Benchmark execution

Ⓗ connecting the dots.

1) Identify benchmarks to run

2) Create BenchmarkInstance(s)

for each identified Benchmark

3) Create a Runner for each

BenchmarkInstance.

4) Prepare repetition of each Runner

5) Run an instance using

Runner.

```
for (auto const& args :vector<int64_t> const& : family->args_) {
  for (int num_threads : *thread_counts) {
    BenchmarkInstance instance(family.get(), family_idx:family_index,
                               per_family_instance_index, args,
                               thread_count:num_threads);

    const auto full_name :string = instance.name().str();
    if (full_name.rfind( s: kDisabledPrefix, pos: 0) != 0 &&
        ((re.Match(full_name) && !is_negative_filter) ||
         (!re.Match(full_name) && is_negative_filter))) {
      benchmarks->push_back(std::move(instance));
```

① Identifying benchmarks involves BenchmarkFamilies.

② Get all benchmarks for BenchmarkFamilies

③ Match a benchmark name against regex, if any

④ Create BenchmarkInstances & collect them.

vector<unique_ptr<Benchmark>>

```
for (auto const& args :vector<int64_t> const& : family->args_) {
  for (int num_threads : *thread_counts) {
    BenchmarkInstance instance(family.get(), family_idx: family_index,
                               per_family_instance_index, args,
                               thread_count: num_threads);

    const auto full_name :string = instance.name().str();
    if (full_name.rfind( s: kDisabledPrefix, pos: 0) != 0 &&
        ((re.Match(full_name) && !is_negative_filter) ||
         (!re.Match(full_name) && is_negative_filter))) {
      benchmarks->push_back(std::move(instance));
```

→ argument

↓

defaults to 1

An important point to note in the loop is that a new instance is created for each argument of Benchmark.

(At this stage, we ignore the loop around thread count)

Run benchmarks

```cpp
// Loop through all benchmarks
for (const BenchmarkInstance& benchmark : benchmarks) {
  BenchmarkReporter::PerFamilyRunReports* reports_for_family = nullptr;
  if (benchmark.complexity() != oNone) {
    reports_for_family = &per_family_reports[benchmark.family_index()];
  }
  benchmarks_with_threads += static_cast<int>(benchmark.threads() > 1);
  runners.emplace_back(benchmark, &perfcounters, reports_for_family);
  int num_repeats_of_this_instance = runners.back().GetNumRepeats();
  num_repetitions_total +=
      static_cast<size_t>(num_repeats_of_this_instance);
  if (reports_for_family != nullptr) {
    reports_for_family->num_runs_total += num_repeats_of_this_instance;
  }
}
assert(runners.size() == benchmarks.size() && "Unexpected runner count.");
```

Vector of Benchmark Instance

Create a Runner for each BenchmarkInstance & add to Runners.

" Runners.size() Must equal Benchmarks.size() "

```cpp
std::vector<size_t> repetition_indices;
repetition_indices.reserve(num_repetitions_total);
for (size_t runner_index = 0, num_runners = runners.size();
     runner_index != num_runners; ++runner_index) {
  const internal::BenchmarkRunner& runner = runners[runner_index];
  std::fill_n(first: std::back_inserter( [&] repetition_indices),
              n: runner.GetNumRepeats(), runner_index);
}
assert(repetition_indices.size() == num_repetitions_total &&
       "Unexpected number of repetition indexes.");
```

A way to repeat a single Runner N times.

Assume two runners r_1 & r_2. r_1 is to be repeated 3 times & r_2 2 times. repetition_indices would look like:

→ len = total_repetitions

r_1	r_1	r_1	r_2	r_2

repetition_indices

```
ſ

for (size_t repetition_index : repetition_indices) {
  internal::BenchmarkRunner& runner = runners[repetition_index];
  runner.DoOneRepetition();
  if (runner.HasRepeatsRemaining()) {
    continue;
  }
```

A brief on Running a Runner.

Each Runner is identified & asked to do one repetition.

Each repetition will have I iterations.

A brief on DoOneRepetition

```
if (!warmup_done) {
  RunWarmUp();
}

IterationResults i;
// We *may* be gradually increasing the length (iteration count)
// of the benchmark until we decide the results are significant.
// And once we do, we report those last results and exit.
// Please do note that the if there are repetitions, the iteration count
// is *only* calculated for the *first* repetition, and other repetitions
// simply use that precomputed iteration count.
for (;;) {
  b.Setup();
  i = DoNIterations();
  b.Teardown();
```

More on this later

(i) Warmup the runner

↑ ONLY ONCE (PER RUNNER)

(ii) Perform Setup → N iterations & teardown.

It looks like Setup may be done more than once in a repetition