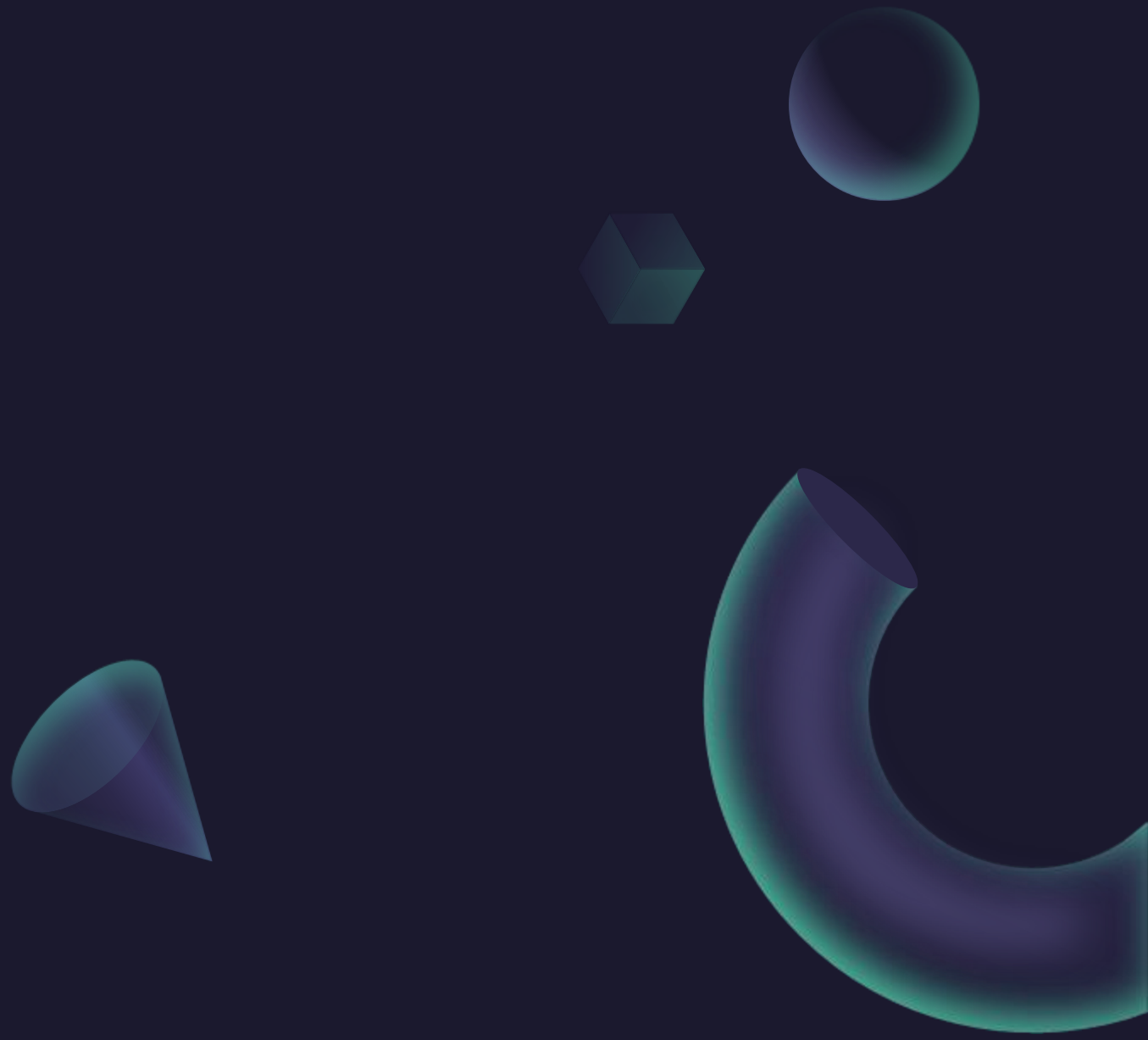
Twitch Data Analysis

The application

- Gather data and process it
- Visualize that data
- Benchmark the improvements
- Difficulties



Use of Async and Parallel

Fetch stage:

