New Pass Manager in LLVM

- Scheduling can also be more involved, such as making sure we visit SCCs in the call graph in the correct order.

- The ability to *use function analysis results for arbitrary functions* from the inliner.

- The legacy PM did not support retrieval of analyses for arbitrary functions in a CGSCC pass. A CGSCC pass runs on a specific strongly connected component (SCC) of the call graph.

- The pass manager makes sure we visit SCCs bottom-up so that callees are as optimized as possible when we get to their callers and callers have as precise information as possible.

- LLVM's inliner is a CGSCC pass due to being a bottom-up inliner.

- Most codegen passes don't work on LLVM IR, but rather machine IR (MIR). Migrating to the new PM for the codegen pipeline likely won't unlock performance gains since there are almost no *interprocedural* codegen passes.

- With the legacy PM, each pass declares which analyses it requires and preserves. The new PM takes a different approach of completely separating analyses and normal passes. Rather than having the pass manager take care of analyses, a separate analysis manager is in charge of computing, caching, and invalidating analyses. In order for a pass to communicate that analyses have been invalidated, it returns which analyses it has preserved.

- The IR nesting in the new PM is module (-› CGSCC) -› function -› loop, where the CGSCC nesting is optional. Requiring the CGSCC nesting was considered to simplify things, but the extra runtime overhead of building the call graph and the extra code for proper nesting to run function passes was enough to make the CGSCC nesting optional.

- The improvements brought on by the new PM tend to be more relevant to large codebases.