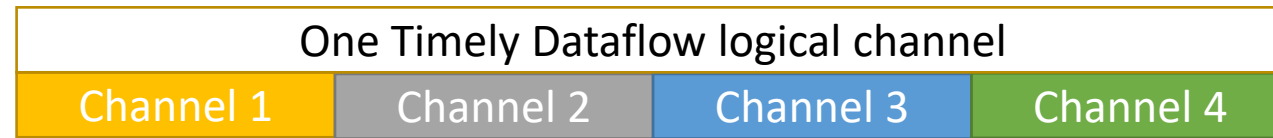
Channel (either data or control plane)



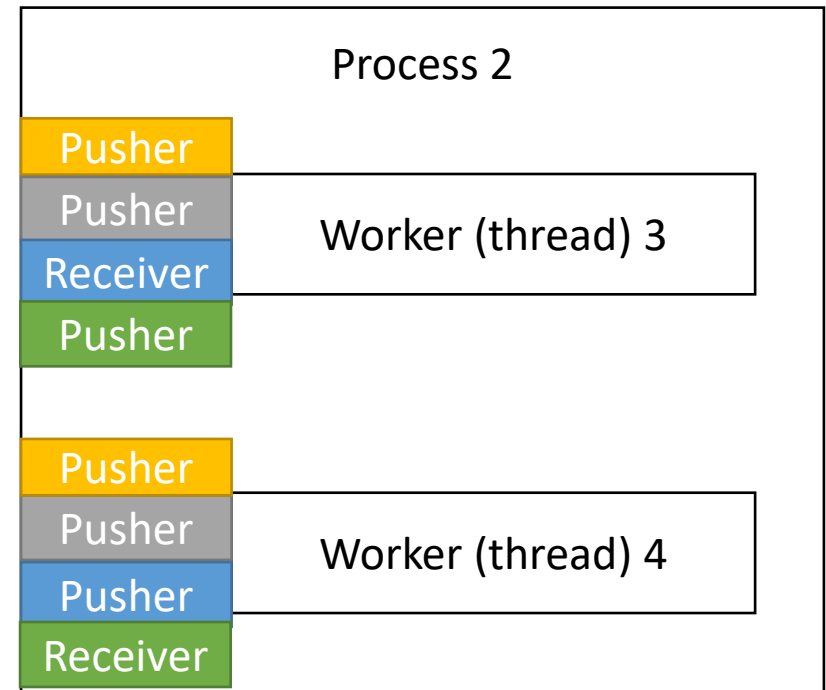
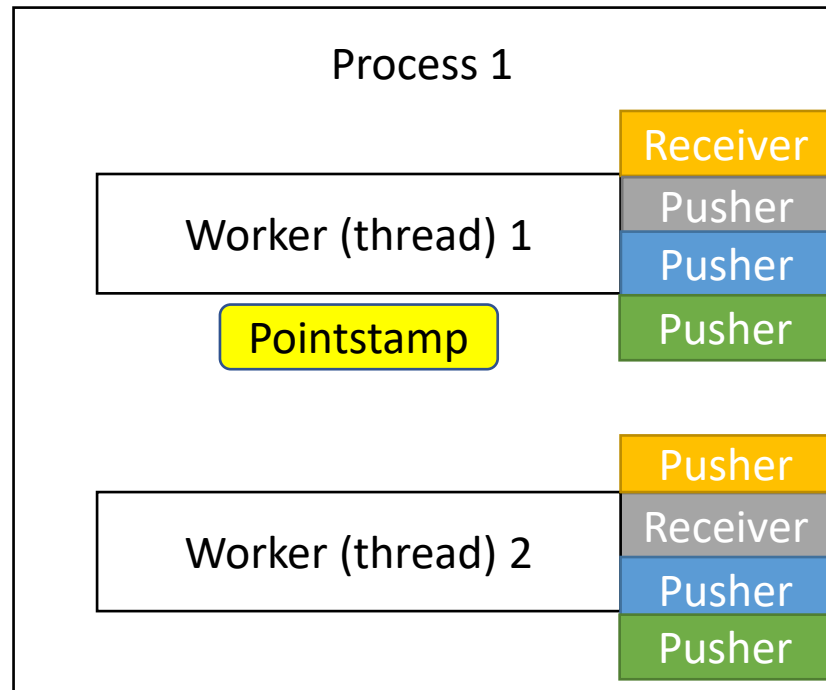
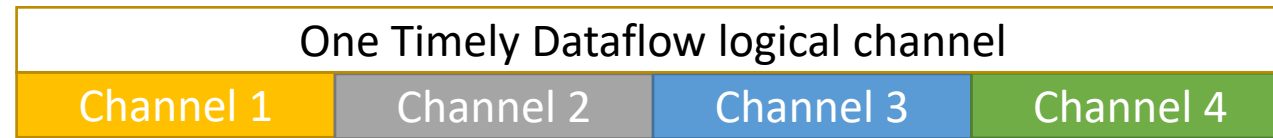
# What's the problem here?



