Setup of benchmark

BENCHMARK (BM_empty)

↑ ↑

Macro benchmark

f^ name.

BENCHMARK Macro

① Generate a unique variable name

② Create a new instance of

FunctionBenchmark → extends Benchmark

↑ takes a name &

a function pointer.

③ Register benchmark (or function benchmark) to BenchmarkFamilies

④ Return the pointer to Benchmark.

```
// Helpers for generating unique variable names
#define BENCHMARK_PRIVATE_NAME(...)                                  \
  BENCHMARK_PRIVATE_CONCAT(benchmark_uniq_, BENCHMARK_PRIVATE_UNIQUE_ID, \
                           __VA_ARGS__)

#define BENCHMARK_PRIVATE_CONCAT(a, b, c) BENCHMARK_PRIVATE_CONCAT2(a, b, c)
#define BENCHMARK_PRIVATE_CONCAT2(a, b, c) a##b##c
// Helper for concatenation with macro name expansion
#define BENCHMARK_PRIVATE_CONCAT_NAME(BaseClass, Method) \
  BaseClass##_##Method##_Benchmark

#define BENCHMARK_PRIVATE_DECLARE(n)                                 \
  /* NOLINTNEXTLINE(misc-use-anonymous-namespace) */                 \
  static ::benchmark::internal::Benchmark const* const BENCHMARK_PRIVATE_NAME( \
      n) BENCHMARK_UNUSED

#define BENCHMARK(...)                                               \
  BENCHMARK_PRIVATE_DECLARE(_benchmark_) =                           \
      (::benchmark::internal::RegisterBenchmarkInternal(             \
          ::benchmark::internal::make_unique<                        \
              ::benchmark::internal::FunctionBenchmark>(#__VA_ARGS__, \
                                                        __VA_ARGS__)))
```

We can use __LINE__

Unique name

new instance of FunctionBenchmark.

FunctionBenchmark(

#__VA_ARGS__, __VA_ARGS__)

assume BENCHMARK(BM_Empty)

→ this will become:

FunctionBenchmark("BM_Empty",

BM_Empty);

↑ function pointer

→ name

```
class BENCHMARK_EXPORT FunctionBenchmark : public Benchmark {
 public:
  FunctionBenchmark(const std::string& name, Function* func)
      : Benchmark(name), func_(func) {}

  void Run(State& st) override;

 private:
  Function* func_;
};
```

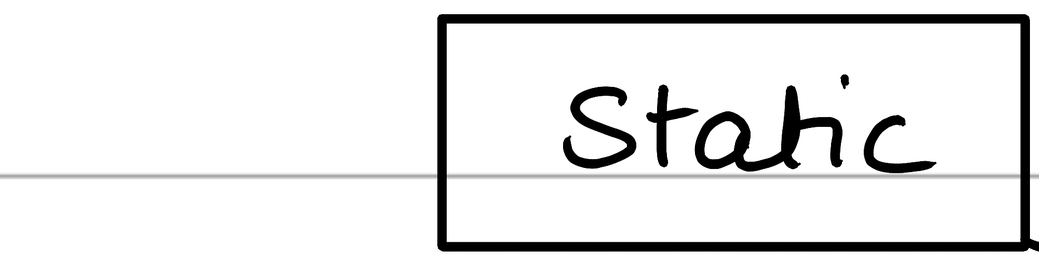
All variables are pointers to

Benchmark & are static.

↓

BENCHMARK_PRIVATE

- DECLARE



local to

the translation

unit

(c file)

life

till the

end of

the program

↳ avoids conflict

across files.

BENCHMARK_PRIVATE_DECLARE also

declares variables with another

macro BENCHMARK_UNUSED

lō avoid compiler warnings on unused variables.

```
BENCHMARK(BM_EMPTY);

BENCHMARK(BM_EMPTY) -> Arg(10);
```

Each of the above 2 will create a new instance of FunctionBenchmark. & a new Unique_ptr<Benchmark>

Open:

1. Benchmark families
2. Benchmark.

Benchmark client

```
void BM_empty(benchmark::State& state) {
  for (auto _ : state) {
    auto iterations = static_cast<double>(state.iterations()) *
                      static_cast<double>(state.iterations());
    benchmark::DoNotOptimize(iterations);
  }
}
BENCHMARK(BM_empty);
BENCHMARK(BM_empty)->ThreadPerCpu();
```