1. What are the target applications and what is the expected sophistication of the target users (naive, knowledgeable, expert)?

- Currently targeting Sandia ATDM applications (EM plasma/PIC kernels, MiniMultiScale, MiniUQ).
- Our end goal is to provide abstractions that naïve users can use easily.
- However, in order to get to there, we are working with users who are on the high end of knowledgeable to refine the abstractions.

2. List the set of features/concepts of your programming model and divide them into two sets: What is the minimum set of things a new user needs to learn to become productive? What are the more advanced features a user can potentially take advantage of?

- In general, minimal features are expressed with positional arguments and advanced features are expressed with keyword arguments.
- AccessHandle, AccessHandleCollection, create_work, and create_concurrent_work are our basic abstractions
- Some advanced features include, for instance, custom mappings between AccessHandleCollections and create_concurrent_work regions, custom index ranges, and performance hints/directives.

3. How do you decide whether a new application is a good fit? What metrics do you use to evaluate whether an application is implemented well in your model?

- Given DARMA's overarching goals, application developers that are willing to co-design abstractions, provide feedback/requirements, and are interested in exploring potential algorithmic changes (e.g., modifying algorithms to become more latency-tolerant) are a "good fit."
- Regarding the second question, the community as a whole lacks a good way to do this. In our studies, we do our best to collect both qualitative (subjective) data as well as quantitative results (e.g., weak and strong scaling studies).
- We recognize that presenting subjective results is difficult and it is hard to get large sample sizes for feedback. However, we feel it is very important to capture these metrics, especially from a user-adoption perspective.
- Although quantitative results are easier to gather and present, there are challenges here as well. In particular, these types of results run the risk of painting an incomplete picture due to lack of context captured with the resulting plots.
- We as a community would benefit from developing a consistent set of community-agreed-upon performance tests and meta-data that should be captured (e.g., average user coder vs ninja coder, are you exploring a straight forward MPI port, or are you making algorithmic changes to the application to expose the potential for asynchrony).
- Following on Ron B.'s comments, not only providing performance results, but explaining the reasons behind those results is critical (but often difficult to do).

4. What is the plan for interoperability with MPI, OpenMP, Kokkos, etc.? If you could add requirements to MPI, what would those be?

- Interoperability is key to application adoption.
- This year we are working on abstractions with the Kokkos team for Kokkos interoperability.
- For MPI, we want to develop abstractions that support epoch-based MPI interoperability (this year) and then add overlap. This abstraction will look something like a special create_concurrent_work.

5. What is the plan for performance portability?

- DARMA is portability layer across AMT runtimes, so from a system-level perspective, performance portability is dependent on the performance portability of the backend that the application developer chooses.
- Application developers could also choose to work with different backends to provide portability (i.e., if a backend is known to be more performant for a class of algorithms and/or platforms).
- Intra-node performance portability we are delegating to on-node runtimes (esp. for data-parallel structures). Our ATDM application partners are using Kokkos.

6. What is the plan for fault tolerance?

- We are coordinating with teams at Sandia focused on this.
- Also, we can leverage whatever fault-tolerance provided by the backend runtime the user chooses.

7. What static analyses and transformations could you do? What do you do today?

We're currently collaborating with a formal methods team to formalize the type system inherent to our AccessHandle abstraction. This would allow us to provide certain safety guarantees at compile time.

8. Questions about task graphs:

a) When is the task graph generated (compile-time, load-time, run-time)?
b) How do you manage task graph generation vs. task graph execution?
c) What is the value of non-ready tasks in the DAG?
d) Do you exploit the repetitiveness of iterative applications that repeatedly execute the same task graph?

a) Task-data dependency capture happens at compile time, task graph generation happens at run-time (by the backend)
b) We have an API that separates concerns into a set of non-racing and order-independent operations; the task graph execution is managed by the backend under these constraints
c) Lookahead allows the backend runtime to prefetch data, predict loads, hide latency, hide overhead, etc. It's up to them to do it.
d) This is dependent on the backend runtime. (The use of task collections generated by create_concurrent_work make this much more doable/scalable. We believe that user-application provided hints can also help with this)

9. Questions about tasks:

a) How is task granularity managed?

b) What is the life-cycle of a task?

a) By the user (future experimental plans to do some automatic task fusion/fission)

b) Up to the backend. (Ownership of the task object is transferred to the backend at registration time)

10. What is the relationship between task and data parallelism --- can one be invoked from the other arbitrarily or are there restrictions?

No restrictions. (Currently there are some practical restrictions on create_concurrent_work, but we plan to lift those at some point)

11. Where exactly is concurrency (meaning the ability to have races and deadlocks) exposed to the programmer, if at all?

Nowhere. (We have an experimental plan for relaxed coherency at some point, but it's not high priority right now).