# What's the problem here?

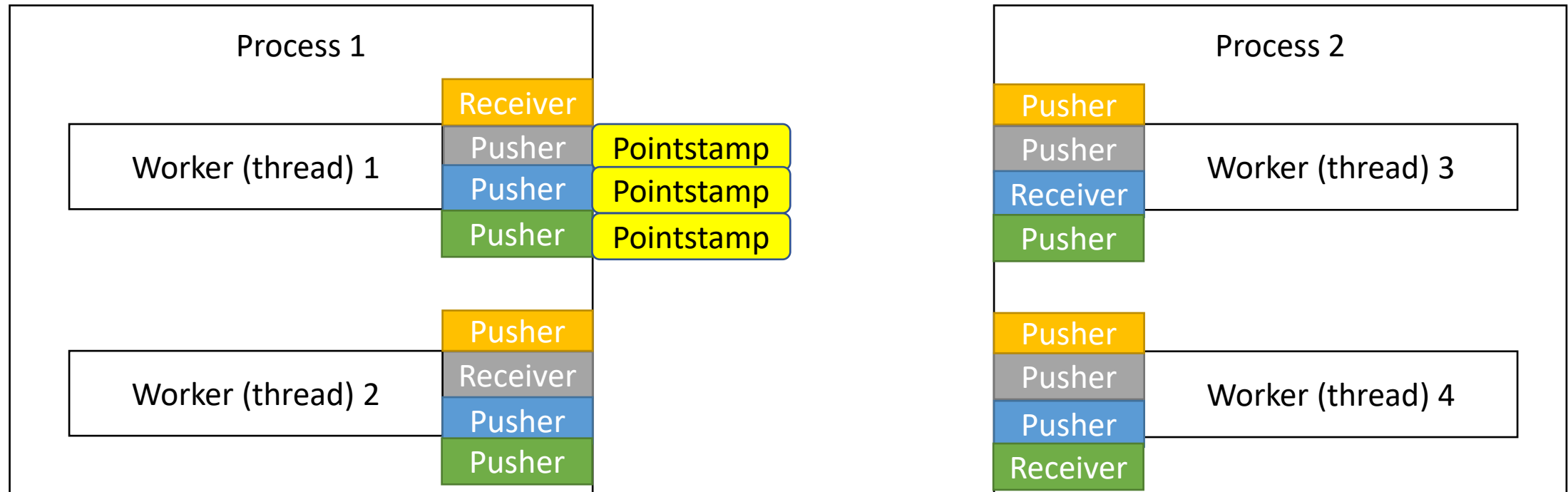
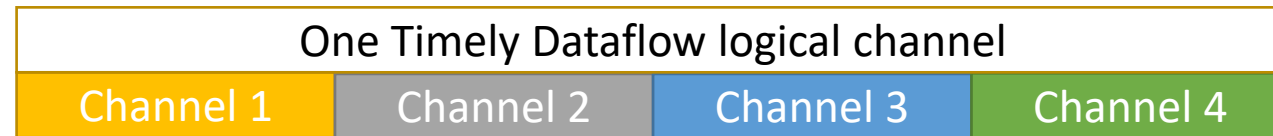


