Threads?

3.3. Profile: Concurrency

- ***Definition***: no data races. No deadlocks. No races for external resources (e.g., for opening a file).
- ***Question***: should we also deal with priority inversion, delays caused by excess contention on a lock? Suggested initial answer: no.
- ***Observation***: The concurrency profile is currently the least mature of the suggested profiles. It has received essentially no work specifically related to profiles, but concurrency problems have received intensive scrutiny in other contexts (including the Core Guidelines and MISRA++) so I can offer a few suggestions for initial work:
  - ***Threads***: prefer **jthread** to **thread** to get fewer scope-related problems.
  - ***Dangling pointers***: consider a **jthread** a container and apply the usual rules for resource lifetime (RAII) and invalidation (§3.9).
  - ***Aliasing***: statically detect if a pointer is passed to another thread. For an initial version, that will require restrictions on pointer manipulation in non-trivial control flows. In general, not all aliasing can be detected statically, and we need to reject too complex code. Defining "too complex" is essential, or we will suffer portability problems because of compiler incompatibilities. See "Flow analysis" (§4).
  - ***Invalidation***: use **unique_ptr** and containers without invalidation (e.g., **gsl::dyn_array**) to pass information between threads.
  - ***Mutability***: Prefer to pass (and keep) pointers to **const**.
  - ***Synchronization***: use **scoped_lock** to lessen the chance of deadlock. Look into the possibility of statically detecting aliases in more than one thread to mutable data and enforce the use of synchronization on access through them. Use **unique_ptr** combined with protecting against aliasing across treads.

We need to look at lock-free programming.