`@_transparent`

Semantically, `@_transparent` means something like "treat this operation as if it were a primitive operation". The name is meant to imply that both the compiler and the compiled program will "see through" the operation to its implementation.

This has several consequences:

- Any calls to a function marked `@_transparent` MUST be inlined prior to doing dataflow-related diagnostics, even under `-Onone`. This may be necessary to *catch* dataflow errors.
- Because of this, a `@_transparent` function is implicitly inlinable, in that changing its implementation most likely will not affect callers in existing compiled binaries.
- Because of this, a public or `@usableFromInline` `@_transparent` function MUST only reference public symbols, and MUST not be optimized based on knowledge of the module it's in. [The former is caught by checks in Sema.]
- Debug info SHOULD skip over the inlined operations when single-stepping through the calling function.

This is all that `@_transparent` means.

When should you use `@_transparent`?

- Does the implementation of this function ever have to change? Then you can't allow it to be inlined.
- Does the implementation need to call private things---either true-`private` functions, or `internal` functions that might go away in the next release? Then you can't allow it to be inlined.
- Is it okay if the function is *not* inlined? You'd just prefer that it were? Then you should use `@inlinable`, rather than `@_transparent`. (If you really need this, you can add `@inline(__always)` as well.)
- Is it a problem if the function is inlined even under `-Onone`? Then you're really in the previous case. Trust the compiler.
- Is it a problem if you can't step through the function that's been inlined? Then you don't want `@_transparent`; you just want `@inline(__always)` (and probably `@inlinable` as well, for cross-module inlining).
- Is it okay if the inlining happens after all the dataflow diagnostics? Then you don't want `@_transparent`; you just want `@inline(__always)`.

If you made it this far, it sounds like `@_transparent` is the right choice.

Interaction with other annotations

- As mentioned above, putting `@_transparent` on a function that is `public` or `@usableFromInline` exposes its body to other modules. It is not necessary to additionally include `@inlinable`.
- Unlike `@inlinable`, however, `@_transparent` does not imply `@usableFromInline`. It is possible to have functions marked `@_transparent` that are only meant for use within the current module or even

the current file.

Current implementation limitations

- When compiling in non-single-frontend mode, no SIL is generated for any functions but those in the primary file (for each frontend invocation), including `@inline(__always)` and `@_transparent` functions, which means they will not be inlined. This is semantically a bug. rdar://problem/15366167