# What's the problem here?

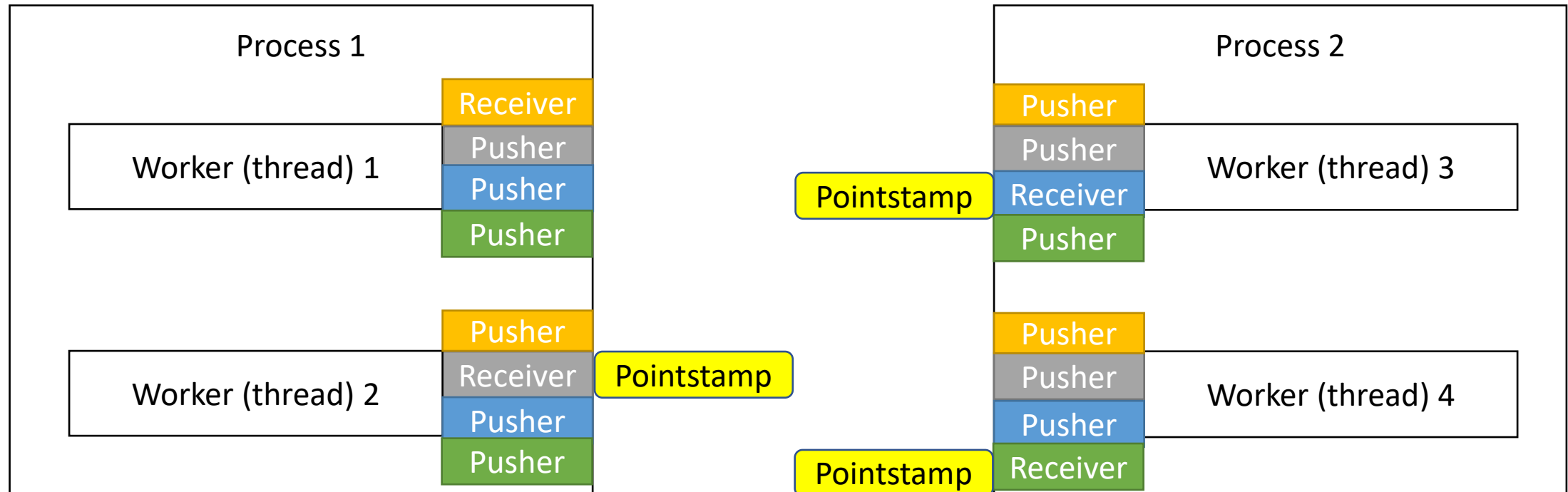
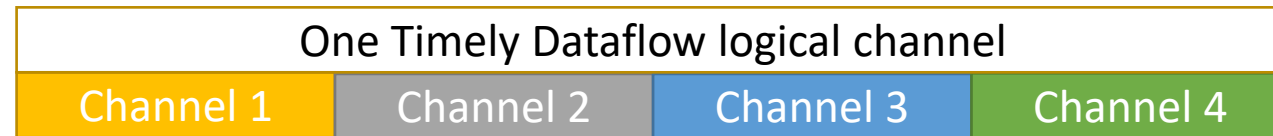




# What's the problem here?



# What's the problem here?



# What's the problem here?

- If we have 4 processes and 8 workers....

```
MESSAGE (11, (Root, Duration { secs: 100, nanos: 0 })), 1818912)
MESSAGE (15, (Root, Duration { secs: 100, nanos: 0 })), 24392448)
MESSAGE (22, (Root, Duration { secs: 100, nanos: 0 })), 162786032)
MESSAGE (26, (Root, Duration { secs: 100, nanos: 0 })), 675490816)
```

- Channel 26 here is the channel for progress update
- Other channels are for sending data
- Progress data is much larger than data actually used in dataflow operators

# What are the possible solutions?

- Timely Dataflow is slightly different from the original Naiad in C#
- Naiad has optimizations for reducing progress update traffic
- Hierarchically accumulate the updates at process or central broadcaster level
  - **Worker level:** Only send update when the updates are needed by other workers to yield notifications (and results)
  - **Process level:** Each process accumulate all updates from local threads and process accumulators communicate progress updates with one another
  - **Cluster level:** All worker updates are dispatched to a central accumulator, and the central accumulator dispatch updates to all workers

