Position: characteristics of programming languages can facilitate support for resilience

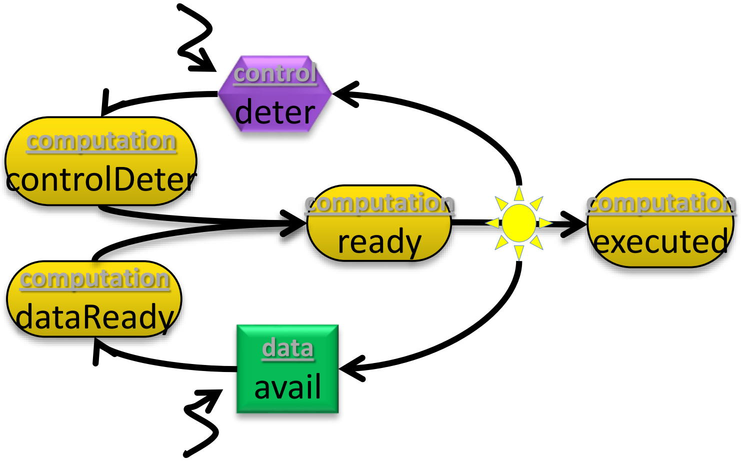
Tripped over resilience while working on CnC
Lessons learned / experience

Chunks of:

- Data
  - available
- Control
  - determined
- Computations
  - data-available
  - control-determined
  - ready

Implementations:

- program execution
  = manage the state



State of app:

- Attributes of chunks
- Value of available data

Just next phase in a sequence of implementations

1. Manage state of dynamic execution
2. Ensure the valid state for a static execution
3. State on disk. Execute a step on any available resource
4. Checkpoint: (combo of 1 and 3) Manage
   - -State of dynamic execution as above
   - -Also asynchronously continuously save state changes
5. Application checkpoint/continue
   - Provided by hierarchical checkpoint/stop/restart

4) What is the impact on resilience of the wide range of expected operating scenarios with respect to dynamically changing resources, application characteristics, and the wide range of possible error and failure rates?

Characteristics

- No synchronization required
  - May not be a state that ever occurred
  - May not even be a valid state
  - Whatever is saved (in what ever order it was saved) is a legal restarting state

- You can restart
  - Different time, configuration, architecture, CnC runtime

- Key traits that make this work:
  - simple application state / continuously updated / conceptually monotonic / order independent
    1) What features of other levels of the stack (algorithm, programming model, compiler, runtime, and hardware) should resilience depend on?

Summary: 2) How can resilience schemes best exploit application, runtime, or programming model semantics?