- Async API calls
- Streams fetched in parallel for 20 games

```
var topGamesRespose = await Api.Helix.Games.GetTopGamesAsync();

var fetchStreamTasks = topGamesRespose.Data
    .Select(game => Api.Helix.Streams.GetStreamsAsync(gameIds: [game.Id], first: 100))
    .ToList();

var streamResult = await Task.WhenAll(tasks: fetchStreamTasks);

var fetchUserTasks = streamResult.Select(gameStreams => Api.Helix.Users.GetUsersAsync(
    ids: [.. gameStreams.Streams.Select(stream => stream.UserId)]
));

var usersResult = await Task.WhenAll(tasks: fetchUserTasks);
```

Process stage:

- Stream and user data processed in parallel

```
var streamResults = await Task.WhenAll(FetchedData.StreamsResponses.Select(gameStreams => computeStreams(gameStreams.Streams)));
var userResults = await Task.WhenAll(FetchedData.UsersResponse.Select(gameUsers => computeUsers(gameUsers.Users)));
```

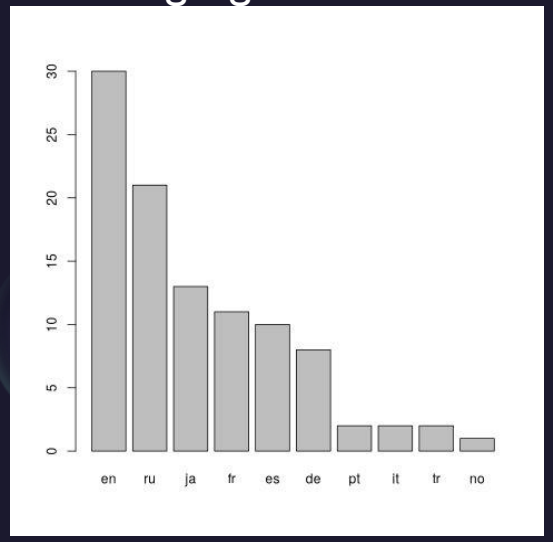
The Results

- Improvements

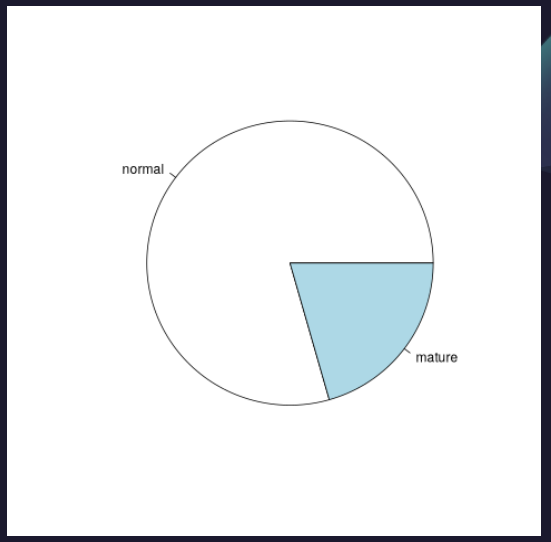
Before	After
Avg Time: 10188.3 fetch: 8928.9 process: 15.7	Avg Time: 5197.6 fetch: 3907.6 process: 14.6

- Some Data Results

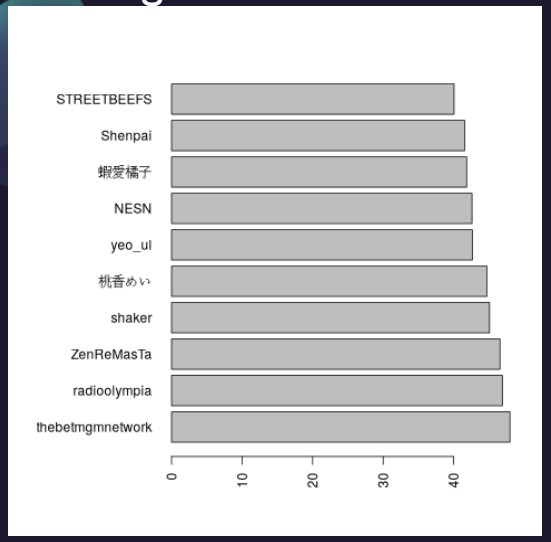
Rust language stats:



Mature content



10 Longest Streames



Streamer Types