# Accumulate progress updates at each worker

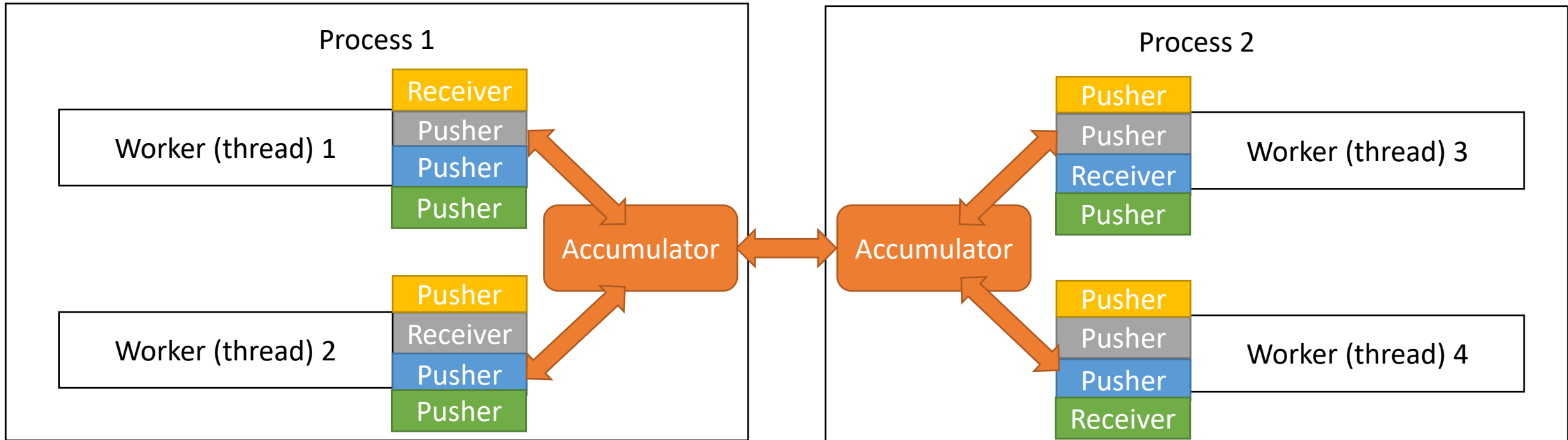
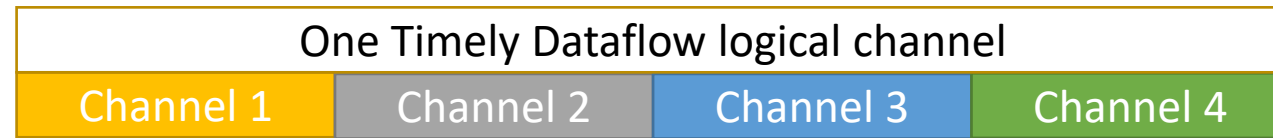
- This solution has been implemented by Frank McSherry
- <https://github.com/TimelyDataflow/timely-dataflow/pull/228>

```
/// Sends local progress updates to all workers.
///
/// This method does not guarantee that all of `self.local_pointstamps` are
/// sent, but that no blocking pointstamps remain
fn send_progress(&mut self) {
    // If we are requested to eagerly send progress updates, or if there are
    // updates visible in the scope-wide frontier, we must send all updates.
    let must_send = self.eager_progress_send || {
        let tracker = &mut self.pointstamp_tracker;
        self.local_pointstamp.iter().any(|((location, time), diff)|
            // Must publish scope-wide visible subtractions.
            tracker.is_global(*location, time) && *diff < 0)
    };

    if must_send {
        self.progcaster.send(&mut self.local_pointstamp);
    }
}
```

- This optimization is not (well) evaluated and disabled by default
- Need some computations to evaluate this optimization

# Accumulate progress updates at each process



# Accumulate progress updates at each process

- No implementation yet
- Every group of workers now have shared progress updates
- Distributed append-only log
- Concurrency and correctness issue
- Performance issue due to message delay and inter-thread synchronization


# (My own) progress updates

- Read the Rust implementation (and learn Rust)
- Communication, scheduling, progress tracking and updates
- Find programs to evaluate the changes
- Implement the evaluation programs
- Implement my own changes
- Evaluate both changes on progress traffic volume and effect on latency and throughput
- Wrap my change as an optional feature and submit a PR



# Contingencies/add-ons

## Optimize progress traffic for some operators #94

 Open frankmcsberry opened this issue on Sep 1, 2017 · 0 comments




frankmcsberry commented on Sep 1, 2017 · edited ▾

Member



Several operators do not require progress traffic.

## Progress information flows slowly #56

 Open frankmcsberry opened this issue on Apr 15, 2017 · 0 comments



frankmcsberry commented on Apr 15, 2017 · edited ▾

Member



Not *that* slowly, not to worry. ;)

Discussion/Q&A